18020 EWN

Easy Wireless Networking Using the Arduino™ Compatible chipKIT™ Platform

Class Objectives

When you walk out of this class you will know….

- Fundamentals of Network Topology
- Fundamentals of the DEIPcK Network Stack
- Fundamentals of HTTP and HTML
- How to build the HTTP Example Server
- How to work with Static HTML pages
- How to Create Dynamic HTML pages

Who am I?

Keith Vogel

Senior Software Engineer

KeithV@Digilentinc.com

Please feel free to ask questions at any time.

MICROCHIP
MASTERs 2014

Class Agenda

- **Network Fundamentals**
  - ARP – Address Resolution Protocol
  - IP Routing
  - DHCP – Dynamic Host Configuration Protocol
  - DNS – Domain Name System
- **Digilent Embedded IP Stack for chipKIT™ (deIP™ / DEIPck)**
- **HTTP Example Server**
- **LAB 1: Build and running the deIP™ HTTP Example Server**

MICROCHIP
MASTERs 2014

Class Agenda Continued

- **HTTP Protocol Fundamentals**
- **HTML Syntax Fundamentals**
- **HTTP Server Architecture**
- **LAB 2: Working with Static HTML Pages**
- **HTTP Server and Dynamic HTML Pages**
- **LAB 3: Working with Dynamic HTML Pages**
- **Additional: Debugging the HTTP Server**
  - At the end of the slide deck for your review

Network Fundamentals

Note: This section is somewhat technical with a lot of good information; the network stack implements this and only a high level understanding is needed to understand the network stack

Typical Home Network

ISP

WAN

69.76.19.248

Modem

Ethernet
802.3

Router

Wi-Fi ®
802.11

LAN

192.168.1.xxx

Router?

NAT
192.168.1.50:49001 <-> 69.76.19.248:49152
192.168.1.51:49001 <-> 69.76.19.248:49153

WAN
69.76.19.248

DNS Forwarding
192.168.1.1 -> 8.8.8.8

DHCP
00-15-C5-53-FF-74 -> 192.168.1.50
00-15-C5-53-FF-88 -> 192.168.1.51

Wi-Fi®
802.11
Access Point

192.168.1.50

Switch

Ethernet
802.3

192.168.1.51

192.168.1.52

LAN
192.168.1.xxx

Network Protocol Layers

1. **Physical Layer**
   - 802.11, 10BASE-T/100BASE
2. **Data Link Layer**
   - ARP, 802.3, Ethernet II
3. **Network Layer**
   - IPv4, IPv6, ICMP
4. **Transport Layer**
   - TCP, UDP, NAT
5. **Application Layer**
   - DNS, HTTP, NTP, FTP, DHCP

Question?

What is Network Address Translation (NAT)?

Network Address Translation (NAT)

- **Masqueraded Networks**
  - Hide an entire IP space under one IP
  - Enables Private IP Spaces
    - **(A)10.0.0.0/8, (B)172.16.0.0/12, (C)192.168.0.0/16**
  - Forces communication to be initiated from within the masqueraded network
  - Implemented by mapping an internal IP:Port to the fixed External IP:and Mapped Port
- **Implemented by a NAT Gateway**
- **Many Gateways allow for port forwarding**
  - Allows for communication to start outside of the masqueraded network to specific ports

Demo

The HTTP Server

Question?

What is the difference between a Hub and Switch?

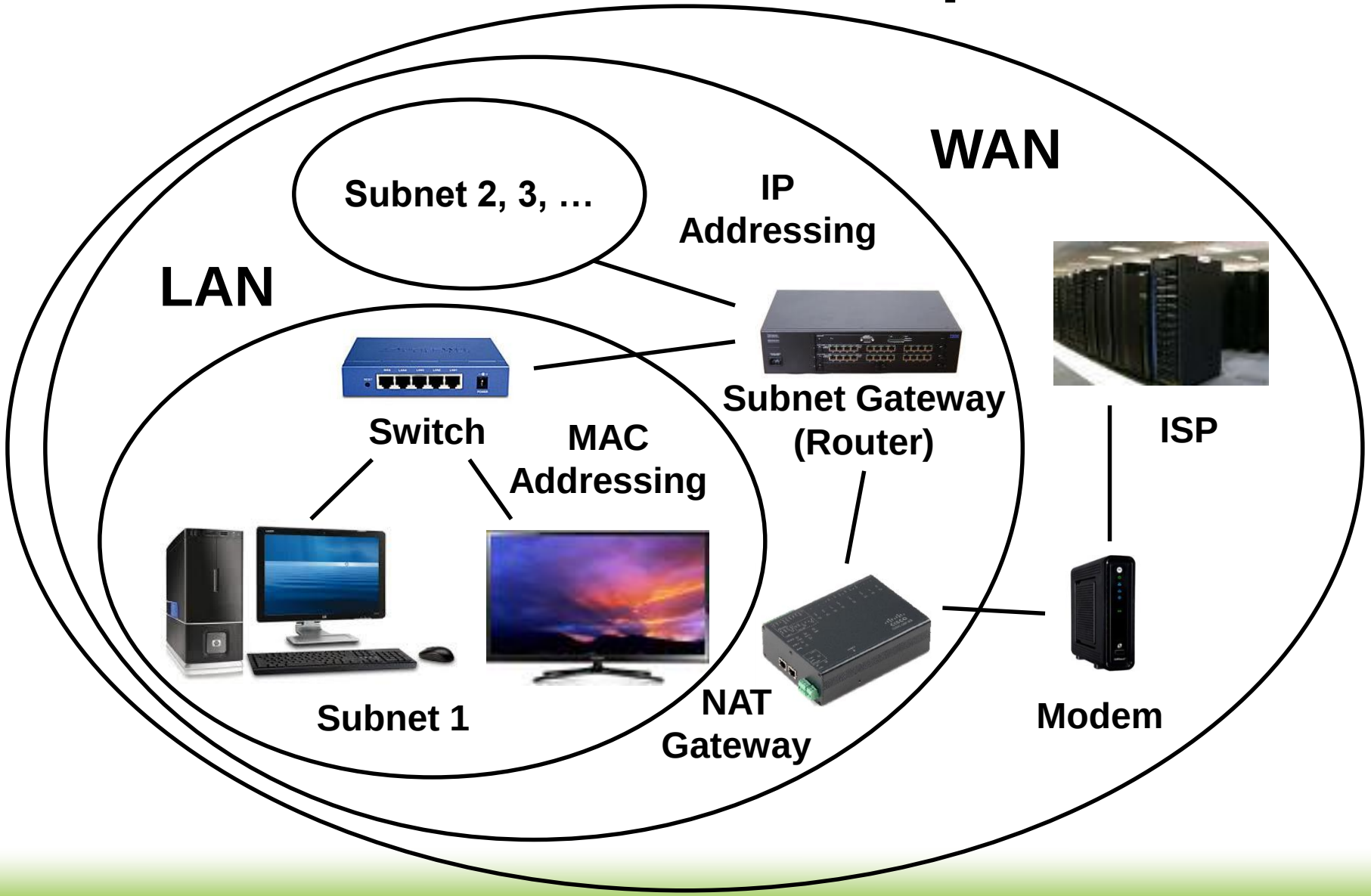
Network Hardware

- **Modem (Physical Layer)**
  - Physical signal bridge; i.e. CAT6 to Cable
- **Hub (Physical Layer)**
  - Packet replication to all ports
- **Access Point (AP, Physical Layer)**
  - Wireless Access to the LAN
- **Switch (Link Layer)**
  - Packet routing by MAC, usually automatic
- **Router (Network Layer) / Subnet Gateway**
  - IP routing; Manual and/or automatic IP routing
- **NAT Gateway (Transport Layer)**
  - NAT translation; Port to IP mapping; WAN to LAN

The name Router is Confusing

- **NAT (Network Address Translation) Gateway to the WAN (Transport Layer)**
- **Router if there are multiple subnets (Network Layer)**
  - Not typical in home environments
- **Switch for the LAN (Link Layer)**
- **AP for Wi-Fi® (Physical Layer)**
- **DHCP for the LAN (App Layer)**
- **DNS forwarder to our ISP (Internet Service Provider) (App Layer)**
- **ARP on the LAN (Link Layer)**
  - Returns Router's MAC for IPs not on the LAN

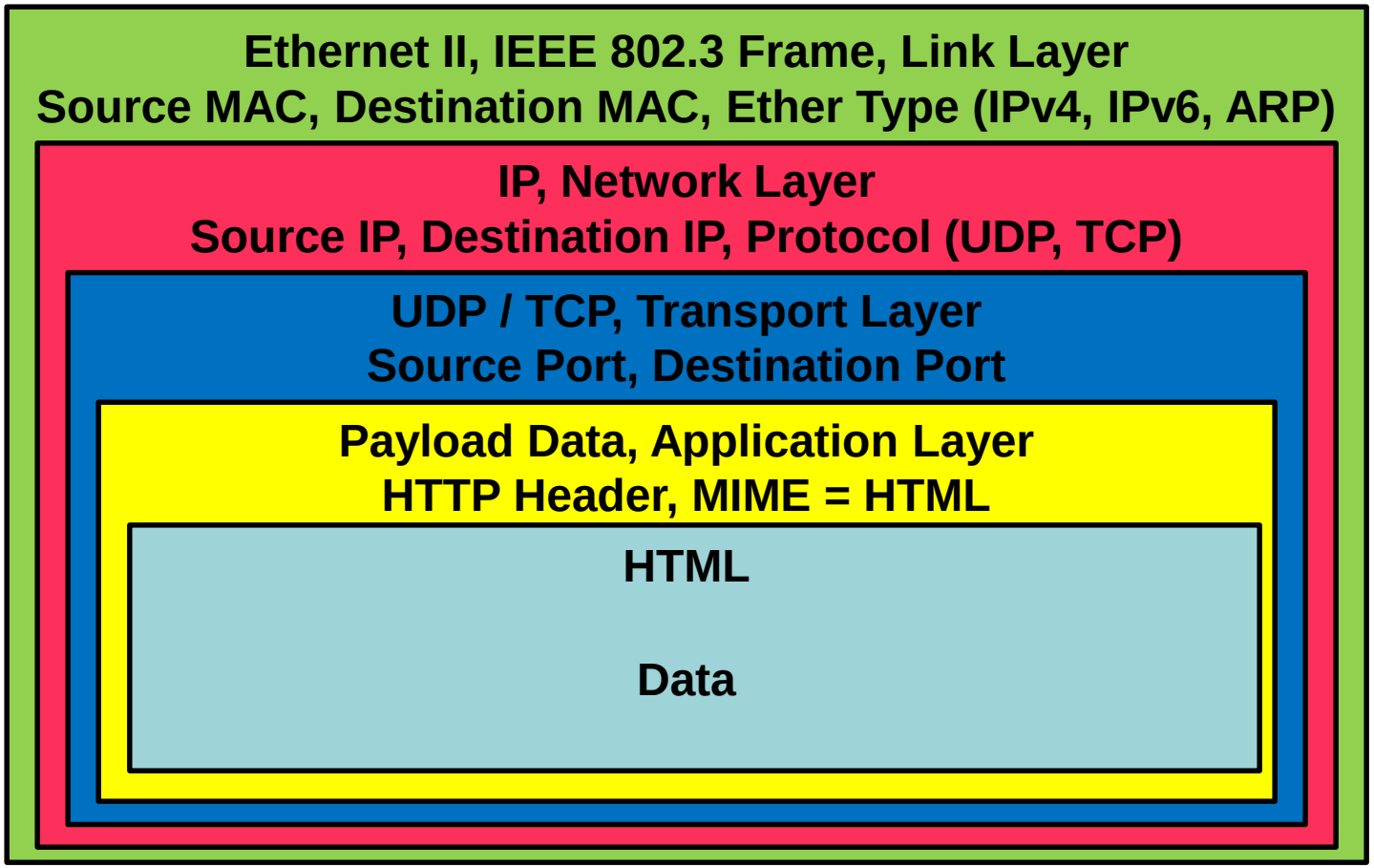
The Internet or "*Router*" Exploded



Network Addressing

- **MAC: Media Access Control (Link Layer)**
  - Assigned by manufacture, unique to the hardware and used in Ethernet addressing
- **IP: Internet Protocol (Network Layer)**
  - IPv4: 32 bit value unique network IP
  - IPv6: 128 bit value unique network IP
- **Domain Name (App Layer)**
  - Hierarchical name that will resolve to a unique IP within the network

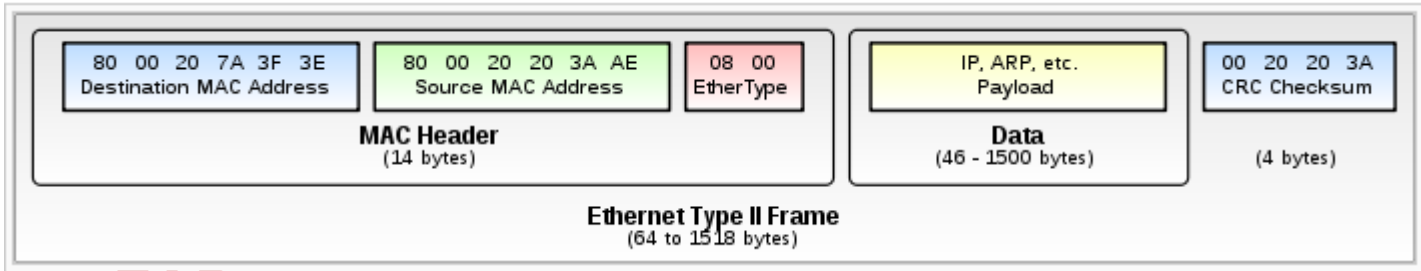
Example Packet Structure



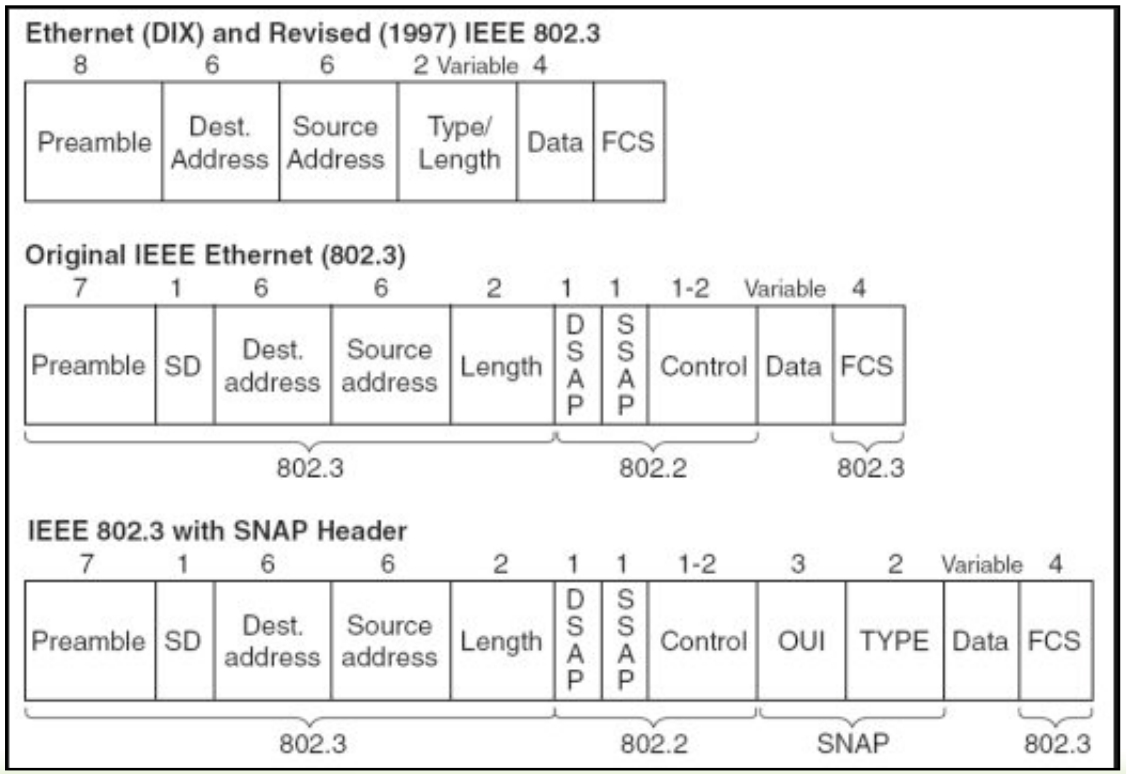
Question?

What is the difference between an Ethernet II Frame and an IEEE 802.3 Frame?

Ethernet Frames



By FAR, the Ethernet Type II Frame is the most common!



Subnet at Link Layer

IP: 192.168.1.1
MAC: 00-B7-C9-44-F6-03

AP

Wi-Fi® 802.11
Physical Layer

IP: 192.168.1.52
MAC: 00-A4-67-29-E8-17

LAN: 192.168.1.0
Subnet Mask: 255.255.255.0
Gateway: 192.168.1.1
DNS: 8.8.8.8, 8.8.4.4

Switch

Ethernet II Framing
OR
802.3 Frame

Addressing by MAC
Link Layer

IP: 192.168.1.50
MAC: 00-15-C5-53-FF-74

IP: 192.168.1.51
MAC: 00-34-AA-53-FF-82

Subnet

- **Addressing by MAC address**
- **IP Addresses are resolved to a MAC by Broadcasted ARP (Address Resolution Protocol)**
- **IP addresses in a subnet identified by AND'ing the IP with a subnet mask**
- **IPs not on the subnet passed to the router to be forwarded to another subnet**
  - This router is often referred to as a gateway that is, a gateway to another subnet

Subnet Addressing

IP addr:	Network Prefix	Subnet Number	Host Number
Network Addr:	Network Prefix	0's	0's
Network Mask:	1's	0's	0's
Subnet Addr:	Network Prefix	Subnet Number	0's
Subnet Mask:	1's	1's	0's
Broadcast Addr:	Network Prefix	Subnet Number	1's

If an IP is a member of the subnet then:

IP address AND Subnet Mask = Subnet Address

i.e. 192.168.1.50 AND 255.255.255.0 = 192.168.1.0

If a target IP is a member of the subnet then the Ethernet Frame is sent directly to the target machine by MAC address.

