Introduction to CAN protocol

Carlos Calvillo

November 2022 / v1.1

Contents

Contents

CAN Overview

CAN stands for "Controller Area Network"

- **Asynchronous** serial communication protocol
- Efficient support for distributed **real-time control** systems
- High **reliability**
- Bit-rates:
  - **Classic CAN** → Up to **1 Mbit/s**
  - CAN **FD** → Up to 5 Mbit/s

CAN Overview

CAN stands for "Controller Area Network"

- **Asynchronous** serial communication protocol
- Efficient support for distributed **real-time** control systems
- High **reliability**
- Bit-rates:
  - **Classic CAN** → Up to **1 Mbit/s**
  - CAN **FD** → Up to 5 Mbit/s

Focus in this presentation

CAN Overview

Increase in Electronic Contents
Increase in Electronic Complexity

- 1980 -
10%
- Electronic Fuel Injection
- Anti-lock Brake System

- 2000 -
22%
- Airbag
- Electronic Stability Control
- Body Electronics
- Navigation System

- 2010 -
35%
- Advanced Driver Assistance
- Active/Passive Safety
- Green Powertrain
- Car-to-cloud connectivity
- Smartphone integration

- 2020 -
45%
- Hybrid-Electric Vehicle
- Full Electric Vehicle
- Autonomous driving

CAN Overview

Brief CAN **history**

1986: Bosch develops CAN protocol as a solution to traditional wiring

1987: First CAN controller chips (Intel 82526, Philips 82C200)

1991: Bosch specification CAN 2.0 (CAN 2.0A: 11 bit, 2.0B: 29 bit)

1993: CAN is standardized in **ISO 11898**

2012: Bosch released the CAN FD 1.0

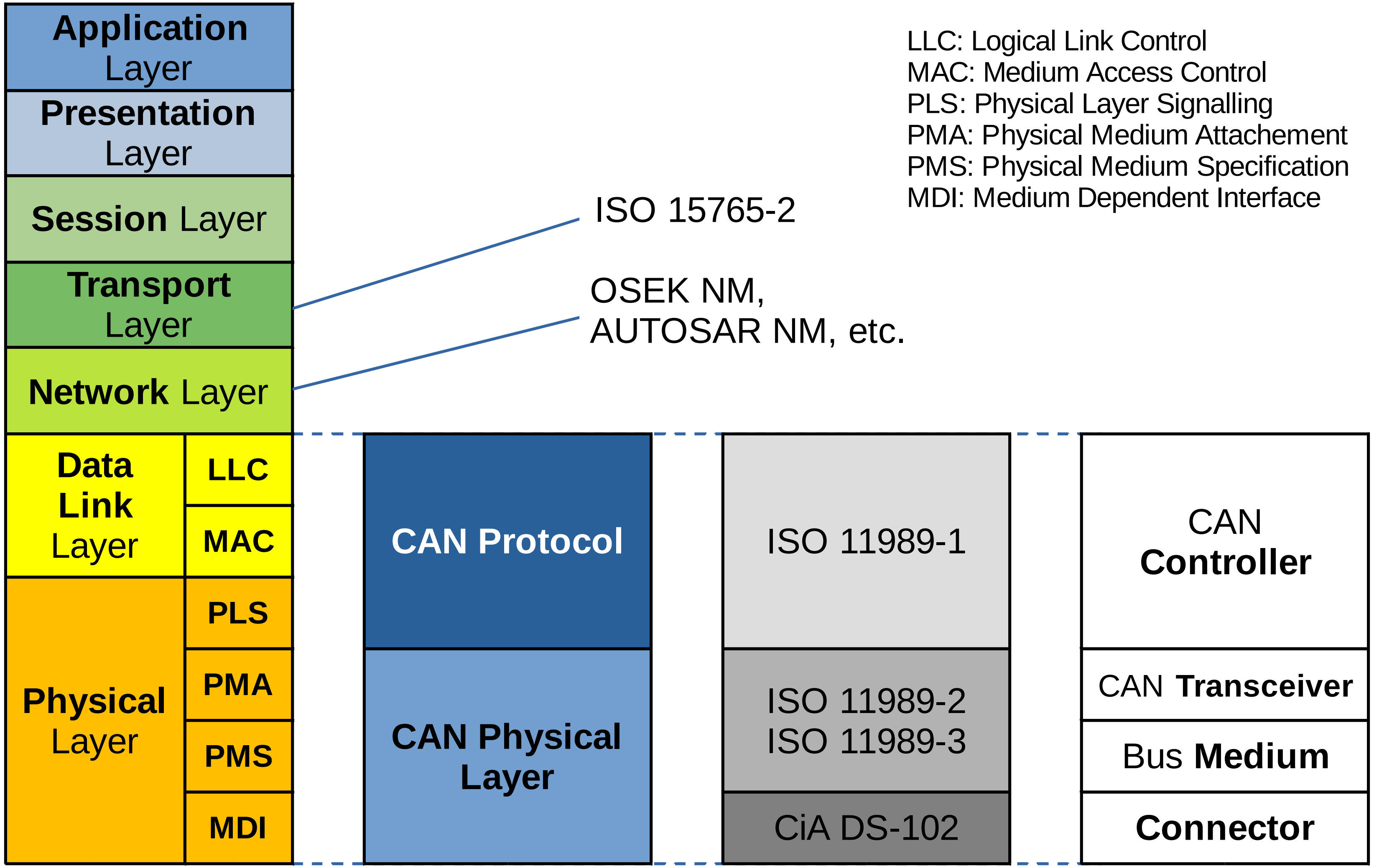
2015: The CAN FD protocol is standardized within ISO 11898

2016: The physical CAN layer for data-rates up to 5 Mbit/s standardized in ISO 11898-2

time

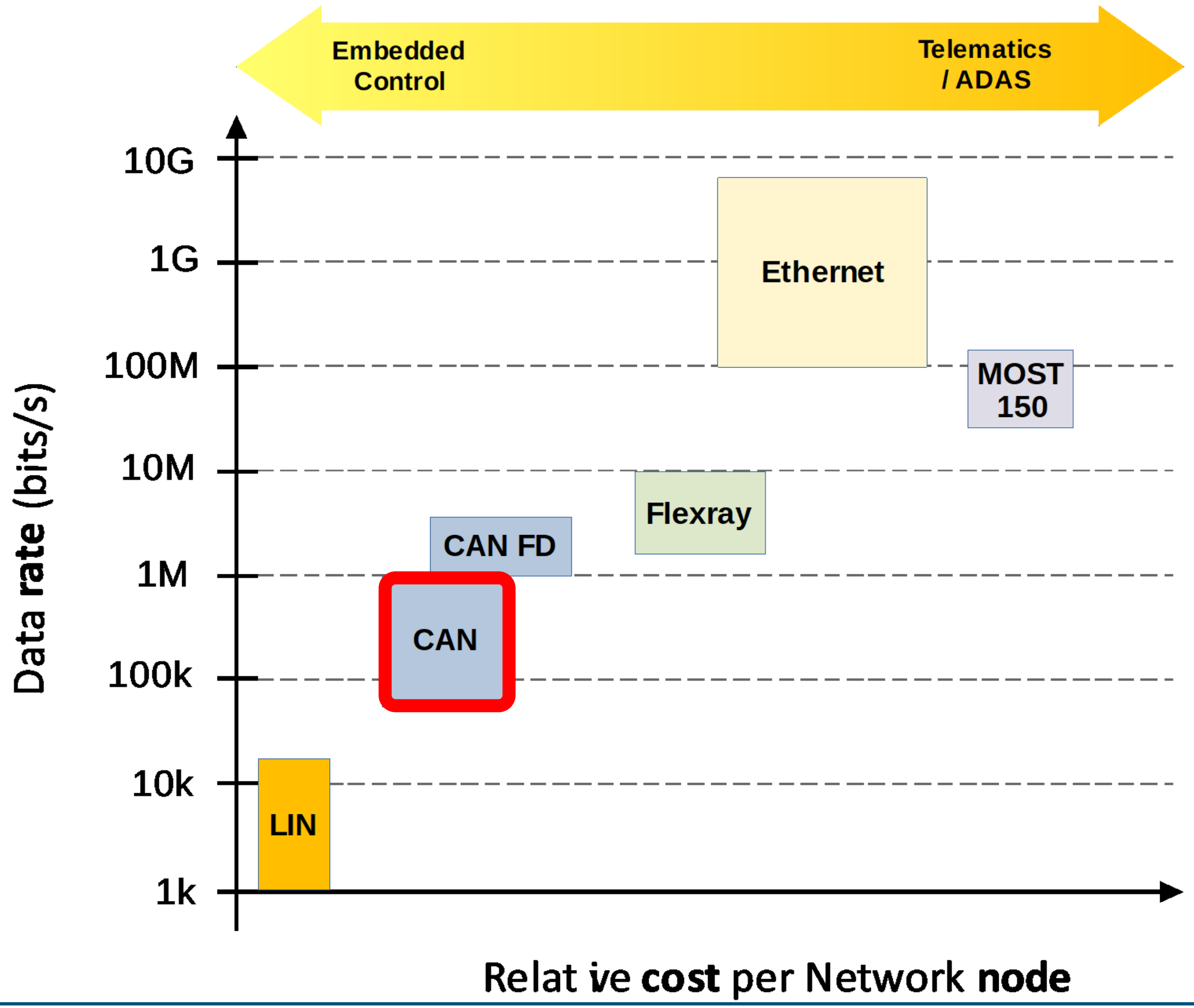
CAN Overview

LLC: Logical Link Control
MAC: Medium Access Control
PLS: Physical Layer Signalling
PMA: Physical Medium Attachement
PMS: Physical Medium Specification
MDI: Medium Dependent Interface



CAN Overview

CAN in **context** with other protocols used in automotive



CAN Overview

- CAN for **distributed** architectures
  - **Bus** network topology
  - Good bandwidth for **control-oriented** applications
  - **Multi-master** protocol
  - **Deterministic** communication
  - Hot-in-out of nodes in network (no need for restarting the network)
  - **Cost** efficient

CAN Overview

- CAN Network classifications
  - **Low speed**
    - 83.3 kbit/s, 125 kbit/s
    - Two-wire or single-wire bus possible (SAE J2422)
    - Common in early vehicles using CAN
  - **High speed**
    - 250 kbit/s, **500 kbit/s,** 1 Mbit/s
    - **Two-wire** bus
    - Most used in modern vehicles

CAN Overview

- CAN Network classifications
  - **Low speed**
    - 83.3 kbit/s, 125 kbit/s
    - Two-wire or single-wire bus possible (SAE J2422)
    - Common in early vehicles using CAN
  - **High speed**
    - 250 kbit/s, **500 kbit/s,** 1 Mbit/s
    - **Two-wire** bus
    - Most used in modern vehicles

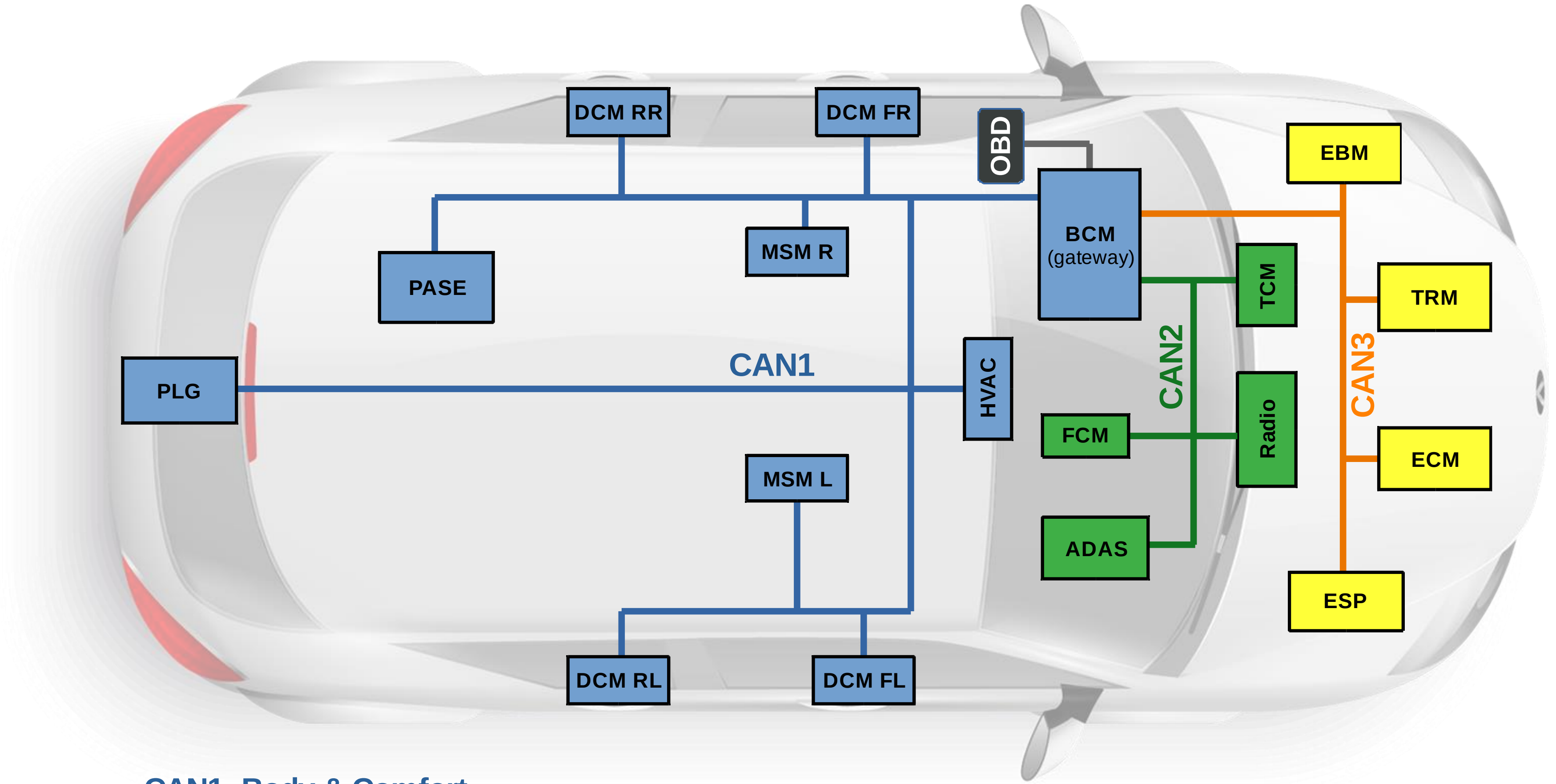
Focus in this presentation

CAN Overview

- Typical vehicle has multiple CAN networks
  - Networks for **powertrain**, **telematics**, **body** control, etc.
  - Each network with multiple **ECUs** (Electronic Control Units)
  - “**Gateway**” ECU:
    - Provides interconnection between different CAN networks

      → By transporting messages from one network to another…
    - Provides connectivity to external **diagnostic** tools (e.g., via OBD connector).

CAN Overview



CAN1: Body & Comfort
CAN2: ADAS & Telematics
CAN3: Powertrain

CAN Overview

~**80%** of CAN applications are in **automotive** industry; however, CAN is also used in other application areas:

- Agricultural Vehicles
- Medical Equipment
- Robotics
- Industrial Applications, etc.




Contents

Protocol Specification

- CAN uses Non-Return-to-Zero (**NRZ**) bit coding
  - Bus level is kept constant for each bit-time (no transition within a single bit)

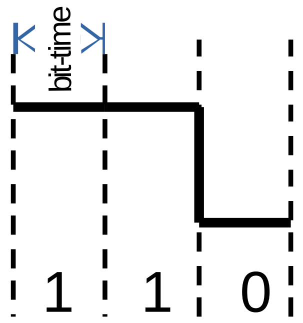
✓Less transitions (as compared to other bit codings) → higher bandwidth

✓Reduced electro-magnetic emissions

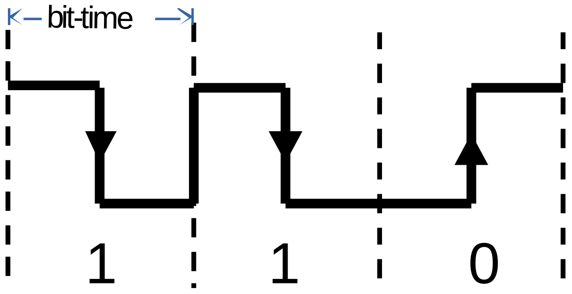
X Receiver may lose synchronization with long sequence of bits with same value

→ *Bit-stuffing technique mitigates this situation*

Example **NRZ**:



Example Manchester Encoding:



Protocol Specification

- **Physical Layer**
  - **Differential** voltage transmission (**two-wires**)
    - CAN High (**CAN-H**), CAN Low (**CAN-L**)
    - Reference to common GND
  - Unshielded Twisted Pair
  - Termination **resistor** at each end of the CAN Bus → **120 Ω**

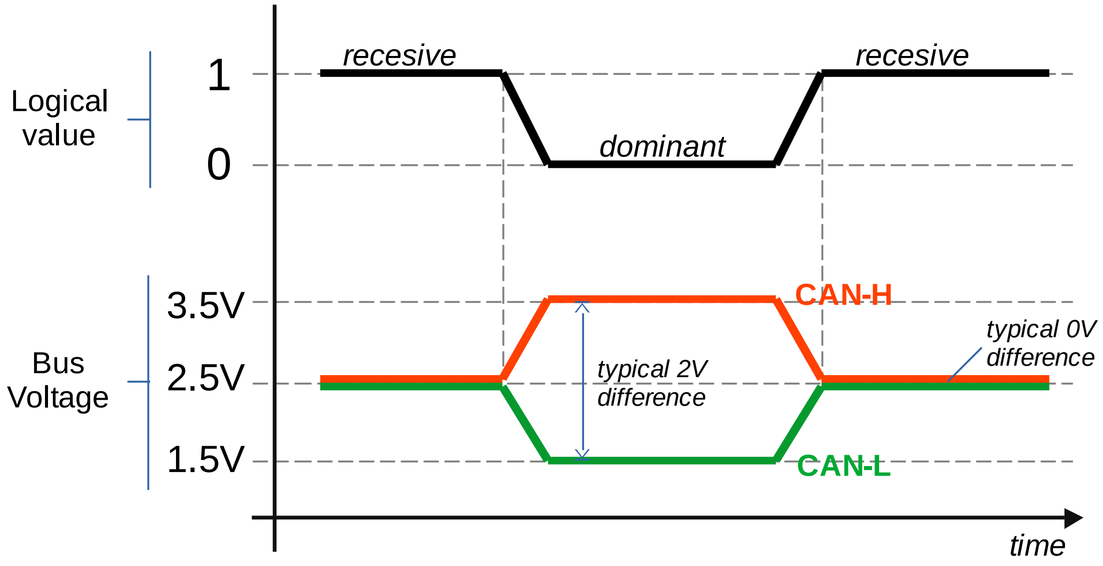
    → Allow power dissipation for generation of “recessive” levels

    → Prevent reflections
  - Line resistance: less than **60 Ω**

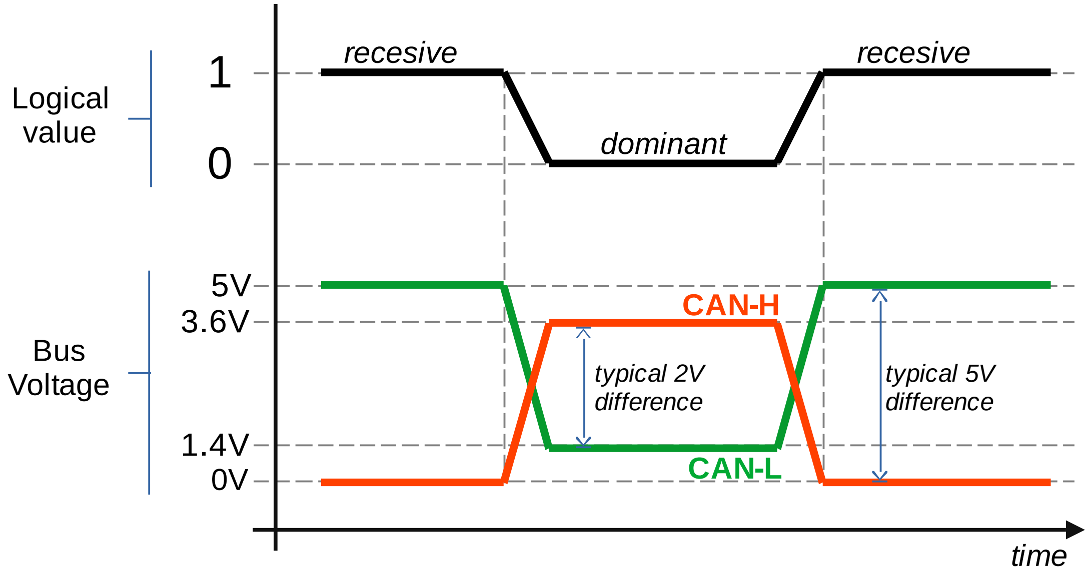
Protocol Specification

- **Physical Layer**

ISO 11989-2: **High** Speed CAN



ISO 11989-3: **Low** Speed CAN



Protocol Specification

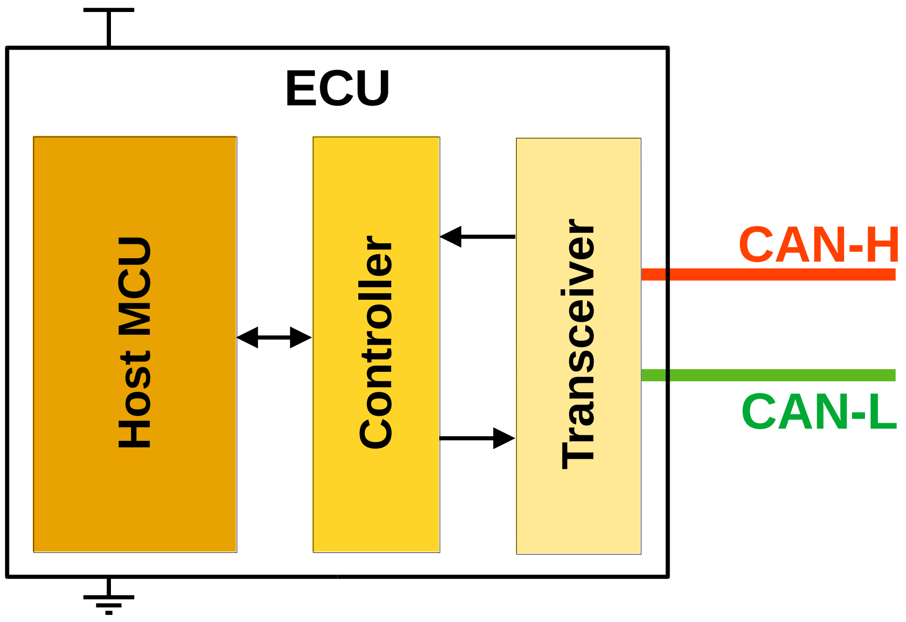
- **Physical Layer**
  - CAN Bus **Length**

Bit Rate	Max. recommended bus length
50 kbit/s	1000 m
125 kbit/s	500 m
250 kbit/s	200 m
500 kbit/s	100 m
1 Mbit/s	40 m

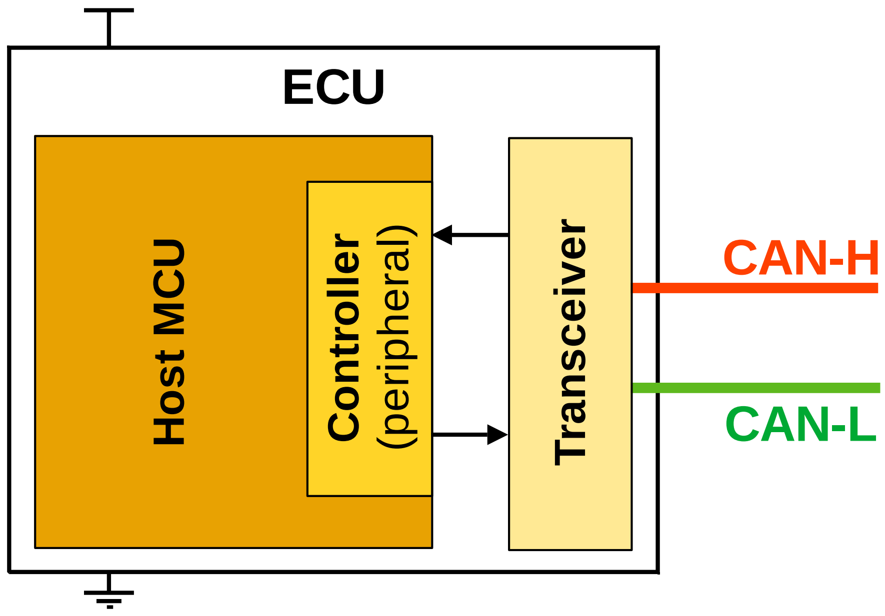
Protocol Specification

- CAN **Transceiver,** CAN **Controller,** CAN **Host**

Stand-alone CAN controller



CAN controller as MCU **peripheral**



Protocol Specification

- **CAN transceiver**
  - Allows connection of node (ECU) to the CAN bus
  - Interfaces between CAN bus voltage levels (CAN-H and CAN-L) and CAN controller voltage levels (e.g., RXD and TXD pins at TTL voltage levels).

