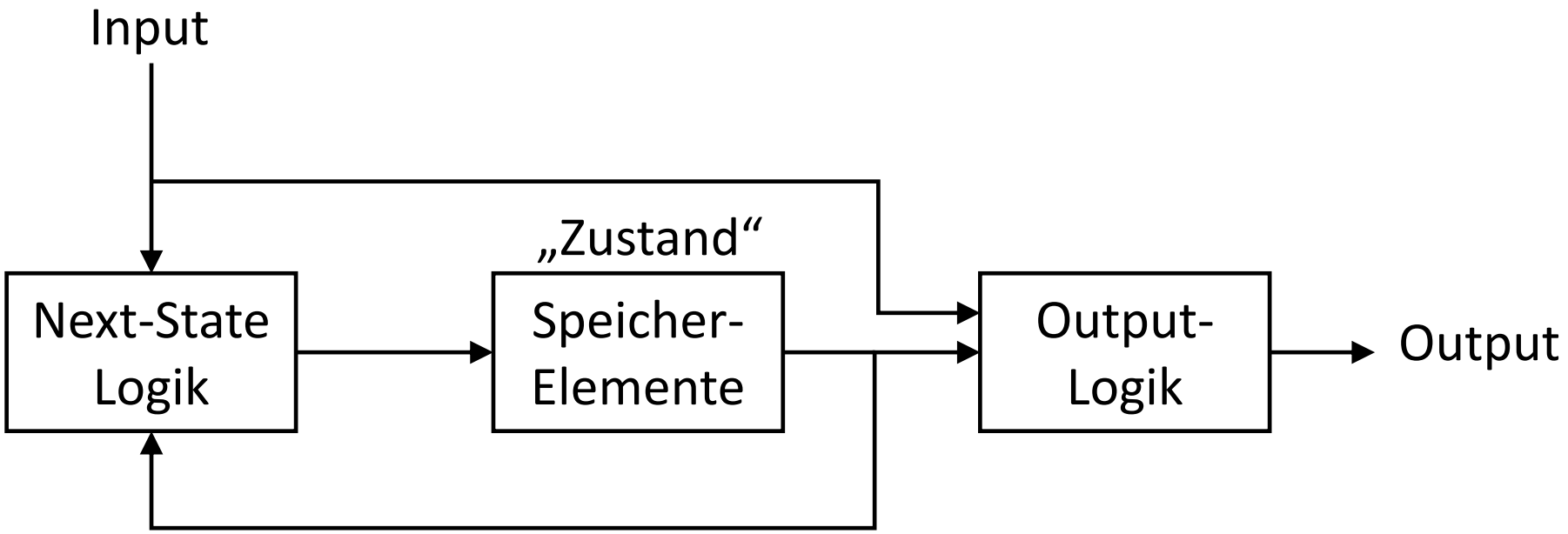
Kapitel 10: Rechnerarchitekturen

Abschnitt 10.1

Von Neumann-Rechner

- Charakteristika des von Neumann-Rechners
- Struktur eines von Neumann-Rechners
- Struktur einer CPU
- Speicherhierarchie
- I/O
- Interrupts
- Klassifikation der von Neumann-Rechner

Sequentieller Rechner



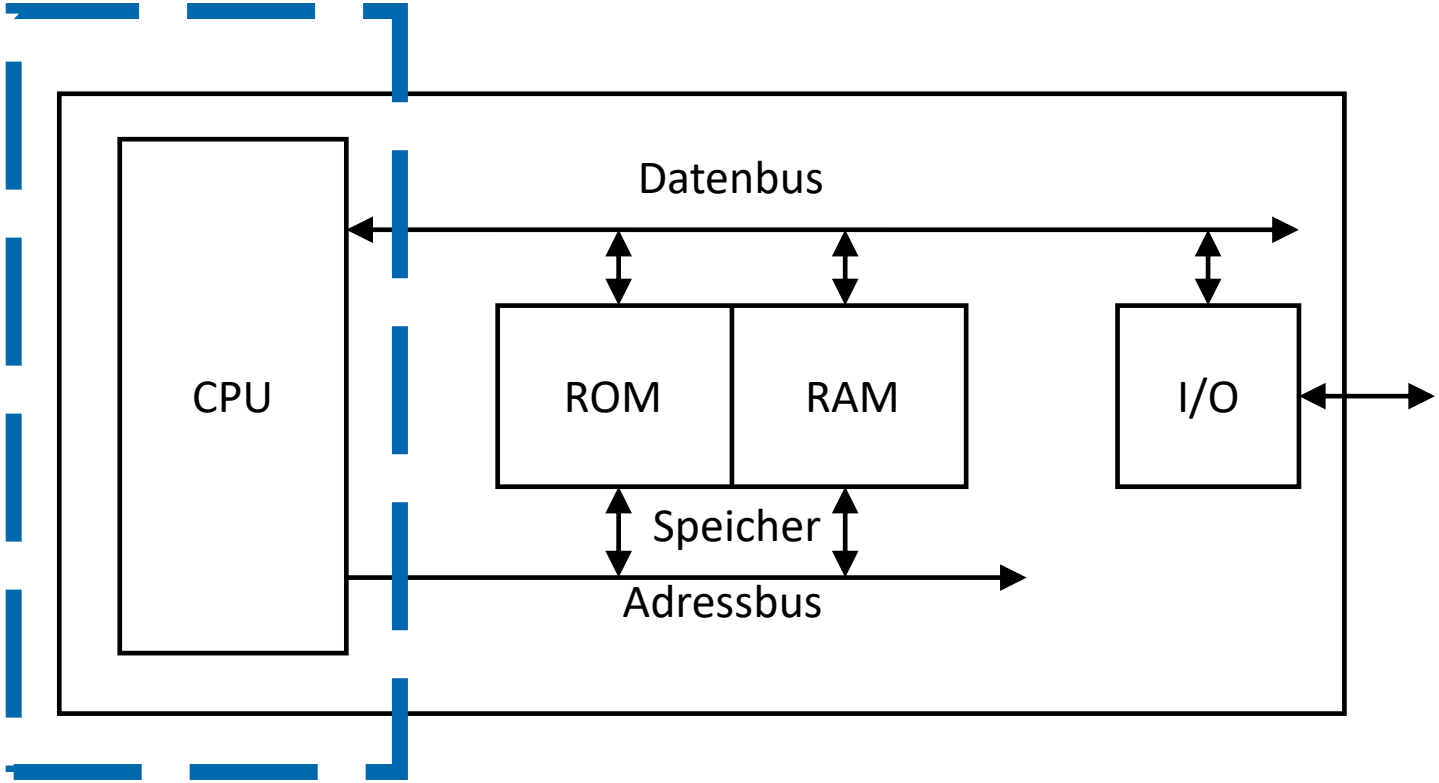
Charakteristika des von Neumann-Rechners

1. Zu jedem Zeitpunkt führt die CPU *genau einen Befehl aus*, und dieser kann (höchstens) *einen* Datenwert bearbeiten. (**SISD-Prinzip**) *SISD = Single Instruction, Single Data*

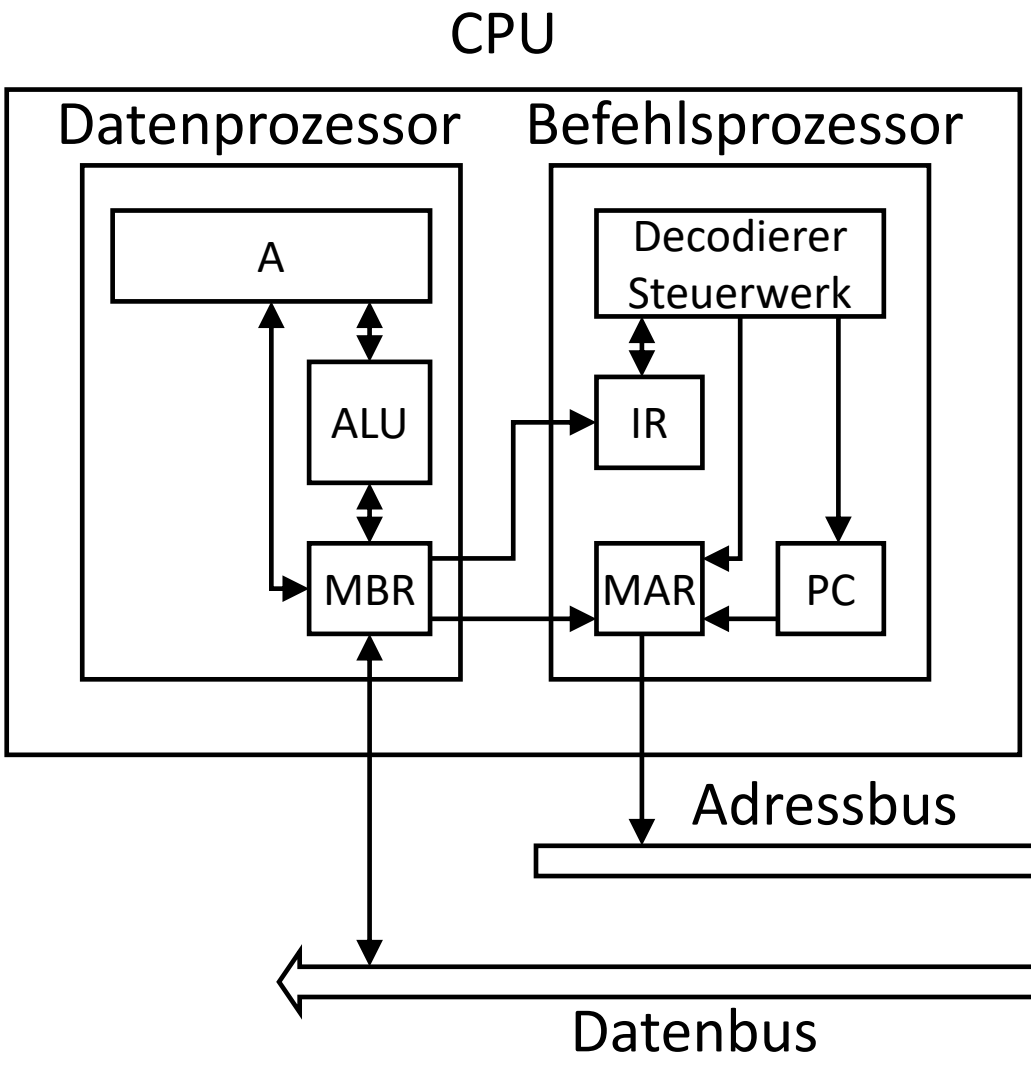
2. Alle Speicherworte (d.h. Inhalte der Speicherzellen) sind als Daten, Befehle oder Adressen brauchbar. Die jeweilige Verwendung richtet sich nach dem momentanen Kontext.

3. Da Daten und Programme nicht in getrennten Speichern untergebracht werden, besteht grundsätzlich keine Möglichkeit, die Daten vor ungerechtfertigtem Zugriff zu schützen.

