BLE Basic & BLE Mesh

Island, 2018.08.24

BLE Overview

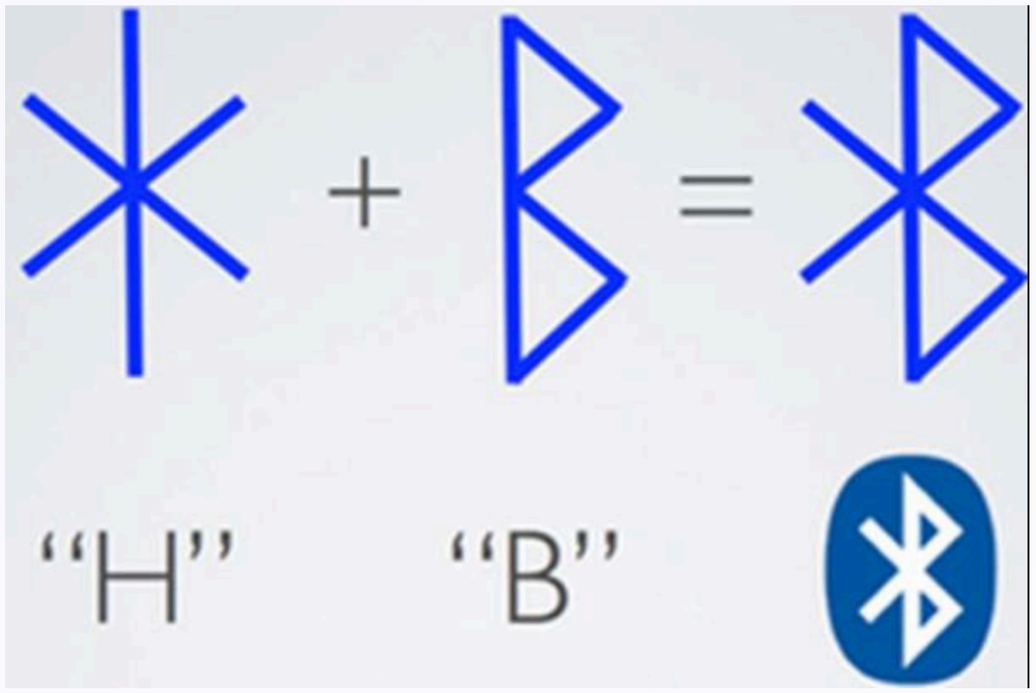
What is Bluetooth?

SIG

- Swedish company Ericsson (1994)
- Special Interest Group (1999): Ericsson, Nokia, Intel, IBM, Toshiba

Harald Blåtand

- King of Denmark, 10th century
- Uniform, Inovation



History Of Bluetooth?

1999.07 V1.0

V1.0: 748~810 Kbps
V1.2: AFH, eSCO, HCI

2004.07 V2.0

Enhanced Data Rate (EDR)
2.1 Mbps

2007.07 V2.1

Secure Simple Pairing (SSP)
Extended Inquiry Response
(EIR)
Sniff Subrating

2009.04 V3.0

24 Mbps
Alternative MAC/PHY (AMP)

History Of BLE?

2010.07 V4.0

First version of BLE
Ultra low power
Ultra low cost
Fast connection
GATT/ATT

Wi-Fi 802.11a (1999)
ZigBee (2004)

2013.12 V4.1

Low Duty Cycle Directed Advertising
LE Link Layer Topology
Co-existence support for LTE

2014.12 V4.2

Secure connection
Data Packet Length Extension
Internet Protocol Support Profile (IPSP) version 6 ready

2017.02 V5.0

Hardware update
2 Mbit/s PHY for LE
LE Long Range
LE Advertising Extensions
LE Channel Selection Algorithm #2
Slot Availability Mask (SAM)

2017 BLE Mesh

Flood mesh
No routing table

Art of BLE

IOT

. . .

Smartphone

. . . . . .

Low Power

. . . . . .

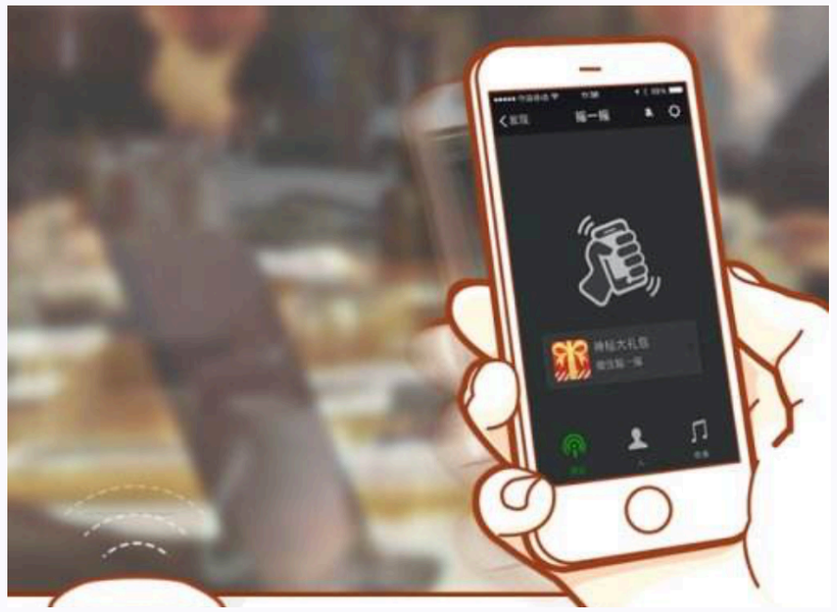
New

. . .

Low Cost

. . . . . .

Applications of BLE?

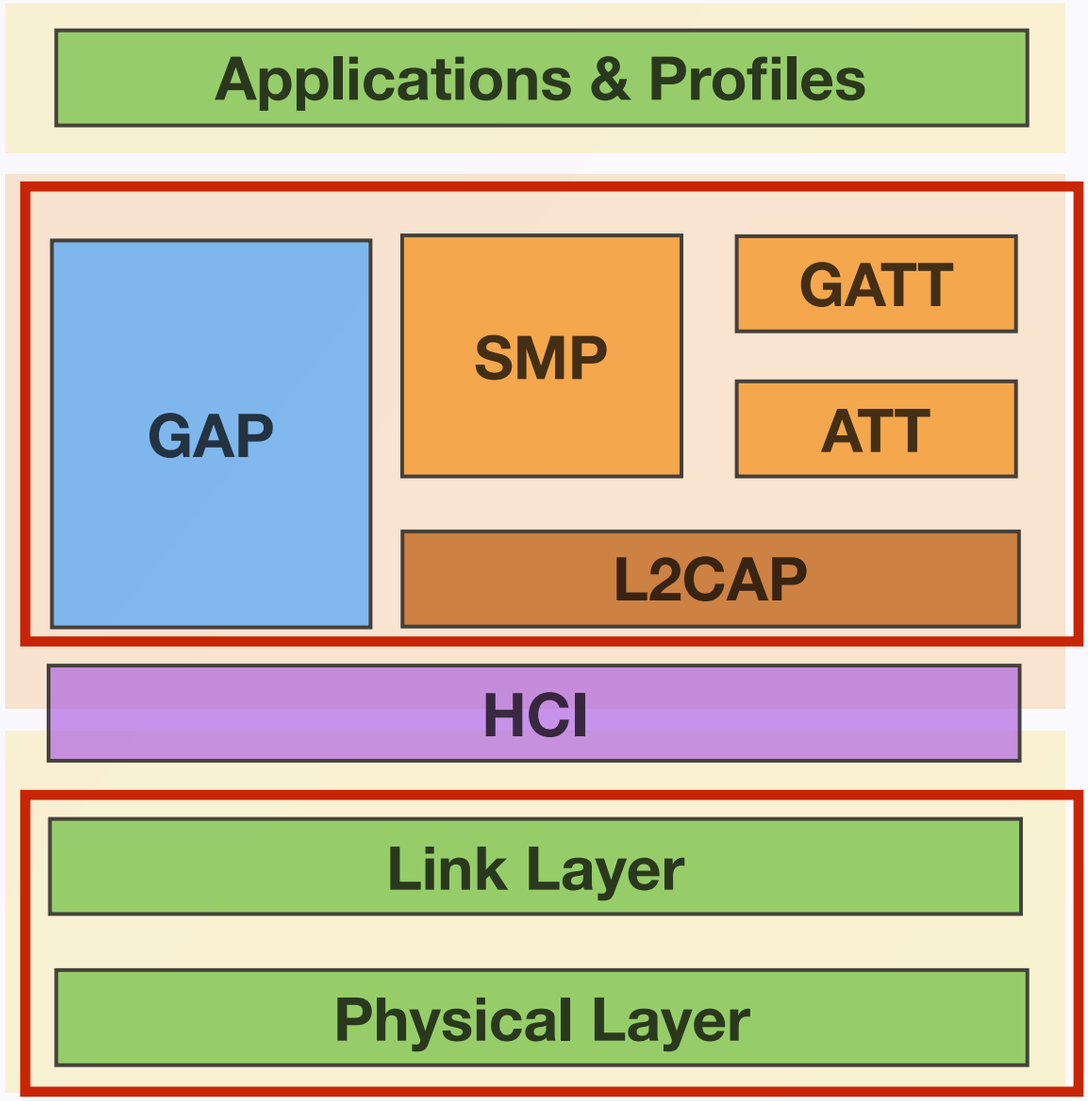


BLE Basic

Architecture of BLE

Architecture

- ➡ Controller and Host
- ➡ Dual-device solution
- ➡ Single-device solution (ESP32)
- ➡ Network Processor (AT Module)

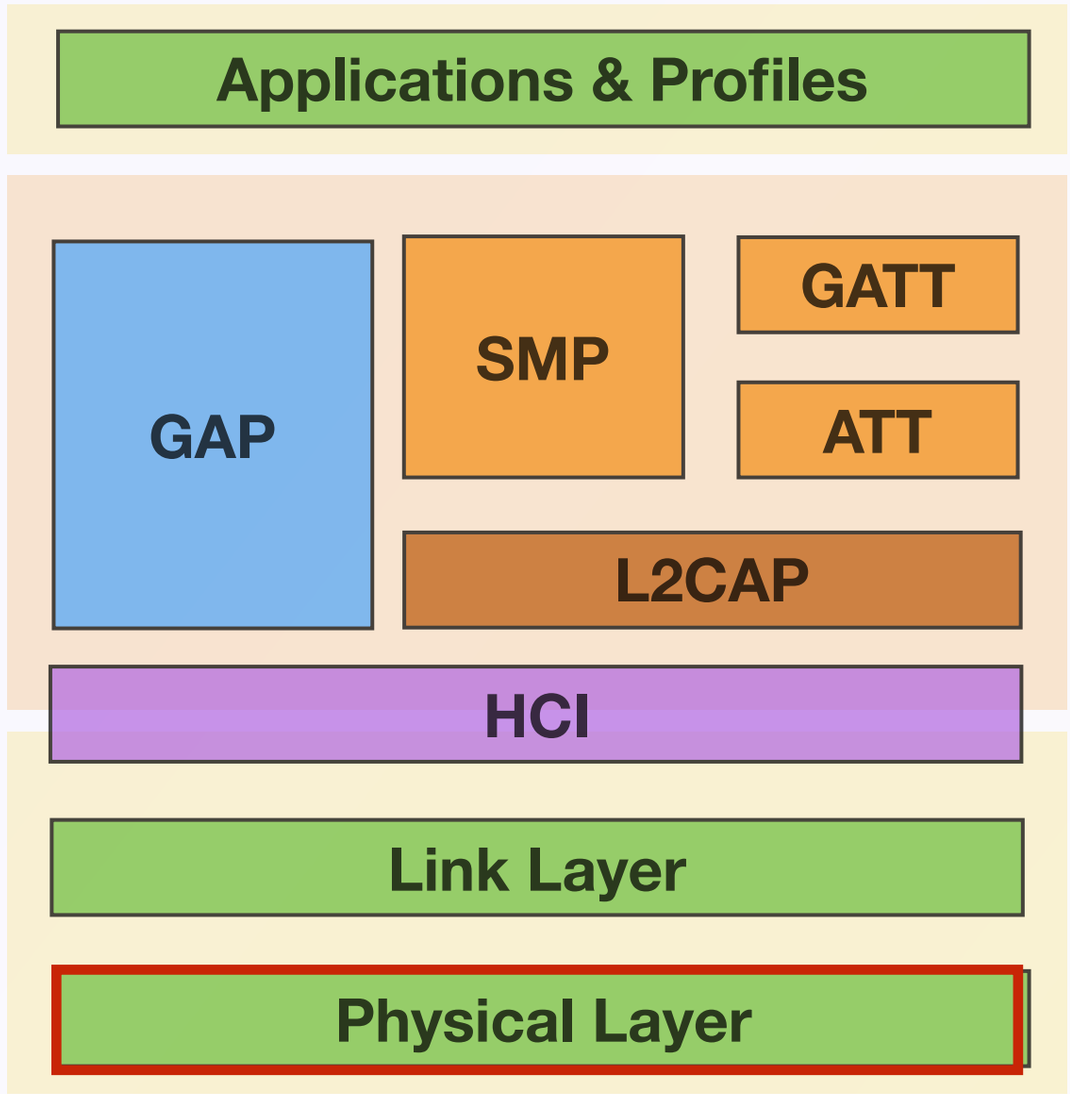


Physical Layer

Physical Layer

PHY

- ➡ Operates in 2.4 GHz ISM band
- ➡ GFSK modulation
- ➡ 40 channels with 2 MHz spacing
- ➡ TX Power <10 dBm (V4.2)
- ➡ RX Sensibility < -70dBm (V4.2)

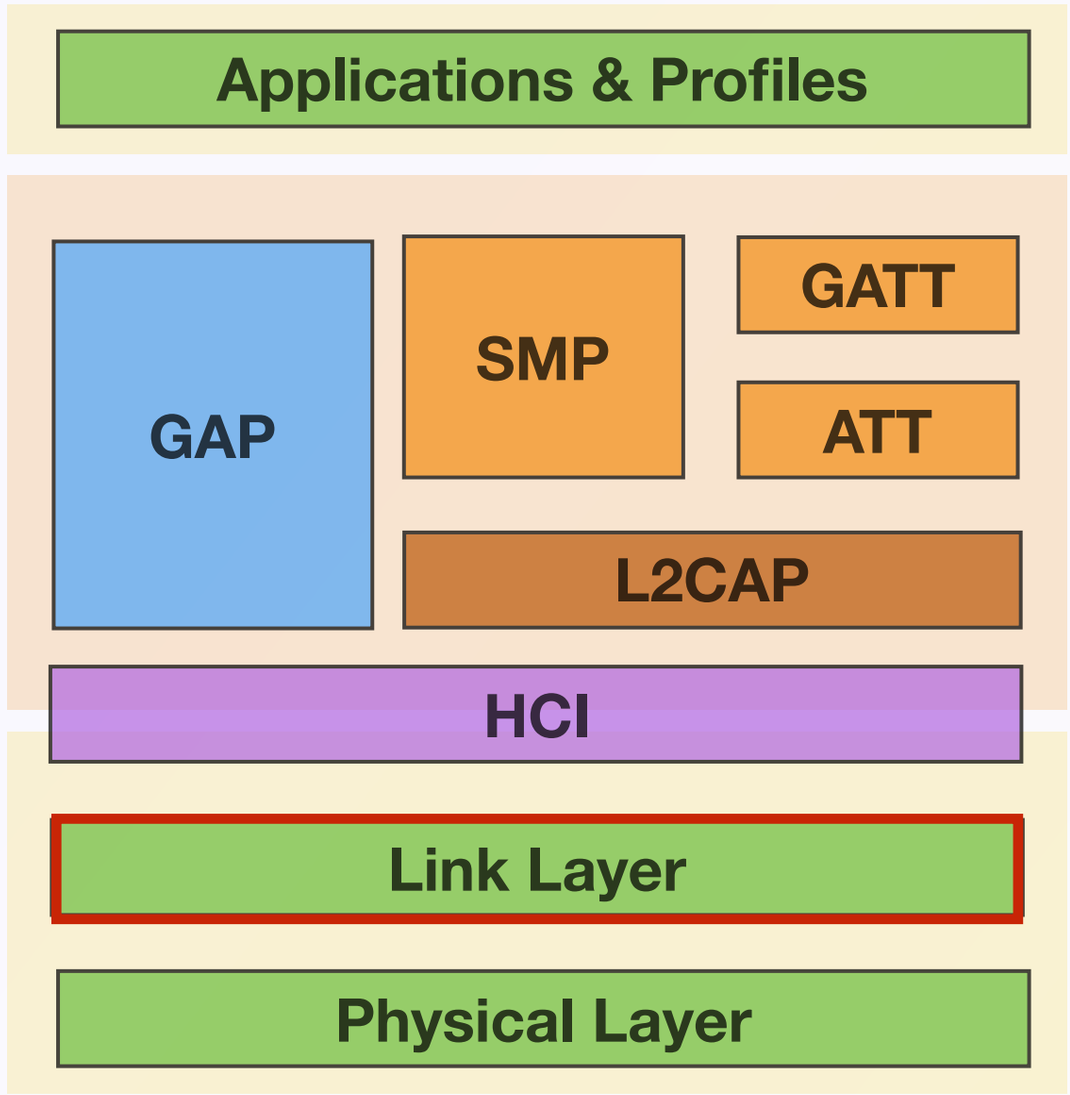


Link Layer

Link Layer

Link Layer

- ➡ Be responsible for low level communication over PHY
- ➡ Manages the sequence and timing of transmitted and received frames
- ➡ Advertising and scanning
- ➡ Encryption
- ➡ Channel hopping
- ➡ Uses HCI to communicate with upper layers of the stack



Master & Slave

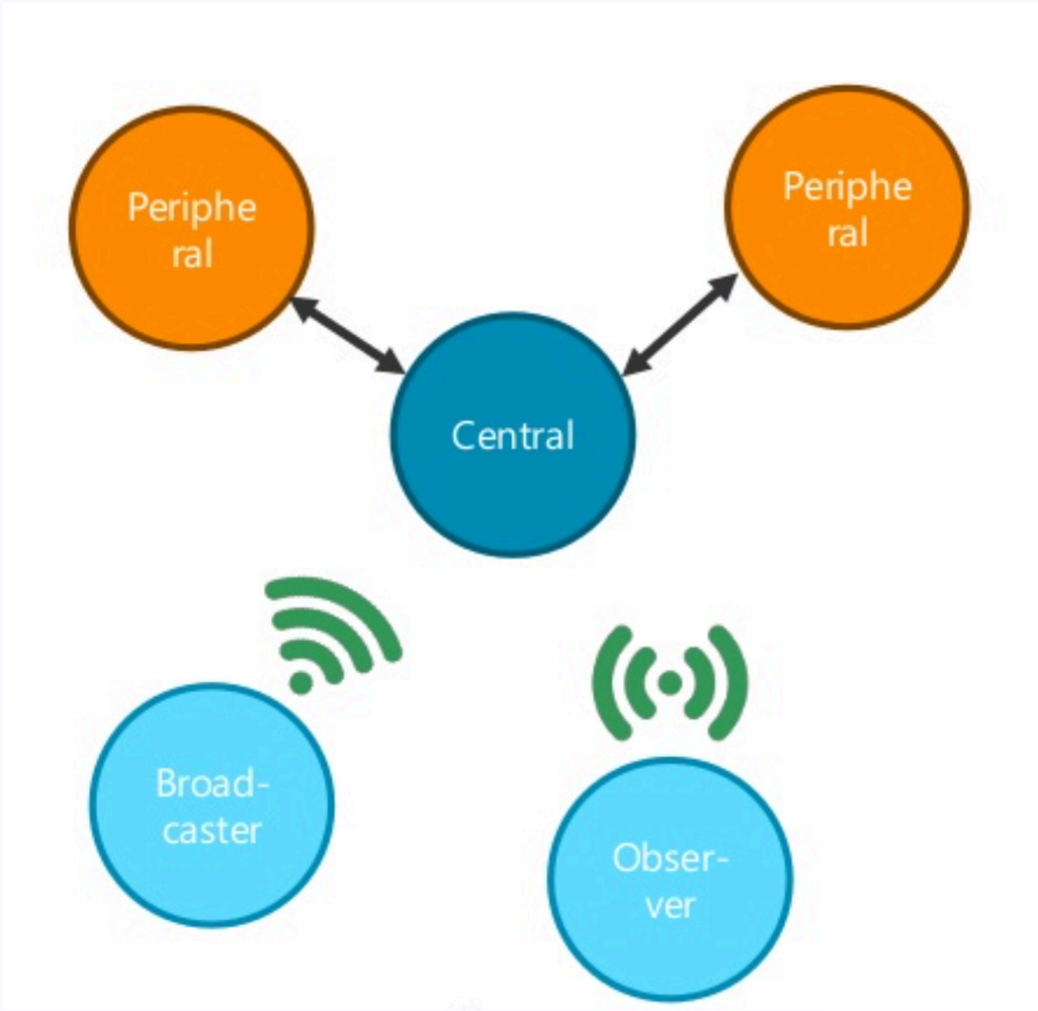
Slave / Peripheral:

➡ Device sending advertising packets

Master / Central:

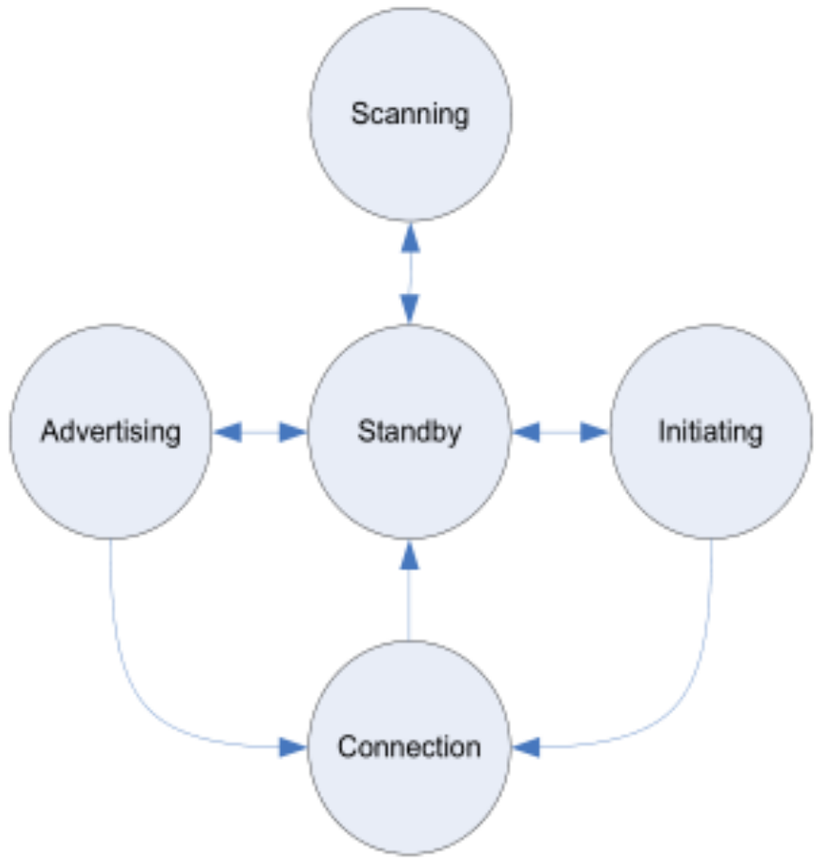
➡ Device initiating the connection

Question: Which simpler role ?



Link Layer Topology

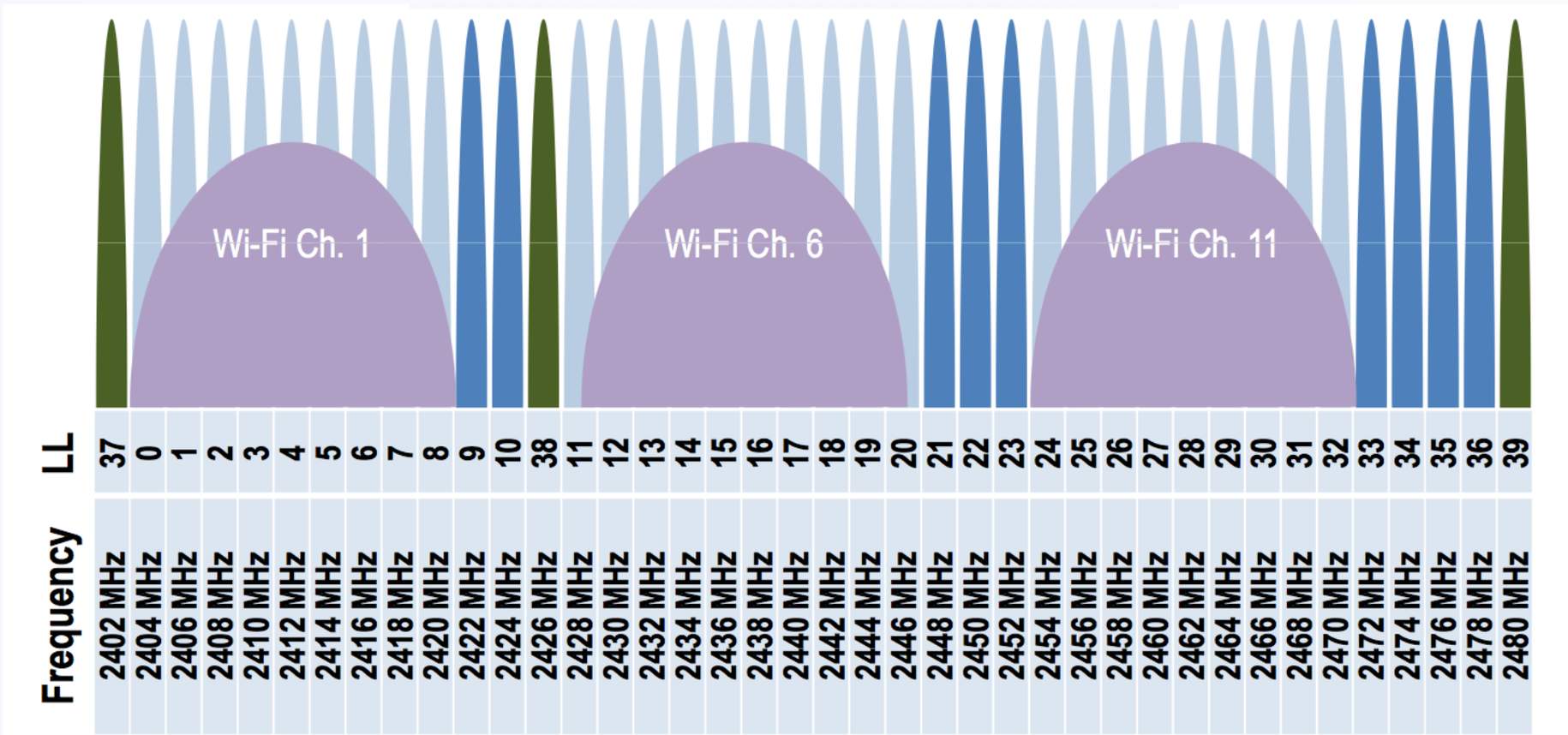
State		State Description
Standby		Does not transmit or receive packets
Advertising		Broadcasts advertisements in advertising channels
Scanning		Looks for advertisers
Initiating		Initiates connection to advertiser
Connection	Master	Communicates with device in the Slave role, defines timings of transmissions
	Slave	Communicates with device in Master Role



Link Layer Channels

3 Advertising Channels

- ➡ Channel 37(2402M), 38(2426M), 39(2480M)
- ➡ Avoid Wifi collision, 9 data channels available

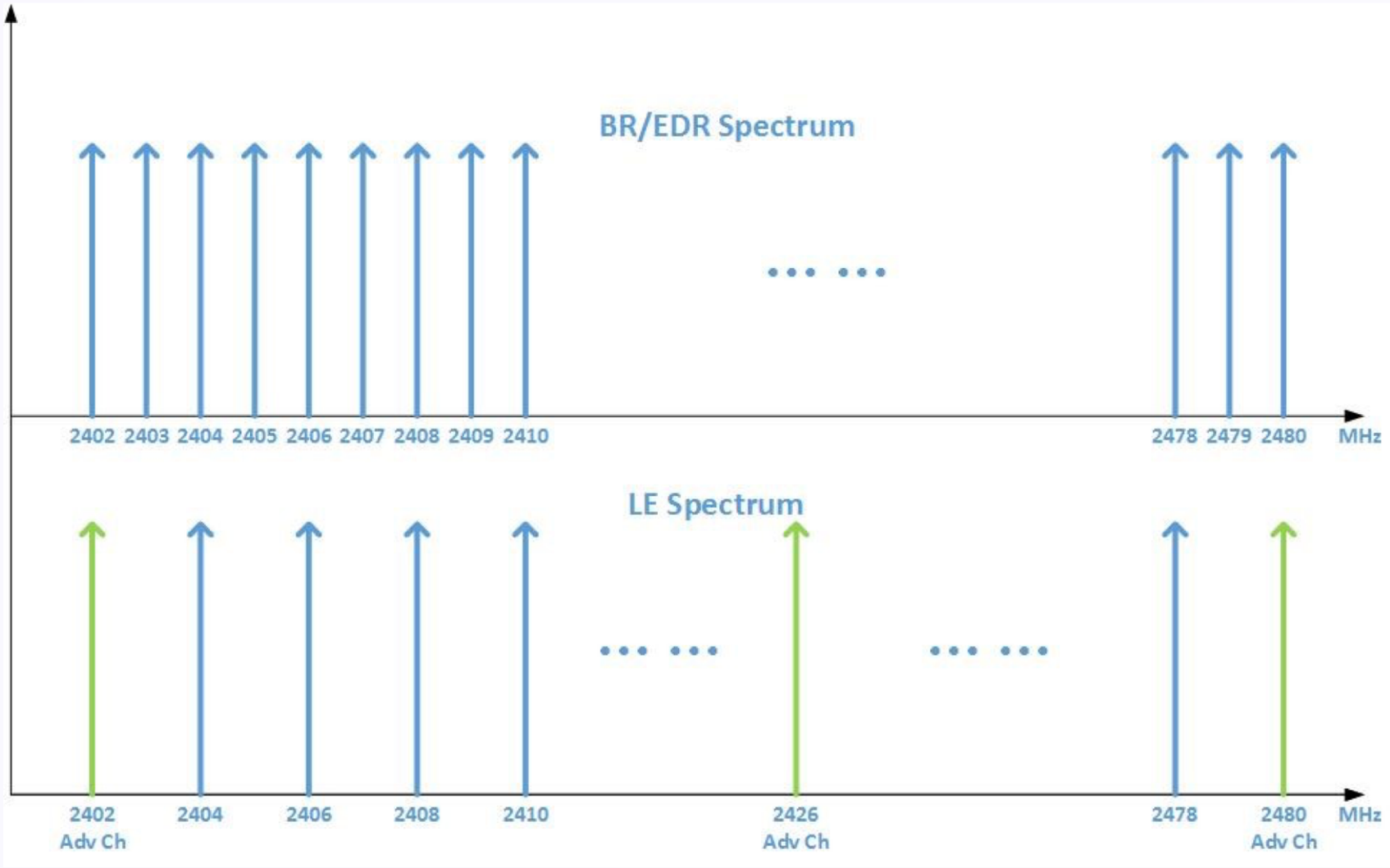


Question: Why 3 channels ?

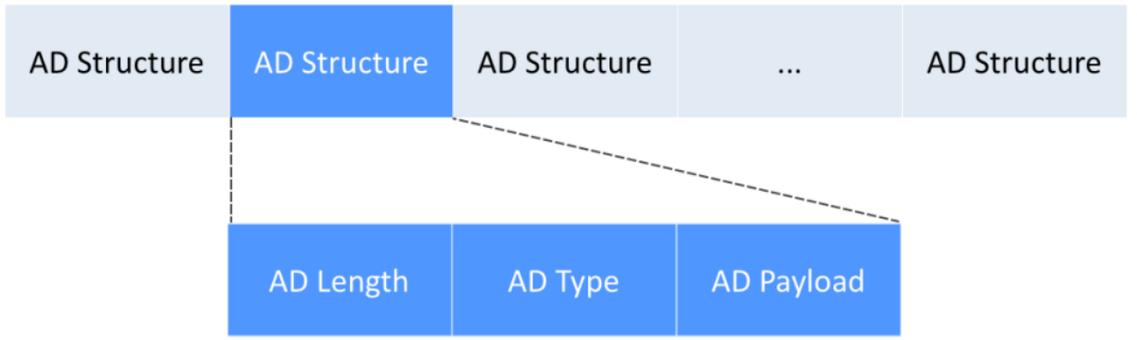
Link Layer Channels

Classic BT Channels

➡ 79 channels, covering from 2400 ~ 2483.5 MHz



Advertising Packets



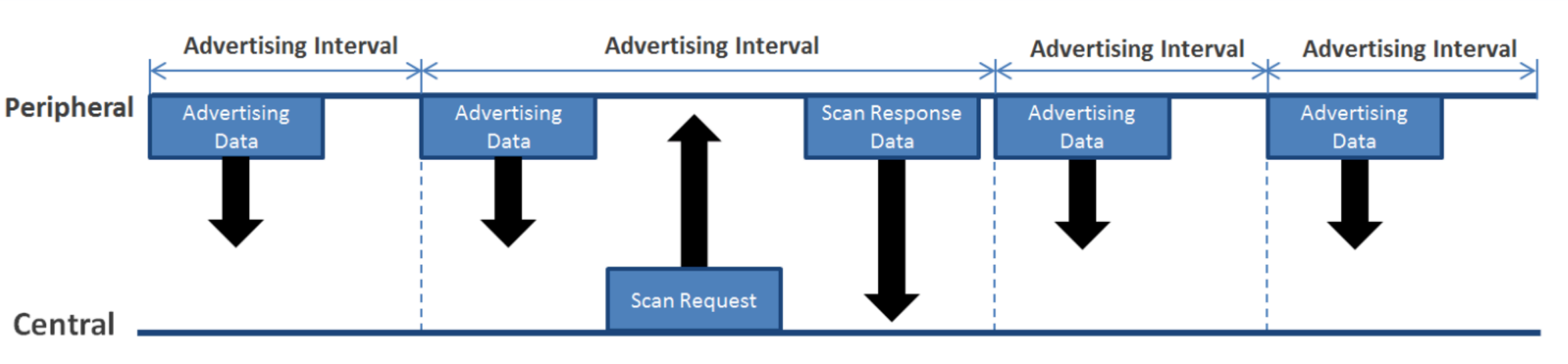
Adv data

- I’am general discoverable
- My name is “ESP BLE”
- I support Battery and Temperature services
- I transmit at 0 dBm

Advertising Packets

Adv packets

- ➡ Advertising interval: 20ms ~10.24s
- ➡ Pseudo-random delay of 10ms added
- ➡ Max 31 bytes (V4.2)



Question: Why add a pseudo-random delay ?

Advertising Packets

Adv packets

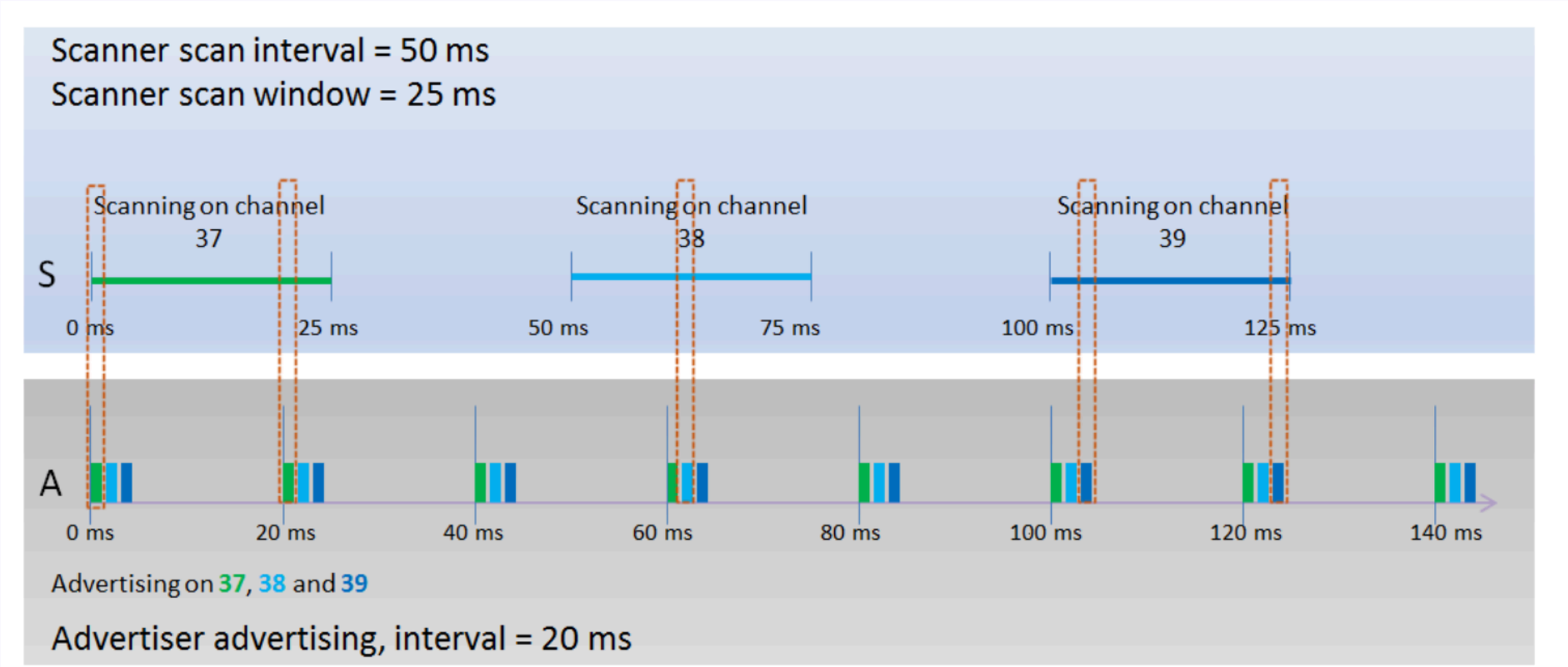
- ADV_IND: Normal
- DIRECT_IND: Fast connection
- NONCONN_IND: Beacon

PDU Type	Packet Name	Description
0000	ADV_IND	connectable undirected advertising event
0001	ADV_DIRECT_IND	connectable directed advertising event
0010	ADV_NONCONN_IND	non-connectable undirected advertising event
0011	SCAN_REQ	Scanner wants information from Advertiser
0100	SCAN_RSP	Advertiser gives more information to Scanner
0101	CONNECT_REQ	Initiator wants to connect to Advertiser
0110	ADV_DISCOVER_IND	non-connectable undirected advertising event

