Betriebssysteme / Operation Systems

JProf. Dr. Gunnar Stevens
Human Computer Interaction
University of Siegen
gunnar.stevens@uni-siegen.de

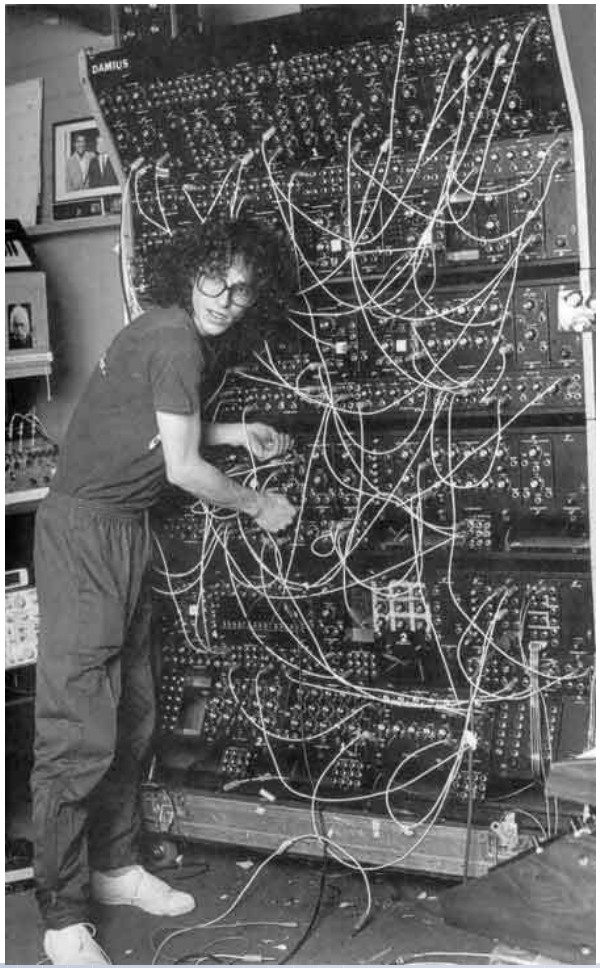
Agenda

- Zweck und Aufgaben von Betriebssystemen
- Betriebssystemvarianten
- Startvorgang
- Verwaltung von Betriebsmittel: Prozessverwaltung
  - Multithreading
  - Synchronization von Prozessen

Zweck und Aufgaben

Warum Betriebssysteme?

- Herausforderungen bei Anwendungsprogrammierung ohne Betriebsysteme
  - Unterschiedliche, austauschbare Hardware
  - Hohes technisches Detailwissen notwendig
  - Wiederholte Lösung gleicher Probleme
  - Globales Zusammenspiel von Anwendungen auf lokaler Ebene nicht lösbar



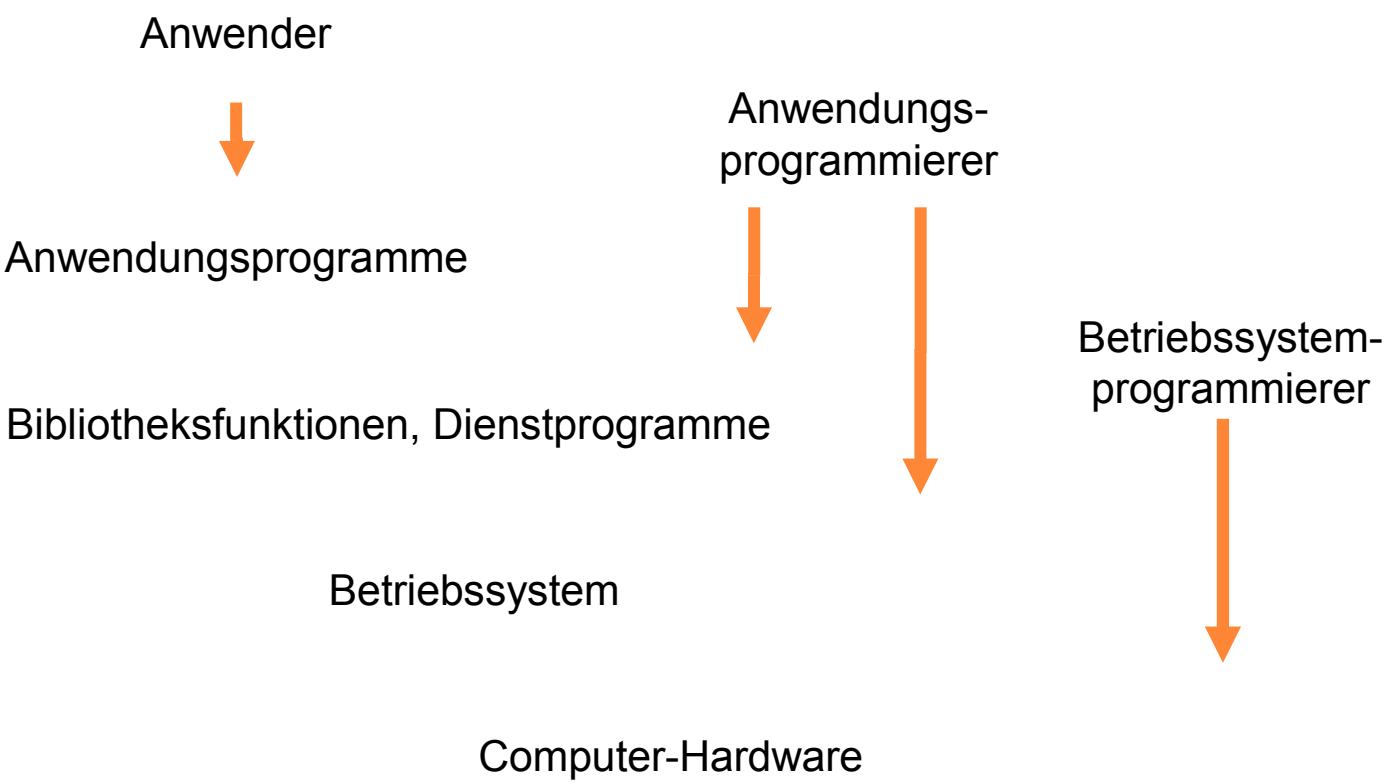
JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Aufgaben von Betriebssystemen

- Ein Betriebssystem ist ein Computerprogramm (Software) mit folgenden grundsätzlichen Funktionen:
  - Bereitstellung
    - Ausführungsumgebung für Anwendungsprogramme und -prozesse
    - Abstraktionsschicht und einheitliche Schnittstellen für Programmierer zur Nutzung der Betriebsmittel („Erweiterte Maschine“)
  - Effiziente Verwaltung, Virtualisierung und Schutz der Betriebsmittel

JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Abstraktionsniveaus



JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Erweiterte Maschine

- Abstraktion
  - von Details realer Hardware-Eigenschaften
  - von der Realisierung von Systemdiensten und Systemfunktionen
- Beispiele
  - Hardware-Abstraktion
    - Ansteuerung des Plattenarms bei der Datenspeicherung
  - Abstraktion von Systemdiensten und Systemfunktionen
    - Einfache Programmierschnittstelle zum Lesen/Schreiben einer Datei unabhängig von dem verwendeten Dateisystem (FAT, NTFS, …)

Systemaufrufe (system calls)

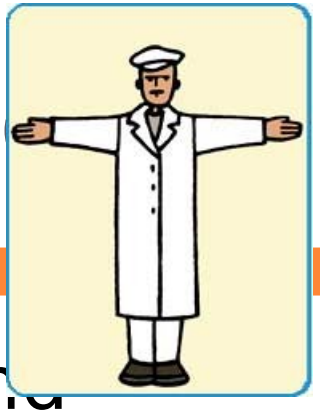
- Das Betriebssystem bietet einen Satz von Systemaufrufen zum Zugriff auf Betriebsmittel:
  - zum Zugriff auf E/A-Geräte
    - z.B. Lesen und Schreiben einer Datei
  - zur Prozesskontrolle
    - z.B. Starten und Stoppen eines Anwendungsprogramms
  - zur Speicherverwaltung
  - zur Überwachung und Kontrolle von Ressourcen

Systemaufrufe (system calls)

- POSIX (UNIX) kennt **etwa 100 Systemaufrufe** und die entsprechende Bibliotheksfunktionen und unter UNIX existiert eine (etwa) **eins zu eins Relation zwischen den Systemaufrufen und Bibliotheksfunktionen**. Unter UNIX arbeitet die graphische Benutzeroberfläche komplett in den User-Modus. Es gibt zwar viele Funktionen, die X Windows für GUI anbietet, aber das sind keine Systemaufrufe.
- Unter Windows sieht das ganz anders aus: die **Bibliotheksfunktionen und die Systemaufrufe werden getrennt voneinander betrachtet**. Die Menge von Schnittstellen-Funktionen ist in Win32 API (Application Program Interface) definiert. Die Funktionen, die GUI verwalten und Ablaufen im Kernel-Modus , gehören dazu. Die Anzahl der **Funktionen in WIN32 API** liegt bei **einigen tausend**. Einige Funktionen arbeiten aber nicht in Kernel-Modus.

JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Betriebsmittelverwaltun



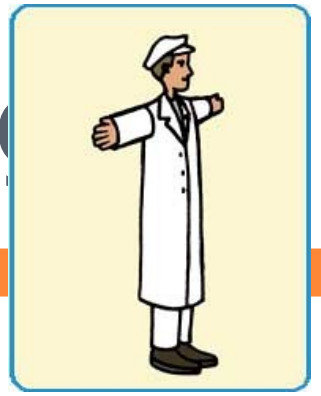
- Betriebsmittel/Systemressourcen sind
  - alles was eine Anwendung zur Ausführung braucht
- Beispiele
  - Prozessoren und Rechenzeit,
  - Speicher
    - Hauptspeicher, Festplatte, etc.
  - Netzwerkanbindung
    - Internet, LAN, WAN
  - Peripheriegeräte
    - Drucker, Graphik, Sound, etc.

JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Betriebsmittelverwaltung

- Aufgaben
  - Geordnete, kontrollierte und gerechte Zuteilung gemeinsam genutzter Betriebsmitteln an konkurrierende Prozesse/Benutzer
  - Auflösung von Konflikten und Schutz der Betriebsmittel gegeneinander
  - Protokollierung und Abrechnung der Betriebsmittelnutzung
- **Schutz**
  - Defektes Programm soll nicht das Verhalten anderer Programme oder des ganze System beeinträchtigen.
  - Schutz vor unerlaubten Zugriffen auf Dateien in Mehrbenutzersystemen
  - Fehlererkennung und -behandlung bei Hardware und Software
- **Konflikt**
  - Gleichzeitiger Zugriff z.B. auf die Festplatte zum Schreiben von Daten

Betriebsmittelverwaltung



- Verwaltung der Betriebsmittel in zwei Dimensionen:
  - Zeit: Verschiedene Benutzer erhalten Betriebsmittel nacheinander.
  - Raum: Verschiedene Benutzer erhalten verschiedene Teile einer Ressource.
- Festlegung der Betriebsmittelzuordnung
  - Zeitpunkt
  - Dauer
  - Art des Zugriffs (lesend, schreibend)
- Durchsetzung der Betriebsmittelzuordnung

JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Beispiel
Benutzer- und Rechteverwaltung

- Motivation
  - Nicht jeder Benutzer ist vertrauenswürdig
  - Nicht alle Dateien sollen für jeden Benutzer lesbar/schreibbar sein
    - Passwörter
    - vertrauliche Briefe
- Aufgabe
  - Betriebssystem regelt wer welche Dateien lesen, schreiben und ausführen darf
  - Stellt jedem Benutzer eine eigene Arbeitsumgebung zur Verfügung
  - Benutzer dürfen i.d.R. nicht auf Dateien in anderen Arbeitsumgebungen zugreifen
  - Wichtige Dateien dürfen nur von privilegierten Benutzern gelesen, geschrieben oder ausgeführt werden

Betriebssystem varianten

Mainframe-Betriebssys

- Betriebssysteme für Großrechner
- Sehr hohe Ein-/Ausgabebandbreite
- Viele Prozesse gleichzeitig mit hohem Bedarf an schneller E/A
- 3 Arten der Prozessverwaltung
  - **Batch-Verfahren:** Erledigung umfangreicher Aufgaben ohne Benutzerinteraktion
  - **Transaktionsverfahren:** Große Anzahl kleiner Aufgaben von verschiedenen Nutzern
  - **Zeitaufteilungsverfahren:** Quasi-parallele Durchführung vieler Aufgaben durch mehrere Benutzer
- Bsp.: IBM OS/390

JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Server-Betriebssysteme

- Betriebssysteme für sehr große PCs, Workstations oder auch Mainframes
- Viele Benutzer über Netzwerk bedient
- Zuteilung von Hard- und Softwareressourcen
- Bsp.: Unix, Windows 2000, Linux

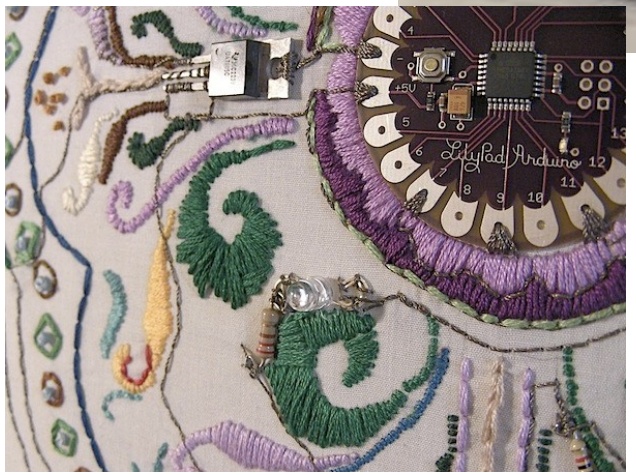
JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

PC-Betriebssysteme

- Betriebssysteme für Personalcomputer
- Meist nur 1 Benutzer (oder wenige über Netzwerk)
- Mehrere Programme pro Benutzer „quasi-parallel“
- Bsp: Windows 3.1 – Windows7, Mac OS, Linux

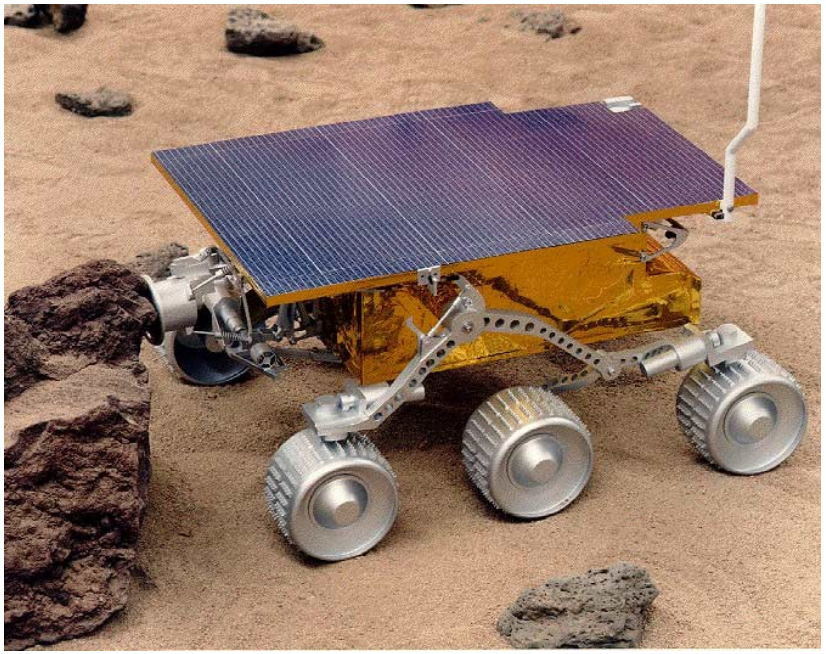
Betriebssysteme für Eingebettete Systeme

- Eingebettete Systeme = „Computer, die man nicht unmittelbar sieht“
- z.B. in KfZ, Mobiltelefon ,Waschmaschine, Kleidung
- Wenig Ressourcen:
  - Kleiner Arbeitsspeicher
  - Geringer Stromverbrauch
- Meist Echtzeitanforderungen
- Bsp.: WindowsCE, Symbian, Linux, Arduino



Echtzeit-Betriebssysteme

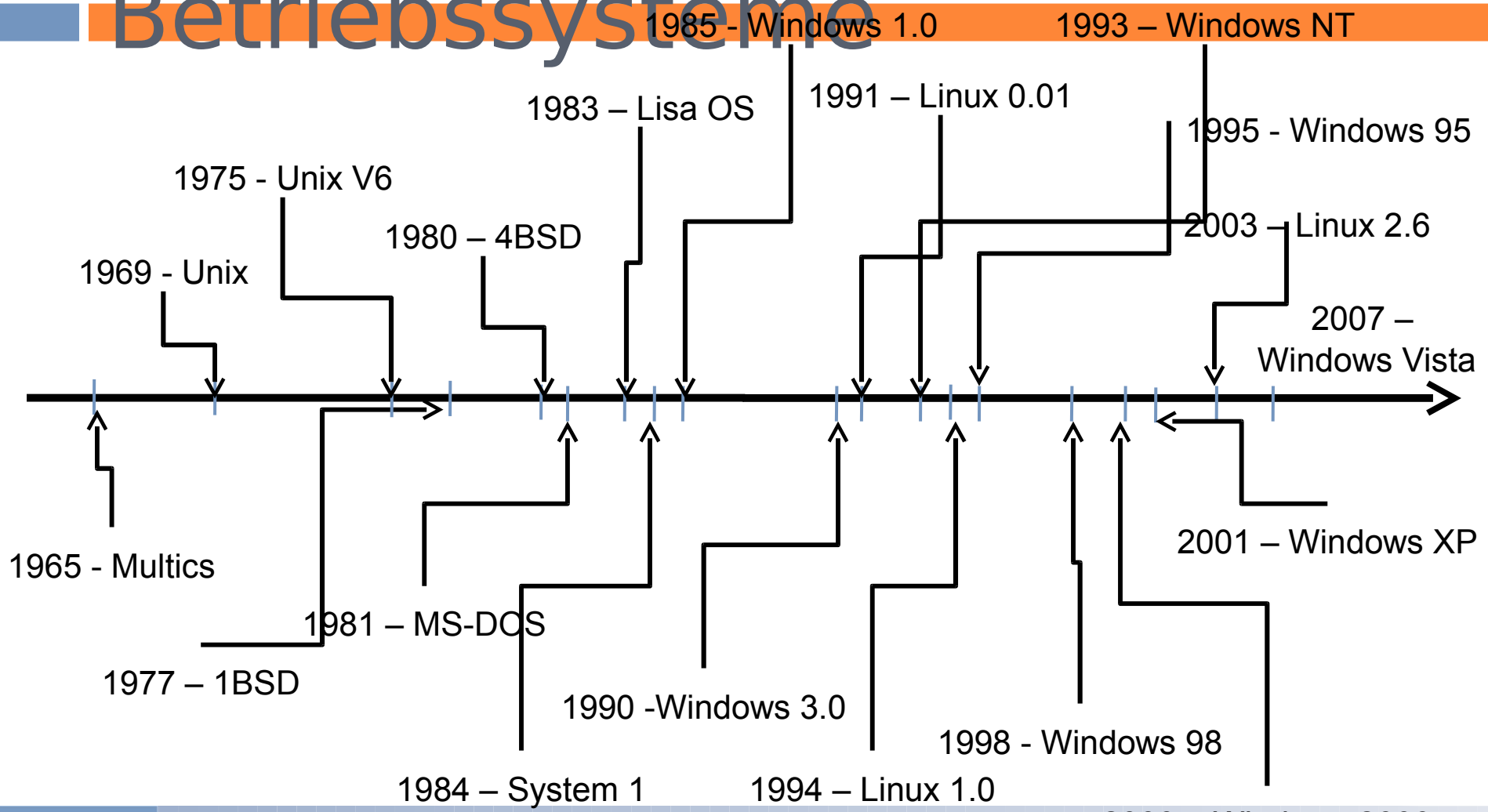
- Betriebssysteme zur Steuerung maschineller Fertigungsanlagen, medizintechnische Geräte, Luft & Raumfahrt, etc.
- Einhalten harter Zeitbedingungen:
  - nicht **im Durchschnitt** schnell, sondern
  - auch **im Worst Case** abschließen von Operationen in fest vorgegebenen Zeitintervallen



JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Bsp.: VxWorks, QNX, OSEK-

Geschichte der Betriebssysteme

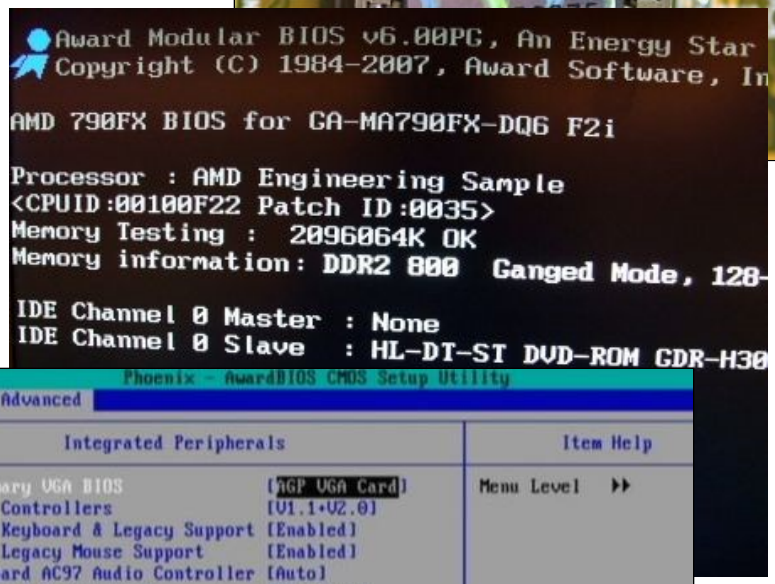


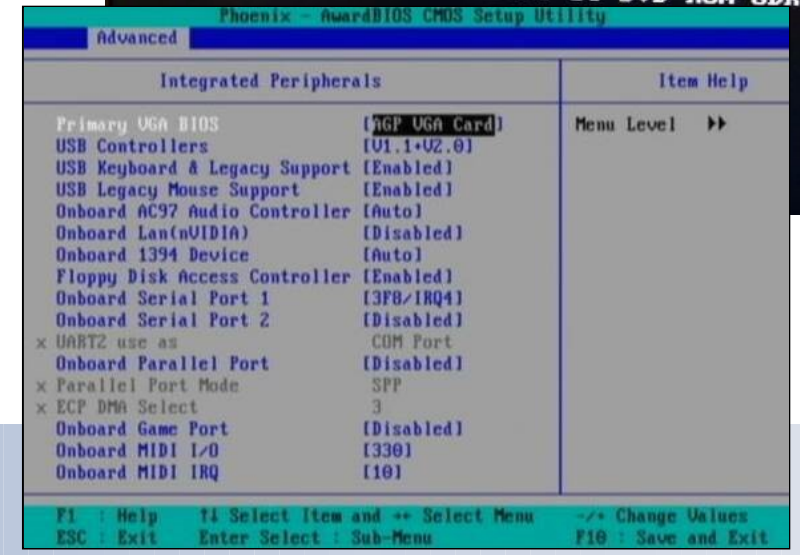
JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Startvorgang

BIOS (Basic Input Output System)

- BIOS-Speicherbaustein
  - Nicht flüchtig
  - Meist beschreibbar (Updatefähigkeit)
- BIOS wird direkt nach dem Einschalten ausgeführt
- Primäre Aufgaben
  - Low-level I/O
  - Ausführen des Boot-Loaders





JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

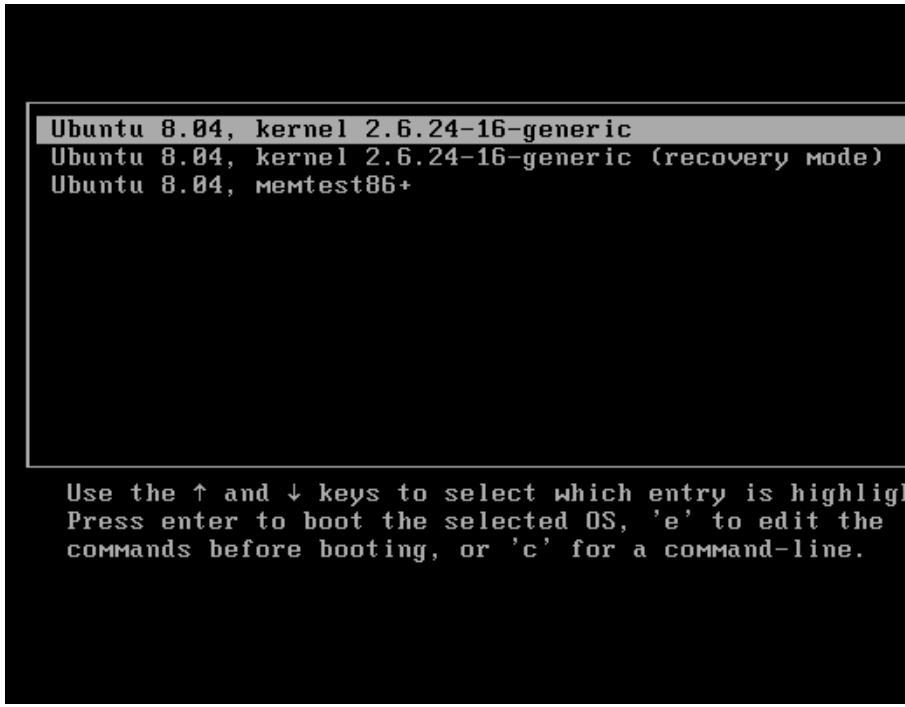
„No Keyboard connected, press any key to continue“

- Beim Bootprozess führt das BIOS folgende Schritte aus
  1. Überprüft wie viel RAM installiert ist
  2. Sucht nach angeschlossenen Geräten
  3. Speichert alle neu gefundenen Geräte in einem speziellen Speicher (CMOS)
  4. Wenn wichtige Geräte nicht vorhanden sind oder nicht reagieren, gibt es eine Warnung aus oder unterbricht den Bootprozess
  5. Bestimmt das Bootlaufwerk mittels durchlaufen einer im CMOS gespeicherten Liste
  6. Führt den auf diesem Laufwerk befindlichen Boot-Loader aus

JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Der Boot-Loader

- Boot-Loader
- Meist in den ersten Sektoren eines Laufwerkes
- Dient zum Starten des Betriebssystems
- Manchmal transparent für den Benutzer (zB. bei

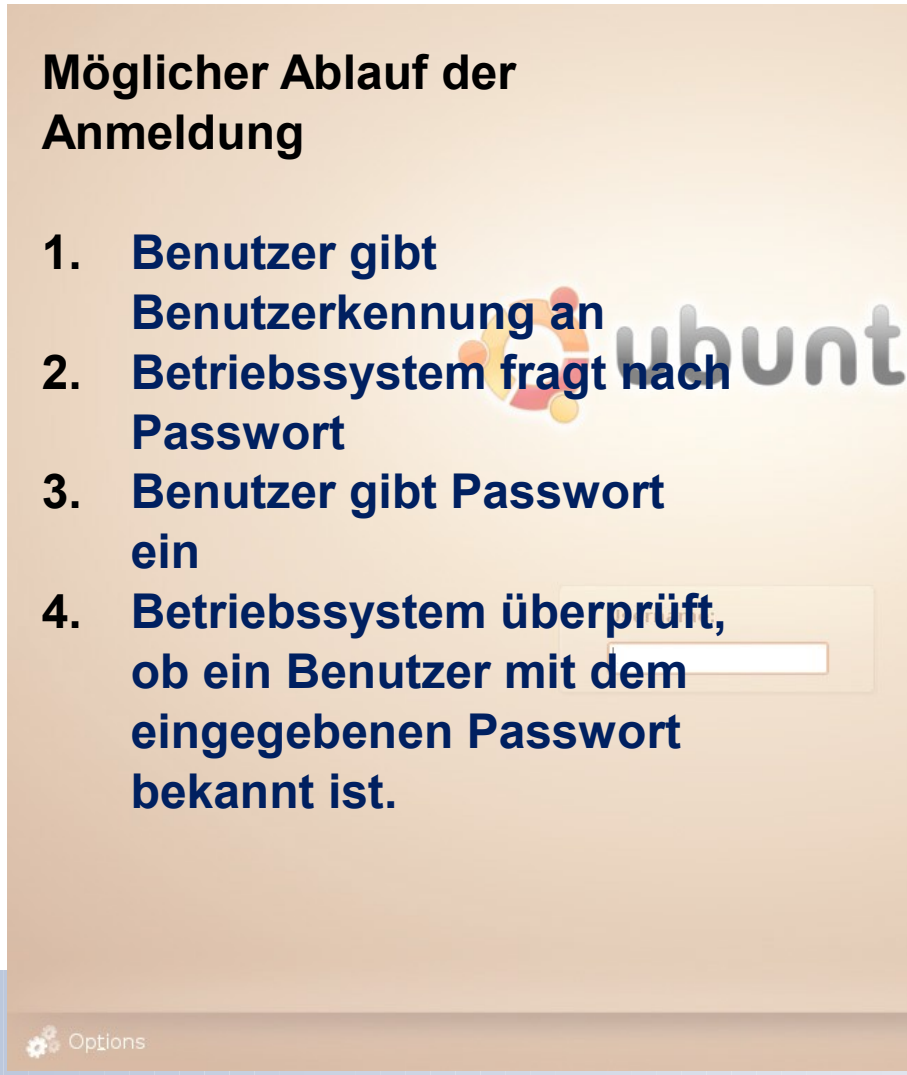


Anmeldung bei Multi User Systemen

- Motivation
  - Mehrere Benutzer benutzen ein System
  - Gegenseitiges Vertrauen?
- Aufgabe
  - Benutzer authentifizieren