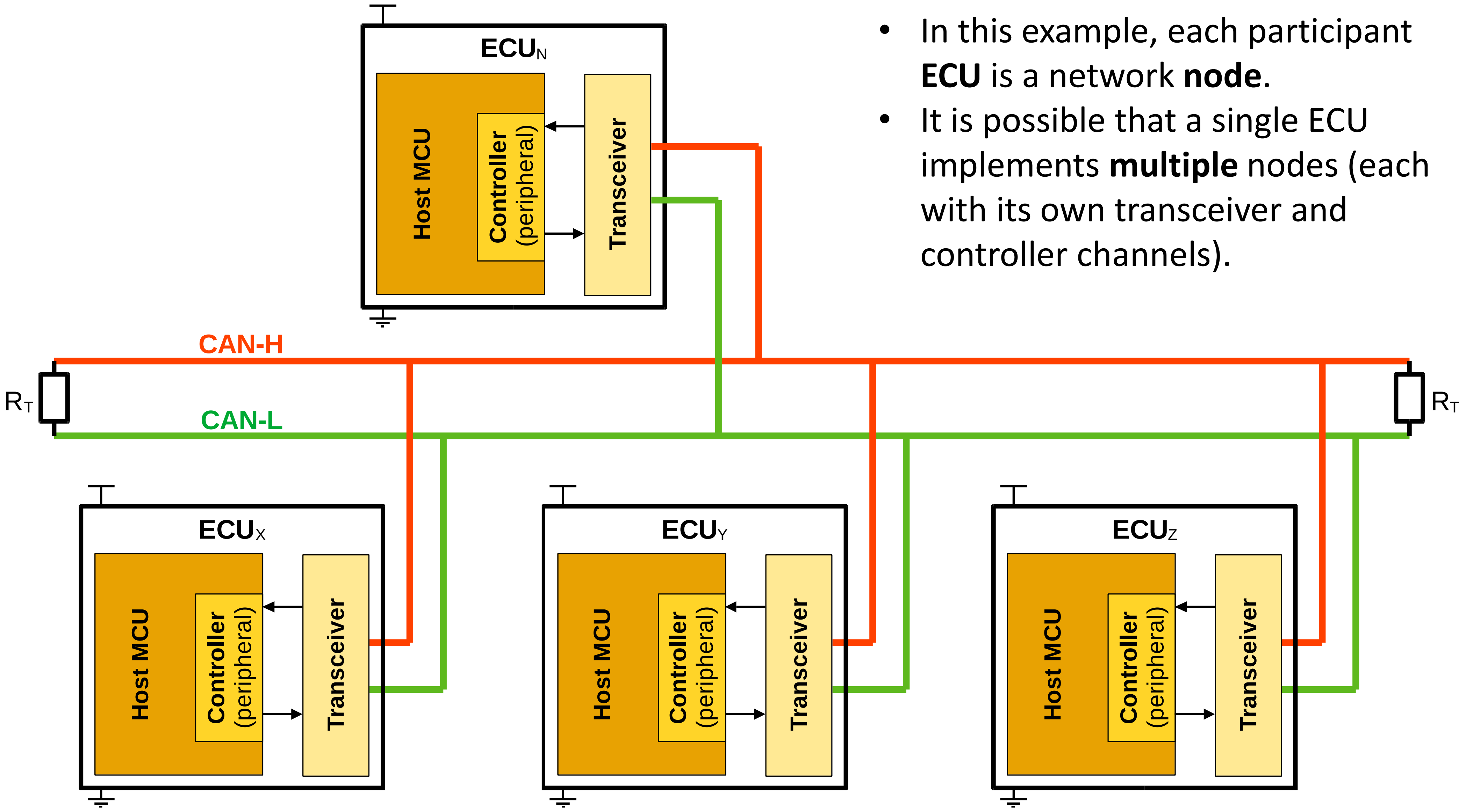
Protocol Specification

- **CAN controller**
  - HW device that handles the CAN messages → decodes the CAN protocol
  - Can be a **standalone** device, e.g.:
    - Microchip **MCP2515** (SPI interface with Host MCU)
  - Can be integrated in the host MCU as a **peripheral** , e.g.:
    - NXP **FlexCAN** peripheral
    - Infineon **MultiCAN** peripheral

Protocol Specification

- **CAN host**
  - Device that host the **application** (an MCU, a computer)
  - Consumes the CAN messages and uses their conveyed signals
  - Generates and packs signals into CAN messages for subsequent transmission in the CAN bus

Protocol Specification

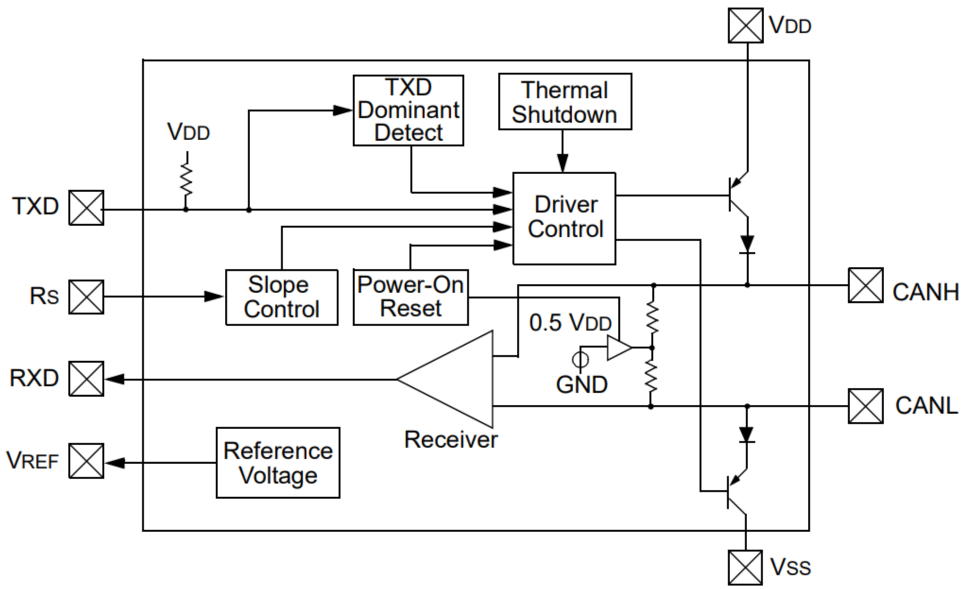


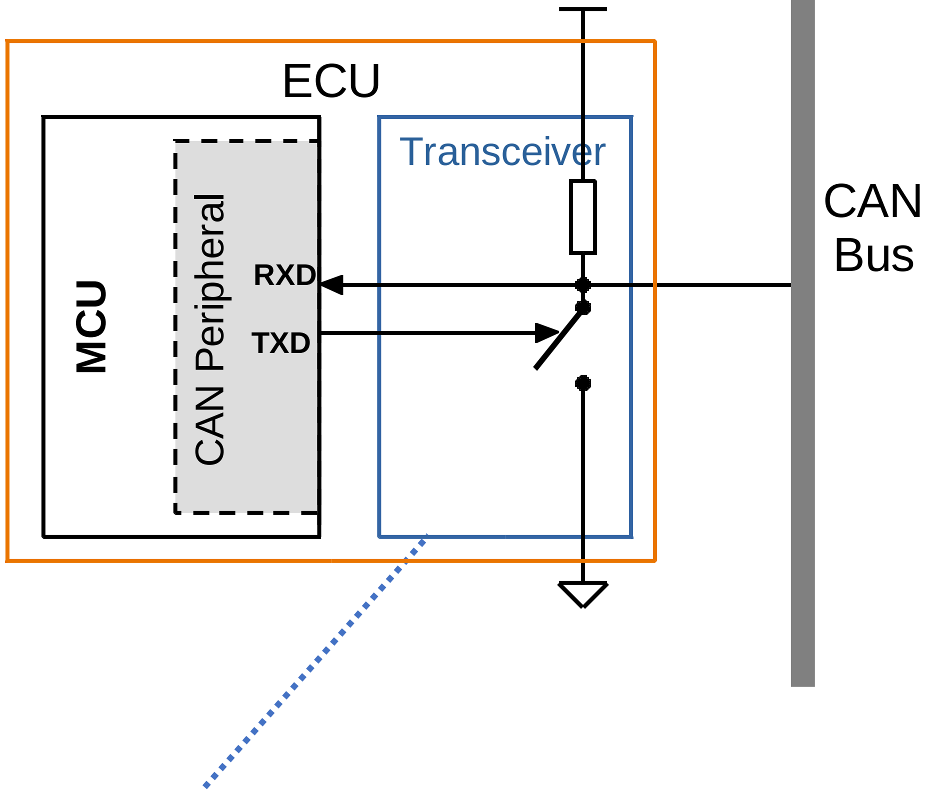
R_T = Termination Resistor

- In this example, each participant **ECU** is a network **node**.
- It is possible that a single ECU implements **multiple** nodes (each with its own transceiver and controller channels).

Protocol Specification

CAN Transceiver
(From Microchip MCP2551)

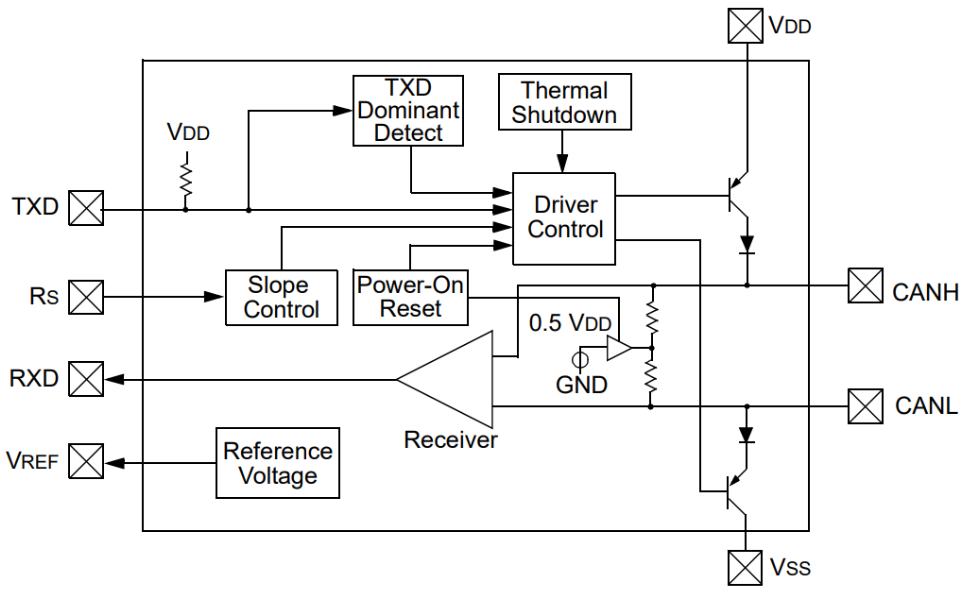




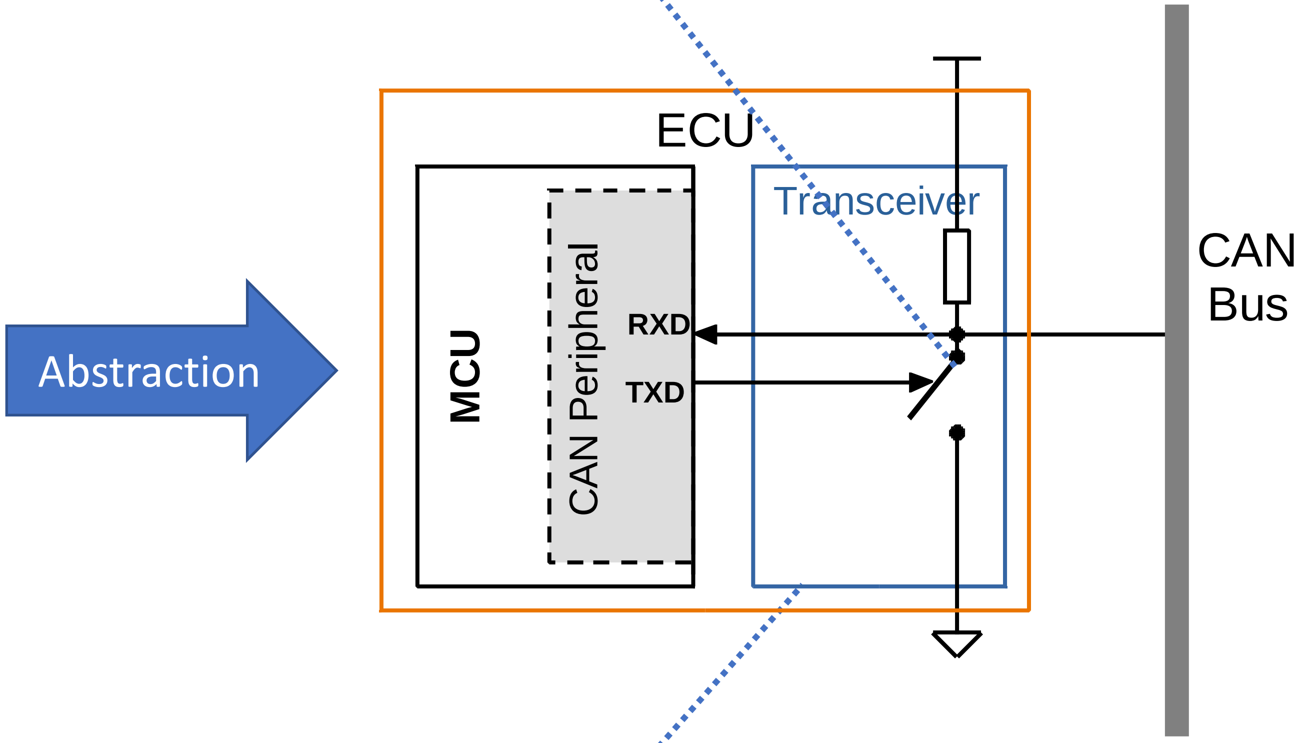
Transceiver **abstraction**

Protocol Specification

CAN Transceiver
(From Microchip MCP2551)

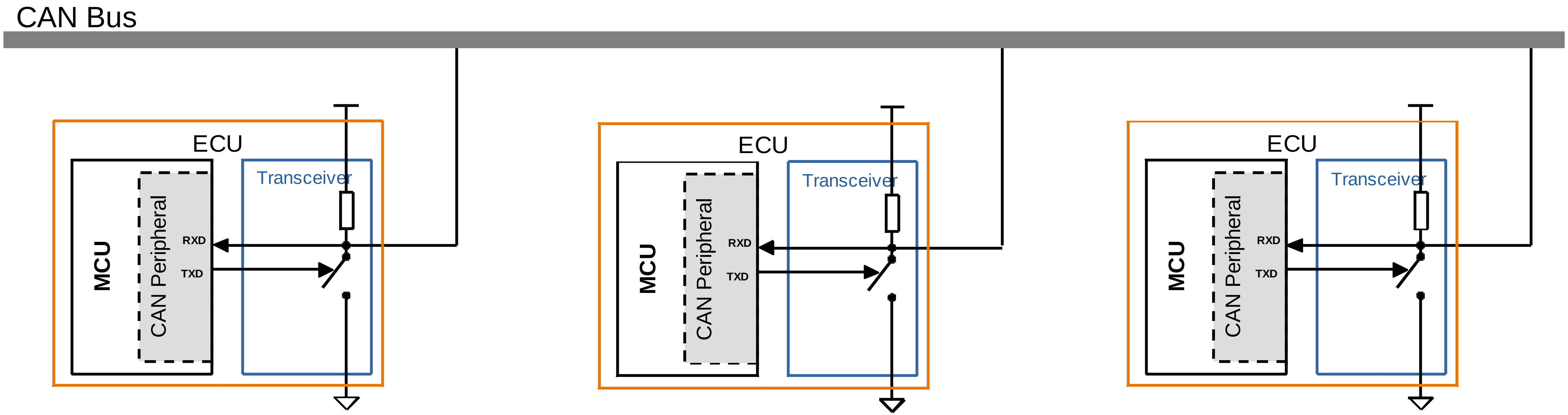


Bus goes to **dominant** level when "switch" **closes**



Transceiver **abstraction**

Protocol Specification



***Note**: This is just an abstraction used for illustrating the dominant and recessive levels behavior.*
Not intended to show actual circuitry of physical layer

Protocol Specification

- **Recessive bus level**
  - Logical **one** is known as **recessive** level
  - When bus is **idle**, it shall be in recessive level
- **Dominant bus level**
  - Logical **zero** is known as **dominant** level
  - If **any** node publishes a **dominant** value, **the whole bus** goes to **dominant** level even if another node is publishing a recessive value at the same time

Protocol Specification

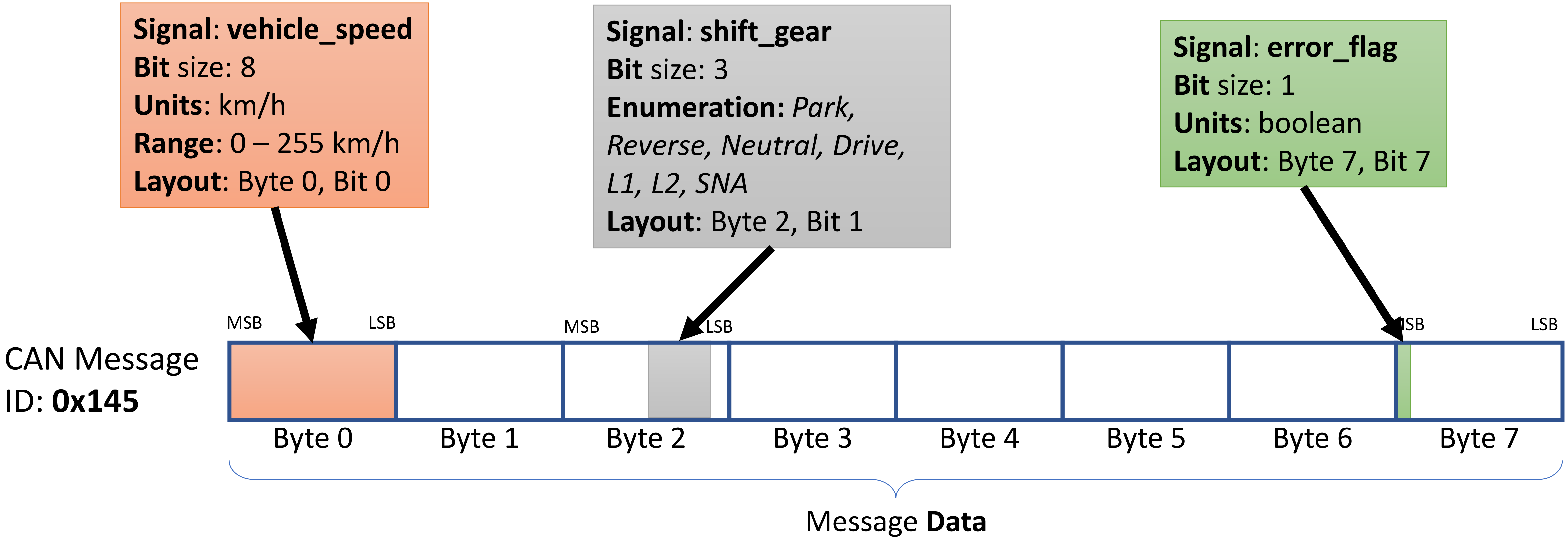
- **Communication Principle**
  - **Carrier-sense, multiple-access with collision detection (CSMA/CD)**
    - **Carrier-sense**: all CAN nodes are sensing the bus, e.g., to see if it is idle
    - **Multiple-access**: CAN is a multi-master protocol, i.e., any node can start transmission if bus is idle
    - **Collision detection:**
      - CAN defines an arbitration technique for detecting collisions when multiple nodes attempt to publish a frame at the same time
      - Non-destructive arbitration: In case of collision, one node always wins the arbitration and can publish its frame

Protocol Specification

- **Communication Principle**
  - **Message based** communication
    - Data signals are packed into **messages** (*for now message = frame*)
    - Message can carry from **0 to 8 data bytes** (single frame)
    - **Unique identifier** (ID) is defined for each message
      - ID can be of **11 bits** (standard) or **29 bits** (extended)
    - Direction of data transfer is defined (**fixed**) per each message:
      - Each message shall have <u>**only one publisher**</u> **node** (transmitter).
      - Each message can be assigned to <u>**one or more subscriber**</u> **nodes** (receivers)

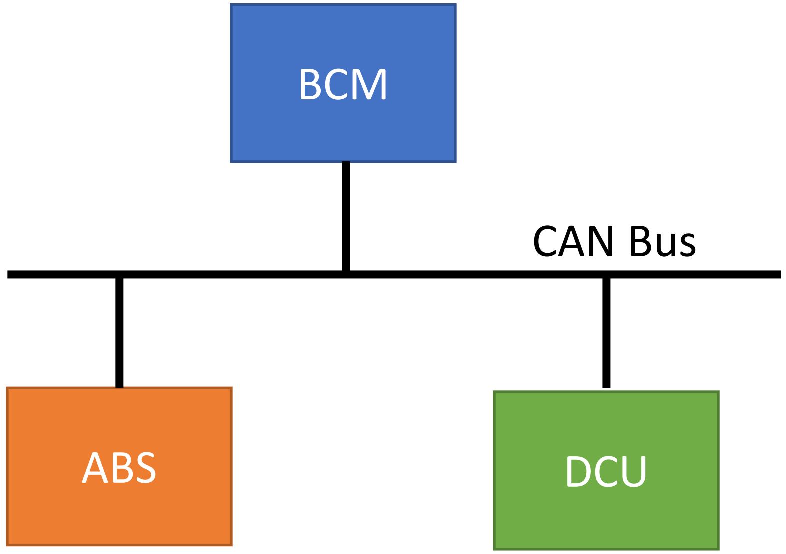
Protocol Specification

- **Communication Principle**
  - Signals are “*packed*” into messages, example:



Protocol Specification

- **Communication Principle**
  - Example of “Message matrix”



Message Name	Message ID	Publisher Node	Subscriber(s) node(s)
BCM_MSG1	**0x145**	BCM	ABS, DCU
BCM_MSG2	**0x150**	BCM	DCU
ABS_MSG1	**0x020**	ABS	BCM
DCU_MSG1	**0x210**	DCU	BCM, ABS
DCU_MSG2	**0x213**	DCU	BCM

Protocol Specification

- **Communication Principle**
  - **Acceptance Filters**
    - All nodes are "*capable*" of seeing all published messages, however, that doesn't mean that all messages are of interest for all nodes.
    - CAN controllers are capable of **filtering** messages not subscribed by the node to reduce SW CPU load.
      - **Acceptance** filters by HW may filter by individual IDs, ranges of IDs, ID masks, etc.
      - Additional SW filtering performed if HW filtering didn't reject 100% of non-subscribed messages.

