Internet-of-Things (IoT)

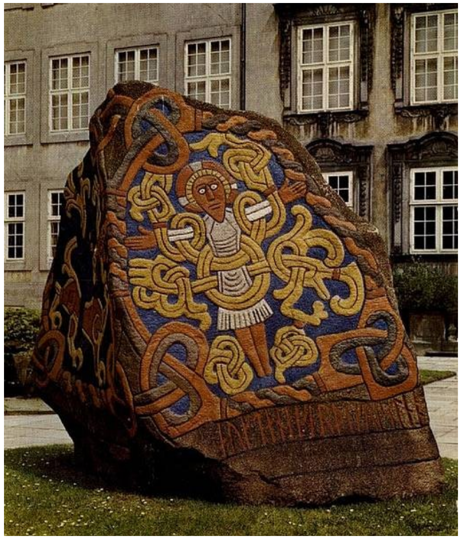
LAB COMPONENT

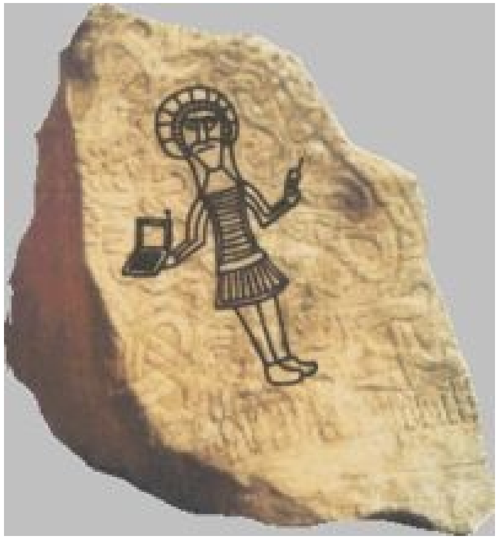
Bluetooth

- Basic idea
  - Universal radio interface for **ad-hoc wireless connectivity**
  - Interconnecting **computer** and **peripherals, handheld devices, PDAs, cell phones** – replacement of IrDA (**I**nfra**r**ed **D**ata **A**ssociation)
  - Embedded in other devices, very cheap
  - Short range (10m), low power consumption, license-free 2.45 GHz ISM
  - Voice and data transmission

Bluetooth

- History
  - 1994: Ericsson worked on "MC-link" project
  - Later renamed to Bluetooth: Harald "Blåtand" Gormsen [son of Gorm], King of Denmark in the 10th century; united Denmark and spread Christianity
  - 1998: foundation of Bluetooth SIG (Special Interest Group): www.bluetooth.org
  - 2001: first consumer products for mass market, spec. version 1.1 released
  - 2005: 250 million chips
  - 2018: 4 billion chips





Key Characteristics

- **2.4 GHz** ISM band, **79 RF channels**
  - Channel 0: 2402 MHz … channel 78: 2480 MHz

- **Frequency hopping**
  - 1600 hops/s
  - Hopping sequence in a pseudo random fashion, determined by a master

- **Two types of traffic:**
  - **Voice link – SCO** (Synchronous Connection Oriented)
  - **Data link – ACL** (Asynchronous Connection Less)

Topology: Piconet & Scatternet

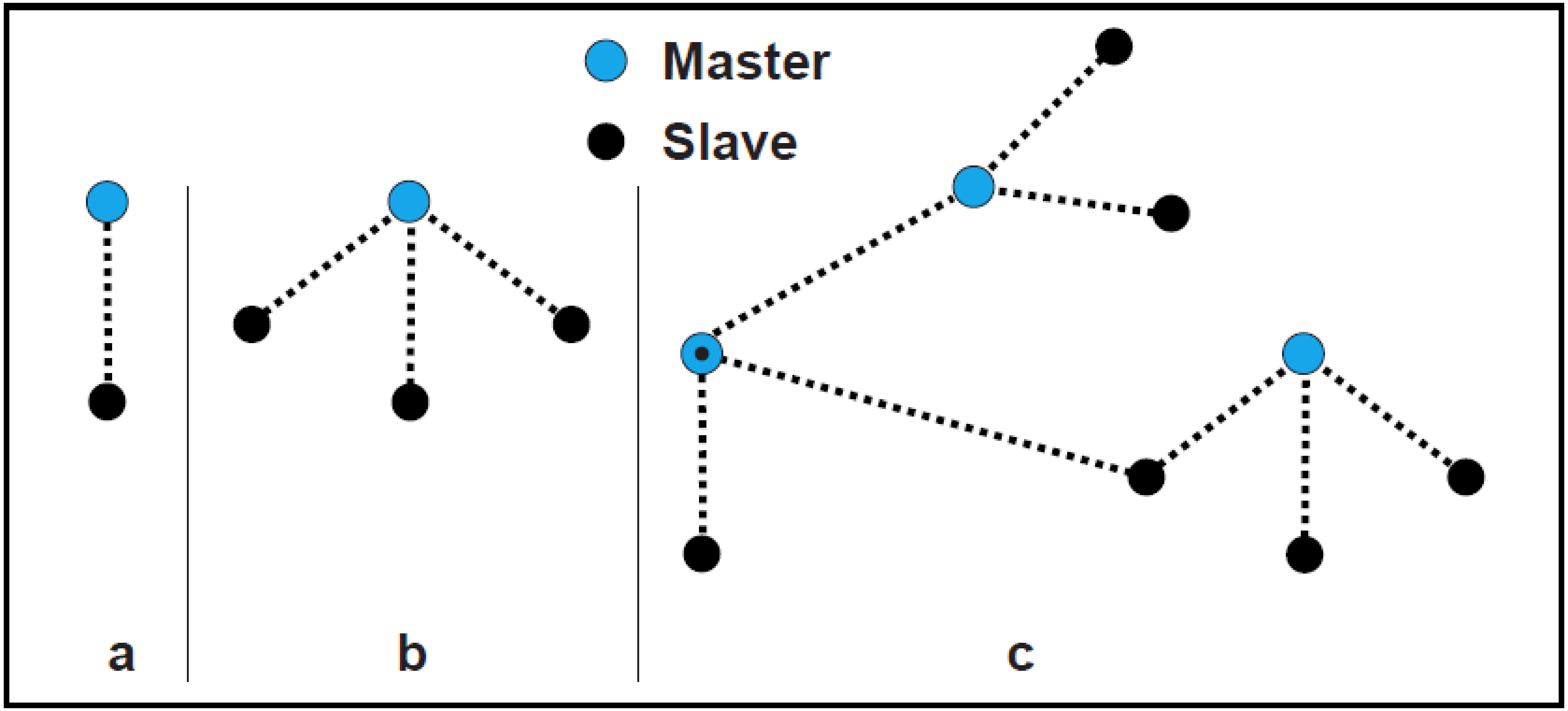
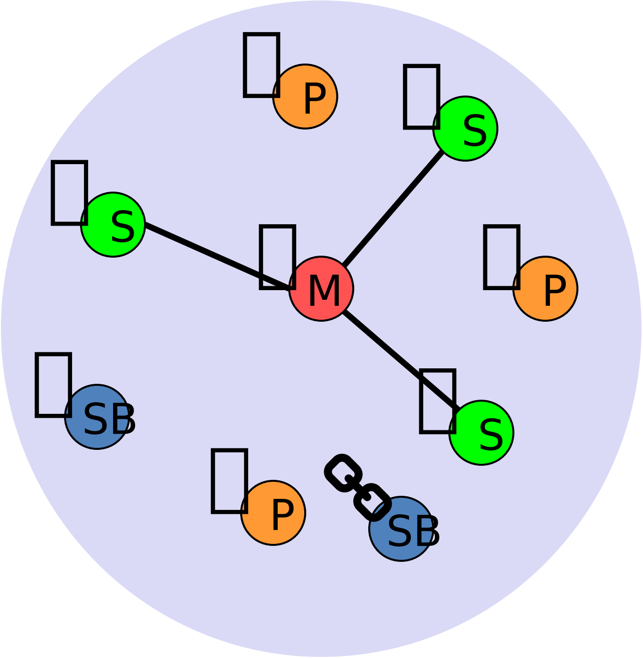


