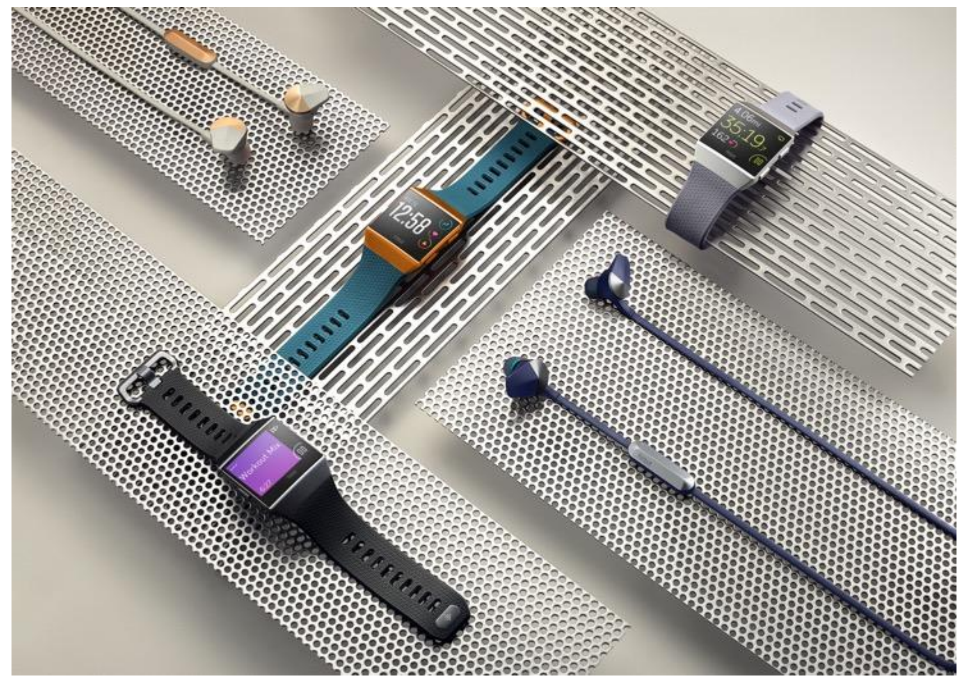


Android: Bluetooth Low Energy Communication

04.07.2019

Objectives

- Learn the basic properties of Bluetooth Low Energy protocol
- Integrate your app with a third-party module, in this case the Nordic *thingylib*
- Develop a simple app that communicates with the Nordic Thingy device