Scanning

Scan

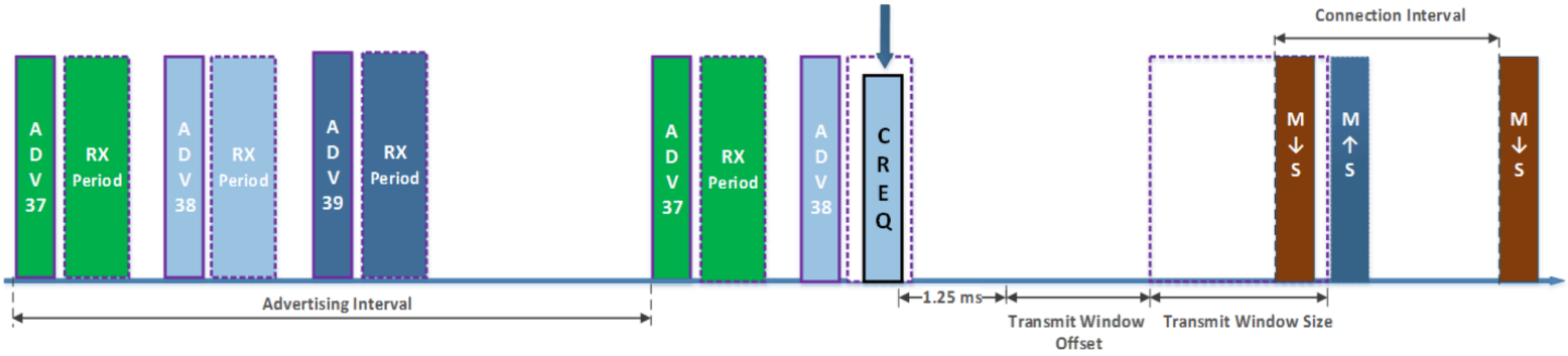
- ➡ Scan interval, scan window
- ➡ Scan on one channel at a time



Initiating Connection

Connection Request

➡ Connection timing, channel hopping, clock



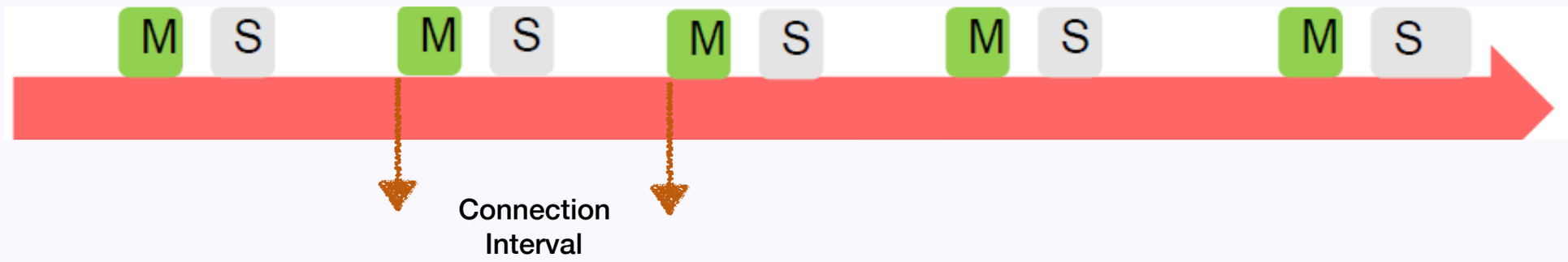
LLData									
AA (4 octets)	CRCInit (3 octets)	WinSize (1 octet)	WinOffset (2 octets)	Interval (2 octets)	Latency (2 octets)	Timeout (2 octets)	ChM (5 octets)	Hop (5 bits)	SCA (3 bits)

Figure 2.11: LLData field structure in CONNECT_REQ PDU's payload

Connection Parameters

Connection Interval

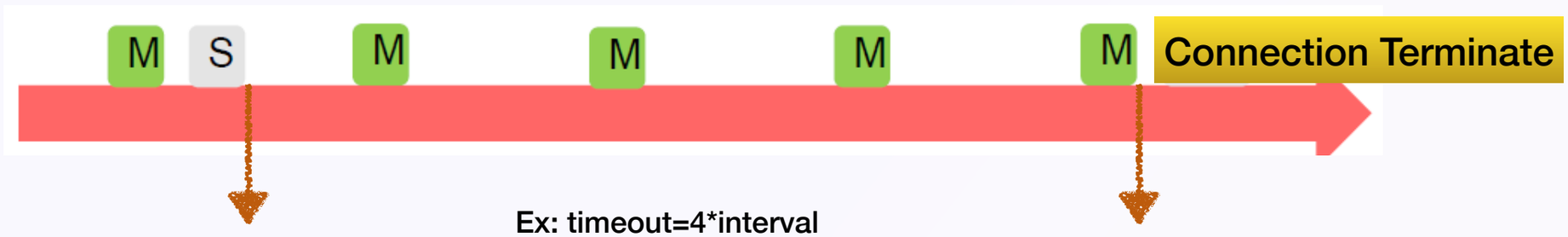
- ➡ How frequently the master will send a connection event packet to slave
- ➡ Interval = value *1.25ms, 7.5ms~4s
- ➡ Tradeoffs: power consumption, throughput, real time



Connection Parameters

Supervision Timeout

- ➡ How long would the master keep sending connection event without response from slave before terminate the connection
- ➡ Timeout = value*10ms, 100ms~32s
- ➡ Must larger than (1+latency)*interval*2

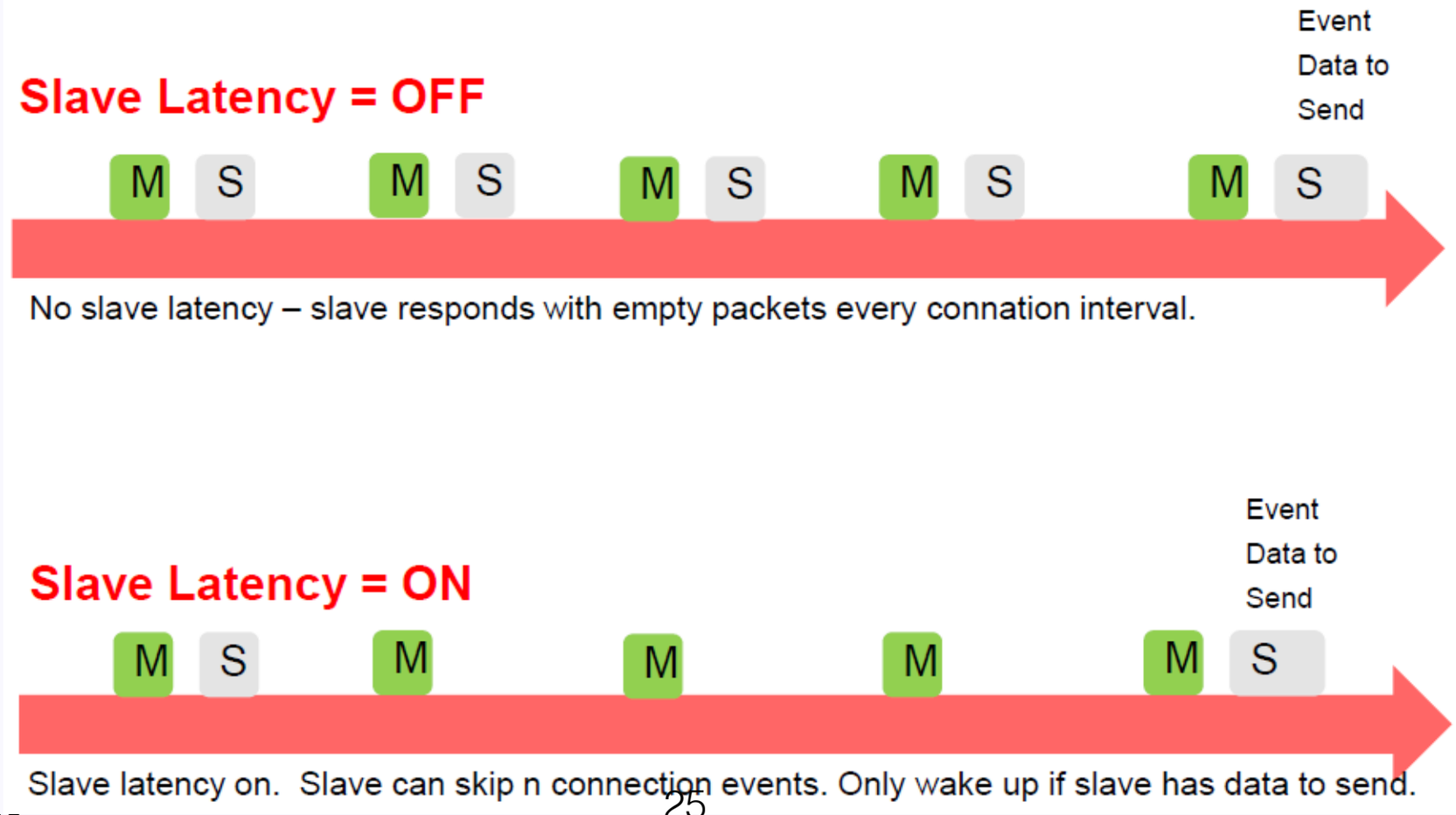


Ex: timeout=4*interval

Connection Parameters

Latency

- ➡ Number of connect event the slave can skip
- ➡ Latency: 0~499
- ➡ Tradeoffs: power consumption, real time



Connection Parameter Update

Slave Initiates Param Update

- ➡ Slave initiates param update if it does not like the connection params
- ➡ Master decide to accept or to reject new params
- ➡ Ex: 0x310*1.25ms=980ms, 0x320*1.25ms=1000ms, 0x258*10ms=6000ms, 0x31E*1.25ms=997.5ms,
- ➡ SIG_Result: 0x00 accepted, 0x01 rejected

L2CAP Header		SIG Pkt Header			SIG_Connection_Param_Update_Req			
L2CAP-Length	ChanId	Code	Id	Data-Length	IntervalMin	IntervalMax	SlaveLatency	TimeoutMultiplier
0x000C	0x0005	0x12	0x01	0x0008	0x0310	0x0320	0x0000	0x0258

LL_Opcode	LL_Connect_Update_Req						CRC
	WinSize	WinOffset	Interval	Latency	Timeout	Instant	
Connection_Update_Req(0x00)	0x03	0x00CC	0x031E	0x0000	0x0258	0x0043	0x11E5FD

CRC	RSSI (dBm)	FCS
0x91F8FB	-54	OK

L2CAP Header		SIG Pkt Header			SIG_Connection_Param_Update_Rsp	CRC	RSSI (dBm)	FCS
L2CAP-Length	ChanId	Code	Id	Data-Length	Result			
0x0006	0x0005	0x13	0x01	0x0002	0x0000	0x9DA0B7	-54	OK

Terminate Connection

Terminate Connection

- ➡ Termination can be initiated by slave or master
- ➡ Termination can also be caused by supervision timeout

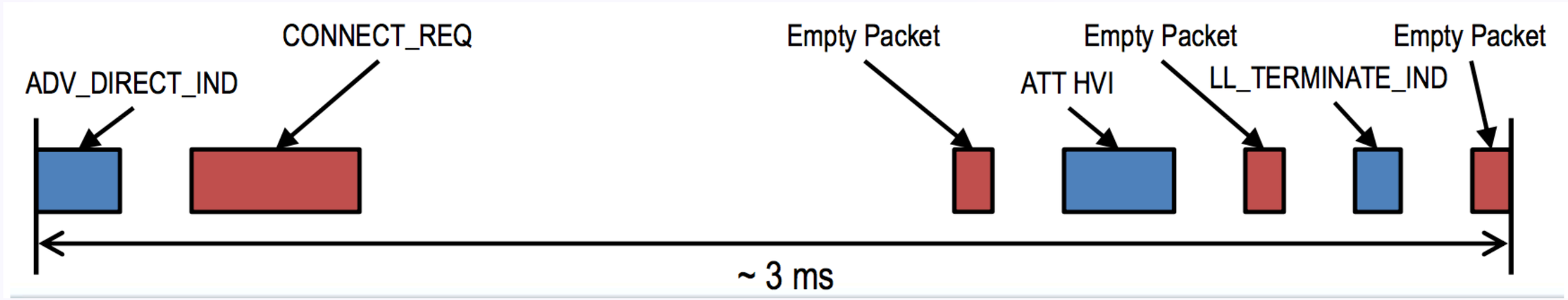
Data Type	Data Header					LL_Opcode	LL_Terminate_Ind
	LLID	NESN	SN	MD	PDU-Length		ErrorCode
Control	3	0	0	0	2	Terminate_Ind(0x02)	0x13

Question: Which simpler role ?

Connectionless Mode

Fast Connection

- ➡ Time from advertising to termination: 3 ms
- ➡ Indicate one byte data: 144 us
- ➡ Terminate command: 96 us
- ➡ Empty packet: 80 us

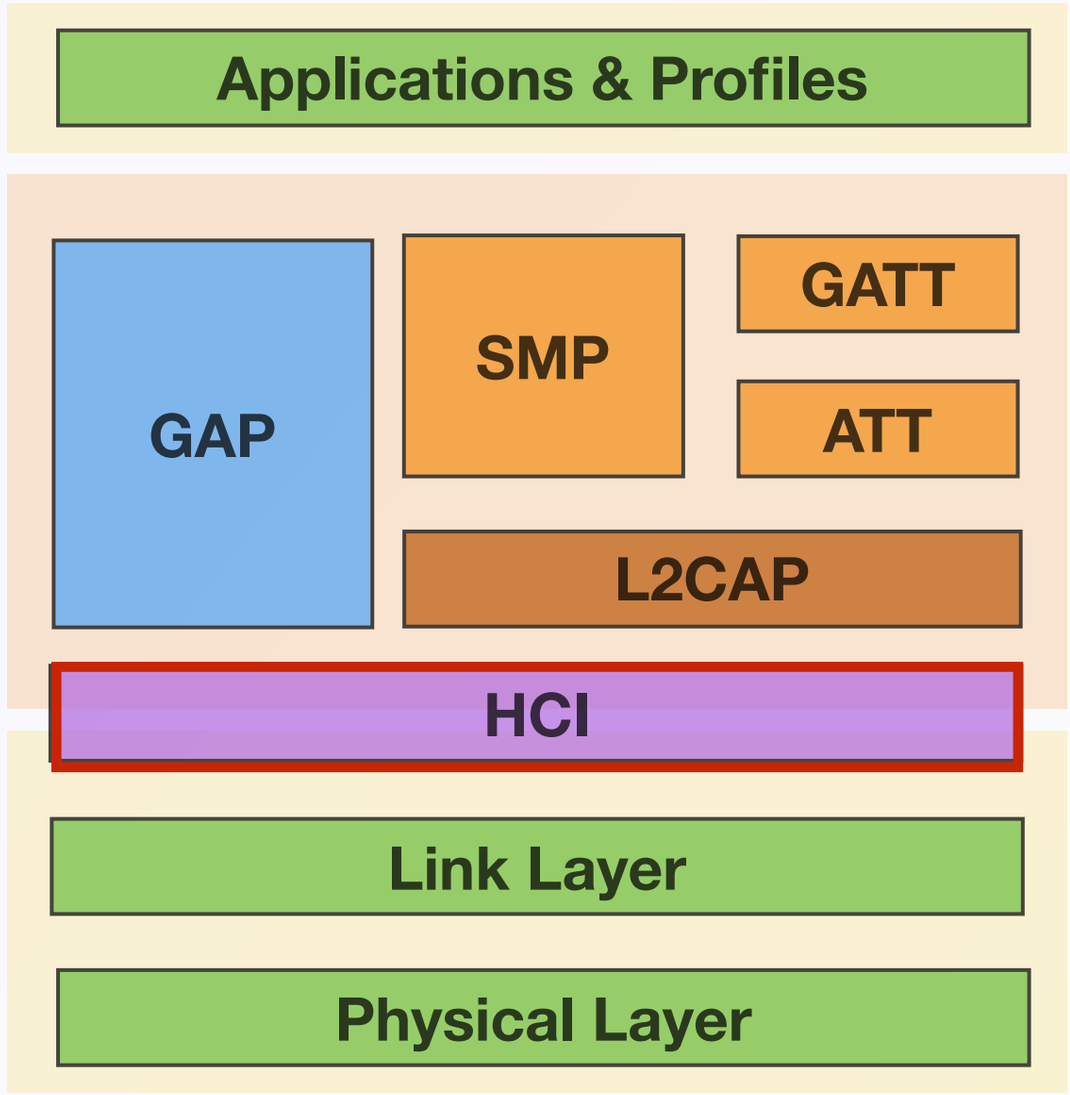


Host Controller Interface

Host Controller Interface

HCI

- ➡ Reused form classic BT
- ➡ Dual-chip solution
- ➡ ESP32, Single-chip solution

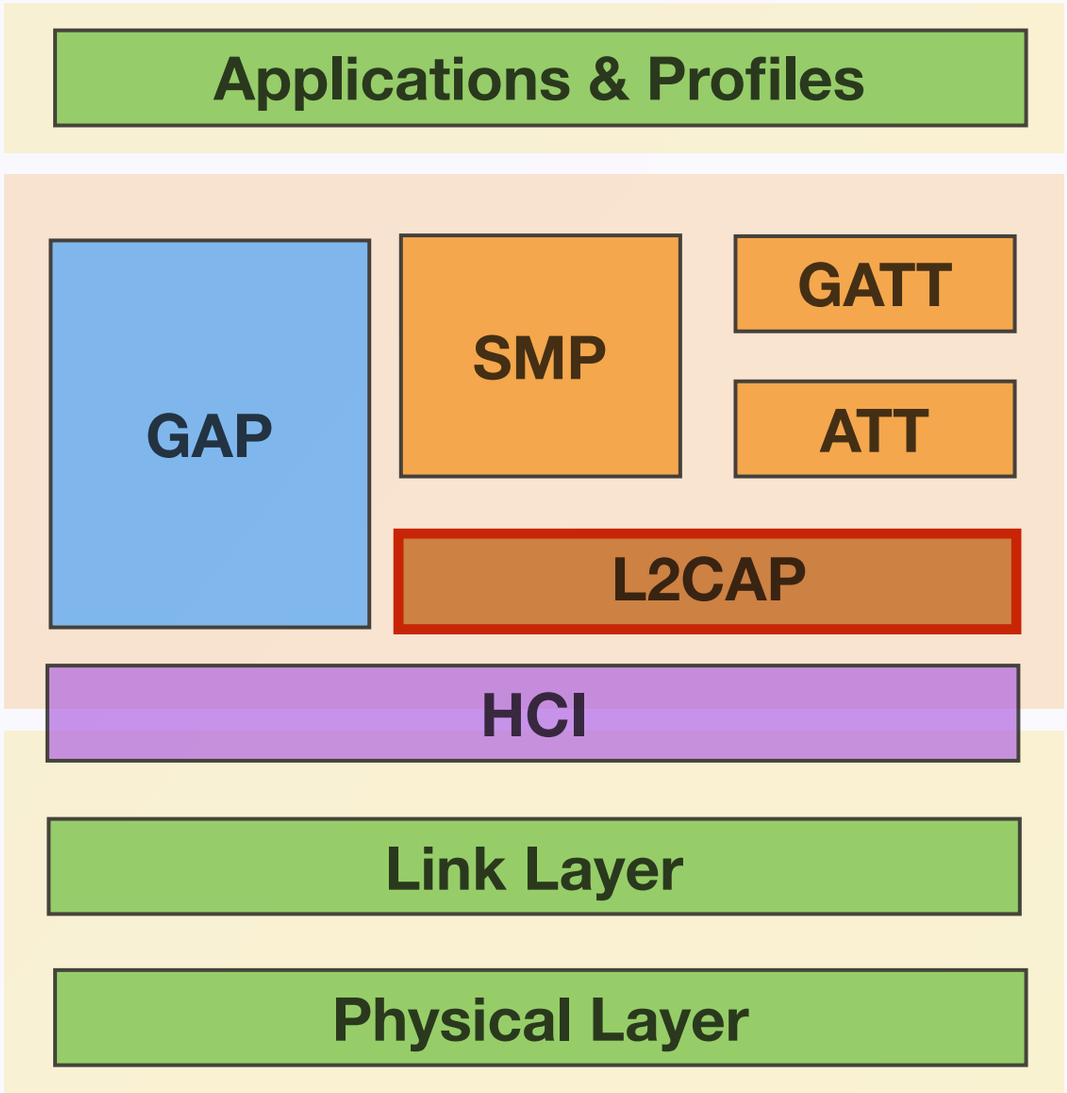


L2CAP

Logical Link Control and Adaptation Protocol

L2CAP

- ➡ Logical channel management
  - ATT Channel 4
  - SIG Channel 5
  - SMP Channel 6
- ➡ Connection parameter update

