Embedded Systems

Introduction

Team Emertxe

Embedded Systems
Definition

"Any combination of **Hardware** and **Software**

which is intended to do a

Specific Task

can be called as an **Embedded System**"

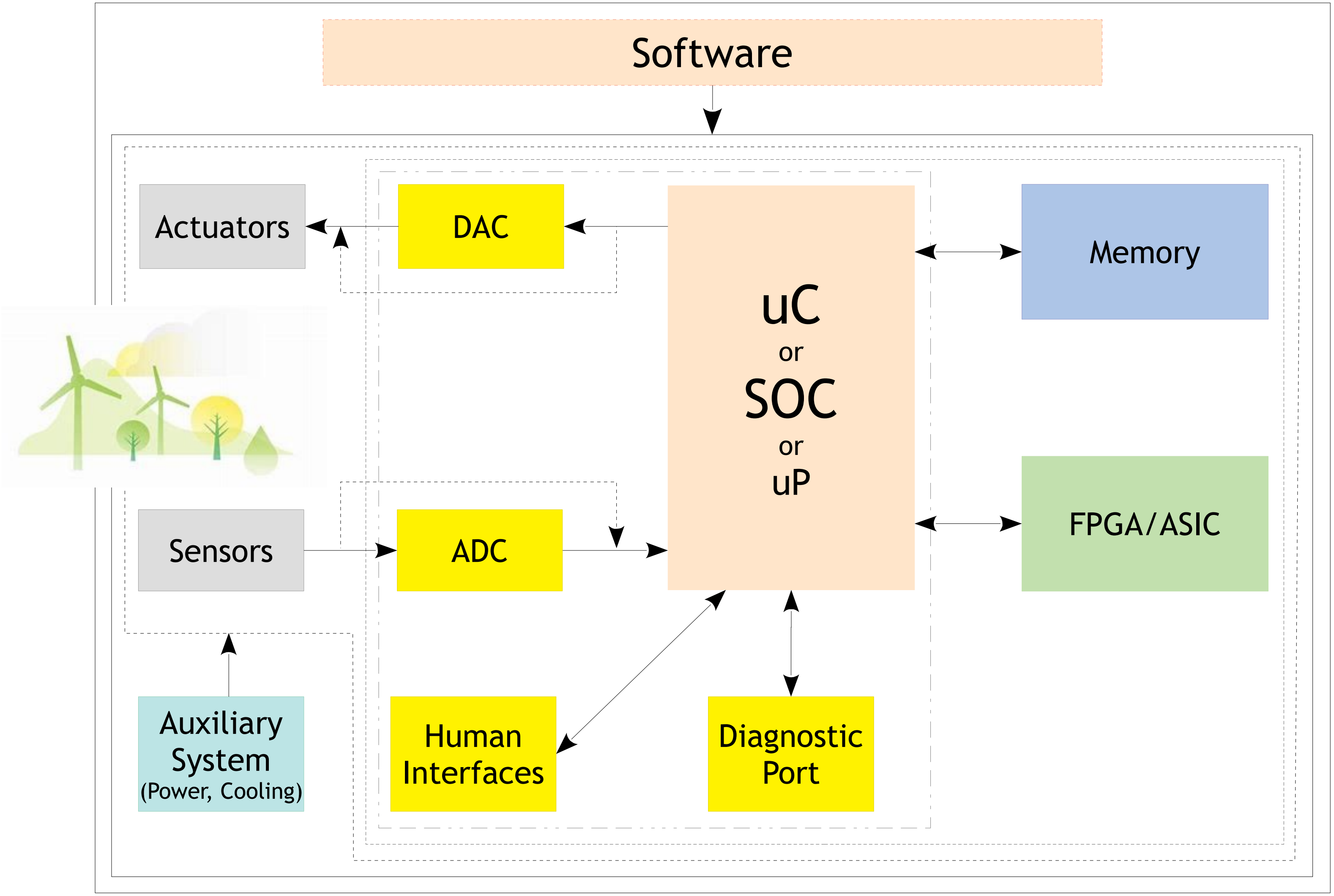
Embedded Systems
Examples

Embedded Systems

Categories

- Stand-alone
- Real Time
- Networked
- Mobile

Embedded Systems
Components



Embedded Systems

Components - Memories - Primary

Type	Volatile?	Writeable?	Erase Size	Max Erase Cycle	Cost per Byte	Speed
SRAM	Yes	Yes	Byte	Unlimited	Expensive	Fast
DRAM	Yes	Yes	Byte	Unlimited	Moderate	Moderate
Masked ROM	No	No	n/a	n/a	Inexpensive	Fast
PROM	No	Once (Ext Prog)	n/a	n/a	Moderate	Fast
EPROM	No	Yes (Ext Prog)	Entire Chip	Limited	Moderate	Fast
EEPROM	No	Yes	Byte	Limited	Expensive	Fast (R) Slow(W/E)
Flash	No	Yes	Sector	Limited	Moderate	Fast (R) Slow(W/E)
NVRAM	No	Yes	Byte	Unlimited	Expensive	Fast

Embedded Systems
Requirements

- Reliability
- Cost-effectiveness
- Low Power Consumption
- Efficient Usage of Processing Power
- Efficient Usage of Memory
- Appropriate Execution Time

Embedded Systems
Challenges

- Efficient Inputs/Outputs
- Embedding an OS
- Code Optimization
- Testing and Debugging

Embedded Systems
Trends in Development

- Processors
- Memory
- Operating Systems
- Programming Languages
- Development Tools

Thank You

Internet of Things (IoT)
Introduction

Team Emertxe

Contents

Internet of Things

Contents

Internet of Things
Watch Intro

Internet of Things
Background

- Collecting information from lots of devices is cool - but it is just telematics.
- Merging perspectives between devices, systems, and humans to build a better understanding of the world around us.
- But tying together insight with action —there lies the promise of IoT.

Internet of Things

Definition

> "The network of physical objects that contain embedded technology to communicate and interact with their internal states or the external environment."
>
> Source: Gartner

Internet of Things

What is it?

- Unique objects connected to Internet
- Devices, not people
- Bi-directional communication
- Large, complex data flows
- New types of insight

Internet of Things

Why is it important?

- Worldwide market for IoT solutions to reach $7.2 trillion in 2020 (IDC)
- Economic value-add is forecast to be $1.9 trillion across sectors in 2020 (Gartner)
- Leading Industry examples :
  utilities, insurance, agriculture, factory, automobiles, transport, consumer, etc

Internet of Things
The End to End Flow

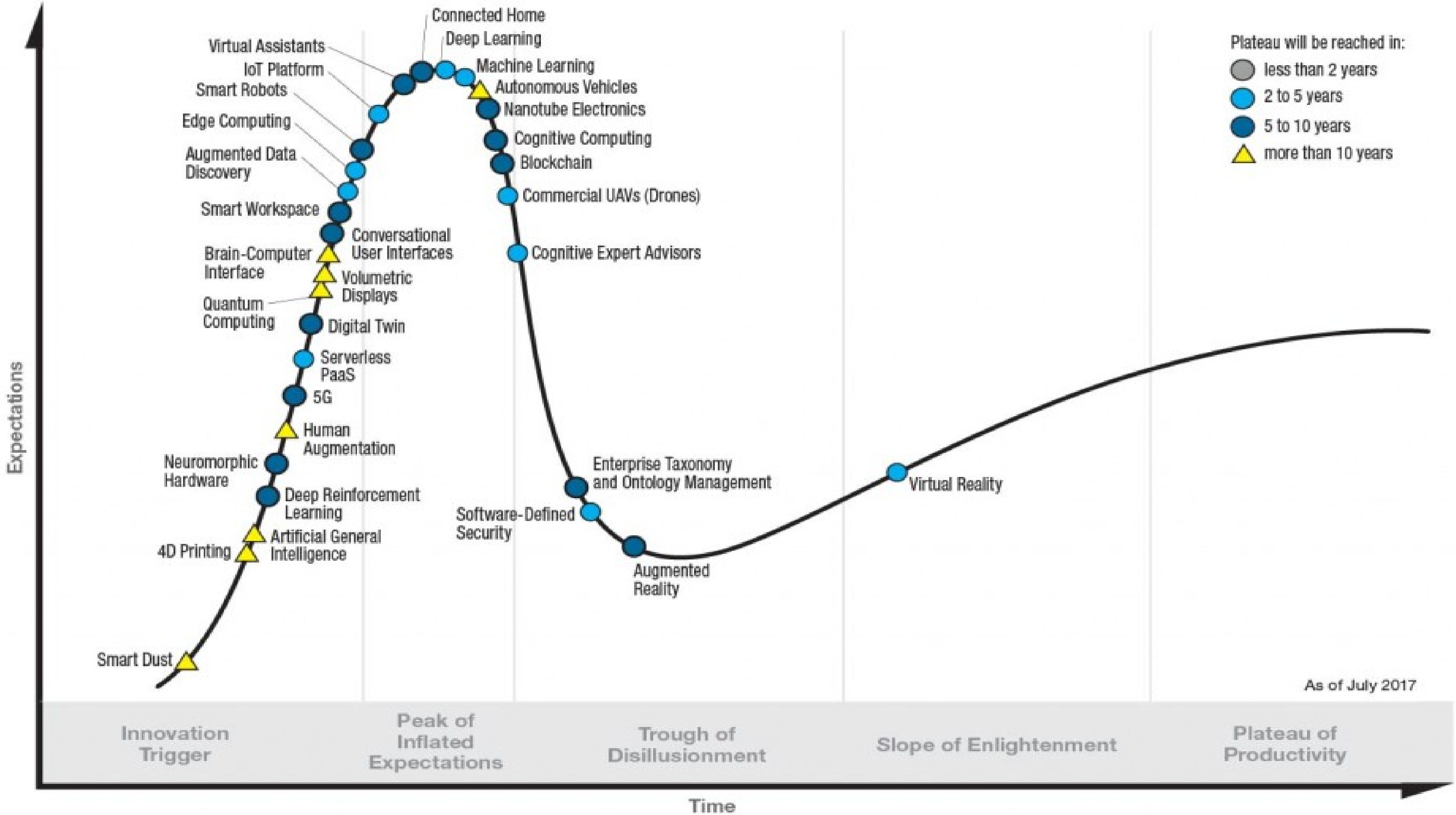
Internet of Things

The End to End Flow

EMERTXE

Internet of Things

The The Gartner Hype Cycle 2017



gartner.com/SmarterWithGartner

Source: Gartner (July 2017)

Gartner

EMERTXE

Internet of Things

Three Trends

Three Trends

Transparently Immersive Experiences

- Human Augmentation
- 4D Printing
- Brain-Computer Interface
- Human Augmentation
- Volumetric Displays
- Affective Computing
- Connected Home
- Nanotube Electronics
- Augmented Reality
- Virtual Reality
- Gesture Control Devices

Perceptual Smart Machine Age

- Smart Dust
- Machine Learning
- Virtual Personal Assistants
- Cognitive Expert Advisors
- Smart Data Discover
- Smart Workspace
- Conversational User Interfaces
- Smart Robots
- Commercial UAVs (Drones)
- Autonomous Vehicles
- Natural-Language Q & A
- Personal Analytics
- Enterprise Taxonomy and Ontology Management
- Data Broker PaaS (dbrPaaS)
- Context Brokering

Platform Revolution

- Neuromorphic Hardware
- Quantum Computing
- Blockchain
- IoT Platform
- Software-Defined Security
- Software-Defined Anything (SDx)

gartner.com/SmarterWithGartner

Source: Gartner

Gartner.

Architectural Overview

Internet of Things

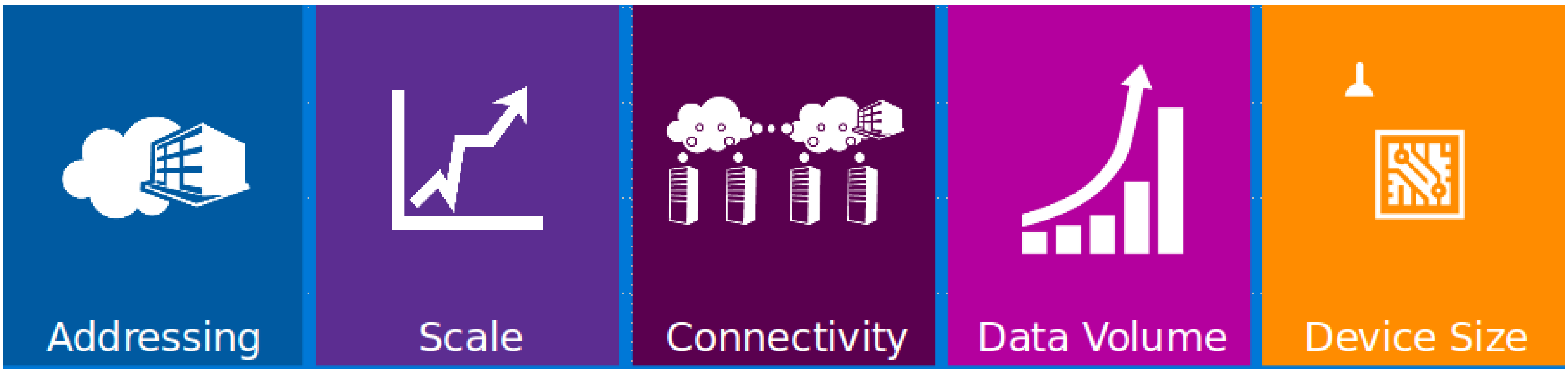
POV: IoT is at an Inflection Point

- Hardware Is getting cheap
- M2M solutions are mainstream
- Connectivity is proliferating
- Software is more advanced
- Cloud cost, scale, flexibility

Internet of Things

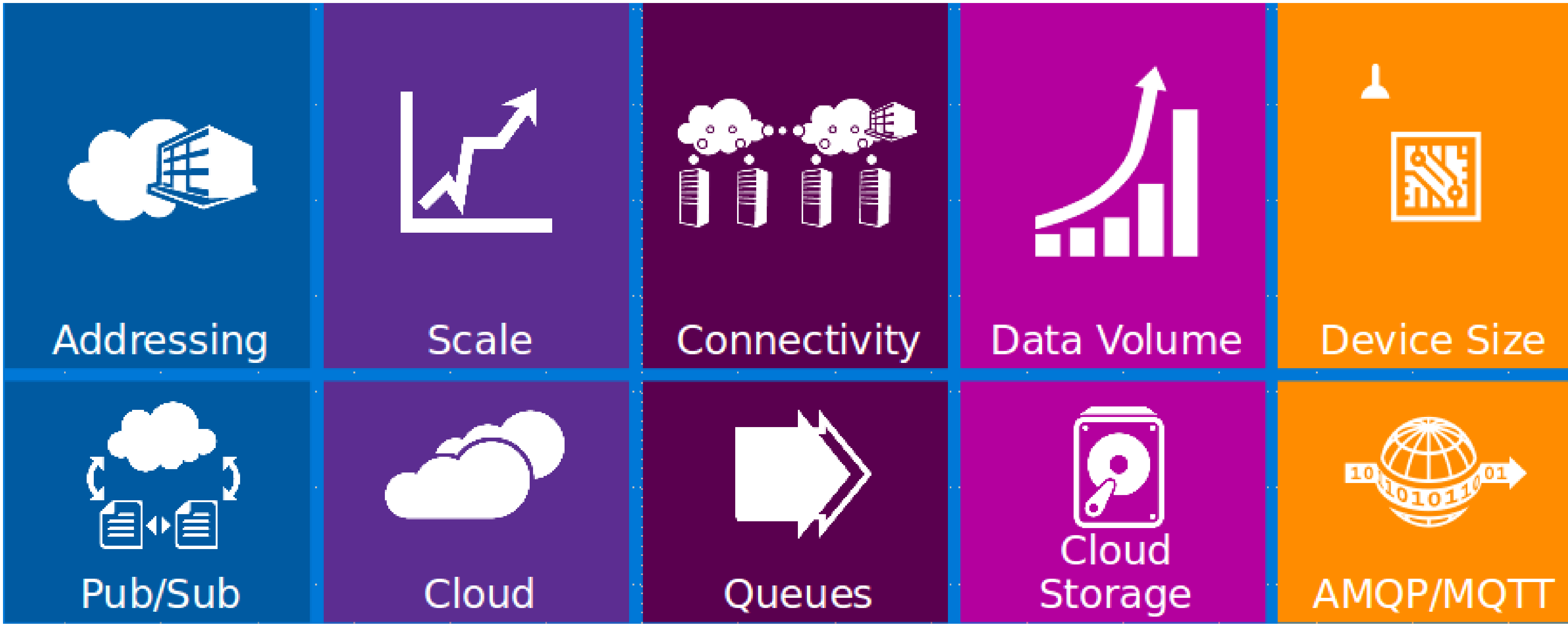
General Technical Requirements

Internet of Things
Challenges

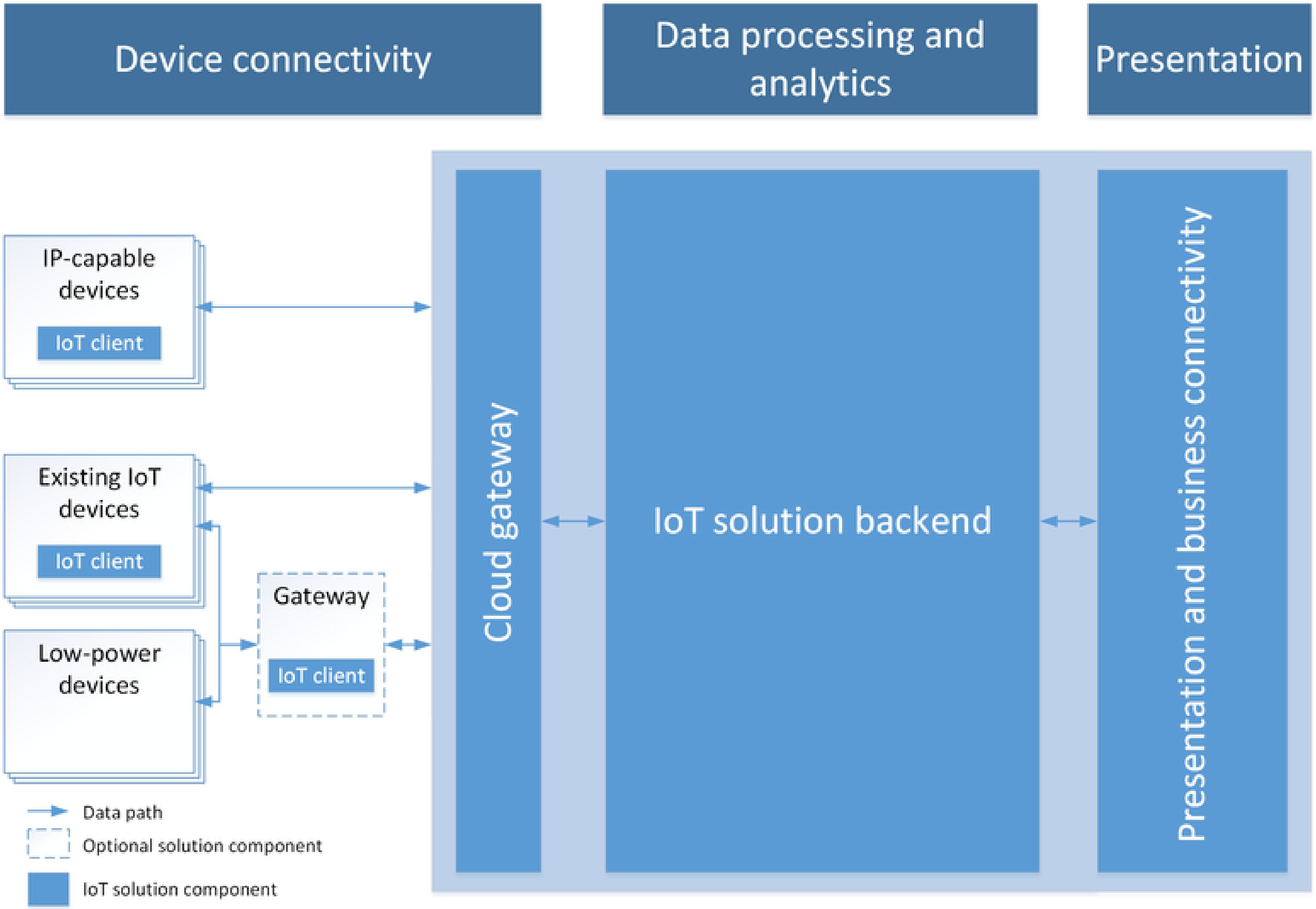


Internet of Things

First Principle



Internet of Things
Reference Architecture

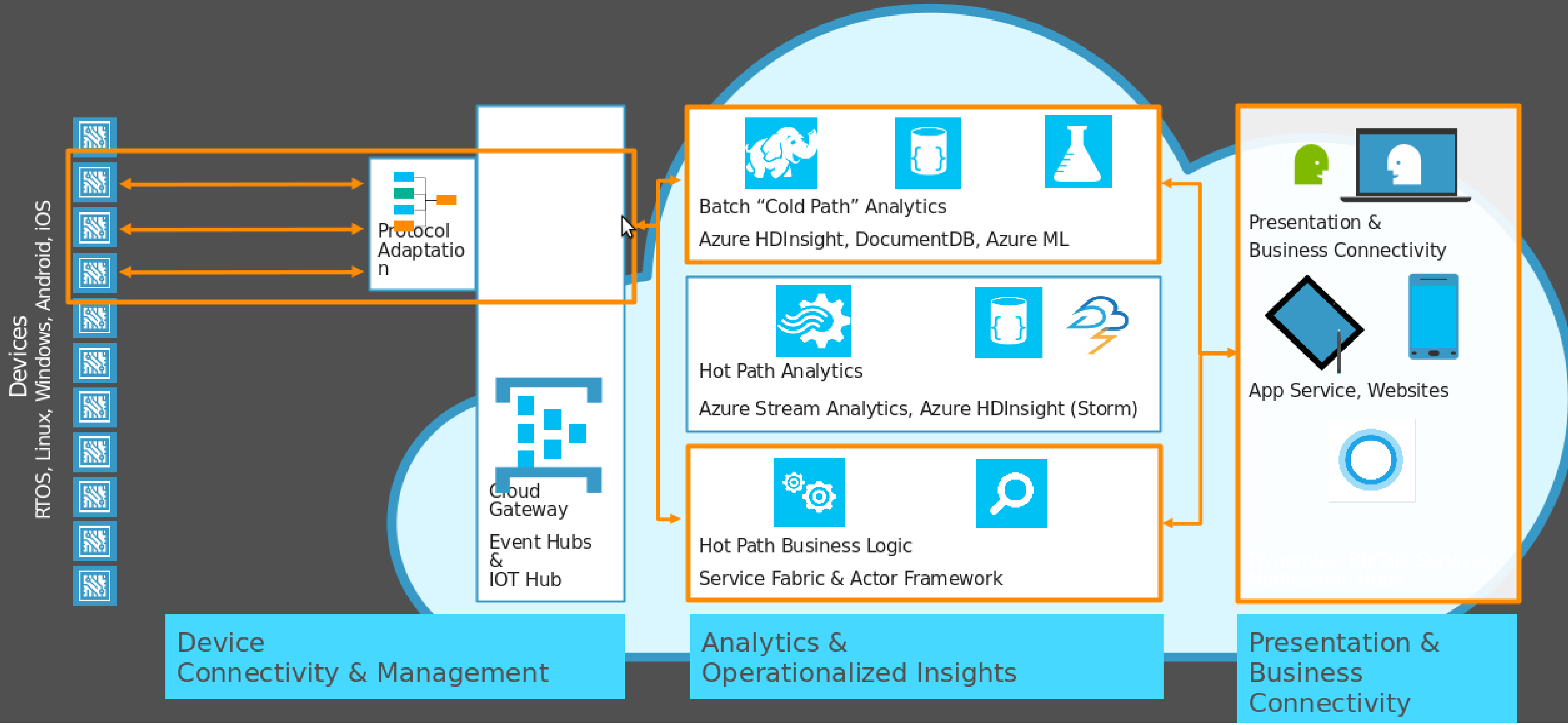


Internet of Things

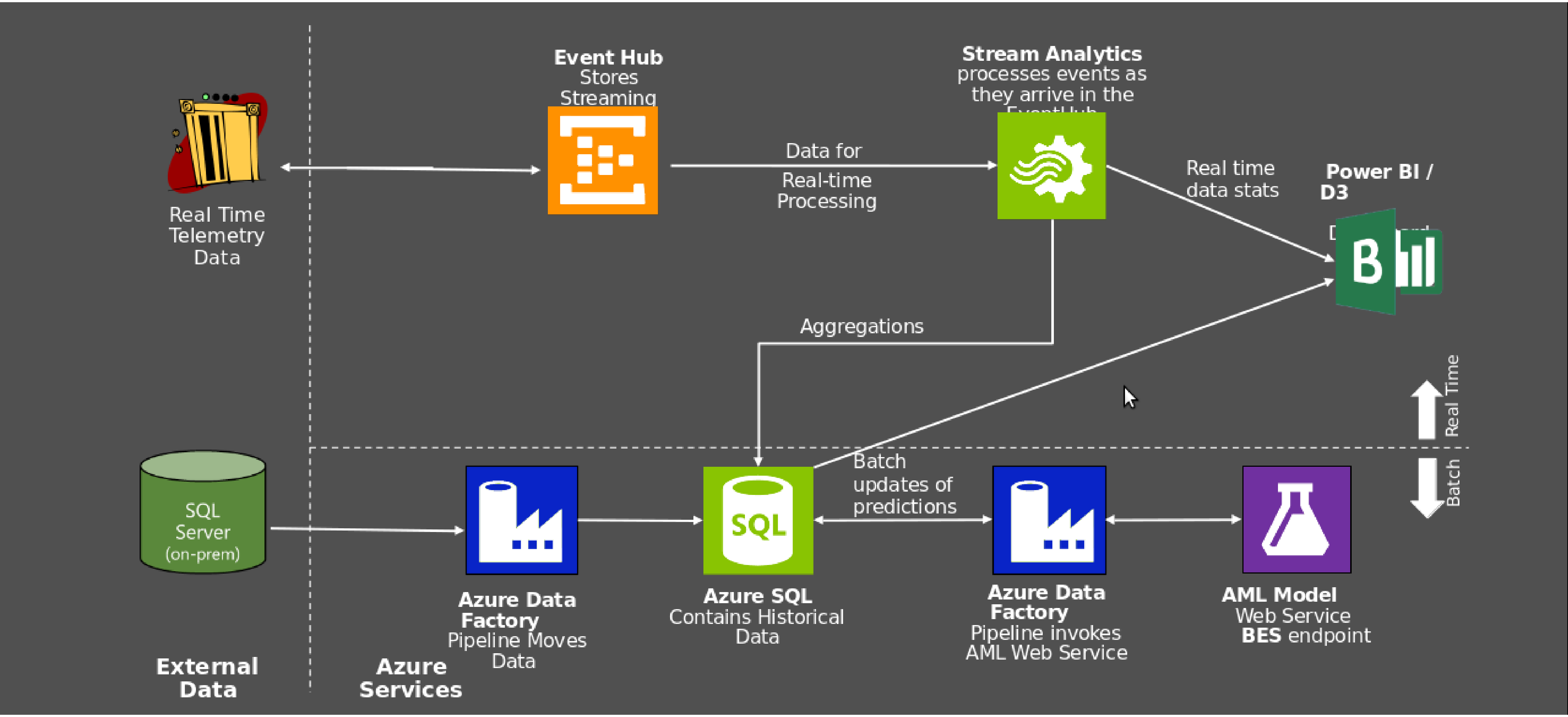
Microsoft Azure IoT Services

Producers	Data Transport	Storage	Analysis	Presentation & action
	Event Hubs (Service Bus)	SQL Database	Machine Learning	Azure Websites
	Heterogeneous client agents	Table/Blob Storage	HD Insight/Storm	Mobile Services
	External Data Sources	DocumentDB	Stream Analytics	Notification Hubs
		External Data Sources	Cloud Services	Power BI
				External Services

Pattern: Predictive Maintenance



Internet of Things
Example Architecture



Internet of Things
Risks

- Old ways of Thinking can be dangerous
- Understand the business model
- Beware of new patterns: eventual consistency, etc.
- Don’t focus on the device
- Avoid analysis paralysis. Get it done!

Internet of Things
Architecture: Summary

- Architecture is at the center of IoT
- IoT is Advanced “Modern” Architecture
- IoT Projects are Complex - Teamwork is necessary
- These projects are mission critical and difficult
- We can’t learn everything - but we need breadth
- Don’t be afraid - get started and learn

Thank You

Devices

Generally known as Things

Team Emertxe

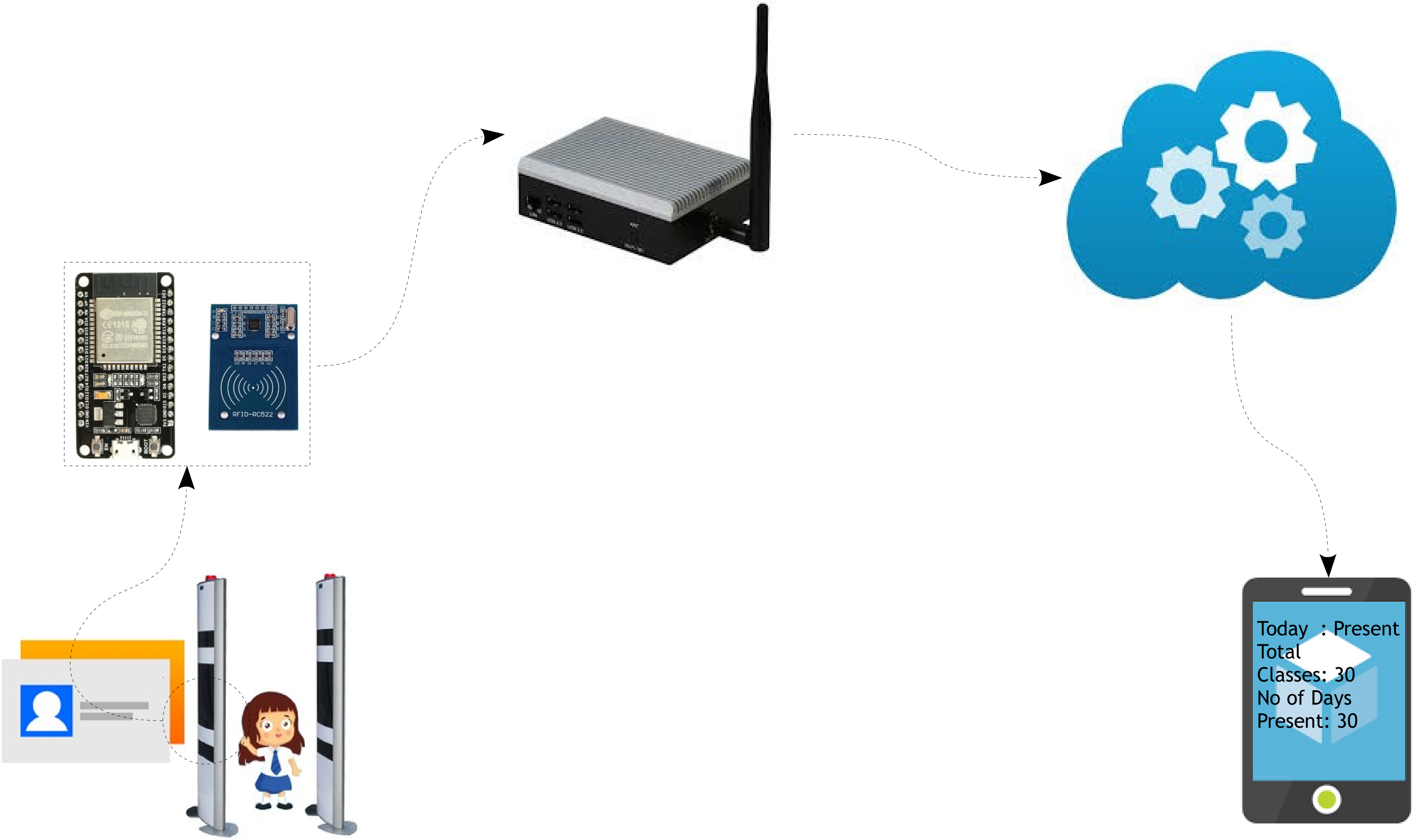
Devices

What is this Module about??

Well lets see the the data generally flows

Devices

The Data Flow

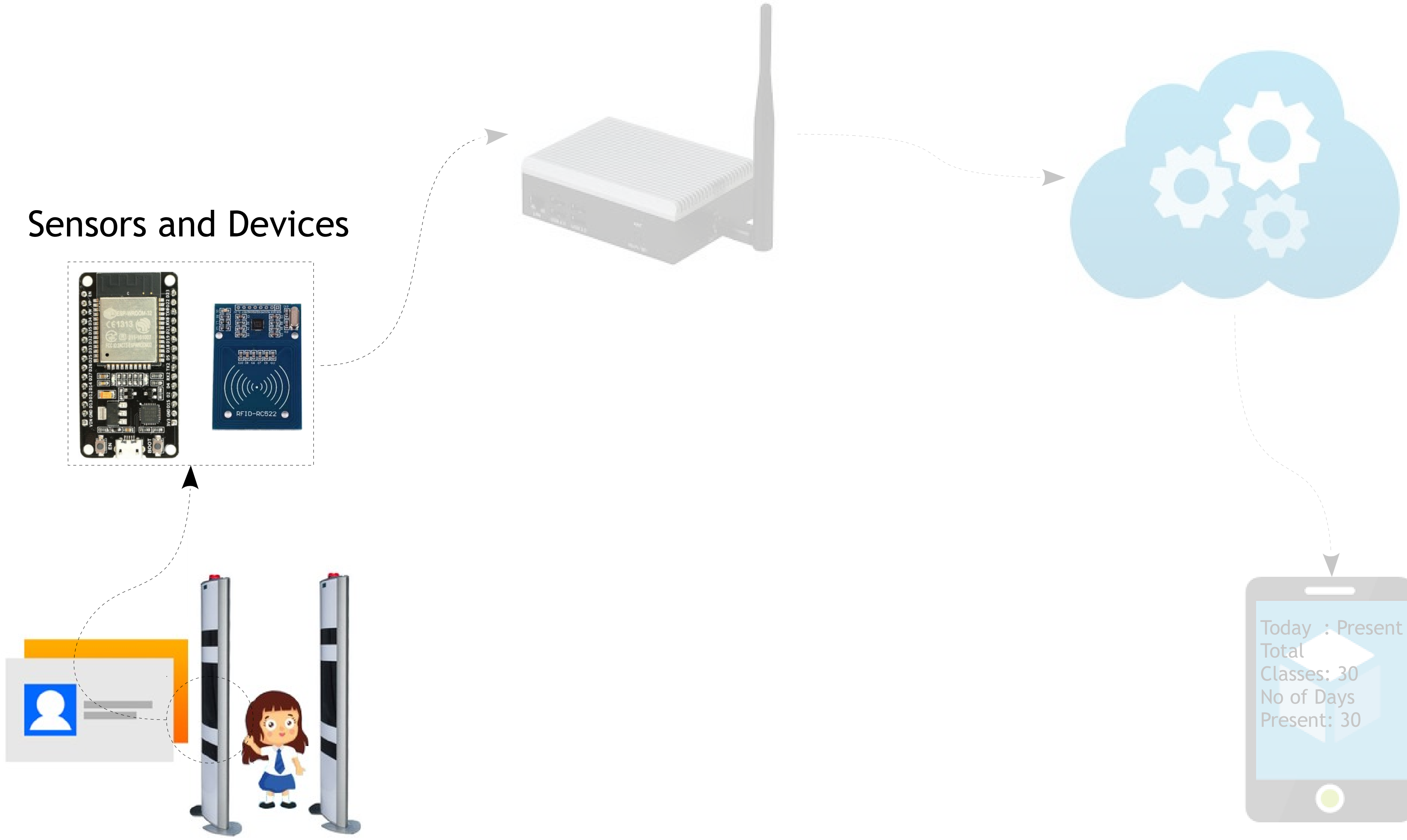


Devices

The Data Flow

- From the previous slide we can see that, the data flows through different layers and every layer is important.
- And it is obvious that, the origin of data is very important which is collected and send to data analysis
- So in this module we concentrate of the Sensor and The Device Part as shown in the next slide.

Devices
The Data Flow



Devices

Introduction - What is it?

A thing made or adapted for a particular purpose, especially a piece of mechanical or electronic equipment.

Source: Google

Thank You

Arduino

Programming Things

Team Emertxe

Introduction

Arduino
Introduction – What?

An open-source electronics platform based on
easy-to-use hardware and software

Source: www.arduino.cc

Arduino

Introduction – Why?

- Inexpensive
- Cross-platform
- Simple, clear programming environment
- Open source and extensible software
- Open source and extensible hardware

Arduino

Introduction – How do I use?

- Code online on the Arduino Web Editor
  - To use the online IDE simply follow these instructions. Remember that boards work out-of-the-box on the Web Editor, no need to install anything.
- Install the Arduino Desktop IDE

Arduino

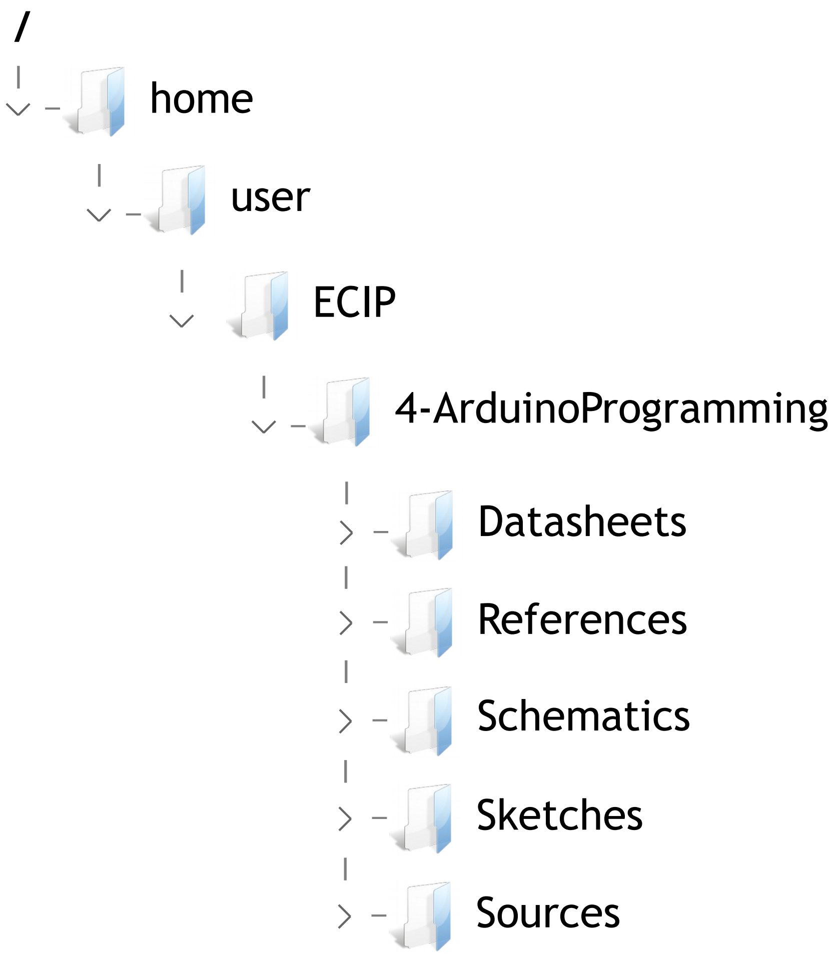
Introduction – How do I use?

- Install the Arduino Desktop IDE
  - To get step-by-step instructions select one of the following link accordingly to your operating system.
    - Windows
    - Mac OS X
    - Linux
    - Portable IDE (Windows and Linux)

Setup

Arduino

Setup – Workspace Creation



Open your favorite terminal and run the following commands

```
user@user:~] cd # Make sure you are in home directory
user@user:~] pwd
/home/user
user@user:~] mkdir -p ECIP/4-ArduinoProgramming
user@user:~] cd ECIP/4-ArduinoProgramming
user@user:4-ArduinoProgramming]
user@user:4-ArduinoProgramming]  mkdir Datasheets
user@user:4-ArduinoProgramming]  mkdir References
user@user:4-ArduinoProgramming]  mkdir Schematics
user@user:4-ArduinoProgramming]  mkdir Sketches
user@user:4-ArduinoProgramming]  mkdir Sources
user@user:4-ArduinoProgramming]  ls
Datasheets  References  Schematics  Sketches  Sources
user@user:4-ArduinoProgramming]
```

Arduino

Setup – Download

- Click the below icon and download the latest version of IDE, Make sure you select the correct Linux Version based on your system

- Assuming you have downloaded the file in the Download directory, proceed with the installation steps mentioned in the next slide

Arduino

Setup – Installation

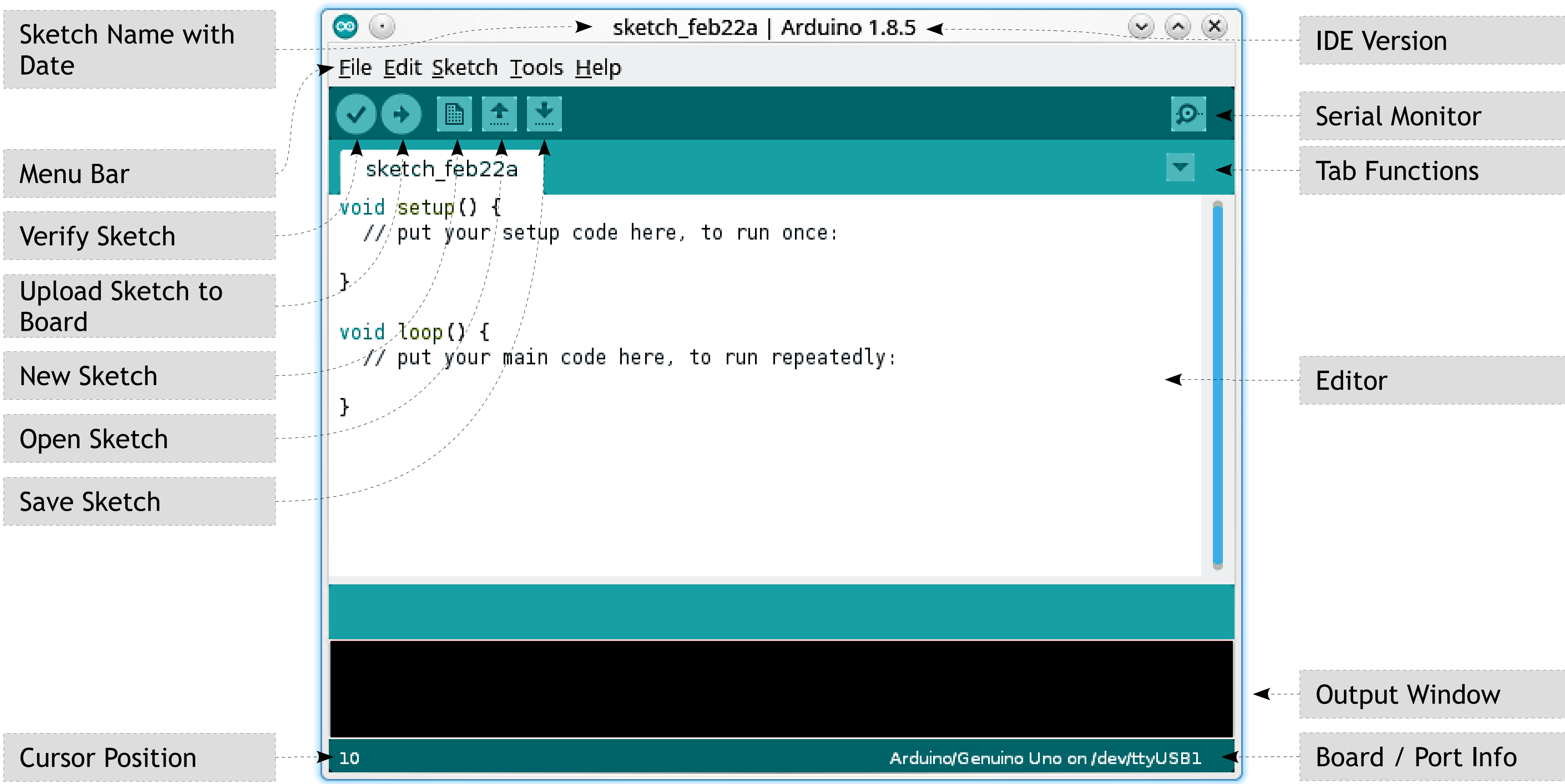
```
user@user:4-ArduinoProgramming] ls
Datasheets  References  Schematics  Sketches  Sources
user@user:4-ArduinoProgramming] cd Sources
user@user:Sources] mv ~/Downloads/arduino-*.tar.xz .
user@user:Sources] tar xvf arduino-*.tar.xz
user@user:Sources] cd arduino-*
user@user:arduino-<version>] chmod +x install.sh
user@user:arduino-<version>] ./install.sh
Adding desktop shortcut, menu item and file associations for Arduino IDE... done!
user@user:arduino-<version>]
```

- In case if you want to uninstall!, you may follow the below step

```
user@user:arduino-<version>] chmod +x uninstall.sh
user@user:arduino-<version>] ./uninstall.sh
Removing desktop shortcut and menu item for Arduino IDE... done!
user@user:arduino-<version>]
```

IDE Overview

Arduino
IDE



Sketch Name with Date
Menu Bar
Verify Sketch
Upload Sketch to Board
New Sketch
Open Sketch
Save Sketch
Cursor Position
sketch_feb22a | Arduino 1.8.5
File Edit Sketch Tools Help
sketch_feb22a
void setup() {
// put your setup code here, to run once:
}
void loop() {
// put your main code here, to run repeatedly:
}
10
Arduino/Genuino Uno on /dev/ttyUSB1
IDE Version
Serial Monitor
Tab Functions
Editor
Output Window
Board / Port Info

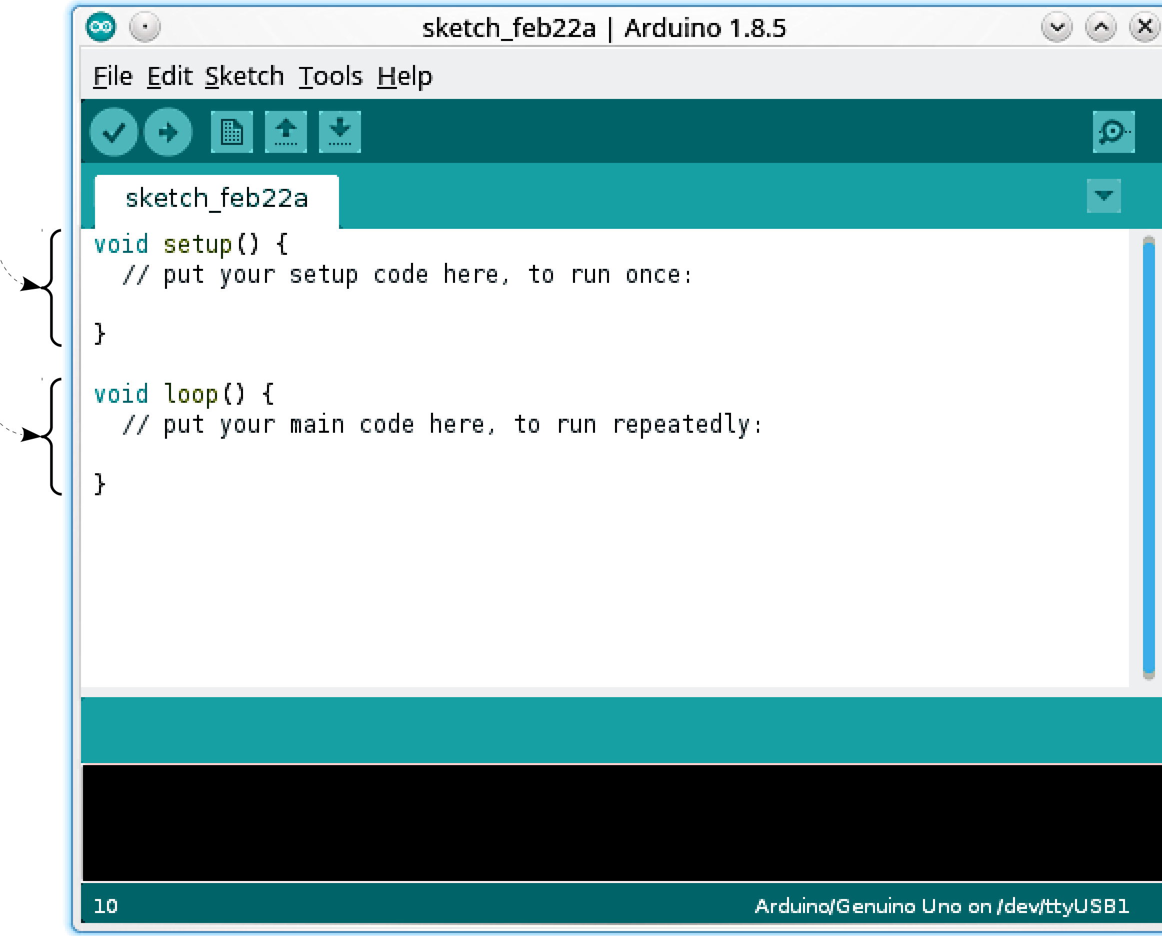
Arduino

Sketch - Default

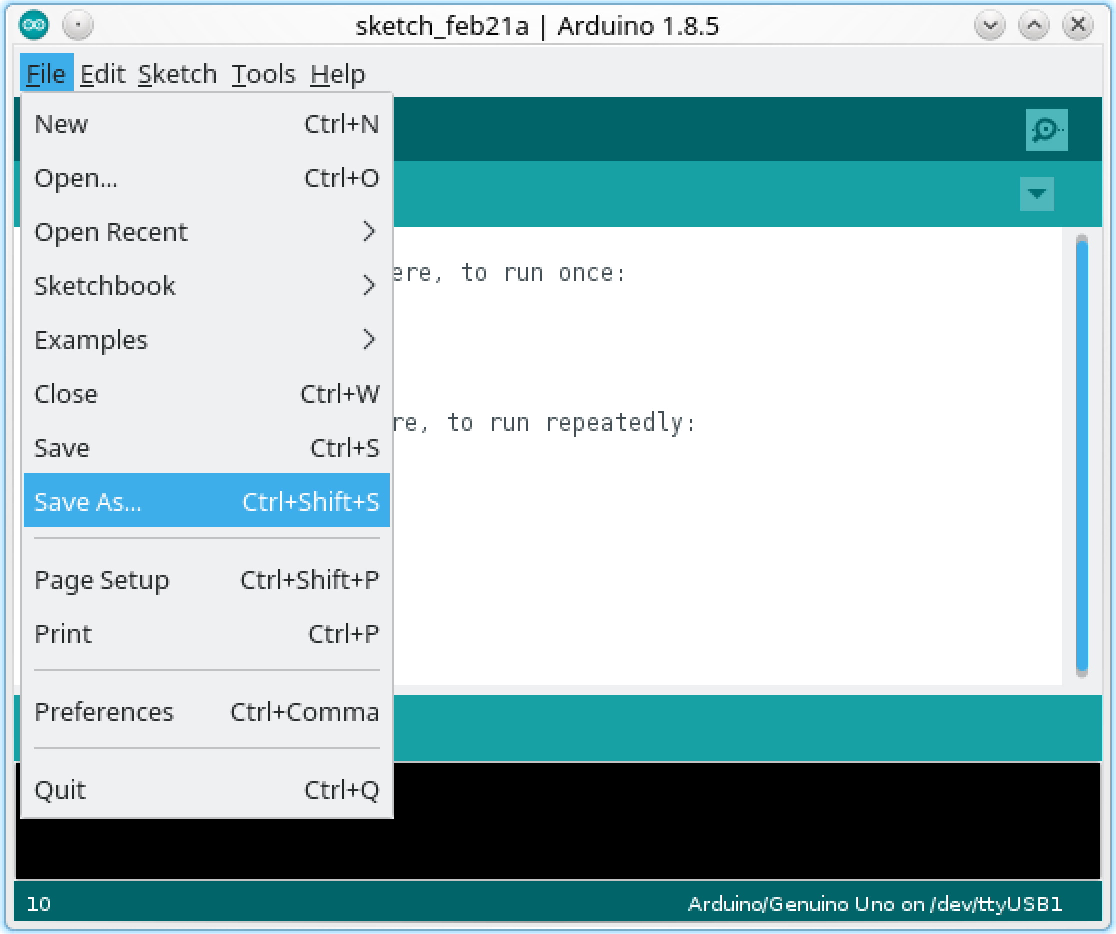
All one time initialization goes here. For example,