Figure 1.1: Piconets with a single slave operation (a), a multi-slave operation (b) and a scatternet operation (c).

Bluetooth Piconet

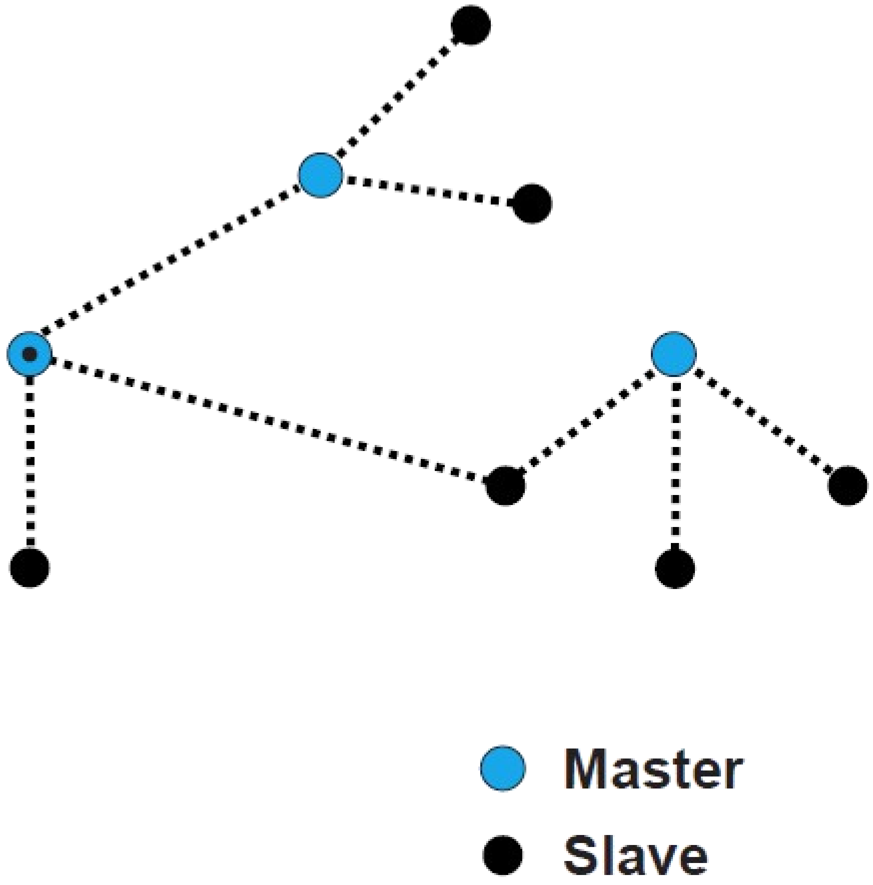


M=Master P=Parked
S=Slave SB=Standby

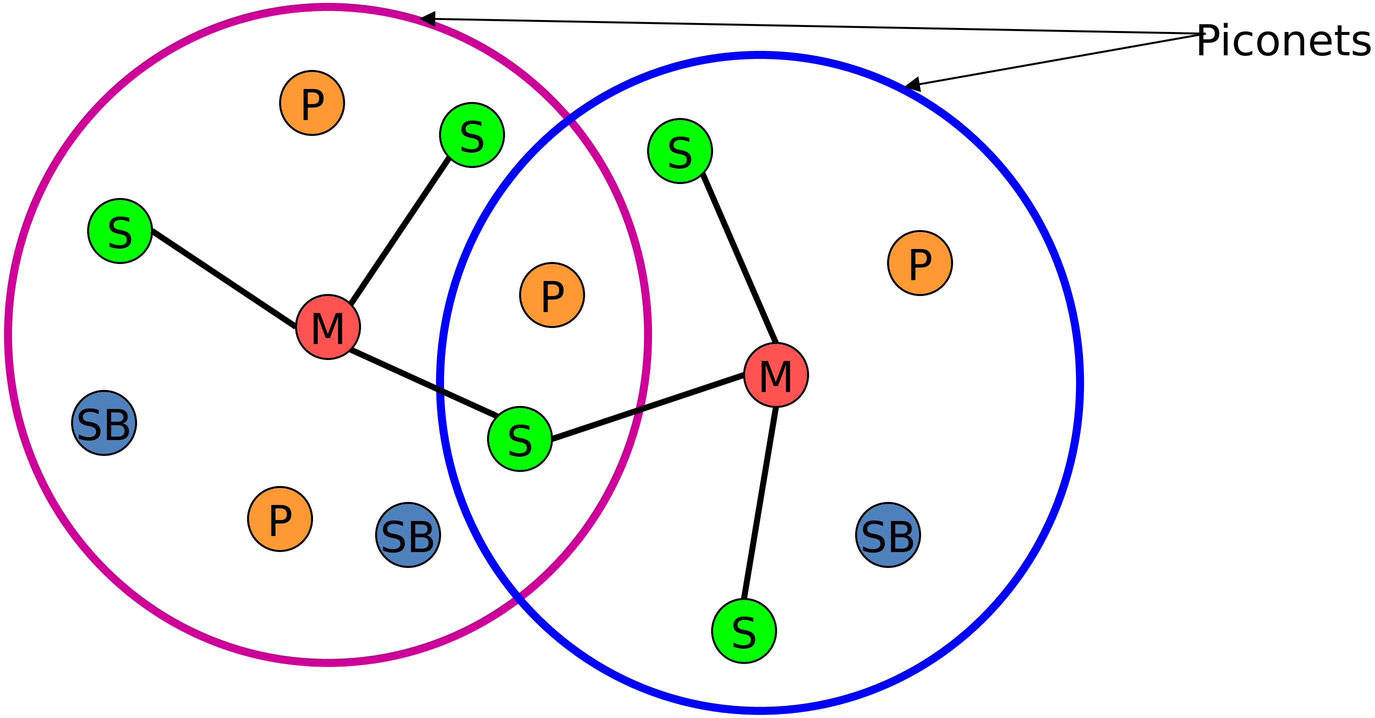
- Master gives slaves its **clock** and **device ID**
  - Hopping pattern: determined by device ID (48 bit, unique worldwide)
  - Pseudo random number generator (device ID + clock)
- **Addressing**
  - Active Member Address (AMA, 3 bit)
  - Parked Member Address (PMA, 8 bit)
  - Standby Member (no address)

