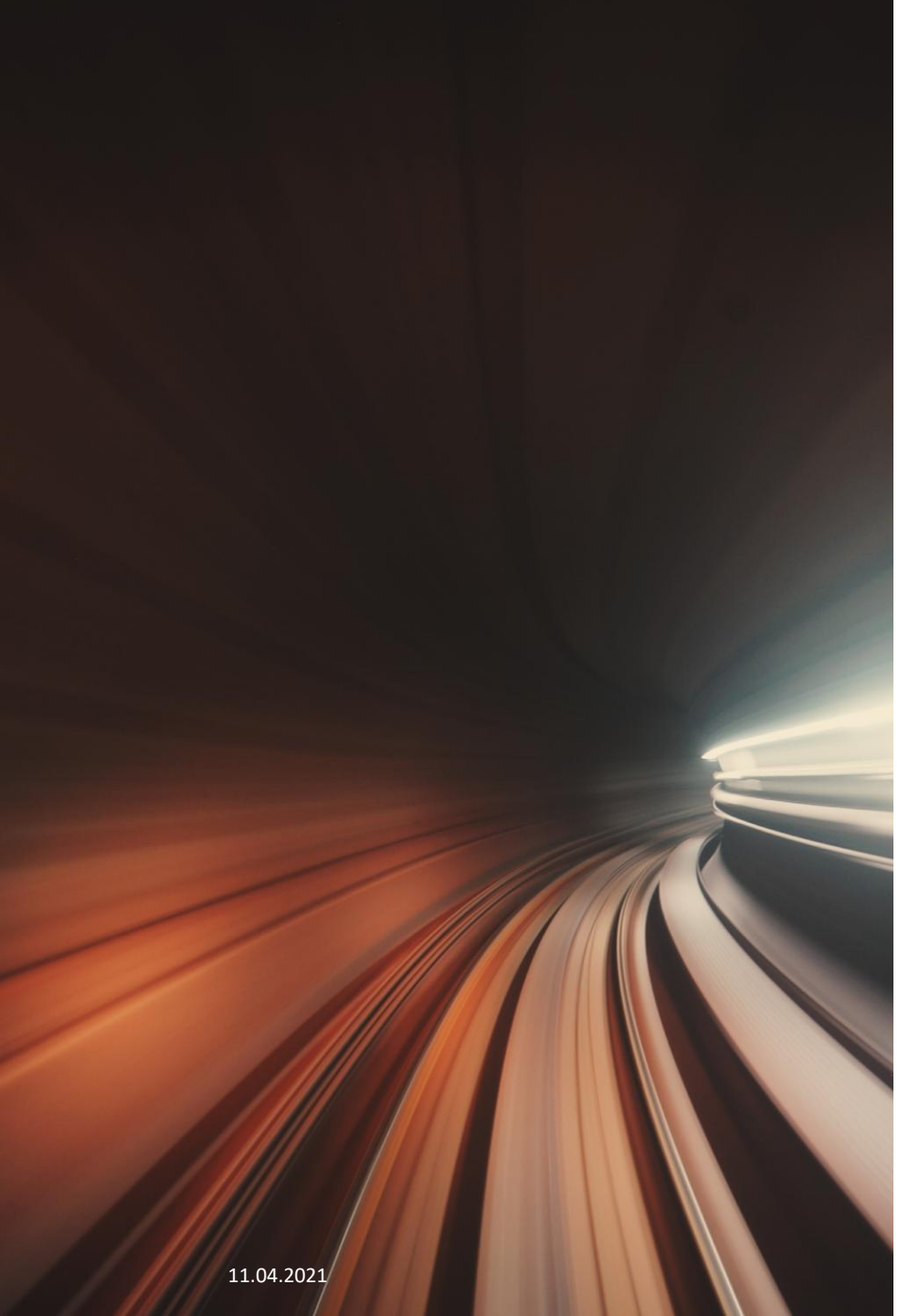
# Verteilte Systeme

PROF. ANDREAS HARTMANN – VERTEILTE ANWENDUNGEN



# Agenda

1. DEFINITION
2. ZIELE
3. EIGENSCHAFTEN

# Quellenverweise

Die Inhalte sind angelehnt an - und Zitate stammen, wenn nicht anders gekennzeichnet, aus:

- [Ta95]Tanenbaum, A. S.: Verteilte Betriebssysteme. Prentice-Hall, München, 1995.

