Notizen PhyTA Infotech 1

Klaus Kusche, 2010 / 2011

1. Stunde

Der Computer als Digitalsystem...

... kennt nur 0 und 1:

- Spannung hoch / niedrig (Leitung, Speicher RAM, ...)
- Leitend / Nichtleitend (Speicher Flash, Taste, ...)
- Magnetisierung N->S oder S->N
- Licht / kein Licht (CD, Lichtleiter)
- Früher: Loch / kein Loch (Lochstreifen, Lochkarte)

Bitte nicht: Strom / kein Strom (technisch falsch!)

Inhalt des Kurses (ganz grob):

- Wie stellt man mit 0 und 1 Daten dar?
- Wie rechnet man nur mit 0 und 1?
- Wie arbeitet der Computer mit 0 und 1?
- Bestandteile und Funktion eines Computers (CPU, Speicher, ...)
- Grundlagen Software, Betriebssysteme
- (Alternativthemen:
  Anwendungen / Office, Programmieren, Netzwerke, Computersicherheit)

Bit / Byte: Einheit der Information:

1 Bit = 0 oder 1
(1 Speicherzelle, "wahr / falsch")
Abkürzung meist *kleines* "b" (Beispiel: Netzwerk-Geschwindigkeiten).

1 Byte = 8 Bits = acht Mal 0 oder 1
(reicht für 0...255, 1 Zeichen, "kleinste praktisch sinnvolle Informationsmenge")
Abkürzung meist *großes* "B" (Beispiel: Speicher-Kapazitäten).

1 KB = 1 KiloByte = 1000 Bytes (bei Verkäufern, $1000 = 10^3$)
oder 1024 Bytes (bei Informatikern, in Software: $1024 = 2^{10}$)
Laut Norm: 1024 Bytes = 1 KiB ("Kibibyte")

Analog:
1 MB (Megabyte):
Entweder $1000*1000$ (10^6)
oder $1024*1024=1048576$ (2^{20}) Bytes (MiB).
1 GB (Gigabyte):
Entweder $1000*1000*1000$ (10^9)

oder 1024*1024*1024=1073741824 (2^{30}) Bytes (GiB).
1 TB (Terabyte):
Entweder 1000*1000*1000*1000 (10^{12})
oder 1024*1024*1024*1024=1099511627776 (2^{40}) Bytes.
1 PB (Petabyte), 1 EB (Exabyte), ...: Na wieviel wohl....

Der Speicher eines Computers...

... enthält sehr viele einzelne Bytes,
die *fortlaufend durchnummeriert* sind:
Jedes Byte hat eine Nummer, seine ***Adresse***.

Das Binärsystem:

Stellenwert-System, mit Stellenwert 2 statt 10

0
1
10 (Übertrag!, sprich "eins-null", nicht "zehn")
11
100
usw.

Siehe altes Skript:
Wert der Zahl =
(letzte Ziffer) * 2^0 +
(vorletzte Ziffer) * 2^1 +
(vorvorletzte Ziffer) * 2^2 + ...

Problem:
1 Dezimalziffer ... rund 3,3 Binärziffern
==› Binärzahlen sind unhandlich lang und unübersichtlich
Umrechnen in Dezimalzahlen ist aufwändig, einzelnes Bit nicht mehr erkennbar
==› Andere "praktische" Darstellung für Binärzahlen gesucht!

1. Versuch: Oktalzahlen

Je 3 Bits zusammenfassen und mit 0...7 anschreiben.
Stellenwertsystem mit Stellenwert 8.

Praktisch verwendet, früher oft, heute selten.
In der Programmiersprache C: Zahl mit führender 0 wird als Oktalzahl interpretiert.

Problem:
Passt nicht zu Byte-Grenzen: 1 Byte = 2,66 Oktalziffern

2. Versuch: Hexadezimalzahlen

Je 4 Bits zusammenfassen und mit 0…9 und a…f (= 10 … 15) als Ziffern anschreiben.
Stellenwertsystem mit Stellenwert 16.
1 Byte = 2 Hex-Ziffern!

Heute für alle "Hardware-nahen" Zahlen verwendet
(und für alle "Adressen", Adressen sind meist 32-Bit-Zahlen ==› 8 Hex-Stellen).

Kennzeichnung meist mit führendem "0x…".
(in der Literatur auch: "…$_{16}$").

Zur Darstellung:

In einem Byte:
"Höchstwertiges Bit" … Stellenwert 2^7 … ganz links.
"Niederwertigstes Bit" … Stellenwert 2^0 … ganz rechts.
(bei seriellen Datenströmen: Was wird zuerst geschickt???)

Bei Zahlen mit mehreren Bytes Länge:
2 Variaten:

- **"Little Endian"**: Niederwertigstes Bytes zuerst
  (an der kleinsten Adresse im Speicher, zuerst am Netz)
  Beispiel: Prozessoren von Intel und AMD
  Zum Lesen "verkehrtherum"
- **"Big Endian"**: Höchstwertiges Byte zuerst
  (an der kleinsten Adresse im Speicher, zuerst am Netz)
  Beispiel: Daten im Internet

Englisch:
MSB … "most significant bit / byte"
LSB … "least significant bit / byte"

Umwandlung Dezimal ‹==› Binär ‹==› Hex

Siehe altes Skript:

- Von dezimal nach binär:
  Fortlaufende Division durch 2, Reste in verkehrter Reihenfolge anschreiben.
- Von binär nach dezimal:
  - Entweder mit Stellenwerten multiplizieren und aufsummieren.
  - Oder nach Horner-Methode (fortlaufende Multiplikation, links beginnend).
- Zwischen binär und hex: Je 4 Bits sind eine Hexziffer…
- Zwischen dezimal und hex:
  - Entweder mit Umweg über binär.
  - Oder direkt (so wie binär, nur mit * 16 und / 16 statt * 2 und / 2).

Rechnen mit Binärzahlen / Hex-Zahlen

... steht auch im alten Skript.
Merke vor allem:

- Wie rechnet man binär vereinfacht * 2 bzw. / 2?
- Computer ignorieren bei ganzen Zahlen fast immer den Überlauf, lassen also einfach Vielfache von 2^{32} "unter den Tisch fallen".

Darstellung negative Zahlen

... im alten Skript (Darstellung, Umrechnung).
Bei ganzen Zahlen kommt nur noch Two's Komplement vor!
Merke: Bei Two's Komplement wird im Rechner mit Vorzeichen-Zahlen genauso addiert / subtrahiert wie mit normalen Binärzahlen!

Gleitkomma-Zahlen

... auch im alten Skript.
Beachte: Rundungsfehler, nicht jede endliche Dezimal-Kommazahl hat eine endliche Binär-Gleitkomma-Darstellung!

Zeichencodierung, Strings, Steuerzeichen

... ebenfalls im alten Skript.

Beachte:

- Sortierreihenfolge von Umlauten, Prüfung `(c >= 'a') && (c <= 'z')`, usw.
- Signed / unsigned-Problematik in C.
- CR/LF-Problematik: Windows verwendet CR/LF für "neue Zeile", Linux / Unix nur LF, Mac früher nur CR

Zum Unicode:

- Ursprüngliche Codierung UCS-2:
  Fix 2 Bytes pro Zeichen (d.h. doppelter Platzbedarf bei reinen ASCII-Zeichen!)..
  Probleme:
  - Es gibt heute mehr als 2^{16} Unicode-Zeichen.
  - Darstellung hängt von Big Endian / Little Endian ab.
- Heute fast immer: Codierung UTF-8: Variabel lange Codierung mit 1-4 Bytes pro Zeichen.
  Codierung:
  - 7-Bit-Werte: 0*xxxxxxx*
  - 11-Bit-Werte: 110*xxxxx* 10*xxxxxx*
  - 16-Bit-Werte: 1110*xxxx* 10*xxxxxx* 10*xxxxxx*
  - 21-Bit-Werte: 11110*xxx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx*

  Vorteil:
  - Keine 0-Bytes in der Codierung ==> alte Konvention "0 ist Stringeende"

funktioniert auch mit UTF-8 (nicht hingegen mit UCS-2)
    - Ein ASCII-String ist auch ein UTF-8-String, weil alle ASCII-Zeichen in 1 Byte dargestellt werden.

  Problem:
    - Textlänge ist nicht mehr direkt aus Länge der Zeichenkette in Bytes ablesbar!
- Andere Darstellungs-Varianten:
    - UTF-16: Variabel lange Codierung mit 2 oder 4 Bytes pro Zeichen.
    - UTF-32, UCS-4: Fix 4 Bytes pro Zeichen.
    - UTF-7: Variabel lange Codierung mit 1-5 Bytes pro Zeichen, jedes Byte <= 127 (z.B. weil E-Mail offiziell immer noch nur 7-Bit-Texte erlaubt).

Programm-Code, AV-Daten, Prüfung, Komprimierung, Verschlüsselung

... siehe altes Skript

Zur Komprimierung:
Verlustfreie Komprimierung nutzt aus, dass die Bits in den Daten eben nicht gleichverteilt bzw. zufällig verteilt sind. Unkomprimierbar sind daher u.a.:

- Zufallszahlen, “weißes Rauschen”
- Verschlüsselte Daten (weil zur Qualität einer Verschlüsselung u.a. gehört, dass das Ergebnis möglichst zufällig aussieht und keine Regelmäßigkeiten bietet, an denen ein “Knack-Versuch” ansetzen könnte)

Zur Verschlüsselung:
Nachfolger von DES: AES (Advanced Encryption Standard, Rijndael-Algorithmus):
128, 192 oder 256 bit Schlüssellänge statt 56 Bits.

Logik / Digitaltechnik

Grundlagen

Terminologie, Geschichte usw.: Siehe altes Skript.

Ganz grob vereinfacht:
Logik: Eine Formel verknüpft boolsche Eingangsvariablen zu einem boolschen Ergebnis.
Digitale Schaltungen: Eine Schaltung verknüpft digitale Eingangssignale zu digitalen Ausgangssignalen.

Beides ist (fast) dasselbe:

- "wahr" und "falsch" entspricht Spannung "high" und "low" (1 und 0).
- Logisch "und", "oder", "nicht" usw. entspricht den Grundbausteinen digitaler Schaltungen.
- Einziger Unterschied: Digitale Schaltungen können ein "Gedächtnis" bzw. einen "Zustand" haben, d.h. die Ausgänge hängen nicht nur von den aktuellen Eingängen ab, sondern auch von der Vorgeschichte.
  Das wird durch Rückkopplungen realisiert, siehe später...

Darstellung:

1.) Als logische Formel mit "und", "oder", "nicht", ...
2.) Als Schaltung (Grundbausteine "und", ...) (kommt gleich...)
3.) Als Wahrheitstabelle: Alle Kombinationsmöglichkeiten von Eingangswerten mit den sich daraus ergebenden Ausgängen.
  Daher: 2^n Zeilen für n Eingänge, eine Spalte pro Eingang / Ausgang / Zwischenergebnis.
  Legt nur das Verhalten fest, nicht die Formel / Schaltung!
  Eine Wahrheitstabelle ==› (unendlich!) viele Formeln / Schaltungen.
4.) Als Spannungs-Zeit-Diagramm:
  (x-Achse: Zeit, y-Achse: Spannung für jeden Ein- und Ausgang)
  Legt ebenfalls nur das Verhalten fest.
  Kann das Verhalten von Schaltungen mit Gedächtnis darstellen!

Grundoperationen:

- Und (= "Konjunktion"), Oder (= "Disjunktion"), Nicht (= "Negation")
  (Vorrang: Nicht vor Und und Oder)
- Nur in der Logik: Implikation ("Wenn ... dann"), Äquivalenz ("Genau wenn ... dann")
  Achtung: Die Implikation ist nicht das, was der Hausverstand erwartet!
- Nur bei den Schaltungen: Xor, Nand, Nor

Schreibweise: Siehe Vortrag und altes Skript!
Bei Schaltungen:

- Alte Schreibweise mit den Halbrund-Symbolen
- Neue Schreibweise mit den quadratischen Kästchen

Normalformen

Reduktion auf eindeutige Formeln / möglichst kurze Formeln / möglichst wenig geschachtelte Formeln:
Normalformen (wie in der Mathematik: Bruch gekürzt und positiver Nenner, Polynom ausmultipliziert und nach Potenzen sortiert, …)

Gründe:

- In der Logik: Vergleich von Formeln (gleich wenn die Normalformen gleich sind)
- Bei Schaltungen: Kurze Signallaufzeiten, Realisierung mit Standard-Bausteinen, …

In der Logik:

- ***Disjunktive Normalform:***
  Disjunktion von Konjunktionen von allen Eingängen oder deren Negation.
- ***Konjunktive Normalform:***
  Konjunktion von Disjunktionen von allen Eingängen oder deren Negation

Beides kann man ***direkt aus der Wahrheitstabelle herausschreiben*** und umgekehrt, siehe altes Skript.

Bei Schaltungen:
NOR oder NAND als einziger Grundbaustein reichen aus, um alle möglichen Logikschaltungen zu bauen!
Und plus Nicht oder Oder plus Nicht reichen ebenfalls, Und alleine oder Oder alleine reichen nicht.

Umformen und Vereinfachen von Formeln:
Siehe Kapitel "Gesetze und Umformungen" im alten Skript.

Ermitteln möglichst kurzer DNF's durch KV-Diagramme:
Siehe Kapitel "KV-Diagramme" im alten Skript.

Logik-Schaltungen

Realisierung technisch:

- Bei mechanischen Kontakten (Relais):
  Und…Serienschaltung, Oder…Parallelschaltung, Not…Ruhekontakt des Relais
- Bei CMOS: Siehe Tafelbild:
  - Immer "gegengleich" schaltende Transistoren zwischen "low" und Ausgang und zwischen "high" und Ausgang.
  - Für NOR und NAND: "Oben" Parallel-Schaltung, "unten" Serien-Schaltung oder umgekehrt.
  - Technische Eigenschaften: Stromverbrauch im Ruhezustand nur durch Leckströme, Stromverbrauch im Umschaltmoment durch Lade- und Entladeströme und durch kurzfristige gleichzeitige Leitung der "oberen" und "unteren" Transistoren.
  - Mechanisches Äquivalent zu CMOS: Kolbenventil betätigt durch Wasserdruck am "Eingang" und Feder-Gegendruck, gegengleiche Varianten:

Durchlass im Ruhezustand (drucklose Position) oder im Druckzustand.

Für reale Schaltungen:

- Keine Eingänge offen lassen (sonst undefinierter Zustand!): Jeder Eingang muss mit einem externen Eingangssignal, einem Ausgang, oder konstant 0 oder 1 verbunden sein.
- Keine zwei Ausgänge direkt verbinden (werden kaputt und liefern undefinierte Summe!). Jeder Ausgang kann unverbunden bleiben, oder mit einem oder mehreren Eingängen und/oder einem externen Ausgangssignal verbunden sein.
- Keine zwei externen Eingangssignale direkt verbinden (Rückwirkung des einen auf den anderen!).
- Rückkopplungen, "Kreise": Siehe später: Führt zu Schaltungen mit Gedächtnis oder zu instabilen Schaltungen (die schwingen, d.h. trotz konstanter Eingänge ständig ihren Ausgang ändern).

Tücken realer Schaltungen:

- Undefinierte Zustände während des Umschaltens.
- Kurze Ausgangsimpulse durch verschiedene Signallaufzeiten.
- Schwingende Ausgangssignale wegen Rückkopplungen.

Halbaddierer, Volladdierer, Negierer, Look Ahead Carry

... siehe altes Skript

Schaltungen mit "Gedächtnis"

... basieren auf Rückkopplungen von Ausgängen auf Eingänge.

Darstellung:

- Entweder mit Zeit-Diagramm.
- Oder (zur Not!) mit Wahrheitstabelle, wobei die Ausgänge mit Index n als Eingänge und mit Index n+1 aus Ausgänge gelistet sind: "Wenn ich zum Zeitpunkt n einen bestimmten Zustand der Ein- und Ausgänge habe, so habe ich zum Zeitpunkt n+1 (d.h. einen Takt oder eine Gatterlaufzeit später) den angegebenen Zustand auf den Ausgängen."

Terminologie (oft schlampig!):

- **Latch**: Reagiert auf Pegel / Zustand der Eingänge.
- **Flip-Flop**: Reagiert auf Flanken der Eingänge (meist eines Takt-Eingangs), wobei immer nur eine der beiden Flanken wirkt:
    - 0-1-Änderung ... "positive" oder "steigende" Flanke
    - 1-0-Änderung ... "negative" oder "fallende" Flanke

Überbegriff für beides: "**Bistabile Kippstufen**"
(***bi*** stabil ... zwei stabile Zustände, nämlich "ein" und "aus")

Eigene Symbole, meist mit *zwei* Ausgängen
(Q und ¬Q, d.h. die beiden Ausgänge sind genau gegenteilig).

RS-Kippstufe:

- Zwei gekreuzte NOR (oder NAND mit je einem invertierten Eingang).
- Zwei Eingänge:
  R ("reset") ==› eine 1 auf R schaltet auf 0
  S ("set") ==› eine 1 auf S schaltet auf 1
- Sind R und S 0, bleibt der letzte Zustand erhalten.
- Sind beide 1, ist der Zustand undefiniert
  (und auch der Folgezustand, wenn beide gleichzeitig auf 0 gehen).
- Der Anfangszustand ist undefiniert
  (darum werden solche Schaltungen meist leicht asymetrisch gebaut).
- Kein Takt-Eingang, reagiert "sofort".

RS-Latch:

- Zusätzlich ***Takt-Eingang C*** ("clock"):
  R und S werden über je ein UND mit C verknüpft
  und dann erst auf eine normale RS-Kippstufe geführt.
  ==› R und S werden erst dann wirksam, wenn der Takt 1 ist.
  ==› Solange C 0 ist, ändern sich die Ausgänge nicht.

RS-Flip-Flop:

- ***Zwei RS-Latches nacheinander***, das zweite mit negiertem Takt C.
  ==› R und S werden gespeichert, wenn C 1 ist,
  aber der Ausgang ändert sich erst bei der 1-0-Flanke von C
  (entsprechend den Werten von R und S genau im Moment dieser Flanke).

D-Latch:

- ***Nur ein Eingang D*** ("data").
- Intern RS-Latch, wobei S mit D und R mit negiertem D gespeist wird.
  ==› Elementarer Speicher: Wenn C 1 ist, wird der aktuelle Wert von D gespeichert.

D-Flip-Flop:

- ***Zwei D-Latches*** mit invertiertem Takt nacheinander.
  ==› Speichert den Wert von D im Moment der Flanke von C.
  (je nachdem, ob die Negation von C am ersten oder zweiten Latch ist:
  0-1-Flanke oder 1-0-Flanke).

JK-Flip-Flop:

- RS-Flip-Flop, wobei die beiden Eingänge J und K ***mit den jeweils gegenteiligen Ausgängen UND-verknüpft*** als R und S dienen.
  ==› Dadurch wird verhindert, das R und S beide gleichzeitig 1 sein können.
  ==› Wenn J und K beide eins sind, wechselt der Ausgang bei jeder Taktflanke,
  sonst ist das Verhalten gleich wie beim RS-Flip-Flop.

Anwendungen von Flip-Flops

Siehe altes Skript!

Schieberegister:
Bei jedem Takt rutschen alle Bits ein Flip-Flop weiter,
vorn kommt eins rein, hinten eins raus.
Anwendung:

1.) Verwandlung paralleler in serielle Daten und zurück.
  Heute von großer Bedeutung, weil die Datenübertragung fast immer seriell ist

(PCI-Express, USB, SATA, DVI, …), die Datenspeicherung und -verarbeitung (RAM, Cache, CPU, Grafikchip) hingegen parallel ist.

2.) Berechnung/Prüfung von CRC- oder ECC-Prüfsummen (auf Platte, Ethernet, …):
Sind durch trickreich rückgekoppelte Schieberegister leicht zu berechnen:
Wenn das XOR vom Ausgang des Schieberegisters und nächstem Datenbit 1 ist, werden bestimmte Bits im Schieberegister invertiert geschoben, sonst unverändert.
   - Generierung: Daten werden durch das Schieberegister geschoben
     ==› CRC steht am Ende im Schieberegister, wird hinter Daten nachgeschoben.
   - Prüfung: Daten samt CRC werden durch das Schieberegister geschoben
     ==› Inhalt sollte am Ende 0 sein.

3.) Laufschriften, Lauflichter und dergleichen.

Zähler:
Jede Stufe “tickt” halb so schnell wie die vorige
==› Erste Stufe = hintestes Bit usw.
Anwendung:

1.) Zählen:
Meist zyklisch: Bestimmte Ausgangswerte generieren ein Reset aller Flip-Flops.
Problem Signallaufzeit ==› Siehe altes Skript, Wikipädia “Synchronzähler”

2.) Takt teilen: Analog

Rechnerarchitektur

Geschichtliches zur Entwicklung der Hardware

Siehe altes Skript, Kapitel 1.3 !

Grobaufbau "klassisch" (am Beispiel des Ur-PC)

An ***einem zentralen Bus*** (PC: ursprünglich ISA, viel später PCI) hängen:

- Die CPU
- Der Speicher (RAM, ROM=Flash für Boot&BIOS, CMOS-RAM für BIOS-Einstellungen)
- Alle Schnittstellen:
    - Platten-Interface (bzw. Floppy, Band, CD, CF-Karte, ...)
    - Grafik-Karte, Soundkarte
    - Externe Schnittstellen (USB, Ethernet, WLAN, ...)
      (früher: Serielle & parallele Schnittstelle, PS/2-Schnittstelle)
    - Uhr, Taktgeber usw.

