# 01 Übung - Einführung

## 1. Geben Sie die Definition eines verteilten Systems wieder.

→ Ein verteiltes System ist eine Sammlung unabhängiger Rechenelemente, die ihren Benutzern als ein kohärentes (zusammenhängendes) System erscheint.

**alternative Definition "ein einziges System"**

- kohärentes System: einzelne Komponenten sind identifizierbar
- einzelnes System: einzelne Komponenten sind verdeckt, einzige "black box"
- letzteres schwieriger (bis unmöglich) zu realisieren; ältere Definition; heute gängig ist die aus van Steen/Tanenbaum.

## 2. Ist nach dieser Definition ein Rechner mit 2 CPUs ein verteiltes System?

→ Nein, da geteilte Hardware und nicht unabhängig funktionsfähig, denn sie verwenden z.B. die gleiche Stromquelle und den gleichen Speicher.

→ Verteilte Systeme sind eine Sammlung unabhängiger Rechenelemente. Darunter versteht man, dass sie autark funktionieren können und über keine geteilten physischen Komponenten (wie z.B. Speicher) verfügen

## 3. Welche Arten von Transparenz möchte man mit verteilten Systemen erreichen?

- **Zugriff**: Verberge, wo sich eine Ressource befindet
- **Lokation**: Verberge, wo sich eine Ressource befindet
- **Migration**: Verberge, dass eine Ressource den Ort wechseln kann (z.B. ein Handy, das bewegt wird)
- **Relokation**: Verberge, dass eine Ressource während der Nutzung an einen anderen Ort verschoben werden kann (z.B. Eine Webseite wird auf einen anderen Server verschoben)
- **Mehrbenutzer**: Verberge, dass mehrere Nutzer konkurrierend auf eine Ressource zugreifen
- **Fehler**: Verbergen von Fehler und das Wiederanlaufen einer Ressource

## 4. Welche Arten der Skalierbarkeit gibt es in verteilten Systemen?

- **In der Größe**: Benutzerzahl, Datenmenge, Prozessanzahl
- In der **maximalen Entfernung zwischen Knoten**. Bsp.: Berlin-Sydney
- In der **Zahl der beteiligten Organisationen** Bsp.: nur für eine Uni oder für ein System das Alle Unis nutzen können

**Wie kann man die Skalierbarkeit entlang der Dimensionen erreichen?**

- In der Größe: → Zahl von Benutzern
  - → Menge von Daten -> mehr Speicherplatz zur Verfügung stellen
    - -> schnellere Festplatten und Caching
    - -> Kaskadierende DBMS (bspw. Optimierung auf bestimmte Suchanforderungen)
    - -> Replikate vorhalten für parallele Nutzung
  - → Zahl von Prozessen -> Parallele Programmierung und mehr Rechenkerne (Kapazität)
    - -> Lastverteilung (Jobs sinnvoll auf Nodes bzw. Standorte verteilen, voneinander abhängige Aufgaben auf nahe Nodes)
    - -> Asynchrone Abarbeitung (nicht-zeitkritische Aufgaben, bspw. Indizierung der Datenbank, außerhalb der Last-/ Nutzungszeiten ausführen)
- in der maximalen Entfernung zwischen Knoten
  - → Server in anderen Regionen (weltweit), um Distanzen zu verringern
  - → Schnellere Übertragungsmedien (Glasfaser, Laser)
  - → Redundanz (Content Delivery Networks)
  - → Asynchrone Kommunikation zwischen Nodes (kein busy-wait auf Antwort)
- In der Zahl der beteiligten Organisationen
  - → Rechteverwaltung
  - → Anpassbarkeit in der Planung und Implementation gewährleisten (verständlichen und anpassbaren Code schreiben)
  - → Einhaltung von Standards (Standardprotokolle und -software; Vorteilhaft für die Kommunikation mit anderen Systemen und Security-Policy

## 5. Was versteht man im Zusammenhang mit verteilten Systemen unter Offenheit?

- Verteiltes System bietet leicht nutz- und integrierbare Komponenten und nutzt selber häufig fremde Komponenten
  - → eigene Entwicklungen quelloffen machen und dokumentieren
  - → OpenSource Software nutzen
  - → Standards nutzen
- wird erreicht durch: Verwendung von Standards/standardisierten Schnittstellen + hohe Flexibilität + Interoperabilität
- wichtig für Weiterentwicklung

## 6. Was macht ein verteiltes Informationssystem aus?

- Es ist ein verteiltes System
- Es geht darum wie man Daten verteilt
- Kommunikation ermöglichen

**Typen verteilter Systeme**

- Verteilte Hochleistungsrechner
- Verteilte Informationssysteme
- Verteilte Systeme für "pervasive computing" (bspw. internet of things / IoT)
- nicht ganz klar abgrenzbar (so könnte man z.B. viele Systeme für pervasive computing als Spezialfall verteilter Informationssysteme sehen). Viele Anforderungen/Probleme gibt es (ggf. mit unterschiedlicher Gewichtung) bei allen drei Typen.

**7. Sie sind Chef der Softwareentwicklung bei einem Startup im Onlinehandel, JENZON. Bislang war das Unternehmen noch recht klein und hatte im wesentlichen Kunden im Raum Jena. Jetzt soll aber deutschlandweit expandiert und die Kundenzahl deutlich erhöht werden. Sie planen daher die Umstellung des bisher zentralisierten Systems auf ein verteiltes Informationssystem. Der Geschäftsführer des Unternehmens recherchiert dazu und trifft auf ein Zitat von Leslie Lamport:"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable". Er findet das eher beunruhigend und fragt, ob sich solches Verhalten nicht verhindern lässt. Erklären Sie ihm**

### 7.a. Warum es nicht möglich ist, alle Fehler zu verbergen? / Warum kann Fehlertransparenz nicht vollständig erreicht werden?

- Skalierbarkeit bei JENZON: → **Größe**: mehr Nutzer, mehr Daten, mehr Prozesse
  → **Entfernung**: Nicht so dramatisch, aber es wird trotzdem eine Rolle spielen.
  → **Organisationen**: z.B. Einbindung von Lieferanten, Zustellern, lokalen Anbietern denkbar
- **Netzwerk**: zu hohe Netzwerklast
- **Programmierfehler**: unvorhergesehenes Verhalten
- **Ausfälle von Komponenten**: lokale Stromausfälle beim Server
- **Andere Abhängigkeiten**: Änderung anderer Systeme

### 7.b. Welche Maßnahmen Sie ergreifen können, damit das System möglichst stabil läuft. / Was kann man tun damit Systeme möglichst stabil laufen?

- Redundanz würde bei Netzwerküberlastung, Ausfällen von Komponenten, anderen Abhängigkeiten und Programmierfehlern helfen
- Kapselung/ Lose Kopplung…
- Qualitätssicherung
- Sicherheitsmechanismen

## 8. Fehlertransparenz ist offenbar nicht vollständig zu erreichen. Wie sieht es mit den anderen Transparenzarten aus? - Welche Transparenzen gibt es? – Welche Maßnahmen um diese zu erreichen?

- **Zugriff:** → Verbergen von unterschiedlicher Datenrepräsentation und -zugriff (z.B. zwischen unterschiedlichen Betriebssystemen)
  → Ansatz, z.B. eigenes internes Format für Zugriffe, Schnittstellen übersetzen oder neutrales Zwischenformat in und aus dem jeder übersetzt -> das möchte man vollständig erreichen
- **Lokation**: → Verbergen, wo sich eine Ressource befindet
  → Ansatz: Ansprechen von Ressourcen über Namen; Dienst anbieten, der Namen in Adressen übersetzt. (z.B. DNS = domain name service, Übersetzung zwischen Adressen und Namen)
  → Grenzen: kann sinnvoll sein, dass Anwendung weiß, das über große Entfernungen/in andere Zeitzonen kommuniziert wird, um Erwartungen an Antwortverhalten anpassen zu können
- **Migration**: → verbergen, dass Ressource sich während der Nutzung bewegt (z.B. Handy während Musik gestreamt wird
  → Ansatz: Lokationstransparenz und entsprechende Kommunikationsprotokolle

- **Relokation:** → verbergen, dass Ressource umgezogen wird.
  → Ansatz: (Lokationstransparenz und entsprechende Kommunikationsprotokolle
- **Replikation**: → verbergen, dass es mehrere Kopien einer Ressource gibt
  → Ansatz (Synchronisation zweier Kopien)
- **Mehrbenutzer**: → verbergen, dass es konkurrierenden Zugriff auf Ressourcen gibt
  → Ansatz: (Rechtesystem, Synchronisation wichtige Rolle, wechselseitiger Ausschluss, Transaktionen, …)
- **Fehler (Umgang mit Teilausfällen)**:
  → verbergen, dass Teile des Systems ausgefallen sind
  → Warum kann sie nicht vollständig erreicht werden?
  -> von außen häufig nicht zu unterscheiden, ob Verbindungsfehler, Knotenfehler oder einfach langsamer Knoten. Je nach Ursache müsste man aber zum Verbergen anders reagieren
  → Was kann man tun, dass Systeme möglichst stabil laufen?
  -> u.a. Redundanz einbauen, Sicherheitsmechanismen

## 02 Übung - Architekturen, Prozesse & Threads

### 1. Beschreiben Sie die klassische Drei-Schichten-Architektur.

Die klassische 3-Schichten-Architektur hat drei Schichten:
→ Das User-Interface Level (Anzeigen von Informationen, Entgegennahme von Benutzereingaben, Validierung der Eingaben)
→ Das Processing Level (Ausführung der Geschäftslogik, Verarbeitung und Validierung der Daten, Kommunikationssteuerung zwischen Präsentations- und Datenhaltungsschicht -> Anwendungsschicht: Kommuniziert über die Datenschicht mit API, e.g. Programmiersprachen wie Java, Python)
→ Das Data Level (Speichern von Abrufen von Daten, Verwaltung der Datenintegrität und -sicherheit, Bereitstellung von Schnittstellen zur Datenmanipulation -> Datenschicht: Datenbanken )

(siehe auch Grafik mit Aufteilung der Schichten auf Client / Server)
Man teilt die Schichten abhängig vom Use-Case auf den Client und Server auf.
Verschiebung der Applikationsebene hin zum Client kam mit günstigeren Herstellungskosten performanter Clients.

