Verteilte Systeme

Oktober – November 2023

2. Vorlesung – 12.10.2023

Kurs:	TINF21AI1
Dozent:	Tobias Schmitt, M.Eng.
Kontakt:	d228143@
	student.dhbw-mannheim.de

Wiederholungsfragen

- Wie ließe sich ein verteiltes System definieren?
- Welche Gründe gibt es für den Einsatz von verteilten Systemen?
- Mit welchen Arten von Verteilungstransparenz haben Sie bei verteilten Systemen zu tun?
- Welche Arten von Systemarchitekturen kennen Sie?
- Was verstehen Sie unter einer Client-Server-Architektur?
- Was verstehen Sie unter einem Thin- und einem Fat-Client?
- Was verstehen Sie unter eine 2-Tier- und unter unter einer 3-Tier-Architektur?
- Bei einer Client-Server-Server-Struktur, welche Aufgaben kann der mittlere Server übernehmen?
- Was verstehen Sie unter den Begriffen Interoperabilität und Portabiblität?

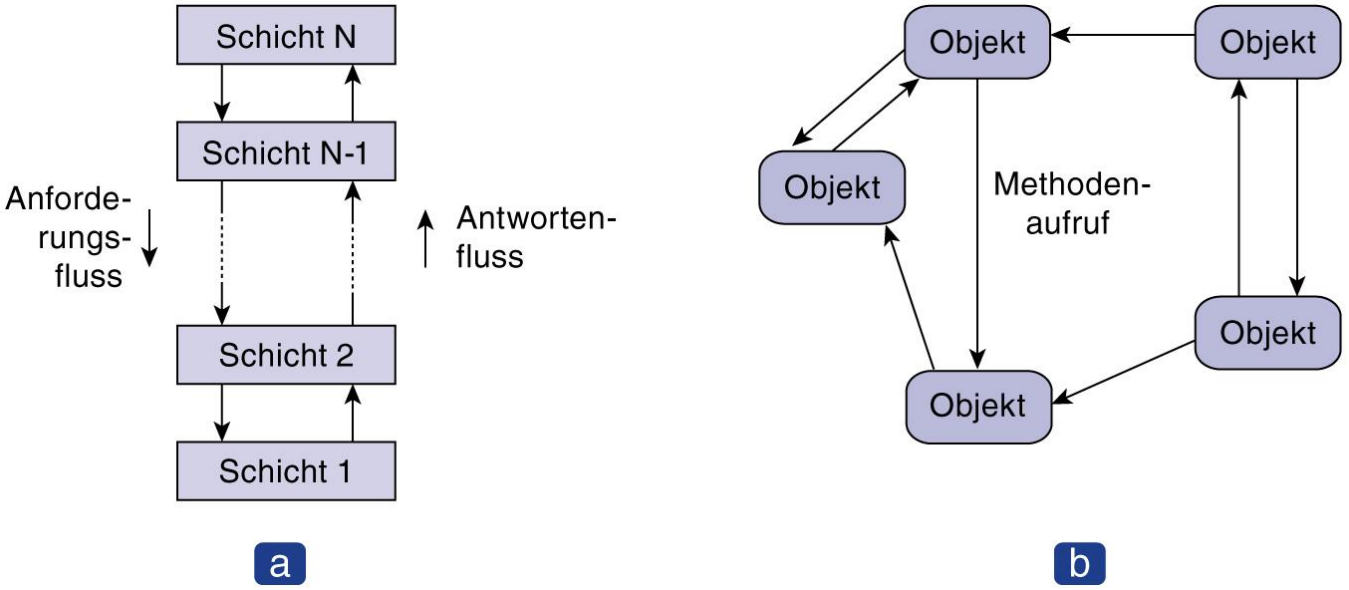
Themenüberblick heute

- **Architekturen**
  - Architekturstile
  - Systemarchitekturen
- Prozesse und Threads

Architekturstile

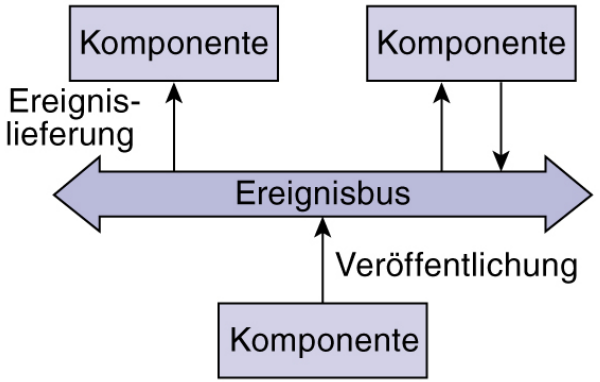
- Geschichtet Architekturen
  - Schicht L_i ruft Schicht L_{i-1} auf (z.B. bei Netzwerken)
- Objektbasierte Architekturen
  - Jede Komponente ein Objekt → losere Anordnung
  - Übereinstimmung mit Client-Server-Architektur

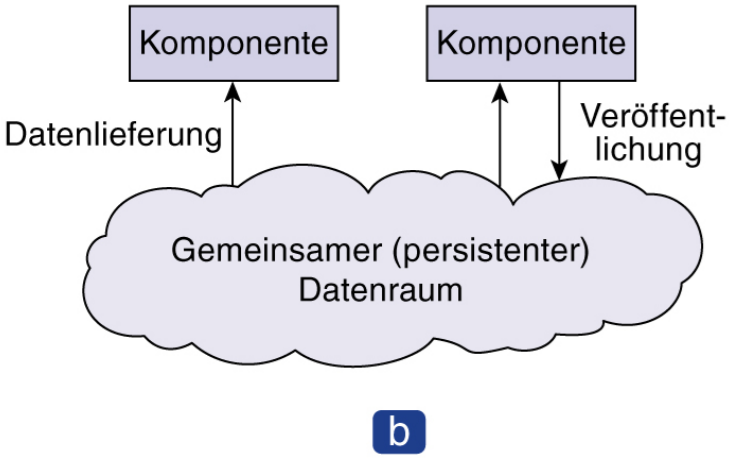
Architekturstile



Architekturstile

- Ereignisbasierte Architekturen
  - Prozesse kommunizieren über Weitergabe von Ereignissen
- Datenzentrierte Architekturen
  - Idee der Kommunikation über gemeinsames Repository



