

# NETZWERK- PROGRAMMIERUNG

Wie programmiert man Verteilte Systeme?

Prof. Dr. Georg Hinkel  
26.04.2024

# GLIEDERUNG

Datum	Vorlesung	Übungsblatt	Abgabe
19.04.2024	Einführung	HamsterLib	06.05.2024
26.04.2024	Netzwerkprogrammierung	Theorie	
03.05.2024	World Wide Web	HamsterRPC 1	20.05.2024
10.05.2024	Remote Procedure Calls	Theorie	
17.05.2024	Webservices	HamsterRPC 2	03.06.2024
24.05.2024	Fehlertolerante Systeme	Theorie	
31.05.2024	Transportsicherheit	HamsterREST	17.06.2024
07.06.2024	Architekturen für Verteilte Systeme	Theorie	
14.06.2024	Internet der Dinge	HamsterIoT	01.07.2024
21.06.2024	Namen- und Verzeichnisdienste	Theorie	
28.06.2024	Authentifikation im Web	HamsterAuth	15.07.2024
05.07.2024	Infrastruktur für Verteilte Systeme	Theorie	
12.07.2024	Wrap-Up	HamsterCluster (Bonus)	16.08.2024

## Agenda

- Grundlagen der Kommunikation
  - Kommunikationsmuster
  - Semantik von Nachrichten, Nachrichtenstruktur
  - Internetprotokolle TCP, IP, UDP (Wiederholung)
- Architekturmodelle für Verteilte Systeme

## Lernziele

- Kommunikationsmuster ableiten können
- Architekturmodelle ableiten können

### Beispiel: Diese Vorlesung

- Kommunikationspartner
    - Wer? Wieviele?
  - Kommunikationsrichtung
  - Nachrichteninhalt
    - Was wird kommuniziert?
  - Kommunikationskanal
    - Wie werden Nachrichten übertragen?
- Dozent (1), Studierende (viele)
  - meistens unidirektional
  - Natürliche Sprache (Deutsch)
  - Verteilte Systeme
  - Schall

### Verteilte Systeme

- 1:1, 1:n
- Verschiedene Kommunikationsmuster
- Nachrichtensemantik
  - JSON, XML oder binär
- Typischerweise TCP oder UDP

- Anzahl der Kommunikationspartner
  - Genau zwei
    - Einfachster Fall
    - Regelfall
  - Mehr als zwei
    - Typischerweise nicht mehr entscheidend, wie viele genau
    - Sog. Multicast-Dienst
    - Spezialfall: Broadcast
- Adressierung
  - Kommunikationspartner haben eindeutige Adressen
  - **Direkt:** Alle kennen alle (symmetrisch) oder Sender kennt Empfänger (asymmetrisch)
  - **Indirekt:** Kommunikation erfolgt über zwischengeschaltete Instanz (~Broker)
  - **Implizit:** Kommunikation im lokalen Netzwerk (Broadcast)

- Verbesserte Modularität
  - Sender und Empfänger können ohne Kenntnis des anderen implementiert werden
  - Beispiel: E-Mail
- Erweiterte Zuordnungsmöglichkeiten
  - 1:n, m:1, n:m
- Kommunikationspartner können transparent restrukturiert werden
  - Replikation
  - Ausfall eines Partners
- Aufgaben der Zwischeninstanz
  - Weiterleiten
  - Speichern und weiterleiten
  - Nachrichten transformieren

# GRUNDLAGEN DER KOMMUNIKATION

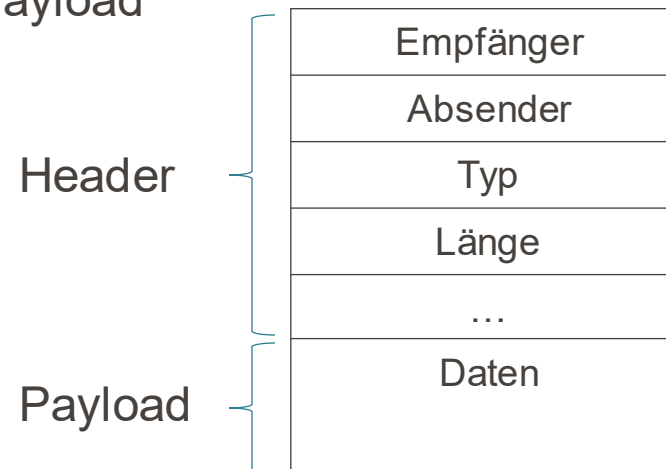
## Kommunikationsmuster für einzelne Nachrichten

