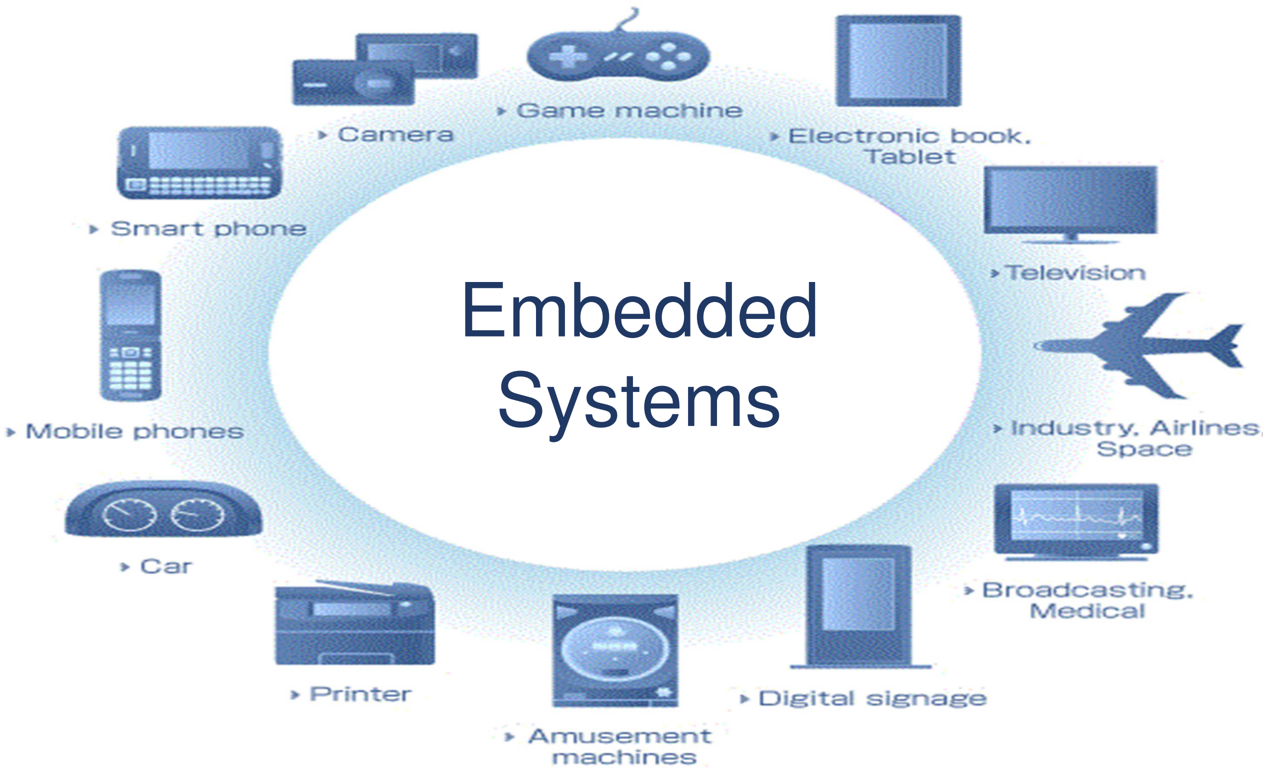
SoRTES Lab - Session 1

Ashok Thangarajan, Stefanos Peros, Emekcan Aras

DistriNet

What are these?



DistriNet

Embedded Computing

- Developed for a specific purpose
- Resource constrained environment
- Low processor speed

General Purpose Computing

- Wide range of tasks
- Unlimited resources (!= embedded world)
- Fast processor speed (!= embedded world)

