Rechnernetze I

SoSe 2025

Roland Wismüller
Universität Siegen
roland.wismueller@uni-siegen.de
Tel.: 0271/740-4050, Büro: H-B 8404

Stand: 26. Juni 2025

Rechnernetze I

SoSe 2025

8 Datendarstellung

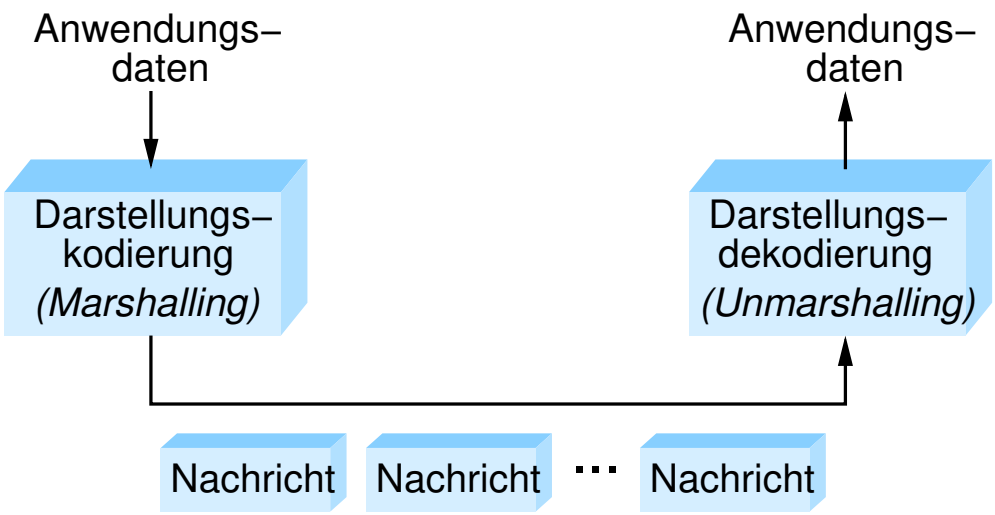
Inhalt

- *Marshalling*, Designalternativen

- Peterson, Kap. 7.1

8.1 Marshalling

- Daten der Anwendungen müssen in eine für die Übertragung geeignete Form umgewandelt werden:
  - **Darstellungsformatierung**
  - ***Marshalling / Unmarshalling***



Problem: Heterogenität

- Rechner haben unterschiedliche Datendarstellung
- Z.B. einfache 32-Bit Ganzzahl
  - 34.677.374 = $0211227E_{16}$ (Hexadezimal)

Big–endian (z.B. Sparc)

(02_{16})	(11_{16})	(22_{16})	$(7E_{16})$
00000010	00010001	00100010	01111110

Little–endian (z.B. x86)

$(7E_{16})$	(22_{16})	(11_{16})	(02_{16})
01111110	00100010	00010001	00000010
Adresse x	x+1	x+2	x+3

