Vorlesung
Component Ware und Web-Services

2. Middleware und Kommunikation

Dr. Hans-Gert Gräbe, F. Schumacher
Wintersemester 2003/2004

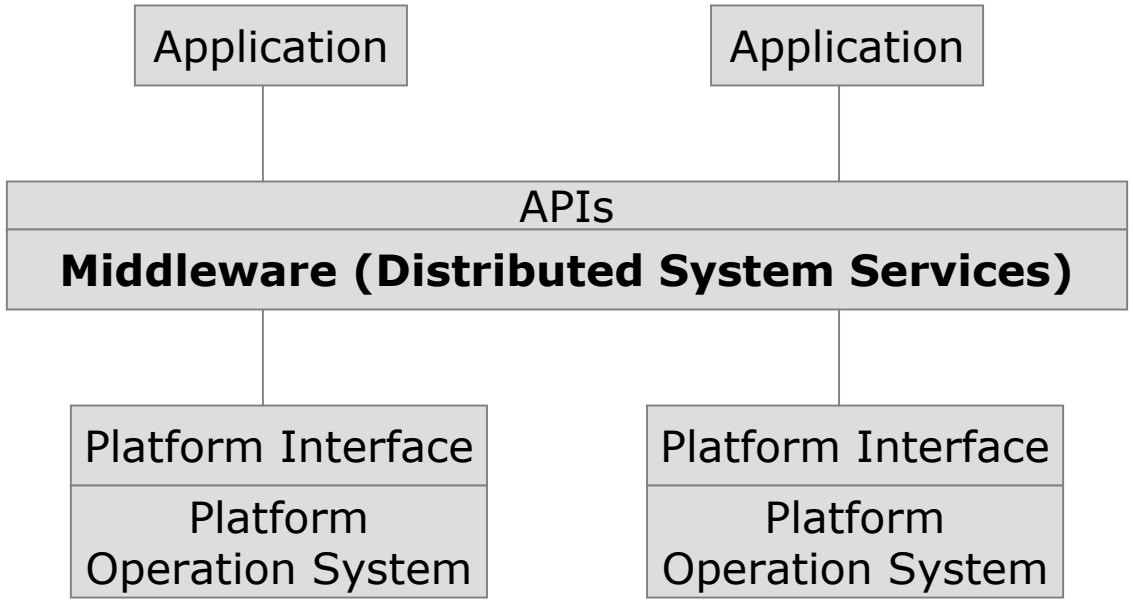
Inhalt

- **Begrifflichkeiten**
  - Middleware
  - Sicherheitsmodell
  - Transaktionen
  - Ortstransparenz
  - Ereignisse
  - **Kommunikation**
    - Blockierung
    - Verbindungsbegriff
    - Synchron
    - Asynchron
- **Middleware**
  - Transaction Processing Monitor (MTS, EJB)
  - Remote Procedure Calls (JavaRMI)
  - Message-Oriented Middleware (MQ Series, MS BizTalk)
  - Object Request Broker (CORBA, DCOM)
  - Database Middleware (ODBC-Treiber)

Einleitung

Was ist Middleware?

- Verbindungssoftware
- erlaubt Interaktion zwischen Prozessen
- Kommunikation von Rechnern in einem Netzwerk
- Client/Server Applikationen
- Kommunikation zwischen heterogenen Plattformen



Einleitung

Middleware stellt eine Reihe funktionaler APIs zur Verfügung

- Vorteile für Anwendungen durch Nutzung dieser APIs
  - Transparente Allokation über ein Netzwerk
  - Interaktion mit anderen Anwendungen/Services
    - Naming Service bei ORB
    - Message Queues bei MOM
    - Zugriff auf Logging- und Verwaltungsfunktionen
  - Unabhängigkeit von Netzwerkservices
  - Zuverlässigkeit/Verfügbarkeit
  - kapazitive Skalierbarkeit ohne Funktionalitätsverlust

Sicherheitsmodell

Sicherheit:

- Definition von Rollen (Benutzergruppen)
- Ausführungsrechte werden den Rollen zugeordnet
- Zuordnung von Benutzern zu Rollen vom Systemadministrator

Vorgehensweise:

- Identifizierung des initiierenden Benutzers bei Aufruf einer Operation durch Komponentenplattform
- Überprüfung der Berechtigung der dem Benutzer zugeordneten Gruppen

Transaktionen

Transaktion – Folge von Aktionen zur kontrollierten Veränderung eines Datenbestandes:

ACID:

- atomic (unteilbar)
  - es werden entweder alle oder keine Aktionen ausgeführt
  - Beendigung nur nach kompletter Abarbeitung der Transaktion
- consistent
  - konsistenter Datenbestand vor und nach der Transaktion
  - inkonsistent nur während der Transaktion
- isolated
  - Ergebnis unabhängig von der Reihenfolge der Transaktionen bei Gleichzeitigkeit
  - inkonsistente Zustände für andere Transaktionen unsichtbar
- durable
  - Änderungen werden persistent gespeichert

Verteilte Transaktionen

- Bei Änderung der Daten auf mehreren Rechnern bzw. Ablauf der Prozesse auf mehreren Rechnern
- Wird von einem System initiiert (master)
- Teilaufgaben werden an andere Systeme übergeben (slave)
- Koordination über ein Zwei-Phasen-Commit-Protokoll
  - Alle slaves müssen ihre Aktionen erfolgreich abgeschlossen haben
  - Bei Erfolg: Commit
  - Bei nur einem Misserfolg: Rollback

Orttransparenz

Orttransparenz:

- Client muss physischen Standort des Servers nicht kennen
- Unterscheidung von entfernten und lokalen Objekten

Namensdienst:

- Stellt Verbindung zu entferntem Objekt her
- Liefert eine Referenz auf angefordertes Objekt
- Erstellt eventuell Objekt

Vorteile der Orttransparenz:

- Keine Unterscheidung von entfernten und lokalen Objekten bei der Implementierung
- Server benötigt keine Informationen über Standort des Clients
- Komponente kann zwischen Servern verschoben werden
  - Bei Serverausfall automatisches Verschieben auf anderen Server
  - Automatische Lastenverteilung

Ereignisse

Ereignisse:

- Client möchte über bestimmte Ereignisse informiert werden
- Übernimmt eine passive Rolle
- Ereignisse werden nach Ereignistypen unterschieden
- Client kann beim Server einen bestimmten Ereignistypen abhören

Ausgehende Schnittstellen:

- Ereignisquelle spezifiziert Schnittstelle
- Ereignisempfänger implementiert Schnittstelle
- Ereignisquelle ruft beim Ereignisempfänger Operationen der Schnittstelle auf
- Jeder Ereignistyp hat separate Schnittstelle

