

CAN Communication

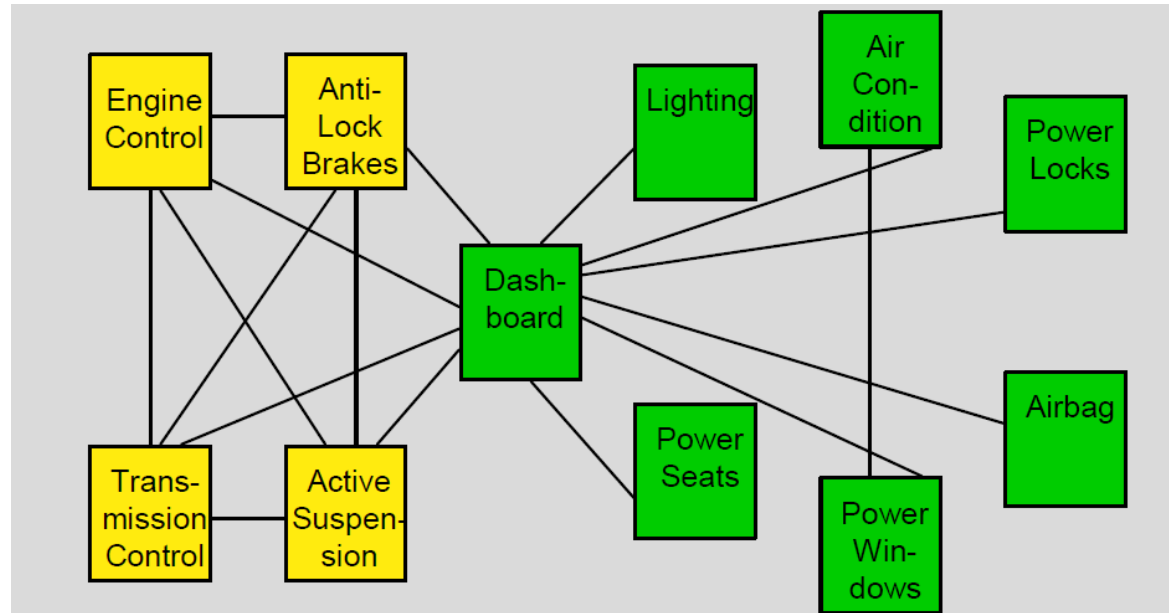
with Raspberry PI



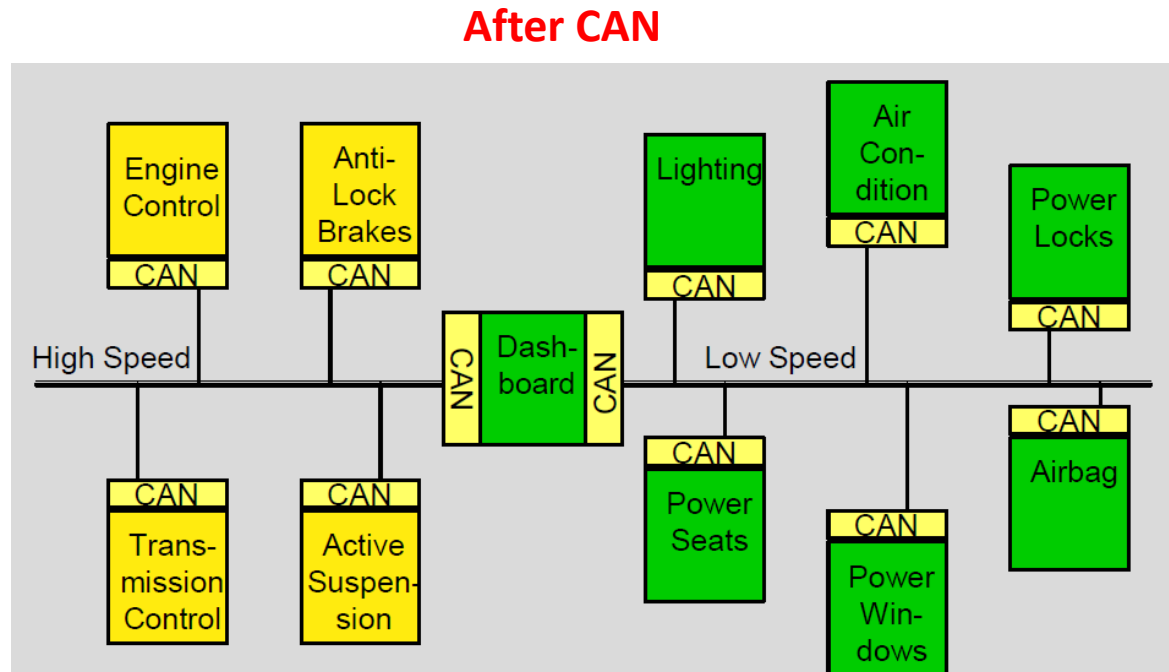
CAN History

1. In **1985** Bosch originally developed CAN, a high-integrity serial bus system for networking intelligent devices, to replace automotive point-to-point wiring systems.
2. As vehicle electronics became pervasive, complex wire harnesses which were heavy, expensive and bulky were replaced with CAN throughout the automotive industry.
3. In **1993** CAN became the international standard known as ISO 11898.
4. Since **1994**, several widely used higher-level protocols have been standardized on top of CAN, such as **CANopen*** and DeviceNet.
5. In **1996** the OnBoard Diagnostics OBD-II standard which incorporates CAN becomes mandatory for all cars and light trucks sold in the United States.
6. **Today** markets including surface transportation, industrial automation, maritime and avionics systems have widely adopted CAN.
7. **Today** CAN is incorporated into many microcontrollers

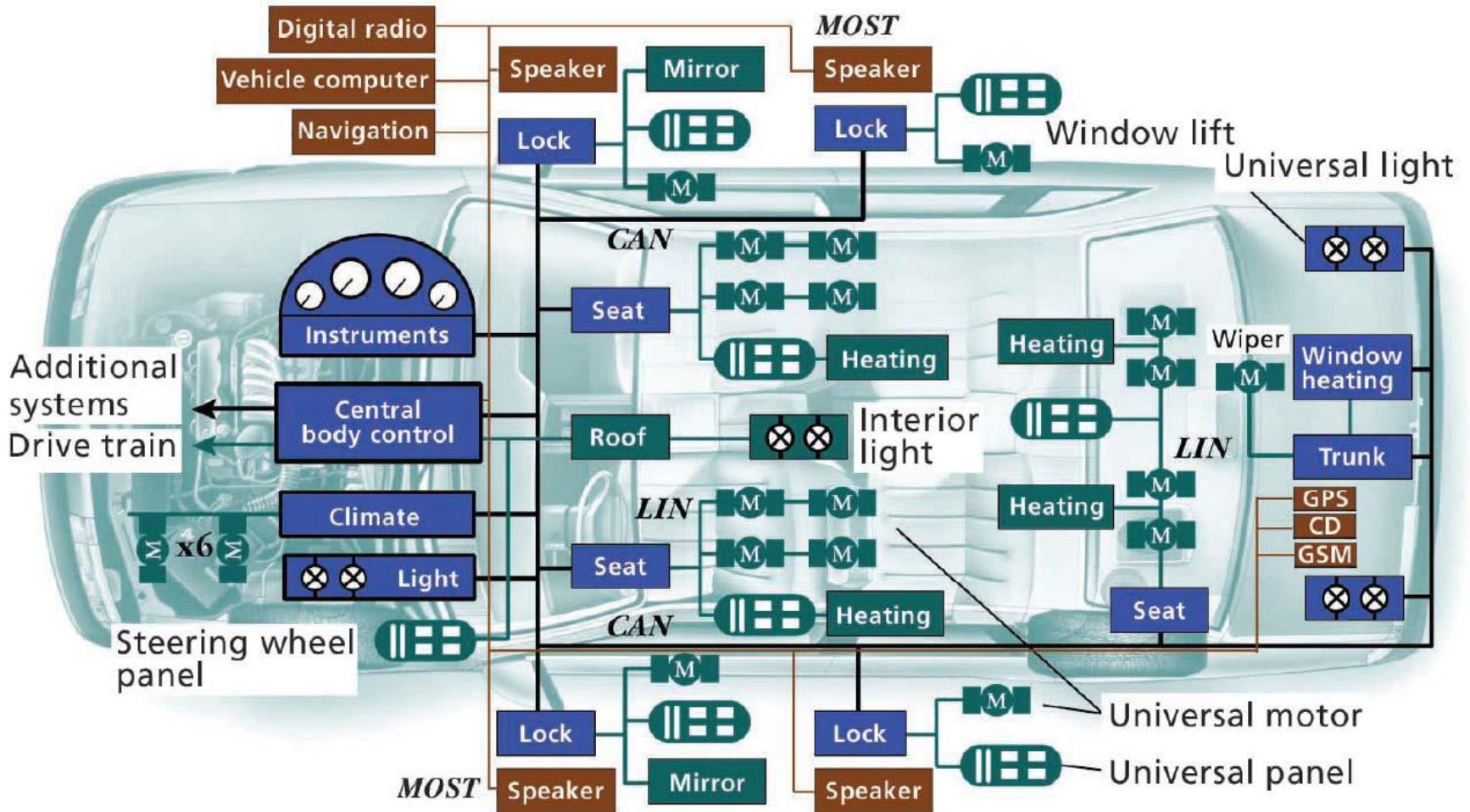
Vehicles Before CAN:
Expensive, bulky point to point wiring, wiring harnesses and many connectors.