- Configuration of DDR register
- Serial port setup etc.,

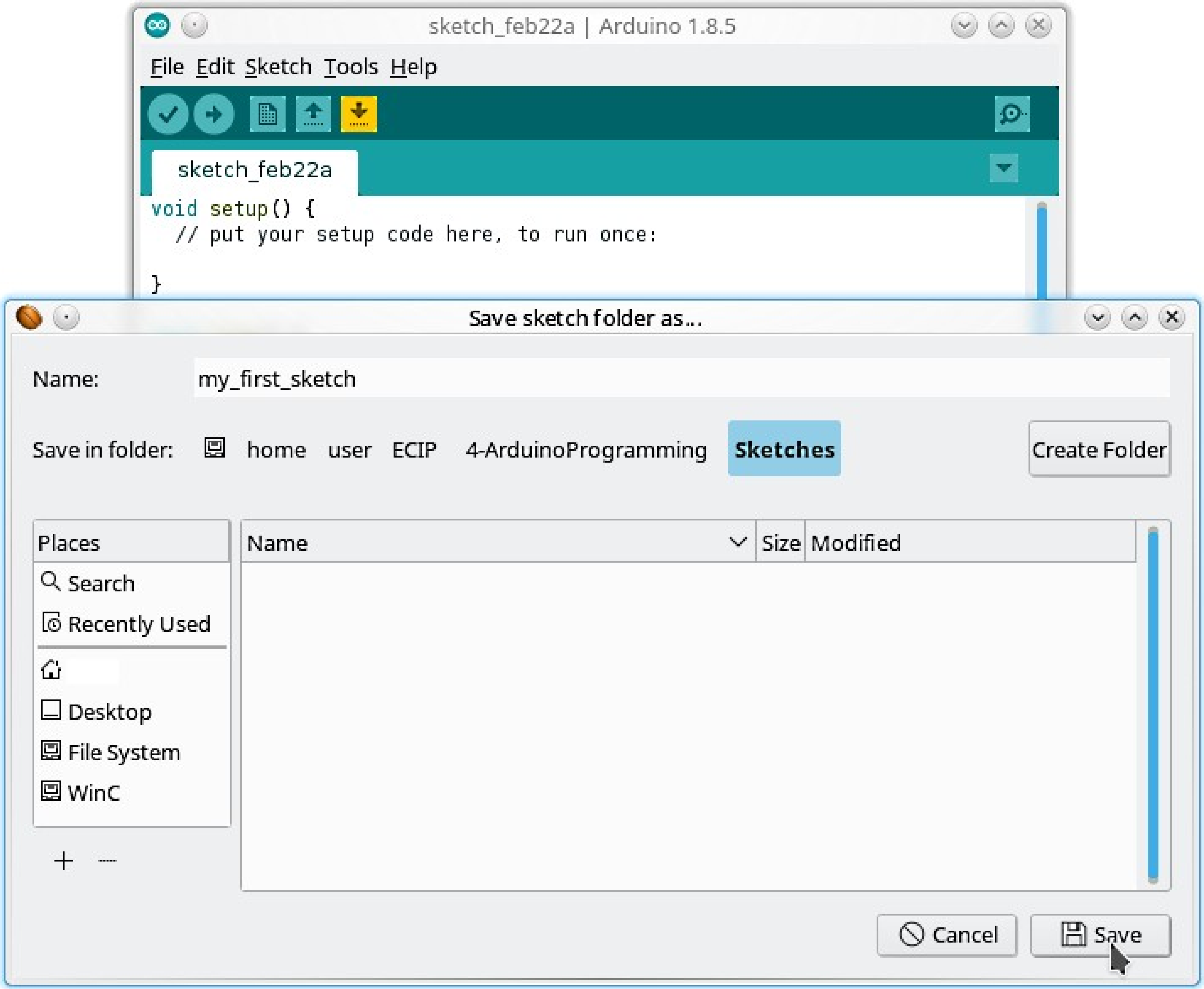
The application code, which should loop forever should be put here



Arduino
Sketch – Save As

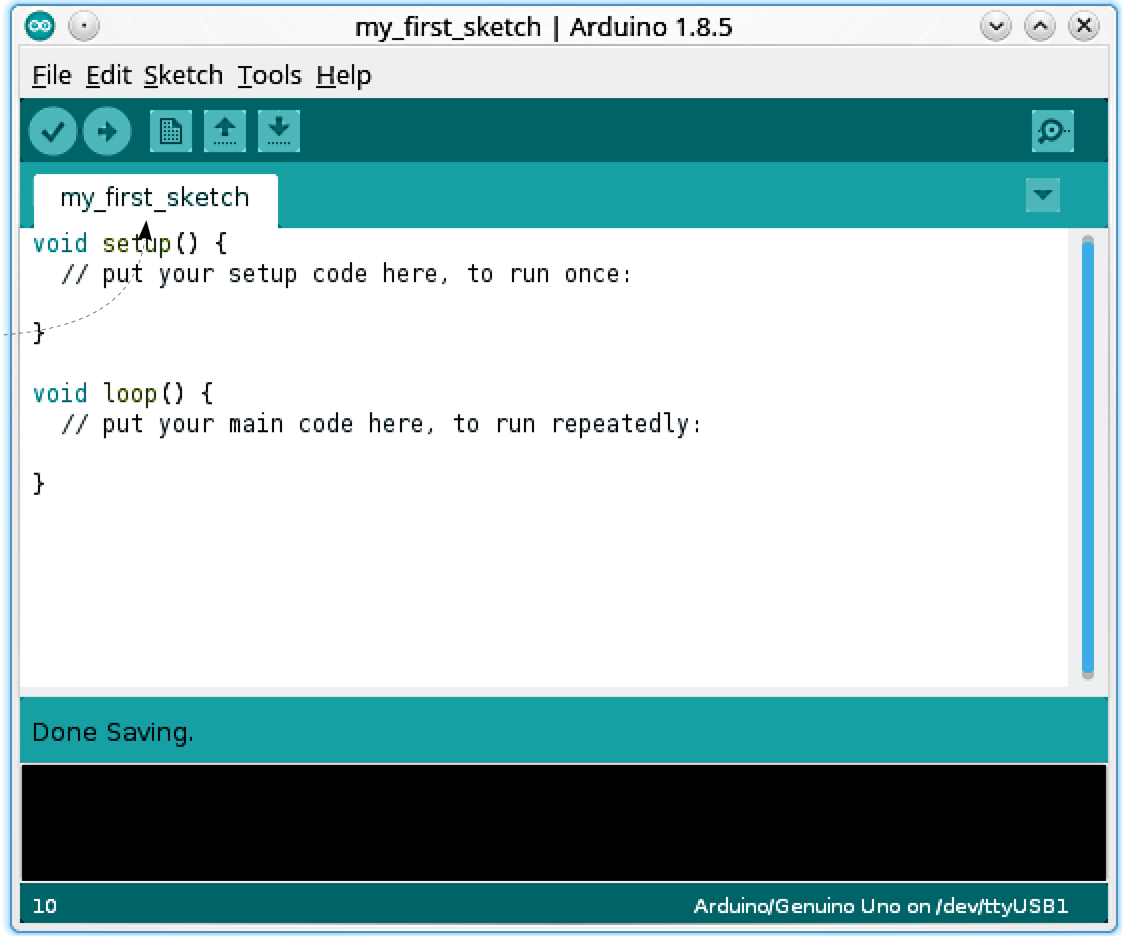


Arduino
Sketch - Save As



Arduino
Sketch – Save As

Saved Sketch as



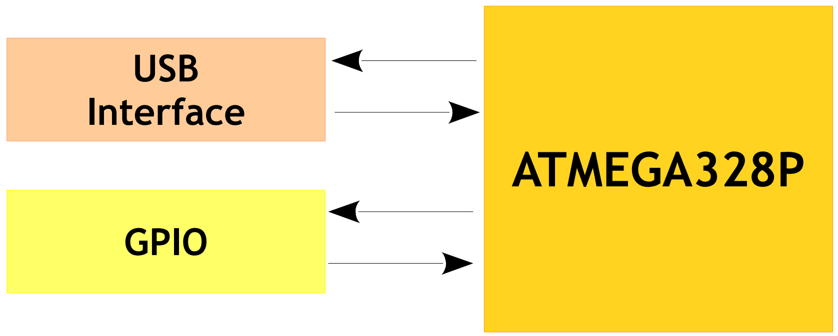
Board Architecture

Arduino
Hardware Architecture

- There are different varieties of boards, modules and shields available
- Can be used for different complexity levels like basic sensor node with non OS firmware to an IoT gateway based on embedded Linux
- Few types of boards and its architectures are mentioned in next slides

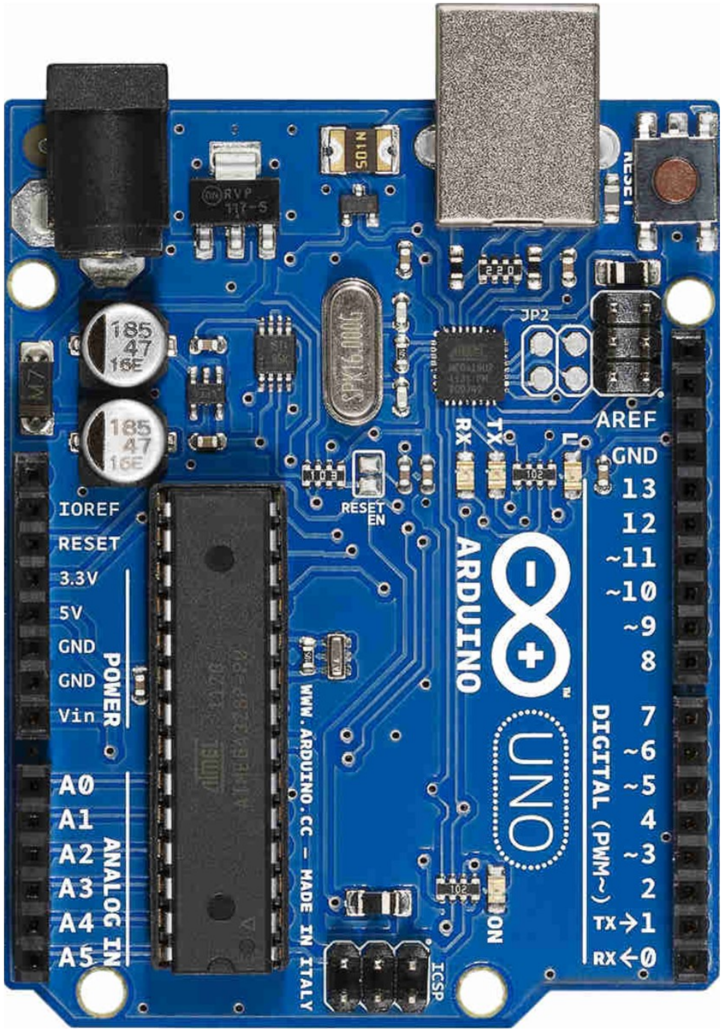
Arduino

Hardware Architecture - UNO



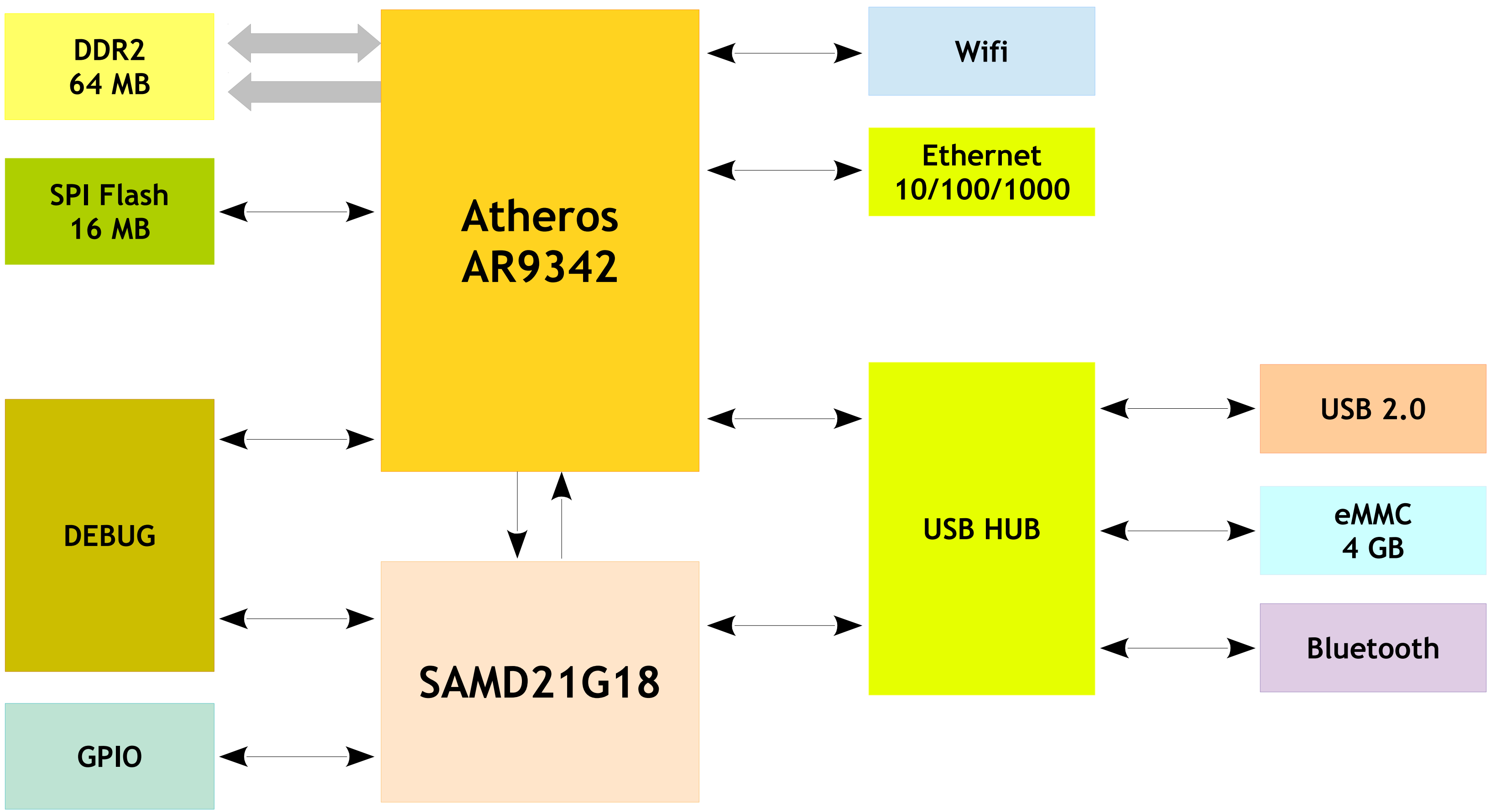
Arduino

Hardware Architecture – UNO - Board



Arduino

Hardware Architecture - TIAN

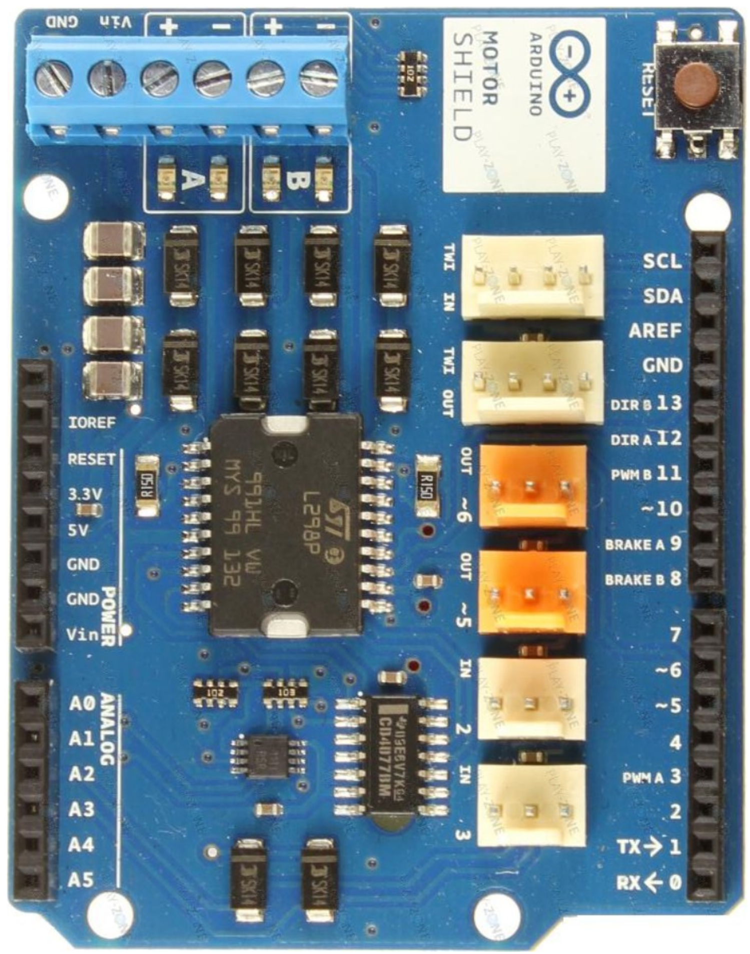


Arduino

Hardware Architecture – TIAN - Board

Arduino

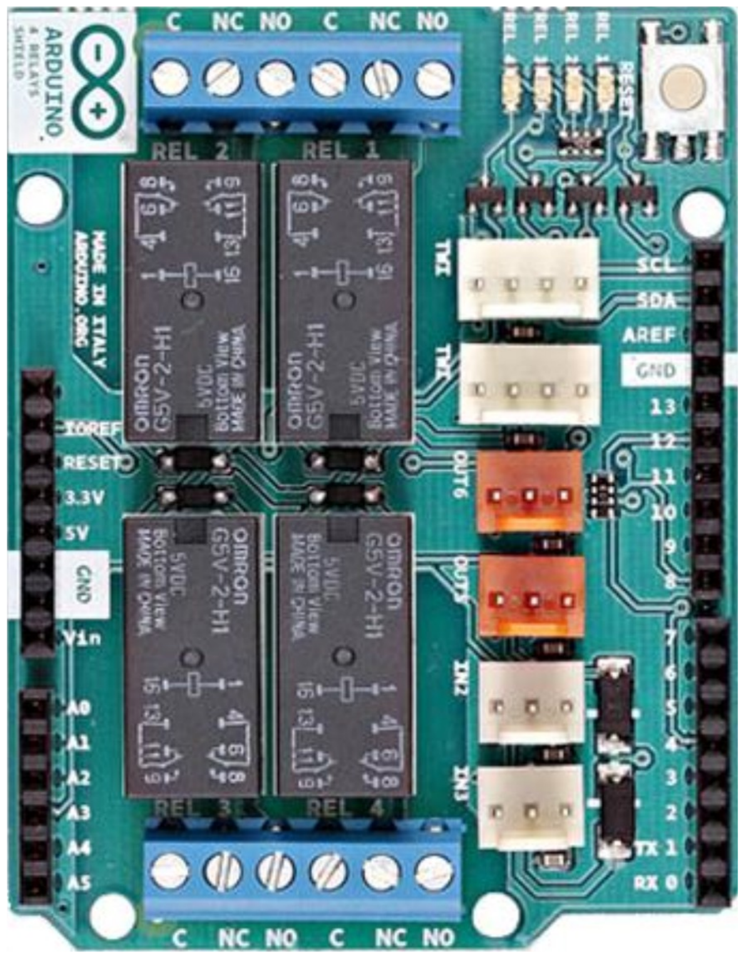
Hardware Architecture – Shield - Motor



ΣMERTXE

Arduino

Hardware Architecture – Shield - Relay



Arduino

Hardware Architecture

- So as summary we lots of open source hardware option to pick upon
- As part this module, we would be concentrating on NodeMCU, based on ESP8266 Wi-Fi Module

The First Sketch

Arduino

The First Sketch

- Well, as general approach, write that first code (irrespective of the hardware you work on), which gives you the confidence that you are on the right path.
- So, identify the simplest possible interface which can be made to work with lesser overhead, which helps us to verify that our,
  - Hardware is working
  - Toolchain setup is working
  - Connectivity between the host and target is established

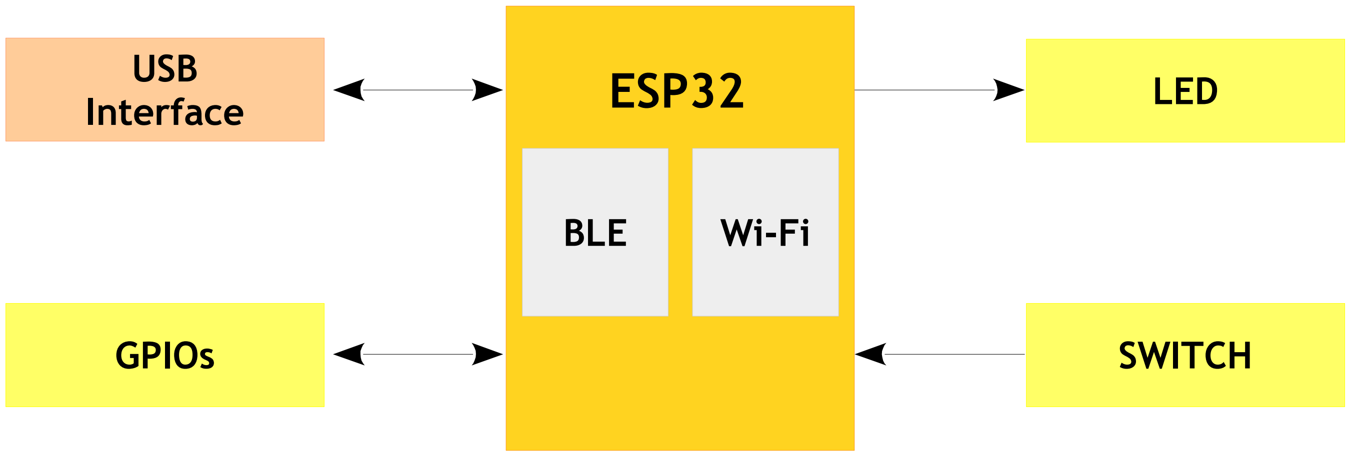
  and so on.

Arduino
The First Sketch

- It is good to know, what your target board is?, what it contains? by its architecture
- Board architecture generally gives you overview about your board and its peripheral interfaces
- In our case, as already mentioned we will be using NodeMCU whose architecture is given in the next slide

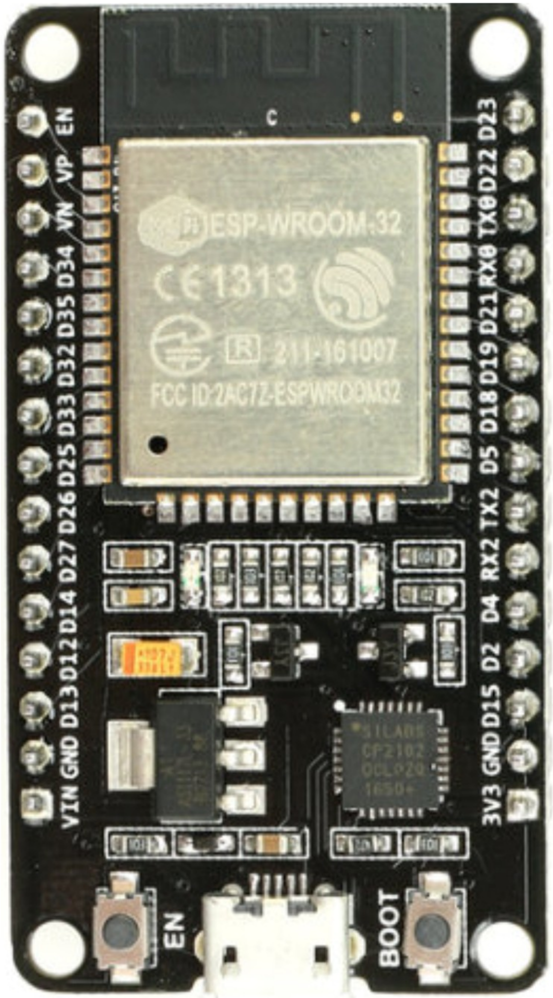
Arduino

The First Sketch – NodeMCU - Architecture

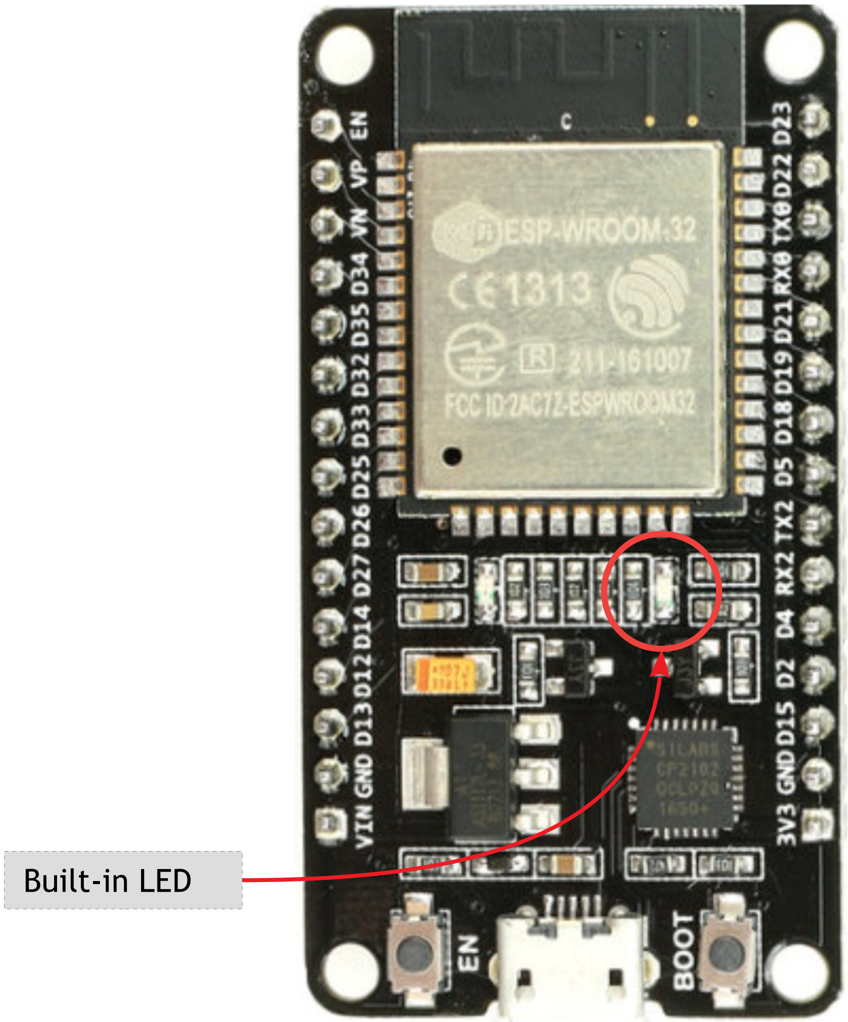


Arduino

The First Sketch – NodeMCU – Module



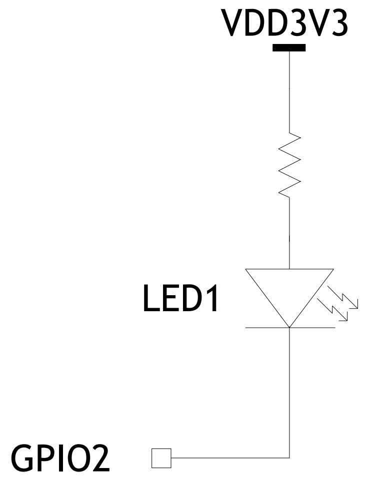
Arduino
The First Sketch



- From the NodeMCU’s architecture, we come to know about a built-in LED!, so why not start with it?
- Well, if you have a bit of microcontroller programming experience, you would certainly ask a question on where and how the LED is connected?
- The board schematic has this answer.

Arduino

The First Sketch – NodeMCU – Schematic (Part)



- The LED is connected to GPIO2
- Its a sinking circuit (0 to glow)
- With these basic information, it should possible to write our first sketch
- Please refer the next slide to proceed further

Arduino

The First Sketch – I/O Configuration

- Almost all the modern controllers have multiple mode on a port pin by design
- We need to set the right mode before we can write our application!
- The LED is connected at **GPIO2** which has to be set as **Output Pin**
- Would like to recall that, The Arduino platform is very popular because of its rich library functions, which make it easy to program embedded devices
- So we need the right set of libraries configured in our IDE for the target board we are using