If a target IP is not a member of the subnet then the Ethernet Frame is sent to the gateway (using the gateway's MAC address)

Network Services

- **ARP: Address Resolution Protocol (Link Layer)**
  - Resolve an IP address to a MAC address
- **IP Routing**
  - Routing packets around the LAN to the final endpoint subnet
- **DHCP: Dynamic Host Configuration Protocol (App Layer)**
  - Dynamically acquiring network parameters
    - IP, Gateway, subnet mask, DNS servers
- **DNS: Domain Name System (App Layer)**
  - Resolving a domain name to an IP address

MICROCHIP
MASTERs 2014

Class Agenda

- Network Fundamentals
  - **ARP – Address Resolution Protocol**
  - IP Routing
  - DHCP – Dynamic Host Configuration Protocol
  - DNS – Domain Name System
- **Digilent Embedded IP Stack for chipKIT™ (deIP™ / DEIPck)**
- **HTTP Example Server**
- **LAB 1: Build and running the deIP™ HTTP Example Server**

ARP

Address Resolution Protocol

Address Resolution Protocol (ARP) within subnet

IP: 192.168.1.50
MAC: 00-15-C5-53-FF-74

Broadcast ARP Request

Src IP: 192.168.1.50
Dest IP: 192.168.1.51
Src MAC: 00-15-C5-53-FF-74
Dest MAC: FF-FF-FF-FF-FF-FF

IP: 192.168.1.51
MAC: 00-34-AA-53-FF-82

Broadcast ARP Request

Src IP: 192.168.1.50
Dest IP: 192.168.1.51
Src MAC: 00-15-C5-53-FF-74
Dest MAC: FF-FF-FF-FF-FF-FF

IP: 192.168.1.52
MAC: 00-A4-67-29-E8-17

Address Resolution Protocol (ARP) within subnet

IP: 192.168.1.50
MAC: 00-15-C5-53-FF-74

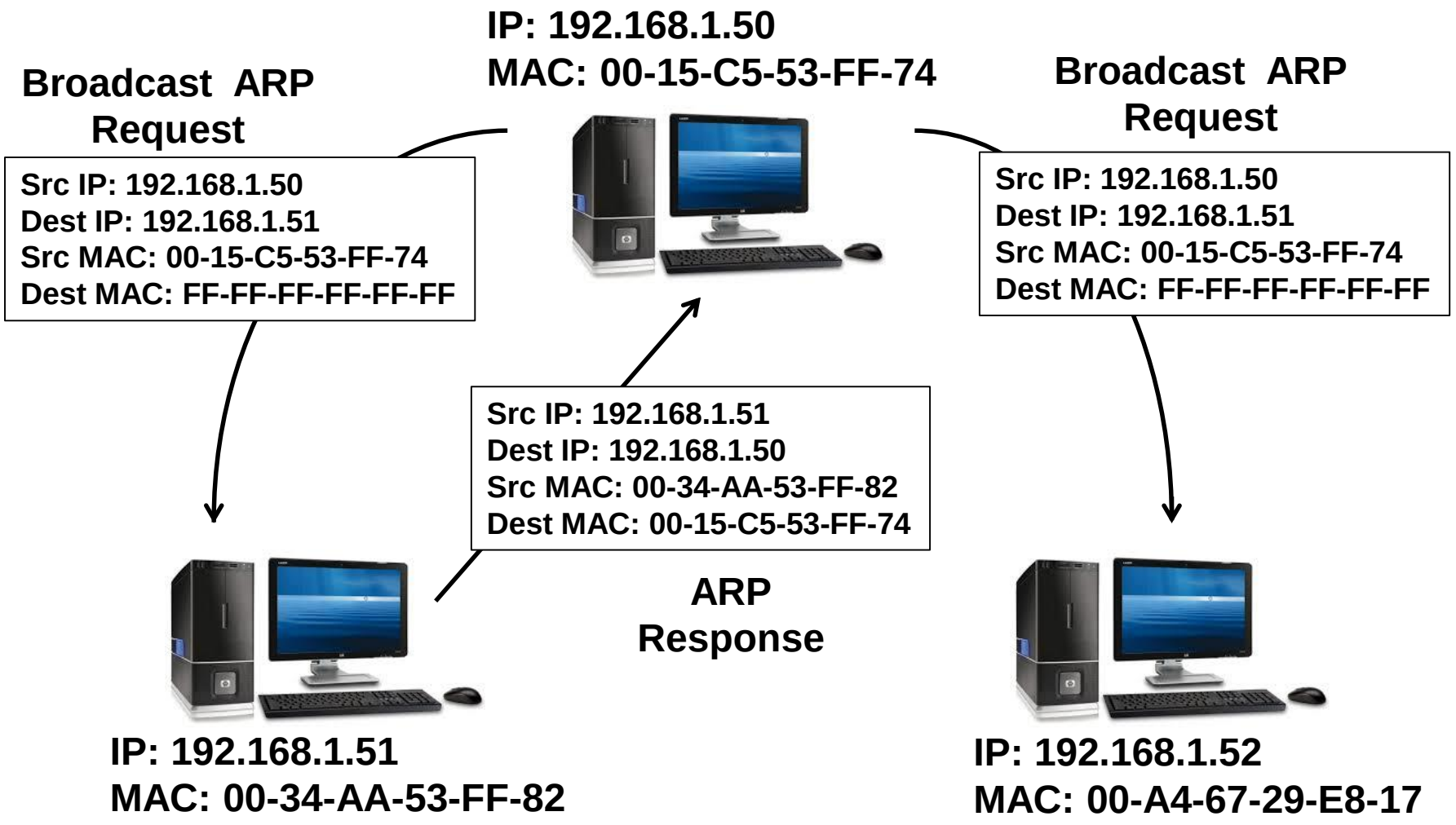
Src IP: 192.168.1.51
Dest IP: 192.168.1.50
Src MAC: 00-34-AA-53-FF-82
Dest MAC: 00-15-C5-53-FF-74

ARP Response

IP: 192.168.1.51
MAC: 00-34-AA-53-FF-82

IP: 192.168.1.52
MAC: 00-A4-67-29-E8-17

*Address Resolution Protocol (ARP) within subnet



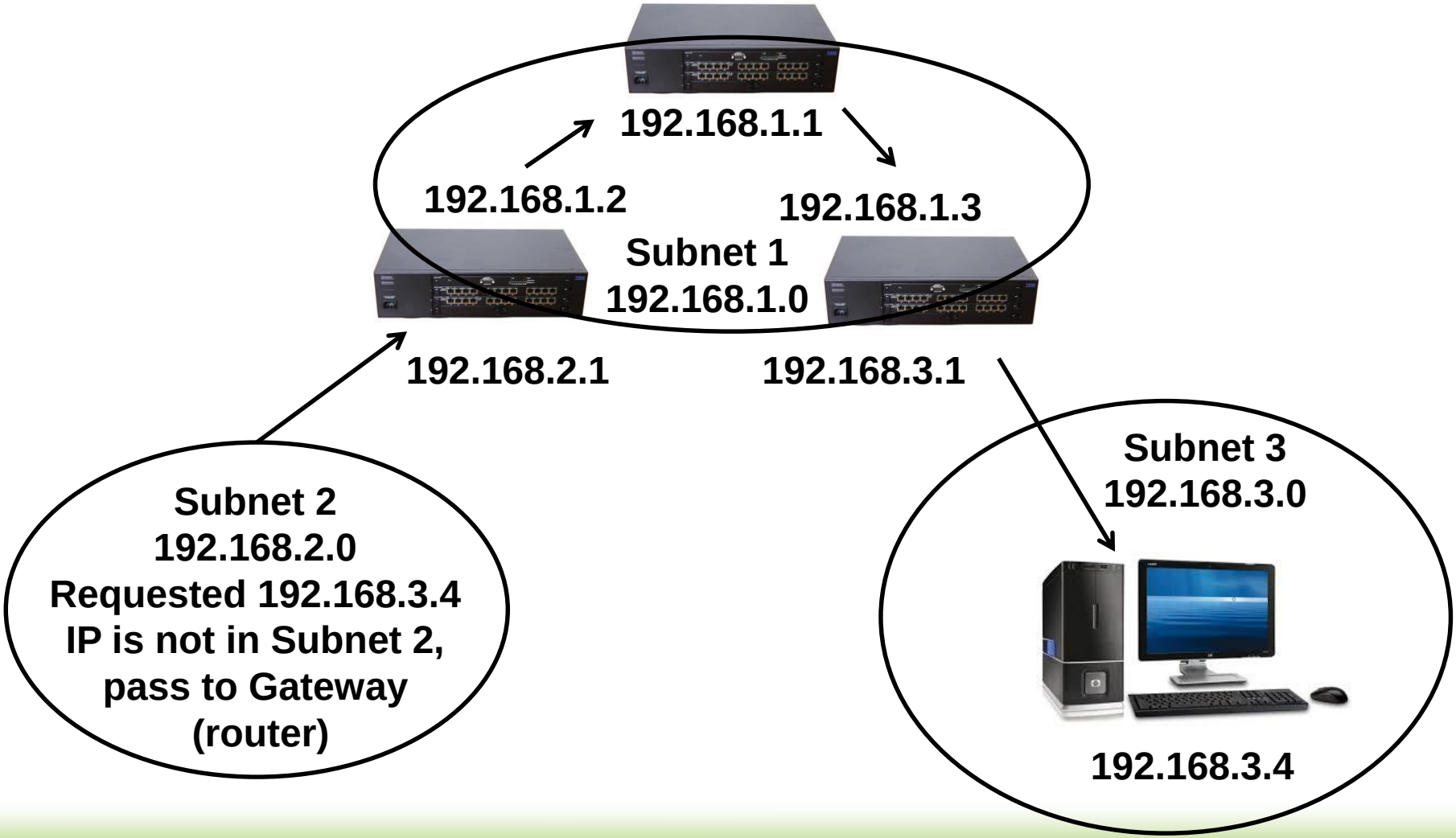
MICROCHIP
MASTERs 2014

Class Agenda

- **Network Fundamentals**
  - ARP – Address Resolution Protocol
  - **IP Routing**
  - DHCP – Dynamic Host Configuration Protocol
  - DNS – Domain Name System
- **Digilent Embedded IP Stack for chipKIT™ (deIP™ / DEIPck)**
- **HTTP Example Server**
- **LAB 1: Build and running the deIP™ HTTP Example Server**

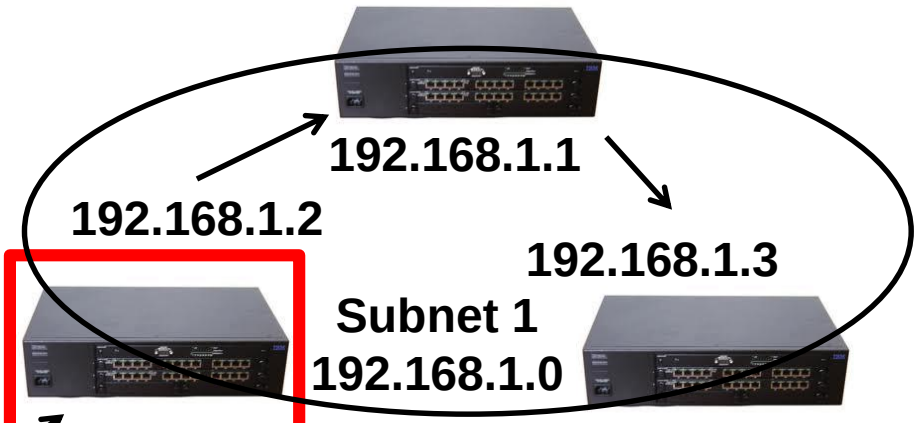
IP Routing

IP Routing



IP Routing

Router is Gateway for subnet 2 but does not have requested subnet 3 IP in tables, pass to his Gateway (router)



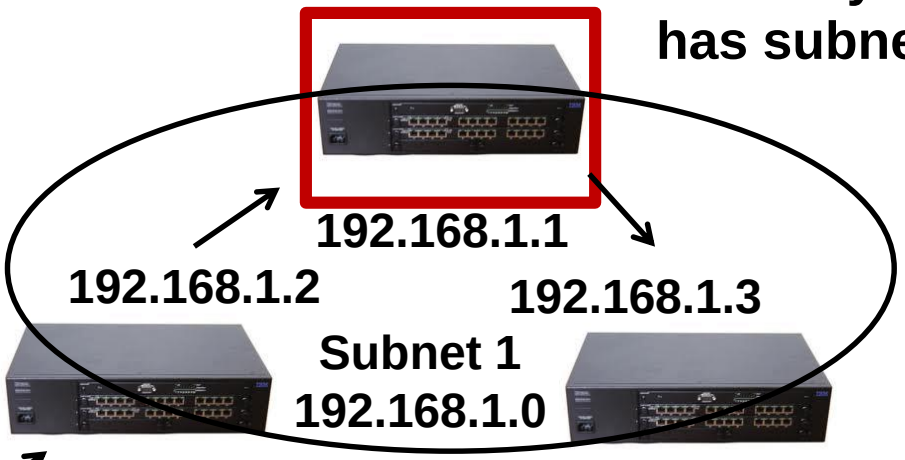
Subnet 2
192.168.2.0
Requested 192.168.3.4
IP is not in Subnet 2,
pass to Gateway
(router)

Subnet 3
192.168.3.0

192.168.3.4

IP Routing

Router is Gateway for subnet 1 and has subnet 3 in his tables and knows who to send it too



192.168.1.1

192.168.1.2

192.168.1.3

Subnet 1
192.168.1.0

192.168.2.1

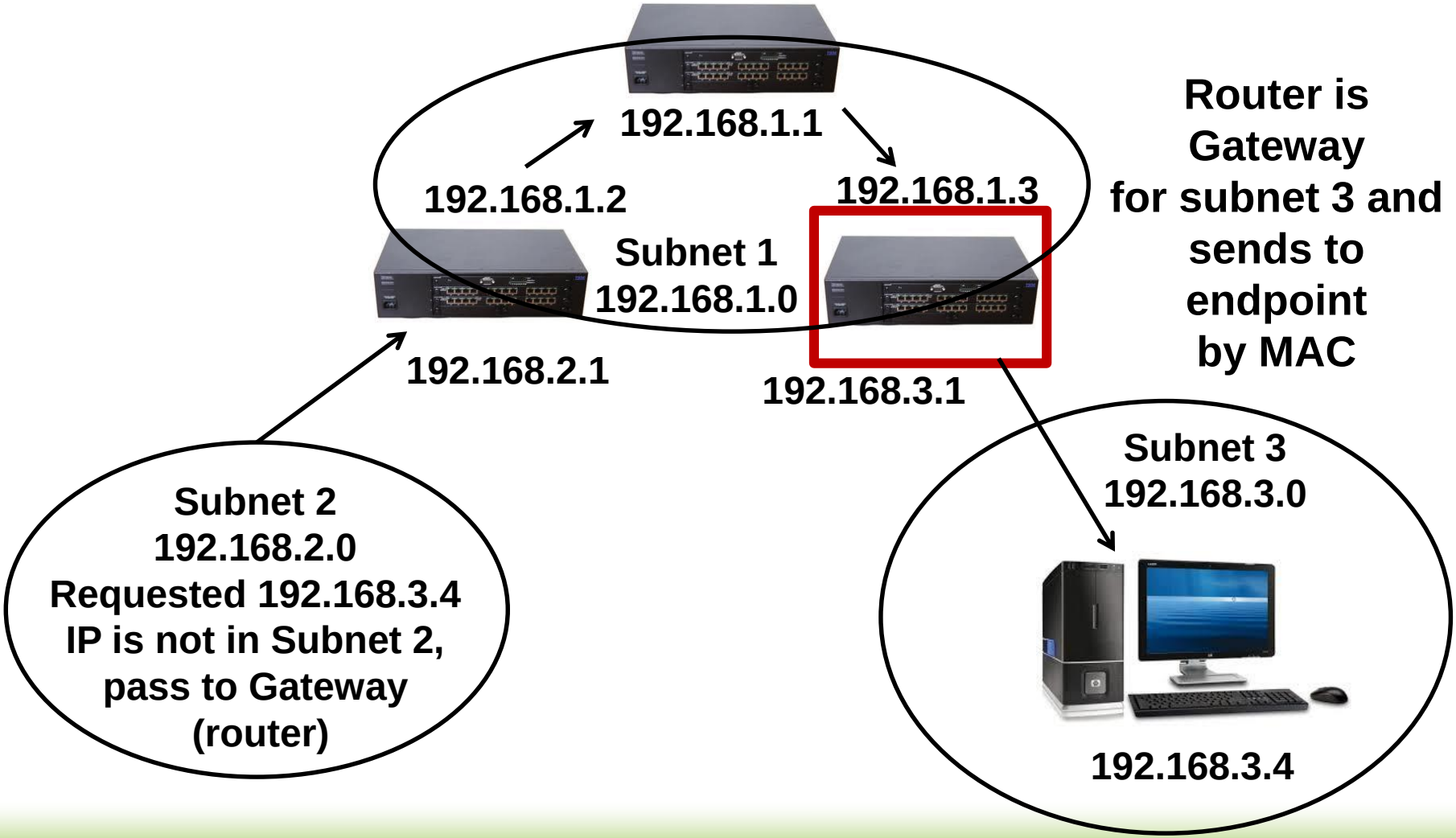
192.168.3.1

Subnet 2
192.168.2.0
Requested 192.168.3.4
IP is not in Subnet 2,
pass to Gateway
(router)

Subnet 3
192.168.3.0

192.168.3.4

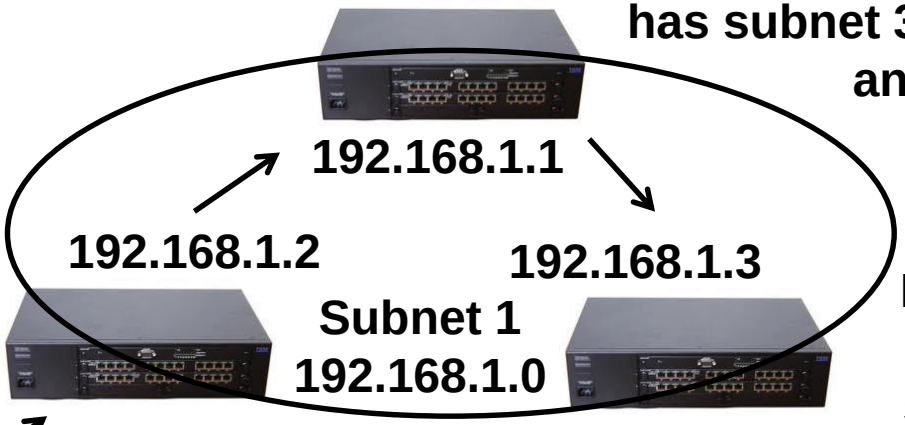
IP Routing



*IP Routing

Router is Gateway for subnet 1 and has subnet 3 in his tables and knows who to send it too

Router is Gateway for subnet 2 but does not have requested subnet 3 IP in tables, pass to his Gateway (router)



192.168.1.1

192.168.1.2

192.168.1.3

Subnet 1
192.168.1.0

192.168.2.1

192.168.3.1

Router is Gateway for subnet 3 and sends to endpoint by MAC

Subnet 2
192.168.2.0
Requested 192.168.3.4
IP is not in Subnet 2, pass to Gateway (router)

Subnet 3
192.168.3.0

192.168.3.4

Routing Table

Destination LAN IP	Subnet Mask	Gateway	Hop Count	Interface
192.168.3.0	255.255.255.0	192.168.1.3	1	LAN & Wireless
192.168.2.0	255.255.255.0	192.168.1.2	1	LAN & Wireless
192.168.1.0	255.255.255.0	0.0.0.0	1	LAN & Wireless
224.0.0.0	240.0.0.0	0.0.0.0	1	LAN & Wireless

224.0.0.0: is DNS Multicast address (mDNS)
Gateway 0.0.0.0 is short hand for this router: 192.168.1.1
Technically 0.0.0.0 is an invalid IP Address

Class Agenda

- **Network Fundamentals**
  - ARP – Address Resolution Protocol
  - IP Routing
  - **DHCP – Dynamic Host Configuration Protocol**
  - DNS – Domain Name System
- **Digilent Embedded IP Stack for chipKIT™ (deIP™ / DEIPck)**
- **HTTP Example Server**
- **LAB 1: Build and running the deIP™ HTTP Example Server**

DHCP

Dynamic Host Configuration Protocol

Question?

