

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Einführung in die Informatik: Systeme und Anwendungen

Kurzsriptum zur Vorlesung

Sommersemester 2002

F. Kröger

INSTITUT FÜR INFORMATIK

Lehr- und Forschungseinheit für Programmierung und Softwaretechnik

Inhaltsverzeichnis

Einleitung	3
1 Betriebssysteme	5
1.1 Rechner und Programme	5
1.2 Prozesse	8
1.3 Systemprogrammierung	12
1.4 Speicherverwaltung	15
2 Datenbank- und Informationssysteme	18
2.1 Grundbegriffe	18
2.2 Die Anfragesprache SQL	19
2.3 Das relationale Datenmodell	23
2.4 Datenbankentwurf	24
2.5 Konstruktion von Informationssystemen	27
3 Rechnernetze	32
3.1 Verteilte Systeme	32
3.2 Netzwerk-Strukturen	33
3.3 Kommunikation in Rechnernetzen	35
3.4 Das Internet	37

Einleitung

Für wen ist diese Lehrveranstaltung gedacht?

Neben-/(Pflicht-)Wahl-/Spezialfach-Hörer der Studiengänge

- BWL
- Biologie
- Chemie
- Geographie
- Geophysik
- Journalistik
- (alle) Magisterstudiengänge
- Mathematik - Studienrichtung Wirt.math. u. Aktuarwiss.
- Mathematik - Lehramt (nicht vertieft)
- Meteorologie
- Mineralogie
- Physik (Grundstudium)
- Psychologie
- Soziologie
- Statistik
- VWL
- Wirtschaftsgeographie
- Wirtschaftspädagogik
- Seniorenstudium
- Studium Generale

Aber: nicht geeignet für Studenten folgender Studiengänge

- Informatik
- Bioinformatik
- Medieninformatik
- Mathematik (andere Studienrichtungen)

Inhaltsübersicht

- Betriebssysteme
- Datenbank- und Informationssysteme
- Rechnernetze

Literaturhinweise

H.-P. Gumm, M. Sommer: *Einführung in die Informatik*.
4. Auflage. Oldenbourg-Verlag 2000.

A. Heuer, G. Saake, K. Sattler: *Datenbanken kompakt*.
MITP 2001.

H.-J. Siegert, U. Baumgarten: *Betriebssysteme*.
4. Auflage. Oldenbourg-Verlag 1998.

L. Peterson, B. Davie: *Computernetze - Ein modernes Lehrbuch*.
dpunkt.verlag 2000.

Organisatorisches

- Vorlesung: Mi 14-16, HS 225.
- Übungen:
 - Zentralübung: Mi 16-17, HS 225.
 - Übungsgruppen: Do 17-19 und 19-20.30 (Tutorium),
Fr 15-17 und 17-18.30 (Tutorium).
- Anmeldung für Rechnerkennungen: siehe Merkblatt.
- Leistungsnachweis:
 - je nach Regelung für die einzelnen Studiengänge gefordert (als Schein oder Teilprüfung) oder freiwillig möglich;
 - erwerbbar durch Bestehen der abschließenden Klausur;
 - Zulassungsvoraussetzungen zur Klausur:
- Informationen zur Vorlesung/Übung unter:
<http://www.pst.informatik.uni-muenchen.de/lehre/SS02/infoeinf/>
- Informationen zum Rechnerbetrieb in der Oettingenstr. 67:
<http://www.rz.informatik.uni-muenchen.de/>

Kapitel 1

Betriebssysteme

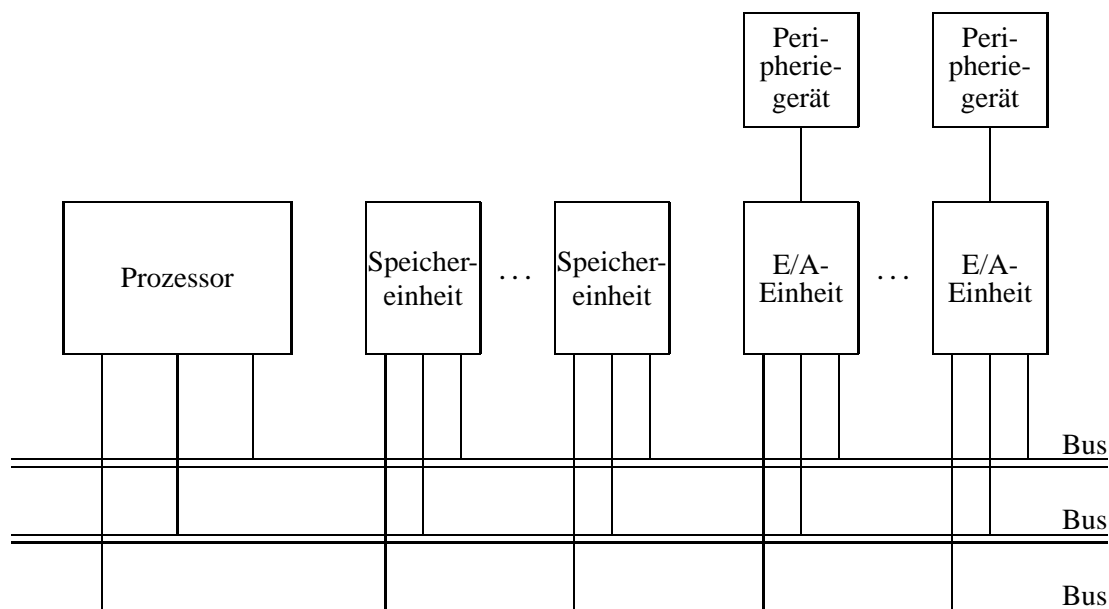
1.1 Rechner und Programme

Rechner

Der grundsätzliche Aufbau heutiger Rechner folgt der *von-Neumann-Architektur* (seit etwa 1950). Die Grundbestandteile (*Hardwarekomponenten*) sind:

1. Die *CPU (Central Processing Unit, Prozessor, Rechnerkern)* führt *Befehle* aus und steuert die zugehörigen Abläufe.
2. Im *Speicher* werden Daten und Programme abgelegt.
3. Über *Ein/Ausgabe-Einheiten* (mit angeschlossenen *Peripheriegeräten*) können Daten und Programme ein- und ausgegeben werden.
4. *Busse* sind Verbindungen zwischen diesen Einheiten zur Informationsübertragung.

Schema der von-Neumann-Architektur



Speicher

Der Speicher ist in (mindestens) zwei „Schichten“ organisiert:

- Im (relativ kleinen) **Hauptspeicher (Arbeitsspeicher)** werden die Daten und Programme gehalten, die gerade ausgeführt und benötigt werden.
- Der (relativ große) **Hintergrundspeicher** dient zur Aufnahme von Daten und Programmen, auf die relativ selten zugegriffen werden muss.

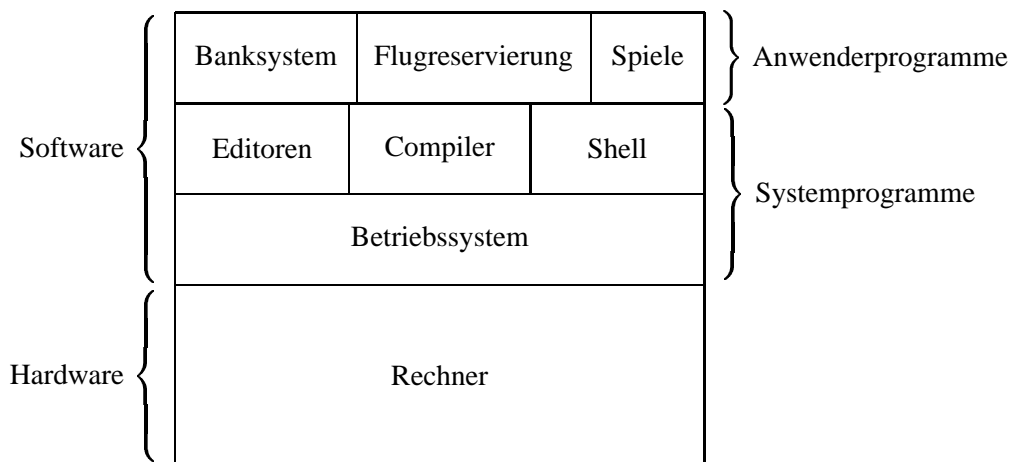
In beiden Speicherbereichen gibt es im wesentlichen zwei Speichertypen:

- **RAM (Random Access Memory, Schreib/Lese-Speicher):**
Speicherinhalte können gelesen und verändert werden.
- **ROM (Read Only Memory, Festwertspeicher):**
Speicherinhalte können gelesen, aber nicht verändert werden.

Ausführung von Programmen auf Rechnern

Auf einem Rechner sollen Programme von Benutzern (**Anwenderprogramme**) ausgeführt werden. Dazu sind **Systemprogramme** nötig, die solche Ausführungen realisieren. Einen wesentlichen Teil der Systemprogramme bildet das **Betriebssystem** (BS).

