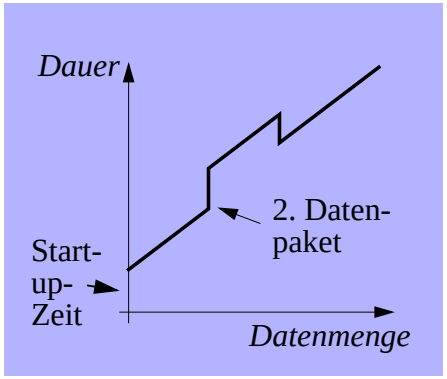
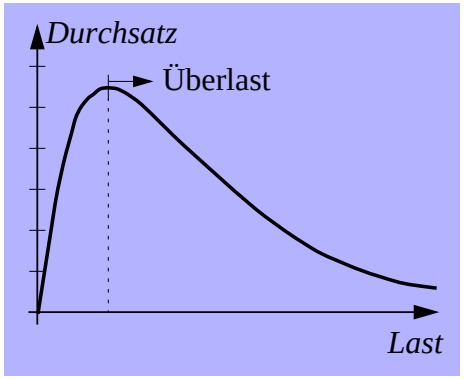
# Leistungsmerkmale von Rechnernetzen

- *Übertragungszeit*
- *Bandbreite* (bzw. “Bitrate”)
- *Durchsatz*
- Daneben spielen auch noch andere Qualitätsmerkmale eine Rolle, z.B. Fehlerrate, Ausfallsicherheit, Kosten



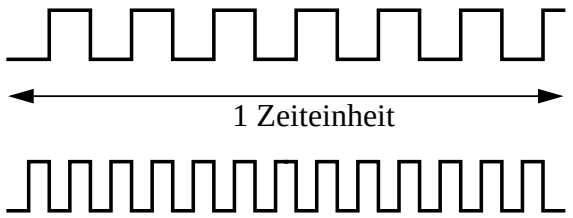




```
----www.inf.ethz.ch PING Statistics----
round-trip (ms)  min/avg/max = 0.2/0.3/1.6, 0% packet loss

----www.ethz.chPING Statistics----
round-trip (ms)  min/avg/max = 0.9/1.4/3.3, 0% packet loss

----www.inf.fu-berlin.de PING Statistics--
round-trip (ms)  min/avg/max = 57/66/96, 7% packet loss

----www.cs.berkeley.edu PING Statistics----
round-trip (ms)  min/avg/max = 297/338/406, 10% packet loss

----services.canberra.edu.au PING Statistics----
round-trip (ms)  min/avg/max = 636/990/1646, 33% packet loss
```



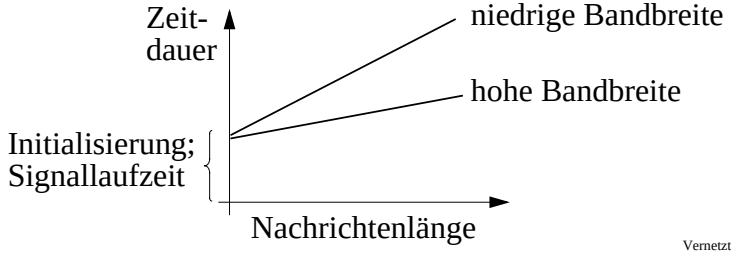
# Bandbreite (“Daten- oder Bitrate”)

- Datenmenge, die pro Zeiteinheit übertragen werden kann
  - Beispiel klassisches Ethernet: 10 Mb/s, Fast Ethernet 100 Mb/s
- Durchsatz = Tatsächlich übertragene Datenmenge / Zeit
  - Unterscheide Bandbreite des Mediums (technisch / physikalisch begrenzt) und Durchsatz vom Sendeprozess zum Empfangsprozess
  - Engpass oft durch Software, die auf verschiedenen Ebenen u.U. mehrfach jedes Bit anfassen (z.B. kopieren) muss
- Achtung: Begriff “Bandbreite” (in etwas anderer Bedeutung) auch bei Analogkanälen; i.a. gemessen in Hz
  - Bsp. Telefon: zwischen 300 Hz und 3400 Hz --> 3100 Hz Bandbreite
- Bandbreite veranschaulicht durch “Bitbreite”:



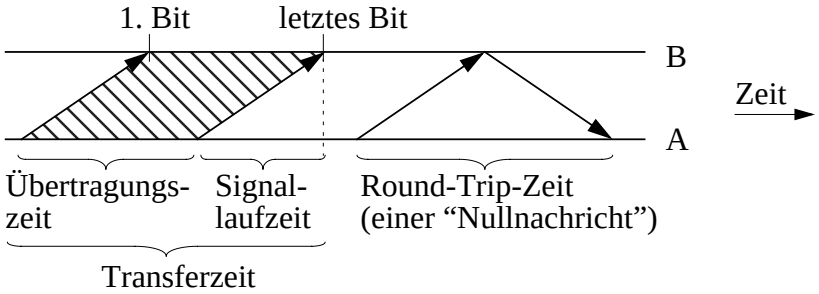

*1 Mb/s*
(jedes Bit eine Mikrosekunde breit)

*2 Mb/s*
(jedes Bit 0.5 Mikrosekunden breit)



# Delay

- Synonym oft: Verzögerung; Transferzeit; Latency,...