Bluetooth Scatternet

- Each piconet has one master, but slaves can participate in different piconets on a **time-division multiplex** basis
- A master in one piconet can be a slave in other piconets
- Each piconet has its own hopping sequence

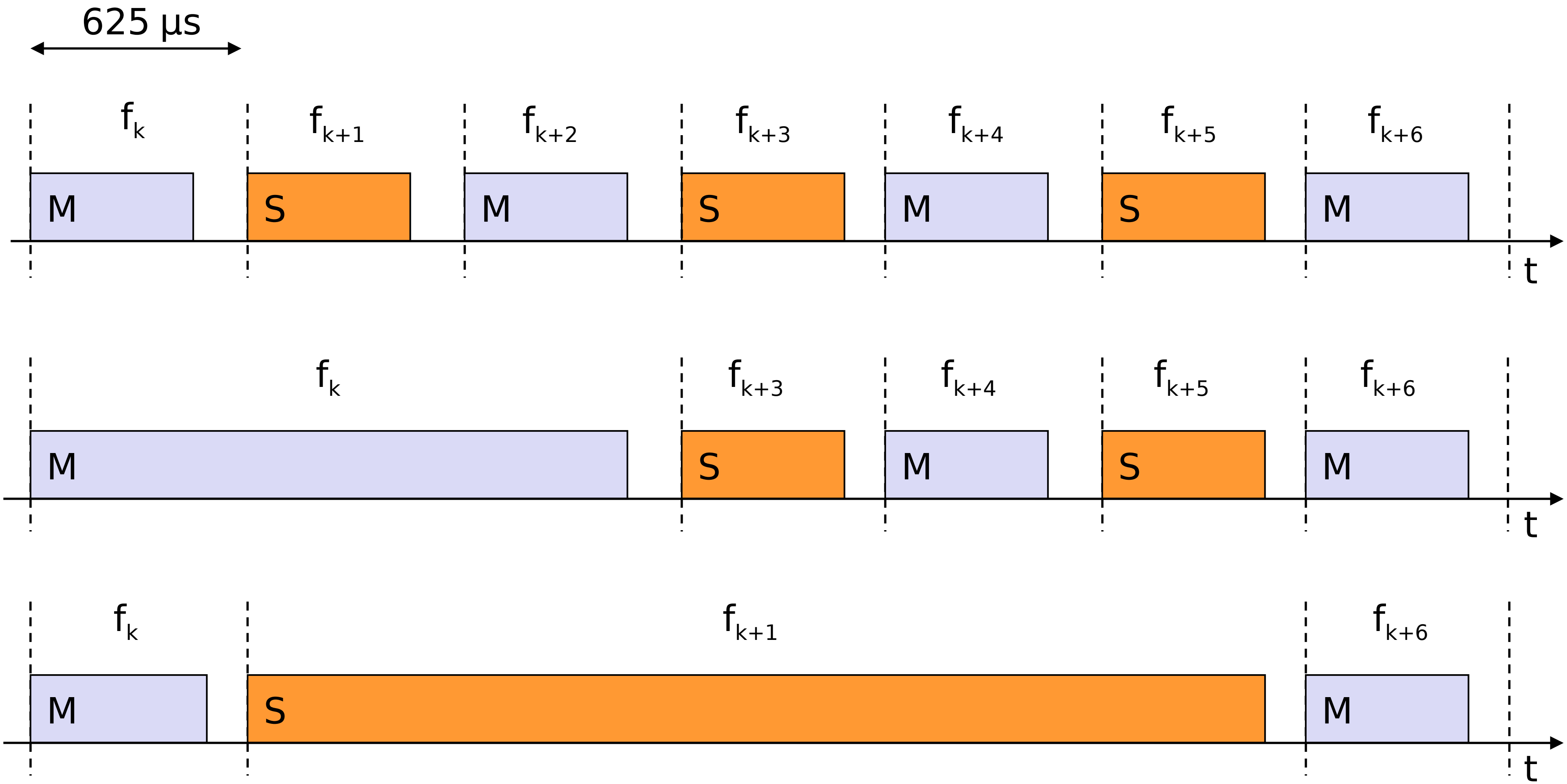


Bluetooth Scatternet



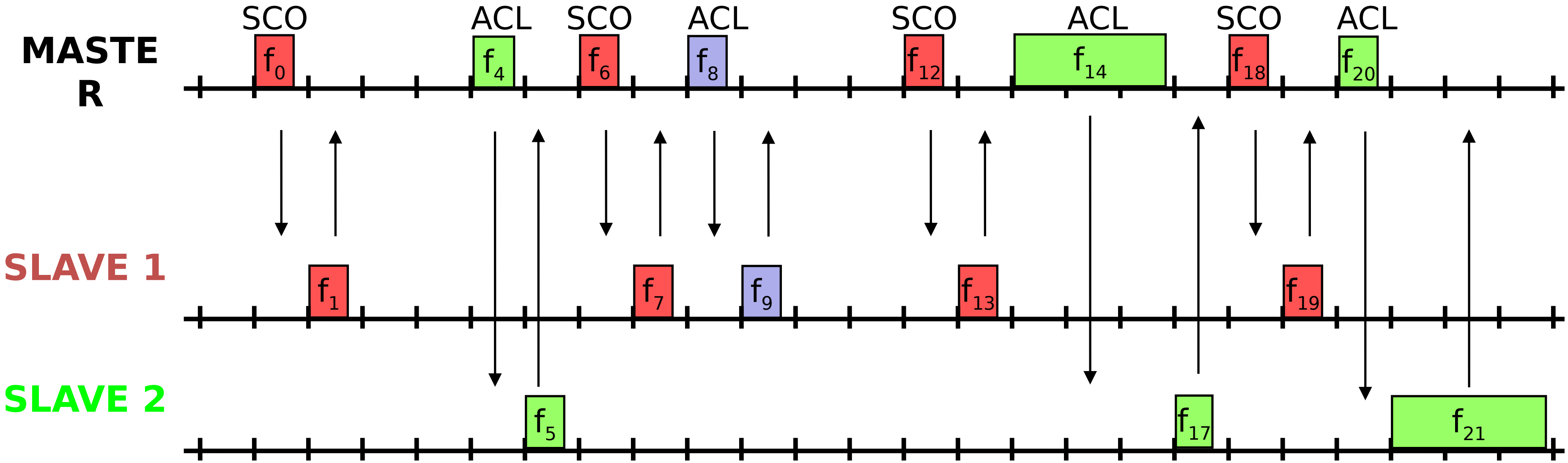
M=Master
S=Slave
P=Parked
SB=Standby

Communication

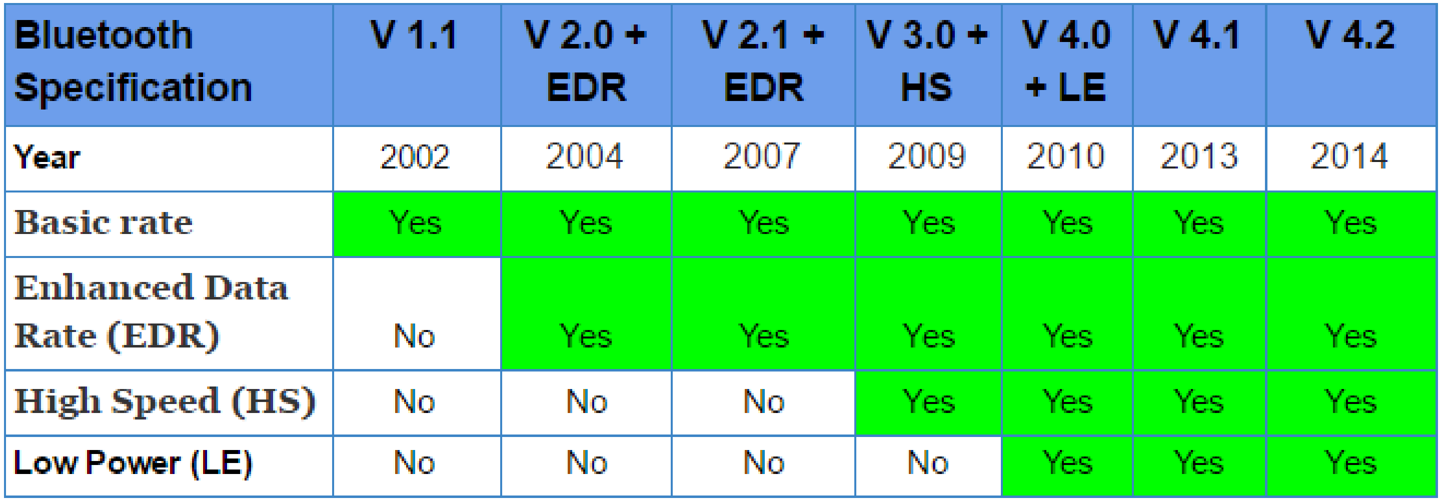


Communication Example

- Polling-based time division multiplex
  - 625µs slots, master polls slaves
- SCO (Synchronous Connection Oriented) – Voice
- ACL (Asynchronous ConnectionLess) – Data
  - Variable packet size (1, 3, 5 slots)