Gesamtaufbau eines Computersystems:



Wesentliche Aufgaben eines Betriebssystems

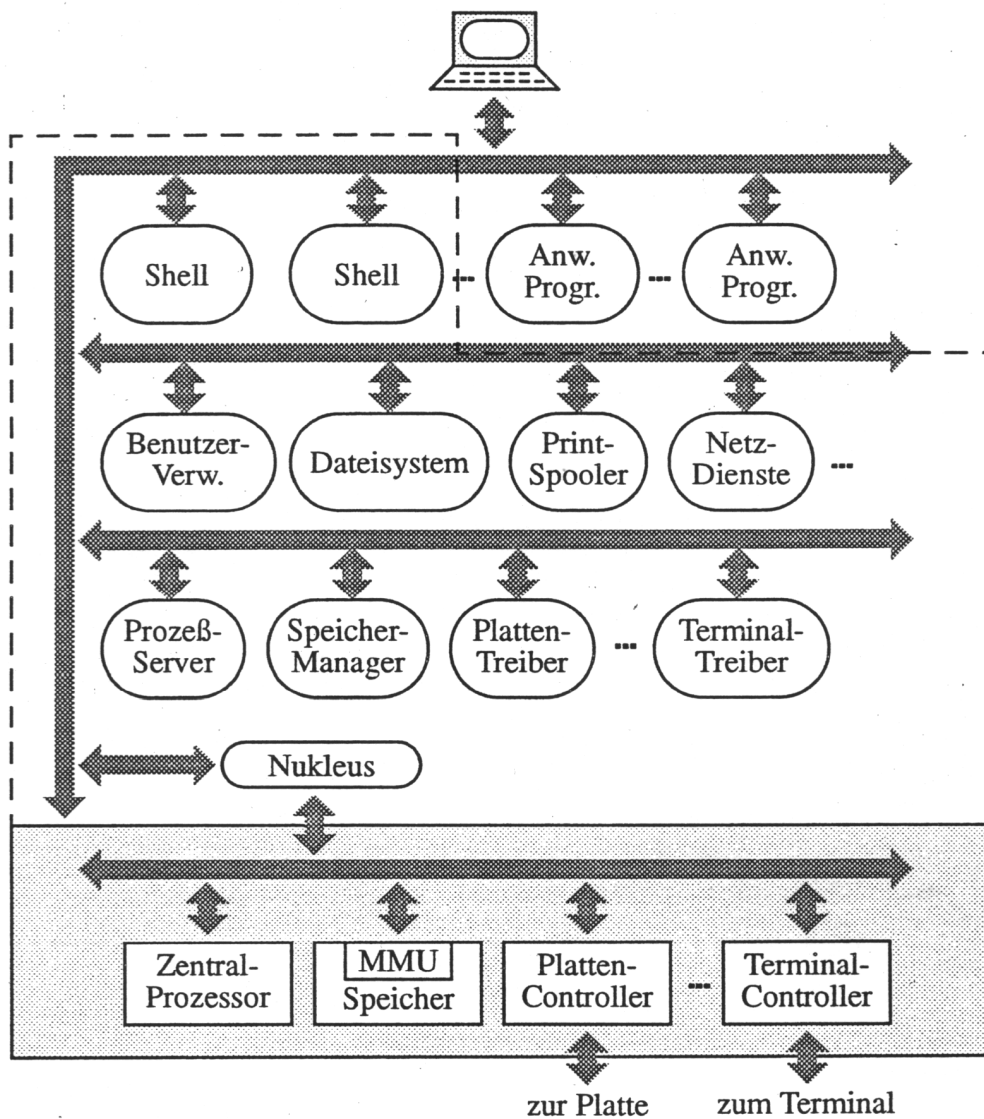
Das Betriebssystem bildet die Schnittstelle für Anwenderprogramme und spezielle Systemprogramme zur Hardware und dient zur Steuerung und Verwaltung von Computersystemen.

Einzelne Aufgaben:

- Prozessverwaltung:
Steuerung der Ausführung eines oder mehrerer Prozesse, insbesondere im Mehrprogrammbetrieb.
- Speicherverwaltung (für den Hauptspeicher)
- Dateiverwaltung (im Hintergrundspeicher)
- Verwaltung der peripheren Geräte

Schichtung des Betriebssystems

Das Betriebssystem ist in verschiedenen *Abstraktionsschichten* organisiert. Höhere Schichten schirmen von den Details unterer Schichten (z.B. speziellen Hardwareeigenschaften) ab.



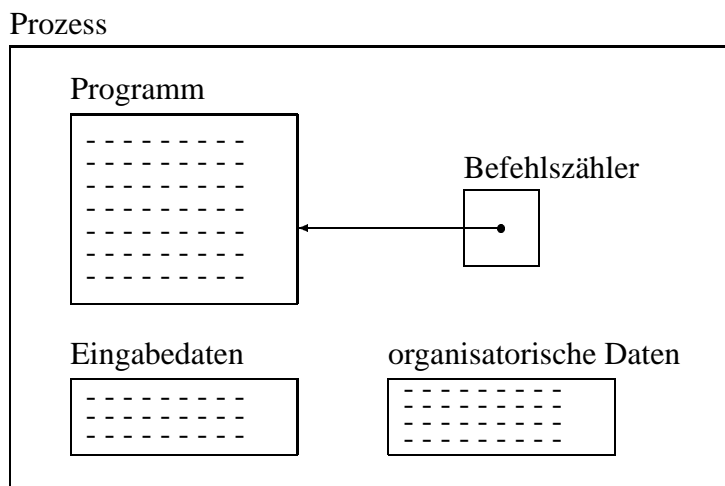
Einige bekannte Betriebssysteme

- Windows 98 für Personal Computer.
- Apple OS X für Personal Computer.
- Windows NT, Windows 2000 und Windows XP für Personal Computer und Workstations.
- Linux für Personal Computer und Workstations.
- Unix-Varianten wie Solaris, HP/UX und Irix für Workstations und viele andere Arten von Rechner.
- MVS, VM/SP, CMS, BS 2000 für Großrechner (Mainframes).

1.2 Prozesse

Das Prozesskonzept

- Auf einem Rechner laufen „gleichzeitig“ verschiedene (Anwender- und System-) Programme.
- Ein **Prozess** ist ein in Ausführung befindliches Programm. Er enthält neben dem Programmtext die Eingabedaten, einen Befehlszähler (der den als nächstes auszuführenden Befehl bestimmt) und weitere organisatorische Informationen:



Parallelität von Prozessen

- (Zumindest) Anwenderprozesse und Systemprozesse sowie Systemprozesse untereinander müssen abwechselnd auf dem Prozessor laufen (dem Prozessor **zugeschrieben** sein).

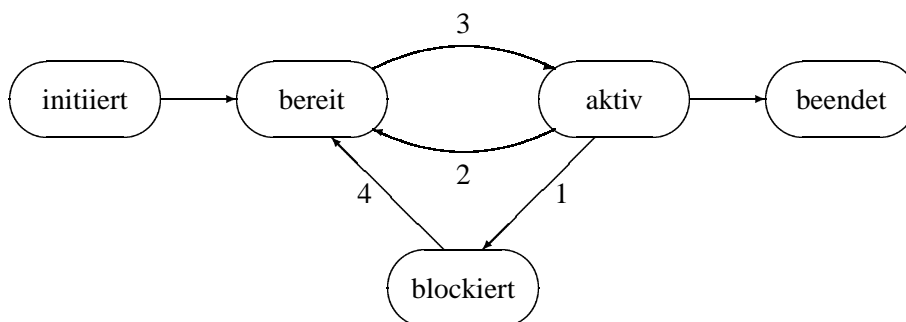
- Auch Anwenderprozesse sollten (in der Regel) dem Prozessor abwechselnd zugeteilt sein (*Mehrprogrammbetrieb, Multiprogramming*).
- Die einzelnen Prozesse durchlaufen somit (zumeist) ihre Anweisungsfolge nicht in einem Schritt. Sie werden häufig unterbrochen. „Nach außen“ entsteht der Eindruck von *Parallelität* aller auf dem Rechner existierenden Prozesse (*Quasi-Parallelität*).
- Das Umschalten des Prozessors zwischen mehreren Prozessen bedingt, dass die zeitliche Dauer eines Prozesses bei verschiedenen Programmausführungen verschieden sein kann.

Prozesszustände

Ein Prozess kann sich in fünf verschiedenen Zuständen befinden:

1. **initiiert:**
Der Prozess ist erzeugt, aber noch nicht ausführbar.
2. **bereit:**
Der Prozess ist ausführbar, aber ein anderer Prozess ist dem Prozessor zugeteilt.
3. **aktiv:**
Der Prozess ist dem Prozessor zugeteilt und wird gerade von diesem ausgeführt.
4. **blockiert:**
Der Prozess kann gerade nicht ausgeführt werden (z.B. weil Eingabedaten fehlen oder weil ein benötigtes Betriebsmittel belegt ist).
5. **beendet:**
Der Prozess ist mit seiner Ausführung am Ende.

Zustandsübergänge



- 1: Prozess kann nicht mehr weiterarbeiten (z.B. weil Eingabe fehlt, oder benötigtes Betriebsmittel belegt ist).
- 2: Ein anderer Prozess wird dem Prozessor zugeteilt.
- 3: Der Prozess wird dem Prozessor zugeteilt.
- 4: Ein „externes“ Ereignis tritt ein, so dass der Prozess wieder lauffähig wird.

Verklemmungen

Eine **Verklemmung** (*deadlock*) ist eine Situation, in der alle Prozesse einer gewissen Prozessmenge blockiert sind.

Beispiel:

- Zwei Prozesse *A* und *B*;
- *A* wartet auf Eingabe von *B*;
- *B* wartet auf Eingabe von *A*.

Prozess-Scheduling

Die Zuteilung eines Prozesses an den Prozessor erfolgt durch den **Scheduler** (Teil des Nukleus). Zuteilungsverfahren sind:

1. "Round-Robin" - Verfahren
(einfach, häufig benutzt)
Bereite Prozesse werden in einer Warteschlange mit First-In-First-Out (FIFO) - Prinzip organisiert. Jedem Prozess wird eine (i.a. gleich große) **Zeitscheibe** zum Rechnen zugeteilt. Die Warteschlange wird zyklisch abgearbeitet.
2. Scheduling nach Prioritäten
Jedem Prozess ist eine Priorität zugeordnet. Der Prozess mit der höchsten Priorität wird aktiviert (Problem: Fairness).
3. Shortest-Job-First - Strategie
Der Prozess mit der kürzesten Laufzeit wird zuerst aktiviert.

Prozess-Synchronisation

Häufig sind parallel ablaufende Prozesse von einander abhängig. Um ein fehlerhaftes Verhalten zu verhindern, sind die Prozesse geeignet zu koordinieren (**Synchronisation der Prozesse**).

Synchronisations-Grundmuster

Es gibt zwei Grundarten von Synchronisationsanforderungen (die auch in Mischformen vorkommen können):

- (a) **Wechselseitiger Ausschluss**
In Prozessen gibt es **kritische Abschnitte**. Die Operationsfolgen kritischer Abschnitte verschiedener Prozesse dürfen nicht gleichzeitig ausgeführt werden (d.h. während der Ausführung eines kritischen Abschnitts darf der Prozess nicht für kritische Abschnitte anderer Prozesse unterbrochen werden).
- (b) **Prozesskooperation**
Prozesse arbeiten zusammen. Die Ausführung von Operationen in einem Prozess setzt voraus, dass gewisse Operationen in anderen Prozessen erledigt sind.

