Einführung in die Wirtschaftsinformatik I

WS 2024/25

Systemsoftware

Dr. Rainer Wieching

Zusammenfassung: Rechnersysteme

Client-Server-Paradigmen

Von-Neumann-Rechner-Architektur

Bus-Systeme

SI-Einheiten und Binärpräfixe

Speichertechnologien

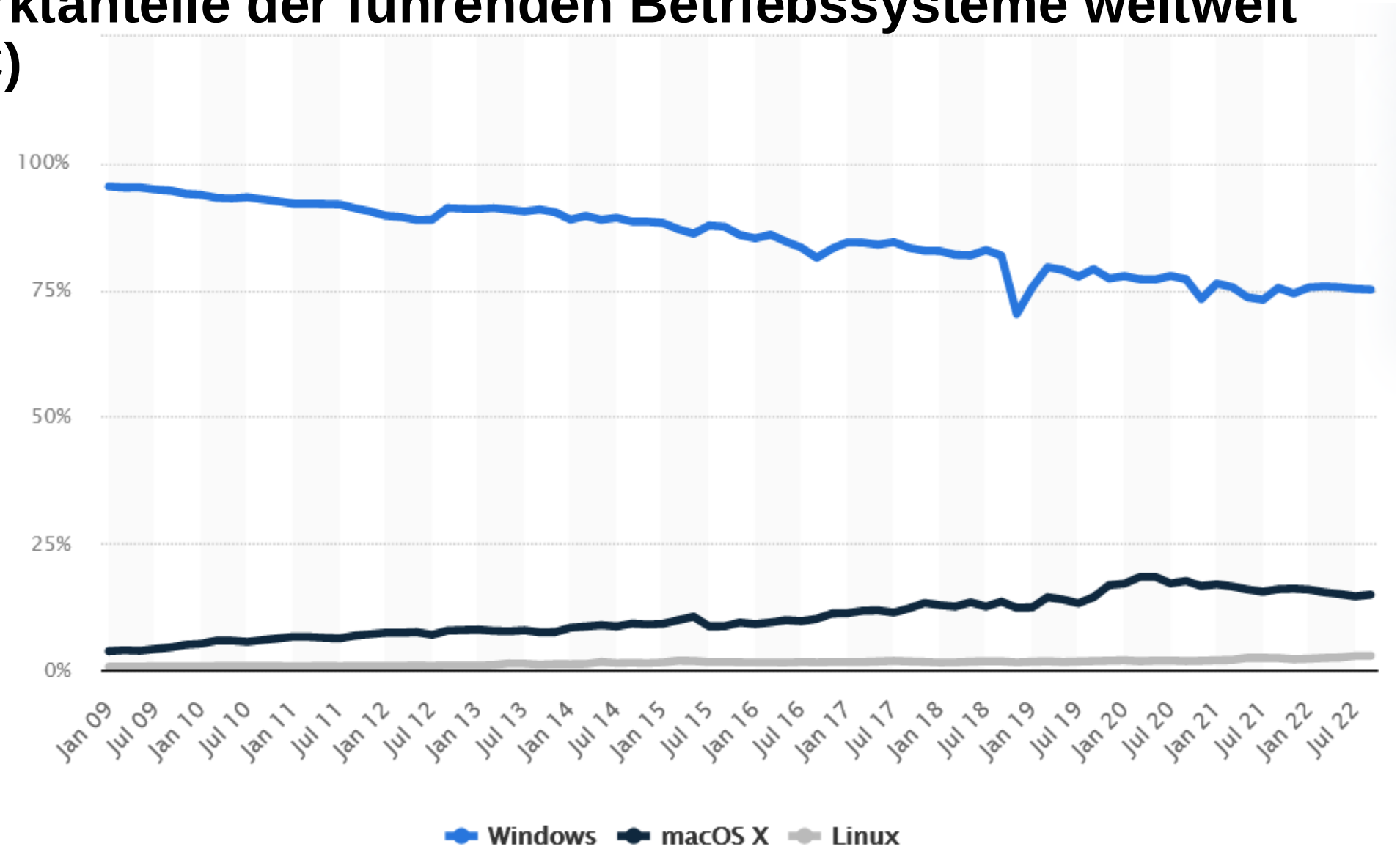
Ergonomie des Tastaturlayouts

Strichcodes und Matrixcodes

Ausgabe

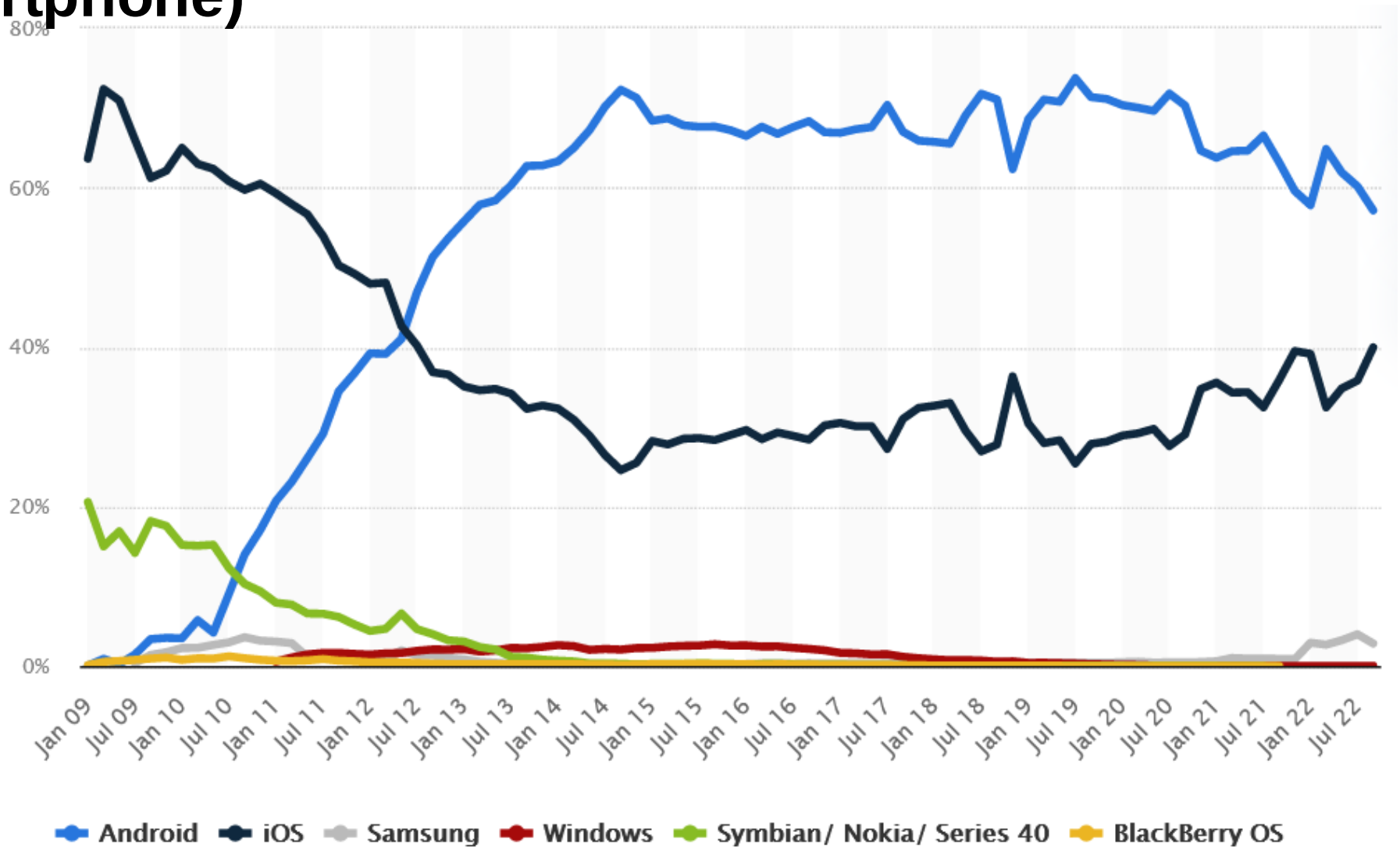
Welches Betriebssystem ist auf den meisten Geräten installiert: Windows, Linux oder MacOS?

Marktanteile der führenden Betriebssysteme weltweit (PC)



https://de.statista.com/statistik/daten/studie/157902/umfrage/marktanteil-der-genutzten-betriebssysteme-weltweit-seit-2009/

Marktanteile der führenden Betriebssysteme weltweit (Smartphone)



https://de.statista.com/statistik/daten/studie/184332/umfrage/marktanteil-der-mobilen-betriebssysteme-in-deutschland-seit-2009/

Linux als führendes Betriebssystem















Images: https://www.irishmirror.ie/lifestyle/technology/best-drones-buy-2018-top-12253446 https://praxistipps-images.chip.de/67IE77Bn7-wTR9VuuWD2z-eZ4uE=/0x0/filters:format(jpeg):fill(000,true):no_upscale()/praxistippss3.amazonaws.com%2Fsmart-tv-apps-installieren_f650ff0d.jpg http://mediad.publicbroadcasting.net/p/wypn/files/201602/Lux_Products_TX9600TS_Thermostat.jpg https://static.digitecgalaxus.ch/Files/6/1/4/3/8/5/4/616y1qzHetL._SL1000_.jpg?fit=inside%7C270:387&output-format=progressive-jpeg https://i.otto.de/i/otto/21314491/siemens-kaffeevollautomat-eq-6-plus-s400-te654509de-mit-milchbehaelter-im-wert-von-uvp-49-90-schwarz.jpg?$formatz$ https://www.chip.de/ii/7/0/3/9/7/9/4/2/6e42e16942a312fc.jpeg

