

## Teil II

### Einführung in den Lehrstoff

# Überblick

## Einleitung

Motivation

Begriffsdeutung

Weiterer Vorlesungsverlauf

Zusammenfassung

# Betriebssysteme...

...bilden das **Rückgrat** jedes Rechensystems, ob groß oder klein

- ▶ was Betriebssysteme sind und was nicht, ruft „Glaubenskriege“ hervor
- ▶ entscheidend ist, dass Betriebssysteme nie dem Selbstzweck dienen

...sind unerlässliches **Handwerkszeug** der Informatik

- ▶ das zu beherrschen ist und gelegentlich auch angefertigt werden muss

...verstehen hilft, **Phänomene** zu begreifen und besser einzuschätzen

- ▶ d.h., Eigenschaften (*features*) von Fehlern (*bugs*) zu unterscheiden
  - ▶ um Fehler kann ggf. „herum programmiert“ werden
  - ▶ um zum Anwendungsfall unpassende Eigenschaften oft jedoch nicht
- ▶ das Systemverhalten kann sich positiv/negativ „nach oben“ auswirken

# Phänomen Speicherverwaltung

## Vorbelegung einer Matrix

### A, j wächst schneller als i

```
void setA (int *mx, int n, int v) {  
    int i, j;  
    for (i = 0; i < n; i++)  
        for (j = 0; j < n; j++)  
            mx[i * n + j] = v;  
}
```

### B, i wächst schneller als j

```
void setB (int *mx, int n, int v) {  
    int i, j;  
    for (j = 0; j < n; j++)  
        for (i = 0; i < n; i++)  
            mx[i * n + j] = v;  
}
```

`mx`                      Zeiger auf Matrix `m[n][n]` mit `n` Zeilen und Spalten  
`mx[i * n + j]`        entspricht `m[i][j]` (*store by row*)

Wirkt sich der Unterschied auf das Laufzeitverhalten aus?

# Phänomen Speicherverwaltung (Forts.)

## Instanzenbildung und Initialisierung der Matrix

```
#include <stdlib.h>

main (int argc, char *argv[]) {
    int *mx, n;
    void (*preset)();

    if (argc == 3) {
        preset = strcmp(argv[1], "setB") ? setA : setB;
        if ((n = atol(argv[2]))
            if ((mx = (int*)calloc(n * n, sizeof(int)))) {
                preset(mx, n, 42);
                free(mx);
            }
        }
    }
}
```

# Phänomen Speicherverwaltung (Forts.)

Laufzeitverhalten:  $10\,000 \times 10\,000$  Matrix (400 MB)

	setA()	setB()
Mac OS X 1.25 GHz PPC G4 512 MB RAM	0.450u 1.150s 1.842r	17.413u 2.037s 20.097r
Linux, 2 × 3 GHz P4 XEON, 4 GB RAM	0.415u 0.734s 1.150r	12.248u 0.733s 12.990r
SunOS, 2 × 1 GHz UltraSPARC IIIi 8 GB RAM	1.645u 0.890s 2.870r	45.495u 0.885s 47.725r
Windows, 2 × 3 GHz P4 XEON, 4 GB RAM	0.843u 0.250s 1.125r	9.937u 0.250s 10.344r

*user, system und real time in Sek.*

## Anlaufphasen...

### Mac OS X und setB()

21.450u	5.250s	2:01.459r
17.570u	2.050s	0:20.277r
17.360u	2.150s	0:19.932r
... iCal, dann wieder setB()...		
19.370u	2.750s	0:42.090r
18.750u	2.530s	0:35.964r
17.540u	2.040s	0:19.908r

Weniger Mac OS X, vielmehr (wohl) der zu kleine Arbeitsspeicher...

Manche Dinge erweisen sich im Nachhinein noch viel komplizierter, als es auf den ersten Blick schien. [5]

# Phänomen Speicherverwaltung (Forts.)

Hardware ist das geringere „Übel“

**Software** ist für die „Ausreißer“ verantwortlich:

der **Kompilierer** bildet die Matrix ab auf einen *logischen Adressraum*

- ▶ Arbeitsspeicher (z.B. RAM) ist eindimensional organisiert
- ▶ Linearisierung bringt Daten in die notwendige „flache“ Struktur
  - ▶ im vorliegenden Fall ein 2-dimensionales Feld ( $m[n][n]$ )

das **Betriebssystem** implementiert zwei weitere Abbildungen:

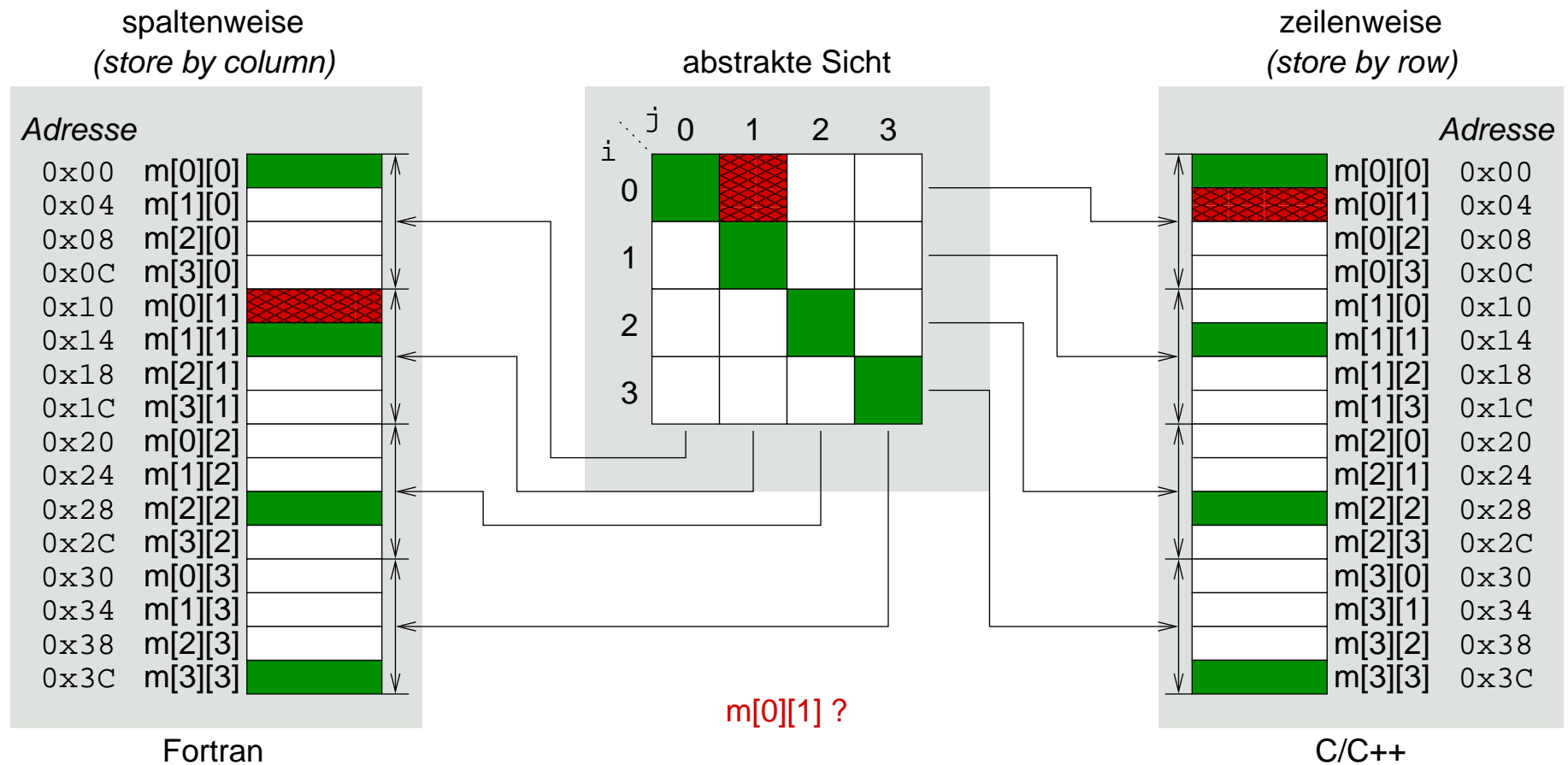
1. vom logischen Adressraum auf einen *virtuellen Adressraum*
2. vom virtuellen Adressraum auf den *physikalischen Adressraum*

Sir Isaac Newton

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

# Abbildung durch den Kompilierer

## Eindimensionale Speicherung eines zweidimensionalen Feldes





# Abbildung durch den Kompilierer (Forts.)

Referenzfolgen auf den Speicher: C/C++ (*store by row*)

