Einführung und Grundlagen

Verteilte Softwaresysteme

Prof. Dr. Oliver Braun

Letzte Änderung: 27.04.2020 17:48

Historische Entwicklung der Rechensysteme

- Batch Processing Systeme
- Timesharing Systeme
- Personal Computer und Workstation
- Client-Server-Systeme
- Verteilte Systeme
- Web-Systeme

begünstigt wurde diese Entwicklung durch:

1. Mächtige Mikroprozessoren
2. Lokale Netzwerke (LANs)
3. Verbindung von LANs $\Rightarrow$ Internet

Gekoppelte Prozessoren

- Zusammenschalten einer beliebigen Anzahl von Mikroprozessoren zu **parallelen** bis hin zu **massiv parallelen** Systemen.
- **eng gekoppelte Multiprozessorsysteme**: über den Bus mit einem gemeinsamen Hauptspeicher gekoppelt
  - Verwendung hauptsächlich in Servern
  - symmetrische Organisation des Betriebssystems
  - alle Prozessoren funktional identisch

Gekoppelte Rechner — Cluster

- Server: hohe Anforderungen an Ausfallsicherheit und Leistungsvermögen
- Idee: Zwei oder mehrere Rechner zu einem verbinden, die wie einzige **virtuelle Maschine** funktionieren
- **Cluster** ist solches System von Rechnern, die sich über spezielle Verbindungen über ihre Einsatzbereitschaft informieren
- fällt ein System aus, werden alle Prozesse an ein anderes übergeben

Peer-to-Peer

- Cluster sind geschlossene Systeme (Insellösungen)
- Idee: Rechner im Netz verbinden, Betriebsmittel gemeinsam nutzen
- **peer-to-peer Netz**
- für Vereinfachung der Organisation zentralisierte Dienste im Netz
    - Beispiel: nur Print-Server kennt alle Drucker, alle anderen Rechner müssen zum Drucken nur Print-Server kontaktieren

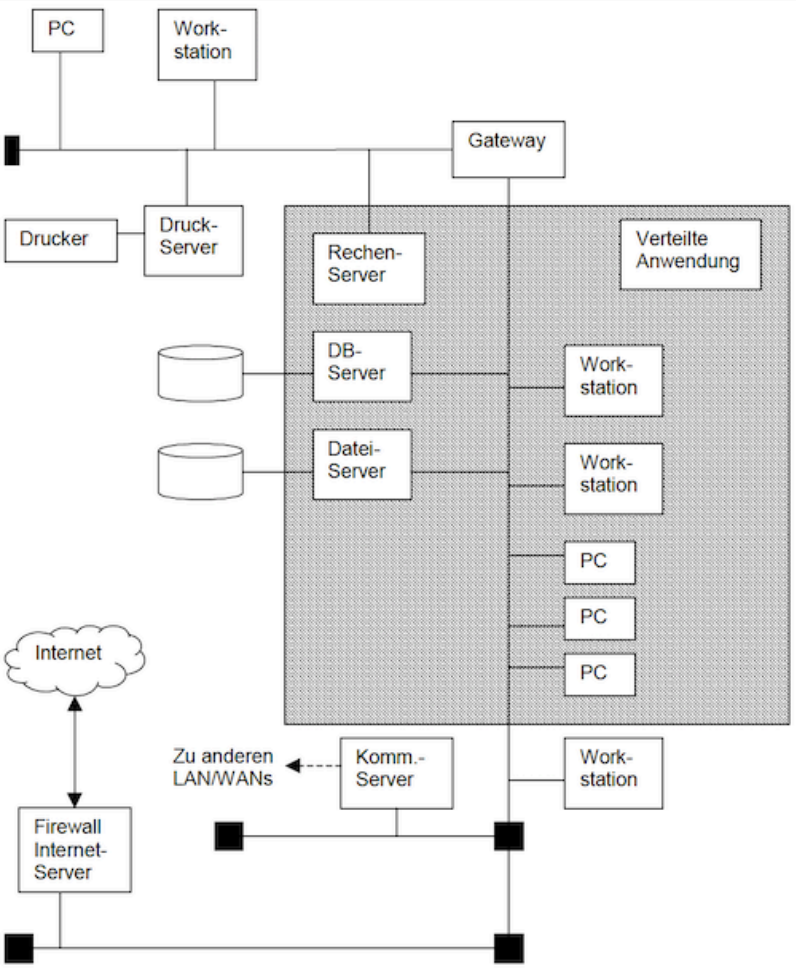
Betriebsysteme

- Netzwerkbetriebssystem
  - Aufsätze auf bestehenden Betriebsystemen
  - die den Zugriff auf entfernte Ressourcen ermöglichen
  - auf Servern den Server-Betrieb organisieren
- Verteilte Betriebssysteme
  - Betriebssystemfunktionalität verteilen und das gesamte Netz von Rechnern wie einen Rechner wirken zu lassen