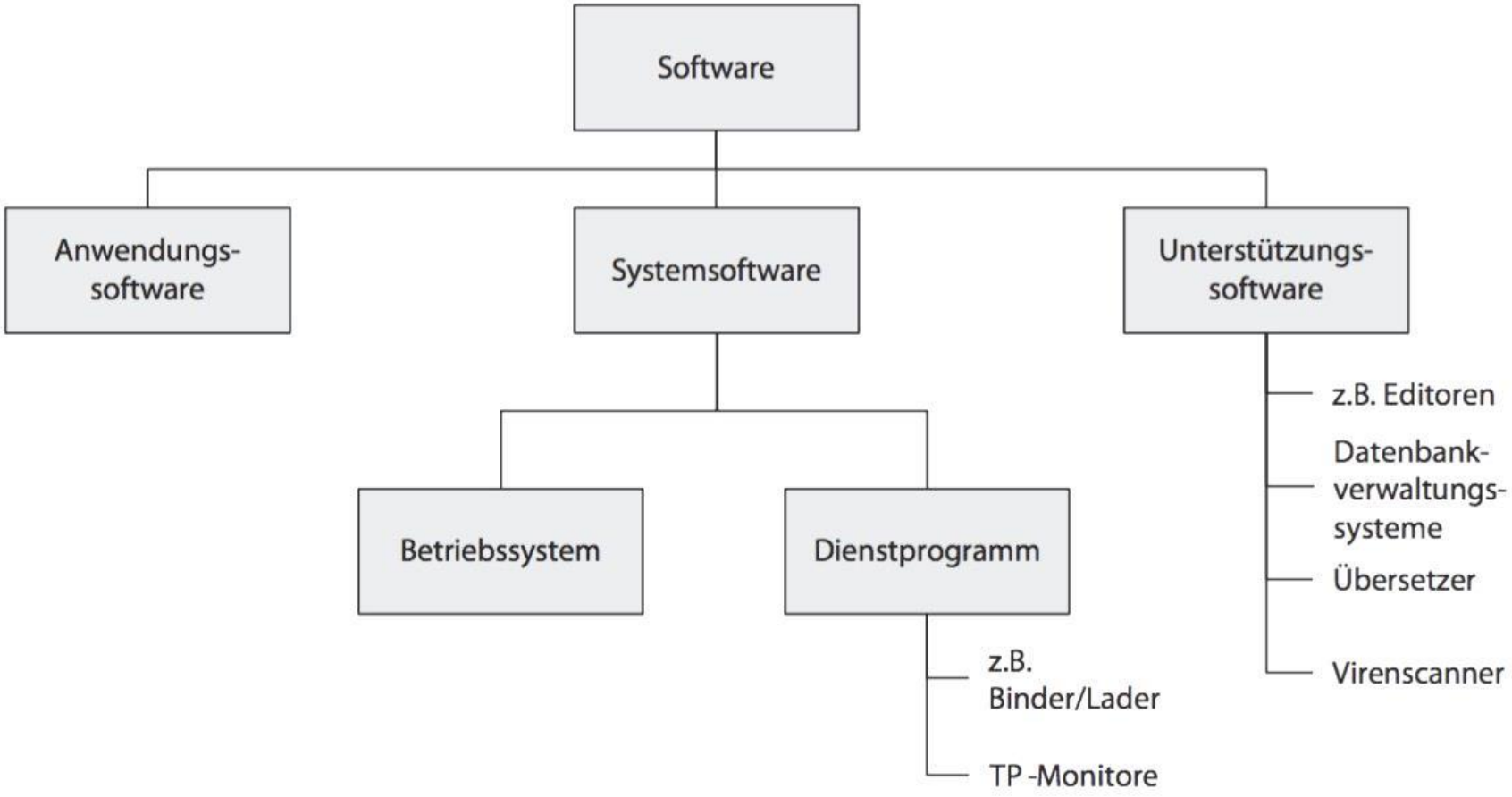
Gliederung – Teil 1

1. Arten von Software

2. Betriebssystem im Überblick

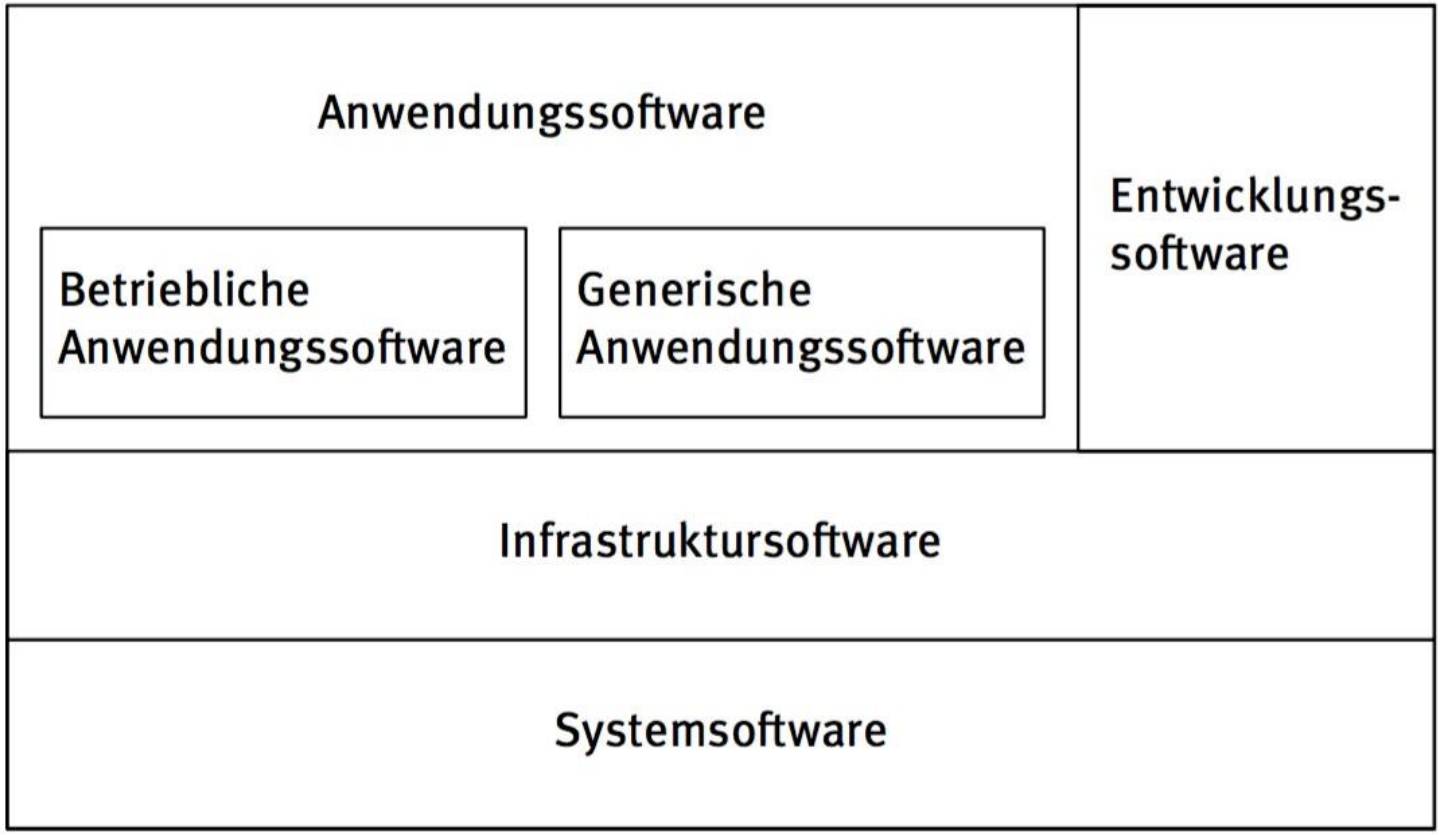
3. Aufgaben des Betriebssystems

Es gibt verschiedene Arten von Software
Klassifikation nach ISO/IEC 2382-1



Leimeister, S. 68

Es gibt verschiedene Arten von Software



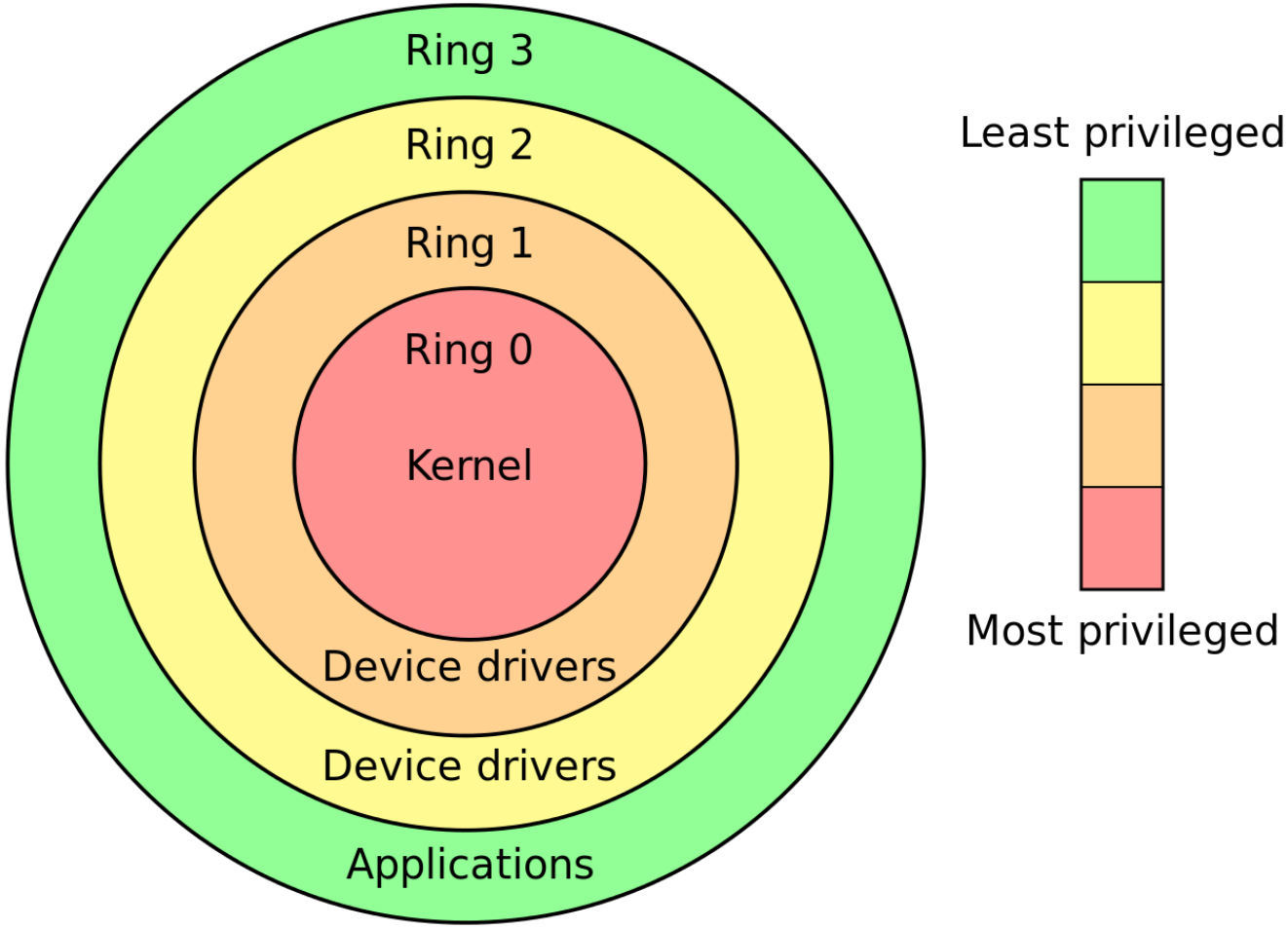
Hansen, S. 535

Systemsoftware

Die **Systemsoftware** stellt die grundlegenden Funktionen für den Betrieb eines Rechners zur Verfügung. Wichtige Bestandteile sind das **Betriebssystem** (engl.: operating system) und Komponenten für die Ansteuerung von Geräten **(Treiber)**, falls diese nicht direkt vom Betriebssystem unterstützt werden. Es ist auf die jeweilige Rechnerplattform abgestimmt und zum Betrieb eines Rechners unbedingt nötig. Es steuert und überwacht die Ausführung von Anwendungsprogrammen auf der Maschinenkonfiguration (der Rechnerhardware).

Hansen, S. 535 ff.

Zur Zugriffskontrolle werden die ausgeführten Programme verschiedenen Ringen zugewiesen.

