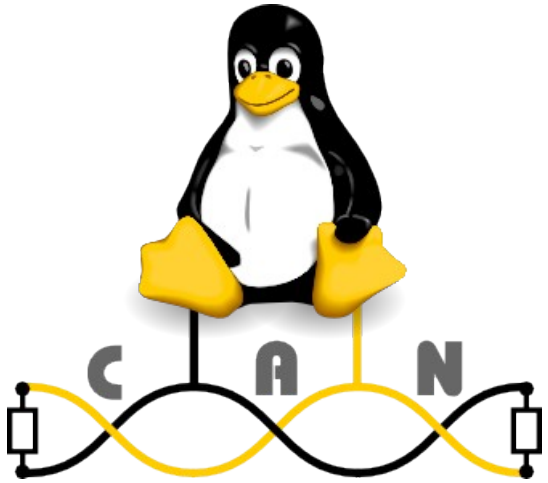
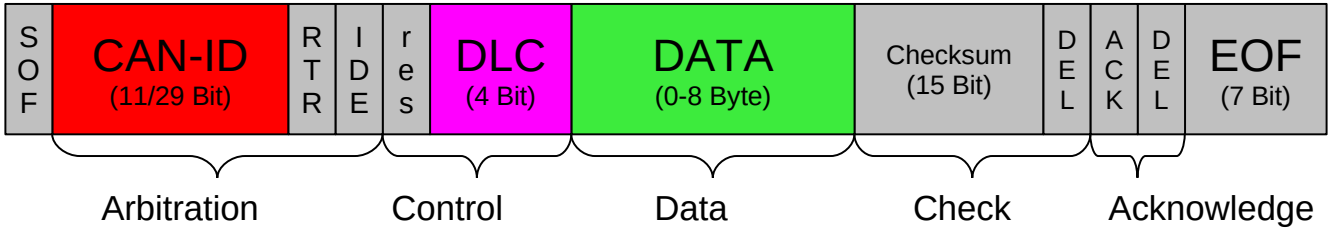


Controller Area Network

Concept and usage

Data packets on the bus - simplified for nerds

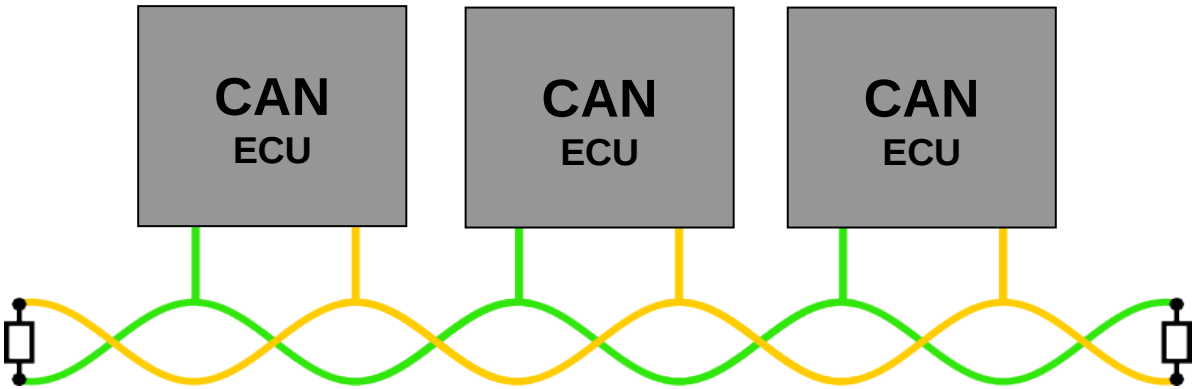
- Media access by CSMA/CA (avoidance) sometimes CSMA/CR (resolution)
- Structure of a CAN frame:



- Simplified: [CAN Identifier] [Data length] [Data 0..8]
- Content addressing (by CAN Identifier & CAN Bus)
- No MAC / Node addresses / ARP / Routing – just plain OSI Layer 2
- Incompatible Upgrade CAN FD (ISO 11898-1:2015)