Vehicles After CAN:
Systems of Systems with multiple CAN busses, simplified wiring harnesses and many Fewer connectors



CAN is Now Central to Automotive Networks



- CAN Controller area network
- GPS Global Positioning System
- GSM Global System for Mobile Communications
- LIN Local interconnect network
- MOST Media-oriented systems transport

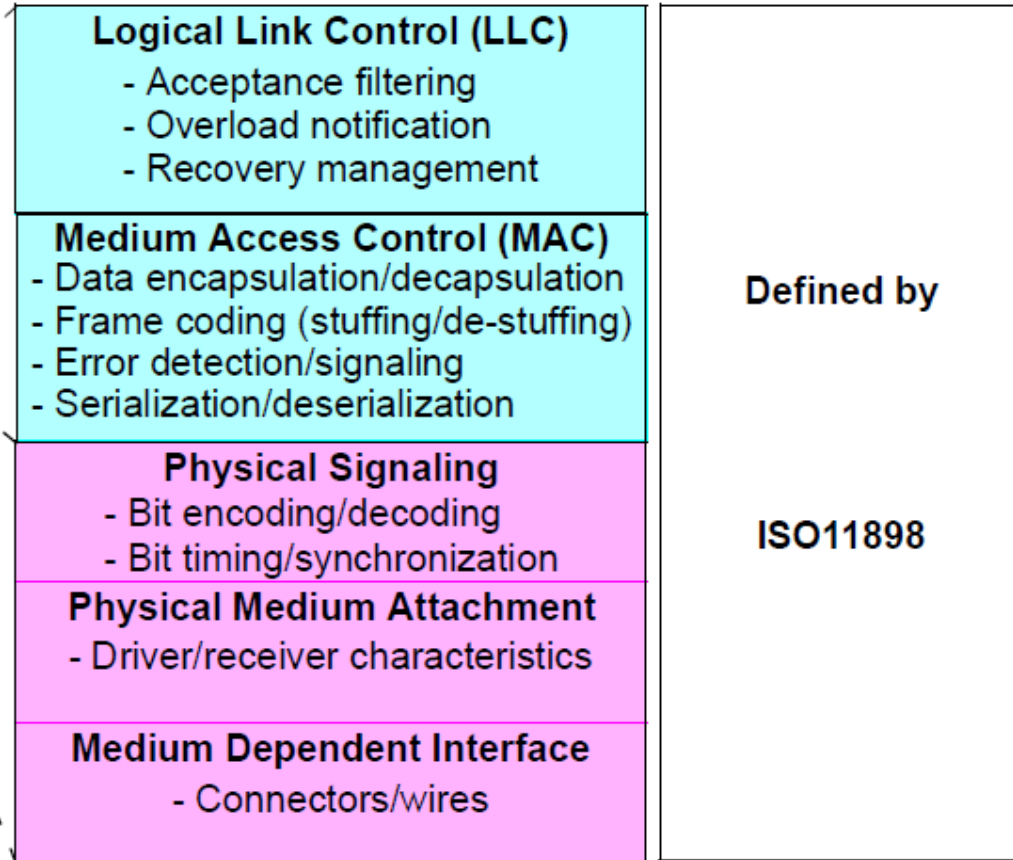
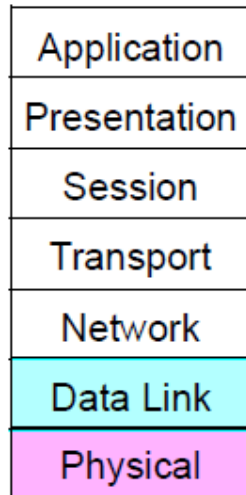
New cars typically contain 50 to 100 microcontrollers

Advantages of CAN

1. Low cost network infrastructure which is often built into microcontrollers.
2. Large market segment with broad availability of hardware, software and systems engineering tools.
3. Light weight, low latency, highly deterministic design specifically for real-time embedded applications.
4. Reliable with strong error detection, fault tolerant versions available.
5. Flexible and highly configurable with various higher level application protocols.
6. Foundation for next generation technology controller area networks.

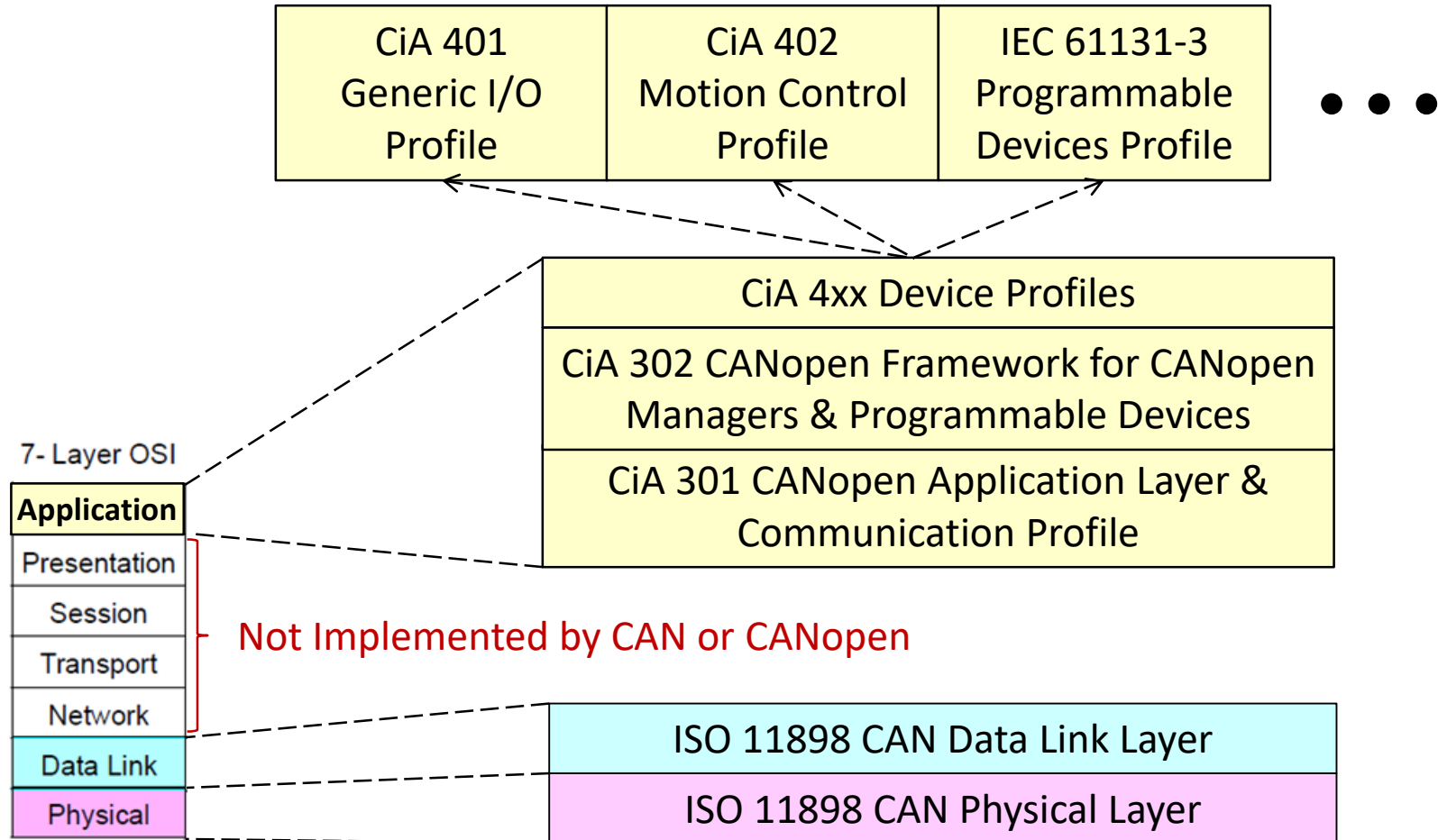
CAN & International Standards Organization (ISO) Open Systems Interconnect (OSI) Reference Model

