

# MAS: Betriebssysteme

## Interruptverarbeitung in Betriebssystemen

T. Pospíšek

# Gesamtüberblick

---

1. Einführung in Computersysteme
2. Entwicklung von Betriebssystemen
3. Architekturansätze
- 4. Interruptverarbeitung in Betriebssystemen**
5. Prozesse und Threads
6. CPU-Scheduling
7. Synchronisation und Kommunikation
8. Speicherverwaltung
9. Geräte- und Dateiverwaltung
10. Betriebssystemvirtualisierung

## Zielsetzung

---

- Konzepte und Abläufe der Interruptverarbeitung in Betriebssystemen kennenlernen und verstehen
- Unterscheidung der verschiedenen Interrupt-Klassen vornehmen können
- Ablauf eines asynchronen Interrupts erläutern können
- Ablauf eines (synchronen) Systemcalls erläutern können

# Überblick

---

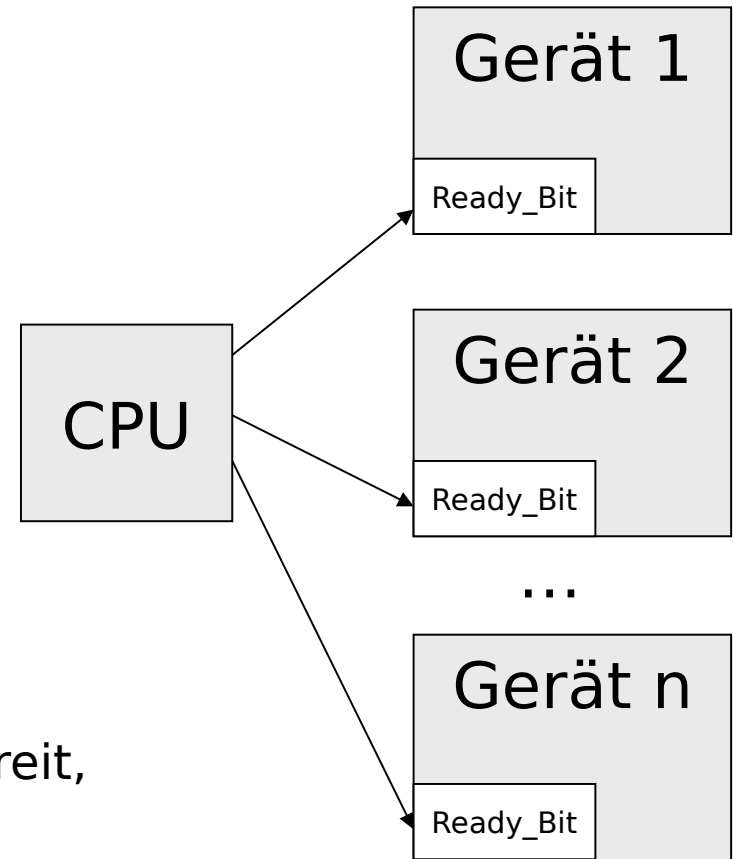
- 1. Begriffe und Klassifizierung**
2. Zusammenspiel der Komponenten
3. Systemdienste und Systemcalls

# Polling

## ■ Polling => Busy Waiting (aktives Warten)

```
while() {  
    for (alle Geräte) {  
        if (Gerät[i].Ready_Bit == 1 ) {  
            doAction();  
        }  
    }  
}
```

Ready\_Bit = 1 bedeutet, Gerät ist bereit,  
es sind z.B. Daten abzuholen



# Polling

---

- **Polling ist eine problematische Technik**
- **„Code Smell“**
- **Verallgemeinerung des Problems:**
  - Kooperation und Synchronisation
  - Pull vs Push

# Interrupt

- **Interrupt** = Unterbrechung = Trap
- Abgrenzung zu **Polling**
  - Kein aktives Abfragen von Ereignisquellen
- Gründe für Interrupts:
  - Betriebssystembedingungen
  - Asynchrone Ereignisse
- Verursacher: Hardware oder Software
- **Abschaltung** (Maskierung) von Interrupts ist möglich, sollte aber für sehr kurze Zeit sein
  - Manchmal unbedingt erforderlich (siehe später → Interrupt Service Routinen)

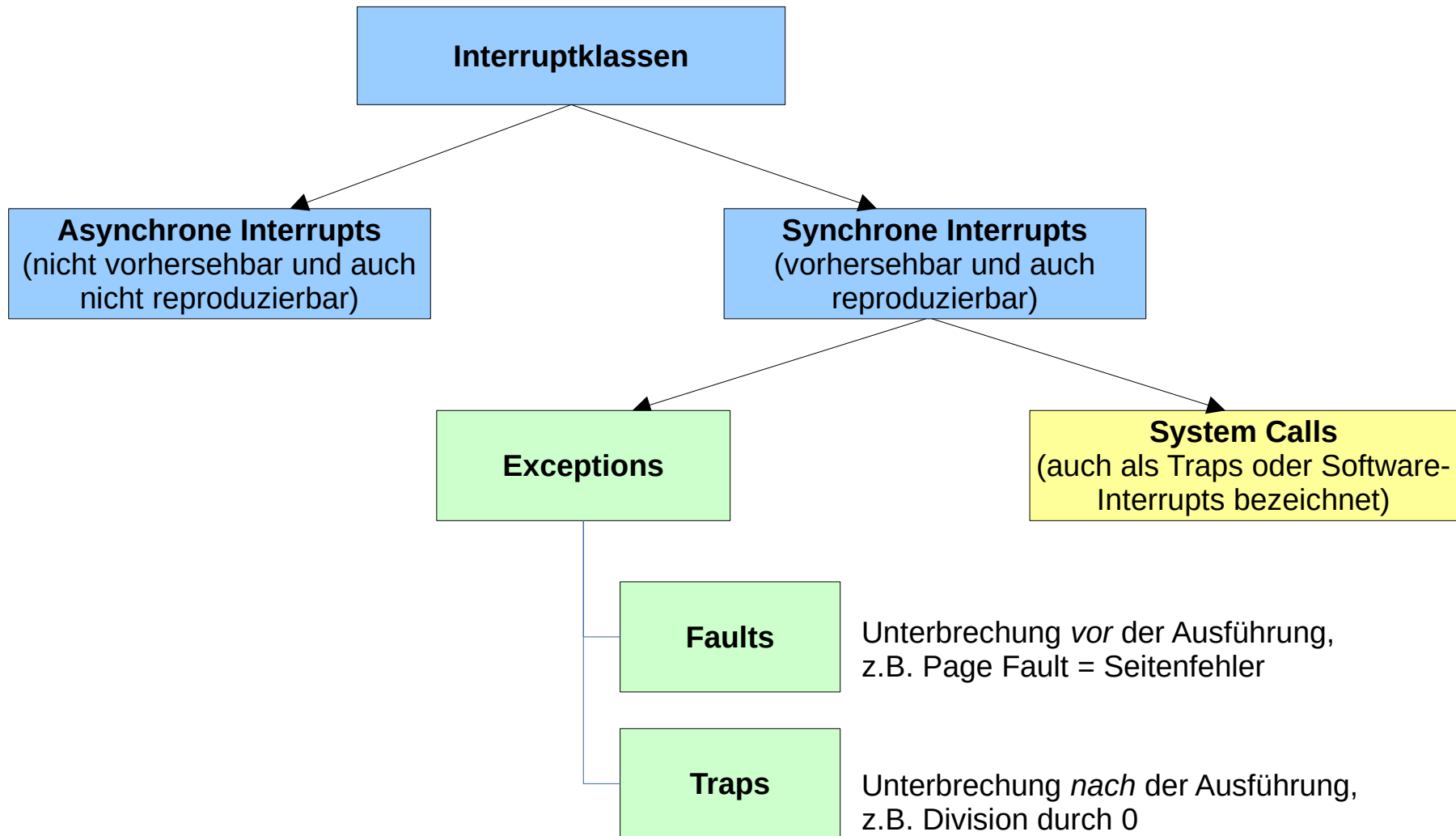
# Synchron und asynchron

---

- **Synchrone** Interrupts: Von der CPU ausgelöste Ausnahmen (für das laufende Programm gedacht)
  - Division durch 0
  - Speicherzugriffsverletzung
- **Asynchrone** Interrupts: Treten unabhängig davon auf, was das System gerade ausführt
  - Netzwerkadapter meldet ankommende Nachricht
  - Plattenspeicher meldet Zustellung eines Blocks



