CoreBluetooth in practice

Ride on!

Christian Menschel
CocoaHeads AC August 2021

Bluetooth LE

Bluetooth Low Energy

Bluetooth LE

- Introduced 2009 as option with Bluetooth 4.0
- Low power consumption 0.01 – 0.50 W
- One battery for several months or years
- Managed by Bluetooth SIG (Special Interest Group)

Bluetooth LE

- Mesh feature
- Range up to 10 meter
- Max 2 Mbit (since Bluetooth 5.0)
- Awesome accuracy (cm) (since Bluetooth 5.1)
- 128-bit AES, user defined application layer

https://www.bluetooth.com/de/learn-about-bluetooth/bluetooth-technology/topology-options/

Bluetooth LE

AirTag

AirPod

iBeacon

Tile

CORONA
WARN-APP

Gemeinsam Corona bekämpfen

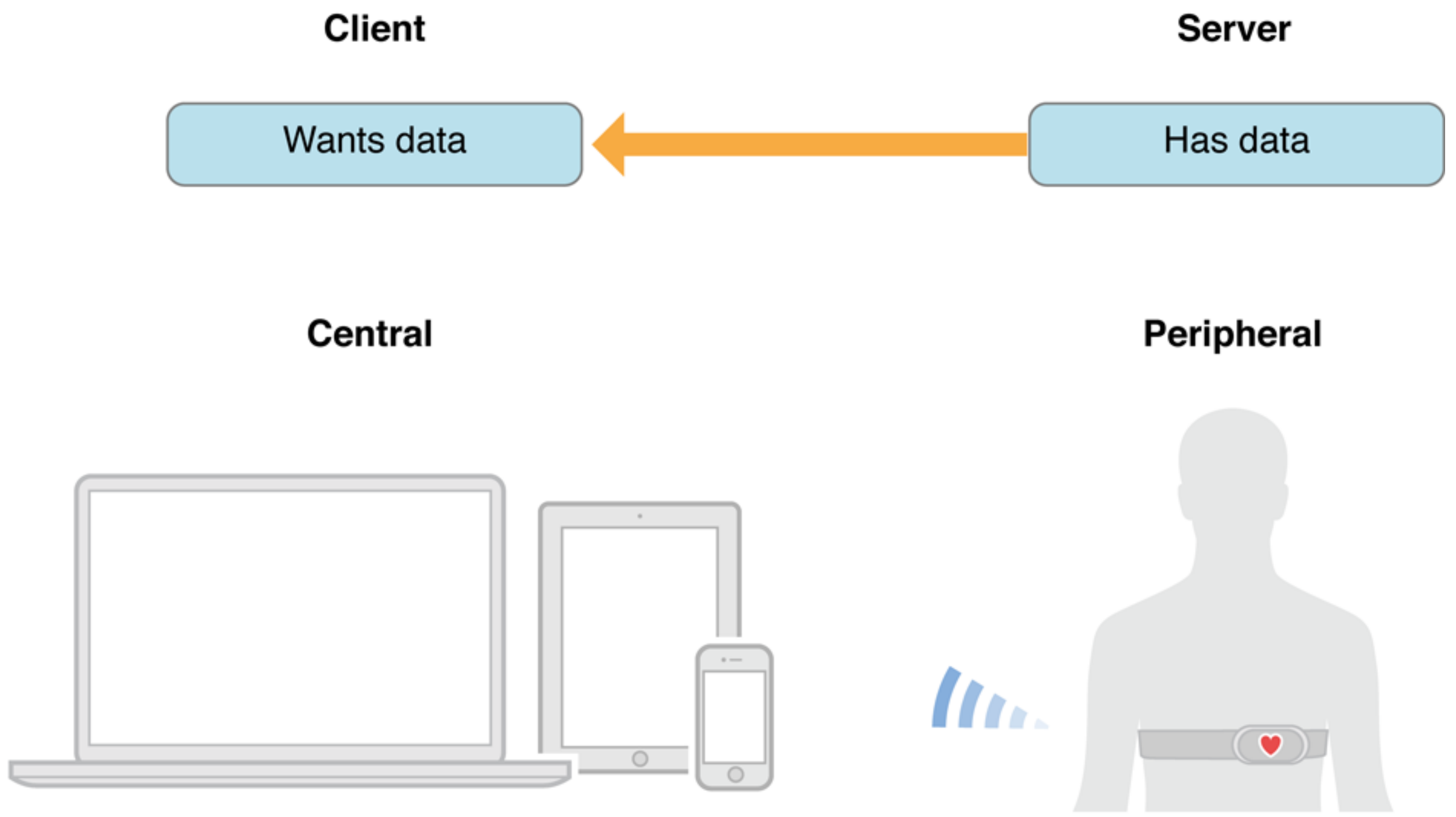
Mehr Schutz für Sie und uns alle. Mit der Corona-Warn-App durchbrechen wir Infektionsketten schneller.

Machen Sie Ihr Smartphone zum Corona-Warn-System. Überblicken Sie Ihren Risiko-Status und erfahren Sie, ob in den letzten 14 Tagen infizierte Personen in Ihrer Nähe waren.

Die App merkt sich Begegnungen zwischen Menschen, indem ihre Smartphones verschlüsselte Zufallscodes austauschen. Und zwar ohne dabei auf persönliche Daten

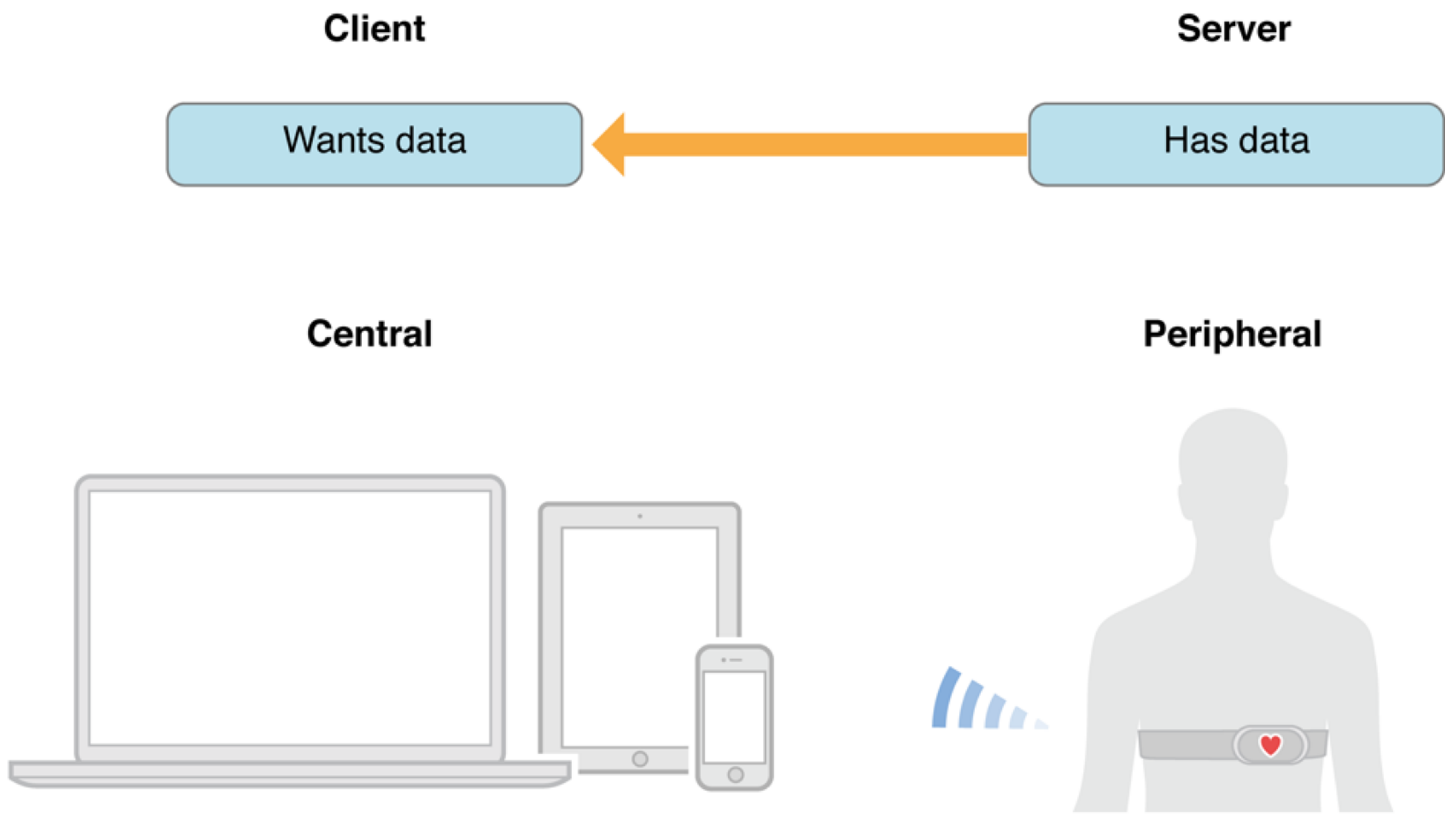
Nuki Smart Lock

Bluetooth LE



https://developer.apple.com/library/archive/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth_concepts/CoreBluetoothOverview/CoreBluetoothOverview.html#//apple_ref/doc/uid/TP40013257-CH2-SW1

