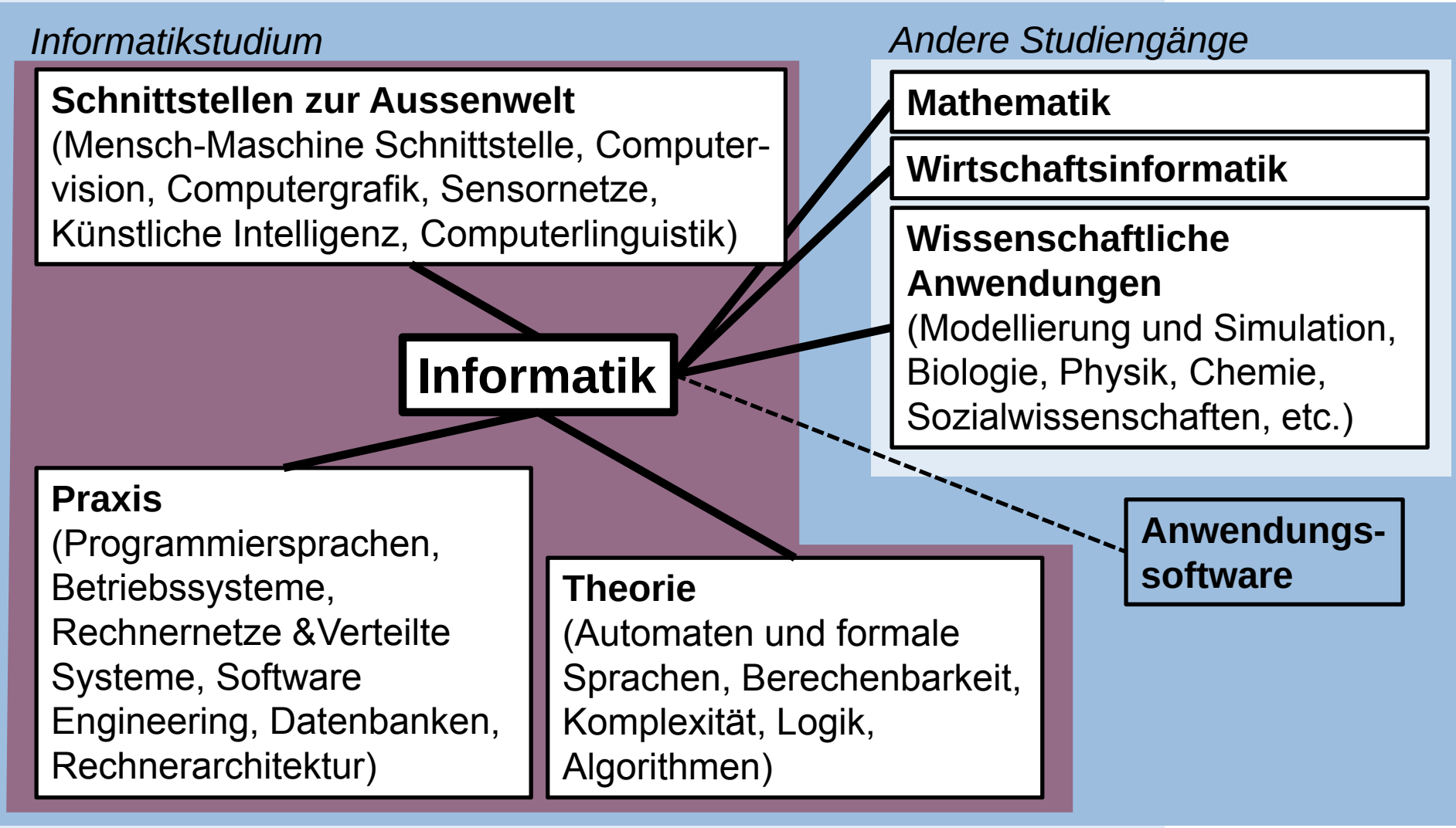
UNIVERSITÄT BERN

Einführung Informatik
Betriebssysteme

Torsten Braun
Institut für Informatik und angewandte Mathematik
Universität Bern
braun@iam.unibe.ch
www.iam.unibe.ch/~rvs

