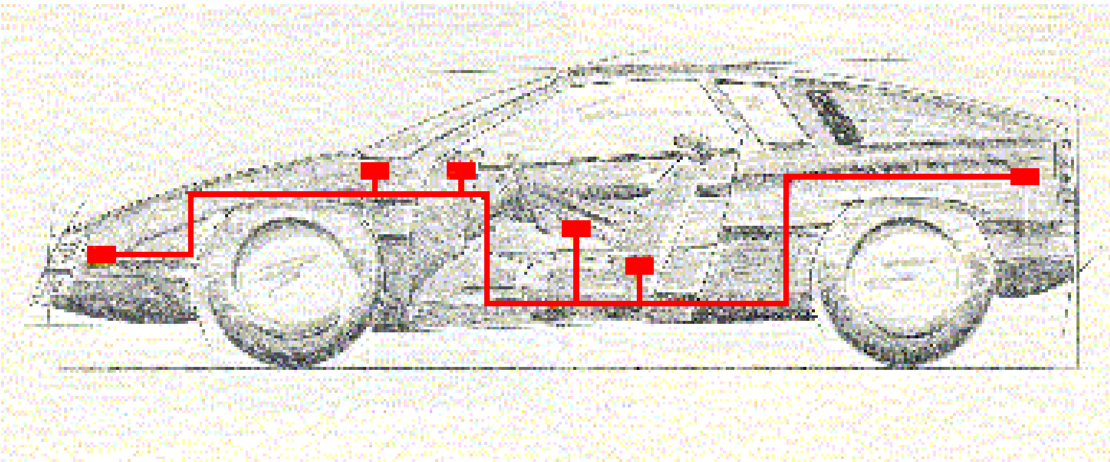
iha.dk

Distributed Real-Time Systems (TI-DRTS) – Track 2

CAN-BUS Introduction

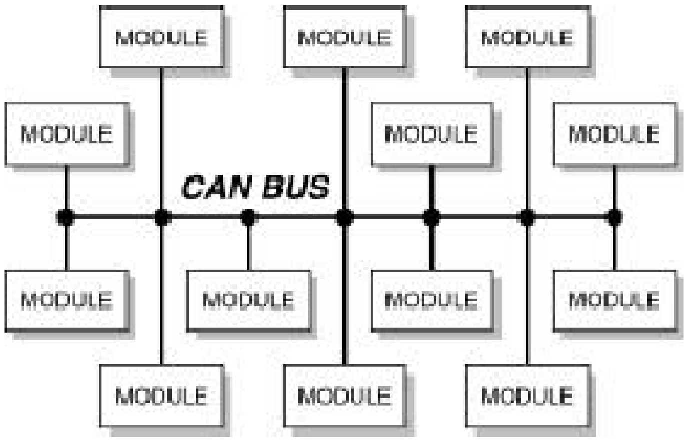


Version 9.11.2009

Ref. VECTOR application note & Motorola note

What is CAN?

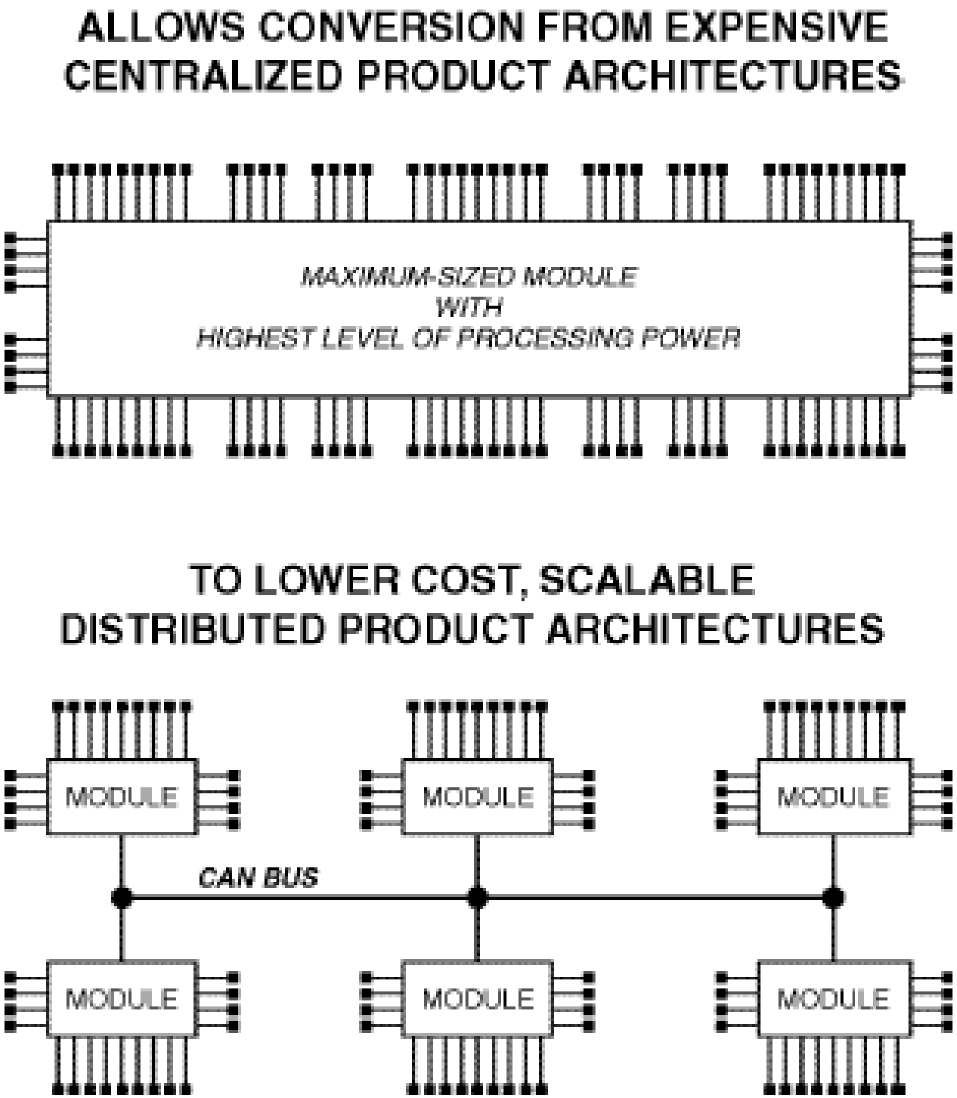
- **Controller Area Network (CAN)** is a common, small area network solution that supports distributed product and **distributed system architectures**
- The CAN bus is used to interconnect a network of electronic nodes or modules
- Typically, **a two wire, twisted pair cable** is used for the network interconnection



CAN - Highlights

- Is a high-integrity serial data communications bus for real-time applications
- Is event driven
- Operates at data rates of **up to 1 Mbits/s**
- Has excellent error detection capabilities
- Was originally developed by Bosch for use in cars
- Is now being used in many other industrial automation and control applications
- Is an international standard: **ISO 11898**