Bluetooth LE



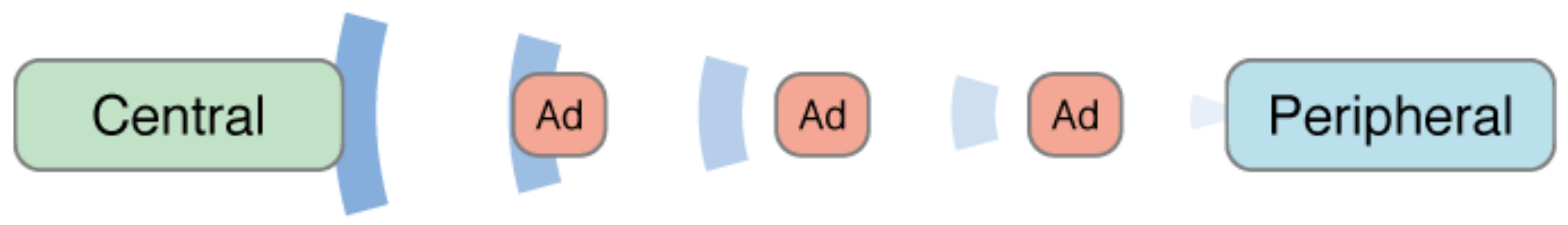
GATT Client

GATT Server

https://developer.apple.com/library/archive/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth_concepts/CoreBluetoothOverview/CoreBluetoothOverview.html#//apple_ref/doc/uid/TP40013257-CH2-SW1

Bluetooth LE

Advertising & Scanning



- Detection through a procedure based on broadcasting advertising packets
- 3 separate channels (frequencies) in order to reduce interference
- Scanner listens for a duration called the scan window

Bluetooth LE

Advertising Packet Format

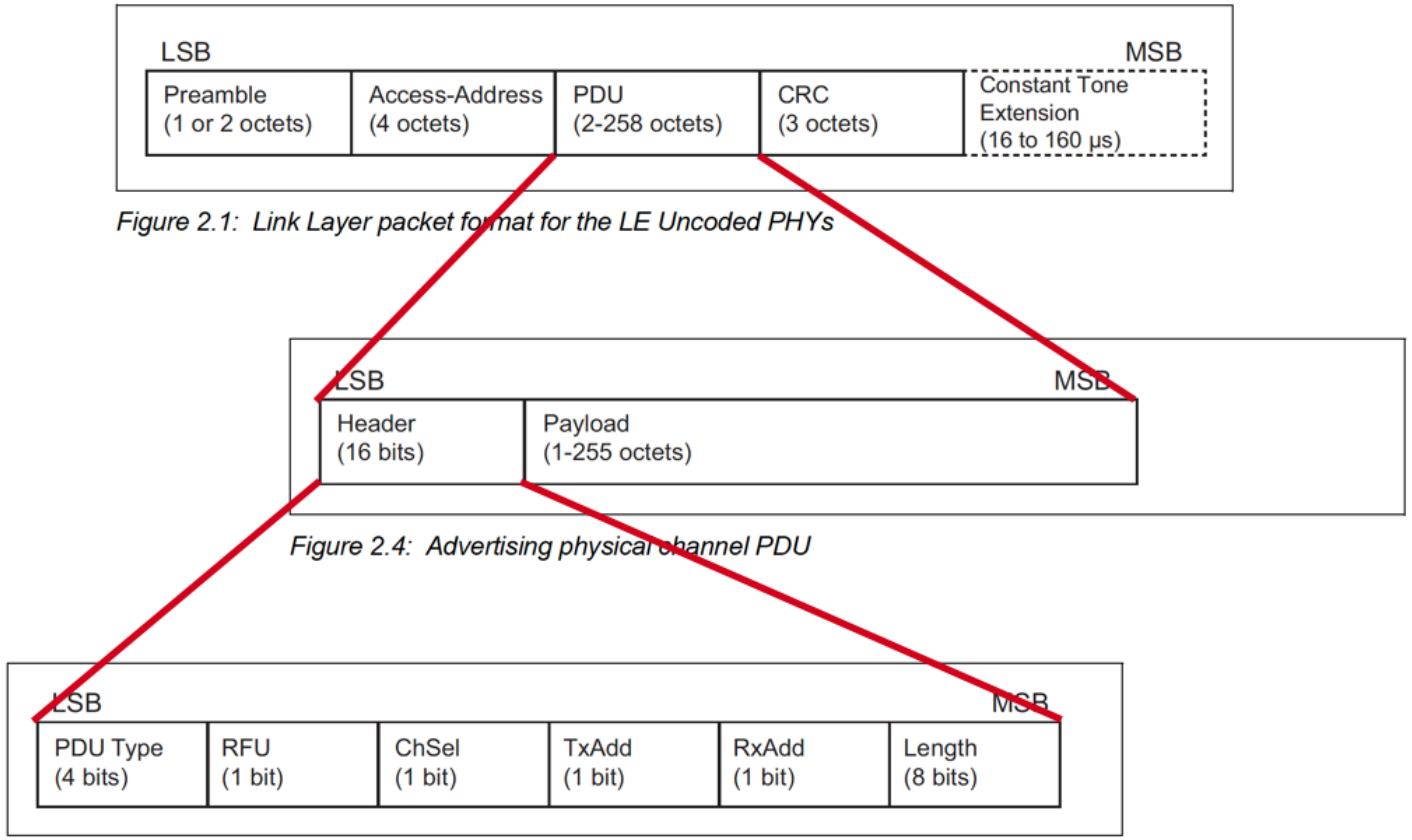


Figure 2.1: Link Layer packet format for the LE Uncoded PHYs

Figure 2.4: Advertising physical channel PDU

Figure 2.5: Advertising physical channel PDU header

https://www.novelbits.io/bluetooth-low-energy-advertisements-part-1/

- Header (16 bits)
  - Information whether an advertising device allows a connection
- Payload (variable size)
  - Service UUIDs
  - Device name
  - Manufacturer Specific Data
  - Company Identifier
  - Transmit power level
  - Advertising Interval
  - …. (many more)

