Hacking bluetooth devices with WHAD

INSA TOULOUSE LAAS CNRS Quarkslab

IeHACK 2026

Romain Cayre, Damien Cauquil

Who am I ?

Damien Cauquil, Quarkslab

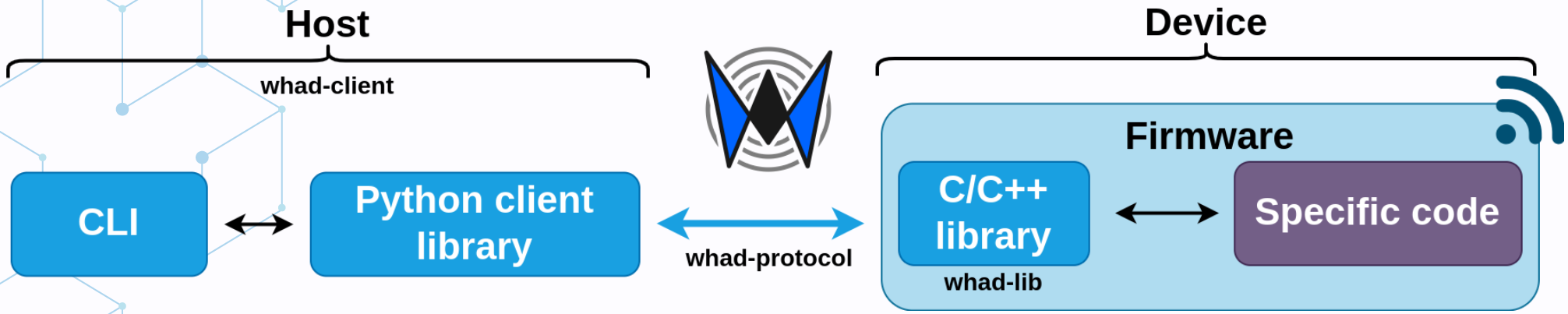
- author of *Btlejack*, a BLE swiss-army tool
- loves reversing stuff, including *embedded systems*

Agenda

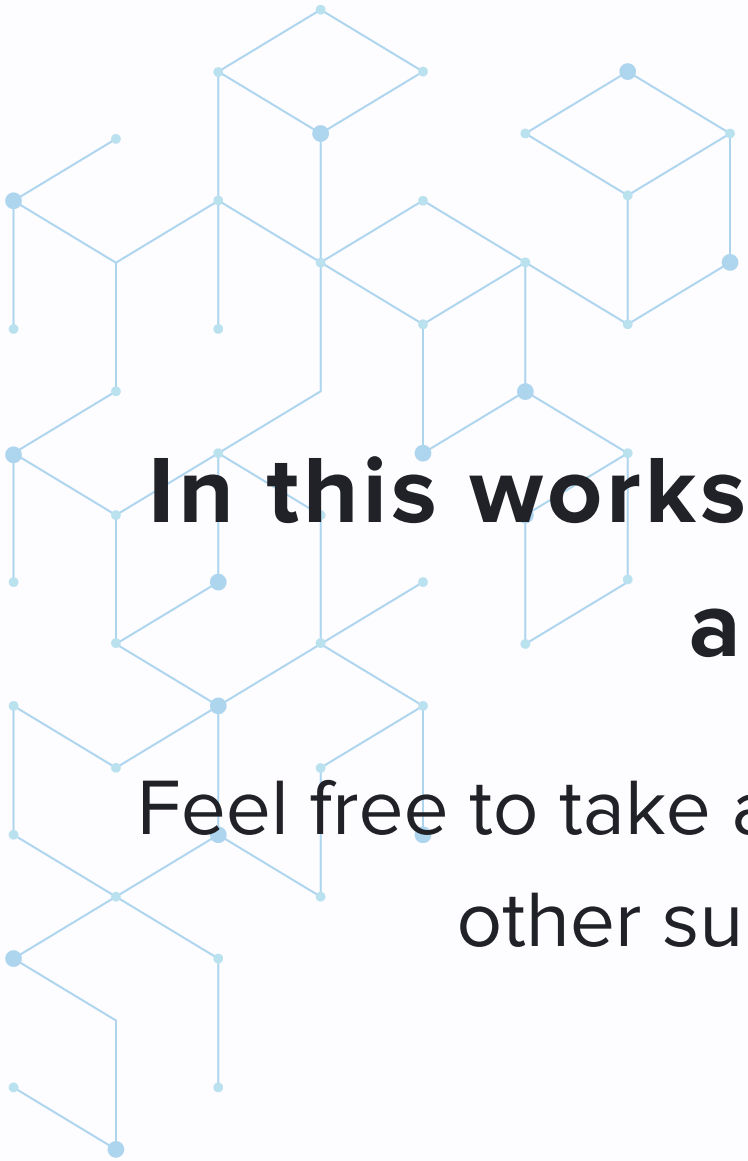
- Workshop setup
- Bluetooth Low Energy 101 *(TLDR version)*