Struktur eines von Neumann-Rechners



Struktur einer CPU



A = Akkumulator

ALU = Arithmetic Logic Unit

MBR = Memory Buffer Register

IR = Instruction Register

MAR = Memory Allocation Register

PC = Program Counter

Fetch/Execute-Zyklus

2-Phasen-Konzept der Befehlsverarbeitung:

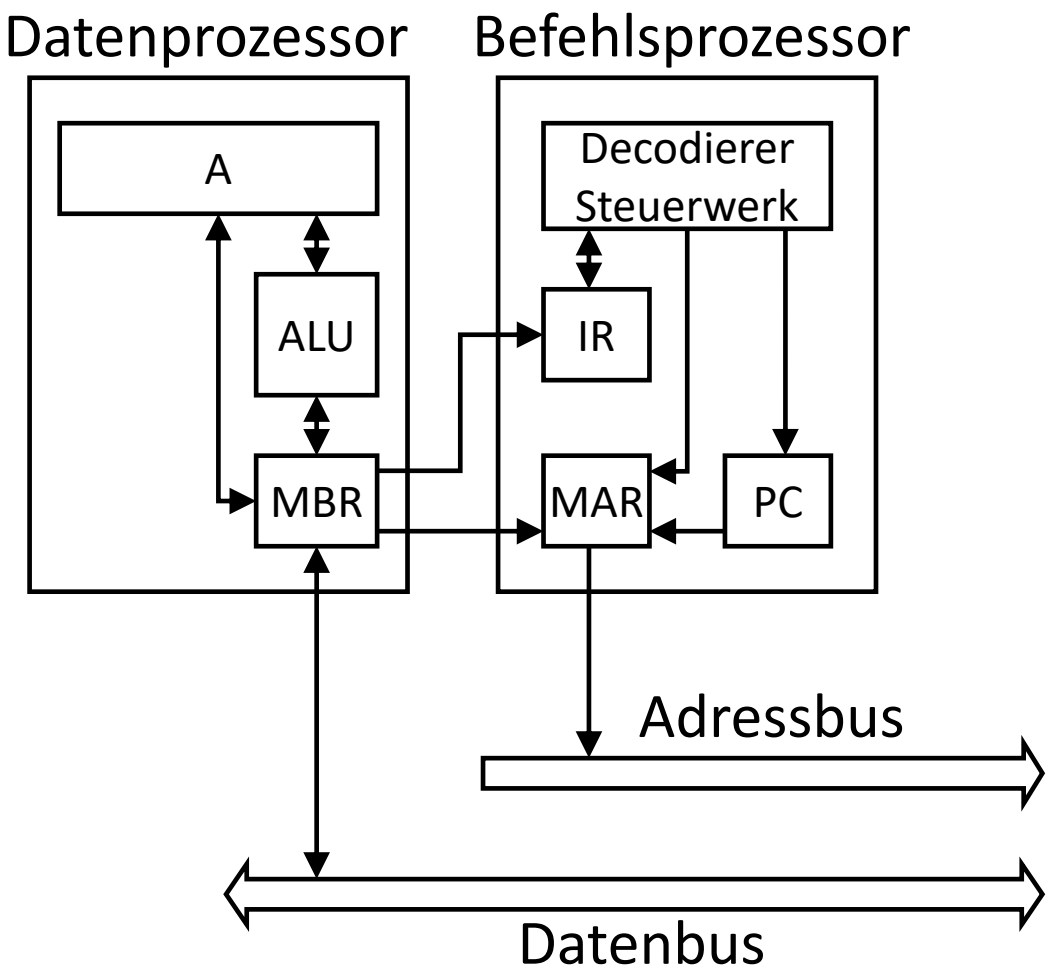
1. In der **Fetch-Phase** wird der Inhalt von PC nach MAR gebracht und der Inhalt dieser Adresse aus dem Speicher über MBR nach IR geholt. Der Rechner geht jetzt davon aus, *dass* es sich um einen Befehl handelt.
   Der Decodierer erkennt, um *welchen* Befehl und um welchen Befehls*typ* es sich handelt und veranlasst ggf. die Bereitstellung von Operanden.
   Schließlich muss der Inhalt von PC aktualisiert werden.

2. In der darauf folgenden **Execution-Phase** erfolgt die Befehlsausführung sowie eine Initiierung der Fetch-Phase für den nächsten Befehl.

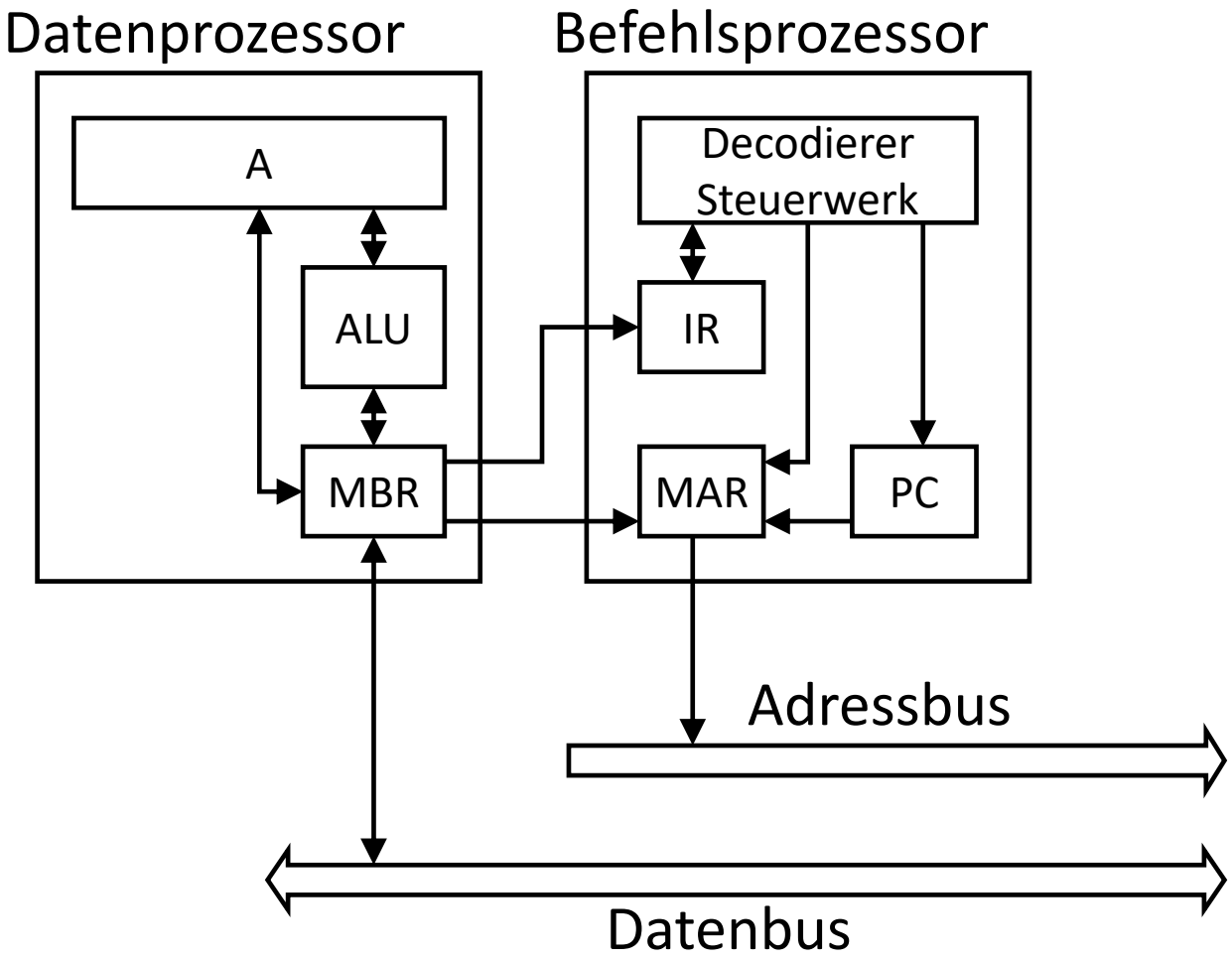
Fetch-Phase

<x> = Wert an Adresse x

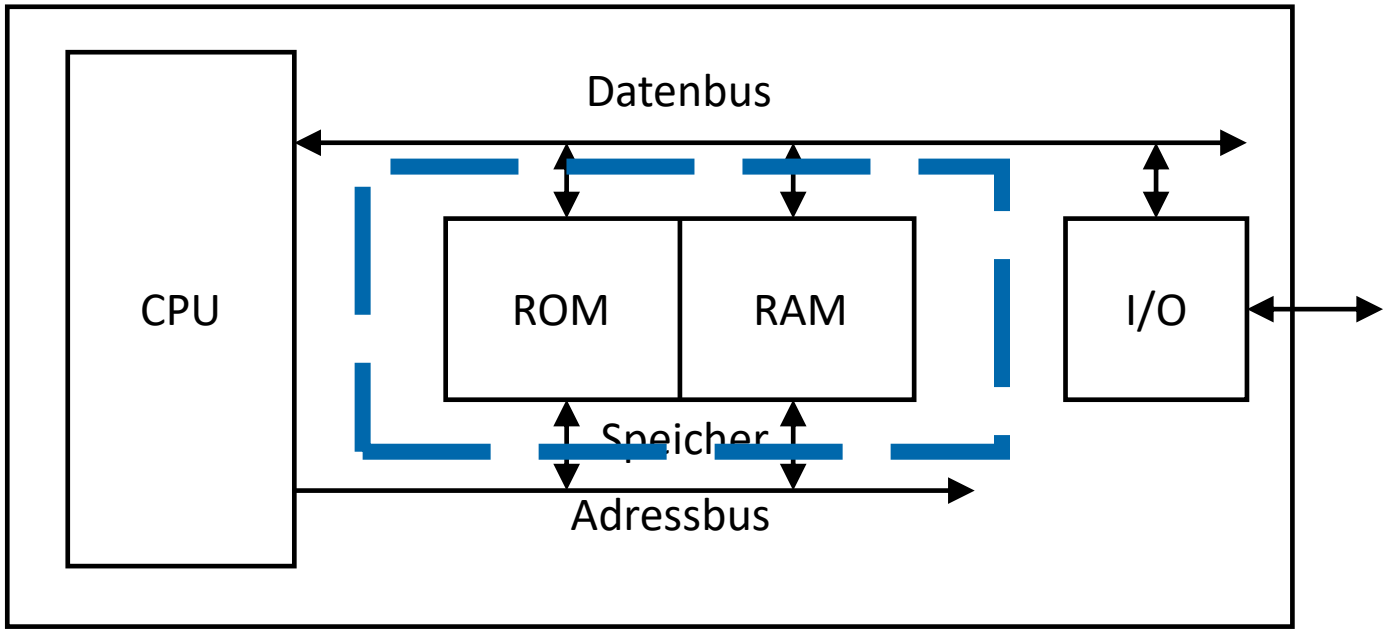
```
MAR <- PC;
MBR <- <MAR>;
IR <- MBR;
Decodiere IR;
Falls unbedingter Sprungbefehl
    dann {
            stelle Zieladresse bereit;
            PC <- Sprungzieladresse;
           }
    sonst {
            stelle Operanden bereit;
            PC <- PC + 1;
           }
```