7- Layer OSI



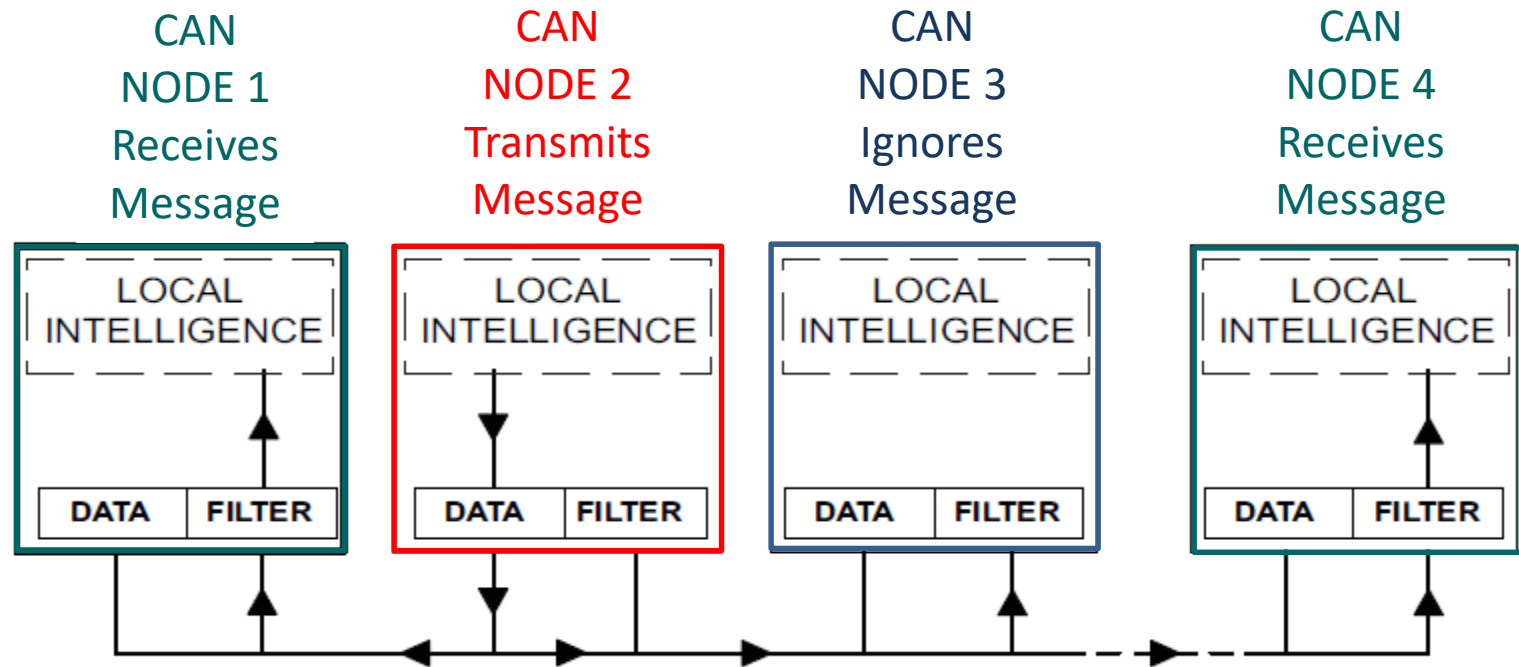
High level CAN Protocols implement Application layer and skip the four intervening layers

The CANopen Application



High level CAN Protocols implement Application layers and skip the four intervening layers

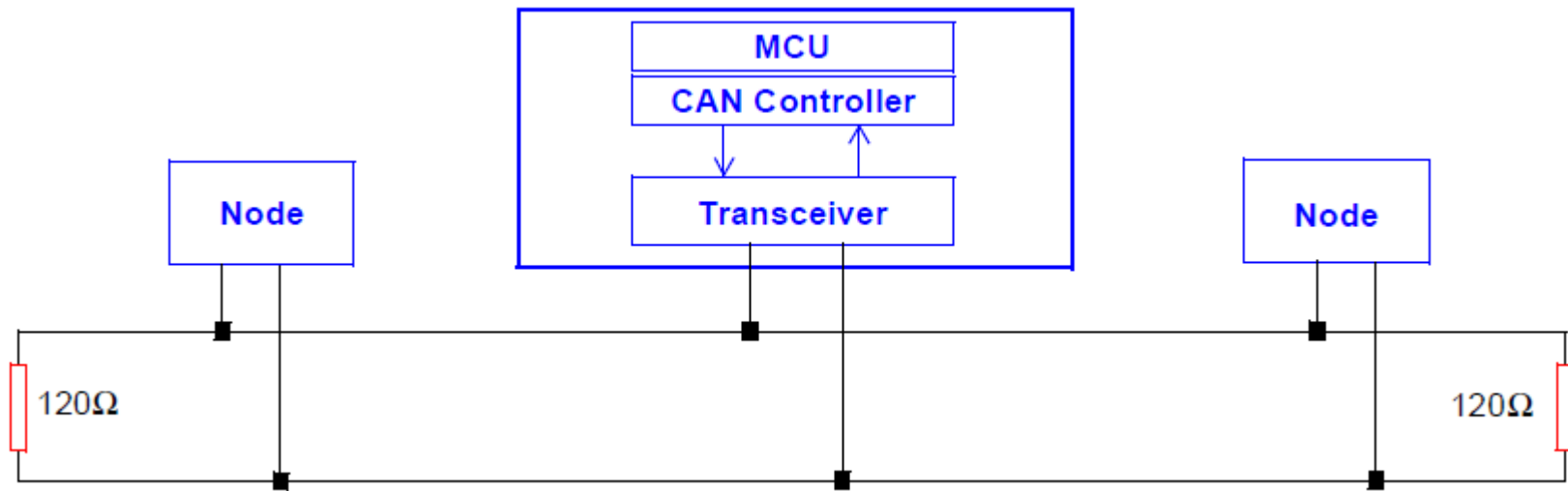
CAN Data-Flow Model



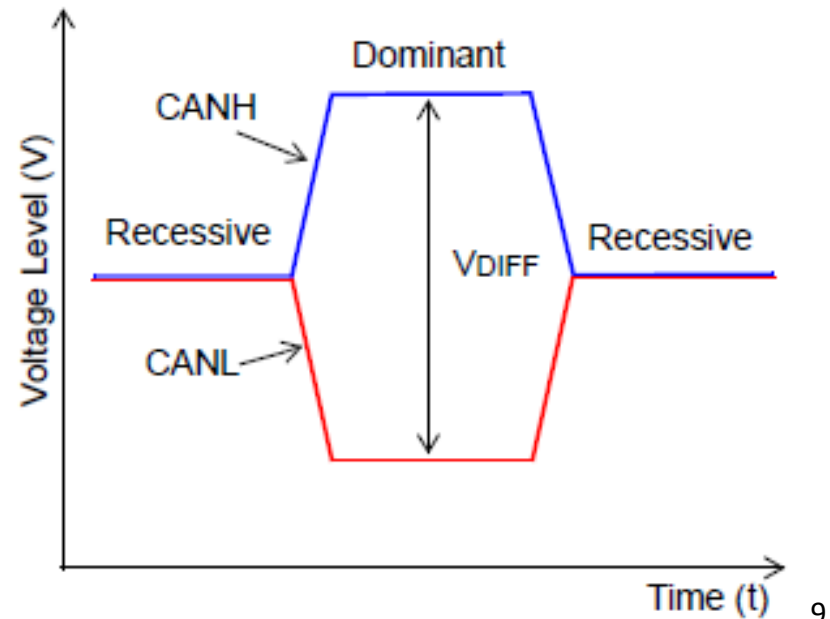
One node transmits, all nodes listen and processor data frames selectively. Message filtering is typically performed in transceiver hardware. This data flow supports a broad range of network communication models:

1. Master / Slave : All communications initiated by master node
2. Peer-to-Peer : Nodes interact with autonomously with equal authority
3. Producer / Consumer : Producer nodes broadcast (push) messages to Consumer nodes
4. Client / Server : Client nodes request (pull) data from Server nodes

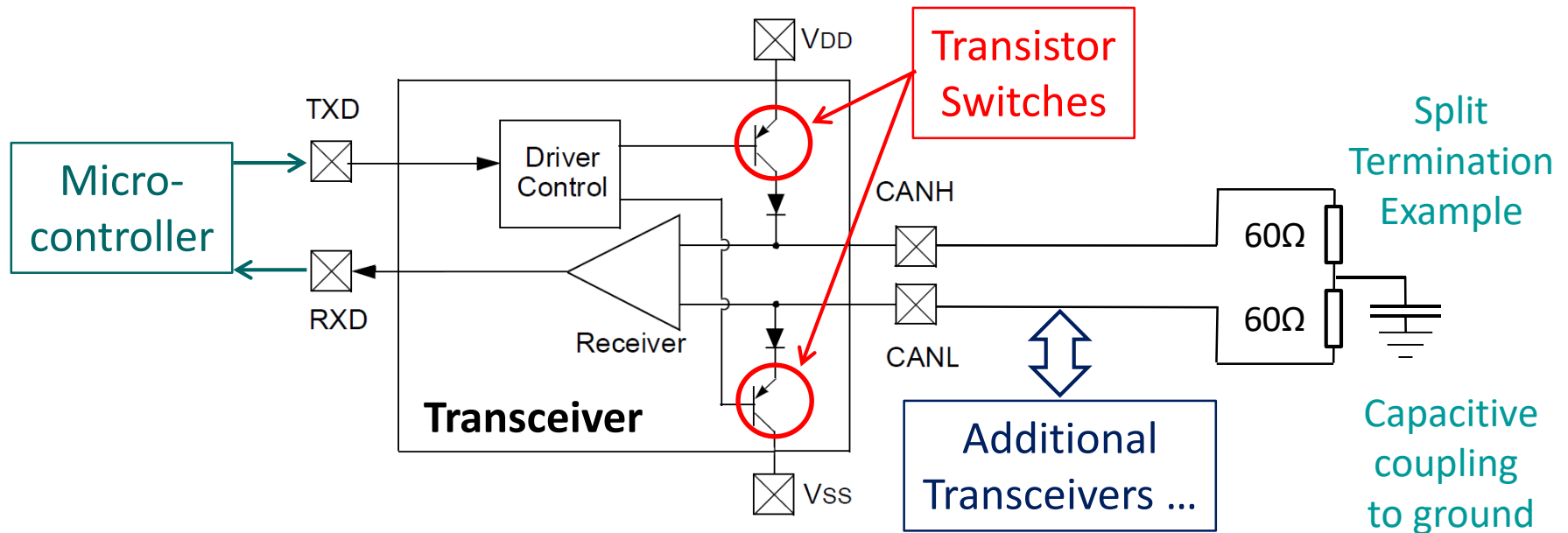
CAN Typical High-Speed Physical Layer



- CAN uses differential signaling to improve signal to noise ratio. Termination resistors reduce signal reflection.
- Idle bus state is **Recessive** with no applied differential signal: $V_{CAN_H} \approx V_{CAN_L}$
- **Dominant** state occurs when one or more nodes drive the bus state to: V_{DIFF}

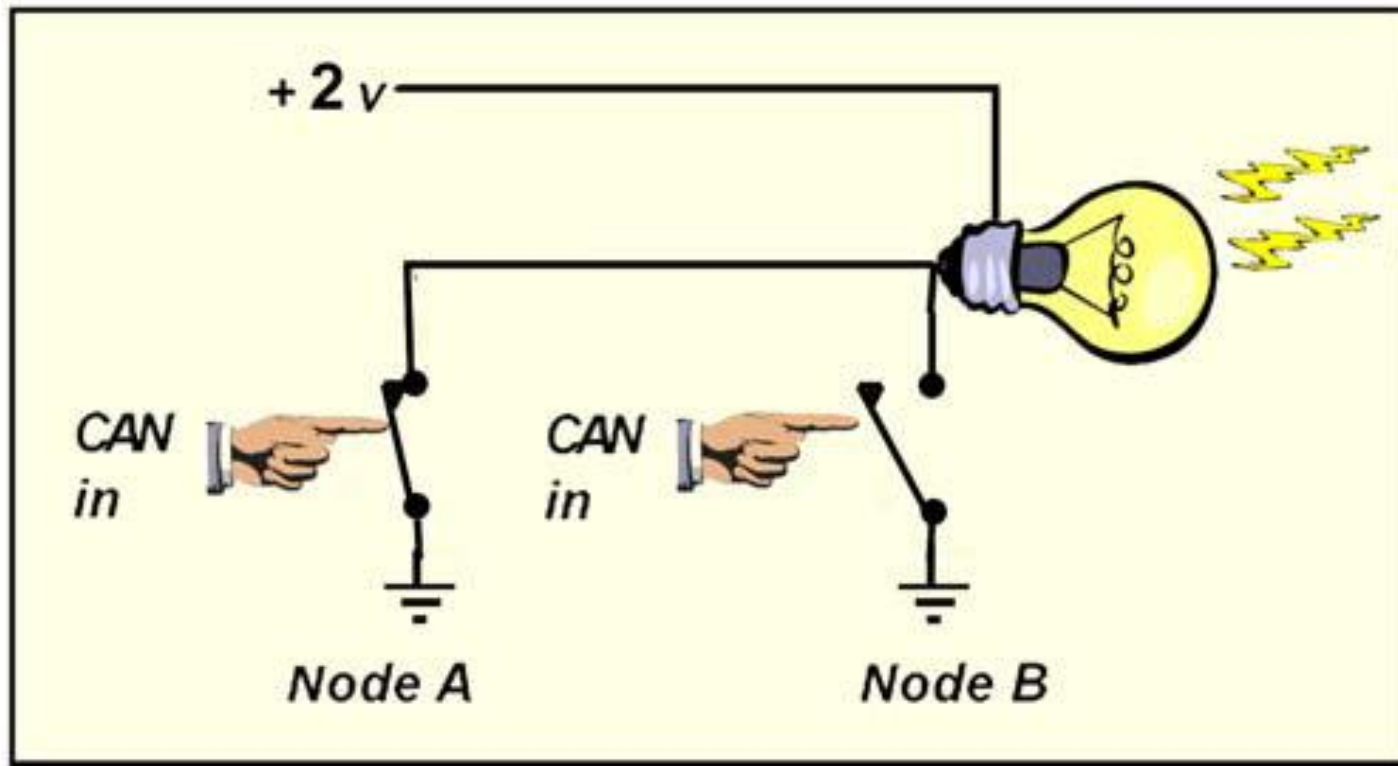


CAN Differential Bus Interface Transceivers



- The CAN idle state presents a recessive state, signaled by a small differential voltage across CANH and CANL. With the indicated split termination, this idle voltage will be halfway between VDD (positive supply) and VSS (ground).
- The CAN dominant state occurs when one or more transceivers simultaneously close the indicated transistor switches driving CANH and CANL toward VDD and VSS, respectively.
- This open collector transistor switch configuration is referred to as a “**wired or**” since any node transmitting a dominant bit always overrides a recessive bit. Since a dominant bit represents a logic 0, this arrangement is sometimes referred to as “**wired and**” since bus a logic “1” state is achieved only if all nodes (node 1 **AND** node 2 **AND** node 3 ...) signal logic “1” recessive bits).