- Zeitbedarf, um eine Nachricht von A nach B zu senden
  - Unterscheide: Ankunft des ersten / des letzten Bits einer Nachricht
  - oft bzgl. Zeit für “Round-Trip” interessiert
  - Beispiel Round-Trip-Zeit Europa-USA ca. 300 ms

Satellit oder Untersee-kabel ?

- Zusammensetzung der Verzögerung
  + Signallaufzeit auf Medium (Entfernung / Lichtgeschwindigkeit)

- $3.0 \times 10^8$ m/s im Vakuum
- $2.3 \times 10^8$ m/s im Kupferkabel
- $2.0 \times 10^8$ m/s im Lichtwellenleiter

  + Übertragungsdauer der Nachricht (Grösse / Bandbreite) (wenn es auf die Ankunft des letzten Bits einer Nachricht ankommt)
  + Queueing-Effekte (kurzzeitige Speicherung von Datenpaketen in Puffern; contention bei den Switches etc.)
  + Software-Overhead und Initialisierungszeiten (insbesondere bei Prozess-Prozess-Betrachtung; Einrichten einer Verbindung, Nachladen von Softwarekomponenten, Ermittlung des Routings...)
  + ggf. Error-Recovery, Protokoll-Overhead etc.

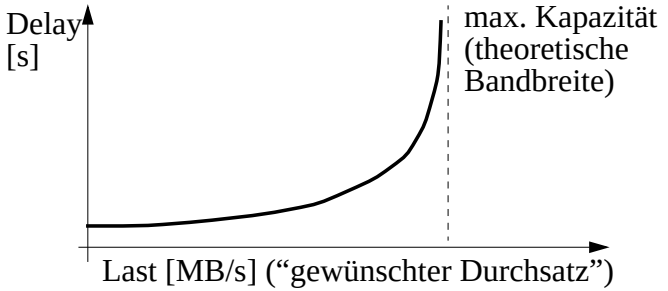


# Bandbreite <--> Delay

- Für kurze Nachrichten (“Null-Message”) ist Delay entscheidend; Bandbreite ist vernachlässigbar
  - z.B. Echo eines Tastendrucks bei remote login
  - Bsp.: Europa-USA mindestens 100 ms; pro Byte sind es bei 1Mb/s zusätzlich 8 µs, bei 100 Mb/s zusätzlich 0.08 µs --> irrelevant!

- Für grossvolumige Daten (z.B. Bilder) ist die Bandbreite der dominierende Faktor
  - Bsp.: Bei 20 s Übertragungszeit für ein Bild spielt Verzögerung von 1 ms oder 100 ms praktisch keine Rolle
  - Frage: Kann die *Bandbreitenerhöhung* einer stark belasteten Leitung, über die viele logische Verbindungen gemultiplext werden, den *Delay* der logischen Verbindungen merklich verringern? (Bsp.: Transatlantik-Verbindung im Internet für WWW)

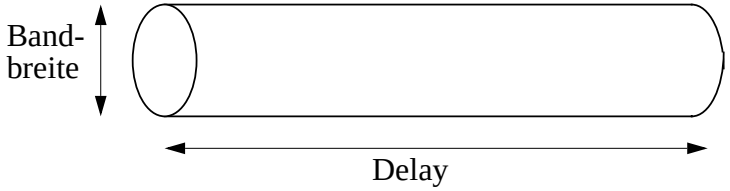
- Delay hängt oft von der Last bzw. der in Anspruch genommenen Bandbreite ab; typische Situation in LANs:




- Beachte: Grössere Verzögerung kann den effektiven Durchsatz auf höherer Ebene vermindern (z.B. Warten auf acknowledgements vor dem Senden des nächsten Blocks)!

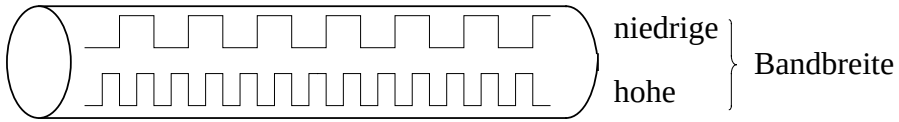
# Bandbreite × Delay

- Bandbreite-Delay-Produkt




- Beispiel: 100 ms Delay und 45 Mb/s Bandbreite
  --> 562 KB Daten, die unterwegs sind

  - entspricht Anzahl der Bits in der Röhre
  - Anzahl der Bits, die der Sender sendet, bevor der Empfänger das erste Bit erhält
  - Acknowledge oder “Stop, Puffer voll!” kommt erst nach vielen bereits gesendeten Bits
  - Implikationen für Protokolle (Paketgrössen, Acknowledgements, Window-Grösse etc.)