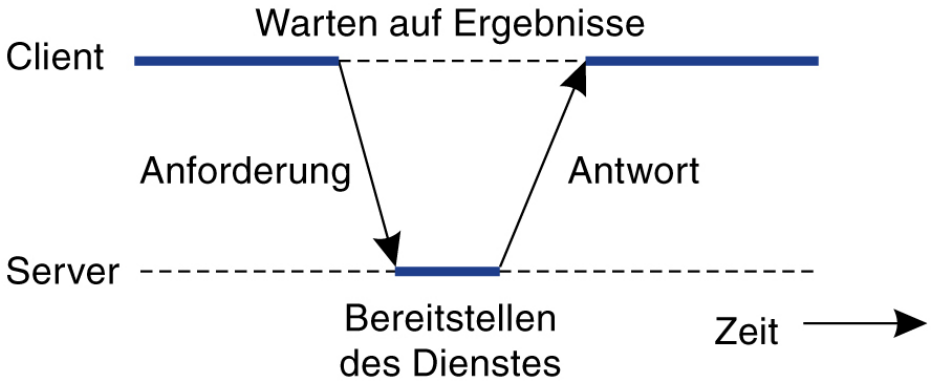


Systemarchitekturen

- Zentralisierte Architekturen
  - Multitier-Architekturen
- Dezentralisierte Strukturen
  - Peer-to-Peer-Systeme
- (Hybridarchitekturen)

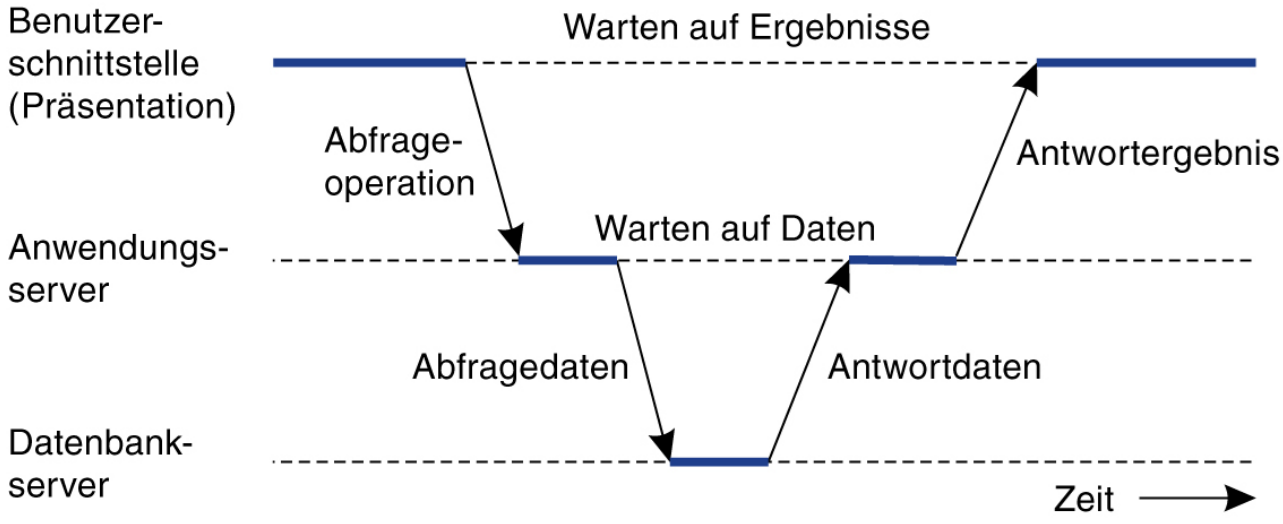
Zentralisierte Architekturen

- Entsprechung: Client-Server-Modell
- Server
  - Prozess, der einen bestimmten Dienst implementiert
- Client
  - Prozess, der einen Dienst von einem Server anfordert
- Client-Server-Zusammenarbeit
  - Entspricht *Anforderungs-/Antwortverhalten*
  - Bzw. *Request/Reply*



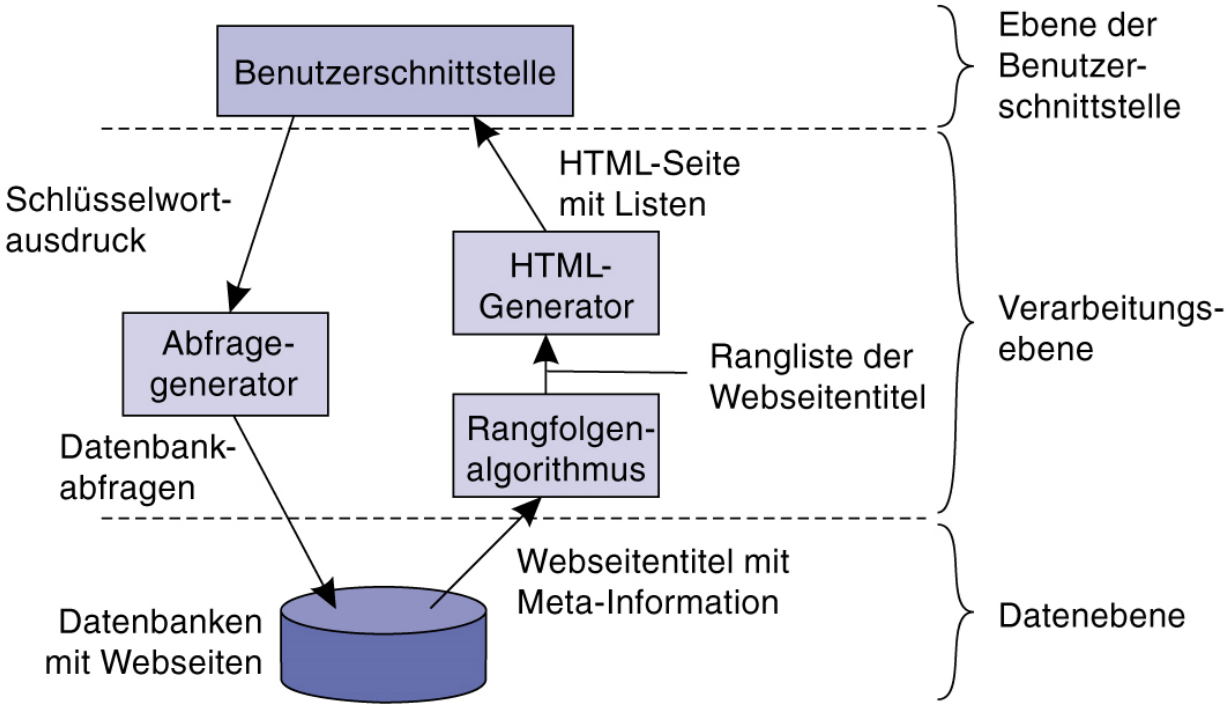
Zentralisierte Architekturen
Anwendungsschichten

- Client-Server-Modell problematisch
  - Klare Unterscheidung nicht immer möglich.
- Aufteilung in 3 Ebenen
  - Ebene der Benutzerschnittstelle (presentation layer)
    - Zusammenarbeit mit dem Benutzer, Darstellungsebene
  - Verarbeitungsebene (application layer)
    - Enthält normalerweise die Anwendung, Kernfunktionalität
  - Datenebene (data access layer)
    - Verwaltung der Daten



Zentralisierte Architekturen

- Beispiele:
  - Suchmaschine
  - Entscheidungssystem für den Aktienhandel
  - Desktop-Paket
  - …