(siehe Grafik Seite 34 mit Client, Application und Database server)
Ein Server ist nicht immer nur Dienstanbieter, sondern kann auch selbst Anfragen stellen (bspw. App Server schickt Anfrage an Database Server)

### 2. Was sind Vor- und Nachteile von Schichtenarchitekturen?

**Vorteile**: → Kapselung
→ Robust (Schichten austauschbar, wenn UDP nicht gut funktioniert [weil Übertragungsqualität schlecht ist] stattdessen TCP verwenden)
→ Wartbarkeit (Klar strukturierter Code -> Trennung der Logik in unterschiedliche Schichten -> einfache Fehlerbehebung)
→ Skalierbarkeit (Leicht erweiterbar, verteilte Entwicklung -> Teams können parallel an verschiedenen Schichten Arbeiten -> schnellere Entwicklungszeit)
→ Flexibilität (Plattformunabhängigkeit -> Unterschiedliche Schichten können auf verschiedenen Plattformen oder Technologien basieren)

**Nachteile**: → Effizienzverluste, Overhead durch Übergang zwischen den Schichten
→ oft starke Abhängigkeit zwischen unterschiedlichen Schichten – starke Kopplung
→ Wiederholte Verarbeitung
→ Komplexe Fehlersuche (Schichtübergreifende Fehler, Isolationsprobleme)
→ Performance (Zusätzliche Latenz durch Datenfluss durch mehrere Schichten, Ressourcenintensiv -> Jede Schicht benötigt ihre eigenen Ressourcen)

### 3. Beschreiben Sie den Ablauf eines entfernten Methodenaufrufs.

- Client-Maschine ruft eine Methode auf.
- Dieser Methodenaufruf wird an einen Proxy (auch genannt Stellvertreterobjekt / Stubt) auf dem Client-Rechner gegeben. Dieser hat die Aufgabe, die Methoden eines Objekts zu emuliert.
- Der Proxy gibt den Methodenaufruf an den Server, auf dem das Objekt tatsächlich liegt, weiter. (Dies ist opak für das Programm auf dem Client-Rechner.)

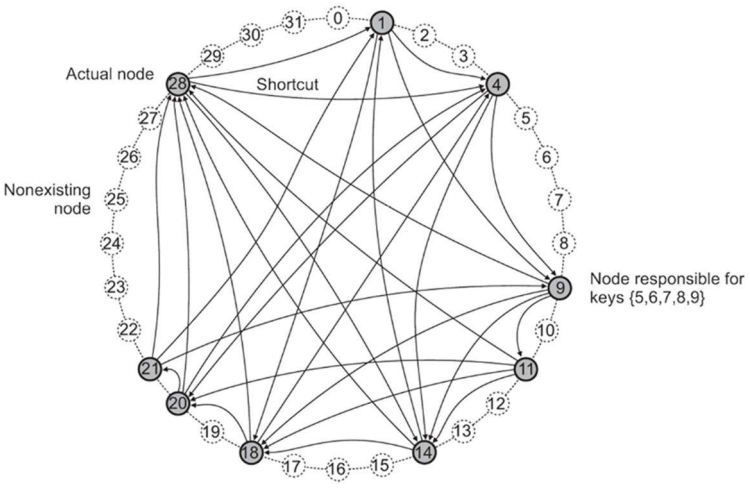
• Der Server hat ein Skeleton (das Gegenstück zum Proxy), welches den Methodenaufruf an das tatsächliche Objekt übergibt. Das Ergebnis wird auf dem gleichen Weg zurückübertragen

-> Für das Programm auf dem Client wirkt es so, als läge das Objekt auf dem eigenen Rechner.#

**4. Im Video/der Vorlesung wurden die http Operationen (PUT, GET, DELETE, POST) erwähnt, aber nicht weiter erläutert. Welche Funktionalität verbirgt sich jeweils dahinter?**

- **PUT**-Anforderung wird verwendet, um eine Ressource auf einem bestimmten Server zu aktualisieren oder zu erstellen.
- **GET**-Anforderung wird verwendet, um Daten von einem bestimmten Server zu erhalten.
- **DELETE**-Anforderung wird verwendet, um eine Ressource auf einem bestimmten Server zu löschen.
- **POST**-Anforderung wird verwendet, um Daten an einen bestimmten Server zu senden.

**5. Betrachten Sie den beispielhaften Chord-Ring auf Folie 8 im GVIS02_2-Foliensatz. Nehmen Sie an, bei Knoten 20 wird eine Anfrage nach dem Datenelement mit dem Hashwert 13 gestellt.**




**5. Wie wird diese Anfrage durch das Netz geleitet?**

via Fingertabellen: 20 → 4 → 9 → 11 → 14

→ ansonsten linear, einmal im Kreis alle Nachfolger, ausgehend von 20 bis 14

**6. Welche Aufgaben übernehmen (gemäß der Beschreibung im Buch) die Super-Peers im Skypenetzwerk?**

Im Allgemeinen sind „Super-Peers" Knoten im Netzwerk, die als „Broker" fungieren, welche Daten zur Ressourcennutzung und Verfügbarkeit für eine Anzahl von Knoten sammelt, die in unmittelbarer Nähe zueinander stehen, um schnell einen Knoten mit ausreichenden Ressourcen auszuwählen.

→ Skype nutzt diese Architektur, wobei die „Super-Peers" so genannte Skype-Super-Nodes sind und ein zusätzlicher zentralisierter Skype-Login-Server existiert.

**7. Erklären Sie kurz: URL, HTTP, HTML.**

- **URL** (Unified Resource Locator):
  - → Zeichenfolge zur Identifikation von Ressourcen im Internet zu identifizieren
  - → besteht aus mehreren Teilen, einschließlich des Protokolls, des Domainnamens oder der IP-Adresse des Servers, auf dem die Ressource gehostet wird, des Pfads zur Ressource
  - → optional zusätzliche Parameter (z.B. Queries)

- **Hypertext:**

  Es bildet die Grundlage für das World Wide Web, das eine riesige Sammlung vernetzter Dokumente darstellt, die über das Internet zugänglich sind. Das Web verwendet das Hypertext Transfer Protocol (HTTP) zur Übertragung von Daten und Hyperlinks zur Navigation zwischen den Seiten.

  Hypertext ist ein Konzept zur Organisation und Darstellung von Informationen, das es ermöglicht, Texte, Bilder oder andere Medien miteinander zu verknüpfen, indem Hyperlinks verwendet werden. Hyperlinks sind interaktive Elemente, die es Benutzern ermöglichen, von einem Dokument oder einer Stelle zu einer anderen zu springen, sei es innerhalb desselben Dokuments oder zu einem anderen Dokument oder einer anderen Ressource. Diese nicht-lineare Navigation ermöglicht es Benutzern, Informationen in einer nicht sequentiellen Reihenfolge zu durchsuchen und zu konsumieren, was eine flexible und anpassungsfähige Informationsdarstellung bietet. Hypertext bildet die Grundlage für das World Wide Web und viele andere digitale Informationsstrukturen.

- **HTTP** (Hyper Text Transfer Protocol):
  - → Transport / Transfer von Hypertext -> Protokoll, das zur Übertragung von Daten über das Internet verwendet
  - → gängiges Protokoll für viele Applikationen
  - → HTTP Methoden (GET: Abfragen von Websites, POST: Absenden von Formularen)

→ ermöglicht die Kommunikation zwischen einem Webbrowser und einem Webserver
→ Wenn ein Benutzer eine Webseite in seinem Browser aufruft, sendet der Browser eine HTTP-Anfrage an den Webserver, der die angeforderte Seite hostet. Der Webserver antwortet dann mit den entsprechenden Daten, die der Browser darstellt
→ HTTP legt die Regeln und das Format fest, nach denen diese Kommunikation stattfindet, einschließlich der Methoden zur Anforderung und Übertragung von Daten

• **HTML** (Hyper Text Markup Language)
→ Standard für den Aufbau von Website
→ Browser haben eigene Implementationen zur Darstellung von HTML (deshalb: Cross-Browser-Kompatibilität schwierig, außer man verwendet Frameworks, die das übernehmen)
→ ist ein Konzept zur Organisation und Darstellung von Informationen, das es ermöglicht, Texte, Bilder oder andere Medien miteinander zu verknüpfen, indem Hyperlinks verwendet werden
→ ermöglicht nicht-lineare Navigation, indem Benutzer von einem Dokument oder einer Stelle zu einer anderen springen können, sei es innerhalb desselben Dokuments oder zu einem anderen Dokument oder einer anderen Ressource

## 8. Was ist ein Prozess? Was ist ein Thread?

• **Prozess**: ist ein eigenständiges Programm mit eigenem Adressraum und eigenen Ressourcen. Prozesse laufen isoliert voneinander und verwenden IPC zur Kommunikation.
→ Laufendes Programm. Übersetzung von persistentem Speicher in den laufenden Zustand.

• **Thread**: ist eine leichtgewichtige Ausführungseinheit innerhalb eines Prozesses, die sich den Speicherraum und die Ressourcen des Prozesses teilt. Threads ermöglichen parallele Ausführung und effizientere Kommunikation, erfordern jedoch Synchronisationsmechanismen zur Vermeidung von Datenkonflikten.
→ Unabhängiger Nebenprozess, der Aufgaben im Namen eines Programms ausführt. Prinzipiell wie Prozesse, teilen sich aber einen gemeinsamen Adressbereich.

### Vergleich zwischen Prozessen und Threads

| Merkmal | Prozess | Thread |
|---|---|---|
| **Isolation** | Eigener Adressraum, isoliert voneinander | Gemeinsamer Adressraum innerhalb eines Prozesses |
| **Ressourcen** | Eigene Ressourcen (z.B. Speicher, Dateien) | Gemeinsame Nutzung der Ressourcen des Prozesses |
| **Erzeugung** | Langsamer, da mehr Ressourcen benötigt | Schneller, da weniger Ressourcen benötigt |
| **Kommunikation** | IPC (aufwendig) | Direkt (schneller und einfacher) |
| **Sicherheit** | Hoher Grad an Sicherheit und Stabilität | Erfordert Synchronisation, um Datenkonflikte zu vermeiden |
| **Wechselaufwand** | Höher | Niedriger |
| **Anwendungsfälle** | Unabhängige Programme | Parallele Aufgaben innerhalb eines Programms |

## 9. Die logische Anordnung von Knoten in einem strukturierten P2P-Netzwerk spiegelt in der Regel die physische Anordnung der Knoten nicht wieder. Was hat das für Vor- und Nachteile?