Möglicher Ablauf der Anmeldung

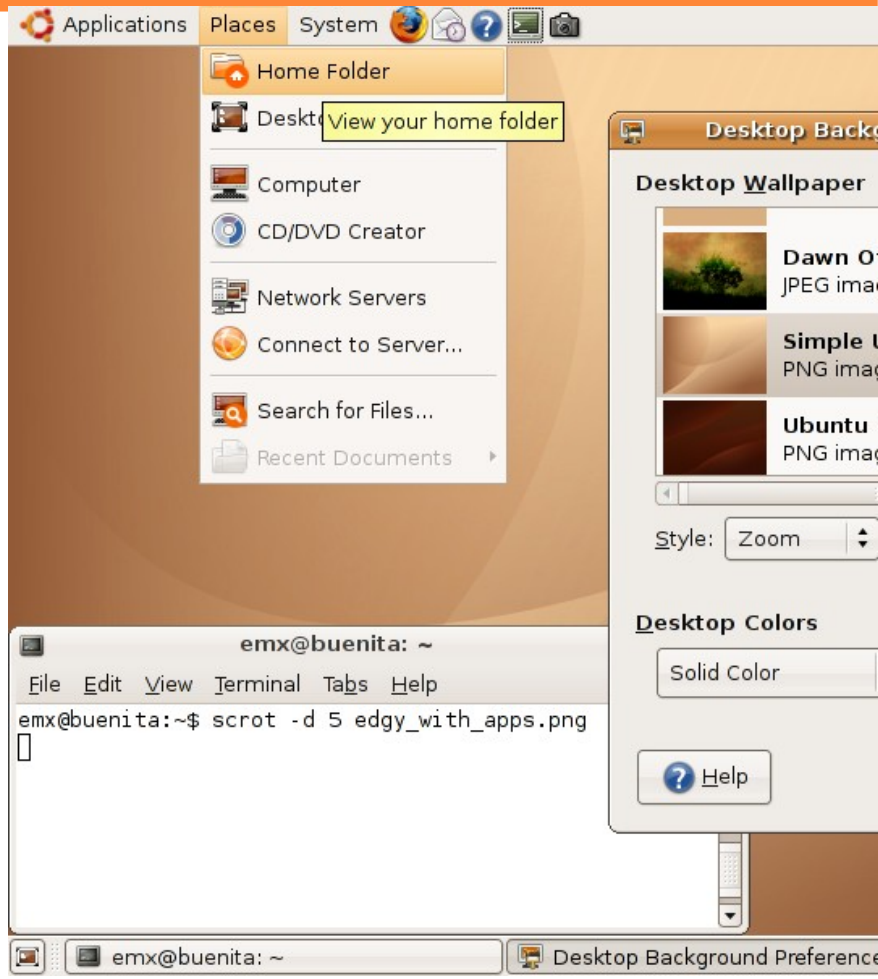
1. **Benutzer gibt Benutzerkennung an**
2. **Betriebssystem fragt nach Passwort**
3. **Benutzer gibt Passwort ein**
4. **Betriebssystem überprüft, ob ein Benutzer mit dem eingegebenen Passwort bekannt ist.**



JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Nach der Authentifizierung

- Ist der Benutzer berechtigt
  - Bestimmte Dateien zu lesen oder schreiben
  - Bestimmte Programme zu starten
- Betriebsmittelnutzung
  - können protokolliert und
  - dem Anwender



JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Verwaltung der Betriebsmittel

Exemplarisch: Prozessverwaltung

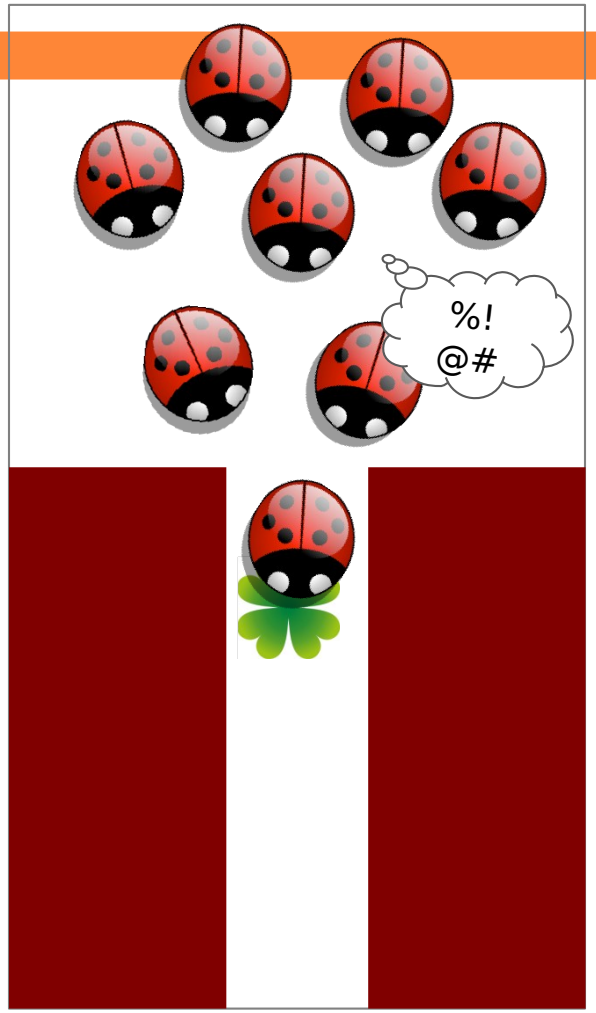
Mehrprogrammfähigkeit

F

- **Beobachtung:**
  - Viel Rechenzeit wird verschwendet durch Warten der CPU auf Beendigung von Ein- / Ausgabeoperationen.
  - Bis zu 90% der Zeit verschwendet bei Hochleistungs-Datenverarbeitung.
- **Idee:**
  - Führe aus Effizienzgründen Jobs nicht streng sequentiell aus.
  - Aufteilung des Speichers in mehrere Bereiche
  - Eigene Partition pro aktiven Job
  - Wartezeiten auf Beendigung von Ein- / Ausgabeoperationen genutzt durch Rechenzeit für andere Jobs.

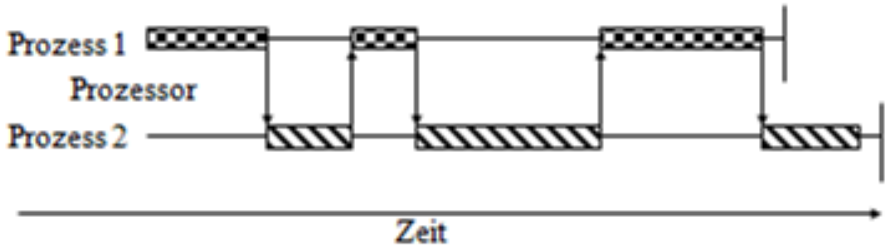
Verwaltung paralleler Prozesse

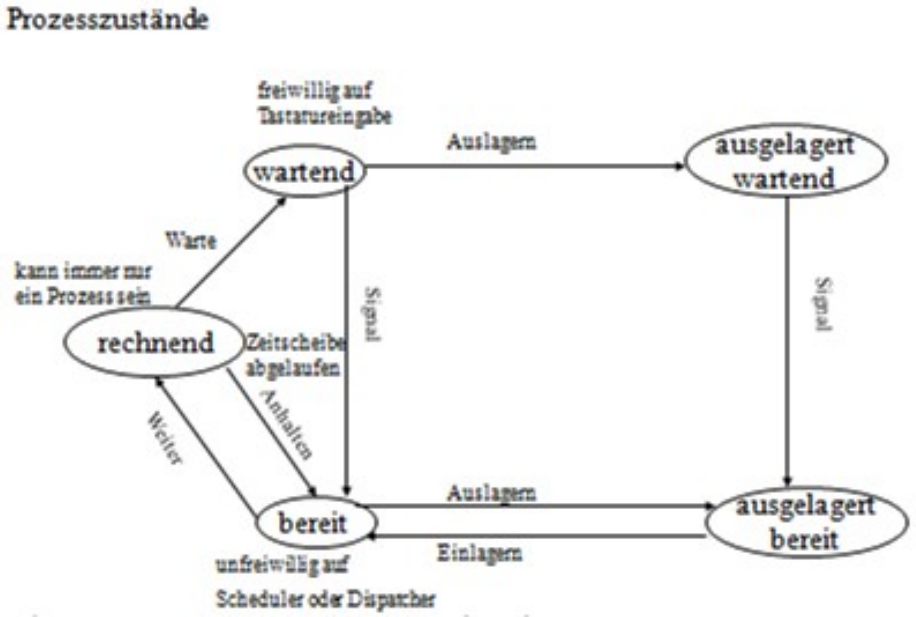
- Was ist ein Prozess?
  - Programm in der Ausführung
  - Einem Prozess ist ein bestimmter Adressraum zugeordnet
  - Benutzt gemeinsame Ressourcen wie CPU, RAM, Festplatte
- Problem
  - Mehrere Prozesse benötigen „gleichzeitig“ Rechenkapazitäten
  - Prozesse laufen unterschiedlich lange
  - Eine CPU kann nur einen Prozess gleichzeitig



