E Einführung

Bis Mitte der achtziger Jahre:

- zentralisierte Systeme mit einfachen ASCII-Terminals.
- Timesharing-Prinzip
- Rechenleistung am jeweiligen Terminal abhängig von der Anzahl der eingeloggten Benutzer.

Zwei grundlegende Entwicklungen führten zur Abkehr von diesem Konzept:

- Die technische Möglichkeit, leistungsfähige Mikroprozessoren in großen Stückzahlen zu produzieren. Das erlaubte die Herstellung von preiswerten, leistungsfähigen Computern.

- Die Entwicklung von kostengünstigen Hochleistungsnetzen.

Dadurch sind Systeme möglich,

- die nicht nur auf einen Prozessor, lokalen Speicher und eigene Peripherie eingeschränkt sind,

- sondern sich sowohl Hardware-, wie auch Software-Ressourcen, teilen können.

Informelle Definition: **Verteiltes System** (nach Andrew Tanenbaum)

Ein verteiltes System ist eine Kollektion unabhängiger Computer, die den Benutzern als ein Einzelcomputer erscheinen.

- impliziert, daß die Computer miteinander verbunden sind und
- die Ressourcen wie Hardware, Software und Daten gemeinsam benutzt werden.
- Es herrscht eine einheitliche Sicht auf das Gesamtsystem vor.

Definition: **Verteiltes System** (nach Leslie Lamport)
Ein verteiltes System ist ein System, mit dem ich nicht arbeiten kann, weil irgendein Rechner abgestürzt ist, von dem ich nicht einmal weiß, daß es ihn überhaupt gibt.

- Arten der Verteilung:

  1. Hardwarekomponennten
  2. Last
  3. Daten
  4. Kontrolle (Betriebssystem)
  5. Verarbeitung (verteilte Ausführung einer Anwendung)

E.1 Charakteristika verteilter Systeme

E.1.1 Verteilte Systeme versus zentralisierte Rechner

Vorteile sind:

- Wirtschaftlichkeit

  ➔ 'Berechnung stark parallelisierbarer Aufgaben (z.B. Wetterprognosen, finite Elemente,Erdbebenvorhersagen...) ' Leistung eines Superrechners oft darstellbar durch **n** Standardrechner aber: n*Preis Standardechner << Preis Superrechner

- Geschwindigkeit

  ➔ Statt einem Server und m-Clients nun n-Server und m-Clients ➔ kürzere Antwortzeiten bei DB-Systemem

- Inhärente Verteilung

  ➔ Sich aus der Problemstellung ergebende lokale Verteilung der Aufgaben ergeben lokale Systeme vor Ort, die die 'vor-Ort' Aufgaben lösen. ➔ Aufgabengrenze ←➔ Teilsystemgrenze ➔ Durch Autonomie der Teilsysteme Steigerung der Verfügbarkeit; 'Natürlichkeit des Designs'

- Zuverlässigkeit

  ➔Z.B. durch Replikation der Daten oder der transparenten online Verlagerung von Anwendungen von einem System auf ein Anderes die Erhöhung der Verfügbarkeit der gesamten Anwendung

- Wachstumspotential, Skalierbarkeit
  ➔ 'schrittweises Ergänzen statt Ersetzen' bei höheren Leistungsanforderungen

E.1.2 Verteilte Systeme versus unabhängige Einzelrechner

Vorteile sind:

- **Ressourcen-Sharing**

  ➔ gemeinsames Dateisystem, Datenbank

- **'neue Anwendungen' ;-))**

  ➔ Client/Server (komplexe GUI und Vorverarbeitung Vor-Ort beim Anwender aber Datenbank zentral);
  ➔ Multitier-Anwendungen (Client/Server1/BackgroundServer) Server1 z.B. Firewall;
  ➔ verteilte objektorientierte/komponentenbasierte Anwendungen (wie Client/Server aber auch der Client ist mal Server und umgekehrt)

- **Flexibilität**

  ➔ Ein 'verteiltes System' hat i.d.R. eine homogene Administrationsmöglichkeit im Laufzeitsystem mit dabei ➔ Lastverteilung, Ortstransparenz

- **Einzelrechnergrenze == primäre Hardware+Softwaregrenze;**

  Aber: 'verteiltes System' besteht aus Teilsystemen die miteinander kommunizieren aber oft nicht unbedingt mit Einzelrechnergrenzen übereinstimmen müssen.

- **'verteiltes System' Grenzen <> Einzelrechnergrenzen;**

Probleme/Nachteile verteilter Systeme:

- **Softwareentwicklung**

  ➔ komplex durch Größe und Nebenläufigkeit der Teilsysteme

- **Netzwerke**

  ➔ Zuverlässigkeit und damit Fehlertoleranz
  ('wann ist was sicher einmal angekommen und nur einmal ausgeführt worden'

- **Sicherheit**

  ➔ Authorisierung, Authentifizierung, Abhörsicherheit ('Verantwortung durch rechtsgültige Unterschriften')

E.2 Hardwarekonzepte verteilter Systeme

Flynn´sche Klassifikation [Fly72]:

- Single Instruction Single Data Stream (SISD)
- Single Instruction Multiple Data Stream (SIMD)
- Multiple Instruction Single Data Stream (MISD)
- Multiple Instruction Multiple Data Stream (MIMD)

