

# GRUNDLAGEN

---

# VERTEILTE SYSTEME

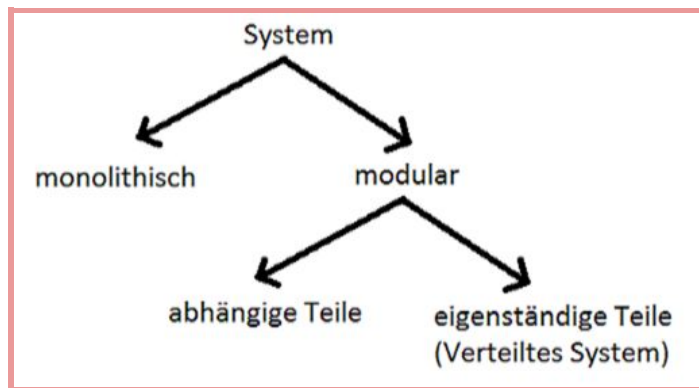
Verteilt = nebeneinander existierend örtlich oder zeitlich verteilt.

## PRÄGEN VERTEILTE SYSTEME

Nebenläufigkeit (unabhängig voneinander) und Kommunikation

## MÖGLICHE VERTEILUNGEN

Methoden, Daten (verteilte Datenbanksysteme), Rechenzeit, Dateisysteme, Anwendungen (z.B. Excel-Tabelle die in Word importiert wird).



## CLOUD-COMPUTING

Anwendungen od. Daten können über Cloud genutzt werden.

Verteilt, da Client und Server (man weiss nicht was in Cloud drin ist, Vertrauen ist hier wichtig).

## VORTEILE UND NACHTEILE

### Zeit

Ersparnis, wegen der Parallelität.

Aber wenn zwei auf eine Ressource zugreifen wollen = Zeitverlust.

### Kosten

Kosten können reduziert werden (z.B. Rechenleistung eines anderen Systems wird genutzt) .

Aber es wird z.B. Personal die unterschiedliche Systeme betreuen können benötigt (einer kann apple der andere windows) = mehr Kosten.

### Skalierbarkeit

Rechner dazunehmen und wieder entfernen = sehr flexibel.

### Sicherheit

Client ist abgeschirmt.

Aber wenn Rechner vernetzt sind, weiss man nicht was dazwischen passiert (es könnte jmd. d. Daten abgreifen).

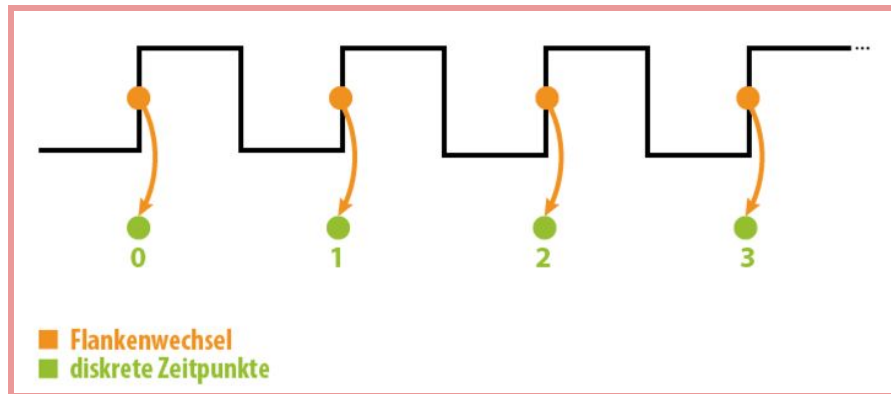
### Wartung

Schlechtere Wartung, da viele Systeme gepflegt werden müssen.

# ZEIT

In jd. Rechner ist ein Taktgeber.

Nur beim Flankenwechsel kann etwas getan werden.



Zu jedem zeitl. Verlauf gibt es ein Intervall  $[t_1, t_2]$  ( $t_1 \in t_2$ ).

$t_1$  = Ereignis beginnt.

$t_2$  = Ereignis endet.

## ZEITPUNKT

= wenn  $t_1 = t_2$ .

Bei Zeitpunkten A, B, ... wird oft  $A[t_1, t_2]$  geschrieben (aber nur wenn Zeitpunkt eine Rolle spielt).

## UHR (TIMER)

= Zähler, der diese Takte zählt.

Zähler für Zeitpunkte.

# SEQUENTIALITÄT

Man nehme zwei Ereignisse  $A[t_1, t_2]$   $B[t_3, t_4]$ .

A liegt vor B (B nach A), genau dann wenn,  $t_2 < t_3$

Man schreibt  $A \rightarrow B$  (A vor B).

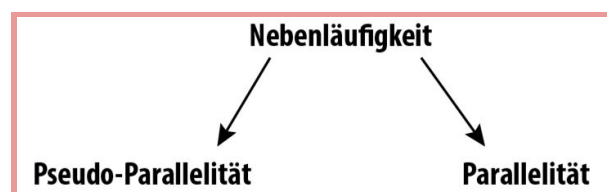
## DEFINITION

Zwei Ereignisse A, B sind sequenziell genau dann, wenn  $A \rightarrow B$  oder  $B \rightarrow A$ .

Sonst liegt Nebenläufigkeit vor.

# NEBENLÄUFIGKEIT

Zwei Ereignisse A, B sind nebenläufig genau dann, wenn sie nicht sequenziell sind.

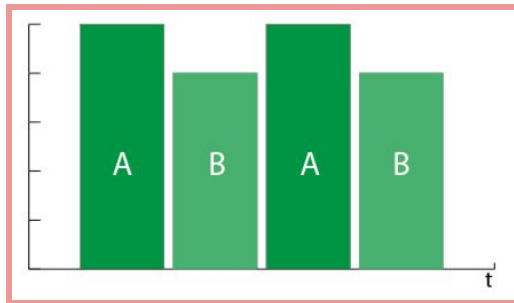


## PSEUDO-PARALLELITÄT

Man benötigt keine 2 Prozessoren, um Nebenläufigkeit zu erreichen (Pseudo = falsch).

Ein Taktgeber = Nebenläufigkeit durch Timesharing.

### Beispiel



Prozess A wird durch Timesharing unterbrochen, B wird ausgeführt usw.

### PARALLELITÄT

Nebenläufigkeit durch mehrfach vorhandener Komponenten.

### PROBLEME

#### Beispiel

$i = 7; i = 8;$

Sequenzielle Deklaration = kein Problem.

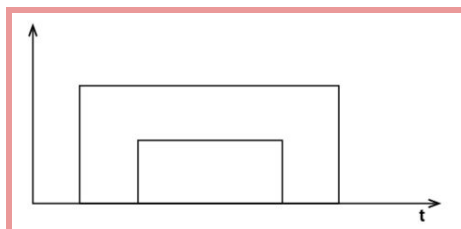
Nebenläufig =  $i$  wird zufällig deklariert (wer als letztes schreibt gewinnt).

$i = 7; k = 8;$

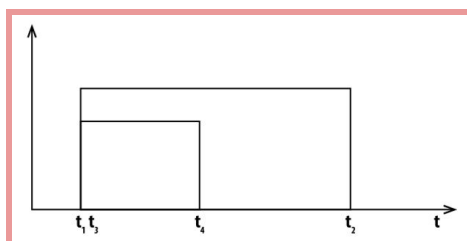
Auch nebenläufig kein Problem.

Es gibt Ereignisse, die nebenläufig arbeiten müssen, andere können nebenläufig laufen.

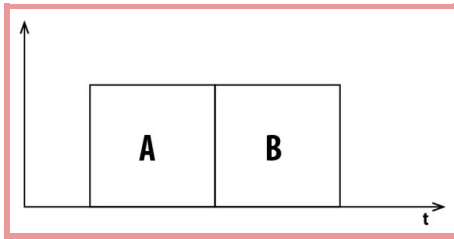
### BEISPIEL



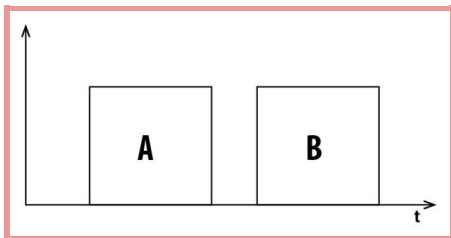
Nebenläufig.



Nebenläufig.



Nebenläufig, da A sofort an B grenzt.



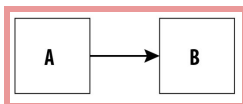
Sequenziell.

## BESCHREIBUNG

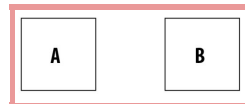
Petri-Netze (prof. Werkzeug)

## Warte-Graph

sequenziell

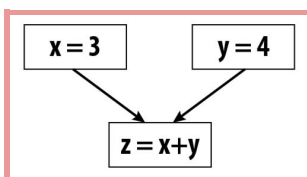


nebenläufig



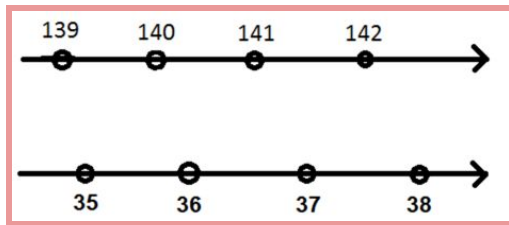
## Beispiel

```
x = 3;
y = 4;           //x und y-Deklaration nebenläufig möglich
z = x+y          //muss warten
```



# MEHRUHRSYSTEME

2 Taktgeber haben nie das selbe Intervall (eins = immer etwas kürzer od. länger als das andere).



Wie soll hier Sequenz festgestellt werden ( $140 < 37$ , aber liegt vor 37)?

## GLOBALE UHREN

Angenommen 2 Systeme sollen auf diese zugreifen + gleiche Zeit haben.

Prozessor wird nicht mitten im Befehle unterbrochen.

→ nur Annäherung möglich.

→ es gibt keine globalen Uhren.

## LOGISCHE UHREN

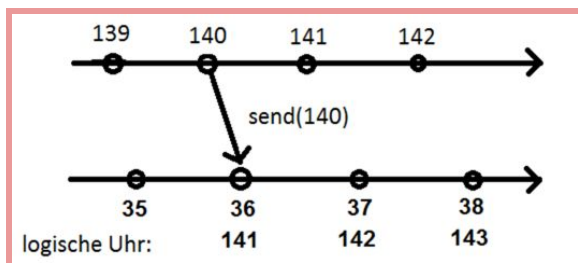
Haben Systeme nichts miteinander zu tun = irrelevant.

Sonst = Mitlieferung eines Zeitstempels.

anderes System nimmt Zeitstempel + addiert 1 drauf.

## Vorteil

Sequenz auch in Mehruhrsystemen feststellbar.



# PROZESSE UND THREADS

---

# PROGRAMME

= Sammlung von Befehlen.

= statisch.

# PROZESSE

Auch Tasks genannt.

= Abarbeitung eines Programms.

Jd. Prozess hat eigenen Hauptspeicherbereich, kann nicht auf anderen Hauptspeicherbereich eines anderen Prozesses zugreifen (Vorteil = Prozess kann nicht Variablen eines anderen Prozesses falsche Werte zuweisen).

## PROZESSENDE

Prozess wird beendet, alle Threads werden mit beendet.

Parameter, damit man sieht an welcher Stelle Prozess beendet wird.

Es gibt kein exit für Threads, d.h. run() immer sauber beenden.

```
System.exit(int);
```

### Argument 0

Standard, wurde Prozess ohne Fehler beendet.

### Argument != 0

Prozess wurde mit Fehler beendet (verschiedene Werte für verschiedene Fehler möglich).

Wertebereich sollte zw. 0 und 255 liegen.

## UNTERSCHIED PROGRAMM UND PROZESS

### Beispiel Backrezept (Programm)

*"Man gebe in eine Schüssel: 200g Mehl, 100g Zucker, und 3 Eier.*

*Man verrühre nun die Zutaten und gebe den Teig in eine Backform.*

*Anschließend backe man diesen Teig 1 Std. lang bei 200 Grad"*

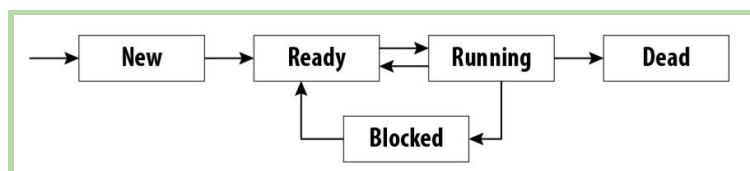
Programm alleine = kein Kuchen.

Kuchen = wenn Programm in einem Prozess "Backen" gestartet wird.

Ausführende Person (Prozessor) kann Prozess "Backen" unterbrechen.

In dieser Zeit kann anderer, wichtigerer Prozess (z.B. "Reaktion auf Haustürglocke") bearbeitet werden.

## PROZESSZUSTÄNDE



### Blocked



z.B. kann andere Prozess stattfinden, wenn IO-Vorgang stattfindet.

### Dead

Wenn Running (Prozess) fertig ist.

## SCHEDULING

Auswahl des nächsten zu bearbeitenden Prozesses und Umschaltung erfolgt durch Scheduling-Programm des Betriebssystems.

Scheduling-Programm unterbricht laufenden Prozess + startet, falls vom Algorithmus vorgesehen, anderen Prozess.

Betriebssysteme sieht nur Prozesse, nicht d. Threads.

Java Virtuelle Maschine = Laufzeitsystem → dort eigener kleiner Scheduler.

### PRIORITY

Wichtigster Prozesse gewinnt.

### Nachteil

Es können immer wichtigere Threads vorhanden sein → Threads, die nie verarbeitet werden.

Lösung: Prioritäten altern lassen (Prozesse bekommen an Wichtigkeit +1) → Nachteil: z.B alle die Priorität 10

## SEQUENZIELLE PROGRAMME

Befehle werden in best. Reihenfolge (Sequenz) ausgeführt.

Programm, das von nur einem sequentiellen Ausführer ausgeführt wird.

### NACHTEILE SEQUENZIELLER PROGRAMME

Langsamere Ausführung.

### Beispiel

Von 2 Tastaturen sollen Zahlen eingelesen werden.

```
while (...){  
    ...  
    Lies Zahl z1 von Tastatur T1;  
    Addiere z1 zur Summe;  
    ...  
    Lies Zahl z2 von Tastatur T2;  
    Addiere z2 zur Summe;  
    ...  
}
```

Zwingt beide Nutzer gleich viele Zahlen einzugeben.

Wenn Nutzer 2 Pause macht, kann Nutzer1 keine weiteren Zahlen eingeben.

### Lösung

Programmierer überlässt dem Ausführer die Ausführungsreihenfolge.

Ausführer spaltet sich in mehrere sequentielle Ausführer, die nebenläufig zueinander (zeitl. unabhängig) Programme od. Proramnteile ausführen können.

## THREADS

= parallel laufende Ablauffäden innerhalb eines Prozesses, die sich alle Ressourcen des Prozesses teilen (können untereinander Daten austauschen).

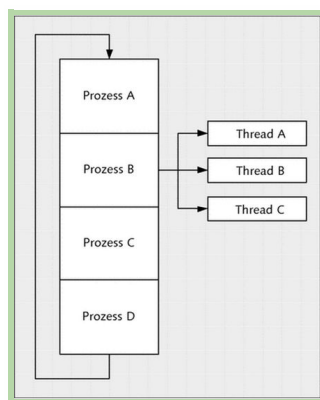
Ermöglichen parallele Abläufe innerhalb eines Programms.

Führt z.B. Programm nach Tastendruck längere Berechnung durch, sollen trotzdem nebenbei andere Fkt. ausgeführt werden.

Umschalten = (wie bei Prozessen) durch Scheduler.

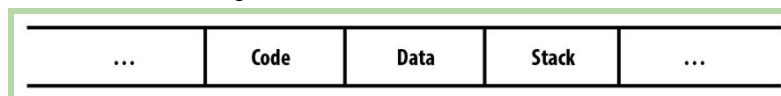
Prozesse mit mehreren Threads haben eigenen kleinen Scheduler.

Threads werden nicht gleichzeitig, sondern abwechselnd ausgeführt.



### ADRESSRAUM

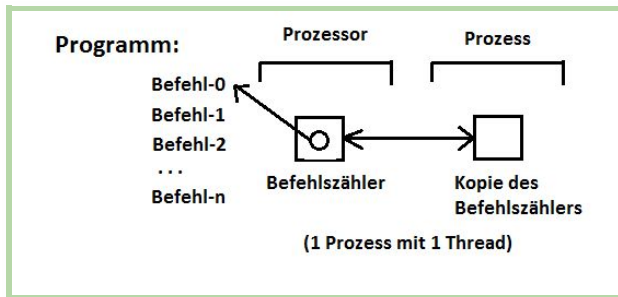
Jeder Thread hat eigenen Stack.



*Hauptspeicherbereich eines Prozesses.*

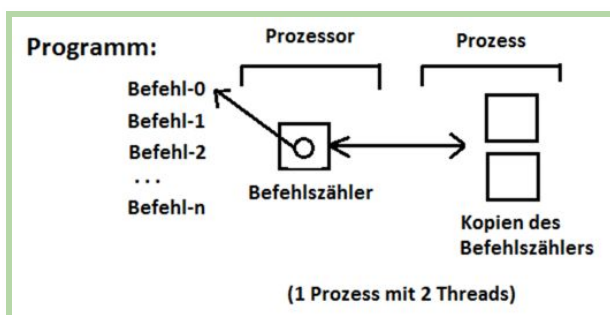
### PROZESS MIT EINEM THREAD

Programm-Abbruch = letzter Stand wird in Kopie d. Befehlszählers kopiert, um da fortfahren zu können.

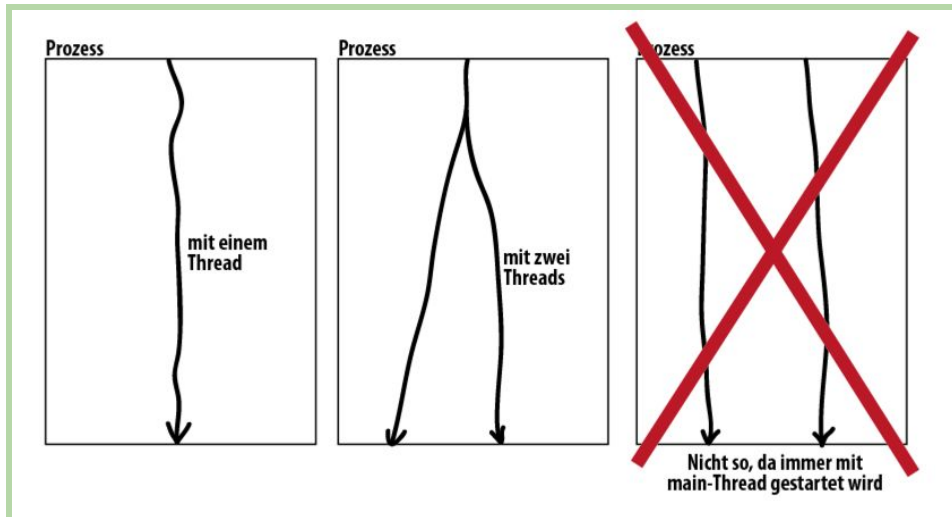


## PROZESS MIT 2 THREADS

2 Kopien d. Befehlszählers → es kann an unterschiedlichen Stellen fortgefahren werden.  
Unabhängige Abarbeitung möglich.



## DARSTELLUNG



## UNTERSCHIED PROZESSE UND TREADS

Prozesse verfügen über eigenen Speicherbereich.

Aber alle (zum gleichen Prozess gehörende) Threads teilen sich (Prozess-) Speicherbereich.

Thread = sequentieller Ausführer.

Java-Programm, in dem Threads erzeugt werden = Programm mit mehreren Ausführern.

## NACHTEILE

Schwerer zu Testen, als sequentielle Programme.

Fehler, die bei best. Ausführung aufgetreten sind, lassen sich nur schwer wieder reproduzieren

## THREADS IN JAVA

Zwei verschiedene Wege Threads zu programmieren.

### THREAD ERWEITERN

In Klasse Thread ist run()-Methode mit leerem Rumpf vereinbart.

Wird überschrieben.

Thread-Objekte haben Namen, den man als Parameter mitliefern kann.

Kein Argument als Name = Thread bekommt Standardnamen (Thread-0, Thread-1,...).

getName() liefert Namen des Objektes.

sleep(500) = Thread wird mind. 500 Millisek. nicht ausgeführt.

```
class Faden1 extends Thread {
    static Random ran = new Random();

    Faden1(String name){
        super(name);
    }

    Faden1(){
        super();
    }

    public void run(){
        String name = getName();
        int millis = 0;

        for(int i=1; i<=5; i++){
            println(i + ". " + name + ", msec: " + millis);

            try{
                /*zufaelliger Wert aus Intervall 0 (einschließlich)
und
                500(ausschließlich)*/

                millis = ran.nextInt(500);
                sleep(millis);

            } catch (InterruptedException ie){
                println(ie);
            }
        }
    }
}
```

```
}
```

### Faden1 erzeugen und starten

start()-Methode = richtet Thread ein + startet run(), fad1 wird gestartet (Ausführer beginnt, run-Methode nebenläufig zu allen anderen Threads auszuführen).

```
public static void main(){
    Faden1 fad1 = new Faden1();
    Faden1 fad2 = new Faden1("Thread1");
    fad1.start();
    fad2.start();
}
```

### RUNNABLE IMPLEMENTIEREN

Schnittstelle Runnable enthält als einzige Methode run().