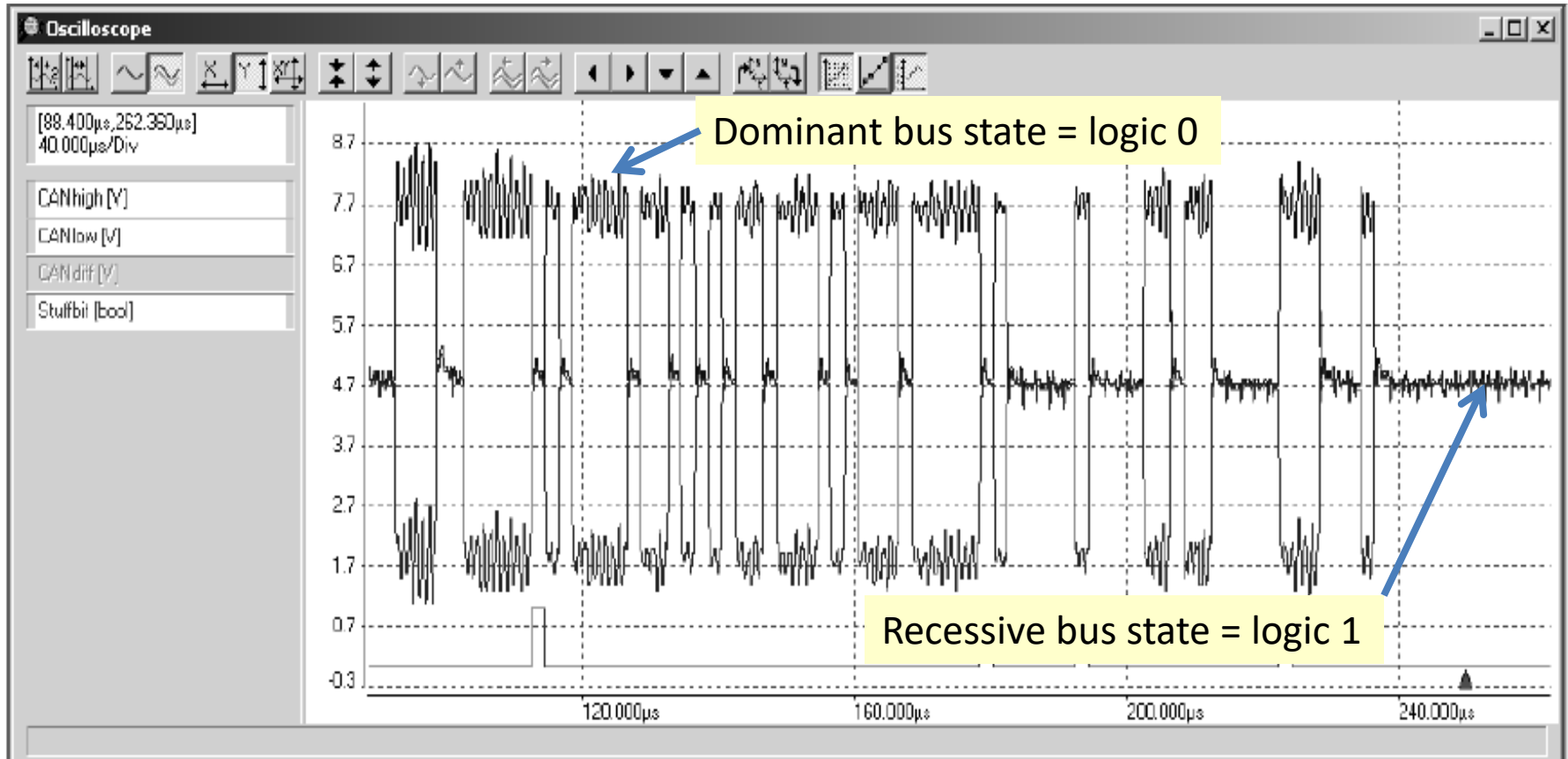
Example of a “Wired OR”



Closing Node A switch **OR** closing Node B switch turns on the light.

Conversely, the light is off unless Node A switch is open **AND** Node B switch is also open.

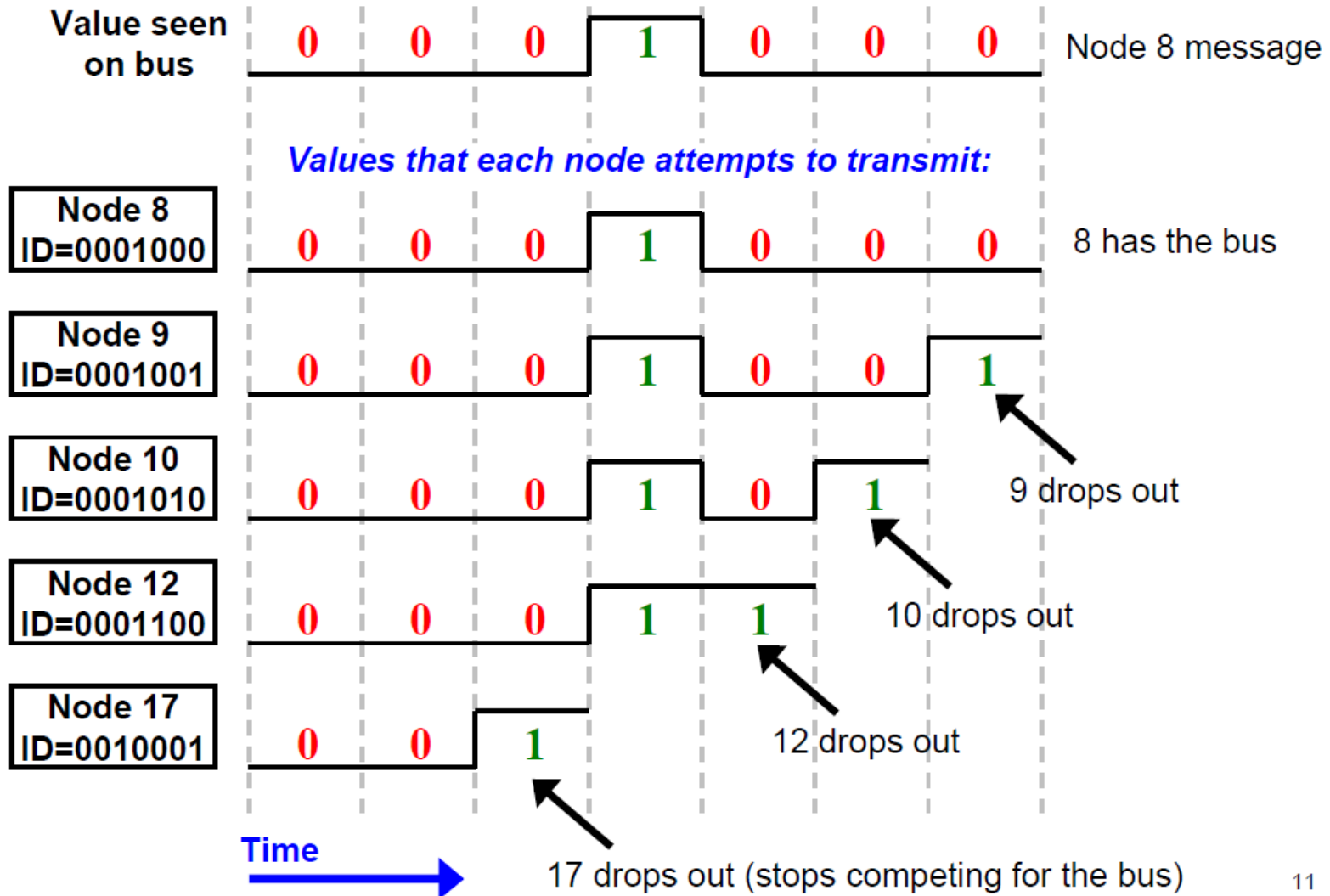
Example CAN Sample Signaling



CAN Logic & Arbitration

1. CAN 2.0A messages begin with an 11-bit message ID which identifies the message type and also establishes the message priority.
2. As with many computer interfaces, the CAN transceivers invert the microcontroller signal. Thus, the **dominant** bus state occurs when a **logic “0”** is transmitted and the **recessive** state occurs when a **logic “1”** is transmitted.
3. CAN uses the message ID to perform bus access arbitration between nodes.
4. Each node waits for an idle bus state then begins to transmit its message ID.
5. Each node also listens to the bus to see if the bus state match its transmission.
6. If a node detects a dominant bus state while transmitting a recessive message ID bit (logic “1”), it drops out of the current arbitration round and will try again the next time the bus is idle