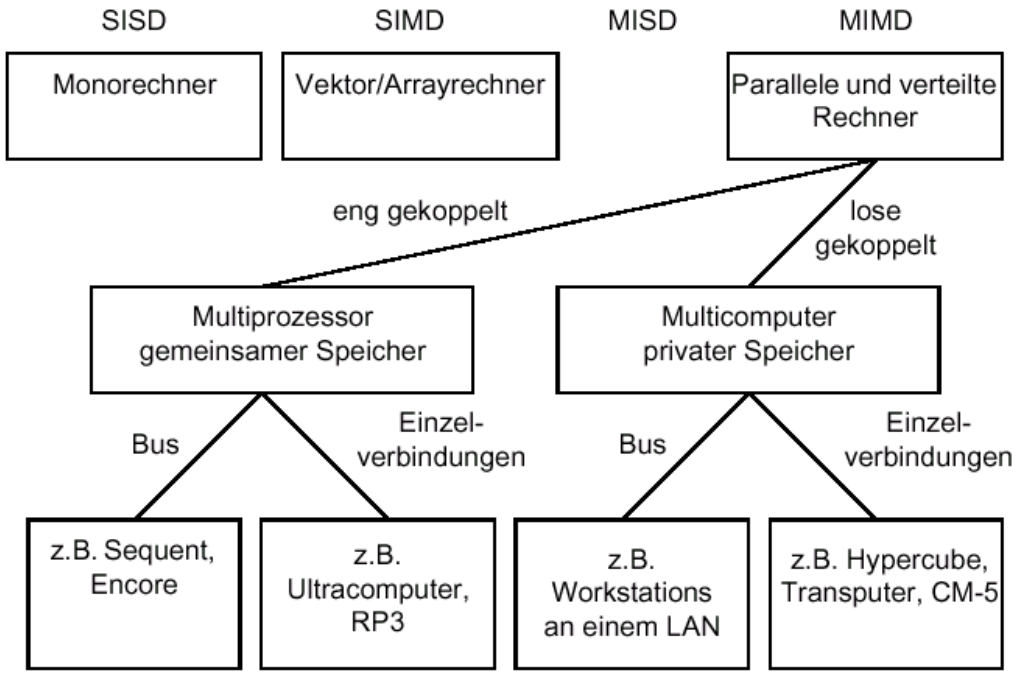

Bild 1.1 Taxonomie von Mehrrechnersystemen

E.3 Softwarekonzepte verteilte Systeme

Einteilung oder Klassifikation von verteilten Softwarekonzepten ist nur recht unzureichend möglich. Hier ein Versuch:

- *Lose gekoppelt*
  - Anwendungen kommunizieren nur kurzfristig mit einer Gegenstelle, im Vergleich zur Gesamtdauer der Anwendung. Z.B. gemeinsames Nutzen eines Druckers, oder eines Dateisystems.
  - Bestimmte Funktionen sind nicht möglich, wenn das Netz ausfällt, jedoch können die Einzelrechner im allgemeinen weiterarbeiten. ➔Oft hierzu der Begriff „**Verteiltes System**“.
- *Eng gekoppelt*
  - Mehrere Anwendungen arbeiten simultan an einem gemeinsamem Problem und teilen sich Zwischenergebnisse mit, die für die weitere Arbeit unabdingbar sind.
  - Exemplarische Problemstellungen dafür wären Schach, Matrixmultiplikation, Wetterberechnungen oder komplexe Simulationen. ➔ Oft hierzu der Begriff „**Paralleles System**“.

Vorlesungs-Definition: **Verteilte Anwendung**

Eine verteilte Anwendung ist eine Anwendung, die auf mehreren Rechnern bzw. Prozessoren abläuft und unter diesen Informationen austauscht.

Vorlesungs-Definition: **Verteiltes System**

Ein verteiltes System ist ein System mit mehreren Rechnern bzw. Prozessoren, auf denen mindestens eine verteilte Anwendung lauffähig installiert ist.

E.3.1 Betriebssystemunterstützung für verteilte Systeme

Multiprozessor-Timesharing-Systeme

- Jeder Prozessor besitzt einen eigenen Programmzähler
- auszuführende Prozesse stehen in einer gemeinsamen Warteschlange
- Kopplung des Systems über den gemeinsamen Speicher