Da die Klasse Faden2 von Oberklasse Object keine getName und sleep-Methoden erbt, werden Klassenmethoden der Klasse Thread aufgerufen.

```
class Faden2 implements Runnable{
    static Random ran = new Random();

    public void run(){
        String name = Thread.currentThread().getName();
        int millis = 0;

        for(int i=1; i<=5; i++){
            pln(i + ". " + name + ", msec: " + millis);
            try{
                /*zufaelliger Wert aus dem Intervall 0(einschließlich) und
                500(ausschließlich)*/
                millis = ran.nextInt(500);
                Thread.sleep(millis);
            } catch (InterruptedException ie){
                pln(ie);
            }
        }
    }
}
```

Faden2-Objekt = eigentlich nur Behälter, der run-Methode enthält.

So verpackte run-Methode kann man Thread-Objekt übergeben als Ersatz für seine run-Methode mit leerem Rumpf.

```
public static void main(String[] args){
    Runnable r1 = new Faden2();
```

```

Runnable r2 = new Faden2();
Thread f1 = new Thread(r1);
Thread f2 = new Thread(r1);
f1.start();
f2.start();
}

```

## THREADS AUSGEBEN

```
System.out.println(Thread.currentThread().getName());
```

Erster Thread der gestartet wird = main, danach default-Namen, wie thr1, thr2, ...

## Beispiel

Ausgabe nicht klar, da der Scheduler entscheidet welcher Thread, wann ausgeführt wird.

Ausgabe entweder main Thr\_0 oder Thr\_0 main.

```

class MyThread extends thread {
    public void rund() {
        syso(Thread.currentThread().getName());
    }
}

class Work {
    public static void main() {
        MyThread mt = new MyThread();
        mt.start();
        syso(Thread.currentThread().getName());
    }
}

```

## START UND JOIN

start()

Startet nebenläufig Programm.

join()

Wartet auf dessen Ende.

Blockiert aufrufenden Thread so lange, bis Thread t seine Ausführung beendet hat.

Realisierung v. Verarbeitungsketten, bei denen Thread auf Fertigstellung eines Zwischenergebnisses wartet, welches wiederum von anderen Threads berechnet wird.

## TIMOUTS

Thread wird, für best. Zeit (in ms), unterbrochen.

```
Thread.sleep(ms);
```

## INTERRUPTS

Um Schlaf zu unterbrechen.

```
interrupt()
```

## JOINS

Synchronisiert Vater-Threads mit Ende seines Sohnes.

```
public final void join() throws InterruptedException
```

## ALIVE

Nicht alive, wenn Thread = dead.

```
public final boolean isAlive()
```

# SCHEDULING IN JAVA

Thread mit höchster Priorität erhält Prozessor.

Gleiche Priorität = Round Robin.

## PRIORITÄTEN

1 = tief und 10 = hoch.

Aber default = 5 Prioritätsstufen.

## GET

Prioritätsstufen abrufen.

```
getPriority()
```

## SET

Prioritätsstufen setzen.

```
setPriority()
```

## YIELD

Thread gibt Prozessor für anderen Thread frei.

```
public static void yield()
```

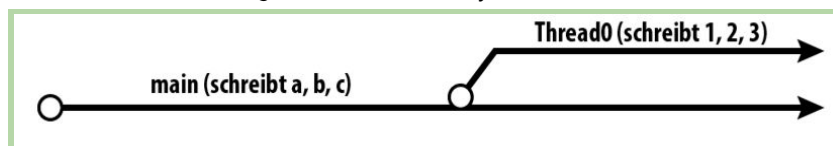
Problem = yield b. Thread mit hoher Priorität bekommen gleich wieder den Prozessor, Priorität müsste runter gesetzt werden.

# THREADSICHERHEIT

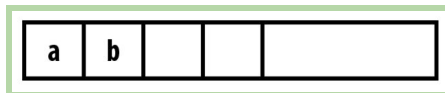
## THREAD-PROBLEM

### Beispiel

Zwei Threads nutzen gemeinsames Array.



1. main schreibt und wird unterbrochen.



2. Thread0 schreibt und wird unterbrochen.



3. Dann schreibt wieder main und wird unterbrochen usw.



## MAßNAHMEN

1. **Schreibkonflikte vermeiden**  
z.B. zwei Stacks.
2. **Datenstruktur die Threadsafe ist**  
bzw. synchronized.  
ArrayList = **nicht** threadsave.  
StringBuffer = threadsave.
3. **Selber threadsave machen**  
z.B. Stacks threadsave machen.  
Mit synchronized.

## SYNCHRONIZED-BLOCK

Jd. Klasse und jd. Instanz hat Sperre (in Object festgelegt).

Es muss festgelegt werden, welche Sperre verwendet werden soll.

Thread setzt Sperre und blockiert nächsten Aufrufer, solange bis er fertig ist.

Werden kaum verwendet, eher synchronized-Methoden.

## Beispiel

```
String abc = "Hallo";
```

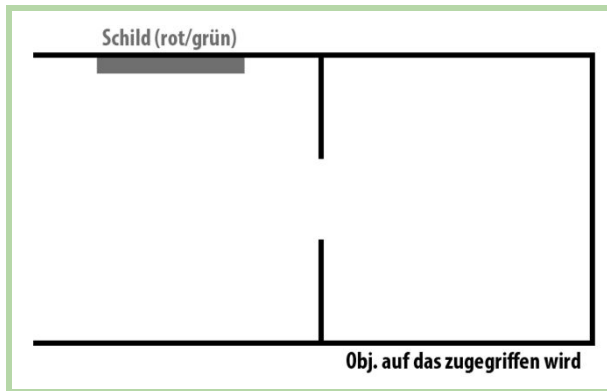
(Sperre 1 = String, Sperre 2 = abc).

```
synchronized (abc){ //Sperre wird gesetzt
    //Befehle
}                    //Sperre wird wieder freigegeben

synchronized (abc.getClass()){ //String-Sperre wird gesetzt
    //Befehle
}
```



## Problem



1. Synchronized von Thread0 guckt nach Schild = grün → kann durch.
2. Synchronized von Thread1 guckt nach Schild = rot → wartet.
3. Thread3 hat aber kein synchronized und geht einfach durch.

Wenn synchronized, dann alles synchronized.

Wenn alle Threads warten, entscheidet Scheduler, wer dran ist (evtl. mit Priorität arbeiten).

## SYNCHRONIZED-METHODEN

Sperre d. eigenen Klasse.

Um Klassen threadsicher zu machen.

Zuerst begonnene synchronized-Methode muss zu Ende ausgeführt werden, bevor neue synchronized Methode in anderem Thread begonnen wird (können nicht konkurrierend zueinander laufen, werden sequentiell ausgeführt).

Tote Threads sperren nicht mehr.

```
void f(n) {  
    synchronized(this) {  
        //Befehle  
    }  
}
```

### Kürzer

```
synchronized void f(n) {  
    //Befehle  
}
```

## LOST UPDATE

### Beispiel

Klasse mit Zähler i.

Thread mit run() = i++

2x start() des Threads.

i++ = nicht Atomar → mind. 3 Befehle:

1. lies i
2.  $i = i+1$
3. Schreibe i

Virtuelle Maschine führt Byte-Befehle aus, keinen java-Code.

Scheduler unterbricht aber Byte-Befehle keine Java-Befehle.

Es kann zw. diesen 3 Befehlen unterbrochen werden (man hat keinen Einfluss darauf).

# PROTOKOLLE

---

# PROTOKOLLE

Regeln Kommunikation zwischen 2 od. mehr Partnern.

= Regelwerk für d. Kommunikation.

z.B. HTTP regelt Kommunikation zw. Webclient und Webserver.

## NORMIERUNGEN

ISO (deutsch DIN).

IEEE.

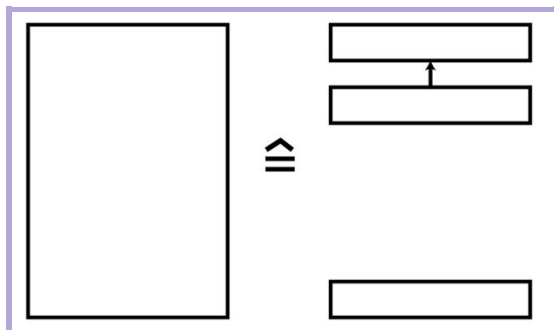
...

## Internet-Normierung

Drafts → RFC (Request for Comments) (Veröffentlichung aller RFCs: [www.w3.com](http://www.w3.com) ).

# SCHICHTENMODELLE

Komplexe Systeme werden in Schichten aufgeteilt.



## BEISPIELE

Protokolle, Betriebssysteme ...

## DOD-MODELLE (Department of Defense)

Beschreibt Internet-Protokoll in 4 Schichten.

Jedes Protokoll gehört in eines der 4 Schichten.



4 = hier gehört HTTP rein (Anwendungsprotokoll).

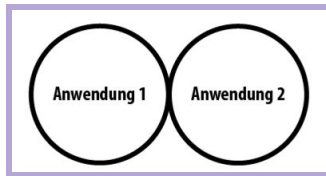
3 = hier gehört TCP rein (Transportprotokoll).

2 = hier gehört IP rein (Internetprotokoll).

1 = hier gehört Ethernet rein (physikalische Ebene).

## Anwendungs-Schicht

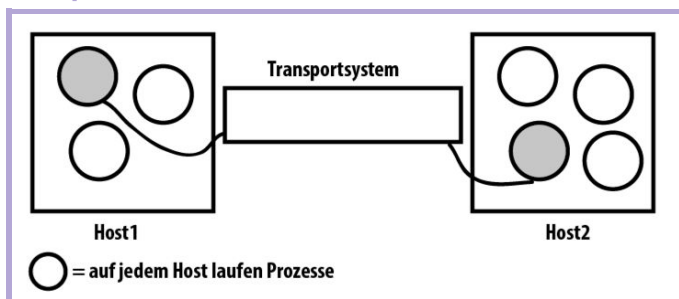
Wie Kommunikation technisch realisiert, wird hier nicht beschrieben.



### Beschäftigt sich mit den Fragen

1. Welche Aufgaben haben d. Anwendungen?
2. Wie sehen Benutzerschnittstellen aus ?
3. Wie gehen Anw. mit Datentypen um?  
(z.B. in Java = festgelegt wie groß int ist, in C nicht).
4. Wie kommunizieren Anw. miteinander, wie ist d. Reihenfolge, wer fängt an?  
(Kommunikationsreihenfolge).

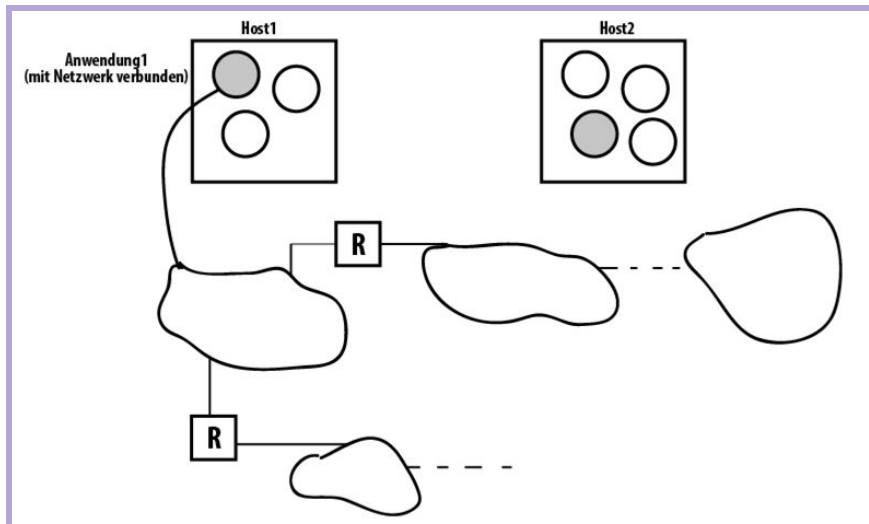
## Transport-Schicht



### Beschäftigt sich mit den Fragen

1. Wie findet eine Anw. d. Partner auf d. anderen Seite? (Ports).  
Ports = identifizieren Anw. auf Hosts (so findet Anw.1 auf Host1 Anw.2 auf Host2).  
Anw. müssen sich kennen (Portnummer), sonst keine Kommunikation möglich.
2. Wie sieht Transportsystem aus?  
Sicher od. unsicher transportieren (sicherer Transport = langsamer, wegen Überprüfungen).

## Internet-Schicht



### Beschäftigt sich mit den Fragen

1. Wie sehen Host-Adressen aus?
2. Wie werden Hosts adressiert?  
Streams od. Datagram-Service (wird im Internet verwendet, IP ist Datagram-Service).
3. Wie geschieht das Routing?
4. Schnittstelle zu physikalischen Netzen muss beschrieben werden.

## Netzzugang

Physikalische Ebene.

### Beschäftigt sich mit den Fragen

1. Was ist ein Bit?
2. Wie bekommt man Bit von rechts nach links? (Transport d. Bits).
3. Transport eines Pakets.
4. Wie geht an mit Kollisionsproblemen um?
5. Hardware-Adresse, wie werden Computer adressiert?
6. ...

## ISO/OSI

7-Schichten-Modell.

|   | OSI-Schicht |
|---|-------------|
| Anwendungsdienste (application layer)   | 7           |
| Darstellung (presentation layer)        | 6           |
| Kommunikationssteuerung (session layer) | 5           |
| Transport (transport layer)             | 4           |
| Vermittlung (network layer)             | 3           |
| Sicherung (data link layer)             | 2           |
| Bitübertragung (physical layer)         | 1           |

## Verlauf

Anwendung → OSI-7 → OSI-6 → ... → OSI-2 → phy. Paket → OSI-2 → ... → OSI-7 → Anwendung

## Beispiel

Webbrowser = get-Befehl → durchläuft d. OSI-Schichten, macht phy. Paket daraus, das durchwandert wieder alle Schichten, Server-Daten durchwandern wiederum wieder diese Schichten.

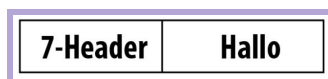
Anwendung → HTTP → TCP → IP → Ethernet → Ethernet → IP → TCP → HTTP → Anwendung

## Header

Sagen Empfänger, was er zu tun hat (z.B. Angabe v. Ports).

Anwendung schickt "Hallo"

1. OSI-7 setzt Header ein



2. OSI-6 setzt auch wieder Header ein



3. ...

## OSI-7

Anwendung → <http://www.beuth-hochschule.de/~brecht>  
macht daraus Get-Request = GET/usr/brechtHTTP/1.1

## OSI-6

Typen werden geprüft.

## OSI-5

Dialog, wer darf anfangen, wie kommt Dialog zustande?

## OSI-4

Fügt Ports ein, Anw. auf Hosts adressieren.

## OSI-3

IP-Adressen werden eingebaut.

## OSI-2

Checksummer auswerten.

## OSI-1

phy. Paket wird daraus gemacht.

## HTTP-Header

GET/~brechtHTTP/1.1

OSI-7.

Accept: text/html

OSI-6, akzeptiert nur das.

Connection: keep-alive

OSI-5.

keep-alive = Client nimmt Folge von Ergebnissen entgegen, statt der klassischen 'Ping-Pong'-Anfrage (z.B. Bilderfolge, denn jd. Bild einzeln senden = nicht so schön).

## Unterschiede DoD und ISO/OSI

Dod = Richtige Protokolle werden beschrieben.

ISO/OSI = Richtlinien an die Hersteller, daran sollten sie sich halten.

| <b>Dod-Modell</b> | <b>ISO/OSI-Modell</b>   |
|-------------------|-------------------------|
| <b>Anwendung</b>  | Anwendungsdienste       |
|                   | Präsentation            |
|                   | Kommunikationssteuerung |
| <b>Transport</b>  | Transport               |
| <b>Internet</b>   | Vermittlung             |
| <b>Netzzugang</b> | Sicherung               |
|                   | Bitübertragung          |



# TRANSPORTPROTOKOLLE

---

# TRANSPORTPROTOKOLLE

## AUFGABEN

Bilden OSI-Schicht 4 bzw. DoD-3-Ebene.  
= Verfeinerung der Anwendungs-Ebene

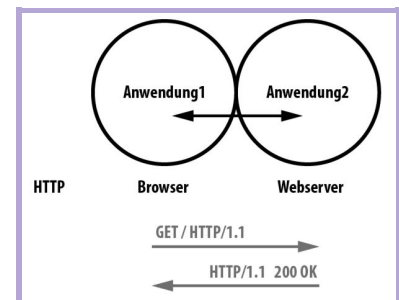
## ANWENDUNGSPROTOKOLL

Beschreibt = Anw. 1 + Anw. 2 = direkt miteinander verbunden und kommunizieren.

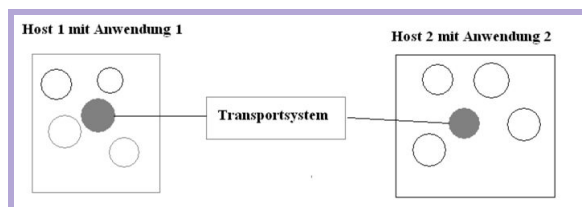
### Beispiel HTTP

HTTP: Anw.1 = Browser + Anw. 2 = Webserver

HTTP beschreibt nur was sie machen (einer schreibt, anderer antwortet), aber wie wird im Anwendungsprotokoll nicht beschrieben.



## TRANSPORTPROTOKOLL



Geht davon aus, dass es mind. 2 Hosts gibt.  
Anwendungen werden mit Transportsystem verbunden.  
TCP (starke Sicherungsmaßnahmen) oder UDP.

### Beschreiben

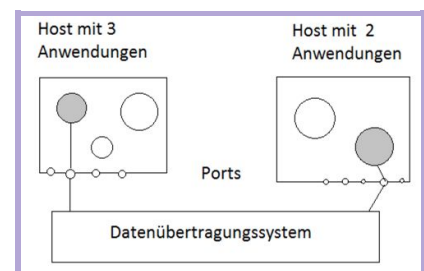
1. Anwendungs-Adressierung.
2. Wie sieht Transportdienst aus, d. zur Verfügung steht?
3. Anwendungsmultiplexing (Jd. Anw. hat ihr eigenes Anwendungsprotokoll, aber alle zusammen haben nur ein Transportprotokoll).

Im Internet = Punkt 1 + 2 stark ausgeprägt.

## ANWENDUNGS-ADRESSIERUNG

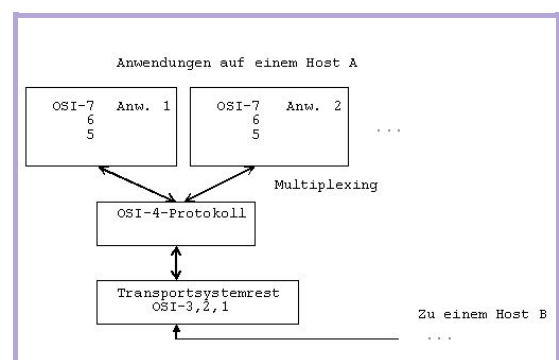
Adressierung durch Portnummern (= 16 Bit-Zahlen, pos. ganze Zahlen [0, 65535]).

Portnummer 0 = ausgezeichnet/reserviert (mit 0 wird Zugriff auf Port-Mapper angefordert, der dann willkürlich freien Port raussucht).



## OSI-4-MULTIPLEXING

Auf einem Host = möglicherweise mehrere Anwendungen.



Jd. Anw. hat ihre eigenen OSI-Schichten, **aber nur ein OSI4-Protokoll** (Heißt nicht, dass entweder nur TCP od. UDP laufen kann, auf versch. Anw. kann beides laufen).

## PORTS

IANA (Internet Assigned Numbers Authority) vergibt Portnummer, verwaltet Ports.

### PORTNUMMERN

(2x Vorhanden (TCP/UDP).

#### 0 ... 1023

= System Ports (Well known Ports).

Von d. IANA vergeben f. weit verbreitet Dienste.

Unter Linux zu finden unter `/etc/services`

#### Beispiele

Portnummern können mit admin-Rechte geändert werden.

|    |               |
|----|---------------|
| 20 | (TCP) SFTP    |
| 22 | SSH (TCP)     |
| 23 | Telnet (TCP)  |
| 53 | DNS (TCP/UDP) |
| 80 | HTTP (TCP)    |

#### 1024 ... 49151

= Registrierte- oder User- Ports.

Von d. IANA auf Anforderung reserviert.

Können z.B. Firmen registrieren, die dann kein Anderer nutzen kann.

#### 49152 ... 65535

(16-Bit -Feld, da größter Port, liegt im Protokoll-Kopf).

= Private Ports.

Frei verfügbar, nicht registriert.

## TRANSPORTDIENSTE IM INTERNET

Zwei Transportprotokolle = TCP (Stream Service) + UDP (Datagram Service).

Anw. müssen sich für Transportdienst entscheiden (entsprechend müssen sie Daten aufbereiten).

### BEISPIELE

File Transfer mit TCP = sinnvoll.

DNS mit UDP = sinnvoll (wäre nicht sinnvoll eine Röhre aufzubauen).

Anw., die Stream-Socket (ServerSocket, Socket) verwendet = für Stream-Service entschieden (muss Stream (Bytestrom) produzieren).

Anw., die (in Java) Klassen DatagramSocket + DatagramPacket verwendet = für Datagram-Service entschieden (muss Datagramme erzeugen).

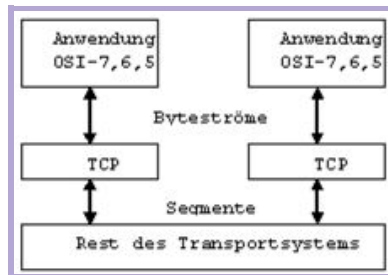
# TCP (Transmission Control Protocol)

Soll gesicherten Transport ermöglichen.