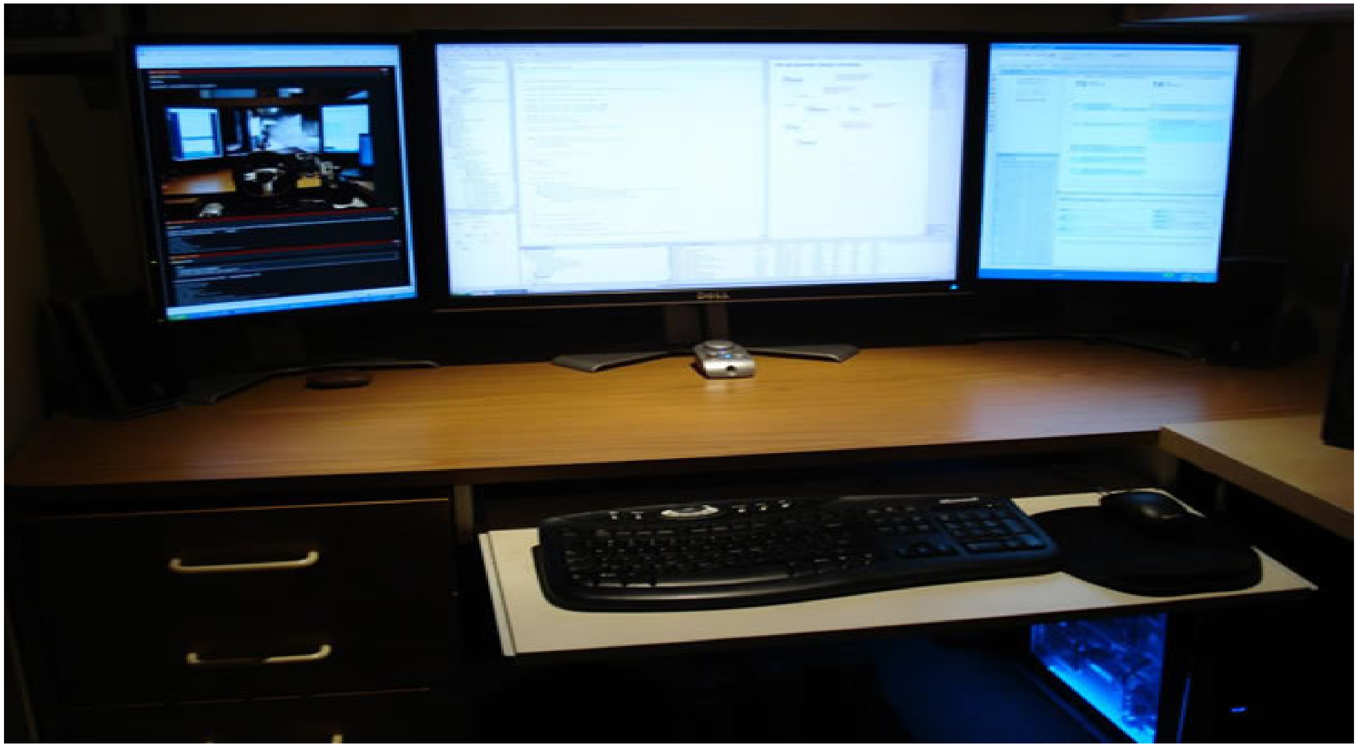
Comparison

	Embedded Computing	General Purpose Computing
Processor Speed	10 - 800 MHz	Several GHz
Memory	Few KB to MB	Several GB’s
Operating system	RTOS – various embedded OS	General purpose OS
Objective	Specialized purpose	General purpose computing

Embedded Programming

How do we program this?

