

Betriebssysteme

Studiengang Wirtschaftsinformatik (042)

Wintersemester 2015/16

Inhalt

1. Einführung 
 2. Grundlagen 
 3. Spezielle Werkzeuge 
 4. Betriebssystemkonzepte 
 5. Verteilte Betriebssysteme 
-
- 

Inhalt

- [Einführung](#)
 - [Was ist ein Betriebssystem ?](#)
 - [Klassifizierung und Standardisierung](#)
- [Grundlagen für die Arbeit mit UNIX \(LINUX\)](#)
 - [Zugang zum UNIX-System](#)
 - [Kommandosprache und Kommandointerpretation](#)
 - [Das LINUX-Dateisystem](#)
 - [Arbeit mit Textdateien](#)
 - [Dateischutz](#)
 - [Programme und Prozesse](#)
 - [Kommandoprozeduren](#)
- [Spezielle Werkzeuge unter UNIX](#)
 - [awk](#)
 - [find](#)
 - [sort](#)
 - [join](#)
 - [tar](#)
 - [Bildschirmeditor vi](#)
 - [Die Programmiersprache C](#)
- [Betriebssystem-Konzepte](#)
 - [Starten eines UNDX-Systems und Anmelden eines Benutzers](#)
 - [Speicherverwaltung](#)
 - [Prozeßverwaltung](#)
 - [Dateisysteme](#)
 - [Das Ein-/Ausgabesystem](#)
 - [Das X-Window-System](#)
- [Verteilte Betriebssysteme](#)
 - [Grundkonzepte](#)
 - [Kommunikation in verteilten Systemen](#)



Spezielle Programme („Werkzeuge“) unter UNIX

awk (*pattern scanning and processing language*)

- ▶ zur Arbeit mit Textdateien
- ▶ ausgewählte Felder (Wörter) in ausgewählten Zeilen auf vielfältige Weise bearbeiten (Umformung von Daten, Erstellung von Berichten, ...)

```
awk 'programm' [ datei ... ]
```

programm → anweisung ...
anweisung → muster { aktion }

- *muster* wird zur Auswahl relevanter Zeilen benutzt.
- für jede ausgewählte Zeile wird *aktion* durchgeführt
- awk führt Anweisungen im *programm* für jede Eingabedatei *datei* durch und gibt Ergebnisse auf `stdout` aus.
- awk bearbeitet sukzessive die Zeilen jeder Datei *datei*

Beispieldatei: personal

Name	Geb.-Datum	Beruf
-----	-----	-----
Ludwig	20.12.1955	Lehrer
Schlosser	15.11.1961	Maurer
Lehmann	10.10.1970	Schlosser
Meyer	03.03.1968	Architekt

Zeilentrenner: Newline

Feldtrenner : Leerzeichen („Feld“ = Spalte)

- ▶ Felder einer Zeile durch Variablen $\$1, \$2, \dots$ ansprechbar
 - ▶ $\$0$ bezeichnet die ganze Zeile
 - ▶ spezielle Variablen:
 - FS für Feldseparator
 - RS für Record Separator

personal

Name	Geb.-Datum	Beruf
-----	-----	-----
Ludwig	20.12.1955	Lehrer
Schlosser	15.11.1961	Maurer
Lehmann	10.10.1970	Schlosser
Meyer	03.03.1968	Architekt

```
~> awk '{print $1 $2}' personal
```

```
Name Geb.-Datum
```

```
-----
```

```
Ludwig 20.12.1955
```

```
Schlosser 15.11.1961
```

```
Lehmann 10.10.1970
```

```
Meyer 03.03.1968
```

```
~> awk '{print $1, $2}' personal
```

```
Name Geb.-Datum
```

```
-----
```

```
Ludwig 20.12.1955
```

```
Schlosser 15.11.1961
```

```
Lehmann 10.10.1970
```

```
Meyer 03.03.1968
```

personal

Name	Geb.-Datum	Beruf
----	-----	-----
Ludwig	20.12.1955	Lehrer
Schlosser	15.11.1961	Maurer
Lehmann	10.10.1970	Schlosser
Meyer	03.03.1968	Architekt

```
~> awk '/L/{print $1, $2}' personal
```

```
Ludwig 20.12.1955
```

```
Lehmann 10.10.1970
```

```
~> awk '/S/{print $1, $2}' personal
```

```
Schlosser 15.11.1961
```

```
Lehmann 10.10.1970
```

- ▶ Einschluss des Programms in Hochkomma:
Schutz der Sonderzeichen vor Entwertung durch die Shell
(Kollision \$0, \$1, ... mit Prozedurparametern)
- ▶ Default-Werte:
 - keine Eingabedatei: Lesen von `stdin`
 - kein *muster*: *aktion* für jede Eingabezeile
 - keine *aktion*: `print $0`
(Ausgabe gesamte Zeile auf `stdout`)
- ▶ Die Angabe mehrerer Zeilen der Form
muster { aktion }
ermöglicht die differenzierte Behandlung von Eingabezeilen

Gegeben sei die folgende Datei `lager`:

3.	Juli	Naegel	+100
12.	Juli	Schrauben	-80
20.	Juli	Naegel	-70
22.	Juli	Duebel	+140
2.	August	Schrauben	+90
10.	August	Duebel	-100

```
~>awk '/Nae/{x=x+$4}
>/Sch/{y=y+$4}
>/Due/{z=z+$4}
>END {print "Naegel:",x,", Schrauben:",y,", Duebel:",z}
>' lager
Naegel: 30 , Schrauben: 10 , Duebel: 40
~> ...
~>awk '/Nae/{x=x+$4}
>/Sch/{y=y+$4}
>/Due/{z=z+$4}
>END {print "Naegel: " x ", Schrauben: " y ", Duebel: " z}
>' lager
Naegel: 30, Schrauben: 10, Duebel: 40
~>
```

Bemerkungen:

- Die END zugeordnete Aktion wird ausgeführt, nachdem alle Eingabezeilen abgearbeitet wurden (analog: BEGIN ... vor allen Eingabezeilen).
- print gibt Variablenwerte und Konstanten aus.
- Numerische Variablen werden automatisch mit 0 initialisiert.
- eine Aktion kann aus mehreren Anweisungen bestehen, die jeweils durch ein Semikolon oder ein Newline-Zeichen getrennt sein müssen. Als Anweisungen stehen alle Operatoren und die Elemente zur Ablaufsteuerung der Sprache C zur Verfügung!

Angabe von Mustern: reguläre Ausdrücke
(analog egrep-Befehl)

```
~>awk '/^([1-4]\.)/' lager
3. Juli Naegel +100
2. August Schrauben +90
```

→ Zeilen, die mit einer Ziffer zwischen 1 und 4, gefolgt von einem Punkt beginnen

find

- ▶ Suchen von Dateien, die bestimmten Bedingungen genügen in Teilbäumen und Ausführen von Kommandos für die gefundenen Dateien

```
find [ verzeichnis ... ] bedingung ... aktion ...
```

- durchsucht alle angegebenen Verzeichnisse nach Dateien, für die die angegebenen Bedingungen erfüllt sind und führt für diese die angegebenen Aktionen aus.
- es werden auch Unterverzeichnisse rekursiv durchsucht
- sind mehrere Bedingungen angegeben, werden diese mit UND verknüpft.
- Bedingungen:

-name dateiname

erfüllt, wenn eine Datei mit dem Dateinamen existiert

- Bedingungen:

`-perm [-]oktalzahl`

erfüllt, wenn eine Datei gefunden wird, deren Zugriffsrechte der angegebenen Oktalzahl entsprechen.

Steht ein minus vor der Oktalzahl, werden nur die mit *oktalzahl* spezifizierten Zugriffsrechte überprüft, die restlichen sind bedeutungslos.

`-type t`

erfüllt, wenn eine Datei mit dem Dateityp *t* existiert.
Mögliche Werte für *t* :

f - gewöhnliche Datei

d - Verzeichnis

b - blockorientierte Gerätedatei

c - character-orientierte Gerätedatei

l - symbolischer Link

p - named pipe

`-links n`

erfüllt, wenn eine Datei mit *n* Links gefunden wird.

`-atime n`

erfüllt, wenn auf die Datei vor *n* Tagen zugegriffen wurde

`-mtime n`

erfüllt, wenn die Datei vor *n* Tagen verändert wurde

`-size n`

erfüllt, wenn die Datei *n* viele Blöcke besitzt

`-user name`

erfüllt, wenn der Eigentümer der Datei *name* heißt

- Aktionen:

- print

- Ausgabe des Dateinamens jeder gefundenen Datei
(implizit angenommen)

- exec *kommando* ;

- auf jede gefundene Datei wird *kommando* angewendet
 - im *kommando* wird anstelle des **Dateinamens { }** verwendet

Beispiel:

```
find /usr/otto -atime +10 -print
```

bewirkt die Anzeige aller Dateien unterhalb des Verzeichnisses /usr/otto, auf die während der letzten 9 Tage nicht zugegriffen wurde

- + steht für gleich oder größer (d.h. mehr als ... Tagen)
- steht für gleich oder kleiner

Beispiel:

```
find /usr/otto ! -user otto -exec ls -l {} \;
```

bewirkt die Anzeige aller Dateien, deren Eigentümer nicht `otto` ist. Die Auflistung der Dateien erfolgt mit `ls` in der Langform.

diff

- Realisiert einen Vergleich von zwei Dateien.
Die Ausgabe auf `stdout` enthält Informationen darüber, welche Zeilen der ersten Datei geändert werden müssen, um diese der zweiten Datei anzugeleichen, und umgekehrt.

```
diff [ option ... ] dateiname1 dateiname2
```

option

- e Ausgabe einer Datei mit a-, c-, d – Befehlen und Daten für `ed`-Editor
- b Folgen von Leerzeichen als ein Leerzeichen

Beispiel:

```
cat dat1
A
B
```

```
cat dat2
A
C
```

```
diff -e dat1 dat2
2c
C
.
```

Sortieren

- ▶ Sortieren von Zeilen einer Textdatei (mit Feldern)
 - ▶ Schlüsselfeld (-felder) festlegen
 - ▶ Ordnungsrelation: Lexikographische Ordnung gemäß ASCII
 - ▶ bei mehreren Schlüsselfeldern:
 - erstrangig
 - zweitrangig
 - ...
- (z.B. - Name
 - Vorname
 - Wohnort bei Personen)

sort

- ▶ Sortieren von Textdateien

```
sort [ option ... ] [ +pos1 [ -pos2 ] ] ... [ -o ausgabedatei ] datei  
...
```

- Im Standardfall (Angaben *pos1* und *pos2* fehlen) wird die gesamte Zeile einer Datei als Schlüsselfeld verwendet, und es wird in aufsteigender lexikographischer Reihenfolge sortiert.
- die Angaben *pos1* und *pos2* bestimmen Schlüsselfelder.
- Optionsangaben steuern die Art der Sortierung
- Bei *-o* erfolgt die Ausgabe in eine Datei *ausgabedatei*, sonst auf *stdout*.

```
~> sort preise
```

Butter	0.95	EUR
Gurken	3.01	EUR
Kaese	7.05	EUR
Kaffee	6.99	EUR
Milch	0.85	EUR

```
~> sort -r preise
```

Milch	0.85	EUR
Kaffee	6.99	EUR
Kaese	7.05	EUR
Gurken	3.01	EUR
Butter	0.95	EUR

```
~> sort +1 preise
```

Milch	0.85	EUR
Kaese	7.05	EUR
Butter	0.95	EUR
Gurken	3.01	EUR
Kaffee	6.99	EUR

```
~> sort -n +1 preise
```

Milch	0.85	EUR
Butter	0.95	EUR
Gurken	3.01	EUR
Kaffee	6.99	EUR
Kaese	7.05	EUR

preise

Gurken	3.01	EUR
Butter	0.95	EUR
Kaffee	6.99	EUR
Milch	0.85	EUR
Kaese	7.05	EUR

[+pos1 [-pos2]]

- bestimmt die Zeichen zwischen pos1 und pos2 zum Schlüsselfeld (einschließlich pos1, ausschließlich pos2).
- wenn pos2 fehlt, werden alle Zeichen bis Zeilenende zum Schlüssel
- Position der Felder und Buchstaben zählen von 0 an, Feldtrenner wird mitgezählt

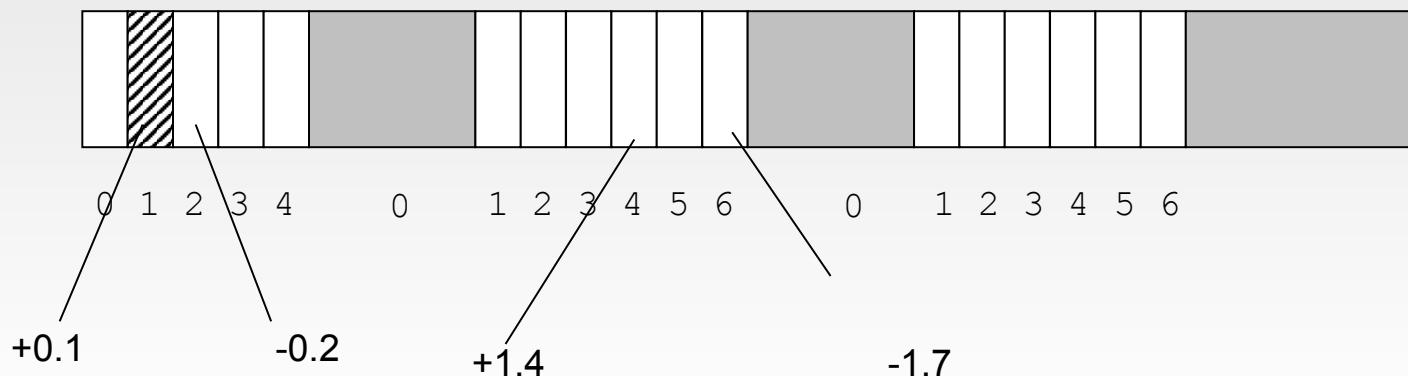
feld [.buchstabe]

- pos1, pos2 hat jeweils die Form:

Feld0

Feld1

Feld2



Optionsangaben:

- b ignoriert führende Leerzeichen
- c prüft, ob Datei bereits sortiert ist, falls nein RC=1 und Fehlermeldung
- d sortiert in alphabetischer Reihenfolge
- f unterscheidet nicht zwischen Groß- und Kleinschreibung
- n numerische Sortierung:
 - ignoriert führende Leerzeichen
 - behandelt – als Vorzeichen
- r sortiert in umgekehrter Reihenfolge
- o *dateiname*
- t *separator* benutzt *separator* als Feldtrenner

Beispiele:

```
fritzsch@isys12:~ > cat datei2  
55555 555555 555555  
55554 555554 555554  
55545 555545 555545  
55544 555544 555544  
55455 555455 555455  
55454 555454 555454  
55445 555445 555445  
55444 555444 555444  
54555 554555 554555
```

```
fritzsch@isys12:~ > sort +1.4 -1.6 datei2  
55444 555444 555444  
55445 555445 555445  
55454 555454 555454  
55455 555455 555455  
55544 555544 555544  
55545 555545 555545  
54555 554555 554555  
55554 555554 555554  
55555 555555 555555
```

```
fritzsch@isys12:~ > sort +1.4 datei2  
55444 555444 555444  
55445 555445 555445  
55454 555454 555454  
55455 555455 555455  
55544 555544 555544  
55545 555545 555545  
55554 555554 555554  
54555 554555 554555  
55555 555555 555555
```

```
fritzsch@isys12:~ > sort +1.4 -1.7 datei2  
55444 555444 555444  
55445 555445 555445  
55454 555454 555454  
55455 555455 555455  
55544 555544 555544  
55545 555545 555545  
55554 555554 555554  
54555 554555 554555  
55555 555555 555555
```

```
fritzsch@isys12:~ > sort +1.4 -2 datei2  
55444 555444 555444  
55445 555445 555445  
55454 555454 555454  
55455 555455 555455  
55544 555544 555544  
55545 555545 555545  
55554 555554 555554  
54555 554555 554555  
55555 555555 555555
```

join

- vereinigen („mischen“) von sortierten Textdateien

```
join [ option ... ] datei1 datei2
```

- **join** fasst diejenigen Zeilen aus den Dateien *datei1* und *datei2* zusammen, deren Schlüsselfelder identisch sind.
- **beide Dateien müssen bzgl. des Schlüsselfeldes sortiert sein !!**
- als Schlüsselfeld kann jedes Feld innerhalb der Zeilen verwendet werden.
- die gemischten Zeilen werden auf `stdout` ausgegeben.