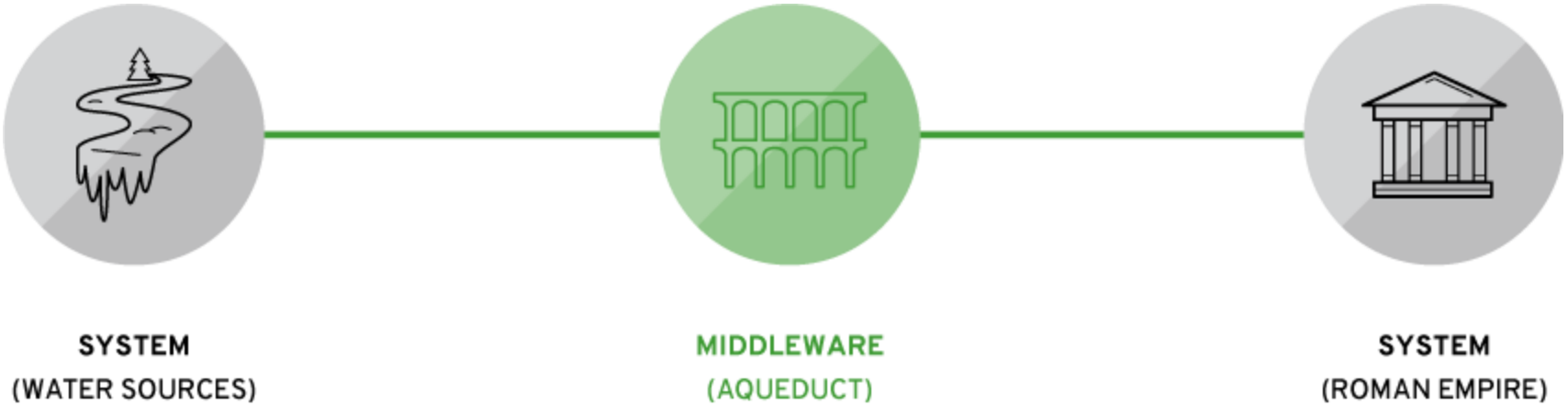


https://en.wikipedia.org/wiki/File:Priv_rings.svg

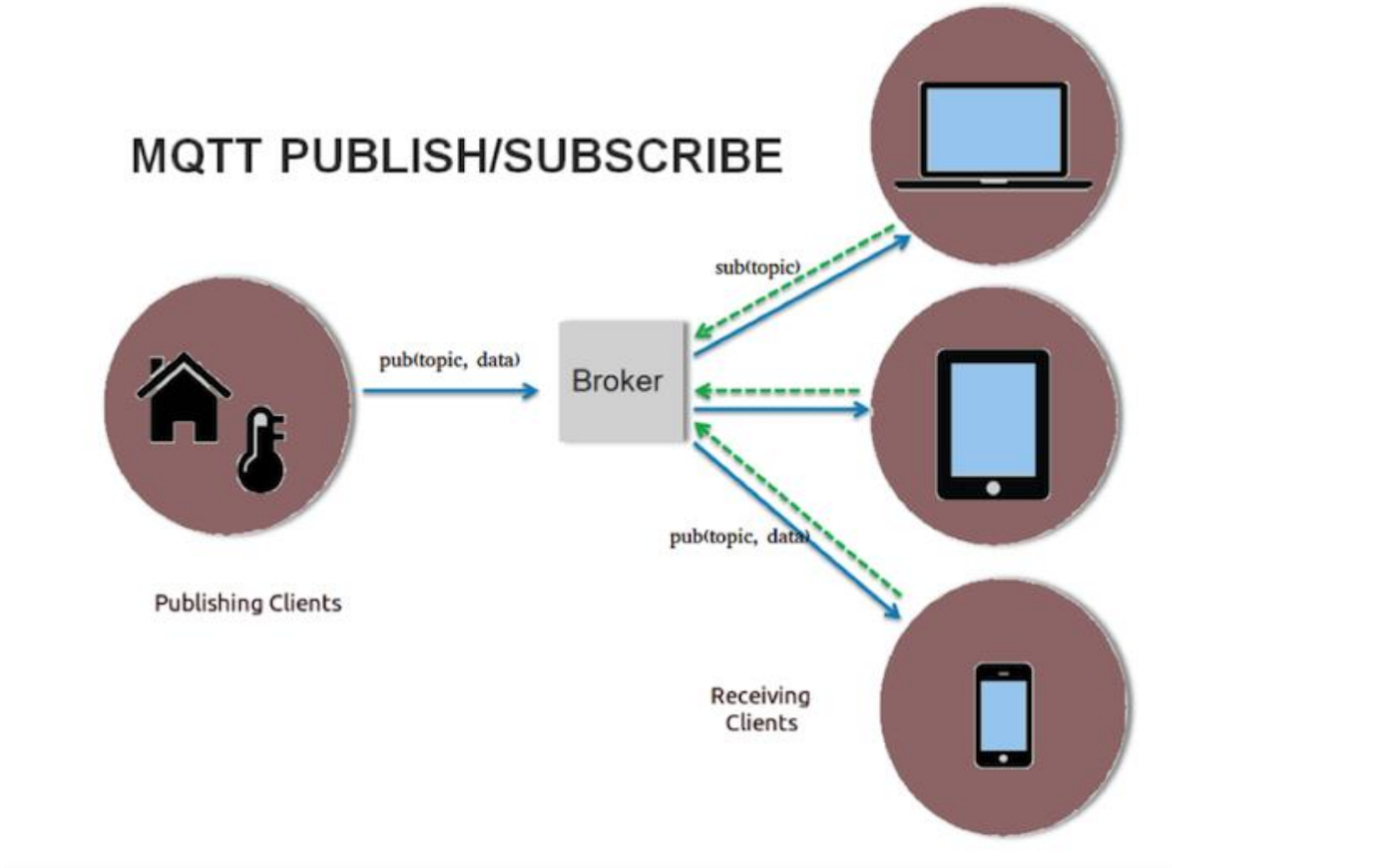
Infrastruktursoftware

Die **Infrastruktursoftware** (Synonym: Middleware) unterstützt andere Komponenten. Sie vereinfacht die Entwicklung und den Betrieb von Anwendungssoftware. Zu den Infrastrukturkomponenten zählen zum Beispiel Datenbankverwaltungssysteme, Web- und E-Mail-Server, grafische Benutzerschnittstellen oder Komponenten zur Netzwerkkommunikation. Bildlich gesprochen befindet sich die Middleware somit **„in der Mitte“** zwischen den Softwarekomponenten, die sie verbindet. Die Infrastruktursoftware baut auf dem Betriebssystem auf.

Hansen, S. 535 ff.



https://www.redhat.com/cms/managed-files/diagram-middleware-aqueduct-850x219.png



https://wso2.com/files/mqtt-article-buddhima.png

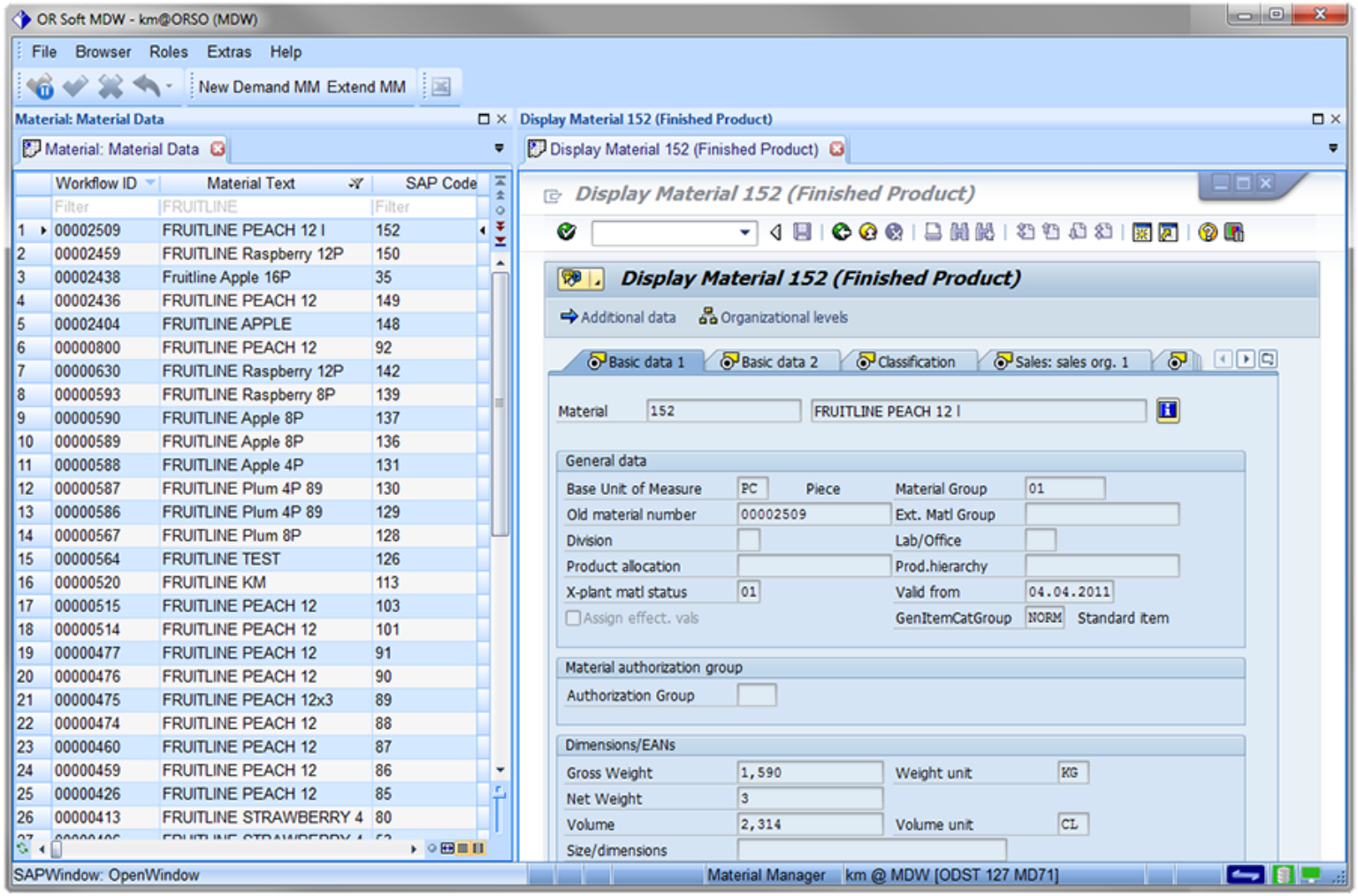
Anwendungssoftware

Die **Anwendungssoftware** (engl.: application software; Synonym: Applikation) bietet Lösungen für fachliche Probleme. Während **betriebliche Anwendungssoftware** direkt die Funktionen und Abläufe eines Betriebs unterstützt (z.B. ERP-Software), bietet **generische Anwendungssoftware** Funktionen, die innerhalb und außerhalb eines Betriebs genutzt werden. Zu der generischen Anwendungssoftware gehören z.B. Textverarbeitungs- oder Tabellenkalkulationssoftware, E-Mail-Programme oder Webbrowser.

Hansen, S. 535 ff.



https://pc-probleme-pforzheim.de/wp-content/uploads/2016/07/office.jpg

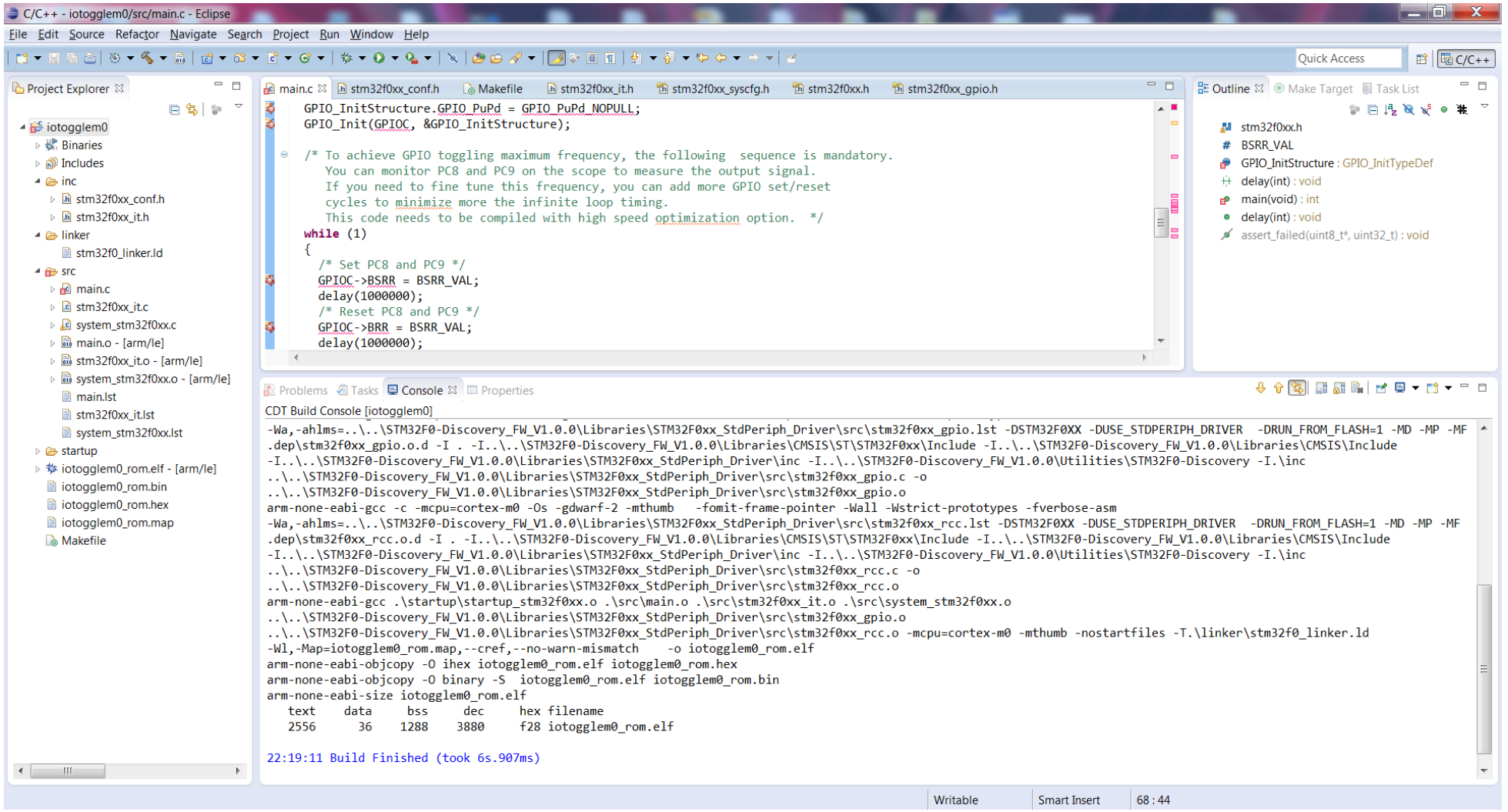


https://www.orsoft.net/sites/default/files/inline-images/MDW_SAP_Integration.png

Entwicklungssoftware

Die **Entwicklungssoftware** (engl.: development software) unterstützt die Programmierung prinzipiell beliebiger Programme. Ein wichtiger Teil einer Entwicklungsumgebung ist die zugehörige Programmiersprache, die aus zumindest einem Compiler oder Interpreter besteht, aber auch häufig angepasste Editoren, eine Ausführungsumgebung und Fehlerbehebungssoftware (sog. Debugger) enthält. Mit Hilfe der Entwicklungssoftware werden sowohl System-, Anwendungs-, als auch die Entwicklungssoftware selbst erstellt.

