Introduction to Internet of Things and Embedded Systems

by Kasun Vimukthi Jayalath
27.12.2024

Kasun Jayalath

Senior Electronics Engineer | Embedded Linux | Embedded Systems | Control Systems | Industrial Automation

Colombo, Western Province, Sri Lanka · **Contact info**

University of Moratuwa

Education

University of Moratuwa
Master's degree, Electronics and Automation
2017 - 2019

Electronic Control Systems, Autonomous Systems (Internet) and +1 skill

Post Viva Submission DESIGN AND IMPLEMENTATION OF AN AUTONOMOUS DRONE FOR HUMAN DETECTION.pdf
Research Thesis

University of Peradeniya
Bachelor of Science (B.Sc.), Electrical and Electronics Engineering
2011 - 2016

Activities and societies: Rotaract, Gavel, Tennis team, Art Circle, Robotics and Inventors club, Astronomy Association

I participated number of Robotics competitions and Exhibitions internally and outside from the university.Gained lot of experiences in the related ares like Programming, Assembling and Mechan …see more

Software Project Management, Electronic Control Systems and +5 skills

Senior Electronic Engineer
Vega Innovations · Full-time
May 2023 - Present · 1 yr 8 mos
Colombo, Western Province, Sri Lanka · On-site

Mechatronics, Team Leadership and +8 skills

Graduate Research Assistant
University at Albany, SUNY · Full-time
Aug 2022 - Sep 2023 · 1 yr 2 mos
United States

C++, C (Programming Language) and +3 skills

GitHub - kasunvj/Radiation-Patterns-RIS
GitHub repo for the RIS Simulator

Research And Development Engineer
Sri Lanka Telecom · Full-time
May 2021 - Sep 2023 · 2 yrs 5 mos
Colombo, Western, Sri Lanka

Internet of Things (IoT), Machine Learning and +12 skills


Automation Engineer
MAS Holdings · Full-time
Jun 2017 - May 2021 · 4 yrs
Colombo

PLC Programming, Electrical Engineering and +13 skills


Content

1. The Beginning: The Industrial Revolution
2. The Things
3. The Internet
4. The Internet of Things
5. Sensors

Activity 1: 'A thing' that collects data and displays

6. Communication Protocols
7. Microcontrollers
8. Microprocessors
9. Case study: Smart Traffic Management System
10. Codebase management

11. Version Control System
12. Git and GitHub
Activity 2: A sustainable development

13. Programming Microcontrollers/Microprocessors
Activity 3: Programming Exercise

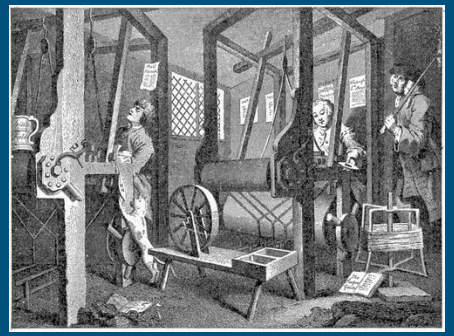
14. Websockets
15. MQTT
Activity 4: Sockets and Brokers

16. Cloud Computing for IOT
17. AWS
Activity 5 : `Internet of a Thing` Controlling Remotely

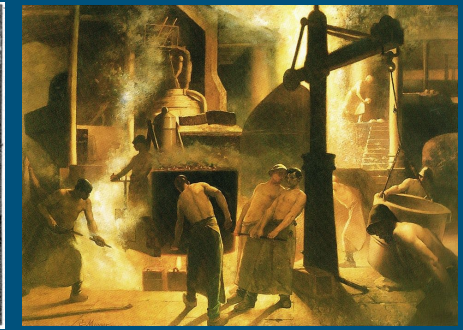
1. The Beginning: The Industrial Revolution

First Industrial revolution (1760-1840)

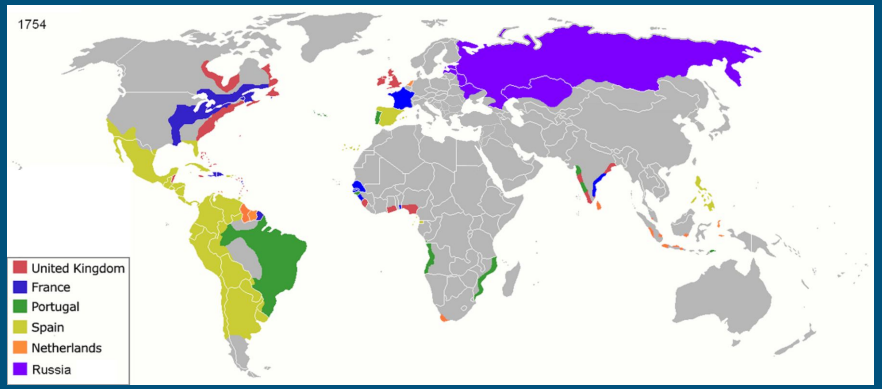
- Hand productions to machines. Tool production, textile industry, agricultural industry and factories emerged.
- Even Though machines undertook the work, still large labour force is required to operate machines. James Watts steam engine was developed in 1770, and even electricity was not commercial until 1800.
- Mail services are the only reliable communication method back then.



Weaving machine in 18XX >



Steel production >



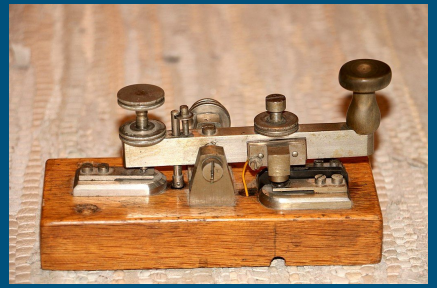
European colonial empires at the start of the Industrial Revolution, superimposed upon modern political boundaries

Second Industrial revolution (1870-1940)

During the 1st and 2nd world war.

Telegraph expansion.

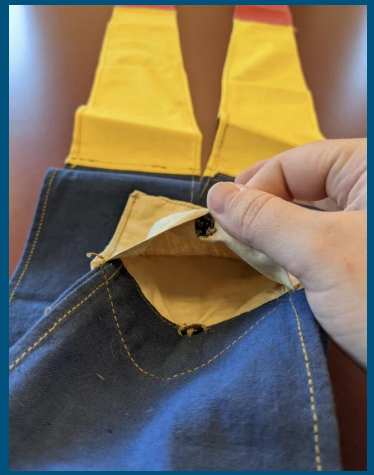
Radio was not invented, the communication between things on ground happens by wire and telegraph. Ground and air communication happens by dropping “Message streamers”. They hold a pocket that has information in it.



Telegraph machine that used Morse Code >

WW1 aircraft >

Message streamers used in ww1 >



Third Industrial revolution (1947-2011) - The information age

Development of Transistor and OpAmp (1947-1957)

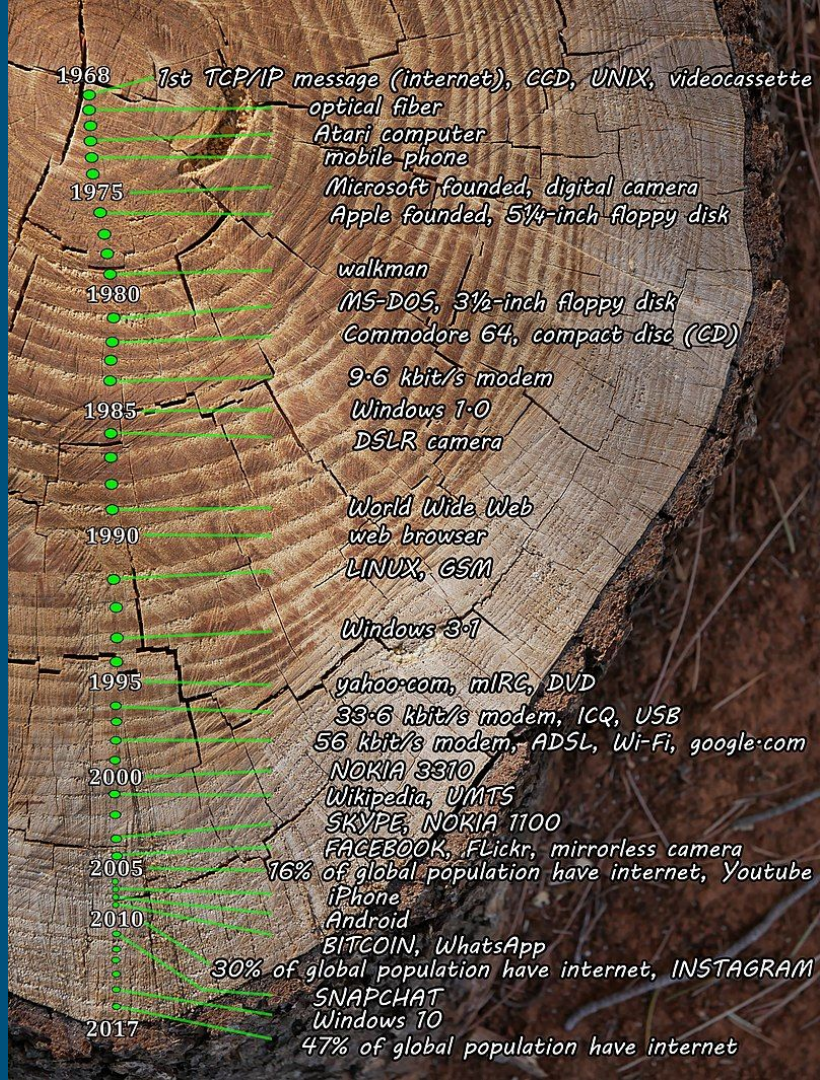
Developments in personal computer

Invention of the internet (1969-1989)

Web 1.0, 1991, mainstream internet that has access only to government and Universities.

Web 2.0 Social media, smartphones and digital TV

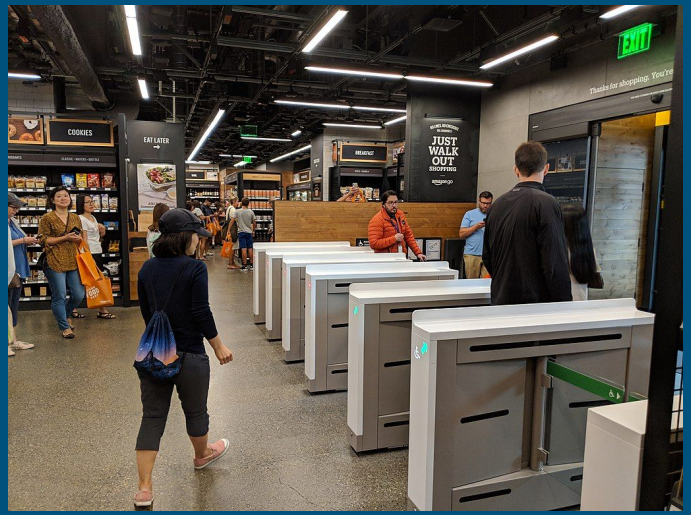
Developments of digital devices such as, mobile phones, cameras and etc



Fourth Industrial revolution (2020 onwards) - The Imaginative age

- Industry 4.0
  - M2M Communication, Full robotic and automation production , Smart factory
- Artificial Intelligence
- Gene Editing
  - Editing a living cell. Creation of new medicine, agricultural products
- Metaverse, The change in the way we experience the world.
- Internet of Things
  - Computer Vision
  - Deeplearning
  - Sensor fusion

IoT in a battle field >



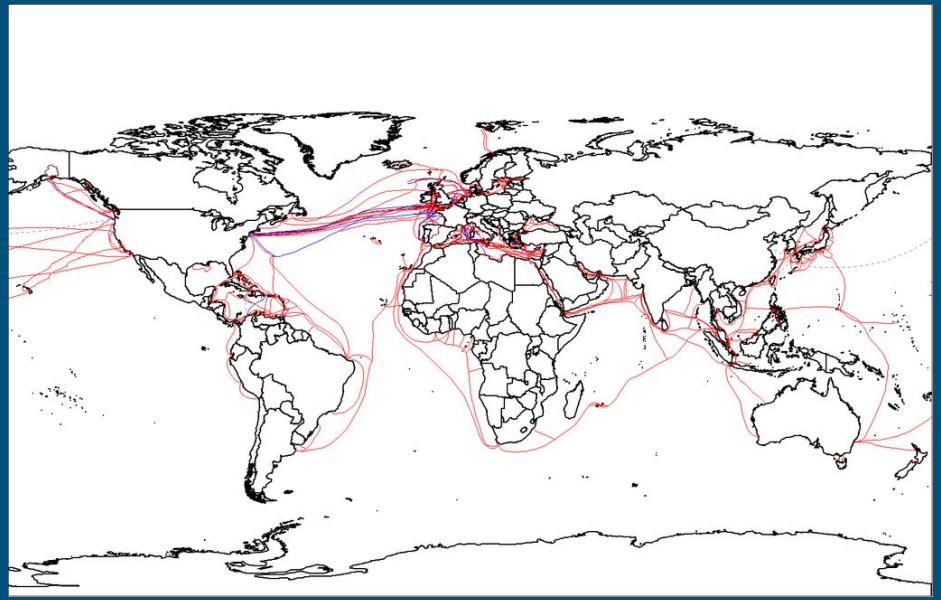
Amazon Go, a cashierless store >

2. The Things

- Printer
  Printers were used to work as standalone machines. Either it is connected to PC or need to connect a USB to get a printout
- Lighting Systems
  Lights were either on or off, controlled by switches.
- Health Monitors
  Used in clinics or homes to get a one-time reading.
- Agricultural Equipment
  Farmers manually watered fields or used timers
- Wearables
  Watches simply told time.

