CANopen in the Shell

Managing and Developing CANopen Applications on Linux

Thomas Flynn

September 29, 2017

Me (Tom Flynn):

- BEng. Mechatronics
- Linux User
- Industrial Electronics
- Engineer

Introduction- Outline

1. Controller Area Networks - What, Why, How?

Introduction- Outline

1. Controller Area Networks - What, Why, How?
2. Peeking at Protocols

Introduction- Outline

1. Controller Area Networks - What, Why, How?
2. Peeking at Protocols
3. CAN Support in the Kernel

Introduction- Outline

1. Controller Area Networks - What, Why, How?
2. Peeking at Protocols
3. CAN Support in the Kernel
4. Cometh the Application Layer

Introduction- Outline

1. Controller Area Networks - What, Why, How?
2. Peeking at Protocols
3. CAN Support in the Kernel
4. Cometh the Application Layer
5. The CANopen FOSS Tool Kit

Introduction- Outline

1. Controller Area Networks - What, Why, How?
2. Peeking at Protocols
3. CAN Support in the Kernel
4. Cometh the Application Layer
5. The CANopen FOSS Tool Kit
6. Smooth and Unproblematic Demonstration

Introduction- GNU Image Manipulation

A Timeline:

- **1980s** Continuing increase of electronic sensors in vehicles. Signalling limited by environmental noise, complexity of wiring, limited I/O on controllers.

A Timeline:

- **1980s** Continuing increase of electronic sensors in vehicles. Signalling limited by environmental noise, complexity of wiring, limited I/O on controllers.
- **1983** Development of the CAN bus concept at Robert Bosch GmbH

A Timeline:

- **1980s** Continuing increase of electronic sensors in vehicles. Signalling limited by environmental noise, complexity of wiring, limited I/O on controllers.
- **1983** Development of the CAN bus concept at Robert Bosch GmbH
- **1986** CAN bus 'protocol' released at SAE conference.

A Timeline:

- **1980s** Continuing increase of electronic sensors in vehicles. Signalling limited by environmental noise, complexity of wiring, limited I/O on controllers.
- **1983** Development of the CAN bus concept at Robert Bosch GmbH
- **1986** CAN bus 'protocol' released at SAE conference.
- **1987** First CAN transceiver chips released by Intel and Phillips

A Timeline:

- **1980s** Continuing increase of electronic sensors in vehicles. Signalling limited by environmental noise, complexity of wiring, limited I/O on controllers.
- **1983** Development of the CAN bus concept at Robert Bosch GmbH
- **1986** CAN bus 'protocol' released at SAE conference.
- **1987** First CAN transceiver chips released by Intel and Phillips
- **1988** First Production vehicle to feature CAN-based system.

A Timeline:

- **1980s** Continuing increase of electronic sensors in vehicles. Signalling limited by environmental noise, complexity of wiring, limited I/O on controllers.
- **1983** Development of the CAN bus concept at Robert Bosch GmbH
- **1986** CAN bus 'protocol' released at SAE conference.
- **1987** First CAN transceiver chips released by Intel and Phillips
- **1988** First Production vehicle to feature CAN-based system.
- **1993** ISO CAN standard released, ISO 11898

A Timeline:

- **1980s** Continuing increase of electronic sensors in vehicles. Signalling limited by environmental noise, complexity of wiring, limited I/O on controllers.
- **1983** Development of the CAN bus concept at Robert Bosch GmbH
- **1986** CAN bus 'protocol' released at SAE conference.
- **1987** First CAN transceiver chips released by Intel and Phillips
- **1988** First Production vehicle to feature CAN-based system.
- **1993** ISO CAN standard released, ISO 11898
- **1995** CAN in Automation (CiA) release CiA 301, CANopen application layer and communication profile - *MembersOnly*.

A Timeline:

- **1980s** Continuing increase of electronic sensors in vehicles. Signalling limited by environmental noise, complexity of wiring, limited I/O on controllers.
- **1983** Development of the CAN bus concept at Robert Bosch GmbH
- **1986** CAN bus 'protocol' released at SAE conference.
- **1987** First CAN transceiver chips released by Intel and Phillips
- **1988** First Production vehicle to feature CAN-based system.
- **1993** ISO CAN standard released, ISO 11898
- **1995** CAN in Automation (CiA) release CiA 301, CANopen application layer and communication profile - *MembersOnly*.
- **2011** CiA 301 V 4.2 made public - *Open*

Controller Area Networks - What, Why, How?- Definition

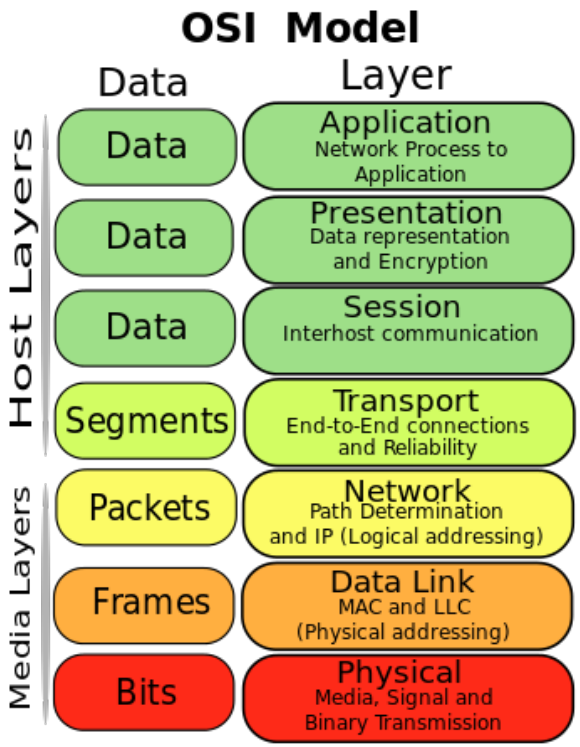
What is a Controller Area Network?

A network of nodes exchanging messages.

Controller Area Networks - What, Why, How?- Technology Context

OSI 7 Layer Model

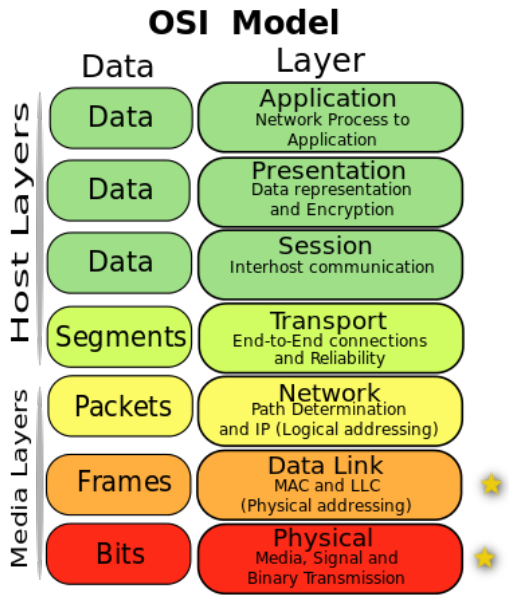
Observing the Open Standards Initiative (OSI) Model we can contextualise the aspects of CAN technology.