Wechselseitiger Ausschluss: Beispiel

Flugreservierung:

Parallel ablaufende Prozesse reservieren jeweils eine bestimmte Anzahl von Plätzen. Eine Variable *anzpl* gibt zu jedem Zeitpunkt die Anzahl der noch verfügbaren Plätze an. Jeder Reservierungsprozess verfährt nach folgendem Schema (*n* sei die Anzahl der zu reservierenden Plätze):

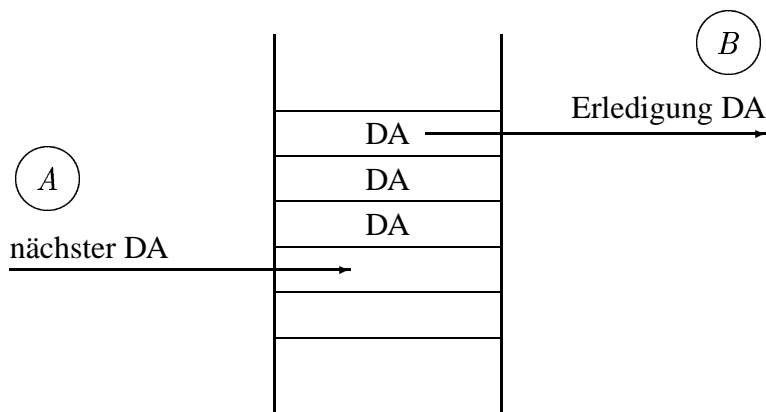
- ⋮
- (a) Durch Inspektion von *anzpl* wird bestimmt, ob noch *n* freie Plätze vorhanden sind;
- (b) Falls noch *n* Plätze frei sind, wird *anzpl* um *n* verringert; andernfalls STOP mit Auskunft *Ausgebucht*;
- (c) Reservierung wird bestätigt.
- ⋮

Kritischer Abschnitt: Operationsfolge (a);(b).

Prozesskooperation: Beispiel

Verwaltung von Druckaufträgen in einer Druckauftragsliste DL:

Ein Prozess *A* setzt in gewissen Abständen Druckaufträge DA in DL ab. Ein Prozess *B* organisiert die betreffenden Druckvorgänge der Reihe nach. *A* und *B* laufen parallel zueinander ab.



Notwendige Synchronisation (**Erzeuger-Verbraucher-Schema**):

Ist DL leer (d.h. *B* hat alle DA in DL erledigt („verbraucht“) und *A* hat noch keinen neuen DA abgesetzt („erzeugt“)), so muss *B* angehalten werden.

1.3 Systemprogrammierung

Beispiel: Programmierung des wechselseitigen Ausschlusses

Problemstellung (2 Prozesse):

Prozess P1	Prozess P2
...	...
< kritischer Bereich >;	< kritischer Bereich >;
...	...

P1 und *P2* laufen parallel ab. Die beiden Programme sind so zu ergänzen, dass die beiden kritischen Bereiche wechselseitig ausgeschlossen sind.

1. Lösungsversuch

Prozess P1	Prozess P2
...	...
WHILE dran <> 1	WHILE dran <> 2
DO {nichts};	DO {nichts};
< kritischer Bereich >;	< kritischer Bereich >;
dran := 2;	dran := 1;
...	...

Diese Lösung ist nicht befriedigend:

- Die Prozesse können nur abwechselnd ihren jeweiligen kritischen Bereich betreten.
- Hält ein Prozess (außerhalb seines kritischen Bereichs) an, so kann der andere nur noch einmal in seinen kritischen Bereich eintreten und wird danach bei einem nochmaligen Versuch, den kritischen Bereich zu betreten, blockiert.

2. Lösungsversuch

Prozess P1	Prozess P2
...	...
anmeld1 := true;	anmeld2 := true;
WHILE anmeld2 DO {nichts};	WHILE anmeld1 DO {nichts};
< kritischer Bereich >;	< kritischer Bereich >;
anmeld1 := false;	anmeld2 := false;
...	...

Diese Lösung vermeidet die Nachteile des 1. Versuchs, aber:

- Die Prozesse können in eine Verklemmung geraten.

Lösung

Prozess P1	Prozess P2
...	...
anmeld1 := true;	anmeld2 := true;
vortritt := 2;	vortritt := 1;
WHILE	WHILE
anmeld2 AND vortritt = 2	anmeld1 AND vortritt = 1
DO {nichts};	DO {nichts};
< kritischer Bereich >;	< kritischer Bereich >;
anmeld1 := false;	anmeld2 := false;
...	...

Semaphore

Synchronisationsaufgaben können mit einem neuen algorithmischen Konzept eleganter programmiert werden:

Ein *Semaphor* s ist eine Variable, die nur nicht-negative ganze Zahlen $0,1,2,\dots$ als Werte aufnehmen kann und auf der folgende 3 Operationen ausgeführt werden können:

- Initialisierung von s mit einem nicht-negativen Wert.
- Die Operation P mit der Wirkung:
Ist $s > 0$, so wird der Wert von s um 1 vermindert. Ist $s = 0$, so wird der ausführende Prozess an dieser Stelle blockiert und in eine Menge W_s von (bezüglich s wartenden) Prozessen eingereiht.
- Die Operation V mit der Wirkung:
Ist W_s nicht leer, so wird einer der in W_s enthaltenen Prozesse aus W_s entfernt und ist damit nicht mehr blockiert (d.h.: bereit). Ist W_s leer, so wird der Wert von s um 1 erhöht.

(Die Ausführung dieser Operationen darf nicht unterbrochen werden.)

Semaphorlösung für wechselseitigen Ausschluss

	semaphore ex;	
	ex := 1;	
Prozess P1		Prozess P2
...		...
P(ex);		P(ex);
< kritischer Bereich >;		< kritischer Bereich >;
V(ex);		V(ex);
...		...

Allgemein für beliebig viele Prozesse:

1 Semaphore ex ,

Initialisierung: $ex := 1$;

Jeder Prozess hat die Gestalt

```
...
P(ex);
< kritischer Bereich >;
V(ex);
...
```

DL-Verwaltung mit Semaphoren

```
semaphore inDL;
inDL := 0;
```

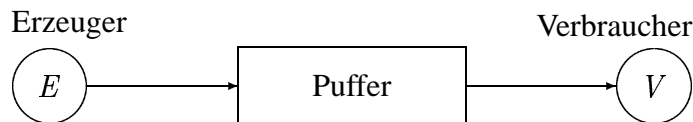
Prozess A

Prozess B

```
...
< DA in DL ablegen >;
V(inDL);
...
```

```
...
P(inDL);
< DA erledigen >;
...
```

Allgemeineres Erzeuger-Verbraucher-Schema



E legt von Zeit zu Zeit jeweils 1 Objekt im Puffer ab („Füllen“). V holt von Zeit zu Zeit jeweils 1 Objekt aus dem Puffer („Leeren“). Der Puffer ist beschränkt mit einer Aufnahmekapazität von max Objekten.

Synchronisationsbedingungen:

- Der Erzeuger darf den Puffer nur füllen, wenn er nicht voll ist.
- Der Verbraucher darf den Puffer nur leeren, wenn er nicht leer ist.

Semaphorlösung:

2 Semaphore (Anzahl der Elemente im Puffer, Anzahl der freien Plätze im Puffer);

Jede „Füllen“-Operation von E und jede „Leeren“-Operation von V wird mit Semaphore-Operationen versehen:

```

        semaphore inPuffer;
        semaphore frei;
        inPuffer := 0;
        frei := max;

Erzeuger                                Verbraucher

...
P(frei);                                ...
< Füllen >;                             P(inPuffer);
V(inPuffer);                             < Leeren >;
...                                       V(frei);
...                                       ...

```

1.4 Speicherverwaltung

Aufgabe des Betriebssystems

(Mehrprozessbetrieb)

- Belegung des Hauptspeichers mit Programmen und Daten
- Freigabe nicht mehr benötigten Speicherplatzes.

Starre Segmentierung

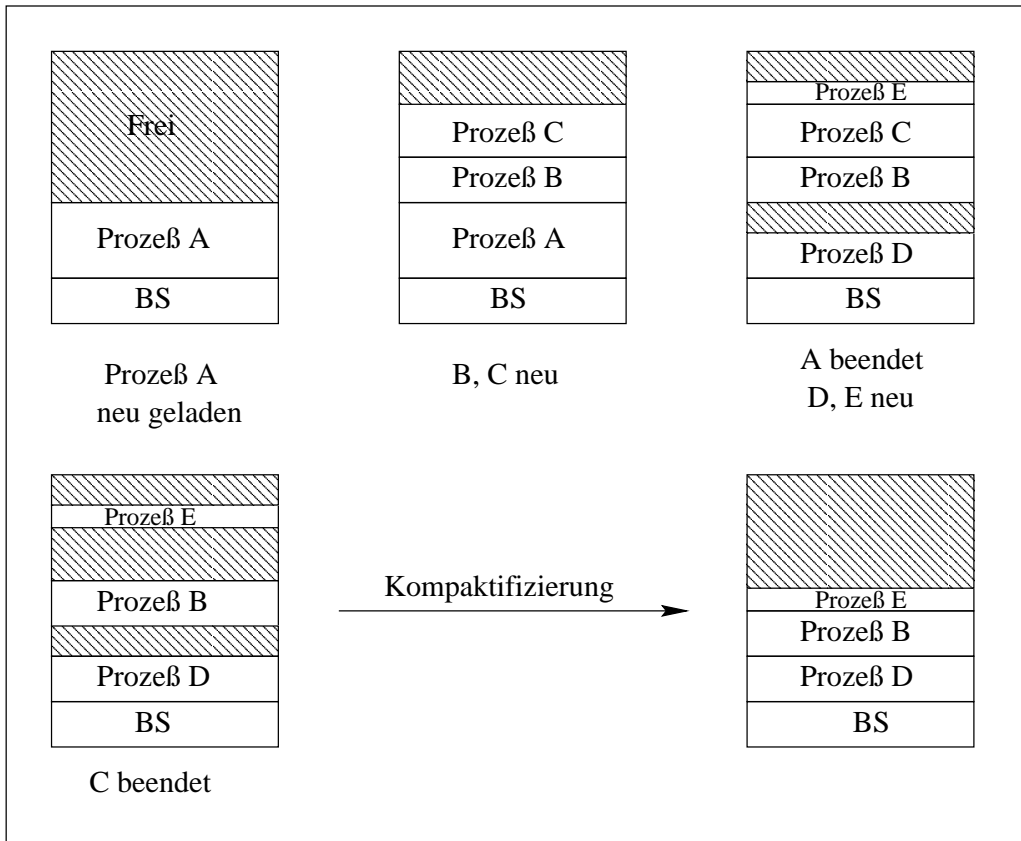
Der Speicher wird in gleich große *Segmente* unterteilt. Jedes Segment enthält einen Prozess (oder ist frei).

Nachteil:

Schlechte Ausnutzung des Speichers, da Prozesse im allgemeinen verschieden groß sind.