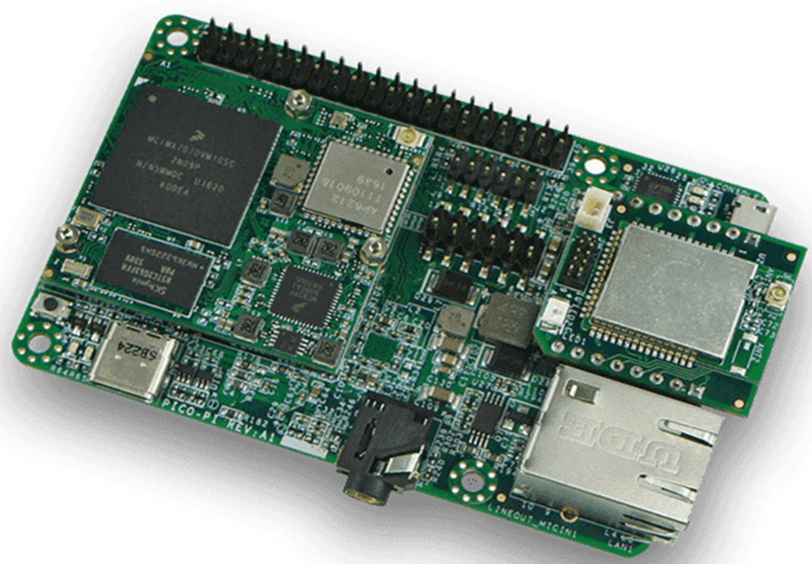
- General purpose application development



DistriNet

How do we program this?

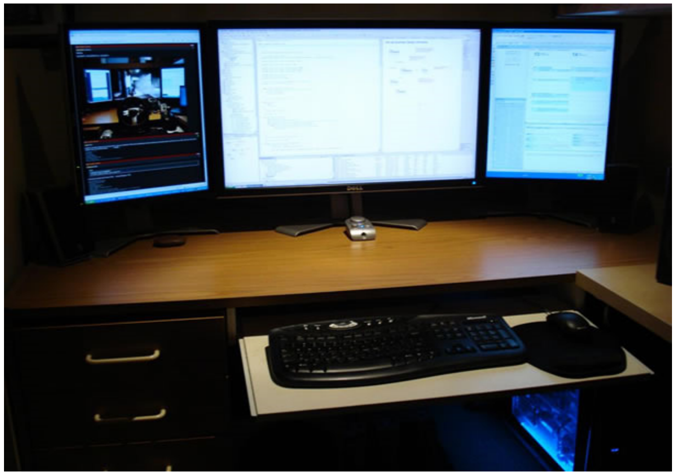
› Typical embedded development space

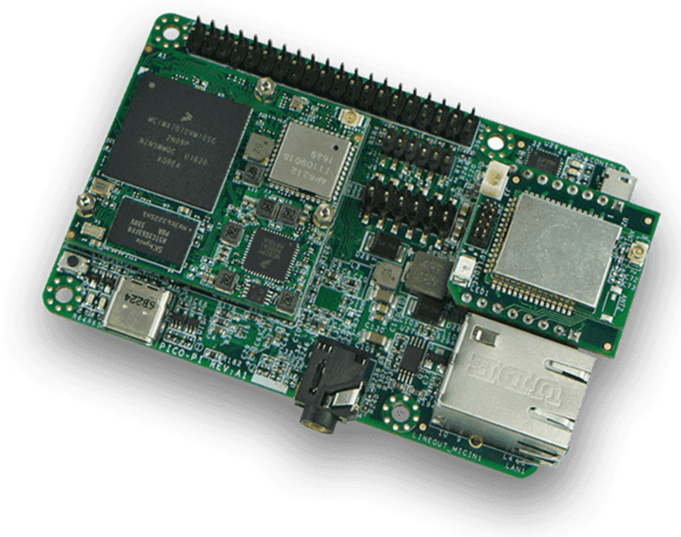


DistriNet

How do we program this?

