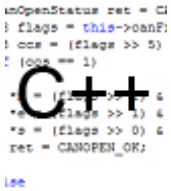
CANopen training

Datalink Engineering
CAN and CANopen experts

ulrik.hagstrom@datalink.se (+46-701-844551)



Goals

- **Basic understanding of CAN**
  - Understand basic such as identifier, datafield, prioritaztion using arbitration and error handling.
- **Understand the product sheet of a CANopen device.**
  - Understand the basic buzzwords. Learn what to expect and what to look for!
- **Please ask questions**
  - There are no stupid questions!
- **Take next step ?**
  - Suit your needs?

Agenda

- **'Basic' CAN (1)**

- History, application examples,CAN frame, priority, characteristics.

- **Market**

- Vendor companies, CANopen in practice.

- **Basic CANopen (1)**

- Hisory, keywords, communcation layer...,

- **CANopen design (1)**

- Tools, files.

- **Advanced CAN (2)**

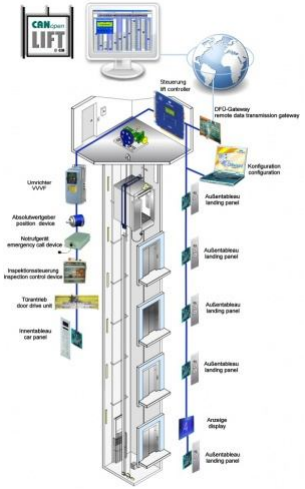
-Signal levels, calculations, error handling in detail, registers.

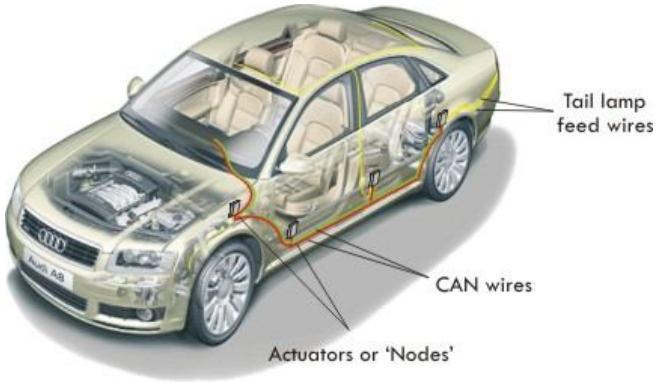
- **?**

Part 1: Basic CAN

Industrial automation (noise-critical env.)

Building automation (designed for control)





Automotive (low cost, reliable, volumes)

Small networks

CAN milestones

1986 - Robert Bosch GmbH requested by Mercedes.

1987 - The first CAN silicon fabricated in by Intel.

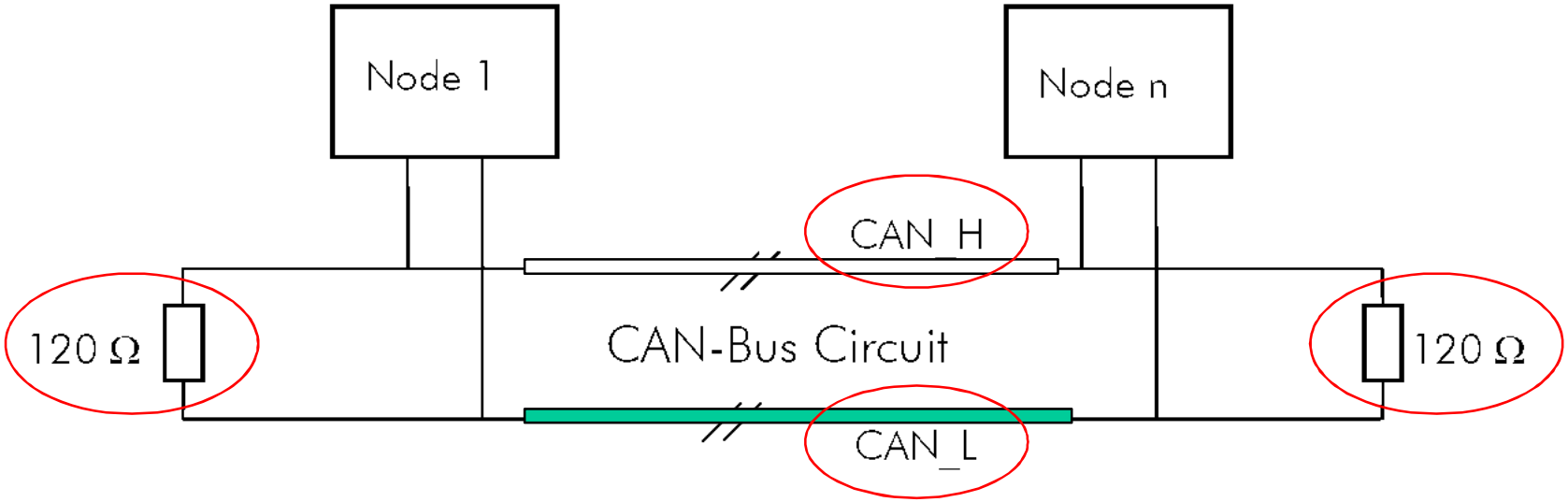
1988 - CAN available for everybody.

1991 - Mercedes S

1993 – **ISO 11898 specification.**

1998 – Volvo S80

Physical CAN network



Additional wires: 24 V + ground + sheild.

(most cases) Diffetential = Twinned CAN HI / CAN LO for best results!

Important numbers

- ~ Max 110 nodes on one *physical* network.
- 1 Mbps ⇔40 m.
  5 Kbps ⇔ 10 000 meters.
- 1 bit error each 0.7 s, 500 kbit/s, 8h / day, 365 days / year statistical average: 1 undetected error in 1000 years (24h: 333 years)

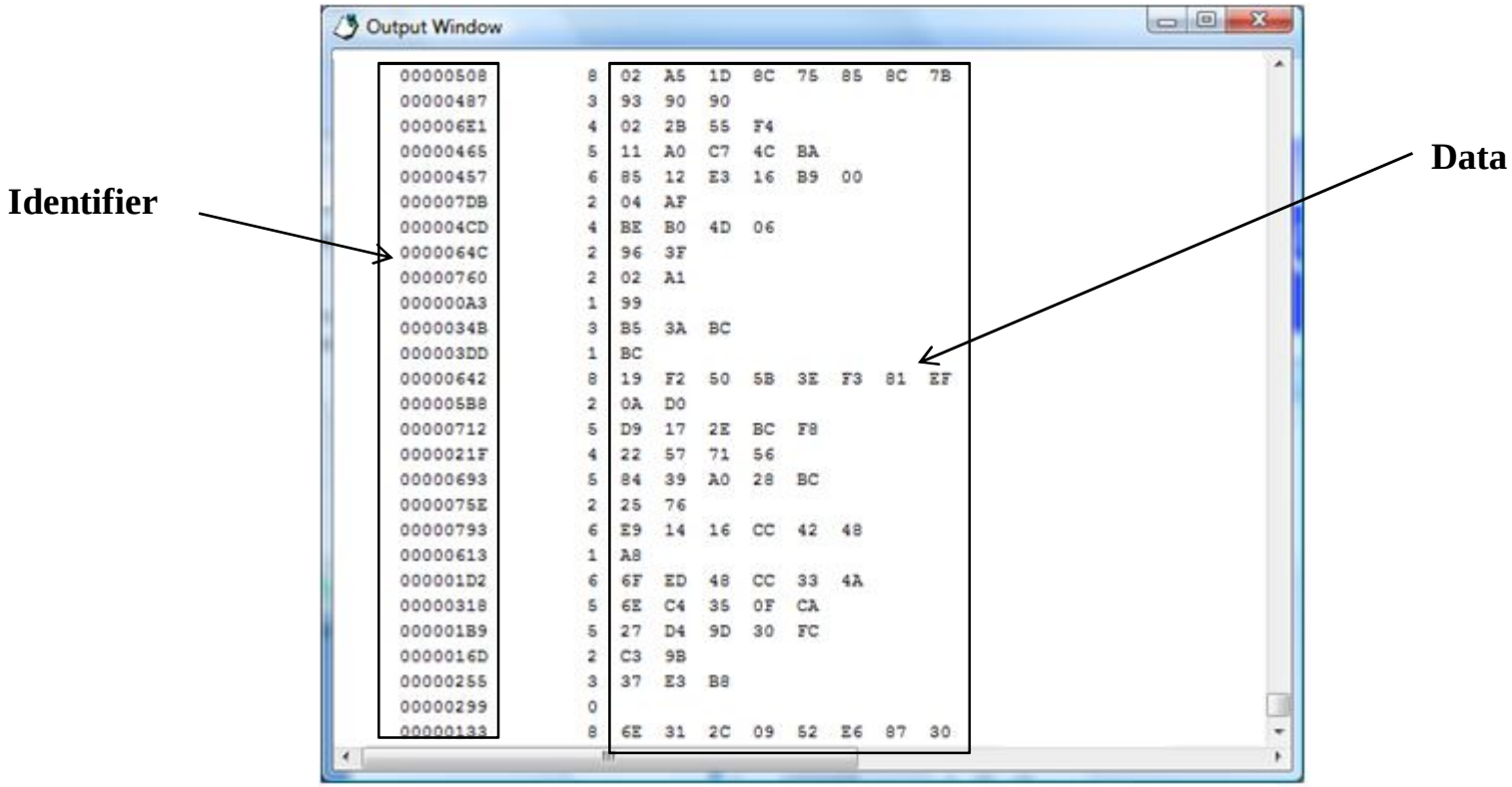
Transmit / Receive

- CAN message contains:
  - identifier (also implements priority)
  - Data (0-8 bytes)
  - CRC checksum and other error protection data fields.
  - Needs to be interpreted by higher layer protocol (HLP).
- Multi-master capability

  - Any CAN node may send a message if bus is idle using non destructive bus arbitration.
- Network wide data consistency

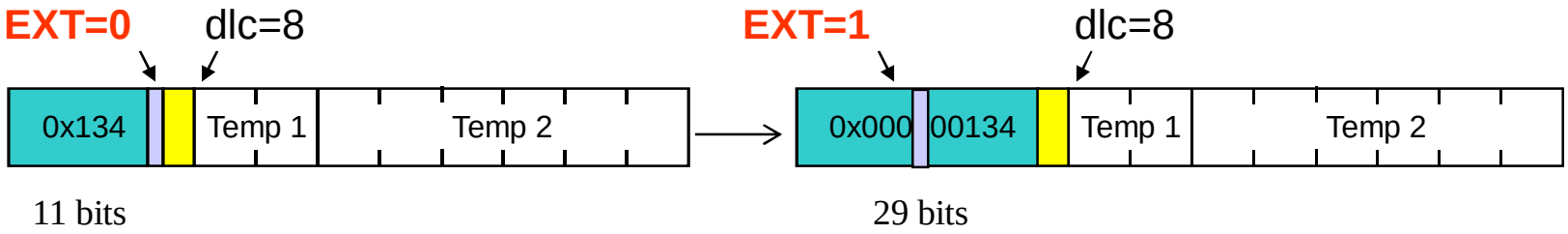
  - All receiving nodes decide if they like to accept the message *but no target receive guarantees to transmitter.*

Transmit / Receive

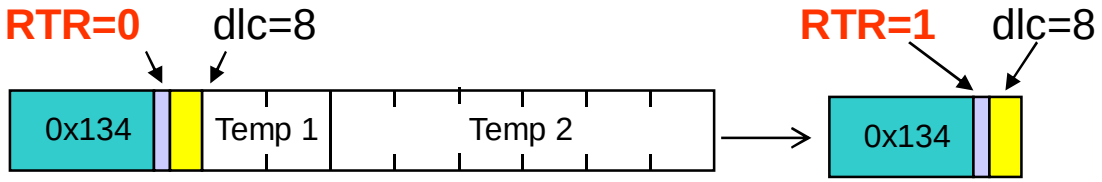


EXT/RTR control bits

EXT – Extended (turns the identifier into 29 bits instead of 11 bits)



RTR- Remote Transmit Request (does not attach any data)



Error handling

- Error checks are done **by all nodes** (transmitter do bit monitoring, receiver verifies CRC, form bit and more…)

- A CAN message is accepted by **all nodes or no node** (network wide consistency).

- Automatic retransmission on error. (Babbling idiot goes error “passive”)

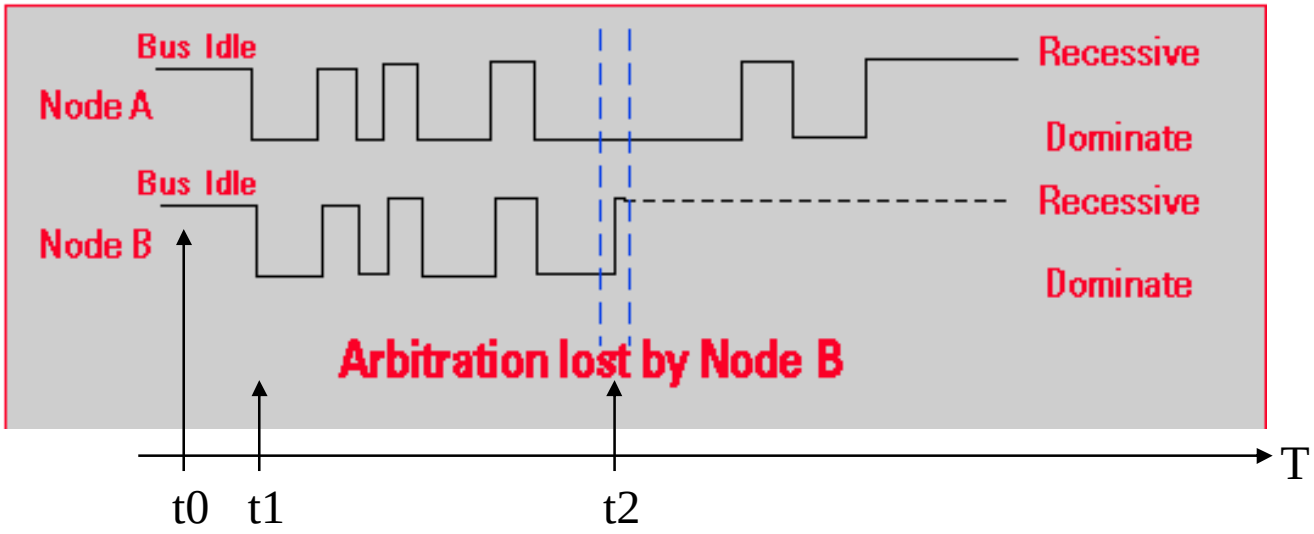
Collision resolution

- Collisions never happens because:

  CSMA/CR = Carrier Sense Multiple Access Collision Resolution

- Collision Resolution is handled using priorities (“non destructive arbitration”).

Non destructive arbitration



t0 Both “Node A” and “Node B” consider bus idle.

t1 Both nodes start transmit “SOF” (Start of frame)

t2 “Node A” transmits dominant bit and “Node B” recessive, and “Node A” wins the arbitration.

(arbitration field = identifier + rtr-bit)

Part 2: CANopen agenda

- Short history.
- Communication model, COBID addressing.
- Object Dictionary (OD).
- Service Data Object (SDO).
- Process Data Object (PDO).
- Error Control Protocol, Emergency Protocol (EMCY).
- Device Profiles (CANopen “plug and play”).
- Design flow, EDS, DCF, Configuration Management.
- Multiplex PDO, Time Stamp.

What is CANopen ?

- Machine automation mainly.
- Higher level protocol (HLP) based on **CAN**. Ethernet is on it's way! ETHERNET POWERLINK
- Developed by CiA (CAN in Automation, can-cia.org, non-profit, 500+ members).
- Open and vendor independent.

1994 CAL/Philips Medical adopted by CiA.
1995 CANopen DS-301 v.2.0
2006 CANopen DS-301 v.4.1

CiA CANopen





Features

- Device profiles give high abstraction for programmer (HW under development?)
- Easy access to all device parameters.
- (Inter-)device syncronization (node to node)
- Cyclic and event driven transfer.
- Sync read inputs, set outputs.

Vendor companies

Protocol stacks

IXXAT

EMBEDDED SYSTEMS ACADEMY

esd www.esd.eu

Embedded Office

port

vector

SYS TEC ELECTRONIC

МАРАФОН

eltas™ Your embedded solution partner.

Controller hardware

esitron

ifm

ASCON TECNOLOGIC

esd www.esd.eu

BECKHOFF

BERGHOF

Schneider Electric

sontheim Industrie Elektronik GmbH

SIW Sensor-Technik Wiedemann GmbH

WAGO INNOVATIVE CONNECTIONS

EPEC

FESTO

SYSTEM Electronics

Telemecanique a brand of Schneider Electric

SYS TEC ELECTRONIC

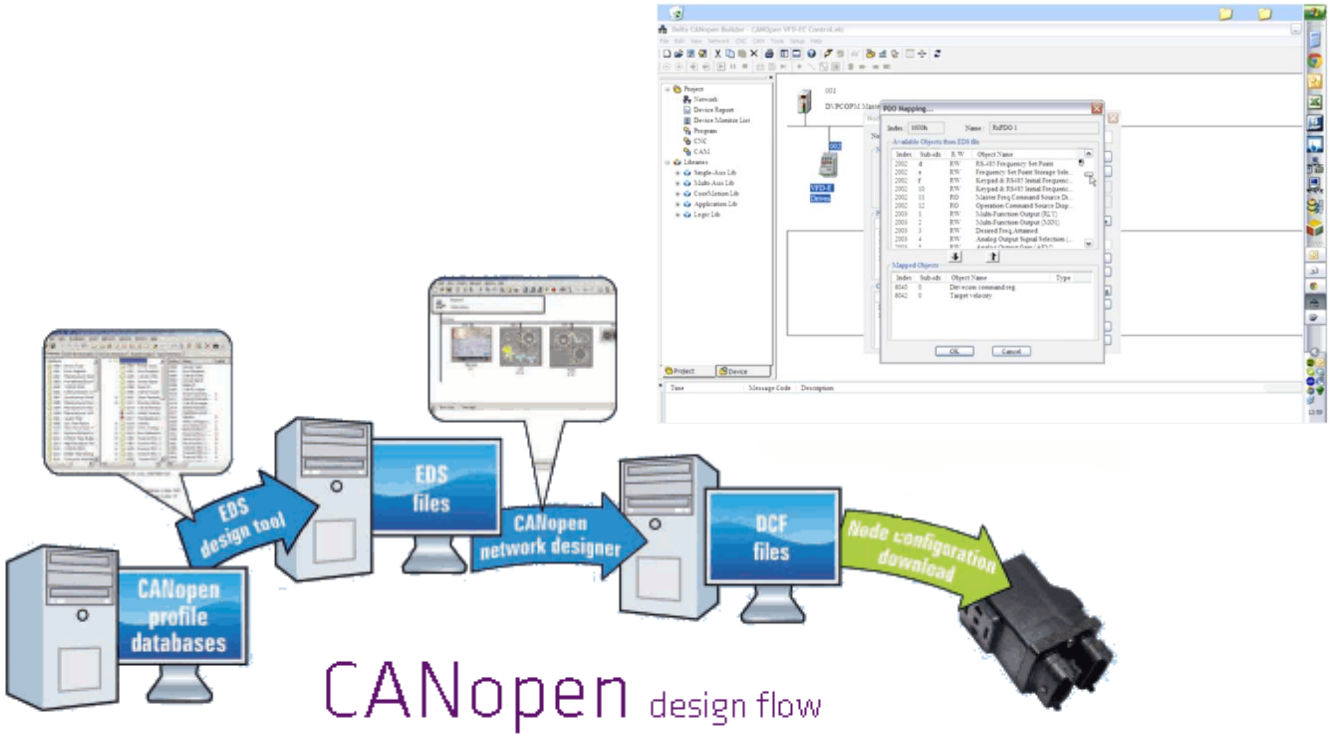
Device Profile	Products
Generic I/O	76
Servo drives	42
Encoders	27

CANopen Product Guide

Home | Profiles | Companies | Categories | Products

http://www.cia-productguides.org/

CANopen in practice



http://vimeo.com/46874684

Advantages using CANopen

→ CANopen unburdens dealing with *CAN-specific details*.

→ Standardized highly *flexible* configuration.

→ Off-the-shelf devices, tools, and protocol stacks at reasonable prices.

→ CANopen device profiles enable *"plug and play"*.