Übersicht

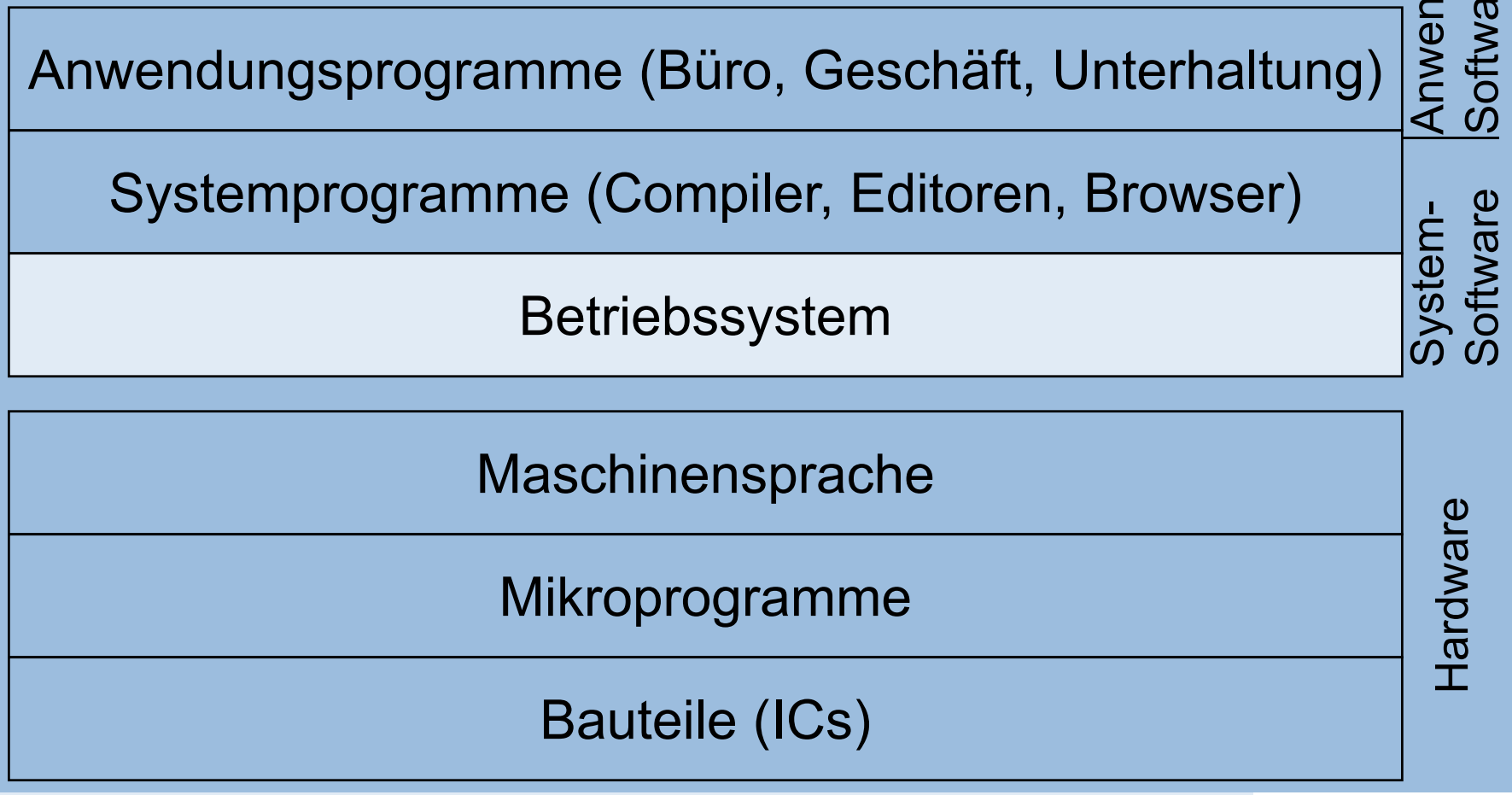

Informatikstudium
Andere Studiengänge
Schnittstellen zur Aussenwelt
(Mensch-Maschine Schnittstelle, Computer-vision, Computergrafik, Sensornetze, Künstliche Intelligenz, Computerlinguistik)
Mathematik
Wirtschaftsinformatik
Wissenschaftliche Anwendungen
(Modellierung und Simulation, Biologie, Physik, Chemie, Sozialwissenschaften, etc.)
Informatik
Praxis
(Programmiersprachen, Betriebssysteme, Rechnernetze &Verteilte Systeme, Software Engineering, Datenbanken, Rechnerarchitektur)
Theorie
(Automaten und formale Sprachen, Berechenbarkeit, Komplexität, Logik, Algorithmen)
Anwendungs-software

Inhalt

1.1 Definition Betriebssystem

- DIN 44300:
  - Die Programme eines digitalen Rechensystems,
    - die zusammen mit den Eigenschaften dieser Rechenanlage die Basis der möglichen Betriebsarten des digitalen Rechensystems bilden und
    - die insbesondere die Abwicklung von Programmen steuern und überwachen.
- Silberschatz:
  - An operating system is like a government. ... Like a government, the operating system performs no useful function by itself.
    It simply provides an environment within which other programs can do useful work.

1.2 Soft- und Hardware eines Rechners



1.3 Ziele von Betriebssystemen

- Programm als Bindeglied zwischen Computer-Benutzer und Computer-Hardware
- Ziele
  - Bequeme Bedienung eines Computers
    - Grafische Benutzerschnittstellen
  - Effizientes Ausnutzen der Computer-Hardware
    - Vermeiden von Überlastsituationen und Idle-Zeiten
    - Geringer Rechenaufwand

1.4 Sichtweisen auf ein Betriebssystem

- Anwendersicht (abstrakte Maschine / top-down)
  - Einfaches Benutzen eines Computers
  - Maximieren der Systemleistung
  - Verdecken der Hardware vor Anwendungsprogrammen
- Systemsicht (Ressourcenverwalter / bottom-up)
  - Verwaltung von Systemressourcen (CPU, Speicher, E/A-Geräte usw.)
  - Robustes Ausführen von Anwendungsprogrammen
  - Sicherheit bzgl. unerlaubtem Zugriff und Ausfällen

1.5.1 Systemkomponenten

- Prozessverwaltung
  - Erzeugen, Löschen, Ausführen, Anhalten und Weiterführen von Prozessen
  - Synchronisation von Prozessen
  - Kommunikation zwischen Prozessen
  - Behandlung von Verklemmungen
- Hauptspeicherverwaltung
  - Belegen und Freigabe von Hauptspeicher
  - Zuordnung des Hauptspeichers zu Prozessen
- Ein-/Ausgabe-System
  - Speicherverwaltung (Puffer, Caches, Spooling)
  - allgemeine Gerätetreiberschnittstelle
  - Treiber für spezielle Hardware-Geräte
- Kommunikationssystem
  - Socket-Schnittstelle
  - TCP/IP
  - Treiber für Netzwerkkarten

1.5.2 Systemkomponenten

- Sekundärspeicherverwaltung
  - Dateisystem-Verwaltung
    - Erzeugen, Öffnen, Schliessen, Lesen, Schreiben, Löschen von Dateien und Verzeichnissen
    - Abbildung von Dateien auf Sekundärspeicher
    - Sicherung auf nicht-flüchtigen Speichermedien
  - Massenspeicherverwaltung
    - Auslagern von Daten im Hauptspeicher auf Sekundärspeicher
    - Zuweisen von Speicherplatz
    - Freispeicherverwaltung
    - Disk-Scheduling
  - Caching
- Schutz- und Sicherheitssysteme
  - Zugriffskontrolle auf System- oder Benutzerressourcen
- Kommando-Interpreter und grafische Benutzerschnittstellen

1.6 Dienste

→ für Programme und Benutzer
→ zur Vereinfachung der Programmierung

- Benutzerschnittstelle
  - Command Line Interface
  - Grafische Benutzerschnittstelle
- Programmausführung
  - Laden und Beenden von Programmen
- Ein-/Ausgabe-Operationen
  - Datei- oder Gerätezugriff
  - Abwicklung über Systemaufrufe
- Dateisystem-Manipulationen
  - Erzeugen, Verändern und Löschen von Dateien und Verzeichnissen
- Kommunikation zwischen Prozessen
  - Implementierung über gemeinsamen Speicher oder Nachrichtenaustausch
- Fehlererkennung
  - Abfangen von Hardware- oder Programmfehlern

1.7 Funktionen

- → zur effizienten Ausführung des Systems, speziell bei mehreren Benutzern
- > Zuweisen von Ressourcen
  - — CPU
  - — Hauptspeicher
  - — Sekundärspeicher
  - — Ein-/Ausgabe-Geräte
- > Accounting
  - — Statistiken zur Systemoptimierung oder fairer Ressourcennutzung
- > Schutzmechanismen
  - — Vermeiden von Wechselwirkungen zwischen Prozessen
- > Sicherheitsmechanismen
  - — Authentifizierung
  - — Verschlüsselung