Verteiltes System

- Anwendung selbst über das Netz verteilen
- kooperative Verarbeitung
- das logische verteilte Anwendungssystem sowie das Trägersystem (Hardware) bezeichnet man als **Verteiltes System**
- allgemein:
  - eine Reihe einzelner Funktionseinheiten
  - über Transportsystem verbunden
- Beispiel: World Wide Web

Verteiltes System – Beispiel



Netzwerkbetriebssysteme

Vorteile

1. Nutzung von gemeinsamen Betriebsmitteln
    - Nutzung von gemeinsamer Hardware
    - Zugriff zu gemeinsamen Daten und Programmen
2. Mailingdienste
    - Bürokommunikation, IP-Telefonie, Videoconferencing, ...
3. Anbindung an Großrechner und zentrale Server

Vergleich zu Großrechnern, Minicomputer, zentrale Server

- Vorteile
    1. Inkrementelle Erweiterbarkeit
    2. Erhöhte Fehlertoleranz
- Nachteile
    1. Jeder einzelne Server ist auch in seiner Leistung begrenzt
    2. Jeder einzelne Server kann **single point of failure** sein
    3. Pflege und Wartung der Clients aufwändig, dezentral

Clients mit einem Netzwerkbetriebssystem

- Kommunikations-Aufsätze auf bestehende Betriebsysteme
- Beispiele:
  - Remote Login — Einloggen auf Server
  - Remote Copy — Dateitransfer zwischen Client und Server
  - gemeinsames Filesystem
    - Fileserver bietet über spezielles Protokoll die Möglichkeit Verzeichnisse zu mounten
    - Beispiel für Protokolle: NFS, SMB
    - Client hat Software die die Dateien so in das Dateisystem integriert, als ob die Dateien lokal vorliegen

Verteilte Betriebssysteme

Vorteile

- Netzwerkaspekte sind vor Benutzer verborgen
- Vorteile:
    1. Einfache Benutzerbedienung: Der Benutzer sieht nur eine (virtuelle) Maschine
    2. Erhöhte Benutzermobilität: An jedem Rechner des Systems findet der Benutzer die gleiche Benutzungsumgebung vor
- Ziel eines verteilten Betriebssystems ist Herstellung von verschiedenen Transparenzeigenschaften

Dezentralisierung

- Benutzer weiß weder wo seine Dateien liegen noch wo ein Prozess ausgeführt wird
- alle Entscheidungen werden vom System getroffen
- zentrale Kontrollschemas und zentrale Ressourcen widersprechen dem Ansatz
- im Idealfall: **funktionale Symmetrie**
    - alle Maschinen spielen die gleiche Rolle und besitzen einen gewissen Grad von Autonomie

Koordination

- einzigen globalen Mechanismus zur Interprozesskommunikation
- einfacher wenn auf jedem Rechner ein identischer Betriebssystemkern läuft
- sinnvoll: Erweiterung des lokalen Prozessmanagements durch Lastinformations- und Lastverteilungssystem

Beispiele für verteilte Betriebssysteme

Plan 9 (1/2)

- Bell Labs, später Lucent Technologies
- erstes Release 1992 (nur für Universitäten)
- 1995 öffentlich, ab 2000 unter Open Source Lizenz
- Beteiligte:
  - Rob Pike (Golang)
  - Ken Thompson (Golang)
  - Dennis Ritchie (C & UNIX)
  - Brian Kernighan (C & UNIX)
  - Bjarne Stroustrup (C++)
  - Douglas McIlroy (UNIX Pipelines & Tools)
  - …

benannt nach dem schlechtesten Science Fiction Film aller Zeiten: *Plan 9 from Outer Space* (1959)

Plan 9 (2/2)

- typische Plan9-Installation:
  - Benutzer sitzen an Terminals auf denen das Windowsystem *rio* läuft
  - einige Server stellen leistungsfähige CPUs zur Verfügung
  - zusätzliche Server stellen permanenten Datenspeicher zur Verfügung
- Designgrundlagen
  1. jeder Prozess hat einen eigenen Namespace
  2. einfaches nachrichtenorientiertes Filesystem-Protokoll
- UNIX-Philosopie: “Alles ist eine Datei”
- Prozess kann einem anderen eine virtuelle Datei als Dienst zur Verfügung stellen
- Plan 9 Demo auf Vimeo

Inferno

- *A compact operating system for building cross-platform distributed systems*
- einer der Plan9-Nachfolger
- kann als Anwendung auf Windows, Mac OS X, Linux, ... laufen
- oder als OS direkt auf ARM, Intel x86, PowerPC & SPARC
- Anwendung darauf
  - entwickeln in eigener Sprache *Limbo*
  - läuft in Dis Virtual Machine