- **Exercise #1: *Abusing GATT permissions***
- **Exercise #2: *Identifying a device's baseband***
- **Exercise #3: *Crashing a BLE tracker***

WHAD



- Harmonised Host / RF Hardware **communication protocol**
- **Python library** supporting multiple wireless protocols
- Multiple user-friendly **CLI tools** that can be combined
- Set of firmwares for **various hardware devices**



In this workshop, we will focus on BLE CLI tools and exploit development

Feel free to take a look on the corresponding Python API and the other supported protocols if you enjoy WHAD !

Hardware requirements

- Computer or VM running a **Linux** operating system
- No hardware required! (NEW)

Installing our new WHAD lab environment

Installing whad labs is as simple as running:

```
$ mkdir whad-workshop && cd whad-workshop
$ python3 -m venv venv
$ source venv/bin/activate
(venv) $ pip install whad-lab
```

Check your setup

`whadup` / `wup` is one of the CLI tools provided by WHAD, used to list the available interfaces and their capabilities.

Run the following command to detect supported devices:

```
$ wup
[i] Available devices
- hci0
  Type: Hci
  Index: 0
  Identifier: hci0
```

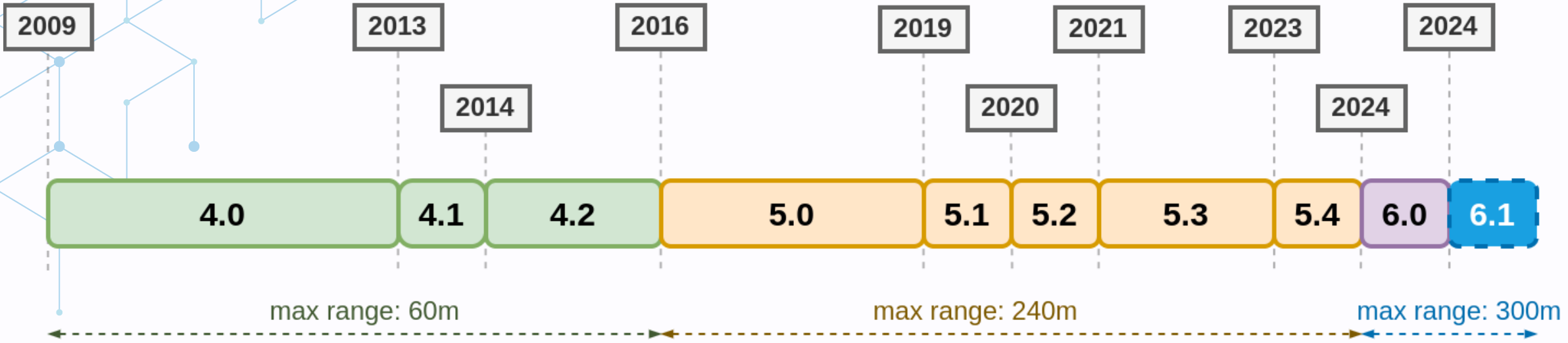
Bluetooth Low Energy 101

(TLDR version)

Bluetooth Low Energy

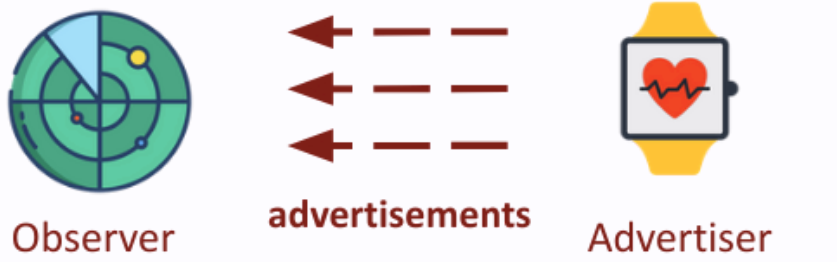
- Same frequency band as WiFi and ZigBee *(2.40 - 2.48 GHz)*
- Widely supported *(smartphones, cars, headphones, TVs, ...)*
- Pretty easy to understand *(but it's getting more and more complex)*
- Easy to use:
  - most laptop's motherboards embed a Bluetooth adapter
  - USB Bluetooth dongles can also be used