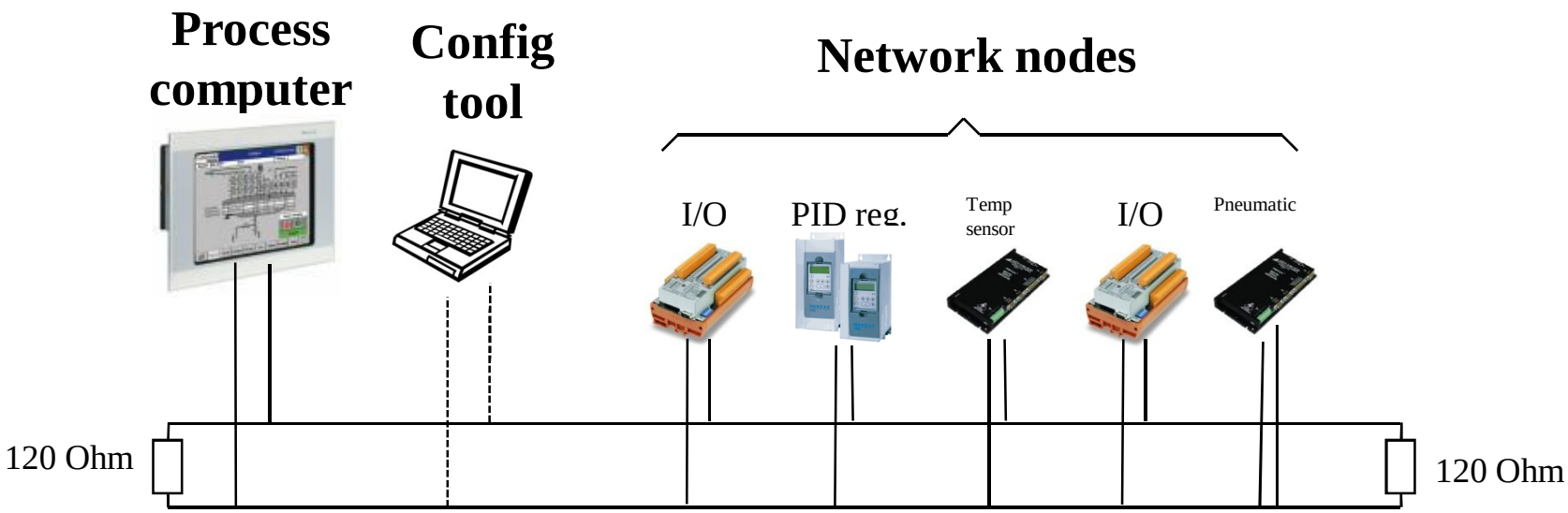
Device Profiles

CiA 401	Generic I/O Modules
CiA 402	Drives and Motion Control
CiA 404	Measuring devices and Closed Loop Controllers
CiA 405	IEC 61131-3 Programmable Devices
CiA 406	Rotating and Linear Encoders
CiA 408	Hydraulic Drives and Proportional Valves
CiA 410	Inclinometers
CiA 412	Medical Devices
CiA 413	Truck Gateways
CiA 414	Yarn Feeding Units (Weaving Machines)
CiA 415	Road Construction Machinery
CiA 416	Building Door Control
CiA 417	Lift Control Systems
CiA 418	Battery Modules
CiA 419	Battery Chargers
CiA 420	Extruder Downstream Devices
CiA 422	Municipal Vehicles – CleANopen
CiA 423	Railway Diesel Control Systems
CiA 424	Rail Vehicle Door Control Systems
CiA 425	Medical Diagnostic Add-on Modules
CiA 445	RFID Devices

www.can-cia.org

- **DS-301 Communication profile, “basic parts of CANopen”.**
- DS-302 Framework for programmable CANopen devices (boot up, configuration manager).
- DS-306 EDS (Electronic data sheet, template), DCF (Device configuration file, values)
- DS-4xx Device profiles (“plug and play I/O, servo etc, HMI”)

Example CANopen network

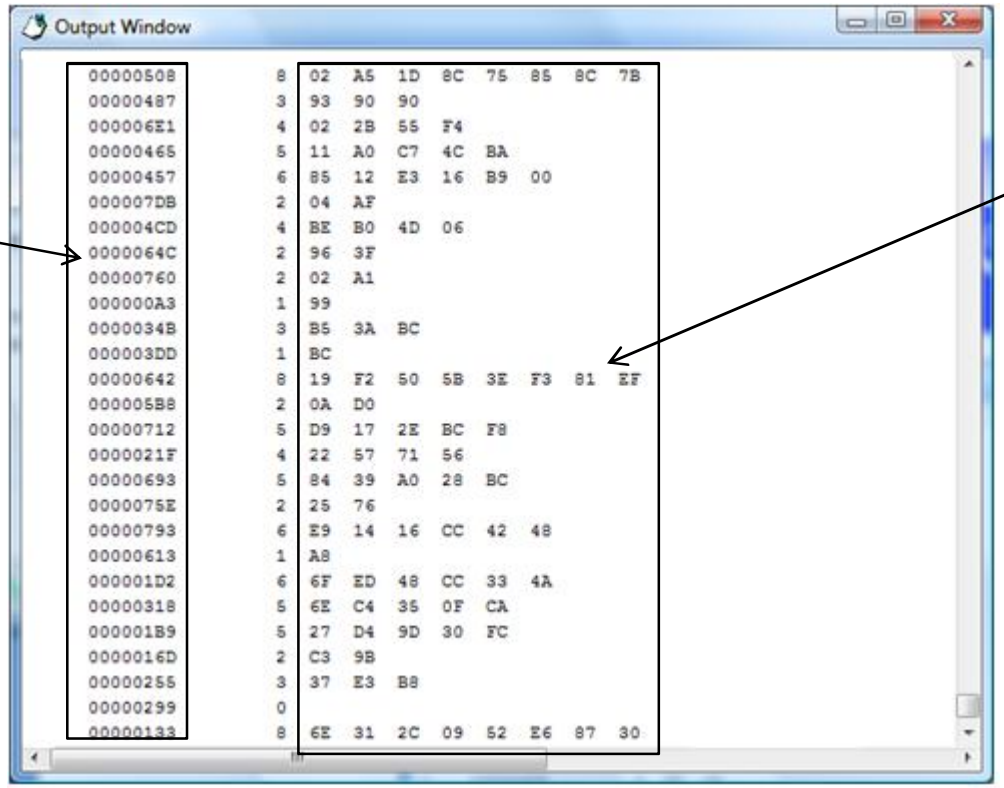


2 wire shielded twisted pair cable (alt. 4 wire with 24 Volt power over the bus).

- All nodes have a **node id** value (1-127, 0 is broadcast adress)
- All nodes are addressed via **default connection set** (base + node id).

COBID

COBID
COB, but we call it COBID for now!
(COBID is 32 bits, this is at most 29bits, the upper 3 bits are control bits).



Data

Node functionality

Process computer

Node ID

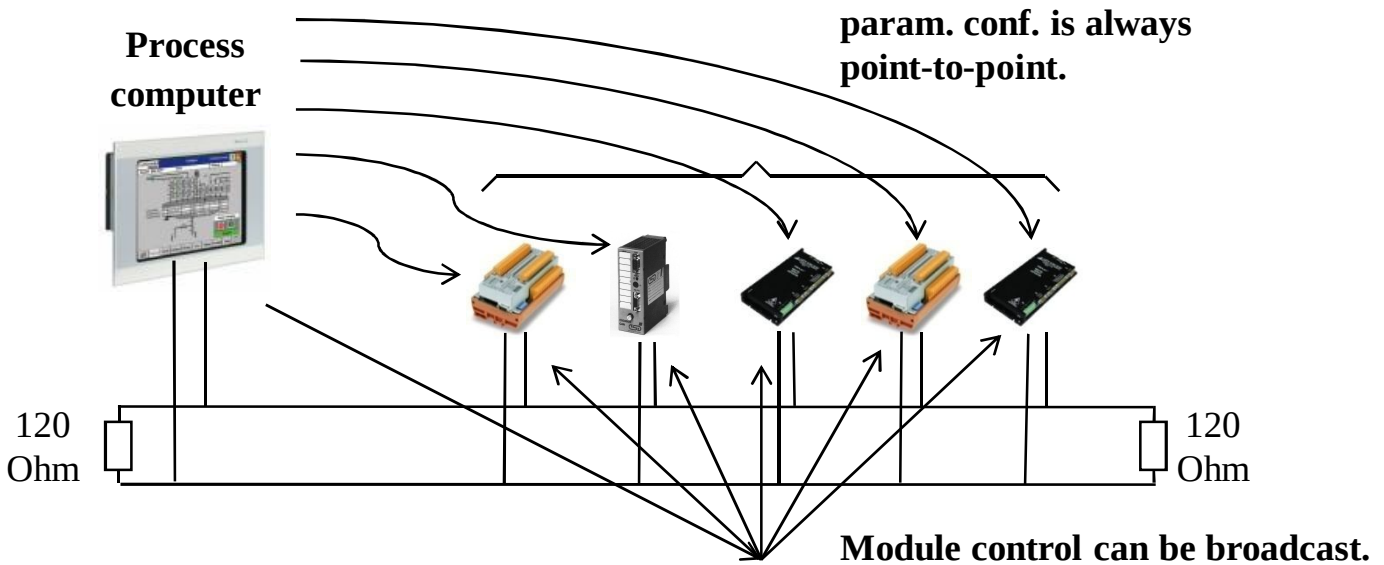
Network node

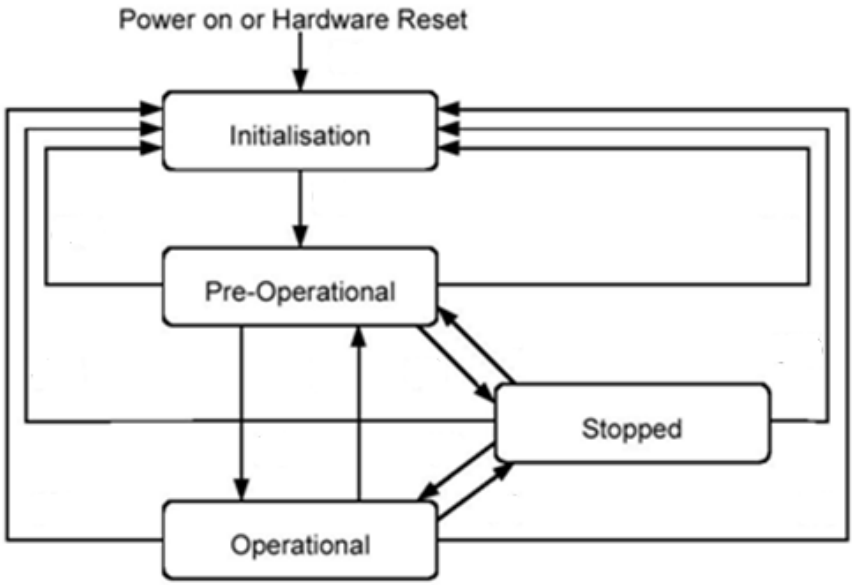
Node ID

Configuration tool

Network initialization process

1. All nodes initializes and enter pre-operational state after power on.
2. Process compter configures the nodes (paramter configuration).
3. Process compter sets nodes in operational state (module control).

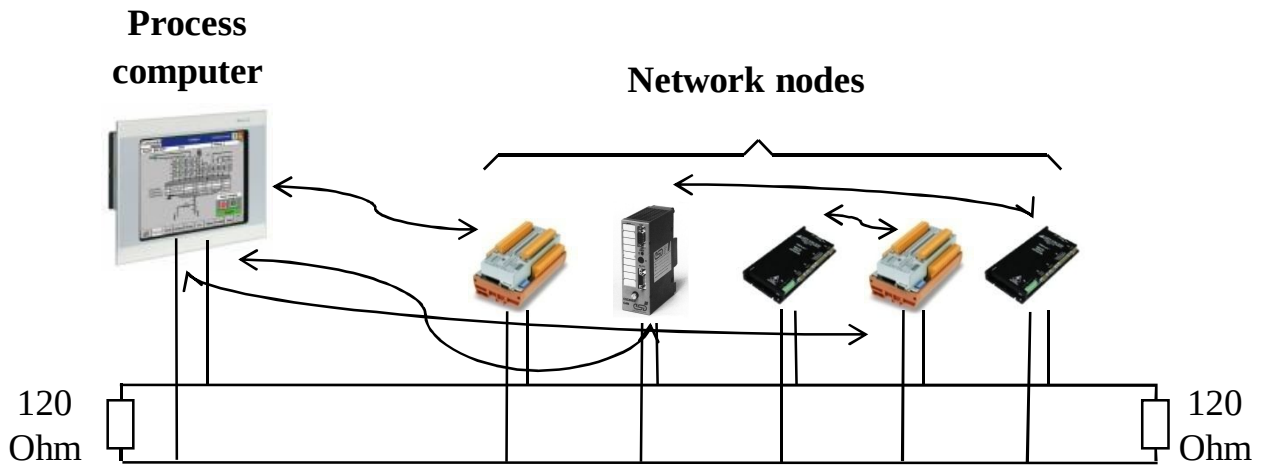




State	Description
Initialization	**Initialization** at power on the CANopen slave (with minimum boot-up capability) performs an initialization sequence and enters automatically into the **Pre-Operational state**.
Pre-operational	Parameter configuration. Node guarding and respond to node-guarding protocol.
Stopped	Node is disabled, no communication except node guard response.
Operational	Have it's process data channels active. Have parameter conf. channels active. Send emergency messages on an error event .

Running network

1. Process computer monitor operational state of nodes.
2. Process data can be sent from process computer to network nodes or between network nodes directly.



Node functionality

Process computer

Node ID

Object Dictionary

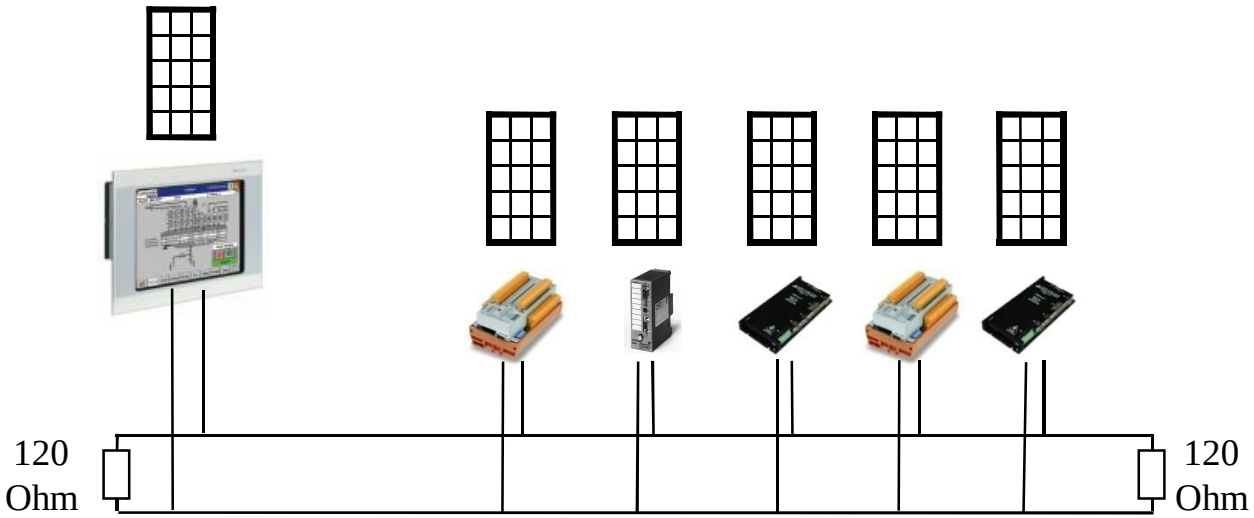
Network node

Node ID

Object Dictionary

Configuration tool

Object Dictionary



OD = ”parameter configuration area”

Object Dictionary

• Ordered grouping of objects
(object index + subindex)

• OD describes the device and network behaviour.

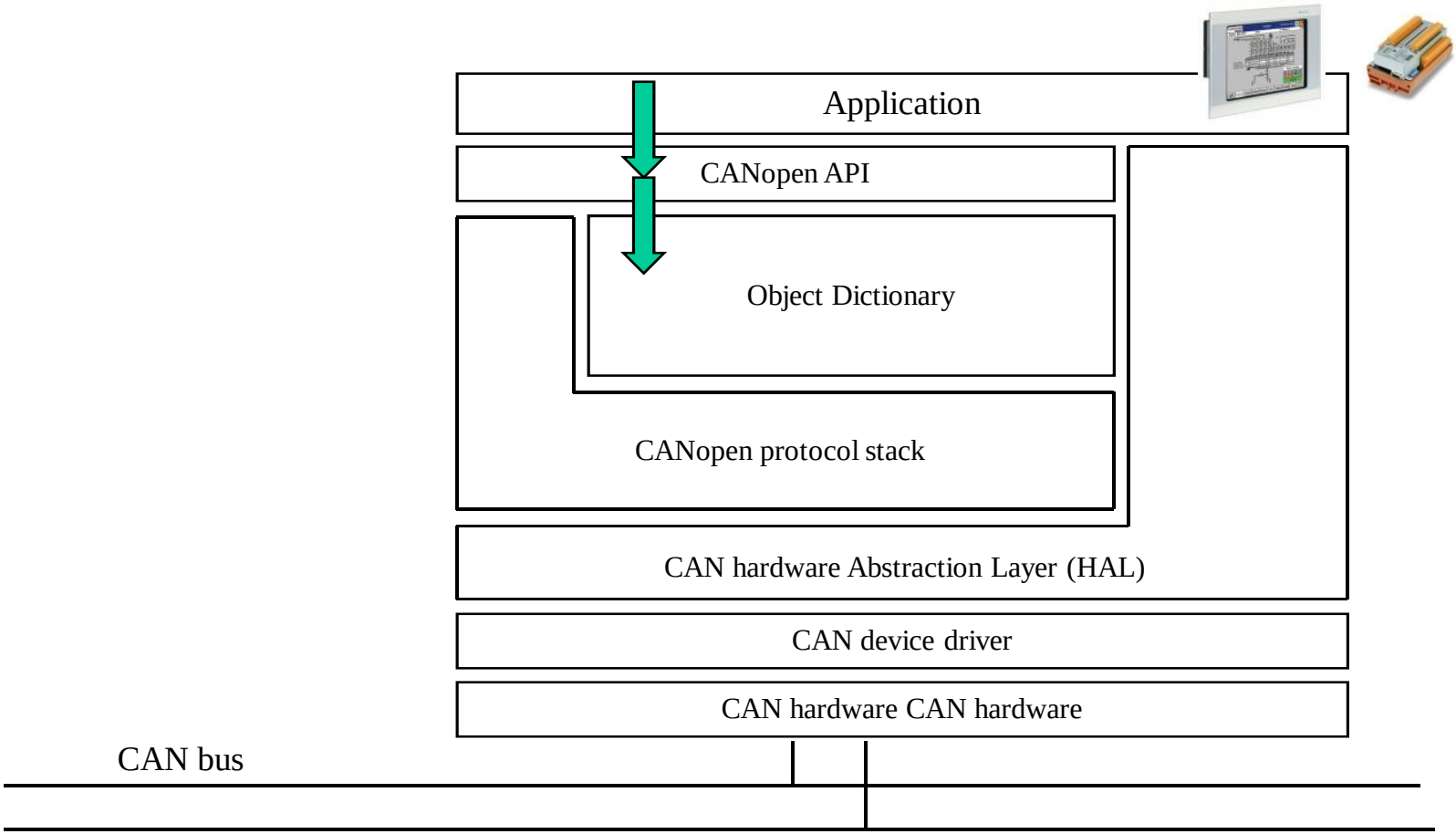
• Some of the OD entries are
mandatory (*quite few*, allows lean implementations)

• EDS-file describes your nodes OD (306)

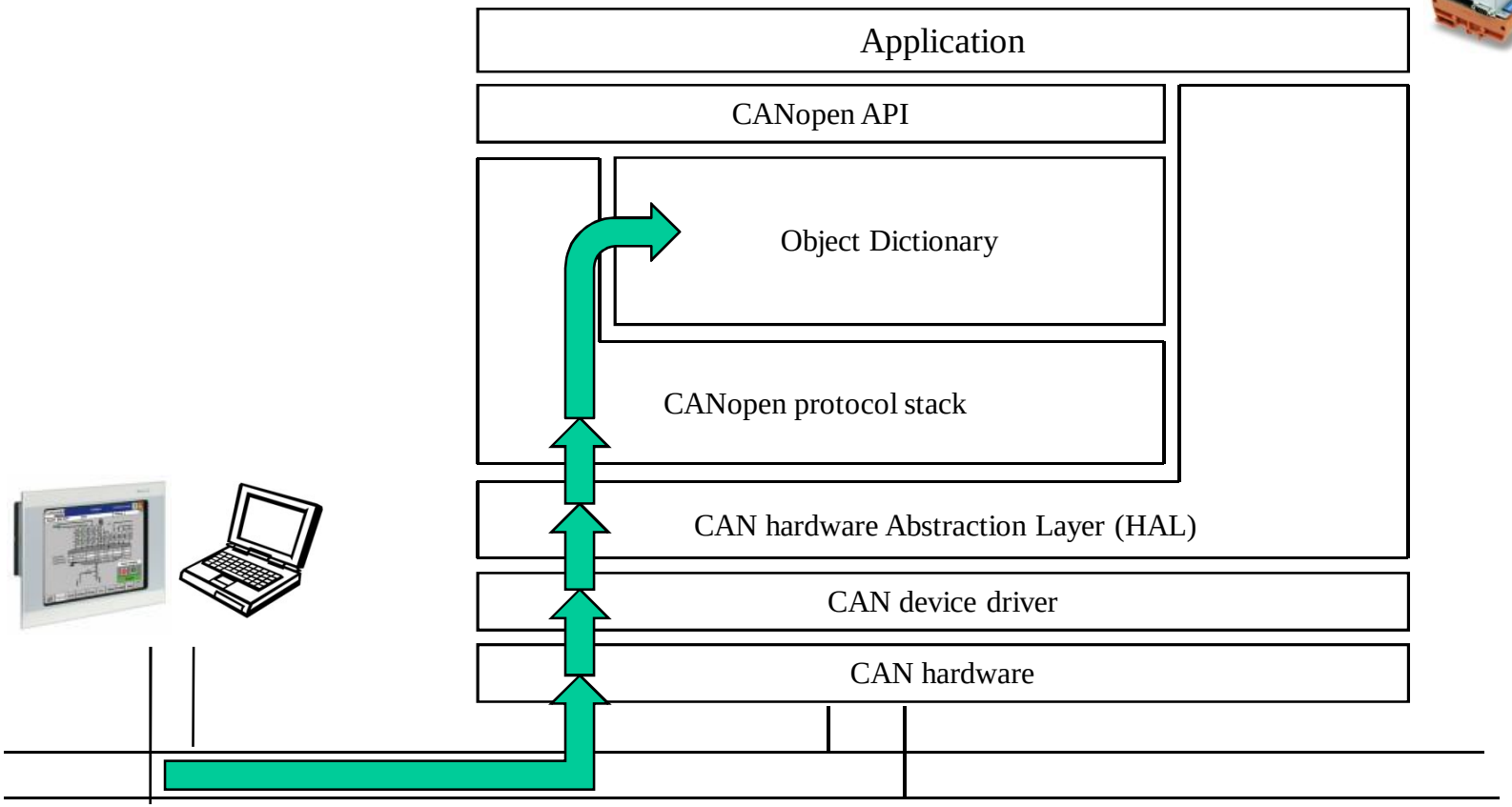
Object Dictionary

Object Index	Sub Index	Data Type	Bit contents	Description
0x1000	0	UINT32		Device Type.
		Byte 1		Device Profile.
		Byte 2		
		Byte 3		Additional info.
		Byte 4		
0x1001		UINT8		Error Register (Read error type on node).
0x1002		UINT32		Manufacturer status register.
0x1003		UINT32		Predefined error field.
0x1004				Reserved (for number of PDOs?)
0x1005		UINT32		COBID SYNC
0x1006		UINT32		Communication cycle period.
...				
0x1008		STRING		Device name.
0x1009		STRING		Hardware version.
0x100A		STRING		Software version.
0x100B				Reserved (for setting new node ID?)
0x100C		UINT16		Guard time
0x100D		UINT8		Lifetime factor

Access to local Object Dictionary



Parameter configuration via CAN-bus?