• **Vorteile:** → Nutzung bestehender Infrastruktur möglich (das P2P-Netzwerk ist nur logisch)
→ Distributed Hash Table (DHT) --> gleichmäßige Verteilung von Peers im Netz

• **Nachteile:** → Geographische Distanz spiegelt die Anordnung im P2P-Netz nicht wider! (aufgrund der Berechnung der Hash-Werte)
→ Durch Wegfall eines Peers oder einer Verbindung zu einem Peer wird die Verbindung im P2P-Netzwerk Unmöglich

## 10. Zu welchem Architekturstil gehören Webservice-basierte Systeme?

Webservice-basierte Systeme gehören typischerweise zum serviceorientierten Architekturstil (Service-Oriented Architecture, SOA). SOA ist ein Architekturstil, der darauf abzielt, Dienste bereitzustellen, die über ein Netzwerk verfügbar sind. Diese Dienste sind lose gekoppelt und können unabhängig voneinander entwickelt, bereitgestellt und verwaltet werden.

## 03 Übung – Kommunikation 1

**1. Nennen Sie die sieben Schichten der ISO/OSI-Architektur und beschreiben Sie, was jeweils ihre Aufgabe ist.**

→siehe Abbildungen bei „03/04 Kommunikation, Grundlagen der Kommunikation“

**2. Wir haben Kommunikationstypen entlang zweier Dimensionen unterschieden: persistente vs. transiente Kommunikation sowie synchrone vs asynchrone Kommunikation. Ordnen Sie ein Telefongespräch, einen Skypechat, eine Postkarte, Email, einen entfernten Prozeduraufruf, eine Vorlesung in BBB ein.**

- ***Telefongespräch***: transient, synchron
- ***Skypechat***: persistent, asynchron
- ***Postkarte***: asynchron, persistent (wenn man die Post als Middleware sieht, die den Brief behält oder zurücksendet) oder transient (wenn sie mit Hermes verschickt wird)
- ***E-Mail***: asynchron, persistent (E-Mail Server versucht Nachricht mehrfach zuzustellen, gibt dann erst auf)
- ***Remote Procedure Call***: synchron, transient (aber man kann es auch persistent machen und auch asynchron, aber dann nimmt man Message-Oriented-Middleware)
- ***Vorlesung in BBB***: synchron, transient (wenn Aufzeichnung asynchron, persistent)

- **persistent**: N wird im Kommunikationssystem gespeichert(?) → Bsp: chat, Postkarte, E-Mail, (BBB)
- **transient**: N wird nicht im Kommunikationssystem gespeichert(?) → Bsp: Telefon, RPC, (Big Blue Button(BBB))
- **synchron**: Client wartet/ist blockiert → Bsp: Telefon, RPC, RPC
- **asynchron**: Nach Senden wird unmittelbar weiter gearbeitet → Bsp: chat, Postkarte, E-Mail, (RPC), BBB

**3. Betrachten Sie die Prozedur incr(x, y) mit den beiden Parametern x und y. Die Funktion erhöht jeden der beiden Parameter um 1. Gehen Sie nun davon aus, dass diese Prozedur lediglich mit einem Parameter i (also incr(i, i)) aufgerufen wird. Initial war i auf 0 gesetzt. Welchen Wert hat i, wenn der Aufruf mit callby-reference bzw. copy/restore stattfand?**

??

## 04 Übung – Kommunikation 2

**1. Welche Arten von RPC gibt es?**

- Synchron, Multicast • Asynchron • Verzögert (Asynchron) mit Callback

**2. Wie werden Parameter beim RPC übergeben?**

- Übersetzung von Parametern in maschinen- und netzwerkunabhängiges Format
- by value
- by reference: → Verbieten (nicht wirklich praktikabel)
  → Nur lesender Zugriff: ersetze copy-by-reference durch copy-by-value
  → Schreibender Zugriff: ersetze copy-by-reference durch copy-by-value/restore
  → Verwende globale Referenzen

**Aufgabe 2,5: Fehlerbehandlung in RPC**

Auf Folien ist ein synchroner RPC-Aufruf abgebildet (der Client macht busy-wait, bis Antwort vom Server kommt). Frage: Was kann alles schief gehen?

- RPC (von Client zu Server) → Adresse falsch / Server ist woanders
  → Netzwerkverbindung getrennt
  → Nachricht falsch (falsche Prozedur, falsche Parameter, falsche Datentypen, …)
  → Routing schlägt fehlt
- RPC-Antwort kommt nicht zurück (same)
- Server: ist aus, crasht während der Verarbeitung, überlastet
- Server crasht während Verarbeitung
- Client crasht während er wartet

**3. Welche Operationen bieten TCP/IP-Sockets an?**

→ Socket (create socket), Bind, Listen, Accept, Connect, Send, Receive, Close.

**4. Nennen Sie je eine Anwendung, für ... a. das Request/Response Muster**

• das Request/Response Muster → RPC, Anfrage an Datenbank

**b. das Publish/Subscribe Muster**

• das Publish/Subscribe Muster → Newsticker

**c. das Pipeline Muster geeignet sind.**

• das Pipeline Muster geeignet sind. → Verteilung von Aufgaben an freie Ressourcen

**5. Welche Operationen bieten Nachrichtenwarteschlangen?**

• **get**: warten, bis Warteschlange ein Element enthält, dann erstes Element rausnehmen (busy wait)
• **poll**: nachschauen, ob etwas in der Queue ist: wenn nicht, weitermachen, wenn doch, Element rausnehmen und verarbeiten
• **notify**: informiere, wenn eine Nachricht in die angegebene Schlange getan wird. (Wer wird informiert? Der Kommunikationspartner? BKR: Derjenige, der notify aufruft)
• **put**: füge eine Nachricht an bestimmte Warteschlangen an

**6. Wozu braucht man Messagebroker?**

• Hilft in Netzwerken, bei denen man zwischen unterschiedlichen Systemen übersetzen muss, die unterschiedliche Anforderungen z.B. an Nachrichtenformate oder Datenstrukturen haben
• Bei Änderungen an der Kommunikation mit einem System, muss man nicht alle Netzwerkteilnehmer updaten, sondern nur den Messagebroker (z.B. wenn neue Komponente mit neuen Anforderungen dazu kommt)

**7. Nennen Sie je ein Beispiel für eine Anwendung, die sinnvollerweise mit FIFO-Kanal realisiert wird und eine, die ohne auskommt.**

• FIFO: Klassische Warteschlange, Chat mit mehreren Beteiligten
• kein FIFO: Upload von (unabhängigen) Dateien auf einen Server

**8. Verteilte Systeme arbeiten nicht immer fehlerfrei. Gehen Sie zunächst einmal davon aus, dass die Middleware Ihnen hier keine Unterstützung bietet. Welche Fehlerbehandlungen würden Sie in einem Programm vorsehen, das RPC nutzt? Welche Maßnahmen zur Fehlerbehandlung können in die Middleware verlegt werden? Welche Rolle spielt das Transportprotokoll in diesem Zusammenhang?**

• wurde gemeinsam mit Aufgaben 2 und 5 besprochen.
• genauere Betrachtung im Kapitel über Fehlerbehandlung
• generell: →

verloren gegangene Nachrichten können durch Transportprotokoll behandelt werden
Middleware z.B. Wiederanlauf nach Abstürzen
Anwendung z.B. korrekte Nachricht

**9. Wie funktioniert Informationsverbreitung mittels epidemischer Verfahren? Für welche Anwendungen sind diese Verfahren geeignet?**

• epidemische Verfahren sorgen dafür, dass Informationen im Hintergrund im Netzwerk verbreitet werden. Sie stellen sicher, dass (unter geeigneten Rahmenbedingungen (z.B. keine dauerhafte Partitionierung) Informationen irgendwann überall bekannt sind, geben aber keine Garantien, wann "irgendwann" ist. Geeignet also für Anwendungen, bei denen es nicht wichtig ist, dass alle Knoten immer aktuellste Informationen haben, sondern auch mit (leicht) veralteten Werten arbeiten können.

**10. Kann man mit IBM WebSphere MQ die Kommunikationsmuster aus Aufgabe 2 nachbilden?**

• Request/Response: ja
• Publish/Subscribe: ja
• Pipeline Pattern: nein, denn man kann nicht garantieren, dass nur der schnellste die Nachricht bekommt

**11. Würden Sie für Warteschlangen eine maximale Größe angeben?**

→ Ja: wir haben nur endliche Ressourcen und es gibt bedingt dadurch eine maximale Kapazität. Der Umgang mit einer vollen Warteschlange muss natürlich geklärt werden

# 05 Übung – Benennung (Naming)

## 1. Was ist ein Identifier? Worin liegt der Unterschied zu einem "normalen" Namen?

- zufällige Zeichenkette, **flat naming**: → Jede Entität wird über exakt einen Identifier referenziert
  → Jeder Identifier referenziert nur eine Entität
  → Nichts interpretieren, nur vergleichen
- Demzufolge ist eine Telefonnummer (auch eine Handynummer) ein schlechter Identifier, denn sie kann neu zugewiesen werden oder es klingeln gleich mehrere Telefone in einer Firma (Rufgruppen).
  → Eineindeutige Zuordnung (Referenz zu einer Entität) und eher zufällig generiert
- **Name**: menschenlesbar, interpretierbar, eventuell semantische Inhalte

## 2. Geben Sie ein Beispiel für einen "wahren" Identifier

- Steueridentifikationsnummer
- Sozialversicherungsnummer / Rentenversicherungsnummer
- MAC-Adresse (eigentlich nicht wirklich, aber wird beim Herstellen eindeutig an eine Netzwerkkarte vergeben)

## 3. Wozu benötigt man "Naming"-Services mit Namensauflösung (Name Resolution)?

- Für Internet
- Auflösen von menschenlesbaren Namen zu Adressen (bspw. IP bei DNS)
- Arbeiten mit IP-Adressen ist unpraktikabel für Menschen
- IP-Adressen können dynamisch vergeben werden (also sich ändern).
- Name bildet eine höhere Abstraktion

## 4. Betrachten Sie die unterschiedlichen Ansätze für "Flat Naming"

## a. Nennen Sie Vor- bzw. Nachteile der Architekturen.

## b. Überlegen Sie sich geeignete Maßnahmen/Ideen um einige dieser Nachteile zu überwinden (Verbesserung)

| Architektur | Nachteile | Verbesserung |
|---|---|---|
| Broadcasting | Kommunikation, Skalierung | nur im LAN (kleinen Netzwerken) einsetzen |
| Verweise | Ketten -> schwächstes Glied (Ausfall von einem Knoten) | weesschne |
| Home-based | Single Point of Failure (Home) | was anderes nehmen |
| DHT (Chord) | geographische Nähe spielt keine Rolle für die Nähe im Overlay-Netzwerk | Geographische Nähe einführen, aber ist technisch schwer umzusetzen (und sorgt dafür, dass weite zusammenhängende Bereiche gemeinsam ausfallen, bspw. Strom in einer Stadt fällt aus) |
| HLS (Hierarchy Location Service) | Die Wurzel muss alle Knoten (Blätter) kennen | Baum in mehrere Wälder aufteilen (aber man benötigt trotz dessen ein Root-Directory) |

## 5. ARP (Broadcasting) wird vor allem in Local-Area-Netzwerken genutzt. Skaliert dieser Ansatz ebenso bei großen Netzwerken? Begründen Sie kurz.