---

- Typische Anwendungsanforderungen:

  - Mindestbandbreite (burst rate <--> peak rate)
  - Varianz des Delay (“Jitter”): Stört bei Video und Audio (Ursache: Queueing bei Switches; asynchrones Multiplexen)

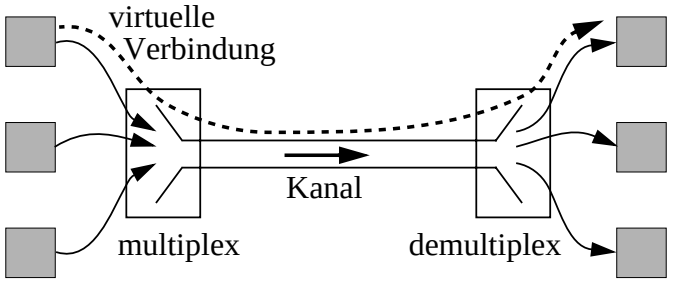


# Multiplexen

- Gemeinsame Nutzung von Kommunikationseinrichtungen (Verbindungen, Switch...) und anderer Ressourcen für mehrere Anwender

  - vgl. Timesharing einer cpu

- Insbesondere mehrere virtuelle Verbindungen / Kanäle über eine (längere) physische Verbindung

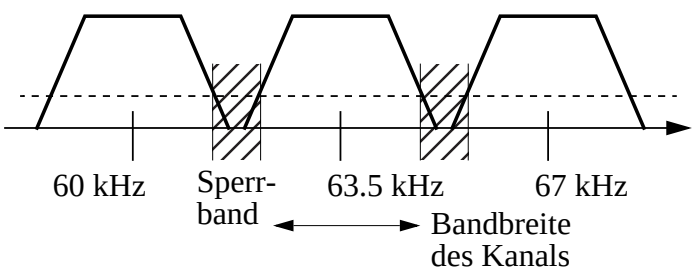


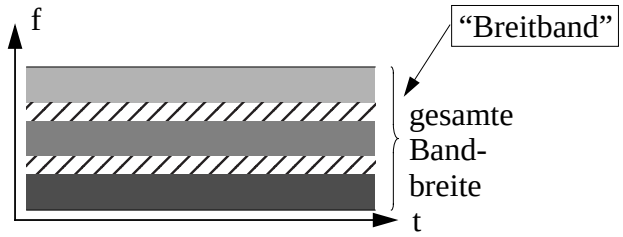

- Vorteile:

  - höhere Auslastung von physischen Verbindungen durch gegenseitigen Ausgleich logischer Verbindungen
  - Kostenvorteil durch gemeinsame Nutzung, da fallende Kosten pro b/s mit steigender Übertragungsrate

# Frequenzmultiplex

- FDM: Frequency Division Multiplexing



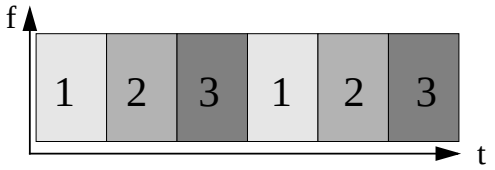




- Aufteilung der Übertragungskapazität in mehrere *Frequenzbänder* mit dazwischenliegenden *Sperrbändern*
  - vgl. TV-Kabel: verschiedene Signale mit einer Bandbreite von je ca. 7 MHz über ein einziges Medium
- Niederfrequente Signale werden durch *Modulation* in das entsprechende Frequenzband "gehoben"
- Klassisches Anwendungsgebiet: Kabelfernsehen



# Synchrones Zeitmultiplex

- STDM: Synchronous Time Division Multiplexing




- Aufteilung in Zeitschlitze
- zeitversetzte Übertragung der Signale der Kanäle
- zyklische Zuteilung

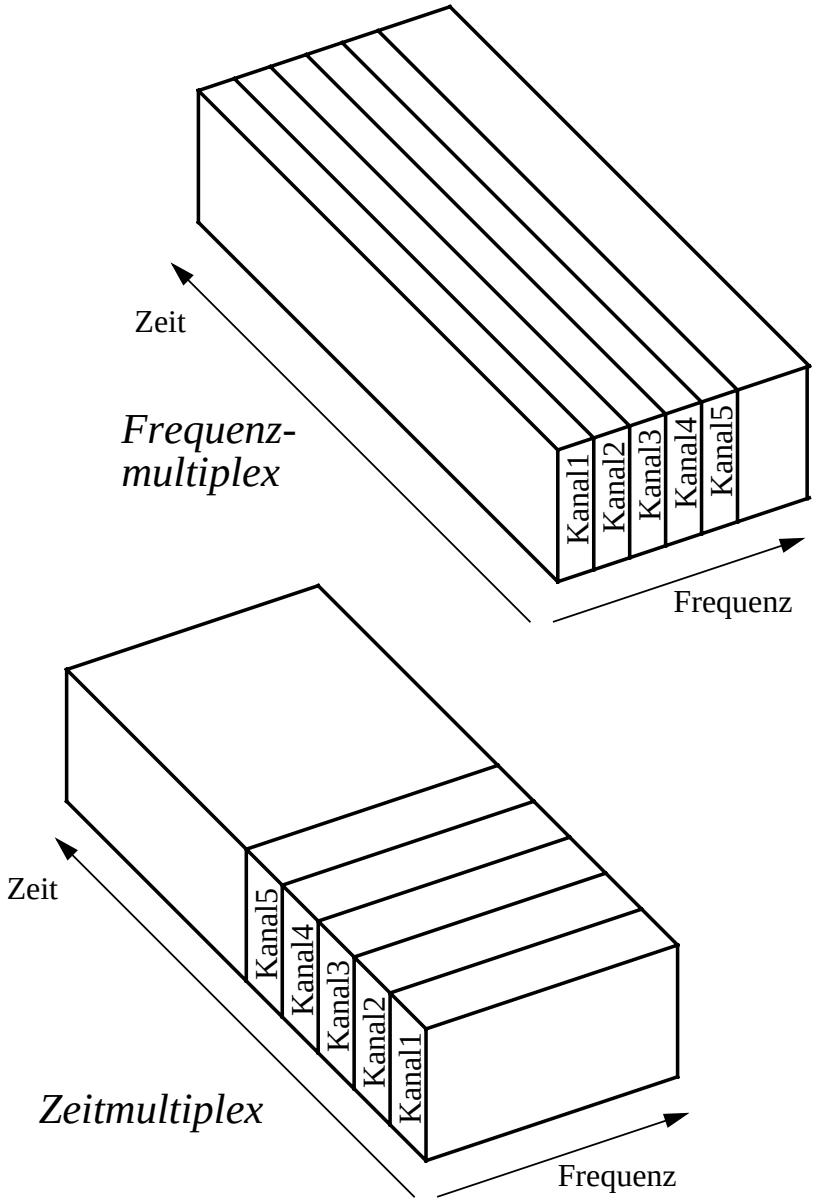
- Bit- oder blockweise Verschränkung ("interleaving")
- Angewendet bei digitalen Signalen