Die Weiterentwicklung des PC

Erste Verkomplizierung (386, 486, Ur-Pentium):
Grund:

- Externe Caches
- "Alte" und "schnelle" Schnittstellen

==› ***Mehrstufige Bus-Hierarchie:***

- Ganz schneller Bus von der CPU zu Cache und externer FPU (Gleitkomma-Einheit)
- Schneller Bus vom Cache zum Speicher
- Mittelschneller Bus (EISA, PCI) vom Cache zu Grafik und schnellen Schnittstellen
- Langsamer Bus (ISA) von EISA/PCI zu langsamen Schnittstellen

Dann über 10 Jahre übliches PC-Design:
Grund:

- Vereinfachung: Caches und FPU wandern wieder in die CPU
- Hochintegrierte "Chipsätze" entstehen
- Mehrere CPU-Sockel
- Bus-Anforderungen und Bus-Vielfalt wachsen

==› Intel-Design mit Chipsatz (***"Northbridge" und "Southbridge"***)

- CPU + Cache auf einem Chip
- Ganz schneller proprietärer Bus zwischen allen CPU-Chips und der Northbridge ("Frontside-Bus")
- Northbridge hat
    - Speicherinterface (meist 2 davon zur Durchsatzerhöhung)
    - Eingebaute Grafik oder Grafikkarten-Interface (AGP, PCI-Express)
    - Schnellen proprietären Bus zur Southbridge
- Southbridge hat
    - PCI-Bus sowie ISA-Bus oder LPC-Bus zu BIOS und zu alten Schnittstellen (Par., ser., PS/2, Uhr, Floppy, ...)

- Einige eingebaute Schnittstellen (Platteninterface, USB, Sound...)

==› AMD-Design: So ähnlich, aber (siehe nächstes Kapitel)

- Speicher hängt direkt an der CPU
- Punkt-zu-Punkt-Verbindungen ("Hyperchannel") statt Bus zwischen CPU's und von der CPU zum Chipstaz

Grobaufbau PC "modern"

Probleme:

- Busse mit mehreren Anschlüssen sind elektrisch bei hohem Takt schwierig
  ==› Umstieg von Bussen auf viele einzelne Punkt-Zu-Punkt-Verbindungen
- Parallele Verbindungen brauchen viele Leitungen (32/64 Bits = viel Platz, viele Pins, viel Wärme) und machen Timing-Probleme (weil alle Leitungen exakt im selben Takt "ticken" müssen, d.h. gleich lang und ähnlich geformt sein müssen)
  ==› Umstieg von parallelen auf serielle Verbindungen
  Die dazu notwendige komplexe und schnelle Logik (Taktraten im GHz-Bereich!) war vor Jahren noch undenkbar, heute fällt sie kaum ins Gewicht!

==› Aus PCI (paralleler Bus) wurde PCI-Express (einzelne Punkt-Zu-Punkt-Kanäle)
==› Aus PATA (parallel, 2 Platten) wurde SATA (seriell, einer pro Platte),
aus SCSI wurde SAS oder FC (Storage-Interface im Server-Bereich).
==› Intel: Der proprietäre schnelle Bus zwischen den CPU-Sockeln und der Northbridge wurde durch proprietäre Punkt-zu-Punkt-Verbindungen abgelöst.
==› Die parallele Schnittstelle wurde von USB abgelöst.
==› Die parallele CF-Karte wird durch die serielle SD-Karte ersetzt.
==› ...

Außerdem:

- Der Speicherzugriff auf dem Umweg über die Northbridge ist zu langsam
  ==› Der Speicher hängt wieder direkt am CPU-Chip,
  getrennt von allen anderen Bussen.
  (AMD weiß das seit vielen Jahren...)
- Gleiches gilt für die Grafik über die Northbridge
  ==› Grafikkarte bzw. Grafikkarten-Interface wandern direkt auf den CPU-Chip
  ==› Northbridge fällt weg (diese Umstellung beginnt erst)
- Aber: Durch den Wegfall des gemeinsamen Frontside-Busses zwischen bis zu 4 Prozessoren und der Northbridge und die Anbindung des Speichers an die CPU's:
  ==› Es gibt jetzt "lokalen" (an eigener CPU)
  und "nichtlokalen" (an anderer CPU) Speicher!
  (aus Programmsicht logisch kein Unterschied, aber in der Performance schon: Nichtlokaler Speicher ist mindestens 50 ns langsamer)

Busse

Klassifizierungskriterien:

- **Parallel** (8 / 16 / 32 / 64 Bits gleichzeitig auf ebensovielen Leitungen) oder **seriell** (Bits einzeln nacheinander auf 1 (meist 2) Leitungen)?
  (z.B. "USB" = "Universal Serial Bus")
- Elektrisch: **Einfache** oder **differentielle** (2 gegengleiche Leitungen) Signale?
- Zwischen **mehreren Geräten** oder **Punkt-zu-Punkt**?

- **Multi-Master** oder **Master-Slave**
  (wie viele Teilnehmer dürfen aktiv Daten verschicken oder anfordern?)?
- **Einzelne Bytes** / Worte (**direkt adressiert**) oder nur ganze **Blöcke** (z.B. 512 Bytes)?

Was überträgt ein Bus:

- **Daten** (ev. mit Paritätsbit zur Prüfung)
- **Adressen** (wohin gehören die Daten im Speicher
  bzw. welche Schnittstelle wird angesprochen?)
  ==› Anzahl der Adressleitungen bestimmt maximal ansprechbare Speichermenge!
  (32 Bit Adressen ==› max. 4 GB = 2^{32} Bytes Speicher)
  Achtung: Bei x86 *getrennte Adressen* für Speicher (32/64 Bit) und I/O (16 Bit)!
- **Kontrollinformation:**
  - Schreiben oder Lesen?
  - I/O- oder Speicheradresse?
  - 1, 2, 4, ... Bytes?
  - Takt, Wartezyklen, ...
  - Bus Arbitration ("wem gehört der Bus als nächstes?")
  - Interrupt
    ("CPU, bitte tu etwas für mich! - Habe neue Daten, bin fertig, Fehler, ...")
    ==› Unterbricht das gerade laufende Programm und startet eine Funktion des Treiber für diesen Baustein ("Interrupt Service Routine")
    ==› Höchst-priorer und am meisten zeitkritischer Code des Systems!
  - DMA Request (Bereit zum Datentransfer)

Heute werden Adressen und Daten nacheinander auf denselben Leitungen übertragen, früher waren das getrennte Leitungen ("Datenbus" und "Adressbus").

Leistungsbestimmende Faktoren: **Busbreite * Takt**
Leistungsbegrenzende Faktoren (wegen physikalischer Eigenschaften):

- **Leitungslänge**
- **Anzahl der Teilnehmer**

Typische Geschwindigkeiten:

- Ur-PC: 1 Byte * 4,77 Mhz
- ISA: 2 Byte * 8 MHz
- PCI: 4 Byte * 33 MHz im PC, 8 Byte * 133 MHz in Servern
  (133 MHz bei 1 Slot, 100 MHz bei 2 Slots, 66 MHz bei 4 Slots)
- PCI Express 2.0:
  1 Bit * 5 GHz (max. 500 MB/s wegen Overhead) pro Adernpaar,
  max. 16 Adernpaare pro Richtung
- Frontside-Bus: Aktuell 8 Byte * 1600 Mhz (genauer: 400 MHz, 4 Transfers / Takt)
- RAM: Aktuell 2 Busse mit 8 Byte * 1600 MHz
  (Grafikkarten: Bis 64 Byte * 2000 MHz)

Beispiele:

- ISA, EISA, PCI, AGP, PCIe
- Cardbus, CF-Card, ...
- Der "Bus" zu den RAM-Chips

- LPC, I2C

Wie werden große Datenmengen von / zu Schnittstellen übertragen (Platte, Ethernet, …)?

- **Programmed I/O**: (bei PATA-Schnittstelle: PIO-Mode)
  - Die CPU schaufelt die Daten per Softwareschleife Byte für Byte…
  - Im Extremfall: "Polling"
    Vor jedem Byte softwaremäßig in Warteschleife die Bereitschaft prüfen!

  ==› Schlecht! (belegt die CPU, langsam)
  ==› Heute in PC's nicht mehr verwendet (in Microcontrollern schon)!
- **DMA**: (bei PATA-Schnittstelle: DMA-Mode)
  - Der DMA-Controller (früher ein eigener Baustein, heute im Chipsatz) wird vorher mit Quelladresse, Zieladresse und Anzahl der Bytes programmiert.
  - Die Schnittstelle signalisiert mit "DMA Request", dass sie jetzt Daten hat oder will.
  - Der DMA-Controller übernimmt den Bus von der CPU und kopiert die Daten.
  - Die Schnittstelle oder der DMA-Controller signalisieren dem Betriebssystem per Interrupt, dass die Daten fertig übertragen sind.
- **Busmaster**: (bei PATA-Schnittstelle: UDMA-Mode)
  - Die Schnittstelle bekommt von der CPU die Adresse der Daten im Speicher.
  - Die Schnittstelle übernimmt selbst den Bus und überträgt die Daten.
  - Fertig-Meldung wieder per Interrupt.

Die CPU – Grundstruktur

Eine heutige CPU enthält:

- Die eigentliche **CPU** ("Core"), bzw. bis zu 16 Stück davon.
- **Caches** (L1-Cache, L2-Cache, teilweise auch L3-Cache):
  Schnelle, transparente Zwischenspeicher für gerade benutzte Daten, siehe später.
- Speicher**interfaces**, Interfaces zum Chipsatz und anderen CPU's:
  Frontside-Bus, Hyperchannel, …

Die eigentliche CPU enthält (im wesentlichen seit 1945 unverändert):

- **Allgemeine Register**:
  Speicher für die Daten und Adressen, mit denen gerade gerechnet wird.
  Meist 8 bis 32 Stück zu je 32 / 64 Bit, d.h. für je einen Int / je eine Adresse, plus ebensoviele für Floats.
  Schnellster Speicher im System, Zugriff verzögerungsfrei!
- **Spezielle Register**: Speicher mit vordefinierter Bedeutung.
  Das wichtigste: **PC** ("**Program Counter**") bzw. **IP** ("**Instruction Pointer**"):
  Adresse des gerade ausgeführten Befehls.
  Das zweitwichtigste: **SP ("Stack Pointer")**
  Adresse des gerade obersten Elementes am Stack
  (für lokale Variablen, Unterprogramm-Aufrufe, …)
- **Flags**: Statusbits.
  Einerseits: Information über den letzten Vergleich / das letzte Rechenergebnis

(Zero, Sign, Carry, Overflow)
==› Werden von bedingten Sprüngen ausgewertet!
(und von "Add With Carry" usw. für lange Arithmetik)
Andererseits: Zustand der CPU
(z.B. Kernel / User Mode, *"Interrupts* abgeschaltet" oder Single-Step-Mode)

- **Rechenwerk (ALU = Arithmetic / Logic Unit)**:
  Führt die Berechnungen aus (auch Vergleiche usw.)
  (meist getrennt für Integer und Float, heute bis zu 6 pro CPU)
- **Leitwerk (Steuerwerk)**:
  Steuert alle Abläufe in der CPU, holt und dekodiert die Befehle
- MMU (Memory Management Unit):
  Für die Umsetzung der virtuellen auf reale Speicheradressen.
  Siehe Betriebssysteme!

Grundsätzliche Arbeitsweise der CPU:

- Nächsten Befehl aus dem Speicher holen (Adresse steht im IP)
- Befehl decodieren.
- Falls der Befehl Speicher-Operanden hat:
  Adresse ausrechnen, Daten aus dem Speicher holen.
- Operation ausführen.
- Falls der Befehl Daten im Speicher ablegt:
  Adresse ausrechnen, Daten speichern.
- Befehlszähler (IP) inkrementieren.
- … und wieder nächsten Befehl holen.

Die Befehle:

- Maschinenbefehle (Instructions), vom Compiler aus C, … erzeugt.
- Plattform-spezifisch, je nach Prozessor-Familie unterschiedlich!
  Bei modernen Prozessoren 100-300 verschiedene Befehle
  (Befehlssatz = Instruction Set = Menge aller Befehle, die ein Prozessor versteht).
- Grobe Einteilung der Befehle:
  - Arithmetische und logische Befehle (für die ALU)
    (Rechnen, Bitoperationen, Schieben und Rotieren, Vergleichen, …)
    getrennt für Int und Float
  - Lade- und Speicherbefehle
    (Kopieren von Daten zwischen Speicher bzw. I/O-Adressen und Registern)
  - Sprungbefehle
    (Unbedingter Sprung, bedingter Sprung, Call, Return, System Call)
  - Kontroll-Befehle
    (z.B. Interrupts sperren / freigeben)
  - Heute fast immer auch: Multimedia-Befehle (Intel: "MMX", "SSE")
    (Rechen- und Speicheroperationen auf mehrere Daten gleichzeitig)
- x86: Befehle sind variabel lang, 1-9 Bytes
  Bei den meisten anderen Prozessor-Familien: Konstant 4 Bytes lange Befehle.
- Aufbau eines Befehls:
  - Befehlsfeld: Um welchen Befehl handelt es sich?
  - Operanden-Feld oder -Felder:

        - Bei arithmetisch/logischen Befehlen und Lade-/Speicher-Befehlen: Operanden, wahlweise:
            - Register
            - Direkte Konstante
            - Speicheradresse, an der die Daten geschrieben / gelesen werden sollen: absolut (Konstante) oder indirekt / indiziert (Register, Register + Offset, Register + Register, …)

          Je nach Architektur:
            - x86: Bei arithmetischen Befehlen + Lade/Speicher-Befehlen:
              Entweder 2 Register
              oder 1 Register + 1 Speicheroperand
              oder Konstante + Register / Speicheroperand
              (d.h. 1 Operand muss zugleich Ergebnis sein, sonst umkopieren)
            - Bei fast allen anderen ("RISC-Architekturen"):
              Lade-/Speicher-Befehle: 1 Register + 1 Speicheroperand
              Konstanten-Lade-Befehle: Konstante + Register
              Arithmetisch/logische Befehle: 3 Register
              (2 Operanden plus Ergebnis)
              (d.h. Berechnungen im Speicher erfordern 4 Befehle)
        - Bei Sprungbefehlen: Ziel-Adressse:
          absolut (Konstante) oder relativ (Offset) oder indirekt (Register)
- Darstellung:
    - Intern binär
    - Für den Menschen in Assembler-Notation
      (wird vom **Assembler** in Binär-Programm übersetzt):
      Zeilenweise, spaltenformatiert, 4 Spalten pro Zeile:
      Label`:` *Opcode Operanden* `;` *Kommentar*
        - Label: Sprungziel, Name einer Speicheradresse oder Variable (optional, ergibt keinen Code!)
        - Opcode: Symbolische Abkürzung ("Mnemonic") für den Befehl
        - Operanden: Label, Hex- Adresse, Int-Konstanten, Registername, …
        - Kommentar: Selbstredend …

      Beispiele:

```
LNEXT: ADD R3,(R1)  ; "Addiere den Inhalt der Speicherzelle,
                    ; deren Adresse im Register 1 steht, zu Register 3."

…
       JNE LNEXT    ; "Wenn der letzte Vergleich 'ungleich' ergab,
                    ;  spring zu der Programmstelle,
                    ; die mit dem Label LNEXT bezeichnet ist."
```

Aktuelle CPU-Familien

Folgende CPU-Familien haben heute praktische Bedeutung:

- ***x86:*** Das ist die vom Intel 8086-Prozessor (1978) und dem IBM-Ur-PC abgeleitete CPU-Architektur, die heute in allen normalen PC's und einem Großteil der Server und wissenschaftlichen Großrechner steckt.

Sie war ursprünglich (als einzige der hier vorgestellten CPU-Familien) nur eine 16-Bit-Architektur (d.h. Register, Rechenwerk usw. waren nur 16 Bit breit, nur mit Tricks konnten 20 Bit Adressen, d.h. 1 MB Speicher, angesprochen werden) und wurde im Laufe der Zeit mehrmals gründlich umgebaut und erweitert (zuerst auf 32 Bit, dann auf 64 Bit).
Der Ur-8086 hatte 29.000 Transistoren (in 3 Mikrometer Strukturbreite), steckte in einem 40-Pin-Gehäuse, und lief mit 5 Mhz Takt. Aktuelle x86-Prozessoren haben über 1 Milliarde Transistoren (in 32 Nanometer Strukturbreite) und über 1000 Pins, und sie laufen mit 3,6 GHz Takt. Außerdem wurden die Operationen schneller (die Multiplikation z.B. von über 30 auf 3 Takte).
Aktuelle Vertreter sind die Prozessoren von Intel, AMD und VIA.
Neben dem x86 hat Intel noch eine zweite, ganz andere Prozessor-Architektur: Den *Itanium* für Server. Er ist aber ziemlich erfolglos und am Aussterben.

- ***ARM:*** Das ist die heute stückzahlmäßig am weitesten verbreitete Architektur (rund 100-fache Stückzahlen des x86), denn sie kommt mit viel weniger Transistoren (==› billiger) und viel weniger Strom als ein x86 aus (je nach Geschwindigkeit zwischen 0,1 und 5 W, im Vergleich zu 130 W bei schnellen x86) und steckt daher in den meisten Handies, Navigationsgeräten, Mediaplayern und anderen mobilen Geräten, aber auch in vielen Microcontrollern und kleinen Netzwerk-Geräten (z.B. Home-Disk-Server).
  Sie stammt ebenfalls von einem Heimcomputer aus den 80er Jahren ab (Acorn RISC Machine, 1983) und ist damals wie heute eine 32-Bit-Architektur.
  ARM-Prozessoren werden von mehr als einem Dutzend Hersteller gebaut (Texas Instruments, Motorola/Freescale, Nvidia, Qualcom, Marvell, ...).
- ***MIPS:*** Diese Architektur (in 32-Bit- und 64-Bit-Varianten) wurde ursprünglich für technisch-wissenschaftliche Arbeitsplatzrechner der oberen Leistungsklasse und für Server und wissenschaftliche Großrechner entwickelt: Der bedeutendste MIPS-basierte Hersteller in den 80er/90er-Jahren war Silicon Graphics mit seinen Grafik-Workstations und Animationscomputern.
  Heute ist diese Familie mit ARM vergleichbar, sie ist beispielsweise neben ARM und x86 die dritte von Windows Mobile unterstützte Plattform. Im mobilen Sektor ist sie allerdings viel weniger present als ARM, dafür wird sie mehr in anwendungsspezifischen Mikrocontrollern eingesetzt.
  Ein Bereich, der fast ausschließlich von MIPS-Systemen beherrscht wird, sind zentrale Netzwerk-Komponenten der oberen und obersten Leistungsklassen (Router, Firewalls, VPN-Konzentratoren, ...). Hier liegen MIPS-CPU's mit bis zu 32 Cores leistungsmäßig weit vor x86-Systemen.
  Weiters steckt im chinesischen Linux-Volks-PC ein MIPS-Nachbau.
- ***PowerPC:*** Das ist eine von IBM in den 80er Jahren für technisch-wissenschaftliche Rechner entwickelte 32-/64-Bit-Architektur. Sie wurde vor allem in Apple-Computern verwendet, bevor Apple auf x86 wechselte.
  Heute steckt sie noch in einigen IBM Server-Familien und vor allem in wissenschaftlichen Größt-Rechnern, in der Sony Playstation, und in vielen Microcontrollern der oberen Leistungsklasse.
- ***IBM Großrechner:*** Diese Architektur hat ihren Ursprung 1964 (!) (32 Bit: IBM System 360, später 370 (1970) und 390 (1990), jetzt 64 Bit: System z (2000)). Sie hat nach wie vor große Bedeutung als Server in Rechenzentren von Banken, Versicherungen, öffentlicher Verwaltung usw..

- ***Sun Sparc:*** So wie MIPS und PowerPC stammt diese 32-/64-Bit-Architektur von technisch-wissenschaftlichen Rechnern aus den 80er/90er-Jahren ab. Sie wird heute nur noch in großen Servern von Sun (jetzt Oracle) und Fujitsu eingesetzt. Weiters gibt es eine Variante, die besonders für die effiziente Verarbeitung vieler gleichzeitiger Requests ausgelegt ist und daher besonders in Webservern, Fileservern usw. verwendet wird.

Daneben gibt es noch rund 10 verschiedene CPU-Designs für Mikrocontroller (z.B. ATMEL) und dutzende "ausgestorbene" CPU-Architekturen.

Arbeitsweise moderner CPU's

Wäre eine CPU heute noch so aufgebaut wie oben beschrieben, wäre sie rund 100 Mal langsamer als aktuelle CPU's.

Was macht heutige CPU's schnell?

Multicore

Ein moderner CPU-Chip enthält heute bis zu 6 (x86) bzw. bis zu 32 (andere Architekturen) voneinander unabhängige, eigenständige, gleichzeitig arbeitende Prozessoren, die sich Speicher- und I/O-Interface sowie L3-Cache teilen.

Mehrere Cores bringen aber nur dann etwas, wenn auch mehrere Programme gleichzeitig ausgeführt werden oder ein Programm intern mehrere parallele Abläufe startet!

Caches

Die Geschwindigkeit des Speichers hat mit der Entwicklung der Prozessoren nicht Schritt gehalten: War beides 1975 noch gleich (1 Speicherzugriff dauerte einen CPU-Takt, damals 1-2 MHz, d.h. 500 – 1000 ns), so liegt heute ein Faktor 200 – 1000 dazwischen: Ein CPU-Takt (1 Rechenoperation) dauert heute 0,3 ns (bei 3,3 GHz), ein Speicherzugriff je nach Systemarchitektur und Art des Zugriffes zwischen 60 und 300 ns (ein Speichertakt von 1600 MHz bei modernen DRAM's sagt nur, dass die sequentiell nächsten Daten *nach dem ersten Zugriff* mit diesem Takt nachgeschoben werden). Die CPU würde also weniger als ein Hundertstel ihrer Zeit rechnen und den Rest der Zeit nur warten.

