# While we wait...

1. Clone the workshop repository (QR code)
2. Install LightBlue and/or nRF Connect app in your smartphone
3. Create an environment for running the examples
   - Use the requirements.txt file to install the necessary packages
   - **OR** convenience commands in README

# Demystifying Bluetooth Low Energy

**Last update:** 2024-10-21

# Who am I?

## Jorge Miranda

- PostDoc student at Aarhus University, Denmark
  - IoT applications for healthcare and agriculture
  - Linux based embedded systems
- Contacts
  - jorge@anycolouryoulike.dev
  - Github: @jmpcm

# What is this workshop about

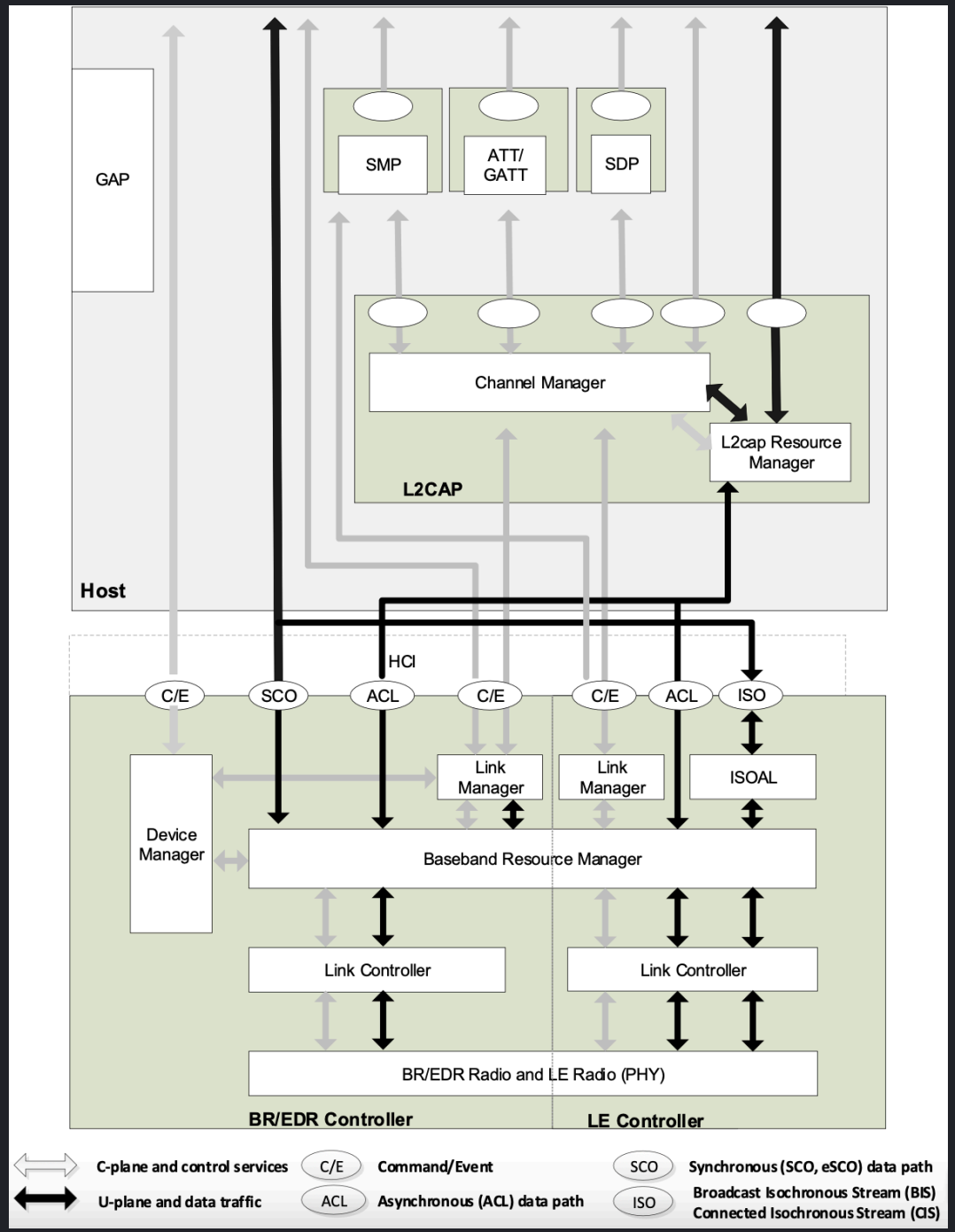
- Enable **Bluetooth Low Energy** (BLE) in your applications
- ***The workshop is just an introduction***
- We'll not address:
  - Bluetooth BR/EDR (version 2.x)
  - Aspects such as security, hardware, audio, serial communication, ...
  - Bluetooth Mesh
- Theory alternated with exercises
  - Roughly 10/15 minutes per exercise
- Let's make this interactive - raise your hand ✋

