1. Geben Sie eine charakterisierende Definition für "Verteiltes System" an. Nennen Sie die wichtigsten Design-Ziele bzw. charakteristischen Eigenschaften von Verteilten Systemen. Stellen Sie weiters den Zusammenhang zu den typischen Fallstricken beim Entwurf verteilter Systeme her.

Definition

Unabhängige verbundene Computer => wirken wie EIN System

Ziele

- Resource sharing: gemeinsames Nutzen von Resourcen
- Transparency: interne Strukturen verbergen
- Openness: Standard Protocols, Portability
- Scalability: Anpassungsfähigkeit an neue Anforderungen
- Concurrency: echte Parallelität
- Fault Tolerance: Fehler verbergen

Eigenschaften

- Einzelen Autonome Komponenten
- Verhält sich wie ein System
- unterschiedliche HW und SW ist möglich (Middleware kann damit umgehen)
- Erweiterbarkeit
- Skalierbarkeit
- trotz Fehler einzelner Komponente Verfügbar

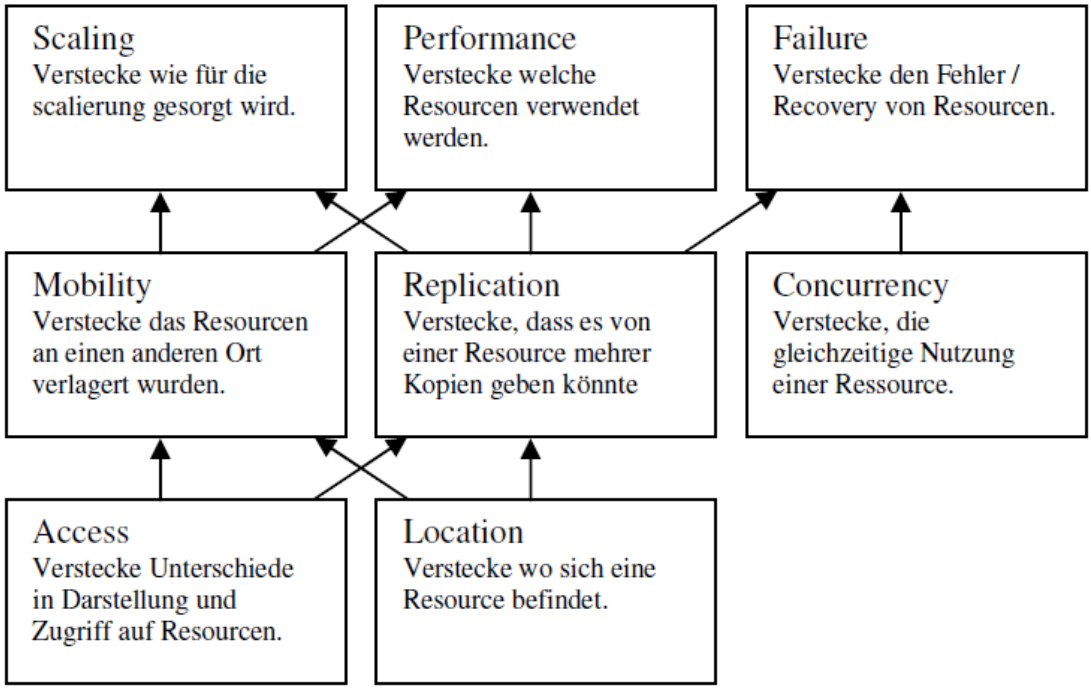
Fallstricke

- Netzwerk ist zuverlässig
- Latenzzeit ist null
- Bandbreite umbegrenzt
- Netzwerk is sicher
- Topologie ändert sich nicht
- Nur ein Administrator
- Keine Übertragungskosten
- Netzwerk ist homogen

2. Beschreiben Sie die standardisierten Arten von Transparenz und erklären Sie den Zusammenhang zischen den einzelnen Transparenz-Definitionen.

Es soll versteckt werden, das es sich um ein verteiltes System handelt.

Transparenzen



Nachteile

- Hoher Aufwand für die Maskierung von Fehlern
- Performance Transparency schwierig: manches dauert einfach länger
- Location Transparency: z.B.: Ausdrucken!
- Evaluierung welche Transparenzen für die jeweilige Aufgabe wichtig
- Manchmal besser ohne Transparenz zu arbeiten - z.B.: bei Druckern (User informieren, dass es ein verteiltes System ist)

3. Was versteht man unter "Openness"?

Openness sind folgende Eigenschaften

- Dienste haben definierte Schnittstellen und diese sind dokumentiert
- Formate sind genau spezifiziert (z.B.: Datumsformat)
- Modular aufgebaut (Austausch einzelner Komponenten)

Schnittstellen sollen vollständig und implementierungsneutral beschrieben warden
Trennung Policy und Mechanismus (Rendern bzw. Cachen im Browser)

4. Erläutern Sie Probleme und Lösungsansätze für "Scalability".

Scalability ist die Fähigkeit mit geänderten Anforderungen umgehen zu können.

Anforderungen

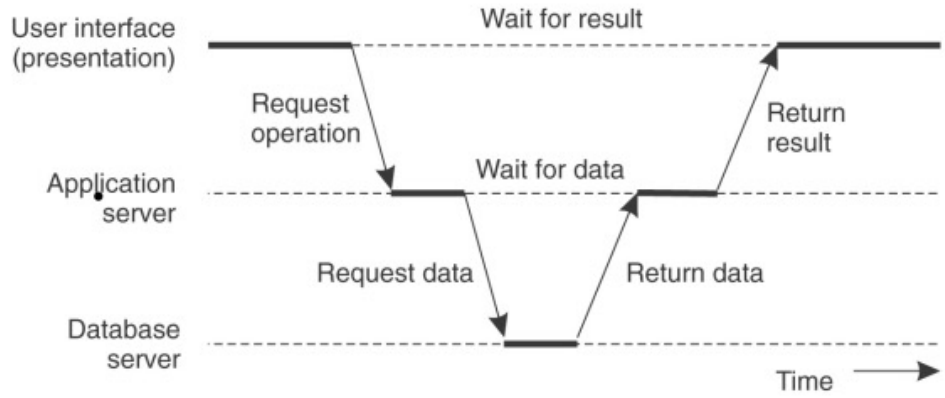
- Größe: z.B.: mehr User müssen Zugreifen/mehr Daten verwaltet werden
  Zentrale Alorithmen stossen an ihre Grenzen
- Geographische Ausdehnung: steigende Latzenzzeiten
- Administrativ: mehrere Organisationen nutzen das System (Achtung: andere Anforderungen)

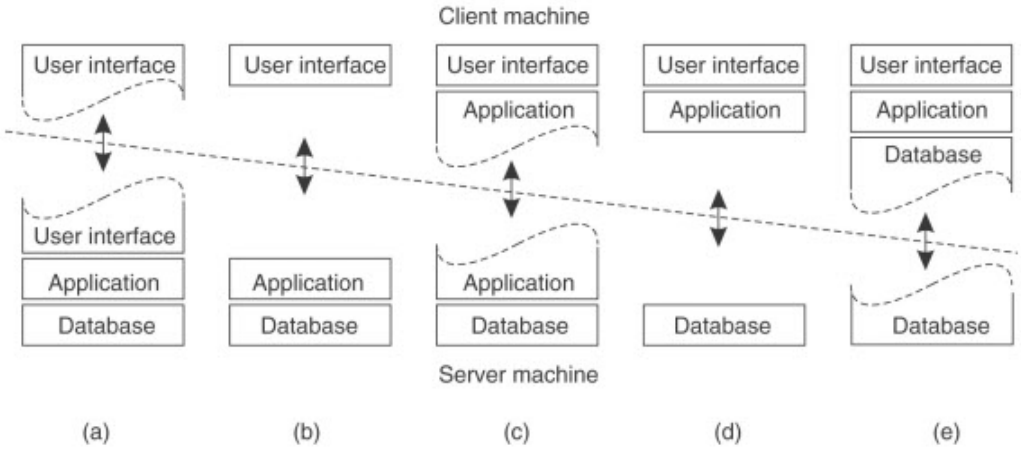
Lösungen

- Replikation
- Asynchrone Kommunikation (verstecken der Latzenzzeit)
- Hierarchien (Verteilung der Funktionalität)

5. Was versteht man unter der vertikalen Verteilung bzw. N-Schichten-Systemen? Diskutieren Sie dabei alle Grundvarianten von Client/Server-Systemen. Ist folglich ein Java Applet eher ein Thick Client oder ein Thin Client?

Aufteilung eines Gesamtsystems in einzelne Schichten (z.B.: OSI ISO Model). In der Anwendungsentwicklung oft 3 tier Systeme (UI, Logic, Data)! Funktionsabgrenzung und fixe Schnittstellen!





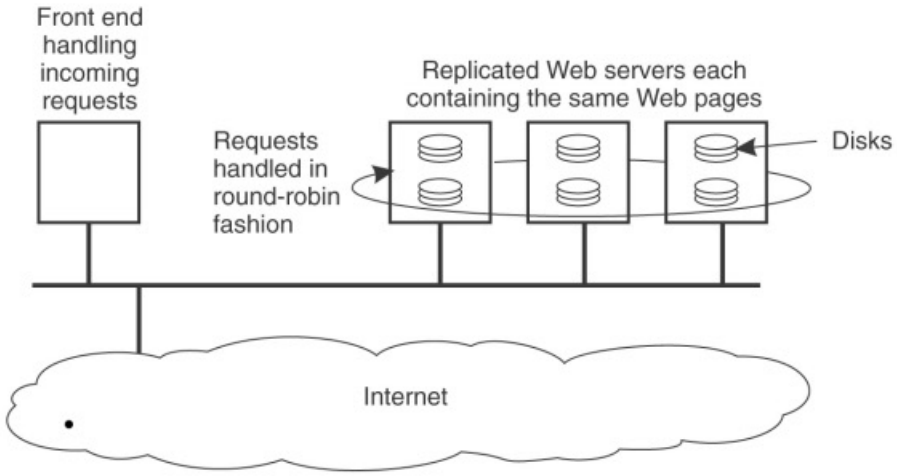
Java Applet abhängig von der Programmierung aber oft c oder d.

6. Was ist horizontale Verteilung? Mit welchen grundlegenden Design-Fragen müssen Sie sich beim Entwurf der horizontalen Verteilung eines Systems beschäftigen? Gibt es einen Zusammenhang zur vertikalen Verteilung?

Horizontale Verteilung ist die Aufteilung einer vertikalen Schicht auf mehrere Prozesse. (Aufteilung der Gesamtlast!). Jeder arbeitet nur auf einem Teil der volständigen Datenmenge.

Design Fragen

- Wann werden Replikate erstellt
- Wie wird Last aufgeteilt (z.B.: Round Robin)
- Haben einige Komponenten spezielle Anforderungen (z.B.: an die HW)
- Sollen einige Komponenten immer nahe zueinander liegen
- Größe der Komponenten (viele => große Flexibilität aber viel Kommunikation; wenige => weniger Kommunikation aber weniger Flexibilität)



7. Beschreiben Sie das ISO-OSI Modell der geschichteten Protokolle (Grundprinzip). Stellen Sie den Bezug zu den Internet-Protokollen (TCP/IP) her. Warum sind Transport-Layer Protokolle für Verteilte Systeme oft nicht ausreichend?

DIESE FRAGE WURDE NOCH NIE GESTELLT! – ICH GLAUB DAS BLEIBT AUCH SO!