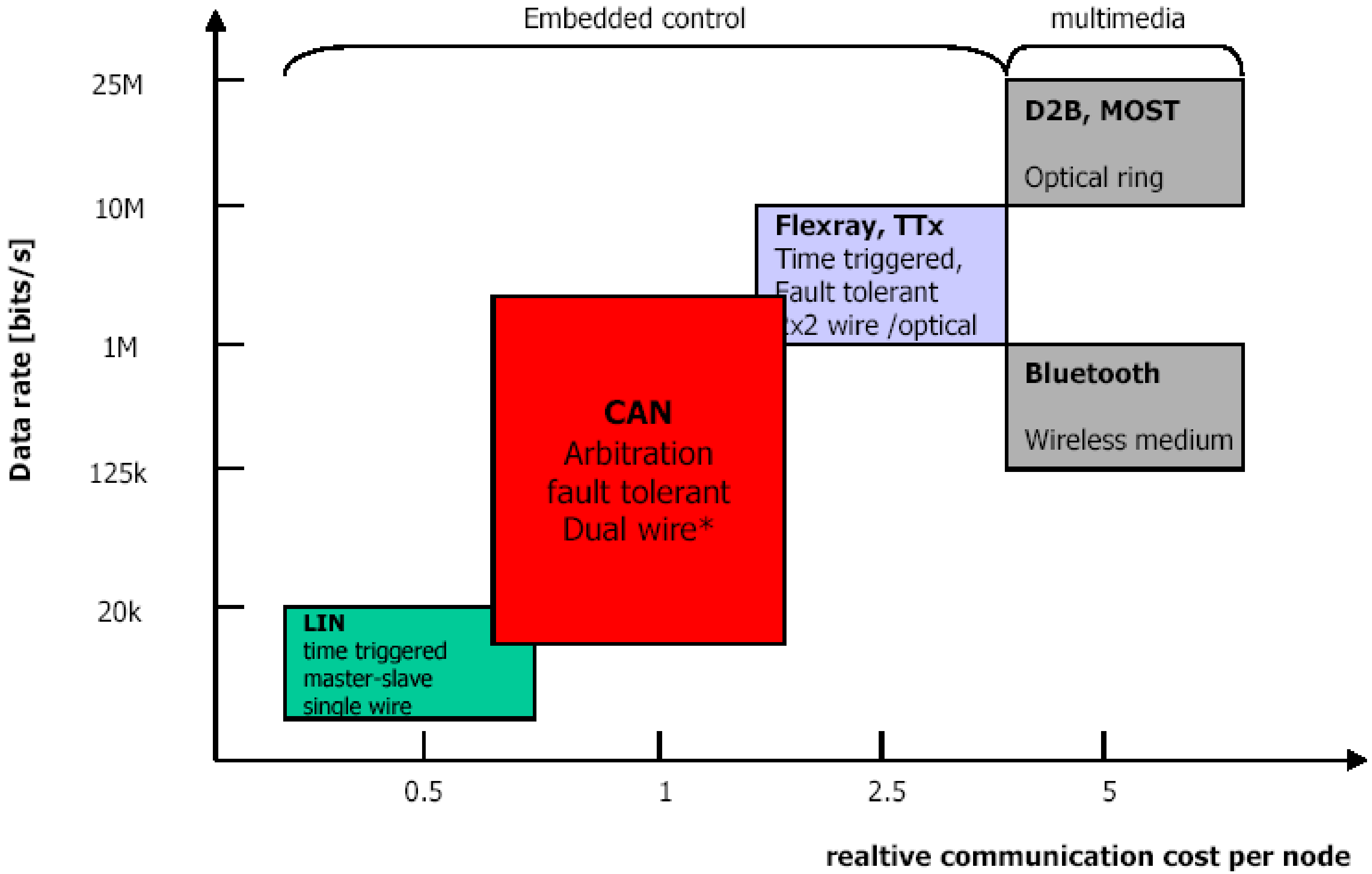
Why use CAN?


ALLOWS CONVERSION FROM EXPENSIVE
CENTRALIZED PRODUCT ARCHITECTURES
MAXIMUM-SIZED MODULE
WITH
HIGHEST LEVEL OF PROCESSING POWER
TO LOWER COST, SCALABLE
DISTRIBUTED PRODUCT ARCHITECTURES
MODULE
MODULE
MODULE
CAN BUS
MODULE
MODULE
MODULE

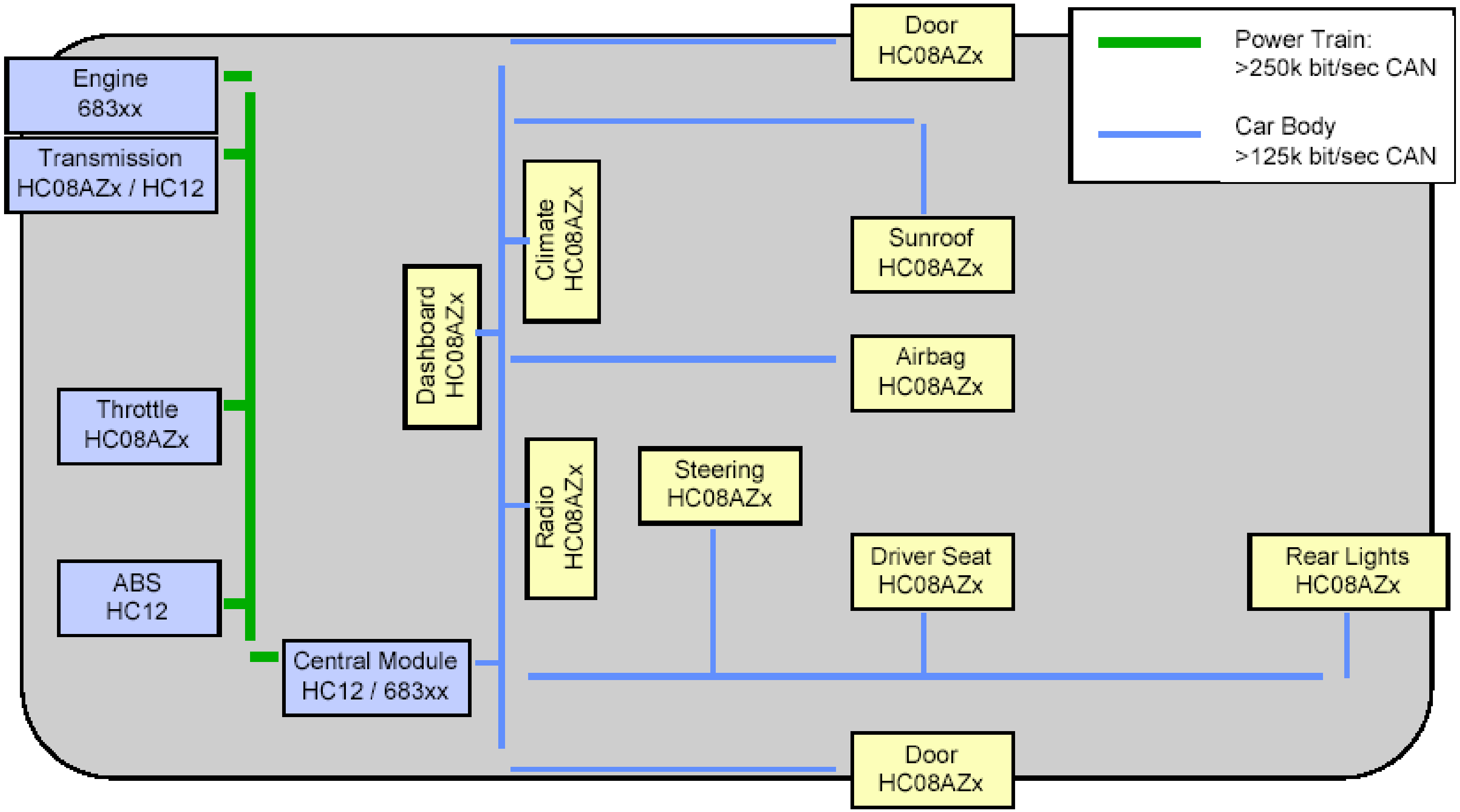
CAN History

- CAN first introduced by Bosch in 1986
- Bosch published CAN specification version 2.0 in 1991
- Official ISO CAN standard in 1993: **ISO 11898**
- In 1999 57 million CAN controller chips sold
- Estimated to 300 million CAN chips in 2003

Field Buses in the Automotive Industry



Typical CAN Network



CAN Standards

- Although CAN was originally **developed in Europe** by Robert Bosch for automotive applications, the protocol has gained wide acceptance and has become **an open, international ISO standard**
- As a result, the Bosch **CAN 2.0B** specification has become the de facto standard that new CAN chip designs follow
- **CAN ISO Standardization:**
  - ISO PRF 16845 : CAN Conformance Test Plan
  - ISO PRF 11898-1: CAN Transfer Layer
  - ISO PRF 11898-2: CAN High Speed Physical Layer
  - ISO DIS 11898-3: CAN Fault Tolerant Physical Layer
  - ISO DIS 11898-4: **TTCAN Time Triggered CAN**