# Agenda

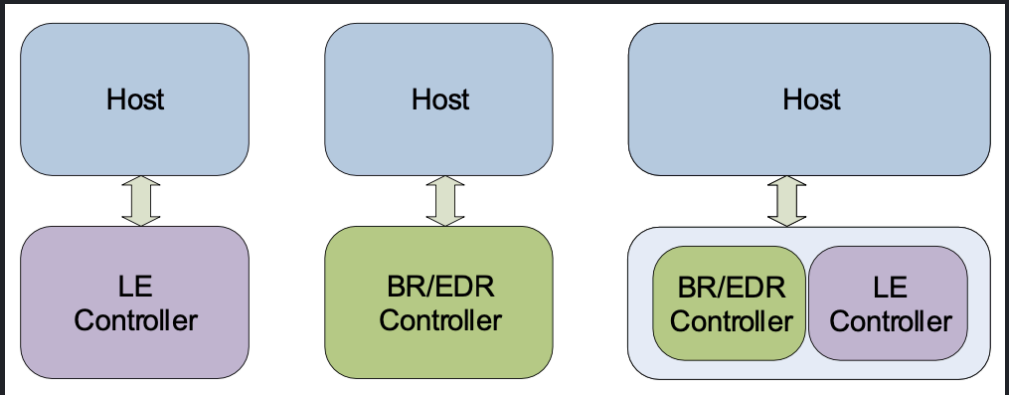
1. Bluetooth Low Energy architecture
2. BLE Central
3. GATT architecture
4. BLE Peripheral

# Part #1: Bluetooth Low Energy Architecture

- Very complex architecture... The Bluetooth Core specification (version 5.4) is 3141 pages long!
  - But it specifies everything: hardware, security, applications
  - Independent from other standards, such as IEEE
  - Lower layers are important for hardware vendors, or OS developers
  - For applications, upper layers (GAP and GATT) are what matters




**Source:** Bluetooth Core Specification Version 5.4, Vol 1, Part A, page 203




**Source:** Bluetooth Core Specification Version 5.4, Vol 1, Part A, page 188

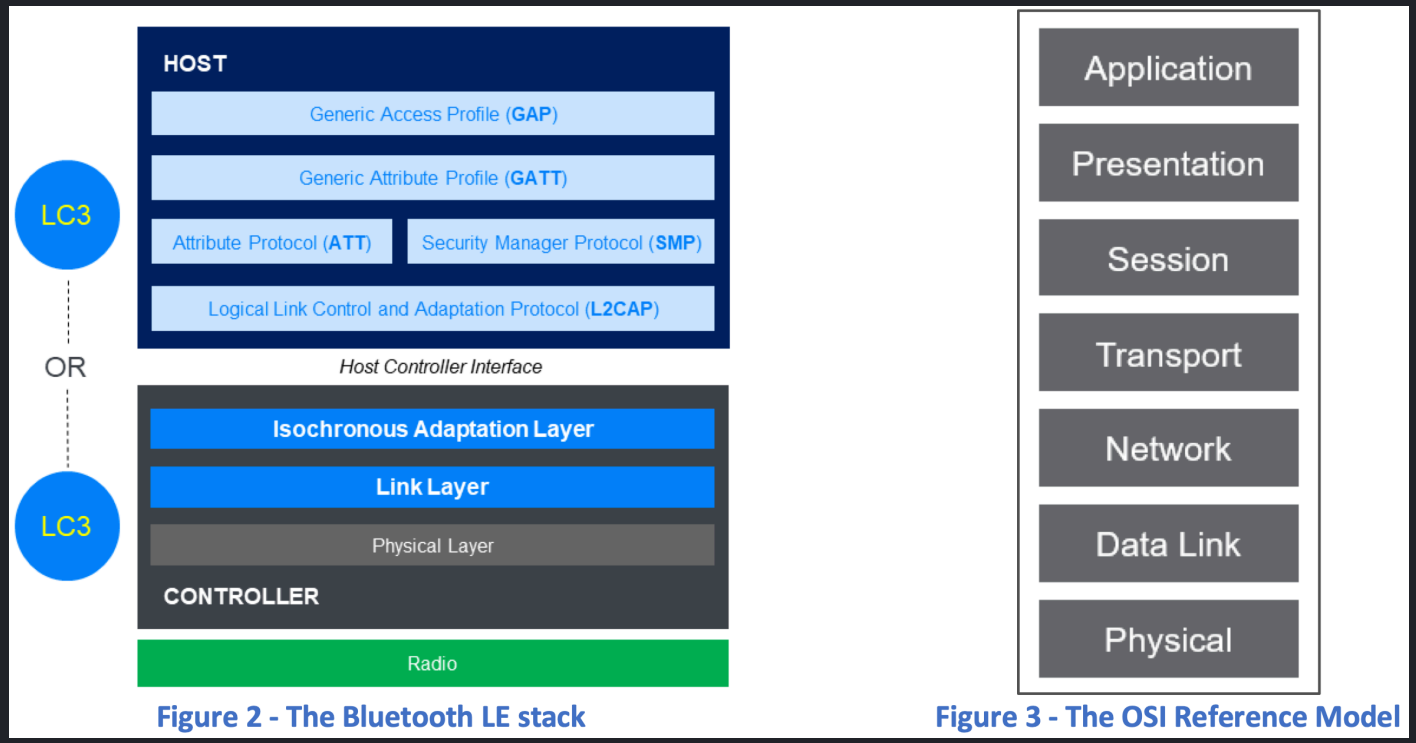
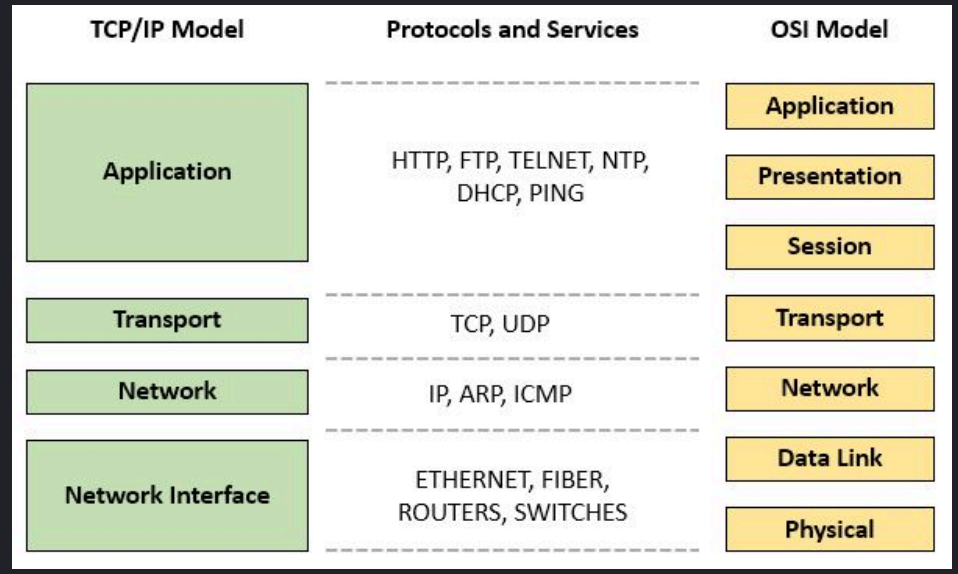