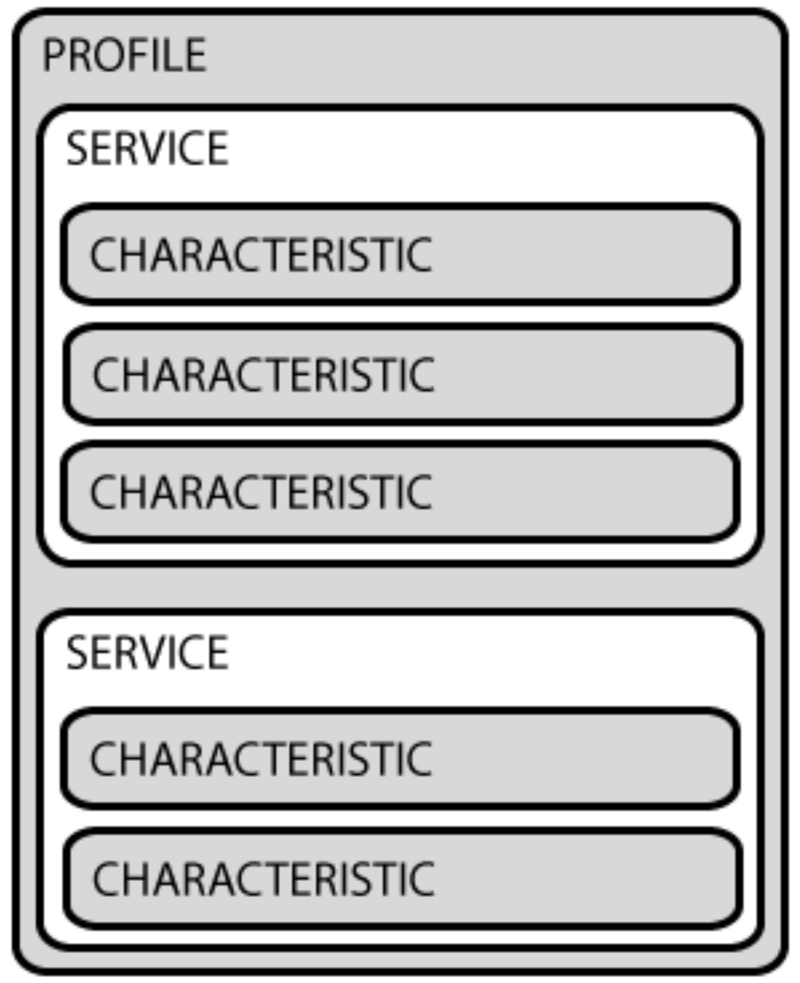
https://www.bluetooth.com/specifications/assigned-numbers/company-identifiers/

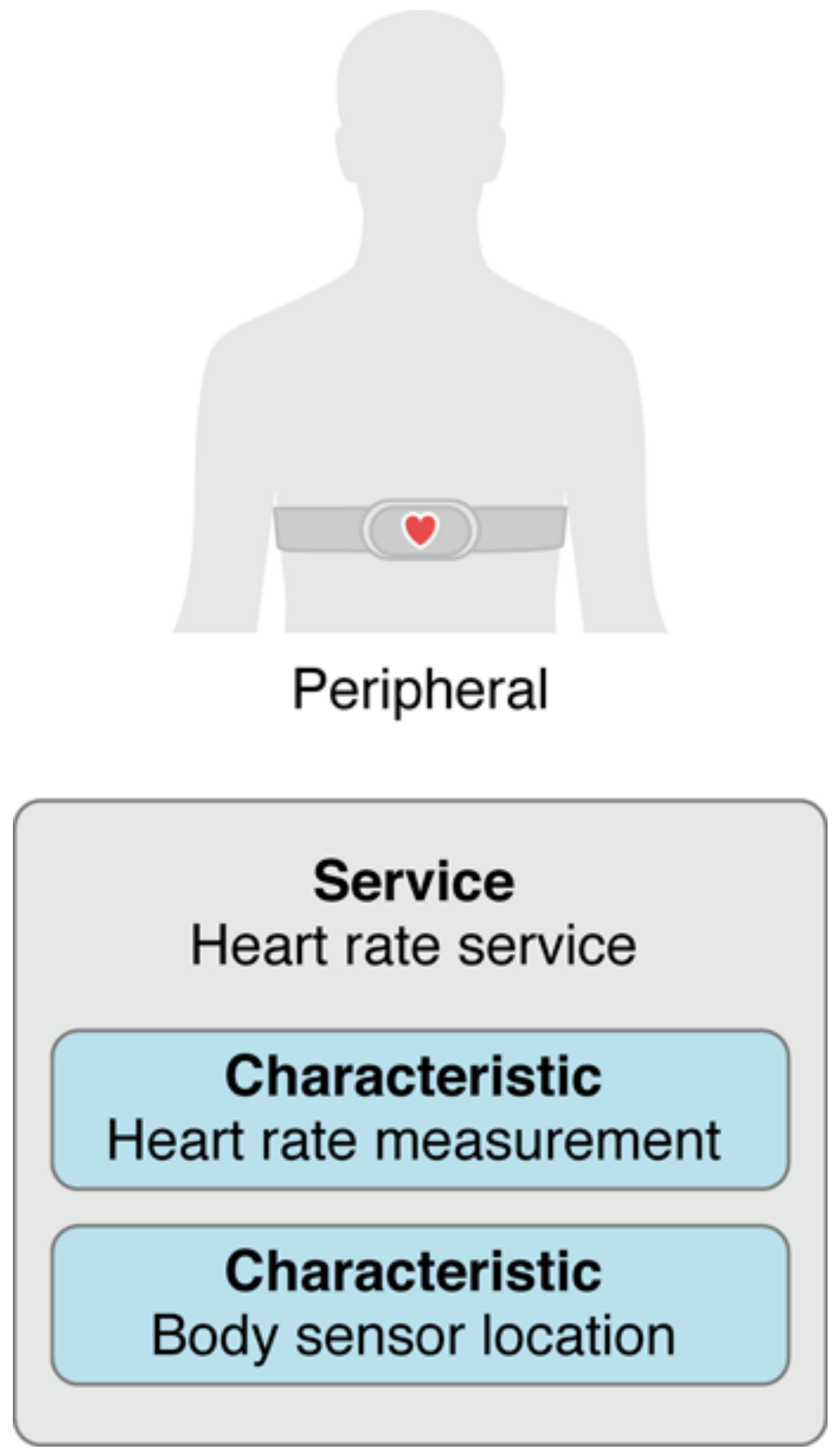
Bluetooth LE

GATT specification

➡ Services

➡ Characteristics





GATT => Generic Attribute Profile,
Defines the way that two Bluetooth Low Energy devices transfer data using concepts called **Services** and **Characteristics**.

- Each Peripheral can connect to one central only at a time (stops advertising after connection)
- Each Service and Characteristic has one UUID (e.g. Heartrate: 0x180D (Service), 0x2A37 (Characteristic)

➡ Standard to connect and transfer data by Bluetooth LE SIG

https://www.bluetooth.com/specifications/in-development/

Image: https://developer.apple.com/library/archive/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth_concepts/CoreBluetoothOverview/CoreBluetoothOverview.html#//apple_ref/doc/uid/TP40013257-CH2-SW1

Bluetooth LE

All GATT specifications

https://www.bluetooth.com/specifications/specs/

Location and Navigation
Internet Protocol Support
Emergency
Cycling Speed and Cadence
Coordinated Set Identification
Reconnection Configuration
Scan Parameters
Time
Media Control
Serial Port
Device Identification
Fitness Machine
Volume Control
Generic A/V Distribution
Synchronization
Object Push
Environmental Sensing
Pulse Oximeter
Cycling Power
Mesh
Advanced Audio Distribution
Object Transfer
Generic PIM
Proximity
SIM Access
Alert Notification
Heart Rate
A/V Remote Control
File Transfer
Blood Pressure
Insulin Delivery
Message Access
Global Navigation Satellite System
Asset Tracking
Automation IO
Basic Imaging
Find Me
Multi Profile
Microphone Control
Device Time
3D Synchronization
Running Speed and Cadence
Physical Activity Monitor
Phone Alert Status
Phone Book Access
Hands-Free
Hardcopy Cable Replacement
Glucose
Weight Scale
Binary Sensor
Continuous Glucose Monitoring
BR/EDR Connection Handover
Call Control
Human Interface Device
Calendar Tasks and Notes
Personal Area Networking
Generic Object Exchange
HID over GATT
Headset
Health Device
Health Thermometer
Dial-Up Networking