= vollduplex Stream-Service.

Zusammen mit IP = wichtigsten Protokolle im Internet.

TCP/IP = entsprechen Internet-Protokolle.



Beim Senden = TCP zerlegt Byte-Stream d. Anwendung in Segmentstream

(Sichern wird möglich → Checksummen bilden, zählen + schauen, ob diese ankommen).

Auf anderer Seite = TCP assembliert Segment-Stream zu Byte-Stream.

TCP arbeitet mit Bestätigungen (acknowledgements), bestätigt Bytes keine Segmente.

## SIMPLEX

Nur in eine Richtung,  $A \rightarrow B$

## DUPLEX (HALBDUPLEX)

Nur in eine Richtung, aber immer abwechselnd (alternierend).

$A \rightarrow B$

$A \leftarrow B$

## VOLLDUPLEX

Gleichzeitig in beide Richtungen.

$A \leftrightarrow B$

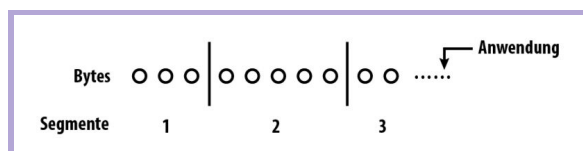
## BESTÄTIGUNGSNUMMER

...von a = Ich bestätige dir korrekten Erhalt aller Bytes bis einschließlich a-1.

Als Nächstes erwarte ich von dir den Byte Nummer a.

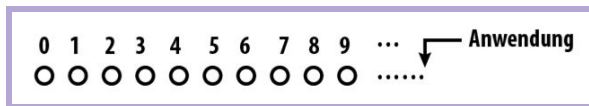
## SICHERUNG ÜBERTRAGUNGSSTROM

TCP nimmt Datenstrom entgegen + teilt diesen in Pakete (Segmente).



(voreingestellte Größe d. Segmente = 536 Bytes)

Beim Empfänger muss aus Segmenten genau d. gleiche Strom wieder reproduziert werden.  
Deshalb werden Bytes (nicht Segmente) d. Streams nummeriert.



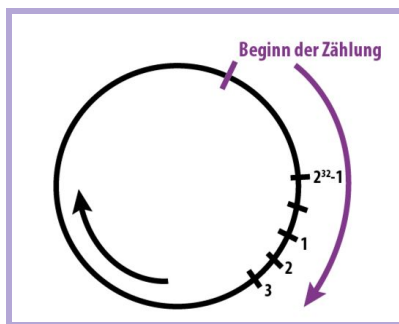
Kanonische Nummerierung

### NACHTEIL KANONISCHE NUMMERIERUNG

1. Stream wird gesendet, dann ein 2., der beginnt wieder bei 0.  
Wenn Erster noch nicht angekommen = Zahlen doppeln sich.

### Lösung

Zählung beginnt bei zufälliger Zahl um  $[0, 2^{32}-1]$



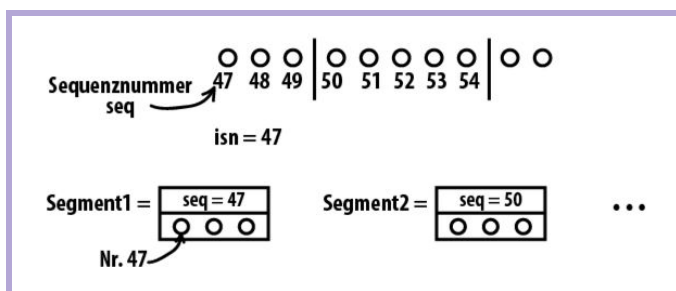
### Nachteil

Empfänger muss zufällige Zahl mitgeteilt werden, da er d. gleichen Stream erzeugen muss.

### SEGMENTE

#### Beginn der Zählung

Zahl b. d. Zählung beginnt = Initial Sequenz Number (isn)



Im 1ten Segment ist d. Nr. d. 1ten Datenbytes gleich d. isn.

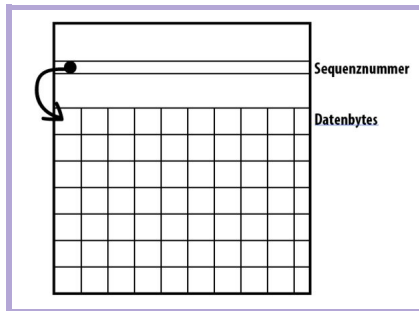
isn = zufällig gewählt.

Nummer steht im TCP-Header (32Bit groß).

In jd. Segmentheader befindet sich 32Bit-Feld f. Sequenznr. :  $0 \dots 2^{32}-1$

Gezählt wird  $\text{mod}(2^{32} - 1)$ , also im Kreis herum.  
Segmente = nur bei TCP.

### Segment



### Sequenznummer

= Nummer des ersten Datenbytes des Segments.  
Beim ersten Segment ist das d. isn.

### ACKNOWLEDGEMENTS

= Bestätigungen.

Bestätigt werden erhaltene Bytes (nicht Segmente).

Dafür = 32-bit-Feld in jd. Segmentheader (bezeichnet als ack = Wert).

Nicht jd. Segment enthält Bestätigung.

Muss signalisiert werden, dass Segment Bestätigung enthält.

Dafür = Kontrollbit im Header (ACK-Bit = 0 und 1, bei 1 = Bestätigung liegt vor).

### Festlegung

Steht im ack-Feld Wert  $a$  ( $\text{ack}=a$ ) + ist ACK gesetzt = alle Bytes, beginnend bei isn bis einschließlich der Nummer wurden korrekt empfangen.

Als nächstes wird Segment erwartet, dessen Sequenznummer  $\text{seq} = a$  ist.

### Berechnung Bestätigungsnummer

Berechnung durch Empfänger.

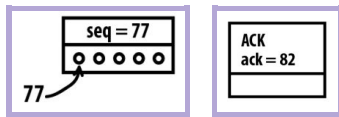
Empfänger eines Segments liest dessen Sequenznr.  $\text{seq}$  + zählt d. damit empfangenen Datenbytes ( $\text{anz}$ ). Dann bildet er:

$$\text{ack} = \text{seq} + \text{anz}$$

$\text{ack}$  = Bestätigungsnummer.

$\text{seq}$  = Sequenznummer.

$\text{anz}$  = Anz. d. Datenbytes.



### Beispiel

TCP-Station erhält Segment mit  $\text{seq} = 38$ .

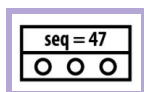
Datenbytes werden gezählt mit  $\text{anz} = 11$ .

$\text{ack} = 49$ .

Erwartet Segment, das mit Nummer 49 beginnt.

### ANDERE DARSTELLUNG

Statt



[ $\text{seq}=47$ , 5 bytes daten]

### Beispiel

[ACK,  $\text{ack}=82$ ,  $\text{seq}=105$ , 3bytes daten]

### AUSNAHMEN

Enthält Segment kein Anwendungsdatum, dann ist  $\text{anz} = 0$  und  $\text{ack} = \text{seq}$ .

$\text{ack} = \text{seq} + 0$

$\rightarrow \text{ack} = \text{seq}$

Von dieser Berechnung = 2 Ausnahmen.

In diesen Fällen = andere Rechnung ( $\text{ack} = \text{seq} + 1$ ).

### SYN-Bit und FIN-Bit

Im Segmentheader = weitere Kontrollbits (außer ACK-Bit).

Im empfangenen Segment = SYN- oder FIN-Bit gesetzt, wird  $\text{ack} = \text{seq} + 1$  gerechnet.

### SYN-BIT (synchronized)

Wird bei Verbindungs-Wunsch gesetzt.

Segment mit gesetztem SYN-Bit darf keine Daten enthalten.

Das ist das erste Segment in einer TCP-Kommunikation.

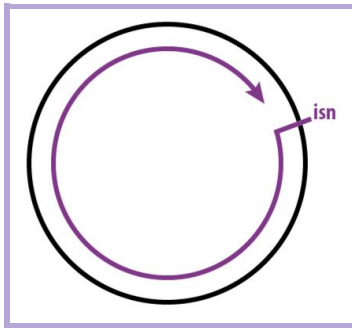
### Rechnung

1tes Segment = [SYN,  $\text{seq}=\text{isn}$ , no data] // geht von A  $\rightarrow$  B

B will bestätigen.

Würde falsch, nach d. Formel, [ACK,  $\text{ack}=\text{isn}$ , no data] rechnen

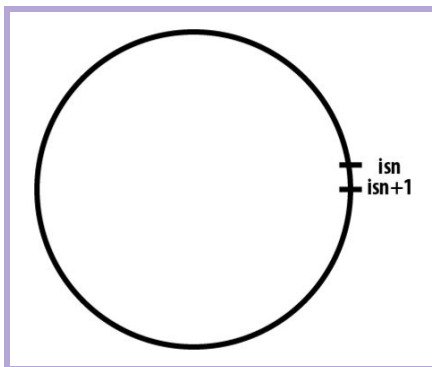
( $\text{ack} = a = \text{Alle Bytes ab isn bis } a-1$ ).



Würde dann alle Bytes bestätigen, obwohl er nichts bekommen hat.

### Lösung

[ACK, ack(alle Bytes v. isn-isn!) = isn+1, no data]



### FIN-BIT

Segment mit gesetztem FIN-Bit darf keine Daten enthalten.

### Situation

A schickt B Daten (z.B. Datei).

B hat für A keine Daten.

Transfer geht gerade zu Ende.

### Dann könnte sich ergeben

A → B : [ACK, ack = b, seq = a, 5 bytes data]

(letzten 5 Bytes, B weiss aber nicht dass es Letzten sind, ...bekommt das mitgeteilt siehe unten).

B → A : [ACK, ack = a+5, seq = b, no data]

A → B : [FIN, ack = b, seq = a+5, no data] (falsche Rechnung)

B → A : [ACK, ack = a+5, seq = b, no data] (Fehlersituation, Duplikat?)

Es gibt hier zwei identische Pakete (2. und 4.)

Bei 4. = Fehler (ack = a+5)

### Richtige Rechnung

B → A : [ACK, ack = a + 5+1, seq = b, no data]



### Warum +1?

Im Segmentkopf = mehrere Steuerbits.

Bei Verbindungs-Wunsch = SYN-Bit wird gesetzt (synchronized).

Dieses Segment darf keine Daten enthalten.

Es enthält aber d. isn. → also Paket das nur aus Header besteht.

### Beispiel

seq = 36 (SYN gesetzt , also keine Daten).

Empfänger rechnet

$$\text{ack} = \text{seq} + 0 \quad \text{f} = \text{seq} + 1$$

→ Ich bestätige dir den Erhalt aller Bytes bis einschließlich 35. Als nächstes erwarte ich von dir 36

→ macht aber keinen Sinn (deshalb = siehe rot).

### Sendungs-Stop

Will Station anderen mitteilen, dass sie Senden einstellt = schickt Segment ohne Daten, indem FIN-Bit gesetzt ist.

Empfänger würde falsch rechnen (da Daten leer) mit  $\text{ack} = \text{seq} + 0$

Muss rechnen  $\text{ack} = \text{seq} + 1$

### PRÜFSUMMENFEHLER

Empfänger bestätigt nicht.

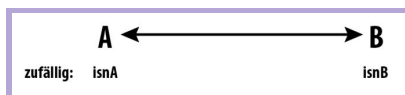
Sender setzt für jedes Segment einen Timeout.

Timeout abgelaufen, ohne dass Bestätigung eingegangen ist = Sendung wird wiederholt

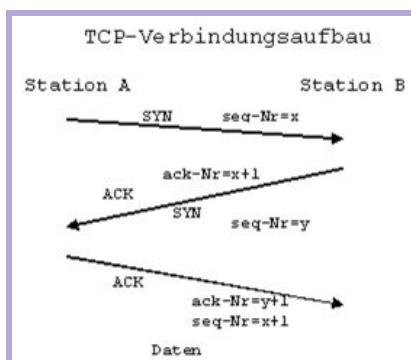
(Empfänger fordert keine Wiederholung, bestätigt einfach nicht).

### AUFBAU TCP-VERBINDUNG

#### 3-Wege-Handshake

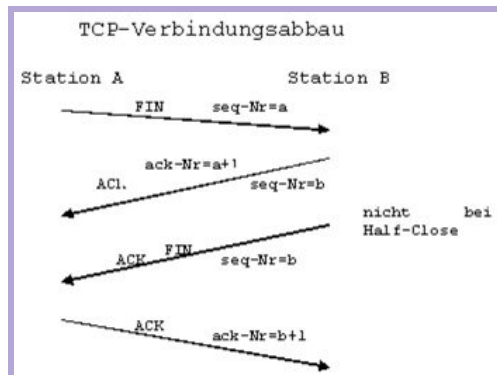


1. A → B [SYN, seq = isnA, no data]
2. B → A [ACK, ack = isnA+1, SYN, seq = isnB, no data]  
(Bestätigt heißt er muss ACK-Bit setzen)
3. A → [ACK, ...]



## ABBAU TCP-VERBINDUNG

Besonderheit: Half Close (Halb-Schließen), eine Station kann aufhören, die andere weitersenden, bis auch sie keine Daten mehr hat.



## TCP-HEADER

|                    |              |                |              |
|--------------------|--------------|----------------|--------------|
| Quellport          |              | Zielpport      |              |
| Sequenznummer      |              |                |              |
| Bestätigungsnummer |              |                |              |
| DO                 | Kontrollbits | Fenster        |              |
| Prüfsumme          |              | Wichtige Daten |              |
| Optionen           |              |                |              |
|                    |              |                | ev. Füllbits |

### Quellport/Zielport

je 16 Bit.

### Sequenznummer

[ seq = ... ] (32-Bit-Feld)

### Bestätigungsnummer

[ ack = ... ] (32-Bit-Feld)

### DataOffset (DO)

Zeiger auf das 1te Datenbyte.

Notwendig wegen der Optionen.

### Fenster (Sliding Window)

Für d. Flusskontrolle.

Verhindert Pufferüberläufe durch Kreditverfahren (Empf. gibt an, wie viel Platz er hat).

### Beispiel

Transport zw. langsamen + schnellen Rechner.

(Schickt langsamer Rechner dem schnellen Rechner = geht gut.

andersherum = schickt Rechner schneller als d. langsame verarbeiten kann).

### Prüfsumme

umfasst Header, Daten, Anfang des darunter liegenden IP-Pakets (das sind die beiden IP-Adressen!)  
= Pseudoheader

### Wichtige Daten

Gibt es wichtige Daten, müssen sie am Anfang d. Anw.Daten im Segm. stehen.  
Das Wichtige-Datum-Feld ist ein Zeiger auf das letzte Byte d. wichtigen Daten.

### Optionen

Können stehen, Müssen aber nicht.

= nicht genormt.

Fehlen meist.

Beispiele = Angaben zur max. Segmentgröße (536) od. Zeitstempel.

### Kontrollbits

#### URG-Bit

Falls gesetzt = Wichtige Daten vorhanden.

#### ACK

#### PSH (Push-Bit)

Falls gesetzt = Daten ohne Pufferung an Anw. geben.

#### RST

gesetzt= Neuaufbau-Wunsch.

#### SYN

"Synchronisiere dich mit mir."

#### FIN

## UDP (User Datagram Protocol)

Kann (weitgehend) ungesichert transportieren.

Daten kommen v. d. Anwendung zunächst in ein byte-Array.

UDP ergänzt dieses Array durch Header.

### ANWENDUNGS-BEISPIELE

DNS.

Fehlerprotokoll im Internet.

TFTP

### TYPISCH

Kein Verbindungsaufbau- bzw. abbau.

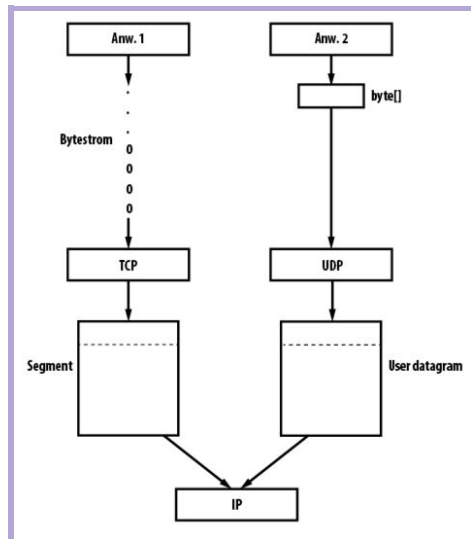
Keine Bestätigung.

Keine Zustellgarantie (das Datagramme ankommen, TCP garantiert das auch nicht, aber garantiert Meldung, wenn Segment nicht ankommt).  
Keine Reihenfolgegarantie.

## DATAGRAM

= Datenpaket, aber nicht jd. Datenpaket = Datagram.  
= Datenpaket mit Adressen v. Sender + Empfänger.

Will eine Anwendung TCP nutzen, muss sie nur Bytestrom erzeugen.



IP sieht nicht, ob es Segment od. Datagram ist.

## UDP-HEADER

Adressiert Anwendungen.  
Datagram-Länge = 16 Bit.  
Prüfzahlen werden wie b. TCP berechnet.  
Ports = 16 Bit zahlen (wie b. TCP).

| Byte Nr. | 0               | 1 | 2         | 3 |
|----------|-----------------|---|-----------|---|
|          | Quell-Port      |   | Ziel-Port |   |
|          | Datagramm-Länge |   | Prüfzahl  |   |

## NETSTAT-BEISPIEL

```
$ netstat
```

| TCP: IPv4                  |                              |        |       |             |
|----------------------------|------------------------------|--------|-------|-------------|
| Local Address              | Remote Address               | Snd    | Rcv   | State       |
| -----                      |                              |        |       |             |
| sun36.tfh-berlin.de.telnet | pcx75.tfh-berlin.de.1044     | 7950   | 24820 | ESTABLISHED |
| sun36.tfh-berlin.de.934    | mail.tfh-berlin.de.nfsd      | 145760 | 24820 | ESTABLISHED |
| sun36.tfh-berlin.de.933    | fileserv1.tfh-berlin.de.nfsd | 196608 | 24820 | ESTABLISHED |

An Rechner pcx75 mittels Telnet = Verbindung zu sun36.  
2. + 3. Zeile = Am Rechner gibt es Mail und File-Anbindung.

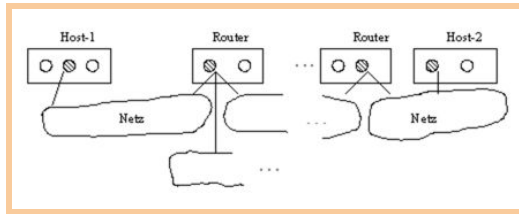
# VERMITTLUNGSSPROTOKOLLE

---

# VERMITTLUNGSSPROTOKOLLE

OSI-3-Ebene (Verbindung, Vermittlung) bzw. DoD-Schicht 2 (Internet).

Hostadressierung wird problematisiert (bei Transportprotokollen = Anwendungs-Adressierung).



Viele Netze + viele Router zwischen Hosts.

## AUFGABEN

### 1. Vermittlung zwischen Hosts

Wie werden Hosts adressiert?

(Logisches Netz: Alle Hosts im Verbund physikalischer Netze sind einheitlich adressiert).

### 2. Routing

Auf welchen Wegen kann d. Quelle (Anw.1) das Ziel erreichen?

Prinzipielle Festlegung: Fester Weg durch d. immer d. Daten fließen = Stream-Service oder Datagram-Service über verschiedene Router?

### 3. Fragmentierung/Assemblierung

Wie werden Daten (in welchen Einheiten) v. physikalischen Netz transportiert?.

Daten müssen fragmentiert (auf Empfänger-Seite defragmentiert/assembliert) werden.

## VERMITTLUNGSSPROTOKOLLE IM INTERNET

IP (v4 und v6) und ICMP.

| DoD OSI |   |   |  |  |  |  |  |  |  |
|---------|---|---|--|--|--|--|--|--|--|
| 4       | 7 | T |  |  |  |  |  |  |  |
|         | 6 | E |  |  |  |  |  |  |  |
|         | 5 | L |  |  |  |  |  |  |  |
|         | 4 | N |  |  |  |  |  |  |  |
| 3       | 3 | E |  |  |  |  |  |  |  |
|         | 2 | T |  |  |  |  |  |  |  |
|         | 1 | P |  |  |  |  |  |  |  |
|         | 0 |   |  |  |  |  |  |  |  |
| 2       | 3 |   |  |  |  |  |  |  |  |
|         | 2 |   |  |  |  |  |  |  |  |
|         | 1 |   |  |  |  |  |  |  |  |
|         | 0 |   |  |  |  |  |  |  |  |
| 1       | 3 |   |  |  |  |  |  |  |  |
|         | 2 |   |  |  |  |  |  |  |  |
|         | 1 |   |  |  |  |  |  |  |  |
|         | 0 |   |  |  |  |  |  |  |  |

Links = Einteilung nach DoD-Modell.

Daneben = OSI-Schichten.

Oben = Anwendungs-Protokolle.

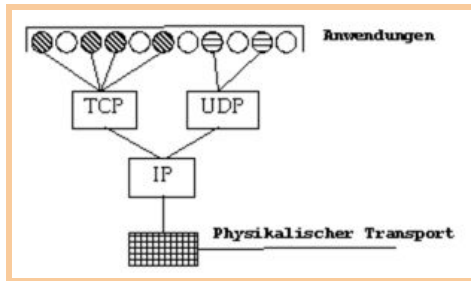
ICMP = Fehlermeldungen im Internet werden damit hin und her geschickt.

