Standard Protokolle und Datenformate in der KFZ-Diagnose

Sammelsurium von Infos rund um Automotive Diagnose als Startpunkt zum Weitergooglen.

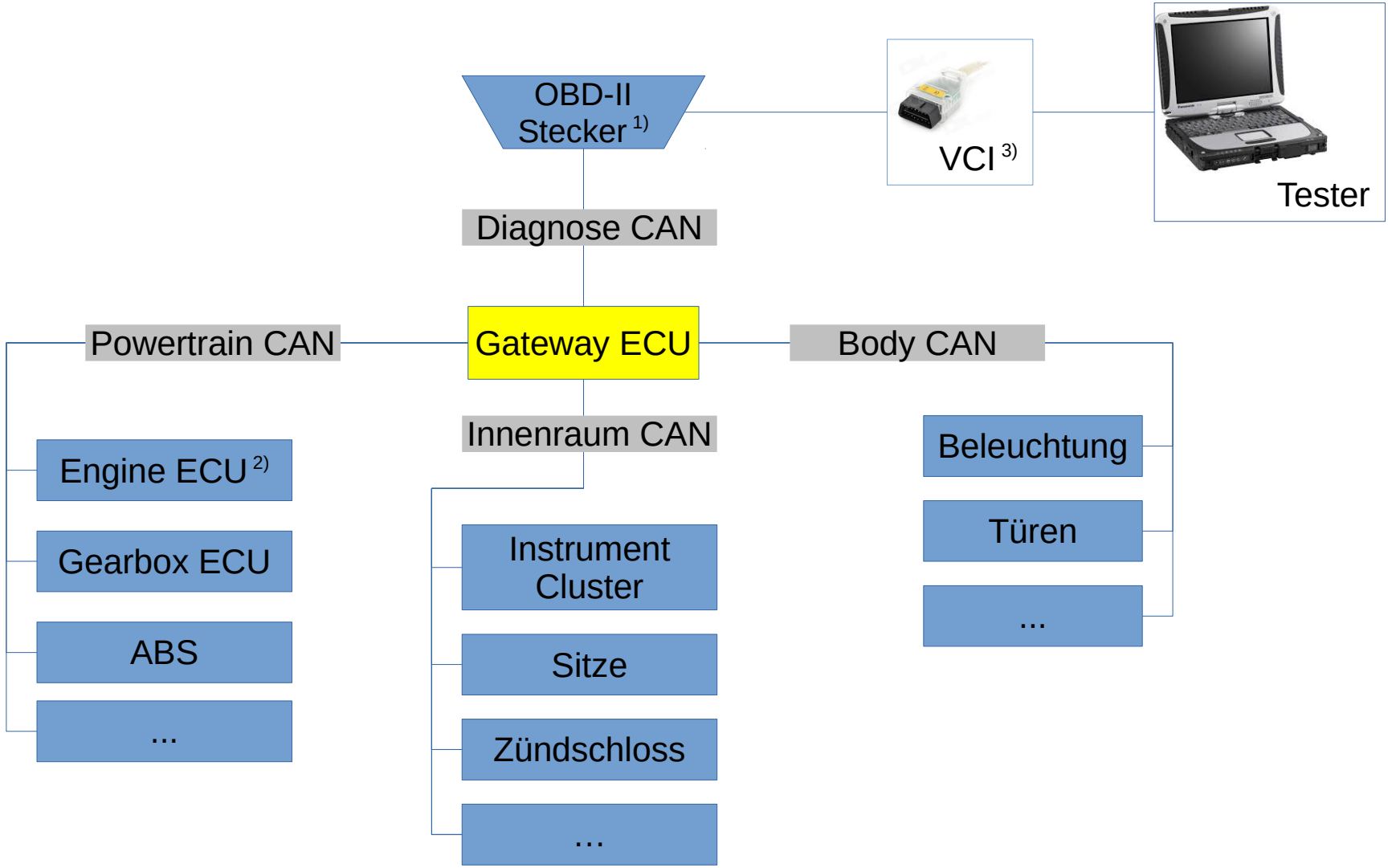
Disclaimer

- Die Folien enthalten
  - Keine Infos die mir nur über meinen Arbeitgeber zugänglich sind
  - Keine Kundenspezifischen Informationen
- Experimentieren auf eigene Gefahr