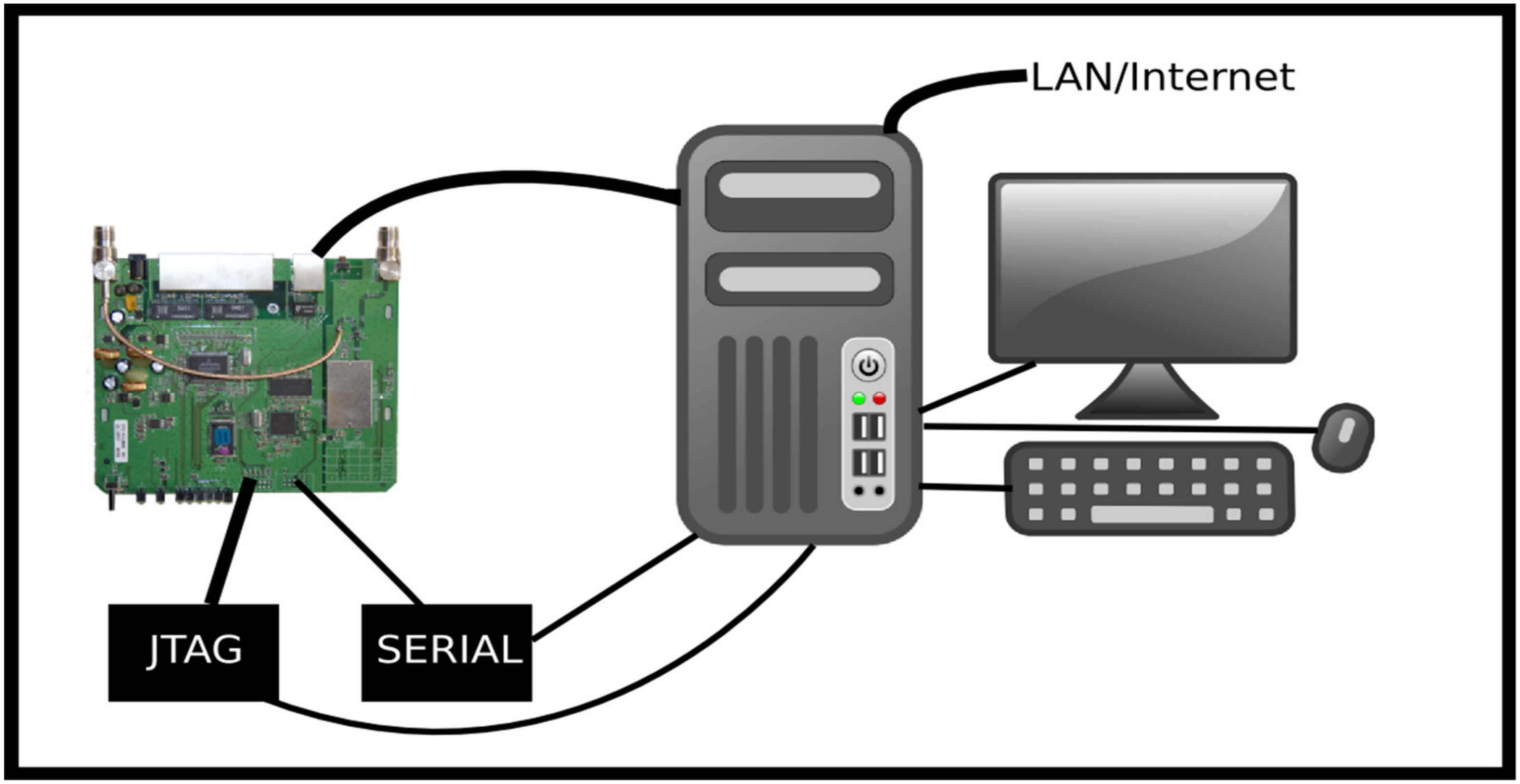
- General purpose application development → Embedded Programming