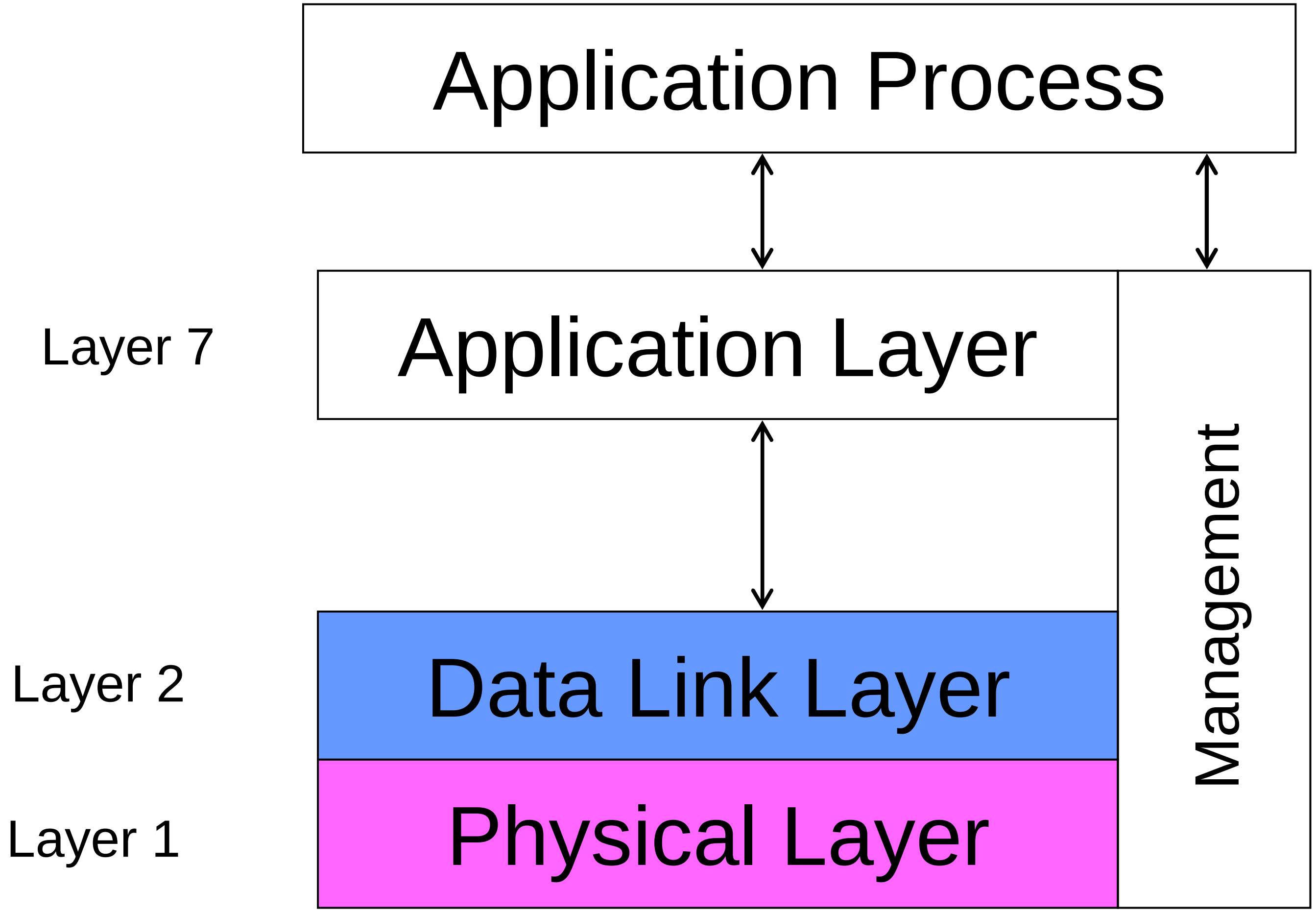
CAN 2.0 Structure of a CAN Node

ISO/OSI Reference Model

Layer 7:	Application Layer
Layer 6:	Presentation Layer
Layer 5:	Session Layer
Layer 4:	Transport Layer
Layer 3:	Network Layer
Layer 2: Data Link Layer	**Logic Link Control** Data Transfer Remote Data Request Message Filtering Recovery Management & Overload Notification
	Medium Access Control Framing & Arbitration Error Checking & Error Flags Fault Confinement Bit Timing
Layer 1:	Physical Layer

CAN Specification, **ISO 11898**, deals only with the **Physical** & **Data Link** Layers for a CAN network

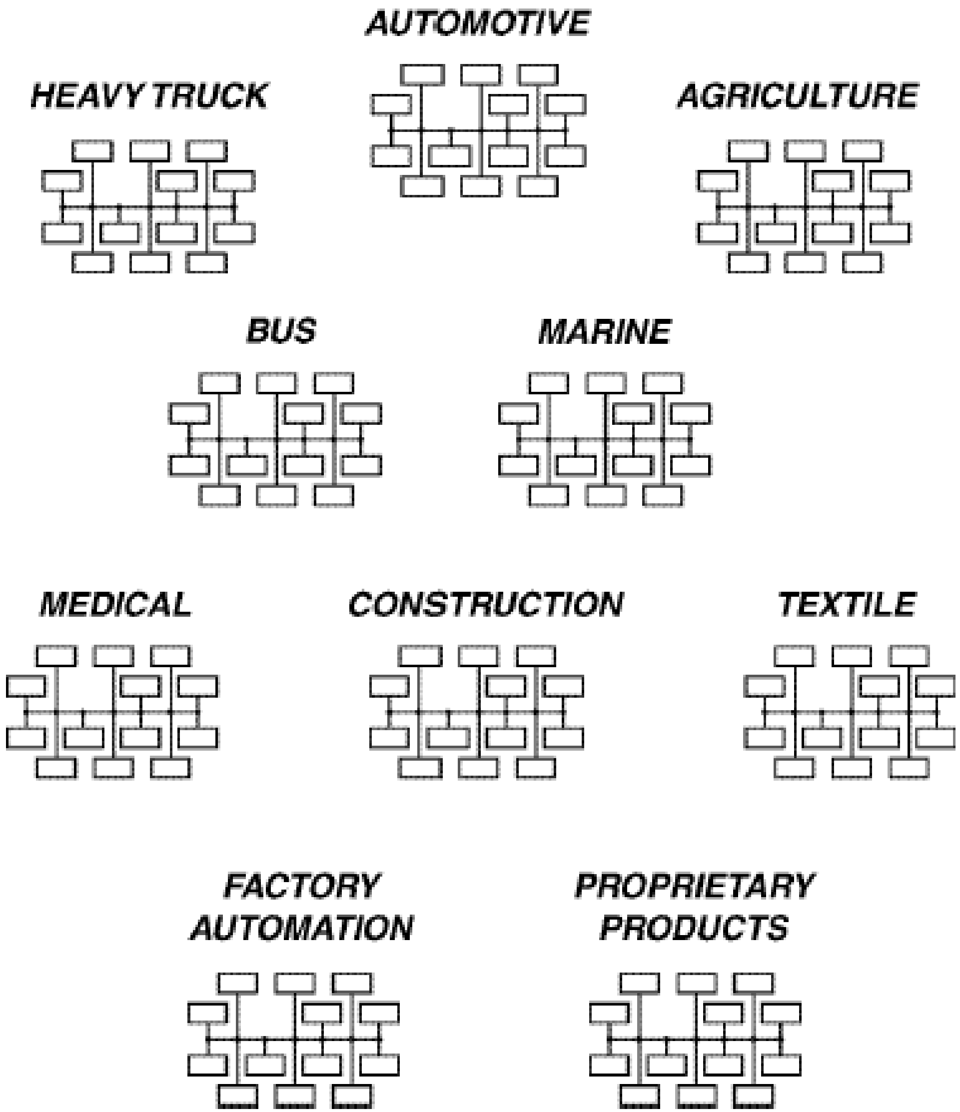
Three-Layer Reference Model



Key Reasons to Use CAN

- Low connect cost
- Low cost components
- Growing number of CAN chips
- Increasing knowledge base
- Increasing integration service base
- Wide variety of CAN-based products
- Wide variety of Off-the-Shelf tools available
- Potential lower wiring costs
- Lower weight

What Industries are using CAN?