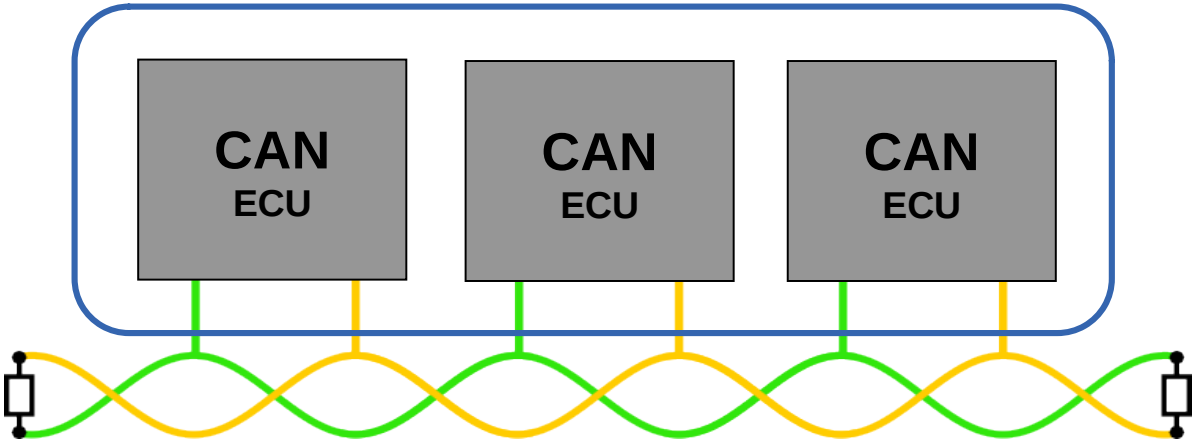
Safe but not secure

- Serial communication protocol, ISO Standard 11898
- Two wires: Unshielded twisted pair, terminated
- Transfer rate up to 1MBit/s (CAN) and up to ~4MBit/s (CAN FD)
- Invented by Robert Bosch GmbH, 1983
- Only defines the Media Access Layer



Multi-master bus topology

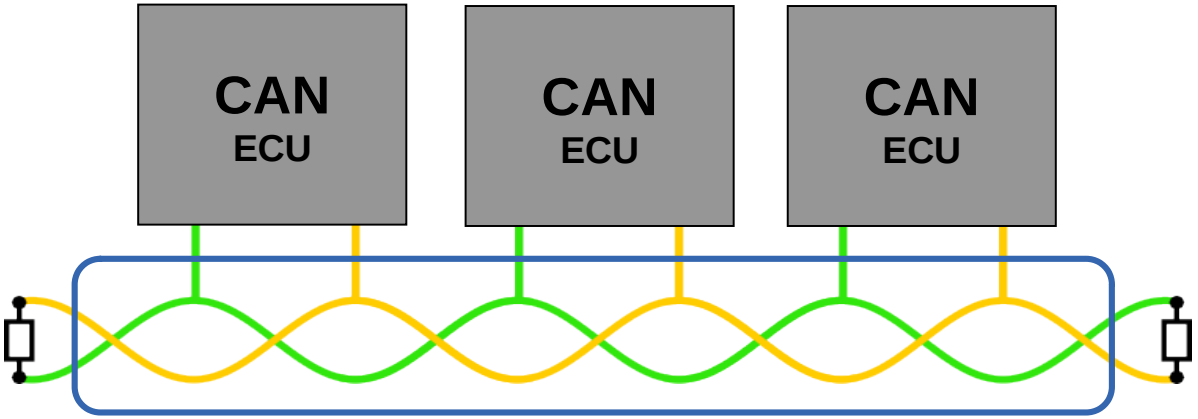
Unlike point-to-point Ethernet all nodes use the same wire



The wires are named CAN_Low and CAN_High

Differential potentials for electro magnetic interference (EMI)

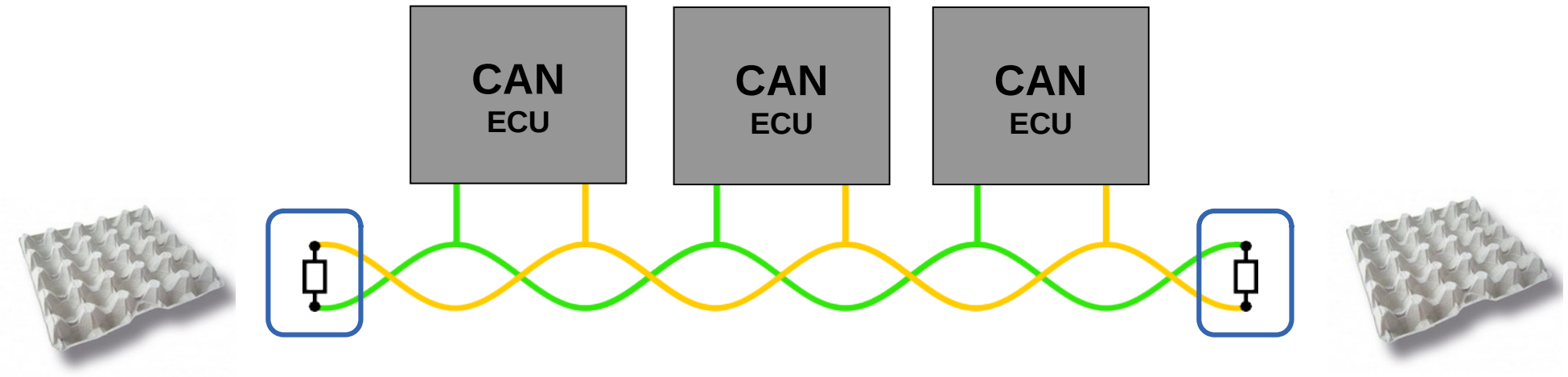
A glitch occurs on both lines at the same time => difference remains stable