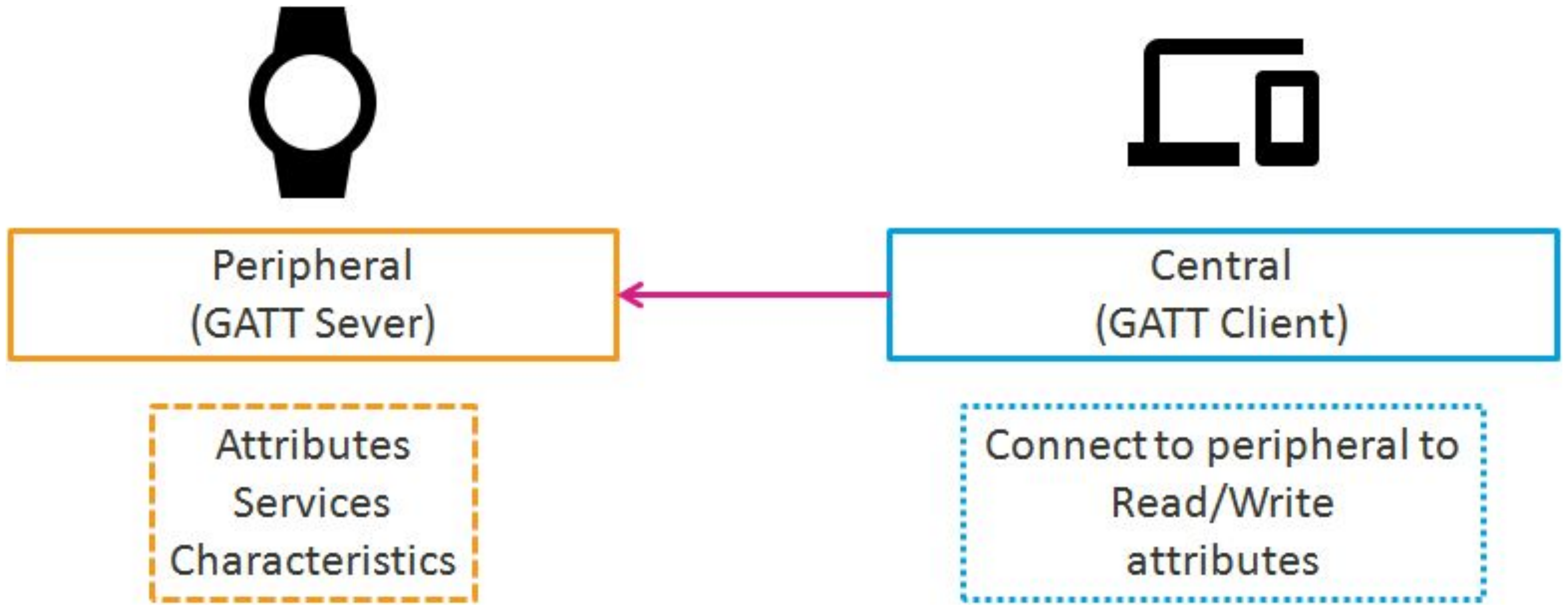
Recap last sessions

- In the previous two sessions you have:
  - Created Activities
  - Used UI elements such as Buttons, TextViews and RecyclerViews
  - Used permissions to enable for Bluetooth and Network communication