2 Vermittlungs-Protokolle vorhanden = IP + ICMP.

Ganz Unten = physikalische Netze.

## ZENTRALES INTERNET-PROTOKOLL

= IP.



## LOGISCHES NETZ

### IPv4-Adressen

4Bytes → Paket-Schreibweise.

### IPv6-Adressen

16 Bytes.

Zweite Adressenart = Domännennamen.

## ROUTING

Prinzipielle Festlegung = IP ist ein Datagram-Service  
(Unterscheidung = Userdatagram (UDP) + IP-Datagram (IP)).

## PAKETVERMITTELND

Alle gängigen physikalischen Netze sind paketvermittelnd.  
→ IP muss assembliert (und fragmentieren).

## IP-CHARAKTERISTIKA

### 1. Verbindungslos

Kein IP-Datagramm hat mit einem anderem etwas zu tun.

### 2. IP = unsicher

IP-Datagramme können verloren gehen, mehrfach vorkommen oder sich überholen.

Es gibt keine Bestätigungen.

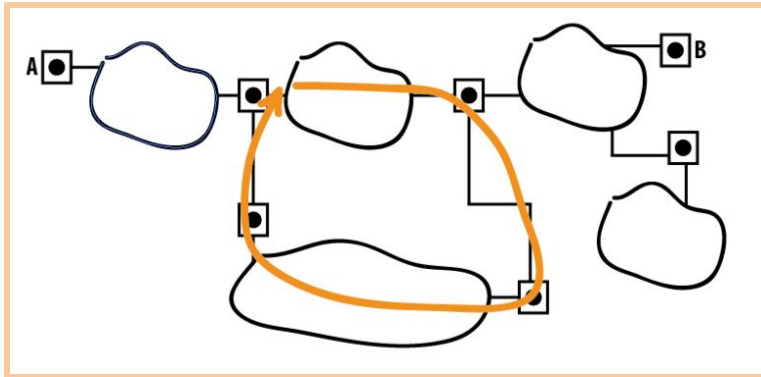
Bei Fehlererkennung b. Empfänger = Paket wird verworfen + Sender bekommt ICMP-Nachricht.

Keine Garantie d. Vollständigkeit einer Übertragung.

Es gibt keine Flusssteuerung.

## LEBENSDAUER

Daten könnten im Netzwerk endlos im Kreis laufen.



Deshalb = Lebensdauer.

Wird im Header eingefügt.

Immer -1.

Bei Lebensdauer = 0 können Daten dann nicht mehr verschickt werden.

Nachteil = kann irgendwo verloren gehen.

## IP

### HOSTADRESSIERUNG

#### IPv4

Benutzt 32-Bit-Adressen (nehmen Großteil des IP-Headers ein).

Das sind logische Adressen.

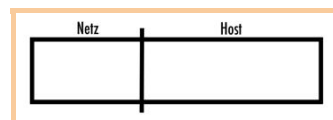
Mac-Adressen müssen herausgefunden werden (191.12.13.0 → Mac-Adresse?).

IP-Header mit Quell- und Zieladressen.

32 Bit → 0,...,  $4 \cdot 10^9$

### GLIEDERUNG

Jd. IP-Adresse ist gegliedert.



### Adress-Klassen

|   |     |       |     |       |
|---|-----|-------|-----|-------|
| A |     | 0     |     | 50%   |
| B |     | 10    |     | 25%   |
| C |     | 110   |     | 12,5% |
| D | ... | 1110  | ... |       |
| E | ... | 11110 | ... |       |

A = 1 Byte für Netze + hat für jd. Netz 3 Bytes für Hosts etc.

### KLASSE-A-ADRESSEN



Von Byte bleiben 7 Bits, somit =  $2^7$  Netzadressen (128).  
Es gibt keine Klasse-A-Adressen mehr.

### Belegte Netzadressen

1. Lauter 0 = zeigt an, dass Standard-Router, benutzt werden soll.
2. Lauter 1 = Loopback-Adresse (Anschaulich: Spiegelung am Medium).

→ In Klasse A können max. 126 Netze adressiert werden.  
Hosts in jd. A-Netz = 3 Bytes →  $2^{24}$  Adressen = 16.777.216

### Belegte Hostsadressen

1. Lauter 0 = Dann ist 32 Bit Adresse d. Netzadresse.
2. Lauter 1 = Broadcast-Adresse in diesem Netz.

### KLASSE-B-ADRESSEN

Netzadressen = 16.384  
Hosts pro Netz = 65.534

### KLASSE-C-ADRESSEN

Netzadressen = 2.097.152  
Hosts pro Netz = 254

### BEISPIEL (typisch)

141.6498.13  
Aus 141 wird Bitmuster gemacht.

$141/2 = 70 \rightarrow \text{Rest} = 1$

$70/2 = 35 \rightarrow \text{Rest} = 0$

...

141 = 10001101

→ Klasse B  
BHT ↔ 141.64.x.x

### SCHREIBWEISE

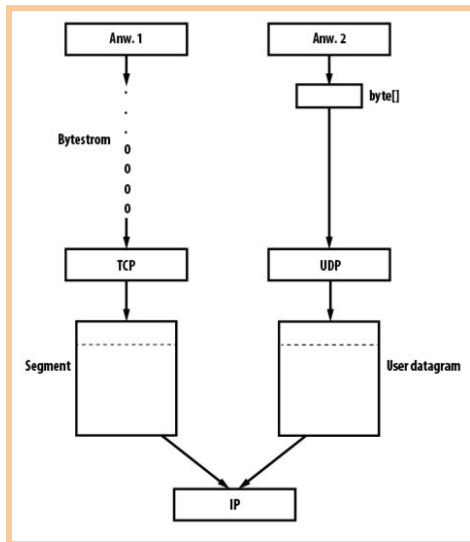
Dezimal, Punkt als Trenner.

### IPv4-HEADER

TTL = Lebensdauer (Time to Live).  
Version = v4 od. v6.  
IHL = Internet Header Length (4Bits)

| 4                | 4   | 8           | 16                            |
|------------------|-----|-------------|-------------------------------|
| Version          | IHL | Service-Typ | Gesamt-Länge                  |
| Identifikation   |     |             | 3 Flags    Fragment-Offset 13 |
| TTL              |     | Protokoll   | IP-Header-Prüfsumme           |
| IP-Quell-Adresse |     |             |                               |
| IP-Ziel-Adresse  |     |             |                               |
| Optionen         |     |             | Füllzeichen                   |

## AUFGABEN VON IP



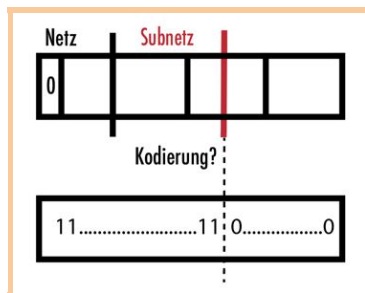
IP sieht nicht, ob es Segment od. Datagram ist (sieht die Header nicht).  
 IP bekommt Datenpaket + muss es vermitteln (von Host zu Host).

## SUBNETZE

IP-Adressraum ist längst erschöpft.  
 Lösung IPv6 16-Byte-Adressen.

## Maßnahmen

1. Subnetze = A- und B-Adressen werden weiter gegliedert.
2. Adressübersetzung.



## Lösung

= Subnetzmaske  
 1 = bei Netz + Subnetz.  
 0 = bei Host.

## Beispiel

Unix/Linux = ifconfig  
 Win = ipconfig

Bei BHT-PC → IP-Address = 141.64|.89.11  
 Subnetzmaske = 255.255|.255.128

Beim Arbeiten mit Subnetzen, muss die Maske mit angegeben werden.  
 141.64.89.11 | 255.255.255.128

## Andere Schreibweise

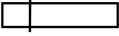
CIDR (Clienten Inter-Domain Routing).

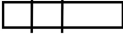
IP-Adresse / Zahl der 1 aus der Maske

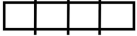
141.64.89.11 / 25

Angestrebt wird Abschaffung der Adressklassen.

## Idee

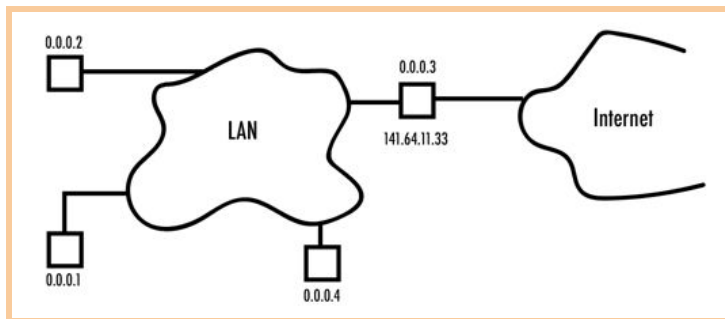
IANA vergibt  an Provider.

Provider vergibt  an Kunde.

Kunde vergibt  an Abteilung.

## NAT: NETWORK ADDRESS TRANSLATION

### Idee



| Externer Host | Externer Port | Lokaler Host | Lokaler Port | Protokoll | NAT-Port |
|---------------|---------------|--------------|--------------|-----------|----------|
| 191.36.17.12  | 80            | 0.0.0.1      | 21023        | TCP       | 10101    |
| 191.36.17.12  | 80            | 0.0.0.2      | 33709        | TCP       | 10102    |
| ...           | ...           | ...          | ...          | ...       | ...      |

## IP-FRAGMENTIERUNG

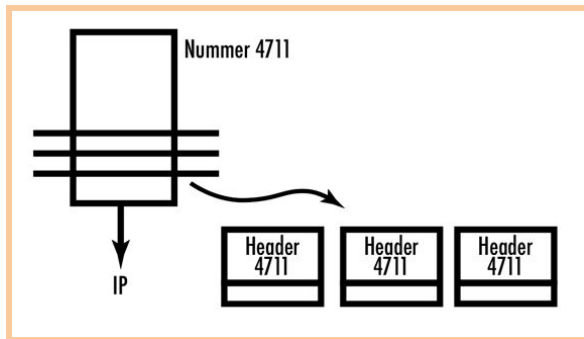
Zu IP gelangt Datenpaket.

IP sieht Bytefolge.

Jedes von OSI-4 kommende Paket erhält eine laufende Nummer (16-Bit-Zahl  $\in 0, \dots, 65.535$ ).

Diese Nummer steht als Identifikation im IP-Header.

IP zerlegt Daten + setzt jeweils Header drauf.



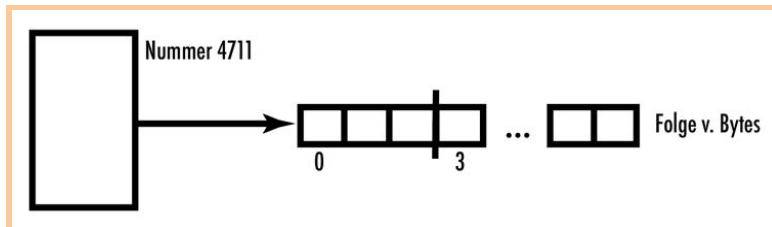
Erst der Ziel-Host assembliert.

Fragment Offset = 13 Bits.

Jd. Offset ist ein Vielfaches von 8.

### Beispiel

Offset = 3 → sein Byte<sub>1</sub> ist Byte Nr. 24 in der ursp. Folge.



### 3 FLAGS IM HEADER VORGESEHEN

Nummer...

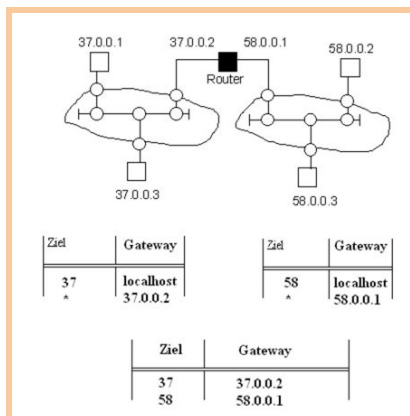
1 = unbedingt.

2 = Don't fragment (1= Paket darf nicht fragmentiert werden, 0= Fragmentierung erlaubt).

3 = More fragments (1 = es kommen weitere Fragmente, 0=ich bin d. letzte Fragment zur Identifikation (4711)).

## IP-ROUTING

Vermittlung v. IP-Paketen zw. IP-Netzen



Oben = 2 Netze

Jd. Host hat Routing-Tabelle.

| Netzwerkziel     | Netzwerkmaske   | Gateway      | Schnittstelle | Anzahl |
|------------------|-----------------|--------------|---------------|--------|
| 0.0.0.0          | 0.0.0.0         | 141.64.89.1  | 141.64.89.11  | 20     |
| 127.0.0.0        | 255.0.0.0       | 127.0.0.1    | 127.0.0.1     | 1      |
| 141.64.89.0      | 255.255.255.128 | 141.64.89.11 | 141.64.89.11  | 20     |
| 141.64.89.11     | 255.255.255.255 | 127.0.0.1    | 127.0.0.1     | 20     |
| 141.64.255.255   | 255.255.255.255 | 141.64.89.11 | 141.64.89.11  | 20     |
| 224.0.0.0        | 240.0.0         | 141.64.89.11 | 141.64.89.11  | 20     |
| 255.255.255.255  | 255.255.255.255 | 141.64.89.11 | 141.64.89.11  | 1      |
| Standardgateway: |                 | 141.64.89.1  |               |        |

Anzahl = Anzahl der Hops (wie viele Router dazwischen sind).

## ROUTING-BEISPIEL

Vorhaben = Host 37.0.0.1 möchte an Host 58.0.0.2 'Hallo' schreiben.

### Ausgangs-IP-Paket

|   |
|---|
| Von IP:<br>37.0.0.1<br>An IP : 58.0.0.2 |
| Inhalt: Hallo                           |

37.0.0.1 konsultiert seine Routing-Tabelle.

### Er sieht

|   |          |
|---|----------|
| * | 37.0.0.2 |
|---|----------|

37.0.0.1 erstellt Paket folgender Art =

Von MAC: Seine Hardware-Adresse (liest seine Netzwerkkarte aus)

An MAC : Hardware-Adresse von 37.0.0.2 (per Broadcast)

|   |
|---|
| Von MAC: Seine Hardware-Adresse<br>An MAC : Hardware-Adresse von 37.0.0.2 |
| Von IP: 37.0.0.1<br>An IP : 58.0.0.2                                      |
| Inhalt: Hallo   |

37.0.0.2 erhält dieses Paket.

### Macht daraus

|  |
|--|
| Von MAC: von 37.0.0.2<br>An MAC : an IP : 58.0.0.2 |
| Von IP: 37.0.0.1<br>An IP : 58.0.0.2               |
| Inhalt: Hallo                                      |

→ Kommt zu 58.0.0.2 (IP-Paket wird nicht verändert).

## ICMP INTERNET CONTROL MANAGE PROTOCOL

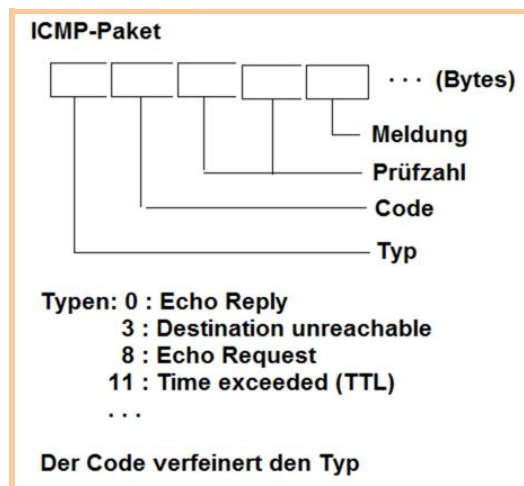
Header hat Fehler für Fehlermeldungen.

Deshalb wurde eigenes Fehlermeldungsprotokoll entwickelt.

### ANWENDUNGEN

1. Durch Herunterzählen des TTL-Feldes wurde 0 erreicht → Time executed → Sender
2. Ein Host ist nicht erreichbar.  
→ ICMP-Meldungen.

### ICMP-PAKET



### KOMMANDOSCHNITTSTELLE

Hat eingeschränkte Kommandoschnittstelle.

→\$ ping Host //ICMP-Echo-Request

→\$ man ping

### JAVA PING

```
import java.net.*;
...
String ziel = "www.tu-berlin.de";
InetAddress addr = InetAddress.getByName(ziel);
if(addr.isReachable(60))
```

```

    System.out.println("Host "+ziel+" erreichbar");
else
    System.out.println(ziel+" nicht erreichbar");

```

## TRACE ROUTE

Damit kann man Spur durchs Internet ziehen (Prinzip: TTL auf 1 setzen, dann TTL auf 2 setzen, etc.).

## Linux

tracert

## Win

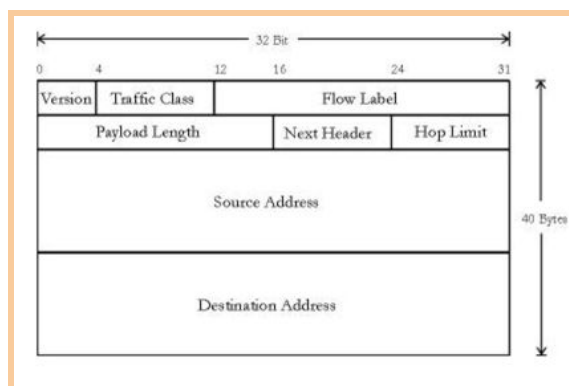
tracert

# IPv6

## ADRESSARTEN

1. **Unicast-Adressen (Hostadressen)**  
Gruppe, geht an genau einem in der Gruppe (der besser zu erreichen ist).
2. **Multicast-Adressen (speziell: Broadcast → alle)**  
Gruppe, geht an jd. in der Gruppe.

## HEADER



Traffic Class = wie Service-Typ bei IPv4.

Hop Limit = TTL-Feld.

## ADRESSEN

16 Bytes (in 2Bytes, hexadezimal, mit ':' als Trenner).

## Beispiel

4043:00BC:0000:00A4:0267:01FF:FE01:7352

→ DNS wird an Wert gewinnen.

## Adressen mit Portangabe

[Adresse]:port

## Sonderadressen

### v4-Einbettungen

Mit v4 beginnen:

139.80.34.5 → 8B.50.22.5 → 8B50:2205 → Links mit 0 auffüllen → ::8B50:2205

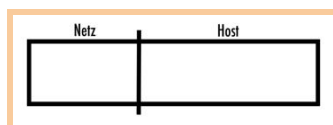
### Loopback-Adresse

v4 = 127.0.0.1

v6 = ::1

## ADRESS-STRUKTURIERUNG

Wie bei v4 strukturiert.



Bei v4 = Trennung mit Netzwerkmaske.

Bei v6 = Adresse/Länge der Netzkomponente.

v6-Adressraum = so groß, dass Hardware-Adressen eingebettet

werden können.

Anfang: FE80 ... FEBF (dahinter 0, dahinter MAC-Adresse → jeweils mit Doppelpkt. getrennt).

## ICMP

Für v6 = ICMPv6 (macht das Selbe wie ICMP bei v4).

## DOD UND ISO/OSI

Im DoD-Modell liegt unter IP Netzzugangsschicht.

Dafür 2 Schichten im ISO/OSI-Modell.



### LLC (Logical Link Control)

Oft fehlt LLC.

### Beispiel

Früher Ethernet.

OSI-2 entspricht MAC.

Spätere Ethernet-Versionen bekamen LLC.

OSI-3 = produziert Daten.

OSI-1 = überträgt Daten bitweise. Je nach Netz wird in OSI-1 ggf. nicht kontrolliert, ob 1 → 1? (1 auch wirklich ankommt).



## MAC (Medium Access Control)-SUBSCHICHT

Regelt auch Zugriff auf Medium.

Es gibt kollisionsbehaftete Medien (z.B Funk → kann sich überlagern).

(MAC regelt Umgang mit Kollisionsgefahr).

Bildet phy. Pakete, die 'Rahmen' heißen.

## Verfahren bei Kollisionsgefahr

### 1. Polling

Zentraler Knoten (Master) fragt untergeordnete Knoten im Netz, nach best. Muster, ab.

Untergeordnete Knoten dürfen nicht von sich aus senden

(Quelle: <http://yukon.e-technik.tu-ilmenau.de/~webkn/Webdaten/Lehre/WS2015/Kommunikationsnetze/V05-Medienzugang.pdf>).

(Einer wird ausgezeichnet alle anderen müssen Klappe halten).

Nachteil = Was, wenn Rechner einfach ausgeschaltet wird?

### 2. Token Passing

Prinzip = Staffellauf.

Nachteil = Tokens können einfach erstellt werden od. wegfallen.

### 3. Kollisionen in Kauf nehmen

z.B. Ethernet, WLAN, etc. → Hat sich eher bewährt.

## Typisches Rahmenformat

| Header | LLC-Felder | OSI-3-Daten | Trailer |
|--------|------------|-------------|---------|
|--------|------------|-------------|---------|

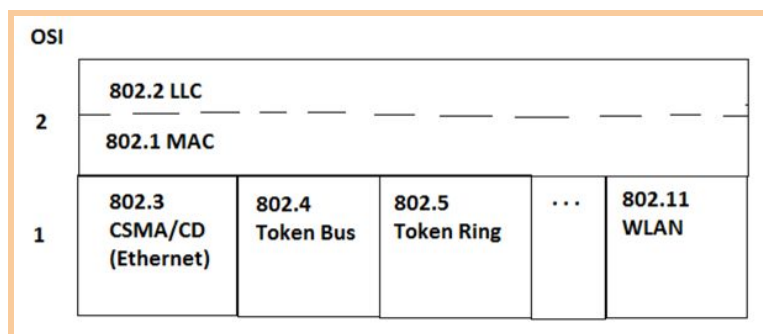
MAC-Schicht sieht IP-Adresse.