- One Way
  - Einzelnachricht ohne Antwort oder Quittung
- Request / Response bzw. Auftrag / Antwort
  - Client-Rolle (Auftraggeber)
  - Server-Rolle (Auftragnehmer)
  - Synchron (blockierend) oder asynchron (nebenläufig) auf beiden Seiten (unabhängig voneinander)
- Publisher / Subscriber
  - Nachricht klassifiziert in Topics / Event Channel
  - Empfänger abonniert Topics (Subscriber)
  - Sender publiziert Nachrichten / Events (Publisher)
- Duplex
  - Alle Kommunikationspartner können immer senden und empfangen

## Beispiele

- Sensorik, eingeschränkte Hardware
- Häufigster Fall
- World Wide Web (HTTP)
- RPC, RMI, REST, ...
- Internet der Dinge (MQTT)
- Robotik (ROS)
- Chat-Protokolle

- Inhalt und Länge der Nachrichten muss immer festgelegt sein, sonst nur Bytestrom
  - Länge manchmal durch Transportprotokoll begrenzt oder festgelegt
  - Falls längere Nachricht notwendig → mehrere Pakete des Transportprotokolls zusammenfassen
- Aufbau der Nachrichten ist allen Kommunikationspartners bekannt
  - Typischer Aufbau: Header und Payload



- Payload kann typisierte Objekte (im Sinne der Objektorientierung) enthalten



- Frage: In welcher Reihenfolge werden die Bytes einer Zahl abgelegt
  - Niederwertige zuerst → Little Endian
  - Höherwertige zuerst → Big Endian

$2023_{10} \rightarrow 7E7_{16} \rightarrow$

07 E7 (Big Endian)
E7 07 (Little Endian)

- Achtung bei ganzen Zahlen: Reihenfolge der Bytes unterschiedlich
  - Host: meist abhängig von der Architektur, üblicherweise Little Endian
  - Netzwerk: Typischerweise Big Endian

# GRUNDLAGEN DER KOMMUNIKATION

## Kommunikationskanal

- Hardwarenahe Abstraktionsschichten typischerweise durch Betriebssystem abstrahiert
  - Schnittstelle für Verteilte Systeme
  - Änderung durch Anwendung nicht möglich
- Obere Netzwerkschichten in Anwendungen implementiert → keine Aktualisierung des Betriebssystems notwendig



# VERBINDUNGSPROTOKOLL

## Das Internet Protocol (IPv4)

- Verbindungslos
- Best-Effort Beförderung von Einzelnachrichten
  - Datagram ~ Datenpaket
  - Prüfsumme, aber nur für den Header
  - Begrenzte Lebensdauer für Pakete
- Adressierung von Kommunikationspartnern mit 32-bit Adressen
- IPv6
  - Immer noch nicht durchgesetzt, da man die Probleme mit IPv4 anderweitig in den Griff bekommen hat (CIDR, NAT)

### Transmission Control Protocol (TCP)

- Zuverlässiger, bidirektionaler Punkt-zu-Punkt-Transport eines Bytestroms
- Verbindungsorientiert
  - Duplex-fähige Verbindung zwischen Transport-Endpunkten
  - Endpunkte durch IP-Adresse und Portnummer adressiert (16bit)
- Sicherungsfunktionen
  - Sequenznummern
  - Prüfsummenbildung (wie IP)
  - Empfangsquittungen, Sendewiederholung nach Timeout
  - Sliding-Window für Flusskontrolle

### User Datagram Protocol (UDP)

- Bidirektionaler Best-effort Punkt-zu-Punkt Transport von Einzelnachrichten
- Verbindungslos
- Multicast / Broadcast-fähig
  - Direkte Umsetzung bei Multicast-fähigen Netzen wie Ethernet
- Keine Sicherungsfunktionen
  - Nur Prüfsummenbildung (laut Standard optional)  
→ Neuordnung, Verlust, Duplikation möglich