CAN Higher Layer Protocols (1)

- CAN is used as the basis for several major "7-layer" protocol developments such as:
  - **CAL: CAN Application Layer** (CiA: Can In Automation)
  - **CAN Kingdom** (Kvasar)
  - **CANopen** (CiA: Can In Automation)
  - **DeviceNet** (Rockwell Automation, ODVA)
  - **Volcano** (Developed by Volvo)
  - **SAE J1939** (Society of Automotive Engineers)
  - **TTCAN: Time Triggered CAN** (Bosch)

CAN Higher Layer Protocols (2)

- Each of these large protocol architectures are essentially **complete industry-specific network solutions** packaged to include defined requirements for the:
  - physical layer,
  - address structure & message structure
  - conversation structure
  - data structure
  - application/network interface

How does CAN operate?

- CAN is a multiplexed serial communication channel
- Can transfer up to 8 data bytes within a single message
- For larger amounts of data, multiple messages are commonly used
- Most CAN based networks select a single bit rate
  - While communication bit rates may be as high as **1 MBit/s**, most implementations are **500Kbit/s** or less
- CAN supports data transfers between multiple peers
- **No master controller** is needed to supervise the network conversation

Bit Rate versus Bus Length

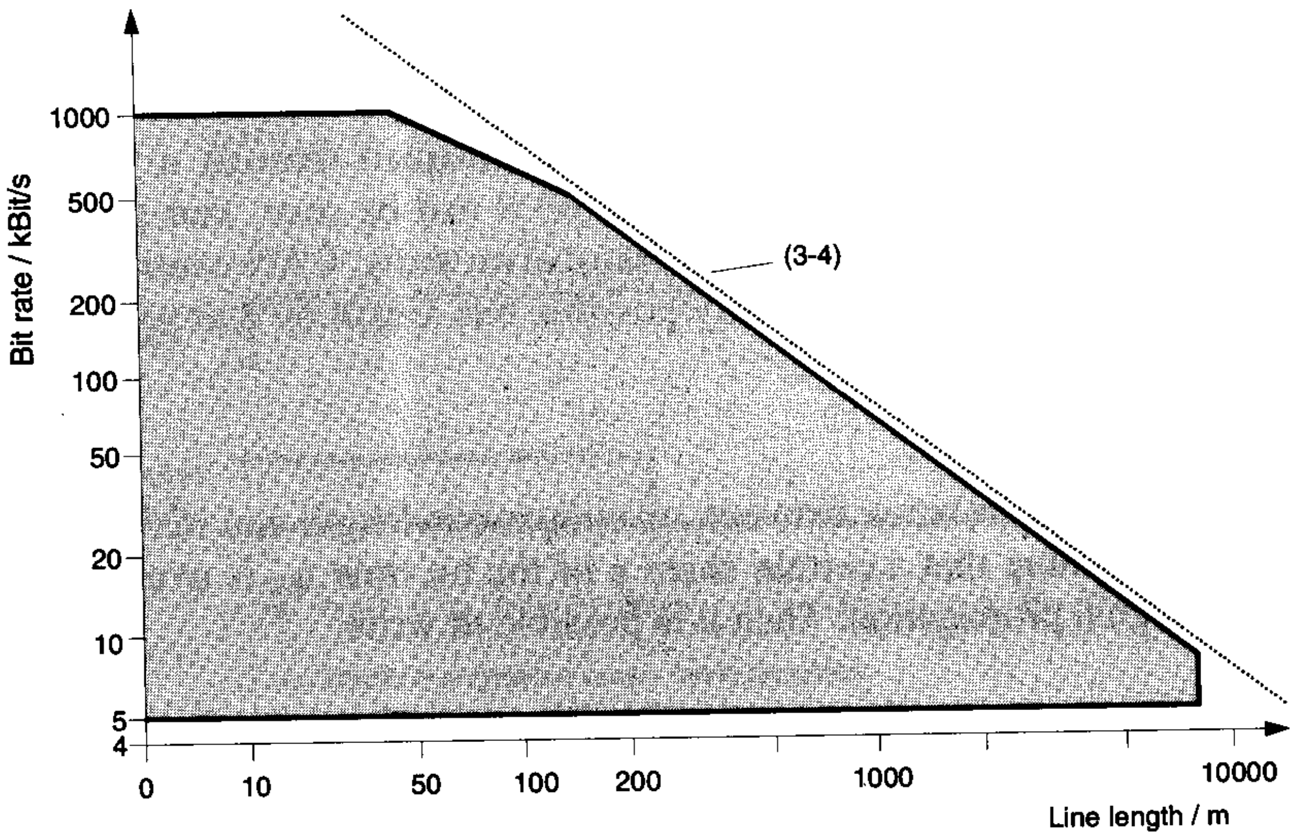
Bit Rate (kBits/s)	Maximum Bus length (m)
1000	**50**
500	110
135	620
100	790
50	1640

A Rule of Thumb for bus length > 100 m:

Bit Rate (Mbit/s) * Lmax (m) <=60

[Ref: Etschberger]

Bit Rate versus Bus Length



[Ref: Etschberger]

CAN Identifiers

- Labels the content (type) of a message used by receivers to select a message
- Used for arbitration & determines the **priority** of the message
  - Low id.number = high priority

CAN 2.0A vs CAN 2.0B

CAN 2.0A:
11 Bit Identifier

M68HC05X Family

- Used by vast majority of current applications.
- **Greater message throughput** and **improved latency times**
- *Less silicon overhead !*

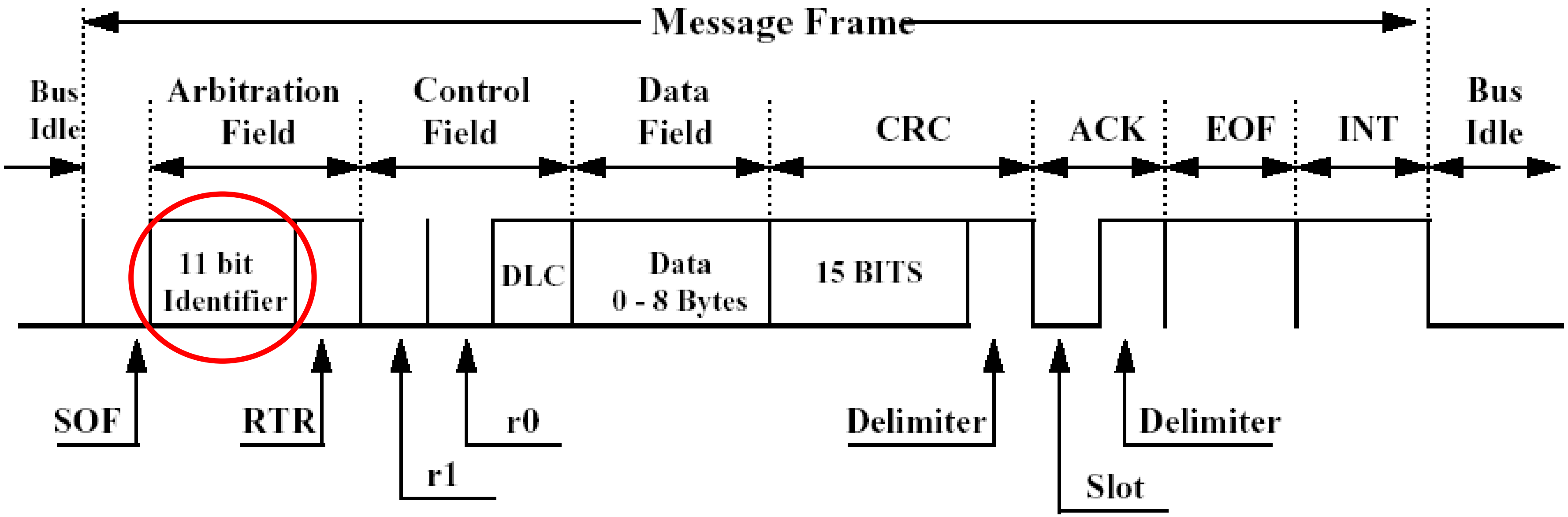
CAN 2.0B
29 Bit Identifier

HC08 / HC11 + MSCAN

- Originally defined for USA Passenger Cars but now their Taskforce decree that it is **not necessary**.
- Allows more information in message but requires **more bus bandwidth**
- ***More silicon cost and less efficient use of bus !***

CAN 2.0A Message Frame

- **CAN 2.0A (Standard Format)**
  - **11 bit message identifier (2048 different frames)**
  - **Transmits and receives only standard format messages**