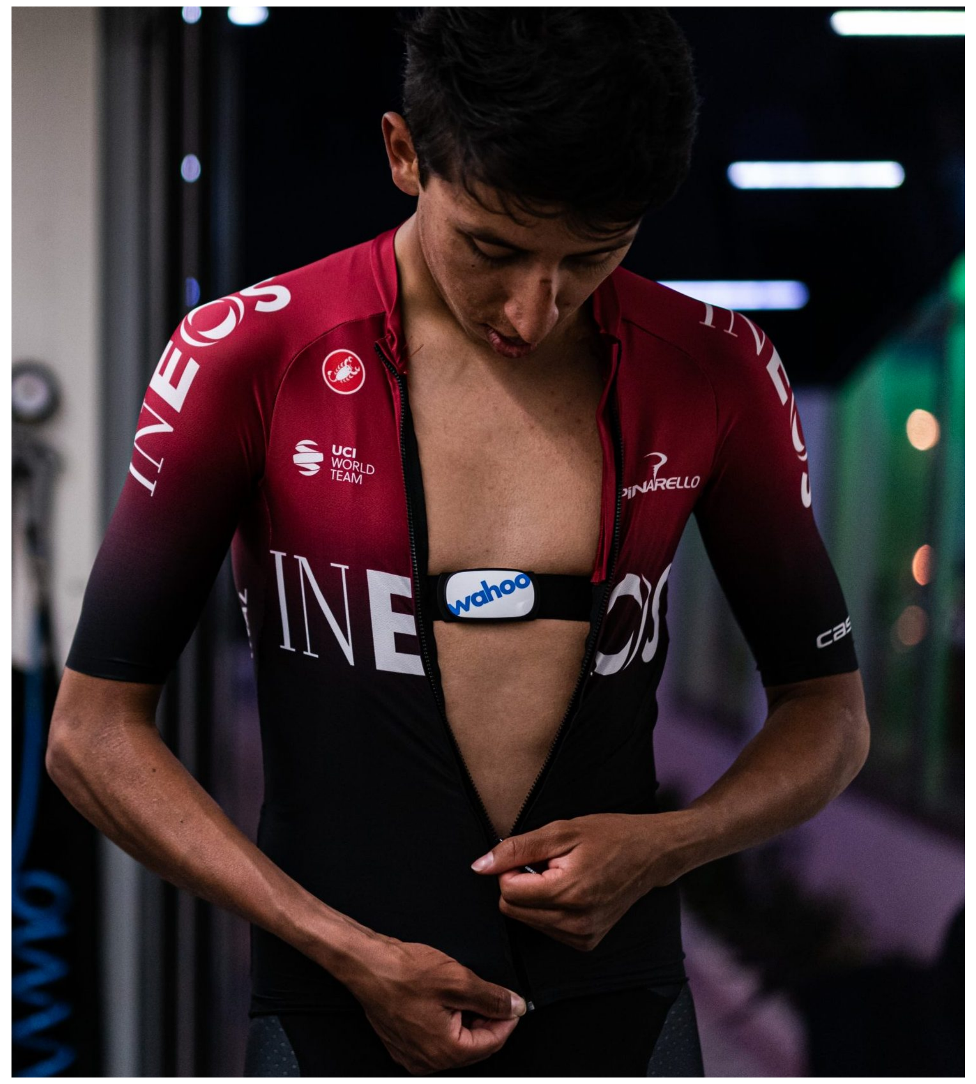
Bluetooth LE

All GATT specifications

https://www.bluetooth.com/specifications/specs/

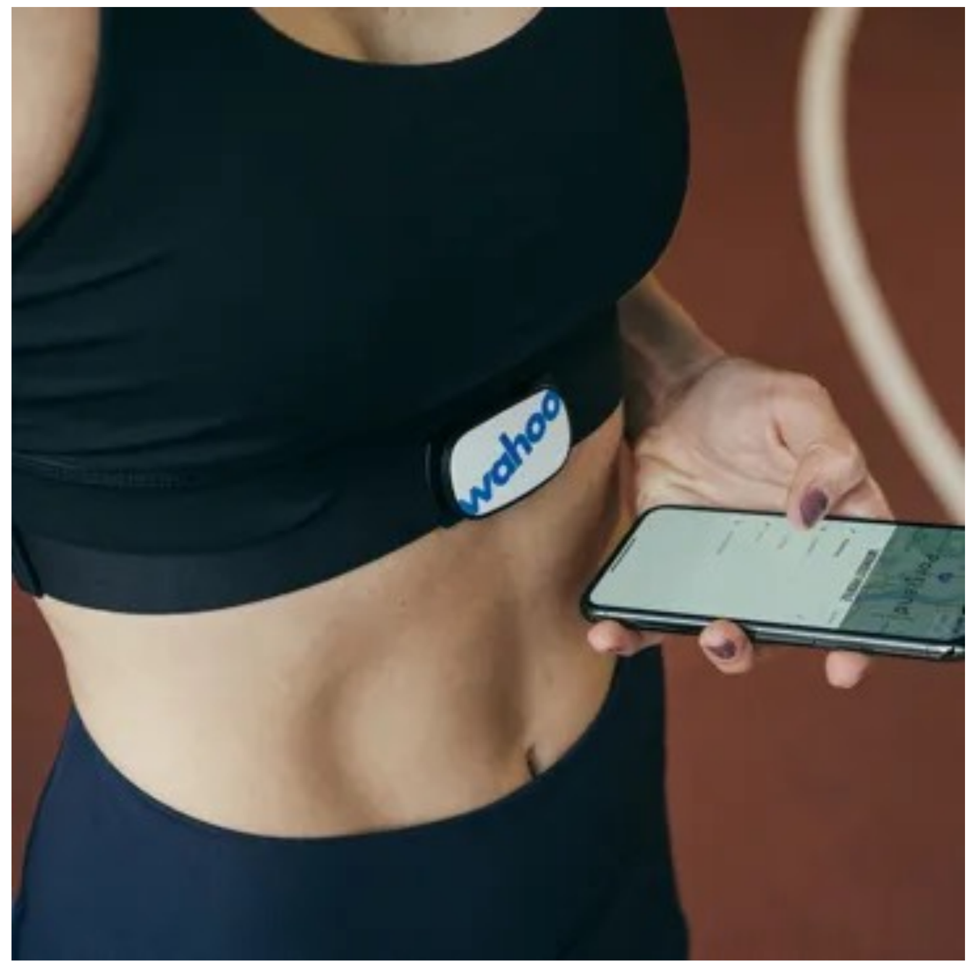
Location and Navigation
Internet Protocol Support
Emergency
Cycling Speed and Cadence
Coordinated Set Identification
Reconnection Configuration
Scan Parameters
Time
Media Control
Serial Port
Device Identification
Fitness Machine
Volume Control
Generic A/V Distribution
Synchronization
Object Push
Environmental Sensing
Pulse Oximeter
Cycling Power
Mesh
Advanced Audio Distribution
Object Transfer
Generic PIM
Proximity
SIM Access
Alert Notification
Heart Rate
A/V Remote Control
File Transfer
Blood Pressure
Insulin Delivery
Message Access
Global Navigation Satellite System
Asset Tracking
Automation IO
Basic Imaging
Multi Profile
Find Me
Microphone Control
Device Time
3D Synchronization
Running Speed and Cadence
Physical Activity Monitor
Phone Alert Status
Phone Book Access
Hands-Free
Hardcopy Cable Replacement
Glucose
Weight Scale
Binary Sensor
Continuous Glucose Monitoring
BR/EDR Connection Handover
Call Control
Human Interface Device
Calendar Tasks and Notes
Personal Area Networking
Generic Object Exchange
HID over GATT
Headset
Health Device
Health Thermometer
Dial-Up Networking

Bluetooth LE



Cycling Power

Cycling Speed and Cadence



Heart Rate

Bluetooth LE

Profiles (GATT)

- Just a collection of Services
- Compiled by either the Bluetooth SIG (Special Interest Group) or by the peripheral designers
- The Heart Rate Profile, for example, combines the Heart Rate Service and the Device Information Service.
- See more -> Profiles Overview.

Bluetooth LE

Profiles (GATT)

- Just a collection of Services
- Compiled by either the Bluetooth SIG (Special Interest Group) or by the peripheral designers
- The Heart Rate Profile, for example, combines the Heart Rate Service and the Device Information Service.
- See more -> Profiles Overview.

Services

- Uses UUID for identification, which can be either 16-bit (for officially adopted BLE Services) or 128-bit (for custom services).
- Heart Rate Service has a 16-bit UUID of 0x180D, and contains up to 3 **characteristic**:

  Heart Rate Measurement, Body Sensor Location and Heart Rate Control Point.

Bluetooth LE

Profiles (GATT)

- Just a collection of Services
- Compiled by either the Bluetooth SIG (Special Interest Group) or by the peripheral designers
- The Heart Rate Profile, for example, combines the Heart Rate Service and the Device Information Service.
- See more -> Profiles Overview.