Inhalt

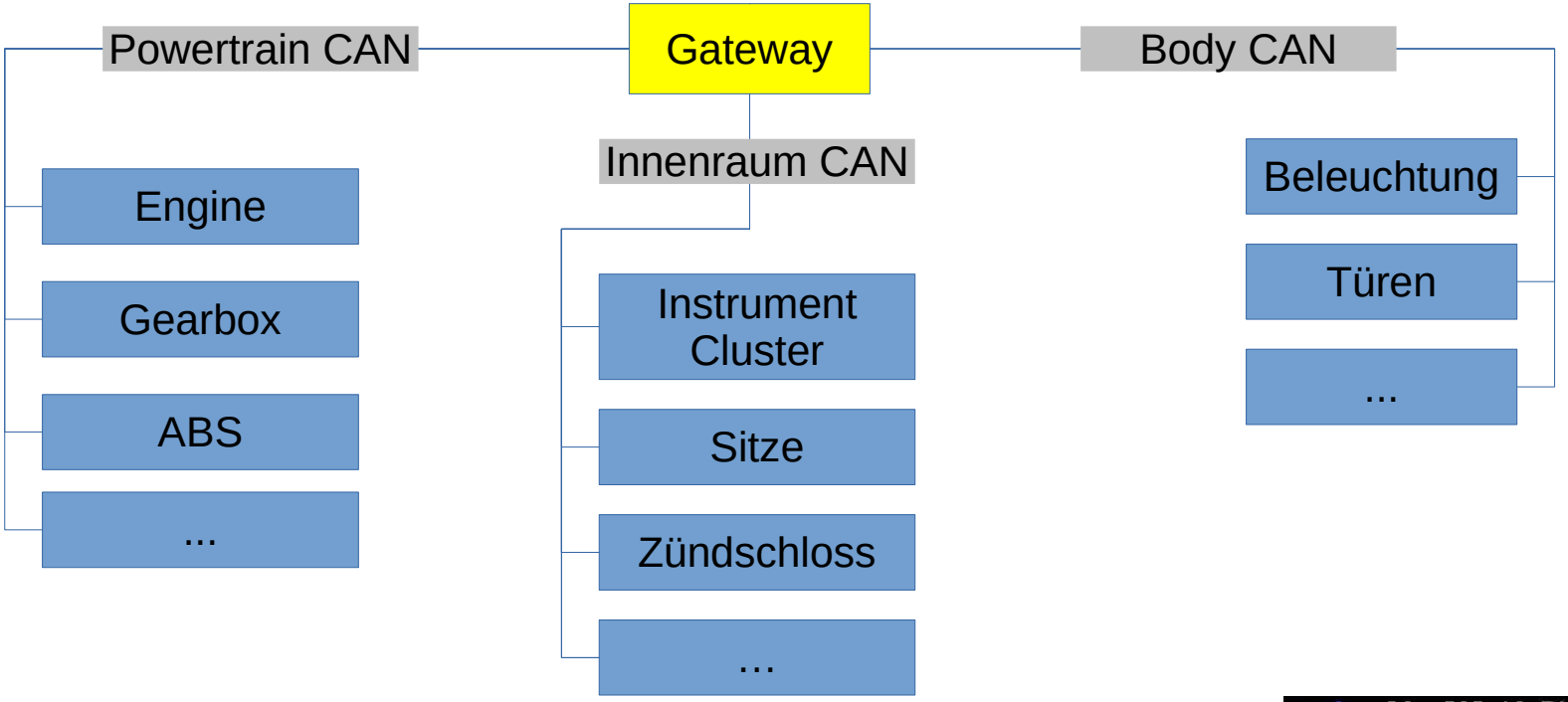
1) CAN-Signale
2) Onboard Diagnose (OBD & UDS)
3) UDS ohne Daten
4) Standards im Werkstatt-Tester

Typische Fahrzeugtopologie



1) J1962 – Stecker
2) **ECU** = Steuergerät (Electronic Control Unit)
3) **VCI** = Vehicle Communication Interface

1) CAN-Signale (Normalzustand)

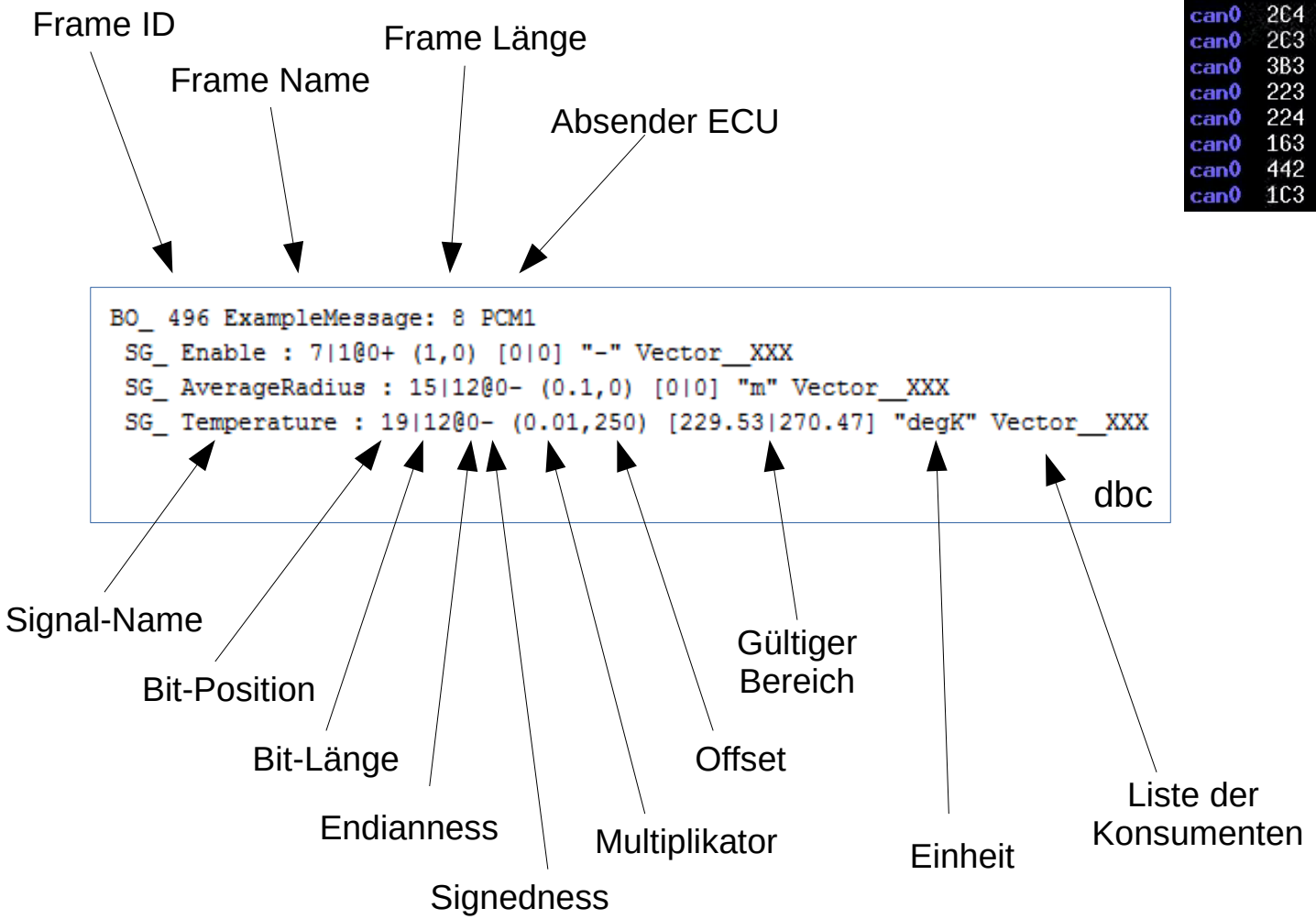



```
can0   B2  [6] 03 50 02 D0 11 0A
can0   B4  [8] 00 00 00 00 9C 03 37 92
can0  2C4  [8] 05 54 00 19 00 00 92 D2
can0  2C3  [8] 10 00 00 04 69 2D 33 1C
can0  3B3  [3] 05 58 2E
can0  223  [8] 40 00 00 00 00 00 00 6D
can0  224  [8] 00 00 00 00 00 00 00 00
can0  163  [5] 0F 3B 01 00 40
can0  442  [8] 40 02 00 00 00 00 00 00
can0  1C3  [1] 04
```