Harvey OS

- *an effort to provide a modern, distributed, 64 bit operating system*
- 03.06.2018: Still here and working
- sogar als Docker-Container verfügbar
- Sourcecode unter https://github.com/Harvey-OS/harvey

Jehanne

- *Jehanne is a new distributed operating system designed for programmers.*
- Schwerpunkte:
  - Simplicity
  - Security
- auch noch aktuell in Arbeit:
  https://github.com/JehanneOS/jehanne/

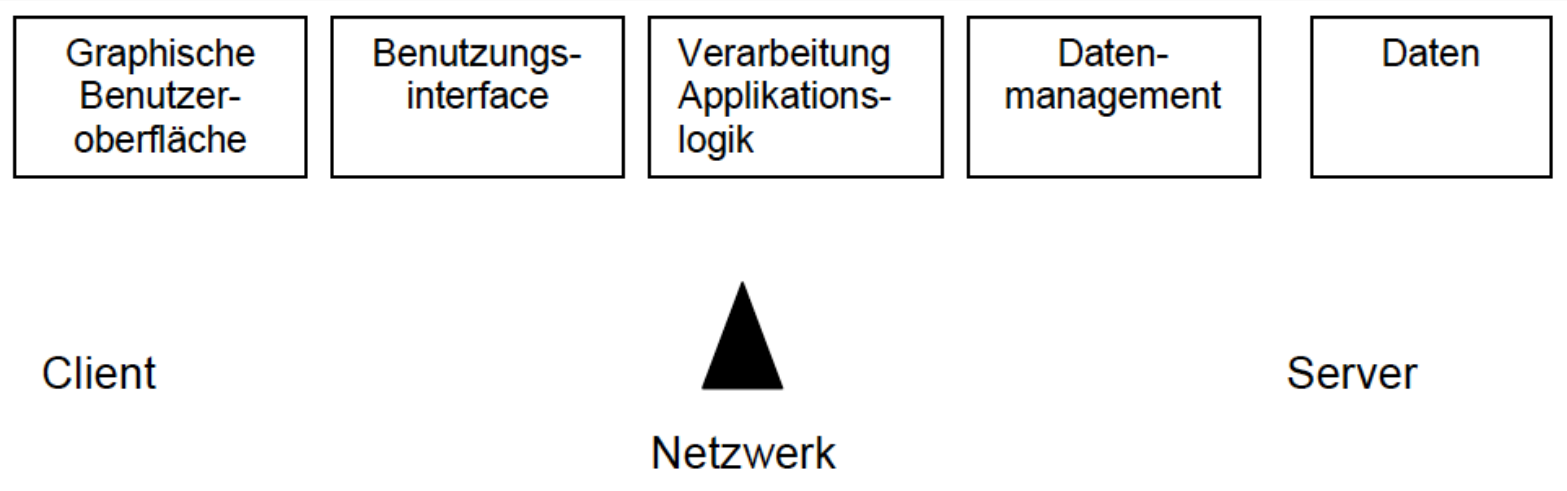
Nix

- *Get rid of the operating system!*
  - *High performance cloud computing is nix*
- Prozessorkerne können Applikationen exklusiv zugeordnet werden
  - ohne Interferenzen mit dem OS, nicht mal Clock Interrupts
- für HPC und Cloud
- relativ aktuelle Paper von 2011 und 2012

Verteilte Anwendungen

Schichtenmodelle

- Verteilte Anwendung ist
  - Menge von Funktionseinheiten oder Komponenten
  - die in Beziehung zueinander stehen (Client-Server)
  - und eine gemeinsame Funktion erbringen
- Komponenten:



Quelle: G.Bengels, Grundkurs Verteilte Systeme, Springer Vieweg

Aufteilung der Komponenten (1/3)

- bei einem Client und einem Server ergeben sich zwei Verteilungsstufen (*two tier model*):
- **Ultra Thin Client** oder **Null Client**
  - nur GUI auf dem Client, Client übergibt nur Tastaturanschläge und Mausbewegungen an den Server
  - ab Benutzungsinterface auf dem Server
  - wenn alles auf einer (Server-)Maschine: **host-based computing**
  - bekanntestes Beispiel: X-Windows

Aufteilung der Komponenten (2/3)

- **Thin Client**
  - Trennung zwischen Benutzerinterface und Anwendung
  - Client verwaltet GUI und einige Zustandsinformationen
  - Modell heißt auch **remote presentation**
  - PC kann abgemagert werden zu Network PC (NetPC)
- **Applet Client** oder **Web Client**
  - Trennung nach dem Benutzerinterface und schließt Teile der Applikationslogik mit ein
  - Applikationslogik läuft in Form von Applet oder JavaScript
  - Speicherung von Daten auf Client in Form von Cookies