---

Nachteile von FDM und STDM:

- vordefinierte Zahl verschiedener logischer Kommunikationskanäle
- Anteil an der Gesamtkapazität bleibt ungenutzt, wenn über einen Teilkanal nichts fliesst (insbesondere bei starker Varianz; vgl. z.B. Anfordern / Lesen einer WWW-Seite)

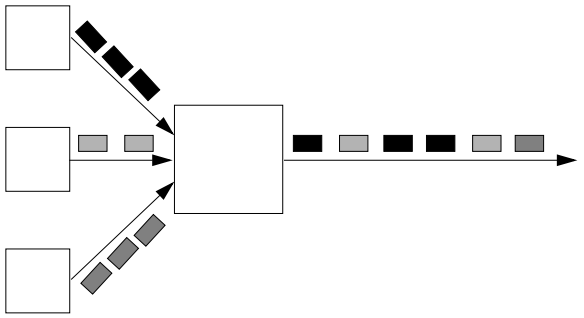
# Frequenz- und Zeitmultiplex



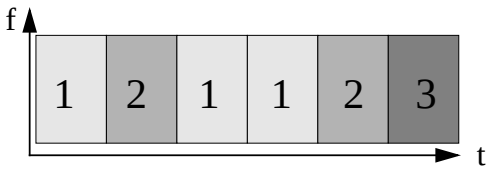



# Asynchrones Zeitmultiplex

- ATD: Asynchronous Time Division
- Auch *statistisches* Zeitmultiplex genannt

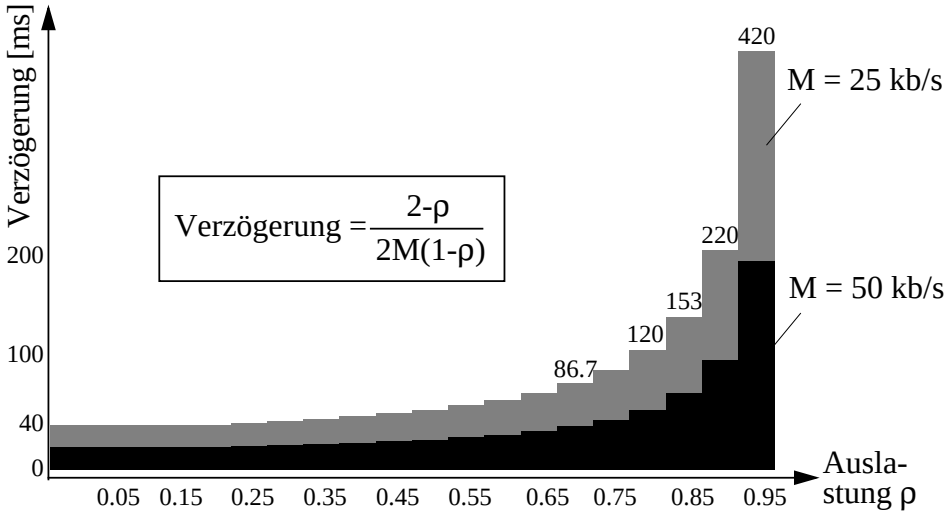


- Zuteilung von Zeitschlitzen (i.a. fester Grösse) nach Bedarf
- Paket = zusammenhängend zu transportierende Daten
- Gute Wahl der Paketgrösse? (Overhead, Zeitvarianz, Fairness...)
- Header eines Paketes muss virtuelle Verbindung kennzeichnen
- Bits pro Kanal zunächst in einem Puffer sammeln, dann stossweise (d.h. i.a. “paketweise”) abgeben
- Verschiedene Kanäle konkurrieren miteinander (“contention”)
- Pufferüberlauf (“congestion”): Pakete werden oft einfach weggeworfen