How many DHCP servers can be on the LAN?

Dynamic Host Configuration Protocol (DHCP)

00-78-22-C7-FE-A4
192.168.1.1

00-AA-E3-B4-12-7A
192.168.1.2

1

1

Discovery
Src IP: 0.0.0.0
Dst IP: 255.255.255.255
Src Port: 68
Dst Port: 67
Src MAC: 00-15-C5-53-FF-88
Dst MAC: FF-FF-FF-FF-FF-FF

1

00-15-C5-53-FF-88
192.168.1.???

Dynamic Host Configuration Protocol (DHCP)

00-78-22-C7-FE-A4
192.168.1.1

2

Offer
Src IP: 192.168.1.1
Dst IP: 255.255.255.255
YIADDR: 192.168.1.50
Src Port: 67
Dst Port: 68
Src MAC: 00-78-22-C7-FE-A4
Dst MAC: 00-15-C5-53-FF-88

2

00-AA-E3-B4-12-7A
192.168.1.2

Offer
Src IP: 192.168.1.2
Dst IP: 255.255.255.255
YIADDR: 192.168.1.100
Src Port: 67
Dst Port: 68
Src MAC: 00-AA-E3-B4-12-7A
Dst MAC: 00-15-C5-53-FF-88

2

2

00-15-C5-53-FF-88
192.168.1.???

Dynamic Host Configuration Protocol (DHCP)

00-78-22-C7-FE-A4
192.168.1.1

00-AA-E3-B4-12-7A
192.168.1.2

3

Request
Src IP: 0.0.0.0
Dst IP: 255.255.255.255
Request: 192.168.1.100
DHCP: 192.168.1.2
Src Port: 68
Dst Port: 67
Src MAC: 00-15-C5-53-FF-88
Dst MAC: FF-FF-FF-FF-FF-FF

3

00-15-C5-53-FF-88
192.168.1.???

Dynamic Host Configuration Protocol (DHCP)

00-78-22-C7-FE-A4
192.168.1.1

00-AA-E3-B4-12-7A
192.168.1.2

4

ACK
Src IP: 192.168.1.2
Dst IP: 255.255.255.255
YIADDR: 192.168.1.100
Src Port: 67
Dst Port: 68
Src MAC: 00-AA-E3-B4-12-7A
Dst MAC: 00-15-C5-53-FF-88
Gateway/SubnetMask/DNS

4

00-15-C5-53-FF-88
192.168.1.100

*Dynamic Host Configuration Protocol (DHCP)

00-78-22-C7-FE-A4
192.168.1.1

00-AA-E3-B4-12-7A
192.168.1.2

Offer
Src IP: 192.168.1.1
Dst IP: 255.255.255.255
YIADDR: 192.168.1.50
Src Port: 67
Dst Port: 68
Src MAC: 00-78-22-C7-FE-A4
Dst MAC: 00-15-C5-53-FF-88

Discovery
Src IP: 0.0.0.0
Dst IP: 255.255.255.255
Src Port: 68
Dst Port: 67
Src MAC: 00-15-C5-53-FF-88
Dst MAC: FF-FF-FF-FF-FF-FF

Offer
Src IP: 192.168.1.2
Dst IP: 255.255.255.255
YIADDR: 192.168.1.100
Src Port: 67
Dst Port: 68
Src MAC: 00-AA-E3-B4-12-7A
Dst MAC: 00-15-C5-53-FF-88

00-15-C5-53-FF-88
192.168.1.100

Request
Src IP: 0.0.0.0
Dst IP: 255.255.255.255
Request: 192.168.1.100
DHCP: 192.168.1.2
Src Port: 68
Dst Port: 67
Src MAC: 00-15-C5-53-FF-88
Dst MAC: FF-FF-FF-FF-FF-FF

ACK
Src IP: 192.168.1.2
Dst IP: 255.255.255.255
YIADDR: 192.168.1.100
Src Port: 67
Dst Port: 68
Src MAC: 00-AA-E3-B4-12-7A
Dst MAC: 00-15-C5-53-FF-88
Gateway/SubnetMask/DNS

Class Agenda

- **Network Fundamentals**
  - ARP – Address Resolution Protocol
  - IP Routing
  - DHCP – Dynamic Host Configuration Protocol
  - **DNS – Domain Name System**
- **Digilent Embedded IP Stack for chipKIT™ (deIP™ / DEIPck)**
- **HTTP Example Server**
- **LAB 1: Build and running the deIP™ HTTP Example Server**

DNS

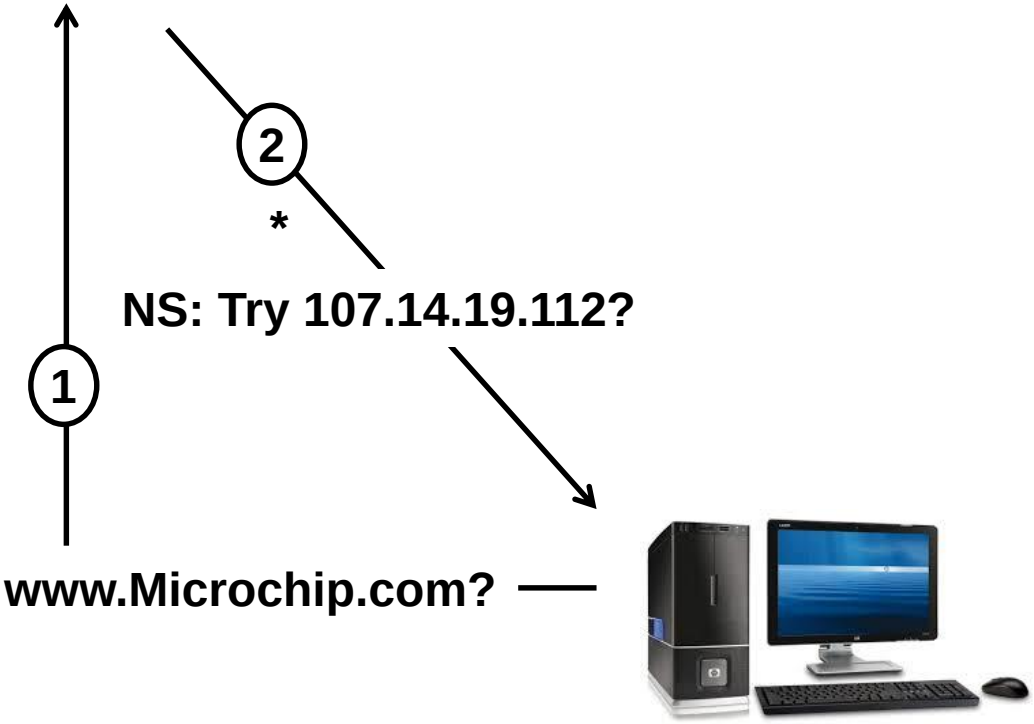
Domain Name System

Domain Name Resolution

8.8.8.8

107.14.19.112

107.14.16.206



***If a DNS server supports recursive resolution, the 1st query will likely return the address record (A) with the requested IP**

Domain Name Resolution

8.8.8.8

107.14.19.112

107.14.16.206

www.Microchip.com?

3

4

CName: e1721.a.akamaiedge.net
NS: Try 107.14.16.206

*If a DNS server supports recursive resolution, the 1st query will likely return the address record (A) with the requested IP

Domain Name Resolution

8.8.8.8

107.14.19.112

107.14.16.206

5

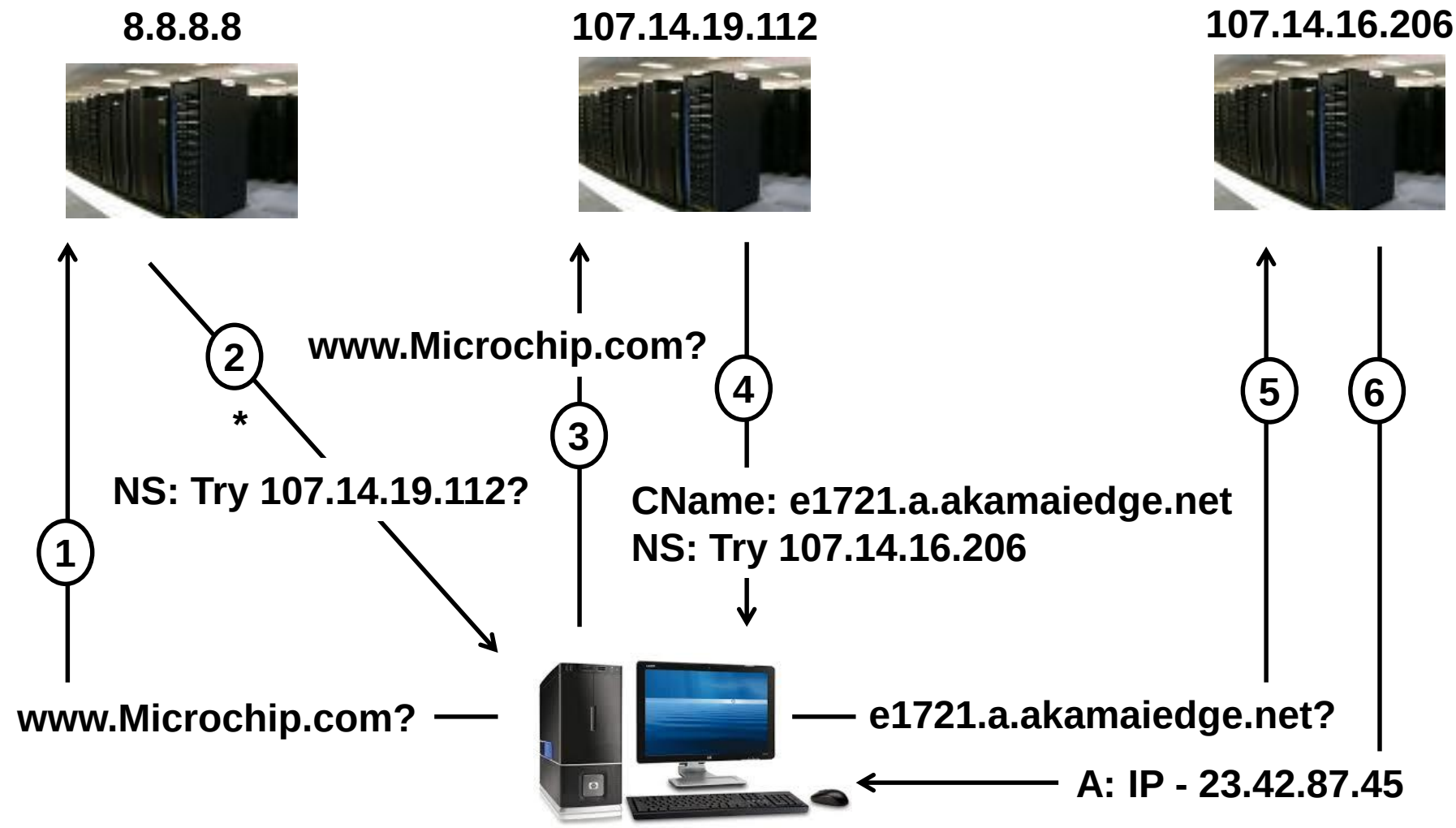
6

e1721.a.akamaiedge.net?

A: IP - 23.42.87.45

*If a DNS server supports recursive resolution, the 1st query will likely return the address record (A) with the requested IP

*Domain Name Resolution



*If a DNS server supports recursive resolution, the 1st query will likely return the address record (A) with the requested IP

Class Agenda

- **Network Fundamentals**
  - ARP – Address Resolution Protocol
  - IP Routing
  - DHCP – Dynamic Host Configuration Protocol
  - DNS – Domain Name System
- **Digilent Embedded IP Stack for chipKIT™ (deIP™ / DEIPck)**
- **HTTP Example Server**
- **LAB 1: Build and running the deIP™ HTTP Example Server**

Digilent Embedded IP Stack

C based deIP™ Core
C++ DEIPcK for chipKIT™

chipKIT™ Network Libraries

- **Digilent provides three network libraries:**
  - chipKITEthernet
  - DNETcK / DWIFIcK
  - DEIPcK / DEWFcK
- **chipKITEthernet is for legacy Arduino compatibility. Don't use it!**
- **DNETcK / DWIFIcK Digilent's Internet Protocol Suite library built on the Microchip MLA**
- **DEIPcK / DEWFcK Digilent's Open Source Internet Protocol Suite library which supports both the PIC32MX and PIC32MZ MCUs**

deIP™

- **Digilent Embedded IP Stack**
  - Mostly, RFC 1122 / 793 compliant
  - Open source under the BSD 3-clause license
  - Supports multiple concurrent network interfaces
- **Written in C**
  - Processor independent
- **Processor Specific Hardware Abstraction Layer**
  - Big/Little Endian, Timers, Checksum, Processor speed
- **MAC/PHY Abstraction Layer (Network Adaptors)**
- **Memory Abstraction Layer**
  - Network Packets and Socket Buffers
- **Designed Specifically for a cooperative non-preemptive embedded environment**

DEIPcK

- **Digilent Embedded IP Stack for the chipKIT™ Environment**
  - deIP™ C++ wrapper classes specifically as an MPIDE library
    - **DEIPcK / DEWFcK**
    - **TCPSocket / TCPServer**
    - **UDPSocket / UDPServer**
  - Closely resembles the DNETcK Network Library

DNETcK vs DEIPcK

DNETcK / DWIFIcK

- **Supports the following MAC/PHY:**
  - PIC32 MAC, SMSC LAN8720 PHY
  - Microchip ENC28J60 MAC/PHY
  - Microchip ENC424J600 MAC/PHY
  - Microchip MRF24WB0MA 802.11b Module
  - Microchip MRF24WG0MA 802.11g Module
- **PIC32MX MCU ONLY**
- **Is not open source, built on a slightly modified private copy of the MLA**

DEIPcK / DEWFcK

- **Supports the following MAC/PHY:**
  - PIC32 MAC, SMSC LAN8720 PHY
  - Microchip MRF24WG0MA 802.11g Module
  - Easy to add support for other MAC/PHY through the Network Adaptor Abstraction Layer
- **PIC32MX / MZ MCU support**
- **Completely Open Source, no plib, no MLA**
- **Memory Abstraction Layer**

Network App Rules

- **No Real Time Kernel**
- **Network stack must be run regularly**
  - DEIPcK::periodicTasks()
- **No operations should block for extended periods**
- **loop() must service everything in application; including the network stack**
- **Keep function operations short to prevent starving other functions in loop()**

DEIPcK

- **Class focus on DEIPcK, our 3rd generation stack, but most rules also apply to DNETcK**
- **All network functions return immediately**
  - Unlike DNETcK, DEIPcK removed all previsions to block on a Method
- **Poll until the operation completes or gets a hard error**
- **Parameters MUST remain valid until the operation completes, so string and structure parameters should be declared static or global**

Network Header Files

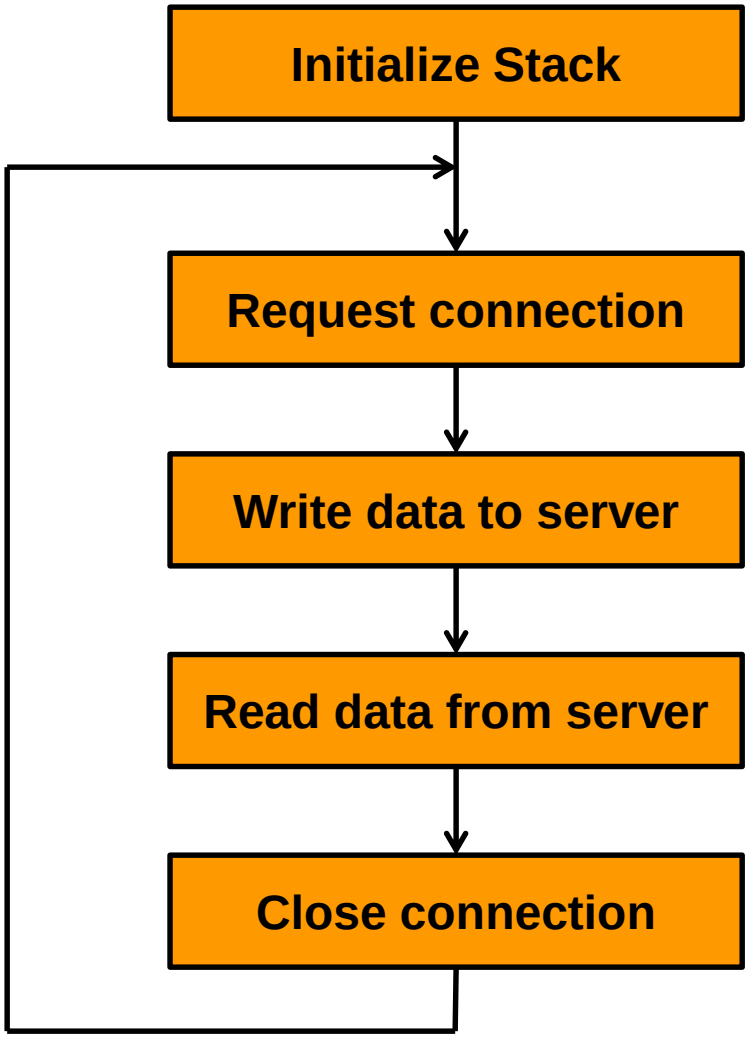
Header files used to specify hardware and stack support required. Must be put in your main sketch .pde

```
// You MUST select 1 and ONLY 1 of the following hardware libraries
// A hardware library specifies the Network Adaptor to use
#include <MRF24G.h>          // This is for the MRF24WGxx
//#include <IM8720PHY.h>  // This is for the Internal MAC and SMSC 8720 PHY

// The base network library is a required library
#include <DEIPcK.h>

//  -----  COMMENT THIS OUT IF YOU ARE NOT USING WIFI  -----
#include <DEWFcK.h>
```