3. The Internet

- Internet was developed in 1970 and WWW become public in 1990.
- Make information and information in devices accessible by far distances in a matter of milliseconds.
- Nowadays access to such information is effortless.
- Due to the lower cost and speed distributed information processing become popular.
- Increase in edge devices that collects data

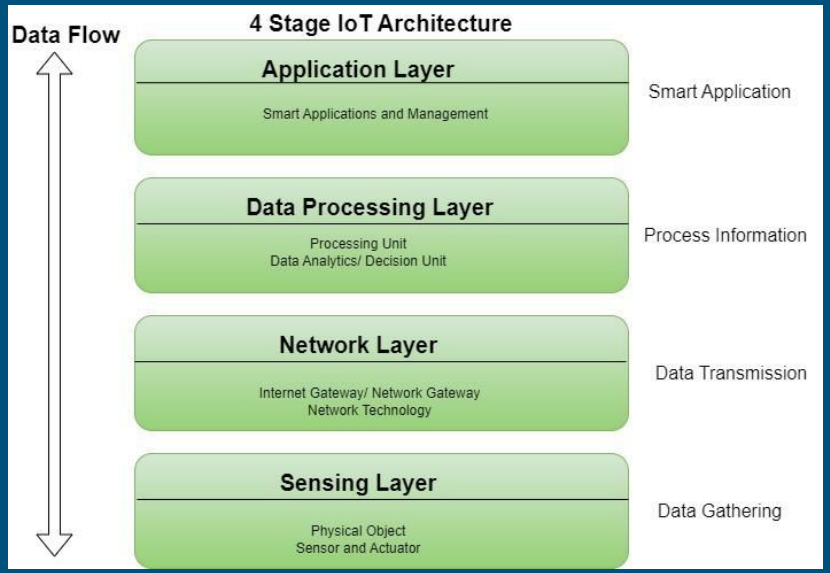


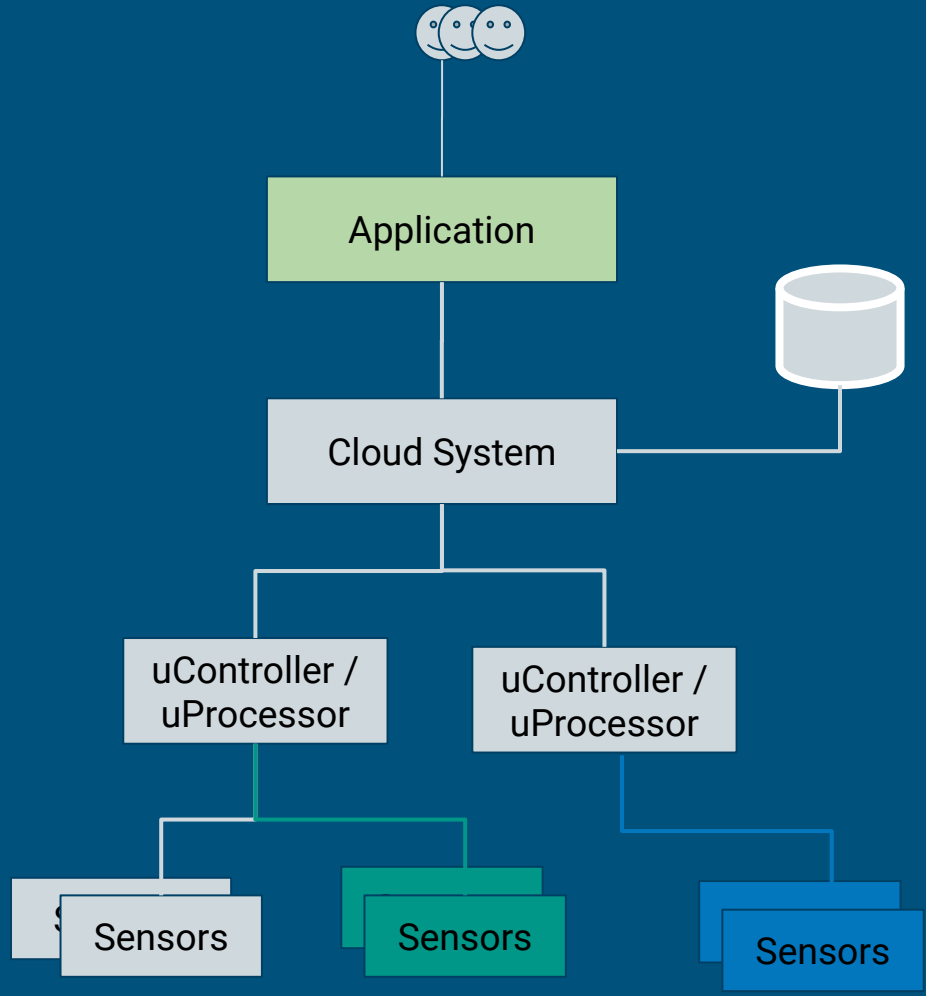
2007 map showing submarine fiber optic telecommunication cables around the world

4. The Internet of Things

- Wikipedia definition,
  Internet of things (IoT) describes devices with sensors, processing ability, software and other technologies that connect and exchange data with other devices and systems over the Internet or other communication networks.

- **Printer** → Connect to internet(local network) : Enable executing a printing job from anywhere
- **Lighting systems** → Connect to home network : Enable monitoring and operating remotely
- **Health monitors** → Connect to internet : Enable monitoring elderly at home
- **Agricultural ->** Connect sensors(humidity, temp, ph and etc) to the internet


Data Flow
4 Stage IoT Architecture
Application Layer
Smart Applications and Management
Smart Application
Data Processing Layer
Processing Unit
Data Analytics/ Decision Unit
Process Information
Network Layer
Internet Gateway/ Network Gateway
Network Technology
Data Transmission
Sensing Layer
Physical Object
Sensor and Actuator
Data Gathering


Application
Cloud System
uController / uProcessor
uController / uProcessor
Sensors
Sensors
Sensors

Let's start from bottom up.. Connecting with the environment

Light
Sound
Temperature
Pressure
Touch
Smell

Sensors

ucontroller

GPIO

SPI/I2C/UART/CAN

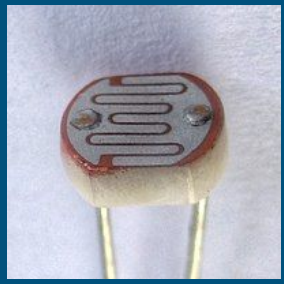
Wifi/LoRa/Bluetooth

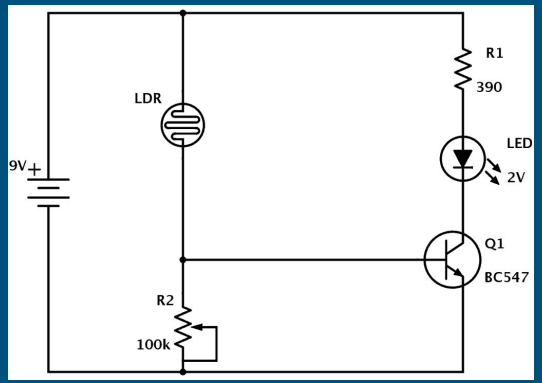
5. Sensors

A sensor is a device that produces an output signal for the purpose of detecting a changes in the environment.

Light Dependent Resistor (LDR)

- Photoconductivity : material becomes more electrically conductive for EM radiation
- Decrease resistance for increase in luminosity
- Cadmium Sulfide semiconductor
- What is the electrical parameter that changes which trigger the transistor.



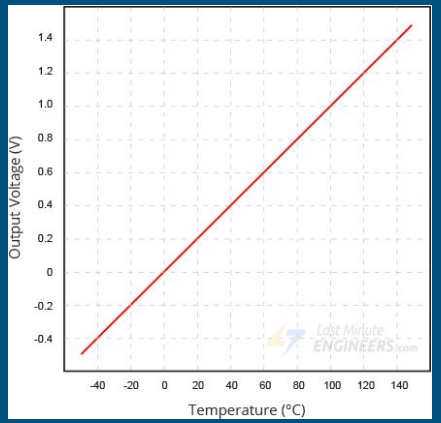


Temperature sensor

$$\text{Temperature}\,(^\circ\text{C}) = \frac{V_{out}}{10\,\text{mV}/^\circ\text{C}}$$

- LM35
- semiconductor temperature-dependent behavior
- The voltage drop between the base and emitter (forward voltage – Vbe) of the Diode-connected transistor decreases at a known rate as the temperature increases.
- By precisely amplifying this voltage change, it is easy to generate an analog signal that is directly proportional to temperature.

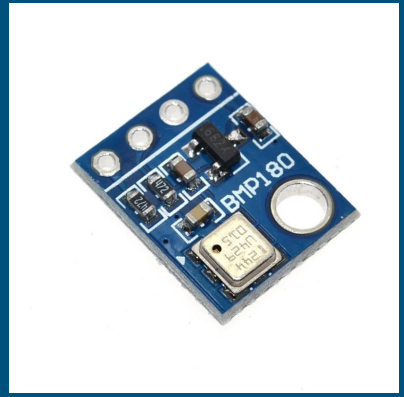
Datasheet



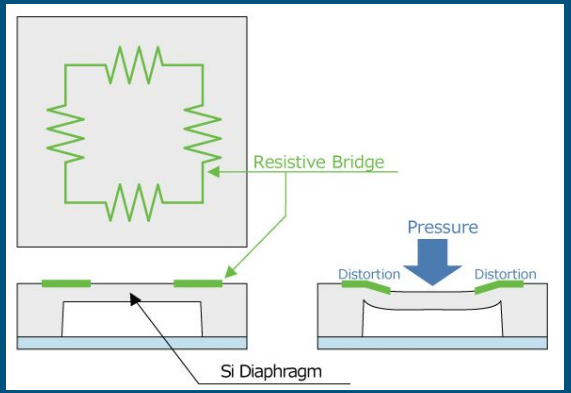
Barometric Pressure sensor

- BMP180
- Piezo-resistive pressure sensors utilize a single Si crystal plate as a diaphragm and diffuses impurities on its surface to form a resistive bridge circuit, making it possible to calculate pressure (atmospheric) by detecting the resistance change resulting from distortion of this resistive bridge when pressure is applied.
- I2C communication

Board package >



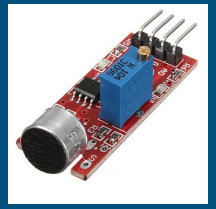
MEMS* diaphragm >



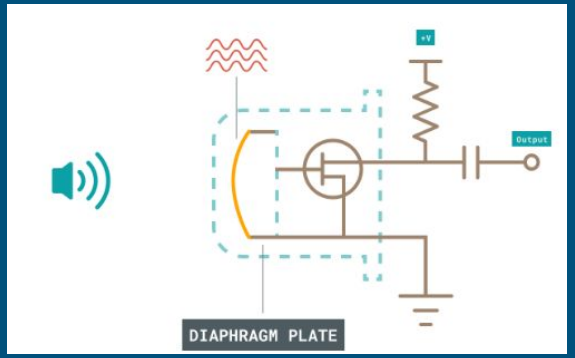
MEMS : Micro electro Mechanical System

Sound sensor

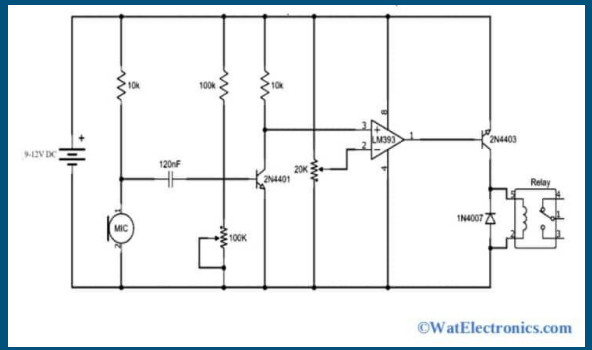
- Works as human ear
- Works in range of 3kHz - 6kHz and 52-48 dB
- contains a microphone, power amplifier, and high precision comparator t produce D0



Sensor Module



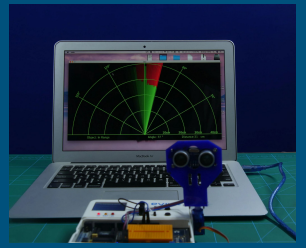
Working principle

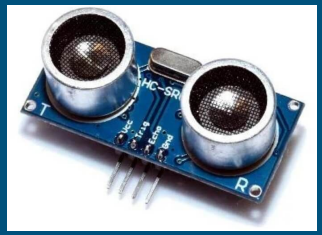


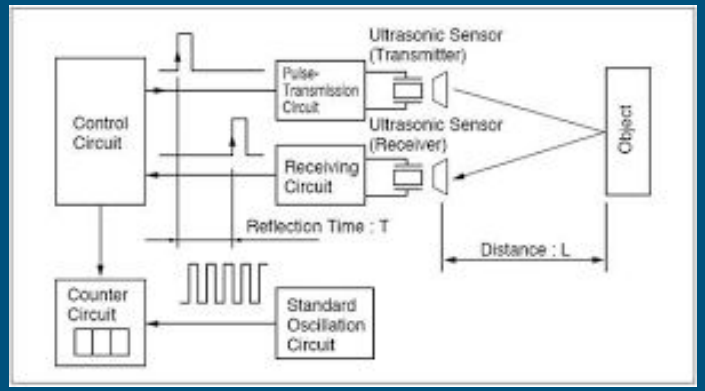
Circuit Diagram (will be explained in future slides)

Ultrasonic sensor

- The sensor’s transmitter generates and sends out ultrasonic sound waves at a high frequency, typically around 40 kHz, beyond the range of human hearing
- The sensor calculates the time taken for the sound waves to travel to the object and back, known as the “time of flight”
- Distance = Time of flight/(2* Speed of sound)

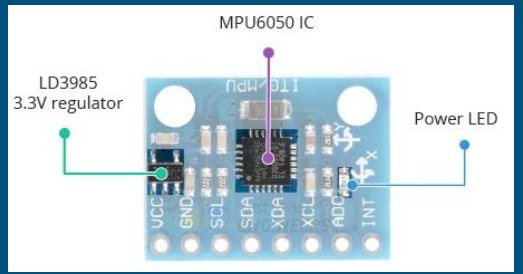




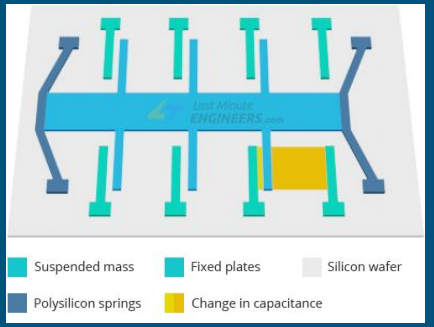


Six-Axis (Gyro + Accelerometer) MEMS MotionTracking™ Device

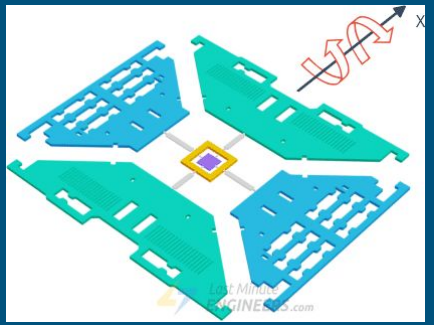
- One of the world's first motion tracking device designed low cost, low power and high performance motion tracking device.
- 3 Axis Gyro, and 3 axis accelerometer
- Gyroscopes/Accelerometer made as MEMS
- Detect change in the capacitance
- Temperature sensor
- I2C communication

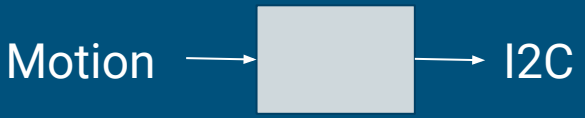


Accelerometer >



Gyroscope's
This sideways force happens whenever something moves in one direction while being rotated >

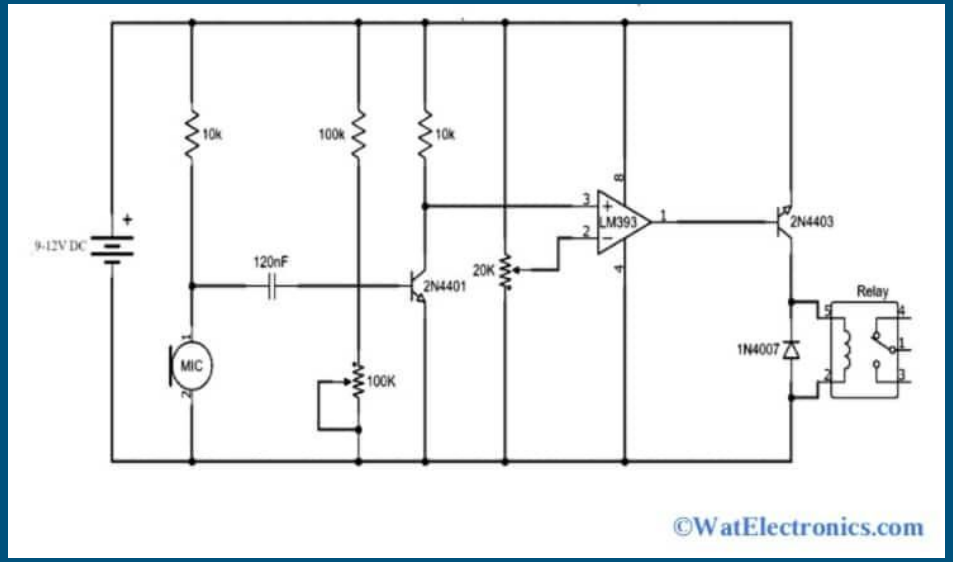




More info

Electronic Construction of Sound Sensor

- MIC : diaphragm produce varying emf proportional to the sound.
- Capacitor : Filter-out high frequency signals
- Potentiometer : Set relevant voltage to trigger transistor
- Comparator : Compare the voltage above which the output should produced
- Relay : Isolate circuits and drive right side in desired way



Activity 1 : `A Thing` That Collects Data and Displays

Reading analog values from LDR, sound sensor from arduino uno.