DistriNet

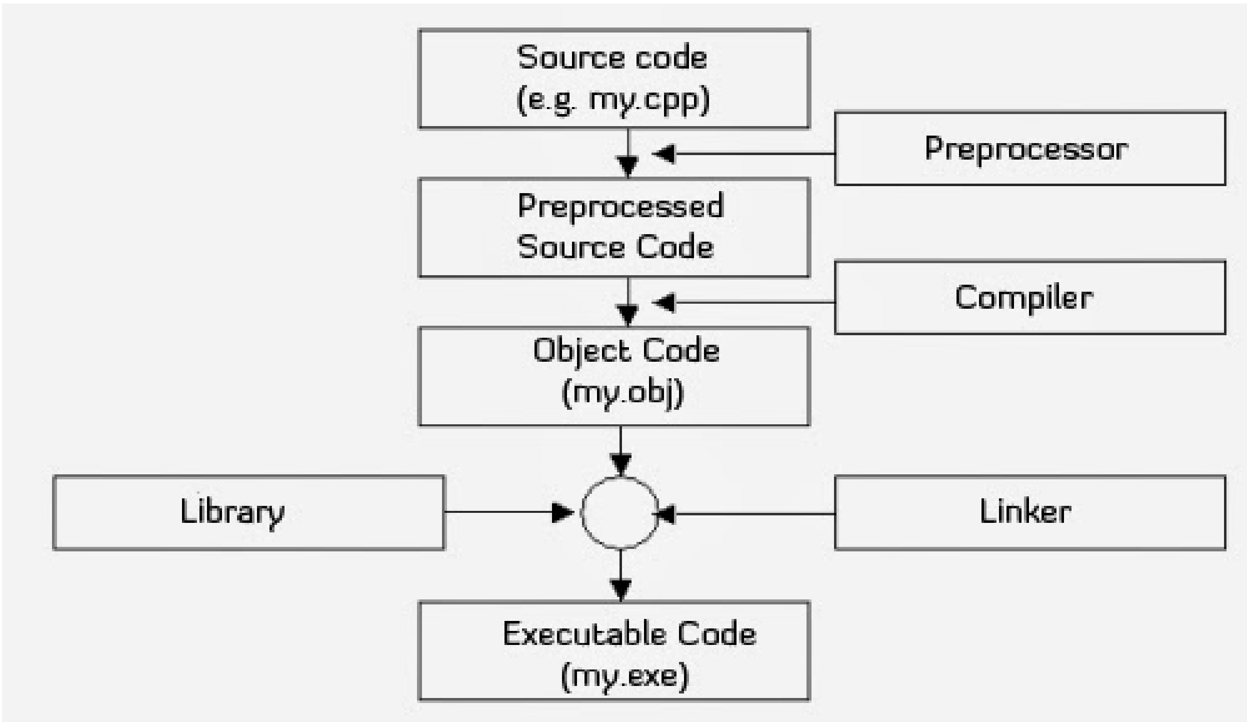
Develop in a host machine



Tool Chain

- Cross Compiler
  - To compile a program for another machine architecture
- Assembler
  - Assembly to object code
- Linker
  - Linking many object code and libraries to an executable or library.
- Debugger
  - Interface to debug programs, generally with external hardware support
- Other helper binaries
  - Archives, nm, readelf, etc..

Typical compilation phases in a compiler



Programming Interface

- Multiple programming interfaces
- Common ones are:
  - JTAG
  - SPI/UART
  - USB
  - Arduino serial bootloader