- Port-Nummern werden von der IANA (Internet Assigned Numbers Authority) vergeben
- Verwaltungsprozedur nach RFC 6335
  - 0-1023: reservierte System-Ports, well-known Port-Nummern von Standarddiensten
  - 1024-49151: registrierte oder User-Ports, können von der IANA zugewiesen werden
  - 49152-65535: private oder Ephemeral Port-Nummern von benutzerdefinierten Diensten
- Beispiele
  - 22: SSH (sichere Shell)
  - 25: SMTP (Email-Versand)
  - 53: DNS (Namensdienst zur Adressauflösung)
  - 80: HTTP (World Wide Web)

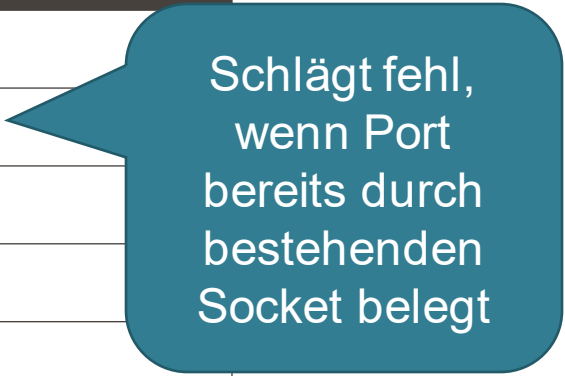
- Einfache API für die Entwicklung von verteilten Anwendungen
  - Eingeführt in 4.X BSD UNIX, heute von praktisch allen Betriebssystemen unterstützt
  - Heutzutage abgebildet in entsprechenden Klassen in allen gängigen Software Plattformen
  - Standardmäßig implementiert durch TCP/IP Implementierung im Betriebssystem
- Kommunikationsendpunkt
  - Sockets werden vom Betriebssystem verwaltet
  - Bietet Anwendung Abstraktion der darunterliegenden Hardware (Netzwerkkarte)
  - Verbreitete Basis für die Implementierung von Protokollen auf der Basis von TCP/IP
- Verschiedene Arten von Sockets
  - **Stream Sockets:** Verbindungsorientiert, verlässlich → TCP
  - **Datagram Sockets:** Verbindungslos, unzuverlässig → UDP
  - Raw Sockets: Zugriff auf unterlagerte Protokolle (e.g. IP), kaum verwendet

# SOCKET-PROGRAMMIERUNG

## API

- Einfache API für die Entwicklung von verteilten Anwendungen
  - Eingeführt in 4.X BSD UNIX, heute von praktisch allen Betriebssystemen unterstützt
  - Heutzutage abgebildet in entsprechenden Klassen in allen gängigen Software Plattformen