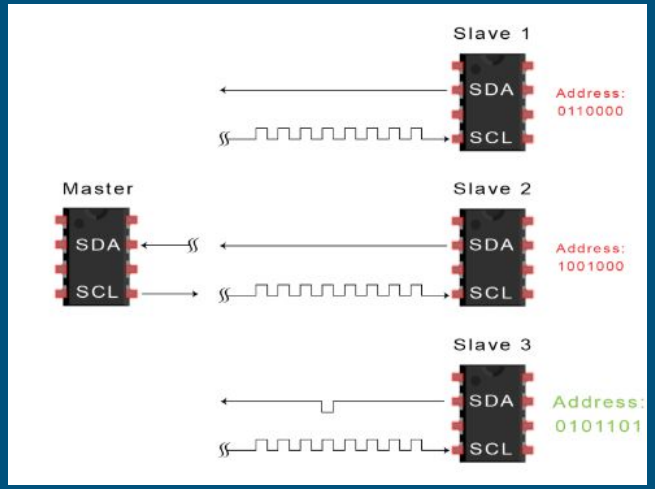
6. Communication Protocols

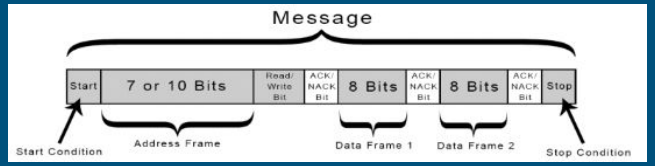
How these outputs to the environment changes send to the nearest data collection point

- Wired
  - I2C
  - SPI
  - RS485
  - RS232
  - CAN
  - UART
- Wireless
  - Wifi
  - Bluetooth
  - Zigbee
  - LoRa
  - NB-IoT
  - Cellular

I2C

- Multi master multi slave. Each slave get an address. Max slaves 1008.
- Synchronous to the clock signal shared by master and slave.
- Procedure
  - Start transmission, SDA low then SCL high to low
  - Master broadcast the address
  - Slave send a ACK
  - Master send/receive data frame
  - Receiving device send ACK
  - To stop the data transmission master switch SCL high before SDA high
- Pull-up SCL and SDA lines

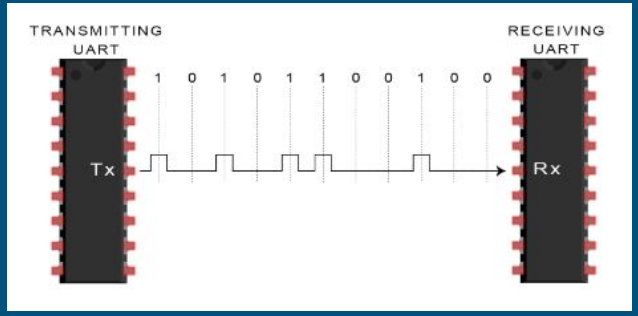


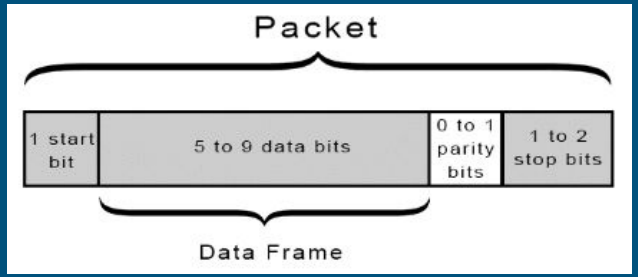


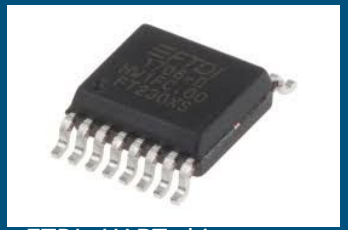
UART

- Transmit and receive data asynchronously.
- When the receiving UART detects a start bit, it starts to read the incoming bits at a specific frequency known as the baud rate.
- Only 2 devices.
- UART is a physical circuit in uc
- Error checking using parity bit.

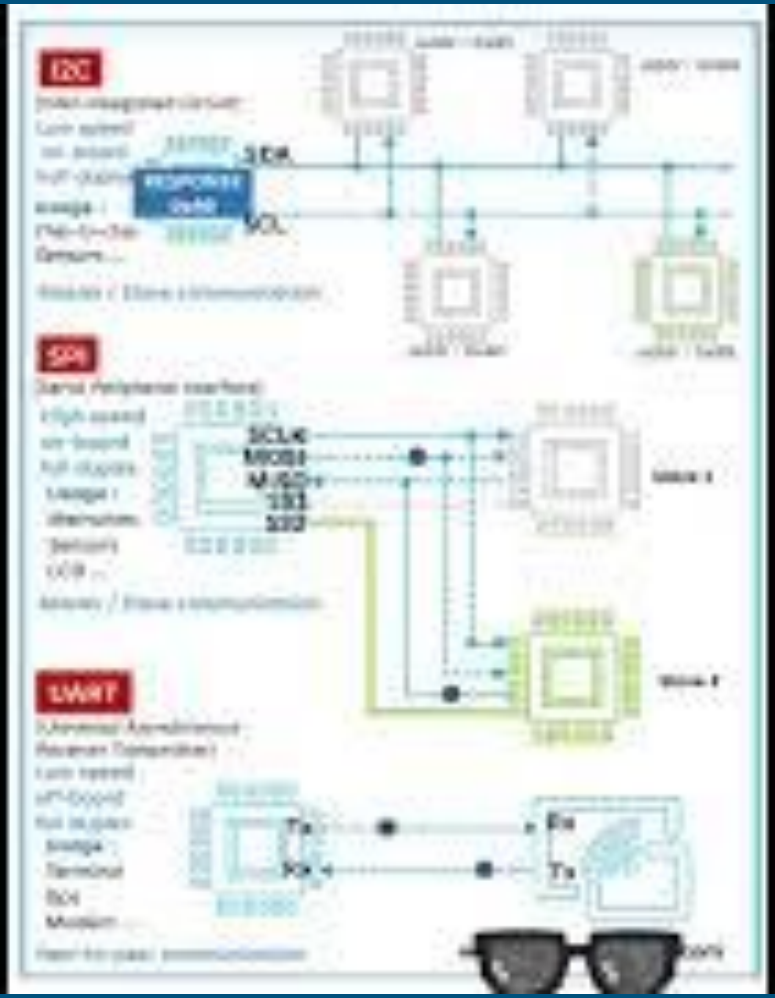
- Data in to the chip parallely and start bit, end bit and parity bit is added. Then data is output by serially.







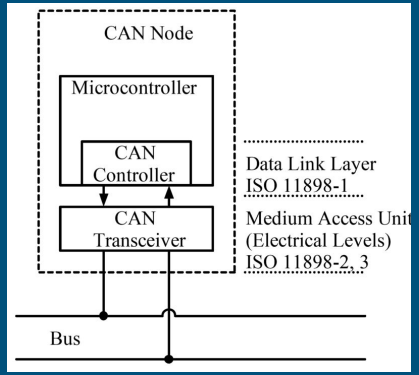
FTDI - UART chip


I2C
SDA
SCL
SPI
SCLK
MOSI
MISO
SS1
SS2
UART
Tx
Rx
Tx
Rx

CAN (Control Area Network)

- Vehicle bus standard, primary used to communicate with ECUs.
- Common versions, CAN2.0, CAN FD
- Multi master serial bus

- Designed to immune to noise in the bus using Differential Signaling
  - Difference in CAN-H and CAN-L represents data
  - Wired AND signal : CAN-H = AND goes to CAN-L
- Voltage levels
  - Logical 0 : CAN-H ~3.5V /CAN-L ~1.5V : Difference 2V
  - Logical 1 : CAN-H ~2.5V /CAN-L ~2.5V : Difference 0V
- Bitwise arbitration to choose the priority on the bus.
  - Lower ID , always win the arbitration and get higher priorities
- Termination resistances at each 2 edges 120Ohm to prevent signal reflection.



NBIoT (Narrowband IoT)

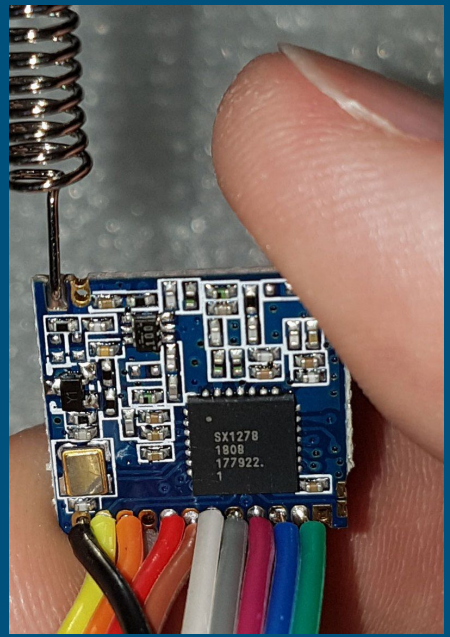
- Low power wide area network
- Focus especially on indoor coverage, low cost, long battery and high connection density.
- Uses subset of LTE bandwidth.
- It is service focus on IoT that will enable by your service provider, to your ara.

LoRa (Long Range)

- Low power and long range, ideal for battery operated devices to communicate.
- Uses license free sub-gigahertz band(433MHz, 868 MHz)
- 15-20 km range. 2-5 km range in urban areas.
- Data rate is low (0.3-50kbps) (tradeoff to range)

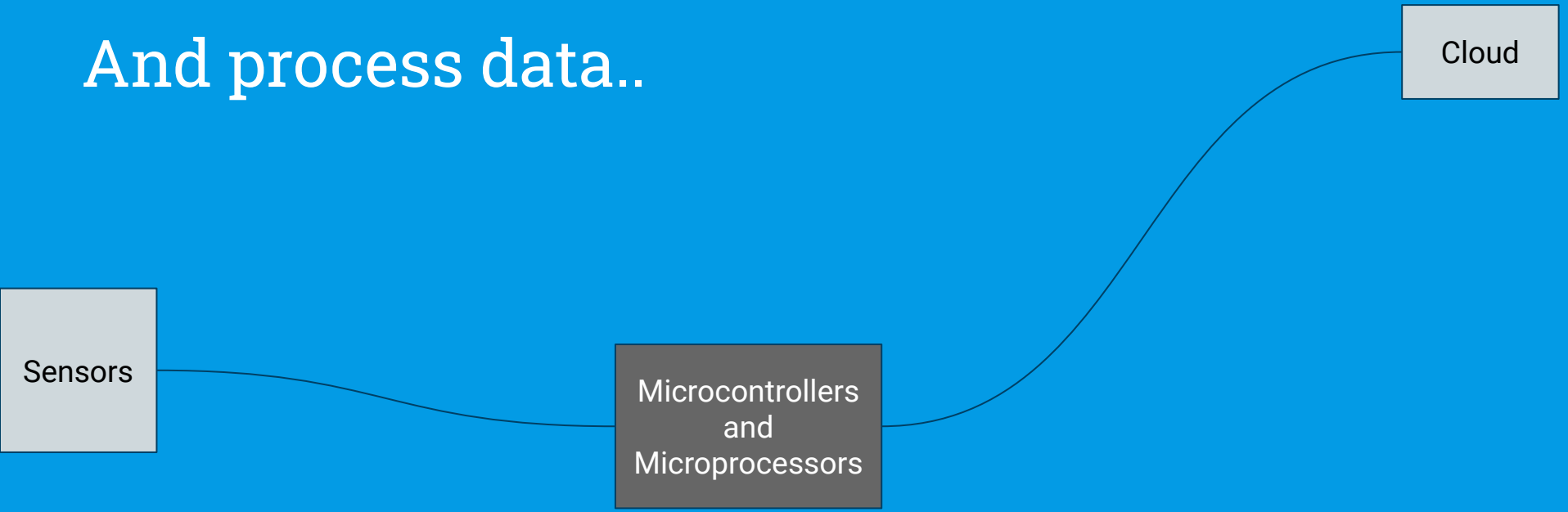
LoRaWAN

- Since LoRa defines the lower, physical, layer, the upper networking layers were lacking. LoRaWAN is a protocol that was developed to define the upper layers of the network.



LoRa Module

And process data..



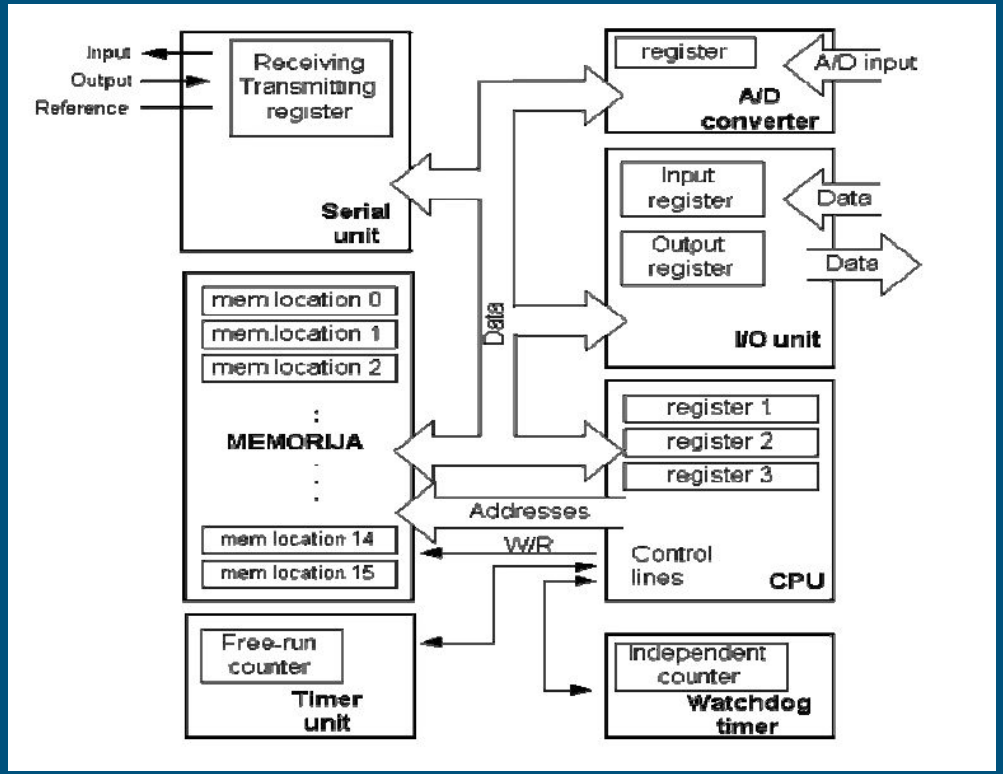
7. Microcontroller

- An integrated system with CPU, memory, and I/O peripherals.
- Includes CPU, RAM, ROM, and I/O on a single chip.
- Lower cost, compact design for embedded systems.
- IoT devices, automotive systems, and home appliances.
- Sufficient for specific, predefined tasks.