Variable Segmentierung

Die Größe eines Segments wird an die Prozessgröße angepasst.



Um eine zu große Zerstückelung des Speichers zu vermeiden, wird zwischendurch der freie Teil des Speichers zusammengefasst (*Kompaktifizierung*).

Speicherzuteilungsverfahren

Freie Segmente werden in einer Frei-Liste verwaltet. Mögliche Strategien der Zuteilung:

- **First-fit**
Das erste freie Segment, das den Speicheranforderungen des Prozesses entspricht, wird zugeteilt.
- **Best-fit**
Das kleinste freie Segment, das den Speicheranforderungen des Prozesses entspricht, wird zugeteilt.

Virtueller Speicher und Paging

Hauptspeicher und Hintergrundspeicher werden zu einem *virtuellen Speicher* zusammengefasst. Der *logische Adressraum* eines Programms kann dadurch größer sein, als der tatsächlich vorhandene (Haupt-) Speicherplatz.

Vorteil:

Viele Prozesse können gleichzeitig bearbeitet werden, auch wenn die Summe ihrer Speicheranforderungen größer ist als der Hauptspeicher.

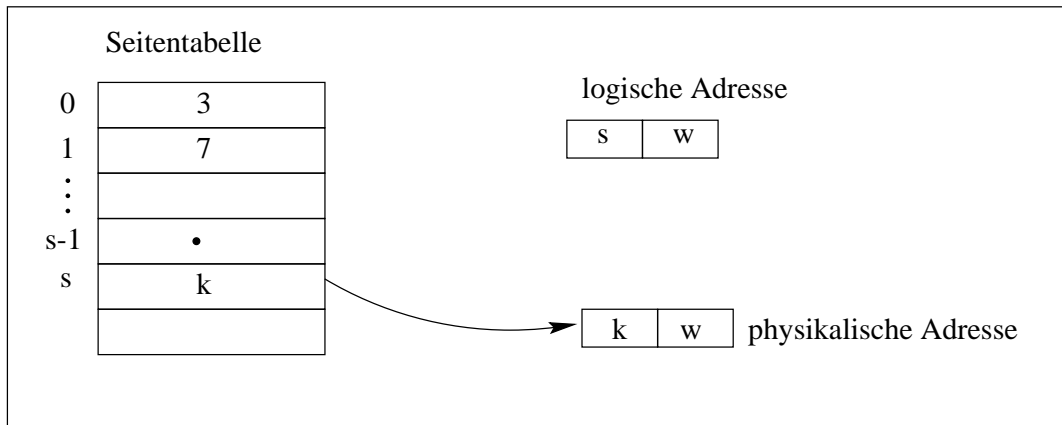
Der logische Adressraum eines Programms wird in *Seiten* fester Größe eingeteilt, die in den Hauptspeicher ein- und ausgelagert werden können (*Paging*). Der Hauptspeicher wird in *Kacheln* eingeteilt, die gleich groß sind wie die Seiten.

Logische Adressierung

Eine Speicherzelle wird „angesteuert“ durch:

- eine Seitennummer s
- eine relative Adresse w , die die Lage der Speicherzelle auf der gegebenen Seite angibt.

Die Zuordnung der Kacheln zu den Seiten erfolgt in einer *Seitentabelle*.



Wird während der Prozessausführung eine Seite benötigt, der keine Kachel im Hauptspeicher zugeordnet ist, so kommt es zu einem *Seitenfehler*. Die fehlende Seite wird dann in den Hauptspeicher geladen und eine andere Seite wird verdrängt (d.h. auf den Hintergrundspeicher ausgelagert).

Verdrängungsstrategien

Muss eine Seite aus dem Hauptspeicher verdrängt werden, so wird typischerweise nach einer der folgenden Strategien verfahren. Man entfernt

- die Seite, die am längsten im Hauptspeicher ist (FIFO);
- die am längsten nicht benützte Seite (LRU, *least recently used*);
- die am seltensten benützte Seite (LFU, *least frequently used*).

Kapitel 2

Datenbank- und Informationssysteme

2.1 Grundbegriffe

Funktion von Datenbanksystemen

Datenbanksysteme (DBS) dienen dazu, große Mengen von Daten zentral und dauerhaft (persistent) zu speichern und zu verwalten. Ein DBS besteht aus

- dem Datenbankmanagementsystem (DBMS, Programm im Hauptspeicher),
- einer oder mehreren Datenbanken (DB, i.a. im Hintergrundspeicher).

Typische Operationen auf Datenbanken:

- Erzeugen und Löschen von DB
- Einfügen, Ändern und Löschen von Daten in DB
- *Anfragen* (zum Auffinden von Daten in einer DB)

Charakteristika von Datenbanksystemen

- Zentralisierung der Information
 - wenig Redundanz
 - Vermeidung von Inkonsistenzen
- Benutzerfreundliche Formulierung von Anfragen
- (Gleichzeitige) Benutzung durch mehrere Programme oder Anwender; Synchronisation paralleler Zugriffe
- Sicherheitsvorkehrungen:
 - Datenschutz: Verhinderung unberechtigter Zugriffe
 - Recovery bei Systemausfällen (Anlegen von Sicherheitskopien)
- Integritätskontrollen
- Datenunabhängigkeit:
 - Die konkrete Datendarstellung im Rechner ist dem Benutzer verborgen.
 - Ihre Änderung erfordert keine Änderung der Anwendungsprogramme

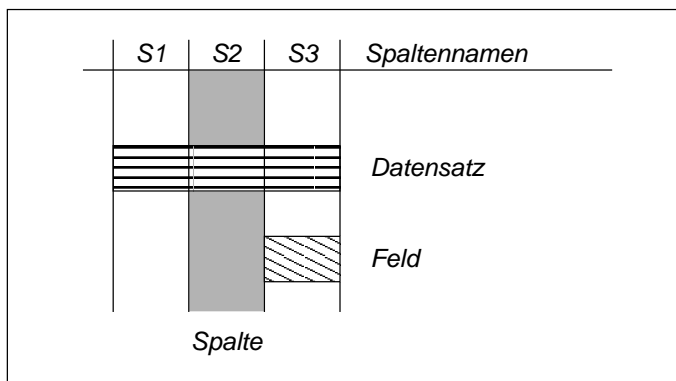
Historische Entwicklung

- Anfang 60er Jahre: elementare Dateien, anwendungsspezifische Datenorganisation
→ geräteabhängig, redundant, inkonsistent
- Ende 60er Jahre: Dateiverwaltungssysteme mit Dienstprogrammen (z.B.: Sortieren)
→ geräteunabhängig, aber redundant und inkonsistent
- ab 70er Jahre: Datenbanksysteme
→ geräte- und datenunabhängig, redundanzfrei, konsistent
- insbesondere (80er Jahre): relationale Datenbanksysteme
→ mächtige Anfragesprachen (SQL),
einfaches und präzises mengentheoretisches Modell
- seit 1990:
 - objektorientierte Datenbanksysteme (für komplexe Daten und ihre Beziehungen; Vererbungshierarchien)
 - deduktive Datenbanksysteme (aus gespeicherten Grundfakten können logische Schlüsse berechnet werden)
 - verteilte Datenbanksysteme
 - Datenbanksysteme für spezielle Anwendungen, wie z.B. Geographie, Biologie, Chemie, Musik

2.2 Die Anfragesprache SQL

Relationale Datenbanken

Eine relationale DB ist eine endliche Menge von Tabellen. Eine Tabelle ist eine endliche Menge von **Datensätzen** (auch Zeilen oder Tupel genannt):



Beispiel:

Weinkeller

Nr	Wein	Lieferant	Jahr	Flaschen	Kommentar
2	Chardonnay	Buena Vista	83	1	
3	Chardonnay	Louis Martini	81	5	
6	Chardonnay	Chappellet	82	4	trocken
11	Jo. Riesling	Jekel	84	10	
12	Jo. Riesling	Buena Vista	82	1	Spätlese
16	Jo. Riesling	Sattui	82	1	sehr trocken
21	Fume Blanc	Ch. St. Jean	79	4	Napa Valley
22	Fume Blanc	Robt. Mondavi	78	2	
25	Wh. Burgundy	Mirassou	80	6	
30	Gewürztraminer	Buena Vista	80	3	
43	Cab. Sauvignon	Robt. Mondavi	77	12	
50	Pinot Noir	Mirassou	77	3	Auslese
51	Pinot Noir	Ch. St. Jean	78	2	
64	Zinfandel	Mirassou	77	9	Jubiläum
72	Gamay	Robt. Mondavi	78	2	

Merkmale von SQL

- Deskriptive Sprache: Man beschreibt die gesuchte Menge von Daten, und nicht die Prozedur zum Finden der Daten.
- Sowohl interaktiv als auch aus einem Programm heraus aufrufbar.
- Entwickelt seit 1974 bei IBM; erster Standard 1986.
- Gegliedert in:
 - DDL (Data-definition-language)
zum Erstellen und Löschen von Datenbanken, Tabellen und Sichten.
 - DML (Data-manipulate-language)
zum Einfügen von Datensätzen, Ändern, Löschen und Abfragen von Daten in Tabellen.

Grundoperationen von SQL

- Erstellen einer DB:

```
CREATE DATABASE Datenbankname ;
```

- Löschen einer DB:

```
DROP DATABASE Datenbankname ;
```

- Erstellen einer Tabelle:

```
CREATE TABLE Tabellename
        (Spaltenname Datentyp [NOT NULL],
         :
         Spaltenname Datentyp [NOT NULL]);
```

- Löschen einer Tabelle:

```
DROP TABLE Tabellename;
```

- Einfügen eines Datensatzes:

```
INSERT INTO Tabellename
        [(Spaltenname, ..., Spaltenname)]
VALUES (Wert, ..., Wert);
```

- Ändern von Werten:

```
UPDATE Tabellename
SET Spaltenname = Ausdruck,
   :
   Spaltenname = Ausdruck,
[WHERE Bedingung];
```

- Löschen von Datensätzen:

```
DELETE FROM Tabellename
[WHERE Bedingung];
```

- Anzeigen gewisser Spalten von Datensätzen:

```
SELECT Spaltenname
        [, Spaltenname, ..., Spaltenname]
FROM Tabellename
[WHERE Bedingung];
```

- Anzeigen einer gesamten Tabelle, d.h. aller Spalten und Datensätze:

```
SELECT * FROM Tabellename;
```

Weitere Sprachelemente

- Anzeigen mit Sortierfunktion:
Die angezeigten Datensätze erscheinen in der durch die Werte einer bestimmten Spalte gegebenen Reihenfolge.