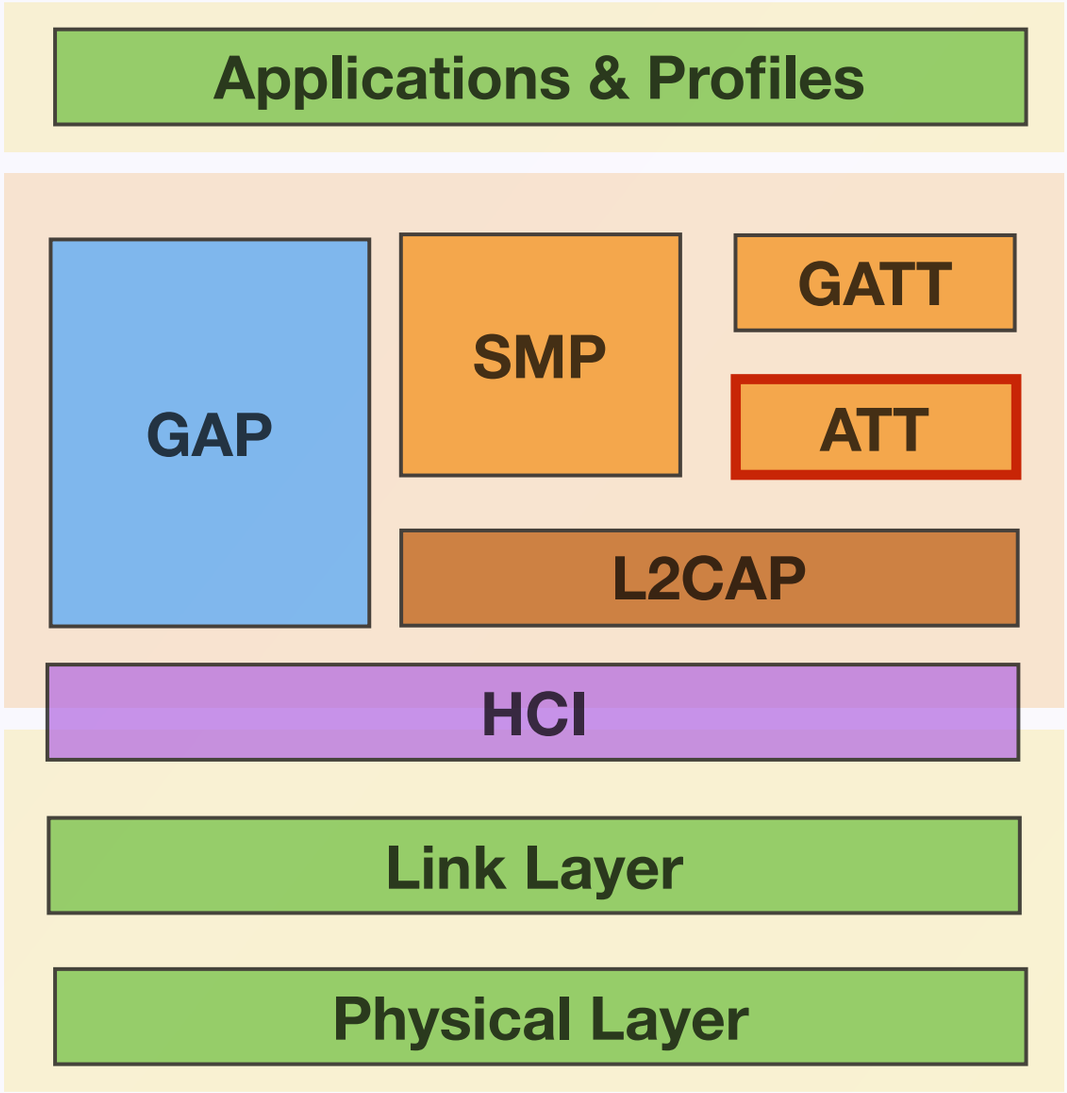


Attribute Protocol (ATT)

Attribute Protocol

ATT

- ➡ Defines the over-the-air protocol for reading, writing, and discovering attributes
- ➡ Allows for different permissions to be assigned to attributes



Client & Server

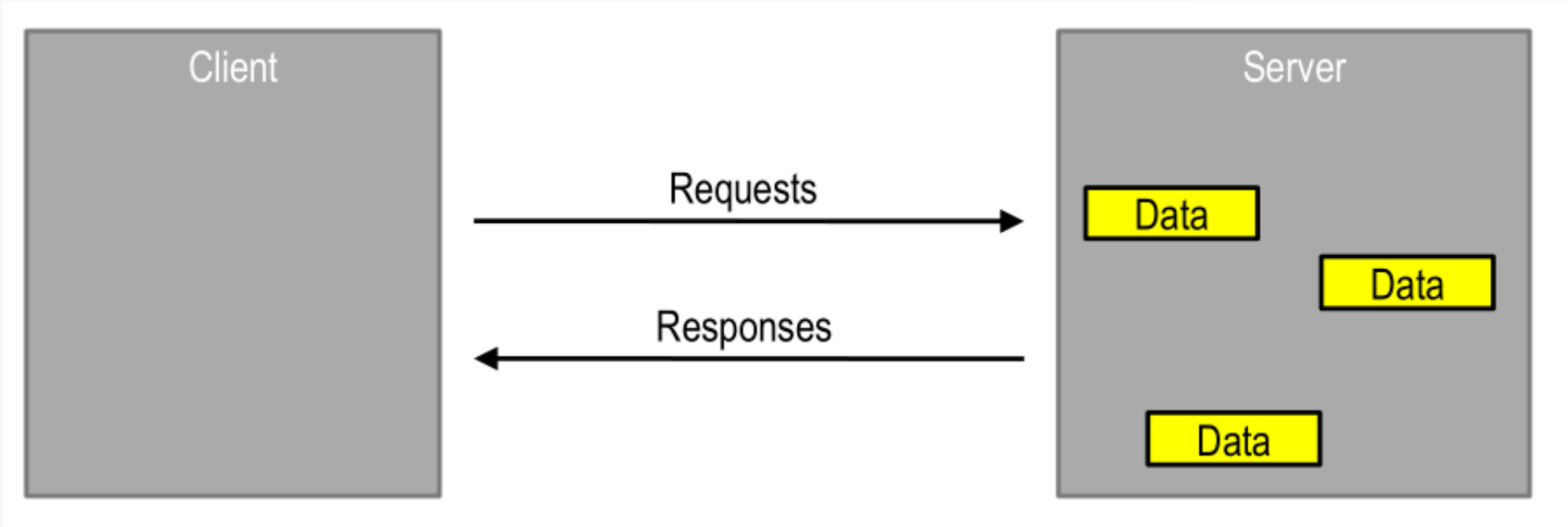
Web-infrastructure

Server:

- ➡ Have data
- ➡ Expose data using attributes

Client:

- ➡ Request data



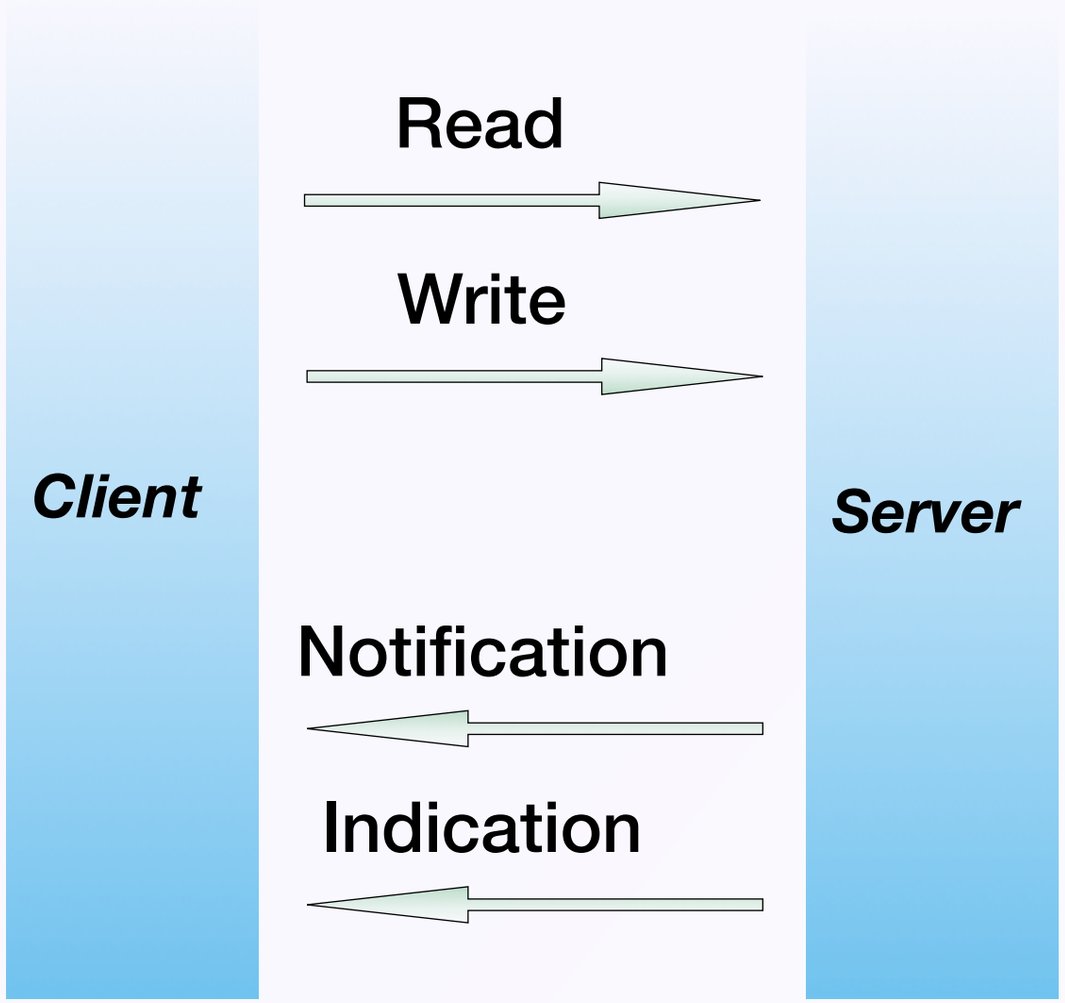
Attribute

Each attribute consists of:

- ➡ Handle: sequence number of attribute, like memory address
- ➡ Attribute type (UUID): what the value means
- ➡ Permissions: clients how to access the value, not discoverable
- ➡ Attribute value: array of octets, length constant or variable

Handle	UUID	Permissions	Value
0x0001			
0x0002			
.....			
0xFFFE			
0xFFFF			

Data Access



Data Access

Write Request vs Command; Indication vs Notification

- ➡ Request/Indication cause a response
- ➡ Only one request/indication at a time, reliable
- ➡ Command/Notification can be sent at any time, may cause buffer overflows, unreliable

Atomic Operations

- ➡ Can’t be affected by another client

Attribute Example

Bluedroid GATT Service Example

Handle	UUID	Permissions	Value
0x0001	0x2800 (Service Delaration)	Read	0x1801 (GATT Service)
0x0002	0x2803 (Characteristic Delaration)	Read	0x20, 0x03, 0x00, 0x05, 0x2A
0x0003	0x2A05 (Service Change Char)	None	0x01 or 0x00
0x0004 (optional)	0x2902 (Client Char Configuration)	Write	0x0002 or 0x0000

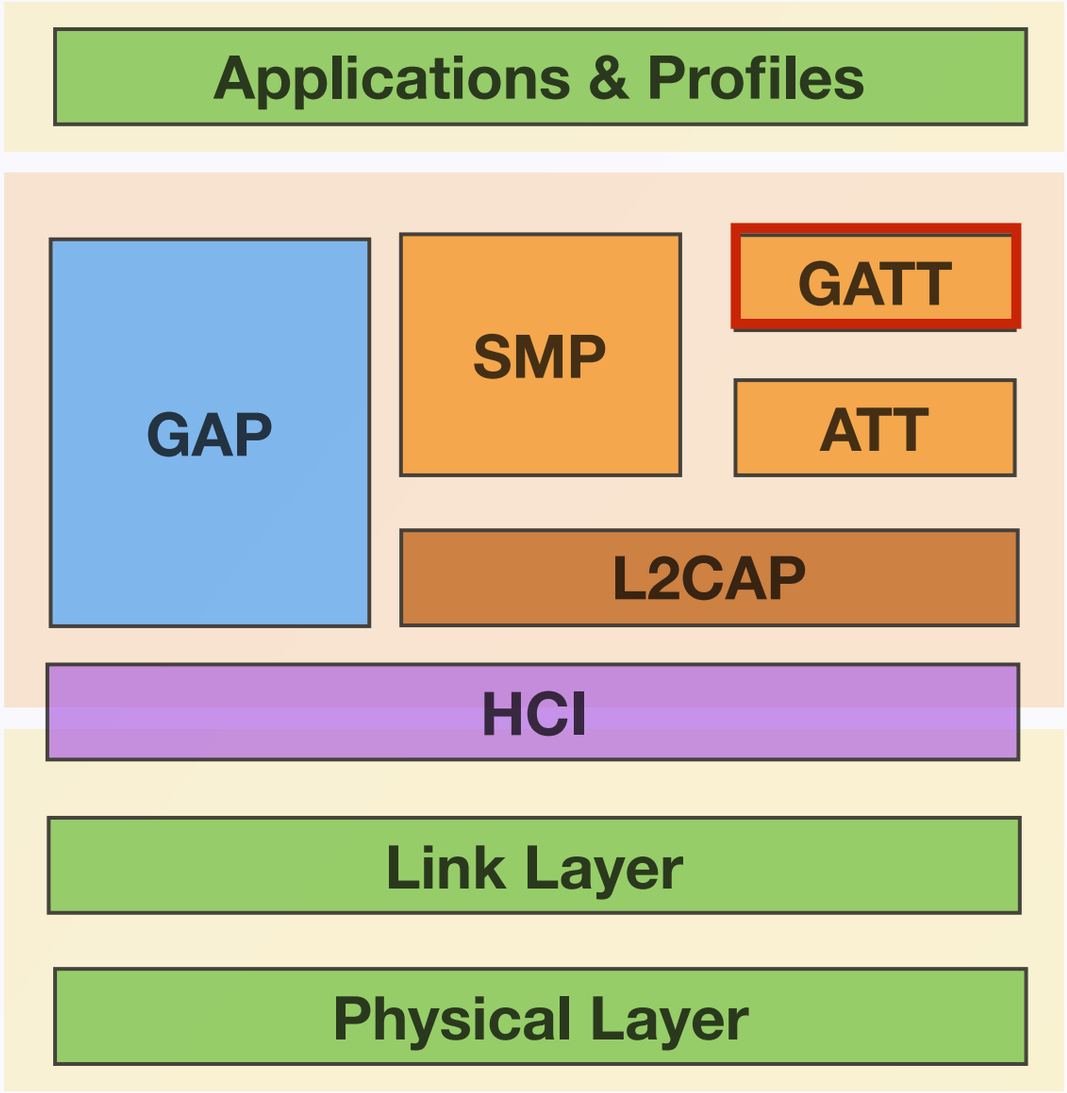
```
/* add Service Changed characteristic
*/
uuid.uu.uuid16 = gatt_cb.gattp_attr.uuid = GATT_UUID_GATT_SRV_CHGD;
gatt_cb.gattp_attr.service_change = 0;
gatt_cb.gattp_attr.handle   =
gatt_cb.handle_of_h_r       = GATTS_AddCharacteristic(service_handle, &uuid, 0, GATT_CHAR_PROP_BIT_INDICATE,
                                                      NULL, NULL);
```

Generic Attribute Profile (GATT)

Generic Attribute Profile

GATT

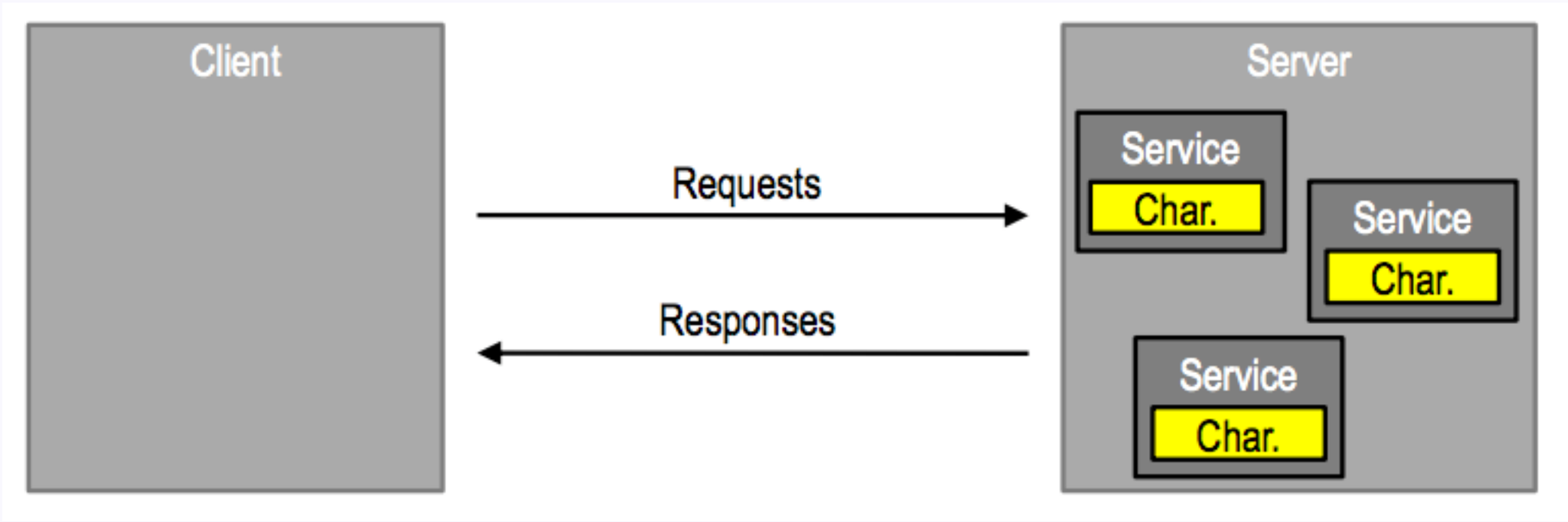
- Uses characteristic to present a data
  - property, unit, name, exponent, etc.
- The grouping and relationship of characteristics within a service or profile
- Procedures for using the attribute protocol (ATT) to discover, read, write, and obtain indications of these attributes