Navigating the Components of PIC Microcontroller Architecture

Components
RAM, ROM, CPU
ADC, DAC
Timers, Counters
CAN, SPI, UART

Architecture	
Harvard architecture(RISC)	• Data bus and address bus are seperate • Can process several instructions in one clock cycle • Great speed • ~20 instructions • Ex: PIC,STM32
Von Neumann Architecture (CISC).	• Use 2 clock cycles per instruction • ~80 instructions • Eg: Intel



Harvard architecture of PIC uC

Brand	Series	Arch	Clock	Flash(ROM)	RAM	Peripherals	Common Use Cases
AVR	ATmega328P	8-bit	20 MHz	32 KB	2KB	UART, SPI, I2C, ADC, PWM	Microchip
Microchip	PIC16F877A	8-bit	20 MHz	14 KB	368 B	**UART, SPI, I2C, ADC, Timers**	Home appliances, basic automation
STM32	STM32F103C8T6	32-bit	72 MHz	64 KB	20 KB	UART, SPI, I2C, ADC, PWM, Timers	Robotics, drones, IoT devices
NXP	LPC1768	32-bit	100 MHz	512 KB	64 KB	USB, Ethernet, CAN, SPI, ADC, PWM	Industrial IoT, motor control

8. Microprocessors

- A general-purpose CPU used for computing tasks.
- Requires external RAM, ROM, and peripherals for operation.
- Higher overall cost due to additional components.
- Used in computers, servers, and high-performance systems.
- Have a OS

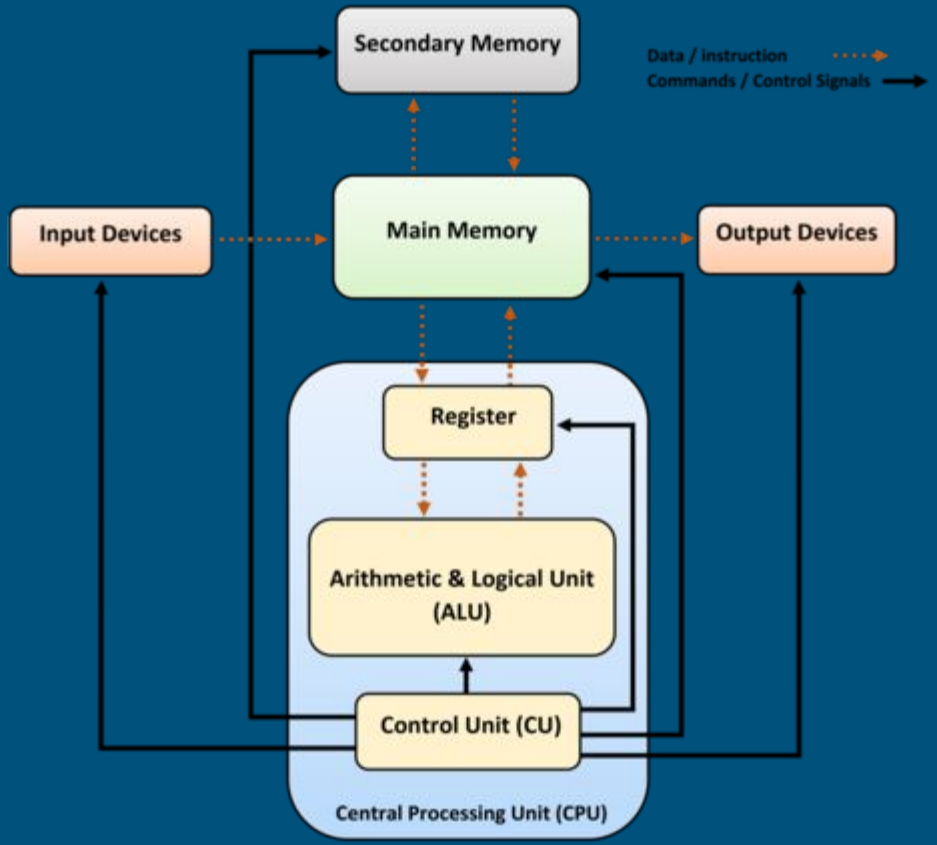
AMD NVIDIA

Microprocessor Architecture

Components
RAM(DDR4/DDR5), ROM, CPU
PCIe, USB, Ethernet
HDMI, MIPI
GPU, Neural Engine

Architecture	
Harvard architecture(RISC)	
Von Neumann Architecture (CISC).	
32-bit and 64 bit	• Data path width • 64 bit processors can process large chunk of data
ISA (Instruction Set Arch)	• Specify the behavior of machine code

Block diagram of a basic computer with uniprocessor CPU. Black lines indicate control flow, whereas red lines indicate data flow. Arrows indicate the direction of flow.



Brand	Series	Arch	Clock	Flash(ROM)	RAM	Peripherals	Common Use Cases
Intel	Core i7-12700K	x86-64 (64-bit)	Up to 5.0 GHz	N/A (uses external storage)	12 MB L3 cache	PCIe, USB, HDMI, Ethernet, DDR4/DDR5	High-performance PCs, gaming, video editing
AMD	Ryzen 9 5950X	x86-64 (64-bit)	Up to 4.9 GHz	N/A (uses external storage)	64 MB L3 cache	**PCIe, USB, HDMI, DDR4**	High-end desktops, content creation
ARM	Cortex-A76	ARMv8-A (64-bit)	Up to 3.0 GHz	N/A (uses external storage)	Configurable	USB, Ethernet, GPU, I2C, SPI	Mobile phones, tablets, IoT devices
NXP	M1	ARMv8.4-A (64-bit)	Up to 3.2 GHz	Unified memory	Unified memory	GPU, Neural Engine, PCIe, USB, Thunderbolt	MacBooks, iPads, energy-efficient systems

Unified mem: single pool of memory is shared between the CPU, GPU, and other processing units in a system. This eliminates the need for separate memory blocks for each processor and allows all units to access the same data without copying it across different memory locations.

9. Case Study: Smart Traffic Management System

Scenario

Your city has decided to implement a Smart Traffic Management System to reduce congestion and improve road safety. The system should perform the following tasks:

1. Real-time Vehicle Detection: Use cameras and sensors to detect the number and type of vehicles at intersections.
2. Traffic Signal Control: Dynamically control traffic lights based on real-time data.
3. Data Transmission: Send live traffic data to a central server for analytics and monitoring.
4. AI-based Decision Making: Use machine learning algorithms to predict traffic patterns and adjust signals.

Additional Requirements

- The system should operate 24/7 with minimal power consumption.
- Each intersection's hardware should be compact and cost-effective.
- The central server will have a high-speed internet connection, but local devices at intersections may rely on wireless communication (e.g., Wi-Fi or Zigbee).
- The device must support real-time operations and interface with sensors (e.g., IR sensors, cameras) and communication modules.

Question

What should we use for the local hardware at each intersection

a microcontroller or a microprocessor? Justify your selection based on the following factors:

Computational Requirements: Real-time processing of sensor and camera data.

Power Efficiency: Ability to operate continuously with minimal power.

Cost and Size: Feasibility for deployment across multiple intersections.

Peripheral Support: Compatibility with sensors, communication modules, and cameras.

Scalability: Flexibility to add more features (e.g., AI processing) in the future.

Computational Requirements

Microcontroller:

Microcontrollers are typically less powerful but sufficient for simple sensor data processing and basic control tasks. However, they struggle with high-computation tasks like AI-based decision-making or real-time image processing from cameras.

Example: An STM32 microcontroller can handle basic traffic signal control but may not process video streams efficiently

Microprocessor:

Microprocessors are designed for complex computations and can handle AI-based predictions and real-time video analysis.

Example: A Raspberry Pi 4 or a Jetson Nano can process video streams and run machine learning models efficiently.

Winner: ? Hm....

Power Efficiency

Microcontroller:

Microcontrollers consume significantly less power as they are designed for energy-efficient embedded systems. Suitable for 24/7 operation in low-power environments.

Example: An STM32 or an AVR controller can run for long periods on minimal power.

Microprocessor:

Microprocessors consume more power due to higher performance capabilities. Prolonged operation might require additional cooling and energy resources.

Example: Raspberry Pi or Jetson Nano may require active cooling and a stable power supply.

Winner: ? Hm…..

Cost and Size

Microcontroller:

Microcontrollers are compact and cost-effective, making them ideal for deployment at multiple intersections.

Example: An AVR or STM32 microcontroller can cost less than $5 per unit.

Microprocessor:

Microprocessors are larger and more expensive due to the need for additional components (e.g., RAM, storage).

Example: A Raspberry Pi 4 costs around $35–$50, excluding accessories.

Winner: ? Hmm ...

Peripheral Support

Microcontroller:

Microcontrollers are excellent for interfacing with sensors and basic communication modules (e.g., Zigbee, UART, I2C). However, their ability to support high-resolution cameras is limited.

Example: STM32 can handle IR sensors and basic wireless modules but not advanced peripherals like HD cameras.

Microprocessor:

Microprocessors support a wide range of peripherals, including cameras, Ethernet, Wi-Fi, and USB. They are ideal for complex tasks requiring diverse interfaces.

Example: Raspberry Pi 4 supports HD cameras and Wi-Fi seamlessly.

Winner: ? Hmmmm…..

Scalability

Microcontroller:

Adding features like AI or advanced analytics is challenging due to limited computational power and memory. Microcontrollers are less scalable for future upgrades..

Microprocessor:

Microprocessors are highly scalable and can easily accommodate new features, such as advanced AI models or additional sensors.

Winner: ? Hmmmmmmm....

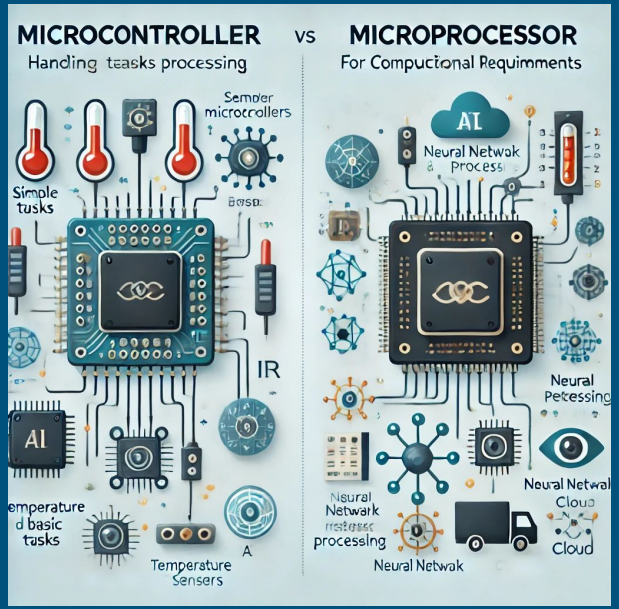
Conclusion

Microcontroller:

Suitable for low-cost, energy-efficient tasks like basic signal control and sensor data processing.

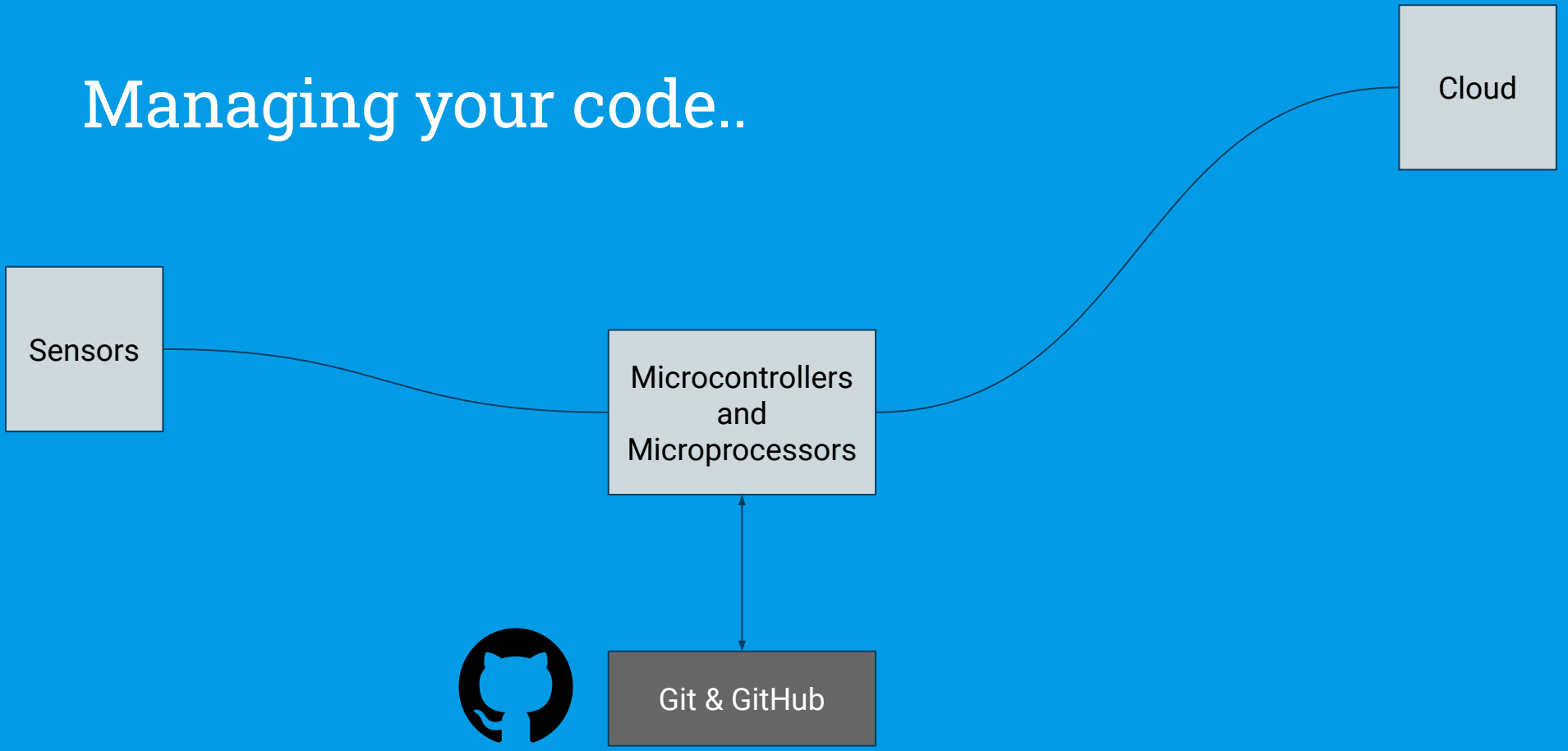
Microprocessor:

Necessary for intersections requiring advanced capabilities, such as AI-based traffic predictions and real-time video analysis.



Use a microprocessor (e.g., Raspberry Pi or Jetson Nano) for intersections with AI and video processing needs. Use a microcontroller (e.g., STM32) for simple intersections with basic control requirements to save costs.

Managing your code..



10 . Codebase management

- Collaboration without Chaos

  When multiple people work on the same project, it's essential to prevent overwriting each other's work. Code management allows seamless collaboration by keeping track of everyone's contributions.

- Track Changes Over Time

  Keep a complete history of your project's progress. Easily find out who made changes, what was changed, and when it happened.Roll back to previous versions if something breaks or goes wrong.

- Manage Multiple Versions

  Branching allows parallel development of features without affecting the main project.Merge and resolve conflicts efficiently when features are ready to be integrated.

- Avoid “It Works on My Machine” Issues

  Code management tools make sure everyone is using the same version of code.Makes debugging and testing easier by syncing the project across machines and environments.

- Prepare for the Future

  Document and save your work for others to build upon.Maintain a robust project history for future updates, bug fixes, and new developers joining the team.