2. Schutzmechanismen

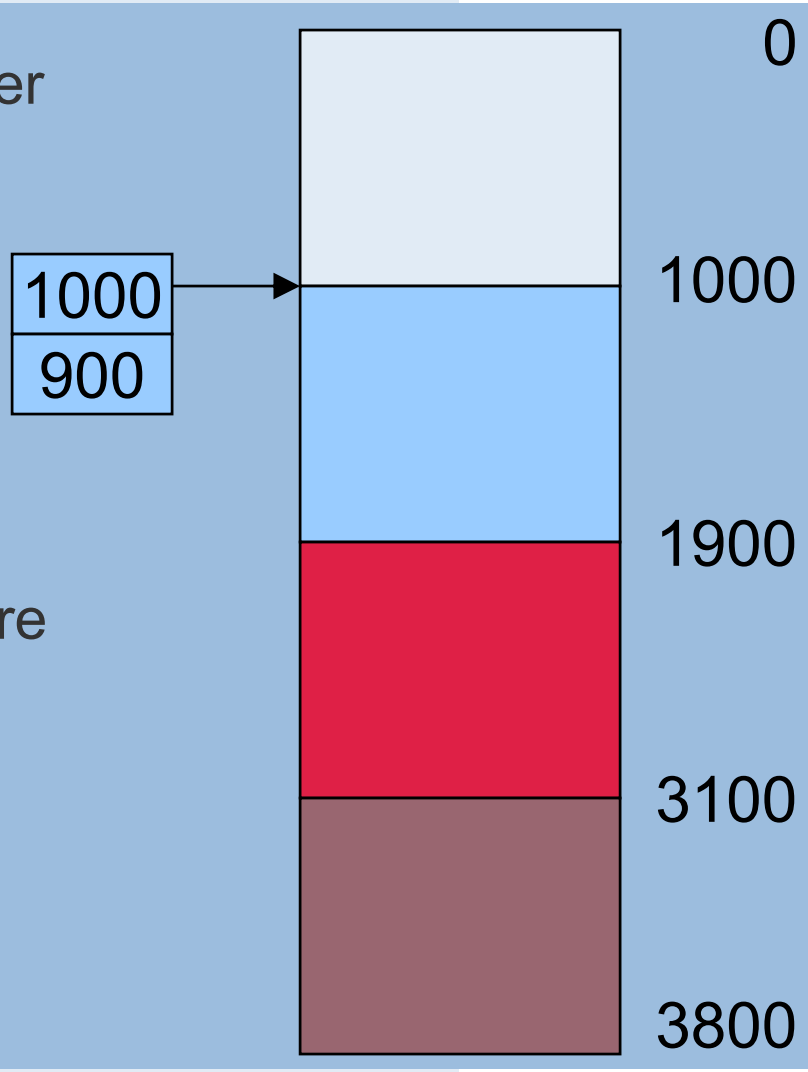
- Schwerwiegende Folgen durch Programmierfehler in Anwendungen, speziell in Mehrprogrammsystemen
- Fehlererkennung häufig durch Hardware, Fehlerbehandlung durch Betriebssystem
- Schutzmechanismen
  - CPU-Schutz
  - Speicherschutz
  - E/A-Schutz
  - Dual-Mode Operation

2.1 CPU-Schutz

- Vermeiden von Monopolisierung der CPU durch eine Anwendung
- Betriebssystem setzt Zeitgeber zur Auslösung eines Interrupts und übergibt danach die Kontrolle der CPU an die Anwendung.
- Time-Sharing
  - Unterbrechung nach bestimmtem Intervall, z.B. N ms, und Übergabe der CPU an andere Anwendung
  - Ggf. Überprüfung, ob Anwendung maximale Laufzeit erreicht hat.

2.2 Speicherschutz

- Schutz der Prozesse untereinander vor inkorrektem Speicherzugriff
- Beschreibung des zulässigen Speicherbereichs durch Basis- (kleinste Adresse) und Limit-Register (Grösse des Speicherbereichs)
- Vergleich von Speicheradressen mit Registern durch CPU-Hardware
- Laden der Register durch Betriebssystem über privilegierte Instruktionen

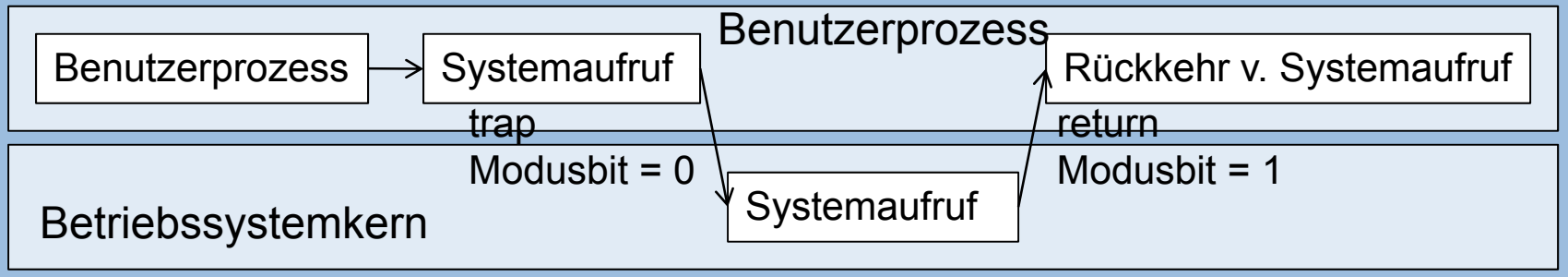


2.3 E/A-Schutz

- E/A-Instruktionen sind privilegiert und können nur vom Betriebssystem aufgerufen werden.
- Ausführen der E/A-Operationen durch Anwendungen über Systemaufrufe

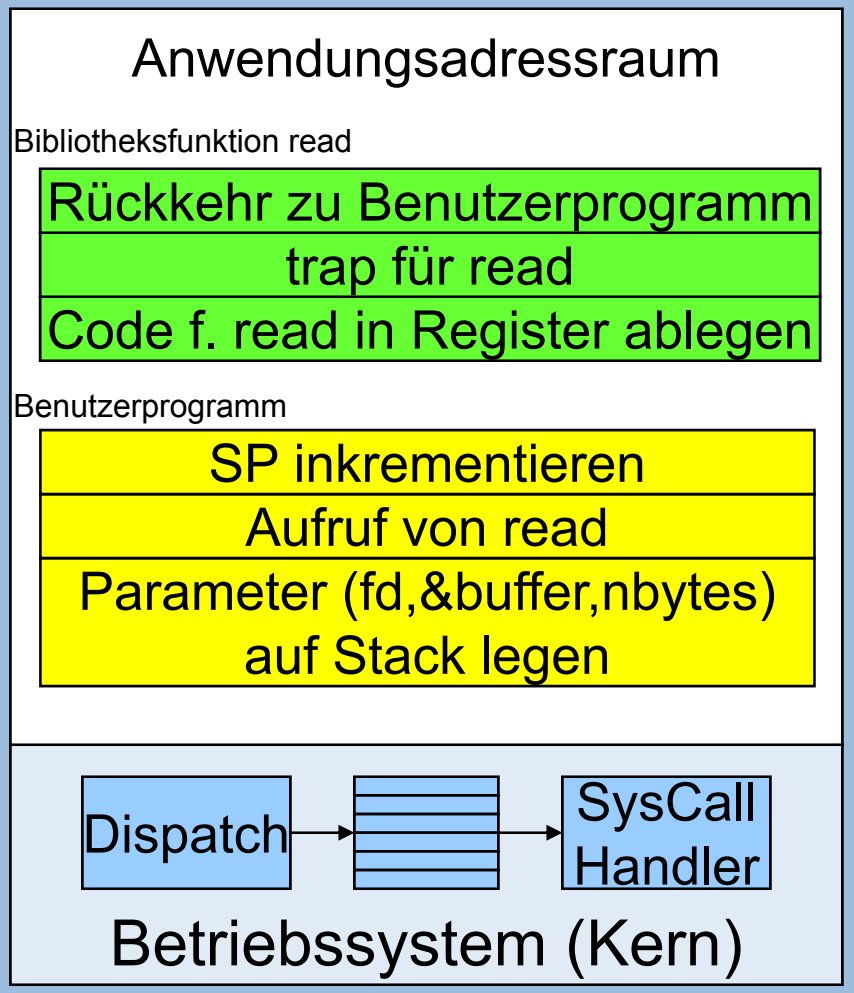
2.4 Dual-Mode Operation

- Dual-Mode Operation
  - Benutzermodus für Anwendungen
  - Systemmodus für Betriebssystem
    - Ausführung sensitiver Instruktionen nur im Systemmodus
    - Ausführung privilegierter Instruktionen verursachen Systemaufruf (Trap)
- Wechsel zwischen Benutzer- und Systemmodus
  - Benutzermodus → Systemmodus
    - durch Hardware nach Systemaufruf (Trap) oder Interrupt
  - Systemmodus → Benutzermodus
    - durch Rücksetzen des Modusbits vor Rückkehr in Benutzerprozess