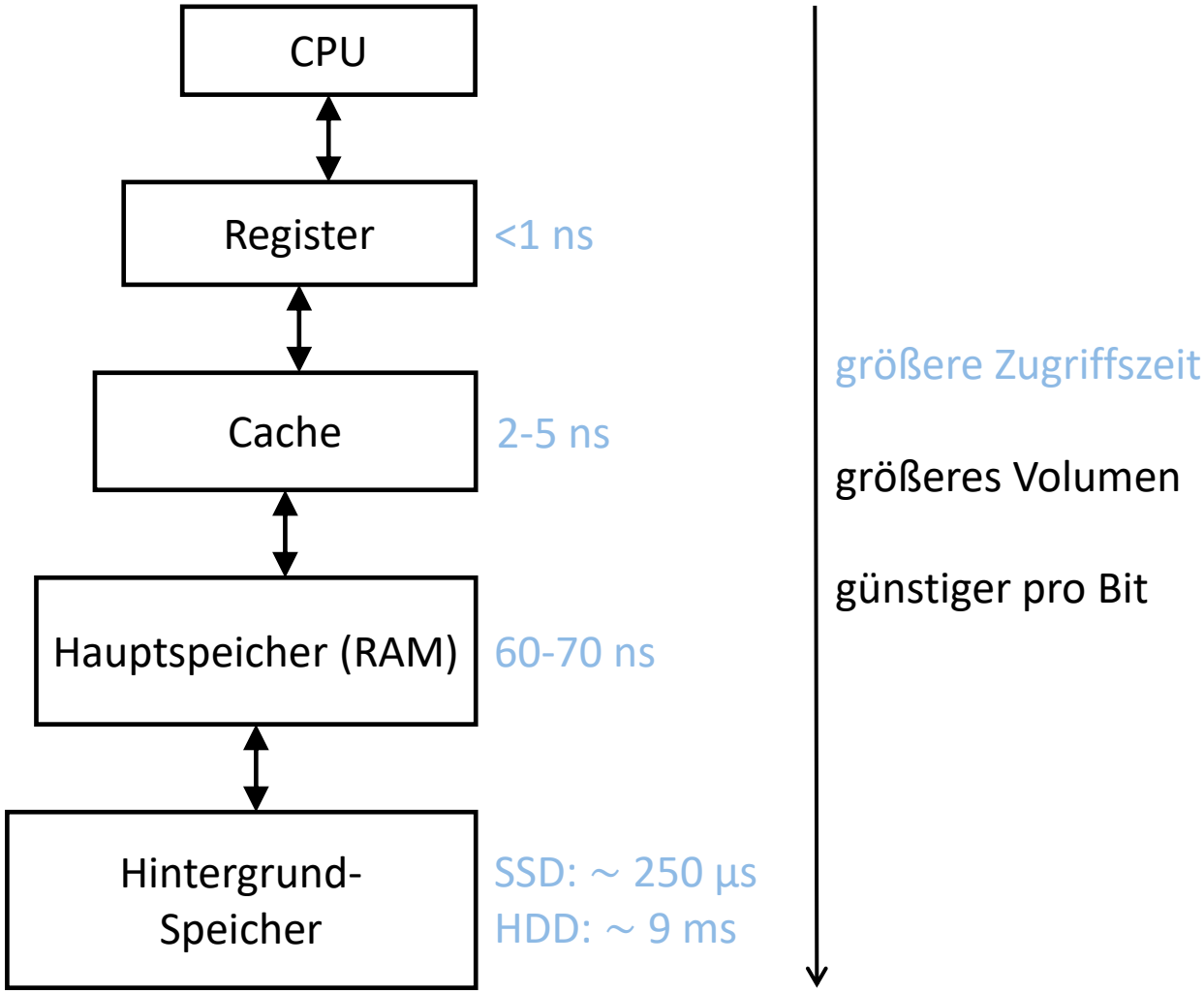
Beispiel



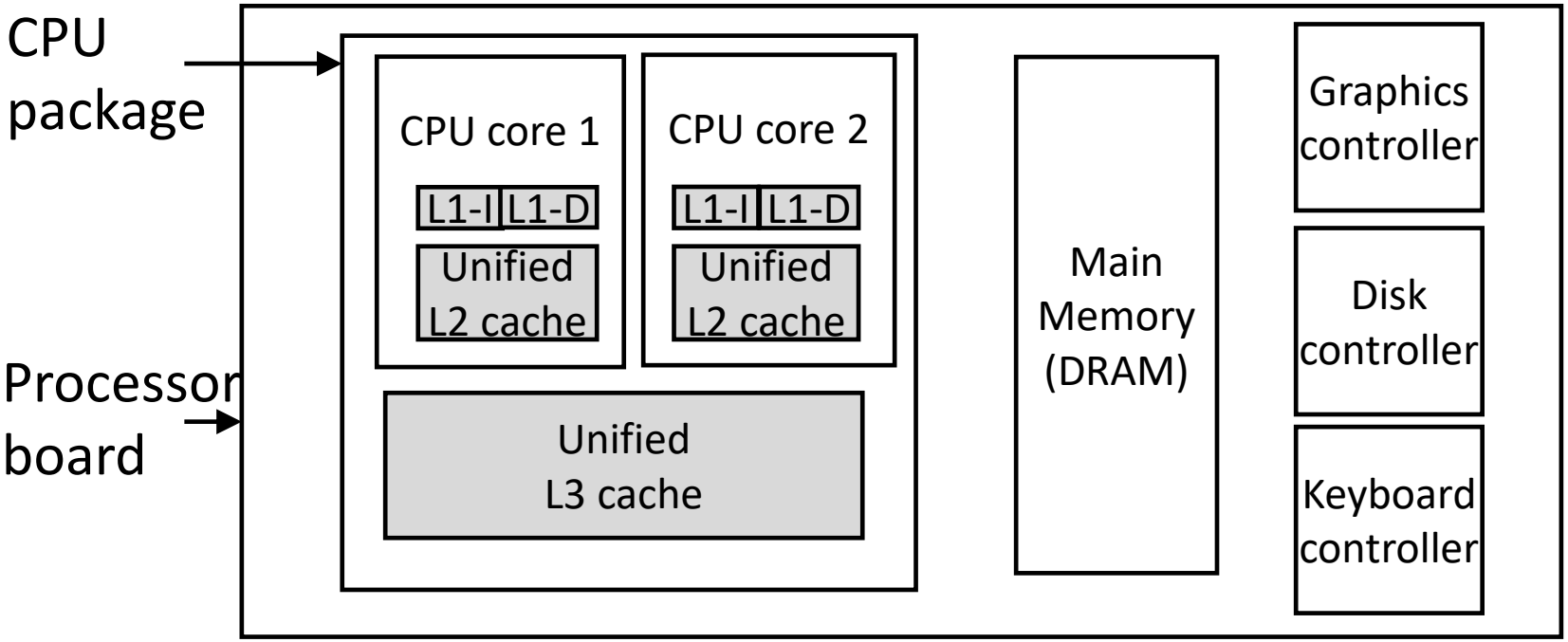
Struktur eines von Neumann-Rechners



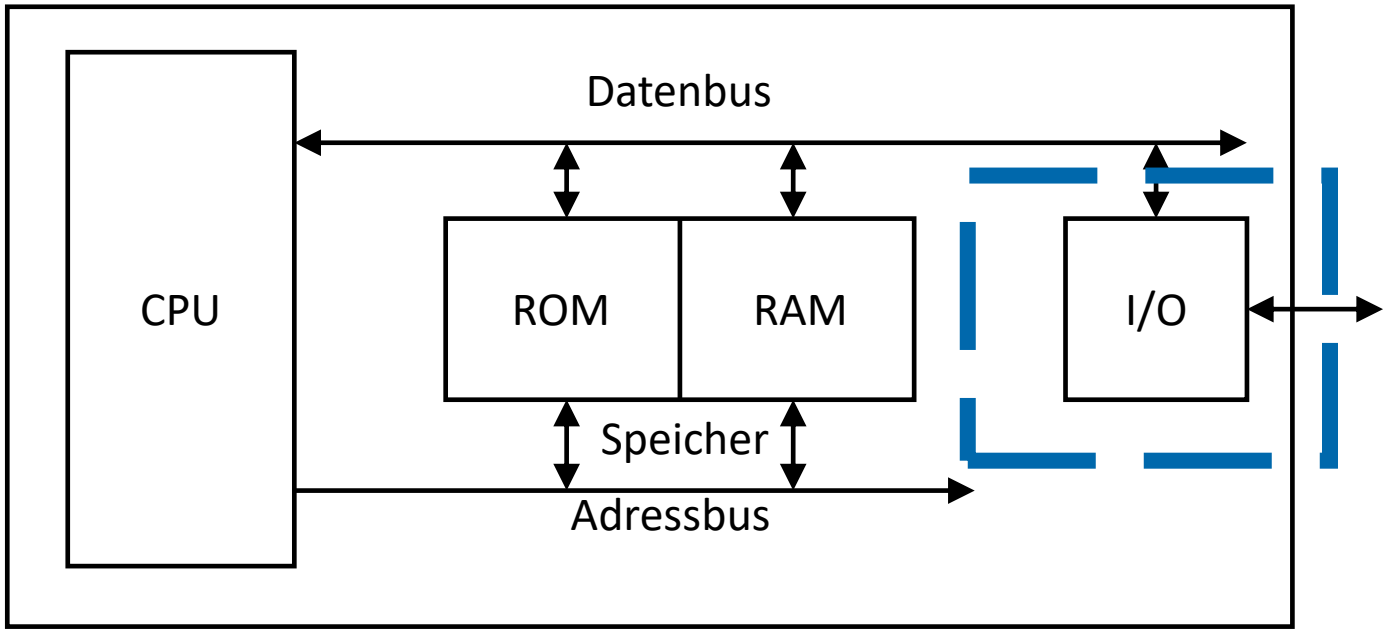
Speicherhierarchie



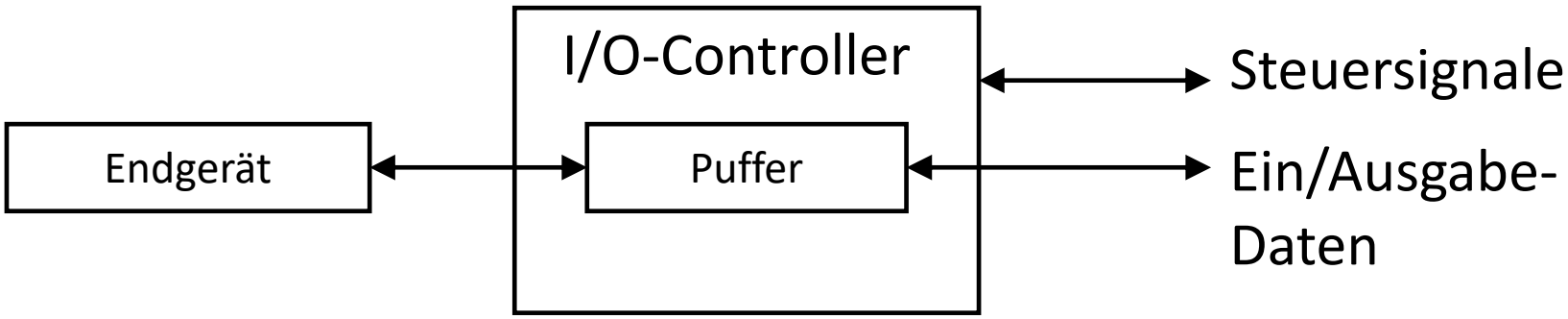
Beispiel einer Cache-Hierarchie
System mit 3 Cache-Ebenen



Struktur eines von Neumann-Rechners



Organisation einer I/O-Einheit



3 Arten der Datenübertragung:

- programmierter I/O
- interrupt-gesteuerter I/O
- Direct Memory Access

Beispiel: Interrupt-Gesteuerter I/O

1. Die Tastatur ist bereit zur Übertragung, der I/O Controller sendet ein Interrupt-Signal an die CPU

2. CPU unterbricht, liest I/O Controller Status, sendet Startsignal an I/O Controller und setzt Programmbearbeitung fort.