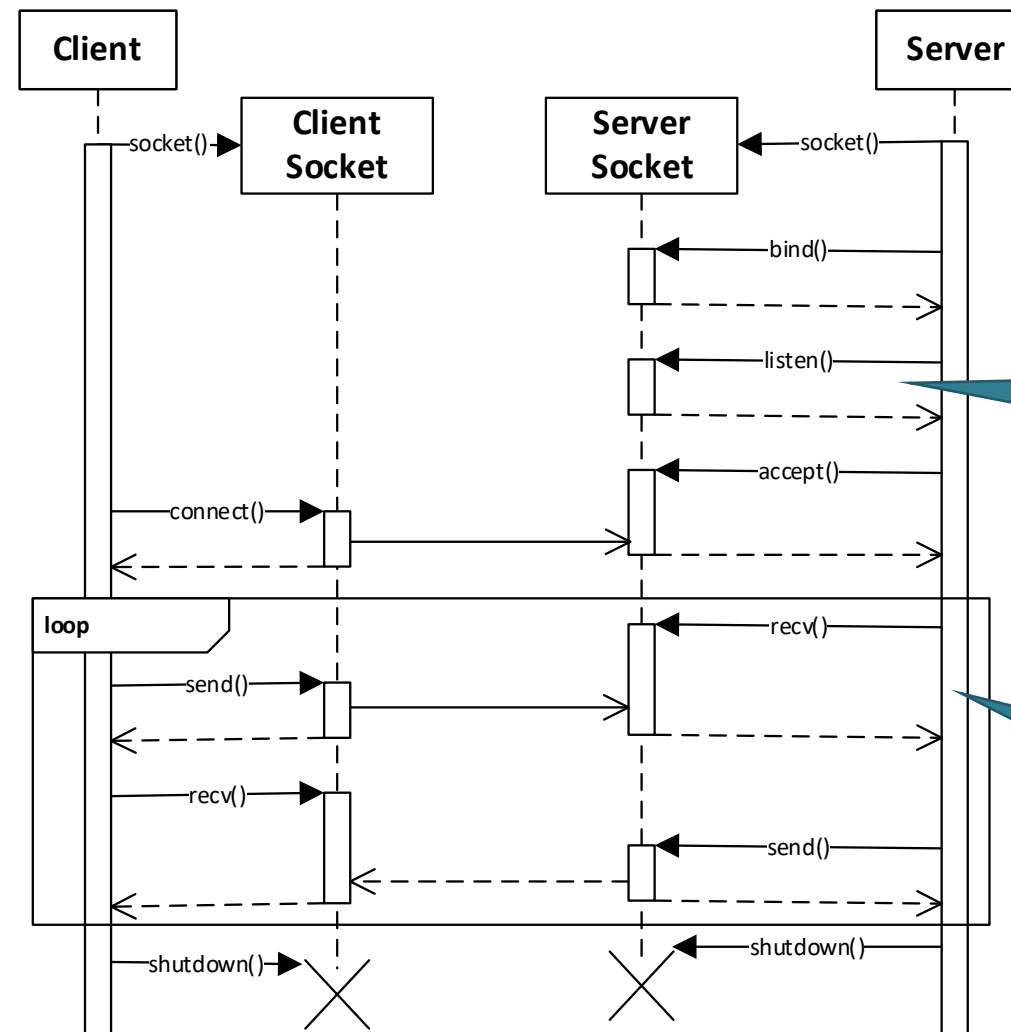
Methode	Funktion
socket()	Socket erzeugen
bind()	Zuordnung eines Sockets zu einer Adresse
listen()	Server: Vorbereiten auf Akzeptieren von Clients
accept()	Server: Warten auf Verbindungsanfrage
connect()	Client: Verbindung aufbauen
send() / recv()	Daten senden / empfangen
shutdown()	Verbindung schließen
close()	Socket freigeben



Schlägt fehl,  
wenn Port  
bereits durch  
bestehenden  
Socket belegt

# SOCKET-PROGRAMMIERUNG

## Ablauf



Listen() existiert in Java  
bzw. C# nicht

Sockets sind per se immer aktiviert  
→ Anwendung ruft Socket auf.  
Heutzutage eher umgekehrt  
→ Inversion of Control



- Abstraktion der tieferliegenden Protokollschichten
  - TCP, UDP, IP und darunter
  - Anwendungsentwickler können sich auf höhere Protokolle konzentrieren
- Schnittstelle sehr technisch
  - Insbesondere, „aktives Warten“ auf Serverseite
- Heutzutage abstrahiert durch Middleware
  - Schicht zwischen Betriebssystem und Anwendung
  - Verwaltet nötige Sockets
- Definiert grundsätzliche Einschränkungen
  - Bind schlägt fehl, wenn Port schon belegt
  - Verwaltung durch Betriebssystem

- Socket API war ursprünglich blockierend → Kontrollfluss wartet bis `send()` / `recv()` erfolgreich
  - Im Fall von TCP: Bis Empfang quittiert
  - Insbesondere für Server nicht akzeptabel (nur eine Verbindung gleichzeitig)
- Heutzutage: Nebenläufige Implementierung
  - Server behandelt Anfragen des Clients in separatem Thread (Middleware)
  - Typischerweise Nutzung eines Thread-Pools, um Gesamtzahl Threads zu begrenzen
  - Erfordert Synchronisation