Node functionality

Process computer

Node ID

SDO client

Object Dictionary

SDO server

Network node

SDO server

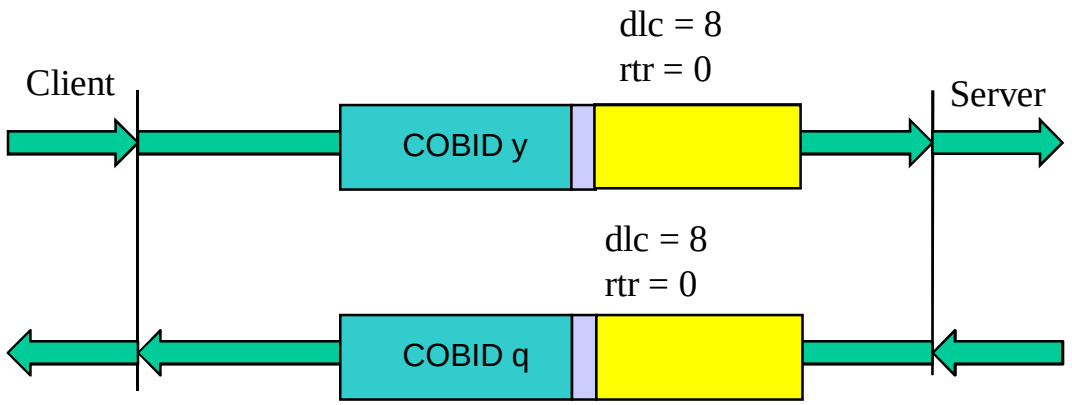
SDO client

Configuration tool

Service Data Object (SDO)

- Mainly used for parameter configuration of remote node.

- Transfer protocol for parameters (also FW upgrade).



Service Data Object (SDO)

SDO Client (0 – 128)

- Provides access (R/W) to OD of a remote node.
- Resides in nodes that needs to access other nodes OD, usually only the network Master node (configuration manager).

SDO Server (1 – 128)

- Responds to SDO client read/write.
- Mandatory to implement at least one.

SDO transfer protocols

Expedited transfer

1 to 4 bytes

Segmented transfer (“normal transfer”)

More than 4 bytes (64 bits value and string for example)

Block transfer

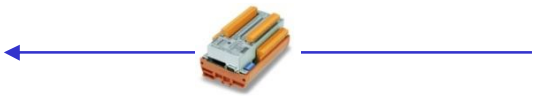
More bandwidth efficient than segmented transfers but “do the same thing”.

The SDO transfer protocols

(**Expedited**-, Segmented- or Block-transfer)

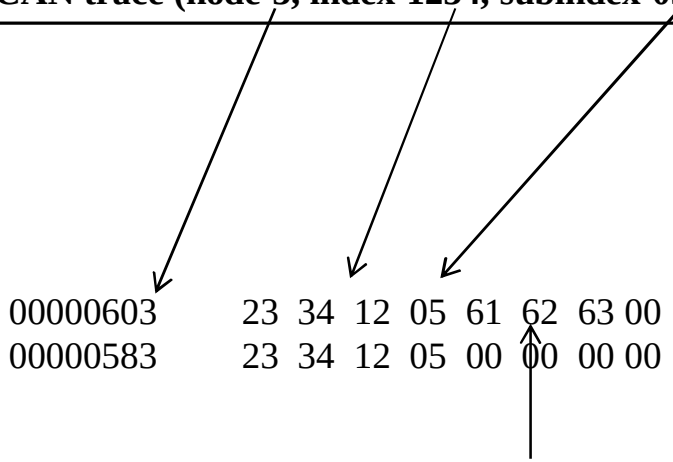
Expedited Download ("write node OD")

Client: *Initiate SDO download request* (ex. 3 bytes) to index 0x1234 and subindex 0x5 (e=1 (expedited), s=1, n=1 (length = 4 -n), d = 61 62 63).



Server: *Initiate SDO download response* (OK for index x, subindex y).

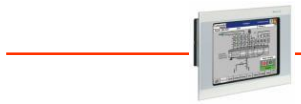
CAN trace (node 3, index 1234, subindex 05)



The SDO transfer protocols

(**Expedited**-, Segmented- or Block-transfer)

Expedited Upload ("read node OD")



Client: *Initiate SDO upload request*
(ex: index = x, subindex = y).

Server: *Initiate SDO upload response*
(ex: 4 byte: e=1 (expedited), s = 1,
n=0 ► length = 4 byte (formula p. 9.24),
d = 10 20 30 40).

CAN trace (node 3, index 4567, subindex 0E)

00000603	40	67	45	0E	00	00	00	00
00000583	42	67	45	0E	10	20	30	40

If 32 bits value: 0x40302010

The SDO transfer protocols

(Expdited-, **Segmented**- or Block-transfer)

Segmented Download ("write node OD")

Client: *Initiate SDO download request* index 0x1235 and subindex 0x5 (ex: 9 bytes - e=0 (not expedited), s=1 (size given), d = **9** (lenght)).

Server: *Initiate SDO download response* (OK).

Client: *Download SDO segment request* (t = 0 (toggle) c = 0 (more segments), n=**7**).

Server: *Download segment response* (t = 0 (toggle))

Client: *Download SDO segment request* (t = 1 (toggle) c = 1 (no more segments), n=**2**).

Server: *Download segment response* (t = 1 (toggle))

CAN trace (node 3, index 1235, subindex 05)

00000603	21	35	12	05	**09**	00	00	00
00000583	60	35	12	05	00	00	00	00
00000603	**00**	*31*	*32*	*33*	*34*	*35*	*36*	*37*
00000583	20	00	00	00	00	00	00	00
00000603	**1B**	*38*	*39*	00	00	00	00	00
00000583	30	00	00	00	00	00	00	00

The SDO transfer protocols

(Expdited-, **Segmented**- or Block-transfer)

Segmented Upload ("read node OD")

Client: *Initiate SDO upload request*
(ex: index = 0x5678, subindex = 0x5).

Server: *Initiate SDO upload response*
(ex: **5** bytes: e=0 (not expedited), s = 1, d = **5**).

Client: *Upload SDO segment request*
(ex: t = 0 (toggle)).

Server: *Download SDO segment response*
(t = 0 (toggle) c = 1 (no more segments), n=**5,** data = 00 01 02 03 04).

CAN trace (node 3, index 0x5678, subindex 0x07)

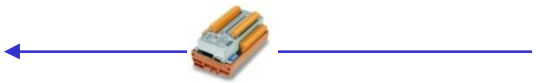
00000603	40 78 56 07 00 00 00 00
00000583	41 78 56 07 **05** 00 00 00
00000603	60 00 00 00 00 00 00 00
00000583	0**5** *00 01 02 03 04* 00 00

The SDO transfer protocols

(Expdited-, Segmented- or **Block-transfer**)

Block transfer Download ("write node OD")

Client: *Initiate SDO block download request* (ex: index = x, subindex = y).



Server: *Initiate SDO block download response* with given (5) max number of segments per block…

1st …

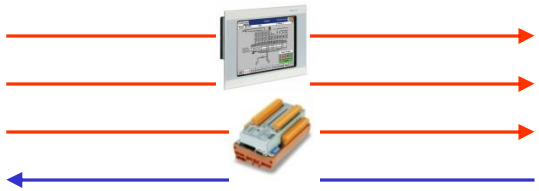
Client: *SDO block download* 1st block segment (sequence numbering).

2nd …

3rd …

4th …

5h …



Client: *SDO block download response.*

00000603	C2 34 12 05 3E 00 00 00
00000583	A0 34 12 05 05 00 00 00
00000603	00 54 68 69 73 20 73 74
00000603	01 72 69 6E 67 20 68 61
00000603	02 73 20 62 65 65 6E 20
00000603	03 73 65 6E 74 20 76 69
00000603	04 61 20 43 41 4E 6F 70
00000583	A2 04 05 00 00 00 00 00
00000603	00 65 6E 20 42 4C 4F 43
00000603	01 4B 20 54 52 41 4E 53
00000603	02 46 45 52 20 50 52 4F
00000603	83 54 4F 43 4F 4C 00 00
00000583	A2 03 05 00 00 00 00 00
00000603	C5 00 00 00 00 00 00 00
00000583	A1 00 00 00 00 00 00 00

CRC

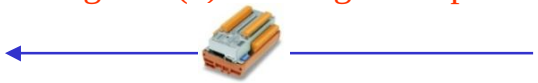
Auto repeat

The SDO transfer protocols

(Expdited-, Segmented- or **Block-transfer**)

Block transfer Upload ("read node OD")

Client: *Initiate SDO block upload request* (ex: index = x, subindex = y) with given (6) max segments per block.



1st …

Server: *SDO block upload* (sequence number 1)

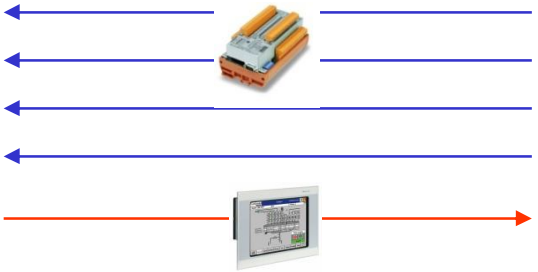
2nd …

3rd …

4th …

5th …

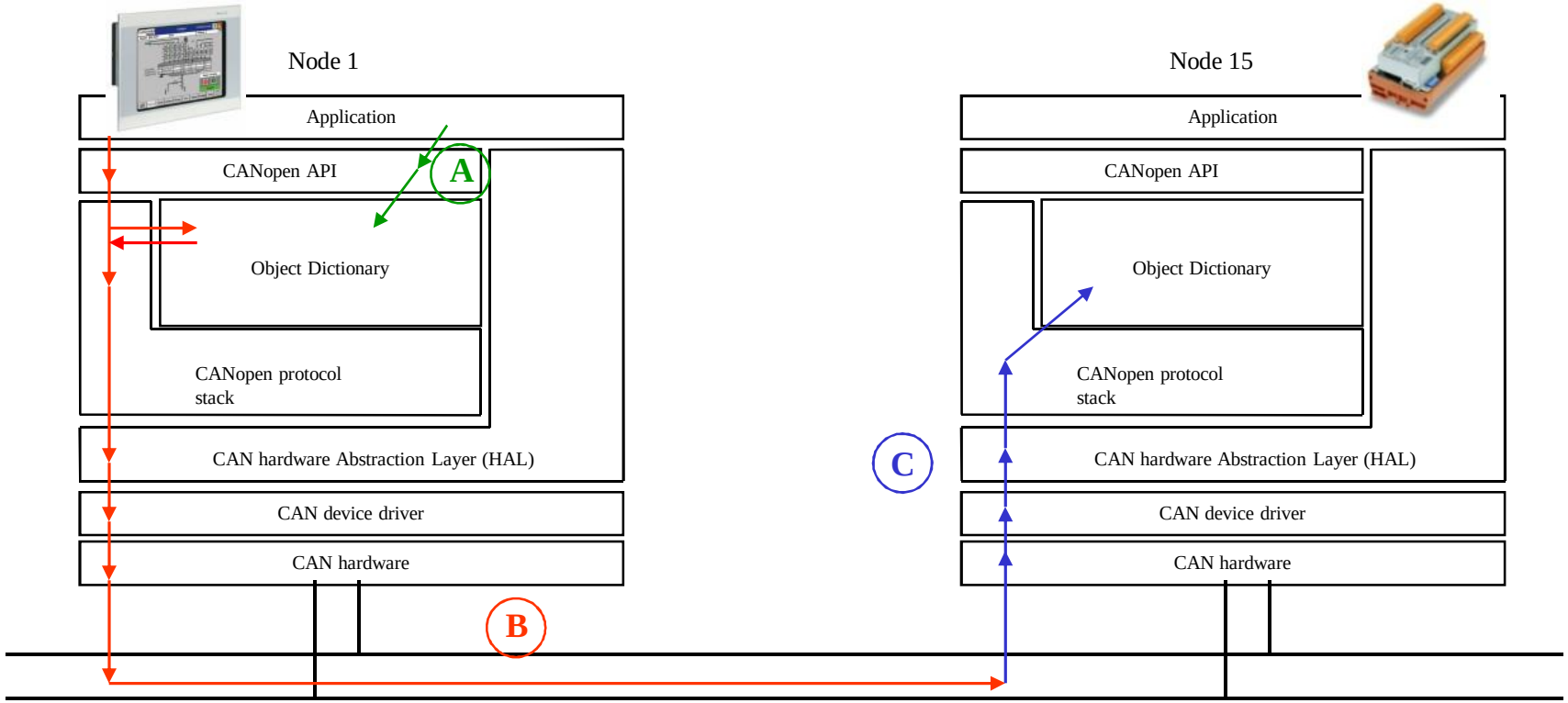
6h …



Client: *SDO block upload response.*

CAN trace N/A

Setup communication



A) Node 1 configures one of it's SDO client (in this example SDO client 1) to connect to the SDO server 1 (default SDO server) on node 15.

B) Node 1 starts the read/write operation via the CANopen API using SDO client no.1.

C) Node 15's Server SDO 1 responds to the request.

Default connection set

Object	COBID
Broadcast Network Management (NMT)	0x000
Synchronization (SYNC)	0x080
Emergency (EMCY)	0x080 + NodeId
Transmit PDO 1	0x180 + NodeId
Receive PDO 1	0x200 + NodeId
Transmit PDO 2	0x280 + NodeId
Receive PDO 2	0x300 + NodeId
Transmit PDO 3	0x380 + NodeId
Receive PDO 3	0x400 + NodeId
Transmit PDO 4	0x480 + NodeId
Receive PDO 4	0x500 + NodeId
Server SDO (TX)	0x580 + NodeId
Server SDO (RX)	0x600 + NodeId
Module error control, boot-up protocol, heartbeat etc.	0x700 + NodeId

Object Dictionary

Object Index	Sub Index	Data Type	Bit contents	Description
0x1020				Verify configuration.
	0x1	UINT32		Configuration Date
	0x2	UINT32		Configuration Time
0x1028				Emergency Consumer
	0x1 – 0x7f	UINT32		Emergency Consumer COBID (1 – 127)
0x1200 – 0x127f		SERVER SDO		SERVER SDO 1 – 128
	0x1	UINT32		COBID Client to server (RX)
	0x2	UINT32		COBID Server to client (TX)
	0x3	UINT8		Node Id of the SDO client
0x1280 – 0x13ff		CLIENT SDO		CLIENT SDO 1 – 128
	0x1	UINT32		COBID Client to server (TX)
	0x2	UINT32		COBID Server to Client (RX)
	0x3	UINT8		Node Id of the SDO server

Connect masters client SDO 1 to default server on node 15.

Object Index	Sub Index	Data Type	Bit contents	Description
0x1028				Emergency Consumer
	0x1 – 0x7f	UINT32		Emergency Consumer COBID (1 – 127)
0x1200 – 0x127f		SERVER SDO		SERVER SDO 1 – 128
	0x1	UINT32		COBID Client to server (RX)
	0x2	UINT32		COBID Server to client (TX)
	0x3	UINT8		NodeId of the SDO client
0x1280 – 0x13ff		CLIENT SDO		CLIENT SDO 1 – 128
	0x1	UINT32		COBID Client to server (TX)
	0x2	UINT32		COBID Server to Client (RX)
	0x3	UINT8		Node Id of the SDO server

15

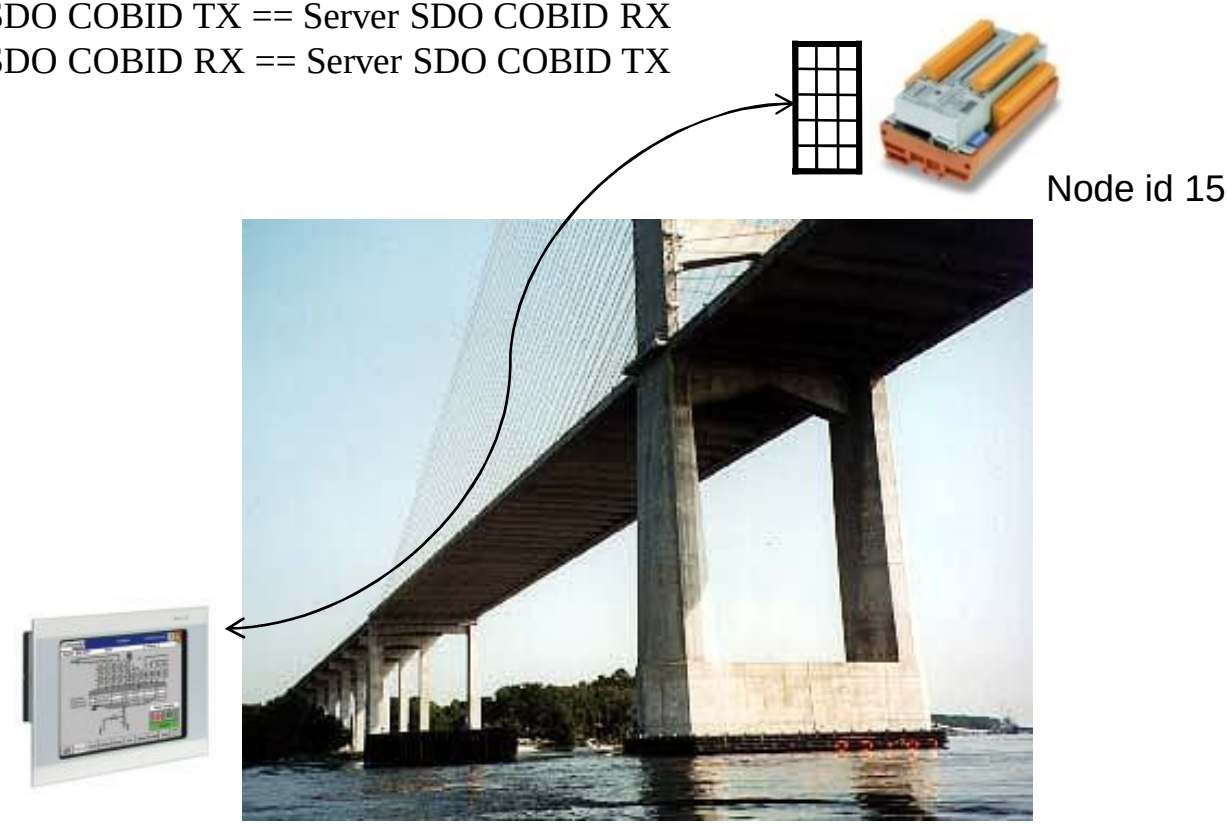
Client SDO 1 is found at object index **0x1280**.

According to default connection set the slave's default SDO server RX on COBID **"0x600 + 15"** and therefore we shall configure or Client SDO 1 to TX on that COBID.

According to default connection set the slave's default SDO server was TX on COBID **"0x580 + 15"** and therefore we shall configure or Client SDO 1 to RX on that COBID.