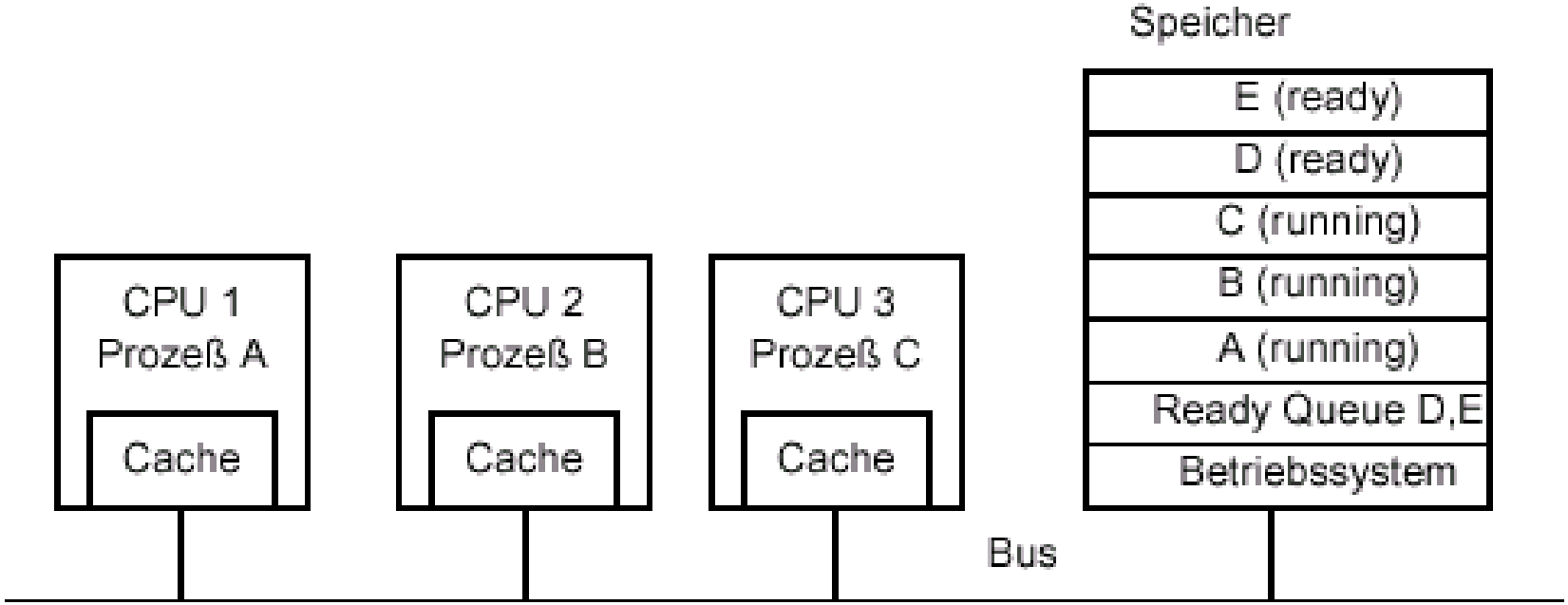

Bild E.8 Multiprozessor Timesharing System [Tan95]

Verteilte Betriebsumgebung über lokalem Betriebssystem

- Kommunikations- und Verteilungsschicht ist logisch über das lokale Betriebssystem aufgesetzt

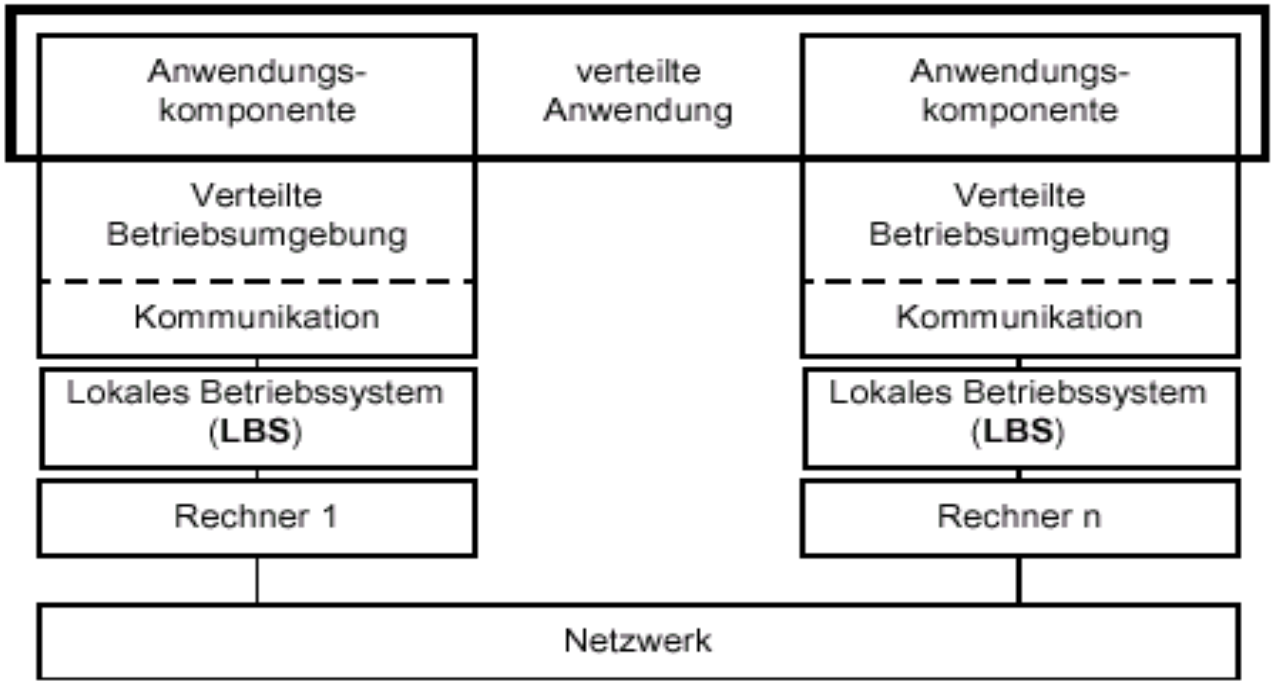

Bild E.9 Verteiltes System ohne Betriebssystemunterstützung

Beispiele:

- CORBA (OMG)
- DCOM/COM + Transaktion-Server (Microsoft)
- Agenten-Systeme
- JINI/EJB (JAVA/SUN)