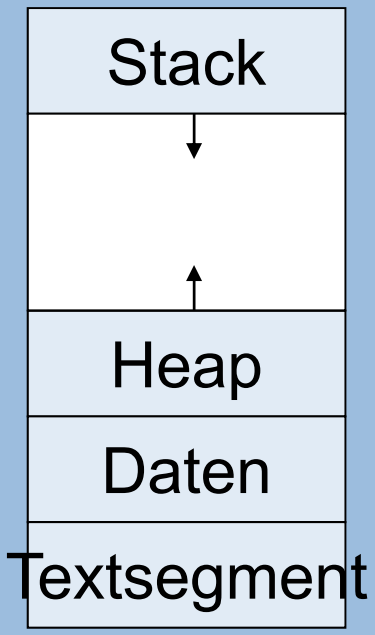
3. Systemaufrufe

- Schnittstelle zwischen Anwendungsprogrammen und Betriebssystem
- vom Betriebssystem bereitgestellte Aufrufe von Systemfunktionen
- Maschinen-Instruktionen (**`trap`**) zum Wechsel vom Benutzer- in Systemmodus
- Parameterübergabe an Betriebssystem
  - Register
  - Speicherbereich
  - Stack
- Beispiel: **`count = read (fd, buffer, nbytes)`**



4.1 Definition Prozess

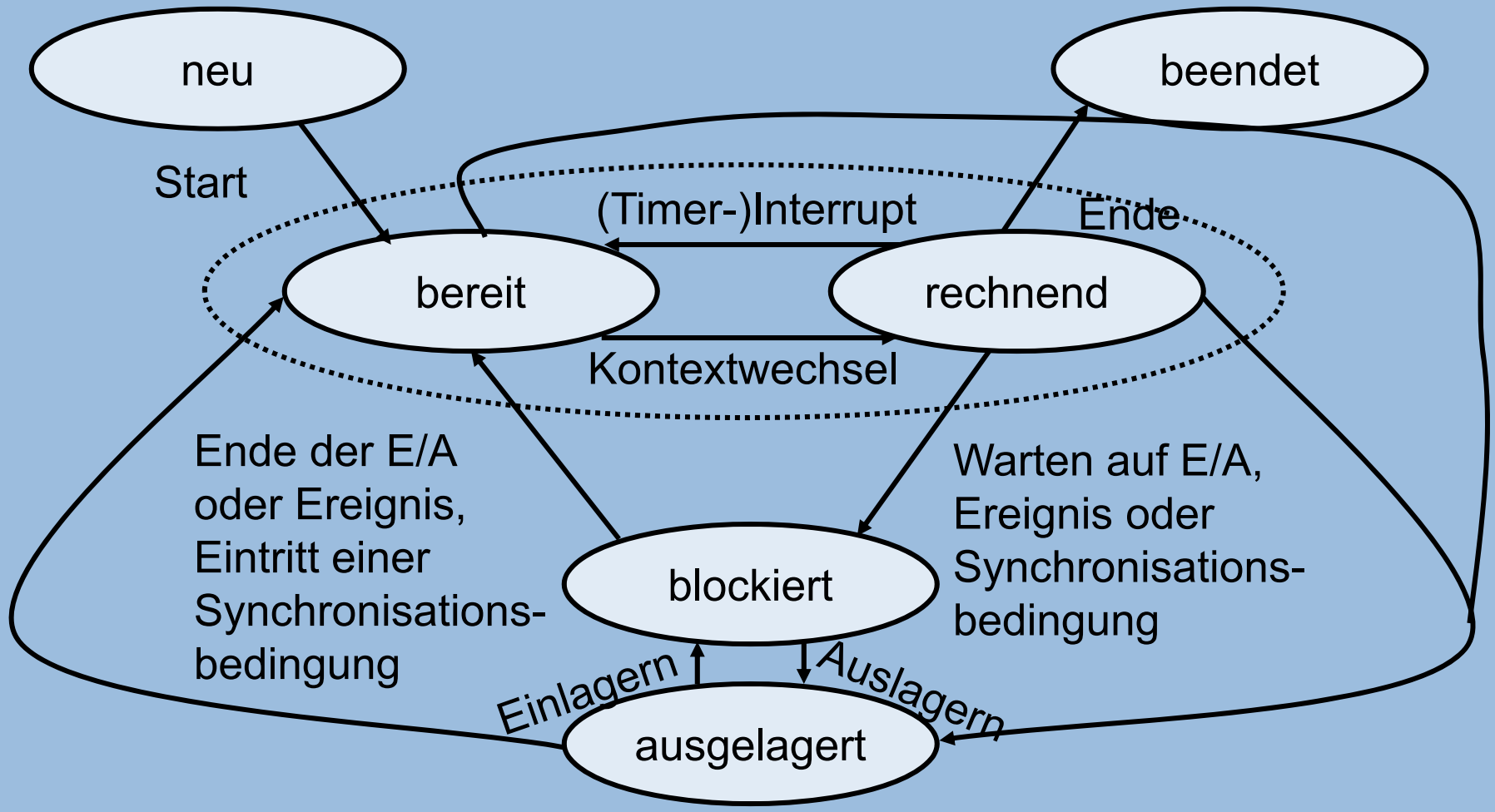
- informell: Prozess = Programm in Ausführung
- besteht aus
  - Programmcode
  - Aktivität: Befehlszähler + Prozessorregisterinhalt
  - Stack (Laufzeitkeller) mit temporären Daten
  - globale Daten
  - Heap für dynamisch zugeteilten Speicher
- Jeder Prozess besitzt eine virtuelle CPU.
- Prozesse stellen selbst Betriebsmittel dar.



4.2 Prozesszustände

- neu
  - Prozess wurde erzeugt.
- rechnend
  - Instruktionen des Prozesses werden auf der CPU ausgeführt.
- blockiert
  - Prozess wartet auf ein Ereignis (z.B. Ende einer E/A-Operation, Signal, Eintritt einer Synchronisationsbedingung).
- ausgelagert
  - Prozess ist wegen Hauptspeichermangels auf Hintergrundspeicher ausgelagert.
- bereit
  - Prozess wartet auf CPU-Zuteilung.
- beendet
  - Prozess hat die Ausführung abgeschlossen.