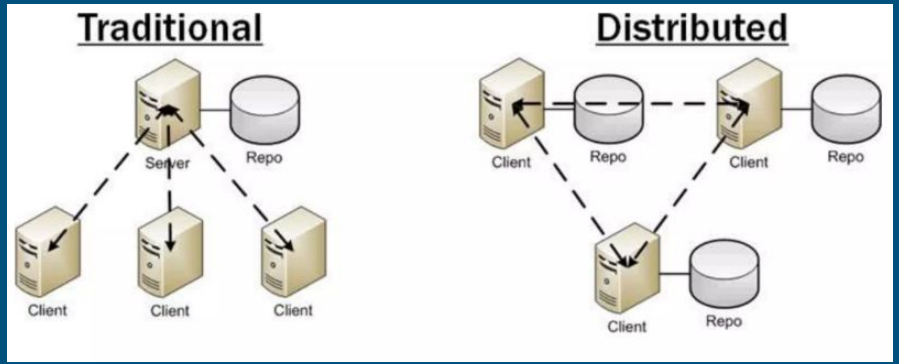
11. Version Control System

What is it?

- Managing files and directories
- Track changes over time
- Recall previous version

What is a ‘Distributed version control system’

- Managing files and directories
- Track changes over time
- Recall previous version



- No central server
- Every repo si a client, server and repo

12. Git & Github : Git

- Distributed Version Control System
- Free, open source
- Cross Platform
- Use checksum to ensure data integrity
- Came Out of linux development community

Features

- Can work without internet
- No single failure point
- Track changes not versions

Background:

Git was developed by Linus Torvalds in April 2005 after the free license for BitKeeper, a proprietary SCM tool used for Linux kernel development, was revoked.

Motivation:

Torvalds needed a distributed version control system with speed, scalability, and strong safeguards, as existing systems didn’t meet Linux kernel development needs.

Git & Github : GitHub

- GitHub is a web-based platform for version control and collaboration, built on top of Git

Features

- Code Hosting
- Collaboration tool
- Community & Open Source
- Continuous development and integration tool
- Software automation tool

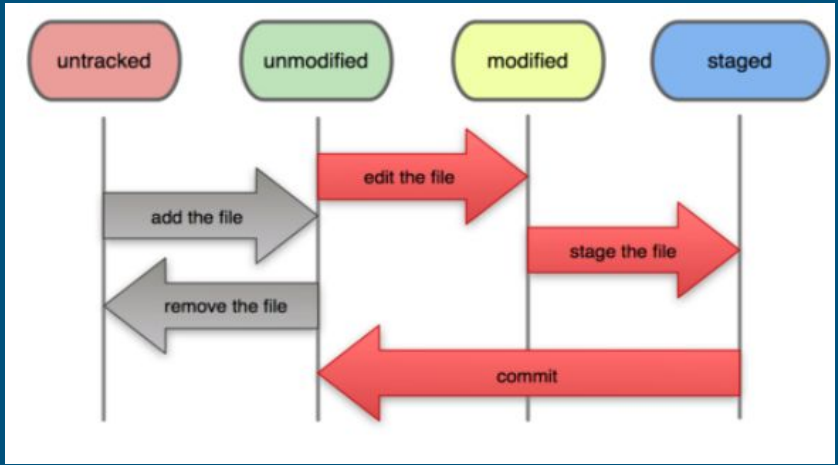
Making your own Github account

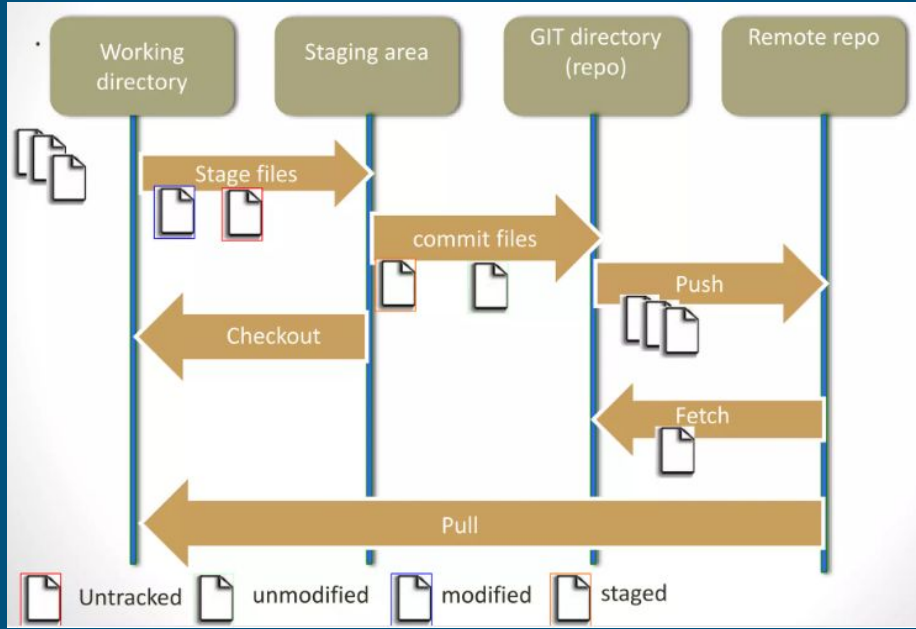
- GitHub is a web-based platform for version control and collaboration, built on top of Git

Features

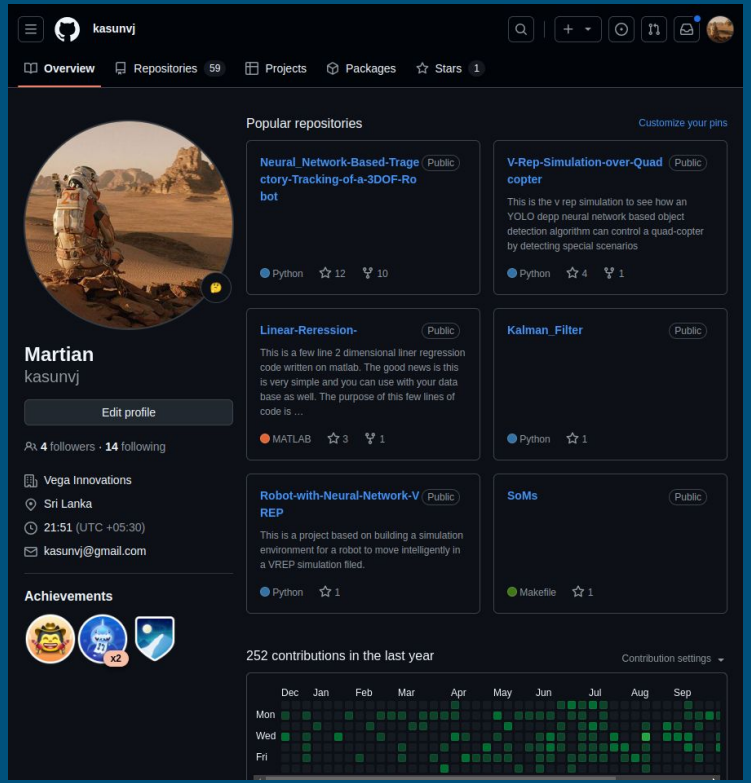
- Code Hosting
- Collaboration tool
- Community & Open Source
- Continuous development and integration tool
- Software automation tool

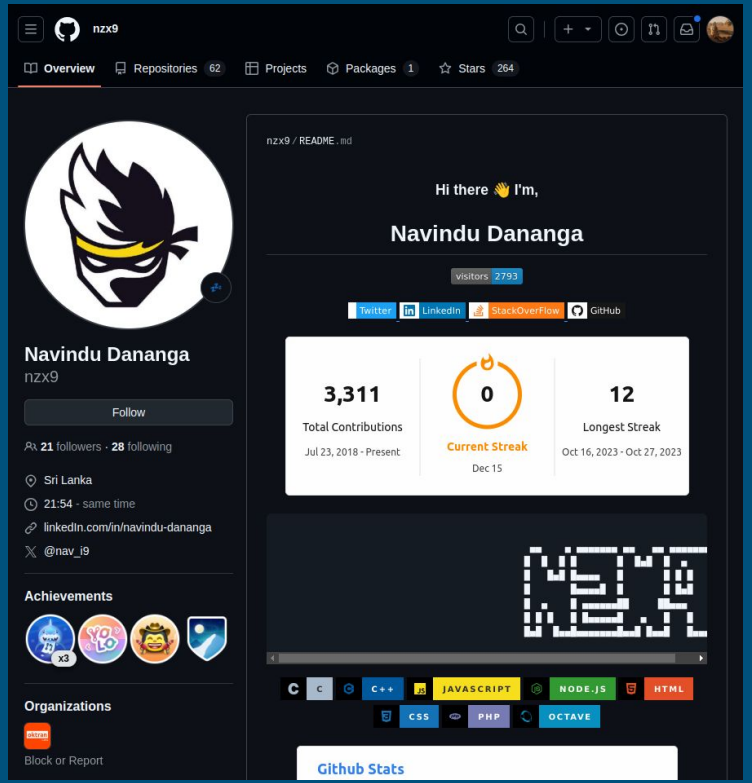
Basic Git workflow





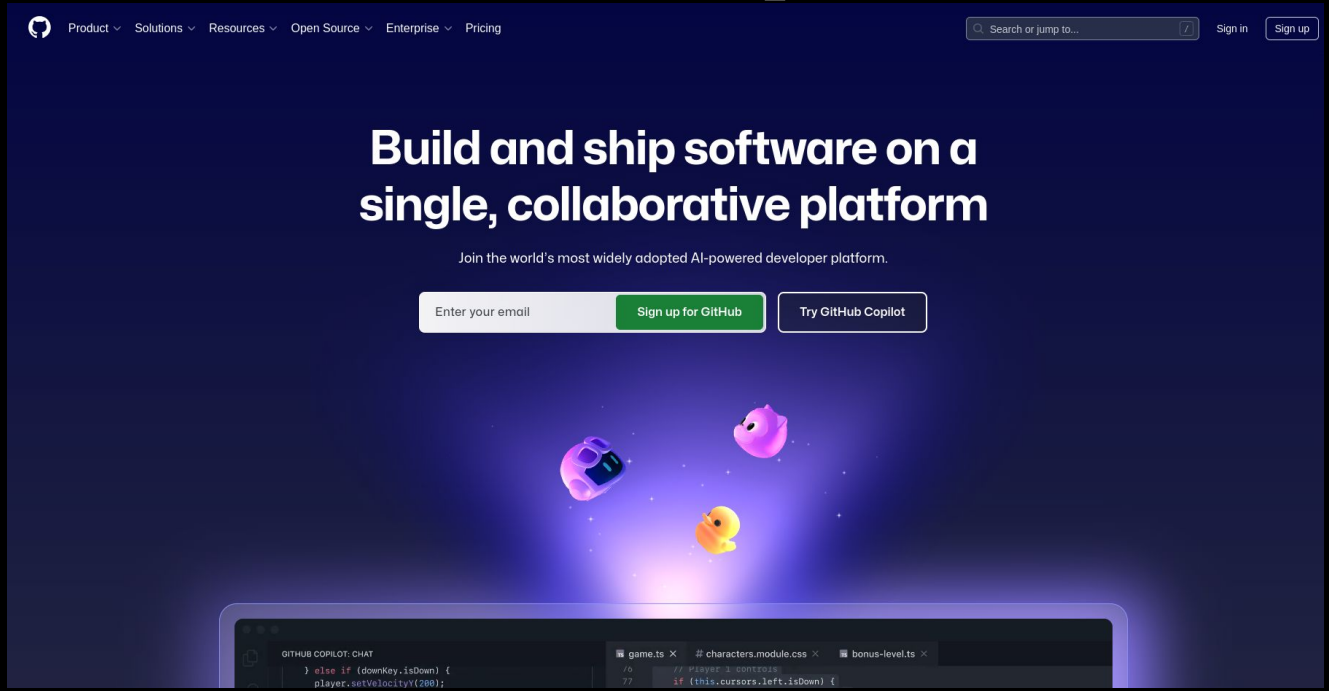
My Git profile





Activity 2 : A sustainable development

1. If you don't have, it is time to create one for your own



Activity 2:

2. Installing git.

Windows :

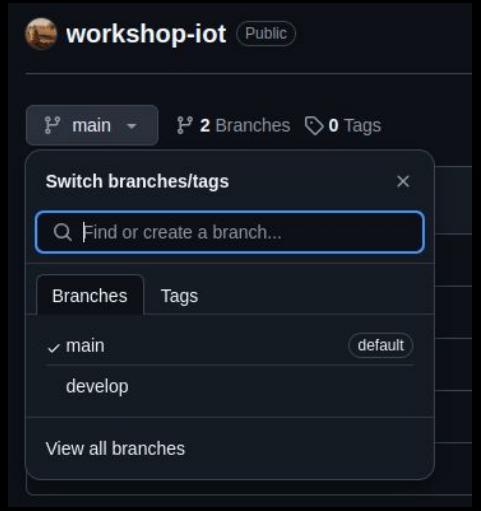
Linux :

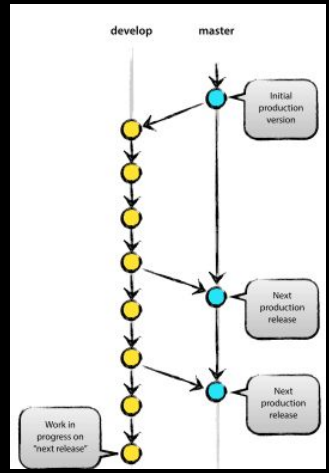
```
sudo apt install git-all
```

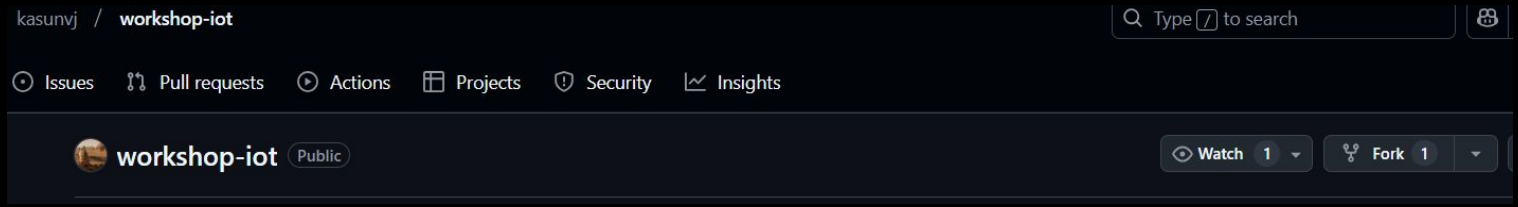
macOS :

Activity 2:

3. Login to your github account
4. Visit the awesome repository
   - Observe there are 2 branches main and develop
5. Fork the origin to your repo

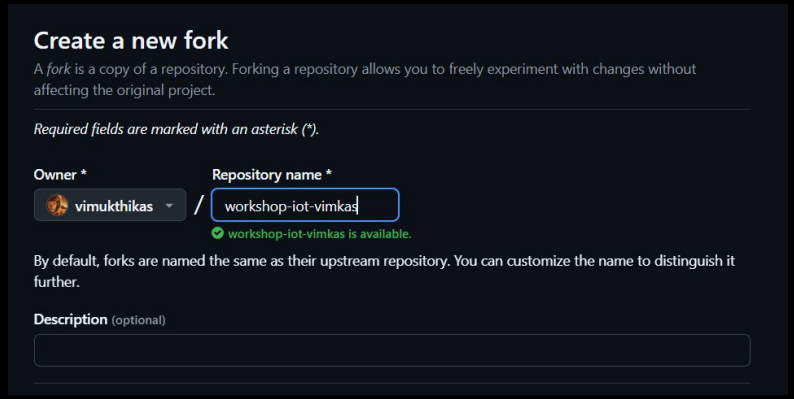


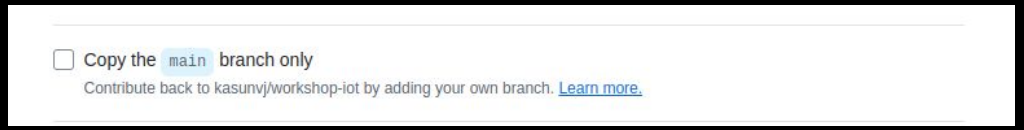




Activity 2:

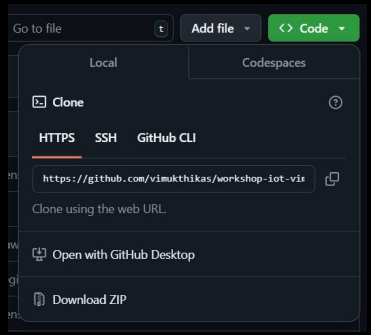
6. While forking the repo you can add a custom name to it.