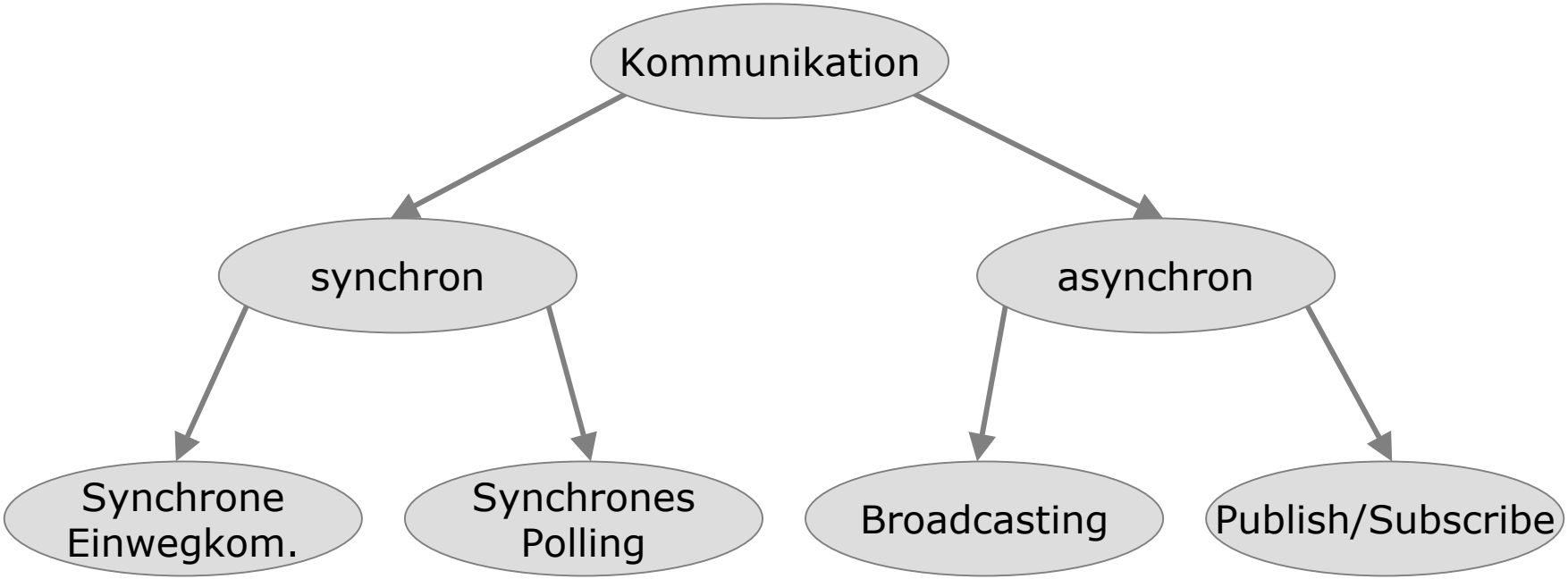
Ereignisse

Ereignis-Objekte:

- Repräsentieren Ereignis
- Ereignisobjekt wird an alle angemeldeten Empfänger verschickt
- Für jeden Ereignistyp existiert eine Klasse
  - ➢ Objekte können ereignisspezifische Daten enthalten

- Kombination beider Modelle (aktiv + passiv) möglich
- Komponentenplattform stellt technische Infrastruktur zur Ereignisübermittlung auch über verteilte Systeme bereit

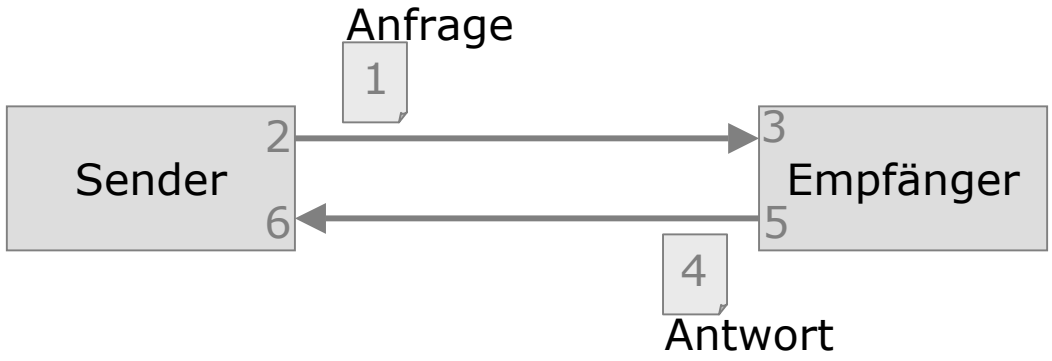
Kommunikationsmodelle



Definition:
Bei **synchroner Kommunikation** sind Sender und Empfänger in ihrem Ablauf aneinander gekoppelt.
Bei **asynchroner Kommunikation** sind sie das nicht.

W. Keller: Enterprise Application Integration

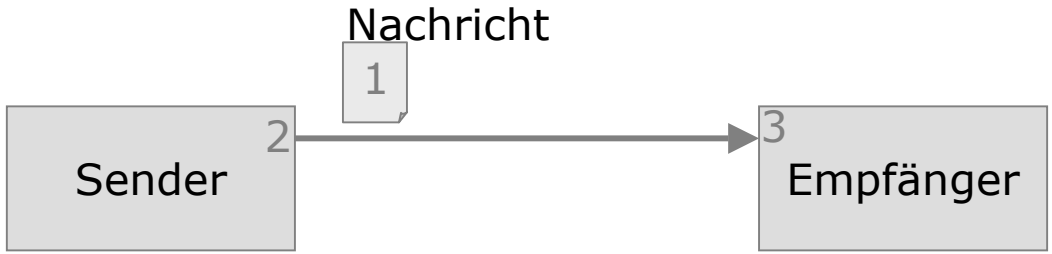
Synchrone Kommunikation – Request/Reply



1. Sender sendet Anfrage an Empfänger
2. Sender blockiert und wartet auf Antwort
3. Empfänger empfängt Nachricht und verarbeitet sie
4. Empfänger schickt eine Antwort ...
5. ... und kann weiterarbeiten
6. Sender empfängt Antwort und kann ebenso weiterarbeiten

- Sender ist bis zum Eintreffen der Antwort blockiert
- Empfänger muss lebendig sein
- Prinzip von Remote Procedure Calls

Asynchrone Kommunikation – Message Passing

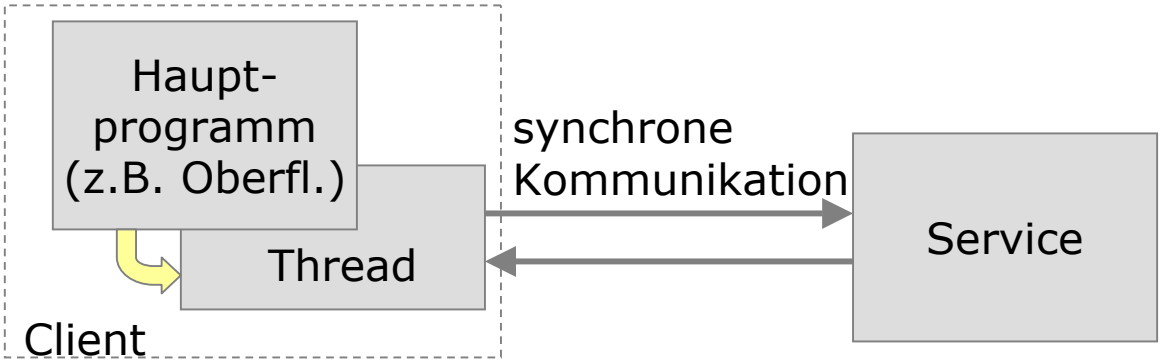


1. Sender sendet Nachricht an Empfänger
2. Sender kann sofort weiterarbeiten und wird nicht blockiert
3. Empfänger empfängt Nachricht und verarbeitet sie