# WAS GENAU IST SO EINE MIDDLEWARE?

Middleware-Konzepte

- Funktionalität
    - Flexibles Abbilden heutiger und künftiger Geschäftsprozesse
    - Integration existierender Systeme (Legacy)
    - Interoperabilität mit Fremdsystemen
  - Niedrige Kosten
    - Verringerung der Entwicklungszeit (time-to-market)
    - Verringerung der Entwicklungskosten (insb. in der Wartung)
    - Verringerung der Betriebs-/Managementkosten (total cost of ownership)
- ➔ Wiederverwendung als wichtiger Lösungsansatz

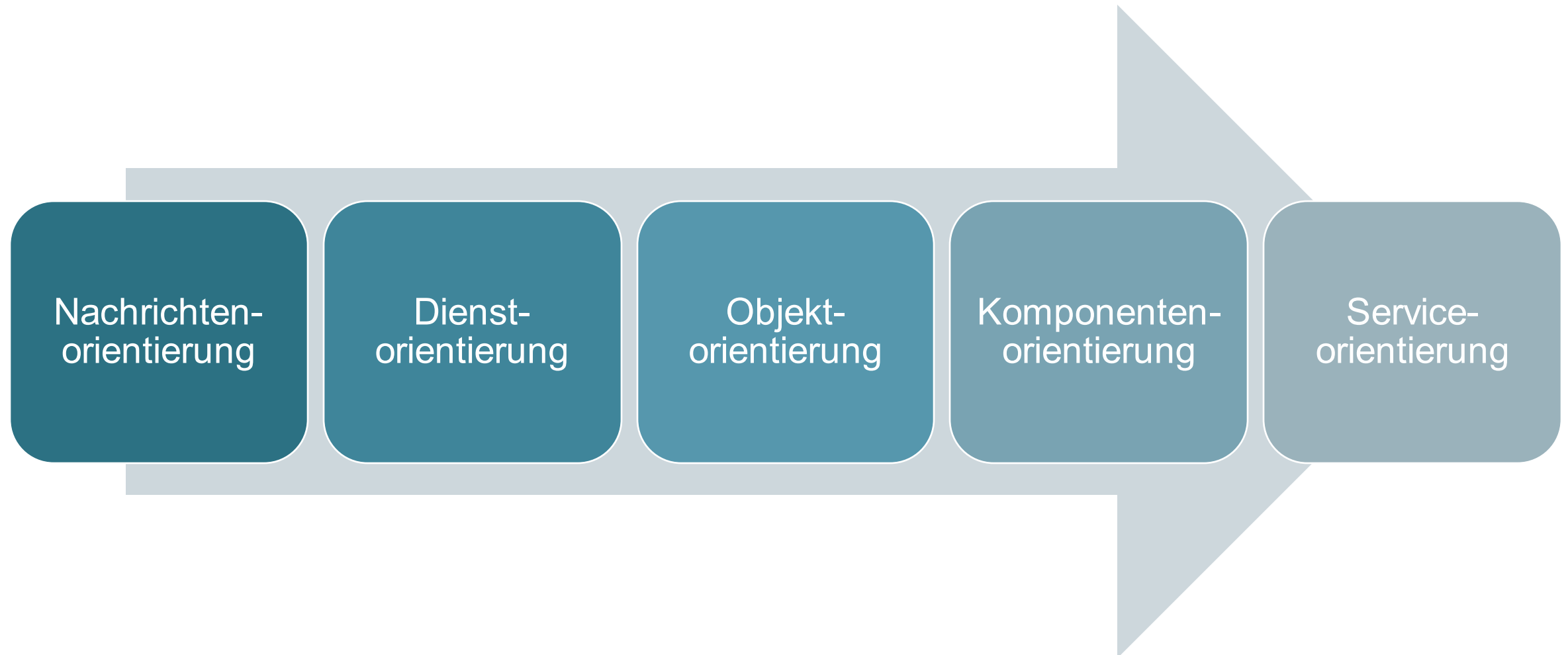
- Schicht aus Standardsoftware als Verteilungsplattform
  - Mehr oder weniger abhängig von Programmiersprache, Betriebssystem, Hardware
- Middleware typischerweise eingeteilt in verschiedene Paradigmen, die Struktur und Dynamik definieren
  - Strukturmodell
    - Verteilbare Einheiten (Programmkomponenten)
    - Benennung und Adressierung
  - Aktivitätsmodell / Dynamik
    - Akteure
    - Interaktionsmuster
    - Kommunizierte Einheiten
    - Synchronisation
- Implementierung durch Delegation auf niedrigere Schichten
  - Socket API