CAN Message Frame

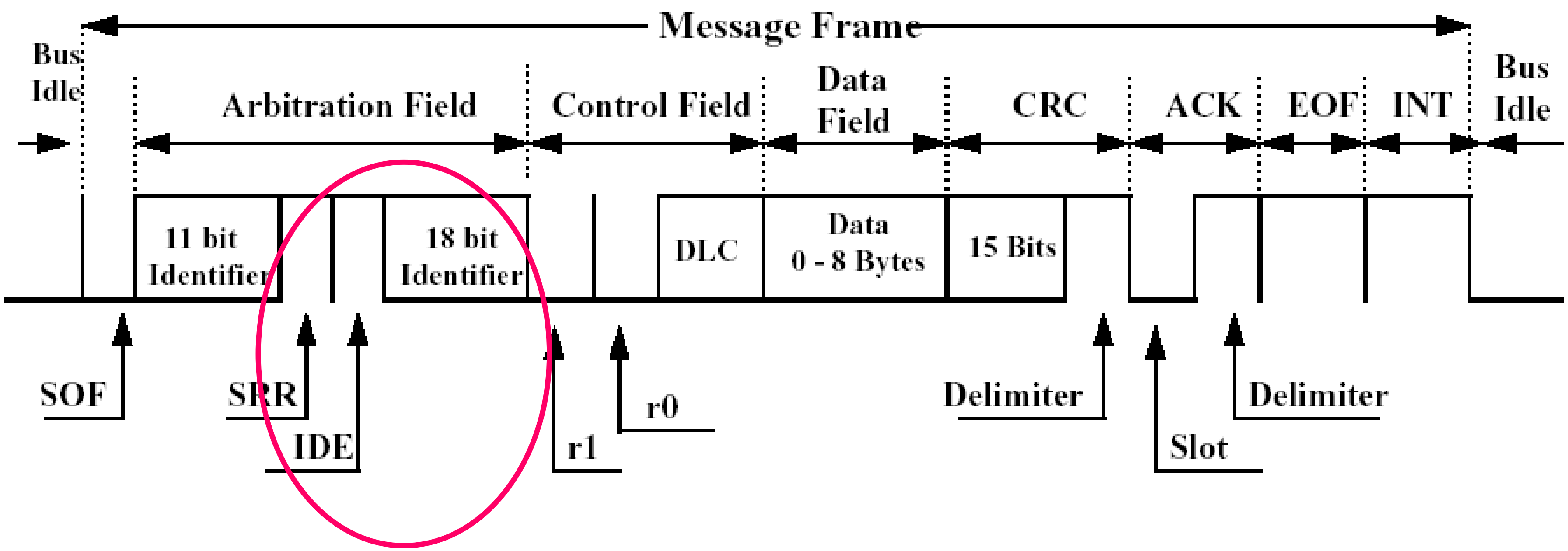
DATA FRAME:

- IS: Interframe space
- SOF: Start of frame, one single D-bit, start only if the bus is IDLE, all devices have to synchronize to the leading edge caused by START OF FRAME.
- ID: Identifier (CAN 2.0A (standard) = 11 bit, CAN 2.0B (extended) = 29 bit)
- RTR: Remote transmission request
  - D-bit: data follows = DATA FRAME
  - R-bit: transmission request to receiver = REMOTE RAME
- DLC: Data Length Code = 6 bit, C[3] - C[0] length of data array, MSB first
  - REMOTE FRAME: number of requested data bytes
  - C[5], C[4] are used for indicating extended IDs (2.0B)
- CRC: Cyclic redundancy checksum; 15 bit and a leading 0, sum and a R-bit delimiter bit
- ACK: Acknowledge (2 bits: ACK slot a and ACK delimiter)
  - The bit in ACK slot is sent as a R-bit and overwritten as a D-bit by those transducers which have received the message correctly.
- EOF: End of frame (7 R-bits)

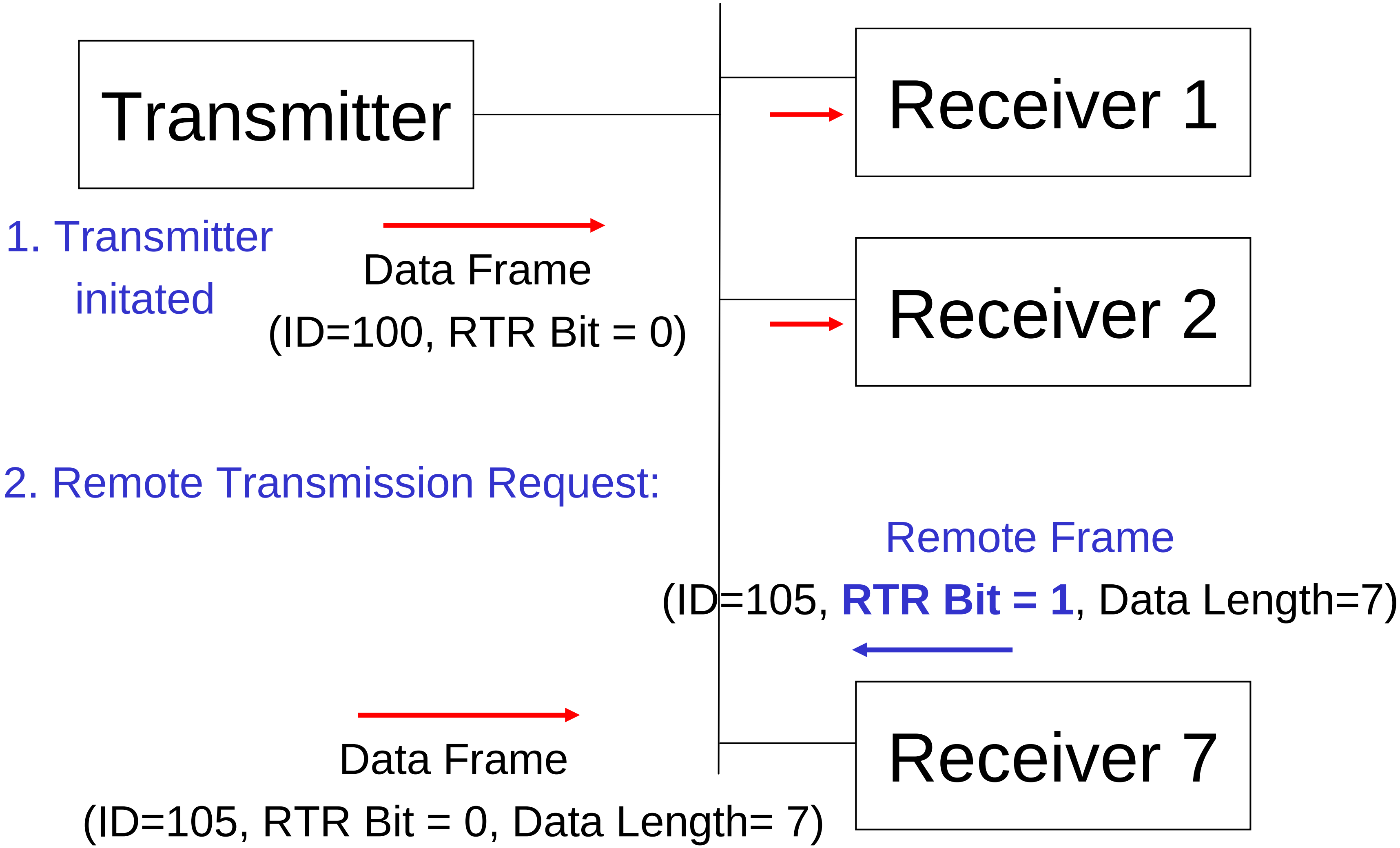
Bit	>3	1	11,1	6	0...64	16	2	7
	IS	SOF	ID, RTR	DLC	DATA	CRC	ACK	EOF

CAN 2.0B Message Frame

- **CAN 2.0B (Extended Format)**
  - **Capable of receiving CAN 2.0A messages**
  - **29 bit message identifier (512 million frames)**
  - **11 bits for a CAN 2.0A message + 18 bits for a CAN 2.0B message**



Two Communcation Types



Arbitration (1)

