

1. Betriebssysteme und Systemprogrammierung

[1.1. Einführung](#)

[1.2. Computer-Hardware: Ein Kurz-Überblick](#)

[1.3. Instruktionsarchitektur \(Instruction Set Architecture, ISA\)](#)

[1.4. Virtuelle Maschinen](#)

[1.5. Java und die Java Virtual Machine](#)

[1.6. Zusammenfassung \(Kapitel 1\)](#)

1.1. Einführung

Ein **Betriebssystem** ist die [Software](#), die die Verwendung (den Betrieb) eines [Computers](#) ermöglicht. Es verwaltet [Betriebsmittel](#) wie Speicher, Ein- und Ausgabegeräte und steuert die Ausführung von Programmen.

Betriebssystem heißt auf Englisch *operating system* (OS). Dieser englische Ausdruck kennzeichnet den Sinn und Zweck: Die in den Anfängen der Computer stark mit schematischen und fehlerträchtigen Arbeiten beschäftigten [Operatoren](#) schrieben [Programme](#), um sich die Arbeit zu erleichtern; diese wurden nach und nach zum *operating system* zusammengefasst.

Betriebssysteme bestehen in der Regel aus einem Kern (englisch: [Kernel](#)), der die [Hardware](#) des Computers verwaltet, sowie grundlegenden Systemprogrammen, die dem Start des Betriebssystems und dessen Konfiguration dienen. Unterschieden werden Einbenutzer- und [Mehrbenutzersysteme](#), Einzelprogramm- und Mehrprogrammsysteme, Stapelverarbeitungs- und Dialogsysteme. Betriebssysteme finden sich in fast allen Computern: als [Echtzeitbetriebssysteme](#) auf [Prozessrechnern](#), auf normalen [PCs](#) und als [Mehrprozessorsysteme](#) auf [Hosts](#) und [Großrechnern](#).

Wikipedia, 5.2.2008

1.1. Einführung

Ein **Betriebssystem**, auch **OS** (von englisch *operating system*) genannt, ist eine Zusammenstellung von Computerprogrammen, die die **Systemressourcen** eines Computers wie Arbeitsspeicher, Festplatten, Ein- und Ausgabegeräte **verwaltet** und diese Anwendungsprogrammen **zur Verfügung stellt**. Das Betriebssystem bildet dadurch die **Schnittstelle zwischen den Hardware-Komponenten und der Anwendungssoftware** des Benutzers.

Betriebssysteme bestehen in der Regel aus einem **Kernel** (deutsch: Kern), **der die Hardware des Computers verwaltet**, sowie speziellen Programmen, die beim Start unterschiedliche Aufgaben übernehmen. Zu diesen Aufgaben gehört unter anderem das Laden von Gerätetreibern.

Betriebssysteme finden sich in fast allen Arten von Computern: Als Echtzeitbetriebssysteme auf Prozessrechnern und Eingebetteten Systemen, auf Personal Computern, Tabletcomputern, Smartphones und auf größeren Mehrprozessorsystemen wie z. B. Servern und Großrechnern.

Die Aufgaben eines Betriebssystems lassen sich wie folgt zusammenfassen:

- Benutzerkommunikation;
- Laden, Ausführen, Unterbrechen und Beenden von Programmen;
- Verwaltung und Zuteilung der Prozessorzeit;
- Verwaltung des internen Speicherplatzes für Anwendungen;
- Verwaltung und Betrieb der angeschlossenen Geräte;
- Schutzfunktionen z. B. durch Zugriffsbeschränkungen.

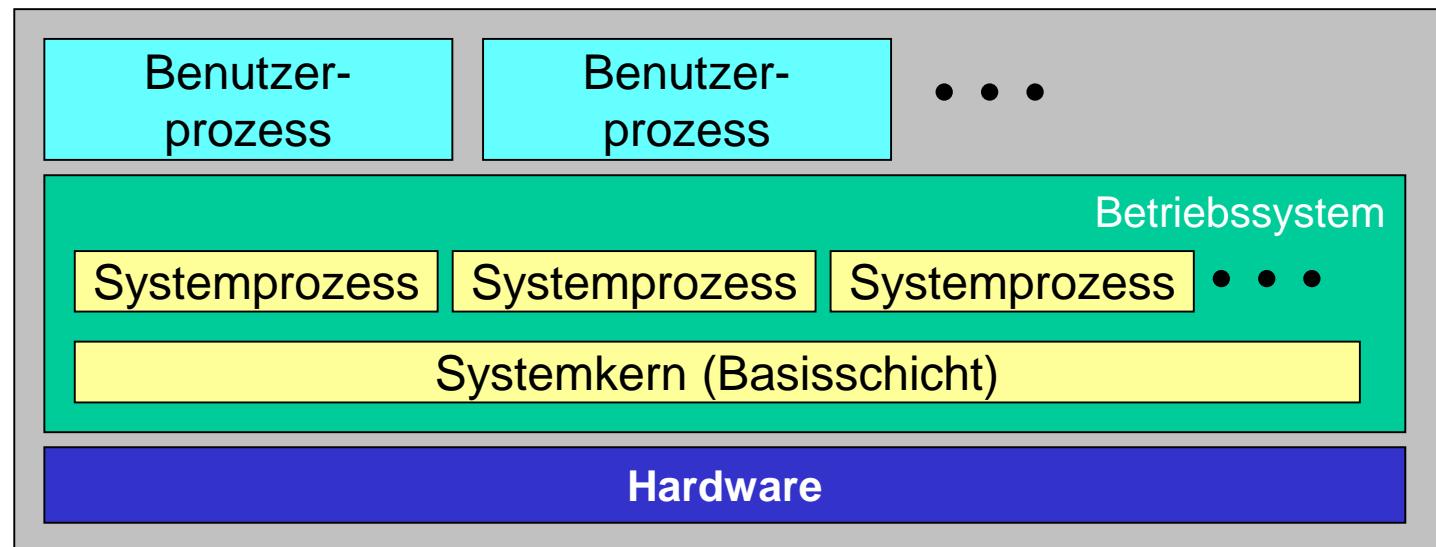
Die Gewichtung zwischen diesen Aufgaben wandelte sich im Laufe der Zeit, **insbesondere wird Schutzfunktionen wie dem Speicherschutz oder begrenzten Benutzerrechten heute eine höhere Bedeutung zugemessen als noch in den 1990er Jahren**. Dies macht Systeme allgemein robuster, reduziert z. B. die Zahl der Programm- und Systemabstürze und macht das System auch stabiler gegen Angriffe von außen, etwa durch Computerviren.

Wikipedia, 11.4.2020

Das Betriebssystem (Operating System, OS)

Zentrales Konzept aller Betriebssysteme ist der „**Prozess**“, eine **Abstraktion eines in Ausführung befindlichen Programms**.

Der Prozessbegriff wird in einem nachfolgenden Kapitel noch detailliert behandelt.



Das Betriebssystem als Teil eines Schichtenmodells

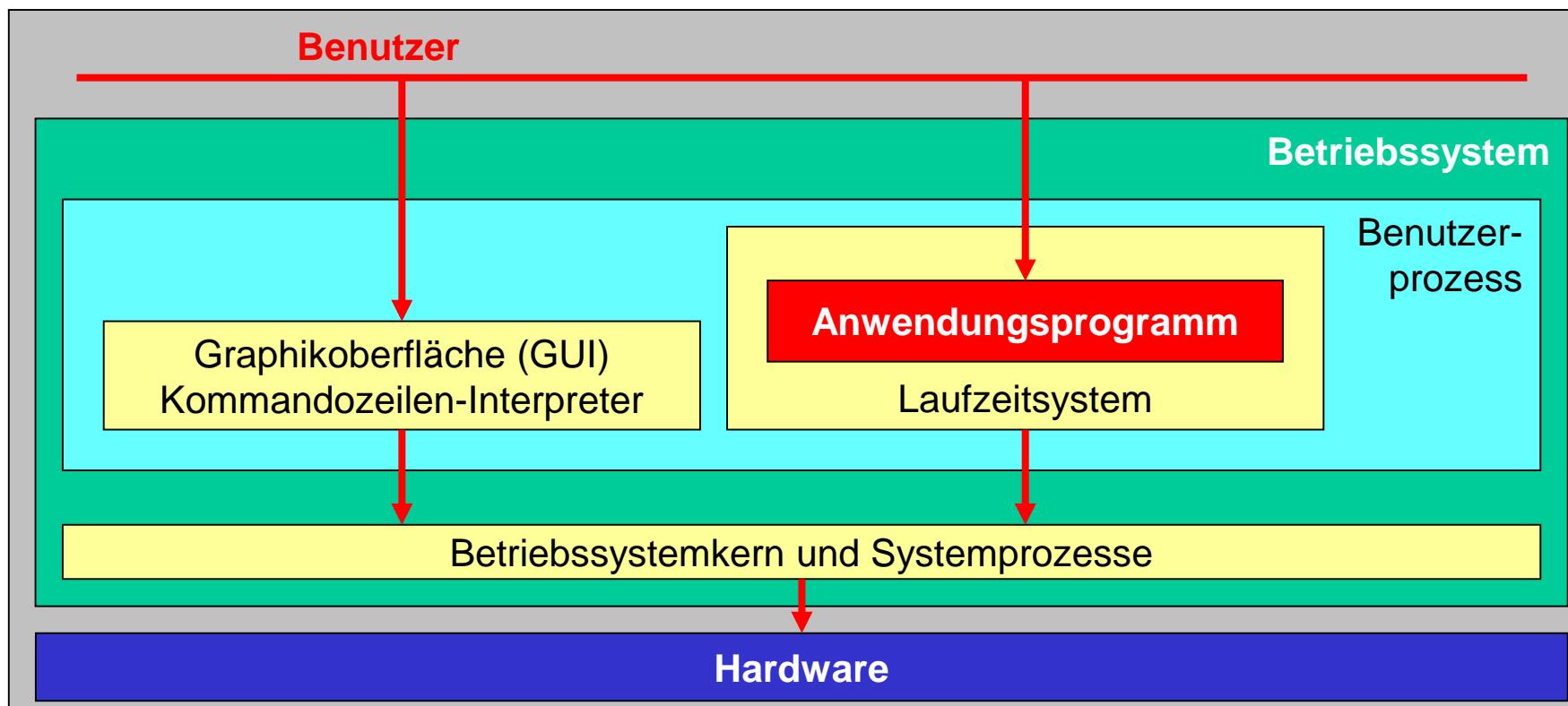
Das Betriebssystem stellt einen von der konkreten Hardware (weitgehend) unabhängigen „Lebensraum für Benutzerprozesse“ bereit.

Direkte Kommunikation mit dem Benutzer:

Reaktion auf Kommandos über GUI bzw. Kommandozeile

Indirekte Kommunikation mit dem Benutzer:

Verwaltung der Anwendungsprogramme



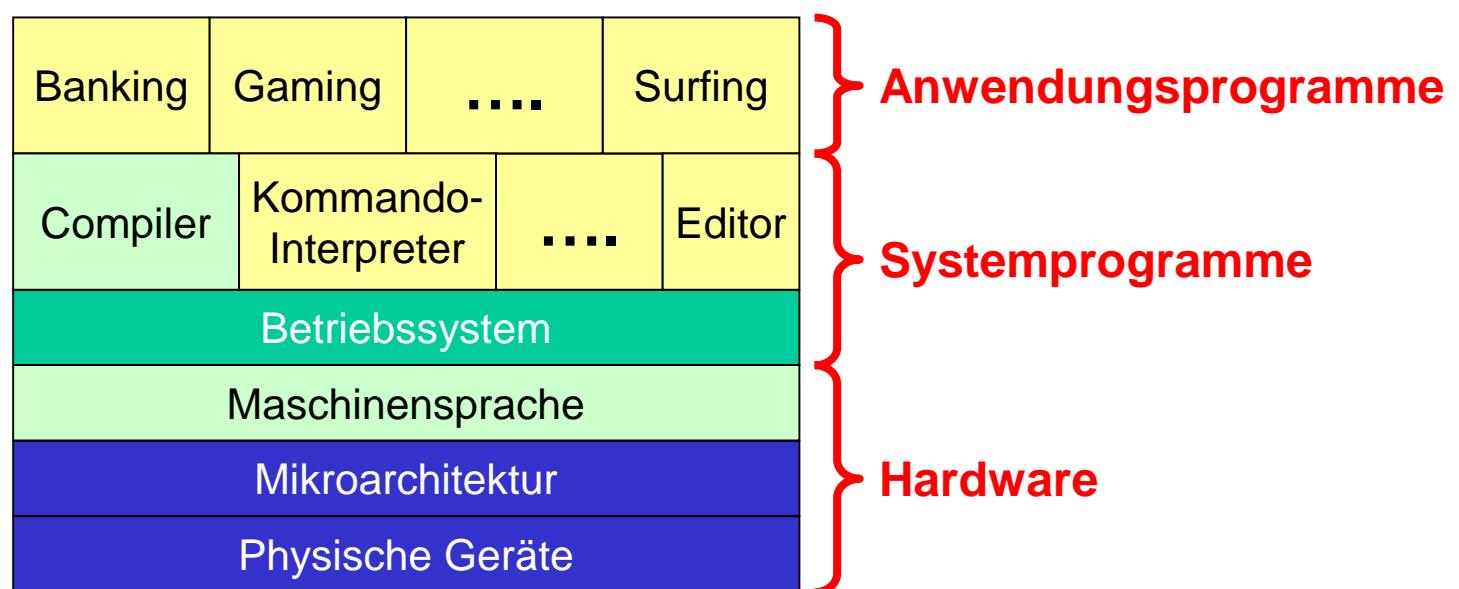
Betriebssystem: Definition gemäß DIN 44300

Betriebssystem:

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.

DIN 44300

Das Betriebssystem stellt (gemeinsam mit weiteren „Systemprogrammen“) eine „**standardisierte Schnittstelle** für die (weitgehend) geräteunabhängigen Anwendungsprogramme bereit.



Die wichtigsten Aufgaben von Betriebssystemen

Verfügbarmachung der angeschlossenen und installierten Hardware

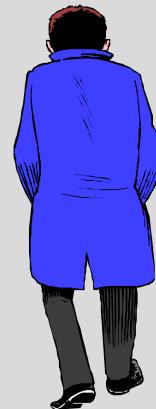
- Laden aller Dateien („**Treiber**“), die zum Betreiben der Hardware-Elemente (Graphikkarte, Bildschirm, Tastatur, Maus, Laufwerke, Netzanschluss, ...) benötigt werden.
- Kaspelung der technischen Details: „**Virtuelle Betriebsmittel**“



Koordination und Zuteilung von Betriebsmitteln

Zuteilung („**Scheduling**“) von

- **Prozessorkapazität** (Zeit, Speicher)
- Platz im **Hauptspeicher**
- Platz in **sonstigen Speichern** (Festplatte, Speichernetz, ...)
- **Ein- und Ausgabegeräte** (z.B. Drucker)



Steuerung und Kontrolle des Datenflusses

- **Laden und Kontrollieren** von Anwendungsprogrammen
- Weitergabe der **Benutzereingaben**
- Behandlung der **Fehler**
- Verwaltung der **Benutzerrechte**



Hochgradig
sicherheits-
relevant

Die wichtigsten Aufgaben von Betriebssystemen (2)

Bereitstellung von Dienstprogrammen

- Zeicheneingabe (Editore), Zeichnen, Rechnen, ...
- Datensicherung
- Telekommunikation



Organisation und Verwaltung des Dateisystems

- Organisationsstruktur, Kataloge / Verzeichnisse
- Anlegen und Löschen von Dateien



Schnittstelle für Anwendungsprogramme

- Application Programming Interface (API)



Benutzerschnittstelle

- GUI bzw. Kommando-Interpreter



**Fast alle Aufgaben des Betriebssystems
sind stark sicherheitsrelevant!**



Systemprogrammierung

Als **Systemprogrammierung** bezeichnet man das Erstellen von **Softwarekomponenten**, die Teil des **Betriebssystems** sind oder die eng mit dem Betriebssystem bzw. mit der darunter liegenden **Hardware** kommunizieren müssen. ...

Systemnahe Software dient als Abstraktionsschicht zwischen einer **Applikation** und dem Betriebssystem. Diese Schicht erleichtert den Zugriff auf die sehr einfach gehaltenen Betriebssystemfunktionen. Aus **Performance-** und **Sicherheitsgründen** ist der Zugriff auf das Betriebssystem auf das Notwendige beschränkt. Der Programmierer muss sich selbst um Synchronisation (z.B. mittels **Semaphore** oder **Shared Memory**) und **Interprozesskommunikation** kümmern. Die Programmierung auf Betriebssystemebene ist dadurch umständlich und fehleranfällig.

Im Gegensatz dazu bevorzugen Applikationsentwickler Schnittstellen, die schnell, fehlertolerant und leicht zu verwenden sind. Das heißt, für den Applikationsprogrammierer steht die Funktionalität im Vordergrund, während ein Systemprogrammierer Aspekte wie Effizienz und Robustheit besonders berücksichtigen muss. Alle Hochsprachen kapseln die Funktionen der Systemprogrammierung, was eine effektive **Anwendungsprogrammierung** mit hohem **Abstraktionsgrad** ermöglicht, jedoch im Bereich der systemnahen Programmierung nicht zielführend ist.

...

Systemprogrammierung wird häufig Sprachen wie **C**, **C++** oder **Assembler** durchgeführt. Hochsprachen wie **Pascal** führen häufig zu großen und wenig effizienten Programmen, die für die systemnahe Programmierung ungeeignet sind. Ausnahmen bilden hier einige **eingebettete Systeme**, die direkt in **Java** programmiert werden können.

C ist zwar selbst auch eine Hochsprache, bietet aber die Möglichkeit, **Assembler-Befehle über Inline-Assembler direkt ins Hochsprachenprogramm einzubinden** und z. B. Variablen direkt mit ihren symbolischen Namen abzufragen, die sonst umständlich über den Stapspeicher übergeben werden müssten.

Wikipedia, 11.4.2020

1.2. Computer-Hardware: Ein Kurz-Überblick

Die größte Verbreitung haben heute Digitalrechner mit einer Architektur, die (zumindest in wesentlichen Teilen) dem Modell folgt, das John von Neumann zurückgeht.

[1.2.1. Der Von-Neumann-Rechner](#)

[1.2.2. „BORIS“: Ein gedachter Mikroprozessor](#)

[1.2.3. Von CISC und RISC](#)

[1.2.4. Pipelining, Super-Pipelining, Superskalar-Technik](#)

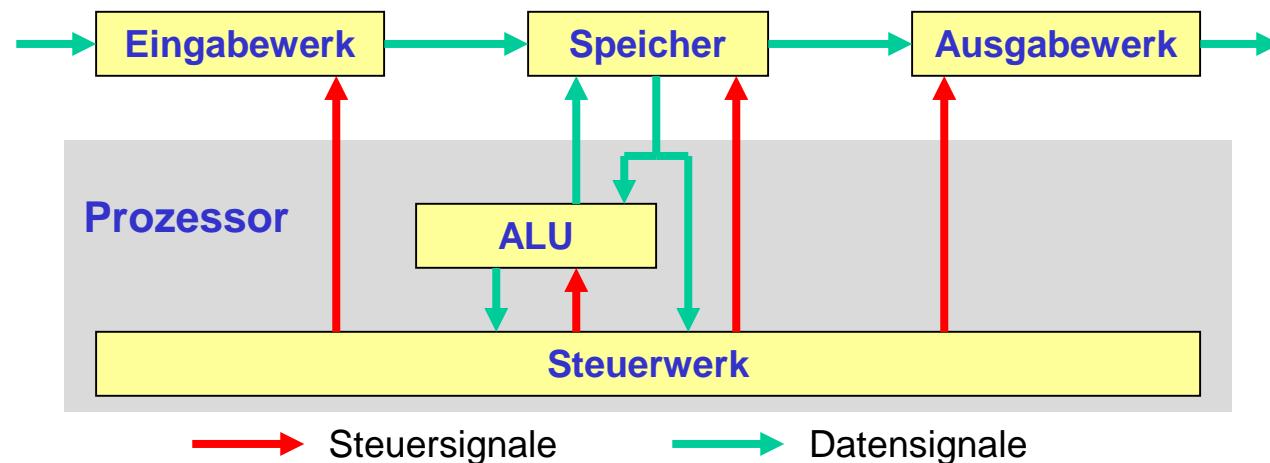
[1.2.5. Aktuelle Entwicklungen](#)

[1.2.6. Computernetze und Verteilte Systeme](#)

1.2.1. Der Von-Neumann-Rechner

Der Von-Neumann-Rechner besteht aus fünf Funktionseinheiten.

- **Steuerwerk** (control unit)
Laden und Decodieren der Programmbefehle, Koordinieren der Befehlsausführung.
- **Arithmetisch-logische Einheit** (arithmetic logical unit, ALU)
Ausführung arithmetischer und logischer Operationen unter Kontrolle des Steuerwerks;
Bereitstellung der Operanden durch das Steuerwerk.
- **Speicher** (memory)
Einteilung des Speichers in fortlaufend nummerierte, gleich große Zellen
Zugriff (Lesen/Schreiben) auf Zellinhalte über ihre Nummer (Adresse)
- **Eingabewerk**
- **Ausgabewerk**



Weitere Charakteristika des Von-Neumann-Rechners

Die Struktur ist problemunabhängig.

- Derselbe Rechner ist für die **Behandlung unterschiedlichster Probleme** einsetzbar.
- Hierzu benötigt er ein von außen eingegebenes Programm.
- **Ohne Programm** ist der Von-Neumann-Rechner **nicht arbeitsfähig**.

Daten und Programme liegen [binär codiert] in demselben Speicher.

- Ob es sich beim Inhalt von Speicherzellen um **Programmanweisungen oder** um **Daten** handelt, ist nur **aus dem Programmablauf erkennbar**.
- **Aufeinanderfolgende Befehle** werden **in aufeinanderfolgende Speicherzellen** gelegt. Das Steuerwerk kann daher den **nächsten Befehl durch Inkrementieren eines Befehlszählers** (Program Counter, PC) ansprechen.
- Durch **Sprungbefehle** kann von der sequentiellen Bearbeitung der Befehle abgewichen werden.