Optionen:

- j *m* das *m*-te Feld wird in beiden Dateien als Schlüsselfeld verwendet (**Zählung beginnend bei 1 !**)
- j1 *m* das *m*-te Feld wird in *datei1* als Schlüsselfeld verwendet
- j2 *m* das *m*-te Feld wird in *datei2* als Schlüsselfeld verwendet
- a1 die Zeilen aus *datei1* ausgeben, für die keine Zeile mit gleichem Schlüsselfeld-Inhalt in *datei2* existiert
- a2 die Zeilen aus *datei2* ausgeben, für die keine Zeile mit gleichem Schlüsselfeld-Inhalt in *datei1* existiert
- o *n.m* legt die Felder fest, die auszugeben sind: aus der *n*-ten Datei ist das *m*-te Feld auszugeben. Für *n* kann entweder 1 (d.h. *datei1*) oder 2 (d.h. *datei2*) stehen.
- tc legt das Zeichen *c* als Trennzeichen für die Felder fest. Gilt dann sowohl für die Eingabe als auch für die Ausgabe.

Beispiel:

```
join -t: -j1 1 -j2 2 -o 1.2 1.1 2.2 datx daty
```

mischt die Dateien *datx* (1. Feld = Schlüsselfeld) und *daty* (2. Feld = Schlüsselfeld). Die Felder sind durch Doppelpunkt (:) getrennt. Bei der Ausgabe des Mischergebnisses wird zuerst das 2. Feld und anschließend das 1. Feld von *datx* ausgegeben. Danach wird das 2. Feld *daty* ausgegeben.

tar Anlegen und Verwalten von Archiven

(Sicherung von Dateien, Kopieren von Dateisystemen)

```
tar [ funktion [ zusatz ] ] [ datei ... ]
```

datei Name der zu sichernden bzw. rückzuspeichernden Datei oder der Teilhierarchie, die in dem angegebenen Verzeichnis beginnt.

funktion

c Dateien werden von Beginn des Sicherungsdatenträgers an auf diesen geschrieben (früher gespeicherte Dateien sind verloren).

r Ausgabe der angegebenen Dateien hinter die schon auf dem Datenträger geschriebenen Dateien.

x die angegebenen Dateien werden vom Sicherungsdatenträger kopiert.

zusatz

f das nächste noch nicht vergebene Argument ist der Name des Archives bzw. eines Gerätes, in dem der Sicherungsdatenträger zur Verfügung steht. Wird - angegeben, erfolgt je nach Funktion die Eingabe über `stdin` oder die Ausgabe auf `stdout`.

v Ausgabe des Namens jeder bearbeiteten Datei.

Beispiele:

1. Anlegen eines neuen Archives

```
tar cvf Appl.tar
```

2. Sukzessives Eintragen von Dateien in das Archiv

```
tar rvf Appl.tar dat1
```

```
tar rvf Appl.tar dat2
```

...

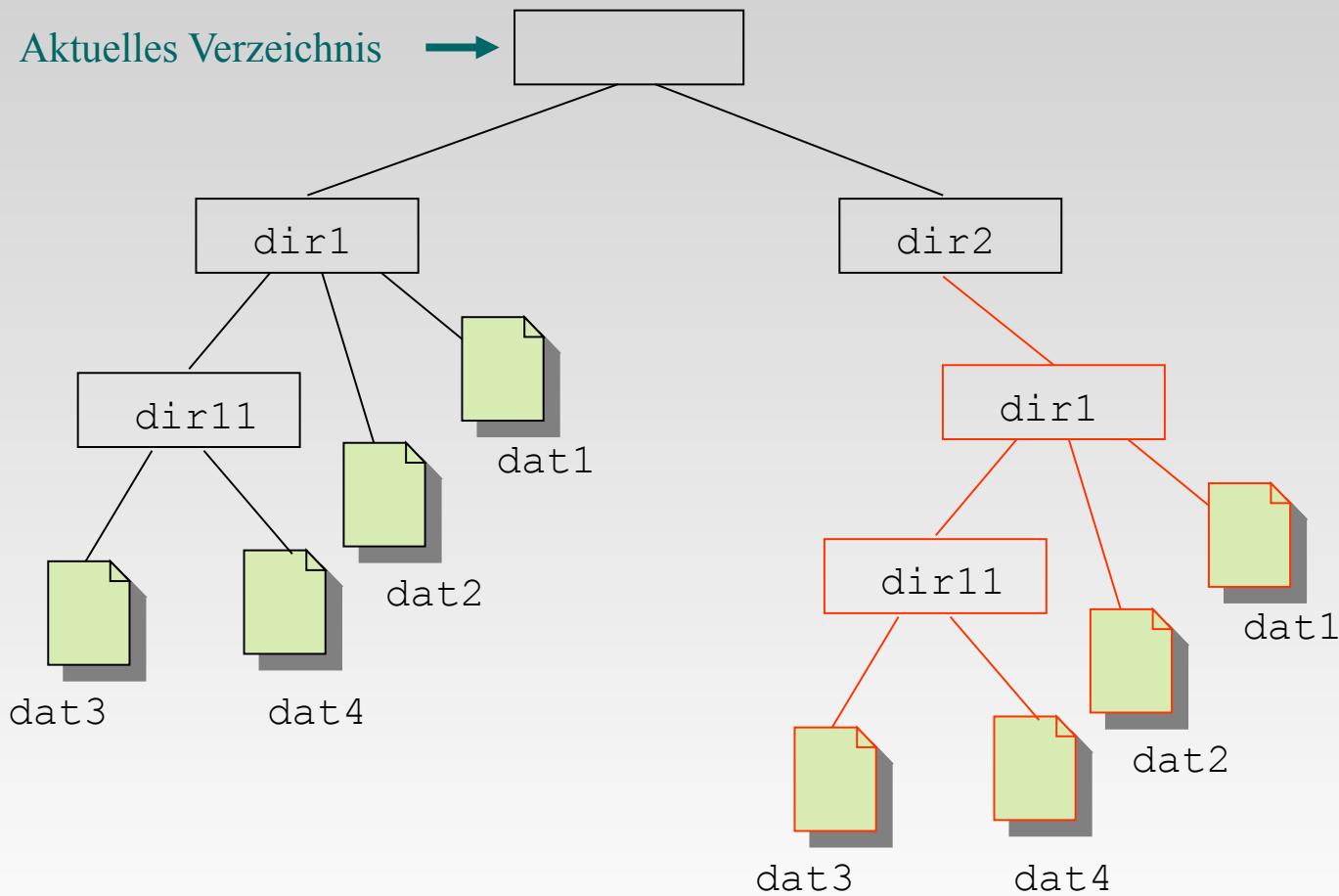
3. Extrahieren

```
tar xvf Appl.tar
```

→ die **Files** dat1, dat2, ... werden in das aktuelle Verzeichnis
geschrieben

Kopieren von Teilbäumen innerhalb einer Dateihierarchie:

```
tar cf - dir1 | (cd dir2 ; tar xf - )
```



- ▶ Starten eines UNIX-Systems
- ▶ Speicherverwaltung (HS)
 - Swapping, Paging
- ▶ Prozessverwaltung
 - Scheduling
 - (Inter-) Prozesskommunikation
- ▶ Dateiverwaltung
- ▶ Ein-/Ausgabesystem
- ▶ X-Window

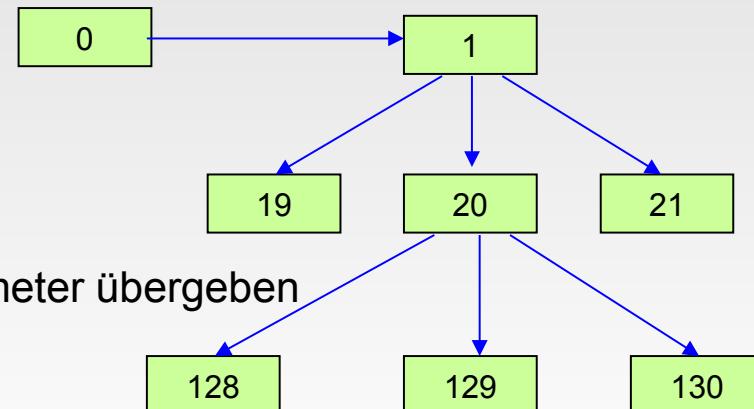


Starten eines UNIX-Systems

1. „Kernel“ in HS laden u. ausführen (Prozess 0) (in: /boot)
2. Kernel-Prozess startet Programm /sbin/init (Prozess 1, Wurzel)
3. /sbin/init
 - liest Systemtabelle /etc/inittab
 - startet Sohnprozess für jeden als zu aktivierend gekennzeichneten Anschluss Jeder dieser Prozesse
 - öffnet stdin, stdout, stderr
 - führt /etc/getty aus

/etc/getty

- gibt Login-Aufforderung auf Terminal aus
- wartet auf Eingabe
- Eingabe wird dem login-Kommando als Parameter übergeben
- bei erfolgreichem Login wird Shell gestartet



Ausschnitt aus /etc/inittab auf isys12:

```
. . .
# getty-programs for the normal runlevels
# <id>:<runlevels>:<action>:<process>
# The "id" field MUST be the same as the last
# characters of the device (after "tty") .
1:2345:respawn:/sbin/mingetty --noclear tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6
. . .
```

► Zugangskontrolle

/bin/login liest Benutzername und Paßwort → Identifizierung, Authentifikation

```
~>cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/bash
daemon:x:2:2:Daemon:/sbin:/bin/bash
lp:x:4:7:Printing daemon:/var/spool/lpd:/bin/bash
mail:x:8:12:Mailer daemon:/var/spool/clientmqueue:/bin/false
games:x:12:100:Games account:/var/games:/bin/bash
wwwrun:x:30:8:WWW daemon apache:/var/lib/wwwrun:/bin/false
ftp:x:40:49:FTP account:/srv/ftp:/bin/bash
```

verschlüsseltes Passwort (meist in /etc/shadow)

/etc/getty → **/bin/login** → **/bin/bash**

endet einer der Prozesse, wird /sbin/init unterrichtet und dieser startet erneut einen Prozess zur Bedienung des entsprechenden Terminals

► Die Datei `.profile`

ist die Datei im Home-Verzeichnis eines Benutzers vorhanden, so erwartet die Login-Shell, dass dies eine Kommandoprozedur ist und führt die Kommandos vor der Ausgabe des 1. Prompt-Zeichens aus.

Beispiel:

```
echo "Los geht's!"  
PS1=" \u@\h: [\w] >"  
umask 077  
...  
Los geht's! fritzsch@ilux150: [~] >
```

umask

- Steuerung der voreingestellten Zugriffsrechte beim Anlegen neuer Dateien
Der Befehl sollte in einer Konfigurationsdatei (`.login`) enthalten sein

umask [*permission*]

- *permission* besteht aus einer Anzahl von Oktalziffern, Angaben analog zum Kommando `chmod`, führende Nullen sind ohne Bedeutung
- In *permission* wird festgelegt, welche Rechte von den standardmäßig für eine erzeugte Datei von dem erzeugenden Programm vergebenen Rechten nicht gegeben (d.h. „maskiert“) werden sollen.
- Ohne Angabe von *permission* werden die aktuellen Einstellungen angezeigt

Beispiel: umask 022

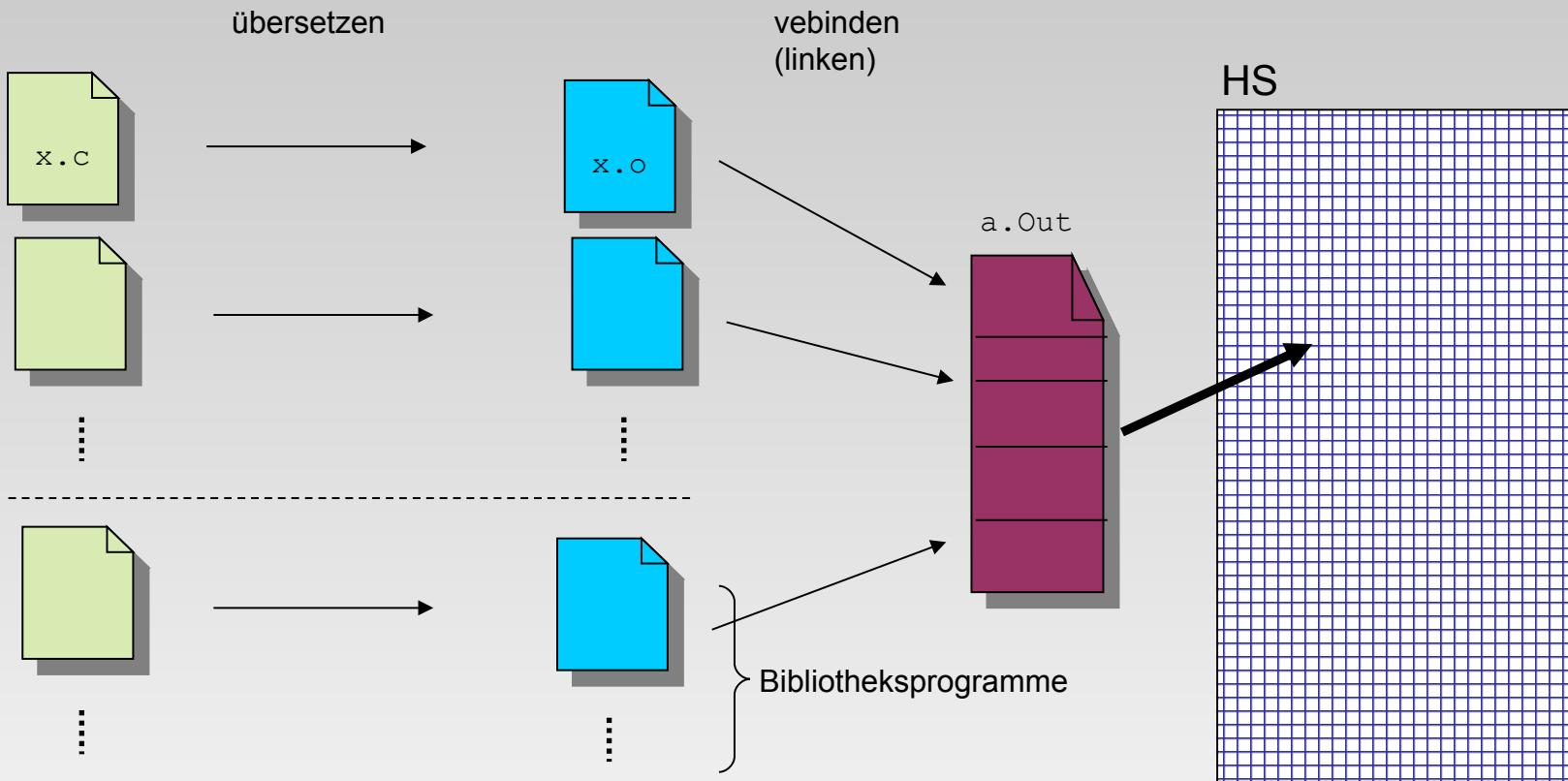
Schreibrechte für die „Gruppe“ und für „andere Benutzer“ werden nicht erteilt

	u			g			o		
	r	w	x	r	w	x	r	w	x
Initial	1	1	0	1	1	0	1	1	0
umask	0	0	0	0	1	0	0	1	0
erteilte Rechte	1	1	0	1	0	0	1	0	0

Speicherverwaltung (Hauptspeicher)

- ▶ Verwaltung freier und belegter Speicherbereiche
 - Zuweisung von Speicherbereichen an Prozesse, wenn diese sie benötigen
 - Freigabe von Speicherplatz, wenn Prozesse abgeschlossen sind
 - *zeitweise Auslagerung von belegtem Hauptspeicher auf Platte, wenn nicht alle Prozesse gleichzeitig im Speicher gehalten werden können.*
- ▶ Klassifizierung
 - Einfache Speicherverwaltungssysteme
 - Speicherverwaltungssysteme mit **Swapping** und **Paging**

Dynamisches Binden (Linken):