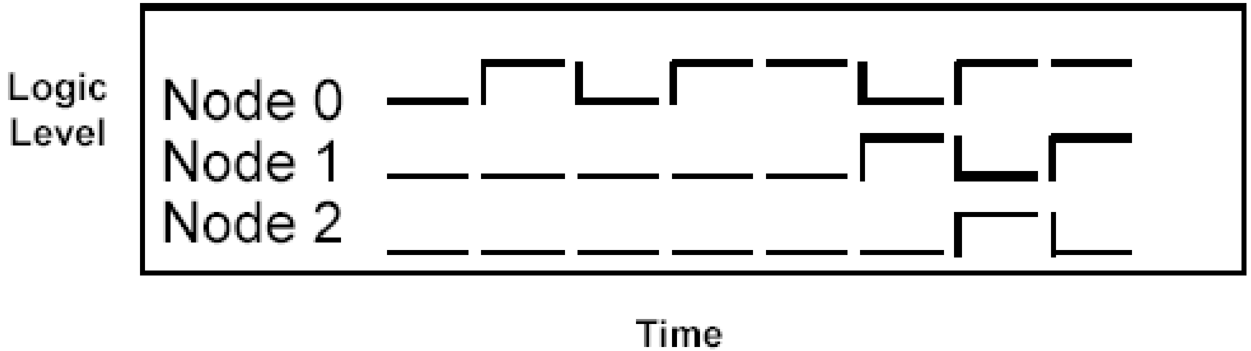
- Carrier Sense, Multiple Access **with Collision Avoidance (CSMA/CA)**
- Method used to arbitrate and determine the priority of messages
- Uses enhanced capability of non-destructive bitwise arbitration to provide collision resolution

Arbitration (2)

- A station may send if the bus is free (carrier sense)
- Any message begins with a field for unique bus arbitration containing the message ID
- The station with the **lowest ID is dominant** (D-Bit)
- **So the lowest ID has highest priority**
- Sending is not interfered since the propagation on the bus is much smaller than a duration of a bit

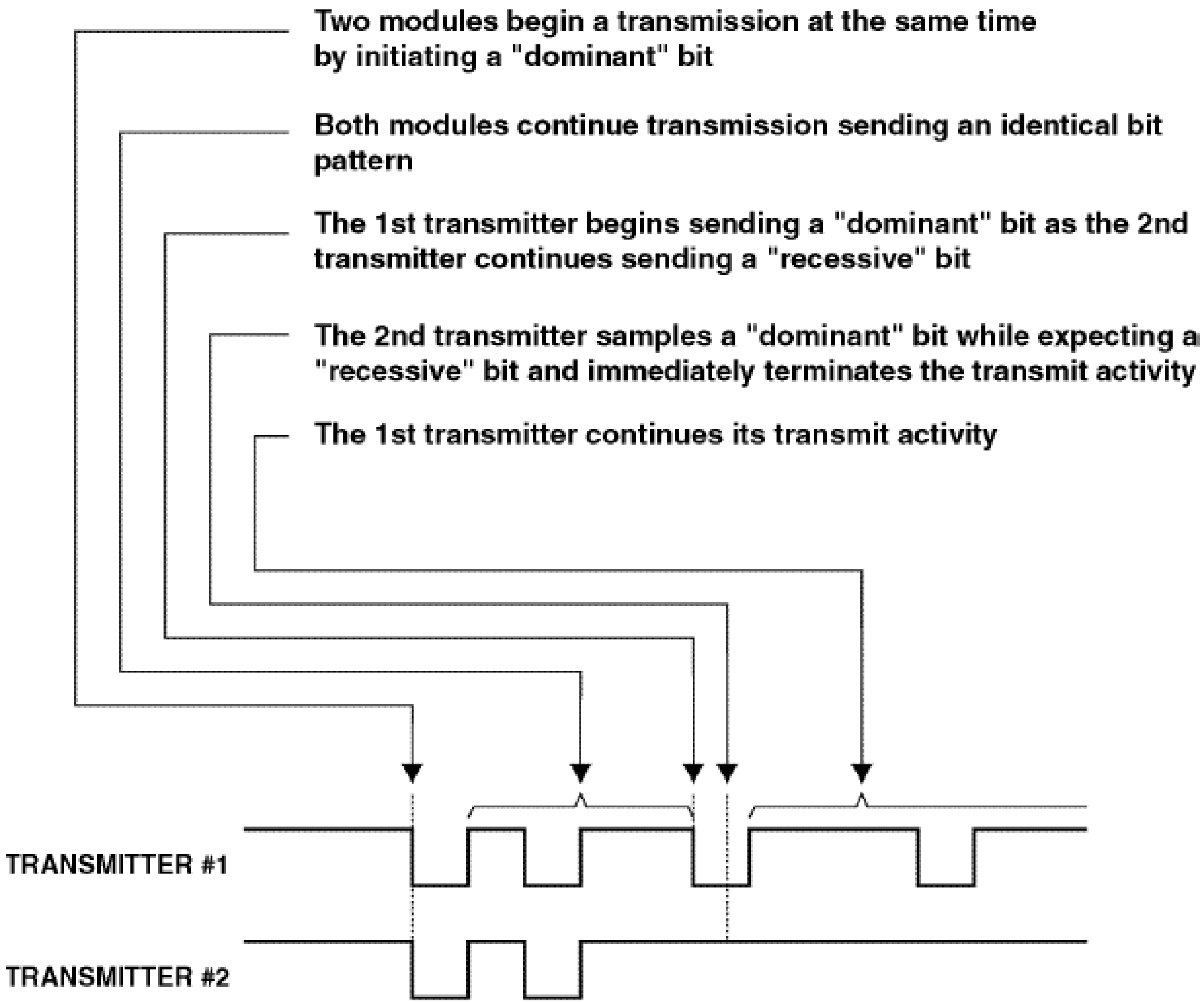
Bitwise Arbitration

- Any potential bus conflicts are resolved by bitwise arbitration
- **Dominant state (logic 0)** has **precedence** over a **recessive state (logic 1)**



- Competition for the bus is won by node 2 .
- Nodes 0 and 1 automatically become receivers of the message
- Nodes 0 and 1 will re-transmit their messages when the bus becomes available again

Example of Bitwise Arbitration



Qualities: Safe Collision and Tx-feedback

The CAN-controller has 2 important features:

- A **collision** do not destroy any message on the bus
  - All Tx's with recessive levels stops immediately and changes to Rx's.
  - The Tx-node with highest priority wins the bus and sends data
- Every **transmitted** message is evaluated by each receiving node, and if the received message is damaged the Tx-node is alerted with a **feedback** at dominant level, sent from the Rx-node

Error Detection

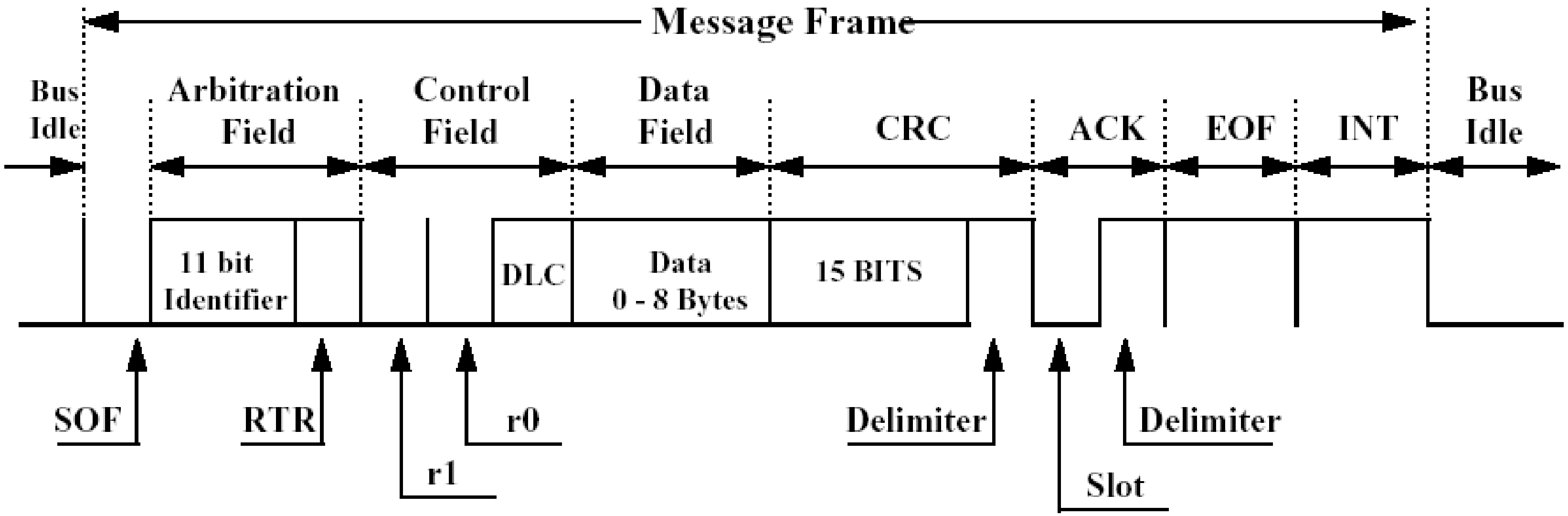
- CAN implements five error detection mechanisms
- Three at the **message level**
  - Cyclic Redundancy Checks (CRC)
  - Frame Checks
  - Acknowledgment Error Checks
- Two at **the bit level**
  - Bit Monitoring
  - Bit Stuffing