Bluetooth Versions



Bluetooth Specification	V 1.1	V 2.0 + EDR	V 2.1 + EDR	V 3.0 + HS	V 4.0 + LE	V 4.1	V 4.2
Year	2002	2004	2007	2009	2010	2013	2014
Basic rate	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Enhanced Data Rate (EDR)	No	Yes	Yes	Yes	Yes	Yes	Yes
High Speed (HS)	No	No	No	Yes	Yes	Yes	Yes
Low Power (LE)	No	No	No	No	Yes	Yes	Yes

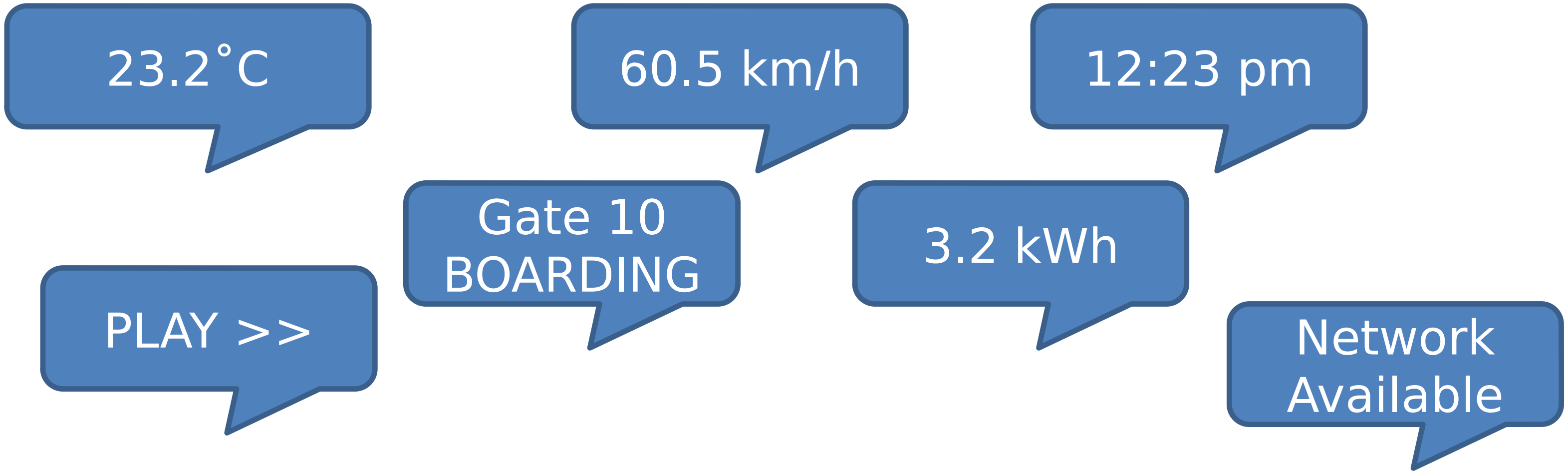
Bluetooth Low Energy (BLE)

- Bluetooth low energy is a new, open, short range radio technology
  - Blank sheet of paper design
  - Different to Bluetooth classic
  - Optimized for ultra low power
  - Enable coin cell battery use cases
    - < 20mA peak current
    - < 5uA average current