- Alle Rechner wären damit beschäftigt Broadcasts zu beantworten und zu verschicken. Bspw. wenn neue Rechner beitreten machen sie erstmal einen Broadcast.

## 6. Wie viele Einträge in einem HLS (Hierarchical Location Service) mit einer Tiefe k müssen maximal geändert werden, wenn eine Entität die Position ändert?

→ **2k+1 :** • gehen den Stang wo es gelöscht wird einmal durch →k
• gehen den Strang, in dem der Knoten hinkopiert wird →k
• Wurzel wird 2x berührt, also: 2k+1

**7. Worin unterscheiden sich "iterativer" von "rekursiver" Namensauflösung? Welche würden Sie verwenden? Begründen Sie kurz.**

- **iterativ:** der Client fragt alle Server in der Hierarchie selbst ab
- **rekursiv**: man fragt den Root-Server und dieser fragt die Server unter ihm ab und liefert dem Client eine Antwort zurück.

- **iterativ**: sicher, aber langsamer
- **rekursiv:** schneller, aber unsicher (bspw. DNS-Cache-poisoning)

**8. Unter welchen Umständen ist der "Home-based" Ansatz nicht performant. Kann man dieses Problem lösen/umgehen?**

•nicht performant: → permanenter Umzug
→ Client ist nah an der Entität (Home liegt weit weg)
-> Lösung: Zweistufige Anfrage, erst an lokalen DNS-Server, welcher entweder Entität finden
→ fertig oder nicht. Falls nicht wird Anfrage an Home gestellt

**9. Würden Sie die URL *http://www.acme.org/index.html* bzw. *http://www.acme.nl/index.html* als ortsunabhängig bezeichnen?**

- ***.nl*** → vermutlich in den Niederlanden (vielleicht ortsabhängig)
- ***.org*** → kein Hinweis auf einen bestimmten Ort (eher ortsunabhängig)

**10. Nehmen Sie für diese Aufgabe den Chord-Ring aus den Folien als Grundlage. Angenommen ein neuer Teilnehmer ist an Position "7" beigetreten. Berechnen Sie für diesen Peer die Fingertabelle. Hinweis: Alle anderen Peers müssen Sie nicht aktualisieren.**

| | |
|---|---|
| 1 | 9 |
| 2 | 9 |
| 3 | 11 |
| 4 | 18 |
| 5 | 28 |

**11. Angenommen wir fügen einen Knoten (Peer) im Chord ein. Müssen wir unmittelbar alle Fingertabellen aktualisieren?**

?

**12. Benennen und beschreiben Sie die wichtigsten Typen von DNS und geben Sie ggfs. ein Beispiel an.**

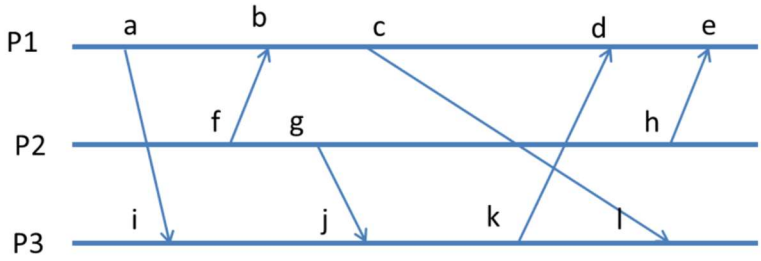
| Type | Description |
|---|---|
| A | IPv4 |
| AAAA | IPv6 |
| MX | Mail Server |
| CNAME | kanonischer Name |

**13. Worin unterscheiden sich "Naming"- von "Directoy"-Services? Wozu gehört "LDAP" bzw. "DNS"?**

| DNS | LDAP |
|---|---|
| Naming | Directory |
| “Telefonbuch” | “Gelbe Seiten” |

# 06 Übung – Koordination 1

**1. Betrachten Sie den hier dargestellten Ablauf. Tragen Sie in die Tabelle für die Ereignisse a bis l jeweils einen möglichen Lamport- und den Vektorzeitstempel ein. Gehen Sie bei der Vergabe der Vektorzeitstempel davon aus, dass diese nur beim Senden einer Nachricht hochgezählt werden.**




**Ohne Prozess-ID**

| Ereignis | Lamportzeitstempel | Vektorzeitstempel |
|---|---|---|
| A | 1 | [1,0,0] |
| B | 4 | [2,1,0] |
| C | 5 | [3,1,0] |
| D | 7 | [4,2,3] |
| E | 8 | [5,3,3] |
| F | 3 | [0,1,0] |
| G | 4 | [0,2,0] |
| H | 5 | [0,3,0] |
| I | 2 | [1,0,1] |
| J | 5 | [1,2,2] |
| K | 6 | [1,2,3] |
| L | 7 | [3,2,4] |

*Vektorzeitstempel wird hier beim Senden und Empfangen hochgezählt*

**Mit Prozess-ID**

| Ereignis | Lamportzeitstempel |
|---|---|
| A | 1.1 |
| B | 4.1 |
| C | 5.1 |
| D | 7.1 |
| E | 8.1 |
| F | 3.2 |
| G | 4.2 |
| H | 5.2 |
| I | 2.3 |
| J | 5.3 |
| K | 6.3 |
| L | 7.3 |

**2. Gehen Sie wie oben davon aus, dass in einem System die Vektorzeitstempel nur beim Versand von Nachrichten hochgezählt werden. Der Zeitstempel von P1 ist VC1 = [3,0,2]. Was sagt das aus?**

???

**3. In diesem System sollen Nachrichten kausal geordnet zugestellt werden. $P_1$ erhält nun eine Nachricht von $P_2$ mit dem Zeitstempel $VC_2$= [0,1,4]. Wie verhält sich P1? Warum?**

- **Option A**: P1 verarbeitet die Aktion sofort (nicht sinnvoll) BKR: das widerspricht der Forderung nach kausal geordneter Zustellung der Nachrichten.
- **Option B**: P2 wartet mit der Verarbeitung Prozess 2 hat bereits vier Aktionen von Prozess 3 verarbeitet. Da es sein kann, dass die Nachricht M2 abhängig von den zwei zwischenzeitlich durchgeführten Aktionen von P3 ist, müssen wir mit der Verarbeitung der Nachricht warten, bis wir auch alle Nachrichten von P3 erhalten haben (sinnvoll).

**4. Braucht man für den total geordneten Multicast unbedingt die Bestätigungsnachrichten?**

???

# 07 Übung – Koordination 2

**1. In dem zentralisierten Ansatz zum gegenseitigen Ausschluss gewährt der Koordinator normalerweise dem ersten Prozess in der Warteschlange die Berechtigung, wenn er eine Nachricht von einem Prozess erhält, der seinen exklusiven Zugriff auf die von ihm verwendete Ressource freigibt. Geben Sie einen anderen möglichen Algorithmus für den Koordinator an.**

→ Möglichkeiten: Zufall --> LIFO, FIFO ( Wer zuerst reinkommt, wird zuerst bedient.)

**2. Betrachten Sie wiederum den zentralisierten Ansatz zum gegenseitigen Ausschluss und nehmen Sie an, dass der Koordinator abstürzt. Fällt dadurch stets das System aus? Falls nicht, unter welchen Umständen tritt das ein? Gibt es eine Möglichkeit, das Problem zu vermeiden, sodass das System Abstürze des Koordinators toleriert?**

→ Es gibt einen Ersatzkoordinator, der im Hintergrund inaktiv ist. Dieser wird bei aktiven Anfragen aktiviert, aber falls kein Prozess gerade am laufen ist, so muss es auch nicht aktiviert werden.

**3. Was sind Vor- und Nachteile des dezentralisierten Verfahrens zum wechselseitigen Ausschluss?**

- **Vorteile**: Ausfallsicherheit/Fehlertoleranz > n/2, Skalierbarkeit
- **Nachteile**: Nachrichten, Konsistenz, Verzögerung

**4. Wie kann das Tokenbasierte Verfahren zum wechselseitigen Ausschluss so gestaltet werden, dass es gut mit Knotenausfällen zurecht kommt?**

- In gewissen Szenarien haben wir generell keine Probleme mit Knotenausfällen.

1) Angenommen ein Knoten mit Interesse an der Ressource fällt aus, ohne jedoch das Token zu besitzen. In dem Fall bemerkt der Vorgänger den Ausfall und überspringt ihn einfach. Sollte der ausgefallene Knoten bis zur nächsten Runder verfügbar sein, kann das Interesse bekundet und das Token genutzt werden.
2) Der Ausfall des Nachfolgers kann mit Hilfe von Timeouts zusammen mit Bestätigungsnachrichten realisiert werden. Möchte ein Prozess den Token weitergeben, erwartet er eine Bestätigungsnachricht vom Nachfolger. Bleibt diese aus, wird der Nachfolger übersprungen.
3) Selbst der Ausfall eines Prozesses, welcher gerade den Token besitzt, kann toleriert werden. In dem Fall kann eine maximale Umlaufzeit des Tokens definiert werden. Realisiert ein Prozess, dass das Token länger als die maximale Zeit no

**5. Nehmen Sie an, dass zwei Prozesse gleichzeitig den Ausfall des Koordinators bemerken und beide entscheiden, eine Wahl unter Verwendung des Bully-Algorithmus abzuhalten. Was geschieht?**

→ Siehe Frage 8. Hier ist ein ähnliches Szenario beschrieben. Selbst mehrere Starts einer Koordinator-Wahl enden stets im gleichen Ergebnis. Unabhängig vom Initiator wird IMMER der gleiche Prozess als neuer Koordinator gewählt.