Netzwerkbetriebssystem

- Die Funktionen zur Kommunikation sind im lokalen Betriebssystem integriert

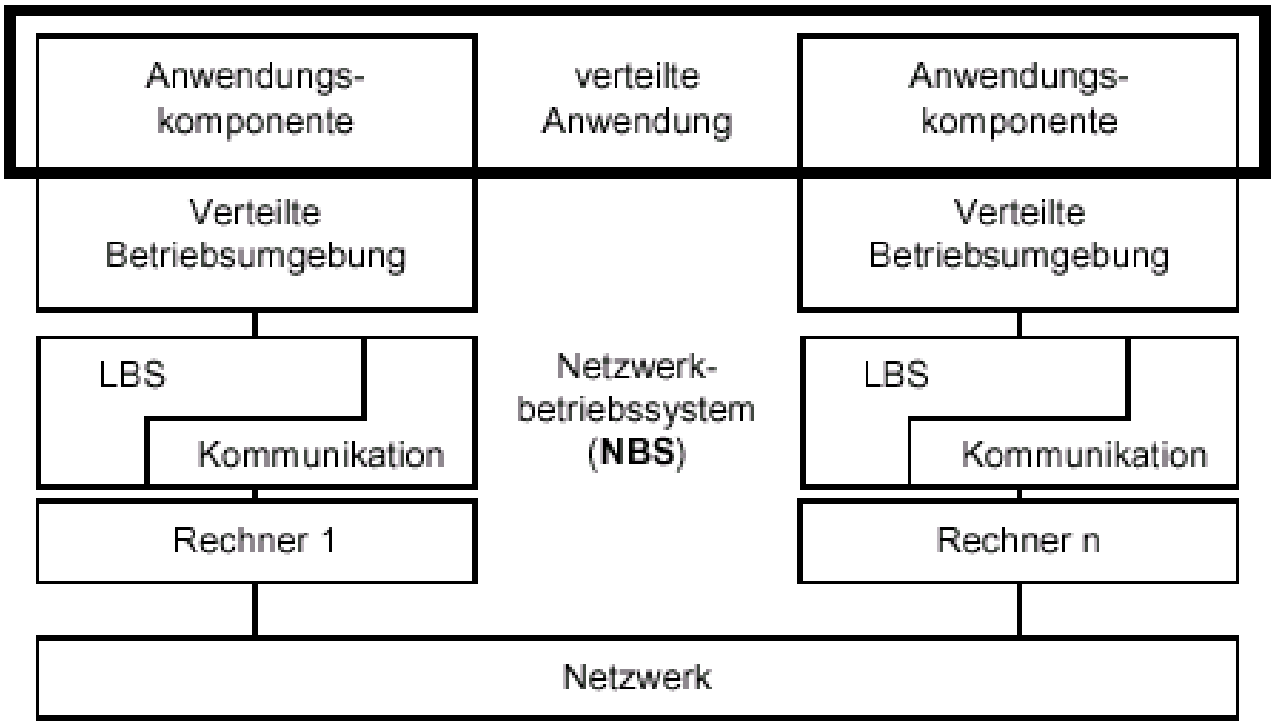

Bild E.10 Netzwerkbetriebssystem

Verteiltes Betriebssystem

- Im verteilten Betriebssystem sind sowohl die Dienste zur Kommunikation, sowie die Verteilungsmechanismen vollständig integriert
- Die Verteilung geschieht für den Benutzer möglichst transparent