Unshielded twisted pair

Bus termination to prevent reflection/echos

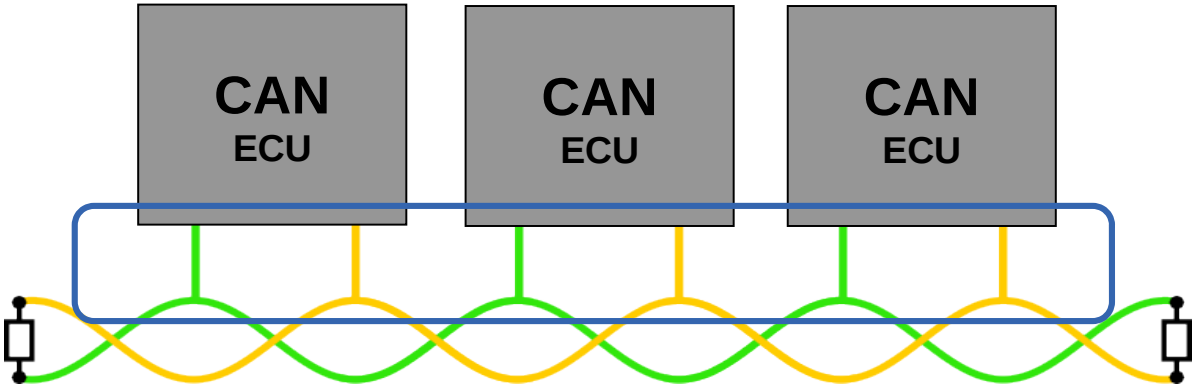
Terminate data transmission to prevent reflection/echos a the end of the wire