Hansen, S. 535 ff.



http://www.hertaville.com/files/uploads/2012/06/art2image12v21.png

Gliederung

1. Arten von Software

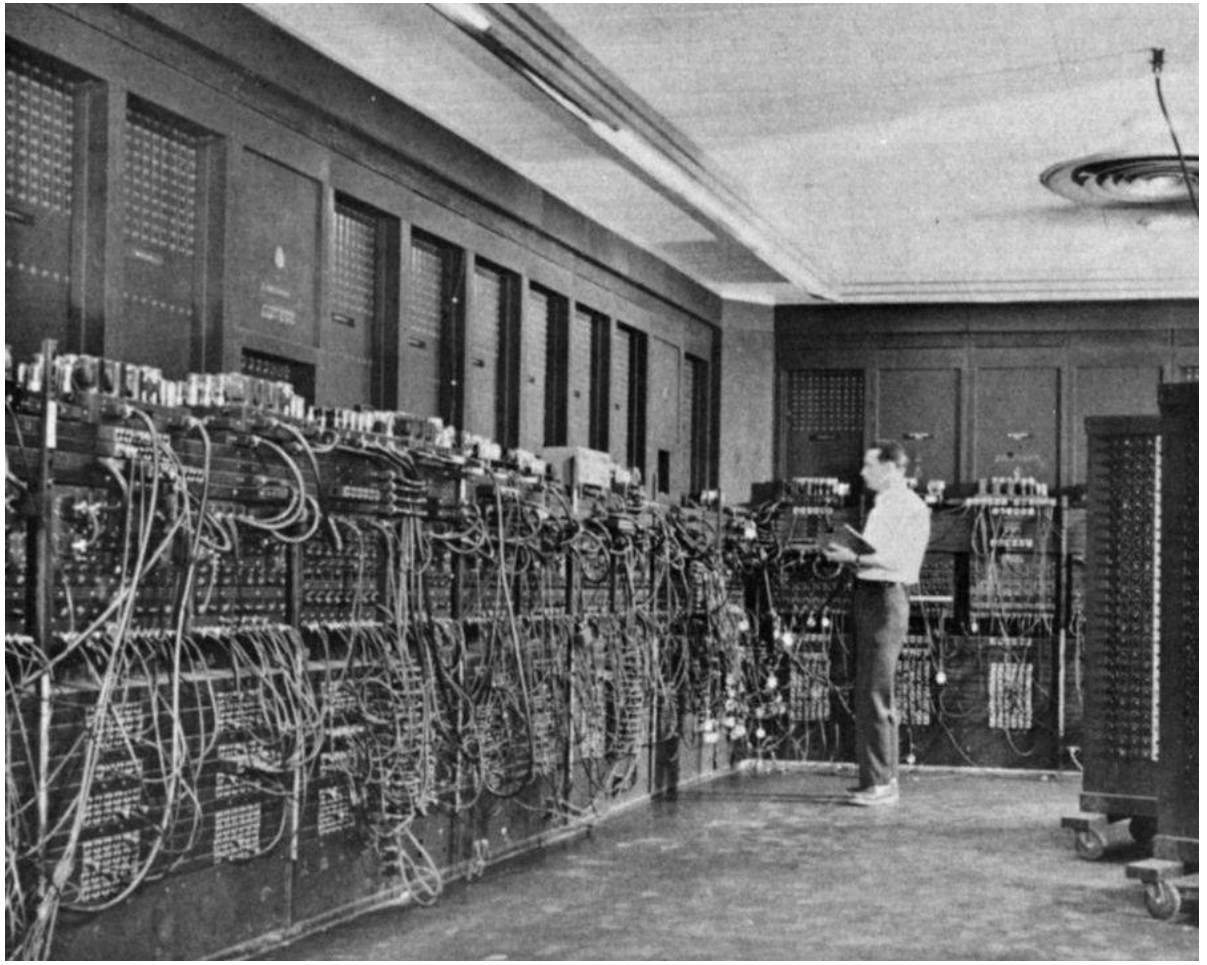
2. Betriebssystem im Überblick

3. Aufgaben des Betriebssystems

Anwendungsprogrammierung ohne Betriebssystem

Herausforderungen:

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



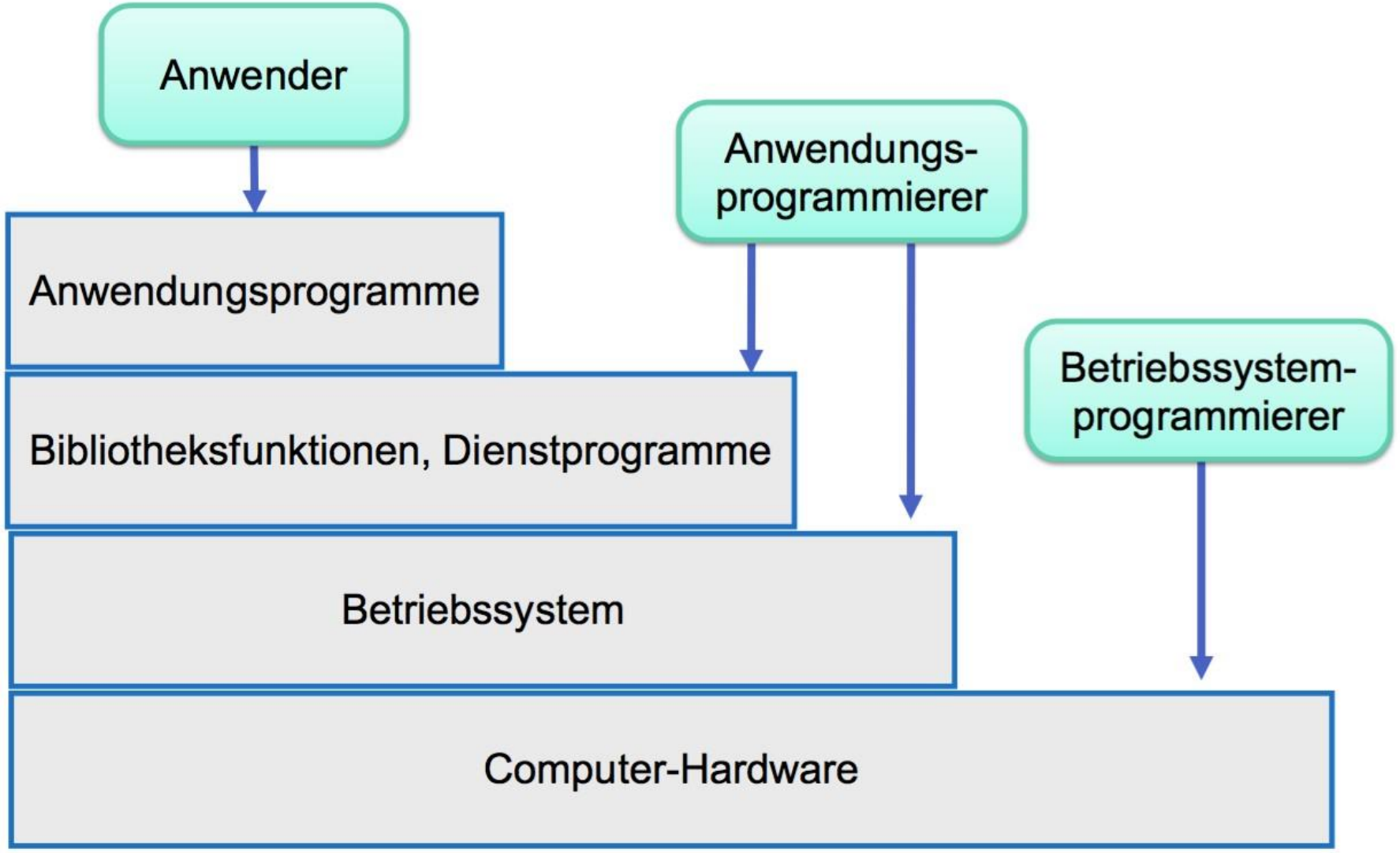
https://www.heise.de/imgs/18/1/3/8/4/9/8/5/Eniac-a38a5224c5b45592.jpeg

Ziele und Aufgaben von Betriebssystemen

Ein Betriebssystem ist ein Computerprogramm (Software) mit folgenden grundsätzlichen Aufgaben:

- Bereitstellung einer **Ausführungsumgebung** für Anwendungsprogramme und -prozesse
- **Abstraktionsschicht** und **einheitliche** Schnittstellen für Programmierer zur Nutzung der Betriebsmittel („Erweiterte Maschine“)
- **Effiziente Verwaltung, Virtualisierung** und **Schutz** der Betriebsmittel

Das Betriebssystem abstrahiert von der Hardware; es stellt Softwareentwicklern eine „erweiterte Maschine“ zur Verfügung.

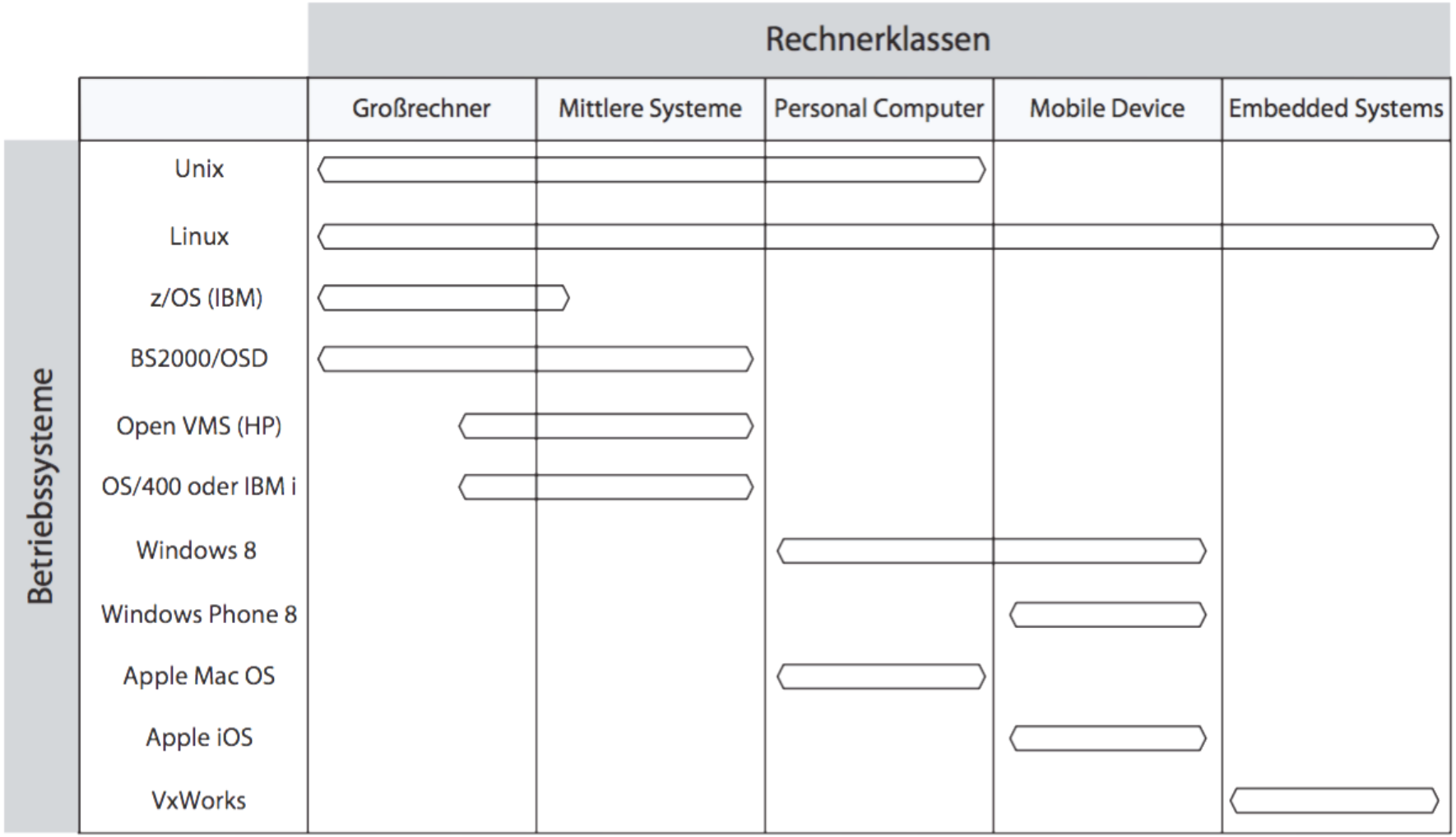


Betriebssystem

Das **Betriebssystem** (engl.: operating system) umfasst die Gesamtheit der Systemprogramme eines Rechners, die zusammen mit den Eigenschaften der Rechnerhardware die grundlegende Infrastruktur für die Ausführung von Anwendungssoftware bilden. Das Betriebssystem bildet eine Abstraktion von Hardwareeigenschaften und ist für die Steuerung und Überwachung von Anwendungsprogrammen zuständig. Es besteht aus einem **Kern** (engl.: kernel) und aus **Dienstprogrammen** (engl.: utility program).