# Definition

„Ein verteiltes System ist eine Sammlung voneinander unabhängiger Computer, die dem Benutzer des Systems den Eindruck vermitteln, es handle sich um einen einzigen Computer.“

[Ta95] S. 16

# Zwei Aspekte

Maschinen sind autonom

- (das sind sie bei verteilten Anwendungen auch)

Eindruck einer einzelnen Maschine

- (der entsteht bei verteilten Anwendungen ebenso)

Daher Schlussfolgerung:

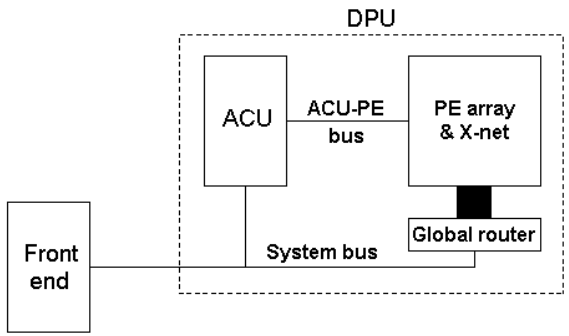
- Verteilte Anwendungen sind zumindest eine Untergruppe von verteilten Systemen.

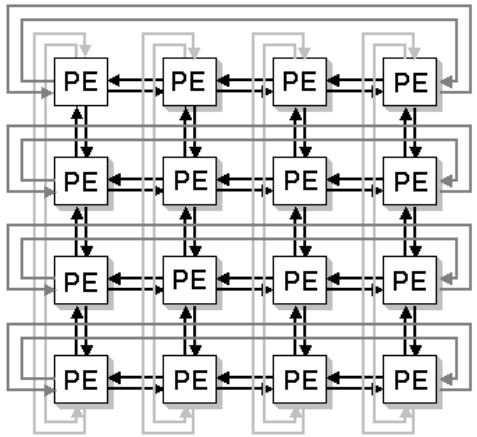
# Verteilte versus parallele Systeme

Parallel Systeme stellen einen Spezialfall von verteilten Systemen dar

Primäres (oder einziges) Ziel ist die maximale Geschwindigkeit einer einzelnen Problemlösung durch Einsatz vieler Prozessoren

- Bsp. MasPar




Quelle: By Dave Pape - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=4312254

# Verteilte Systeme

Welches Ziel verfolgen Verteilte Systeme?

Welche Arten gibt es?

Welche Eigenschaften haben sie?

Was davon lässt sich direkt auf Verteilte Anwendungen übertragen und was davon ist unterschiedlich?

# Ziele von verteilten Systemen

Wirtschaftliche Interessen

- Höhere Rechenleistung durch Verteilung auf viele Prozessoren
- Billigere Rechenleistung durch Einsatz preiswerter Prozessoren
- Flexiblere Rechenleistung durch bessere Skalierbarkeit

Inhärente Verteilung und dadurch höhere Zuverlässigkeit möglich

Beachten Sie: diese Aussagen stammen aus dem Jahr 1995!

# Ziele zusammengefasst

| Wirtschaftlichkeit | Geschwindigkeit | Inhärente Verteilung | Zuverlässigkeit | Inkrementelles Wachstum |
|---|---|---|---|---|
| • Mikroprozessoren bieten ein besseres Preis-Leistungs-Verhältnis als Mainframes | • Ein verteiltes System kann mehr Gesamtleistung bieten als ein Mainframe | • Einige Applikationen bedingen räumlich voneinander getrennte Maschinen | • Wenn eine Maschine ausfällt, kann das System als Ganzes weiterarbeiten | • Die Rechenleistung kann schrittweise erhöht werden |