Bsp.: Suchmaschine

Zentralisierte Architekturen

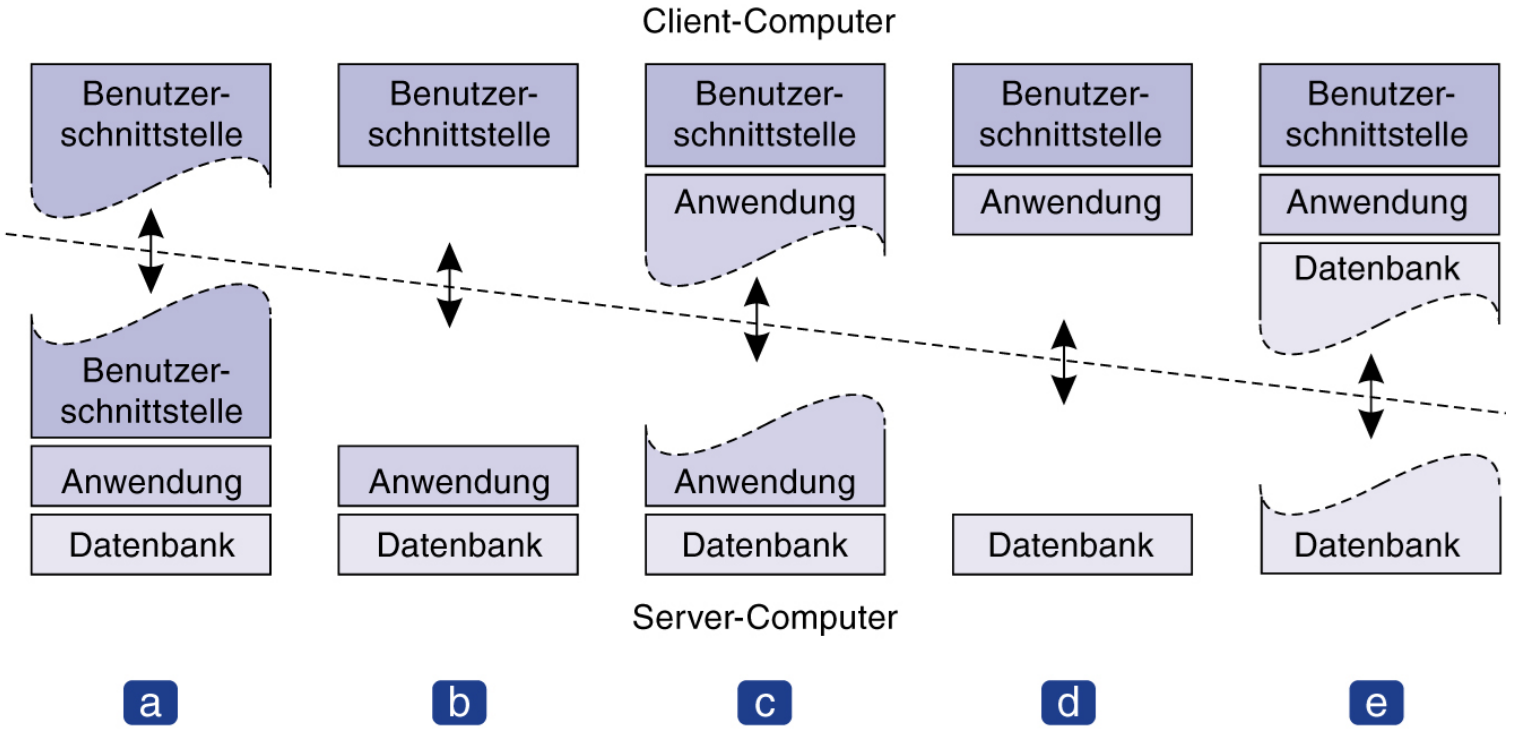
- Rahmenbedingung:
  - 1 Computer = Client
  - 1 Computer = Server
- Frage:

  Wie lassen die 3 Ebenen auf Client und Server verteilen?

- Erinnerung:
  - Benutzerschnittstelle
  - Verarbeitungsebene
  - Datenebene

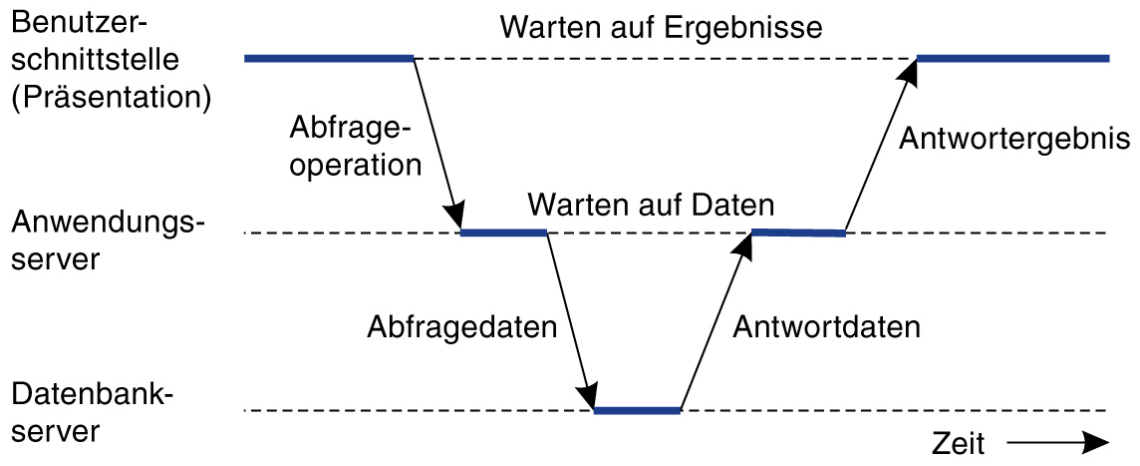
Zentralisierte Architekturen
Multitier-Architekturen

- Zwei-Tier-Architektur / Zwei-Schicht-Architektur
  - Aufteilung der Schichten auf 2 Computer
  - Grundlage: 1 Client und 1 Server
  - Stichwort: Thin Clients und Fat Clients (Thick Clients)



Zentralisierte Architekturen
Multitier-Architekturen

- Drei-Tier-Architektur
  - Typisches Beispiel: Webseiten
  - Einfacher Aufbau
    - Benutzerschnittstelle (Präsentation)
    - Anwendungsserver
    - Datenbankserver



Beispiel:

Web-Frontend

Apache-Server

SQL-Server
(MySQL, MariaDB)

Zentralisierte Architekturen Erweiterungen

- Fragen:

  Welche Vorteile könnte haben eine Client-Server-Server-Struktur zu haben?

  Welche Rollen / Aufgaben könnte der mittlere Server haben?