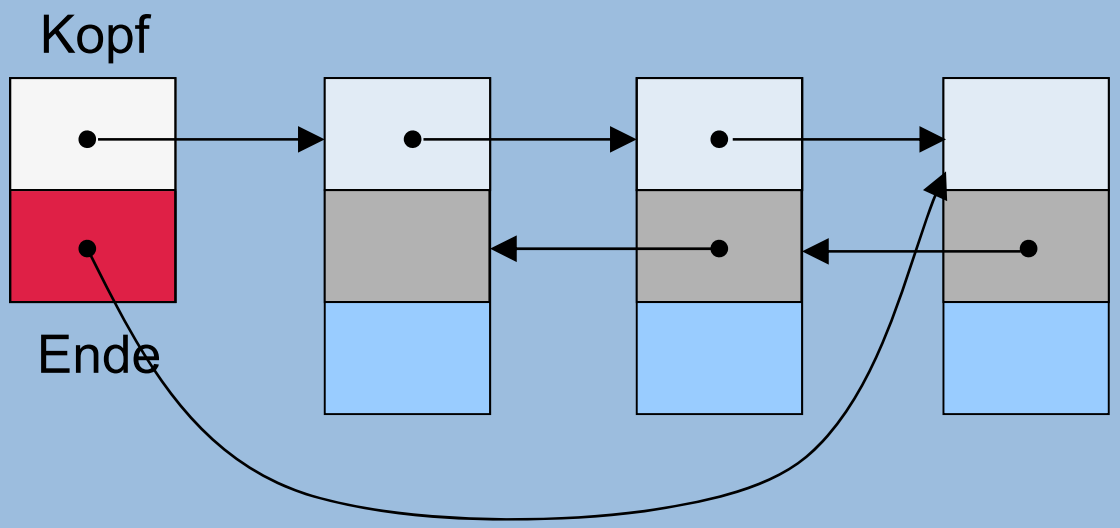
4.3 Prozesszustandsdiagramm

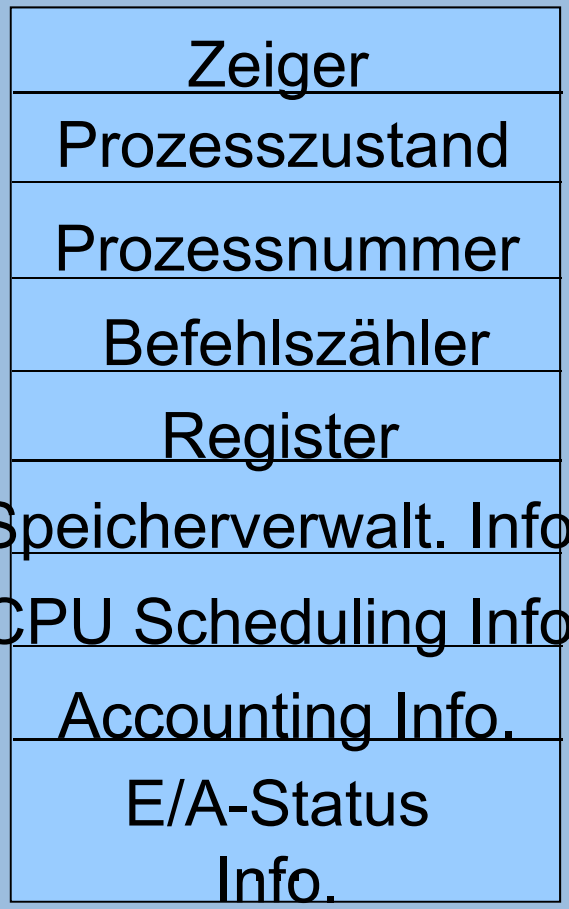


k rechnende Prozesse zu einem Zeitpunkt (k: Anzahl der Prozessoren)

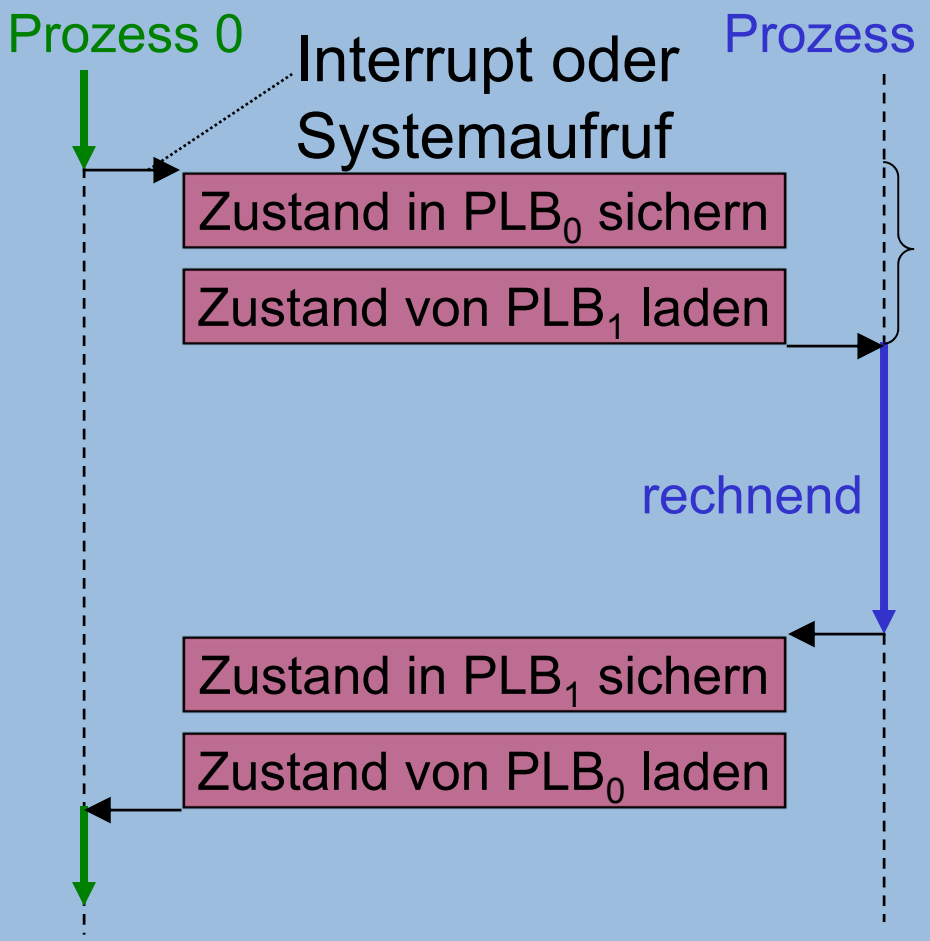
4.4 Prozessleitblock

- Beschreibung des Prozesszustands, z.B. auch der zur Ausführung benötigten Betriebsmittel
- Prozessleitblock (PLB) repräsentiert einen Prozess im Betriebssystem.
- Verkettung von Prozessleitblöcken in den Warteschlangen





