

# Systemprogrammierung

*Grundlagen von Betriebssystemen*

## Teil B – IV. Einleitung

Wolfgang Schröder-Preikschat

12. Mai 2020



## Agenda

---

Einordnung

Fallstudie

Speicherbelegung

Analyse

Begriffsdeutung

Literaturauszüge

Zusammenfassung



## Einordnung

### Fallstudie

Speicherbelegung

Analyse

### Begriffsdeutung

Literaturauszüge

### Zusammenfassung



## Systemprogrammierung $\leadsto$ Betriebssysteme

- Infrastruktursoftware für Rechensysteme
  - was Betriebssysteme sind, hat schon „Glaubenskriege“ hervorgerufen
    - das Spektrum reicht von „Winzlingen“ bis hin zu „Riesen“
    - simple Prozeduren  $\Leftrightarrow$  komplexe Programmsysteme
  - entscheidend ist, dass Betriebssysteme nie dem Selbstzweck dienen
- jedes Rechensystem wird durch ein Betriebssystem betrieben
  - Ausnahmen bestätigen die Regel...



- Betriebssysteme sind unerlässliches **Handwerkszeug** der Informatik
  - mit dem umzugehen ist zur Benutzung eines Rechensystems
  - das gelegentlich zu beherrschen, anzupassen und auch anzufertigen ist



- IBM: z/VM (vormals VM/CMS), z/OS



- DEC: VAX/VMS



- DOS (16-/32-Bit)-, NT- und CE-Linie



<sup>1</sup>Shakespeare zur Unabänderlichkeit oder Beeinflussbarkeit des Schicksals.

## Variantenvielfalt

„Kleinvieh macht Mist“

- funktionale und nichtfunktionale Eigenschaften von Betriebssystemen werden durch die **Anwendungsdomäne** vorgegeben
    - gelegentlich passen bestehende „unspezifische“ Lösungen (z.B. Linux)
    - viel häufiger sind jedoch **anwendungsspezifische Lösungen** erforderlich
  - Systeme für den **Allgemeinzw**eck
    - Rechensysteme für die gängigsten Aufgaben einer Anwendungsdomäne
    - die Domäne der umseitig (S. 5) genannten Vertreter
  - Systeme für den **Spezialzweck**
    - Rechensysteme zur Steuerung oder Regelung „externer“ Prozesse
    - für gewöhnlich mit vorhersagbarem Laufzeitverhalten (**Echtzeitsysteme**)
- **eingebettete Systeme** (vgl. [11])
- das „Klugfon“ nicht mitgerechnet



# Gliederung

Einordnung

Fallstudie

Speicherbelegung

Analyse

Begriffsdeutung

Literaturauszüge

Zusammenfassung



## Untersuchung eines Phänomens

Ganzheitsmethode<sup>2</sup>

Die Funktionsweise (auch) von Betriebssystemen zu verstehen, hilft **bemerkenswerte Erscheinungen** innerhalb eines Rechensystems zu begreifen und in ihrer Bedeutung besser einzuschätzen.

- **Eigenschaften** (*features*) von Betriebssystemen erkennen:
  - funktionale** ■ Verwaltung der Betriebsmittel (Prozessor, Speicher, Peripherie) für eine Anwendungsdomäne
  - nichtfunktionale** ■ dabei anfallender Zeit-, Speicher-, Energieverbrauch
  - d.h., **Gütemerkmale** einer Implementierung
- aus den funktionalen Eigenschaften resultierendes **Systemverhalten** unterscheiden von Fehlern (*bugs*) des Systems
  - um Fehler kann ggf. „herum programmiert“ werden
  - um zum Anwendungsfall unpassende Eigenschaften oft jedoch nicht

<sup>2</sup>Analytische Lernmethode, die die Vermittlung eines Stoffes als Gesamtheit in den Mittelpunkt stellt, um dann konstituierende Elemente weiter zu untersuchen.