Zentralisierte Architekturen Erweiterungen

- Aufbau: **Client-Server-Server**
- Vorhandensein
  - mehrerer Clients
  - mehrerer Endserver
- Aufgaben des Zwischenservers
  - Proxy
  - Broker
  - Trader
  - Filter
  - Balancer
  - Koordinator
  - Agent

Zentralisierte Architekturen Erweiterungen

- Proxy
  - Vertretung mehrerer Server
  - Ggf. Caching von aufgerufenen Dokumenten
  - Ggf. Entlastung des Web-Servers oder sogar der allgemeinen Netzlast
- Broker
  - Vermittler zwischen Clients und Serven
  - Besitzt Informationen über benannte Services und bietet Zugriffsmöglichkeit darauf für Clients
  - Zweck: Ortstransparenz

Zentralisierte Architekturen Erweiterungen

- Broker (noch mehr Wissenswertes)
  - Unterteilung
    - Intermediate Broker / forwarding Broker
      - Weiterleitungseigenschaften
    - Separater Broker / handle-driven Broker
      - Rückgabe aller Infos an Client, damit dieser mit entsprechenden Service eines Servers zu interagieren
    - Hybrider Broker
      - Funktionalität je nach Anfrage
  - Problem
    - Zentraler Broker → Ausfallproblematik und Flaschenhals
  - Lösung
    - Replikation des Brokers

Zentralisierte Architekturen Erweiterungen

- Trader
  - Ausgangspunkt: mehrere Server für einen Service, aber ggf. in unterschiedlicher Qualität
  - Trader übernimmt Auswahl des geeignetesten Servers für Client
- Filter
  - Aufgabe: Anfragen an oder Antworten vom Server analysieren und modifizieren
  - Möglichkeit zur Auslagerung wiederkehrender Tätigkeiten des Servers zur Vor- bzw. Nachbearbeitung
  - z.B. für Verschlüsselung, Datenkomprimierung, Logging

Zentralisierte Architekturen Erweiterungen

- Balancer
  - Annahme: mehrere identische bzw. replizierte Server stehen zur Verfügung
  - Sorgt für Gleichauslastung (kein unter- oder überbelasteter Server)
- Koordinator
  - Annahme: Service besteht aus mehreren Einzelservices (Durchlauf nach Reihenfolge, alternative Reihenfolgen mgl. oder parallele Ausführung)
  - Verbirgt replizierte Server, sich ergänzende Server, mehrere Teilservices
  - Stichwort: Service Composition

Zentralisierte Architekturen Erweiterungen

- Agent
  - Zweck: mehrere und komplexere Serviceleistungen in Anspruch nehmen
  - Serviceanfrage wird zerlegt und koordiniert
  - Rückantworten werden gesammelt und zu einziger Rückantwort zusammengestellt
  - Abgrenzung zum Koordinator: Selbstbestimmung des Agenten über aufzurufende Services
  - Arbeitsweise zwischen Agenten und Servern
    - iterativ oder parallel

Serverbetrachtung

- Entwurfsentscheidung bei einem Server
  - Zustandsloser Server
  - Zustandsbehafteter Server

Serverbetrachtung

- **Zustandsloser Server**
  - Speichert keine Zustandsinfos der Clients
  - Zustandsänderung möglich ohne Clients zu informieren
  - Beispiel: Webserver
  - Protokollierung von Clientinfos nicht ausgeschlossen
    - Zweck: Optimierung der Leistung (z.B. Infos, ob Replizierung einzelner Dokumente nötig ist)

Serverbetrachtung

- Zustandsloser Entwurf mit „weichem“ Status („Soft State“)
  - Sonderform
  - Aufzeichnung von Zustandsinfos für Client für begrenzte Zeit
  - Nach Zeitablauf: Standardverhalten des Servers
  - Anwendung: Server, der Client über Aktualisierungen für best. Zeit auf dem Laufenden halten muss
    - z.B. Protokollentwurf bei Computernetzwerken

Serverbetrachtung

- **Zustandsbehafteter Server**
  - Vorhalten der Infos über Clients
    - Unterscheidung zwischen (temporären) Sitzungsstatus und permanentem Status
  - Explizites Löschen vom Server nötig
  - Beispiel: Dateiserver, der Client ermöglicht lokal Kopien anzulegen
  - Vorteil: Leistungsverbesserung

Serverbetrachtung

- **Zustandsbehafteter Server**
  - Nachteil: Absturz des Servers verursacht Probleme
    - Wiederherstellung des gesamten Zustandes vor dem Absturz notwendig
  - Workaround z.B. bei Webbrowsern
    - Verwendung von Cookies

Zentralisierte Architekturen Erweiterungen

- Client-Server-Ketten
  - z.B. Client – Agent – Broker – Server ($C^+S_AS_{Br}S^+$)
  - Rekursiv – Anfrage an nächsten Prozess der Rekursionsstufe und Rückantwort wieder an die vorhergehende Rekursionsstufe
  - Transitiv – Anfrage an nächsten Prozess der Rekursionsstufe, aber Rückantwort an den Client
- Client-Server-Bäume

Dezentralisierte Architekturen

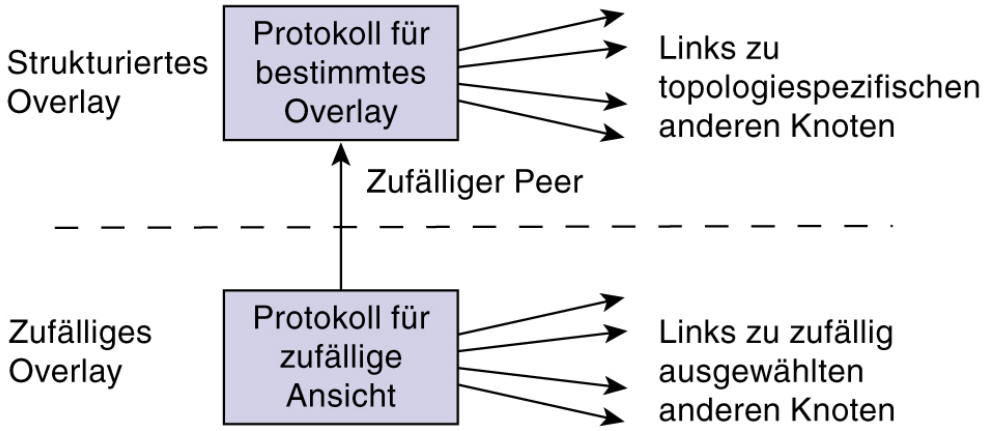
- Peer-to-Peer-Systeme
  - Client und Server horizontal verteilt
  - Prozesse sind symmetrisch und agieren gleichzeitig als Client und als Server (Servent = Server+Client)
  - Anordnung des Overlay-Netzwerks
    - Strukturierte Peer-to-Peer-Systeme
    - Unstrukturierte Peer-to-Peer-Systeme