Ruft ARP (Address Resolution Protocol) auf, um Hardware-Adresse zu erhalten.

ARP führt kleine (~10 Einträge) Tabelle der bisherigen Ermittlungen.

→ \$ arp -a

## IEEE-Normen



# KOMMUNIKATION

---

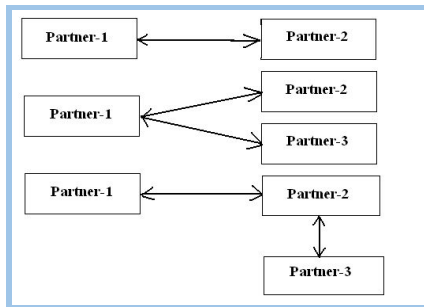
# KOMMUNIKATION

= Austausch v. Nachrichten (Daten).

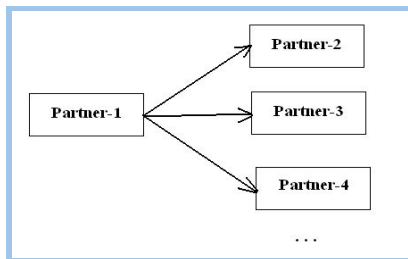
Daten um Informationen zu kodieren.

Unterscheidung von Kommunikation zw. Partnern und Broadcast-Kommunikation.

## INDIVIDUALKOMMUNIKATION

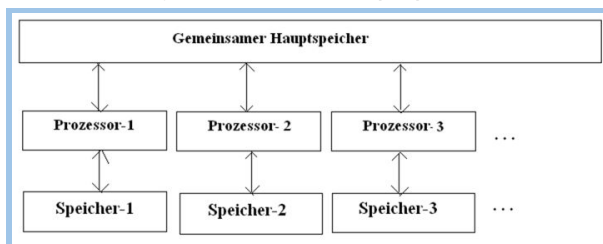


## MASSENKOMMUNIKATION



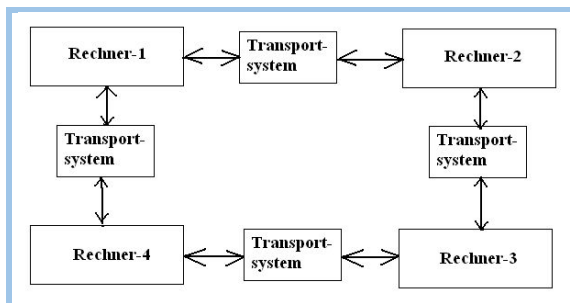
## ENGE KOPPLUNG

Können über gemeinsamen Hauptspeicher kommunizieren.



## LOSE KOPPLUNG

Nachrichtentransport in einem Netzwerk.



## UN SICHERER KANAL

Unsicher, ob Nachricht sicher ankommt (z.B. können IP-Pakete verloren gehen → IP = unsicherer Kanal).

Daten können sich verzögern, können dupliziert werden oder ein Datum kann auch ein anderes überholen.

### MÖGLICHKEITEN

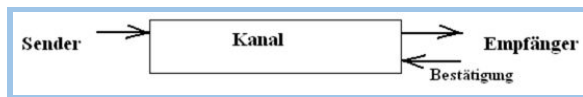
1. Idealfall.



2. Kanal verschluckt Nachricht.



3. Kanal verschluckt Bestätigung.



### 2-ARMEEN-PROBLEM

Annahme = Armee mit meisten Soldaten gewinnt.

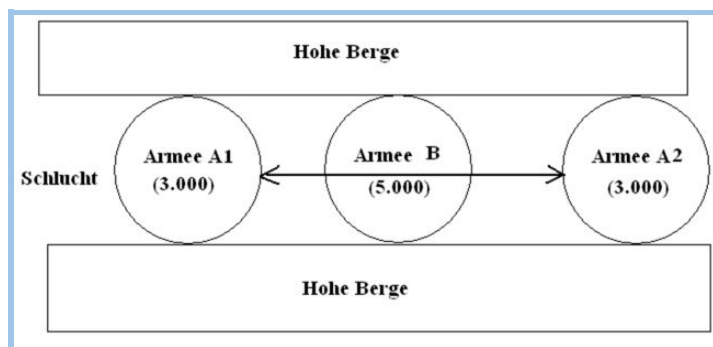
Armee A1 und A2 = nur Chance, wenn sie sich zusammen schließen (dann 6000 zu 5000).

Bote A1 durch unsicheren Kanal (Armee B) zu General A2 (morgen um 8 angreifen).

Wieder zurück durch unsicheren Kanal.

A2 kann aber nicht wissen, ob Bote A1 durchgekommen ist und kann nicht angreifen.

A1 kann auch nicht, da sie wissen, dass A2 nicht sicher sein kann.



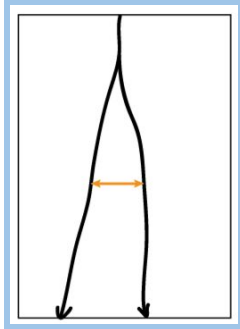
# KOMMUNIKATIONSPARTNER

## INTERTHREAD-KOMMUNIKATION

Inter = Zwischen.

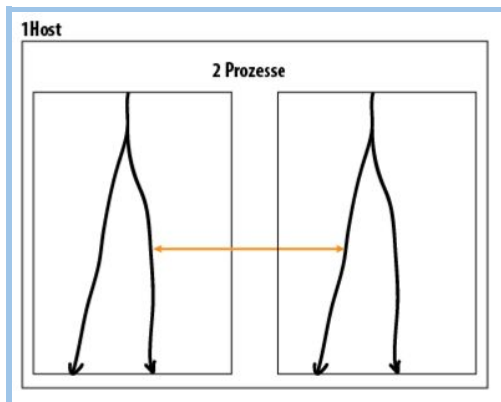
Prozess-Lokale Kommunikation.

Threads möchten miteinander kommunizieren.



## LOKALE INTERPROZESS-KOMMUNIKATION

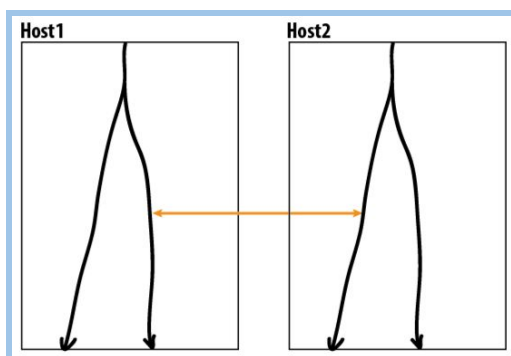
Threads d. beiden Prozesse möchten kommunizieren.



## FERNE INTERPROZESS-KOMMUNIKATION

Netzwerk-Kommunikation.

Thread d. einen Hosts möchte mit Thread d. anderen Hosts kommunizieren.



# SOCKETS

In allen Kommunikationsmodellen möglich.

## INTERPROZESSKOMMUNIKATION

Betriebssystem muss einbezogen werden, da es Prozesse verwaltet (schützt u.a. vor unberechtigten Zugriffen).

Ähnlich wie: Prozess geht zu Betriebssystem: 'ich möchte das machen, darf ich das?'.

### Realisierungs-Techniken

Shared Memory (weit verbreitet).

Pipes.

Message Passing.

## VORAUSSETZUNG

TCP/IP (= Internet protokolle) vorhanden.

## REALISIERUNG

Röhre (Pipe) zw. d. Threads (Stream-Sockets), durch die Datenströme fließen.  
oder

Paket-Dienst (Datagram-Sockets).

# SHARED MEMORY BEI PROZESSEN

= schnelle Datenübertragungs-Technik.

Prozess 1 + 2 haben d. selben Host.

Von Betriebssystem eingerichtet.

Hat Zugriffsberechtigungen f. Prozesse.

Gaukelt Prozessen vor, Shared Memory wäre Teil seines Speicher-Bereichs → Schreibkonflikte möglich.

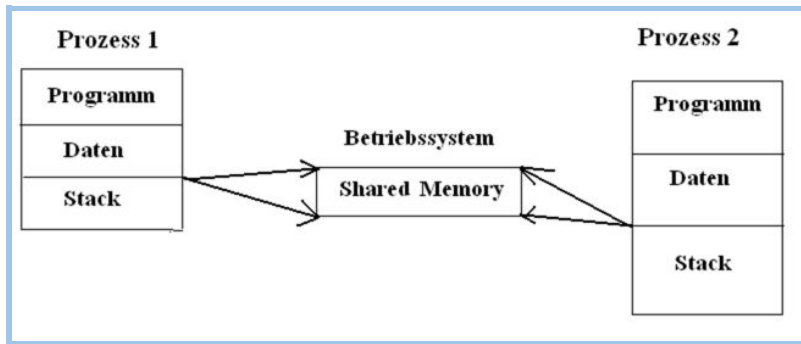
Prozess zu Betriebssystem: gib mir Shared Memory BS: ja aber nur du, kommt d. nächster Prozess + möchte auch Shared Memory, Betriebssystem fragt anderen Prozess, sagt ja und benachrichtigt fragenden Prozess. In Realität mittels Schutz-bits

Prozess legt Shared Memory 'abc' an Prozess fordert über BS auch Shared Memory 'abc' an (beide Prozesse merken nicht, dass sie sich Speicherplatz teilen).

Schreibkonflikte möglich.

Keine Synchronisation möglichkeiten.

Schnelle Kommunikations-Technik.



## PIPE

= Gerichtete Röhre.

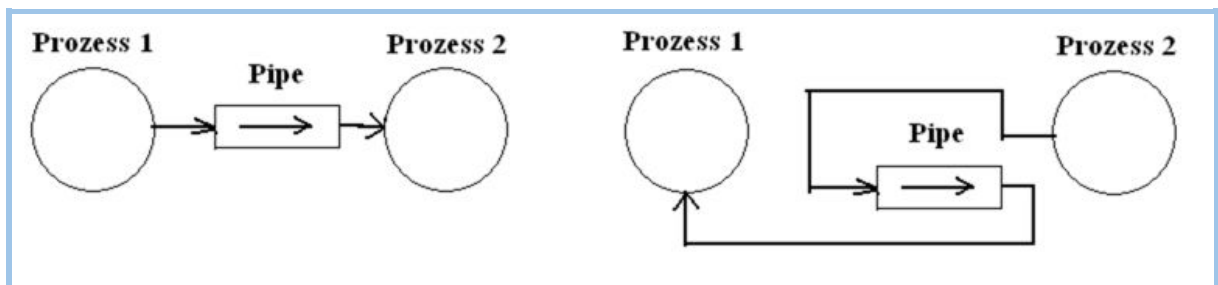
Datenübermittlung über Pipe.

Nutzen Shared Memory mit Synchronisation.

Blockiert Read-Aufruf wenn leer und write wenn voll ist.

Ein Prozess legt Pipe an.

Ein Prozess kann lesen und anderer schreiben.



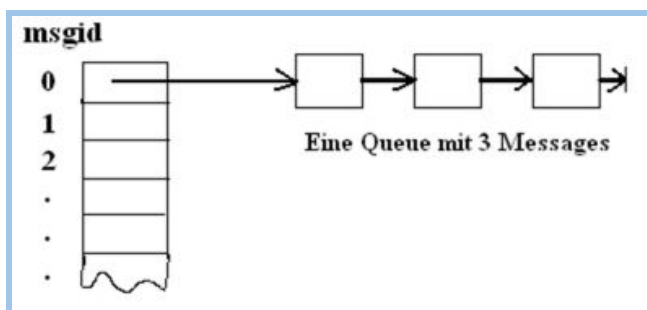
## MESSAGE PASSING

= asynchron

Betriebssystem realisiert Warteschlangen mit Nachrichten.

Jd. Prozess hat Slot von Nachrichten.

Prozess d. lesen will nimmt d. älteste Nachricht od. Nachricht mit best. Nummer.



In jd. Nachricht = Nummer.

Leser kann damit minimal selektieren (z.B. gib mir Nachricht aus Slot 0 mit d. Nr. 7).

### Beispiel

Wenn p1 an p5 schickt = man sagt p5 dass er nach nachr . mit nr. 5 und man muss sagen in welchen slot seine nachr stehen.

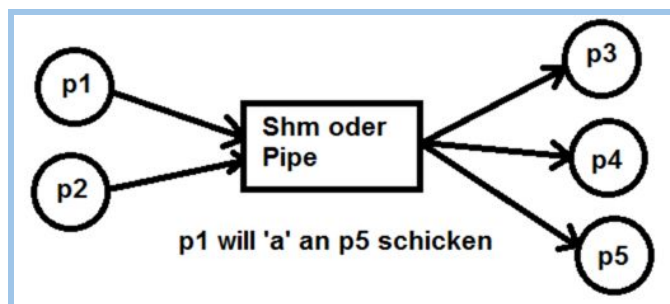
### Realisierungs-Sprache

C, C++

Kann man in Java einbinden.

Prozessraum besteht aus Regions, Codereafion date region stackregion

### PROBLEM BEI SHARED MEMORY UND PIPES



P5 möchte an p1 'a' schreiben, wie soll das funktioniert?

Nachteil = man weiss nicht wer Nachricht liest.

Vielleicht Marker: Absender = 1, Empfänger p5, Nachricht = 'a' → aber man weiss immer noch nicht ob es wirklich gelesen wird.

### SHARED MEMORY IN JAVA-THREAD

```
class Mem {    // Eine int-Variable als Shm
    int i;
    Mem(int param) {
        i=param;
    }
}

class MyThread extends Thread {
    Mem m;
    MyThread(Mem param) {
        m=param;
    }
    public void run() {
        m.i = m.i + 5;
    }
}

class Shm {
    public static void main(String[] args) throws Exception {
        Mem m = new Mem(3);
```



```

    MyThread mt1 = new MyThread(m);
    MyThread mt2 = new MyThread(m);
    mt1.start();    // Beide Threads greifen auf das
    mt2.start();    // gemeinsame m.i zu
    mt1.join();
    mt2.join();
    System.out.println(m.i);
}
}

```

'i' wird Shared Memory.

Manchmal 8, manchmal 13 ...wenn Wert überschrieben wird → nicht vorhersagbar.

Lösung: Immer Referenz übergeben, damit kein Schreibkonflikt.

### SHARED MEMORY IN JAVA-THREAD

Thread zu Thread – Pipe innerhalb d. Prozesses.

Mit PipedReader und PipedWriter.

```

class Work {
    public static void main(String[] args) {
        // main richtet eine Pipe ein
        // -----
        PipedReader pr = new PipedReader();
        PipedWriter pw = new PipedWriter();
        try {
            pr.connect(pw);
        } catch (Exception e) {
            System.out.println("Work: connect-Error");
        }
        // main startet die beiden Threads
        // -----
        Schreiber sch = new Schreiber(pw);
        Leser les = new Leser(pr);
        sch.start();
        les.start();
    }
}

// =====

// Datei: Schreiber.java
// Thema: Ein Thread schreibt einen String in eine Pipe
// -----
import java.io.*;
class Schreiber extends Thread {
    String msg = "Hallo lieber Leser";
    BufferedWriter bw = null;
}

```

```

        Schreiber(PipedWriter b) {bw=new BufferedWriter(b);}

    public void run() {
        System.out.println("Schreiber: schreibe in die Pipe");
        System.out.println("Schreiber: Text->" + msg);
        try {
            bw.write(msg); bw.newLine(); bw.flush();
        } catch(Exception e) {
            System.out.println("Schreiber: write-error");
        }
    }
}

// =====

// Datei: Leser.java
// Thema: Ein Thread liest einen String aus einer Pipe
// -----
import java.io.*;
class Leser extends Thread {

    BufferedReader br = null;
    String          s = null;

    Leser(PipedReader a) { br = new BufferedReader(a); }

    public void run() {
        System.out.println("Leser: lese aus der Pipe");
        try {
            s = br.readLine();
        } catch(Exception e) {
            System.out.println("Leser: read-error");
        }

        System.out.println("Leser: habe gelesen->" + s);
    }
}

```

## KOMMUNIKATIONSMODELLE

Abbild eines realen Gegenstandes auf reduzierte Darstellung.

### CLIENT/SERVER – MODELL

= Asymmetrisch (Client macht was anderes als Server).

Client fordert Dienst d. Servers an.  
Im Internet weit verbreitet.  
Client wendet sich an Server.  
2 Prozesse.

### Beispiel

Server kann drucken, Client möchte drucken.

### PEER-TO-PEER-MODELL

= Symmetrisch.  
Für gleichberechtigte Partner.

### Beispiel

Windows = Freigabe von Verzeichnissen.

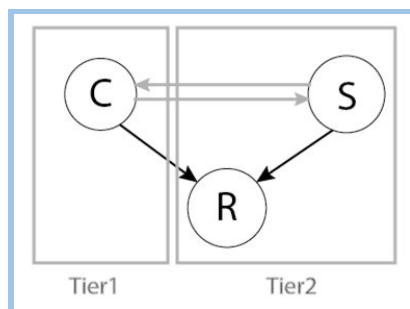
### MODIFIKATION

#### Client/Registry/Server-Modell

Client wendet sich an Registry.  
3 Prozesse (Server, Registry, Client).  
Server wendet sich an Registry + erzählt was er kann.  
Client wendet sich an Registry und fordert best. Dienst.  
Registry sucht dann den passenden Server.

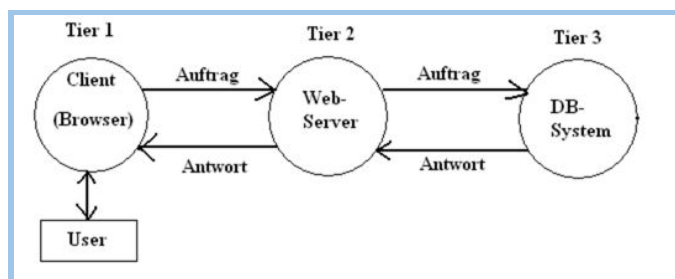
### Problem

Mehrere Server =  
Server 1 kann das und Server 2 kann das auch, Was soll Registry nun machen?



= 2 Tiers-Modell

### 3-TIERS-MODELL



Datenbank-Client tut so als wäre er ein Server.

## TCP

Braucht einen möglichst sicheren Kanal.

### FERNE INTERPROZESS-KOMMUNIKATION

Thread d. einen Hosts möchte mit Thread d. anderen Hosts kommunizieren.

# PROGRAMMIERUNG VS

---

# PROGRAMMIERUNG VERTEILTER SYSTEME

## ABSTRAKTE EBENE

Remote Procedure Calls (RCPs).

## KONKRETE EBENE

Sockets.

## RPC (REMOTE PROCEDURE CALL)

= Internetprotokoll.

Ermöglicht Aufruf v. Funktionen in anderen Adressräumen.

Gehört zur OSI-Schicht 5 (Kommunikationssteuerung).

Arbeitet synchron (Client arbeitet während Server schläft, arbeitet Server schläft Client).

## IDEE

```
main (...) {  
    ...  
    y = sin(3.7);  
    ...  
}
```

Library berechnet sin().

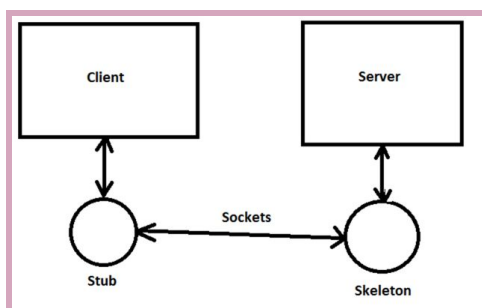
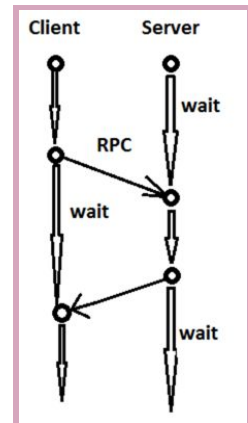
Keine Socket-Verbindung.

## Bibliothek

Library berechnet sin().

Hier wird Socket-Verbindung aufgebaut, weg v. Programmierer.

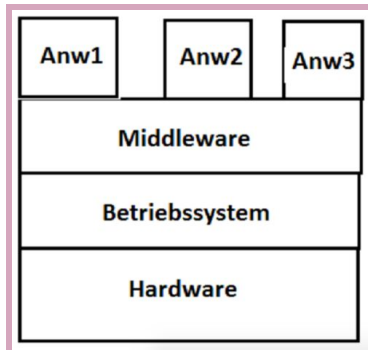
```
sin(alpha) {  
    sende x zum Server;  
    Warte auf ergebnis  
    return ergebnis;  
}
```



Compiler macht daraus Socketmechanismus.

## MIDDLEWARE

Programm = sowohl Client als auch Server.



## RMI (REMOTE METHOD INVOCATION)

= Technologie von Java zum entfernten Aufruf von Methoden und Zugriffsvariablen.

Benutzt Sockets.

Arbeitet mit Registries.

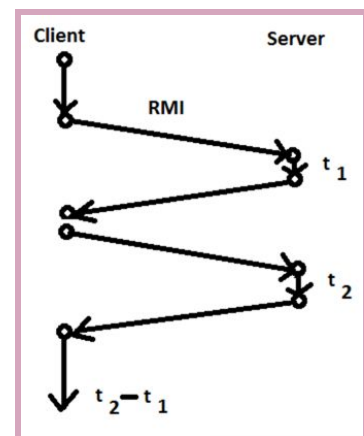
### BEISPIEL

Client lässt sich 2x Uhrzeit vom Server geben und berechnet Differenz der beiden Zeiten.