8. Was ist Middleware? Welche Anforderungen stellt man an Middleware? Welche Services soll Middleware bieten? Erläutern Sie den Zusammenhang zwischen Middleware und Architectural styles.

Middleware is about reuse! Sie stellt oft benötigte Funktionalität in einem konfigurierbaren Standardpaket zur Verfügung. Sie ist zwischen dem Transport und dem Application Layer einzuordnen!
Desweiteren stellt sie die Transparenzen zur Verfügung. Es muss die richtige Middleware gewählt werden.

Anforderungen

- Transparenz
- Offenheit
- Skalierbarkeit
- Wartbarkeit
- Life-Cycle Management für Komponenten
- Interface Definition
- Kommunikationsgrundlage
- Concurrency Library

Services

- Naming
- Persistency
- Verteilte Transaktionen
- Security
- Replikation
- Zugriffstransparenz

Styles

- Layered
- Object based (Component based)
- Data based
- Event based
- Combinations

9. Wie kann man die Flexibilität der Middleware erhöhen sowie die Zusammenarbeit von Middleware und Anwendung effizienter gestalten? Erläutern Sie dabei die Grundprinzipien von Interceptoren, Adaptivität und Self-Management.

Middleware Systeme bieten Anpassungsmöglichkeiten. Damit ist die Anpassung an die eigenen Anforderungen möglich! => Flexibel und die bessere Nutzbarkeit.

Interceptors

Unterbrechen den normalen Programmcode und erlauben dem Applikationsentwickler eigenen Code hinzuzufügen. (z.B.: alles vor dem senden zu verschlüsseln)

Adaptivität

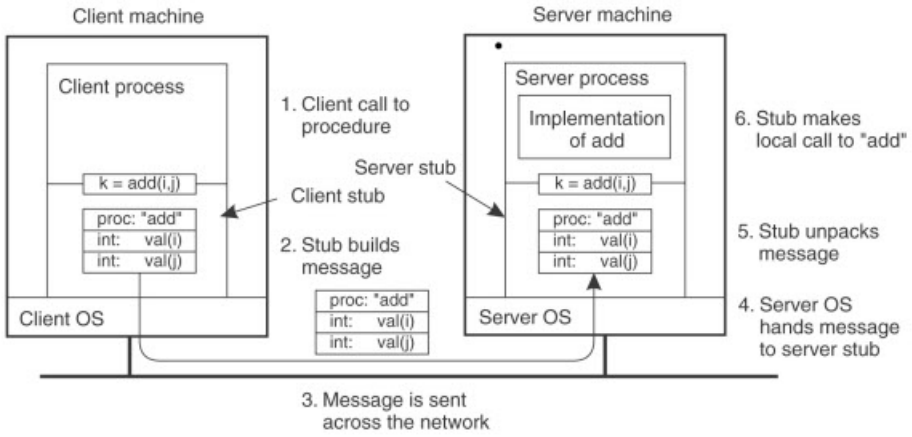
- Seperation of Concern: Trennung der Aufgaben
- Computational reflection: Selbstüberprüfung und Anpassbarkeit
- Modularer Aufbau (Component-based Design); Austausch zur Laufzeit

Self Management

Feedback Control Loops: Systeme überwachen sich selbst und können gewisse Parameter selbst anpassen (z.B.: wie lange werden Clients über Änderungen informiert (Push))

10. Erläutern Sie das Grundprinzip des Remote Procedure Call. Gehen Sie auf die Begriffe "client stub" und "server stub" näher ein.

RPCs sind entfernte Funktionsaufrufe und für den Entwickler grundsätzlich transparent! (Achtung: Latenz!)



Client Stub

Wird wie ein lokaler Funktionsaufruf programmiert. Wird durch den IDL Compiler zur Verfügung gestellt

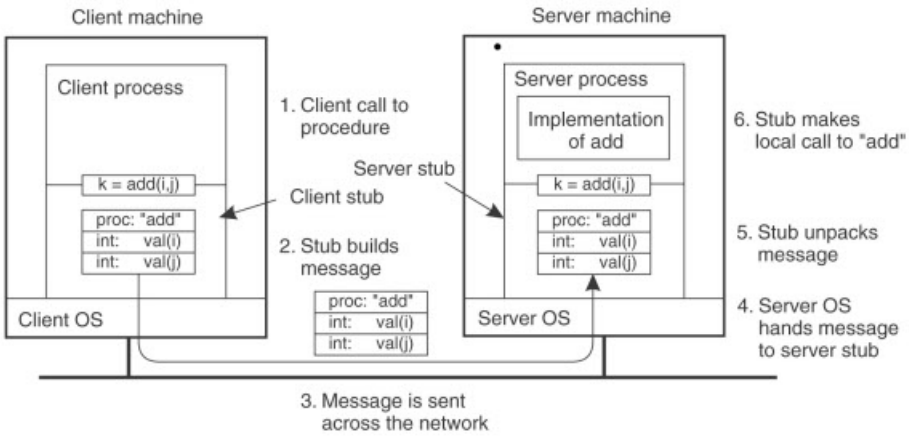
Server Stub

Dieser führt die jeweiligen Calls auf die richtige Funktion durch. (durch IDL Compiler)

Marshaling

- Serialize
- ??? (es waren 4 Punkte)

11. Wie können Variablen bei Prozedur-Aufrufen grundsätzlich übergeben werden? Wie werden Sie bei RPC gehandhabt und welche Probleme gibt es dabei?



Lokale Aufrufe

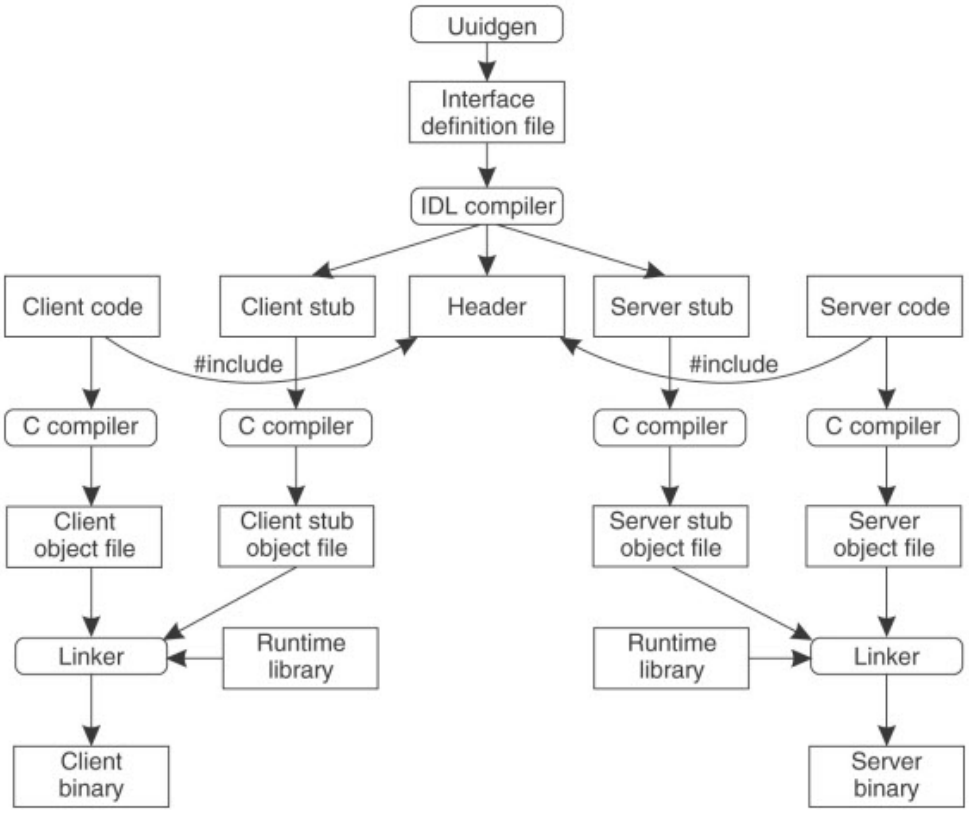
- Call by Value (z.B.: int)
- Call by Reference (z.B.: Pointer)
- Call by Copy/Restore (z.B.: Object wird Serialisiert; geändert; zurückgeschrieben)

Remote Aufrufe

Es kann eigentlich nur Call by Value und Copy/Restore verwendet werden.
Folgendes ist zu bedenken

- Anpassung der Typen (z.B.: Least Significant Bit; Most Significant Bit)
- Call by Reference macht oft keinen Sinn (locale Adresse nicht am Server verfügbar => Copy/Restore)
- Bei Copy und Restore: was wird alles übertragen bei großen Datenmengen

12. Wie schreibt man für RPCs Client und Server und welche Rolle spielt dabei die IDL? Was versteht man in diesem Zusammenhang unter "binding"?



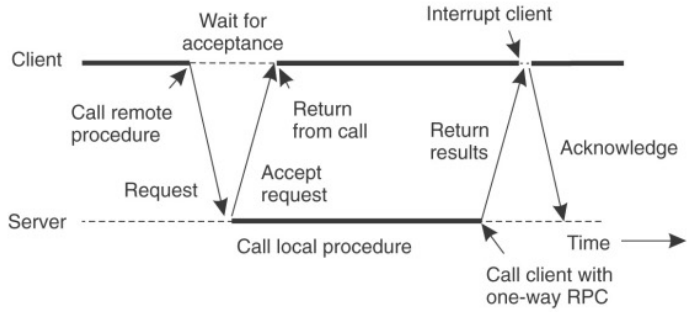
Binding:

- Über ein Naming Service wird der richtige Server für eine Funktion gesucht
- Möglicherweise DCE
- Auf diesen Server verbindet sich der Client Stub und kann dort RPC Calls ausführen
- Wichtig die eindeutige ID (uuidgen) => wird beim verbinden genutzt

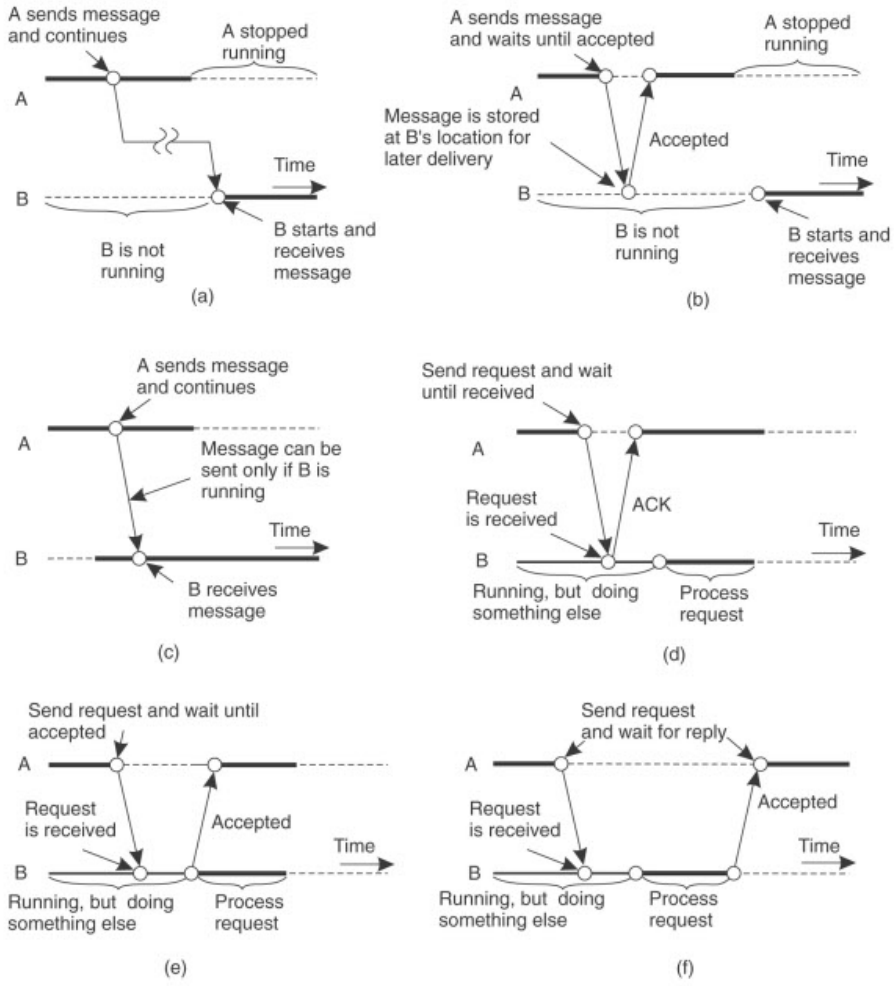
13. Welche Arten von asynchronen RPCs gibt es? Geben Sie auch einen verallgemeinerten Überblick über verschiedene Typen der Kommunikation (persistent/transient bzw. synchron/asynchron).