Protocol Specification

- **CAN Frame**
  - Frames are the **basic units** of transmission/reception in CAN protocol
  - CAN defines different frame formats to convey data, errors, etc.
  - Data is transmitted in **Data Frames**

Protocol Specification

- Classic CAN Frame Types
  - **Data** Frame
    - *Standard* **Data** Frame
    - *Extended* **Data** Frame
  - **Remote** Frame
    - *Standard* **Remote** Frame
    - *Extended* **Remote** Frame
  - **Error** Frame
    - **Active Error** Frame
    - **Passive Error** Frame
  - **Overload** Frame

Protocol Specification

- Classic CAN Frame **Types**
  - **Data** Frame
    - *Standard* **Data** Frame
    - *Extended* **Data** Frame
  - **Remote** Frame
    - *Standard* **Remote** Frame
    - *Extended* **Remote** Frame
  - **Error** Frame
    - **Active Error** Frame
    - **Passive Error** Frame
  - **Overload** Frame

Data Frame Format

- **Data Frame**
  - Frames intended for **publishing data**
  - **Most used** frames in CAN networks
  - Two variants:
    - **Standard** Data Frame → 11-bit message identifier
    - **Extended** Data Frame → 29-bit message identifier

Data Frame Format

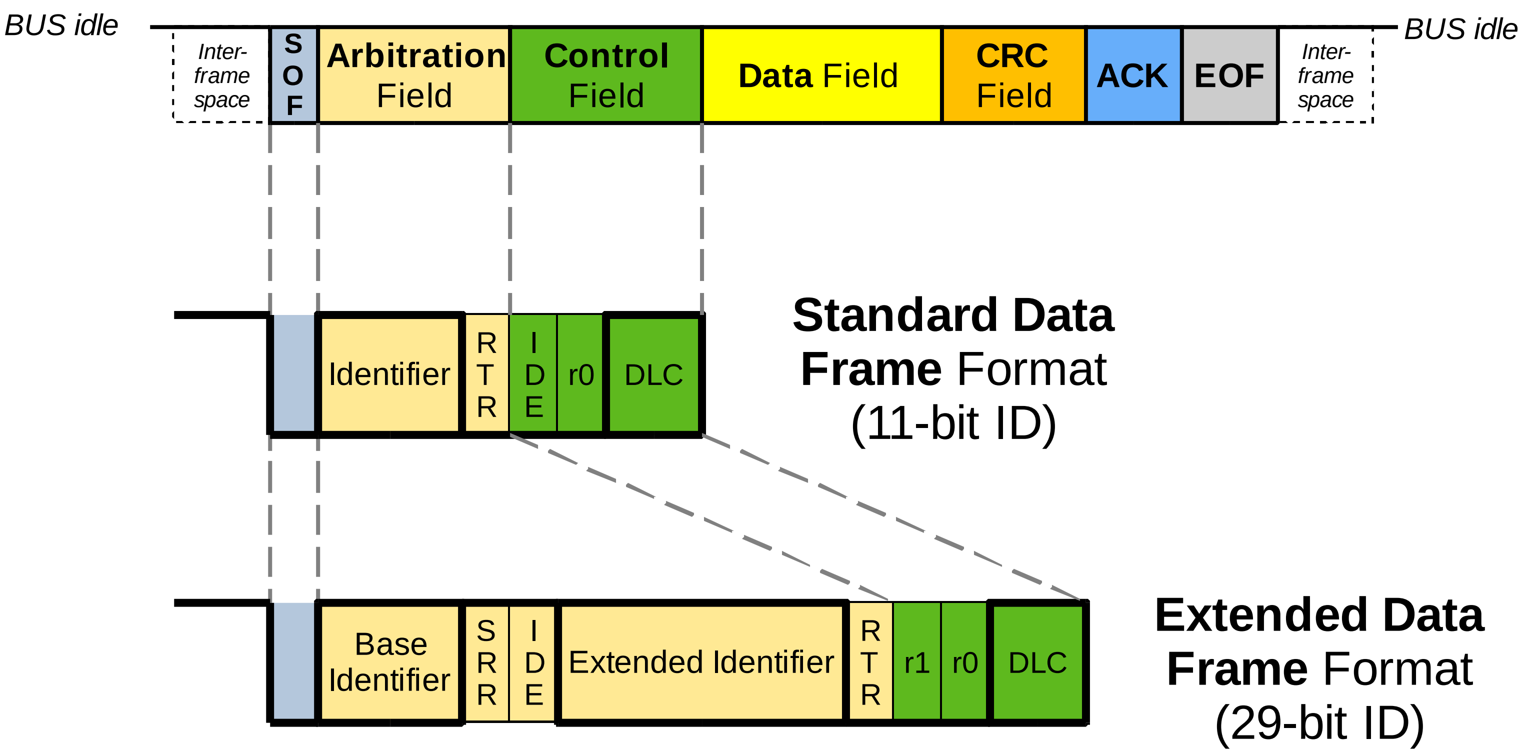
Data Frame **Fields:**

- Start-of-Frame (**SOF**)
- **Arbitration**
- **Control**
- **Data**
- **CRC**
- Acknowledge (**ACK**)
- End-of-Frame (**EOF**)

Inter-Frame Space: Mandatory space between subsequent frames

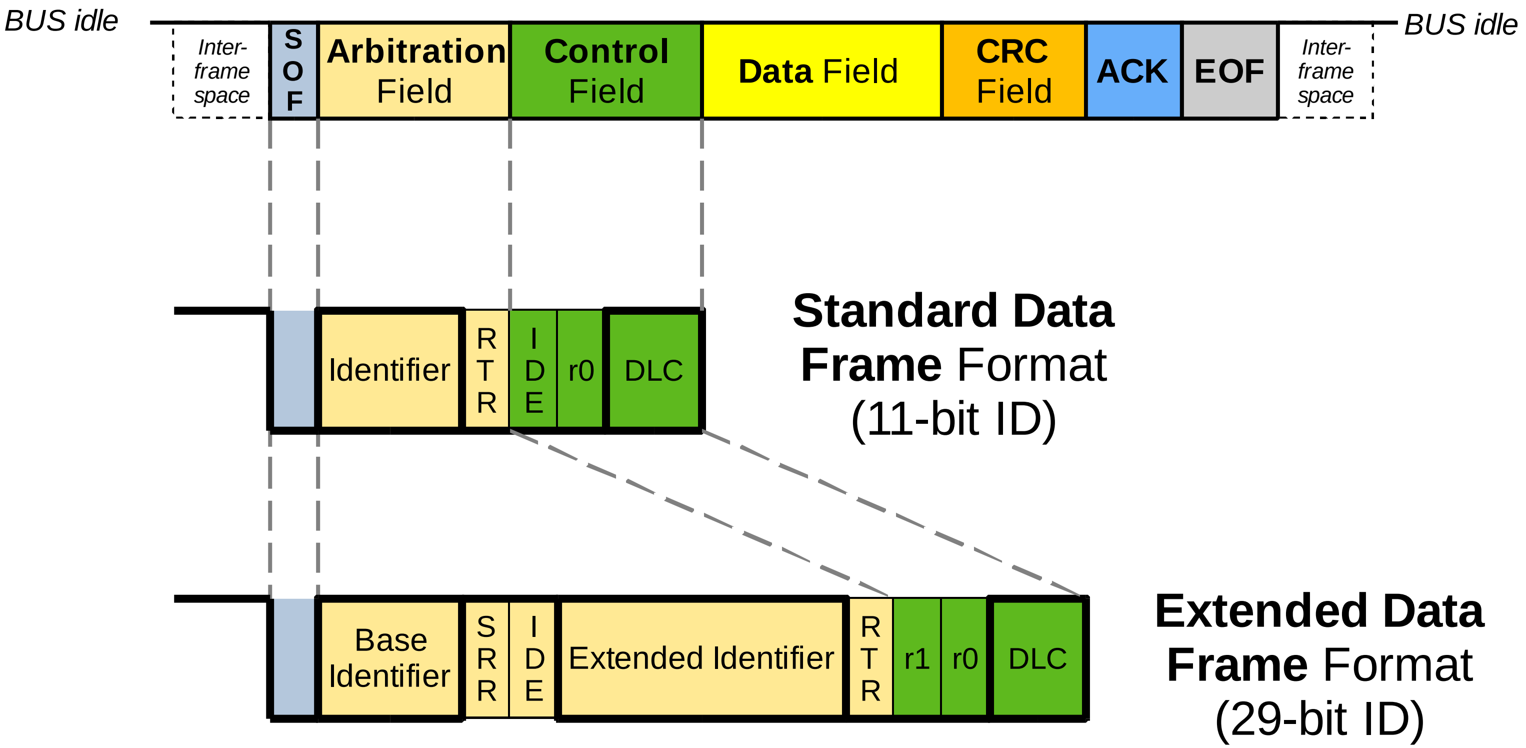
Bus IDLE: bus is considered **idle** if at least **11 bits** with **recessive** value are observed

Data Frame Format



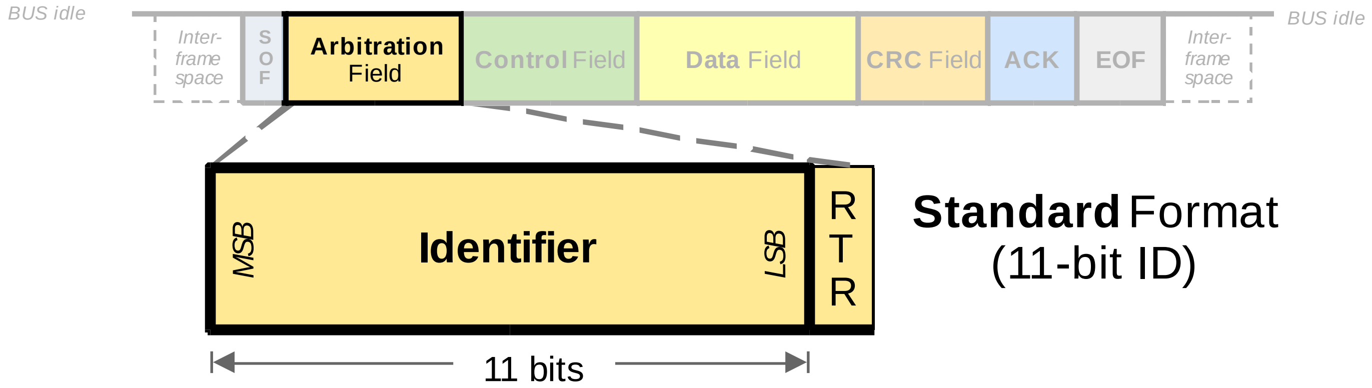
- **Start-of-Frame (SOF)**
  - Single bit with dominant level
  - Indicates the beginning of a new frame

Data Frame Format



- **Standard Format →** Message identifier of 11 bits (2^{11} different messages)
- **Extended Format →** Message identifier of 29 bits (2^{29} different messages)

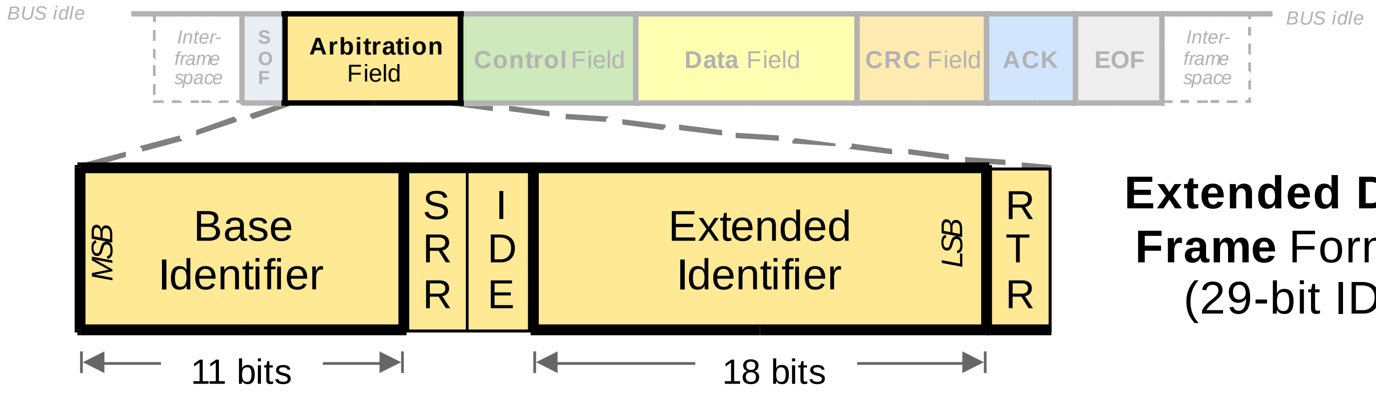
Data Frame Format



Standard Format
(11-bit ID)

- **Identifier:** Used to uniquely identify each message in the network
  - 11-bits for **standard** format frames
- **Remote Transmission Request (RTR)**: 1-bit; indicates if current frame is a data frame (provides data) or a remote frame (requests data)
  - **Dominant** value for **data** frames
  - **Recessive** value for **remote** frames

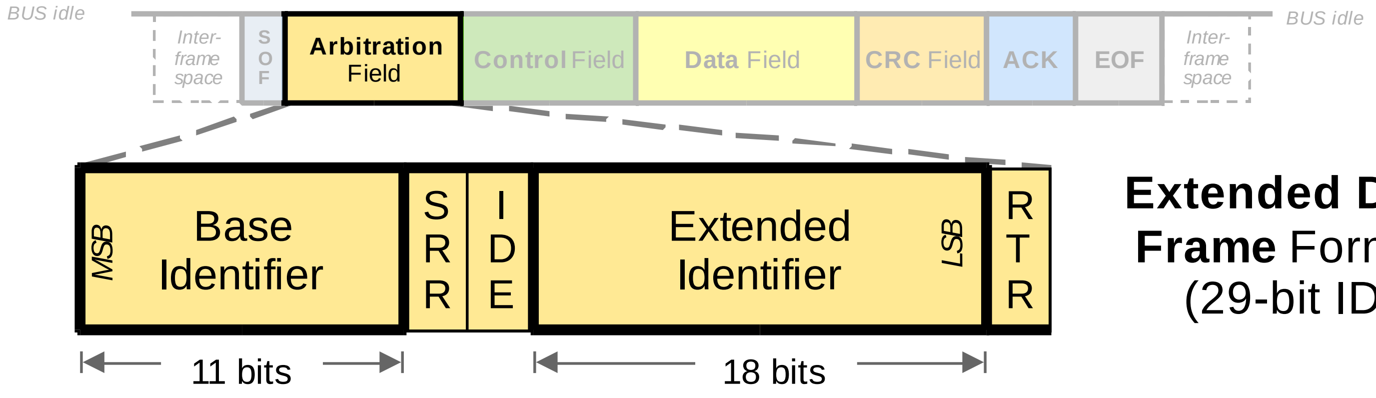
Data Frame Format



Extended Data Frame Format (29-bit ID)

- **Identifier:** 29-bits for **extended** format frames.
  - Formed by the Base Identifier of 11 bits (most-significant bits of the ID) plus the Extended identifier of 18 bits (least-significant bits of the ID).
- **Identifier Extension bit (IDE)**: 1-bit; Indicates if current frame is a **standard** frame (**dominant** value) or an **extended** frame (**recessive** value).

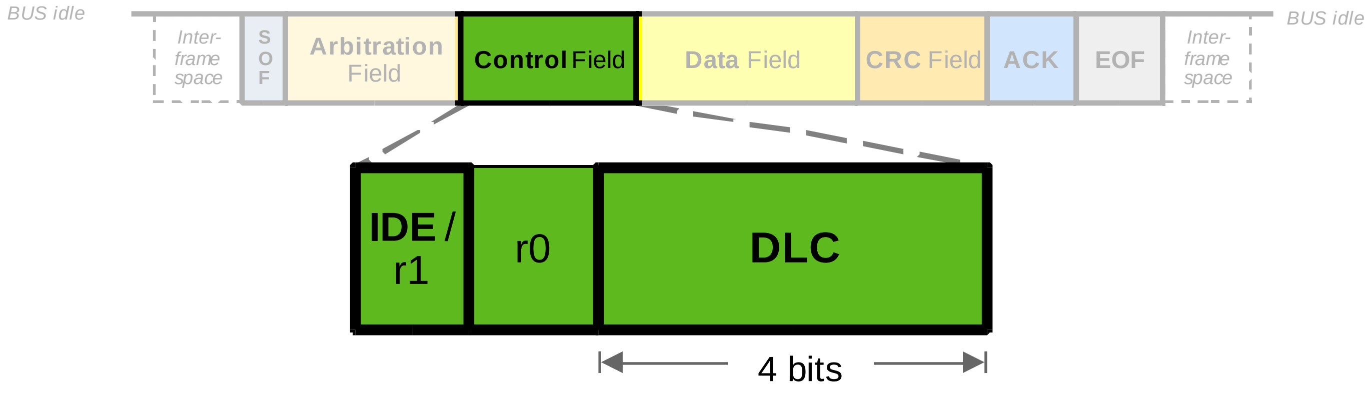
Data Frame Format



Extended Data Frame Format (29-bit ID)

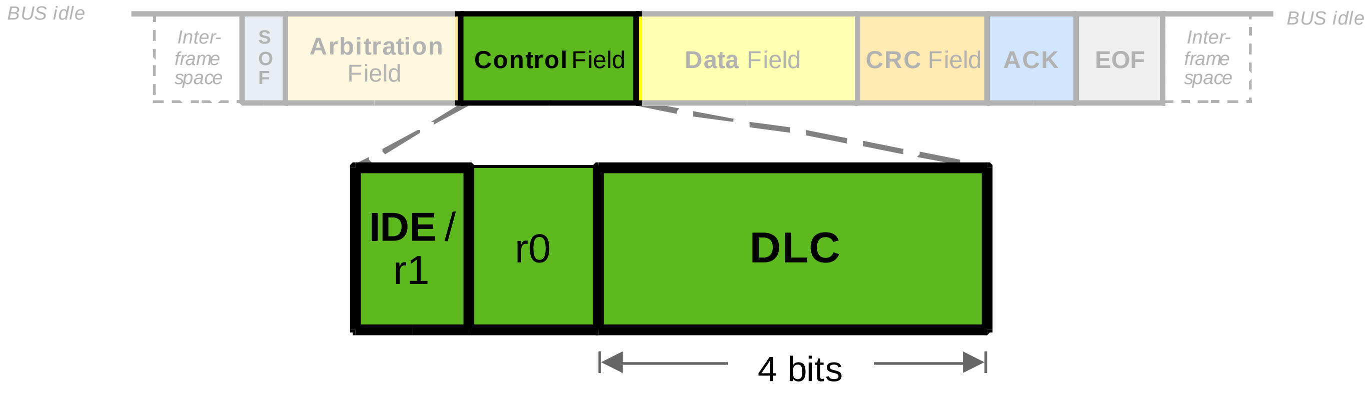
- **Substitute Remote Request (SRR)**: 1-bit. Only applicable for extended format. Substitutes the position of the RTR bit in standard format.
  - Transmitted always with **recessive** value in extended format
  - Allows Standard Frames to have higher priority than Extended Frames with same Base identifier values.
- **RTR**: Same meaning as in standard format but located after the "extended identifier" field.