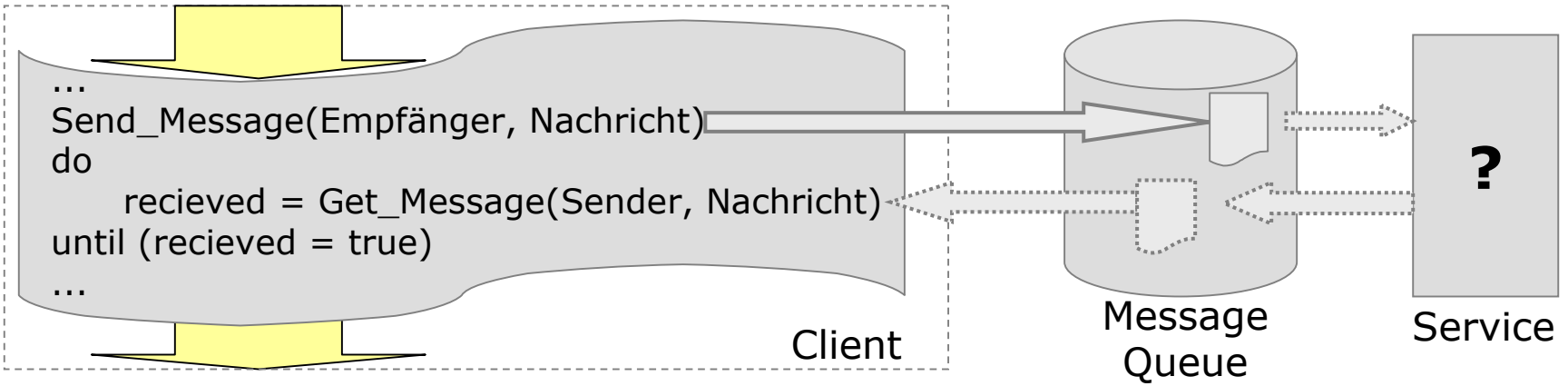
- keine Blockierung des Senders
- Empfänger muss nicht lebendig sein
- Prinzip von Message-Oriented Middleware

Blockierung

- Synchrone Kommunikation **ohne** Blockierung
  - Auslagerung des Aufrufes in einen Thread



- Asynchrone Kommunikation **mit** Blockierung
  - Erzwungenes Warten des Clients auf eine Antwort



Begriff der Verbindung

- Verbindungsorientierte Kommunikation
  - Sender muss Empfänger kennen oder suchen
  - Anbinden an eine Instanz des Empfängers (bind)
  - Bis zum Abschluss der Kommunikation teilen Sender und Empfänger einen gemeinsamen Zustand

  ➡ **synchron**

- Verbindungslose Kommunikation
  - Sender muss Empfänger weder kennen noch sich an ihn binden
  - Sender weiß nicht, ob Nachricht empfangen oder verarbeitet wurde

  ➡ **asynchron**

Begriff der Verbindung

Verbindungsorientierte Kommunikation ist zu empfehlen wenn ...

- Kommunikationspartner mit hoher Wahrscheinlichkeit verfügbar
- Ergebnis/Quittung für die Weiterarbeit des Senders essenziell notwendig

Verbindungslose Kommunikation ist zu empfehlen wenn ...

- Verfügbarkeit des Kommunikationspartners nicht sichergestellt ist
- kein sofortiges Ergebnis/Quittung erforderlich

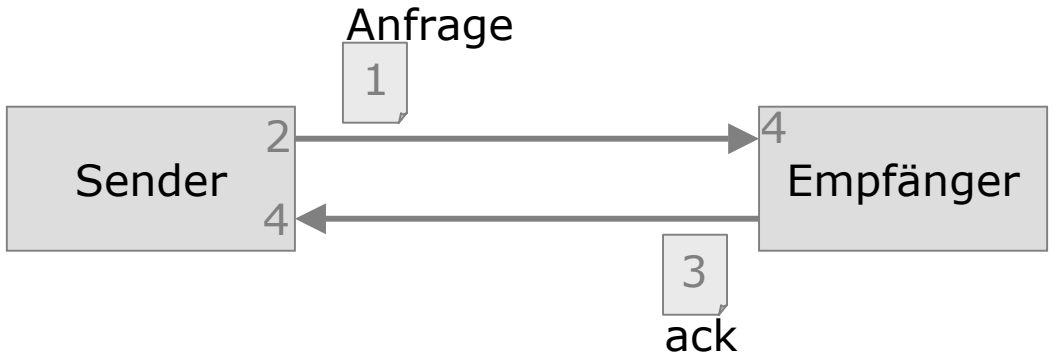
Fehlerbehandlung

- sowohl für verbindungslose als auch verbindungsorientierte Kommunikation
- sollte in Basissoftware enthalten sein
- muss möglichen Ausfall **beider** Kommunikationspartner berücksichtigen

Varianten synchroner Kommunikation

Synchrone Einwegkommunikation

- Prozeduraufruf ohne Rückgabeparameter
- Empfänger bestätigt lediglich den Eingang

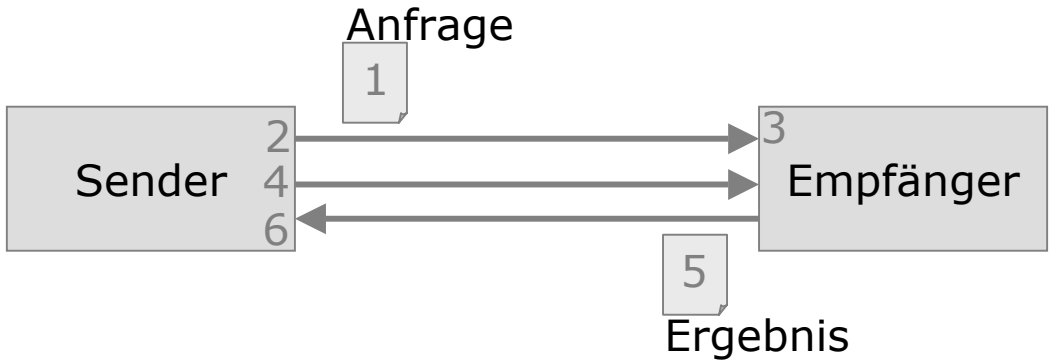


1. Sender sendet Anfrage an Empfänger
2. Sender blockiert und wartet auf Antwort
3. Bevor der Empfänger die Anfrage bearbeitet, sendet er eine Empfangsbestätigung zurück (ack=Acknowledgement)
4. Sender und Empfänger können sofort weiterarbeiten, Empfänger erledigt Anfrage

Varianten synchroner Kommunikation

Synchrones Polling

- Sender fragt periodisch nach einer Antwort
- Nachfrage über speziellen Aufruf, gemeinsamen Speicherbereich oder Datei