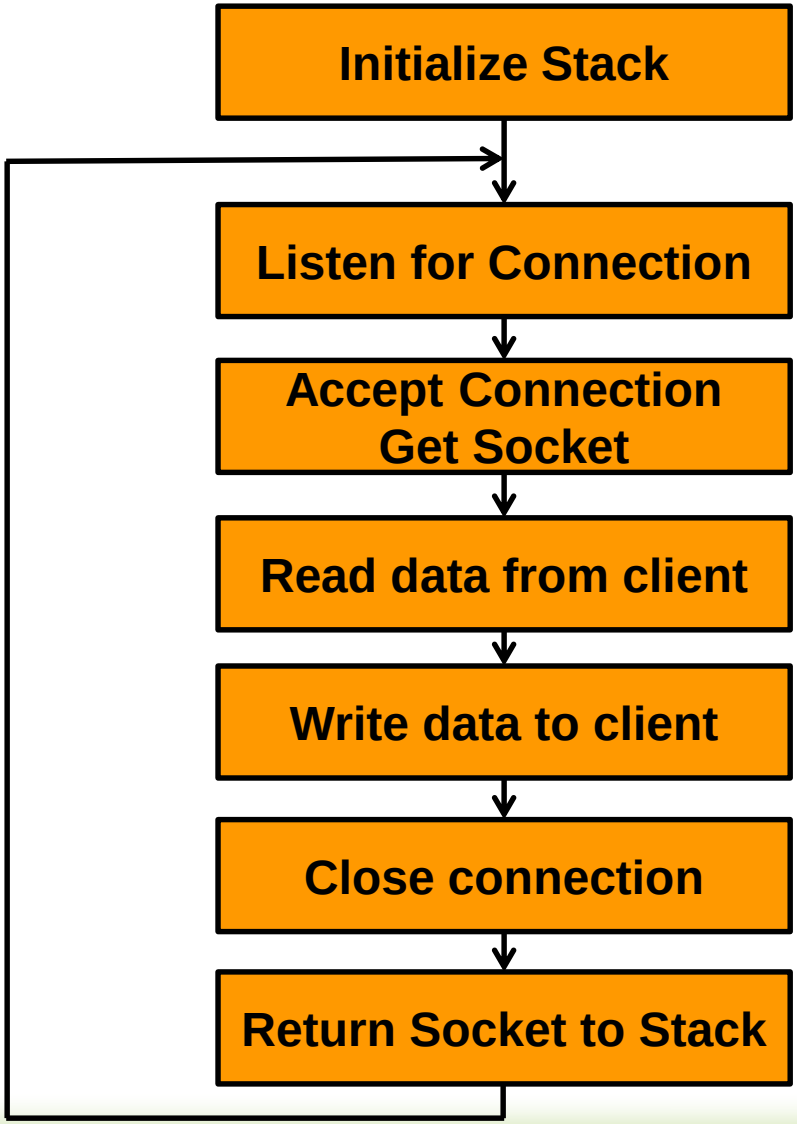
Core Network Concepts

- **Client-Server vs Peer-To-Peer**
- **Endpoint addresses**
  - IP Address and Port
- **Sockets**
  - Endpoint pairs, Socket Buffers
- **TCP, connections, reliability**
- **UDP, connectionless datagrams, unreliable**

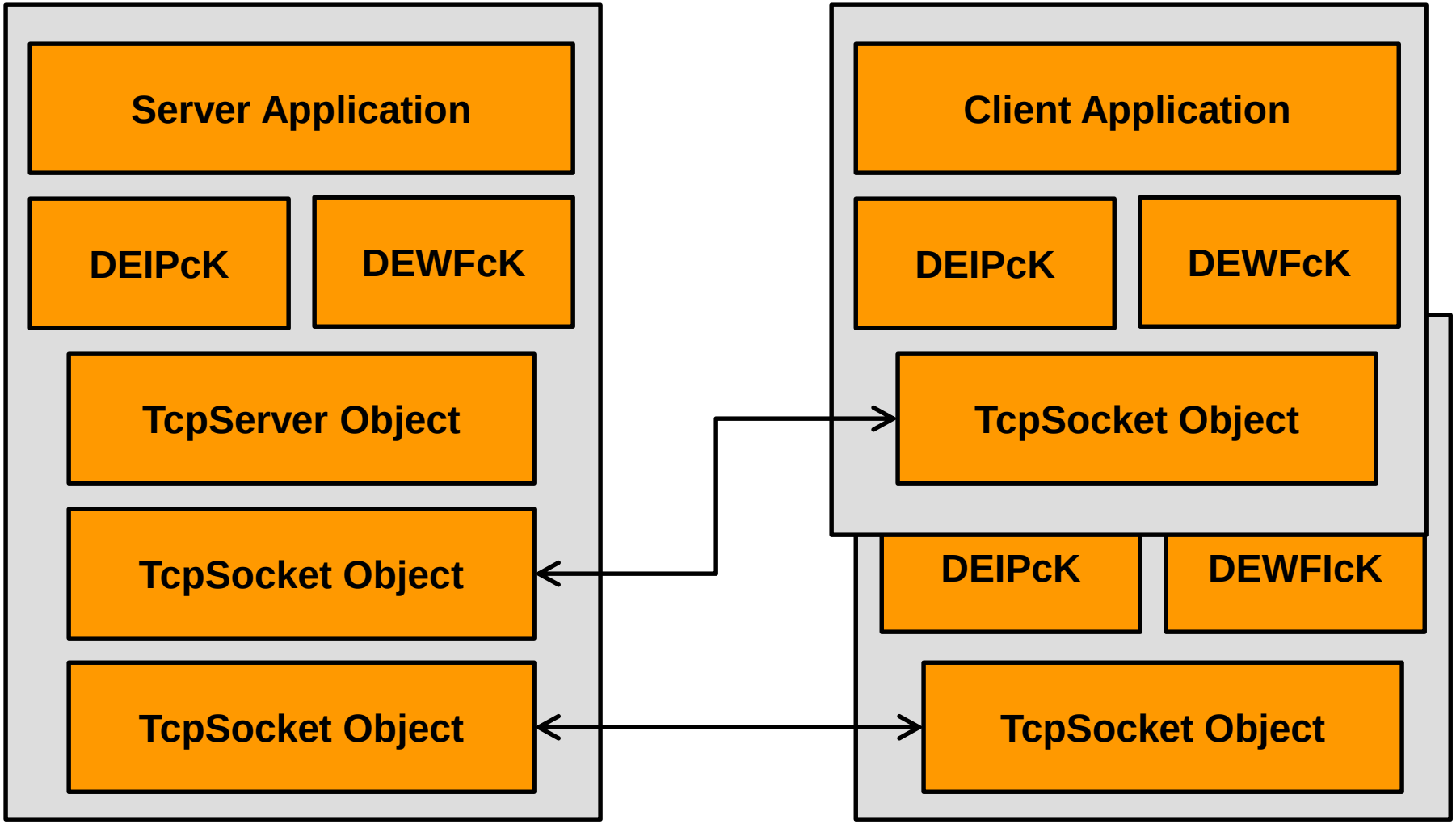
Client Application



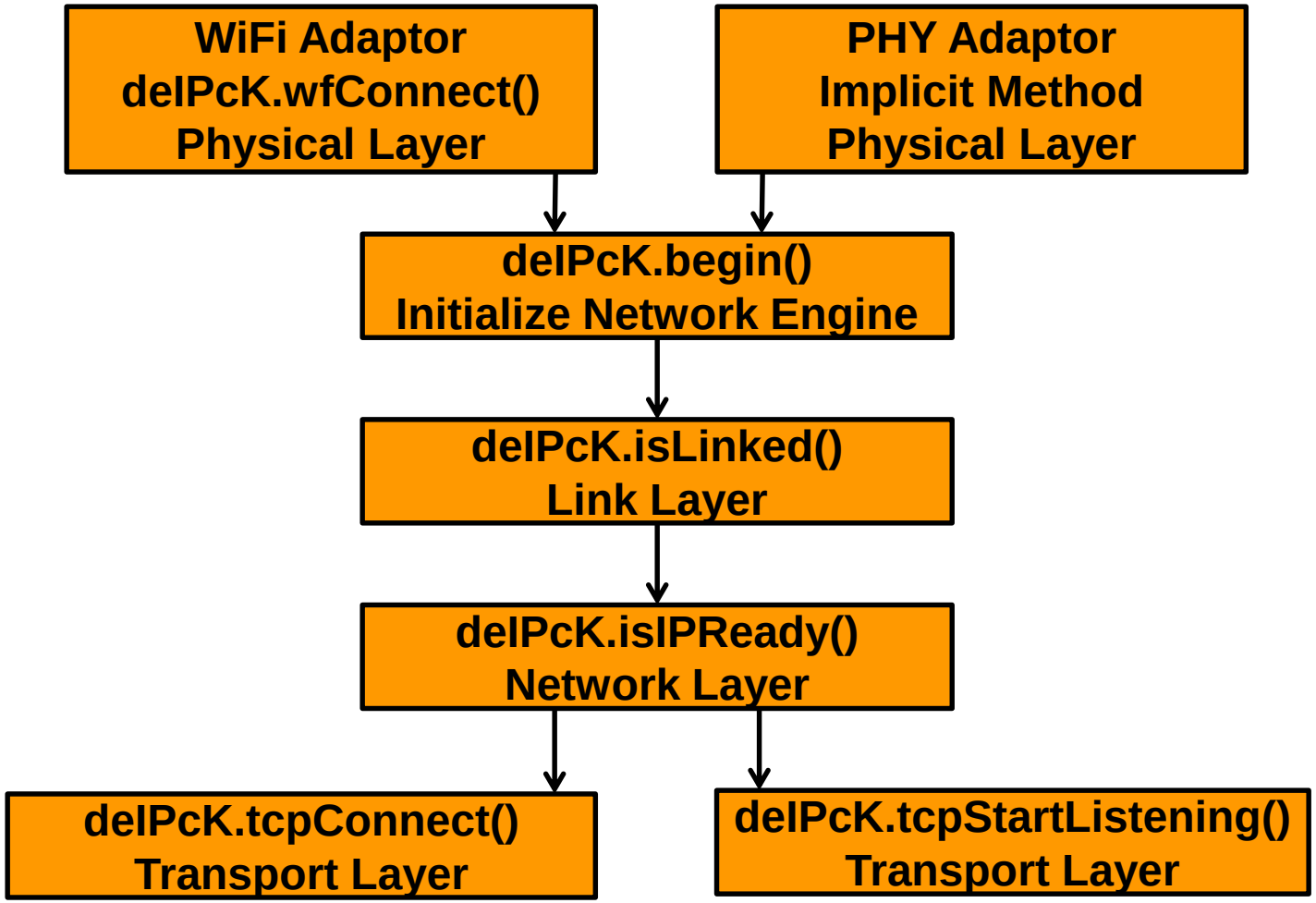
Server Application



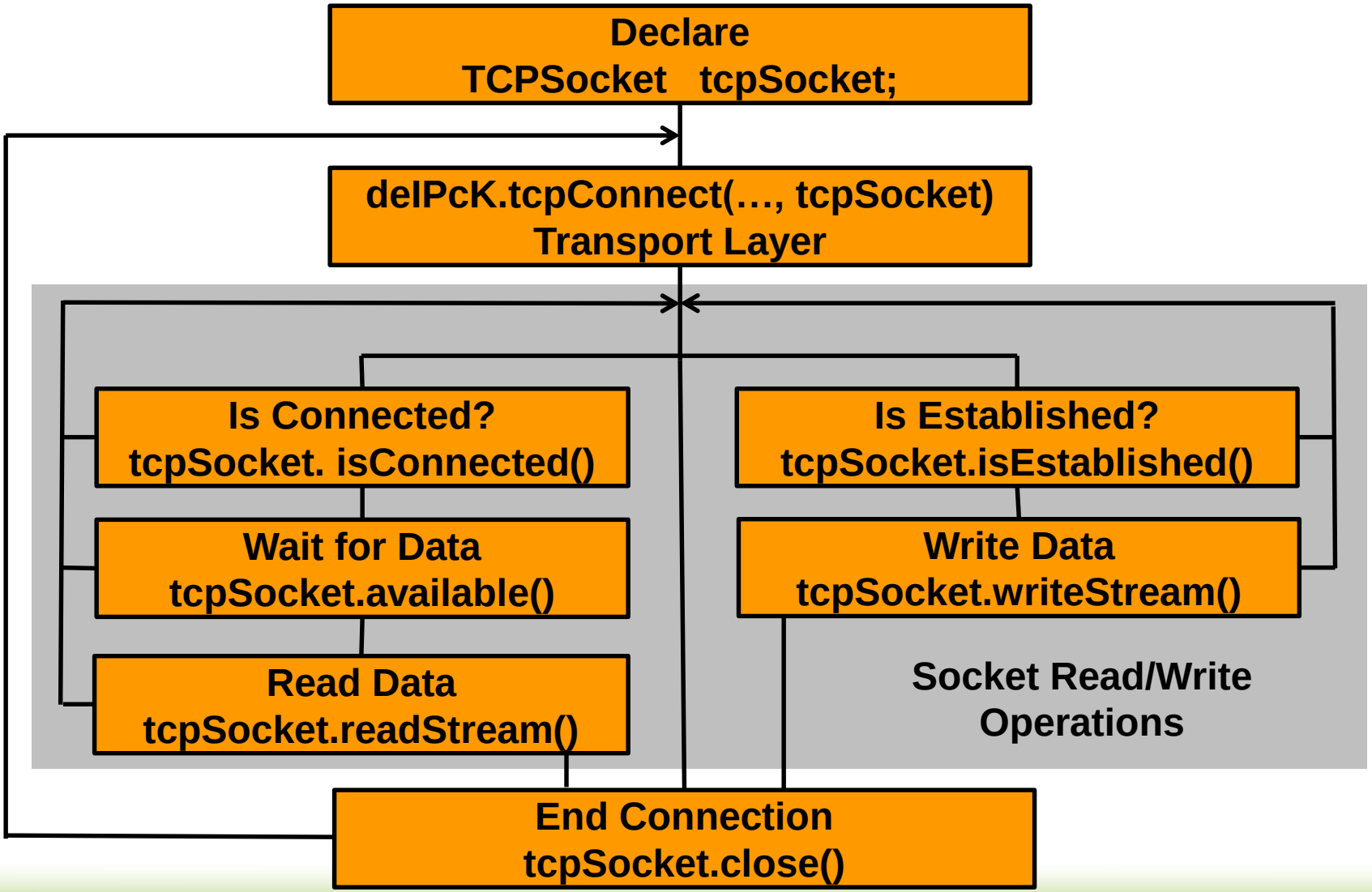
Network Applications



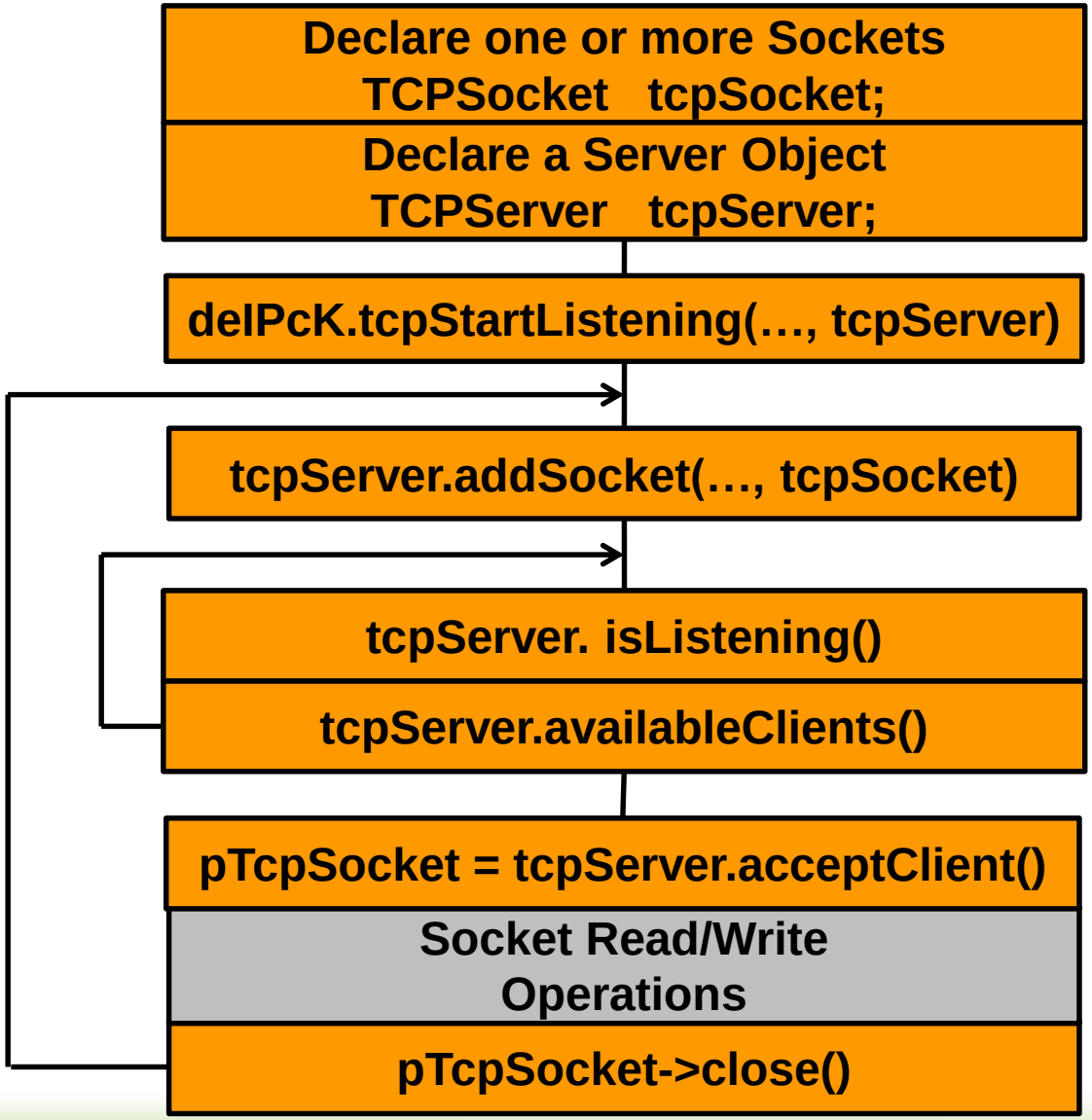
DEIPcK Initialization



TCP Socket



TCP Server



Generic State

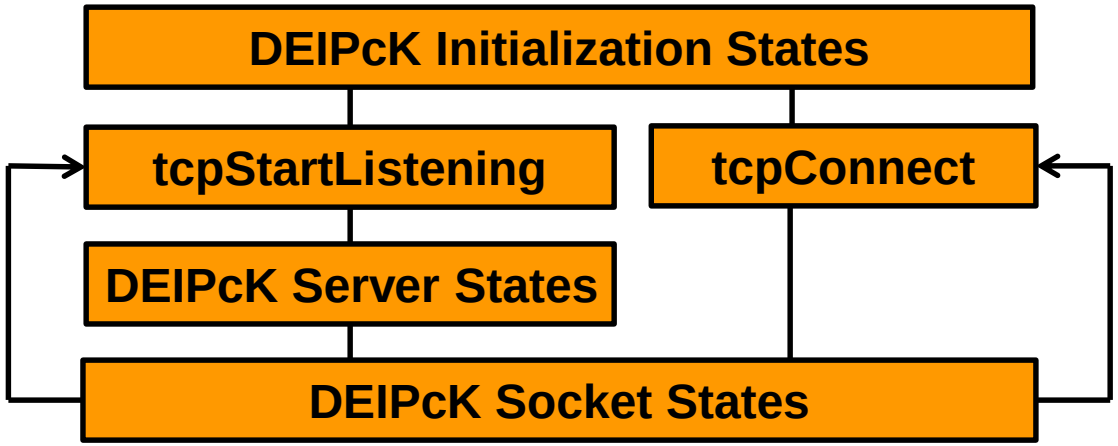
```
case currentState:

    // the primary method call
    if( delPcK.method(Param1, Param2, ParamX, &status) )
    {
       …. code …
       state = nextState;
    }

    // Error condition
    else if( IsIPStatusAnError(status) )
    {
       state = errrorState;
    }

    // optional timeout condition
    else if( millis() - tStartTime >= TIMEOUT )
    {
        state = timeoutState;
    }
    break;
```