  - Interacted with a public API using Retrofit for HTTP operations
  - Used Intents and BroadcastReceivers to communicate between Android components

Bluetooth Low Energy - BLE

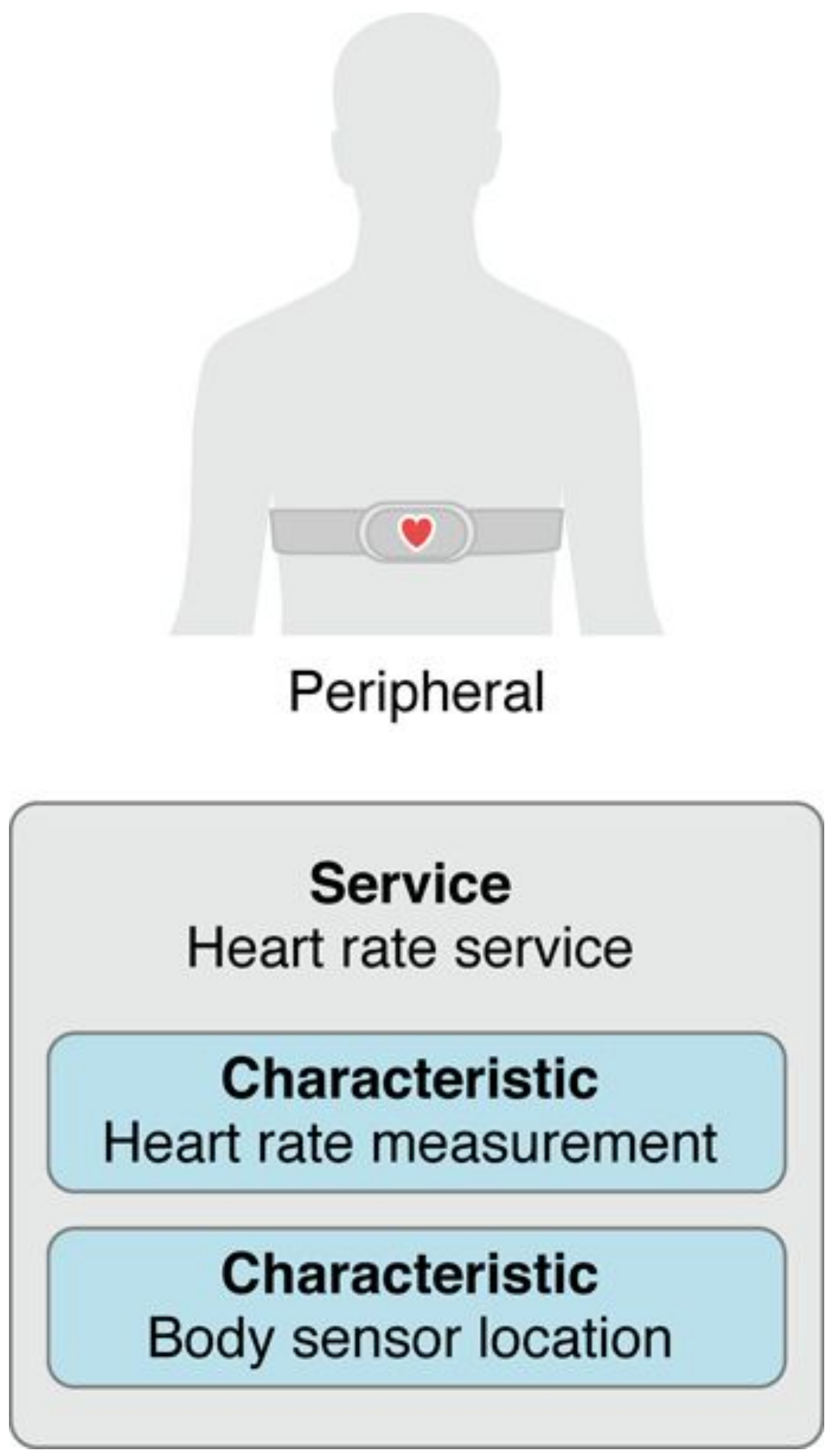
- Bluetooth LE or simply BLE
- Short-range radio waves, up to 100m
- 2.400 - 2.4835 GHz
- up to 2Mbits/s