Services

- Uses UUID for identification, which can be either 16-bit (for officially adopted BLE Services) or 128-bit (for custom services).
- Heart Rate Service has a 16-bit UUID of 0x180D, and contains up to 3 **characteristic**:

  Heart Rate Measurement, Body Sensor Location and Heart Rate Control Point.

Characteristics

- The lowest level concept in GATT transactions to encapsulate a single data point (like Heart Rate Measurement)
- Also uses 16-bit or 128-bit UUID like 0x2A37 for Heart Rate Measurement

Bluetooth LE

Profiles (GATT)

- Just a collection of Services
- Compiled by either the Bluetooth SIG (Special Interest Group) or by the peripheral designers
- The Heart Rate Profile, for example, combines the Heart Rate Service and the Device Information Service.
- See more -> Profiles Overview.

Services

- Uses UUID for identification, which can be either 16-bit (for officially adopted BLE Services) or 128-bit (for custom services).
- Heart Rate Service has a 16-bit UUID of 0x180D, and contains up to 3 **characteristic**:

  Heart Rate Measurement, Body Sensor Location and Heart Rate Control Point.

Characteristics

- The lowest level concept in GATT transactions to encapsulate a single data point (like Heart Rate Measurement)
- Also uses 16-bit or 128-bit UUID like 0x2A37 for Heart Rate Measurement

Descriptor

- A descriptor provides additional information about a characteristic
- For instance, a temperature value characteristic may have an indication of its units (e.g. Celsius)

Bluetooth LE

UUIDs

- UUIDs are unique 128-bit (16 byte) numbers
  - like: **75BEB663-74FC-4871-9737-AD184157450E**

- To avoid transmitting 16 bytes for Service & Characteristics UUIDs

  Bluetooth SIG has adopted a UUID base like

  XXXXXXXX-0000-1000-8000-00805F9B34FB

- The 32 bits (X) are variable and can be used by the manufacturer
- The remaining 96 bits are defined by the Bluetooth SIG