# Vorteile gegenüber autonomen PCs

Gemeinsame Datennutzung

- Viele Benutzer können auf eine gemeinsame Datenquelle zugreifen (DB)

Gemeinsame Nutzung von Geräten

- Viele Benutzer können teure Peripheriegeräte teilen, z.B. Scanner/Plotter oder 3D Drucker

Kommunikation

- Vereinfachte und deutlich schnellere Kommunikation, z.B. Email im Vergleich zu Telefon/Post

Flexibilität

- Arbeitsauslastung wird kostengünstiger auf vorhandene IT-Ressourcen aufgeteilt

# Nachteile

## STAND 1995

- Software
  - wenig Software für verteilte Anwendungen
- Netzwerk
  - Netzwerke instabil (Überlastung) und weniger leistungsfähig
- Sicherheit
  - Da hat sich nichts geändert – Daten/Systeme werden leichter angreifbar

## STAND HEUTE

- Es gibt zwar deutlich mehr Software, jedoch sind Komplexität und technische Abhängigkeit exponentiell gestiegen.
- Sicherheit, Sicherheit, Sicherheit

# Konzepte verteilter Systeme

## Hardware

- SIMD
- MISD
- MIMD

## Software

- Netzwerk-Betriebssysteme
- Echte verteilte Systeme
- Multiprozessor-Timesharing-Systeme

# Klassifikationen nach Flynn

SIMD

- Single Instruction Stream, Multiple Data Stream
- Eine Befehlseinheit, mehrere Dateneinheiten
- Einige Supercomputer