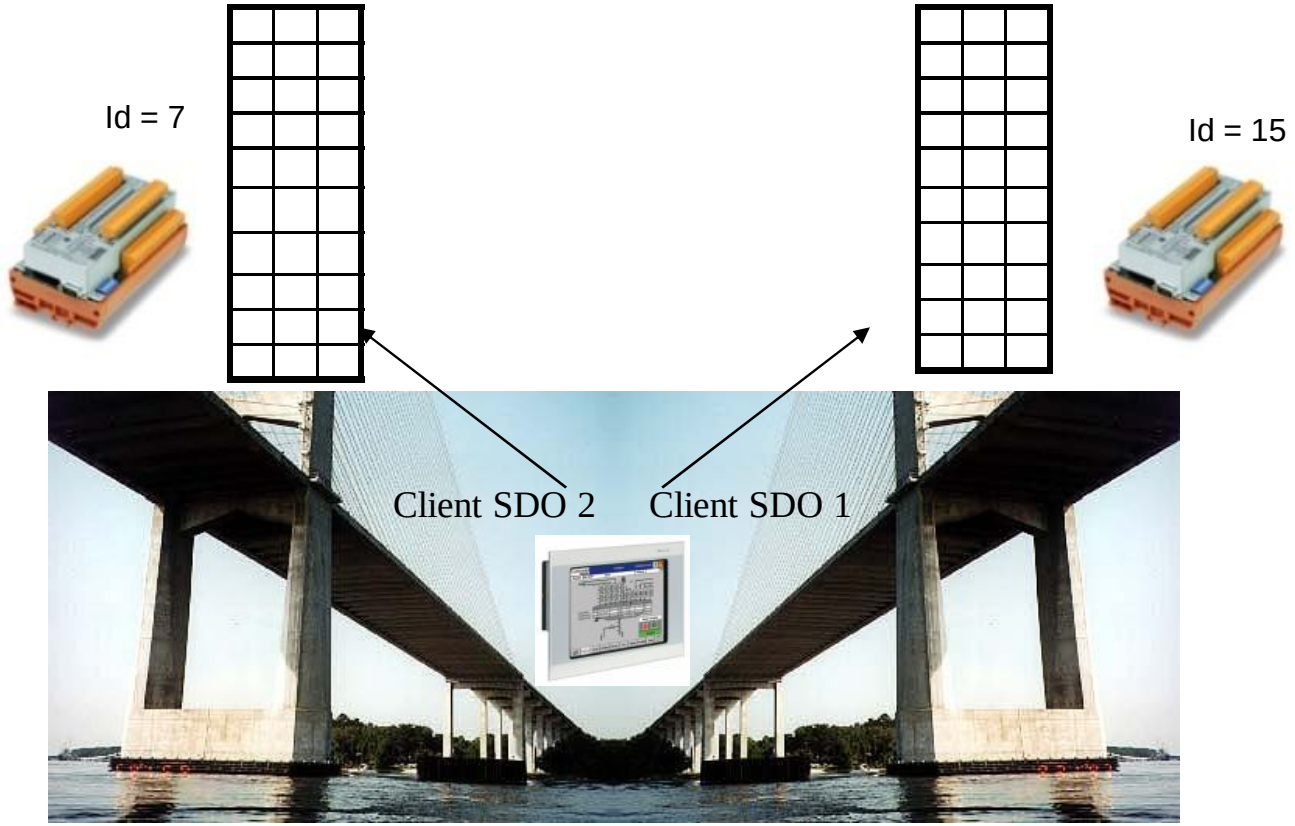
Result of SDO client configuration

Client SDO COBID TX == Server SDO COBID RX
Client SDO COBID RX == Server SDO COBID TX



Node id 15

If more nodes and more Client SDOs are available…



CANopen master can have up to 128 client SDOs (=128 connections)

Configure a SDO connection

Connect Client SDO to *connect to default SDO of* remote node 15 & 7.

Object Index	Sub Index	Data Type	Description +Value
0x127f		SERVER SDO	SERVER SDO 128
0x1280	0x0	CLIENT SDO	CLIENT SDO 1 (subIdx == 3)
	0x1	UINT32	0x600 + 15 (COBID Client to server (TX))
	0x2	UINT32	0x580 + 15 (COBID Server to Client (RX))
	0x3	UINT8	15
0x1281	0x0	CLIENT SDO	CLIENT SDO 2 (subIdx == 3)
	0x1	UINT32	0x600 + 7 (COBID Client to server (TX))
	0x2	UINT32	0x580 + 7 (COBID Server to Client (RX))
	0x3	UINT8	7
– 0x13ff	....	...	CLIENT SDO 128

Process Data Object (PDO)

- Sent in run-time to control the running process.
- Carry the real time process data

2 types of PDOs

Transmit PDO

Receive PDO

Node functionality

Process computer

Node ID

SDO client

Object Dictionary

SDO server

Receive PDO

Transmit PDO

Network node

Node ID

Object Dictionary

SDO server

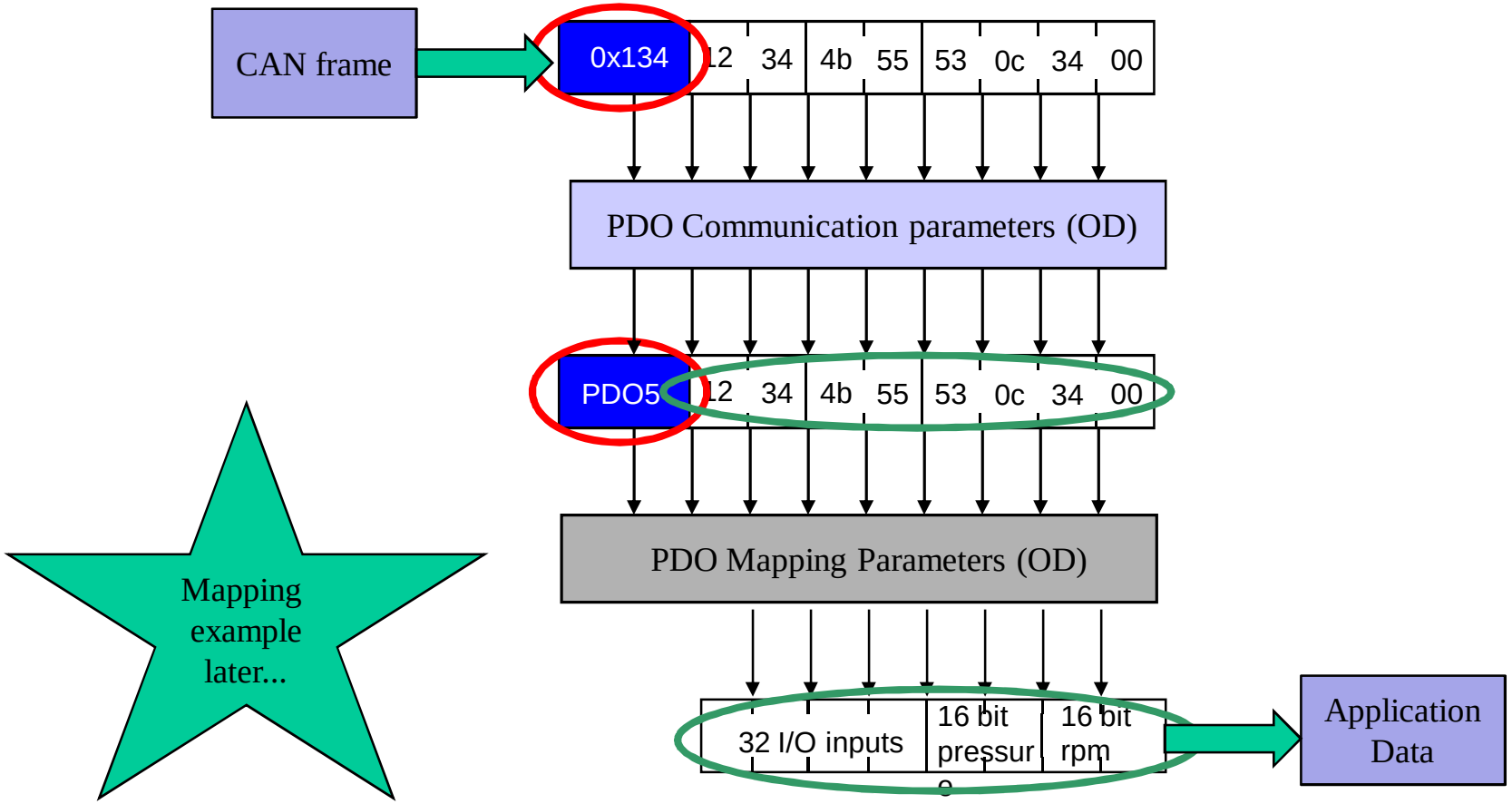
Transmit PDO

Receive PDO

Configuration tool

SDO client

PDO decoding example

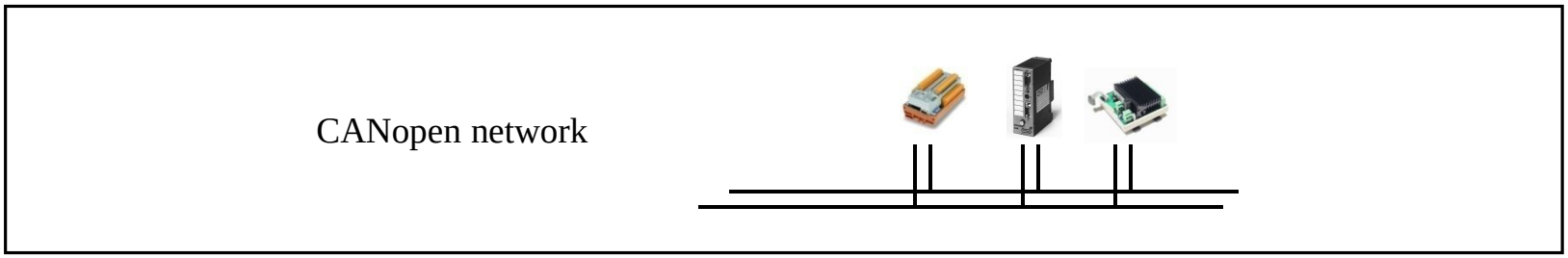


Types of PDOs

- Event driven
- Timer driven
- Remotely requested
- Synchronized

Event driven TPDO

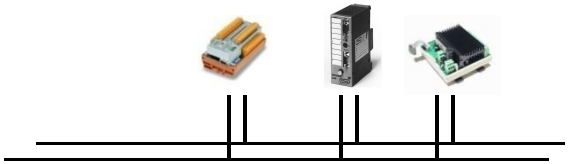
(can cause delay problem)



Event driven TPDO with inhibit time

(solves the event burst problem)

CAN bus configuration



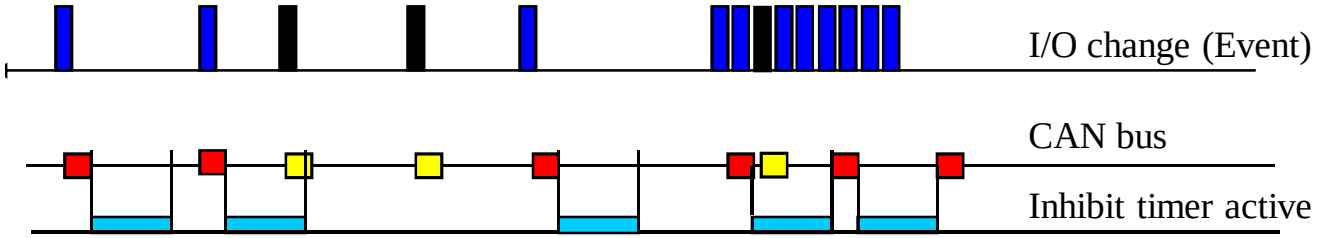
= Events that trig to be sent

= Events that trig to be sent

= Pending inhibit time timer

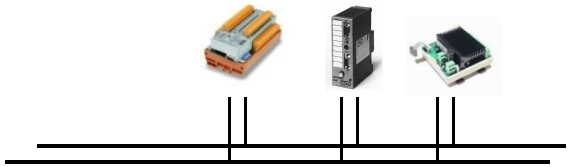
= PDO sent from upon event (I/O input change)

= PDOs sent from other nodes luckily not delayed)



Timer driven PDO transfer

CAN bus configuration



= Events that trig to be sent

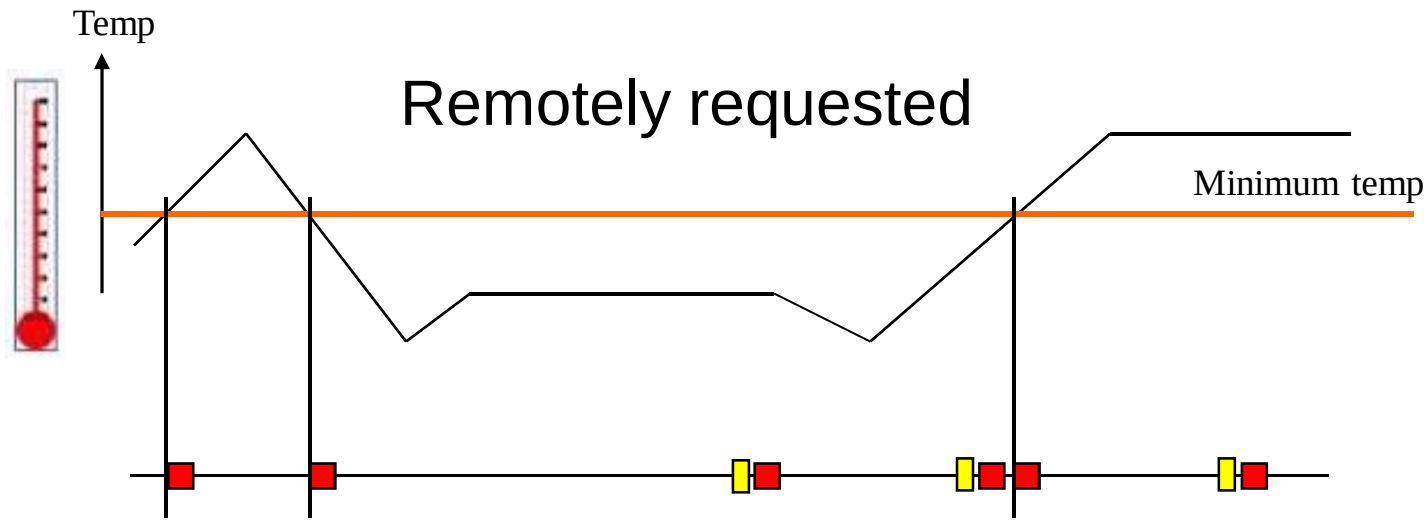
= Events that trig to be sent

= Pending Event timer

= PDO sent from on event (I/O input change)

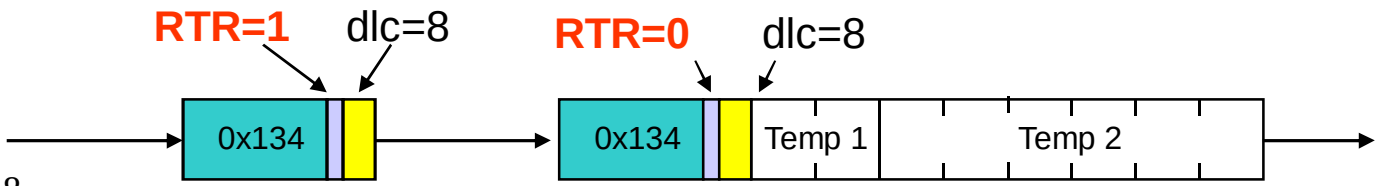
= PDOs sent from other node.

Remotely requested PDO

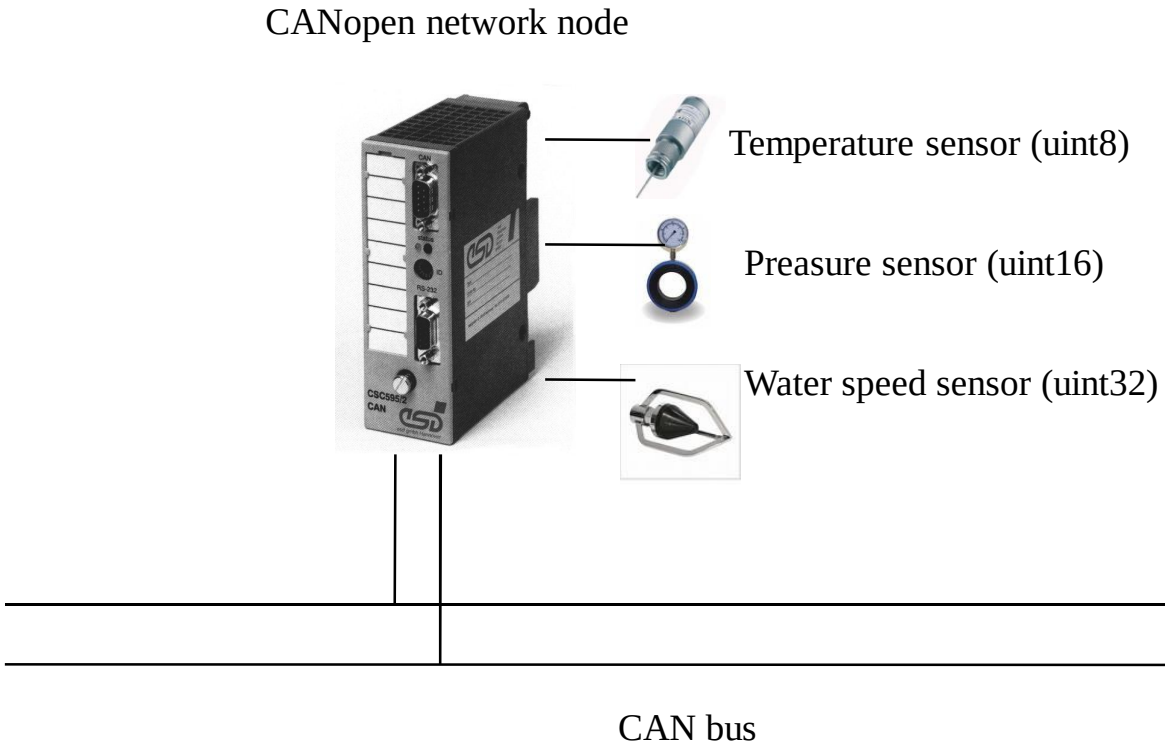


= Temp PDO sent from temp node if the value passes a ”Minimum temp” or it gets requested by master.

= Remote request PDO sent from master (can be sent at any time)



PDO mapping example (1/6)



PDO mapping example (2/6)

Register the programming variables in
the Object Dictionary of the node.

Object Index	Sub Index	Data Type	Bit contents	Description
0x2200	0	UINT8		TEMP
0x2201	1	UINT16		PREASURE SENSOR
0x2201	2	UINT32		WATER SPEED

PDO mapping example (3/6)

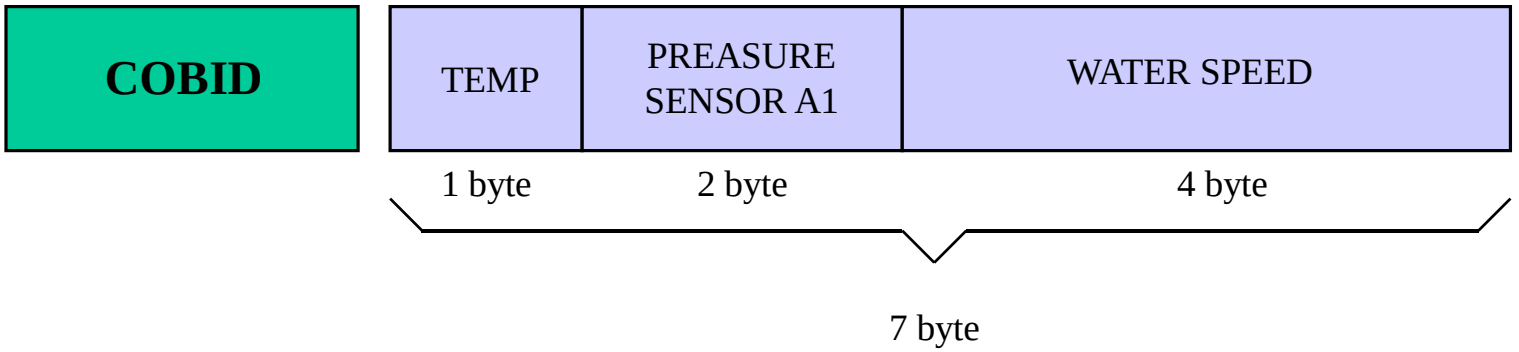
Object Index	Sub Index	Data Type	Bit contents	Description
0x2200	0	UINT8		TEMP
0x2201	1	UINT16		PREASURE SENSOR A1
0x2201	2	UINT32		WATER SPEED

Object Index	Sub Index	Data Type	Description
0x1a00		TPDO MAP	Transmit PDO1 mapping parameters.
0x1a01		TPDO MAP	Transmit PDO2 mapping parameters
	0x0	UINT8	Number of objects to map.
	0x1	UINT32	0x2200-00-08
	0x2	UINT32	0x2201-01-16
	0x3	UINT32	0x2201-02-32
0x1a02		TPDO MAP	Transmit PDO3 mapping parameters
0x1a03		TPDO MAP	Transmit PDO4 mapping parameters
… -0x1bff		TPDO MAP	Transmit PDOn mapping parameters

PDO mapping example (4/6)

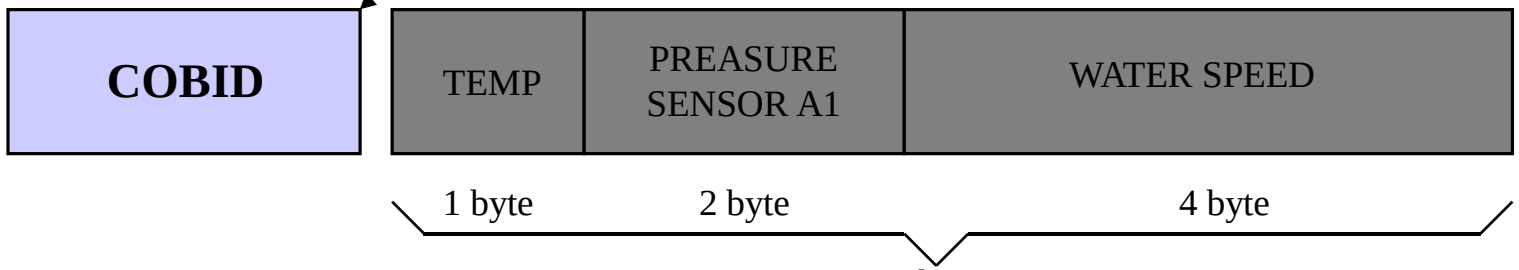
Object Index	Sub Index	Data Type	Bit contents
0x1a01	0	UINT8	3
0x1a01	1	UINT32	0x2200-00-8
0x1a01	2	UINT32	0x2201-01-16
0x1a01	3	UINT32	0x2201-02-32

3



PDO mapping example (5/6)

Object Index	Sub Index	Data Type	Description
0x1600 – 0x17ff		RPDO MAP	Receive PDO mapping (RPDO1 - ...)
	0x1 – 0x40	UINT32	PDO mapping for n-th object to be mapped.
0x1800	0x0 – 0x5	PDO COMM PARAMS	Transmit PDO 1 Communication Parameters.
0x1801	0	UINT8	Transmit PDO 2 Communication Parameters (SubIdx 0 == 5.)
	0x1	UINT32	COBID
	0x2	UINT8	Transmission Type.
	0x3	UINT16	Inhibit Time.
	0x4	-	Comp ability entry.
	0x5	UINT16	Event Timer
0x1802			Transmit PDO 3 Communication Parameters.
…	0x1 – 0x40	UINT32	Transmit PDO n Communication Parameters.