# Verzögerung bei ATD

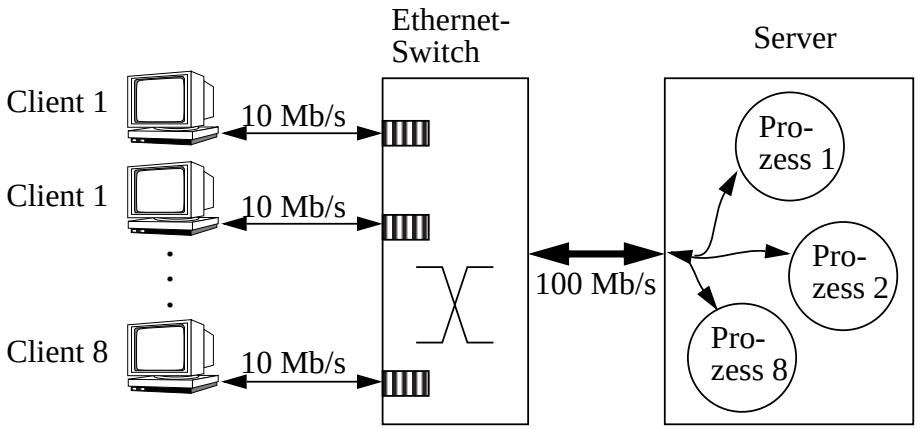
- Berechnung von Verzögerung und Wahrscheinlichkeit von Pufferüberläufen mittels Warteschlangentheorie
  - dazu i.a. typische Annahmen wie statistische Unabhängigkeit der Eingangssignale (nicht immer gerechtfertigt!), Poisson-Prozesse etc.
  - daraus Abschätzung der benötigten Puffergrösse, Abschätzung der Dienstgüte (Verzögerung, mittlere Paketverlustrate etc.)
- Beispiel: Mittlere Verzögerung einer 1000-Bit-Nachricht als Funktion der Auslastung (mit M = Bitrate):
  - Verzögerung steigt oberhalb einer Auslastung von ca. 80% drastisch an
  - Grund: Langes Warten von Datenpaketen in Eingangspuffern des Multiplexers (da auch andere Kanäle öfters gleichzeitig Daten senden)
  - 100% Auslastung ist nicht erreichbar!




- Wir behandeln die Leistungsanalyse mittels Warteschlangentheorie in dieser Vorlesung nicht näher
  - Alternative zur Analyse mittels Warteschlangentheorie: Simulation



# Ein Beispiel für ATD




- Was geschieht, wenn alle Clients gleichzeitig mit 10 Mb/s mit ihren Server-Prozessen kommunizieren?
- Was geschieht, wenn man statt 8 Clients z.B. 24 Clients an den Switch anschliesst?
  - Clients nutzen Kapazität von 10 Mb/s gar nicht ganz aus
  - Prinzip Hoffnung: Selten senden viele gleichzeitig
  - erwartete Gesamtlast am Eingang im Mittel unter 100 Mb/s
  - Lastspitzen können hoffentlich über Puffer abgefangen werden
- Konsequenzen, wenn Verzögerung auf dem Kanal vom Client zum Server-Prozess exorbitant steigt?
  - Timeouts, Verlust von Datenpaketen, Wiederholungen
  - effektive Datenrate sinkt dadurch!
  - Anwendungen stürzen u.U. ab (“Server down”)

# Kommunikationsprotokolle

- Kommunikation in verteilten Systemen geschieht ausschliesslich über *Nachrichten*

- Hierbei sind gewisse *Normen* (engl.: *standards)* zu beachten, damit die Kommunikation klappt

- *Protokoll* = Festlegung der Regeln und des algorithmischen Ablaufs bei der Kommunikation

---

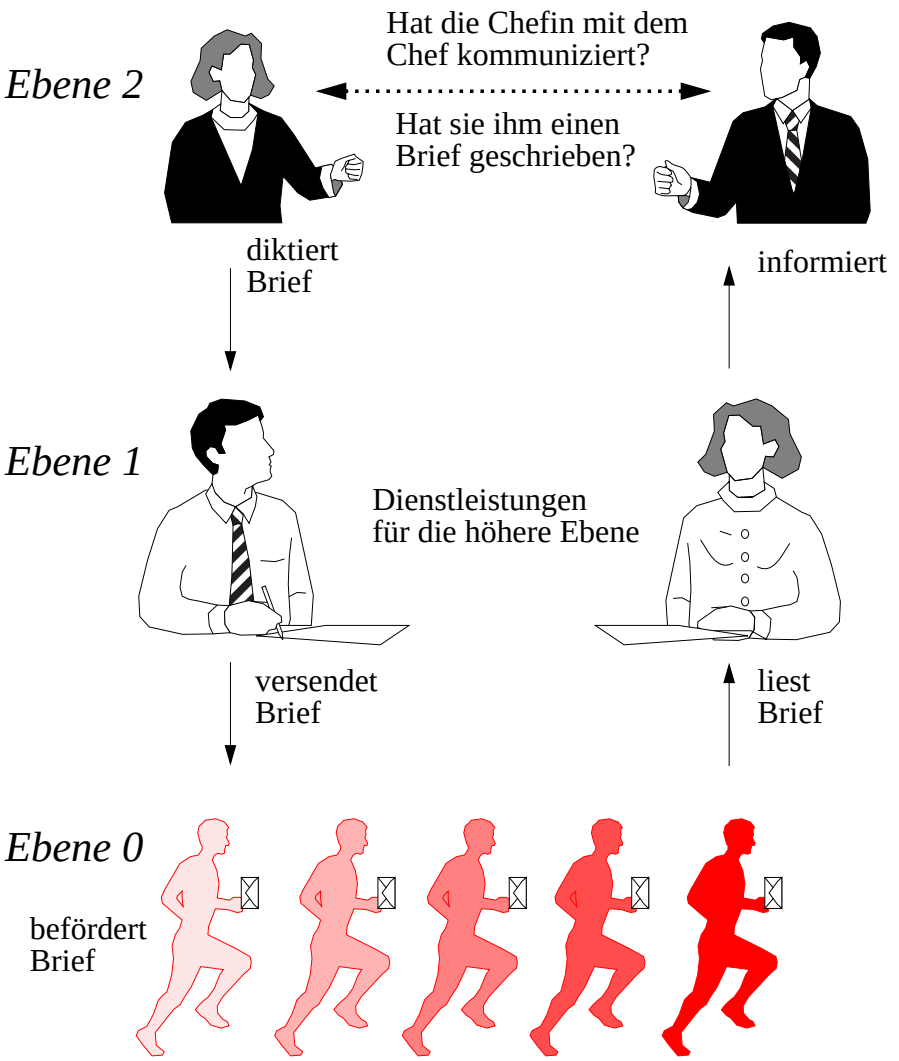
- Es müssen viele Vereinbarungen getroffen werden, z.B:
  - Steckergrösse
  - wieviel Volt repräsentieren “0” bzw. “1”?
  - ist das erste Bit vorne oder hinten?
  - was tun bei einer fehlerhaften Übertragung?
  - wird ein Rasterbild zeilen- oder spaltenweise übertragen?
  - ...