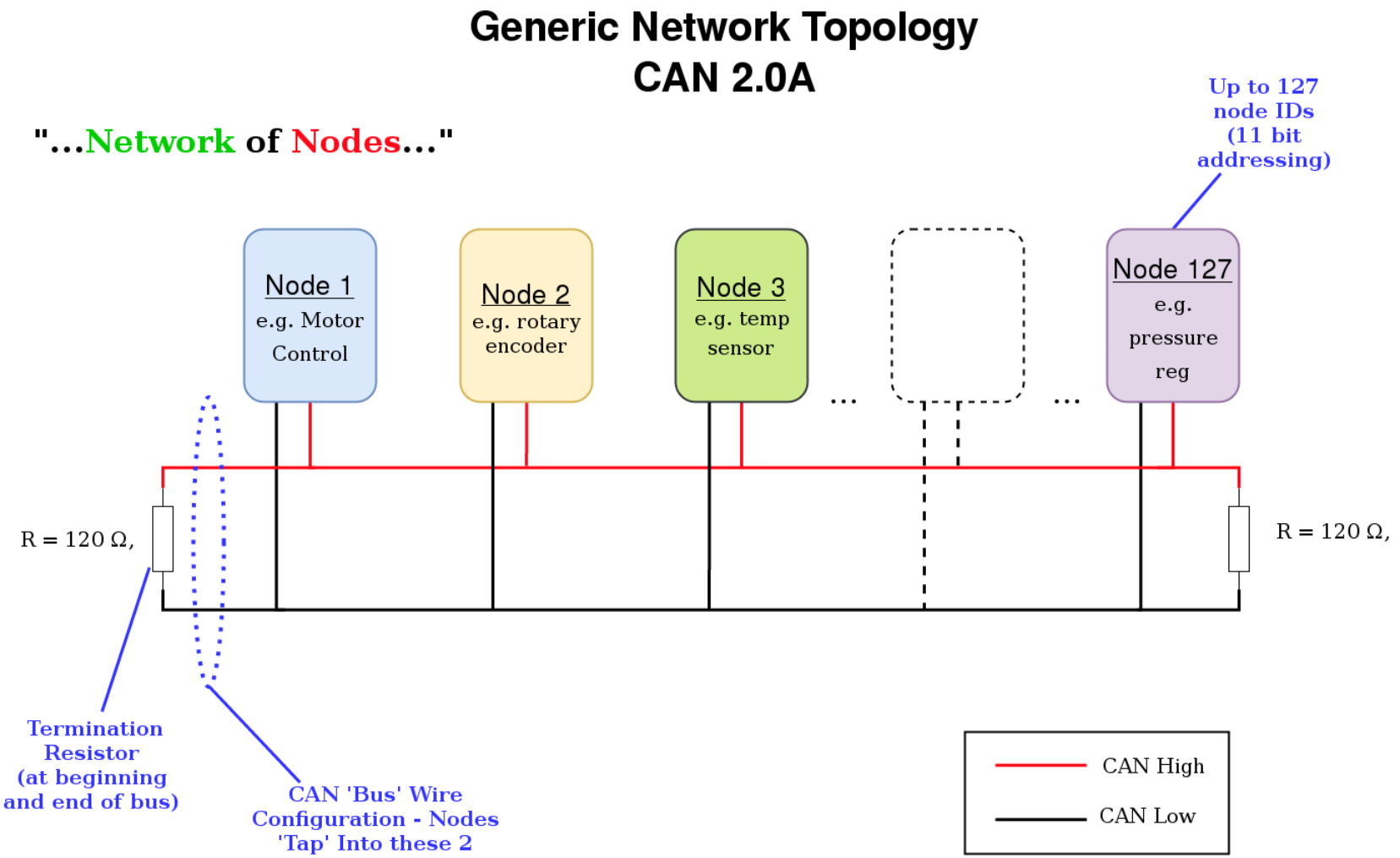
Controller Area Networks - What, Why, How?- CAN as Foundation

Layers 1 and 2

From the bottom up, a CAN network needs to be looked at in terms of its physical implementation. We will consider the Physical (L1) and Data Link (L2) layers.



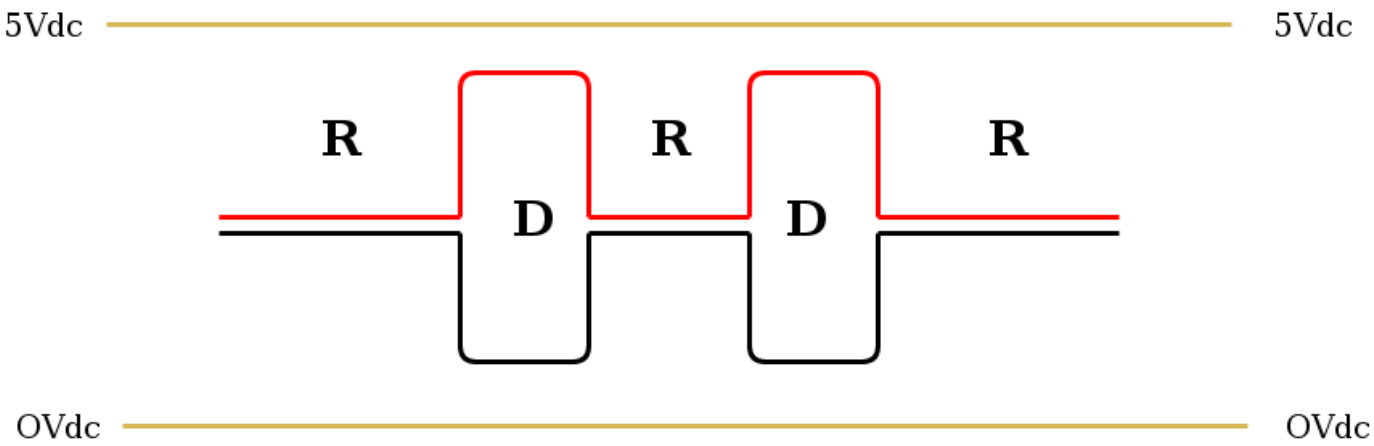
Controller Area Networks - What, Why, How?- Network



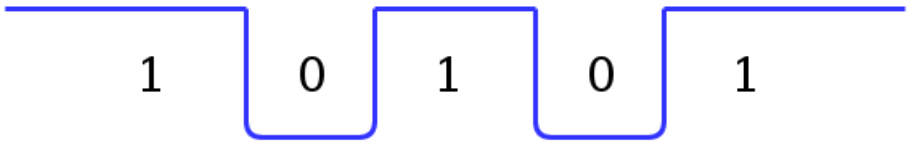
Controller Area Networks - What, Why, How?- Exchange

CAN Signaling

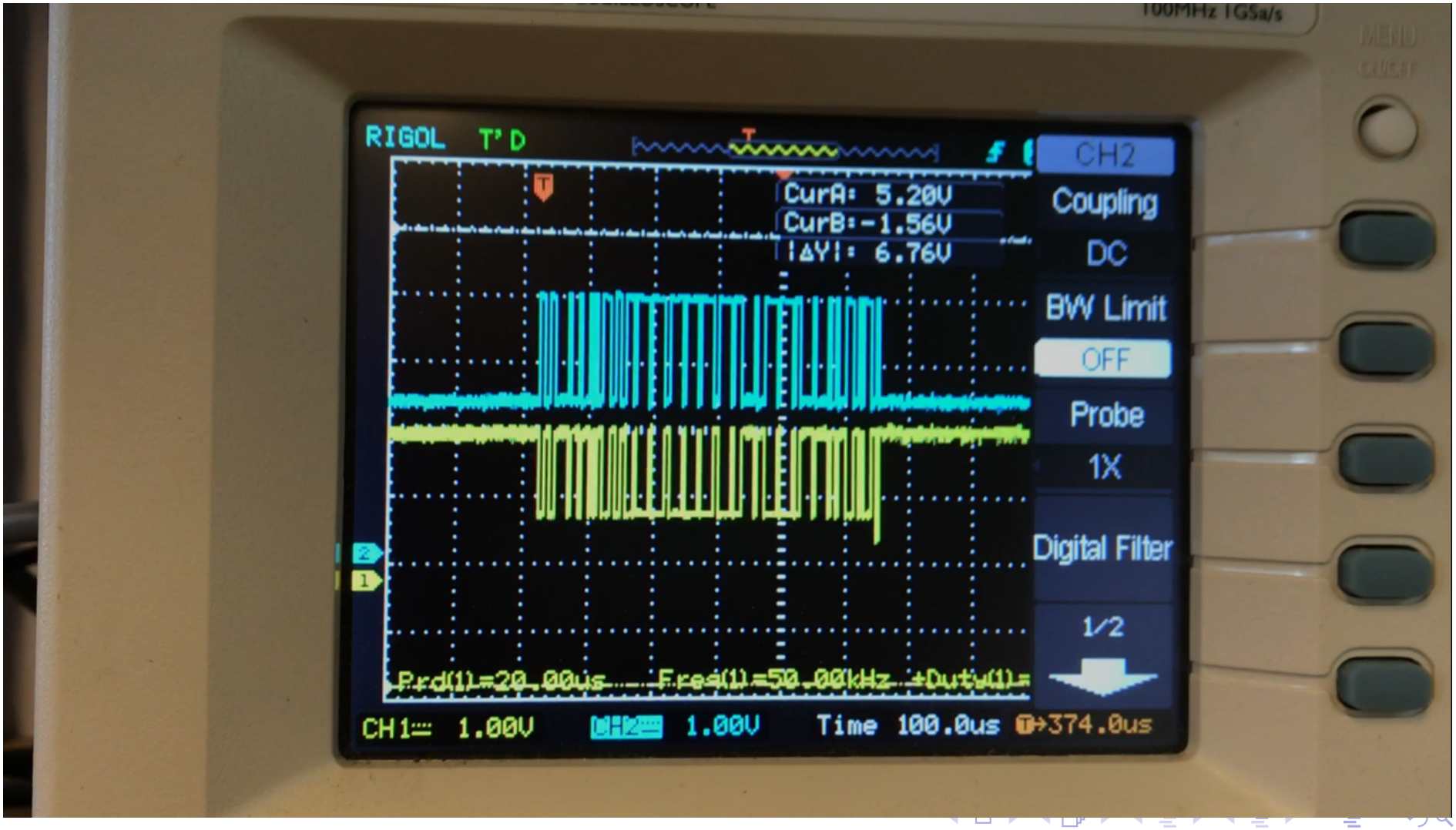
D - Dominant - Both lines in 'on' state - Logical 0
R - Recessive -Both lines in 'off' state - Logical 1