Versions vs. Time vs. range



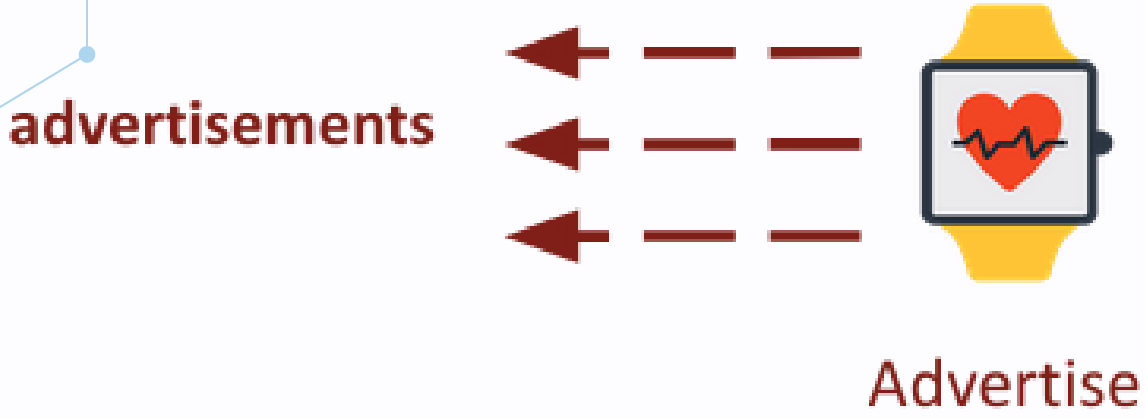
Device role(s)

- **Advertiser**: only advertise itself *(does not accept connections)*
- **Observer**: only search for devices *(does not initiate connection)*
- **Peripheral**: advertise itself and accept connections
- **Central**: search for devices and initiate connections



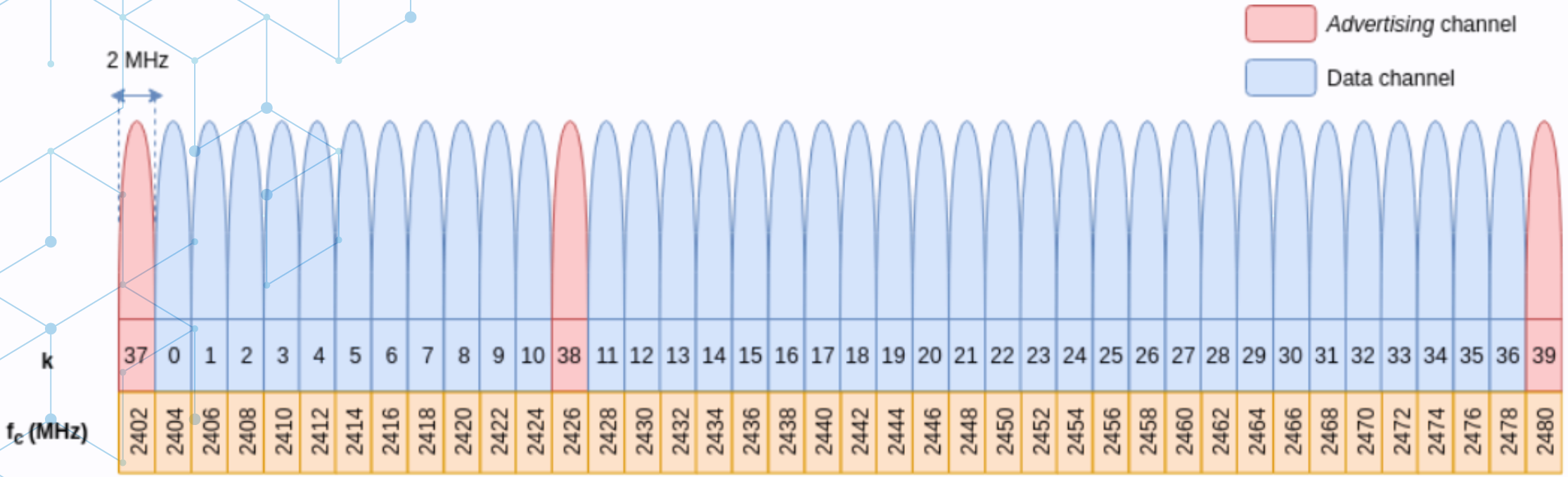
Some devices implement both the **Peripheral** and **Central** roles.