BLE Basic Concepts

- Everything is optimized for lowest power consumption
  - Short packets reduce TX (transmit) peak current
  - Short packets reduce RX (receive) time
  - Fewer RF channels to improve discovery and connection time
  - Simple state machine
  - Single protocol
  - …

“Exposing State” (IoT)



23.2˚C

60.5 km/h

12:23 pm

Gate 10 BOARDING

3.2 kWh

PLAY >>

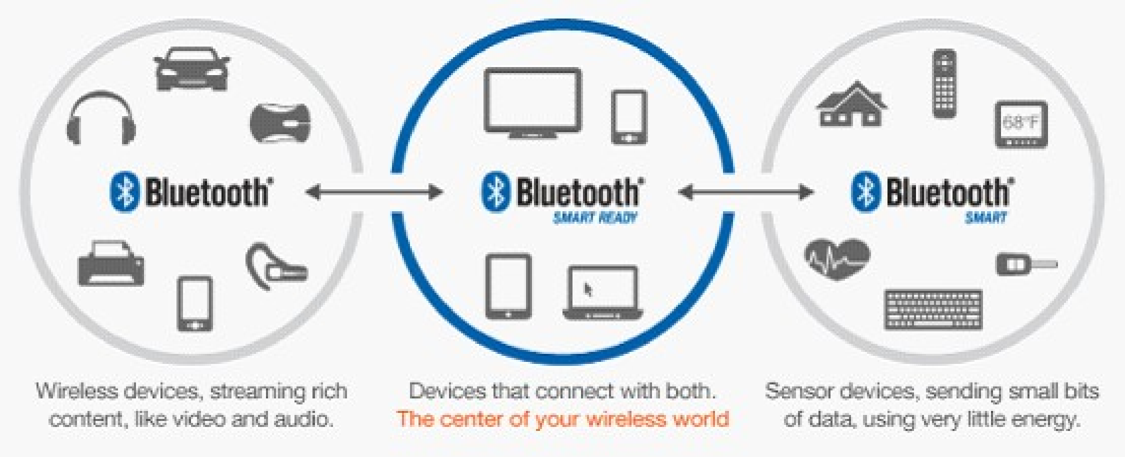
Network Available

- It’s good at small, discrete data transfers
- Data can triggered by local events
- Data can be read at any time by a client

BLE Device Modes

- Dual Mode
  - Bluetooth Classic and LE
  - Used anywhere BT Classic is used today

- Single Mode
  - Implements only Bluetooth low energy
  - Will be used in new devices / applications



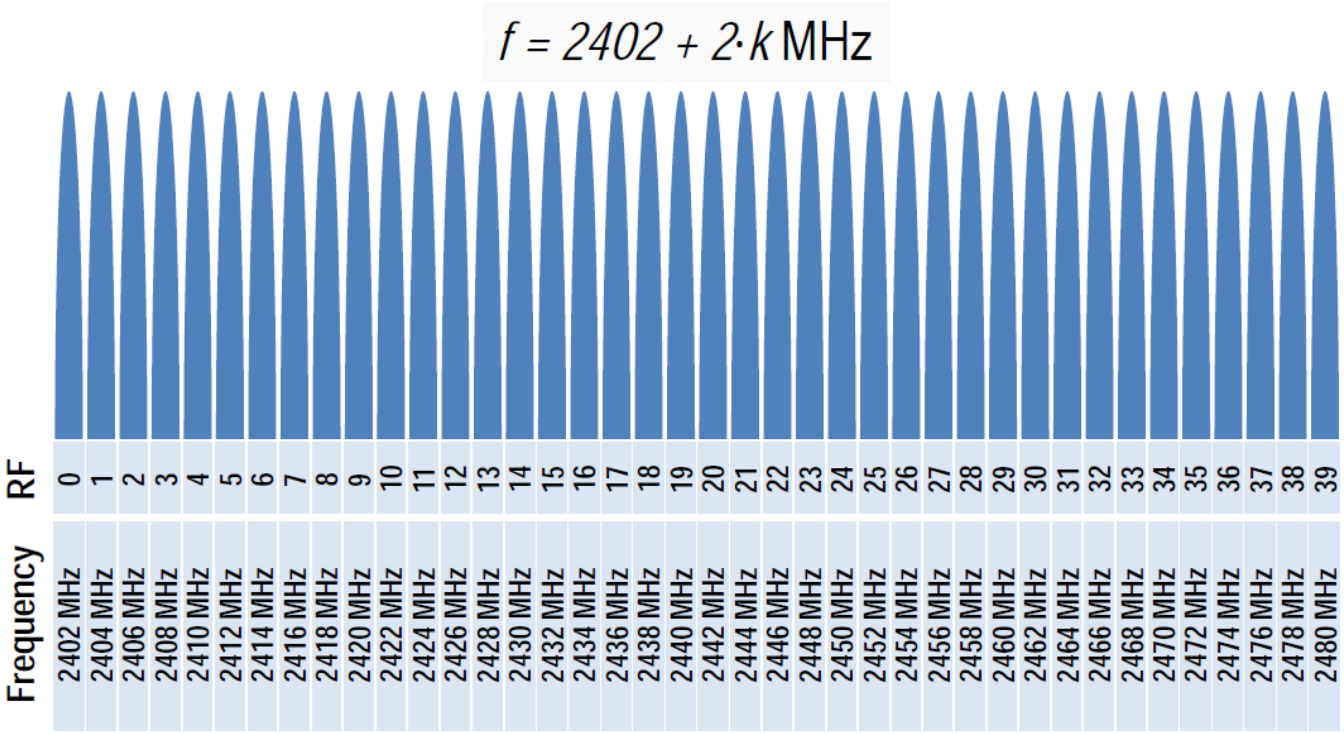
Wireless devices, streaming rich content, like video and audio.

Devices that connect with both. The center of your wireless world

Sensor devices, sending small bits of data, using very little energy.