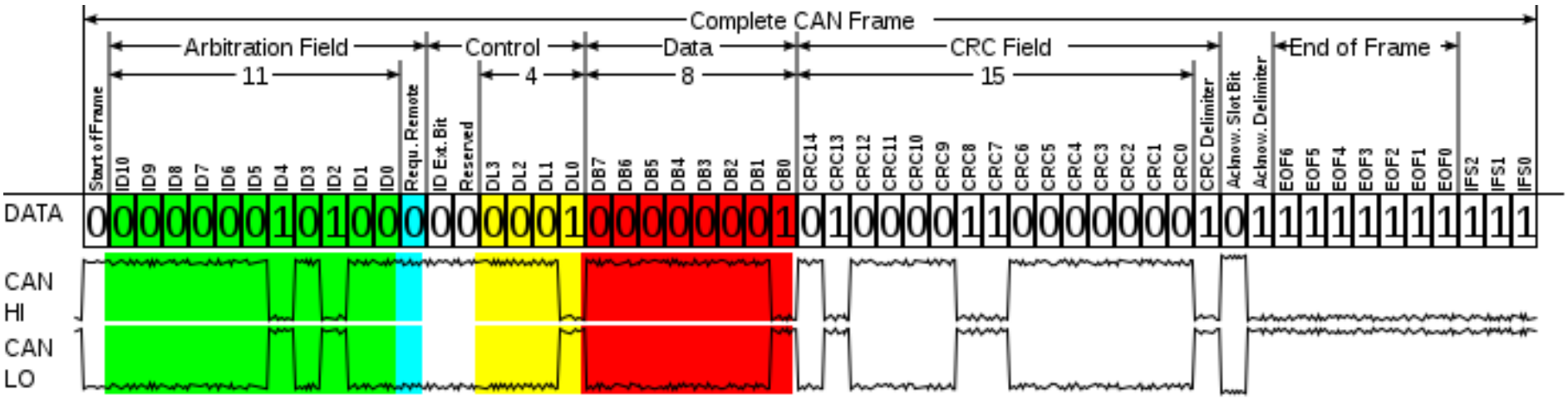
Resulting Data:



Controller Area Networks - What, Why, How?- Signaling Example



Controller Area Networks - What, Why, How?- Messages



Controller Area Networks - What, Why, How?- Theory to Practice

To implement the hardware of a CAN-based device, we need to provide the L1 and L2 features.

Controller Area Networks - What, Why, How?- Theory to Practice

To implement the hardware of a CAN-based device, we need to provide the L1 and L2 features.
Two Options:

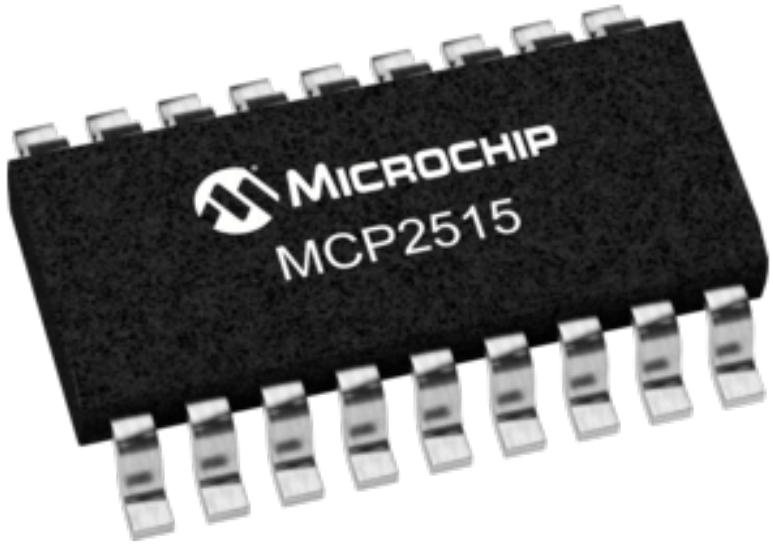
1. Use a standalone IC that covers both L1 and L2 requirements *OR*

Controller Area Networks - What, Why, How?- Theory to Practice

To implement the hardware of a CAN-based device, we need to provide the L1 and L2 features.
Two Options:

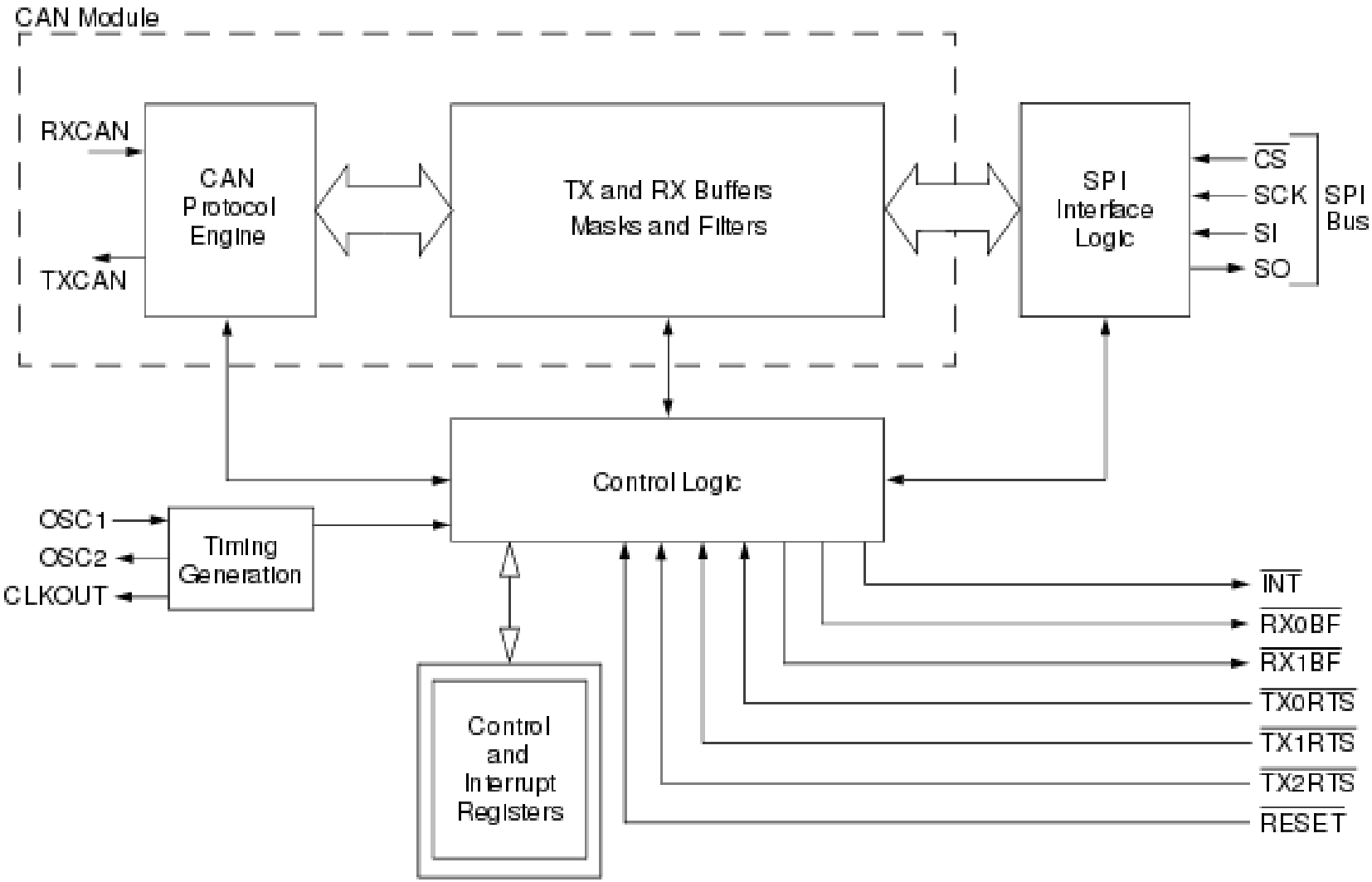
1. Use a standalone IC that covers both L1 and L2 requirements *OR*
2. Use an SoC/MCU with built in CAN module (L2) and possibly a transceiver IC (L1)