Data Frame Format



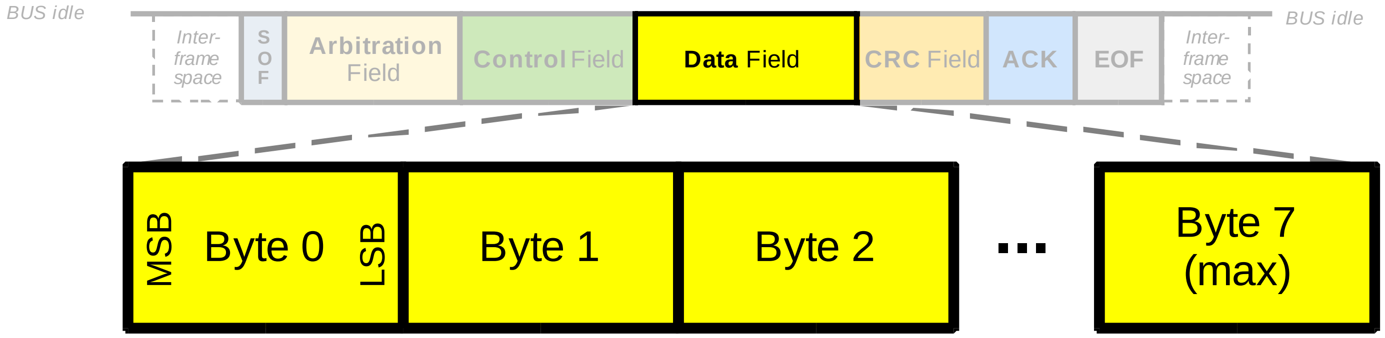
- **IDE**: In standard format, the IDE bit is part of the "Control Field". Same meaning as before → Indicates if current frame is a **standard** frame (**dominant** value) or an **extended** frame (**recessive** value).
- **r1**: In extended format, this is a reserved bit not being used. Transmitted with dominant value.
- **r0**: Reserved bit not being used. Transmitted with dominant value.

Data Frame Format



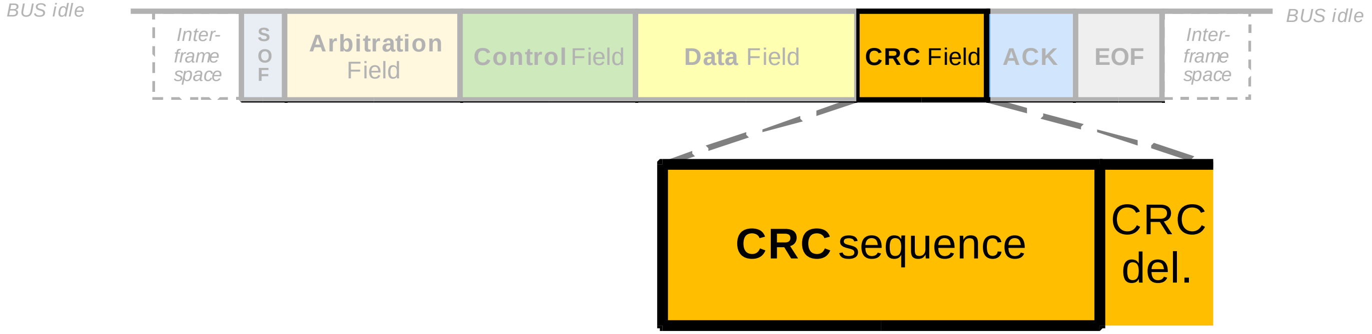
- **DLC (Data Length Code)**: 4-bits. Indicates how many **data** bytes are conveyed in this frame.
  - From 0 to 8 bytes max.
  - Assumed max of 8 data bytes even if DLC value is higher than 8.

Data Frame Format



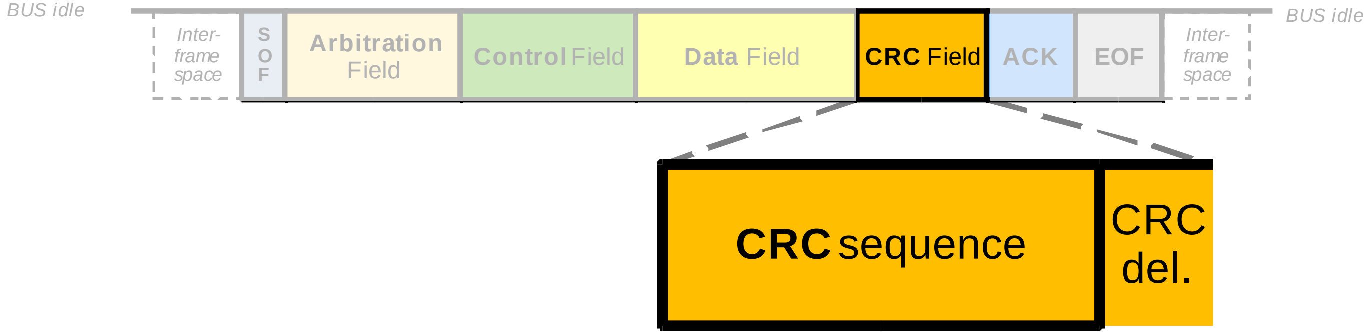
- **Data Field**
  - From 0 to max. 8 data bytes
  - Number of data bytes depend on value of DLC
  - Most-significant-bit transmitted first, Least-significant-bit last.

Data Frame Format



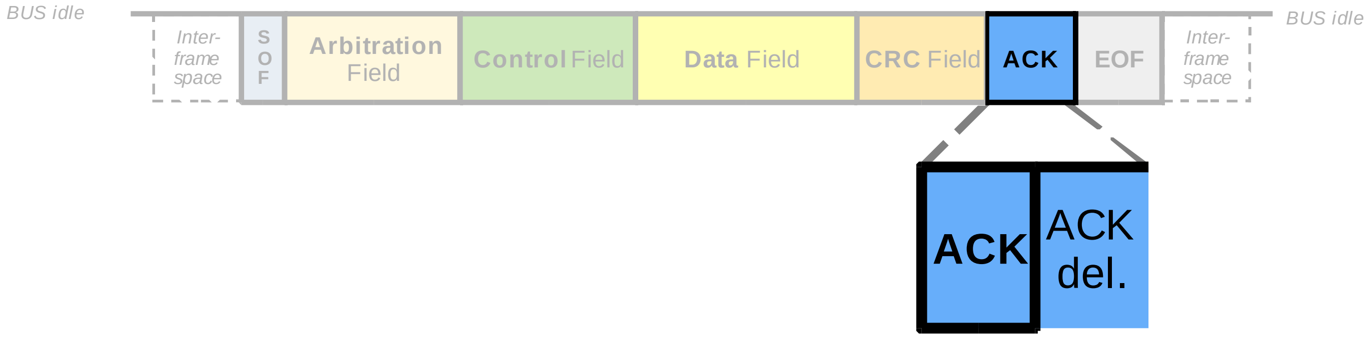
- **Cyclic Redundancy Check (CRC) sequence**: **15-bits**. Used for error detection during the transfer of a Frame.
  - CRC calculation uses fields **SOF**, **Arbitration** Field, **Control** Field and **Data** Field (if present)
  - CRC is calculated without stuffing bits
  - Receiver node validates the received bit stream using the received CRC
  - In case of CRC fail, the data in the frame is discarded

Data Frame Format



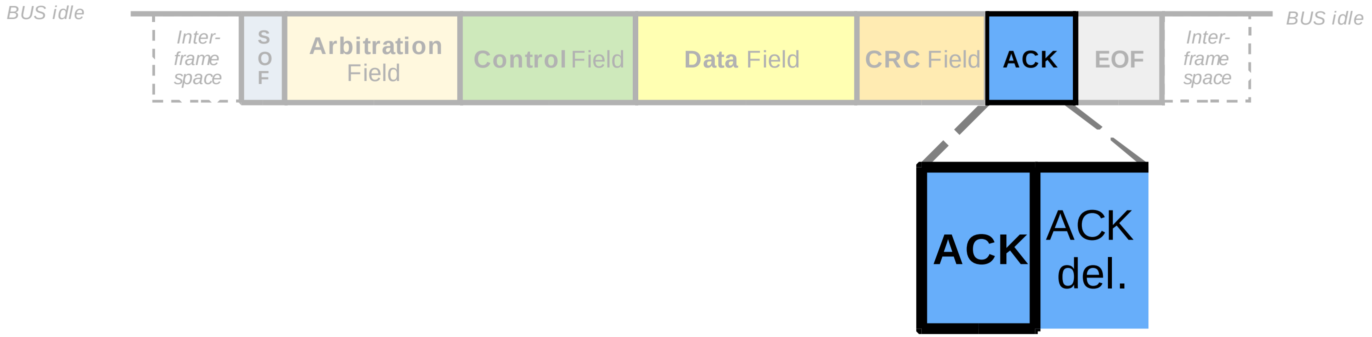
- **CRC delimiter**: 1-bit.
  - Signals the end of the CRC field
  - A single bit with recessive value

Data Frame Format



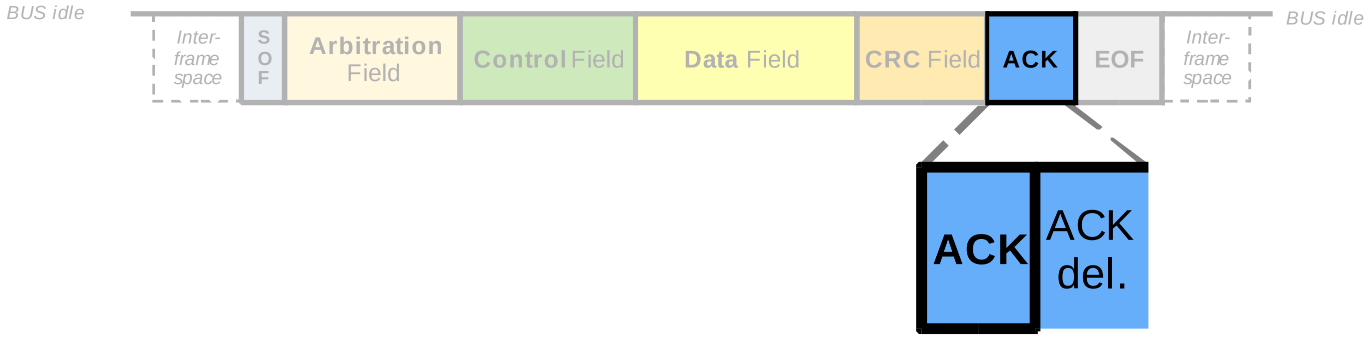
- **Acknowledge (ACK) bit**: Indicates if *at least one* node correctly received the frame
  - Transmitted with **recessive** value by the **publisher**
  - Any node shall overwrite this bit to **dominant** value if the frame was received **correctly** including CRC check (before acceptance filter of receivers)

Data Frame Format



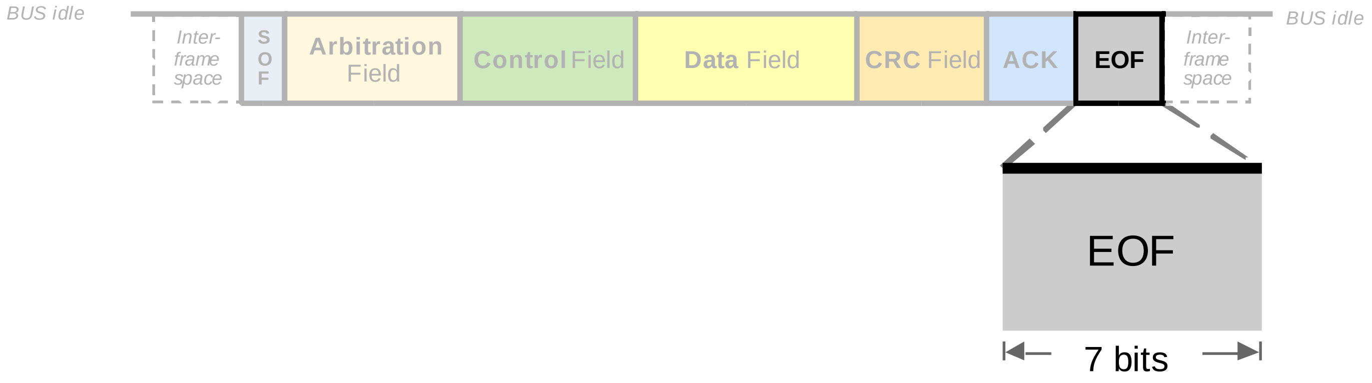
- **Acknowledge (ACK) bit**
  - If publisher sees this bit with **recessive** value, it may mean that:
    - Frame had an **error** during the transmission and, hence, no other node acknowledged it, or,
    - There is **no other node** connected to the network

Data Frame Format



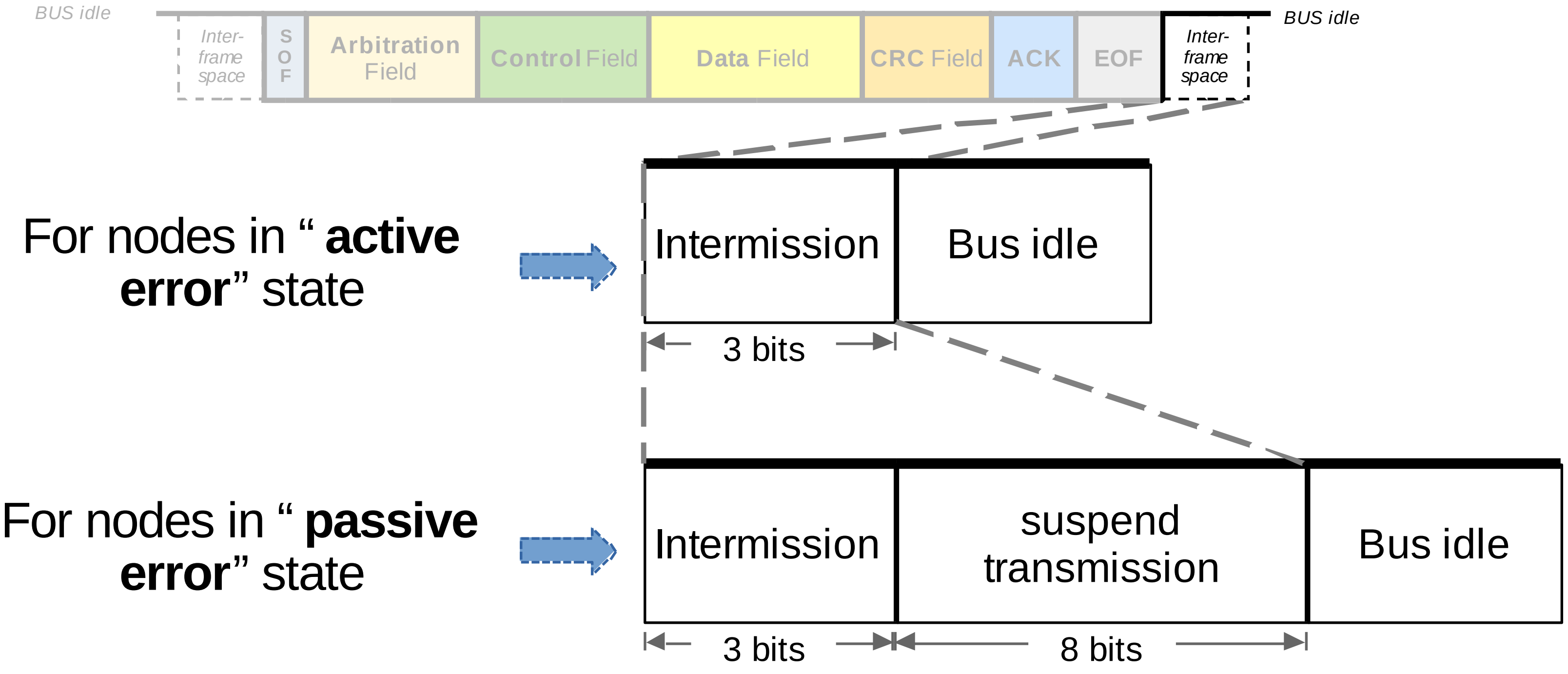
- **ACK delimiter**:
  - Signals the end of the Acknowledge field
  - A single bit with recessive value

Data Frame Format



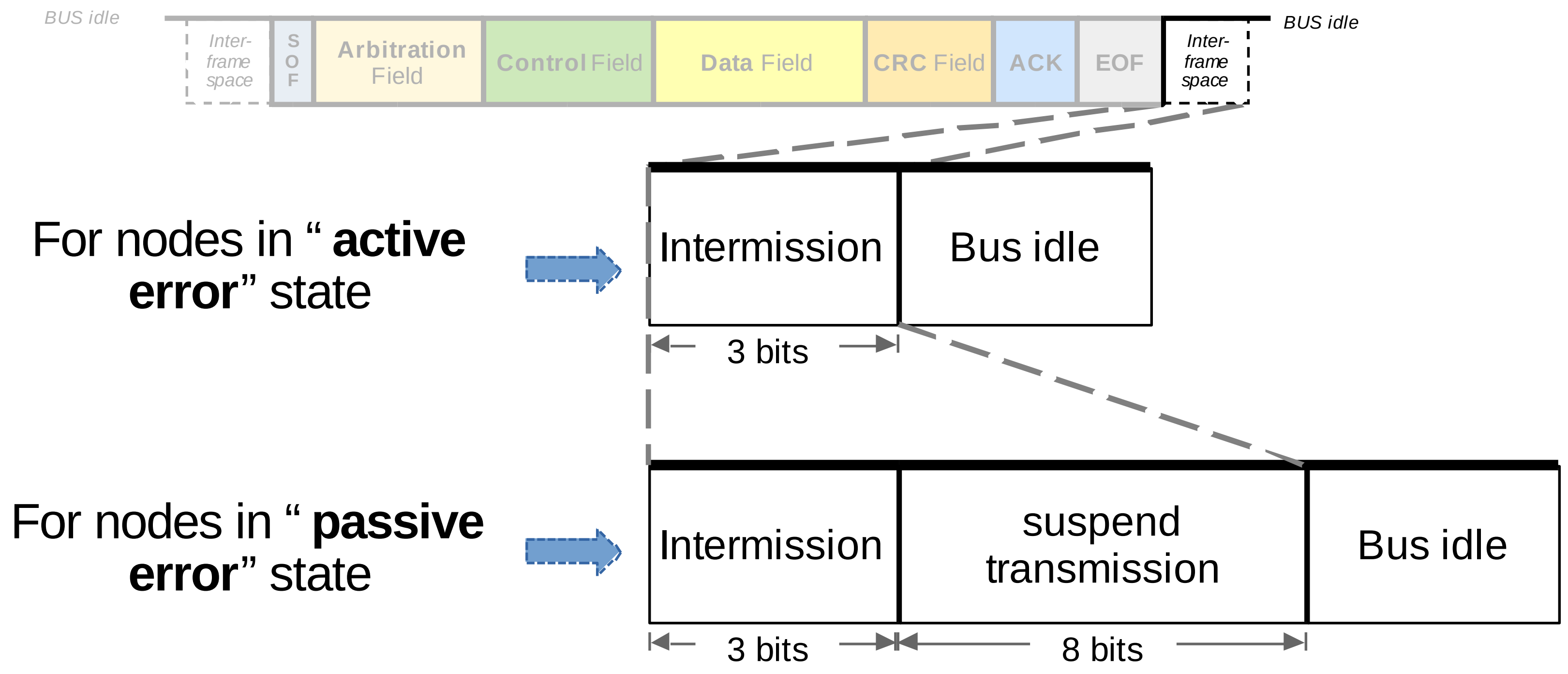
- **End-of-Frame (EOF)**: 7-bits.
  - Signals the end of the frame
  - **7 bits** with **recessive** level

Data Frame Format



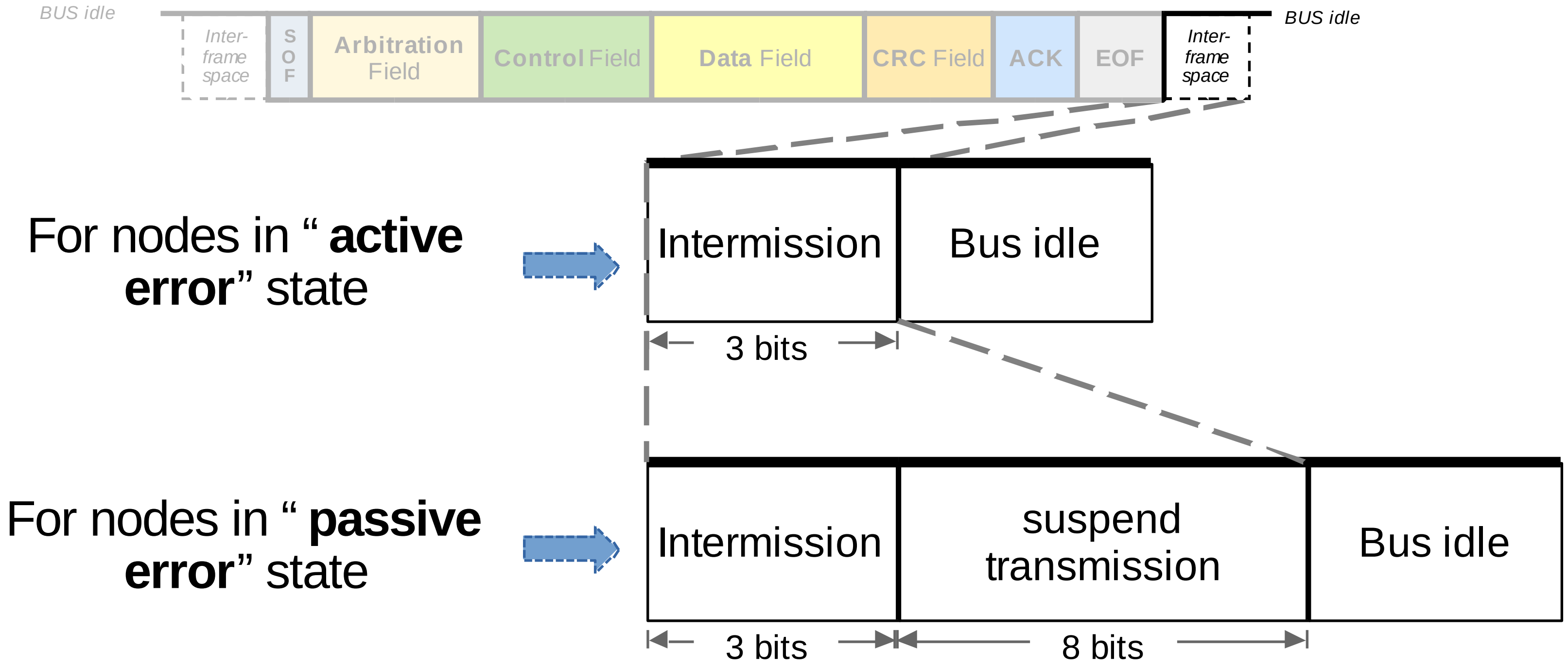
- **Intermission field**:
  - **3 recessive** bits
  - During intermission, no transmission of data or remote frames is allowed

Data Frame Format



- **Suspend transmission field**:
  - Only generated by nodes in **"passive error"** state and associated to their own frame transmissions
  - **8 recessive** bits

Data Frame Format



- **Bus idle**
  - Bus is available for nodes to begin transmission of frames
  - Arbitrary length

Protocol Specification

- Classic CAN Frame **Types**
  - **Data** Frame
    - *Standard* **Data** Frame
    - *Extended* **Data** Frame
  - **Remote** Frame
    - *Standard* **Remote** Frame
    - *Extended* **Remote** Frame
  - **Error** Frame
    - **Active Error** Frame
    - **Passive Error** Frame
  - **Overload** Frame

Remote Frame Format

- Purpose of a **Remote Frame** is to **ask** for data rather than to publish it as in the case of a Data Frame.