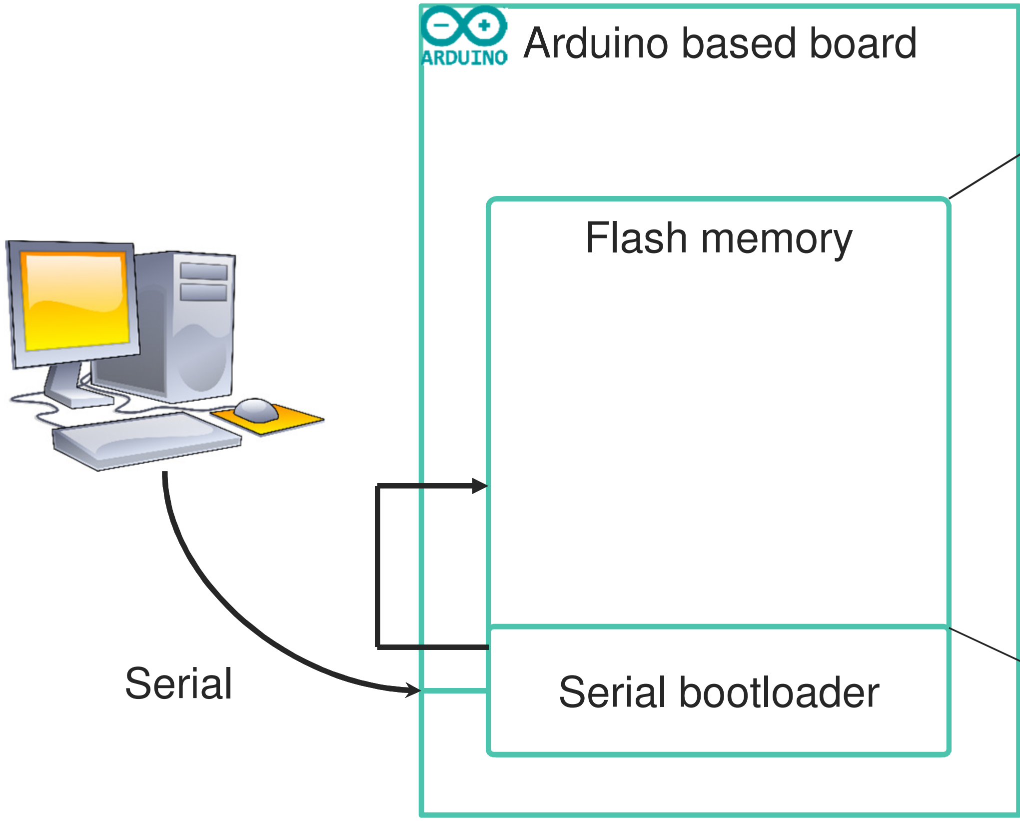
Boot-up Sequence

- Initialize processor
- Done by processor-specific boot sequence, generalized to:
  - Power on Reset
  - Initial State
  - Bootloader starts running
  - Bootstraps the operating system (OS)
  - OS starts up and completes boot up sequence
  - Starts the application
  - Wait to load program on **programming interface**

Boot-up Sequence

- Some processors boot sequence is simple
- After reset vector, starts executing code at 0x0000
- Many advanced controllers and embedded processors have complex boot sequences

Arduino Bootloader




```
int main(void)
{
    init();

    initVariant();

#if defined(USBCON)
    USBDevice.attach();
#endif

    setup();

    for (;;) {
        loop();
        if (serialEventRun) serialEventRun();
    }
```

Arduino IDE

- Code Editor
- Tool chain (For various architectures)
- Serial bootloader
- Various standard libraries
- Support for many boards
- Serial monitor

Arduino code structure

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```


DistriNet

Example code

```
int pushButton = 2;

void setup() {
    Serial.begin(9600);
    pinMode(pushButton, INPUT);
}

void loop() {
  int buttonState = digitalRead(pushButton);

  Serial.println(buttonState);
  delay(1);
}
```

Why and why not Arduino?

- Why
  - Easy to learn
  - Focus on the application aspects
  - Rapid prototyping

- Why not?
  - Libraries not optimized
  - Non-functional requirements of product may not be fulfilled
    - Meeting industry standards for industrial products
    - Meeting safety standards for safety related products

Data Sheet

- Sometimes we have to read a Digital IO
  - How do we know if a pin is an IO pin or it is exposing an internal peripheral?
  - I2C timing characteristics?
  - How much current an IO pin can drive?