Wi-Fi® Connect Example
Application Code

```
case WFCONNECT:
        if(delPcK.wfConnect(szSsid, szPassPhrase, &status))
        {
           Serial.println("WiFi connected");
           state = BEGIN;
        }
        else if(IsIPStatusAnError(status))
        {
           Serial.print("Unable to connect WiFi, status: ");
           Serial.println(status, DEC);
           state = WFERROR;
        }
       else if( millis() - tStartTime >= TIMEOUT )
        {
           state = WFTIMEOUT;
        }
        break;
```

DEIPcK State Structure

```
loop()
{
        switch(delPState)
        {
        }

}
```



DEIPcK::periodicTasks();
Run at lease once per loop()

MICROCHIP MASTERs 2014

Class Agenda

- **Network Fundamentals**
  - ARP – Address Resolution Protocol
  - IP Routing
  - DHCP – Dynamic Host Configuration Protocol
  - DNS – Domain Name System
- **Digilent Embedded IP Stack for chipKIT™ (deIP™ / DEIPck)**
- **HTTP Example Server**
- **LAB 1: Build and running the deIP™ HTTP Example Server**

HTTP Example Server

delP™ HTTP Example Server

- Embedded HTTP Server
- Built on the chipKIT™ DEIPcK Open Source Stack… built on the Open Source C delP™ Stack
- Standalone implementation to host HTML pages from the µSD card, no coding required
- Extensible to add dynamically created pages
- Highly cooperative embedded model allows for concurrent multiple connections

MICROCHIP MASTERs 2014

Class Agenda

- **Network Fundamentals**
  - ARP – Address Resolution Protocol
  - IP Routing
  - DHCP – Dynamic Host Configuration Protocol
  - DNS – Domain Name System
- **Digilent Embedded IP Stack for chipKIT™ (deIP™ / DEIPck)**
- **HTTP Example Server**
- **LAB 1: Build and running the deIP™ HTTP Example Server**

LAB 1

Build and run the deIP™ HTTP Example Server

Class Agenda Continued

- **HTTP Protocol Fundamentals**
- HTML Syntax Fundamentals
- HTTP Server Architecture
- LAB 2: Working with Static HTML Pages
- HTTP Server and Dynamic HTML Pages
- LAB 3: Working with Dynamic HTML Pages
- Additional: Debugging the HTTP Server
  - At the end of the slide deck for your review

HTTP Protocol Fundamentals

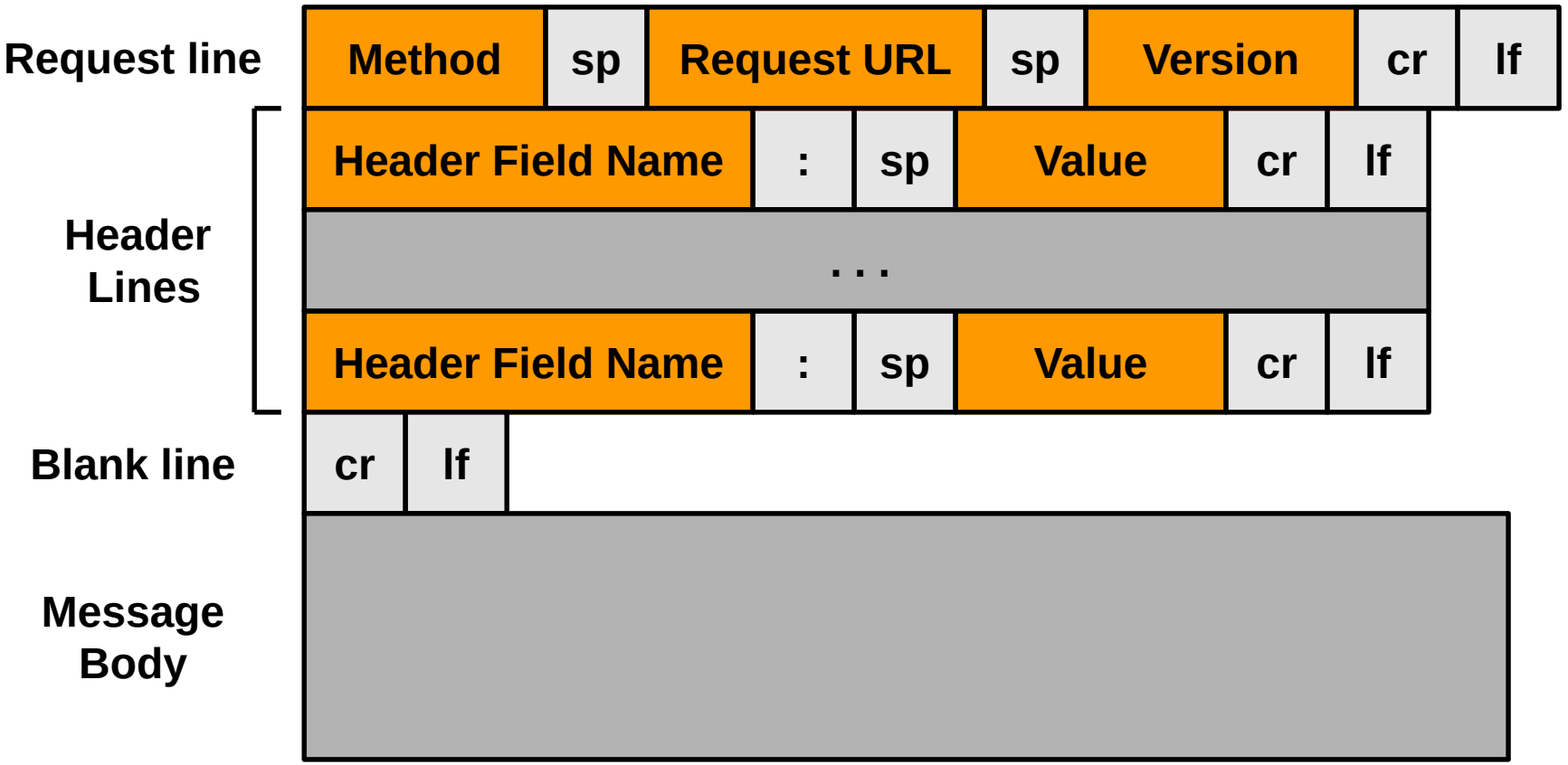
HTTP Protocol

- HTTP – Hyper-Text Transfer Protocol is application level protocol used by World Wide Web
- Text based, streaming protocol that uses TCP connections for network transport
- Client-server model used. Web browser is typically client application accessing HTTP servers that serve web pages
- Client sends HTTP Request messages to server
- Server sends HTTP Response messages to client

HTTP Messages

- **HTTP messages either request or response**
- **Made up of three parts:**
  - Request Line
  - Header Lines
  - Body
- **Request line specifies kind of request or response (message type… GET, PUT…)**
- **Request line and headers are lines of text delimited by <CR><LF>**

Request Message



HTTP Messages

- **Request line is required**
- **Header lines are optional, there can be a variable number of them**
- **Blank line delimits the header from the message body**
- **Message body is optional, will not be present in some messages**
- **Message body can contain text or arbitrary binary data**
- **Type and length of data in message body is described using header lines**
  - Content-Type, Content-Length, Content-Language, etc.

HTTP Methods

- **GET**
  - Request the URL resource
- **POST**
  - Annotate an existing URL resource on the HTTP server
- **PUT**
  - Create/Modify a resource under the provided URL
- **HEAD, DELETE, TRACE, OPTIONS, CONNECT, PATCH**
  - Go look them up: http://en.wikipedia.org/wiki/HTTP_method#Request_methods

HTTP Header Lines

- **None are required but some are good to have**
  - ContentType: <MIME type> i.e. text/html
    - Defines what content is in the body
  - ContentLength: <number of bytes>
    - Defines how long the body is in bytes
  - Cache-Control: no-cache
    - Tells the browser not to cache the page
  - Connection: close / keep-alive
    - Tells what to do with the TCP connection when done

HTTP GET Request

```
GET / HTTP/1.1
Accept: */*
Accept-Language: en-US
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0;
    Windows NT 6.1; WOW64; Trident/4.0; SLCC2;
    .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET
    CLR 3.0.30729; Media Center PC 6.0; .NET4.0C;
    .NET4.0E; .NET CLR 1.1.4322; MS STORE
    DMC2.8.4431.2)
Accept-Encoding: gzip, deflate
Host: 192.168.10.153
Connection: Keep-Alive
```

HTTP Response

HTTP/1.1 200 OK
Server: Apache
Content-language: en
Vary: Accept-Encoding,Cookie
Last-Modified: Sun, 15 Jun 2014 16:56:01 GMT
ContentEncoding: gzip
ContentType: text/html; charset=UTF-8
ContentLength: 46764
Accept-Ranges: bytes
Date: Tue, 17 Jun 2014 06:20:56 GMT
Age: 32601
Connection: close

Class Agenda Continued

- HTTP Protocol Fundamentals
- **HTML Syntax Fundamentals**
- HTTP Server Architecture
- LAB 2: Working with Static HTML Pages
- HTTP Server and Dynamic HTML Pages
- LAB 3: Working with Dynamic HTML Pages
- Additional: Debugging the HTTP Server
  - At the end of the slide deck for your review

HTML Syntax Fundamentals

HTML

- HTML, Hypertext Markup Language, primary data format for rich text applications on worldwide web
- Markup languages used to ‘mark up’ text for formatting and annotation purposes or describe document structure
- SGML (ISO 8879), Standard Generalized Markup Language, a meta-language used to define markup languages
- HTML is ‘almost syntactically correct’ SGML markup language

HTML Example

```
<!DOCTYPE html>
<html>
<head>
  <title>An Example HTML Document</title>
</head>
<body>
  <!-- This is a comment -->
  <h2>My first HTML document!</h2>
 <p>Hello from <b>Microchip MASTERs</b> </p>
 <p> Produced by: <i>Gene Apperson</i>. </p>
</body>
</html>
```

HTML Tags

- Fundamental element of markup in HTML is tag
- Tag, and the corresponding end tag, bracket content elements of the document
- Tagged elements can (and generally will) be nested within other tagged elements
- Tag is made up of tag name inside ‘<‘ ‘>’ characters, e.g. <tag>
- End tag is the same except tag name is preceded with ‘/’, e.g. </tag>
- http://en.wikipedia.org/wiki/HTML_tag
- Very complex, use an HTML editor

Class Agenda Continued

- HTTP Protocol Fundamentals
- HTML Syntax Fundamentals
- **HTTP Server Architecture**
- LAB 2: Working with Static HTML Pages
- HTTP Server and Dynamic HTML Pages
- LAB 3: Working with Dynamic HTML Pages
- Additional: Debugging the HTTP Server
  - At the end of the slide deck for your review

HTTP Server Architecture

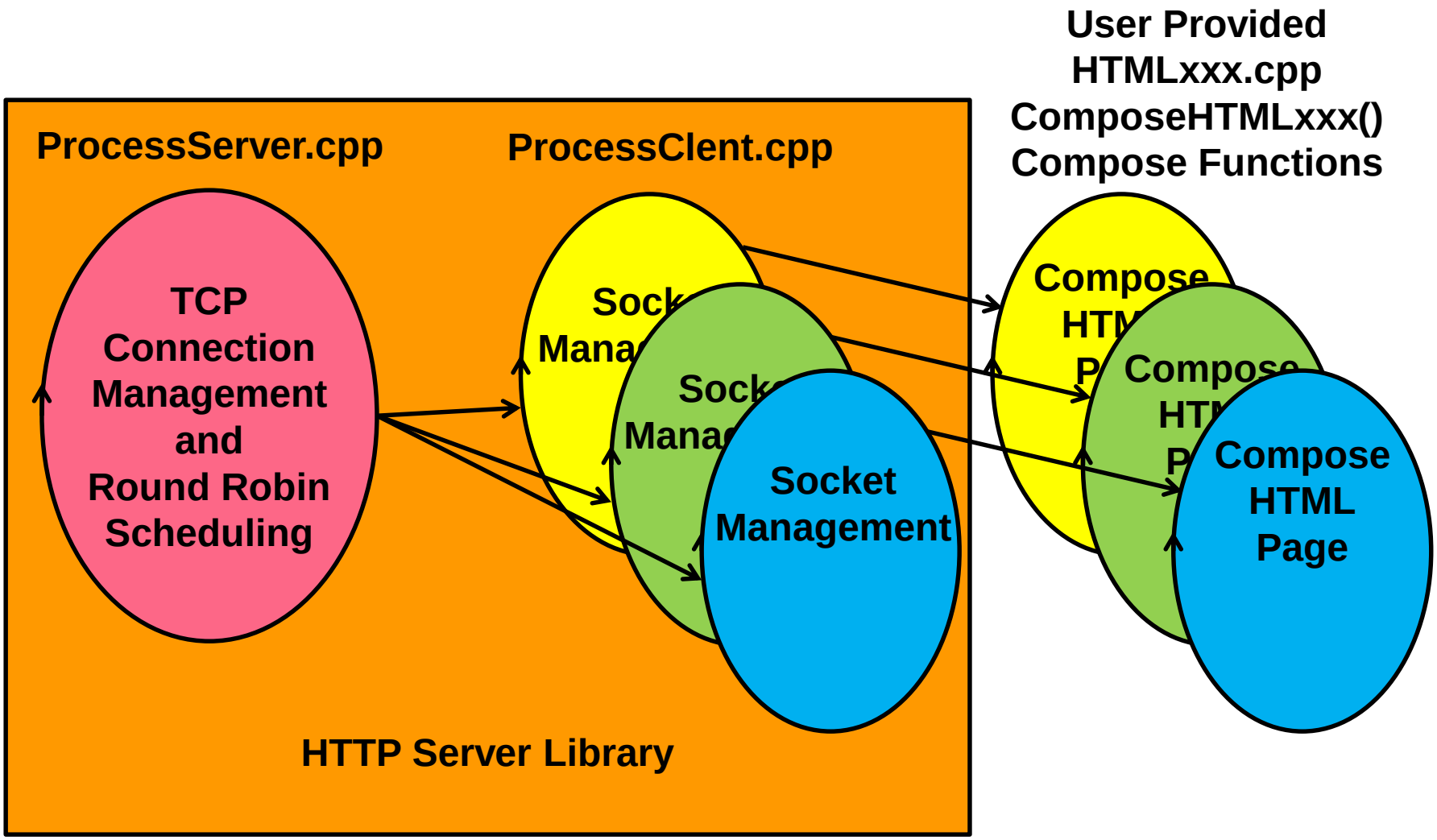
HTTP Example Server

- Light weight HTTP Server application framework; dirt simple!
- Built on DEIPcK networking classes
- Originally written as WebCam server (skunk project) adapted for general applications
- Abstracts networking and static HTML page hosting
- Provides for dynamic HTML page creation
- Provides helper functions to create basic HTTP headers
- Enables multiple concurrent connections and page processing

HTTP Server Library

- Implements base HTTP Server framework
- ProcessServer.cpp manages Network / WiFi connections, TCP sockets, cooperative task scheduling
- ProcessClient .cpp manages reading / writing data from/to the TCP socket, URL identification, line parsing, and calling Compose functions
- Implements some default and helpful HTML Compose functions, such as ComposeHTMLSDPage() for reading HTML pages off µSD card
- Implements other helpful functions

HTTP Server



HTTP Server Sketch Sources

- **HTTPServerConfig.h**
  - WiFi and Network Configuration
- **deWebServer.pde**
  - Main Sketch Source
- **HTTP/HTMLxxx.cpp**
  - User provided HTTP/HTML dynamic page implementations; Compose Functions

HTTPServerConfig.h

- **#define cMaxSocketsToListen 5**
- **IPv4 ipMyStatic = {0,0,0,0};**
  - If 0, DHCP is used to assign the IP
  - If non-zero, you must set the following properly
    - **IPv4 ipGateway = {192,168,1,1};**
    - **IPv4 subnetMask = {255,255,255,0};**
    - **IPv4 rgIpDNS[] = {{8,8,8,8}, {8,8,4,4}};**
- **byte localStaticIP = 0;**
  - If 0, DHCP is used to assign the IP
  - If non-zero, DHCP is used to get network parameters and this will be the last octet of the IP
- **unsigned short listeningPort = 80;**

HTTPServerConfig.h

- **WiFi Config**
  - #define USE_WPA2_PASSPHRASE
  - const char * szSsid = “MySSID";
  - const char * szPassPhrase = “MyPassword";
  - Used by HTTPServer Lib / ProcessServer.cpp
- **Future development will allow for configuration to be on the µSD card**

Key setup() Components

- **Compose Function Forward Reference**
  - GCMD::ACTION ComposeHTMLMyPage(
    CLIENTINFO * pClientInfo);
- **Method/URL Match String**
  - const char szHTMLMyPage[] = "GET /MyPage.htm ";
  - You must get this correct, there is no syntax checking
- **Binding a Method/URL to Compose Func**
  - *AddHTMLPage*(szHTMLMyPage,
    ComposeHTMLMyPage);
- **Binding the Default Compose Func**
  - *SetDefaultHTMLPage*(ComposeHTMLSDPage);

deWebServer.pde setup()

Declare Forward Ref to Extern Compose Functions

Declare HTTP Method/Match URL Strings

```
void setup(void)
{
```

Bind the Match URLs with the Compose Functions with AddHTMLPage()

Define a Default Compose Function with SetDefaultHTMLPage(); Typically ComposeHTTP404Error() or ComposeHTMLSDPage()

Run ServerSetup() and optionally SDSetup() if the µSD card is used.

```
}
```

setup() Example

```
GCMD::ACTION ComposeHTMLSelectPicture(CLIENTINFO * pClientInfo);
GCMD::ACTION ComposeHTMLPostPicture(CLIENTINFO * pClientInfo);

// This is HTTP Request Line….
static const char szHTMLGetSelPic[]    = "GET /Post.htm ";
static const char szHTMLPostPic[]      = "POST /Post.htm ";

void setup(void)
{
   // Bind Match URL to Compose Function
   AddHTMLPage(szHTMLGetSelPic,    ComposeHTMLSelectPicture);
   AddHTMLPage(szHTMLPostPic,      ComposeHTMLPostPicture);

   // Bind Default Compose Function
   SetDefaultHTMLPage(ComposeHTMLSDPage);

   SDSetup();                  // Init SD card

   ServerSetup();              // Init Process Server
}
```

deWebServer.pde loop()

```
void loop(void)
{
    // process the HTTP Server
    ProcessServer();
}
```

This is it, nothing more.

