Betriebssysteme Vorlesung 10

Verteilte Systeme

Client/Server Architekturen

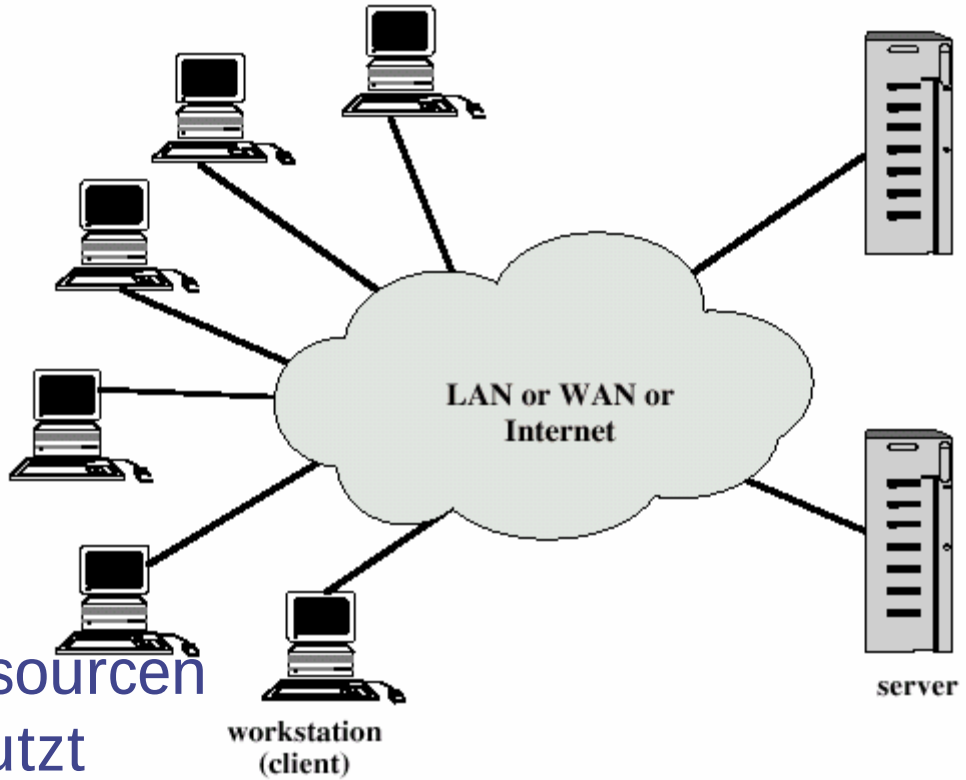
Literatur: Stallings Kapitel 13

Übersicht

- Client / Server Computing
  - Varianten, Middleware, Cache Konsistenzproblem
- Verteiltes Message Passing
  - Zuverlässigkeit, Blocking vs Non-Blocking
- Remote Procedure Calls
  - Funktionsweise, Parameterübergabe, Binding,
  - Synchrone vs Asynchrone RPCs
  - Objekt-orientierte Mechanismen
- Cluster Computing
  - Skalierbarkeit, Verfügbarkeit, Preis/Leistungsverhältnis
  - Konfigurationen, Failure Management, Last Balancierung
  - Architektur
- Beispiele für Cluster Architekturen
  - Windows 2000, Sun, Beowulf und Linux Cluster

Potentielle Vorteile durch dezentrale Strukturen

- Responsiveness
  Lokalität verkürzt Antwortzeiten, kleinere Aufgaben je Komponente
- Verfügbarkeit
  Ausfall von Komponenten weniger kritisch
- Ressourcen teilbar
  teure / leistungsstarke Ressourcen werden von mehreren genutzt
- inkrementelles Wachstum
  Größe der Architektur kann an Bedarf angepasst werden
- Eingriffsmöglichkeiten des Anwenders
  Anwender hat Gestaltungsmöglichkeiten
- Produktivitätssteigerung für den Endanwender
  durch bessere Berücksichtigung des lokalen Bedarfs



Grundbegriffe

- Kommunikationsarchitektur
  - Software für Kommunikation zwischen unabhängigen Rechnern
  - Beispiel: TCP/IP, (Transmission Control Protocol, Internet Protocol)
- Netzwerk Betriebssystem
  - Client / Server Architekturen
  - Rechner mit eigenem Betriebssystem, von denen einige als Server zentrale Dienste, z.B. File Management, Printer Management, für andere zur Verfügung stellen
  - Anwender erkennen Rechner aus dem Netzwerk, können diese „remote“ nutzen
- Verteiltes Betriebssystem
  - Betriebssystem als virtuelle Maschine lässt das Netzwerk als einen Rechner mit uniformer Anwendungsschnittstelle erscheinen
  - alle Rechner des Netzes werden durch ein gemeinsames Betriebssystem verwaltet

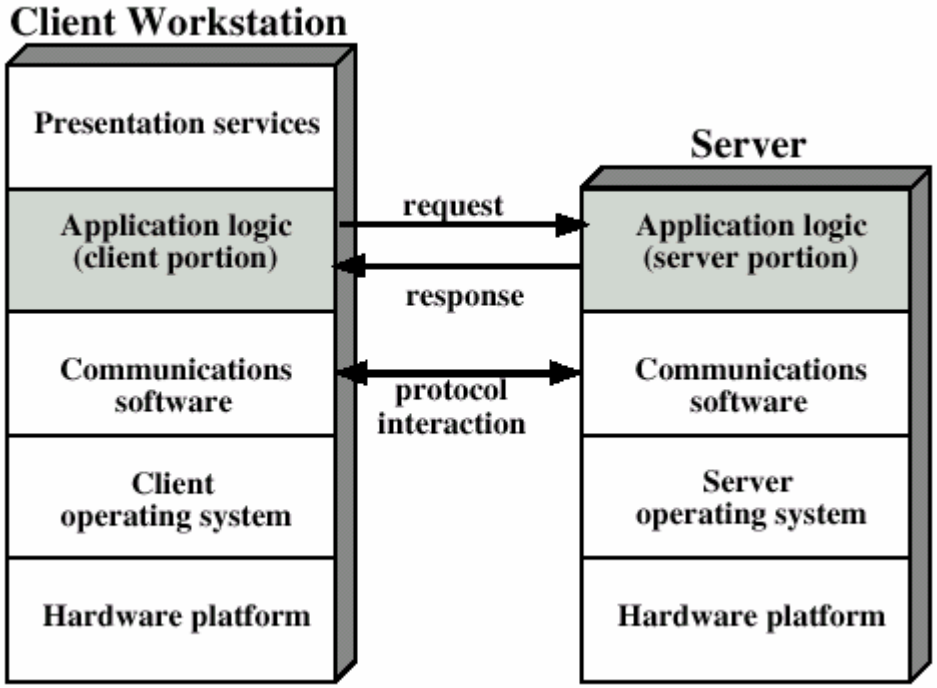
Grundbegriffe im Client / Server Umfeld

- Application Programming Interface (API):
  - Menge von Funktionen zur Client / Server Kommunikation
- Client
  - aktiver Kommunikationspartner, der Daten über ein Netzwerk abfragt, erzeugt, ändert, löscht
  - meist PC oder Workstation
- Server
  - Kommunikationspartner, der Daten über ein Netzwerk auf Anfrage zur erzeugt, zur Verfügung stellt, ändert, löscht
  - meist Hochleistungs-PC / Workstation, Mainframe
- Middleware
  - Menge von Treibern, APIs, Software, die die Verbindung zwischen Clientanwendung und Server erleichtert

Eine häufige Anwendung ist eine zentrale Datenhaltung in einer Datenbank mit Clients, die über ein Netzwerk zugreifen können

- Relational Database
- Structured Query Language (SQL)

Client Server Anwendungen



Client & Server dürfen bzgl Hardware und OS heterogen sein.