- **Zyklische Broadcasts** von CAN Signalen per Raw-CAN Frames (ISO 11898)
  - Payload max. 8-bytes, Adressen 11- oder 29-bit lang
  - **Ungerichtete** Kommunikation
- Ggf. noch Kommunikation per SAE-J1939 (eher Nutzfahrzeuge)
- Gateway leitet Frames zwischen den CAN-Bussen weiter (unterschiedliche Baudraten)
  - Weiterleitung gemäß **VMM** (Vehicle Message Matrix) aka. **K-Matrix**
  - Definiert üblicherweise im Vector **.dbc** Format (wird aber nicht veröffentlicht)

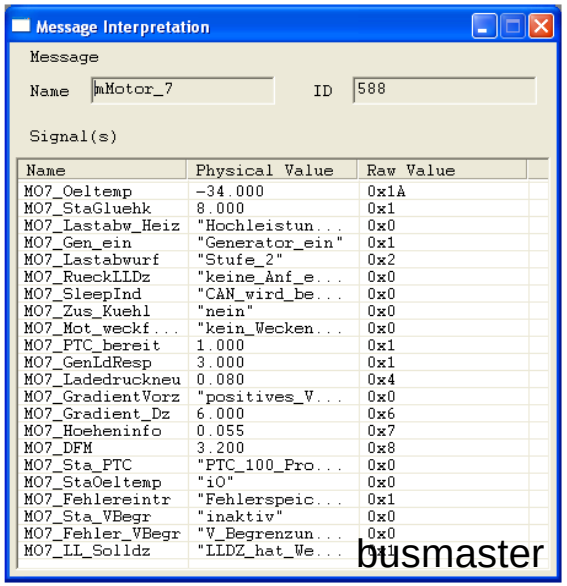
1) CAN-Signale (VMM im DBC Format)

```
can0   B2  [6] 03 50 02 D0 11 0A
can0   B4  [8] 00 00 00 00 9C 03 37 92
can0  2C4  [8] 05 54 00 19 00 00 92 D2
can0  2C3  [8] 10 00 00 04 69 2D 33 1C
can0  3B3  [3] 05 58 2E
can0  223  [8] 40 00 00 00 00 00 00 6D
can0  224  [8] 00 00 00 00 00 00 00 00
can0  163  [5] 0F 3B 01 00 40
can0  442  [8] 40 02 00 00 00 00 00 00
can0  1C3  [1] 04
```



1) CAN-Signale (Tools)

- **Busmaster** (GPL) als alternative zu Vector CANoe (~10k€)
  - Kann .dbc importieren
  - Unterstützt einer Reihe von CAN-Hardware
  - Nicht gerade ein Performance-Wunder aber dafür frei



- Für DYI Projekte kann man gut auf **SocketCAN** basieren (z.B. rPI mit einem CAN-Shield)
  - hier gibt's dann Tools wie **cansend/candump**
  - oder man nimmt tcpdump etc.

```
root@raspberrypi:/home/pi/can-test# ./candump can0
  can0  7DF   [8] 02 01 05 00 00 00 00 00
  can0  7E8   [8] 03 41 05 FF 00 00 00 00
  can0  7DF   [8] 02 01 05 00 00 00 00 00
  can0  7E8   [8] 03 41 05 FF 00 00 00 00
  can0  7DF   [8] 02 01 05 00 00 00 00 00
  can0  7E8   [8] 03 41 05 FF 00 00 00 00
  can0  7DF   [8] 02 01 05 00 00 00 00 00
  can0  7E8   [8] 03 41 05 EA 00 00 00 00
  can0  7DF   [8] 02 01 05 00 00 00 00 00
  can0  7E8   [8] 03 41 05 E1 00 00 00 00
  can0  7DF   [8] 02 01 05 00 00 00 00 00
  can0  7E8   [8] 03 41 05 C9 00 00 00 00
  can0  7DF   [8] 02 01 05 00 00 00 00 00
  can0  7E8   [8] 03 41 05 C9 00 00 00 00
  can0  7DF   [8] 02 01 05 00 00 00 00 00
  can0  7E8   [8] 03 41 05 C4 00 00 00 00
  can0  7DF   [8] 02 01 05 00 00 00 00 00
```