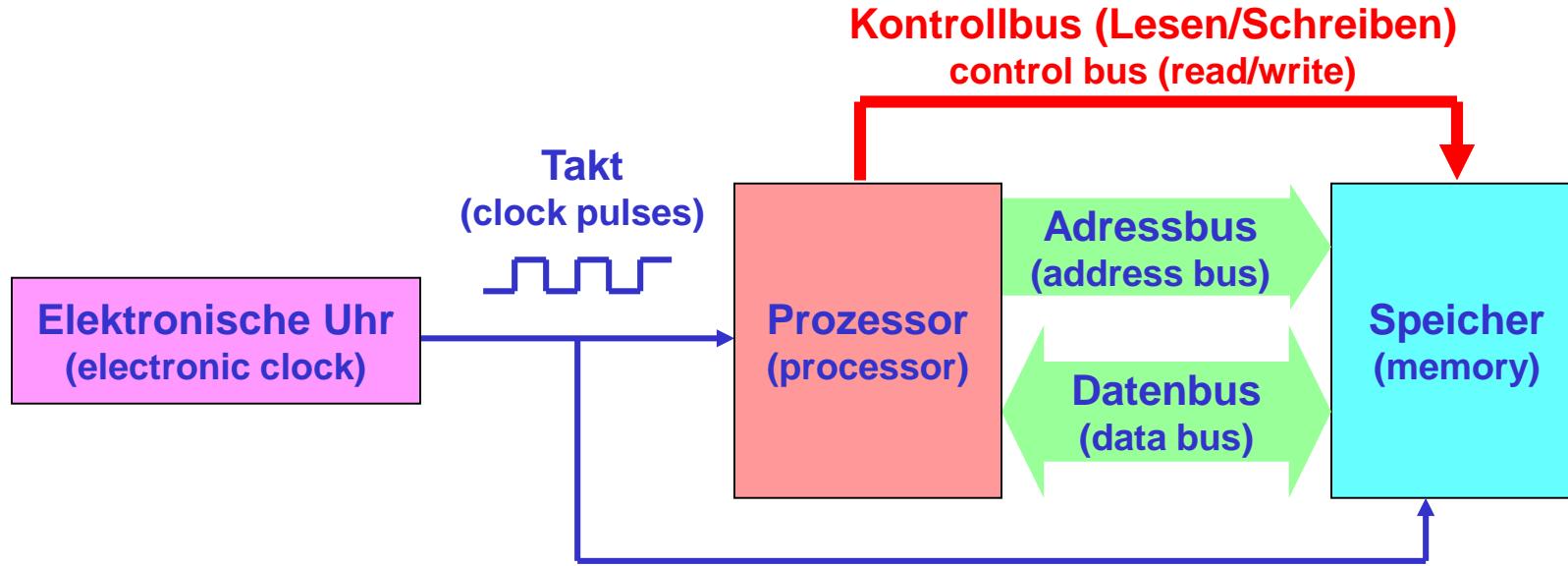
Die Architektur fällt in die Klasse SISD (Single Instruction, Single Data).

- **Es gibt nur einen Prozessor**, bestehend aus Steuerwerk und arithmetisch-logischer Einheit.
- Dieser bearbeitet einzeln nacheinander die auszuführenden Befehle.

Der Datentransport nimmt eine zentrale Stellung ein.

- Befehle, Operanden, Ergebnisse werden zwischen dem Speicher und dem Prozessor auf einem einzigen Datenpfad: „**Von-Neumann-Flaschenhals**“.

Wichtige Komponenten eines sehr einfachen (Von-Neumann-) Rechners



Der Prozessor (Mikroprozessor, Central Processing Unit, CPU)

- beinhaltet ALU und Steuerwerk,
- holt Befehle aus dem Speicher, decodiert sie und führt sie aus (fetch-decode-execute),
- holt Daten aus dem Speicher und legt Daten im Speicher ab.

Die elektronische Uhr

- legt fest, wann der Prozessor das nächste tut (nächsten Befehl holen oder ausführen),
- wird realisiert als 0101...-Folge, wobei die 0-1- oder 1-0-Wechsel den Takt angeben,
- hat großen Einfluss auf die Leistungsfähigkeit des Rechners.

Der Adressbus

- umfasst Leitungen zur Angabe der gewünschten Zelle für den aktuellen Speicherzugriff.

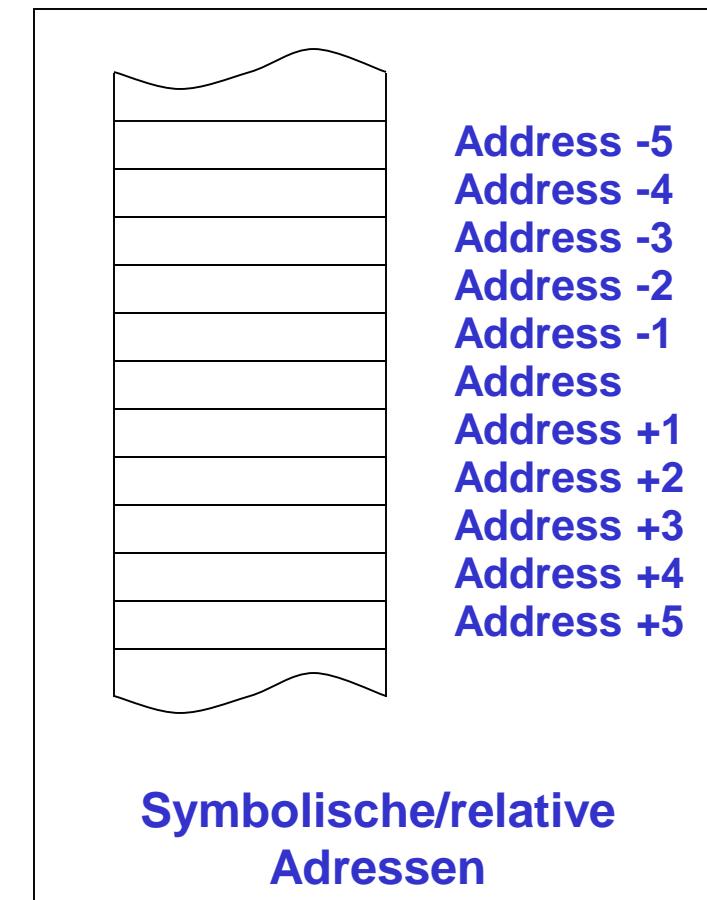
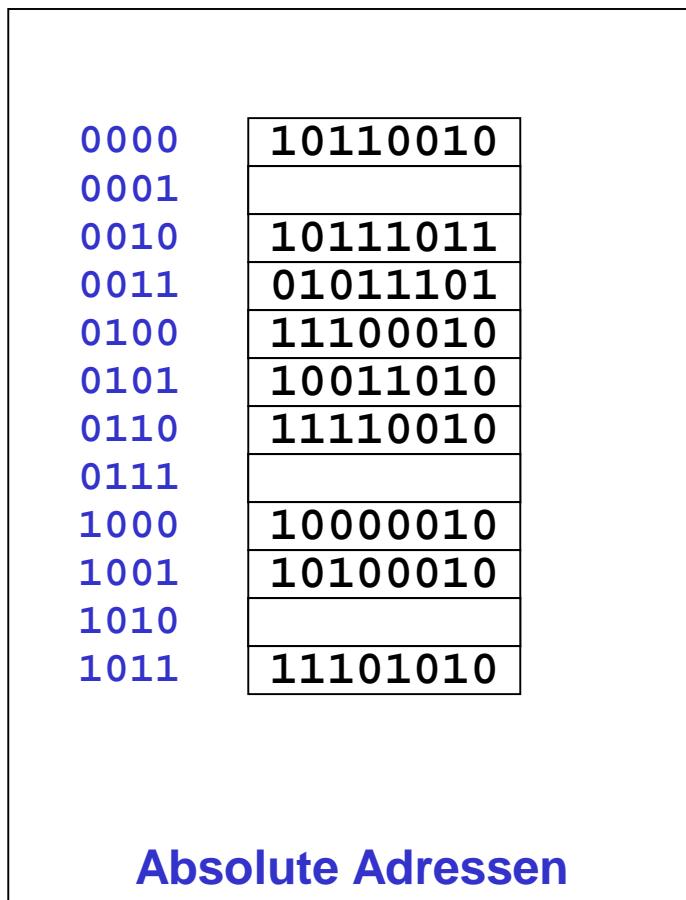
Der Datenbus

- ist eine Sammelleitung zur (bit-parallelen) Übertragung von Daten bzw. Befehlen.

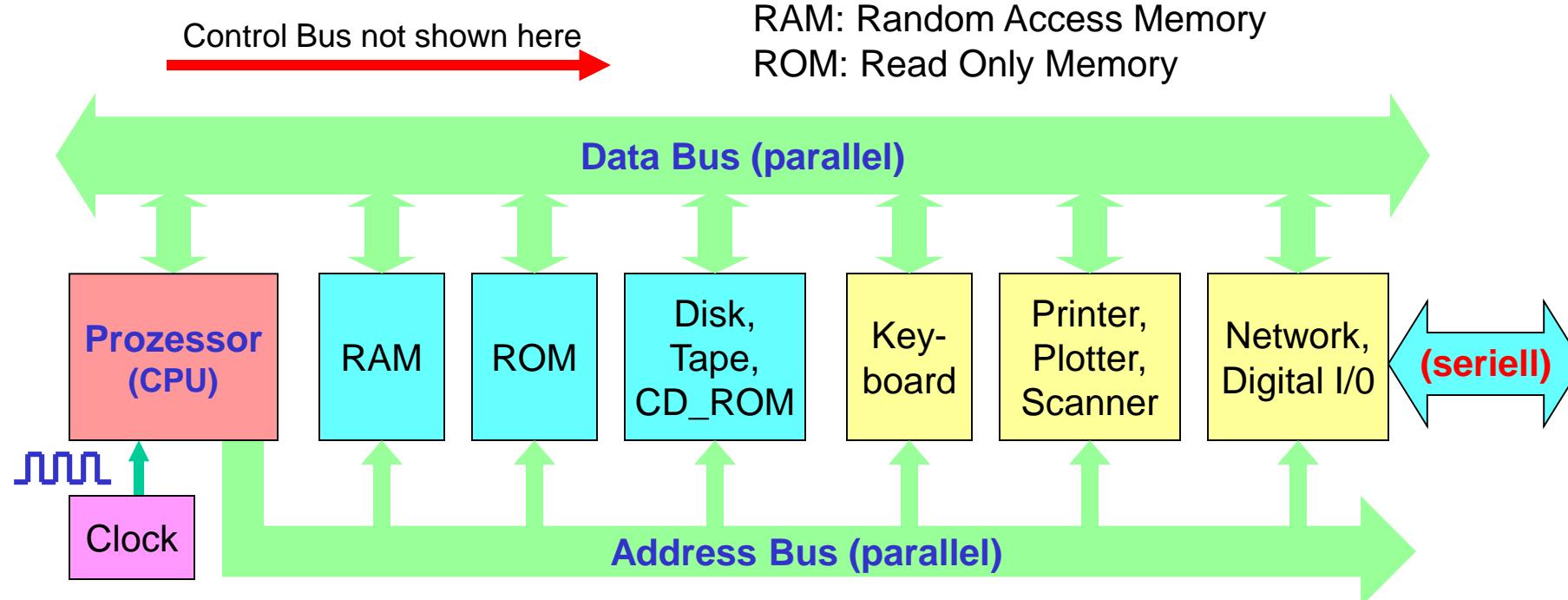
Der Speicher

Der Speicher ist unterteilt in „**Zellen**“, denen jeweils eine Adresse zugeordnet ist.
Bei Angabe ihrer Adresse kann auf den aktuellen Inhalt der Zelle zugegriffen werden.

Da der Speicher häufig sehr groß ist, wird oft eine **Speicherdarstellung mit symbolischen und relativen Adressen** verwendet.



Eine realitätsnähere Rechnerarchitektur



Hardware-Zugriffe sind für die CPU „transparent“:

- Den Hardware-Einheiten (Keyboard, Printer, ...) wird ein **Abbild im Speicher zugeordnet**.
- Durch **enge Kooperation von Hardware und „System-Software“** wird sichergestellt, dass die Daten an den richtigen Platz gelangen bzw. von dort geholt werden.
- **Hardware-Zugriffe stellen sich** so für die CPU **als Speicherzugriffe dar**.

1.2.2. „BORIS“: Ein gedachter Mikroprozessor

BORIS =

*B*eginner’s
*O*ptimized
*R*educed
*I*nstruction
Set Microprocessor

[1.2.2.1. Einbindung von BORIS in ein Gesamtsystem](#)

[1.2.2.2. Interne Struktur von BORIS](#)

[1.2.2.3. Ein einfaches Programm und dessen Bearbeitung](#)

BORIS ist ein [gedachter] Mikroprozessor, der

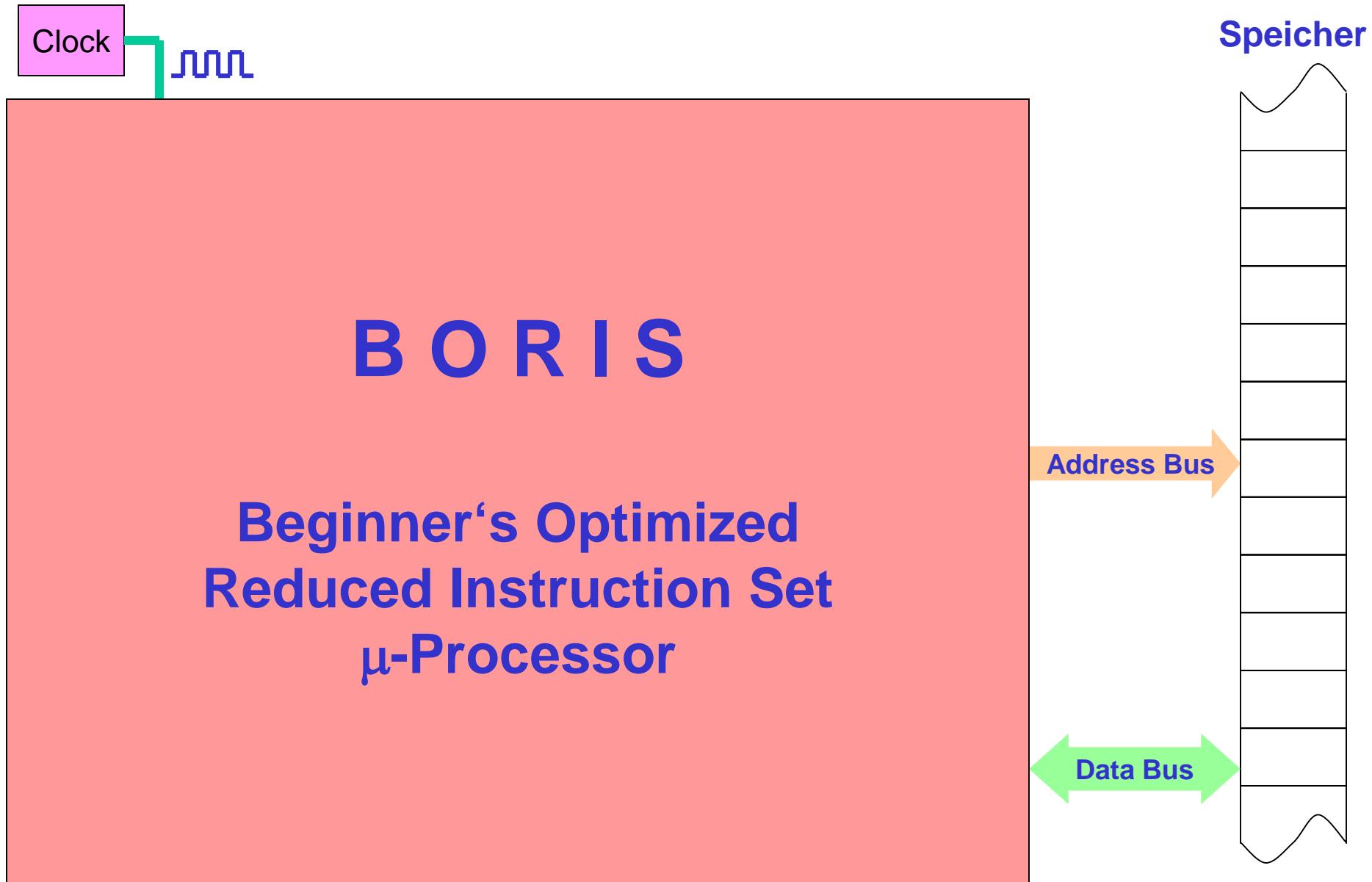
- nur drei Befehle kennt: **LOAD, ADD und STORE**,
- zeigt, wie ein **sehr einfacher Prozessor** ein **sehr einfaches Programm bearbeitet**,
- das **zu bearbeitende Programm und die zugehörigen Daten im Speicher** vorgegeben bekommt.

BORIS beinhaltet - wie wirkliche Mikroprozessoren - einige Speicherplätze. Diese sog. „**Register**“ sind Speicher mit extrem kurzen Zugriffszeiten, in denen aktuelle Werte

- **zwischengespeichert** und
- **bearbeitet** werden.

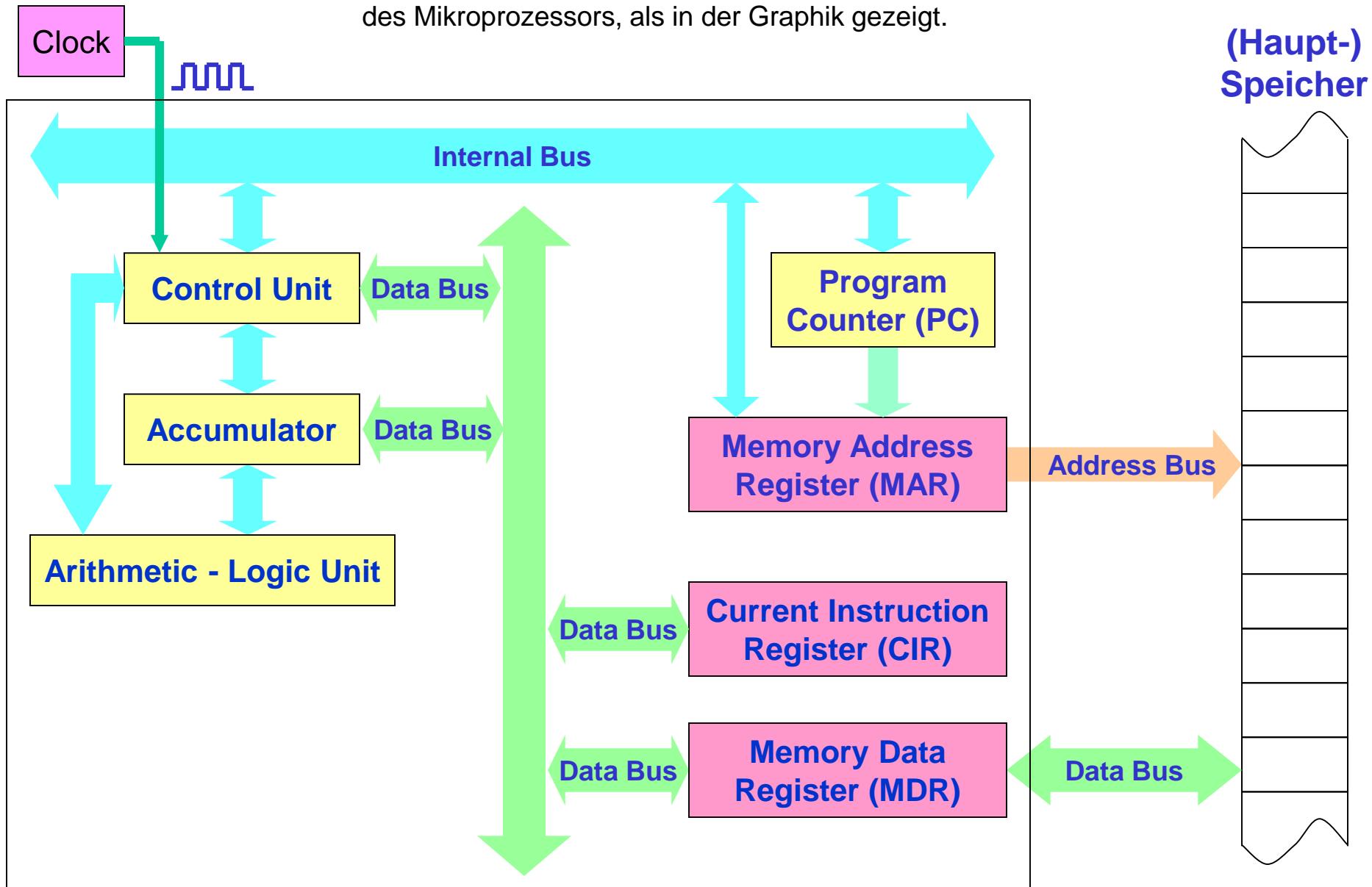
Reale Mikroprozessoren beinhalten zusätzlich **auch „Flags“** (spezielle 1-Bit-Register) zur Speicherung von Statusinformation.

1.2.2.1. Einbindung von BORIS in ein Gesamtsystem



1.2.2.2. Interne Struktur von BORIS

Die Kontrolleinheit hat mehr Verbindungen zu den Komponenten des Mikroprozessors, als in der Graphik gezeigt.



Interne Struktur von BORIS (2)

Der Akkumulator (Accumulator)

- ist ein **Schnellspeicher mit sehr kurzer Zugriffszeit**,
- in dem ein **Operand** vor Ausführung einer Operation **bereitgestellt** wird und
- **(Zwischen-) Ergebnisse gespeichert** werden,
- um anschließend **weiterbearbeitet**
- **oder zum Speicher transferiert** zu werden.

Die Arithmetisch-Logische Einheit (Arithmetic Logic Unit, ALU)

ist das „Gehirn“ des Mikroprozessors. Hier werden insbesondere

- „**einfache**“ arithmetische bzw. logische Operationen ausgeführt, deren
- **Ergebnis im Akkumulator abgelegt** wird.

Die Kontrolleinheit (Control Unit)

unterliegt dem **Takt** der Uhr, **steuert** aber ihrerseits **den gesamten µ-Prozessor**:

- **Datentransfer**: Welche Daten müssen wann wohin?
- **ALU-Steuerung**: Wann ist welche Operation auszuführen?
- **Synchronisation** des Gesamtsystems

Interne Struktur von BORIS (3)

Der Programmzähler (Program Counter, PC)

stellt sicher, dass die Befehle in der richtigen Reihenfolge ausgeführt werden.
Da es bei BORIS keine Programmsprünge gibt, wird der Programmzähler

- mit der **Speicheradresse des Programmanfangs** initialisiert und
- nach dem Laden eines Befehls **jeweils um 1 erhöht**
(zur Speicherzelle mit dem nächsten Befehl).

Das Befehlsregister (Current Instruction Register)

ist das Register, in dem der **zuletzt aus dem Speicher geholte Befehl** liegt.

Anmerkung:

Einem 0-1-Wort ist nicht „anzusehen“, ob es sich um Programmcode oder um Daten handelt. Zeigt der **Programmzähler fehlerhafterweise auf** eine Speicherzelle, die **Daten** enthält, dann

- wird deren Inhalt ins Befehlsregister geladen und der Computer versucht,
- die **Daten als Programm zu interpretieren** und
- dieses „Programm“ zu bearbeiten.

Interne Struktur von BORIS (4)

Das Datenregister (Memory Data Register)

enthält die **Daten bzw.** den **Befehl**, die/der **zuletzt** aus dem Speicher **gelesen** wurde(n) **oder** dorthin **geschrieben** wurde(n) **bzw.** **werden soll**(en).

Ein Befehl wird vor seiner Ausführung vom Memory Data Register zum Befehlsregister geleitet, damit er dort bearbeitet werden kann.

Das Adressregister (Memory Address Register)

enthält die **Adresse der Daten /des Befehls**, auf die/den aktuell zugegriffen wurde bzw. werden soll.