3. I/O Controller puffert Daten. Wenn Eingabe beendet oder Puffer voll ist, weiteres Interrupt-Signal an die CPU.

4. CPU unterbricht, überträgt Daten von I/O Controller in den Speicher, setzt Programmbearbeitung fort.

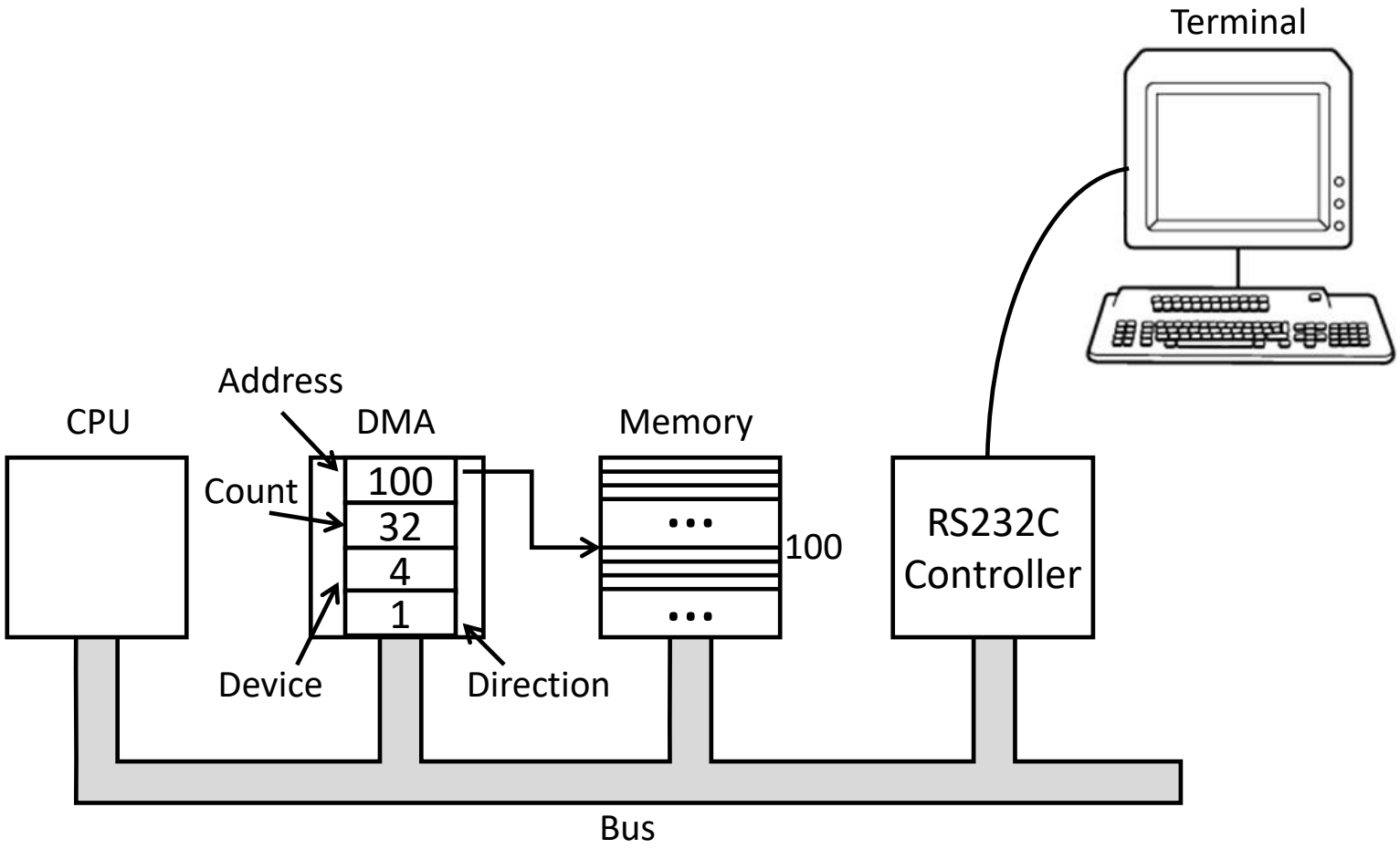
Interrupts

- **Externer Interrupt**: Wird außerhalb der CPU erzeugt (z.B. von I/O Controller)
- **Interner Interrupt**: Wird von der CPU erzeugt (z.B. Division durch 0) und startet damit Fehlerbehandlung

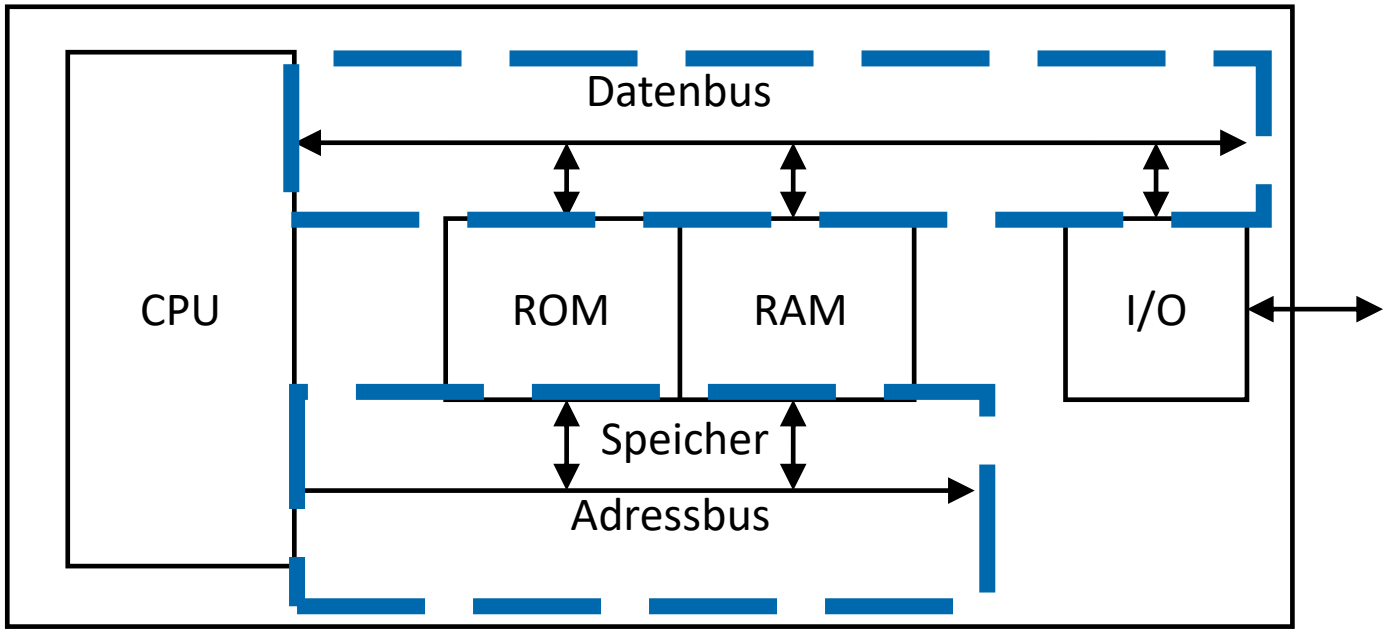
- **Maskierter Interrupt**: Wird nicht sofort behandelt, sondern erst nach Beendigung des aktuellen Programms
- **Unmaskierter Interrupt**: Unterbricht die Programmbearbeitung sofort

- **Priorität von Interrupts**: Legt die Reihenfolge der Interruptbehandlung fest.

Beispiel: Direct Memory Access (DMA)

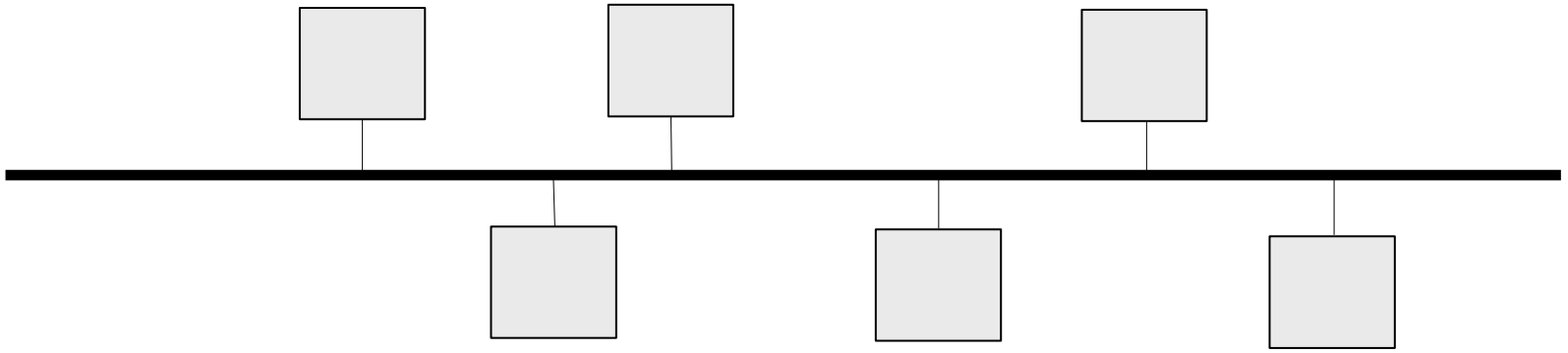


Struktur eines von Neumann-Rechners



Bus-Kommunikation

- Bus: gemeinsames Übertragungsmedium
  - omnibus (*lateinisch: "für alle, von allen"*)



- Vorteile gegenüber anderen Netzwerk-Topologien:
  - Geringe Kosten / geringe Leitergesamtlänge
  - Flexible Anbindung zusätzlicher Komponenten
- Dafür u.U. komplizierte Protokolle
- Anwendungsgebiet (neben Rechnern) z.B. Automobil (CAN-Bus)

Bus-Charakteristika

- Verbindet mehrere Komponenten eines Systems
- Kann aus mehreren Kanälen bestehen
- Besitzt ein Protokoll (von der physikalischen bis zur Anwenderschicht)
- Ist entweder
  - Synchron (taktgesteuert, z.B. CPU-Speicher-Busse) oder
  - Asynchron (kein Takt, sondern Handshake o.ä.)
- Muss evtl. Konflikte (gleichzeitige Anfragen) lösen. Z.B. nach Priorität oder Zufallsprinzip
- Kann entweder seriell oder parallel Daten übertragen

Serielle/Parallele Datenübertragung

- Serielle Übertragung
  - Bietet sich für Verbindungen über große Entfernungen an.
  - Falls Blöcke/Wörter übertragen werden, ist beim Sender eine parallel/seriell und empfangsseitig eine seriell/parallel Umwandlung notwendig.