Dann weiss man, wie lange ein RPC gedauert hat (etwa 20 ms).

### RMI-INTERFACE

Im Interface deklariert und im Server wird die Methode dann definiert/implementiert.



```
public interface ZeitInterface extends Remote {  
    java.util.Date getDate() throws RemoteException;  
}
```

### RMI-SERVER

```
public class ZeitServer extends UnicastRemoteObject  
    implements ZeitInterface {  
  
    public ZeitServer() throws RemoteException {}  
  
    public Date getDate() throws RemoteException {  
        return new Date();  
    }  
  
    public static void main(String[] args) throws Exception {  
  
        LocateRegistry.createRegistry(1099);    // Port 1099=DEFAULT  
        ZeitServer zeit = new ZeitServer();  
    }  
}
```

```

// Anmeldung des Dienstes mit
// rmi://Serverhostname/Eindeutige Bezeichnung des Dienstes
// -----

//Rebind, da bei bind() der Port blockiert
//Rechner namens 'compute' und Server namens 'MyService'

    Naming.rebind("rmi://compute/MyService", zeit);
    System.out.println("Server wartet auf RMIs");
//An dieser Stelle entsteht Enlosschleife, weil Server auf RMIs wartet
}
}

```

## RMI-CLIENT

```

public class ZeitClient {

    public ZeitClient() {}

    public static void main(String[] args) throws Exception {
        long t1=0, t2=0;
        ZeitInterface remoteZeit =
            (ZeitInterface)Naming.lookup("rmi://compute/MyService");

//2x RPCs / 2x RMIs
        t1 = remoteZeit.getDate().getTime();
        t2 = remoteZeit.getDate().getTime();
        System.out.println("RMI brauchte " + (t2-t1) + " ms");
    }
}

```

## PARAMETER

### BEI METHODEN-AUFRUFEN

#### 1. Wertübergabe (call by value)

Harmlos.

#### 2. Referenzübergabe (call by reference)

Problem = verschieden Adressräume.

Warum geht das in Java?

→ Adressen stehen im selben Adressraum, Dafür Call by Copy/....

#### 3. (Call by Copy/Restore)

In Java.

Ganze Objekte werden übertragen.

Alle Adressen = relativ zum Objektbeginn.



# SOCKETPROGRAMMIERUNG

---

# SOCKETS

= Programmierschnittstelle zu d. Transportprotokollen.

Verbindung  $\hat{=}$  Socketpaar.

## SOCKET ALS DATENSTRUKTUR

= 5 stelliges Array.

Auf beiden Seiten = entweder UDP od. TCP (es gibt keine TCP-UDP-Verbindung).

### Server-Socket

| Protokoll    | localhost | loc. Port | remote Host | remote Port |
|--------------|-----------|-----------|-------------|-------------|
| TCP oder UDP | sun 65    | 80        | sun 67      | 1088        |

### Client-Socket

Ist dann im Client gespiegelt.

|              |        |      |        |    |
|--------------|--------|------|--------|----|
| TCP oder UDP | sun 67 | 1088 | sun 65 | 80 |
|--------------|--------|------|--------|----|

## STREAM SOCKETS

TCP.

Internet = meist TCP-Stream-Sockets.

Realisieren Client-Server-Modell.

= Bidirektional (PIPES die in beiden Richtungen arbeiten).

Röhren (Pipes) zw. Anwendungen.

Der eine schickt Bytes der Andere liest (Geht auch beides gleichzeitig).

1. Server richtet Socket ein (Serversocket).
2. Server horcht an diesem Socket.
3. Client richtet Socket ein.
4. Client verbindet sich mit d. Server.
5. Sobald Client b. Server angekommen, verdoppelt Server seinen Socket (damit Server noch Socket frei hat, um auf weitere Clients zu horchen).

## DATAGRAM SOCKETS

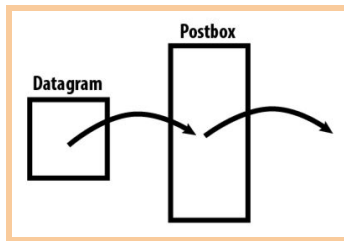
UDP.

Datenpacket, mit vollständiger Adressinformation von Sender + Empfänger.

Kein Client-Server-Modell.

|                      |             |
|----------------------|-------------|
| Ports<br>IP-Adressen | z.B. char[] |
|----------------------|-------------|

Ähnlich dem Postmodell.



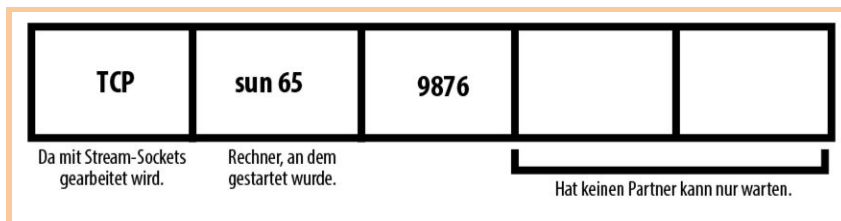
## JAVA-STREAM-SOCKETS

### SERVER

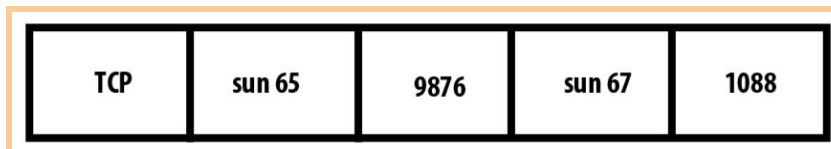
Benötigt 2 Klassen = ServerSocket + Socket (StreamSocket-Klasse).

```
ServerSocket serversocket = new ServerSocket(9876);
```

### Im Array



Im accept() verdoppelt sich Socket, dann übernimmt Server vom Client d. Werte aus Index [1]+[2]



### Server wartet

```
Socket s = serversocket.accept();
```

### CLIENT

Benötigt nur Socket-Klasse.

Benötigt vollständig spezifizierten Socket.

```
Socket clientsocket = new Socket("sun65.beuth-hochshcule.de", 9876);
```

Konstruktor von Socket stellt Verbindung her.

## Im Array

| TCP                                    | sun 67                           | 1088  | sun 65                        | 9876 |
|--|----------------------------------|---|-------------------------------|------|
| Da mit Stream-Sockets gearbeitet wird. | Rechner, an dem gestartet wurde. | Client wendet sich an Portmapper (ordnet Client einen Port zu). | Hat Client über DNS bekommen. |      |

## TCP-ECHOSERVER

### SERVER

```
class TcpEchoServer {
    public static void main(String[] args) throws Exception {
        int serverPort = 9021;
        System.out.println("S: Start auf Host sun65" +
                           " am Port "+serverPort);

        // ServerSocket einrichten und im accept() warten
        // -----
        ServerSocket ss = new ServerSocket(serverPort);
        Socket      s = null;
        System.out.println("S: Vor dem accept()");
        s = ss.accept();

        // Leser und Schreiber einrichten
        // -----
        BufferedReader sbr = new BufferedReader(
                                new InputStreamReader(
                                    s.getInputStream()));
        PrintWriter spw = new PrintWriter(s.getOutputStream());

        // Die wiederholten Echos mit diesem Client abwickeln
        // -----
        String zeile;
        while(true) {
            System.out.println("S: Vor dem readLine()");
            zeile = sbr.readLine();
            System.out.println("S: Aus dem Socket kommt->" + zeile);
            spw.println(zeile);
            spw.flush();
            if(zeile.equals("quit")) break;
        } // while

        spw.close();
        sbr.close();
    }
}
```

```

        ss.close();
        s.close();

    } // main()
} // class

```

## CLIENT

```

class TcpEchoClient {
    public static void main(String[] args) throws Exception {
        System.out.println("C: Start");

        // Socket einrichten und verbinden
        // -----
        System.out.println("C: Socket einrichten und verbinden");
        Socket s = new Socket("sun65.beuth-hochschule.de", 9021);

        // Leser und Schreiber fuer den Socket einrichten
        // -----
        BufferedReader sbr = new BufferedReader(
            new InputStreamReader(
                s.getInputStream() ));
        PrintWriter spw = new PrintWriter(s.getOutputStream());

        // Echostring vom Benutzer anfordern, schreiben und lesen
        // -----
        String echo = null;
        String back = null;
        BufferedReader br = new BufferedReader(
            new InputStreamReader(System.in));
        while(true) {
            System.out.println("\nGib Echostring (Ende mit quit)");
            System.out.print("->");
            echo = br.readLine();
            echo = echo.trim();
            System.out.println("C: Vor dem println() mit:"+echo);
            spw.println(echo);           // Schreiben und
            spw.flush();
            System.out.println("C: Vor dem readLine()");
            back = sbr.readLine();      // Lesen
            System.out.println("Zurueck kommt:" + back);
            if(back.equals("quit")) break;
        } // end while

        spw.close();
        sbr.close();
        br.close();
    }
}

```

```
s.close();  
}  
}
```

## JAVA-DATAGRAM-SOCKETS

Unabhängiger Transport der Datagramme.

Verbindungsloser Transport.

Bilden Peer-to-Peer-Modell.

Kein Client-Server-Modell.

### JAVA-KLASSEN

1. DatagramSocket
2. DatagramPacket

### EMPFÄNGER-AKTIONEN

```
byte [] b = new byte[512]; //groß genug machen, keine Vorgaben  
//lenght, damit auch kleineres als b.lenght verchickt werden kann  
DatagramPacket dp = new DatagramPacket(b, b.lenght);
```

```
//Port = muss der Sender kennen  
DatagramSocket ds = new DatagramSocket(8641);
```

```
//aus dem Socket lesen  
ds.receive(dp);
```

### SENDER-AKTIONEN

```
String s = "Das ist ein String";  
byte[] b = s.getBytes();
```

```
//benoetigt IP-Adresse d. fernen Empaengers  
//Angenommen er will zu: hrz.beuth-hochschule.de  
//dann muss er d. DNS abfragen
```

```
InetAddress ia = InerAddress.getByName("hrz.beuth-hochschule.de");  
DatagramPacket dp = new DatagramPacket(b, b.lenght, ... , 8641);  
DatagramSocket ds = new DatagramSocket();  
ds.send(dp); // → in den Socket
```

## BEIM EMPFÄNGER NACH RECEIVE()

```
dp.getData()
dp.getPort()
dp.getAddress()
dp.getLength() ...
```

## ECHO-SERVER

```
class UdpEchoServer {
    public static void main(String[] args) throws Throwable {
        System.out.println("S: Start");
        // Einen Socket einrichten
        // Am angegebenen Port wird auf Requests gewartet
        // -----
        DatagramSocket ds = new DatagramSocket(8888);
        // Ein leeres Datagramm bereit stellen
        // Ist es zu klein, wird abgeschnitten
        // -----
        byte[] b = new byte[5];
        DatagramPacket dp = new DatagramPacket(b, b.length);
        // Datagramme empfangen und zuruecksenden
        // Bei leerem Socket blockiert das receive()
        // -----
        String inhalt = null;
        byte[] bi = null;
        while(true) {
            ds.receive(dp);
            ds.send(dp);
            bi = dp.getData();
            inhalt = new String(bi);
            System.out.println("S: Trace: " + inhalt);
            if(inhalt.startsWith("quit")) break;
        }
        ds.close();
        System.out.println("S: Ende");
    }
}
```

## ECHO-CLIENT

```
class UdpEchoClient {
    public static void main(String[] args) throws Throwable {
        System.out.println("C: Start");
        // Socket einrichten, Adressinformation steht im Datagramm
        // -----
        DatagramSocket ds = new DatagramSocket();
        // Vom Benutzer einen zu uebertragenden String anfordern
        // -----
    }
}
```

```

String echo = null;
byte[] b = null;
String back = null;
byte[] c = null;
BufferedReader br = new BufferedReader(
    new InputStreamReader(System.in));
InetAddress sa = InetAddress.getByName(
    "hrz.beuth-hochschule.de");

DatagramPacket raus = null;
DatagramPacket rein = null;
while(true) {
    System.out.println("\nGib Echostring (Ende mit quit)");
    System.out.print("->");
    echo = br.readLine();
    echo = echo.trim();
    // Datagramm bilden und senden
    // -----
    b = echo.getBytes();
    raus = new DatagramPacket(b, b.length, sa, 8888);
    System.out.println("C: Vor dem send()");
    ds.send(raus);
    // Echo empfangen, leeres Datagramm bereitstellen
    // -----
    c = new byte[b.length];
    rein = new DatagramPacket(c, c.length);
    ds.receive(rein);
    back = new String(rein.getData());
    System.out.println("C: Echo: " + back);
    if(back.equals("quit")) break;
} // end while
ds.close();
System.out.println("C: Ende");
}
}

```

## BROWSER

Jd. Browser richtet Client-Socket ein.

### BEISPIEL

Browser-URL = <http://www.beuth-hochschule.de:9876>

### Was schreibt Client in Socket?

Macht daraus einen Get-Request, schreibt GET `_/_` HTTP/1.1 → Server.



■ = Server Rout, Blanks ( ) entstehen, weil nach URL und Port v. Client nichts mehr geschrieben wurde.

### Angenommen

Server hat als Ergebnis eine HTML-Seite erzeugt.

### Server schreibt in Socket

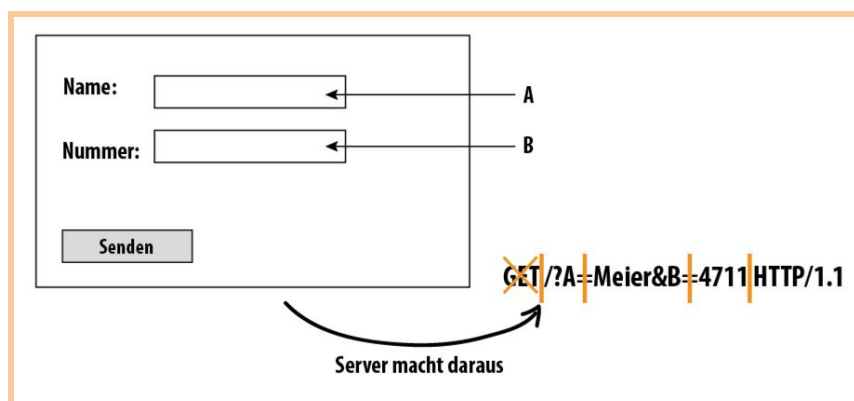
HTTP/1.1 200 OK (Hier werden auch Fehler wie 401 übergeben)

Connection: close

Content-Typ: text/html

leer → (Return, hier endet Header. Bis hier hin = HTTP)

HTML Seite



### HTTP-SERVER

```
ServerSocket ss = new ServerSocket(9876);
Socket cs = ss.accept();
BufferedReader br = new BufferedReader(new InputStreamReader(
    cs.getInputStream() ));
String zeile = br.readLine();
Zeile verarbeiten;
PrintWriter pw = new PrintWriter(cs.getOutputStream() );
pw.println("HTTP/1.1 200 OK");
pw.println("Connection:close");
pw.println("Content-Type:text/html");
pw.println("\n"); //Header wird beendet
pw.println(""); //ab hier HTML-Seite
...
pw.println("");
pw.flush();
cs.close();
ss.close();
```

Bei erster Anfrage wird nach favorisierten Icon gefragt, kann man ignorieren mit:

```
if(zeile.startsWith("GET /favicon")) {
    System.out.println("Favicon-Request");
}
```

```

        br.close();
        continue;                                // Zum naechsten Request
    }

```

## REQUESTS

### GET

Überträgt Variablen in Adresszeile (URL), limitierte Anzahl / Länge an Variablen.



An test.php wurde die Variable vorname mit Wert 'Foo' + die Variable nachname mit dem Wert 'Bar' übergeben.

Diese Werte können mit PHP mit vordefinierten Array \$\_GET angesprochen werden.

```

<?php
echo $_GET['vorname']; // Ausgabe: Foo
echo $_GET['nachname']; // Ausgabe: Bar
?>

```

Nach Dateinamen = Fragezeichen (?).

Dann erster Variablenname, Gleichheitszeichen (=) und Wert.

Weitere Variablen werden mit &-Zeichen, gefolgt von Variablenname, Gleichheitszeichen und Wert angehängt.

### Vorteile

Einfache Übergabe der Variablen in Links.

Client kann Webseite inklusive Variablen Favoriten hinzufügen.

### Nachteile

Client sieht Parameter in Adresszeile und kann diese ändern.

Durch Adresslänge begrenzte Anzahl an Variablen bzw. Länge der Werte.

Übertragung von Dateien nicht möglich.

### Verwendung

#### Menülink mit Übergabe einer/weniger Variablen

Üblicherweise = <a>-Tag.

Angehängte Variablen werden automatisch mitgeschickt.

### POST

Überträgt Variablen für Client unsichtbar im Hintergrund, unlimitierte Anzahl / Länge an Variablen.

Häufiger Anwendungsfall = Formulare.

"name" Attribut = Variablenname, mit welchem später auf Variable zugegriffen wird.

```
<form action="senden.php" method="POST">
  <input type="text" name="vorname" />
  <input type="text" name="nachname" />
  <input type="submit" value="Senden" />
</form>
```

2 Variablen werden per POST an Datei senden.php übergeben: vorname und nachname.  
Zugriff auf Inhalt des Formulars über vordefiniertes Array \$\_POST.

```
<?php
    echo $_POST['vorname'];
    echo $_POST['nachname'];
?>
```

### Vorteile

Fast unbegrenzte Anzahl von Variablen + Wertlängen.  
Übertragung von Dateien in Formular möglich.  
Client sieht die Parameter nicht.

### Nachteile

Seite mit übergebenen POST Werten kann nicht zu Favoriten hinzugefügt werden.  
Verwendungsmöglichkeit beschränken sich auf Formulare.

### Verwendung

Sobald Daten zum Server übertragen werden sollen, da mehr Daten als mit GET-Methode transportiert werden können, und normaler Benutzer keinen Zugriff auf die Daten hat.

### Kontaktformular

Anfragetext kann etwas länger werden, deshalb besser POST verwenden.

### Dateiuploadformular

Datei ist oft etwas größer, deshalb POST.

# ARCHITEKTUREN

---

# INTERNET

## FILE TRANSFER

Ältester Dienst.

Am Anfang nur d. Dienst, für diesen ist Internet entwickelt worden.

## REMOTE LOGIN

Zweit ältester Dienst.

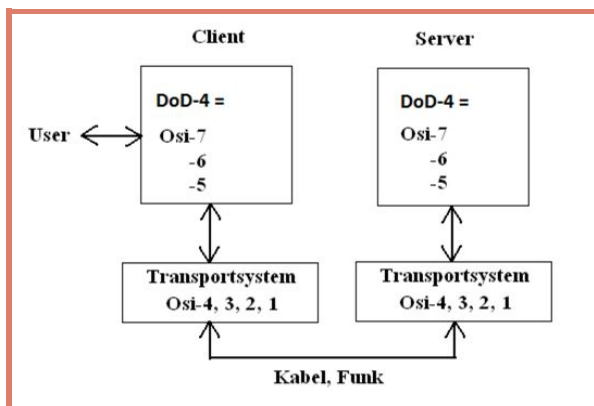
Einmaliges Einloggen + mit fernen Rechner arbeiten als säße man dort (Nutzung v. nur einem Bildschirm).

## E-MAIL

File Transfer mit Briefkasten-Funktion.

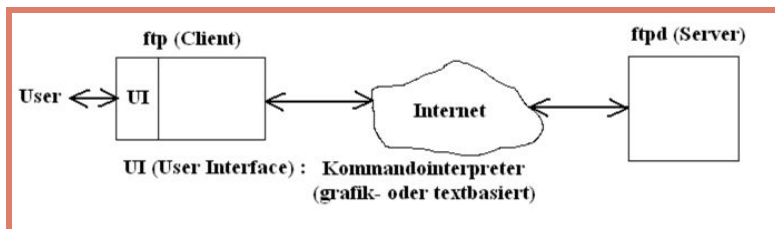
## Basis Struktur der Internet-Dienste

Client - Server-System (E-Mail weicht aber etwas ab).



# FTP-ARCHITEKTUR

Client liest User-Eingabe (+ prüft diese auf Richtigkeit) + schickt diese weiter zum Server.



## NACHTEILE ALTER DIENSTE

Passwort wird mit übertragen.

## Beispiel

Einloggen → FTP-Verbindung → Aufforderung Passwort-Eingabe → Ersten Dienste übertragen dieses ohne Verschlüsselung → Deshalb SFTP (S = Secure).

## SFTP-VERBINDUNG

Funktioniert nur, wenn man auf fernem Rechner auch ein Login besitzt.

### 1. SFTP-Verbindung

```
sftp sun70
```

#### Unterschiedliche Nutzernamen

```
sftp <userName>@<hostname>
```

### 2. Ferner Rechner

```
please login  
user:          //Eingabe  
passwd:        //Eingabe
```

### 3. Eingabe von SFTP-Kommandos

```
sftp > //Eingabe  
z.B  
sftp > quit
```

## SFTP-KOMMANDOS

= Nicht genormt.

### quit

Client beenden.

### pwd

= print working directory.

Ferner Pfad.

### lpwd

Lokaler Pfad.

### cd

Fernes Verzeichnis wechseln.

### lcd

Lokales Verzeichnis wechseln.

### ls

Fernes Verzeichnis auflisten.

### lls

Lokales Verzeichnis auflisten.

**get <Datei>**

Datei vom Server holen.

**put <Datei>**

Datei zum Server schicken.

## TELNET-ARCHITEKTUR

Unverschlüsselt, Nachfolger = SSH (Secure Shell).