Figure 2 - The Bluetooth LE stack

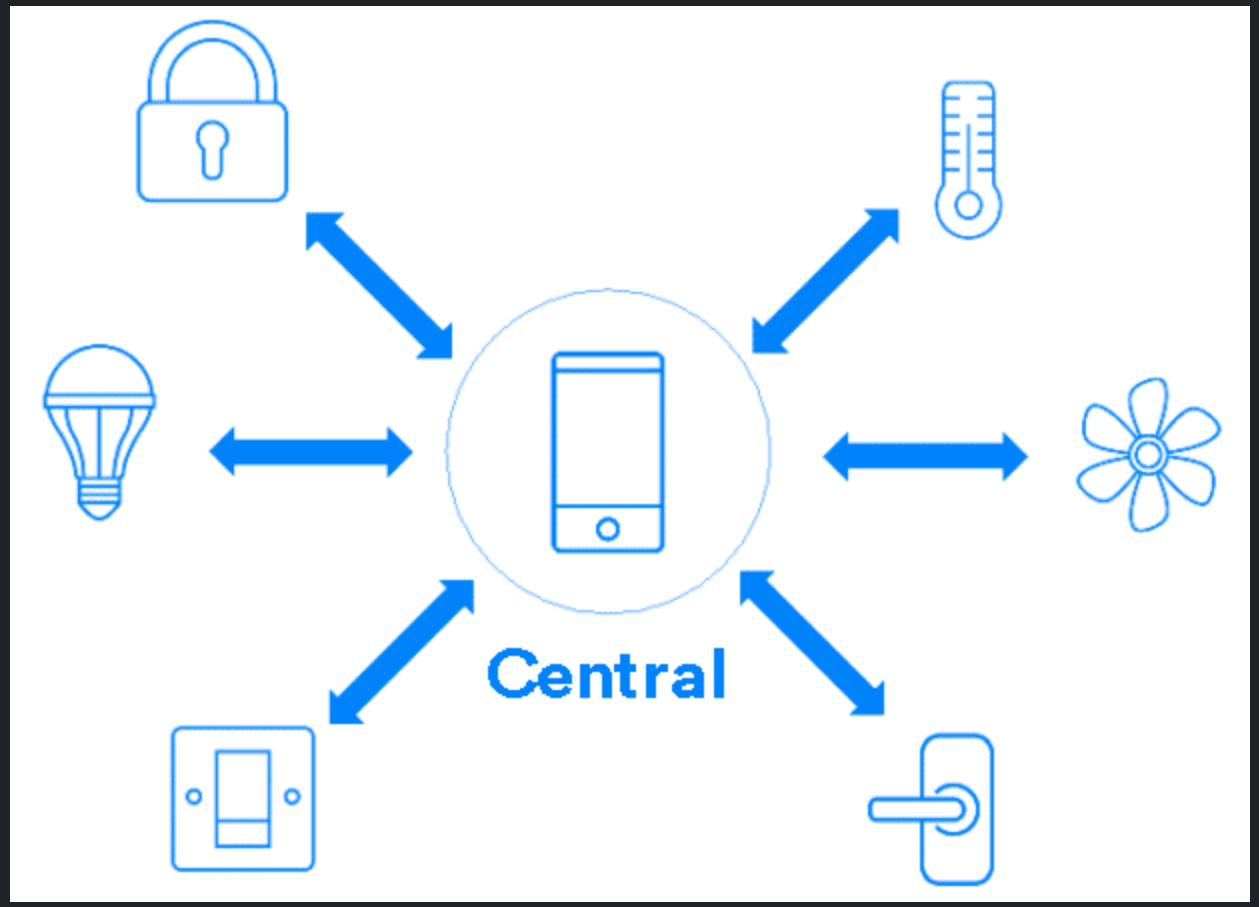



Figure 3 - The OSI Reference Model




**Source:** TCP/IP model vs OSI model https://mzacki.github.io/osi/

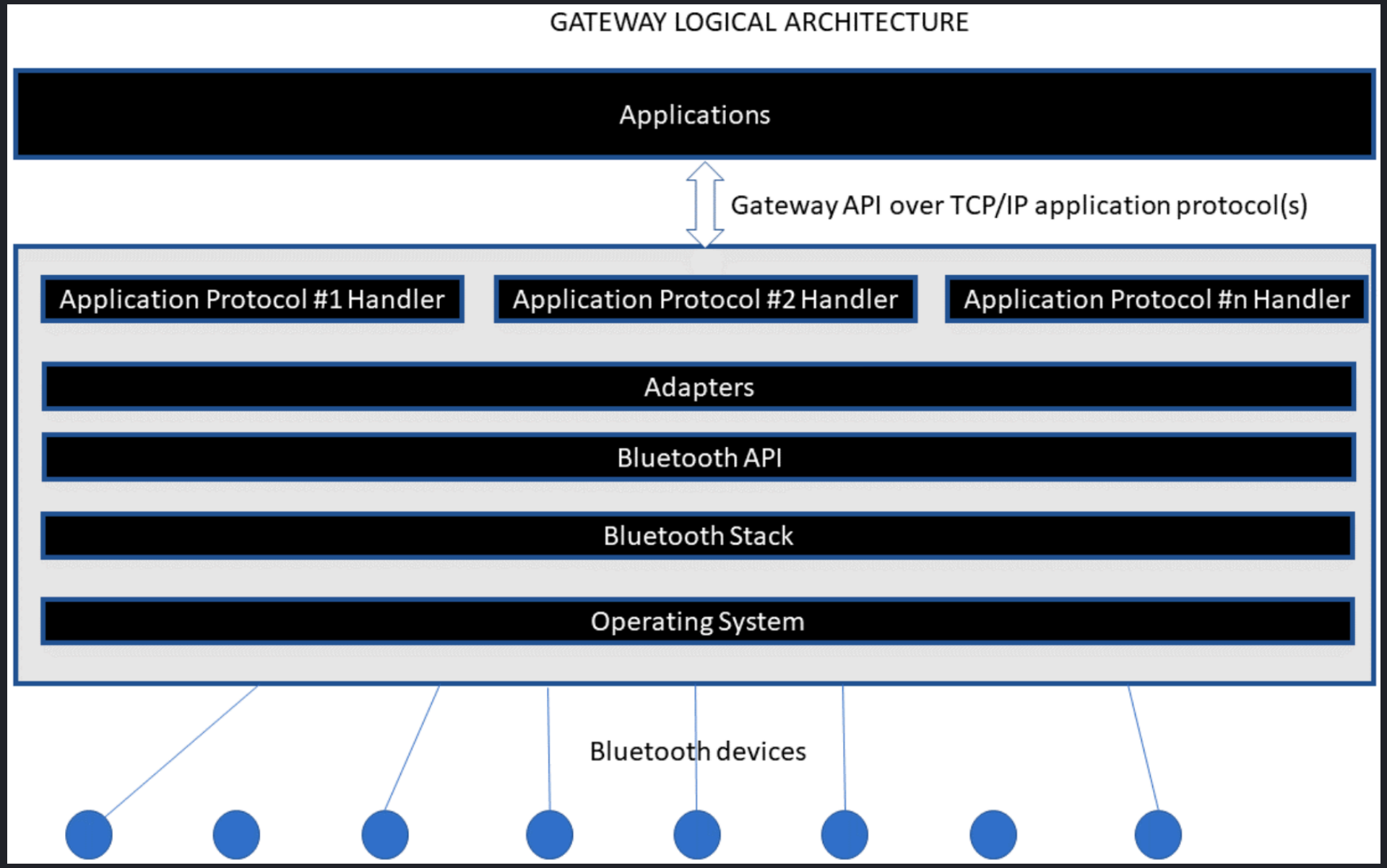
**Source:** The Bluetooth Low Energy Primer




**Source:** BLE Central and Peripherals
https://www.bluetooth.com/blog/how-one-wearable-can-connect-with-multiple-smartphones-or-tablets-simultaneously/

