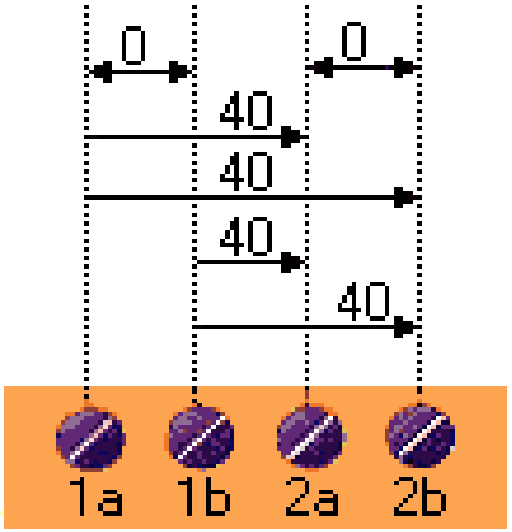
DSL / Netzmanagement – 7. Übung

Nachtrag aus den letzten Übungen

Ethernet & ISDN

Stromversorgung ISDN

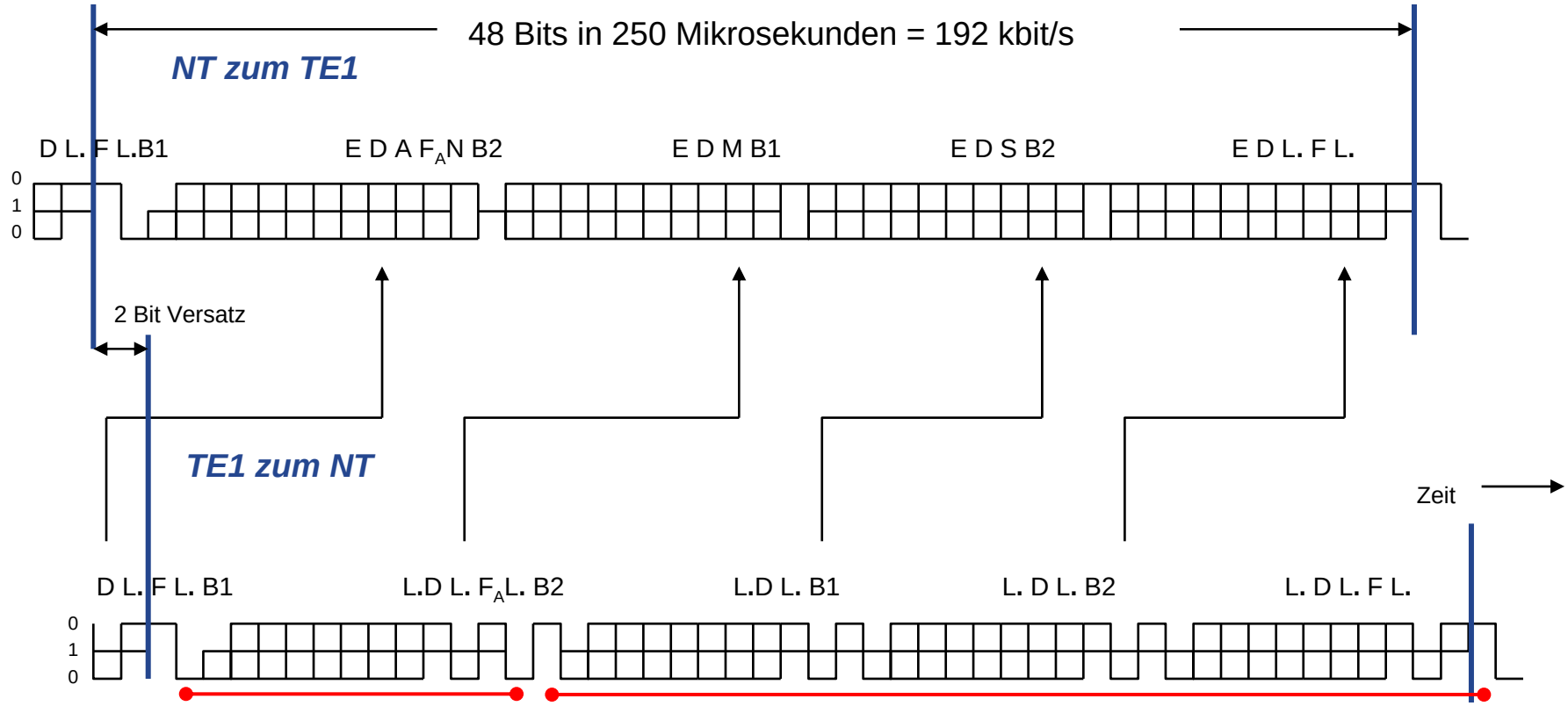
- Stromversorgung über den S_0-Bus
  - Zwischen zwei Adern derselben Leitung (Richtung) liegt keine Spannung an
  - Zwischen zwei Adern verschiedener Leitungen liegen 40 V an.
  - Signal selbst wird über Hochfrequenz-übertrager (eine Art Transformator) eingespeist
  - Transformatoren benötigen Wechselstrom / Polarisationsänderungen um Auswirkungen zu verursachen