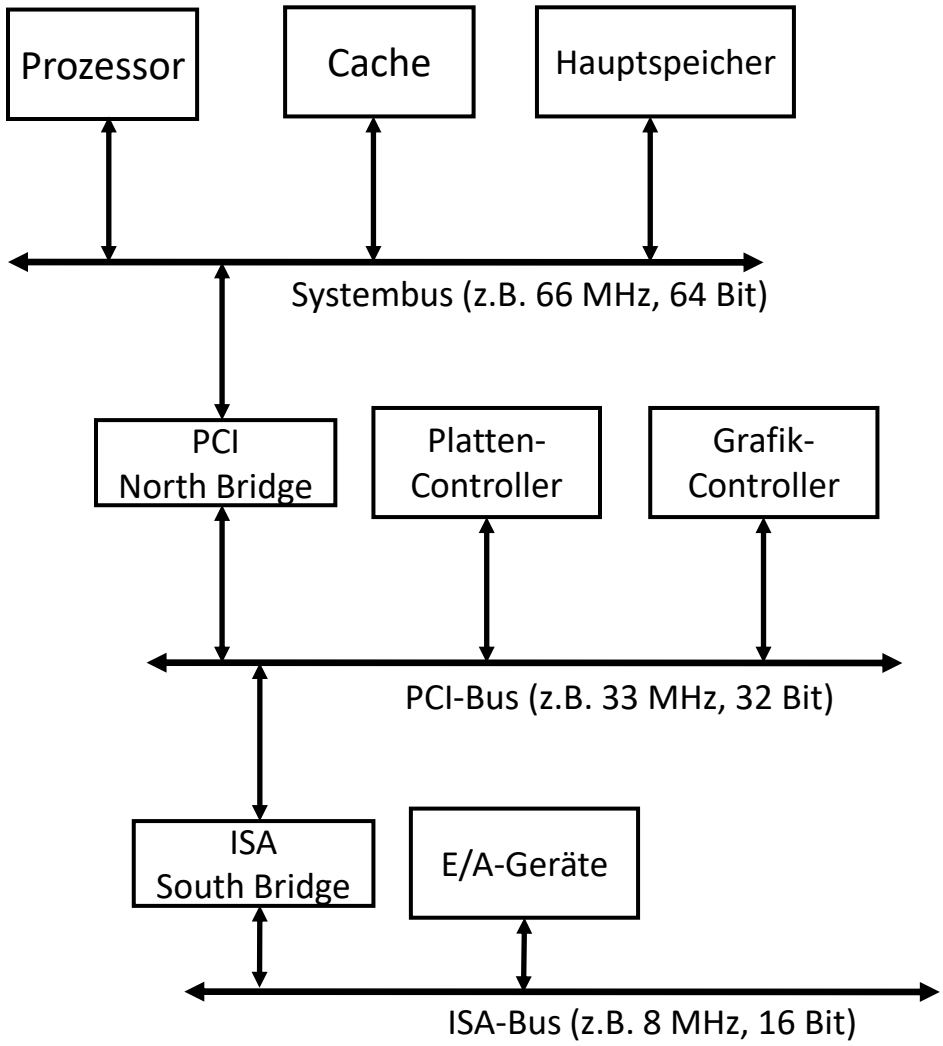
- Parallele Übertragung
  - Zur Verbindung sehr nahe beieinanderliegender Komponenten (z.B. rechnerintern).
  - Auf Platinen ergibt sich das Problem, eine möglichst effiziente Leiterführung zu finden.

Aufbau eines parallelen Busses

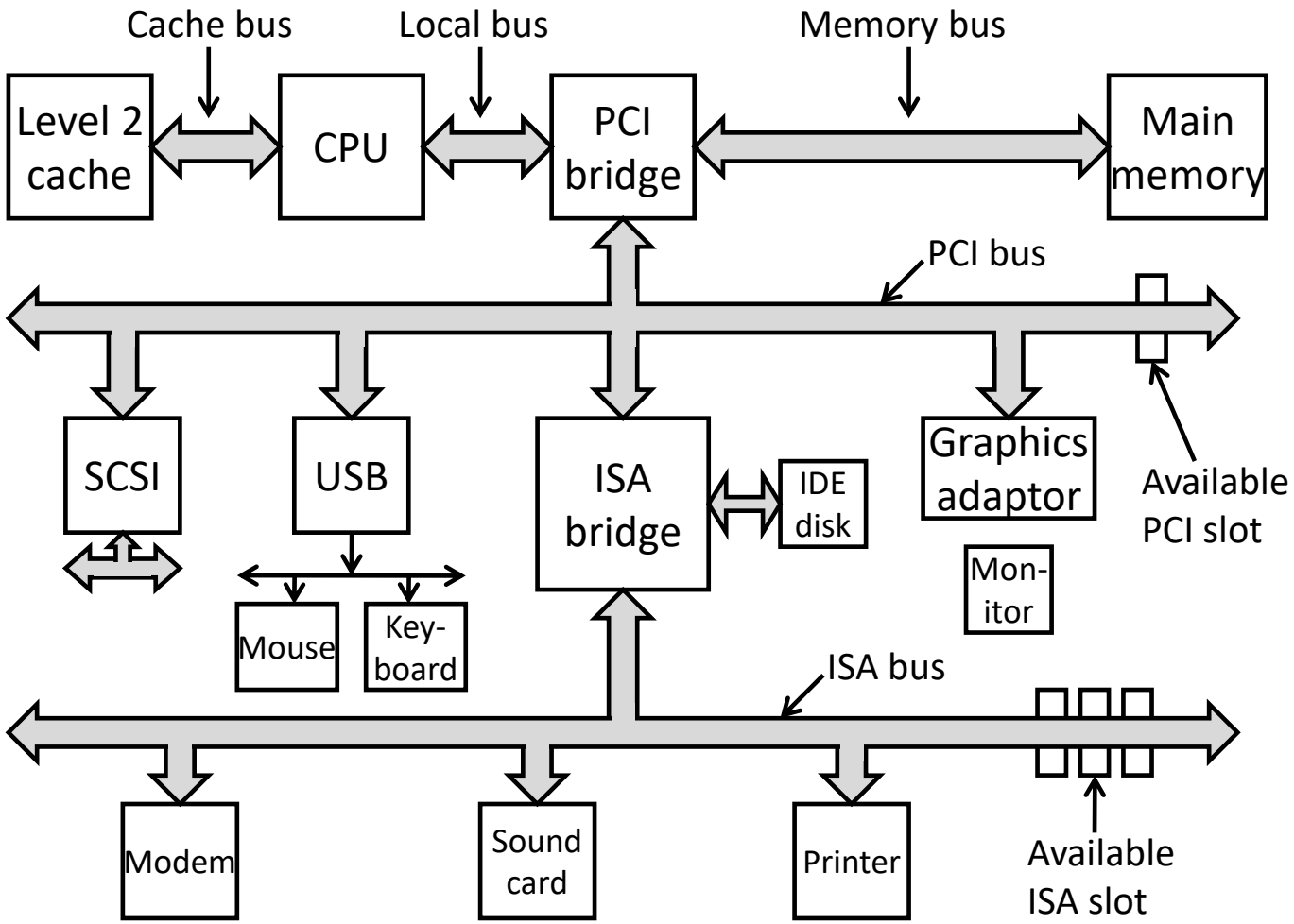
- Adressbus
  - Busbreite bestimmt wie viel Speicher adressiert werden kann
  - logische Busbreite kann durch Chips größer als die physikalische sein
  - Wird vom Busmaster (CPU, evtl. DMA-Chips) angesteuert
  - Adressdekoder bestimmt und informiert adressierte Komponente (z.B. über Chip-Select Eingänge)
- Datenbus für den (bidirektionalen) Datenaustausch
- Steuerbus
  - Lese- / Schreibsteuerung
  - Interrupt-Steuerung
  - Taktung

Mehrbussysteme

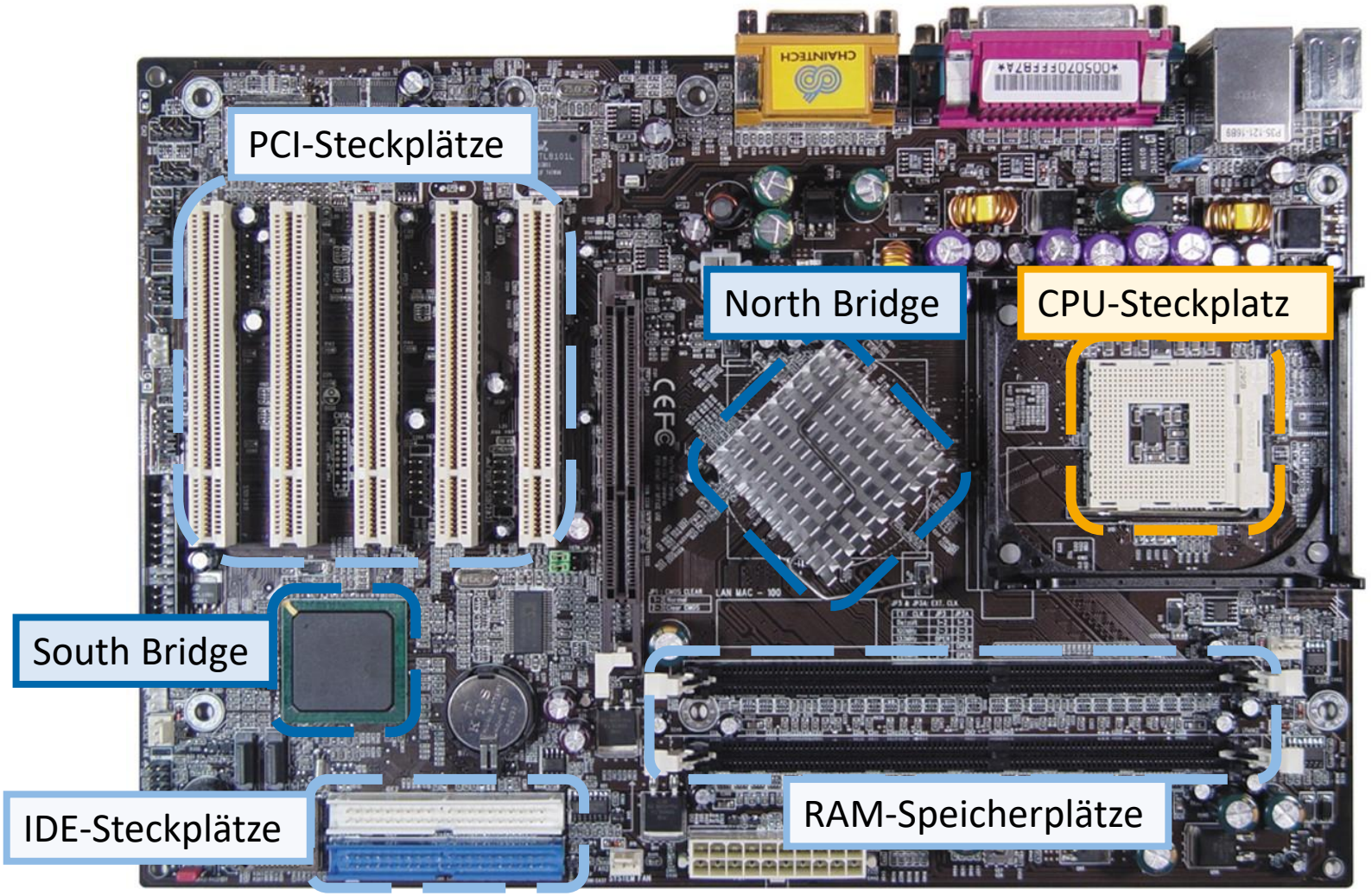
- Werden eingesetzt wenn
  - Geräte mit verschiedenen Geschwindigkeiten arbeiten
  - Synchrone und asynchrone Geräte vorhanden
- Schnittstellen zwischen den Bussen heißen Bridges