Arten des Antwortverhalten des Servers

- Server bestätigt erhalt der Nachricht
- Einweg RCP: Client fährt nach Absendung der Nachricht sofort weiter
- Verzögerter synchroner RCP (siehe Grafik)

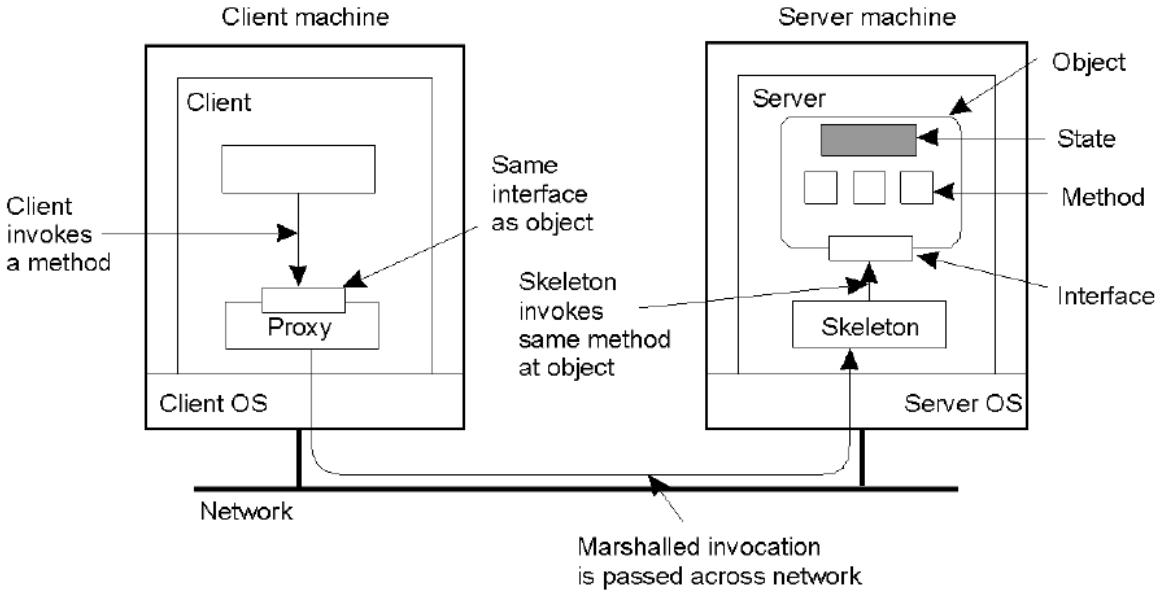


Typen der Kommunikation



(a) Persistente asynchrone Kommunikation; (b) Persistente synchrone Kommunikation; (c) Transiente asynchrone Kommunikation; (d) Empfangsbasierte transiente synchrone Kommunikation; (e) Auslieferungsbasierte transiente synchrone Kommunikation; (f) Antwortbasierte transiente synchrone Kommunikation

14. Erläutern Sie die Grundprinzipien verteilter Objekte sowie der Remote Object (bzw. Method) Invocation. Gehen Sie auf die Begriffe "proxy" und "skeleton" ein.



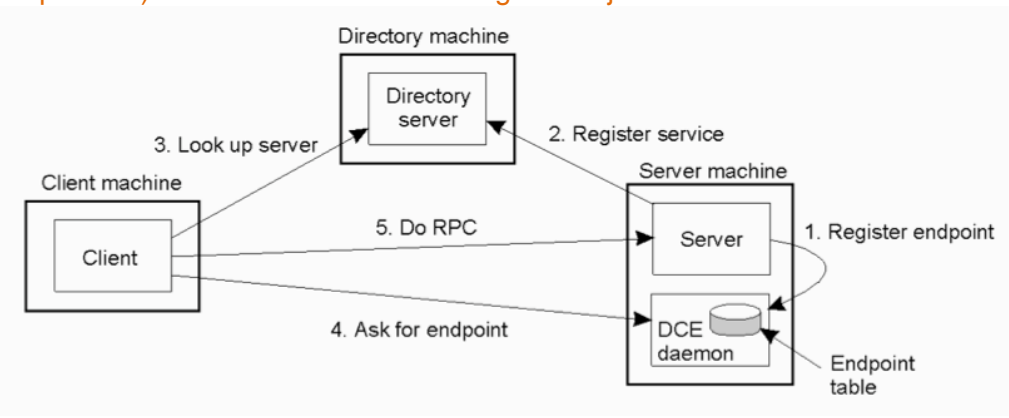
Proxy

Proxy ist die lokale Repräsentation des Serverobjekts am Client. (bei RPC Client Stub)

Skeleton

Der Skeleton leitet Aufrufe an das jeweilge Objekt weiter

15. Wie funktioniert das Binding bei RMI? Welchen Zusammenhang gibt es zu den verschiedenen Arten, eine object reference zu implementieren. Vergleichen Sie (exemplarisch) CORBA und Java in Bezug auf Objektreferenzen.



Object Referenzen

- Implizit: das Object wird automatisch bei einer Nutzung gebunden
- Explizit: das Object muss vor der ersten Nutzung explizit gebunden werden
- Local References: nur local gültige Referencen
- Global References: sind im gesamten System gültig und weisen auf das selbe Object
- In Java: keine Probleme mit unterschiedlicher HW (z.B.: Little/Big Endian) weil JVM
- Corba: IOR (Interoperable Object References): neutral Objectreference (nun auch z.b.: Java -> C++)
- Binding benötigt immer einen Identifiert (z.B.: IP, Port, Object ID)

16. Was versteht man unter "static" und "dynamic" RMI?

Static

Eine Schnittstellendefinition ist zur Compiletime vorhanden. Dadurch kann auch direct auf das Objekt zugegriffen werden.

rmiObj.doSomething(param1,param2);

Dynamic

Hier ist die Schnittstellendefinition erst zur Laufzeit bekannt. Es muss über eine Invoke Methode gearbeitet werden.

Invoke(rmiObj,id(doSomething), param1, param2);

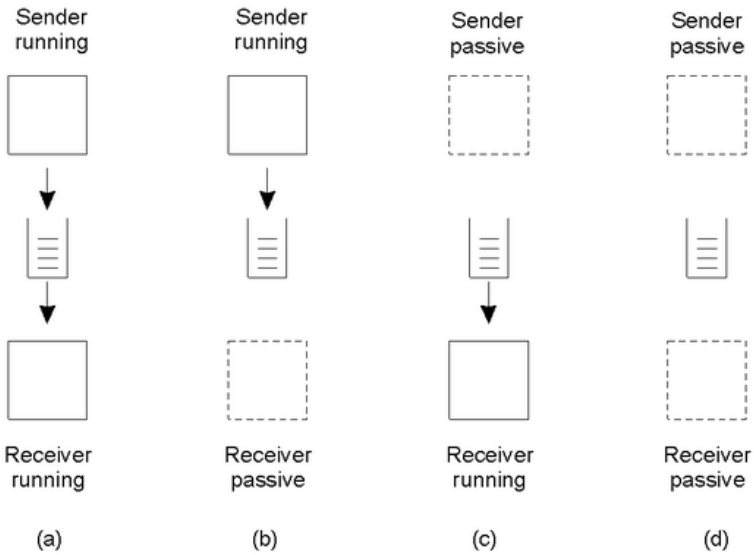
Anwendung: Object Browser, Runtime Inspection

17. Wie funktioniert die Parameterübergabe bei RMI? Gehen Sie dabei auf jene Eigenschaften der Objektorientierung ein, welche den Vorteil von RMI gegenüber RPC bewirken.

Grundsätzlich funktioniert es wie bei RPC.
Der große Vorteil bei RMI ist, dass es global gültige Referencen auf Objekte gibt. Dadurch können diese, als reine Reference übergeben werden! D.h. es muss hier nicht mit copy/restore gearbeitet werden.

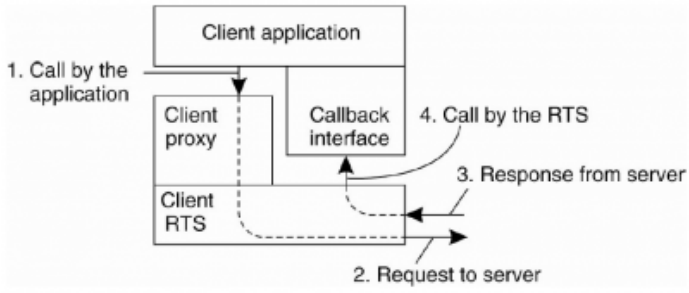
18. Erläutern Sie die Grundprinzipien von Message-orientierter Kommunikation und gehen Sie auf CORBA Messaging exemplarisch ein.

Es werden Nachrichten ausgetauscht (persistent/transparent und synchron/asynchrony). Es gibt keine Garantie, das eine Nachricht jemans gelesen wird.



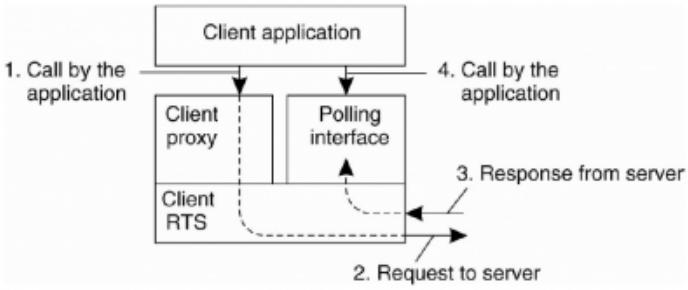
CORBA unterstützt dabei 2 Arten von asynchronen Aufrufen

Callback



CORBA's callback model for asynchronous method invocation.