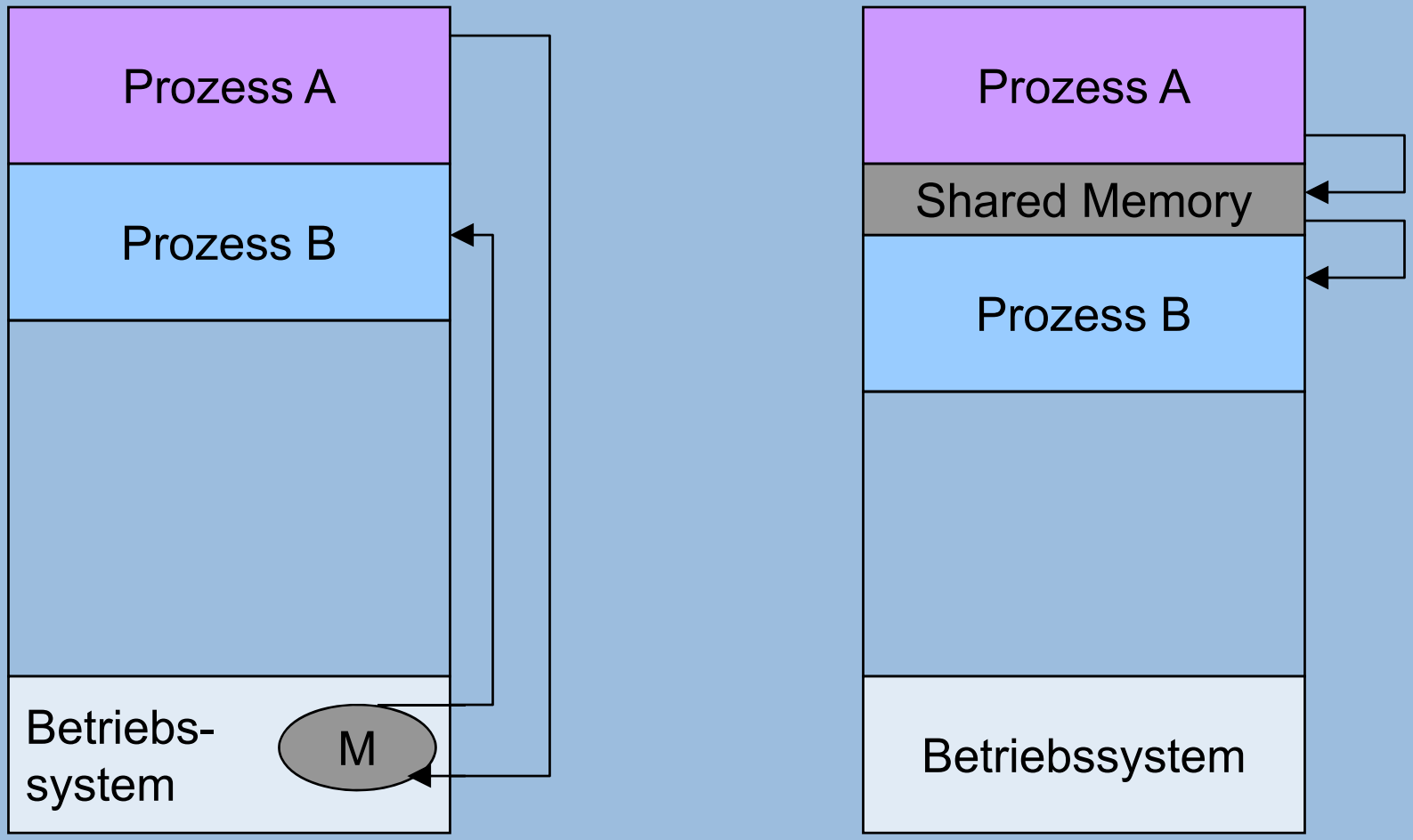
4.5 Prozessumschaltung



> Kontextwechsel durch Dispatcher
  - Sicherung des alten Prozesszustands in Prozessleitblock (PLB)
  - Laden des neuen Prozesszustands aus PLB

> relativ hoher Overhead (1-1000 µs)

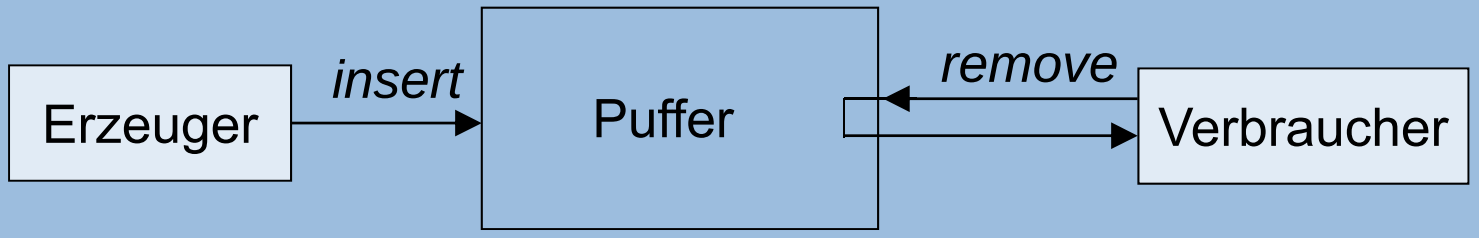
5.1 Prozessinteraktion: Message Passing und Shared Memory



M: Mailbox: Objekt, in das Nachrichten gestellt bzw. aus dem Nachrichten entnommen werden können

5.2 Speicherbasierte Prozessinteraktion

- Parallele Prozesse mit Zugriff auf gemeinsame Daten, wodurch Inkonsistenzen verursacht werden können.
- Mechanismen zur Synchronisation von parallelen Prozessen erforderlich
- Beispiel: Erzeuger/Verbraucher-Problem
  - — Puffer kann N Informationselemente speichern.
  - → Erzeuger darf nicht in vollen Puffer speichern.
  - → Verbraucher darf nicht aus leerem Puffer entfernen.



5.3 Erzeuger/Verbraucher-Implementierung


```
int in=0,out=0,counter=0;
item_type buffer[N];
```

```
item_type next_prod;
while (true){
      produce(&next_prod);
      while(counter==N)
      /*voller Puffer*/
            no_op();
      buffer[in]=next_prod;
      in=(in+1)%N;
      counter++;
}
```

```
item_type next_cons;
while (true){
      while(counter==0)
      /*leerer Puffer*/
            no_op();
      next_cons=buffer[out];
      out=(out+1)%N;
      counter--;
      consume(&next_cons);
}
```

Problem: `counter++` und `counter--` müssen atomar sein.