Beispiel: Busse eines Pentium 4-Systems

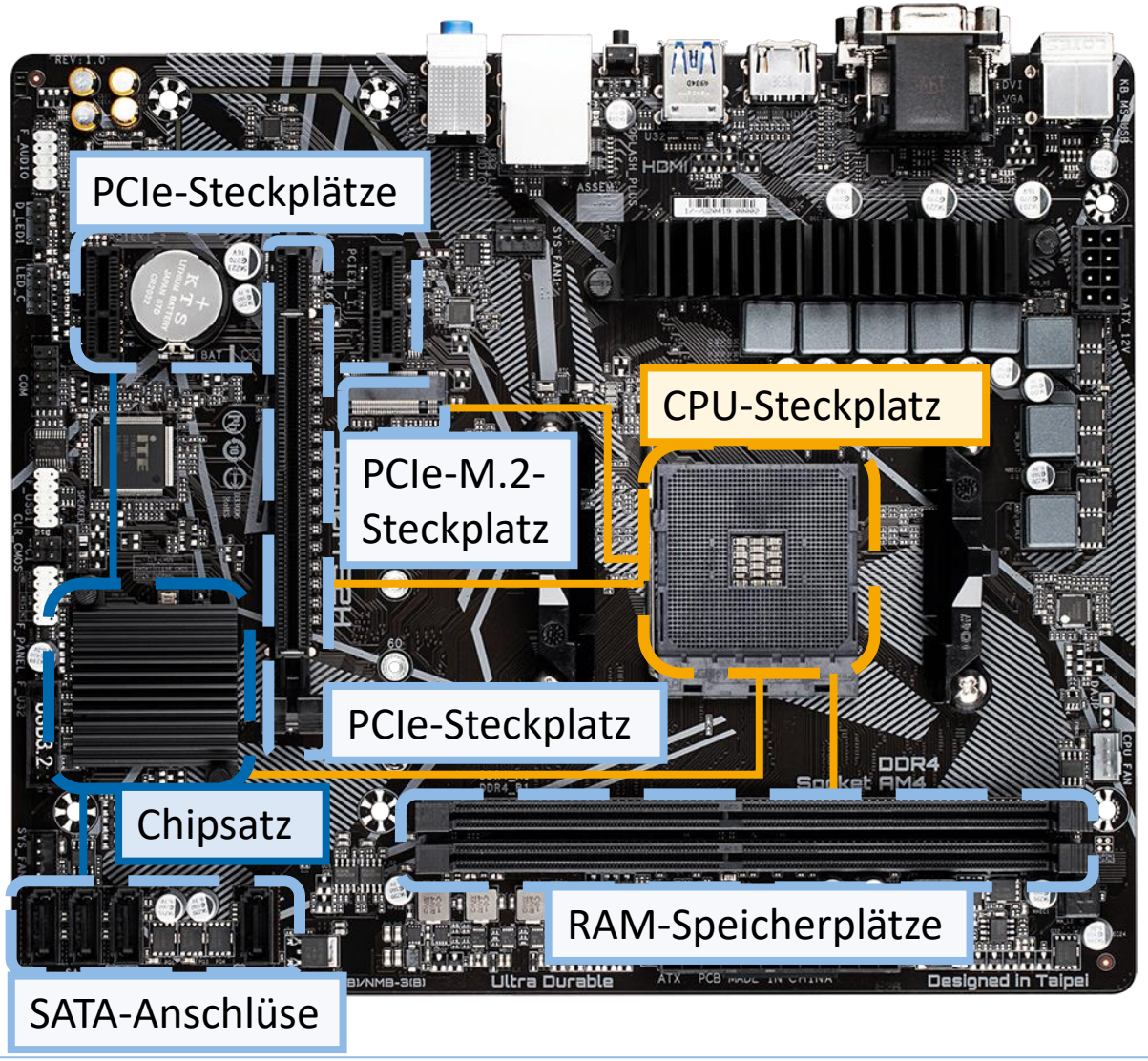


Beispiel: Busse eines Pentium 4-Systems



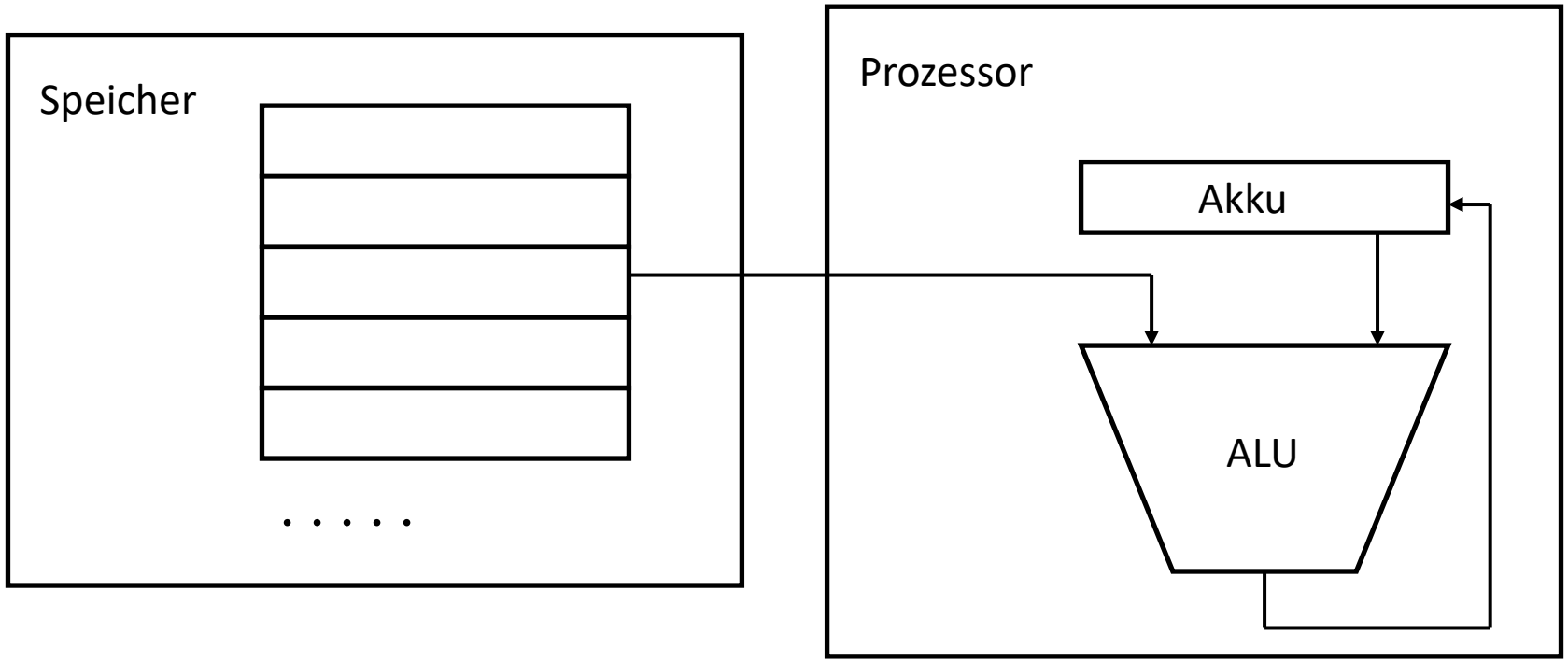
Exkurs: Moderne I/O Systeme (z.B. eines Zen 3- Systems)

- CPU enthält:
  - RAM Controller
  - PCIe Controller
  - SATA Controller
  - USB Controller
- PCIe:
  - Punkt zu Punkt
  - Höhere Datenraten
  - Geringere Latenz
- Chipsatz:
  - Über PCIe mit CPU verbunden
  - Fungiert als PCIe Switch
  - Enthält ebenfalls SATA/USB Controller



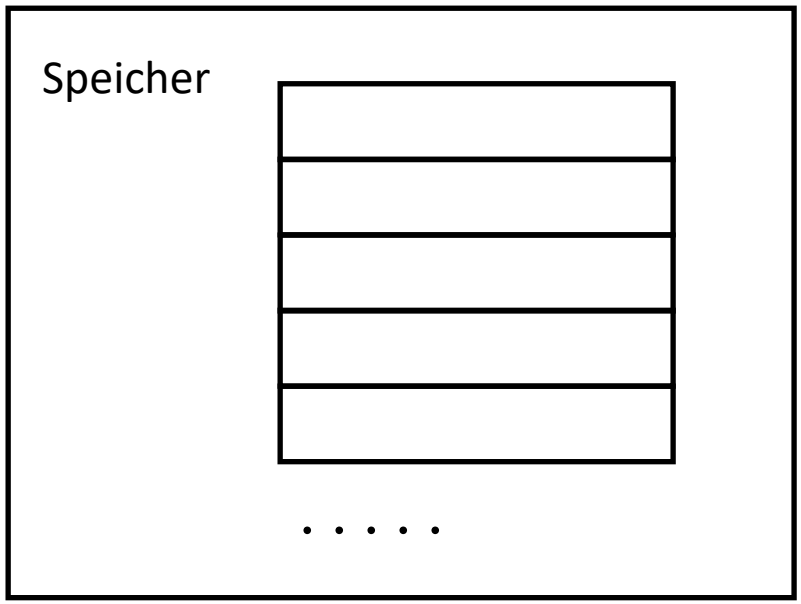
Klassifikation der von Neumann-Rechner

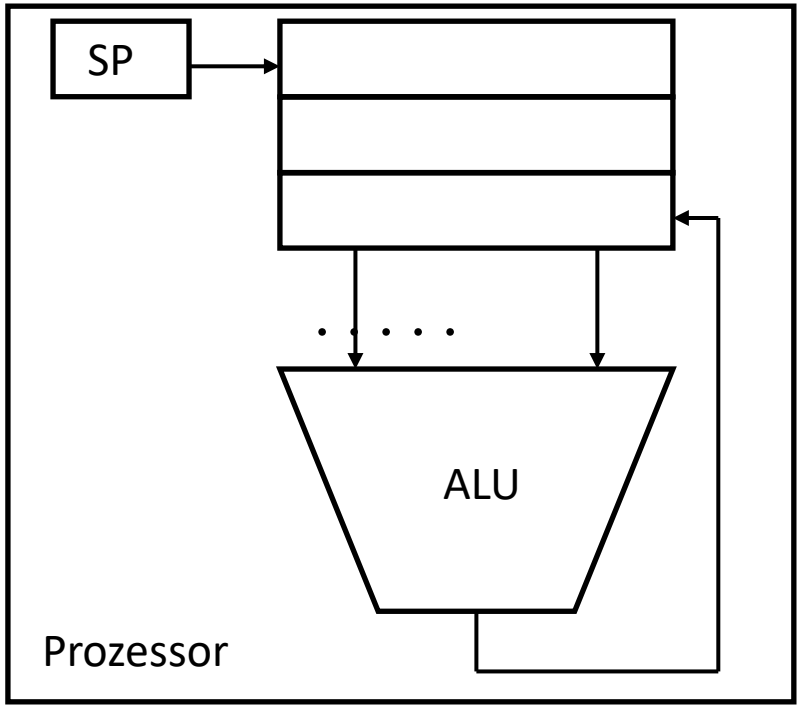
- **Speicherprogrammierbare Steuerung (SPS)**
- **Lokale Sicht**: Prozessor-Klassifikationen
  - Akkumulator-Architektur