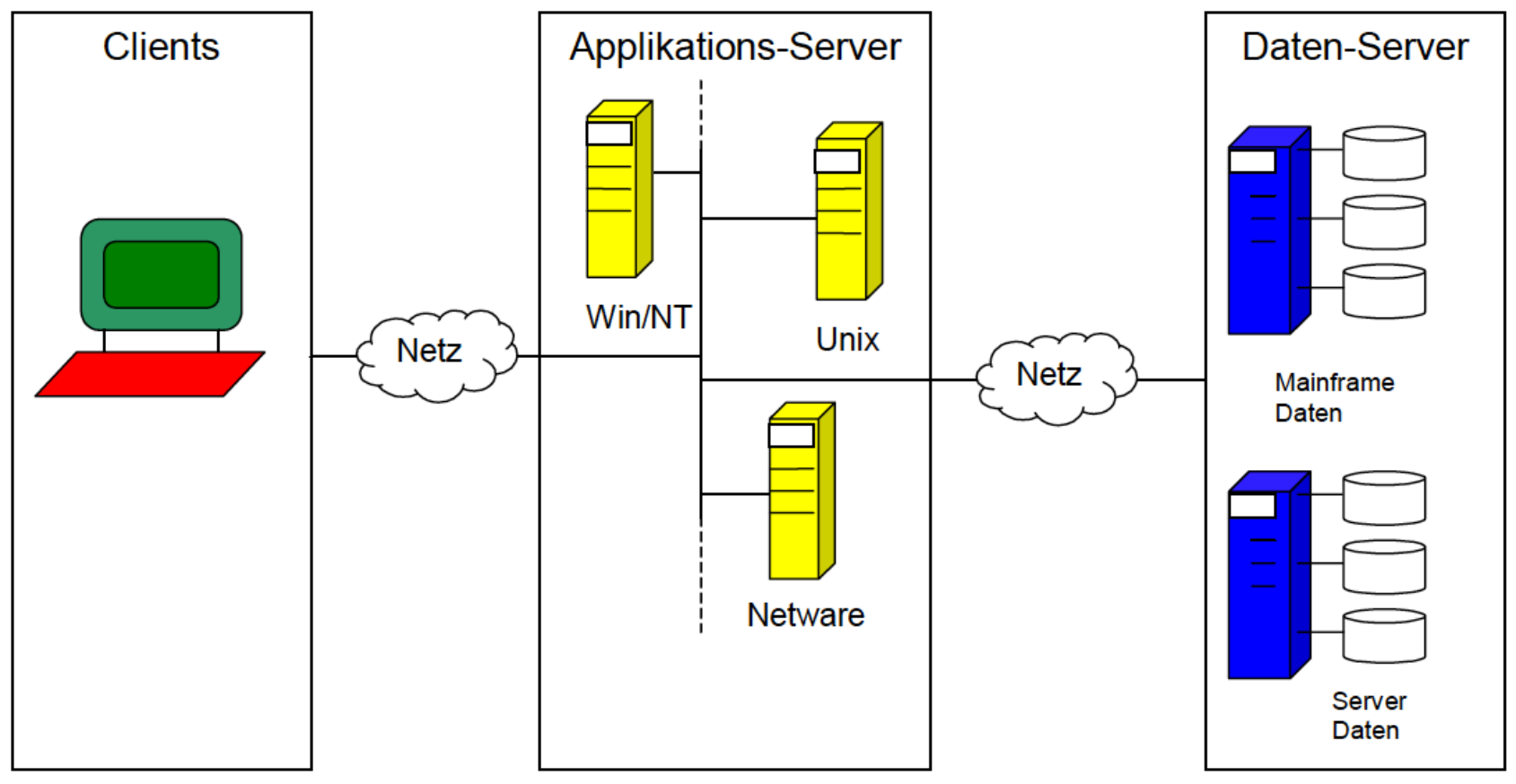
Aufteilung der Komponenten (3/3)

- Fat Client
  - Trennung in der Applikationslogik $\Rightarrow$ **cooperative processing**
  - Trennung zwischen Applikationslogik und Datenmanagement $\Rightarrow$ Server ist File- oder DB-Server