Dezentralisierte Architekturen

- Strukturierte Peer-to-Peer-Architekturen
  - Konstruierung des Overlay-Netzwerks durch deterministische Verfahren
  - Häufig eingesetzt: verteilte Hash-Tabellen (distributed Hash-Table, DHT)
  - Struktur vereinfacht Zugriff auf angefragte Datenelemente
- Unstrukturierte Peer-to-Peer-Architekturen
  - Zufallsalgorithmen zur Konstruktion des Overlay-Netzwerks
  - Idee: jeder Knoten hat Liste von Nachbarn
  - Datenelemente sind zufällig auf Knoten verteilt
  - Suchanfrage auf bestimmte Datenelemente führen zum Überfluten des Netzwerkes

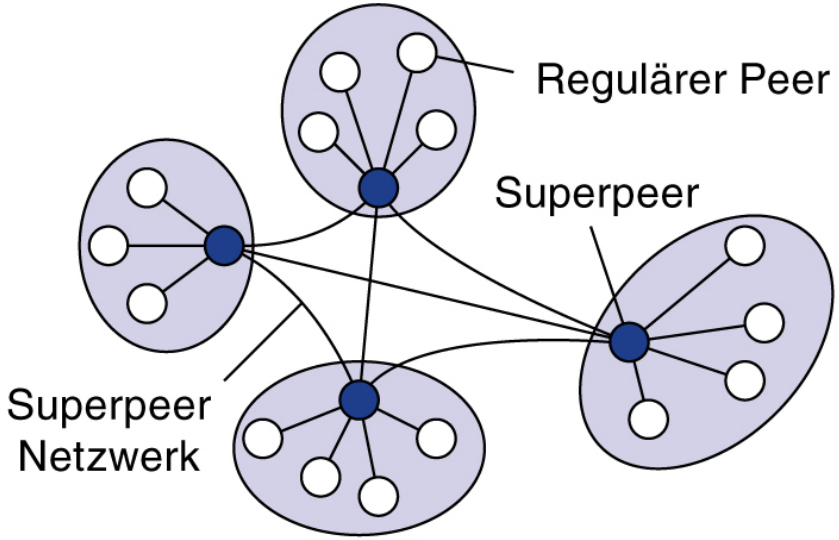
Dezentralisierte Architekturen

- Topologieverwaltung in Overlay-Netzwerken
  - Idee: unstrukturierte Peer-to-Peer-Systeme tauschen Einträge der Listen zu Nachbarn aus
  - Sorgfältiges Austauschen und Auswählen der Einträge kann zu strukturiertem Overlay-Netzwerk führen
  - Realisierung durch 2-Schicht-Ansatz



Dezentralisierte Architekturen

- Superpeer-Konzepte
  - Ansatz bei anwachsenden unstrukturierten Peer-to-Peer-Systemen
    - Problem: Anordnung relevanter Datenelemente
    - Last auf das Netzwerk wächst beim Durchsuchen
  - Auswahl eines Knotens als Superpeer
    - Agiert als Makler (Broker)
    - Oder verwaltet Index (Caching)
  - Client-Superpeer-Beziehung meist fest
  - Problem bei Ausfall eines Superpeers
    - Führerauswahlproblem



Themenüberblick

- Architekturen
- **Prozesse und Threads**
  - Threads in verteilten Systemen
  - Virtualisierung
  - Codemigration / Prozessmigration

Prozesse und Threads - Fragen

- Welche Komponenten machen einen Prozess aus?
- Wo werden Threads verwaltet?
  - Wie kann man die beiden unterschiedlichen Verwaltungsformen unterscheiden?
- Welche Rolle spielen Threads in verteilten Systemen?

Prozesse und Threads

- Modell eines Prozesses
  - Codesegment
    - Befehlssatz
  - Ressourcensegment
    - Referenzen auf externe Ressourcen
    - z.B. Dateien, Drucker, ...
  - Ausführungssegment
    - Speicherung des aktuellen Ausführungszustandes
    - u.a. private Daten, Stapel (Stack), Programmzähler

Prozesse und Threads

- Betriebssystem: Multiprozessbetrieb ist **transparent**.
- Aspekte, die verborgen werden:
  - Erstellung eines Prozesses
    - Initialisierung des Speichers
    - Kopie des Programmcodes im Textsegement
    - ...
  - Wechsel eines Prozesses
    - Sicherung des CPU-Kontexts (Registerwerte, Programmzähler, Stack-Zeiger, ….)
    - Register der Memory Management Unit ändern
    - ...

Prozesse und Threads

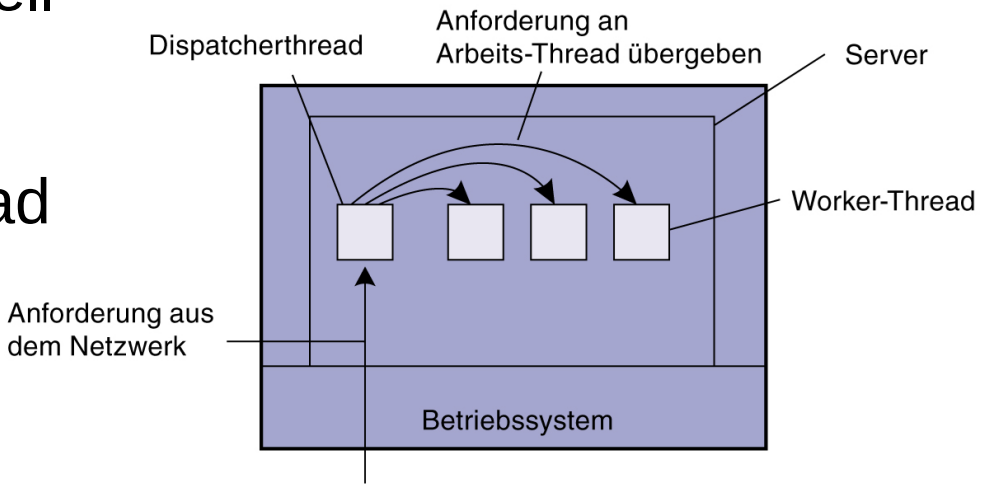
- Möglichkeiten der Thread-Verwaltung
  - Variante 1 – im Benutzerraum - **Benutzerthreads**
    - Threadtabelle und Scheduling im Benutzerraum
    - sehr schnelle Anlage und Zerstörung von Threads
    - Kern nimmt nur einen Prozess wahr
    - Problem bei blockierenden Systemaufrufen

Prozesse und Threads

- Mischform zwischen Benutzer- und Kernalthreads
  - Variante 2 – im Kernadressraum - **Kernelthreads**
    - Unterstützung durch das Betriebssystem notwendig
    - Threadtabelle wie Prozesstabelle im Kern
    - Erzeugung und Vernichtung kostspieliger, da Systemaufrufe
  - Lightweight-Prozesse (LWG)

Threads in verteilten Systemen

- Multithread-Clients
  - Ziel: Verbergen von Kommunikationsverzögerungen
  - Beispiel: Webbrowser (außer Chrome)
    - Jeder Thread hat eigene Verbindung zu Server.
- Multithread-Server
  - z.B. Dispatcher/Worker-Modell
  - Dispatcher liest Anforderung
  - Weitergabe an Worker-Thread
    - Blockierende Aufrufe im
    - Worker-Thread möglich



Interludium – Apache Server

- Arbeitsweisen des Apache Servers
  - u.a. Apache-MPM* prefork und Apache-MPM* worker
- Apache-MPM prefork
  - Ein einzelner Steuerprozess ist für den Start von Kindprozessen verantwortlich, die auf Verbindungen warten und diese bedienen, sobald sie eintreffen. Der Apache versucht immer, mehrere freie oder unbeschäftigte Serverprozesse vorzuhalten, die zur Bedienung eingehender Anfragen bereit stehen. Auf diese Weise müssen Clients nicht darauf warten, dass neue Kindprozesse geforkt werden, bevor ihre Anfrage bearbeitet werden kann.