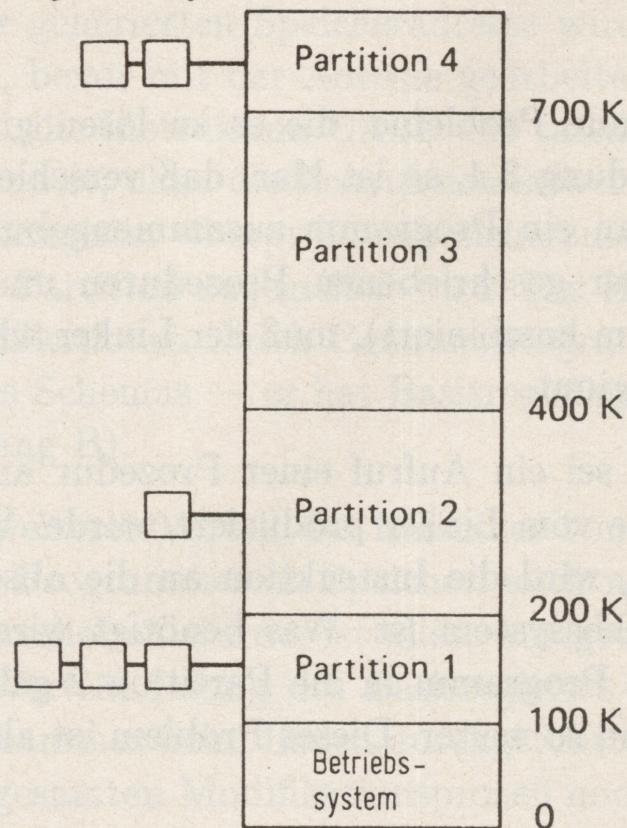


Quellprogramme

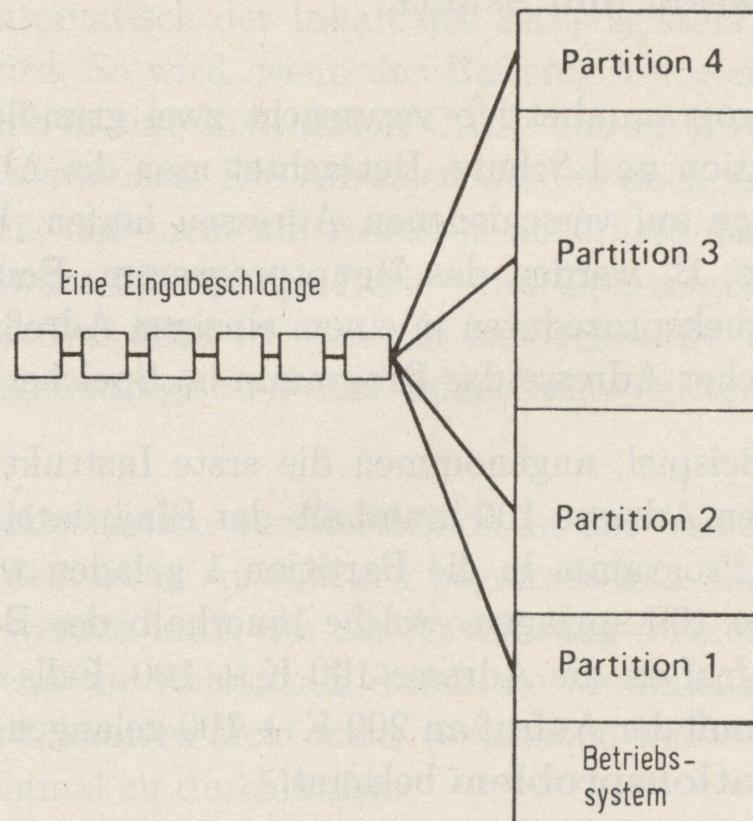
Objektprogramme
(Zielprogramme)

Ausführbares Programm
(Lademodul, Executable)

Mehrere
Eingabeschlangen



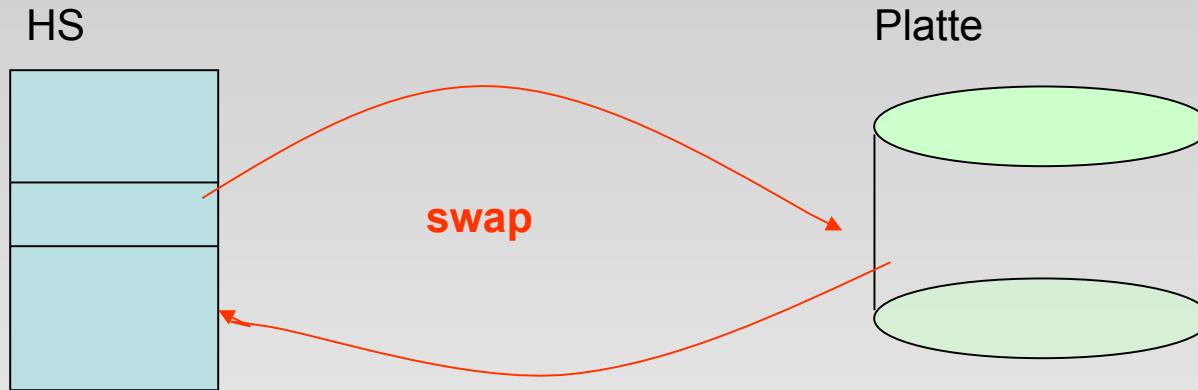
- a) Feste Partitionen
getrennte Auftragsschlangen



- b) Feste Partitionen
eine Auftragsschlange

Problem: Zahl Prozesse > für entsprechende Programme
(Multiuser !) verfügbarer Speicher

Lösungsidee: blockierende Prozesse zwischenzeitlich auf Platte auslagern

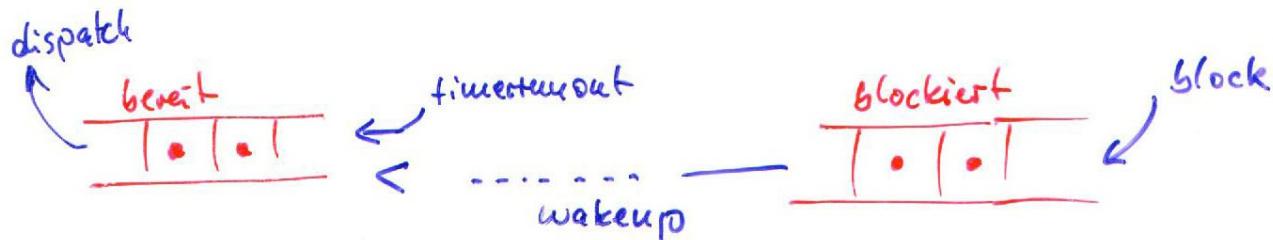
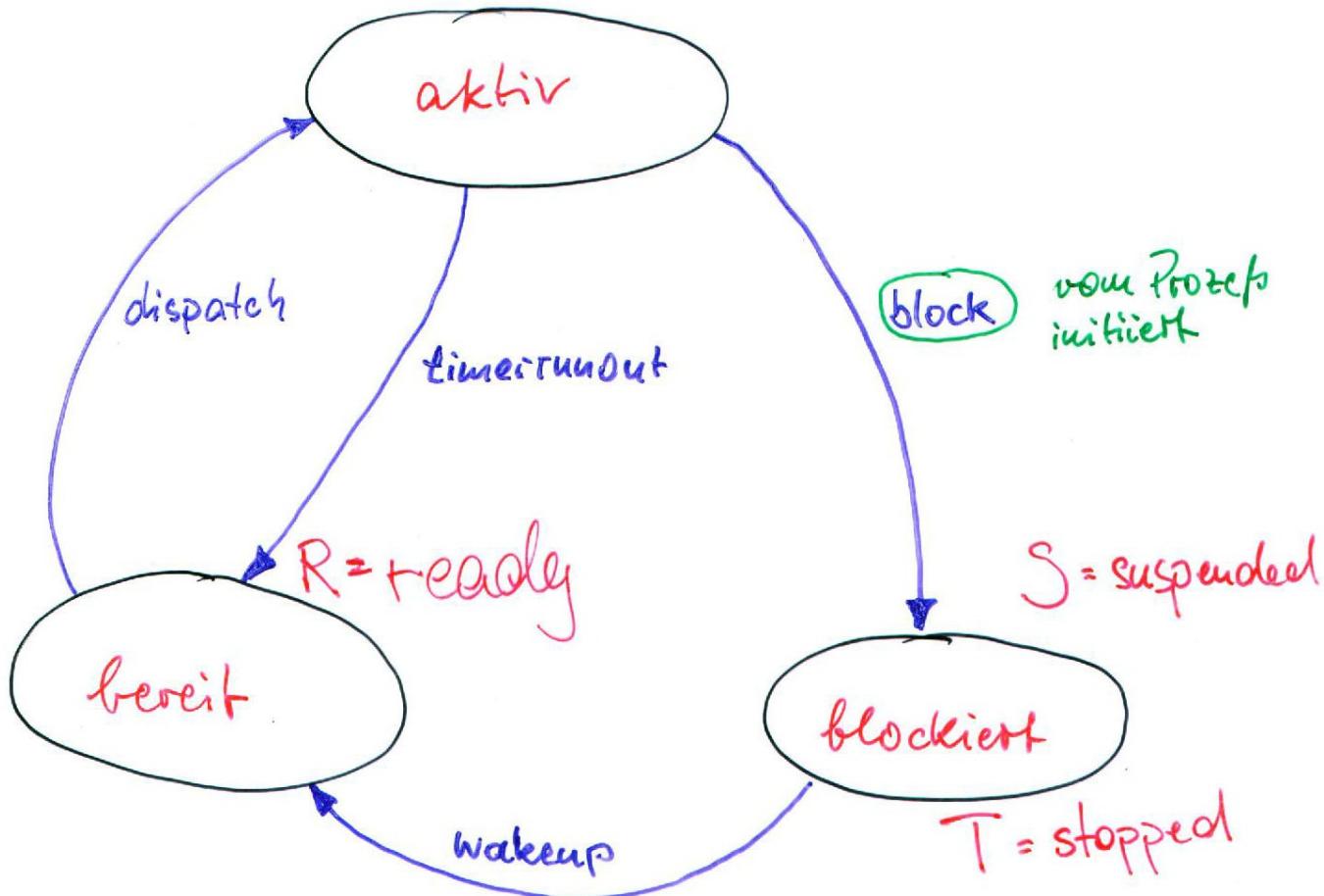


Ob ein Prozess **ausgelagert** wird, hängt ab von der

- Größe des belegten Hauptspeichers
- Art des erwarteten Ereignisses
- Verweilzeit im Hauptspeicher

→ Rechenbereite Prozesse werden nicht ausgelagert, bevor ihnen Rechenzeit zugeteilt wurde.

R = running



Problem: Programm > verfügbarer HS ?

Lösung:

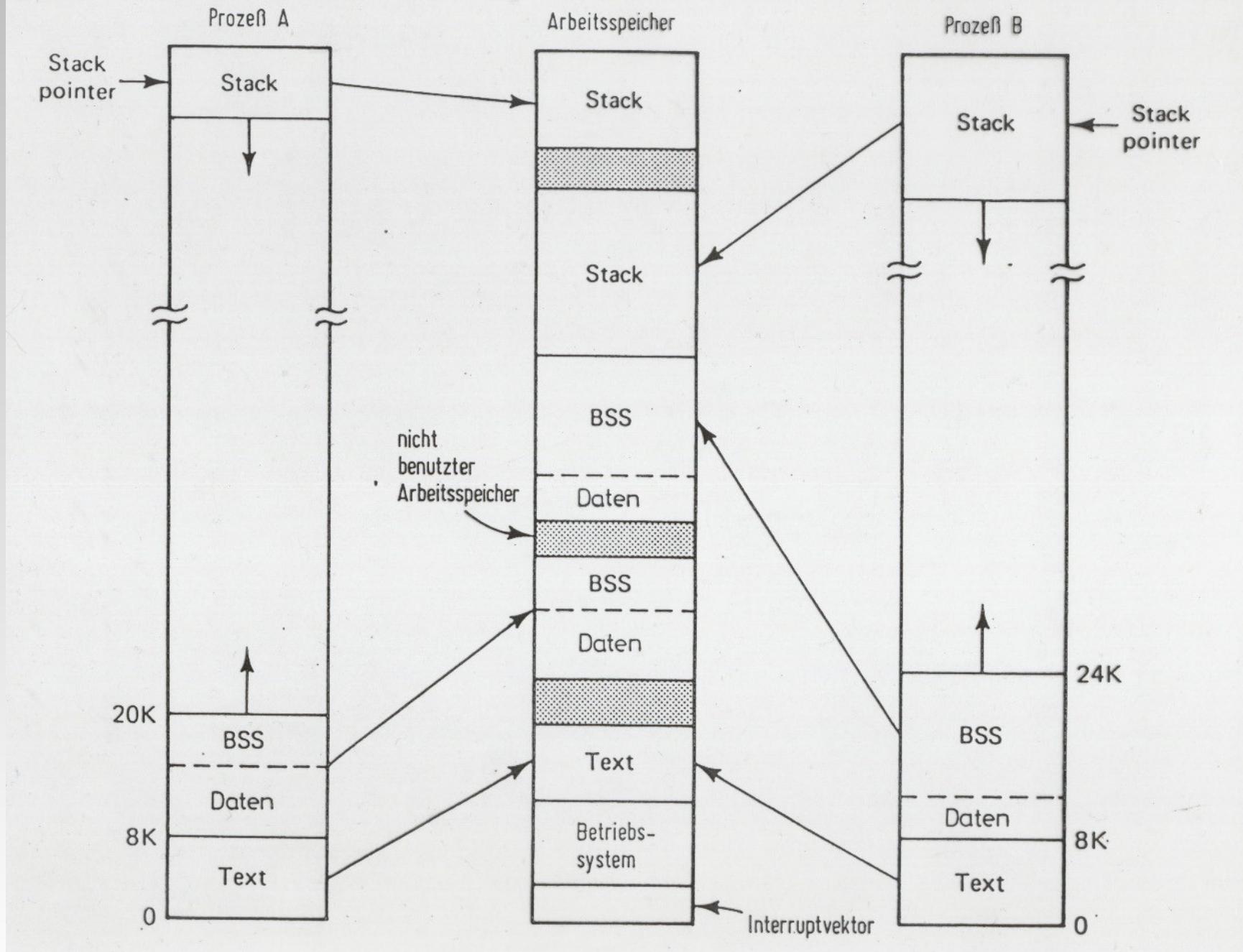
- 1) Teilung des Programms manuell (→ Überlagerungsstrukturen)
- 2) Virtueller Speicher

Virtueller Speicher: Es werden nur noch benötigte Teile von Prozessen im Speicher gehalten.

- Teile: „Seiten“ (pages), z.B. 4 KByte
- Seitenwechsel wird als **Paging** bezeichnet
- Programm nutzt **virtuellen Adressraum** in Maschinenbefehlen.
Die Speicherverwaltungseinheit MMU (*Memory Management Unit*) rechnet virtuelle Adressen in reale Adressen um.

Vorteil: Es können Programme ausgeführt werden, die größer sind als der verfügbare Hauptspeicher !

- Jeder Prozess in UNIX besitzt einen Adressraum, der aus drei Segmenten besteht:
1. Das Textsegment (*user text segment*)
 - ausführbaren Code (Maschinencode)
 - normalerweise nur lesender Zugriff
 - ggf. von mehreren Prozessen gemeinsam genutzter Speicher (*shared text segment*)
 2. Datensegment (*user data segment*)
 - Speicherplatz für Variablen, Zeichenketten, Felder, ...
 - initialisierte Daten
 - nicht initialisierte Daten (BSS)
 3. Stacksegment (*system data segment*)
 - Umgebungsvariablen
 - Argumentübergabe (Kommandozeile Shell)
 - Registerstände



(a) Virtueller Adressraum Prozess A (b) Physikalischer Speicher

(c) virtueller Adressraum Prozess B
Quelle: Tanenbaum, Moderne Betriebssysteme

- Der Zugriff auf Text- und Datensegmente der Prozesse erfolgt mittels Prozeßtabelle und Texttabelle. (siehe → Prozessverwaltung)

Diese Tabellen befinden sich ständig im Hauptspeicher.

Beispiel:

Das Kommando `size` gibt Größe des Text-, Daten- und bss-Segments in Bytes aus.
4./5. Spalte: Gesamtgröße aller drei Segmente dezimal und hexadezimal.

```
fritzs@ilux150:[BS-Programmbeispiele] >size /bin/awk /usr/bin/cc
    text      data      bss      dec      hex filename
 279702      1272     20452   301426   49972 /bin/awk
 232305      2400     2404   237109   39e35 /usr/bin/cc
fritzs@ilux150:[BS-Programmbeispiele] >[ ]
```

Prozessverwaltung

► Zu einem Prozess gehören

- das auszuführende Programm
 - der Prozesszustand
 - die geöffneten Dateien
 - die Benutzer- und Gruppenkennungen
 - das Prozess-Environment
 - das aktuelle Directory
- ...