Controller Area Networks - What, Why, How?- Example - Standalone IC

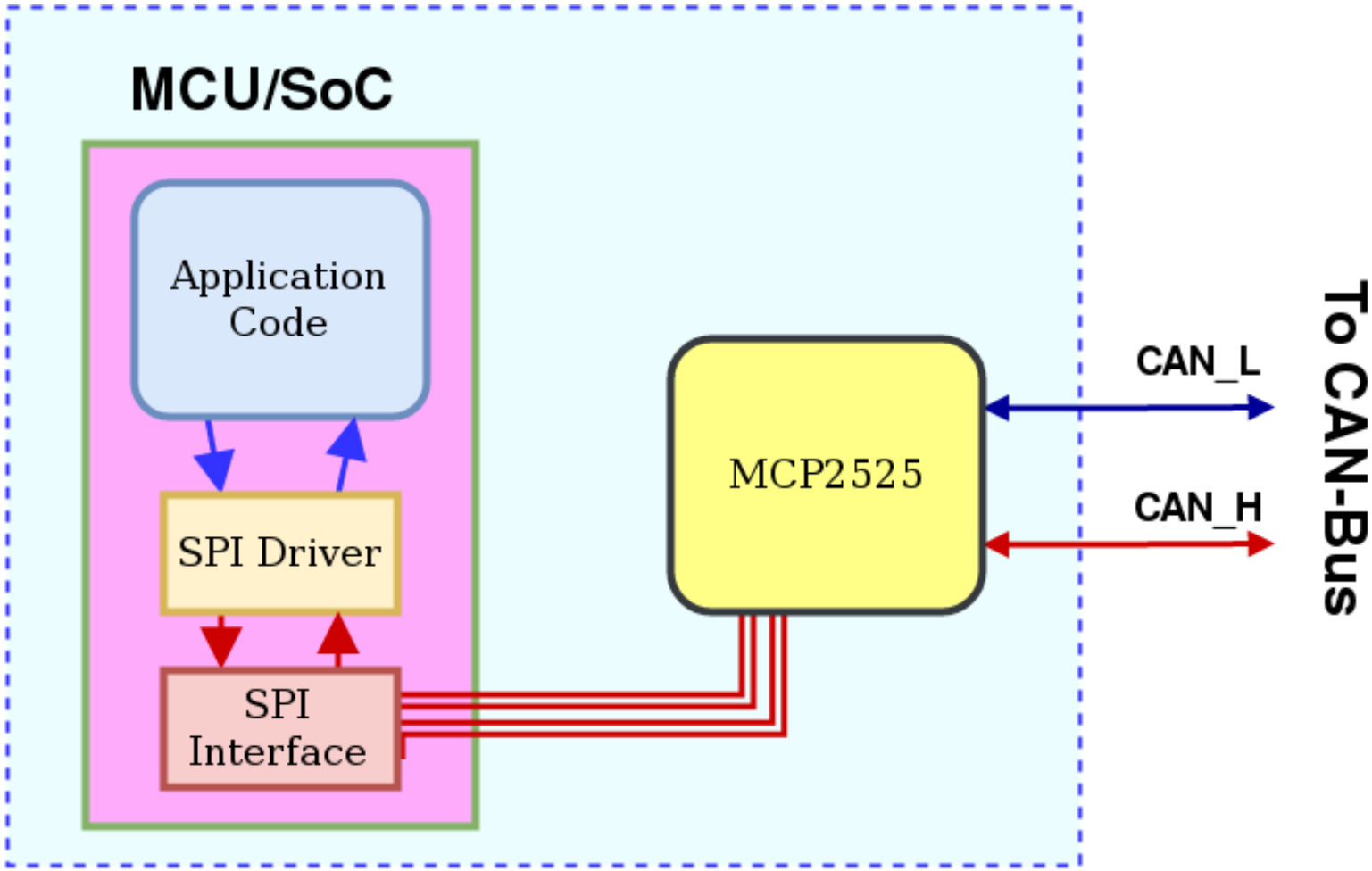


Microchip MCP2515

Controller Area Networks - What, Why, How?- Example - MCP2525



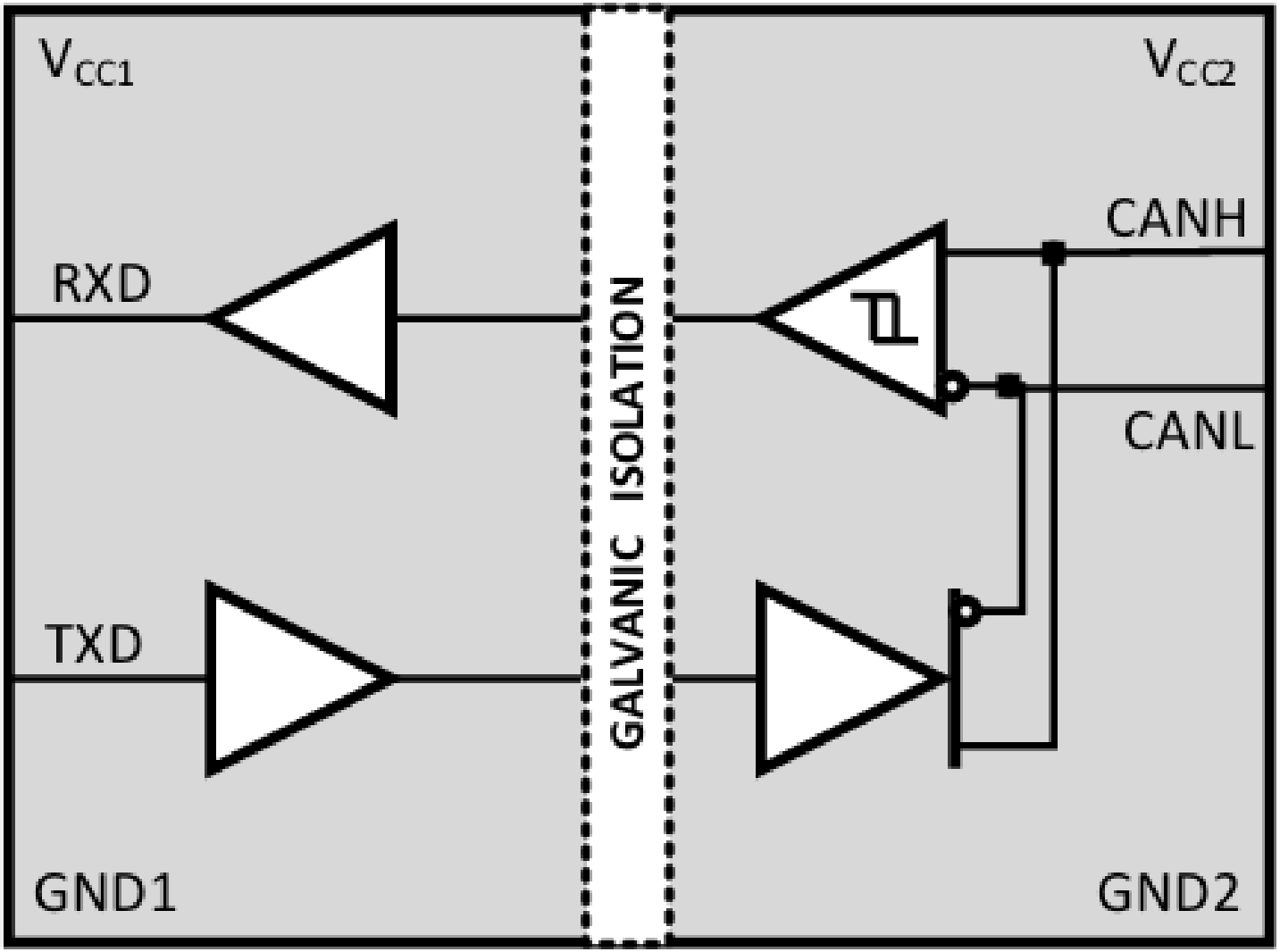
Controller Area Networks - What, Why, How?- Hardware Solution with MCP2525