# Interrupt-Klassifizierung



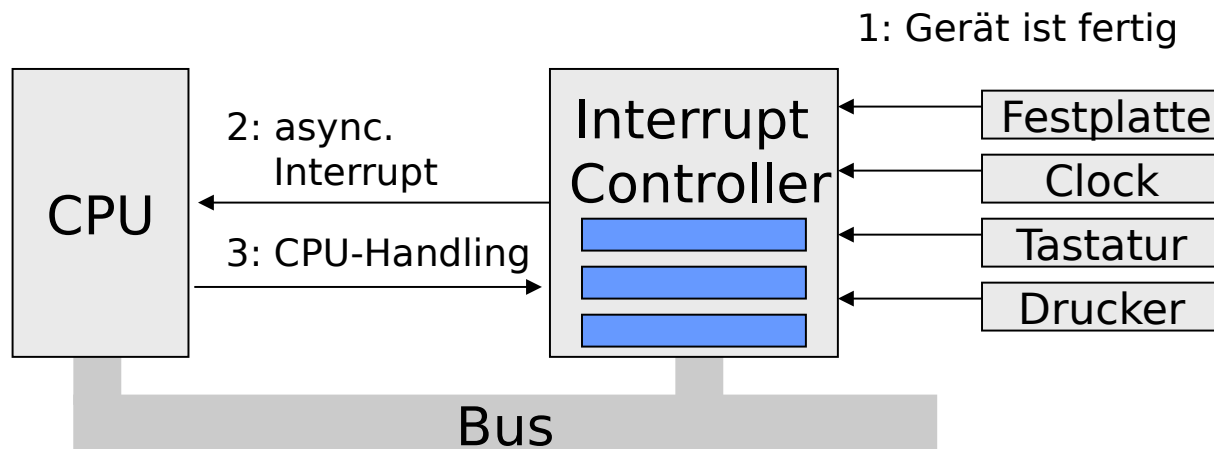
# Überblick

---

1. Begriffe und Klassifizierung
- 2. Zusammenspiel der Komponenten**
3. Systemdienste und Systemcalls

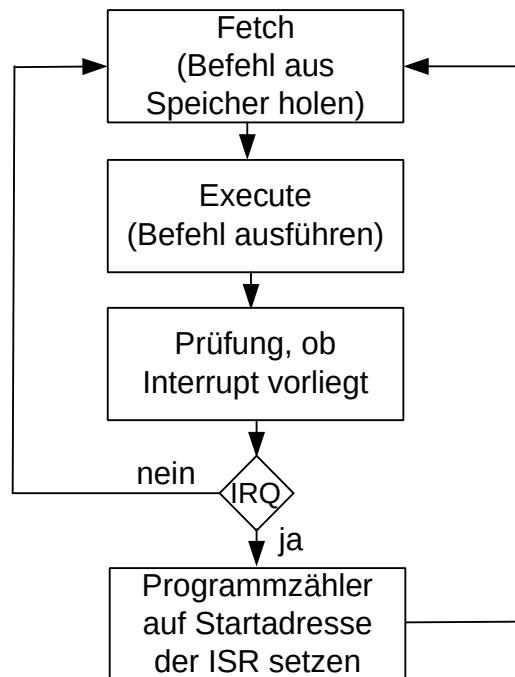
# Interrupt-Bearbeitung: Zusammenspiel

- Interrupts führen dazu, dass Code außerhalb des normalen Programmflusses ausgeführt wird
- Steuerung wird an eine definierte Position im Kernel übergeben  
→ **Interrupt-Service-Routine (ISR)**



# Interrupt-Bearbeitung: Befehlszyklus

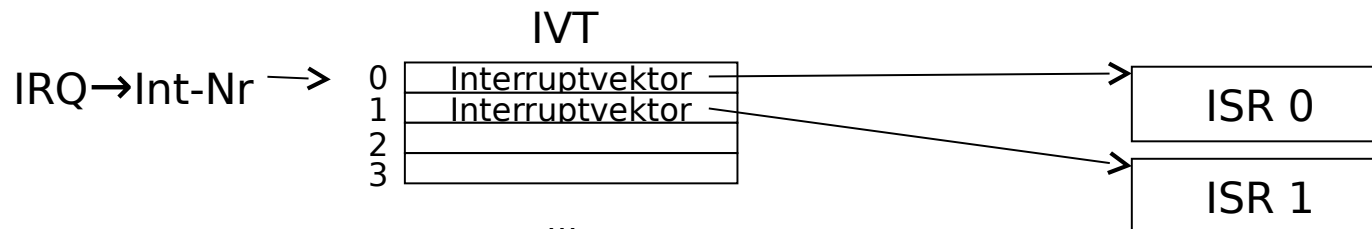
- Prüfung, ob Interrupt anliegt, ist Teil des CPU Befehlszyklus
- Prüfung am Ende der Ausführung eines Maschinenbefehls



- Bei Multiprozessoren bzw. Mehrkernprozessoren:
  - Dispatching eines Prozessors/Kerns notwendig, um anstehenden Interrupt zu bearbeiten

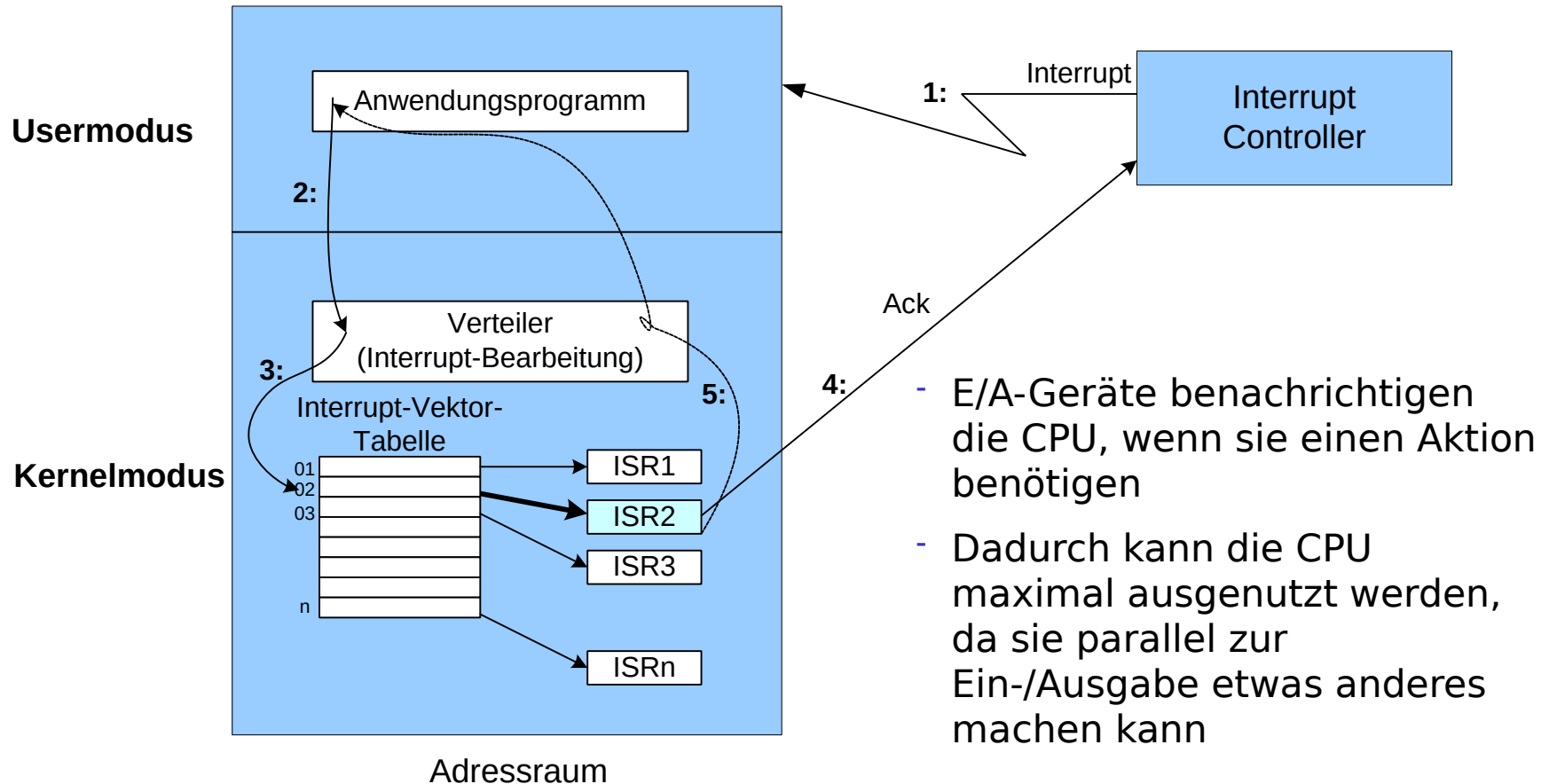
# Interrupt-Vektor-Tabelle und Adressierung

- Interrupt Request (IRQ) wird vom Gerät gesendet und identifiziert das Gerät
- Abbildung IRQ → Int-Nr durch Hardware in einem Interrupt Controller
  - Int-Nr ist der Index für die Interrupt-Vektor-Tabelle (IVT)
  - Die IVT wird von der CPU verwendet
- IVT-Aufbau durch Prozessor vorgegeben:
  - Bei Intel 256 IVT-Einträge für Exceptions, Systemcalls (Traps) und Geräteinterrupts