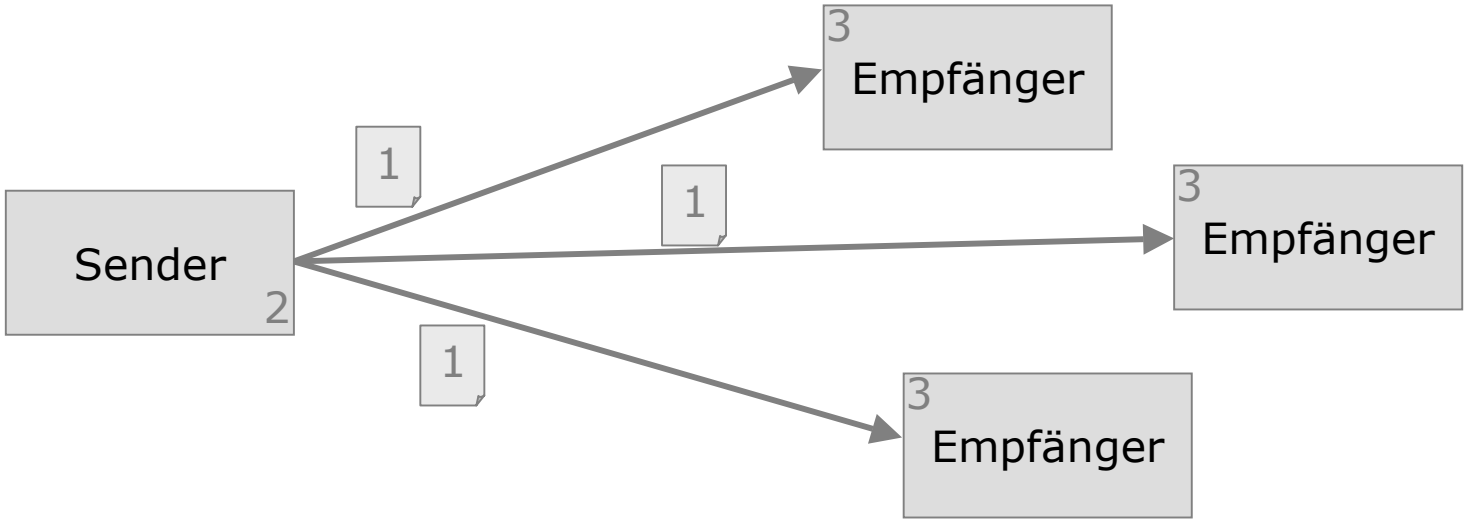


1. Sender sendet Anfrage an Empfänger
2. Sender wird nicht blockiert, sondern tut zunächst etwas anderes
3. Empfänger bearbeitet die Anfrage
4. Sender fragt periodisch nach einem Ergebnis
5. Ergebnis wird gesendet
6. Sender arbeitet mit dem Ergebnis weiter

Varianten asynchroner Kommunikation

Broadcasting

- Sender schickt Nachrichten über ein allgemein abhörbares Medium
- Low-Level-Protokolle (Netzwerk)

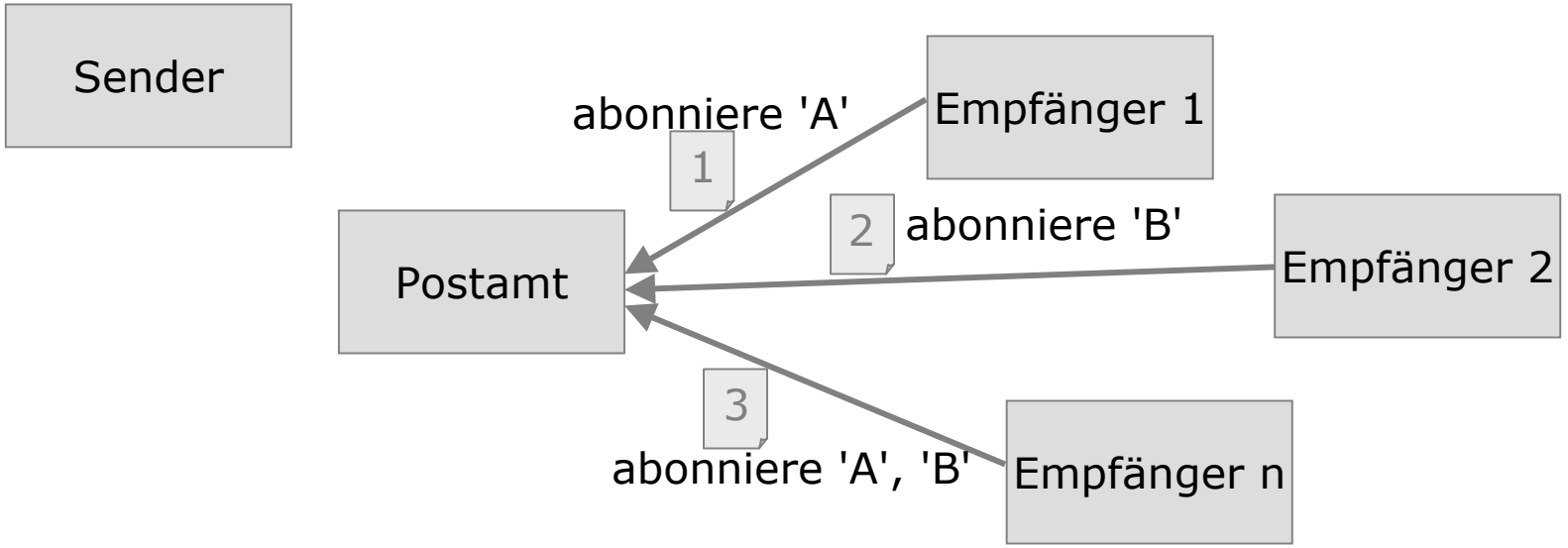


1. Sender schickt eine Nachricht ab, ohne einen Empfänger zu spezifizieren
2. Sender wird nicht blockiert und arbeitet sofort weiter
3. Empfänger **kann** auf die Nachricht reagieren

Varianten asynchroner Kommunikation

Publish/Subscribe (Schritt 1)

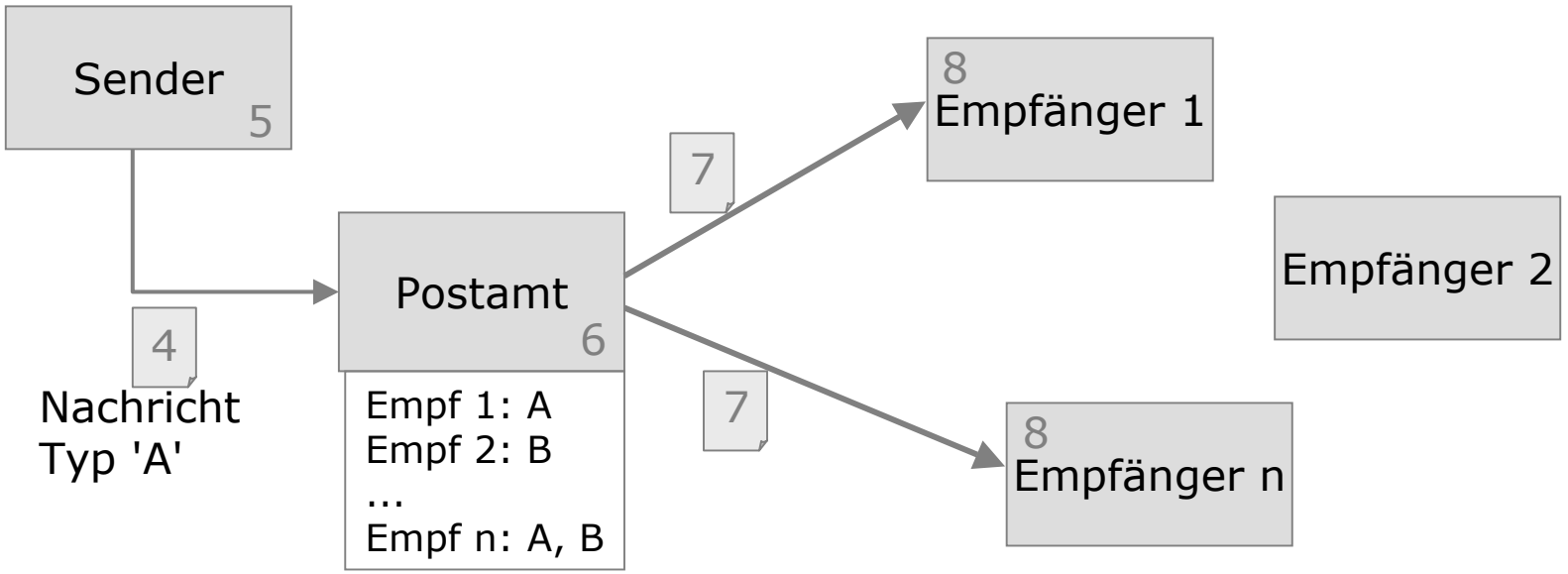
- Nachrichten werden nur solchen Empfängern zugestellt, die sich vorher für diese Nachricht eingetragen haben