(https://httpd.apache.org/docs/2.4/de/mod/prefork.html)

*MPM - Multi-Processing-Module

Interludium – Apache Server

- Apache-MPM worker
  - Ein einzelner Steuerprozess (der Elternprozess) ist für den Start der Kindprozesse verantwortlich. Jeder Kindprozess erstellt eine feste Anzahl von Server-Threads, wie durch die *ThreadsPerChild*-Direktive angegeben, sowie einen "Listener-Thread", der auf Verbindungen wartet und diese an einen Server-Thread zur Bearbeitung weiterreicht, sobald sie eintreffen.
  - Der Apache versucht immer, einen Vorrat von freien oder unbeschäftigten Threads zu verwalten, die zur Bedienung hereinkommender Anfragen bereit stehen. Auf diese Weise brauchen Clients nicht auf die Erstellung eines neuen Threads oder Prozesses zu warten, bevor ihre Anfrage bedient werden kann. Die Anzahl der Prozesse, die anfangs gestartet wird, wird mit der Direktive *StartServers* festgelegt. ...

(https://httpd.apache.org/docs/2.4/de/mod/worker.html)

Themenüberblick

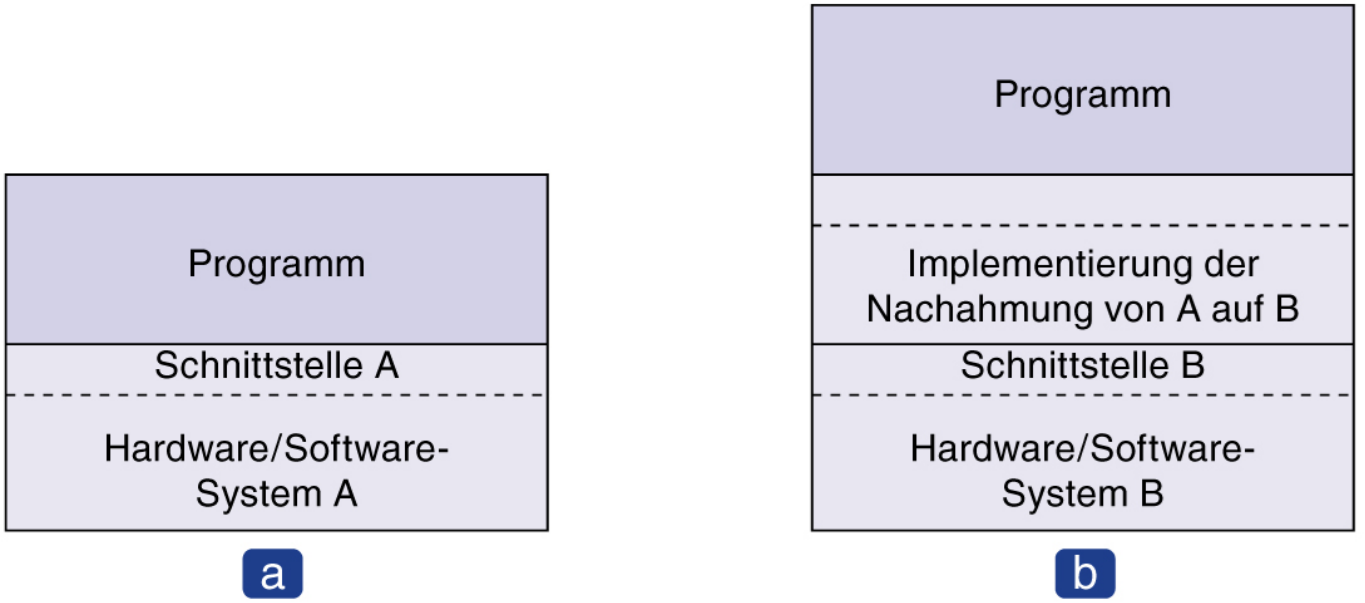
- Architekturen
- **Prozesse und Threads**
  - Threads in verteilten Systemen
  - **Virtualisierung**
  - Codemigration / Prozessmigration

Virtualisierung - Fragen

- Was verstehen Sie unter Virtualisierung?
- Welche Gründe gibt es für die Verwendung von Virtualisierung?
- Welche Einsatzmöglichkeiten für Virtualisierung kennen Sie?
- Welchen Vorteil bietet Virtualisierung im Rahmen verteilter Systeme?

Virtualisierung

- Ziel der Virtualisierung
  - Bestehende Schnittstelle so zu erweitern oder zu ersetzen, dass sie das Verhalten eines anderen Systems nachahmt



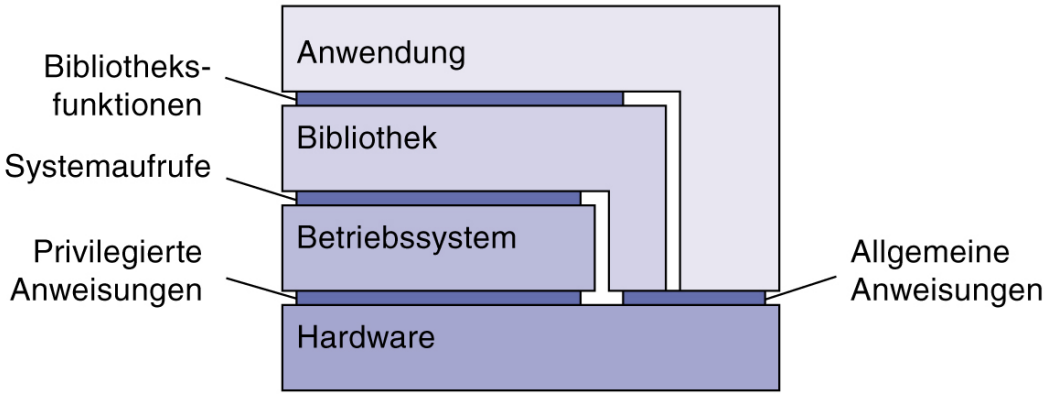
(a) Allgemeine Gliederung von Programm, Schnittstelle und System;
(b) allgemeine Gliederung eines virtuellen Systems A oberhalb von B