Verbessert wird das durch Caches, die zuletzt gelesene oder geschriebene Daten zwischenspeichern, in der Erwartung, dass darauf vermutlich bald wieder zugegriffen wird. Weiters werden die "Nachbarn" zuletzt gelesener Daten "auf Verdacht" in den Cache geladen, mit demselben Hintergedanken: Jeder Speicher-Lese-Zugriff bei x86-Prozessoren lädt immer gleich 64 oder 128 Bytes in den Cache, auch wenn die CPU eigentlich nur 1 Byte haben will.

Beim Schreiben wird zuerst einmal nur in den Cache geschrieben, und die CPU kann weitermachen. Erst wenn der Cache den Platz für andere Daten braucht, schreibt er sie in den Speicher.

Das geschieht, ohne dass die Software irgendetwas davon merkt oder sich darum kümmern muss: Der Cache schaut bei jedem Speicherzugriff mit, und wenn er die angeforderten Daten selbst hat, liefert er sie, anstatt sie aus dem Speicher zu holen. Die Software kann allerdings die Wirksamkeit des Caches unterstützen, indem sie möglichst lokal und sequentiell auf möglichst wenige Daten zugreift und nicht wild durcheinander auf riesige Speicherbereiche.

Typische Zugriffs-Geschwindigkeiten und Größen:
L1-Cache (16-64 KB): 1-3 Takte
L2-Cache (64 KB – 4 MB): 10-20 Takte
L3-Cache (4 – 16 MB): 25-50 Takte
Hauptspeicher (1 – 16 GB): 200 – 1000 Takte
Meist gibt es für Befehle und Daten getrennte L1-Caches,
damit gleichzeitig neue Befehle geholt und auf Daten zugegriffen werden kann.

Die Caches sind heute die flächen- und transistorzahlmäßig größten Einheiten auf CPU-Chips.

Pipelining

Moderne CPU's machen nicht alle Schritte eines Befehls fertig, bevor sie mit dem nächsten Befehl beginnen: Während der erste Befehl seine Ergebnisse speichert, macht der nächste schon seine Berechnung, der darauffolgende holt seine Daten, der danach berechnet die Adressen seiner Operanden, der nächste wird dekodiert, und der übernächste inzwischen aus dem Cache geholt (bzw. gleich ein paar Befehle auf Vorrat).

Typische Prozessoren haben heute 10-20 Pipeline-Stufen und ebensoviele Befehle gleichzeitig in Bearbeitung, der Pentium 4 hatte sogar über 30 (nach dem Motto: Je mehr Stufen, umso weniger hat jede Stufe zu tun, umso einfacher ist sie, und umso höher kann der Takt sein). Die Befehle "rutschen" sozusagen bei jedem Takt eine Verarbeitungsstufe weiter.

So wird jeden Takt ein Befehl fertig, auch wenn die "Durchlaufzeit" eines Befehls in Wirklichkeit über 20 Takte dauert.

Sprungvorhersage

Pipelining hat ein Problem mit bedingten Sprüngen (die von den Ergebnissen irgendwelcher Berechungen abhängen): Die Pipeline "weiß" nicht, welchen der beiden Zweige sie weiterverfolgen soll (den mit oder den ohne Sprung), denn bei einem Sprung geht es ja ganz woanders im Programm mit ganz anderen Befehlen weiter. Dummerweise entsteht das Rechenergebnis, von dem abhängt, ob gesprungen wird oder nicht, erst sehr weit hinten in der Pipeline.

Es gibt 2 Möglichkeiten: Entweder, man schiebt keine neuen Befehle in die Pipeline, bis feststeht, ob der Sprung gemacht wird oder nicht (in jedem Fall › 10 untätige Wartezyklen), oder die Pipeline arbeitet "auf Verdacht" an einem der beiden Zweige weiter und macht die laufenden Befehle wieder rückgängig, wenn sie falsch geraten hat (je nach Sprung oder nicht Sprung keine oder noch mehr Wartezeiten).

Die Sprungvorhersage merkt unter anderem, wohin die letzten paar hundert Sprünge gingen, und versucht dadurch (unter der Annahme, dass derselbe bedingte Sprung auch wieder dasselbe Ergebnis haben wird), so gut wie möglich den "richtigen" Zweig vorherzusagen und dadurch die Wartezeiten zu minimieren, die durch das Ausleeren der Pipelines bei falschen Sprüngen entstehen.

Trotzdem sind Sprünge (if's, Schleifen) in Relation zu Rechenbefehlen heute eher "teure" (langsame) Operationen.

Mehrere Rechenwerke

Moderne CPU's haben pro Core 2-6 Rechenwerke, die gleichzeitig arbeiten können. Intels aktuelle CPU's können z.B. in einem einzigen Takt bis zu 4 Integer-Operationen und eine Gleitkomma- bzw. SSE-Operation ausführen. Das Leitwerk durchsucht daher aufeinanderfolgende Befehle nach solchen, deren Operanden nicht voneinander abhängen, und die daher unabhängig voneinander und gleichzeitig berechnet werden können. Das nennt sich "Multi-Issue" (Dual-Issue bei 2 gleichzeitigen Befehlen, Quad-Issue bei 4 usw.).

Dabei kann das Leitwerk auch Befehle umordnen ("Out-Of-Order Execution", "Instruction Reordering") und sogar Befehle "auf Verdacht" ausführen (und ev. wieder rückgängig machen), bei denen noch nicht sicher ist, ob sie gebraucht werden oder nicht ("Speculative Execution").

Insgesamt hat ein moderner x86-Prozessor rund 100 Befehle gleichzeitig "in Arbeit" bzw. "im Auge", um seine Hardware optimal auszulasten.

"Hyperthreading"

["Hyperthreading" ist Intel-Terminologie] Auch diese Erfindung hat ihre Ursache im langsamen Speicher: Hier hat ein Core zwar nur ein Leitwerk, ein Rechenwerk, usw., aber zwei (bei Sun Sparc und MIPS: Bis zu 8) komplette Registersätze und Program Counter. Er enthält daher den aktuellen Zustand von 2 (bzw. 8) Programmen und kann wahlweise bzw. abwechselnd Befehle des einen oder des anderen Programms ausführen.

- Bei Intel wird das dazu genutzt, den Core mit dem anderen Programm sinnvoll zu beschäftigen, wenn das eine Programm mehrere Warte-Takte (wegen eines Speicherzugriffes oder einer langen Gleitkomma-Operation) hätte.
- Bei Sun und MIPS wird mehr oder weniger starr reihum von allen 4 oder 8 Programmen jeweils 1 Befehl ausgeführt. Das hat den Vorteil, dass ein paar Takte vergehen, bis wieder ein Befehl desselben Programms drankommt. Das "versteckt" nicht nur Cache- und Speicher-Zugriffszeiten in der Rechenzeit anderer Programme: Die CPU hat auch mehr Zeit, Abhängigkeiten zwischen Befehlen desselben Programms zu behandeln (Datenabhängigkeiten, bedingte Sprünge usw.).

Das einzelne Programm gewinnt dadurch nichts (eher im Gegenteil), aber die Summe des Durchsatzes (über alle Programme betrachtet) steigt.

Multimedia-Erweiterungen

Fast alle Prozessor-Familien (x86, ARM, MIPS, PowerPC) haben heute Multimedia-Register

und Multimedia-Befehle(x86: Multi Media Extension MMX / Streaming Single Instruction / Multiple Data Extension SSE): Diese Register haben Platz für bis zu 256 Bits, also z.B. für 4 `double`-Gleitkomma-Zahlen oder 8 normale `int`'s oder 32 einzelne Bytes. Sie können mit einem Befehl geladen und gespeichert werden, es können auch mit einem Befehl entsprechend viele Gleitkomma- oder Integer-Berechnungen zugleich gemacht werden.

Anwendungen findet das z.B. bei Rechnungen mit grafischen Daten (3 Farb-Komponenten oder aufeinanderfolgende Pixel), geometrischen Werten (3 Koordinaten), komplexen Zahlen (Real- und Imaginärteil) oder bei Verschlüsselungs-Algorithmen (Bitoperationen auf viele Bytes gleichzeitig).

Großrechner für technisch-wissenschaftliche Anwendungen

In Größtrechnern für den technisch-wissenschaftlichen Bereich werden heute 3 Arten von Systemen eingesetzt:

- Systeme mit ***vielen*** (mehrere 10000 bis 100000) ***"normalen" Prozessoren*** (x86 oder PowerPC). Jeder Prozessor hat seinen eigenen Speicher und arbeitet relativ autonom (wie ein abgespeckter PC). Die "Kunst" dieser Systeme liegt in der Kommunikationstechnik zwischen den einzelnen Prozessoren: Hier kommen entweder viele Ethernet-Verbindungen oder eigene Netzwerk-Technologien (Infiniband, Hyperchannel, ...) zum Einsatz.
- Systeme wie oben, in denen jeder Prozessor zusätzlich über modifizierte ***Grafikkarten*** zur Gleitkomma-Berechnung verfügt. Grafikkarten haben theoretisch sehr hohe Gleitkomma-Rechenleistung (hunderte parallel arbeitende Rechenwerke!) und viel höheren Speicherdurchsatz als normale CPU's, sind aber nur schwer für allgemeine mathematische Berechnungen zu nutzen.
- ***Vektorrechner:*** Das ist die "klassische" Architektur wissenschaftlicher Supercomputer seit der Cray-1 (1976). Hier arbeiten nicht zehntausende normale, über Netz verbundene CPU's, sondern wenige bis dutzende hochspezialisierte CPU's auf einem zentralen Speicher: Sowohl die Hardware als auch der Befehlssatz sind für Gleitkomma-Berechnungen auf langen Vektoren und Matrizen optimiert: Jede CPU kann typischerweise in jedem einzelnen Takt 2-3 Matrix-Elemente aus dem Speicher holen, multiplizieren, addieren und in den Speicher zurückschreiben (auch für Daten, die so groß sind, dass sie im Cache nicht Platz haben). Ermöglicht wird das durch umfangreiches Pipelining nicht nur für die Befehlsabarbeitung, sondern auch in den Rechenwerken und Speicherinterfaces. Heute werden derartige Rechner vor allem von japanischen Firmen gebaut: NEC ("Earth Simulator"), Fujitsu, Hitachi.

Halbleiter-Speichertechnik

Halbleiter-Speicher dienen in einem PC

- als **Hauptspeicher** (RAM) für das Betriebssystem und gerade bearbeitete Daten und Programme (mehrere GB),
- als **Grafikspeicher** für Bildinhalte, 3D-Daten, Fonts, Texturen usw. auf der Grafikkarte (technologisch ident zum Hauptspeicher-RAM) (einige MB – 1 GB),
- als Speicher (ROM) für das **BIOS** und die **Firmware**, sowohl am Motherboard als auch auf Grafik-Karten, Controllern, Platten usw. (wenige MB)

- als Speicher für die **BIOS-Einstellungen** (CMOS-RAM) (typ. 256 Bytes),
- als **Cache-Speicher** auf der CPU und in Platten, CD-Laufwerken usw. (einige MB),
- und als **Festplatten-Ersatz** (SD-Karte, SSD, USB-Stick, ...) (viele GB).

Grundsätzliche Funktionsweise:

- Daten werden **byteweise** (oder maximal in Einheiten von 4 oder 8 Bytes = 64 Bits) gespeichert.
- Jedes Byte (bzw. jeder 8er-Block) hat eine **Adresse** ("Hausnummer", eine fortlaufende Zahl), über die es direkt und einzeln angesprochen werden kann. Zu jedem Speicherbaustein führen daher **Daten- und Adressleitungen**.
- **Direkter Zugriff** auf jedes einzelne Byte ("direkt adressierbar"): Der Zugriff auf alle Bytes ist prinzipiell gleich schnell (in der Praxis nicht ganz: Je nach Speichertechnik ist ein kleiner Unterschied, ob aufeinanderfolgende Bytes, Bytes in einem kürzlich benutzten 4-KB-Block, oder andere Bytes angesprochen werden).
- Intern bestehen die Chips aus einer rechteckigen Matrix aus Speicherzellen: Die Adresse wird auf eine Zeilen- und eine Spaltennummer abgebildet.

Einteilung:

- **RAM ("Random Access Memory"):**
  - Gleichschnell **schreib- und lesbar**.
  - Beliebig oft schreibbar, byteweise schreibbar, altert nicht.
  - **Verliert Daten bei Stromabschaltung.**
- **ROM ("Read-Only Memory"):**
  - Nicht oder nur **sehr langsam schreibbar**
    (> 1000 Mal langsamer schreib- als lesbar).
  - Meist nur in großen Blöcken lösch- oder schreibbar (z.B. 256 KB), nicht byteweise.
  - **"altert" beim Schreiben**
    (d.h. überlebt nur eine bestimmte Anzahl von Schreibvorgängen).
  - **Behält Daten auch ohne Strom.**

Unterteilung von RAM:

- **Dynamisches RAM ("DRAM"):**
  - Speicherung durch einen ***Kondensator*** (Halbleiter, nicht metallisch) pro Bit
    (+ 1 Transistor pro Zelle für den Zugriff)
    ==> Klein und billig
    ==> Kondensator-Ladung muss periodisch nachgeladen werden
    (alle paar ms: "Refresh", macht der Speicherkontroller, kostet Strom!)
  - Größe: Derzeit 4 Gbit pro Chip (4GB pro Speicherriegel mit 8 Chips)
  - Zugriffszeit:
    Über 30 ns für Erstzugriff auf beliebiges Byte.
    Rund 5-10 ns für Bytes im selben 4-BK-Block.
    Sequentiell folgende Bytes <= 1 ns.
    (Grund: Intern wird immer eine ganze 4-KB-Zeile aus den Zellen geholt)
    ==> Sequentielle Datenrate bei 64 Bit breiten Speicherriegeln rund 10 GB/sec, weitere Verdopplung durch 2 oder 4 Speicherbänke.
  - Einsatz für *Hauptspeicher, Grafikkarten, Disk-Cache*, teilweise L3-Cache,...

    - Aktuelle Bezeichnungen von Speicherriegeln ("DDR = Dual Datarate DRAM", ...) definieren nur die Anschluss-Technik (Datenübertragung auf beiden Taktflanken), nicht die Speichertechnik an sich!
- **Statisches RAM ("SRAM"):**
    - Speicherung durch ***Flipflop*** (==› 6 – 8 Transistoren pro Zelle)
      ==› Weniger Speicherkapazität (einige MB/Chip) & viel teurer als DRAM
      ==› Hält die Daten ohne Refresh, braucht keinen Speicherkontroller, ist wirklich inaktiv, wenn nicht gelesen / geschrieben wird
    - Wahlweise *viel schneller* als DRAM (Zugriffszeit bis <= 1 ns!)
      ==› für CPU-Caches usw.
      oder *viel stromsparender* (kein Refresh ==› nur wenige µA wenn kein Zugriff)
      ==› jahrelang mit Knopfzelle versorgbar,
      z.B. CMOS-RAM für BIOS-Einstellungen
      (aber: SRAM braucht unter Volllast mehr Strom als DRAM)

Historische Entwicklung von ROM:

- **ROM**: Maskenprogrammiert bei der Herstellung, nicht änderbar.
- **PROM**: Einmal schreibbar, nicht löschbar (Schreiben = Sicherungen durchbrennen)
- **EPROM**: Mehrmals (einige 100 Mal) schreibbar: Löschbar als Ganzes mit UV-Licht.
- **EEPROM**: Wie EPROM, aber löschbar durch hohe Spannung.
- **Flash**:
    - In großen Blöcken elektrisch mit Normalspannung löschbar und dann byteweise schreibbar.
    - Begrenzte Schreib-Lebensdauer!
    - Löschen / schreiben dauert rund 1000 Mal länger als Lesen.
    - Auch Lesen langsamer als RAM:
      Rund 50-100 ns Zugriffszeit, rund 20 MB/sec pro Chip.
    - 2 grundsätzliche Technologien:
      NAND-Flash:
      Billiger, mehr Speicherplatz (64 Gbit / Chip), aber langsamer und viel komplizierter anzusprechen (nur blockweiser Zugriff, Fertigungsfehler, ...)
      ==› für Massenspeicher (USB-Stick, SSD, ...)
      NOR-Flash:
      Teurer, weniger Speicherplatz pro Chip, aber "ganz normal" adressierbar und fehlerfrei ==› für BIOS und Firmware
    - 2 Varianten:
      SLC ("Single Level Cell"):
      1 Bit pro Zelle ==› weniger Speicherplatz, teurer, aber schneller und langlebiger (100000 Schreibzyklen) ==› Industrieprodukte
      MLC ("Multi Level Cell"):
      2-4 Bit pro Zelle ==› mehr Speicherplatz, billiger, aber langsamer und kurzlebiger (10000 Schreibzyklen) ==› Consumer-Produkte

Zukünftige Entwicklungen:

- **Ziele:**
    - Wie RAM byteweise und gleichschnell schreibbar und lesbar
    - Beliebig oft schreibbar
    - Behält Daten auch ohne Strom

- **Technologien:**
  - Magnetische Effekte (MRAM, FeRAM, ...)
  - Materialstruktur-Effekte (Phase Change RAM, nutzt Unterschied zwischen kristallinem und amorphem Festzustand eines Materials)

Fehlerkorrektur:

- RAM hat mit gewisser Wahrscheinlichkeit "flüchtige" Fehler (Strahlung, ...), aber sollte keine harten Fehler haben
  ==› wie besprochen: Parity, Hamming-ECC
- Flash (vor allem NAND-Flash) hat flüchtige und harte Fehler:
  ==› ECC plus teilweise blockweise Prüfsummen wie auf Platte
  ==› Aufwändige Fehlerverwaltung: Reserve-Blöcke, ...

Grafikkarte und Bildschirm

Grundlegendes

Im Computer wird die Anzeige in einzelne **Pixel** zerlegt: Beim VGA-Schirm des Ur-PC waren es 640*480 Pixel, später 800*600 bzw. 1024*768, dann war lange Jahre 1280*1024 Standard (sowie 1600*1200 bei großen Schirmen), also Seitenverhältnisse von 4:3 oder 5:4. Heute sind es meist schon 1920*1080 (oder 1600*900, beides Verhältnis 16:9), dazwischen waren einige Jahre 16:10-Schirme üblich (1920*1200, 1680*1050 oder 1440*900). Die "Übergrößen" sind heute 2560*1440 (16:9) bzw. 2560*1600 (16:10).

Für jedes Pixel wird bei Computerbildschirmen e**in Rotwert, ein Grünwert und ein Blauwert** gespeichert (**RGB-Darstellung**). Für jeden einzelnen Farbwert stehen heute normalerweise 8 Bit zur Verfügung, d.h. Werte zwischen 0 und 255, das ergibt 2^{8*3} Kombinationsmöglichkeiten bzw. rund 16 Mio. verschiedene Farben. Üblicherweise speichert man diese 24 Bit pro Pixel in 4 Bytes (32 Bits), um sich "krumme" Adressen und Rechnungen zu ersparen, 1 Byte pro Pixel bleibt dabei unbenutzt (oder speichert den Alpha-Wert, d.h. die Transparenz der Farbe bei Überlagerungen). Das ergibt für einen 1920*1080-Schirm ziemlich genau 8 MB Bilddaten.

Kameras und medizinische Geräte haben höhere Farb-Auflösung (12 oder 16 Bit pro Farbwert, d.h. je 4096 oder 65536 Abstufungen für rot, grün und blau), früher waren es wesentlich weniger: Die Ur-VGA-Karte hatte insgesamt 4 Bit pro Pixel für 16 vordefinierte Farben, später waren es 8 Bit bzw. 256 Farben, die als Index in eine (an sich vordefinierte, aber umprogrammierbare) Farbtabelle interpretiert wurden, schließlich 16 Bit pro Pixel (je 5 für rot, grün und blau, eines leer).

In anderen Bereichen sind andere Farbmodelle üblich:

- **Drucker** arbeiten mit Farbwerten für cyan, magenta und gelb und einem Schwarzwert (**CMYK-Darstellung**).
- Im **Fernsehen** bzw. bei **Videos** ist die sogenannte **YUV-Darstellung** üblich: Y ist der Helligkeitswert, U und V sind Farbwerte (Koordinaten in einer zweidimensionalen Farb-Fläche).
- Weiters gibt es noch Systeme mit einem Farbwert, einer Farb-Sättigung und einem

Helligkeitswert.

Die Grafikkarte

Eine Grafikkarte besteht aus 2 Hauptkomponenten:

- Dem **Grafik-Chip**, der die Logik enthält und die Bildsignale generiert.
- Dem **Grafik-Speicher** für die Bilddaten ("Frame-Buffer").

Ursprünglich enthielt die VGA-Grafikkarte ("Video Graphics Array") nur den Bildspeicher (beim PC ab Adresse Hex A0000) wie oben beschrieben, sie hat ihn 60 Mal pro Sekunde komplett Pixel für Pixel gelesen (zeilenweise) und daraus die analogen rot-, grün- und blau-Signale für die Bildröhre generiert (siehe unten).

Daneben kannte die klassische VGA-Karte auch einen **Textmode**: Das Programm hat nur die anzuzeigenden Buchstaben-Codes (ASCII) in den Grafik-Speicher geschrieben (1 Byte pro Zeichen, d.h. 2 KB für einen Schirm mit 25 Zeilen zu 80 Spalten), und die Grafikkarte hatte einen fix einprogrammierten Font und generierte daraus bei jedem Bilddurchlauf die richtigen Pixel für die angegebenen Buchstaben.

Im nächsten Schritt kam **2D-Beschleunigung** dazu: Die Grafikkarte konnte im Bildspeicher selbständig Bildinhalte verschieben, Rechtecke füllen oder löschen, Bildinhalte überlagern usw. ("Bit Blit"), und sie konnte selbst an der gewünschten Stelle einen Cursor ins Bild einblenden ("Hardware-Cursor").