## ■ zeilenweises Vorgehen: Spaltenelemente j von Zeile i aufzählen

```

1 void by_row (int mx[], unsigned int n, int v) {
2     unsigned int i, j;
3     for (i = 0; i < n; i++)
4         for (j = 0; j < n; j++)
5             mx[i * n + j] = v;          /* "mx[i][j]" = v */
6 }

```

## ■ spaltenweises Vorgehen: Zeilenelemente i von Spalte j aufzählen

```

7 void by_column (int mx[], unsigned int n, int v) {
8     unsigned int i, j;
9     for (j = 0; j < n; j++)
10        for (i = 0; i < n; i++)
11            mx[i * n + j] = v;          /* "mx[i][j]" = v */
12 }

```

### Gemeinsamkeit und Unterschied (von vertauschten Zeilen abgesehen)

**funktional** erfüllen beide Varianten denselben Zweck

**nichtfunktional** unterscheiden sie sich ggf. im Laufzeitverhalten

<sup>3</sup>Erst zur Laufzeit bekannte Feldgrenzen (vgl. S. 34). Beachte:  $mx[] \equiv *mx$ .



# Instanzenbildung und Initialisierung der Matrix

```

13 #include <stdlib.h>
14
15 main (int argc, char *argv[]) {
16     if (argc == 3) {
17         unsigned int n = atol(argv[2]);
18         if (n != 0) {
19             int *mx = (int*)calloc(n*n, sizeof(int));
20             if (mx != 0) {
21                 if (*argv[1] == 'R') by_row(mx, n, 42);
22                 else by_column(mx, n, 42);
23                 free(mx);
24             }
25         }
26     }
27 }

```

16 Verwendung: `<name> <way> <count>`

17 Größe einer Zeile bzw. Spalte einlesen

19 Matrixspeicher anfordern und löschen

21–22 zeilen-/spaltenweise Initialisierung mit 42

23 Matrixspeicher der Halde zurückgeben



| Betriebssystem         | Zentraleinheit                             | Speicher | by_row() | by_column() |     |
|------------------------|--|----------|----------|-------------|-----|
| Solaris                | $2 \times 1\text{ GHz}$<br>UltraSPARC IIIi | 8 GB     | 3.64r    | 27.07r      | (a) |
|                        |  |          | 2.09u    | 24.68u      |     |
|                        |  |          | 1.11s    | 1.10s       |     |
| Windows XP<br>(Cygwin) | $2 \times 3\text{ GHz}$<br>Pentium 4 XEON  | 4 GB     | 0.87r    | 11.94r      | (b) |
|                        |  |          | 0.65u    | 11.62u      |     |
|                        |  |          | 0.21s    | 0.21s       |     |
| Linux 2.6.20           | $2 \times 3\text{ GHz}$<br>Pentium 4 XEON  | 4 GB     | 0.89r    | 14.73r      | (c) |
|                        |  |          | 0.48u    | 14.34u      |     |
|                        |  |          | 0.40s    | 0.39s       |     |
|                        | $2 \times 2.8\text{ GHz}$<br>Pentium 4     | 512 MB   | 51.23r   | 39.84r      | (d) |
|                        |  |          | 0.47u    | 14.34u      |     |
|                        |  |          | 2.17s    | 2.09s       |     |
| Mac OS X 10.4          | 1.25 GHz<br>PowerPC G4                     | 512 MB   | 10.24r   | 106.72r     | (e) |
|                        |  |          | 0.69u    | 23.15u      |     |
|                        |  |          | 2.12s    | 17.08s      |     |
|                        | 1.5 GHz<br>PowerPC G4                      | 512 MB   | 10.11r   | 93.68r      | (f) |
|                        |  |          | 0.46u    | 23.71u      |     |
|                        |  |          | 2.08s    | 6.85s       |     |
|                        |  | 1.25 GB  | 2.17r    | 27.95r      | (g) |
|                        |  |          | 0.43u    | 22.35u      |     |
|                        |  |          | 1.50s    | 4.22s       |     |

<sup>4</sup>time ./main x 11174, mit  $x \in (\mathbb{R}, \mathbb{C})$

## Wo uns der Schuh drückt...

- (a)–(g) **Linearisierung** AuD, GRA
  - zweidimensionales Feld  $\mapsto$  eindimensionaler Arbeitsspeicher
- (a)–(g) **Zwischenspeicher (cache)** GRA
  - Zugriffsfehler (*cache miss*), Referenzfolgen
- (d) **Kompilierer & Magie** UEB
  - semantische Analyse, Erkennung gleicher Zugriffsmuster
- (d)–(f) **virtueller Speicher** SP
  - Seitenfehler (*pagefault*), Referenzfolgen
- (e)–(g) **Betriebssystemarchitektur** SP
  - Verortung der Funktion zur Seitenfehlerbehandlung