BLE device advertisements



- Two types of PDU sent by BLE devices:
  - **Advertisement PDUs**: *ADV_IND*, *ADV_DIRECT_IND*, *ADV_NONCONN_IND*, etc...
  - **Scan-related PDUs** (active mode): *SCAN_REQ* and *SCAN_RSP*

BLE channels

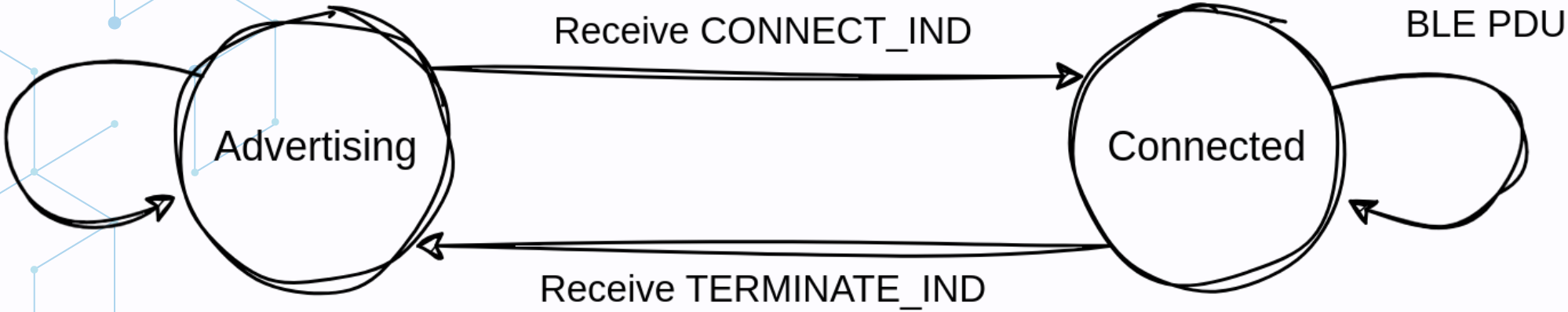


- Advertisement and Scan PDUs are sent (at least) on **advertising channels (37, 38, 39)**

Connected mode

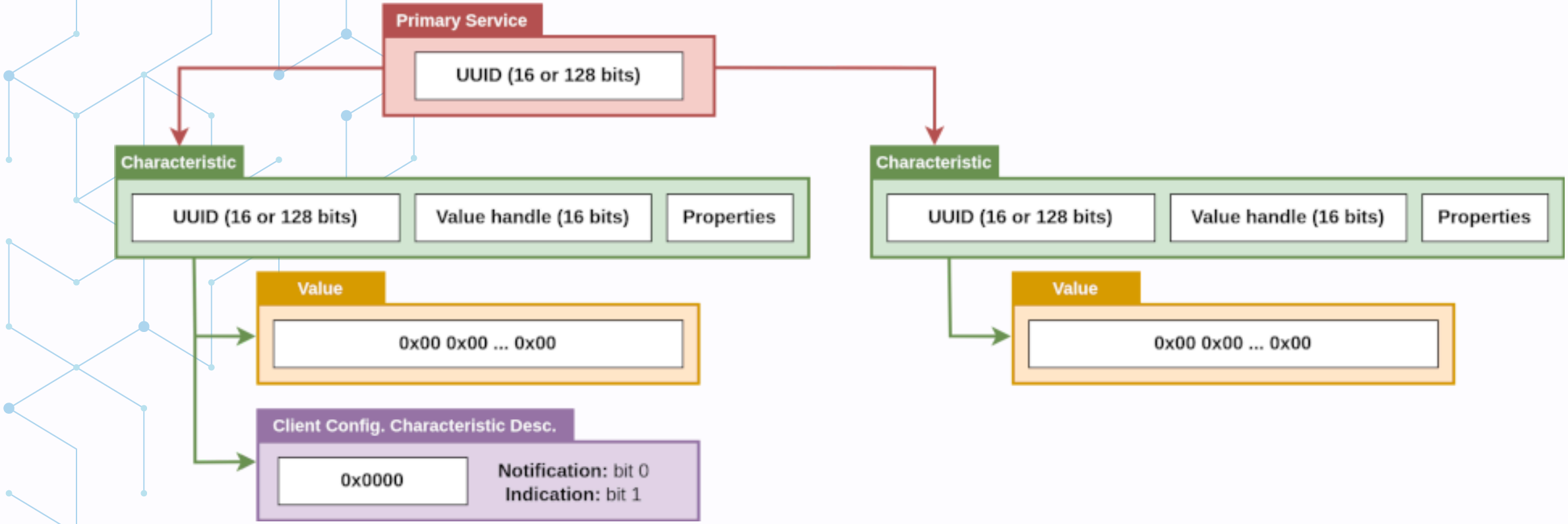
- BLE devices can establish a **point to point** connection based on a **Central / Peripheral topology**
- Connection relies on a **channel frequency algorithm** to prevent interferences with WiFi / other connections (hard to sniff !)