A, d.h. j wächst schneller als i

0x00, 0x04, 0x08, 0x0C,  
0x10, 0x14, 0x18, 0x1C,  
0x20, 0x24, 0x28, 0x2C,  
0x30, 0x34, 0x38, 0x3C

B, d.h. i wächst schneller als j

0x00, 0x10, 0x20, 0x30,  
0x04, 0x14, 0x24, 0x34,  
0x08, 0x18, 0x28, 0x38,  
0x0C, 0x1C, 0x2C, 0x3C

**Spaltenweises Vorgehen** (Fall B) „springt“ durch den Speicher

- ▶ zeigt bei zeilenweiser Speicherung keine Lokalität
- ▶ ist unkritisch bei gleichförmigem Speicherzugriff
  - ▶ *uniform memory access*, UMA
- ▶ führt im vorliegenden Fall jedoch zu ungleichförmigen Zugriffen
  - ▶ da das Betriebssystem den logischen Adressraum virtualisiert!

# Abbildung durch das Betriebssystem

## Virtueller Speicher

Programme sind trotz (evtl.) Arbeitsspeicherknappheit ausführbar...

- ▶ nicht benötigte Programmteile liegen im Hintergrundspeicher (Platte)
- ▶ sie werden erst bei Bedarf („*on demand*“) in den RAM geladen
- ▶ das Betriebssystem führt den Zugriff auf die ausgelagerten Teile aus
- ▶ die Einlagerung ist um Größenordnungen langsamer als RAM-Zugriffe

...trotz UMA-Hardware sind Arbeitsspeicherzugriffe i.A. ungleichförmig

- ▶ je sprunghafter die Zugriffe, desto wahrscheinlicher die Einlagerung
  - ▶ umso mehr Arbeit ist für das Anwendungsprogramm zu leisten
  - ▶ umso höher ist der Laufzeitanteil für den Benutzer (*user time*)
- ▶ die Programmlokalität beeinflusst das Laufzeitverhalten wesentlich

# Phänomen hin oder her...

Was macht ein Betriebssystem [aus] ?

# Betriebssystem als „lebenswichtiges Organ“

Ewert et al[6]

*Ein Computer ist, wenn er genau betrachtet wird, nur eine Ansammlung von Plastik und Metall, das zur Leitung von Strom benötigt wird. Dieser „Industriemüll“ kann somit nicht ausschließlich das sein, was wir unter einem modernen Computer verstehen, etwas, das dem Computer „Leben“ einhaucht und ihn zu dem Werkzeug unseres Jahrhunderts macht.*

*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.*

# Betriebssystem als „Programmbündel“

## Universalwörterbuch Rechtschreibung

**Be'triebs·sys·tem** *Programmbündel, das die Bedienung eines Computers ermöglicht.*

## Lexikon der Informatik

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

# Betriebssystem als „Norm“

## DIN 44300 [7]

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

## Betriebssystem aus Sicht zweier „Altmeister“

### Hansen [8]

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

### Habermann [9]

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

# Betriebssystem als „Lehrbuchklassiker“

## Silberschatz/Galvin/Gagne [3]

*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.*

## Tanenbaum [10]

*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].*



# Betriebssystem als „Gegenstand der Philosophie“

## Hofstadter [11]

*The operating system is itself a programm 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.*

# Betriebssystem als „Gegenstand der Philosophie“ (Forts.)

## Kittler [12]

*Die neue Devise von IBM in ihrer neuen, absolut rasant neuen Allianz mit Motorola, die ja wieder ein bisschen gefährdet ist, die gehen ja davon aus, und Microsoft macht es ja versuchsweise auch schon wieder nach, daß sie die Schnittstelle zum real existierenden Prozessor softwaremäßig formulieren und dann kann man irgendwas druntersetzen, was dann gerade technologisch verfügbar ist.*

*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.*

# Abriss des Vorlesungsstoffes

Wir werden. . .

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

# Grundzüge der Lehrveranstaltung

Zusammenhänge stehen im Vordergrund, Spezialitäten im Hintergrund

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

Ausprägungen von Betriebssystemen dürfen nicht dogmatisiert werden

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

Betriebssysteme sind immer im [Anwendungskontext](#) zu sehen/beurteilen

## Ein Betriebssystem (engl. *operating system*)...

... ist 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

... hat die Aufgabe, die **Betriebsmittel** des Rechners zu verwalten

- ▶ d.h., Ressourcen zu kontrollieren und ggf. gerecht zu verteilen

... definiert sich nicht über die Architektur, sondern über Funktionen