Mehrschichtige Anwendungen

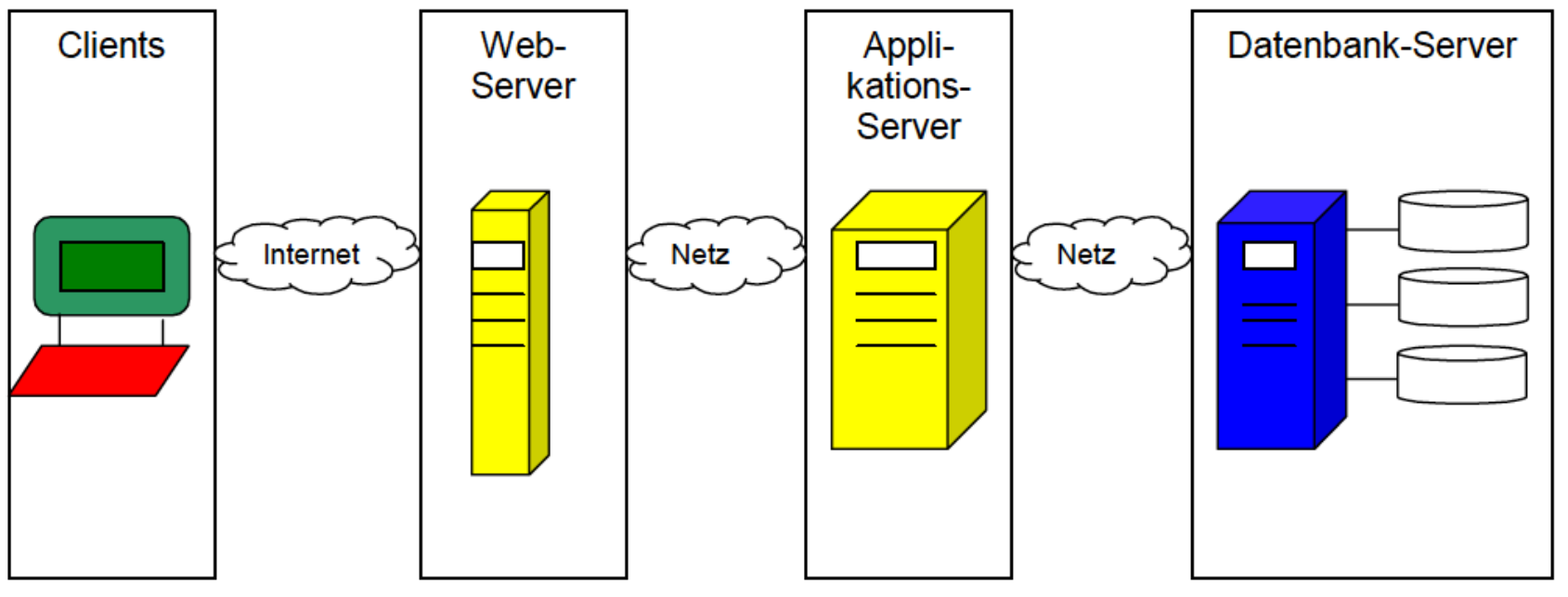
- heute immer mehr mehrschichtige Anwendungen (**multi tier model**)
- bessere Skalierbarkeit bei den Servern
- Einschluss der web-basierten Technologien