GATT

GATT

- ➡ Defines access protocol to discover and access services and their characteristics



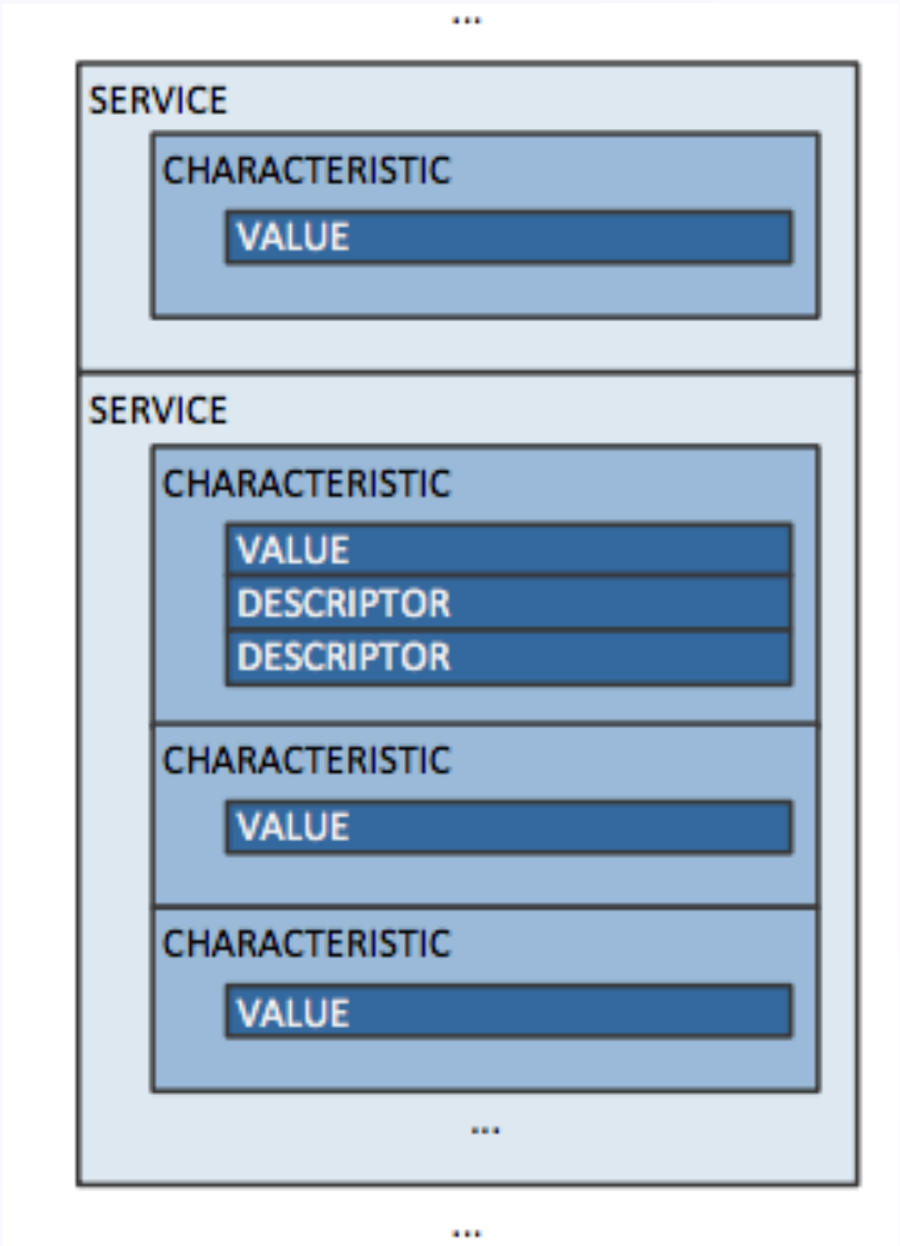
Service & Characteristic

Service:

- ➡ Profile is composed of services
- ➡ Service is composed of characteristics

Characteristic:

- ➡ Characteristic declaration
- ➡ Characteristic value
- ➡ Characteristic descriptor (optional)
- ➡ At least 2 handles



Service Example

Bluedroid GATT Service Example

- ➡ Char Declaration
  - 0x20 Indication
  - 0x0003 Char value handle
  - 0x2A05 Char value UUID
- ➡ Char Value 0x01 or 0x00
- ➡ Char Descriptor 0x0002 or 0x0000

Handle	UUID	Permissions	Value
0x0001	0x2800 (Service Delaration)	Read	0x1801 (GATT Service)
0x0002	0x2803 (Characteristic Delaration)	Read	0x20, 0x03, 0x00, 0x05, 0x2A
0x0003	0x2A05 (Service Change Char)	None	0x01 or 0x00
0x0004 (optional)	0x2902 (Client Char Configuration)	Write	0x0002 or 0x0000

Create Attribute Table

```c
typedef struct
{
    uint16_t uuid_length;
    uint8_t  *uuid_p;
    uint16_t perm;
    uint16_t max_length;
    uint16_t length;
    uint8_t  *value;
} esp_attr_desc_t;
```

```c
#define     ESP_GATT_PERM_READ                  (1 << 0)
#define     ESP_GATT_PERM_READ_ENCRYPTED        (1 << 1)
#define     ESP_GATT_PERM_READ_ENC_MITM         (1 << 2)
#define     ESP_GATT_PERM_WRITE                 (1 << 4)
#define     ESP_GATT_PERM_WRITE_ENCRYPTED       (1 << 5)
#define     ESP_GATT_PERM_WRITE_ENC_MITM        (1 << 6)
#define     ESP_GATT_PERM_WRITE_SIGNED          (1 << 7)
#define     ESP_GATT_PERM_WRITE_SIGNED_MITM     (1 << 8)
typedef uint16_t esp_gatt_perm_t;
```

```c
#define     ESP_GATT_CHAR_PROP_BIT_BROADCAST    (1 << 0)
#define     ESP_GATT_CHAR_PROP_BIT_READ         (1 << 1)
#define     ESP_GATT_CHAR_PROP_BIT_WRITE_NR     (1 << 2)
#define     ESP_GATT_CHAR_PROP_BIT_WRITE        (1 << 3)
#define     ESP_GATT_CHAR_PROP_BIT_NOTIFY       (1 << 4)
#define     ESP_GATT_CHAR_PROP_BIT_INDICATE     (1 << 5)
#define     ESP_GATT_CHAR_PROP_BIT_AUTH         (1 << 6)
#define     ESP_GATT_CHAR_PROP_BIT_EXT_PROP     (1 << 7)
typedef uint8_t esp_gatt_char_prop_t;
```

```c
/// Full HRS Database Description - Used to add attributes into the database
static const esp_gatts_attr_db_t heart_rate_gatt_db[HRS_IDX_NB] =
{
    // Heart Rate Service Declaration
    [HRS_IDX_SVC]                       =
    {{ESP_GATT_AUTO_RSP}, {ESP_UUID_LEN_16, (uint8_t *)&primary_service_uuid, ESP_GATT_PERM_READ,
      sizeof(uint16_t), sizeof(heart_rate_svc), (uint8_t *)&heart_rate_svc}},

    // Heart Rate Measurement Characteristic Declaration
    [HRS_IDX_HR_MEAS_CHAR]              =
    {{ESP_GATT_AUTO_RSP}, {ESP_UUID_LEN_16, (uint8_t *)&character_declaration_uuid, ESP_GATT_PERM_READ,
      CHAR_DECLARATION_SIZE,CHAR_DECLARATION_SIZE, (uint8_t *)&char_prop_notify}},

    // Heart Rate Measurement Characteristic Value
    [HRS_IDX_HR_MEAS_VAL]               =
    {{ESP_GATT_AUTO_RSP}, {ESP_UUID_LEN_16, (uint8_t *)&heart_rate_meas_uuid, ESP_GATT_PERM_READ,
      HRPS_HT_MEAS_MAX_LEN,0, NULL}},

    // Heart Rate Measurement Characteristic - Client Characteristic Configuration Descriptor
    [HRS_IDX_HR_MEAS_NTF_CFG]       =
    {{ESP_GATT_AUTO_RSP}, {ESP_UUID_LEN_16, (uint8_t *)&character_client_config_uuid, ESP_GATT_PERM_READ|ESP_GATT_PERM_WRITE,
      sizeof(uint16_t),sizeof(heart_measurement_ccc), (uint8_t *)heart_measurement_ccc}},
```

Service discovery

Find Service

- ➡ Handle, 0x0001~0x0005; Service, 0x1801 (GATT)
- ➡ Handle, 0x0014~0x001C; Service, 0x1800 (GAP)

Service discovery

Find Characteristic

➡ Handle, 0x0002~0x0003; Characteristic, 0x2A05 (Service Change Char)

L2CAP Header		ATT_Read_By_Type_Req			
L2CAP-Length	ChanId	Opcode	StartingHandle	EndingHandle	AttType
0x0007	0x0004	0x08	0x0001	0x0005	03 28

L2CAP Header		ATT_Read_By_Type_Rsp		
L2CAP-Length	ChanId	Opcode	Length	AttData
0x0009	0x0004	0x09	0x07	02 00 20 03 00 05 2A

L2CAP Header		ATT_Read_By_Type_Req			
L2CAP-Length	ChanId	Opcode	StartingHandle	EndingHandle	AttType
0x0007	0x0004	0x08	0x0004	0x0005	03 28

L2CAP Header		ATT_Error_Response			
L2CAP-Length	ChanId	Opcode	ReqOpCode	AttHandle	ErrorCode
0x0005	0x0004	0x01	0x08	0x0004	ATT_NOT_FOUND(0x0A)

Service discovery

Find Descriptor

➡ Handle, 0x0004; Client Char Configuration, 0x2902

L2CAP Header		ATT_Find_Info_Req			
L2CAP-Length	ChanId	Opcode	StartingHandle	EndingHandle	UUIDFilter
0x0005	0x0004	0x04	0x0004	0x0005	None

L2CAP Header		ATT_Find_Info_Rsp		
L2CAP-Length	ChanId	Opcode	Format	InfoData
0x0006	0x0004	0x05	0x01	04 00 02 29

ATT_Find_Info_Req			
Opcode	StartingHandle	EndingHandle	UUIDFilter
0x04	0x0005	0x0005	None

L2CAP Header		ATT_Find_Info_Req			
L2CAP-Length	ChanId	Opcode	StartingHandle	EndingHandle	UUIDFilter
0x0005	0x0004	0x04	0x0005	0x0005	None

Service discovery

Write Descriptor

➡ Handle, 0x0004; Client Char Configuration, 0x2A05

L2CAP Header		ATT_Write_Req		
L2CAP-Length	ChanId	Opcode	AttHandle	AttValue
0x0005	0x0004	0x12	0x0004	02 00

L2CAP Header		ATT_Write_Rsp
L2CAP-Length	ChanId	Opcode
0x0001	0x0004	0x13

Security Manager Protocol (SMP)

Security Manager Protocol

Pairing

➡ Exchange Key

Authentication

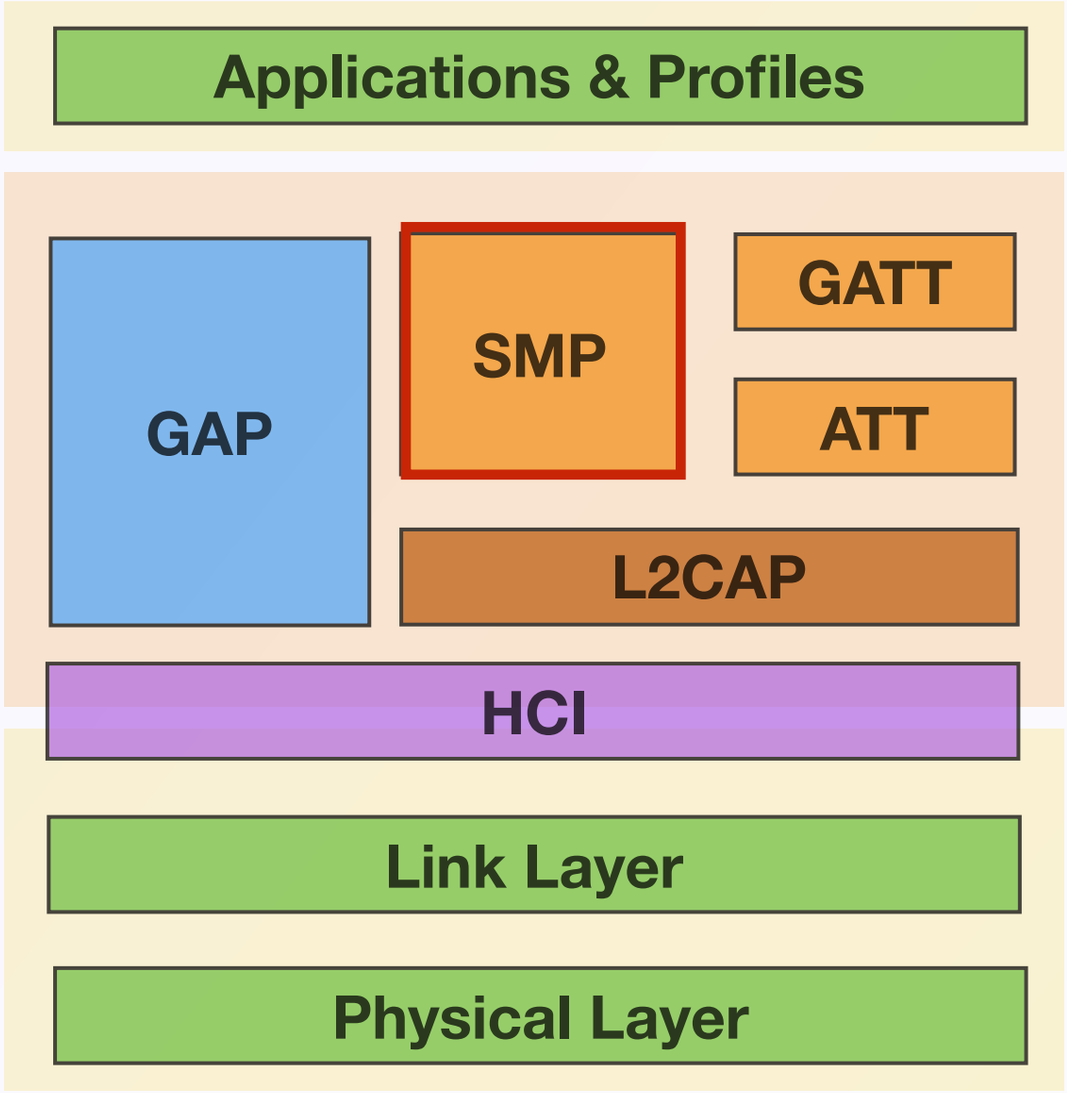
➡ Verify that a peer device can be trusted

➡ Against “Man-in-the-Middle”

Bonding

➡ Store Key

➡ Reconnect



Homework

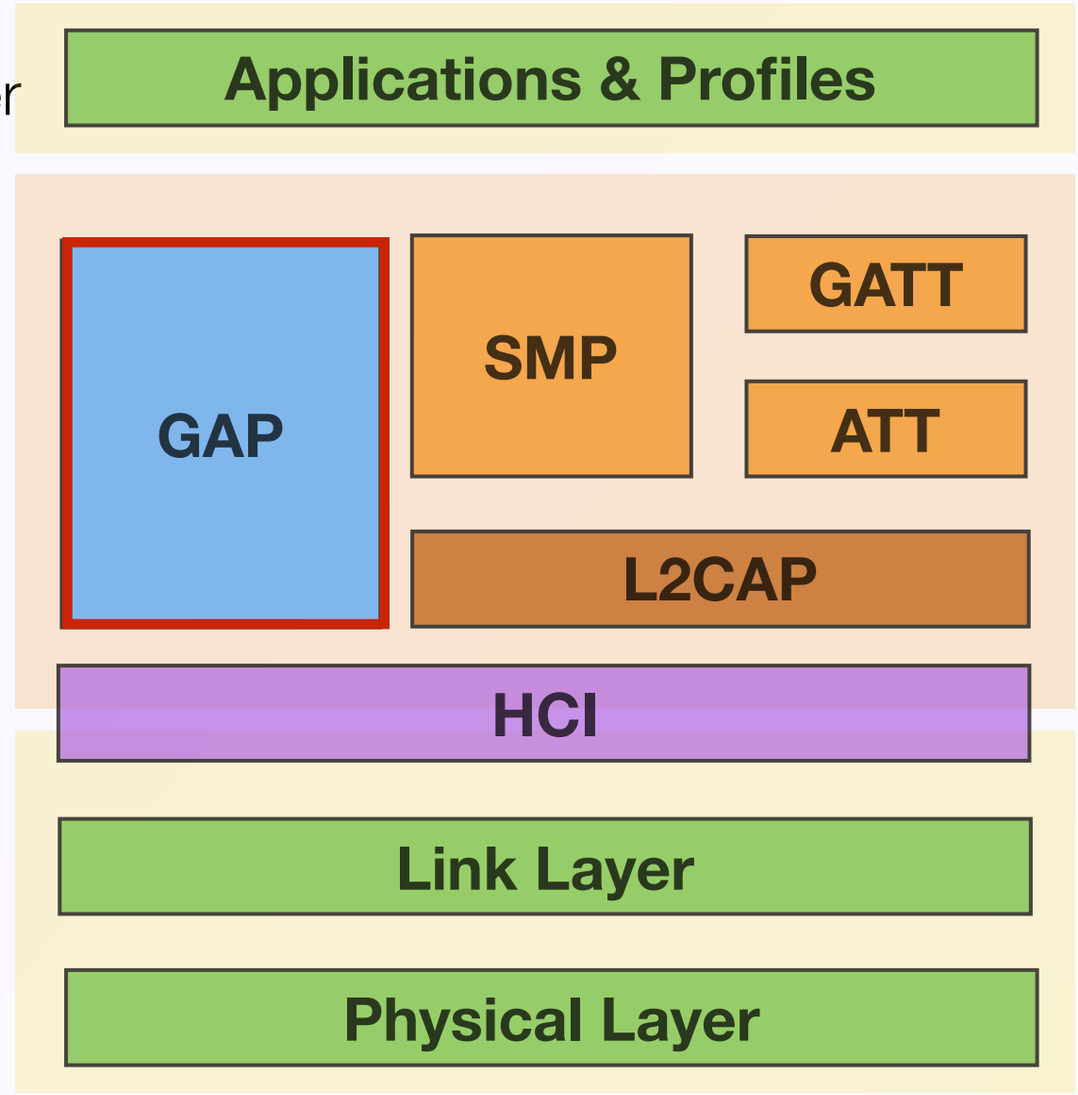
- Do research about “BLE Legacy Pairing”
- Do research about “BLE Secure Pairing”
- Do research about “Elliptic Curve Cryptography”

Generic Access Profile (GAP)