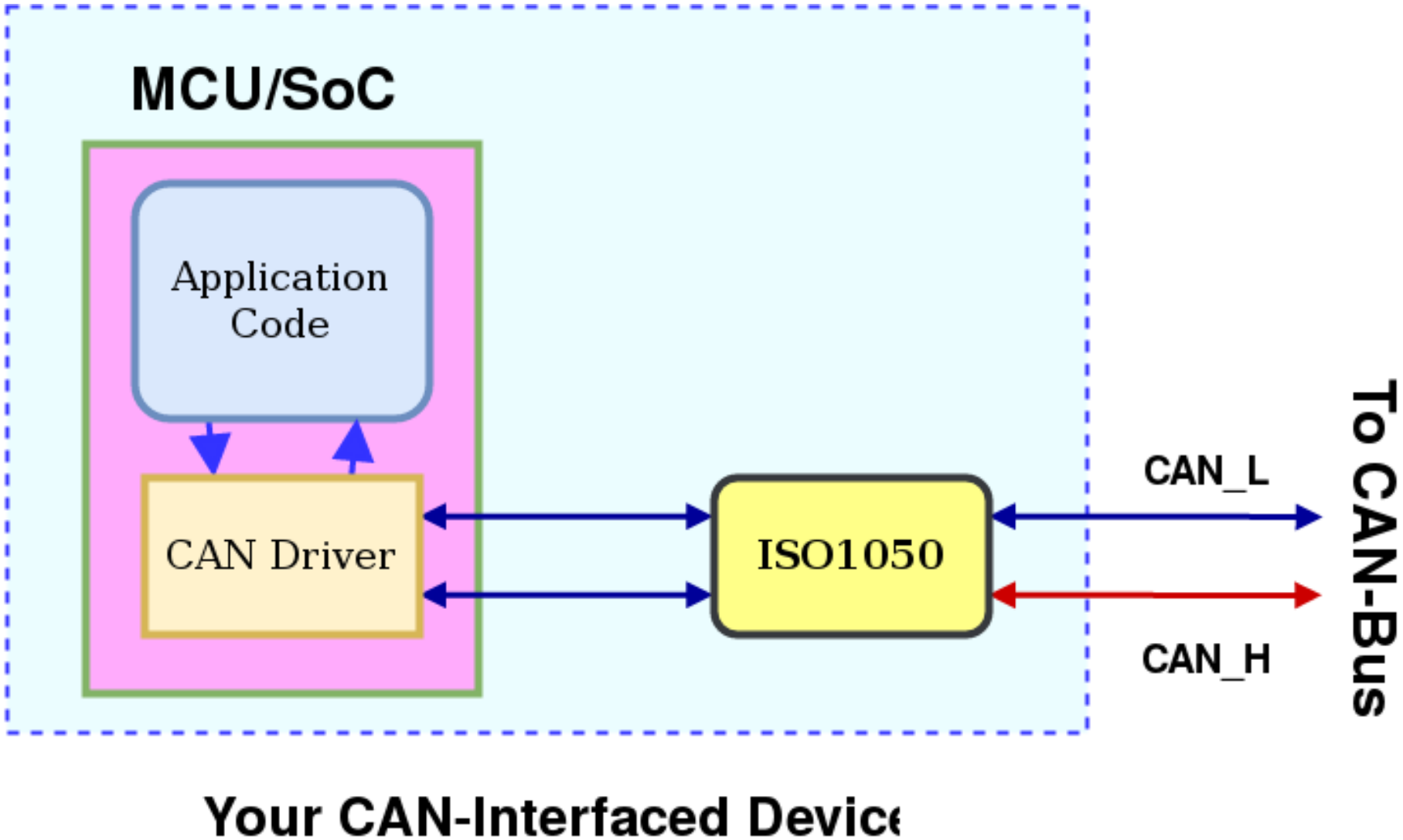
Controller Area Networks - What, Why, How?- Example - Use Integrated L2 CAN with L1 Transceiver

Texas Instruments ISO1050 Isolated CAN Transceiver

Controller Area Networks - What, Why, How?- Example - TI ISO1050



Controller Area Networks - What, Why, How?- Solution with ISO1050



Peeking at Protocols- Moving up the Stack

Q: What to do with all this reliable L1 and L2 infrastructure?

Peeking at Protocols- Moving up the Stack

Q: What to do with all this reliable L1 and L2 infrastructure?

A: Write standards of how our devices will use it!

Peeking at Protocols- Horses for Courses

Some examples of the protocols using CAN as a foundation:

J1939 **1990** Control in heavy machinery e.g. Trucks, Tractors. Created and governed by SAE. Baud rate of 250kbit/s, up to 30 nodes.

Peeking at Protocols- Horses for Courses

Some examples of the protocols using CAN as a foundation:

J1939 **1990** Control in heavy machinery e.g. Trucks, Tractors. Created and governed by SAE. Baud rate of 250kbit/s, up to 30 nodes.

DeviceNet **1992** Industrial control. Created by Allen Bradley, eventually governed by ODVA. Baud rate of 125/250/500kbits/s, up to 64 nodes.

Peeking at Protocols- Horses for Courses

Some examples of the protocols using CAN as a foundation:

J1939 **1990** Control in heavy machinery e.g. Trucks, Tractors. Created and governed by SAE. Baud rate of 250kbit/s, up to 30 nodes.

DeviceNet **1992** Industrial control. Created by Allen Bradley, eventually governed by ODVA. Baud rate of 125/250/500kbits/s, up to 64 nodes.

CANopen **1994** Industrial control. Created/governed by CiA. Up to 1Mbit/s, up to 127 nodes.

How come all these protocols?

How come all these protocols?

- Purpose

How come all these protocols?

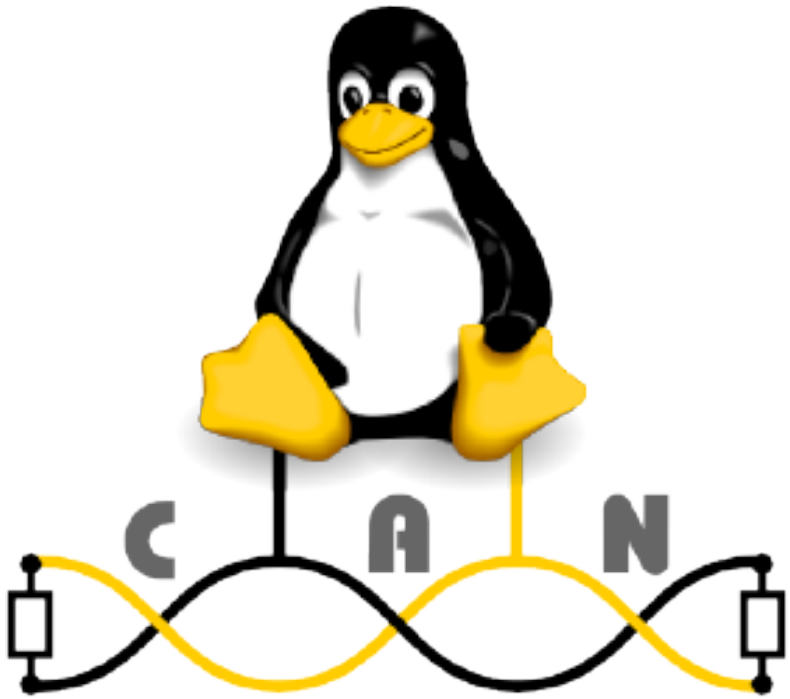
- Purpose
- Flexibility

How come all these protocols?

- Purpose
- Flexibility
- Implementation

Road Block: Before we can write application code (e.g. implement a protocol-compliant application), we need an API to talk to the CAN hardware/network.

Road Block: Before we can write application code (e.g. implement a protocol-compliant application), we need an API to talk to the CAN hardware/network.



SocketCAN:

- **2008** 2.6.25 Linux Kernel - Patched with CAN support

SocketCAN:

- **2008** 2.6.25 Linux Kernel - Patched with CAN support
- Known as Socket CAN (`linux\can\core.h`), contributed by Urs Thuermann, Oliver Hartkopp + More from Volkswagen Group Electronic Research

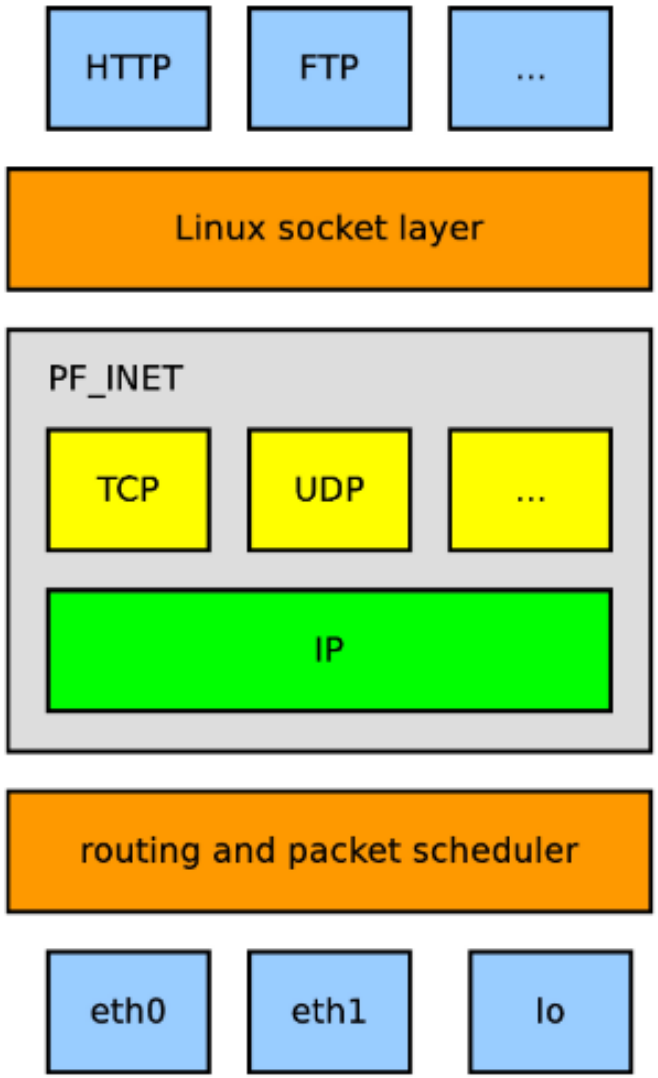
SocketCAN:

- **2008** 2.6.25 Linux Kernel - Patched with CAN support
- Known as Socket CAN (`linux\can\core.h`), contributed by Urs Thuermann, Oliver Hartkopp + More from Volkswagen Group Electronic Research
- Previous CAN drivers used 'Character Device Driver' - somewhat of a hack

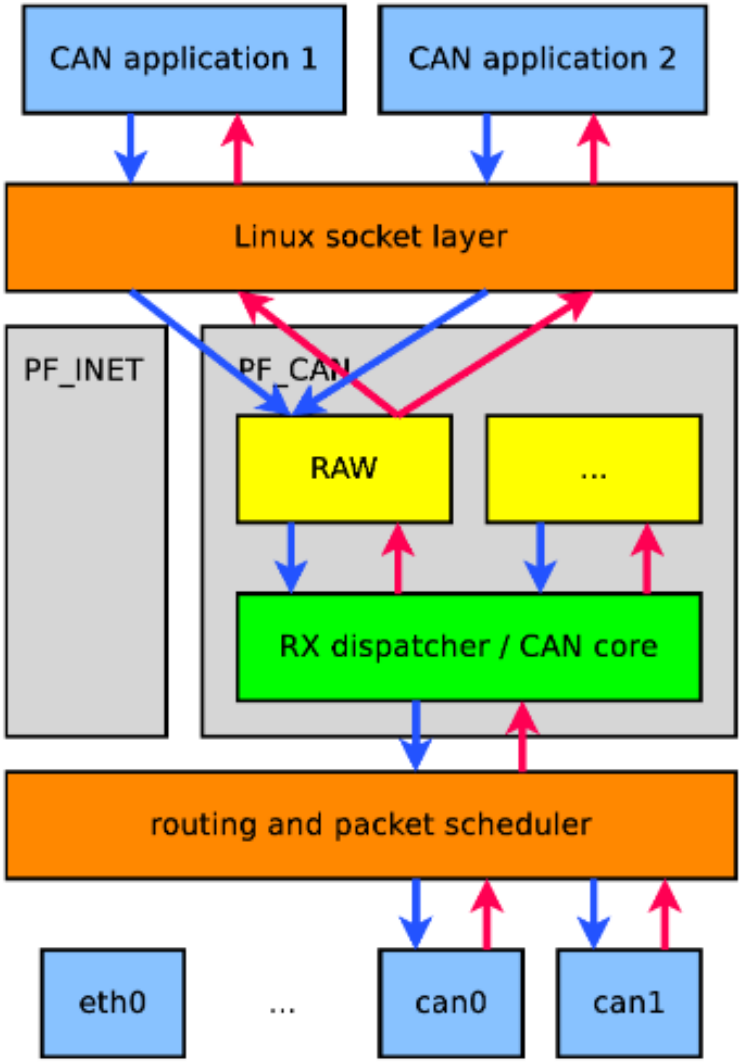
SocketCAN:

- **2008** 2.6.25 Linux Kernel - Patched with CAN support
- Known as Socket CAN (`linux\can\core.h`), contributed by Urs Thuermann, Oliver Hartkopp + More from Volkswagen Group Electronic Research
- Previous CAN drivers used 'Character Device Driver' - somewhat of a hack
- SocketCAN works within the Kernel's generic networking framework e.g. `#include sys\socket.h`, protocol family `PF_CAN`

CAN in the Kernel- Generic Linux Network Stack



CAN in the Kernel- PF_CAN



CANopen- CANopen Protocol

CANopen provides several communication objects, which enable device designers to implement desired network behavior into a device. With these communication objects, device designers can offer devices that can communicate process data, indicate device-internal error conditions or influence and control the network behavior.

- **CAN in Automation** description of CANopen

CANopen- CANopen Protocol

Key concepts:

1. The **Object Dictionary** (OD)

CANopen- CANopen Protocol

Key concepts:

1. The **Object Dictionary** (OD)
2. CAN Frames are called **messages**

CANopen- CANopen Protocol

Key concepts:

1. The **Object Dictionary** (OD)
2. CAN Frames are called **messages**
3. Messages are **Communication Objects**

CANopen- CANopen Protocol

Key concepts:

1. The **Object Dictionary** (OD)
2. CAN Frames are called **messages**
3. Messages are **Communication Objects**
4. 6 Message types are defined

CANopen- CANopen Protocol

Key concepts:

1. The **Object Dictionary** (OD)
2. CAN Frames are called **messages**
3. Messages are **Communication Objects**
4. 6 Message types are defined
5. Network Management (**NMT**) $\Rightarrow$ one node to rule them all [not necessarily!]

CANopen- CANopen Protocol

Key concepts:

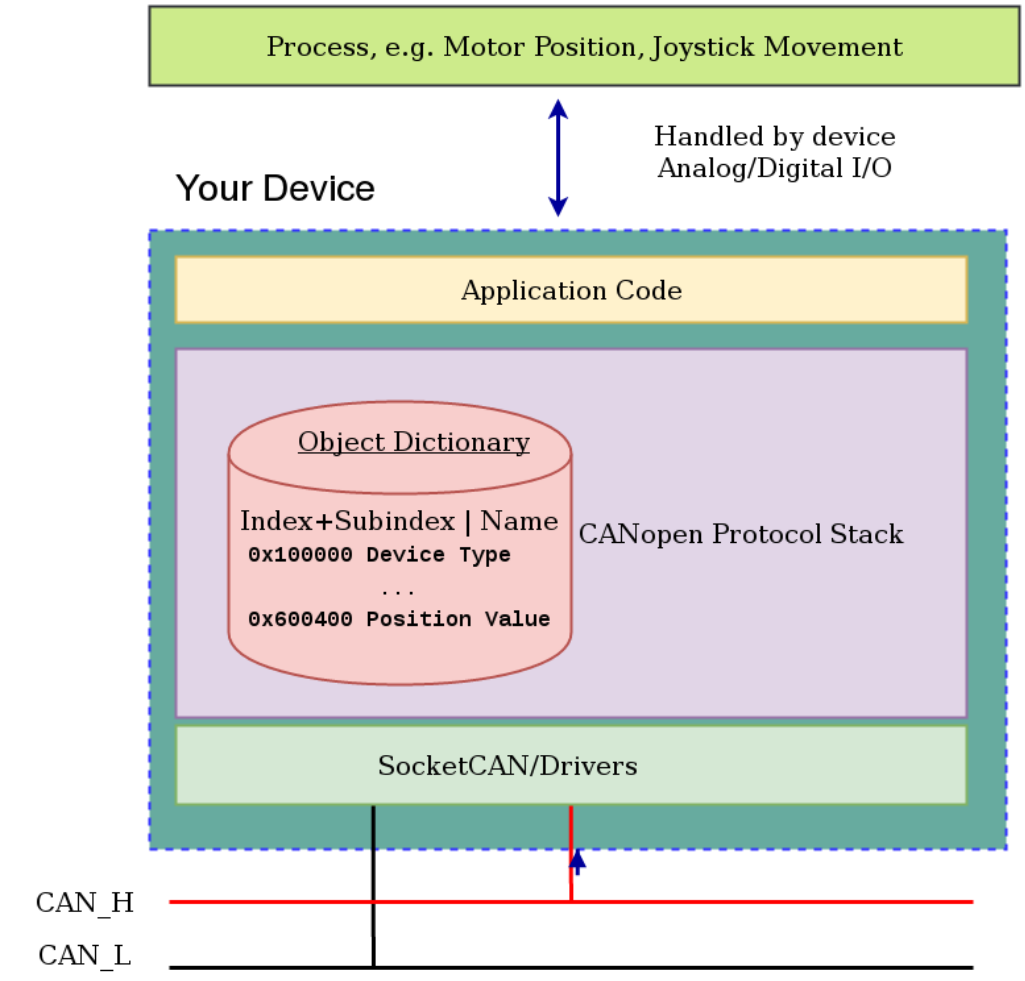
1. The **Object Dictionary** (OD)
2. CAN Frames are called **messages**
3. Messages are **Communication Objects**
4. 6 Message types are defined
5. Network Management (**NMT**) $\Rightarrow$ one node to rule them all [not necessarily!]
6. Network nodes adhere to an 'NMT State Machine', defined in CiA 301

CANopen- CANopen Protocol

Key concepts:

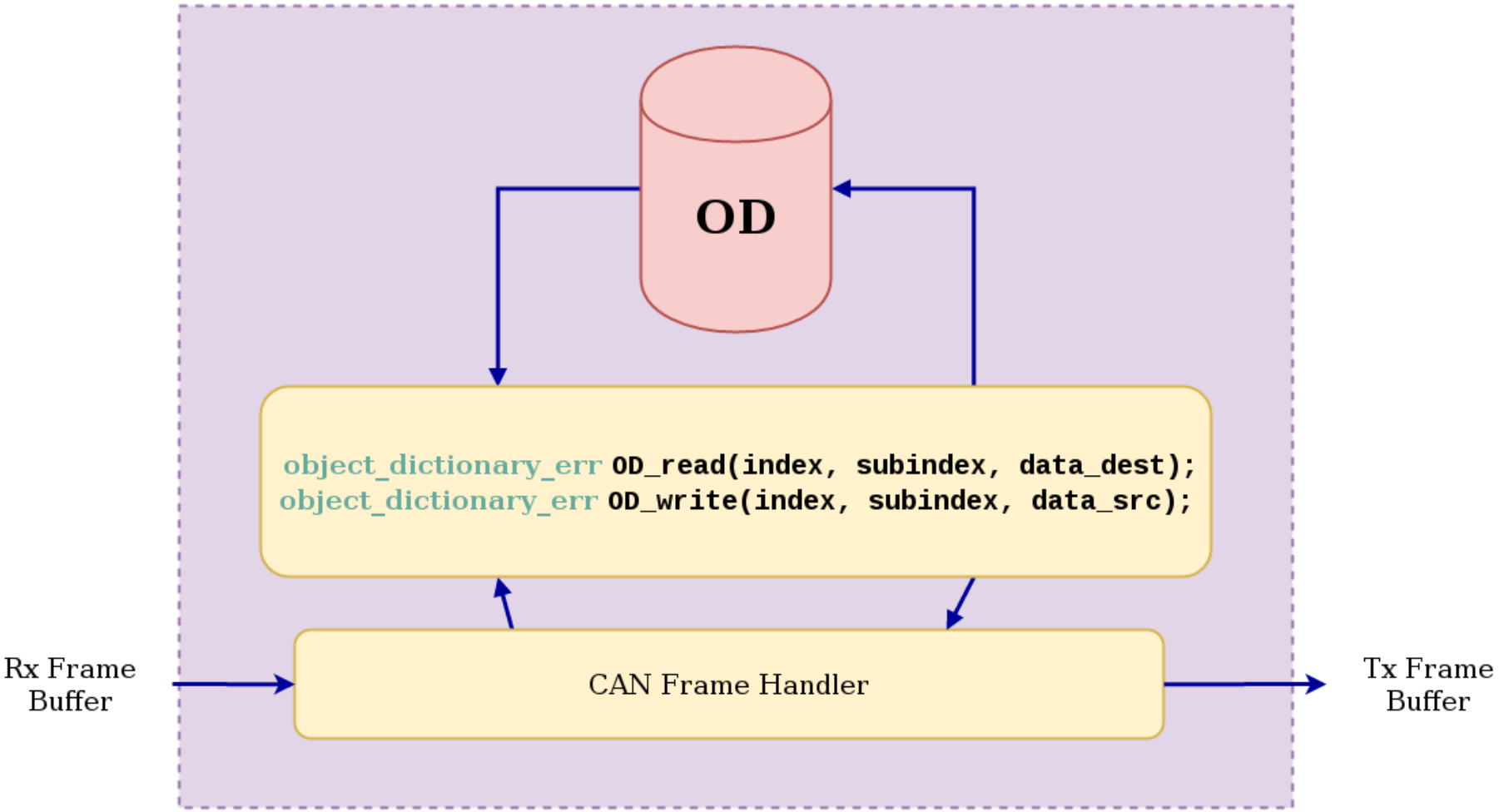
1. The **Object Dictionary** (OD)
2. CAN Frames are called **messages**
3. Messages are **Communication Objects**
4. 6 Message types are defined
5. Network Management (**NMT**) $\Rightarrow$ one node to rule them all [not necessarily!]
6. Network nodes adhere to an 'NMT State Machine', defined in CiA 301
7. Client (Network manager) + Server (network nodes/devices) functionality

CANopen- Object Dictionary



Text

CANopen- Object Dictionary



CANopen- Device State and Network Management

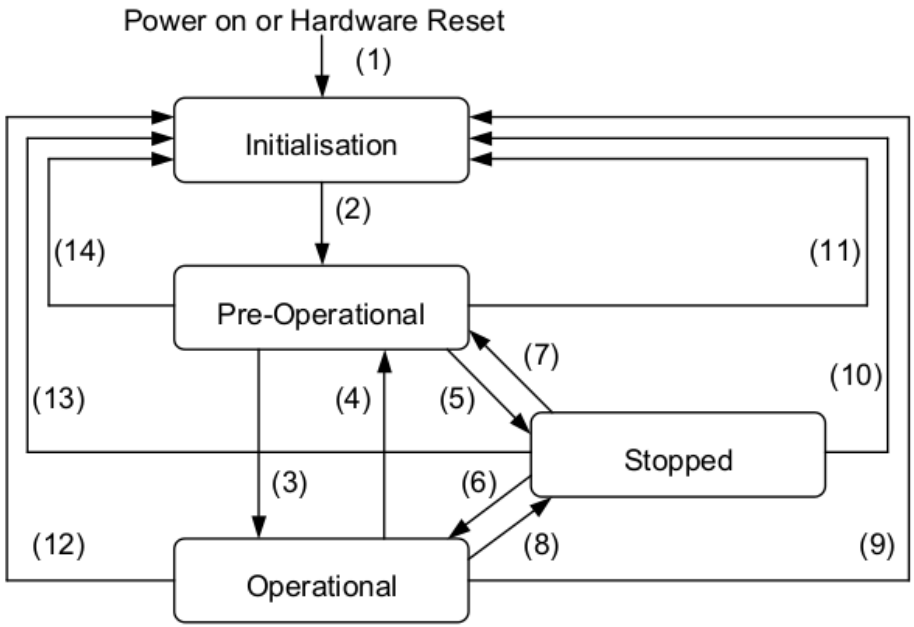


Table 31: Trigger for State Transition

(1)	At Power on the initialisation state is entered autonomously
(2)	Initialisation finished - enter PRE-OPERATIONAL automatically
(3),(6)	Start_Remote_Node indication
(4),(7)	Enter_PRE-OPERATIONAL_State indication
(5),(8)	Stop_Remote_Node indication
(9),(10),(11)	Reset_Node indication
(12),(13),(14)	Reset_Communication indication

CANopen- Communication Objects

Messages are Categorised into 6 types:

1. Boot-Up

CANopen- Communication Objects

Messages are Categorised into 6 types:

1. Boot-Up
2. Service Data Object (**SDO**)

CANopen- Communication Objects

Messages are Categorised into 6 types:

1. Boot-Up
2. Service Data Object (**SDO**)
3. Emergency (**EMCY**)

CANopen- Communication Objects

Messages are Categorised into 6 types:

1. Boot-Up
2. Service Data Object (**SDO**)
3. Emergency (**EMCY**)
4. SYNC/TIME

CANopen- Communication Objects

Messages are Categorised into 6 types:

1. Boot-Up
2. Service Data Object (**SDO**)
3. Emergency (**EMCY**)
4. SYNC/TIME
5. Heartbeat/Nodeguard

CANopen- Communication Objects

Messages are Categorised into 6 types:

1. Boot-Up
2. Service Data Object (**SDO**)
3. Emergency (**EMCY**)
4. SYNC/TIME
5. Heartbeat/Nodeguard
6. Process Data Object (**PDO**)