- Heartrate would be: 0000180D-0000-1000-8000-00805F9B34FB

  short: 0x180D

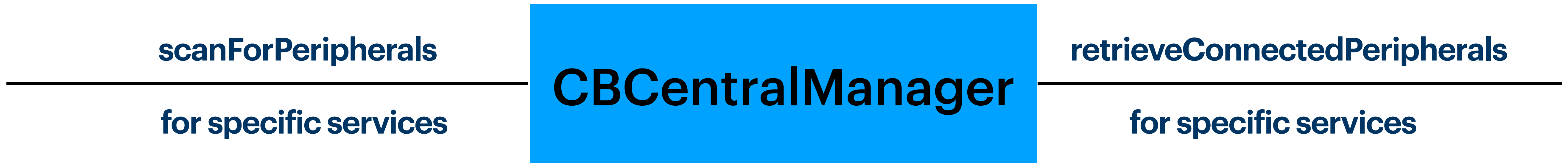
Bluetooth LE

CoreBluetooth

Bluetooth LE

CoreBluetooth

Discover

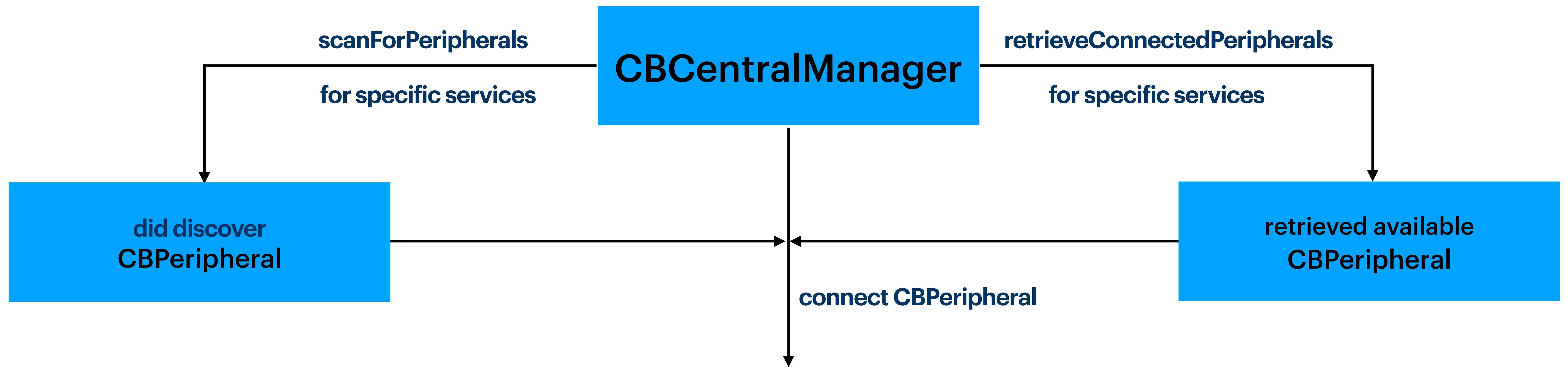


https://developer.apple.com/documentation/corebluetooth

Bluetooth LE

CoreBluetooth

Discover

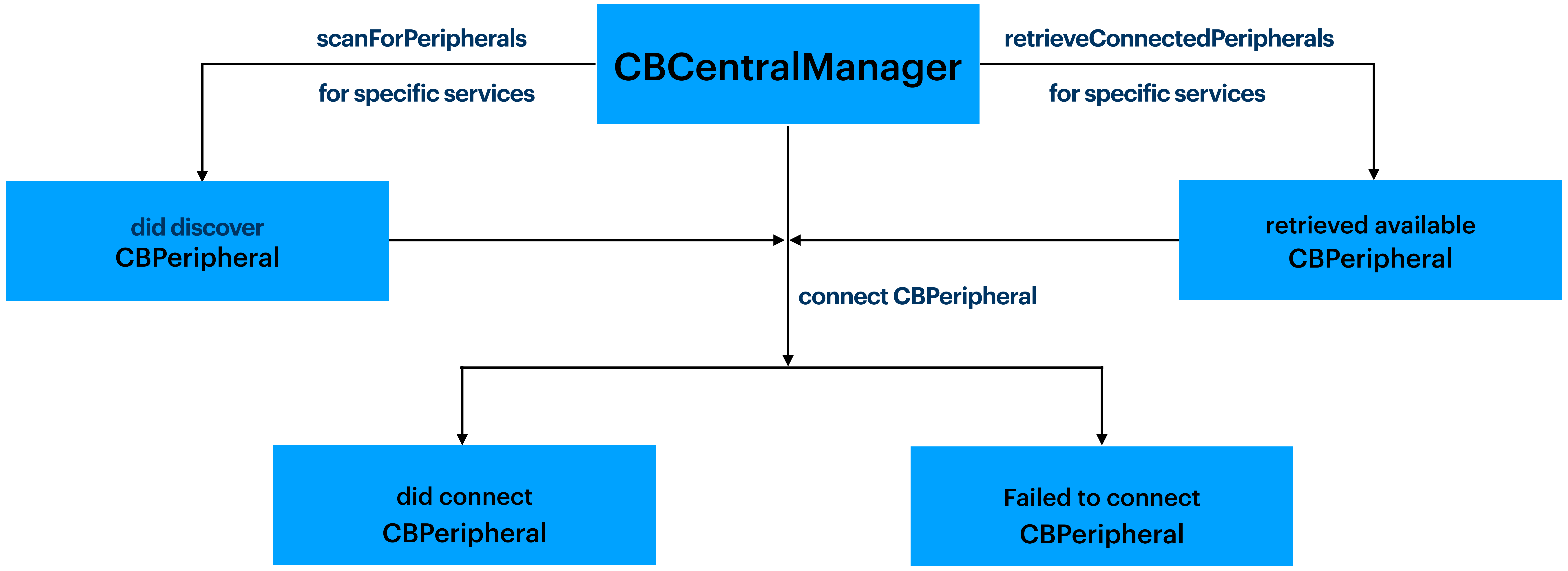


https://developer.apple.com/documentation/corebluetooth

Bluetooth LE

CoreBluetooth

Discover

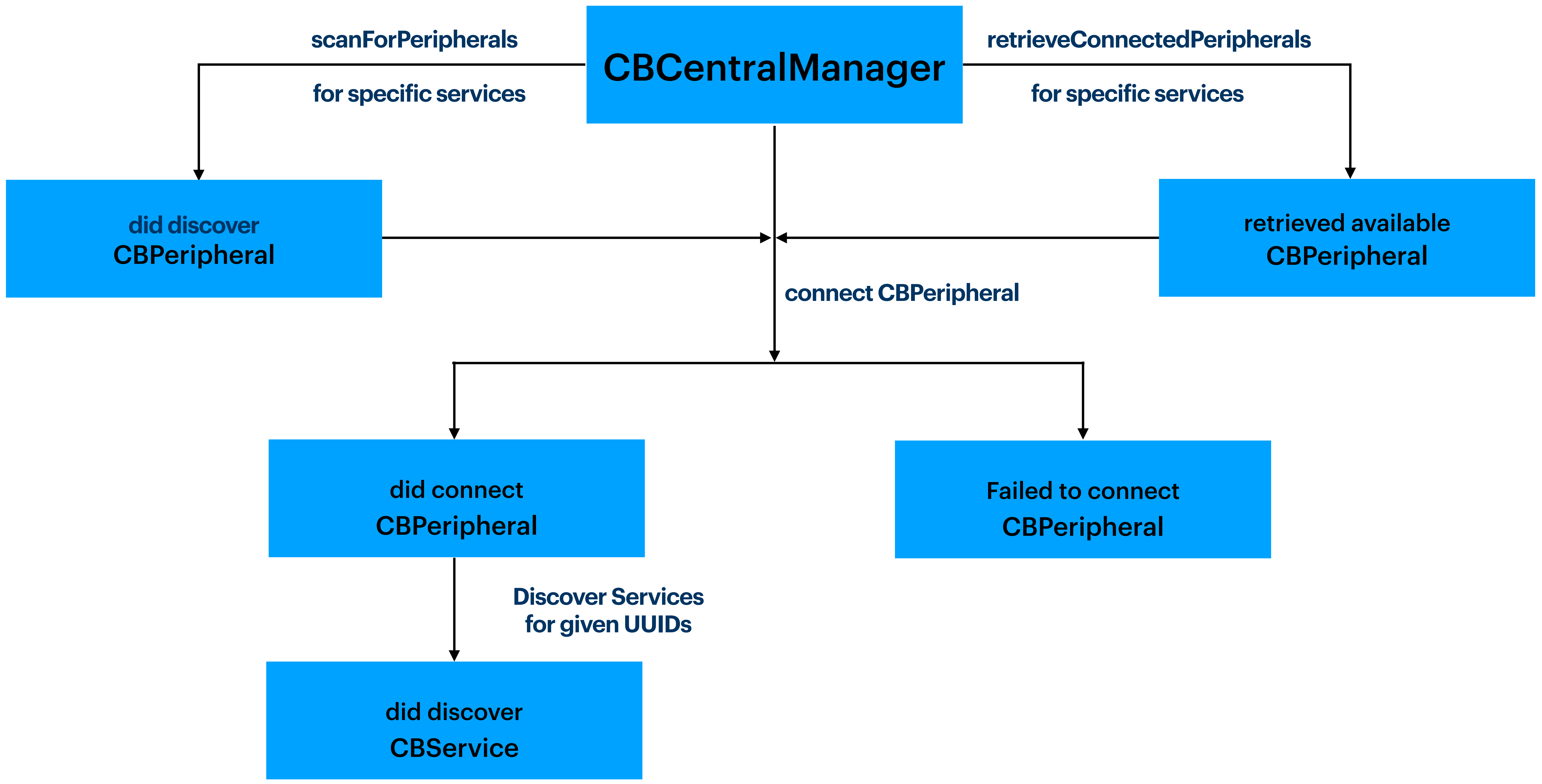


https://developer.apple.com/documentation/corebluetooth

Bluetooth LE

CoreBluetooth

Discover



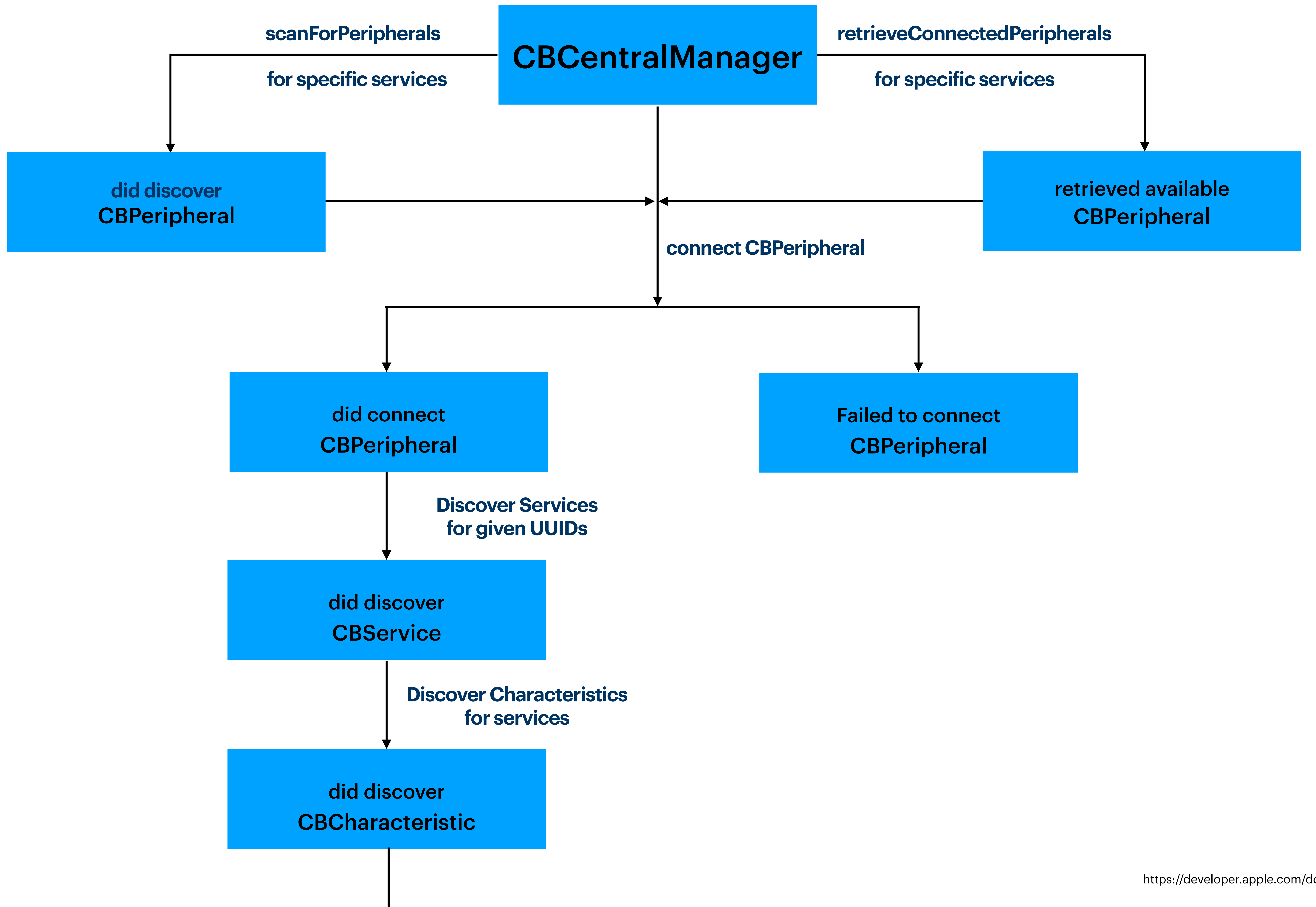
CBCentralManager
scanForPeripherals
for specific services
retrieveConnectedPeripherals
for specific services
did discover CBPeripheral
retrieved available CBPeripheral
connect CBPeripheral
did connect CBPeripheral
Failed to connect CBPeripheral
Discover Services for given UUIDs
did discover CBService