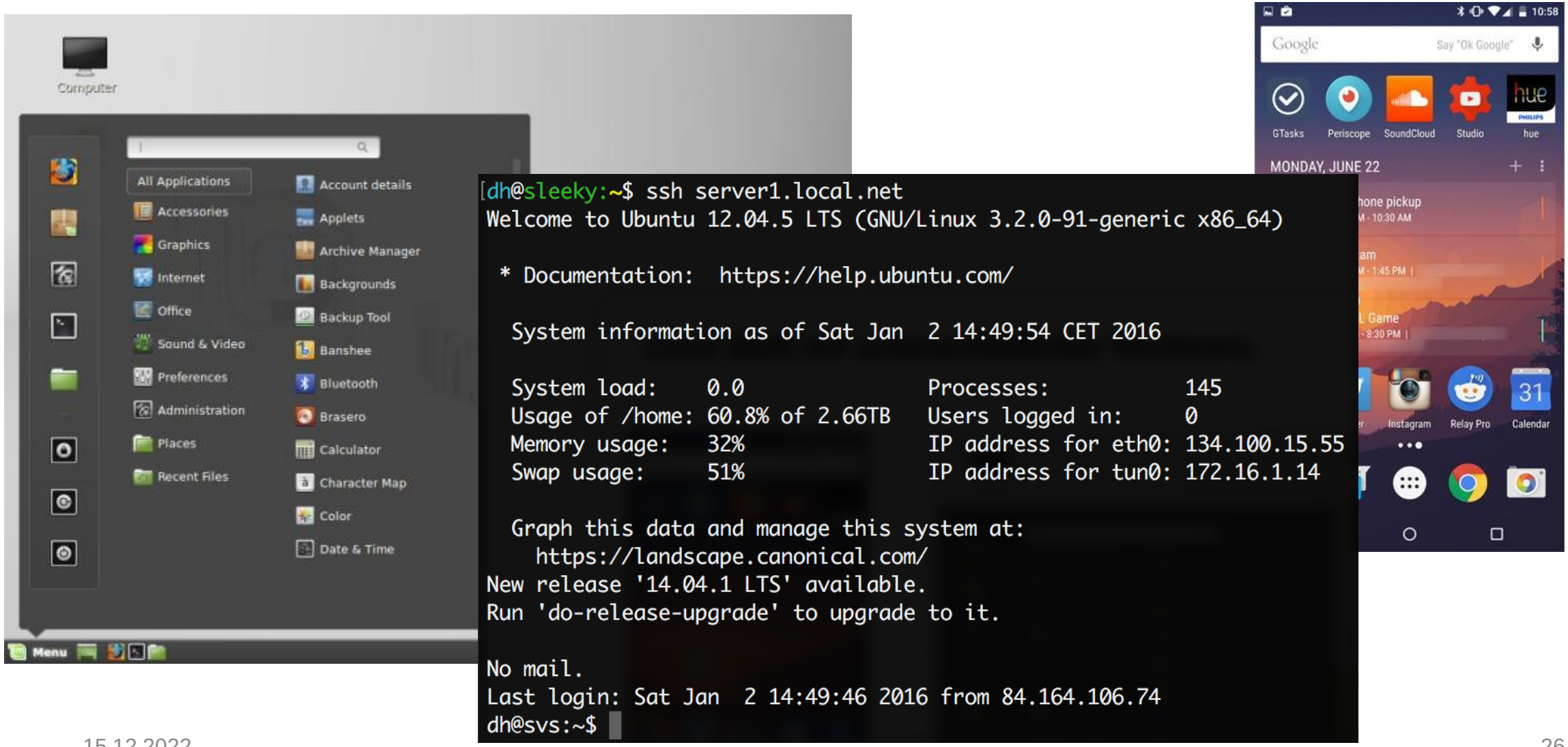
Hansen, S. 539.

Es gibt spezialisierte Betriebssysteme und solche, die auf allen Rechnerklassen eingesetzt werden können.

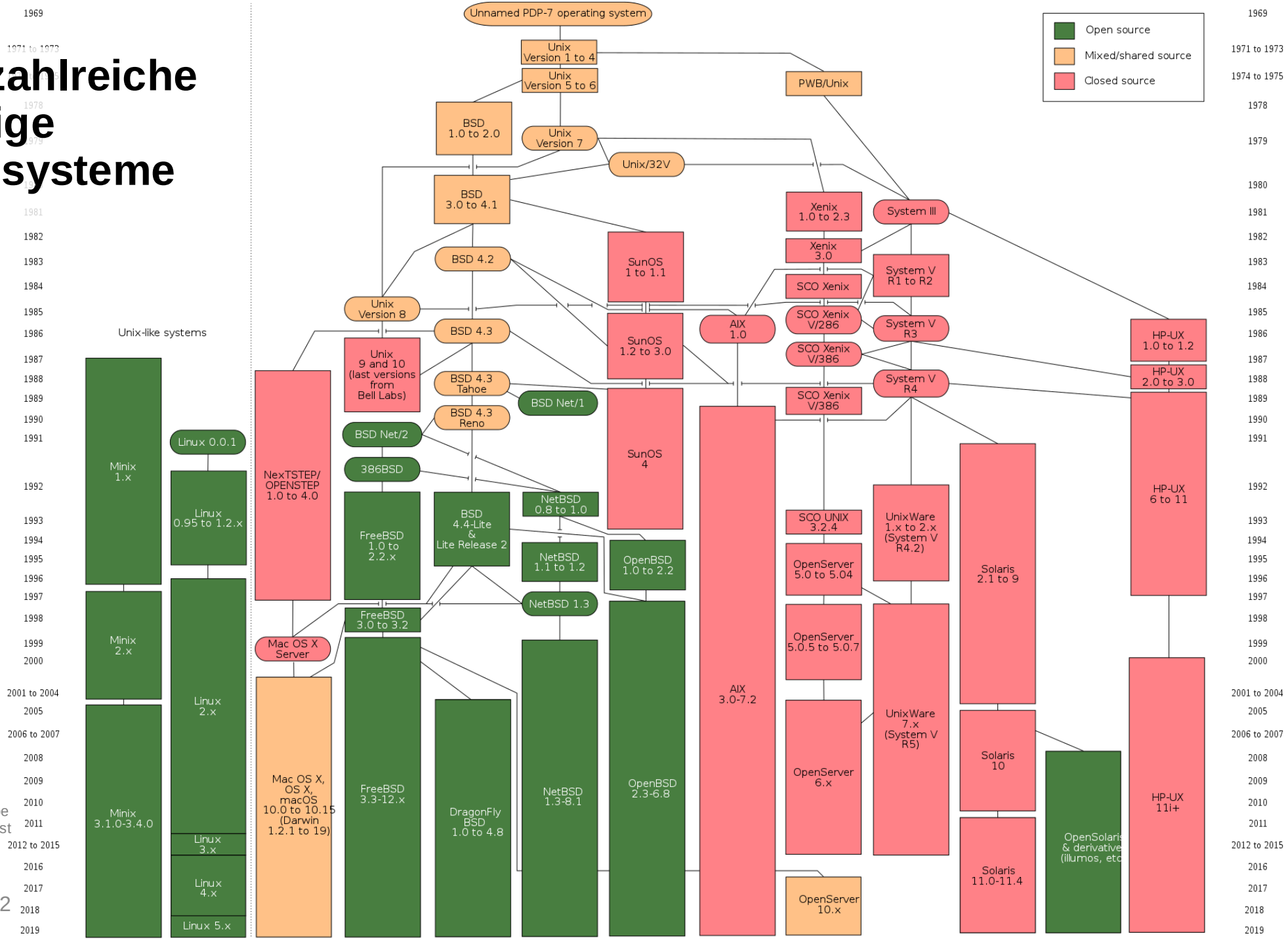


Leimeister, S. 74

Heute sind v.a. Unix-artige Betriebssysteme verbreitet, u.a. weil 1991 Linux als Open-Source-Software veröffentlicht worden ist.



Es gibt zahlreiche Unix-artige Betriebssysteme



https://upload.wikimedia.org/wikipedia/commons/thumb/7/77/Unix_history-simple.svg/1920px-Unix_history-simple.svg.png

Linux gilt als größter Erfolg der Open-Source-Software-Bewegung.

Open Source: Quellcode wird unter einer Lizenz öffentlich zur Verfügung gestellt, die das Ansehen, Verändern und Verbreiten erlaubt.



Free Software: “Free as in free speech”, d.h. ohne Restriktionen.

Weitere Beispiele: Firefox, MySQL, LibreOffice, GIMP, Java, Python, …



Open-Source-Lizenzen sind entweder “permissive” (MIT License, BSD License) oder “copyleft” (z.B. GNU General Public License), d.h. Derivate müssen unter derselben Lizenz stehen





Die OSS-Bewegung will die Einschränkungen kommerzieller Software vermeiden.

Ausgewählte Prinzipien aus dem Essay **“The Cathedral and the Bazaar”:**

Good programmers know what to write. Great ones know what to rewrite (and reuse).

When you lose interest in a program, your last duty to it is to hand it off to a competent successor.

Release early. Release often. And listen to your customers.

Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone.

Vorteile:

- Benutzer hat Zugriff auf Quellcode und kann ihn an seine Bedürfnisse anpassen
- Sicherheitskritische Funktionen können von jedem begutachtet werden
- Fehler werden schneller aufgedeckt
- Die Pflege hängt nicht von einem einzelnen Unternehmen ab

Nachteil:

- Es gibt keine Garantie für Sicherheit
- Niemand garantiert für Funktionalität

http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/index.html

Gliederung

1. Arten von Software

2. Betriebssystem im Überblick

3. Aufgaben des Betriebssystems

Das Betriebssystem sorgt dafür, dass Aufträge in Form von Programmen möglichst effizient abgearbeitet werden.

Programm: ausführbarer Code (inaktiv).

Task: ein ablaufendes Programm im Arbeitsspeicher eines Rechners.

Prozess: eine Task, deren Ablauflogik und Daten im Arbeitsspeicher als eine Einheit aufgefasst und geschützt werden.

Thread: Teil einer Task, der parallel verarbeitet wird; die Threads haben Zugriff auf einen gemeinsamen Datenbereich im Arbeitsspeicher.

Einprogrammbetrieb: Tasks werden nacheinander bearbeitet. Es ist jeweils nur eine Task im Arbeitsspeicher, die über alle Betriebsmittel verfügen kann.

Mehrprogrammbetrieb: (multi-tasking) mehrere Tasks werden parallel bearbeitet und sind gleichzeitig ganz oder teilweise im Arbeitsspeicher. Das Betriebssystem teilt den Tasks während der Ausführung die benötigten Betriebsmittel abwechselnd zu.

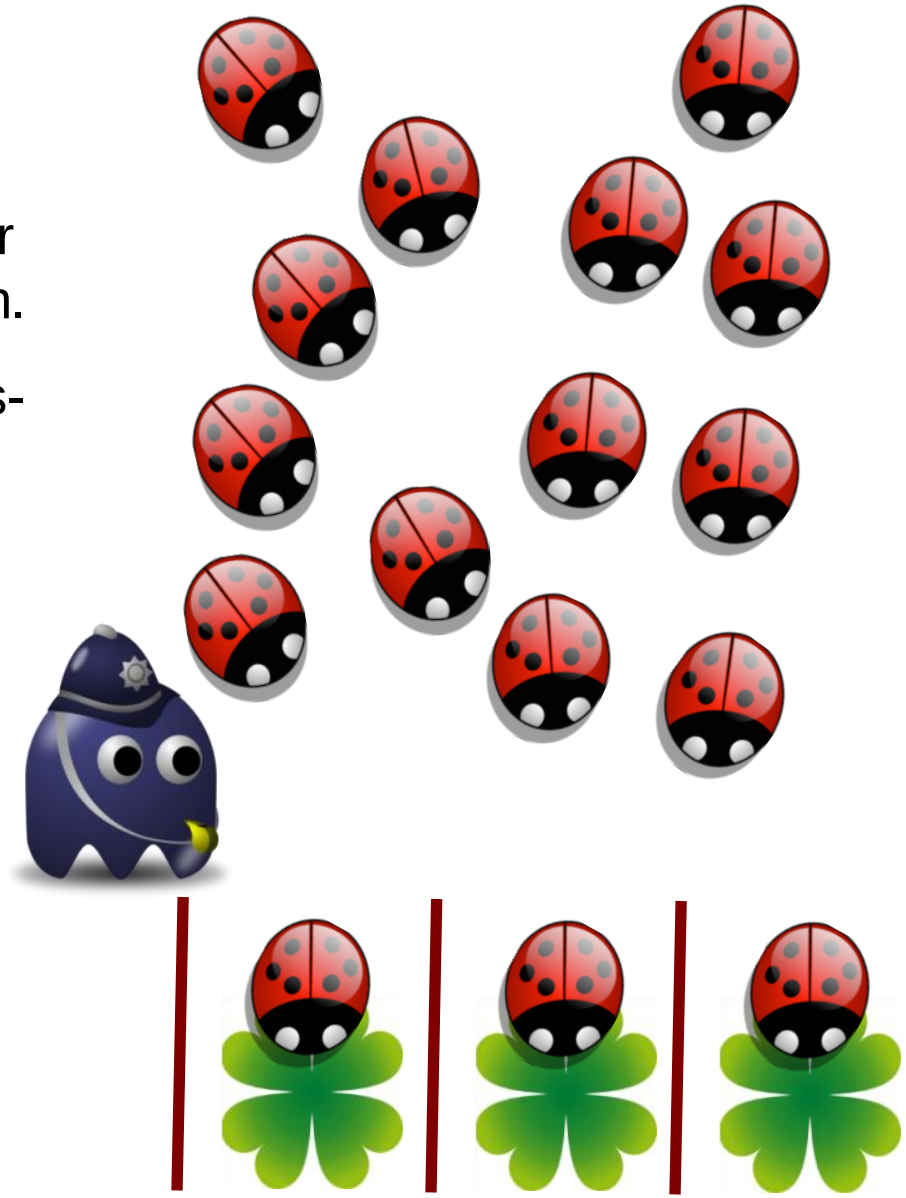
Mehrprogrammfähigkeit

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/Ausgabe-operationen durch Rechenzeit anderer Jobs nutzen.



Verwaltung paralleler Prozesse



Was ist nochmal ein Prozess?