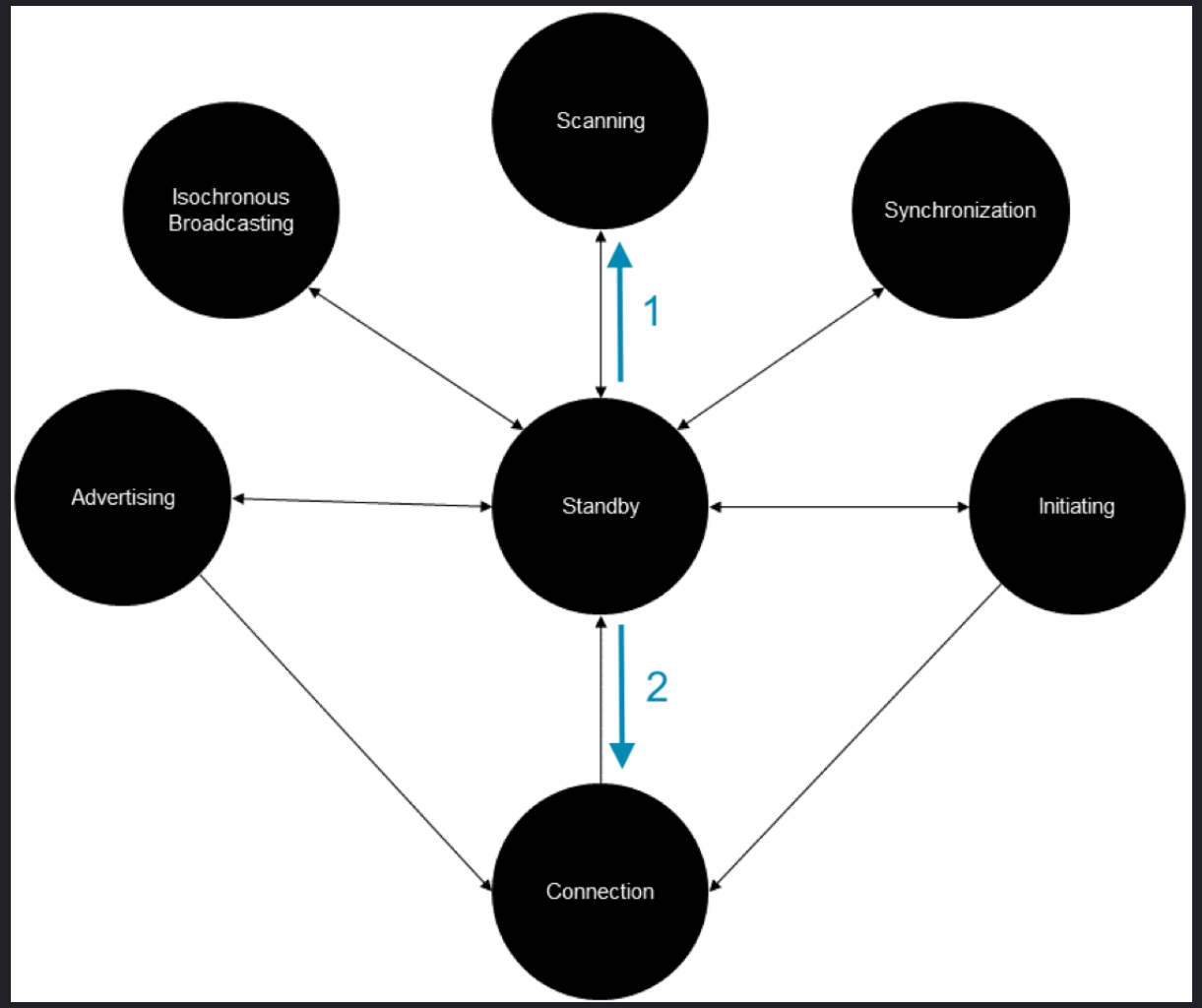
1. **Peripheral**: can be connected to by a Central device
2. **Central**: is able to initiate the establishment of a connection with a Peripheral device

# Part #2: BLE Central




**Source:** Bluetooth Gateway Study Guide: First Steps

# How does a BLE Central work




**Source:** The Bluetooth Low Energy Primer

1. Scan/discover peripherals
2. Connect to the peripheral
3. Get GATT services and characteristics of the peripheral
4. Read/write GATT characteristic, or subscribe to notifications/indications

# Great!

Let's write some code!

# WAIT!

The Bluetooth stack is different in all operating systems!...

# Bleak

*GATT client software, capable of connecting to BLE devices acting as GATT servers. It is designed to provide a asynchronous, cross-platform Python API to connect and communicate with e.g. sensors.*

GitHub | Documentation

# Exercise #1: discover BLE peripherals

*class* **bleak.BleakScanner**(*detection_callback: Optional[AdvertisementDataCallback] = None, service_uuids: Optional[List[str]] = None, scanning_mode: Literal['active', 'passive'] = 'active', *, bluez: BlueZScannerArgs = {}, cb: CBScannerArgs = {}, backend: Optional[Type[BaseBleakScanner]] = None, **kwargs*)

*async classmethod* **BleakScanner.discover**(*timeout: float = 5.0, *, return_adv: Literal[False] = False, **kwargs*)→ List[BLEDevice] [source]

*async classmethod* **BleakScanner.discover**(*timeout: float = 5.0, *, return_adv: Literal[True], **kwargs*)→ Dict[str, Tuple[BLEDevice, AdvertisementData]]

Documentation

# BLE Advertisements

- *LE Advertising Broadcast (or simply advertising) provides a connectionless communication mode. It may be used to* ***transfer data*** *or to* ***indicate the availability of a Peripheral device*** *to be connected to.* [1]
- *(...) supports the communication of data in* ***one direction only****, from the advertising device to scanning devices but such devices* ***may reply to advertising packets*** *with PDUs that request further information* [1]
- Defined in the GAP (Generic Access Profile) layer

# (Some) Advertisement data types [2]

- **Service UUID:** list of Service or Service Class UUIDs
- **Local name:** local name assigned to the device
- **TX power:** transmitted power level of the packet containing the data type. (...) should be the radiated power level
- **RSSI:** received signal strength, in dBm
- **Service data:** service UUID with the data associated with that service
- **Manufacturer data:** used for manufacturer specific data. The first two data octets shall contain a company identifier from Assigned Numbers. The interpretation of any other octets (...) shall be defined by the manufacturer (...)