7. Clone it to your device. Have a local copy. Check both main and develop branches are exist

```
$git clone
https://github.com/vimukthikas
/workshop-iot-vimkas.git
```



Activity 2:

6. Ensure the develop branch is up to date: Before creating a new branch from develop, ensure you have the latest change

```
$git checkout develop
$git pull origin develop
```

note: `$git status` *provides the details of the existing branch/staging level and etc*

```
On branch develop
Your branch is up to date with 'origin/develop'.
```

7. Create a branch for your development as a feature branch. Feature branches are only exist in developer repo.

```
$git checkout -b myfeature
develop
```

```
On branch myfeature
nothing to commit, working tree clean
```

Activity 2:

note: If you've forked a repository and cloned it, the origin points to your fork.

To verify where is your origin points `$git remote -v`

note: When you are pushing, they will ask for a username and access token. Before doing that you need to create a access token for your account.

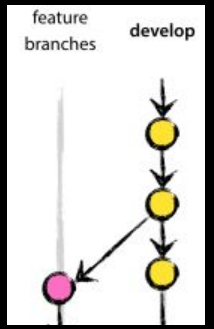
User name : 

Password : settings > Developer settings > Personal access token > Tokens (classic) > generate new tokens > copy token

8. Push your feature branch

```
$git push -u origin myfeature
```

Activity 2:



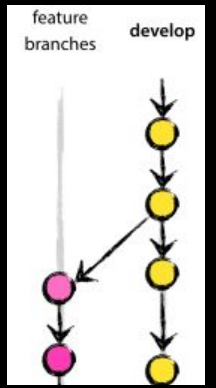
7. Go to activity3_git folder. Create a folder and a text file with your name inside the folder ‘student register’

Ex:
workshop-iot/activity3_git/student_register
/name/name.txt

7. $git status results something like this

```
On branch myfeature
Your branch is up to date with 'origin/myfeature'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   activity3_git/students_register/vimkas/vimkas.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Activity 2:



8. Stage your changes

```
$git add --all
$git commit -m “Comment”
```

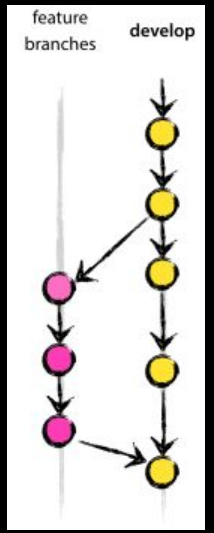
```
s$ git commit -m "adding my name"
[myfeature 1bb2eae] adding my name
 1 file changed, 1 insertion(+)
```

9.Push to your feature branch and delete the branch

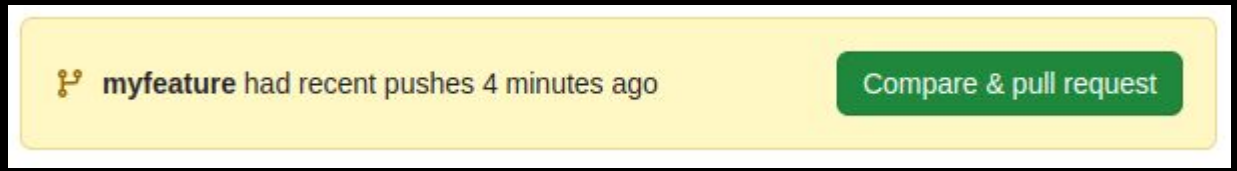
```
$git push origin myfeature

git branch -d myfeature
```

Enter your username and personal access token once requested

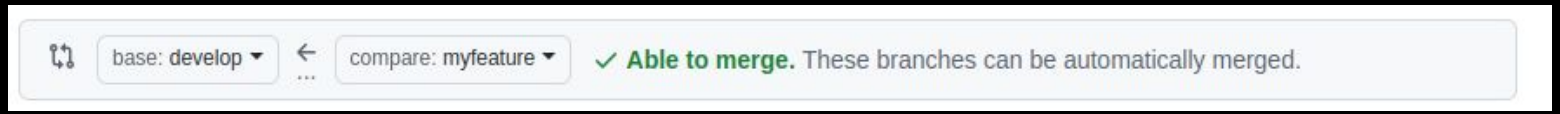
Activity 2:



10 . Now when you login to your repo you will see,

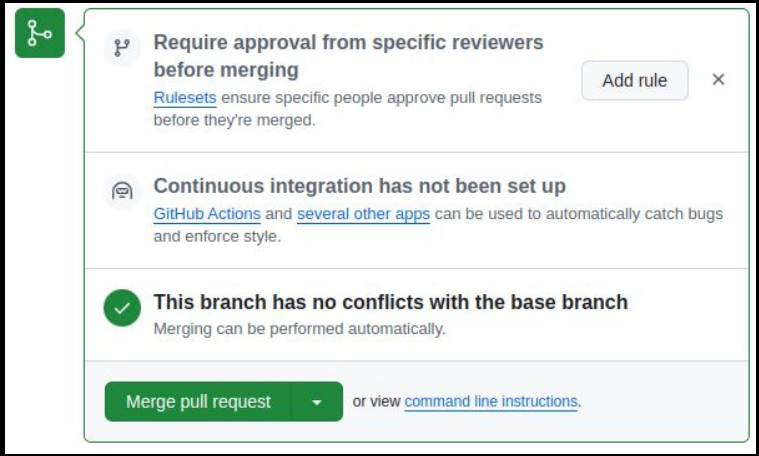


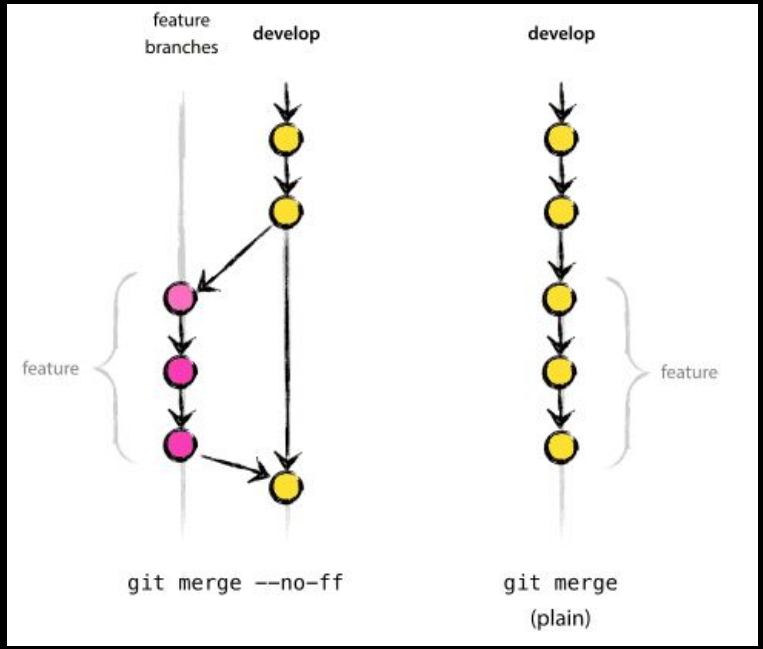
11. Compare and pull request. Select to do the pull request to developer branch from your myfeature branch



Activity 2:

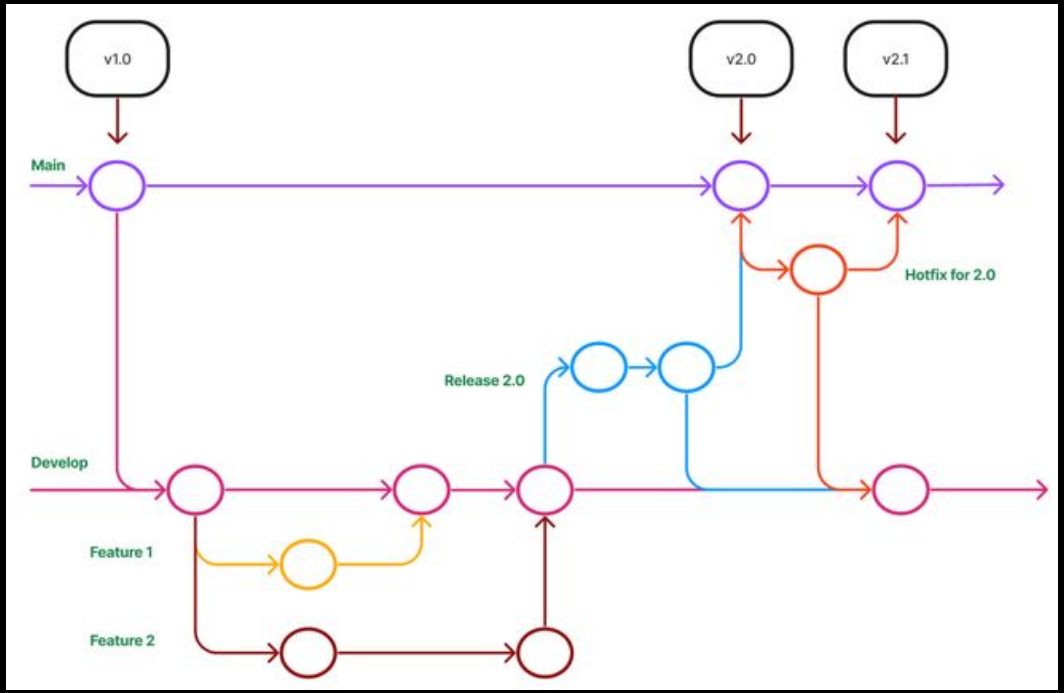
12. What happens when merging





Activity 2:

13. Release



Other GitHub features

That is where github becomes more handy

- Actions
- CI/CD pipelines
- Github Codespace

13. Programming Microcontrollers/Microprocessors

One of the favorite languages for programming is C.

C is a general-purpose programming language created by Dennis Ritchie at the Bell Laboratories in 1972

It is a very popular language, despite being old. The main reason for its popularity is because it is a fundamental language in the field of computer science.

Practice basic programming with C

- Hello world
- Variables
- Taking inputs
- If else
- Loops (for loop and while loop)
- Functions
- Switch case

- Structure of a C Program

```c
#include <stdio.h> // Header files

int main() {       // Entry point
    printf("Hello, World!\n"); // Output statement
    return 0;      // Exit status
}
```

Key Components:

- `#include` for including libraries.
- `main()` as the entry point.
- `printf()` for printing output.
- `return` to exit the program.

Data Types in C

Basic Data Types:

- `int` (integer), `float` (floating-point), `double` (double precision), `char` (character).

Derived Data Types:

- Arrays, Pointers, Structures, and Unions.

Example:

```
int age = 25;
float height = 5.8;
char grade = 'A';
```

- Operators in C

Arithmetic: +, -, *, /, %
Relational: ==, !=, >, <, >=, <=
Logical: &&, ||, !
Assignment: =, +=, -=, *=, /=

```
int a = 10, b = 20;
int sum = a + b;  // sum = 30
```

- Control Structures
  - If -else

```
if (x > 0) {
    printf("Positive");
} else {
    printf("Non-positive");
}
```

  - Loops

```
for (int i = 0; i < 5; i++) {
    printf("%d", i);
}
```

- Functions in C

Modularize code for reuse and clarity.

```
return_type function_name(parameters) {
    // code
    return value;
}
```

```
int add(int a, int b) {
    return a + b;
}
```

- Arrays and Strings

Arrays: Collection of similar data types

```
int numbers[5] = {1, 2, 3, 4, 5};
```

Strings: Array of characters ending with `\0`

```
char name[] = "John";
```

- Basic Input/Output in C

```c
int age;
printf("Enter age: ");
scanf("%d", &age);
```

- Switch Case in C

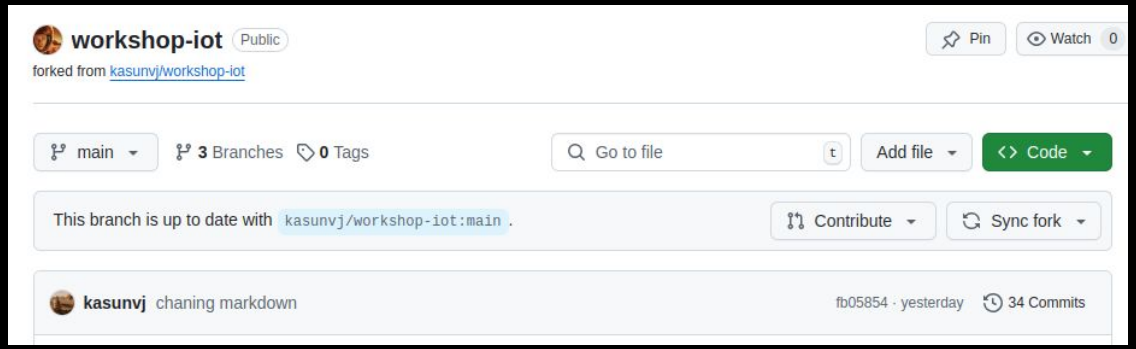
```
switch (expression) {
    case constant1:
        // Code to execute if expression == constant1
        break;
    case constant2:
        // Code to execute if expression == constant2
        break;
    ...
    default:
        // Code to execute if none of the cases match
}
```