CANopen- Communication Objects

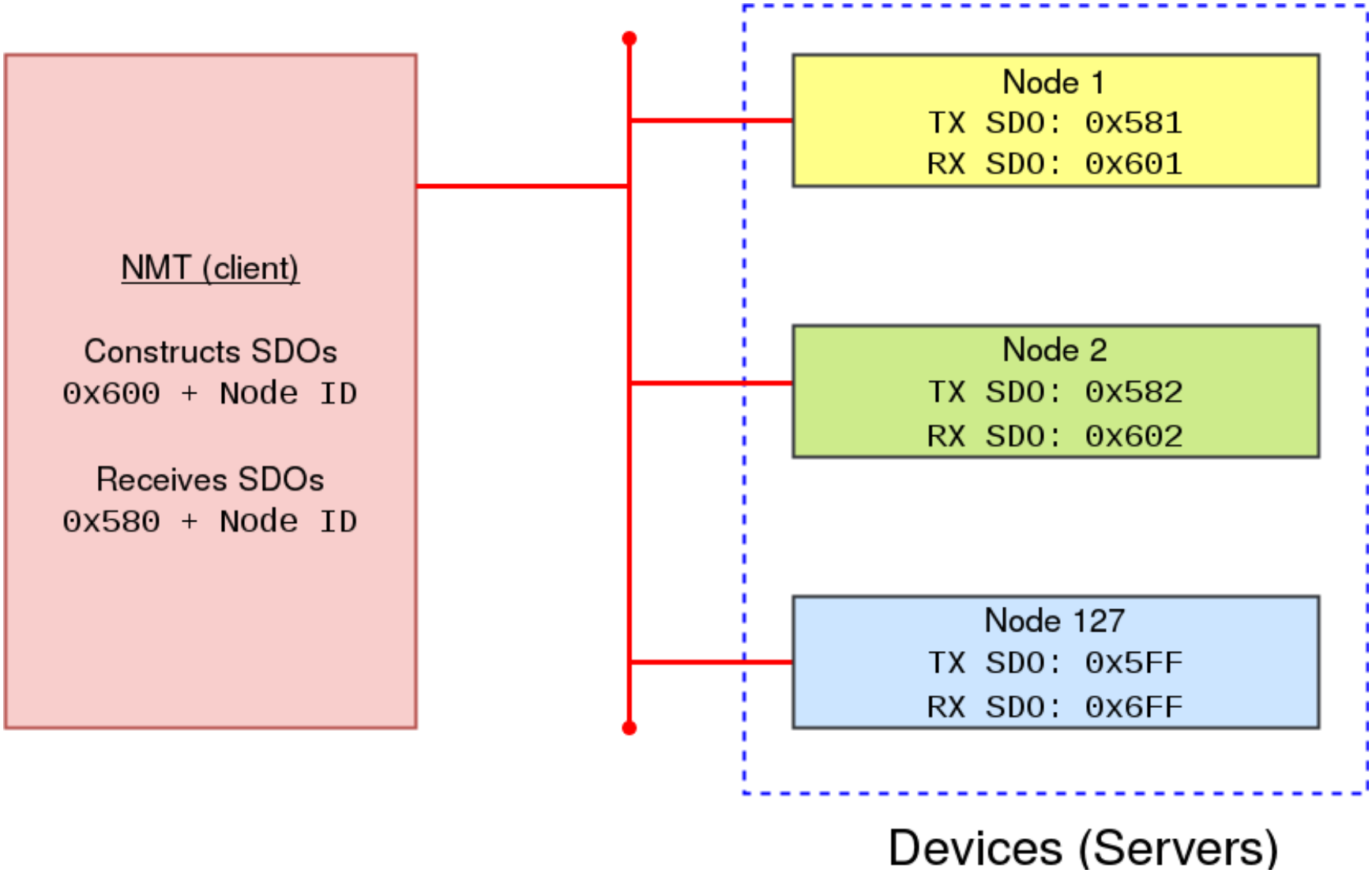
We can determine which type of message we are receiving by examining the CAN ID field (aka arbitration field) in the CAN Frame:

	Range		
Construction	**From**	**To**	**Object**
--	0x00	--	NMT Service – Sent by Network Manager
--	0x80	–	Sync
0x80 + Node ID	0x81	0xFF	Emergency Message
--	0x100	–	Time Stamp
0x180 + Node ID	0x181	0x1FF	PDO Transmit - 1
0x200 + Node ID	0x201	0x27F	PDO Receive - 1
0x280 + Node ID	0x281	0x2FF	PDO Transmit - 2
0x300 + Node ID	0x301	0x37F	PDO Transmit - 2
0x380 + Node ID	0x381	0x3FF	PDO Receive - 3
0x400 + Node ID	0x401	0x47F	PDO Transmit - 3
0x480 + Node ID	0x481	0x4FF	PDO Receive - 4
0x500 + Node ID	0x501	0x57F	PDO Transmit - 4
0x580 + Node ID	0x581	0x5FF	Transmit SDO
0x600 + Node ID	0x601	0x67F	Receiver SDO
0x700 + Node ID	0x701	0x77F	NMT Error Control

CANopen- Service Data Objects

SDOs work around a request/response model. Provide generic access method to Object Dictionary.

CANopen- Service Data Objects - Example



CANopen- Process Data Objects

PDOs are transmitted by devices according to their configuration:

- Event Driven

CANopen- Process Data Objects

PDOs are transmitted by devices according to their configuration:

- Event Driven
- Time Driven

CANopen- Process Data Objects

PDOs are transmitted by devices according to their configuration:

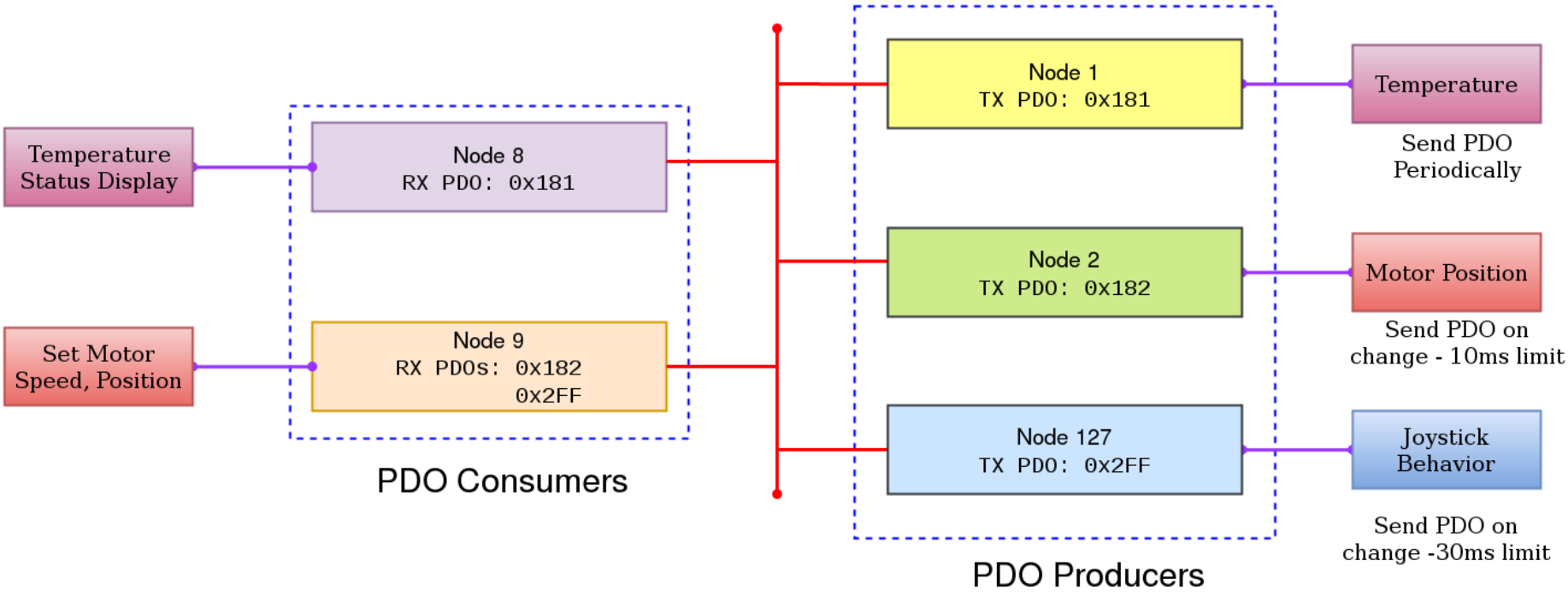
- Event Driven
- Time Driven
- Individual Polling

CANopen- Process Data Objects

PDOs are transmitted by devices according to their configuration:

- Event Driven
- Time Driven
- Individual Polling
- Synchronised, Group polling

CANopen- Process Data Objects - Example



There are numerous FOSS libraries for writing custom CANopen applications along with user-space tools for device development, network maintenance and monitoring.

FOSS CAN Tools- Some of the Options

CANopenNode `C` Library to implement a node. Includes object dictionary implementation, methods to send/receive all message types

FOSS CAN Tools- Some of the Options

CANopenNode `C` Library to implement a node. Includes object dictionary implementation, methods to send/receive all message types

can-open `python` Another library to implement a node. Object Dictionary can be supplied as text file. Includes `socketCAN` api.

FOSS CAN Tools- Some of the Options

CANopenNode `C` Library to implement a node. Includes object dictionary implementation, methods to send/receive all message types

can-open `python` Another library to implement a node. Object Dictionary can be supplied as text file. Includes `socketCAN` api.

can-utils User space tools to interact/observe a CAN network through the `socketCAN` layer. Example - `candump`, dumps CAN frames.

Demonstration- CANopen Joystick

Control Devices

Joysticks • Control Grips • Sensors • Encoders • Custom Electronics

Demonstration- Setup