1. Empfänger 1 registriert sich für Nachrichtentyp 'A'
2. Empfänger 2 registriert sich für Nachrichtentyp 'B'
3. Empfänger n registriert sich für Nachrichtentyp 'A' und 'B'

Varianten asynchroner Kommunikation

Publish/Subscribe (Schritt 2)



4. Sender schickt Nachricht vom Typ 'A' an das 'Postamt'
5. Sender kann sofort weiterarbeiten
6. 'Postamt' ermittelt alle Empfänger, die sich für Nachrichtentyp 'A' registriert haben
7. 'Postamt' sendet die Nachricht an die ermittelten Empfänger
8. Empfänger verarbeiten die Nachricht

Arten von Middleware

- **Transaction processing (TP) monitors**
  - Optimierung und Steuerung von Clientzugriffen auf knappe Server-Ressourcen
- **Remote Procedure Calls (RPCs)**
  - simpler Aufruf von Programmlogik auf entfernten Systemen
- **Message-Oriented Middleware (MOM)**
  - asynchroner Datenaustausch zwischen verteilten Anwendungen
- **Object Request Brokers (ORB)**
  - Verteilen von Objekten in einem heterogenen Netzwerk
- **Database Middleware**
  - Zugriff auf entfernte/verteilte Datenbanken

Transaction processing monitors

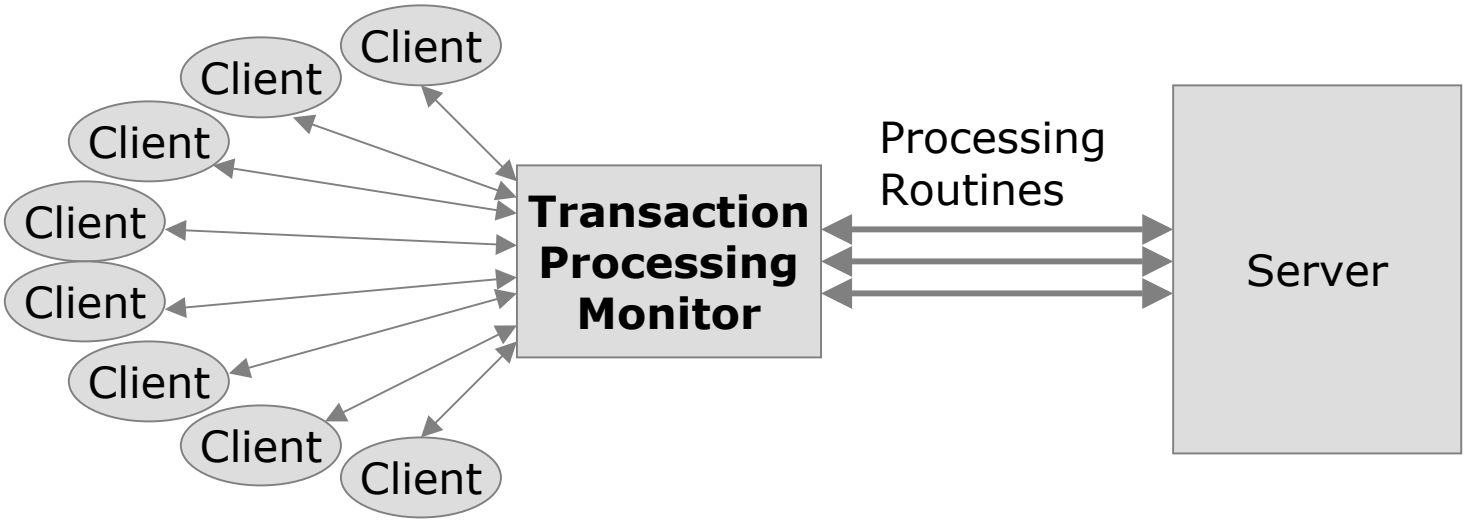
Aufgaben von Transaktionsmonitoren

- Kontrolle transaktionsorientierter Applikationen
- Ausführung von Geschäftslogik
- Datenbankupdates
- kostengünstige Alternative zum Upgrade von Datenbankmanagementsystemen

Anwendungsgebiete

- Datenmanagement
- Netzwerkzugriff
- Sicherheitssysteme
- Bearbeitung von Auslieferungsaufträgen
- Flugreservierungen
- Kundenservice

Transaction processing monitors



- Clients werden an zustandslose Server gebunden
  - minimaler Overhead
  - Mapping der Client-Anfragen auf Applikationsdienste
  - keine Transaktionslogik auf dem Client
- Datenbank kommuniziert mit Satz von Verarbeitungsroutinen
- Unabhängigkeit von der Datenbankarchitektur

Transaction processing monitors

Transaktionsmonitore umfassen ...