- A node can ask for a given message to be transmitted **by its publisher** by generating a Remote Frame with ID equal to the Data Frame ID being requested

- When a node sees a Remote Frame asking for an ID which it owns, the node shall start transmission of the corresponding Data Frame as soon as possible.

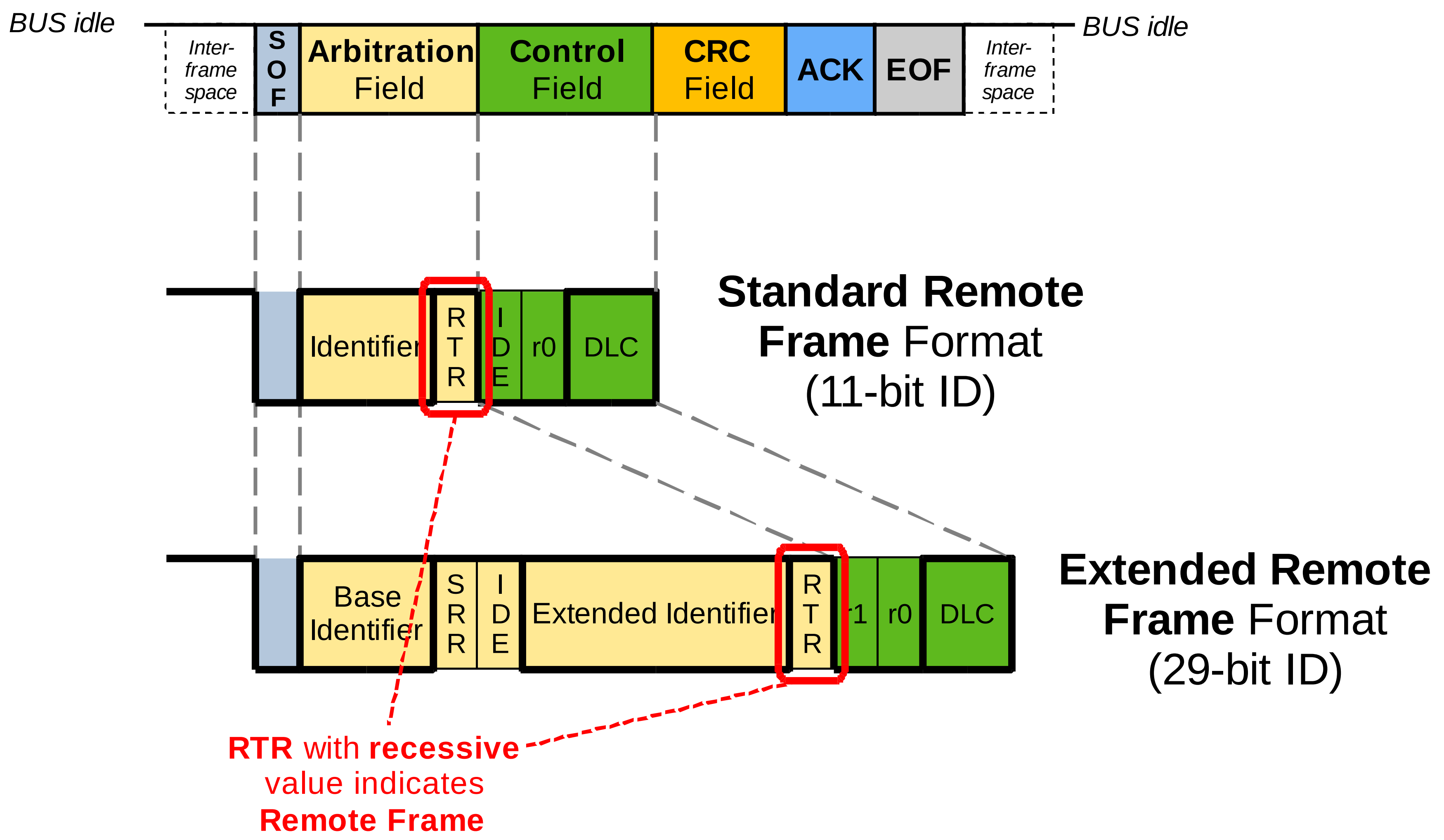
Remote Frame Format

- **Remote Frame format** is very similar to the format of a Data Frame → It contains all fields of a Data frame **except** for the "**Data Field**"

- Remote Frame can also have **standard** or **extended** formats analogue to those of Data Frames

- A Remote Frame is distinguished from a Data Frame by the **RTR** bit being **recessive**.

Remote Frame Format



Frame arbitration

- The last part of a Data or Remote frame consists of at least **11 recessive** bits

  →ACK delimiter (1-bit) + EOF (7-bits) + Intermission field (3-bits) = 11 bits

- During normal transmission of a Data or Remote frame, the bus shall never have 11 consecutive recessive bits (due to bit stuffing)

- Therefore, nodes can consider **bus** as being **idle** after seeing **11 consecutive recessive bits**

Frame arbitration

- **Scenario**
  - A Data Frame is currently being transmitted by Node A
  - 3 nodes (Node B, Node C and Node D) are waiting for the bus to become idle in order to begin Data Frame transmission

    →Waiting to see 11 consecutive recessive bits
  - The 3 nodes see the bus in idle at the same time and they start publishing their Data Frames
  - A **collision occurs** since multiple nodes are trying to publish different messages in the bus **at the same time**!

Frame arbitration

- **CAN Message Arbitration**
  - CAN provides a non-destructive message arbitration mechanism
  - **At least one node** competing from the bus will **win** the bus arbitration and complete its frame transmission
  - Loser nodes will **wait** for the next time the bus is idle in order to attempt their Frame transmissions

    → Typically, CAN controllers manage automatic re-transmission attempts

Frame arbitration

- **CAN Message Arbitration**
  - CAN message arbitration occurs during the publication of the "**Arbitration** Field", in particular during the publication of the "Frame Identifier".

    → The message **Identifier** is at the center of the arbitration criteria

  - Frame Identifier is transmitted bit by bit starting with the Most-Significant one.

  - During transmission of the "Identifier Field", CAN nodes transmit each bit and **read it back** to verify the actual bus level for that bit

Frame arbitration

- **CAN Message Arbitration**
  - If the read back bit level from the bus **corresponds** to the transmitted one, then arbitration **has not been lost** and the Node **proceeds** with the transmission of the next identifier bit

    → If a Node can publish all the Frame identifier bits, then it has **won** the bus arbitration and can proceed with the rest of the Frame transmission!

  - If the read back bit level from the bus is **opposite** to the transmitted one, then arbitration **has been lost**.

    →The node shall **stop** transmission of further bits of the Frame

Frame arbitration

- **CAN Message Arbitration**
  - Bit differences between transmitted bit levels and actual bus levels can be due to a node publishing a “dominant” level against “recessive” levels from other nodes

- **Arbitration Example**
  - **Node B** wants to transmit message **0x35**
  - **Node C** wants to transmit message **0x26**
  - **Node D** wants to transmit message **0x343**

At the same time!

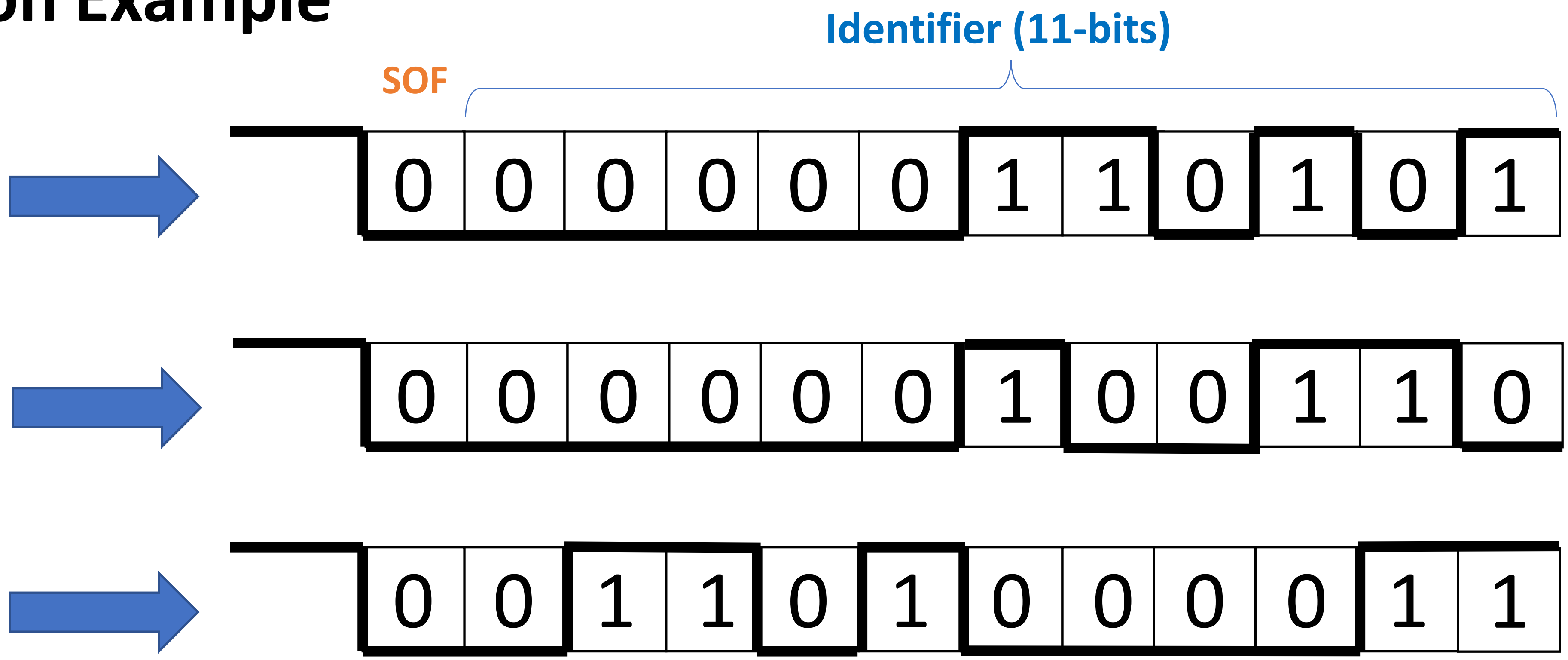
Frame arbitration

- **Arbitration Example**

Node B
Msg **0x35**

Node C
Msg **0x26**

Node D
Msg **0x343**



Frame arbitration

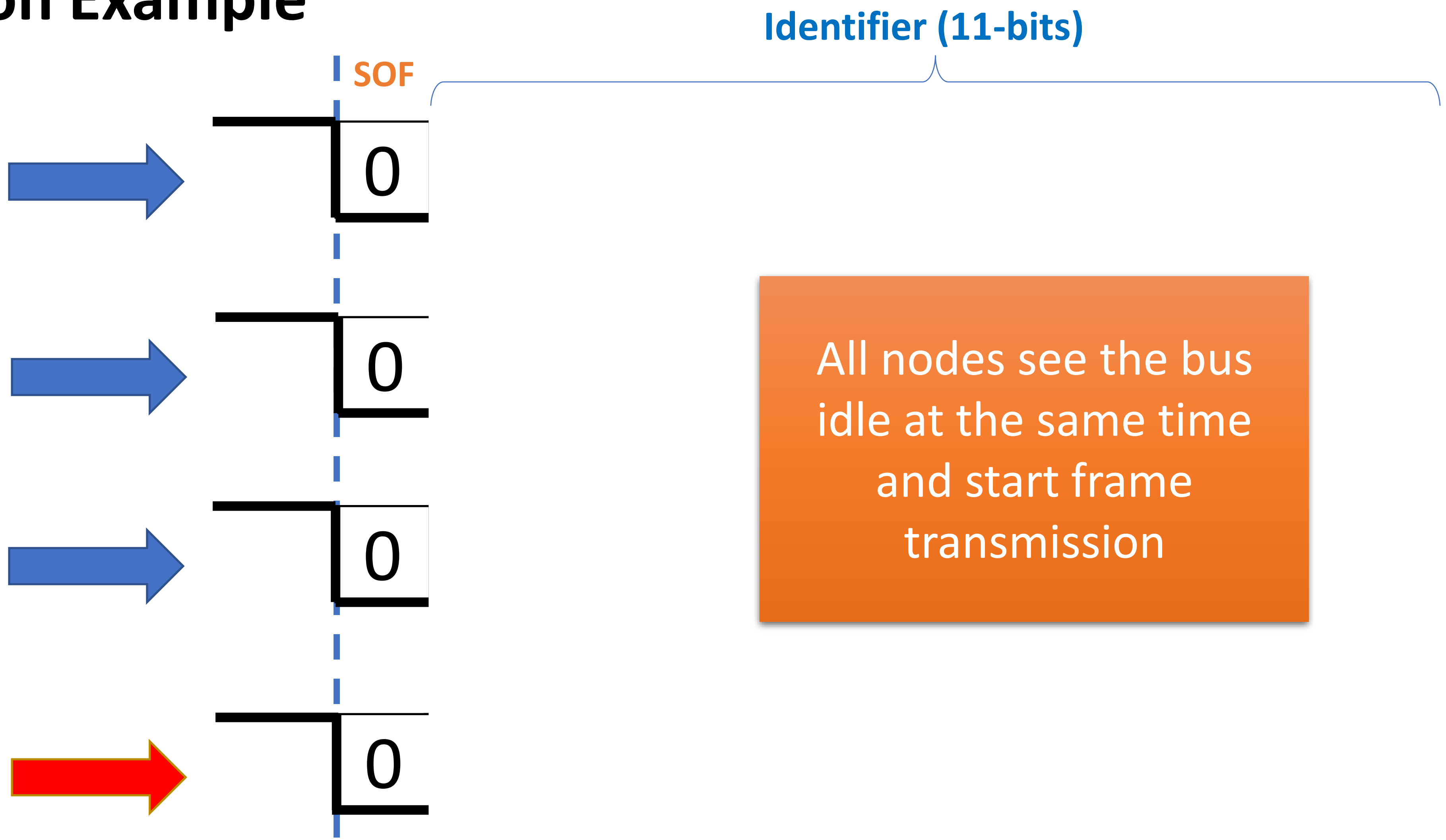
- **Arbitration Example**

Node B
Msg **0x35**

Node C
Msg **0x26**

Node D
Msg **0x343**

Actual Bus Level



All nodes see the bus idle at the same time and start frame transmission

Frame arbitration

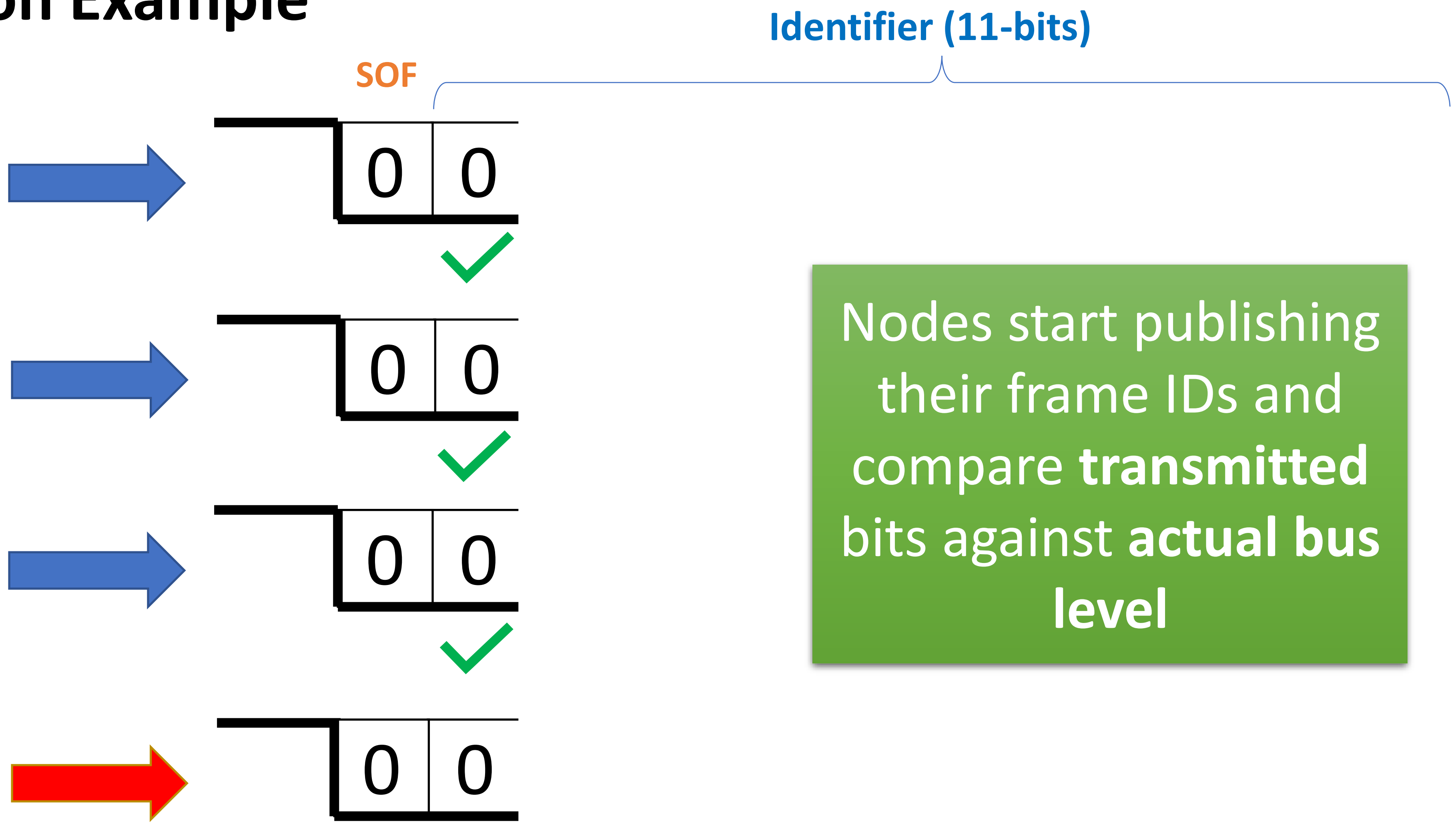
- **Arbitration Example**

Node B
Msg **0x35**

Node C
Msg **0x26**

Node D
Msg **0x343**

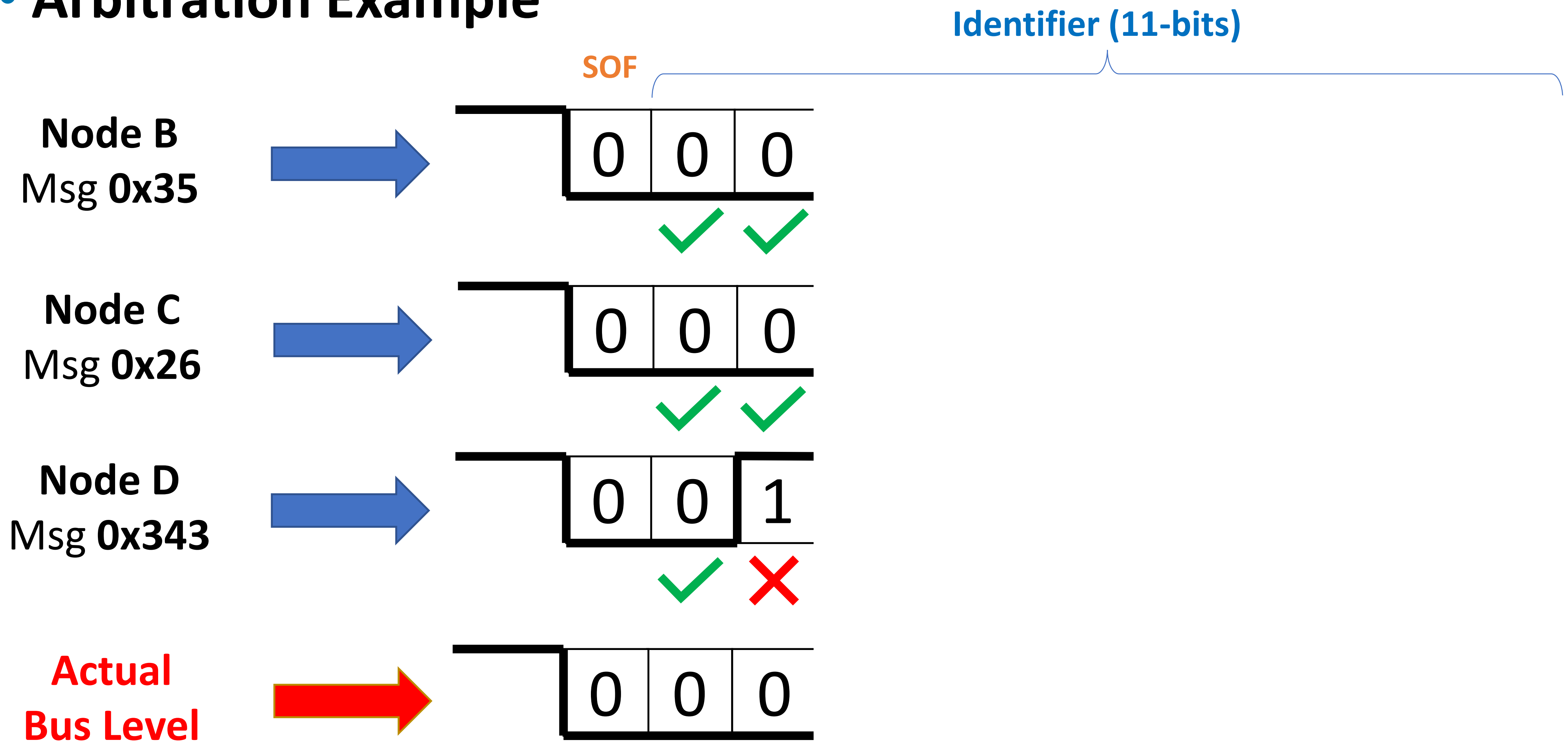
Actual Bus Level



Nodes start publishing their frame IDs and compare **transmitted** bits against **actual bus level**