Klassifikation der von Neumann-Rechner

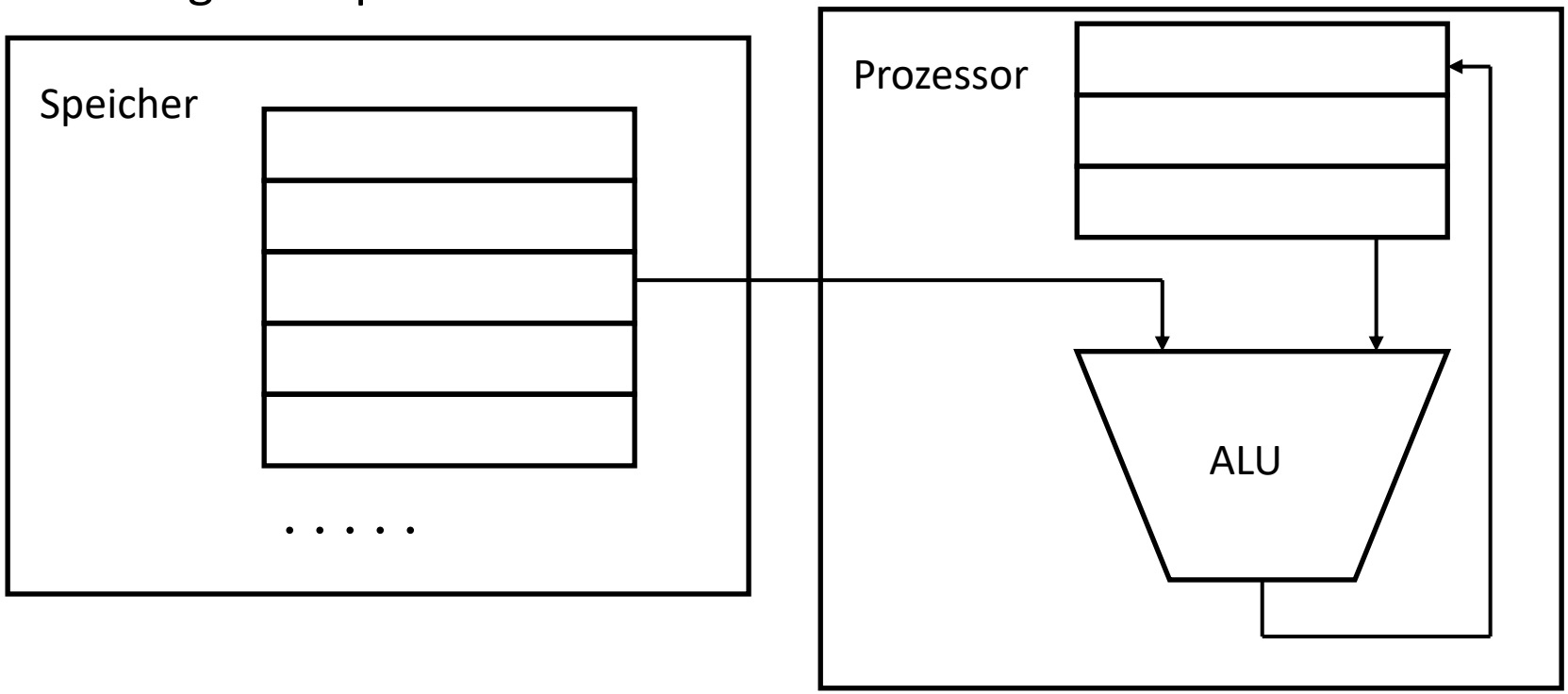
- **Umgekehrte polnische Notation (UPN)**
- **Lokale Sicht**: Prozessor-Klassifikationen
  - Stack-Architektur





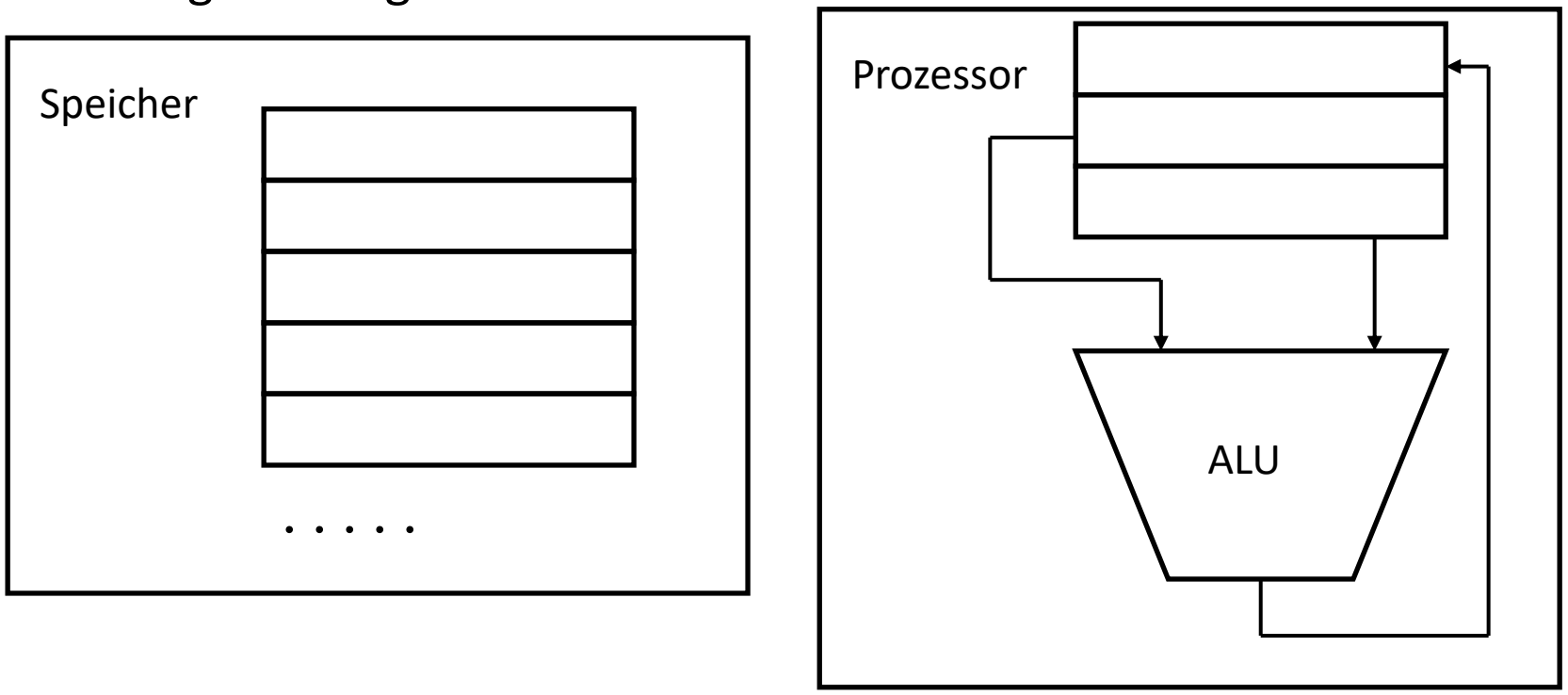
Klassifikation der von Neumann-Rechner

- **Complete Instruction Set Computer (CISC)**
- **Lokale Sicht**: Prozessor-Klassifikationen
  - Register-Speicher-Architektur



Klassifikation der von Neumann-Rechner

- **Reduced Instruction Set Computer (RISC)**
- **Lokale Sicht**: Prozessor-Klassifikationen
  - Register-Register-Architektur



Probleme der CISC-Prozessoren

1. Der von Neumannsche Flaschenhals verhindert, dass die Geschwindigkeit, mit welcher auf einen Speicher zugegriffen werden kann, mit der einer CPU vergleichbar ist.

2. Eine Reihe von Instruktionen bzw. Kombinationen von Instruktion und Adressierungsart wird nur in speziellen Anwendungen verwendet; dennoch muss der entsprechende Mikrocode vorgesehen werden.

3. Die Mikroprogrammierung des Steuerwerks ist langsamer als eine fest verdrahtete Steuerung.

Merkmale von RISC-Prozessoren

- Der Befehlssatz umfasst i.A. nur wenige Instruktionen sowie Adressierungsarten
- Durch Beschränkung auf elementare Grundfunktionen können die meisten Befehle innerhalb von einem Maschinen-Zyklus ausgeführt werden
- Auf den Hauptspeicher wird nur mit speziellen Load- und Store-Befehlen zugegriffen („Load/Store-Architektur“)
- Die Befehlsausführung wird unterstützt durch zusätzliche Hardware wie etwa eine große Anzahl von Registern
- Der Befehlscodierer bzw. das Steuerwerk ist fest verdrahtet
- Zur Unterstützung einer schnellen Befehls- und Operanden-Decodierung haben alle Instruktionen ein festes Format