Connectable device state machine

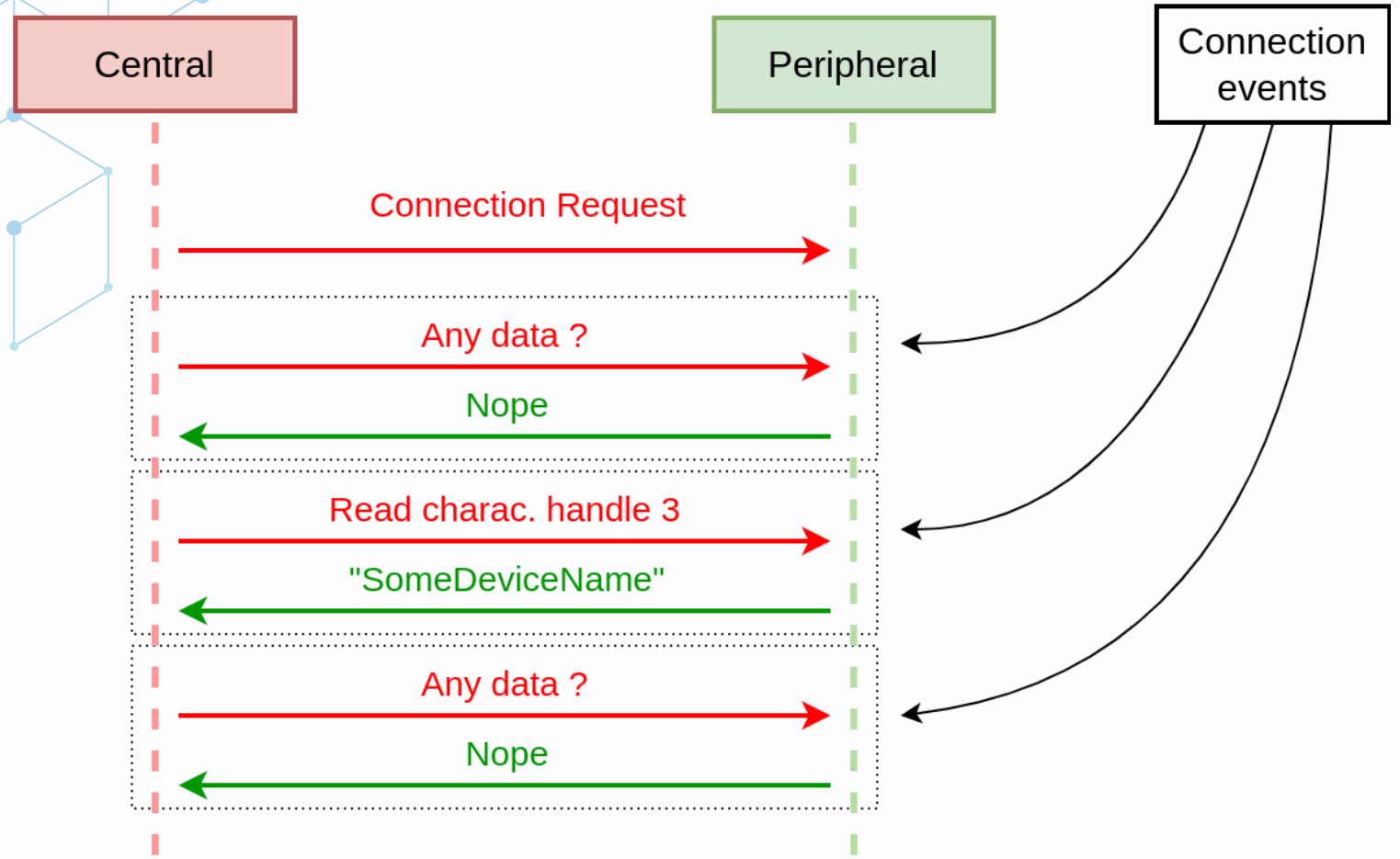


Most devices only accept a single connection at a time.