Beispiel:

```
#include <stdio.h>

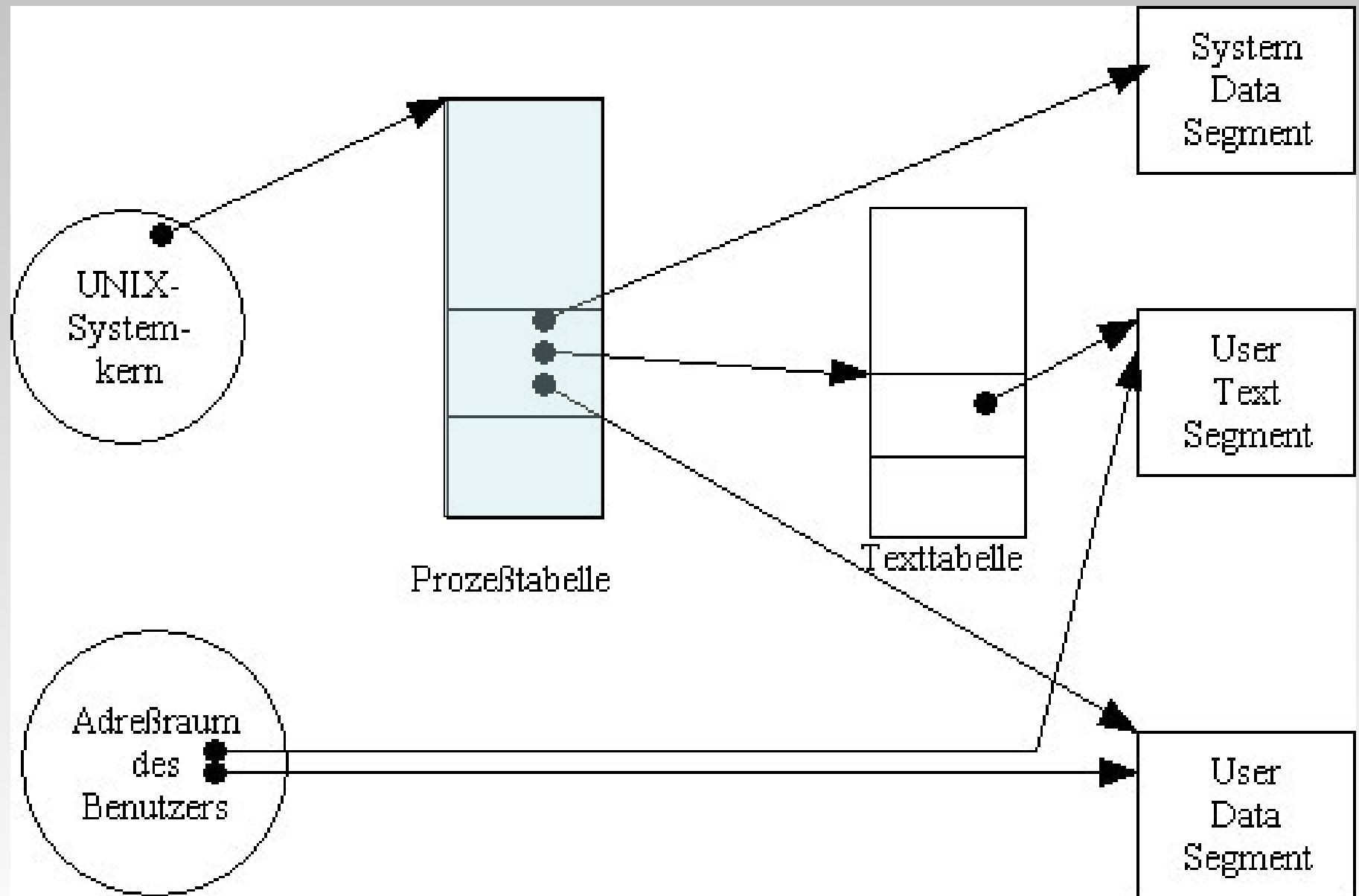
int main(void) {
    printf("uid = %d, gid = %d\n", getuid(),getgid());
    exit(0);
}
```

► Ein Prozess kann (mittels Systemaufruf)

- die Ausführung eines anderen Prozesses veranlassen,
 - sein aktuelles Verzeichnis ändern,
- ...

► Die Prozessnummer bleibt immer erhalten

- ▶ Die Prozessverwaltung regelt die Zuteilung von Betriebsmitteln (d.h. CPU, HS, externer Speicher, ...) zu Prozessen.

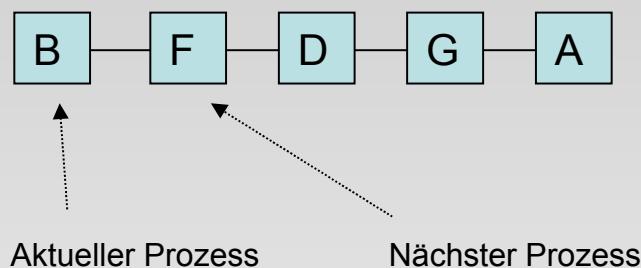


- ▶ Die Prozesstabelle enthält für jeden Prozess einen Eintrag
- ▶ Bei Zuteilung von CPU und Speicher muss entschieden werden:
 - Welcher der rechenbereiten Prozesse als nächster ausgeführt wird
→ **Scheduling**
 - Welche Prozesssegmente aus dem HS ausgelagert werden, um Platz für Datenbereiche des auszuführenden Prozesses zu schaffen, die sich zur Ausführungszeit im Speicher befinden müssen.
→ **Swapping, Paging**

Prozess-Scheduling

- Realisiert Prozess-Zustandsübergänge aktiv \leftrightarrow rechenbereit
- Es gibt viele Algorithmen für konkurrierende Forderungen nach **Effizienz** und **Fairness** gegenüber einzelnen Prozessen
- Ob einem Prozess CPU-Zeit zugewiesen wird, hängt ab von
 - Priorität des Prozesses
 - Platzbedarf des Prozesses
 - verstrichene Wartezeit des Prozesses
- Regeln:
 - Jeder Prozess in der Systemphase hat höhere Priorität als alle Prozesse in der Benutzerphase
 - die Priorität eines Prozesses in der Benutzerphase ist um so größer je kleiner das Verhältnis
$$\frac{\text{verbrauchte CPU-Zeit}}{\text{insgesamt verbrauchte Zeit}}$$
ist.

Round-Robin-Scheduling



Wenn ein Prozess blockiert oder seine Ausführung beendet ist, bevor das Quantum abgelaufen ist, wird ihm der Prozessor entzogen.



Kommunikation zwischen Prozessen

Prozesse können auf unterschiedliche Art und Weise kommunizieren. Es gibt dazu folgende Konzepte:

- temporäre Dateien
 - Pipes („named pipes“)
 - Signale (Unterbrechungssystem)
 - Shared Memory (gemeinsam genutzter Speicher)
 - Semaphore
 - Messages
 - Sockets
- IPC

Signale

Signale werden durch ganzzahlige Nummern identifiziert.

Sie sind über symbolische Namen ansprechbar, die in `<signal.h>` definiert sind.

Beispiele für Signalnummern (`/usr/include/bits/signum.h`):

```
/* Signals. */
#define SIGHUP 1      /* Hangup (POSIX). */
#define SIGINT 2      /* Interrupt (ANSI). */
#define SIGQUIT 3     /* Quit (POSIX). */
#define SIGILL 4      /* Illegal instruction (ANSI). */
#define SIGTRAP 5     /* Trace trap (POSIX). */
#define SIGABRT 6     /* Abort (ANSI). */
#define SIGIOT 6      /* IOT trap (4.2 BSD). */
#define SIGBUS 7      /* BUS error (4.2 BSD). */
#define SIGFPE 8      /* Floating-point exception (ANSI). */
#define SIGKILL 9     /* Kill, unblockable (POSIX). */
#define SIGUSR1 10    /* User-defined signal 1 (POSIX). */
#define SIGSEGV 11    /* Segmentation violation (ANSI). */
#define SIGUSR2 12    /* User-defined signal 2 (POSIX). */
#define SIGPIPE 13    /* Broken pipe (POSIX). */
#define SIGALRM 14    /* Alarm clock (POSIX). */
#define SIGTERM 15    /* Termination (ANSI). */
#define SIGSTKFLT 16  /* Stack fault. */
```

Signale stellen asynchrone Ereignisse dar. Bei Auftreten eines Signals kann

- das Signal ignoriert werden
- das Signal abgefangen werden und zu einer bestimmten C-Funktion verzweigt werden
- eine Standard-Prozedur ausgeführt werden (meist Prozess beenden)

Senden und Afangen von Signalen in Programmen:

```
kill(prozessnr,signalnummer)
signal(signalnr,SIG_IGN)
signal(signalnr,SIG_DFL)
signal(signalnr,funktionsname)
```

Mit `SIG_IGN` wird das System angewiesen, das Signal zu ignorieren.

Mit `SIG_DFL` wird das System angewiesen, wieder die Standardprozedur zuzuordnen.

Beispiel: sigloop.c

```
#include <signal.h>
main() {
    void sigfunkt();
    signal(SIGINT,sigfunkt);
    signal(SIGQUIT,sigfunkt);
    while(1);
}

sigfunkt() {
    printf("Signal empfangen\n");
    signal(SIGINT,SIG_DFL);
    return;
}
```

Anwendung:

```
fritzs@isys13:~/BSII> ./sigloop
(drücken ^C)
Signal empfangen
(drücken ^C)
fritzs@isys13:~/BSII>
```

Kommunikation mittels Shared Memory :

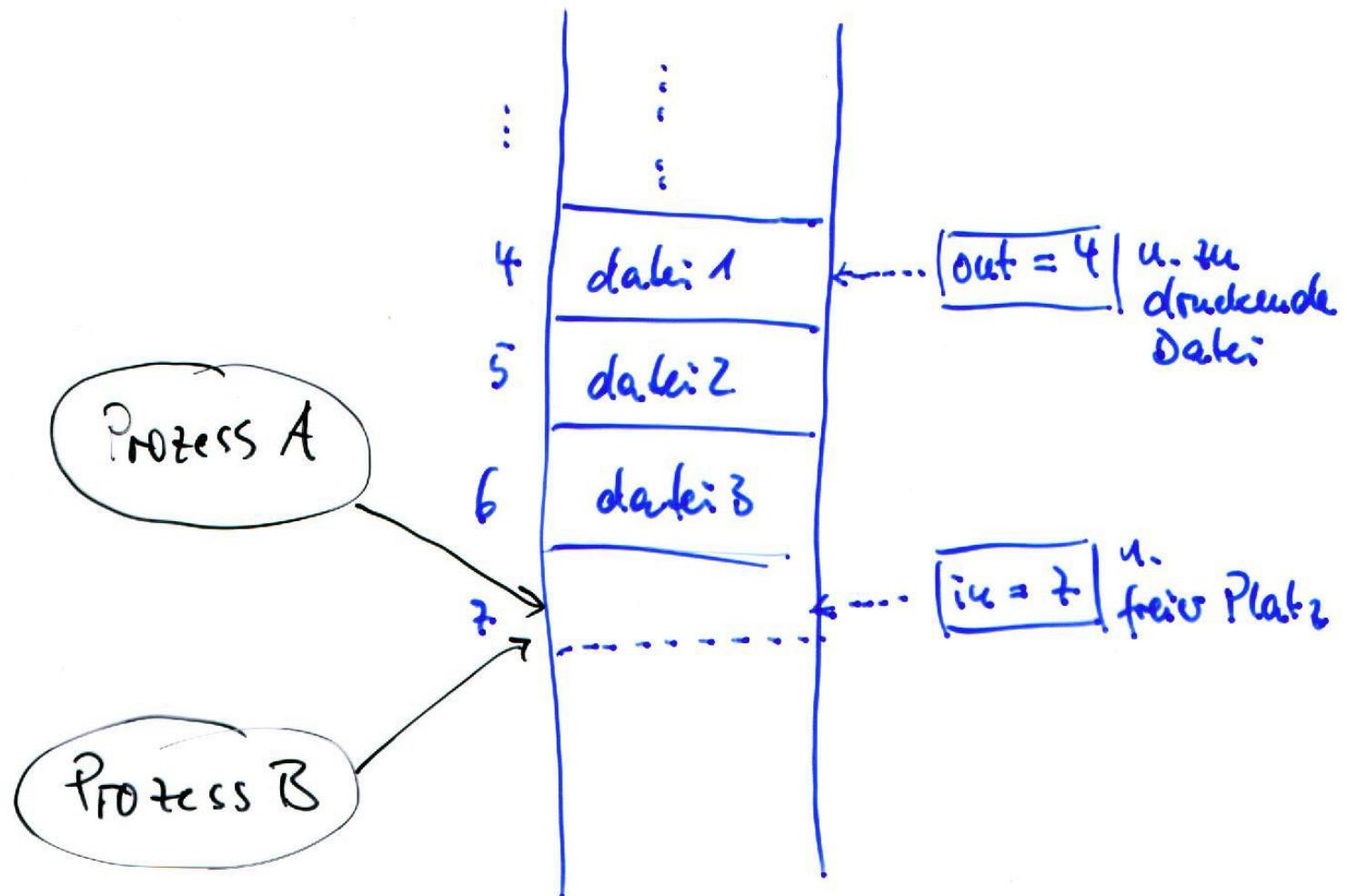
- Zeitkritische Abläufe (race conditions)
- Kritische Bereiche
- Wechselseitiger Ausschluss
- Schlafen und Aufwecken
- Semaphore
- Monitore

Zeitkritische Abläufe: Probleme bei gemeinsam genutzttem Speicher (HS oder Dateien)
Mehrere Prozesse lesen/schreiben gemeinsam genutzte Daten, Ergebnisse hängen von zeitlicher Reihenfolge der Lese-/Schreiboperationen ab.

Beispiel: Drucker-Warteschlange

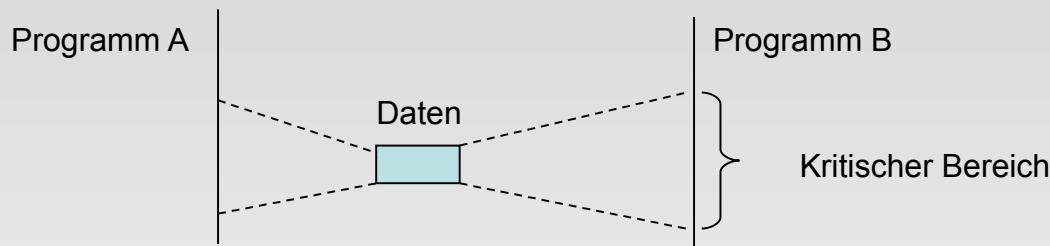
(Spooler)

Queue



Kritische Bereiche: Die Teile von Programmen, in denen auf gemeinsam benutzten Speicher zugegriffen wird.

Finden eines Verfahrens, um zu verhindern, dass zu einem Zeitpunkt mehr als ein Prozess gemeinsame Daten liest oder schreibt.



Wechselseitiger Ausschluss:

- Sperren aller Unterbrechungen vor Eintritt in kritischen Bereich
- Aktives Warten mittels Sperrvariablen (0 – kein P. im KB., 1 – P. im KB.)
- Schlafen und Aufwecken

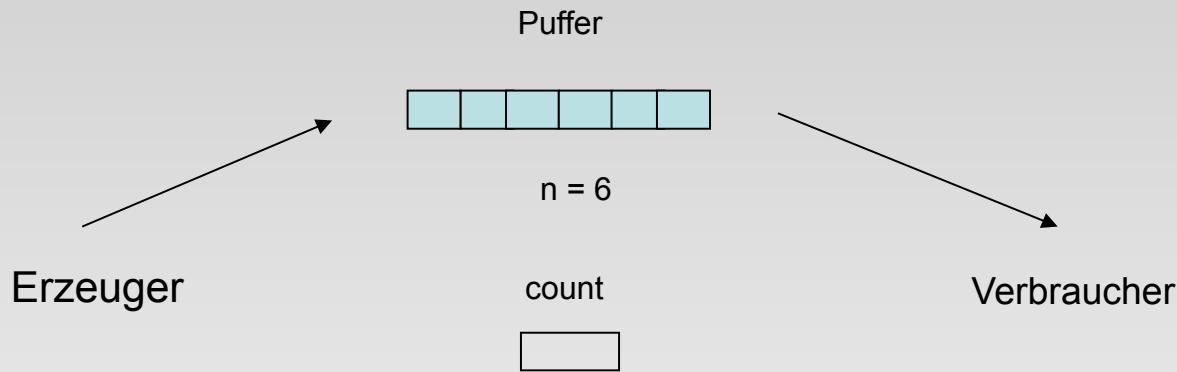
Aktives Warten verschwendet Prozessorzeit (Warteschleife, falls Eintritt in den kritischen Bereich nicht möglich ist).