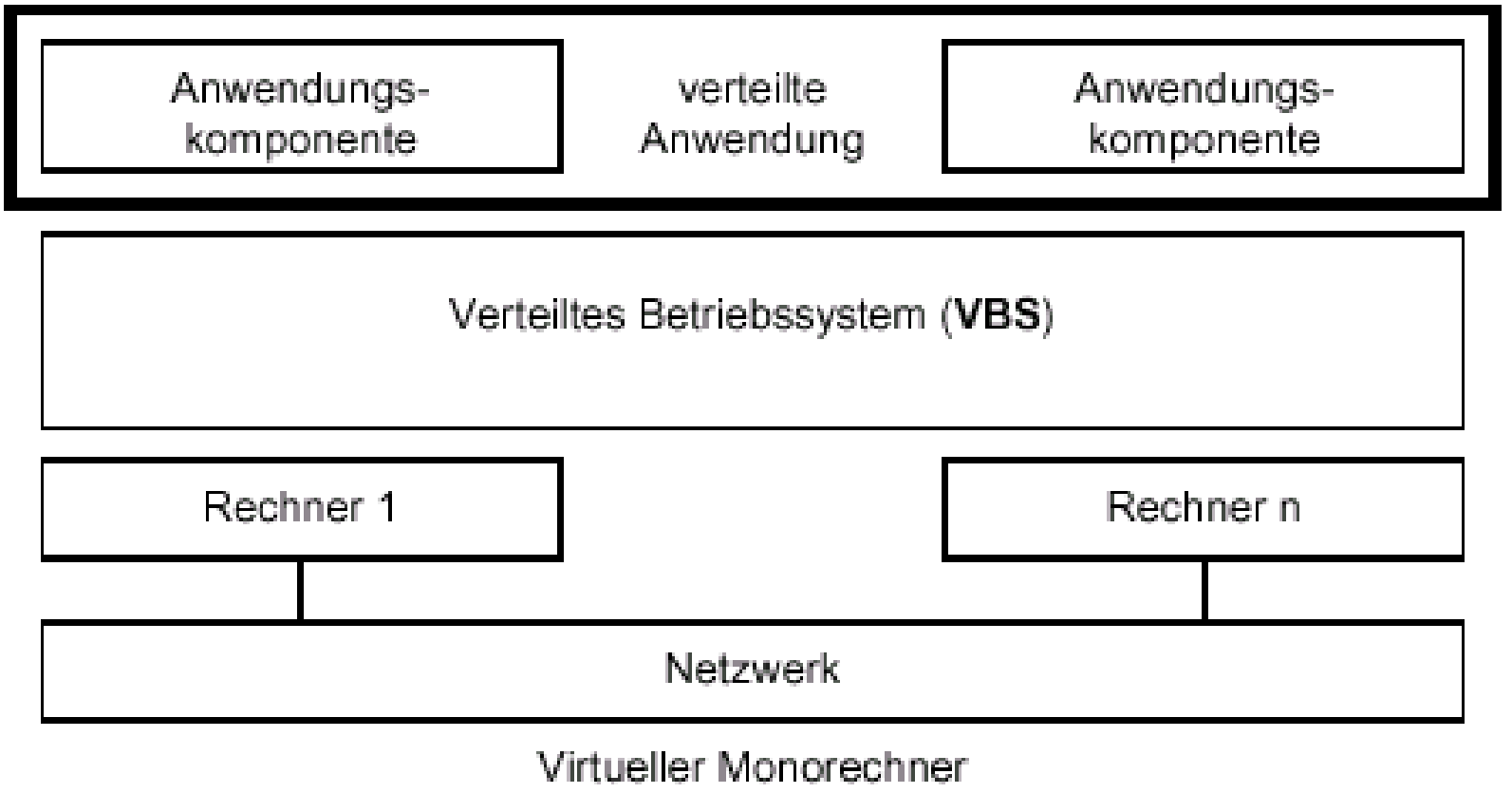

Bild E.11 Verteiltes Betriebssystem

- OSF (Mach) ➔ DCE

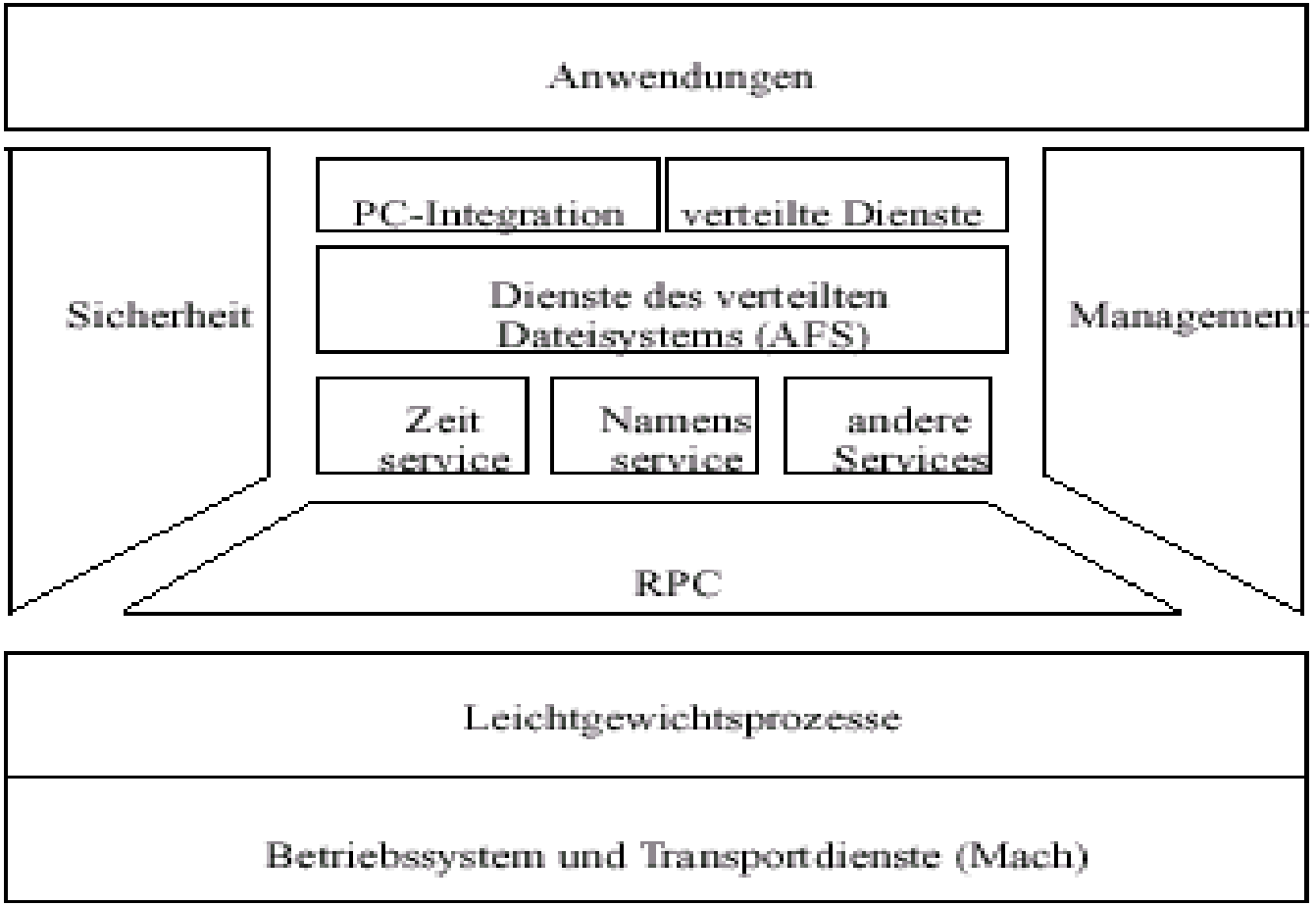

Bild E.11b DCE [Schlichter VA 99]

E.3.2 Programmiersprachliche Unterstützung

Die Verteilung kann auch durch die Programmiersprache unterstützt werden, indem entsprechende Strukturen und Konstrukte für verteilte Programmausführung bzw. für Kommunikation von der Sprache angeboten werden.