Unterschiede in der Datendarstellung

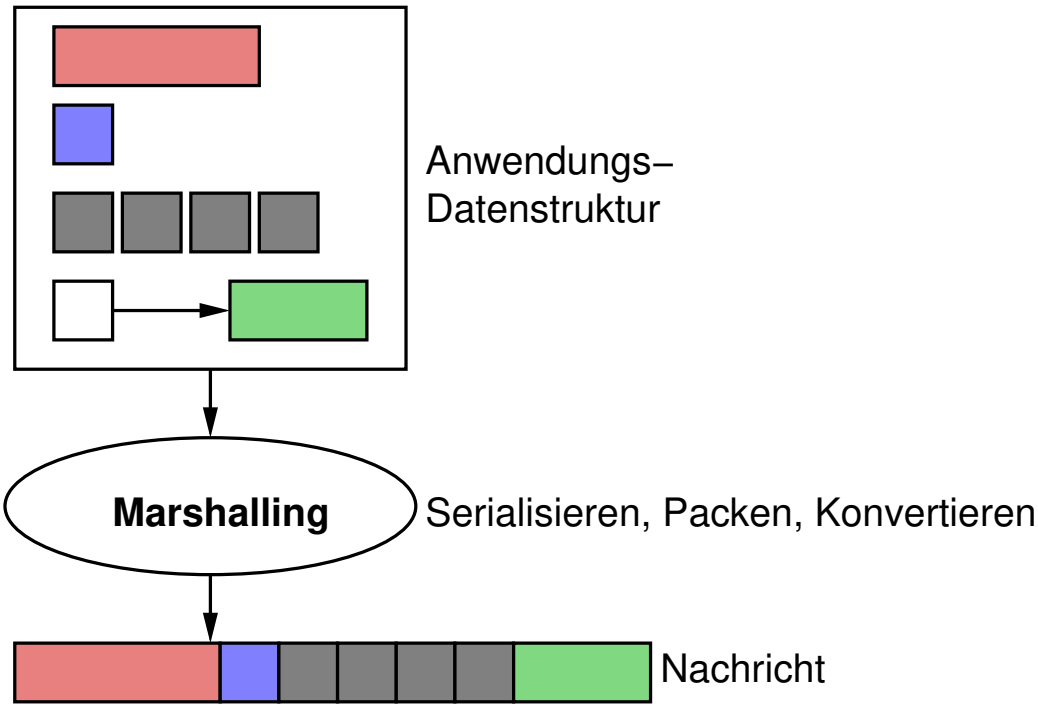
- Ganzzahlen:
  - Größe (16, 32, 64 Bit)
  - Byte-Reihenfolge: *Big Endian* vs. *Little Endian*
- Gleitkommazahlen
  - IEEE 754 vs. proprietäre Formate
  - Größe (bes. bei *extended precision*: 80, 96, 128 Bit)
  - Byte-Reihenfolge
- Datenstrukturen:
  - *Alignment* der Komponenten im Speicher
    - z.B. darf eine 16-Bit Zahl an einer ungeraden Adresse beginnen?

★★

Datentypen beim *Marshalling*: drei Ebenen

- **Basistypen**: int, double, Boolean, char, ...
  - Formate und Byte-Reihenfolge konvertieren
- **Flache Datentypen**: Strukturen, Arrays
  - Füllbytes für *Alignment* entfernen (**packen**) bzw. einfügen (**auspacken**)
- **Komplexe Datentypen (mit Zeigern)**: Listen, Bäume, ...
  - **Serialisierung** der Datenstruktur notwendig
    - Zeiger sind Speicheradressen, sie können nicht übertragen werden
  - Empfänger **deserialisiert** wieder

Marshalling veranschaulicht



Konvertierungsstrategien

- **Kanonisches Netzwerk-Datenformat**
  - Sender konvertiert in Netzwerk-Datenformat
  - Empfänger konvertiert dann in sein Format
- ***Receiver-Makes-Right***
  - Sender sendet in seinem eigenen Format
    - serialisiert lediglich
  - Empfänger konvertiert in sein Format, falls nötig
- Diskussion:
  - *Receiver-Makes-Right* ist N-mal-N-Lösung
    - jeder Empfänger muß alle Formate kennen
  - Meist sind die Formate aber identisch (N ist klein)

Datentyp-Kennzeichnung (Tags)

- **Tag**: Information, die hilft, die Nachricht zu dekodieren, z.B.:
  - Typ-Tag
  - Längen-Tag (etwa für Arrays)
  - Architektur-Tag (für *Receiver-Makes-Right*)
- Beispiel: 32-Bit-Ganzzahl mit Tags:

- Oft kann man auch ohne Tags arbeiten:
  - Sender und Empfänger wissen, was übertragen wird
  - Darstellungsformatierung dann aber Ende-zu-Ende