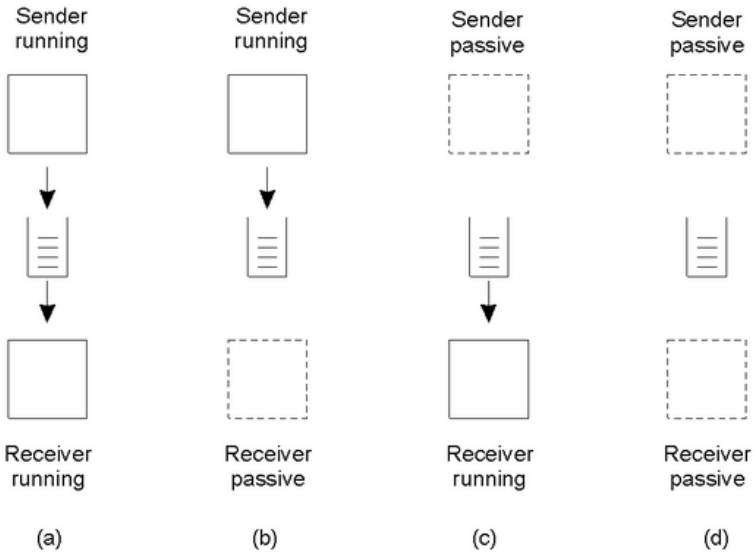
Polling

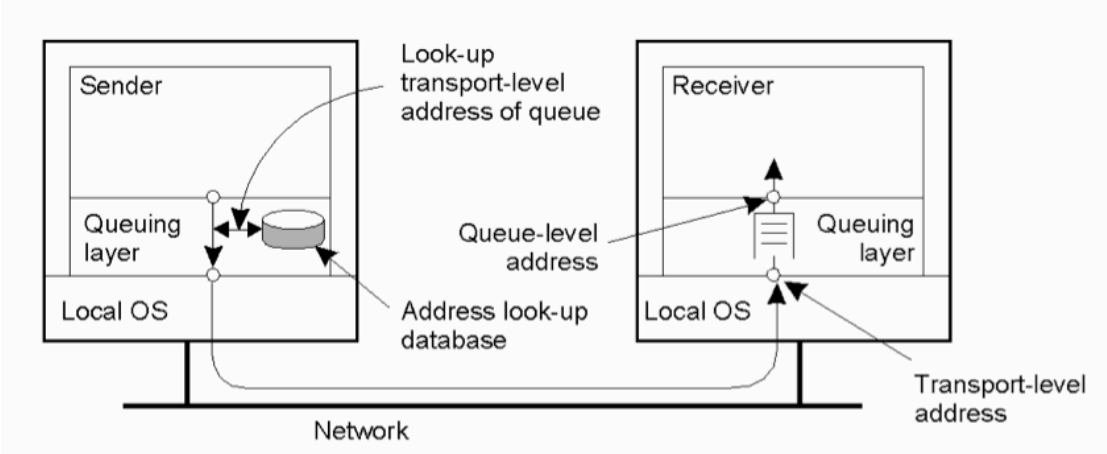


CORBA's polling model for asynchronous method invocation.

19. Was versteht man unter "Message-oriented Middleware MoM"? Erläutern Sie Modell und Architektur solcher "Message-Queueing"-Systeme. Gehen Sie insbesonders auf den Begriff des Message Brokers und dessen Bedeutung für EAI ein.

Message Oriented Middleware ist, Middleware die Nachrichten in Form von “Messages” teilt. Jede Nachricht wird dabei in eine lokale Queue eingeordnet und an die jeweilige Queue beim Empfänger übertragen. Es gibt keine Sicherheit, dass die Nachricht jemals bearbeitet wird.





Get: blockt solange die Queue leer ist, gibt die erste Nachricht zurück
Put: ordnet eine Nachricht in die Queue ein
Poll: testet ob die Queue leer ist
Notify: installiert einen Handler

Message Brocker werden eingesetzt um die Enterprise Application Integration (EAI) zu unterstützten. Hier werden sie oft eingesetzt, um Daten auf die jeweilige Applikation anzupassen. Ausserdem kann er als “Router” für Messages im Public/Subscribe Verfahren eingesetzt werden.

20. Wiederholen Sie den Unterschied zwischen Prozess und Thread. Welche Bedeutung haben Threads in verteilten Systemen, insbesonders in Client/Server-Umgebungen?

Prozess

Wird auf eigenem Speicherbereich und eigenem “virtuellen” Prozessor ausgeführt. Dadurch kann ein Prozess keinen anderen Behindern. Das erstellen von Prozessen ist jedoch relative Resourcen intensive.

Thread

Threads laufen innerhalb eines Prozesses und können somit auch den Speicherbereich anderer Threads lesen/schreiben. Sie sind günstiger in der Erstellung! Falls ein Thread blockiert, laufen die anderen auch weiter (wichtig in der Serverprogrammierung), sofern Kernel Level Threads (relative teuer) oder LWPs (Lightweigh Processes) verwendet werden.

Multithreading hat viele Vorteile:

- Einfachere Implementierung (divide & conquere): jeder muss nur mehr ein Teilproblem lösen
- Höhere Durchsatz (Server)
- Höhere Transparenz (Client)

Folgendes ist jedoch zu bedenken

- Abschierung der Threads untereinander
- Performance durch zuviele Context Switches
- Locks auf Resourcen

21. Welche Aspekte Verteilter Systeme sind auf Client-Seite zu berücksichtigen? Wie werden User Interfaces in die Architektur Verteilter Systeme eingebunden? Wie können dabei verschiedene Arten der Transparenz unterstützt werden?

Wichtige Aspekte

- Client muss mit User und Verteiltem System gleichzeitig kommunizieren (Multithreading)
- Wie wird mit Kommunikationsfehlern umgegangen (anderen Replika; User informieren)
- Interpretation der Daten am Client/Server
- Wie wird der Client aktualisiert (z.B.: Applikation oder Java Applet)

UIs

- vom dump client bis zum fat client alles möglich
- Modern trennung laut MVC Modell
- Immer weniger verschiedene UI Modelle => HTML Interfaces
- Lastverteilung zwischen Server/Client (z.B.: Validierung der Daten)

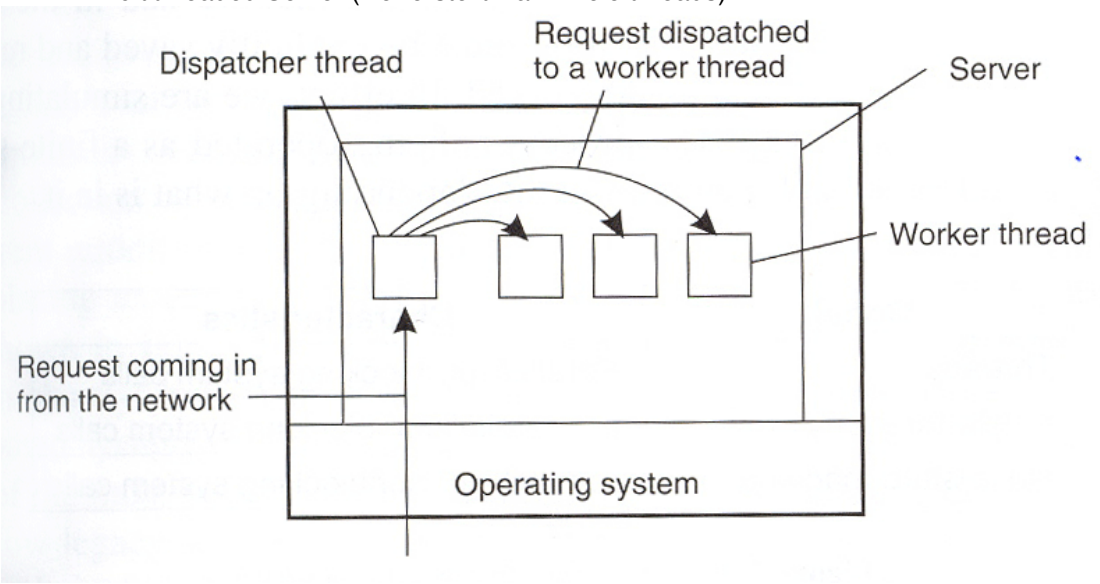
Arten der Transparenz

- Zugriffstransparenz durch Client Stub oder Proxy
- Orts, Migration und Relokationtrasparenz durch Naming Services
- Fehlertoleranz durch Serverwechsel/Retrys

22. Geben Sie grundlegende Design-Entscheidungen für Server an und bewerten Sie diese.

Design Entscheidungen

- Singlethreaded Server
- Finite state Machine (nimmt anfrage entgegen, bearbeitet wenn zeit, antwort später)
- Multithreaded Server (wer erstellt wann viele threads)



Interruptbehandlung

- Server Interrupt: einfaches disconnect
- Out of Band Controller: z.B.: FTP Control

Infos des Servers über seine Clients

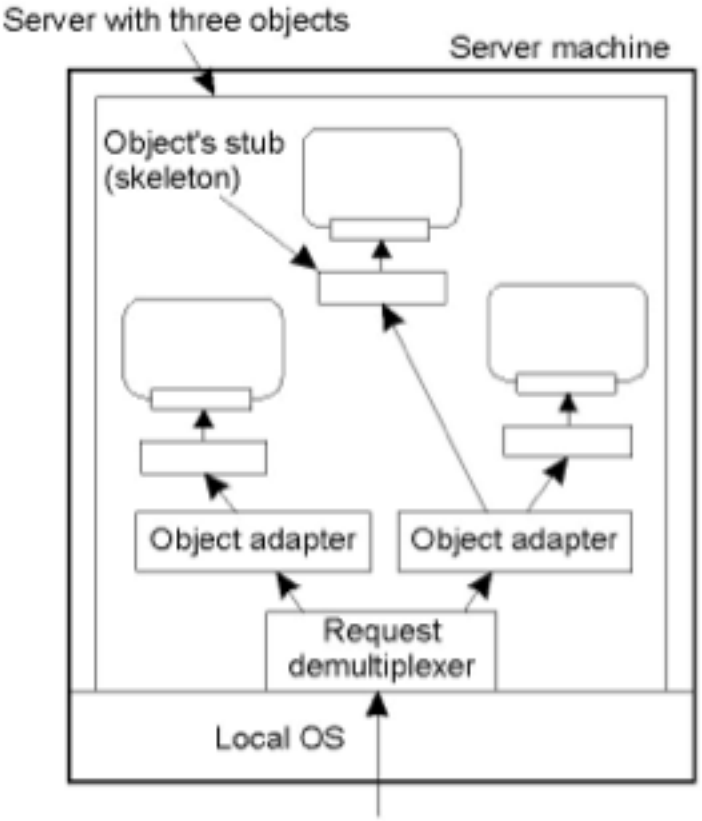
- Stateless: Server hat keine Infos über seine Clients; jede Anfrage ist neu (z.B.: HTTP)
- Stateful: Der Server kennt alle Informationen über den Client (aktueller Status). Bei Absturzt: Recovery!
- Soft state: wie Stateful, aber Infos werden verworfen
- Session State: Aktionen werden innerhalb einer Session gespeichert; Ablaufdatum!

23. Was sind die Besonderheiten von Objekt-Servern? Welche Arten gibt es dabei für die "Invocation", also den Aufruf (evtl. auch die Aktivierung) eines Objektes auf Server-Seite? Was ist in diesem Zusammenhang ein Objekt-Adapter? Object Server stellen keine Dienste zur Verfügung sondern hosten Objekte.

Object Adapter

Object Adapter sind generische Policys in denen konfiguriert wird, wie ein Objekt verwaltet werden soll.