„Blockieren“, wenn ein Prozess nicht in den Kritischen Bereich eintreten kann:

Der Aufrufer wird suspendiert, bis er von einem anderen Prozess aufgeweckt wird.

→ Ereigniszählung erforderlich

Beispiel: Das Erzeuger-Verbraucher-Problem



- im Zustand $\text{count}=0$ liest der Verbraucher count .
- Verbraucher startet den Erzeuger und will sich schlafen legen.
- bevor der Verbraucher sich schlafen legen kann, wird der Verbraucher-Prozess unterbrochen.
- der Erzeuger wird geweckt, erhöht count ($\text{count}++$) und weckt den Verbraucher.
- der Verbraucher schläft noch nicht, **das Wecksignal bleibt wirkungslos**.
- der Verbraucherprozess wird fortgesetzt, der Verbraucher legt sich schlafen.
- der Erzeuger füllt den Puffer und legt sich schlafen.
- beide Prozesse schlafen für immer → **Deadlock**

Messages

- ▶ Nachrichtenaustausch über eine im BS-Kern verwaltete Datenstruktur.
Systemaufrufe:

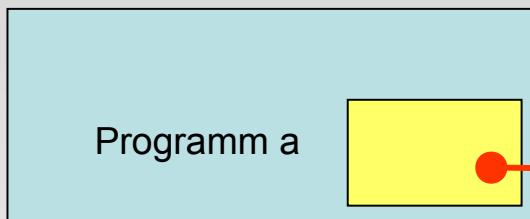
```
send(ziel, &message)  
receive(quelle, &message)
```

- ▶ Messages werden vom Sender-Prozess in eine Message-Queue geschrieben und vom Empfänger-Prozess dort abgeholt („Mailbox“).
- ▶ Falls keine Nachricht verfügbar ist, kann der Empfänger blockieren, bis eine Nachricht eintrifft.
- ▶ Messages können mit einem Typ versehen werden. Beim Empfang von Nachrichten kann dann auf die älteste Nachricht eines bestimmten Typs zugegriffen werden.

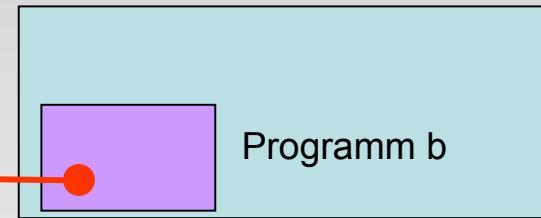
Sockets

- ▶ repräsentieren Kommunikationsendpunkte
- ▶ IP-Adresse + Port-Nummer

Rechner A



Rechner B



Einrichten eines Socket:

Systemruf

`sd = socket(...)`

Stream-Sockets (feste Verbindung) vs. Datagramm-Sockets (keine feste Verbindung)

Datenaustausch:

`sendto(...)`

`recvfrom(...)`

Arbeit mit Stream-Sockets

bind:
Adresse an Socket
binden

listen:
Teilt Kernel mit,
dass Anzahl von
Verbindungen
akzeptiert werden.

accept:
Warten auf
Verbindungswünsche

Prozeß 1 (Server)

socket(...)
bind(...)
listen(...)
accept(...)

accept blockiert den Server --->

connect(...)
←---- connect entblockiert den Server

send(...) -----> recv(...)

recv(...) <----- send(...)

:

close()

Prozeß 2 (Client)

socket(...)

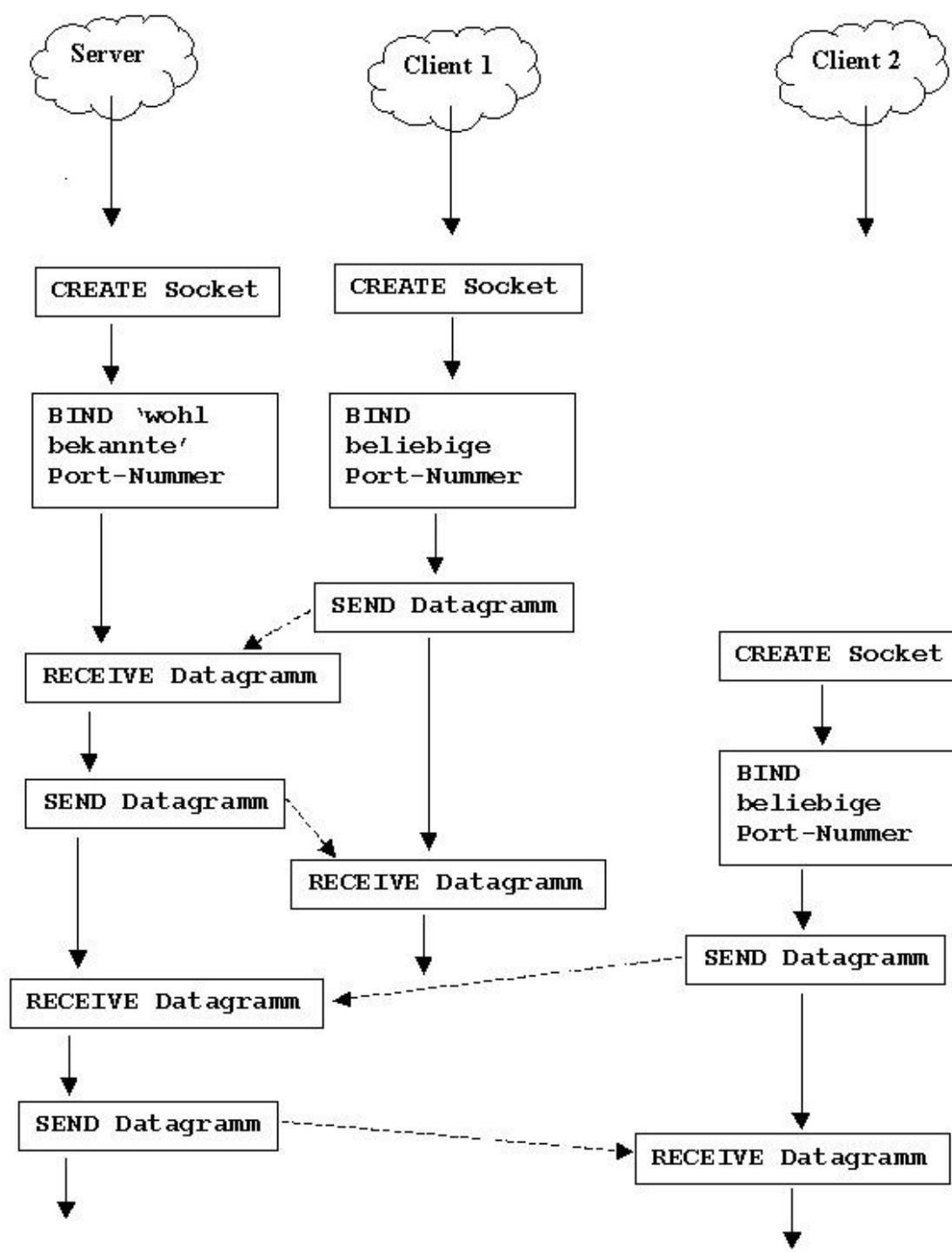
connect(...)

:

close()

Arbeit mit Datagramm-Sockets

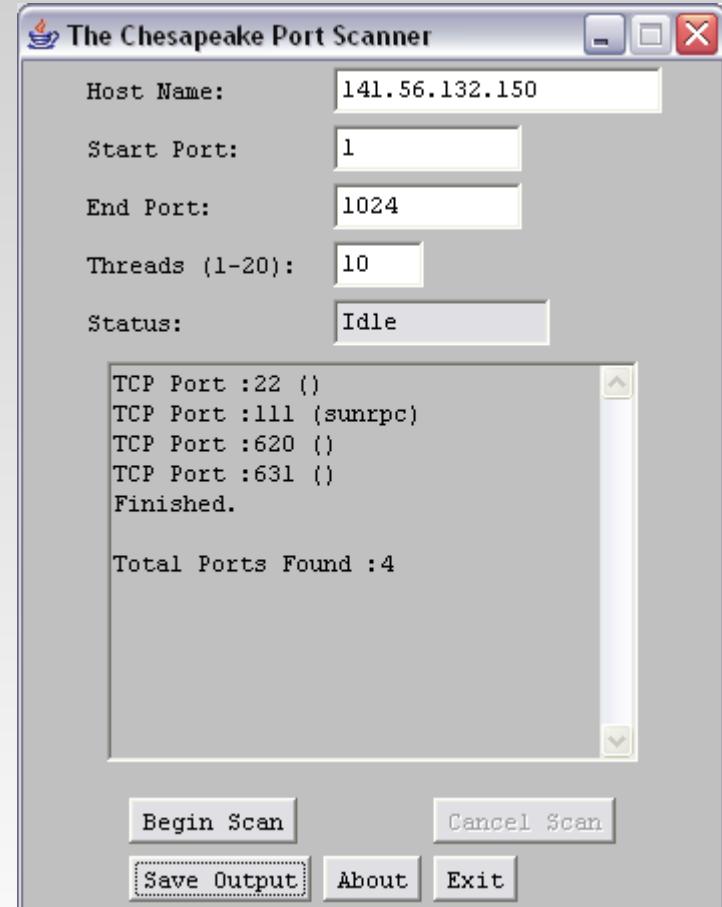
Bei einer verbindungslosen Kommunikation muss der Client zuerst ein Datagramm an den Server senden, sonst weiß der Server nichts von der Existenz des Clients



Portbelegungen - Portscanner

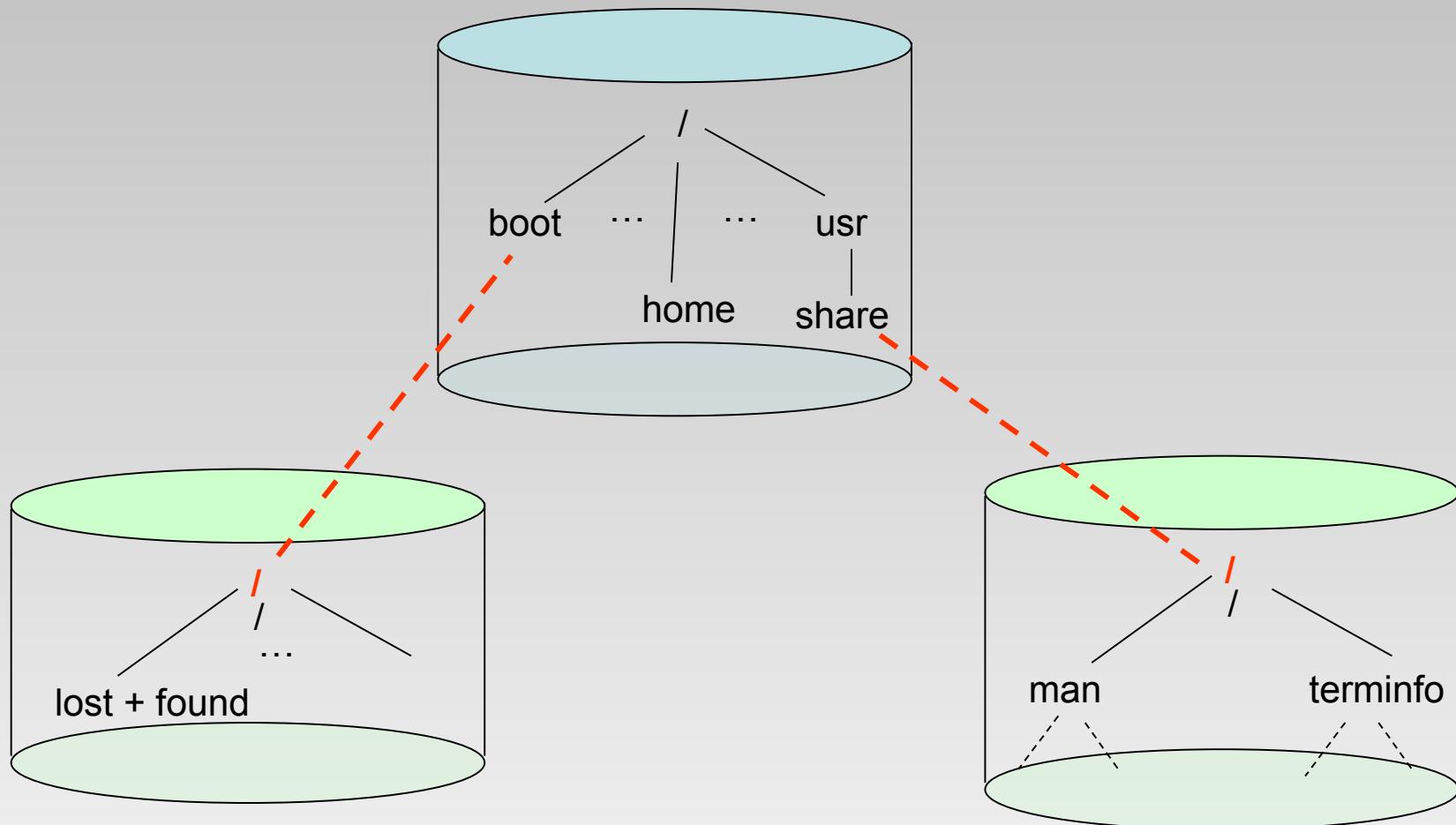
Portnummern: /etc/services bzw. www.isi.edu/in-notes/rfc1700.txt

Port	Protokoll	Name	Beschreibung
88	UDP	kerberos	Der offizielle Kerberos-Port. Sie müssen diesen Port durchschleusen, wenn Sie externen Login auf Ihrer Anlage gestatten wollen, sei es mit direkter oder Interrealm-Authentifikation, ansonsten blockieren (Siehe 13.2). Das selbe gilt für Port 750, den originalen Kerberos-Port. Blockieren Sie auch Port 749 und 751, den jetzigen und ursprünglichen Kerberos-Port zum Passworttausch. Die Ports für die Kerberos-geschützten Dienste sind jedoch wahrscheinlich sicher.
95	TCP	supdup	Außer durch Hacker kaum genutzt. Noch eine prima Port für einen Alarm.
109	TCP	pop-2	Blockieren, außer jemand will von draußen seine Mail lesen.
110	TCP	pop-3	Dito.
111	UDP, TCP	sunrpc	Blockieren, denken Sie aber daran, daß Angreifer Ihren Ports dennoch durchmustern können (Siehe 2.5.1).
113	TCP	auth	In der Regel sicher. Wenn Sie ihn blockieren, dann senden Sie keinesfalls eine ICMP-Ablehnung (Siehe 7.3).
119	TCP	nntp	Wenn Sie ihn durchschleusen, sollten Sie nach Quell- und Zieladressen filtern (Siehe 2.8.2).
123	UDP	ntp	Sicher, falls Sie NTPs eigene Zugangskontrollen nutzen (Siehe 2.4.3).
144	TCP	NeWS	Ein Window-System – wie X11 behandeln.
161	UDP	snmp	Blockieren.
162	UDP	snmp-trap	Blockieren, es sei denn, Sie überwachen Router außerhalb Ihres Netzes.
177	UDP	xdmcp	Für X11-Logins. Natürlich blockieren.
512	TCP	exec	Blockieren. Mit einer Variante von <i>rcp</i> könnte er nützlich sein; wie es steht, wurde er bisher bloß vom Internet-Worm genutzt. Außerdem protokolliert er garnichts.
513	TCP	login	<i>Ohje</i> . Blockieren (Siehe 2.7).
514	TCP	shell	<i>Ohje-Ohje</i> . Protokolliert auch nichts. Blockieren (Siehe 2.7).