- Anwendung von statistischen (*Aggregats-*) Funktionen:

SUM (Spaltenname)	Summe der Spaltenwerte
SUM (DISTINCT Spaltenname)	Summe der verschiedenen Spaltenwerte
AVG (Spaltenname)	Durchschnitt der Spaltenwerte
AVG (DISTINCT Spaltenname)	Durchschnitt der verschiedenen Spaltenwerte
MAX (Spaltenname)	maximaler Spaltenwert
MIN (Spaltenname)	minimaler Spaltenwert
COUNT (*)	Anzahl der Datensätze
COUNT (DISTINCT Spaltenname)	Anzahl der verschiedenen Datensätze in der Spalte

- Gruppierung bei statistischen Anfragen:
Anwendung statistischer Funktionen auf eine Gruppe von Datensätzen, die in einer bestimmten Spalte den gleichen Wert haben.
- Auswahl von Gruppen:
Anwendung statistischer Funktionen auf eine Gruppe von Datensätzen, die eine bestimmte Bedingung erfüllen.
- Produktanfragen:
Daten werden nicht nur aus einer Tabelle, sondern aus mehreren Tabellen ausgewählt und die Kombinationen der jeweiligen Datensätze in einer Ergebnistabelle zusammengefasst.
- *Equi-Join*:
Es werden bei einer Produktanfrage nur solche Zeilenkombinationen ausgewählt, die in bestimmten Spalten den gleichen Wert haben.
- *Natürlicher Join*:
Wie Equi-Join, jedoch werden doppelte Spalten nur einmal ausgewählt.

Unteranfragen

- Anfragen können geschachtelt werden. Dies erlaubt die Auswahl von Daten einer Tabelle in Abhängigkeit von den Werten in einer anderen Tabelle
- Schema:

```
SELECT ...
FROM Tabellename
WHERE Spaltenname IN (SELECT ...
                      :
                      );
```

Die *Unteranfrage* liefert eine Menge von Ergebniswerten. Der IN-Operator überprüft, ob der jeweilige Datenwert der genannten Spalte Teil der Menge ist. Falls ja, wird die zugehörige Zeile ausgewählt.

- Eine Unteranfrage kann eine weitere Unteranfrage enthalten.
- Unteranfragen können auch gebildet werden mit:
 - NOT IN (Negation von IN)
 - Vergleichsoperatoren

(Bei Verwendung von Vergleichsoperatoren muss die Unteranfrage genau einen Wert liefern.)

Sichten

- Mit Hilfe von Anfragen können aus einer oder mehreren Tabellen **Sichten** (*virtuelle Tabellen*) erzeugt werden. Aus diesen können Daten wie aus („realen“) Tabellen abgefragt werden.
- Schema:

```
CREATE VIEW Sichtname
AS Anfrage;
```

- Löschen von Sichten:

```
DROP VIEW Sichtname;
```

2.3 Das relationale Datenmodell

Datenmodelle

- Eine Datenbank legt eine gewisse Struktur der in ihr enthaltenen Daten und deren Beziehungen zugrunde (**Datenmodell**).
- Für relationale Datenbanken ist die Tabellenstruktur charakteristisch; diese kann mathematisch einfach und präzise beschrieben werden (**relationales Datenmodell**).

Mathematische Definition

Seien A_1, A_2, \dots, A_n Mengen. Eine **Relation** R über A_1, A_2, \dots, A_n ist eine Teilmenge des kartesischen Produkts $A_1 \times A_2 \times \dots \times A_n$:

$$R \subseteq A_1 \times A_2 \times \dots \times A_n.$$

D.h.: Ein Element $r \in R$ ist ein **Tupel** $r = (a_1, a_2, \dots, a_n) \in A_1 \times A_2 \times \dots \times A_n$.

Tabellen als Relationen

- Eine Tabelle T einer relationalen Datenbank stellt eine endliche Relation dar. Genauer: Sind S_1, \dots, S_n die Spaltennamen (*Attribute*) von T und $Dom(S_1), \dots, Dom(S_n)$ die Typen (d.h. Mengen von möglichen Werten) von S_1, \dots, S_n , so stellt T eine Relation über $Dom(S_1), \dots, Dom(S_n)$ dar. Ein Datensatz von T ist ein Element der Relation.
- Eine relationale Datenbank kann somit formal als eine endliche Menge von endlichen Relationen aufgefasst werden.

Formale Bedeutung von Datenbankoperationen

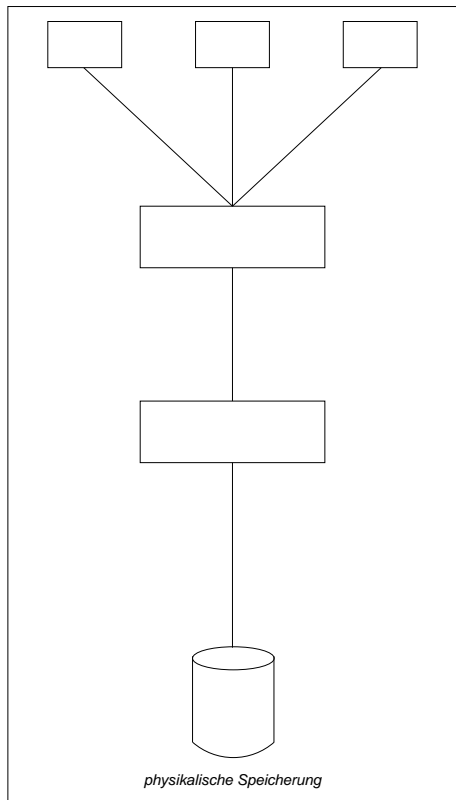
Die Bedeutung von Operationen auf Datenbanken lässt sich in diesem Modell durch Mengenoperationen präzise beschreiben. Beispiele:

Operation	Bedeutung
INSERT INTO T VALUES (w1, ..., wn)	Bildung von $T \cup \{(w1, \dots, wn)\}$
DELETE FROM T WHERE B	Bildung von $T \setminus \{z = (w1, \dots, wn) \mid z \text{ erfüllt } B\}$
SELECT DISTINCT S1, S2 FROM T	Bildung von $\{(s_1, s_2) \mid (\dots, s_1, \dots, s_2, \dots) \in T\}$ (wobei s_1 und s_2 die „S1- bzw. S2-Komponenten“ von T sind)
SELECT * FROM T1, T2	Bildung von $T1 \times T2$

2.4 Datenbankentwurf

Die 3-Ebenen-Architektur

- Erste wesentliche Anforderung beim Entwurf einer Datenbank: Alle Anwenderprogramme müssen die gleiche Darstellung der Daten benutzen.
- Zur Realisierung: Die Beschreibung der Daten einer Datenbank erfolgt auf drei verschiedenen Ebenen (**3-Ebenen-Architektur**):



- **externe Ebene**
verschiedene, individuelle Benutzersichten
- **konzeptionelle Ebene**
logische Gesamtsicht der Daten und ihrer Bezüge
- **interne Ebene**
Realisierung der Datenstrukturen in konkreter Programmiersprache

- Bemerkung: In kleinen Systemen fallen externe und konzeptionelle Ebene häufig zusammen.

Relationaler Datenbankentwurf

Problem auf konzeptioneller Ebene einer relationalen Datenbank:

- Gegeben: Große Menge von verschiedenartigen Daten.
 Fragen: Repräsentation dieser Daten?
 Welche Attribute?
 Welche Tabellen?
 Ziele: Tabellen gemäß logischer Zusammenhänge der Daten.
 Vermeidung von Redundanzen und Anomalien.

Beispiel: Redundanzen und Anomalien

Tabelle:	Lieferant	Ort	km	Weine
	Buena Vista	Napa Valley	1200	15
	Mirassou	Paris	1000	10
	Mondavi	Napa Valley	1200	7

- Redundanz:
Die Entfernung des Ortes Napa Valley wird zweimal gespeichert.

- **Änderungsanomalie:**
Bei Änderung der Entfernungsangabe von Napa Valley muss die Änderung mehrfach durchgeführt werden (Gefahr der Inkonsistenz).
- **Löschanomalie:**
Wird der Lieferant Mirassou gelöscht, so geht die Information über die Entfernung seines Standorts verloren.
- **Einfügeanomalie:**
Eine neue Zeile kann nur eingefügt werden, wenn die Werte für alle vier Attribute vorliegen. Teilinformationen (z.B. Entfernungsangaben) können nicht für sich gespeichert werden.

Schlüssel und funktionale Abhängigkeit

Ein Schlüssel besteht aus einem oder mehreren Attributen, deren Werte eine gesamte Zeile einer Tabelle eindeutig bestimmen.

Formal:

- Es seien R eine Tabelle (d.h. eine Relation) und A_1, \dots, A_n, B Attribute von R .
- **Definition:**
 B heißt **funktional abhängig** von A_1, \dots, A_n (Schreibweise: $A_1, \dots, A_n \rightarrow B$), wenn zu jedem Zeitpunkt der Wert von B eindeutig durch die Werte von A_1, \dots, A_n bestimmt ist (d.h. zu den Werten von A_1, \dots, A_n gibt es genau einen Wert von B).
- **Definition:**
 A_1, \dots, A_n heißt
 - **Superschlüssel (identifizierend)** für R , wenn jedes Attribut von R funktional abhängig von A_1, \dots, A_n ist.
 - **Schlüssel** für R , wenn A_1, \dots, A_n ein minimaler Superschlüssel für R ist (d.h. keine echte Teilmenge von A_1, \dots, A_n ist Superschlüssel für R).
- **Definition:**
Ein Attribut von R heißt **Nicht-Schlüssel-Attribut**, wenn es zu keinem Schlüssel von R gehört.
- **Definition:**
 B heißt **voll funktional abhängig** von A_1, \dots, A_n (Schreibweise $A_1, \dots, A_n \Rightarrow B$), wenn B funktional abhängig ist von A_1, \dots, A_n , aber von keiner echten Teilmenge von A_1, \dots, A_n .

Normalisierung

Redundanzen und Anomalien können dadurch vermieden werden, dass Tabellen in eine bestimmte **Normalform** gebracht werden (**Normalisierung**). Dieser Vorgang wird in den folgenden drei Schritten beschrieben.