- *Concurrent Sequential Processes (CSP)* ➔ Nebenläufigkeit und Nachrichtenaustausch

- *Occam (obsolet)* ➔ speziell für Transputerprogrammierung ,Nebenläufigkeit und Nachrichtenaustausch

- *Ada (USA-Militär)* ➔ konkurrierende Tasks verschiedener Prioritäten, Entry als Nachrichten/Methodenempfang

- *Java, C#* ➔ Threads und Kommunikations-Packages (Socket, URL, RMI, ...)

- *Dialekte* ➔ Kommunikationsroutinen zu Programmiersprachen wie Pascal, C oder Fortran hinzugefügt.

- *spezielle objektorientierte Sprachen* ➔ Objekte sind nicht an einen Ausführungsort gebunden, sondern können verteilt werden. (z.B. Emerald, Agentensysteme)

- *Bibliotheken (ATL, MFC, TAO ....)* ➔ Bibliotheken werden zu „normalen“ Programmen hinzugebun den, ohne den Sprachumfang explizit zu erweitern.

E.4 Anforderungen und Problemstellungen verteilter Systeme

Wunsch: Gemeinsames effektives Nutzen von Ressourcen

- Hardwarekomponenten wie Drucker, Platten und Prozessoren
- gemeinsames Nutzen von Daten und Diensten

➔Problematik:
- Regelung nebenläufiger Zugriffe
- Fragen der Konsistenz und der Fehlertoleranz.

Verteilte Systeme zu entwerfen bringt eine Vielzahl von zu lösenden Einzelproblemen mit sich.
Vorgehensstrategie:

➔ Trennung der anwendungsspezifischen Probleme von den allgemeinen Problemen und

➔ Lösung der allgemeinen Probleme so, daß die entstehenden Mechanismen **transparent** zur eigentlichen Problemstellung sind

Transparenz

- *Zugriffs-Transparenz (access)*
  Lokale und entfernte Zugriffe sind identische Mechanismen.
  ➔*open ('drucker'...) egal wo sich der Drucker befindet*

- *Lokations-Transparenz (location)*
  Der Benutzer weiß nicht, wo sich physikalisch die Ressourcen befinden. Er spricht sie mittels eines unabhängigen Namens (z.B. Symbol) an.
  ➔*Gegenbeispiel: seck-server:\usr\test.txt*

Zugriffs- und Lokationstransparenz werden auch als Netzwerktransparenz bezeichnet.

- *Migrations-Transparenz (migration)*

  Ressourcen können im System physikalisch bewegt werden,ohne ihre Namen und Zugriffsmechanismen zu ändern.
  ➔ geht oft einher mit Orts- und Zugriffstransparenz
  ➔ Online/offline Migration

- *Fehler-Transparenz (failure)*
  Das Gesamtsystem wird nicht von einem Ausfall von Teilkomponenten beeinflußt. Die Daten bleiben konsistent.

- *Replikations-Transparenz (replication)*
  Ressourcen können für den Benutzer transparent repliziert werden.

  ➔ Fehlertoleranz bedeutet Redundanz in Hardware/ Softwareausführung/ Daten
  ➔ Konsistenz der redundanten Teile ohne! Anwenderwissen

- *Nebenläufigkeits-Transparenz (concurrency)*
  Ressourcen können konkurrierend von mehreren Nutzern zugegriffen werden.

- *Parallelitäts-Transparenz*
  Aktivitäten können parallel ablaufen, ohne sich zu stören.

- *Programmier-Transparenz (language)*
  Teilkomponenten einer Anwendung sind mittels unterschiedlicher Programmiersprachen realisiert ohne, daß darauf bei ihrem Erstellen Rücksicht genommen werden muß

- *Leistungs-Transparenz (performance)*
  dynamische Rekonfiguration zur Verbesserung der Performance (➔verteilte Datenbanken)

- *Prozeß-Transparenz (execution)*
  Ausführungsort eines Prozesses hat keinen Einfluß

- *Skalierungs-Transparenz (scalability)*
  Erweiterung des Systems ohne Systemstruktur oder Anwendungsalgorithmen neu definieren zu müssen

  Näher: Soll ein System beliebig erweiterbar sein, müssen einige wichtige Punkte beachtet werden.

  • Die Skalierung sollte für die Software transparent sein, damit nicht bei jeder Erweiterung Veränderungen an den Programmen notwendig werden.

  • Das System darf keine zentralen Komponenten benötigen, sonst bildet sich schnell ein Flaschenhals, da alle Rechner auf dieselbe Zentralkomponente zugreifen müssen.

  • Im System dürfen keine zentralen Daten und Tabellen gehalten werden.

  • Algorithmen dürfen nicht zentralisiert ausgelegt werden, d.h. die Ausführung sollte nicht an einen bestimmten Ort gebunden sein.

Offenheit

Es gibt nicht immer homogene Hardwarebedingungen und einheitliche Software, deshalb:

• Die Schlüsselschnittstellen der Hardware- und Softwarekomponenten müssen offengelegt sein.

• Die Interprozesskommunikation auf Anwendungsebene muß einheitlich und offengelegt sein.