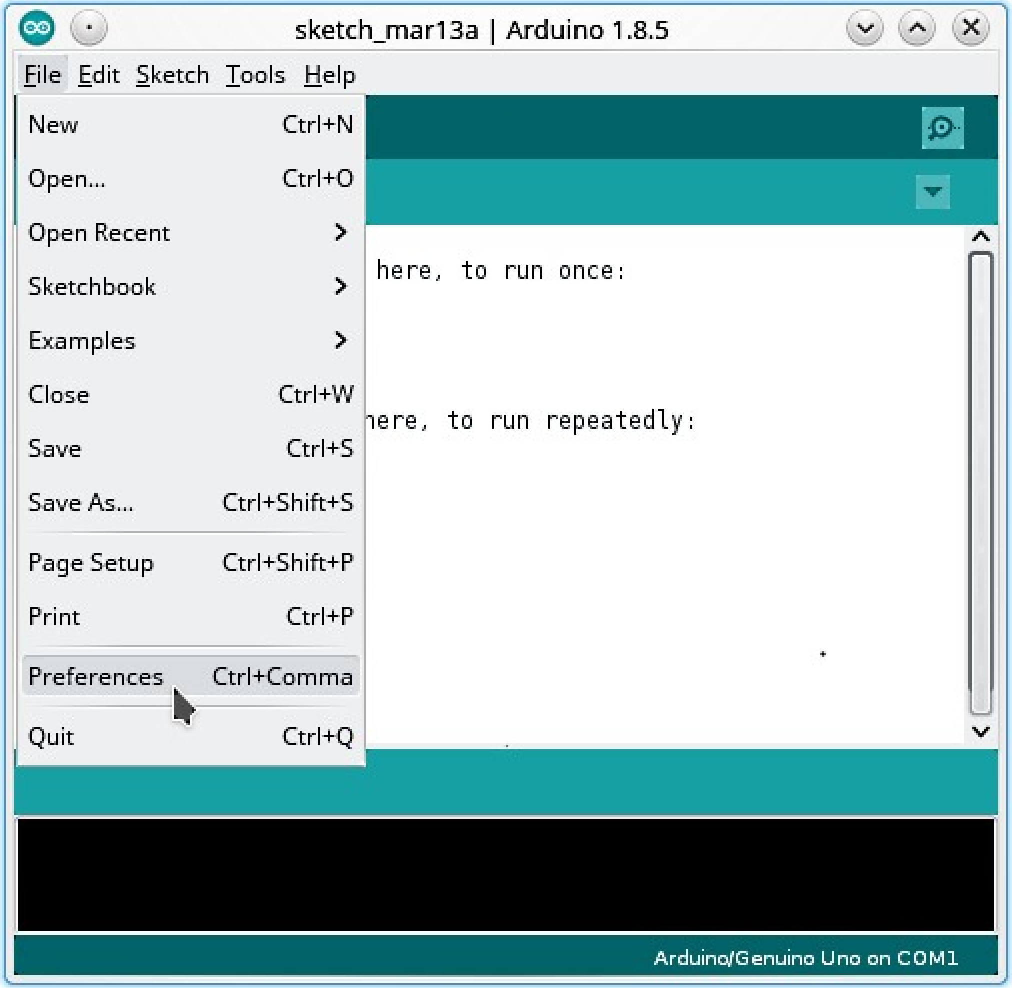
Arduino

The First Sketch – Libraries

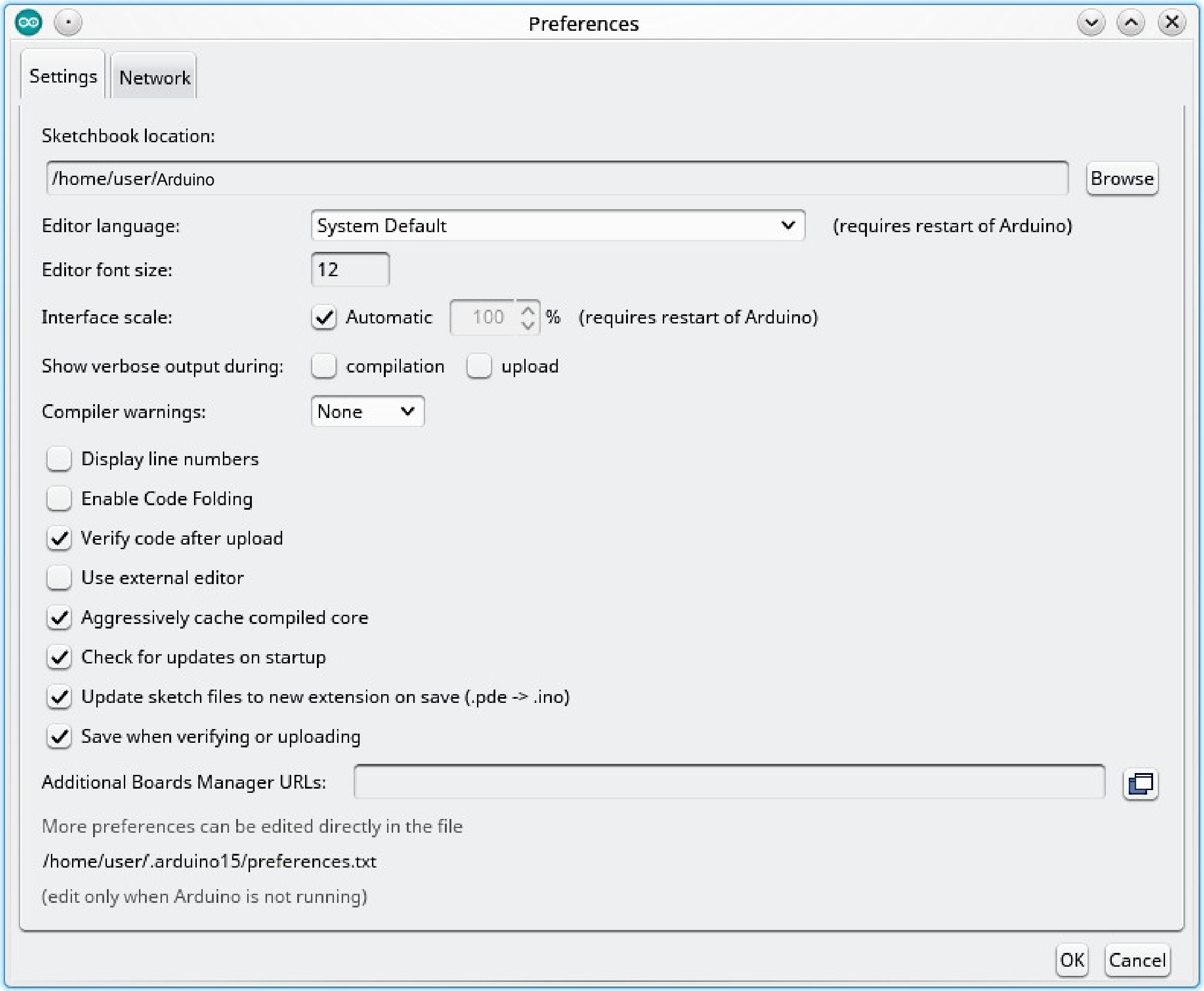
EMERTXE

Arduino

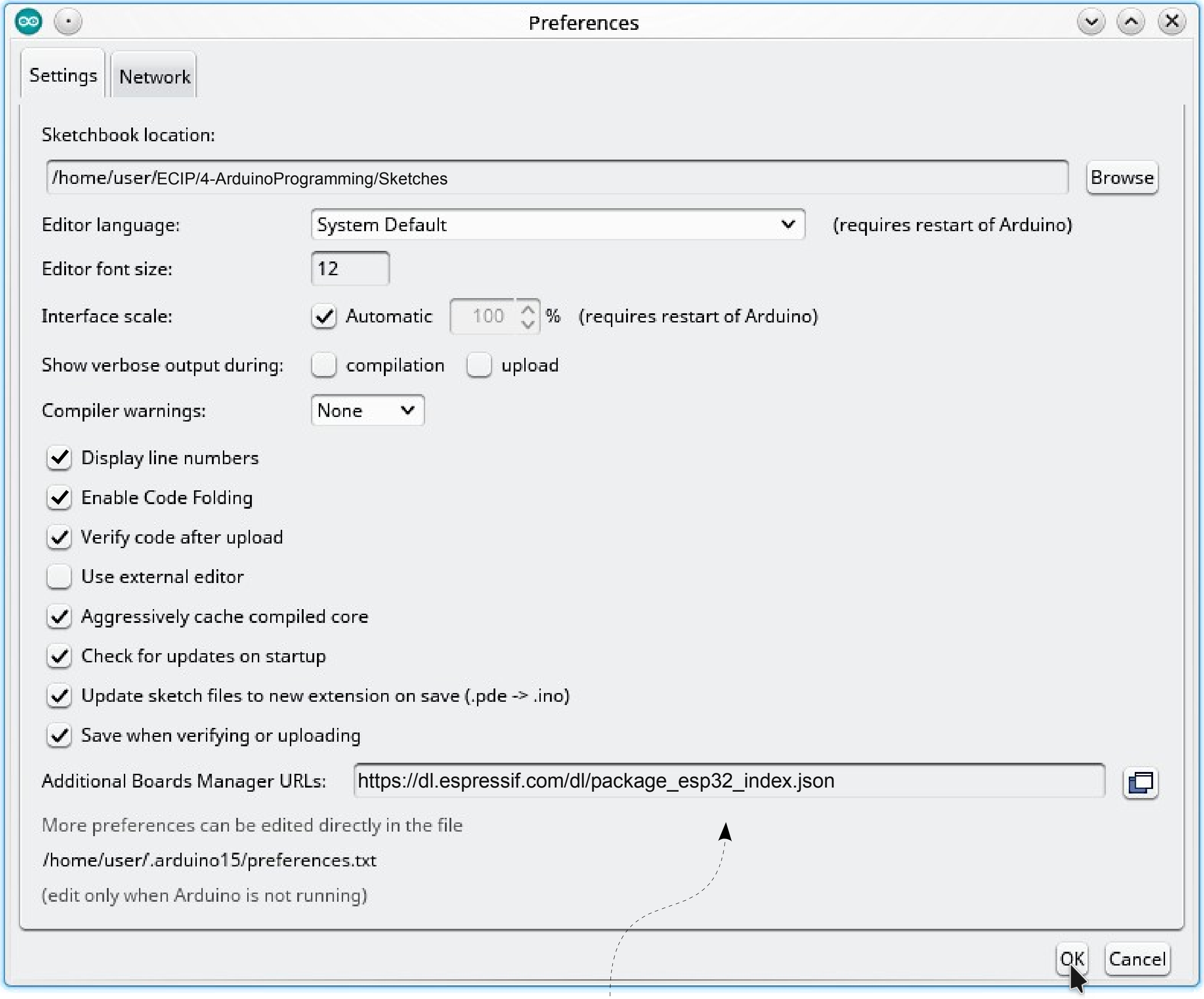
The First Sketch – Libraries



Arduino
The First Sketch – Libraries



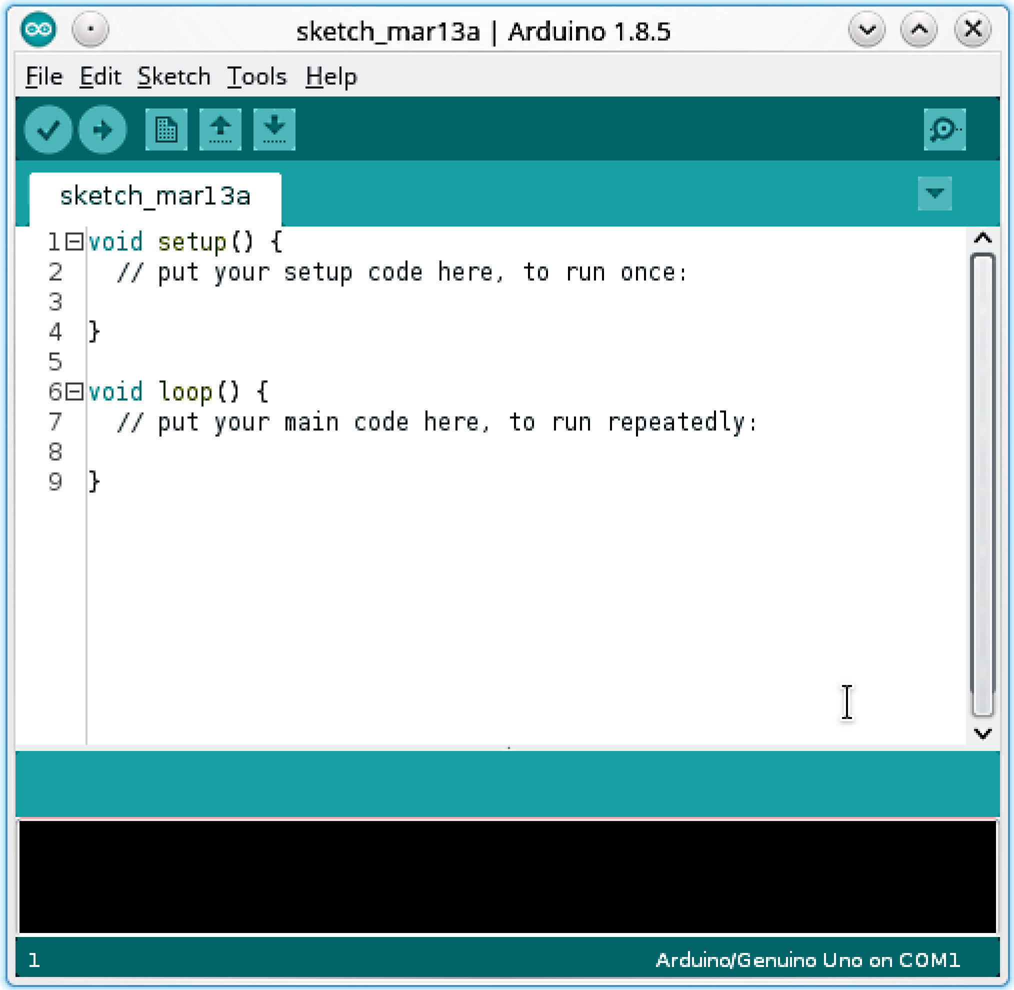
Arduino
The First Sketch – Libraries



https://dl.espressif.com/dl/package_esp32_index.json

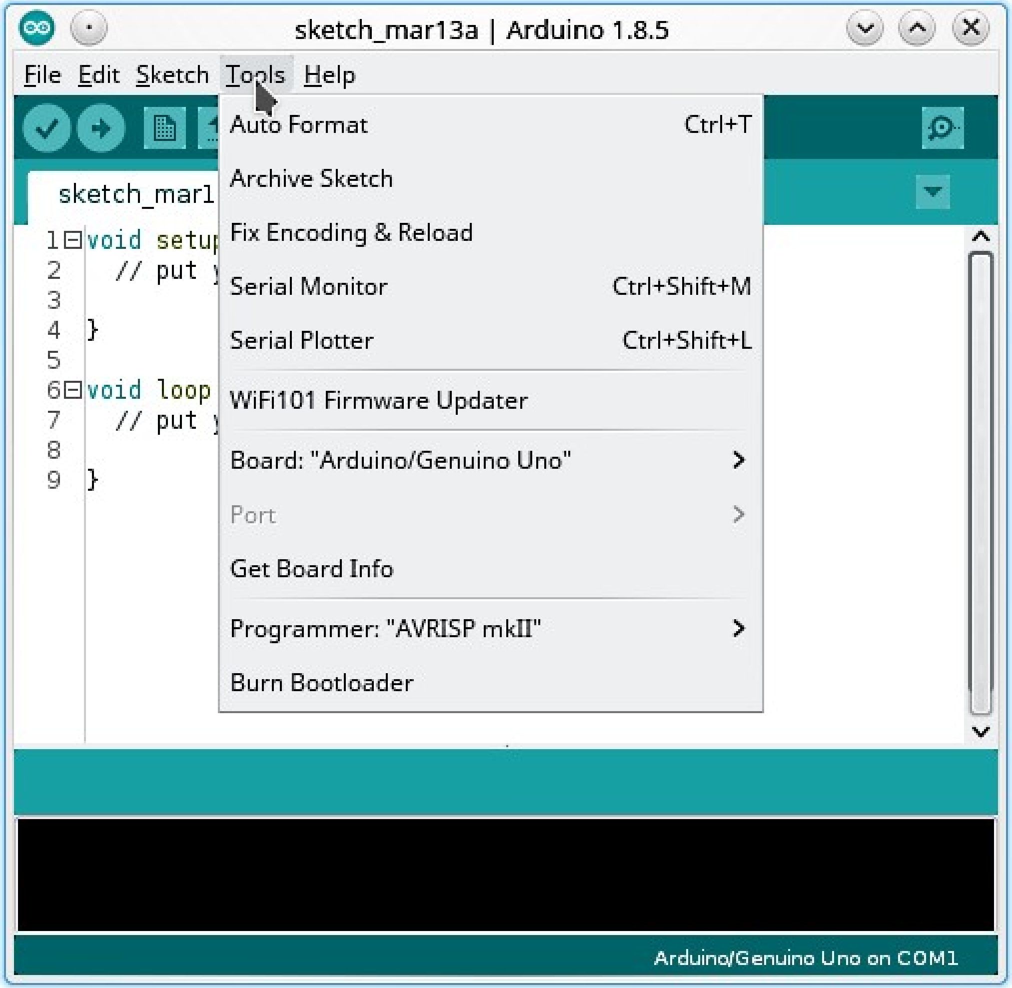
Arduino

The First Sketch – Libraries



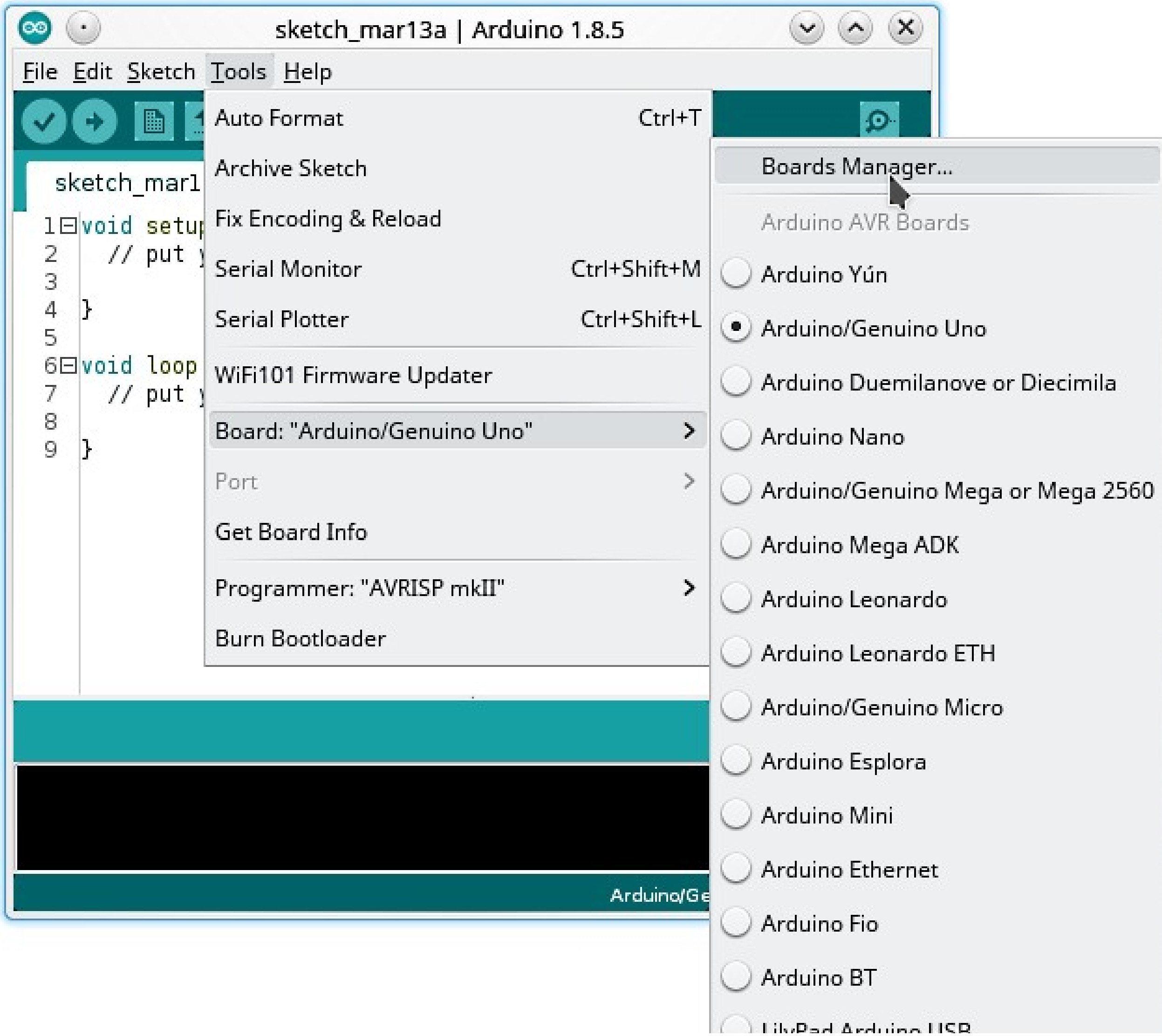
Arduino

The First Sketch – Libraries



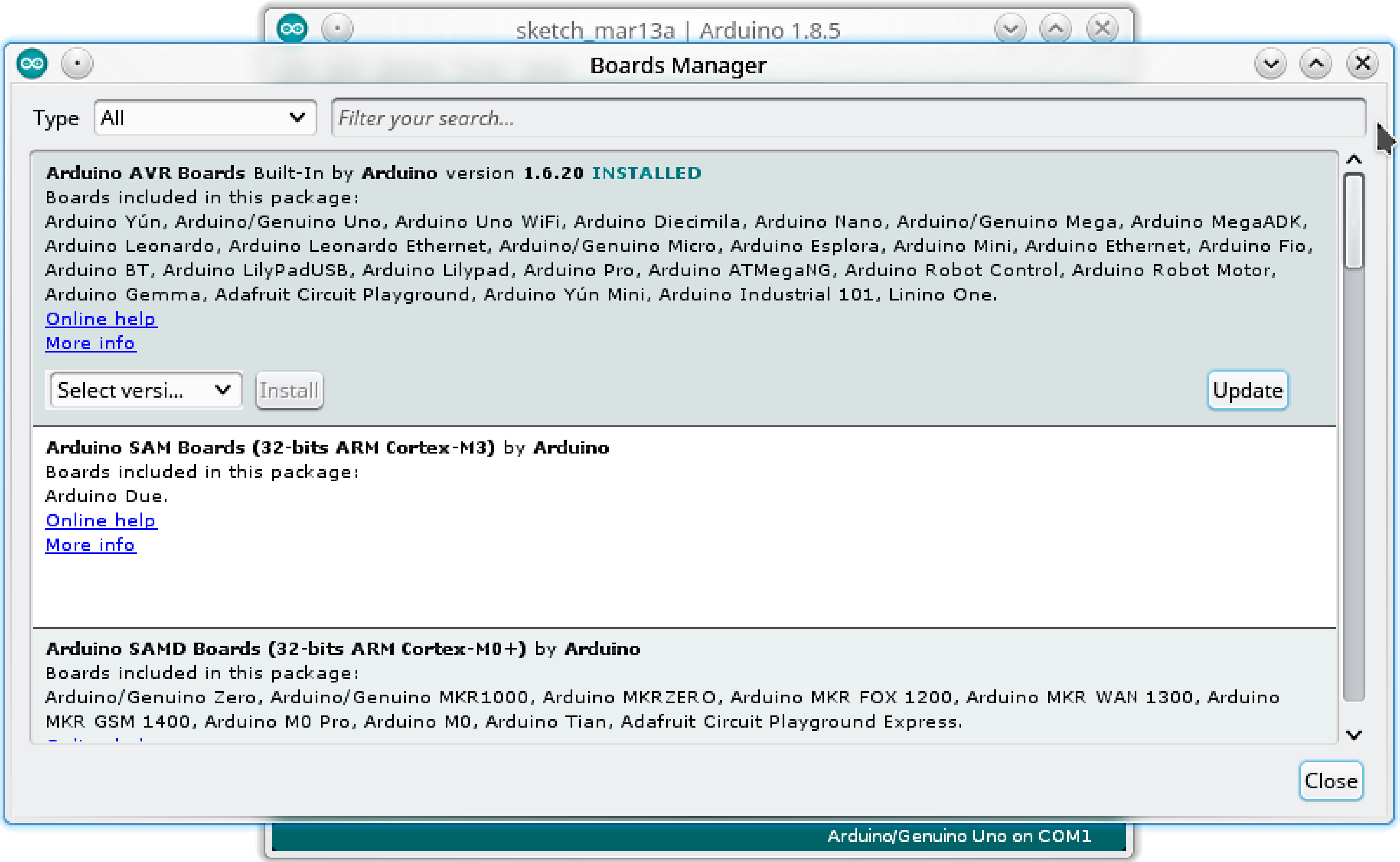
Arduino

The First Sketch – Libraries



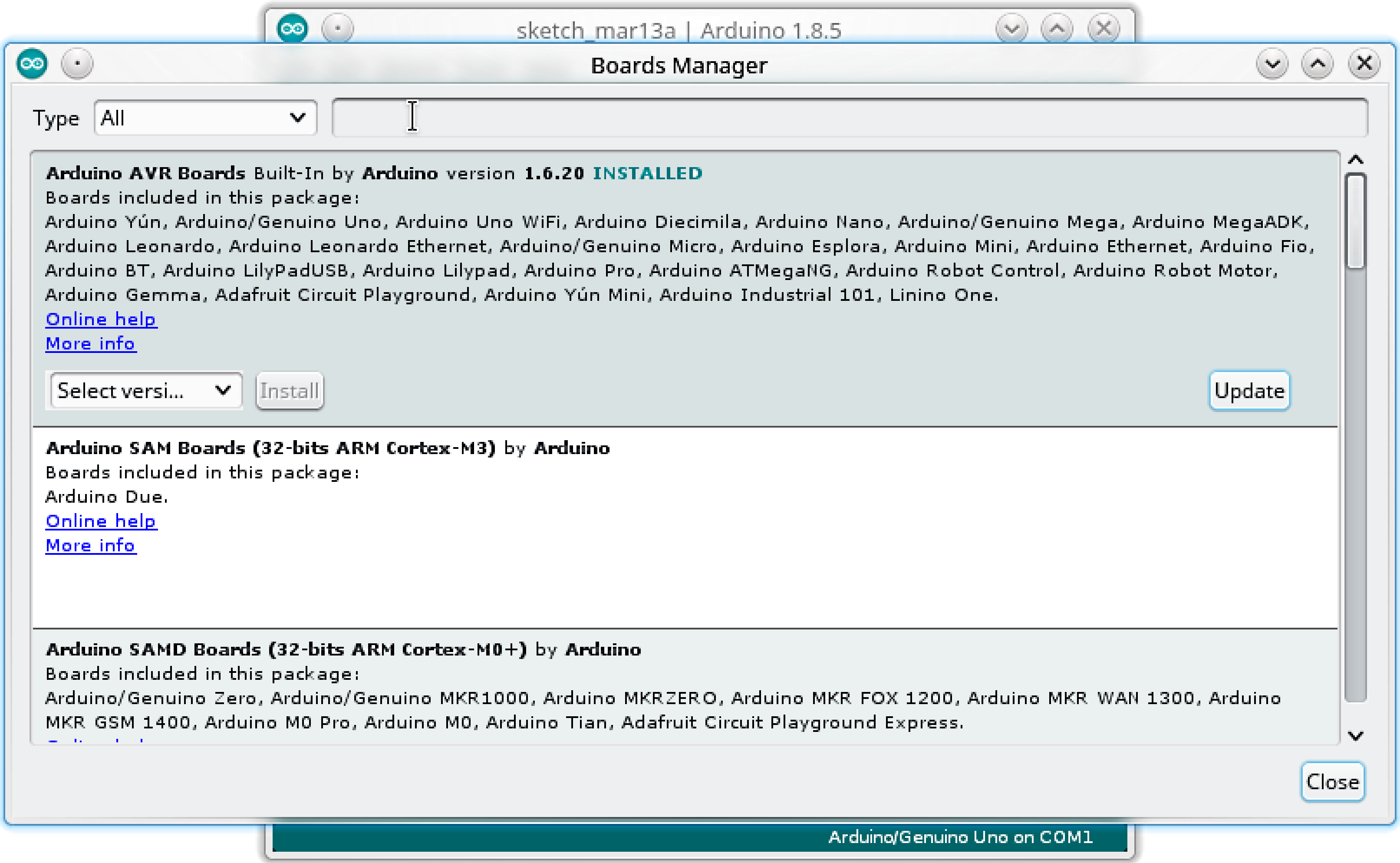
Arduino

The First Sketch – Libraries



Arduino

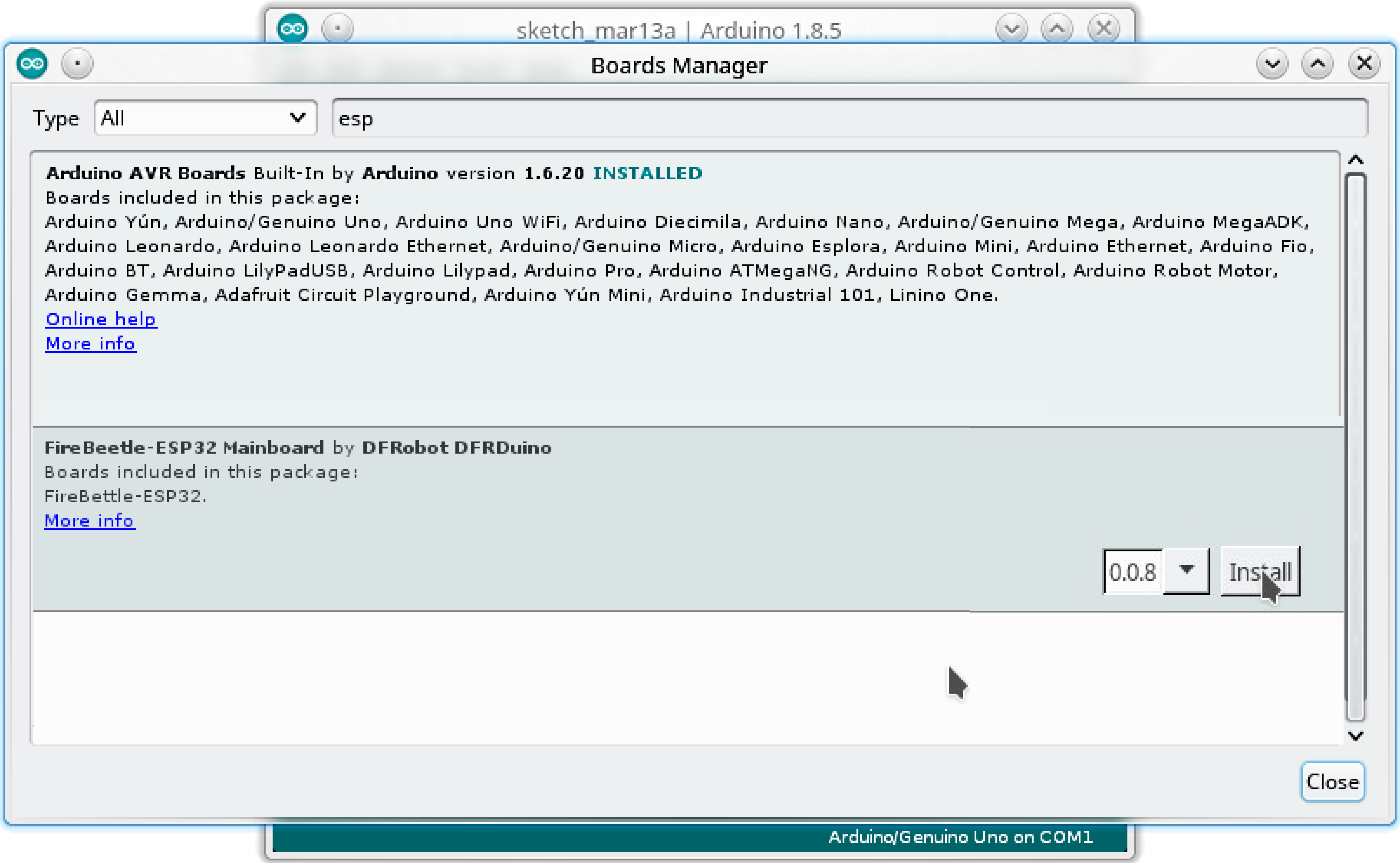
The First Sketch – Libraries



ΣMERTXE

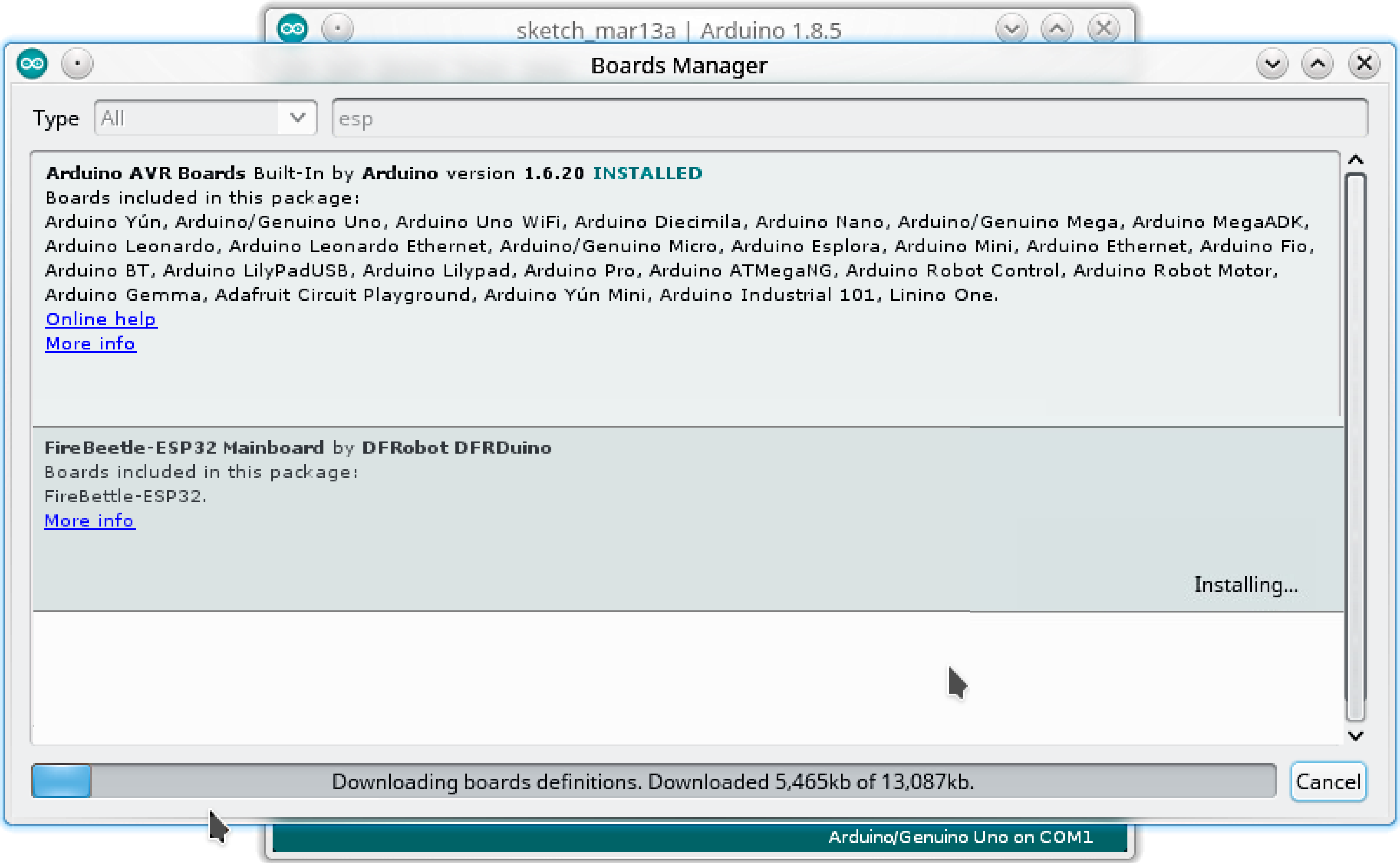
Arduino

The First Sketch – Libraries



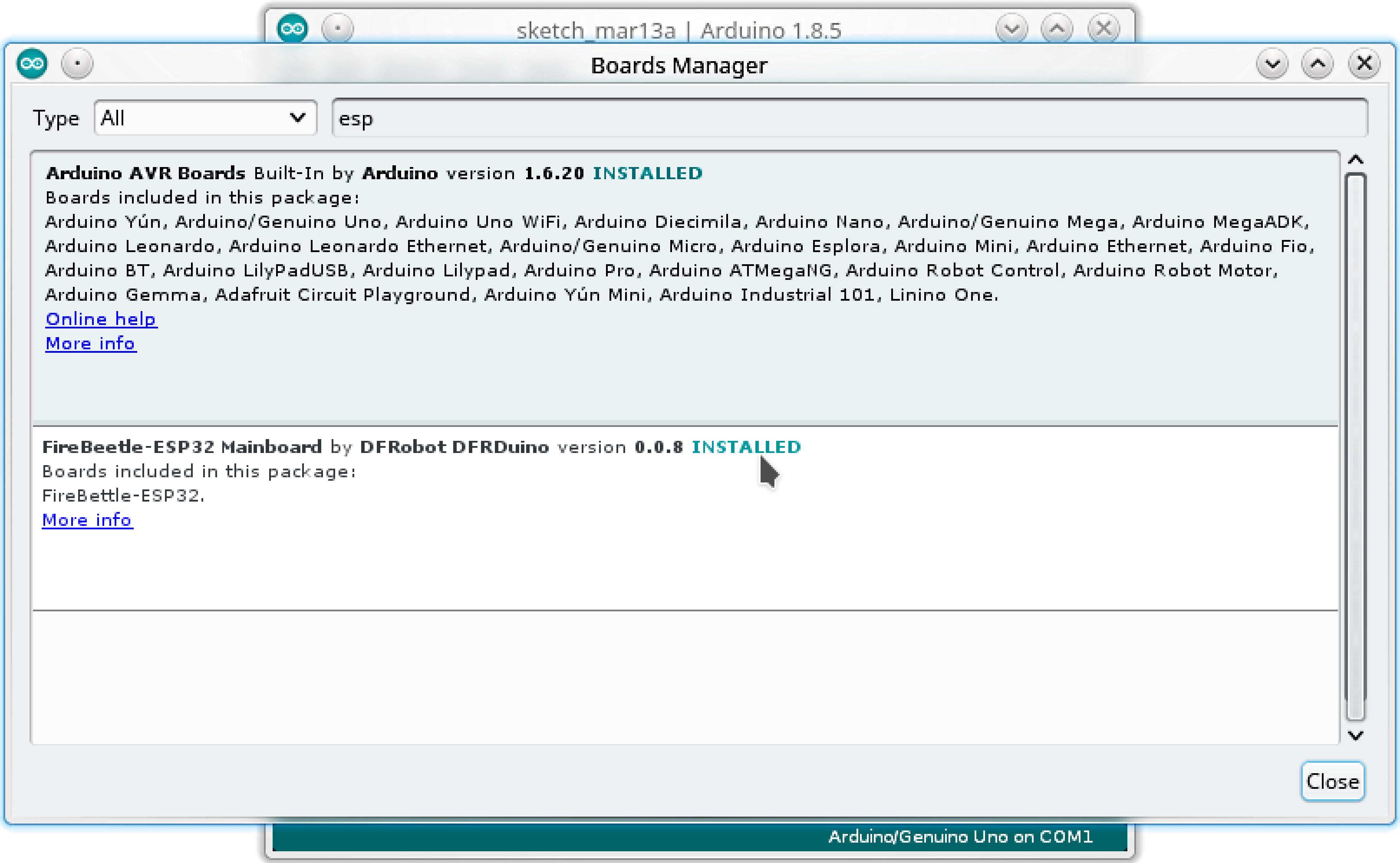
Arduino

The First Sketch – Libraries



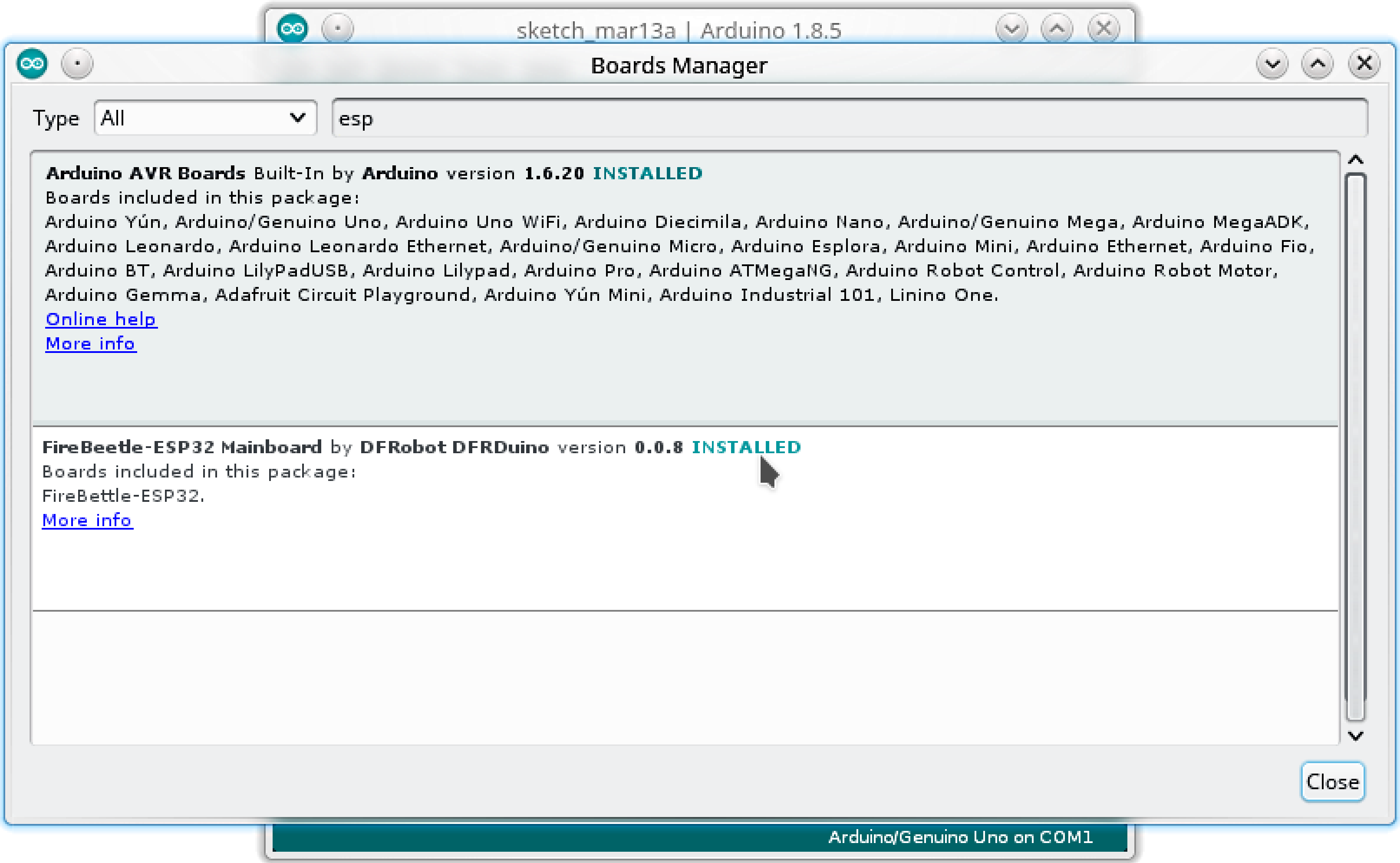
EMERTXE

Arduino
The First Sketch – Libraries



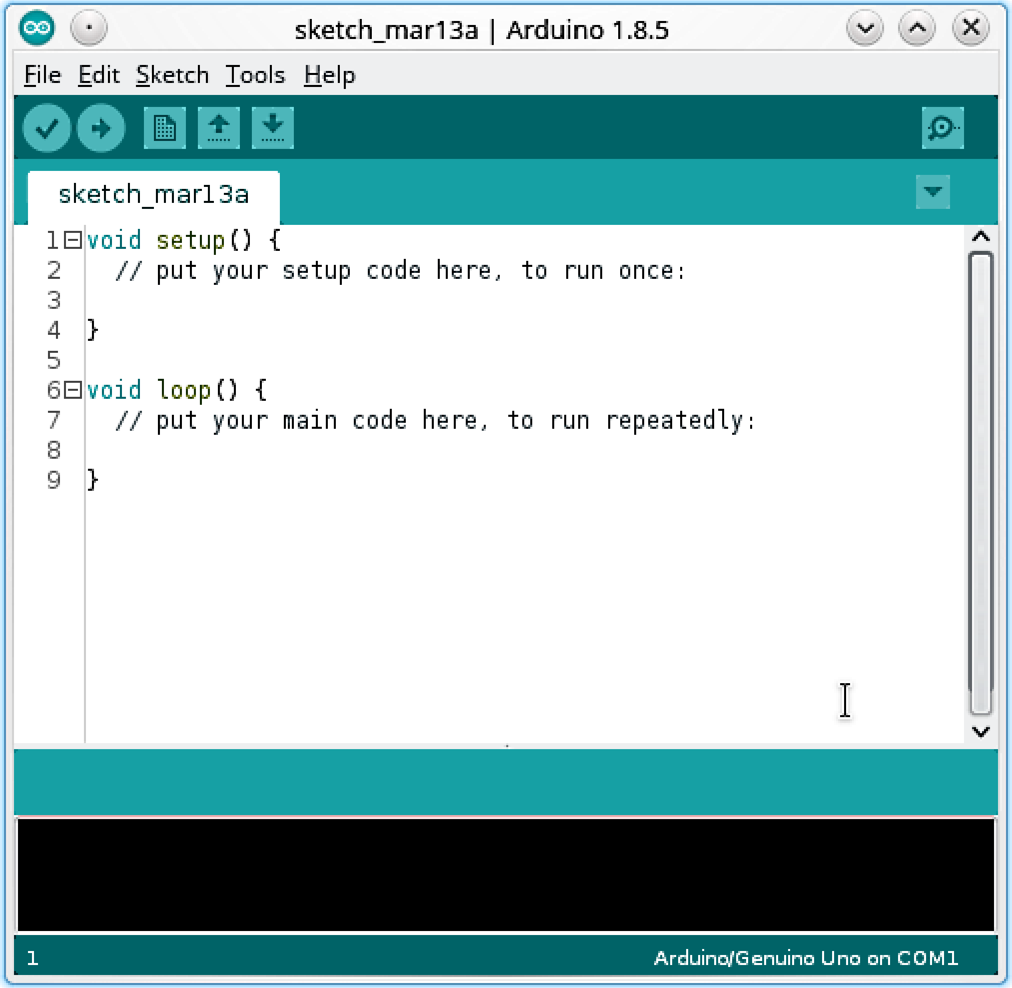
Arduino

The First Sketch – Libraries



Arduino

The First Sketch – Libraries



Arduino

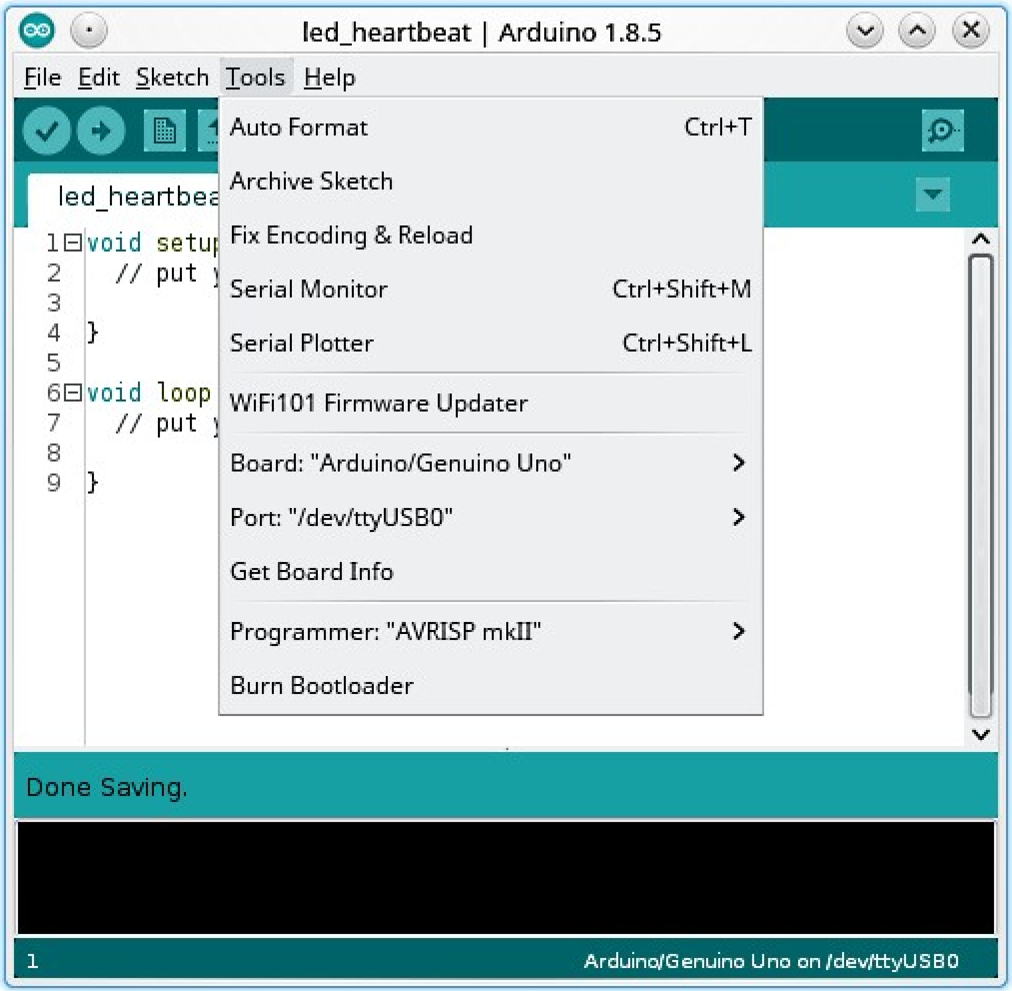
The First Sketch – Libraries

- If you have followed all the steps upto the previous slide then, the library for ESP32 would have been installed
- The next step would be selecting your target board
- Make sure you have **connected the Target board** before proceeding further
- Save the existing sketch as **led_heartbeat** (You may the follow the steps given in IDE introduction)

Arduino
The First Sketch – Host and Target Interface

Arduino

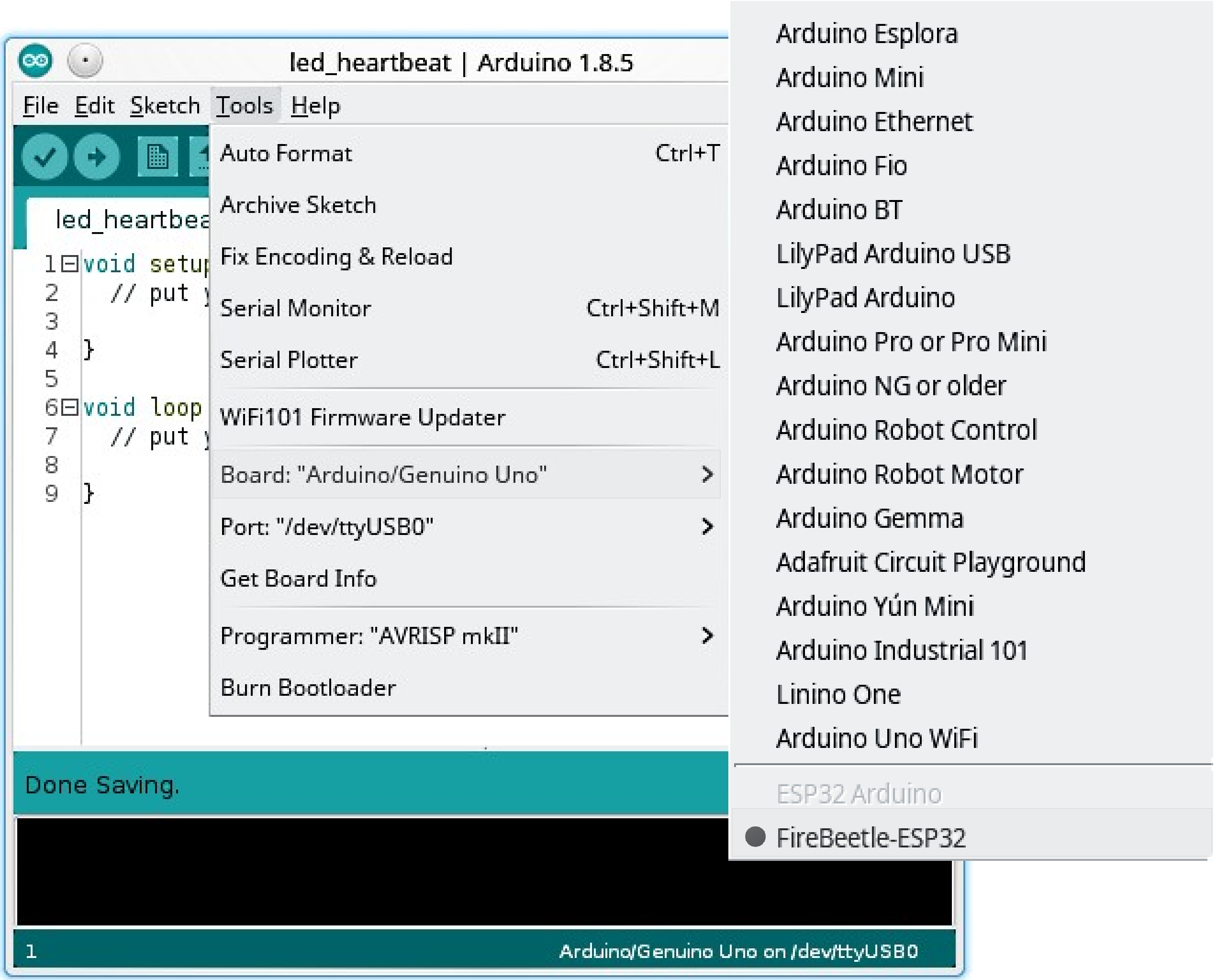
The First Sketch – Host and Target Interface



EMERTXE

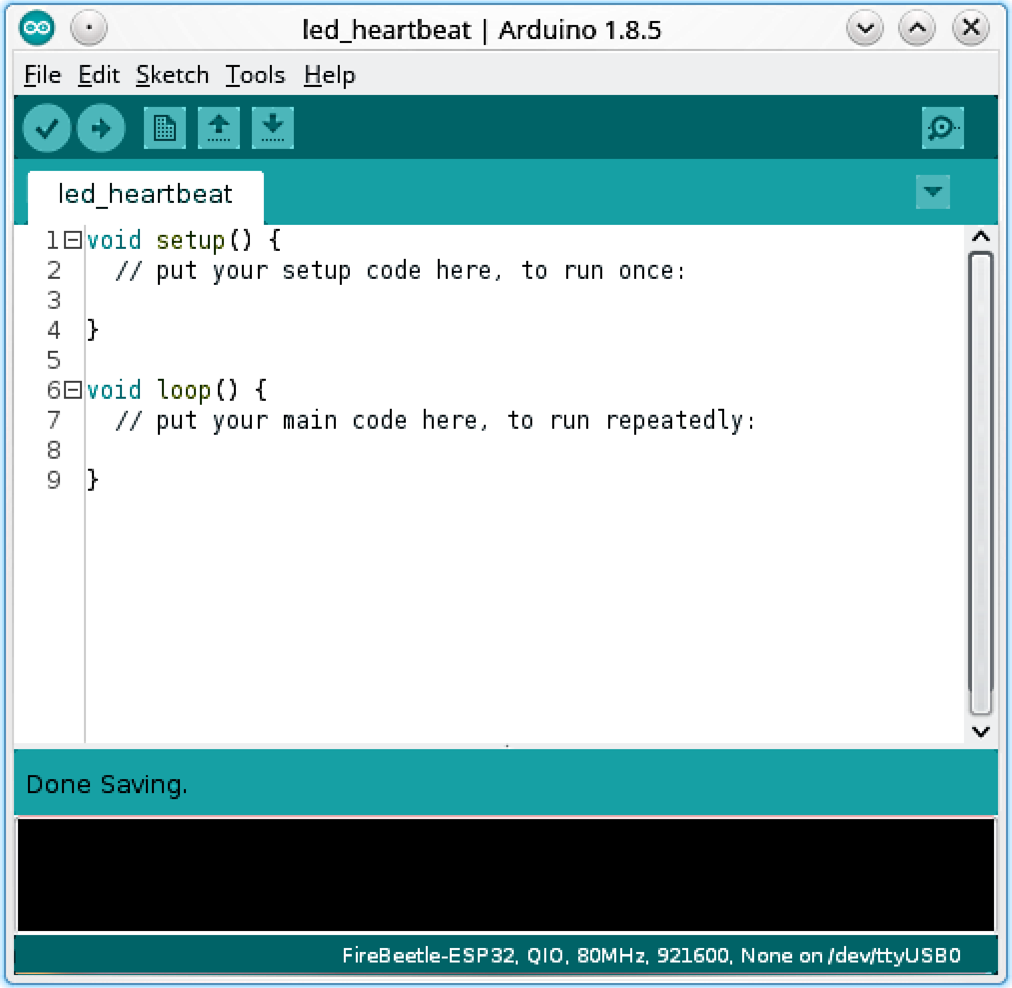
Arduino

The First Sketch – Host and Target Interface



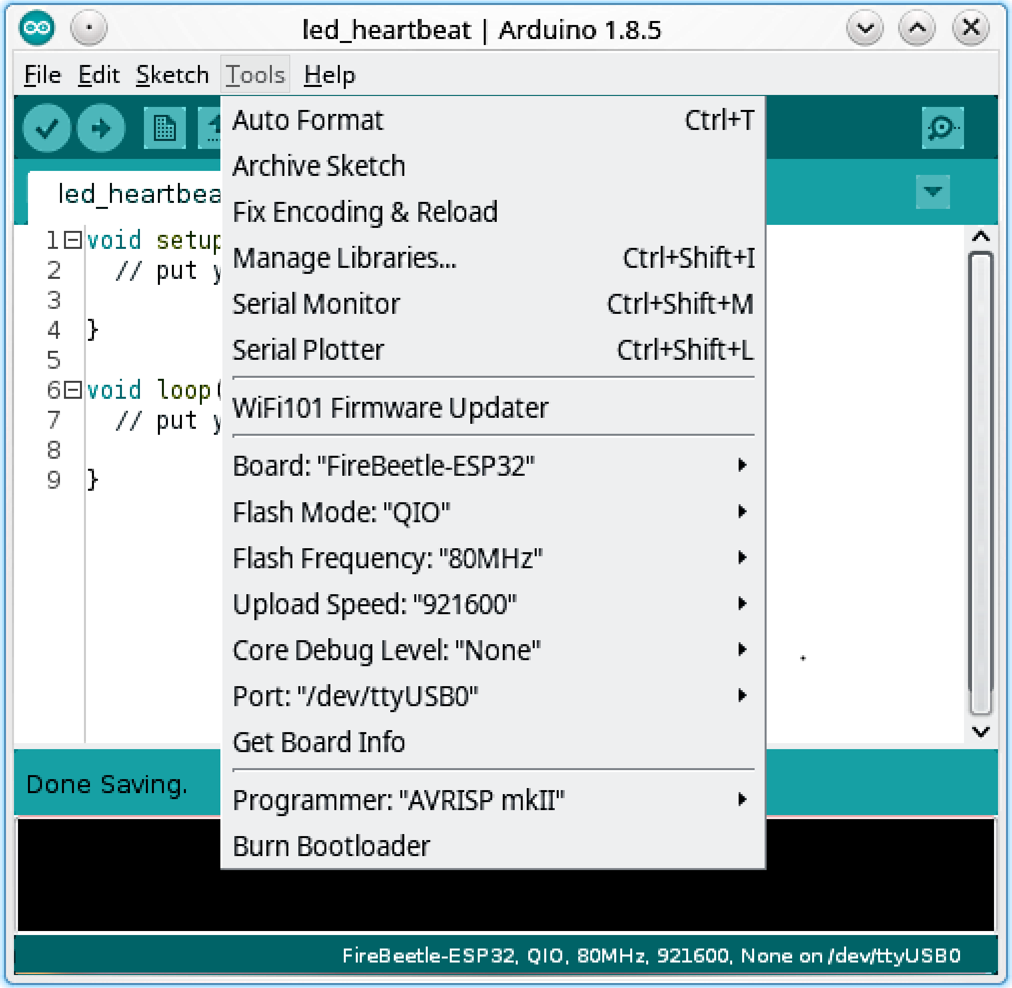
Arduino

The First Sketch – Host and Target Interface



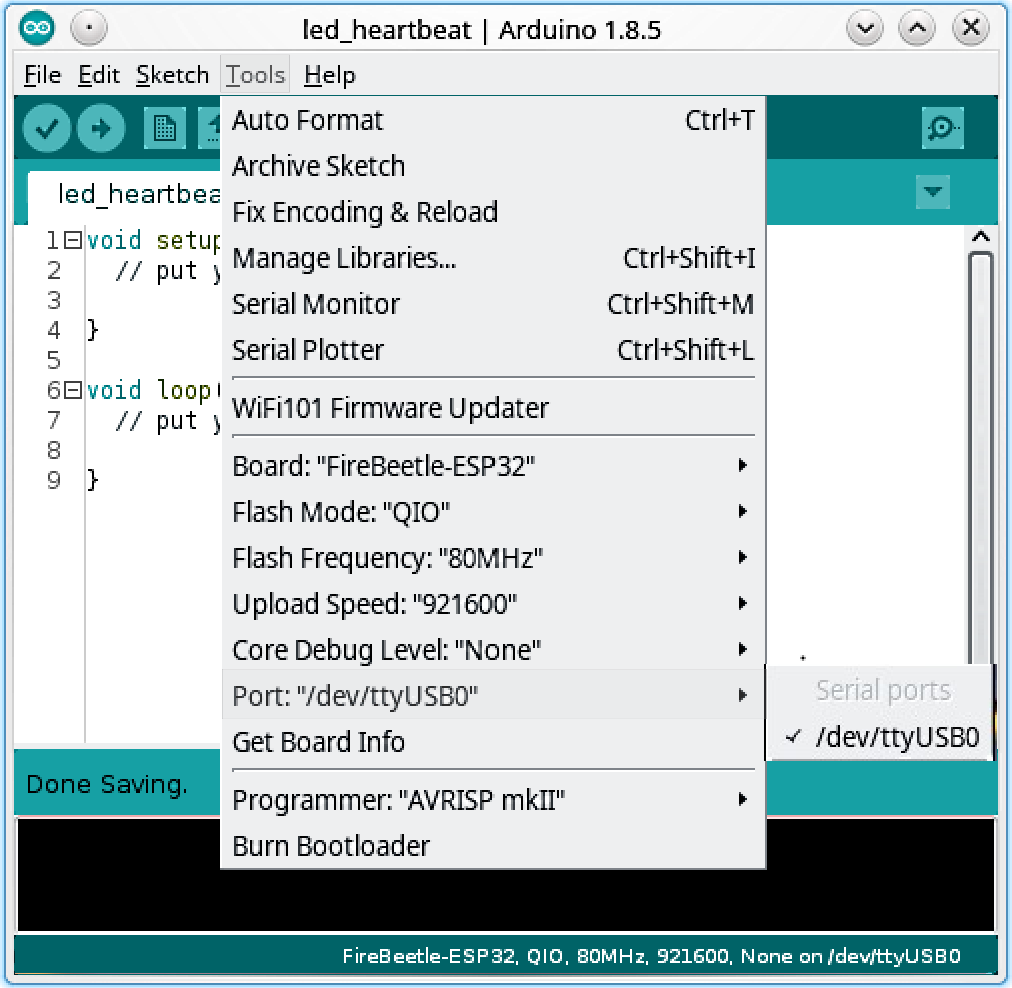
Arduino

The First Sketch – Host and Target Interface



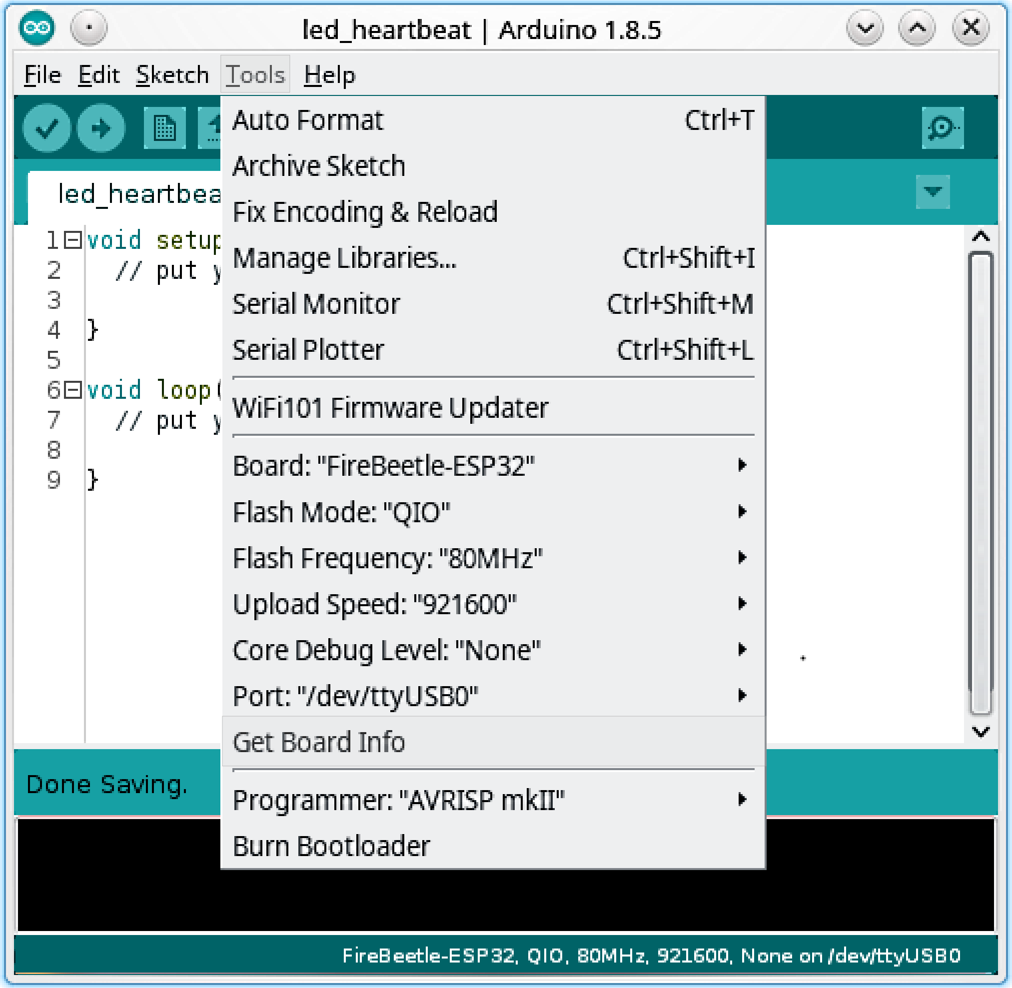
Arduino

The First Sketch – Host and Target Interface



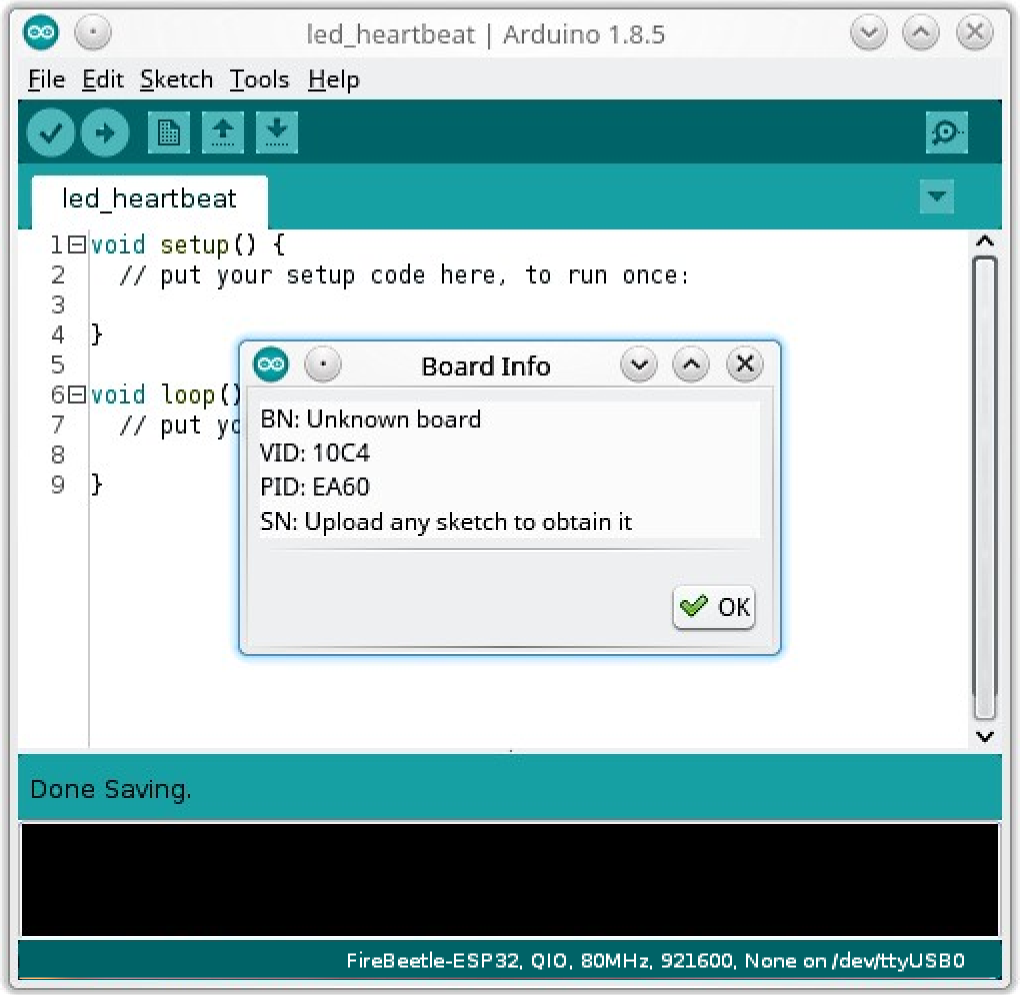
Arduino

The First Sketch – Host and Target Interface



Arduino

The First Sketch – Host and Target Interface



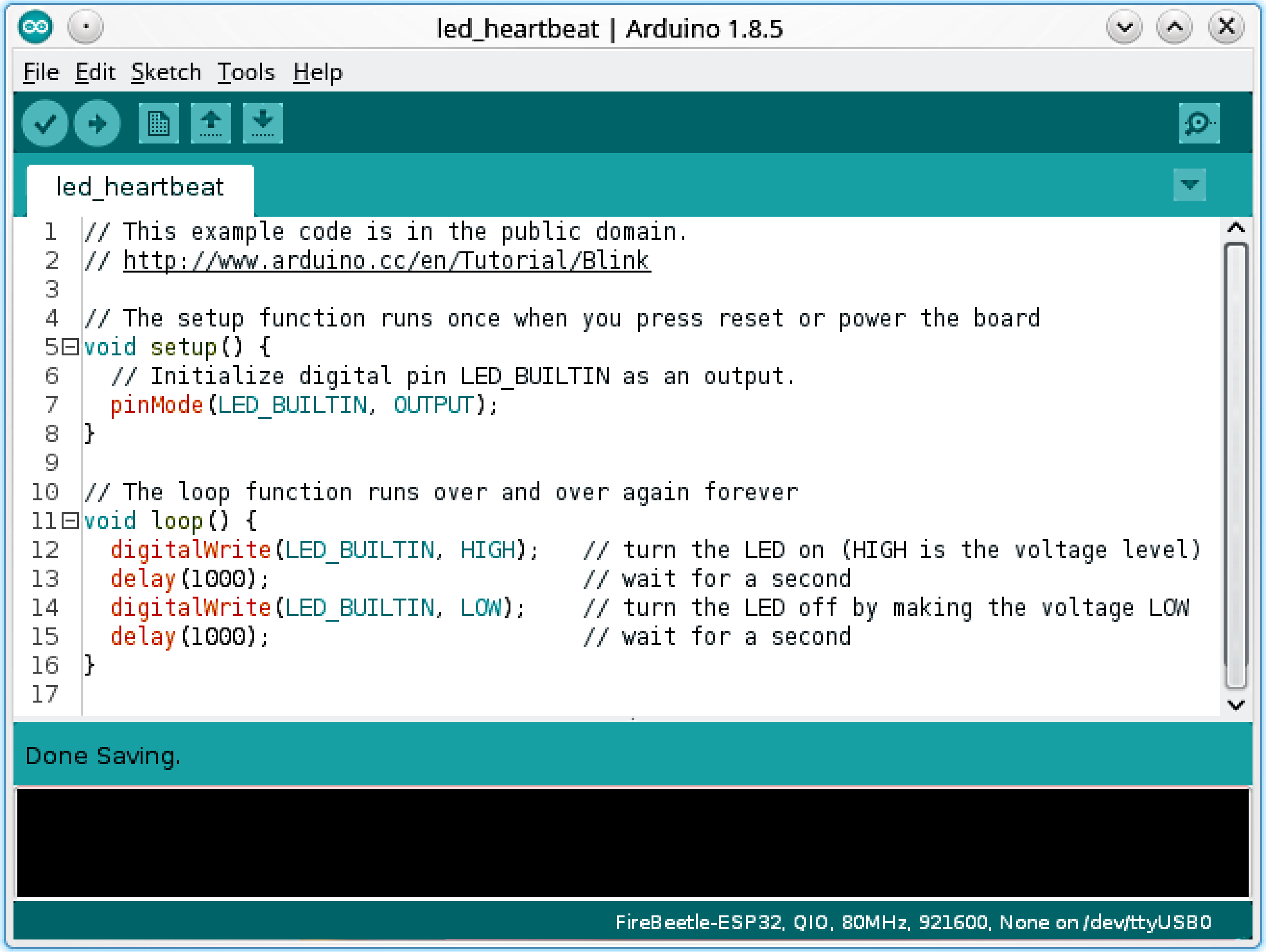
Arduino

The First Sketch – Coding

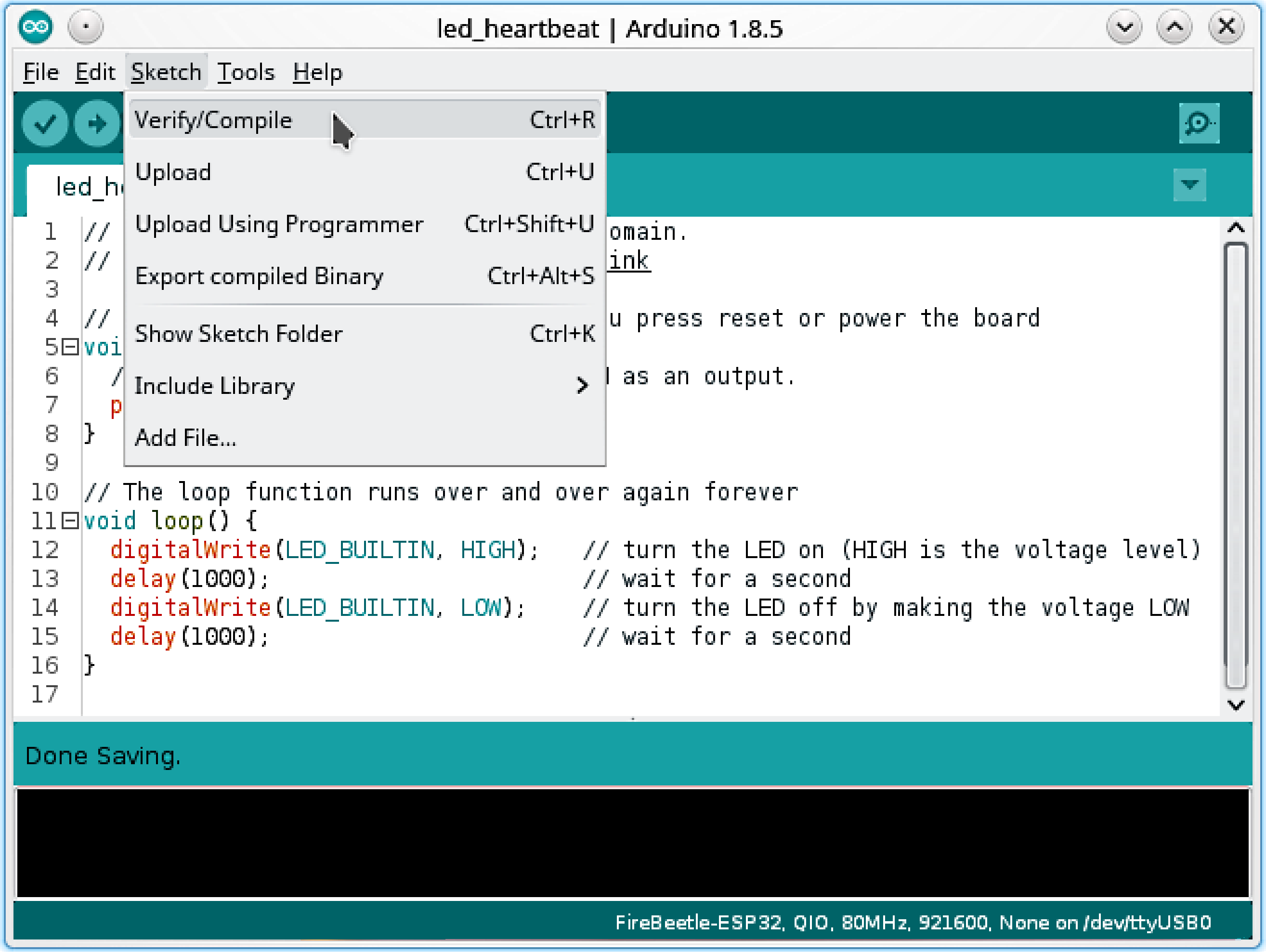
- Now that everything is ready let’s move toward programming the target board
- From all the information we have in previous slides, we can use the LED blinky example from the arduino website as is!!
- Please refer the next slide