PDO mapping example (6/6)

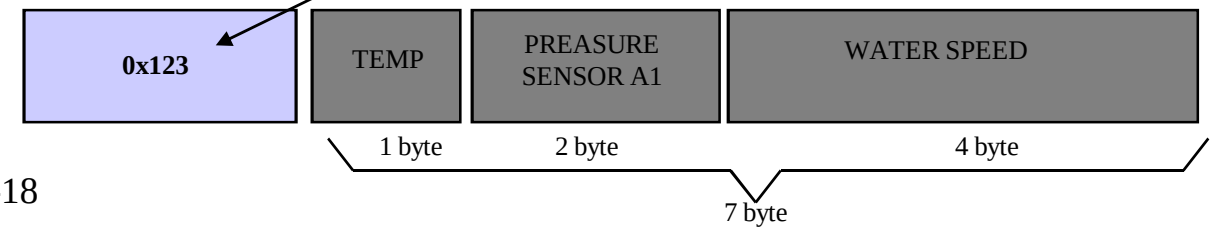
Object Index	Sub Index	Data Type	Bit contents
0x1a01	0 (no. objects to me mapped)	UINT8	3
0x1a01	1 (1st object to be mapped)	UINT32	0x2200-00-8
0x1a01	2 (2nd object to be mapped)	UINT32	0x2201-01-16
0x1a01	3 (3rd object to be mapped)	UINT32	0x2201-02-32

3

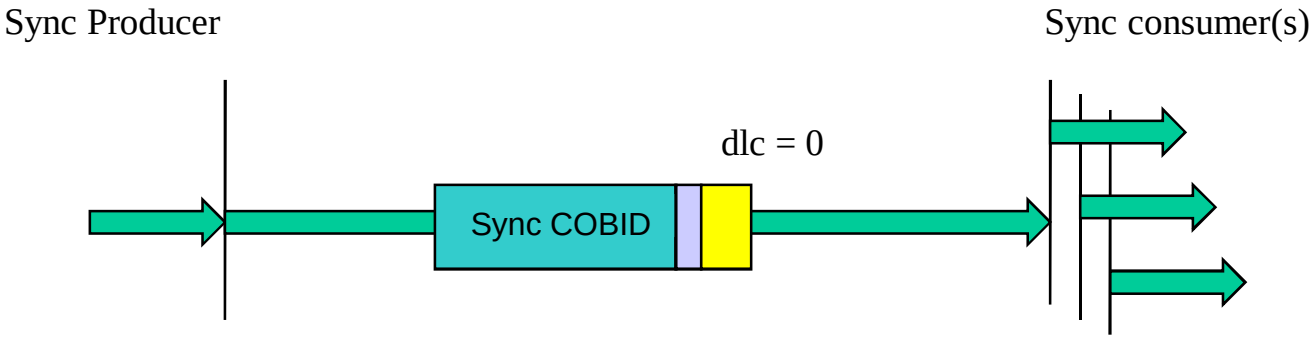
+

Object Index	Sub Index	Data Type	Bit contents
0x1801	1 (COBID)	UINT32	0x123
0x1801	2 (Transmission type)	UINT8	254
0x1801	3 (Inhibit time)	UINT16	100 (*10us)
0x1801	4 ()	0	0
0x1801	5 (Event timer)	UINT16	1000 (*1ms)

=



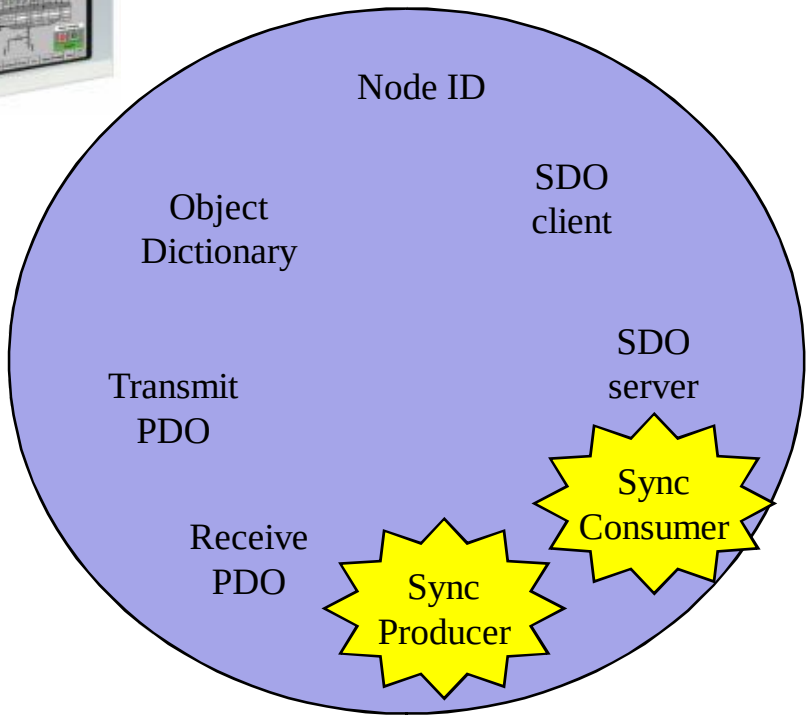
Sync “pulse” for sync PDO



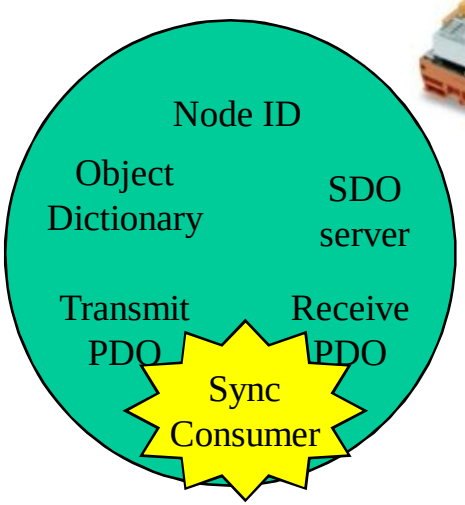
Object	Parameter
0x1005	COB-ID (Bit 30: consumer/producer flag)
0x1006	Communication cycle period

Node functionality

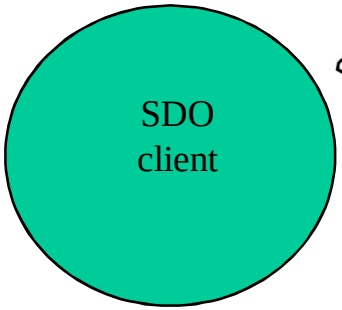
Process computer



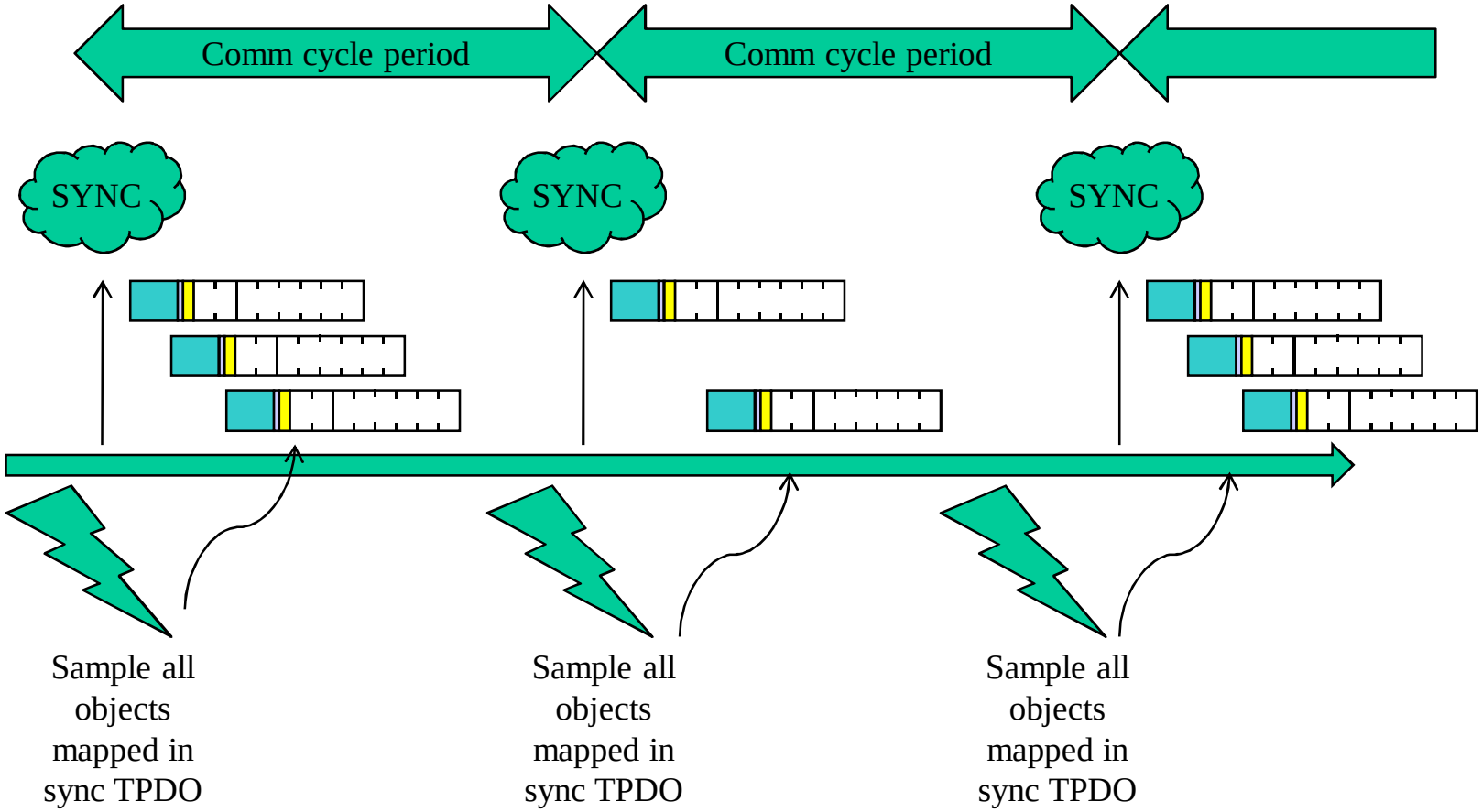
Network node



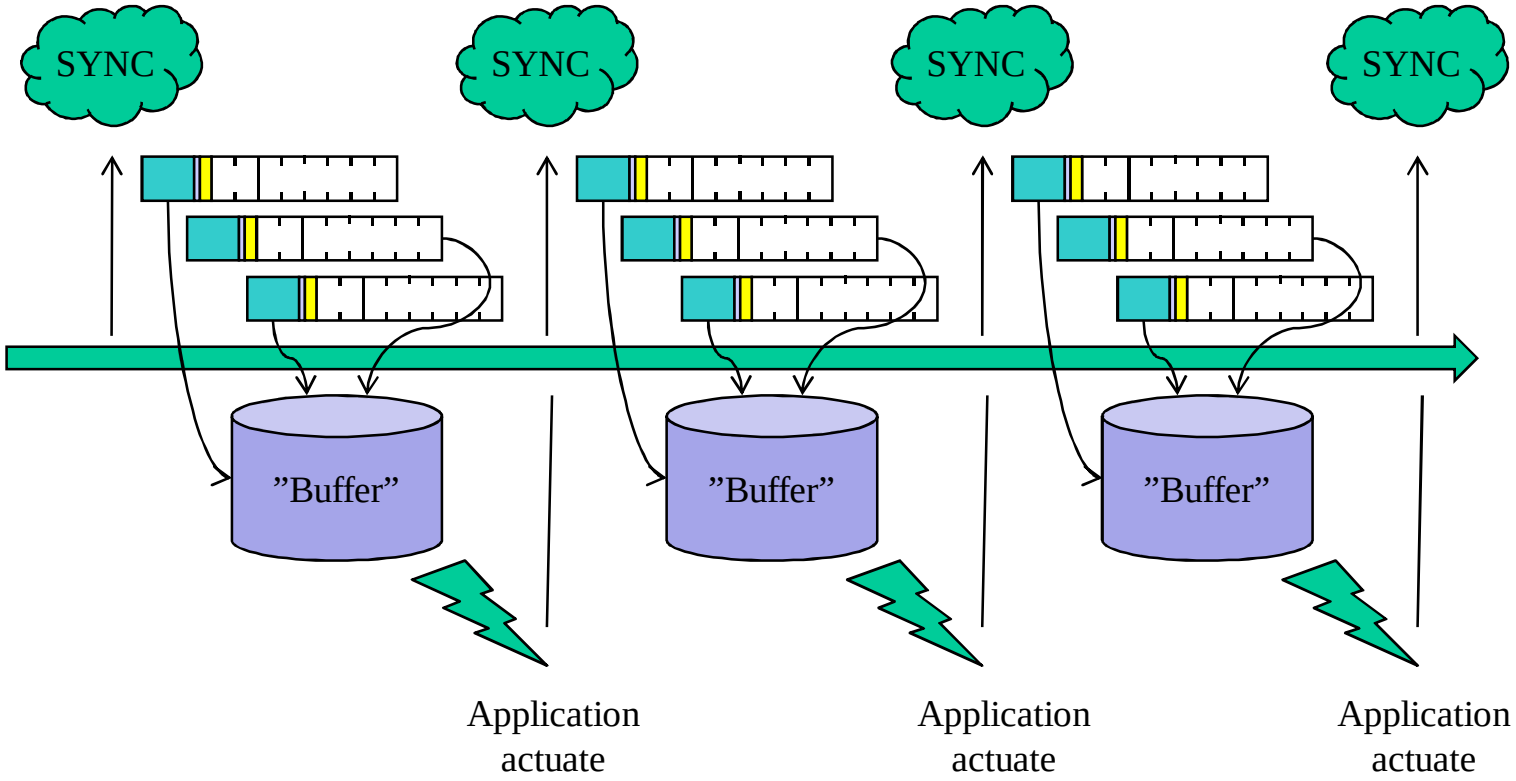
Configuration tool



Sync of TPDO


Comm cycle period
Comm cycle period
SYNC
SYNC
SYNC
Sample all objects mapped in sync TPDO
Sample all objects mapped in sync TPDO
Sample all objects mapped in sync TPDO

Sync of RPDO


SYNC
SYNC
SYNC
SYNC
”Buffer”
”Buffer”
”Buffer”
Application actuate
Application actuate
Application actuate

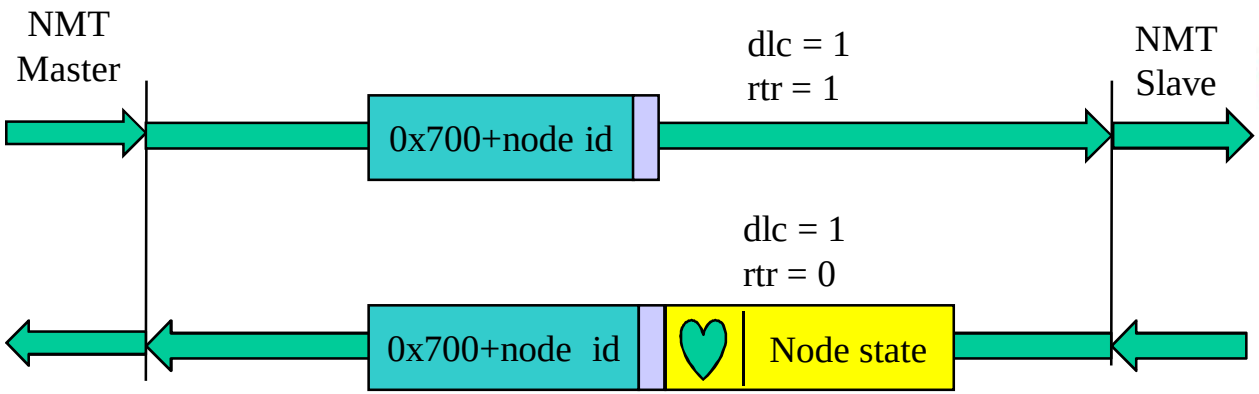
PDO transmission type

Transm. Type	Meaning for a transmit PDO	Meaning for a receive PDO
0	Sent on next SYNC if event or request has been made.	Application updated on next SYNC.
1 < n < 240	Sent on every n SYNC	Application updated on next SYNC.
241 <= n < 252	UNDEFINED	UNDEFINED
252	Sent on next SYNC if PDO has been requested.	UNDEFINED
253	Sent independent of SYNC upon request.	UNDEFINED
254 -255	Sent independent of SYNC in all cases	Application is updated upon reception of PDO

Error Control Protocols

- Node Guarding Protocol
- Heartbeat Protocol
- Bootup Protocol

Node Guarding Protocol



Name	Description
Node guard time	Period time guard request.
Node life time	=”Life time factor” * ”Node guard time”. If no response from Slave node – node considered dead (Node guard event is trigged).
Life Guard Event	Event trigged if NMT slave has not been polled within ”Node Life time” (jumps to pre-operational mode).
Node Guard Event	Event trigged if node guard time time has elapsed and no response from slave.