Erste Normalform

- Definition:
Eine Relation ist in *erster Normalform* (1NF), wenn jedes Attribut elementar ist, d.h. Werte von einem Grunddatentyp (INTEGER, CHAR(N), ...) hat. (Der Wert eines Attributs darf also nicht wieder eine Relation sein.)
- Verfahren zur Konvertierung einer Relation R in 1NF:
 1. Identifiziere einen Schlüssel für R .
 2. Bilde eine Relation mit den Attributen dieses Schlüssels und allen elementaren Attributen von R .
 3. Gibt es ein nicht-elementares Attribut von R mit einer Relation R' als Wert, so identifiziere einen Schlüssel für R' und bilde eine weitere Relation mit den Attributen der Schlüssel von R und R' und den weiteren Attributen von R' .
 4. Wiederhole diesen Vorgang für jede Relation, die noch nicht-elementare Attribute besitzt.

Zweite Normalform

Definition:

Eine Relation R ist in *zweiter Normalform* (2NF), wenn R in 1NF ist und jedes Nicht-Schlüssel-Attribut voll funktional abhängig ist von jedem Schlüssel.

Beachte: Eine Relation, in der jeder Schlüssel nur aus einem Attribut besteht, ist in 2NF.

Dritte Normalform

Definition:

Eine Relation R ist in *dritter Normalform* (3NF), wenn R in 2NF ist und jedes Nicht-Schlüssel-Attribut (außer von sich selbst) nur von jedem Schlüssel voll funktional abhängig ist.

Beachte: Eine Relation mit nur einem Nicht-Schlüssel-Attribut ist in 3NF.

2.5 Konstruktion von Informationssystemen

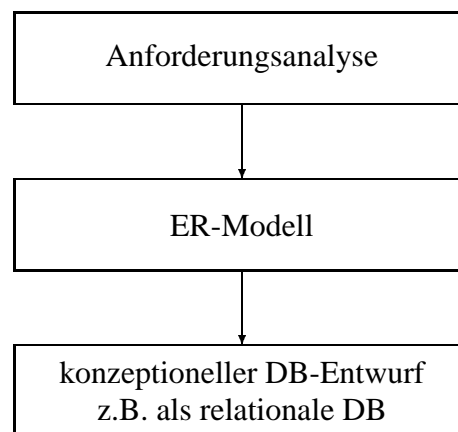
Informationssysteme

- Ein Informationssystem (IS) ist ein Softwaresystem zur Speicherung, Wiedergewinnung, Verknüpfung und Auswertung von Informationen (eines bestimmten Anwendungsbereichs).
- Ein IS besteht aus einem DBS und den Auswertungsprogrammen. Der Zugriff auf die Informationen kann direkt (über das DBMS) oder innerhalb eines Auswertungsprogramms erfolgen.

- Spezielle IS:
 - Informationsgewinnungssysteme (Information Retrieval Systems):
Hauptaufgabe ist die Speicherung und die schnelle und gezielte Bereitstellung gesuchter Informationen.
 - Dokumentationssysteme:
Informationen sind Merkmale von Dokumenten (Bücher, Zeitschriftenartikel u.ä.)
 - Managementinformationssysteme:
Informationen sind alle Größen und Kenndaten, die zur optimalen Führung eines Unternehmens notwendig und sinnvoll sind.

Das Entity-Relationship-Modell

Beim Entwurf von Informationssystemen wird typischerweise eine abstrakte Datenmodellierung verwendet, die die in dem System zu verarbeitenden Daten und ihre Beziehungen noch unabhängig von dem (später) gewählten Datenmodell des DBS darstellt (**Entity-Relationship-Modell, ER-Modell**):



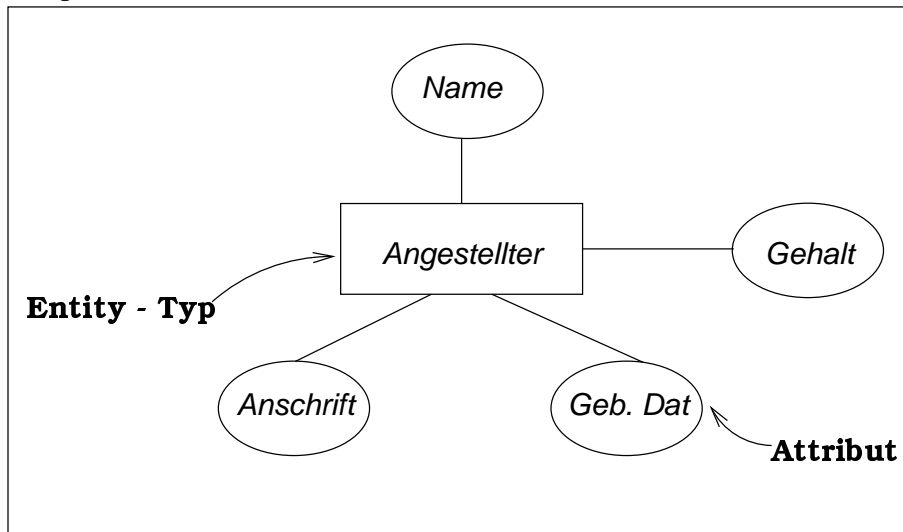
Die ER-Modellierung wird grafisch in **ER-Diagrammen** repräsentiert.

Entity-Typen

- Ein Entity-Typ beschreibt eine Klasse von Objekten (Entities), die untereinander ähnlich sind.

- Ein Entity-Typ hat bestimmte Merkmale (Attribute).

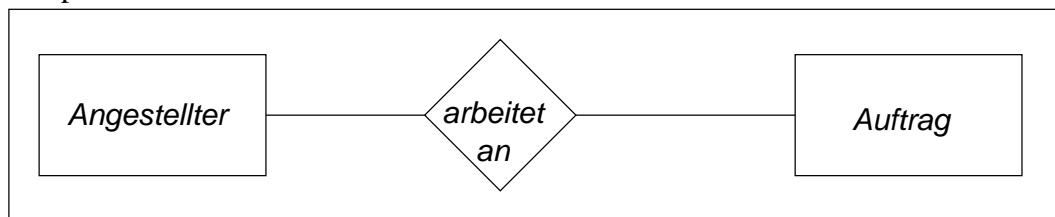
Beispiel:



Relationships

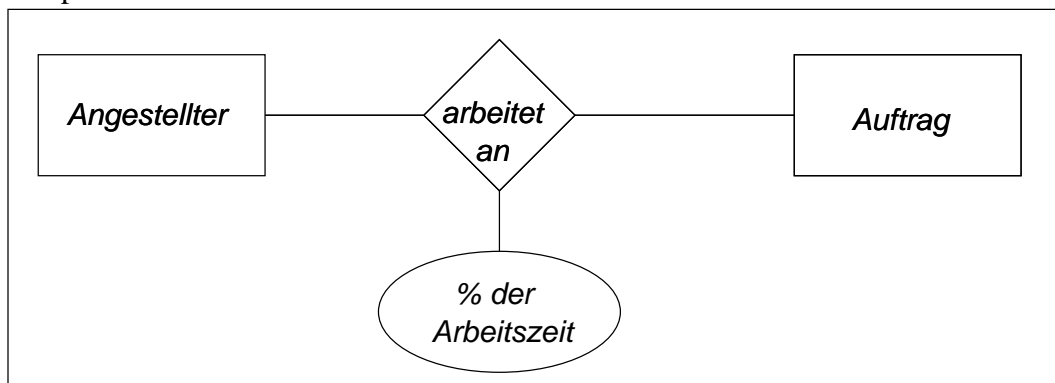
- Relationships beschreiben Beziehungen zwischen Entity-Typen.

Beispiel:



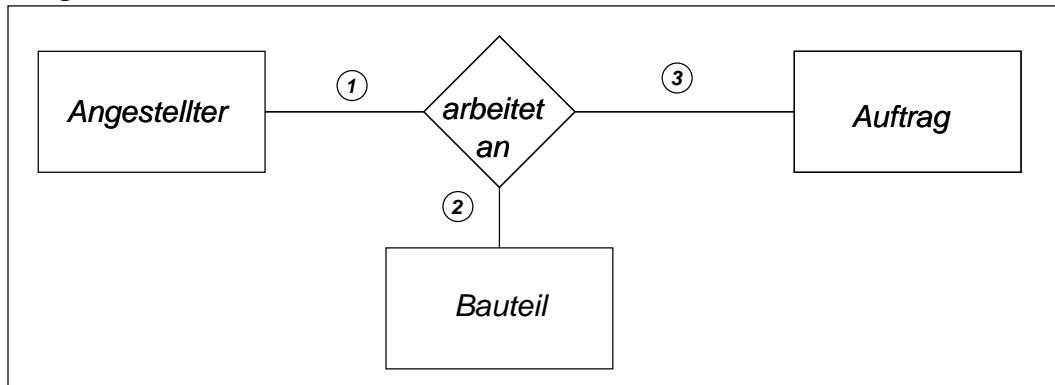
- Relationships können auch Attribute haben.

Beispiel:



- Es kann auch Beziehungen zwischen mehr als zwei Entity-Typen geben.

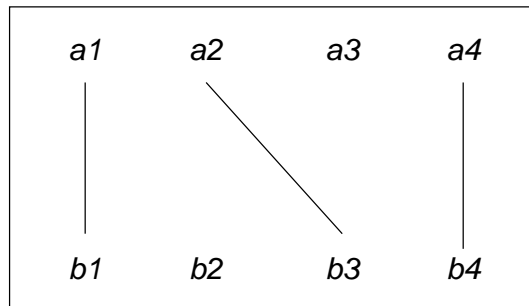
Beispiel:



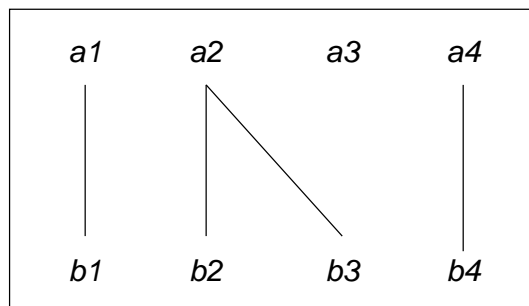
Klassifizierung von Relationships

R sei eine Beziehung zwischen zwei Entity-Typen A und B . R kann klassifiziert werden nach der Anzahl der Entities von Typ A , die mit einer Entity vom Typ B in Beziehung stehen können und umgekehrt.