Arduino

The First Sketch – Code

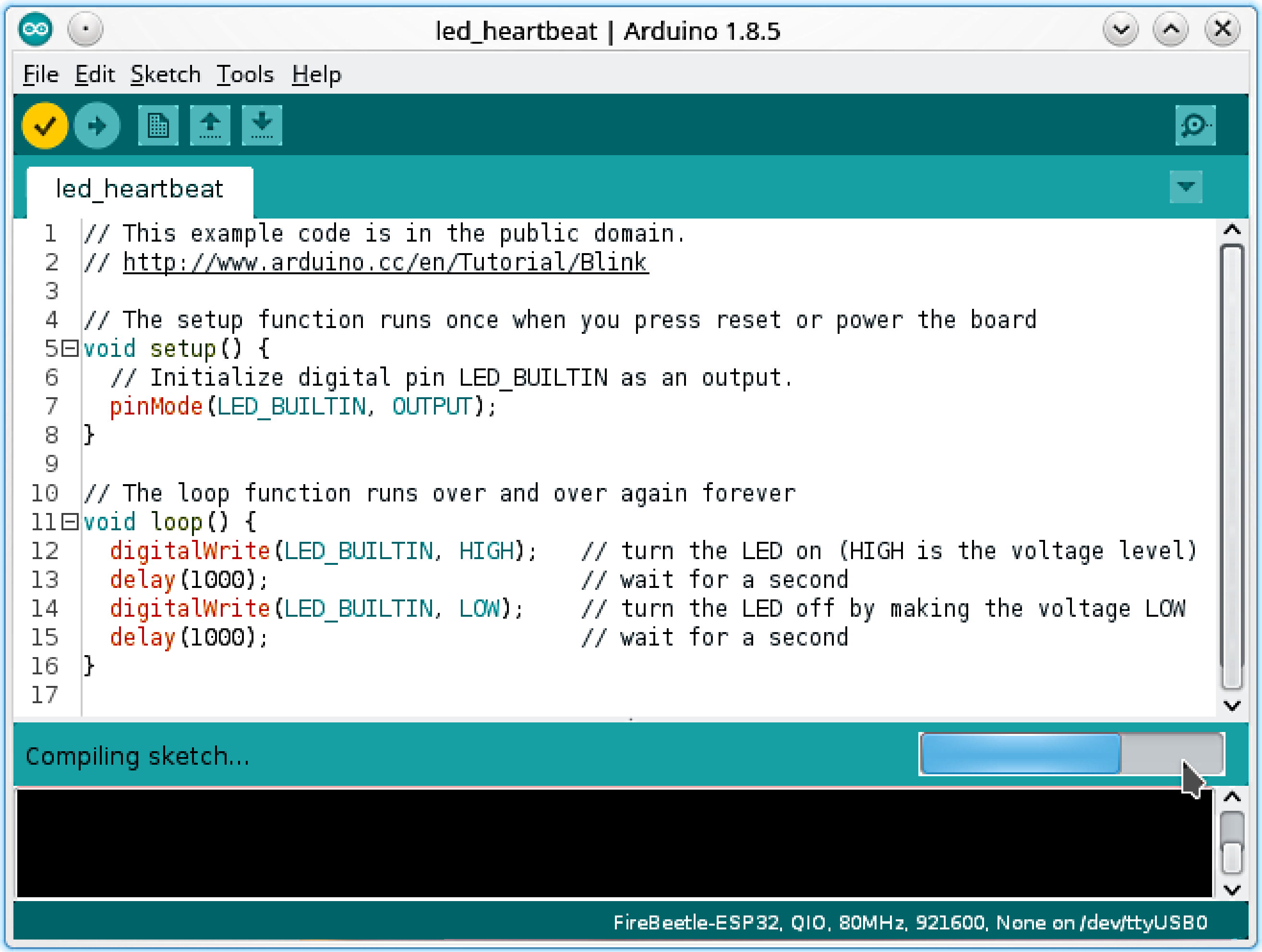


Arduino
The First Sketch – Compile



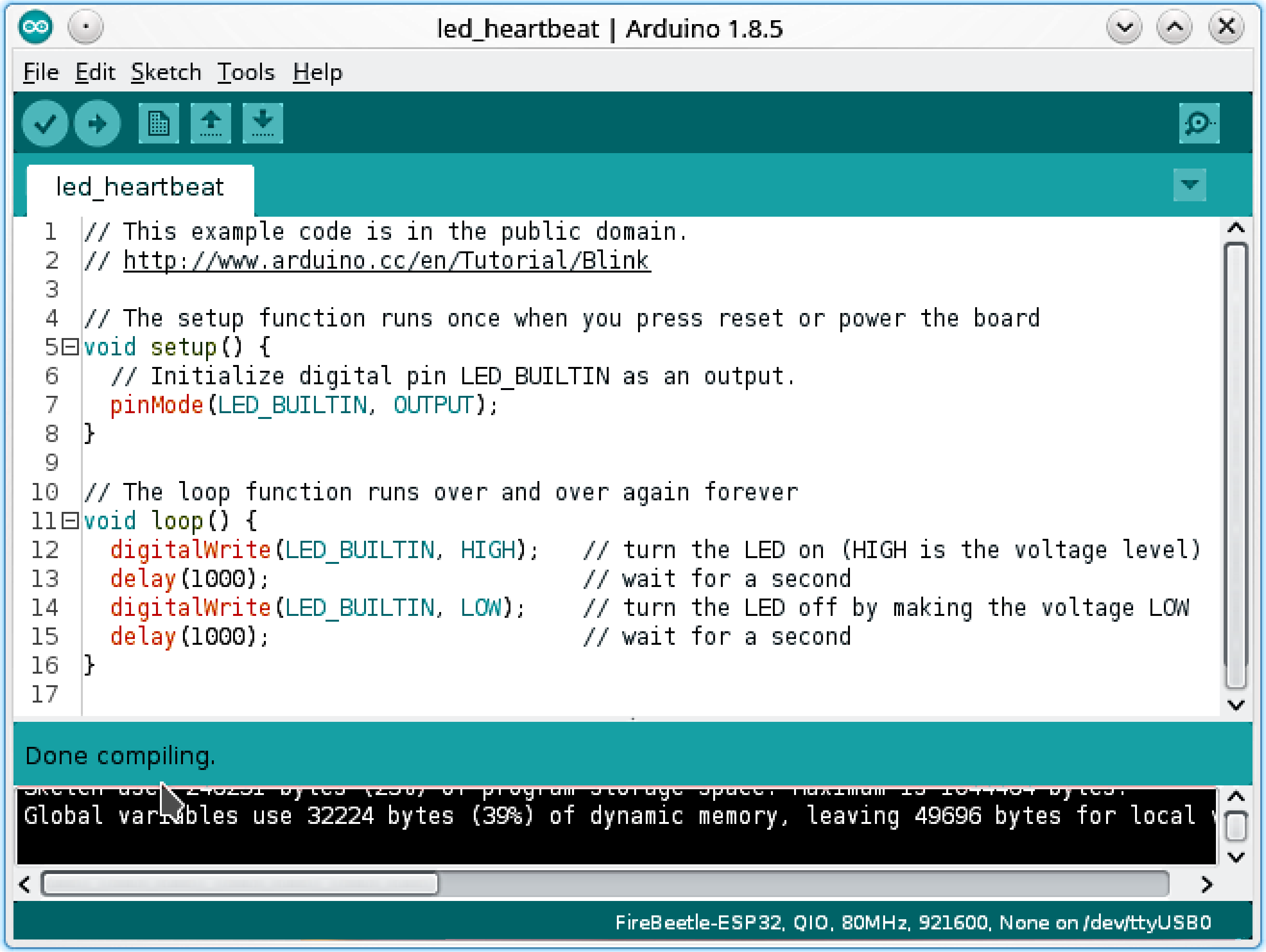
Arduino

The First Sketch – Compile



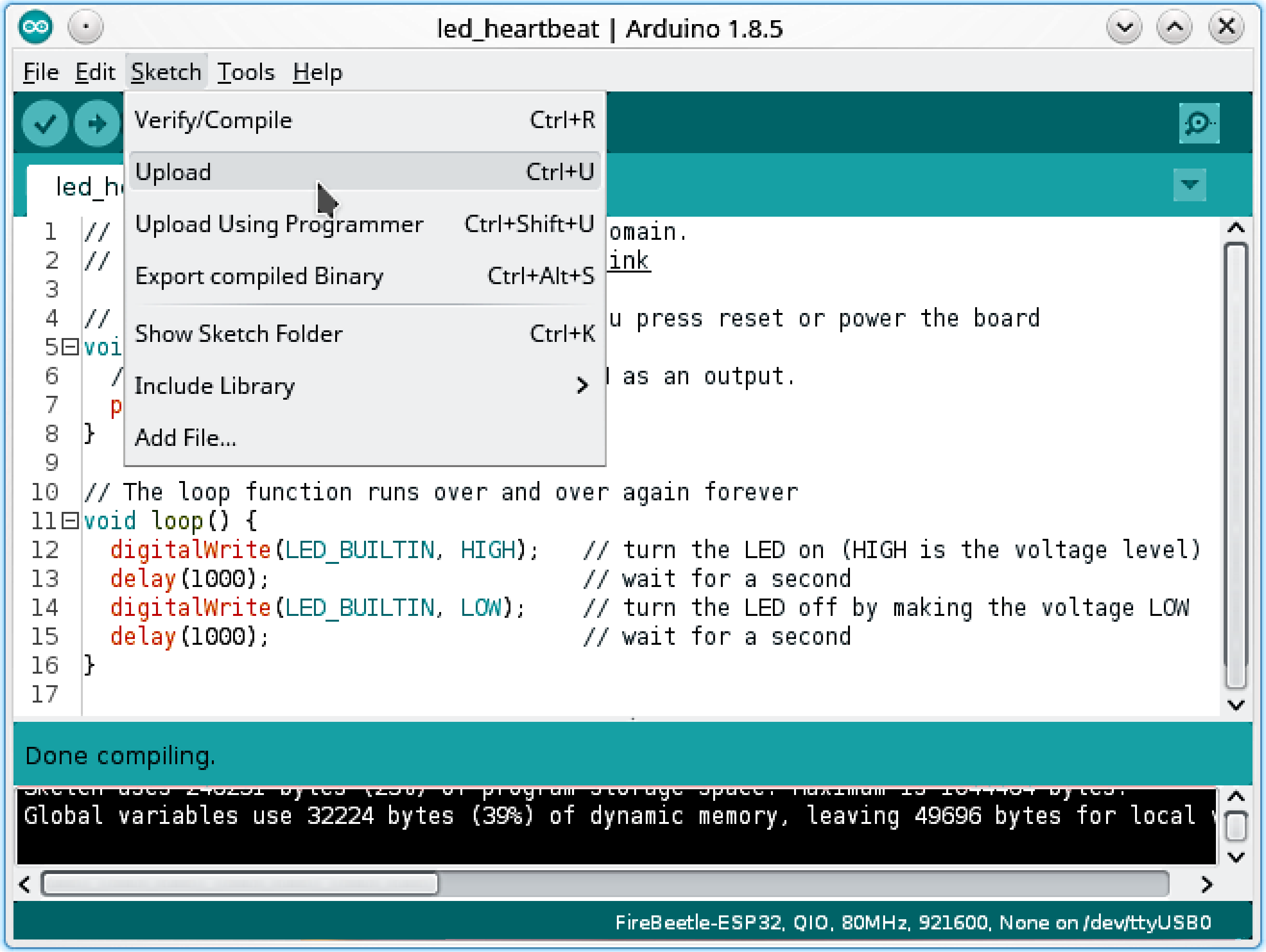
Arduino

The First Sketch – Compile



Arduino

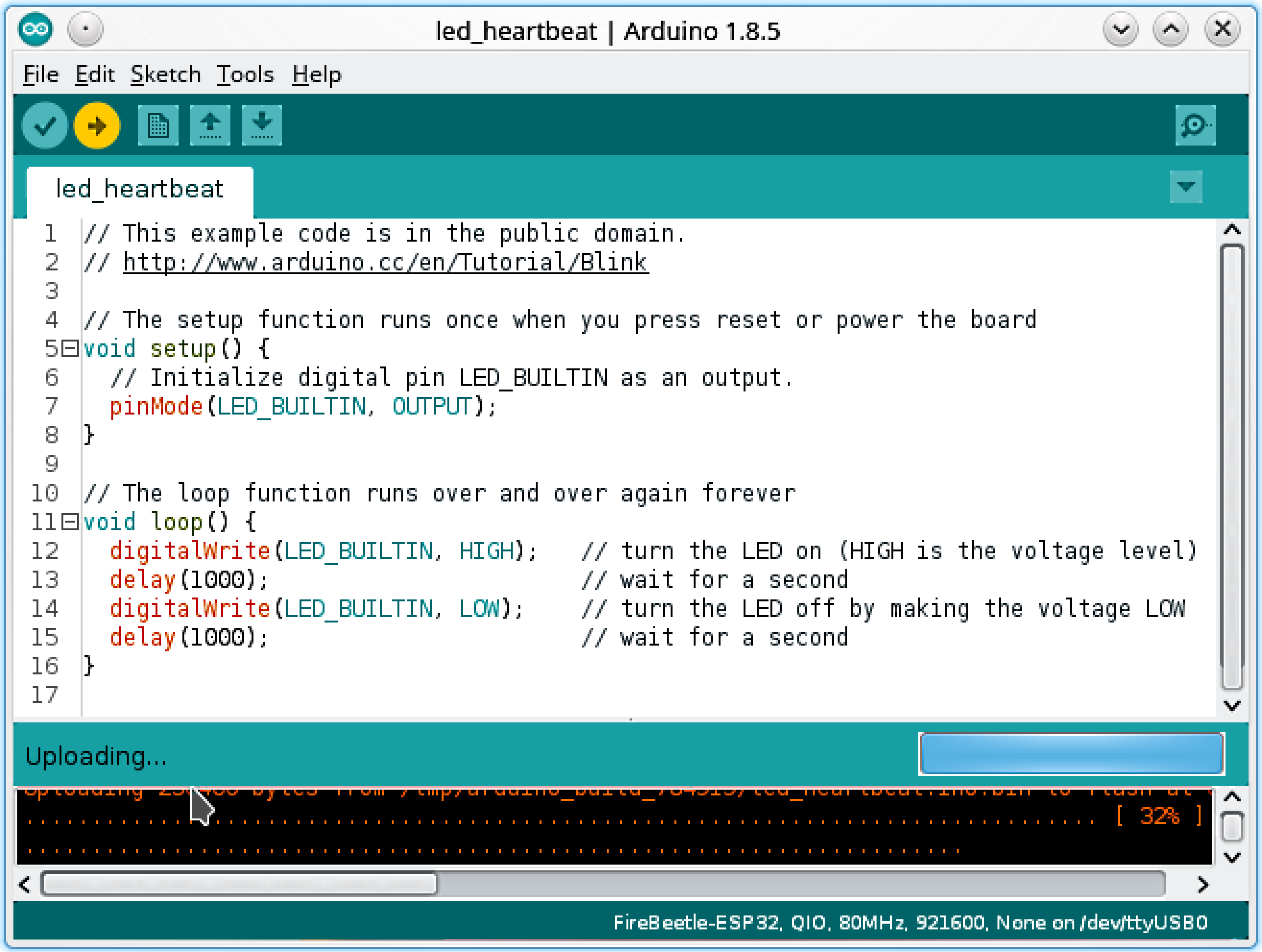
The First Sketch – Upload



EMERTXE

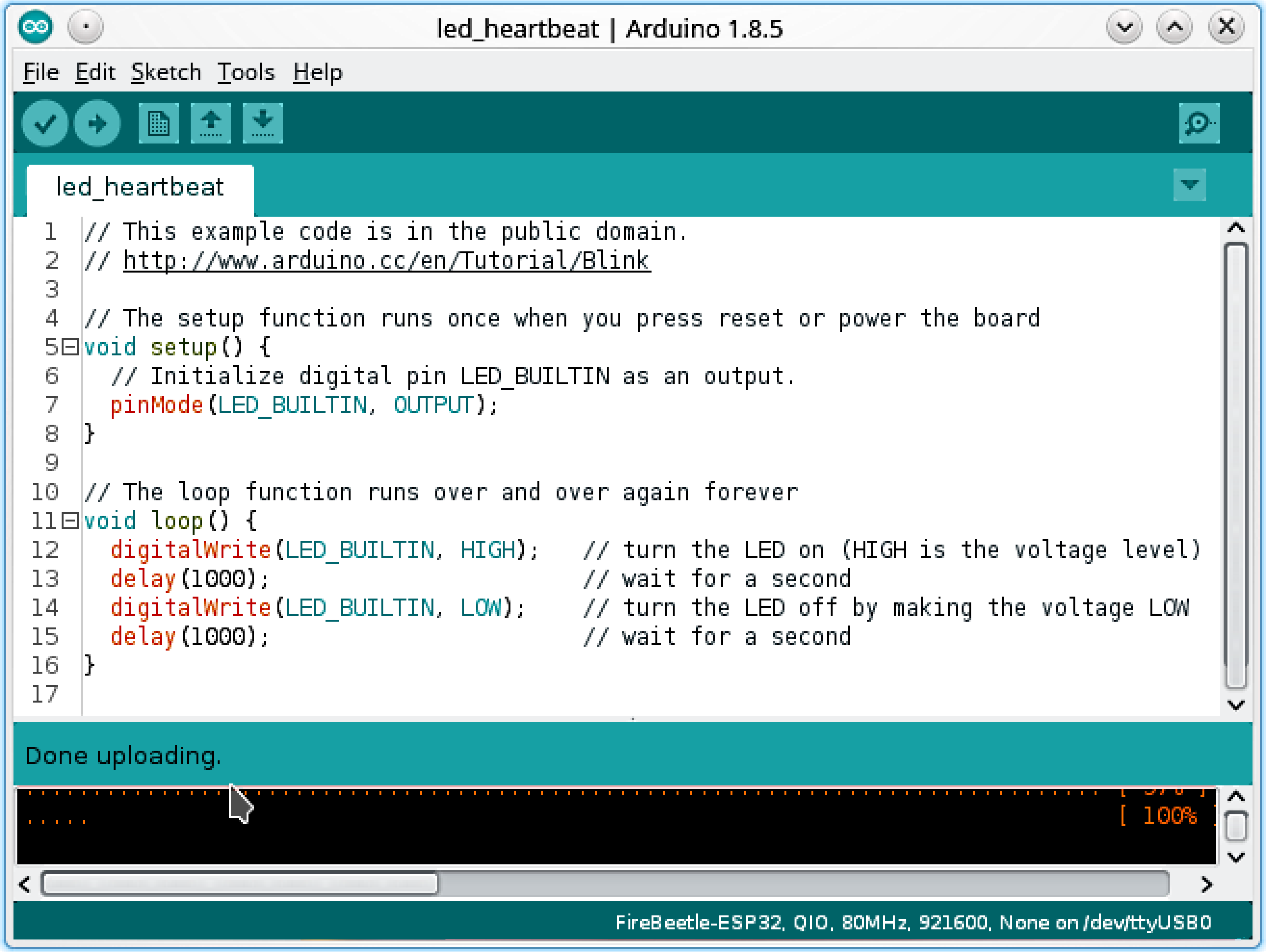
Arduino

The First Sketch – Upload



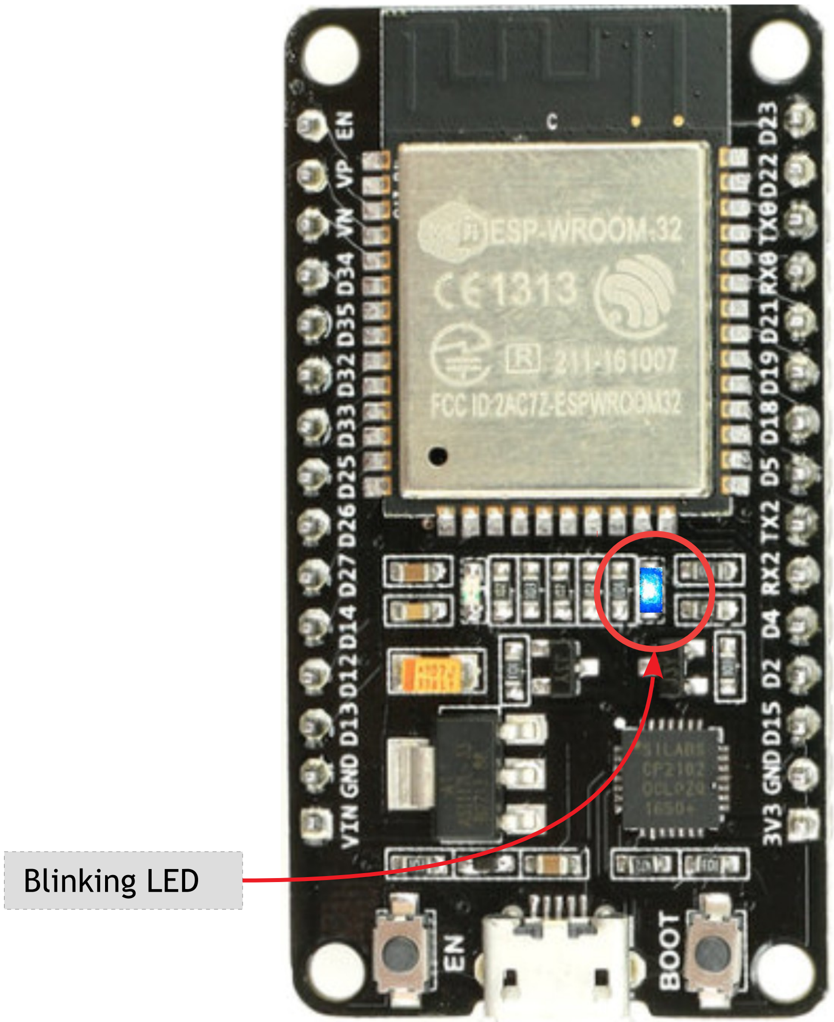
Arduino

The First Sketch – Upload



Arduino

The First Sketch – Output



Arduino

The First Sketch – Summary

- So from our first sketch we come to know that there are some built-in functions or classes to be used!
- The next topic covers some of the most commonly used functions or classes in Arduino

Thank You

Arduino

Classes and Functions

Team Emertxe

Classes

Arduino

Class – What?

A set or category of things having some property or attribute in common and differentiated from others by kind, type, or quality.

Source: google

Template definition of the methods and variables in a particular kind of object

Source: google

Arduino

Class – Why?

- A technique which helps to describe the object completely from properties to its implementation
- Acts as blue print, which helps us to create objects of the same type!.
- What do you understand from the image put in the next slide?

Arduino
Class – Why?






Arduino
Class – Why?

- What is to be understood here is the blueprint of bicycle will always be same, like its going to have 2 tiers, a seat, a handle etc.,
- We may create different types of bicycles with a defined class

Arduino

Class – Where?

- Since Arduino is a open source platform, there many classes available to use.
- More information regarding these function can be obtained in link given below

  https://www.arduino.cc/reference/en/#functions

Communication

Arduino

Classes – Communication – Serial.begin()

Description	Sets the data rate in bits per second (baud) for serial data transmission
Syntax	`Serial.begin(speed)` `Serial.begin(speed, config)`

Parameters	`speed`: in bits per second (baud) - long
	`config`: sets data, parity, and stop bits. Some valid values are
	
	`SERIAL_5N1`
	`SERIAL_6N1`
	`SERIAL_7N1`
	`SERIAL_8N1` (the default)

Return	Nothing

Arduino

Classes – Communication - Serial.print()

Description	Prints data to the serial port as human-readable ASCII text. This command can take many forms. Numbers are printed using an ASCII character for each digit.
Syntax	`Serial.print(val)` `Serial.print(val, format)`
Parameters	`val`: the value to print - any data type `format`: specifies the number base (for integral data types) or number of decimal places (for floating point types)
Return	`size_t: print()` returns the number of bytes written, though reading that number is optional.

Example	`Serial.print(78) gives "78"`
	`Serial.print(1.23456) gives "1.23"`
	`Serial.print('N') gives "N"`
	`Serial.print("Hello world.") gives "Hello world."`
	`Serial.print(78, BIN) gives "1001110" where BIN can be replaced with OCT, DEC or HEX`
	`Serial.print(1.23456, 0) gives "1"`
	`Serial.print(1.23456, 2) gives "1.23"`

Arduino

Classes – Communication - Serial.println()

Description	Prints data to the serial port as human-readable ASCII text followed by a carriage return character (ASCII 13, or '\r') and a newline character (ASCII 10, or '\n'). This command takes the same forms as `Serial.print()`.
Syntax	`Serial.println(val)` `Serial.println(val, format)`
Parameters	`val`: the value to print - any data type `format`: specifies the number base (for integral data types) or number of decimal places (for floating point types)
Return	`size_t: println()` returns the number of bytes written, though reading that number is optional.

Arduino

Classes – Communication - Serial.write()

Description	Writes binary data to the serial port. This data is sent as a byte or series of bytes; to send the characters representing the digits of a number use the print() function instead.

Syntax	`Serial.write(val)`
	`Serial.write(str)`
	`Serial.write(buf, len)`

Parameters	`val`: a value to send as a single byte `str`: a string to send as a series of bytes `buf`: an array to send as a series of bytes
Return	`size_t: write()` will return the number of bytes written, though reading that number is optional

Arduino

Classes – Communication - Serial.read()

Description	Reads incoming serial data. read() inherits from the Stream utility class.
Syntax	`Serial.read()`
Parameters	Nothing
Return	The first byte of incoming serial data available (or -1 if no data is available) - int.

Arduino

Classes – Communication - Serial

Want to more on Serial functions?
Click the below link

Digital I/O

Arduino
Classes – Digital I/O - pinMode()

Description	Configures the specified pin to behave either as an input or an output
Syntax	`pinMode(pin, mode)`
Parameters	`pin`: the number of the pin whose mode you wish to set `mode`: INPUT, OUTPUT, or INPUT_PULLUP. (see the (digital pins) page for a more complete description of the functionality.)
Return	Nothing
Notes and Warnings	The analog input pins can be used as digital pins, referred to as A0, A1, etc.

Arduino

Classes – Digital I/O - digitalWrite()

Description	Write a HIGH or a LOW value to a digital pin.
Syntax	`digitalWrite(pin, value)`
Parameters	`pin`: the pin number `value`: HIGH or LOW
Return	Nothing

Arduino

Classes – Digital I/O - digitalRead()

Description	Reads the value from a specified digital pin, either HIGH or LOW.
Syntax	`digitalRead(pin)`
Parameters	`pin`: the number of the digital pin you want to read
Return	`HIGH` or `LOW`

Time

Classes – Time - delay()

Description	Pauses the program for the amount of time (in milliseconds) specified as parameter. (There are 1000 milliseconds in a second.)
Syntax	`delay(ms)`
Parameters	`ms`: the number of milliseconds to pause (unsigned long)
Return	Nothing

Arduino

Classes – Time - delayMicroseconds()

Description	Pauses the program for the amount of time (in microseconds) specified as parameter. There are a thousand microseconds in a millisecond, and a million microseconds in a second. Currently, the largest value that will produce an accurate delay is 16383
Syntax	`delayMicroseconds(us)`
Parameters	`us`: the number of microseconds to pause (unsigned int)
Return	Nothing

Arduino

Classes – Time - micros()

Description	Returns the number of microseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 70 minutes.
Syntax	`time = micros()`
Parameters	Nothing
Return	Returns the number of microseconds since the Arduino board began running the current program.(unsigned long)

Arduino

Classes – Time - millis()

Description	Returns the number of milliseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days.
Syntax	`time = millis()`
Parameters	Nothing
Return	Number of milliseconds since the program started (unsigned long)
Notes and Warnings	Please note that the return value for millis() is an unsigned long, logic errors may occur if a programmer tries to do arithmetic with smaller data types such as int's. Even signed long may encounter errors as its maximum value is half that of its unsigned counterpart.

Analog I/O

Arduino

Classes – Analog I/O - analogRead()

Description	Reads the value from the specified analog pin. The number of channels depends on the board used, assuming 10-bit analog to digital converter, mapping voltages between 0 and 5 volts into integer values between 0 and 1023. This yields a resolution between readings of: 5 volts / 1024 units or, .0049 volts (4.9 mV) per unit. The input range and resolution can be changed using analogReference(). It takes about 100 microseconds to read an analog input,
Syntax	`val = analogRead(pin)`
Parameters	`pin`: the number of the analog input pin to read from
Return	int (0 to 1023) Depends on the board
Notes and Warnings	If the analog input pin is not connected to anything, the value returned by analogRead() will fluctuate based on a number of factors (e.g. the values of the other analog inputs, how close your hand is to the board, etc.).

Arduino

Classes – Analog I/O - analogWrite()

Description	Writes an analog value (PWM wave) to a pin. Can be used to light a LED at varying brightnesses or drive a motor at various speeds
Syntax	`analogWrite(pin, value)`
Parameters	`pin`: the pin to write to. Allowed data types: int. `value`: the duty cycle: between 0 (always off) and 255 (always on). Allowed data types: int
Return	Nothing
Notes and Warnings	Please click on the below link icon for more info

Arduino

Classes – Analog I/O - analogWrite()

Description	Configures the reference voltage used for analog input (i.e. the value used as the top of the input range)
Syntax	`analogReference(type)`
Parameters	`type`: which type of reference to use (see list of options in the description).
Return	Nothing
Notes and Warnings	Please click on the below link icon for more info

Thank You

Arduino

Peripherals and Interfaces

Team Emertxe

Interfaces

Arduino

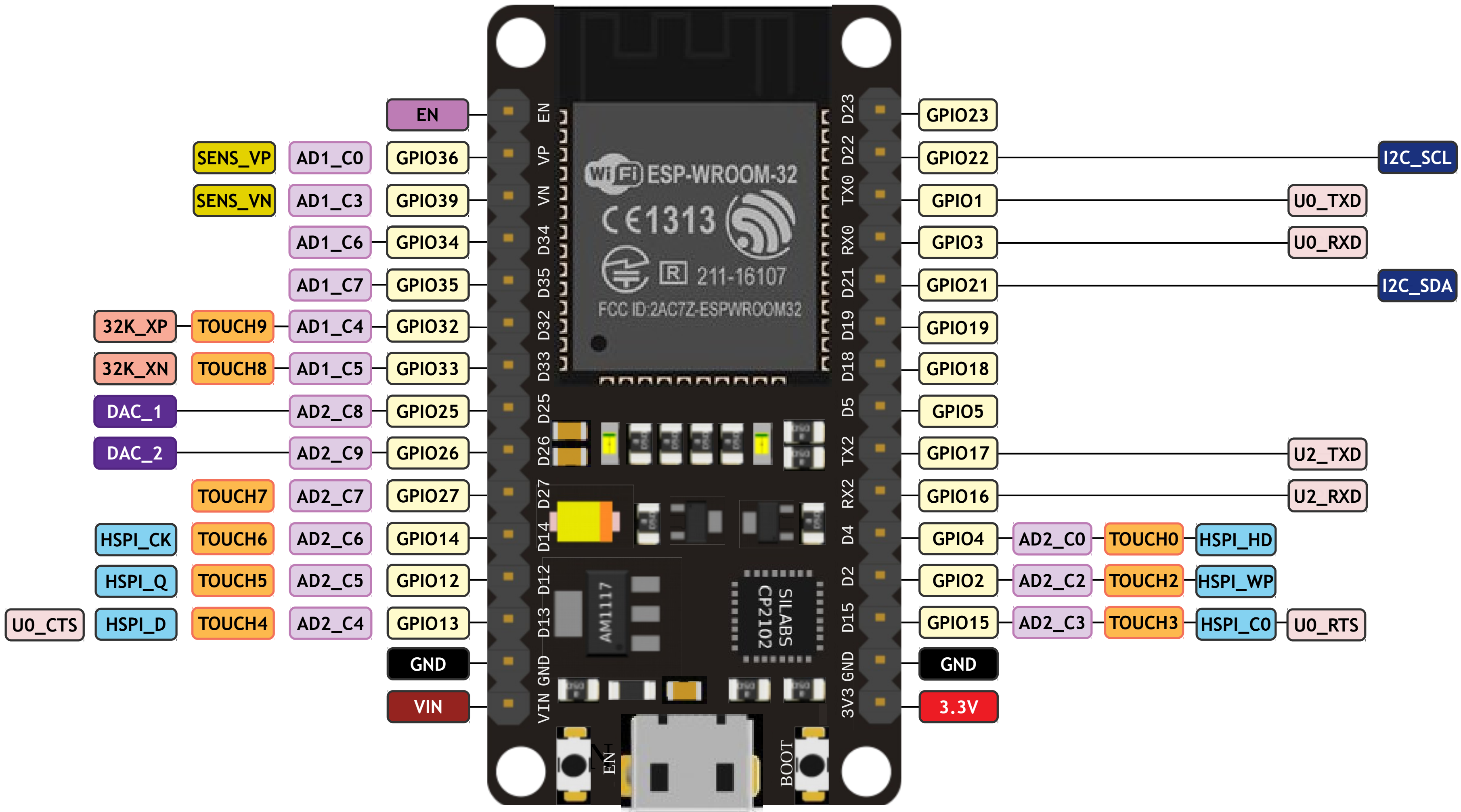
Interface – What?

A shared boundary across which two or more separate components of a computer system exchange information

Source: wiki

Arduino

Interface - Pinout



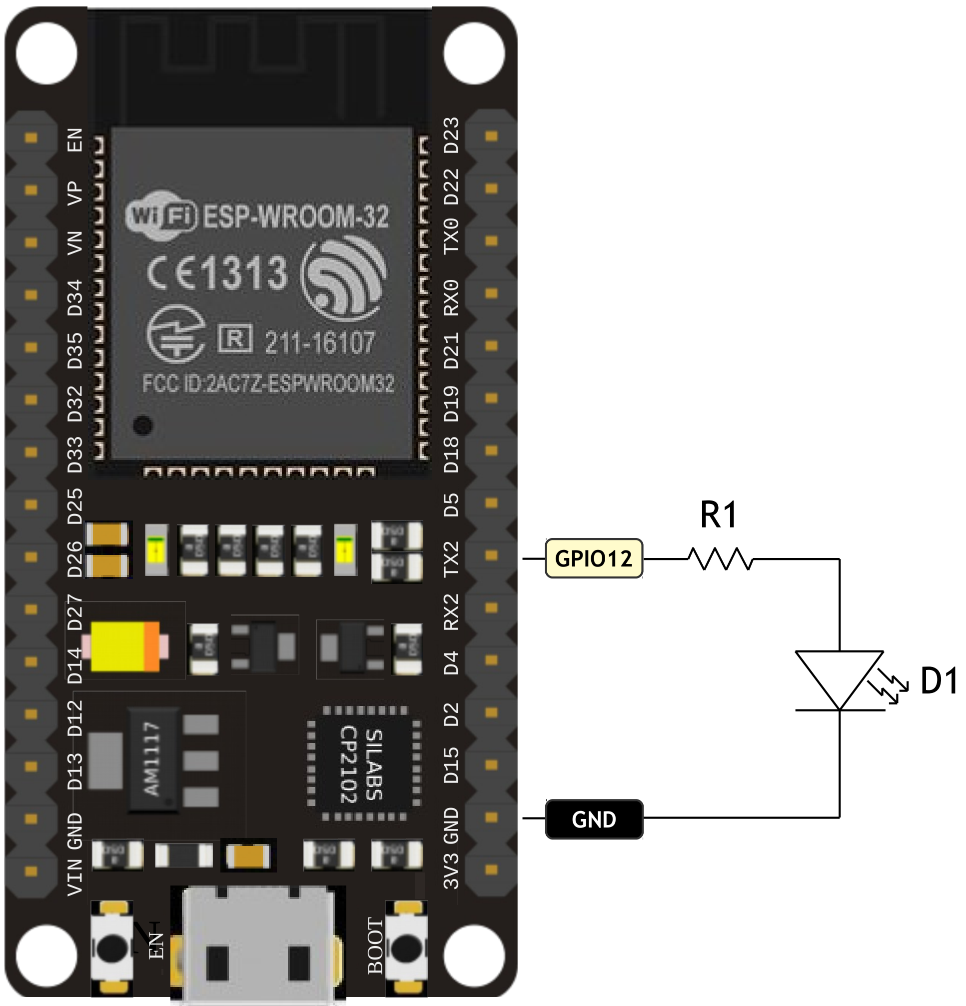
EMERTXE

Light Emitting Diodes

Arduino
Interface - LEDs

- Simplest device used in most on the embedded applications as feedback
- Works just like diodes
- Low energy consumption, longer life, smaller size, faster switching make it usable in wide application fields like
  - Home lighting,
  - Remote Controls, Surveillance,
  - Displays and many more!!

Arduino
Interface - LEDs



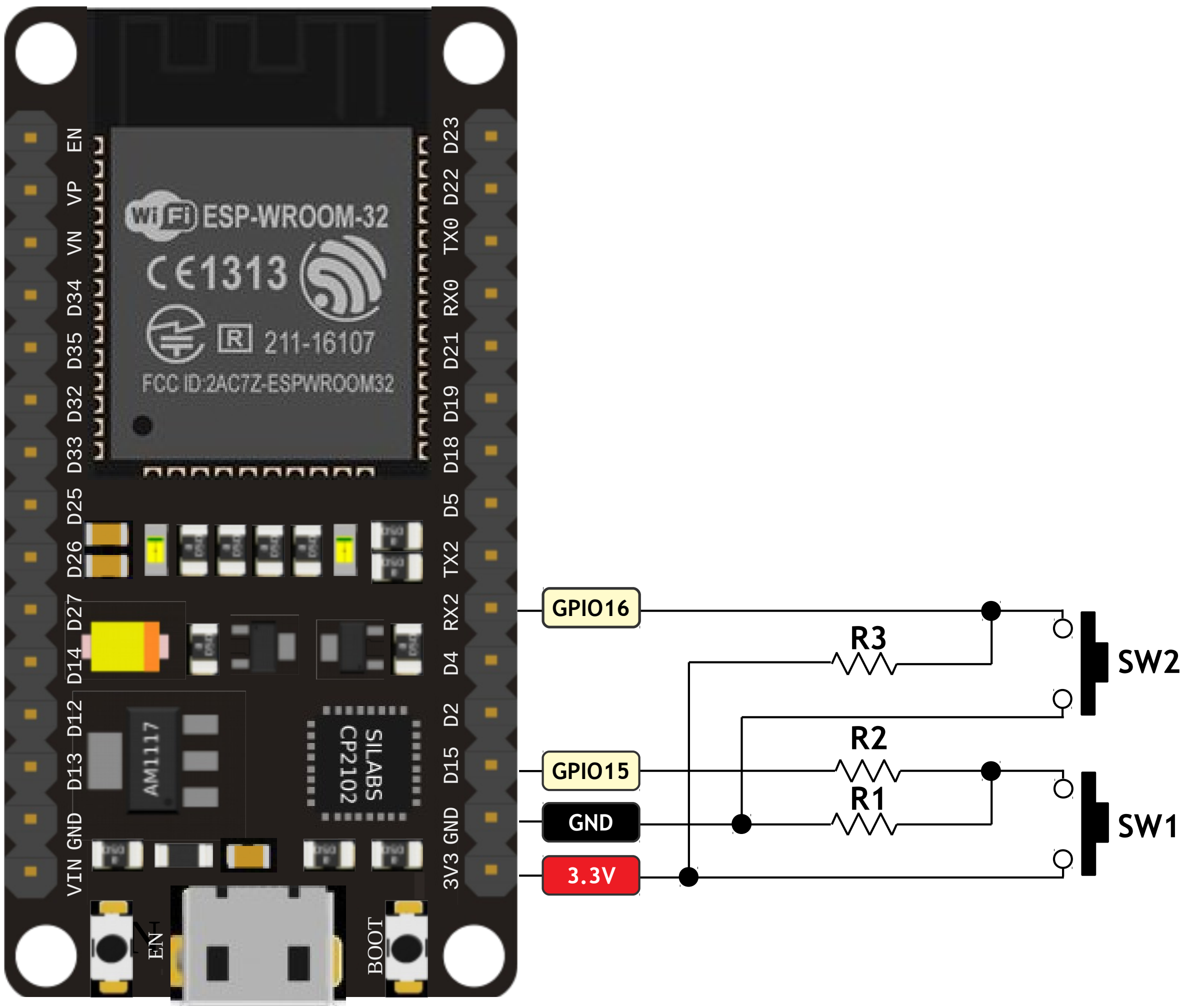
Tactile Switch

Arduino

Interface - Tactile Switches

- Provides simple and cheap interface
- Comes in different shapes and sizes
- Preferable if the no of user inputs are less
- Some common application of tactile keys are
  - HMI
  - Mobile Phones
  - Computer Mouse etc,.

Arduino
Interface - Tactile Switches



Analog Input

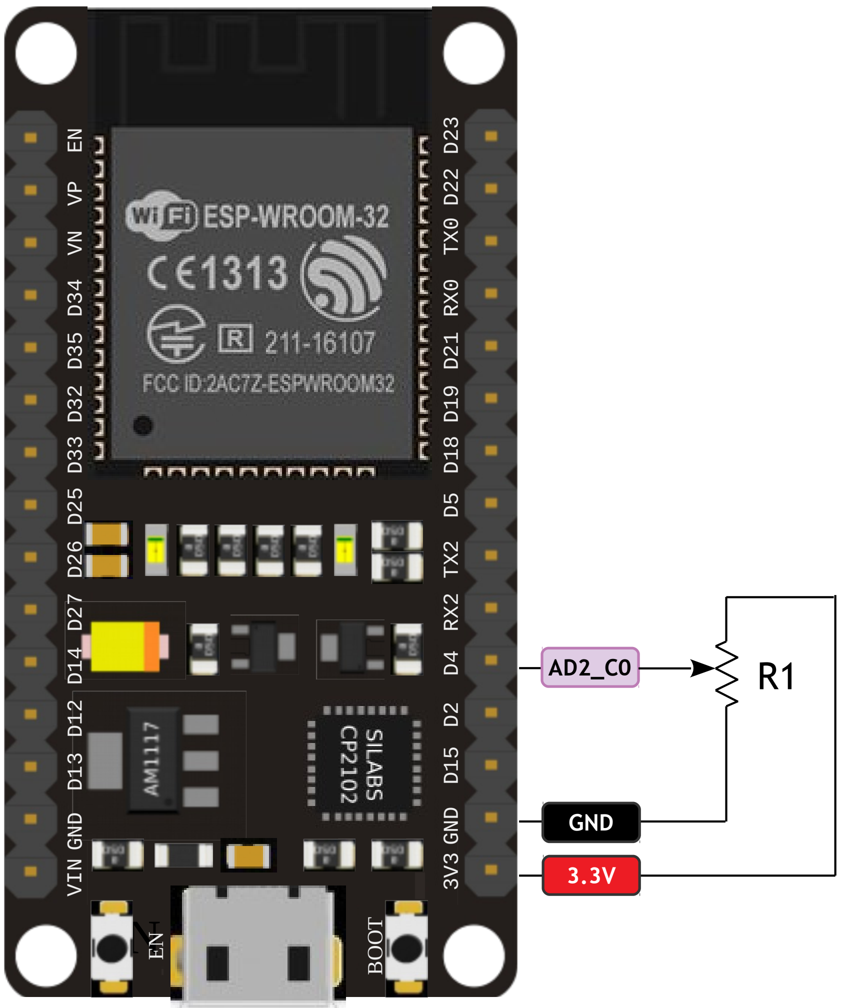
Arduino

Interface – Analog Inputs

- Very important peripheral in embedded systems for real time activities
- The controller understands only digital signals, so an real time linear signals have to be converted into digital form
- Multiplexed with GPIO
- Comes with different architecture, SAR is most commonly used

Arduino

Interface - Analog Inputs - Potentiometer



Interrupts

Arduino

Peripheral – Interrupt - Contents

- Basic Concepts
- Interrupt Source
- Interrupt Classification
- Interrupt Handling

Arduino

Peripheral – Interrupt – Basic Concept

- An interrupt is a communication process set up in a microprocessor or microcontroller in which:
  - An internal or external device requests the MPU to stop the processing
  - The MPU acknowledges the request
  - Attends to the request
  - Goes back to processing where it was interrupted
- Polling

Arduino

Peripheral – Interrupt – Vs Polling

- Events Detection
- Response
- Power Management

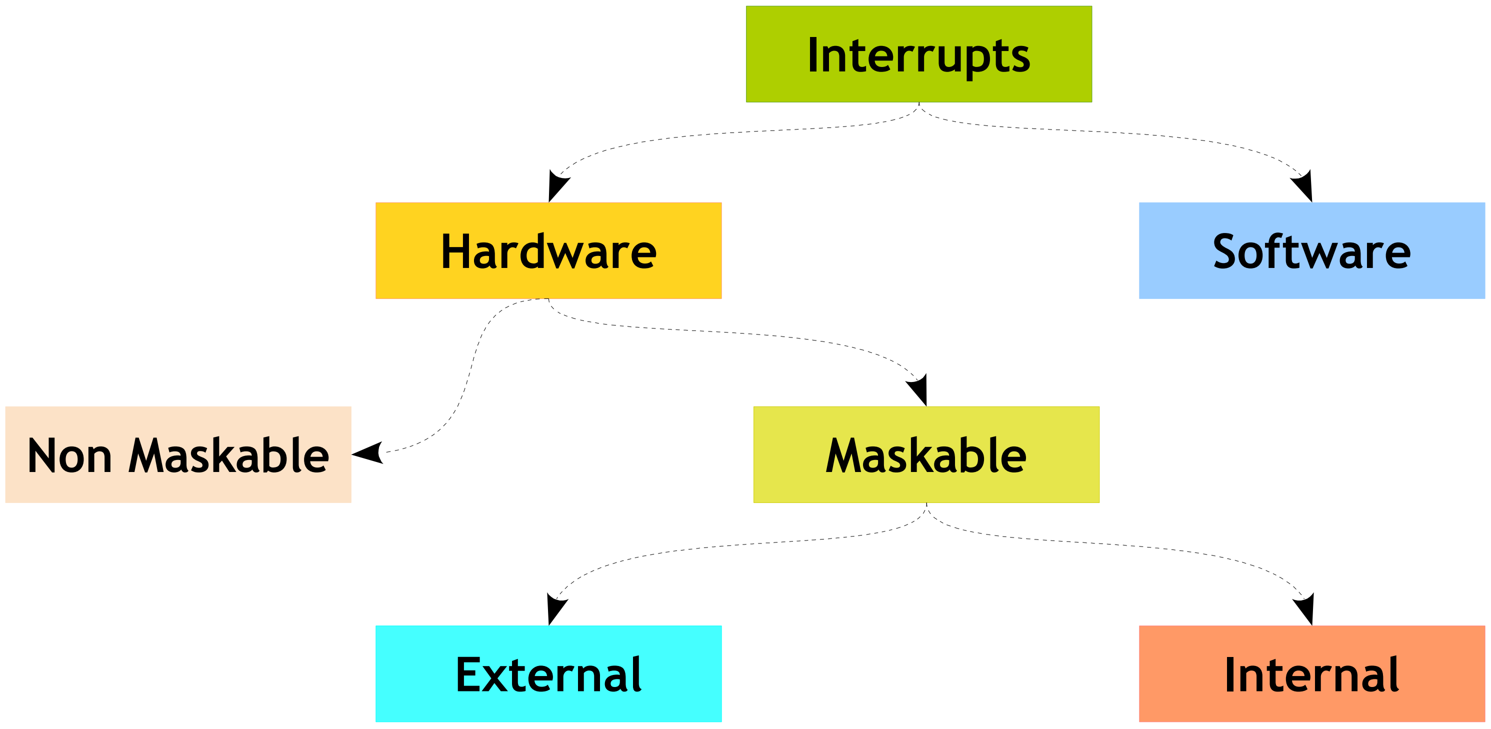
Arduino

Peripheral – Interrupt – Sources

- Timers
- External
- Peripherals

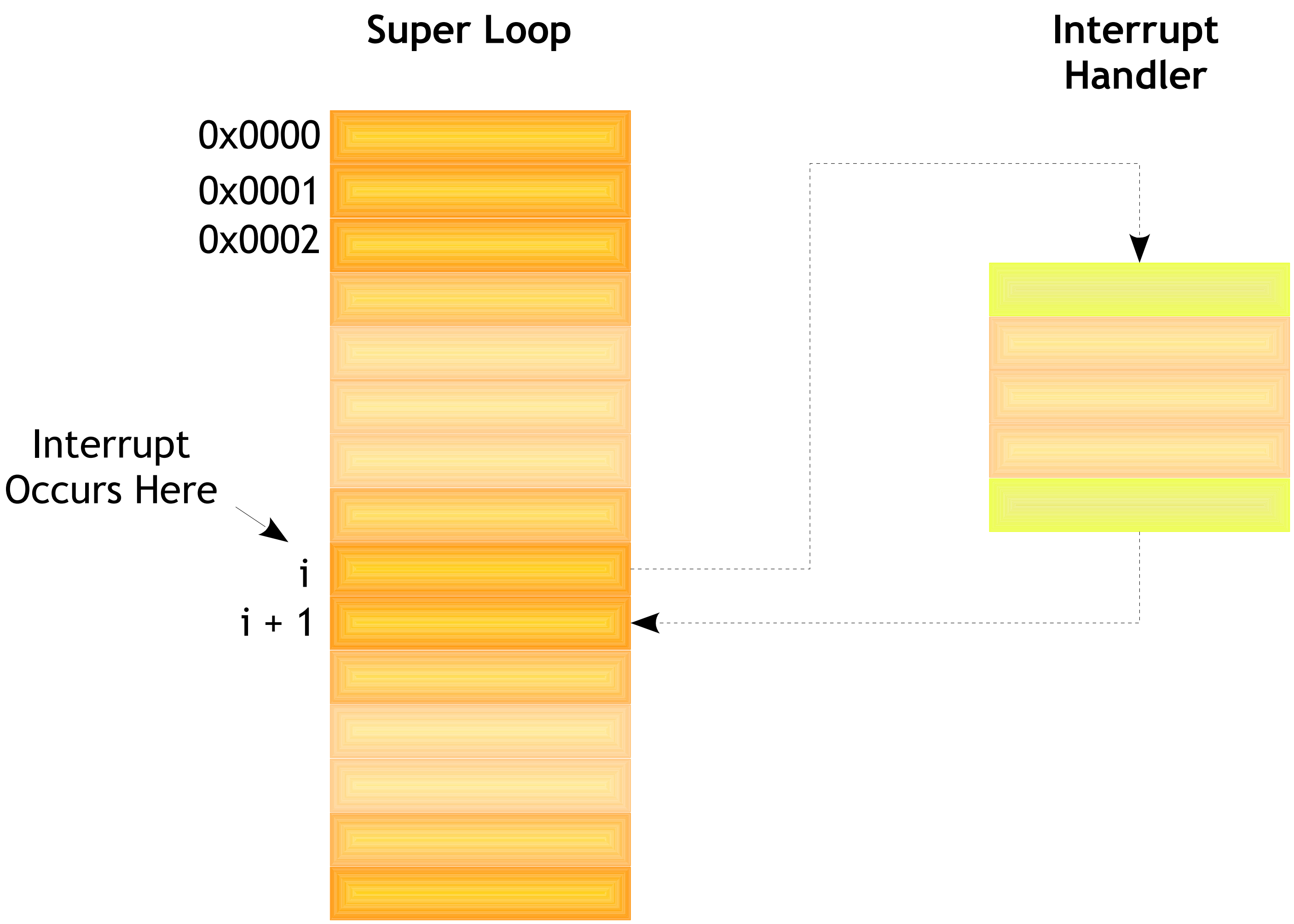
Arduino

Peripheral – Interrupt – Classifications



Arduino

Peripheral – Interrupt – Handling



Arduino

Peripheral – Interrupt – Handling - ISR

- Similar to a subroutine
- When an interrupt occurs, the MPU:
  - Completes the instruction being executed
  - Disables global interrupt enable
  - Places the address from the program counter on the stack
- Attends to the request of an interrupting source
  - Clears the interrupt flag
  - Should save register contents that may be affected by the code in the ISR
  - Must be terminated with the instruction RETFIE
- Return from interrupt

Arduino

Peripheral – Interrupt – Handling - ISR

- What / What Not

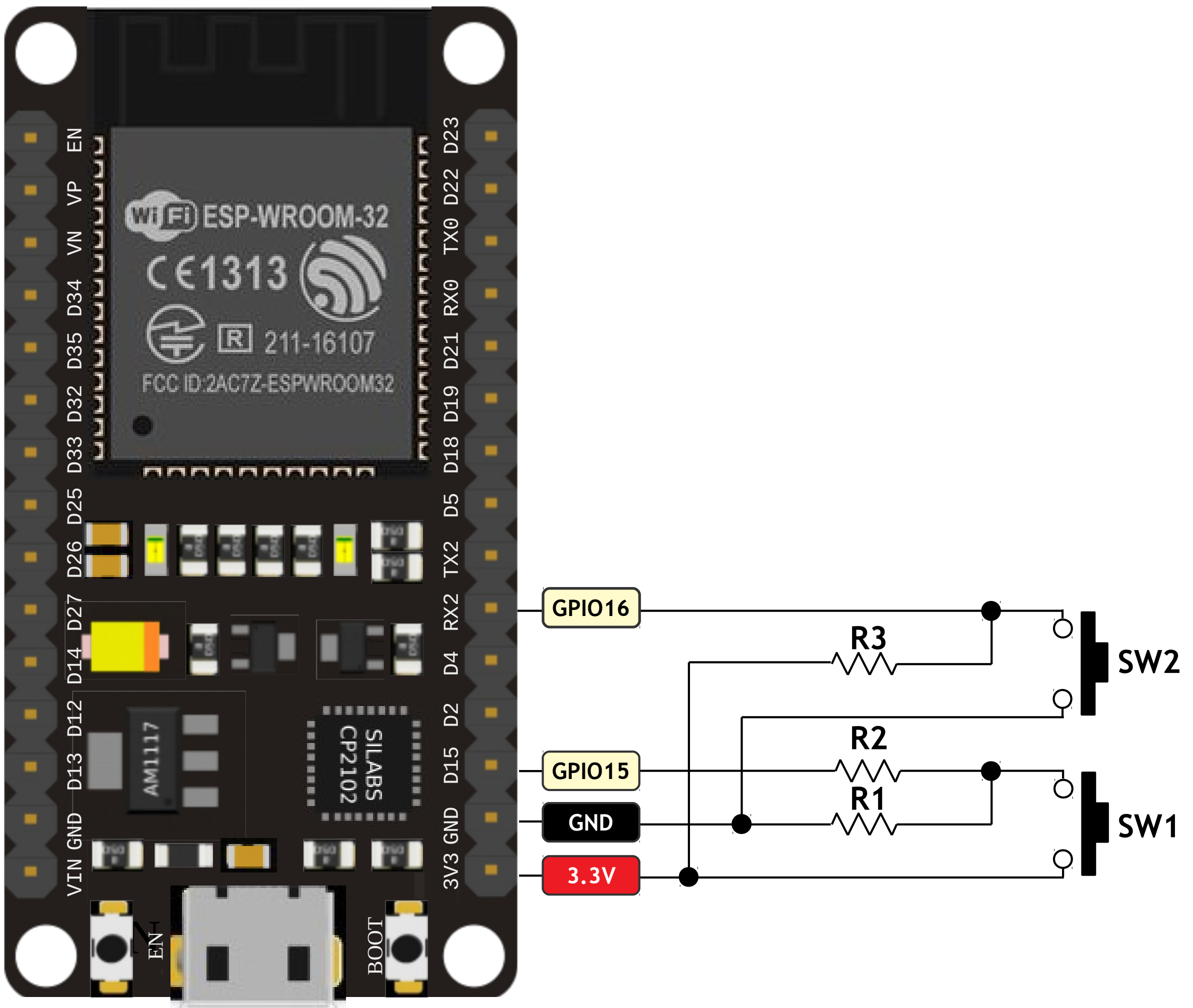
Arduino

Peripheral – Interrupt – Handling - ISR

- Latency is determined by:
  - Instruction time (how long is the longest)
  - How much of the context must be saved
  - How much of the context must be restored
  - The effort to implement priority scheme
  - Time spend executing protected code

Arduino

Peripheral – Interrupt – Interface



Timers

Arduino

Peripherals - Timers

- Resolution → Register Width
- Tick → Up Count or Down Count
- Quantum → System Clock settings
- Scaling → Pre or Post
- Modes
  - Counter
  - PWM or Pulse Generator
  - PW or PP Measurement etc.,

Arduino

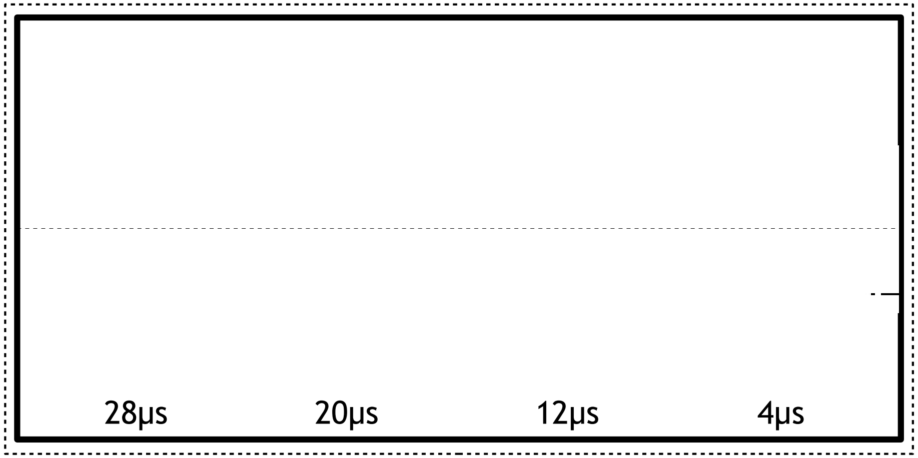
Peripherals – Timers - Example

- Requirement – 5 pulses of 8 µsecs
- Resolution – 8 Bit
- Quantum – 1 µsecs
- General

Arduino
Peripherals – Timers - Example

Timer Register: 252

Overflows: 0



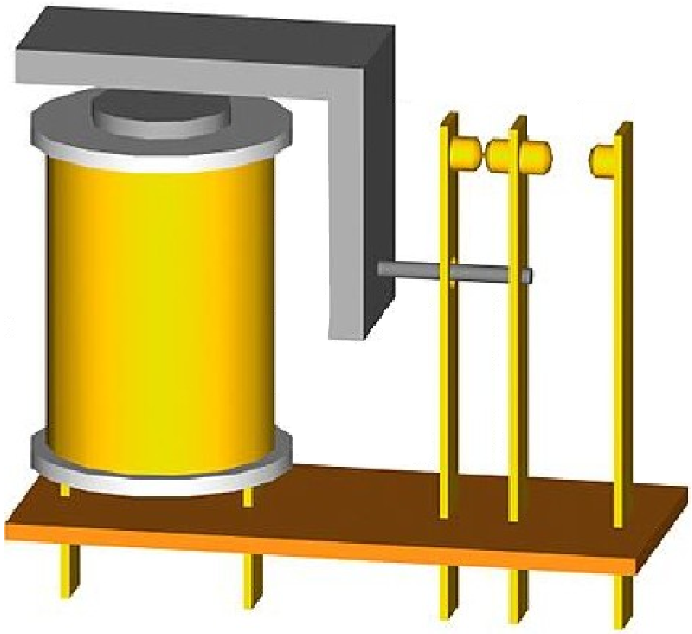
Relay

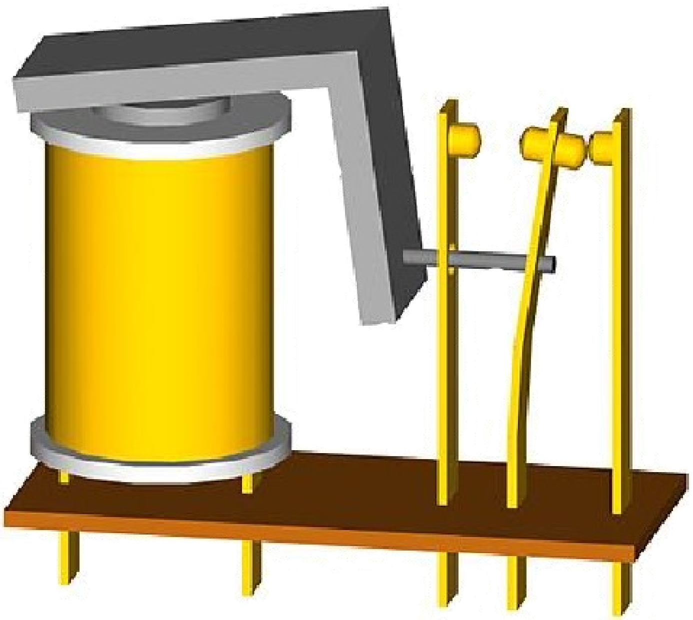
Arduino

Interface - Relay

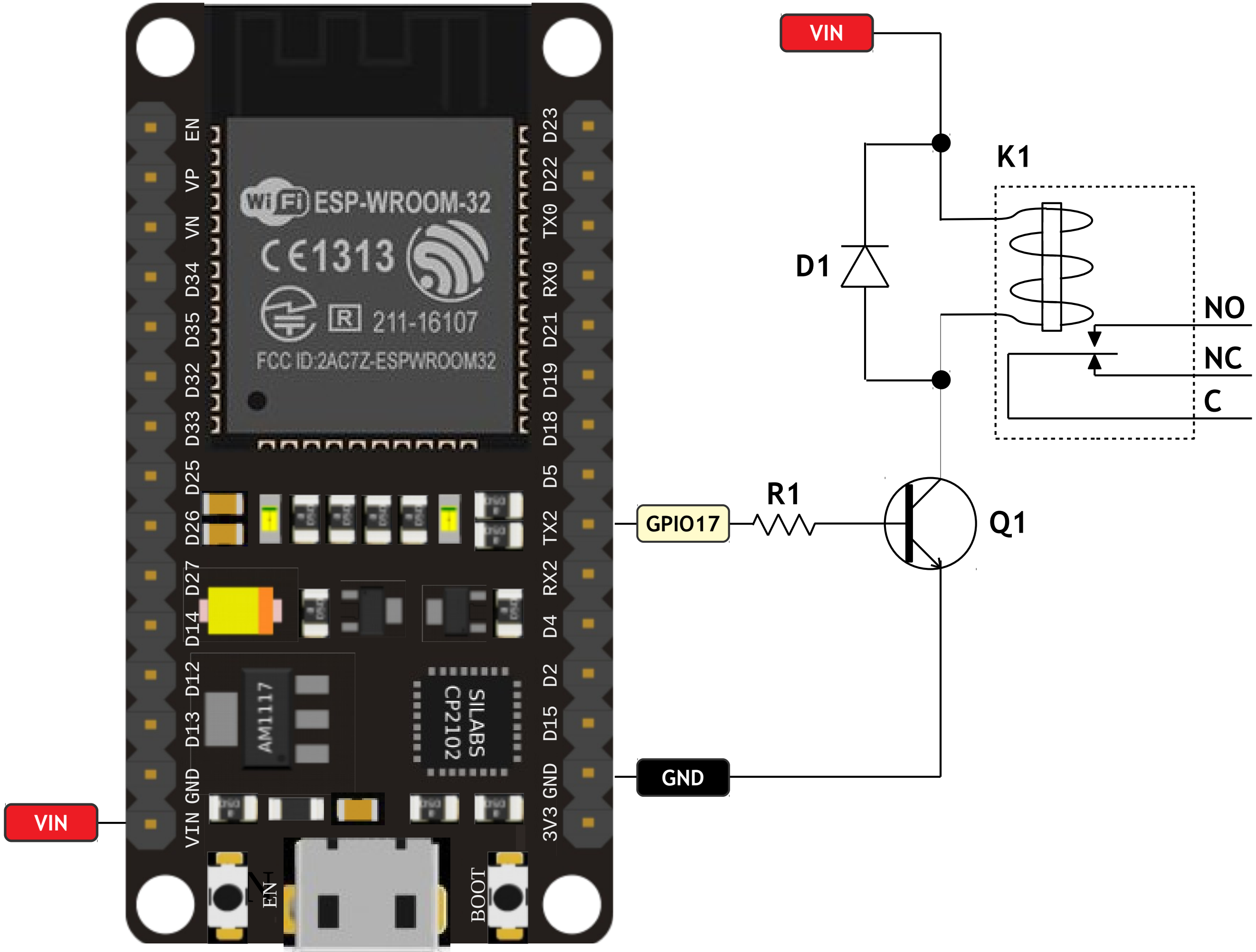
- Most commonly used electromechanical switch
- Uses a electromagnet to operate
- Used to control high power devices using low power signal
- Provides isolation between the control and controlled circuit
  - Home Automation,
  - Automotive applications
  - Industrial application and many more !!