Full list in Assigned numbers; more info in [3]

# BLE UUIDs

- Used to identify GATT services, characteristics and descriptors
- SIG defines a list of standard UUID values. The full list of is available in the Assigned Numbers document (section 3.4 GATT Services).
- Possible for the vendor to define custom UUIDs
- The standard UUID is `0000XXXX-0000-1000-8000-00805F9B34FB`, where `XXXX` is a 16-bit UUID
  - Not mandatory to use this format
  - In the remaining of this workshop, we'll refer just to the 16-bit UUID
- In our example, an environmental sensing device has the service UUID `0000181A-0000-1000-8000-00805F9B34FB`

# Example standard BLE UUIDs

- Device Information Service: `0x180A`
- Environmental Sensing Service: `0x181A`
- Health Thermometer Service: `0x1809`
- Heart Rate Service: `0x180D`
- Weight Scale Service: `0x181D`

# Exercise #2: use BleakScanner's context manager

(homework 😉)

# Exercise #3: connect to a peripheral

*class* **bleak.BleakClient**(*address_or_ble_device: Union[BLEDevice, str], disconnected_callback: Optional[Callable[[BleakClient], None]] = None, services: Optional[Iterable[str]] = None, *, timeout: float = 10.0, winrt: WinRTClientArgs = {}, backend: Optional[Type[BaseBleakClient]] = None, **kwargs*) [source]

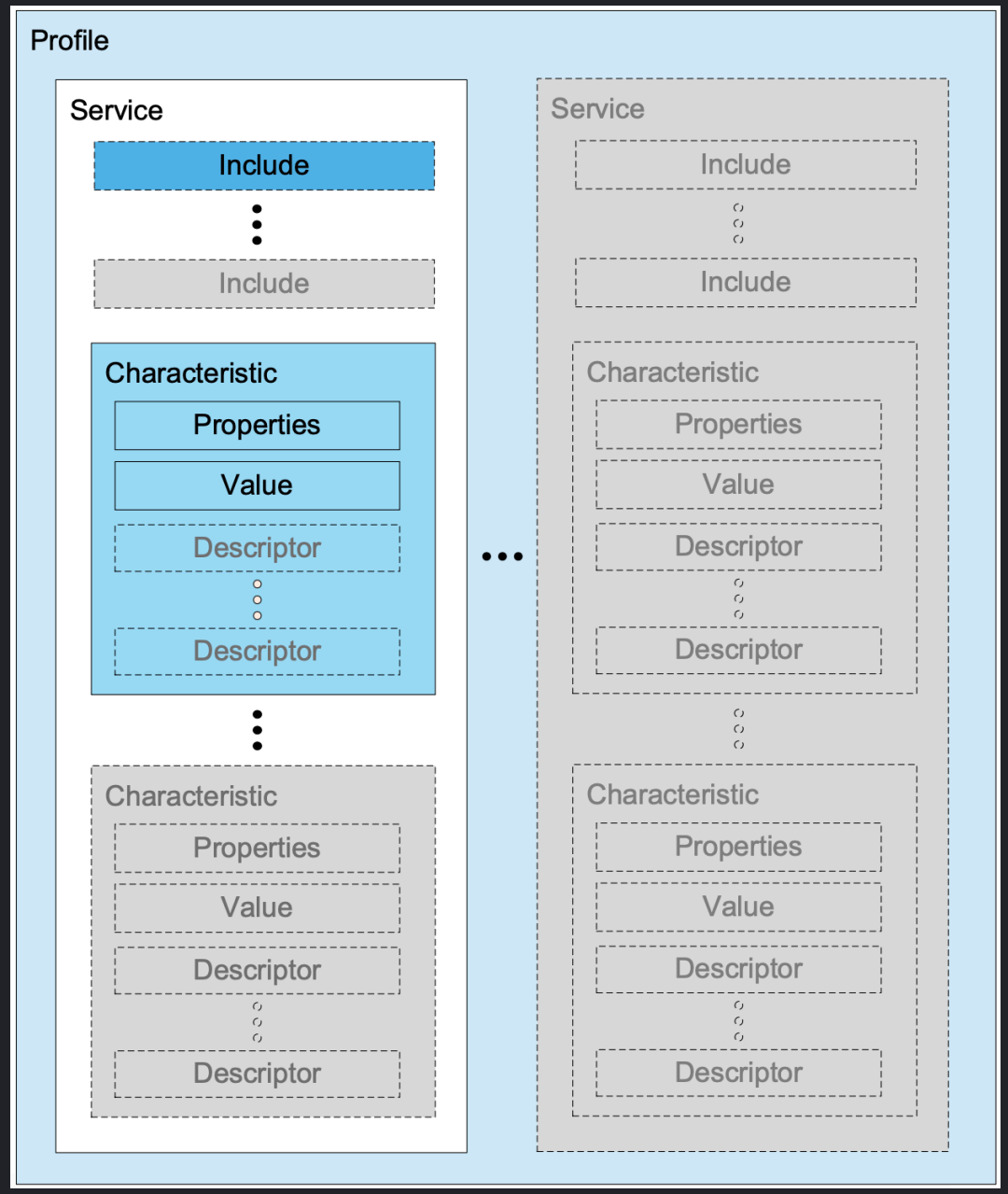
```python
import asyncio
from bleak import BleakClient

async def main():
    async with BleakClient("XX:XX:XX:XX:XX:XX") as client:
        # Read a characteristic, etc.
        ...

    # Device will disconnect when block exits.
    ...
```

Documentation

# Part #3: Generic Attribute Profile (GATT)

# What is GATT?

- Establishes how data will be organized and exchanged over a BLE connection [4]
- The Bluetooth Special Interest Group (SIG) defines several GATT profiles and services. These can be found in [5]
- The attributes (or data) in a peripheral can be accessed after a connection with a central is established. Once the connection is established, the peripheral stops advertising.
- Information is exchanged exclusively between two connected devices, i.e. a peripheral can only be connected to a central at a time