Sir Isaac Newton

*Was wir wissen, ist ein Tropfen, was wir nicht wissen, ist ein Ozean.*

Spalte  $j$  →

Zeile  $i$  ↓

|   |   |   |   |
|---|---|---|---|
| A | B | C | D |
| E | F | G | H |
| I | J | K | L |
| M | N | O | P |

zeilenweise Abspeicherung/Aufzählung

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

spaltenweise Abspeicherung/Aufzählung

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | E | I | M | B | F | J | N | C | G | K | O | D | H | L | P |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

- im Abstrakten könnte uns diese **Abbildung** egal sein
- im Konkreten jedoch nicht

| Abspeicherung | Aufzählung  |              |
|---------------|-------------|--------------|
|               | zeilenweise | spaltenweise |
| zeilenweise   | 😊           | 😞            |
| spaltenweise  | 😞           | 😊            |

### Fälle (a)–(g)

- Abspeicherung zeilenweise (C bzw. `main()`)
- Aufzählung zeilen- (`by_row()`) und spaltenweise (`by_column()`)



# Linearisierung II

## Referenzfolgen auf den Arbeitsspeicher

Entwicklung der Adresswerte  $A$  beim Zugriff auf die Elemente einer zeilenweise abgespeicherten Matrix mit Anfangsadresse  $\gamma$ :

| $i \setminus j$ | 0              | 1              | 2              | ...      | $N - 1$            |
|-----------------|----------------|----------------|----------------|----------|--------------------|
| 0               | 0              | 1              | 2              | ...      | $N - 1$            |
| 1               | $N + 0$        | $N + 1$        | $N + 2$        | ...      | $N + N - 1$        |
| 2               | $2N + 0$       | $2N + 1$       | $2N + 2$       | ...      | $2N + N - 1$       |
| $\vdots$        | $\vdots$       | $\vdots$       | $\vdots$       | $\ddots$ | $\vdots$           |
| $N - 1$         | $(N - 1)N + 0$ | $(N - 1)N + 1$ | $(N - 1)N + 2$ | ...      | $(N - 1)N + N - 1$ |

- $A = \gamma + (i * N + j) * \text{sizeof}(\text{int})$ , für  $i, j = 0, 1, 2, \dots, N - 1$

### Fälle (a)–(g)

- linear im homogenen Fall (`by_row()`), Abspeicherung und Aufzählung sind gleichförmig  $\leadsto$  **starke Lokalität**
- sprunghaft, sonst (`by_column()`)  $\leadsto$  **schwache Lokalität**



- **Normalfall:** Datum befindet sich im Zwischenspeicher
  - Zugriffszeit  $\approx$  Zykluszeit der CPU, Wartezeit = 0
- **Ausnahmefall:** Datum befindet sich *nicht* im Zwischenspeicher
  - Zugriffsfehler  $\leadsto$  Einlagerung (Zwischenspeicherzeile, *cache line*)
    - Zugriffszeit  $\geq$  Zykluszeit des RAM, Wartezeit  $> 0$
    - **schlimmster Fall** (*worst case*): Zwischenspeicher ist voll  $\mapsto$  **GAU**
      - Zugriffsfehler  $\leadsto$  Ein- und ggf. Auslagerung (Zwischenspeicherzeile)
        - Zugriffszeit  $\geq 2 \times$  Zykluszeit des RAM, Wartezeit  $\gg 0$
- die Effektivität steht und fällt mit der **Lokalität** der Einzelzugriffe
  - starke Lokalität erhöht die **Trefferwahrscheinlichkeit** erheblich

### Fälle (a)–(g)

- beide Varianten verursachen bei Ausführung den **GAU**
- `by_column()`  $\leadsto$  schwache Lokalität: **schlechte Trefferquote**



## Kompilierer

- **funktionale Eigenschaft**  $\mapsto$  „was“ etwas tut
  - beide Varianten tun das gleiche — nur in verschiedener Weise
- **nichtfunktionale Eigenschaft**  $\mapsto$  „wie“ sich etwas ausprägt
  - `by_row()` zählt Feldelemente entsprechend Feldabspeicherung auf
  - `by_row()` zeigt für gegebene Hardware günstigere Zugriffsmuster
  - ⋮
  - `by_row()` wird schneller als `by_column()` ablaufen können

### Fall (d): Beispiel eines wahren Mysteriums...

- `gcc -O -m32 -S` zeigte identischen Assemblersprachenkode
  - `by_column()` ist Kopie von `by_row()`
  - statische Analyse sagt gleiches Verhalten beider Varianten voraus
- Experiment brachte Messreihen mit extremen Ausschlägen hervor
  - „dynamische Umgebung“ verhält sich zugunsten von `by_column()`





Arbeitsspeicher hat mehr Kapazität als der Hauptspeicher<sup>a</sup>

<sup>a</sup>Begrifflich sind Arbeits- und Hauptspeicher verschiedene Dinge!