**6. Vervollständigen Sie die folgende Tabelle.**

| Algorithmus | Nachrichten pro Eintritt/Austritt | Eintrittsverzögerung (in # Nachrichten) |
|---|---|---|
| Zentralisiert | 3 | 2 |
| Verteilt | 2(n - 1) | 2(n - 1) |
| Tokenbasiert | 1 bis oo | 0 bis n - 1 |
| Dezentralisiert | 3mk, k = 1,2, ... | 2m |

**7. Wie ändern sich die Einträge in der oberen Abbildung, wenn wir annehmen, dass die Algorithmen in einem LAN implementiert werden können, das Hardware-Broadcasts unterstützt?**

| Algorithmus | Nachrichten pro Eintritt/Austritt | Eintrittsverzögerung (in # Nachrichten) |
|---|---|---|
| Zentralisiert | 3 | 2 |
| Verteilt | 1 + (n − 1) | 1 + (n − 1) |
| Tokenbasiert | 1 bis oo | 0 bis n - 1 |
| Dezentralisiert | 3mk, k = 1,2, ... | 2m |

**8. Kann der Bully-Algorithmus mit Netzwerkpartitionen, also der Unterbrechung der Kommunikation zwischen zwei Teilen des Netzwerks, so dass zwei unverbundene Teilnetze entstehen, umgehen?**

- Jeder Prozess mit einer höheren Nummer als die zwei genannten Prozesse erhält zwei ELECTION-Nachrichten, wird aber die zweite ignorieren. Die Wahl geht wie gewohnt von statten. Dies ist der Fall, wenn zwei Prozesse innerhalb des gleichen Teilnetzwerkes sind. Andernfalls wählen beide Teilsysteme ungeachtet von einander einen eigenen/separaten Koordinator. Hierbei müssen gewisse Sonderfälle betrachtet werden, die allgemein nicht auftreten können.

→ Ein paralleler Zugriff wäre auf Grund von zwei unabhängigen Koordinatoren möglich. Das Verhalten muss jedoch unterbunden werden.

→ Offen bleibt auch, wie geordnet nach Wiedervereinigung der Teilnetzwerke die Koordinatoren zusammengeführt werden können. Im Netzwerk darf es natürlich nur einen Koordinator geben. Ansonsten können Schreib-Schreib-Konflikte entstehen.

**9. Ein verteiltes System kann mehrere unabhängige Ressourcen besitzen. Stellen Sie sich vor, dass Prozess O auf Ressource A und Prozess 1 auf Ressource B zugreifen möchte. Kann der Algorithmus von Ricart und Agrawala zu Deadlocks führen? Erläutern Sie Ihre Antwort.**

???

**10. Beim zentralisierten Verfahren zum wechselseitigen Ausschluss gibt der Koordinator nach der Freigabe in der Regel dem ersten Element in der Warteschlange den Zugriff. Welche anderen Möglichkeiten gäbe es hier?**

???

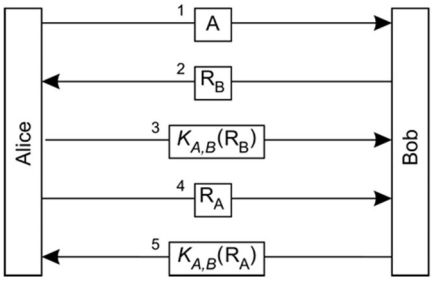
## 08 Übung – Sicherheit

**1. Nehmen wir an, dass Sie gebeten wurden, eine verteilte Anwendung zu entwickeln, die Lehrern das Vorbereiten von Prüfungen erlauben würde. Benennen Sie mindestens drei Anweisungen, die in den Sicherheitsrichtlinien für eine derartige Anwendung enthalten wären.**

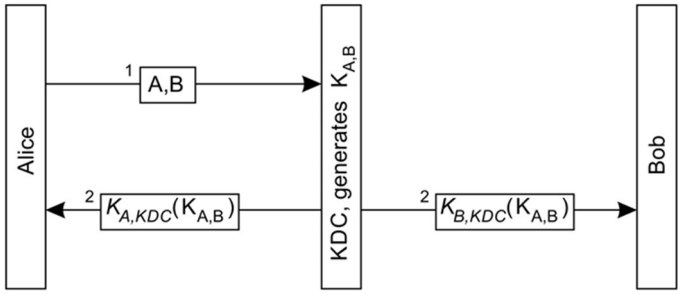
- **Zugriffskontrolle und Authentifizierung**: Implementieren Sie eine robuste Zugriffskontrolle und Authentifizierungsmechanismen, um sicherzustellen, dass nur autorisierte Benutzer (Lehrer) Zugriff auf die Anwendung und die Prüfungsdaten haben.
- **Verschlüsselung und Datensicherheit**: Sicherstellen, dass alle sensiblen Daten sowohl während der Übertragung als auch im Ruhezustand verschlüsselt sind, um unbefugten Zugriff und Datenverlust zu verhindern.
- **Datenschutz und Compliance**: Implementieren Sie Datenschutzmaßnahmen und sorgen Sie dafür, dass die Anwendung den geltenden Datenschutzgesetzen und -vorschriften entspricht.

**2. Wäre es sicher, in dem dargestellten Authentifizierungsprotokoll Nachricht 3 und Nachricht 4 zu $K_{A,B}$ (RB,RA) zusammenzufassen?**

→ ja, man könnte die Nachrichten zusammenfassen.
→ Bob muss die Nachricht dennoch entschlüsseln, um seine Nachricht herauszunehmen.



**3. Warum ist es in der folgenden Abbildung nicht nötig, dass das KDC sicher weiß, dass es mit Alice spricht, wenn es eine Anfrage für einen geheimen Schlüssel erhält, den Alice gemeinsam mit Bob nutzen kann?**




→ Eine dritte Person würde einen verschlüsselten (verpackten) Schlüssel erhalten, welcher durch den dritten Akteur nicht entpackt/entschlüsselt werden kann und somit für diesen wertlos ist.
→ Annahme ist, dass das KDC für jeden Nutzer einen Schlüssel hat, um den Austausch mit diesen separat zu verschlüsseln

**4. Was spricht dagegen, eine Nonce als Zeitstempel zu implementieren?**

→ Man verliert den Aspekt der Eineindeutigkeit und Zugehörigkeit, Methode ist weiter Angreifbar.
→ Umsetzung mit großen Zufallszahlen ist besser
• Zeitsynchronisation (Zeitstempel basieren auf der Systemuhr des Clients und des Servers → nicht synchronisierte Uhren)
• Begrenzte Granularität (Granularität von Zeitstempeln (z.B. in Sekunden oder Millisekunden) könnte nicht fein genug sein, um sicherzustellen, dass jede Anfrage eine wirklich einzigartige Nonce hat)
• Replay-Angriffe (Zeitstempel alleine schützen nicht effektiv vor Replay-Angriffen, wenn ein Angreifer in der Lage ist, eine Nachricht mit einem gültigen Zeitstempel abzufangen und erneut zu senden, solange der Zeitstempel noch als gültig betrachtet wird)
• Erhöhter Verwaltungsaufwand

**5. Warum benötigt man für eine sichere Kommunikation beides, "Authentifizierung" und "Integrität"? Überlegen Sie sich zusätzlich zwei entsprechende Szenarien bei denen jeweils nur einer der beiden Aspekte erfüllt ist.**

• **Authentifizierung**: Dies stellt sicher, dass die Kommunikationspartner tatsächlich diejenigen sind, für die sie sich ausgeben. Ohne Authentifizierung könnte ein Angreifer die Identität eines legitimen Benutzers vortäuschen.
• **Integrität**: Dies stellt sicher, dass die übermittelten Daten während der Übertragung nicht manipuliert wurden. Ohne Integrität könnte ein Angreifer die Daten ändern, ohne dass der Empfänger dies bemerkt.

• **Warum beide notwendig sind:**
→ ***Authentifizierung ohne Integrität:*** Selbst wenn sichergestellt ist, dass die Kommunikation von einer authentifizierten Quelle stammt, könnten die Daten während der Übertragung geändert werden. Der Empfänger würde den manipulierten Inhalt fälschlicherweise für authentisch halten.
→ ***Integrität ohne Authentifizierung:*** Auch wenn sichergestellt ist, dass die Daten nicht verändert wurden, könnte der Sender ein Angreifer sein, der die Daten an den Empfänger schickt. Der Empfänger hätte keine Garantie, dass die Daten von einer vertrauenswürdigen Quelle stammen.

• **Szenario 1: Authentifizierung ohne Integrität**
→ ***Beispiel***: Ein Benutzer loggt sich erfolgreich in ein Online-Banking-System ein, wodurch seine Identität authentifiziert wird. Ein Man-in-the-Middle-Angreifer fängt die Nachrichten ab und verändert den Betrag einer Überweisung, bevor sie die Bank erreicht.
→ ***Gefahr***: Die Bank weiß zwar, dass die Überweisung vom authentifizierten Benutzer stammt, bemerkt jedoch nicht, dass die Daten (Überweisungsbetrag) während der Übertragung manipuliert wurden.
→ ***Konsequenz***: Der Benutzer könnte einen falschen Betrag überweisen, was zu finanziellen Verlusten führt.

• **Szenario 2: Integrität ohne Authentifizierung**
→ ***Beispiel***: Eine unverschlüsselte E-Mail wird mit einer Prüfsumme gesendet, die sicherstellt, dass der Inhalt während der Übertragung nicht verändert wurde. Allerdings kann jeder beliebige Absender die E-Mail schicken.
→ ***Gefahr***: Ein Angreifer könnte eine E-Mail mit einem schädlichen Inhalt senden, die Prüfsumme korrekt berechnen und anhängen.
→ ***Konsequenz***: Der Empfänger würde glauben, dass die E-Mail nicht manipuliert wurde, hätte aber keine Garantie, dass sie von einer vertrauenswürdigen Quelle stammt. Dies könnte zu Phishing-Angriffen oder der Verbreitung von Malware führen.

**6. Wieso werden "Sitzungsschlüssel" (Session-Keys) eingesetzt und nicht vorhandene Schlüssel genutzt?**

→ Aus großen Datenmengen könnten sonst möglicherweise Informationen über den Schlüssel gewonnen werden.
• **Begrenzte Lebensdauer**: Sie minimieren das Risiko durch kürzere Schlüsselverwendungszeiten.
• **Schlüsselerschöpfung:** Sie verringern das Risiko kryptografischer Angriffe durch regelmäßigen Schlüsselwechsel.
• **Replay-Schutz**: Sie erschweren Replay-Angriffe durch zeitlich begrenzte Schlüsselverwendung.

- **Perfect Forward Secrecy**: Sie schützen vergangene Sitzungen auch bei Kompromittierung eines Langzeitschlüssels.
- **Angriffsvektor Minimierung:** Sie reduzieren die Möglichkeiten für Angreifer, Schlüssel zu analysieren und zu kompromittieren.
- **Isolation von Sicherheitsvorfällen**: Sie begrenzen die Auswirkungen eines kompromittierten Schlüssels auf einzelne Sitzungen.