Kommunikationsprotokolle und Anwendungslogik müssen kompatibel sein.

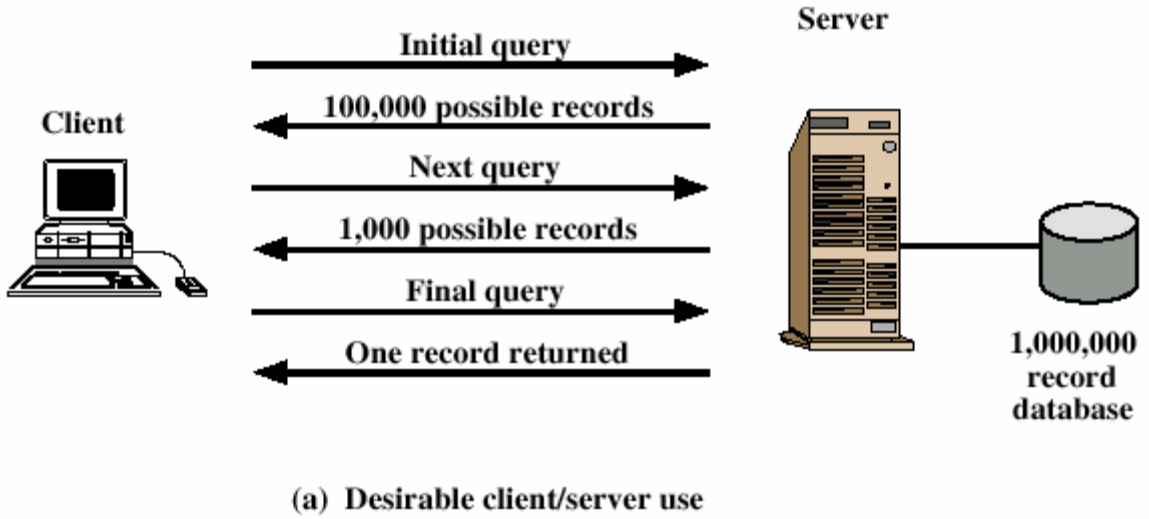
Verbindungselement:
Kommunikationssoftware, z.B. TCP/IP Implementierung

Eine Klassifizierung für Datenbank C/S Applikationen

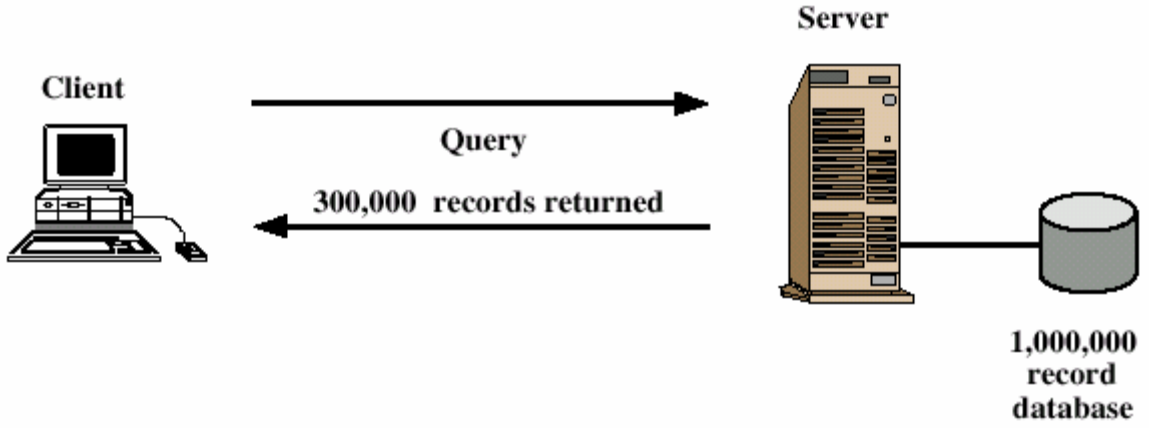
Klassifizierung nach Aufteilung der Präsentations-, Anwendungs- und Datenbanklogik

- Host-based Processeing
  - der Server berechnet alle Daten inklusive graphischer Darstellung
- Server-based Processing
  - der Server berechnet Daten bis zur Ebene der Anwendungslogik
  - der Client bearbeitet die Serverdaten für die Präsentationslogik
- Cooperative Processing
  - Client & Server kooperieren auf Ebene der Anwendungslogik
- Client-based Processing
  - Client & Server kooperieren auf Ebene der Datenbanklogik
  - der Client bearbeitet die Daten für die Anwendungs- und Präsentationslogik

Günstige und ungünstige Aufgabenverteilung



(a) Desirable client/server use

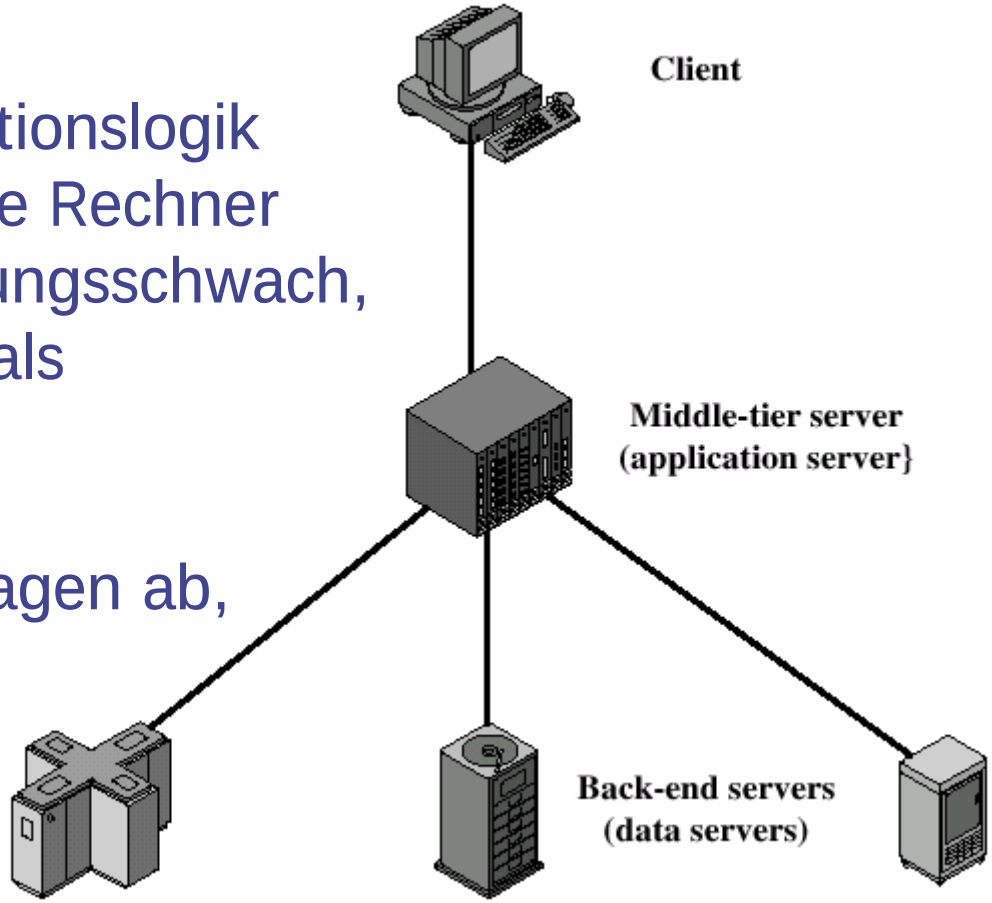


günstig:
leistungsstarker Server bearbeitet umfangreiche Operationen auf der Datenbank,
nur wenige Daten werden über das Netz transferiert

ungünstig:
wenig Aufwand für den Server,
viele Daten übermittelt, berechnet Resultat.
Erfordert Fat-Client + leistungsstarke Netzwerk.
Skaliert wg Netzwerkband-breite schlecht.
Wartungsaufwand wg Verteilung hoch.