5.4 Race Conditions

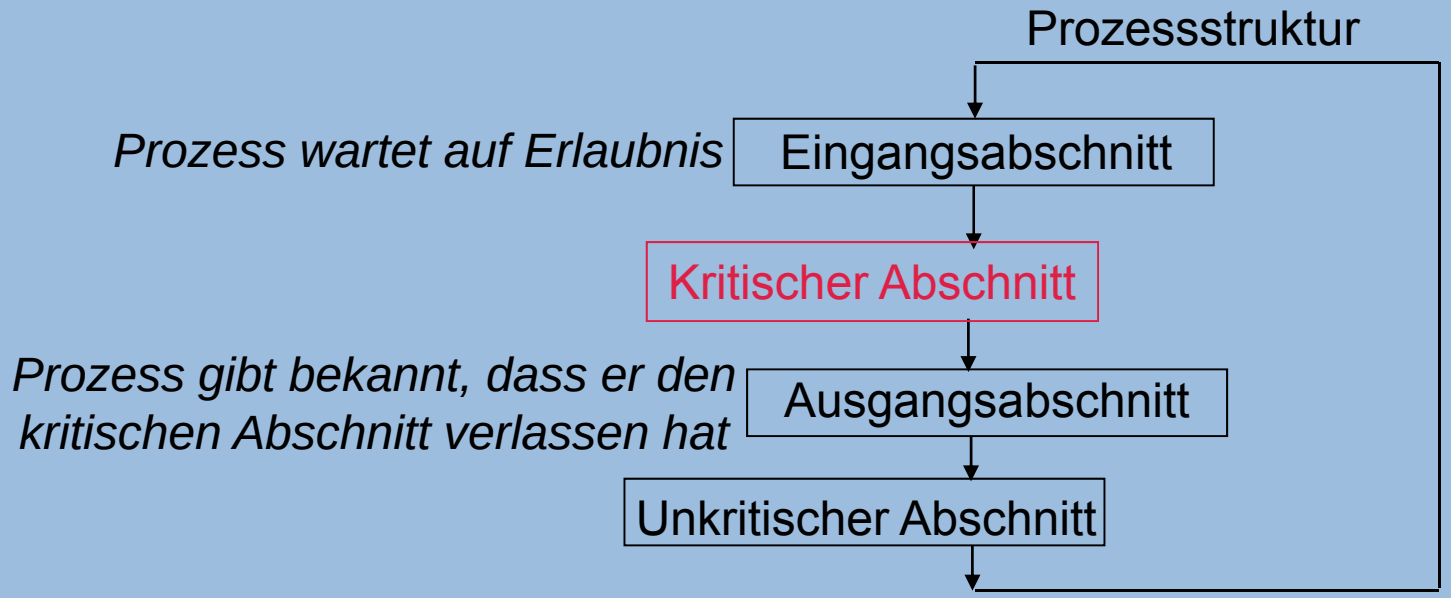

```
counter++:                    counter--:
R1=counter;                   R2=counter;
R1=R1+1;                      R2=R2-1;
counter=R1;                   counter=R2;
```

counter: 5

```
R1=counter;                   R2=counter;
R2=counter;                   R1=counter;
R1=R1+1;                      R2=R2-1;
R2=R2-1;                      R1=R1+1;
counter=R1;                   counter=R2;
counter=R2; /* =4 */          counter=R1; /* =6 */
```

5.5 Kritischer Abschnitt

- Kritischer Abschnitt: Folge von Anweisungen oder Code-Segment mit Zugriff auf gemeinsame Daten
- Höchstens ein Prozess darf sich zu einem Zeitpunkt in einem kritischen Abschnitt befinden.
- Beispiele:
  - Zugriff auf ein exklusives Betriebsmittel
  - Manipulation der Counter-Variable in Erzeuger/Verbraucher-Implementierung



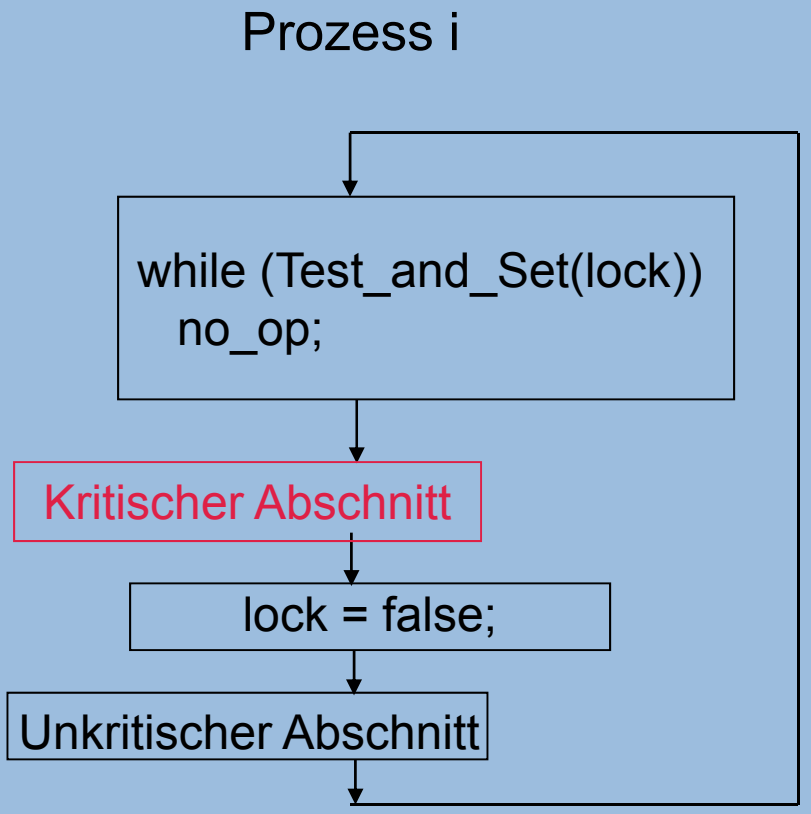
5.6 Synchronisations-Hardware: Test and Set

- gemeinsame Variable:

```
bool lock=false;
```

- atomare Funktion *Test_and_Set*

```
bool Test_and_Set
  (bool &target)
{
  bool rv = target;
  target = true;
  return rv;
}
```



5.7 Semaphore

- geschützte Variable s, auf der nur atomare Operationen **`wait(s)`** und **`signal(s)`** ausgeführt werden können.
- Werte
  - > 0: frei
  - ≤ 0: belegt
- Implementierung:
  - füge Prozess bei belegter Semaphore einer Liste wartender Prozesse (L) zu.

```
typedef struct s {int value; process_list L}
        semaphore;
```

  - Zählende Semaphor
    - negativer Wert von **`value`**: Anzeige der Anzahl wartender Prozesse
    - positiver Wert von **`value`**: Anzahl der Prozesse, die noch eintreten dürfen