1. R ist vom Typ 1:1 (*one-to-one*), wenn zu jedem Zeitpunkt jede Entity vom Typ A mit höchstens einer Entity vom Typ B in Beziehung stehen kann und umgekehrt:



2. R ist vom Typ 1:m (*one-to-many*), wenn zu jedem Zeitpunkt jede Entity vom Typ A mit $m \geq 0$ Entities vom Typ B in Beziehung stehen kann und jede Entity vom Typ B mit höchstens einer Entity vom Typ A in Beziehung stehen kann:



3. R ist vom Typ m:1 (*many-to-one*), wenn die Beziehung in umgekehrter Richtung betrachtet vom Typ 1:m ist.

4. R ist vom Typ $m:n$ (*many-to-many*), wenn zu jedem Zeitpunkt jede Entity vom Typ A mit beliebig vielen Entities vom Typ B in Beziehung stehen kann und umgekehrt.
5. R ist *verpflichtend*, wenn zu jedem Zeitpunkt jede Entity vom Typ A mit einer Entity vom Typ B in Beziehung stehen muss.

Übergang vom ER-Modell zum relationalen DB-Schema

Ein ER-Modell kann leicht in ein relationales Datenmodell konvertiert werden.

Vorgehensweise:

1. Zeichne für jeden Entity-Typ ein Attribut a aus, und bilde eine Relation mit a als Schlüssel und den übrigen Attributen.
2. Es sei R eine Beziehung zwischen zwei Entity-Typen A und B . a_1, \dots, a_j seien die Attribute von A , b_1, \dots, b_k die Attribute von B und r_1, \dots, r_l die Attribute von R . a_1 und b_1 seien die ausgezeichneten Attribute von A bzw. B .

Bilde eine Relation mit den Attributen $a_1, r_1, \dots, r_l, b_1$ und mit dem Schlüssel

$a_1, b_1,$	falls R eine $m:n$ - Beziehung ist,
$b_1,$	falls R eine $1:m$ - Beziehung ist,
$a_1,$	falls R eine $m:1$ - Beziehung ist,
a_1 oder $b_1,$	falls R eine $1:1$ - Beziehung ist.

3. Normalisierung der erhaltenen Relationen (häufig nicht mehr nötig).
4. Optimierung: Zusammenfassung der Relationen mit gleichem Schlüssel.

Anbindung relationaler Datenbanken an Java-Programme

Mit der Schnittstelle JDBC (Java DataBase Connectivity) kann von Java-Programmen aus auf SQL-Datenbanken zugegriffen werden.

Schichten einer JDBC - Anwendung:

- JDBC - Treiber
Gehört nicht zur JDBC. Ist abhängig vom benutzten DBMS und stellt die Verbindung zu diesem her. Muss zusätzlich zur JDBS beschafft und importiert werden, z.B.:

```
import.java.sql.*; // JDBC
import.imaginary.sql.*; //Treiber für MiniSQL
```
- Driver-Manager
Gehört zur JDBC. Registriert Treiber und wählt für eine gegebene Datenbankadresse (URL) einen geeigneten Treiber aus.
- Programmierschnittstelle
Vom Anwender zum Programmierer (unter Verwendung der JDBC).

Kapitel 3

Rechnernetze

3.1 Verteilte Systeme

Rechnerentwicklung

- Bis etwa 1985:
Zentrale Systeme (Großrechner, Rechenzentren): bestehend aus einem Prozessor, einem Speicher und Peripheriegeräten.
- Seit 1985:
Verteilte Systeme: mehrere Rechner, die durch ein **Rechnernetz** miteinander verbunden (**vernetzt**) sind.

Ausdehnung der Vernetzung

- Lokales Netz (LAN, *local area network*)
Verbindung mehrerer Rechner (PCs, Workstations) in einem räumlich begrenzten Gebiet (bis 5 km);
Vernetzung durch private Leitungen.
Beispiel: Netz des Instituts für Informatik
- Stadtnetze (MAN, *metropolitan area network*)
Ausdehnung über (Teil-) Gebiet einer Stadt, Gelände einer größeren Firma;
Vernetzung durch gemietete Standleitungen.
Beispiel: Münchner-Hochschul-Netz (MHN)
- Weitverkehrsnetz (WAN, *wide area network*)
Ausdehnung über ein oder mehrere Länder;
Vernetzung über öffentliche Datenübertragungssysteme (z.B. ISDN).
Beispiel: Wissenschaftsnetz (WIN)
- Globales Netz, Netz von Netzen
Weltweite Verbindung verschiedener Netze.
Beispiel: Reservierungssystem, Internet

Nutzen von verteilten Systemen

- **Datenverbund:** Nutzung von Datenbeständen, die auf einzelnen Rechnern des Netzes verstreut sind.
- **Betriebsmittelverbund:** Nutzung teurer Software und Hardware, die nicht auf jedem Rechner des Netzes bereit gehalten werden kann.

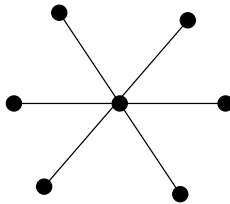
- **Lastverbund:** Gleichmäßige Verteilung der benötigten Rechenleistung auf die an das Netz angeschlossenen Rechner, Kompensation bei Rechnerausfall.

3.2 Netzwerk-Strukturen

Rechnernetze unterscheiden sich in charakteristischer Weise in der Ausprägung ihrer Vernetzung (*Topologie* des Netzes). Typisch sind:

- Sternnetze
- Busnetze
- Ringnetze
- Netze von Netzen

Sternnetze

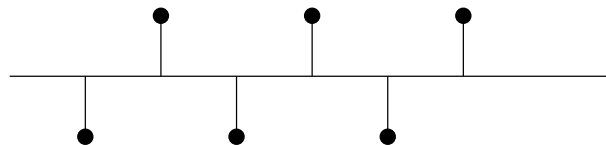


Alle Rechner sind mit einer Zentrale verbunden, die als Vermittler den Informationsaustausch regelt.

Nachteile:

- Wenn die Zentrale ausfällt, fällt das ganze Netz aus.
- Überlastung der Zentrale, wenn alle Stationen senden wollen.

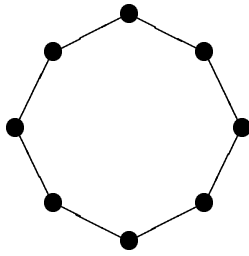
Busnetze



Alle Rechner sind an ein gemeinsames Medium angeschlossen.

Nachteil: Der Netzzugang ist schwieriger zu regeln als beim Sternnetz.

Ringnetze



Benachbarte Rechner werden verbunden.

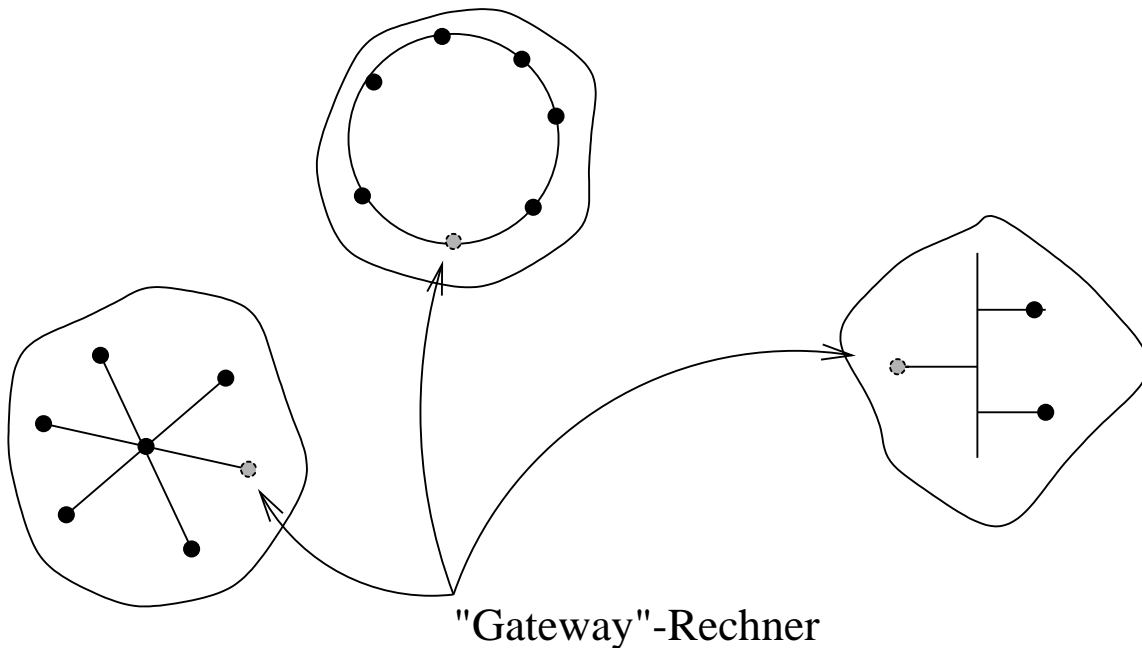
Zuteilungsverfahren bei Ringnetzen

Es gibt ein Bitmuster als Berechtigungsmarke (*Token*), das von Station zu Station weitergereicht wird und zum Senden der Nachricht berechtigt (*Token-Ring*).

Nachteil: Falls das Netz frei ist, muss eine Station trotzdem warten, bis sie das Token erhält.

Vorteil: Jede Station kommt auch bei hoher Auslastung garantiert an die Reihe.

Netze von Netzen



Die Gateway - Rechner stellen die Verbindung zwischen einzelnen Stationen in unterschiedlichen Netzen her. Der Weg, über den Datenpakete verschickt werden, wird vom *Router* festgelegt.

3.3 Kommunikation in Rechnernetzen

Protokolle

In verteilten Systemen erfolgt die Kommunikation durch *Nachrichtenaustausch* (da kein gemeinsamer Speicher vorhanden ist). *Protokolle* spezifizieren den Ablauf möglicher Kommunikationen.