Three Tier Model



Quelle: G.Bengels, Grundkurs Verteilte Systeme, Springer Vieweg

Vierschichtige Web-Anwendung

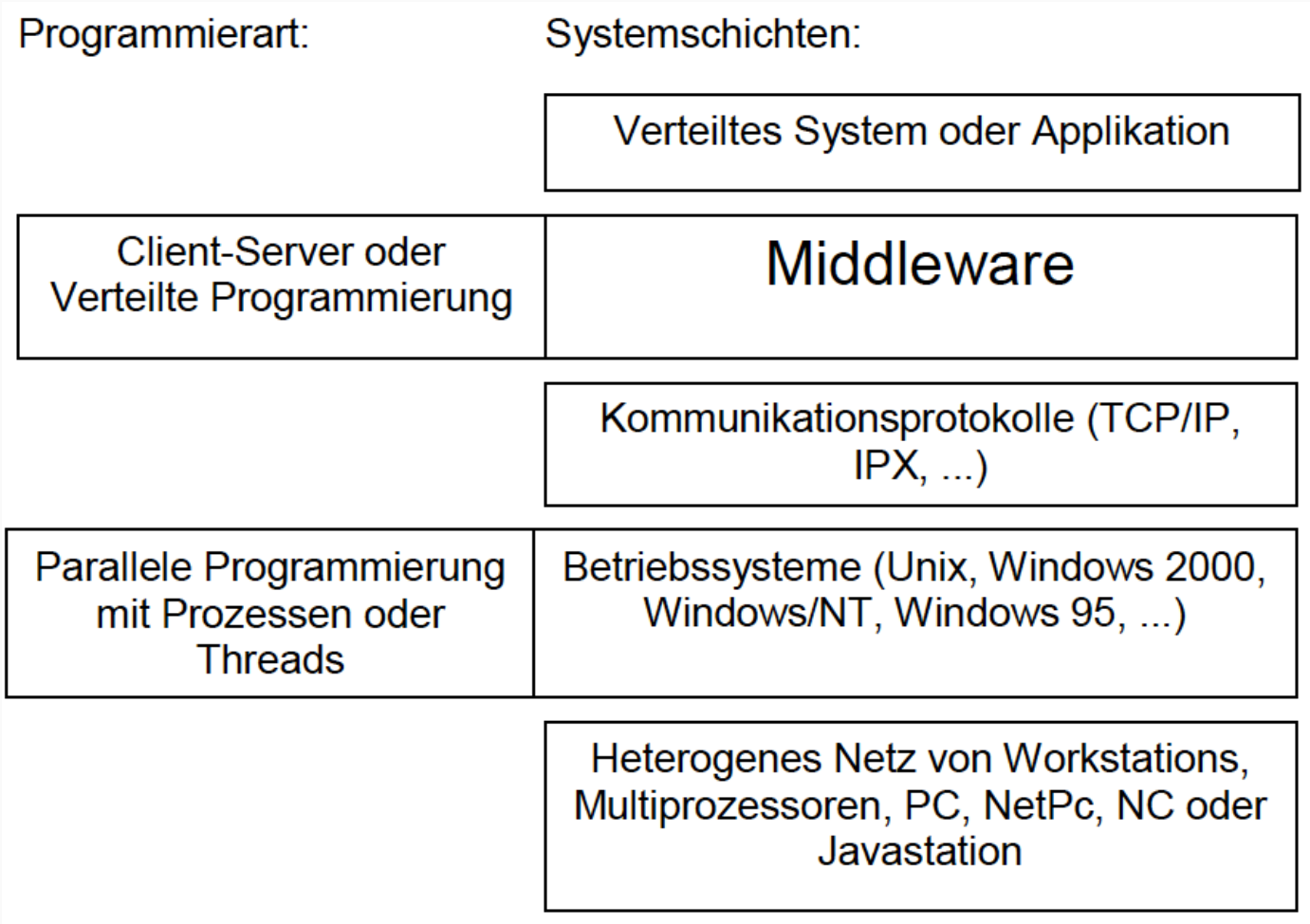


Quelle: G.Bengels, Grundkurs Verteilte Systeme, Springer Vieweg

Middleware (1/2)

- Verteilung der Softwarekomponenten auf heterogene Hardwarekomponenten
- zur Lösung von Heterogenitäts- und Verteilungsproblemen bei verteilten Anwendungen:
    - **verteilte Systemservices** mit Standardschnittstelle und -protokoll
- diese Services heißen **Middleware Services** oder kurz **Middleware**
- liegen zwischen Betriebssystem mit Netzwerksoftware und der spezifischen Anwendung

Middleware (2/2)



Quelle: G.Bengels, Grundkurs Verteilte Systeme, Springer Vieweg

Middleware-Service

- ein Middleware-Service ist verteilt
  - erlaubt entweder entfernten Zugriff (z.B. Datenbank- oder Präsentationsservice)
  - oder befähigt andere Services und Anwendungen zu entferntem Service (z.B. Kommunikationsservice)
- besteht üblicherweise aus
  - Client, welcher API des Services unterstützt und im Adressraum der Anwendung läuft
  - Server, der Dienst liefert und in anderem Adressraum läuft (z.B. auf anderem System)

World Wide Web

Web 2.0

- Tim O'Reilly (2005)
  1. Collaboration-, Participation- , Social- oder Read/Write-Web
     - Instant Messaging
     - Blogs
     - Wikipedia
     - YouTube
     - Facebook, ...
  2. Web-Services / Dienstleistungsservices
     - ebay
     - Google Maps, ...
  3. Service-Orientierte Architekturen
     - "Programmieren mit dem Web"
     - Web-Services
  4. Internet of Things
     - Smart Grid/Home/Factory/Car/Glasses/... Smarter Planet

Web 3.0 — Semantic Web

- maschinenlesbare und bearbeitbare Daten
- Software-Agenten (*virtuelle Handlungsreisende*)
  - übernehmen Such-, Kommunikations- und Entscheidungsaufgaben
  - Benutzerschnittstelle zum Semantic Web
- Wissensbereich wird mit Hilfe standardisierter Terminologie beschrieben: **Ontologie**
- gemeinsames Vokabular: **Taxonomie**
  - Klassen, Relationen, Funktionen, Axiome
- vergleichbar mit UML-Klassendiagramm
  - statt Klassen werden **Begriffe** modelliert

Web 4.0?