- Hauptspeicher ist Zwischenspeicher  $\mapsto$  **Vordergrundspeicher**
  - von Programm- bzw. Adressraumteilen eines oder mehrerer Prozesse
- ungenutzte Bestände im Massenspeicher  $\mapsto$  **Hintergrundspeicher**
  - z.B. Plattenspeicher, SSD oder gar Hauptspeicher anderer Rechner
- gleiches Problem wie beim Zwischenspeicher (vgl. S. 15)
  - **Seitenumlagerung** (*paging*)  $\leadsto$  zeitaufwändige Ein-/Ausgabevorgänge
  - Zugriffszeit verlangsamt sich um einige Größenordnungen: ns  $\leadsto$  ms

Fälle (a)–(g): Zwickmühle wegen Hauptspeicherkapazität...

(d)–(f) beide Varianten verursachen den GAU (S. 15)

- kontraproduktiver **Seitenvorabruf** (*prepaging*) SP2

sonst fallen „nur“ Einlagerungsvorgänge an

- Prozess zieht sein Programm selbst in den Hauptspeicher



## Betriebssystemarchitektur

*Schönheit, Stabilität, Nützlichkeit — Venustas, Firmitas, Utilitas:  
Die drei Prinzipien von Architektur [7]*

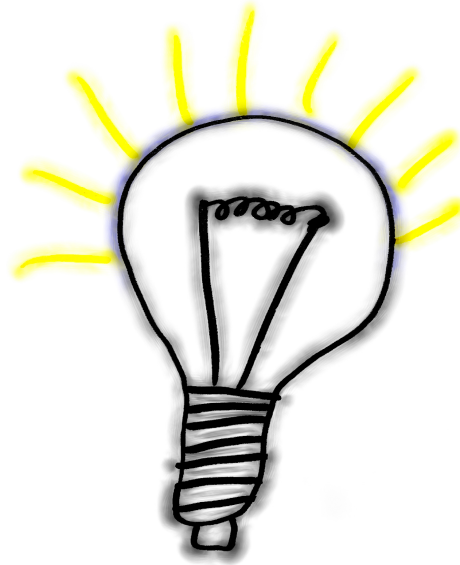
- Systemfunktionen sind architektonisch verschieden ausgeprägt
  - sie teilen sich dieselben **Domänen** oder eben auch nicht
  - bzgl. Adressraum, Ausführungsstrang, Prozessor oder Rechnersystem
- architektonische und funktionale Merkmale widersprechen sich nicht
  - beide Arten bewirken jedoch gewisse **nichtfunktionale Eigenschaften**
    - z.B. verursachen domänenübergreifende Aktionen ggf. **Mehraufwand**

Fälle (e)–(g)

- Mac OS X = NeXTStep  $\cup$  Mach 2.5  $\leadsto$  **mikrokernbasiert**
  - Systemfunktionen laufen als *Tasks* in eigenen Adressräumen ab
  - Tasks bieten Betriebsmittel für ggf. mehrere Ausführungsstränge
  - Ausführungsstränge sind die Zuteilungseinheiten für Prozessoren
- **externer Seitenabruf** (*external pager*) zur Seitenumlagerung
  - außerhalb des klassischen Kerns  $\leadsto$  domänenübergreifende Aktionen



„...und denen [...] ist ein Licht aufgegangen“<sup>5</sup>



<sup>5</sup>Matthäus 4.16

© wosch

SP (SS 2020, B–IV)

2.2 Fallstudie – Analyse

IV/19

## Gliederung

Einordnung

Fallstudie

Speicherbelegung

Analyse

Begriffsdeutung

Literaturauszüge

Zusammenfassung

© wosch

SP (SS 2020, B–IV)