CAN trace (node 2)

```
00000702  R    [no data]
00000702       81
00000702  R    [no data]
00000702       01
00000702  R    [no data]
00000702       81
```

Heart Beat Protocol

Node functionality

Process computer

Node ID

Object Dictionary

SDO client

NMT Master

Transmit PDO

Heartbeat Consumer

SDO server

Receive PDO

Sync Producer

Network node

Node ID

Object Dictionary

NMT Slave

SDO server

Transmit PDO

Heartbeat Producer

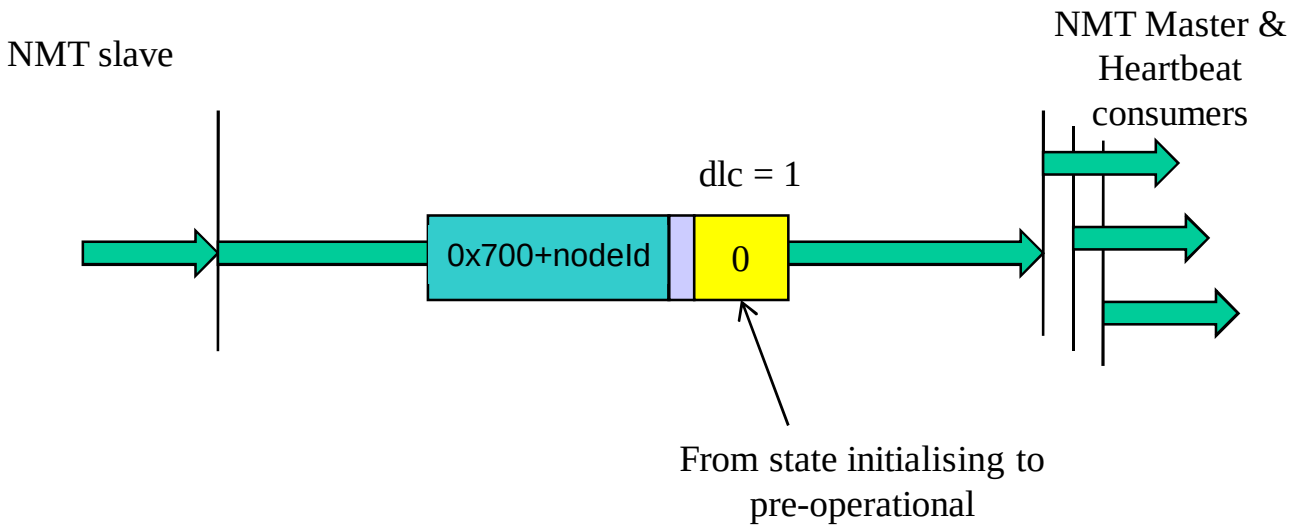
Receive PDO

Sync Consumer

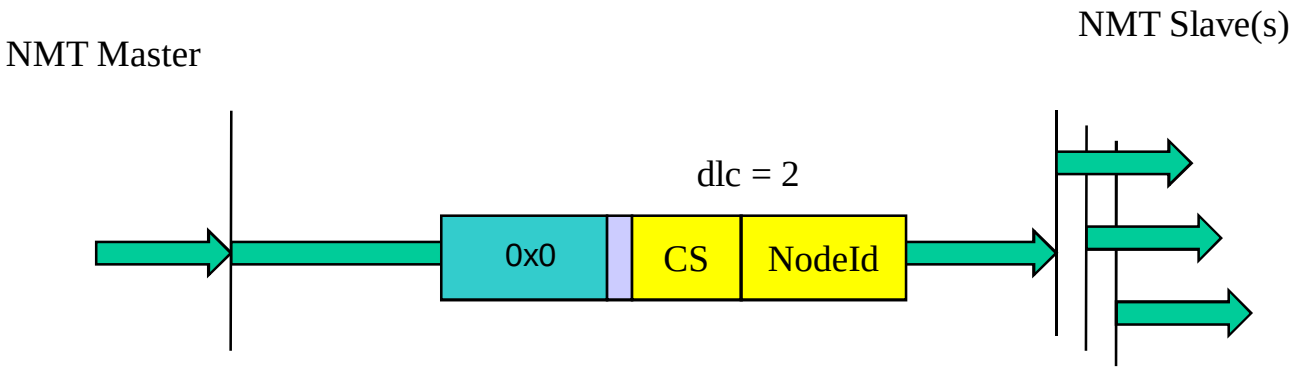
Configuration tool

SDO client

Bootup Protocol



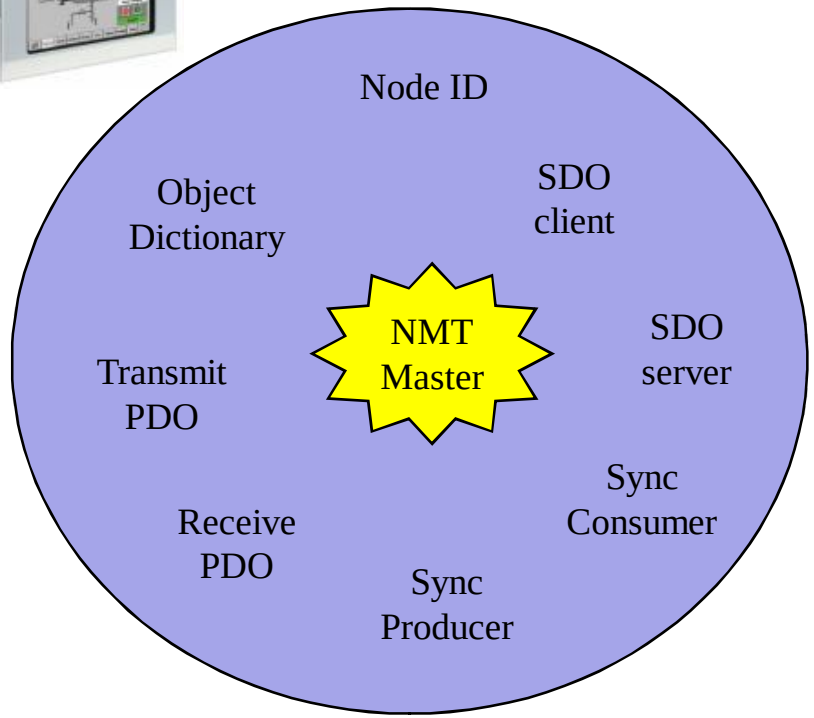
Module Control Protocols



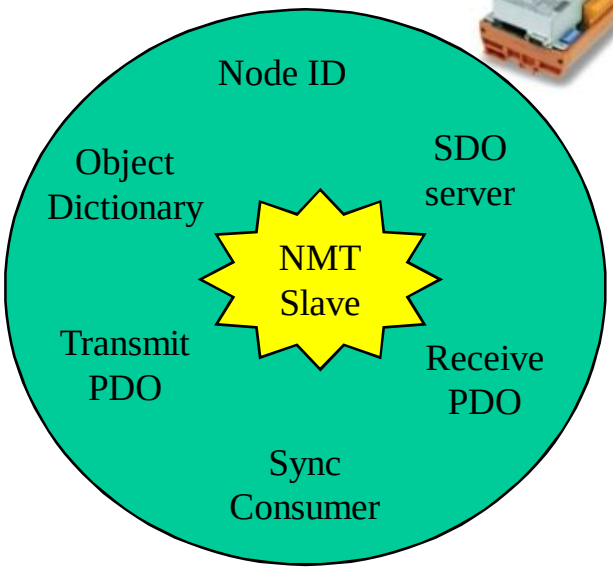
CS (Command Specifier)	Command
1	Start Remote Node
2	Stop Remote Node
128	Enter Pre-Operational Mode
129	Reset Node
130	Reset Communication

Node functionality

Process computer



Network node



Configuration tool

Node functionality

Master

Node ID

Object Dictionary

SDO client

NMT Master

SDO server

Transmit PDO

Heartbeat Consumer

Receive PDO

Sync Producer

Slave

Node ID

Object Dictionary

NMT Slave

SDO server

Transmit PDO

Heartbeat Producer

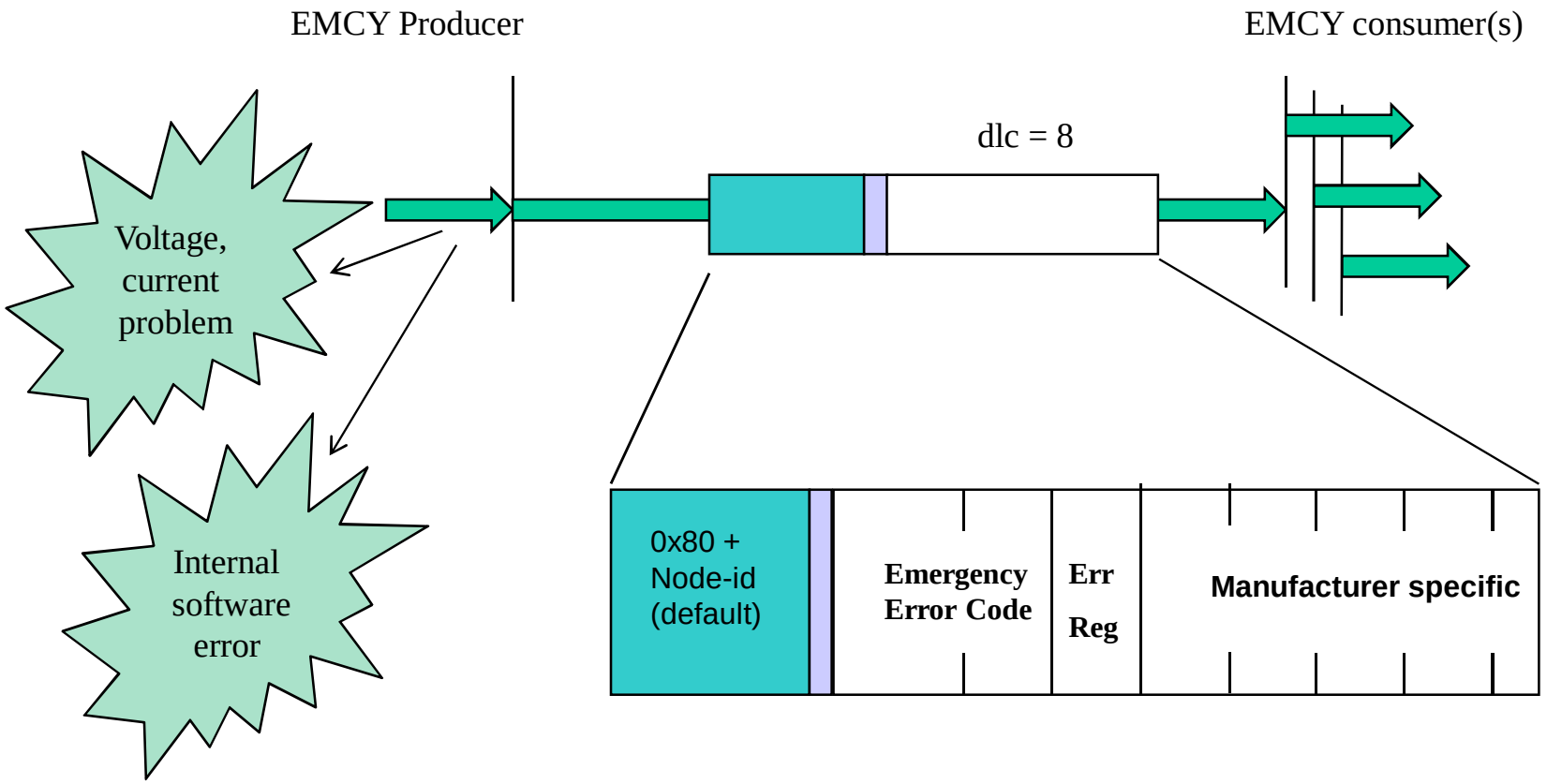
Receive PDO

Sync Consumer

Configuration tool

SDO client

Emergency Protocol



Pre-defined error field (0x1003)

• The object at index 0x1003 holds the errors that have occurred on the device and have been signaled via the **Emergency Object**.

•The entry at sub-index 0 contains the number of actual errors

• Every new error is stored at sub-index 1, the older ones move down the list.

•Writing a 0 to sub-index 0 deletes the entire error history (empties the array).

Node functionality

Master

Node ID

Object Dictionary

SDO client

NMT Master

SDO server

Transmit PDO

Heartbeat Consumer

Emcy Consumer

Receive PDO

Sync Producer

Slave

Node ID

Object Dictionary

NMT Slave

SDO server

Receive PDO

Heartbeat Producer

Transmit PDO

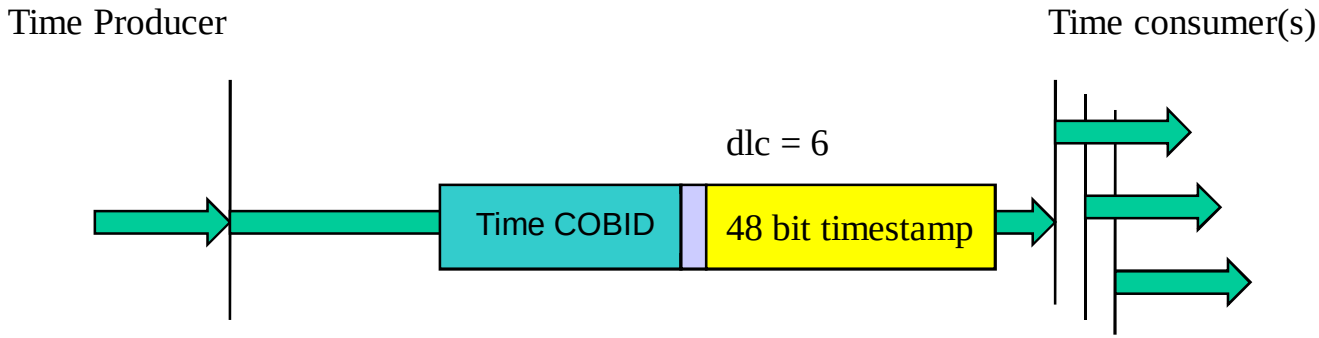
Emcy Producer

Sync Consumer

Configuration tool

SDO client

Time Stamp Object



Object	Paramter
0x1012	COB-ID (Bit 30: consumer/producer flag)

Node functionality

Master

Node ID

SDO client

Object Dictionary

NMT Master

SDO server

Transmit PDO

Heartbeat Consumer

Emcy Consumer

Receive PDO

Sync Producer

Time Producer

Slave

Node ID

SDO server

Object Dictionary

NMT Slave

Receive PDO

Heartbeat Producer

Transmit PDO

Emcy Producer

Sync Consumer

Time Consumer

Configuration tool

SDO client

Press release for Device Profile 445 (RFID) example

Erlangen, Germany, Septmeber 4, 2007 -- The CAN in Automation (CiA) nonprofit organization has released the CiA 445 CANopen device profile for RFID readers/writers. The objective of the profile is to enable easy system integration of RFID readers into networks in factory automation, laboratory automation, medical systems, product and asset management, identification systems, etc. *The device profile will make CiA 445-compliant RFID readers from different manufacturers interchangeable with a minimum of time and configuration effort…*

...The following companies have participated in the development of the profile specification within the CANopen Special Interest Group RFID:

DeLaval International, FH Regensburg, Hans Turck, ifm electronic, Ixxat Automation, RM Michaelides, Schneider Electric, Sick, Siemens Medical Solutions, Vector Informatik, and others.

EDS (Electronic data Sheet) file

•CiA-306

• The EDS belongs to the standard documents supplied with a CANopen device.

• A proper EDS file is required to pass the CiA-CANopen conformance test.

•In the future these files will be replaced by XML
device descriptions according to ISO 15745 (CiA-311)

•CANeds free-of-charge EDS generator/editor (www.canopen-forum.com)

EDS example

```
[FileInfo]
FileName=example1.eds
FileVersion=2
...
[DeviceInfo]
VendorName=xyz
...
BaudRate_50=0
BaudRate_125=1
BaudRate_250=1
...
[3000]
SubNumber=2
ParameterName=Demo_object
ObjectType=8
[3000sub00]
ParameterName=Highest sub-index supported
ObjectType=0x7
DataType=0x5
AccessType=ro
DefaultValue=0x1
PDOMapping=0
```

DCF (Device configuration File)

• Equal to a EDS file containg the device configuration paramters.

• Used by Configuration Manager (either stand alone tool or CANopen Master node).

•Stand alone tool for service engineers: www.tke.fi

• Configuring node standalone prevents node-id overlapping, faulty bitrate etc. that would cause bus chaos.

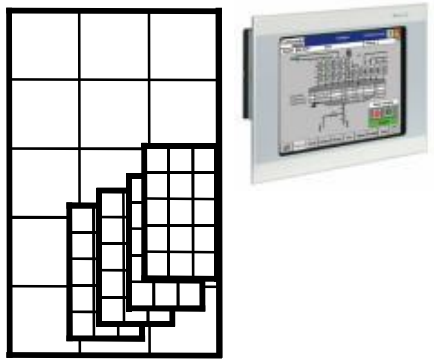
DS-302
Framework for Programmable CANopen Devices

CANopen Manager:

{ NMT Master, SDO Manager, Configuration Manager (opt) }

CANopen Manager entries in Object Dictionary:

* Power on bootup configuration (Am I the NMT master ?)
* Slave bootup { network list, device type, serial number,
  product code, keep alive (node guard),
  application SW version (optional upgrade) }
* Configuration Management (CMT)



This makes it possible
to use third party CANopen
design tools.

Object Dictionary

Object Index	Sub Index	Data Type	Bit contents	Description
0x1f26		ARRAY		Expected configuration time.
0x1f27		ARRAY		Expected configuration date.
0x1f50		ARRAY		Download program data.
0x1f51		ARRAY		Program Control.
0x1f52		APPL SW		Verify application software.
	0x1			Application software date.
	0x2			Application software time.
0x1f53		ARRAY		Expected application sw date.
0x1f54		ARRAY		Expected application sw time.
0x1f80		UINT8		NMT Startup
			BIT 0	0 = NMT slave, 1 = NMT master.
			BIT 1	0 = Start node by node, 1 = start all nodes.
			BIT 2	0 = Automatic start, 1 = Application allows start (node local)
			BIT 3	0 = Start slaves, 1 = Application starts slave.
			BIT 4	0 = slave specific err handling, 1 = Err control event handling
0x1f81 – 0x1f89		ARRAY		DS302 Boot Parameters...
0x1fa0 – 0x1fcf		ARRAY		Object Scanner List (Multiplexed PDOs)
0x1fd0 – 0x1fff		ARRAY		Object Dispatcher List (Multiplexed PDO)

Advanced CAN

- Controller types.
- Signal levels.
- Oscilloscope pictures.
- Error detectoion
- Connectors

Three types of CAN controllers

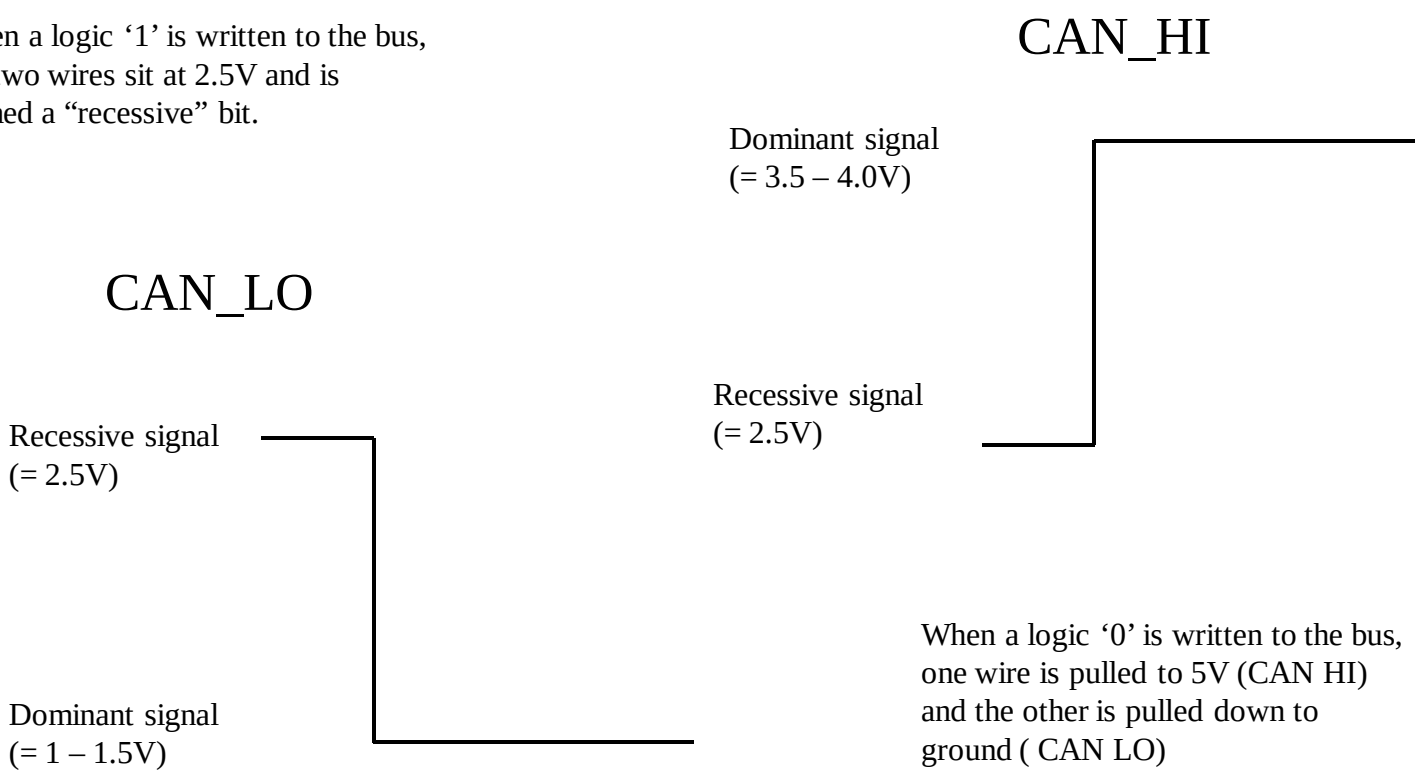
Part A and Part B comp ability

There are three types of CAN controllers: Part A, Part B passive and Part B. They are able to handle the different parts of the standard as follows:

CAN chip type	Part A	Part B passive	Part B
11 bit identifier	OK.	OK.	OK.
29 bit identifier	ERROR!	Tolerated on the bus, but ignored.	OK.