```
#include <stdio.h>

int main() {
    int choice;

    printf("Enter a number (1-3): ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("You selected Option 1\n");
            break;
        case 2:
            printf("You selected Option 2\n");
            break;
        case 3:
            printf("You selected Option 3\n");
            break;
        default:
            printf("Invalid choice\n");
    }
    return 0;
}
```

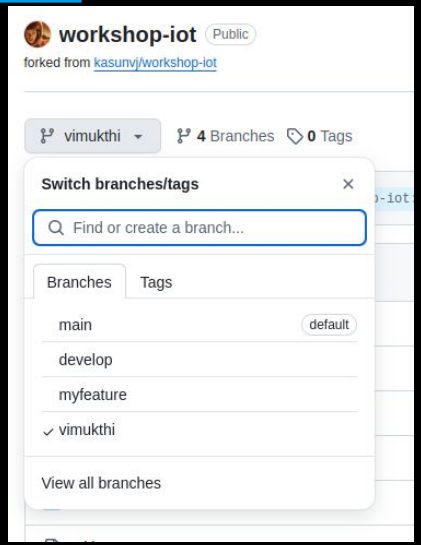
Activity 3: Programming Exercise

Use github codespace for coding testing and submit.

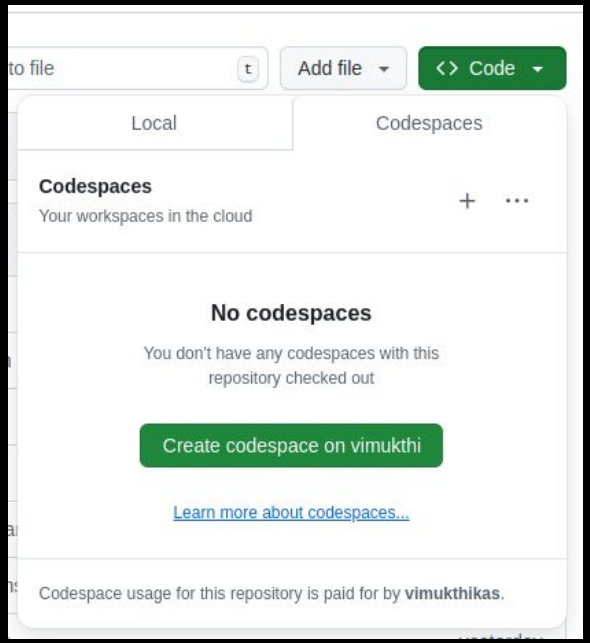
1. Fork the repo



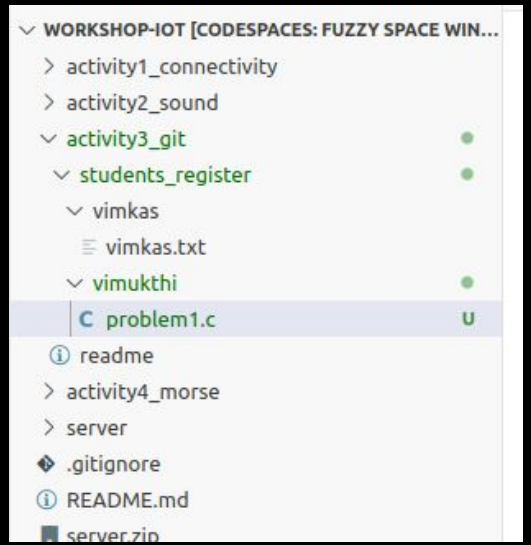
2. Create a branch



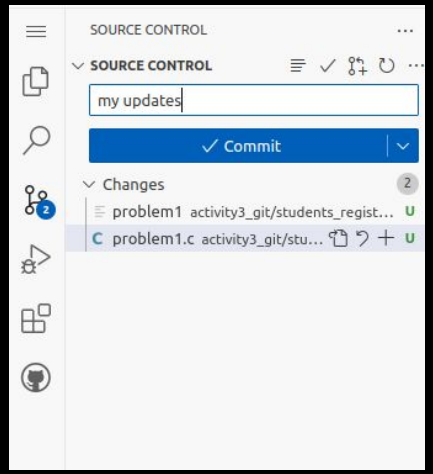
3. Create a code space



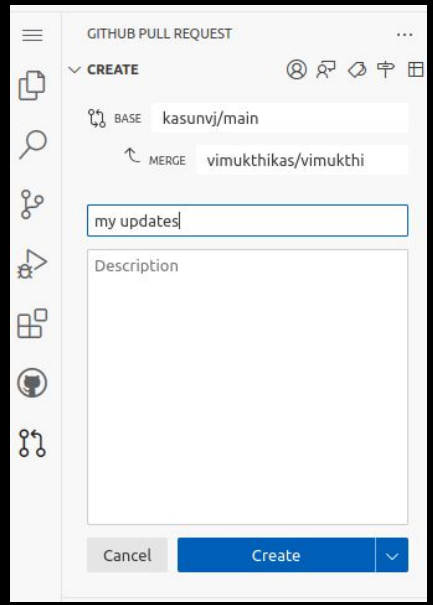
4. Inside activity3_git, create a folder with your name. Code name should go as ‘problemX.c’



5. After you complete all tasks, stage your change and sync it with another comment



6. Do a pull request to origin



Task 1 :

- Write a program to print a triangle of stars. The number of rows n is fixed to 4.

Example Input: n = 4

Example Output :

*

**

***

****

```c
#include <stdio.h>

void printTriangle(); // Declare the function

int main() {
    printTriangle();
    return 0;
}
```

```
void printTriangle() {
    for (int i = 1; i <= 4; i++) {
        for (int j = 1; j <= i; j++) {
            printf("*");
        }
        printf("\n");
    }
}
```

Task 2 :

- Write a program to print a pyramid of stars of height n.

Example Input: n = 4

Example Output :

```
   *
  ***
 *****
*******
```

```c
#include <stdio.h>
void printPyramid(int n);

int main() {
    int n = 4;
    printPyramid(n);
    return 0;
}
```

```c
void printPyramid(int n) {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n - i; j++) printf(" ");
        for (int j = 1; j <= 2 * i - 1; j++) printf("*");
        printf("\n");
    }
}
```

Task 3 :

- Write a function to reverse a given string.

Example Input: str = "hello"

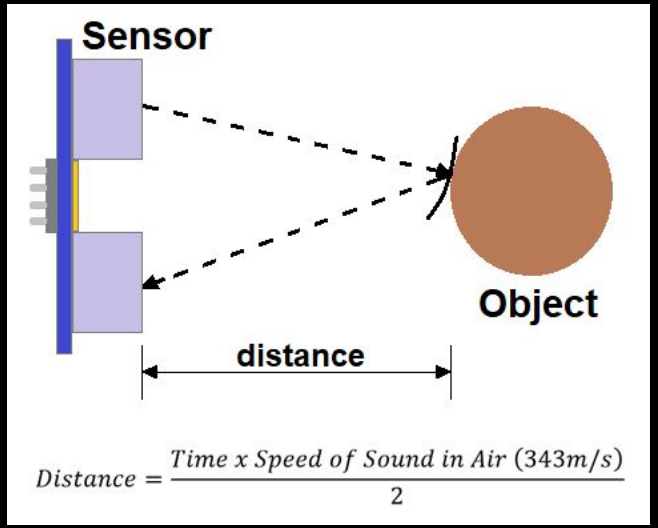
Example Output : `olleh`

```c
#include <stdio.h>
void reverseString(char str[]);

int main() {
    char str[] = "hello";
    reverseString(str);
    printf("Reversed: %s\n", str);
    return 0;
}
```

```c
#include<stdio.h>
#include<string.h>

int main(){
    char str[] = "hellow";
    for (int i=1; i<=strlen(str); i++){
        printf("%c",str[strlen(str) - i ]);
    }
    return 0;
}
```

Task 4 :

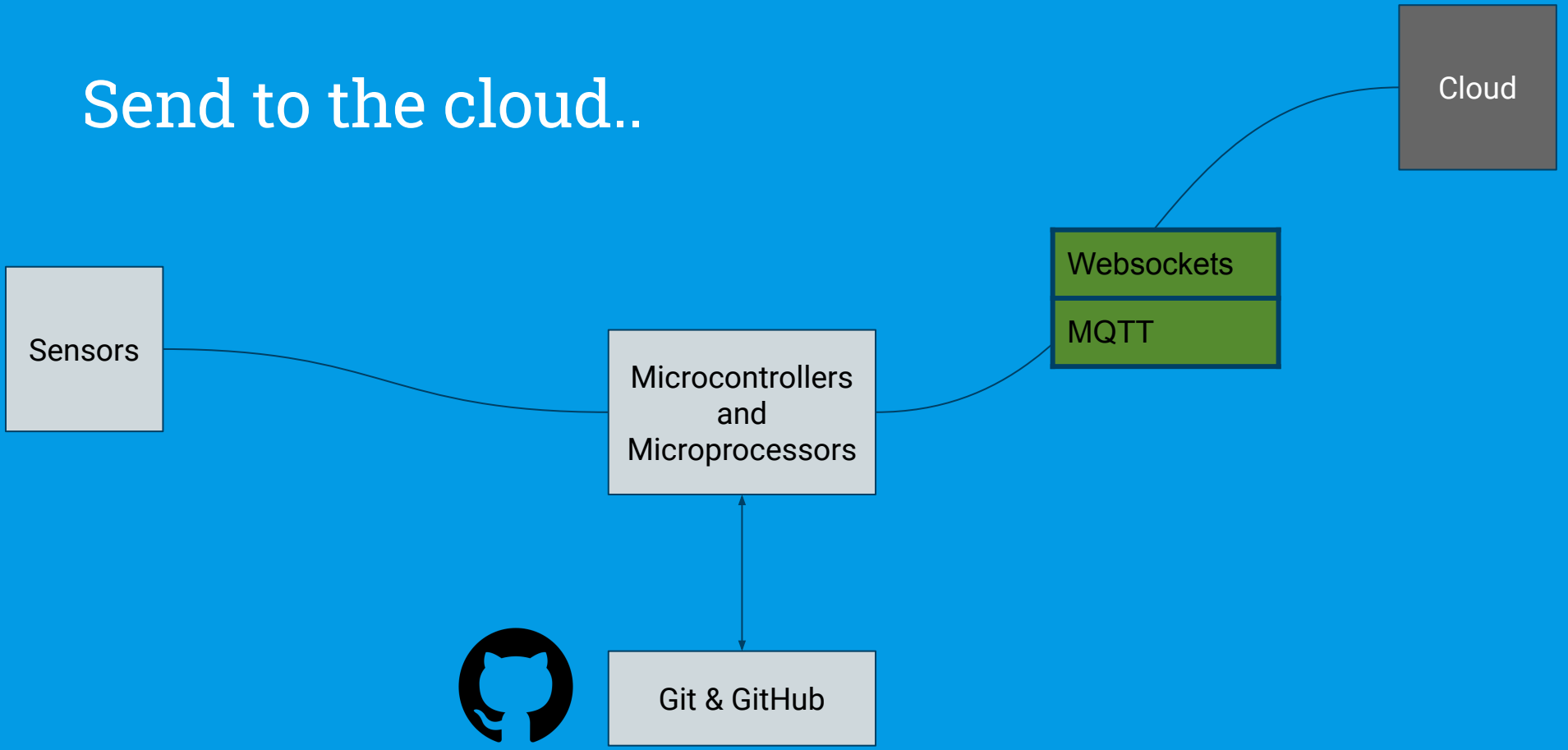
- Write a program to calculate the distance to an object using the time of flight of sound. The speed of sound is `343 m/s`. The time between the transmission and reception (time_us, in microseconds) is provided.




```c
#include <stdio.h>

float calculateDistance(float time_us); // Declare the function

int main() {
    float time_us = 1000.0; // Example: time in microseconds
    float distance = calculateDistance(time_us);
    printf("Calculated Distance: %.2f meters\n", distance);
    return 0;
}
```

```
float calculateDistance(float time_us) {
    float time_s = time_us / 1e6; // Convert time to seconds
    float distance = (time_s * 343) / 2; // Calculate distance in meters
    return distance;
}
```

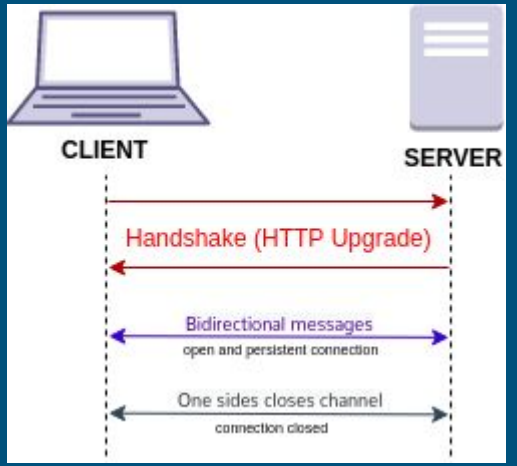
Send to the cloud..



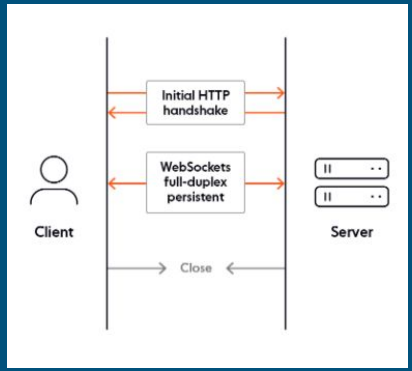
14. Websockets

TCP - packet transmission protocol over IP(TCP/IP). provides reliable, rendered and error check delivery of packets. UDP(User datagram protocol)

- Communication protocol improving HTTP
- TCP (Transmission Control Protocol) - 443 port
- Persistent connection once connected (not like HTTP)
- Full duplex (lower overhead than HTTP polling) - This is made possible by providing a standardized way for the server to send content to the client without being first request
- Power live chat, Broadcast realtime event data, such as live scores and traffic updates, Deliver notifications and alerts, Keep your backend and frontend in realtime sync,Add live location tracking capabilities to urban mobility and food delivery apps



WebSockets - API



- Define normally as ‘ws’ and ‘wss’
- Provide interface creating and managing websockets

How do WebSockets works

- Establishing a connection
  - Client initiate - opening handshake is a GET request to upgrade HTTP connection to websocket
  - Server responds : HTTP 101 Switching Protocols
  - API level that can be done by

```
const socket = new WebSocket('wss://example.org');
```

```
socket.onopen = function(e) { console.log('Connection open!');};
```

Other commands

socket.onmessage , socket.onclose, socket.onerror

- Data transmission
  - Fragmenting : Fragment message into multiple frames
  - Masking : data sent by the client to server is masked(make unclear)
  - Payload : text and binary. 127 bytes

```
socket.send(JSON.stringify({'msg': 'payload'}));
```

- Closing
  - Can close with a human readable code and message

```
socket.close(1003, "Unsupported data type!");
```

Pros :

- enable realtime communication between the client and server without the need for frequent HTTP requests/responses
- WebSockets are also more efficient, as they allow data to be transmitted without the need for repetitive HTTP headers and handshakes
- Low bandwidth usage

Cons

- not optimized for streaming audio and video data
- don’t automatically recover when connections are terminated.
- WebSockets are stateful, which makes them hard to use in large-scale system

MQTT(Message Queuing Telemetry Transport)

- Publish-subscribe messaging protocol dating back in 1999
- M2M communication with IBM machines with limited bandwidth, cpu power, battery life.
- Build on top of TCP/IP
- publisher - broker - one(or more) client(s)
- The publisher does not require any configuration concerning the number or location of subscribers receiving messages. Likewise, subscribers do not need publisher-specific setup
- Delivery control based on hierarchy

```
var options = {
  keepalive: 60,
  username: 'xVLyHw.YA9fyg',
  password: 'yU4BEnHoxMf7Svne',
  port: 8883
};

var client = mqtt.connect('mqtts:mqtt.ably.io', options);
```

- MQTT uses plain text to transmit security credentials. Then SSl frameworks comes.
- One of the distinguishing characteristics of MQTT is its leverage of channels

```
channel = "user/path/channel"
```

- Message delivery models controlled by a QoS(Quality of Service)

```
client.subscribe('test/topic', { qos: 0 }); // Subscribing with QoS 0
```

MQTT - QoS

```
client.subscribe('test/topic', { qos: 0 });
// Subscribing with QoS 0
```

- At most once :

Description: Messages are delivered **at most once**, with no guarantee of delivery. This is a "fire-and-forget" approach. The broker sends the message to the subscriber without waiting for acknowledgment.

Use Case: Useful for non-critical messages like sensor data (e.g., temperature readings) where occasional loss is acceptable.

Key Points:

- Fastest delivery.
- No retransmission of lost messages.

MQTT - QoS

```
client.subscribe('test/topic', { qos: 1 });
// Subscribing with QoS 1
```

- At least once :

Description: Messages are delivered at least once, ensuring that the subscriber receives the message. However, duplicates may occur.

Use Case: Suitable for scenarios where delivery is critical, but duplicates can be handled (e.g., financial transactions with idempotent operations)

Key Points:

- Ensures message delivery with possible duplicates.
- Acknowledgment (PUBACK) is required.

MQTT - QoS