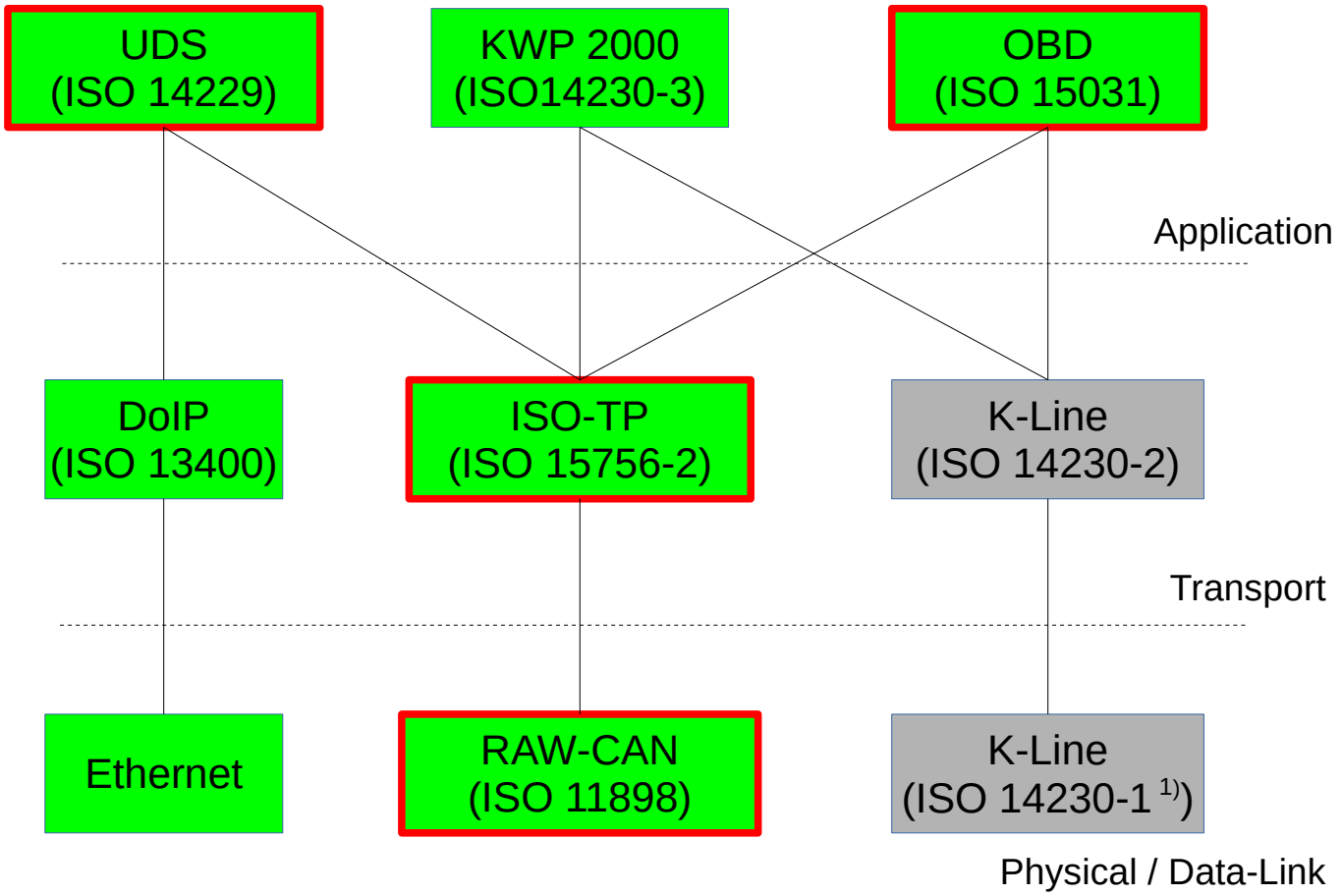
1) CAN-Signale (Zusammenfassung)

- Primäre Kommunikation im Fahrzeug,

  ABER:

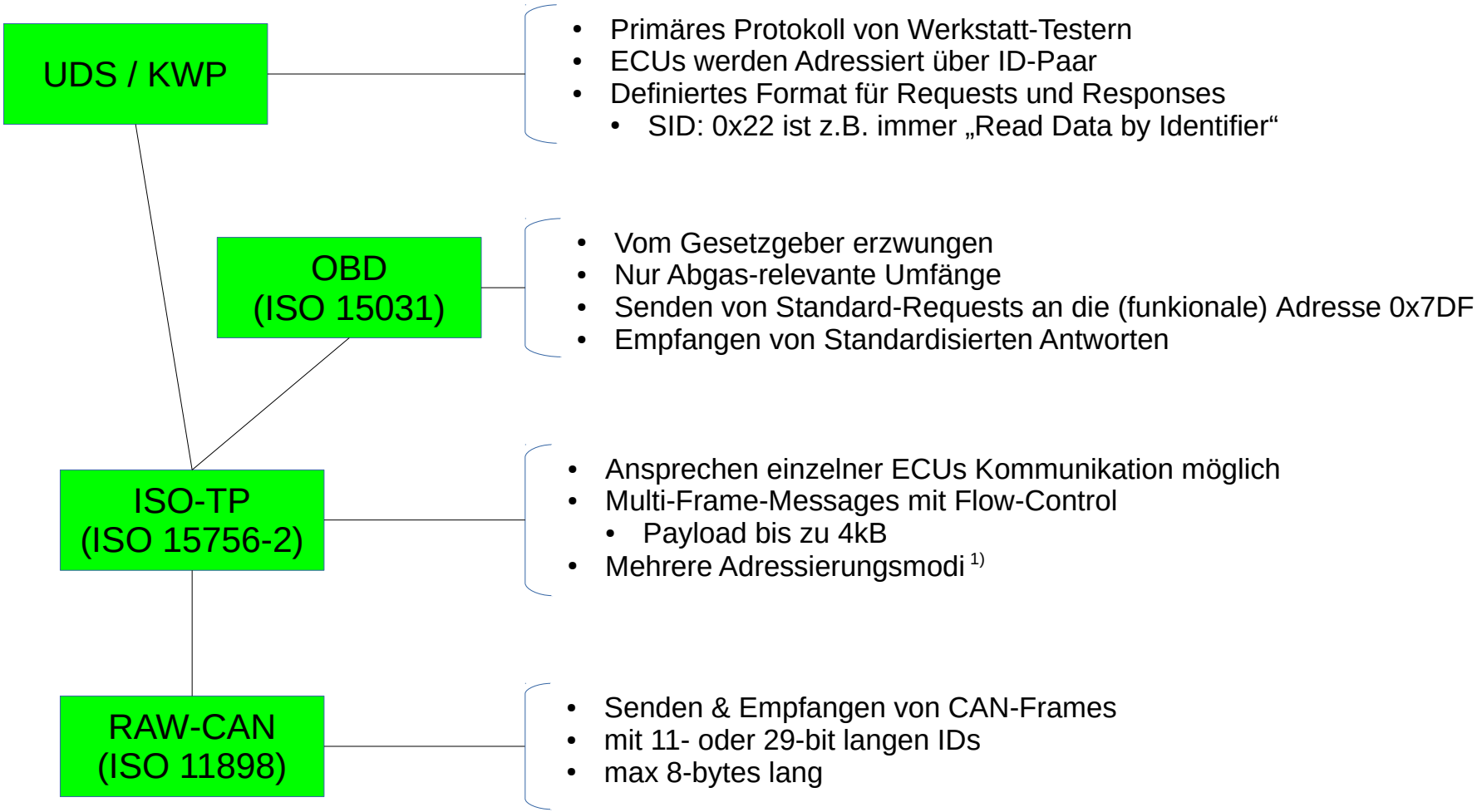
  - man sieht davon normal nichts auf dem Diagnose-Bus (OBD-Stecker) und ist somit schwer zugänglich
  - schwierig zu reversen, da keinerlei Struktur in den CAN-Frames (außer man kommt an die VMM)

2) Diagnose On-Board (Übersicht)



1) ISO 14230-1 in etwa gleich ISO 9141-2

2) Protokolle



1) Format bei 29-Bit IDs lassen hier faktisch nur 256 Adressen zu (0x18DAxxF1)

2) OBD Beispiel (Kühlmitteltemp)

- OBD Request: `01 05`
  - 0x01 = Show current data
  - 0x05 = PID für Coolant temperature (in °C – 40) (siehe Wikipedia)
- OBD Response: `41 05 64`
  - 0x41 = Positive response (0x01 + 0x40)
  - 0x64 = 100 = 60 °C

Beispiel anhand ELM327

- `AT SP 0`
  Protokoll setzen: 0=AUTO
  (oder z.B. 6 für 11-bit ISO-TP)
- `01 05`
  Request senden

CAN-Trace:

```
can0  7DF   [8] 02 01 05 CA FE BA BE 01
can0  7E8   [8] 03 41 05 64 AA AA AA AA
```

PIDs: https://en.wikipedia.org/wiki/OBD-II_PIDs
Examples: http://www.obdsol.com/knowledgebase/obd-software-development/reading-real-time-data/
ELM327 Doku: http://elmelectronics.com/DSheets/ELM327DS.pdf

2) UDS Beispiel (Fehlerspeicher lesen)

- UDS Request auf CAN-ID `0x723`:
  `19 02 FF`
  - 0x19 0x02 = „Read-DTC by status“*
  - 0xFF = alle

- UDS Response auf CAN-ID `0x542`:
  `59 02 AA 12 34 56 08`
  - 0x59 = Positive Response (0x19 + 0x40)
  - 0xAA = Unterstützte Status-Bits
  - 0x123456 = Fehlercode
  - 0x08 = Status

Beispiel anhand ELM327

- `AT SP 6`
  Protokoll setzen: 0=ISO-TP 11-bit
- `AT SH 723`
  Request ID setzen
- `AT CRA 542`
  Response ID setzen
- *`AT FC SH 723`*
  `AT FC SD 30 00 00`
  `AT FC SM 1`
  Möglicherweise nötig für multi-frame Support (definiert flow-control frames)
- `19 02 FF`
  Request senden

CAN-Trace:

```
can0  723   [8] 03 19 02 FF F0 00 BA AA
can0  542   [8] 07 59 02 AA 12 34 56 08
```