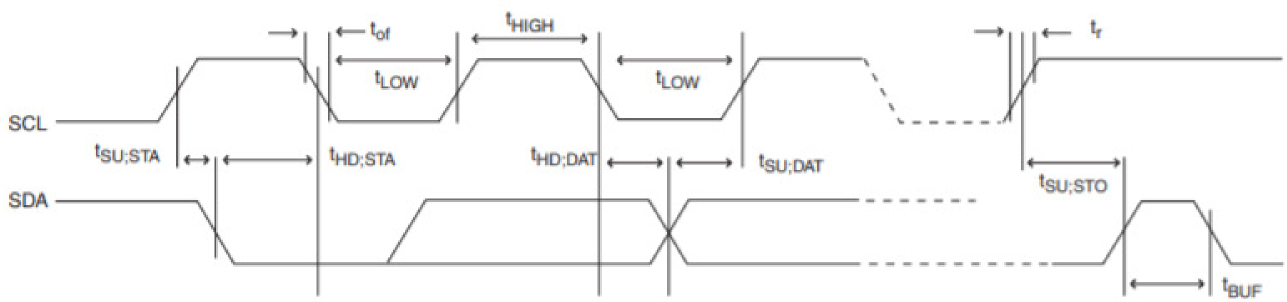
Data Sheet – from AtMega32U4 Datasheet

- Current per IO pin

DC Current per I/O Pin 40.0mA

- I2C timing characteristics

Figure 29-3. 2-wire Serial Bus Timing



Data Sheet

- Datasheet and Technical specifications provide this info
- This is important due to the heterogeneity of embedded domain
- We may program various machine architecture, and wide range of microcontrollers with varied peripherals. Eg: 8051, PIC, AVR, MIPS, Xtensa, V850, ARM

Sensors

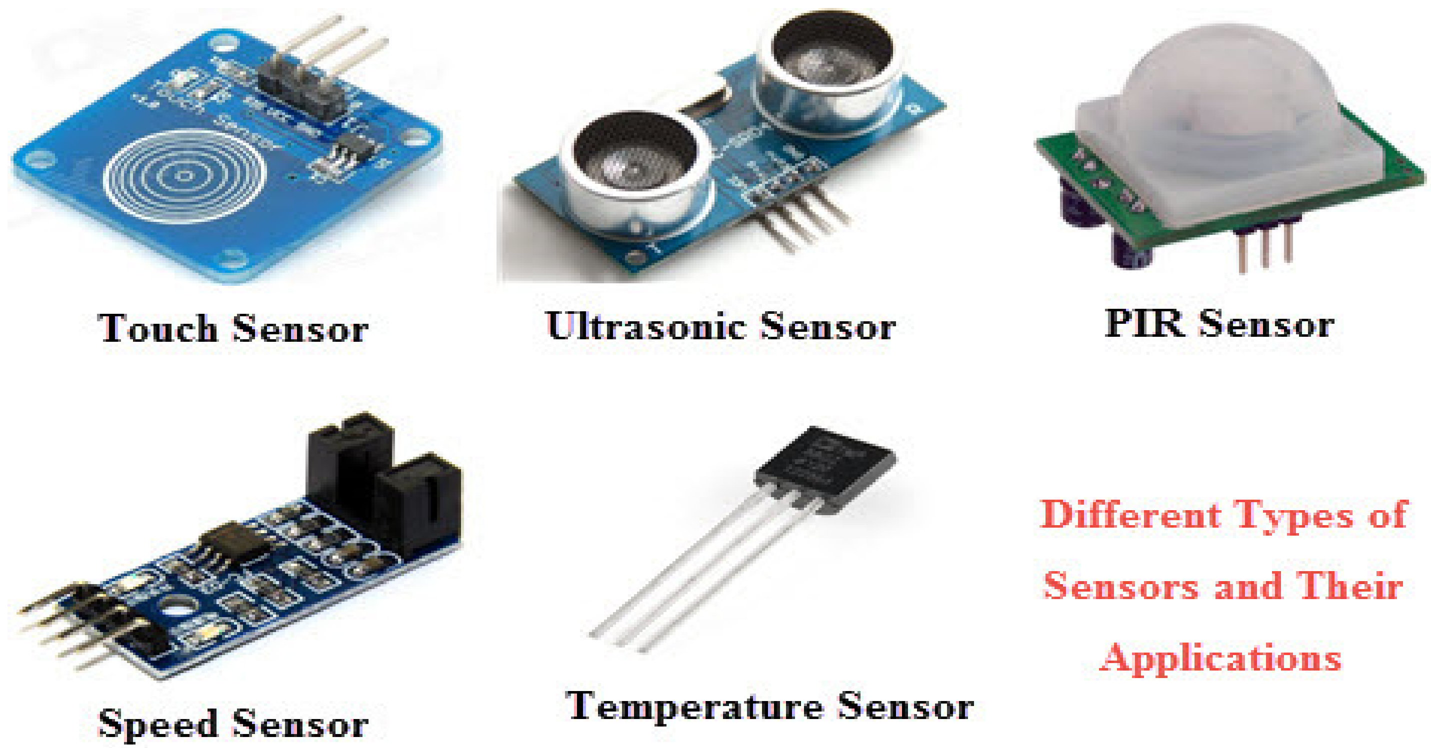
Sensors – (1/2)