120 Ohms each => 60 Ohms total

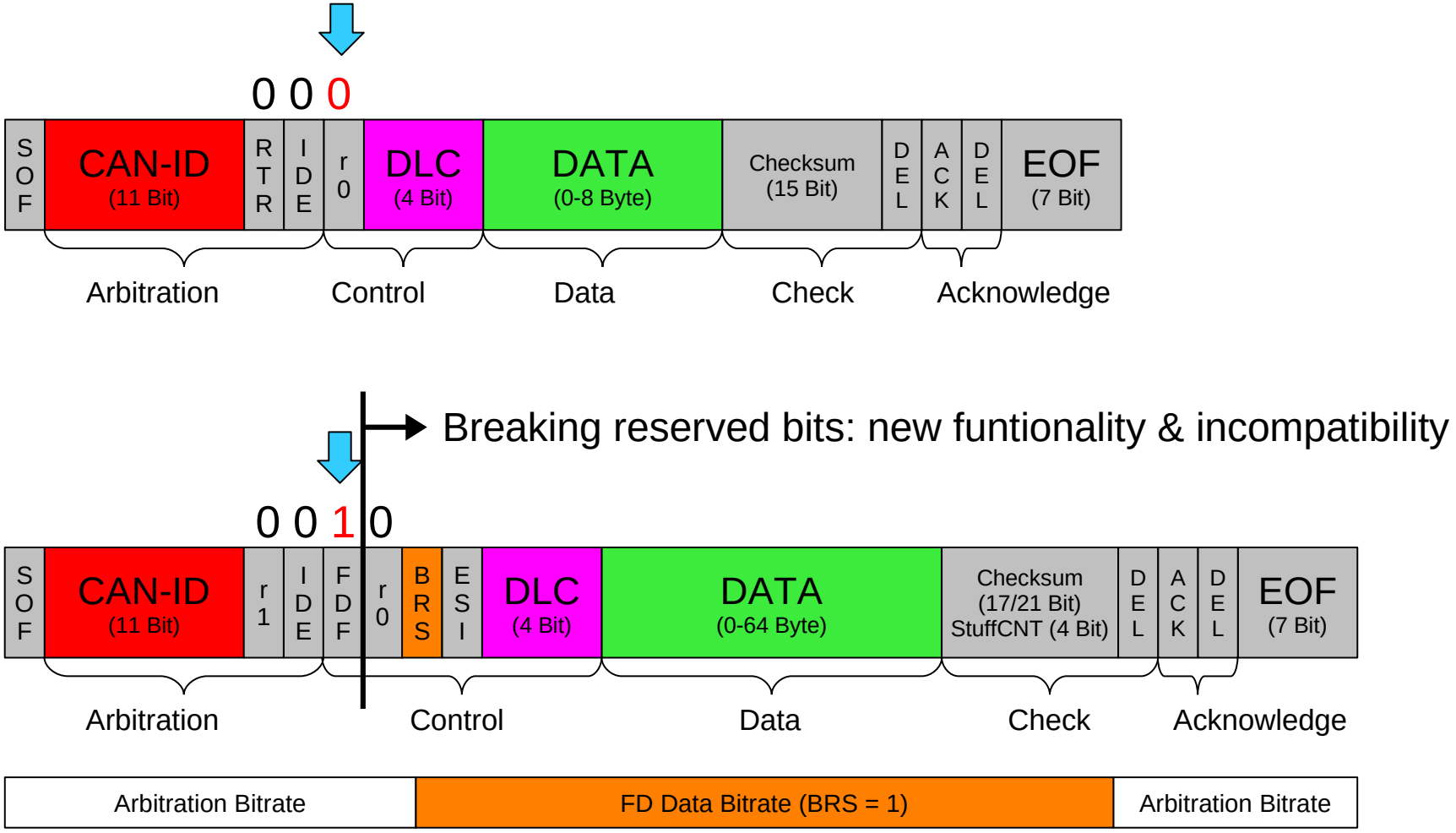
Data acknowledge by receive node

Only acknowledged CAN frames are valid CAN frames.

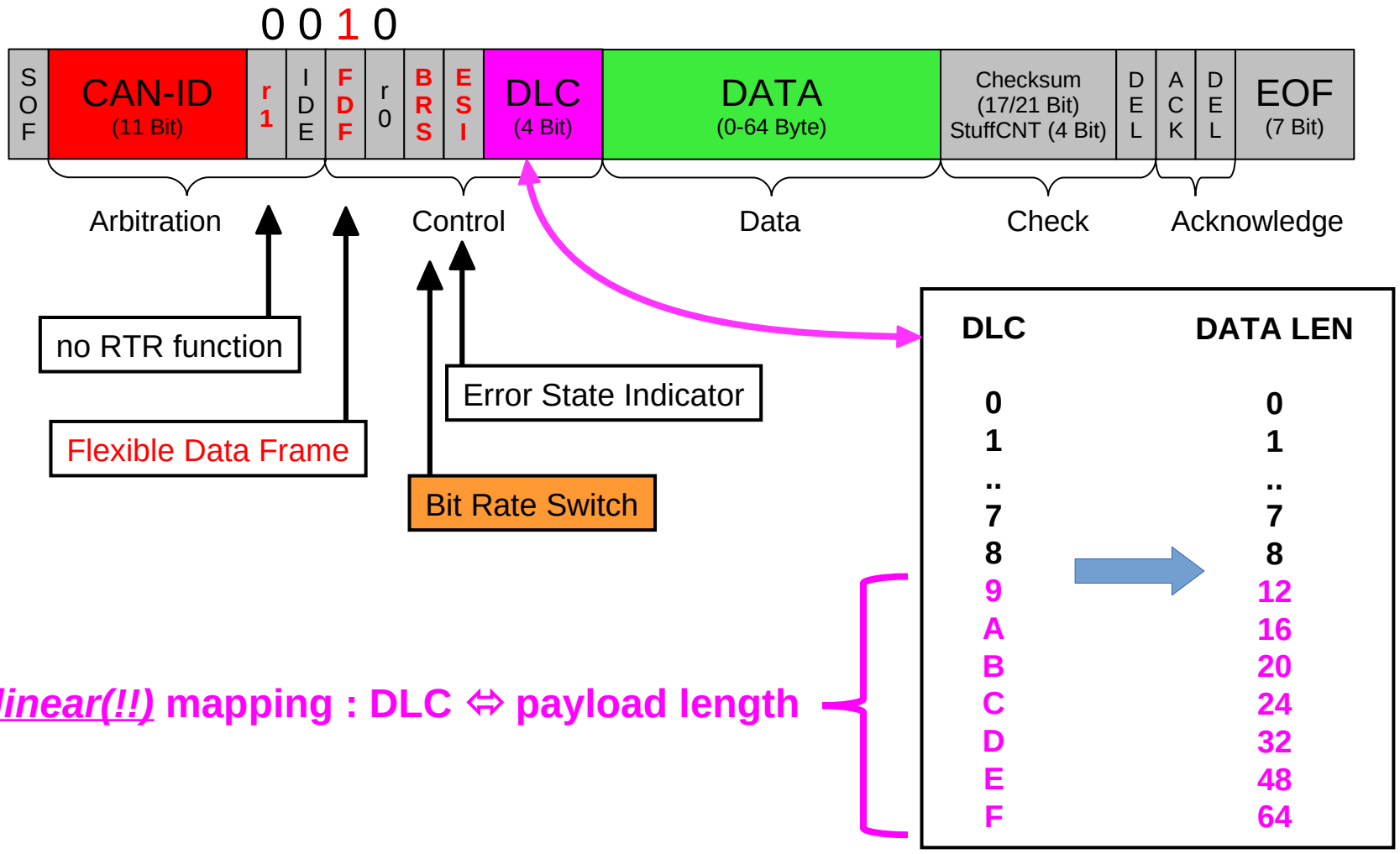


Protocol violations lead to a destroyed CAN frame (error flag)