Arduino
Interface - Relay





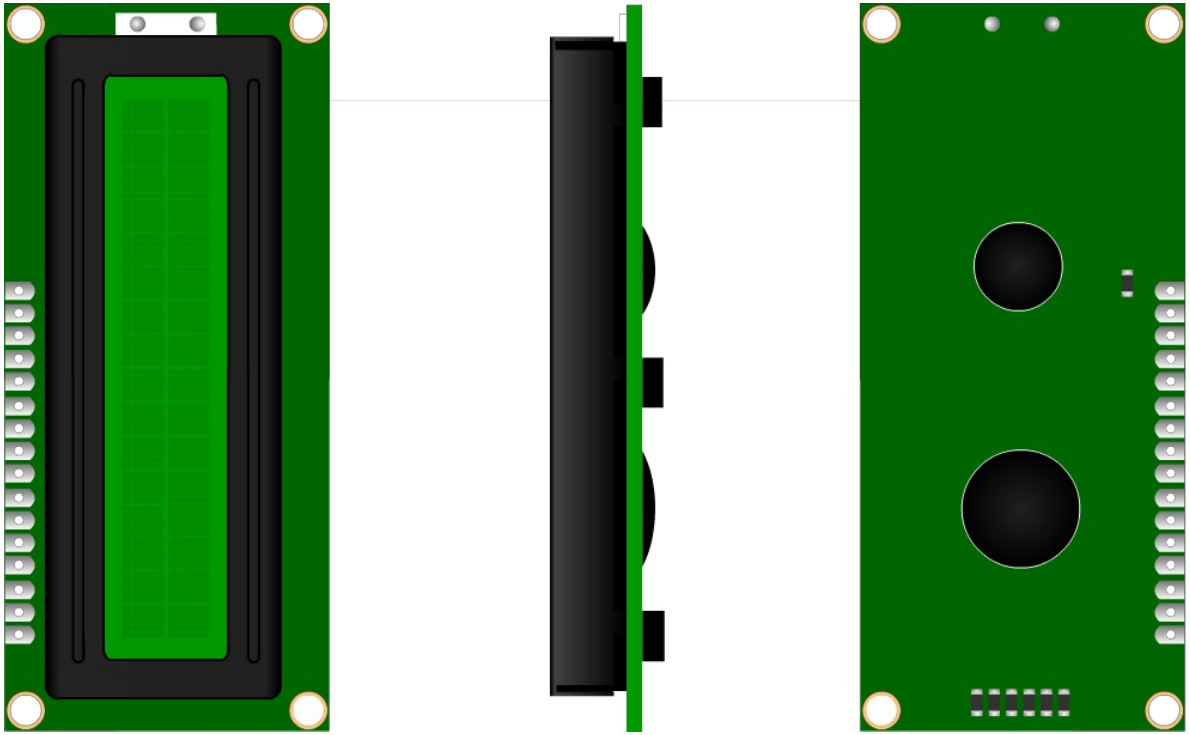
Arduino
Interface - Relay



CLCD

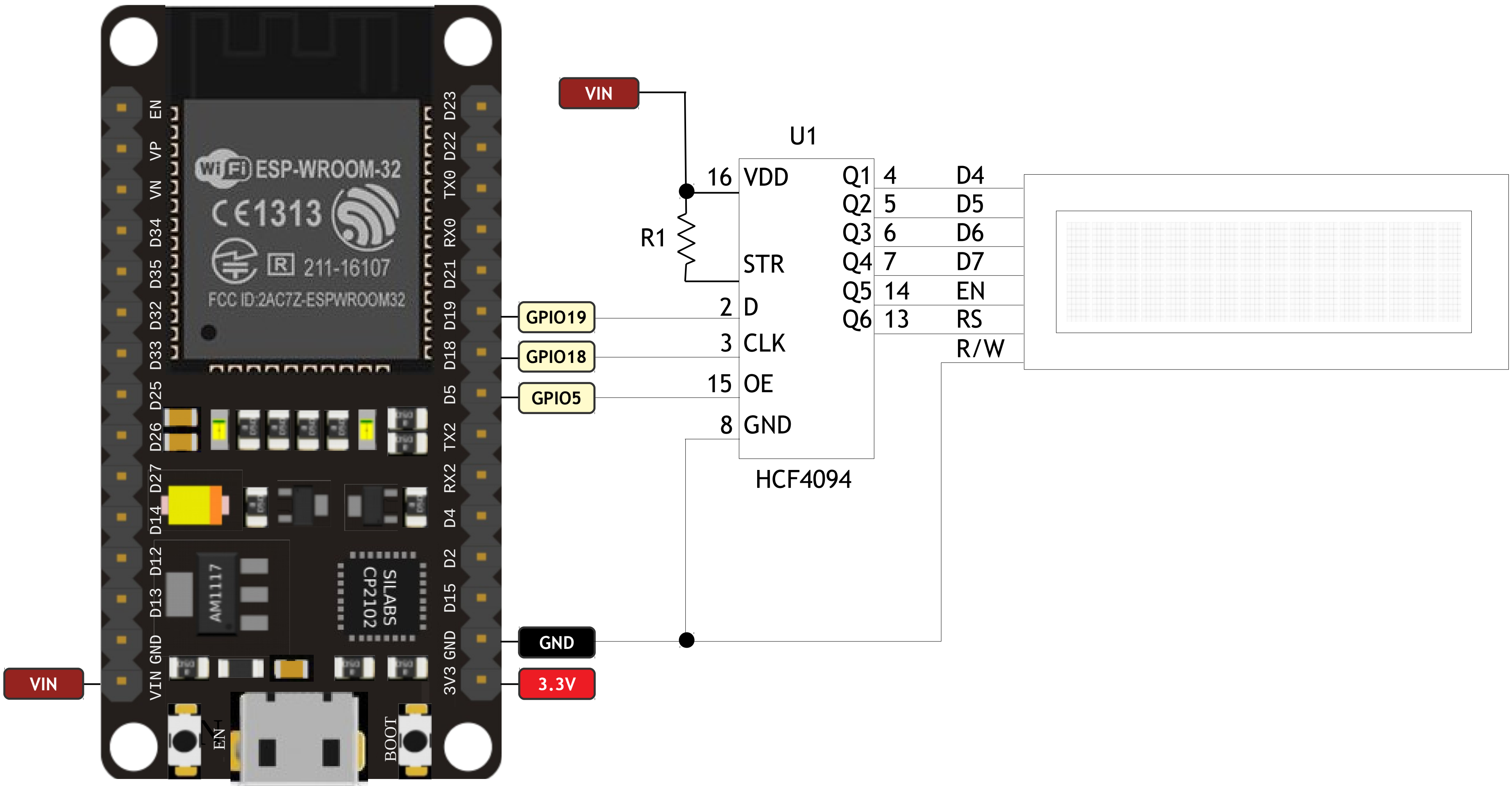
Arduino
Interface - CLCD

- Most commonly used display ASCII characters
- Some customization in symbols possible
- Communication Modes
  - 8 Bit Mode
  - 4 Bit Mode



Arduino

Interface - CLCD



Sensors

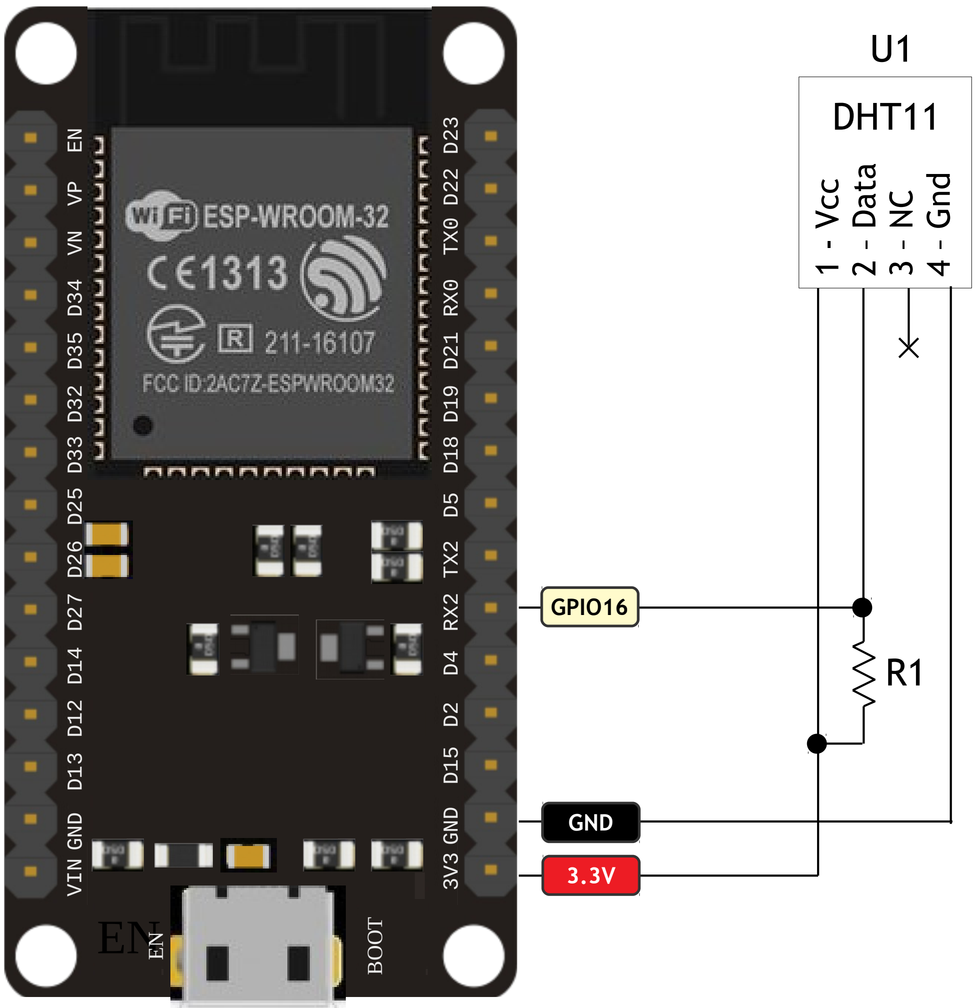
DHT11

Arduino

Interface – Sensors - DHT11

- A cheap and very simple sensor to measure Temperature and Humidity
- It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and sends digital signal on the data pin

Arduino
Interface - Sensors - DHT11



Thank You

Communication Protocols I

Wired

Team Emertxe

Communication Protocols I

- Introduction
- UART
- SPI
- I^2C
- CAN

Introduction

Introduction

- What do mean by Communication?
- Mode of Communications
- Type of Communications
- Why Protocols?

UART

UART

- Introduction
- Interface
- Hardware Configurations
- Frame Format

UART

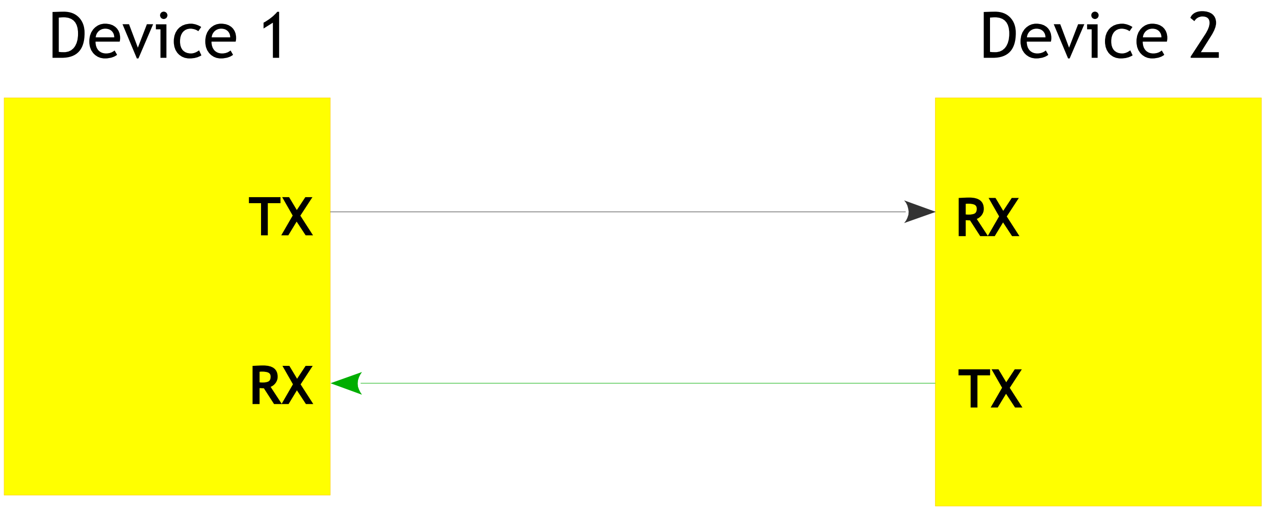
Introduction

- Asynchronous
- Duplex - Any
- Master / Slave

UART
Interface

- RX
- TX

UART
Hardware Configuration



UART

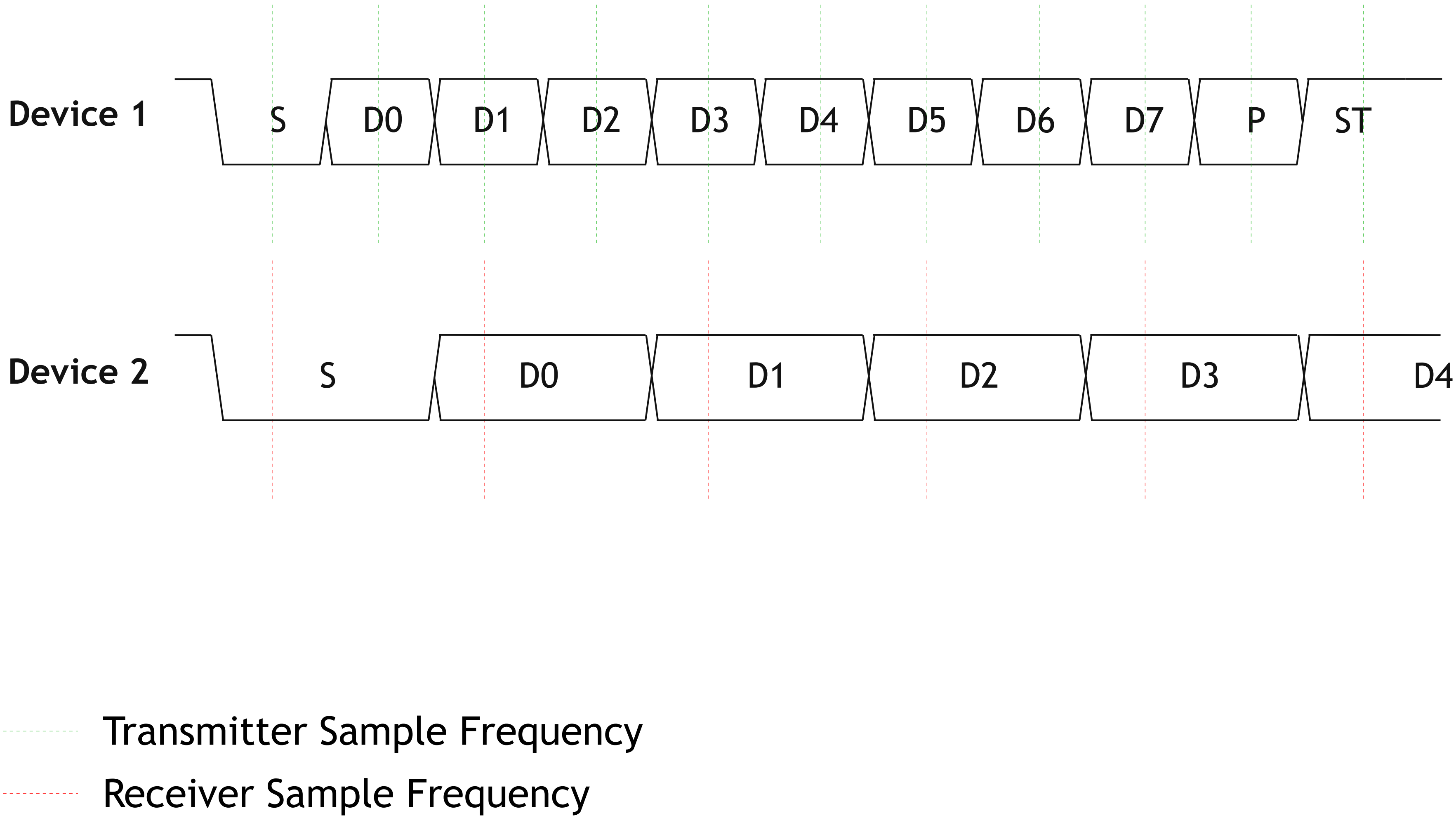
Frame Format

- Data part can be 5 to 9 bits
- Stop could be 2 bits
- Parity could be 0 or 1 bit

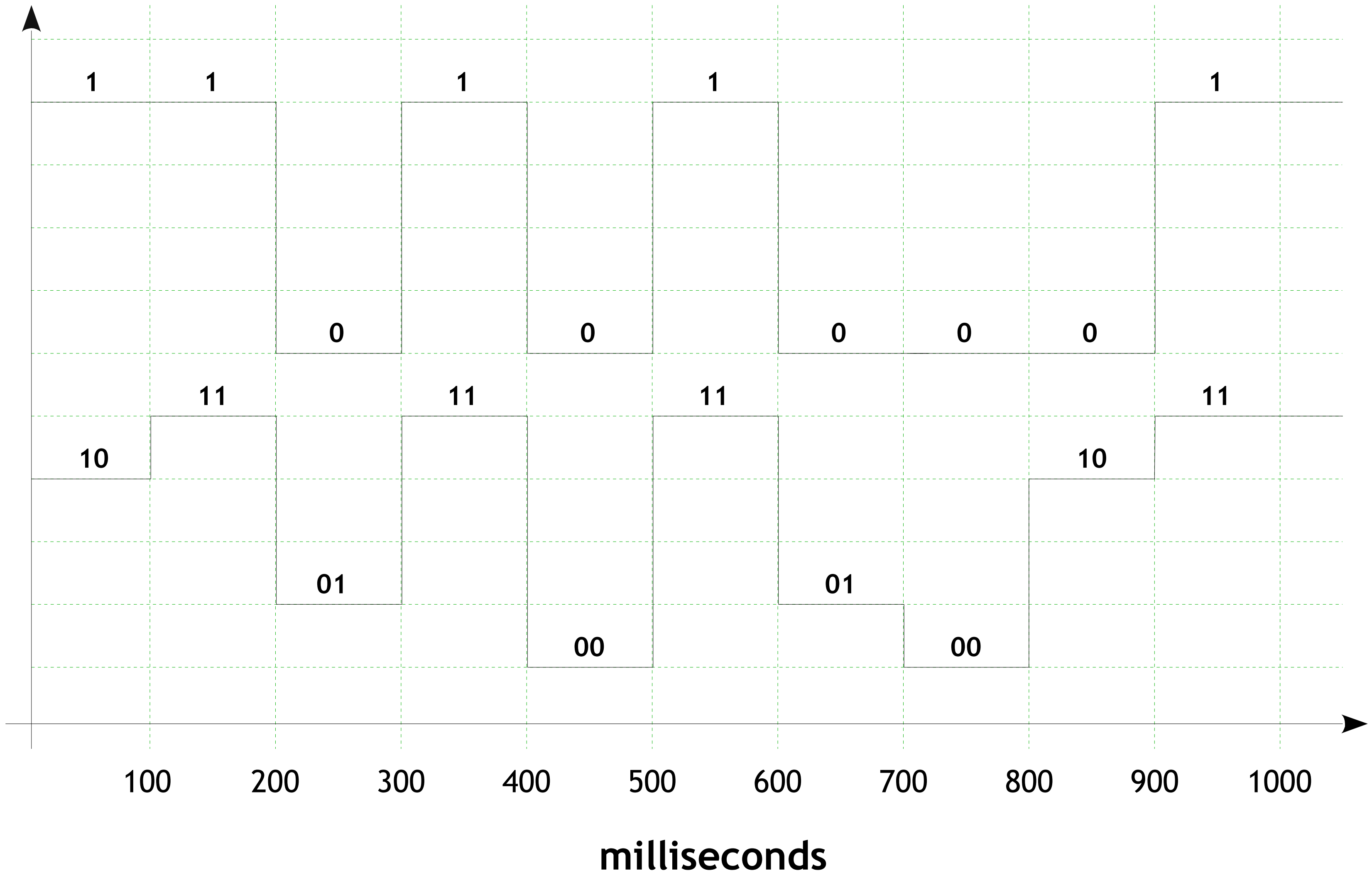
UART
Baud Rate

- Number of symbols per second (In this context the a symbol is a bit)
- So, sometimes referred as Bit Rate (No of bits per second)
- The frequency of the data transfer
- Both transmitter and receiver has to agree upon a common frequency for data integrity

UART
Baud Rate



UART
Baud Rate vs Bit Rate



Serial Peripheral Interface

Serial Peripheral Interface

- Introduction
- Interface
- Hardware Configurations
- Data Transmission
  - Data Validity

SPI
Introduction

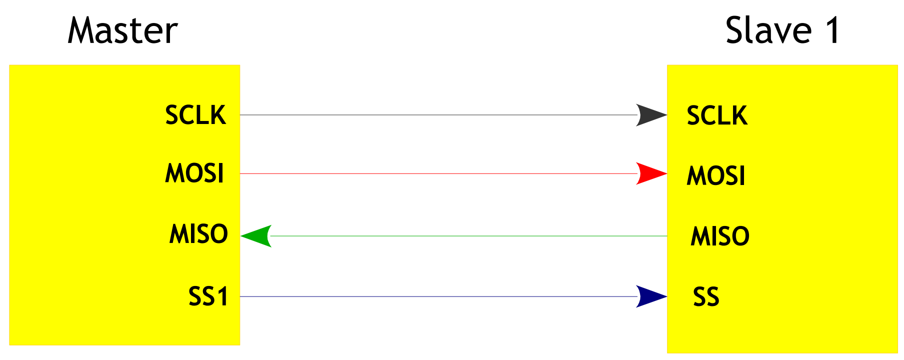
- Synchronous
- Full Duplex
- Master / Slave

SPI
Interface

- SCLK
- MOSI
- MISO
- nSS

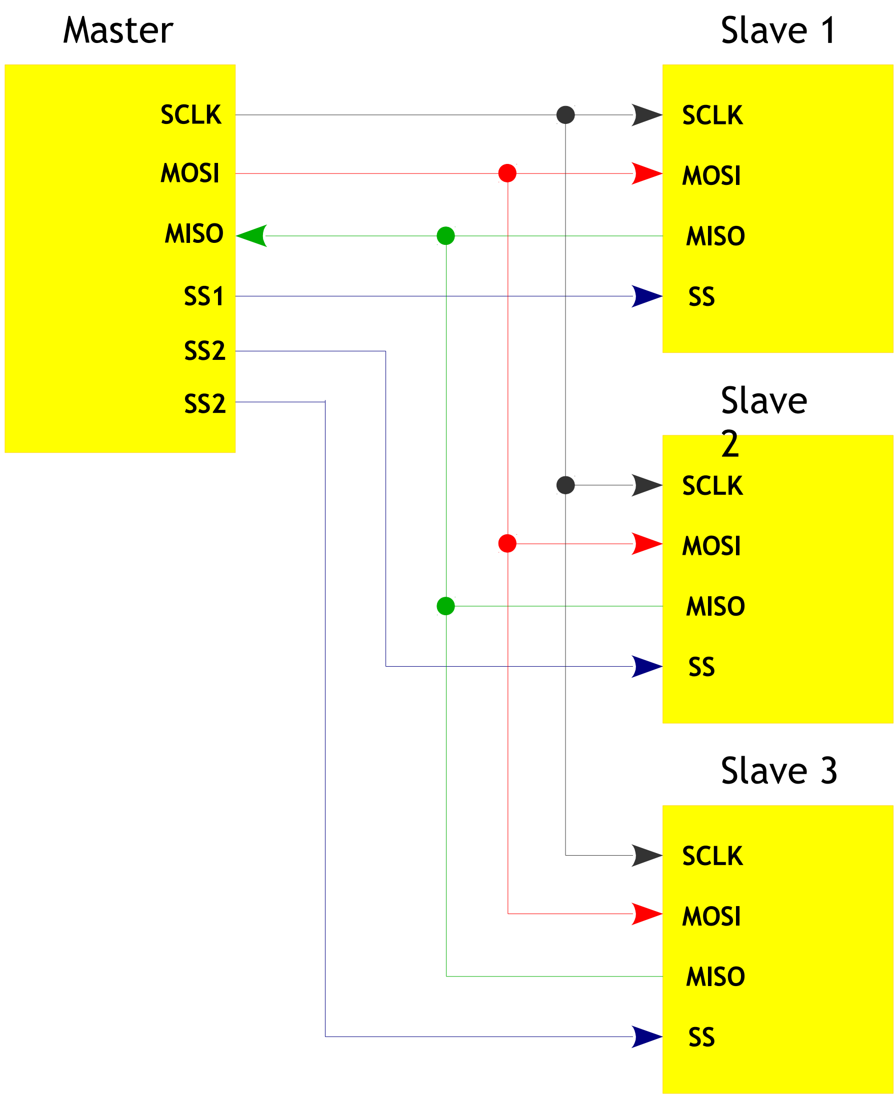
SPI

Hardware Configuration



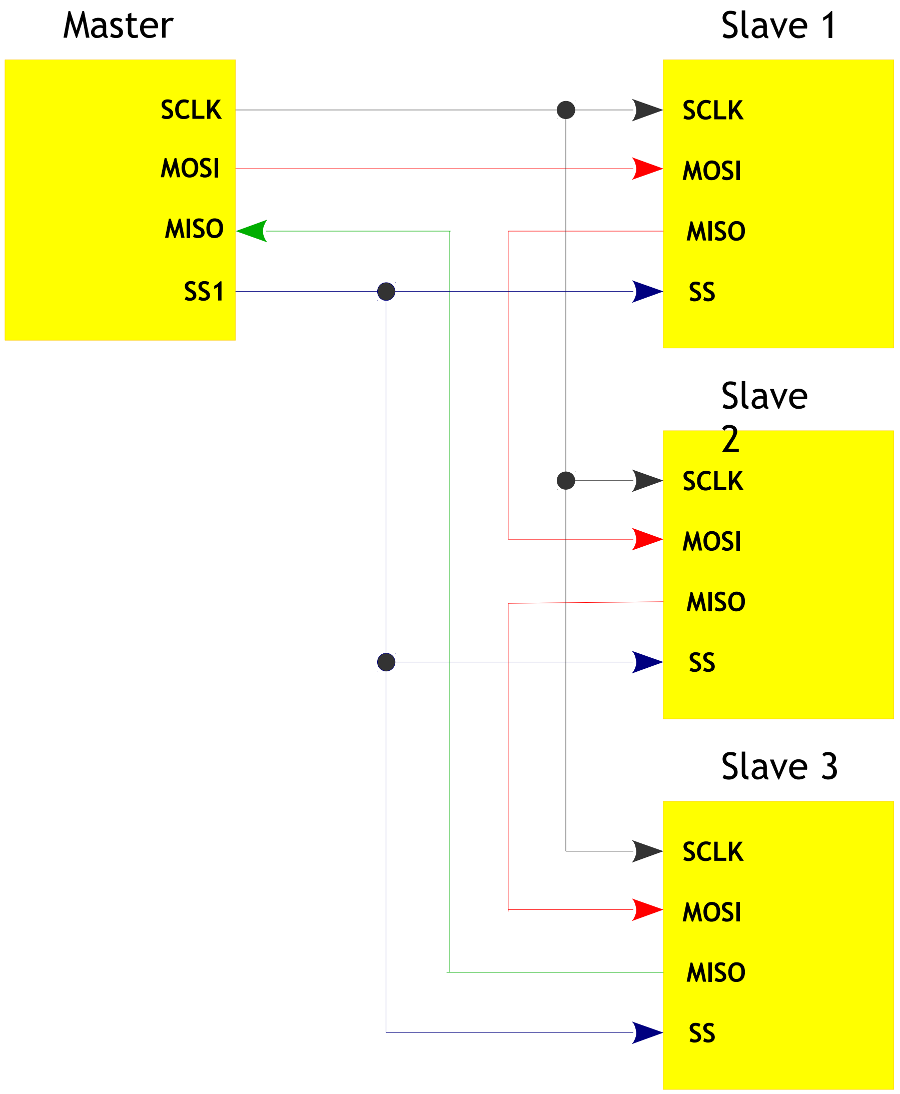
Single Master and Single Slave

SPI
Hardware Configuration



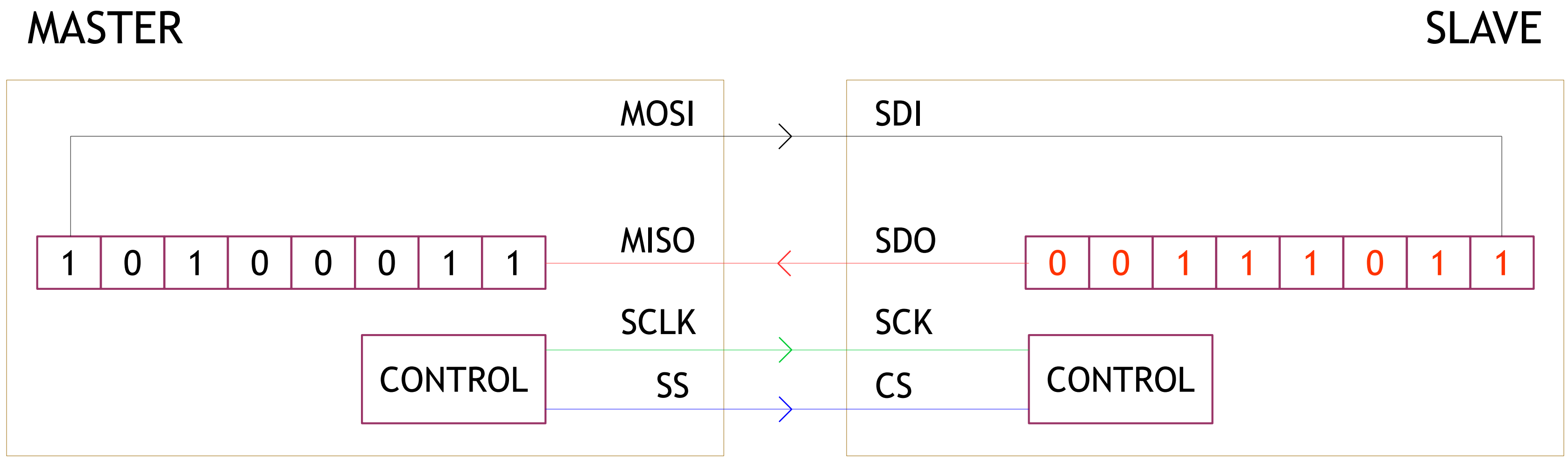
Single Master and Three Slaves

SPI
Hardware Configuration

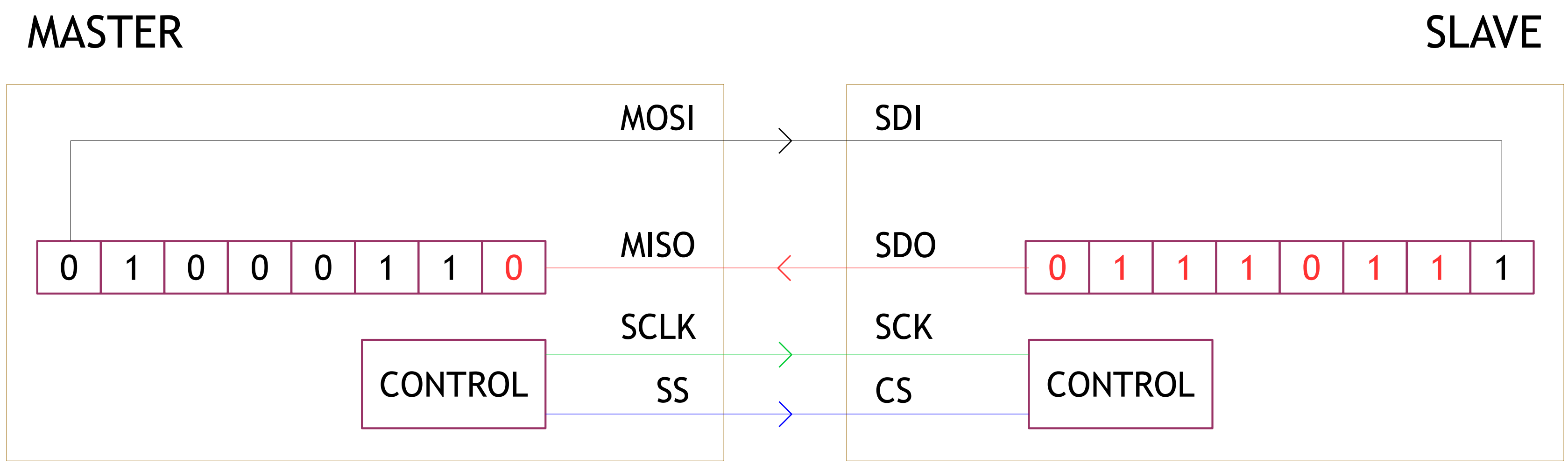


Single Master and Three Daisy Chained Slaves

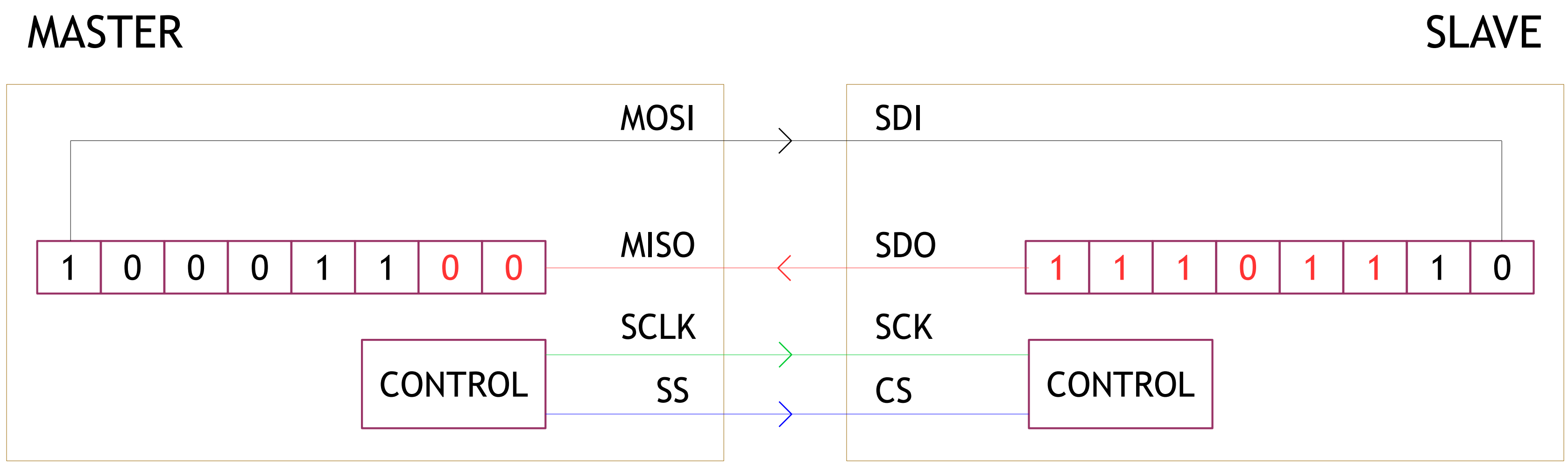
SPI
Data Transmission



SPI
Data Transmission

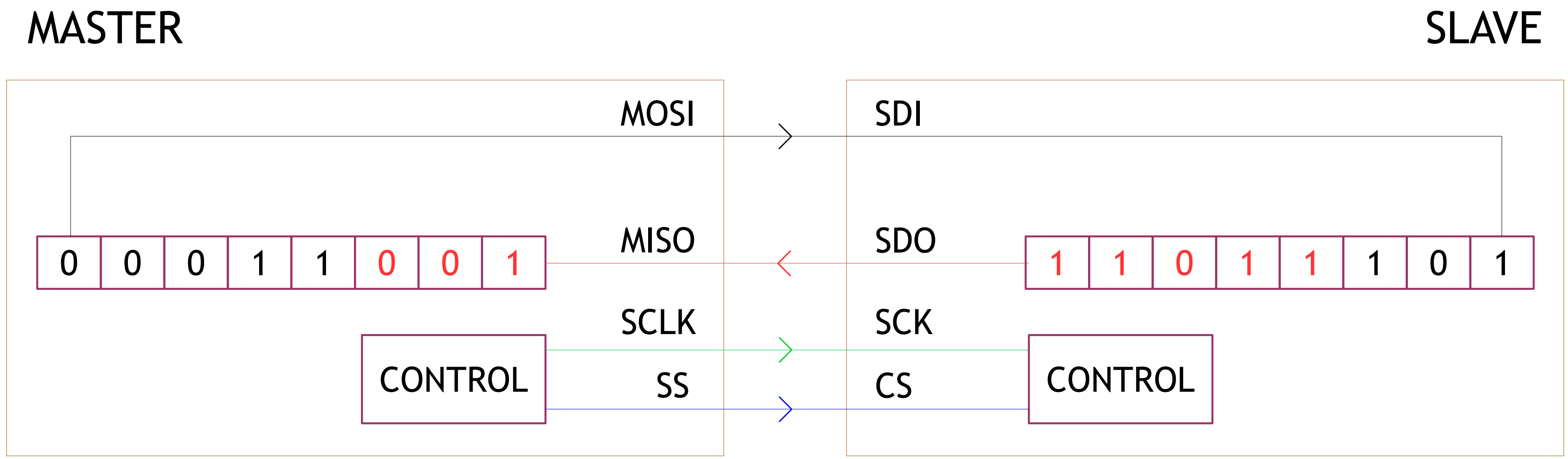


SPI
Data Transmission

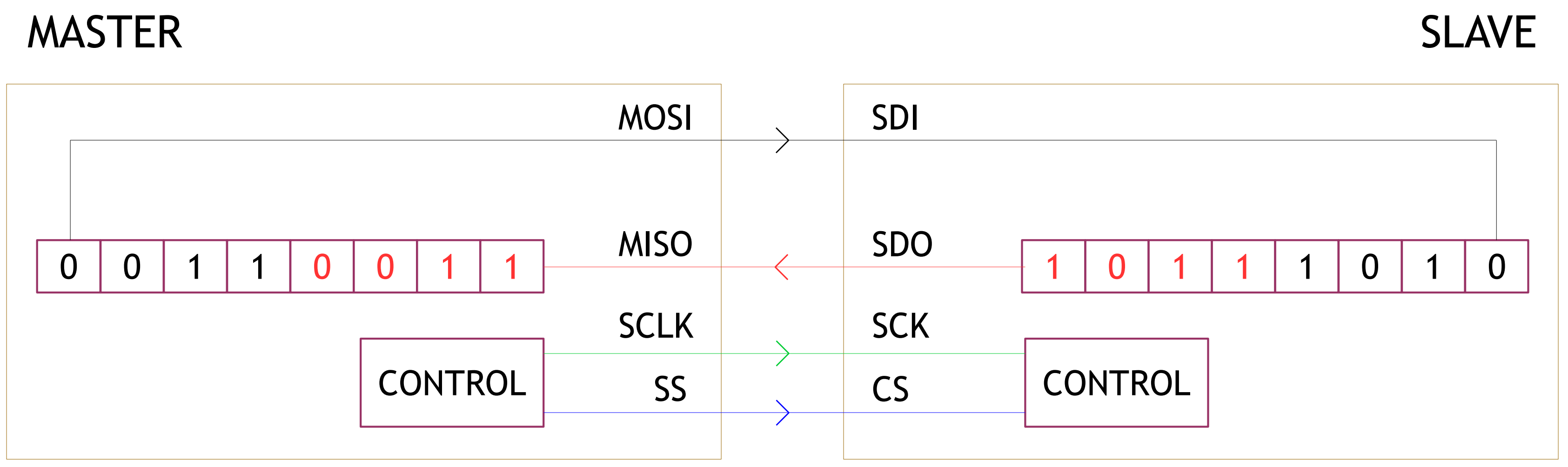


SPI

Data Transmission

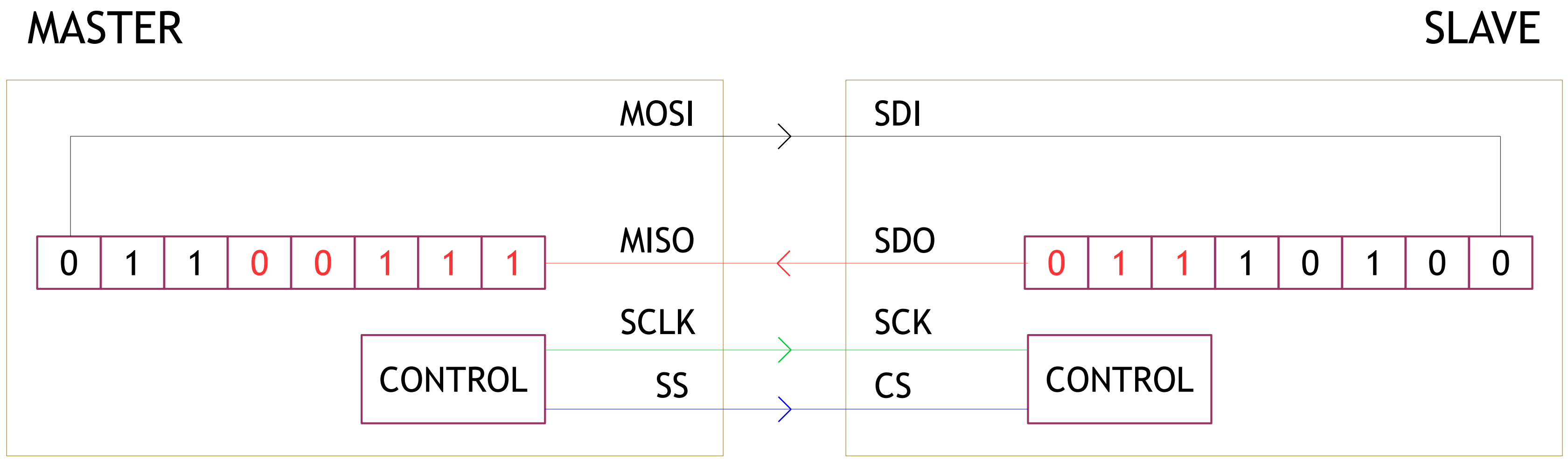


SPI
Data Transmission



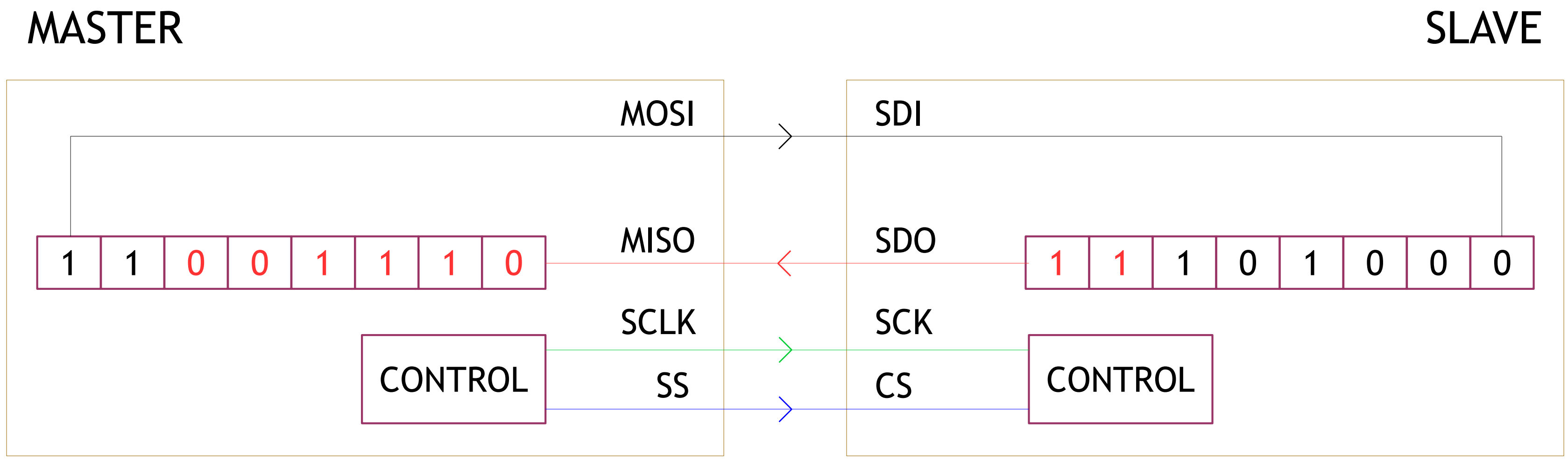
SPI

Data Transmission

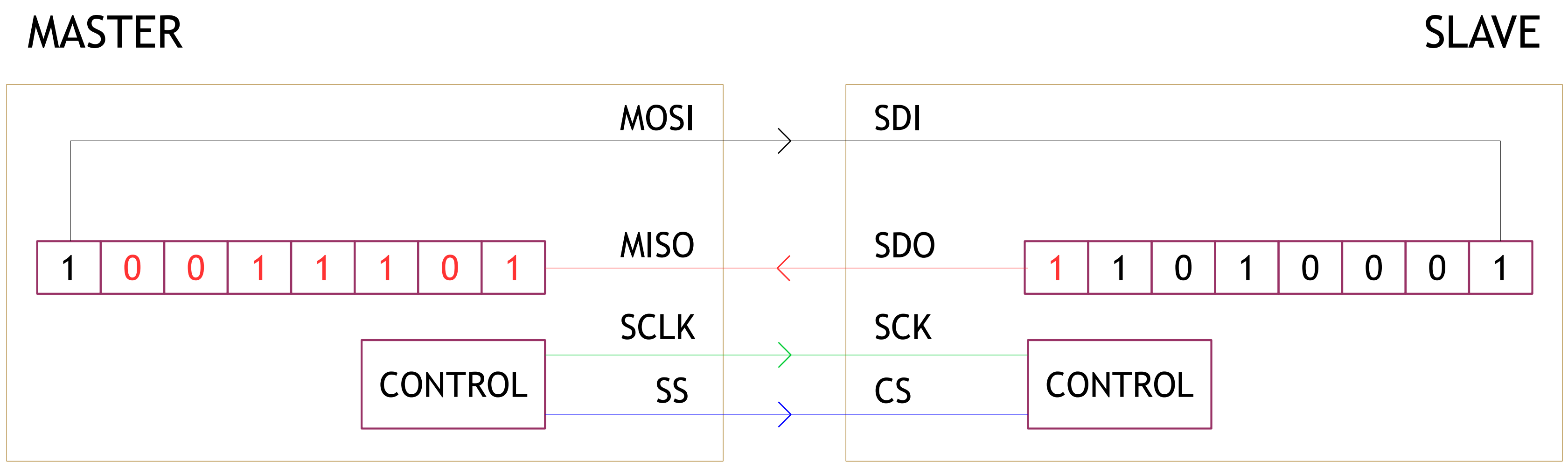


SPI

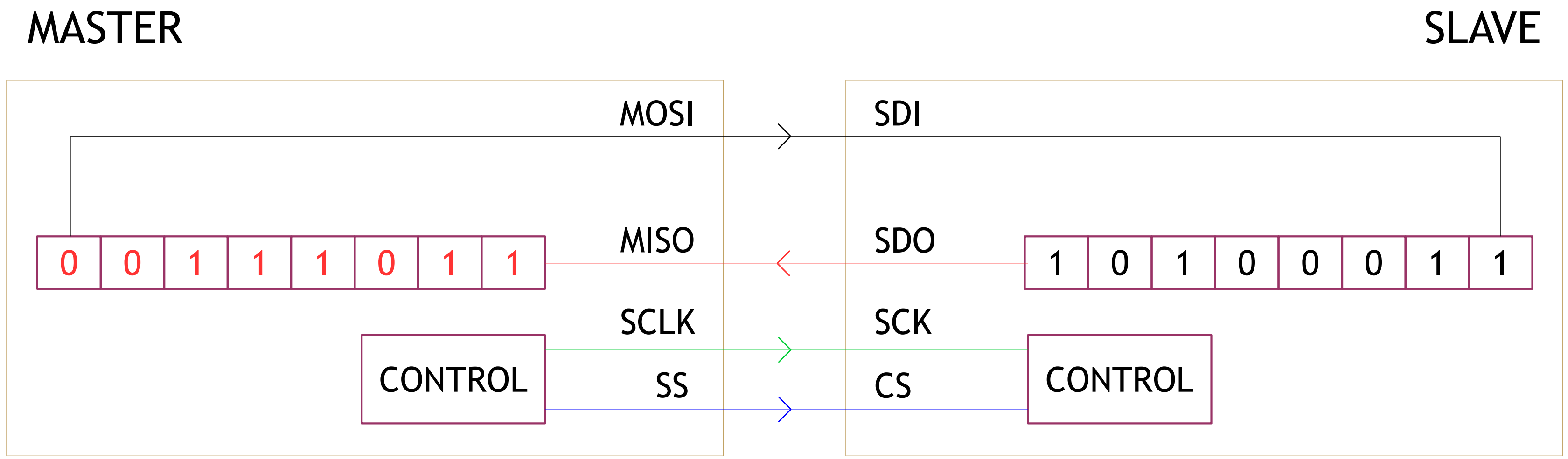
Data Transmission



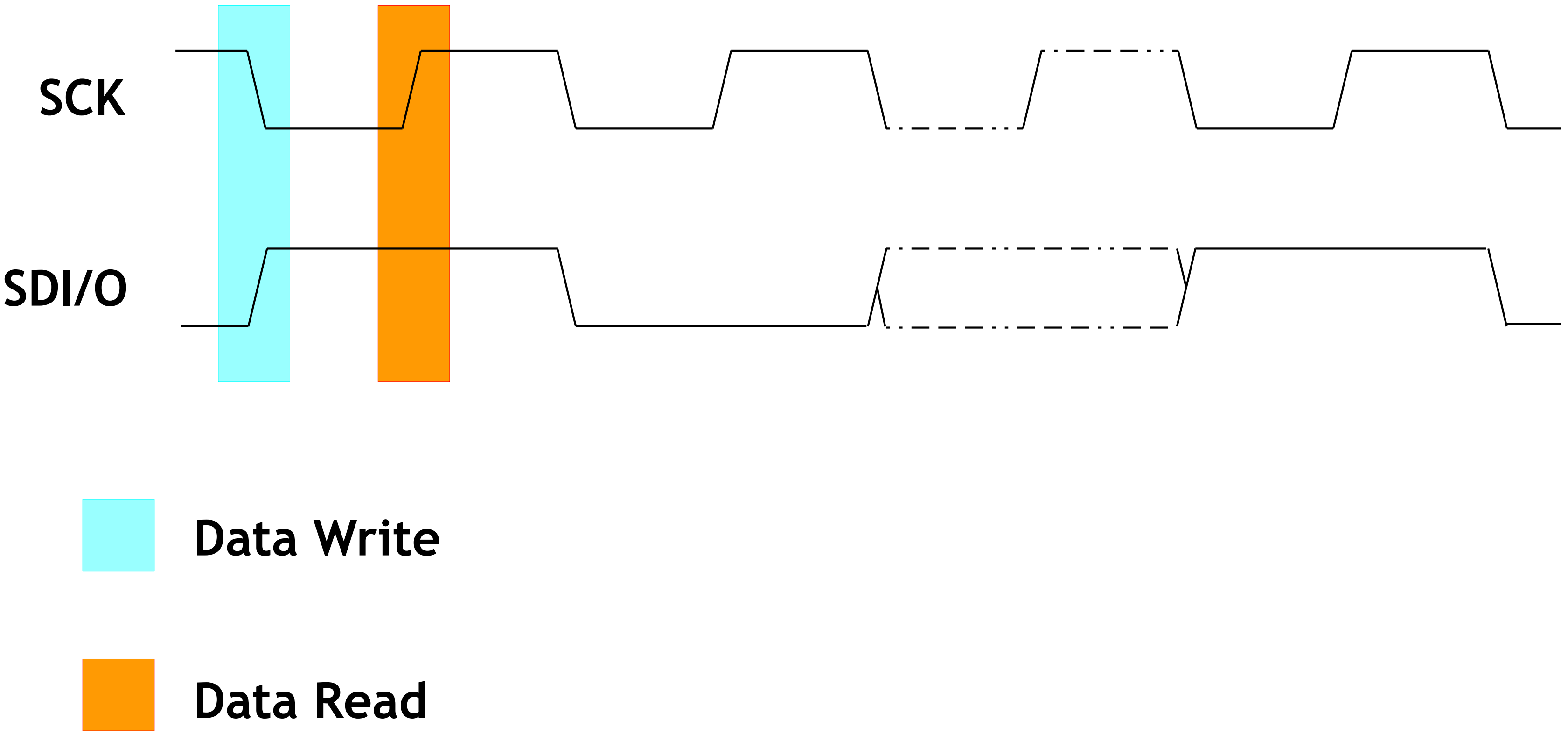
SPI
Data Transmission



SPI
Data Transmission



SPI
Data Validity



Inter Integrated Circuits

Inter Integrated Circuits

- Introduction
- Bus Features
- The Protocol
- Bus Speeds

I²C
Introduction

- Synchronous
- Half Duplex
- Multi Master / Slave

I^2C

Bus Features

- Two Line Interface
- Software Addressable
- Multi Master with CD
- Serial, 8 bit Oriented, Bidirectional with 4 Modes
- On Chip Filtering