3 Stufen C / S Architektur

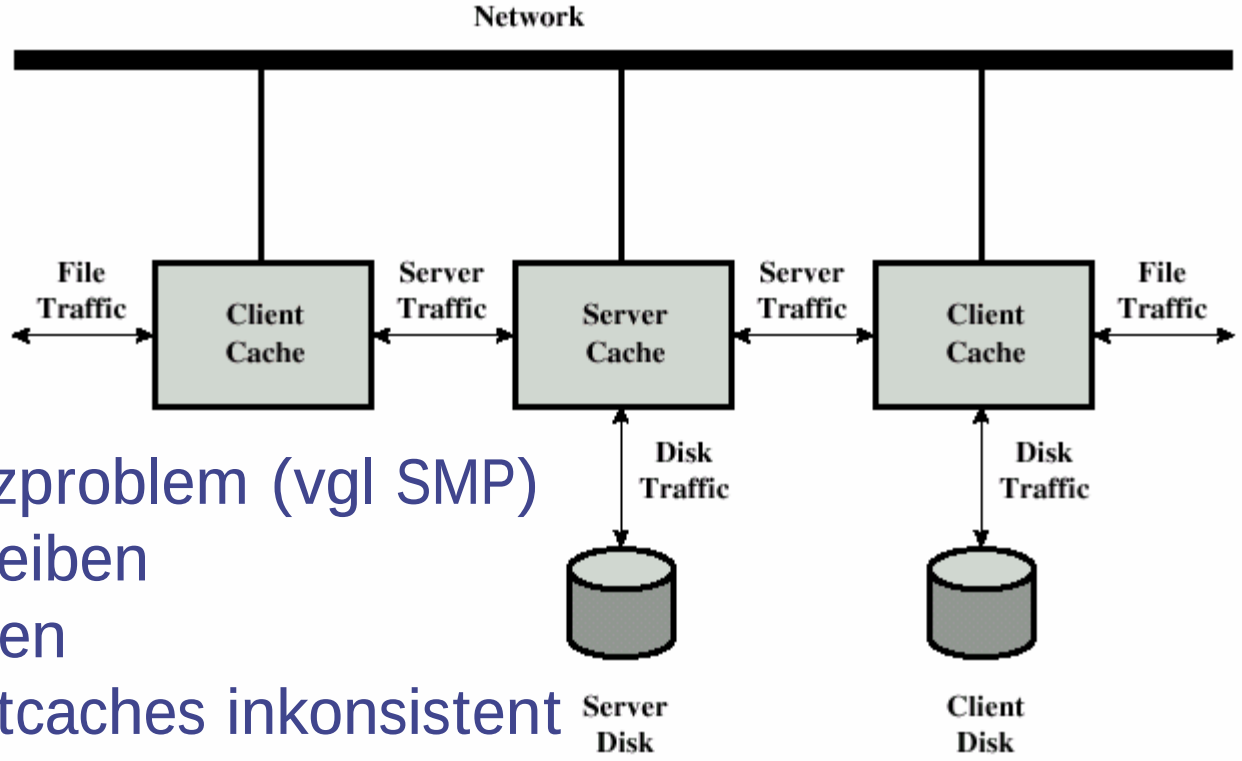
Aufteilung der Schichten:
Präsentationslogik, Applikationslogik und Datenbank auf einzelne Rechner
Client als „thin client“ leistungsschwach,
Middle-tier Server agieren als Verbindungsrechner,
bilden Client-Anfragen auf Server-Transaktionen/Anfragen ab,
verbinden unterschiedliche Protokolle.
Ein Möglichkeit zur Integration von Backend Legacy Systemen in Netzwerke.



(Legacy Systeme: historisch gewachsene Software, deren interne Struktur und Abhängigkeiten nicht mehr unter Kontrolle sind)

Konsistenzproblem bei Datei Cacheing

Weitere Anwendung: File Server



verschärft Cache Konsistenzproblem (vgl SMP)

- sofortiges Schreiben modifizierter Daten

=> andere Clientcaches inkonsistent

- verzögertes Schreiben modifizierter Daten

=> zusätzlich Serverdaten inkonsistent

Lösungen:

a) Locking, d.h. exklusiver Zugriff

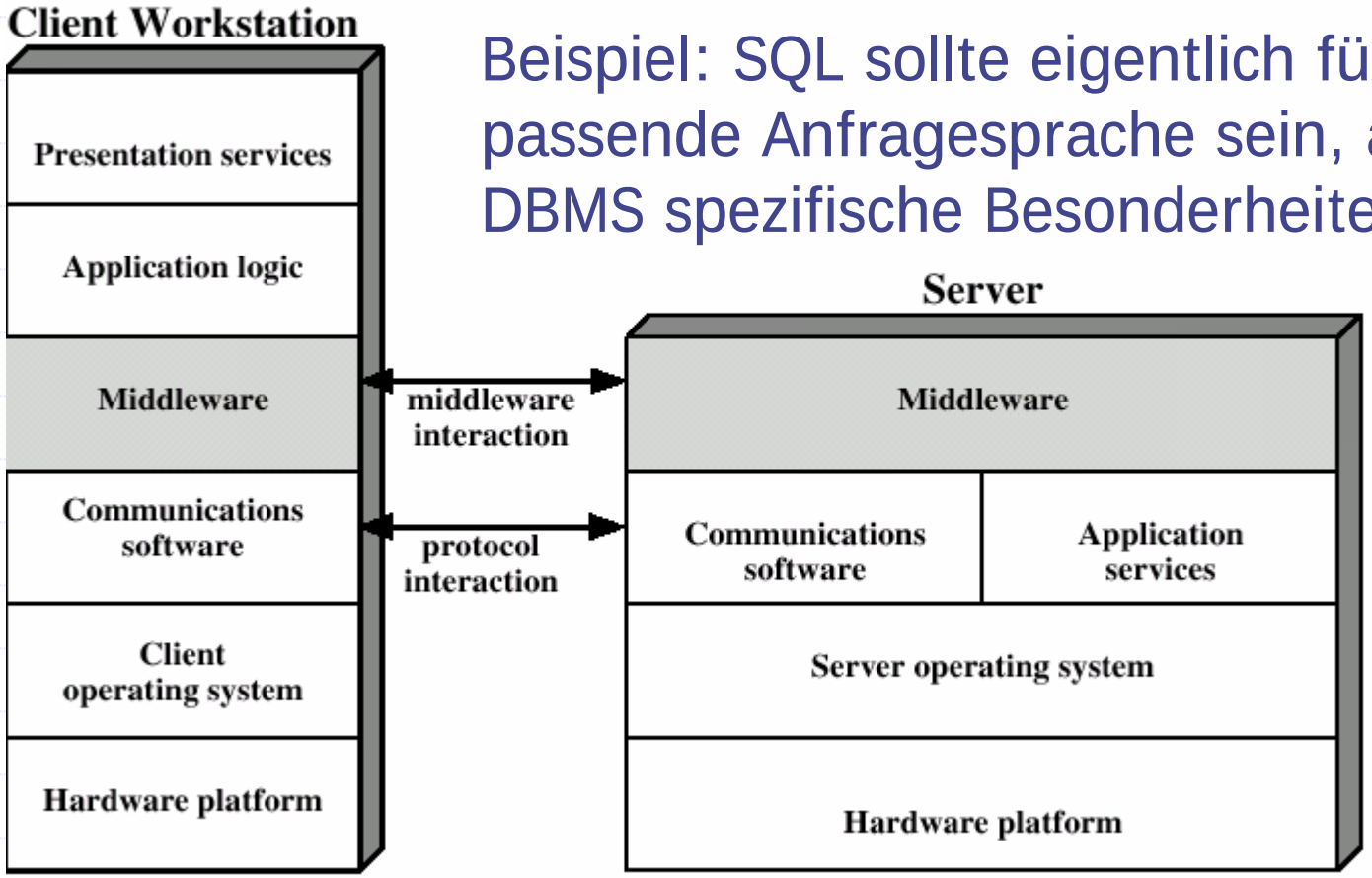
b) (in SPRITE):