Dann kamen Karten mit **3D-Beschleunigung**, die grundsätzlich aus 2 Schritten besteht:

- **Koordinatentransformation (Geometrie-Berechnung)**: Dieser Schritt berechnet aus den dreidimensionalen Koordinaten der darzustellenden Objekte (bzw. ihrer aus Dreiecksflächen modellierten Oberfläche) deren Position auf der zweidimensionalen Bildfläche (incl. Perspektivenberechnung).
  Dabei wird das Clipping (Abschneiden außerhalb des sichtbaren Bereiches liegender Objektteile) und die Verdeckungsberechnung (Entfernen rückseitiger oder durch andere Objekte verdeckter Flächen) durchgeführt.
- **Texturing (Farb-Berechnung)**: Dieser Schritt füllt die so transformierten Dreiecke mit Farbe, indem die Texturen entsprechend skaliert und perspektivisch verzerrt auf die Oberflächen projeziert werden.
  Weiters werden je nach Winkel und Form der Oberfläche deren Helligkeit und andere Beleuchtungseffekte berechnet. und die Alpha-Werte berücksichtigt.
- Heute sind 3D-Einheiten **universell programmierbar** und können Aufgaben übernehmen, die über die klassische 3D-Anzeige hinausgehen, z.B. **Physik-Berechnungen** oder die Berechnung von Partikelwolken.

Diese Berechnungen sind aufwändig, aber gut auf viele gleichzeitig rechnende Einheiten zu verteilen. Moderne Grafikkarten haben daher bis zu 500 gleichzeitig arbeitende 3D-Einheiten, um tatsächlich zumindest 30 Mal in der Sekunde den kompletten Bildinhalt (oft über 1 Million 3D-Dreiecke) neu berechnen zu können. Moderne Grafikkarten enthalten so wie CPU's über 1 Milliarde Transistoren und laufen mit Takten im GHz-Bereich.

Weiters wurde die **Video-Beschleunigung** in Hardware implementiert, mit folgenden Einheiten:

- **MPEG-Dekoder**: Diese Einheit übernimmt einen Teil der zum Dekodieren von MPEG-Videos oder DVD's notwendigen Rechenoperationen und verschönert Bewegungs- und Komprimierungseffekte.
- **Scaler**: Diese Einheit transformiert Videobilder auf die gewünschte Auflösung bzw. Größe und versucht, dabei entstehende Unschönheiten zu glätten.
- **Farbraum-Konvertierer**: Diese Einheit wandelt die in den Videosignalen enthaltene YUV-Farbinformation in RGB-Werte um.
- **Video Overlay**: Diese Einheit legt die von einer Videoquelle (Fernsehempfänger-Karte, Kamera, ...) oder der Video-Dekodierung gelieferten Bilddaten ohne Zutun der CPU in Echtzeit in einem Ausschnitt des Bildspeichers ab und überlagert ev. Schriften usw..

Bei 3D-Karten ist der Grafikspeicher mehrere hundert MB groß und enthält viel mehr als nur die Bildpunkte:

- Meist 2 Kopien der Bildpixel (eine zum Anzeigen, eine zum Neuberechnen), zwischen denen bei jedem Bild hin- und hergeschaltet wird ("Doppelpufferung").
- Die **3D-Modelle** der Objekte und deren auf 2D transformierten Positionen.
- **Texturen**.
- **Fonts** (d.h. die Pixelmuster der Buchstaben).
- Den **Programmcode** für die 3D-Einheiten.

Der Grafikspeicher ist der Speicher mit der höchsten Durchsatzrate im System, da er ja einige hundert 3D-Einheiten mit Daten versorgen muss: Möglich sind heute Takte bis 2 GHz, das ergibt bei 512 Bit breitem Speicherbus über 100 GB/s maximale Datenrate. Bei in den Prozessor oder in die Northbridge integrierten Grafik-Karten, die ihre Daten im "normalen" RAM haben, ist daher mit deutlichen Leistungseinschränkungen sowohl für die Grafik als auch für die CPU zu rechnen, weil der Speicher den Engpass darstellt.

Die Verbindung zum Bildschirm

- Sie erfolgte ursprünglich beim "**VGA-Anschluss**" (15-polig in 3 Reihen) **analog**, wie beim Fernsehschirm: Je eine Kabelader für das rote, grüne und blaue Bildsignal, dazu noch Adern für die Synchronisation (Zeilensprung und Bildwechsel: Hsync und Vsync).
- Heute werden die Pixeldaten **digital** und **seriell** (als Bits, teils verschlüsselt) zum Bildschirm geschickt, da die Verwandlung auf analog und zurück bei Flachbildschirmen Qualitätsverluste bringt.
  Die erste derartige Verbindung hieß **DVI** (Digital Video Interface, etwas größer als VGA mit 18-24 kleinen und 1-5 großen Flachstiften), es folgten **HDMI** (High Definition Multimedia Interface, auch mit Tonkanal) und **DisplayPort**.
  Bei DVI gibt es eine reine Digital-Variante (DVI-D, nur 1 großer Kontakt) und eine kombinierte Variante, bei der auch noch die analogen VGA-Signale mitgeführt werden (DVI-I, 5 große Kontakte).
  Diese Verbindungstechniken sind sehr anspruchsvoll: Bei 60 Hz full HD sind immerhin 360 MB/s zu übertragen. Deshalb ist über 1920*1200 bei DVI ein zweiter Kanal erforderlich (Dual-Link-DVI, 24 statt 18 Pins).
- Alle Verbindungen haben daneben noch einen langsamen seriellen Rück-Datenkanal, auf dem der Bildschirm seine Identifikation, Auflösung usw. zum Computer schickt (**DDC** bzw. **EDID**).

Der Bildschirm

Heute haben 4 Techniken Bedeutung:

- Die klassische **Bildröhre**: Drei durch Hochspannung erzeugte Elektronenstrahlen (einer pro Farbe) werden horizontal und vertikal durch elektrische und magnetische Felder abgelenkt und bringen fluoreszierende Substanzen punktuell zum Leuchten ("schreiben" also das Bild Pixel für Pixel, Zeile für Zeile, 60 Mal pro Sekunde).
- Die **LCD-Flachbildschirme** (Liquid Crystal Display): Sie bestehen aus
    - der weißen Hintergrund-Beleuchtung (mittels Leuchtstoffröhren oder Leuchtdioden),
    - einer Lage Polarisations- und Farbfiltern,
    - den eigentlichen Flüssigkristallen (einem pro Farb-Pixel) samt Ansteuer-Elektronik,
    - und noch einer Lage Polarisations- und Farbfiltern.

  Je nach angelegter Spannung ändern die Flüssig-Kristalle die Polarisationsrichtung des durchgehenden Lichtes, und je nach Änderung der Polrichtung kann das vom ersten Filter polarisierte Licht den zweiten Filter passieren (Pixel leuchtet) oder nicht (Pixel dunkel).
- Den **OLED-Displays** (Organic Light Emitting Diode): Hier besteht jedes Pixel aus drei Leuchtdioden für die Grundfarben, leuchtet also von von sich aus in der gewünschten Farbe. Diese Leuchtdioden bestehen allerdings nicht aus Halbleiter-Material, sondern aus organischen Substanzen.
- **Plasma-Schirme**: Hier besteht jedes Pixel aus 3 kleinen gasgefüllten Kammern (eine pro Grundfarbe). Das Gas gibt durch Hochspannungs-Gasentladung UV-Licht ab, dieses wird wie bei der Fernseh-Röhre durch fluoreszierende Substanzen in sichtbares farbiges Licht verwandelt.

Drucker

- ***Verbindung*** mit dem Computer: Früher mittels "**paralleler Schnittstelle**" ("LPT", 25-poliger Stecker), heute mittels **USB** oder **Ethernet**.
- ***Ansteuerung***: Entweder mit **proprietärem Treiber**, der die Bildpunkte im Computer berechnet und die fertigen Pixeldaten in Hersteller-spezifischem Format zum Drucker schickt ("**GDI-Drucker**", **Host Based Printer**, meist nur unter *Windows* nutzbar), oder in einer **standardisierten Druckersprache**: Im professionellen Bereich **Postscript**, im Home-Bereich auch **PCL**. Zu Postscript- und PCL-Druckern kann man ganz normale **ASCII-Textdateien** auch direkt schicken. PCL ist ein Satz von Steuerzeichen (Escape-Sequenzen), Postscript ist eine vollwertige Programmiersprache, die im Drucker (relativ aufwändig) seitenweise interpretiert wird.
- ***Drucktechnik***:
    - **Tinte:** Die Tinte wird tröpfchenweise auf das Papier gespritzt, entweder mittels **Piezo-Kristall-Pumpen** (Epson) oder mittels **Hitze / Verdampfung** (praktisch alle anderen).
    - **Laser:** Der Druckvorgang ist folgender:
        - Die "**Fototrommel**" (ein Band oder eine Trommel aus bei Licht

leitendem Halbleiter-Material) wird mit Hochspannung statisch aufgeladen.
        - Dann belichtet ein **Laser** (oder Leuchtdioden) das Druckbild auf die Trommel: Dort, wo der Laser trifft, wird der Halbleiter leitend und entlädt sich.
        - Anschließend läuft die Trommel durch das (ebenfalls elektrisch vorgeladene) **Toner**-Pulver: An den entladenen Stellen der Trommel bleibt der Toner haften, die geladenen Stellen stoßen ihn ab.
        - Jetzt bewegt sich die Trommel zum Papier, der Toner wird auf das Papier übertragen (auch hier hilft eine elektrische Ladung des Papiers).
        - Danach wird der Toner bei knapp 200 Grad thermisch am Papier **fixiert** (eingeschmolzen).
        - Die Trommel wird elektrisch neutralisiert und von Toner-Resten gereinigt, bevor sie wieder an der Ladestation vorbeiläuft.
    - **Mechanisch: Farbband** und Anschlag mit Punkte-Matrix (elektromagnetisch).
    - **Thermisch:** Entweder mittels **wärme-empfindlichem Papier** (bei Kassen usw.) oder mittels **Thermosublimationsdruck** (Farbwachs wird pixelweise geschmolzen / verdampft, bei Foto-Druckern).

Motherboard und PC als Ganzes

Das Gehäuse

- schirmt den Computer gegen **elektromagnetische Störstrahlung** von außen ab,
- schirmt die im Computer entstehende Störstrahlung (reichlich!) ab (daher die "Funk-Dicht-Leisten"),
- und dient neben der Befestigung auch der Kühlluft-Führung.

Standard-Formate für Motherboard und Gehäuse:

- ATX
- Mini-ATX (um 4 Slots gekürztes ATX)
- ITX (für ganz kleine Computer: 17*17 cm)

Schnittstellen extern:

- Alte serielle und parallele Schnittstelle (Modem- und Druckerschnittstelle), alte PS/2-Schnittstelle (Tastatur & Maus), alter Gameport
- USB, Firewire, eSATA
- VGA, DVI, HDMI, Display-Port
- Sound (digital & analog)
- Ethernet

Schnittstellen intern:

- Stromversorgung (2 Mal)
- Steckkarten (PCI-Express *1 / *16, PCI, ISA)
- RAM-Slots
- SATA, PATA=IDE, ev. noch alte Floppy-Schnittstelle
- Pfostenstecker für weitere USB-Ports usw., für vordere Audio-Anschlüsse
- Pfostenstecker für Power- und Disk-LED's, Power, Suspend, Reset, alten Piepser

- ev. Analog-Sound-Anschluss für CD-Laufwerk
- Lüfter-Anschlüsse
- Batterie für CMOS-RAM (BIOS-Einstellungen) und Uhr
- Jumper: Meist: CMOS löschen
- Bei Notebooks:
  Steckkarten: PCMCIA, Cardbus, Expresscard
  Card-Reader: CF-Card, SD-Card

Das Netzteil:

- +12 V: Laufwerke, Spannungsregler für CPU & Grafik-Chip
- +5 V, +3.3 V: Großteil der Elektronik und Schnittstellen
- -5 V, -12 V: Seltene Schnittstellen, heute kaum noch verwendet
- +5V Standby: Zum Start via Taste, Netz, ...
  ==› Wenn Computer wirklich stromlos sein soll:
  Netzkabel ziehen oder am Netzteil selbst ausschalten!!!
- Die internen Spannungen sinken immer weiter:
  CPU knapp über 1 V, DDR3-RAM 1.5 V regulär, ...
- Bei Servern: redundante Netzteile

Platten, optische Laufwerke, Bandlaufwerke, ...

Platten (HDD = Hard Disk Drive)

Geschichte:

- Erste Platte: IBM 1956, 60 cm Durchmesser, 1 Tonne Gewicht, 5 MB
- Rund 1982: Erste Platten im 5,25-Zoll-Format für PC's: 5-10 MB

Arbeitsweise:

- 1-5 schnell rotierende (4200 – 15000 U/min) Platten mit magnetischer Oberfläche.
- Pro Oberfläche ein knapp (‹= 10 Nanometer!) über der Platte schwebender Schreib-Lese-Kopf. (Bei "Aufsetzern" ("Head Crash"): Totalschaden...)
  Kopf sitzt am Ende eines beweglichen Armes ==› auf beliebigen Radius verstellbar.
  "Parkzone" zum Aufsetzen des Kopfes bei Stillstand der Platte und bei freiem Fall:
  Ganz innen auf der Platte oder auf eigener Fläche neben den Platten.
- Daten in Blöcken fixer Größe (seit ›30 Jahren 512 Bytes, seit neuestem 4 KB)
  (oft auch "Sektoren" genannt):
  Es werden nicht einzelne Bytes, sondern immer ganze Blöcke gelesen/geschrieben.
  Jeder File belegt eine ganze Anzahl von Blöcken.
  Jeder Block hat eine Prüfsumme / Fehlerkorrektur.
- Blöcke in konzentrischen Kreisen ("Spuren").
  Die "Spur 0" ist ganz außen, außen ist schneller
  (längere Spur ==› mehr Daten pro Umdrehung)
- Datendichte: Derzeit rund 10000 Spuren pro Millimeter,
  rund 50000 Bits pro Millimeter in Längsrichtung ==› rund 2 TB pro Laufwerk.

Geschwindigkeit:

- *Zugriffszeit:* Wie lange braucht es, bis die Datenübertragung beginnt?
  - Mittlere Kopf-Positionier-Zeit: Rund 10 ms beim PC, 3,5 ms bei Servern.

    - Mittlere Rotations-Wartezeit: Rund 4 ms beim PC, 2 ms bei Servern.
- *Datenrate:* Bis zu 150 MB/s
  (nicht täuschen lassen von der Datenrate des Interfaces: Bis 600 MB/s)
- Cache (bis 64 MB RAM auf der Platte) zur Beschleunigung:
  Liest voraus & speichert zu schreibende Daten zwischen.

Interface:

- Im PC: Früher **PATA** bzw. **IDE**, jetzt **SATA**, bei externen Laufwerken über USB.
- Bei Servern: Früher **SCSI**, jetzt **SAS**, im High End **Fibre Channel** (Glasfaser)
- Was ist "ATAPI"?
  Das Software-Protokoll, um über PATA/SATA auch CD-Laufwerke, Bänder usw. anzusprechen ("SCSI-Befehle verpackt auf ATA-Leitung").

Adressierung:

- Früher, teilweise noch im BIOS: **CHS-Adressierung**: 3 getrennte Nummern:
    - Cylinder (welche Spur?)
    - Head (welcher Kopf, d.h. welche Platte?)
    - Sector (wievielter Block in der Spur?)

  Kann von Platten heute teilweise noch "vorgetäuscht" werden,
  entspricht aber nicht mehr der wirklichen Geometrie!
- Heute: **LBA-Adressierung** ("Logical Block Adressing"):
  Fortlaufende Nummerierung der Blöcke ab 0.

Grenzen der Adressierung ==› Probleme, Platten richtig anzusprechen:

- LBA im MBR kann nur max. 32 Bit Sektornummern
  ==› MBR nur für Platten bis 2 TB möglich (2^{32} * 512 = 2 TB)
  ==› Anderer Partitions- und Bootmechanismus ("EFI-GUID" = "GPT")
  oder 4 KB Sektoren (2^{32} * 4 KB = 16 TB) notwendig!
- "Altes" PATA konnte nur LBA mit max. 28 Bit Sektornummern
  ==› Ging nur für Platten bis max. 128 GB
  (Betroffen: Win XP vor SP2 !)
- Magische Grenzen der CHS-Adressierung: 504 MB, 8 GB, 32 GB

Inhalt:

- Ganz vorne (im 1. Sektor): **"Master Boot Record" (MBR)**:
  Partitionstabelle und Bootloader
  Wenn kaputt / überschrieben: Oh je…
- Dahinter: Ein oder mehrere **Partitionen**
  (ursprünglich 4 "primary partitions", heute + "extended partitions"):
    - Bekommen unter Windows je einen Laufwerksbuchstaben
      ("logisches Laufwerk")
    - Können verschiedenen Betriebssystemen gehören
    - Werden einzeln und unabhängig voneinander verwaltet:
      Jede für sich enthält ein Filesystem
    - Ev. auch: Partitionen *ohne* Filesystem, z.B. Unix/Linux "Swap Partition" (entspricht dem PAGEFILE.SYS in Windows), "Suspend"-Partitionen (zum Sichern des Hauptspeichers beim Schlafenlegen des Notebooks/PCs), versteckte Systemwiederherstellungs-Partitionen, "rohe" Datenbank- oder Backup-Partitionen, …

- Bei Servern: “Volume Manager”:
  Kann Partitionen über mehrere Platten hinweg, dynamisch ändern, …

Zur Partitionierung:
Geschieht meist automatisch bei der Betriebssystem-Installation.
Aber: Automatische Aufteilung (1 logisches Laufwerk) oft unklug!
Bei mehr als einem Betriebssystem: Auf jeden Fall selber machen!
Bei nur einem Betriebssystem: Besser auch selber machen:

- Mindestens 2 Partitionen: System und Benutzerdaten,
  ev. weitere z.B. für Datenbanken, …
  Kriterium oft auch: Je nach Backup-Strategie der Daten!
- Je nach zu erwartender Systembenutzung:
  Viele Benutzerdaten, wenig Software: Kleines C:, großes D:
  Viel Software, wenig Benutzerdaten: Umgekehrt!
- Ev. Platz auf Platte freilassen, um Partitionen bei Bedarf zu vergrößern
  (falls das Betriebssystem das kann).

Beim Partitionieren (DOS-Befehl `fdisk`):
Größte Vorsicht bei Platten, auf denen schon Daten sind!!!

SSD’s (Solid State Disks)

- Gehäuse, Interface, Zugriff, … wie auf Platte
- Intern: Halbleiter-Speicher (Flash, wie im USB-Stick)
- Vorteile:
  - Keine bewegten Teile:
    Mechanisch unempfindlicher, weniger Stromverbrauch, lautlos, …
  - Zugriffszeit praktisch 0 (< 0,1 Millisek), Lesen sehr schnell (> 250 MB/sec)
- Nachteile:
  - Schreiben viel langsamer als Lesen (in der Praxis: max. 100 MB/s auf Dauer),
    wird mit zunehmender Benutzung / zunehmendem Füllstand langsamer!
  - “Altert” beim Schreiben, jedes Bit lässt sich nur rund 10000 Mal schreiben.
  - Noch teuer.

Optische Laufwerke

Entwickelt aus der Audio-CD bzw. Video-DVD:

- Kapazität laut CD-Standard 650 MB = 74 Minuten Audio (max. 700 MB = 80 Min.,
  alles darüber verletzt den Standard, weil die Spuren enger als vorgeschrieben oder
  außerhalb des offiziellen Bereiches liegen).
  Bei DVD’s: 4,7 GB einlagig / 8,4 GB zweilagig
  Bei Blu Ray’s: 25 GB einlagig / 50 GB zweilagig
- Geschwindigkeit oft als x48, x12 usw. angegeben:
  CD: x1 ist die alte Audio-CD-Datenrate (150 KB/sec).
  DVD: x1 ist die normale Video-DVD-Datenrate (1,385 MB/sec).
  Blu Ray: x1 ist 4,5 MB/sec

Aufbau:

- Kunststoffträger
- Reflexionsschicht (Silber, Aluminium, bei speziellen Langzeit-Archiv-CD's Gold)
- Datenschicht:
    - Bei fertig bespielten CD's: Keine. Daten sind als Erhöhungen / Vertiefungen in die Reflexionsschicht geprägt. Farbe silber.
    - Bei einmal beschreibbaren CD's: Organischer Farbstoff, wird beim Beschreiben durch den Laser punktuell verbrannt / ausgebleicht.
      Farbe grün, blau, ...
      Achtung: Organische Farbstoffe "altern" stark
      (durch UV-Licht und Oxydation durch Luftsauerstoff).
    - Bei wiederbeschreibbaren CD's: Anorganisches Material (Metall-Legierung), ändert durch punktuelle Erhitzung (700 Grad!) seinen Erstarrungszustand zwischen kristallin und amorph ("phase change material") und damit seine optischen Eigenschaften. Ist stabiler als organische Farbstoffe, aber altert durch oftmaliges Löschen / Neuschreiben (einige 1000 Mal).

Beschreiben und Auslesen mittels Laser (optisch, berührungslos).

Unterschiede zur Platte:

- Nicht mehrere zylindrische Spuren, sondern eine durchgehende, spiralförmige.
- Wird von innen nach außen beschrieben.
- Sektorgröße 2 KB.
- Keine Partitionierung, aber Inhaltsverzeichnis am Anfang.
- Anderes Filesystem:
    - ISO 9660 und Erweiterungen zum Beschreiben "in einem Rutsch":
        - ISO 9660: Ursprünglich Filenamen nur 8+3, später 31 Zeichen, nur Großbuchstaben, keine Umlaute, max. 8 Verzeichnis-Ebenen, keine Verweise, ...
        - Joliet: Microsoft-proprietäre Erweiterung
        - Rock Ridge: Erweiterung für Linux / Unix.
    - UDF: Nachfolger, auch zum sektorenweisen Beschreiben (wie eine Platte).
- Anderer Boot-Mechanismus: "El Torito"
- Muss normalerweise komplett in einem Rutsch beschrieben werden.
  Ausnahmen:
    - "Multisession-CD's": Solange die CD nicht "abgeschlossen" ist, können hinten Daten dazugefügt werden.
    - "Packet Writing" mit UDF: Schreiben einzelner Sektoren / Dateien.