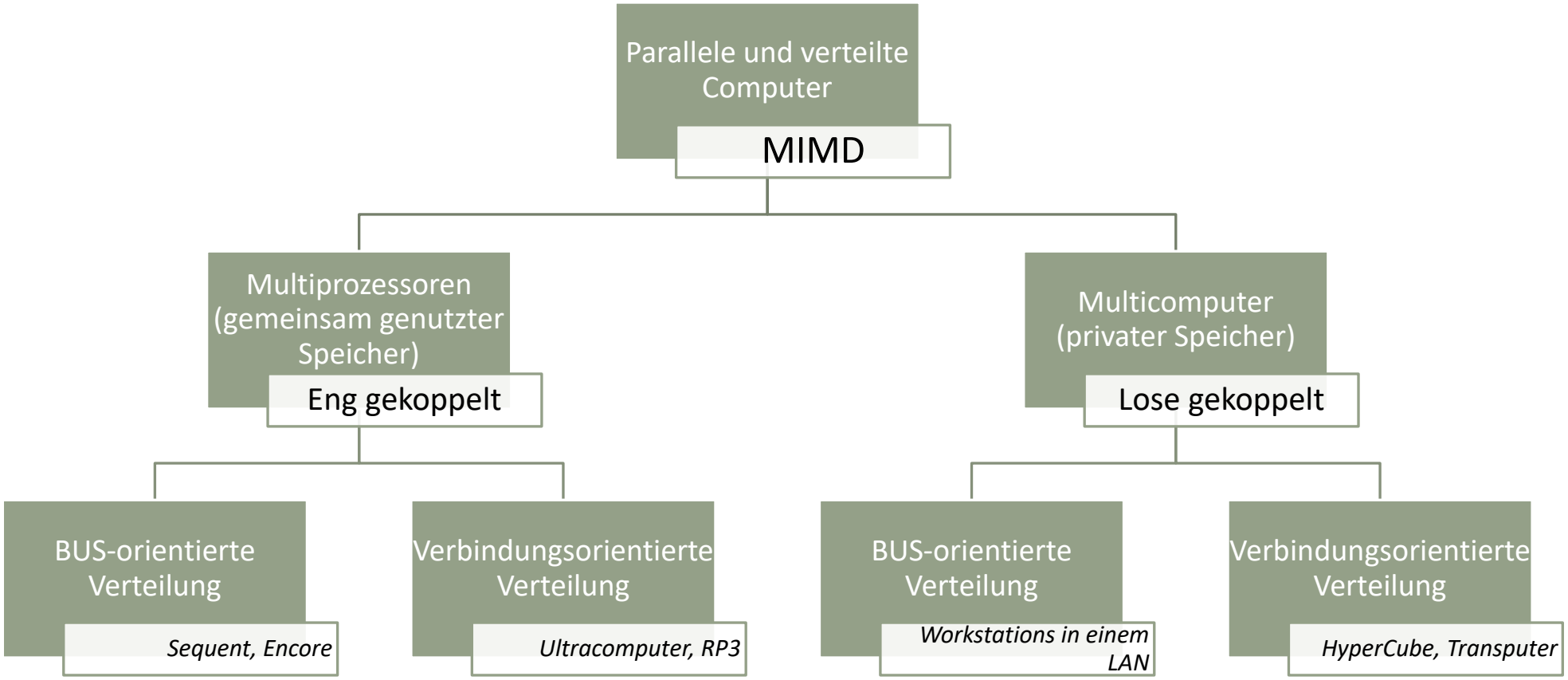
MISD

- Multiple Instruction Stream, Single Data Stream
- Nicht weiter verwendet

MIMD

- Multiple Instruction Stream, Multiple Data Stream
- Gruppe unabhängiger Computer (entspricht der Definition eines Verteilten Systems)

# Klassifikation nach Tanenbaum



# Auszug: BUS-orientierter Multiprozessor

Mehrere CPUs und ein Speichermodul über BUS angeschlossen

Z.B. ein entsprechendes Mainboard mit CPU-Steckkarten

- *Hinweis: Intel baut gerade wieder ähnliche Architekturen mit PCIe-Steckkarten

32 / 64 Adressleitungen und Datenleitungen

CPU gibt Speicheradresse auf Adressleitung und Speicher sendet Wort auf die Datenleitung

Problem: skaliert nicht, BUS schnell überlastet

Lösung: private Speicher, Cache-Strategien

Kohärente und Inkohärente Systeme

# Auszug: Verbindungsorientierter Multiprozessor

Multiprozessor mit mehr als 64 Prozessoren ist bei einem 64bit breiten BUS nicht möglich

Lösungsansätze sind z.B. *Kreuzschienenverteiler* oder ein *Omega-Switching-Netzwerk*

Vorteile: viele CPUs können gleichzeitig auf Speicher zugreifen

# Softwarekonzepte

Netzwerk-Betriebssysteme

- Lose gekoppelte Software auf lose gekoppelter Hardware
- Beispiel: Typische LAN-Struktur

Echte verteilte Systeme

- Eng gekoppelte Software auf lose gekoppelter Hardware
- „virtueller Einprozessor“ – Eindruck eines einzelnen Systems

Multiprozessor-Timesharing-Systeme

- Eng gekoppelte Software auf eng gekoppelter Hardware
- Beispiel: UNIX-Timesharing auf 30 MIPS CPUs (Multiprozessor)

# Eigenschaften und Unterschiede

| Aspekt | Netzwerkbetriebssystem | Verteiltes Betriebssystem | Multiprozessor Betriebssystem |
|---|---|---|---|
| Eindruck eines virtuellen Einprozessor-Systems? | Nein | Ja | Ja |
| Dasselbe Betriebssystem auf allen Maschinen? | Nein | Ja | Ja |
| Anzahl der Instanzen des Betriebssystems? | N | N | 1 |
| Kommunikation | Gemeinsam genutzte Dateien | Nachrichten | Gemeinsam genutzter Speicher |
| Netzwerkprotokolle? | Ja | Ja | Nein |
| Einzelne Prozesswarteschlange? | Nein | Nein | Ja |
| Wohl-definierte Semantik für gemeinsame Nutzung von Dateien? | Meist nicht | Ja | Ja |

# Design-Aspekte

- Transparenz
- Flexibilität
- Zuverlässigkeit
- Performance
- Skalierbarkeit

# Transparenz

Ortstransparenz
- Nutzer erkennen nicht, wo sich (einzelne) Ressourcen befinden

Migrationstransparenz
- Ressourcen können verlagert werden, ohne dass sich ihr Name verändert

Replikationstransparenz
- Nutzer erkennen nicht, wie viele Kopien einer Ressource es gibt

Nebenläufigkeitstransparenz
- Nutzer können Ressourcen automatisch gemeinsam nutzen

Parallelitätstransparenz
- Aktivitäten (Transaktionen) können parallel ausgeführt werden, ohne dass Nutzer es bemerken

# Flexibilität

Wer macht was?