Abschnitt 10.2

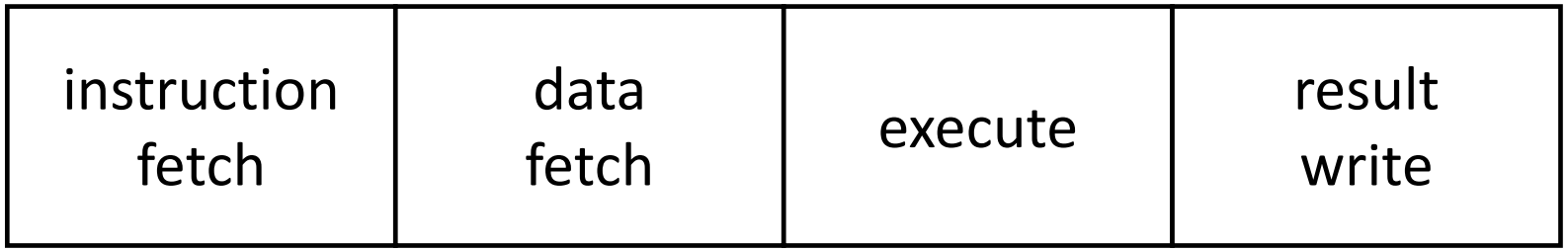
Alternativen und Optimierung

- Befehlsphasen-Pipelining
- Superskalar-Architektur
- Harvard-Architektur

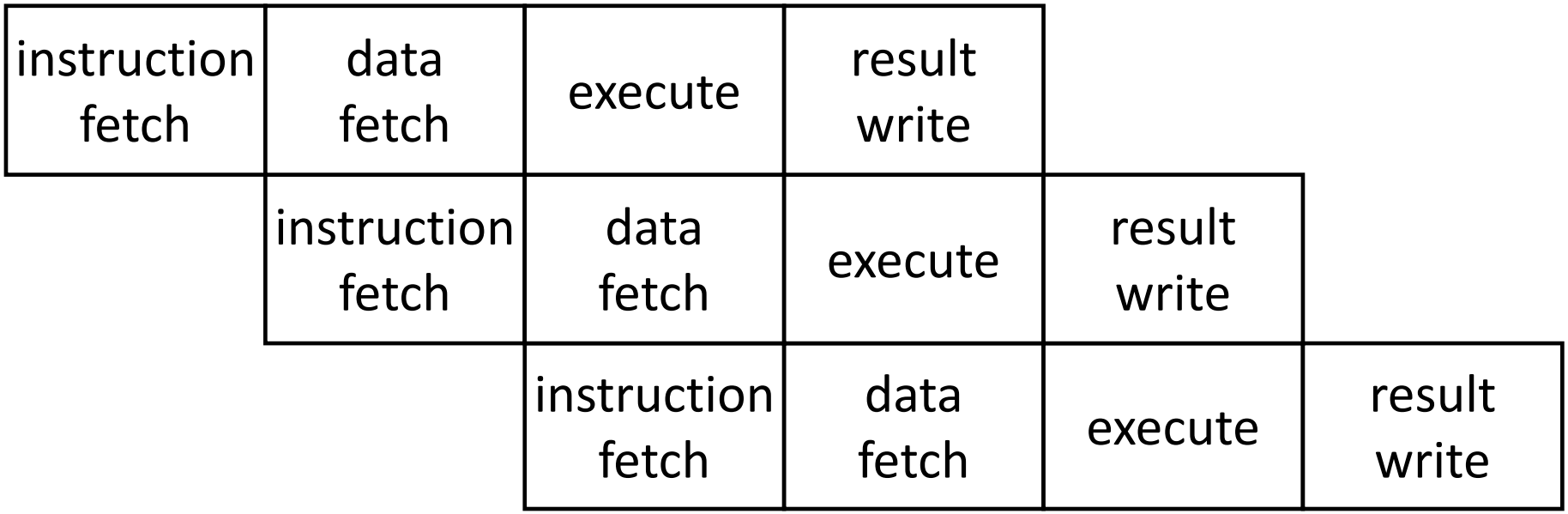
Optimierung

- Ziel: Erhöhung des Durchsatzes der CPU

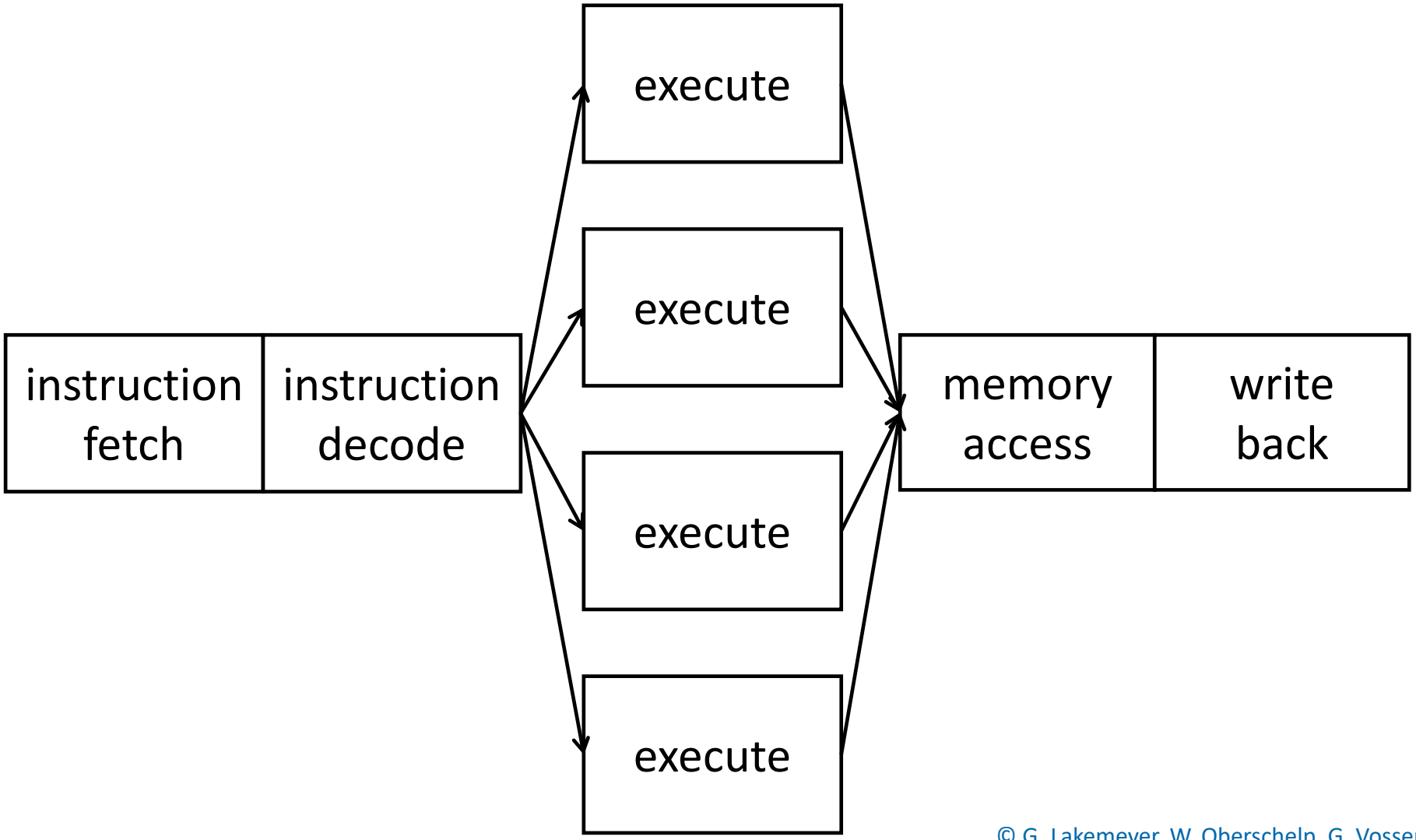
- Parallelität auf Instruktionsebene (ILP):



Einfaches Befehlsphasen-Pipelining



Superskalar-Architektur



Mögliche Alternativen zum von Neumann-Rechner

- Grundsätzlich sind folgende Kombinationen denkbar: (nach M. Flynn)
  - **Single Instruction - Single Data** **SISD**
    (von Neumann Rechner)
  - **Single Instruction - Multiple Data** **SIMD**
    (manchmal bei Parallelrechnern)
  - **Multiple Instruction - Single Data** **MISD**
    (keine Anwendung bekannt)
  - **Multiple Instruction - Multiple Data** **MIMD**
    (manchmal bei Parallelrechnern)

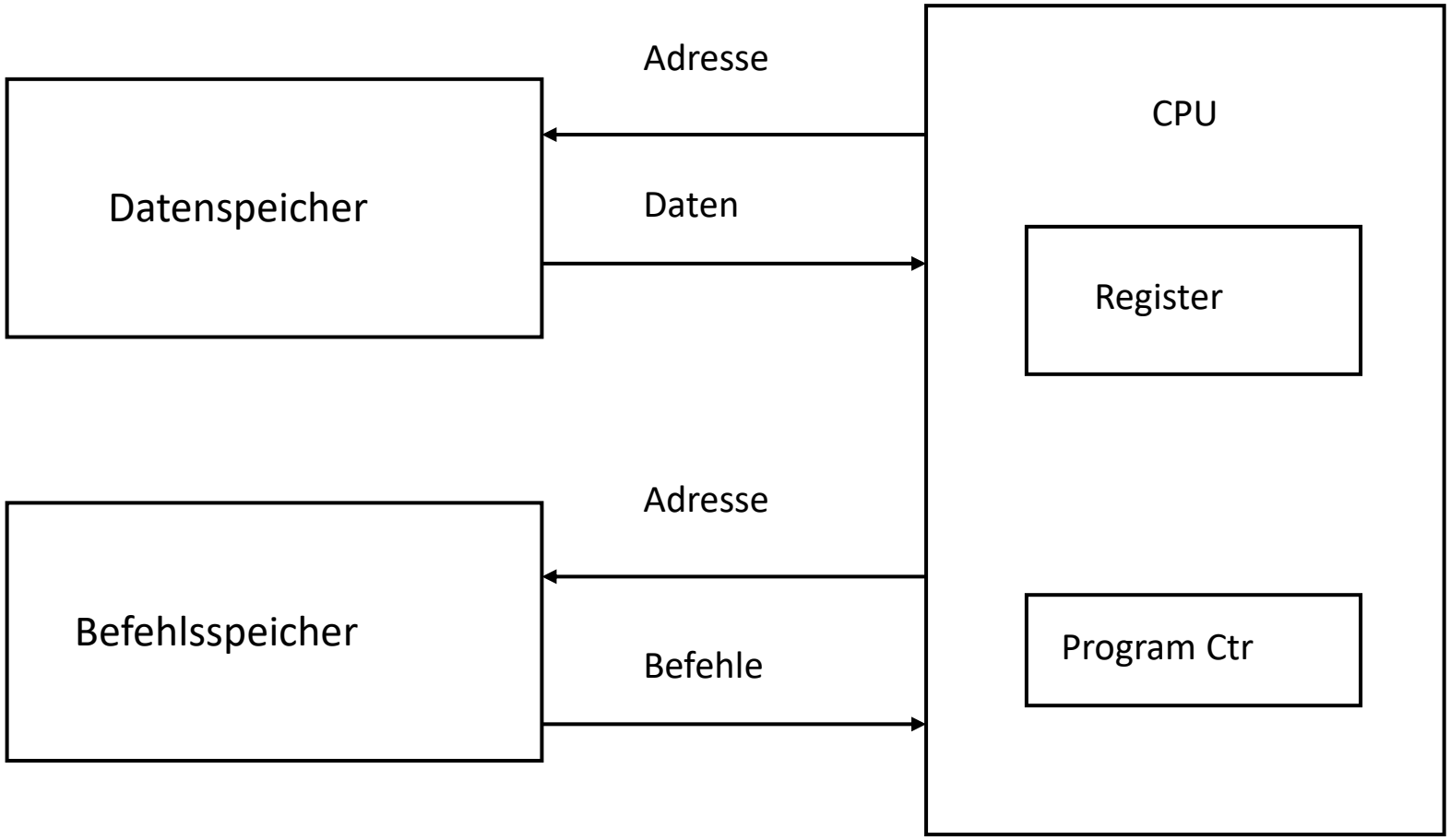
Harvard-Architektur

- Unterschied zur von Neumann-Architektur liegt in der Organisation von Befehls- und Datenspeicher

  - von Neumann-Architektur
    - Instruktionen und Daten teilen sich denselben Speicherbereich
    - Unterscheidung nur aus dem Kontext möglich

  - Harvard-Architektur
    - Instruktionen und Daten sind physisch getrennt
    - Werden über getrennte Busse angesteuert
    - Haben unabhängige Adressräume

Harvard-Architektur



Vor- und Nachteile der Harvard-Architektur

- Vorteile gegenüber von Neumann-Architektur
  - Schneller: Gleichzeitiges Laden von Befehlen und Daten
  - Sicherer: Programmcode weder versehentlich noch vorsätzlich überschreibbar
  - Optimiert: Datenwortbreite und Befehlswortbreite unabhängig voneinander

- Nachteile gegenüber von Neumann-Architektur
  - Nichtbenötigter Datenspeicher nicht als Programmspeicher nutzbar und umgekehrt

Anwendung der Harvard-Architektur

- Die meisten heutigen DSPs (Digitale Signalprozessoren) besitzen Harvard-Architektur
- Zahlreiche Microcontroller: z.B. AVR von Atmel und ARM

Variante der Harvard-Architektur

- Ein gemeinsamer Hauptspeicher
- Getrennte Befehls- und Daten-Caches mit separaten Busses, MMUs