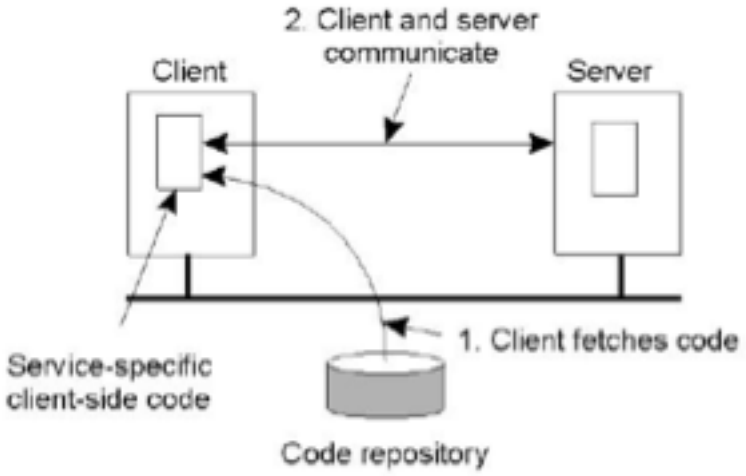
- Wann wird es Instanziert (bei Anfragen oder beim Server start)
- Wie lange bleibt es geladen?
- Wird es gespeichert oder nicht (persistent oder transient)
- Code sharing?
- Wird ein Thread per Object oder eine Queue angelegt



24. Erläutern Sie die wichtigsten Aspekte der Code Migration.

Umzug von (laufenden) Programmen (meistens wegen Performance Gründen).

- Auch der Umzug von Code am Client auf den Server (näher an den Daten; Performance) ➔ Weniger Kommunikationsoverhead.
- Oder der Download von z.B.: einem Videocodec vom Server auf den Client



Sender initiated migration

Wird vom System gestartet, wo der Code gerade ist (File Uploads)

Reveiver initiated migration

Wird vom System gestartet, wo der Code benötigt wird (z.B.: Java Applet)

Prozess besteht aus

- Code
- Resourcen (z.b.: Files)
- Execution (Variablen, CPU Context, …)

Mobility

- Weak Mobility
  Hier wird das Codesegment & Initialisierungsdaten übertragen (z.B.: Java Applet)
- Strong Mobility
  Hier wird auch das Execution Segment mitübertragen! Schwieriger zu implementieren!

25. Erläutern Sie das Konzept der Virtualisierung und in weiterer Folge deren Bedeutung für die Code Migration in heterogenen Umgebungen.

Es wird eine zusätzliche Schicht eingezogen, um die Eigenheiten der Hardware zu verstecken. Dies kommt vor allem der Code Migration zu gute, weil nun ein Code, auf jeder Hardware, für die die virtuelle Machine verfügbar ist, lauffähig ist.

Es gibt hier 2 unterschiedliche Verfahren (VMware/Java)

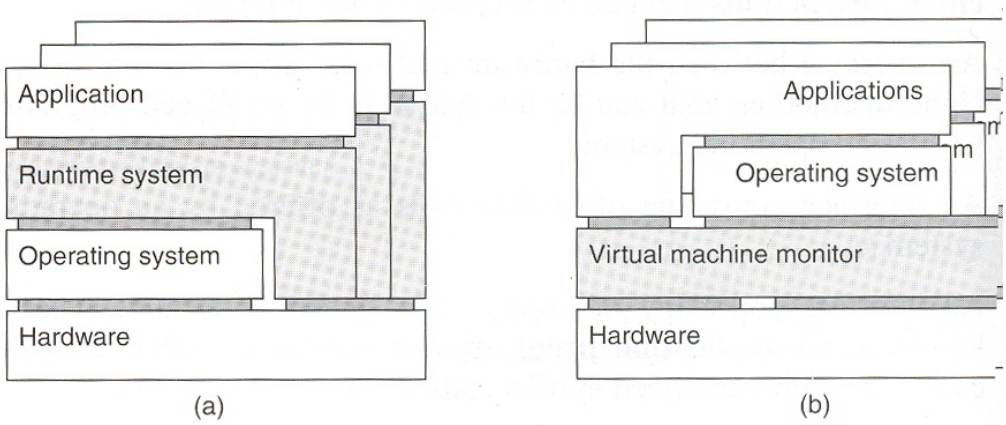


Figure 3-7. (a) A process virtual machine, with multiple instances of (application, runtime) combinations. (b) A virtual machine monitor, with multiple in-

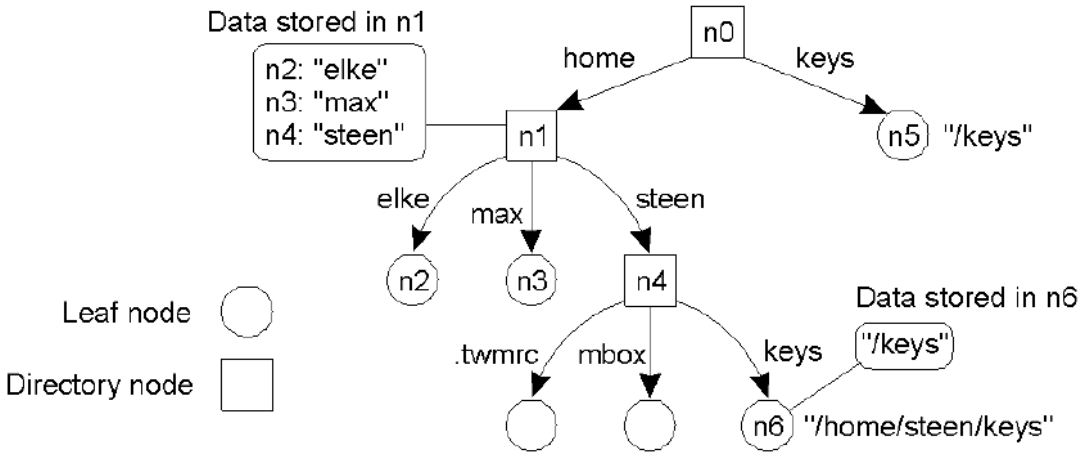
26. Erläutern Sie die Begriffe "Name", "Identifier", "Address" sowie den Bezug zwischen diesen Begriffen in der Praxis.

DIESE FRAGE WURDE NOCH NIE GESTELLT! – ICH HOFFE DAS BLEIBT AUCH SO!

27. Was ist ein "Name Space"? Erläutern Sie das Grundprinzip des "Closure Mechanismus" anhand eines Beispieles (zB Unix File System).

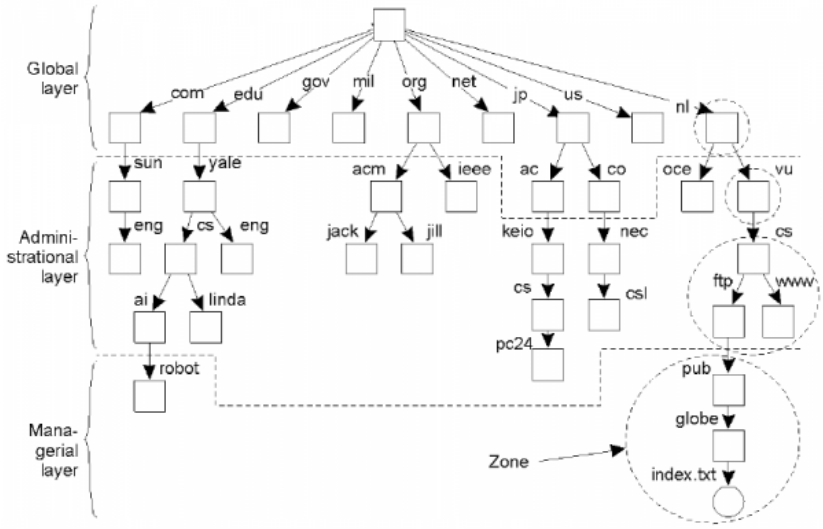
Namen bezeichnen Objekte in Verteilten Systemen. Zu einem Namen ist jedoch ein Kontext wichtig (Namensraum)

Beispiel: Dateiname keys bezeichnet die Datei nur wenn der Ordner /home/steen angegeben wird.



Closure Machanism: Externer Mechanismus um den Beginn zu finden. Bei Dateisystemen der Superblock, welcher an einer fix definierten Stelle der Festplatte steht. (Bei DNS die Root Server)

28. Erklären Sie die Schichten der Verteilung von Name Spaces und darauf aufbauend verschiedene (hierarchische) Möglichkeiten der iterativen/rekursiven "name resolution".



Global Layer

- Wenige Konten
- Sehr Stabil
- Sehr Schnelle Reaktionen
- Viele Replikas
- Effektives Caching

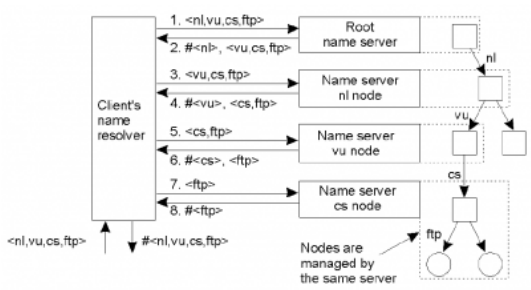
Administrational Layer

- mehr Konten
- mehr Änderungen
- schnelle Reaktionen
- wenige Replikas
- relative effektives Caching

Managerial Layer

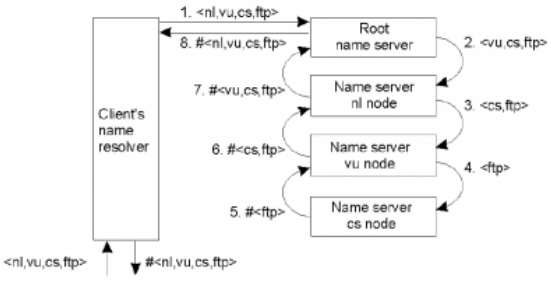
- viele Konten
- viele Änderungen
- eher langsame Reaktionen
- keine Replikas
- oft kein effektives Caching

iterative



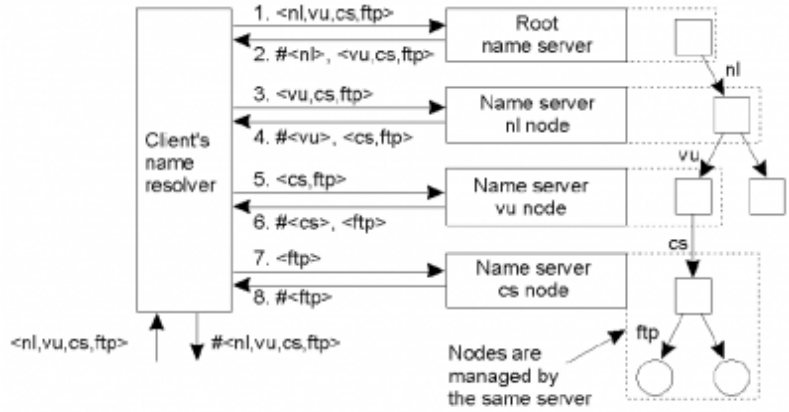
ftp://ftp.cs.vu.nl/pub/globe/index.html

recursive



ftp://ftp.cs.vu.nl/pub/globe/index.html

29. Erläutern Sie das Domain Name System DNS, sowie den Ablauf bei der Namens-Auflösung anhand der DNS Database (Resource Records). Was ist reverse lookup?



ftp://ftp.cs.vu.nl/pub/globe/index.html

Das DNS ist der Namespace der Internets. Es ist als hierarchisches System aufgebaut und skaliert bis heute!