1.2.2.3. Ein einfaches Programm und dessen Bearbeitung

BORIS soll nun ein einfaches Programm ausführen, das **ab Position 100 im Speicher** liegt und die folgenden Befehle umfasst:

```
LOAD A, [10]
ADD A, [11]
STORE A, [12]
```

LOAD A, [10]

Lade den Akkumulator mit dem Inhalt der Speicherzelle 10.

ADD A, [11]

Addiere den Inhalt der Speicherzelle 11 zum Inhalt des Akkumulators und lege das Ergebnis im Akkumulator ab.

STORE A, [12]

Lege den Inhalt des Akkumulators in Speicherzelle 12 ab.

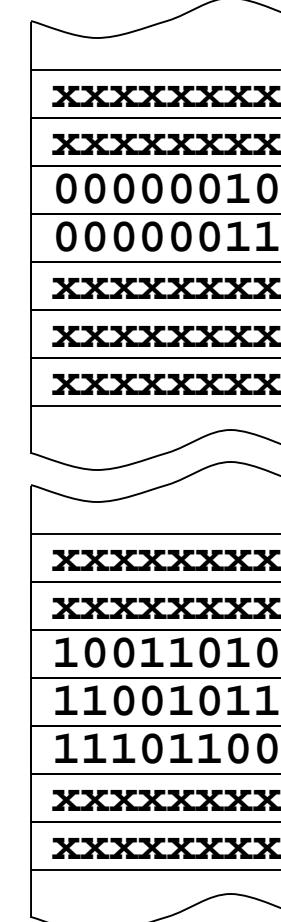
Die Zahl 2_{10}
Die Zahl 3_{10}
reserviert

```
LOAD A, [10]
ADD A, [11]
STORE A, [12]
```

Adressen

8
9
10
11
12
13
14

98
99
100
101
102
103
104



Format des BORIS-Maschinen-Codes

Der Wert der im Speicher abgelegten **Operanden** ist **decodierbar**, wenn die Art der Darstellung bekannt ist. Bei der hier gewählten Binärdarstellung natürlicher Zahlen: „2“ und „3“.

Offenbar muss auch für die **Befehle** ein **Code** gewählt werden, damit für BORIS erkennbar wird, was jeweils getan werden soll.

Wir gehen hier davon aus, dass

- die ersten **4 Bits** den **Befehlscode** darstellen und
- die dann folgenden **4 Bits** zur **Adressangabe** verwendet werden.

1	0	0	1	1	0	1	0
---	---	---	---	---	---	---	---

LOAD A, [X] Binär für „10“

1	1	0	0	1	0	1	1
---	---	---	---	---	---	---	---

ADD A, [X] Binär für „11“

1	1	1	0	1	1	0	0
---	---	---	---	---	---	---	---

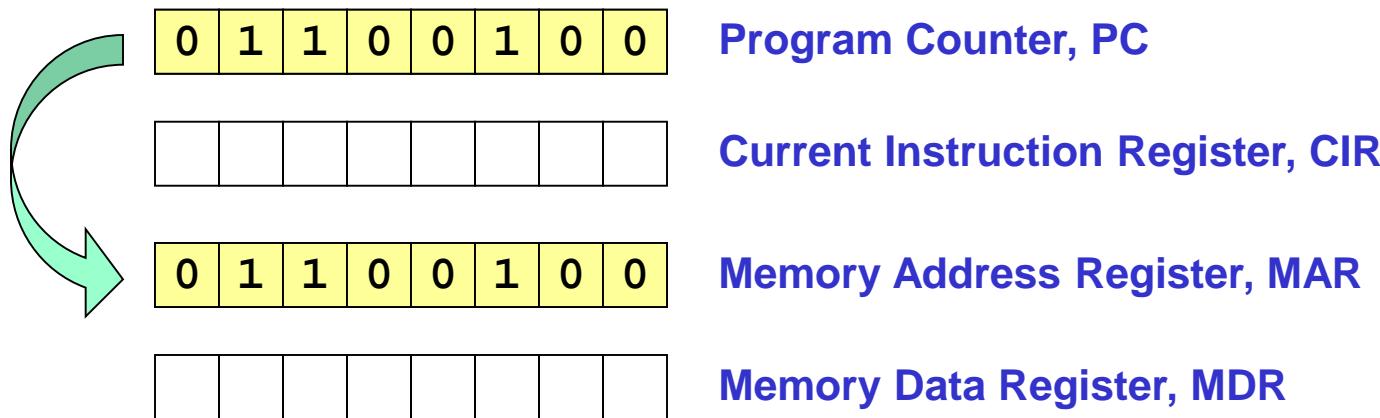
STORE A, [X] Binär für „12“

Anmerkung:

Wir sind hier davon ausgegangen, dass **pro Befehl nur eine Speicherzelle benötigt wird**.

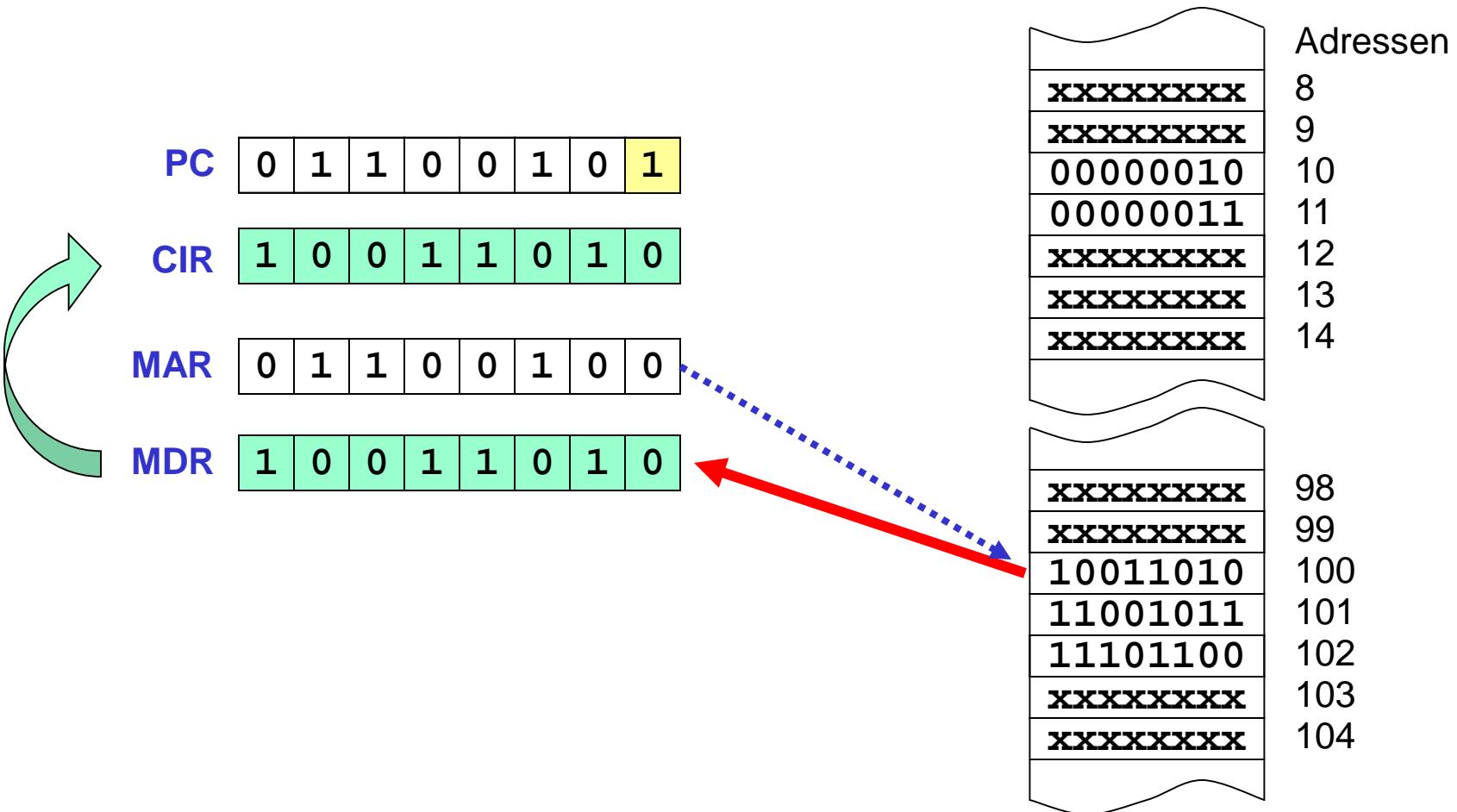
Dies ist in der Realität sehr häufig nicht der Fall!

Initialisierung



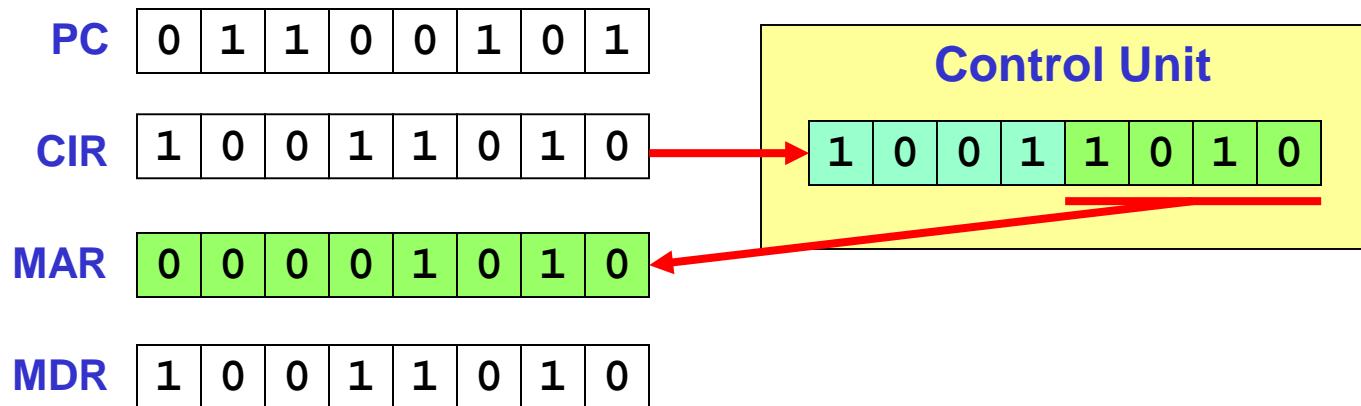
1. Setze den Program Counter auf 100_{10}
2. Lade das MAR mit dem aktuellen Inhalt des Program Counters

Laden des ersten Befehls + Aktualisierung des PC



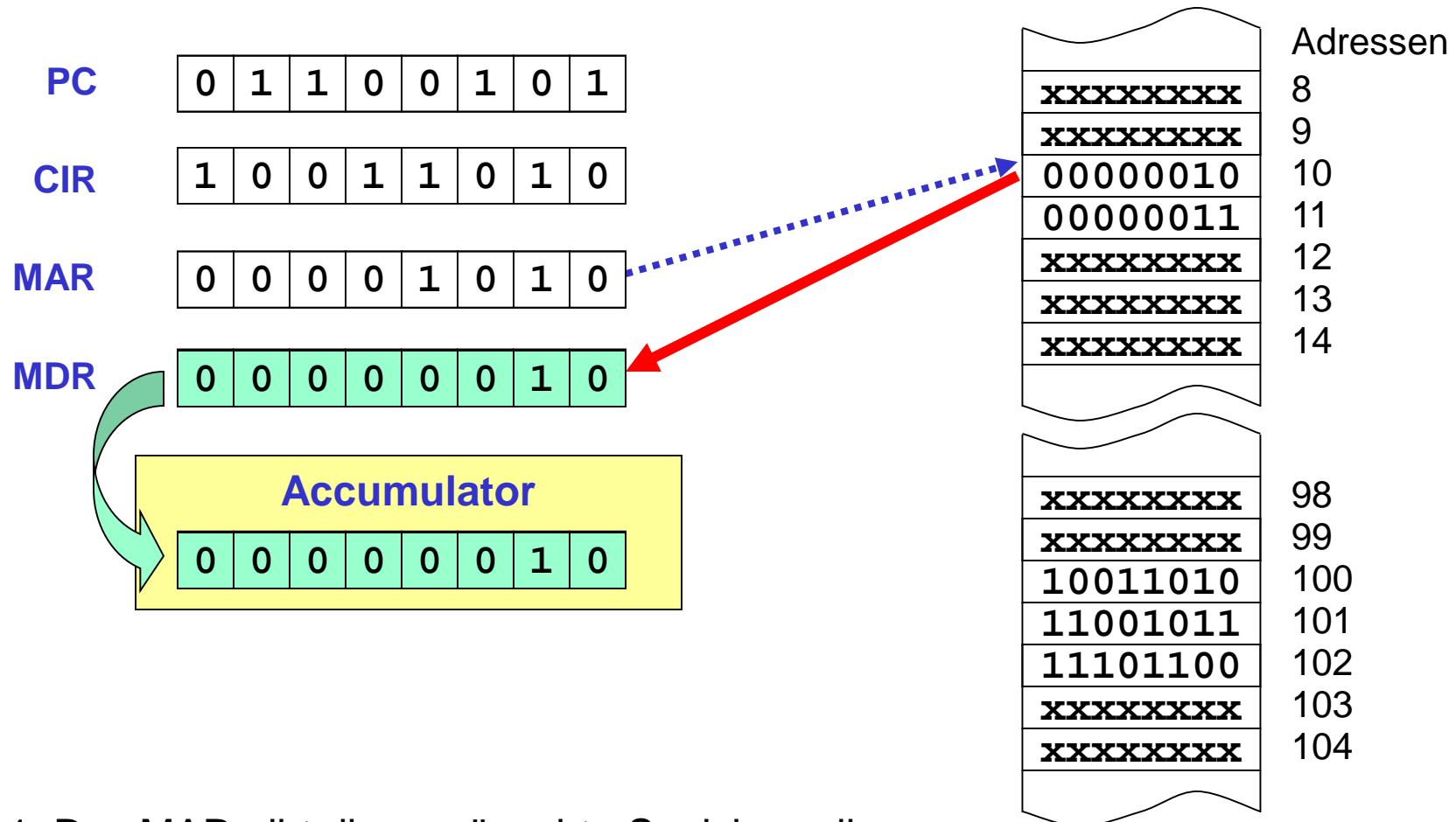
1. Laden des ersten Befehls ins Memory Data Register (MDR)
2. Laden des ersten Befehls ins Current Instruction Register (CIR)
3. Aktualisierung des Program Counters (PC)

Decodierung des ersten Befehls und Setzen des MARs



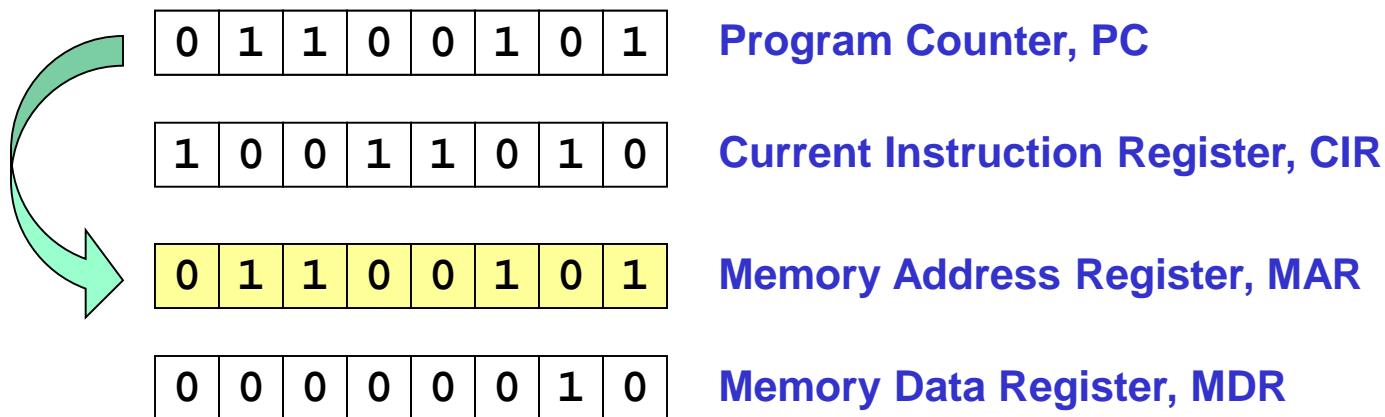
1. Die Kontrolleinheit decodiert den Befehl zu **LOAD A, [X]**
2. Die Kontrolleinheit erkennt die **Adresse 10_{10}**
3. Das Memory Address Register (MAR) wird aktualisiert.

Ausführung des Befehls LOAD A,[10]



1. Das MAR gibt die gewünschte Speicherzelle an.
2. Der Inhalt der gewünschten Zelle gelangt ins MDR.
3. Der Inhalt des MDR wird in den Akkumulator geladen.

Laden der Adresse des zweiten Befehls

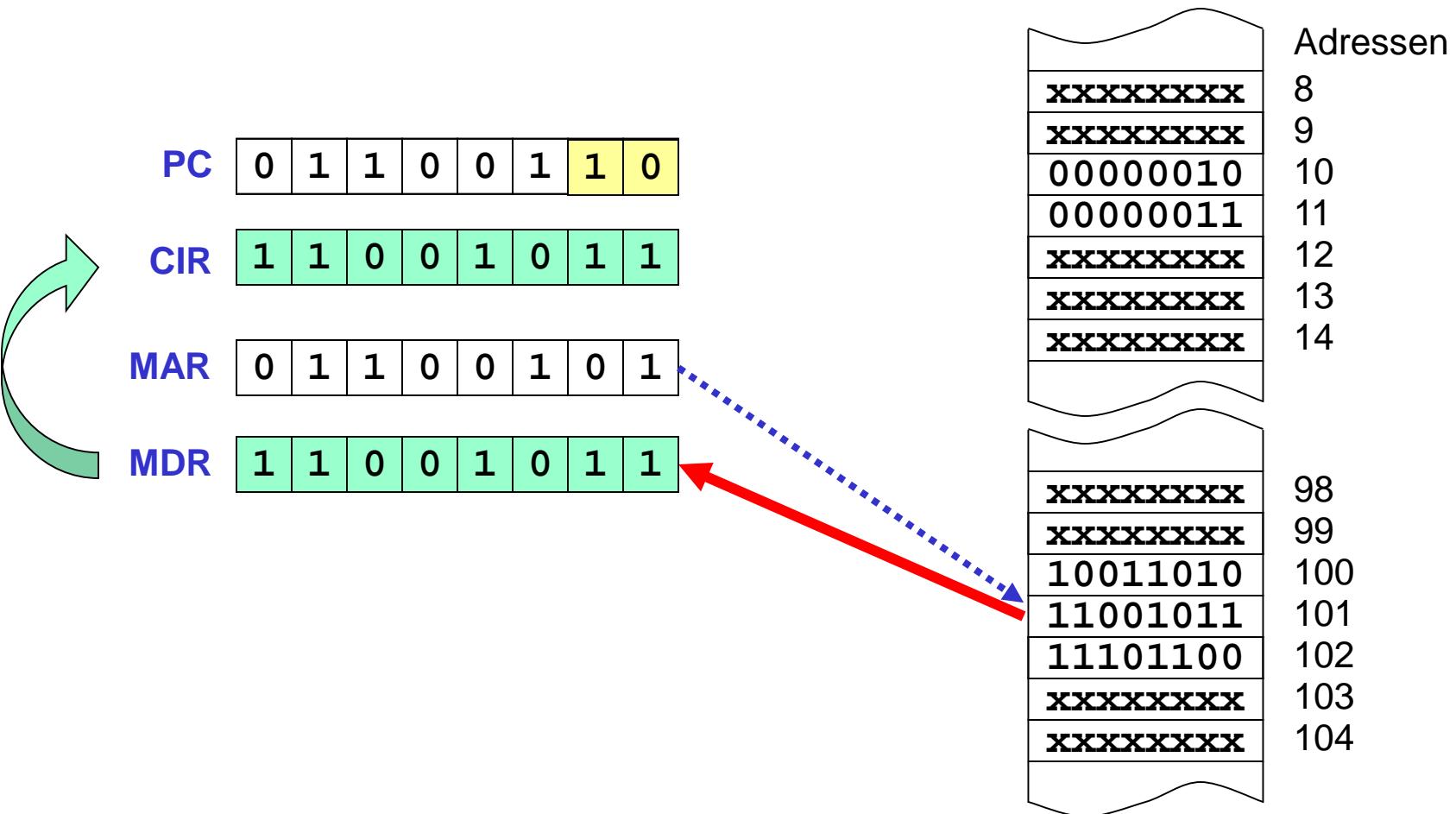


Die bereits im Program Counter enthaltene Adresse des zweiten Befehls wird in das Memory Address Register kopiert.

Anmerkung:

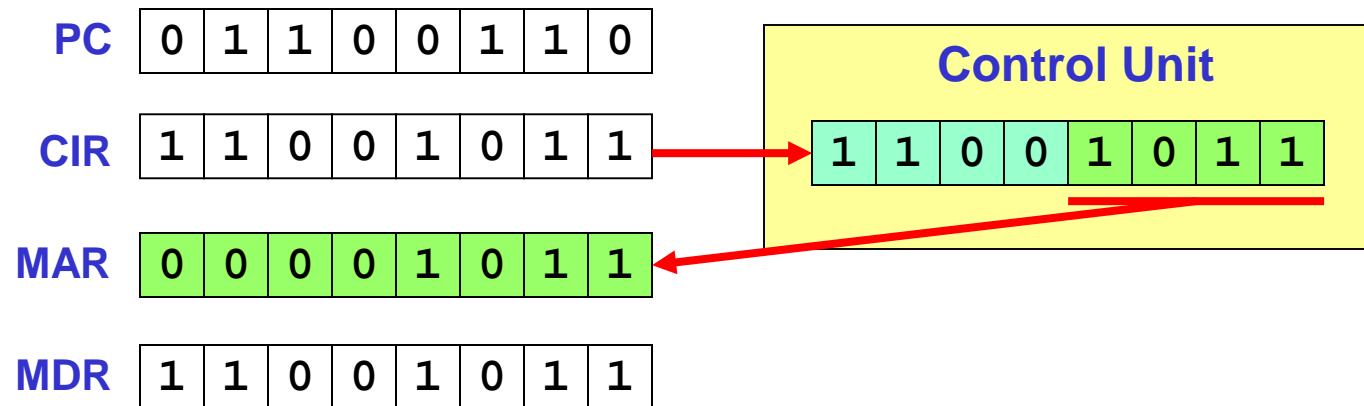
Die Inhalte der übrigen Register bleiben unverändert.