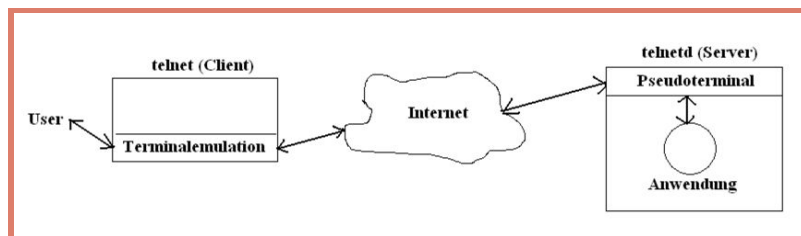
Es gibt keine Telnet od. SSH-Kommandos, nur Kommandos d. fernen Rechners.

Auf Clientseite muss Terminal emuliert werden.

Schwieriger bei unterschiedlichen Betriebssystemen (z.B. Windows + Linux = auf Windows muss Linux-Terminal emuliert werden).

Auf Server-Seite = Anw. zur Ausführung (Anw. muss fern gestartet werden, z.B. Java = ProzessBuilder).

Input, Output, Error muss vom Server umgeleitet werden.



### BEISPIEL

```
->$hostname  
->sun65  
->$ssh sun70  
->user: Meier  
->passwd: xxxxx  
->$hostname  
->sun70  
->$
```

Ab hier könnten dann z.B. java-Dateien ausgeführt werden, die vorher per SFTP übertragen wurden.

### Unterschiedliche Nutzernamen

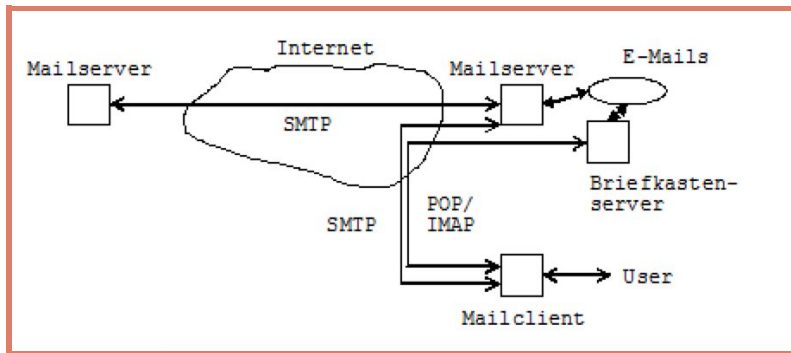
```
ssh <userName>@<hostname>
```

## E-MAIL-ARCHITEKTUR

Client = unterhält sich entweder mit Mail od. Briefkasten-Server.

Manipulation d. Mails (z.B. Ordnen d. Mails) = nicht d. Mailserver, sondern Briefkasten-Server.

Server untereinander bilden Peer-to-Peer.



## SMTP-BEISPIEL

Ein Benutzer meier am Host sun32 sendet Mail an schmitt@ibm.de.  
E-Mail-Client führt folgenden Dialog.

```
HELO sun32.beuth-hochschule.de
250 mailbox2.beuth-hochschule.de
MAIL FROM: <meier@beuth-hochschule.de>
250 ok
RCPT TO: <schmitt@ibm.de>
250 ok
DATA Dies ist ein Test.
.                                     //Mail geht bis zu diesem Punkt

250 ok: queued as ...
```

## E-MAIL-ADRESSEN

### Aufbau

<user>@<domain>

Hostname fehlt → durch DNS wissen Router trotzdem, wo sie hinschicken müssen.

### URI

= Uniform Ressource Identifier

## MODERNE INTERNET-TOOLS

### DNS

= Domain Name System.

Vermittelt zw. d. beiden Adressarten (bildet d. eine Adresse auf die andere ab).

### Adressen im Internet

#### v4

4 Bytes.

Zum Beispiel = 141.64.66.7



v6

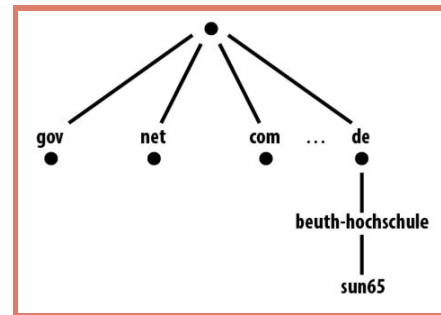
16 Bytes.

### Domain-Adressen

Kam später dazu (Nummern = umständlich).

Gliederung.

Schreibweise von rechts nach links (Top-Level-Domain, Domain, Subdomains, Host).



### LOKALER NAME-SERVER

Linux: /etc/resolv.conf

### Beispiel

```
domain      beuth-hochschule.de
nameserver  141.64.3.55
nameserver  141.64.3.40    //Ersatz-Nameserver, falls erster ausfällt
```

### Benutzer-Eingabe d. sun 67

```
$ sftp sun77.beuth-hochschule.de
```

1. sftp-Programm schaut nach Nameserver in Domain-Datei.
2. An diesen nameserver schickt er den string nach sftp.
3. Gibt dem Tool IP-Adresse der sun 77 um sich an diesen zu wenden.

Adresse www.abm.com nicht gefunden = wird an com-Knoten weiter gereicht.  
Wird dann gefunden oder an unteren Knoten weiter gereicht.

### Funktionen

```
getByName()  
getByAddress()
```

### Möglichkeiten

1. Ans DNS wird Domain-Name gesendet + IP-Adresse wird zurück gesendet.
2. IP-Adresse wird geschickt + Domain-Name wird zurück gesendet.

### Kommando DNS

```
c:\> nslookup www.beuth-hochschule.de  
Standardserver: dnsmaster.beuth-hochschule.de  
Address: 141.64.3.55
```

```
/* hier ist zu erkennen, das www.beuth-hochschule.de nur ein Aliasname ist */  
Name: rs1050.beuth-hochschule.de  
Address: 141.64.226.50  
Aliases: www.beuth-hochschule.de  
c:\>
```

## Anfrage Name Mailserver

```
c:\> nslookup -type=mx tu-berlin.de
Standardserver: dnsmaster.beuth-hochschule.de
Address: 141.64.3.55
```

```
tu-berlin.de MX preference = 100, mail exchanger = mail.tu-berlin.de
mail.tu-berlin.de internet address = 130.149.7.33
c:\>
```

## Mit Java

```
class Dns {

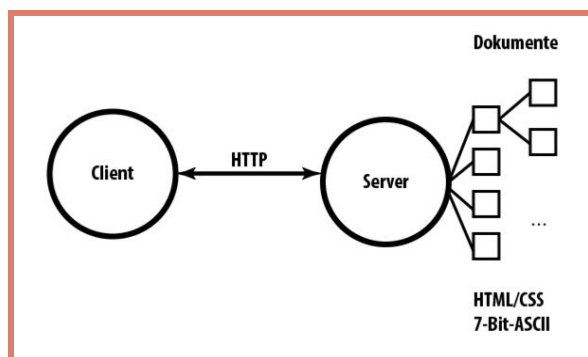
    public static void main(String[] args) throws Throwable {
        System.out.println("Zugriff auf das DNS");

        String ziel = "www.tu-berlin.de";
        InetAddress addr = InetAddress.getByName(ziel);
        String ipAdresse = addr.getHostAddress();
        System.out.println(ziel+"->" + ipAdresse);

        ziel = "141.64.226.50";
        byte b[] = new byte[] { (byte)141,
                                (byte)64,
                                (byte)226,
                                (byte)50
                                };
        addr = InetAddress.getByAddress(b);
        String host = (String)addr.getHostName();
        System.out.println(ziel+"->" + host);
    }
}
```

## WEB

Steht unter der Obhut v. <http://www.w3.org>.  
Client-Server-System.



## QUERYSTRING

Wird v. Browser aus Forminhalt erzeugt.

### Beispiel

Input A = Meier

Input B = 4711

### QueryString

A=Meier&B=4711 → dann get od. post → Server

## UMLAUTE UND WORTLÜCKEN

Browser macht aus Umlauten + Wortlücken (immer) URL-Kodierung.

Muss rückgängig gemacht werden.

### URL-Kodierung

Blank = +

Alle anderen Sonderzn. = %xx (=hexacode d. Zn.)

### Beispiel

ä = %e4

+ = %2b

...

Peter Pän = A=Peter+P%e4n&...

## METHOD ARTEN

= Protokoll, das Kommunikation zw. Server und Webbrowser regelt.

## REQUESTS

**GET, POST**, HEAD, PUT, OPTIONS, DELETE, TRACE, CONNECT.

## KOMMUNIKATIONSZYKLUS

1. Server wartet auf Client.
2. Client → Request → wartet auf Antwort.
3. Dann wieder von vorne.

## ANFORDERUNGS-PAKETE

### get-Request

Datenpaket besteht nur aus Header.

### Browser schreibt

GET/Anforderung Protokollversion

Meta-Daten

Leerzeile

Zwei aufeinanderfolgende newlines beenden Header (/n nach Meta-Daten + /n nach Leerzl.).

### Anwendung

Bei wenig Daten.

### post-Request

Datenpaket besteht aus Header + Datenteil.

### Browser schreibt

```
POST / Protokollversion
Meta-Daten
Leerzeile //Bis hier hin = Header
Anforderung //Daten
```

### Anwendung

Bei vielen Daten.

### URL

#### Aufbau

[Protokoll://] (HTTP, wird v. meisten Browsern ersetzt) [Rechner] Domain[.Domain]\*[:Port][Pfad zu einer Datei]

Fehlen Teile, werden Default-Werte eingesetzt.

Auf Server-Host gibt es Verzeichnis namens ServerHost.

Fehlt Pfad, wird in ServerHost nach welcome.html, welcome.htm, ... gesucht.

### Beispiel

Benutzer-Eingabe im Browser = http://www.beuth-hochschule.de

1. Browser baut Socket-Verbindung zum Host in Domäne auf  
www.beuth-...,Port=80 (default)

2. Schreibt  
/GET/HTTP/1.1  
Metainfos

http://www.beuth-hochschule.de/**abc/a.txt**

→ GET /**abc/a.txt** HTTP/1.1

### BROWSEEREINGABEN

1. Benutzung der URL-Zeile

Daraus entsteht immer get-Request. (immer, nicht änderbar).  
Form wird nur 'vorgegaukelt'.

## 2. Link auf einer HTML-Seite

Daraus entsteht immer get-Request.

## 3. Form in einer HTML-Seite

GET od. POST je nach method-Angabe.

## HTTP-GET-REQUEST

Entstehung einer erweiterten URL.

`http:// ...?Query-String`

→ `GET /?Query-String HTTP/1.1`

## Beispiel

Suchanfrage = `http://public.beuth-hochschule.de/~brecht`

■ = Host      ■ = Home

## Request - Browser an Server

`GET /~brecht HTTP/1.1`

`Host: public.beuth-hochschule.de:80`

`User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:16.0) Gecko/20100101  
Firefox/16.0`

`Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8`

`Accept-Language: de-de,de;q=0.8,en-us;q=0.5,en;q=0.3`

`Accept-Encoding: gzip, deflate`

`Connection: keep-alive`

## Server-Antwort

`HTTP/1.1 200 OK`

`Date: Thu, 22 Nov 2012 15:10:02 GMT`

`Server: Apache/2.2.9 (Debian) mod_ssl/2.2.9 OpenSSL/0.9.8g`

`Set-Cookie: fe_typo_user=c85242ccb09dbb6ae91e206ae62a0e6d; path=/`

`Cache-Control: max-age=82800`

`Expires: Fri, 23 Nov 2012 14:10:02 GMT`

`Content-Length: 7546`

`Connection: keep-alive`

`Content-Type: text/html; charset=utf-8`

**/\* HTTP-Seite, die er zurück schickt \*/**

`<?xml version="1.0" encoding="utf-8"?>`

`<!DOCTYPE html`

`PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"`

`"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`

`<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de" lang="de">`

`<head>`

...

### Mit Java aus Url lesen

```
class UrlRead {
    public static void main(String[] args) throws Exception {
        // Ziel festlegen (lokal: file:///e:/abc.htm)
        URL url = new URL("http://www.beuth-hochschule.de/");

        // Einige Infos
        System.out.println("Proto ->" +url.getProtocol());
        System.out.println("Host  ->" +url.getHost());
        System.out.println("File  ->" +url.getFile());
        System.out.println("\n");

        // Lesen und ausgeben
        BufferedReader in = new BufferedReader(
            new InputStreamReader(url.openStream() ));

        String zeile = null;
        while( (zeile = in.readLine()) != null) System.out.println(zeile);
    }
}
```

### Post-Request

```
POST / HTTP/1.1
Host: sun65.beuth-hochschule.de:9876
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:16.0) Gecko/20100101
Firefox/16.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de-de,de;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
```

**/\* Bytes v. A bis Abschicken, s.u. \*/**

Content-Length: 23

**/\* Formular-Eingaben \*/**

A=Meier&B=&C=Abschicken

**Antwort = wie beim get-Request.**

# VERTEILTE DATEISYSTEME

Benutzer merkt nicht, dass sein File-System (Teil davon) nicht lokal ist (=transparent).

## Nicht Transparent

`$sftp sun 65` (eigene Angabe v. fernen Host).

## Transparent

`$cut a.txt` (lokal? od. remot?)

## DATEISYSTEME

Rechner, die nur physikalische Dateisysteme kennen = z.B. Windows (Laufwerk = phy. Dateisystem).

Rechner, d. beides kennen = z.B. Linux/Unix.

### 1. Physikalische Dateisysteme

Physikalisch = Datenträger mit eigener Verzeichnisstruktur (od. Teil davon = Partition).

Baumartige Struktur.

z.B. USB-Stick.

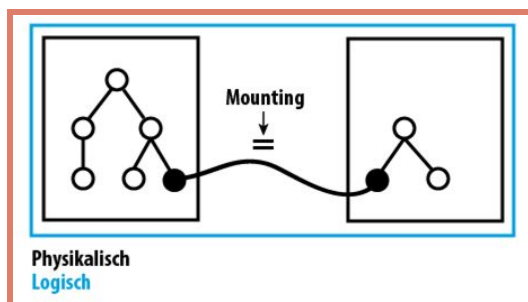
### 2. Logische Dateisysteme

Logisch hier = nicht Physikalisch.

#### Entsteht durch

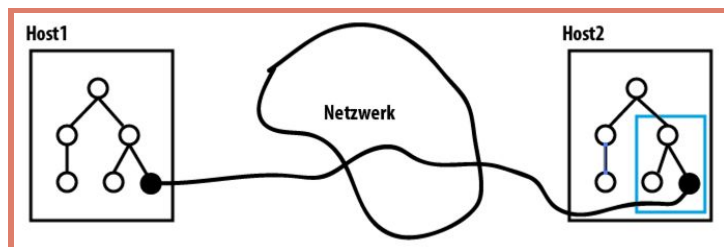
1. Physikalisches Dateisystem wird als logisch erklärt.

Gaukelt großes Dateisystem vor, dass in Wirklichkeit aus vielen Einzelnen besteht.

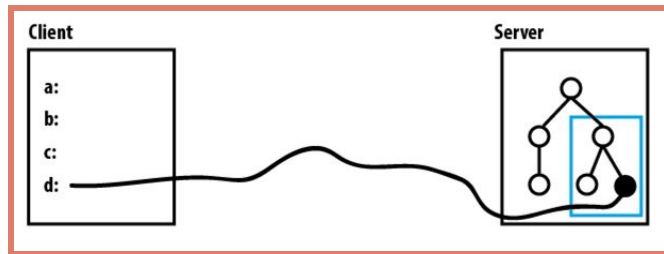


2. Weitere Dateisysteme durch Mounting einbinden.

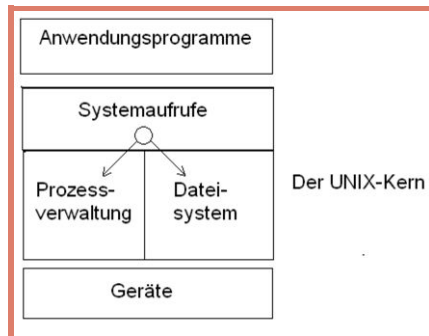
## VERALLGEMEINERUNG DES MOUNTING



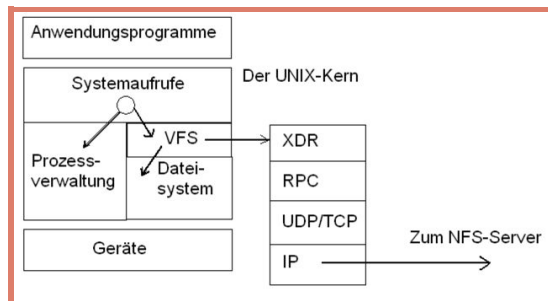
## Bei Windows



## SCHICHTENMODELL BETRIEBSSYSTEME



## VIRTUELLES FILESYSTEM



## XDR

External Data Representation.

OSI-6.

Protokoll, das d. Datenkodierung festschreibt.

## RPC

Remote Procedure Calls.

Ursprünglich UDP-Anwendung, heute meist TCP-Anwendungen.

## Standard-System

= NFS (Network File System).

Entwickelt von Sun → Oracle.

Wenn Nutzer auf beiden Rechner d. selbe User-ID hat, ist Mounting sofort zulässig.



# REPLIKATIONSSYSTEME

= spezielle verteilte Systeme.

## Beispiel

DNS (weltweit werden Kopien geführt).

Anlage einer Kopie entspricht auch Replikation.

Damit verbunden, d. Frage = sind d. Daten konsistent (vollständig + auf allen Kopien identisch)?

## MASTER / SLAVE

Wird in d. Regel verwendet.

Master = betreibt Original

Slave = betreibt Kopie.

## ASYNCHRONE REPLIKATION

Auf Kopien wird unabhängig voneinander gearbeitet.

### Nach einem $\Delta t$ :

Kopien werden gesperrt.

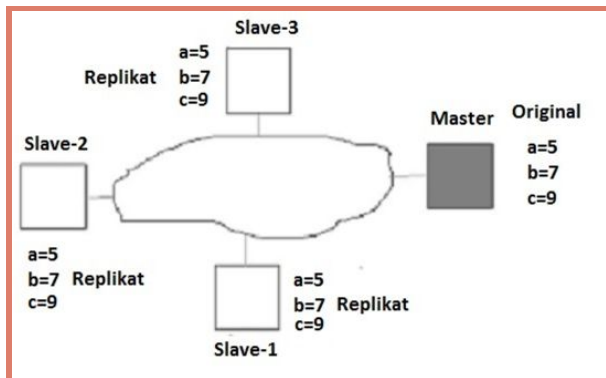
Kopien an Master.

Master führt zusammen.

Master schickt zurück.

## SYNCHRONE REPLIKATION

Wird an Slave Datum geändert = Änderungsoperation wird erst dann beendet, wenn Original + alle Kopien geändert sind.



Ganzes System = konsistent (Daten = überall die selben).

## 2-PHASEN-SPERRPROTOKOLL

Würde Slave1 naiv aus B 8 machen = inkonsistent.

### Stattdessen

1. Slave1 sperrt Datum B.
2. Schickt an d. Master = B auf 8 ändern.
3. Wird geändert.

4. Master sperrt  $b=7$ .  
(Wenn schon gesperrt = Master gibt Slave1 bescheid + Slave1 spielt wieder  $B = 7$  ein).
5. Master verständigt alle, dass sie B sperren sollen.
6. Alle sollen B auf 8 stellen.
7. Master gibt bescheid, dass sie wieder freigeben können.

# NETZWERKE

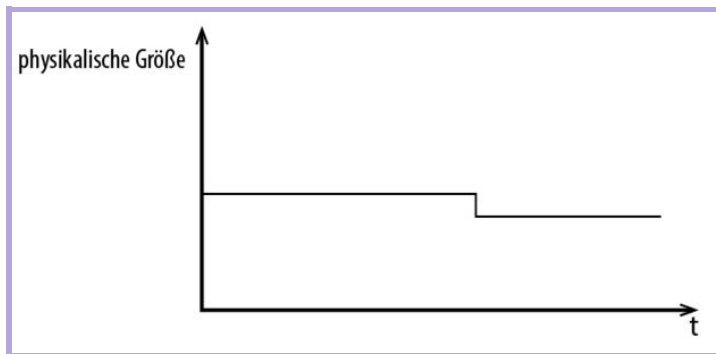
---

# SIGNAL

= Änderung einer physikalischen Größe, über die Zeit gemessen.

## BEISPIELE

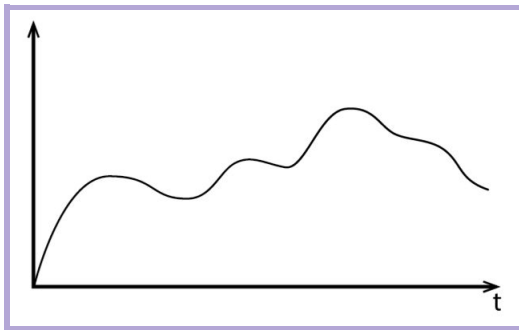
c, Temperatur, Gewicht, Masse, Fläche, Länge, Spannung, Stromspannung etc.



## SIGNALARTEN

### 1. y-Achse ist $\subseteq \mathbb{R}$

Signale heißen analog.

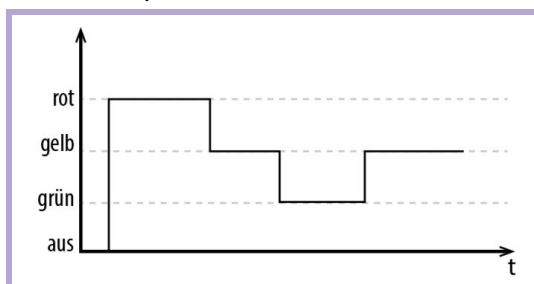


### 2. y-Achse kann nur diskrete Werte annehmen

Signale heißen digital.