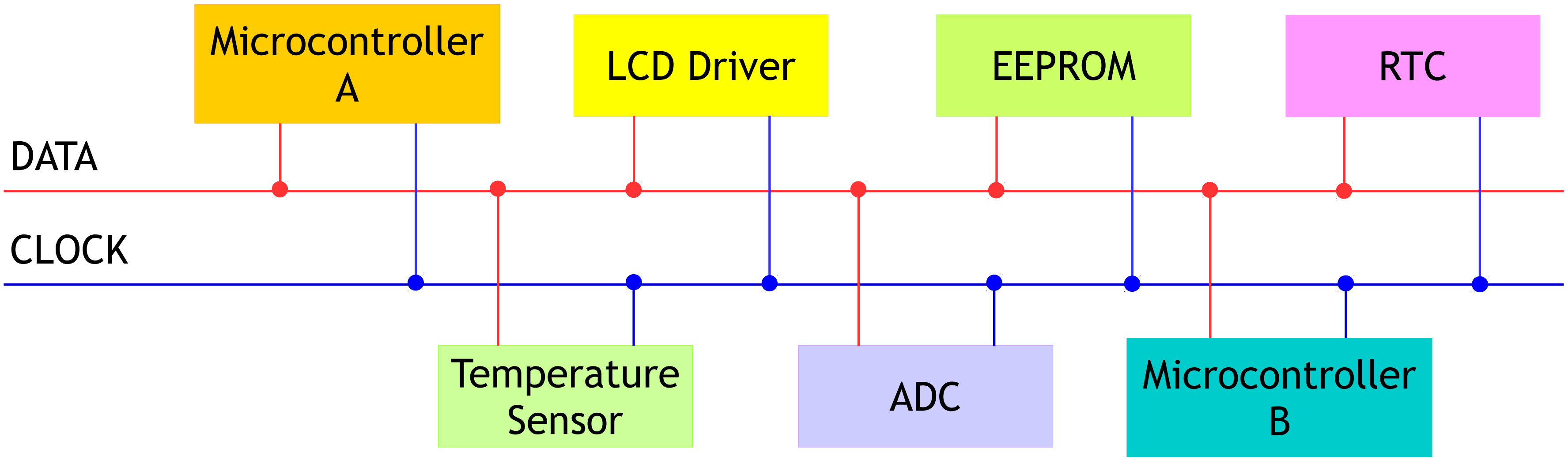
I^2C

Protocol

- Example
- Signals
- A Complete Data Transfer

I^2C

Example



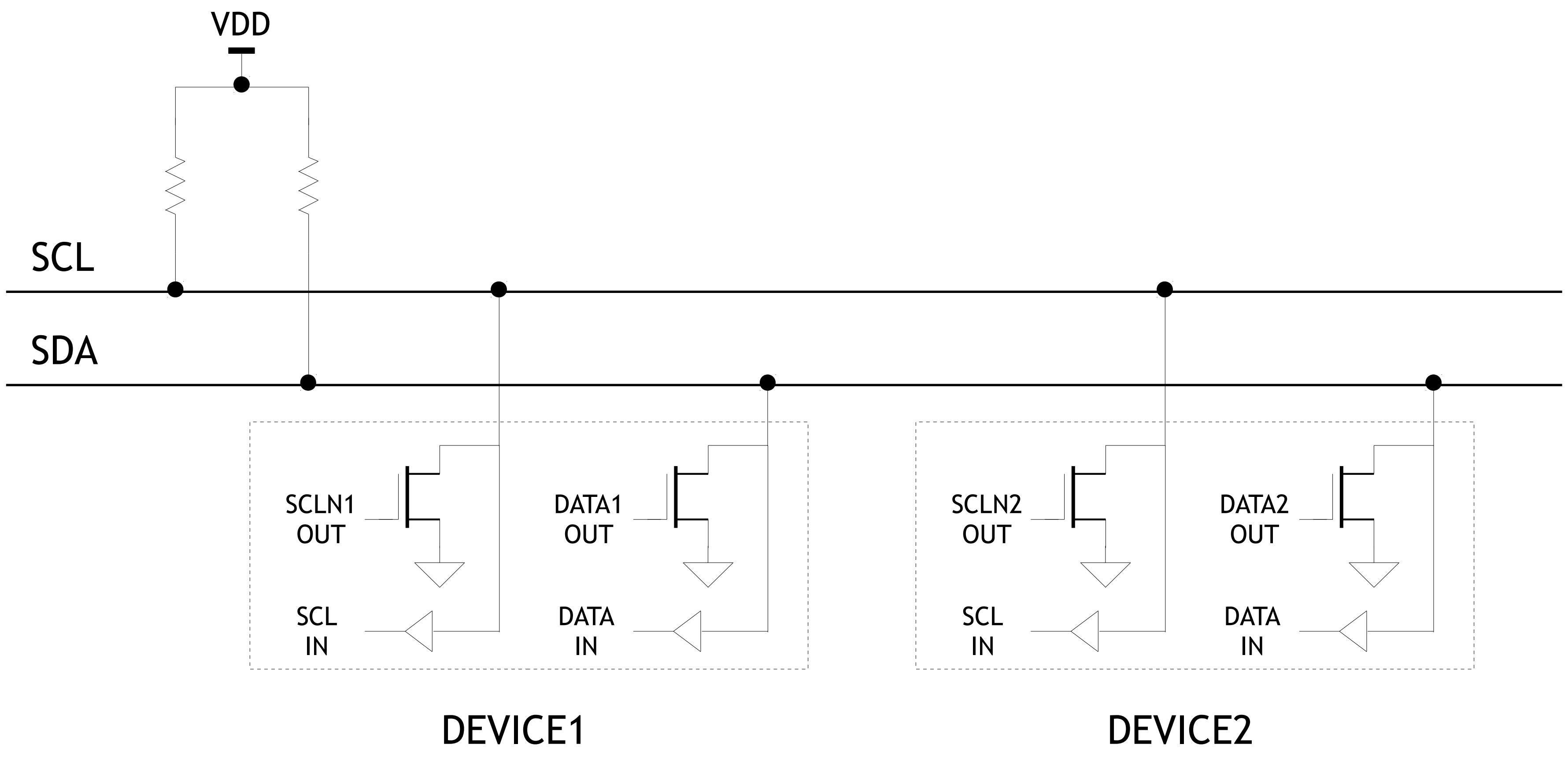
Microcontroller A
LCD Driver
EEPROM
RTC
DATA
CLOCK
Temperature Sensor
ADC
Microcontroller B

I^2C

Signals

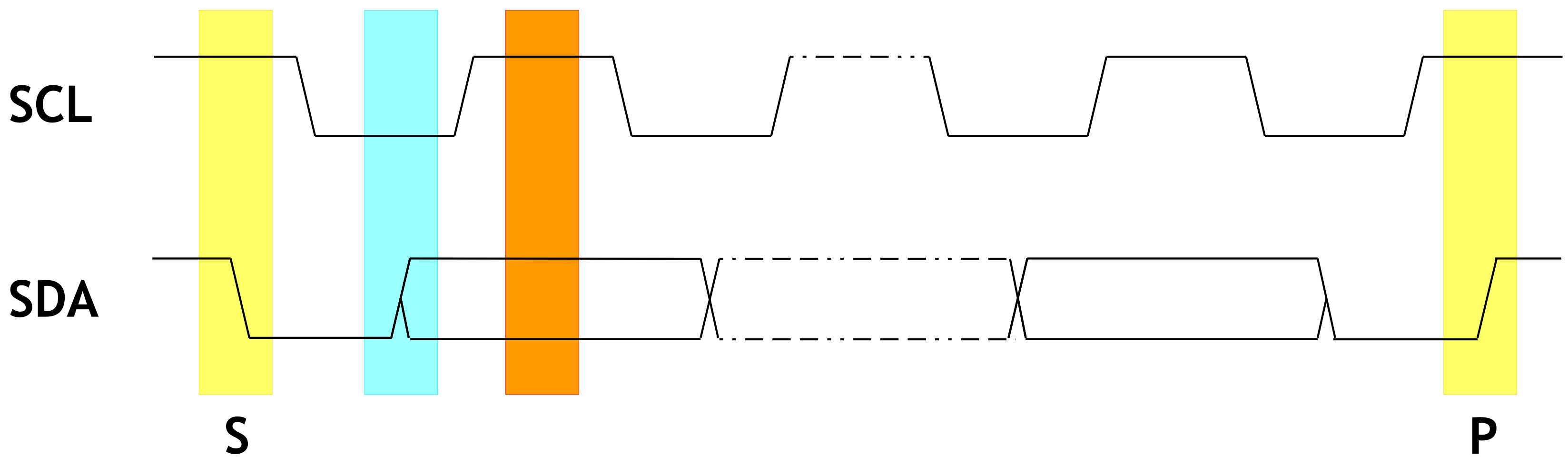
- Two-wired Interface
  - SDA
  - SCL
- Wired-AND
- Conditions and Data Validity
- Transmission

I²C
Signals – Wired-AND



I²C

Signals – Conditions and Data Validity



Data Write

Data Read

Conditions

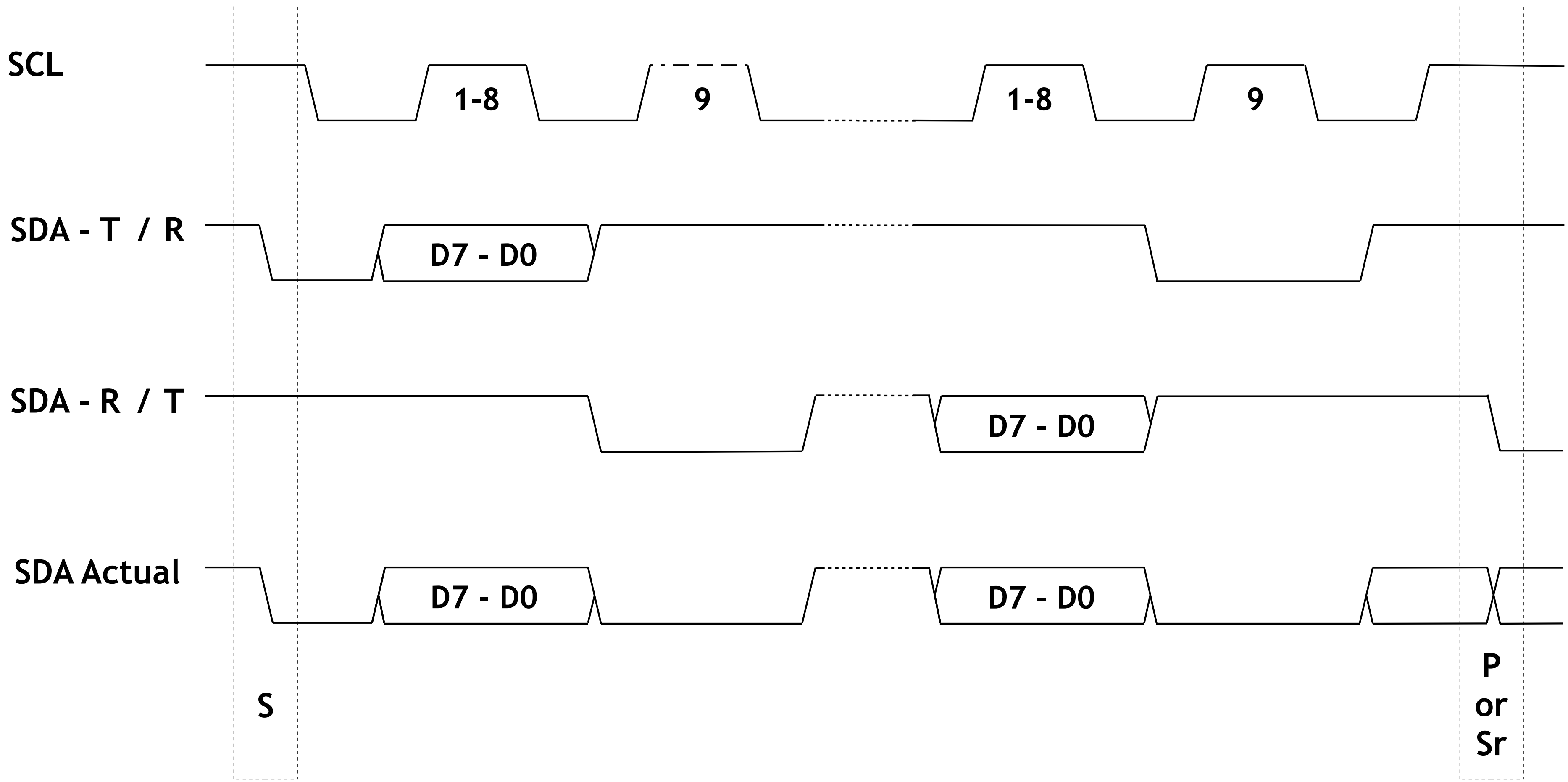
I^2C

Signals – Transmission

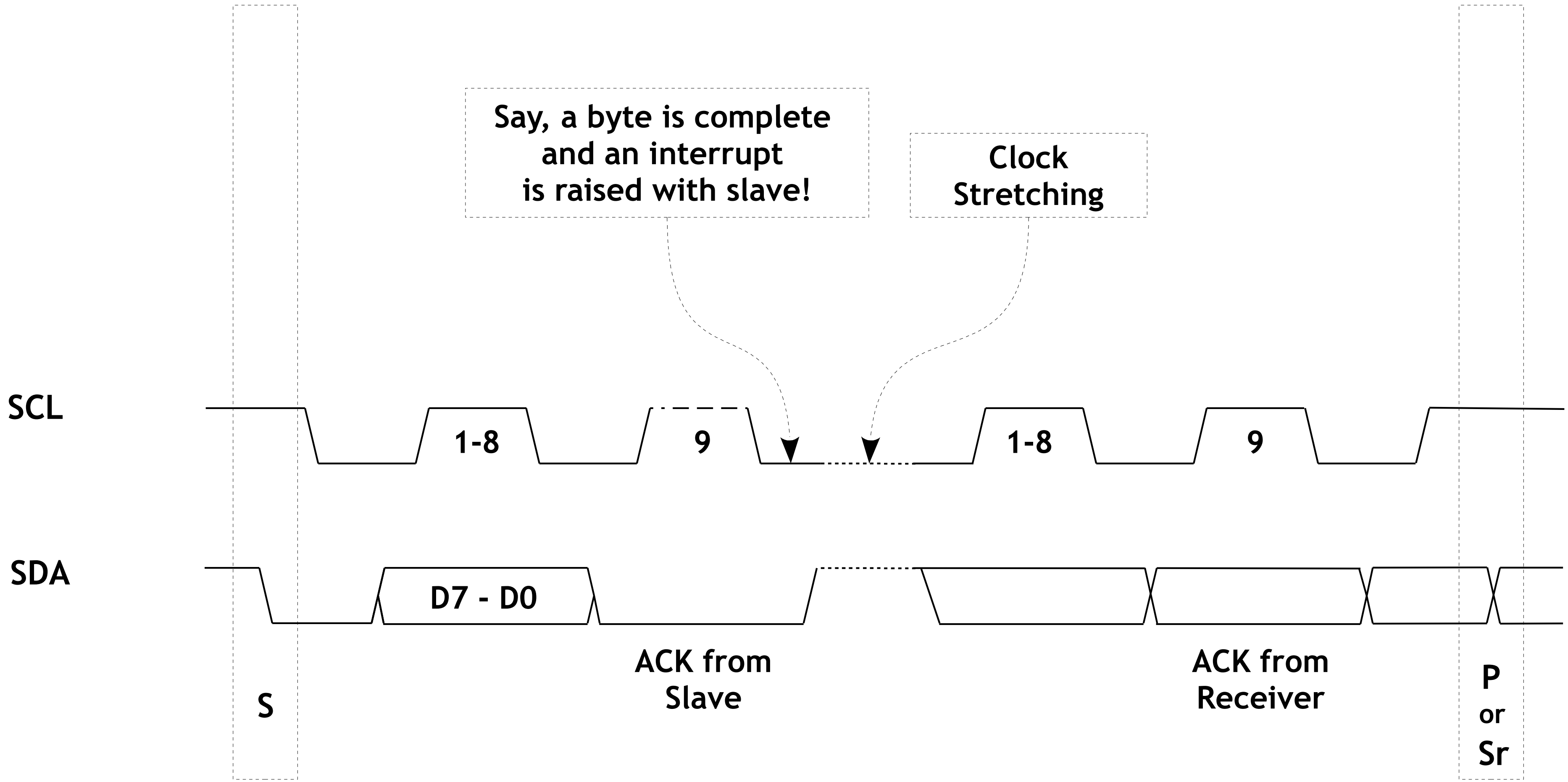
- Data on SDA
- Clocking on SCL
- Clock Synchronization
- Data Arbitration

I²C

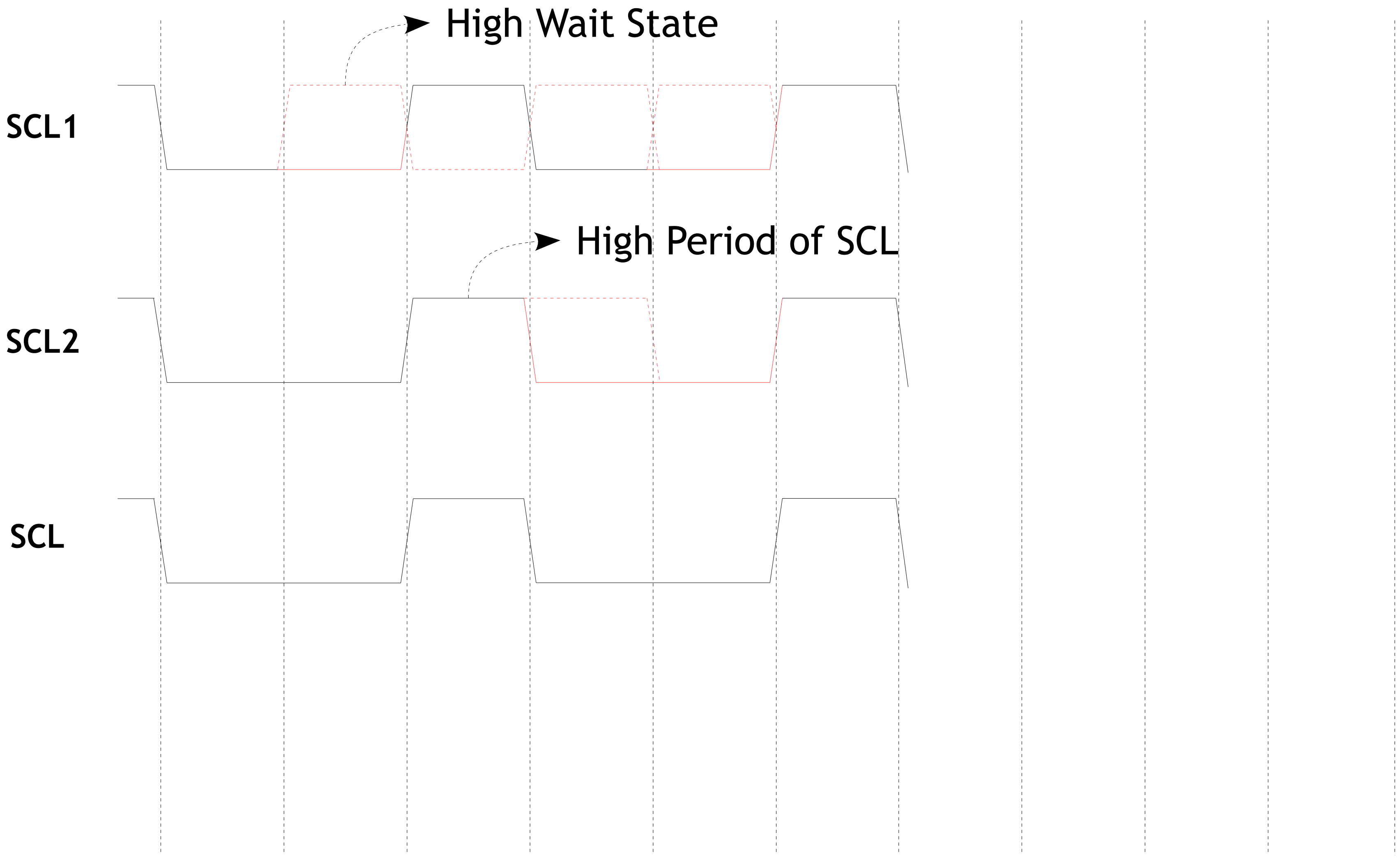
Signals – Data on SDA



I²C
Signals – Clocking on SCL

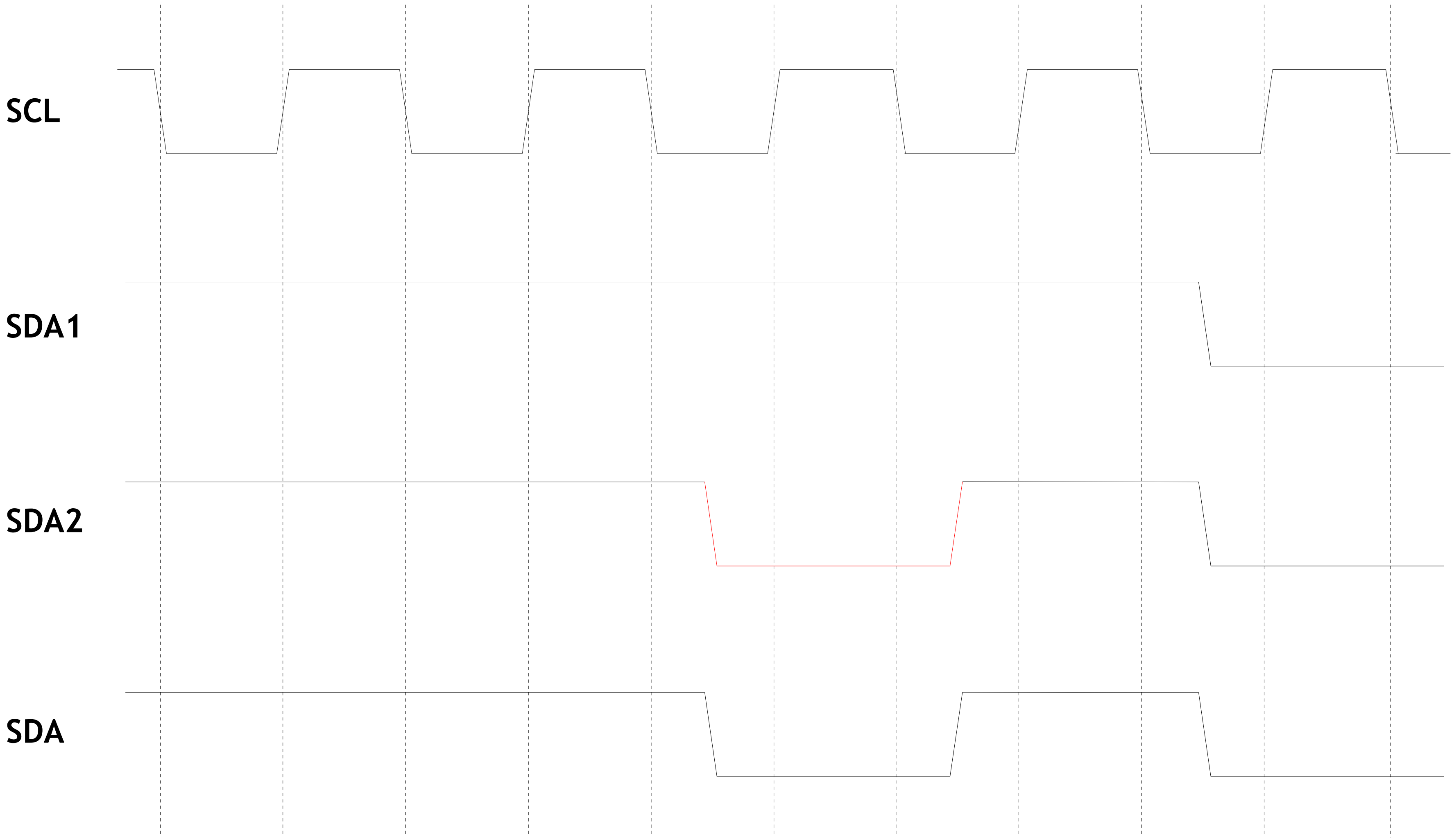


I²C
Signals – Clock Synchronization



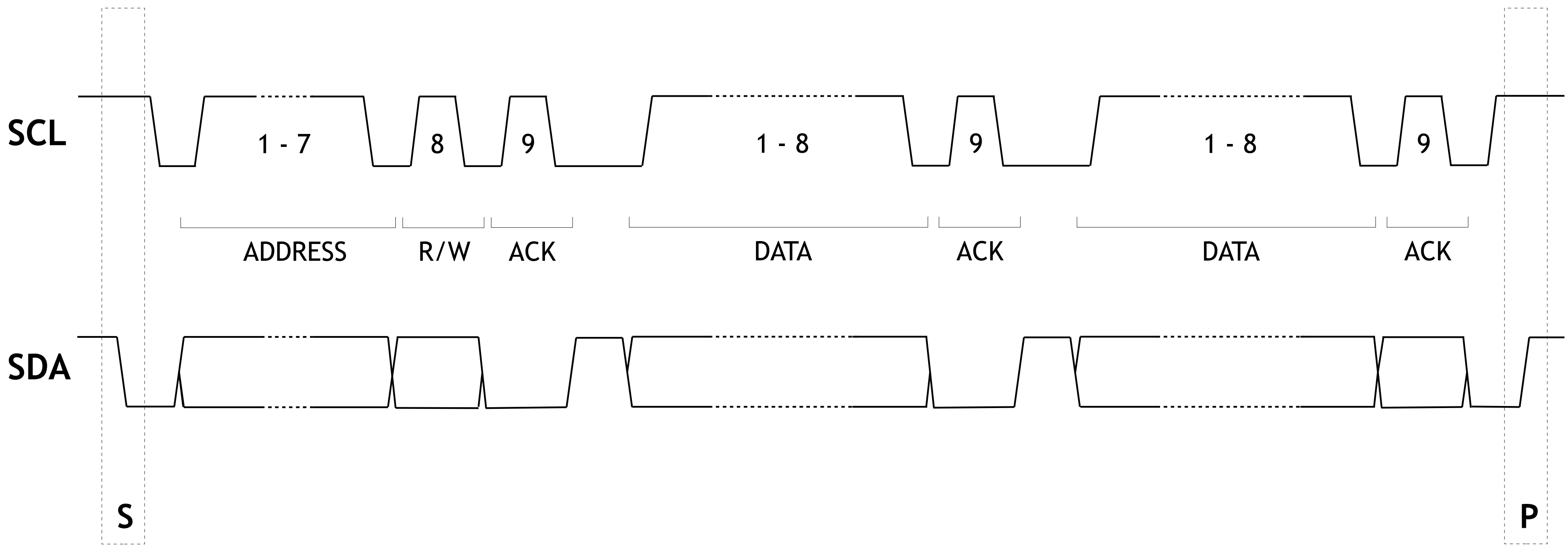
I²C

Signals – Data Arbitration



I^{2}C

A Complete Data Transfer



I^2C
Bus Speeds

- Bidirectional Bus
  - Standard Mode - 100 Kbit/s
  - Fast Mode - 400 Kbits/s
  - Fast Mode Plus - 1 Mbits/s
  - High Speed Mode - 3.4 Mbits/s
- Unidirectional Bus
  - Ultra Fast Mode – 5 Mbits/s
    - Uses Push-Pull Drivers (No Pullups)

Controller Area Network

Controller Area Network

- Introduction to CAN
- Basic Concepts
- Message Transfer
- Error Handling
- Fault Confinement

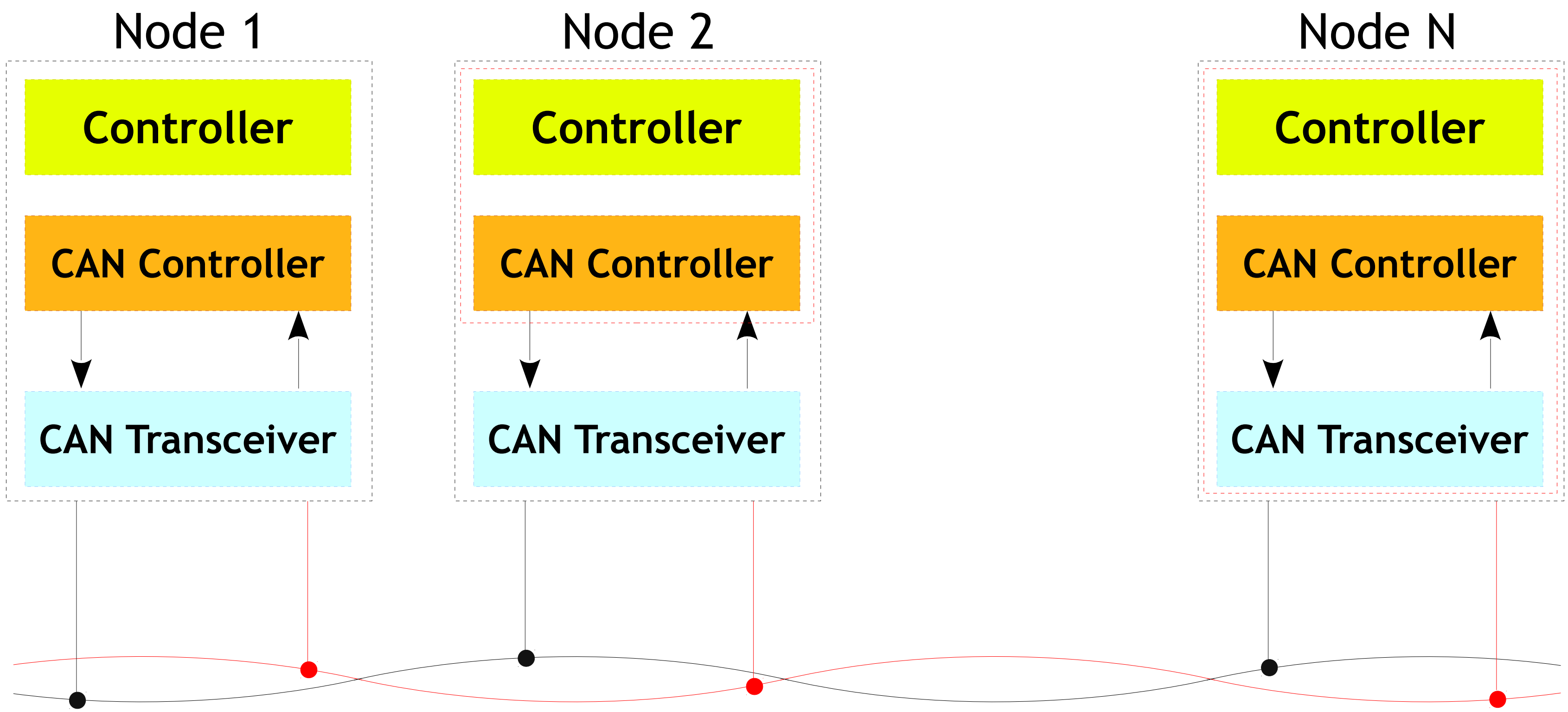
CAN
Introduction

- Asynchronous
- Half Duplex
- Multi Master / Slave

CAN
Basic Concepts

- Example
- Versions
- Absence of node addressing
  - Message identifier specifies contents and priority
  - Lowest message identifier has highest priority
- Non-destructive arbitration system by CSMA with collision detection
- Simple Transmission Medium
  - Twisted pair – CAN H and CAN L
- Properties
- Layered Architecture

CAN
Basic Concepts - Example



CAN

Basic Concepts - Versions

NOMENCLATURE	STANDARD	MAX SIGNALING RATE	IDENTIFIER
Low Speed CAN	ISO 11519	125 kbps	11 bit
CAN 2.0A	ISO 11898:1993	1 Mbps	11 bit
CAN 2.0B	ISO 11898:1995	1 Mbps	29 bit

CAN

Basic Concepts - Properties

- Prioritization of Messages
- Guarantee of Latency Times
- Configuration Flexibility
- Multicast Reception with Time Synchronization
- System wide Data Consistency
- Multi master
- Error Detection and Error Signaling
- Automatic Retransmission
- Distinction between temporary errors and permanent failures of nodes and autonomous switching off of defect nodes

EMERTXE

CAN

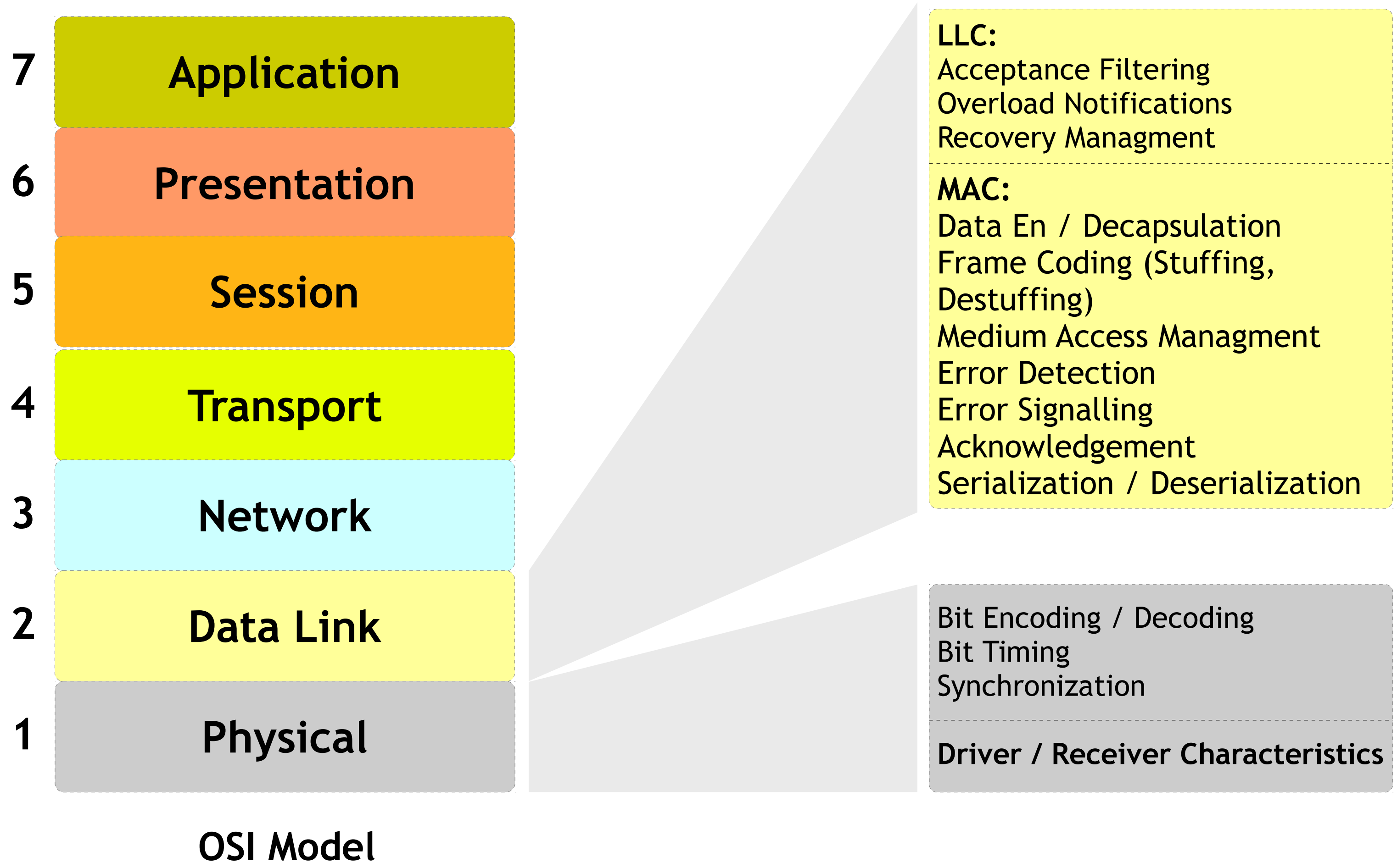
Basic Concepts - Layered Architecture

	OSI Model	
7	Application	Application
6	Presentation	Presentation
5	Session	Session
4	Transport	Transport
3	Network	Network
2	Data Link	Data Link
1	Physical	Physical

OSI Model

CAN

Basic Concepts - Layered Architecture

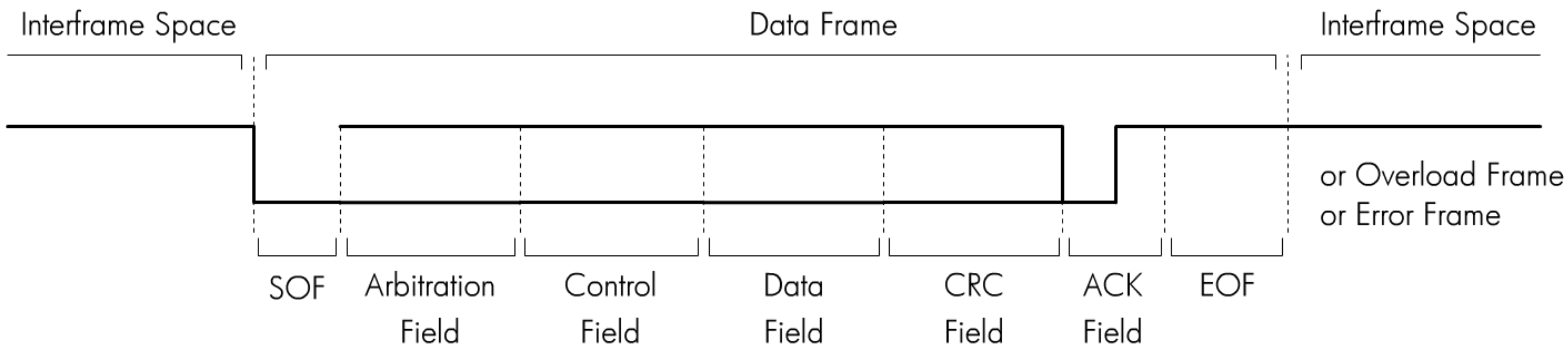


CAN
Message Transfer

- Frame Formats
  - Standard Frame - 11 bits Identifiers
  - Extended Frame - 29 bits Identifiers
- Frame Types
  - Data Frame
  - Remote Frame
  - Error Frame
  - Overload Frame
- Frame Fields

CAN

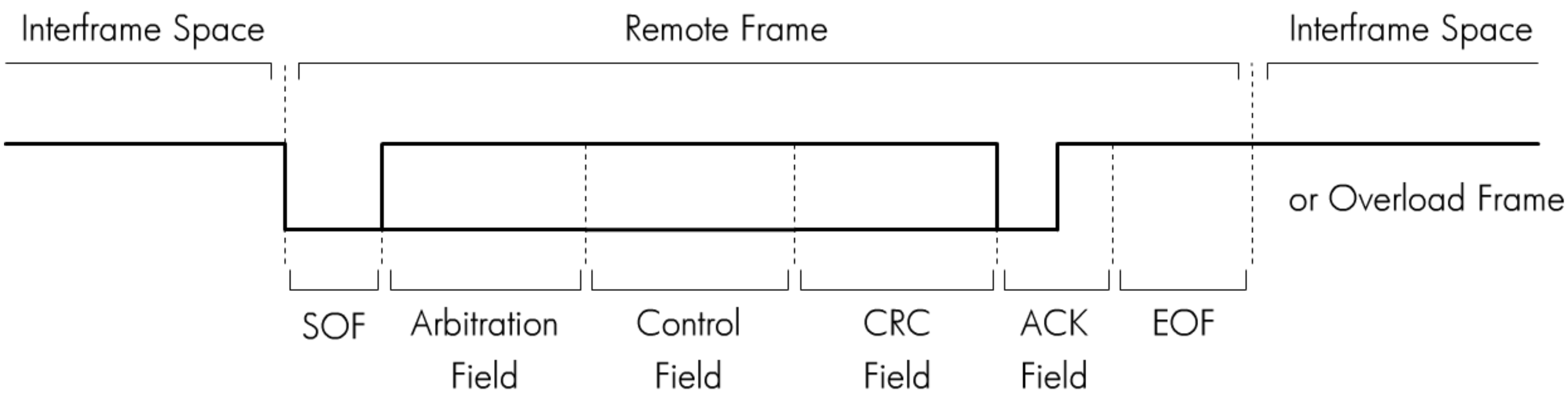
Message Transfer – Data Frame



- A data frame consists of seven fields: start-of-frame, arbitration, control, data, CRC, ACK, and end-of-frame.

CAN

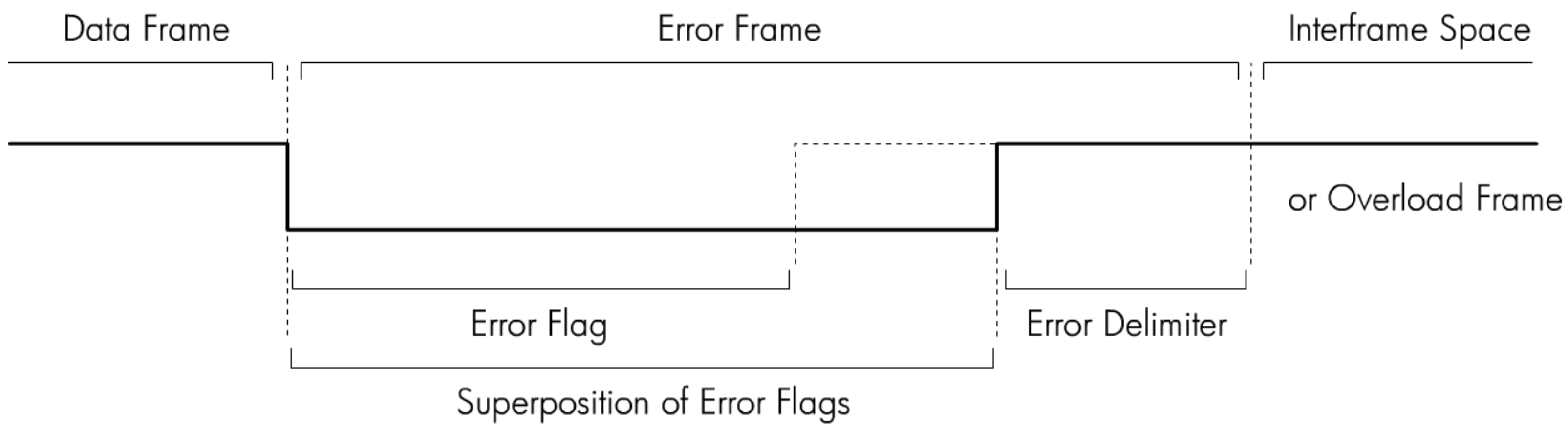
Message Transfer – Remote Frame



- Used by a node to request other nodes to send certain type of messages
- Has six fields as shown in above figure
  - These fields are identical to those of a data frame with the exception that the RTR bit in the arbitration field is **recessive** in the remote frame.

CAN

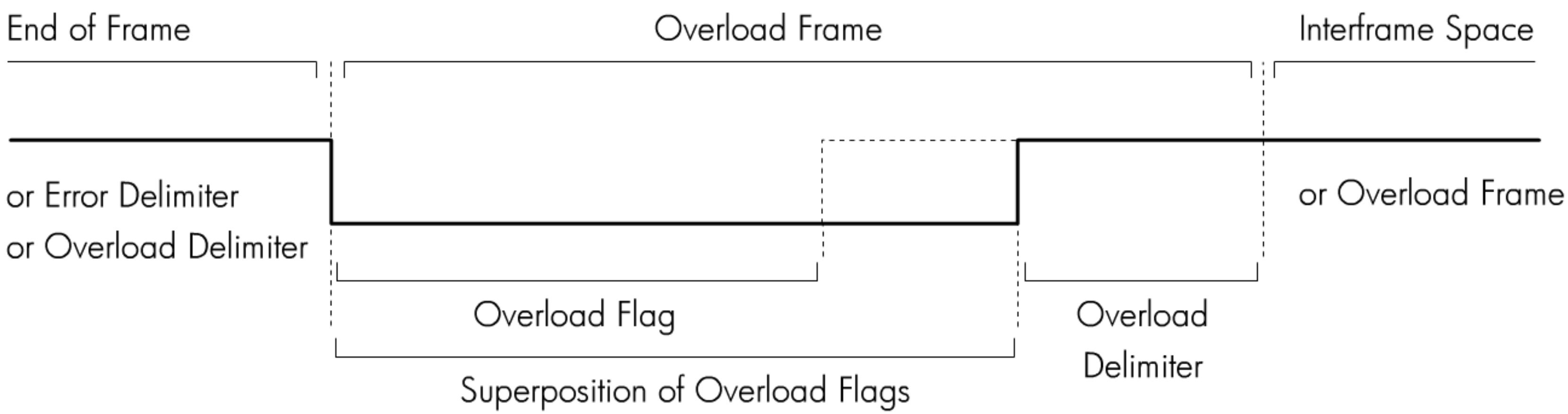
Message Transfer – Error Frame



- This frame consists of two fields.
  - The first field is given by the superposition of error flags contributed from different nodes.
  - The second field is the error delimiter.
- Error flag can be either active-error flag or passive-error flag.
  - Active error flag consists of six consecutive dominant bits.
  - Passive error flag consists of six consecutive recessive bits.
- The error delimiter consists of eight recessive bits.

CAN

Message Transfer – Overload Frame



- Consists of two bit fields: overload flag and overload delimiter
- Three different overload conditions lead to the transmission of the overload frame:
  - Internal conditions of a receiver require a delay of the next data frame or remote frame.
  - At least one node detects a dominant bit during intermission.
  - A CAN node samples a dominant bit at the eighth bit (i.e., the last bit) of an error delimiter or overload delimiter.
- Format of the overload frame is shown in above fig
- The overload flag consists of six dominant bits.
- The overload delimiter consists of eight recessive bits.