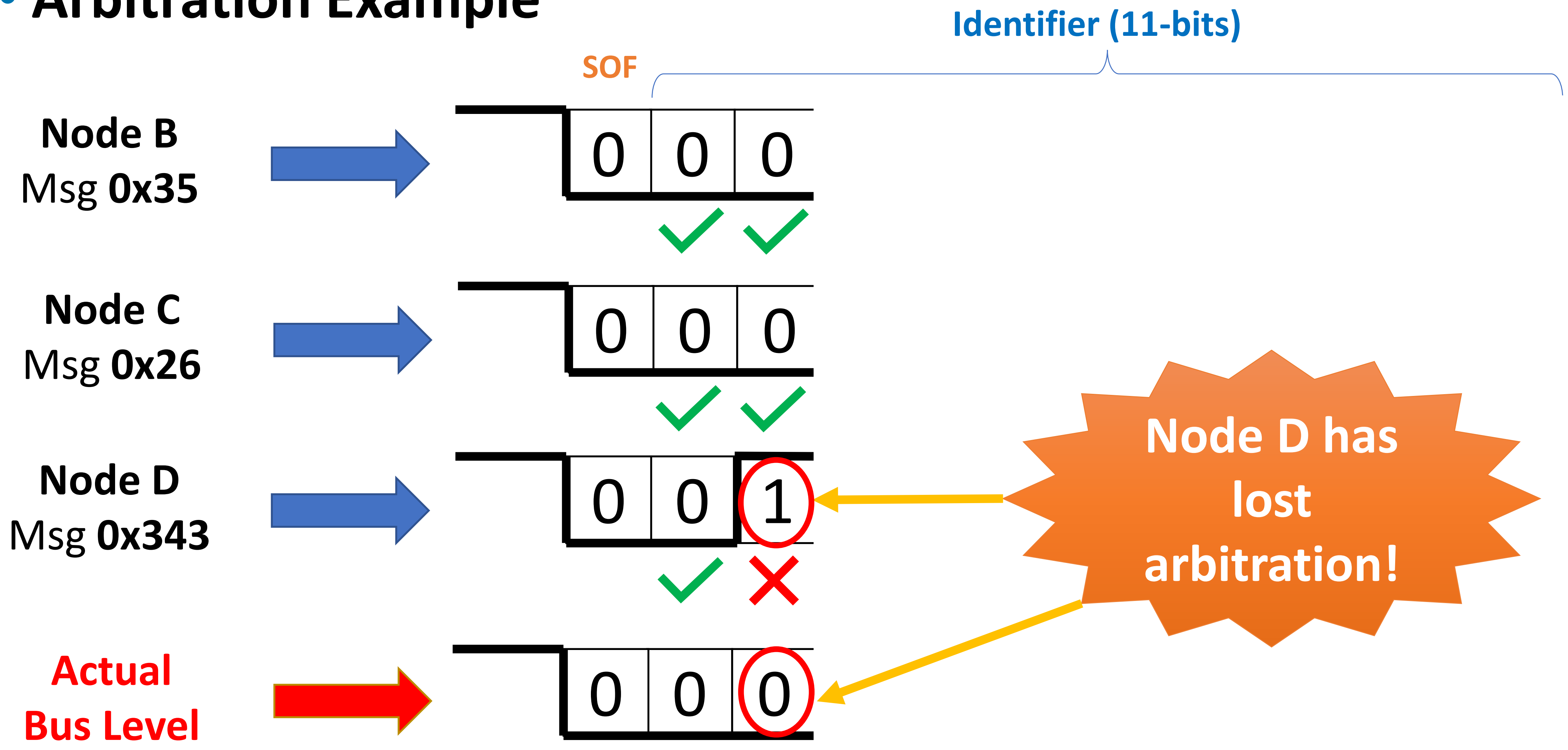
Frame arbitration

- **Arbitration Example**



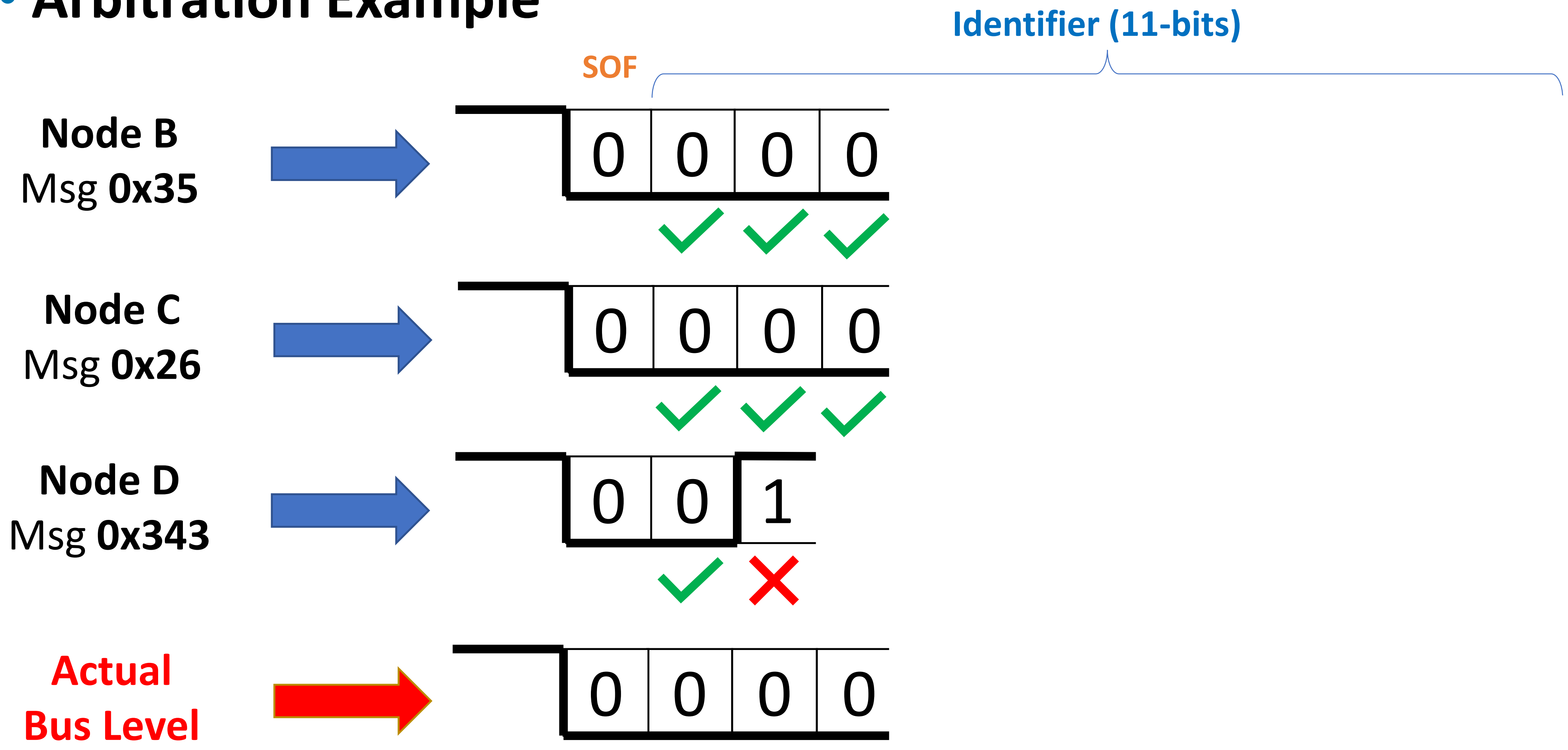
Frame arbitration

- **Arbitration Example**



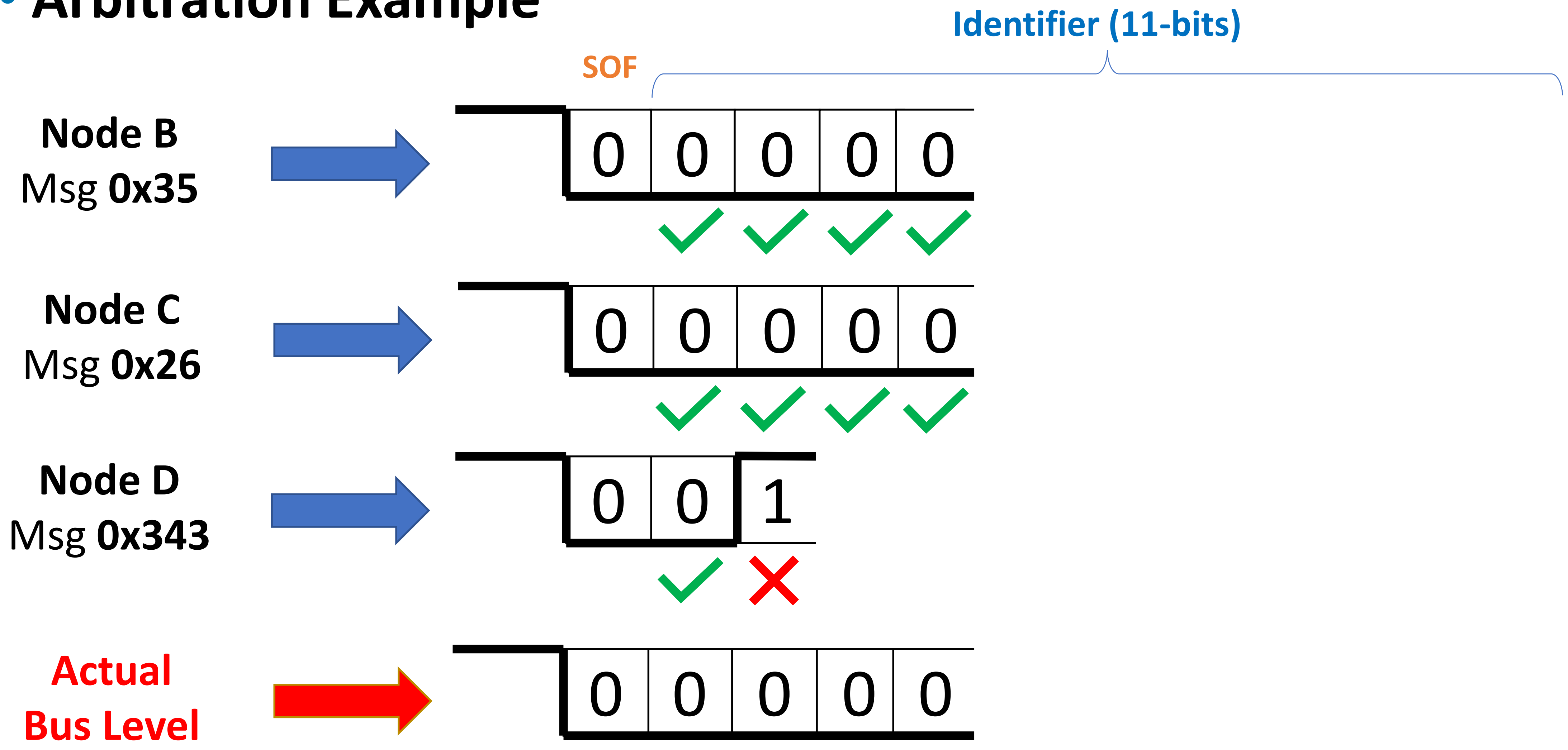
Frame arbitration

- **Arbitration Example**



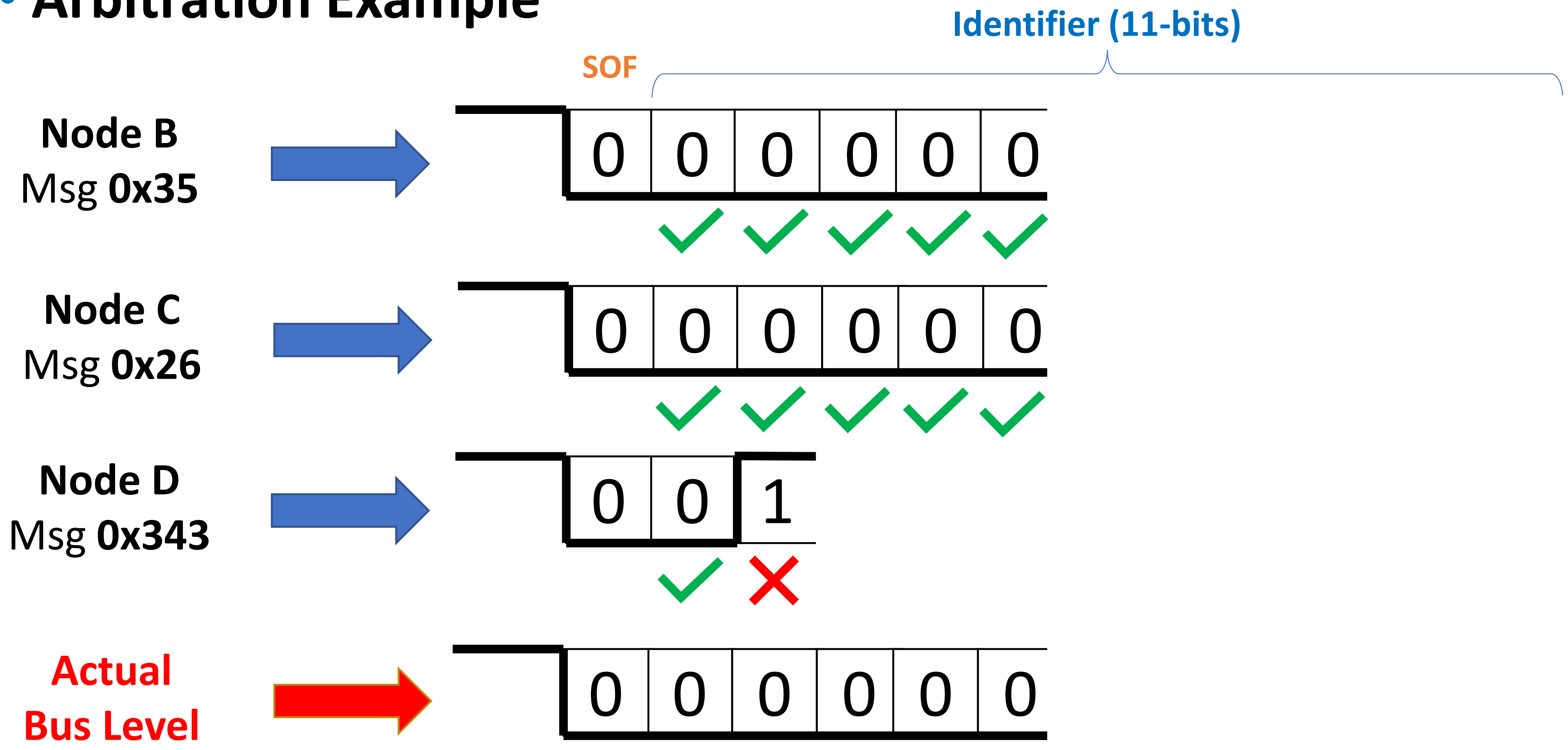
Frame arbitration

- **Arbitration Example**



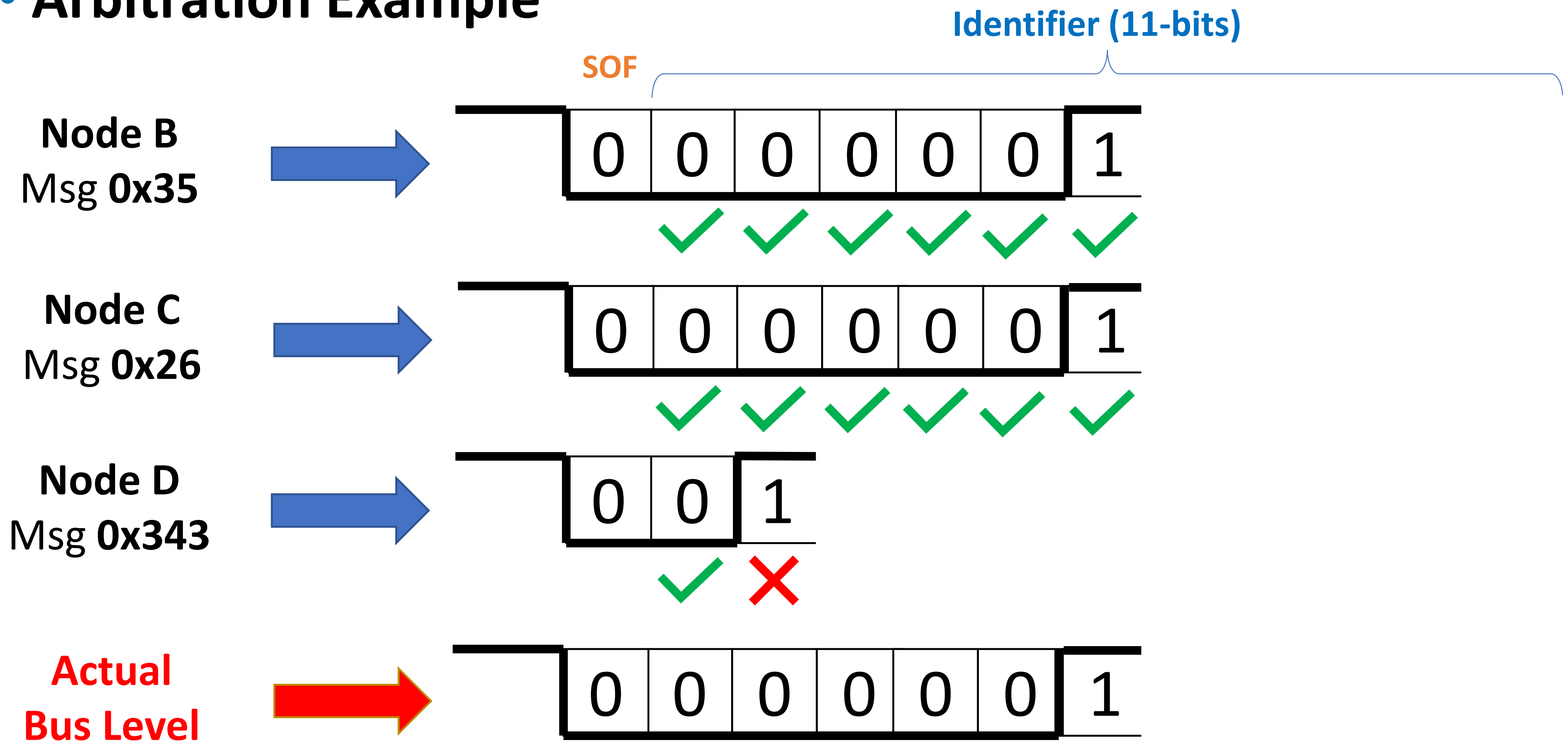
Frame arbitration

- **Arbitration Example**



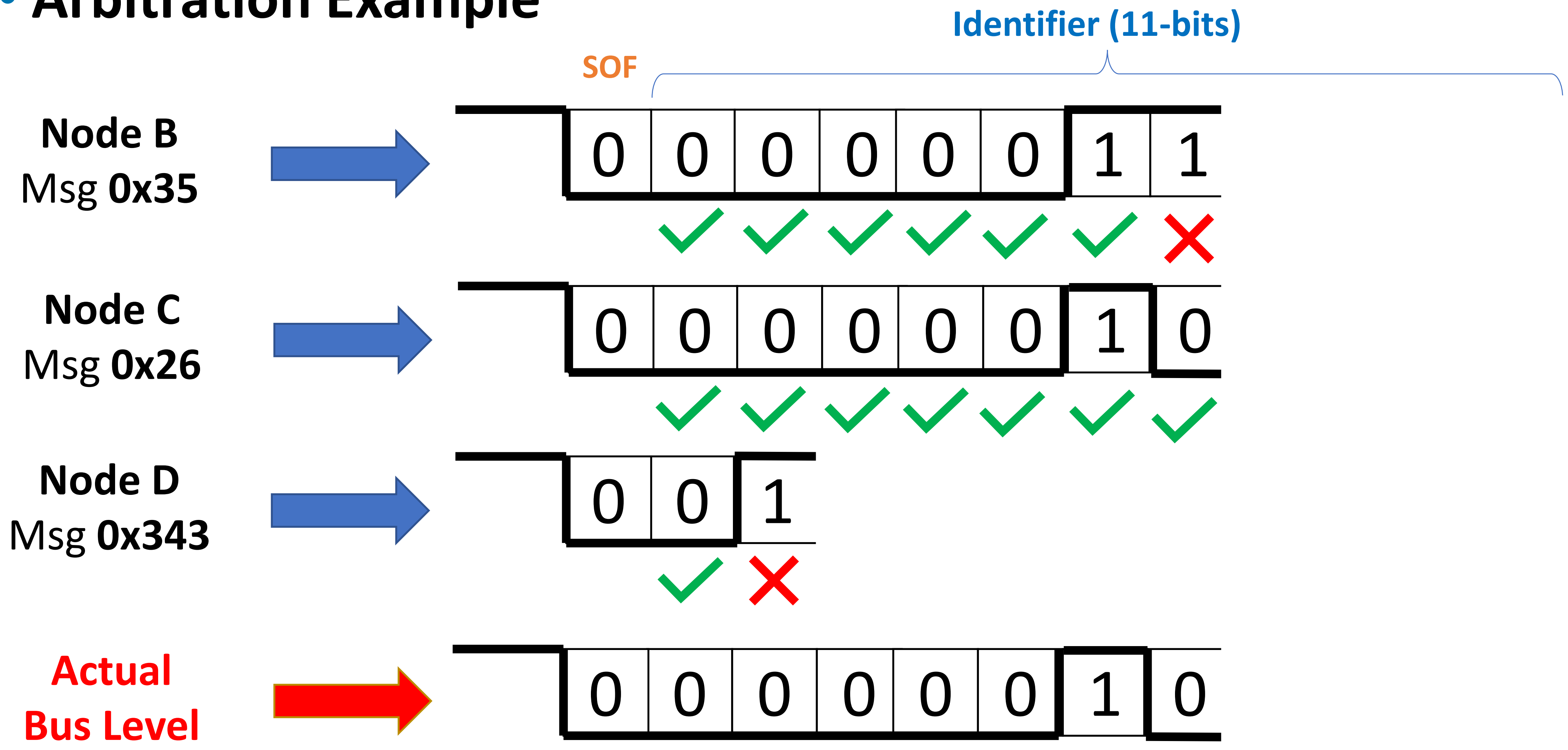
Frame arbitration

- **Arbitration Example**



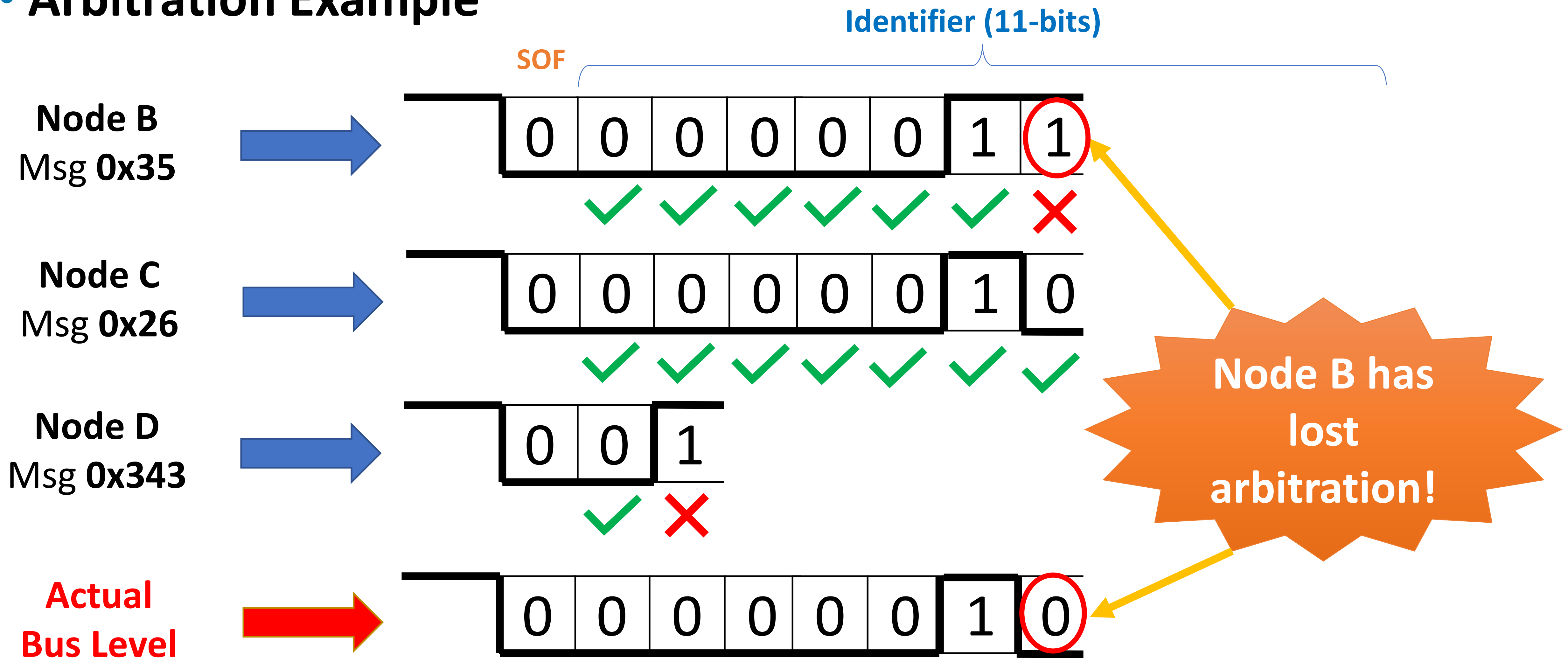
Frame arbitration

- **Arbitration Example**



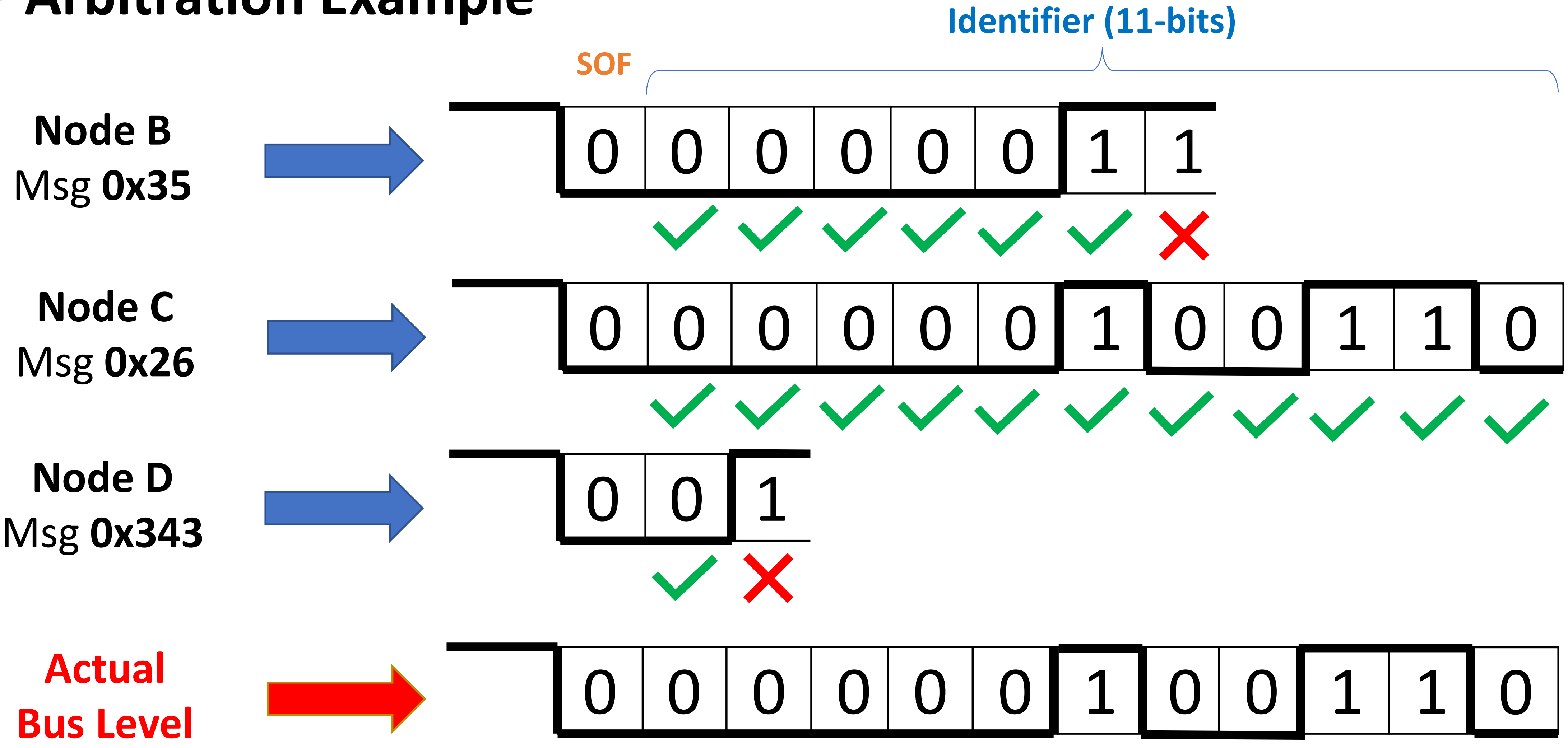
Frame arbitration

- **Arbitration Example**



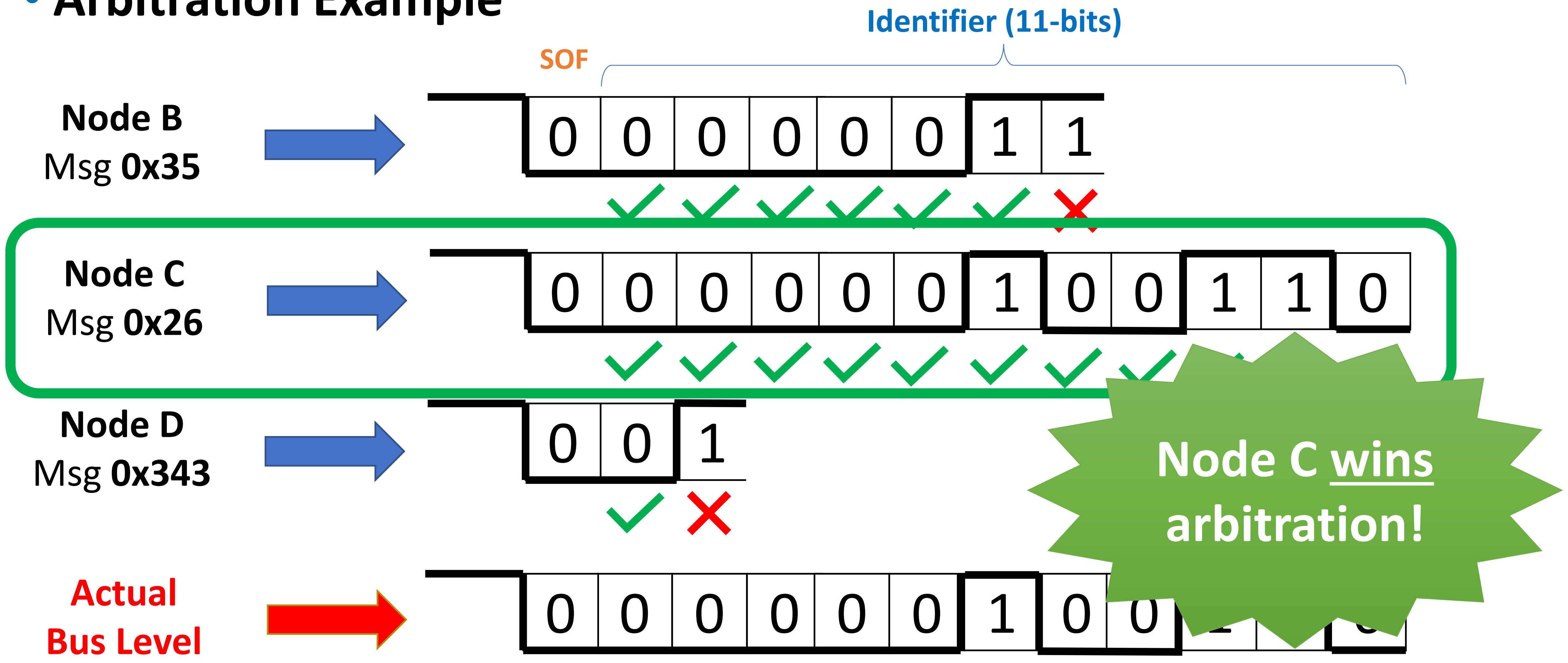
Frame arbitration

- **Arbitration Example**



Frame arbitration

- **Arbitration Example**



Frame arbitration

- **CAN Message Arbitration**

The **lower** the ID value the **higher** the priority!

- Critical data shall be transmitted in messages with low ID value

Bit Stuffing

- CAN provides a technique to overcome the problem of lost synchronization due to its **asynchronous** nature and its **NRZ** bit codification

  → **Bit Stuffing technique**

Bit Stuffing

- **Bit Stuffing** technique:
  - Never allow **more than 5 bits with the same level** during Data or Remote Frame transmission
  - Each time 5 bits with same level are observed in the bus, a bit (known as stuffing bit) shall be **inserted with opposite level**.
    - This ensure edges at least every 5 bits
    - Edges are opportunities for re-synchronization of nodes
  - **The 5 bits count shall also include the inserted stuffing bits**

Bit Stuffing

- **Bit Stuffing** technique:
    - Bit stuffing applies from **Start-of-Frame** up to and including the **CRC sequence** field
    - CRC calculations are done without stuffing bits (after de-stuffing from receiver side)

Bit Stuffing

- **Bit Stuffing** example **1**:

Original Bit Stream Sequence

0	1	1	0	0	0	0	0	1	0

Bit Stream Sequence **with stuffing bits**

0	1	1	0	0	0	0	0	**1**	1	0
								Stuff bit		

Bit Stuffing

- **Bit Stuffing** example **2**:

Original Bit Stream Sequence

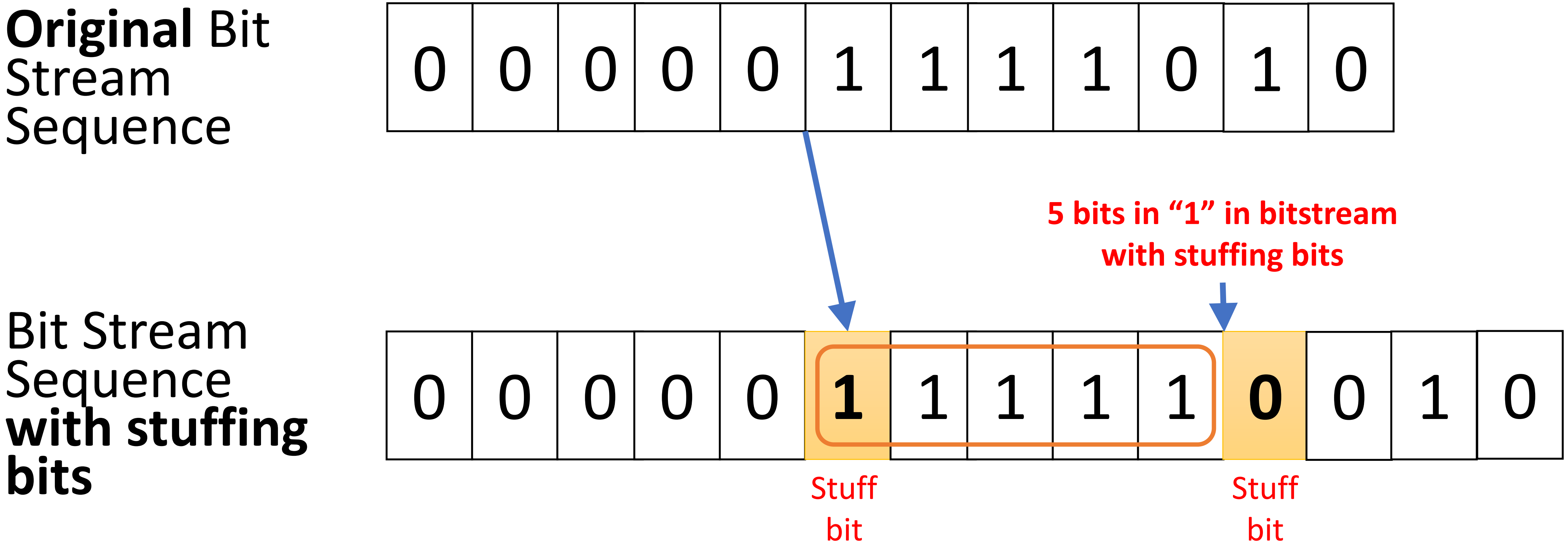