**7. Begründen Sie kurz, ob die vier Bedrohungen (Interruption, Interception, Modification, Fabrication) trotz Verschlüsselung weiterhin funktionieren bzw. genutzt werden können.**

- **Interruption**: Verschlüsselung kann Unterbrechungen nicht verhindern.
  → ***Angriffe***: -> (D)DoS (bspw. Deauthentication-Angriff), -> Trennung der Verkabelung
  → ***Gegenmaßnahmen***: Firewalls, Replikation, Backups

- **Interception**: Verschlüsselung schützt die Dateninhalte, aber nicht die Metadaten.
  → ***Angriffe***: -> Schnüffeln (Packet Sniffing), -> Lauschangriff (Eavesdropping)
  → ***Gegenmaßnahmen***: -> Verschlüsselung, -> Datenverkehr (Traffic Padding)

- **Modification**: Verschlüsselte Daten können manipuliert werden, wenn keine Integritätsprüfungen vorhanden sind.
  → ***Angriffe***: -> Änderungen in Dateien, -> Änderungen in Nachrichten des Netzwerkes
  -> Re-Konfiguration der Hardware oder Netzwerktopologie
  → ***Gegenmaßnahmen***: IDS (Intrusion Detection System), Verschlüsselung, Traffic Padding, Backups

- **Fabrication**: Fälschungen sind möglich, wenn keine starke Authentifizierung und Integritätsprüfungen vorhanden sind.
  → ***Angriffe***: -> SQL Injections, -> Email-Spoofing (bspw. spear phishing)
  → ***Gegenmaßnahmen***: Authentifizierung & Autorisierung, Firewalls, Signaturen

**8. Nennen Sie einige Vor- und Nachteile der Verwendung zentralisierter Server zum Schlüsselmanagement.**

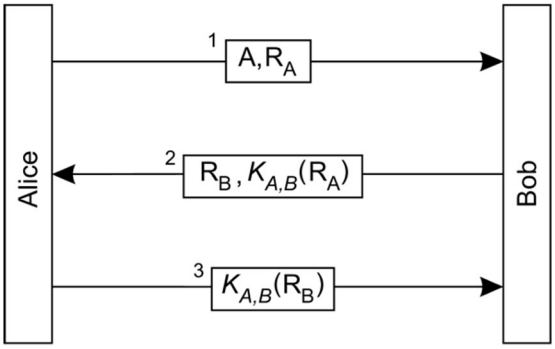
- **Vorteile**: → ***Zentrale Kontrolle und Verwaltung*** (zentrale Stelle ermöglicht eine konsistente und einheitliche Verwaltung aller kryptografischen Schlüssel, was die administrative Komplexität reduziert und die Implementierung von Sicherheitsrichtlinien erleichtert.)
  → ***Erhöhte Sicherheit*** (Ein zentralisierter Server kann besser gesichert werden, sowohl physisch als auch durch Software-Sicherheitsmaßnahmen. Es ist einfacher, starke Sicherheitsprotokolle und Zugriffskontrollen an einem zentralen Ort durchzusetzen)
  → ***Auditierbarkeit und Compliance*** (zentrales Schlüsselmanagementsystem erleichtert das Logging und die Überwachung aller Schlüsseloperationen, was für Audits und die Einhaltung von Sicherheitsstandards und -vorschriften wichtig ist)
  → ***Skalierbarkeit*** (zentralisiertes System kann leichter skaliert werden, um eine wachsende Anzahl von Schlüsseln und Benutzern zu unterstützen.)
  → ***Konsistente Richtlinienimplementierung*** (Sicherheitsrichtlinien und Schlüsselmanagementpraktiken können konsistent über alle Systeme und Anwendungen hinweg angewendet werden.)
- **Nachteile**: → ***Single Point of Failure*** (zentraler Server stellt einen einzigen Ausfallpunkt dar. Wenn der Server ausfällt oder kompromittiert wird, können alle abhängigen Systeme beeinträchtigt werden.)
  → ***Leistungsengpässe*** (Zentralisierte Systeme können zu Leistungsengpässen führen, insbesondere wenn viele Anwendungen und Benutzer gleichzeitig auf den Server zugreifen.)
  → ***Hohe Sicherheitsanforderungen*** (Der zentrale Server muss extrem gut gesichert sein, da er ein attraktives Ziel für Angreifer darstellt. Dies erfordert kontinuierliche Investitionen in Sicherheitsmaßnahmen und Überwachung.)
  → ***Netzwerkabhängigkeit*** (ständige Verfügbarkeit und Zuverlässigkeit des Netzwerks ist kritisch, da alle Schlüsseloperationen über das Netzwerk abgewickelt werden müssen.)
  → ***Skalierungsprobleme bei großer geografischer Verteilung*** (Wenn Nutzer und Systeme weltweit verteilt sind, können zentrale Server zu Latenzproblemen führen und die Benutzererfahrung beeinträchtigen)

**9. Das Protokoll für den Diffie-Hellman-Schlüsselaustausch kann auch verwendet werden, um einen gemeinsamen geheimen Schlüssel zwischen drei Parteien einzurichten. Erklären Sie, wie das geht.**

- Private Schlüssel wählen
- Öffentliche Schlüssel berechnen
- Öffentliche Schlüssel austauschen
- Zwischenschlüssel berechnen
- Endgültigen gemeinsamen Schlüssel berechnen

**10. Beim Diffie-Hellman-Schlüsselaustausch gibt es keine Authentifizierung. Durch Ausnutzen dieser Eigenschaft kann ein böswilliger Dritter, Chuck, leicht in den Schlüsselaustausch zwischen Alice und Bob eindringen und dann die Sicherheit zerstören. Erklären Sie, wie das ginge.**

11. Betrachten Sie folgenden "verbesserten" Ablauf der "Shared-Secret-Key"-Authentifizierung, welche weniger Nachrichten verschickt, aber nicht mehr "sicher" ist. Begründen Sie warum und nennen Sie eine Schwachstelle.




12. In der Vorlesung wurden drei unterschiedliche Authentifizierungsmechanismen für zwei Akteure/Teilnehmer vorgestellt. Versuchen Sie diese auf eine Gruppe von Teilnehmern zu übertragen oder lesen Sie die entsprechenden Abschnitte im Buch ("Secure group communication" & "Secure replicated servers", Seite 527f )

a. Wie viele Schlüssel müssen im System verwaltet werden?

b. Welche Vor- bzw. Nachteile haben die Mechanismen?

# 09 Übung - Replikation und Konsistenz

**1. Was sind die drei Dimensionen, entlang derer beim „continuous consistency model“ Abweichungen erlaubt sein können? Nennen Sie je ein Beispiel, bei denen eine bestimmte Art von Abweichung sinnvoll ist.**

- **Werte**: → absolut (Rundungsfehler, Ortsinformation (zBsp Abweichung < 100m))
  → relativ (Sensormesswert Änderung < 5%)
- **Aktualität (Zeit)**: → periodische Updates (einmal täglich)
  → Um x Stunden veraltete Werte (z.B. periodische Updates der Website)
- **Ausführungsreihenfolge (Zahl und Operationen):**
  → Addition / Subtraktion (Zählen von Votes od. Views auf einem Post) könnte man auch unter Aktualität einordnen
  → Versionsverwaltung
  → (gemeinsames) Editieren

**2. Geben Sie einen Ablauf an, der kausal aber nicht sequentiell konsistent ist.**

- **Kausale Konsistenz:** bedeutet, dass alle Operationen, die kausal miteinander verbunden sind, in der gleichen Reihenfolge gesehen werden müssen. Das bedeutet, wenn eine Operation A kausal vor einer Operation B steht, dann müssen alle Knoten im System sehen, dass A vor B kommt
- **Sequentielle Konsistenz:** bedeutet, dass alle Operationen in einer globalen Reihenfolge erscheinen, die die Reihenfolge der Operationen auf jedem einzelnen Knoten widerspiegelt. Das heißt, es gibt eine totale Reihenfolge der Operationen, die in der Reihenfolge auf allen Knoten sichtbar ist.

- Ein Ablauf, der kausal konsistent aber nicht sequentiell konsistent ist, könnte so aussehen:
  → Angenommen, wir haben zwei Operationen: O1 und O2.
  → Operation O1 wird auf Knoten A ausgeführt.
  →Operation O2 wird auf Knoten B ausgeführt, und O2 ist kausal von O1 abhängig (das bedeutet, O2 hat Auswirkungen auf oder ist von O1 beeinflusst)

**3. Geben Sie einen Ablauf an, der weder kausal noch sequentiell konsistent ist.**

- **Szenario:**
  →Operationen: - O1: Benutzer A veröffentlicht einen Beitrag auf einem sozialen Netzwerk.
  - O2: Benutzer B kommentiert den Beitrag von Benutzer A.
  - O3: Benutzer C sieht den Kommentar von Benutzer B.
  →Reihenfolge der Operationen: - Benutzer A veröffentlicht den Beitrag O1.
  - Benutzer B kommentiert den Beitrag O2.
  - Benutzer C sieht den Kommentar O3.
  →Verteilte Knoten: - Knoten A: Speichert die Operationen in der Reihenfolge O1, O2.
  - Knoten B: Speichert die Operationen in der Reihenfolge O2, O1.
  - Knoten C: Speichert nur O3 und hat keinen Einblick in O1 oder O2.

- **Beispiel für einen Ablauf, der weder kausal noch sequentiell konsistent ist:**
  → Knoten A (zeigt O1 und O2): Benutzer A sieht den Beitrag und den Kommentar in der Reihenfolge O1, O2.
  → Knoten B (zeigt O2 und dann O1): Benutzer B sieht zunächst den Kommentar O2 und dann den Beitrag O1.
  → Knoten C (zeigt nur O3): Benutzer C sieht nur den Kommentar O3 ohne den Beitrag O1.