Zone Files

Zonefiles sind die Konfigurationsdateien der DNS Server. Sie beinhalten alle Informationen (SOA, A, MX, NS, CNAME, PTR)

Beispiels

- Der Rechner X sucht in seiner Hosts-Datei, ob die IP-Adresse für „de.wikipedia.org“ dort hinterlegt ist. Falls dem nicht so ist, fragt er beim DNS-Server nach.
- Hat der DNS-Server von Rechner X eine IP-Adresse für den angefragten Namen zwischengespeichert, antwortet er damit und die Anfrage kommt zum Ende (siehe letzter Punkt). Andernfalls fragt er einen der 13 Root-Nameserver nach „de.wikipedia.org.“.
- Der Root-Nameserver findet heraus, dass die Auflösung dieses Namens in der „org.“-Zone weitergeht und sendet die Namen und die IP-Adressen der „org.“-Nameserver zum DNS-Server von Rechner X.
- Nun fragt der DNS-Server von Rechner X einen der Nameserver für „org.“-Domains nach „de.wikipedia.org.“.
- Der „org.“-Nameserver sendet ihm die Namen der Nameserver für die Zone „wikipedia.org.“.
- Anschließend fragt der DNS-Server von Rechner X einen „wikipedia.org.“-Nameserver wie die IP-Adresse des Namens "de.wikipedia.org." ist.
- Mit dieser Adresse wird an den DNS-Server von Rechner X geantwortet und der …
- … sendet sie an den Rechner X, welcher nun zum Beispiel seine HTTP-Anfragen an die IP-Adresse von „de.wikipedia.org.“ senden kann.

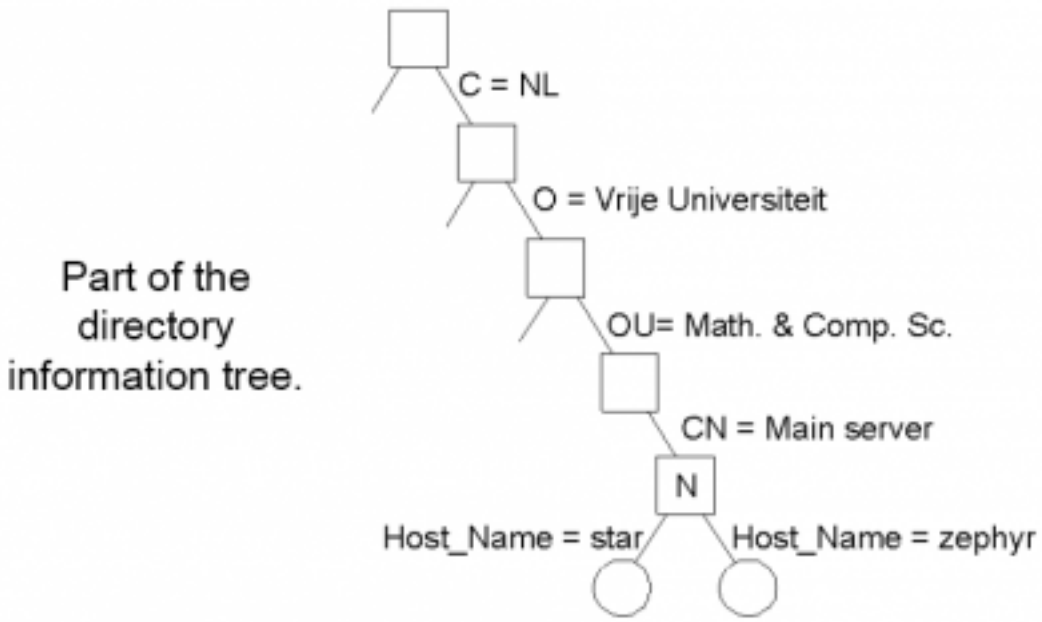
Reverse LookUps

Reverse LookUps funktionieren nach dem selben Prinzip. Die IP Adresse (192.168.0.12) wird dabei in umgekehrter Form inkl. des zusatzes in-addr.arpa an den DNS Server als Anfrage gesendet.

30. Was ist Directory Service bzw. "Attribute-based naming"? Beschreiben Sie den prinzipiellen Aufbau des X.500 Name Space sowie dessen LDAP Implementierung.

Jede Entität hat Eigenschaften über die es Identifiziert werden kann. Über diese Eigenschaften wird es auch in ein hierarchisches System eingeordnert. LDAP (implemetierung des OSI Modells) ist das Leightweight Directory Protocol und wird oft für die Benutzerverwaltung genutzt und kann über mehrere Server verteilt werden.

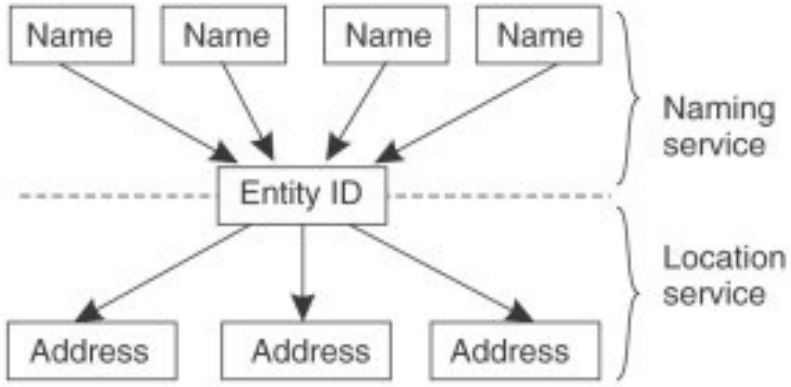
DIT=Directory Information Tree
DN = Distinguished Name



Innerhalb eines LDAP Trees können Entities über ihre Attribute gesucht werden.
Beispielsweise: search(“C=NL;O=*;OU=*Math*”)

31. Wie funktioniert in flachen Namensrämen das Location Service? Geben Sie das Grundprinzip möglicher Lösungen an und gehen Sie dabei auch auf die Begriffe Mobility und Discovery ein.

Location Service ist ein Mapping zwischen Namen und Adresse. Location Services werden genutzt um Resourcen zu registrieren bzw. zu finden. Sie unterscheiden sich zu Naming Services, da sie hier Einträge oft ändern.



Mögliche Lösungen

- ARP
- Multicast

Mobility

Mobility wird als Wechseln der jeweiligen Adresse bereichnet. Eine Entität wandert also weiter. Es gibt mehrere Möglichkeiten um sie dennoch zu Discovery (finden)

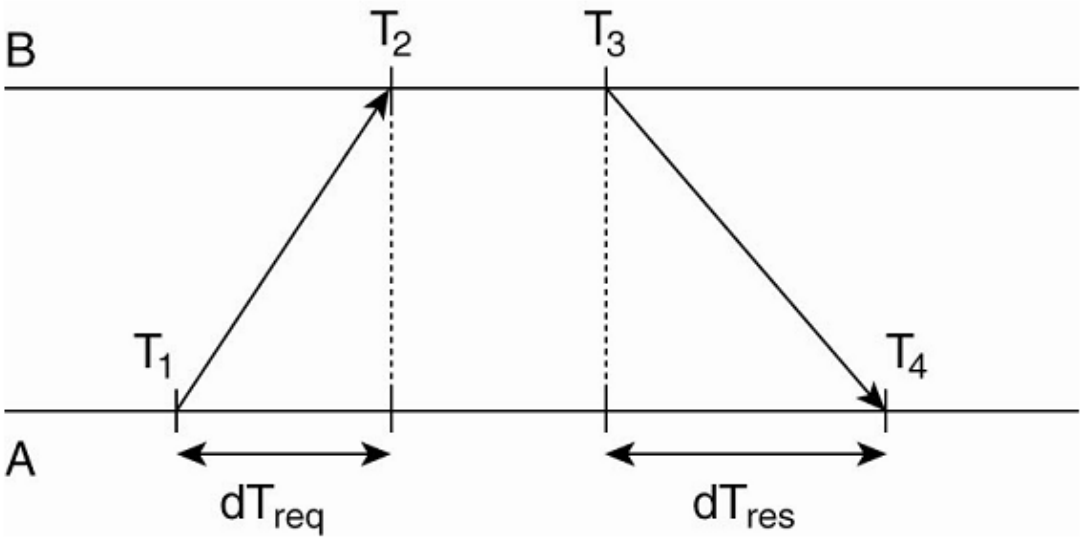
- Forwarding Pointer
  Sie hinterlässt immer einen Pointer zu ihrer nächsten Adresse (Achtung: broken Pointer)
- Homebased Approach
  Sie meldet ihre neue Adresse immer an ihre Home Adress

32. Wozu braucht man Uhrensynchronisation? Erläutern Sie das NTP und den Berkeley Algorithmus. Was ist die Problematik bei der Synchronisation von Physical Clocks?

In Verteilten Systemen muss man oft Ereignisse zeitlich einordnen. Dies kann beispielsweise mittels TimeStamps und Uhrensyncronisation erfolgen.
Uhrzeit darf nie zurückgestellt werden => Verlangsamung

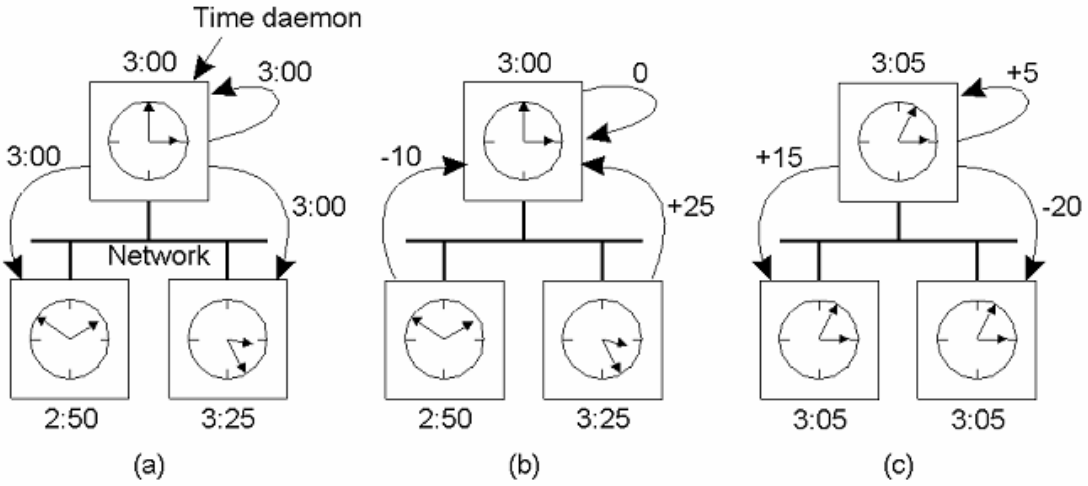
NTP

Genauigkeit wird über Stratum definiert (je kleiner desto genauer)
Anfrage wird 8x wiederholt => geringstes Delay wird genommen (wenn über Grenzwert verworfen)



Berkely Algorithmus

nur interne Sync; keine “richtige” Uhr aber dafür einheitlich



33. Was sind die Gründe für die Verwendung von Logical Clocks? Erklären Sie die Unterschiede zu den Physical Clocks. Was ist die "happened-before" Beziehung und wie funktionieren die "Lamport-Timestamps"?

Oft benötigt man keine wirkliche Uhrzeit, es reicht zu wissen was vorher passiert ist (happened before). Diese Reihung von happened before sind logical Clocks.