UDS: https://de.wikipedia.org/wiki/Unified_Diagnostic_Services
Etwas genauer (unvollständig): http://unifieddiagnosticservices.blogspot.de/2011/12/automotive-diagnostics-services.html
ELM327 Doku: http://elmelectronics.com/DSheets/ELM327DS.pdf
* DTC = Diagnostic Trouble Code (Fehlerspeicher-Eintrag)

2) UDS / KWP Use-Cases

- Main Use-Cases
  - Fehlerspeicher lesen/löschen
  - Werte lesen (Statisch/Dynamisch)
  - Ansteuern
  - Werte schreiben (Kodierung)
  - Flashen
- Safety measures
  - Session handling
  - Interne Precondition-Checks
- Security measures
  - „Security-Access“:
    Seed/Key checks zum entsperren diverser Funktionen
  - Singnaturprüfung von Firmware

3) Möglichkeiten: UDS ohne Daten

- Alle CAN-IDs durchprobieren und auf antworten Ausschau halten und Request-ID <-> Response-ID map erstellen
  - Suche vermutlich auf Raw-CAN-layer
  - Mögliche Adressen:
    - 11-bit-CAN: 0x000 - 0x7FF
    - 29-bit-CAN: 0x18DA**00**F1 - 0x18DA**FF**F1
    - 11-bit-CAN mit NAE: theoretisch 19-bit, vermutlich weniger
  - UDS-Service, z.B.: Tester-Present (3E 00) *

```
...
218.402154  can0  73B  [8] 02 3E 00 00 00 00 00 00
218.602477  can0  73C  [8] 02 3E 00 00 00 00 00 00
218.802799  can0  73D  [8] 02 3E 00 00 00 00 00 00
219.003107  can0  73E  [8] 02 3E 00 00 00 00 00 00
219.151510  can0  7B2  [8] 02 7E 00 AA AA AA AA AA
219.203436  can0  73F  [8] 02 3E 00 00 00 00 00 00
219.326518  can0  7B3  [8] 02 7E 00 AA AA AA AA AA
219.403773  can0  740  [8] 02 3E 00 00 00 00 00 00
219.604091  can0  741  [8] 02 3E 00 00 00 00 00 00
...
```

  - Logger basteln und in die Werkstatt fahren

* Unterscheidungsmöglichkeit UDS/KWP z.B.:
https://github.com/rbei-etas/busmaster/issues/731

3) Möglichkeiten: UDS ohne Daten

- Versuchen Steuergeräte zu identifizieren (UDS-Layer)
  z.B.
  - DTCs Provozieren und lesen (19 …)

```
ECU[732/78C], Req:19020C -> Resp: 590299
ECU[733/78F], Req:19020C -> Resp: 5902990386A208
ECU[735/792], Req:19020C -> Resp: 590219
ECU[742/79B], Req:19020C -> Resp: 590299
ECU[746/79E], Req:19020C -> Resp: 590299
ECU[749/7A9], Req:19020C -> Resp: 590219
...
```

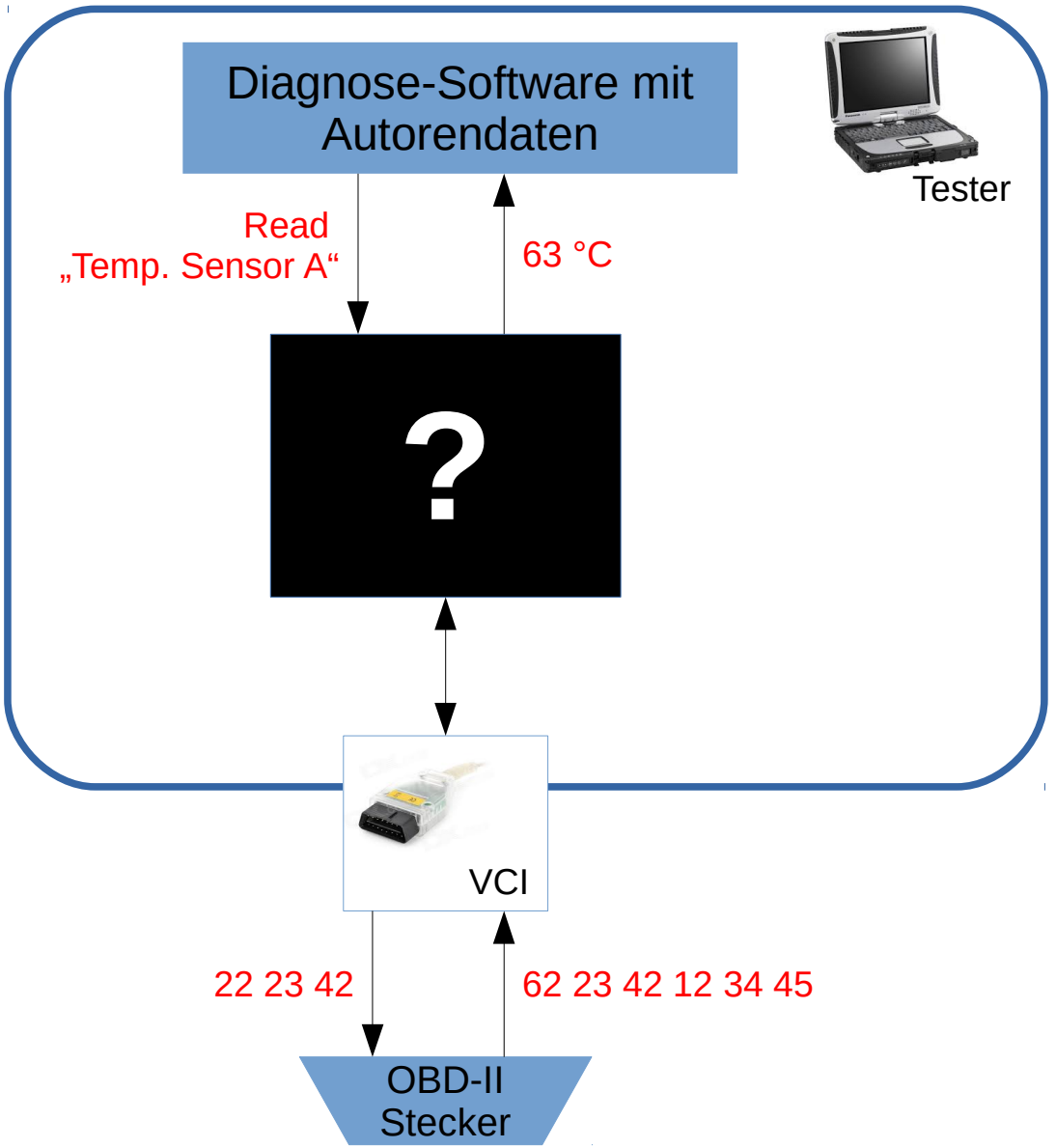
  - Werte lesen und vergleichen (22 xx yy)

```
...
ECU[735/792], Req:220000 -> Resp: 7F2231 *
ECU[735/792], Req:220001 -> Resp: 7F2231
...
ECU[735/792], Req:220122 -> Resp: 620122F00BAA
ECU[735/792], Req:220123 -> Resp: 620123123456
...
```