Ausführung mehrerer Prozesse durch einen Rechner (Multitasking)

- Multitasking
  - Ausführung mehrerer Prozesse gleichzeitig
  - durch Zuteilung (Scheduling) von Prozessorzeit an jedem Prozess
- kooperatives Multitasking
  - jeder Prozess entscheidet selbst
  - Ein Prozess kann das gesamte System blockieren, wenn er nicht mehr ins Betriebssystem zurückkehrt!
- präemptives Multitasking
  - zentraler Dispatcher

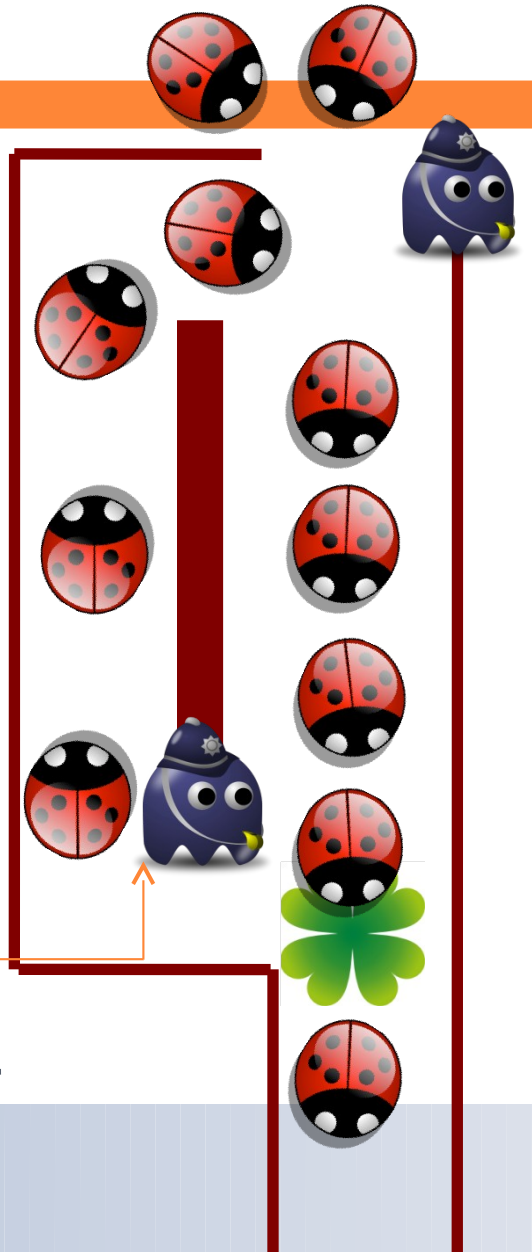




JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Präemptive Multitasking
Round Robin 1/4

- Idee von Round Robin
  - Prozesse wechseln sich ab
  - Jeder Prozess darf eine feste Zeiteinheit die CPU belegen
    - Nach Ablauf, nächster Prozess
    - Falls Prozess vorher fertig, nächster Prozess
- Es wird ein Zeitplan zur Nutzung erstellt
  - Nutzungsdauer
  - Nutzungszeitpunkt
  - Prozesse in Warteschlange
  - Priorität des Prozesses

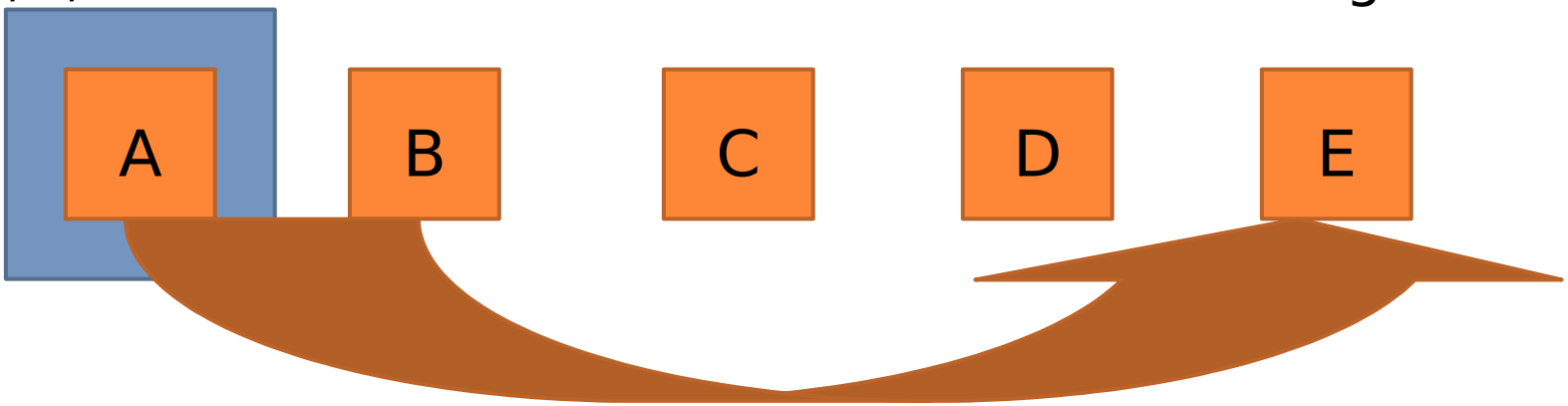


JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Präemptive Multitasking
Round Robin 2/4

Annahmen

- Anwendung A wird ausgeführt
- B,C,D und E befinden sich in einer Warteschlange



A wird ans Ende der Warteschlage verschoben

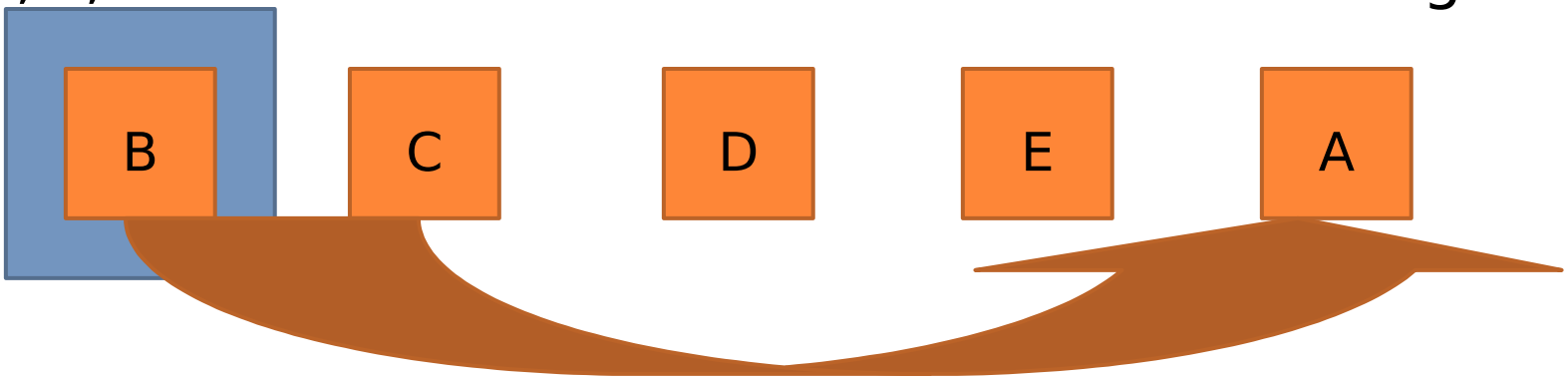
Die Ausführung von Anwendung A wird nach 10ms unterbroc

JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Präemptive Multitasking
Round Robin 3/4

Annahmen

- Anwendung B wird ausgeführt
- C,D,E und A befinden sich in der Warteschlange



B wird ans Ende der Warteschlage verschoben

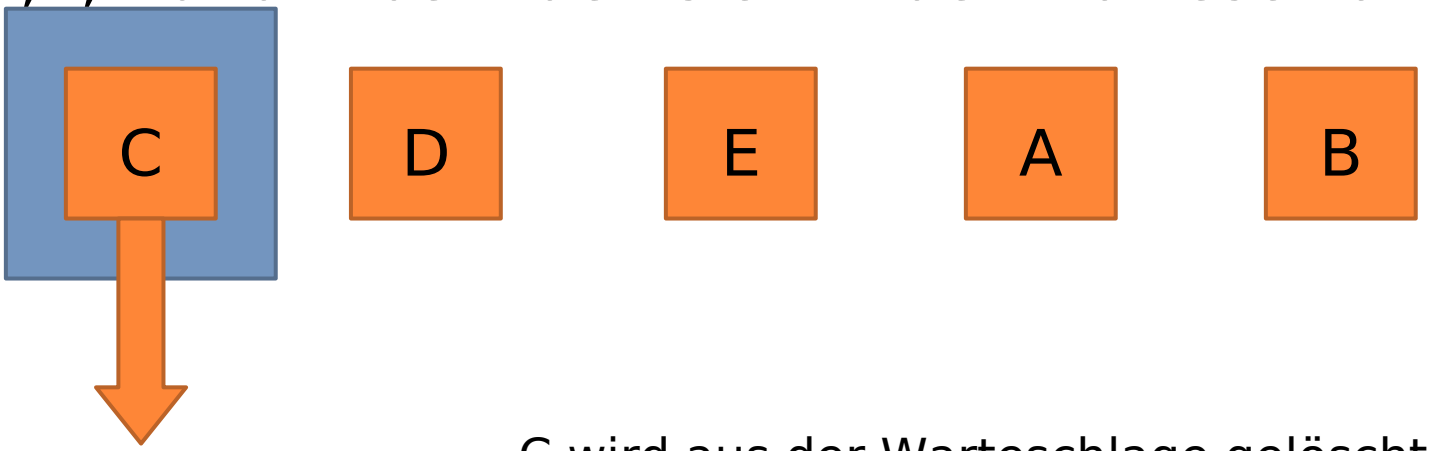
Die Ausführung von Anwendung B wird nach 10ms unterbro

JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Präemptive Multitasking
Round Robin 4/4

Annahmen

- Nun wird Anwendung C ausgeführt
- D,E,A und B befinden sich in der Warteschlange



C wird aus der Warteschlage gelöscht

C hat nach 3ms seine Berechnungen beendet

JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Prozess-Synchronisation
Annahmen

- Grundlage
  - Prozesse benötigen teilweise Informationen von anderen Prozessen
  - Informationen sind nicht immer verfügbar, sondern nur zur einer bestimmten Zeit
- ⇒ Synchronisation notwendig

- Annahmen für das folgende Beispiel
  - Prozess 1 ist ein Erzeuger
  - Prozess 2 ist ein Verbraucher
  - Prozess 1 erzeugt nur solange, bis das Lager voll ist
  - Prozess 2 kann nur konsumieren, solange das Lager noch gefüllt ist

JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Prozess-Synchronisation
Beispielprozesse

Erzeuger

```
while (true){
  while (anzahl < 5){
    neuesProdukt = new Produkt();
    lager[anzahl] = neuesProdukt;
    anzahl++;
  }
}
```

Verbraucher

```
while (true){
  while (anzahl > 0){
    produkt = lager[anzahl-1];
    lager[anzahl-1] = null;
    anzahl--;
    produkt.verbrauche();
  }
}
```

Problem

- While-Schleife eines Prozesses muss nicht am Stück ausgeführt werden

⇒ Inkonsistenten können auftreten

Prozess-Synchronisation
Ablauf

Erzeuger

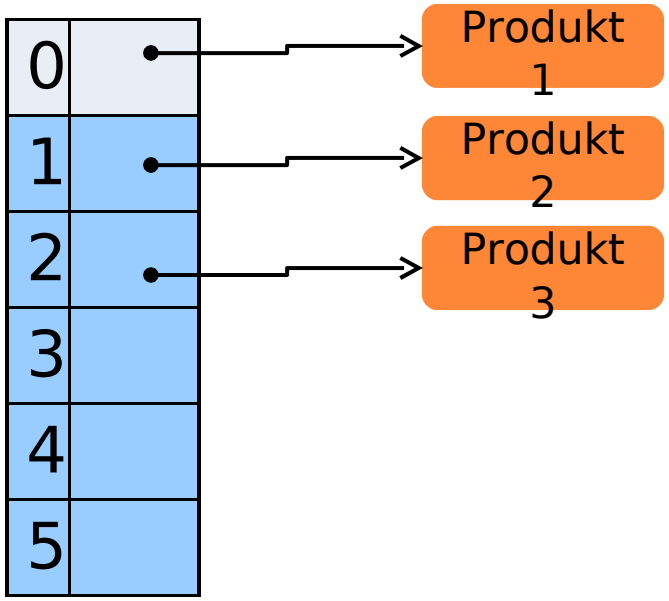
```
1: while (true){
2:   while (anzahl < 5){
3:    neuesProdukt = new Produkt();
4:    lager[anzahl] = neuesProdukt;
5:    anzahl++;
6:   }
7: }
```

Verbraucher

```
1: while (true){
2:   while (anzahl > 0){
3:    Produkt = lager[anzahl-1];
4:    lager[anzahl-1] = null;
5:    anzahl--;
6:  }
7: }
```

anzahl = 3

lager



JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Prozess-Synchronisation
Ablauf

Erzeuger

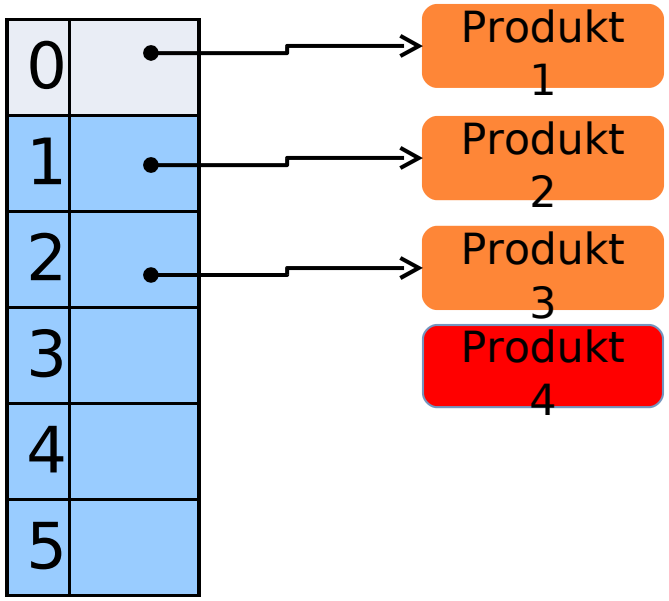
```
1: while (true){
2:   while (anzahl < 5){ < 5){
3:    neuesProdukt = new Produkt();
4:    lager[anzahl] = neuesProdukt;
5:    anzahl++;
6:   }
7: }
```

Verbraucher

```
1: while (true){
2:   while (anzahl > 0){
3:    produkt = lager[anzahl-1];
4:    lager[anzahl-1] = null;
5:    anzahl--;
6:  }
7: }
```

anzahl = 3

lager



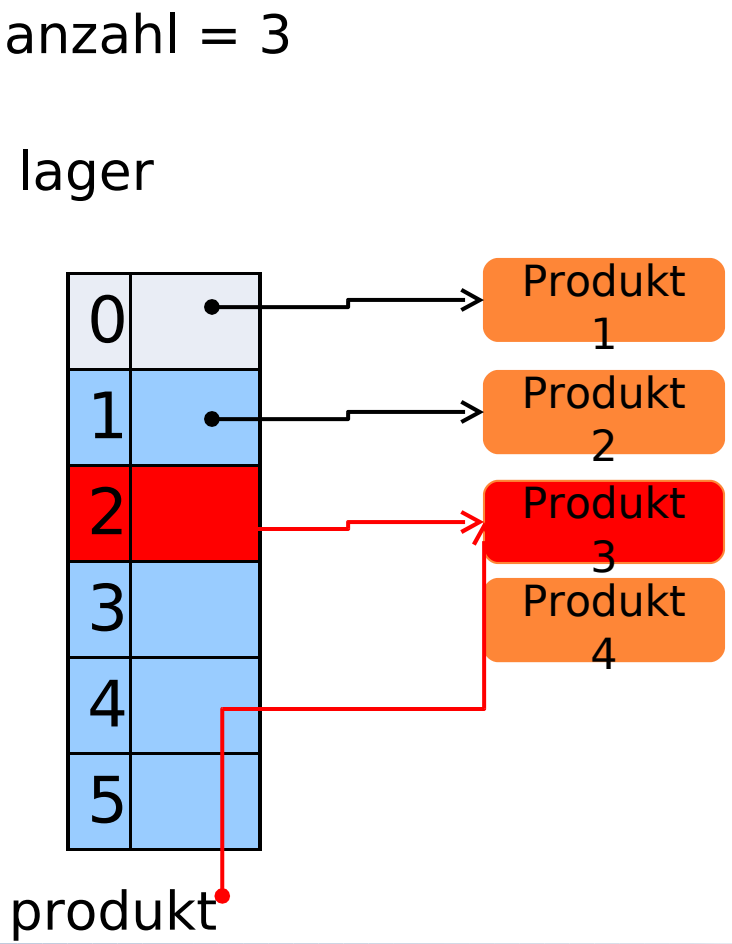
Prozess-Synchronisation
Ablauf

Erzeuger

```
1: while (true){
2:   while (anzahl < 5){5){
3:    neuesProdukt = new Produkt();
4:    lager[anzahl] = neuesProdukt;
5:    anzahl++;
6:   }
7: }
```

Verbraucher

```
1: while (true){
2:   while (anzahl > 0){
3:    produkt = lager[anzahl-1];
4:    lager[anzahl-1] = null;
5:    anzahl--;
6:  }
7: }
```



JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

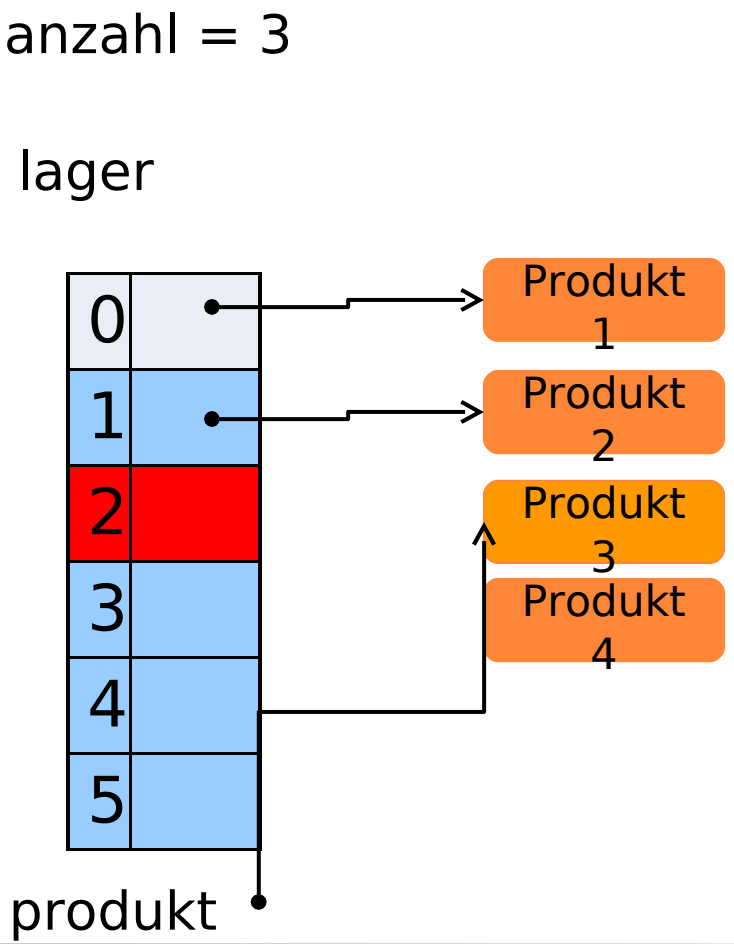
Prozess-Synchronisation
Ablauf

Erzeuger

```
1: while (true){
2:   while (anzahl < 5){5){
3:    neuesProdukt = new Produkt();
4:    lager[anzahl] = neuesProdukt;
5:    anzahl++;
6:   }
7: }
```