Laden des zweiten Befehls + Aktualisierung des PC



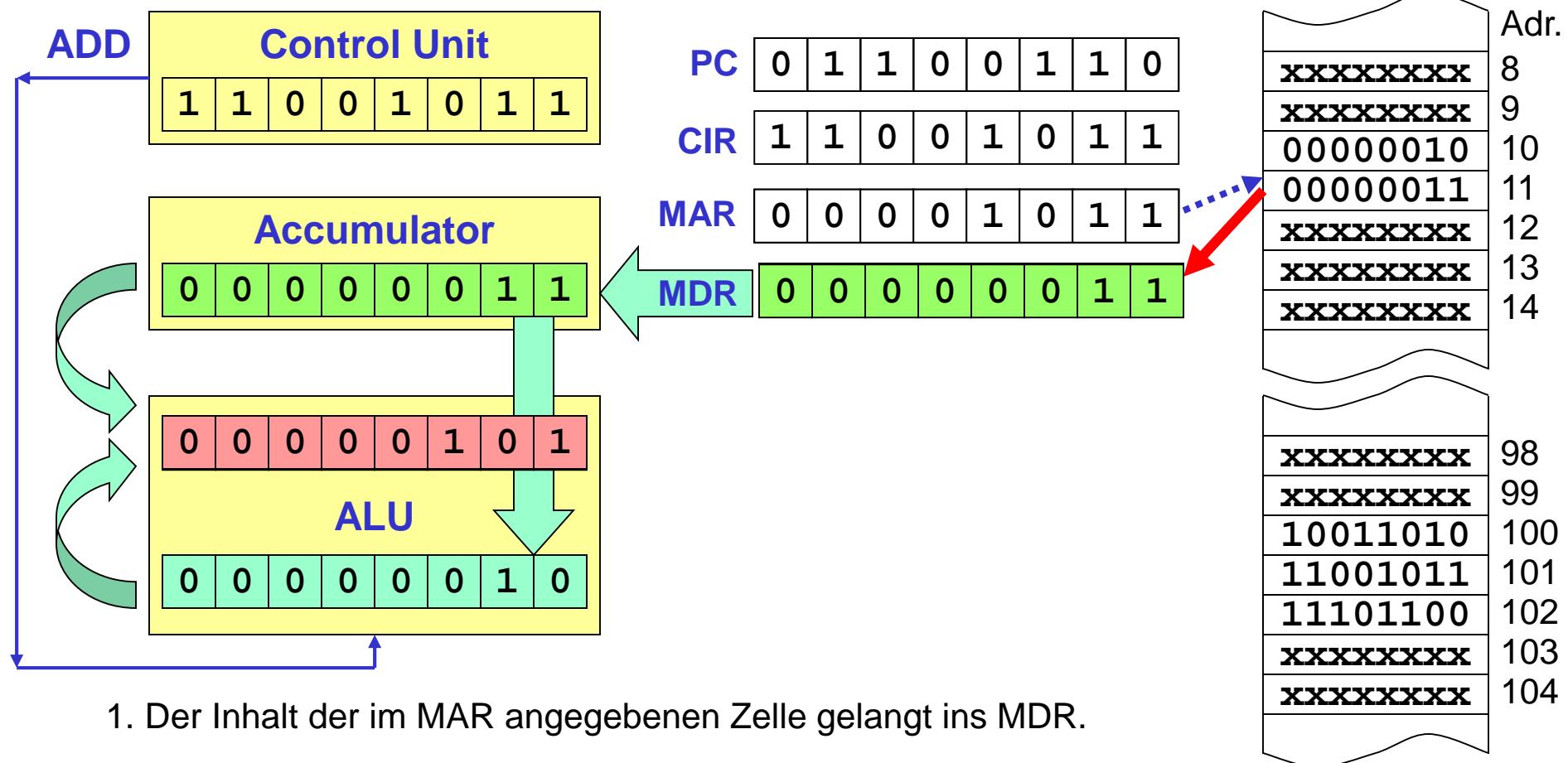
1. Laden des zweiten Befehls ins Memory Data Register (MDR)
2. Laden des zweiten Befehls ins Current Instruction Register (CIR)
3. Aktualisierung des Program Counters (PC)

Decodierung des zweiten Befehls und Setzen des MARs



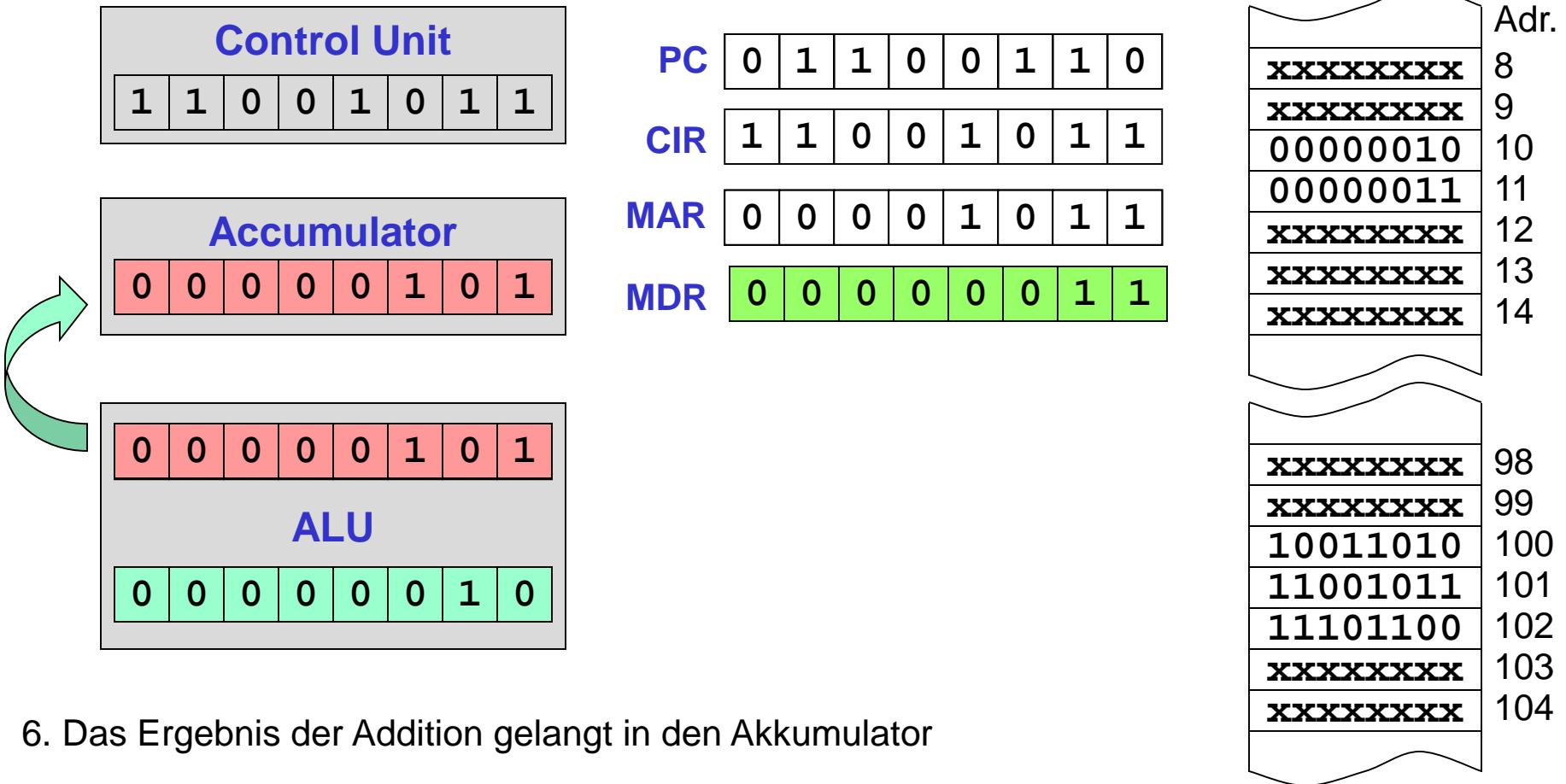
1. Die Kontrolleinheit decodiert den Befehl zu **ADD A, [X]**
2. Die Kontrolleinheit erkennt die **Adresse 11_{10}**
3. Das Memory Address Register (MAR) wird aktualisiert.

Der Befehl ADD A,[11] wird ausgeführt



1. Der Inhalt der im MAR angegebenen Zelle gelangt ins MDR.
2. Der Inhalt des Akkumulators wird in die ALU kopiert.
3. Der Inhalt des MDR wird in den Akkumulator geladen.
4. Die ALU erhält die Aufforderung, die Addition durchzuführen.
5. Das Ergebnis der Addition wird in der ALU zwischengespeichert.

Der Befehl ADD A,[11] wird ausgeführt (2)

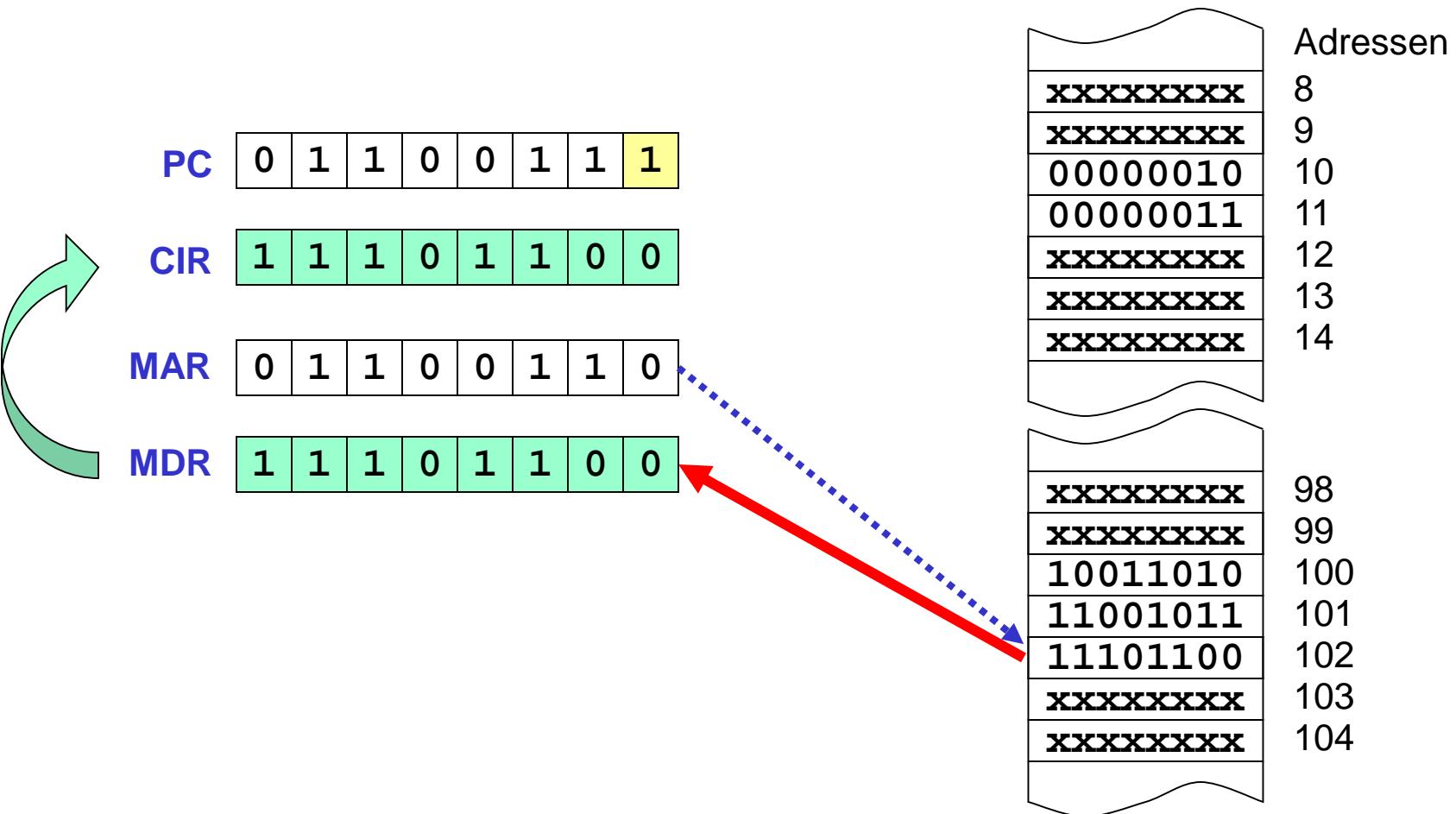


Laden der Adresse des dritten Befehls



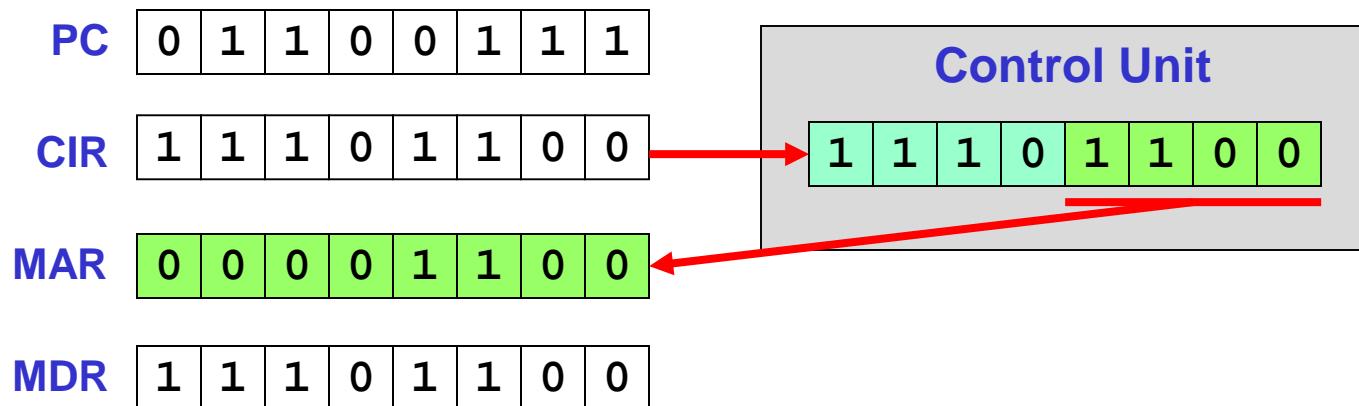
Die bereits im Program Counter enthaltene Adresse des dritten Befehls wird in das Memory Address Register kopiert.

Laden des dritten Befehls und Aktualisierung des PC



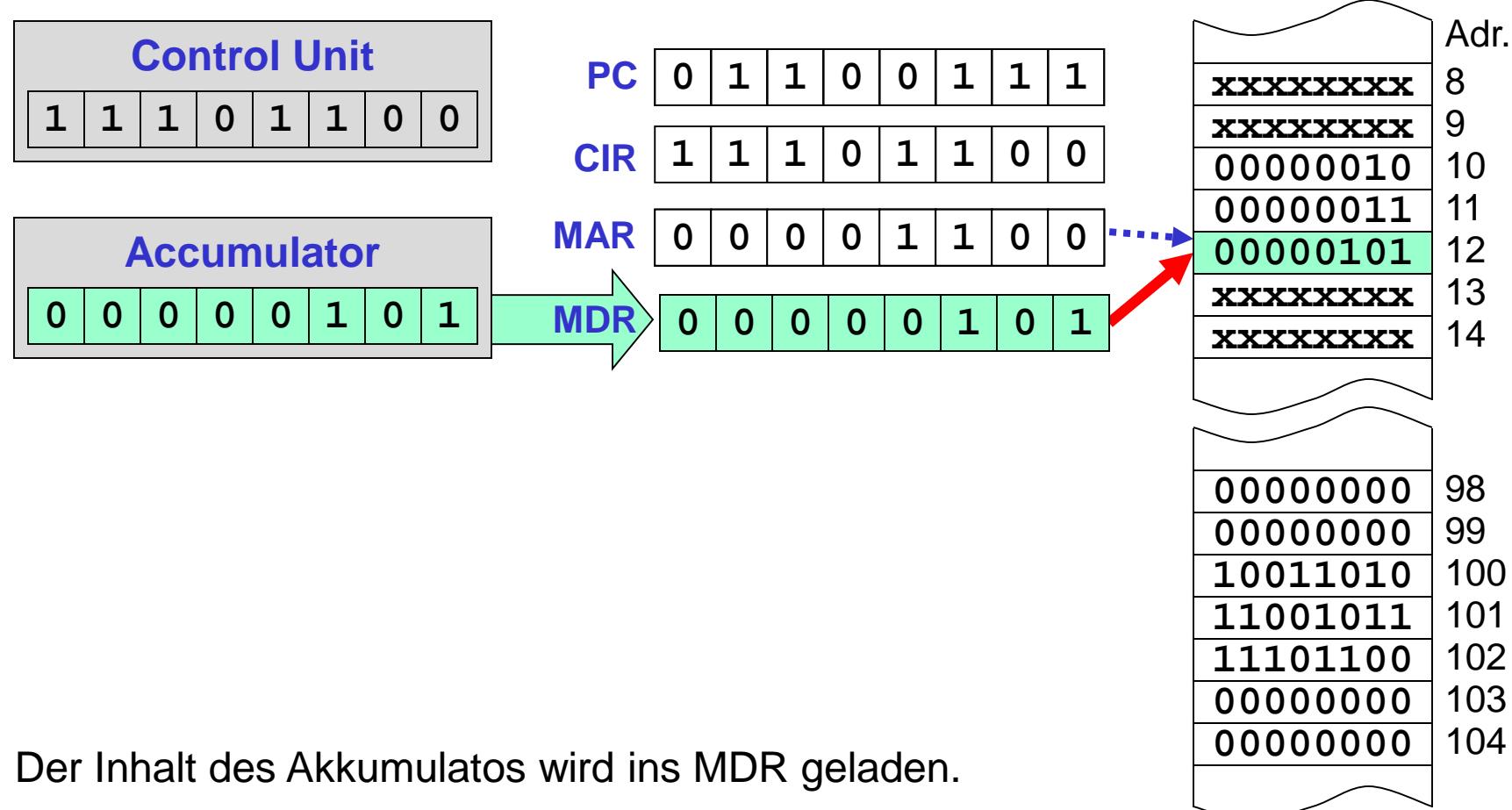
1. Laden des dritten Befehls ins Memory Data Register (MDR)
2. Laden des dritten Befehls ins Current Instruction Register (CIR)
3. Aktualisierung des Program Counters (PC)

Decodierung des dritten Befehls und Setzen des MARs



1. Die Kontrolleinheit decodiert den Befehl zu **STORE A, [X]**
2. Die Kontrolleinheit erkennt die **Adresse 12_{10}**
3. Das Memory Address Register (MAR) wird aktualisiert.

Der Befehl STORE A,[12] wird ausgeführt



Und dann ?

BORIS ist **ständig aktiv**. Er

- greift munter als nächstes auf die **Speicherzelle 103_{10}** zu,
- um den darin enthaltenen **Befehl** zu **decodieren** und **auszuführen**.



Problem:

Das, was (zufällig?) in Speicherzelle 103_{10} steht,

- ist ggf. gar **nicht** als Befehl **decodierbar**,
- ist in unserem Beispiel sicher **nichts, was dort gezielt abgelegt wurde**.

Was passiert in der Realität beim Ende der Programmbearbeitung?

- Der **Prozessor kehrt zurück** zu dem Programm, aus dem der Aufruf erfolgte.
- Bei einem aktiven Prozessor ist also **stets ein Programm in Bearbeitung**.
- Ist kein Anwendungsprogramm aktiv, dann wird ein **Programm zur** allgemeinen **Systemverwaltung** bearbeitet (Systemprogramm).

1.2.3. Von CISC und RISC

Reduced Instruction Set Computer (RISC) wurde Anfang der 80er Jahre zum Begriff für eine Design-Philosophie, bei der es nur wenige Befehle gibt.

Zeitgleich wurde „**Complex Instruction Set Computer**“ (**CISC**) zum Begriff für „Nicht-RISC“.

[1.2.3.1. Complex Instruction Set Computer \(CISC\)](#)

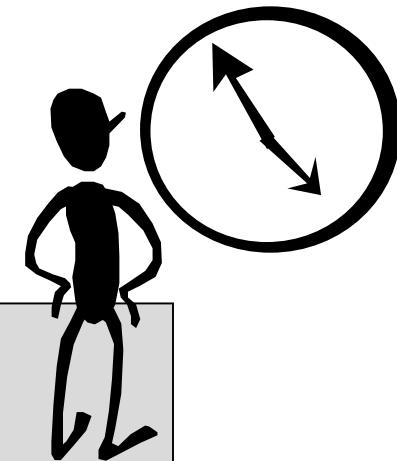
[1.2.3.2. Reduced Instruction Set Computer \(RISC\)](#)

[1.2.3.3. RISC und CISC im Vergleich](#)

1.2.3.1. Complex Instruction Set Computer (CISC)

CISC-Prozessoren verwenden einen umfangreichen und stark heterogenen Befehlssatz:
Meist 200 – 400 Befehle.

Ziel: Kurze Laufzeit **wenige Befehle / Programm.**



„Performance Equation“:

$$\frac{\text{time}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \cdot \frac{\text{cycles}}{\text{instruction}} \cdot \frac{\text{time}}{\text{cycle}}$$



Wenige Code-Zeilen, kurze Programmlaufzeit?

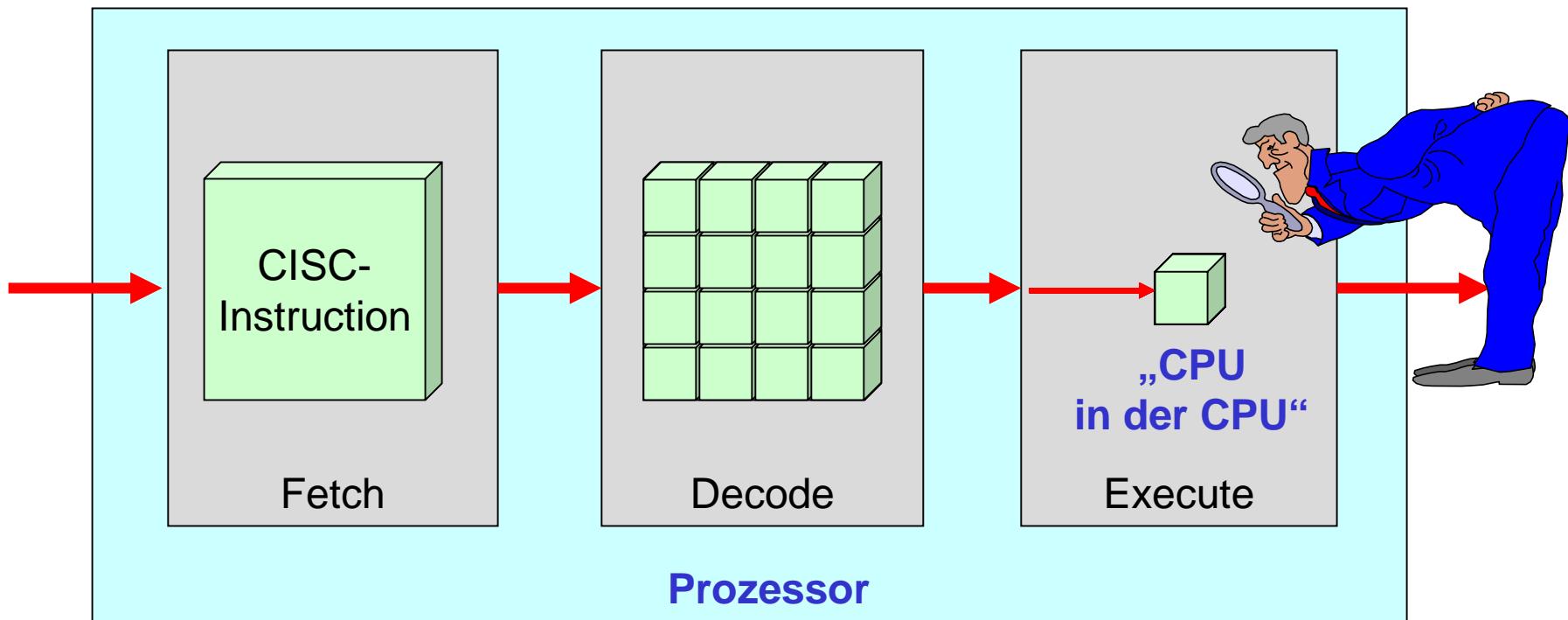
Die Rechnung geht nur auf, wenn die anderen Faktoren nicht zu stark wachsen.