für k Leser: benachrichtigt, dass Daten nicht cachebar

für 1 Schreiber: sofortiges Schreiben

Middleware

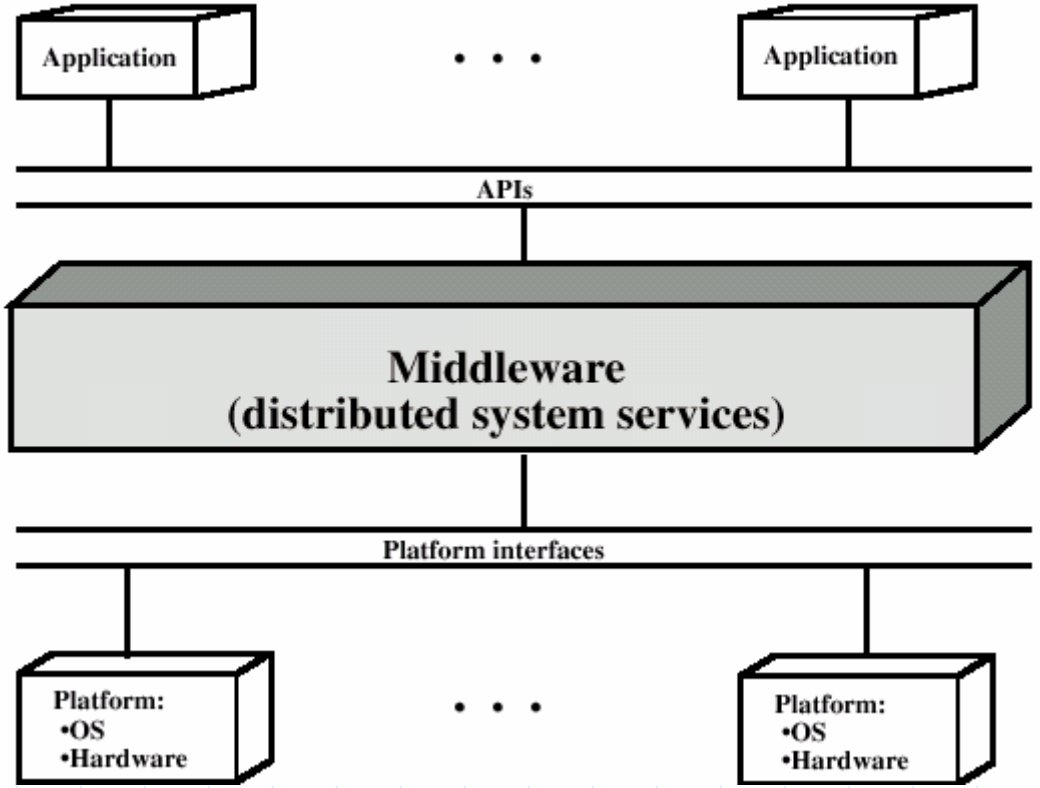
Standardisierte APIs und Protokolle um heterogene Systeme gleichartig für verteilte Applikationen nutzen zu können.
Rolle der Middleware variiert mit der Art der C/S Nutzung (client-based, server-based, cooperative)



Beispiel: SQL sollte eigentlich für diverse DBMS passende Anfragesprache sein, aber jeweils DBMS spezifische Besonderheiten ...