Alltags-Beispiel

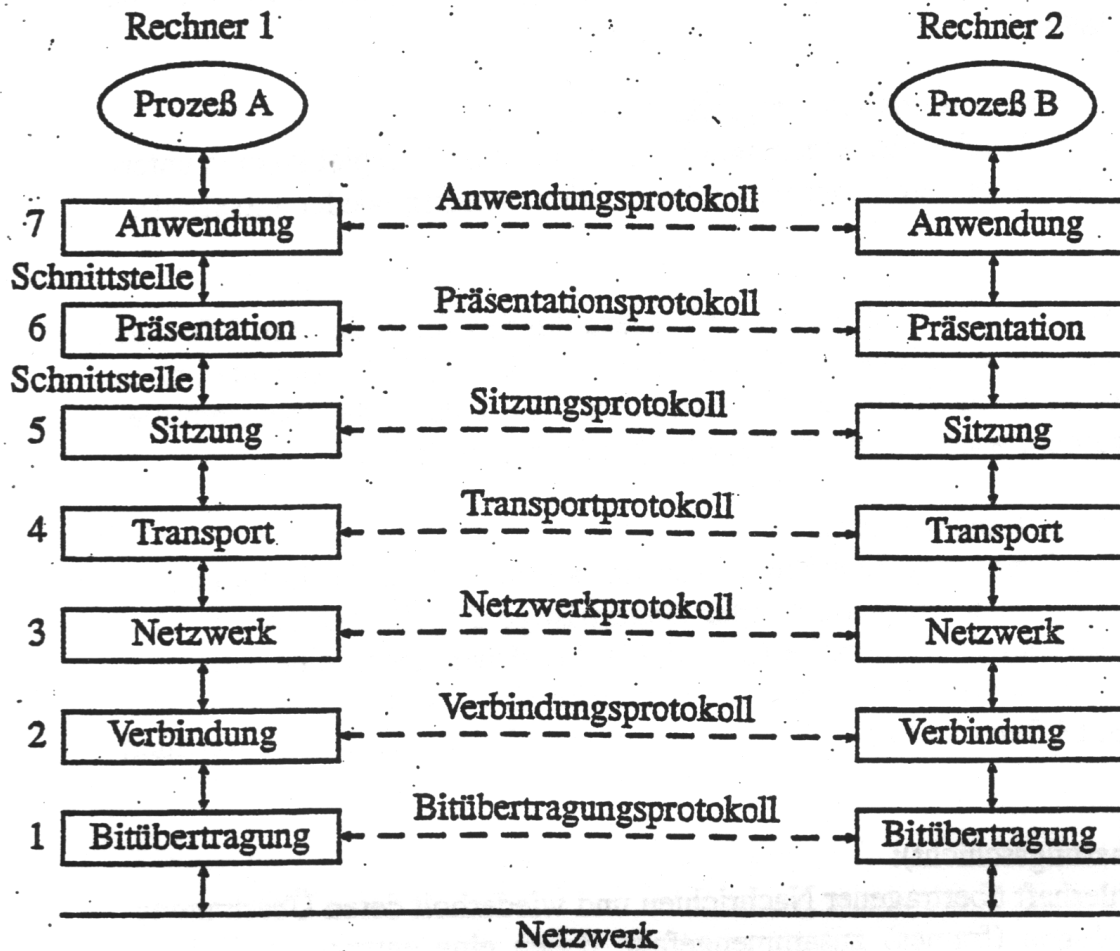
Protokoll, das den Ablauf einer Prüfung beschreibt.

Verbale Beschreibung (P - Prüfer, K - Kandidat):

```
P: fragt nach Prüfungsfähigkeit
K: erklärt, ob prüfungsfähig
wenn Kandidat prüfungsfähig, dann {
  wiederhole {
    P: stellt Prüfungsfrage
    K: gibt Antwort
    wenn P mit Antwort nicht zufrieden, dann {
      P: stellt Nachfrage
      K: gibt Antwort
      wenn P weiterhin unzufrieden, dann {
        P: stellt weitere Nachfrage
        K: gibt Antwort
        wenn P immer noch nicht zufrieden, dann {
          P: teilt richtige/erwartete Antwort mit
        }
      }
    }
  }
} solange Prüfungszeit noch nicht um
P: teilt K Prüfungsergebnis mit
}
```

Das ISO/OSI-Modell

Im ISO/OSI-Modell (*Open Systems - Interconnection*) werden von der „International Standard Organization“ sieben Schichten festgelegt, in die ein Protokoll zerlegt wird. Diese Schichten bauen aufeinander auf und reichen von der Anwendungsebene bis zur physikalischen Ebene der Bitübertragung.



- **Anwendungsschicht:**
Betrifft die anwenderspezifischen Aspekte der Kommunikation. Die Protokolle hängen von der Art der Anwendung ab (z.B. E-Mail, Dateiübertragung, ...).
- **Präsentationsschicht (auch: Darstellungsschicht):**
Festlegung der Ausdrucksmittel (z.B. Zeichenvorrat, Datenstrukturen), die zur (eindeutigen) Darstellung der Information verwendet werden (und Festlegung der Regeln zum Austausch der jeweiligen Darstellungen).
- **Sitzungsschicht (auch: Kommunikationssteuerungsschicht):**
Regelt den Dialog und bietet Synchronisationsmöglichkeiten an. (Zum Beispiel können unterbrochene Übertragungen an vorher vereinbarten Synchronisationspunkten wieder ansetzen.)
- **Transportschicht:**
Teilt die Nachrichten in einzelne Pakete auf und regelt Fragestellungen folgender Art: Wieviele Pakete sind gesendet? Wieviele empfangen? Wieviele kann der Empfänger momentan noch aufnehmen? Insbesondere ist es die Aufgabe der Transportschicht, Pakete, die über verschiedene Routen geschickt wurden, wieder in die richtige Reihenfolge zu bringen.
- **Netzwerkschicht (auch: Vermittlungsschicht):**
Bestimmt den Weg, auf dem Daten zum Empfänger gesendet werden (*Routing*). Kri-

terien sind etwa die aktuelle Auslastung einer Verbindung oder längerfristige Durchschnittswerte. Man unterscheidet verbindungsorientierte Protokolle (z.B. X.25), bei denen zuerst eine feste Verbindung aufgebaut wird, auf der dann die gesamte Nachricht übertragen wird, und verbindungslose Protokolle (z.B. IP (Internet Protokoll)), bei denen jedes Paket unabhängig von allen anderen verschickt wird.

- Verbindungsschicht (Sicherheitsschicht):
Regelt die Erkennung fehlerhaft übertragener Nachrichten und wiederholt deren Übertragung. Dazu werden Bits zu Rahmen (*frames*) zusammengefasst, denen eine entsprechende Prüfsumme zugeordnet wird.
- Physikalische Schicht:
Regelt, wie die einzelnen Bits übertragen werden (z.B. mit welcher Spannung, wieviele Bits pro Sekunde, ...).

3.4 Das Internet

Entwicklung

- ab 1969: "ARPANET"
(*Advanced Research Project Agency Net*, USA)
Vernetzung von Universitäten und Forschungseinrichtungen, die mit dem amerikanischen Verteidigungsministerium zusammenarbeiten.
- 80er Jahre
Aufteilung in einen nichtöffentlichen und einen öffentlichen Teil; aus letzterem hat sich das Internet entwickelt.

Dienste im Internet

Zunächst:

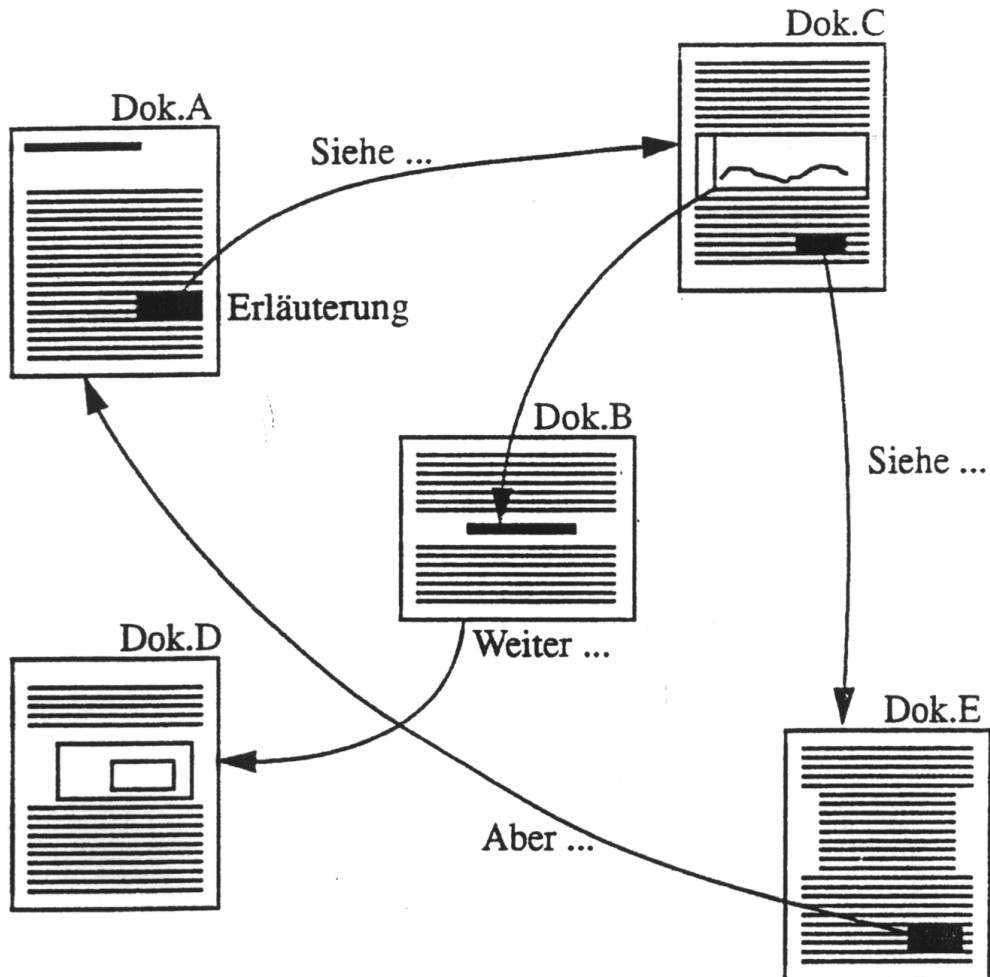
- E-Mail (*Electronic Mail*)

Dann:

- News (Diskussionsforen)
- ftp (*File Transfer Protocol*)
zum Transfer von Daten
- telnet, rlogin (Login auf entfernte Rechner)
- Gopher (Zugriff auf Dokumente durch einfache Menüauswahl)
- WWW (*World Wide Web*)

Das World Wide Web

- Weiterentwicklung von Gopher. Initiiert von CERN (europäisches Forschungslabor für Teilchenphysik). An beliebigen Stellen in einem Text können Verweise zu anderen Dokumenten (auch auf anderen Rechnern), oder zu anderen Stellen im selben Text integriert werden. (*Hypertext*)



- Da solche Dokumente auch Grafiken, Musik oder Filme enthalten können, spricht man auch von *Hypermedia*.
- Hypertexte werden im Standardformat HTML (*Hypertext Markup Language*) erstellt und als HTML-Dateien gespeichert.