**Source:** The Bluetooth Low Energy Primer

- **Profile:** this entity does not exist in a peripheral, but it is only a concept to describe a group of services available.
- **Service:** *(...) provide a context within which to use the characteristics that they contain (...). Often services correspond to a primary feature or capability of a device.* [1]
- **Characteristic:** individual items of state data (or attributes) that are part of a service. These can belong to more than one service.
- **Descriptor:** *(...) belong to some characteristics and can contain metadata (...) or might provide some means of controlling the behaviour of a characteristic. Characteristics have zero or more descriptors attached to them.* [1]

# Exercise #4: list GATT services and characteristics

# Device Information Service

Service UUID `0x180A`

| Characteristic Name | Characteristic Qualifier | Mandatory Properties | Optional Properties | Security Permissions |
|---|---|---|---|---|
| Manufacturer Name String | O | Read | | None |
| Model Number String | O | Read | | None |
| Serial Number String | O | Read | | None |
| Hardware Revision String | O | Read | | None |
| Firmware Revision String | O | Read | | None |
| Software Revision String | O | Read | | None |
| System ID | O | Read | | None |
| IEEE 11073-20601 Regulatory Certification Data List | O | Read | | None |
| PnP ID | O | Read | | None |

**Source:** Device Information Service specification, section 3

Specification | Assigned numbers

# Device Information Characteristics

## Manufacturer Name String `0x2A29`

| Field | Data Type | Size (in octets) | Description |
|---|---|---|---|
| Manufacturer Name | utf8s | variable | UTF-8 string |

## Model Number String `0x2A24`

| Field | Data Type | Size (in octets) | Description |
|---|---|---|---|
| Model Number | utf8s | variable | UTF-8 string |

## Serial Number String `0x2A25`

| Field | Data Type | Size (in octets) | Description |
|---|---|---|---|
| Serial Number | utf8s | variable | UTF-8 string |

**Source:** GATT Specification Supplement, sections 3.147, 3.153 & 3.197

Assigned numbers | GATT Specification Supplement

# Exercise #5: read the characteristics of the Device Information service

# Environmental Sensing Service

Service UUID `0x181A`

| Characteristic | Requirement | Mandatory Properties | Optional Properties | Security Permissions |
|---|---|---|---|---|
| ESS Characteristic | C.1 | Read | Notify, Extended Properties | None |
| Descriptor Value Changed | C.2, C.3 | Indicate | | None |

***Table 3.1:*** *Requirements for each ESS Characteristic*

C.1: At least one ESS Characteristic shall be exposed.

C.2: Mandatory if at least one of the following descriptors can be changed by the Server for at least one ESS Characteristic: ES Measurement, ES Trigger Setting, ES Configuration, Characteristic User Description.

C.3: Mandatory if the Write property is supported for the Characteristic User Description descriptor.

## 3.1 ESS Characteristics

The Server shall expose at least one ESS Characteristic.

Each ESS Characteristic is defined in the Environmental Sensing Service Characteristics table in [2].

**Source:** Environmental Sensing Service specification, sections 3 & 3.1

Specification | Assigned numbers

# Environmental Sensing Characteristics

## 6.1 Environmental Sensing Service

### 6.1.1 Permitted Characteristics

The list below specifies the characteristics that are permitted for use with the Environmental Sensing Service [7].

- Ammonia Concentration
- Apparent Wind Direction
- Apparent Wind Speed
- Barometric Pressure Trend
- Carbon Monoxide Concentration
- Dew Point
- Elevation
- Gust Factor
- Heat Index
- Humidity
- Irradiance
- Magnetic Declination
- Magnetic Flux Density - 2D
- Magnetic Flux Density - 3D
- Methane Concentration
- Nitrogen Dioxide Concentration
- Non-Methane Volatile Organic Compounds Concentration
- Ozone Concentration
- Particulate Matter - PM1 Concentration
- Particulate Matter - PM10 Concentration
- Particulate Matter - PM2.5 Concentration
- Pollen Concentration
- Pressure
- Rainfall
- Sulfur Dioxide Concentration
- Sulfur Hexafluoride Concentration
- Temperature
- True Wind Direction
- True Wind Speed
- UV Index
- Wind Chill

**Source:** Assigned Numbers, section 6.1

# Environmental Sensing Characteristics

## Temperature `0x2A6E`

| Field | Data Type | Size (in octets) | Description |
|---|---|---|---|
| Temperature | sint16 | 2 | Base Unit: org.bluetooth.unit.thermodynamic_-temperature.degree_celsius<br>Represented values: M = 1, d = -2, b = 0<br>Unit is degrees Celsius with a resolution of 0.01 degrees Celsius.<br>Allowed range is: -273.15 to 327.67.<br>A value of 0x8000 represents 'value is not known'.<br>All other values are prohibited. |

## Humidity `0x2A6F`

| Field | Data Type | Size (in octets) | Description |
|---|---|---|---|
| Humidity | uint16 | 2 | Base Unit: org.bluetooth.unit.percentage<br>Represented values: M = 1, d = -2, b = 0<br>Unit is in percent with a resolution of 0.01 percent.<br>Allowed range is: 0.00 to 100.00<br>A value of 0xFFFF represents 'value is not known'.<br>All other values are prohibited. |