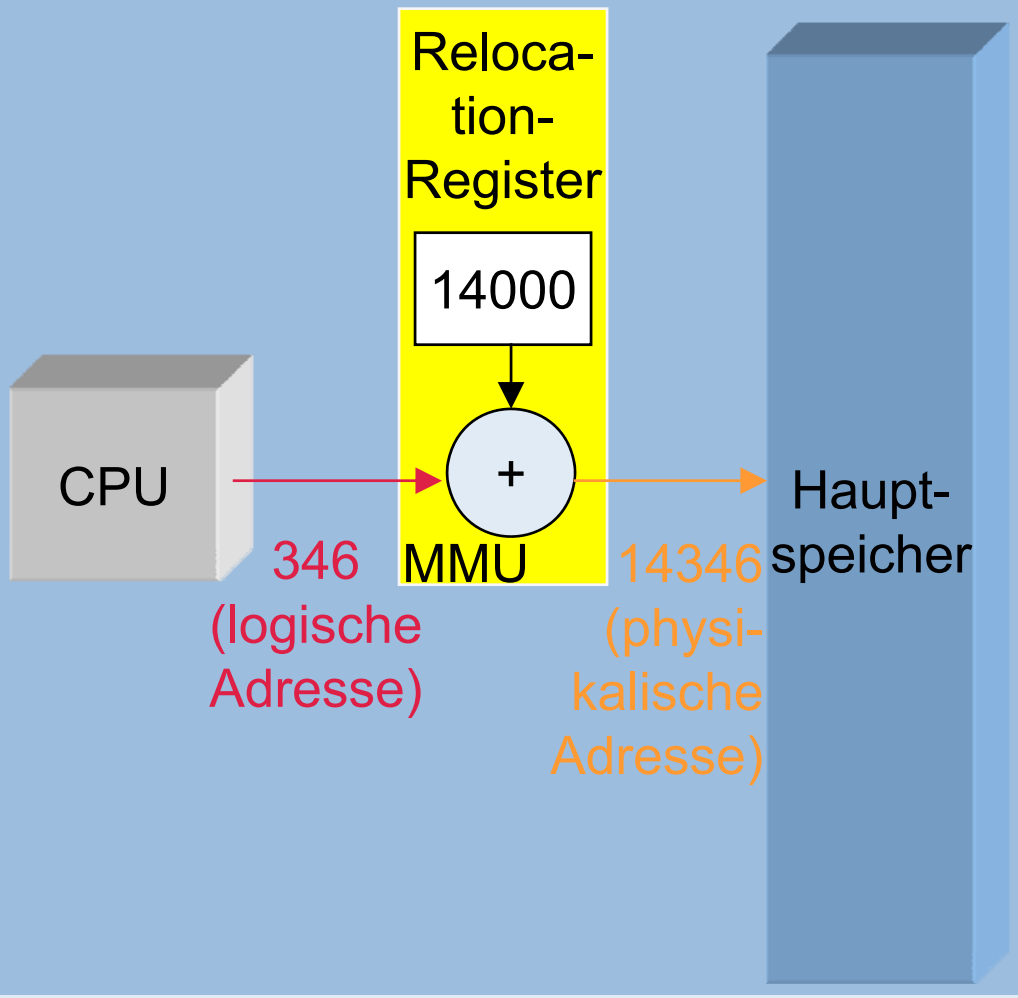
5.7.1 Implementierung von Semaphoren


```
wait(S):
S.value--;
if (S.value < 0) {
    add_this_process_to(S.L);
    block;}
    /* Blockieren des aufrufenden Prozess */

signal(S):
S.value++;
if (S.value ≤ 0){
    P=remove_a_process_from(S.L);
    wakeup(P);} /* Deblockieren von P */
```

6.1 Logische und Physikalische Adressen

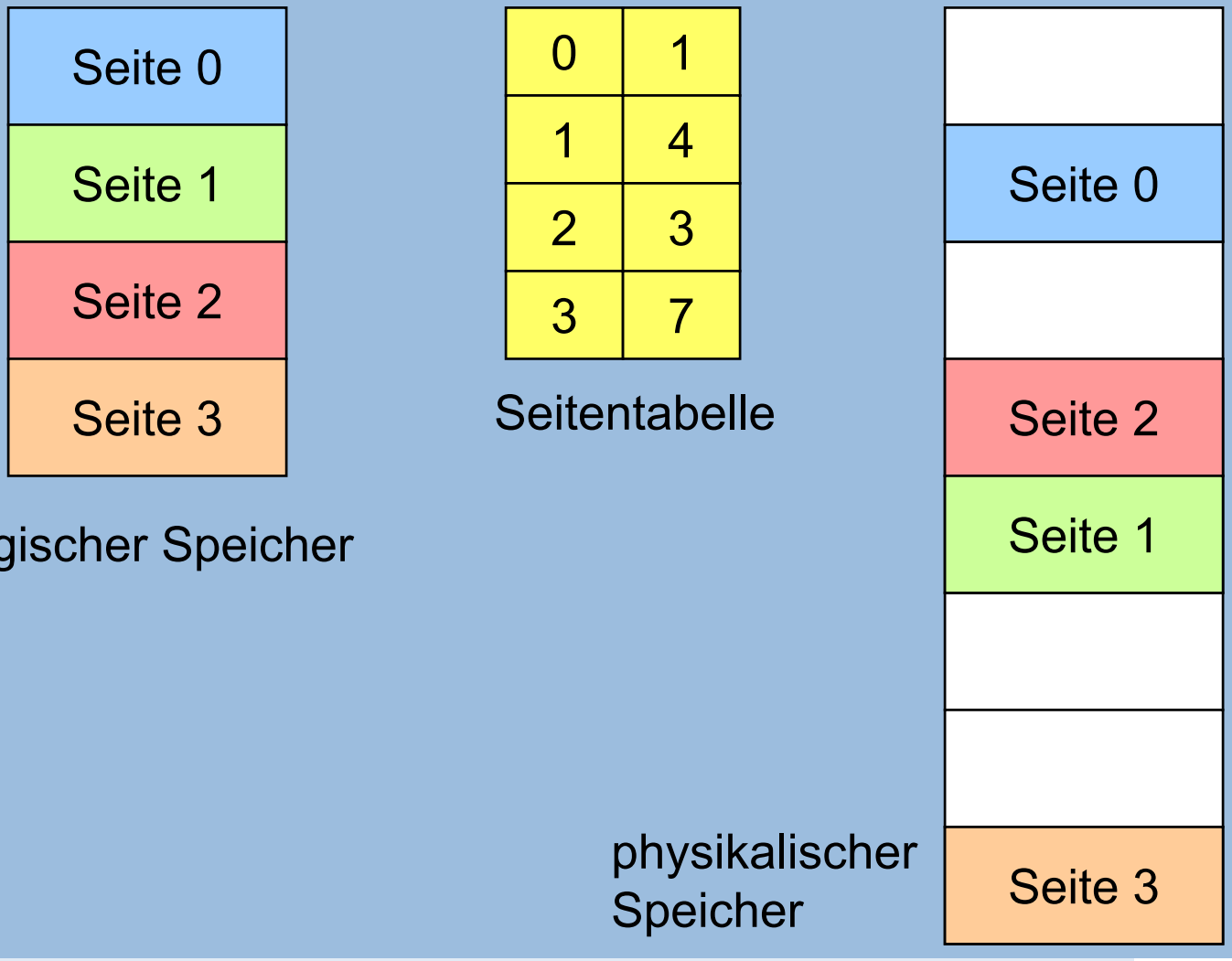
Laden des Relocation-Register bei Prozesswechsel



6.2 Paging

- nicht zusammenhängender physikalischer Adressraum
- Unterteilung
  - physikalischer Speicher in Kacheln (Frames, 2^x Bytes, z.B. 512 oder 8192 Bytes)
  - logischer Speicher in Seiten gleicher Grösse
- Programm mit Speicherbedarf von N Seiten benötigt N freie Hauptspeicherkacheln.
- Zuordnung Seiten/Kacheln über Seitentabelle
- interner Verschnitt

6.2.1 Beispiel: Paging



6.3 Demand Paging