Virtualisierung

- Gründe für Virtualisierung
  - Relativ schnelle Änderungen bei der Hardware und Systemsoftware der niedrigeren Ebene
  - Höhere Abstraktionsebenen (z.B. Middleware und Anwendungen) viel stabiler
  - Reduzierung der Unterschiedlichkeit der Plattformen und Computer
    - Arbeitserleichterung für Systemadmins
  - Bietet hohen Grad an Portabilität

Architekturen virtueller Maschinen

- Verschiedene Arten von Virtualisierung möglich
- Allgemeine Schnittstellen
  - Zwischen Hardware und Software mit Maschinenbefehlen für alle Programme
  - Zwischen Hardware und Software mit Maschinenbefehlen für privilegierte Programme (z.B. für Betriebssysteme)
  - Systemaufrufe, wie sie von einem Betriebssystem aufgerufen werden
  - Schnittstelle aus Bibliotheksaufrufen
    - API (Application Programming Interface)
    - Verbergung von Systemaufrufen

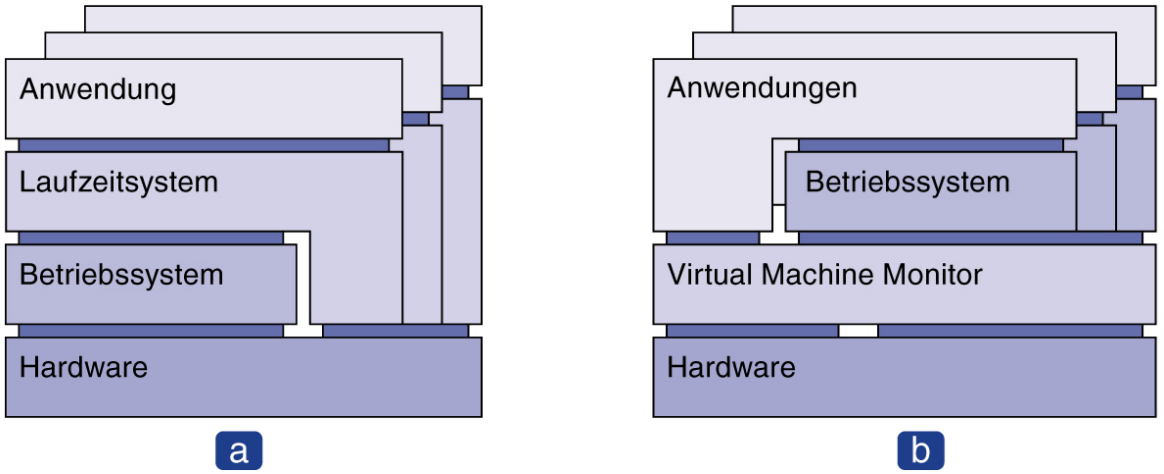


Architekturen virtueller Maschinen

- Art und Weisen von Virtualisierungen
  - Erstellung eines Laufzeitsystem
  - Bereitstellung eines Systems in Form einer Schicht
- Laufzeitumgebung**: Virtuelle Prozessmaschine**
  - Bereitstellung eines abstrakten Befehlssatzes zum Ausführen von Anwendungen
  - Virtualisierung nur für einen Prozess
  - Befehle können **interpretiert** werden
    - siehe Java-Laufzeitumgebung
  - Befehle können **emuliert** werden
    - z.B. beim Ausführen von Windows-Anwendungen auf UNIX-Plattformen

Architekturen virtueller Maschinen

- Virtualisierung in Form einer Schicht
  - Virtual Machine Monitor
  - Vollständige Abschirmung der ursprünglichen Hardware
  - Aber Bereitstellung des vollständigen Befehlssatzes der Hardware
  - Schnittstelle kann mehreren Programmen gleichzeitig angeboten werden
  - Gleichzeitigkeit mehrerer Betriebssysteme mgl.
    - VMware oder Xen oder ...



Containervirtualisierung

- Problem: Virtualisierung mittels Hypervisor ist ressourcenaufwändig
- Ziel: Isolation von Prozessen bzw. Prozessgruppen voneinander
- Prozesse im Container sehen
  - Teil des Dateisystems
  - Prozesse im gleichen Container
  - Ggf. eigene Netzwerkadresse
- Vorteile:
  - kein Overhead durch extra Betriebssystem
  - Erstellungen von Images → aus Image kann Container gestartet werden
  - Leichtes Verschieben von Images → keine extra Konfigurationen

Empfehlung: https://www.youtube.com/watch?v=ukYEk2S6akI

Themenüberblick

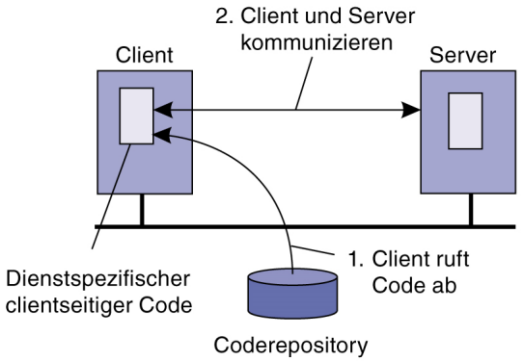
- Architekturen
- **Prozesse und Threads**
  - Threads in verteilten Systemen
  - Virtualisierung
  - **Codemigration / Prozessmigration**

Codemigration / Prozessmigration Fragen

- Was verstehen Sie unter Codemigration / Prozessmigration?
- Was ist der Unterschied zwischen Code- und Prozessmigration?
- Welche Gründe kann es für Codemigration / Prozessmigration geben?
- Welche Beispiele für Codemigration kennen Sie?
- Welche Schwierigkeiten gibt es bei der Prozessmigration?
- Wo gibt es Anwendungen, bei denen Prozessmigration eingesetzt wird?

Codemigration / Prozessmigration

- Gründe für Prozessmigration
  - Leistungssteigerung (Optimierung der Rechenzeit)
- Gründe für Codemigration
  - Minimierung der Kommunikation
  - Leistungsoptimierung aufgrund qualitativer Überlegungen
  - Verlagerung von Datenverarbeitung
    - Formularauswertung beim Client
    - Datenverarbeitung beim Server
    - …
  - Leistungsverbesserung durch Parallelverarbeitung
  - Flexibilität
    - z.B. für dynamische Konfigurationen
    - Client erhält Code erst, wenn Client an den Server gebunden wird



Codemigration / Prozessmigration

- Unterscheidungsaspekte zur Code- und Prozessmigration
  - „schwache“ und „starke“ Mobilität
  - Initiierung durch Sender oder Empfänger

Codemigration / Prozessmigration

- Erinnerung an das Prozessmodel
  - Codesegment
  - Ressourcensegment
  - Ausführungssegment
- „schwache Mobilität“
  - Übertragung des Codesegmentes und ggf. Initialisierungsdaten
  - Code muss portierbar sein
  - Vorteil: Einfachheit
  - Bespiele
    - Java Applets, Javascript, ActionScript (FlashPlayer), ...