➔ Firmenstandards sind i.d.R. kontra Offenheit

Flexibilität, Dynamik

Die Erweiterbarkeit um neue Entwicklungen, neue Dienste und neue Komponenten muß einfach durchzuführen sein.

Anti-Beispiel: ➔ Betriebssysteme, die all ihre Funktionalität in einem einzigen monolithischen Kern vereint haben.

positives Beispiel: ➔ Mikrokern-Betriebssystem stellt komplexe Dienste durch Server bereit.

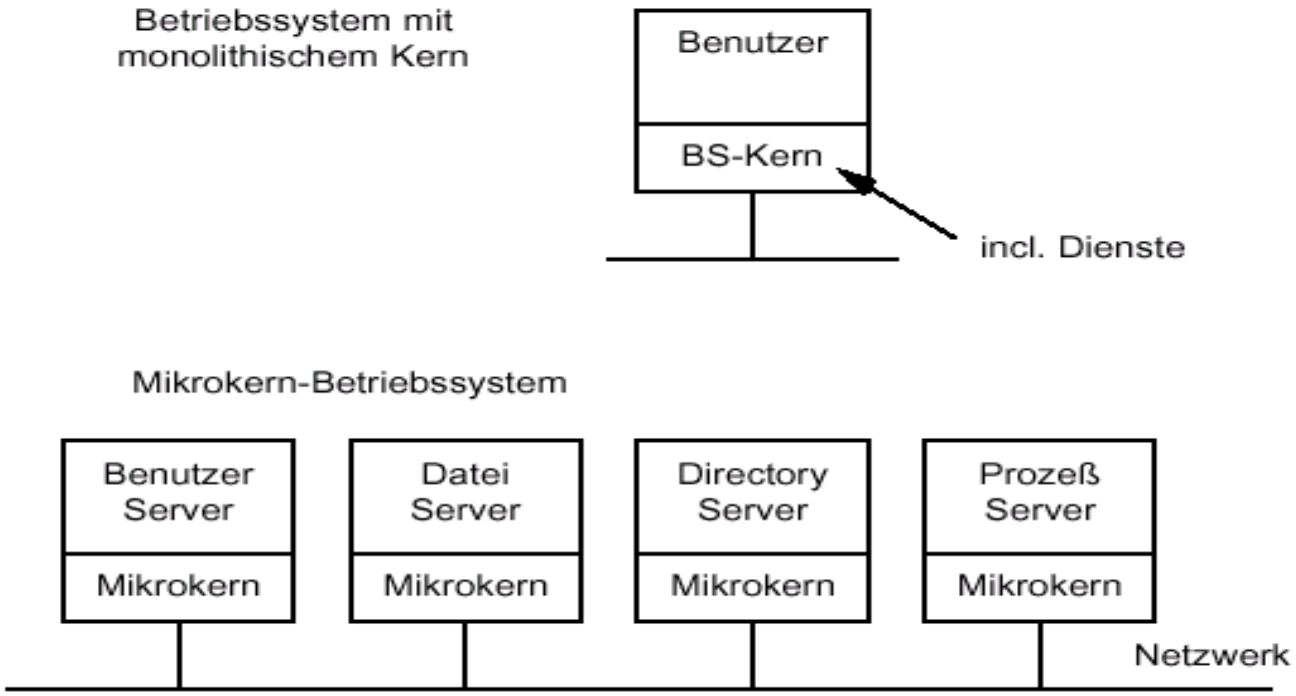

Bild E.12 Monolithisches Betriebssystem und Mikrokern-Ansatz

Zuverlässigkeit

- Verfügbarkeit
- Fehlertoleranz

Effizienz

- abhängig von der Granularität der nebenläufig oder parallel ausgeführten Programmportionen.
- Kommunikationsaufwand nimmt mit steigender Feinheit der Granularität zu.