--> Vereinbarungen auf z.T. unabhängigen “Ebenen”

--> Bildung sinnvoller Schichten



# Schichten (“layers”) bzw. Ebenen



# Schichten (2)

- Eigentliche (“physische”) Kommunikation geschieht auf der untersten Ebene

- Auch mittlere Ebenen (Sachbearbeiter) kommunizieren miteinander

- Nehmen dazu *lokale Dienste* einer tieferen Ebene in Anspruch

- Ebene 0 kann ausgetauscht werden (z.B. Fax statt gelbe Post), ohne dass sich auf höherer Ebene etwas ändern muss (--> Transparenz)

- Ebenen 0 und 1 können ausgetauscht werden (z.B. wenn Ebene 2 den Telefondienst statt dessen in Anspruch nimmt)

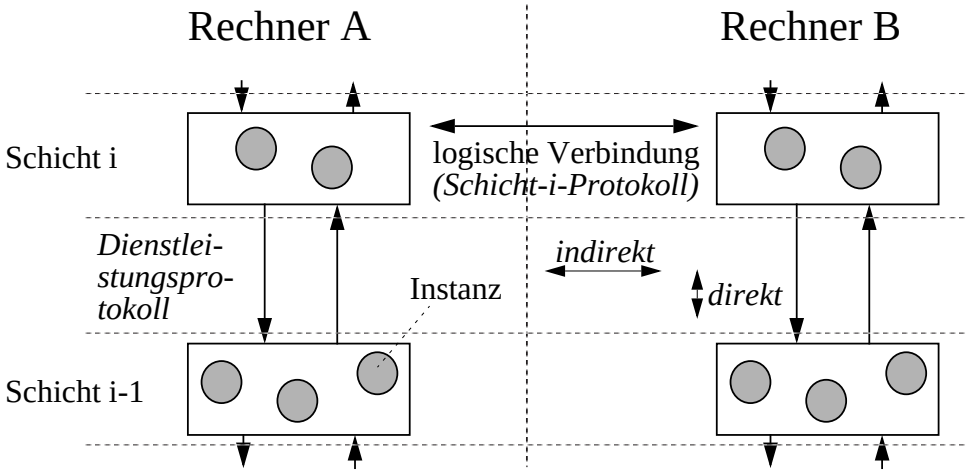
---

Vgl. auch Speditionsfirma, die verscheidene Transportdienste in Anspruch nehmen kann:

- auf jeder Ebene gelten jeweils eigene Regeln (Strassenverkehr: rechts vor links...), die aber nicht nach oben durchschlagen
- Leistungsfähigkeit eines Dienstes (Kosten, Geschwindigkeit, Güte...) sind allerdings weiter oben spürbar
- verschiedene Transportdienste bieten i.w. den gleichen Service --> sind austauschbar



# Schichtenmodell




- Schicht i benutzt Protokoll der Ebene i und *lokale Dienste*, die von der Ebene i-1 angeboten werden

- Damit der nächst höheren Schicht ein Dienst angeboten werden kann, kommunizieren Instanzen der Schicht i gelegentlich mit ihren Partnern auf entfernten Rechnern

“entity”