**4. Nennen Sie je zwei Anwendungen, für die sequentielle bzw. kausale Konsistenz benötigt wird.**

- **kausal**: Chatanwendung, Forum: Antwort auf einen Beitrag
- **sequentiell**: - Banktransaktion: Buchung einer Reise vs. Kontolimit erreicht (einmal wird abgebucht und einmal wird Buchung abgelehnt)
  - Sensorwert Broadcasting
  - Robotersteuerung

**5. Welche der klientenzentrischen Konsistenzmodelle sollte ein verteiltes Mailsystem unterstützen? Begründe.**

- Aktionen auf Mails: abrufen, markieren, schreiben, löschen, …
- monotones Lesen → abrufen (es wird mindestens die letzte gelesene Version [Nachrichten im Postfach] gesehen)
  → Nachrichten werden stets in der richtigen Reihenfolge vom Server bereitgestellt.
  →markieren (gelesen / ungelesen)
- monotones Schreiben → Entwürfe, die auf dem Server gespeichert werden
- writes-follow-read → Antwort auf Nachricht, Weiterleiten
- read-your-writes → Gesendete Nachrichten

**6. Kann man mit Eintrittskonsistenz sequentielle Konsistenz nachbauen? Wenn ja: wie?**

→ Ja, wenn bei jedem Zugriff Locks verwendet werden (wenig sinnvoll, aber möglich)

**7. Neben den von uns betrachteten Konsistenzmodellen gibt es auch die Linearisierbarkeit. Hier gilt: Das Ergebnis der Ausführung ist dasselbe, als wären die Lese- und Schreiboperationen aller Prozesse in einer (überall gleichen) sequentiellen Reihenfolge ausgeführt worden. Die Operationen jedes einzelnen Prozesses erscheinen in der Reihenfolge, in der sie von den jeweiligen Programmen vorgegeben werden. Ist der Zeitstempel einer Operation Op1 kleiner als der einer Operation Op2, so erscheint Op1 vor Op2 in dieser Reihenfolge. Ist dieses Konsistenzmodell schwächer oder stärker als das der sequentiellen Konsistenz? Belegen Sie Ihre Aussage durch ein Beispiel – geben Sie also einen Ablauf an, der das eine Konsistenzmodell erfüllt, das andere aber nicht.**

→ stärker als sequentiell: bei sequentiell können Operationen in einer anderen Reihenfolge geschehen als in Echtzeit, man muss sich nur auf eine Reihenfolge einigen. Bei Linearisierbarkeit muss zusätzlich dazu die Reihenfolge mit der echten Reihenfolge übereinstimmen

**8. Van Steen und Tanenbaum beschreiben Konsistenzmodelle als Vertrag zwischen Anwendungsprogramm und Middleware. Welche Zusagen machen beide Partner in diesem Vertrag?**

- **Anwendungsprogramm**: Arbeitet korrekt (und zwar immer), wenn Middleware die Regeln einhält
- **Middleware**: Unterwirft sich gewissen Regeln des Konsistenzmodells und erzeugt nur Abläufe, die diesem Modell Entsprechen

**9. Einschubfrage: In a multi-user game, the players move figures around a common scene. The state of the game is replicated at the players' workstations and at a server, which contains services controlling the game overall, such as collision detection. Updates are multicast to all replicas.**

**(i) The figures may throw projectiles at one another and a hit debilitates the unfortunate recipient for a limited time. What type of update ordering is required here? Hint: consider the 'throw', 'collide' and 'revive' events.**

→ sequentiell (oder kausal, wenn nur der Aspekt "Projektile" betrachtet wird, nicht Gesamtspielsituation)

**(ii) The game incorporates magic devices which may be picked up by a player to assist them. What type of ordering should be applied to the pick-up device operation?**

→ egal, wenn unendlicher Vorrat an Zaubergeräten; sequentiell, wenn konkurrierender Zugriff

## 10 Übung - Replikation, Konsistenz 2

**1. Wozu braucht man serverinitierte Replikate? Wozu dienen clientseitige Replikate (Caches)? Wie unterscheiden sich server- und clientinitierte Replikate?**

| Aspekt | Serverinitierte Replikate | Clientseitige Replikate (Caches) |
|---|---|---|
| Zweck | - Datenverfügbarkeit<br>- Lastverteilung<br>- Fehlertoleranz | - Leistungssteigerung<br>- Netzwerkoptimierung |
| Initiierung | - Vom Server initiiert und verwaltet | - Vom Client initiiert und verwaltet |
| Verwaltung und Kontrolle | - Zentral durch den Server<br>- Kontrolle über Konsistenz und Verfügbarkeit | - Lokal durch den Client<br>- Kontrolle über Cache-Strategien |
| Datenquelle | - Hauptserver liefert Daten an Replikate | - Client speichert Kopien von Daten, die vom Server abgerufen werden |
| Update-Frequenz | - Regelmäßig vom Server an Replikate | - Regelmäßig oder bei Bedarf vom Client aktualisiert |
| Hauptvorteile | - Erhöht Verfügbarkeit und Skalierbarkeit<br>- Verbessert Fehlertoleranz | - Verbessert Zugriffszeiten<br>- Reduziert Server- und Netzwerklast |
| Typische Nutzung | - Datenbanken<br>- Verteilte Systeme<br>- Hochverfügbarkeitslösungen | - Webbrowser-Caches<br>- Anwendungs-Caches<br>- Proxy-Caches |

| | Permanent | Serverinitiiert | Clientinitiiert |
|---|---|---|---|
| Wer? | Admin | Server | Client |
| stat. o. dyn | statisch | dynamisch | dynamisch |
| schreibbar | ja | ja | meist nein |
| aktualisiert | ja | ja | in Verantwortung des Clienten |

**2. Quorumbasierte Protokolle verlangen bei Systemen mit N Knoten, dass**

• gegeben: Es sind 12 Replikate abgebildet. Minimales Quorum = 7, Maximales Schreibquorum = 12

**a. das Schreibquorum größer als N/2 sein muss.**

•Man kann bei einem Ausfall von bis zu (N/2)−1 schreiben und trotz dessen ist Konsistenz garantiert

•bei 2 Schreiboperationen gibt es mindestens einen Knoten, der beide Schreiboperationen bekommen hat, daraus lässt sich eindeutige Reihenfolge ermitteln

**b. das Lese- und Schreibquorum größer als N sein muss.**

• Man liest immer von mindestens einem Knoten die aktuellen Änderungen

**3. Quorumbasierte Protokolle geben im Rahmen der o.g. Regeln Spielräume bei der Wahl der Quoren. Wann würden Sie ein möglichst großes Schreibquorum wählen, wann ein möglichst kleines?**

• **großes Schreibquorum**:
- möglichst aktuelle Werte
- wenig write, viel read, weil Lesequorum dann kleiner gewählt werden kann und weniger aufwendig ist

• **kleines Schreibquorum:**
- Wenn es ausreicht, auch einen veralteten Wert zu lesen, dann kann das Schreibquorum kleiner sein
- viel write, wenig read
- Verfügbarkeit: wenn ein Server ausfällt und von einem writequorum angefragt wird, kann das quorum nicht durchgehen, d.h. write-operation geht nicht durch und geht verloren. bei kleinem writequorum ist die Wahrscheinlichkeit gering, dass ein ausgefallener Server gefragt wird also geht das quorum mit einer höheren Wahrscheinlichkeit durch

• **Kann es sinnvoll sein die Bedingung aufzuweichen?**
- R+W muss nicht unbedingt größer als N sein, wenn es nicht so wichtig ist dass bei read eine mögchlichst aktuelle Antwort kommt
- W kleiner als N/2, wenn die Konsistenz nicht so wichtig ist
- Amazon Dynamo: kleines write-quorum, damit write möglichst immer funktioniert (z.B. Artikel in den Warenkorb tun), Inkonsistenz in Kauf genommen

**4. Was ändert sich, wenn die Bestätigung der Schreiboperation an den Klienten nicht ganz am Ende sondern nach Annahme der Operation durch den Primärserver erfolgt? Welche Vor- und Nachteile bringt das mit sich?**

• **Bestätigung nach dem Schreiben**: +keine veralteten Werte
-geringere Fehlertoleranz
-Client länger blockiert, insbesondere wenn ein Knoten ausfällt

• **Bestätigung nach der Annahme**: -echt veraltete Werte
+höhere Fehlertoleranz
+Client kürzer blockiert

**5. Was passiert, wenn für unterschiedliche Datenelemente unterschiedliche Server die Primärkopie halten? Erreicht das beschriebene Protokoll dann noch sequentielle Konsistenz? Erreicht es kausale Konsistenz? Welche Änderungen sind ggf. nötig?**

→ sequentielle Konsistenz nur möglich, wenn zusätzlich zum Protokoll ein total geordneter Multicast mit Bestätigungen implementiert wird

→ **Sequentielle Konsistenz** wird schwer zu gewährleisten, wenn unterschiedliche Server die Primärkopien für unterschiedliche Datenelemente halten, da eine globale Koordination der Operationen notwendig ist. Um sequentielle Konsistenz zu erreichen, müssen Mechanismen zur globalen Reihenfolgenkontrolle implementiert werden.

→**Kausale Konsistenz** ist möglich, auch wenn unterschiedliche Server die Primärkopien halten, solange kausale Beziehungen korrekt verwaltet und erkannt werden. Es sind jedoch Mechanismen erforderlich, um sicherzustellen, dass kausale Abhängigkeiten über die Server hinweg korrekt erfasst und aufrechterhalten werden.

**6. Zustandsbasierte Leases werden verwendet, um einen Server zu entlasten, indem es ihm ermöglicht wird, so wenige Clients wie gerade nötig zu überwachen. Führt dieser Ansatz zwangsläufig zu einer besseren Leistung?**

- **Lease**: Versprechen vom Server, dass er innerhalb der Lease Updates an den Clienten pusht. Nach Ablauf der Lease muss Client neue Lease anfragen oder alle Updates selbstständig pullen. Leasedauer kann ein festes Zeitintervall sein oder vom Zustand des Servers abhängen

- **Zustandsbasierte Leases**: Die Dauer und Anzahl der verteilten Leases hängt von der Belastung des Servers ab. Hohe Belastung -> geringe Leasedauer an weniger Clienten

- **Bessere Leistung**: generell ja, außer bei hoher read-to-update-ratio, dann lohnt sich pushen nicht mehr

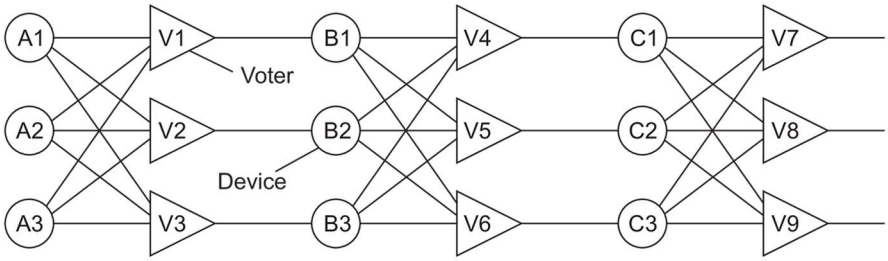
**7. Eine Datei wird auf zehn Servern repliziert. Listen Sie alle Kombinationen eines Lese- und Schreibquorums auf, die der Abstimmungsalgorithmus erlaubt**