Generic Access Profile

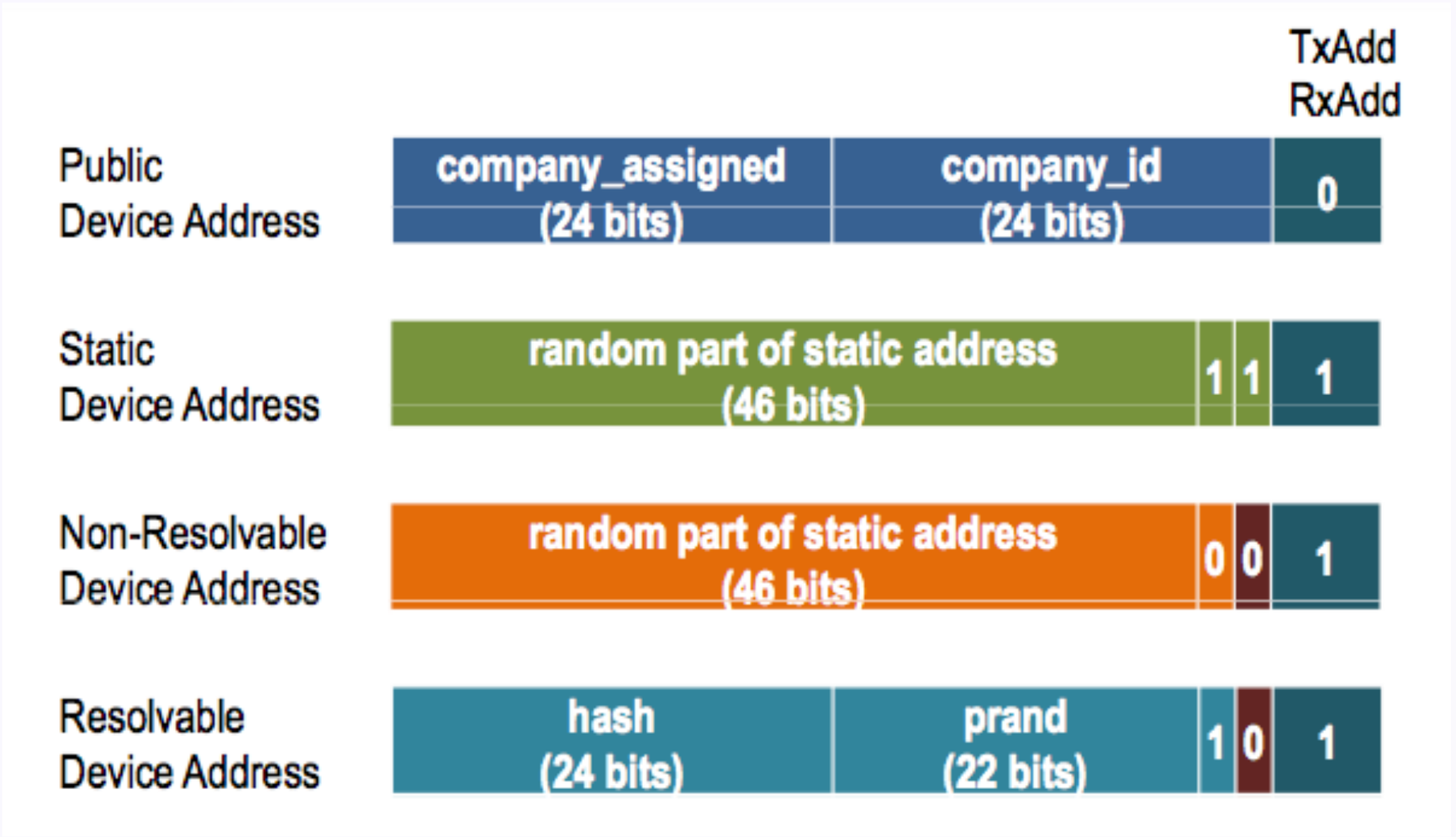
Connection-related procedures

- ➡ Interface between app and link layer
- ➡ Ex: Device discovery
- ➡ Ex: Link Establishment



Address Types

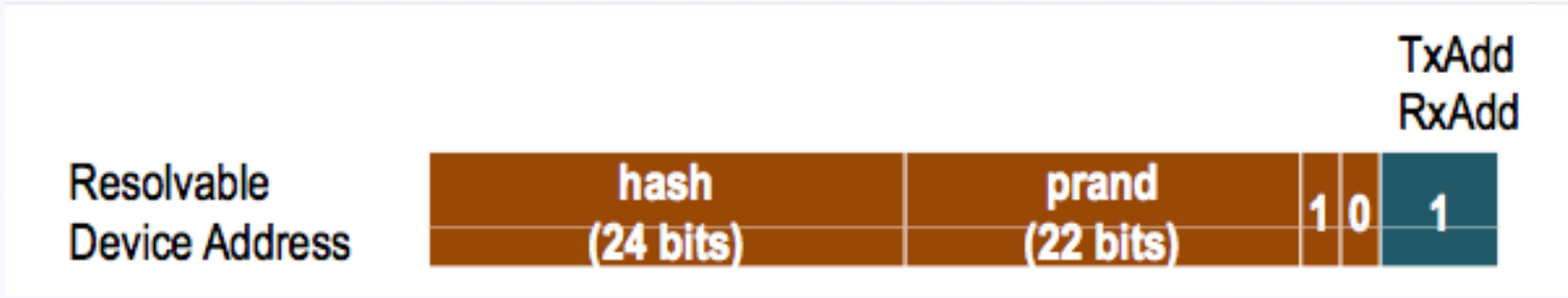
Random & Public Device Address



Question

Q1: Why Need Random Address?

Q2: How to Resolve Random Address?



GAP Roles

Master

Client

Can read/write data to Slave/Server

Central

Slave

Server

Has read/write data

Peripheral

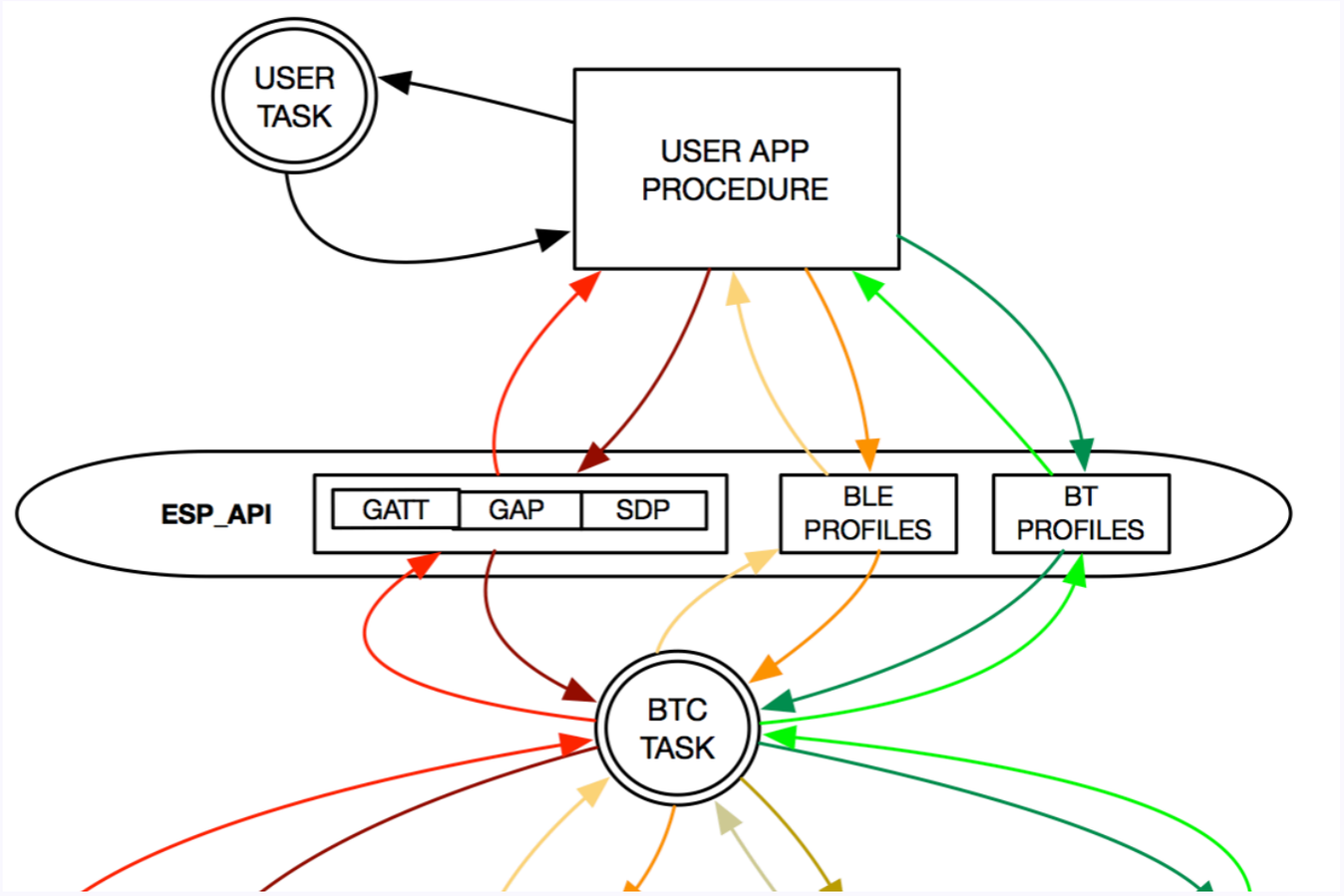
Can receive broadcast data

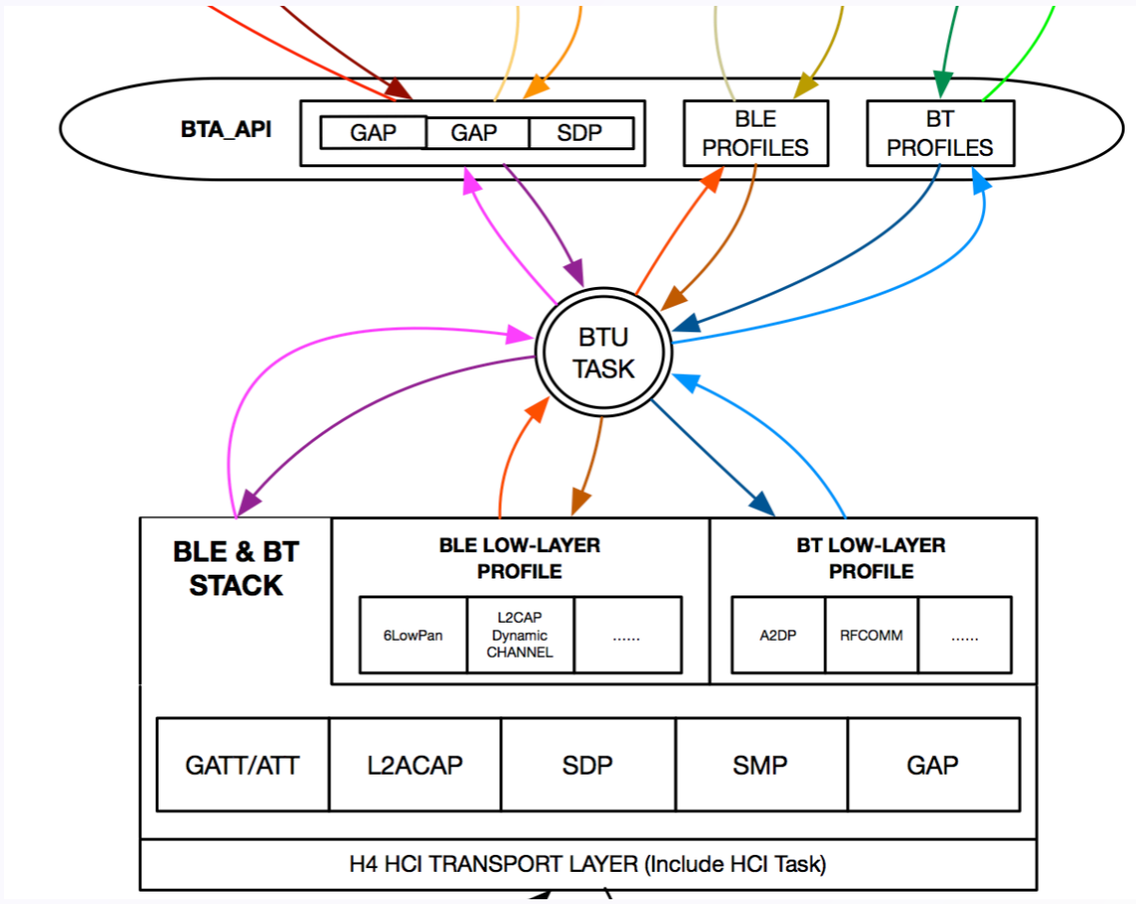
Observer

Has read-only broadcast data

Broadcaster

Architecture of ESP32 Stack





Bluetooth 5.0

Bluetooth 5.0 New Features

Bluetooth 5.0:

- ➡ Increase Max TX Power, Enhance RX Sensibility
- ➡ 2Mbps/250kbps
- ➡ Adv data, 31 bytes -> 255 bytes

BLE Mesh

Why we need mesh?

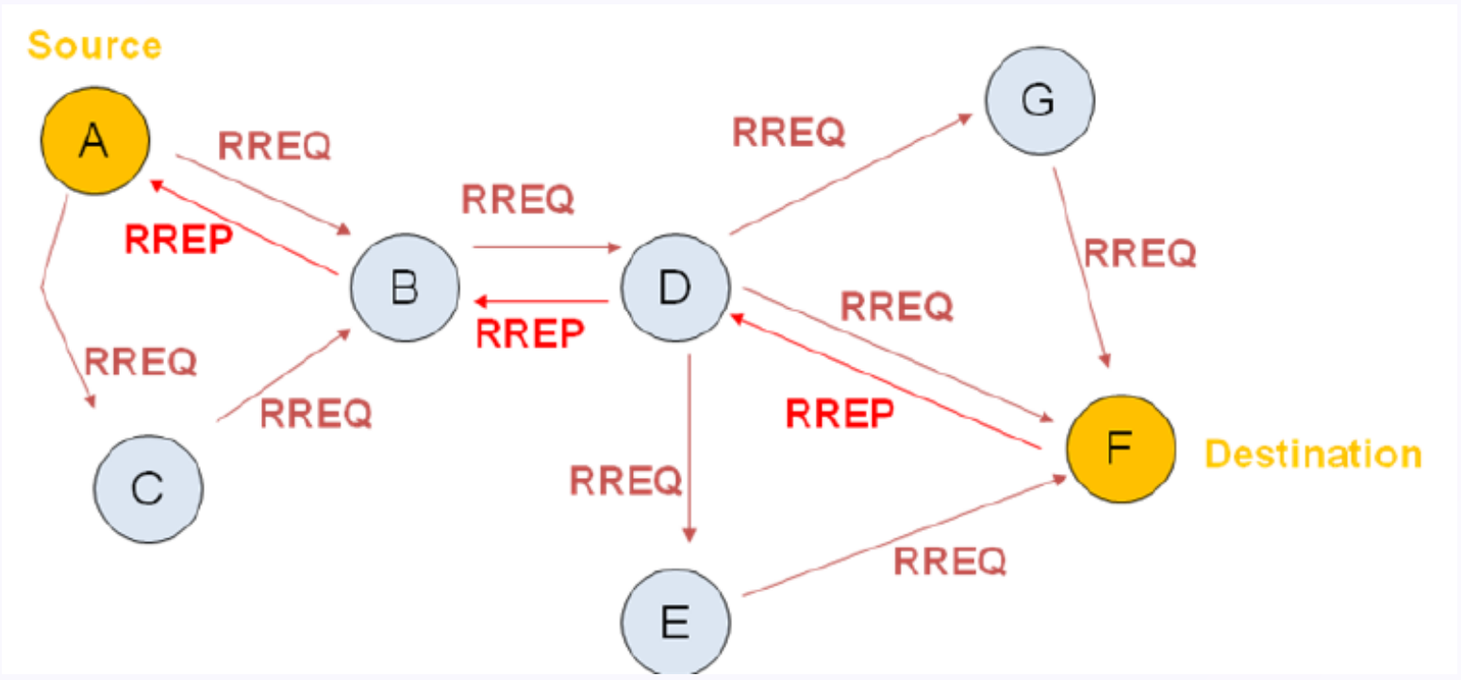
To Communicate:

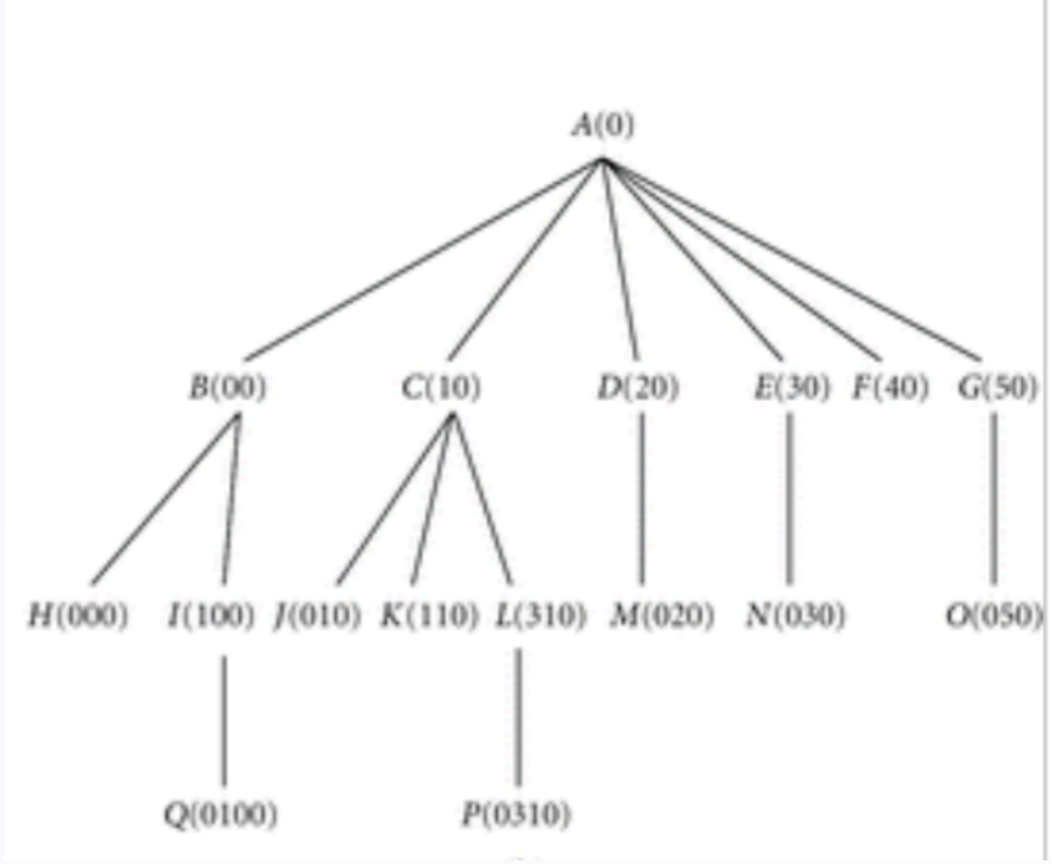
- ➡ With device which is far way
- ➡ If there are some physical obstacle
- ➡ With multiple devices at the same time
- ➡ With each others

How Zigbee mesh works?