Recessive & Dominant levels

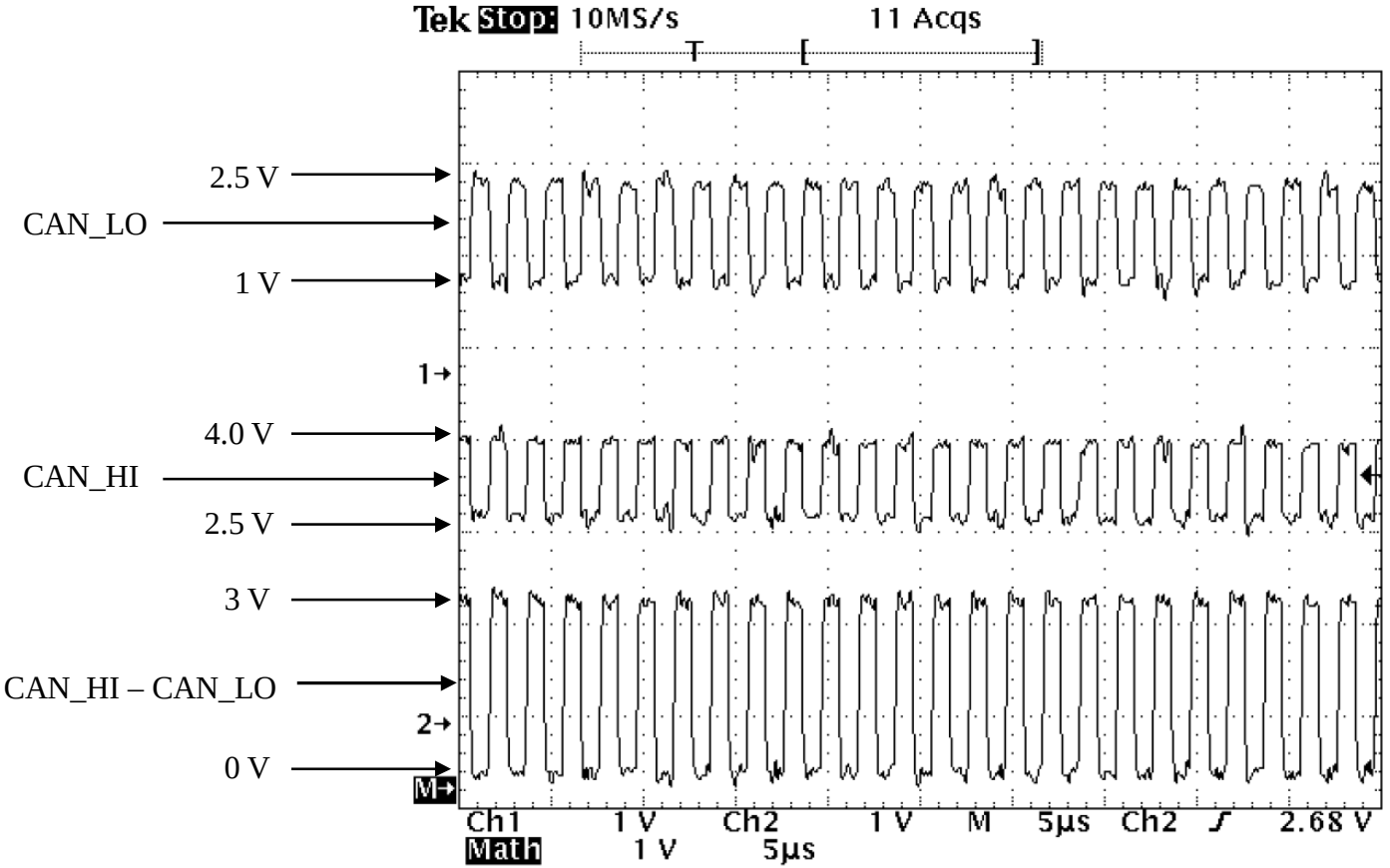
When a logic ‘1’ is written to the bus, the two wires sit at 2.5V and is termed a “recessive” bit.



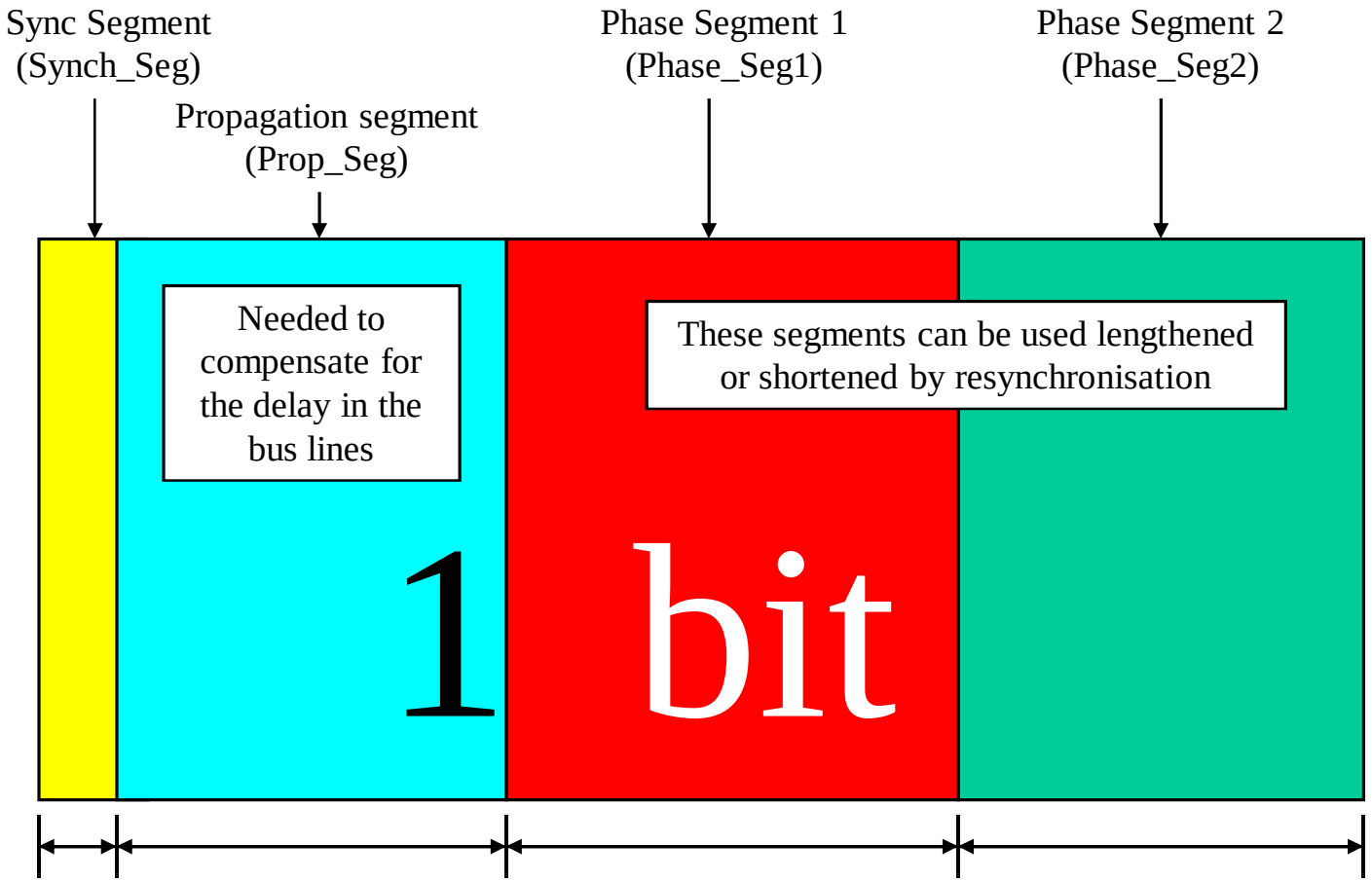
When a logic ‘0’ is written to the bus, one wire is pulled to 5V (CAN HI) and the other is pulled down to ground (CAN LO)

If both lines are at the same voltage, the signal is a recessive bit. **If the CAN_HI line is higher than the CAN_LO line by 0.9V, the signal line is a dominant bit**. If just one node is driving the bus to a logical 0 (=dominant bit), then the whole bus is in that state regardless of the number of nodes transmitting a logical 1.

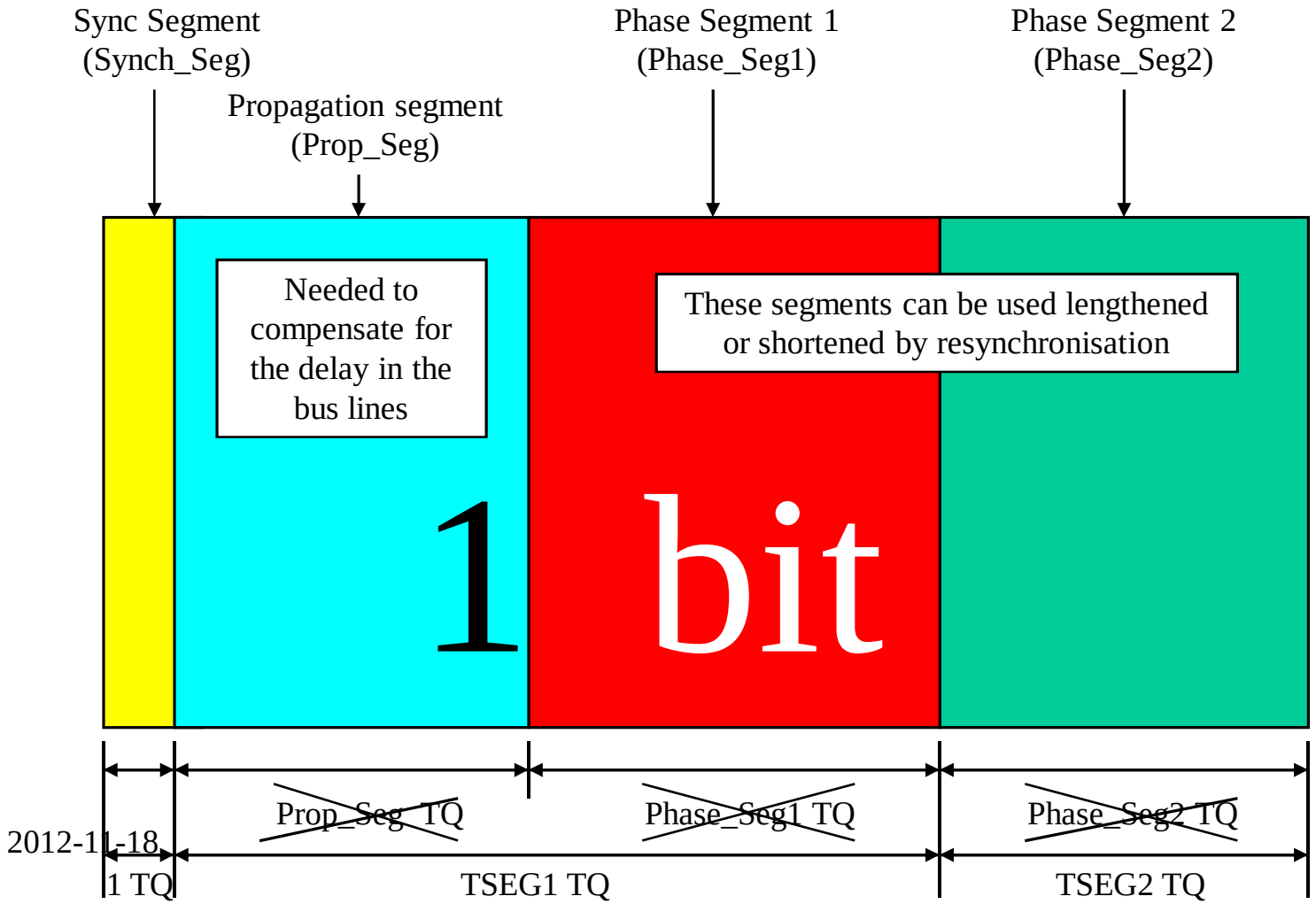
CAN signals


Tek Stop: 10MS/s
11 Acqs
2.5 V
CAN_LO
1 V
4.0 V
CAN_HI
2.5 V
3 V
CAN_HI – CAN_LO
0 V
Ch1 1 V Ch2 1 V M 5µs Ch2 2.68 V
Math 1 V 5µs

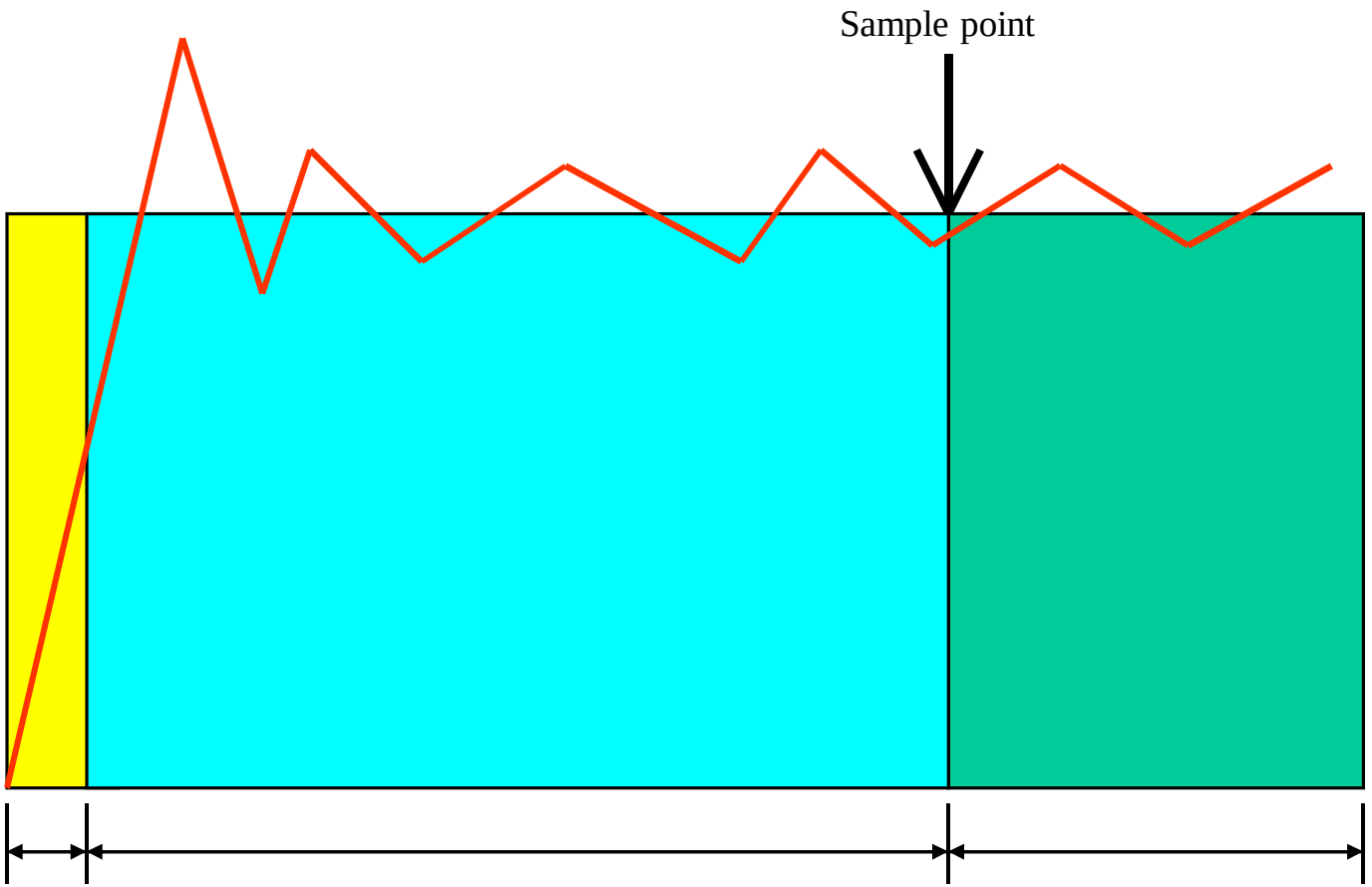
1 bit BOSCH-specification



1 bit on ISO11898-specification



Sample point per bit



Bitrate settings

$$n = SYNCHSEG+TSEG1+TSEG2$$

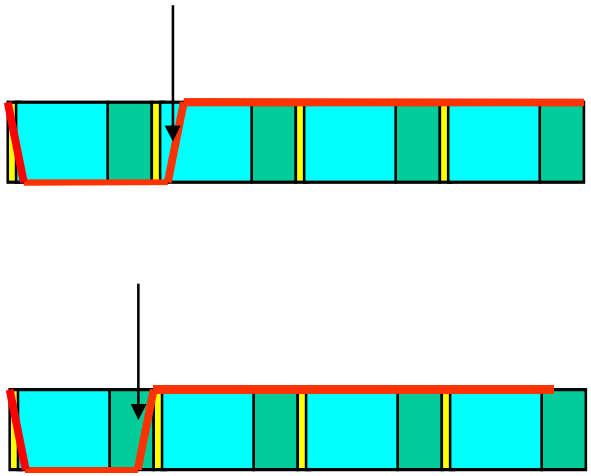
BRP = value of the Bit rate Prescaler
(register in the CAN controller)

$$Bitrate = \frac{f_{crystal}}{2*n*(BRP+1)}$$

Resync and SJW

Hard resynchronization

Resynchronization within a frame



Typical settings…

Bit rate Bus length [1]	Nominal bit time t_b	Number of time quanta per bit	Length of time quantum t_q	Location of sample point
1 Mbit/s 25 m	1 µs	8	125 ns	6 t_q (750 ns)
800 kbit/s 50 m	1,25 µs	10	125 ns	8 t_q (1 µs)
500 kbit/s 100 m	2 µs	16	125 ns	14 t_q (1,75 µs)
250 kbit/s 250 m [2]	4 µs	16	250 ns	14 t_q (3,5 µs)
125 kbit/s 500 m [2]	8 µs	16	500 ns	14 t_q (7 µs)
50 kbit/s 1000 m [3]	20 µs	16	1,25 µs	14 t_q (17,5 µs)
20 kbit/s 2500 m [3]	50 µs	16	3,125 µs	14 t_q (43,75 µs)
10 kbit/s 5000 m [3]	100 µs	16	6,25 µs	14 t_q (87,5 µs)

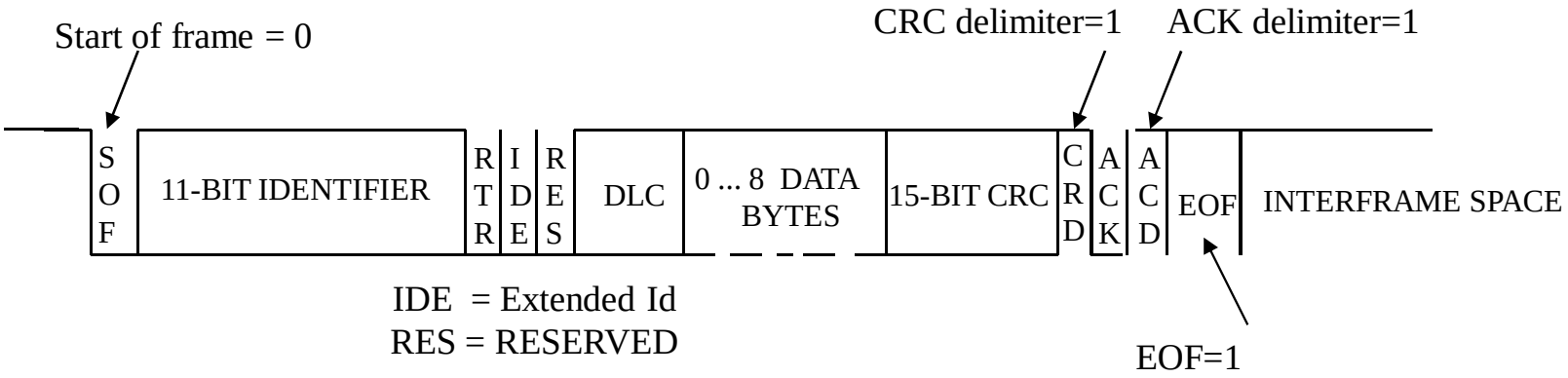
The five error checks…

- **Bit monitoring (read back)**
- **Bit stuffing (toggle required)**
- **Frame check (predefined values)**
- **Acknowledgement check (received?)**
- **CRC check.**

→ Error Frame → Automatic retransmission

Form- and biterror

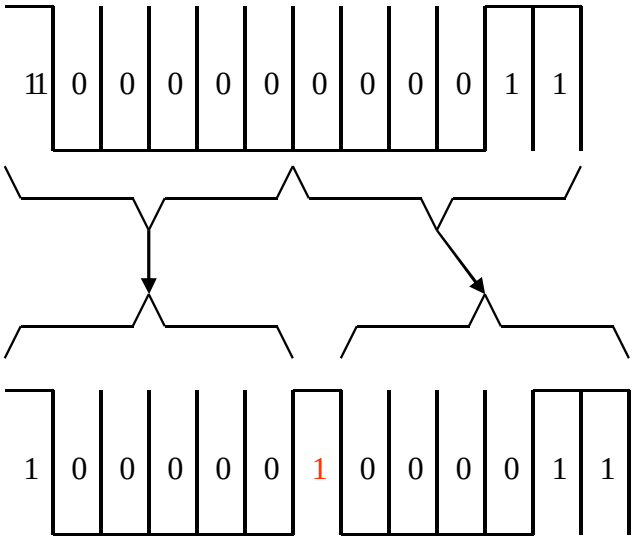
- Form error



- Bit error

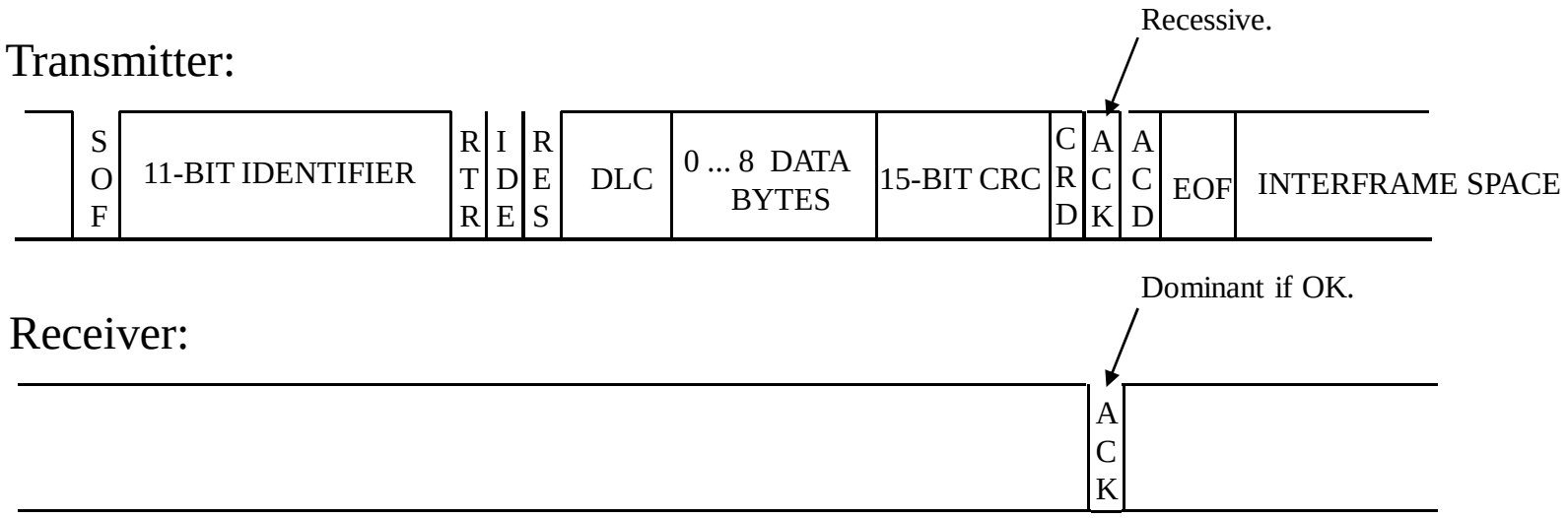
If a bit is written onto the bus and its compliment is **read back** a “Bit error” is generated (DOES NOT APPLY to IDENTIFIER or ACK-bit)