Je weniger Funktionen in einer Komponente gebündelt sind, umso flexibler können Funktionen eingesetzt werden.

Beispiel: Dateisysteme

| Monolithischer Kernel (UNIX, Linux) | Mikrokernel |
| --- | --- |
| • Dateisysteme über Kernel fest vorgegeben<br>• Anwenderprozesse kommunizieren direkt mit Kernel | • Kernel mit minimalen Funktionen<br>• Dateisysteme über Server implementiert<br>• Anwenderprozesse kommunizieren mit Server, Server mit Kernel<br>• Deutlich flexibler |

# Zuverlässigkeit

Wichtiger Aspekt: Verfügbarkeit

Bei redundanten Services im verteilten System gilt:

- Ausfallwahrscheinlichkeit = Boolesche ODER-Verknüpfung aller Instanzen
- Beispiel: 4 File-Server (redundant) mit einzelner Ausfallwahrscheinlichkeit von 0,05 ergibt im verteilten System eine Wahrscheinlichkeit von $0{,}05^4 = 0{,}000006$ also Verfügbarkeit von 0,999994.

Bei Abhängigkeiten kehrt sich das Argument um:

- Ausfallwahrscheinlichkeit = Boolesche UND-Verknüpfung aller abhängigen Instanzen

Weitere Aspekte sind: Konsistenz, Sicherheit, Fehlertoleranz

# Performance

Antwortzeit, Durchsatz, Systemnutzung, Netzwerkkapazität

Bspw. ist zu berücksichtigen, dass Nachrichten über (längere) Netzwerkverbindungen eine begrenzte Geschwindigkeit haben. Daher werden zeitkritische Infrastrukturen in physischer Nähe gebaut – vergleiche Börse Frankfurt.

Lösungsansätze beziehen sich daher auf die optimale Größe der verteilten Komponenten, bzw. die Granularität

- Fein-granulare Parallelität
- Grob-granulare Parallelität

# Skalierbarkeit

Potenzielle Engpässe für große verteilte Systeme sind:

- Zentralisierte Komponenten (z.B. zentraler Emailserver)
- Zentralisierte Tabellen (z.B. ein einziges Nutzerverzeichnis)
- Zentralisierte Algorithmen (z.B. bestimmte Analysetechniken)

Dezentralisierte Algorithmen besitzen folgende Eigenschaften:

- Keine Maschine besitzt vollständige Informationen über den Systemstatus
- Maschinen treffen nur Entscheidungen, die auf lokalen Informationen basieren
- Der Ausfall einer Maschine hat keinen Einfluss auf den Algorithmus
- Es gibt keine implizite Annahme, dass eine globale Uhr existiert

# CAP-Theorem

Auch bezeichnet als Brewer‘s theorem nach Eric Brewer

Bei einem verteilten Datenspeicher können nur zwei der drei folgenden Eigenschaften garantiert werden:

- (*Consistency*) Konsistenz – Lesezugriffe liefern den aktuellsten Wert oder einen Fehler
- (*Availability*) Verfügbarkeit – Jeder Request erhält eine Antwort (die kein Fehler sein darf)
- (*Partition tolerance*) Ausfalltoleranz – das System arbeitet weiter, auch wenn Nachrichten ausfallen oder sich verspäten

# CAP - Erläuterung

In verteilten Systemen kann jederzeit das Netzwerk oder die Verbindung ausfallen. Daher kann P nie garantiert werden!