GATT hierarchy

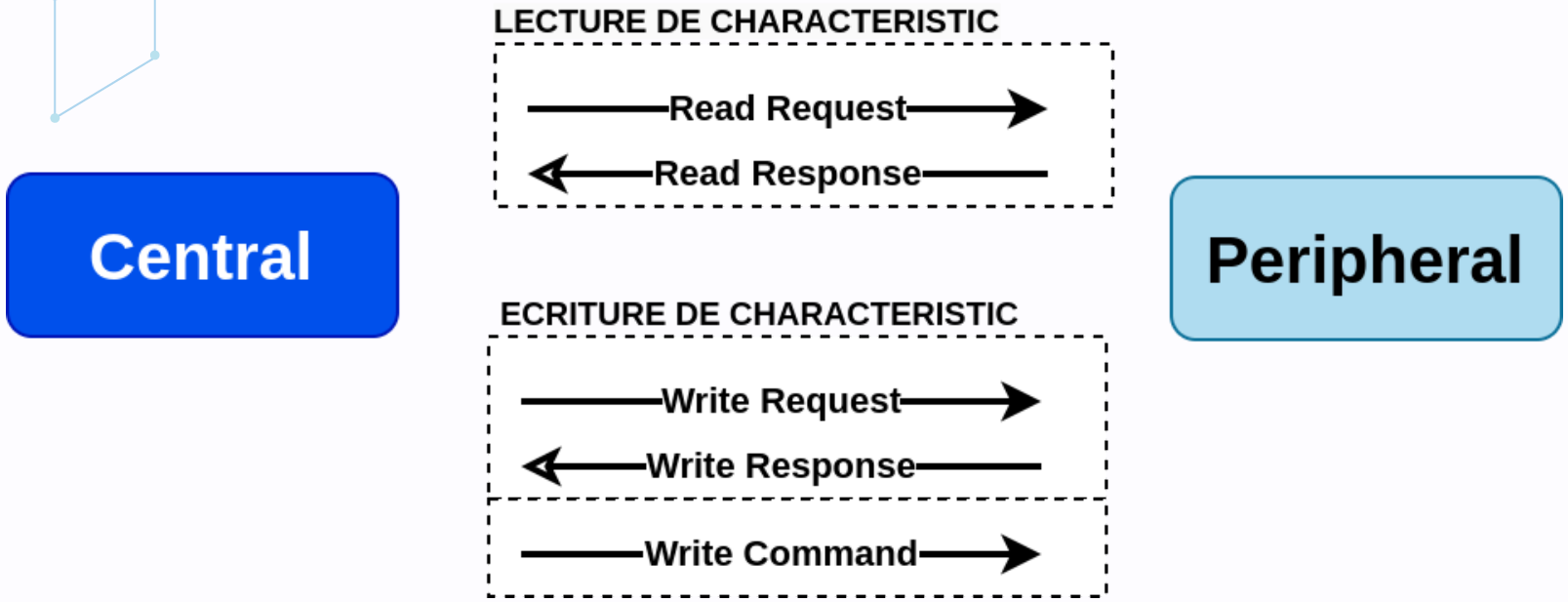


Central drives communication



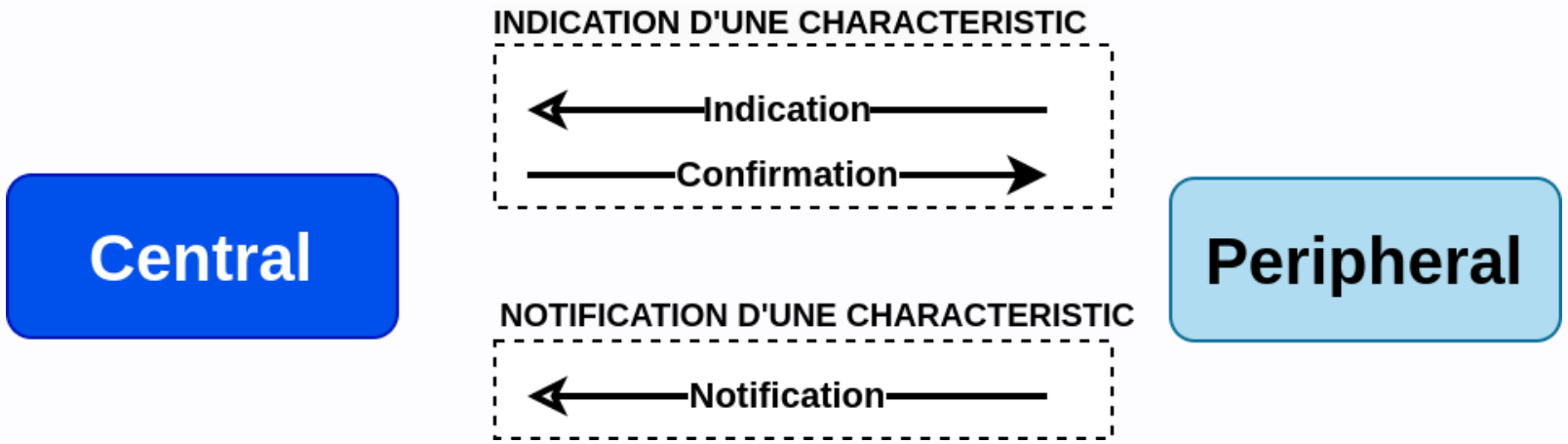
Write and Read Request & Commands

- **Central** acts as a GATT client / **Peripheral** acts as a GATT server
- Most of the time, **Central** initiates the data exchange with **Peripheral** with a set of requests & commands:



Notifications / Indications