Server - Client architecture

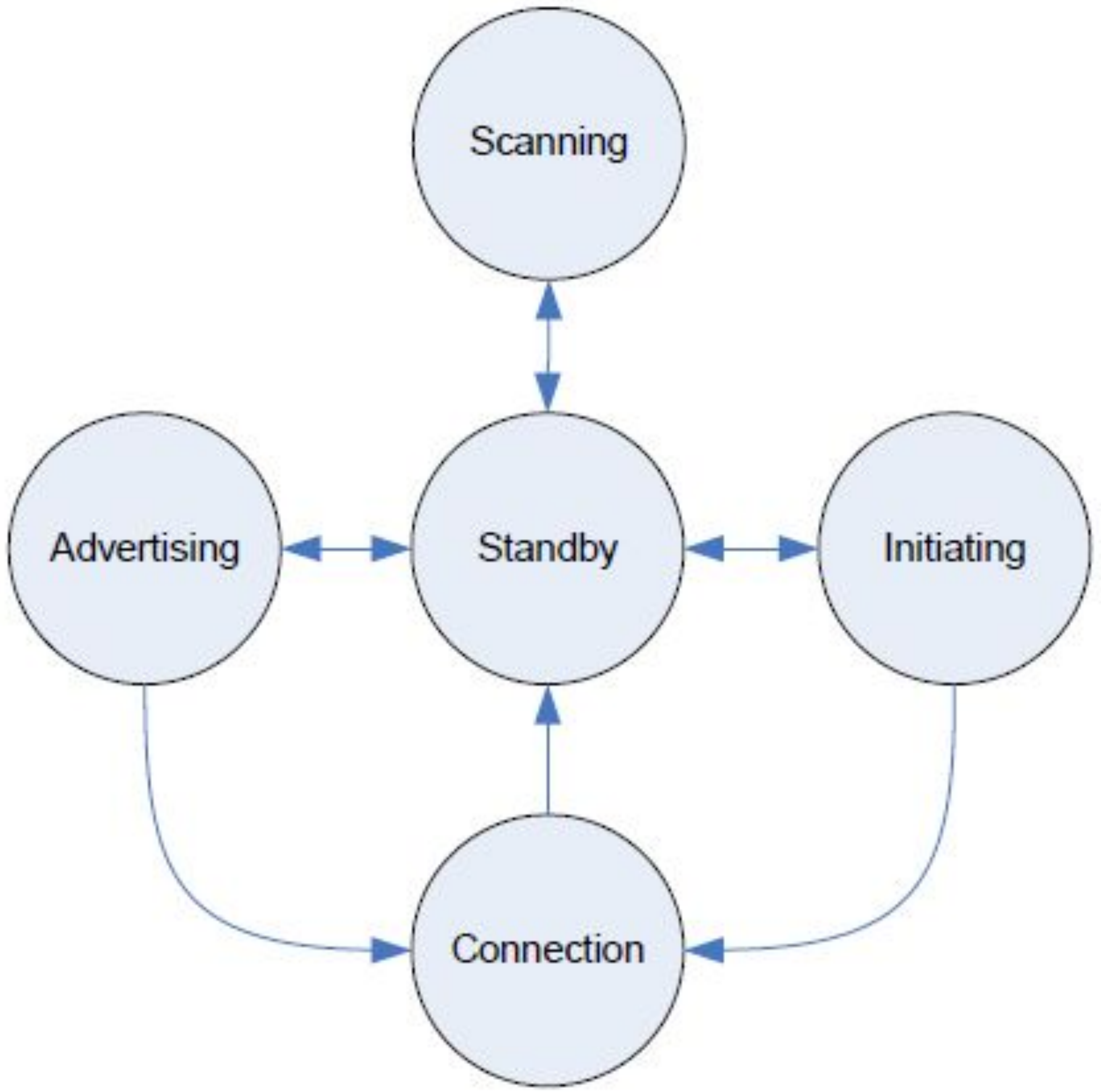


BLE profile

- Service
- Characteristic
- Descriptor



BLE States



Nordic Thingy Board

- Prototyping platform
- Out-of-the box experience, easy-to-use
  - firmware already installed
  - iOS and Android apps for configuration and showing its sensors values in real-time, using Bluetooth LE communication
- Lots of sensors
  - temperature, pressure, air quality and color
  - motion sensors
  - Microphone & speaker
  - RGB led & button