0	0	0	0	0	0	1	1	1	1	1	1

Bit Stream Sequence **with stuffing bits**

0	0	0	0	0	**1**	0	1	1	1	1	1	**0**	1
					Stuff bit							Stuff bit	

Bit Stuffing

- **Bit Stuffing** example **3**:



Error Handling

Error Sources

- **Format errors**
  - Invalid format, early end of frame, etc…
- **Bit errors**
  - Actual bus level don’t correspond with the transmitted level
  - Exceptions: during arbitration and in ACK field
- **Bit-stuffing errors**
  - More than 5 bits with same level observed in the bus during Data or Remote Frame transmission

Error Handling

Error Sources

- **Acknowledge errors**
  - ACK bit is not asserted (dominant level) by any node subscribing node
  - May indicate an actual error or that the transmitting node is “alone” in the network
- **CRC errors**
  - CRC check fails (data corruption due to EMI, physical bus failures, etc.)

Error Handling

- CAN nodes have **error counters**
  - **<u>Transmit</u> error counter (TEC)**
  - **<u>Receive</u> error counter (REC)**

Error Handling

- **Transmit error counter (TEC)**
  - Starts with value of **zero** upon **initialization**
  - Min value is zero, max value is 255
  - Increased when the node detects an error on the frame it is **transmitting**
    - Typically, **increased by 8** when detecting a TX error
  - Decreased when the node **successfully transmits** a frame
    - **Decreased by 1**

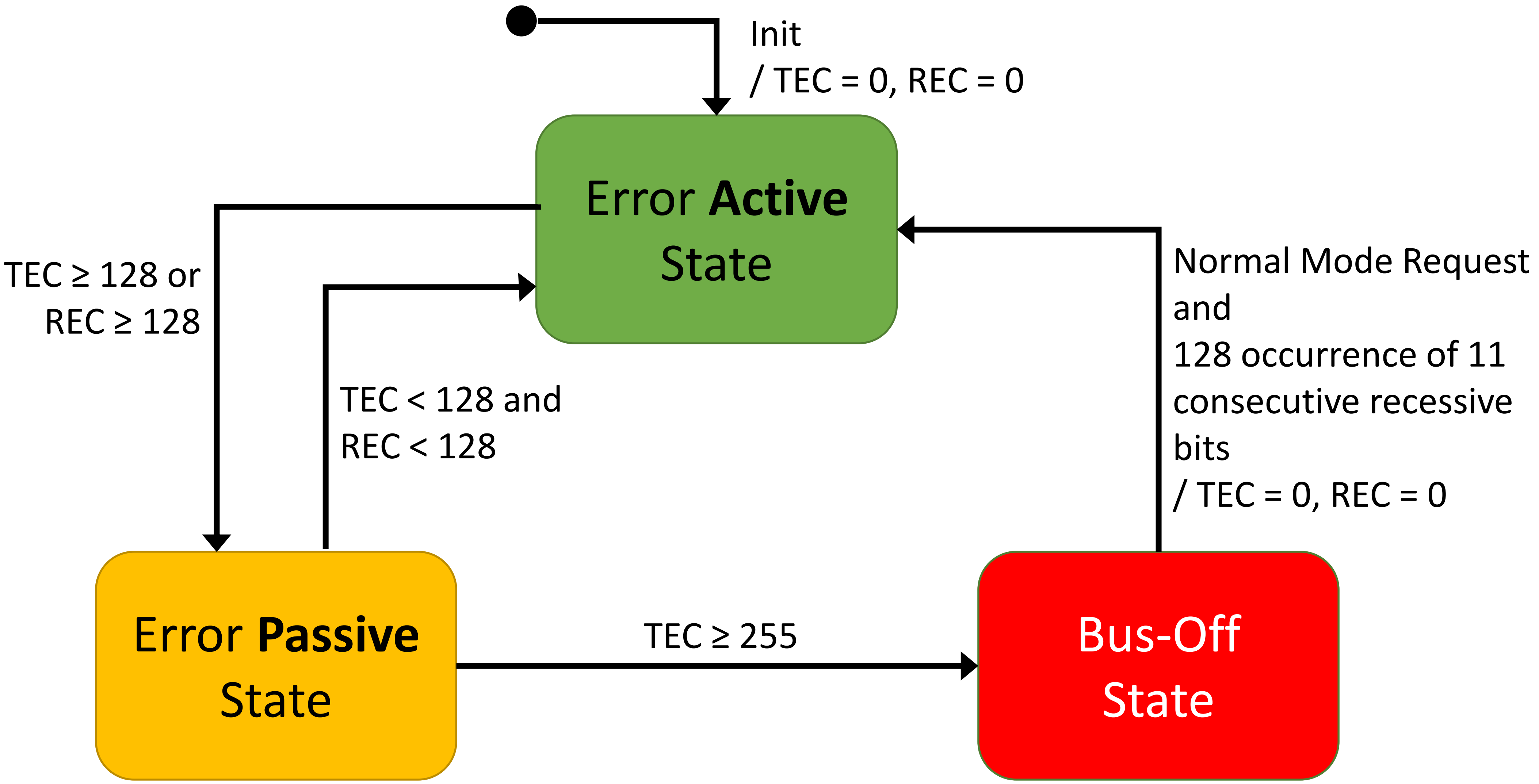
Error Handling

- **<u>Receive</u> error counter (REC)**
  - Starts with value of **zero** upon **initialization**
  - Min value is zero, max value is 255 (this value typically doesn't exceed 135)
  - Increased when the node detects an error on the frame it is **receiving**
    - **Increased by 8** when the node identifies the error by itself, increased by 1 otherwise
  - Decreased when the node **successfully receives** a frame
    - **Decreased by 1**

Error Handling

- Depending on the values of the Transmit and Receive error counters, the node can be in any of the following **states**:
  - **Error active** state
    - TEC < 128 **and** REC < 128
  - **Error passive** state
    - TEC ≥ 128 **or** REC ≥ 128
  - **Bus-off** state
    - TEC ≥ 255