BLE Physical Layer

- 2.4 GHz ISM band
- 40 Channels on 2 MHz spacing



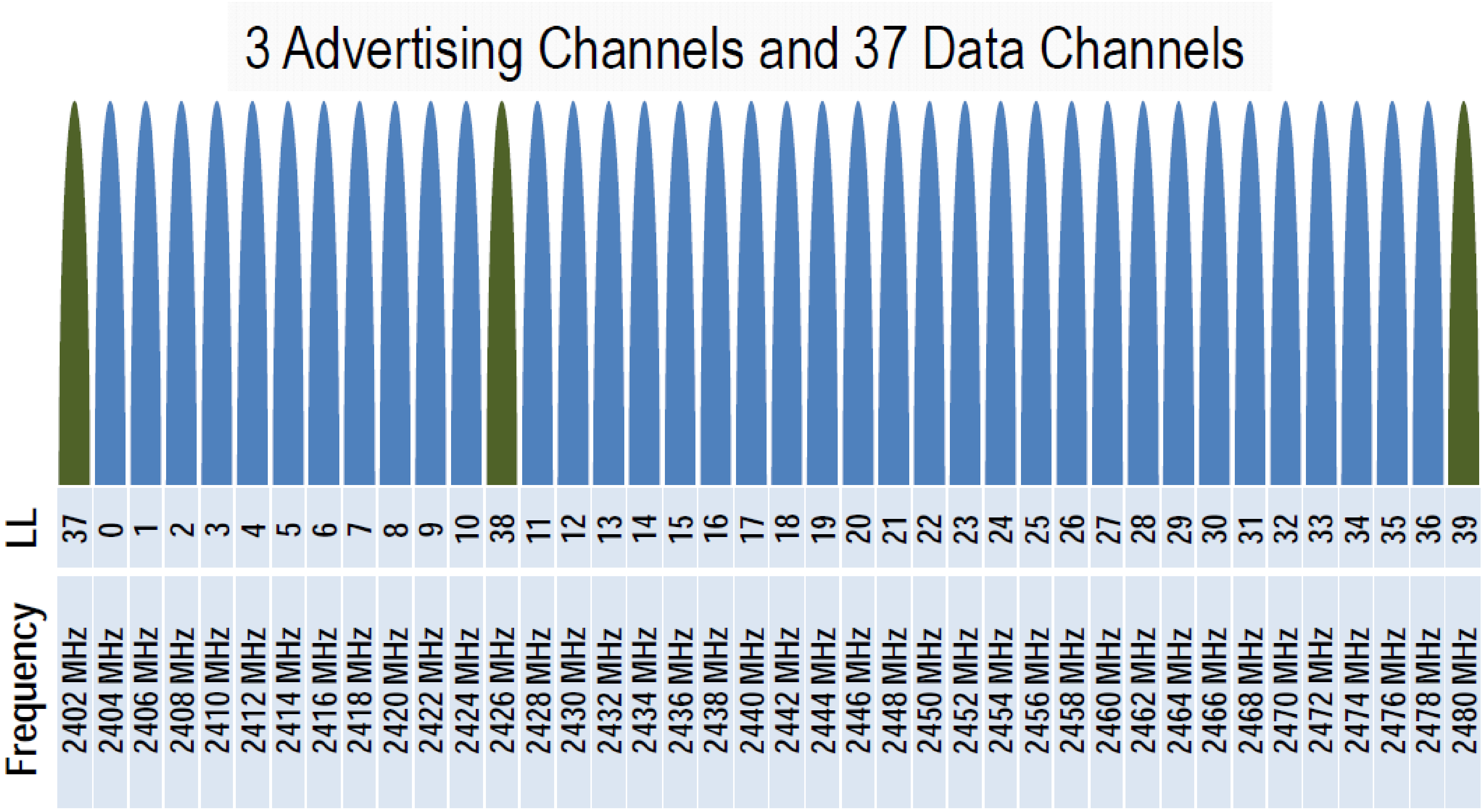
$$f = 2402 + 2 \cdot k \text{ MHz}$$

RF	Frequency
0	2402 MHz
1	2404 MHz
2	2406 MHz
3	2408 MHz
4	2410 MHz
5	2412 MHz
6	2414 MHz
7	2416 MHz
8	2418 MHz
9	2420 MHz
10	2422 MHz
11	2424 MHz
12	2426 MHz
13	2428 MHz
14	2430 MHz
15	2432 MHz
16	2434 MHz
17	2436 MHz
18	2438 MHz
19	2440 MHz
20	2442 MHz
21	2444 MHz
22	2446 MHz
23	2448 MHz
24	2450 MHz
25	2452 MHz
26	2454 MHz
27	2456 MHz
28	2458 MHz
29	2460 MHz
30	2462 MHz
31	2464 MHz
32	2466 MHz
33	2468 MHz
34	2470 MHz
35	2472 MHz
36	2474 MHz
37	2476 MHz
38	2478 MHz
39	2480 MHz