Coderegelverletzungen ISDN

- Codier-Vorschrift
  - Rahmen beginnt mit positiv codiertem F-Bit und passendem L-Ausgleichsbit
  - Zwischen zwei Ausgleichsbits wird die erste Null immer mit einem negativen Puls codiert
    - Die letzte logische Null innerhalb eines Rahmens ist dadurch immer positiv codiert und verursacht mit dem darauf folgenden F-Bit des nächsten Rahmens eine Coderegelverletzung
    - Falls in den Kanälen keine Nullen codiert werden tritt diese Verletzung mit dem L-Bit nach dem F_A-Bit auf
  - Die zweite CV tritt zwischen dem auf dem F-Bit folgenden L-Bit und der nächsten codierten Null auf. Spätestens jedoch mit dem F_A-Bit

S_0-Rahmen



A	=	Bit für Aktivierungsprozedur
B1, B2	=	B-Kanäle
D	=	D-Kanal („0“ überschreibt „1“)
E	=	Bit für D-Echo-Kanal
F_A, F	=	Zusätzliches Rahmenbit (=0)

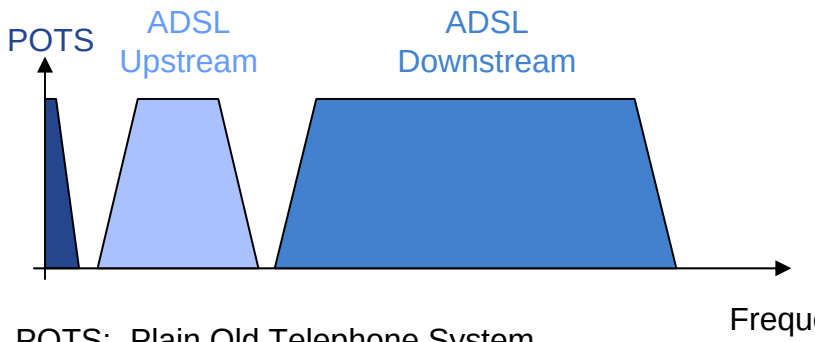
.	=	zwischen je 2 Punkten (.) ist der Rahmen gleichstromfrei
L	=	Gleichstrom-Ausgleichsbit (positive Anzahl Pulse)
N	=	F_A logisch negiert
S	=	S-Bit für S-Kanal (Schleifentests)
M	=	M-Bit für Mehrfachrahmen

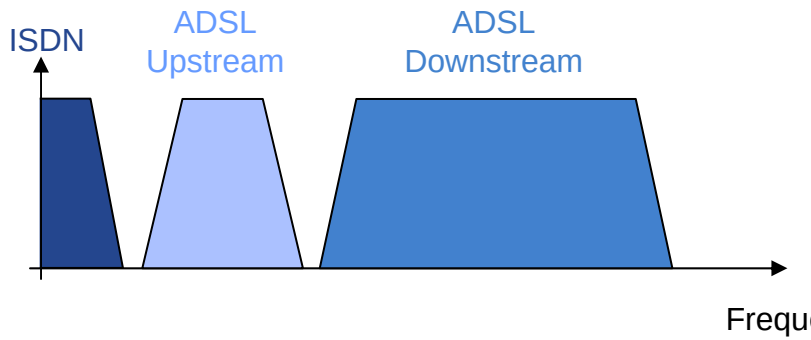
Aufgabe 1

DSL Übertragungstechnik

Rückblick: ADSL Übertragungstechnik

- Einsatz von Frequenzmultiplex
  - Nutzung der unteren Frequenzen für Telefonsignal
    - Annex A
      - Es werden nur 3 kHz für analoge Telefonie (POTS) benötigt
    - Annex B
      - Es werden 120 kHz für ISDN-Telefonie benötigt
      - Maximaler Downstream in diesem Fall nur 6 Mbit/s (statt 8 Mbit/s)
  - Höhere Frequenzen werden für Datensignal genutzt
    - Die verbleibenden Frequenzen bis 1,104 MHz werden für ADSL Upstream und Downstream genutzt
    - Annex B: 138-276 kHz für Upstream, 276-1104 kHz für Downstream