- Programm in der Ausführung
- Einem Prozess ist ein bestimmter Adressraum zugeordnet
- Benutzt gemeinsame Ressourcen wie CPU, RAM, Festplatte

Problem:

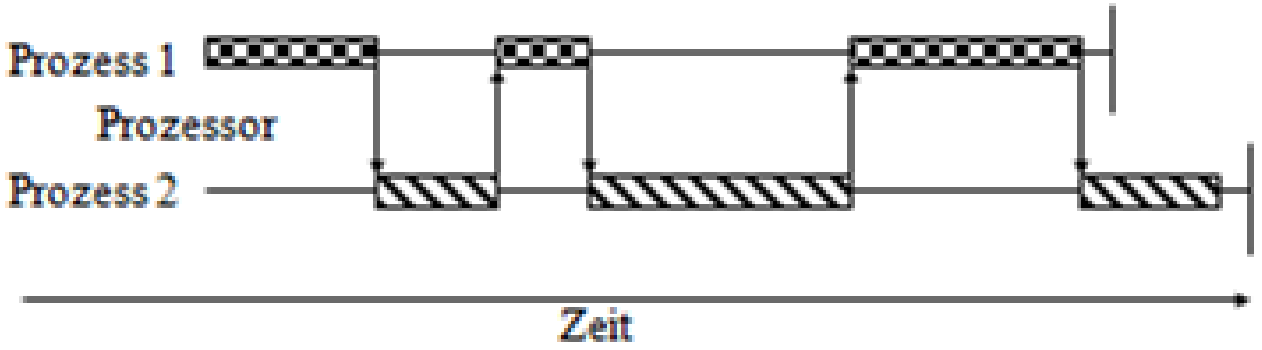
- Mehrere Prozesse benötigen „gleichzeitig“ Rechenkapazitäten
- Prozesse haben unterschiedliche Laufzeit
- Eine CPU kann nur einen Prozess gleichzeitig verarbeiten



Ausführung mehrerer Prozesse durch einen Rechner (Multitasking)

Multitasking

- Ausführung mehrerer Prozesse gleichzeitig
- durch Zuteilung (Scheduling) von Prozessorzeit an jedem Prozess



Kooperatives Multitasking

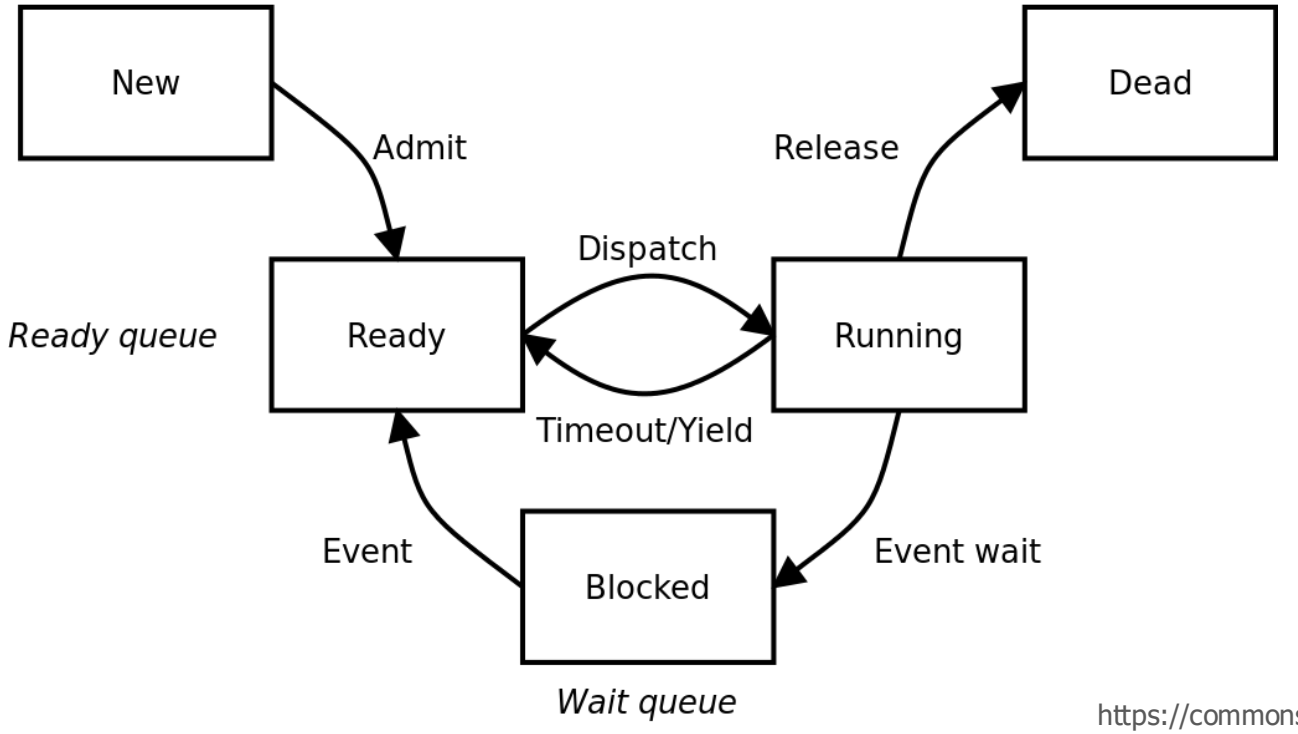
- jeder Prozess entscheidet selbst
- Ein Prozess kann das gesamte System blockieren, wenn er nicht mehr ins Betriebssystem zurückkehrt!

Präemptives Multitasking

- zentraler Dispatcher entscheidet über die Prozessorzuteilung

Prozesse durchlaufen verschiedene Zustände. Der Dispatcher steuert, wann welcher Prozess ausgeführt wird.

Beim sog. **präemptiven Multitasking** wird jeder Prozess vom Dispatcher automatisch nach einer festgelegten Zeit im Running-Zustand unterbrochen, sodass andere Prozesse, die gerade „ready“ sind, zum Zug kommen. Muss ein Prozess auf I/O-Operationen (z.B. Daten von der Festplatte lesen) warten, ist er „blocked“ und wird ebenfalls unterbrochen.



https://commons.wikimedia.org/wiki/File%3AProcess_state.svg

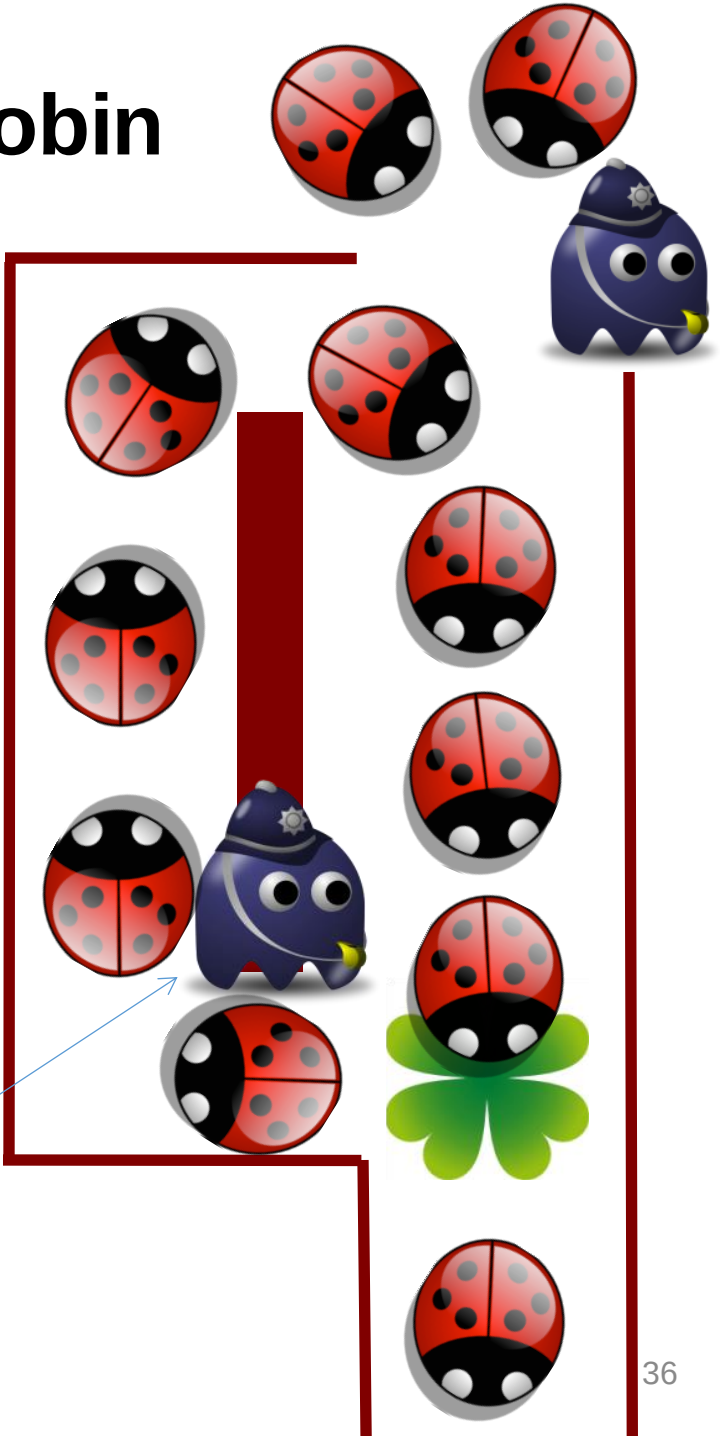
Präemptives Multitasking – Round Robin

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

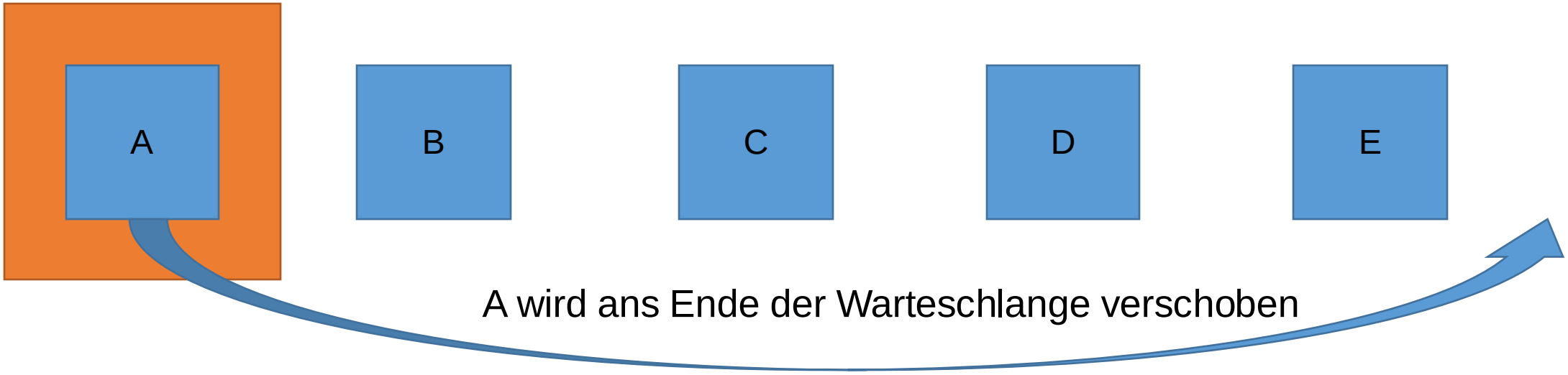
- Nutzungsdauer /-zeitpunkt
- Prozesse in Warteschlange
- Priorität des Prozesses