```
client.subscribe('test/topic', { qos: 2 });
// Subscribing with QoS 2
```

- exactly once :

Description: Messages are delivered exactly once, ensuring no duplicates and no message loss. This involves a two-step handshake process (PUBREC and PUBREL).

Use Case: Suitable for mission-critical applications (e.g., billing systems) where both delivery and duplicate prevention are essential.

Key Points:

- Most reliable but slower due to additional handshake.
- Guaranteed one-time delivery.

- Other important features of MQTT
  - **Retained Messages** : When a publisher sends a retained message, the broker stores it. New subscribers to the topic immediately receive the retained message upon subscribing.
    - Useful for sending the latest state or configuration of a device (e.g., "light is ON").
  - **Last Will and Testament** : A "last will" message is defined during connection. If the client disconnects unexpectedly, the broker publishes this message to a specified topic.
  - **Clean Session vs. Persistent Session**
    - **Clean Session:**
      When `clean: true`, no session state is stored between connections. All subscriptions and unacknowledged messages are cleared upon disconnection.
    - **Persistent Session:**
      When `clean: false`, session state is stored by the broker. Subscriptions and messages (QoS 1 or 2) remain active across connections.

MQTT brokers

An MQTT broker is the central hub in the MQTT protocol. It facilitates communication between publishers (devices or applications sending messages) and subscribers (devices or applications receiving messages).

Their tasks:

- Message Routing
- QoS Management
- Session Management
- Security and Auth
- Offline Messaging
- Scalability

Small-Scale Projects: Eclipse Mosquitto, Aedes, or NanoMQ.
Enterprise/Industrial IoT: HiveMQ, EMQX, or VerneMQ.
Cloud-Based Systems: AWS IoT Core or Google Cloud IoT Core.
Multi-Protocol Needs: RabbitMQ with MQTT plugin.

Activity 4: Sockets and Brokers

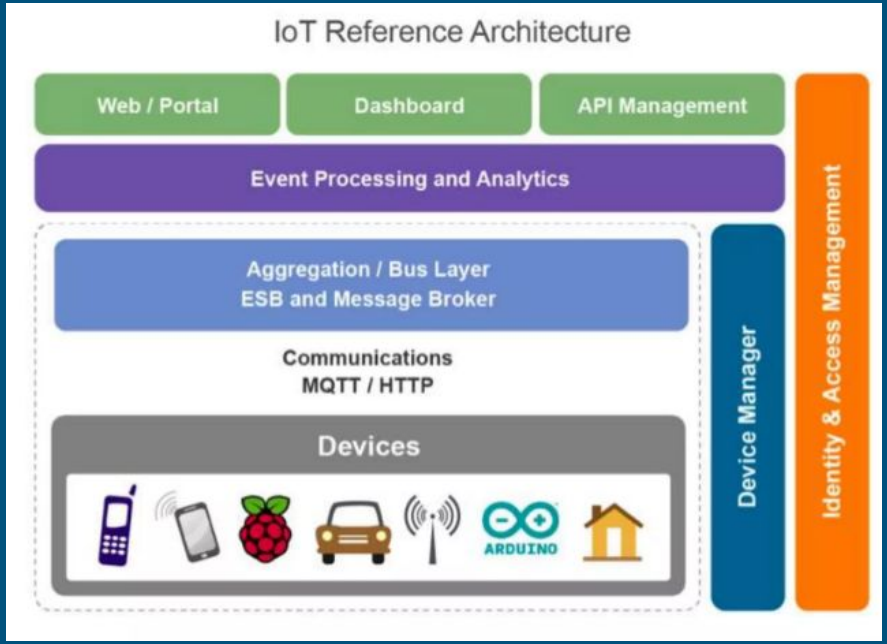
- Install Nodejs
- Basic Introduction of NodeJS
- Running client scripts and receive data
- ExpressJS web framework for simple IoT dashboards

- Docker commands
  - docker ps
  - docker stop CONTAINER_ID
  - docker remove CONTAINER_ID

16. Cloud computing for IOT

Cloud infrastructure refers to the network of servers and storage systems that host, manage, and process data collected from IoT devices. It enables seamless communication between devices and centralized data analytics.

- Scalability
- Remote Management
- Data storage and processing



Scalability

Dynamic Resource Allocation

- Cloud infrastructure enables IoT systems to scale resources (compute power, storage, and bandwidth) dynamically based on real-time demand.
- For example, during peak usage, additional resources can be automatically allocated to handle increased device communication and data processing.

Handling Growing Device Networks

- IoT networks can expand from a few devices to thousands without requiring physical upgrades to on-premises servers.
- Cloud platforms seamlessly integrate new devices and manage larger data flows.

Elastic Computing

- Cloud services offer elastic computing, meaning resources can be increased or decreased flexibly, reducing costs when demand is low and ensuring performance when demand is high.

Real-Time Data Processing

- As the volume of IoT data grows, scalable cloud platforms ensure that analytics and decision-making remain real-time, even with vast datasets.

Support for Big Data and AI

- IoT systems generate massive amounts of data; cloud scalability ensures this data can be stored, processed, and analyzed efficiently, supporting advanced AI models and machine learning.

Cost Efficiency

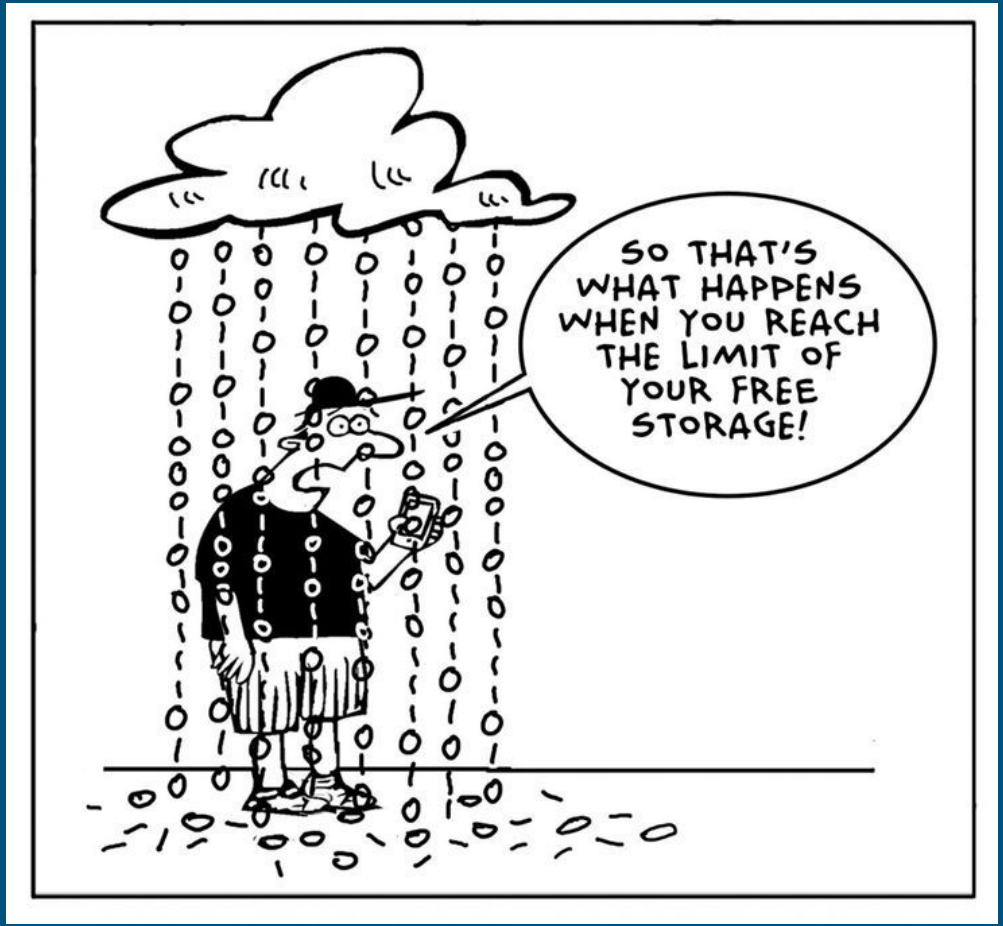
- Pay-as-you-go models ensure businesses only pay for the resources they use, avoiding upfront investments in large-scale infrastructure.

Disaster Recovery and Backup

- Scalable cloud solutions ensure data redundancy and quick recovery, even as the size of IoT systems and their data grows.

Example in Practice

- Smart traffic systems in cities dynamically scale to handle data from an increasing number of sensors and vehicles during peak traffic hours or special events.


SO THAT'S WHAT HAPPENS WHEN YOU REACH THE LIMIT OF YOUR FREE STORAGE!

Remote Management

Centralized Monitoring and Control

- Cloud infrastructure allows IoT devices to be monitored and controlled from a centralized dashboard, accessible from anywhere with an internet connection.
- Reduces the need for on-site personnel for device management.

Real-Time Access

- IoT systems provide real-time data updates to the cloud, enabling users to monitor device status and performance metrics in real-time.
- Supports instant decision-making and problem resolution.

Seamless Device Updates

- Firmware and software updates can be pushed remotely to IoT devices via the cloud, ensuring all devices run the latest versions without requiring physical access.
- Improves security by addressing vulnerabilities promptly.

Automation of Tasks

- Cloud-based platforms enable automation of repetitive tasks such as diagnostics, configuration updates, and device health checks, reducing manual intervention.

Enhanced Troubleshooting

- Cloud platforms allow remote diagnostics and troubleshooting of IoT devices, minimizing downtime.Logs and analytics can be accessed remotely to identify and resolve issues.

Geo-Independent Management

- IoT devices deployed in multiple geographic locations can be managed from a single interface, making global operations more efficient.Ideal for applications like logistics, agriculture, or industrial monitoring.

User Customization and Control

- Cloud platforms offer tools for users to set custom alerts, thresholds, and automation rules for their IoT systems.

"I'll be torturing you remotely today. We'll start with a series of goal setting meetings."

Data storage and processing

Centralized Data Storage

- Cloud platforms provide centralized repositories for IoT data, allowing seamless aggregation from distributed devices.Simplifies data management and ensures consistent access to records across locations.

Scalability for Massive Data

- Cloud infrastructure scales effortlessly to accommodate the vast amounts of data generated by IoT devices, from simple temperature readings to complex video streams.Ideal for handling exponential growth in IoT deployments.

Real-Time Data Processing

- Cloud-based platforms process incoming IoT data in real-time, enabling timely insights and actions.Useful for applications requiring immediate responses, such as predictive maintenance or security monitoring.

Advanced Analytics

- Cloud systems leverage powerful computing resources for in-depth analysis of IoT data, using techniques like AI and machine learning.
- Enables predictive analytics, trend identification, and anomaly detection.

Data Backup and Recovery

- Cloud platforms ensure reliable data backups and fast recovery in case of device failure or data loss.
- Enhances system reliability and minimizes downtime.

Cost-Efficient Storage Solutions

- Pay-as-you-go cloud storage eliminates the need for costly on-premises hardware.
- Flexible pricing models allow businesses to optimize costs based on storage and processing needs.

Integration with Other Services

- Cloud platforms easily integrate with other tools and services (e.g., data visualization tools, APIs, or external databases) to enhance IoT data utility.
- Simplifies workflows and improves decision-making processes.


CS602413
"Don't sit there hiding behind the data. Scrape it!"

17. AWS (Amazon Web Services)

Origins (2002-2003)

- Born from Amazon’s internal tools to handle e-commerce infrastructure.
- Vision: Scalable IT services accessible globally.

Official Launch (2006)

- Introduced **Amazon S3** (storage) and **Amazon EC2** (compute).
- Pioneered the **"pay-as-you-go"** model, democratizing IT resources.

Global Expansion (2007-2015)

- Added services like RDS, Elastic Beanstalk, CloudFront, and Lambda.
- Supported major clients like Netflix and NASA.

Market Leader (2016-Present)

- Offers 200+ services in AI, IoT, serverless, and more.
- Supports industries from finance to healthcare, contributing $80B+ revenue in 2022.

AWS global infrastructure

AWS tools for IOT

IOT core

- Full managed MQTT Message Broker.
- HTTPS and LoraWAN
- Mange device fleets
- End to end encryption
- Filter, transform, and act upon device data on the fly, based on your defined business rules.

Other Useful tools of AWS

1. **Amazon EC2 (Elastic Compute Cloud)** 🖥️

Overview:
Amazon EC2 provides secure, resizable compute capacity in the cloud, allowing businesses to scale their applications seamlessly.

Key Features:

- Wide Range of Instance Types:
  - Instances optimized for compute, memory, storage, and GPU workloads.
  - Examples: General Purpose (T2, T3), Compute Optimized (C5, C6), Storage Optimized (I3, D2).
- Flexible Pricing Options:
  - On-Demand Instances: No long-term commitments, pay-as-you-go.
  - Reserved Instances: Save up to 75% compared to On-Demand with 1- or 3-year terms.
  - Spot Instances: Bid for unused capacity at a significantly lower cost.
- Auto Scaling:
  - Automatically adjusts the number of instances to handle traffic demands.

- Secure Networking:
  - Supports Virtual Private Cloud (VPC) integration for secure networking.
  - Allows granular access control using Security Groups and IAM roles.
- Global Availability:
  - EC2 instances are available in multiple Availability Zones across AWS regions.

2. Amazon S3 (Simple Storage Service) 🗂️
Overview:

A highly durable and scalable object storage service, perfect for data backup, archiving, big data analytics, and application data hosting.

Key Features:

- Storage Classes for Cost Optimization:

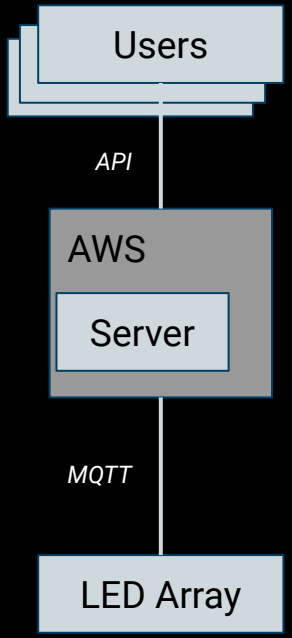
  S3 Standard: For frequently accessed data.
  S3 Glacier: For long-term archival storage with retrieval options.
  S3 Intelligent-Tiering: Automatically moves data to the most cost-effective storage class.

- Data Security: Supports encryption for data at rest and in transit (AWS Key Management Service integration. Access controls using Bucket Policies, ACLs, and IAM policies.
- Lifecycle Management:Automate the movement of objects to different storage classes based on policies.
- High Durability and Availability:99.999999999% (11 nines) durability. Automatic replication across multiple Availability Zones.
- Event Notifications:Trigger AWS Lambda functions, Amazon SQS, or SNS when changes occur in an S3 bucket.

3. Amazon Lambda

- serverless and event-driven computing service that lets you run code for virtual applications or backend services automatically.
- You no need to worry about servers and clusters when working with solutions using Amazon Lambda.
- It is also cost-effective where you have to only pay for the services you use.
- As a user, your responsibility is to just upload the code and Lambda handles the rest. Using Lambda, you get precise software scaling and extensive availability.
- With hundreds to thousands of workloads per second, AWS Lambda responsibly handles code execution requests. It is one of the best services provided by AWS for developers.

Activity 4 : `Internet of a Thing` Controlling Remotely



Activity 4 : `Internet of a Thing` Controlling Remotely

- Setting UP AWS and walkaround in EC2 instance
- Setting up a NodeJS server
- Setting postman in clients setup
- Sending API requests over postman and control GPIO