Switching from CAN 2.0B to CAN FD by using the reserved bit

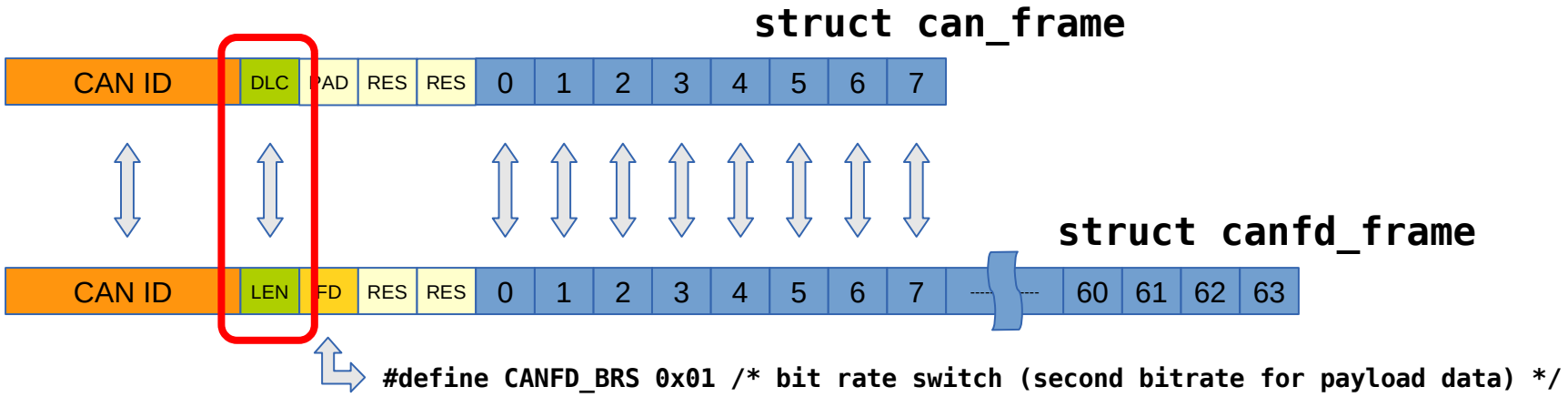


CAN FD – new bits and definitions in detail



Linux CAN FD length information and data structure

- DLC mostly has been used as plain payload length information (1:1 mapping)
- But CAN FD implements a **non-linear length** definition
- Introduce a structure element ‘len’ for CAN FD to preserve common usage
- The mapping of DLC ⇔ LEN and vice versa is done *invisible* in the CAN driver



Compatible data structure layout for CAN2.0B and CAN FD

- CAN2.0B data structure (from 2012)

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

- CAN FD data structure (from 2012)

```
struct canfd_frame {
        canid_t can_id;  /* 32 bit CAN_ID + EFF/RTR/ERR flags */
        __u8    len;     /* frame payload length in byte (0 .. 64) */
        __u8    flags;   /* additional flags for CAN FD */
        __u8    __res0;  /* reserved / padding */
        __u8    __res1;  /* reserved / padding */
        __u8    data[64] __attribute__((aligned(8)));
};
```

Preserve common processing of length information

- Processing length information with CAN data structure

```
struct can_frame cframe;

for (i=0; i < cframe.can_dlc; i++)
        printf("%02X ", cframe.data[i]); /* print payload *
```