Cyclic Redundancy Check (CRC) (Message Level)

- The 15 bit CRC is computed by the **transmitter** based on the message content
- All **receivers** that accept the message, recalculates the CRC and compares against the received CRC
- If the two values do not match a **CRC error** is flagged

Frame Check (Message Level)

- If a receiver detects an invalid bit in one of these positions a **Form Error** (or Format Error) will be flagged:
  - CRC Delimiter
  - ACK Delimiter
  - End of Frame Bit Field
  - Interframe Space (the 3 bit INTermission field and a possible Bus Idle time)



ACK Error Check (Message Level)

- Each receiving node writes a dominant bit into the ACK slot
- If a transmitter determines that a message has not been ACKnowledged then an ACK Error is flagged.
- ACK errors may occur because of transmission errors because the ACK field has been corrupted or there is no operational receivers

Bit Monitoring (Bit Level)

- Each bit level (dominant or recessive) on the bus **is monitored by the transmitting node**
  - Bit monitoring **is not performed during arbitration** or **on the ACK Slot**

Bit Stuffing (Bit Level)

- Bit stuffing is used to guarantee enough edges in the NRZ bit stream to maintain synchronization:
  - After five identical and consecutive bit levels have been transmitted, **the transmitter will automatically inject (stuff) a bit** of the opposite polarity into the bit stream
  - Receivers of the message will automatically **delete (destuff) such bits**
  - If any node detects six consecutive bits of the same level, **a stuff error is flagged**

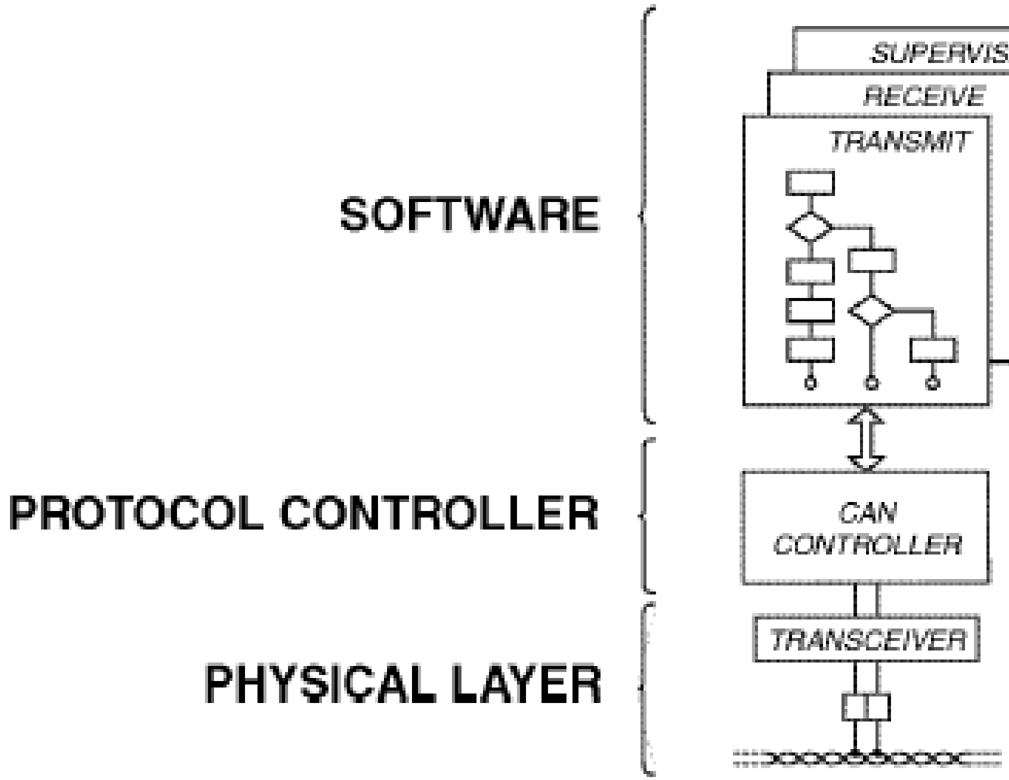
Error Flag

- If an error is detected by at least one node
  - The node that detects the error will immediately abort the transmission by sending an Error Flag
- An Error Flag consists of six dominant bits
  - This violates the bit stuffing rule and all other nodes respond by also transmitting Error Flags

What is needed to implement CAN?