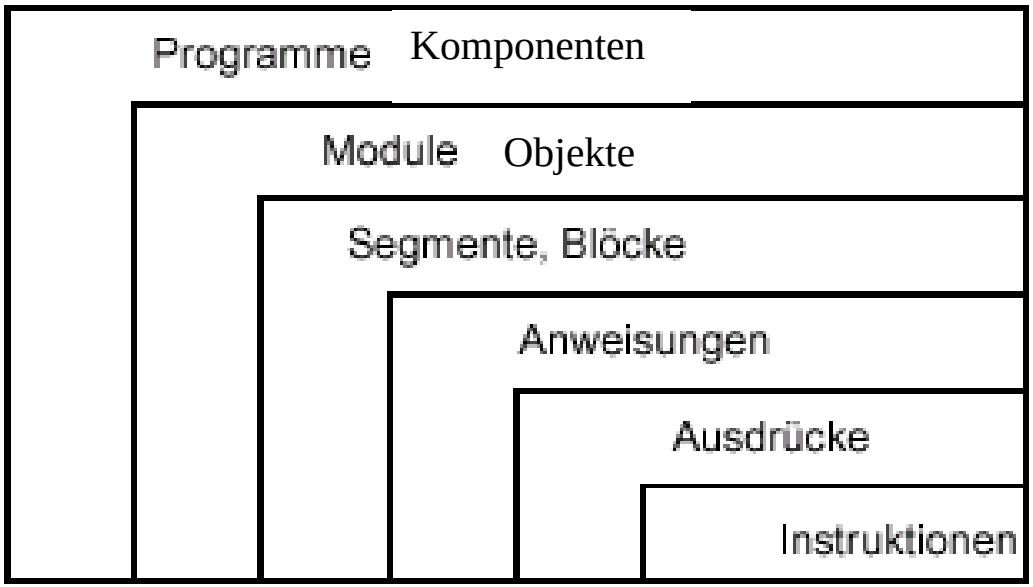

Bild E.13 Granularitäten in Programmen

E.5 Aufbau verteilter Softwaresysteme

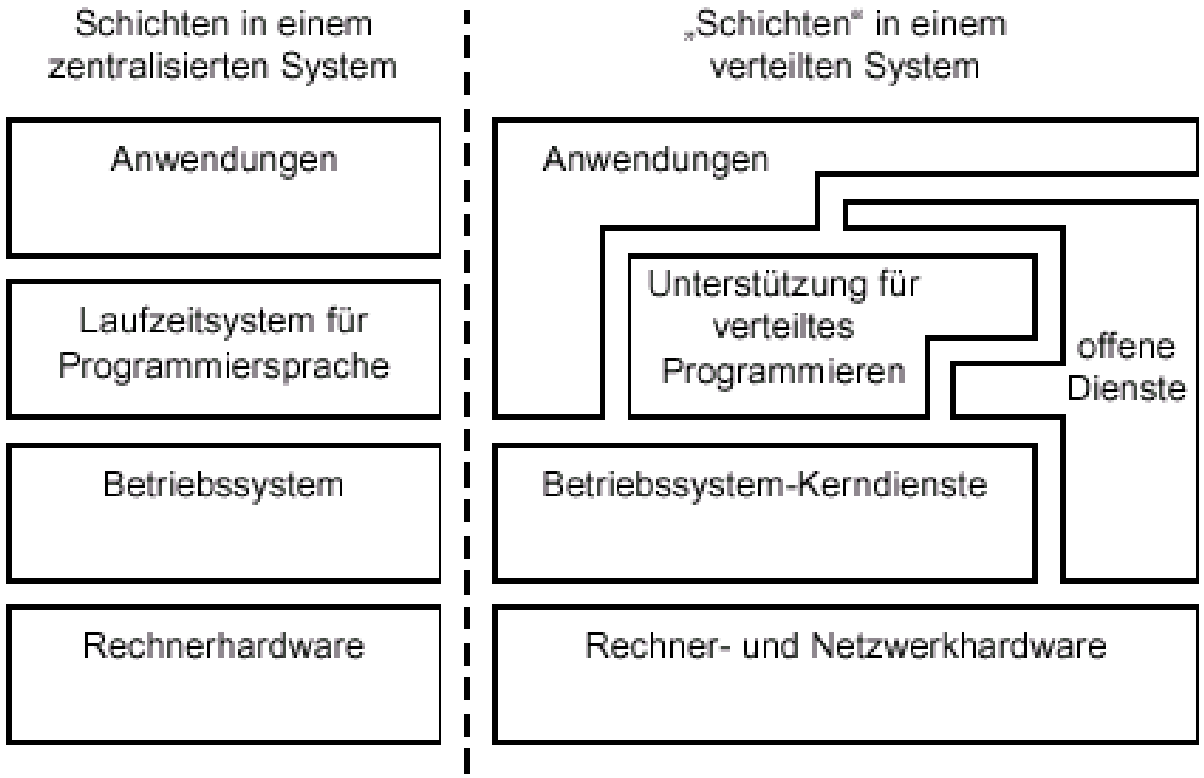

Bild E.14 Aufbau von zentralisierten und verteilten Systemen

Phänomene (Probleme) in verteilten Systemen

- Die Übertragung über ein Netzwerk ist unsicher.
- Nachrichten können verloren gehen oder Bits umkippen.
- unbestimmte Nachrichtenlaufzeiten durch unterschiedliches Routing oder veränderliche Übertragungsleistungen.
- später abgeschickte Nachrichten können vor früher abgeschickten beim Empfänger ankommen.
- keine gemeinsame Zeitbasis.
- der Zustand des Systems ist verteilt.
- Teilkomponenten aus dem Gesamtverbund können ausfallen.

Deshalb im Folgenden die theoretischen Grundlagen diese Probleme zu lösen.

Übungen Einführung

1. **Beispiele Verteilter Systeme**

   Nennen Sie ein paar Beispiele für verteilte Systeme, mit denen Sie regelmäßig arbeiten bzw. in Kontakt kommen. Handelt es sich dabei jeweils um eng- oder lose-gekoppelte Systeme ?

2. **Anforderungen an Verteilte Systeme**

a. Erläutern Sie die Begriffe Zugriffs-, Lokations-, Migrations-, Replikations-, Nebenläufigkeits-, Parallelitäts- und Fehlertransparenz anhand Ihnen bekannter verteilter Systeme (z.B. Internet-, Unix-, Windows-Dienste).

b. Ein Fileserver kann pro Sekunde 100 File-Zugriffe seiner Klienten bedienen. Der Server kann repliziert werden, benötigt dann jedoch zur Synchronisation eine Verbindung zu jedem anderen Server. Aufgrund des damit verbundenen Aufwands kann er **pro Verbindung zu einem Server** 10 Filezugriffe von Klienten pro Sekunde weniger bedienen. Wieviele Server müssen mindestens laufen, um 400 Klienten Zugriffe pro Sekunden abzuarbeiten ? Wie beurteilen Sie die Skalierbarkeit des Systems ?

➔ **Selbststudium anhand Buch:**

Michael Weber "Verteilte Systeme", ISBN 3-8274-0221-2