Static HTML Pages

- Static pages reside on µSD card.
- HomePage.htm must exist at root of µSD filesystem; this is default page much like index.htm
- All filenames must use 8.3 naming convention
- Static pages processed by default (*SetDefaultHTMLPage)* system provided Compose Function ComposeHTMLSDPage()
- Use HTML editor to create pages

Class Agenda Continued

- HTTP Protocol Fundamentals
- HTML Syntax Fundamentals
- HTTP Server Architecture
- **LAB 2: Working with Static HTML Pages**
- HTTP Server and Dynamic HTML Pages
- LAB 3: Working with Dynamic HTML Pages
- Additional: Debugging the HTTP Server
  - At the end of the slide deck for your review

LAB 2

Working with Static HTML Pages

Class Agenda Continued

- HTTP Protocol Fundamentals
- HTML Syntax Fundamentals
- HTTP Server Architecture
- LAB 2: Working with Static HTML Pages
- **HTTP Server and Dynamic HTML Pages**
- LAB 3: Working with Dynamic HTML Pages
- Additional: Debugging the HTTP Server
  - At the end of the slide deck for your review

HTTP Server and Dynamic HTML Pages

Dynamic HTML Pages

- All Pages Dynamic Pages
- All Pages created in Compose Functions
- System provided Dynamic Compose Function reads static HTML pages off µSD Card; ComposeHTMLSDPage()
- Technically Compose Functions render HTTP Response Header and Body
- HTTP Response Header typically created using BuildHTTPOKStr() helper function

Compose Functions

- **Callback functions are of the form:**
  - GCMD::ACTION ComposeHTMLxxxx(CLIENTINFO * pClientInfo);
- **Implemented as state machine with each state only implementing small fraction of the work; no more than about a millisecond (i.e. keep it short!)**
- **ProcessServer() assigns TCP connection and Socket to each HTTP Method Request (GET / POST)**
- **ProcessClient() manages socket and calls the bound Compose Function assigning it a CLIENTINFO structure for duration of connection**
- **Possible for multiple connections to be operating on the same URL, and same Compose Function concurrently (but different CLIENTINFOs)**
- **Compose Function typically occupies 1 .cpp file**

ClientInfo Structure

```
typedef struct CLIENTINFO_T
{
   // TCP ProcessClient state machine variables
   TCPSocket *     pTCPClient;
   uint32_t        clientState;
   uint32_t        nextClientState;
   uint32_t        tStartClient;
   uint32_t        cbRead;
   byte            rgbIn[CBCLILENTINPUTBUFF];
   byte            rgbOverflow[4];

   // HTML processing variables; Used in Compose functions
   uint32_t       htmlState;     // Compose function state
   uint32_t       cbWrite;       // How many bytes to write out
   uint32_t       cbWritten;     // How many bytes written
   byte           rgbOut[CBCLILENTOUTPUTBUFF];  // buffer space
   const byte *   pbOut;         // Data for processClient to write

   // pointer to <this> HTML page rendering function
   FNRENDERHTML    ComposeHTMLPage;
} CLIENTINFO;
```

ComposeHTMLxxxx()

```
GCMD::ACTION ComposeHTMLMyPage(CLIENTINFO * pClientInfo)
{
   switch(pClientInfo->htmlState)
   {
         case HTTPSTART:                  // ProcessClient() initializes here
               …
               break;

         case YourStates:
               ….
               break;

         case HTTPTIMEOUT:                // ProcessClient() calls if network
               …                          // times out, you should cleanup
               break;

         case HTTPDISCONNECT:             // ProcessClient() calls if network
               …                          // connection is dropped for
               return(GCMD::DONE);        // any reason, including normal
    }                                     // termination. You should cleanup
    return(GCMD::CONTINUE);
}
```

Example Compose Function

```
GCMD::ACTION ComposeHTMLSamplePage(CLIENTINFO * pClientInfo)
{
  GCMD::ACTION retCMD = GCMD::WRITE;

   switch(pClientInfo->htmlState)
   {
       case HTTPSTART:
         pClientInfo->cbWrite = BuildHTTPOKStr(true, sizeof(szSample)-1,
               ".htm", (char *) pClientInfo->rgbOut, sizeof(pClientInfo->rgbOut));
         pClientInfo->pbOut = pClientInfo->rgbOut;
         pClientInfo->htmlState = WRITECONTENT;
         break;
       case WRITECONTENT:
          pClientInfo->pbOut = (const byte *) szSample;
          pClientInfo->cbWrite = sizeof(szSample)-1;
          pClientInfo->htmlState = DONE;
          break;
      case DONE:
      default:
         pClientInfo->cbWrite = 0;
         retCMD = GCMD::DONE;
         break;
   }
   return(retCMD);
}
```

```
static const char szSample[] =
  "<head>\r\n\
  <title> HTTP Sample </title>\r\n\
  </head>\r\n\
  <body>\r\n\
  This is a simple HTML sample page.\r\n\
  <br />\r\n\
  </body>\r\n";
```

PreDefined Compose States

- **ProcessClient() will call the Compose Function with 3 predefined states**
  - HTTPSTART
    - **Initial state when new connection is accepted**
  - HTTPTIMEOUT
    - **Only called if no activity on connection in timeout period; user code SHOULD clean up;**
  - HTTPDISCONNECT
    - **Always called if connection is dropped, or closed for any reason, including normal completion. User code MUST clean up**

Compose Return Actions

- **GCMD::CONTINUE**
  - Current state is complete, no external action is needed
- **GCMD:: READ**
  - Read all available bytes from socket into input buffer pointed to by pClientInfo->rgbIn of length pClientInfo->cbRead
- **GCMD:: GETLINE**
  - Continues to read from socket until end of line (\r\n); returned in pClientInfo->rgbIn of length pClientInfo->cbRead
- **GCMD:: WRITE**
  - Writes out to socket pClientInfo-> cbWrite bytes from pClientInfo-> pbOut
- **GCMD:: DONE**
  - Compose function is done and connection to be closed

PreDefined Compose Functions

- **ComposeHTMLSDPage()**
  - Looks up a page from the µSD card
- **ComposeHTTP404Error()**
  - Returns an HTTP 404 File Not Found Error
- **ComposeHTMLRestartPage()**
  - Restarts the Network (DEIPcK)
- **ComposeHTMLTerminatePage()**
  - Halts the HTTP Server
- **ComposeHTMLRebootPage()**
  - Executes a soft reset of the Processor

Helper functions

- **BuildHTTPOKStr()**
  - Builds a minimal HTTP Header with content length and content type; used when successfully returning an HTML page
- **JumpToComposeHTMLPage()**
  - Jumps from one compose function to another; often used on error to jump to the ComposeHTTP404Error() HTTP page

MICROCHIP
MASTERs 2014

Class Agenda Continued

- HTTP Protocol Fundamentals
- HTML Syntax Fundamentals
- HTTP Server Architecture
- LAB 2: Working with Static HTML Pages
- HTTP Server and Dynamic HTML Pages
- **LAB 3: Working with Dynamic HTML Pages**
- Additional: Debugging the HTTP Server
  - At the end of the slide deck for your review

LAB 3

Working with Dynamic HTML Pages

Class Summary

- **Today we covered:**
  - Fundamentals of Network Topology
  - Fundamentals of the DEIPcK Network Stack
  - Fundamental Structure of HTTP and HTML
  - How to build the HTTP Example Server
  - How to work with Static HTML pages
  - How to Create Dynamic HTML pages

Dev Tools For This Class

- **Wi-Fi® Router**
- **MPIDE**
  - http://chipkit.net/started/install-chipkit-software/
- **WEB Browser (IE)**
  - http://us.downloadinfo.co/lp/internet-explorer/457/?sl=2
- **chipKIT™ uC32 Board**
  - http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,892,1035&Prod=CHIPKIT-UC32
  - https://www.microchipdirect.com/ProductSearch.aspx?Keywords=TDGL017

Dev Tools For This Class

- **Wi-Fi® Shield**
  - http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,892,1037&Prod=CHIPKIT-WIFI-SHIELD
  - https://www.microchipdirect.com/ProductSearch.aspx?Keywords=TDGL016
- **Basic IO Shield**
  - http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,892,936&Prod=CHIPKIT-BASIC-IO-SHIELD
  - https://www.microchipdirect.com/ProductSearch.aspx?Keywords=TDGL005
- **Micro SD Card & Reader/Writer**
  - http://www.staples.com/SanDisk-SDSDQM-MicroSD-High-Capacity-Flash-Memory-Card-With-Adapter-4GB/product_IM1DV7840

Dev Tools For Debugging

- **MPLAB® X IDE v2.10**
  - http://www.microchip.com/mplabx
- **chipKIT™ Programmer**
  - http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,892,1078&Prod=chipKIT PGM
  - https://www.microchipdirect.com/ProductSearch.aspx?Keywords=TDGL015

References

- **Links to chipKIT™ Documentation**
  - http://chipkit.net
- **Links to MPIDE**
  - http://chipkit.s3.amazonaws.com/index.html
- **Digilent's Website**
  - http://www.digilentinc.com
- **Microchip's Website**
  - http://www.microchip.com/
- **Links to MPLAB® X IDE**
  - http://www.microchip.com/mplabx
- **IETF RFCs**
  - http://ietfreport.isoc.org/rfc/PDF/

MICROCHIP
MASTERs 2014

LEGAL NOTICE

SOFTWARE:

You may use Microchip software exclusively with Microchip products. Further, use of Microchip software is subject to the copyright notices, disclaimers, and any license terms accompanying such software, whether set forth at the install of each program or posted in a header or text file.

Notwithstanding the above, certain components of software offered by Microchip and 3rd parties may be covered by "open source" software licenses – which include licenses that require that the distributor make the software available in source code format. To the extent required by such open source software licenses, the terms of such license will govern.

NOTICE & DISCLAIMER:

These materials and accompanying information (including, for example, any software, and references to 3rd party companies and 3rd party websites) are for informational purposes only and provided "AS IS." Microchip assumes no responsibility for statements made by 3rd party companies, or materials or information that such 3rd parties may provide.

MICROCHIP DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, INCLUDING ANY IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY DIRECT OR INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND RELATED TO THESE MATERIALS OR ACCOMPANYING INFORMATION PROVIDED TO YOU BY MICROCHIP OR OTHER THIRD PARTIES, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBLITY OF SUCH DAMAGES OR THE DAMAGES ARE FORESEEABLE. PLEASE BE AWARE THAT IMPLEMENTATION OF INTELLECTUAL PROPERTY PRESENTED HERE MAY REQUIRE A LICENSE FROM THIRD PARTIES.

TRADEMARKS:

The Microchip name and logo, the Microchip logo, dsPIC, FlashFlex, flexPWR, JukeBlox, KeeLoq, KeeLoq logo, Kleer, LANCheck, MediaLB, MOST, MOST logo, MPLAB, OptoLyzer, PIC, PICSTART, PIC32 logo, RightTouch, SpyNIC, SST, SST Logo, SuperFlash and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

The Embedded Control Solutions Company and mTouch are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, BodyCom, chipKIT, chipKIT logo, CodeGuard, dsPICDEM, dsPICDEM.net, ECAN, In-Circuit Serial Programming, ICSP, Inter-Chip Connectivity, KleerNet, KleerNet logo, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, RightTouch logo, REAL ICE, SQI, Serial Quad I/O, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

GestIC is a registered trademarks of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

Class Agenda Continued

- HTTP Protocol Fundamentals
- HTML Syntax Fundamentals
- HTTP Server Architecture
- LAB 2: Working with Static HTML Pages
- HTTP Server and Dynamic HTML Pages
- LAB 3: Working with Dynamic HTML Pages
- **Additional: Debugging the HTTP Server**
  - At the end of the slide deck for your review

Additional Topic

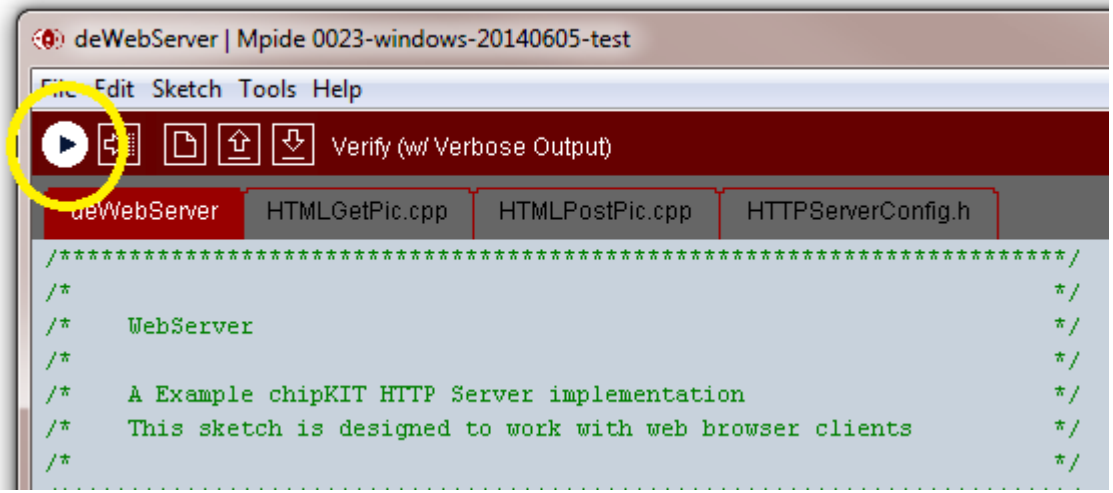
Debugging with MPLAB® X IDE

Debugging

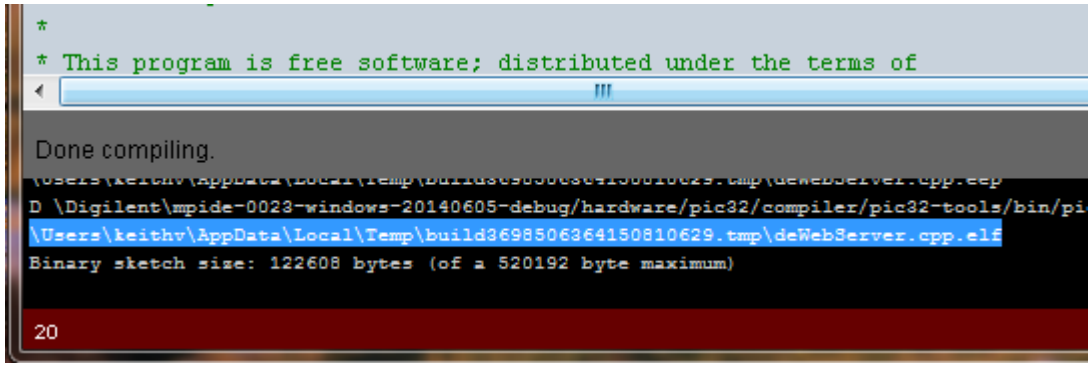
- **Compose Functions can be complex and Debugging in MPLAB® X IDE Highly Desirable**
  - Build in MPIDE
  - Import into MPLAB X IDE as a prebuilt project
  - Debug
  - Restore Sketch and Bootloader

Build in MPIDE

Do a verbose compile by doing a <shift> + compile



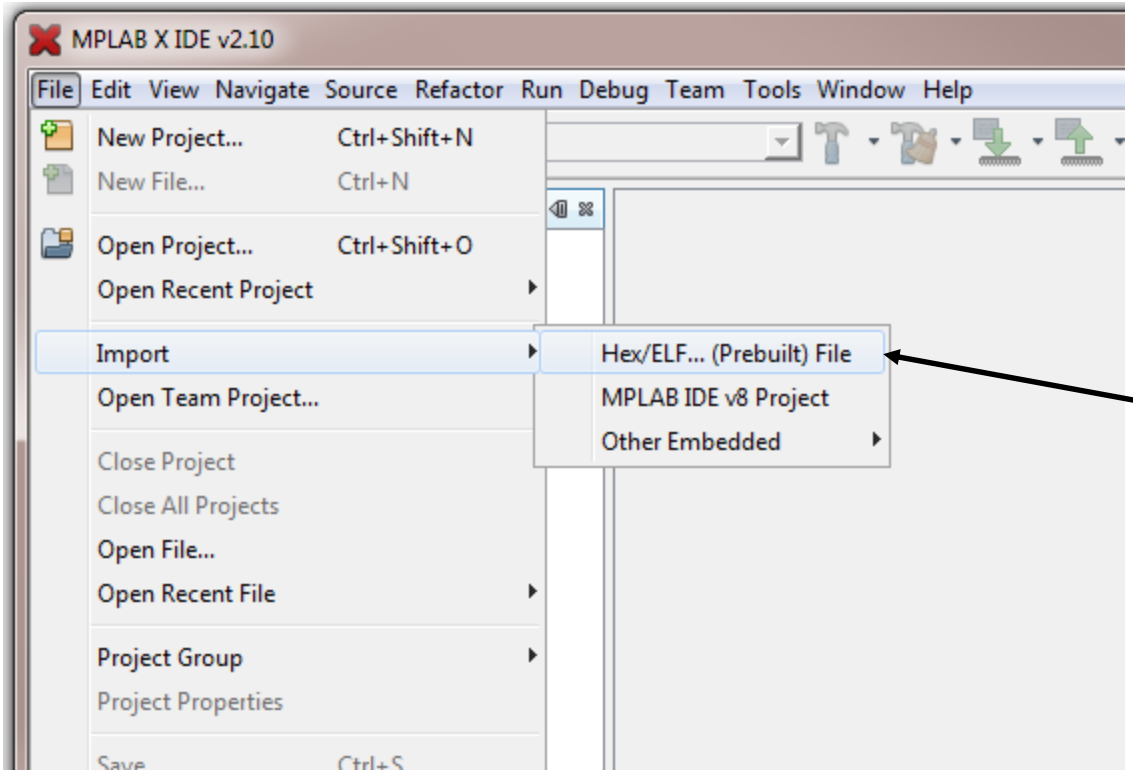
Copy into your clipboard the .elf file



Hardware Debugger

- **To use MPLAB® X IDE you need an ICSP™ Hardware Debugger**
  - chipKIT™ Programmer
  - PICkit™ 3 Programmer
- **Ensure Hardware Debugger is plugged into ICSP port on the board and USB connected to the computer**