Verbraucher

```
1: while (true){
2:   while (anzahl > 0){
3:    produkt = lager[anzahl-1];
4:    lager[anzahl-1] = null;
5:    anzahl--;
6:  }
7: }
```



JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

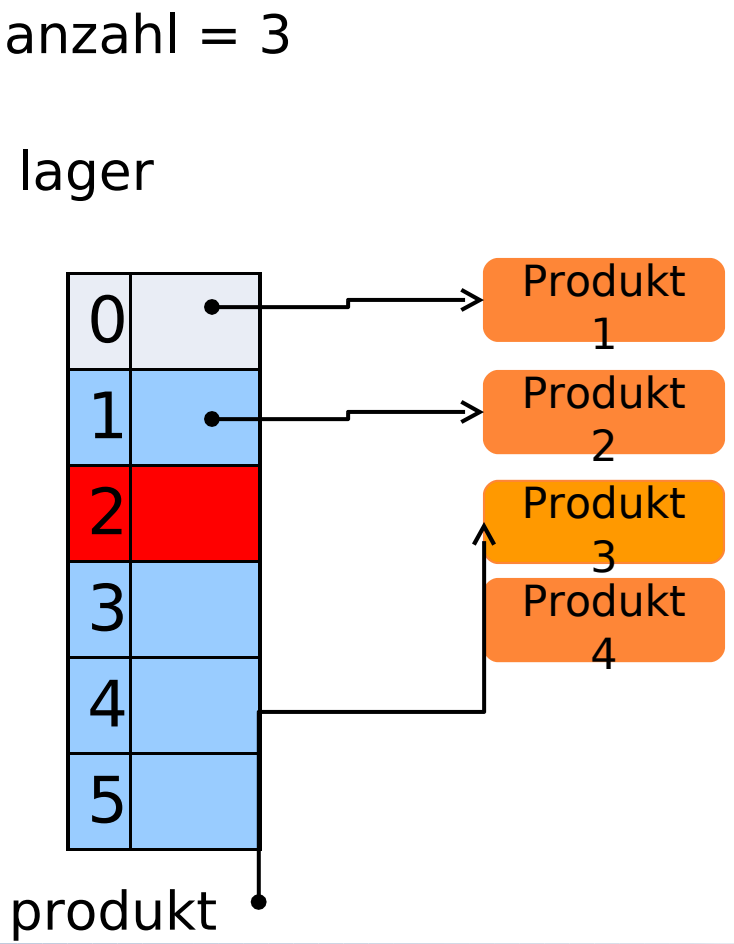
Prozess-Synchronisation
Ablauf

Erzeuger

```
1: while (true){
2:   while (anzahl < 5){5){
3:    neuesProdukt = new Produkt();
4:    lager[anzahl] = neuesProdukt;
5:    anzahl++;
6:   }
7: }
```

Verbraucher

```
1: while (true){
2:   while (anzahl > 0){
3:    produkt = lager[anzahl-1];
4:    lager[anzahl-1] = null;
5:    anzahl--;
6:  }
7: }
```



JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

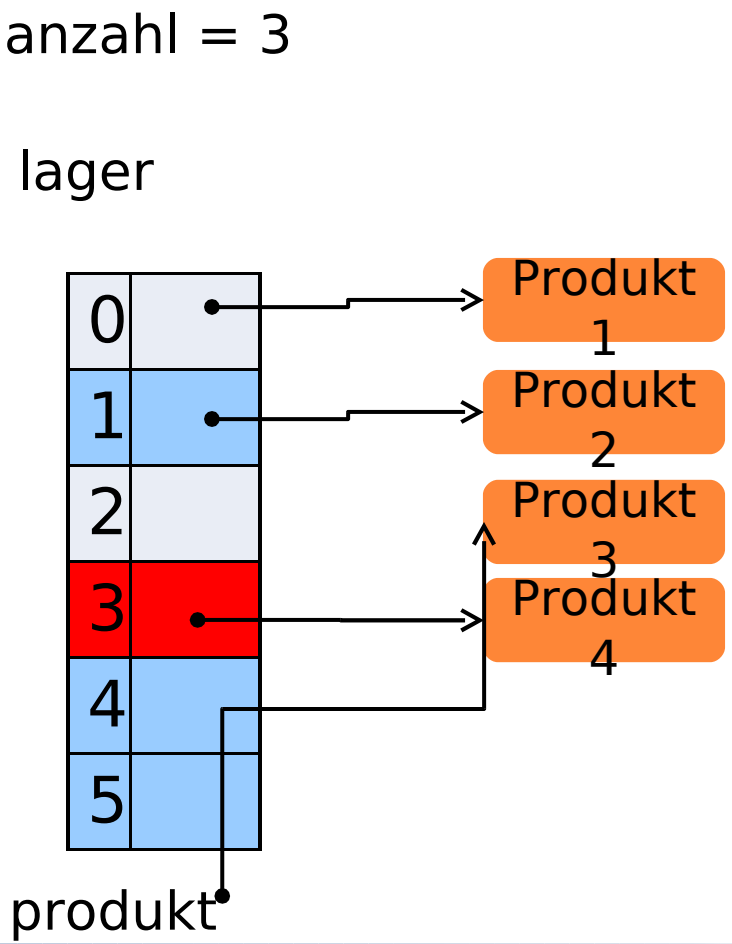
Prozess-Synchronisation
Ablauf

Erzeuger

```
1: while (true){
2:   while (anzahl < 5){5){
3:    neuesProdukt = new Produkt();
4:    lager[anzahl] = neuesProdukt;
5:    anzahl++;
6:   }
7: }
```

Verbraucher

```
1: while (true){
2:   while (anzahl > 0){
3:    produkt = lager[anzahl-1];
4:    lager[anzahl-1] = null;
5:    anzahl--;
6:  }
7: }
```



JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

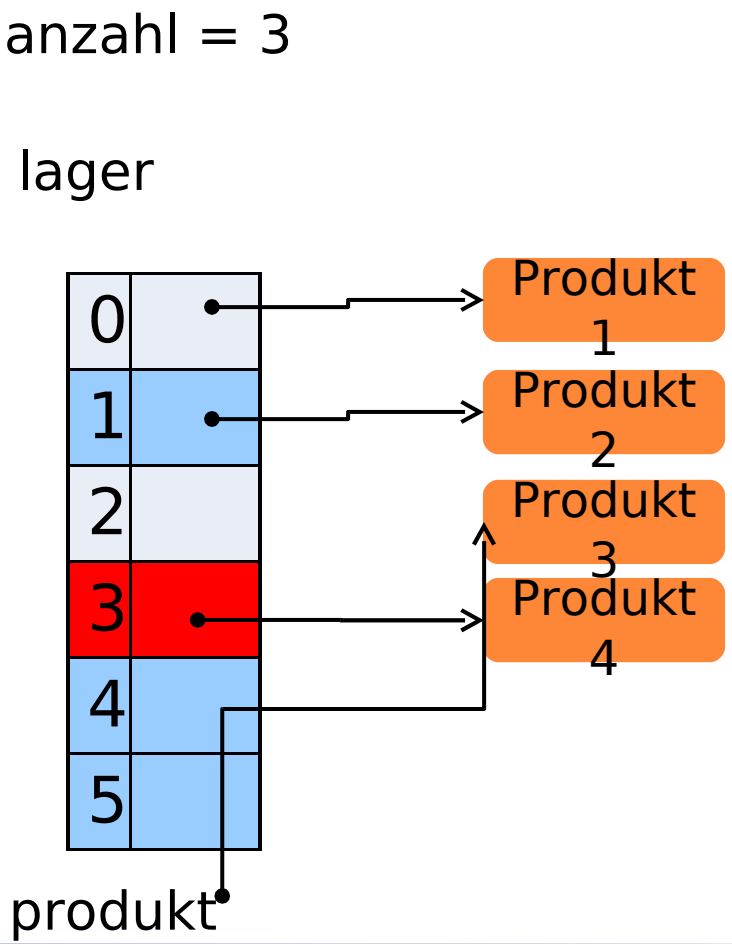
Prozess-Synchronisation
Ablauf

Erzeuger

```
1: while (true){
2:   while (anzahl < 5){5){
3:    neuesProdukt = new Produkt();
4:    lager[anzahl] = neuesProdukt;
5:    anzahl++;
6:   }
7: }
```

Verbraucher

```
1: while (true){
2:   while (anzahl > 0){
3:    produkt = lager[anzahl-1];
4:    lager[anzahl-1] = null;
5:    anzahl--;
6:  }
7: }
```



JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

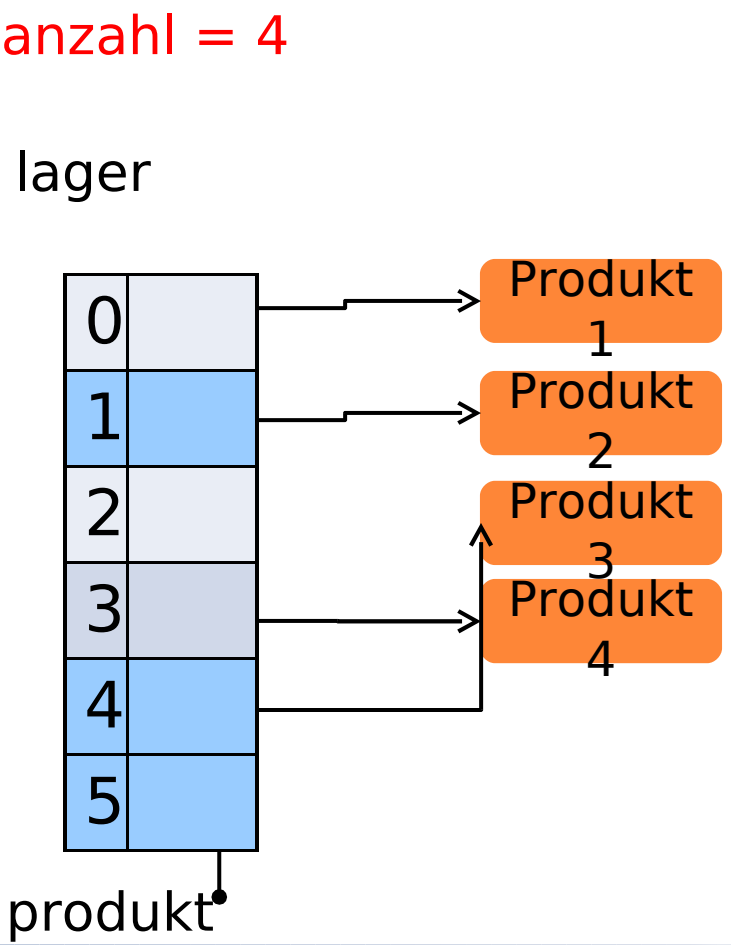
Prozess-Synchronisation
Ablauf

Erzeuger

```
1: while (true){
2:   while (anzahl < 5){5){
3:    neuesProdukt = new Produkt();
4:    lager[anzahl] = neuesProdukt;
5:    anzahl++;
6:   }
7: }
```

Verbraucher

```
1: while (true){
2:   while (anzahl > 0){
3:    produkt = lager[anzahl-1];
4:    lager[anzahl-1] = null;
5:    anzahl--;
6:  }
7: }
```



JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

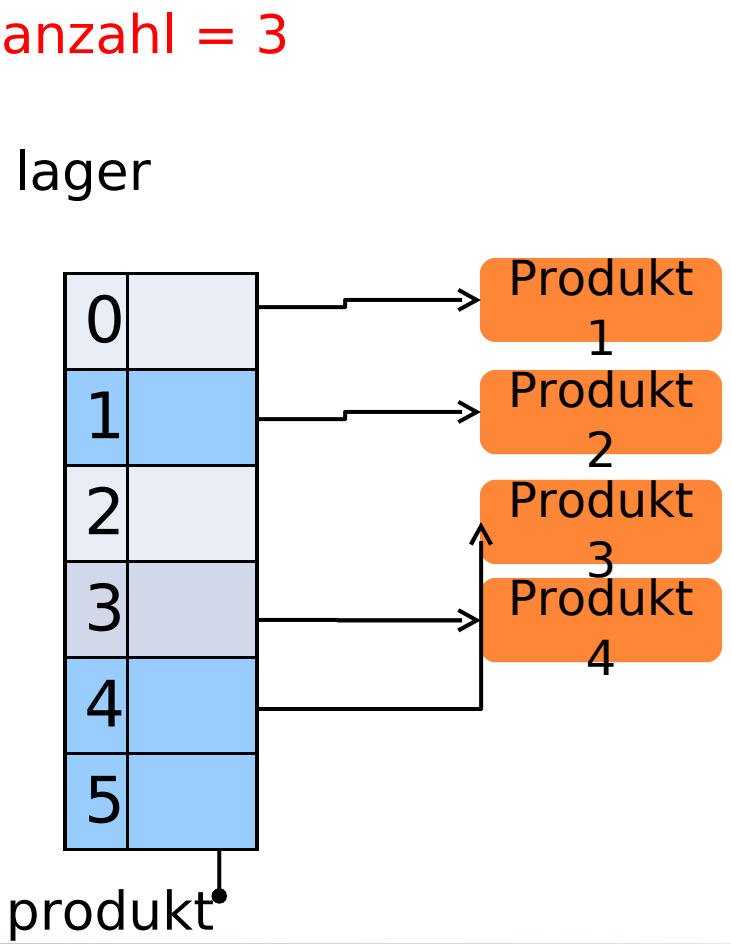
Prozess-Synchronisation
Ablauf

Erzeuger

```
1: while (true){
2:   while (anzahl < 5){5){
3:    neuesProdukt = new Produkt();
4:    lager[anzahl] = neuesProdukt;
5:    anzahl++;
6:   }
7: }
```

Verbraucher

```
1: while (true){
2:   while (anzahl > 0){
3:    produkt = lager[anzahl-1];
4:    lager[anzahl-1] = null;
5:    anzahl--;
6:  }
7: }
```



JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Prozess-Synchronisation 4/4 - Lösungsansätze

- Ursache
  - Programm in den Schleifen wird nicht zwangsweise am Stück ausgeführt
- Idee
  - Kennzeichne „kritische Bereiche“, die nicht unterbrochen werden dürfen
- Mögliche Ansätze
  - Sperren (Locks)
  - Semaphoren
  - Monitore

Verbraucher

```
while (true){
  while (anzahl > 0){
    produkt = lager[anzahl-1];
    lager[anzahl-1] = null;
    anzahl--;
  }
}
```

kritischer Bereich

Erzeuger

```
while (true){
  while (anzahl < 5){
    neuesProdukt = new Produkt();
    lager[anzahl] = neuesProdukt;
    anzahl++;
  }
}
```

kritischer Bereich

JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Idee der Sperre

- Führe atomare Funktion ein, die angibt, ob ein kritischer Bereich betreten werden darf
- Beim Betreten des kritischen Bereiches, wird Sperre gesetzt
- Beim Verlassen des kritischen Bereiches, wird Sperre aufgehoben

atomare Ausführung

```
public boolean testUndSperre(){
    //Sperre bereits aktiv?
    if (sperre == true)
         //Prozess kritischen Bereich verbieten
        return true;
    else{
        sperre = true; //Sperre aktivieren
        //Prozess kritischen Bereich erlauben
        return false;
    }
}
```

```
public void freigabe(){
    sperre = false;
}
```

Idee der Sperre

Erzeuger

```
while (true){
  while (anzahl < 5 &&
         testUndSperre()){
    neuesProdukt = new Produkt();
    lager[anzahl] = neuesProdukt;
    anzahl++;
    freigabe()
    }
  }
}
```

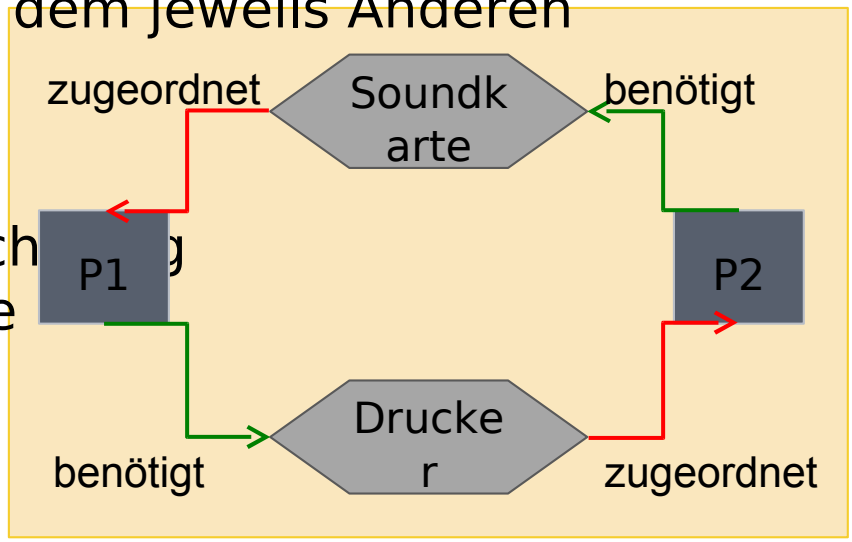
Verbraucher

```
while (true){
  while (anzahl > 0 &&
         testUndSperre()){
    produkt = lager[anzahl-1];
    lager[anzahl-1] = null;
    anzahl--;
    produkt.verbrauche();
    freigabe()
  }
}
```

!!! Beim parallelen Zugriff auf gemeinesame Ressourcen das Problem der Deadlocks beachten!!!

Problem
Deadlocks/Verklemmungen

- Grundlegendes Problem
  - Zwei unterschiedliche Prozesse benötigen mehrere Betriebsmittel
  - Das benötigte Betriebsmittel ist dem jeweils Anderen zugeordnet
- Annahme
  - P1 und P2 benötigen beide gleich[illegible] den Drucker und die Soundkarte
    - P1 ist der Drucker zugeteilt
    - P2 ist die Soundkarte
- Mögliche Lösungen
  - Anforderung der Betriebsmittel muss zusammen (atomar) geschehen
    - → Zentrale Entität entzieht Betriebsmittel



JProf. Dr. Gunnar Stevens
Human Computer Interaction, University of Siegen
gunnar.stevens@uni-siegen.de

Nächstes Mal: Verteilte Systeme