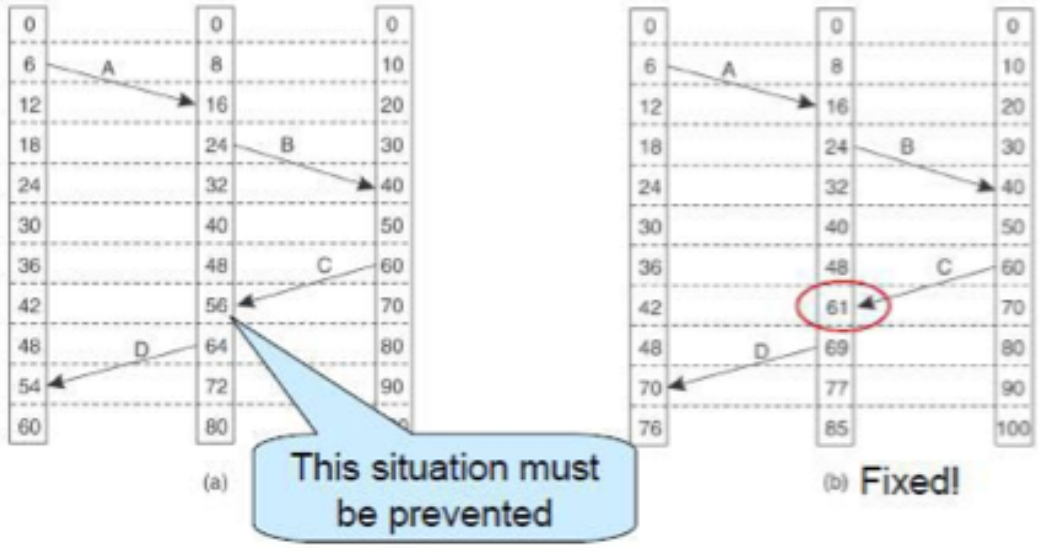
A ➔ B (A passierte vor B)

- z.B.: senden muss immer vor empfangen sind T(senden) ➔ T(empfangen)

A ➔ B ➔ C = A ➔ C

Laptort Timestamps

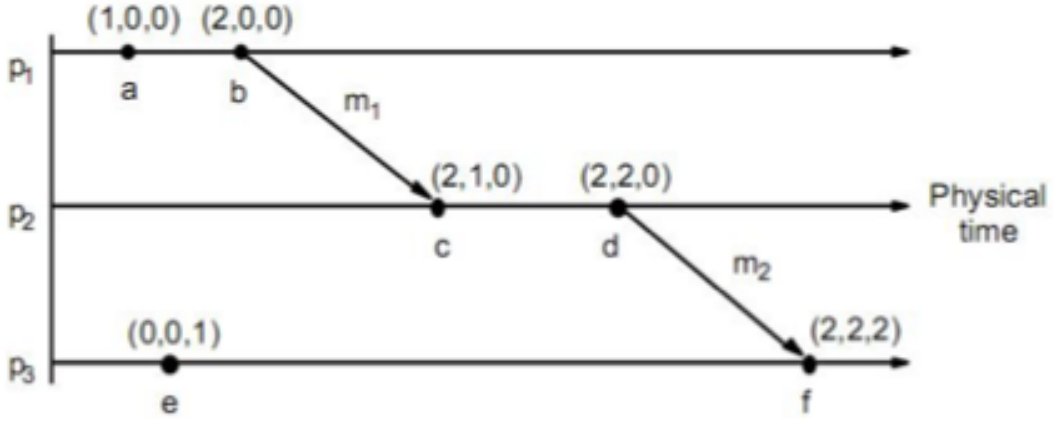
Jeder Prozess hat einen Counter. Dieser wird bei jeder Aktion erhöht (z.B.: Senden, Empfangen, …) und an jede Nachricht angehängt. Wenn er nun eine Nachricht empfängt überprüft er den Timestamp der Nachricht. Wenn dieser größer ist, als sein interner Counter, wird dieser durch den Timerstamp Wert ersetzt.



A➔B (A happened before)
A == B ➔ A hat nicht vor B stattgefunden (entweder gleich oder B vor A)

34. Welchen Nachteil haben die Lamport-Timestamps und wie kann dieser durch Vector-Timestamps überwunden werden?

Lamport Timestamps haben folgenden Nachteil:
T(A)<T(B) ➔ A hat NICHT vor B stattgefunden ➔ entweder gleichzeitig oder nachher ➔ sehr restrektives Ordering.
Bei Vektor Timestamps, wird ein Vektor mit dem Counter eines jeden bekannten Prozesses durch jeden Prozess geführt. Dadurch kann weit besser geordnet werden.



35. Wie funktioniert Distributed Mutual Exclusion. Wie verhalten sich verschiedene Algorithmen hinsichtlich Skalierbarkeit und Fehlertoleranz?

Explusiver Zugriff auf eine Resources. Kann nicht skalieren weil ja immer nur expluziver Zugriff!

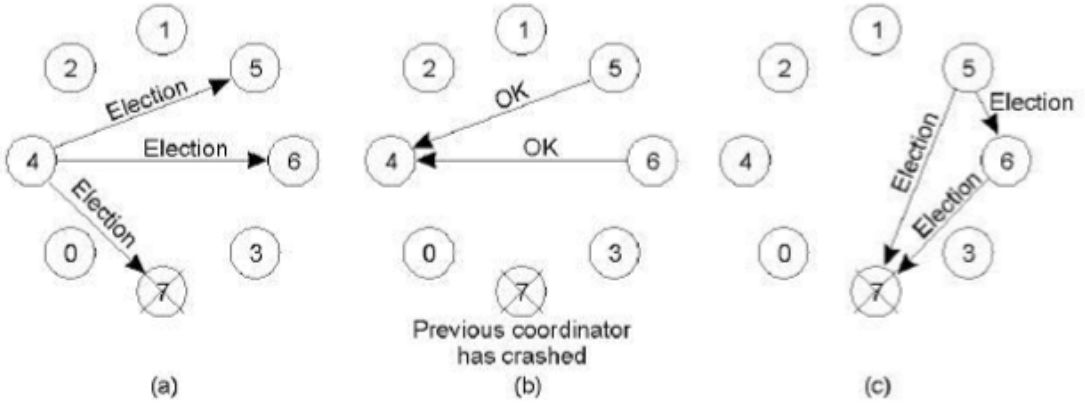
Algorithmen

- Token Ring: Es wird ein Token durchgegeben: wenn ihn hat darf auf der resource arbeiten, dann weitergeben. (keine Starvation weil geregelte Reihenfolge; keine Deadlocks; aber Problem wenn Prozess mit Token abstürzt; Token Weitergabe wenn keiner Zugriff möchte
- Centralized: Ein Koordinator (hohe Effizient; keine Starvation; bottleneck (aber egal weil sowieso nicht skalierbar); Single Point of Failure)
- Distributed ➔ bringt nichts!
    - Nachricht an alle ich hätte gerne die Resource
    - Wenn der halter sie gerade benötigt: Denied; sonst OK
    - Ordering mit Timestamps

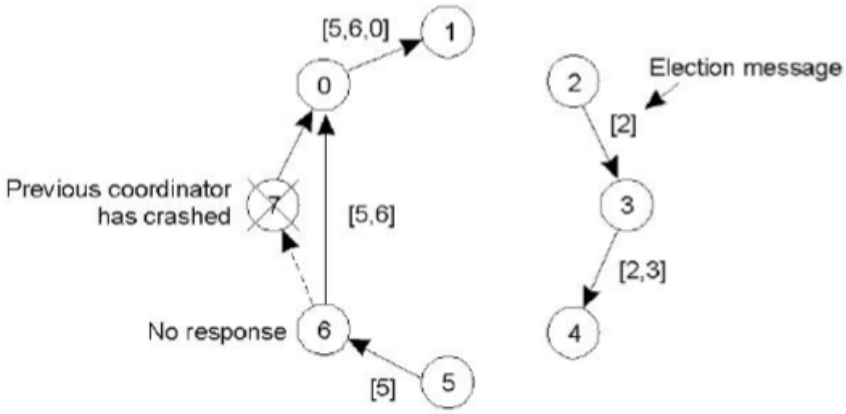
  Achtung: hoher Kommunikationsaufwand; n Points of Failure (wenn einer der Zugreifen möchte abstürzt)

36. Vergleichen Sie den "Bully" und den "Ring"-Algorithmus für Election hinsichtlich Fehlertoleranz. Warum sind diese Algorithmen für ad-hoc oder large-scale Systeme weniger geeignet und welche grundsätzlichen Lösungsansätze verfolgt man daher dort?

Bully



Ring



Ad Hoc

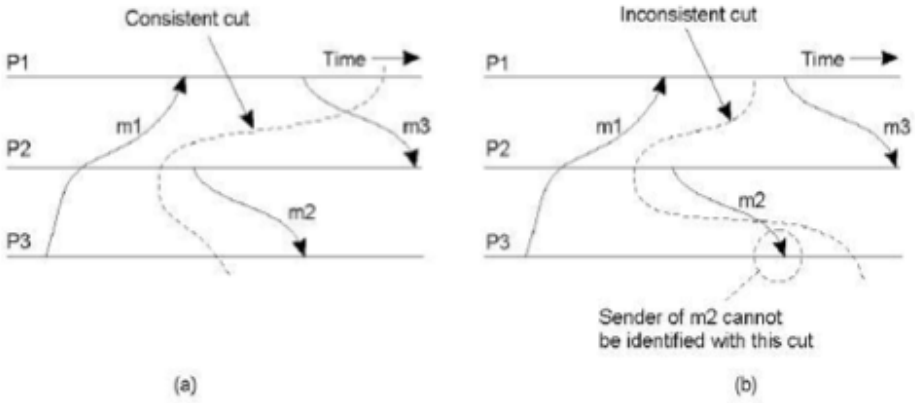
Tree Aufbau, wo jeder nur seinen Tree überwacht und den Besten an den jeweiligen Root weiterleitet; wenn oben angekommen => bester (anhand von Kriterien wie Bandbreite) Knoten gefunden

Large Scale Networks

Lösung über Superpeers. Es gibt eine definierte Anzahl an Superpeers welche aufgeteilt werden. Dazu werden Token im Netzwerk verteilt, welche sich abstrossen (Gossiping Protokol); wenn ein Knoten den Token lange genug halt ➔ Superpeer!

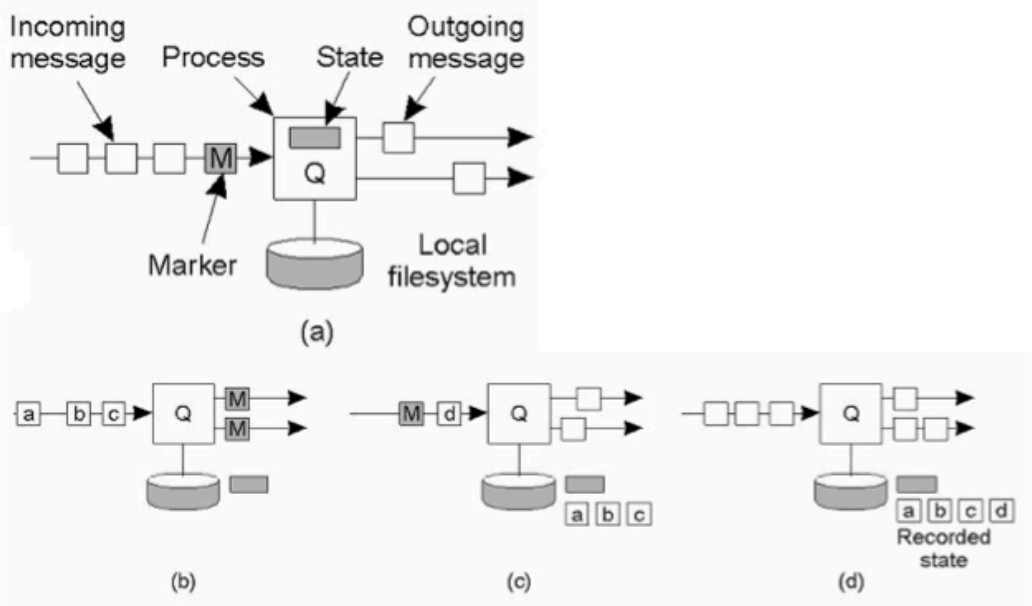
37. Welche Probleme gibt es bei der Ermittlung des "Global State" und wie können diese überwunden werden? Geben Sie zumindest einen Algorithmus an.