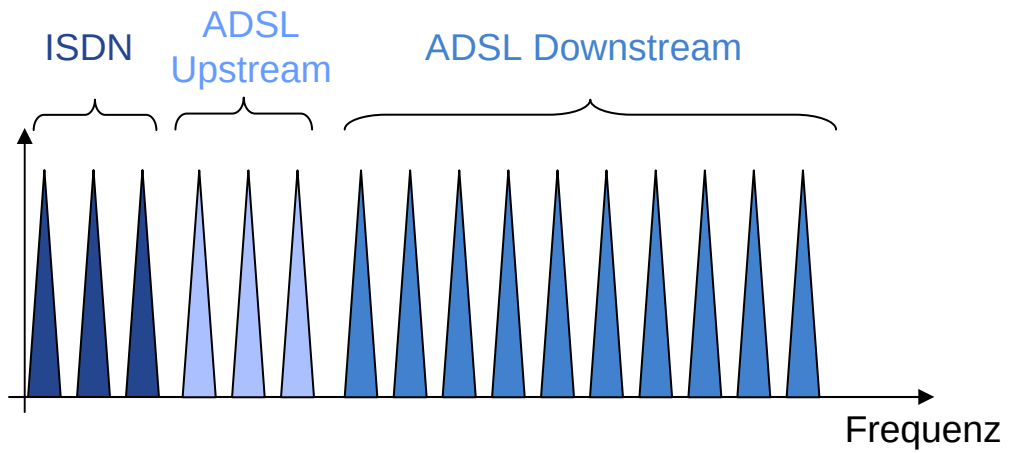




POTS: Plain Old Telephone System
ISDN: Integrated Services Digital Network

Rückblick: ADSL Übertragungstechnik

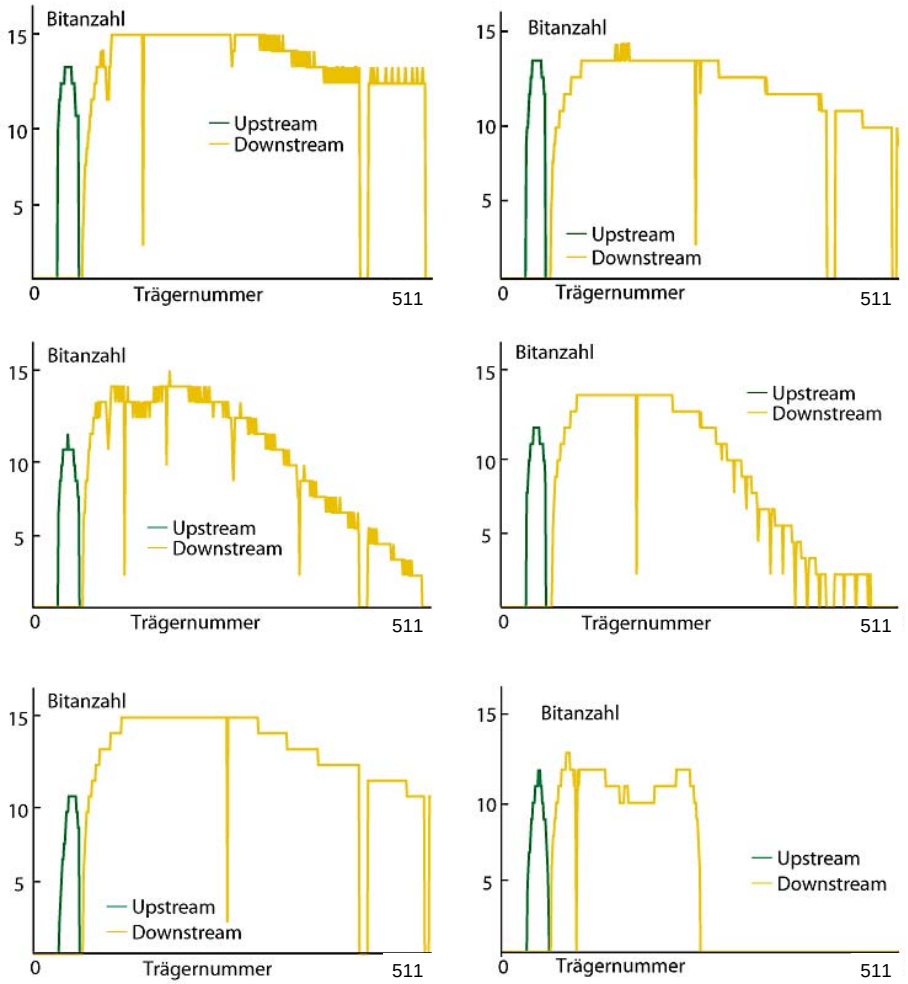
- Einsatz von Discrete Multitone (DMT) zur Modulation
  - Aufteilung des gesamten Frequenzbereichs in Menge von Trägern
    - Wenig störanfällig
      - Auf häufig gestörten Trägern werden weniger Bits übertragen
    - Vernachlässigbare Laufzeitverzerrung
    - Jeder Träger belegt eine Bandbreite von 4,3125 kHz
      - Bei Annex B werden jeweils 32 der 256 Träger für ISDN und für ADSL Upstream benötigt
    - Ausmessen der Träger zu Beginn der Übertragung ermöglicht Anpassung an jeweilige Umgebung



Rückblick: ADSL Übertragungstechnik

- Ausmessen der Träger zu Beginn der Übertragung

- Jeder Subkanal wird vermessen und (theoretisch) bestmöglich ausgenutzt
- ADSL2+: Nachmessen während Verbindung möglich
- Reale Beispiele:



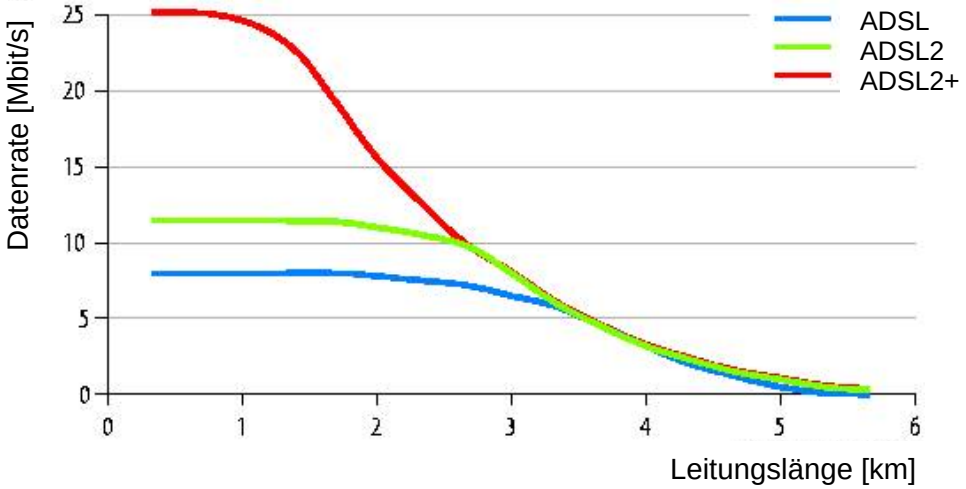
- ADSL2+:
  24 Mbit/s ↓
  1 Mbit/s ↑
- Praxis etwa
  16 Mbit/s ↓
  erzielbar

[ZiEA06]

Aufgabe 2

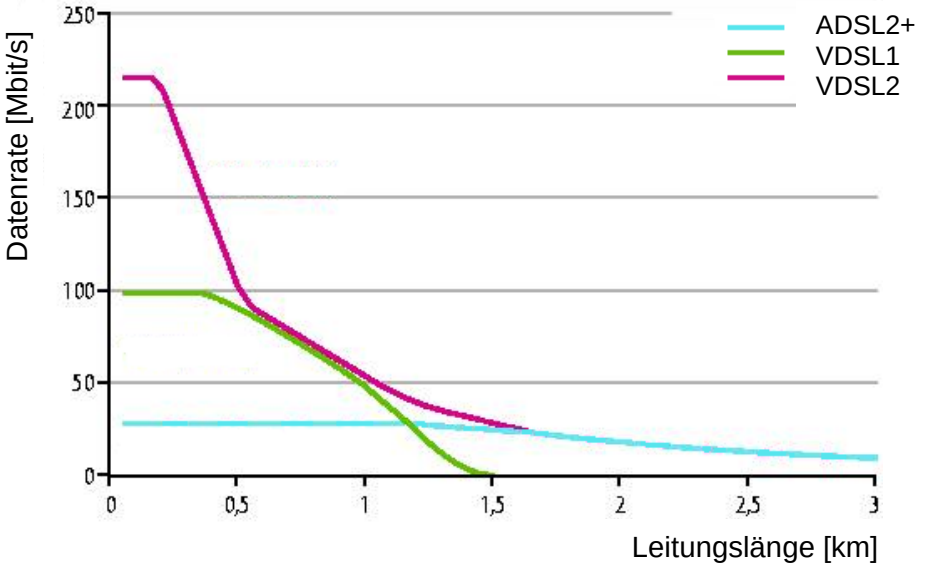
Weiterentwicklungen von DSL

Weiterentwicklungen



	Frequenz	Datenrate*
ADSL	1,104 MHz	8
ADSL2	1,104 MHz	12
	(verbesserte Kodierung und Signalverarbeitung)	
ADSL2+	2,2 MHz	25