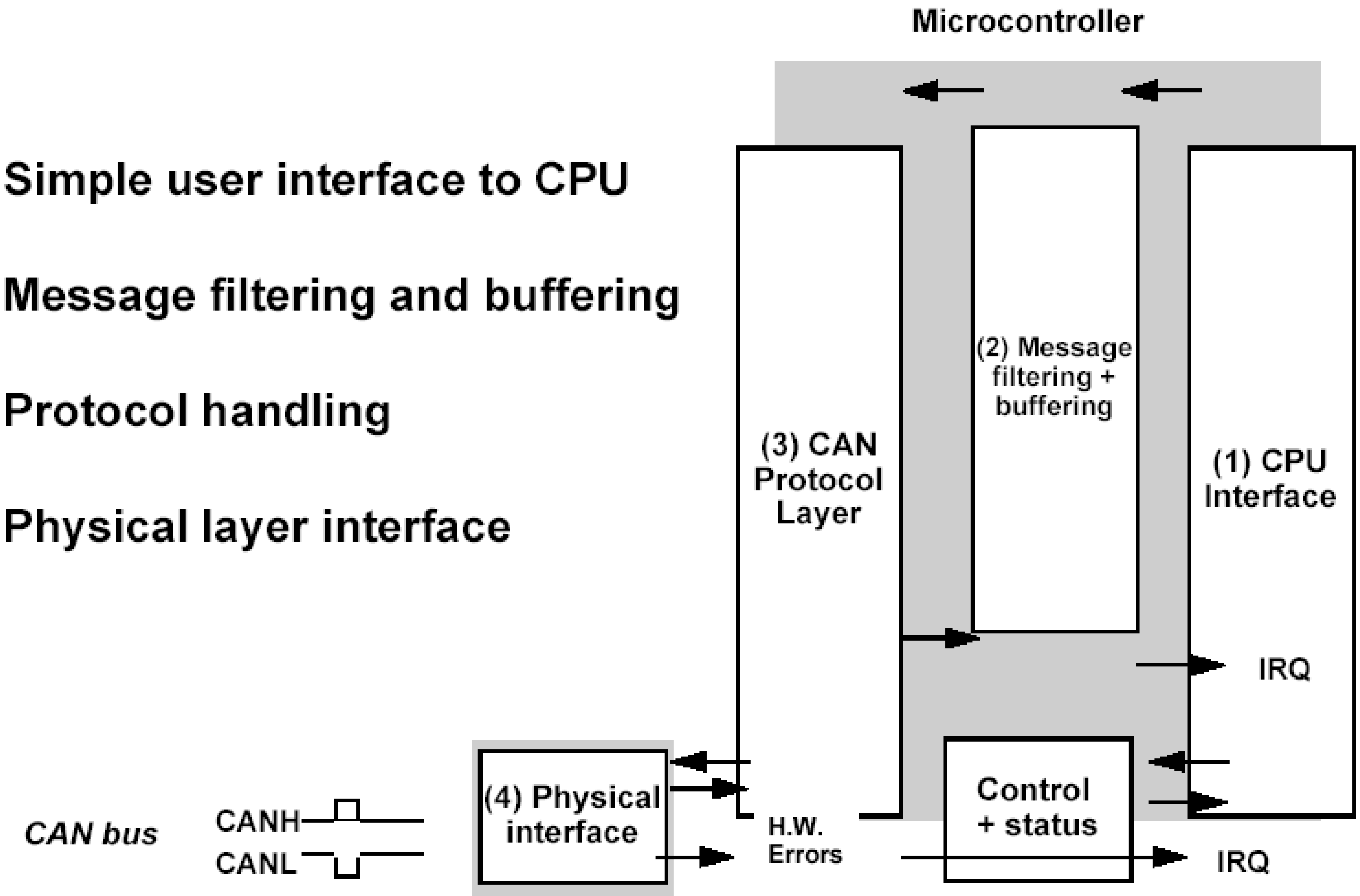
iha.dk

To implement CAN, three components are required - software, a CAN controller, and a physical layer



Requirements for a CAN Controller

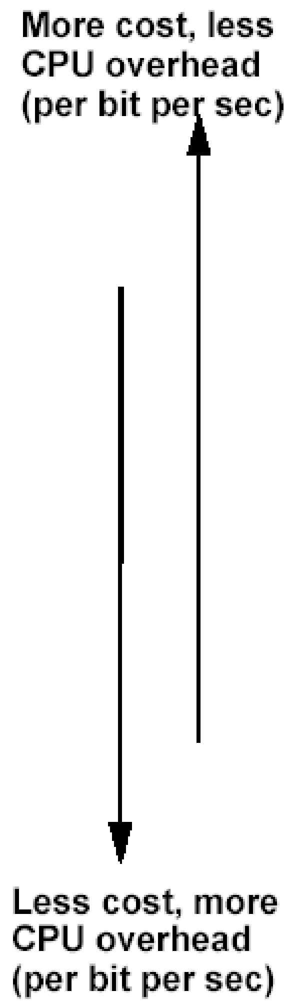
- Simple user interface to CPU
- Message filtering and buffering
- Protocol handling
- Physical layer interface



FullCAN vs BasicCAN Controller

- **FullCAN Controller:**
  - Typically 16 message buffers, sometimes more
  - Global and Dedicated Message Filtering Masks
  - Dedicated H/W for Reducing CPU Workload
  - More Silicon => more cost
    - e.g. Powertrain

- **BasicCAN Controller:**
  - 1 or 2 Tx and Rx buffers
  - Minimal Filtering
  - More Software Intervention
  - Low cost
    - e.g. Car Body



CANopen

- **CANopen**: a standardized application for distributed industrial automation systems
- Based on CAN standard and CAL (Can Application Layer)
- In **Europe** the definitive standard for the implementation of industrial CAN-based system solutions
- Standardized by CiA (CAN-in-Automation)
- Devices profiles
  - e.g. digital/analog I/O modules, drives, encoders, MMI-units, controllers
- Two types of communication mechanisms:
  - unconfirmed transmission of data frames to transfer process data
  - confirmed transmission of data (for configuration purpose)

DeviceNet

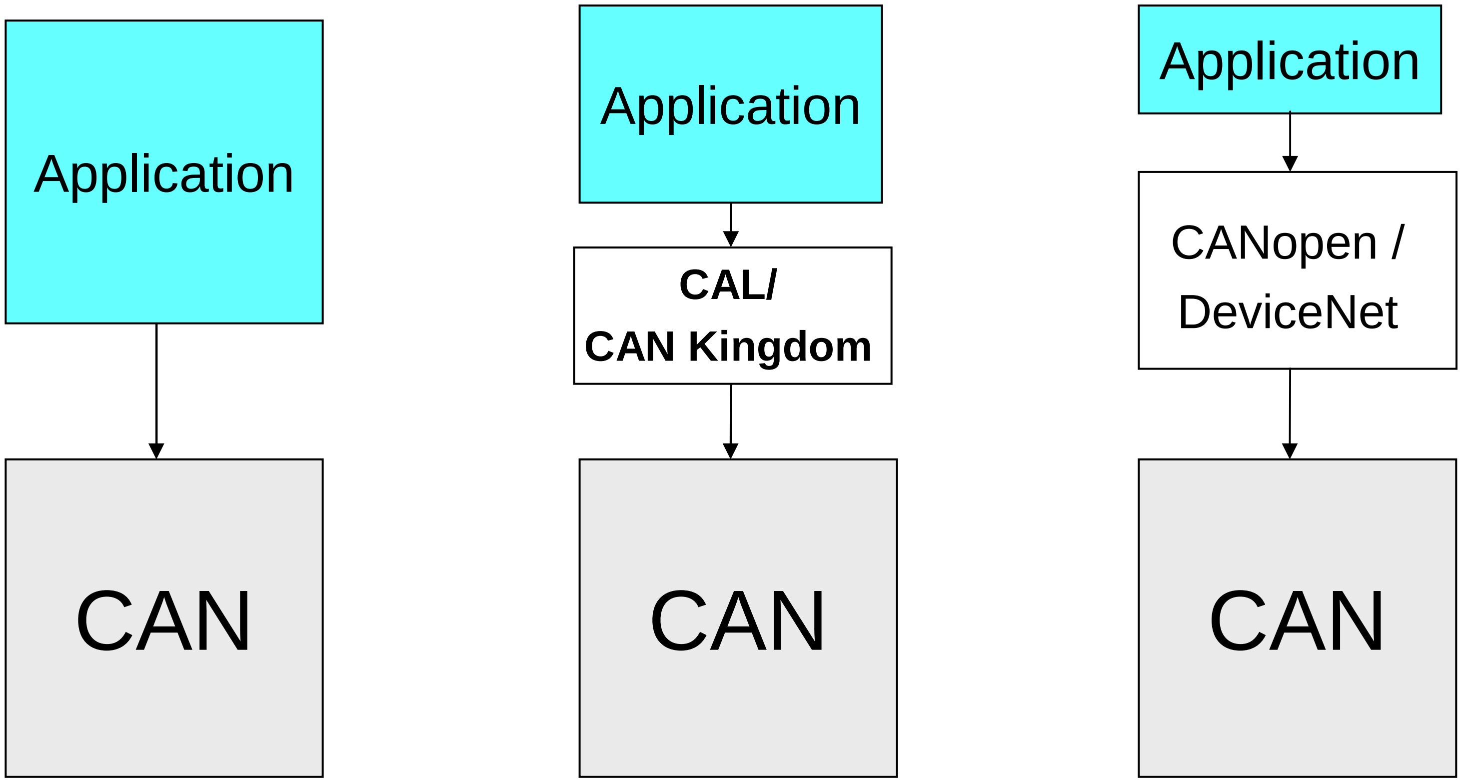
- **DeviceNet** developed by Rockwell Automation in 1995
- Main CAN automation technology in **USA** and **Asia**
- ODVA: Open DeviceNet Vendor Assocation (>300 members)
- DeviceNet is a connection-based communication model (ConnectionId = CAN identifier)
- Two message types: explicit and I/O messages
- Max 64 nodes in a DeviceNet network

Three Development Scenarios

based on a layer-2 implementation

based on a standard application layer

based on a standardized application profile



CAN Summary

- CAN is designed for asynchronous communication (event communication) with little information contents (8 bytes)
- Max 1MBit/s
- Useful for soft real-time systems
- Many microcontrollers comes with an integrated CAN controller
- A low-cost solution
- New invention:
  - TTCAN – a Time-Triggered CAN protocol

References

- [Etschberger]: "**Controller Area Network** – basics, protocols, chips and applications", by Konrad Etschberger, IXXAT Press, 2001
- www.can.bosch.com
  - Contains specification documents
  - References
  - Links
- ODVA: Open DeviceNet Vendors Association www.odva.org
- CiA: CAN in Automation http://www.can-cia.org/