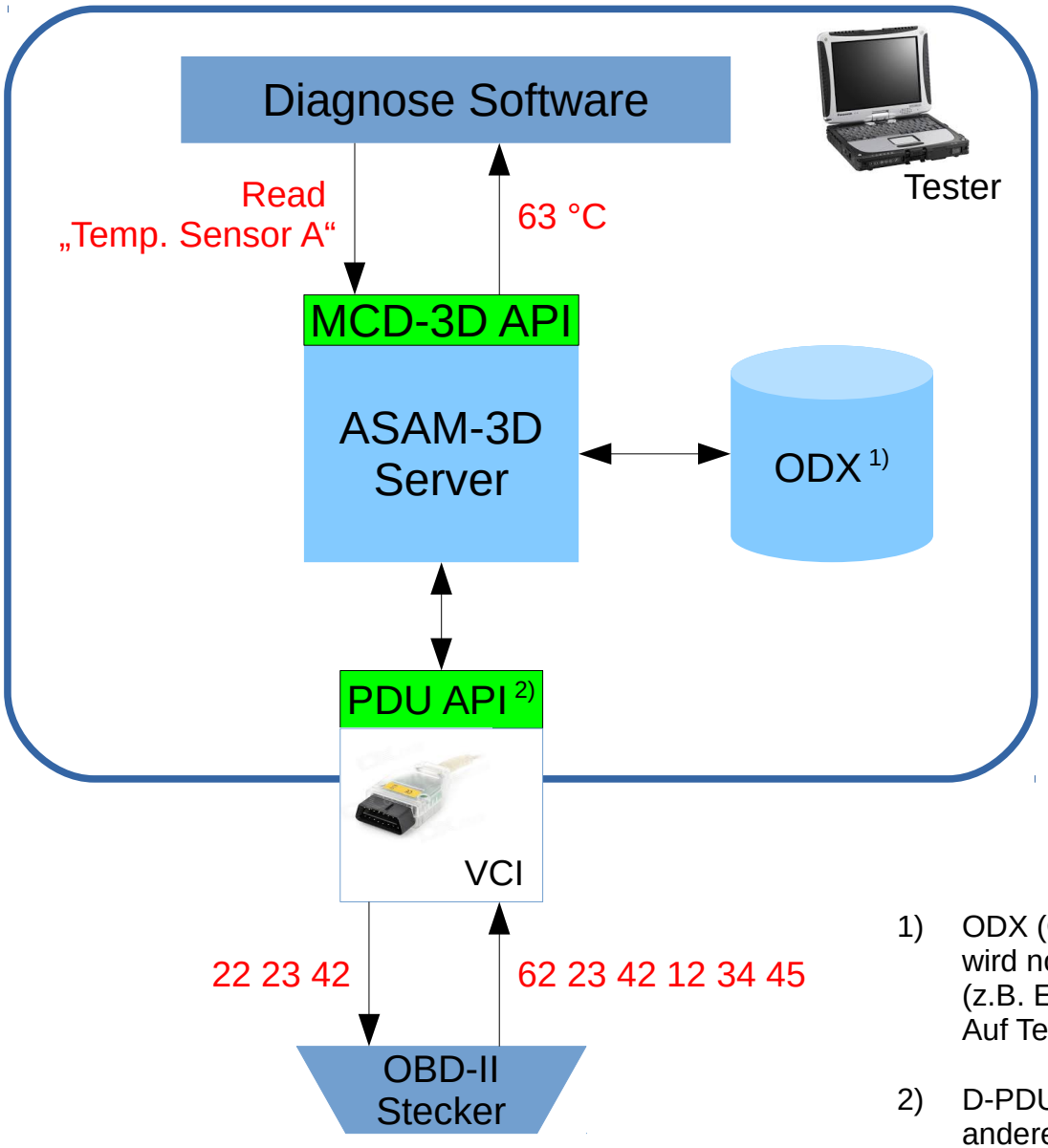
  - ECU-Reset und beobachten (11 xx)