- **Web Intelligence (WI)**
  - untersucht Auswirkungen und Effekte von **KI** und fortgeschrittenen Informationstechnologien auf das Web
- betroffene Gebiete der Künstlichen Intelligenz:
  - Knowledge Representation, Knowledge Planning, Knowledge Discovery
  - Data Mining
  - Intelligent Agents
  - Social Network Intelligence
- fortgeschrittene Informationstechnologien umfassen z.B.
  - Wireless Networks
  - Ubiquitous Devices
  - Social Networks
  - Data/Knowledge Grids
- **World Wide Wisdom Web**, kurz **W4**, vielleicht zukünftig **Web 4.0**

E-World

- mobile und allgegenwärtige Endgeräte, führte zur Neugestaltung von Applikationen auf allen Lebensgebieten
  - hin zu **E-Applikationen** (**e**lectronic bzw. **e**nhanced)
  - zur zukünftigen **E-World**
- Web 1.0
  - Information at your fingertips
- Web 2.0
  - komplexe Services durch Machine2Machine-Kommunikation
- Web 3.0
  - Services mit sematischer Information für bessere Auswahl
- Web 4.0
  - wissensbasierte Systeme $\Rightarrow$ intelligente E-Applikationen

Selbstorganisierende Systeme

Begriffe

- verteilte Rechensysteme müssen installiert, konfiguriert, überwacht Sicherheitsanforderungen realisiert, umkonfiguriert und bei auftretenden Fehlern repariert werden
- hohe Kosten, die sog. **Total Cost of Ownership (TCO)**
- zur Senkung dieser Kosten gibt es verschiedene Wege
  - **Outsourcing** der Rechenressourcen
  - **Virtualisierung**, besonders zur Server-Konsolidierung
  - **selbstorganisierende Systeme** (selbstlaufend, selbstkonfigurierend, fehlertolerant)

On Demand Computing (ODC) (1/2)

- **Business on Demand** (IBM, 2002)
- ODC: Technologien und Ressourcen werden dem Benutzer zur Verfügung gestellt, wenn es sie benötigt
  - CPU-Zyklen
  - Bandbreite
  - Speicher
  - auch Anwendungen und Dienste
- Rechnerpool wird vom Anbieter durch Virtualisierungstechniken in logische Ressourcen aufgeteilt
- einer Task oder einer Applikation wird keine bestimmte Ressource zugeordnet, sondern eine beliebige, zur Laufzeit freie Ressource

On Demand Computing (ODC) (2/2)

- Leistungen werden vom Serviceprovider bei **pay-per-use** abgerechnet
- Serviceprovider bieten Katalog von standardisierten Diensten an
  - u.U. mit verschiedenen **Service Level Agreements**
- der Kunde hat keinen Einfluss auf zugrunde liegende Technologien
- Technologien und Geschäftsmodelle, mit denen Serviceprovider IT-Leistungen als Service erbringt, nennt man **Utility Computing**
- Unterteilung
  - **Internal Utility**: unternehmensinterner Dienstleister
  - **External Utility**: verschiedene Unternehmen nutzen Rechnerpool
  - **Mischformen**
- Sun Microsystemes hatten ein External Utility: **Sun Grid**
  - On-demand Grid Computing Service
  - Rechenleistung und -ressourcen über das Internet

Autonomic Computing

- **Autonomic Computing** (IBM, 2001)
- Ziel ist selbstorganisierendes autonomes Computer-System
  1. **Selbst-Konfiguration** (*Self-Configuration*)
     - automatische Konfiguration und Management der Komponenten
  2. **Selbst-Heilend** (*Self-Healing*)
     - automatische Entdeckung von Fehlern und deren Korrektur
  3. **Selbst-Optimierend** (*Self-Optimization*)
     - automatische Überwachung und Kontrolle der Ressourcen zur Sicherstellung der optimalen Funktionstüchtigkeit
  4. **Selbst-Schützend** (*Self-Protection*)
     - Identifikation und Schutz vor willkürlichen Angriffen
- der Mensch hat neue Rolle
  - statt Kontrolle
  - muss er Strategien und Regeln definieren

Organic Computing

- Netzwerk von autonomeren intelligenten Systemen auf Basis des Autonomen Computing
- Systeme alle autonom, voneinander unabhängig
- aber müssen flexibel aufeinander reagieren
- d.h. lebensähnliche Eigenschaften, daher **organisch**
- um Grids oder Cluster mit Tausenden Prozessoren zu beherrschen, sind die Selbst-Eigenschaften notwendig

Parallele vs. Verteilte Verarbeitung

Parallele Verarbeitung

- vorrangiges Ziel: **Geschwindigkeitssteigerung**
- Programm wird in Einheiten zerlegt die parallel ausgeführt werden:
  - sog. **Tasks** oder (aus OS-Sicht) **Prozesse**
- zur Reduktion der Zeiten für den Prozesswechsel und zur effizienten Server-Implementierung: **Threads**
- wird ein paralleles Programm auf nur einem Prozessor ausgeführt, heisst es **quasiparallel**
- Verteilung der Prozesse oder Threads durch **Scheduler**

Nebenläufige Prozesse

- zwei oder mehr **nebenläufige** (*concurrent*) Prozesse oder Handlungsstränge laufen gleichzeitig ab, haben aber nicht notwendigerweise etwas miteinander zu tun
- bei der Ausführung **konkurrieren** sie um die Ressourcen