Außerdem wird vorausgesetzt, dass der **Compiler** die „mächtigen“ **Operationen auch wirklich einsetzt** und nicht durch mehrere einfache Operationen nachbildet.

Programmausführung mit Mikro-Code

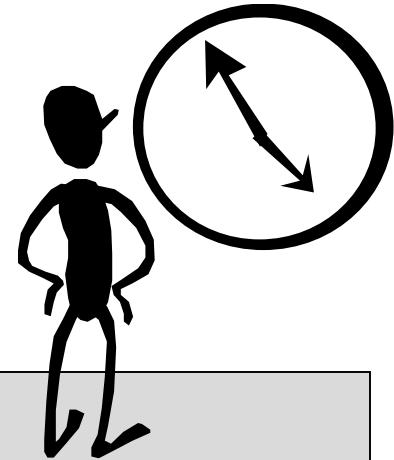
Die **Mikroprogrammierung** gestattet es

- **komplexe Befehle mittels „Mikro-Befehlen“ zu realisieren**
- statt jeden Befehl einzeln zu verdrahten



1.2.3.2.. Reduced Instruction Set Computer (RISC)

Die RISC-Philosophie strebt danach, die Laufzeit für die Bearbeitung eines vorgegebenen Programms dadurch gering zu halten, dass **jeder einzelne (Maschinen-Befehl) sehr schnell ausführbar** ist.



„Performance Equation“:

$$\frac{\text{time}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \cdot \frac{\text{cycles}}{\text{instruction}} \cdot \frac{\text{time}}{\text{cycle}}$$



Wenige, leicht zu bearbeitende Befehle
nach Möglichkeit: 1 Cycle per Instruction
(CPI)

Anmerkung:

Befehle eines **RISC-Assemblers** sind nach Art und Umfang sehr **ähnlich den Mikro-Code-Befehlen** einer CISC-Maschine.

RISC: Load, Store, Registerbefehle, neuartige Compiler

Neben der Reduktion des Befehlssatzes gab es weitere Innovationen, die der RISC-Architektur zum Erfolg verhalfen:

a) Verzicht auf komplexe Adressierung

Zugriffe auf den **Hauptspeicher** können nur noch mit **Load** und **Store** erfolgen.

Vor der Durchführung von Operationen (z.B. Arithmetik) ist es erforderlich, die Speicherinhalte in Register zu kopieren. Auch das Ergebnis von arithmetischen und logischen Operationen kann nur noch in Registern abgelegt werden.

b) Einsatz vieler Register

Bei vielen Registern müssen **nur selten Operanden aus dem Hauptspeicher** geholt werden.

Codeanalysen ergaben, dass die große Mehrzahl der Operanden lokale Skalare sind. Durch Vergrößerung der Anzahl der Register kann die Zahl der Speicherzugriffe drastisch reduziert werden: Fast immer ist der benötigte Wert aufgrund früherer Verwendung schon in einem Register.

c) Dem Compiler kommt zentrale Bedeutung zu

Den Compilern kommt nun die Aufgabe zu, sehr geschickten **Gebrauch von den stark begrenzten Ressourcen** zu machen.

RISC-Maschinen garantieren extrem schnelle Ausführung der (wenigen) vorhandenen Operationen. Sie verlassen sich darauf, dass der generierte Code die verfügbaren Ressourcen sehr effizient nutzt.

1.2.3.3. RISC und CISC im Vergleich

Die nachfolgende Auflistung erläutert Design-Entscheidungen der CPU-Architekten.

Eine **eindeutige Zuordnung** konkreter Prozessoren zu „RISC“ oder „CISC“ ist **unmöglich**:

Hersteller greifen meist das beste aus beiden „Philosophien“ auf.

CISC	RISC
Komplexität: Von der Software in die Hardware	Komplexität: Von der Hardware in die Software
Performance: weniger Instruktionen/Programm, aber mehr CPIs	Performance: Weniger CPIs, mehr Instruktionen/Programm
Befehlssatz: groß und stark heterogen (trivial bis komplex)	Befehlssatz: klein und homogen
Unterstützung höherer Programmiersprachen: sehr starke Hardware-Unterstützung	Unterstützung höherer Programmiersprachen: ausschließlich in Software (Compiler)
Adressierung: umfassende Speicher-zu-Speicher-Adressierung	Adressierung: nur Load, Store und Register-Operationen
Mikro-Code: wird intensiv genutzt	Mikro-Code: wird nicht genutzt.
Register: nur wenige Transistoren sind in Register investiert	Register: viele Transistoren sind in Register investiert

1.2.4. Pipelining, Super-Pipelining, Superskalar-Technik

Im Mai 2000 veröffentlichte das „**Spektrum der Wissenschaft**“ den Artikel „**Neue Architekturen für Mikroprozessoren**“. Zur Erläuterung des Geschehens innerhalb eines Mikroprozessors verwendet der Artikel das Bild eines kleinen Männchens, das im Inneren des Prozessors gewisse Aktionen ausführt.

Bei der **klassischen CISC-Architektur** arbeitet das Männchen **wie ein klassischer Handwerker: Alles selbst machen, eins nach dem anderen.**



Quelle: A. Bode, „Neue Architekturen für Mikroprozessoren“, Spektrum der Wissenschaft, Mai 2000

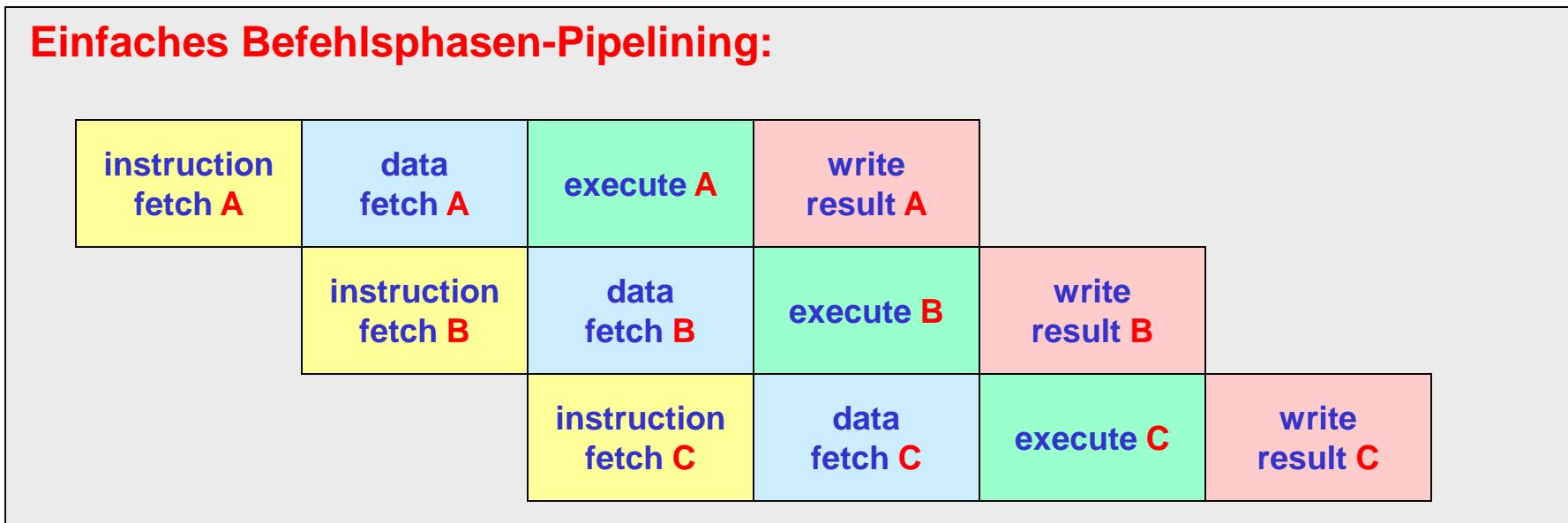
Was tun, wenn wir **mehrere „Männchen“** einsetzen können ?

Wie kann eine **geordnete Arbeitsteilung** erfolgen ?

Einfaches Befehlspipelining

Pipelining (+ Verbesserungen dazu) wird **in allen gängigen Prozessoren** eingesetzt.

Beim **Pipelining** wird eine **komplexe Aufgabe** so in **Teilaufgaben zerlegt**, dass eine **zeitlich überlappende Bearbeitung** möglich und realisiert wird.



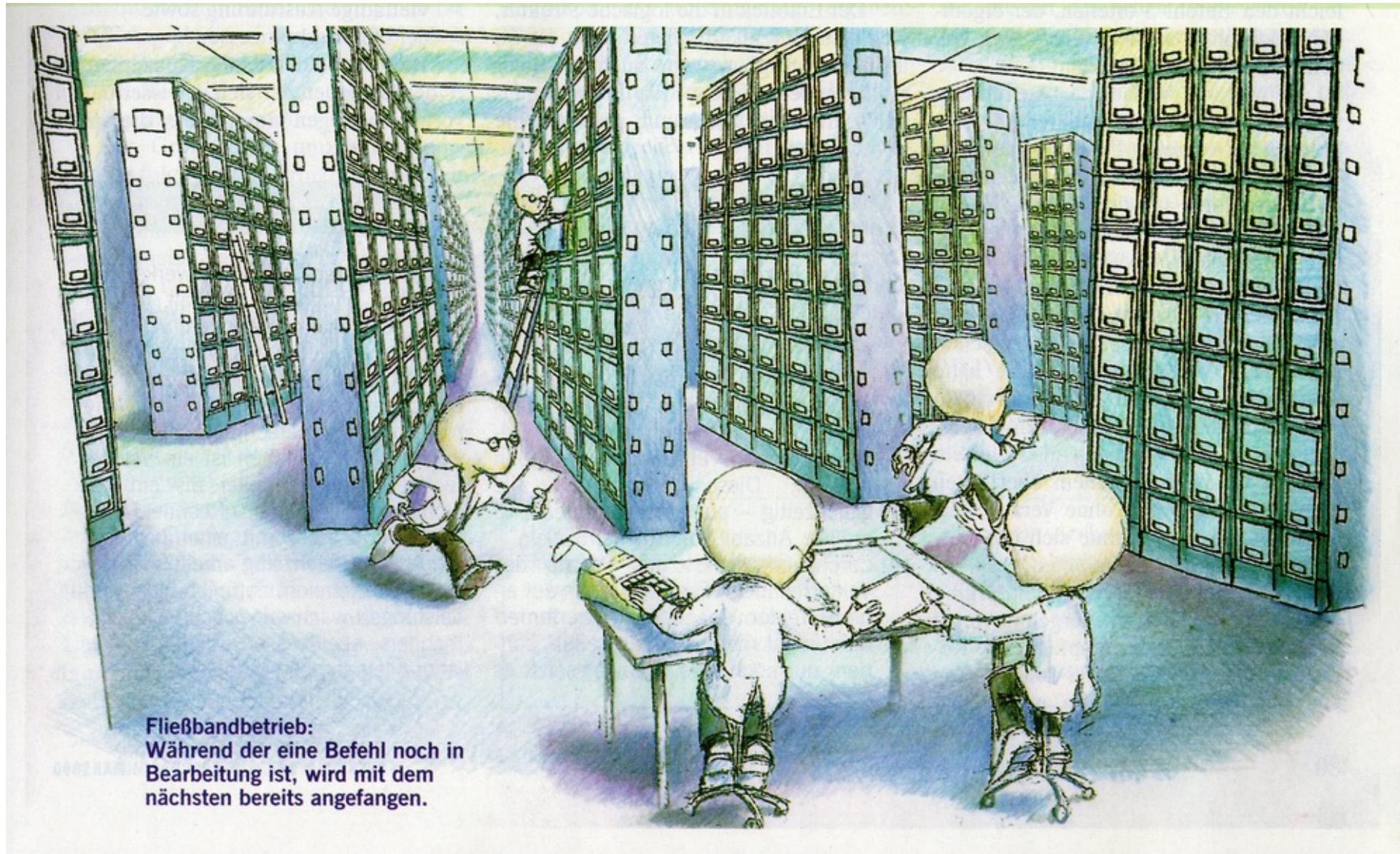
Die Bearbeitung der jeweiligen Phasen kann durch getrennte Einheiten erfolgen.

Besonders günstig: ist diese Technik, wenn **alle Phasen gleich lang** sind.

Besonders schwierig: ist das **Bestimmen der nächsten Instruktion(en)**.

→ **Sprungvorhersage (Branch Prediction)**

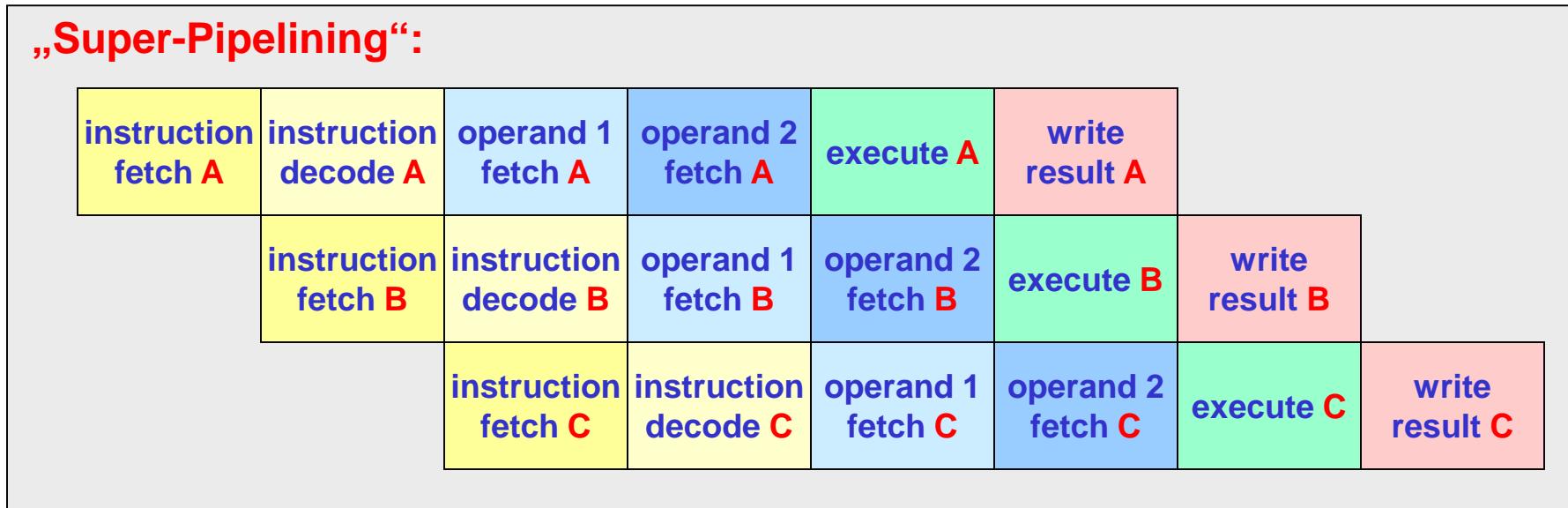
Pipelining



Quelle: A. Bode, „Neue Architekturen für Mikroprozessoren“, Spektrum der Wissenschaft, Mai 2000

Super-Pipelining

Eine weitere Parallelisierung ist erzielbar, indem die „natürlichen“ funktionalen Einheiten weiter aufgespalten werden:



Zentrale Bedeutung kommt auch hier der geschickten Vorhersage der nächsten Instruktion zu:
„Branch Prediction“.

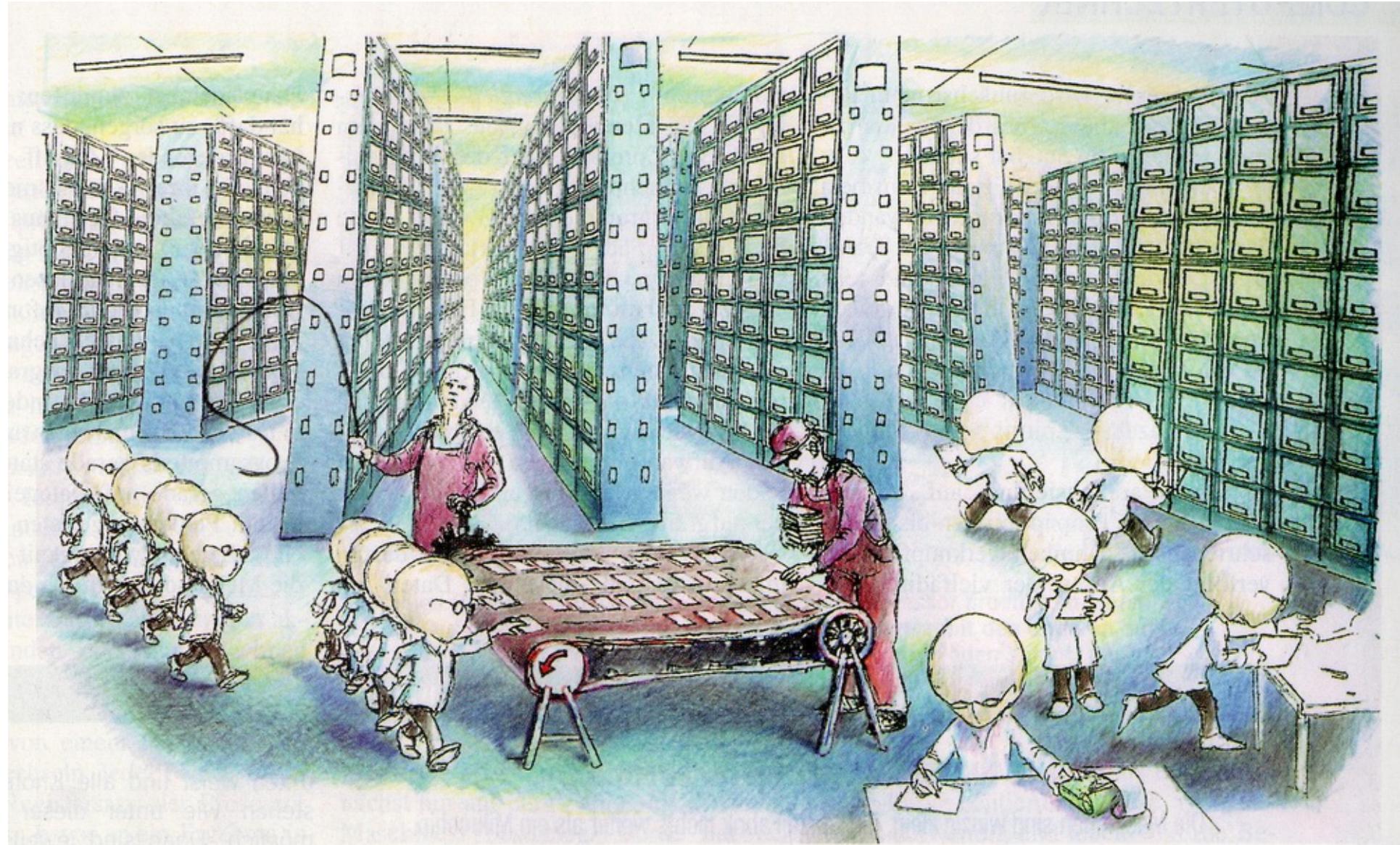
Die „Superskalar-Technik“

Eine weitere Möglichkeit der Parallelisierung besteht darin, den Befehlsfluss so zu zerlegen, dass innerhalb der einzelnen Bearbeitungseinheiten **mehrere Befehle** (bzw. deren Teile) **gleichzeitig bearbeitet** werden können.