# Interrupt-Service-Routine (ISR)

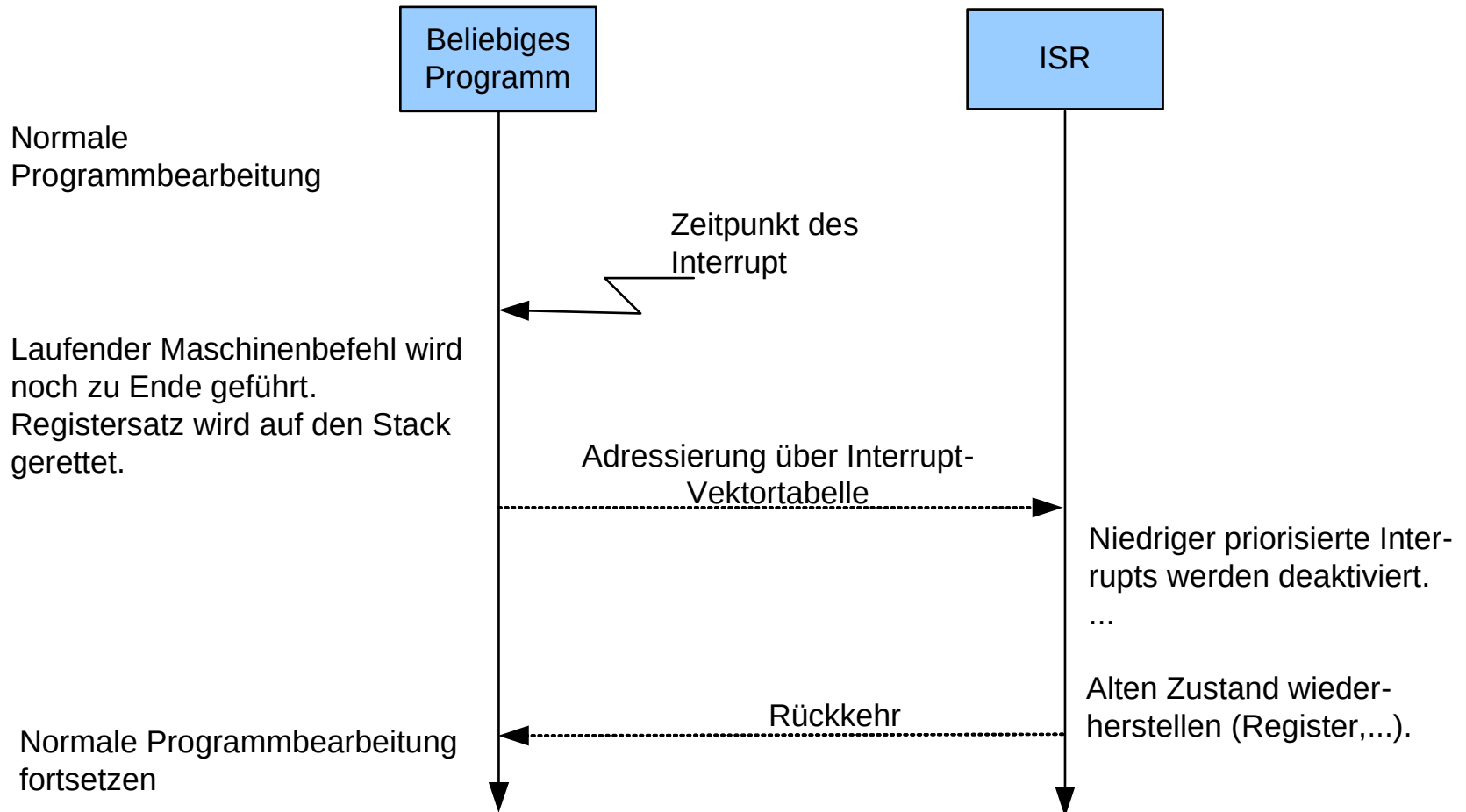
Schritte, welche CPU ausführt:



- E/A-Geräte benachrichtigen die CPU, wenn sie einen Aktion benötigen
- Dadurch kann die CPU maximal ausgenutzt werden, da sie parallel zur Ein-/Ausgabe etwas anderes machen kann

ISR = Interrupt Service Routine

# Interrupt-Bearbeitung: Ablauf



## Speichern des Zustands bei Interrupt

- damit Interrupt bearbeitet werden kann, ohne das ausgeführte Programm zu beeinflussen, muss dessen Zustand abgespeichert werden
- manche Prozessoren machen dies automatisch und haben dafür einen eigenen, speziellen Stack
- bei anderen Prozessoren muss der Interrupt-Handler den Zustand selbst abspeichern



# Interrupt Controller

- (A)PIC – (Advanced) Programmable Interrupt Controller = Multiplexer für Hardware-Interrupts
  - ursprünglich Intel 8259A PIC in IBM PC XT (1983)
  - Schaltkreis zur Verwaltung mehrerer Hardware-Interrupts
  - überwacht Interrupt-Leitungen (IRQ), speichert sie und gibt sie an die CPUs weiter
  - ist in modernen CPUs *integriert*



Bild: [https://en.wikipedia.org/wiki/File:Ibm\\_px\\_xt\\_color.jpg](https://en.wikipedia.org/wiki/File:Ibm_px_xt_color.jpg)

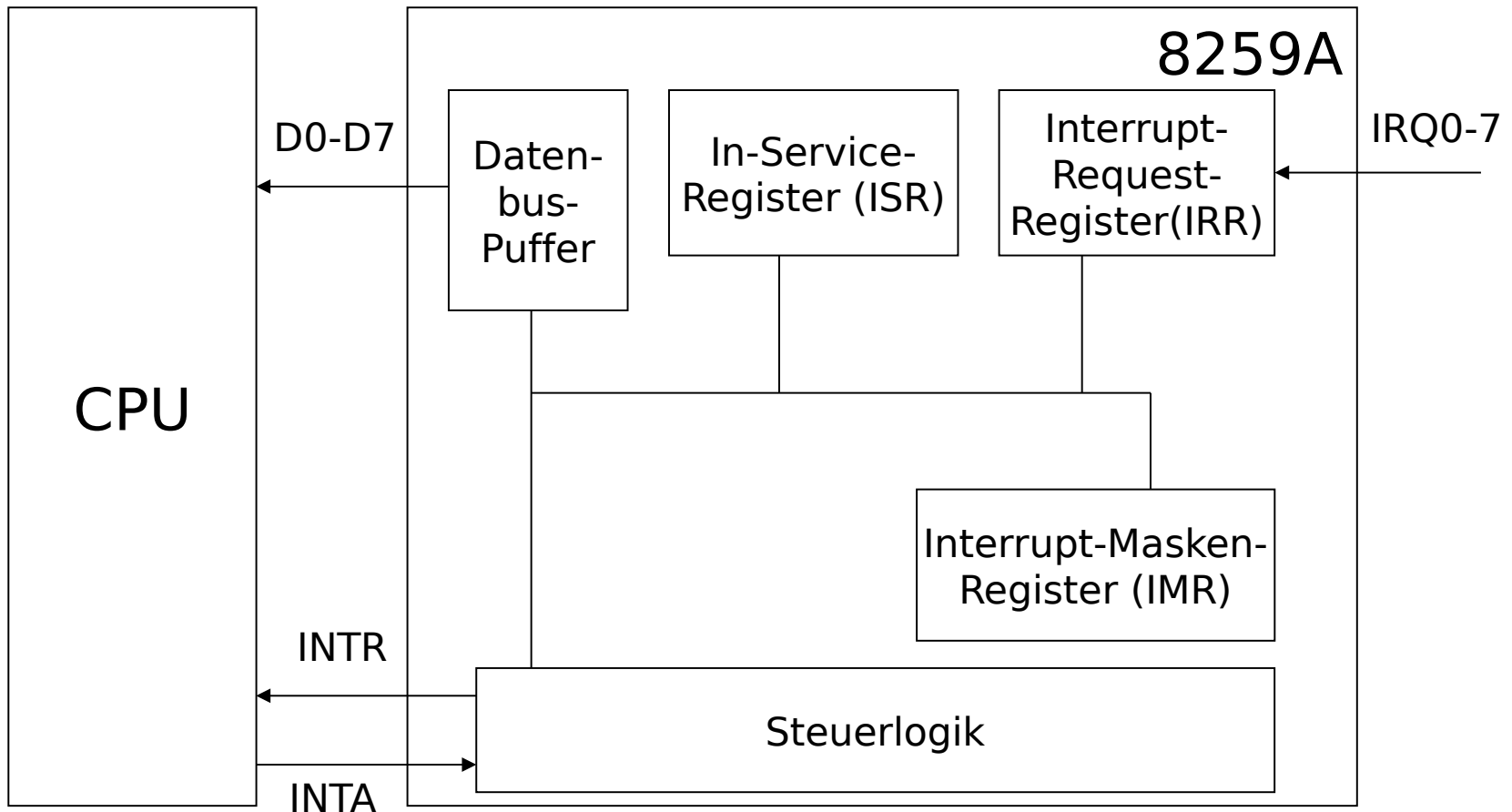
The diagram illustrates the interrupt architecture. On the left, a box labeled 'Betriebs-system' contains 'Interrupt-Handling' and 'CPU'. The 'CPU' box is connected to the 'Intel 8259 Master' box. The 'Intel 8259 Master' box contains an 'IMR' (Interrupt Mask Register) and is connected to the 'Intel 8259 Slave' box. The 'Intel 8259 Slave' box is connected to various devices: '08 real time clock', '09 s/w - int a0h', '10 reserved', '11 reserved', '12 PS/2 mouse', '13 coprocessor', '14 hard disk', '15 reserved', '03 serial 2', '04 serial 1', '05 parallel 2', '06 floppy', and '07 parallel 1'. The 'Intel 8259 Master' box is connected to the 'CPU' via 'D0 - D7' (Adressierung des Interrupts) and 'CPU INTR'. The 'Intel 8259 Slave' box is connected to the 'Intel 8259 Master' via 'IRQ 2'. The 'Intel 8259 Master' box is connected to the 'Interrupt-Handling' box via 'INTA'.