- Microcontrollers need digital inputs from environment
- Sensing changes in physical elements – Sensors
- Sensors sense changes in environment
- Convert them to voltage levels
- Or covert them to processor's language

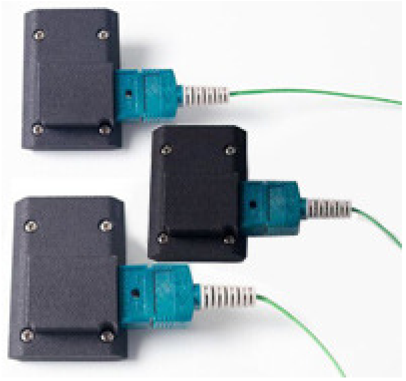
Sensors – (2/2)

- Sensor output to microcontrollers:
  - I2C/SPI/UART
  - Analog output to processors ADC
  - Digital output via IO pins

Some sensors for Arduino



Industrial Sensors



DistriNet

Calibration

Calibration – (1/2)

- Sensors need calibration
- Slight offsets caused by:
  - Internal variations
  - Manufacturing variations

Calibration – (2/2)

- Use a known quantity
- Measure it using the sensor
- Adjust for variations
- Eg: Weighing machines

Example - Weighing machine calibration

DistriNet

SoRTES Project

Time synchronized sensing device

- Based on LoRa
- Gateway sends beacons in intervals of 2-10 sec
- Receive the beacon
  - Write the beacon contents to a database
  - Acquire temperature and write to the database
  - Send the temperature to the gateway

Time synchronized sensing device

- After transmitting temperature, go to low power mode
- After receiving 20 beacons, go to ultra low power mode
  - Wake up only through external interrupt
- Controller should additionally support 3 commands:
  - Command to read latest temperature
  - Command to print all temperature
  - Enter low power mode

Time synchronized sensing device

- Mandatory to use FreeRTOS
- You may use FreeRTOS lib for Arduino

List of materials

- 2x LoRa board (868 MHz, AtMega32U4)
- 2x Micro-USB cable for Arduino

- Note:
  - Sign and get one for your team
  - Keep it safe and return it back (04-Dec)

Libraries for the board

- Refer to the link below
- https://bsfrance.fr/lora-long-range/1345-LoRa32u4-II-Lora-LiPo-Atmega32u4-SX1276-HPD13-868MHZ-EU-Antenna.html

Exercises

Exercise 1

- Display the following in serial port (console), from your Arduino application
  - “Enter LED status (on/off):”
- Read the user input value from serial port into your application
- If value is “on”, display the following in serial port from your Arduino application
  - “Enter the blink rate (1-60 sec):”
- Print user provided values to serial port
  - “You have selected LED on/off. Blink rate is xx sec”
- After the user selected values are displayed in serial port, the blink pattern should be seen on the onboard LED
  - Info: Use a timer library to set the blink rate

Exercise 2

- Create a timestamp module (A simple counter should be enough)
- Create a small database (You may use Arduino library)
- Calibrate and read the temperature value from Arduino and store it into the database
- Power off and on the board, your database must be maintained without being deleted. (Look for Arduino EEPROM module)

Exercise 3

- Instead of database from the previous exercise, use a linked list for storage
- The linked list must be persisted across board power on/off cycle.
- If you need to refresh pointers in C, you can check this out:
  - https://www.learn-c.org/en/Welcome

DistriNet

DistriNet

Thank you!

https://distrinet.cs.kuleuven.be/