https://developer.apple.com/documentation/corebluetooth

Bluetooth LE

CoreBluetooth

Discover



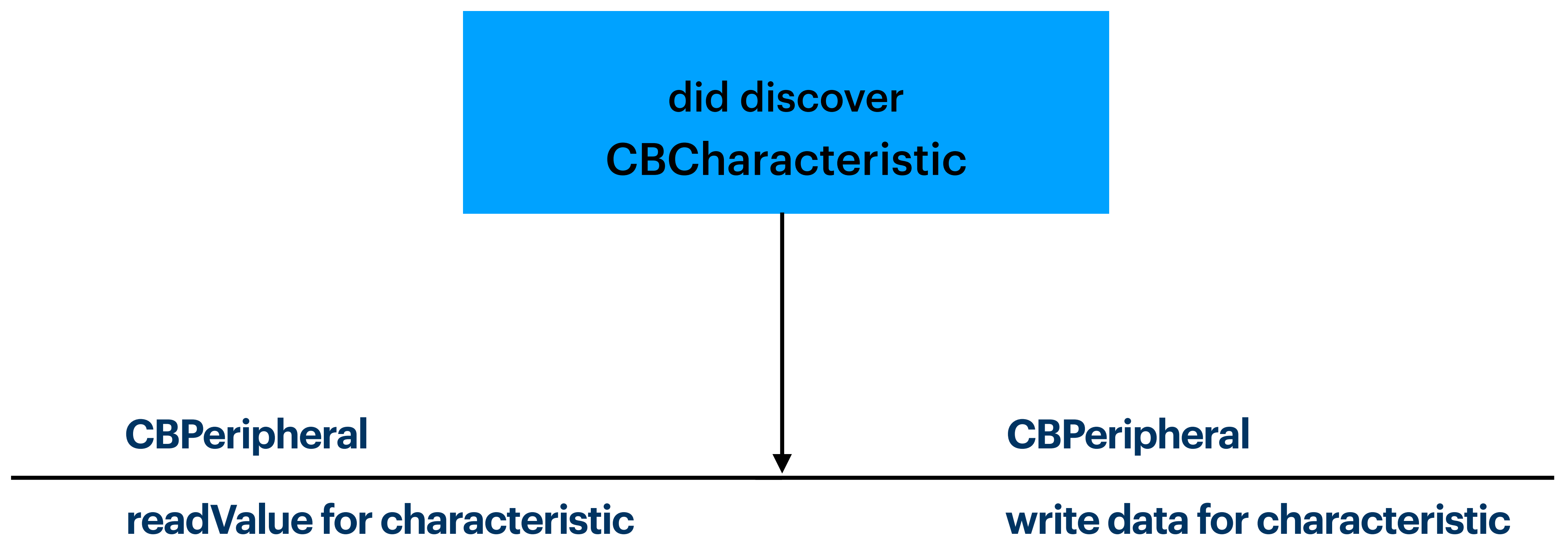
CBCentralManager
scanForPeripherals
for specific services
retrieveConnectedPeripherals
for specific services
did discover CBPeripheral
retrieved available CBPeripheral
connect CBPeripheral
did connect CBPeripheral
Failed to connect CBPeripheral
Discover Services for given UUIDs
did discover CBService
Discover Characteristics for services
did discover CBCharacteristic