Demo

Asynchronous programming

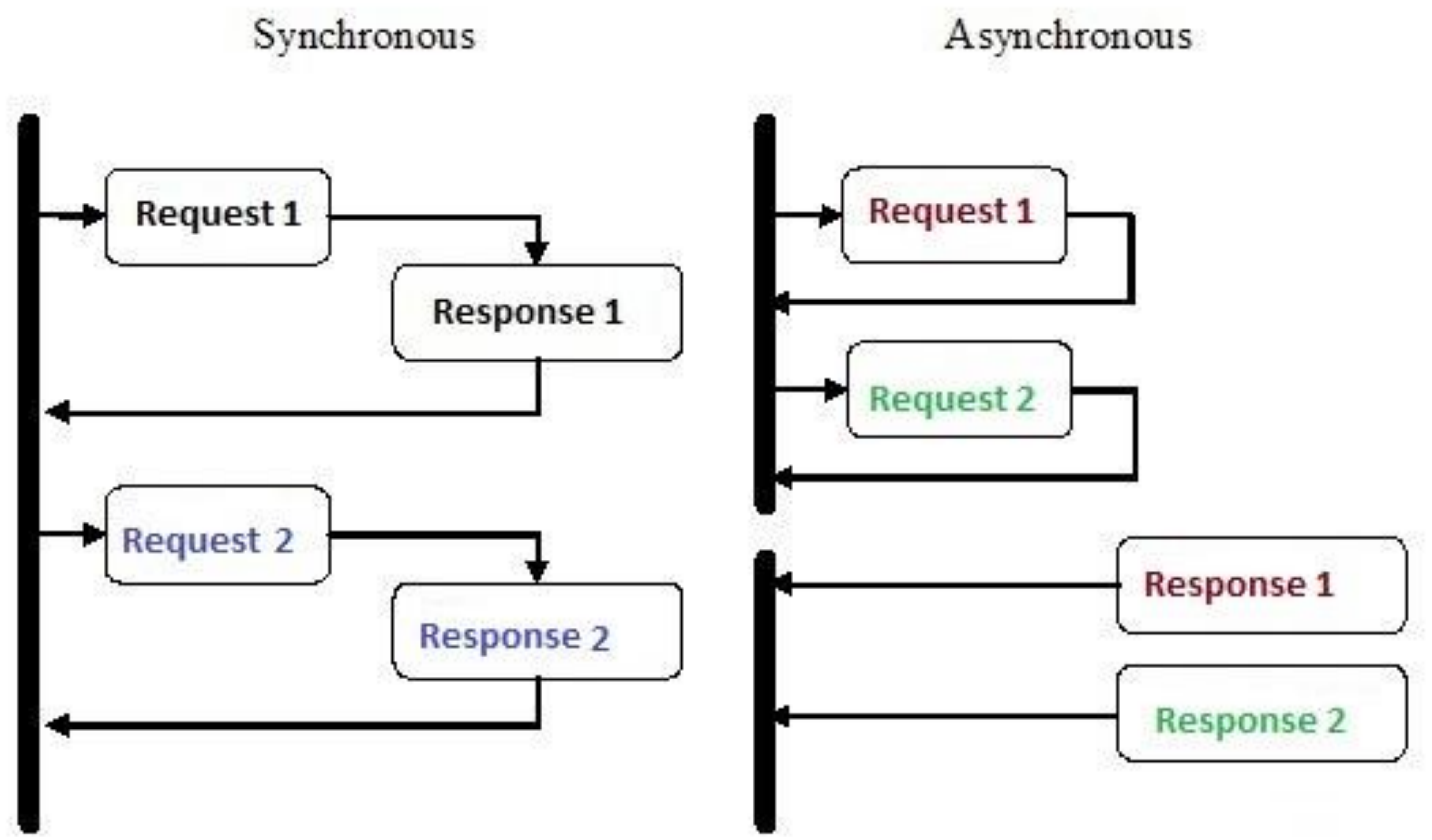


Image source

Asynchronous programming

- Explicitly handle the fact that the operations finished
- Trigger an operation and your code continues to run
  - Needs a mechanism of being informed about operation's completion, most common one is through callbacks
    - If your app/program heavily relies on asynchronous actions we recommend using a reactive programming library, such as rx (available for Java, Kotlin, Swift etc)
- What asynchronous actions have you encountered in the last two sessions?
- The asynchronous model is different than the multi-threaded model, but they are usually encountered together

Services in Android

- Main Android component, non-UI, used for background work
- Code is executed on the application's main thread (UI)
  - Can be in a different process but in the Android world multi-process communication inside an app is not encouraged (yet)
  - IntentServices can run on a different thread and perform a task
- Can "live" even when the activity is destroyed
- Let's think of logic from the previous sessions that could be put in a service
- Do not use services for CPU intensive or blocking operations because they run on the UI thread.

Services in Android

- All Services inherit the SDK's Service class
- The system instantiates your services, not you!
  - The component that starts it doesn't have an instance to that service
- All classes that extend Context inherit the *startService* method
  - Call startService/stopService from your activities
- See code exemples and flows for services on the wiki

Bound Services

- A mechanism to "connect" various components with services
  - in this case an activity can obtain a reference to the service
- The lifecycle of such a service is influenced by the number of components bound to it
  - no component bound -> the service will die (calling also startService makes it a StartedService and prevents that)
- Let's look at how the binding is done in the thingylib library for the ThingyService

Android BLE Communication - Prerequisites

1. Add permissions for Bluetooth and for Location
    a. See in the first session why and how
2. Explicitly check the Location at runtime, otherwise you cannot receive any results from scanning
3. Check that Bluetooth and the BLE feature is supported and that Bluetooth is enabled
    a. See the code in the first session solution

Android BLE Communication - Scanning

- Asynchronous model: start a scan and receive the results later
  - In the case of classic bluetooth, the scan stops after ~1 minute, in case of BLE you need to explicitly stop it
- Use the API methods for starting a scan (example)
  - Provide a **callback** to receive the results. It must implement the interface *ScanCallback*
  - Optionally include filters
- Will receive the results for each device in range that is advertising

Android BLE - BluetoothGatt profile

- The API for the Gatt Profile allowing your phone to transfer data to/from the peripheral
  - Discover services
  - Read characteristics
  - Read descriptors
- Constants for describing the connection state

Android BLE Communication - Connectivity

- Connect to the Gatt server of the peripheral received as scan result using BluetoothDevice's connectGatt method
  - It returns a BluetoothGatt object
- Implement a BluetoothGattCallback to receive updates from the device for changes on the BluetoothGatt profile
  - In the thingylib it is implemented in ThingyConnection

This session's App

- Create a basic communication prototype for the ThingyBoard
- Features and flows:
  - Scan for Bluetooth Low Energy devices
    - Display the scan results in a RecyclerView list with clickable rows
  - Click on a scan result that represents your Thingy
    - Start a new blank activity
    - Connect to the Thingy and listen for changes on a sensor/element of your choice, e.g. that the button was pressed
  - Display a toast when you receive updates from Thingy
- Use the code skeleton for the UI and scanning
  - It is based on the app created during the first session

Let's get started!