Error Handling

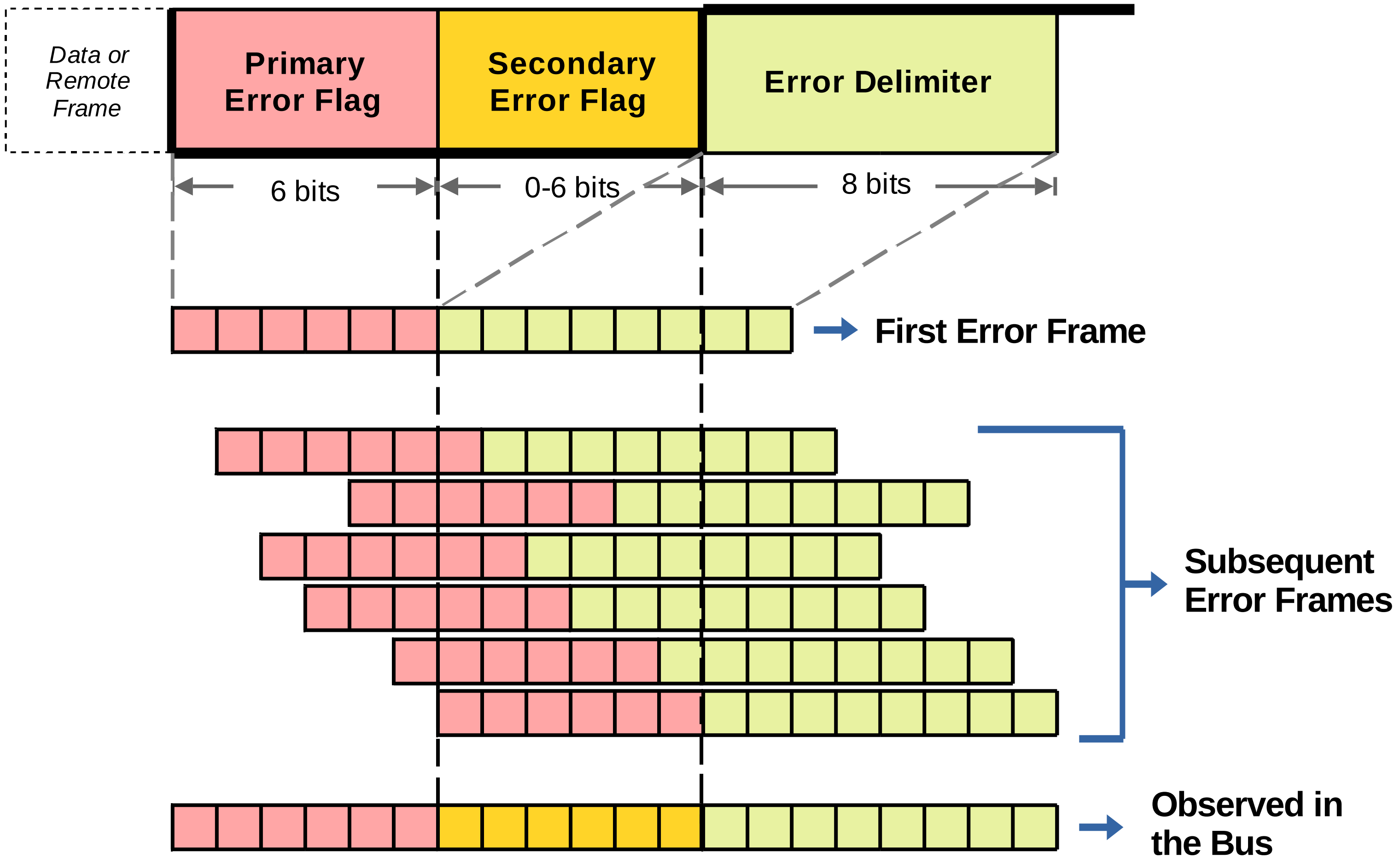


Error Handling

- **Error Active** State
  - Is the “normal” state to be if there are no abnormal errors in the network (TEC and REC values are “low”)
  - In this state the node **transmits and receives** frames **normally**
  - In this state, if a node detects an error, it informs the rest of the nodes in the bus by transmitting an **<u>active error frame</u>**

Error Handling

- **Active Error** Frame Format



Error Handling

- **Active Error** Frame Format
  - **Primary Error Flag:** 6 dominant bits
    - Explicitly violates bit-stuffing rule
    - Other nodes in the network may generate further error frames due to the bit-stuffing violation
  - **Secondary Error Flag**: From 0 to 6 bits (depending on subsequently generated error frames)

Error Handling

- **Active Error** Frame Format
  - Publication of this frame **abruptly** interrupts current frame being transmitted
  - Publishing node can retry transmission later

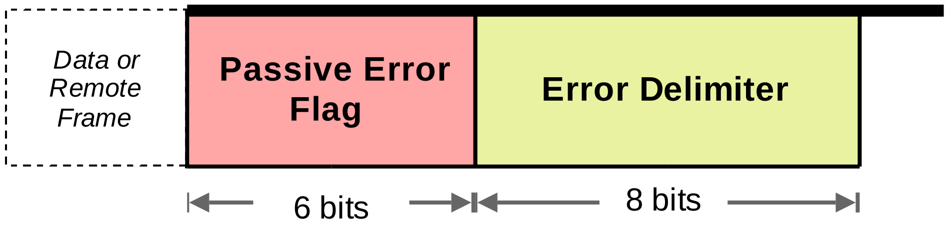
Error Handling

- **Error Passive** State
  - This state indicates a problematic situation in the bus since the REC or TEC counters are relatively high
  - Node **stills transmits** and **receives** frames, but, in the case of transmission, it "**sabotages**" itself by including additional 8 bits in the interframe space via the "suspend transmission" field.

    → Other competing nodes will likely take the bus before since they don't have those extra 8 bits in the interframe space.
  - In this state, if a node detects an error, it transmits a **passive error frame**

Error Handling

- **Passive Error** Frame Format



  - **Passive Error** Flag: **6 recessive bits**
  - **Error delimiter**: **8 recessive bits**
  - The passive error frame is “**invisible**” to the CAN bus
    - → Consists of only recessive bits
    - → Doesn’t disrupt any on-going communication

Error Handling

- **Bus-Off** State
  - This state indicates a **critical** situation in the bus most likely caused by the node itself
  - Node "**disconnects**" itself from the network
    - **No reception** of frames
    - **No transmission** of frames

Error Handling

- **Bus-Off** State
  - **Exit criteria** from this state may imply (application specific):
    - Waiting time
    - Re-initialization of at least the CAN controller
    - Wait for bus in idle for a given time (128 occurrences of 11 consecutive recessive bits)

Contents

Transmission Types

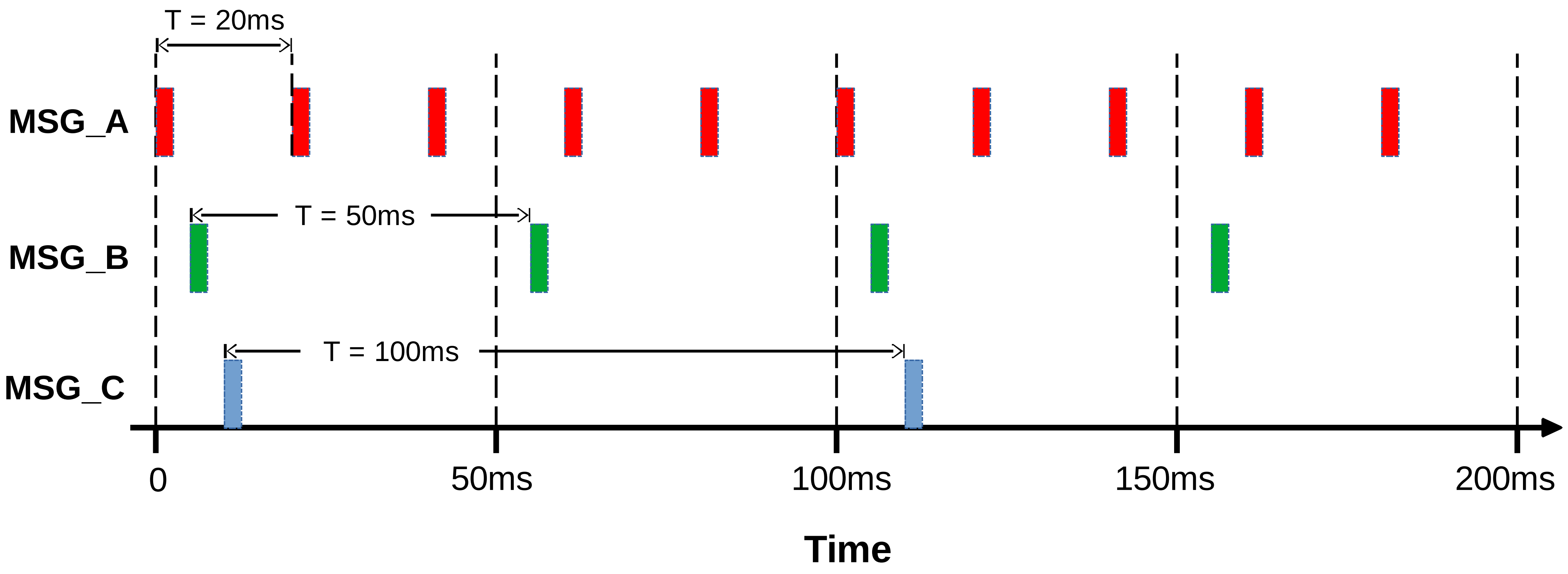
- **Message Transmission Types**
  - **Cyclic** *(most used)*
  - **Spontaneous**
  - **Cyclic and Spontaneous**
  - **By-Active-Function (BAF)**
- *Note: These transmission types are not part of the CAN specification but are defined at "application" level*

Transmission Types

- **Cyclic messages**
    - Are transmitted cyclically at a defined transmission period
    - Examples of period values
        - 10ms to 50ms for critical highly-changing data messages
        - 100ms to 500ms for non-critical messages
        - ≥1000ms for slowly-changing data messages

Transmission Types

- **Cyclic messages**



Transmission Types

- **Cyclic messages**
  - For cyclic messages is common to define a **timeout** time to check for data availability
  - This timeout time is often defined as a multiple of the message's period time, examples:
    - 10 times its period

      →A message with T=20ms will have timeout time of 200ms

      →A message with T=100ms will have timeout time of 1000ms

Transmission Types

- **Cyclic messages**
  - If a message is **not received** within its defined **timeout** time, the data is considered as **not available**.
    - Application of receiver node needs to assume data values (for each signal) to use for its functional purposes.
    - E.g., Use “last-known-value”, assume a fail-safe value by default, etc.

Transmission Types

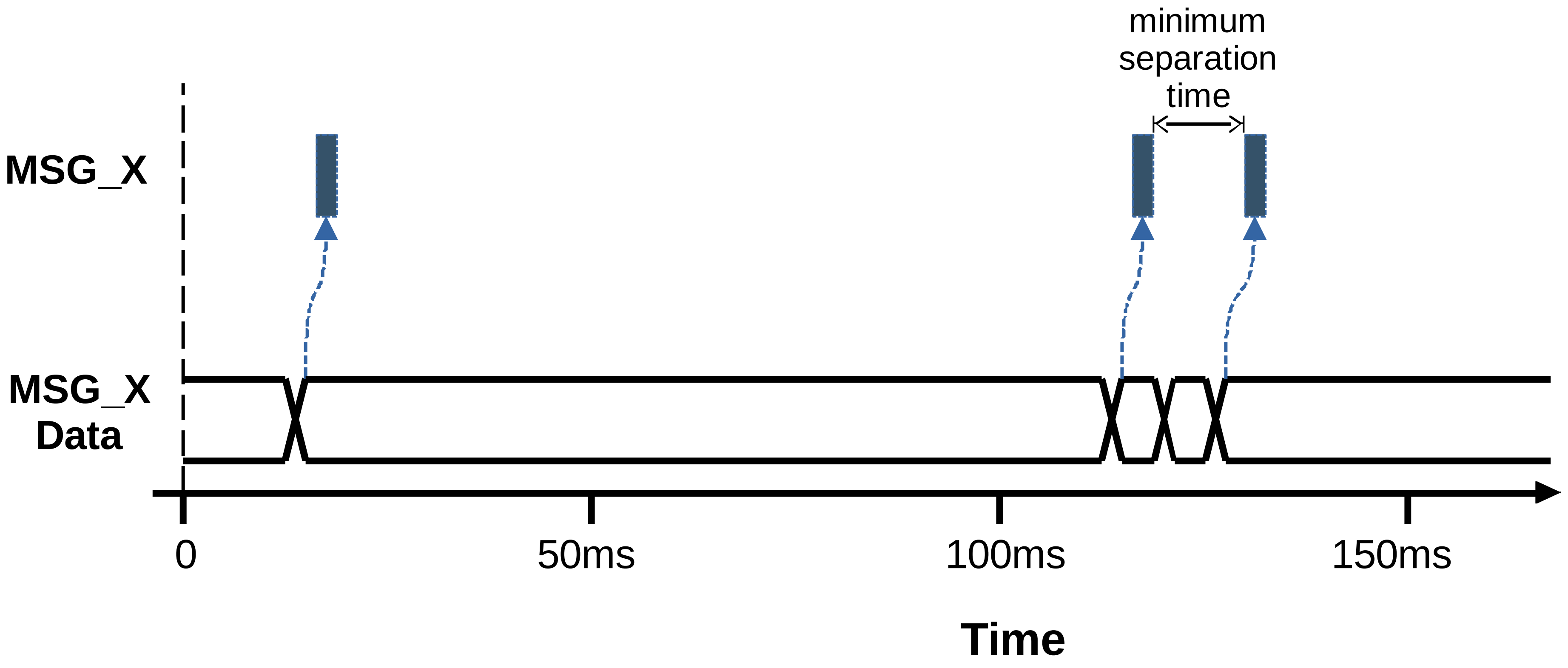
- **Spontaneous messages**
  - Transmitted only when the data of the message changes

    → Conveyed signal changes
  - No defined period
  - A minimum separation time is defined

    → e.g., 20ms

Transmission Types

- **Spontaneous messages**

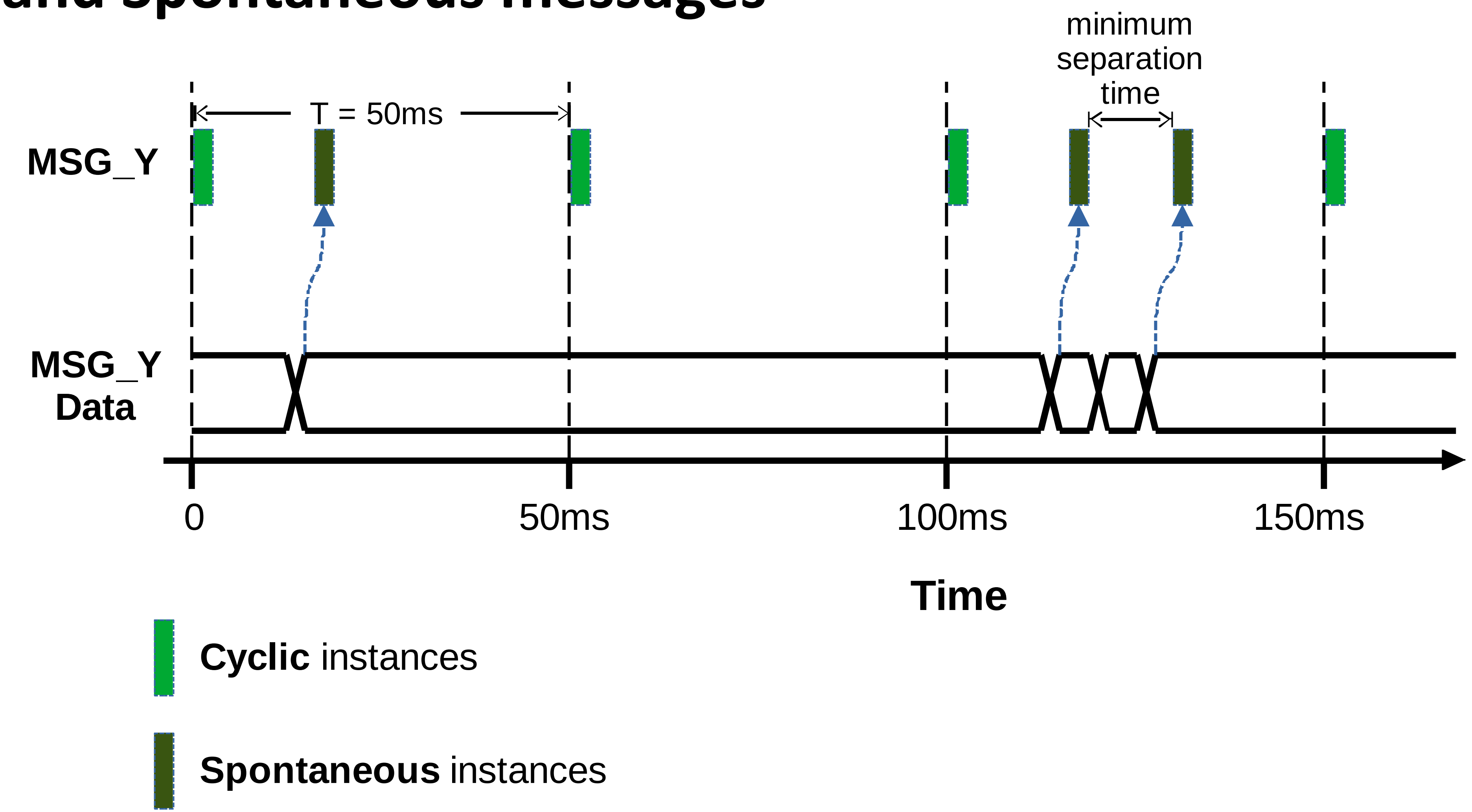


Transmission Types

- **Cyclic and Spontaneous messages**
  - Have a defined period for transmission but are also transmitted if there is a data change in between periodical instances
  - Period needs to be defined
  - A minimum separation time is defined
  - Cyclic and spontaneous message instances need to be handled properly, e.g., to respect minimum separation time, etc.

Transmission Types

- **Cyclic and Spontaneous messages**

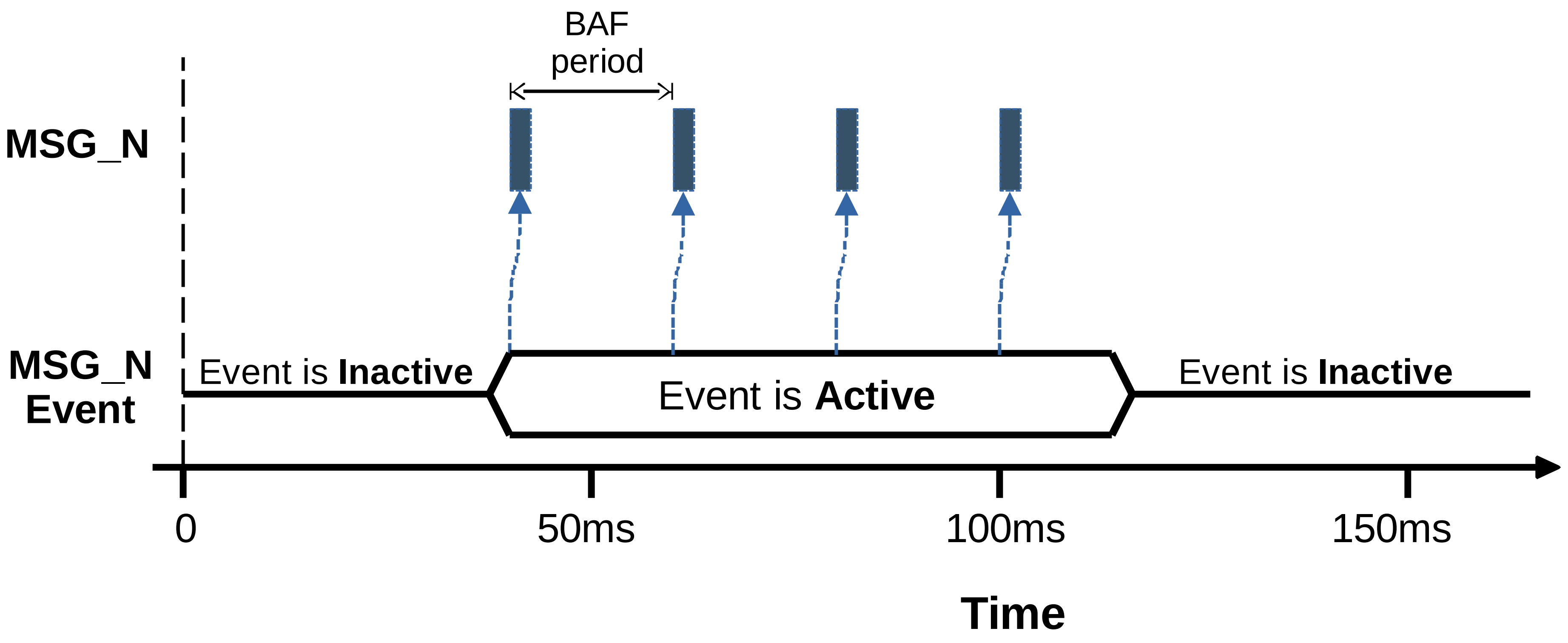


Transmission Types

- **By-Active-Function**
  - Message transmission is associated to the occurrence of an “event”
  - If the event hasn’t occurred or is not active, then there is NO message transmission
  - If the event occurs or becomes active, then the messages start being transmitted with a defined period
  - Message stops being transmitted if the event is no longer active or if certain number of transmissions have been executed

Transmission Types

- **By-Active-Function**



Network Design

- CAN Network is designed "**offline**"
- Once defined, each nodes gets **implemented** according to their applicable messages/signals
- Common **formats** to document a CAN network design
  - Tables (e.g., MS Excel, LibreOffice Calc)
  - DBC files (Vector proprietary format)
  - ARXML files (AUTOSAR)

Network Design

- Basic steps for **designing a CAN network**

  ***Note**: Steps shown here are not exhaustive, just an example. Some steps have dependencies from other steps, so the order mentioned here is not strict*

  - Define **Network characteristics**
    - Bit-rate (according to data needs, wire length, etc.)
    - Physical bus to use (single-wire, two-wire)

Network Design

- Basic steps for **designing a CAN network**
  - Define Participant **Nodes**
    - Node name
    - Node description
    - Map nodes to ECUs

      → E.g., Gateway ECUs implement multiple nodes of different CAN networks

Network Design

- Basic steps for **designing a CAN network**
  - Define Data **Signals**
    - Signal **Name**
    - Signal **description**
    - Signal **characteristics**
      - Length
      - Unit, resolution, range, enumerated values, default value, etc.

Network Design

- Basic steps for **designing a CAN network**
    - Define Network **Messages**
        - Message **Name**
        - Message **description**
        - Message **ID** (lower value → higher priority)
        - Message **Format** (standard, extended)
        - Message **transmission type** (cyclic, spontaneous, etc.)

Network Design

- Basic steps for **designing a CAN network**
  - Define **Message Publisher** and **Subscriber(s)**
    - Which node will transmit the message?
    - Which node or nodes will receive the message?

      → Define timeout times for subscriber nodes if required

Network Design

- Basic steps for **designing a CAN network**
  - **Pack signals** into **messages**
    - Assign signals to messages
    - Define signal layout within message:
      - Starting Byte
      - Starting Bit

Thanks for your attention!

Questions, Comments?

Bibliography

- **ISO 11898:2015**, “Road vehicles — Controller area network (CAN)”, International Standards Organization, 2015
- **Online** sources:
  - https://elearning.vector.com/mod/page/view.php?id=333
  - https://www.autovision-news.com/electrification/changes-vehicle-electrical-architecture/
  - https://resources.altium.com/p/Controller-Area-Network-Bus-Introduction-and-History
  - https://en.wikipedia.org/wiki/CAN_bus
  - https://www.csselectronics.com/pages/can-bus-simple-intro-tutorial
  - https://store.chipkin.com/articles/can-bus-protocol-page-2
  - https://www.linkedin.com/pulse/hitchhikers-guide-hacking-connected-cars-ecus-alissa-valentina-knight/

Bibliography

- **Stock** images:
  - https://www.freepik.com/free-photo/metallic-color-sport-car-bridge_5896016.htm#page=6&query=automotive&position=30&from_view=search&track=sph