Anwendung

Middleware

Betriebssystem

Netzwerk



- Grundmodell verteilter Systeme
  - Prozesse als verteilbare Einheiten
  - Nachrichten als kommunizierte Einheiten
- Beispiel: Socket-Programmierung
- Beispiel: Programmierung hochgradig paralleler Anwendungen
  - Message Passing Interface (MPI) als de-facto Standard
  - High Performance Computing, GPUs
- Beispiel: Message Queues

# NACHRICHTENORIENTIERUNG

## Message Queues

- Message-oriented Middleware (MOM)
  - Typischerweise eingesetzt um Spitzenlasten abzufedern
- Beispiele
  - IBM WebSphere MQ
  - Microsoft Message Queue (MSMQ)
  - Java Messaging Service (JMS)
  - RabbitMQ
  - ZeroMQ

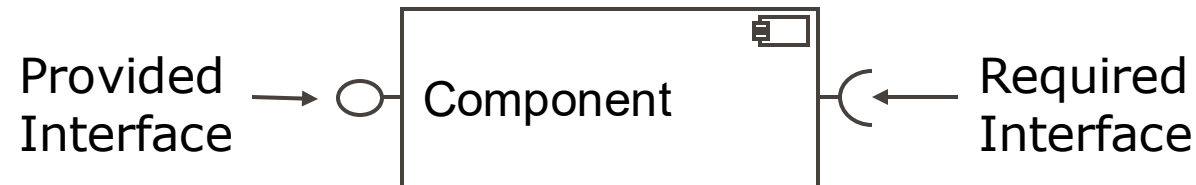




- Basis: Remote Procedure Call (RPC) → Separate Vorlesung
  - Dienste als verteilbare Einheiten
  - Dienst ist Menge angebotener Operationen / Funktionen
  - Nutzung entfernter Dienste durch Prozeduraufruf
  - Kommunizierte Einheiten sind Requests / Responses, enthalten typisierte Parameter in Netzdatendarstellung
- Basis für Client/Server-Anwendungen
- Bindung von Client und Server oft relativ statisch
- Beispiele
  - SunRPC
  - OSF DCE RPC
  - Apache Thrift
  - gRPC

- Objekte (OOP) als verteilbare Einheiten
- Kommunizierte Einheiten sind Methodenaufrufe, basierend auf RPCs
- Verteilte Anwendung ist Geflecht verteilter Objekte
- Wiederverwendung von Klassen auf Quellcodeebene
  
- Beispiele
  - OMG CORBA
  - Microsoft DCOM
  - Java RMI
  - OPC UA
  
- Bedeutung stark gesunken (bis auf OPC UA)

- Eigentlich entstanden als Mittel um Software zu strukturieren...
  - **“Components are for composition, much beyond is unclear...”** (Clemens Szyperski)