- Processing length information with CAN FD data structure

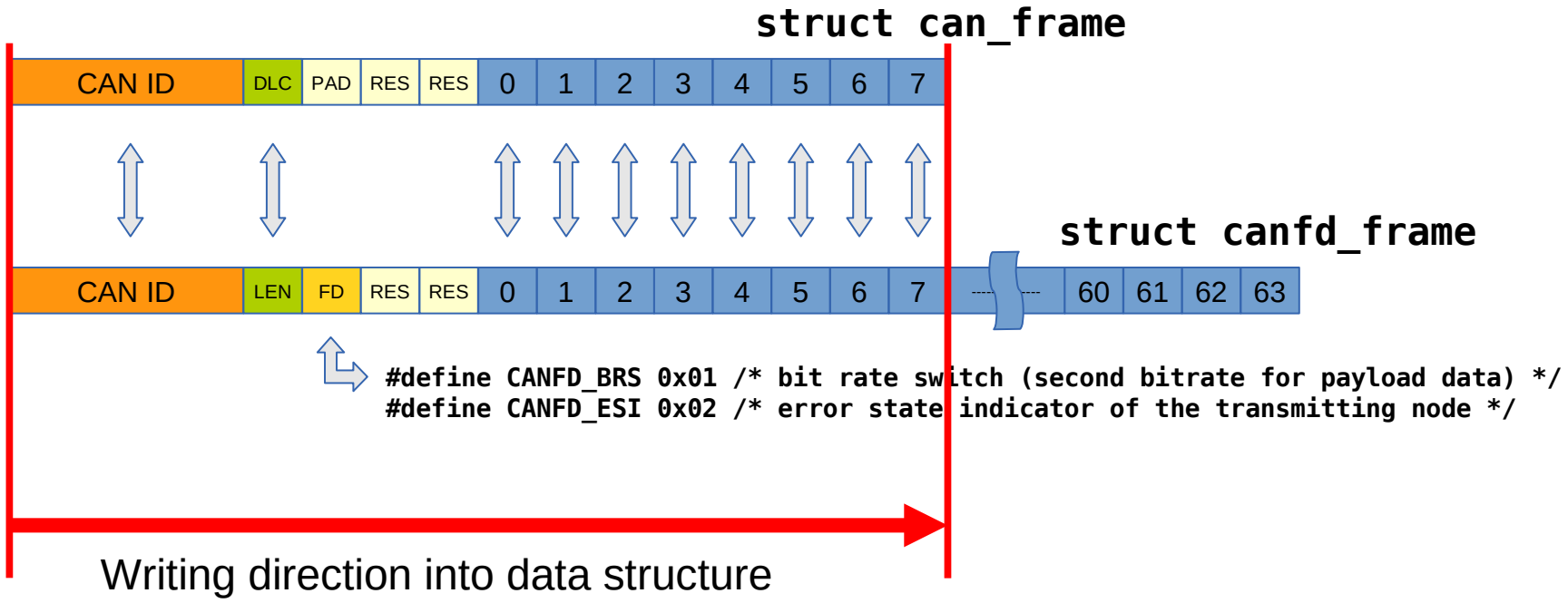
```
struct canfd_frame cframe;

for (i=0; i < cframe.len; i++)
        printf("%02X ", cframe.data[i]); /* print payload *

/* cframe.len = plain data length from 0 to 64 byte */
```

CAN FD data structure – dual use with Classic CAN layout

Writing CAN 2.0B data into a CAN FD data structure creates valid content.



How to activate CAN FD on a CAN_RAW socket

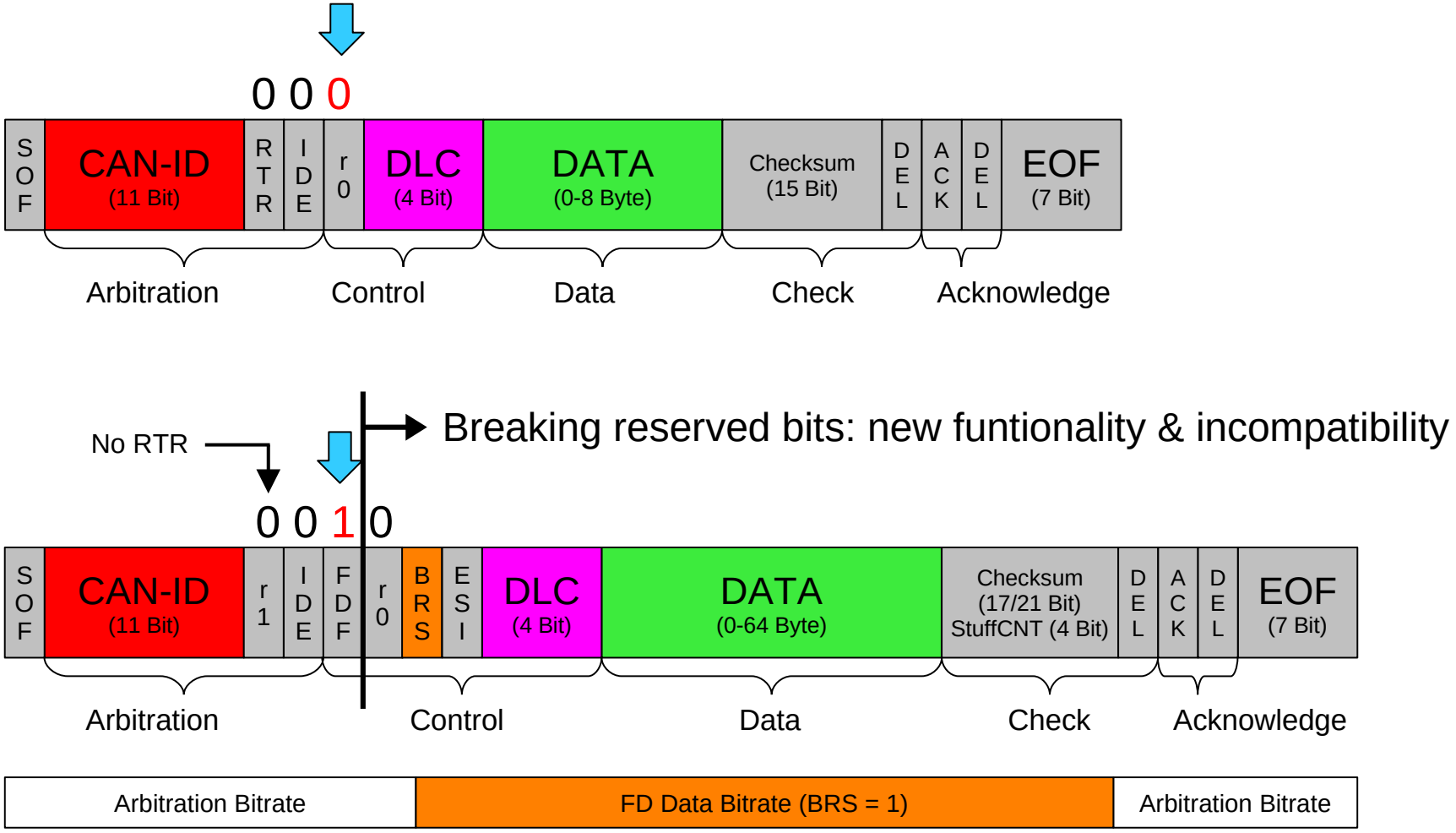
- Reading and writing CAN data structures

```
struct can_frame cframe;
int s = socket(PF_CAN, SOCK_DGRAM, CAN_RAW);
(..)
nbytes = read(s, &cframe, sizeof(struct can_frame));
```