BLE Physical Layer

- Two types of channels

3 Advertising Channels and 37 Data Channels

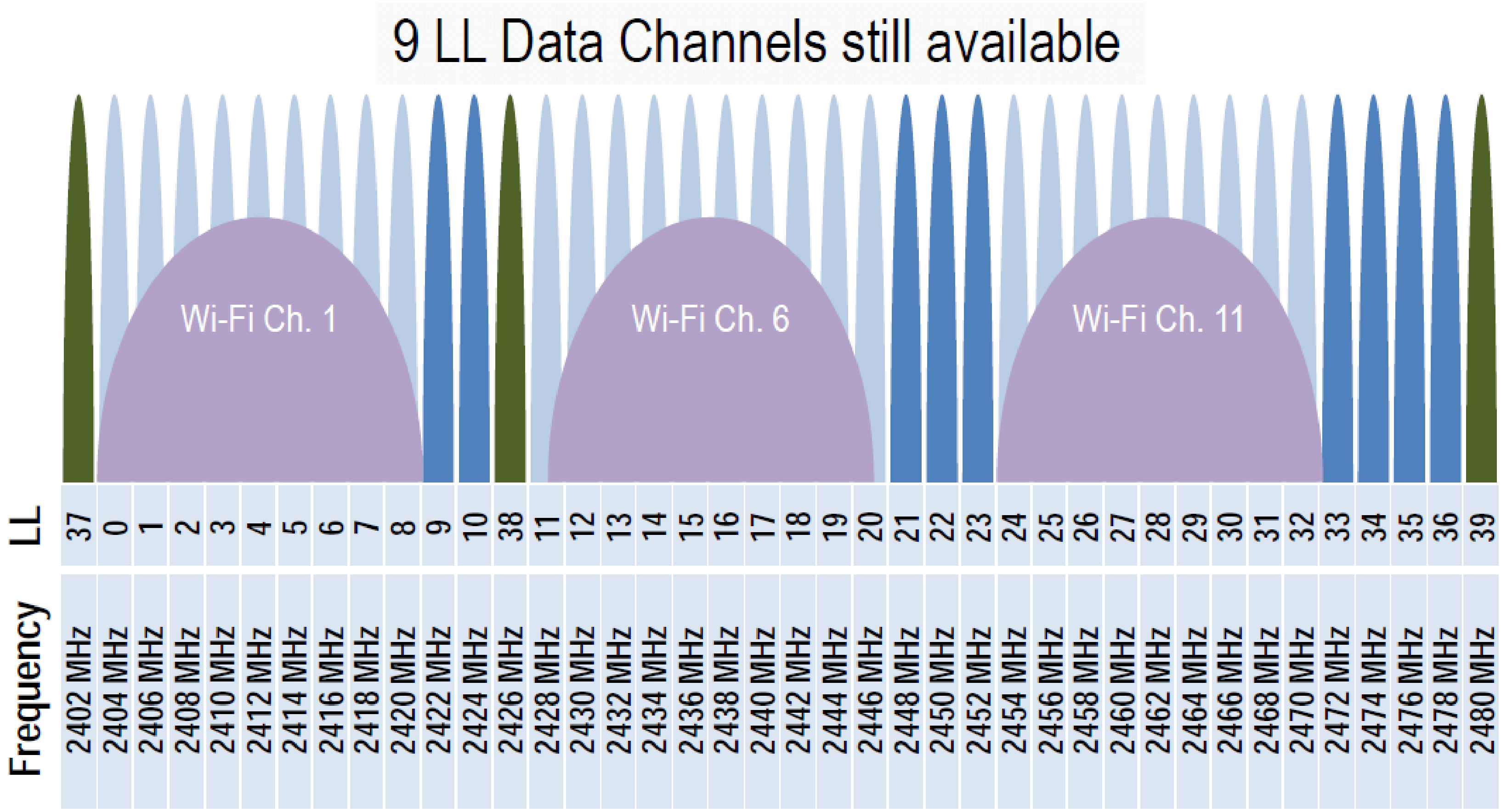


LL	37	0	1	2	3	4	5	6	7	8	9	10	38	11	12	13	14	15	16	17
Frequency	2402 MHz	2404 MHz	2406 MHz	2408 MHz	2410 MHz	2412 MHz	2414 MHz	2416 MHz	2418 MHz	2420 MHz	2422 MHz	2424 MHz	2426 MHz	2428 MHz	2430 MHz	2432 MHz	2434 MHz	2436 MHz	2438 MHz	2440 MHz

LL	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	39
Frequency	2442 MHz	2444 MHz	2446 MHz	2448 MHz	2450 MHz	2452 MHz	2454 MHz	2456 MHz	2458 MHz	2460 MHz	2462 MHz	2464 MHz	2466 MHz	2468 MHz	2470 MHz	2472 MHz	2474 MHz	2476 MHz	2478 MHz	2480 MHz

BLE Physical Layer

- Advertising channels avoid 802.11



LL	37	0	1	2	3	4	5	6	7	8
Frequency	2402 MHz	2404 MHz	2406 MHz	2408 MHz	2410 MHz	2412 MHz	2414 MHz	2416 MHz	2418 MHz	2420 MHz

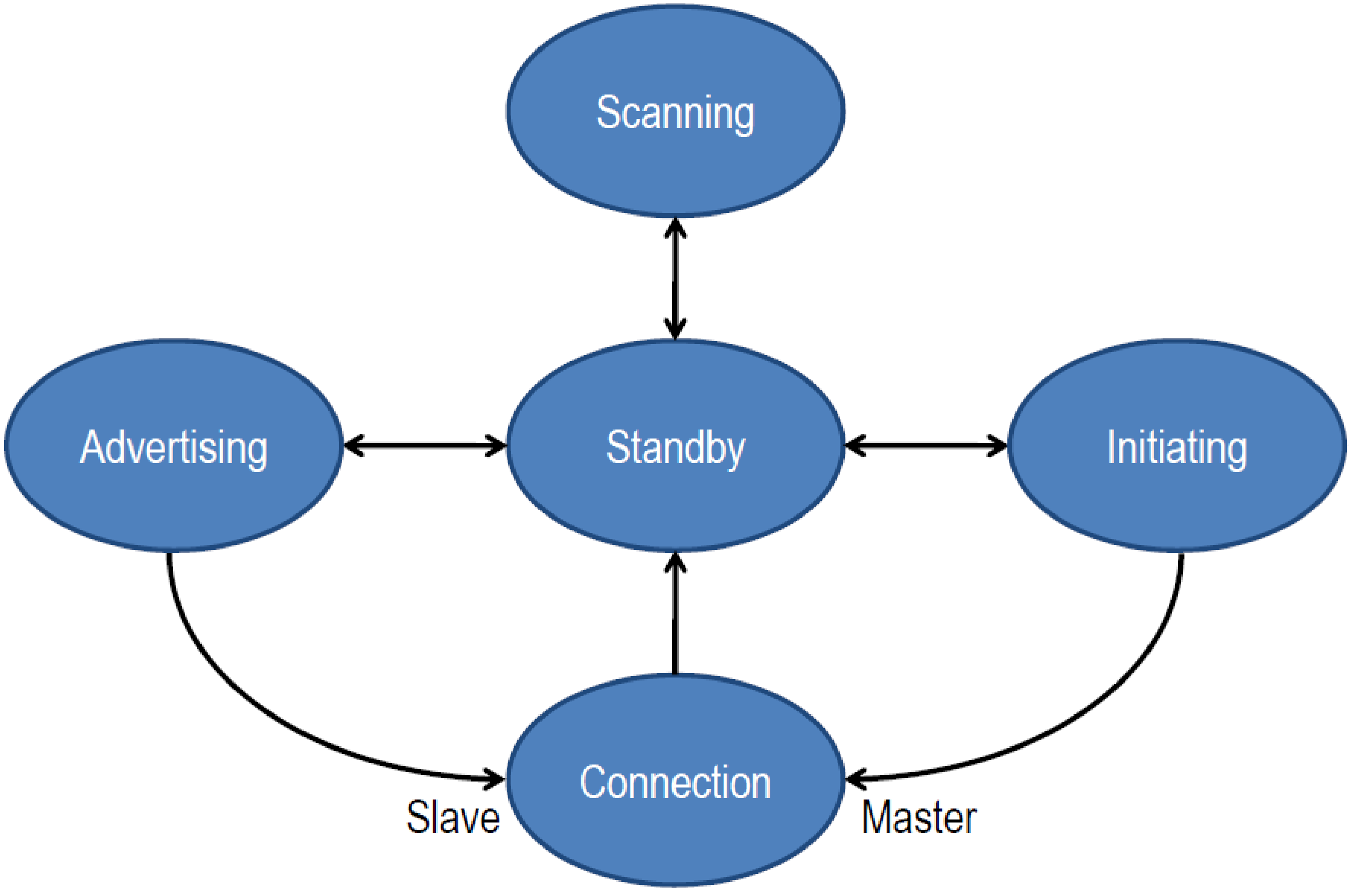
LL	9	10	38	11	12	13	14	15	16	17
Frequency	2422 MHz	2424 MHz	2426 MHz	2428 MHz	2430 MHz	2432 MHz	2434 MHz	2436 MHz	2438 MHz	2440 MHz