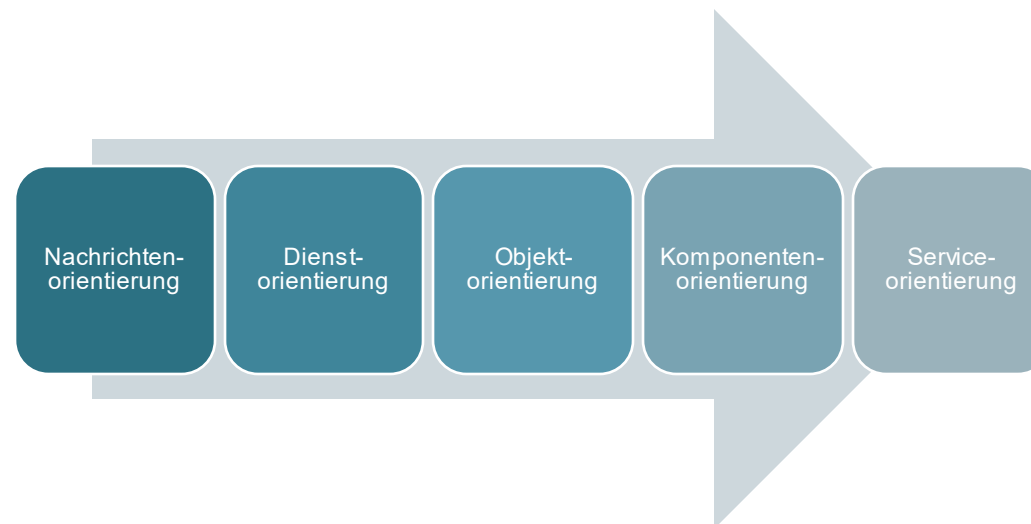
- Verwendung auch als Middleware-Konzept
  - Komponente als verteilbare Einheiten
  - Kommunizierte Einheiten sind Methodenaufrufe von Schnittstellen

- Komponentenmodell für Java
  - Komponenten heißen Bundles, bestehen aus Code und spezieller Klasse für Aktivierung
  - Komponenten können Dienste in Dependency Injection Container registrieren bzw. auflösen
  - Verbreitung in verteilten Systemen (siehe unten), aber auch komplexen Anwendungen
    - Eclipse Equinox
- Idee: Öffentliche Schnittstellen einer Komponente sind von außen per RPC erreichbar
- Enterprise Java Beans (EJB)
  - Teil der Spezifikation von Java-Schnittstellen für Server-seitige Komponenten (J2EE, heute JEE)
  - Skalierbarkeit durch Replikation einzelner Komponenten
  - Idealerweise flexibles Deployment der Komponenten auf Hardware
  - Beispielprodukte: JBoss, Apache Geronimo, JOnAS, IBM WebSphere, SAP NetWeaver, GlassFish

- Dienste als verteilte Einheiten
- Kommunizierte Einheiten sind Dienstaufrufe
- Anwendungen bestehen aus Integrationen von Diensten
- Architekturansatz für Geschäftsanwendungen
  - Aufteilung der benötigten Funktionalität in fachlich und organisatorisch getrennte Dienste
  - Separate Entwicklung der einzelnen Dienste, Wiederverwendung
- Selbstbeschreibung
  - E.g. Web Service Definition Language (WSDL), W3C-Standard
  - Definiert Parameter und Rückgaben durch XML Schema
- Zentrale Registrierung der Dienste (Service Registry)
  - Aufruf der Dienste über zwischengeschaltete Stelle (Enterprise Service Bus)

- Geschäftsprozess: Ablaufplan zur Erfüllung der Unternehmensziele
  - Interpretation in SOA: komplexe Interaktion zwischen Diensten
- Idee: Formale Modellierung von Geschäftsprozessen
  - Erlaubt automatische Ausführung → Web Service Orchestration
  - „Programmieren im Großen“ (Web Services als Einheiten)
- WS-BPEL (Business Process Execution Language)
  - OASIS Standard
  - Mittlerweile nicht mehr bedeutend
- BPMN (Business Process Model and Notation)
  - OMG Standard, verwandt zu UML Aktivitätsdiagramm
  - ISO/IEC 19510

- Sockets als allgemeine Schnittstelle für verteilte Systeme
  - Plattformunabhängig
  - Low-level, keinerlei Transparenz
- Middleware
  - Nachrichtenorientiert
  - Dienstorientiert
  - Objektorientiert
  - Komponentenorientiert
  - Service-orientiert



- Welches Betriebssystem-Primitiv wird für die Programmierung verteilter Systeme verwendet? Welche Protokolle werden auf diese Weise zur Verfügung gestellt und wie unterscheiden sich diese?
- Was versteht man unter indirekter Adressierung?
- Was sind gebräuchliche Middleware-Konzepte und wie unterscheiden sie sich?
- Welches Kommunikationsmuster würden Sie für eine gegebene Anwendung verwenden und warum?