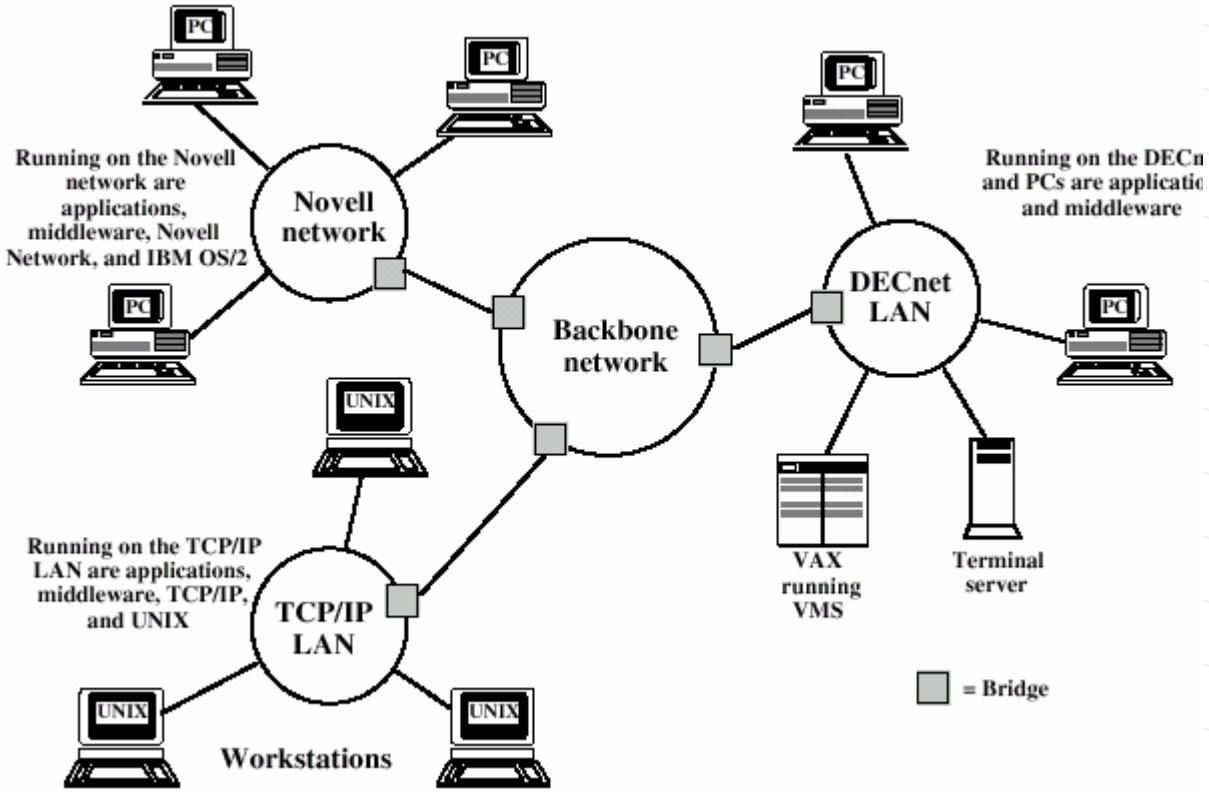
Middleware: logische Sichtweise

Middleware macht z.B. verschiedene Datenbanken einheitlich nutzbar



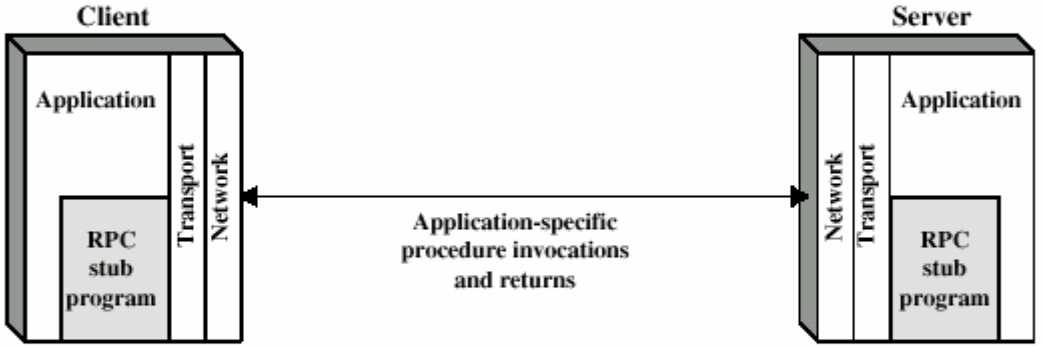
Middleware: zur Verdeckung heterogener Netze

Middleware verdeckt z.B. Unterschiede der eingesetzten Kommunikationssoftware: Novell, DECnet, TCP/IP und die Verbindung über das Backbone Netz

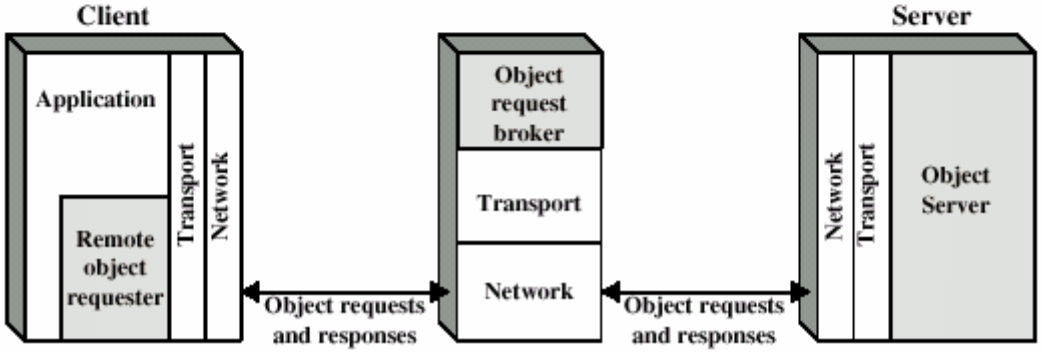


Message Passing und Remote Procedure Calls

(a) Message-Oriented Middleware



(b) Remote Procedure Calls

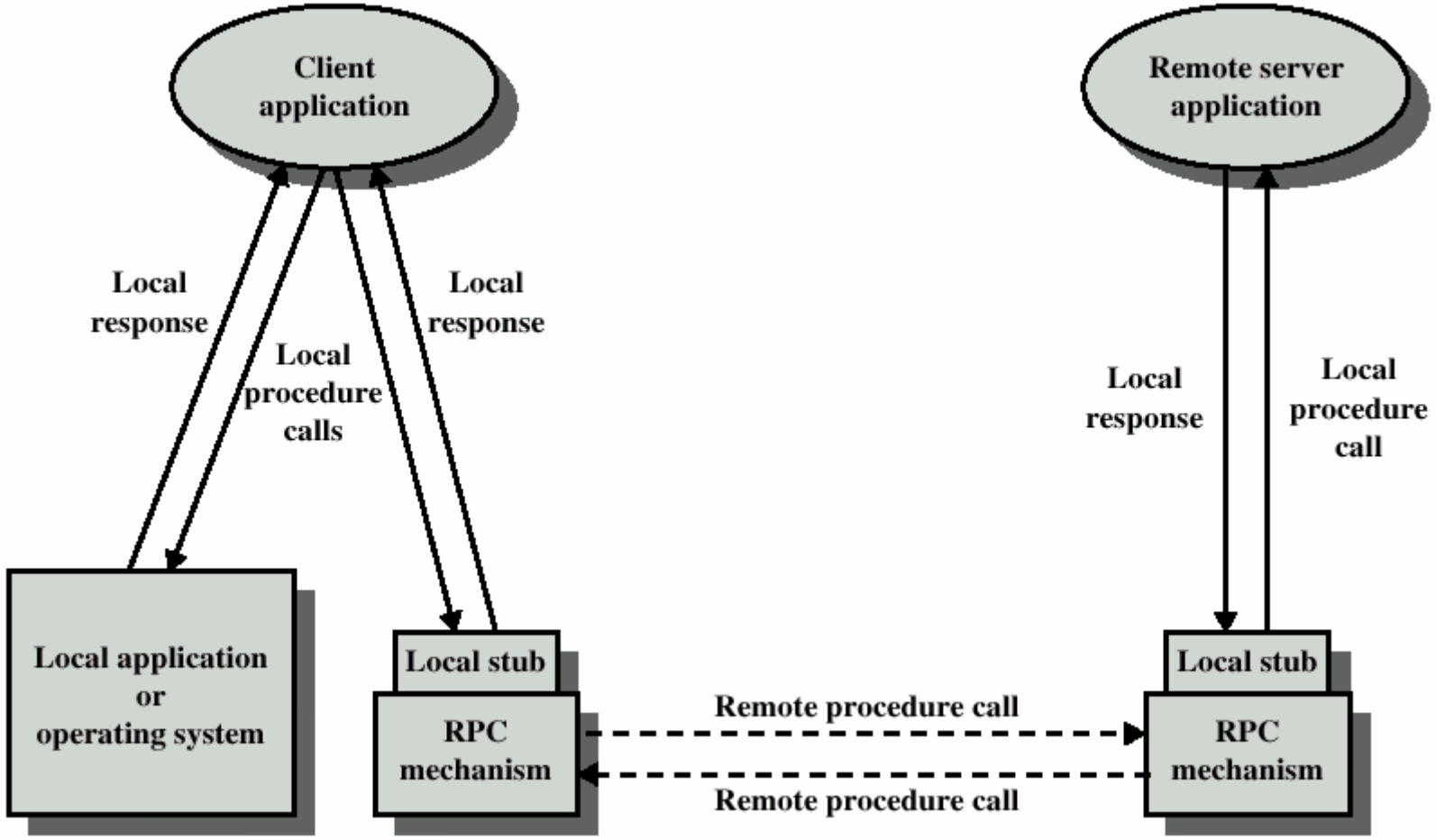


(c) Object request broker

Message Passing

- basiert im wesentlichen auf
  - send und receive Operationen
  - Eingangspuffer beim Empfänger für eingehende Nachrichten
  - synchrone Übermittlung (blocking): Sender-send und Receiver-receive nur paarweise, Kommunikationspartner warten
  - asynchrone Übermittlung (non-blocking): Receiver nimmt stets Nachrichten in den Puffer, receive entnimmt aus dem Puffer
- Nachrichten sind Tupel (ID,Nachricht)
  - plus Overhead beteiligter Kommunikationsprotokolle, die für die Übertragung genutzt werden
  - z.B. IP für Übermittlung, TCP für sichere Verbindung
- zuverlässige vs unzuverlässige Übermittlung
  - falls Übertragung vom Message Passing Modul garantiert wird, braucht der Sender nicht auf Bestätigung (Ack) zu warten
    - erfordert natürlich komplexeres Message Passing Modul
  - falls Übertragung unsicher, muß Sender & Empfänger durch Anfragen und Bestätigungen gelungene Übertragungen feststellen
    - erfordert nur ein elementares Message Passing Modul