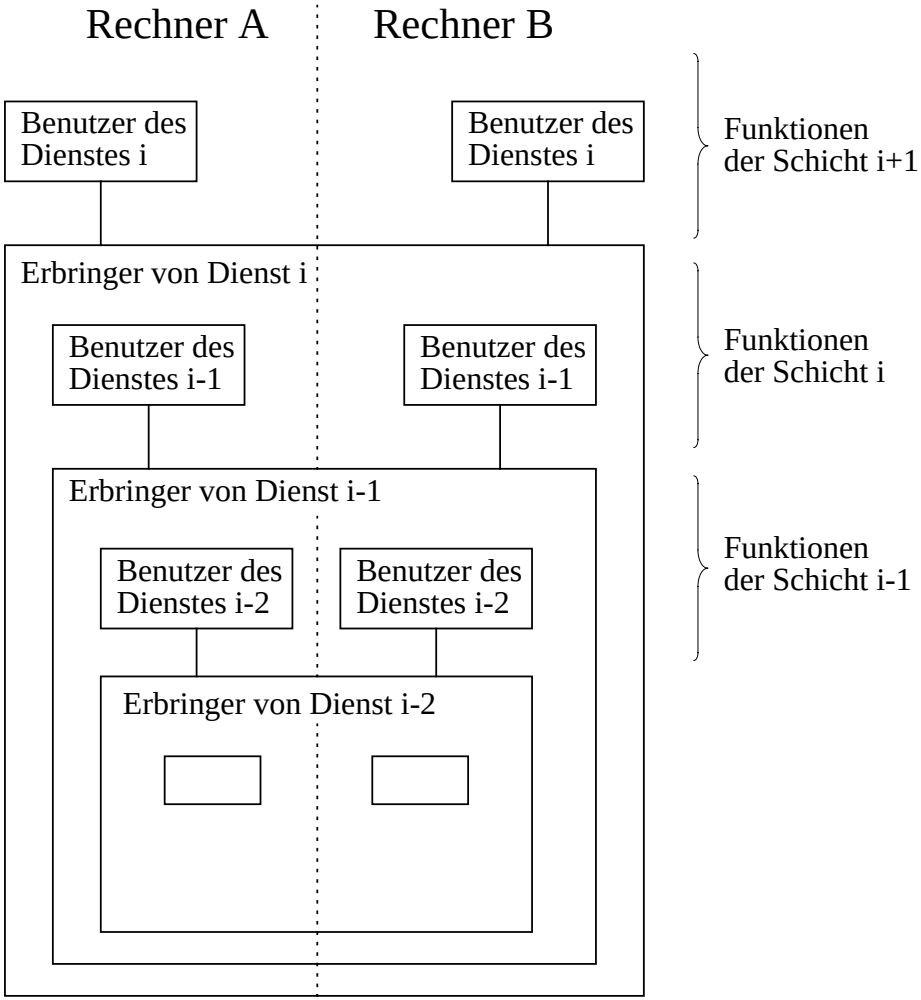
“peer entities”

- Die *logische Verbindung* zwischen entfernten Instanzen ist indirekt; die Kommunikation auf dieser Ebene wird von einem *Schicht-i-Protokoll* geregelt

- Nachrichten auf Schicht i können nicht nur von Schicht i+1 veranlasst sein, sondern auch von der eigenen Schicht (z.B. durch timeouts)

- Jede Schicht kostet... (Zusatzaufwand, “overhead”)

# Hierarchische Dienststruktur

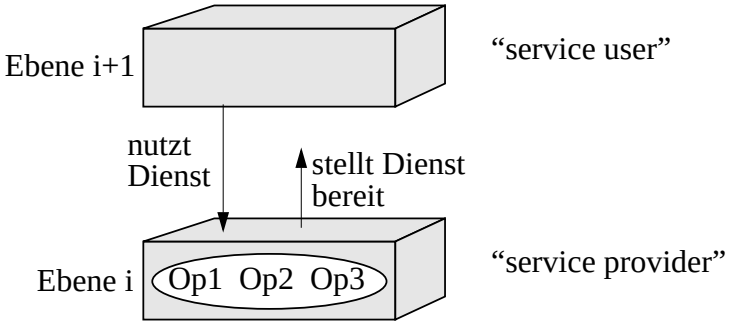





# Dienst, Interface, Protokoll

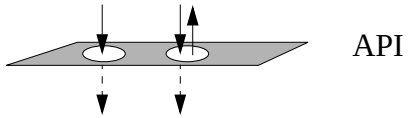
- Dienst: Menge von Operationen ("Dienstprimitive"), die eine Schicht der darüberliegenden anbietet

  - Dienstdefinition sagt aus, *was* ein Dienst tut (nicht *wie* oder *wer* ihn nutzt)

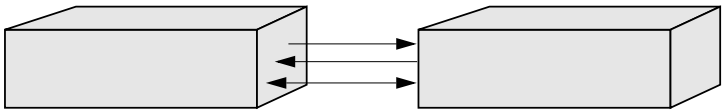



- Interface einer Schicht: *Wie* der Dienst der Schicht in Anspruch genommen werden kann

  - z.B. Parameter etc. oder Dienstleistungsprotokoll



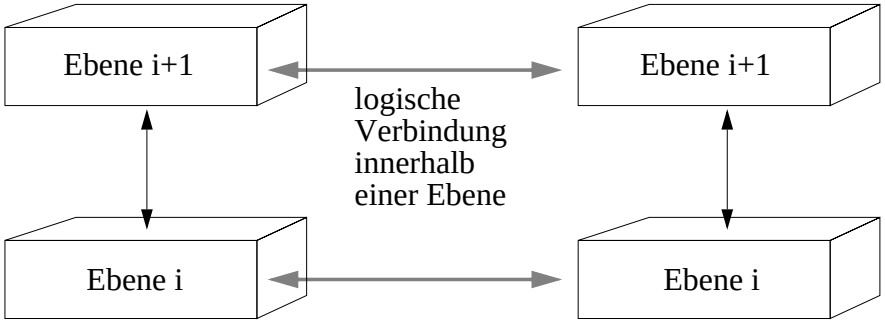

- Protokoll beschreibt die (verteilte) *Realisierung* eines Dienstes einer Schicht ("Schicht-i-Protokoll")



-Protokoll kann ("problemlos") verändert werden, solange der gleiche Dienst nach oben (mit gleichem Interface) bereitgestellt wird

# Wiederkehrende Aufgaben verschiedener Schichten

- Jede Schicht hat eine dedizierte Aufgabe; dennoch existieren u.U. gleichartige Teilaufgaben in unterschiedlichen Schichten




- Adressierung von Empfänger (und ggf. Absender)
- Fehlererkennung und ggf. -behebung
- Flusssteuerung (zu schnellen Sender bremsen)
- Paketisierung grosser Nachrichten in Maximallänge des Dienstes der tieferen Schicht
- Multiplexen
- Komprimierung
- kryptographische Verschlüsselung

*Denkübung:* Diskutieren, auf welchen Ebenen sinnvollerweise diese Aufgaben angesiedelt werden sollten!