Prinzip der „Superskalar-Architektur“:

instruction fetch A	data fetch A	execute A	write result A	
instruction fetch B	data fetch B	execute B	write result B	
instruction fetch C	data fetch C	execute C	write result C	
instruction fetch D	data fetch D	execute D	write result D	
instruction fetch E	data fetch E	execute E	write result E	
instruction fetch F	data fetch F	execute F	write result F	

„Superskalar-Technik“



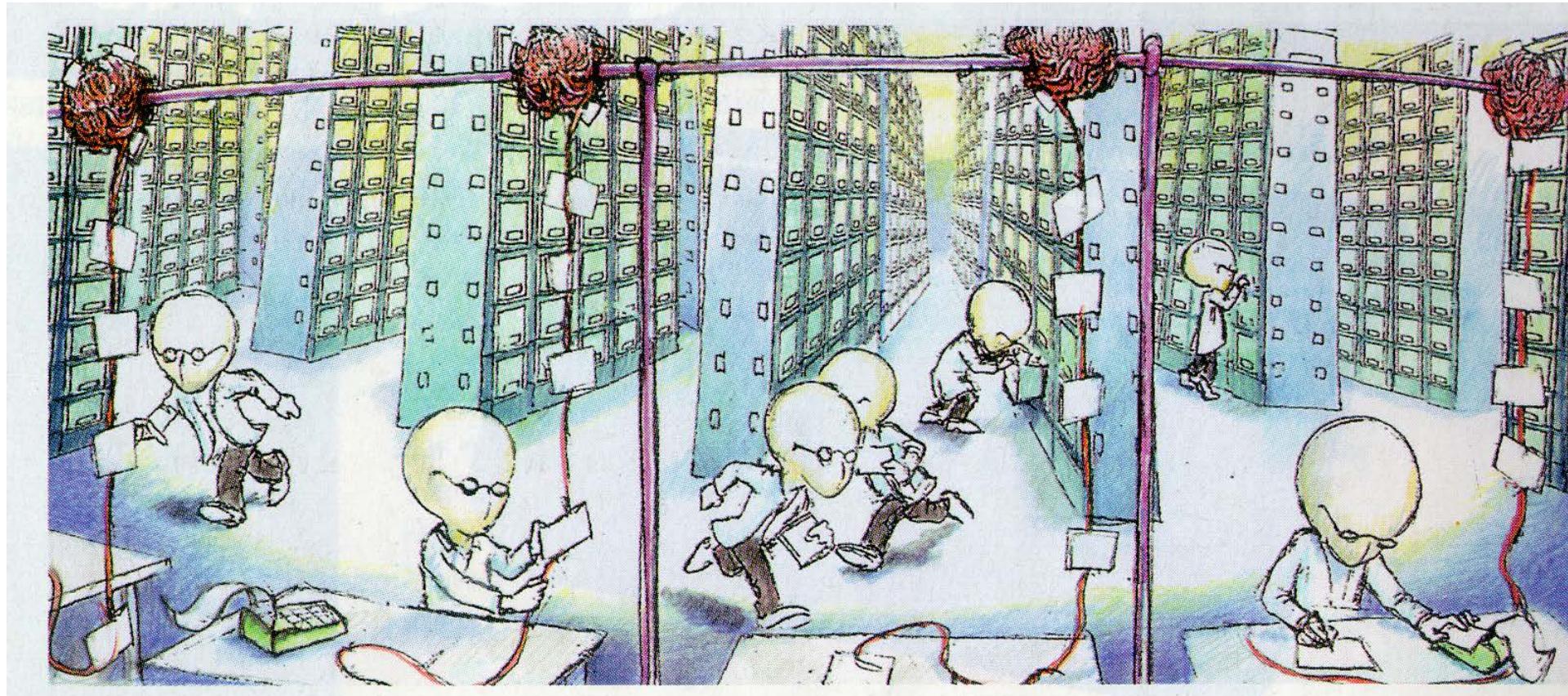
Quelle: A. Bode, „Neue Architekturen für Mikroprozessoren“, Spektrum der Wissenschaft, Mai 2000

„Vielfädige Programmierung“ (Multithreading)

Die Krönung: Vielfädige Programmausführung

Jeder einzelne Prozessor / Prozessorkern arbeitet an einem Teil der Aufgabe, einem Faden, **ohne sich nennenswert mit den anderen zu verstndigen.**

Es ist die Aufgabe des **Compilers**, der die Fden spinnt, dafr zu sorgen, dass die **Prozessoren einander nicht ins Gehege kommen.**



Quelle: A. Bode, „Neue Architekturen fr Mikroprozessoren“, Spektrum der Wissenschaft, Mai 2000

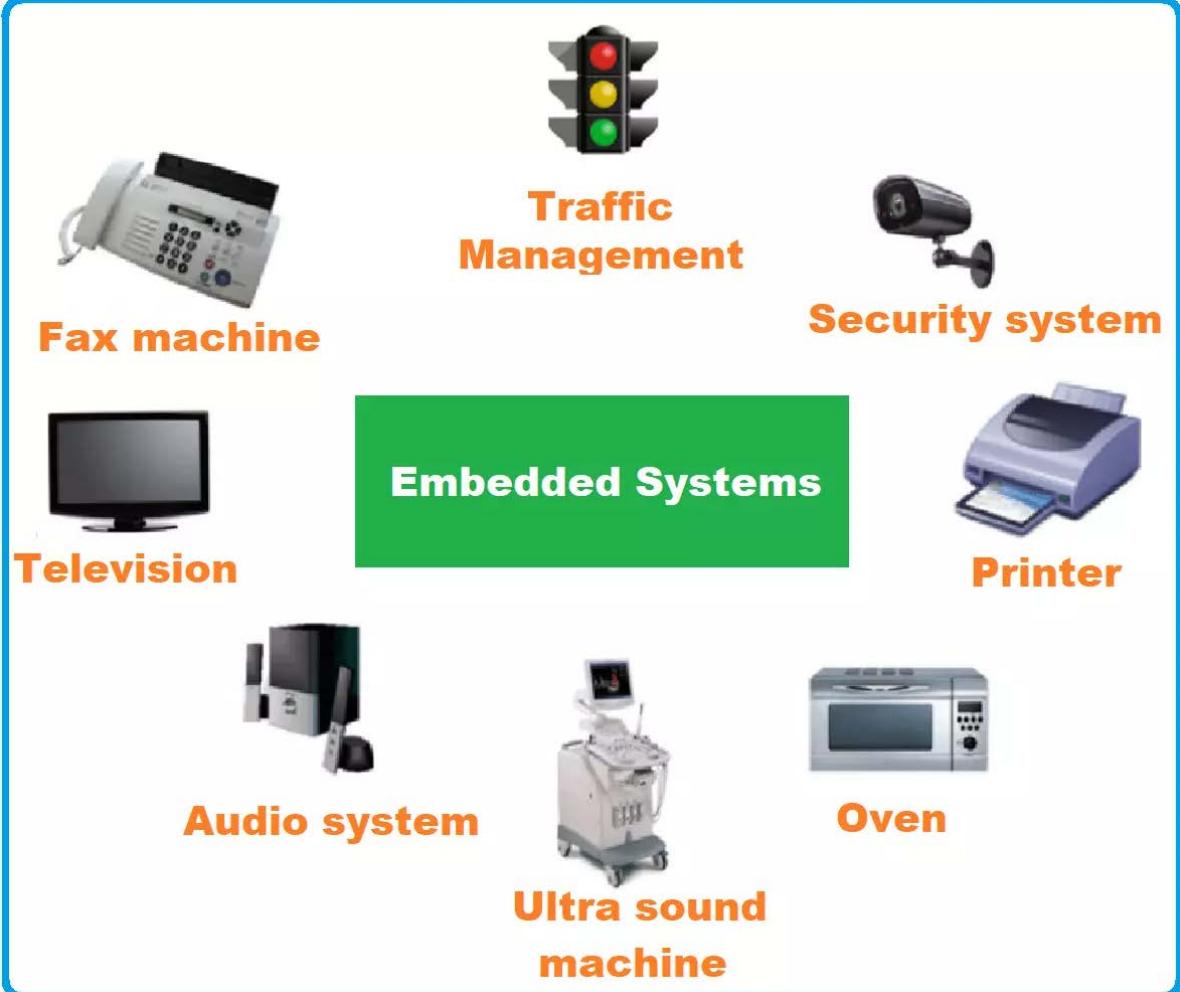
1.2.5. Aktuelle Entwicklungen

Wichtige Anforderungen an die Architekturen der Zukunft:

- **Skalierbarkeit** im Hinblick auf die **Leistungsfähigkeit der Einzelkomponenten**
- **Skalierbarkeit** im Hinblick auf die **Anzahl aktiver Einheiten**
- **meist:** gute Unterstützung von „**Multimedia**“
- **meist:** Integrationsfähigkeit in **sehr schnelle Netze**
- **zunehmend:** **geringer Energieverbrauch** (Wearable Computers)

Es würde den Rahmen der Vorlesung sprengen, detailliert auf die aktuellen Entwicklungen der führenden Hersteller einzugehen. Einschlägige Informationen finden sich im Internet.

Die unsichtbaren Computer



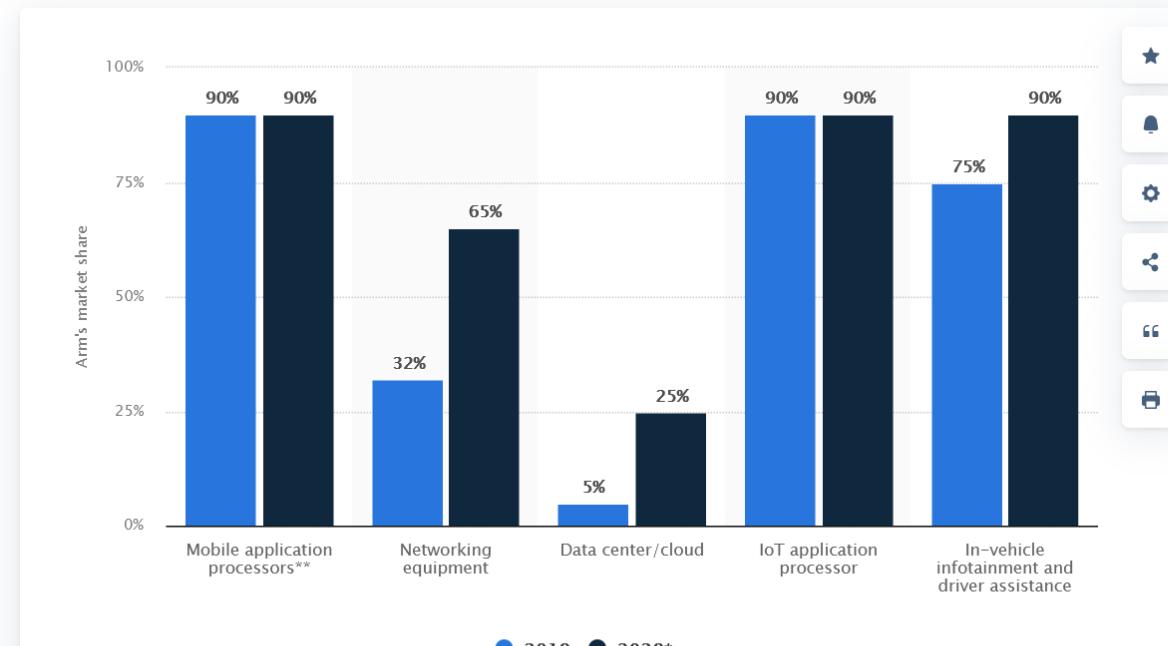
Buderus



Technology & Telecommunications › Hardware

PREMIUM +

Arm's market share and targets across key technology markets in 2019 and 2028 fiscal years



Additional Information

© Statista 2023 Show source

DOWNLOAD

PDF XLS PNG PPT

August 2020

Region

Worldwide

Survey time period

2019

Supplementary notes

* Based on Arm's internal modeling and analysis of the markets.

** Includes processors for smartphones, tablets, and laptops.

SoftBank's fiscal year ends 31 March each year.

SoftBank group acquired UK headquartered Arm Holdings on 5 September 2016.

Citation formats

View options

Auslieferung von 6,7 Milliarden ARM-Basierten Chips im Quartal 4/2020

The screenshot shows a news article from the Arm website. The header includes the Arm logo and navigation links for Products, Markets, Partners, Developers, Support & Training, and Company. The main title is "The Arm ecosystem ships a record 6.7 billion Arm-based chips in a single quarter". Below the title is a date: February 11, 2021. The text discusses the fourth quarter of 2020 (Arm FY Q320) and highlights several key points:

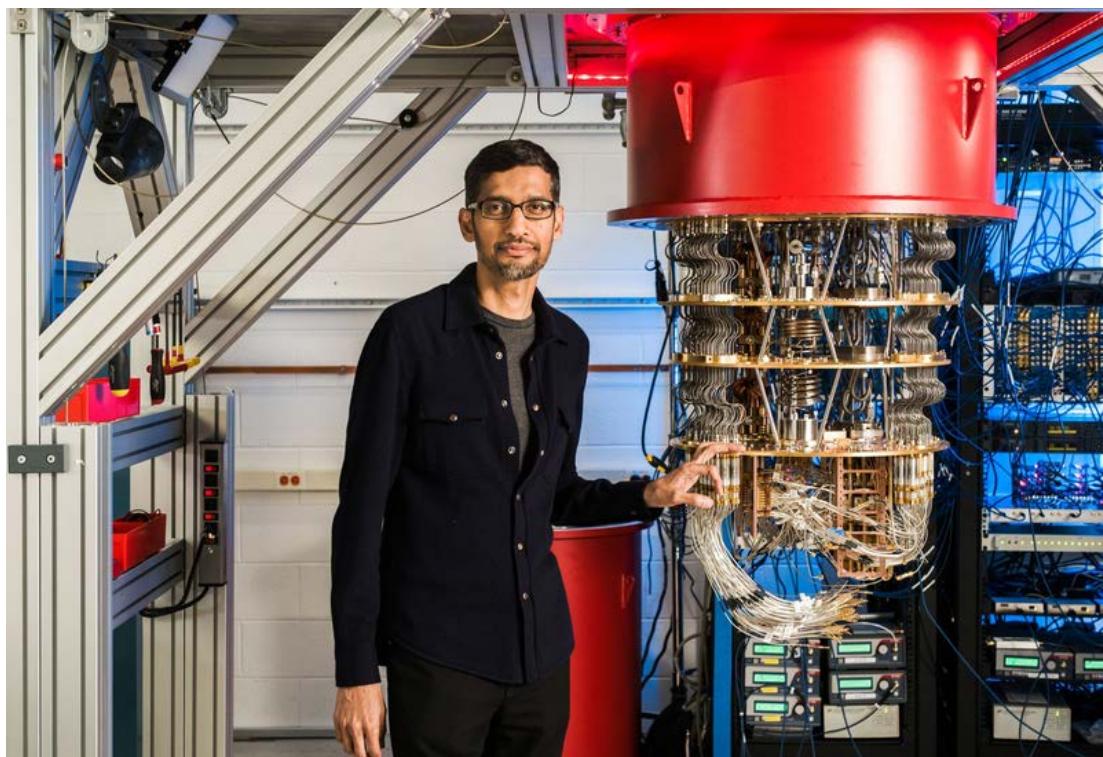
- Arm reported its silicon partners shipped in the prior quarter a record 6.7 billion Arm-based chips, which equates to ~842 chips shipped per second. To-date, Arm partners have shipped more than 180 billion Arm-based chips.
- Arm continues to be the leading architecture for IoT and embedded devices with a record 4.4 billion chips based on Arm Cortex-M reported as shipped in the quarter.
- Arm Mali graphics processors remain the number one shipping GPU.

The text also notes that the Arm ecosystem continues to grow, with Arm partners signing a record 175 new licenses in calendar year (CY) 2020, bringing the total to 1,910 licenses and 530 licensees. A graph titled "Arm is Everywhere Compute Happens" illustrates the exponential growth of Arm chip shipments over time, starting at 1bn in 1991 and reaching >180bn by 2021. The graph shows various device icons representing the wide range of applications where Arm technology is used.

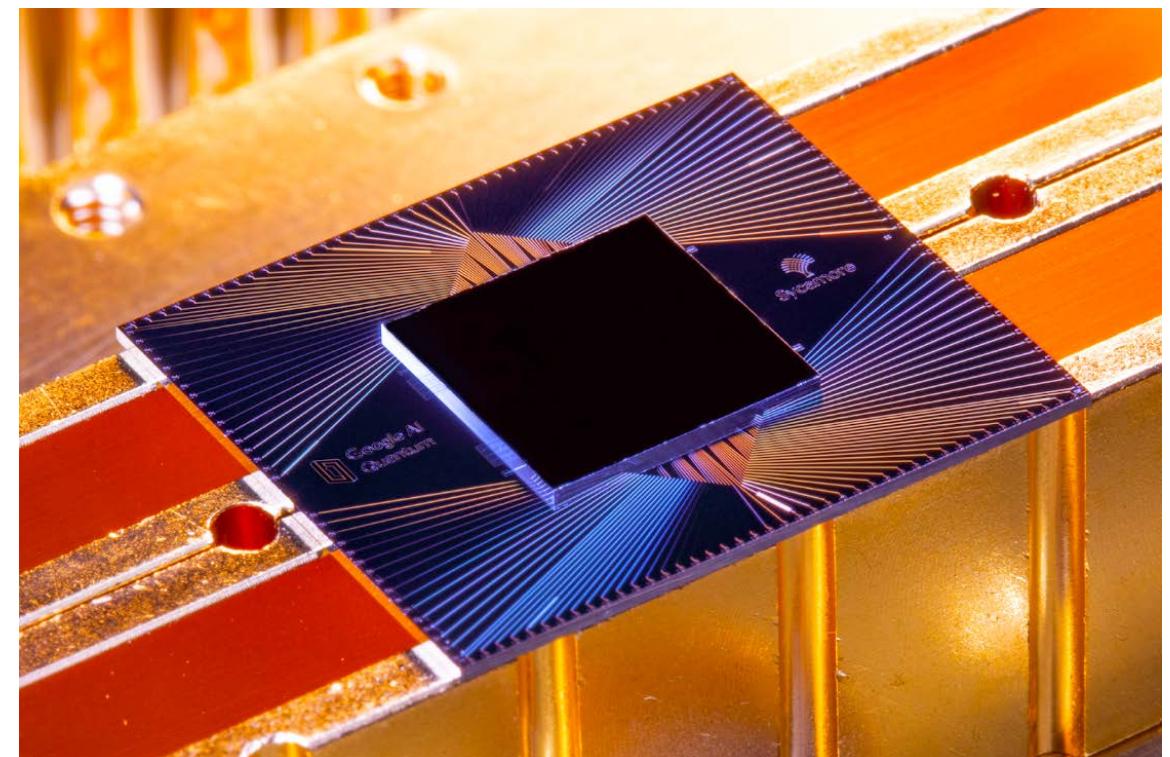
Included within the record shipments are Arm Mali-based graphics processors which have been the number one shipping GPU since 2015.

<https://www.arm.com/company/news/2021/02/arm-ecosystem-ships-record-6-billion-arm-based-chips-in-a-single-quarter>

Quantencomputer



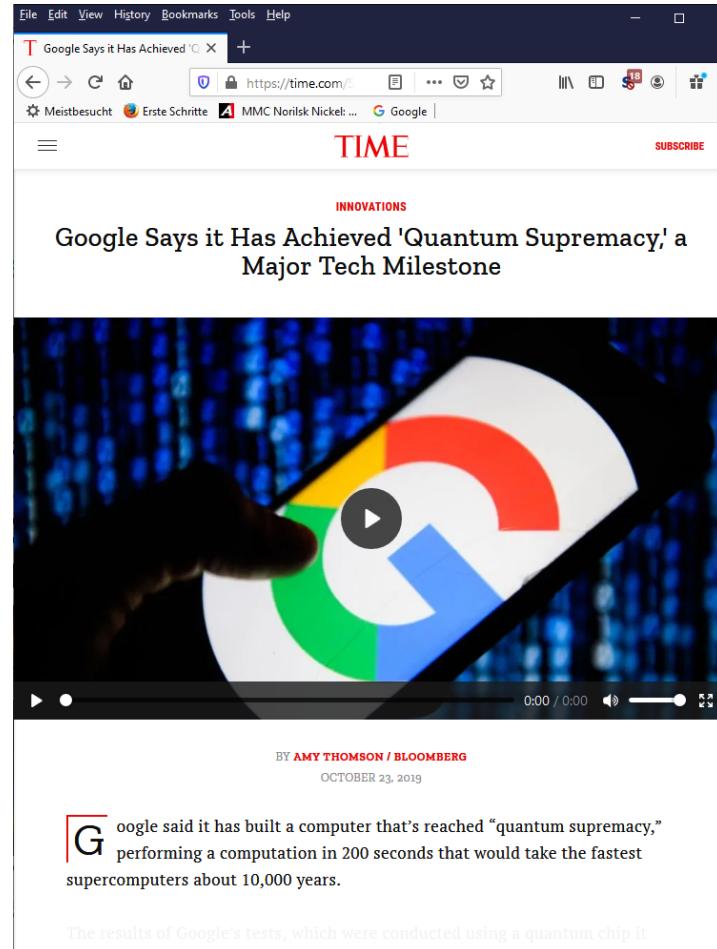
<https://www.it-times.de/news/google-meldet-fortschritte-beim-quantum-computer-sycamore-133457/>



Quelle: <https://www.nature.com/articles/d41586-019-03213-z>

„Quantenüberlegenheit“

Google: Quantenprozessor „Sycamore“ braucht nur 3 Minuten – statt 10.000 Jahre



Quantenphysik

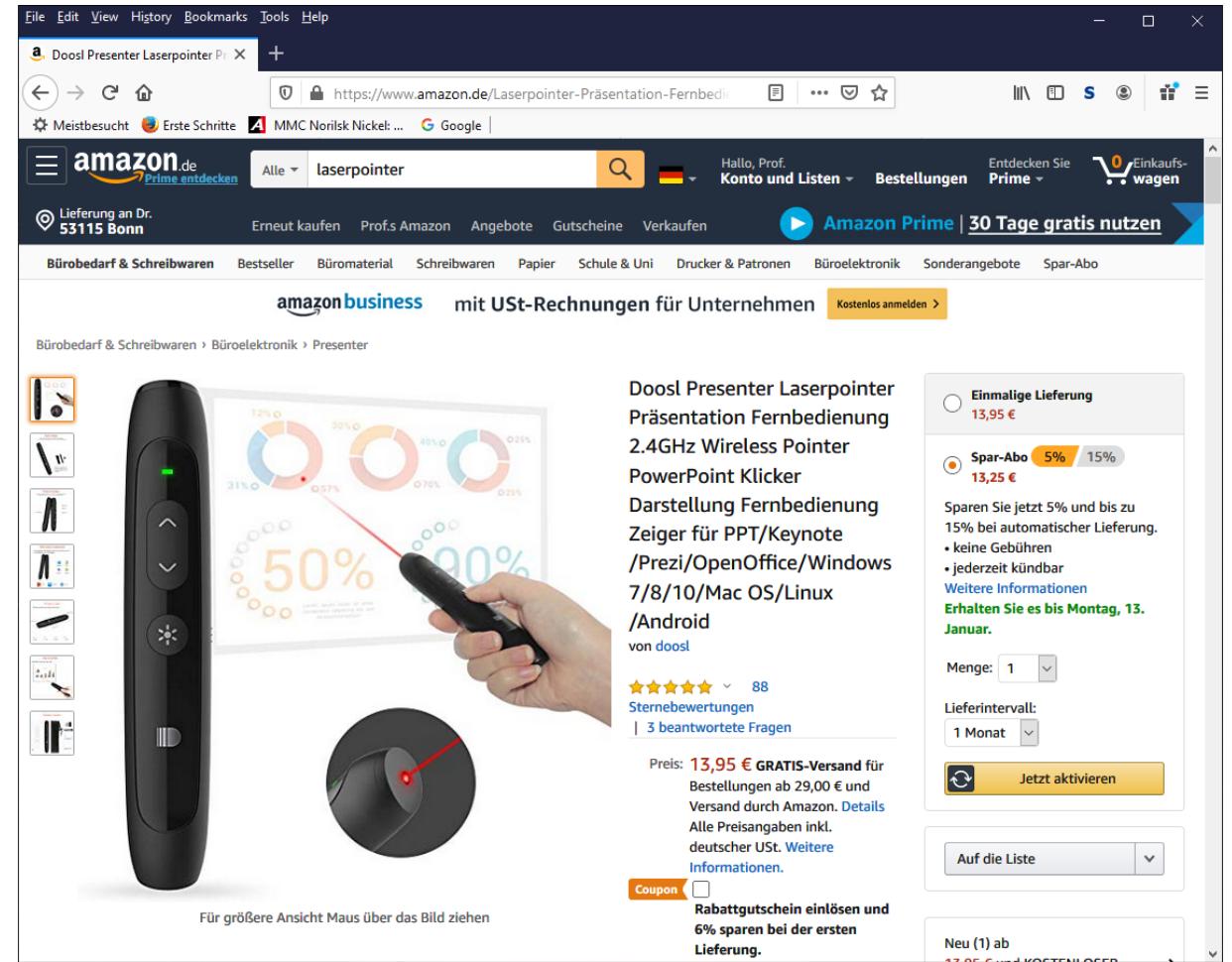
Grundlage vieler moderner Technologien, insbes. Halbleiter-, Laser- und Nanotechnologie



Smartphones



Supercomputer



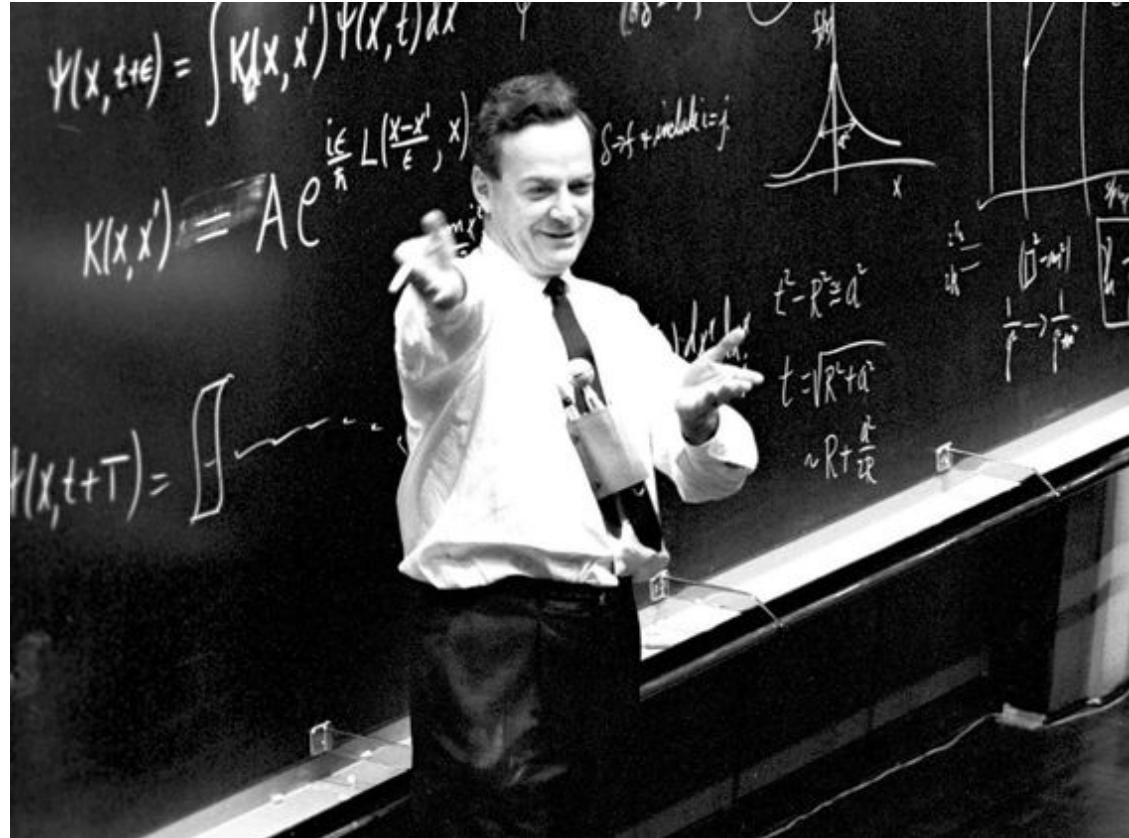
Doosl Presenter Laserpointer
Präsentation Fernbedienung
2.4GHz Wireless Pointer
PowerPoint Klicker
Darstellung Fernbedienung
Zeiger für PPT/Keynote
/Prezi/OpenOffice/Windows
7/8/10/Mac OS/Linux
/Android
von doosl

Preis: 13,95 € GRATIS-Versand für Bestellungen ab 29,00 € und Versand durch Amazon. Details Alle Preisangaben inkl. deutscher USt. Weitere Informationen.