Remote Procedure Call Mechanismus



verbreiteter Ansatz, kapselt Kommunikation in Funktionsaufrufen

RPC: CALL P(X,Y)

mit P = Funktionsbezeichner

X= übermittelte Parameter

Y = Rückgabewerte

Remote Procedure Call: Vorteile des Konzeptes

- Funktionsaufrufe sind als Abstraktionsmechanismus bekannt, üblich, akzeptiert
- Festlegung des Formates erlaubt die Entwicklung von Schnittstellen mit festgelegtem Funktionsumfang, getypten Parametern.

  Das Interface kann
  - dokumentiert werden,
  - statisch gegen Typfehler geprüft werden,
  - mit automatisch generiertem Kommunikationscode unterstützt werden
- Interface erlaubt die Entwicklung plattformunabhängiger Client / Server Module
- Dummy / Stub Module können den RPC vor dem Anwender verdecken

RPC

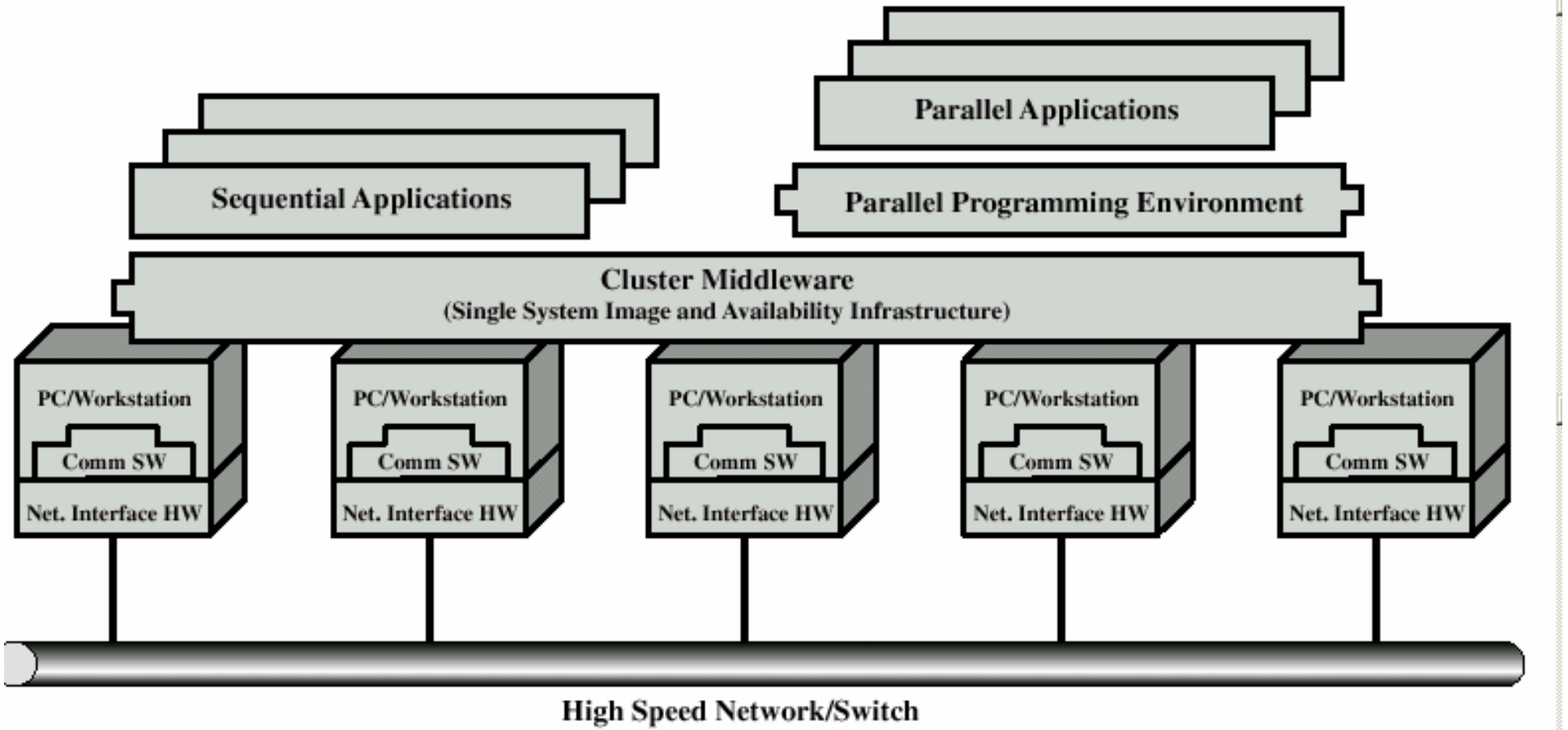
- Parameterübergabe
  - call by value: einfach möglich
  - call by reference: schwierig, erfordert eigentlich systemweite einheitliche logische Adressen (Pointer)
- Parameterdarstellung
  - erfordert einheitliche Typisierung oder passende Typkonvertierung für den Transfer, z.B. Integer, Float, Double, ...
  - wird bei Kommunikationsarchitektur von der Präsentationsschicht übernommen, häufig jedoch Overkill für RPCs
- Bindung von Prozessen bei RPCs
  - serverseitige Bearbeitung des RPCs erfordert einen Prozess und damit Ressourcen
  - bei non-persistenter Bindung wird der Prozess nur für die Zeit eines RPC Aufrufes gebunden
  - bei persistenter Bindung ist die Bindung etwas länger in der Hoffnung Aufwand für den Verbindungsaufbau/abbau bei wiederholten Aufrufen eines Clients einzusparen, Zeitbegrenzung durch Timeouts

RPC

- Kommunikation ist
  - üblicherweise synchron, d.h. der Client wartet auf die Beendigung des Funktionsaufrufs (sequentielle Programmierung)
  - bei asynchronem RPC blockiert der Client nicht,
    - er kann weitere RPCs absenden, um die Verbindung besser zu nutzen und dem Server eine Menge von Aufgaben zu übermitteln
    - Rückgabe der Werte kann unkontrolliert und jederzeit durch den Server erfolgen oder durch eine abschließende Synchronisation bewirkt werden
- Objektorientierte Mechanismen für RPC
  - via Object Request Broker werden Nachrichten zwischen Objekten übermittelt
  - Beispiele:
    - Microsofts Common Object Model (COM) als Basis für Object Linking und Embedding (OLE),
    - Object Management Groups (OMG) Common Object Request Broker Architecture (CORBA)

Cluster

Unabhängige Rechner werden per Netzwerk verbunden, erscheinen als 1 (Parallel-) Rechner.
Alternative zu SMP