Bandlaufwerke

Zur Speicherung / zum Backup sehr großer Datenmengen
(zahlt sich erst ab vielen TB aus!)
Nur für sequentielle Aufzeichnung (Such-Zeiten im Minutenbereich!).

Im halbprofessionellen Bereich: DAT (Digital Audio Tape)
Schrägspuraufzeichnung mit schnell rotierender Kopftrommel und langsamem Band
(wie früher Video-Recorder)

Kassetten klein und relativ billig, aber unzuverlässiger als Linearaufzeichung.
Derzeit max. 160 GB pro Kassette / 12 MB/sec Geschwindigkeit (unkomprimiert).
Interface USB oder SAS.

Im professionellen Bereich: LTO (Linear Tape Open), früher DLT (Digital Linear Tape)
Lineare Aufzeichnung (Spuren in Längsrichtung): Kopf fix, sehr schnell bewegtes Band (wie klassisches Spulen-Magnetband)
Kassetten und Laufwerke teurer, aber robuster.
Meist in Kombination mit Band-Robotern.
Derzeit max. 1500 GB pro Kassette / 140 MB/sec Geschwindigkeit (unkomprimiert).
Interface SAS oder Fibre Channel.

Das Filesystem

... legt fest, wie die Dateien auf einer Partition / einer Floppy / einem USB-Stick gespeichert werden:

- Welche *Verwaltungsinformation* wird wo und wie gespeichert?
- Wie dürfen *Datei- und Verzeichnisnamen* aussehen (Länge, Zeichensatz)?
- Wo liegen die *Inhalte der Dateien* und wie findet man sie?
  Wie findet man freien Platz für neue Daten?
- Welche *Grenzen* bestehen
  - für die Größe des gesamten Dateisystems
  - für die Größe eines einzelnen Files
  - für die Länge von Filenamen und Verzeichnis-Pfaden
  - für die maximale Anzahl von Dateien und Verzeichnissen
  - ...

Bekannte Filesysteme:

- Universell (DOS, Windows, Linux, Mac, Kameras, Media-Player, Playstation, ...): **FAT**
  Ursprünglich FAT12 auf Floppies, dann FAT16 bis rund 1 GB, heute FAT32
  Heute noch auf USB-Sticks, CF- und SD-Cards, ...
  Nicht mehr auf Platten!
  Achtung: Original DOS kann kein FAT32!
- Windows: **NTFS**
- Unix / Linux: Dutzende verschiedene, Linux meist Ext2/3/4 oder XFS
- CD's: ISO 9660, UDF

FAT ("File Allocation Table"):

- Ohne VFAT nur Filenamen 8+3
  Erweiterung VFAT für lange Dateinamen (max. 255) + Dateinamen in Unicode.
- Nur minimale Dateiattribute, alle direkt im Verzeichniseintrag gespeichert:
  Read-Only, System, Hidden, Archive,
  Änderungsdatum (nur Tag), Filegröße
- Max. Filegröße 4 GB !
- Platzverwaltung in Einheiten von "Clustern" (0,5 bis 32 KB).
  Cluster klein: Viel Overhead, max. Filesystem-Größe 128 GB (2^{28} Cluster)

Cluster groß: Viel Verschnitt, max. Filesystem-Größe 8 TB
- File Allocation Table (2 mal vorhanden: Original + eine Sicherheitskopie):
  Zentrale Tabelle aller Cluster
  mit verketteter Liste von Clustern pro File
  und verketteter Liste freier Cluster.
  ==› sehr ineffizient bei großen Dateien & direktem statt sequentiellem Zugriff!
  ==› fragmentiert im Lauf der Zeit!
- Keine Rechteverwaltung pro Benutzer oder Gruppe
- Keine Verweise

Linux Ext2/3/4:
- Verzeichnisse (nur Namen und Filenummer ("Inode-Nummer"))
  und Datei-Verwaltungsdaten ("Inode", in eigenem Speicherbereich) getrennt
  ==› Max. Anzahl der Dateien wird beim Formatieren bestimmt!
- Platzverwaltung in Blöcken von 1 bis 4 KB,
  aber nicht verkettet, sondern mit dem Konzept der indirekten Blöcke
  (siehe Tafelbild!).
- "Harte" und "symbolische" Verweise:
  Harte … Mehrere Verzeichnis-Einträge mit selber Filenummer
  Symbolische … Verweis enthält Filenamen des Original-Files
- Zugriffsrechte: Ursprünglich nach dem Konzept Eigentümer / Gruppe / Andere,
  heute auch ACL's
- 3 Datumsangaben (Sekunden- oder Nanosekunden-genau):
  modification time … Zeitpunkt der letzten Änderung des Inhalts
  change time … Zeitpunkt der letzten Änderung der Verwaltungsdaten
  access time … Zeitpunkt des letzten Zugriffs

"Klassische" Zugriffsrechte unter Unix / Linux:
- Jeder File gehört einem Benutzer und gehört zu einer Gruppe.
  Beides ist in den File-Verwaltungsdaten gespeichert.
- Jeder File hat 3 * 3 Berechtigungen:
  Lesen, Schreiben und Ausführen (bei Verzeichnissen: Hineinwechseln)
  jeweils für den Eigentümer, für die Gruppe und für alle anderen.
- Jeder Benutzer gehört zu einer oder mehreren Gruppen
  (z.B. Administratoren, Verwaltung, Schüler, Gäste, …).
- Beim Zugriff gilt:
  - Für den Eigentümer gelten die Eigentümer-Rechte.
  - Für Benutzer, in deren Gruppenliste auch die Gruppe des Files vorkommt,
    gelten die Gruppenrechte.
  - Für alle anderen Benutzer gelten die "andere"-Rechte.

ACL's ("Access Control Lists": Bei Windows, optional bei Linux, und bei allen sicherheitszertifizierten Betriebssystemen):
- Die Zugriffsrechte sind eine beliebig lange Liste von Einträgen.
- Jeder Eintrag enthält einen einzelnen Benutzer oder eine Benutzergruppe
  sowie deren Zugriffsrechte; dazu kommt ein Eintrag mit Zugriffsrechten für "alle".
- Bei jedem Zugriff werden die für den Benutzer passenden Einträge gesucht und
  deren Rechte geprüft.

"Formatierung": Schreibt ein neues, leeres Filesystem auf ein (neues) Speichermedium bzw. eine leere Partition
==› Vor erster Benutzung notwendig
==› Alle alten Daten sind weg!
Bei Platten: Bei der System-Installation wenn C:-Platte, sonst manuell
Bei Wechselmedien (Stick): Meist vom Hersteller vorformatiert.

"File System Check" (DOS-Befehl "chkdsk"): Prüft Filesystem auf Konsistenz:

- Doppelt zugeordneter Platz (in 2 Dateien oder gleichzeitig in Datei + Freiliste)
- Unzugeordneter Platz (weder in Datei noch in Freiliste)
- Platzreservierung und Filegröße verschieden
- "In der Luft hängende" Files und Verzeichnisse.
- Falsche '.' und '..'-Verzeichnisse, ...

Notwendig bei "hartem" Ausschalten oder Absturz, oder bei Entfernen eines Mediums ohne Auswerfen, wenn das Betriebssystem das Filesystem nicht mehr vollständig aktualisieren und auf einen wohldefinierten Stand bringen konnte.
Ob ein Filesystem "sauber" beendet wurde, erkennt man bei FAT nicht, bei allen anderen schon!

Ein Filesystem-Check kann bei großen Filesystemen Stunden dauern
(alle Directories und alle Verwaltungsdaten müssen gelesen werden)!

Bei Filesystemen mit "Journal" (alle modernen Filesysteme): Alle Verwaltungsdaten-Änderungen werden in getrenntem Bereich der Platte davor und danach mitprotokolliert
==› Mehr Aufwand, aber viel schnellerer File System Check (in Sekundenbruchteilen)
(nur Vergleich Journal ‹==› zuletzt geänderte echte Verwaltungsinformation)
Schützt aber nicht vor Hard- und Software-Fehlern (oder menschlicher Dummheit),
und hilft nur gegen verlorene Verwaltungsdaten, nicht gegen verlorene Nutzdaten.

Unterscheide (im Windows eine Option des File System Checks):
Check des Mediums (alle Blöcke einmal lesen ==› lesbar?)

Achtung: Bei FAT: "Datei löschen" ändert nur ersten Buchstaben des Namens im Verzeichnis und hängt den Platz in die Freiliste.
==› Verzeichnis-Eintrag incl. Nummer des Anfangs-Clusters bleibt bestehen!
==› Daten der Datei noch auffindbar!

RAID

"Redundant Array of Independent Disks"

Zur Erhöhung der *Datensicherheit* / der Verfügbarkeit:

- Kompensiert den Ausfall *eines* von mehreren Plattenlaufwerken
  ==› Daten bleiben verfügbar
- Erlaubt den Tausch von "bedenklichen" (hohe Anzahl korrigierter Fehler / kaputter Sektoren) oder defekten Laufwerken im laufenden Betrieb.

Ersetzt regelmäßiges Backup *nicht*:

- Schützt nicht vor Fehlern in der Software / im Betriebssystem.
- Schützt nicht vor menschlicher Dummheit (gelöschten Daten).
- Schützt nicht vor Hardware-Fehlern im Controller.

Weiters:

- RAID erlaubt logische Laufwerke, die größer als eine Platte sind.
- RAID erhöht den Datendurchsatz (zumindest beim Lesen).

Realisierung:

- Entweder in Software im Betriebssystem mit mehreren normalen Platten.
- Oder in Hardware mit speziellen Disk-Controllern.
- Oder bei externen Platten im externen Plattensystem selbst.
- Achtung: Alle beteiligten Platten müssen *gleichgroß* sein!

Im Optimalfall: Auch Controller, Stromversorgung, ... sind "zweigleisig" ("redundant").

4 in der Praxis wichtige Hauptvertreter:

- **RAID 0 ("Striping"):**
  Die Daten werden in Blöcken von 16-256 KB "reihum" gleichmäßig
  auf mehrere Platten verteilt.
  Lesen *und* Schreiben geht schneller bzw. mehr Zugriffe pro Sekunde möglich,
  weil auf mehreren Platten gleichzeitig gelesen / geschrieben werden kann.
  Problem: *Keine erhöhte Datensicherheit* ("0 Sicherheit"), im Gegenteil:
  1 Platte kaputt ==› *Alle* Daten (auf allen Platten) weg!
  (weil: Kein zusätzlicher Platz- bzw. Platten-Bedarf ==› keine redundanten Daten)
- **RAID 1 ("Mirroring", "Spiegelplatten"):**
  Je 2 Platten mit identem Inhalt.
  Schreiben geht etwas langsamer (alles muss auf 2 Platten geschrieben werden).
  Lesen geht etwas schneller (es kann von zwei Platten gleichzeitig gelesen werden).
  Eine Platte darf ausfallen ==› Alle Daten sind auch auf der anderen Platte.
  (nach Austausch der defekten Platte wird wieder eine Kopie gemacht)
  Problem: Doppelter Platzbedarf (doppelt so viele Platten notwendig)!
- **RAID 10: Kombination von 0 und 1:**
  2 idente Gruppen von Platten, Inhalt auf alle Platten jeder Gruppe verteilt.
  Kombiniert Geschwindigkeit und Sicherheit,
  aber braucht auch doppelt so viele Platten wie die Netto-Datenmenge.
- **RAID 5 ("Striping with distributed parity"):**
  Platten sind in Gruppen von n (n ›= 3, meist 4-6) Platten zusammengefasst.
  Die jeweils gleichen Blöcke auf allen n Platten gehören zusammen:
  Von je n Blöcken enthalten n – 1 Nutzdaten, der n-te die Parity der anderen n – 1,
  wobei die Parity-Blöcke reihum auf alle Platten verteilt werden.
  Schreiben ist bestenfalls halb so schnell (alte Nutzdaten & alte Parity-Daten lesen,
  neue Parity berechnen, neue Nutzdaten & neue Parity-Daten schreiben),
  Lesen ist schneller (es kann von n Platten gleichzeitig gelesen werden).
  Bei Ausfall *einer* Platte pro Gruppe können deren Daten rekonstruiert werden
  (aber das System ist in der Ausfall- und Rekonstruktionsphase sehr langsam!).
  Platzbedarf: Nutzdaten * (n / (n - 1))

Das Backup

Kopie der Daten auf ein anderes Medium (externe Platte, Band, DVD, ...).

Zweck:

- Rückspielen des ganzen System im Katastrophen-Fall:
  Nach Platten-Totalausfall, groben Systemfehlern, Brand, Diebstahl usw.
- Rückspielen einzelner Dateien / Verzeichnisse:
  Nach menschlichen Fehlern, Datenmanipulation, ...
- Archivierung selten benutzter Daten, die noch nicht gelöscht werden sollen / dürfen.

Achtung: Ein Backup darf nicht das vorige überschreiben
(Datenverlust bei Fehler während des Backups: Kein altes und kein neues Backup!).
In professionellen Environments: Sicherung in Generationen, z.B.

- Die letzten 3 Tagessicherungen werden aufgehoben,
  zyklisch wird die 4 Tage alte Sicherung mit der neuen überschrieben.
- Die letzten 3 Sonntags-Sicherungen werden aufgehoben,
  d.h. eine Sonntags-Sicherung wird erst nach 4 Wochen überschrieben.
- Die erste Sonntags-Sicherung im Monat wird "ewig" aufgehoben.

Unterschiedliche Methoden:

- Vollsicherung: Alle Daten werden kopiert.
  Eine Vollsicherung allein reicht aus, um das System wiederherzustellen.
- Inkrementelle Sicherung ab Vollsicherung: Es werden nur die Daten gesichert, die sich seit der letzten Vollsicherung geändert haben ==› viel weniger!
  Beim Rückspielen muss man zuerst die letzte Vollsicherung und dann die letzte inkrementelle Sicherung einspielen.
- Inkrementelle Sicherung ab inkrementeller Sicherung: Es werden nur die Daten gesichert, die sich seit der letzten inkrementellen Sicherung geändert haben ==› ganz wenig!
  Aber: Beim Rückspielen muss man zuerst die letzte Vollsicherung und dann *alle* seitdem erstellten inkrementellen Sicherungen einspielen
  (fehlt eine oder ist eine defekt ==› Pech!).

Beispiel:
Sonntag-Sicherung ist Vollsicherung, dazwischen täglich inkrementelle Sicherungen.
(Langzeit-Sicherungen müssen immer Vollsicherungen sein, eine inkrementelle Sicherung allein ist nutzlos!!)

Bei Voll-Backups: 2 Möglichkeiten:

- Filebasiertes Backup: Sichert Verzeichnis für Verzeichnis, Datei für Datei.
- Image-Backup: Kopiert eine ganze Platte oder Partition Sektor für Sektor (lässt ev. unbenutzte Sektoren aus), *ohne* Kenntnis des Inhalts.

Ablauf beim Booten

1.) Das BIOS testet und initialisiert die Hardware, sucht die angeschlossenen Platten (funktioniert auch ohne Platte).

2.) Das BIOS startet den Bootloader im MBR der ersten bootfähigen Platte (oder den Bootsektor einer Floppy oder eines USB-Sticks oder einer CD).

3.) Der Bootloader im MBR schaut, welche Partition als bootfähig markiert ist (oder bietet ein Bootmenü an) und startet das Boot-Programm des Betriebssystems in der jeweiligen Partition.

4.) Das Boot-Programm des Betriebssystems startet das Betriebssystem.

Software

Die rechtliche Seite...

Software-Entwicklung kostet viel Geld
==› Dieses Geld muß mit der Software wieder verdient werden.
==› Die EDV-Branche macht *viel* mehr Umsatz mit Software als mit Hardware!

Software ist geistiges Eigentum des Entwicklers
==› Unterliegt dem Urheberrecht (Copyright, wie Bücher, Musik, ...) und Patenten.
==› Gegen Bezahlung (einmalig oder laufend) erwirbt man ein durch den Lizenzvertrag geregeltes Nutzungsrecht (nicht die Software selbst!) .

Sonderfälle:

- **Shareware:**
  Darf man kostenlos aus dem Internet laden und ausprobieren, zahlt man erst bei Gefallen (freiwillig oder wegen Ablaufdatum, Demoversion, ...)
- **Freeware:** 3 mögliche Bedeutungen von "frei":
  - Frei von Kosten = Gratis: (z. B. Microsoft Internet Explorer)
    Meist Marketing-Strategie!
  - Frei von Rechten = ohne Lizenz nutzbar:
    Der Autor verzichtet auf sein Copyright
    ==› jeder darf mit der Software machen, was er will .
  - Frei von Geheimnissen = *"Open Source"*:
    - Der Quelltext der Software ist öffentlich verfügbar, für jeden einsehbar (bei kommerziellen Produkten ist der Quelltext das wichtigste Firmengeheimnis!) .

      Das gesamte "Know How" des Programms steht damit der Allgemeinheit zur Verfügung .
    - *Nicht* frei von Copyright / Lizenzbestimmungen (typischerweise GPL = "GNU Public License", verbietet z. B. Geheimhaltung, garantiert das Recht auf Weiterentwicklung und Weiterverwendung).
    - Nicht notwendigerweise kostenlos (aber meistens schon) (Verpackungskosten, Kosten für Doku und CD, ... erlaubt).
    - Beispiele: Linux, OpenOffice, Firefox, Apache, GNU C Compiler, ...

Wovon lebt die Open-Source-Softwareentwicklung?

- Von Idealismus.
- Vom Sponsoring durch Firmen, die sich davon Marktvorteile erhoffen.
- Von Einnahmen durch Schulung, Beratung, Support, Dienstleistung.

Schichten der Software

Die Software eines Rechners lässt sich wie folgt einteilen:

1.) Die Treiber: Kümmern sich um die einzelnen Hardware-Geräte (ein Treiber pro Gerät). Logisch im oder unter dem Betriebssystem-Kern angesiedelt.

2.) Das Betriebssystem (genauer: der Betriebssystem-Kern):
"Oberster Chef", verwaltet den Rechner.
Läuft mit höheren Hardware-Privilegien (x86: "Ring 0") als alles andere,
darf z.B. Speicherverwaltungsbefehle und I/O-Befehle ausführen.

3.) Bibliotheken, Shared Libraries (Windows: .dll, Linux: .so):
Sammlungen von Codestücken, die von mehreren Programmen benutzt werden (nicht eigenständig ausführbar).

Beispiele:

- Mathematische Funktionen, Text-Funktionen, ...
- Grafische Elemente (Menüs, Buttons, ...)
- Anzeige von JPEG's, von HTML, ...

4.) Die Dienste / Services / Daemons: Ständig laufende Programme, von denen jedes eine bestimmte Funktionalität allgemein zur Verfügung stellt.
Beispiele: Druckdienst, Webserver, Fileserver, Terminal-Server, Datenbanken, Systemprotokoll, Ausführung von Programmen zu einem bestimmten Zeitpunkt, ...
Unter Linux ist auch Bildschirm-, Tastatur- und Maus-Verwaltung ein Dienst (der Xwindows-Server), unter Windows ist das Teil des Betriebssystems.

5.) Commandline Interpreter:
Das "DOS-Fenster" (command.com bzw. cmd.exe) unter Windows
bzw. die Shell (in Kombination mit einem Terminal-Fenster) unter Linux.

Meist gemeinsam mit den Befehlszeilen-Programmen betrachtet
(diese sind aber technisch großteils normale Anwendungen – nur ohne GUI – und *kein* Bestandteil des Commandline Interpreters).

6.) Grafische Benutzeroberfläche:

- Verwaltung der Fenster
- Taskleiste bzw. Panel usw.

7.) Anwendungen:
Office-Programme, Web-Browser, Media-Player, Mail-Programm, Editor, Photoshop, AutoCAD, Spiele, ...

Auch: Compiler und Interpreter, Werkzeuge zur Software-Entwicklung.

Auch: Hilfsprogramme und System-Anwendungen (File Manager, Backup, Systemeinstellungen und Systemverwaltungsprogramme, Systemlog, ...).

Unterscheide:

- Standard-Software: ``fertiges Packerl aus dem Regal''
- Individual-Software: Auf Auftrag eigens angefertigt ==› Teurer!!!

Zwischen Betriebssystem & Treiber einerseits und dem gesamten Rest andererseits liegt eine dicke Grenze (betreffend Rechten, Ausführung, ...): Alles, was nicht Betriebssystem ist (4.-7.), läuft aus technischer Sicht als Anwendung (Windows: "Task", Linux: "Prozess").

Das “Schalenmodell” (wobei 4.-7. als eine Schale zählt):

- Jede Schicht nutzt nur Funktionen der unmittelbar darunterliegenden Schicht.
- Jede Schicht stellt der darüberliegenden Schicht bestimmte Funktionen zur Verfügung.

Sinn:

- Modularisierung .
- “Abstraktion”, Verstecken der Details .

Beispiel:

- Ein C-Programm will eine Zahl in eine Datei schreiben und ruft dazu die Library-Funktion `fprintf` mit der Zahl auf.
- Die Library verwandelt die Zahl in einzelne Zeichen und macht mit diesen Zeichen einen `write`-Betriebssystem-Aufruf.
- Das Betriebssystem weiß, in welchen Sektoren welcher Platte die Daten dieser Datei stehen, und schafft dem Treiber an, die Daten in Sektor Nummer … der Platte zu schreiben.
- Der Treiber weiß, welche Befehle er an den SATA-Controller schicken muss, damit die Daten geschrieben werden.