Rabattgutschein einlösen und 6% sparen bei der ersten Lieferung.

Quantenphysik

Die Physik des ganz, ganz Kleinen - jenseits der Grenzen unserer Intuition...



<https://www.pinterest.de/kpings/richard-feynman/>

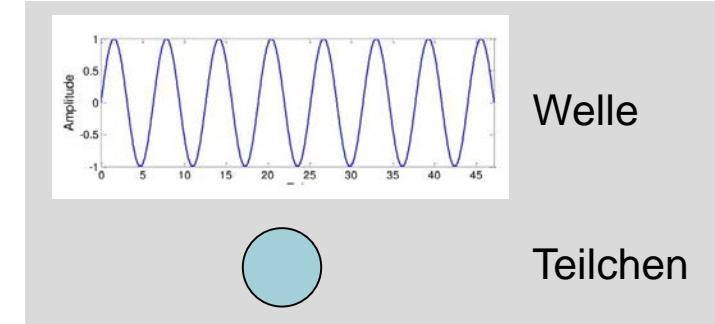
„Ich denke, man kann mit Sicherheit sagen,
dass niemand die Quantenphysik versteht“

Richard Feynman, 1918-1988
Physiker, Nobelpreisträger 1965

Quantenphysik

... Jenseits der Grenzen unserer Intuition...

- Quantenphysik entzieht sich unserer sinnlichen Wahrnehmung.
- Die Quanten-Teilchen können auch Wellen sein.
- Sie haben vielfältige Möglichkeiten zufälligen Verhaltens ... und nutzen diese auch !
- Erst bei Beobachtung („Messung“) „entscheiden“ sie sich und bleiben dann dauerhaft bei dieser Wahl!
- Verschränkte Quanten-Teilchen stehen in einer „**spukhaften Fernbeziehung**“ ... augenblicklich, also auch ohne jeglichen Zeitverlust.



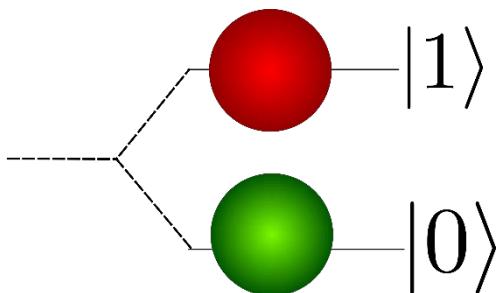
- Derartige Effekte können wir **nicht verstehen**.
- Wir können sie **nur hinnehmen**.

Die 2. Quantenrevolution: Quantencomputer

Vom Bit zum Quanten-Bit (Qubit)

Ein Quantencomputer arbeitet nach den Gesetzen der Quantenphysik:

- An die Stelle des Bits tritt das **Quanten-Bit**, das „Qubit“.
- Qubits sind beliebig manipulierbare **quantenmechanische Zweizustandsysteme**.



<https://de.wikipedia.org/wiki/Quantencomputer>

Zustände:

Quantenmechanische Zustandsvektoren in einem 2-dimensionalen komplexen Raum,
z.B. Spin eines Elektrons, Polarisation eines Photons, ...

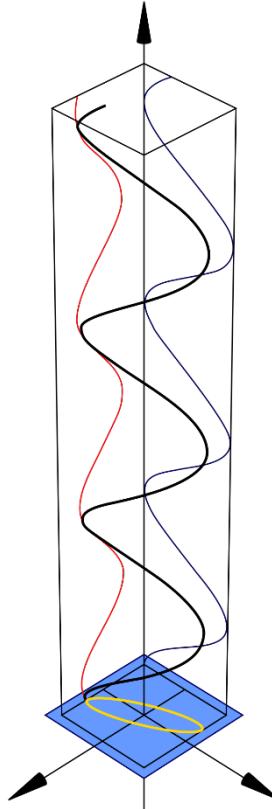
Klassische Bits: Entweder 0 oder 1.

Qubits:

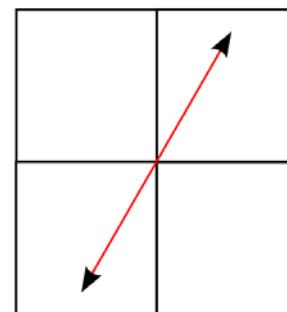
Beliebige Superpositionen aus 0 und 1
... auch 0 und 1 gleichzeitig, mit komplexen Wahrscheinlichkeiten

Die 2. Quantenrevolution

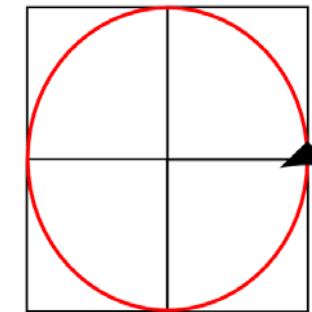
Von der bloßen Nutzung zur aktiven Kontrolle einzelner Quanten



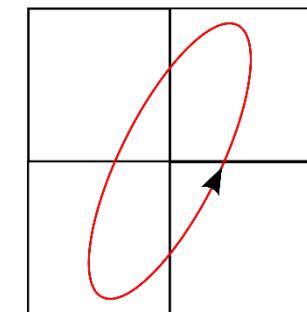
Polarisation eines Photons



Lineare
Polarisation



Zirkulare
Polarisation

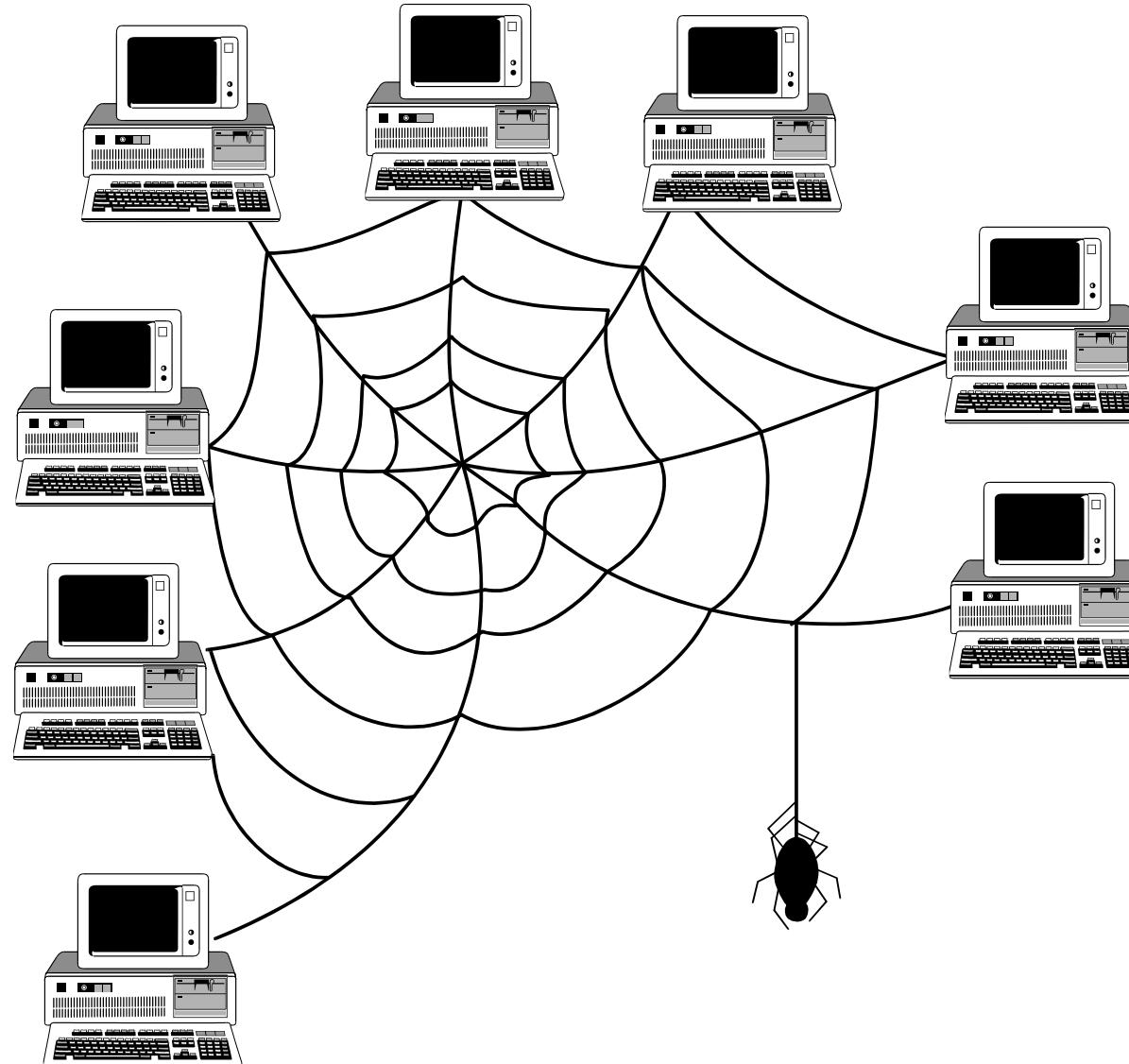


Elliptische
Polarisation

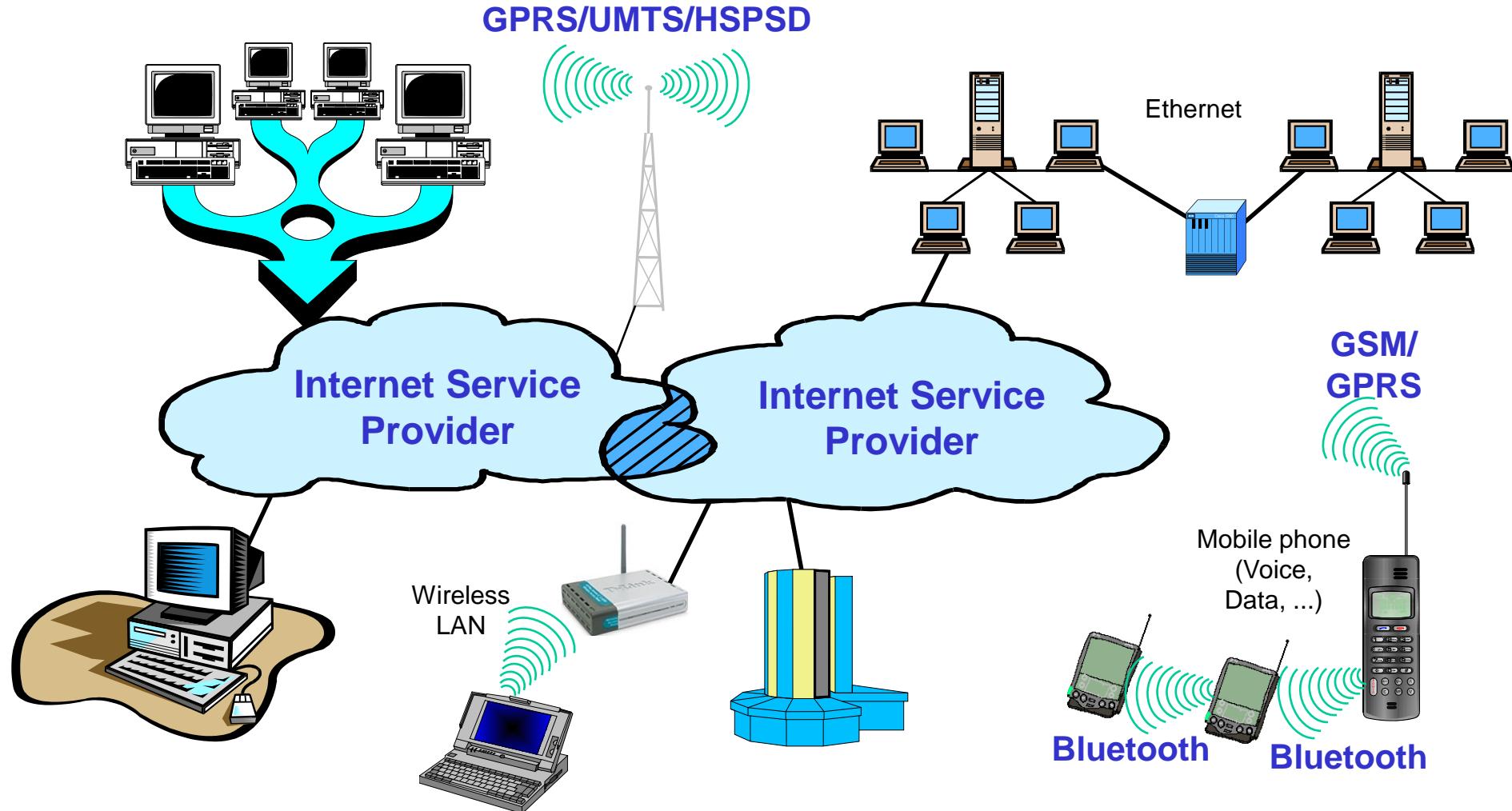
Quelle: <https://de.wikipedia.org/wiki/Qubit>

1.2.6. Computernetze und Verteilte Systeme

Heutige Computer sind häufig eingebunden in **komplexe Gesamtsysteme**.

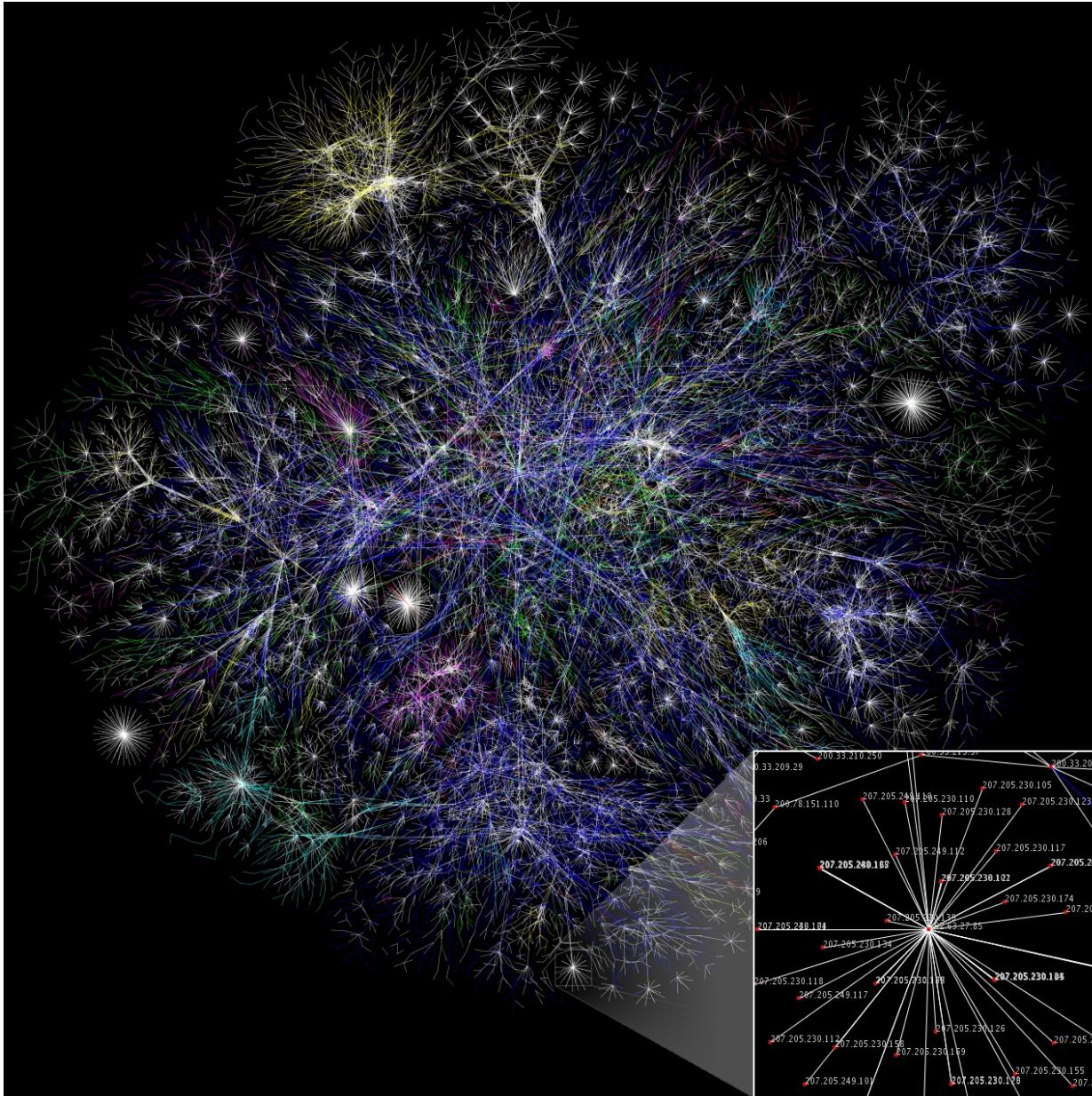


Das Internet



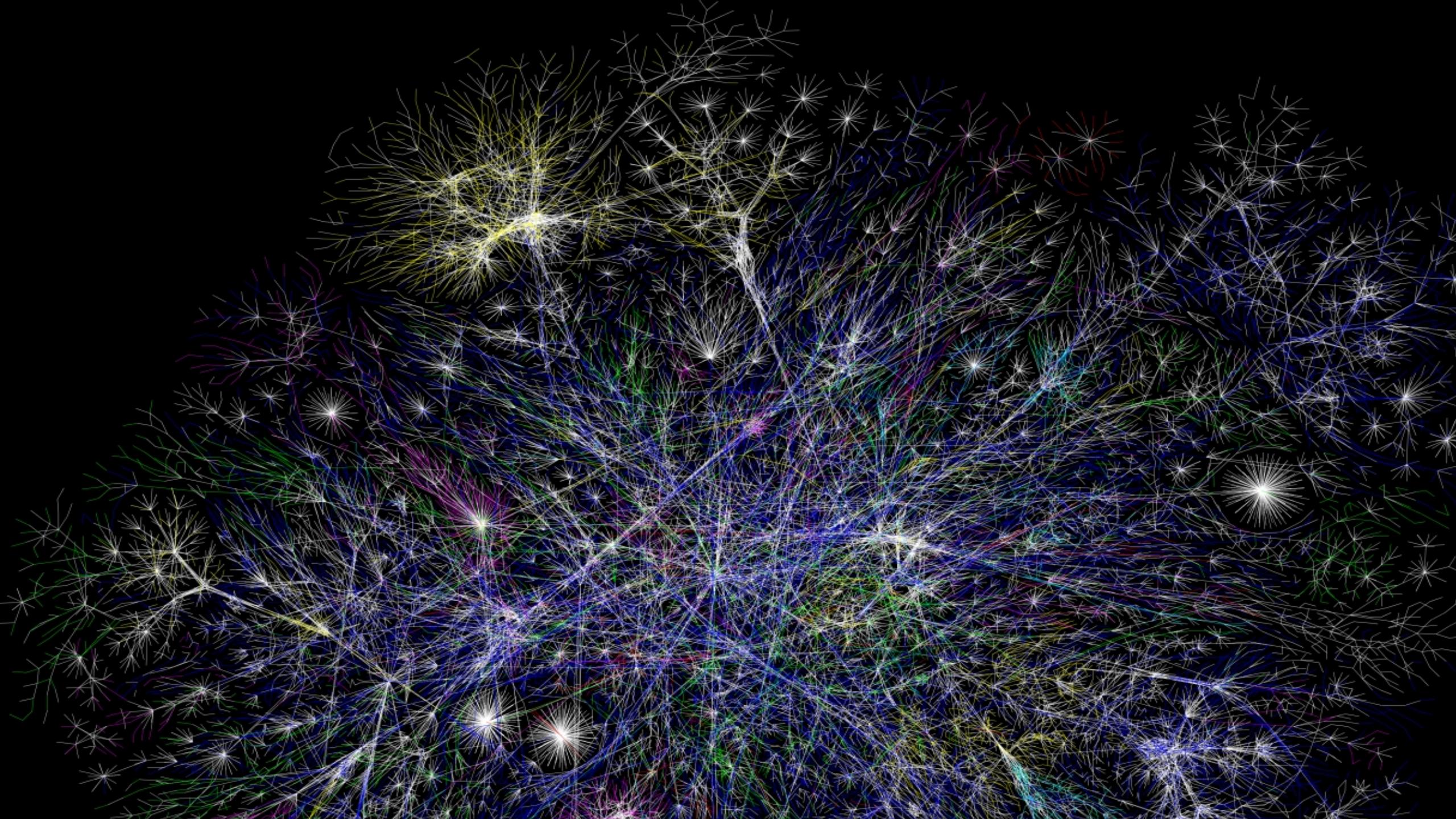
Die Kommunikationsmechanismen der Internet-Familie ermöglichen den Austausch von Daten und Steuerinformationen zwischen unterschiedlichsten Computer-Systemen und über unterschiedlichste Netze hinweg.