Ziele, Hoffnungen, Vorteile durch Cluster

- Absolute Skalierbarkeit
  - Hunderte von (Mehrprozessor-) PCs machbar
- Inkrementelle Skalierbarkeit
  - neue Subsysteme können in kleinen Einheiten hinzugefügt werden
- Hohe Verfügbarkeit
  - Ausfall einzelner (Knoten-)rechner aufgrund der Unabhängigkeit und der Anzahl unkritisch
  - Fehlertoleranz kann durch Software erzielt werden
- Gutes Preis / Leistungsverhältnis
  - durch den Einsatz handelsüblicher Komponenten

Cluster Konfigurationen

- nach Verbindungen
  - Rechner sind untereinander verbunden
    z.B. LAN (geteilt oder exklusiv genutzt)
  - Rechner haben gemeinsam Zugriff auf eine Platte, RAID System
- nach Funktionalität
  - passiver Standbyrechner: 2. Rechner einsatzbereit, rechnet jedoch nicht, solange „Heartbeat“ Signale des Einsatzrechners eintreffen, steigert Verfügbarkeit, jedoch nicht die Performance,
    - kein Cluster im eigentlichen Sinne
  - Separate Server: eigenständige Rechner bearbeiten Tasks, Ausfallsicherheit erfordert Kopieroperationen zwischen den Rechnern, um bei Ausfall Daten zugreifbar zu haben
  - Shared Nothing: Recher teilen sich eine Platte, darin jedoch durch Volumes / Partitionen voneinander getrennt, so dass bei Ausfall eines Rechners Rekonfiguration erforderlich ist
  - Shared Disk: Rechner teilen sich eine Platte, können jeweils auf alle Partitionen zugreifen, erfordert daher Lock Mechanismus

Cluster erfordern Erweiterungen des Betriebssystems

- Failure Management, Behandlung von Ausfällen
  - Hochverfügbarkeitscluster: Komponenten sind mit hoher Wahrscheinlichkeit intakt
  - Fehlertolerante Cluster: Platten sind mehrfach (redundant) vorhanden, Transaktionskonzepte und Mechanismen zum Transferieren von Aufgaben (Failover) bei Ausfällen oder Failback Routinen zum Wiederaufsetzen nach Reparatur des Ausfalls
- Lastbalancierung
  - Last gleichmäßig auf Rechner verteilen
  - Inkrementelle Erweiterungen, Ausfälle berücksichtigen
  - Middleware muß Migrationsmöglichkeiten erkennen und nutzen
- Parallelisierung von Berechnungen
  - parallelisierende Compiler erkennen parallel berechenbare Teile, z.B. Schleifendurchläufe
  - parallele Applikationen, bei Softwareentwicklung bereits auf Cluster und Message Passing berücksichtigt
  - Parametric Computing: gleiches Programm wird mit Vielzahl unterschiedlicher Parameterwerte gestartet, z.B. Simulationsstudien, Optimierung mit evolutionäre Algorithmen (Computational Intelligence)

Middleware in einem Cluster

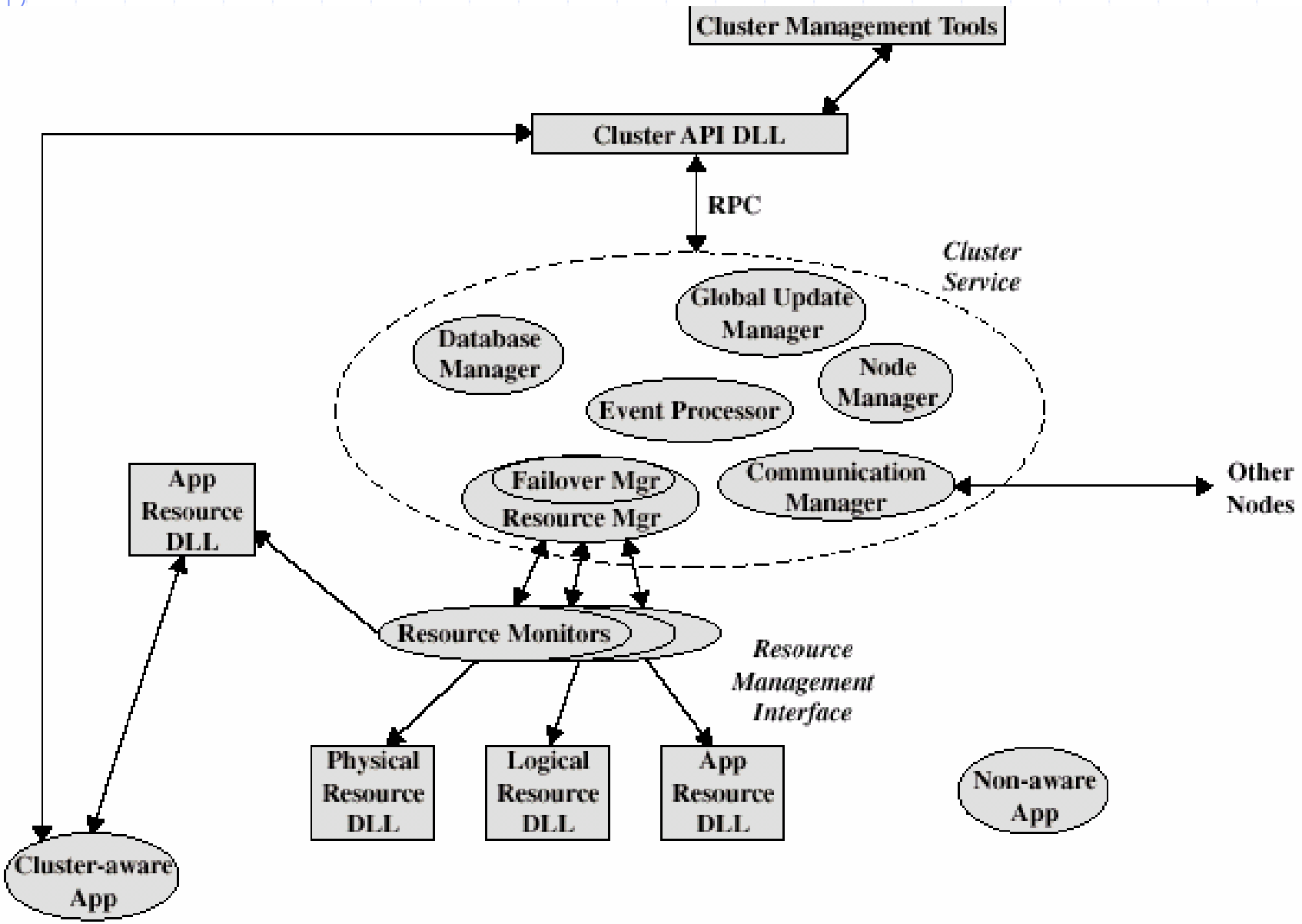
- tragende Rolle, bewirkt Illusion „single system image“
- leistet Lastbalancierung, Ausfallbehandlung, ...
- Anforderungen, jeweils nur 1 einziger/einzige
  - Zugangspunkt für Anwender Login
  - Dateihierarchie für Anwender
  - Steuerungspunkt, 1 Rechner für Cluster Management
  - virtuelles Netzwerk, alle Rechner (virtuell) mit allen verbunden
  - Adressraum, virtueller geteilter Speicher trotz Verteilung
  - Job-management System: Anwender startet Prozeß ohne Rechner zu kennen
  - Anwenderschnittstelle unabhängig vom Zugangsrechner
  - E/A Gerätemenge: unabhängig von der realen Gegebenheit
  - Prozeßraum: Prozeßids sind im ganzen Cluster eindeutig, Prozeßkommunikation zwischen Prozessen auch auf entfernten Rechnern
- darüberhinaus: Checkpointing und Prozeßmigration