Anwendungen dürfen **nicht** direkt auf die Hardware zugreifen!
(das wird vom Betriebssystem technisch verhindert)

Der Betriebssystem-Kern

Aufgaben:

- Fährt das System hoch und startet alle anderen Programme.
- Verwaltet / verteilt die Hardware (Speicher, Platte, Bildschirm & Tastatur, …):
    - Teilt allen anderen Programmen Speicher zu.
    - Teilt allen anderen Programmen zeitweise die CPU (d.h. Rechenzeit) zu.
    - Regelt die Hardware-Zugriffe aller anderen Programme (oder verhindert z.B. unzulässige gleichzeitige Zugriffe), verteilt die Eingaben auf die Programme.
- Implementiert die Filesysteme, verteilt den Plattenplatz.
- Implementiert die Netzwerk-Protokolle (TCP/IP).
- Überwacht Benutzer und Rechte (wer darf was?),
  schützt Programme und Benutzer voreinander.
- Ermöglicht die Kommunikation zwischen Programmen.
- Stellt allen anderen Programmen eine einheitliche, hardware-unabhängige Schnittstelle zur Hardware, Filesystem, Netzwerk usw. zur Verfügung.

Der Benutzer hat normalerweise **nicht** direkt mit dem Betriebssystem-Kern zu tun (“sieht” nichts von ihm), sondern nutzt ihn nur indirekt (über Programme).

Die Treiber

Das Betriebssystem enthält den allgemeinen Code (ohne Kenntnis, welche Hardware jetzt genau im Rechner steckt), die Treiber den hardware-spezifischen Code für ganz bestimmte Plattencontroller, Netzwerk-Chips, Grafik- und Soundkarten usw.:

- Nur die Treiber greifen direkt auf die I/O-Hardware zu.
- Die Treiber werden aktiv, wenn ein Gerät signalisiert, dass es etwas zu tun gibt (Daten da, Fehler aufgetreten, externes Gerät ein- oder ausgesteckt), oder wenn das Betriebssystem den Befehl dazu gibt.
- Die Treiber kommen oft vom Hardware-Hersteller, nicht vom Betriebssystem-Hersteller.

Tasks, Threads usw.

Alles, was außer dem Betriebssystem selbst und den Treibern auf einem Computer läuft (Anwendungen, Dienste, Commandline-Interpreter bzw. Batch-Skripts, ...) wird vom Betriebssystem in Tasks und Threads verwaltet.

Ein **"Task"** (Windows-Ausdruck, unter Unix/Linux heißt dasselbe "Prozess") ist ein laufendes Programm (egal, ob es gerade wirklich rechnet oder auf etwas wartet):

- Wenn ein Programm gestartet wird, erzeugt das Betriebssystem einen neuen Task dafür. Endet das Programm, verschwindet auch der Task.

  Wenn dasselbe Programm mehrmals gleichzeitig gestartet wird, werden im Normalfall entsprechend viele getrennte Tasks erzeugt.
- Jeder Task führt einen bestimmten .exe-File, d.h. genau ein Programm, aus.
- Ein Task bekommt vom Betriebssystem Speicher zugeteilt, der ihm und nur ihm gehört: Das Betriebssystem sorgt dafür, dass jeder Task nur auf den ihm zugeteilten Speicher zugreifen kann und nicht auf den Speicher anderer Tasks.

  Stürzt ein Task gröber ab (versucht beispielsweise, auf fremden Speicher zuzugreifen, das hieß früher "Schutzverletzung" bzw. nach dem Fehlersuch-Programm "Dr. Watson"), schreibt das Betriebssystem auf Wunsch den gesamten Speicherbereich des Tasks zwecks Fehlersuche in eine Datei.
- Ebenso verwaltet das Betriebssystem für jeden Task einzeln eine Liste von Dateien, Geräten und Netzwerk-Verbindungen, auf die der Task gerade zugreift.

  Endet der Task, schließt das Betriebssystem die von ihm belegten Dateien und Netzverbindungen.

  Weiters merkt sich das Betriebssystem für jeden Task getrennt, in welchem Arbeitsverzeichnis er gerade ist.
- Ein Task läuft mit den Rechten und Privilegien eines bestimmten Benutzers. Diese legen fest, was der Task darf und was nicht (welche Dateien und Geräte er sieht bzw. öffnen darf, ob er Systemeinstellungen ändern darf, ob er andere Tasks starten oder beenden darf, ob er Netzwerkverbindungen machen darf, ...).

  Weiters können im Betriebssystem Limits pro Task konfiguriert sein: Maximal zugeteilte Speichergröße, maximaler Rechenzeit-Verbrauch, maximale Anzahl

offener Dateien und Netzverbindungen, maximale Größe angelegter Dateien, maximale Anzahl von Threads im Task, …

Unter Windows sieht man die laufenden Tasks im Task-Manager.

In einem Task läuft zumindest ein **"Thread"**, ev. auch mehrere. Während der Task die Einheit der Rechte- und Ressourcenverwaltung ist, ist der "Thread" die Ausführungseinheit:

- Das Betriebssystem teilt die CPU (bzw. einen Core der CPU) zu jedem Zeitpunkt einem bestimmten Thread zu (und nimmt ihm die CPU auch wieder weg, wenn er zu lange rechnet oder ein wichtigerer Thread die CPU braucht).
- Jeder Thread hat daher immer einen bestimmten Ausführungs-Status:
  - Er rechnet selbst gerade.
  - Er hat gerade einen Betriebssystem-Aufruf gemacht, d.h. das Betriebssystem rechnet gerade für ihn.
  - Er würde gerne rechnen, aber hat vom Betriebssystem noch keine CPU zugeteilt bekommen.
  - Er wartet auf irgendetwas (Benutzereingabe, Netzwerk-Daten, Plattenzugriff, Ablauf eines Timers, Daten von anderen Threads oder Programmen, …).
  - Er ist angehalten (z.B. durch den Debugger).

  Zu jedem Thread gehört daher zu jedem Zeitpunkt eine bestimmte Stelle im Code des Programms (jenes Tasks, zu dem der Thread gehört), an der der Thread gerade rechnet oder steht.
- Jeder Thread gehört zu genau einem Task und "erbt" dessen Speicher, dessen Rechte, dessen offene Dateien und Netzwerkverbindungen usw..

  Umgekehrt sehen alle Threads eines Tasks denselben Speicher und dieselben Dateien bzw. Netzwerkverbindungen, sind also nicht voreinander geschützt.

Hat ein Task nur einen Thread (das ist der Normalfall), so kann er auch nur einen Core nutzen und nicht mehrere Dinge gleichzeitig tun. Soll ein Programm eine Dual-Core oder Quad-Core-CPU sinnvoll ausnutzen, muss es daher explizit so programmiert sein, dass es intern mehrere Threads gleichzeitig startet: Dann kann das Programm auch auf mehreren Cores gleichzeitig rechnen (pro Core ein Thread: "Multithreaded").

Das Erzeugen von Tasks und das Umschalten der CPU zwischen Threads verschiedener Tasks ist ziemlich aufwändig (weil jedesmal auch die Speicherzuordnung umgeschaltet werden muss), das Erzeugen von Threads und der Wechsel zwischen Threads desselben Tasks geht im Vergleich dazu sehr schnell (weil sich an Speicher- und Rechte-Verwaltung nichts ändert).

Der Scheduler

Der Scheduler ist jener Teil des Betriebssystems, der die CPU an die Threads verteilt. Er vereint im Normalfall zwei Mechanismen:

- **Prioritätssteuerung:** Jeder Thread hat eine Priorität (Multimedia-Anwendungen

sind z. B. sehr zeitkritisch, Word ist mittelkritisch, Hintergrund-Anwendungen wie der Druckdienst oder ein Backup eher unkritisch). Wird ein hochpriorer Thread bereit (Daten kommen, Timer läuft ab, ...), während ein weniger wichtiger Thread gerade rechnet, nimmt der Scheduler dem weniger wichtigen Thread sofort die CPU weg und gibt sie dem wichtigen.

- **Zeitsteuerung:** Wollen mehrere Threads gleicher Priorität gleichzeitig rechnen, so bekommen sie reihum einer nach dem anderen die CPU für kurze Zeit (1 – 20 ms) zugeteilt ("Round Robin"). Für den Benutzer sieht es so aus, als ob alle gleichzeitig rechnen, aber eben nur mit 1/2, 1/3 oder entsprechend weniger der CPU-Leistung.

  Auch weniger wichtige Threads, die gleichzeitig rechnen wollen, bekommen selbst unter Vollast normalerweise periodisch ein kleines bisschen CPU (unter Windows: Alle 3 Sek max. 16 ms), damit sie nicht ganz "verhungern".

Der Scheduler läßt sich im Idealfall der Nutzungsart des Computers anpassen: Bei Servern (ohne interaktive Benutzer) steht die Effizienz im Vordergrund: Man möchte möglichst selten Threads wechseln, um den Overhead gering zu halten, und nimmt dafür hohe Reaktionszeiten und eine "ruckelige" Ausführung in Kauf. Bei Desktop-Rechnern und Notebooks oder gar Steuerungen ist es umgekehrt: Man möchte möglichst unmerklich kurze Reaktionszeiten und nimmt dafür den Overhead vieler Threadumschaltungen in Kauf.

Echtzeit-Betriebssysteme

Das sind Betriebssysteme, die unter genau definierten Umständen maximale Reaktionszeiten garantieren (z.B. "maximal 5 µs nach Eintreffen der Daten wird der dazugehörige Thread aktiv" oder "die Ausführungszeitpunkte eines Thread, der an einem 1000-Hz-Timer hängt, schwanken um max. 1 %").

Beispiele für Computer, die Echtzeit-Systeme erfordern:

- Steuerungen, Messwerterfassungen und Sicherheitssysteme.
- Multimedia-Systeme, Telekommunikations-Systeme = Telefonvermittlungen usw. (wegen Bild-Rucklern und vor allem wegen Knacksern oder Frequenzschwankungen im Ton: Man hört Aussetzer <= 1 ms !).
- Pilotentrainer usw. (dort müssen 30 oder 60 Frames per Sec sowie mechanisches Feedback < 1/30 Sek garantiert werden).

Es gibt für diesen Zweck zahlreiche Spezialsysteme. Windows ist nicht Echtzeit-tauglich, Linux kann optional auf Echtzeit-Betrieb umgeschaltet werden, gibt aber keine hundertprozentige Garantie.

Die Speicher-Verwaltung

Jedes "ausgewachsene" Betriebssystem bietet heute **"virtuelle Speicherverwaltung"** (Das Original-DOS in den 80-er-Jahren und manche Steuerungs- oder primitive Handy-Betriebssysteme kennen noch keine virtuelle Speicherverwaltung). Die Idee ist, dass die Speicheradressen, die ein Programm bei der Ausführung verwendet, nicht die tatsächlichen Hardware-RAM-Adressen sind.

- Vom gesamten möglichen Adressbereich (bei 32-Bit-Rechnern 2^{32} Bytes, also 4 GB)

ist ein Teil (normalerweise das obere Ende) fix für das Betriebssystem und die I/O-Geräte reserviert (z.B. 1 GB), der Rest für die normalen Programme (z.B. 3 GB).

- Für jeden Task sieht es so aus, als ob er die kompletten 3 GB von 0 beginnend (bzw. von 0x10000 weg, die unteren 64 KB gehören meist auch dem System) für sich allein hätte. Jeder Task glaubt also, ganz allein auf einem PC mit 3 GB RAM zu laufen (auch wenn der PC in Wirklichkeit weniger RAM hat!), vom Speicher anderer Tasks weiss und sieht er nichts.

  Das heißt auch, dass alle Tasks immer dieselben Speicheradressen verwenden: Ein Programm muss beim Start nicht an den Speicher angepasst werden, der gerade wirklich frei ist.

- Eine Hardware-Einheit (die MMU = Memory Management Unit) in der CPU ordnet bei jedem Speicherzugriff, den der gerade laufende Task macht, jeder Speicheradresse im Task ("virtuelle Adresse") eine tatsächliche RAM-Adresse ("reale Adresse") zu.

  Das geschieht in fixen Blöcken von 4 KB: Jedem 4-KB-Block im Adressbereich eines jeden Tasks (oder des Betriebssystems) kann ein beliebiger 4-KB-RAM-Bereich zugeordnet werden, oder der 4-KB-Block kann als "nicht zugeordnet" markiert werden.

  Die Programmierung der MMU (d.h. die Zuordnung des Speichers) geschieht durch das Betriebssystem.

- Da das tatsächlich vorhandene RAM oft nicht reichen würde, um allen Tasks ständig ihre jeweils 3 GB zuzuordnen, verwaltet das Betriebssystem auch noch einen speziellen File (Windows: PAGEFILE.SYS) oder eine eigene Platten-Partition (Linux: Swap-Partition) als RAM-Erweiterung:

  Selten benutzte 4-KB-RAM-Blöcke werden bei Speicherknappheit in den Pagefile verschoben, damit der Platz im RAM für gerade wichtigere Daten verwendet werden kann, und bei Bedarf wieder (an eine andere Stelle im RAM) zurückgeholt.

- Greift ein Task auf einen 4-KB-Bereich zu, dem kein RAM zugeordnet ist, wird der Zugriff von der MMU abgefangen, der Task unterbrochen, und das Betriebssystem informiert. Es gibt folgende Möglichkeiten:

  - Der Speicherbereich war schon vom Task benutzt (enthält also schon Daten des Tasks), wurde aber vom Betriebssystem wegen RAM-Mangels in den Pagefile verschoben:

    Dann werden freie 4 KB RAM gesucht (ev. muss man dafür andere 4 KB desselben oder eines anderen Tasks in den Pagefile verschieben), die alten Daten aus dem Pagefile ins RAM hereinkopiert und dem Task zugeordnet, und der Task normal fortgesetzt, ohne dass er etwas davon merkt.

    Da ein Plattenzugriff knapp eine Million mal langsamer ist als ein Speicherzugriff, bricht die Geschwindigkeit eines Rechners stark ein, wenn das RAM knapp wird und ständig Daten zwischen Pagefile und RAM hin- und herkopiert werden müssen, bevor die Programme weiterrechnen können.

  - Der Speicherbereich war noch nie vom Task benutzt, aber darf benutzt

werden:

Dann sucht das Betriebssystem freie 4 KB RAM, ordnet sie dem Task zu, und lässt ihn normal weiterrechnen.

Das passiert, weil nicht jeder Task gleich zu Beginn vorbeugend die vollen 3 GB zugeordnet bekommt, sondern zuerst einmal gar nichts: Es wird nur jener Teil der 3 GB im Laufe der Zeit reserviert und zugeordnet, den der Task wirklich nutzt (in der Realität nutzen die meisten Tasks viel weniger als ihre 3 GB, denn sonst würden ja nie mehrere Tasks gleichzeitig in einem Rechner mit 1 oder 2 GB RAM Platz haben!).

Unter Linux kann es passieren, dass das System keinen freien Speicher mehr findet, obwohl der Task eigentlich noch neuen Speicher belegen dürfte. Dann wird irgendein Task (der "gefräßigste") abgebrochen, um wieder freien Platz zu schaffen (OOM-Killer, Out-Of-Memory-Killer). Windows ist da vorsichtiger: Wenn RAM und Pagefile nicht mehr reichen, um den theoretischen Speicher-Maximalbedarf eines Tasks abzudecken, lässt sich der Task gleich von vornherein nicht starten - wenn er einmal gestartet ist, ist normalerweise auch sein Speicher garantiert.

- Der Speicherbereich gehört zum Programmcode oder den DLL's des Task:

  In diesem Fall lädt das Betriebssystem das entsprechende Stück Programmcode von der Platte in ein freies 4-KB-RAM-Stück, ordnet dieses dem Task zu, und lässt ihn normal weiterrechnen.

  Hier gilt dasselbe wie für den Datenspeicher: Es wird nicht gleich beim Start der gesamte Programmcode eines Tasks ins RAM geladen, sondern er wird Stück für Stück bei Bedarf von der Platte geholt, um RAM zu sparen.

  Bei DLL's ist es oft der Fall, dass schon ein anderer Task dasselbe Stück Code bekommen hat. Dann bekommt der aktuelle Task das schon vorhandene Codestück im RAM zugeordnet: Die DLL steht also nur einmal im Speicher, aber ist für mehrere Tasks gleichzeitig verwendbar.

- Der Task darf gar nicht auf diesen Speicherbereich zugreifen.

  Das kann passieren, wenn der Task auf einen Betriebssystem-Bereich außerhalb seiner 3 GB zugreift, oder auf einen Bereich innerhalb seiner 3 GB falsch zugreift (z.B. versucht, einen Programm- oder DLL-Bereich zu überschreiben oder einen reinen Datenbereich auszuführen).

  In diesem Fall muss es sich um einen Programmfehler oder böswilligen Code handeln: Der Task wird vom Betriebssystem abgebrochen ("Schutzverletzung").

  Die Speicherverwaltung dient also nicht nur der effizienten RAM-Verwaltung, sondern auch der Sicherheit: Bei entsprechender Einstellung wird verhindert, das ein Programm gerade selbst erzeugten oder z.B. aus dem Internet oder einem Office-Dokument geladenen Schadcode ausführt: Daten im Datenbereich lassen sich zwar lesen und schreiben, aber nicht ausführen, und der bestehende Programmcode (vom .exe-File oder einer DLL) lässt sich zwar ausführen, aber nicht ändern.

Unter 32-Bit-Windows ist die Aufteilung des Adressraumes 2 GB pro Task plus 2 GB fix für

das Betriebssystem (ein Programm kann also maximal 2 GB RAM nutzen, egal, wie viel der Rechner eingebaut hat), beim Booten kann stattdessen eine Aufteilung 3 GB / 1 GB festgelegt werden. Unter Linux kann die Grenze bei der Konfiguration des Kernels frei konfiguriert werden (Standard ist 3+1), alternativ ist ein Modus mit 4 GB pro Task + 4 GB für das System wählbar, der aber wegen komplizierterer und häufigerer MMU-Umschaltungen einige Prozent langsamer ist. Unter 64-Bit-Systemen ist das kein Thema: Je 8 ExaByte (2^{63} Bytes) pro Anwendung und für das System sollten reichen...

Virtuelle Maschinen

Das Konzept des virtuellen Speichers lässt sich auf Hardware- und Software-Ebene noch weiterführen:

- Es lässt sich nicht nur der Speicherbereich der einzelnen Tasks, sondern auch der Speicherbereich des Betriebssystems (unter Kontrolle eines anderen Betriebssystem) virtualisieren.
- Es lassen sich nicht nur die RAM-Adressen, sondern auch die Adressen der I/O-Geräte von der MMU abfangen. Dann wird jedesmal das "übergeordnete" Betriebssystem aufgerufen, wenn das virtuelle System einen I/O-Zugriff macht, und kann dem virtuellen System die I/O-Geräte "vortäuschen".

Auf diese Weise kann man auf einem realen PC ein oder mehrere virtuelle PC's (jeweils mit eigenem Speicher und eigenen I/O-Geräten) vortäuschen und – unter der Kontrolle eines "Wirts-Betriebssystems" - in jedem dieser virtuellen PC's ein eigenes Betriebssystem ("Gast-Betriebssystem") samt Anwendungen starten.

Da der virtuelle PC samt seinem Betriebssystem und den Anwendungen ja ein reines Gebilde im Speicher des echten PC's ist, lässt er sich mitten im Betrieb vom Wirts-System aus einfrieren, auf einen File kopieren, und später auf demselben oder einem anderen Rechner in demselben Zustand wieder laden und fortsetzen.

Netzwerke

Das OSI/ISO-7-Schichten-Modell

Idee:

- Oben (Schicht 7): Anwendung / unten (Schicht 1): Leitung.
- Jede Schicht kümmert sich um eine Teilaufgabe der Datenübertragung.
- Jede Schicht redet nur mit der Schicht unmittelbar darüber / darunter:
    - Die Daten wandern beim Sender von Schicht 7 Schicht für Schicht auf Schicht 1 und beim Empfänger wieder durch alle Schichten zurück.
    - Dazwischen können die Daten während der Übertragung ein paar Schichten hoch- und wieder runterwandern, z.B. an einer Firewall (von 1 bis 4 und zurück auf 1) oder beim Übergang auf ein anderes Übertragungsmedium (z.B. WLAN / Ethernet) (bis Schicht 2 oder 3 und zurück).
- Beim Sender
    - übernimmt jede Schicht die zu sendenden Daten der darüberliegenden Schicht (als "Black Box", d.h. ohne ihren Aufbau oder Inhalt zu kennen),
    - zerstückelt sie eventuell in Pakete bestimmter Größe oder fasst mehrere Daten zu einem Paket zusammen, oder komprimiert oder verschlüsselt die Daten,
    - hängt einen Header mit ihren Protokolldaten vor jedes einzelne Paket
    - und eventuell eine Prüfsumme o.ä. hinter jedes einzelne Paket,
    - und gibt die Pakete an die darunterliegende Schicht zum Senden weiter.
- Beim Empfänger
    - bekommt jede Schicht von der darunterliegenden Schicht paketweise die empfangenen Daten,
    - prüft / verarbeitet ihren Header und ihre Prüfsumme und entfernt sie,
    - sammelt eventuell die Daten mehrere Pakete zusammen,
    - und gibt sie dann an die darüberliegende Schicht zur Verarbeitung weiter.

Die Schichten:

- Schicht 1: Bitübertragungsschicht (engl. *Physical Layer*)

  Physikalische Datenübertragung einzelner Bits oder Bytes: Kabeleigenschaften, Stecker, elektrische / optische Spezifikation der Darstellung der Bits am Kabel, Signal-Timing, …

  Auf dieser Ebene bewegen sich z.B. die Antenne oder der Repeater (Kabel-Verlängerer bzw. -Verstärker) und ganz alte Ethernet-Hubs.