3. Begriffsdeutung

IV/20

# Was macht ein Betriebssystem [aus] ?



## Nachschlagewerke

*Summe derjenigen Programme, die als **residenter Teil** einer EDV-Anlage für den Betrieb der Anlage und für die Ausführung der Anwenderprogramme erforderlich ist. [8]*



**Be'triebs·sys·tem** <n.; -s, -e; EDV> **Programmbündel**, das die Bedienung eines Computers ermöglicht. [12]



## Lehrbücher I

Der Zweck eines Betriebssystems [besteht] in der *Verteilung von Betriebsmitteln* auf sich bewerbende Benutzer. [4]



Eine Menge von Programmen, die die Ausführung von Benutzerprogrammen und die *Benutzung von Betriebsmitteln steuern*. [3]



## Lehrbücher II

Eine *Softwareschicht*, die alle Teile des Systems verwaltet und dem Benutzer eine Schnittstelle oder eine *virtuelle Maschine* anbietet, die einfacher zu verstehen und zu programmieren ist [als die nackte Hardware]. [10]



Ein Programm, das als *Vermittler* zwischen Rechnernutzer und Rechnerhardware fungiert. Der Sinn des Betriebssystems ist eine Umgebung bereitzustellen, in der Benutzer bequem und effizient Programme ausführen können. [9]



The operating system is itself a program which has the functions of *shielding the bare machine* from access by users (thus protecting the system), and also of *insulating the programmer* from the many extremely intricate and messy problems of reading the program, calling a translator, running the translated program, directing the output to the proper channels at the proper time, and passing control to the next user. [5]



Ein Betriebssystem kennt auf jeden Fall keinen Prozessor mehr, sondern ist neutral gegen ihn, und das war es vorher noch nie. Und auf diese Weise kann man eben *jeden beliebigen Prozessor auf jedem beliebigen anderen emulieren*, wie das schöne Wort lautet. [6]



## Sachbücher und Normen

Es ist das Betriebssystem, das die Kontrolle über das Plastik und Metall (Hardware) übernimmt und anderen Softwareprogrammen (Excel, Word, ...) eine *standardisierte Arbeitsplattform* (Windows, Unix, OS/2) schafft. [2]



Die Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften der Rechanlage die Grundlage der möglichen Betriebsarten des digitalen Rechensystems bilden und insbesondere die *Abwicklung von Programmen steuern* und überwachen. [1]



# Gliederung

---

Einordnung

Fallstudie

Speicherbelegung

Analyse

Begriffsdeutung

Literaturauszüge

Zusammenfassung



# Resümee

---

**Be'triebs·sys·tem** <n.; -s, -e; EDV> (*operating system*)

- eine Menge von Programmen, die
  - Programme, Anwendungen oder BenutzerInnen assistieren sollen
  - die Ausführung von Programmen überwachen und steuern
  - den Rechner für eine Anwendungsklasse betreiben
  - eine abstrakte Maschine implementieren
- verwaltet die Betriebsmittel eines Rechensystems
  - kontrolliert die Vergabe der (Software/Hardware) Ressourcen
  - verteilt diese ggf. gerecht an die mitbenutzenden Rechenprozesse
- definiert sich nicht über die Architektur, sondern über Funktionen



### ■ wir werden...

- SP1
1. einen „Hauch“ von Rechnerorganisation „einatmen“
  2. Betriebssysteme in ihrer Grobfunktion „von aussen“ betrachten
  3. Rechnerbetriebsarten kennen- und unterscheiden lernen
- SP2
4. eine kurze Zwischenbilanz von SP1 ziehen
  5. Funktionen von Betriebssystemen im Detail untersuchen
  6. den Stoff rekapitulieren

### ■ Zusammenhänge stehen im Vordergrund!

- Leitfaden ist die ganzheitliche Betrachtung von Systemfunktionen
- skizziert wird eine logische Struktur ggf. vieler Ausprägungsformen
- klassischer Lehrbuchstoff wird ergänzt, weniger repetiert oder vertieft



## Nachwort

### ■ Typen von Betriebssystemen dürfen nicht dogmatisiert werden

- etwa: ☆❄️■◆| „ist besser als“ ❄️❄️❄️❄️❄️ — umgekehrt dito
- oder: ☆❄️❄️❄️❄️ „schlägt beide um Längen“...