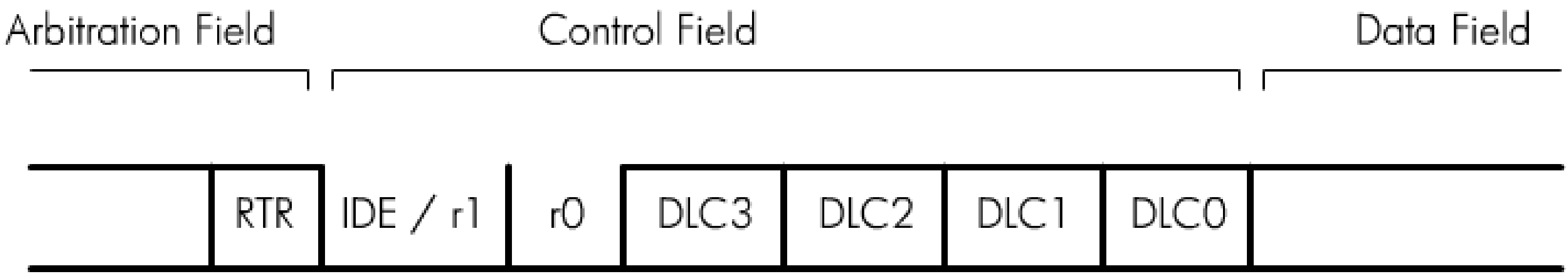
CAN

Message Transfer – Frame Fields

- Control Field
- Arbitration Field
- Data Field
- CRC Field
- ACK Field

CAN

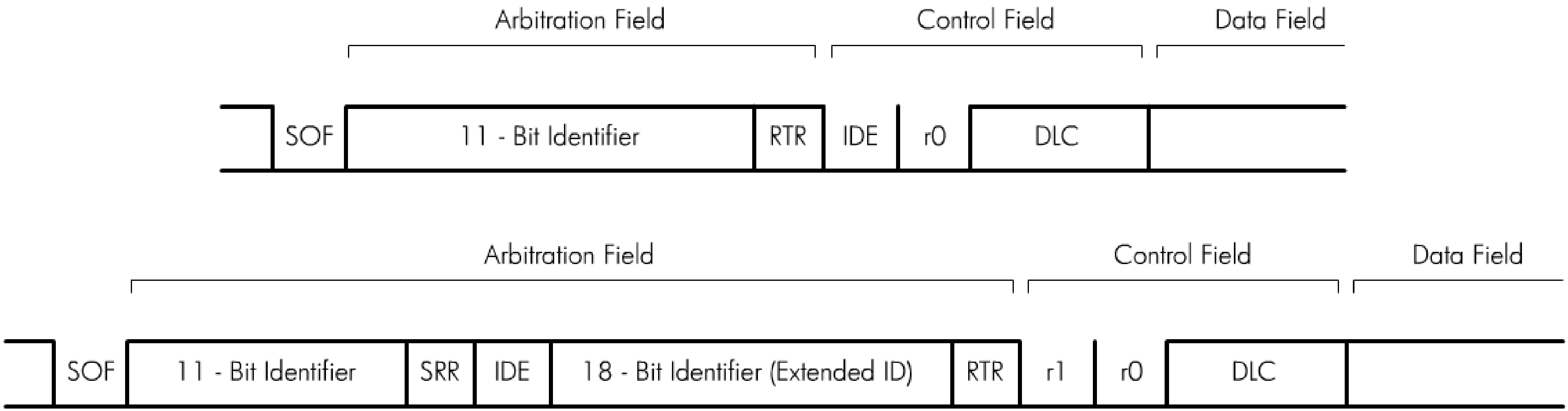
Frame Fields – Control Field



- The first bit is IDE bit for the standard format but is used as reserved bit r1 in extended format.
- r0 is reserved bit.
- DLC3…DLC0 stands for data length and can be from 0000 (0) to 1000 (8).

CAN

Frame Fields – Arbitration Field



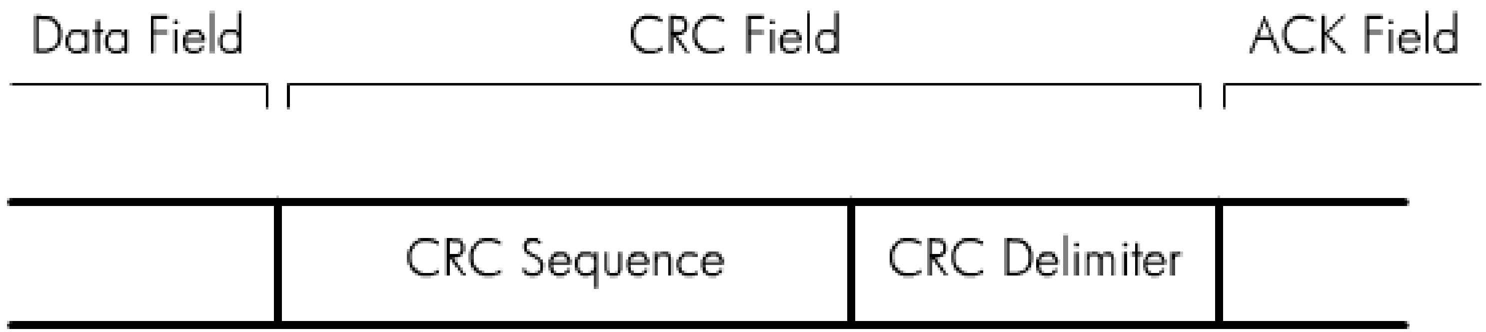
- The identifier of the standard format corresponds to the base ID in the extended format.
- The RTR bit is the remote transmission request and must be 0 in a data frame.
- The SRR bit is the substitute remote request and is recessive.
- The IDE field indicates whether the identifier is extended and should be recessive in the extended format.
- The extended format also contains the 18-bit extended identifier.

CAN
Frame Fields – Data Field

- May contain 0 to 8 bytes of data

CAN

Frame Fields – CRC Field



- It contains the 16-bit CRC sequence including CRC delimiter.
- The CRC delimiter is a single **recessive** bit.

CAN

Frame Fields – Ack Field

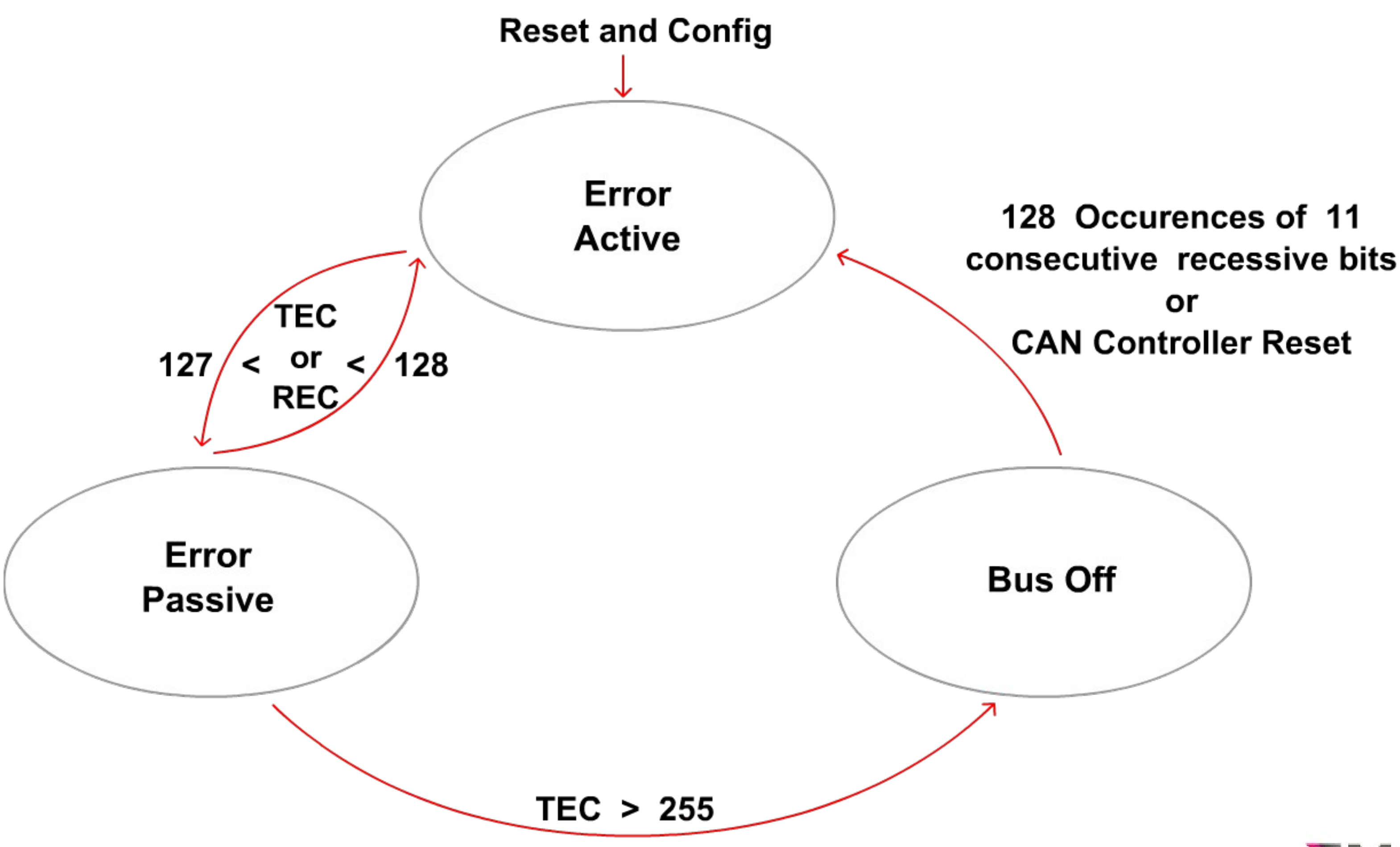
- Consists of two bits
- The first bit is the **acknowledgement bit**.
- This bit is set to recessive by the transmitter, but will be reset to dominant if a receiver acknowledges the data frame.
- The second bit is the **ACK delimiter** and is recessive.

CAN
Error Handling

- Error Detection
  - Bit Error
  - Stuff Error
- Error Signaling
  - CRC Error
  - Form Error
  - Acknowledgment Error

CAN
Fault Confinement

- Counters
  - Transmit Error Counter & Receive Error Counter



Thank You

Communication Protocols II

Wireless

Team Emertxe

Communication Protocols II

Introduction – Wireless - What?

- Transmission of signals (Voice, Video, Data etc..) using Electromagnetic Waves (RF) in open space
- The transmitter and receiver will have a defined channel to carry information across
- Multiple channels can co-exist with a fixed frequency bandwidth & capacity (bit rate) to transmit information in parallel and independently

Communication Protocols II

Introduction – Wireless - Why?

- Eliminates the need of messy and costly wires.
- Can communicate with devices where wiring is infeasible
- Global coverage
  - Buildings and Compounds
  - Towns and Cites
- Freedom to communicate on the move

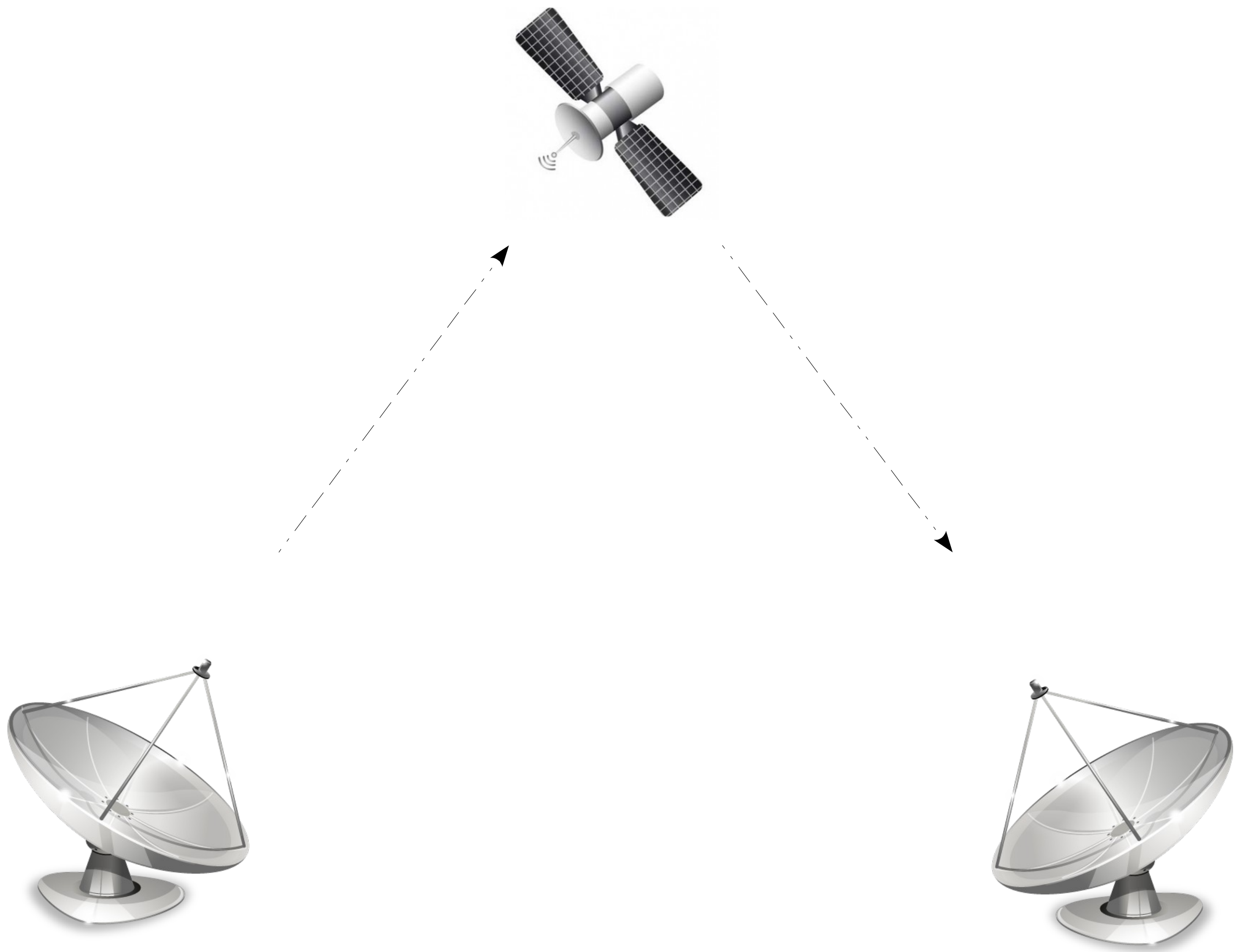
Communication Protocols II

Introduction – Wireless – General Frequencies

• FM Radio	- 88 MHz
• TV Broadcast	- 200 MHz
• Mobiles	- 900 MHz
• GPS	- 1.2 GHz
• PCS Phones	- 1.8 GHz
• Wi-Fi	- 2.4 / 5 GHz
• Bluetooth	- 2.4 / 5 GHz

Communication Protocols II

Introduction – Wireless – How does it happen?



Communication Protocols II

Introduction – Wireless – Types

- Radio: Easily generated, Omnidirectional , travel long distance , easily penetrates buildings.
  - Issues: Frequency dependent , relatively low bandwidth for data communication , tightly licensed by government.
- Microwave: Widely used for long distance communication , relatively inexpensive.
  - Issues: don't pass through buildings , weather and frequency dependent.
- IR and MM Waves: Widely used for short range communication,used for indoor wireless LANs, not for outdoors.
  - Issues: unable to pass through solid objects
- Light Waves: Unguided optical signal such as laser , unidirectional , easy to install , no license required.
  - Issues: Unable to penetrate rain or thick fog , laser beam can be easily diverted by air.

Communication Protocols II

Introduction – Wireless – Technologies

- Radio and Television Broadcasting
- Radar Communication
- Satellite communication
- Cellular Communication
- Global Positioning System
- Wi-Fi
- Bluetooth
- Radio Frequency Identification

Contents

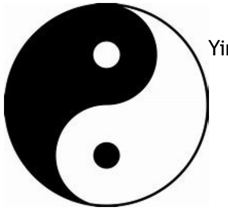
Contents

Wi-Fi

Communication Protocols II
Wi-Fi – Introduction

- WLAN based on IEEE 802.11 Standard
- IEEE generally build standards and thus does not test devices for compliance
- To fill this gap an alliance of different groups of companies was created named “Wi-Fi Alliance”
- Wi-Fi is trademark of Wi-Fi Alliance (NPO), help in conforming to certain standards of interoperability,

  The logo symbolizes this



Yin Yang

Communication Protocols II
Wi-Fi – Introduction

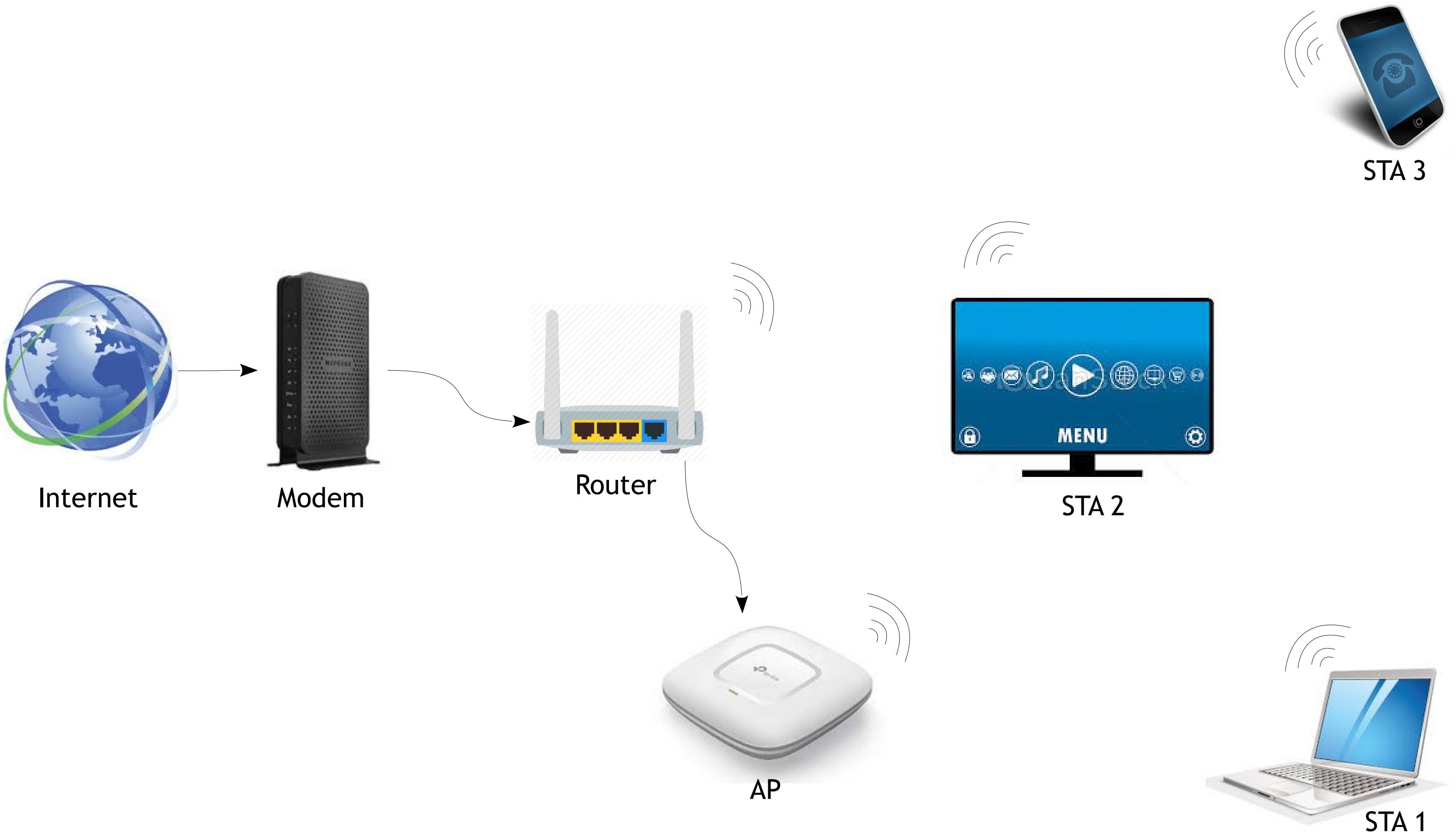
- Phil Belanger, a founding member of the Wi-Fi Alliance who presided over the selection of the name "Wi-Fi" writes:
  - Wi-Fi doesn't stand for anything.
  - It is not an acronym. There is no meaning.
- The above point should remove the misconception that the WiFi stands for Wifi Fidelity

Communication Protocols II
Wi-Fi – IEEE Standard

IEEE 802.11 PHY Standards

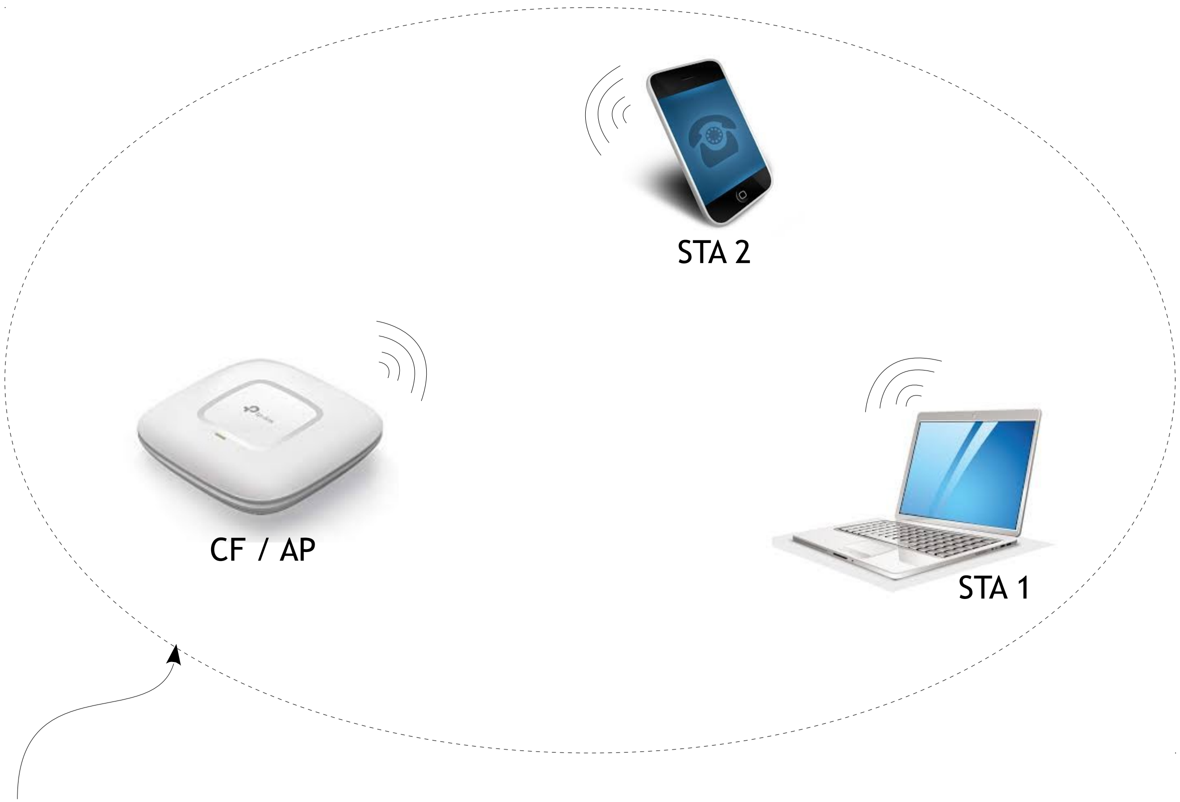
Release Date	Standard	Frequency Band (GHz)	Bandwidth (MHz)	Modulation	Antenna Technologies	Maximum Data Rate	Range (Mts)
1997	802.11	2.4 GHz	20 MHz	DSSS, FHSS	SISO	2 Mbps	20
1999	802.11b	2.4 GHz	20 MHz	DSSS	SISO	11 Mbps	35
1999	802.11a	5 GHz	20 MHz	OFDM	SISO	54 Mbps	35
2003	802.11g	2.4 GHz	20 MHz	DSSS, OFDM	SISO	542 Mbps	70
2009	802.11n	2.4, 5 GHz	20, 40 MHz	OFDM	MIMO, upto 4 spatial streams	640 Mbps	70
2013	802.11ac	5 GHz	40, 80, 160 MHz	OFDM	MIMO, MU-MIMO upto 8 spatial streams	6.93 Gbps	35
2013	802.11ad	60 GHz	2.16 GHz	SC, OFDM	10 x10 MIMO	6.76 Gbps	10
2013	802.11af	54-740 MHz	6, 7, 8 MHz	SC, OFDM	-	26.7 Mbps	> 1 K
2016	802.11ah	900 Mhz	1, 2, 3, 4, 5 MHz	SC, OFDM	-	40 Mbps	1 K

Communication Protocols II
Wi-Fi – Components



Communication Protocols II

Wi-Fi – Basic Service Set (BSS)



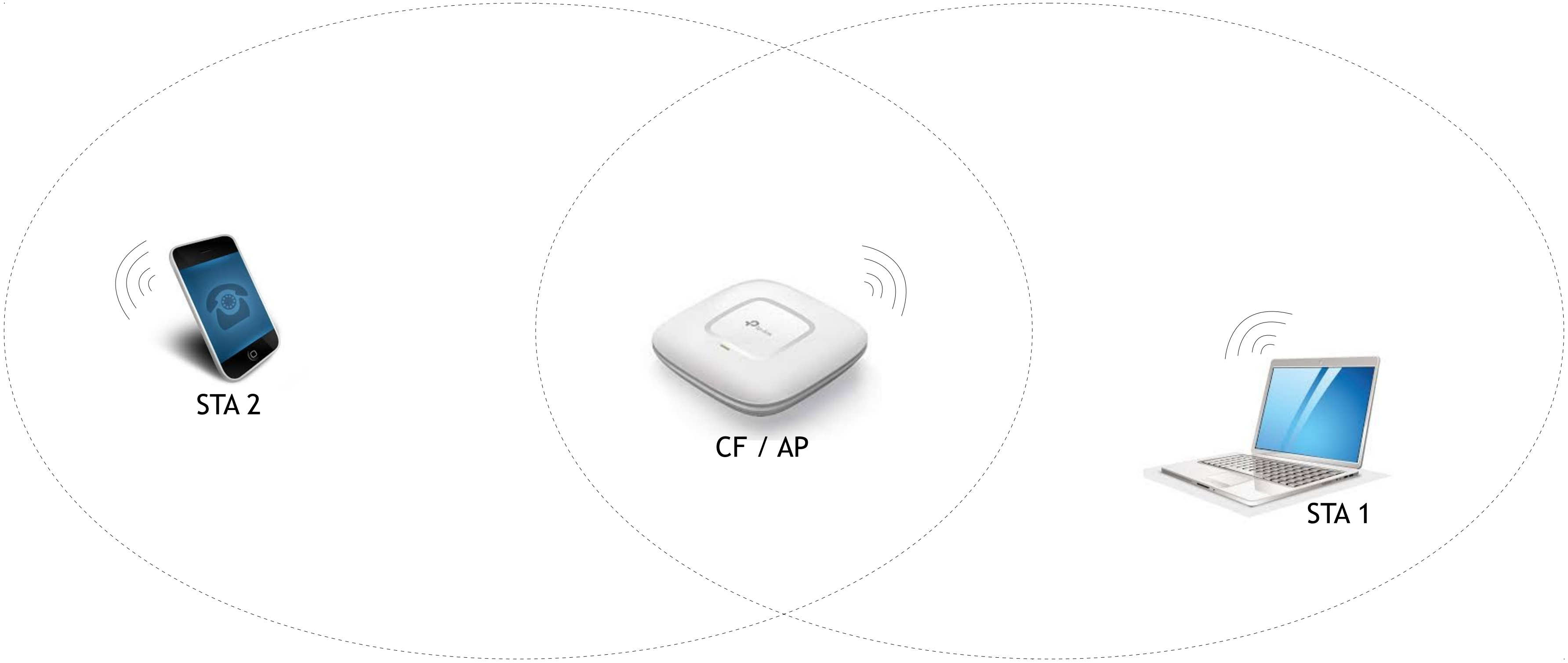
Rough coverage area influenced by
different environmental factors!

Communication Protocols II

Wi-Fi – Basic Service Set (BSS)

- All wireless devices that join a Wi-Fi network, are called as wireless stations
- When two or more stations are wirelessly connected they form a Basic Service Set
- A BSS is a set of STAs controlled by a single coordination function (CF). The CF is a logical function that determines when a STA transmits and when it receives.
- Not all STAs in a BSS can necessarily communicate directly. In the next diagram shown, STA 1 and 3 are mutually out of range, thus require use of STA 2 to relay messages.

Communication Protocols II
Wi-Fi – Basic Service Set (BSS)



Communication Protocols II

Wi-Fi – Operating Modes

- IEEE 802.11 standard: infrastructure mode and ad-hoc mode. Each one makes use of the BSS, but they yield different network topologies
  - Ad-hoc
  - Infrastructure

Communication Protocols II
Wi-Fi – Operating Modes – Ad-hoc

- An independent BSS (IBSS) is the simplest type of 802.11 network. Wireless stations communicate directly with one another forming peer-to-peer model
- A BSS operating in ad-hoc mode is isolated. There is no connection to other Wi-Fi networks or to any wired LANs.

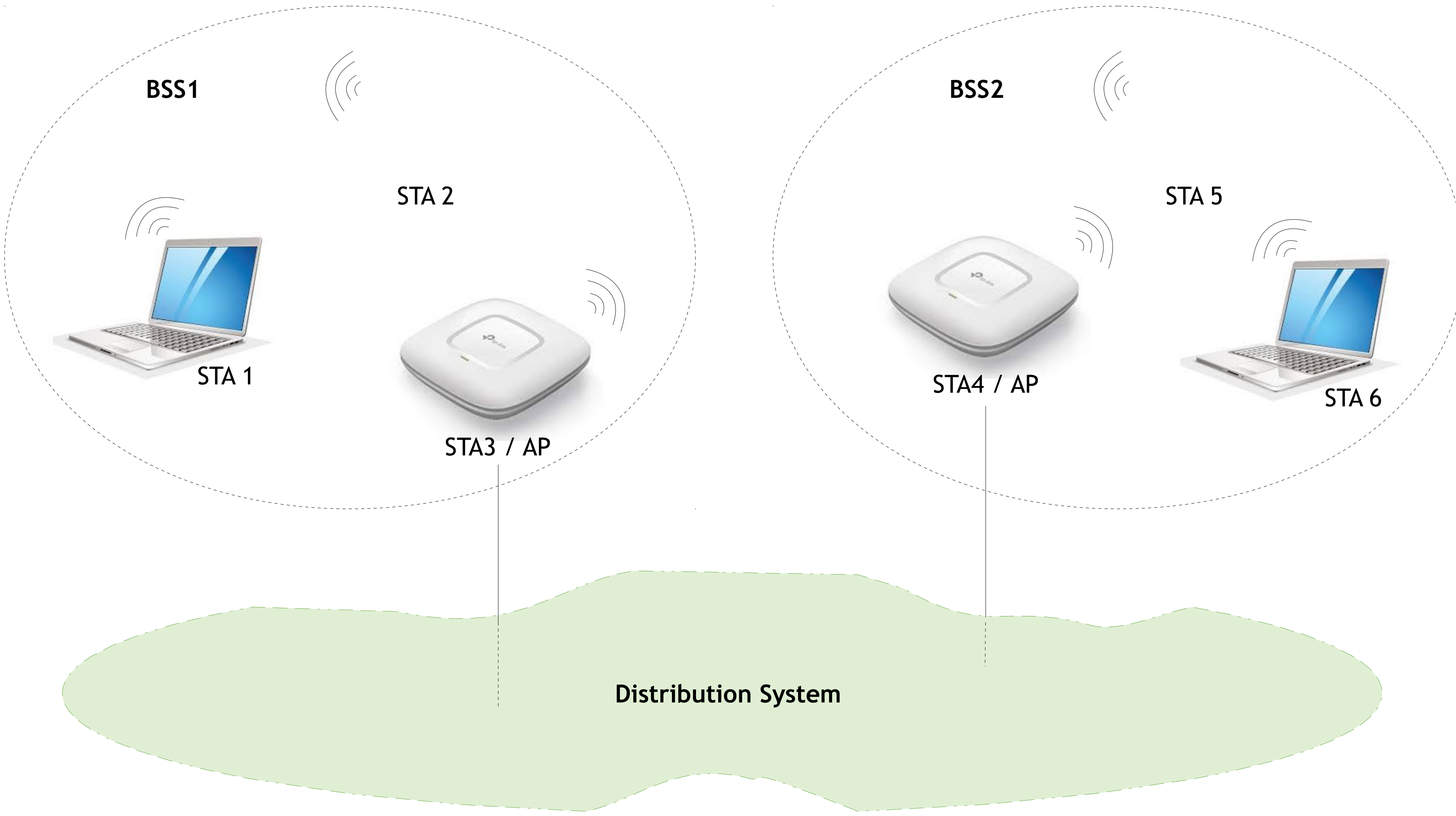
Communication Protocols II

Wi-Fi – Operating Modes – Infrastructure

- Requires a BSS containing one wireless access point (AP)
- An AP is a STA with additional functionality. A major role for an AP is to extend access to wired networks for the clients of a wireless network
- All wireless devices trying to join the BSS must associate with the AP. An AP provides access to its associated STAs to what is called the distribution system (DS). The DS is an architectural component that allows communication among Aps
- The IEEE 802.11 specification does not define any physical characteristics or physical implementations for the DS. Instead, it defines services that the DS must provide

Communication Protocols II

Wi-Fi – Operating Modes – Infrastructure



Communication Protocols II

Wi-Fi – Operating Modes – Infrastructure - DS

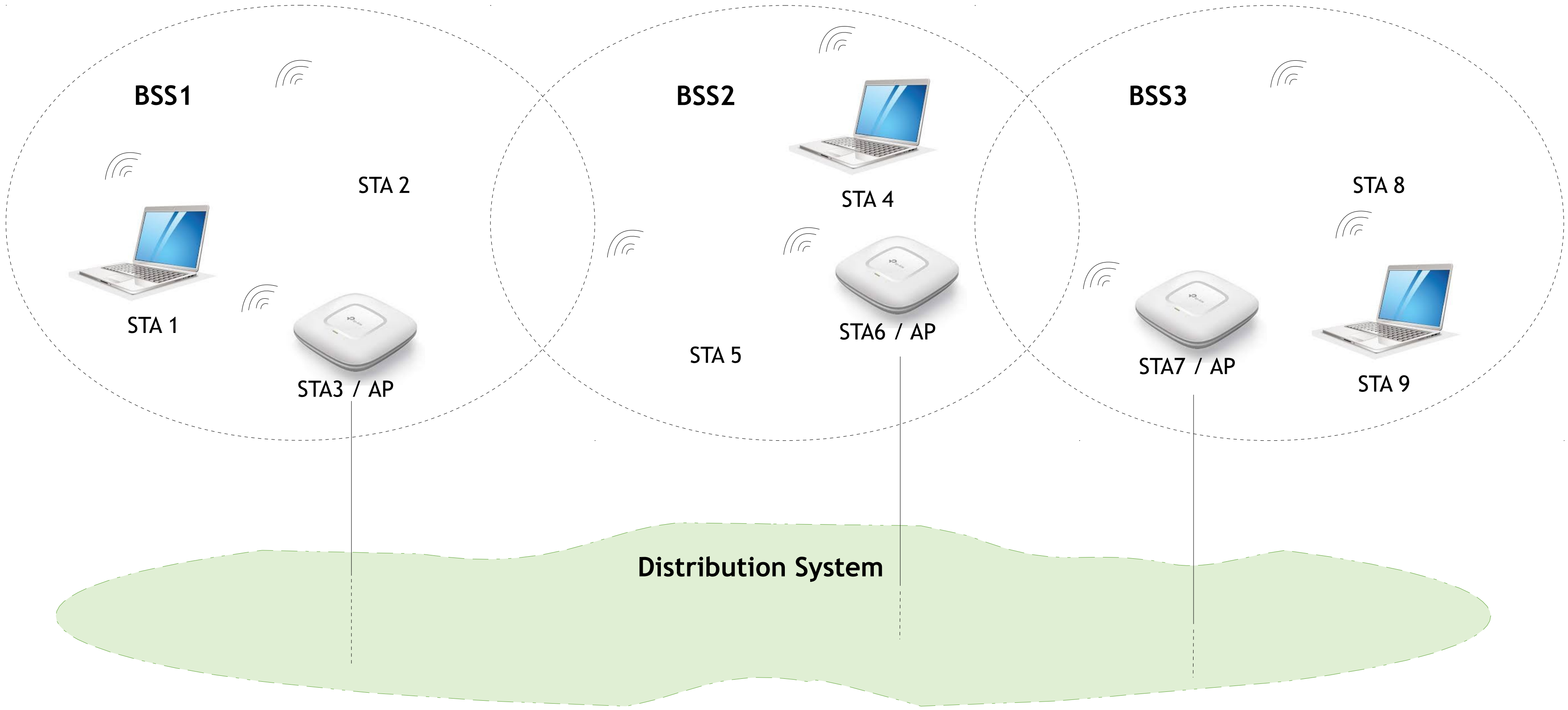
- Physical connection with Coaxial cabling or fiber optic cabling
- Logically different from the wireless medium
- Addresses used on the DS medium do not have to be the same as used in AP
- This setup is similar to the host/hub model (or “star topology”) used frequently in wired networks.

Communication Protocols II
Wi-Fi – ESS

- A common distribution system (DS) and two or more BSSs create what is called an extended service set (ESS)
- An ESS is a Wi-Fi network of arbitrary size and complexity
- The DS enables mobility in a Wi-Fi network by a method of tracking the physical location of STAs, thus ensuring that frames are delivered to the AP associated with the destination STA.
  - Mobility: move anywhere within the coverage area of the ESS and keep an uninterrupted connection

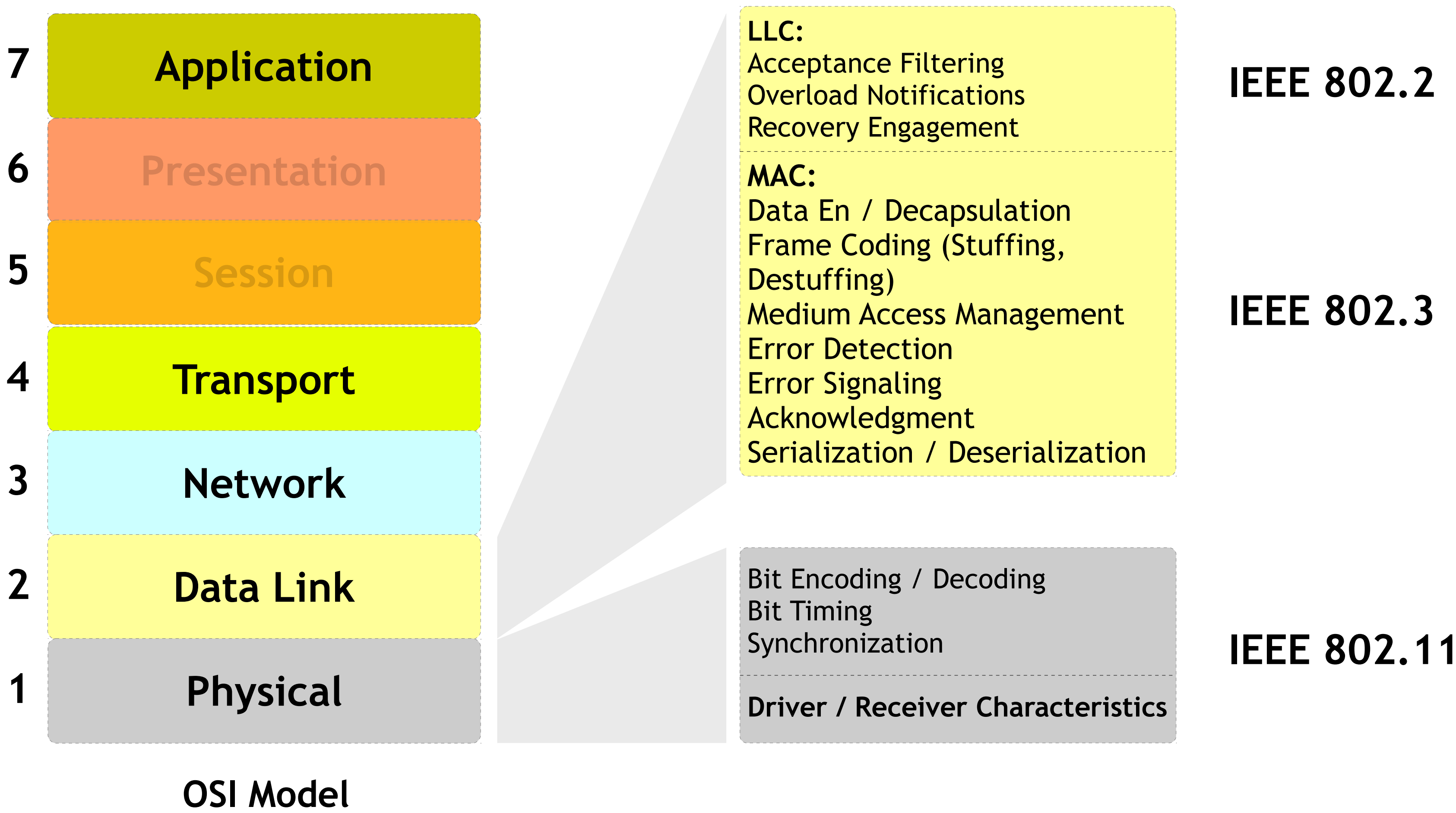
Communication Protocols II
Wi-Fi – ESS

- The network name, or SSID, must be the same for all APs participating in the same ESS.



Communication Protocols II

Wi-Fi – Layers



Communication Protocols II

Wi-Fi – Layers - PHY

- Responsible for such things as modulation methods, encoding schemes and the actual transmission of radio signals through space.
- PHY implementations operate in specific bands. A band defines the frequencies allocated for particular applications.
- Many Wi-Fi devices are designed for use in the Industrial, Scientific and Medical (ISM) band.
- The ISM band is for license-free devices; regulatory requirements demand that license-free devices use spread-spectrum technology. Direct sequence spread spectrum (DSSS) PHYs are the most widely deployed at this point in time.

Communication Protocols II

Wi-Fi – Layers - MAC

- A sublayer of the data link layer (DLL). It rides above the physical layer, controlling transmission of data and providing interaction with a wired backbone, if one exists.
- The MAC layer also provides services related to the radio and mobility management.
- To move data packets across a shared channel, the MAC layer uses CSMA/CA (Carrier Sense Multiple Access / Collision Avoidance), which is very similar to the strategy used in 802.3 MAC layers: CSMA / CD (Collision Detection).

Communication Protocols II

Wi-Fi – Layers - MAC

- CSMA / CA and CSMA / CD are both peer-to-peer protocols, but unlike CSMA / CD, which deals with transmissions after a collision has occurred, CSMA / CA acts to prevent collisions before they happen.
- The 802.11 MAC layer is required to appear to a logical link control (LLC) layer as an IEEE 802 LAN, thus Wi-Fi and Ethernet both use MAC addresses in the same format, i.e., 6 octets that are globally unique.

Wi-Fi – IEEE 802.11 Services

- The IEEE 802.11 standard does not define any specific implementations. Instead, nine services are specified that all implementations must provide.

Communication Protocols II

Wi-Fi – IEEE 802.11 Services

- Station Services
  - Authentication
    - A wireless station needs to be identified before it can access network services. This process is called authentication. It is a required state that comes before the STA may enter the association state
  - Deauthentication
    - This service voids an existing authentication
  - Privacy
    - A wireless station must be able to encrypt frames in order to protect message content so that only the intended recipient can read it
  - MAC Service Data Unit (MSDU) Delivery
    - An MSDU is a data frame that must be transmitted to the proper destination

Communication Protocols II

Wi-Fi – IEEE 802.11 Services

- Distribution System Services (DSS)
  - Association
    - This service establishes an AP/STA mapping after mutually agreeable authentication has taken place between the two wireless stations. A STA can only associate with one AP at a time. This service is always initiated by the wireless station and when successfully completed enables station access to the DSS.
  - Reassociation
    - This service moves a current association from one AP to another AP.
  - Disassociation
    - This service voids a current association

Wi-Fi – IEEE 802.11 Services

- Distribution
  - This service handles delivery of MSDUs within the distribution system; i.e., the exchange of data frames between APs in an extended service set (ESS).
- Integration
  - This service handles delivery of MSDUs between the distribution system and a wired LAN on the other side of a portal. Basically this is the bridging function between wireless and wired networks

Communication Protocols II

Wi-Fi – State Variable

- Each wireless station maintains two state variables, one for authentication and one for association.
- A wireless station is authenticated or unauthenticated.
- Once in an authenticated state, the STA is either associated or unassociated.
- So possible states are
  - State 1: Unauthenticated and unassociated.
  - State 2: Authenticated, not associated.
  - State 3: Authenticated and associated.

- There are different types of IEEE 802.11 frames with multiple subtypes
  - Management
  - Control
  - Data

Communication Protocols II

Wi-Fi – Frames - Management

- 802.11 management frames make up a majority of the frame types in a WLAN.
- Management frames are used by wireless stations to join and leave the basic service set (BSS).
- Another name for an 802.11 management frame is Management MAC Protocol Data Unit (MMPDU).
- Information fields are fixed-length fields in the body of a management frame
- Information elements are variable in length

Communication Protocols II

Wi-Fi – Frames - Controls

- 802.11 control frames assist with the delivery of the data frames
- Control frames are transmitted at one of the basic rates
- Control frames are also used to:
  - Clear the channel
  - Acquire the channel
  - Provide unicast frame acknowledgments
- They contain only header information

Communication Protocols II

Wi-Fi – Frames - Data

- Most 802.11 data frames carry the actual data that is passed down from the higher-layer protocols.
- The layer 3 – 7 MSDU payload is normally encrypted for data privacy reasons.
- Some 802.11 data frames carry no MSDU payload at all but do have a specific MAC control purpose within a BSS.
- Any data frames that do not carry an MSDU payload are not encrypted because a layer 3 – 7 data payload does not exist.

Communication Protocols II

Wi-Fi – Frames - Data

- The simple data frame has MSDU upper-layer information encapsulated in the frame body.
- The integration service that resides in access points and WLAN controllers takes the MSDU payload of a simple data frame and transfers the MSDU into 802.3 Ethernet frames.
- Null function frames are sometimes used by client stations to inform the AP of changes in Power Save status.

Communication Protocols II

Wi-Fi – Security

- Service Set Identifier (SSID)
- Wired Equivalent Privacy (WEP)
- Wireless Protected Access (WPA)
- IEEE 802.11i

Communication Protocols II

Wi-Fi – Security

- Wired Equivalent Privacy (WEP),not that secure
- Wi-Fi Protected Access (WPA), a subset of the upcoming 802.11i security standard, will replace the flawed Wired Equivalent Privacy (WEP).
- Without your SSID, people will not be able to join your Wi-Fi hotspot.

Bluetooth

Communication Protocols II

Bluetooth – Introduction

- WPAN based on IEEE 802.15.1 Standard which no longer maintained by IEEE
- The Bluetooth SIG (Special Interest Group) oversees development of the specification, manages the qualification program, and protects the trademarks
- It was originally conceived as a wireless alternative to RS-232 data cables.
- Short distance communication using ISM band from 2.4 to 2.485 GHz

Communication Protocols II

Bluetooth – Introduction – Applications

Cable Replacements

- All modern accessories like
  - Keyboard
  - Mouse
  - Speakers
  - Phones

  get connected wirelessly

Communication Protocols II

Bluetooth – Introduction – Applications

Ad-hoc Networking

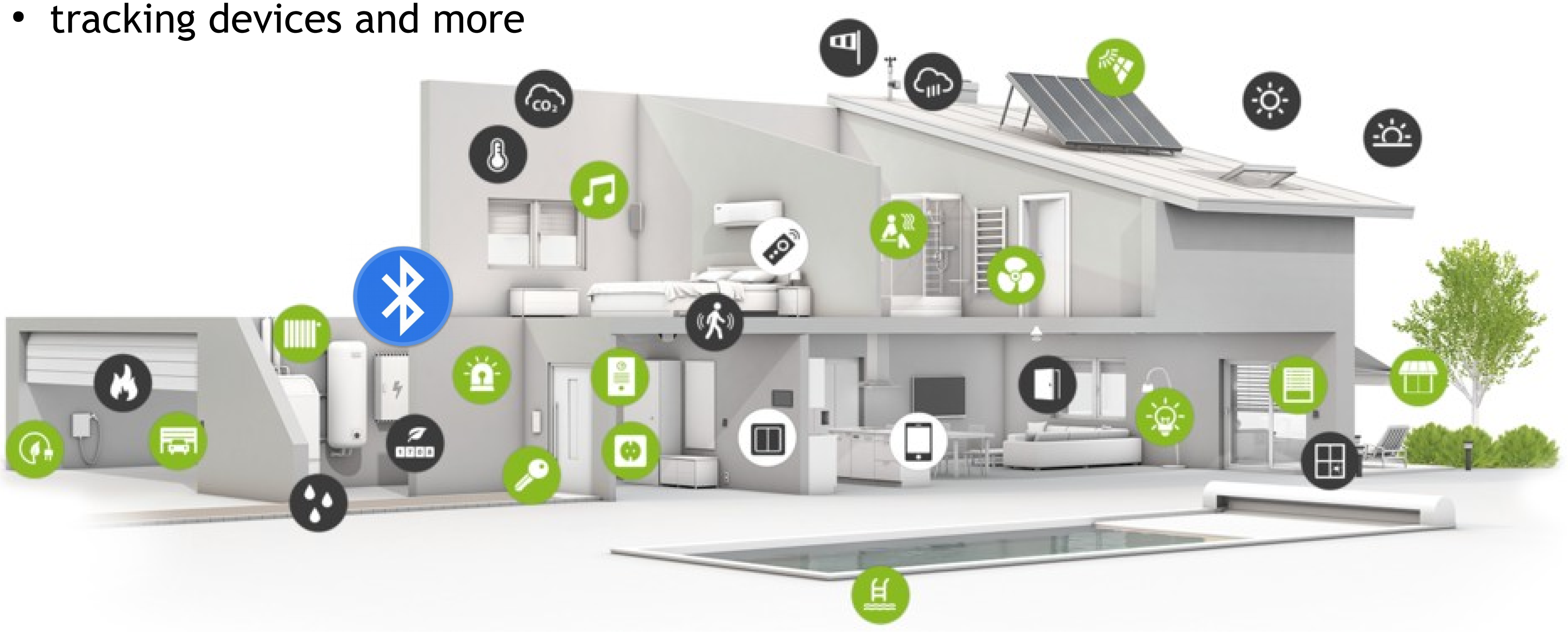
- The infotainment system in automotive application is good example

- Some example could be like
  - File transfers between two phones/pcs
  - And some we saw in the previous slides

Communication Protocols II
Bluetooth – Introduction – Applications

Access Point

- As the future belong to the IoT Bluetooth Special Interest Group showed off a bunch of upcoming smart home products that will use the wireless standard with
  - light bulbs
  - home hubs
  - tracking devices and more



Bluetooth – Introduction - Class

Class	Maximum Permitted Power (Milli Watts)	Approximate Range (Meter(s))
1	100	100
2	2.5	10
3	1	1

Communication Protocols II

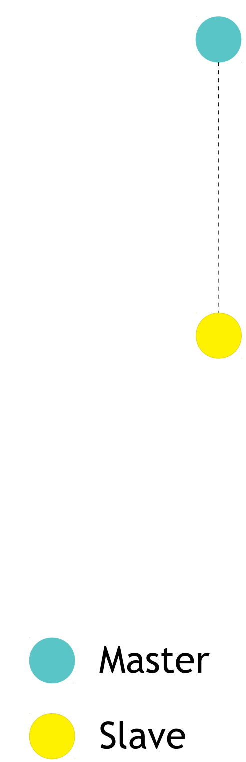
Bluetooth – Introduction

- Uses frequency hoping spread spectrum (FHSS)
- Omni directional, no requiring line of sight
- Bluetooth offers data speeds of up to 1 Mbps up to 10 meters (Short range wireless radio technology)
- Unlike IrDA, Bluetooth supports a LAN-like mode where multiple devices can interact with each other.
- The key limitations of Bluetooth are security and interference with wireless LANs.
- Short range wireless radio technology

Communication Protocols II

Bluetooth – Topology – Point to point

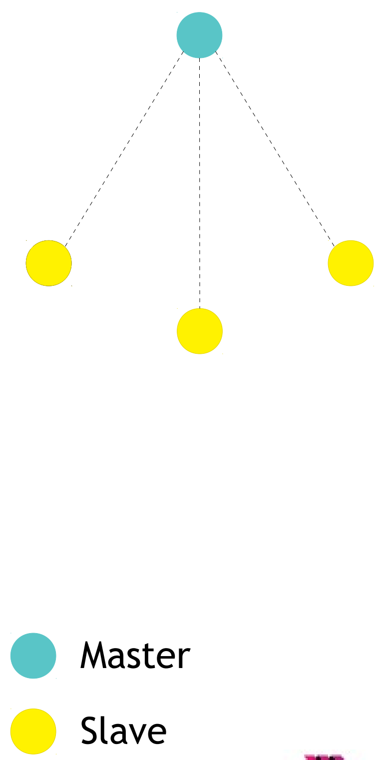
- For establishing one-to-one (1:1) device communications.
- The point-to-point topology available on Bluetooth Basic Rate/Enhanced Data Rate (BR/EDR) is optimized for audio streaming and is ideally suited for a wide range of wireless devices, such as speakers, headsets, and hands-free car kits.
- In Bluetooth Low Energy (LE), it is optimized for data transfer and is well suited for connected device products, such as fitness trackers, health monitors, and PC peripherals and accessories.



Communication Protocols II

Bluetooth – Topology – Broadcast

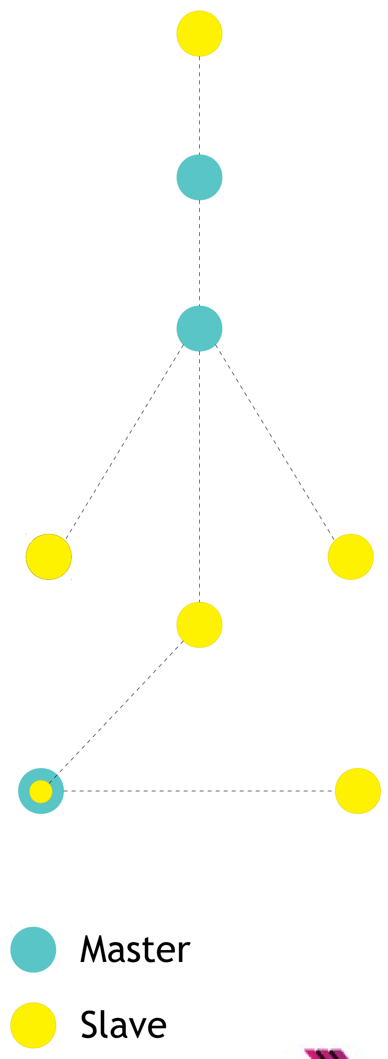
- For establishing one-to-many (1:m) device communications.
- In Bluetooth LE, it is optimized for localized information sharing and is ideal for location services such as retail point-of-interest information, indoor navigation and way finding, as well as item and asset tracking.