Übertragung der Interrupt-Request-Nummer, Mapping IRQ → Int-Nr.

INTA = Interrupt Acknowledge  
INTR = Interruptmeldung an CPU  
IMR = Interrupt Mask Register  
IRQ = Interrupt Request

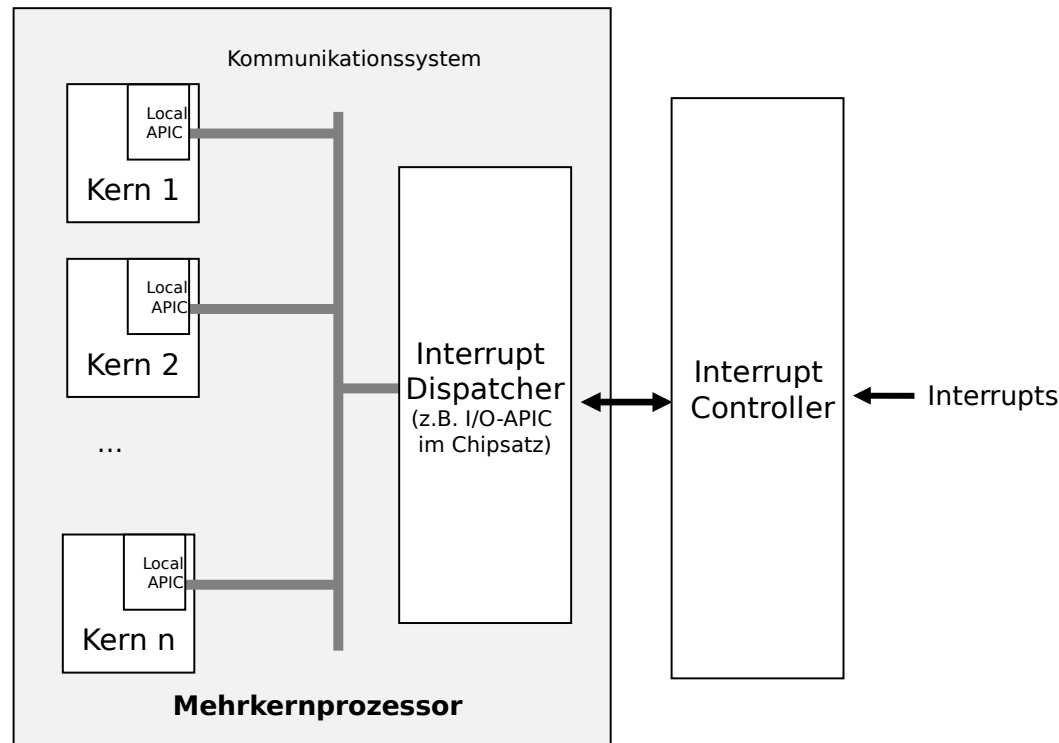
## Beispiel: Intel 8259A-Controller

- Innenleben Intel 8259A, vereinfacht

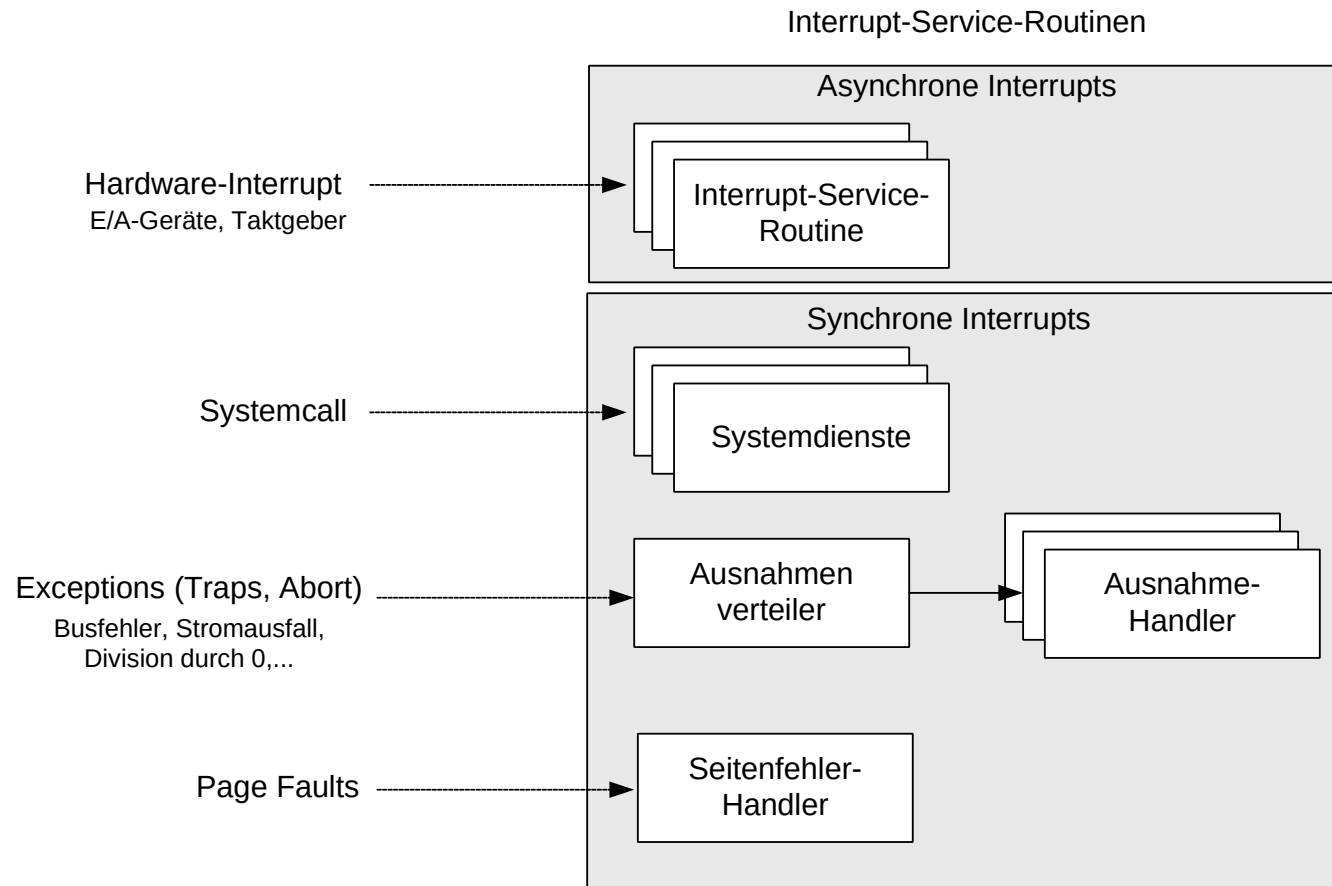


# Interrupt-Dispatching bei Mehrkernprozessoren

- Dispatching-Algorithmus im Interrupt-Dispatcher wählt einen Kern aus, der den Interrupt bearbeiten soll