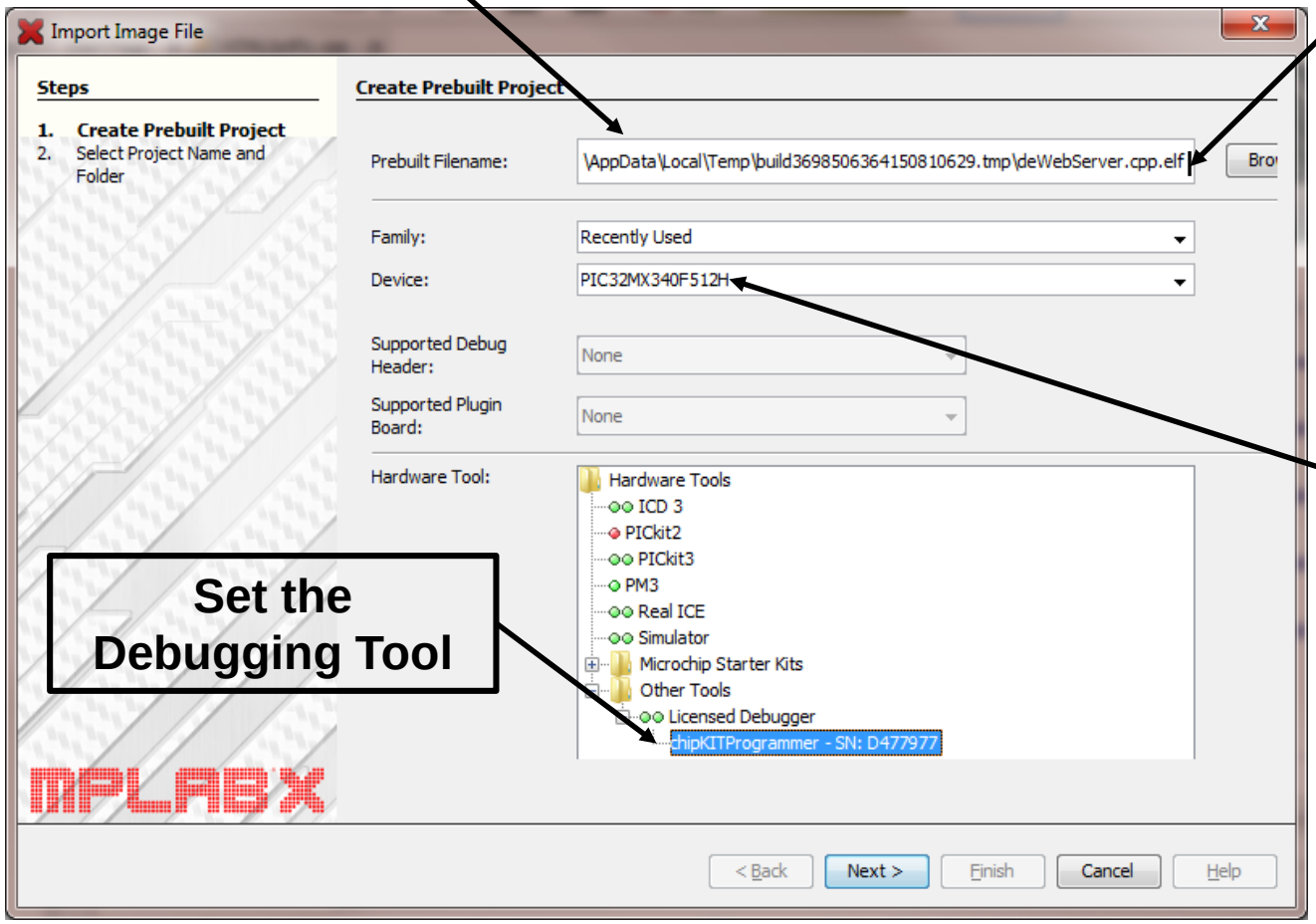
Create a Prebuilt Project



Start the Prebuilt Project Wizard

Import the .ELF File

Paste from your clipboard the .elf file

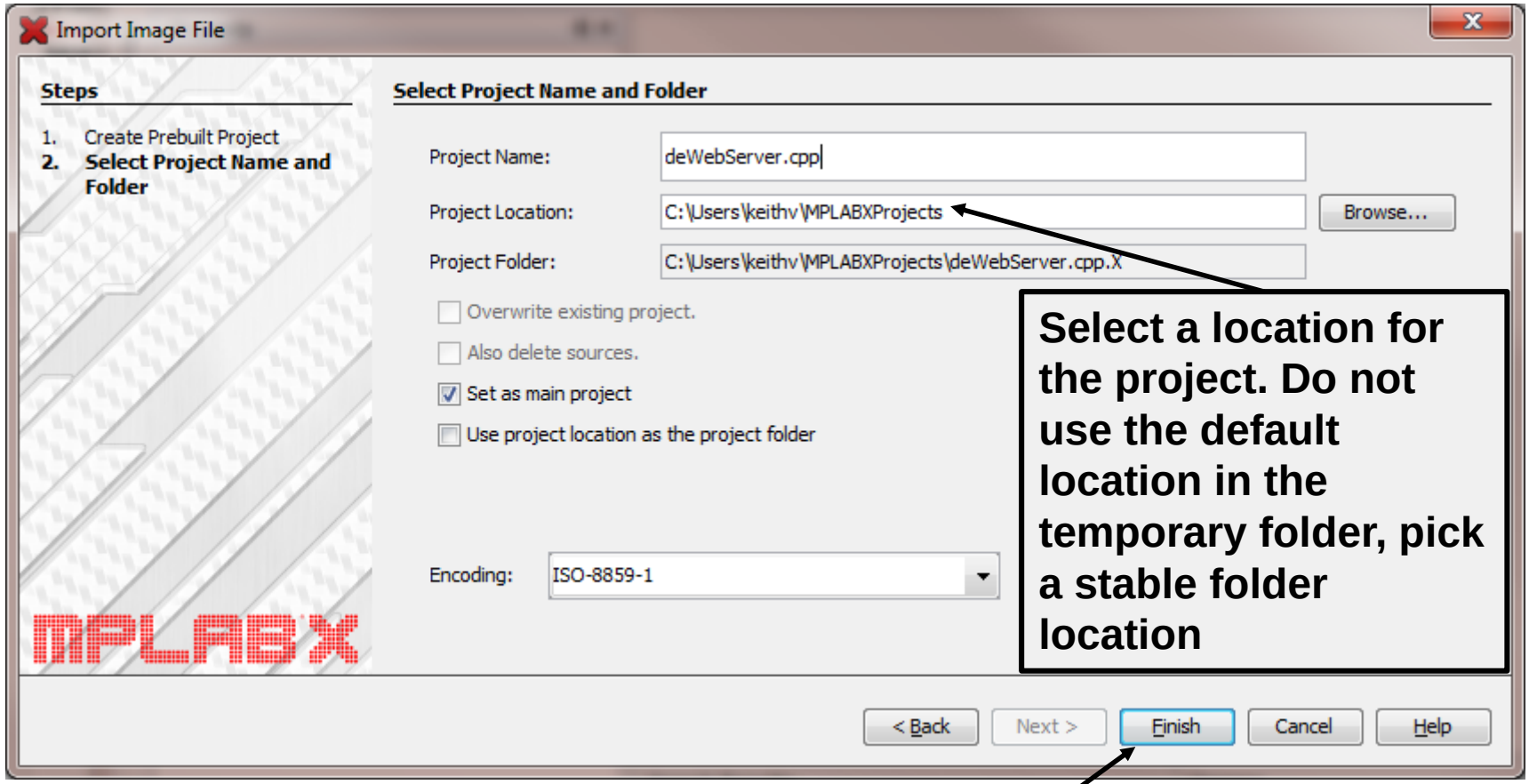


Hint: there will be a white space at the end of the filename and you will need to delete that space or MPLAB® X IDE will hang

Set the Processor

Set the Debugging Tool

Define a folder for the X Project



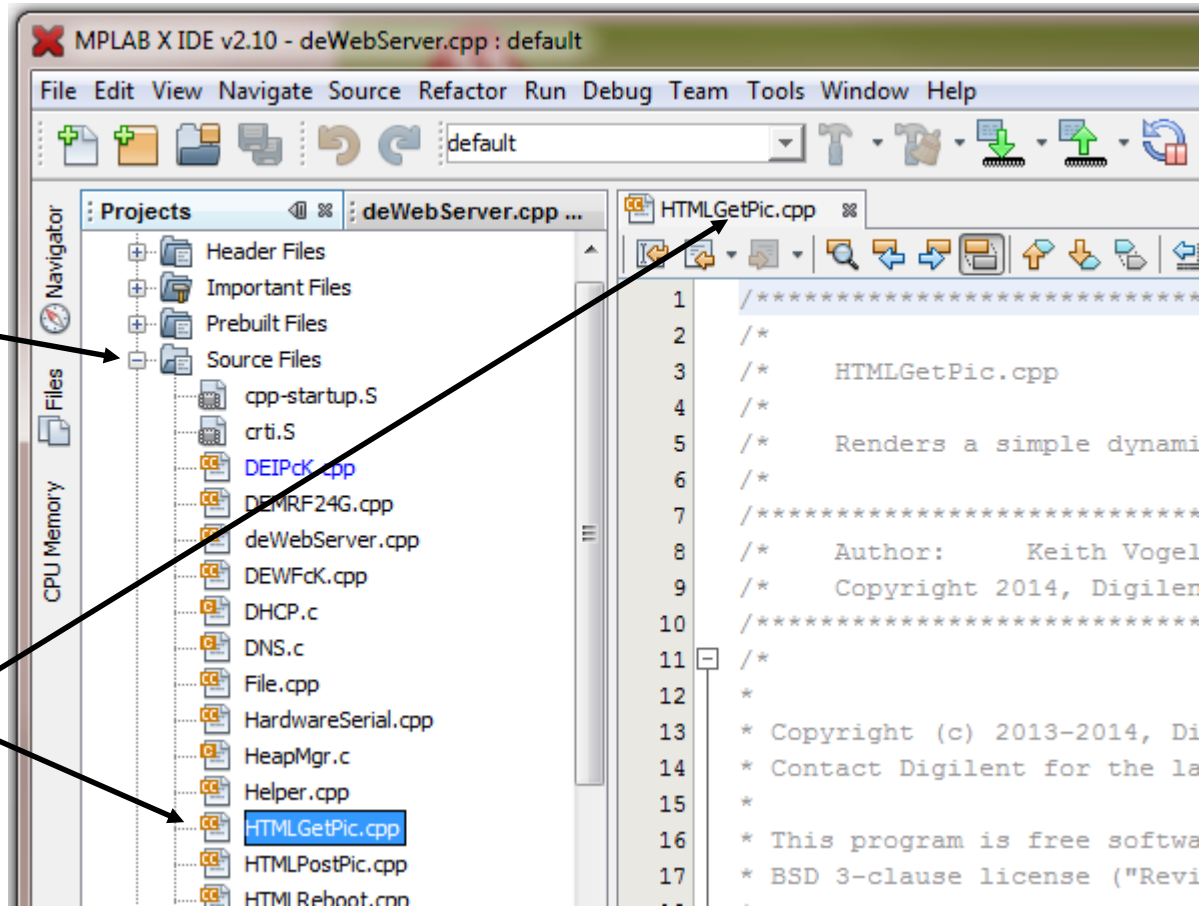
Select a location for the project. Do not use the default location in the temporary folder, pick a stable folder location

Finish the Wizard

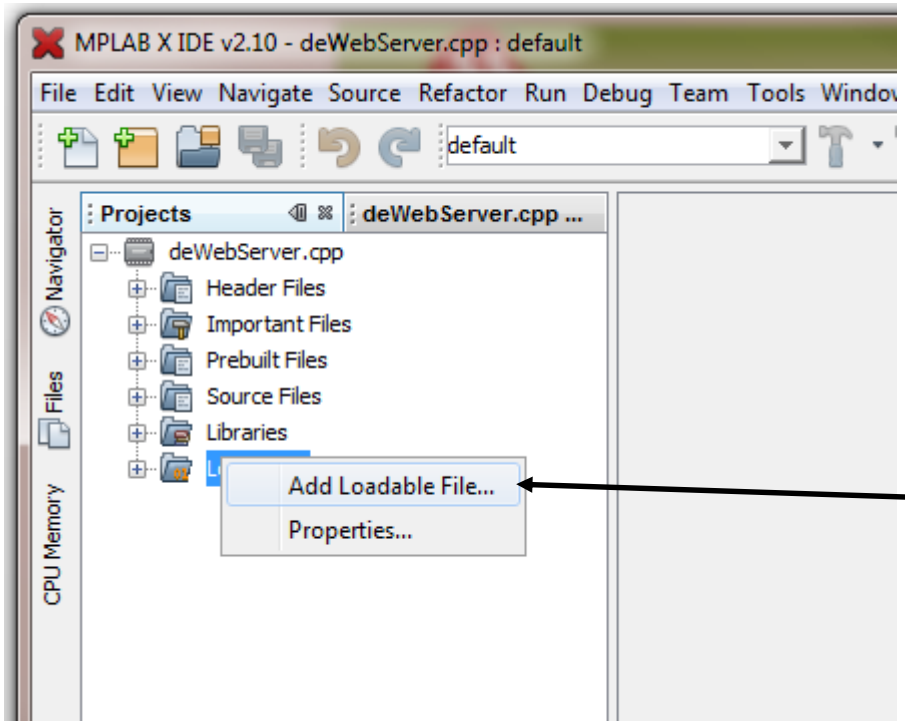
Automatic Source Load

The Wizard will automatically add you source files as discovered from the .elf file

Select the source, it will show up as a source tab

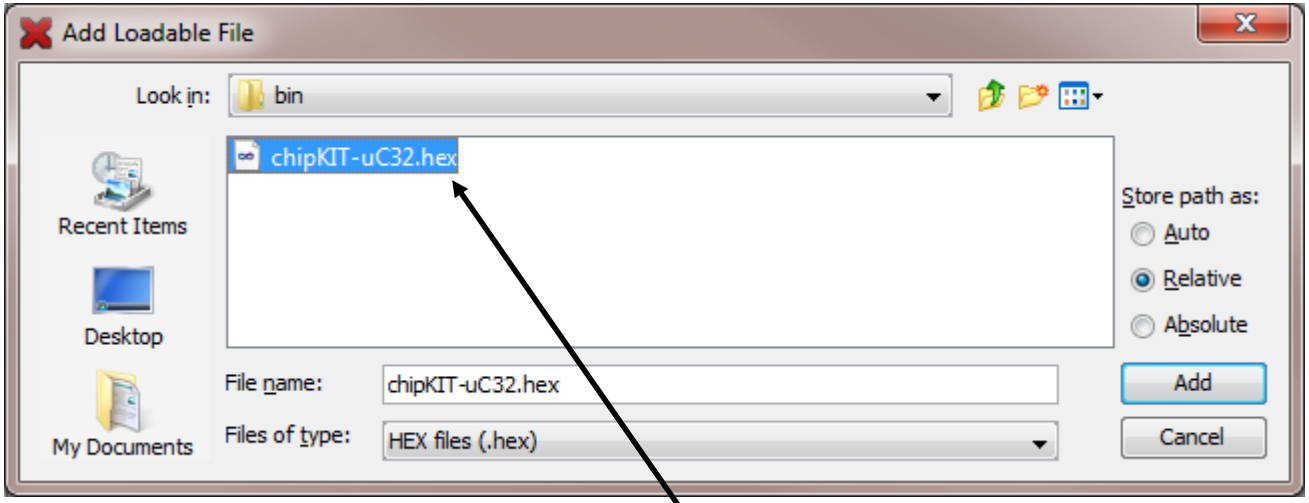


Add the Bootloader



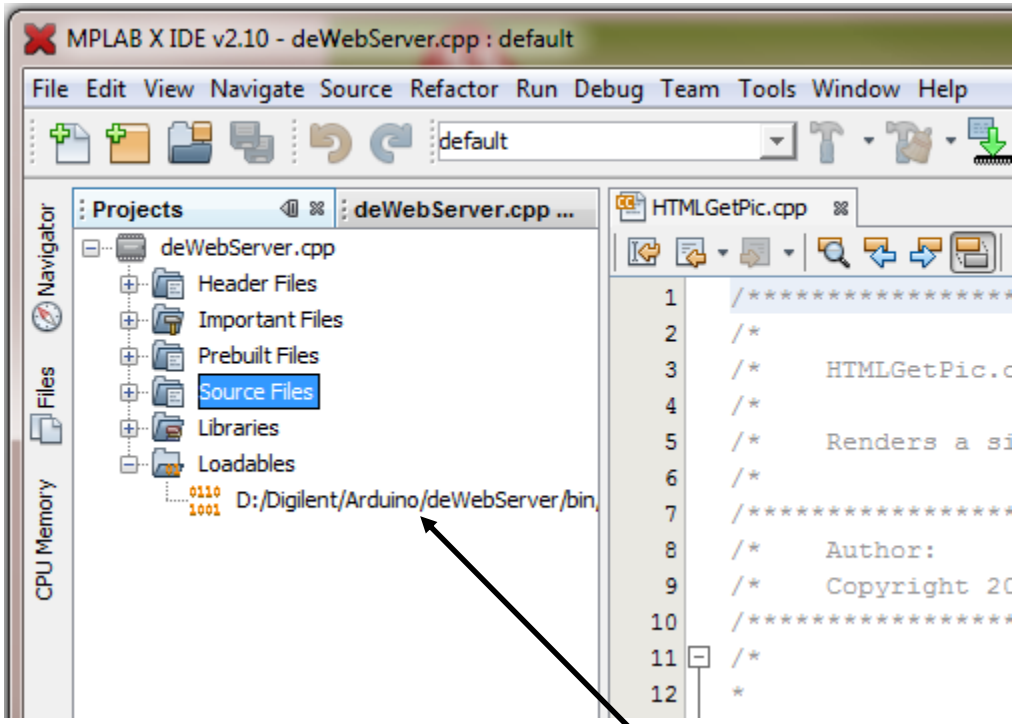
Now you must also add the bootloader as a Loadable File