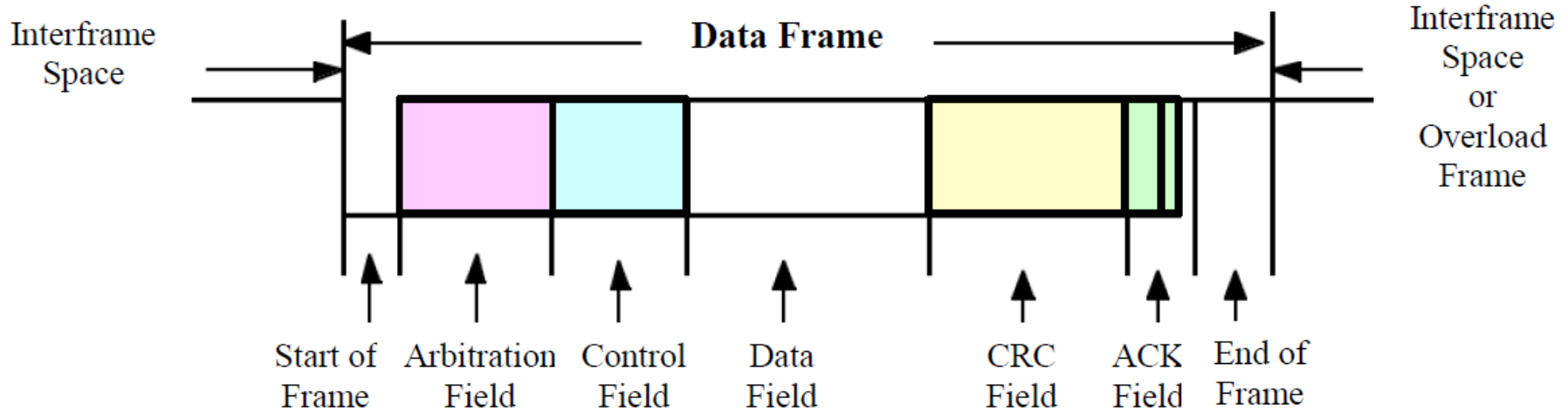
7-bit CANopen Node ID Arbitration Example



Key Advantages of CAN Bus Arbitration

1. Fast & deterministic.
2. Highest priority message gets immediate access once the bus is available.
3. Arbitration is essential “free” since message ID encodes message priority.
4. Unlike Carrier Sense Multiple Access with Collision Detect (CSMA/CD) arbitration propagation delays never cause message collisions.

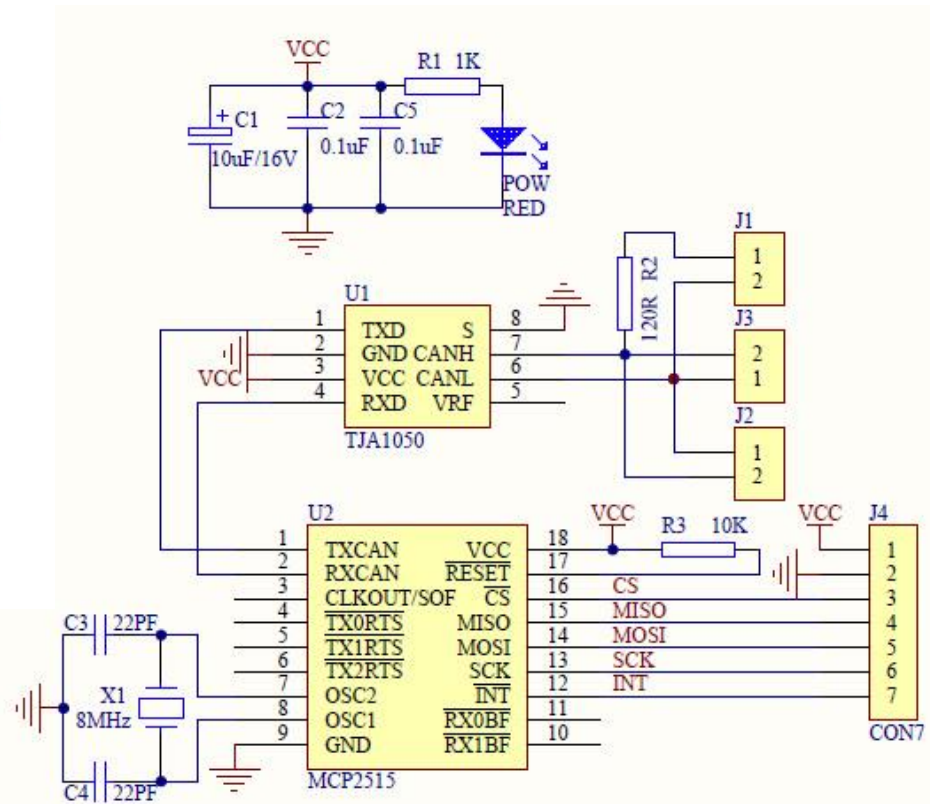
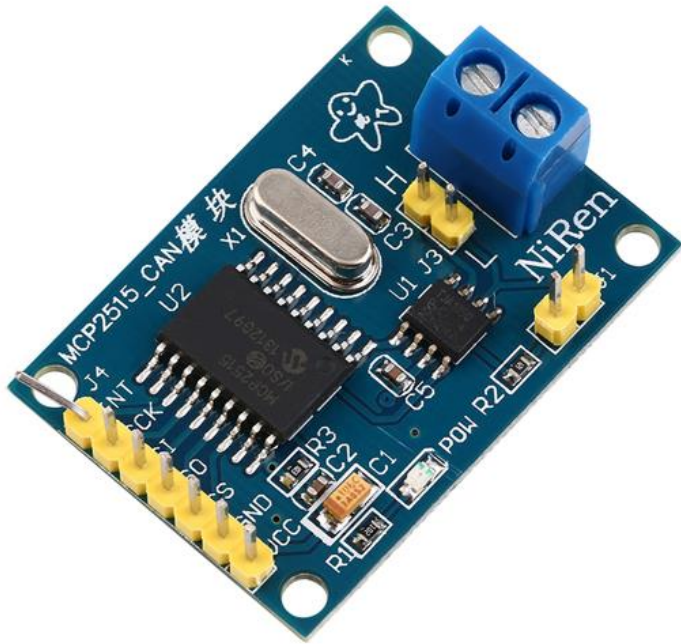
CAN Data Frame Format