# Fallbeispiel: Windows, Interrupt-Bearbeitung (1)



# Fallbeispiel: Windows, Interrupt-Bearbeitung (2)

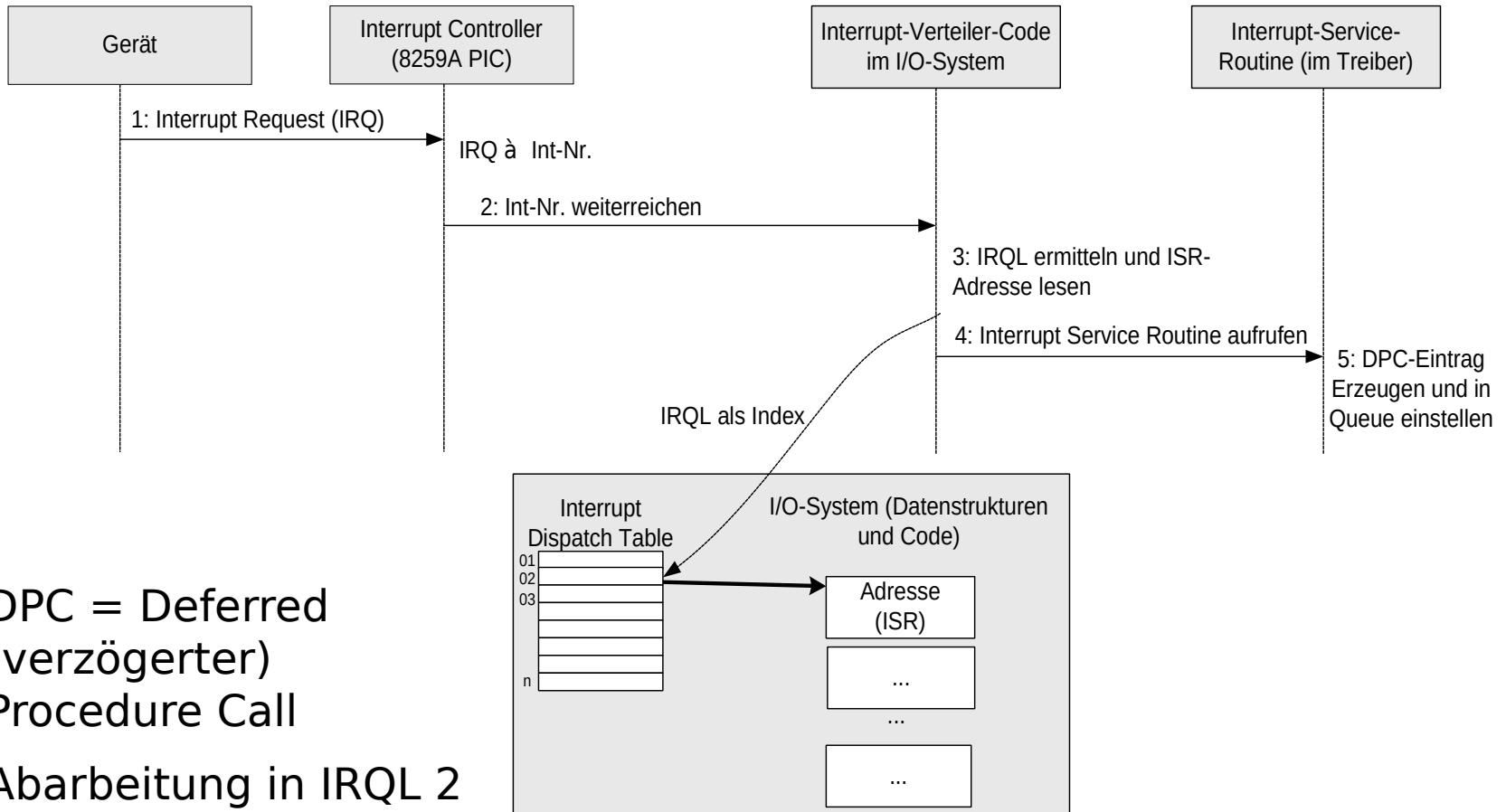
- Windows hat eine eigene Interrupt-Verwaltung
- Über sog. **Interrupt Request Levels** (IRQL) ordnet der Kernel den Interrupts eigene Prioritäten zu
- Nur Interrupts mit höherem IRQL können Interruptbearbeitung auf niedrigerem IRQL unterbrechen
- Über eine Interrupt Dispatch Tabelle (IDT) (Microsofts Bezeichnung für die Interrupt Vector Table) wird festgehalten, welche ISR für welchen Interrupt zuständig ist

# Fallbeispiel: Windows, Interrupt-Bearbeitung (4)

- Interrupt Request Levels (IRQLs) in der x64- und der IA64-Architektur

IRQL	Bezeichnung	Beschreibung
15	High-Level	Maschinen-Check und katastrophale Fehler
14	Power-Level	Strom/Spannungsproblem und Interprozessor-Interrupt
13	Clock-Level	Clock-Interrupt
12	Synch-Level	Prozessorübergreifende Synchronisation
3-11	Device-Levels	Abbildung auf IRQs der Geräte je nach verbautem Interrupt-Controller
2	Dispatch/DPC-Level	Dispatching und Ausführung von Deferred Procedure Calls
1	APC-Level	Ausführung von Asynchronous Procedure Calls
0	Passive-Level	Normale Threadausführung

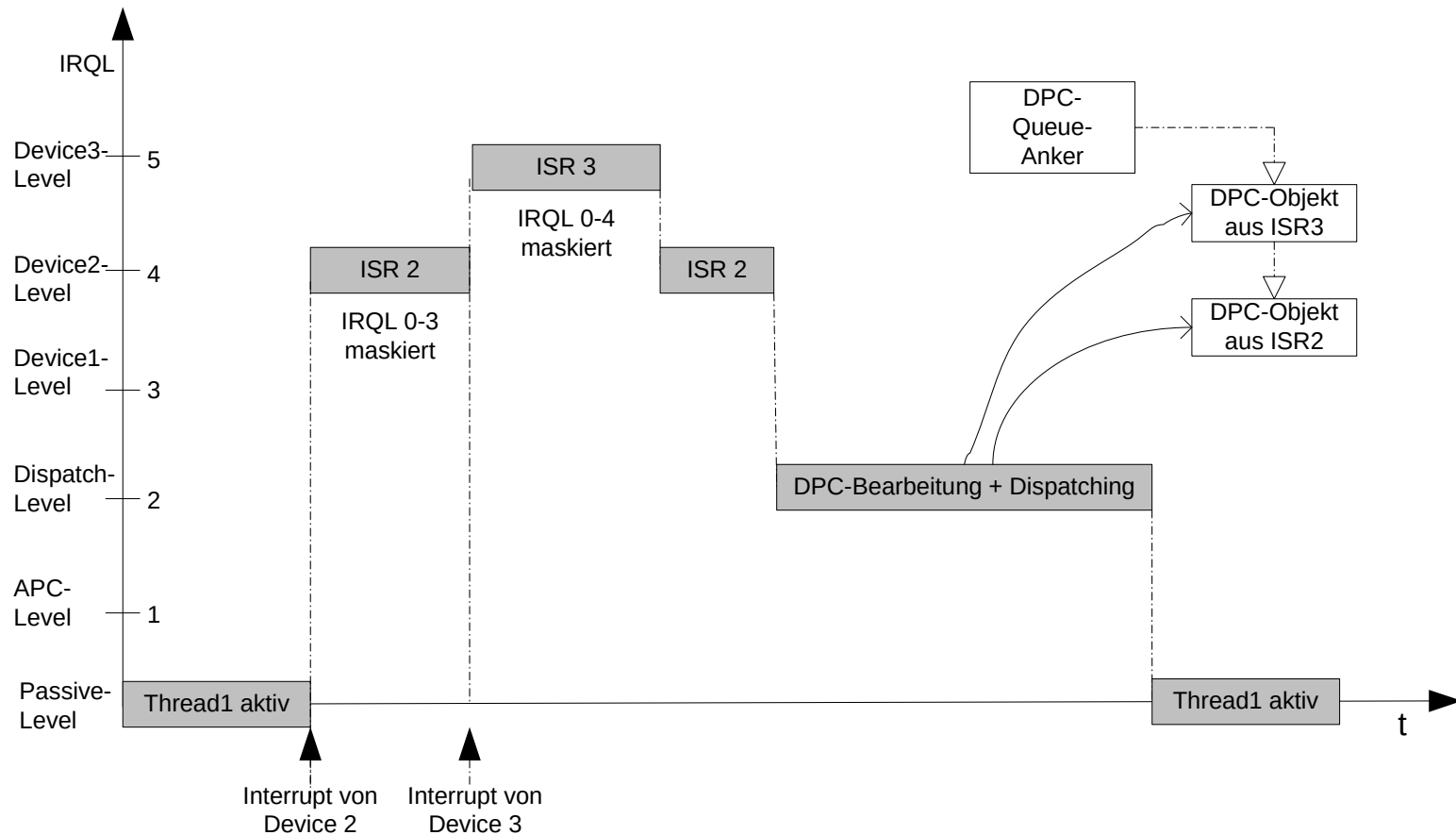
# Fallbeispiel: Windows, Interrupt-Bearbeitung (5)



- DPC = Deferred (verzögerter) Procedure Call
- Abarbeitung in IRQL 2



# IRQL-Nutzung am Beispiel Singleprozessor



# Fallbeispiel: Windows, Interrupt-Bearbeitung (6)

- Bei x86-Systemen unterbricht der Interrupt-Controller die CPU auf einer Leitung (Hardware)
- Die CPU fragt den Controller ab, um eine Unterbrechungsanforderung (IRQ) zu erhalten (Hardware)
- Interrupt-Controller übersetzt IRQ in eine Interrupt-Nummer (Hardware)
- Aus Interrupt-Nummer wird IRQL ermittelt (Software, Windows)
- Die IDT enthält Adresse der anzustoßenden Interruptverteiltroutine

# Fallbeispiel: Windows, Interrupt-Bearbeitung (7)

- Interruptverteilroutine wird aufgerufen und ermittelt die Adresse der ISR
- ISR wird aufgerufen
- ISR erzeugt meist nur ein **DPC-Objekt** (Deferred Procedure Call) und hängt es in eine DPC-Queue ein
  - Kurze Bearbeitungszeiten in der ISR werden angestrebt
  - Kurze Bearbeitung in hoher Prioritätsstufe (IRQL)
  - Systembelastung gering halten
  - die Linux Entsprechung zu einem DPC-Objekt wird als „Tasklet“ bezeichnet
- DPC-Routine wird dann später mit niedrigerer Priorität (IRQL = 2) aufgerufen und ist unterbrechbar(er)