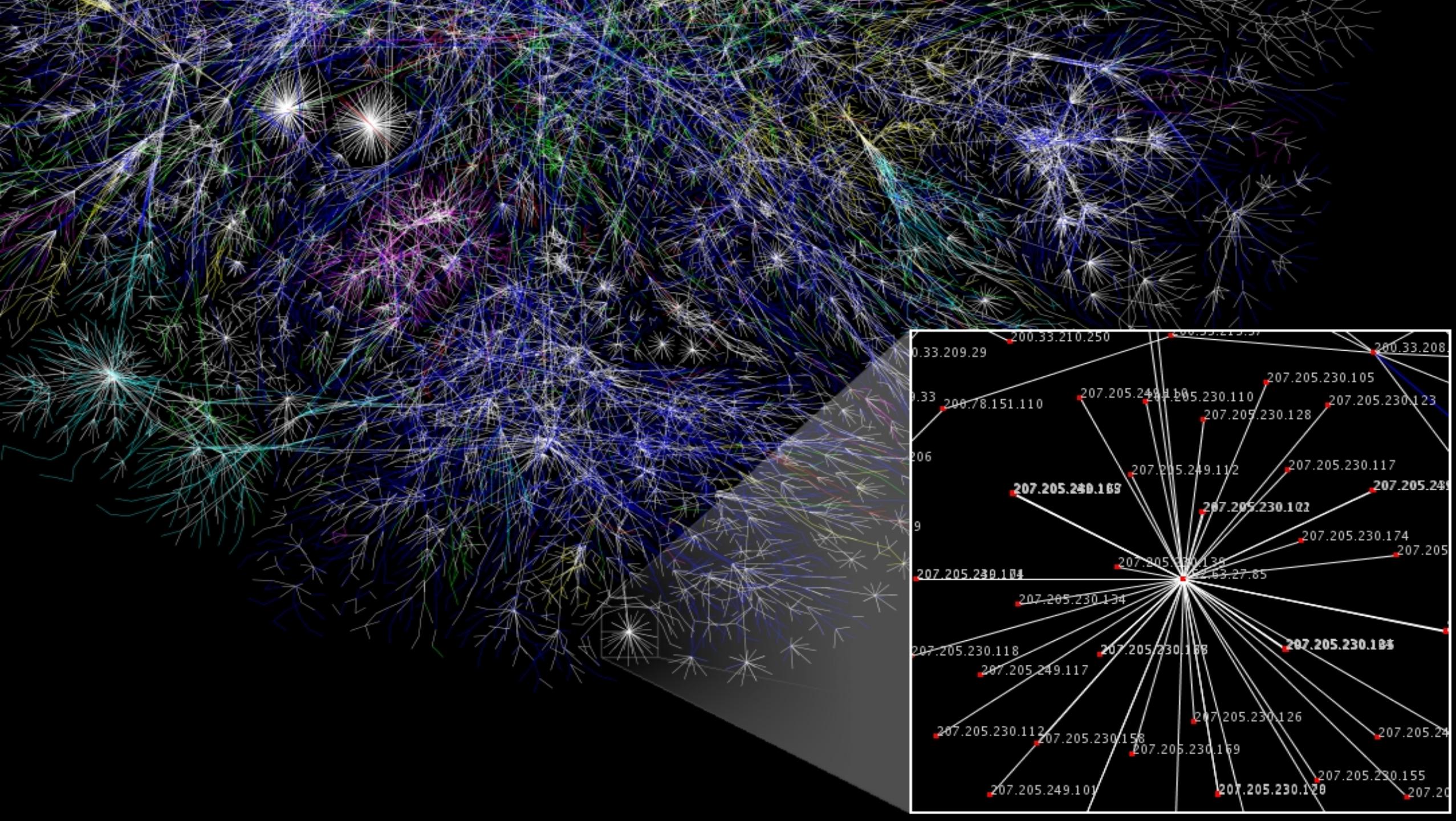
Visualisierung einer kleinen Auswahl von Internet-Routen (ca. 40.000)

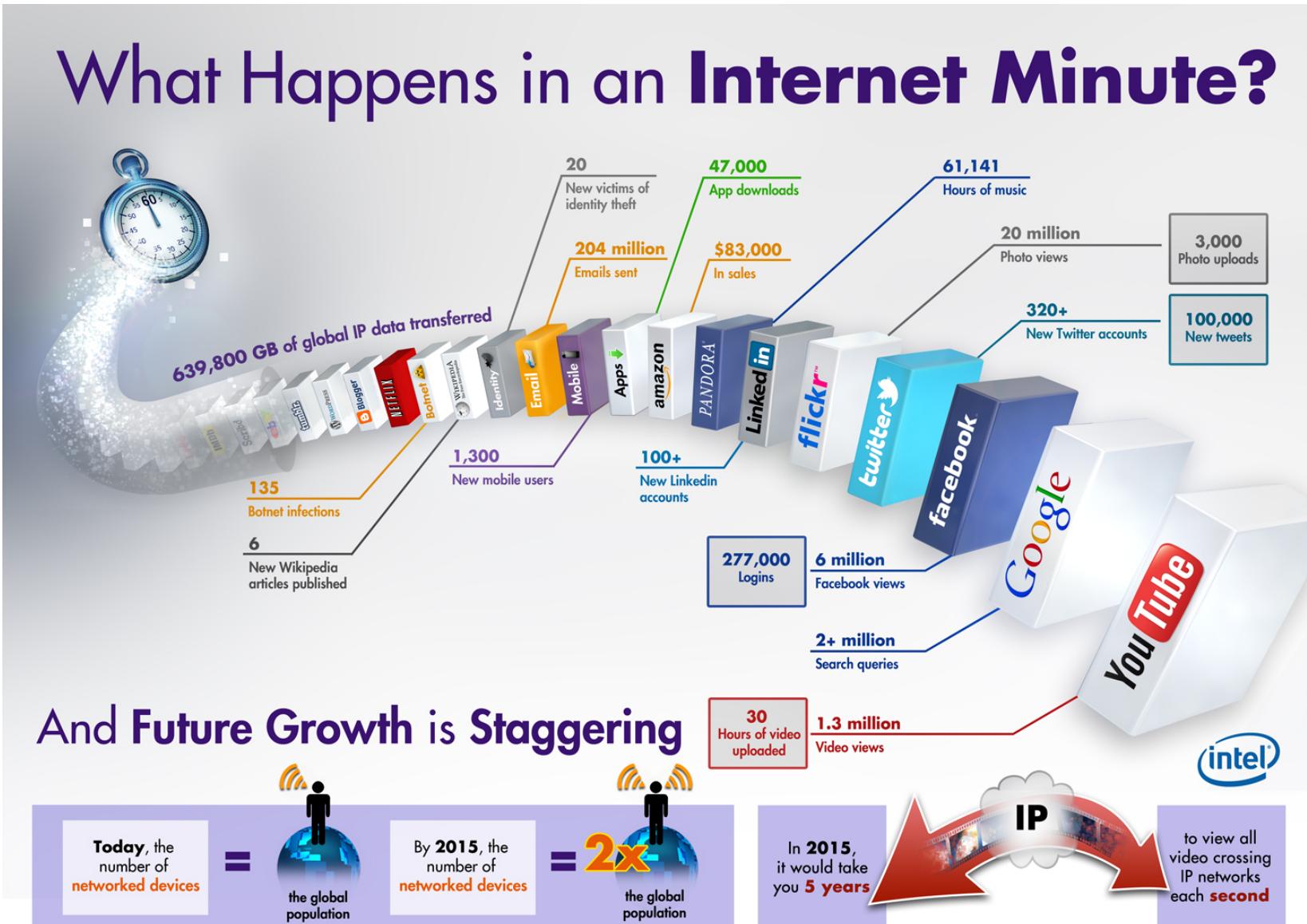


Length of lines indicative of the delay between the two nodes.

Source: www.citizenium.org







2018 This Is What Happens In An Internet Minute



2020 This Is What Happens In An Internet Minute



<https://www.allaccess.com/merge/archive/31294/infographic-what-happens-in-an-internet-minute>

<https://liefalcon.com/what-happens-in-an-internet-minute/>

	2013	2014	2015	2016	2017	2018	2019
 Facebook	461805 logins	600,000 logins	650,890 status updated	701,389 logins	835,620 logins	973,000 logins	1 million logins
 Instagram	38,000 photos uploaded	42,000 photos uploaded	85,000 photos uploaded	110,800 photos uploaded	150,950 scrolling IG	174,000 scrolling IG	347,222 scrolling IG
 Twitter	387,000 tweets	433,000 tweets	547,200 tweets	347,222 tweets	420,200 tweets	481,000 tweets	87,500 tweets
 Pinterest	2700 pins added	3400 pins added	4500 Pins added	6500 pins added	8200 pins added	9600 pins added	12,000 pines added
 Google	4.11 million search queries	4.19 million search queries	4.28 million search queries	4.12 million search queries	3.12 million search queries	3.7 million search queries	3.8 million search queries
 Vines	360 vines uploaded	450 vines uploaded	570 vines uploaded	650 vines uploaded	720 vines uploaded	700 vines uploaded	Shutting Down
 YouTube	103 hrs of YT content uploaded	306hrs of YT content uploaded	910hrs of YT content uploaded	100,50 hrs of YT content viewed	800,222 hrs of YT content viewed	1.5 million hrs of YT content viewed	4.5 million hrs of YT content viewed
 Apple and Google Store	37,000 apps downloaded	50,200 apps downloaded	72,400 apps downloaded	95,000 apps downloaded	185,222 apps downloaded	375,000 apps downloaded	390,030 apps downloaded
 Spotify	20 songs added	35 songs added	90 songs added	150 songs added	300 songs added	450 songs added	666 songs added
 Amazon	\$66,200 amazon sales	\$80,000 amazon sales	\$120,000 amazon sales	\$420,200amazon sales	\$700,000 amazon sales	\$862,823 amazon sales	\$996,959 amazon sales
 Emails	127,013,889 emails sent	127,013,889 emails sent	150 million emails sent	162 million emails sent	175million emails sent	187 million emails sent	188 million emails sent

LifeFalcon.com

Life
falcon

ChatGPT Sprints to One Million Users

Time it took for selected online services to reach one million users



* one million backers ** one million nights booked *** one million downloads
Source: Company announcements via Business Insider/LinkedIn



statista

<https://cdn.statcdn.com/Infographic/images/normal/29174.jpeg>

Verteilte Systeme

Zum Begriff „Verteilte Systeme“ (engl. “Distributed Systems“) gibt es unterschiedlichste Definitionen.

“We define a distributed system as one in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages. This simple definition covers the entire range of systems in which networked computers can usefully be deployed.”

Coulouris et al, “Distributed Systems – Concepts and Design”, Addison-Wesley, 2005, p. 2

“A distributed system is a collection of independent computers that appears to its users as a single coherent system.”

Tanenbaum, van Steen, “Distributed Systems – Principles and Paradigms”, Prentice-Hall, 2002

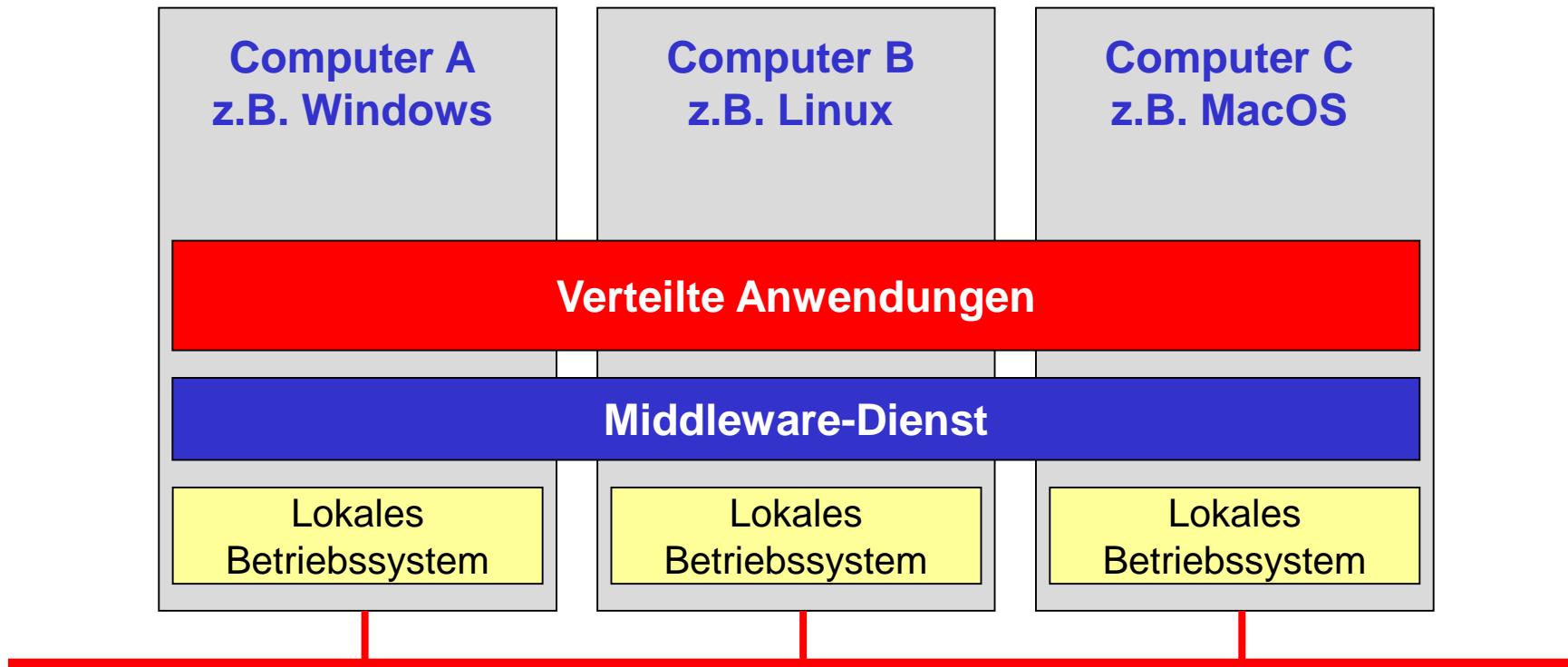
Der Eindruck eines „einzelnen, kohärenten Systems“ kann bei heterogenen Systemen nur erreicht werden, wenn in allen beteiligten Komponenten eine vereinheitlichende Schicht vorhanden ist, die sog.

„**Middleware**“.

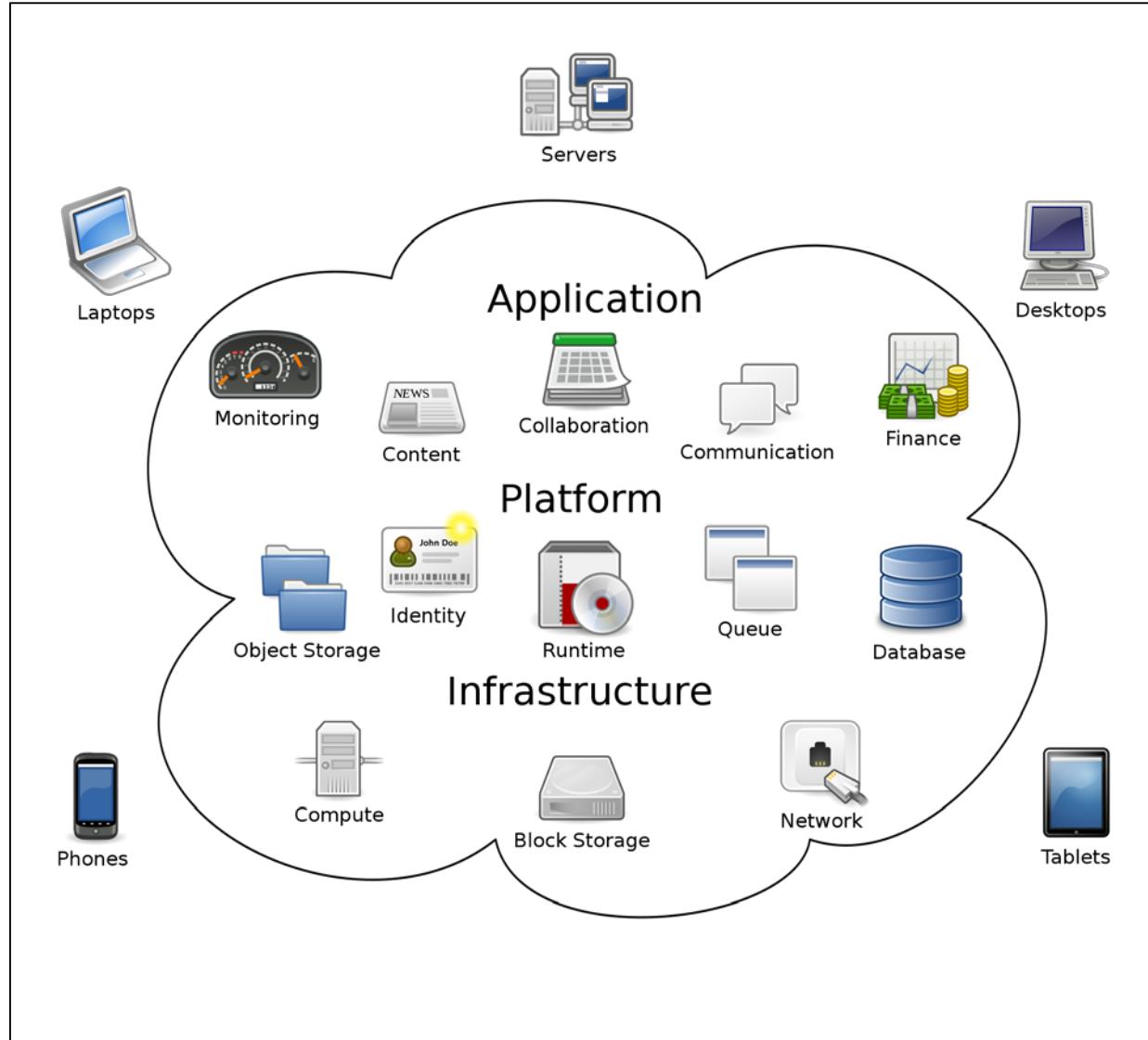
„Middleware“

Middleware ermöglicht den objektorientierten, direkten Datenaustausch zwischen Anwendungsprogrammen, die unter verschiedenen Betriebssystemen oder in heterogenen Netzen arbeiten. Beispiele für Middleware-Standards sind Corba, Java RMI, oder DCOM.

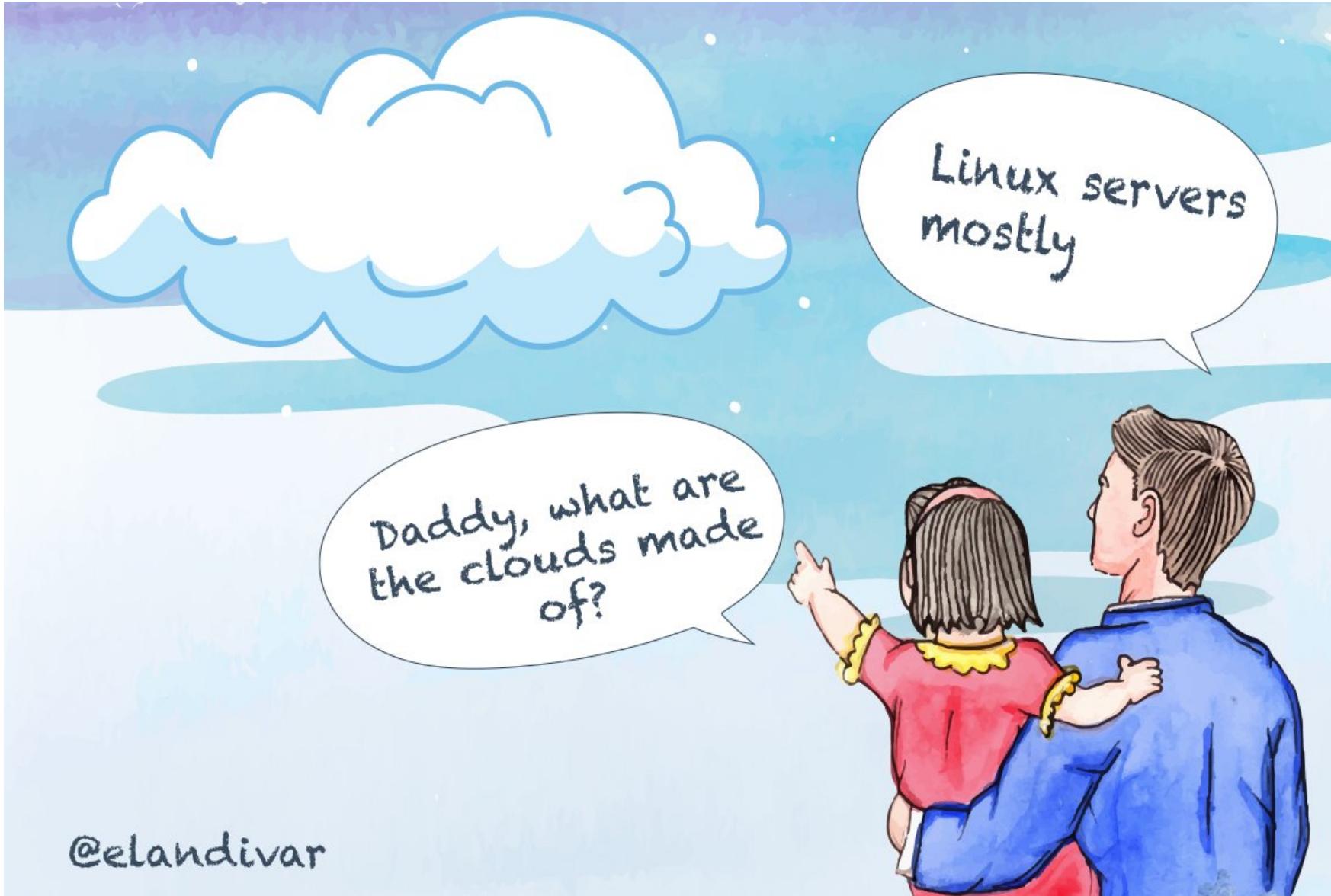
Fachlexikon „Der Brockhaus – Computer und Informationstechnologie“, 2003



„Cloud“



„Cloud“



@elandivar