Codemigration / Prozessmigration

- „starke Mobilität“
  - Übertragung des Codesegmentes und des Ausführungssegmentes
  - Idee
    - Anhalten eines laufender Prozesses
    - Verschieben auf einen anderen Computer
    - Fortsetzen des Prozesses
  - Vorteil: universell einsetzbar
  - Nachteil: schwieriger zu implementieren

Codemigration / Prozessmigration

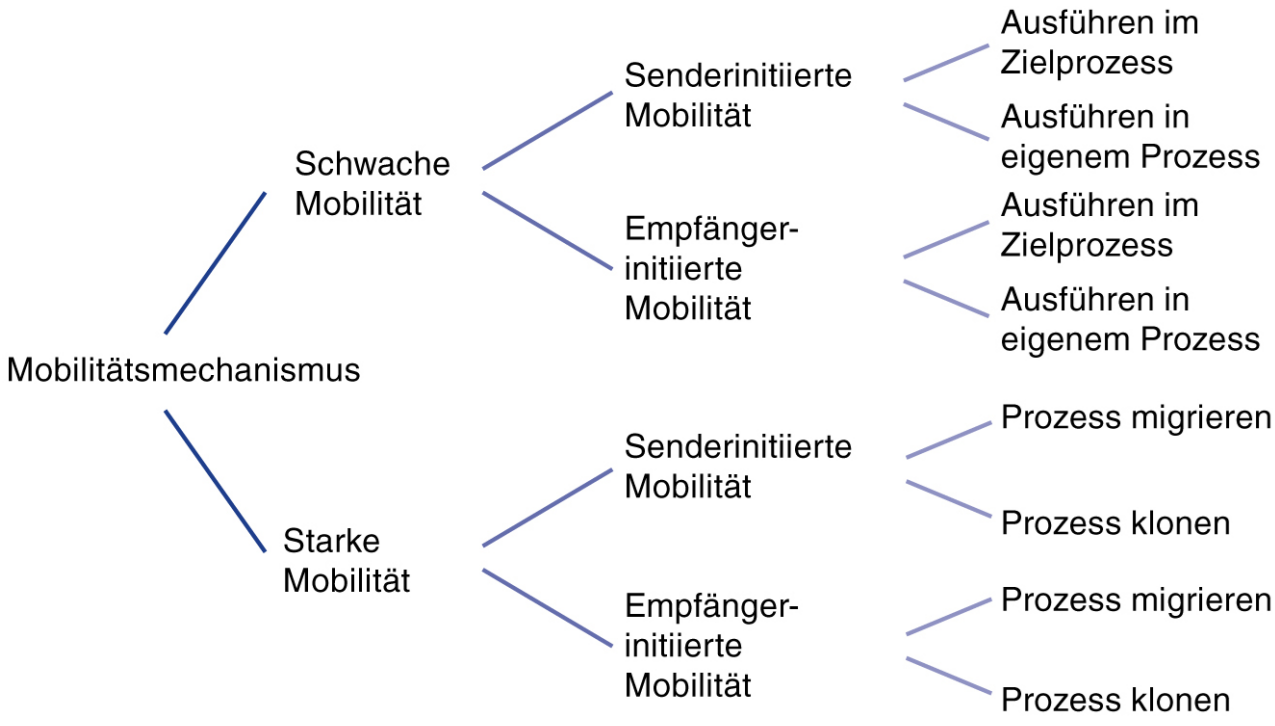
- Senderinitiierte Migration
  - Auslösung durch Computer, auf dem sich Code befindet oder ausgeführt wird
  - z.B. Hochladen von Programmen zu einem Server
  - Problem des Vertrauens / Sicherheit
    - Client sollte registriert und authentifiziert sein
    - Mgl. Ausgangspunkt für Hackerangriffe (z.B. NoPetya)

Codemigration / Prozessmigration

- Empfängerinitiierte Migration
  - Initiative durch Zielcomputer
  - Meist anonym ausgeführt
  - Codemigration meist nur zur Leistungsverbesserung
  - Begrenzte Anzahl von Ressourcen muss geschützt werden (z.B. Speicher und Netzwerkverbindungen)
  - Ausführungsmöglichkeiten
    - z.B. im Kontext des Webbrowsers
    - Erstellung eines eigenen Prozesses

Codemigration / Prozessmigration

- Spielart der Prozessmigration (der starken Mobilität) – Entferntes Klonen
  - Klonen erstellt genau Kopie
  - Möglichkeit zur Erhöhung der Verteilungstransparenz
- Möglichkeiten der
- Codemigration



Migration und Ressourcenumgang

- Problem: Wie geht man mit dem Ressourcensegement um?
- Unterscheidungen
  - Prozess-Ressourcen-Bindung
  - Ressourcen-Computer-Bindung
- Prozess-Ressourcen-Bindung durch
  - Bezeichner
    - Stärkste Bindung
    - z.B. Referenz auf lokale Kommunikationsendpunkte oder auf eine URL
  - Wert
    - z.B. Referenz auf Standardbibliothek
  - Typ
    - z.B. Referenz auf lokale Geräte (Monitor, Drucker, ...)

Migration und Ressourcenumgang

- Ressourcen-Computer-Bindung
  - Ungebundene (lose) Ressourcen
    - Einfaches Verschieben zw. Computern möglich
    - z.B. (Daten-)Dateien, die nur mit Prozess verknüpft sind
  - Gebundene Ressourcen
    - Theoretisch unabhängige, aber Verschieben in andere Umgebung meist nicht möglich
    - z.B. Lokale Datenbanken
  - Fixe Ressourcen
    - Eng an Computer oder Umgebung gebunden und nicht verschiebbar
    - z.B. lokale Geräte, lokaler Kommunikationsendpunkt

Migration und Ressourcenumgang

- Mögliche Umgangsformen je nach Bindung:

Prozess-Ressourcen-Bindung	Ressourcen-Computer-Bindung: Lose	Befestigt	Fix
Durch Bezeichner	V (oder GR)	GR (oder V)	GR
Durch Wert	WK (oder V, GR)	GR (oder WK)	GR
Durch Typ	NB (oder V, WK)	NB (oder GR, WK)	NB (oder GR)

GR	Einrichten einer globalen, systemweiten Referenz
V	Verschieben der Ressource
WK	Kopieren des Wertes der Ressource
NB	Neubinden des Prozesses an eine lokal verfügbare Ressource

Migration in heterogenen Systemen

- Achtung: Codemigration „leicht“ bei homogenen Systemen
- Lösung basiert auf Nutzung virtueller (Prozess-)Maschinen
  - Nutzung von Skriptsprachen oder hochgradig portierbare Sprachen
  - Umgang:
    - Quellcode wird interpretiert (Skriptsprachen)
    - Ausführung eines vom Compiler generierten Zwischencodes (Java)
- Migration virtueller Maschinen
  - Idee: Eines in Betrieb befindlichen Systems inkl. Betriebssystem wird zwischen Computern verschoben
  - Beispiel: VMware vSphere