Kooperierende Prozesse

- **Kooperierende Prozesse**, wenn
  1. sie eine gemeinsame Aufgabe erfüllen oder Teile eines übergeordneten Programms sind und
  2. sie logisch so miteinander verknüpft sind, dass sie synchronisiert werden müssen
- Beispiel: Erzeuger-Verbraucher-Problem
- **kommunizierende Prozesse**, wenn Prozesse nur über Nachrichtenaustausch kooperieren können
- Beispiel: *Communicating Sequential Processes (CSP)* von Tony Hoare
  - umgesetzt in Go (Goroutines und Channels)

Verteilte Verarbeitung

- **Distributed Computing**, Verteiltes Rechnen, Verteilte Verarbeitung
  - Koordination von vielen Rechnern
  - an (möglicherweise) weit entfernten Standorten
  - zur Erledigung einer gemeinsamen Aufgabe
- Rechner müssen eine “gemeinsame Sprache sprechen”
  - heute üblicherweise TCP/IP
- Rechner müssen die Nachrichten verstehen bzw. ungültige Nachrichten abweisen/ignorieren
- auch Senden von Software oder Code zu einem anderen Rechner muss grundsätzlich gegeben sein

Positive Eigenschaften der verteilten Verarbeitung

- neben der Geschwindigkeitssteigerung durch Parallelisierung...
- **Ausfalltoleranz**: Ausfälle sind partiell und legen nicht das gesamte System lahm
- **Fehlertoleranz**: z.B. Verwerfen einer korrumpierten Nachricht (und Informieren des Clients)
- **Erhöhte Verfügbarkeit durch Redundanzen**

Eigenschaften eines Verteilten Systems

Wesentliche Eigenschaften

- Skalierbarkeiten
  - Lastskalierbarkeit
  - geographische Skalierbarkeit
  - administrative Skalierbarkeit
- Offenheit
  - jedes System ist immer offen zur Interaktion mit anderen Systemen
- Transparenzen
  - ...

Transparenzeigenschaften, welche die Verteilung verbergen

- Ortstransparenz
  - der Ort einer Ressource ist dem Benutzer nicht bekannt
- Zugriffstransparenz
  - auf alle Ressourcen wird in der selben Weise zugegriffen, egal ob sie lokal oder entfernt vorhanden sind
- Nebenläufigkeitstransparenz
  - mehrere Benutzer können das System zeitgleich nutzen, aber jeder Benutzer weiß nur von seiner eigenen Benutzung
- Skalierungstransparenz
  - System kann flexibel erweitert werden

Transparenzeigenschaften, welche die Verteilung zur Leistungssteigerung und Fehlertoleranz ausnutzen (1/3)

- Migrationstransparenz
  - Prozess oder Datei kann aus Gründen der Performanz, Zuverlässigkeit oder Sicherheit auf anderen Rechner verschoben werden
  - dabei gilt
    1. Entscheidungen, welcher Prozess oder welche Datei wohin verlagert wird, trifft das System automatisch
    2. Ortstransparenz muss eingehalten werden (Name bleibt gleich)
    3. Wird an Prozess Nachricht geschickt oder auf Datei zugegriffen, während Verschiebung, muss das System den Erfolg selbst sicherstellen

Transparenzeigenschaften, welche die Verteilung zur Leistungssteigerung und Fehlertoleranz ausnutzen (2/3)

- Leistungstransparenz
  - das gesamte Netz von Rechnern stellt Rechenleistung zur Verfügung
  - welcher Rechner die Leistung erbringt, muss unsichtbar sein
  - Aufgaben und Last werden automatisch verteilt

Transparenzeigenschaften, welche die Verteilung zur Leistungssteigerung und Fehlertoleranz ausnutzen (3/3)

- Replikationstransparenz
  - liegen mehrere Kopien einer Ressource vor, ist dies für den Benutzer unsichtbar
  - System sorgt für Konsistenz
- Fehler- und Ausfalltransparenz
  - Benutzer soll nicht merken, wenn Fehler oder Ausfälle auftreten

Probleme durch Offenheit des Verteilten Systems

- Verbreitungs-Monotonie
  - ist eine Information oder Nachricht verbreitet, kann sie nicht mehr zurück genommen werden
- Pluralismus
  - verschiedene Subsysteme können heterogene, überlappende und sogar in Konflikt stehende Informationen besitzen
  - es gibt keine zentrale Instanz der Wahrheit
- Unbegrenzter Nichtdeterminismus
  - Subsysteme können beliebig kommen und gehen
  - genauso Kommunikationskanäle oder Verbindungen
  - daher ist nicht absehbar, wann eine Operation abgeschlossen ist