Communication Protocols II

Bluetooth – Topology – Mesh

- For establishing many-to-many (m:m) device communications.
- In Bluetooth LE, it enables the creation of large-scale device networks and is ideally suited for control, monitoring, and automation systems where tens, hundreds, or thousands of devices need to reliably and securely communicate with one another.



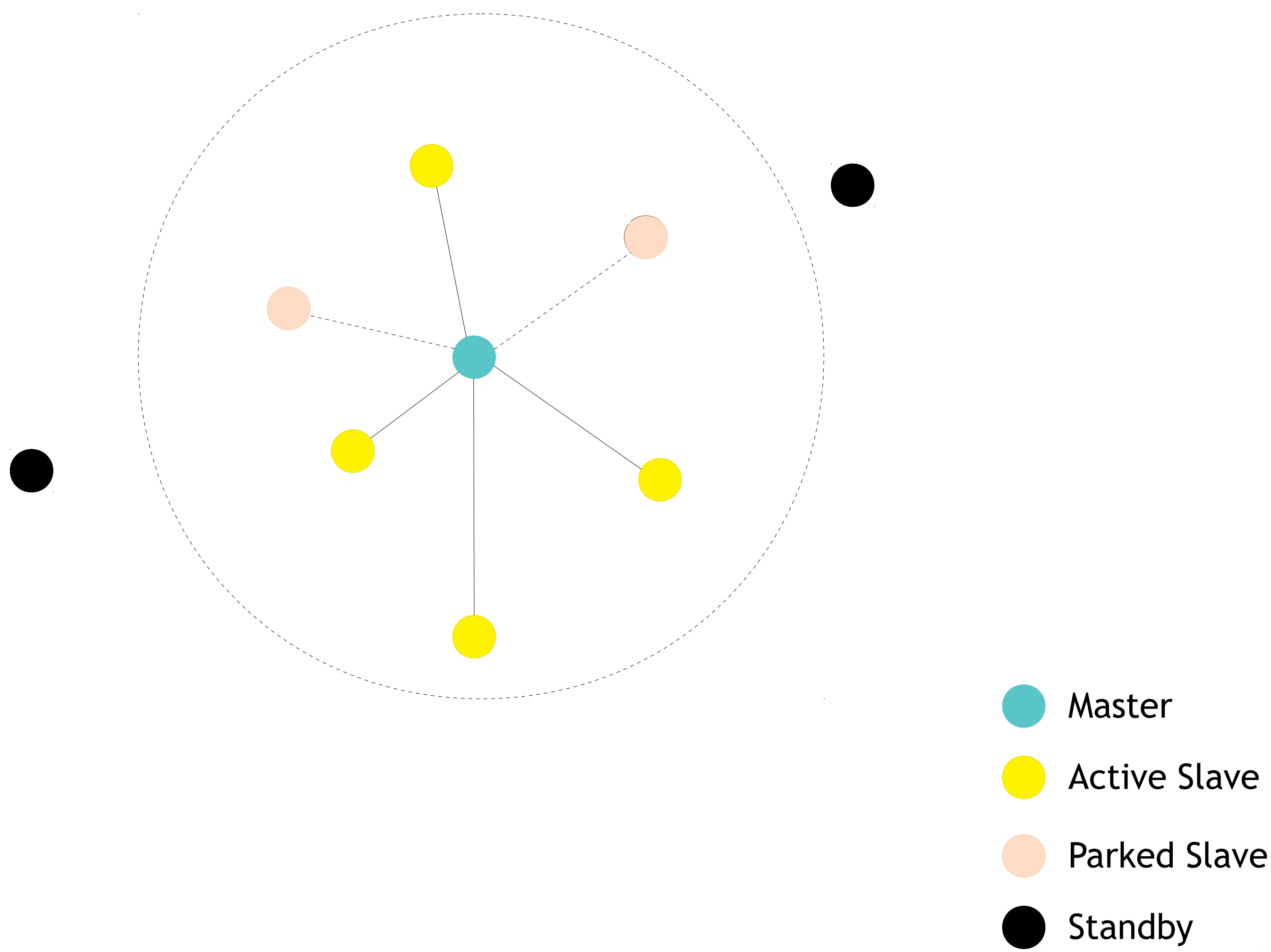
Communication Protocols II

Bluetooth – Topology – Piconet

- Ad-hoc network of devices with one master which can interconnect with up to seven active slave devices forming total 8 devices per network
- Up to 255 further slave devices can be inactive, or parked, which the master device can bring into active status at any time.

Communication Protocols II

Bluetooth – Topology – Piconet



Communication Protocols II

Bluetooth – Topology – Scatter Net

- Interconnection of 2 or more piconets
- Interconnected piconets that supports communication between more than 8 devices.
- Scatternets can be formed when a member of one piconet (either the master or one of the slaves) elects to participate as a slave in a second, separate piconet
- The device participating in both piconets can relay data between members of both ad hoc networks
- However, the basic Bluetooth protocol does not support this relaying - the host software of each device would need to manage it

Communication Protocols II

Bluetooth – Topology – Point to be noted

- Devices can automatically locate each other
- Master controls and setup the network
- One master per Piconet
- A device can’t be masters for two piconets
- The slave of one piconet can be the master of another piconet
- All devices operate on the same channel and follow the same frequency hopping sequence
- Two or more piconet interconnected to form a scatter net

Communication Protocols II

Bluetooth – Topology – Point to be noted

- Devices participating in scatter net may act as gateway
- Salves notify the master before going to parked mode

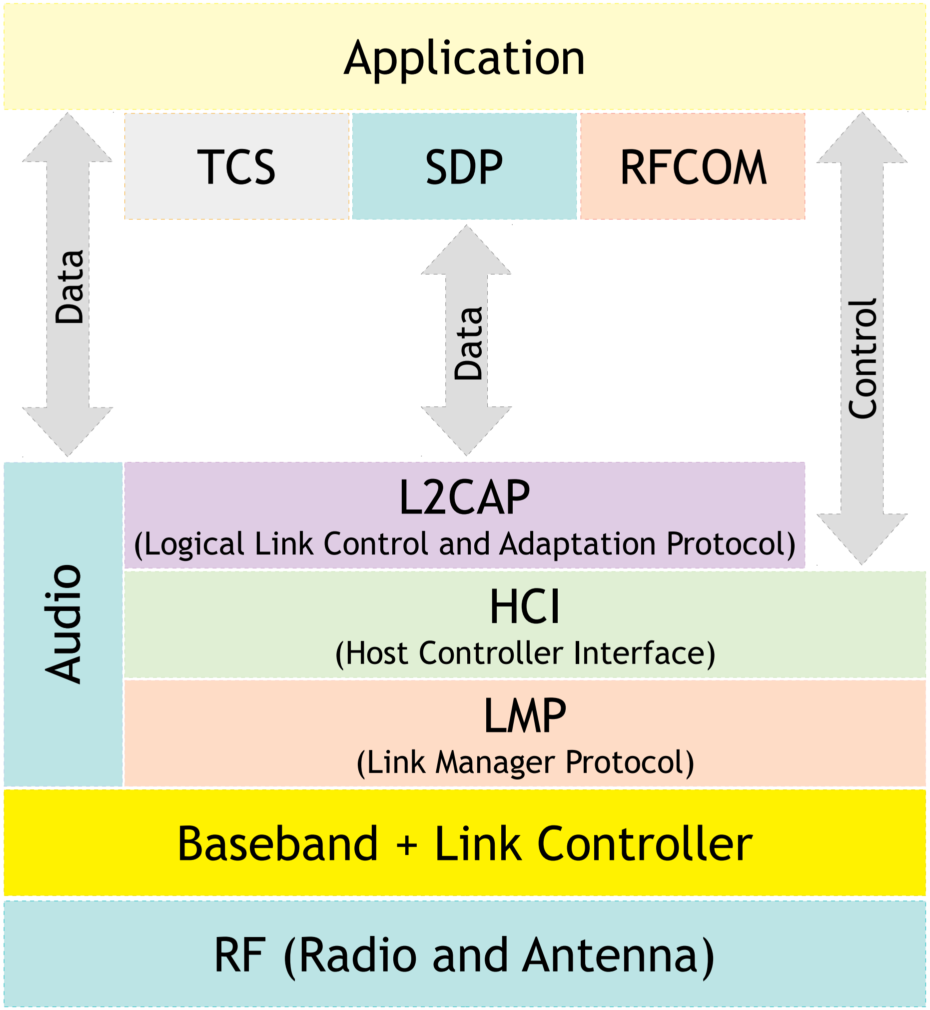
Communication Protocols II

Bluetooth – Versions

- Bluetooth 1.0 & 1.0B – Non interoperable, Mandated BD_ADDR
- Bluetooth 1.1 - Ratified as IEEE standard 802.15.1-2002
- Bluetooth 1.2 - Faster connection and discovery
- Bluetooth 2.0 + EDR - Enhanced Data Rate
- Bluetooth 2.1 - Secure Simple Pairing - SSP
- Bluetooth 3.0 - High speed data transfer
- Bluetooth 4.0 + LE - Low Energy consumption
- Bluetooth 4.1 - Incremental software update to 4.0
- Bluetooth 4.2 - Introduces features for the IoT
- Bluetooth 5 – Focus on emerging IoT technologies

Communication Protocols II

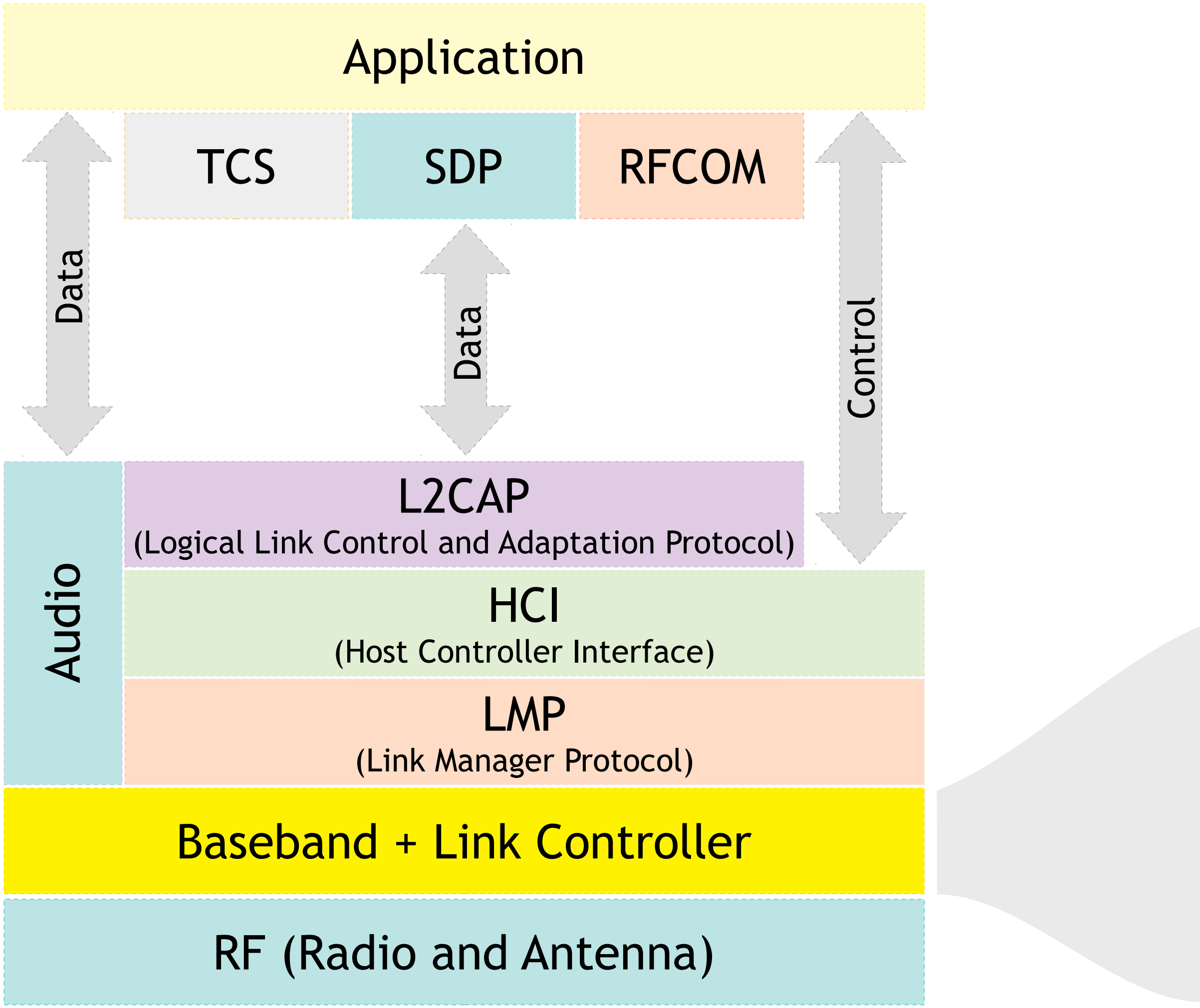
Bluetooth – Protocol Stack



- Defines the requirements for a Bluetooth transceiver operating in the 2.4 GHz ISM band
- Modulation is GFSK (Gaussian Frequency Shift Keying) with gross bit rate of 1Mbps
- 1600 hops/sec (625 μsec) frequency hopper
- 79 One MHz channels
- Time Division Duplex

Communication Protocols II

Bluetooth – Protocol Stack

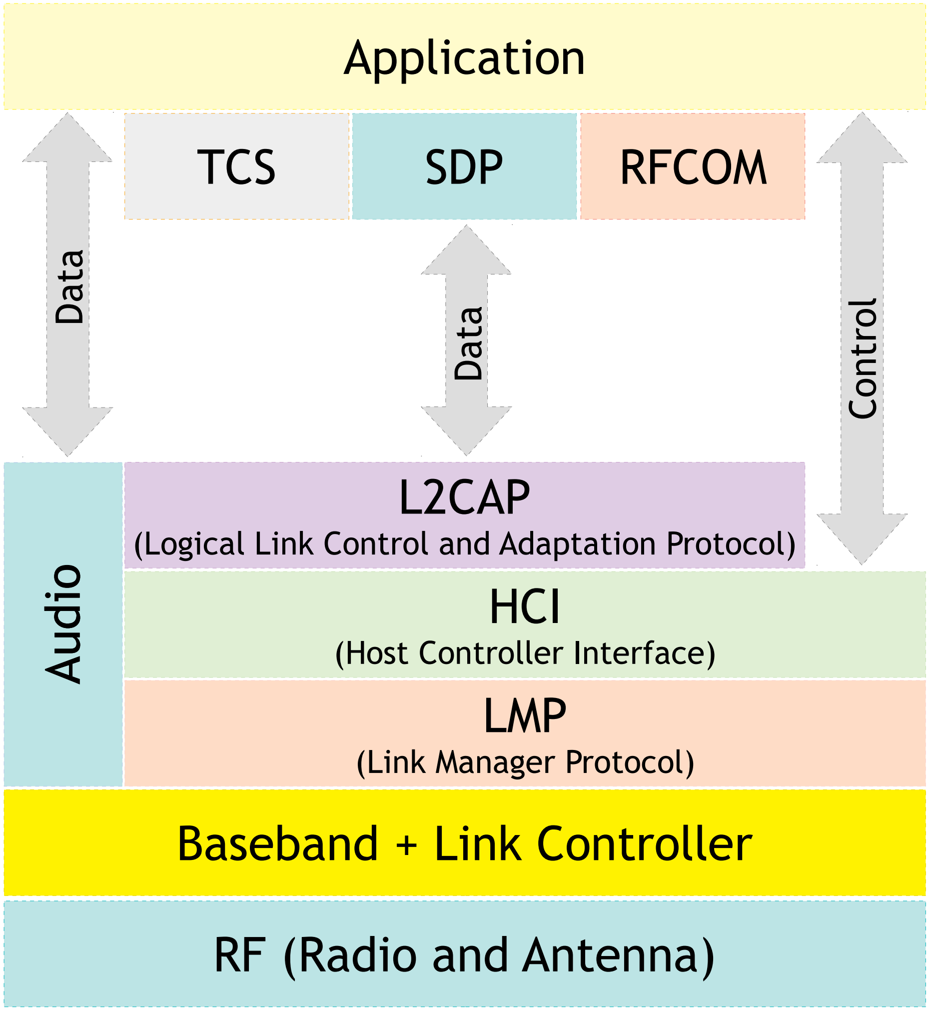


- Addressing
- Channel Establishment
  - Inquiry and Paging
  - Attach and Detach
- Error detection and correction
- Synchrnous Connection Oriented **(SCO)** or Asynchrnous Connection Less **(ACL)**

Physical Layer

Communication Protocols II

Bluetooth – Protocol Stack



Addressing

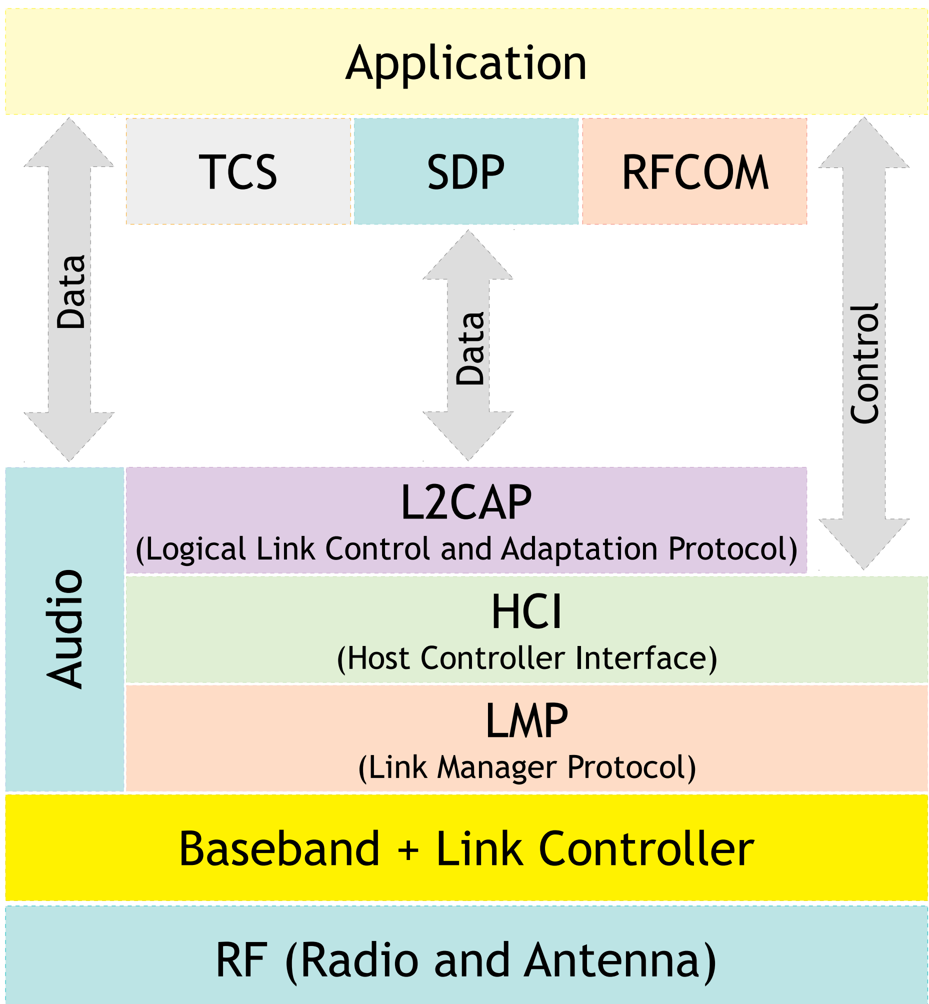
- Bluetooth Device Address (BD_ADDR)
  - 48 Bit IEEE MAC Address
- Active Member Address (AM_ADDR)
  - 3 Bit Active Slave Address
  - All 0s - Broadcast Address
- Parked Member Address (PM_ADDR)
  - 8 Bit Parked Slave Address
- Access Request Addres (AR_ADDR)
  - Used by the parked slave

MAC Address

- Non-significant Address Part (NAP)
  - Used for encryption seed
- Upper Address Part (UAP)
  - Used for error correction seed initialization and frequency hop sequence generation
- Lower Address Part (LAP)
  - Used for FH sequence generation

Communication Protocols II

Bluetooth – Protocol Stack



Piconet Management

- Channel Control
  - Managed by master
    - Polling
  - Master initiates the connection
- Master/Slave Switch

Link Config

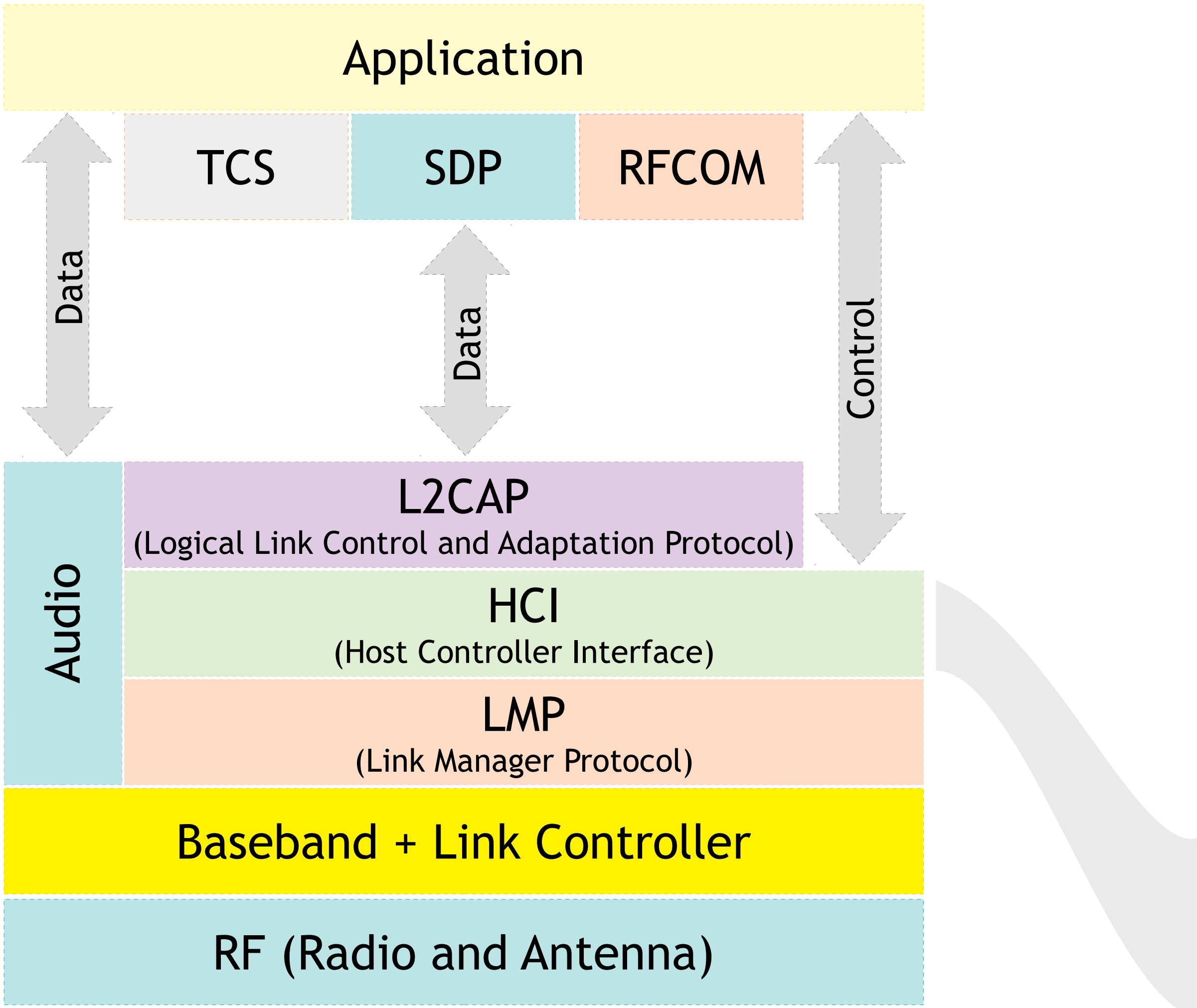
- Low power modes
- Quality of Service (QoS)
- Packet Type Selection

Security

- Authentication
- Encryption

Communication Protocols II

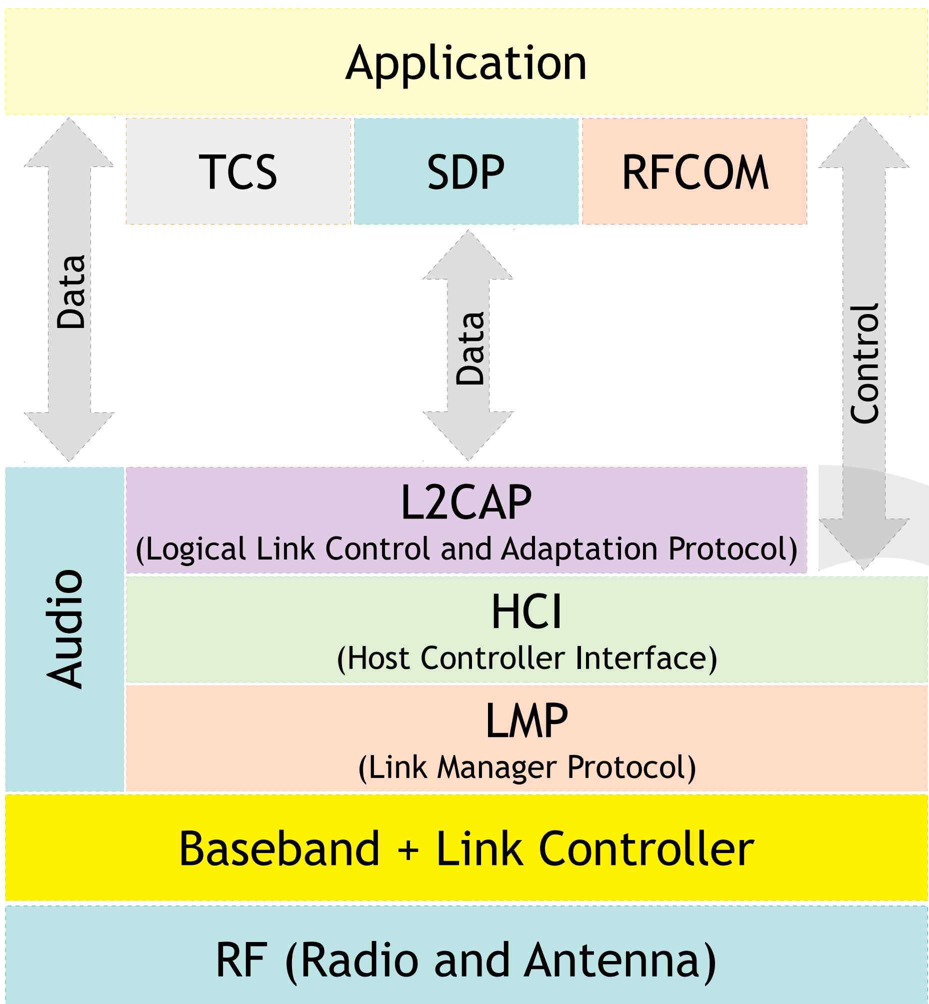
Bluetooth – Protocol Stack



- Provides a command interface to the Baseband Link Controller and Link Manager, and access to hardware status and control registers

Communication Protocols II

Bluetooth – Protocol Stack

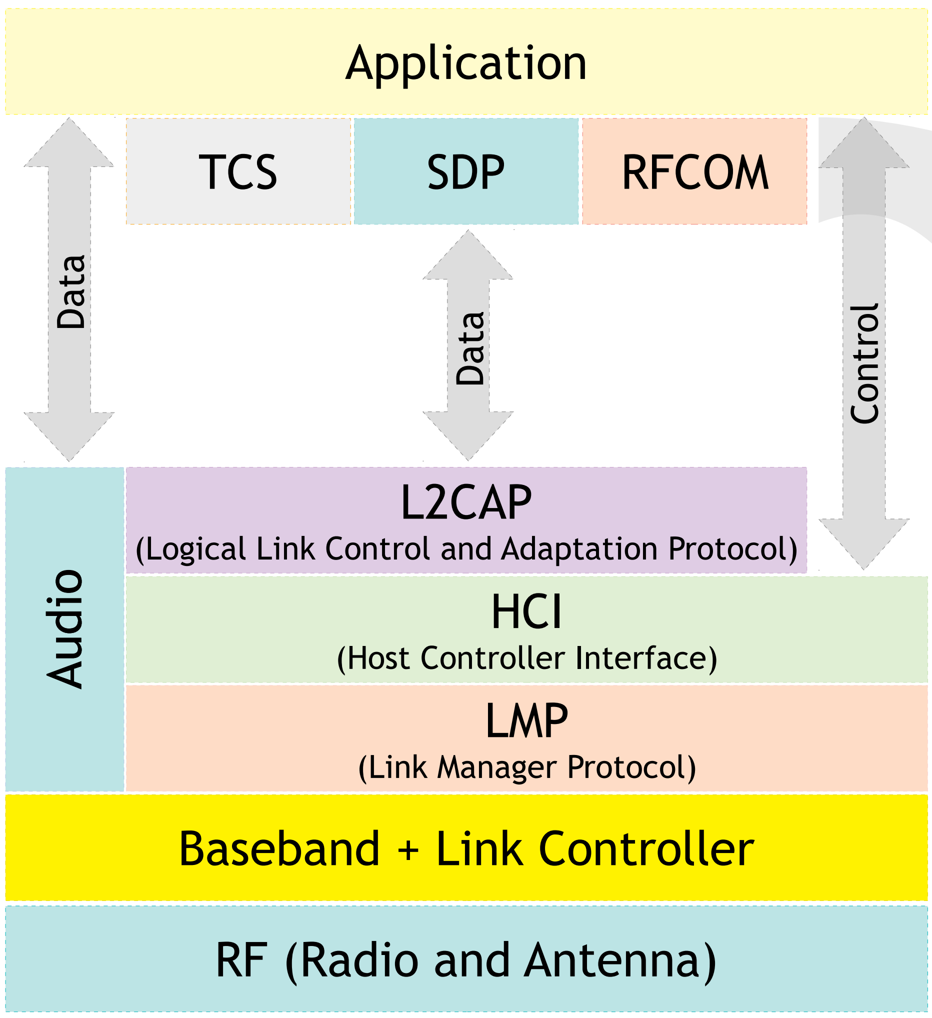


L2CAP Layer
Service provider to the higher layer

- Provides connection-oriented and connection less data services
- Protocol multiplexing and demultiplexing capabilities
- Segmentation & reassembly of large packets
- Provides data packets up to 64 kilobytes length

Communication Protocols II

Bluetooth – Protocol Stack



Middleware Protocol Group

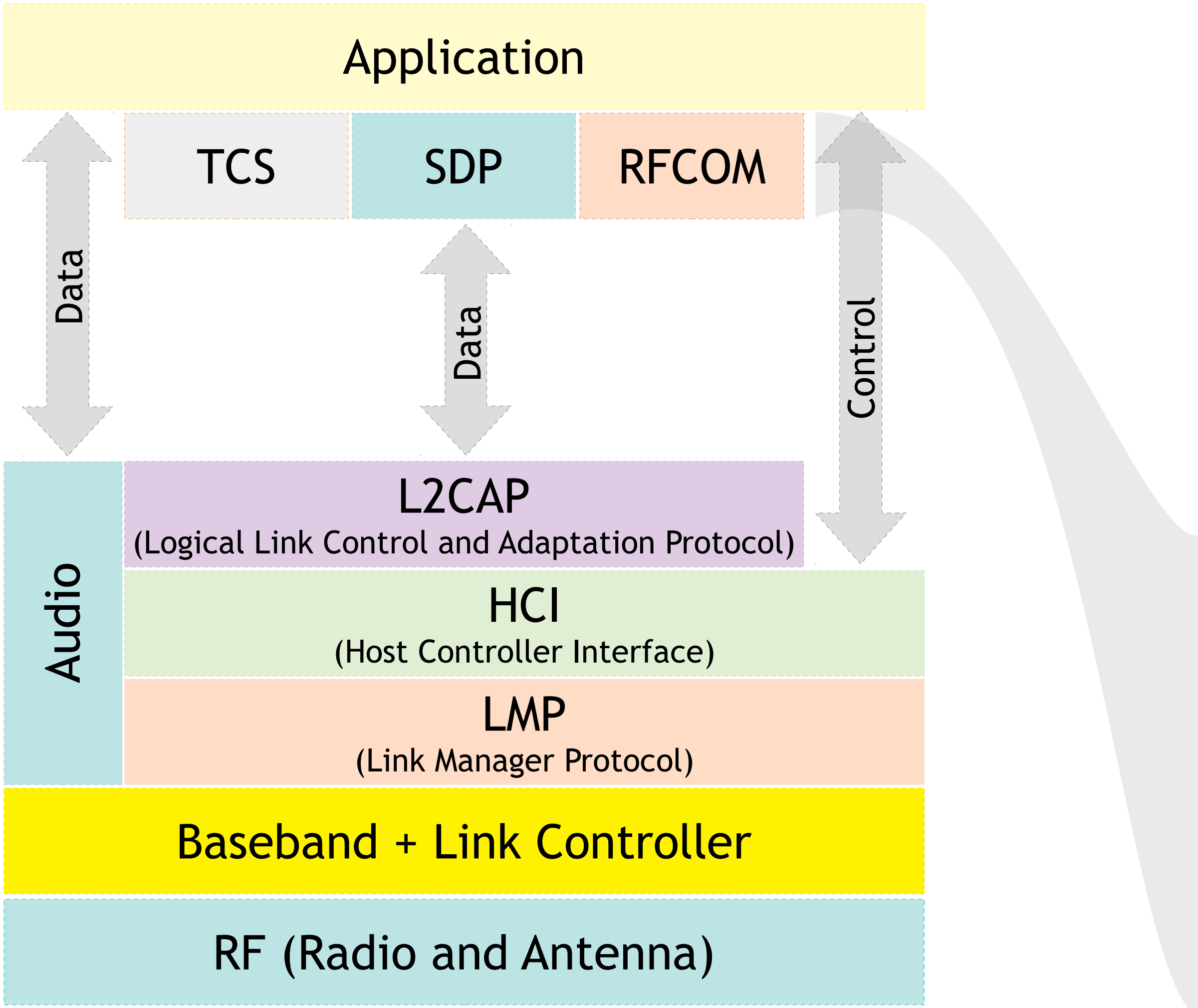
- Additional transport protocols to allow existing and new applications to operate over Bluetooth.

Radio Frequency Comm (RFCOM)

- Most important layer in the Bluetooth architecture.
- Takes care of the communication channel between two devices or between a master and a slave. It connects the serial ports of all the devices according to the requirement.
- Has to accommodate two kinds of devices:
  - Communication end-points such as computers or printers.
  - Devices that are a part of communication channel such as Modems.
- Emulation of Serial Port over wireless network

Communication Protocols II

Bluetooth – Protocol Stack



Middleware Protocol Group

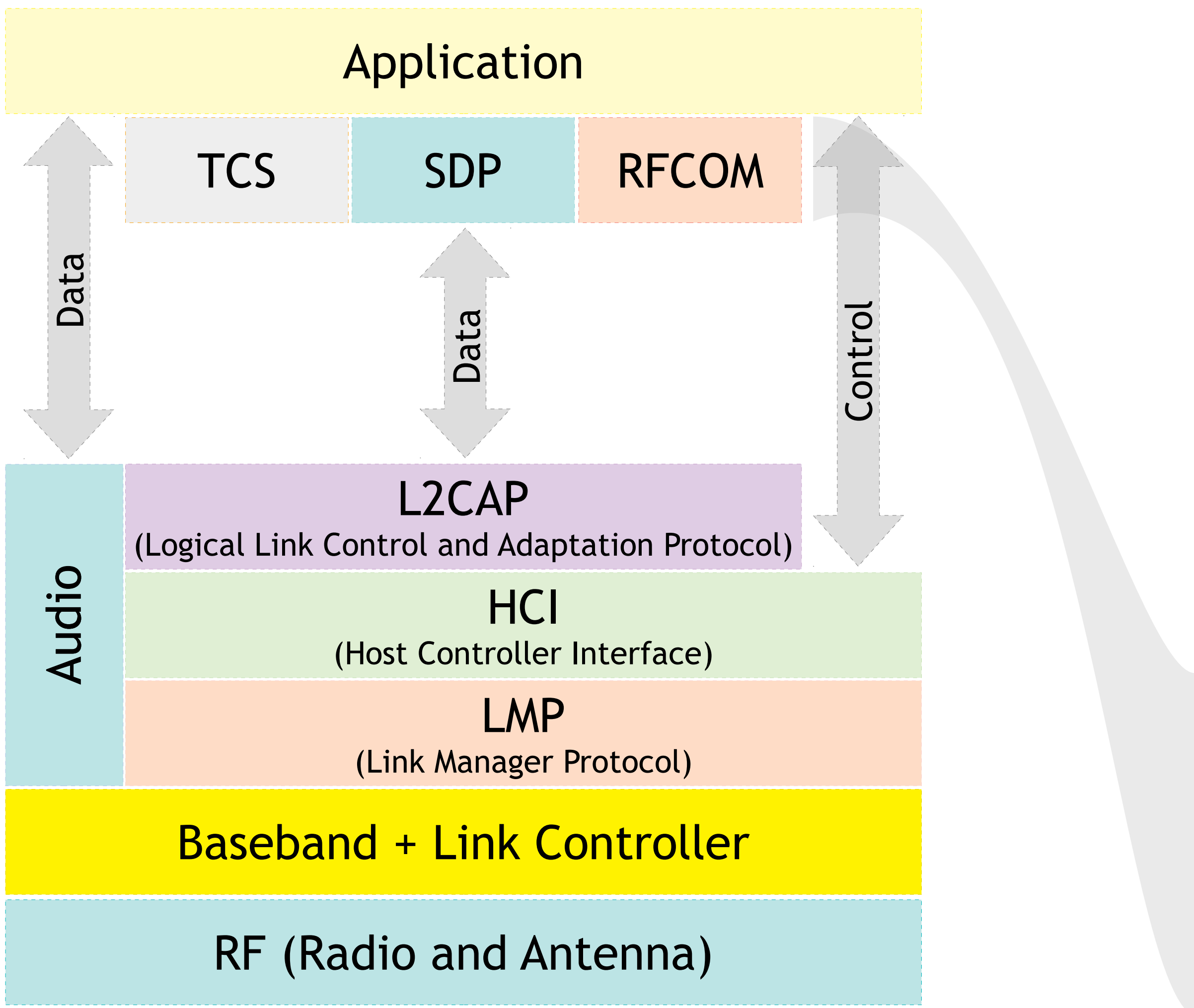
- Additional transport protocols to allow existing and new applications to operate over Bluetooth.

Service Discovery Protocol (SDP)

- Provides a means for applications to discover which services are available and to determine the characteristics of those available services
- Intended to address the unique characteristics of the Bluetooth environment.

Communication Protocols II

Bluetooth – Protocol Stack



Middleware Protocol Group

- Additional transport protocols to allow existing and new applications to operate over Bluetooth.

Telephony Control Protocol Spec (TCS)
Basic function of this layer is call control (setup & release) and group management for gateway serving multiple devices.

Communication Protocols II

Bluetooth – Specification

- Bluetooth® specifications define the technology building blocks that developers use to create the interoperable devices that make up the thriving Bluetooth ecosystem.
- Bluetooth specifications are overseen by the Bluetooth Special Interest Group (SIG) and are regularly updated and enhanced by Bluetooth SIG Working Groups to meet evolving technology and market needs.
- Includes a profile document with a template to ensure a common structure

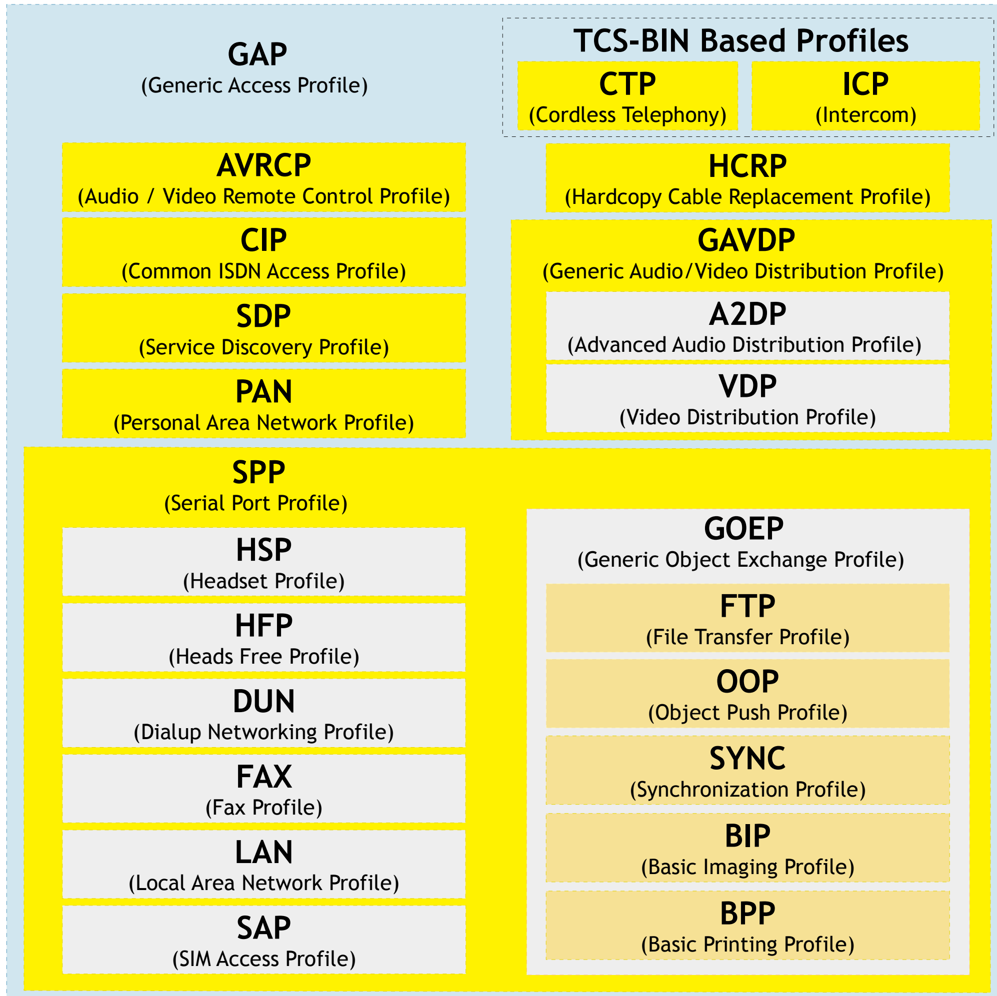
Communication Protocols II

Bluetooth – Profiles

- Can be seen as a wireless interface specification for communication between Bluetooth devices
- Describes the application-level usage models and their implementation, needed for interoperability reasons
  - A Bluetooth headset from vendor X will work with a smartphone vendor Y
- Interoperability on different levels
  - Radio: Devices can get in contact with each other
  - Protocol: Devices can communicate with each other
  - Usage: Devices can execute applications together an meet end-users’ expectations

Communication Protocols II

Bluetooth – Profiles



Communication Protocols II

Bluetooth – Profiles - GAP

- Basic profile – All other profiles are built upon it and use its facilities
- Ensures that all devices can successfully establish a baseband link
  - Minimum conformance requirement for devices
  - Generic Procedures for Discovering devices
  - Link Management Facilities for connection to devices
  - Naming Conventions
  - Modes of Operation

Communication Protocols II

Bluetooth – Profiles – GAP - Mode(s)

- Discovery
  - Governs the use of inquiry scan and whether other devices can discover a Bluetooth device when it comes within their area of radio coverage.
    - Non-Discoverable
    - Limited-Discoverable
    - General-Discoverable

Communication Protocols II

Bluetooth – Profiles – GAP - Mode(s)

- Connection
  - Governs the use of page scan and whether other devices can connect to a Bluetooth device when it comes within their area of radio coverage
    - Non-Connectable
    - Direct-Connectable
    - Undirect-Connectable

Communication Protocols II

Bluetooth – Profiles – GAP - Mode(s)

- Security
  - Mode 1
    - Level 1 - No Security (No authentication and no encryption)
    - Level 2 - Unauthenticated pairing with encryption
    - Level 3 - Authenticated pairing with encryption
    - Level 4 - Authenticated LE Secure Connections pairing with encryption
  - Mode 2
    - Level 1 - Unauthenticated pairing with data signing
    - Level 2 - Authenticated pairing with data signing

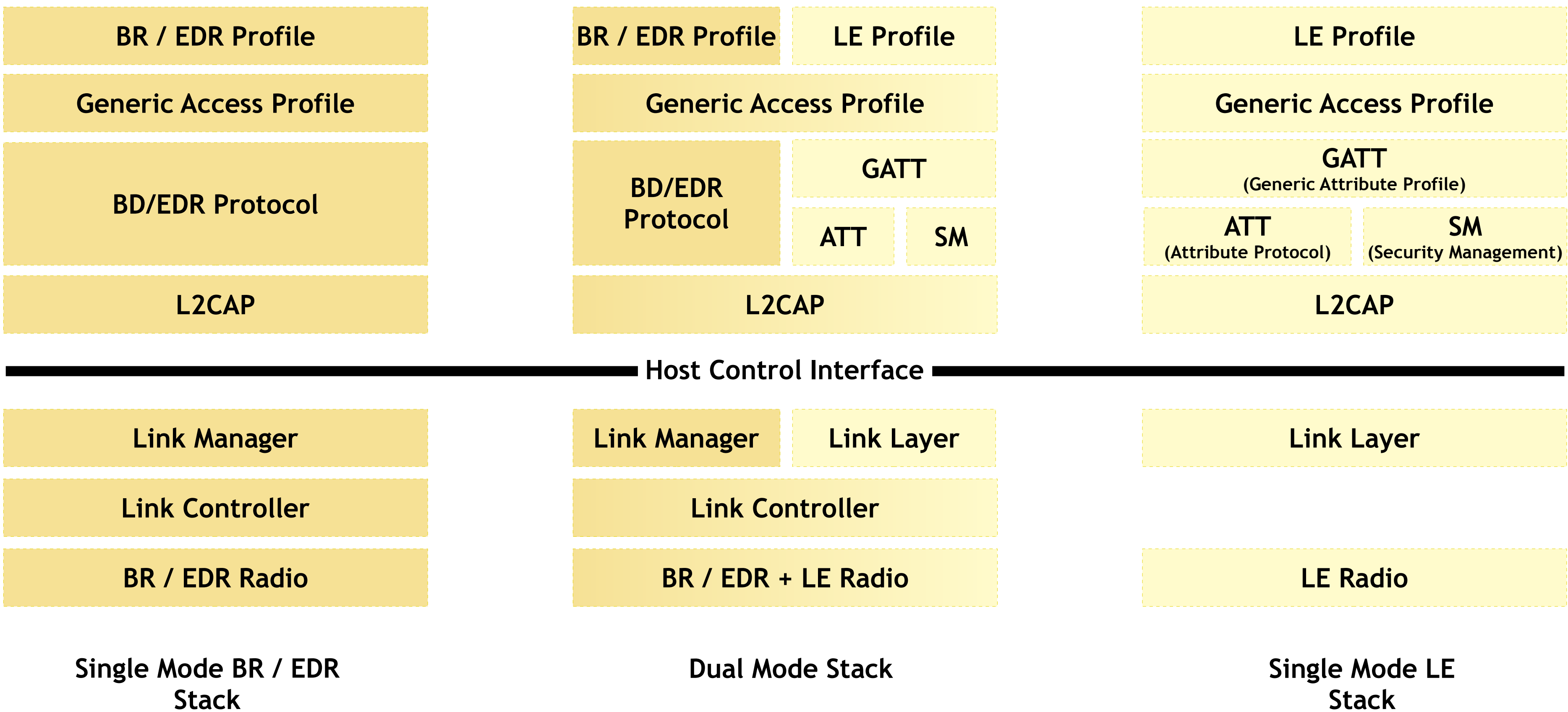
Communication Protocols II

Bluetooth – Profiles – GAP - Mode(s)

- Pairing
  - Governs the use of the link manager's pairing facilities, which are used to create link keys for use on encrypted links
    - Non-Bondable
    - Bondable
  - A pairing procedure involves an exchange of Security Manager Protocol packets to generate a temporary encryption key called the Short Term Key (STK)

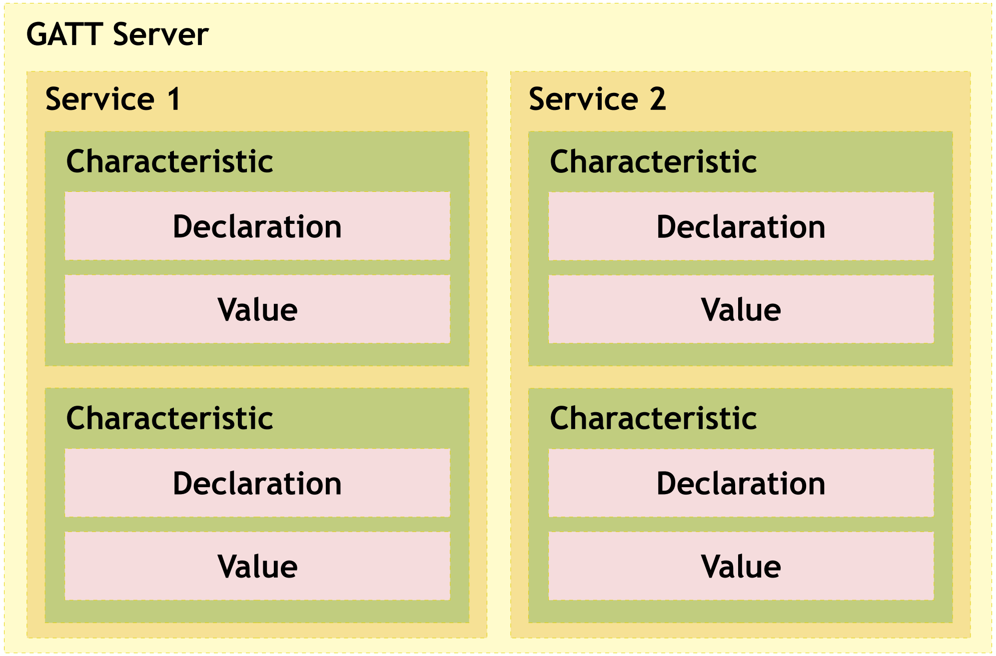
Communication Protocols II

Bluetooth – Profiles - GATT



Communication Protocols II

Bluetooth – Profiles – GATT Server



Thank You

Communication Protocols II
References

- An Introduction® to Wi-Fi – Rabbit Product Manual
- https://dot11ap.wordpress.com/802-11-frame-format-and-types/
- http://microchipdeveloper.com/wireless:ble-gap-modes-procedures

Communication Protocols II
Differences

	Wi-Fi IEEE 802.11b	Bluetooth IEEE 802.15.1	ZigBee IEEE 802.15.4
Radio	Direct Sequence Spread Spectrum DSSS	Frequency Hopping Spread Spectrum FHSS	Direct Sequence Spread Spectrum DSSS
Data rate	11 Mbps	1 Mbps	250 kbps
Nodes per master	32	7	64,000
Slave enumeration latency	up to 3 s	up to 10 s	30 ms
Data type	video, audio, graphics, pictures, files	audio, graphics, pictures, files	small data packet
Range [m]	100	10	10 - 100
Extendibility	roaming possible	no	yes
Complexity	Complex	very complex	simple
Positioning technology	CoO, (tri)lateration, fingerprinting	CoO	(tri)lateration, fingerprinting

Communication Protocols II

Differences

IOT WIRELESS TECHNOLOGIES							
Technologies	Standards & Organizations	Network Type	Frequency (US)	Max Range	Max Data Rate	Max Power	Encryption
WiFi	IEEE 802.11 (a,b,g,n,ac,ad, and etc)	WLAN	2.4,3.6,5,60 GHz	100 m	"6-780 Mb/s 6.75 Gb/s @ 60 GHz"	1 W	WEP, WPA, WPA2
Z-Wave	Z-Wave	Mesh	908.42 MHz	30 m	100 kb/s	1 mW	Triple DES
Bluetooth	Bluetooth (formerly IEEE 802.15.1)	WPAN	2400-2483.5 MHz	100 m	1-3 Mb/s	1 W	56/128-bit
Bluetooth Smart (BLE)	IoT Interconnect	WPAN	2400-2483.5 MHz	35 m	1 Mb/s	10 mW	128-bit AES
Zigbee	IEEE 802.15.4	Mesh	2400-2483.5 MHz	160 m	250 kb/s	100 mW	128-bit AES
THREAD	IEEE 802.15.4 + 6LoWPAN	Mesh	2400-2483.5 MHz	160 m	250 kb/s	100 mW	128-bit AES
RFID	Many	P2P	13.56 MHz, etc.	1 m	423 kb/s	~1 mW	possible
NFC	ISO/IEC 13157 & etc	P2P	13.56 MHz	0.1 m	424 kb/s	1-2 mW	possible
GPRS (2G)	3GPP	GERAN	GSM 850/1900 MHz	25 km / 10 km	171 kb/s	2 W / 1 W	GEA2/GEA3/GEA4
EDGE (2G)	3GPP	GERAN	GSM 850/1900 MHz	26 km / 10 km	384 kb/s	3 W / 1 W	A5/4, A5/3
UMTS (3G) HSDPA/HSUPA	3GPP	UTRAN	850/1700/1900 MHz	27 km / 10 km	0.73-56 Mb/s	4 W / 1 W	USIM
LTE (4G)	3GPP	GERAN/UTRAN	700-2600 MHz	28 km / 10 km	0.1-1 Gb/s	5 W / 1 W	SNOW 3G Stream Cipher
ANT+	ANT+ Alliance	WSN	2.4 GHz	100 m	1 Mb/s	1 mW	AES-128
Cognitive Radio	IEEE 802.22 WG	WRAN	54-862 MHz	100 km	24 Mb/s	1 W	AES-GCM
Weightless-N/W	Weightless SIG	LPWAN	700/900 MHz	5 km	0.001-10 Mb/s	40 mW / 4 W	128-bit