Präemptives Multitasking – Round Robin

Annahmen

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

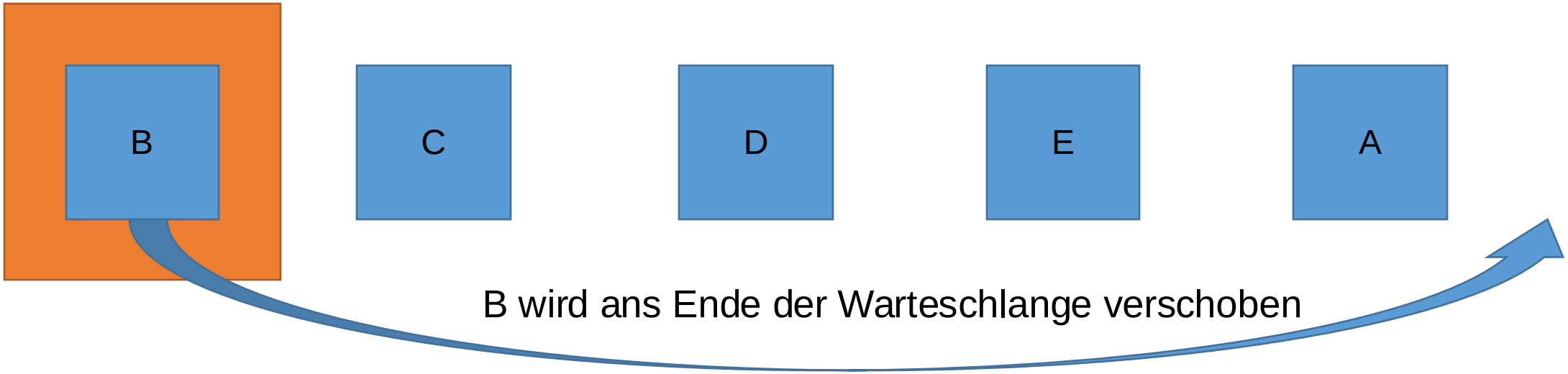


Die Ausführung von Anwendung A wird nach 10ms unterbrochen

Präemptives Multitasking – Round Robin

Annahmen

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

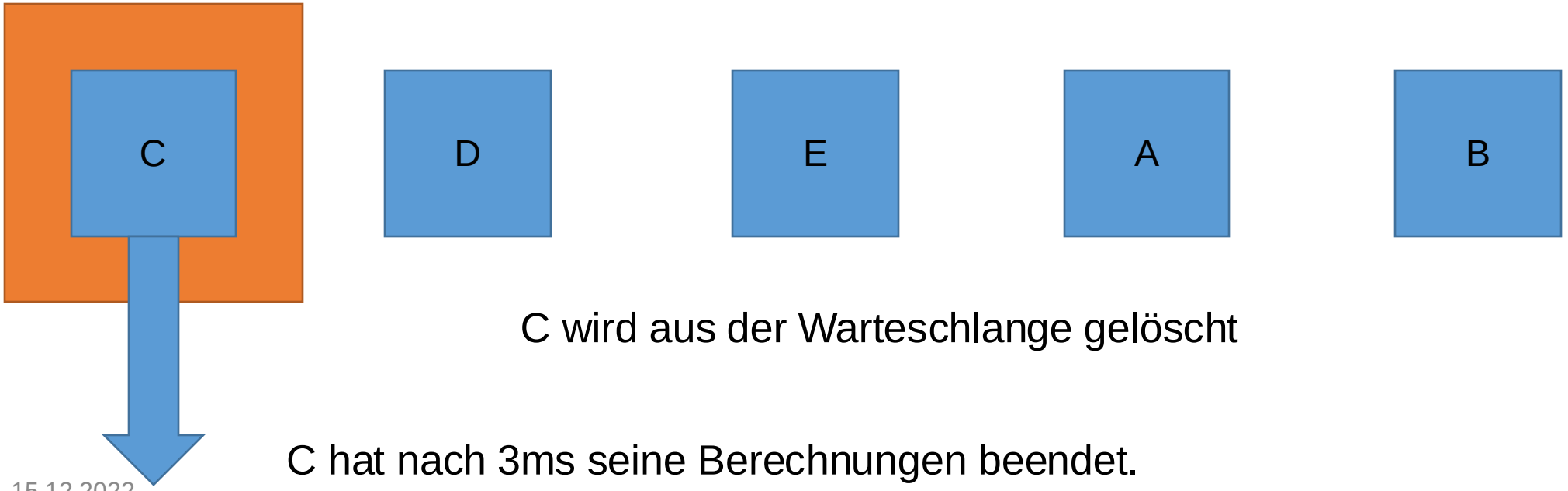


Die Ausführung von Anwendung B wird nach 10ms unterbrochen

Präemptives Multitasking – Round Robin

Annahmen

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



C wird aus der Warteschlange gelöscht

C hat nach 3ms seine Berechnungen beendet.

Prozess-Synchronisation

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

Prozess-Synchronisation: Beispiel

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

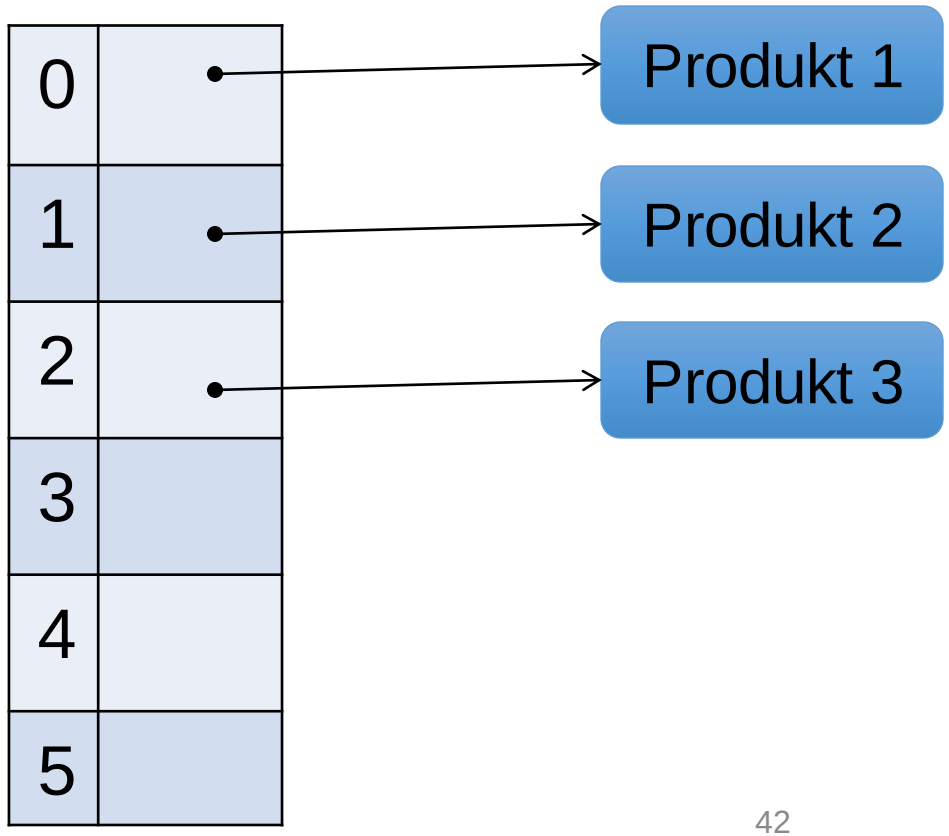
```
1: while (true){
2: while (anzahl < 5){
3: neuesProdukt = new Produkt();
4: lager[anzahl] = neuesProdukt;
5: anzahl++;
6: }
7: }
```

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



Prozess-Synchronisation: Ablauf

Erzeuger

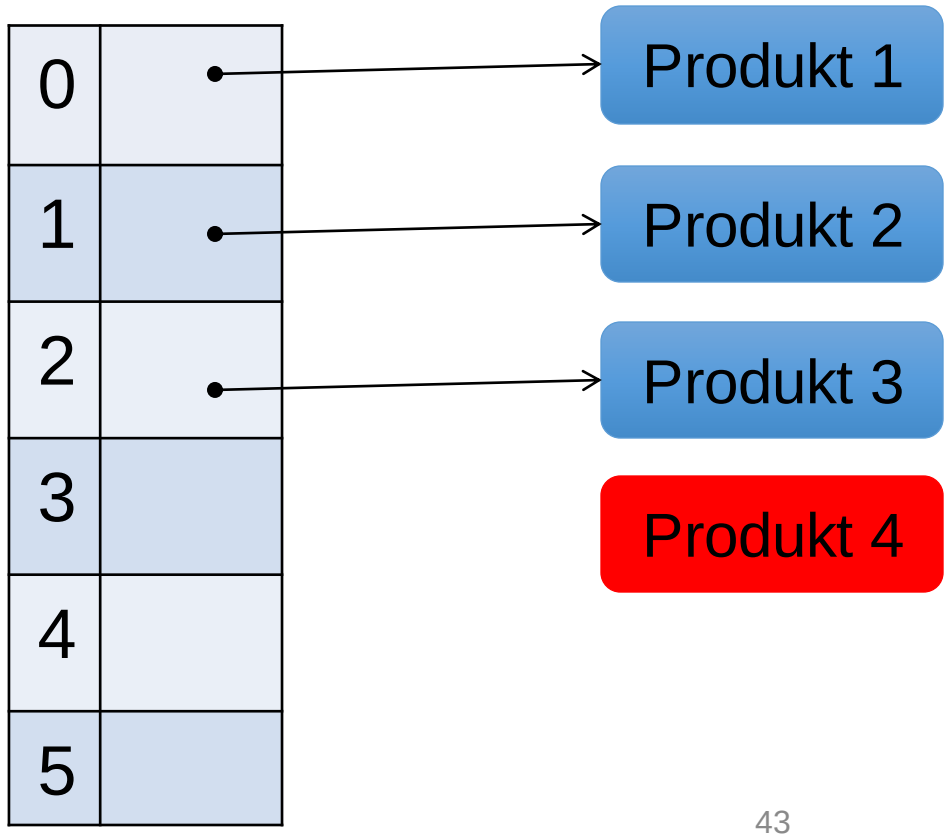
```
1: while (true){
2: while (anzahl < 5){
3: neuesProdukt = new Produkt();
4: lager[anzahl] = neuesProdukt;
5: anzahl++;
6: }
7: }
```

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



Prozess-Synchronisation: Ablauf

Erzeuger

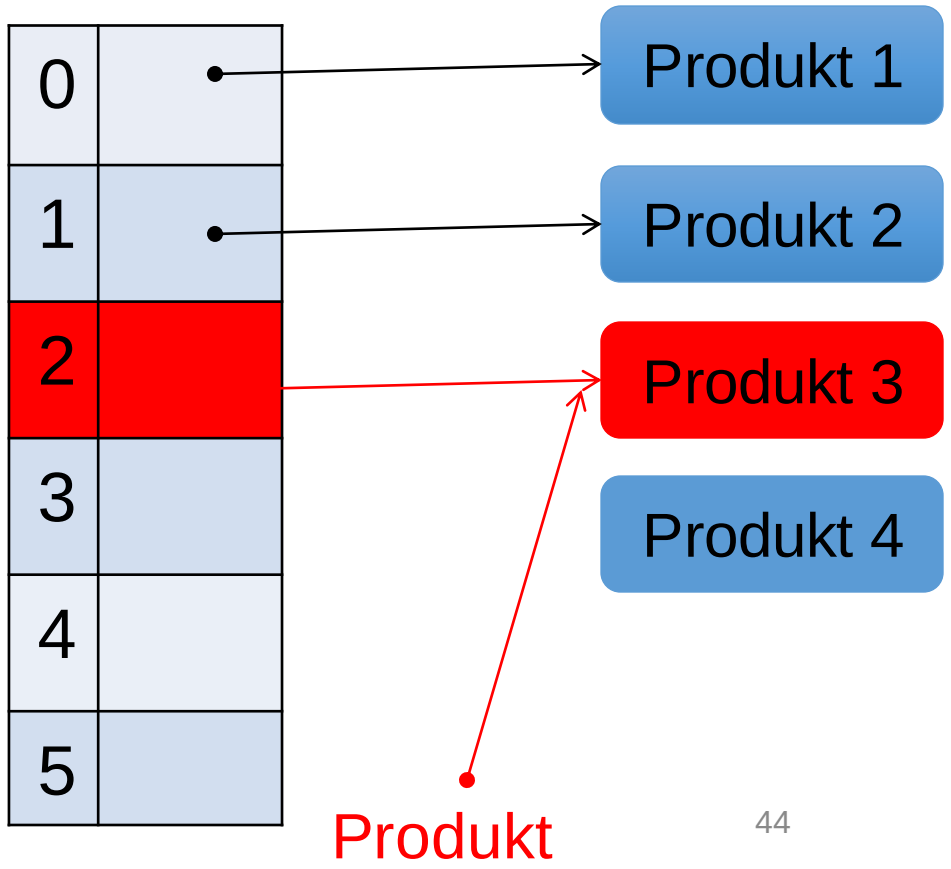
```
1: while (true){
2: while (anzahl < 5){
3: neuesProdukt = new Produkt();
4: lager[anzahl] = neuesProdukt;
5: anzahl++;
6: }
7: }
```

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



Prozess-Synchronisation: Ablauf

Erzeuger

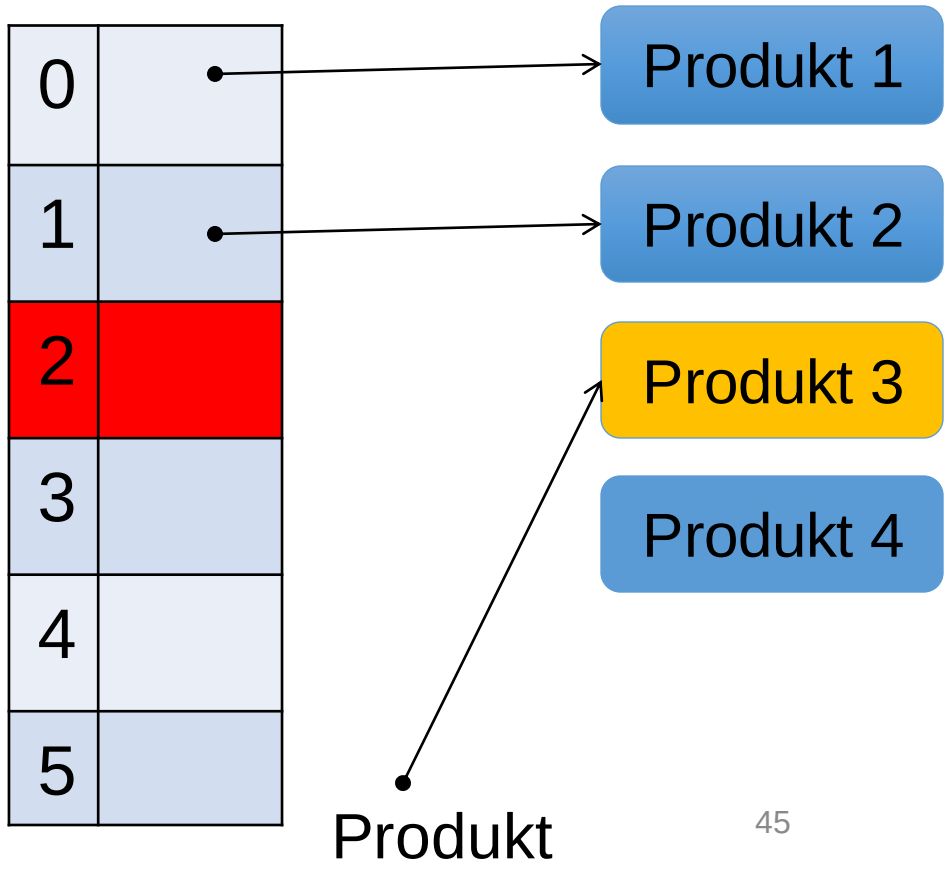
```
1: while (true){
2: while (anzahl < 5){
3: neuesProdukt = new Produkt();
4: lager[anzahl] = neuesProdukt;
5: anzahl++;
6: }
7: }
```