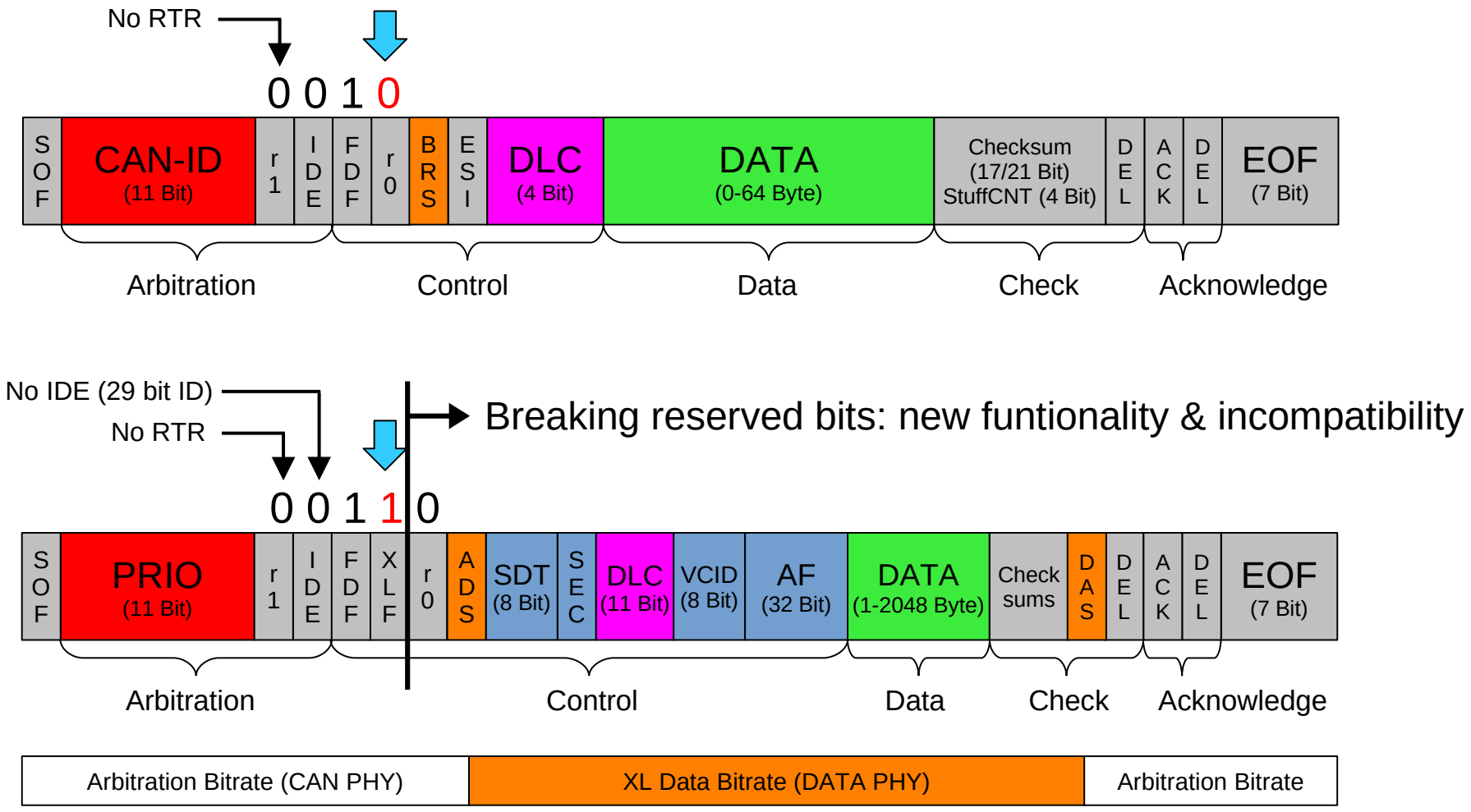
- Switch the socket into CAN FD mode with **setsockopt()** syscall

```
struct canfd_frame cframe;
int s = socket(PF_CAN, SOCK_DGRAM, CAN_RAW);
setsockopt(s, SOL_CAN_RAW, CAN_RAW_FD_FRAMES, ...);
(..)
nbytes = read(s, &cframe, sizeof(struct canfd_frame));
```