### ■ Betriebssysteme sind immer im **Anwendungskontext** zu beurteilen

- „Universalbetriebssysteme“ gibt es nicht wirklich, wird es nie geben
- allen Anwendungsfällen wird **nie gleich gut** Genüge getragen



„Universalbetriebssystem“



„Spezialbetriebssystem“



# Literaturverzeichnis I

- [1] DEUTSCHES INSTITUT FÜR NORMUNG:  
*Informationsverarbeitung — Begriffe.*  
Berlin, Köln, 1985 (DIN 44300)
- [2] EWERT, B. ; CHRISTOFFER, K. ; CHRISTOFFER, U. ; ÜNLÜ, S. :  
*FreeHand 10.*  
Galileo Design, 2001. –  
ISBN 3-898-42177-5
- [3] HABERMANN, A. N.:  
*Introduction to Operating System Design.*  
Science Research Associates, 1976. –  
ISBN 0-574-21075-X
- [4] HANSEN, P. B.:  
*Betriebssysteme.*  
Carl Hanser Verlag, 1977. –  
ISBN 3-446-12105-6



# Literaturverzeichnis II

- [5] HOFSTADTER, D. R.:  
*Gödel, Escher, Bach: An Eternal Golden Braid — A Metaphorical Fugue on Minds and Machines in the Spirit of Lewis Carrol.*  
Penguin Books, 1979. –  
ISBN 0-140-05579-7
- [6] KITTLER, F. :  
*Interview.*  
<http://www.hydra.umn.edu/kittler/interview.html>, 1993
- [7] POLLIO, V. M. V.:  
*De Architectura Libris Decem.*  
Primus Verlag, 1996 (Original 27 v. Chr.)
- [8] SCHNEIDER, H.-J. :  
*Lexikon der Informatik und Datenverarbeitung.*  
München, Wien : Oldenbourg-Verlag, 1997. –  
ISBN 3-486-22875-7
- [9] SILBERSCHATZ, A. ; GALVIN, P. B. ; GAGNE, G. :  
*Operating System Concepts.*  
John Wiley & Sons, Inc., 2001. –  
ISBN 0-471-41743-2



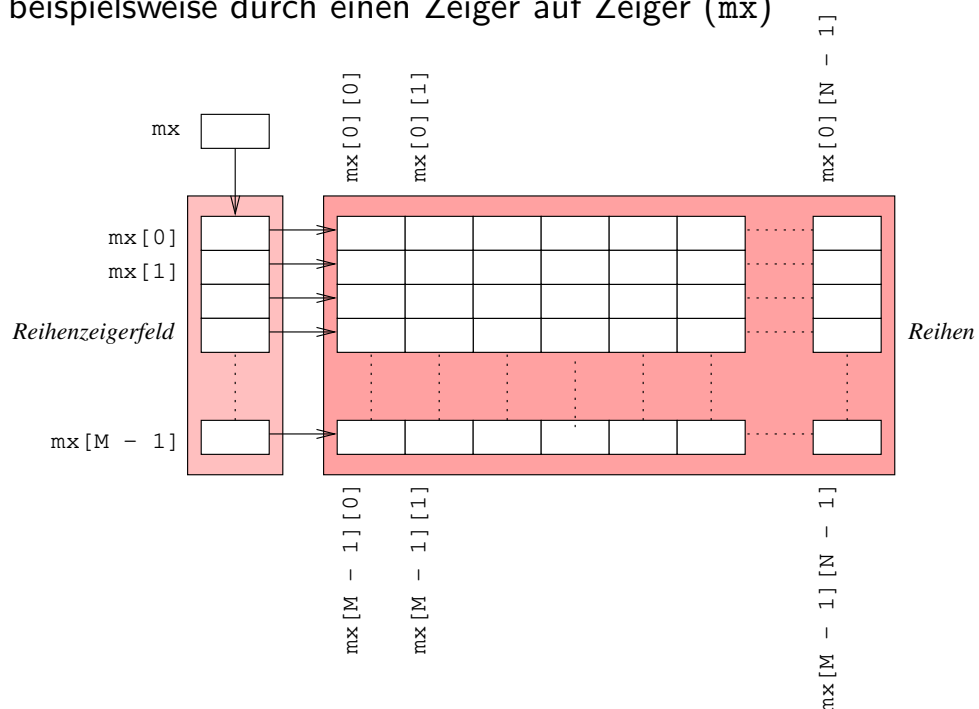


- [10] TANENBAUM, A. S.:  
*Operating Systems: Design and Implementation.*  
Prentice-Hall, Inc., 1997. –  
ISBN 0-136-38677-6
- [11] TENNENHOUSE, D. :  
*Proactive Computing.*  
In: *Communications of the ACM* 43 (2000), Mai, Nr. 5, S. 43–50
- [12] WAHRIG-BURFEIND, R. :  
*Universalwörterbuch Rechtschreibung.*  
Deutscher Taschenbuch Verlag, 2002. –  
ISBN 3-423-32524-0