Beispiel: Windows 2000 Server Cluster

- Shared Nothing Cluster,
  - jeder Knoten mit eigenem Plattenvolumen, eigenen Ressourcen
- Aspekte
  - Cluster Service: Softwaremodule die auf jedem Knotenrechner Clusteraktivitäten verwalten/bewirken
  - Ressource: verwaltetes Objekt des Cluster Services, repräsentiert reale oder logische Systemressource, durch DLLs implementiert
  - Online: Ressource ist an Knoten online, wenn sie dort verfügbar ist
  - Gruppe: Zusammenstellung von Ressourcen, die eine spezifische Applikation unterstützen, wird als Einheit verwaltet

    Gruppen reduzieren die Anzahl verwalteter Objekte für Failover und Lastbalancierung, Gruppen können verschoben werden
- Nodemanager:
  - lokal, erkennen Ausfälle, aktualisieren Knotenmenge im Cluster
- Konfigurationsdatenbankmanager
  - sorgt für Informationen bzgl Ressourcen, Gruppen, Zuordnung der Objekte zu Rechnern
- Ressource Manager/Failover Manager, ...

Beispiel: Windows 2000 Cluster Server (Wolfpack)

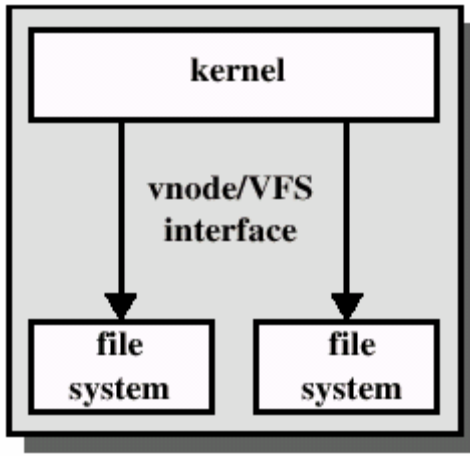


Beispiel: Sun Cluster

- verteiltes Sun Solaris Betriebssystem,
- Cluster erscheint als 1 Rechner mit Solaris (UNIX)
- zentrale Komponenten
  - Unterstützung von Objekten und Kommunikation mit CORBA
    - für RPCs und ein einheitliches Objektmodell,
  - Prozeßmanagement (global)
    - Aufenthaltsort des Prozesses ist für den Anwender nicht erkennbar
    - eindeutige, globale Prozeßids,
    - Prozeßmigration möglich, jedoch nur mit allen Threads
  - Netwerkfunktionalität
    - Cluster erscheint extern mit nur 1 IP Adresse
    - Paketfilter realisieren die interne Adressierung und Weiterleitung
    - Eingehende Verbindungen werden mit Lastbalancierung verteilt
    - Protokolloverhead entsteht jeweils am Start-/Zielknoten

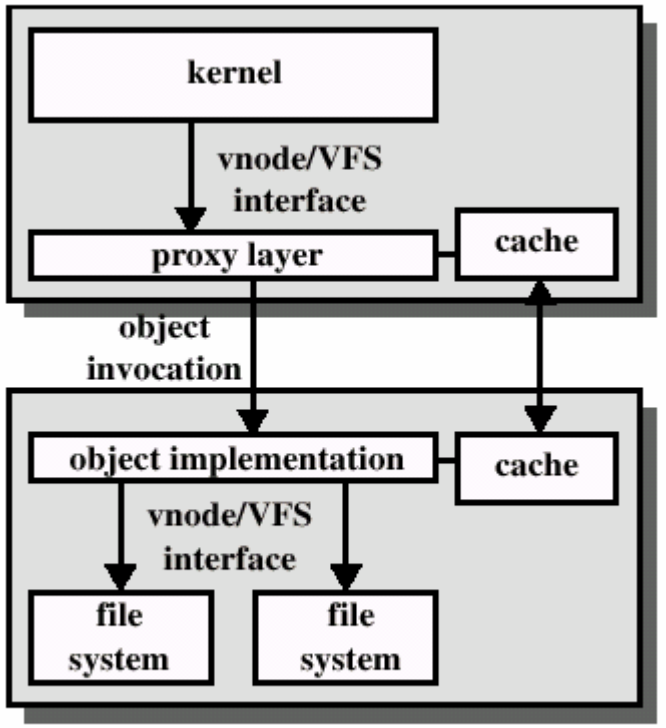
Sun Cluster, Erweiterungen des Dateisystems

Solaris: v=virtual node und file system
sind dateisystemunabhängig
inode war nur für UNIX Dateisysteme
vnode kapselt objektspezifische Realisierung
der read, write Operationen



(a) Standard Solaris

Im Cluster muß das Dateiobjekt
nicht lokal verfügbar sein.
Proxy = Stellvertreter, delegiert im
Netz Funktionsaufrufe für das
Dateisystem an das passende Objekt
im Cluster.
Cacheing wird wie üblich genutzt
um häufig verwendete Inhalte
parat zu haben:Dateiinhalte,-attribute
Directory Information



(b) Sun Cluster

Beispiel: Beowulf und Linux Cluster

- frei verfügbar: u.a. www.beowulf.org
- Feature
  - basiert auf preiswerter Hardware, PCs, keine Spezialkomponenten, keine speziellen Hersteller, Workstations und PCs können unterschiedlich sein (heterogen), übliche Verbindungskomponenten, z.B. LAN Ethernet
  - Prozessoren und LAN werden fest und exklusiv zugeordnet
  - Skalierbare E/A
  - freiverfügbare Softwarebasis, insbesondere auf LINUX mit Beowulferweiterungen
- jeder Knoten hat eigenen Kernel, ist unabhängig
- Cluster mit verteiltem, geteiltem Speicher

Beowulf Besonderheiten

- verteilter Prozeßraum:
  - Prozeßids über mehrere Knoten eindeutig,
  - erlaubt u.a. Erzeugung entfernter Prozesse, die in der lokalen Prozesstabelle dann ebenfalls erscheinen
- Ethernet Channel Bonding:
  - verbindet mehrere (preiswerte) Netzverbindungen in eine logische Verbindung größerer Bandbreite,
  - erlaubt Lastbalancierung über mehrere Ethernetze, die ein Cluster verbinden
- PVMSync:
  - Programmierumgebung um Synchronisation und geteilte Datenobjekte für Prozesse zu unterstützen
- ENFuzion:
  - Werkzeugset zur Unterstützung von Parametric Computing, also der Durchführung einer Vielzahl von Programmaufrufen mit jeweils variierten Eingangsparameterwerten

Zusammenfassung

- Client / Server Computing
  - Varianten, Middleware, Cache Consistency
- Verteiltes Message Passing
  - Zuverlässigkeit, Blocking vs Non-Blocking
  - Remote Procedure Calls
  - Funktionsweise, Parameterübergabe, Bindung von Prozessen
  - Synchrone vs Asynchrone RPCs
  - Objekt-orientierte Mechanismen, CORBA, COM
- Cluster Computing
  - Skalierbarkeit, Verfügbarkeit, Preis/Leistungsverhältnis
  - Konfigurationen, Failure Management, Last Balancierung
  - Architektur
- Beispiele für Cluster Architekturen
  - Windows 2000, Sun, Beowulf und Linux Cluster