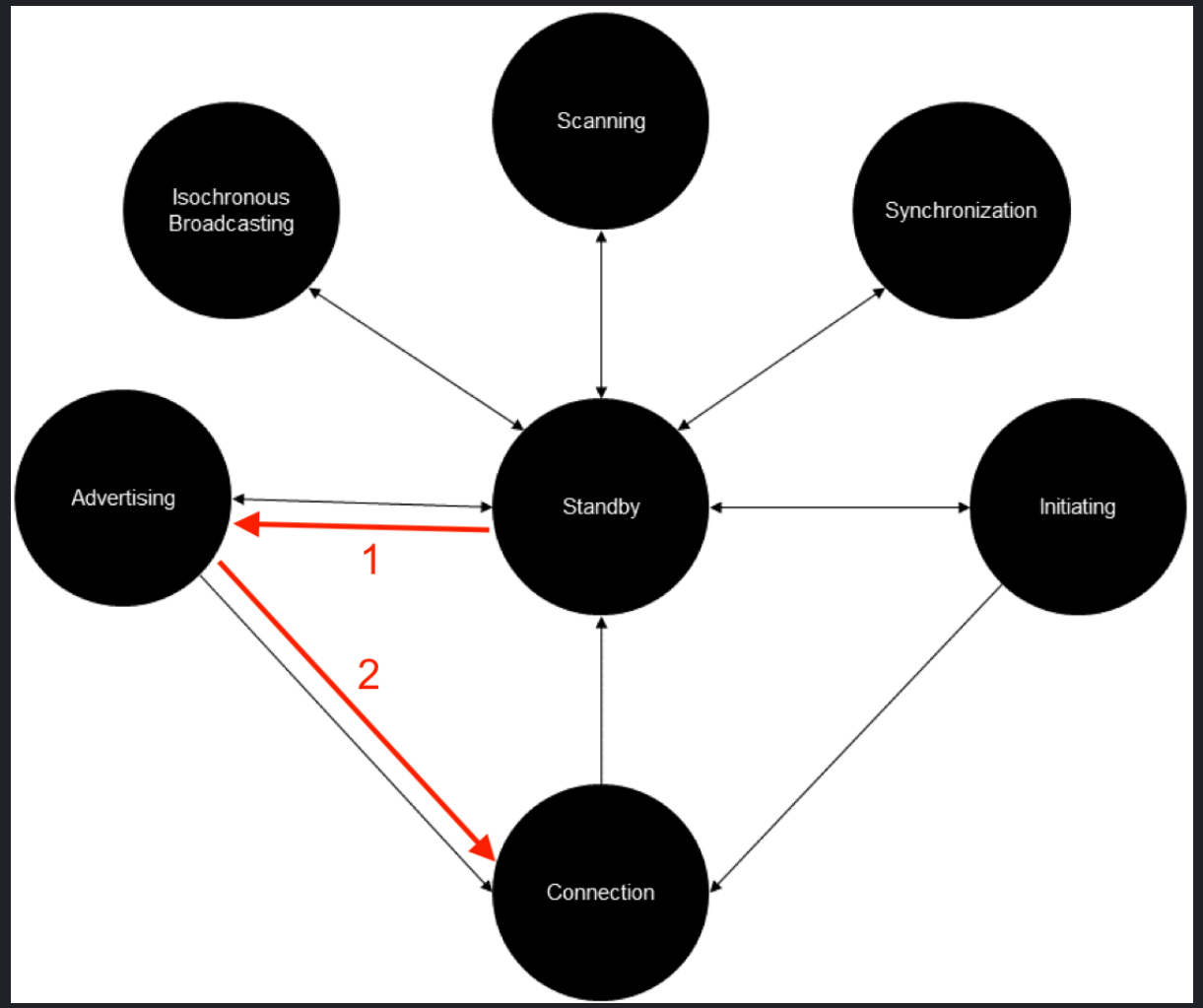
**Source:** GATT Specification Supplement, sections 3.324 & 3.206

Assigned numbers | GATT Specification Supplement

# Exercise #6: receive notifications from the characteristics of the Environmental Sensing service

# Part #4: BLE Peripheral

# How does a BLE Peripheral work




**Source:** The Bluetooth Low Energy Primer

1. Advertise
2. Establish connection with a Central, on request

# Bless

*OS-independent python package for creating a BLE Generic Attribute Profile (GATT) server to broadcast user-defined services and characteristics.*

GitHub

# Exercise #7: implement a GATT peripheral

```
class BlessServer(
    name: str,
    loop: AbstractEventLoop | None = None,
    **kwargs: Any)

async BlessServer.add_new_service(uuid: str) -> Coroutine[Any, Any, None]

async BlessServer.add_new_characteristic(
    service_uuid: str,
    char_uuid: str,
    properties: GATTCharacteristicProperties,
    value: bytearray | None,
    permissions: GATTAttributePermissions
  ) -> Coroutine[Any, Any, None]

async BlessServer.start()

async BlessServer.stop()
```

# Q&A

# Thank you!

# References


[1] The Bluetooth Low Energy Primer
[2] Supplement to the Bluetooth Core Specification
[3] Advertising Works, Part 2; Bluetooth Blog
[4] Microchip Introduction to Bluetooth Low Energy, Generic Attribute Profile (GATT)
[5] Bluetooth Specifications and Documents
[6] Designing and Developing Bluetooth® Internet Gateways