Es geht um den Fall, dass P eintritt. Dann muss entschieden werden:

- C – Operation abbrechen; Konsistenz bewahren, Verfügbarkeit verzichten
- Beispiele sind DBMS mit ACID-Philosophie
- A – Operation (mit Fehler) fortführen; Verfügbarkeit bewahren, Konsistenz verzichten
- Beispiele sind DBMS mit BASE-Philosophie

Solange das Netzwerk korrekt arbeitet, sind C + A gewährleistet

Eine Weiterführung ist das PACELC Theorem

# ACID

Atomicity (Atomizität)

- Transaktionen werden entweder ganz oder gar nicht ausgeführt.
- Bei einem Fehler werden alle Änderungen rückgängig gemacht.

Consistency (Konsistenz)

- Daten sind vor und nach der Änderung konsistent.
- Validierungen werden nicht verletzt.

Isolation (Isolation)

- Transaktionen sind voneinander getrennt.

Durability (Dauerhaftigkeit)

- Änderungen einer Transaktion werden gespeichert und stehen auch nach Abstürzen noch zur Verfügung.

# BASE

Basically Available

- basic reading and writing operations are available as much as possible (using all nodes of a database cluster), but without any kind of consistency guarantees (the write may not persist after conflicts are reconciled, the read may not get the latest write)

Soft state

- without consistency guarantees, after some amount of time, we only have some probability of knowing the state, since it may not yet have converged

Eventually consistent

- If the system is functioning and we wait long enough after any given set of inputs, we will eventually be able to know what the state of the database is, and so any further reads will be consistent with our expectations

Quelle: https://en.wikipedia.org/wiki/Eventual_consistency

# Uhren

Stellen Sie sich vor, ich würde auf OneDrive die Uhr *vorstellen* und damit einen „alten“ Zustand auf alle Clients synchronisieren?!

- Synchronisationsregel: neuere Dateien überschreiben alte Dateien
- Ja, dass ist ein Angriffsvektor in verteilten Systemen!

In verteilten Systemen ist es zwingend notwendig, die Uhren zu synchronisieren.

Aber wie?

# Quarzkristalle – logische Uhren

Logische Uhren messen keine Uhrzeit sondern entsprechen eher einem Timer, der nach vorne zählt

Verantwortlich ist ein Uhren-Quarzkristall, der auf einem speziellen Chip implementiert ist

Der Quarzkristall oszilliert in einer stabilen Frequenz (typisch sind hier 32.768 Hz), was entsprechend genormt wird

Somit wird ein Takt erzeugt, der z.B. jede Sekunde einen Zähler weiterzählt

Benötigt: Synchronisation mit einer Echtzeituhr

# RTC - Echtzeituhr

Physikalische Uhr

Früher: astronomische Uhren durch Sonnenbeobachtung (GMT)

Heute: Cäsium-Atom-Uhren (UTC – Universal-Coordinated-Time)

- Es werden letztlich Zustandsübergänge der Cäsium-133-Atome *gezählt*

BIH (Bureau International de L`Heure) erstellt einen Mittelwert aus verschiedenen Uhren

Regelmäßige Schaltsekunden, da das Sonnenjahr immer länger wird

Beachten: RTC in der technischen Informatik repräsentiert in der Regel nur einen Timer, der mit einer Echtzeituhr synchronisiert wird (keine Cäsium-Atom-Uhr)