## Matrix als zweidimensionales offenes Feld

- **offene** (dynamische) **Felder** als Datentypen kennt C nicht, sie sind bei Bedarf durch das **Zeigerkonzept** zu implementieren
  - hier beispielsweise durch einen Zeiger auf Zeiger (mx)



# Instanzenbildung einer $N \times N$ Matrix

```
1  #include <stdlib.h>
2  #include <stdbool.h>
3
4  int main (int argc, char *argv[]) {
5      if (argc == 3) {
6          unsigned int n = atol(argv[2]);
7          if (n != 0) {
8              /* allocate row pointer field */
9              int **mx = (int**)calloc(n, sizeof(int*));
10             if (mx != 0) {
11                 /* succeeded, setup 2nd dimension */
12                 bool goon = true;
13                 for (int i = 0; i < n; i++) { /* allocate integer rows */
14                     mx[i] = (int*)calloc(n, sizeof(int));
15                     if (mx[i] == 0) { /* rows incomplete, skip */
16                         goon = false;
17                         break;
18                     }
19                 }
20
21                 if (goon) { /* all complete, initialize...*/
22                     if (*argv[1] == 'R') by_row(mx, n, 42);
23                     else by_column(mx, n, 42);
24                 }
25
26                 for (int i = 0; i < n; i++) { /* deallocate integer rows */
27                     if (mx[i] == 0) break;
28                     free(mx[i]);
29                 }
30                 free(mx); /* deallocate row pointer field */
31             }
32         }
33     }
34 }
```

## Initialisierung der Matrix

$*mx[] \equiv **mx$

### ■ zeilenweises Vorgehen: Spaltenelemente j von Zeile i aufzählen

```
1 void by_row (int *mx[], unsigned int n, int v) {
2     unsigned int i, j;
3     for (i = 0; i < n; i++)
4         for (j = 0; j < n; j++)
5             mx[i][j] = v;
6 }
```

### ■ spaltenweises Vorgehen: Zeilenelemente i von Spalte j aufzählen

```
7 void by_column (int *mx[], unsigned int n, int v) {
8     unsigned int i, j;
9     for (j = 0; j < n; j++)
10         for (i = 0; i < n; i++)
11             mx[i][j] = v;
12 }
```

### Gemeinsamkeit und Unterschied (vgl. S. 9)

**funktional** nutzen beide ein zweidimensionales dynamisches Feld  
**nichtfunktional** unterscheiden sie sich im Laufzeitverhalten

- kritischer Lastpunkt (*hotspot*) ist die Zuweisung der inneren Schleife<sup>6</sup>

- dynamisch angelegtes Feld festen Ausmaßes (S. 9)

– by\_row

```
1 LBB0_3:
2     movl %ecx, (%edx,%ebx,4)
3     incl %ebx
4     cmpl %ebx, %eax
5     jne  LBB0_3
```

– by\_column

```
6 LBB1_3:
7     movl %ecx, (%ebx)
8     addl %esi, %ebx
9     decl %eax
10    jne  LBB1_3
```

- dynamisch angelegtes Feld offenen Ausmaßes (S. 36)

– by\_row

```
11 LBB0_3:
12     movl %ecx, (%edi,%ebx,4)
13     incl %ebx
14     cmpl %ebx, %eax
15     jne  LBB0_3
```

– by\_column

```
16 LBB1_3:
17     movl (%edx,%edi,4), %ebx
18     movl %ecx, (%ebx,%esi,4)
19     incl %edi
20     cmpl %edi, %eax
21     jne  LBB1_3
```

- zeilenweises Vorgehen ist in beiden Varianten im Zeitverhalten gleich, aber der offene Fall benötigt mehr Speicher (Reihenzeigerfeld)
- spaltenweises Vorgehen benötigt bei der offenen Variante mehr Zeit, da komplexere und mehr Befehle benötigt werden

<sup>6</sup>gcc -O -m32 -static -S