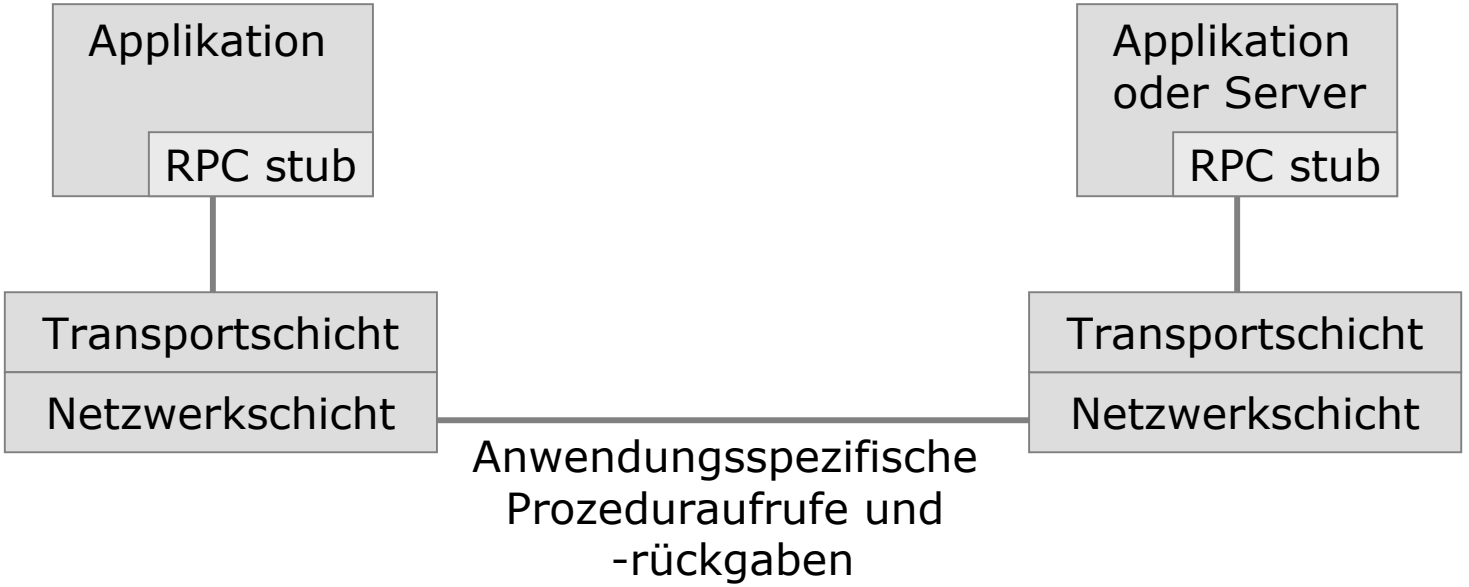
- Zahlreiche Managementfunktionen
- Wiederholung fehlgeschlagener Prozesse
- Dynamisches Loadbalancing
- Erzwingung der Konsistenz verteilter Daten
- Leichte Skalierung auf mehrere physikalische Server
- APIs unterstützen Komponenten wie ...
  - heterogene Clientbibliotheken
  - Datenbank- und Ressourcenmanager
- Unterstützung zahlreicher Modelle der Programm-zu-Programm-Kommunikation
  - store-and-forward
  - Asynchrone und synchrone Kommunikation
  - Remote Procedure Calls

Transaction processing monitors

- verbesserte Erreichbarkeit
- Systemressourcen-Overhead wird minimiert
- Architektur fördert Verwendung modularer, wiederverwendbarer Funktionen
- Erfolgreicher Einsatz dieser Technologie seit ca. 30 Jahren
  - Reservierungsdienste
  - Elektronischer Geldtransfer
  - Wertpapierhandel
  - Ressourcenplanung in der verarbeitenden Industrie

Remote Procedure Calls

- Client/Server-Architektur
- Applikation kann über mehrere heterogene Plattformen verteilt sein
- Abstraktion von den Einzelheiten verschiedener Betriebssystem- und Netzwerkschnittstellen
- Konzept seit 1976 – erste Implementierungen Ende 70er, Anfang 80er Jahre



Remote Procedure Calls

- RPCs sind im Client eingebettet
  - keine eigenständige Middlewareschicht
- Compiler erzeugt jeweils einen Stub für Client- und Serverseite
- in der Regel synchrone Aufrufe zwischen Client und Server
  - Request-Reply (call/wait) Protokolle
    - Client wird blockiert, bis der Server die Anforderung verarbeitet hat
  - Asynchrone Protokolle (call/nowait) sind seltener
- Semantik der entfernten Aufrufe gleichbleibend
  - Reduzierung der Komplexität der Entwicklung verteilter Anwendungen
- Entwickler wird in Komplexität der Master-Slave Natur der Client/Server-Systeme verwickelt

Remote Procedure Calls

- Erhöhung der Flexibilität einer Architektur durch die Möglichkeit, einen Server auf einer entfernten Maschine anzusprechen
- Zugriff ohne Kenntnis detaillierter Systeminformationen
- Geeignet für Anwendungen, wo Client eine Anforderung an den Server stellt und auf die Antwort wartet
- Ständige Verfügbarkeit des Servers muss gewährleistet sein
- Mechanismen zur Verarbeitung einer "Geblockt-Situation"
  - Fehlermeldungen
  - Anfragezeitmesser
  - Wiederholte Übertragung
  - Weiterleitung an alternativen Server
- Komplexität eines RPC-Systems ist abhängig von der Ausgereiftheit des Recovery-Mechanismus

Remote Procedure Calls

Netzwerkperformanz

- Blockierungsmechanismen schützen vor Netzwerküberlastung
- Durch Recovery-Mechanismen (z.B. wiederholte Übertragung) kann Netzlast ansteigen
  - ungeeignet für bereits belastete Netzwerke
- RPCs verwenden statische Routingtabellen, die während der Kompilierung erstellt werden
  - Loadbalancing schwierig

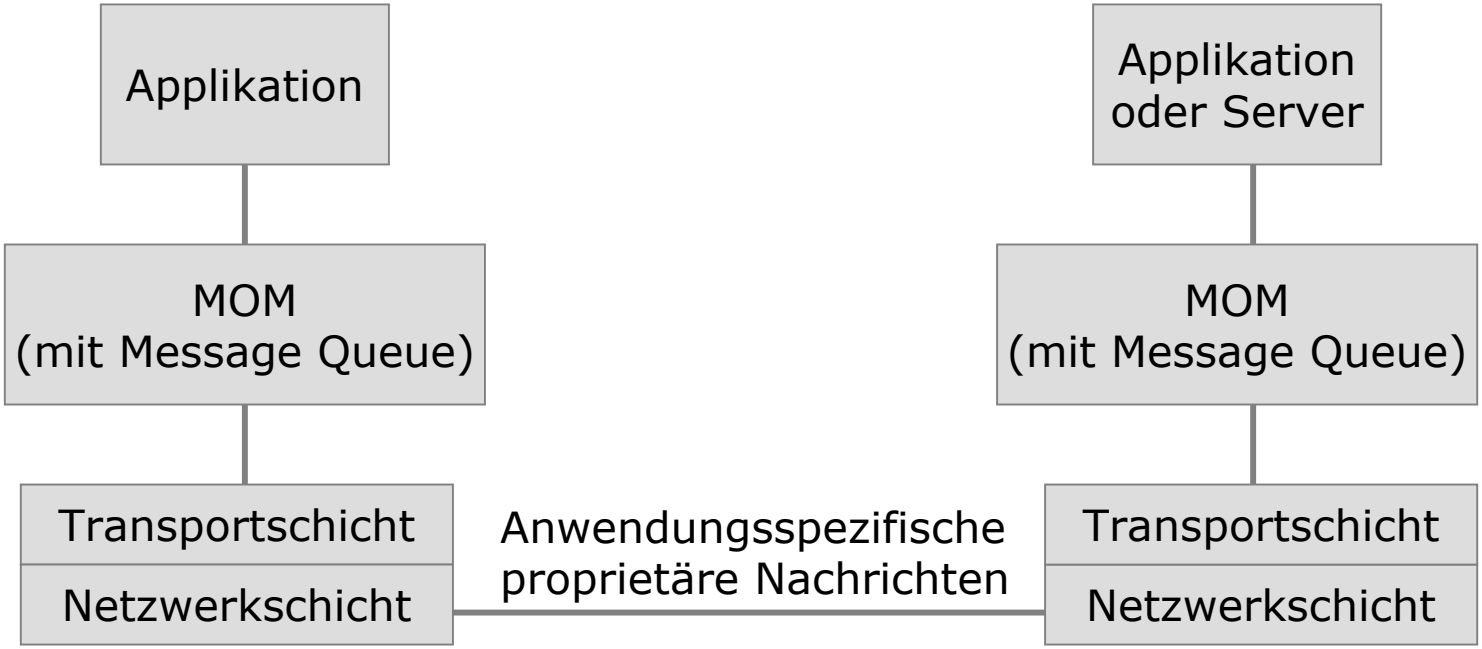
Remote Procedure Calls

- RPC-Implementierungen sind in der Regel proprietär und untereinander inkompatibel