# Zeitsynchronisation

Früher musste man die Uhrzeit zum Systemstart manuell eingeben

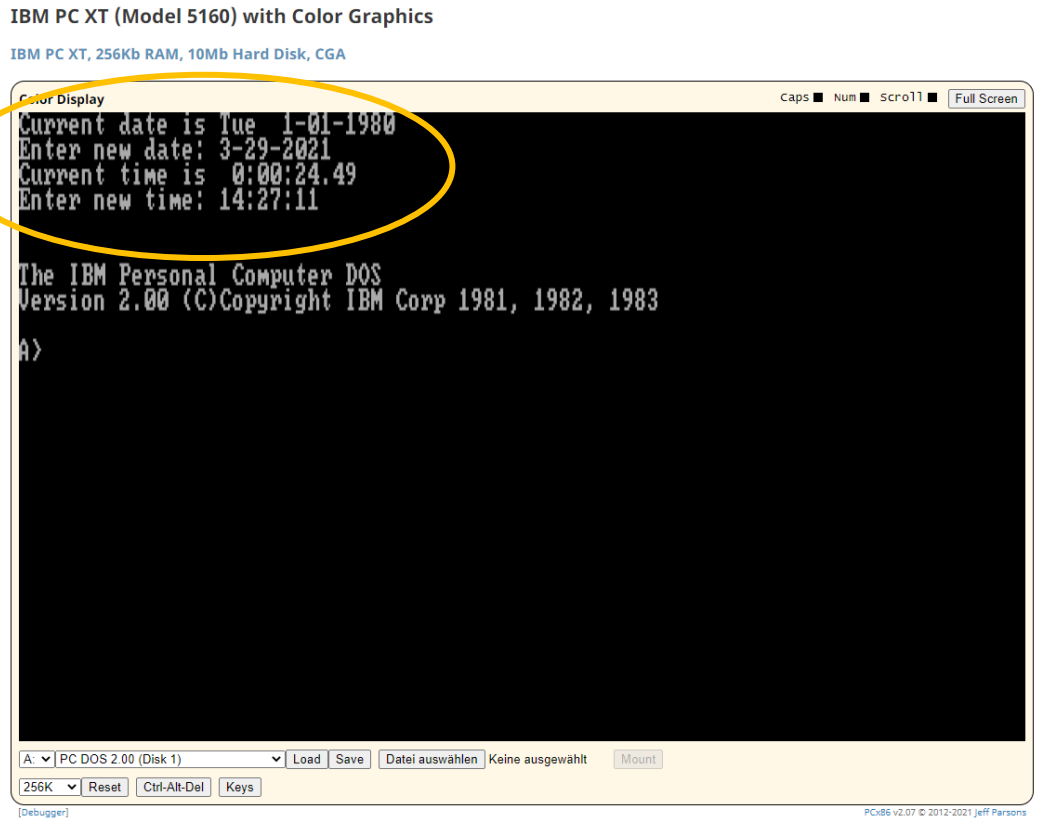
Später gab es Funksignale mit der aktuellen UTC, z.B. WWV vom NIST

NIST = National Institute of Standards and Technology

Heute betreiben NIST und andere Anbieter entsprechende UTC-Zeitserver im Internet (NTP-Server)

Damit können die lokalen Timer (Software-Uhren, Hardware-Uhren) mit der aktuellen Echtzeit (UTC) synchronisiert werden

Quelle: https://www.nist.gov/pml/time-and-frequency-division/time-distribution/internet-time-service-its




Quelle: https://www.pcjs.org/machines/pcx86/ibm/5160/cga/

# Probleme mit der Zeitsynchronisation

Übertragungszeit vom Zeitserver zur Maschine (Leitungslänge)

Übertragungsart, so kann es bei Ethernet zu Kollisionen kommen und die Auslieferung der Nachrichten verzögern

Verarbeitung der lokalen Maschine, so kann der Prozessor erst andere Aufgaben erledigen, bevor er die Nachricht mit der Zeit verarbeitet

Darüber hinaus: Quarze arbeiten nicht exakt gleich; es muss also immer wieder mit externen Quellen synchronisiert werden

# Andere Services des NIST

ACTS – Automated Computer Time Service (analoge Telefonleitung)

GPS Time Transfer

ITS - Internet Time Service

Radio Station WWV auf 5,10,15 MHz

Radio Station WWVB und WWVH

…

# Hausaufgabe zur Vertiefung des Stoffs

Vergleichen Sie Ziele, Vorteile, Nachteile und Designaspekte von verteilten Systemen nach Definition von Tanenbaum mit modernen Cloud-Lösungsarchitekturen, z.B. Microservices!