# Fallbeispiel: Windows, APC-Bearbeitung (8)

- APC = Asynchronous Procedure Call
- Anwendungsbeispiel:
  - Bei read-Aufrufen an eine Festplatte wird die Threadbearbeitung zunächst unterbrochen
  - Nachdem der read-Aufruf vom Kernel abgearbeitet wurde, wird ein **APC-Objekt** erzeugt und in die APC-Queue des rufenden Threads eingehängt
  - Die nachfolgende Bearbeitung der APC-Queue erfolgt dann mit **IRQL 1**
- Je eine **APC-Queue** pro Thread im Kernel- und im Usermodus, je nachdem, von wo der read-Aufruf abgesetzt wurde

Hinweis: Threads sind leichtgewichtige Prozesse (später mehr dazu)

# Fallbeispiel: Linux

---

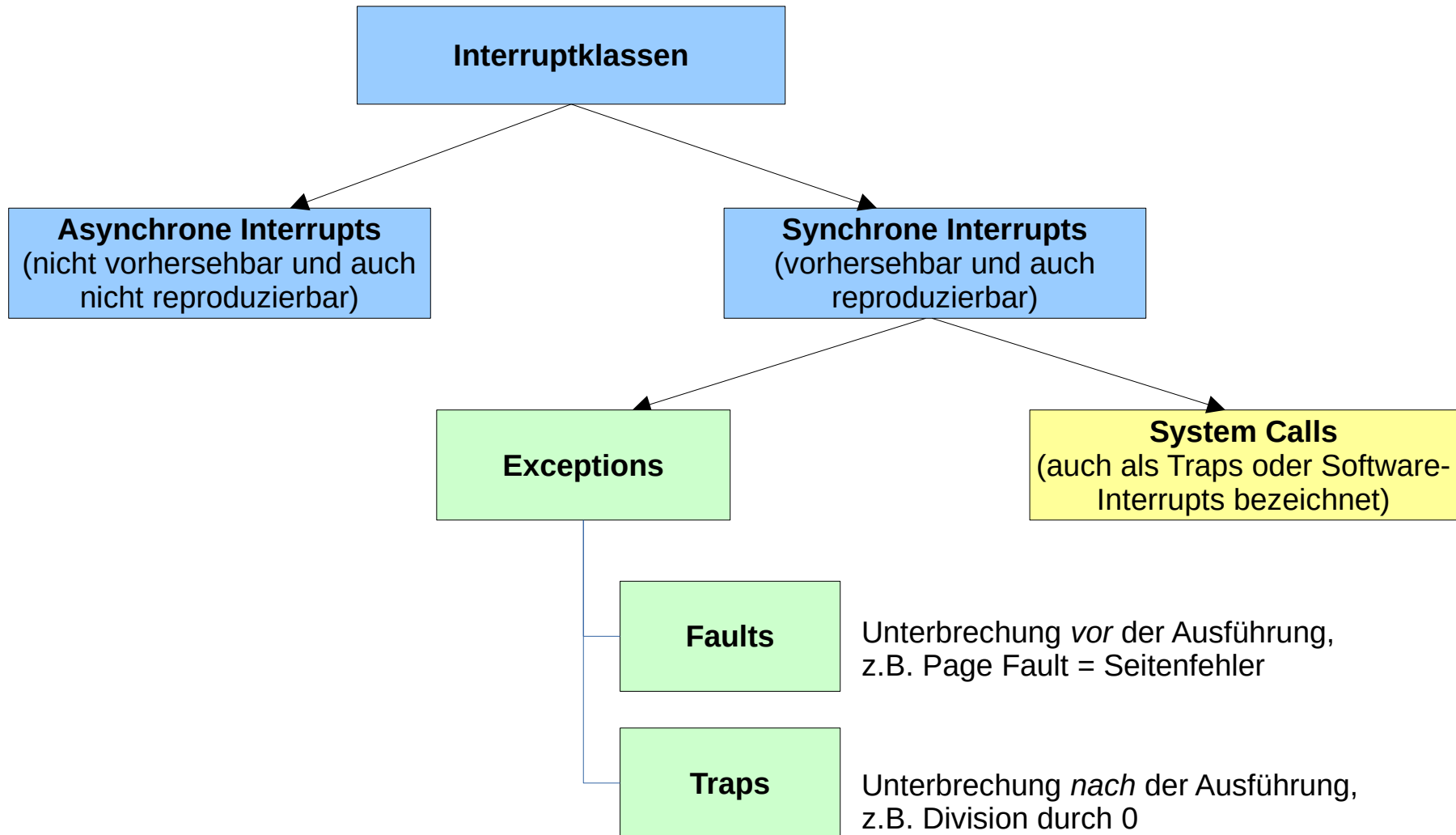
- optionaler Stoff: optional/04-3\_Interruptverarbeitung\_in\_Linux.odp

# Überblick

---

1. Begriffe und Klassifizierung
2. Zusammenspiel der Komponenten
- 3. Systemdienste und Systemcalls**

# Wiederholung: Interrupt-Klassifizierung



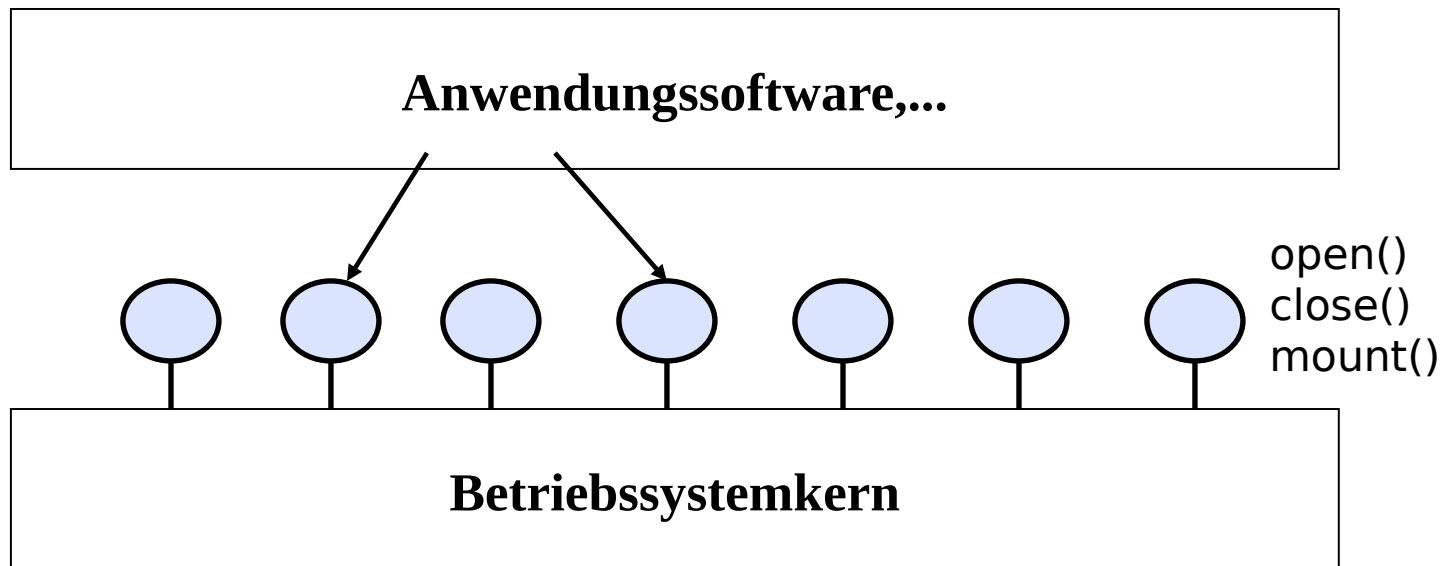
# Dienste des Betriebssystems

- Anwendungsprogramme nutzen die **Dienste** des Betriebssystems, die über sog. Systemcalls aufgerufen werden
- Wohldefinierte Einsprungpunkte ins Betriebssystem
- Spezieller Aufrufmechanismus für einen Systemcall
  - **Software-Interrupt** (als **Trap** bezeichnet) oder **Supervisor Call** (SVC)
  - Vorteil: Anwendungsprogramm muss Adressen der Systemroutinen nicht kennen
- Alle Systemcalls zusammen bilden die Schnittstelle der Anwendungsprogramme zum Betriebssystemkern
  - Zugang zu Systemcalls wird meist via Bibliotheken bereitgestellt