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



Prozess-Synchronisation: Ablauf

Erzeuger

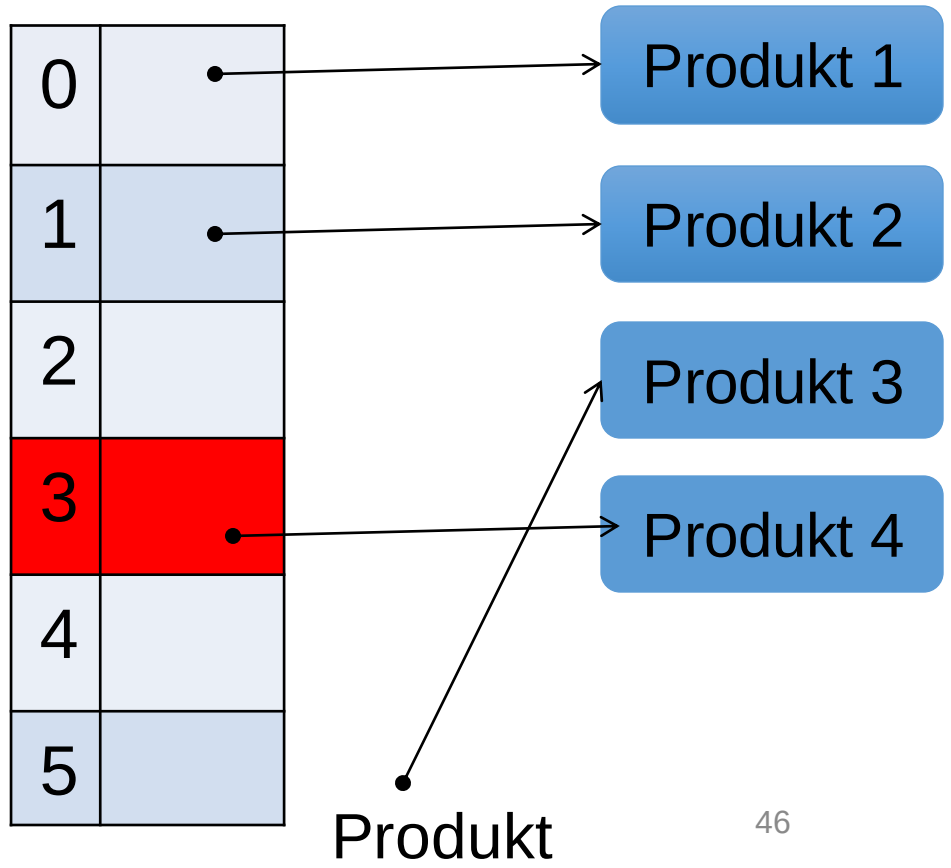
```
1: while (true){
2: while (anzahl < 5){
3: neuesProdukt = new Produkt();
4: lager[anzahl] = neuesProdukt;
5: anzahl++;
6: }
7: }
```

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



Prozess-Synchronisation: Ablauf

Erzeuger

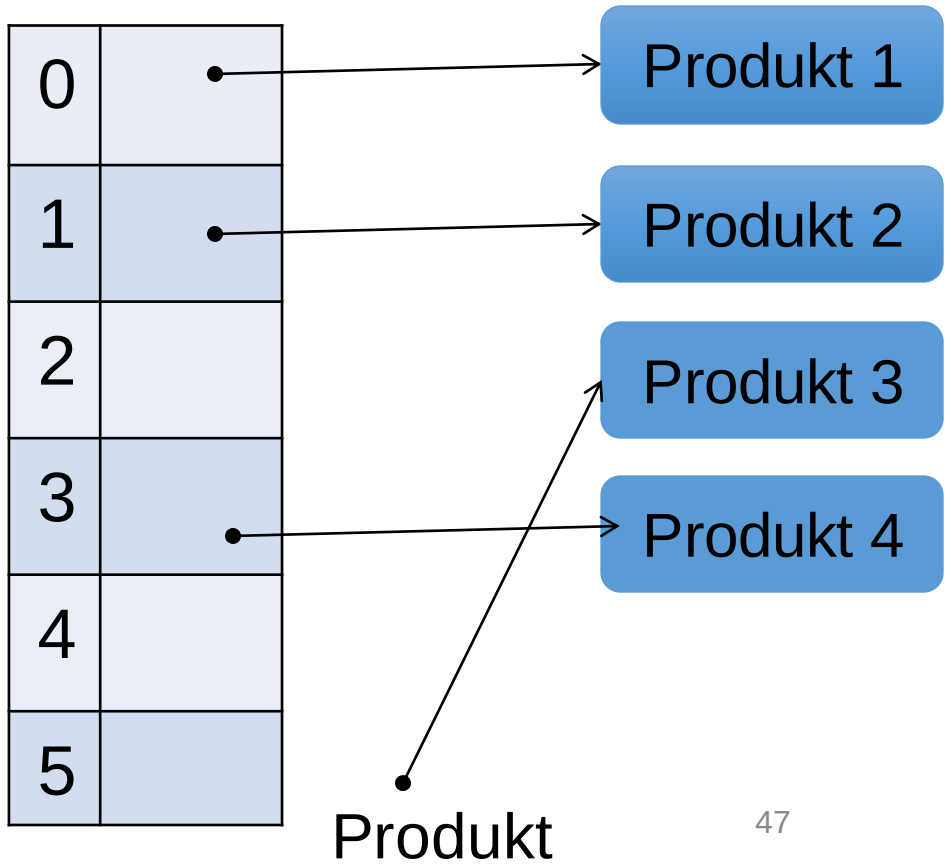
```
1: while (true){
2: while (anzahl < 5){
3: neuesProdukt = new Produkt();
4: lager[anzahl] = neuesProdukt;
5: anzahl++;
6: }
7: }
```

Verbraucher

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

anzahl = **4**

Lager



Prozess-Synchronisation: Ablauf

Erzeuger

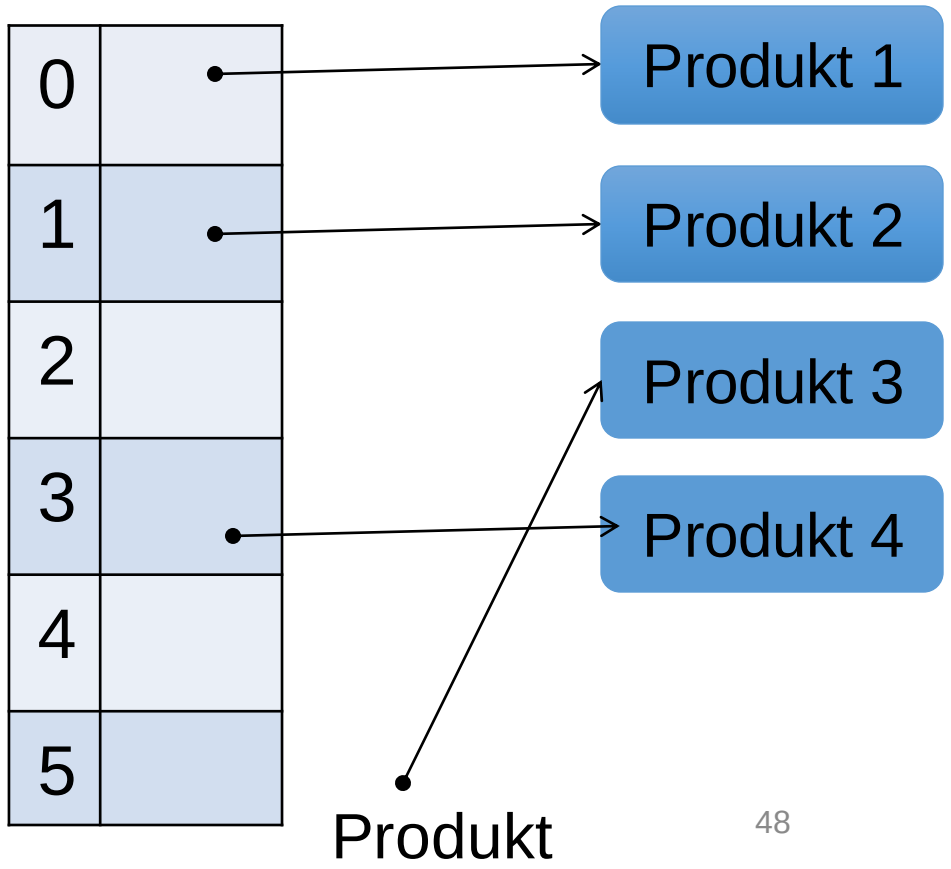
```
1: while (true){
2: while (anzahl < 5){
3: neuesProdukt = new Produkt();
4: lager[anzahl] = neuesProdukt;
5: anzahl++;
6: }
7: }
```

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



Prozess-Synchronisation: Lösungsansätze

Erzeuger

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

kritischer Bereich

Verbraucher

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

kritischer Bereich

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

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

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

Idee der Sperre

Erzeuger

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

Verbraucher

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

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

Deadlocks / Verklemmungen

Grundlegendes Problem:

- Zwei unterschiedliche Prozesse benötigen mehrere Betriebsmittel
- Das benötigte Betriebsmittel ist dem jeweils anderen Prozess zugeordnet

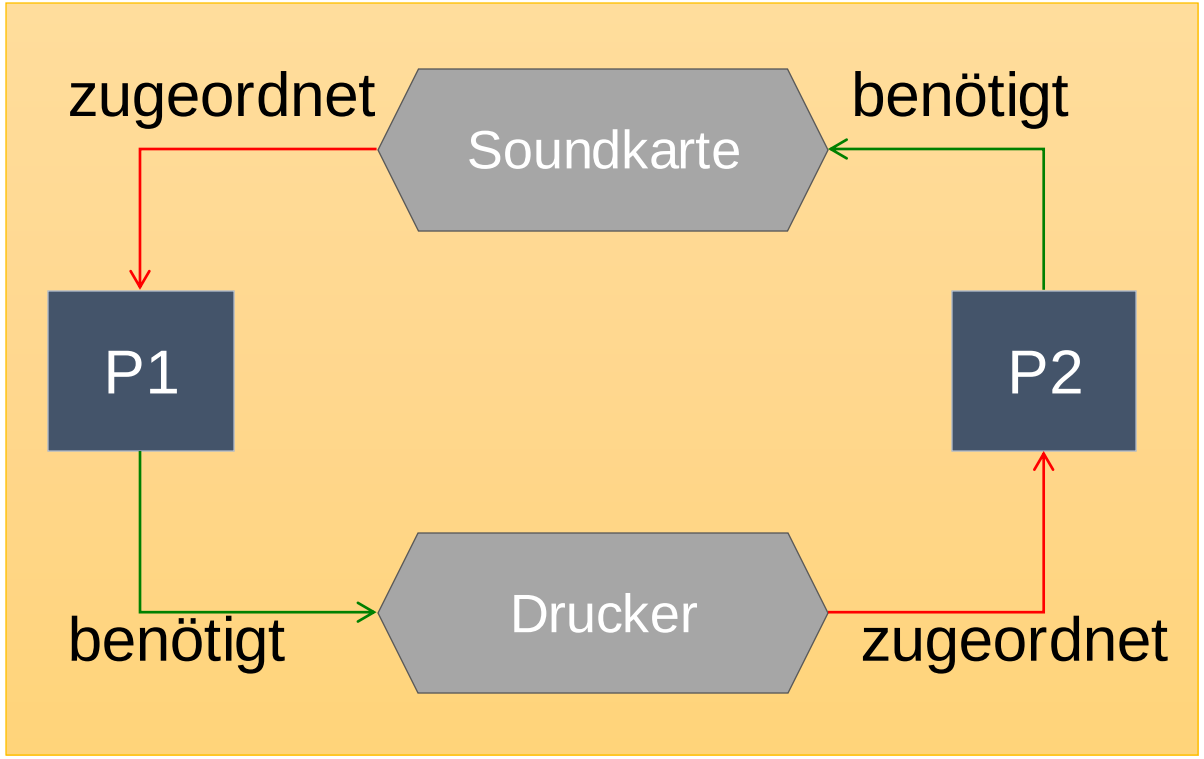
Deadlocks / Verklemmungen

P1 und P2 benötigen beide gleichzeitig 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



Mehr dazu im MA-Modul „Betriebssysteme“

Nächste Termine

Nächste Vorlesung:

15.01.2025