https://developer.apple.com/documentation/corebluetooth

Bluetooth LE

CoreBluetooth

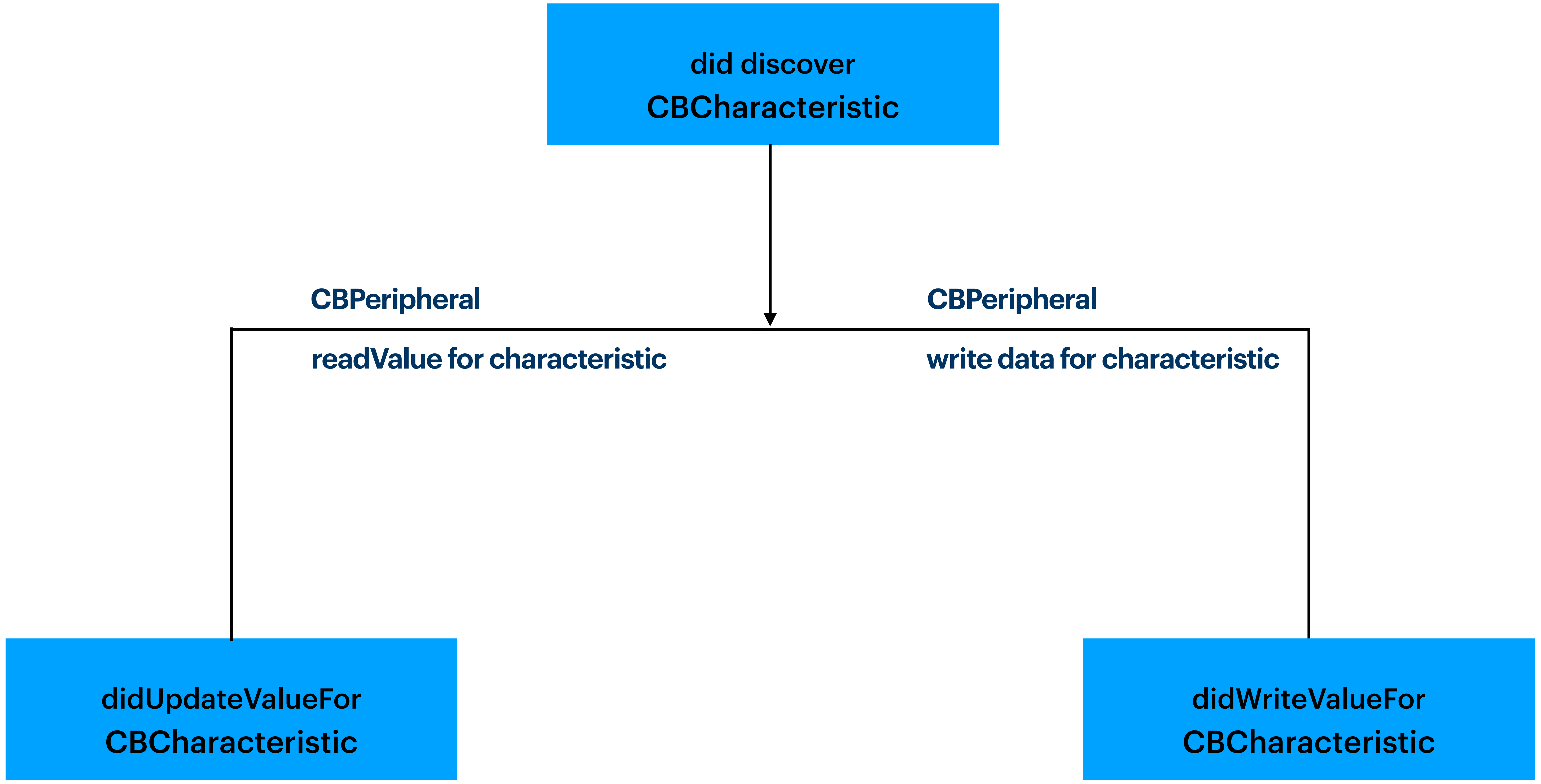
Read/Write



Bluetooth LE

CoreBluetooth

Read/Write

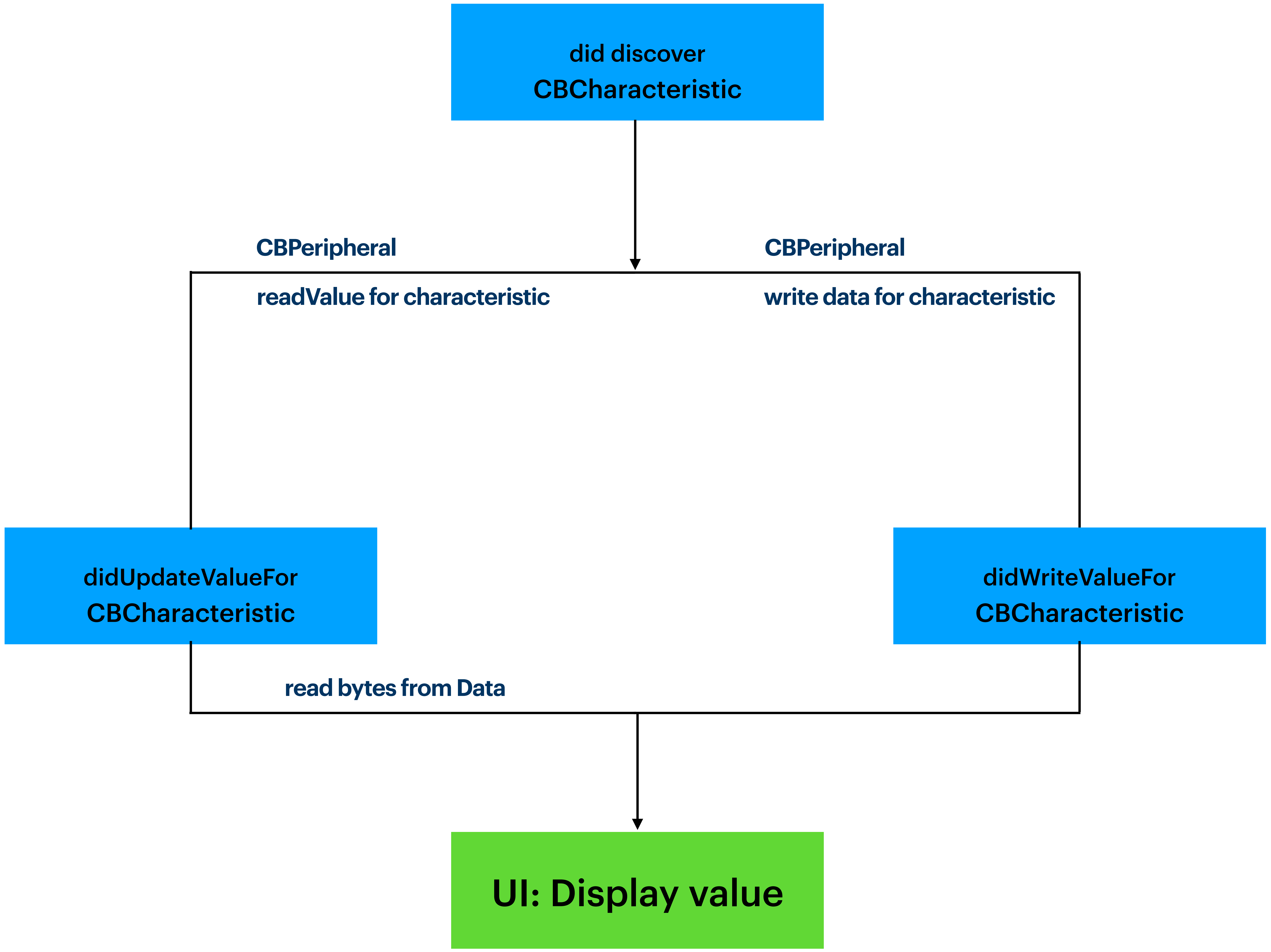

did discover CBCharacteristic
CBPeripheral
readValue for characteristic
CBPeripheral
write data for characteristic
didUpdateValueFor CBCharacteristic
didWriteValueFor CBCharacteristic

https://developer.apple.com/documentation/corebluetooth

Bluetooth LE

CoreBluetooth

Read/Write


did discover CBCharacteristic
CBPeripheral
readValue for characteristic
CBPeripheral
write data for characteristic
didUpdateValueFor CBCharacteristic
didWriteValueFor CBCharacteristic
read bytes from Data
UI: Display value

https://developer.apple.com/documentation/corebluetooth

Bluetooth LE

CoreBluetooth: Scan and connect

```
func scan() {
    guard !centralManager.isScanning, centralManager.state == .poweredOn else { return }
    centralManager.scanForPeripherals(withServices: BluetoothDataType.allServiceUUIDs)
}
```

Bluetooth LE

CoreBluetooth: Scan and connect

```
func scan() {
    guard !centralManager.isScanning, centralManager.state == .poweredOn else { return }
    centralManager.scanForPeripherals(withServices: BluetoothDataType.allServiceUUIDs)
}

func centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral, advertisementData: [String : Any], rssi RSSI: NSNu
    peripheral.delegate = self
    peripherals.append(peripheral)
    central.connect(peripheral)
}
```

Bluetooth LE

CoreBluetooth: Scan and connect

```
func scan() {
    guard !centralManager.isScanning, centralManager.state == .poweredOn else { return }
    centralManager.scanForPeripherals(withServices: BluetoothDataType.allServiceUUIDs)
}

func centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral, advertisementData: [String : Any], rssi RSSI: NSNu
    peripheral.delegate = self
    peripherals.append(peripheral)
    central.connect(peripheral)
}

func centralManager(_ central: CBCentralManager, didConnect peripheral: CBPeripheral) {
    peripheral.discoverServices(BluetoothDataType.allServiceUUIDs)
}
```

Bluetooth LE

CoreBluetooth: Scan and connect

```
func scan() {
    guard !centralManager.isScanning, centralManager.state == .poweredOn else { return }
    centralManager.scanForPeripherals(withServices: BluetoothDataType.allServiceUUIDs)
}

func centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral, advertisementData: [String : Any], rssi RSSI: NSNu
    peripheral.delegate = self
    peripherals.append(peripheral)
    central.connect(peripheral)
}

func centralManager(_ central: CBCentralManager, didConnect peripheral: CBPeripheral) {
    peripheral.discoverServices(BluetoothDataType.allServiceUUIDs)
}

func peripheral(_ peripheral: CBPeripheral, didDiscoverServices error: Error?) {
    guard let services = peripheral.services else { return }
    services.forEach {service in
        peripheral.discoverCharacteristics(nil, for: service)
    }
}
```

Bluetooth LE

CoreBluetooth: Scan and connect

```
func scan() {
    guard !centralManager.isScanning, centralManager.state == .poweredOn else { return }
    centralManager.scanForPeripherals(withServices: BluetoothDataType.allServiceUUIDs)
}

func centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral, advertisementData: [String : Any], rssi RSSI: NSNu
    peripheral.delegate = self
    peripherals.append(peripheral)
    central.connect(peripheral)
}

func centralManager(_ central: CBCentralManager, didConnect peripheral: CBPeripheral) {
    peripheral.discoverServices(BluetoothDataType.allServiceUUIDs)
}

func peripheral(_ peripheral: CBPeripheral, didDiscoverServices error: Error?) {
    guard let services = peripheral.services else { return }
    services.forEach {service in
        peripheral.discoverCharacteristics(nil, for: service)
    }
}

func peripheral(_ peripheral: CBPeripheral, didDiscoverCharacteristicsFor service: CBService, error: Error?) {
    guard let characteristics = service.characteristics else { return }
    characteristics.forEach {characteristic in
        if characteristic.properties.contains(.read) {
            peripheral.readValue(for: characteristic)
        }
        if characteristic.properties.contains(.notify) {
            peripheral.setNotifyValue(true, for: characteristic)
        }
    }
}
```

Bluetooth LE

CoreBluetooth: Read with Bitmasking and Bitshifting

```swift
func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic, error: Error?) {
    guard let data = characteristic.value else {
        return nil
    }
    let bytes = [UInt8](data)
    let values: Values
    let firstBitValue = bytes[0] & 0x01
    if firstBitValue == 0 {
        // Heart Rate Value Format is 8-bit value and in the 2nd byte
        values = Values(bpm: Int(bytes[1]))
    } else {
        // Heart Rate Value Format is 16-bit and in the 2nd and 3rd bytes
        values = Values(bpm: (Int(bytes[1]) << 8) + Int(bytes[2]))
    }
}
```

The heart rate measurement is in the 2nd, or in the 2nd and 3rd bytes, i.e. one one or in two bytes
The first byte of the first bit specifies the length of the heart rate data, 0 == 1 byte, 1 == 2 bytes

https://www.bluetooth.com/wp-content/uploads/Sitecore-Media-Library/Gatt/Xml/Characteristics/org.bluetooth.characteristic.heart_rate_measurement.xml

Bluetooth LE

CoreBluetooth: Read with NSData getBytes and NSRange

```
var crank: UInt16 = 0
var crankTime: UInt16 = 0
var location = 0
let length16Bit = MemoryLayout<UInt16>.size
(data as NSData).getBytes(&crank, range: NSRange(location: location, length: length16Bit))
location += length16Bit
(data as NSData).getBytes(&crankTime, range: NSRange(location: location, length: length16Bit))
location += length16Bit
```

Demo time