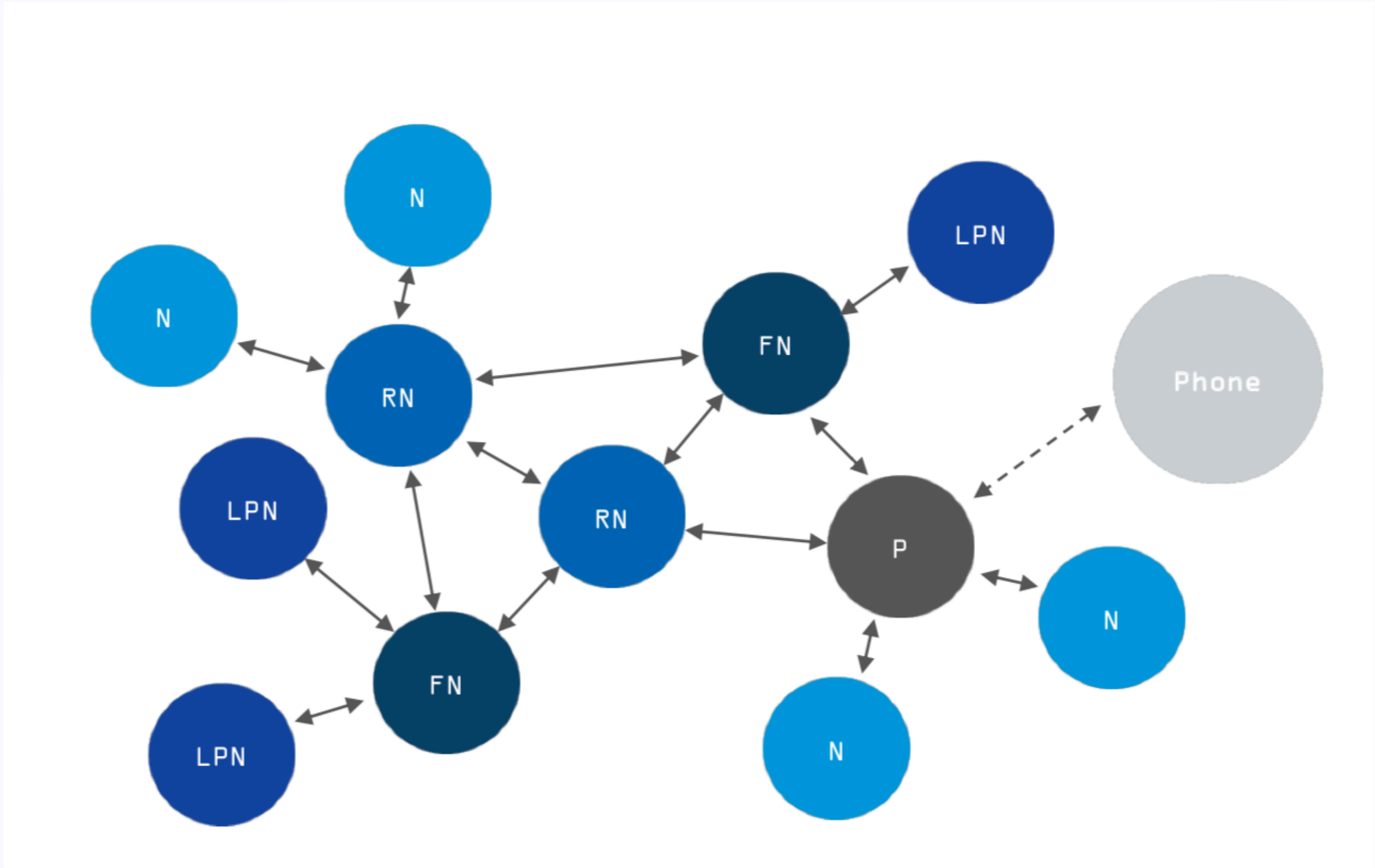
Different Zigbee Mesh Routing Protocols:

- ➡ Routing Mesh
- ➡ AODV: Ad hoc On-Demand Distance Vector Routing
- ➡ Cluster-Tree algorithm





What is BLE mesh?



Pros & Cons of Flooding Mesh

Pros:

- ➡ Don’t need to maintain the connection and routing map
- ➡ Network size is not limited by the device memory
- ➡ Easy to implement
- ➡ No “Single Point of Failure (SPoF)”

Cons:

- ➡ Collision may happen when traffic is high

BLE Mesh “Traffic Jam”

Reasons:

- ➡ Only 3 channels are used (37, 38, 39)
- ➡ Channel congestion in case of heavy traffic (e.g. dense network)
- ➡ Additional Wi-Fi or other 2.4 GHz signals interference

Results:

- ➡ The communication latency will increase
- ➡ Some messages may be lost

How to Solve “Traffic Jam” - 1

Message Cache:

- ➡ Drop the message which is already in the message cache
- ➡ Prevent the duplicated relay message between nodes

Ack Message:

- ➡ Fixed the “message lost” issue
- ➡ May increase the “Traffic Jam”

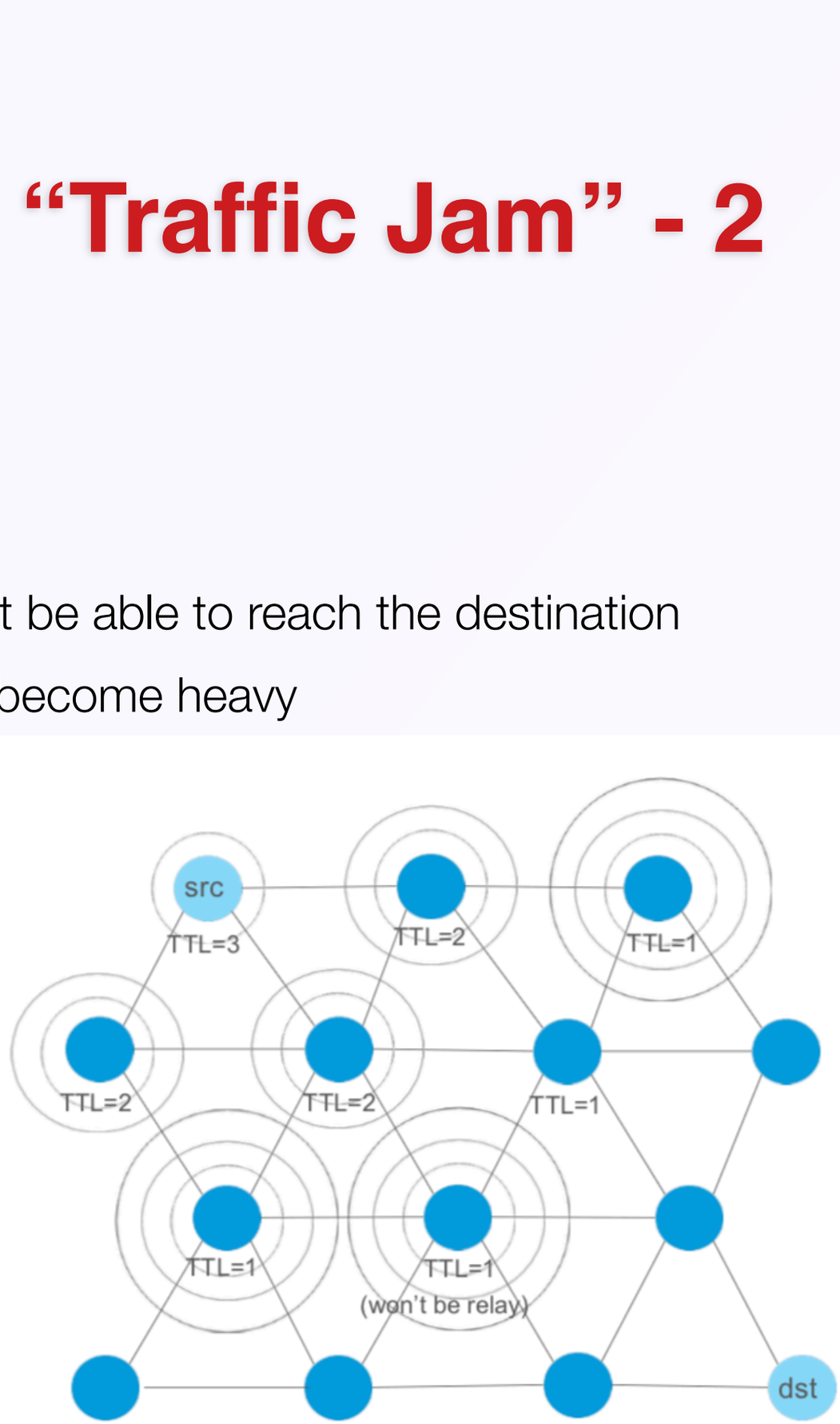
Scan Filter:

- ➡ Focus on Mesh type packets only
- ➡ Drop duplicate packets

How to Solve “Traffic Jam” - 2

Time To Live:

- ➡ If TTL <= 1, message won’t be relayed
- ➡ If TTL is too small, the message will not be able to reach the destination
- ➡ If TTL is too large, the mesh traffic will become heavy
- ➡ “Heartbeats” message can do help



If TTL is too small, the message cannot reach destination

How to Solve “Traffic Jam” - 3

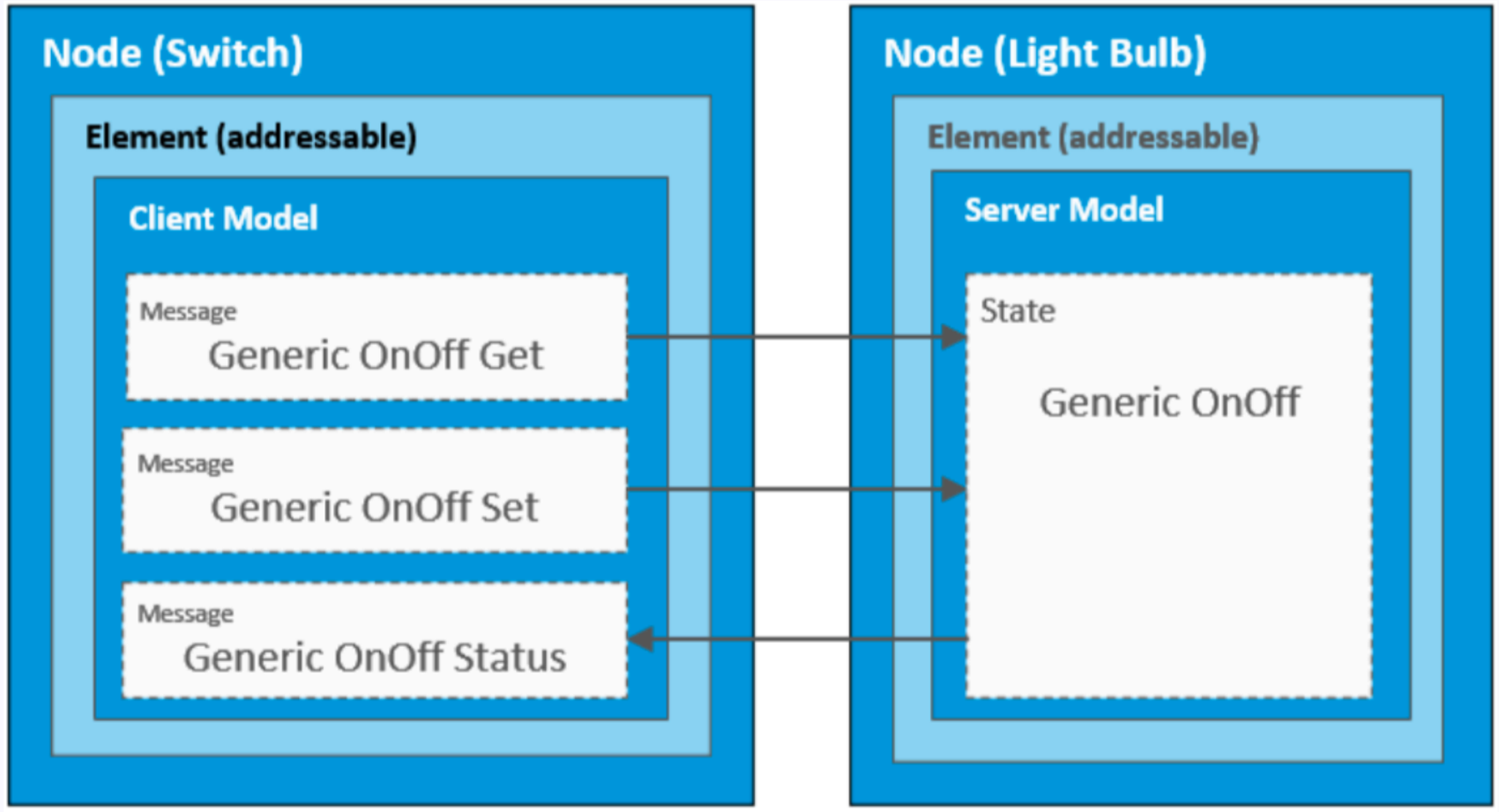
Different Node Features:

- ➡ Not all the nodes relay messages

How Device “Talk” to Each Other - 1

Client-Server Model:

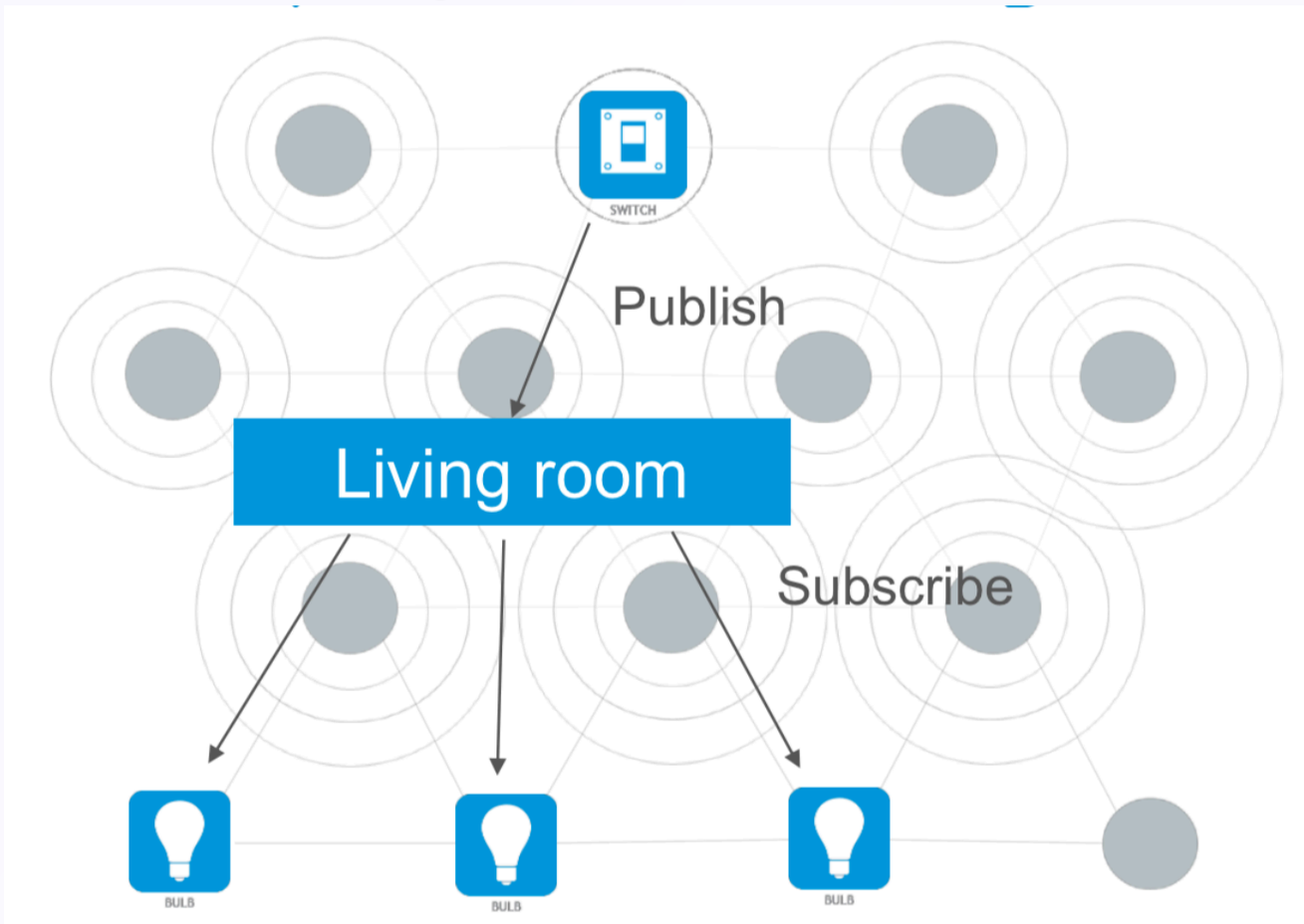
- ➡ Use Publish/Subscribe to exchange information (Server -> Client)
- ➡ Use Get/Set (Client -> Server)



How Device “Talk” to Each Other - 2

Group addressing:

- ➡ Publish to a group address
- ➡ Subscribers receives the message
- ➡ Easier to replace nodes (don’t require to re-config other nodes)



Power Save - 1

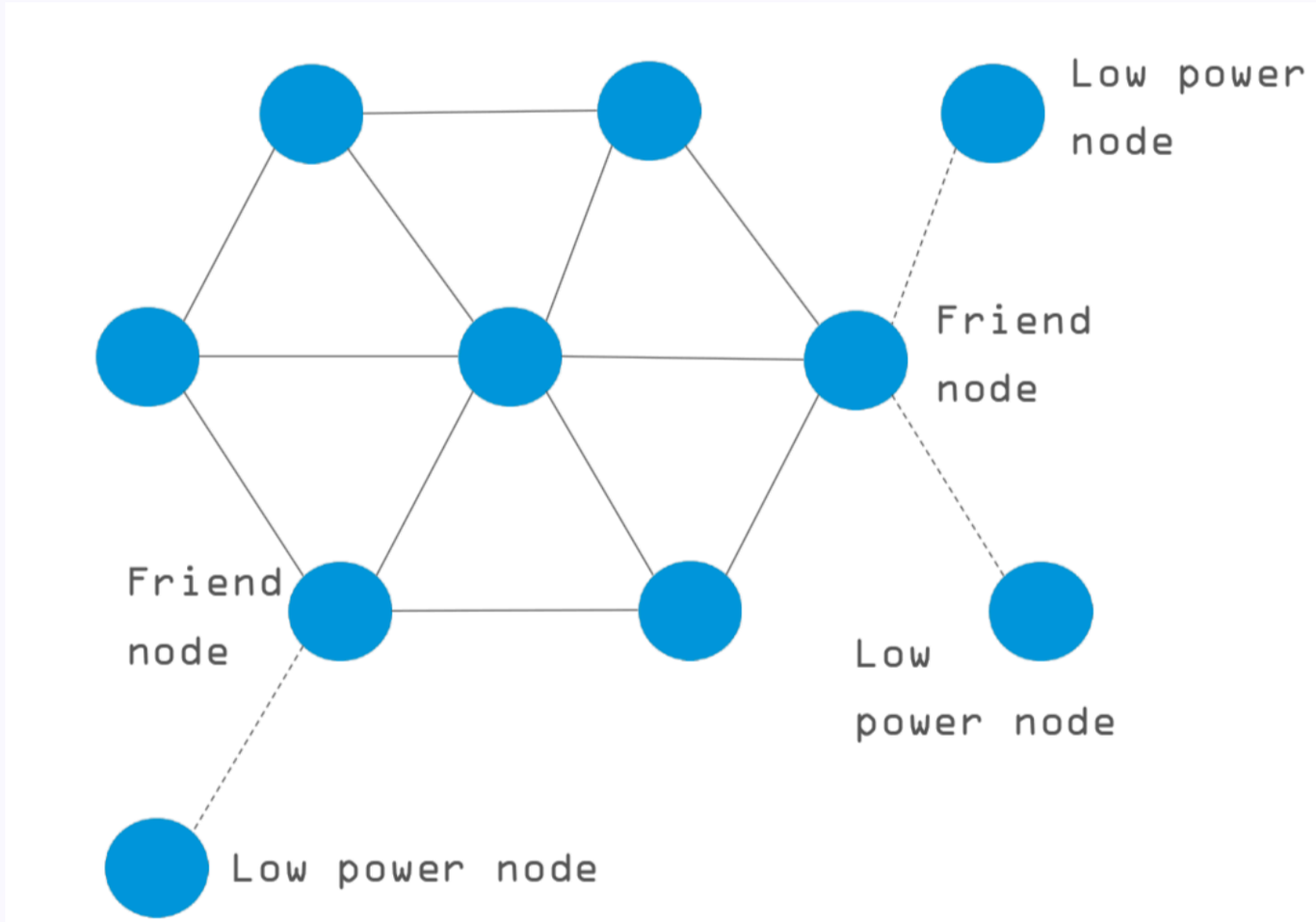
Normal Nodes:

- ➡ Nodes continuously in RX
- ➡ Normally mains powered

Power Save - 2

Low Power Nodes:

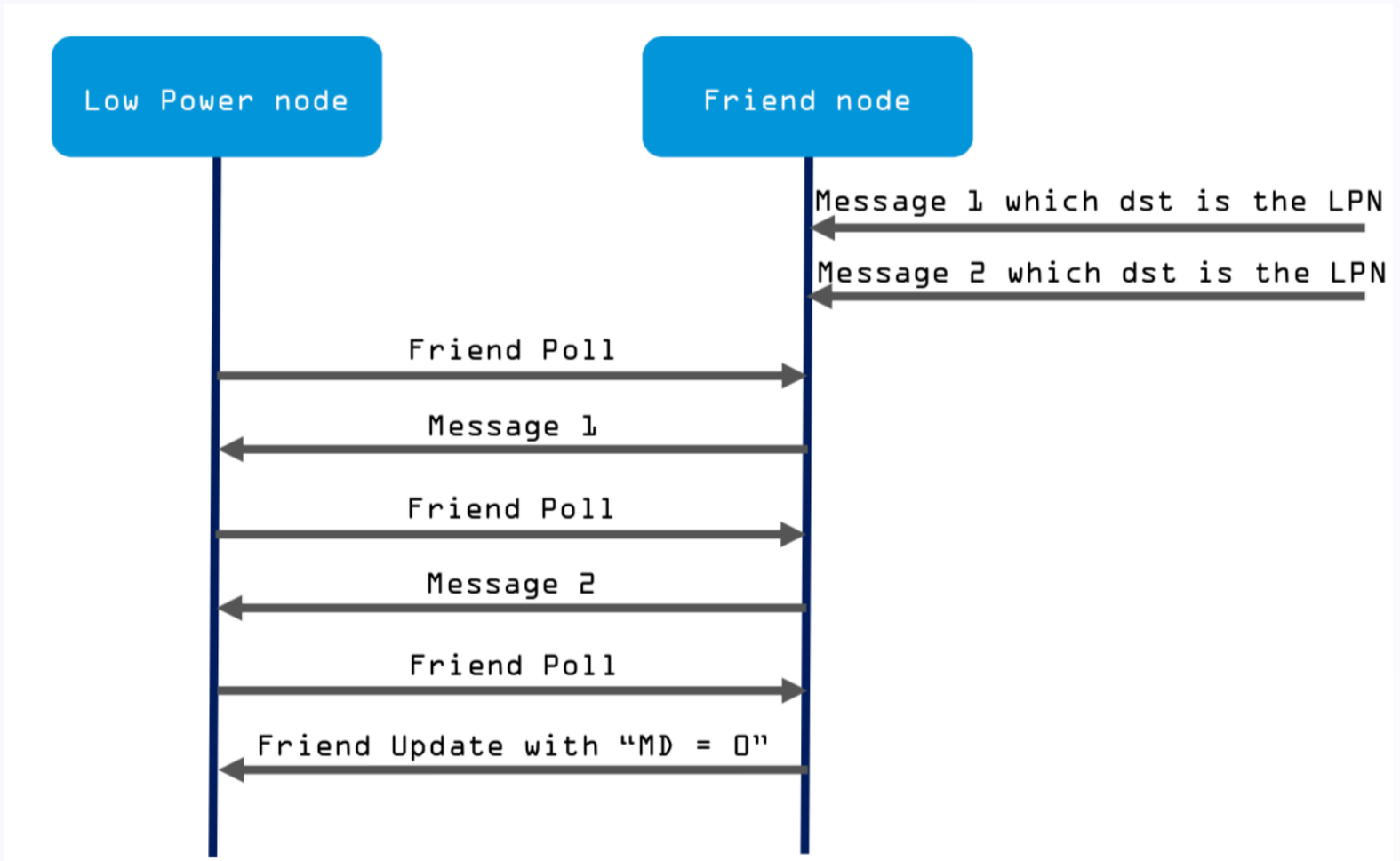
- ➡ Low power nodes sleep
- ➡ Friend caches and forwards



Power Save - 3

How friendship works:

- ➡ Low power nodes sends friend poll when it wakes up
- ➡ Friend node sends the queued message



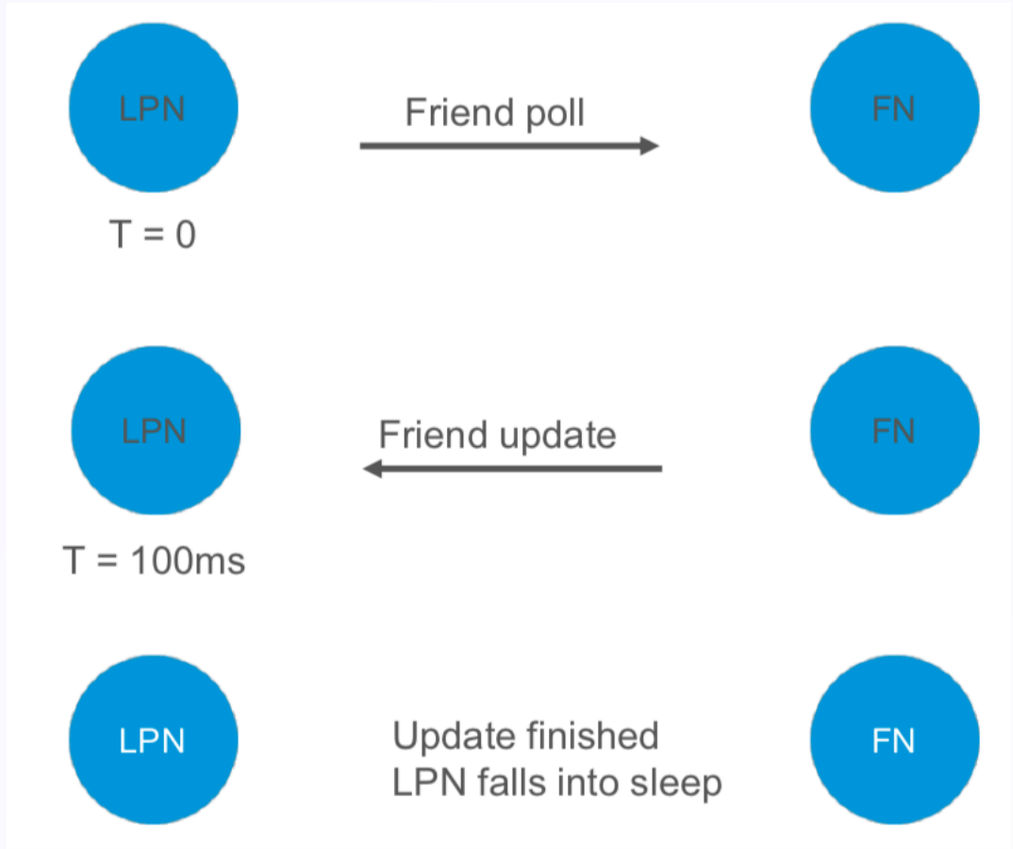
Power Save - 4

How can LPN save power:

- ➡ If a LPN sends a “friend poll” every 15 minutes
- ➡ If FN responds after 100ms
- ➡ The duty cycle for a day will be
  0.101 * (60/15) * 24 ≈ 10 sec

Question: Is it possible to get

a lower power consumption?



Built-in Security - 1

Mesh Security:

- ➡ Security mechanism is not optional, is mandatory
- ➡ All traffic encrypted with AES-128

Question: What are the security risks for a Mesh network ?

Built-in Security - 2

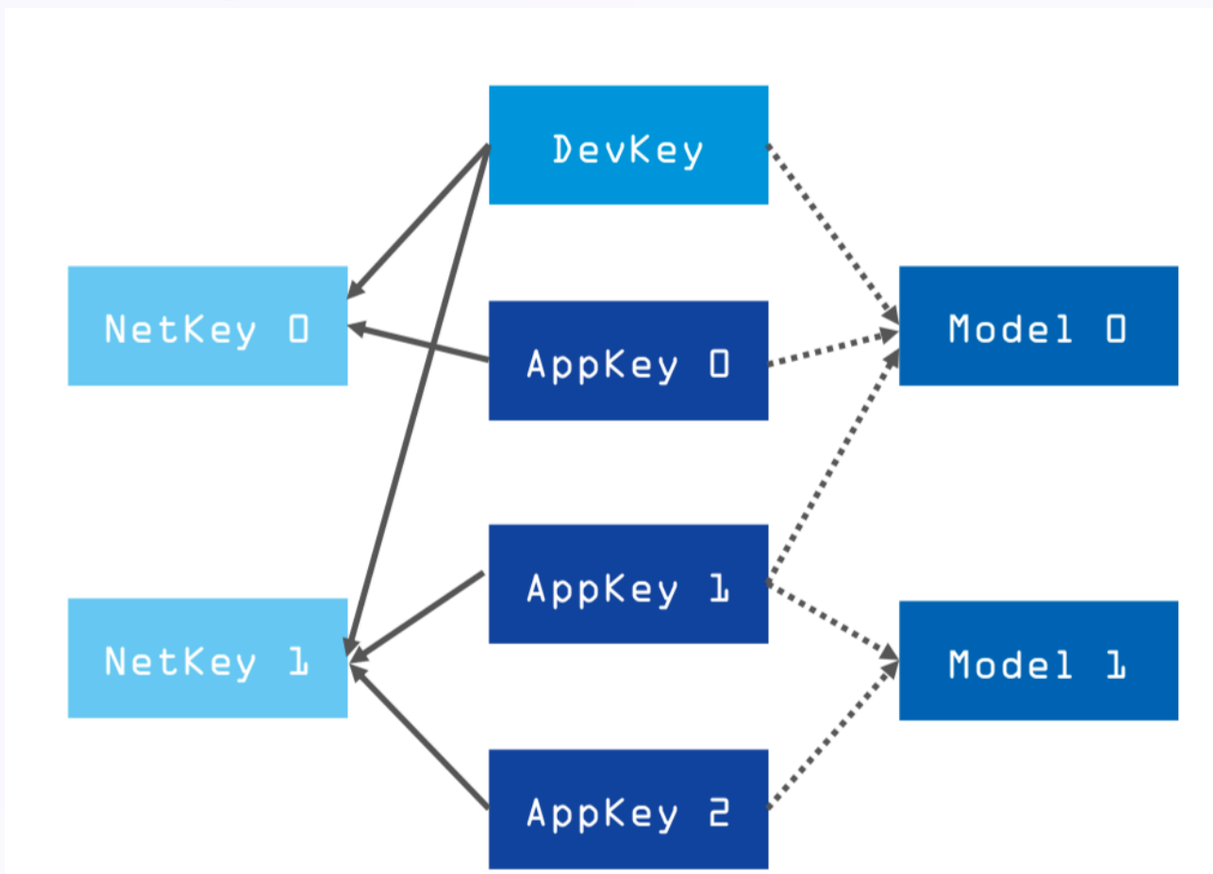
Provisioning:

- ➡ Not every device can join a mesh network
- ➡ Smartphone acts as a provisioner

Built-in Security - 3

Security Keys:

- ➡ Network Key: Relay
- ➡ Application Key: Light bulb will not unlock the door
- ➡ Device Key: Only the provisioner can config the nodes



Built-in Security - 4

Replay attack:

- ➡ Sequence number
- ➡ IV update

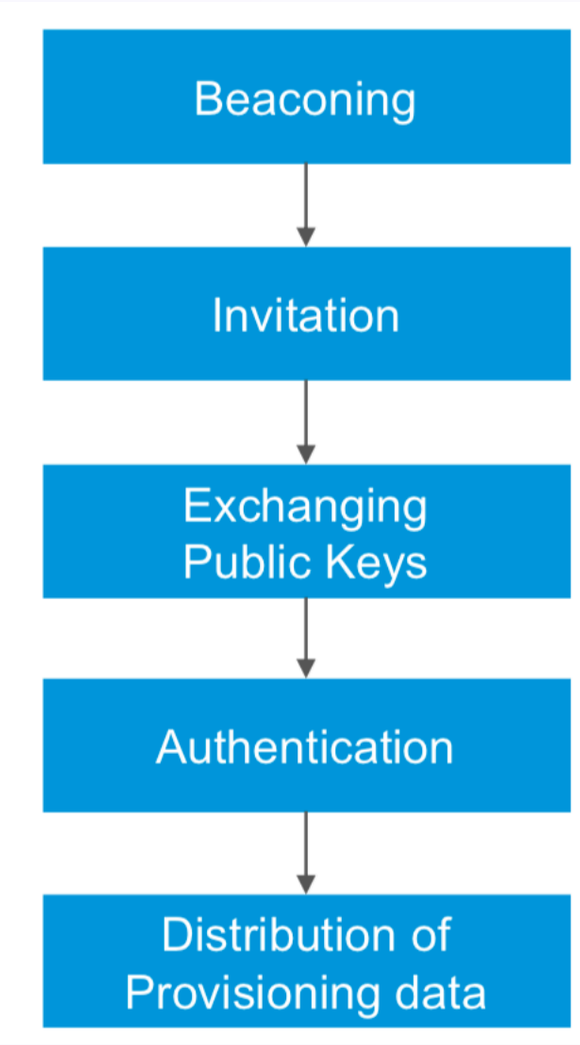
Trash-can attack:

- ➡ Key refresh procedure
- ➡ Device Key
- ➡ Black list

Provisioning

Procedure:

- ➡ Unprovisioned device should advertising
- ➡ Provisioner listens for un-provisioned devices
- ➡ Initiates the process with interested device
- ➡ Asymmetric encryption
- ➡ Input/Output capability
- ➡ Distribute network key, device key, unicast address, etc



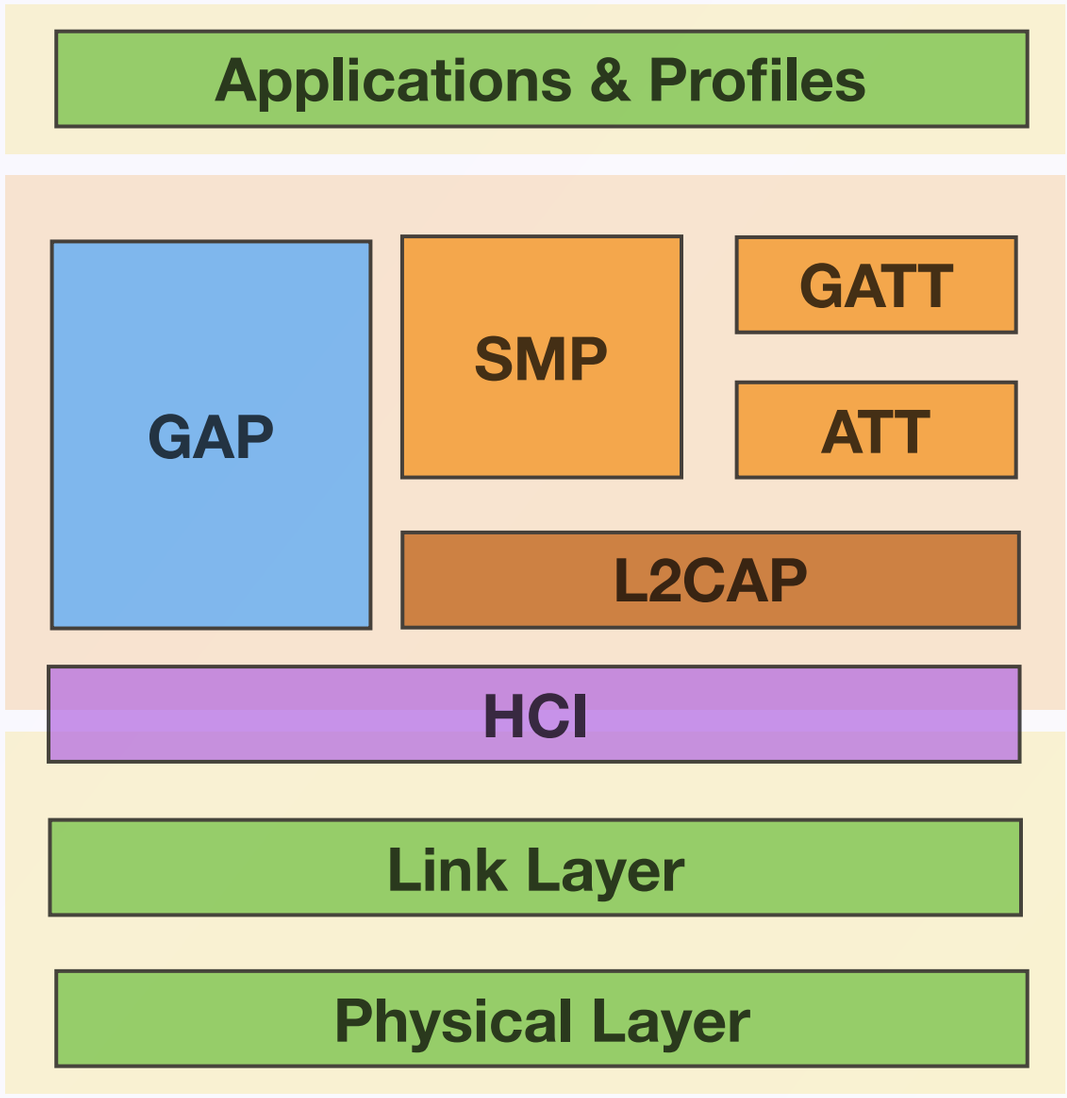
Review

BLE Introduction

- ➡ History
- ➡ Applications

BLE Protocol

- ➡ Controller: PHY, LL
- ➡ HCI
- ➡ Host: L2CAP, SMP, GATT, ATT, GAP



Review

BLE Mesh

- ➡ Introduction
- ➡ “Traffic Jam”
- ➡ Power Save
- ➡ Built-in Security
- ➡ Provisioning

Reference

- Bluetooth Specification Version 5.0
- TI BLE Training Doc
- Nordic BLE Training Doc
- CSR BLE Training Doc
- www.bluetooth.com

Thanks