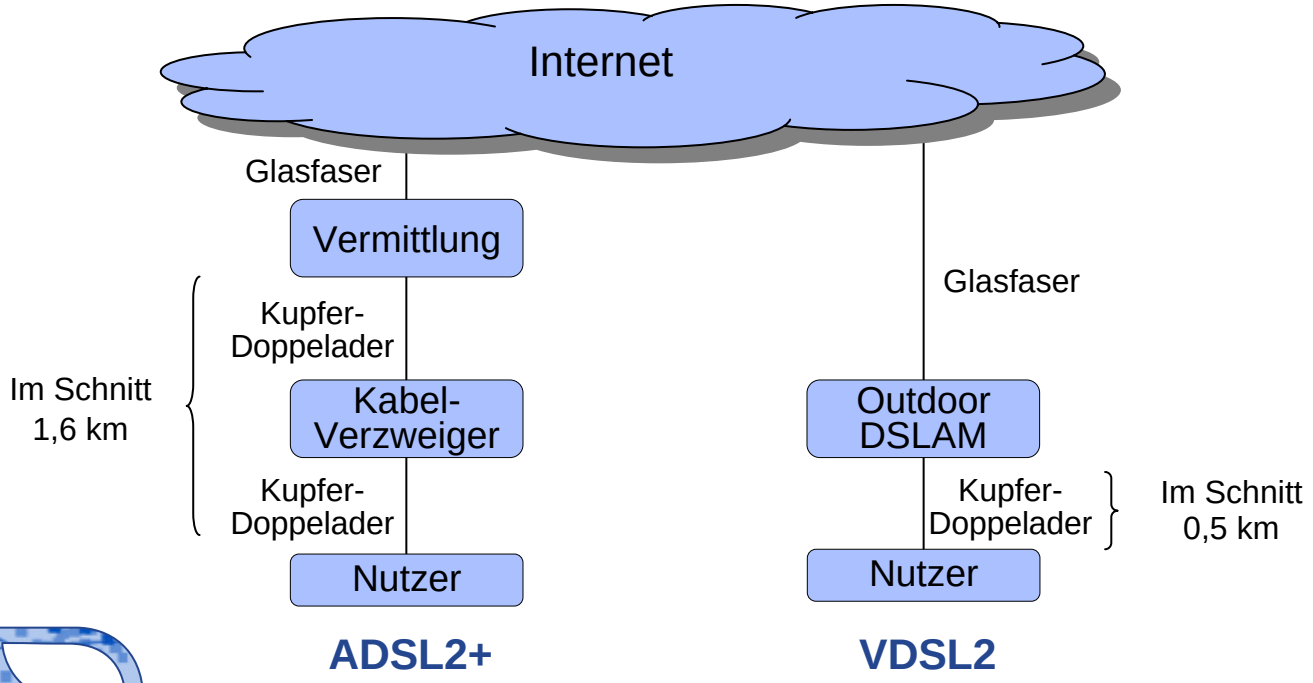
	Frequenz	Datenrate*
ADSL2+	2,2 MHz	25
VDSL1	12 MHz	100
VDSL2	30 MHz	210



*Annahme: Annex A, d.h. DSL-over-POTS

VDSL2

- Telekom baut Outdoor-DSLAMs
  - Umsetzung zwischen VDSL2-Übertragung auf Kupferseite und optischer Übertragung auf Glasfaserseite
    - Verkürzung der Kupferstrecken zwischen Nutzer und Vermittlungsstelle
    - „Fiber to the Curb“ (FTTC, Glasfaser zum Bordstein)



Waldhornstrasse, Karlsruhe

Werbung

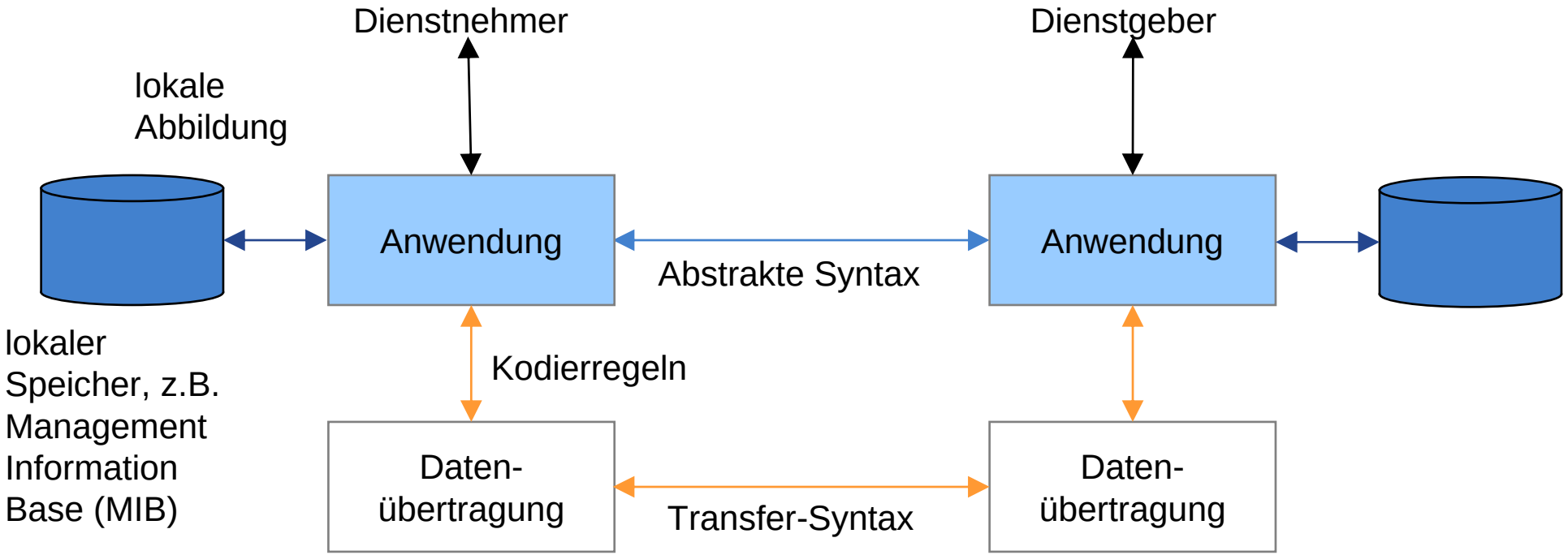
Kurzer Einschub: Werbung in eigener Sache ;-)