- Wichtige Gesichtspunkte für die Einschätzung eines RPC-Systems
  - Unterstützung von synchronen/asynchronen Aufrufen
  - Unterstützung verschiedener Netzwerkprotokolle
  - Abhängigkeit vom Betriebssystem
  - Qualität der Recovery-Mechanismen

Message-Oriented Middleware

- Asynchrone Aufrufe zwischen Client- und Serveranwendungen
- Temporärer Speicher in Form von Message queues
- Nachrichtenaustausch mit anderen Programmen ohne Kenntnis über deren Plattform oder Rechner
- erste Implementationen Mitte bis Ende der 80er



Message-Oriented Middleware

- Nachrichten enthalten formatierte Daten, Aktionsaufrufe oder beides
- typische Nachrichtenweiterleitung asynchron und peer-to-peer
  - meistens ebenfalls Unterstützung für synchrone Nachrichtenweiterleitung
- Geeignet für ereignisgesteuerte Anwendungen
  - MOM für Benachrichtigung des Servers über eingetretenes Ereignis verantwortlich
- Geeignet für objektorientierte Systeme
  - MOM liefert konzeptionellen Mechanismus für peer-to-peer Kommunikation zwischen Objekten
- MOM isoliert Entwickler von Problemen der Konnektivität

Message-Oriented Middleware

Netzwerkperformanz

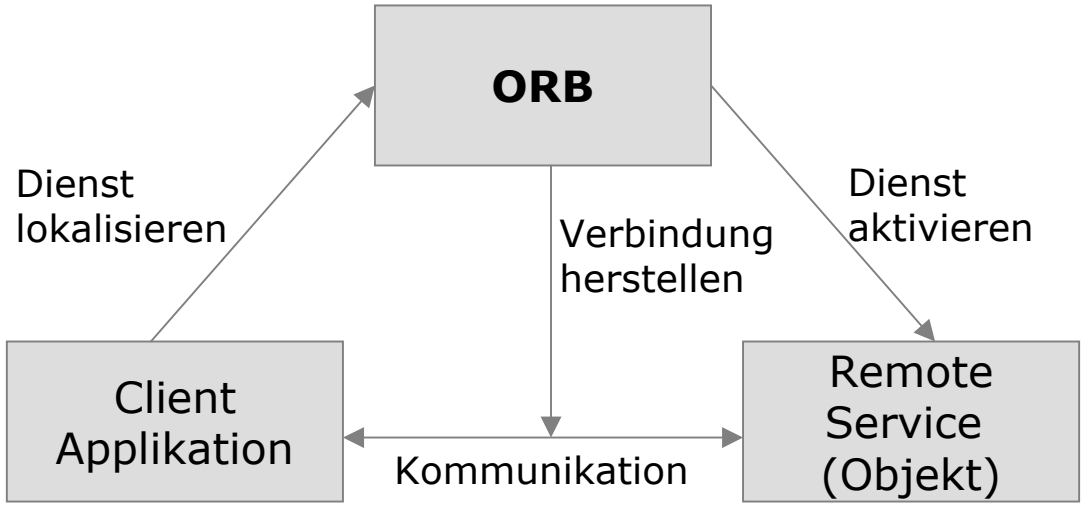
- Asynchrone Mechanismen schützen nicht vor Überlastung des Netzwerkes
  - Client kann kontinuierlich Daten an einen überlasteten Server schicken
- Möglichkeit zur standardmäßigen synchronen Übertragung
  - nur bei Nichterreichbarkeit des Servers asynchrone Mechanismen

Message-Oriented Middleware

- MOM Software (kernel) muss auf jedem Rechner des Netzwerkes laufen
  - Implementierung muss benötigte Plattformen und Protokolle unterstützen
  - Lokale Ressourcen müssen auf jedem Rechner für den MOM-Kernel aufgebracht werden
  - Administrativer Aufwand steigt bei großer Anzahl von Systemen, vor allem in einer heterogenen Umgebung
  - Höhere Kosten bei der Benötigung mehrerer Kernels (für unterschiedliche Betriebssysteme)

Object Request Broker

- Verwaltet Kommunikation und Datenaustausch zwischen Objekten
- Ermöglicht den Aufbau von Systemen durch Kombination von Objekten unterschiedlicher Hersteller
- Für Entwickler eines verteilten Systems sind nur die Schnittstellen des ORBs für Interesse



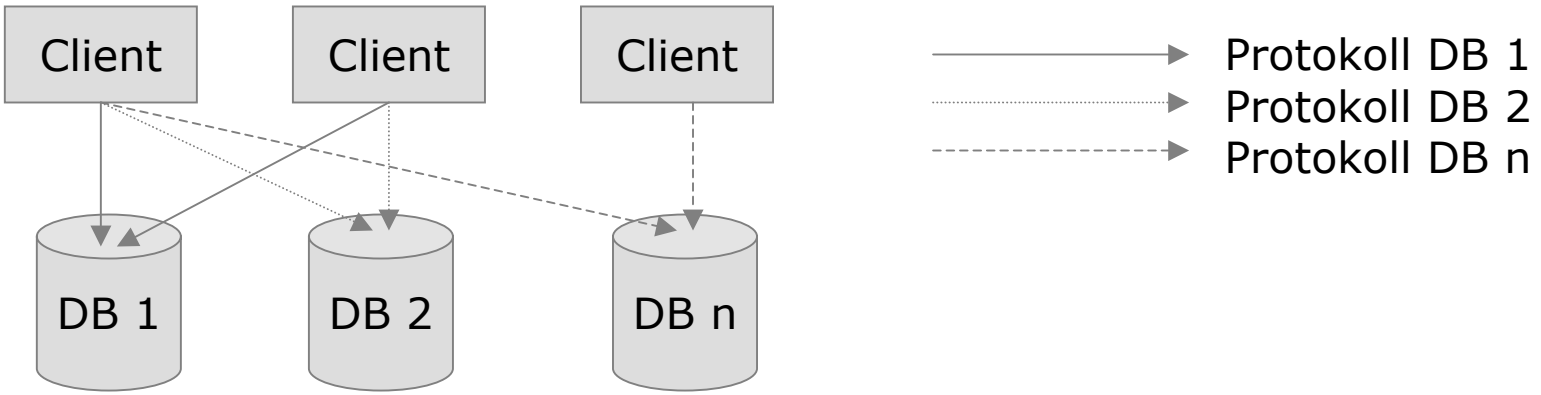
Object Request Broker

- Funktionalitäten zur Unterstützung der Objektkommunikation über Maschinen-, Software und Herstellergrenzen
  - Schnittstellendefinition
  - Auffinden und Aktivieren von entfernten Objekten
  - Kommunikation zwischen Client und Server
- ORB stellt Verzeichnis von Diensten und Erstellung der Verbindung zwischen Clients und diesen Diensten zur Verfügung
- Illusion der "Lokalität"
- Objekte verstecken Implementierungsdetails wie Programmiersprache, Betriebssystem, Hardware und Standort
- Unterschiedliche Implementierungsmöglichkeiten für ORBs
  - Funktionen in Client kompiliert
  - Eigenständige Prozesse
  - Teil des Betriebssystems