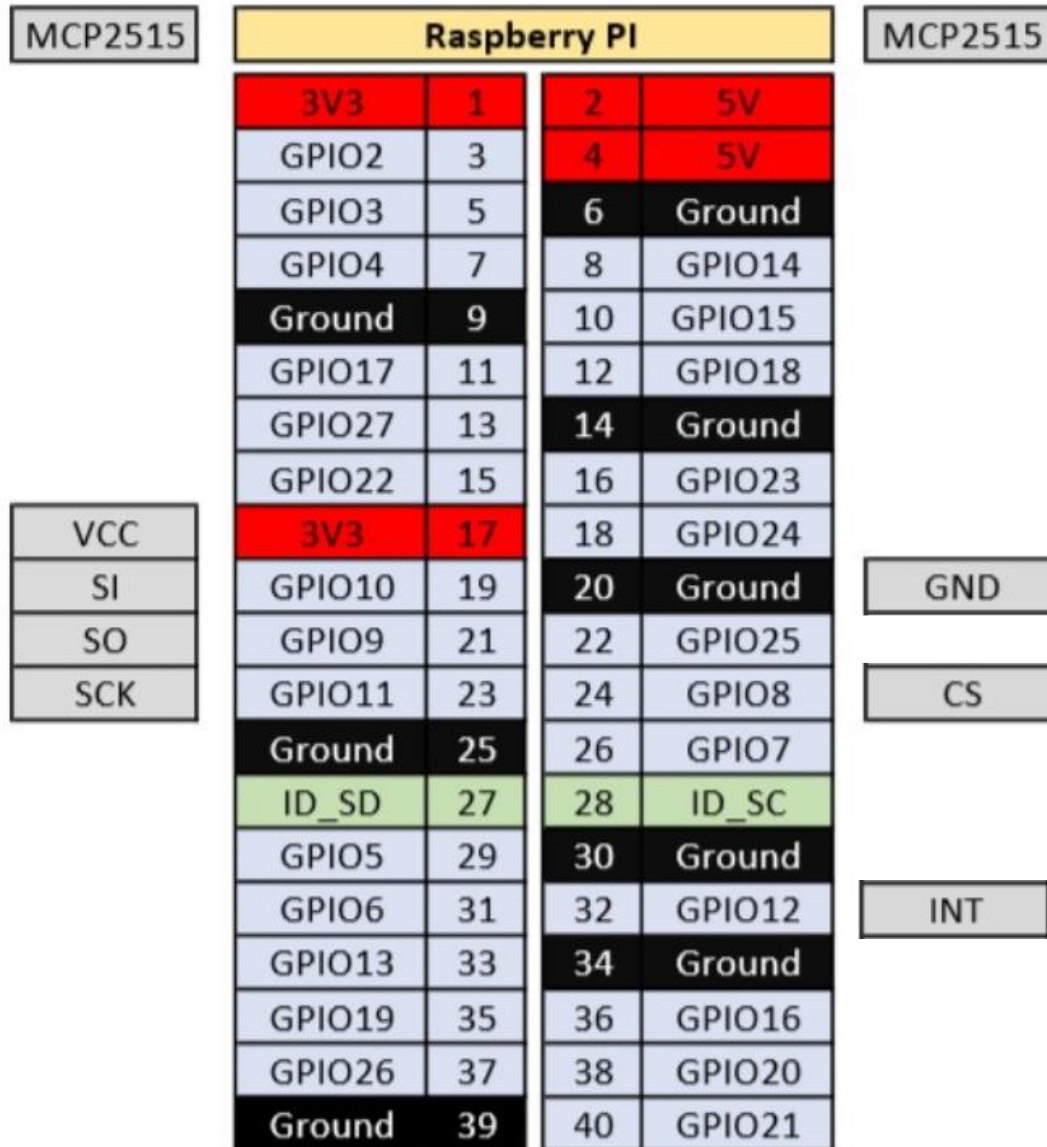
Base Field Name	Length (bits)	Purpose
Start of Frame	1	Denotes the start of frame transmission
Message Identifier / Arbitration Field	11	Message identifier also represents the message priority
Remote Transmission Request (RTR)	1	Dominant (0): Data is included in message Recessive (1): Remote Frame request for data
Identifier extension bit (IDE)	1	Must be dominant (0) for 11 bit message IDs
Reserved bit (r0)	1	Reserved bit should be dominant (0) for 11 bit IDs
Data length code (DLC)	4	Number of bytes of data (0–8 bytes)
Data Field	0–64	0 to 8 bytes of data (length dictated by DLC field)
CRC Filed	15	Cyclic redundancy check
CRC delimiter	1	Must be recessive (1)
ACK slot	1	Transmitter sends recessive (1) and any receiver can assert a dominant (0) to acknowledge message
ACK delimiter	1	Must be recessive (1)
End-of-frame (EOF)	7	Must be recessive (1)

MCP2515 CAN Controller

■ SPI interface



Raspberry PI I/O



/boot/config.txt

SmarTTY - 192.168.0.134

```
File Edit View SCP Tools Help
43 #arm_freq=800
44
45 # Uncomment some or all of these to enable the optional hardware
46 dtparam=i2c_arm=on
47 #dtparam=i2s=on
48 #dtparam=spi=on
49
50 # Uncomment this to enable infrared communication.
51 #dtoverlay=gpio-ir,gpio_pin=17
52 #dtoverlay=gpio-ir-tx,gpio_pin=18
53 dtoverlay=mcp2515-can0,oscillator=8000000,interrupt=12
54 dtoverlay=spi-bcm2835-overlay
55
```

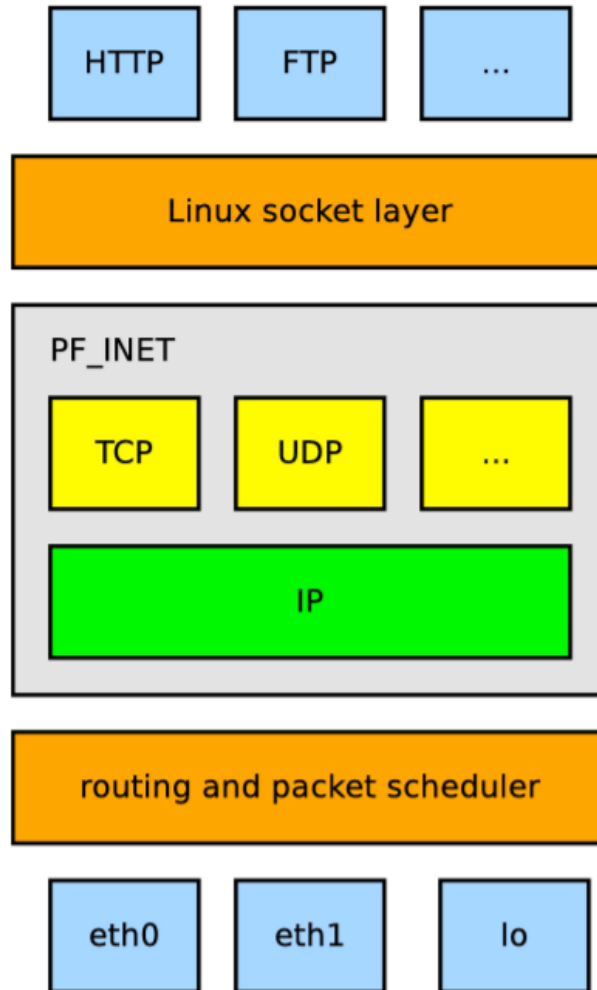
sudo ip link set can0 up type can bitrate 500000

```
pi@192.168.0.134:/etc$ sudo ifconfig
can0: flags=193<UP,RUNNING,NOARP> mtu 16
    unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 txqueuelen 10 (UNSPEC)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

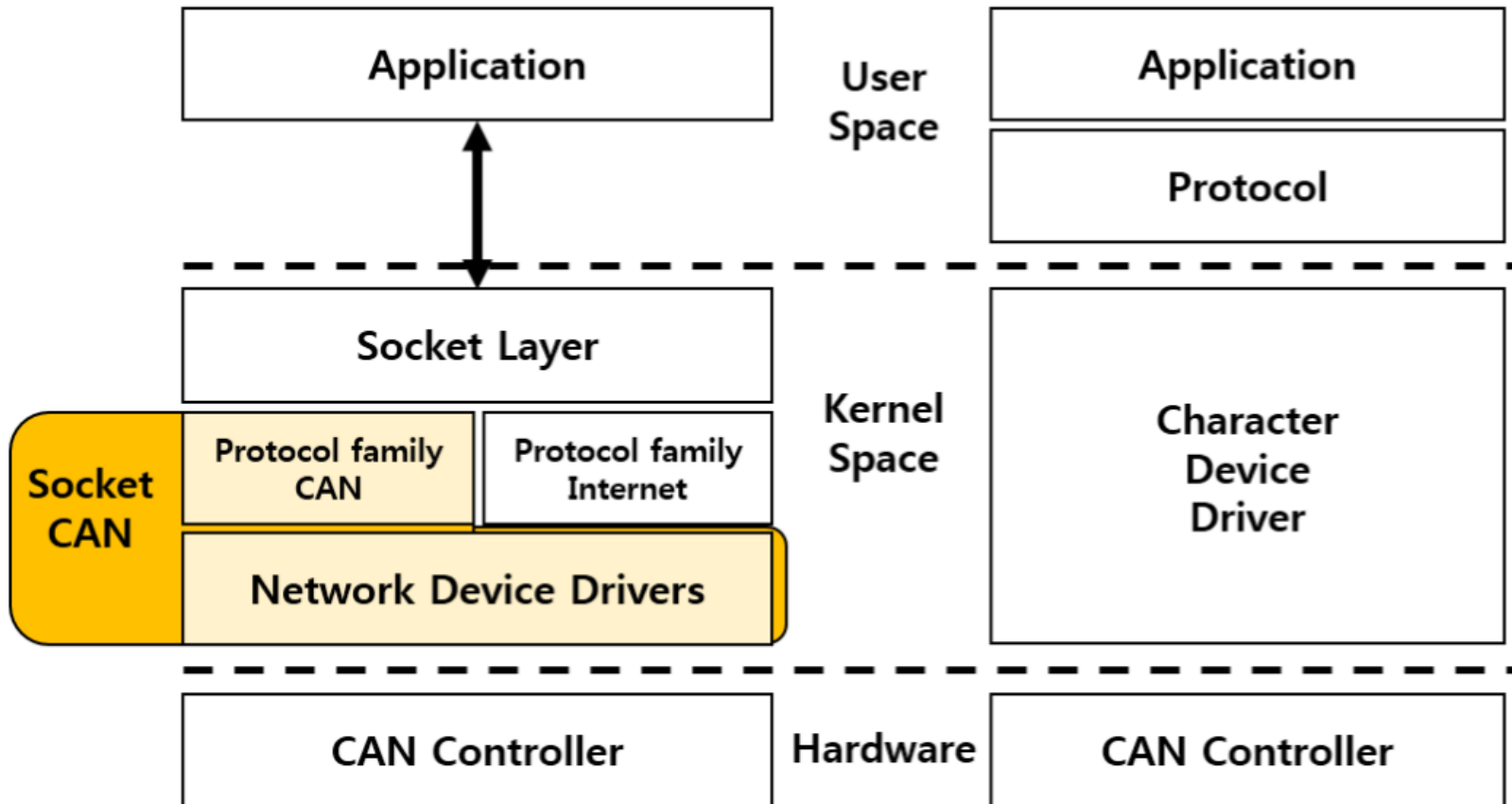
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.0.134 netmask 255.255.255.0 broadcast 192.168.0.255
    inet6 fe80::13bb:1982:d6f9:8d71 prefixlen 64 scopeid 0x20<link>
    ether dc:a6:32:7c:4a:04 txqueuelen 1000 (Ethernet)
    RX packets 411 bytes 45700 (44.6 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 362 bytes 98793 (96.4 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
```