- Wir (das ITM) haben natürlich noch mehr zu bieten
  - Vertiefende Vorlesungen
  - Studienarbeiten
  - Bachelorarbeiten
  - Diplomarbeiten
  - Masterarbeiten
  - Praktika / Projektpraktika
  - Seminare

Aufgabe 4

ASN.1 & BER

Abstrakte Syntax und Transfersyntax



Probleme durch Heterogenität

- Unterschiedliche Wertebereiche
  - z.B. Typ `long`
    - 4 Bytes auf i386, PowerPC, Sparc
    - 8 Bytes auf Alpha, IA64, Sparc64, x86_64
- Speicherausrichtung
  - Byte-Padding von Compiler
    - GCC ermöglicht bspw. Angabe von `__attribute__((packed))`
- Nicht unterstützte Datentypen
  - Bsp.: Typ `unsigned int` unter Java nicht vorhanden
- Little oder Big Endian
  - Bsp.: Datei test.c

```
struct foo {
    char s[6];
    int i;
};

struct foo bar = {"FooBar", 0x12345678};
```

  - Objektcode erzeugen mittels
    - `gcc -c -o test.o test.c`
  - Assembler-Ausgabe erzeugen mittels
    - `gcc -S -o test.s test.c`

Endianess und Speicherausrichtung

```
$ hexdump -C test.o
00000000  fe ed fa ce 00 00 00 12  00 00 00 00 00 00 00 01  |................|
00000010  00 00 00 03 00 00 01 6c  00 00 20 00 00 00 00 01  |.......l.. .....|
00000020  00 00 01 04 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

```
00000130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 50  |...............P|
00000140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000180  00 00 00 00 00 00 00 00  46 6f 6f 42 61 72 00 00  |........FooBar..|
00000190  12 34 56 78                                       |.4Vx|
00000194
```

```
$ hexdump -C test.o
00000000  7f 45 4c 46 01 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  01 00 03 00 01 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020  b8 00 00 00 00 00 00 00  34 00 00 00 00 00 28 00  |........4.....(.|
00000030  07 00 06 00 46 6f 6f 42  61 72 00 00 78 56 34 12  |....FooBar..xV4.|
00000040  00 47 43 43 3a 20 28 47  4e 55 29 20 34 2e 31 2e  |.GCC: (GNU) 4.1.|
00000050  33 20 32 30 30 37 30 39  32 39 20 28 70 72 65 72  |3 20070929 (prer|
```

```
000001c0  00 00 00 00 00 00 00 00  01 00 00 00 00 00 00 00  |................|
000001d0
```

- test.s auf PowerPC (Big Endian)

```
   .section __TEXT,__text,[…]
   .section __TEXT,[…]
   .machine ppc
   .globl _bar
   .data
   .align 2_
bar:
   .ascii "FooBar"
   .space 2
   .long 305419896
   .subsections_via_symbols
```

- test.s auf Intel x86 (Little Endian)

```
   .file "test.c"
.globl bar
   .data
   .align 4
   .type  bar, @object
   .size  bar, 12
bar:
   .ascii "FooBar"
   .zero  2
   .long 305419896
   .ident "GCC: (GNU) 4.1.3 […]
```

Aufgabe 4 – Datentypdefinitionen

```
Interface ::= [APPLICATION 0] IMPLICIT SET {
      index   [0] INTEGER,
      descr       IA5STRING,
      type    [1] INTEGER{other(0), ethernetCsmacd(6)},
      addr        InterfaceAddress,
      data        InterfaceData
}