Dateisysteme

- ▶ Die UNIX-Dateihierarchie setzt sich aus mehreren Teilhierarchien zusammen, die sich auf verschiedenen Speichermedien (Festplatten, CD-ROMs, Floppy-Disks, über Netz erreichbare Systeme) befinden können.
- ▶ Die Abgrenzung der Teilhierarchien erfolgt dadurch, dass jede von ihnen einer anderen Gerätedatei zugeordnet ist. Eine so abgegrenzte Teilhierarchie heißt Dateisystem.
- ▶ Das Root-Dateisystem ist bereits beim Starten des Systems vorhanden, denn es befinden sich Dateien darauf, die für die Initialisierung des Betriebssystems benötigt werden.
- ▶ Alle anderen Dateisysteme werden erst nach dem Systemstart "montiert". Dabei werden die Dateisysteme in den sog. Montierungspunkten (das sind Verzeichnisse im Root-Dateisystem) angekoppelt. Nach dem Montieren bildet das Dateisystem eine Einheit.



```
~>cd /usr/share/man
```

```
~>/bin/mount
```

<code>/dev/hda3</code>	on	<code>/</code>	type	<code>ext2 (rw)</code>
<code>/dev/hda1</code>	on	<code>/boot</code>	type	<code>ext2 (rw)</code>

Erzeugen eines Dateisystems (nur Superuser)

```
mkfs      /dev/mfod      1000
```

Gerät (Diskette)	Anzahl der Blöcke
---------------------	----------------------

- ▶ Dateisystem enthält 1 leeres Verzeichnis
- ▶ Größe der i-Liste wird automatisch errechnet

An- und Abkoppeln nicht residernter Dateisysteme („montierte Dateisysteme“)

- Das Root-Dateisystem ist bereits beim Systemstart vorhanden
- Ankoppeln:

```
mount [gerätedatei datei [-r] ]
```

Bsp: `mount /dev/mfod /usr/otto/doku`

- Abkoppeln:

```
umount gerätedatei
```

Aufbau Dateisystem:

Folge frei
adressierbarer
Blöcke mit je
1024 Byte

Blocknr.

0

Boot - Block

1

Superblock

2

i - Liste

isize + 2

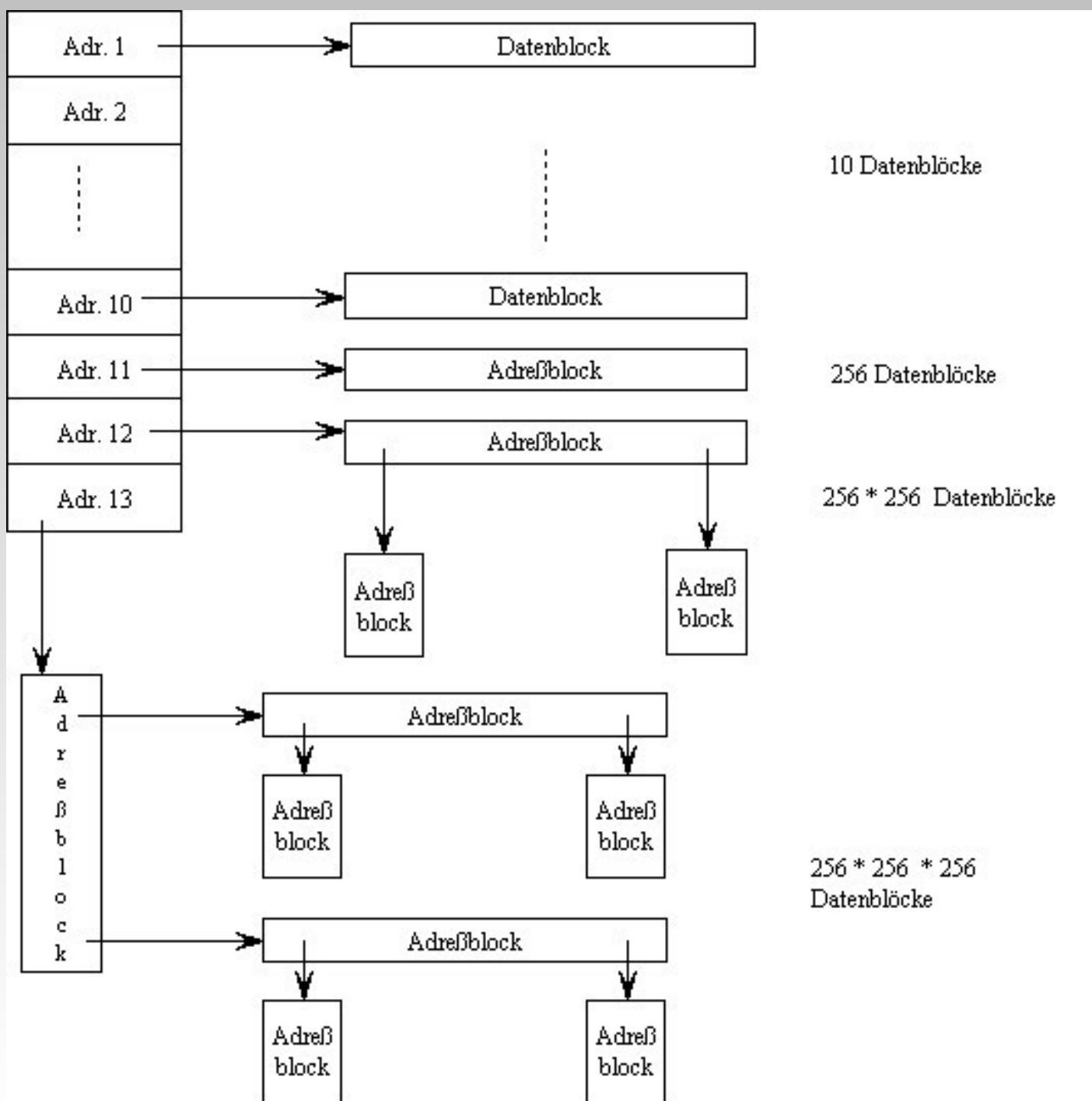
Datenblöcke

indirekte
Adressblöcke

freie Blöcke

- Jedes über
- Größe Dateisystem
- freie Blöcke

Blöcke können
jeweils bis zu
16 i-Bereiche
(= i-Nodes)
aufnehmen



Das Ein-/Ausgabesystem

- Teil des BS-Kerns, Verbindung zwischen Benutzer u. physikalischen Geräten
- Eine Gerätedatei repräsentiert ein (idealisiertes) logisches Gerät
- Geräte sind bestimmt durch:
 - Geräteklaasse
 - Hauptgerätenummer (major device number)

Beispiel:

```
~>ls -l /dev/mfod  
brw----- 1 root 4, 35 Apr 2005 /dev/mfod
```

Geräteklaasse

Hauptgerätenummer

b	<u>B</u> lock-E-/A-Gerät
c	<u>C</u> haracter- Block-E-/A-Gerät
p	fifo-Datei (<i>named pipe</i>)
l	symbolischer <u>L</u> ink
s	<u>S</u> ocket



Grundkonzepte Verteilter Systeme

- ▶ große Anzahl von Prozessoren über Hochgeschwindigkeitsnetz miteinander verbunden
- ▶ Netzwerkbetriebssystem vs. „echtes“ verteiltes System
- ▶ Protokollfamilien (Überblick)

TCP/IP Internet Protocol Suite

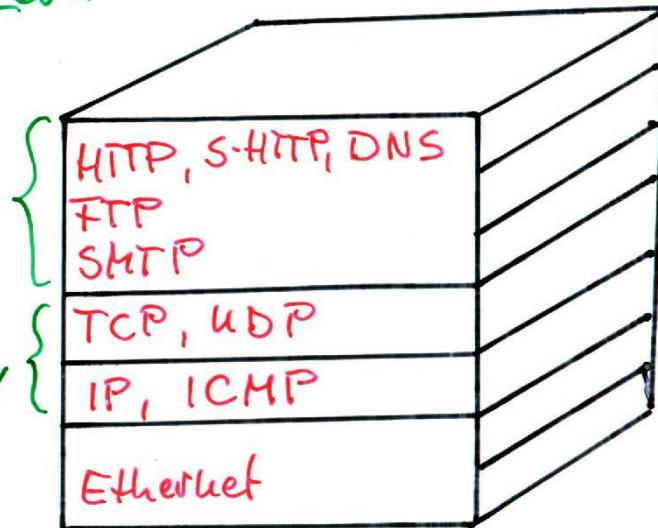
„Application“-Level

Anwendung

Transport

Internet

Netzwerk



„Internet Protocol“ - Level

Schichten im OSI - Referenzmodell

Anwendung

Darstellung (Präsentation)

Kommunikation (Sitzung)

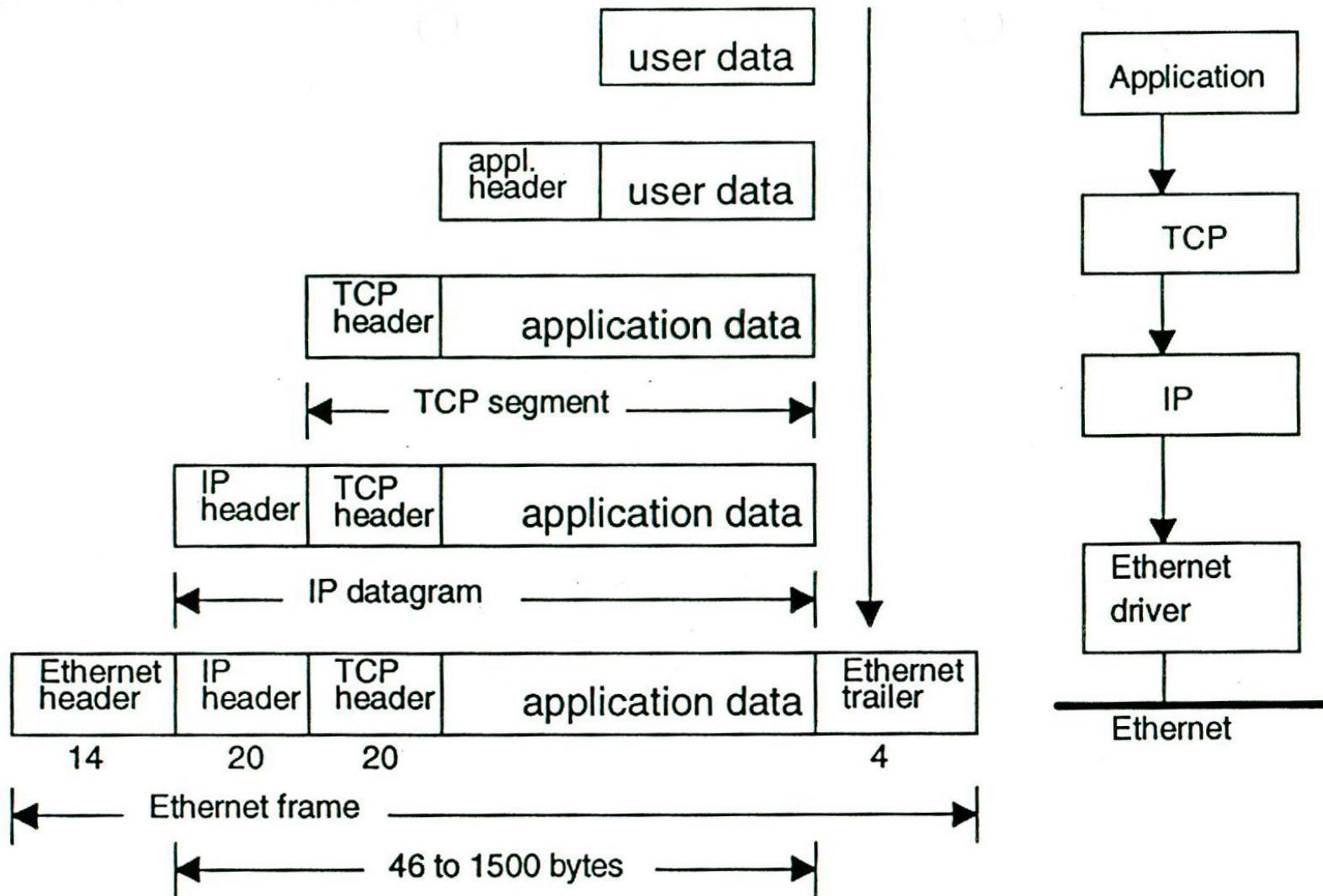
Transport

Vermittlung (Netzwerk)

Sicherung (Verbindung)

Bitübertragung

(Open Systems Interconnection)



IP-Adressen: 4 Zahlen zwischen 0 und 255, jeweils durch Punkt getrennt, weltweit eindeutig vergeben.
(IPv4)

Beispiel: 192.168.24.105
192.168.24 105 **Netzadresse**
255.255.255.0 **Rechneradresse**
 Subnetzmaske

Klasse	W-Werte	Netzwerk-ID	Host-Id	max Netzwerke	max Hosts
A	1-126	w	x.y.z	126	16.777.214
B	128-191	w.x	y.z	16384	65.534
C	192-223	w.x.y	z	2.097.151	254

Von den 255 theoretisch verfügbaren Adressen eines Klasse C-Netzwerkes wird

- 255 als Broadcast-Adresse verwendet
- 0 als Adresse für das Netzwerk verwendet (als Hostadresse unzulässig).

Adressvergabe:

- Jede IP-Adresse ist weltweit eindeutig und wird von der IANA an die Organisationen APNIC, ARIN und RIPE vergeben, die diese dann wiederum verteilen.

IANA (*Internet Assigned Numbers Authority*) :

organisatorisch Unterabteilung der ICANN

(*Internet Corporation for Assigned Names and Numbers*)

APNIC (*Asia-Pacific Network Information Center*)

ARIN (*American Registry for Internet Numbers*)

RIPE NCC (*Reseaux IP Europeens*)

LACNIC

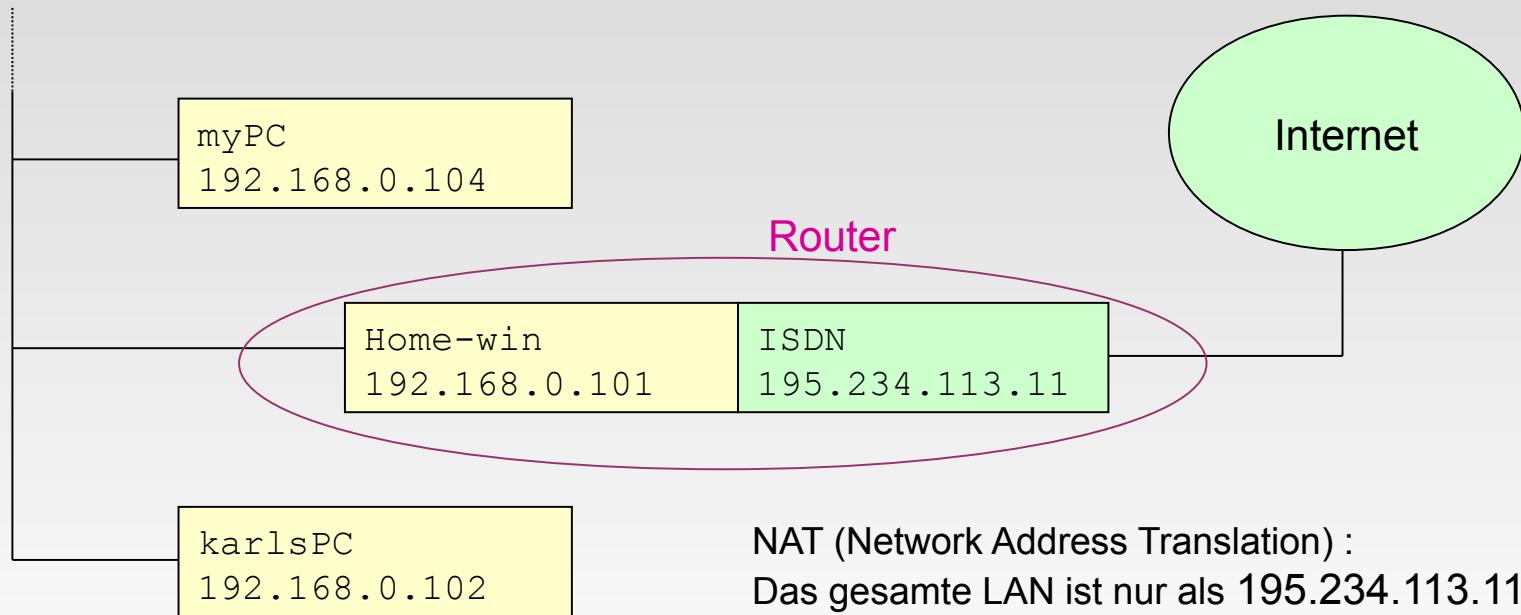
AfriNIC

- Für jede der Klassen (A, B, C) gibt es Bereiche, die nicht zentral vergeben werden, frei verfügbar (z.B. 192.168.0.0 – 192.168.255.0).

Adressübersetzung

- ▶ Um Datenpakete aus Netzwerken weiterleiten zu können, werden Router eingesetzt.
- ▶ Routing ist das Weiterleiten von Datenpaketen aus einem Netzwerk in ein anderes.