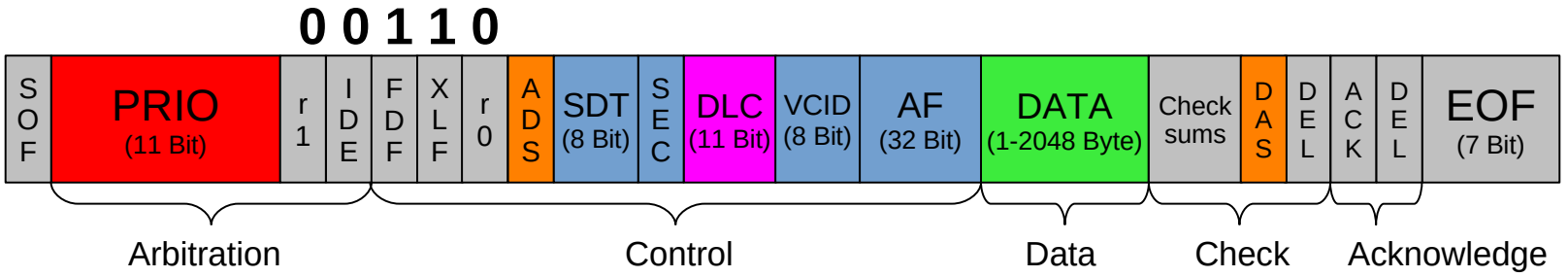
Switching from Classical CAN to CAN FD (recap)



Switching from CAN FD to CAN XL by using the reserved bit



New/changed CAN XL frame content (simplified checksums)



- **PRIO** – 11 bit priority for arbitration (former 11 bit CAN Identifier)
- **ADS/DAS** – switch sequence between arbitration and data phase
- **SDT** – 8 bit SDU (service data unit) type
- **SEC** – 1 bit simple extended content (e.g. for security/segmentation)
- **DLC** – 11 bit data length code (0 .. 2047) => 1 .. 2048 data bytes
- **VCID** – 8 bit virtual CAN network identifier (analogue to VLAN identifier)
- **AF** – 32 bit acceptance field (function depending on SDT)
- **DATA** – 1 .. 2048 data bytes (depenting on DLC value)

Data structure layout for CAN XL with 16 bit length information

- CAN FD data structure

```
struct canfd_frame {
        canid_t can_id;  /* 32 bit CAN_ID + EFF/RTR/ERR flags */
        __u8    len;     /* frame payload length in byte (0 .. 64) */
        __u8    flags;   /* additional flags for CAN FD */
        __u8    __res0;  /* reserved / padding */
        __u8    __res1;  /* reserved / padding */
        __u8    data[64] __attribute__((aligned(8)));
};
```

- CAN XL data structure (wip, VCID is stored in socket buffer analogue eth VID)

```
struct canxl_frame {
        canid_t prio;    /* 11 bit priority for arbitration (canid_t) */
        __u8    flags;   /* additional flags for CAN XL */
        __u8    sdt;     /* SDU (service data unit) type */
        __u16   len;     /* frame payload length in byte (1 .. 2048) */
        __u32   af;      /* acceptance field */
        __u8    data[2048];
};
```

CAN XL frame content: SDU (service data unit) type

SDU type (0x00 .. 0xFF)

- 0x00 : reserved
- 0x01 : content based addressing (AF = PDU Identifier)
- 0x02 : node addressing (AF = DST/SRC address)
- 0x03 : Classical CAN / CAN FD mapped tunneling (AF = CAN ID)
- 0x04 : IEEE 802.3 (Ethernet) tunneling (DATA = ethernet MAC frame)
- 0x05 : IEEE 802.3 (Ethernet) mapped tunneling (VCID = lower 8 bit VID, AF = MAC truncated dest addr, DATA = ethernet MAC frame)
- 0x06 .. 0xDF : reseved for future use
- 0xE0 .. 0xFE : manufacturer specific
- 0xFF : reserved

CAN XL summary

- CAN XL is like „ethernet with CSMS/CR arbitration“
- High reliability (CRC/bitstuffing) with 10 Gbit/s data rate
- CAN XL controller support CAN XL / CAN FD / Classical CAN
- CAN XL transceivers with switchable physical layer (arbitration/data)
- Virtual CAN interface identifiers (analogue ethernet VLANs)
- SDU (service data unit) types for multiple content use-cases