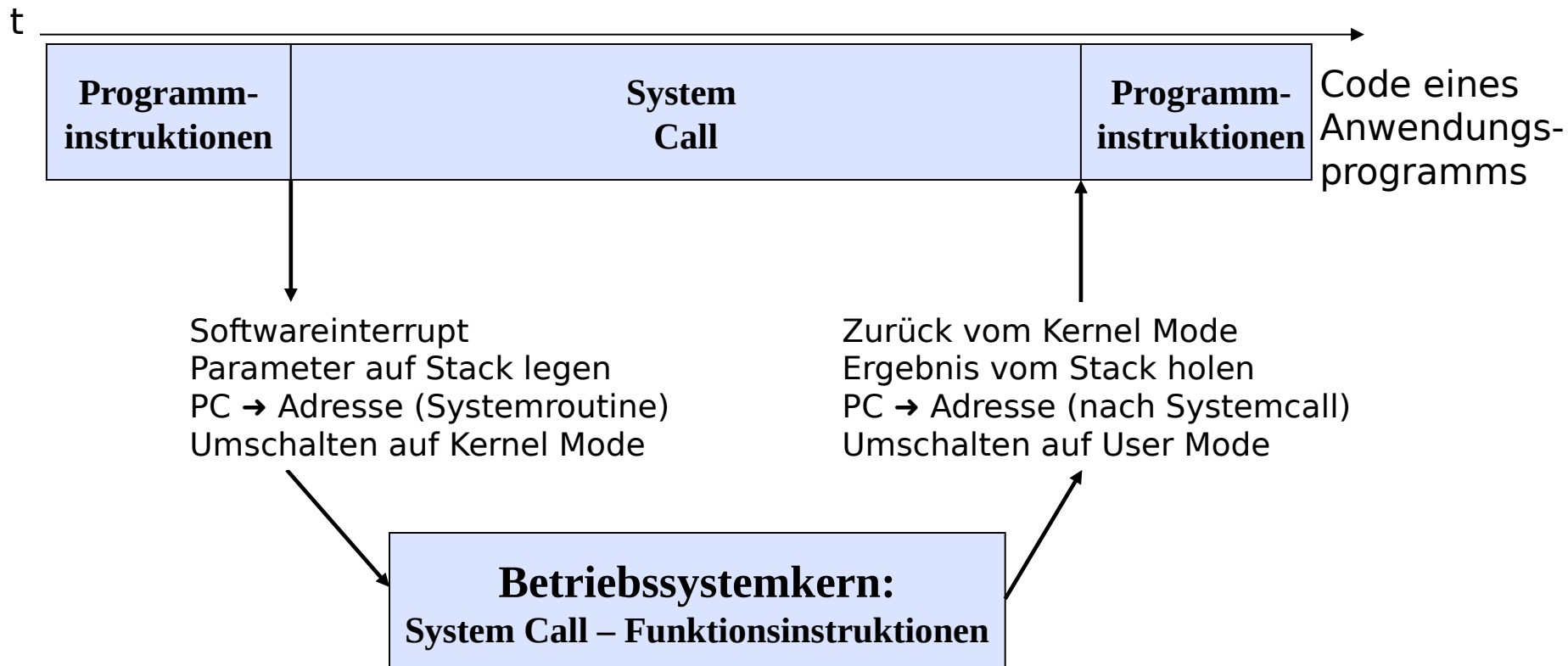
# Umschaltung in den Kernelmodus

- Systemcalls werden im **Kernelmodus** ausgeführt
- Beim Aufruf wird durch den Prozessor vom Usermodus in den Kernelmodus **umgeschaltet**



# Systemcall-Ablauf

## ■ Befehlsfolge eines Systemaufrufs



# Systemcall-Ablauf unter Linux/x86-CPUs (1)

## **x86 Maschinensprache Befehle:**

```
int n           // Aufruf des Interrupts mit der Interrupt-Vektor-Nummer n
int $0x80       // Trap, Systemcall
```

## **C-Code:**

```
#include <stdio.h>
main()
{
    open("mandl.txt",1);    // Datei zum Lesen öffnen
}
```

## **Maschinencode** (gcc -S datei.c):

```
.LCO:
.string "mandl.txt"
.text
.globl main
main:
...
call open
...
```

# Systemcall-Ablauf unter Linux/x86-CPUs (2)

## Open-Routine in der Linux C-Library

(gcc -static ..., 386 Architektur):

`__libc_open:`

```
...  
mov 0xc(%esp,1), %edi // Parameter für Open in Register laden  
mov ...  
mov $0x5, %eax        // Systemcall-Code für open-Funktion  
int $0x80              // Systemcall (auf x86_64: `syscall`)  
                        // - CS (Code Segment) Register auf Stack gepusht  
                        // - IP (Instruction Pointer) auf Stack gepusht
```

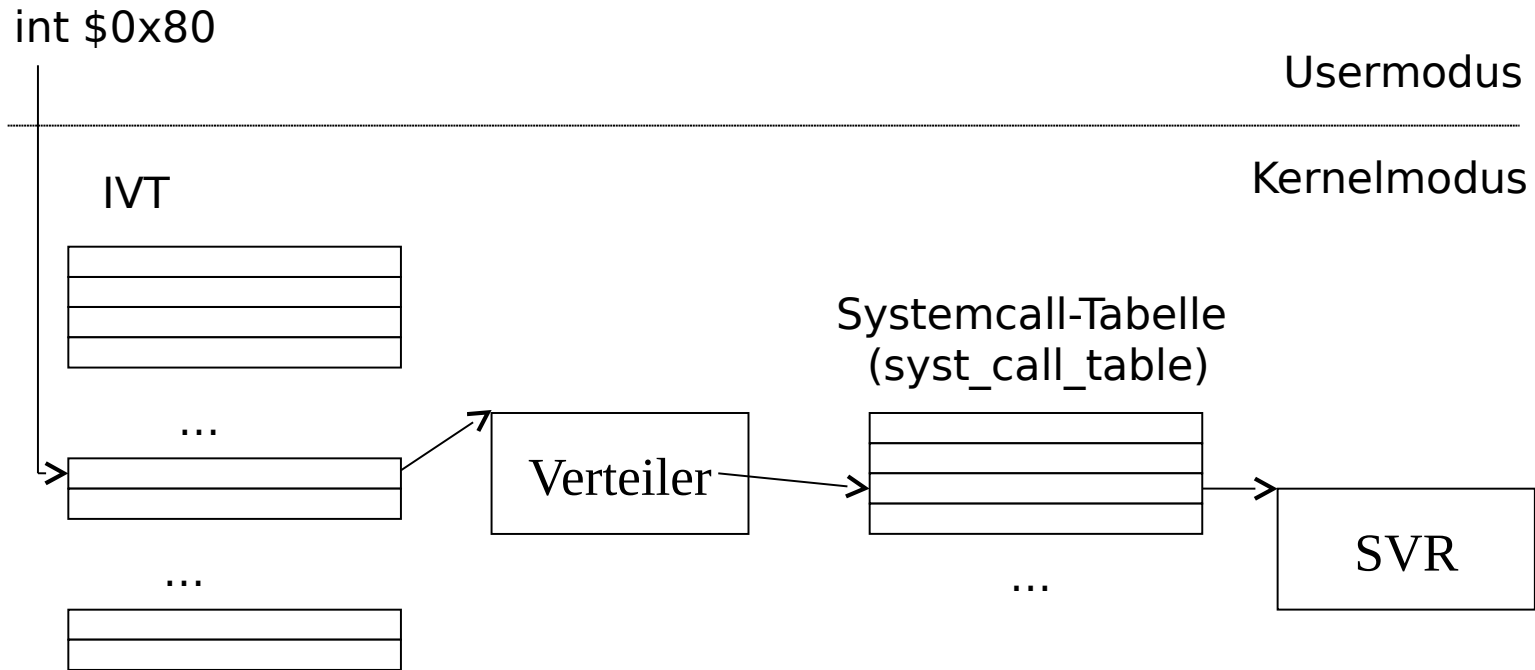
...

im Kernel:

```
...  
iret                // zurück aus dem Interrupt zum Aufrufer  
                        // - CS und IP werden von Stack gepopt
```