- Schicht 2: Sicherungsschicht (engl. *Data Link Layer*)

  Übertragung eines Paketes zwischen zwei Geräten desselben Kabels oder Netzes.

Legt fest:

- Aufbau des Paketes:
  Physikalische Adresse (MAC-Adresse, ist nicht gleich TCP/IP-Adresse!), Prüfsumme, Paketgröße, …
- Zugriff auf das Übertragungsmedium (Wer darf wann senden bzw. wie teilt man sich das Kabel?)

Die gebräuchlichen Übertragungstechnologien (Ethernet, WLAN, …) umfassen Schicht 1 und Schicht 2.

- Schicht 3: Vermittlungsschicht (engl. *Network Layer*)

  Diese Schicht sorgt dafür, dass die Daten von einem Ende der Kommunikation zum anderen finden: Sie regelt, wie die Daten von einem Rechner zum nächsten, von einem Teilnetz ins nächste, von einem Provider zum nächsten gelangen, denn die Schichten 1 und 2 "sehen" nur das jeweils nächste Gerät, nicht den ganzen Weg der Daten.

  Dazu gehört das Routing (was ist der nächste Rechner am Weg zum Ziel?), aber auch das Zerteilen von Daten in handliche Stücke (Limit rund 1500 Bytes pro Paket).

  Der wichtigste (und heute fast einzige) Vertreter auf dieser Schicht ist IP ("Internet Protocol"), die untere Hälfte von TCP/IP.

  Schicht 3 ist die unterste *Medium-unabhängige* (für alle Übertragungsmedien gleiche) Schicht. Zugleich ist es die oberste Schicht, die nur mit einzelnen Paketen und nicht mit Verbindungen zu tun hat.

- Schicht 4: Transportschicht (engl. *Transport Layer*)

  Diese Schicht stellt sicher, dass die Daten einer Verbindung richtig ankommen: Sie ergänzt die Daten um Prüfsummen, wiederholt falsche oder verlorengegangene Pakete, regelt die Sendegeschwindigkeit, um Stau zu vermeiden, bringt Pakete, die in vertauschter Reihenfolge ankommen, wieder in die richtige Reihenfolge, verschickt Bestätigungen für korrekt empfangene Daten, usw..

  Bei TCP/IP liegt auf dieser Ebene auch der logische Verbindungs-Auf- und Abbau (entspricht dem "Wählen" und dem "Auflegen" beim Telefon): Jedes verschickte Paket gehört zu einer Verbindung.

  Der wichtigste Vertreter ist TCP ("Transmission Control Protocol"), die obere Hälfte von TCP/IP. Will man nur einzelne Pakete ohne Verbindung versenden, wird stattdessen UDP verwendet (selten), das dritte Schicht-4-Protokoll der TCP/IP-Protokollfamilie ist SCTP (noch sehr neu und selten).

  Bei den TCP/IP-Protokollen ist das die oberste *anwendungs-unabhängige* Schicht (d.h. die gemeinsame Basis für Mail-, Web-, Filetransfer- usw. -Protokolle): Sie bietet den darüberliegenden Schichten eine gesicherte Verbindung für einen beliebigen Strom von Bytes in beide Richtungen.

- Schicht 5: Sitzungsschicht (engl. *Session Layer*)

  Hier geht es um den Aufbau und die Verwaltung von Sitzungen (bzw. die Wiederherstellung einer Sitzung nach einer Verbindungsunterbrechung).

Diese Ebene bleibt bei TCP/IP meist ohne eigene Ausprägung, mit Ausnahme von RPC (einem Protokoll für Filesharing und Client-Server-Anwendungen), das auf Schicht 5 angesiedelt ist.

- Schicht 6: Darstellungsschicht (engl. *Presentation Layer*)

  Auf dieser Schicht wird die Darstellung der Anwendungsdaten festgelegt: Hier sind (zumindest theoretisch) Zeichesatzumwandlung, Big-Endian / Little-Endian-Konvertierung, Komprimierung und Verschlüsselung angesiedelt, ebenso die Serialisierung interner Daten ("Wie schicke ich aus verschiedenen Elementen zusammengesetzte Daten, z.B. Personendaten, byteweise nacheinander auf das Netz?").

  Bei TCP/IP ist diese Schicht meist nicht separat ausgeprägt.

- Schicht 7: Anwendungsschicht (engl. *Application Layer*)

  Hier sind die eigentlichen Anwendungsprotokolle und -programme angesiedelt (siehe getrennte Übersicht).

Die Kopplungsgeräte:

- Repeater und Hub:

  Ein Repeater verlängert eine Leitung zwischen zwei Geräten.
  Ein Hub koppelt Leitungen zu mehreren Geräten.

  Beide arbeiten rein physikalisch, also auf Schicht 1: Sie schauen sich die durchfließenden Bits nicht an, sondern verstärken nur die elektrischen / optischen Signale.

- Bridge und Switch:

  Eine Bridge koppelt *zwei* Segmente eines Netzes oder *zwei* Geräte (auf Schicht 2, d.h. es verhält sich aus logischer Sicht wie ein einziges Netz oder Kabel).

  Ein (klassischer) Switch koppelt *mehrere* Geräte oder Netz-Segmente analog zu einer Bridge (Schicht 2) (moderne Switches arbeiten teilweise auf Schicht 3).

  Beide koppeln die angeschlossenen Geräte oder Netzsegmente aber nicht elektrisch, sondern empfangen und analysieren die Pakete (genauer: Die Ziel-MAC-Adresse) und senden sie frisch: Bei einem Repeater oder Hub (Schicht 1) sehen alle angeschlossenen Netzsegmente alle Daten, bei einer Bridge oder einem Switch werden die Daten nur mehr in das Netz-Segment / an das Gerät geschickt, an dem der Empfänger hängt.

- Der Router:

  Ein Router koppelt verschiedene Netze (derselben Art, also beide TCP/IP-basiert): Er kümmert sich darum, jene Datenpakete, deren Ziel nicht im Netz des Senders liegt, in das richtige Netz bzw. zum richtigen nächsten Zwischenknoten auf dem Weg zum Ziel zu schicken.

  Dazu muss der Router die logische Adresse (IP-Adresse) des Paketes analysieren, in seiner Routing-Tabelle nachsehen, wohin das Paket gehört, eine frische physikalische Adresse (MAC-Adresse des nächsten Zwischengerätes) in das Paket eintragen, und es auf der richtigen Leitung weiterschicken. Er arbeitet also auf Schicht 3.

- Die Firewall:

  Ist im Normalfall ein Router, der eben nicht alles durchläßt bzw. weiterschickt, sitzt also auf Schicht 3 oder 4 und schaut sich die Ziel-Adressen oder die einzelnen Verbindungen an (bzw. bei NAT-Routern: Ändert sie auch), aber nicht die Daten selbst.

  Manche Firewalls sehen sich auch den Inhalt an (Virenscanner bei Mails, Filterung bei Web-Zugriffen, ...), dekodieren die Daten also komplett (bis Schicht 7).

- Gateway und Proxy:

  Diese Programme bzw. Geräte arbeiten auf Ebene der Anwendungsprotokolle oder der Nutzdaten (also Schicht 7), indem sie z.B. Daten zwischen zwei verschiedenen Protokollen oder Netztypen übertragen (Gateway, z.B. zwischen IP-Telefonie und -Fax und "echtem", altem Telefonnetz, oder "echtes" Fax zu E-Mail, oder E-Mail zu SMS), oder indem sie Daten zwischenspeichern (Web-Proxy).

TCP/IP

Der "kleinste gemeinsame Nenner" praktisch aller lokalen Netzwerk-Anwendungen und aller Internet-Anwendungen ist TCP/IP ("Transmission Control Protocol" / "Internet Protocol") und alle darauf aufbauenden / damit in Zusammenhang stehenden Protokolle (die TCP/IP-Protokoll-Familie).

Einige wichtige Anwendungs-Protokolle (basieren alle auf TCP/IP):

- SMTP ("Simple Mail Transfer Protocol"): Senden von Mails, Transfer von Mails zwischen Mail-Servern
- POP ("Post Office Protocol"), IMAP ("Internet Message Access Protocol"): Empfangen von Mail
- HTTP ("Hyper Text Transfer Protocol"): Webseiten
- FTP ("File Transfer Protocol"): Filetransfer (lesen, schreiben, Verzeichnisse anlegen, ...)
- NNTP ("Network News Transfer Protocol"): Das klassische, alte (1986!) Protokoll für Diskussionsgruppen (nur Text)
- NFS ("Network File System"), SMB / CIFS ("Common Internet File System"): Fileshares, Unix und Windows
- CUPS ("Common Unix Print System"): Netzwerk-Drucker
- LDAP ("Lightweight Directory Access Protocol"): Verzeichnisdienst, u.a. für Benutzernamen
- Telnet, SSH ("Secure Shell"): Remote Login
- RDP ("Remote Desktop Protocol"): Windows-Desktop auf anderem PC, Windows Terminal Server
- Xwindows: Dasselbe für Linux / Unix
- PXE ("Preboot Execution Environment"): Booten vom Netzwerk
- SIP ("Session Initiation Protocol"): IP-Telefonie

Ganz grob lassen sich die Netzwerkdienste und -Protokolle wie folgt einteilen:

- Lokale Dienste:
    - Netzwerk-Laufwerke
    - Druckserver
    - Anmelde-Dienst, zentrale Benutzer- und Passwort-Verwaltung
- Protokolle, mit denen man auf einem anderen Rechner arbeiten kann:
    - Telnet und SSH/Putty
    - Remote Desktop, VNC, X windows
- "Klassische" Internet-Dienste:
    - FTP File Transfer
    - WWW (+WebDAV, ...)
    - Mail
    - NNTP
- Internet-Medien und -Kommunikation:
    - Telefon: SIP, Skype
    - Internet-Radio & TV Streaming (RealNetworks)
    - Chat-Dienste: IRC, ICQ
- Online- und Multiuser-Games
- Filesharing -Netzwerke

Einige wichtige System-Protokolle (teils "neben" oder "unter" TCP/IP, teils darauf aufbauend):

- ICMP ("Internet Control Message Protocol"): Für "ping" und "traceroute", einige Fehlermeldungen auf IP-Ebene (z.B. "ich finde keine Route zum Ziel-Rechner"), "brems dich ein" (langsamer senden, Empfänger kommt nicht mit oder Leitung ist verstopft) und andere Verwaltungsaufgaben.
- ARP / RARP ("(Reverse) Adress Resolution Protocol"): Zur Zuordnung von IP-Adressen zu MAC-Adressen (Schicht 2 ‹==› Schicht 3) und umgekehrt.
- DHCP ("Dynamic Host Configuration Protocol"): Zur automatischen TCP/IP-Netzwerk-Konfiguration beim Starten eines Rechners.
- DNS ("Domain Name System"): Zur Zuordnung von Netzwerk-Adressen zu Netzwerk-Namen, siehe später.
- NTP ("Network Time Protocol"): Zeitsynchronisation
- SNMP ("Simple Network Management Protocol"): Zur Konfiguration und Überwachung von professionellen Netzwerk-Komponenten.
- BGP ("Border Gateway Protocol"): Das Protokoll, mit dem die Provider untereinander Routing-Information austauschen.

Arten der Kommunikation:

- *Client-Server-Kommunikation:* Das Programmier- und Kommunikationsmodell der meisten Netzwerk-Dienste:
    - Der Server (Webserver, Mailserver, Fileserver, Druckserver, ...):
        - Sitzt meist auf einem zentralen Rechner (Rechenzentrum, Provider).
        - Läuft ständig und ohne Benutzer-Interface.
        - Verwaltet irgendwelche Daten (Mails, Webseiten, Datenbanken, Files, Benutzernamen und -Passwords, ...) oder Geräte (Drucker).
        - Wartet auf Verbindungen von Clients und schickt ihnen auf deren Anfrage eine Antwort (baut selbst keine neuen Verbindungen auf).
        - Kann normalerweise viele Verbindungen zu vielen Clients gleichzeitig bedienen.
    - Der Client (Webbrowser, Mailprogramm, ...):
        - Sitzt meist am Arbeitsplatz.
        - Wird vom Benutzer nur bei Bedarf gestartet.
        - Baut bei Bedarf eine Verbindung zu einem Server auf und holt Daten von dort (oder schickt welche hin).
        - Bleibt normalerweise nicht ständig verbunden.
        - Hat im Normalfall zu einem Zeitpunkt nur eine Verbindung offen.
- *Peer-To-Peer-Kommunikation:* Das Gegenteil von Client-Server-Kommunikation: Hier kommunizieren gleichartige und gleichberechtigte Partner miteinander, d.h. jeder kann mit jedem eine Verbindung aufbauen, jeder spielt abwechselnd oder gleichzeitig Client und Server.

Netzwerkkonfiguration

MAC-Adresse ("Physikalische Adresse")

- Eine pro Anschluss / pro Netzwerk-Karte.
- MAC-Adresse = Media Access Control-Adresse, d.h. Adresse für den Zugriff auf das Kabel / das lokale WLAN: Schicht 2
- Eine MAC-Adresse besteht bei Ethernet aus 6 Bytes.
- Diese werden als 6 mal 2 Hexziffern mit ':' oder '-' dazwischen angeschrieben.
- Normalerweise sind sie fix auf der Karte eingebrannt (und stehen irgendwo auf einem Aufkleber), bei fast allen modernen Karten können sie umkonfiguriert werden.
- Die vorderen 3 Bytes geben den Hersteller der Netzwerkkarte an.
- Die "alles 1"-Adresse "ff:ff:ff:ff:ff:ff" ist die Broadcast-Adresse ("an alle").

  Auch sonst sind einige Adressen für Spezialfälle oder Schicht-2-Protokolle reserviert.

- Normalerweise braucht man sich um MAC-Adressen nicht zu kümmern / nichts konfigurieren.
- Auslesen der eigenen MAC-Adresse in Windows mit `ipconfig /all` im DOS-Fenster, Setzen bei der Treibereinstellung im Gerätemanager.
- Weitere Windows-Befehle: `getmac`, `arp` (zeigt alle dem lokalen Rechner bekannten MAC-Adressen an und kann sie – z.B. wenn ein Rechner ausgetauscht wurde – auch löschen).
- Zuordnung von MAC-Adressen zu IP-Adressen und umgekehrt:
  Automatisch über Protokoll ARP
  ("Welche MAC-Adresse gehört zur IP-Adresse *xxx*?")

IP-Adresse

- Pro Netzwerk-Karte, in Sonderfällen auch mehrere pro Karte.
- Schicht 3.
- International verwaltet:
  Weltweit durch IANA ==› Verwaltung pro Kontinent (in Europa: RIPE)
  ==› Firma / Provider
- Bei IPv4: 4 Bytes, geschrieben als 4 Dezimalzahlen 0-255 mit '.' dazwischen.
  (==› 4 Milliarden Adressen, inzwischen alle vergeben!)
- Bei IPv6: 16 Bytes, geschrieben als 8*4 Hexziffern mit ':' dazwischen.
- Konfiguration:
  - Entweder automatisch beim Rechner-Start mittels DHCP (ein zentraler Rechner am Netz vergibt die Adressen und verteilt die Netzwerk-Einstellungen): "Adresse automatisch beziehen" einstellen.

    Achtung: DHCP-Adressen werden meist nicht "auf ewig" zugeteilt, sondern müssen regelmäßig erneuert werden ("Lease bis …").

    `ipconfig` erlaubt es auch, eine neue Konfiguration via DHCP anzufordern.
  - Oder manuell mit den Windows-Netzwerkeinstellungen.
- Anzeige mit `ipconfig /all` im DOS-Fenster, Anzeige / Änderung in den Windows-Netzwerkeinstellungen.

Weitere Netzwerk-Einstellungen

- **Die Subnetz-Maske:**

  Welcher Teil der IP-Adresse ist die Adresse des eigenen Netzes (vorne, 1-Bit in der Subnetz-Maske ==› für alle Geräte in diesem Netz gleich), welcher Teil ist die Geräte-Adresse innerhalb des Subnetzes (hinten, 0-Bit in der Subnetz-Maske)?

  Teilung oft Byte-weise (8 / 16 / 24 Bits), aber jede Aufteilung ist möglich.

  Bei den Geräte-Adressen:
  Alles 1 (also z.B. 123.111.15.255) ==› Reserviert: Die Broadcast-Adresse ("an alle")
  (teilweise auch 255.255.255.255)
  Alles 0 (also z.B. 123.111.15.0) ==› Eigentlich als "Netzadresse" reserviert
  (funktioniert oft auch als normale Geräte-Adresse, aber besser freilassen…)

==› 2 Geräte weniger als prinzipiell Adressen möglich sind.

Beispiel:
255.255.255.0 ==› 24 Bit Netz, 8 Bit Gerät ==› max. 254 Geräte
255.255.255.252 ==› 30 Bit Netz, 2 Bit Gerät ==› max. 2 Geräte ==› Punkt-zu-Punkt-Verbindung

Oft auch Notation mit /n, wobei n die Anzahl der Bits der Netz-Adresse ist:
Netz 54.0.0.0/8 = 54.xxx.xxx.xxx mit Netzmaske 255.0.0.0
Netz 123.111.15.0/24 = 123.111.15.xxx mit Netzmaske 255.255.255.0

- **Das Gateway:**

  Geräte mit gleicher IP-Netzwerk-Adresse wie das eigene Gerät gehören zum eigenen Subnetz ==› Daten an diese Geräte werden direkt an das Ziel geschickt.

  Geräte mit anderer IP-Netzwerk-Adresse als das eigene Gerät sind nicht im selben Subnetz ==› sind nicht direkt erreichbar ==› Daten werden zur Weiterleitung an die Gateway-Adresse geschickt (oft Geräte-Adresse .1 oder .254, meist ein Router).

  Das Gateway muss immer eine Adresse im eigenen Subnetz haben!!!

- **Die DNS-Server:**

  Einstellung für das DNS-Protokoll: Liste der Server, die zur Namensauflösung (Zuordnung Rechnername ==› IP-Adresse) befragt werden. Kommt später!

Reservierte IP-Adressen

- 127.0.0.0/8 “Loopback-Netz”: Software-Netz innerhalb des eigenen Gerätes.

  Sonderfall: 127.0.0.1 “localhost”:
  Immer “ich selbst” (eigenes Gerät), ohne dass Daten auf die Leitung gehen.

- 224.0.0.0/4, 240.0.0.0/4 (d.h. alle Adressen ab 224.0.0.0):
  Reserviert für Broadcast / Multicast

- 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16: Private Netze

  Nur innerhalb einer Firma /Uni / ..., nicht am Internet sichtbar / weitergeleitet ==› Vielfach benutzt!

- 169.254.0.0/16 “Local Link”:

  Default-Adresse wenn nichts konfiguriert ist und DHCP auch nicht funktioniert (Zeroconf-Konfiguration). Privat.

- 192.0.2.0/24 “example.net”:

  Test- und Beispiel-Netz für Bücher, Demo’s, ... Privat.

- 0.0.0.0/8 “Lokales Netz” (reserviert, aber so gut wie nie verwendet)

Befehle

- `ping` (“Habe ich eine Verbindung zu einem Rechner?”)
- `traceroute` (unter Windows: `tracert` oder `pathping`) (“Welchen Weg gehen die Daten zu einem Rechner?”)
- `netstat` (verschiedenste Informationen: Welche Netzverbindungen sind gerade offen? Welche Programme gehören dazu? Wie ist das Routing konfiguriert?

Statistiken zum Netz?)

Routing

Das Routing ist eine Grundfunktion jedes Betriebssystems und die Hauptaufgabe jedes Routers. Es trifft zwei Entscheidungen:

- Wenn ein Gerät zwei oder mehr aktive Netzverbindungen hat (z.B. Ethernet + WLAN + virtuellen Netzadapter zu einer VM): Auf welcher Netzverbindung wird ein Paket hinausgeschickt?
- Wenn ein Paket eine Zieladresse hat, die in keinem der Netze liegt, mit denen der Rechner direkt verbunden ist: Zu welcher Adresse wird das Paket weitergeschickt?

Im einfachsten Fall (1 Netzverbindung) wird das Routing automatisch bzw. per DHCP konfiguriert:

- Alles, was mit 127 anfängt, kommt auf den "Loopback"-Pseudo-Netzadapter.
- Alles, was im eigenen Subnetz liegt, wird direkt an das Ziel geschickt.
- Alles andere wird an das Gateway geschickt.

Im Fall mehrerer Netzverbindungen (die verschiedene IP-Adressen in verschiedenen Subnetzen haben) wird das Netz automatisch so konfiguriert, dass jeder Adapter genau die Pakete für Rechner in "seinem" Subnetz bekommt (und der, in dessen Subnetz das Gateway liegt, den Rest).

Wenn mehrere Netzverbindungen existieren und in mehreren der dazugehören Subnetze ein Gateway mit anderen dahinterliegenden Netzen existiert, muss man händisch konfigurieren, welches Gateway für welche fremden Netze zuständig ist (und welches die Pakete für alle nicht explizit zugeordneten Adressen bekommt).

Der Windows-Befehl zum Anzeigen der Routing-Tabelle lautet "`route print`" oder "`netstat -r`", mit "`route` …" können auch händisch Routen für bestimmte Geräte oder Subnetze eingetragen oder gelöscht werden.

Das Routing trifft nicht nur eigene (auf diesem Rechner entstandene) Pakete, sondern (falls der Rechner in irgendeinem der angeschlossenen Netze als Gateway dient) auch Pakete "auf der Durchreise" zwischen zwei der angeschlossenen Netze. Je nachdem, welche Rolle des PC's im Netzwerk im Windows konfiguriert ist, gibt es 3 Möglichkeiten:

- Windows blockiert bzw. ignoriert alle Pakete, die an den Rechner geschickt werden, aber nicht für den eigenen Rechner, sondern ein anderes Netz bestimmt sind (siehe "`IP-Routing aktiviert`" in "`ipconfig /all`").
- Windows leitet auch fremde Pakete gemäß seiner Routing-Einstellungen weiter.
- Windows leitet die Pakete weiter und macht zusätzlich NAT (kommt später).