InterfaceData ::= [APPLICATION 2] IMPLICIT SEQUENCE {
      packetsIn       [0] IMPLICIT INTEGER,
      packetsOut      [1] IMPLICIT INTEGER,
      packetsTotal    [2] IMPLICIT INTEGER OPTIONAL
}
```

Aufgabe 4 (a) – Instanziierung

- LAN-Netzwerkkarte
  - Index 0
  - Hersteller 3Com
  - Eing. Dateneinh. 521
  - Ausg. Dateneinh. 130

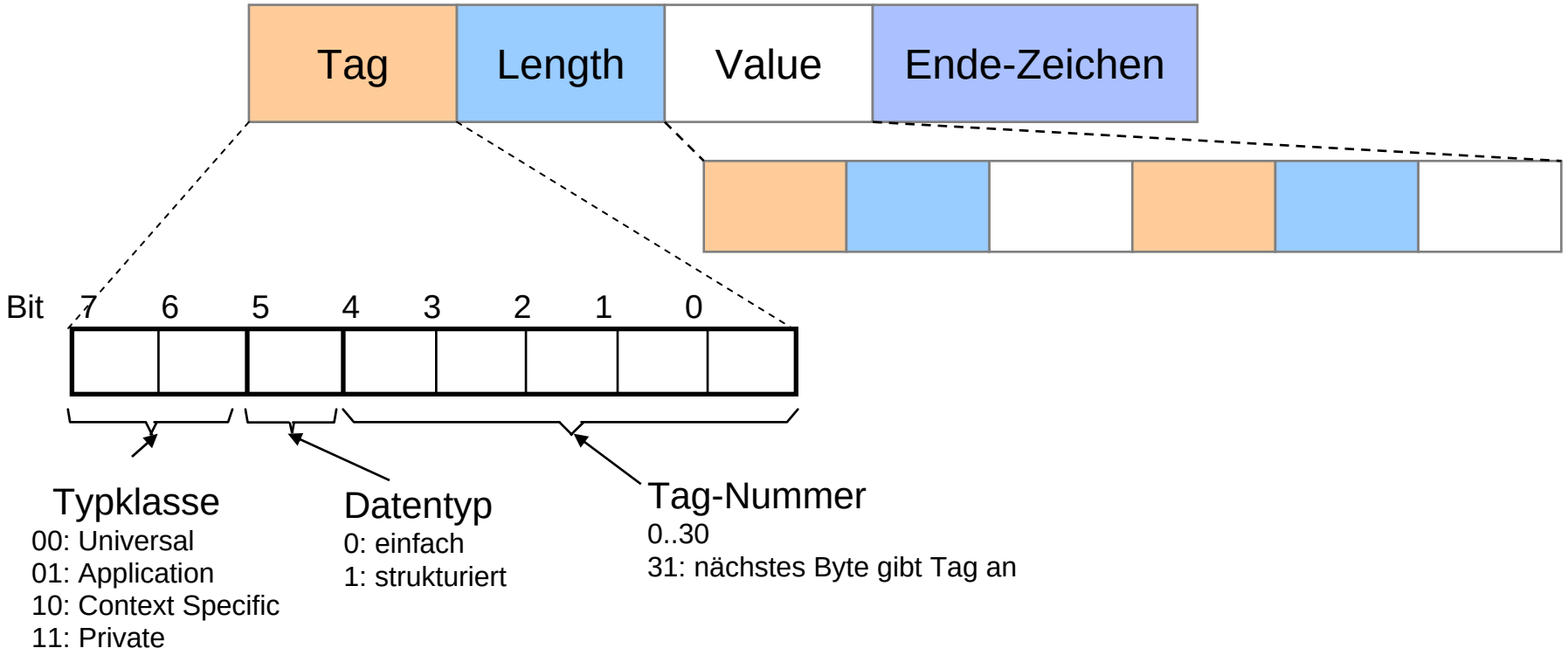
```
Interface ::= [APPLICATION 0] IMPLICIT SET {
     index   [0] INTEGER,
     descr       IA5STRING,
     type    [1] INTEGER{other(0),ethernetCsmacd(6)},
     addr        InterfaceAddress,
     data        InterfaceData
}

InterfaceData ::= [APPLICATION 2] IMPLICIT SEQUENCE {
     packetsIn       [0] IMPLICIT INTEGER,
     packetsOut      [1] IMPLICIT INTEGER,
     packetsTotal    [2] IMPLICIT INTEGER OPTIONAL
}
```

- Lösung