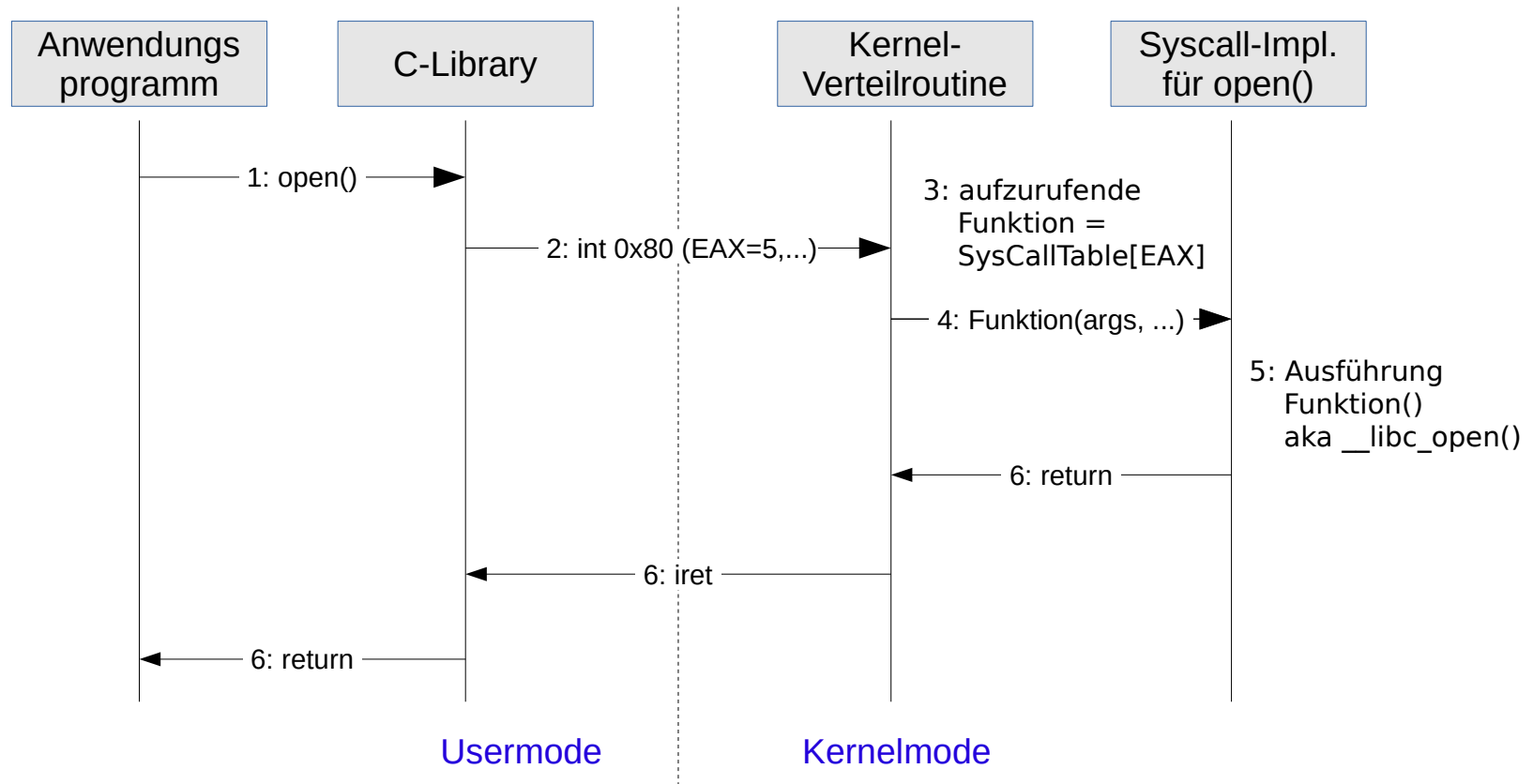
- Interrupt-Vektor-Nummer \$0x80 ist Index für die Adressierung der Interrupt-Vektor-Tabelle (IVT)
- (CS,IP) → Adresse der nächsten Anweisung

## Systemcall-Ablauf unter Linux/x86-CPU's (3)



- IVT = Interrupt-Vektor-Tabelle,
- SVR = Serviceroutine, bearbeitet Systemcall
- Systemcall-Tabelle verweist auf Implementierungen der Systemdienste

# Systemcall-Ablauf unter Linux: Sequenzdiagramm



- Systemcall-Tabelle enthält alle Verweise auf Systemcalls (struct sys\_call\_table)

## Vergleichbare Befehle in anderen Prozessoren

- **EPC-Befehl** = Enter Privileged Mode in IA64
  - Übergabe von max. 8 Parametern in Registern und die restlichen Parameter im Kernelstack
- **syscall-Befehl** in x64
  - Übergabe der Systemcall-Nr. im EAX-Register
  - Max. 8 Parameter in Registern und die restlichen Parameter im Kernelstack
- **sysenter-Befehl** in x86 Pentium II
- **SVC-Befehl** in ARM-Architektur
  - früher SWI = Software-Interrupt

# Systemcalls bei POSIX

- Systemcalls sind standardisiert in ISO 9945-1
- POSIX-Konformität erfüllen die meisten Unix-Derivate
- Beispiele:
  - fork() : Prozesserzeugung
  - execve() : Aufruf eines Programms
  - exit() : Beenden eines Prozesses
  - open() : Datei öffnen
  - close() : Datei schließen
  - read() : Daten aus Datei lesen
  - write() : Daten in Datei schreiben

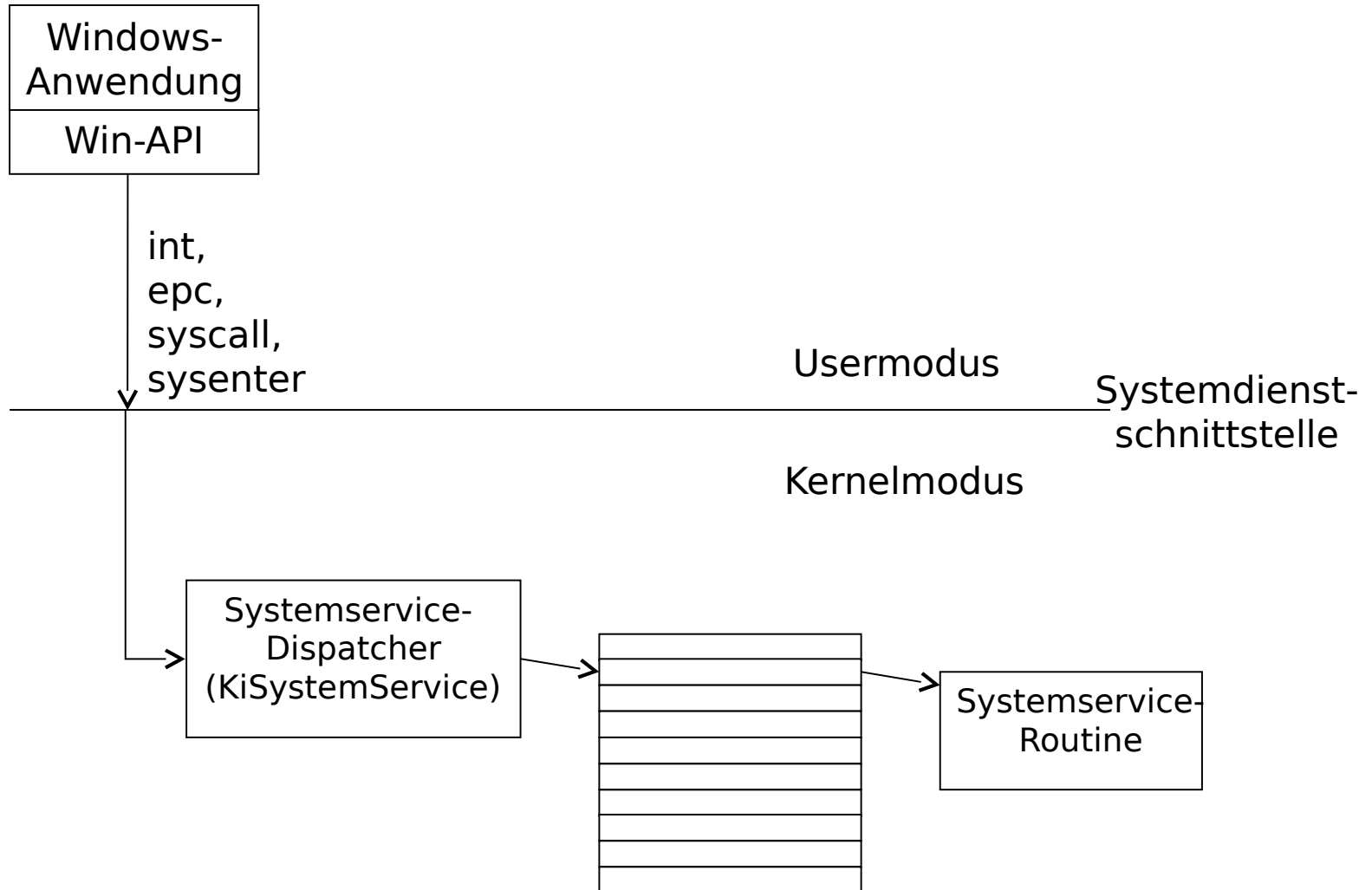
POSIX = **P**ortable **O**perating **S**ystem **U**nix



## Systemcalls bei Win32-API

- In Windows-Systemen sind die Systemcalls in der Windows-API (Application Programming Interface) definiert
- Es gibt einige 1000 API-Funktionen, einige davon sind Systemcalls
- Beispiele:
  - CreateProcess(): Prozesserzeugung
  - exitProcess(): Beenden eines Prozesses
  - CreateFile(): Erzeugen und Öffnen einer Datei
  - CloseHandle(): Datei schließen
  - ReadFile(): Daten aus Datei lesen
  - WriteFile(): Daten in Datei schreiben

# Systemcall-Ablauf unter Windows



# Überblick

---

1. Begriffe und Klassifizierung
2. Zusammenspiel der Komponenten
3. Systemdienste und Systemcalls

# Zusammenfassung Interruptverarbeitung

---

- Polling und Interrupts
- Synchrone und asynchrone Interrupts
- Interrupt-Requests (IRQ) und Interrupt-Bearbeitung
- Interrupt-Vektor-Tabelle (IVT) zur Adressierung von Interrupt-Service-Routinen (ISR) für synchrone und asynchrone Interrupts
- Systemcalls = synchrone Software-Interrupts, (oft auch, etwas verwirrend als Traps bezeichnet)

# Gesamtüberblick

---

- ✓ Einführung in Computersysteme
- ✓ Entwicklung von Betriebssystemen
- ✓ Architekturansätze
- ✓ Interruptverarbeitung in Betriebssystemen
- 5. Prozesse und Threads
- 6. CPU-Scheduling
- 7. Synchronisation und Kommunikation
- 8. Speicherverwaltung
- 9. Geräte- und Dateiverwaltung
- 10. Betriebssystemvirtualisierung