Bei großen Providern werden die Routing-Tabellen nicht per Hand fix eingegeben. Erstens wären sie zu groß und kompliziert (tausende Einträge), und zweitens ändert sich das optimale Routing laufend je nach Last auf den Backbone-Leitungen und eventuellen Leitungsausfällen.

Hier gibt es mehrere Mechanismen:

- Geroutet wird nicht nach Subnetzen, sondern nach den sogenannten AS-

Nummern: Jeder Provider hat eine AS-Nummer. Jede Ziel-Adresse wird einer AS-Nummer (dem Provider, in dessen Netz das Zielgerät liegt) zugeordnet. Jeder zentrale Router eines Providers weiß, was der kürzeste Weg zu einer bestimmten AS-Nummer (zum Netz eines anderen Providers) ist.

- Die zentralen Router tauschen über das Protokoll BGP laufend automatisch Informationen über Routen (bzw. über Leitungsausfälle usw.) aus.
- Zu einem Ziel gibt es normalerweise mehrere Routen mit unterschiedlicher Priorität, die Priorität wird dynamisch angepasst.

Beim Routing (vor allem beim dynamischen Routing) kann es passieren, dass Pakete im Kreis geschickt werden. Damit dadurch nicht immer mehr Pakete ewig rotieren und das Netz verstopfen, gibt es den TTL-Mechanismus ("Time to live"):

- Jedes Paket bekommt beim Wegschicken eine "Lebenszeit", je nach Systemeinstellungen z.B. 60 oder 255. Das ist keine Zeit, sondern die maximale Anzahl der Zwischenknoten bis zum Ziel.
- Auf jedem Router wird die TTL eins runtergezählt. Ist sie 0, wird das Paket verworfen und eine Fehlermeldung zurückgeschickt.

Wenn er die TTL des Senders kennt, kann der Empfänger daher die Anzahl der Router ermitteln. Auch `traceroute` und `pathping` nutzen diesen Mechanismus: Sie schicken bewusst Pakete mit zu kurzer TTL (1, 2, ...) und schauen, von welchem Zwischenknoten die Fehlermeldung kommt.

Das DNS (Domain Name System)

Das DNS ist eines der zentralen Mechanismen des Internet. Sein Hauptzweck ist die Zuordnung von Netzwerkadressen (z.B. 193.99.144.85) zu Netzwerknamen (z.B. www.heise.de) und umgekehrt. Daneben beantwortet es auch Fragen wie "Welcher DNS-Server ist für einen Namen zuständig?" und "An welchen Mail-Server gehört Mail für eine Domäne geschickt?" (das sogenannte MX-Record).

Das DNS besteht aus zwei Komponenten:

- Einem Stück Code im Betriebssystem oder zentralen Libraries, dem "Resolver", das von jedem Programm aufgerufen wird, das einen Namen auflösen muss, und eine entsprechende Anfrage ins Netz schickt.
- Zehntausenden sogenannten DNS-Servern am Internet, die bei der Beantwortung der Fragen zusammenspielen, und zwar unter oberster Kontrolle der sogenannten Root-Server.

Eine Abfrage eines Namens verläuft wie folgt:

- Wenn der Resolver die Adresse dazu nicht von einer vorigen Anfrage zwischengespeichert hat oder im `hosts`-File findet, fragt er den ersten in den lokalen Internet-Einstellungen konfigurierten DNS-Server.

  Sollte dieser nach einer bestimmten Zeit nicht oder negativ antworten, wird der zweite, dritte, ... Server befragt.
- Der befrage Server schaut
  - ob er selbst für die gesuchte Domäne verantwortlich ist, d.h. selbst das

Original der Konfigurationsdaten für diese Domäne verwaltet. Wenn ja, gibt er eine "autoritative" Antwort.

Wenn er selbst für die Domäne verantwortlich ist, aber der gefragte Name in einer Subdomäne davon liegt, für die bei ihm ein eigener Nameserver konfiguriert ist, holt er sich die "autoritative" Antwort von dort.

- oder ob er die Antwort zufällig zwischengespeichert hat. Wenn ja, gibt er eine "nichtautoritative" Antwort, d.h. eine aus zweiter Hand für eine fremde Domäne, die veraltert oder gefälscht sein könnte.

- Ist beides nicht der Fall, hat der DNS-Server zwei Möglichkeiten:
  - Er leitet die Anfrage unverändert an die für ihn konfigurierten übergeordneten DNS-Server weiter, und das Spiel wiederholt sich dort.
  - Er fragt einen Root-Server, was der für die gesuchte Toplevel-Domain zuständige Server ist, fragt diesen nach dem zuständigen Nameserver für die Second Level Domain, usw., bis er den Nameserver herausgefunden hat, der für den gesuchten Namen "autoritativ" ist.

    Von diesem bekommt er dann die endgültige Antwort.

Lokale Namen und Adressen, die nicht im DNS registriert sind, können händisch in die `hosts`-Datei eingetragen werden (Unterverzeichnis `drivers\etc` im Windows-System-Verzeichnis), diese Datei wird durchsucht, bevor das DNS befragt wird.

Unter Windows kann man mit dem Befehl `nslookup` selbst Namensabfragen an einen beliebigen Nameserver schicken und die Antwort anschauen. Mit "`ipconfig /displaydns`" kann man die zwischengespeicherten Namen und Adressen anschauen, mit "`ipconfig /flushdns`" löschen, damit sie beim nächsten Versuch frisch aus dem Netz geholt werden.

Bisher gab es folgende Top-Level-Domains:

- Die 6 Domains aus den Gründungszeiten des Internet:
  - `.com`: Firmen, weltweit ("Kommerziell")
  - `.edu`: US-amerikanische Universitäten und Bildungseinrichtungen
  - `.org`: Non-Profit-Organisationen und -Projekte, freie Software, ..., weltweit
  - `.net`: Provider, Netzwerk-Infrastruktur, ..., weltweit
  - `.mil`: US-amerikanische militärische Organisationen
  - `.gov`: US-amerikanische Regierungsstellen
- Alle UN-Länderkürzel (plus `.eu`) für das jeweilige Land, wobei einige Länder (z.B. `.to` oder `.tv`) ihre Domains weltweit vermarkten.

Seit wenigen Jahren werden immer mehr neue Top-Level-Domains eingeführt, eine der ersten war `.info`.

Die Portnummern

Bis jetzt wissen wir nur, wie Daten zwischen zwei Rechnern ihren Weg finden, aber nicht, wie man die richtige Anwendung (Web, Mail, ...) auf einem Rechner anspricht, bzw.

wie ein Rechner die einzelnen Verbindungen (zu Web, Fileserver, ...) unterscheiden und auseinanderhalten kann bzw. die zusammengehörigen Pakete einer Verbindung erkennt.

Das geschieht über die Ports. Ports sind in diesem Fall ein reines Software-Konzept, keine Hardware: Es sind 16-Bit-Zahlen, also 0 bis 65535.

Zu jeder Verbindung gehören neben einer Quell- und einer Zieladresse auch ein Quell- und ein Zielport, diese bleiben konstant, solange die Verbindung besteht. Sie werden auch in jedem Paket mitgeschickt und erlauben die eindeutige Zuordnung des Paketes zu einer Verbindung.

Jedem Dienst bzw. Server ist ein fixer Port zugeordnet, eine Liste aller offiziell registrierten Internet-Dienste findet man unter Unix unter `/etc/services` oder am Internet. Beispiele:

- SMTP (Mailversand): 25
- FTP: 21
- SSH / Putty (sichere Remote Logins): 22
- HTTP (WWW): 80
- HTTPS (verschlüsseltes WWW): 443
- DNS: 53
- POP3 (Mail empfangen): 110
- Microsoft File- und Print-Sharing, Domänencontroller usw.: 137-139, 445

Auf diesem Port horcht das zuständige Serverprogramm ständig auf einlangende Verbindungswünsche. Man sieht das im `netstat -a` : Genau jene Zeilen, bei denen als Status LISTEN steht, entsprechen Servern, die auf Verbindungen warten.

Man kann ein Serverprogramm natürlich auch auf einem anderen Port horchen lassen, aber dann muss man auch dem Client sagen, auf welchen Port er sich verbinden muss, um den Server zu finden. Wichtig ist das, wenn mehrere Server desselben Typs auf einem Rechner laufen sollen (da ja pro Port nur ein Programm horchen kann): Der erste bekommt den Standard-Port, die weiteren müssen auf anderen Ports gestartet werden.

Bei WWW sind neben 80 z.B. 88 und 8080 weit verbreitet. Sie müssen dann in der URL nach dem Hostnamen mit : angegeben werden, z.B.

`http://www2.corepower.com:8080/~relfaq/relativity.html`

Ein Verbindungsaufbau läuft wie folgt ab:

- Das Clientprogramm sagt dem Betriebssystem, zu welchem Port auf welchem Host es eine Verbindung haben möchte.
- Das Betriebssystem sucht irgendeine gerade unbenutzte lokale Portnummer und reserviert sie für diese Verbindung.
- Das Betriebssystem schickt ein Verbindungs-Anfrage-Paket mit dem soeben gewählten Quellport und dem vom Client gewünschten Zielport an die angegebene Adresse.
- Wenn dort ein Server horcht, kommt eine Verbindung zu Stande, und die Zielmaschine schickt ein Antwort-Paket mit vertauschtem Quell- und Zielport (d.h.

dem vom System für diese Clientverbindung reservierten Port als Zielport) zurück.

Ausgehende Verbindungen (Clients) haben daher an ihrem Ende einen Port für sich allein, ankommende Verbindungen können mehrere denselben Zielport (den des jeweiligen Serverprogramms) haben.

Versucht sich ein Client auf einen Port zu verbinden, auf dem kein Server horcht, weist das Betriebssystem des Zielrechners die Verbindung mit "Connection refused" zurück.

Verbindet sich ein Client mit einem falschen Server (d.h. sitzt z.B. auf Port 80 gemeinerweise ein Mailserver und kein Webserver), so kommt zwar auf TCP/IP-Ebene eine Verbindung zu Stande (TCP/IP "weiß" ja nicht, um welchen Dienst es sich handelt), aber die beiden Enden werden einander "nicht verstehen", d.h. mit den Daten, die das Gegenüber schickt, nichts anfangen können: Entweder macht ein Ende die Verbindung als Reaktion auf die falschen Daten wieder zu, oder beide warten "ewig", dass richtige Daten kommen.

Wenn man das Protokoll des Servers kennt, kann man allerdings mit einem Telnet-Client (der die eingetippten Daten unverändert auf das Netz schickt und die empfangenen Daten unverändert anzeigt) oder dem Tool `netcat` einen richtigen Client vortäuschen und so z.B. eine Mail händisch verschicken oder eine Webseite abrufen.

Das Tool `nmap` klopft eine fremde Maschine auf Server ab, indem es einfach Verbindungen auf alle Ports durchprobiert. Seine Verwendung ist in den meisten Ländern illegal.

Die Geschichte des Internet

- Die Forschungs-Firma BBN (Bolt/Beranek/Newman) verband im Auftrag der ARPA (der amerikanischen Forschungs-Förderungs-Behörde) am 1. Oktober 1969 die ersten beiden Computer an kalifornischen Universitäten zum ARPAnet, dem Vorläufer des heutigen Internets.
- In den 70/80-er-Jahren erfolgte der erste große Entwichlungsschub des Internet, begünstigt durch folgende Faktoren:
    - Die ARPA erhob es zu ihrem Standard: Wer an ARPA-Forschungsprojekten teilnehmen wollte, war zum ARPAnet und zu TCP/IP verpflichtet.
    - TCP/IP war kostenlos und patentfrei nutzbar, es war herstellerunabhängig und für alle Rechner und Betriebssysteme verfügbar, und es gab mit BSD Unix eine im Source verfügbare Referenz-Implementierung.
    - In den amerikanischen Universitäten begannen Rechner mit Unix als Betriebssystem direkt am Arbeitsplatz populär zu werden, und Unix nutzte als primäres Netzwerk-Protokoll TCP/IP. Rasch entstanden große lokale Netzwerke auf Basis Ethernet und TCP/IP.
- Der zweite große Entwicklungsschub kam Mitte der 90-er-Jahre mit dem WWW (1990 von Tim Berners-Lee am CERN erfunden): Das WWW brachte das Internet in Privathaushalte und Firmen.

Organisationen, Normen usw.

Die Verwaltung des Internet ist ziemlich aufgesplittet:

- TCP- und Internet-Normen heißen RFC (mit einer Nummer), z.B. RFC 5321 (und seine Vorgänger RFC 821, RFC 974, RFC 1869 und RFC 2821) für das Mail-Protokoll SMTP.

  Sie werden von der IETF (Internet Engineering Task Force) gemacht.
- Daten-Inhalte sind getrennt standardisiert: HTML und andere Web-Inhalte werden z.B. vom W3C (World Wide Web Consortium) entwickelt.
- Die Nummern-Vergabe (IP-Adressen, AS-Nummern für das Routing, Port-Nummern für Dienste) werden von der IANA (Internet Assigned Numbers Authority) verwaltet und zugeteilt. Sie verwaltet auch die DNS Rootserver und genehmigt neue Toplevel-Domains.

  Die Über-Organisation der IANA ist die ICANN, eine non-Profit-Firma mit starkem US-Einfluß (beauftragt vom US-Wirtschaftsministerium, das früher das Internet selbst leitete).
- IP-Adressen werden von der IANA an eine Unterorganisation pro Kontinent vergeben, das ist in Europa die RIPE NCC und in Nordamerika die ARIN. Diese wiederum verkaufen sie über mehrere Ebenen an Firmen und Provider weiter.
- Bei Domain-Namen gibt es pro Toplevel-Domain einen Verwalter, der die Namen dieser Domäne vergibt. Das kann eine Behörde des jeweiligen Landes oder eine private Firma sein, in Deutschland z.B. die DENIC, eine Genossenschaft von Internet-Firmen. .com und .net werden von der Firma Verisign betrieben,
- Der technische Teil des Internets (Leitungen, Netzknoten, ...) wird großteils von privaten Firmen betrieben und ist fast durchwegs profitorientiert.
- Ebenso in privater Hand (Marktführer Verisign) ist die Vergabe und Verwaltung von Sicherheitszertifikaten für verschlüsselte Verbindungen (HTTPS usw.).

Firewalls

Firewalls haben den Zweck, nur bestimmten Netzverkehr durchzulassen, entweder – bei Firmen – zwischen zwei Netzen (also zwischen dem firmeneigenen Netz und dem Internet, die "Firewall" ist hier ein eigener Rechner mit Spezial-Software) oder zwischen dem Netz und den Programmen am eigenen Rechner (beim Privat-PC, hier ist die Firewall nur ein Stück Software im Windows).

Es gibt 3 Arten von Firewalls:

- **Paketfilter**.

  Das ist der einfachste und häufigste Typ. Hier schaut sich die Firewall bei jedem einzelnen Paket in einer langen Liste von Regeln nach, ob sie das Paket durchlassen oder blockieren soll. Für die Entscheidung spielt nur der TCP/IP-Header des Paketes eine Rolle, nicht die Nutzdaten (d.h. eine Paketfilter-Firewall weiß nicht, ob das Paket eine Mail, eine Webseite oder eine Videodatei enthält, und sie kann auch nicht auf Viren usw. prüfen).

  Kriterien sind u.a.:

  - Quell- und Zieladresse
  - Quell- und Zielport

- Verbindungs-Status (gehört das Paket zu einer schon offenen Verbindung oder ist es eine neue Verbindungs-Anfrage?)
- Netzwerk-Interface (kam das Paket vom vertrauenswürdigen hausinternen Ethernet oder von der WLAN-Karte).
- Unter Windows auch: Programm, das das Paket sendet / empfängt.

Beispiele für Regeln sind z.B.:

- Pakete von draußen dürfen nur rein, wenn sie zu einer schon offenen Verbindung gehören. Neue Verbindungsanfragen werden blockiert.
- DNS-Pakete müssen vom eigenen Rechner kommen oder zum eigenen Rechner gehen, sie werden nicht durchgeroutet.
- Hinaus dürfen nur Pakete mit HTTP- oder HTTPS-Zielport (von jedem internen Rechner), sowie Pakete mit Ports der Mail-Protokolle (nur vom Mailserver).
- Pakete von der Internet-Netzwerkkarte mit einer privaten Quell-Adresse werden jedenfalls verworfen.

- **Proxies, Application Level Firewalls, Gateways, ...:**

  Das sind eigene Programme, die zusätzlich zur Paketfilter-Firewall meist am selben Rechner wie die Firewall laufen und nur für ein ganz bestimmtes Anwendungs-Protokoll zuständig sind, z.B. der Webproxy für HTTP, ein Mail-Proxy für SMTP und POP, oder ein DNS-Proxy für DNS-Anfragen.

  Die Paketfilter-Firewall ist in diesem Fall so eingestellt, dass sie alle Pakete dieses Protokolls von drinnen nach draußen und umgekehrt sperrt und nur Pakete von drinnen zum Proxy und von draußen zum Proxy und umgekehrt zulässt.

  Der gesamte Verkehr dieses Protokolls läuft also über das Proxy-Programm. Man muss die Anwendungen auf den privaten PC's (z.B. den Webbrowser, den Mail-Client oder den DNS-Resolver) explizit so einstellen, dass sie ihre Anfragen nur an den Proxy schicken und nicht direkt zum Ziel.

  Der Proxy schaut sich dann die Anfrage genau an, schickt von sich aus eine entsprechende Anfrage ins Internet, bekommt die Antwort, analysiert sie auf Unbedenklichkeit, und schickt sie an den Client weiter.

  Da jeder Proxy nur für eine Art von Anwendung zuständig ist und das Protokoll dieser Anwendung (z.B. HTTP) genau kennt, kann er auch den Inhalt der Pakete anschauen und z.B. Viren in Mails oder bösartige oder unerwünschte Inhalte in Webseiten filtern.

- **Socks-Server:**

  Das Socks-Protokoll und der auf der Firewall laufende Socks-Server erlauben internen PC's, Netzwerk-Verbindungen direkt von der Firewall ins Internet zu machen, die Firewall spielt sozusagen "Vorposten" für Netzwerkverbindungen von internen PC's.

  Dieser Typ ist selten, wir behandeln ihn nicht weiter.

NAT (Network Address Translation, d.h. Netzwerk-Adress-Übersetzung)

Im Internet herrscht Adress-Knappheit, es gibt viel mehr Rechner als offizielle Adressen. Viele Firmen wollen aus Sicherheitsgründen auch gar nicht, dass alle ihre internen Rechner am Internet sichtbar sind.

- Die meisten Firmennetze verwenden daher intern die privaten Adressbereiche 10.0.0.0/8, 172.16.0.0/12 oder 192.168.0.0/16. Die einzigen öffentlichen Adressen der Firma haben normalerweise die Firewall, der Webserver und der Mailserver (falls diese nicht ohnehin bei einem Provider stehen).
- Bei allen Internet-Providern bekommt ein Privatkunde ebenfalls nur eine einzige öffentliche Adresse. Hängt er mit einem einzigen PC direkt am ADSL- oder Handy-Modem, so bekommt der PC eine öffentliche Adresse.

  Hängt hingegen ein Router oder ein WLAN-Router am ADSL, so bekommt der Router die einzige öffentliche Adresse, und alle Geräte im hauseigenen Ethernet oder WLAN bekommen vom eigenen Router mittels DHCP eine private Adresse aus den Netzen 10.0.0.0/8 oder 192.168.0.0/16.

Private Adressen sind aber am Internet ungültig, Pakete mit einer privaten Zieladresse werden nicht weitergeroutet, sondern sofort gefiltert bzw. verworfen. Selbst wenn der Router oder die Firewall daher die Pakete aus dem privaten Netz ins Internet weiterleiten würde, würden die Antwort-Pakete nicht zum privaten PC zurückfinden.

Daher macht der (WLAN-) Router oder die Firewall (oder was immer für ein Gerät zwischen privatem Netz und Internet steht) "heimlich" (ohne dass der PC am Ende der Verbindung etwas davon weiß oder merkt) Folgendes:

- Bei jedem Paket aus dem privaten Netz ins Internet ersetzt er die (private) Quell-Adresse (z.B. 10.10.10.123) durch seine eigene (öffentliche) Adresse und den Quell-Port (z.B. 4242) durch einen freien Port auf seiner eigenen Maschine (z.B. 12345).
- Dann schickt er das so modifizierte Paket ins Internet. Es schaut also für den Empfänger so aus, als ob das Paket nicht vom privaten PC, sondern vom Router käme, der Router ist das einzig sichtbare Gerät am Internet. Der Webserver, Mailserver oder was immer wird seine Antwort daher an die Adresse des Routers und den vom Router eingesetzten Quellport schicken.
- Gleichzeitig merkt sich der Router intern in einer Tabelle für jede gerade offene Verbindung von drinnen nach draußen, welche seiner Portnummern er für welche Verbindung von welchem privaten PC verwendet hat, z.B. "mein Port 12345 gehört zur Verbindung vom internen PC 10.10.10.123, Port 4242".
- Bekommt der Router jetzt ein Antwortpaket vom Internet (z.B. auf seinen Port 12345), schaut er nach, zu welcher Verbindung es gehört, ersetzt in der Zieladresse seine Adresse durch die interne Adresse des privaten PC's (10.10.10.123) und im Zielport seinen Port durch den ursprünglichen Port (4242), und schickt das Paket ins private Netz weiter.
- Für den PC im privaten Netz schaut es so aus, als ob er ganz normal und direkt eine Antwort vom Internet bekommen hätte.