LL	18	19	20	21	22	23	24	25	26	27
Frequency	2442 MHz	2444 MHz	2446 MHz	2448 MHz	2450 MHz	2452 MHz	2454 MHz	2456 MHz	2458 MHz	2460 MHz

LL	28	29	30	31	32	33	34	35	36	39
Frequency	2462 MHz	2464 MHz	2466 MHz	2468 MHz	2470 MHz	2472 MHz	2474 MHz	2476 MHz	2478 MHz	2480 MHz

BLE Link Layer

- Link Layer state machine



BLE and Internet-of-Things

My pulse is …

My blood glucose is …

My temperature is …

Low power
Low data rate

Assignment 3

- Task 1: Configure Bluetooth
- Task 2: Write a client-server based intruder detection system using 2 Pis, a PIR sensor, and an LED (and/or sounder)
  - One Pi acts as server and uses a callback function that triggers when motion is detected
  - The other Pi acts as client and queries the server for the PIR value once every 5 seconds; if an intrusion is detected, the alarm is raised
  - Test both your client AND server program with the help of one (or more) of your classmates

Bluetooth Preparation

- Make sure Internet sharing is w0rking (e.g., ping a remote IP address)
- Install the Bluez Bluetooth module on your Pi:
  - *sudo apt-get update*
  - *sudo apt-get install python-bluez*
- Configure Bluetooth:
  - *sudo bluetoothctl*
    - type: *agent on*
    - type: *default-agent*
    - type: *scan on*
    - type: *quit*
- Activate scanning:
  - *sudo hciconfig hci0 piscan*
  - This call needs to be done after each reboot of the Pi!
- If needed, you can find your Pi‘s BT address using the following command:
  - *hciconfig* (the address format is: XX:XX:XX:XX:XX:XX)

Bluetooth Server Template

```
import bluetooth

hostMACAddress = ''
port = 3
backlog = 1
size = 1024
s = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
s.bind((hostMACAddress, port))
s.listen(backlog)
try:
    client, clientInfo = s.accept()
    while 1:
        data = client.recv(size)
        if data:
            print(data)
            client.send("Returning: " + data)
except:
    client.close()
    s.close()
```

Bluetooth Client Template

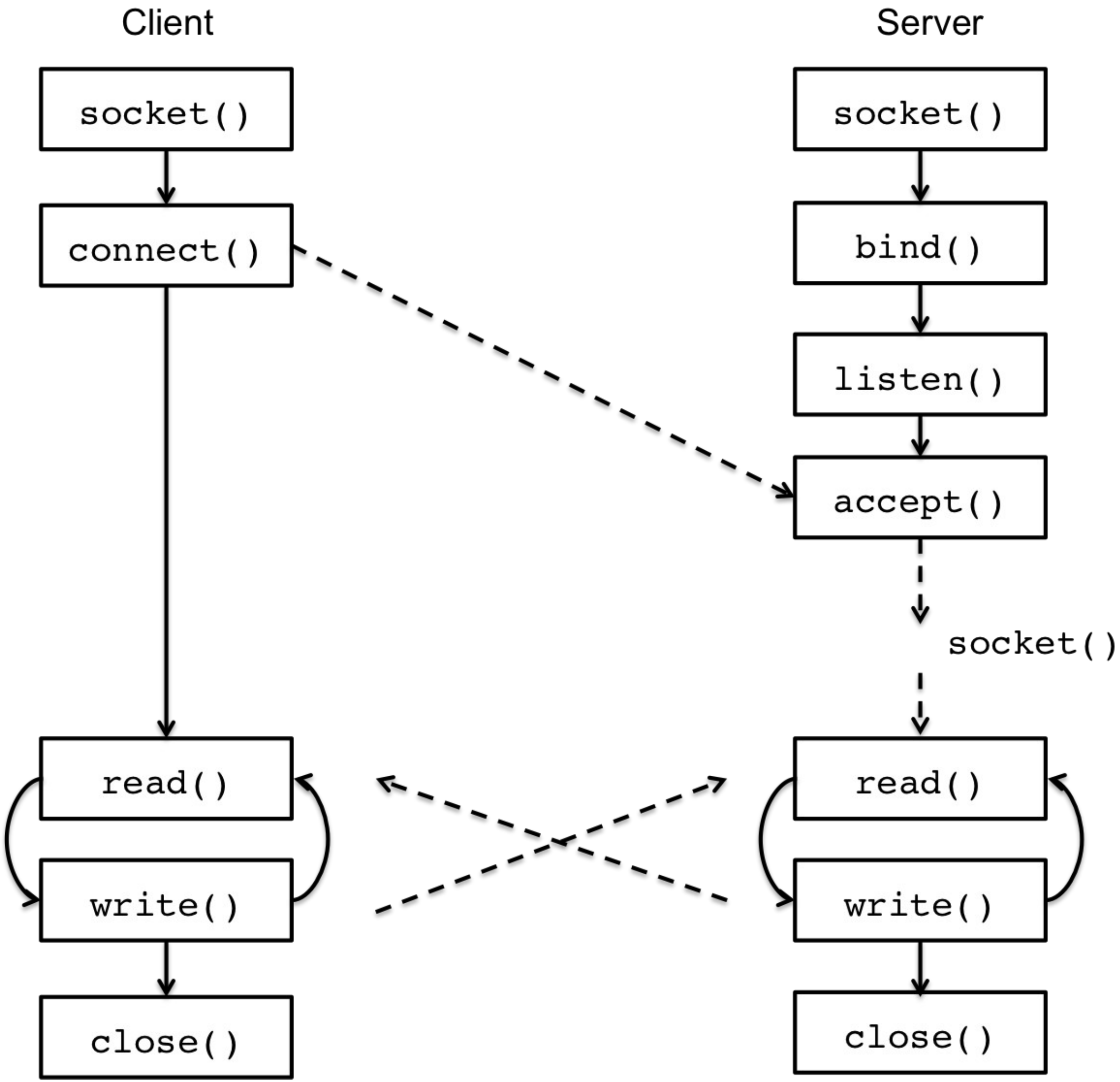
```
import bluetooth

serverMACAddress = ‘XX:XX:XX:XX:XX:XX'
port = 3
size = 1024
s = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
s.connect((serverMACAddress, port))
while 1:
    text = raw_input("Text to send: ")
    if text == "quit":
        break
    s.send(text)
    data = s.recv(size)
    print(data)
s.close()
```

Notes

- The client specifies the address of the server
- The server specifies an empty address (the server will find its BT adapter address by itself)
- The port number can be changed, but must be the same in both client & server
- The size value indicates the maximum size of a data transfer

Client/Server Communication


Client
Server
socket()
connect()
read()
write()
close()
socket()
bind()
listen()
accept()
socket()
read()
write()
close()