- $\frac{10}{2} = 5$ -> mindestens 6 Schreibquoren

| write quorum | min. read quorum |
|---|---|
| 6 | 5 |
| 7 | 4 |
| 8 | 3 |
| 9 | 2 |
| 10 | 1 |

## 11 Übung – Fehlertoleranz

Abbildung: Modell der dreifach modularen Redundanz:




**1. Geben Sie die Definitionen für Defekt, Mangel und Fehler an und nennen Sie jeweils ein Beispiel**

- **Defekt** (Failure): eine Komponente kann nicht mehr (vollständig) seine definierten Aufgaben erfüllen →Bsp: Programmabsturz
- **Mangel** (Error): Teil der Komponente, der zu einer Fehlfunktion führen kann → Bsp: Programmierfehler
- **Fehler** (Fault): Ursache eines Mangels → Bsp: übermüdeter Programmierer

**2. Nennen Sie Beispiele für Maßnahmen zur Fehlervermeidung**

- **Audits** (ohne Ausführung, Code-Qualitäts-Standards einhalten)
- **Automatisierte Tests** (mit Ausführung, wenn Änderungen an Programmen vorgenommen werden, können mit automatisierten Tests sichergestellt werden, dass alle Funktionalitäten von vorher weiterhin funktionieren)
- Aufbau von Redundanzen
- technisch / organisatorische Maßnahmen (Brandschutz)
- **Weiterbildung** (und natürlich einstellen von ausreichend qualfizierten Mitarbeitern) / Personalmanagement (keine Überarbeitung von Programmierern und Systemadministratoren) / Pair Programming / Kaffeemaschine

**3. Nennen Sie Beispiele für Anwendungen, für die at-most-once/at-least-once Semantik sinnvoll ist und begründen Sie.**

- **at-most-once:** → Fernsteuerung Drohne (fliege 5 Meter nach unten)
  → Banküberweisung
- **at-least-once:** → Fernsteuerung Drohne (fliege auf 5 Meter Höhe, Idempotenz)
  →Alarm (Feuer, Einbruch, Bewegung)

→ Nicht alles was auch idempotent ist, sollte man auch mit at-least-once ausführen: großer Rechenauftrag für Computing Cluster sollte at-most-once ausgeführt werden, weil die Kosten zur Erledigung sehr hoch sind

**4. Was passiert beim 2PC, wenn der Koordinator in der Unsicherheitsphase abstürzt?**

- Es kann dazu führen, dass alle Teilnehmer im READY-Zustand stecken bleiben.

(Unterschied zwischen WAIT & READY:

→ **WAIT**: Knoten hat VOTE-REQUEST gesendet und wartet auf Antwort ABORT/COMMIT

→ **READY**: Knoten hat sein VOTE-ABORT/-COMMIT gesendet und wartet auf GLOBAL-ABORT/-COMMIT vom Koordinator, bereit zum Ausführen der Aufforderung)

- **Lösung**: Die Teilnehmer könnten sich dann untereinander verständigen und die Zustände abfragen. Man könnte sogar einen neuen Koordinator wählen.

**5. Kann das Modell der dreifach modularen Redundanz (siehe oben) byzantinische Fehler handhaben?**

→ Ja. Das sind genau solche Ausfallfehler die das Modell behandeln soll.

**6. Wie viele ausgefallene Elemente (Geräte plus Wähler) kann die dreifach modulare Redundanz (siehe oben) kompensieren? Geben Sie ein Beispiel für den schlimmsten Fall, der maskiert werden kann.**

→ Es darf immer 1 von 3 ausfallen. Nicht jede Kombination darf ausfallen (zum Beispiel alle Stränge dürfen einzeln ausfallen, also A1, V1, B1, V4, C1, V7).

→Bei 5 Strängen dürften 2 Stränge ausfallen. Für 4 funktioniert es nicht.

7. Kann man das Modell der dreifach modularen Redundanz (oben) für fünf bzw. x Elemente pro Gruppe statt für drei verallgemeinert werden? Wenn ja, welche Eigenschaften hat es?

**8. Beim 2PC erfolgen Einträge in ein Log. Erläutern Sie, warum diese vor/nach der jeweiligen Operation geschrieben werden müssen**

→ Als erstes merkt er sich, dass er Koordinator ist für eine bestimmte Abstimmung (Zustand: WAIT)

→ Nach dem alle vote requests rausgeschickt wurden, sollte er sich merken, dass er jetzt in die nächste Phase WAIT wechseln kann

→ Wenn alle Votes eingegangen sind, soll er sich merken, ob er in COMMIT oder ABORT steht

- **Warum ist ein ACKNOWLEDGEMENT (ACK) wichtig?**

  →Man kann bei Ausbleiben eines ACK erneut die GLOBALs schicken und wenn alles abgeschlossen ist, kann man die vielen (jetzt unwichtigen Informationen) aus dem Log entfernen.

**9. JENZON ist ein Startup im Bereich des Onlinehandels, welches recht gut etabliert ist und viele Kunden hat. Wie stellen Sie Fehlertoleranz Ihres Systems sicher? Insbesondere: Welche Maßnahmen ergreifen Sie, um Datenverlust zu vermeiden?**

- Statische Inhalte: CDN
- Replikate liegen an unterschiedlichen Standorten mit unterschiedlichen Systemen (gleichartige Geräte haben auch meist gleichartige Probleme, also sollte man auch unterschiedliche Systeme einsetzen)
- Kritische Prozesse mehrmals implementieren

**11. Warum ist es so schwierig, das Fail-Stop-Modell im Fall von Absturzfehlern zu implementieren?**

→ schwierig, zu erkennen, ob ein Element im System abgestürzt/nicht erreichbar ist. Auch der Grund für einen Ausfall ist schwer für andere Systemteilnehmer zu erkennen. Verlässlich erkennbarer Ausfall ist nicht möglich.

**12. Betrachten Sie einen Webbrowser, der eine veraltete Seite aus dem Cache zurückgibt, statt einer neueren, die auf dem Server aktualisiert wurde. Ist das ein Fehler und wenn ja, welche Art von Fehler?**

→ Kommt auf die Anforderungen an die Konsistenz an. Wenn diese sehr strikt ist, handelt es sich um einen Konsistenzfehler

**13. Geben Sie an, für welche der folgenden Anwendungen die Höchstens-einmal-Semantik oder die Mindestens-einmal-Semantik am besten geeignet ist. Begründen Sie Ihre Antwort.**

**a. Lesen und Schreiben von Dateien auf einem Dateiserver**

→ Beim Lesen egal, beim Schreiben kann es jedoch möglich sein, dass ungewollt mehrfach geschrieben wird, was zu Duplikaten führen kann.

→ Mindestens-einmal-Semantik ist besser geeignet, um sicherzustellen, dass Daten korrekt geschrieben werden.

**b. Ein Programm kompilieren**

→ Mindestens-einmal-Semantik ist besser geeignet, um sicherzustellen, dass der Kompilierungsprozess abgeschlossen wird.

**c. Online-Banking**

→ höchstens einmal -> Beispiel bei einer Transaktion, die nicht mehrfach ausgeführt werden sollte

→ Eine genau-einmal-Semantik (idealerweise) ist notwendig, aber wenn man sich zwischen Höchstens-einmal- und Mindestens-einmal-Semantik entscheiden muss, sollte man die Anforderungen an Konsistenz und Fehlerbehandlung besonders berücksichtigen

**14. Warum kann bei einem Zwei-Phasen-Commit-Protokoll das Blockieren nie vollständig eliminiert werden, selbst wenn die Teilnehmer einen neuen Koordinator wählen?**

→ Wir haben keine Garantie, dass ein Client die Rolle des Koordinators übernimmt. z.B. Client übernimmt Koordinatorrolle und stürzt aber ab.

→ Das Zwei-Phasen-Commit-Protokoll kann aufgrund seiner Abhängigkeit von der Erreichbarkeit und Synchronisation aller Teilnehmer und des Koordinators niemals vollständig garantieren, dass Blockieren eliminiert wird. Das Problem des Blockierens tritt auf, wenn entweder der Koordinator oder einer oder mehrere Teilnehmer ausfallen oder nicht korrekt miteinander kommunizieren. Auch durch die Wahl eines neuen Koordinators kann die zugrunde liegende Problematik der unvollständigen oder inkonsistenten Informationen nicht vollständig gelöst werden, was dazu führt, dass Blockieren weiterhin möglich ist.

**15. Kann TMR für fünf (bzw. x) Elemente pro Gruppe statt für drei verallgemeinert werden? Wenn ja, welche Eigenschaften hat es?**

→ System hätte größere Robustheit (es können mehr Elemente ausfallen) bei größerem Aufwand

→ Es können immer k Geräte ausfallen für 2k+1 Geräte im System

→ Ja, TMR kann auf eine beliebige Anzahl $x$ von Elementen pro Gruppe verallgemeinert werden, und diese Verallgemeinerung wird als N-Modular Redundancy (NMR) bezeichnet. NMR bietet eine skalierbare Methode zur Erhöhung der Fehlertoleranz durch den Einsatz von Mehrheitsabstimmung, wobei die Fehlertoleranz proportional zur Anzahl der Module und deren Fehlertoleranz ist. Es erfordert jedoch zusätzliche Hardware und kann in Bezug auf Kosten und Komplexität aufwändiger sein.

**16. Muss ein zustandsloser Server Kontrollpunkte aufzeichnen?**

→ In der Regel ist es nicht erforderlich, dass ein zustandsloser Server Kontrollpunkte aufzeichnet. Die Natur eines zustandslosen Servers basiert auf der Annahme, dass jede Anfrage unabhängig und ohne Kontext zu vorherigen Anfragen verarbeitet wird. Das Fehlen eines Zustands zwischen Anfragen eliminiert die Notwendigkeit für Kontrollpunkte, da keine konsistenten oder wiederherstellbaren Zustände erforderlich sind.

**17. In unserer Erklärung des Drei-Phasen-Commit-Protokolls scheint es, dass das Festschreiben einer Transaktion mit Commit auf einer Mehrheitsabstimmung basiert. Stimmt das?**

→ Das Drei-Phasen-Commit-Protokoll (3PC) basiert nicht auf einer Mehrheitsabstimmung, sondern auf einem strukturierten, sequenziellen Ablauf von Phasen, der sicherstellt, dass alle Teilnehmer die Transaktion konsistent und sicher abschließen können. Die Verwendung von Pre-Commit-Phasen und der explizite Commit-Befehl sorgen dafür, dass die Transaktion nur dann endgültig abgeschlossen wird, wenn alle Teilnehmer in den Prozess eingebunden sind und keine Unsicherheit besteht.