LAN 192.168.0.0



Crypto system

M : plaintext space

C : ciphertext space

K : key space

E = Encrypt

D = Decrypt

$$E : M \times K \rightarrow C$$

$$D : C \times K \rightarrow M$$

M : plaintext message

C : ciphertext message

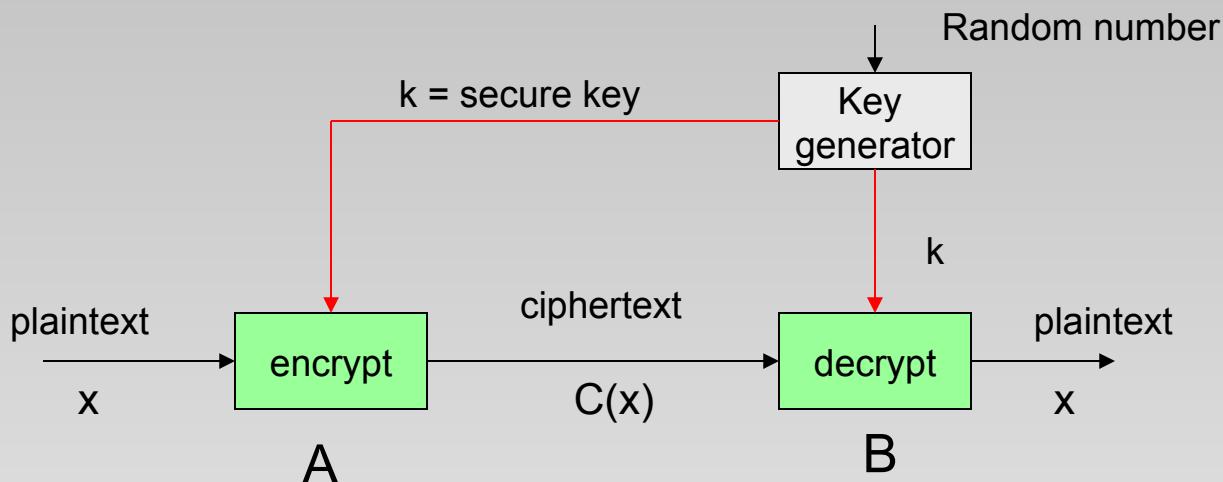
K : key

$$E(M) = C$$

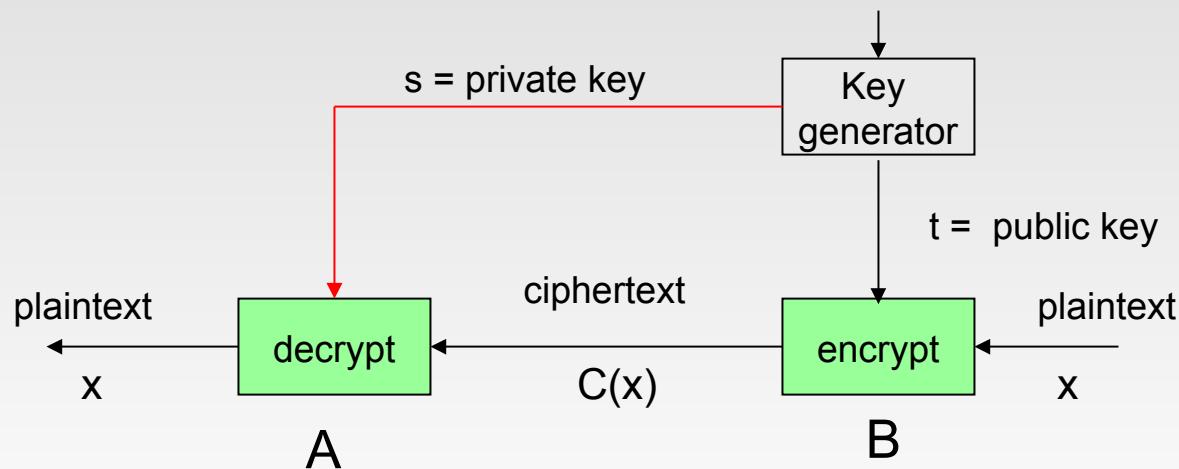
$$D(C) = M$$

$$D_K(E_K(M)) = M \quad | \quad \forall M \in M$$

symmetric encryption



asymmetric encryption



Schlüsselaustausch nach Diffie/Hellman (1976)

Die Kommunikationspartner A und B können jeder einen gemeinsamen Schlüssel K erzeugen, ohne geheime Informationen austauschen zu müssen.

Beispiel:

A

A und B vereinbaren zwei Zahlen $g = 4$ und $p = 11$

A wählt seinen privaten Schlüssel $X_A = 3$

A berechnet seinen öffentlichen Schlüssel

$$\begin{aligned} Y_A &= g^{X_A} \bmod p \\ &= 4^3 \bmod 11 = 64 \bmod 11 = 9 \end{aligned}$$

A sendet Y_A an B

A berechnet den Sitzungsschlüssel K:

$$K = Y_B^{X_A} = 3^3 \bmod 11 = 27 \bmod 11 = 5$$

B

B wählt seinen privaten Schlüssel $X_B = 4$

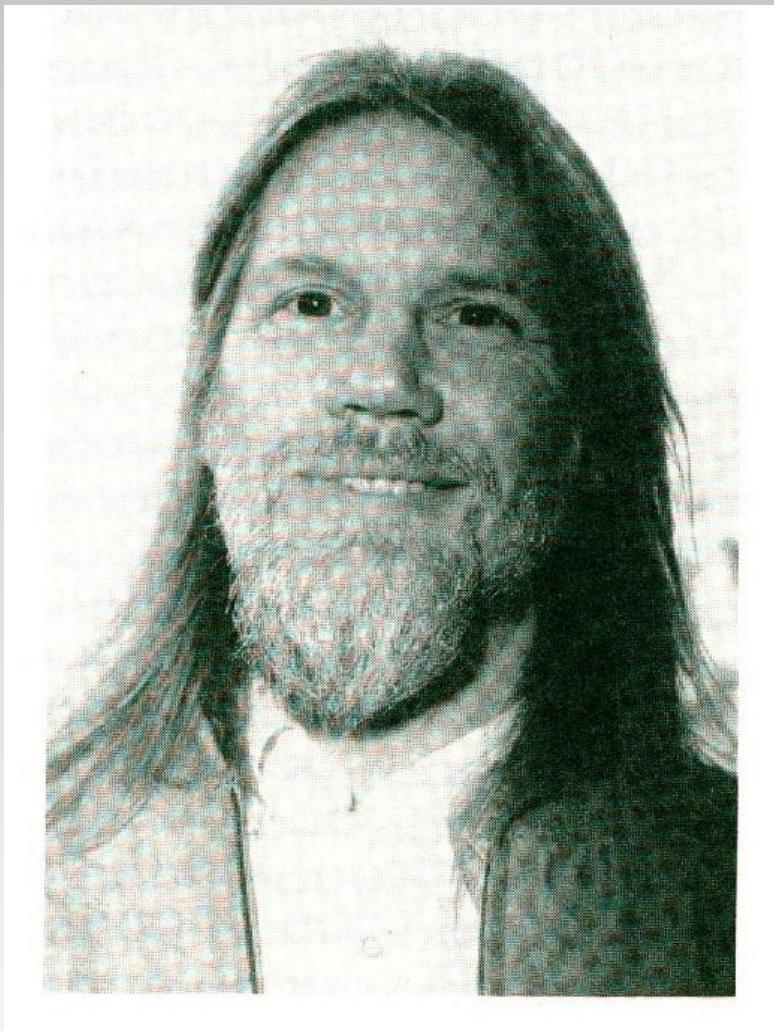
B berechnet seinen öffentlichen Schlüssel

$$\begin{aligned} Y_B &= g^{X_B} \bmod p \\ &= 4^4 \bmod 11 = 256 \bmod 11 \\ &= 3 \end{aligned}$$

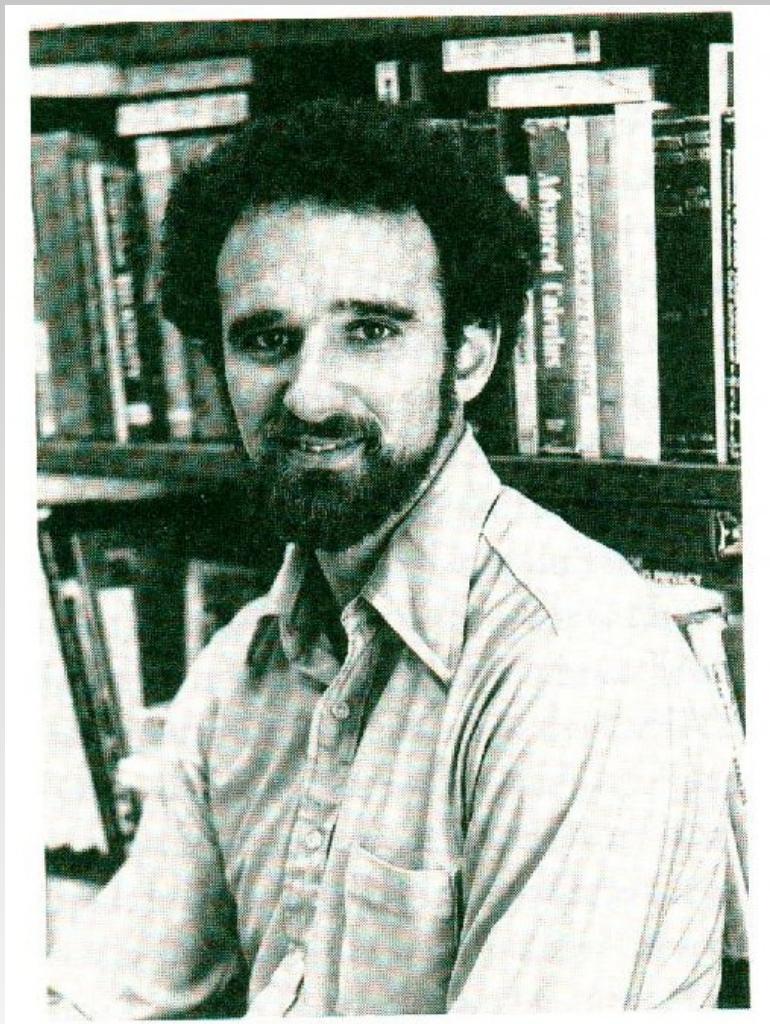
B sendet Y_B an A

B berechnet den Sitzungsschlüssel K:

$$K = Y_A^{X_B} = 9^4 \bmod 11 = 6561 \bmod 11 = 5$$

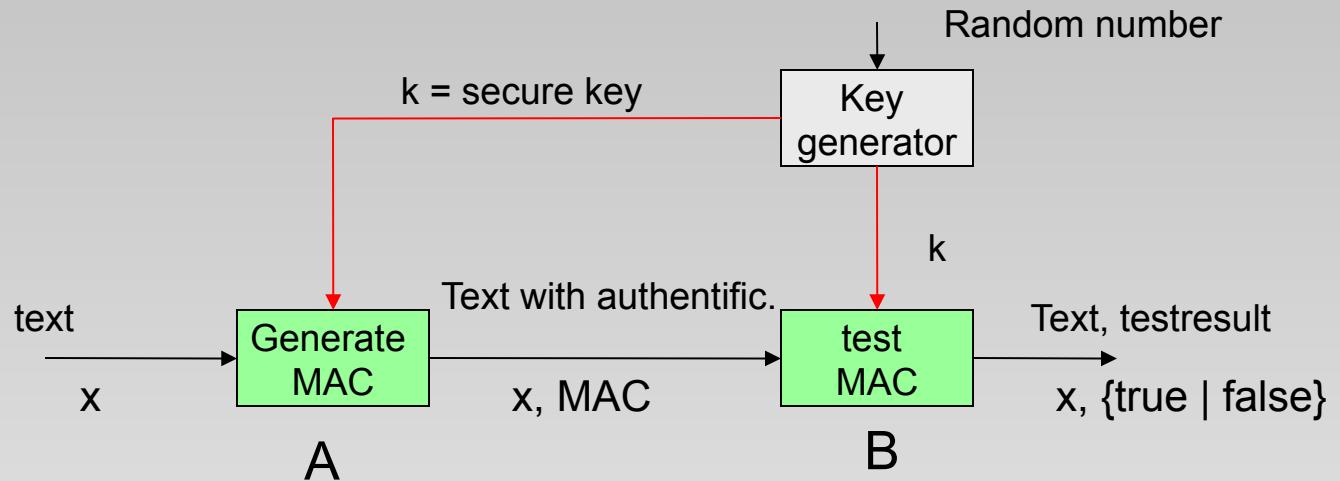


Whitfield Diffie, * 1944

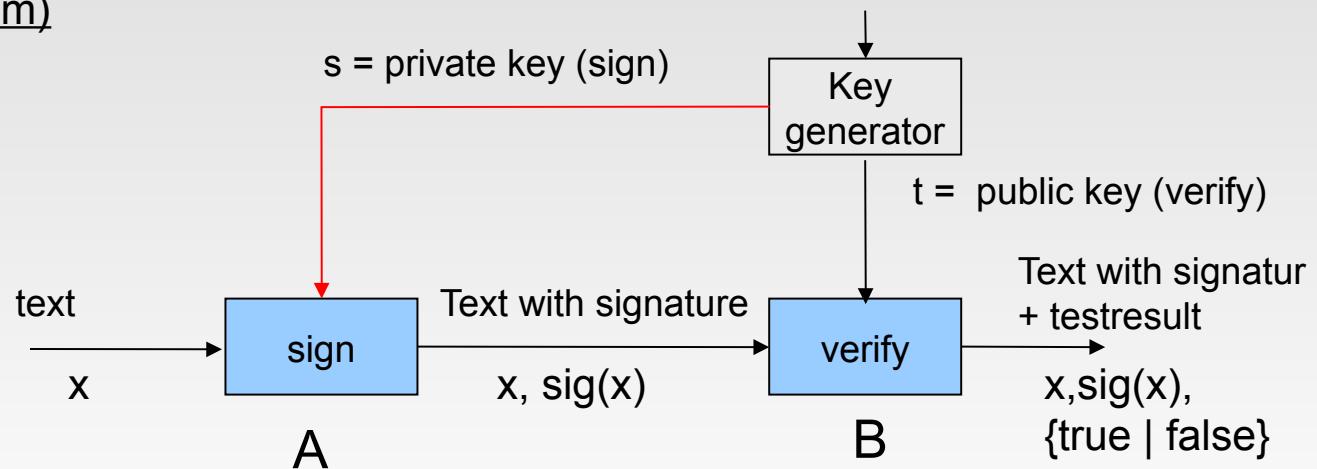


Martin Hellman, * 1946

symmetric authentification



asymmetric authentication (signature system)



Verteilte Dateisysteme

Konzepte

- Dateidienste (Beschreibung verfügbarer „Primitiven“)
 - Dateiserver (Prozess, der Implementationen des Dienstes realisiert)

Komponenten

- Dateidienst (Dateien lesen, schreiben)
 - Verzeichnisdienst (Verzeichnisse erzeugen, verwalten, hinzufügen, entfernen von Dateien)

Datendienst-

schnittstelle :

- Upload/Download - Modell
- Modell des entfernten Zugriffs

Verzeichnisdienst-

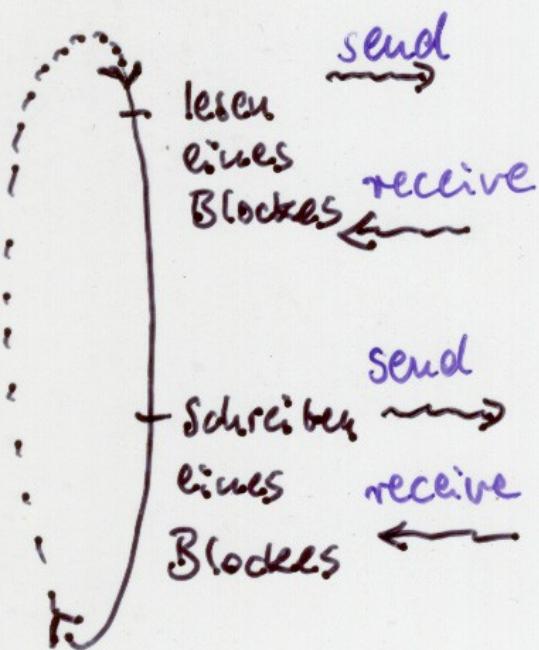
schnittstelle :

- Syntax für Aufräen von Namen
- durch Links werden aus
Bäumen Graphen
(Problem in vert. Systemen!)