Bit stuffing error

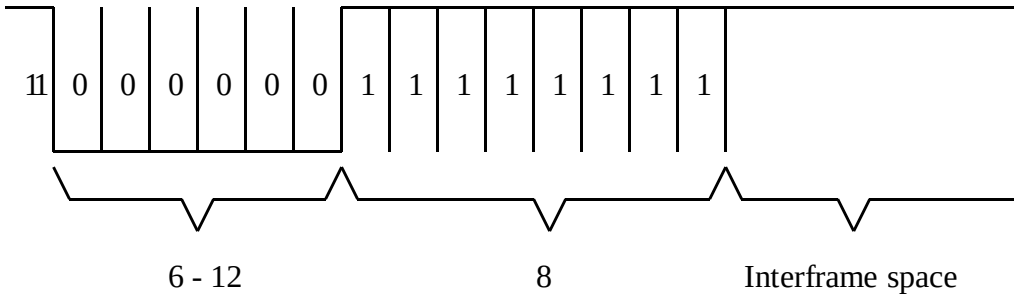


Acknowledgement check

CRC check and the acknowledge slot (”Form error”, “bit stuffing error”, ”CRC error”, ”Ack Error”)



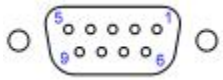
CAN error frame

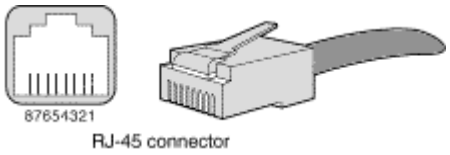


CAN controller error modes

- **Error active**
  Tx error counter <= 127 AND Rx error counter <= 127
- **Error passive**
  (Tx error counter > 127 OR Rx error counter > 127) AND Tx error counter <= 255.
- **Bus off**
  (Tx error counter > 255)

CAN connectors





http://www.erni.com/DB/PDF/M8M12/ERNI-M8M12-e.pdf

Pin #	Signal Names	Signal Description
1	Reserved	Upgrade Path
2	CAN_L	CAN Low
3	CAN_GND	Ground
4	Reserved	Upgrade Path
5	CAN_SHLD	Shield, Optional
6	GND	Ground, Optional
7	CAN_H	CAN Hign
8	Reserved	Upgrade Path
9	CAN_V+	Power, Optional

RJ45 Pin #	RJ10 Pin #	Signal Name	Signal Description
1	**2**	CAN_H	CAN High
2	**3**	CAN_L	CAN Low
3	**4**	CAN_GND	Ground
4	-	Reserved	Upgrade Path
5	-	Reserved	Upgrade Path
6	-	CAN_SHLD	CAN Shield, Optional
7	-	CAN_GND	Ground
8	**1**	CAN_V+	Power, Optional

One logic to several physical

Contact: sales@datalink.se

Rx error counter rules

• Receiver detects error (any): the Rx error counter will be increased by 1, except when the detected error was a bit error during the sending of an active error flag or an overload flag (=this specific node did not see the error that an other node saw).

• Receiver detects a dominant bit as the first bit after sending an error flag: the Rx error counter will be increased by 8.

• If a receiver detects a bit error (“what was written was not read”) while sending an active error flag or an overload flag the Rx error counter is increased by 8.

• *After the successful reception of a message (reception without error up to the acknowledge slot and the successful sending of the acknowledge bit), Rx error counter is decreased by 1 if it was between 1 and 127. If Rx error counter was 0 it stays 0, and if it was greater than 127, it will be set to a value between 119 and 127.*

Tx error counter rules

• When a transmitter sends an error flag, the Tx error counter is increased by 8. Important exception: If a node is the only one on the bus (or during start-up the only one that has become active), and it transmits a message, it will get an acknowledgement error, and will retransmit the message. This may lead to that node going to error passive mode – but it will not go bus off (=“oscillate”)

• If a transmitter detects a bit error while sending an active error flag or an overload flag, the Tx error counter is increased by 8.

• *After the successful transmission of a message (getting ack and no error until end of frame is finished) Tx error counter is decreased by 1 unless it was already 0.*

Advanced CANopen

- Multiplex PDO.
- Object Dispatcher List.
- Object Scanner List.

Multiplexed PDO

•Multiplexed PDOs are SDO/PDO hybrids for objects with a size of 1 – 32bits.

• Write to any OD entry (1-32 bits) on remote node without using SDO transfer.

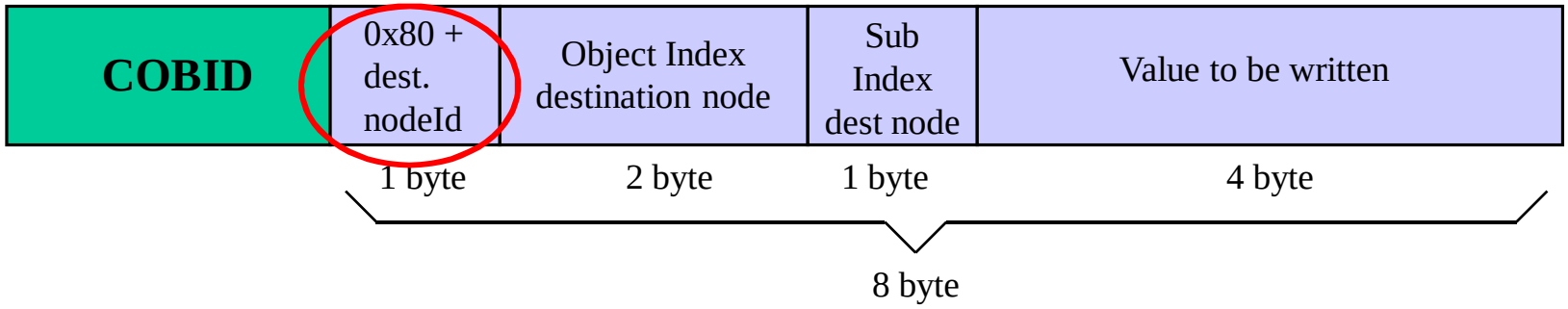
Multiplexed PDO types

- Destination Addressing Mode MPDO (DAM MPDO)

- Source Addressing Mode MPDO (SAM MPDO)

DAM MPDO

Destination Addressing Mode Multiplexed PDO



PDO mapping parameters RPDO.

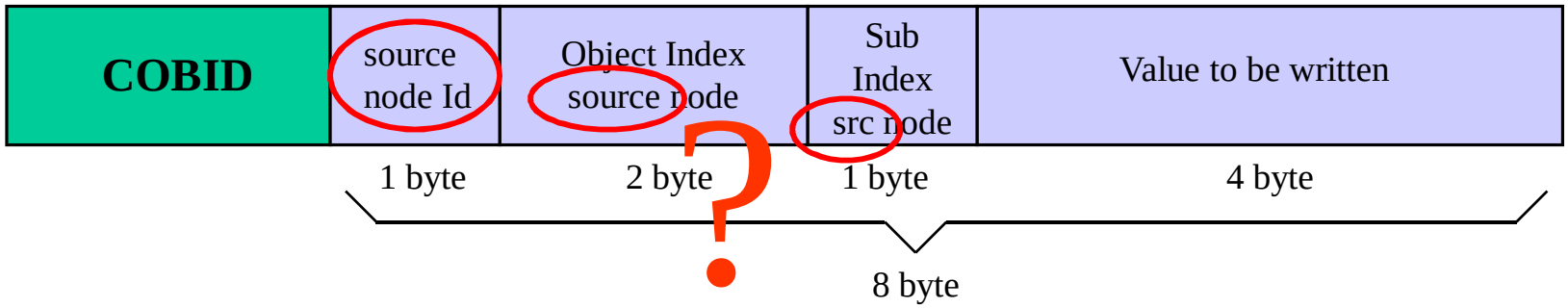
Object Index	Sub Index	Data Type	Value
1x1600 – 0x17ff	0 (no. obj to me mapped)	UINT8	255 (=DAM MPDO)
…	…	…	…
0x1a00 – 0x1bff	0 (no. obj to me mapped)	UINT8	255 (=DAM MPDO)

PDO mapping parameters TPDO.

Normal PDO:s are only 0..64

SAM MPDO

Source Addressing Mode Multiplexed PDO



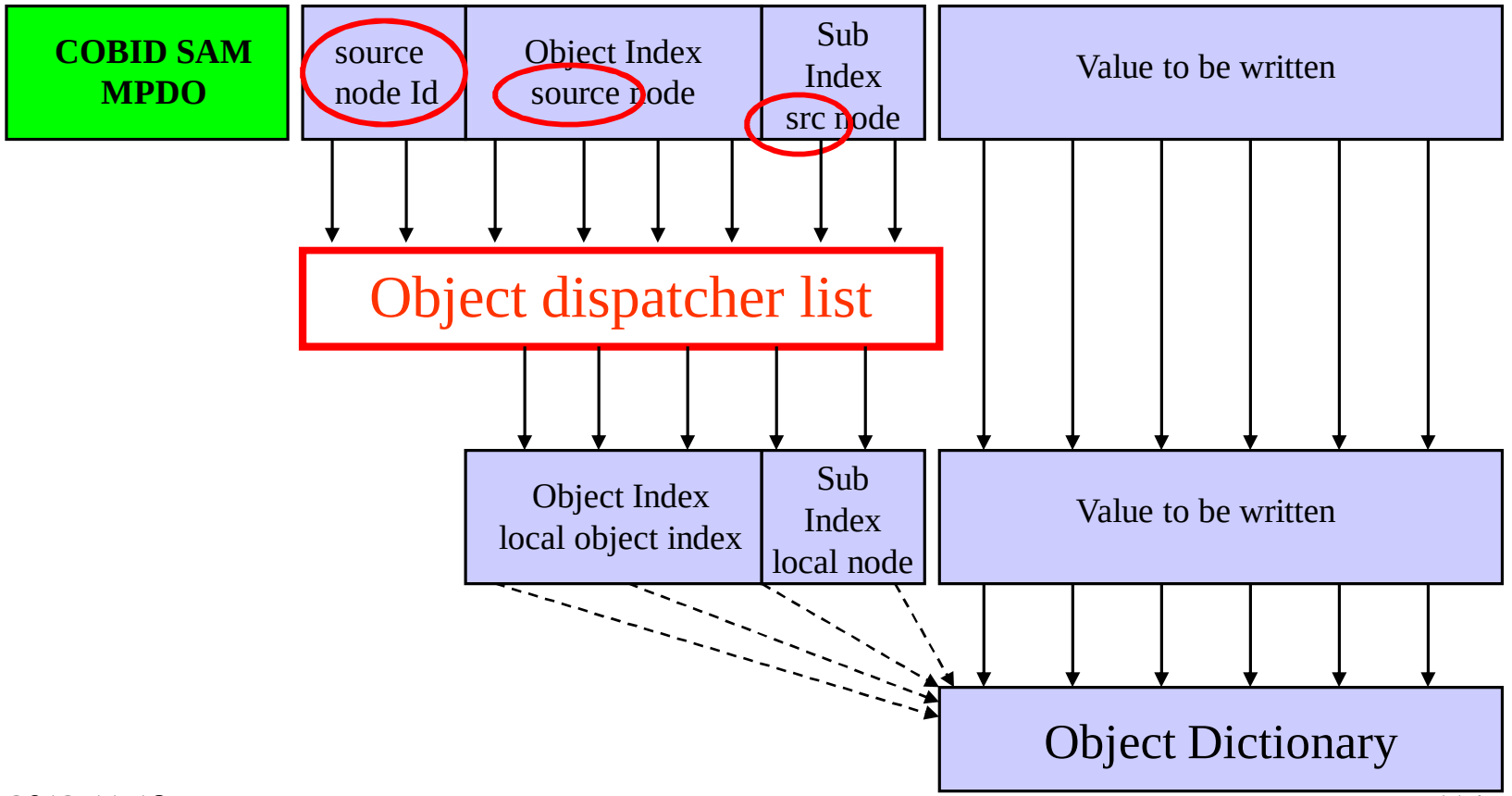
PDO mapping parameters RPDO.

Object Index	Sub Index	Data Type	Value
1x1600 – 0x17ff	0 (no. obj to me mapped)	UINT8	254 (=SAM MPDO)
…	…	…	…
0x1a00 – 0x1bff	0 (no. obj to me mapped)	UINT8	254 (=SAM MPDO)

PDO mapping parameters TPDO.

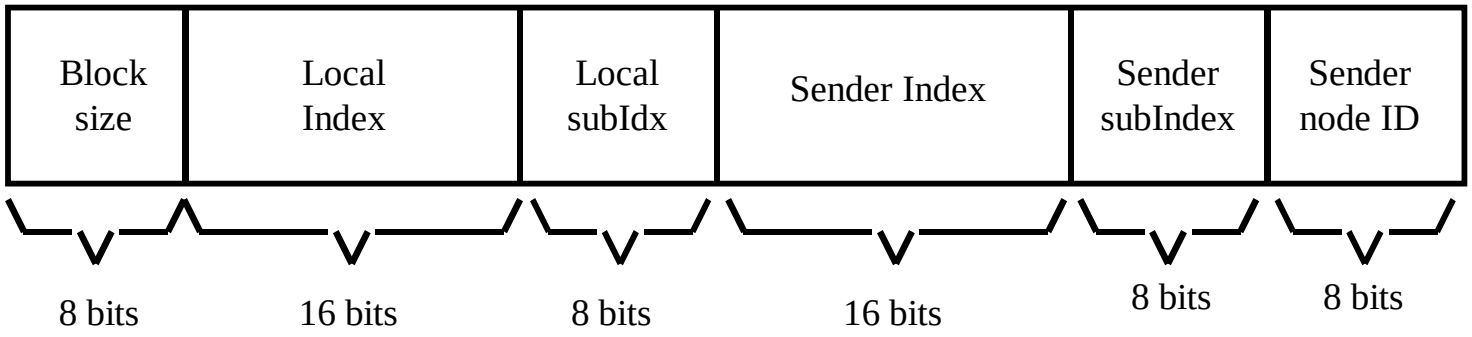
Normal PDO:s are only 0..64

Object Dispatcher List
(used when node receive a SAM-MPDO)

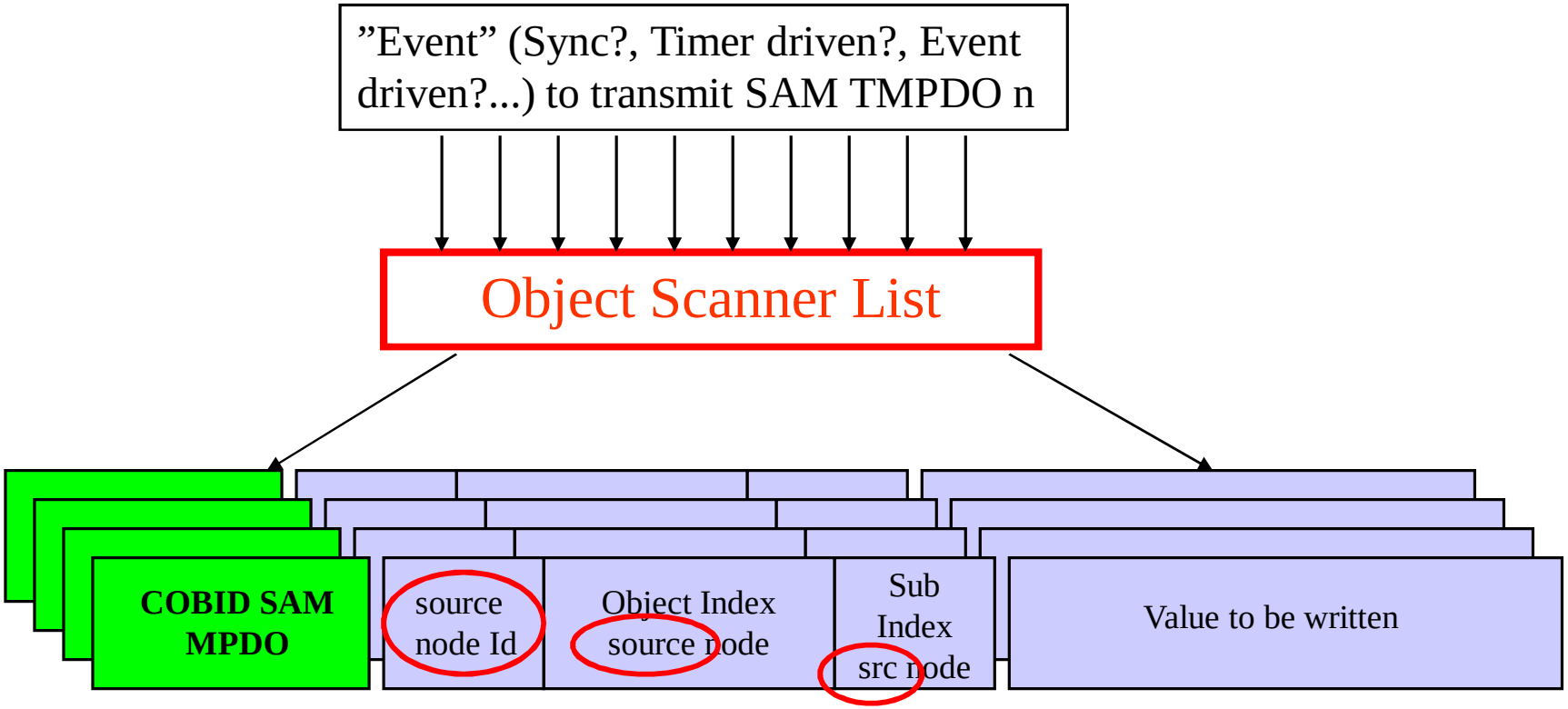


Object Dispatcher List
(object used for configuring)

Object Index	Sub Index	Data Type	Description
0x1fd0 – 0x1fff	0	UINT8	Number of configured dispatchers
	0x1	UINT64	Object Dispatching 1
	-- 0xfe	UINT64	Object Dispatching ...254

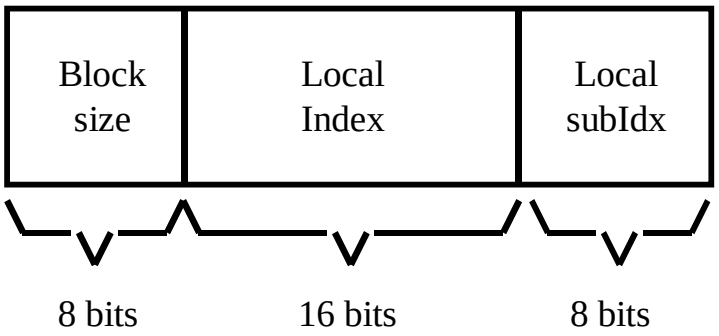


Object Scanner List
(used when node transmits SAM-MPDO)



Object Dispatcher List
(object used for configuring)

Object Index	Sub Index	Data Type	Description
0x1fd0 – 0x1fff	0	UINT8	Number of configured dispatchers
	0x1	UINT32	Object Dispatching 1
	-- 0xfe	UINT32	Object Dispatching ...254



Transmission Type

Object Index	Sub Index	Data Type	Bit contents
0x1801	1 (COBID)	UINT32	0x123
0x1801	2 (Transmission time)	UINT8	254
0x1801	3 (Inhibit time)	UINT16	100 (*10us)
0x1801	4 ()	0	0
0x1801	5 (Event timer)	UINT16	1000 (*1ms)

Transmission Type	Meaning for a transmit PDO	Meaning for a receive PDO
0	Sent on next SYNC if event or request has been made.	Application updated on next SYNC.
1 < n < 240	Sent on every n SYNC	Application updated on next SYNC.
241 <= n < 252	UNDEFINED	UNDEFINED
252	Sent on next SYNC if PDO has been requested.	UNDEFINED
253	Sent independent of SYNC upon request.	UNDEFINED
254 -255	Sent independent of SYNC in all cases	Application is updated upon reception of PDO