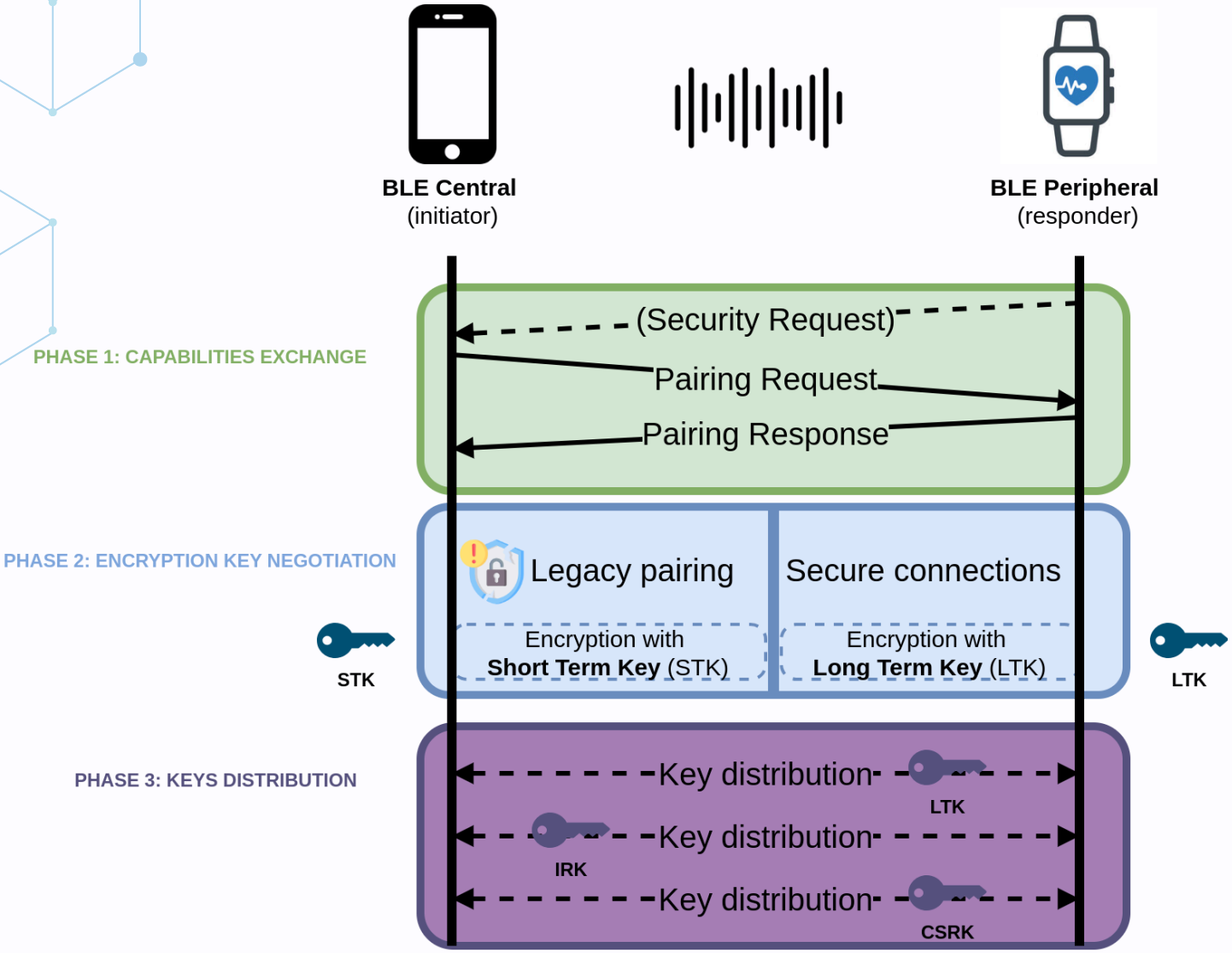
- **Peripheral** can notify to the **Central** that an attribute has changed using *Notification* and *Indication*.
- First, the **Central** subscribe to notification (or indication) for a given characteristic by writing into a descriptor.
- Then, at any time, **Peripheral** can transmit either a **Notification** (no response) or an **Indication** to notify a change:



BLE pairing vs bonding

- Bluetooth Low Energy **pairing** allows to negotiate security keys (e.g., encryption) to encrypt and authenticate the link
- Bluetooth Low Energy **bonding** is a variant of BLE pairing where the devices will store the distributed keys for later use

BLE pairing overview



BLE CLI tools

wble-central

- Implements a BLE central device
- Offers an interactive shell

```
$ wble-central -i <INTERFACE>
wble-central>
```

Scanning devices

```
wble-central> scan
 RSSI Lvl  Type  BD Address          Extra info
[-082 dBm] [RND] 41:d9:56:2d:8b:cc
[-084 dBm] [RND] 69:9f:0d:53:a6:e5
[-070 dBm] [RND] 5d:af:e7:0f:39:3c
[-075 dBm] [PUB] 88:0e:85:f6:76:72
```

Connecting to a device

```
wble-central> connect 88:0e:85:f6:76:72
Successfully connected to 88:0e:85:f6:76:72
wble-central|88:0e:85:f6:76:72>
```

- `wble-central` automatically detects if the device uses a random or public address
- Use <TAB> to quickly pick the correct bluetooth device address

Enumerating services and characteristics

`wble-central` provides a command to enumerate a GATT server's primary services and their related characteristics, `profile`.

```
wble-central|88:0e:85:f6:76:72> profile
```

When lost, ask for help!

- `wble-central` provides a command to ask for help: `help`
- If you're somehow lost or don't know what to do, raise your hand and we'll come see you

Starting the lab environment

Start WHAD's lab environment by running this command with your virtual environment active:

```
$ wlab
```

Then open a browser and connect to http://127.0.0.1:8080/.

Lab #1: abusing GATT permissions

Lab #1

- ATT attributes have their own permissions (Read/Write)
- GATT characteristics have *properties* (Read/Write/Notify/Indicate)
- Some stacks let developers handle stack-related tasks, what could go wrong?

Lab #1

- Start the lab "GATT Permissions"
- Follow the steps and exploit a vulnerable characteristic

Lab #2: LL_VERSION_IND

Lab #2

- The Bluetooth specification provides a feature to query a remote device about its controller vendor name and firmware version
- The LL_ VERSION_IND PDU must be sent before any other PDU (*theoritically*)

Lab #2

- Start the lab "BLE Baseband Identification"
- Follow the steps and identify the remote SoC and firmware version

Note: **Wireshark needs to be installed on your machine**

Lab #3: crashing a device BLE stack

Lab #3

- Some BLE stacks do not correctly handle GATT attributes and this may sometimes lead to a crash (HardFault), putting the device's firmware into
  an endless loop causing a denial of service
- Some unexpected PDUs may also this kind of behavior, or at least cause a reset of the device.

Lab #3

- Start the lab "BLE tracker Denial of Service attack"
- Follow the steps and render your BLE tracker irresponsive!
- Bonus: write an exploit in Python (using WHAD's Python API)

Thank you !