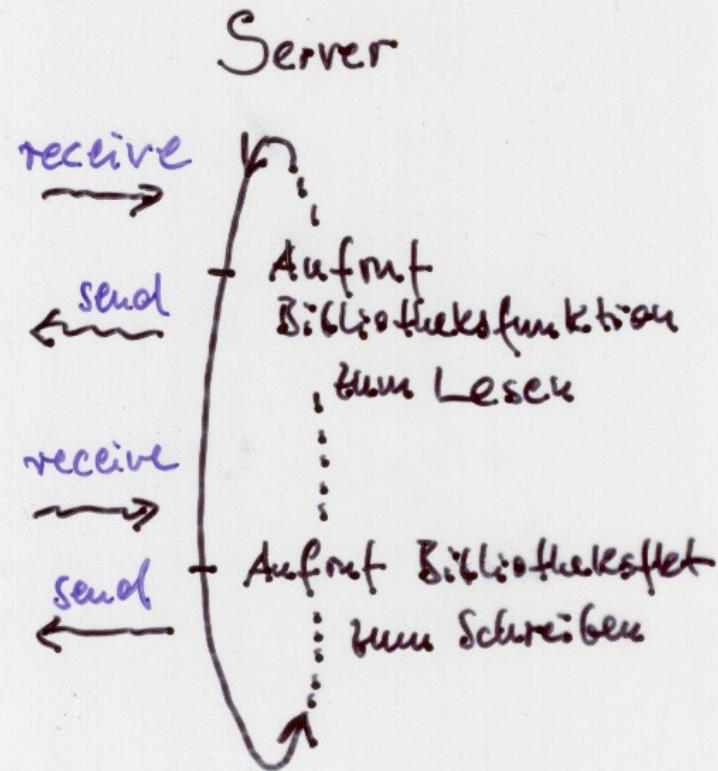
Beispiel: Client + Datei-Server

copy (source, destination)

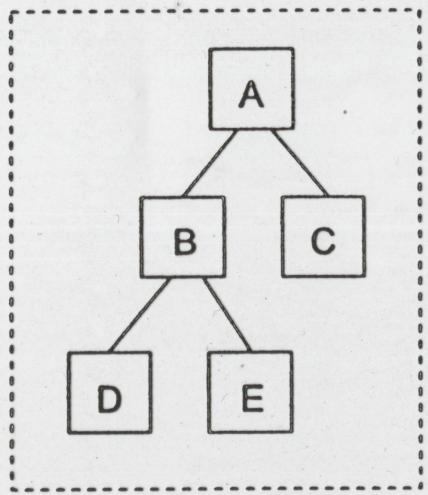
Client



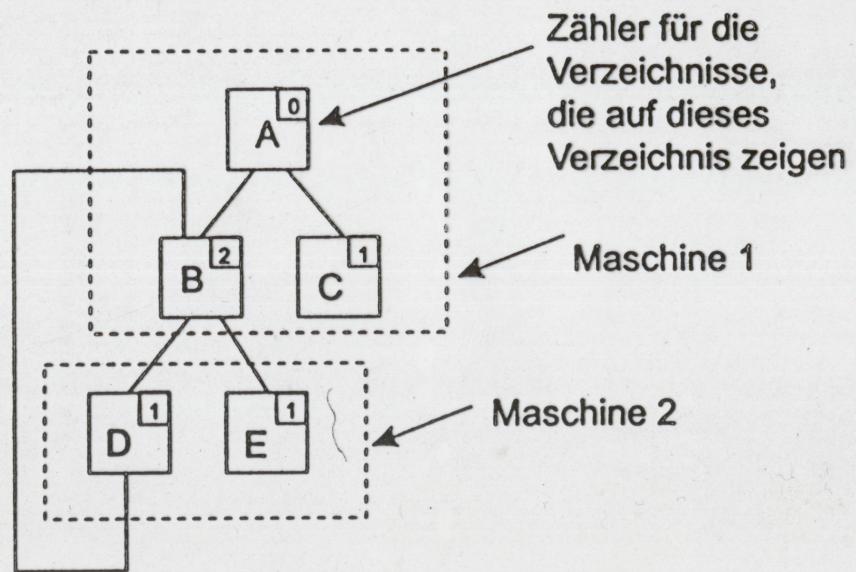
Server



- (a) Ein Verzeichnisbaum auf einer Maschine
(b) Ein Verzeichnisgraph, der auf zwei Maschinen verteilt ist



(a)



(b)

Beispiel:

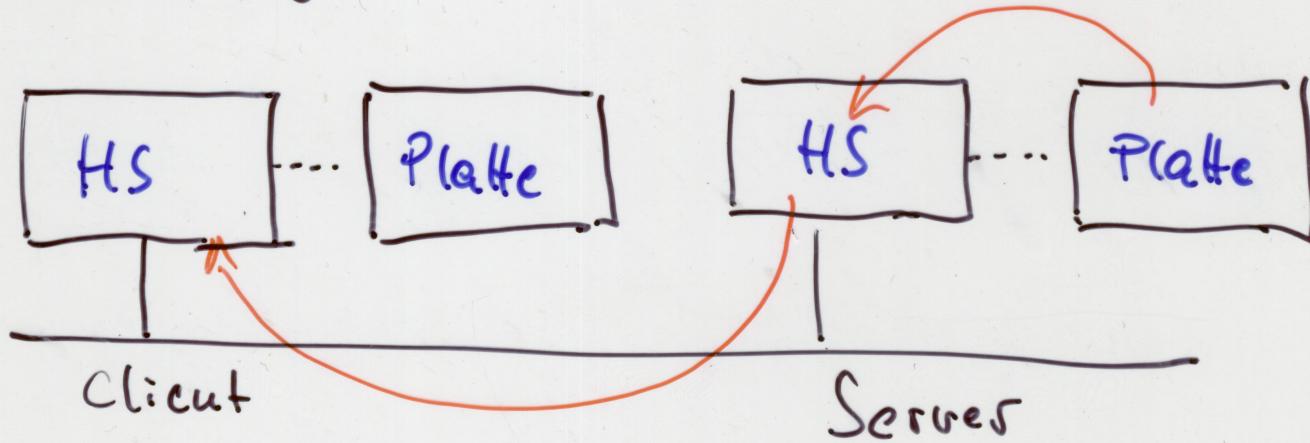
NFS (Network File Sharing, Fa. Sun)

- NFS-Server exportiert ein oder mehrere Verzeichnisse
 - NFS-Clients importieren Verzeichnisse
- (Dateisystem wird anderen Rechnern durch Eintrag in die Tabelle /etc(exports verfügbar gemacht, NFS-Clients „hängen Dateisysteme von Servern mit mount ein“)

NFS definiert zwei Protokolle:

1. zum Importieren (Datei-“Handle“ wird geliefert)
2. Behandlung der Zugriffe auf Dateien u. Verzeichnisse

Caching



- Dateien auf Platte des Servers bereitstellen
 - a) Übertragung in HS-Server
 - b) u. ins Netz zum Client

- Caching im fIS des Servers
- Caching im fIS des Clients
 - innerhalb Adressraum Benutzerprozess
 - im kernel
 - spezieller Cache - Manager - Prozess auf Benutzerebene

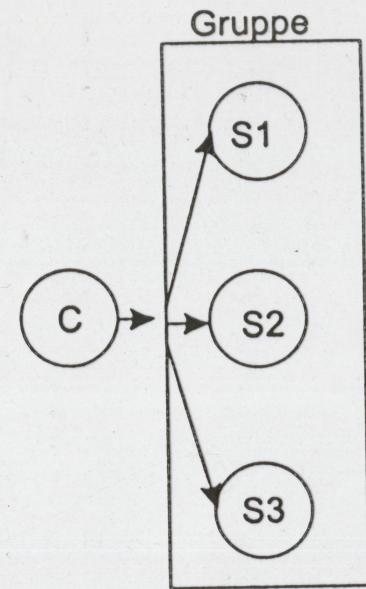
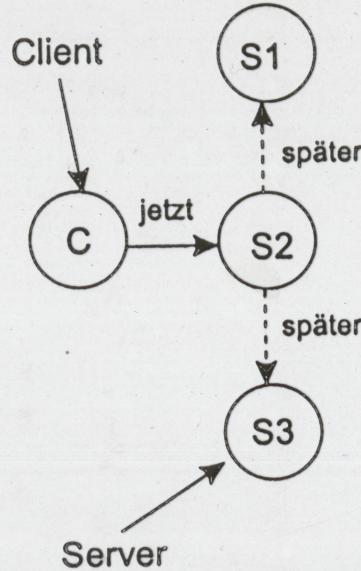
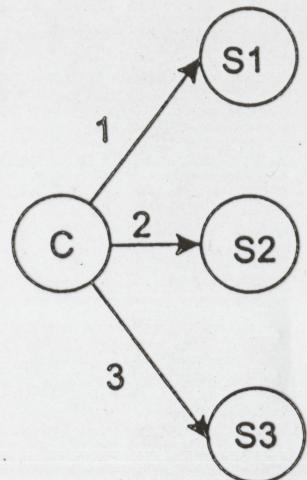
Replikation

- ▶ Dienst, der von verteilten Dateisystemen zur Verfügung gestellt wird
- ▶ Verwaltung mehrerer Kopien ausgewählter Dateien
- ▶ Gründe: Sicherheit, Verfügbarkeit

Wege der Replikation:

- Der Programmierer steuert den Vorgang: **explizite Replikation**
- Der Server erzeugt später selbst Repliken, ohne den Programmier **davon** in Kenntnis zu setzen: **langsame (träge) Replikation**
- „write“-Systemrufe gleichzeitig an alle Server übertragen: **mittels einer Gruppe**

- (a) Explizite Datei-Replikation
- (b) Träge Datei-Replikation
- (c) Datei-Replikation mit Hilfe einer Gruppe



Datei	1.14	2.16	3.19
prog.c	1.21	2.43	3.41

symbolischer
Name

mehrere binäre Adressen
(für S1, S2 und S3)

(a)

(b)

(c)

Kommunikation in verteilten Systemen

```
ping host
```

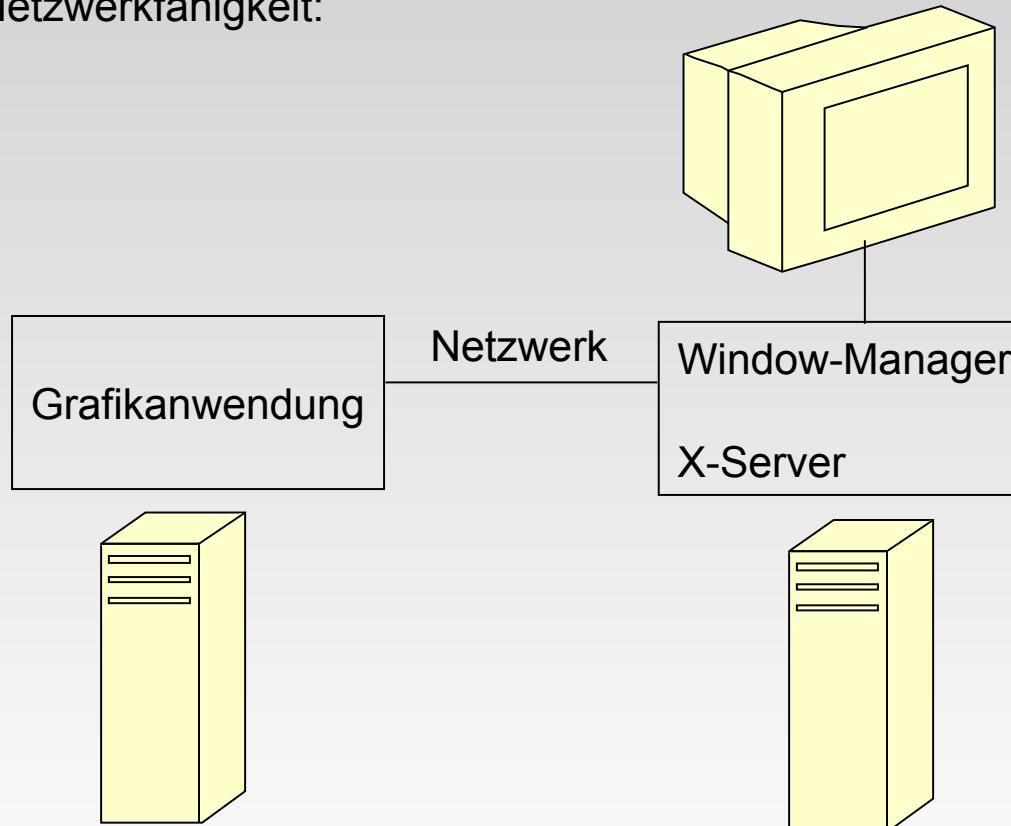
- `finger` überprüft die Netzwerkverbindung zu entfernten Computern

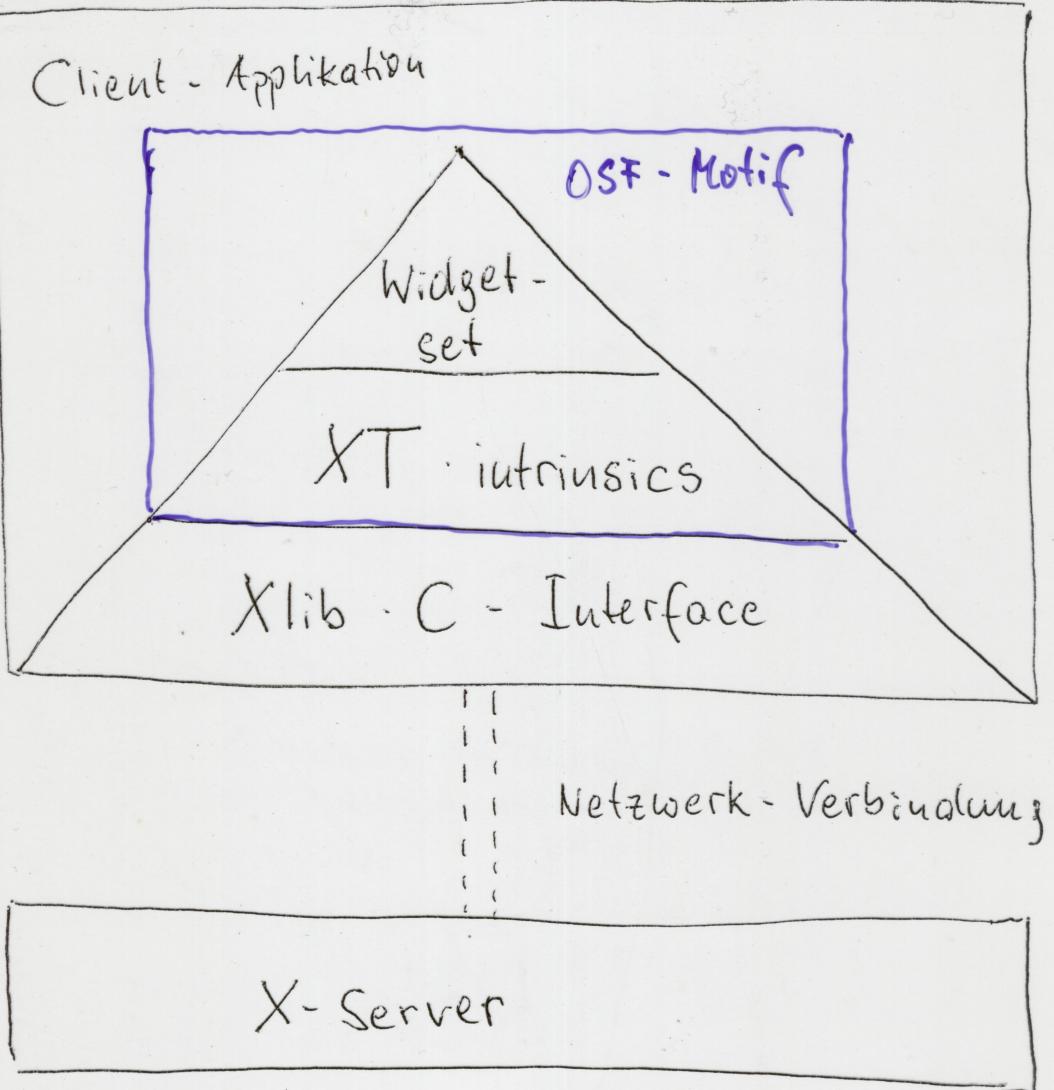
```
finger [ user ... ]
```

- `finger` liefert Informationen zu Nutzern (lokal oder remote)
- Gibt vom Nutzer in einer Datei `.plan` im Home-Verzeichnis angegebene Informationen aus
(ein einfacher Informationsdienst)

Das X Window System

- Basis für grafische Oberflächen unter UNIX und LINUX
(Sammlung von Protokollen und Funktionen)
- Netzwerkfähigkeit:





X Window System - Client/Server - Modell