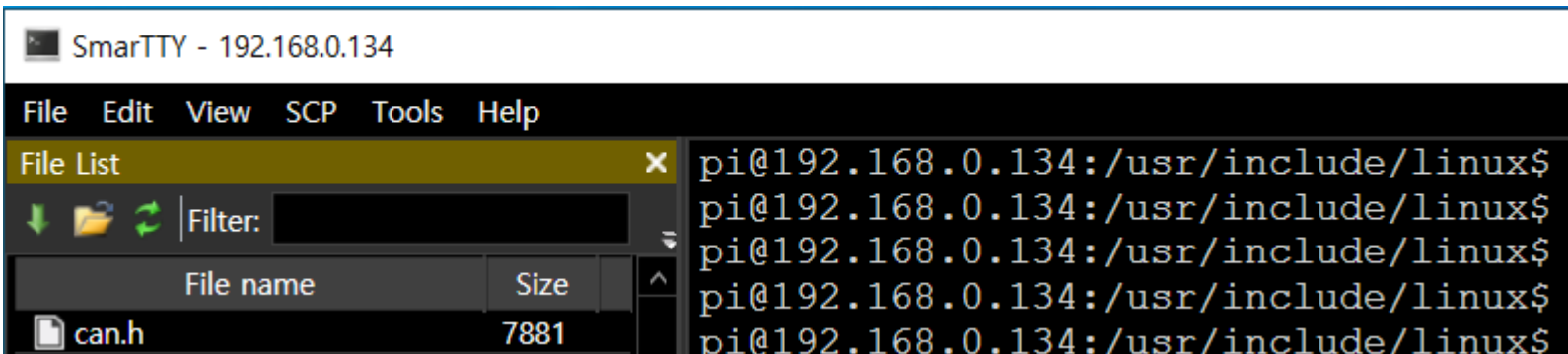
Linux Socket Layer



Socket CAN



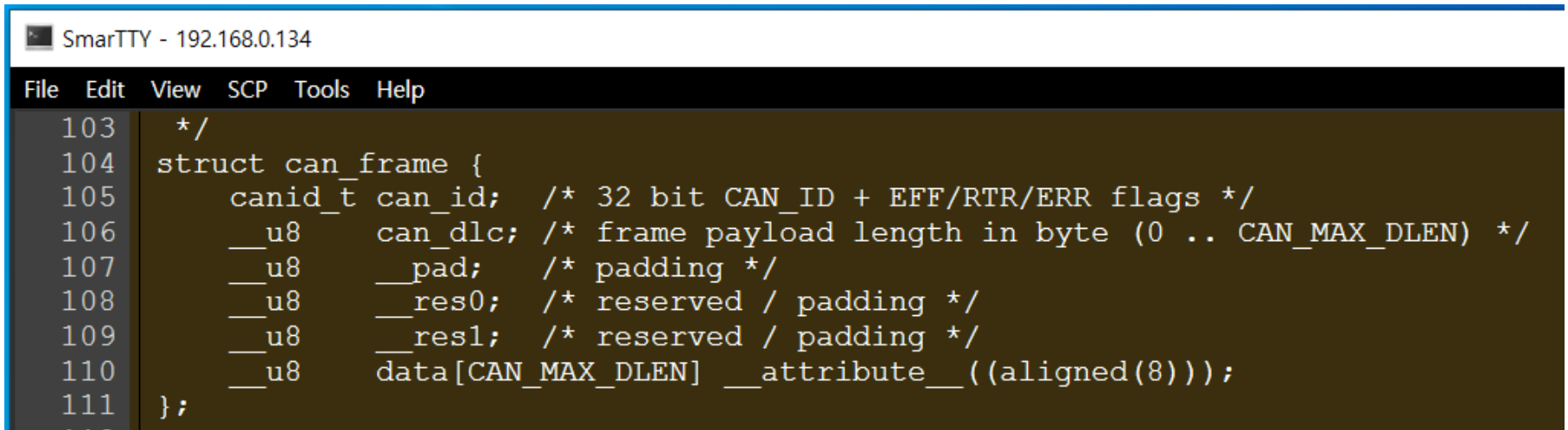
CAN frame



The screenshot shows a terminal window titled "SmarTTY - 192.168.0.134". The menu bar includes "File", "Edit", "View", "SCP", "Tools", and "Help". A "File List" window is open, displaying a table with columns "File name" and "Size". The table contains one entry: "can.h" with a size of "7881". To the right of the file list, the terminal shows five repeated prompts: "pi@192.168.0.134:/usr/include/linux\$".

File name	Size
can.h	7881

```
pi@192.168.0.134:/usr/include/linux$
pi@192.168.0.134:/usr/include/linux$
pi@192.168.0.134:/usr/include/linux$
pi@192.168.0.134:/usr/include/linux$
pi@192.168.0.134:/usr/include/linux$
```



The screenshot shows a terminal window titled "SmarTTY - 192.168.0.134". The menu bar includes "File", "Edit", "View", "SCP", "Tools", and "Help". The terminal displays C code defining a CAN frame structure. The code is as follows:

```
103  */
104  struct can_frame {
105      canid_t can_id; /* 32 bit CAN_ID + EFF/RTR/ERR flags */
106      __u8 can_dlc; /* frame payload length in byte (0 .. CAN_MAX_DLEN) */
107      __u8 __pad; /* padding */
108      __u8 __res0; /* reserved / padding */
109      __u8 __res1; /* reserved / padding */
110      __u8 data[CAN_MAX_DLEN] __attribute__((aligned(8)));
111  };
```

Exercise

- Raspberry PI 를 이용하여 Ubidots.com IoT server에 현재의 온도와 습도를 전송하는 프로그램을 작성한다.(LabCortexWiFi.pdf의 Exercise 2: IoT Exercise using WIFI Module와 동일한 기능)
- 현재의 온도와 습도는 STM32F407 Discovery 보드에 연결된 DHT22 센서를 이용하여 측정하며, 이 측정 데이터는 CAN 통신을 통해서 Raspberry PI 로 전송한다.
- STM32 보드에서 측정하는 온도와 습도 값을 Ubidots server에 전송할 때 반드시 동기화 될 필요는 없으며 몇 초간의 지연은 허용한다.