Database Middleware

Remote-SQL, ODBC, JDBC

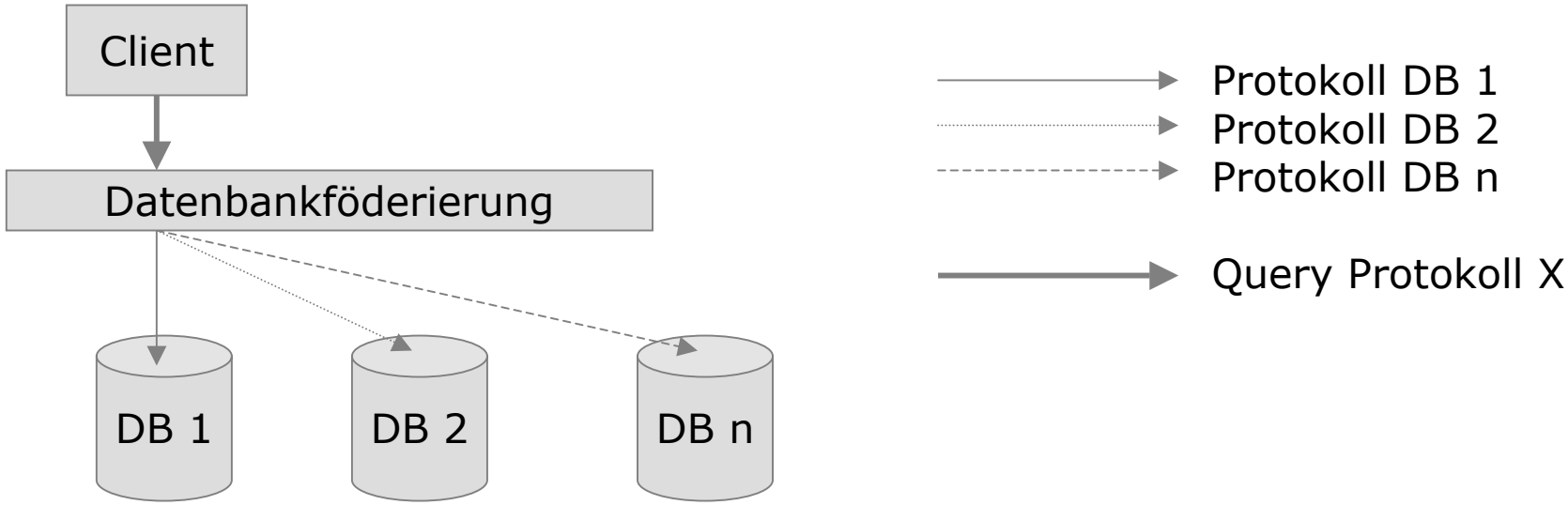
- Anwendung bei "Fat-Clients"
- Kommunikationsprotokoll zwischen Anwendungskern und Datenbankserver
- Client schickt SQL-Kommando an Datenbank und erhält Ergebnisdatenstrom in einer, zur Programmiersprache passenden Datenstruktur zurück



Database Middleware

Föderierte Datenbanken

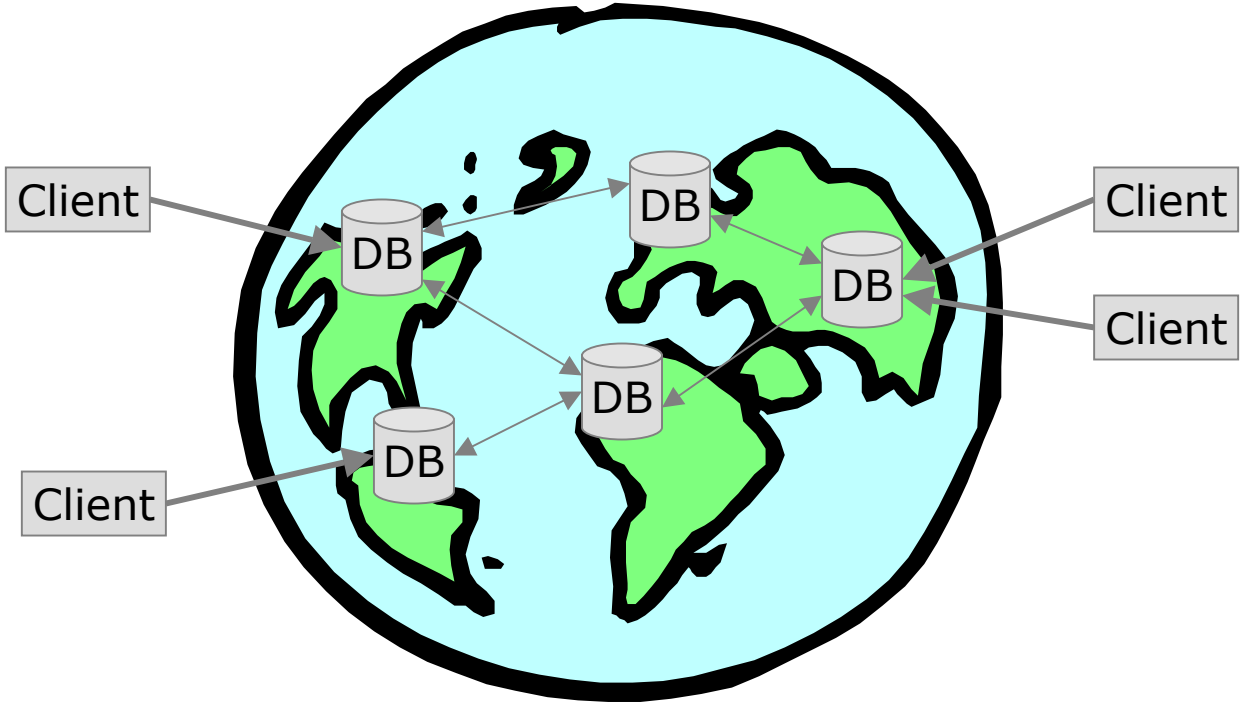
- Zugriffe von Clients auf die so genannte föderierte Datenbank
- Alle Clients verwenden dasselbe Protokoll
- Verbindung nur auf eine Datenbank nötig
- Föderierte Datenbank erstellt Unterqueries an Einzeldatenbanken
- Two Phase Commit



Database Middleware

Verteilte Datenbanken

- Verteilte Abspeicherung homogener Datenbestände auf vielen Orten für Clients auf vielen Orten
- Aufsplittung einer logischen Datenbank auf mehrere Server
- Verbindung zwischen Datenbanken für Client nicht sichtbar



Quellen

- "Enterprise Application Integration" von Wolfgang Keller
  - Kap. 4.3. – 4.4. | S. 76 – 95
- Carnegie Mellon – Software Engineering Institute
  - http://www.sei.cmu.edu/str/descriptions/middleware.html
  - http://www.sei.cmu.edu/str/descriptions/tpmt.html
  - http://www.sei.cmu.edu/str/descriptions/rpc.html
  - http://www.sei.cmu.edu/str/descriptions/momt.html
  - http://www.sei.cmu.edu/str/descriptions/orb.html
- Balzert H., Lehrbuch Grundlagen der Informatik, Spektrum Akademischer Verlag, Heidelberg, 1999
- Weisbecker A., Software-Management für komponentenbasierte Software-Entwicklung, Jost Jetter Verlag, Heimsheim 2002