Provide the bootloader .hex



Select the bootloader HEX file for the board you are debugging

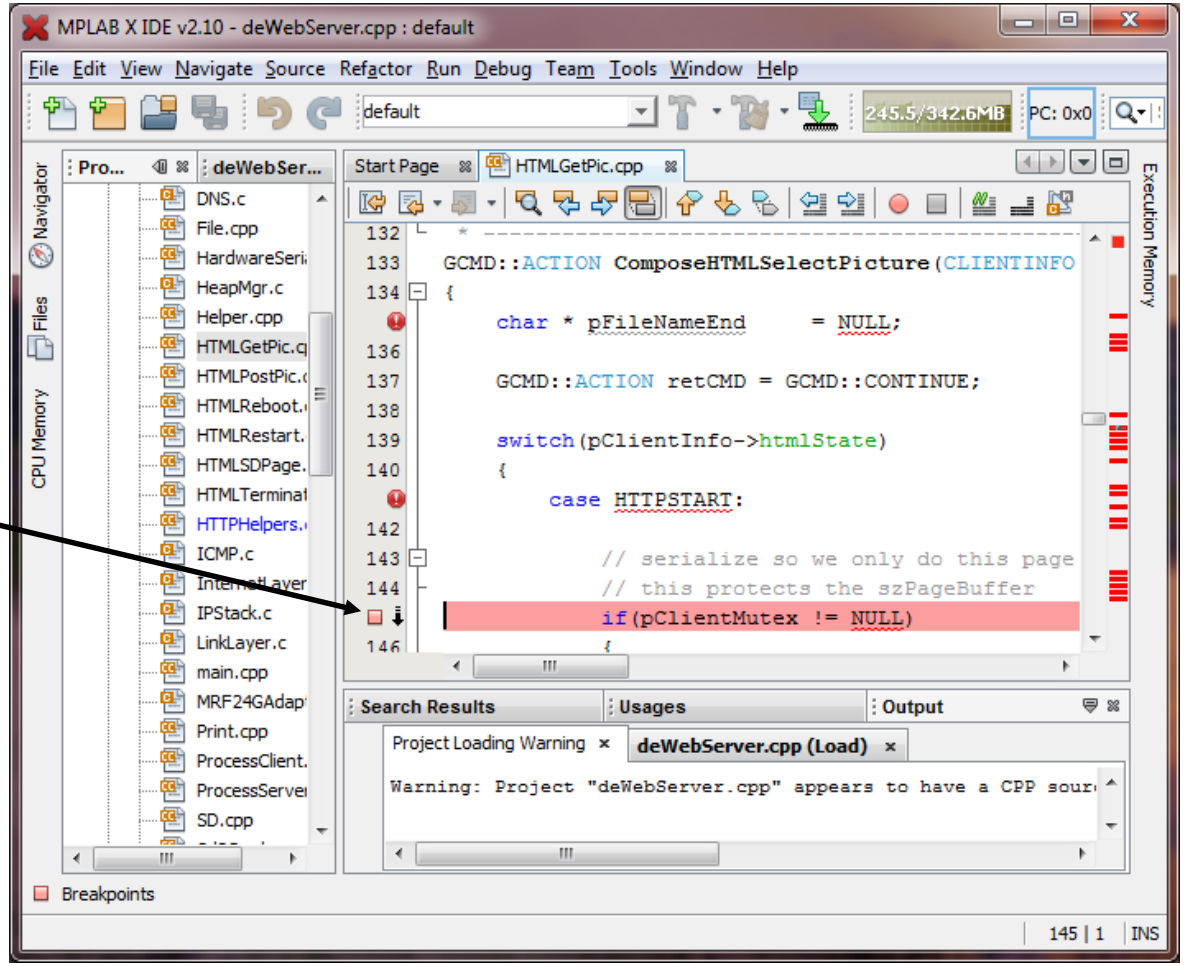
Loadables



Once added you will see the bootloader as a Loadable

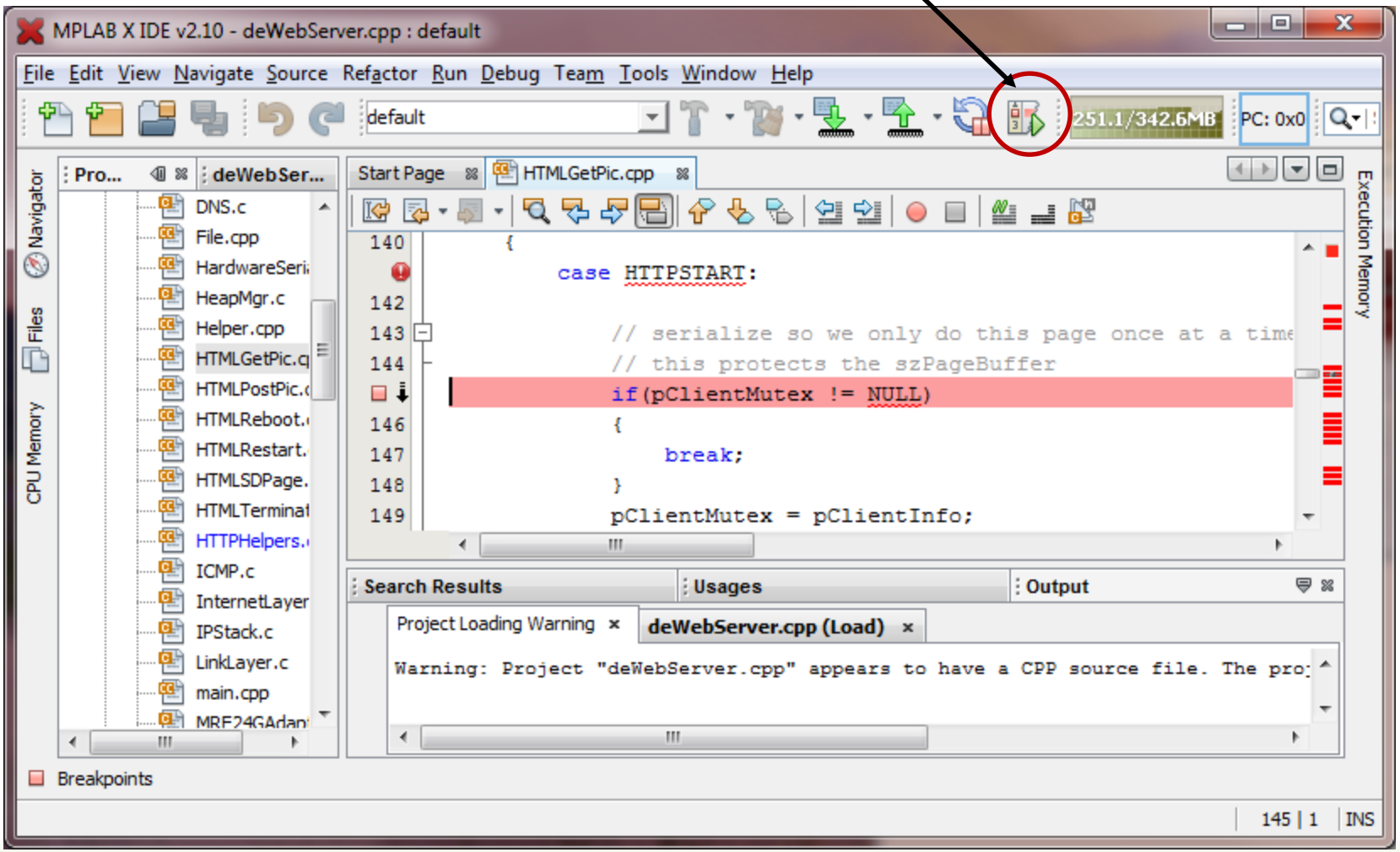
Set a Breakpoint

Left Mouse click on a line number to set a breakpoint



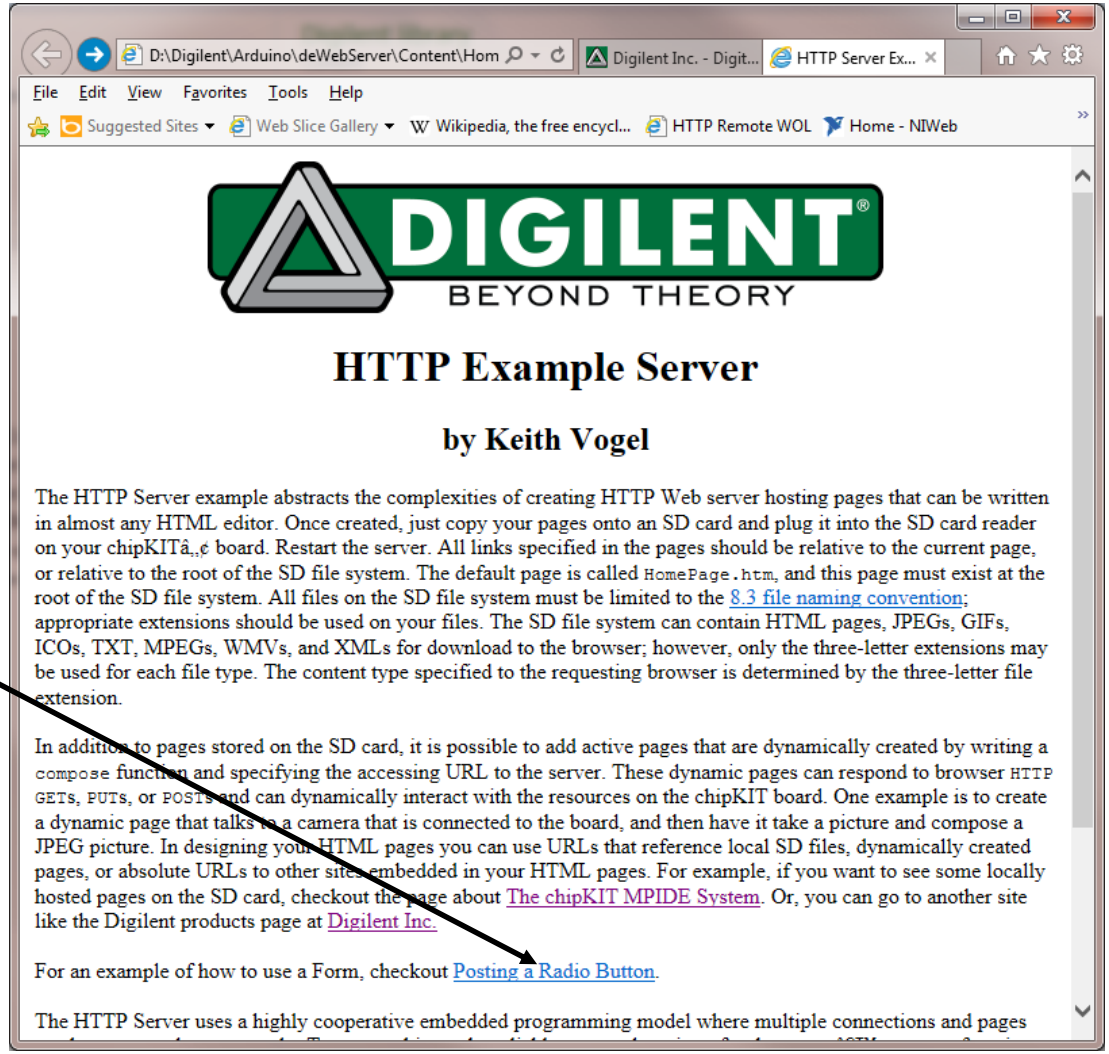
Debug the Sketch

Have MPLAB® X IDE Program sketch for Debugging



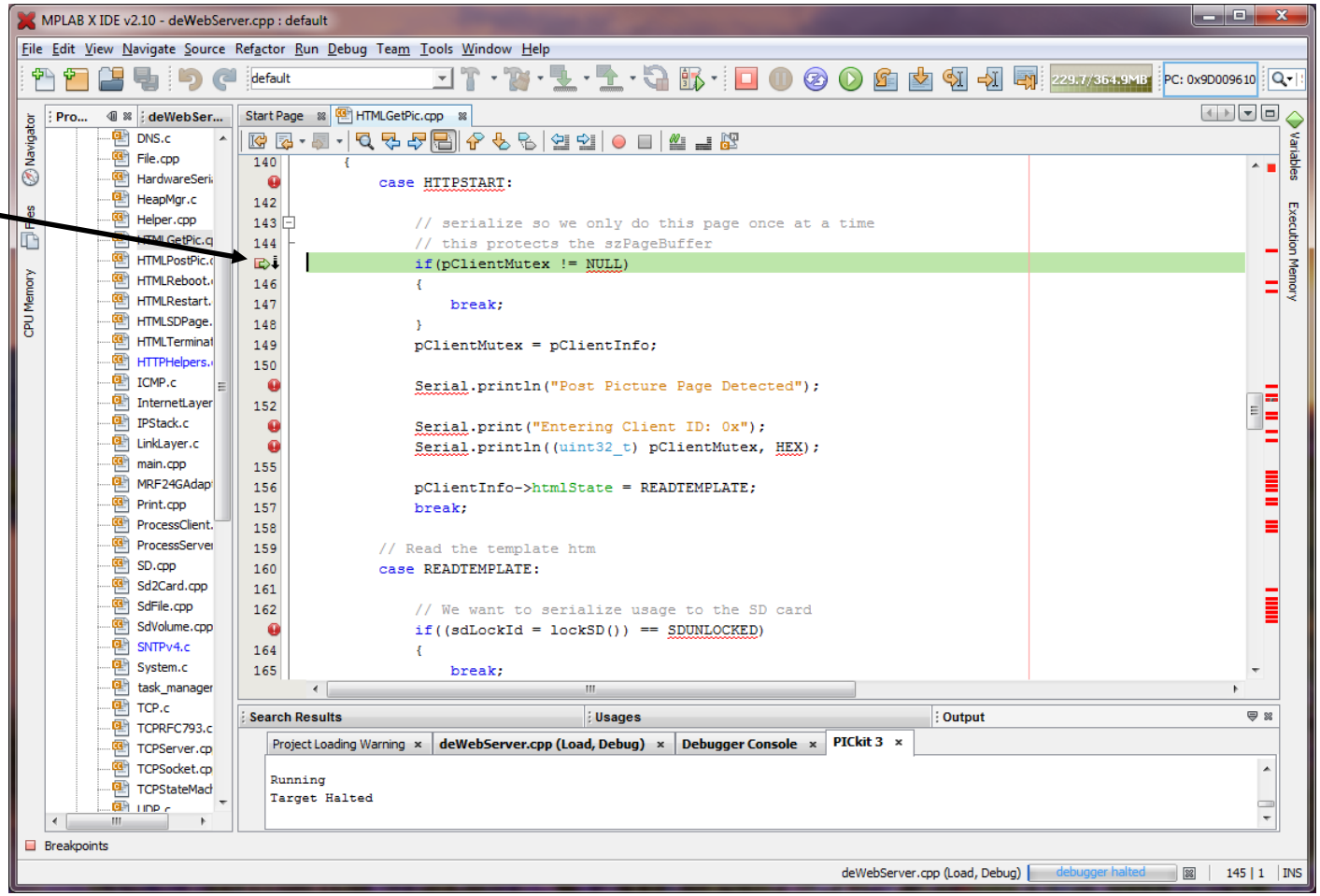
Trigger the Page

From the browser request the URL to execute in the Compose Function with the breakpoint



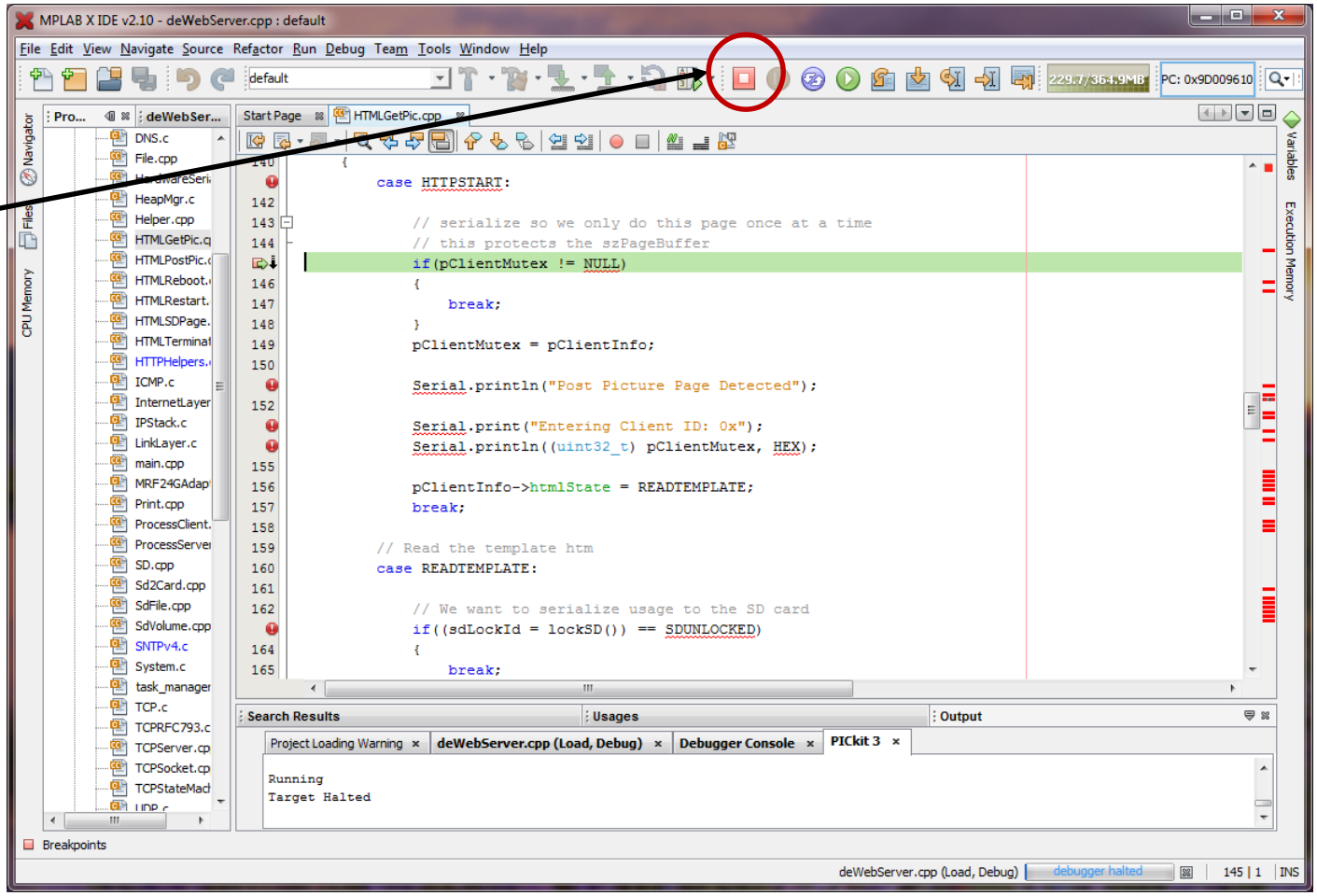
You are Now Debugging

The breakpoint is hit, you are now Debugging



Stop Debugging

Hit the STOP Debugging to stop Debugging



Restore the Bootloader

Do a Release Program to Restore the Bootload AND Program the Sketch

Everything is Restored when the Programming Completes