```
LAN-Netzwerkkarte Interface ::= {
          index   0,
          descr   "3Com"
          type    6,
          data    {
                  packetsIn    521,
                  packetsOut   130,
          }
}
```

Aufgabe 4 (b) – BER-Kodierung

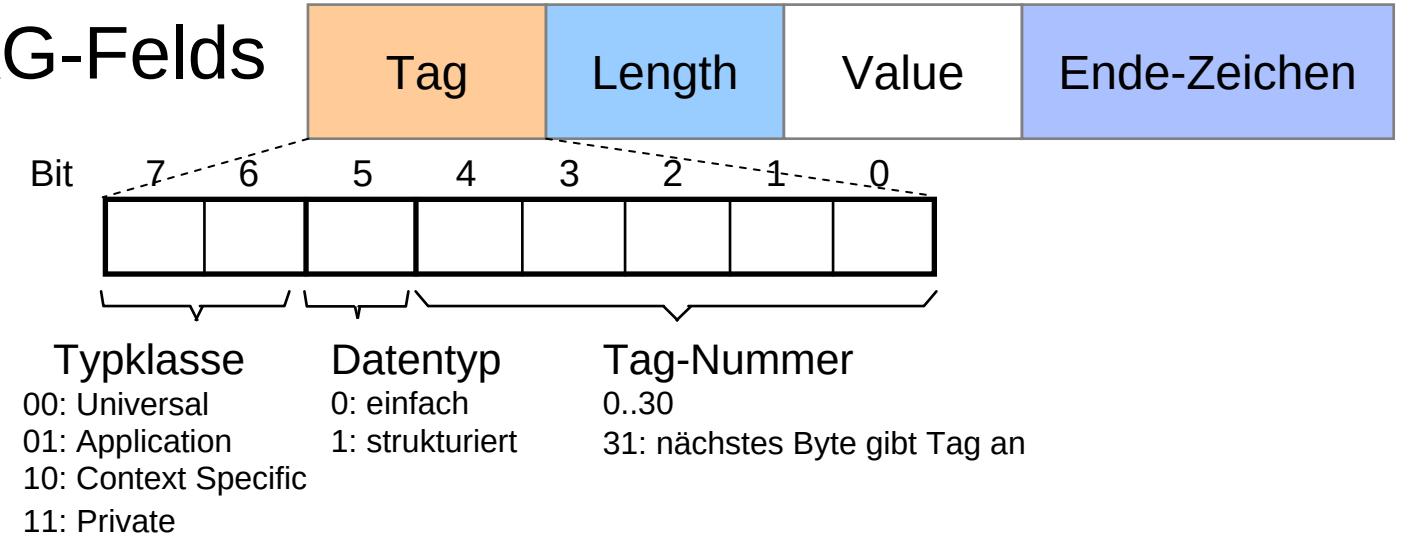
- Grundlegender Aufbau



- Beispiel

Aufgabe 4 (b) – BER-Kodierung

- Aufbau des TAG-Felds




```
Interface   = 01|1|00000 (Appl. Spec., Constructed, Implicit Tag 0) = 0x60
index       = 10|1|00000 (Context spec., Constructed, Explicit Tag 0) = 0xA0
            = 00|0|00010 (UNIVERSAL, Primitive, Integer) = 0x02
descr       = 00|0|10110 (UNIVERSAL, Primitive, IA5STRING) = 0x16
type        = 10|1|00001 (Context spec., Constructed, Explicit Tag 1) = 0xA1
            = 00|0|00010 (UNIVERSAL, Primitive, Integer) = 0x02
data        = 01|1|00010 (Appl. Spec., Constructed, Implicit Tag 0) = 0x80
packetsIn   = 10|0|00000 (Context spec., Primitive, Implicit Tag 0) = 0x80
packetsOut  = 10|0|00001 (Context spec., Primitive, Implicit Tag 1) = 0x81
```

Aufgabe 4 (b) – BER-Kodierung (Forts.)

- Resultierende Kodierung in Hex-Schreibweise

```
LAN-Netzwerkkarte Interface ::= {
                        0x60 0x19
            index  0,
            0xA0 0x03 0x02 0x01
            descr  "3Com"
            0x16 0x04 0x33 43 6F 6D
            type   6,
            0xA1 0x03 0x02 0x01 0x06
            data   {
            0x62 0x07
                   packetsIn    521,
                   0x80 0x02 0x02 09
                   packetsOut   130,
                   0x81 0x01 0x82
            }
}
```

Beispiel ASN.1 Compiler – asn1c

- Open-Source Compiler [*http://lionet.info/asn1c/*]
- Datei `rectangle.asn1` erstellen

```
RectangleModule1 DEFINITIONS ::=
BEGIN
Rectangle ::= SEQUENCE {
    height  INTEGER,
    width   INTEGER
}
END
```

- ASN.1 Datei kompilieren
  - `asn1c -fnative-types rectangle.asn1`
- Datei `main.c` ggf. anpassen

```
#include <stdio.h>
#include <sys/types.h>
#include <Rectangle.h>   /* Rectangle ASN.1 type  */
[...]
```

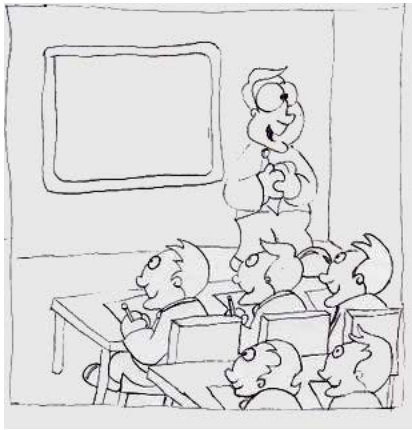
Beispiel ASN.1 Compiler – asn1c (Forts.)

- Binary erstellen, dass uns die BER- (bzw. auch XER-) Kodierung erzeugt
  - `cc -I. -o rencode *.c`
- BER-Datei erstellen mittels
  - `./rencode ber-sample`
- Binäre BER-kodierte Datei

```
$ hexdump –C ber-sample
00000000 30 06 02 01 2a 02 01 17
```

Fragen?

Vielen Dank für zahlreiches Erscheinen
Und (tatkräftige) Unterstützung!



Viel Erfolg auf dem weiteren Studienweg!