- Diese Aufgaben werden jedoch oft unterschiedlich ("schichtspezifisch") gelöst

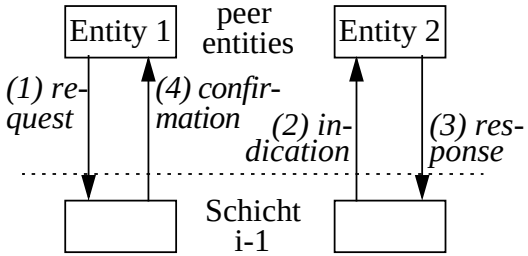
- Gelegentlich verzichtet man auf eine Aufgabe (z.B. Fehlererkennung), wenn dies von einer anderen Schicht quasi perfekt miterledigt wird
  - sogen. "End-zu-End-Argument"

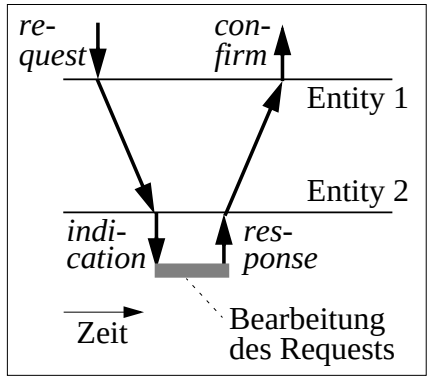


# Schichtenarchitekturen

- Schichtenbildung bei Protokollen:
  - Erleichtert Entwurf, Implementierung, Klassifikation
  - "Layering" ist gängiges Prinzip in der Informatik (Abstraktion; information hiding; Reduktion der Komplexität)

- "Kunst", eine geeignete Schichtenarchitektur zu definieren:
  - Verschiedene Grundaufgaben sollen in verschiedenen Schichten liegen
  - Ähnliche Aufgaben in gleichen Schichten
  - Jeder Schicht sollte eine klare Gesamtaufgabe zukommen
  - Anzahl der Schichten sollte nicht zu gross werden
  - Eine höhere Schicht sollte von den Aufgaben tieferer Schichten abstrahieren
  - Informationsfluss zwischen den Schichten sollte gering sein



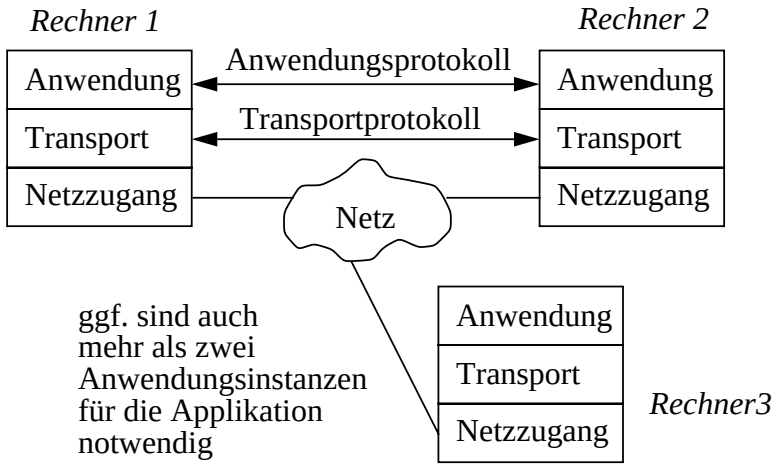




- Typen von Service-Primitiven zur Realisierung von Services (OSI-Modell):
  - *Request*: Entity auf Ebene i fordert Service von Entity auf Ebene i-1 an
  - *Indication*: Entity auf Ebene i-1 teilt Request einer i-Entity mit
  - *Response*: i-Entity antwortet auf Indication einer i-1 Entity
  - *Confirmation*: i-Entity antwortet auf Indication einer i-1 Entity

(Vgl. Herstellen einer Telefonverbindung: Nummer wählen, klingeln...
Kritik an diesem Modell: Telefone klingeln; Rechner nicht)

# Ein 3-Ebenen-Modell

- Grob lassen sich 3 Ebenen identifizieren:




- Spezifika des Netzes; Treiber der Netzkarte; Adressen der Rechner etc. werden in einer eigenen Ebene verborgen (*Netzzugangsebene)*

- Die sichere (korrekte) Datenübertragung ist Zweck der *Transportschicht*
  - ist eine Aufgabe, die damit für viele Anwendungen “gleichzeitig” erledigt wird
  - isoliert die Anwendungen von der Technologie, den Spezifika und den Unvollständigkeiten des verwendeten Netzes

- Die *Anwendungsschicht* enthält die (verteilten) Applikationen (z.B. Homebanking-Software)