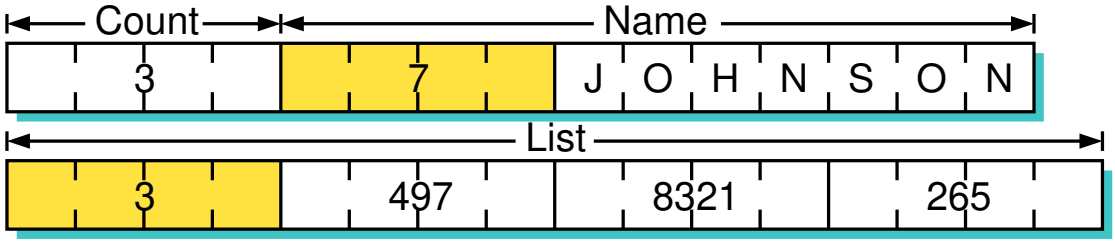
XDR (*eXternal Data Representation*)

- Netzwerkformat von SunRPC
  - RPC: *Remote Procedure Call*
    Aufruf von Diensten auf anderen Rechnern analog zu normalen Prozeduraufrufen
- Vollständiges C-Typsystem (bis auf Funktionszeiger)
- Kanonisches Netzwerk-Datenformat
  - z.B. Ganzzahl mit Vorzeichen:
    32-Bit, Zweierkomplement, Big-Endian
- Keine Verwendung von Tags (außer Array-Längen)

Beispiel zur Verwendung von XDR:

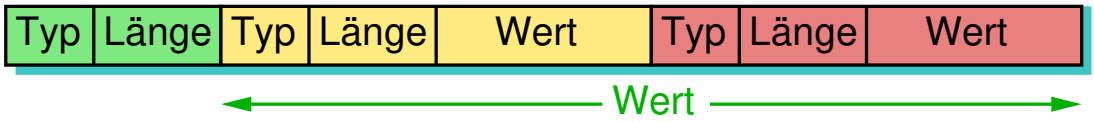
```
struct item {
  int  count;
  char name[MAXNAME];
  int  list[MAXLIST];
};

bool_t xdr_item(XDR *xdrs, struct item *ptr)
{
  return xdr_int(xdrs, &ptr->count)
  && xdr_string(xdrs, &ptr->name, MAXNAME)
  && xdr_array(xdrs, &ptr->list, &ptr->count,
  MAXLIST, sizeof(int), xdr_int);
}
```



ASN.1 (*Abstract Syntax Notation One*)

- ISO-Standard
- Teil von ASN.1: *Basic Encoding Rules* (BER)
  - definieren Datendarstellung
- Vollständiges C-Typsystem (bis auf Funktionszeiger)
- Kanonisches Netzwerk-Datenformat
- Verwendung von Typ-Tags
  - Jedes Datenelement wird als Tripel dargestellt:
    < Typ, Länge, Wert >
  - Bei Strukturen auch verschachtelt:



XML (*eXtensible Markup Language*)

- Erlaubt die Darstellung komplexer (hierarchischer) Datenstrukturen in Textform
- Zwei Bestandteile:
  - XML-Schema bzw. DTD (*Document Type Definition*)
    - legt Struktur der Daten fest (Datentyp)
  - XML-Dokument enthält Daten und Tags:

```
<address>
  <name>John Smith</name>
  <city>Siegen</city>
</address>
```

- Problem: sehr ineffizient

8.2 Beispiele ...

JSON (*JavaScript Object Notation*)

- Einfach lesbare Textdarstellung verschachtelter Datenstrukturen
- Kann alle Datentypen von JavaScript darstellen
  - Null, Boolean, Zahlen, Strings, Arrays, Objekte mit Attributen
- Datentyp beschrieben durch JSON-Schema (in JSON codiert)
  - wie bei XML kann JSON-Dokument automatisch gegen Schema geprüft werden
- Darstellung lesbarer und mit weniger Overhead als XML, z.B.:

```
{
  "address": { "name": "John Smith", "city": Siegen }
}
```

- Große Verbreitung in Netzwerk-Anwendungen

8.3 Zusammenfassung

- Daten werden in Rechnern unterschiedlich dargestellt
- Bei der Übertragung ist eine Anpassung erforderlich
  - 3 Schritte: Serialisieren, Packen, Konvertieren
- Realisierungs-Alternativen:
  - kanonisches Format vs. *Receiver-Makes-Right*
  - mit / ohne Datentyp-Kennzeichnung

Nächste Lektion:

- Anwendungsprotokolle