#### Beispiel

Verkehrsampel



#### Spezialfall

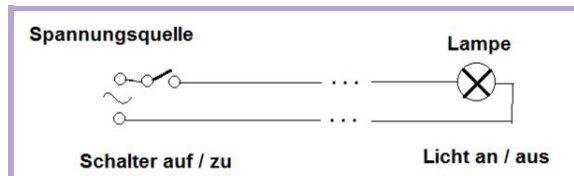
Es gibt genau 2 Werte auf der y-Achse (Signal heißt binär).

# SIGNALÜBERTRAGUNG

Wird phy. Größe an Ort A verändert und diese Veränderung an anderem Ort gemessen (direkt od. indirekt) = Signalübertragung von A nach B.

## BEISPIEL

Lampe misst Spannung, je nach Spannung geht dann Lampe an od. nicht.



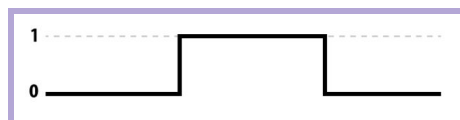
# BITCODIERUNG (ALS SIGNALE)

## 1Bit

= Objekt mit genau 2 Zuständen.

## DARSTELLUNG ALS BINÄRES SIGNAL

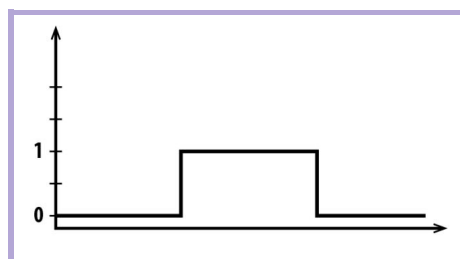
y-Achse wird nicht mehr gezeichnet.



## Allgemeiner

Digitales Signal mit mehr als 2 Werten.

2 Werte auszeichnen und als 0 bzw. 1 markieren.



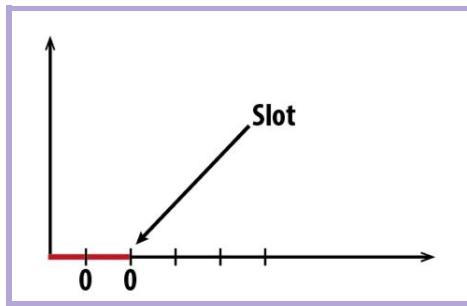
## TRENNUNGSPROBLEM

Übertragung: 00 \_\_\_\_\_  
→ Wie viele Nullen?

## Lösung 1

Die Zeit rastern.

Alle Bits sind gleich lang.

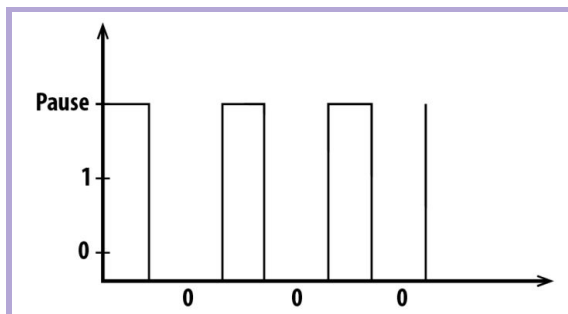


### Problem

Sender + Empfänger müssen gleiche Uhr haben.  
Sonst = Ständige Synchronisation erforderlich.

### Lösung 2

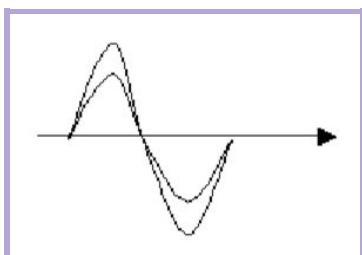
Benutze 3-wertige Signale!



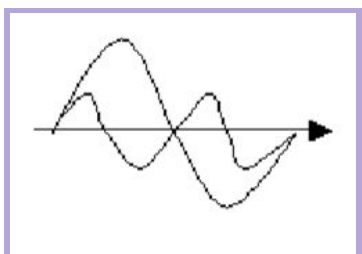
## BITCODIERUNG ANALOGE SIGNALE

Benutze periodische analoge Signale.

### AMPLITUDENMODULATION

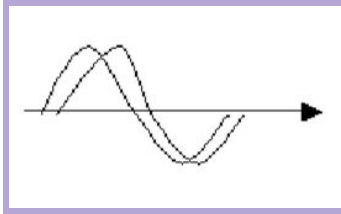


### FREQUENZMODULATION



## PHASENMODULATION

Elektrisch am stabilsten.



## PHYSISCHE VERBINDUNGSMÖGLICHKEITEN

### Kabelverbindungen

Kupferkabel

Adernpaare

Koaxialkabel

Glasfaserkabel

### Funkverbindungen

Terristischer Funk

Zellularfunk

Richtfunk

Satellitenfunk

### Optische Verbindungen

Infrarot-Verbindung

Laserverbindung

## ÜBERTRAGUNGSVERFAHREN

### 1. SERIEL

1 Bit nach dem anderen.

### 2. PARALELL

### SIMPLEX

Übertragung in genau eine Richtung:  $A \rightarrow B$

### HALBDUPLEX

In eine Richtung, aber Wechsel möglich  $A \rightleftharpoons B$

### (VOLL)DUPLEX

In beide Richtungen gleichzeitig  $A \leftrightarrow B$

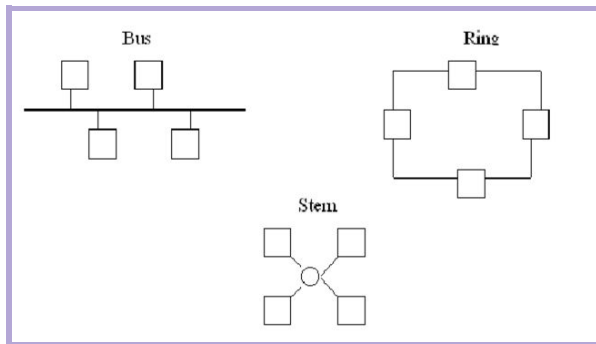
# PHYSIKALISCHE NETZWERKE

Verbinden Rechner physikalisch.

Übertragen Bits.

## NETZWERKTOPOLOGIE

= Struktur der Netze.



## BUS

Alle Rechner im Netz verwenden das selbe phys. Medium (Kabel, Funk).

Bus-Netz = stabil gegen Rechnerausfälle.

Nachricht erreicht jd. Rechner, diese müssen dann jeweils entscheiden, ob sie Nachricht weiterleiten od. nicht.

## RING-NETZE

Unidirektional, bidirektional (können gerichtet od. ungerichtet sein).

Nachricht erreicht jd. Rechner, diese müssen dann jeweils entscheiden, ob sie Nachricht weiterleiten od. nicht.

Verbindung mit Token Passing.

Problem = Token Management.

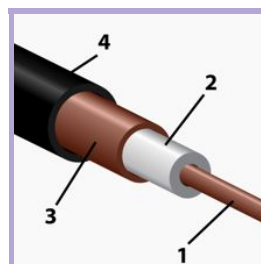
## STERN-NETZE

Ähnlich wie Bus, aber Knoten in der Mitte = aktives Element (enthält Prozessor).

## MEDIEN

### KOAXIALKABEL

1. Kupferkabel
2. Isolator
3. Abschirmung.
4. Schutzhülle.



### TWISTED PAIR

Verdrilltes Kupferkabel.

### FUNKSTRECKEN

Vermindern Kabelverbindungen.



### Grobe Klassifikation nach räumlicher Ausdehnung

1. **LAN (Local Area Network)** (Raum, Gebäude)  
WLAN. (Raum, Gebäude)
2. **MAN (Metropolitan Area Network)**  
(Berliner HS-Glasfaserung).
3. **WAN (Wide Area Network)**  
Internet.

### ETHERNET (ÄTHERNETZ)

Entwickelt 1972 bei Xerox.

Dann standatisiert von Xerox, DEC und Intel.

Ursprünglich ohne LLC.

### NORMIERUNGEN

ISO 8802/3

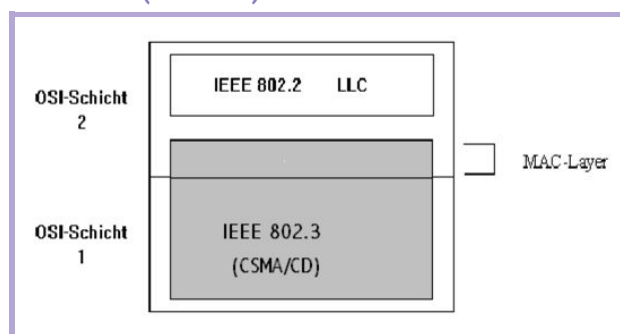
IEEE 802.3

ECMA 80/81/82

### OSI

|          |                 |
|----------|-----------------|
| <b>2</b> | <b>MAC</b>      |
| <b>1</b> | <b>Ethernet</b> |

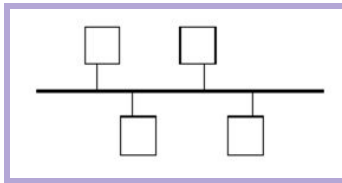
### IEEE 802.3 (Ethernet)



Ethernet ist das am häufigsten eingesetzte Netzwerk.

Ethernet überträgt Pakete.

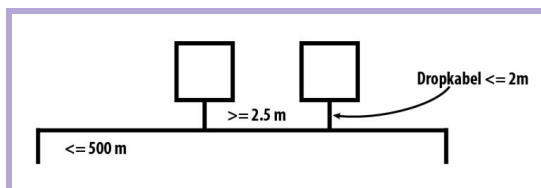
## TOPOLOGIE



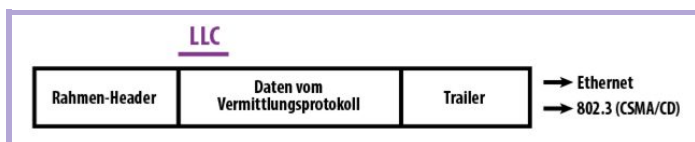
→ Kabel, Funk

### Angenommen: Kabel

Unendlich langes Kabel wird vorgetäuscht, Bits verhungern am Ende.



Ethernet bekommt vom Vermittlungsprotokoll Paket, packt es in einem phy. Rahmen und sendet diesen.



## ETHERNET UND 802.3-RAHMEN

| Ethernet                                |          |             |              |                                   |           |          |            |          |
|---|----------|-------------|--------------|-----------------------------------|-----------|----------|------------|----------|
|   | Präambel | Zieladresse | Quelladresse | Typ                               | Nutzlast  | Prüfzahl |            |          |
| Bytes                                   | 8        | 6           | 6            | 2                                 | 46 - 1500 | 4        |            |          |
| Mindestlänge: 72 Bytes (mit Präambel)   |          |             |              |                                   |           |          |            |          |
| Höchstlänge: 1.526 Bytes (mit Präambel) |          |             |              |                                   |           |          |            |          |
| IEEE 802.3                              |          |             |              |                                   |           |          |            |          |
|   | Präambel | Zieladresse | Quelladresse | Länge                             | LLC       | SNAP     | Nutzlast   | Prüfzahl |
| Bytes                                   | 8        | 6           | 6            | 2                                 | 3         | 5        | 38 - 1.492 | 4        |
| Mindestlänge: 72 Bytes (mit Präambel)   |          |             |              | SNAP: Sub Network Access Protocol |           |          |            |          |
| Höchstlänge: 1.526 Bytes (mit Präambel) |          |             |              | 3 Bytes für den Hersteller        |           |          |            |          |
|   |          |             |              | 2 Bytes als Typ (vom Ethernet)    |           |          |            |          |

### PRÄAMBEL

8 Bytes 010101...0111

Der Empfänger synchronisiert sich damit.

## ADRESSEN

6 Bytes "-" Trenner, hexa

08-00-14-F1-05-92

■ Hersteller

Inzwischen programmierbar.

## BEZEICHNUNG

CSMA/CD (für 802.3).

Carrier Sense, Multiple Access with Collision Detection.

Trägerprüfung, Mehrzugriff mit Kollisionserkennung.

Ursprung: ALOHA an der Uni Hawai (viele Inseln)

→ Funknetz

Kollisionsproblem

(→ Senden = Sender prüft, ob Kollision eintritt.

Ja = Abbruch. Warten (zufällige Zeit, da sonst wieder gleichzeitig angefangen wird), dann Wiederholung – max. 16-mal, dann Empfänger unerreichbar).

## KOLLISION

macht nur bei der Überlagerung Probleme, danach laufen Daten normal weiter.

## Angenommen

Sender möchte wissen, ob sein Paket zur Kollision geführt hat (nicht so einfach).

→ Sender muss vom Senden abgehalten werden.

## ZUSAMMENHANG MIN PAKETGRÖÖE UND MAX KABELLÄNGE

Ur-Ethernet:  $10 \text{ Mb/s}$  ( $10 * 10^6 \text{ b}$ )

$10 * 10^6 \text{ b in } 1 \text{ s}$

$512 \text{ b in } x \text{ sek}$

---

$$x = 1 * 512 / 10 * 10^6 = 51.2 \mu\text{s}$$

## Kabellänge

=  $25.6 \mu\text{s}$

## Ausbreitung auf Kabel

=  $0.77c$

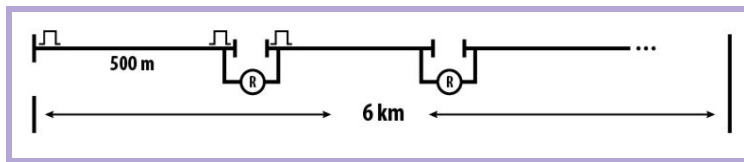
In  $1 \text{ s}$  werden  $0.77 * 0.3 * 10^6 \text{ km}$

In  $25.6 * 10^{-6} \text{ s}$  " "  $x$

$$x = 0.77 * 0.3 * 10^6 * 25.6 * 10^{-6} / 1 \approx 6 \text{ km}$$

## VORSCHRIFT ETHERNET-NORM

min. Paketgröße (ohne Präambel): 64 Bytes entspricht 512 b



## VERBINDUNGSGERÄTE

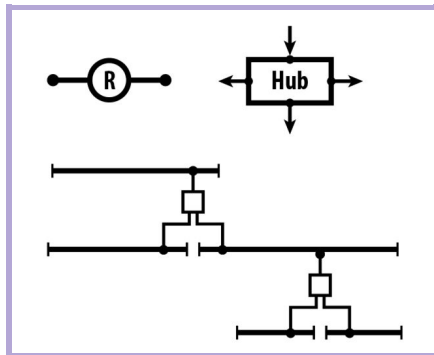
Repeater (Signalverstärker) arbeiten auf OSI-1.

Bridges arbeiten auf OSI-2 (sehen d. Rahmen).

Router arbeiten auf OSI-3 (sehen d. Vermittlungspakete).

## HUB

=Multiport-Repeater.



Damit sind Kaskaden möglich.

## SWITCH

= Multiport-Bridge.

## ARP (ADDRESS RESOLUTION PROTOCOL)

Broadcast (IP-Adresse) → MAC-Adresse zur IP-Adresse

## RARP (REVERSE ARP)

Schickt einen Broadcast los (eigene MAC-Adresse) → Die zur MAC-Adresse gehörende IP-Adresse.

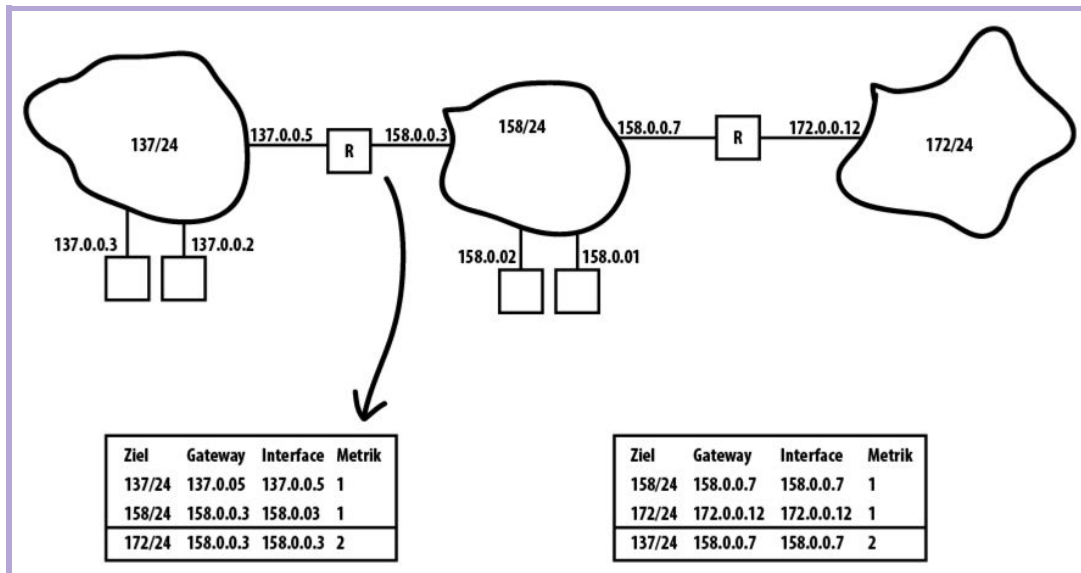
→ = DHCP (Dynamic Host Configuration Protocol).

## ROUTING-PROTOKOLLE

In Router eines Netzwerkverbundes, tauschen ihre Routingtabellen aus.

Üblich = Nachbarschaftsaustausch.

(RIP : Router Information Protocol).



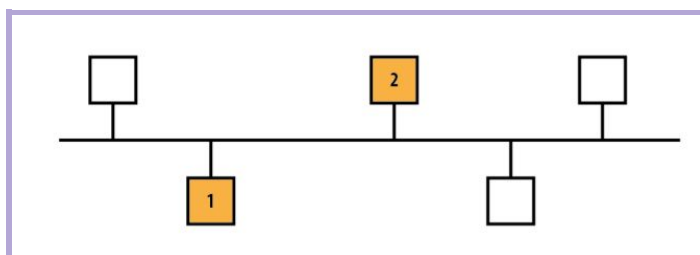
Routingtabellen können sehr groß werden.

### Routingtabelle von 137.0.0.2

| Ziel   | Gateway   | Interface | Metrik |
|--------|-----------|-----------|--------|
| 137/24 | 137.0.0.2 | 137.0.0.2 | 1      |
| 158/24 | 137.0.0.5 | 137.0.0.2 | 2      |
| 172/24 | 137.0.0.5 | 137.0.0.2 | 3      |

## FIREWALL

= Steuerbarer Paketfilter.

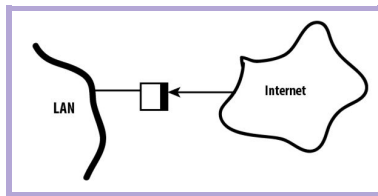


Host 1 möchte Host 2 "Hallo" schicken.

Alle bekommen das Paket und müssen entscheiden, ob es für sie ist.

### 1. DESKTOP FIREWALL

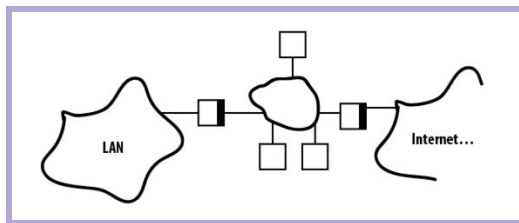
Rechner mit diesem Programm



## 2. HARDWARE FIREWALL

Rechner betreibt nur die Firewall.

Recht sicheres Verfahren



## PAKETFILTER

= Programm, das vorbeilaufende Pakete untersucht.

### Arten

#### 1. Zustandslose Pakete

Alle Pakete sind unabhängig voneinander.

#### 2. Zustandsbehaftete Pakete

Infos von behandelten Paketen werden gespeichert.

### Bekannt

Protokollanalysatoren.

Entwickelt für Datei-Admins.

Wireshark.

## WIRESHARK-BEISPIEL

The screenshot shows the Wireshark interface with a packet capture filter set to `ip.addr == 192.168.2.157 && icmp`. The packet list displays several ICMP Echo (ping) requests and replies. The selected packet (No. 6086) is an ICMP Echo (ping) request from 192.168.2.157 to 66.249.93.104. The packet details pane shows the following information:

- Frame 6086 (74 bytes on wire, 74 bytes captured)
- Ethernet II, Src: 192.168.2.157 (00:30:f1:ae:90:7b), Dst: 192.168.2.1 (08:00:f1:1f:10e:5b)
- Internet Protocol, Src: 192.168.2.157 (192.168.2.157), Dst: 66.249.93.104 (66.249.93.104)
- Internet Control Message Protocol
  - Type: 8 (Echo (ping) request)
  - Code: 0
  - Checksum: 0x475c [correct]
  - Identifier: 0x0300
  - Sequence number: 0x0300
  - Data (32 bytes)

The packet bytes pane shows the raw data in hexadecimal and ASCII:

```
0000 00 30 f1 f5 08 5b 00 30 f1 ae 90 7b 08 00 45 00  .0...[.0 ...{..E.
0010 00 3c 0f d3 00 00 80 03 c7 47 c0 a8 02 9d 42 f9  .<.....G...B.
0020 5d 68 80 40 00 00 00 00 00 00 00 00 00 00 00  .].....
0030 5d 68 80 40 00 00 00 00 00 00 00 00 00 00 00  .].....
0040 f7 01 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e  .wbcdefgh
```