* 0x7F = „Negative Response“
  0x31 = „Request Out Of Range“

4) Standards im Werkstatt-Tester



4) Standards im Werkstatt-Tester

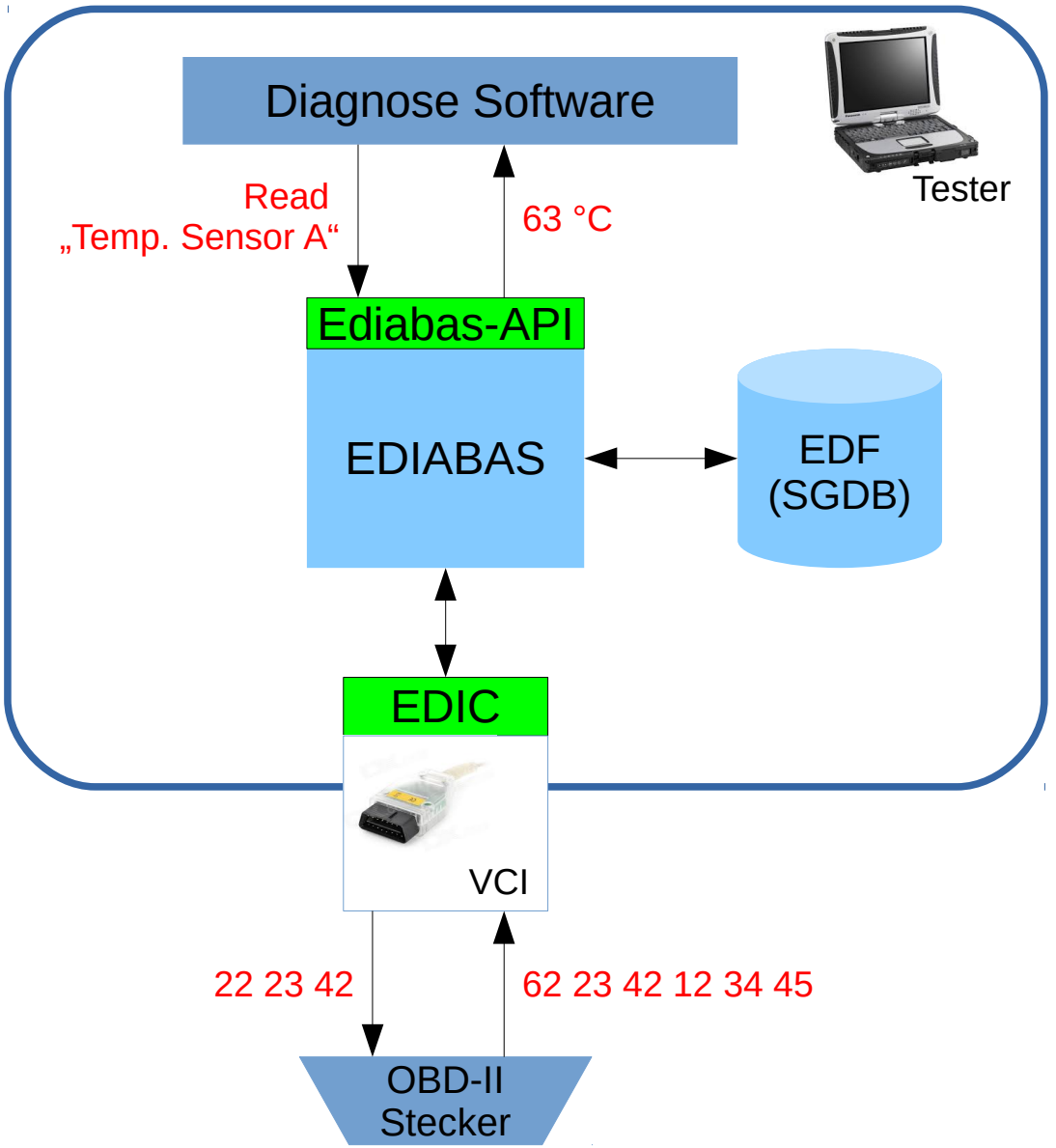


- Beispiel
  ASAM-3D Server

1) ODX (Open Diagnostic Data Exchange aka. ISO-22901)
wird normal nicht per Hand geschrieben
(z.B. Export aus CandelaStudio)
Auf Testern liegt das ODX normal serialisiert/verschlüsselt

2) D-PDU API aka ISO-22900
andere Standards hier sind J2534 (Passthru), RP1210

4) Standards im Werkstatt-Tester



- Beispiel
  Softing Ediabas

Weitere Links

Bei der Vorberitung drüber gestolpert

- recht nette Zusammenfassung
  http://www.emotive.de/documents/WebcastsProtected/Transport-Diagnoseprotokolle.pdf
- UDS Cheat-Sheet (mit Standard-Fehlern, etc.)
  http://automotive.softing.com/fileadmin/sof-files/pdf/de/ae/poster/uds_info_poster_v2.pdf
- ISO-TP explained
  http://www.canbushack.com/blog/index.php?title=iso-15765-2-can-transport-layer-yes-it-can-be-fun&more=1&c=1&tb=1&pb=1
- CAN-Monitoring in einem Toyota (per socket-can)
  https://fabiobaltieri.com/2013/07/23/hacking-into-a-vehicle-can-bus-toyothack-and-socketcan/