Um den Global State aufzuzeichen ist es wichtig einen Consistenten Schnitt zu ziehen (Nachrichten müssen versendet worden sein, bevor sie empfangen werden können!)
Achtung: keiner globale Sicht; kein Stichtag möglich



a) A consistent cut
b) An inconsistent cut (effect without cause)

Lösung durch Chandy & Lamport Algorithmus



Garantiert konsistent, aber vl nie aufgetretene lokale Zustände!

38. Wozu benötigt man Replikation? In welcher Beziehung stehen Replikation und Skalierbarkeit zueinander? Erläutern Sie in diesem Zusammenhang verschiedene Varianten der Content Replication und des Content Placement.

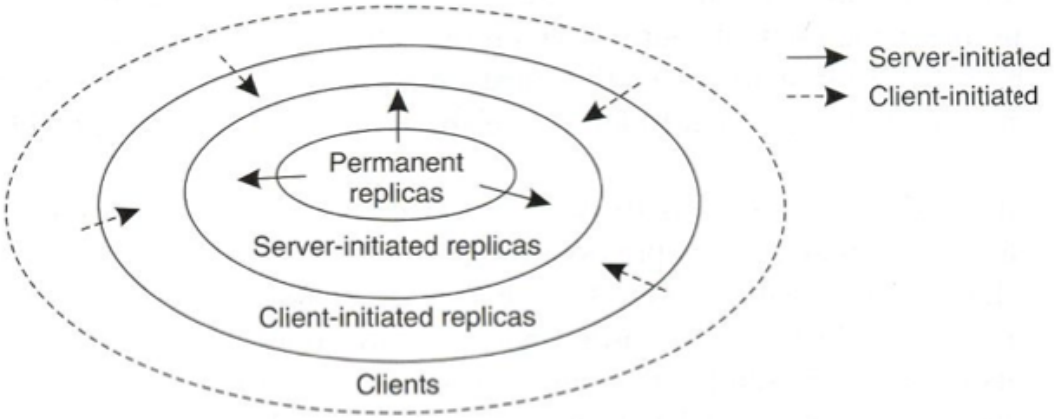
Zuverlässigkeit

- Umschalten im Fehlerfall
- Schutz vor beschädigten Daten

Skalierbarkeit

- Größe
- Geographische Lange

Deutlicher Mehraufwand wegen Konsistenzhaltung



- Permanente (durch Cluster oder Mirroring)
- Server initiated
  - Näher zu den Clients
  - Viele Anfrageb abzudecken
  - Wann wird erstellt/Wann gelöscht
- Client initiated (z.B.: lokaler Browser cache; Company Cache durch Proxy)

39. Geben Sie verschiedene Möglichkeiten der Update Propagation (Content Distribution) an und bewerten Sie diese.

Was wird übertragen

- Nur eine Nachricht, dass etwas aktualisiert wurde
- Übertragung der neuen Daten
- Übertragung der Operationen

Wie wird übertragen

- Push (schnell, aber für den Server aufwändig)
- Poll (nicht so schnell, aber weniger last für den Server)

Mischung der beiden Ansätze durch Leases

- Age based: höhere Leases für relative alte Daten
- Renewal Frequence based: höhere Leases für relative statische Daten
- State space overhead: Abhängig von der Last des Servers

Aspekt	Push-basiert	Pull-basiert
Status am Server	Liste mit Client-Repliken und Caches	Keine
Gesendete Nachrichten	Aktualisierung (und möglicherweise später die Aktualiserung laden)	Pull und Aktualisieren
Antwortzetien auf dem client	Unmittelbar	Laden/Aktualiserungszeit

(Wichtig Unicast & Multicast nicht vergessen!)

40. Erläutern Sie die Funktionsweise der "primary-based" Protokolle. Bewerten und Vergleichen Sie die verschiedenen Arten.

Daten werden immer auf dem Primary Server geschrieben. Achtung: blocking (sicher); non-blocking: kann zu fehlern führen (primary-backup based)

Remote write

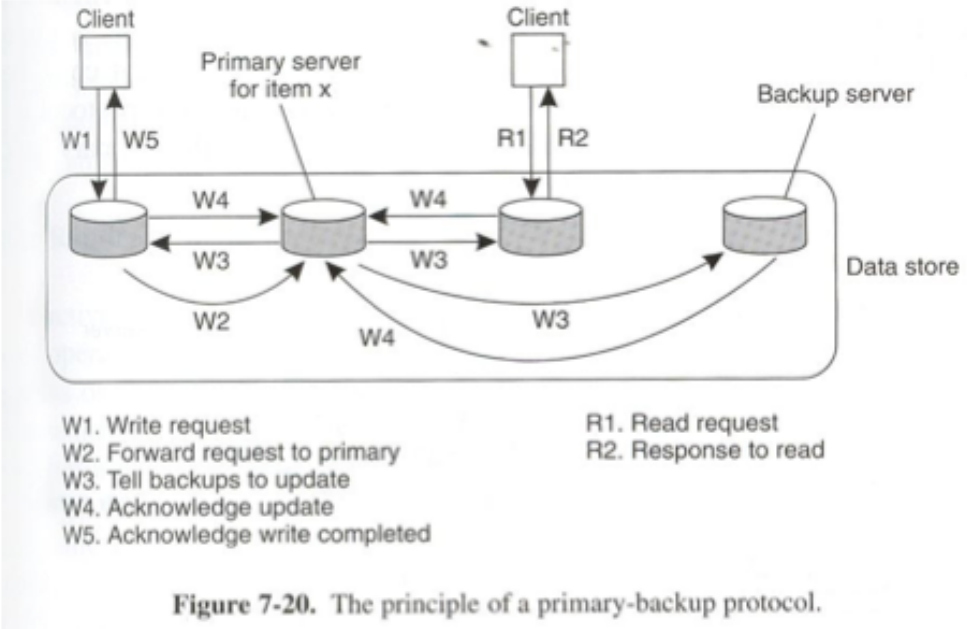


Figure 7-20. The principle of a primary-backup protocol.

Local Write

inkl. Umzug des Primary

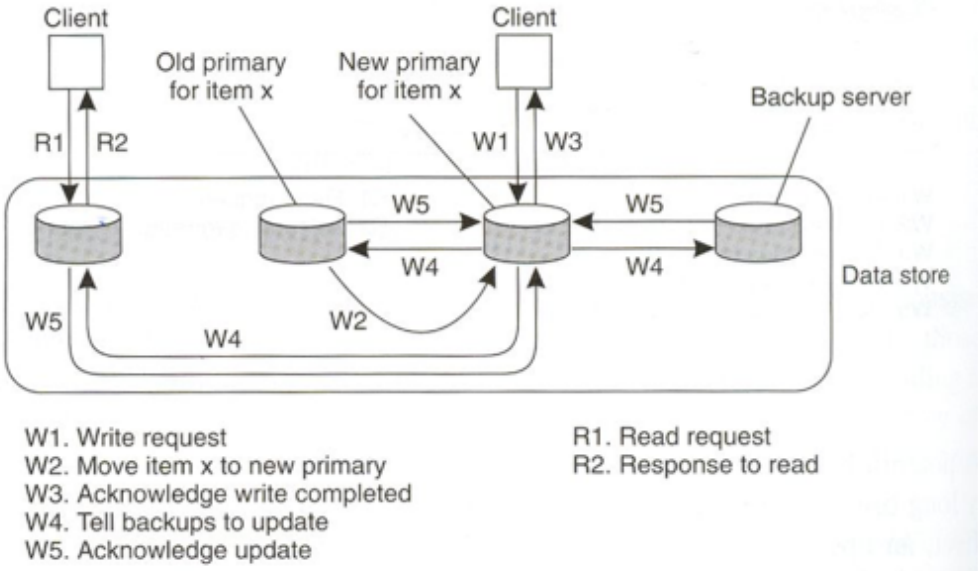


Figure 7-21. Primary-backup protocol in which the primary migrates to the process wanting to perform an update.

41. Erläutern Sie die Funktionsweise der "replicated-write" Protokolle. Bewerten und Vergleichen Sie die verschiedenen Arten.

Active Replication

Operation wird an jedes Replica gesendet (totally ordered multicast - viel Overhead oder Sequenzer)

Quorum based

Nr + Nw > N
Nw > N/2

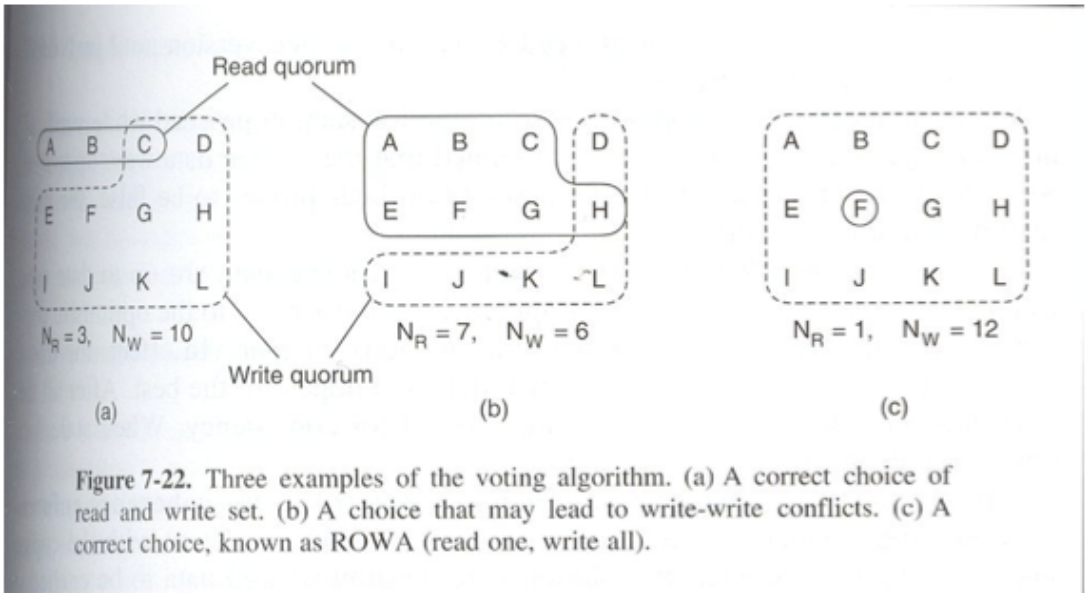


Figure 7-22. Three examples of the voting algorithm. (a) A correct choice of read and write set. (b) A choice that may lead to write-write conflicts. (c) A correct choice, known as ROWA (read one, write all).

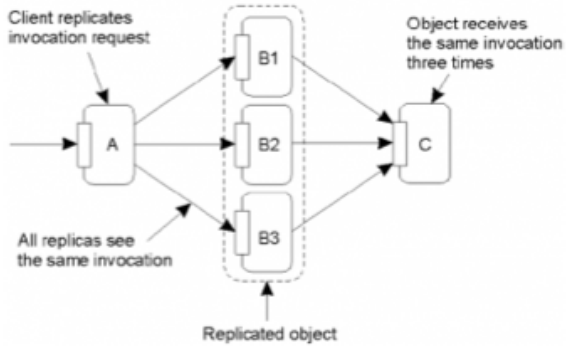
42. Welche Besonderheiten sind bei der Replikation von Objekten zu beachten?

Replikation muss auch alle Replikate verteilt werden

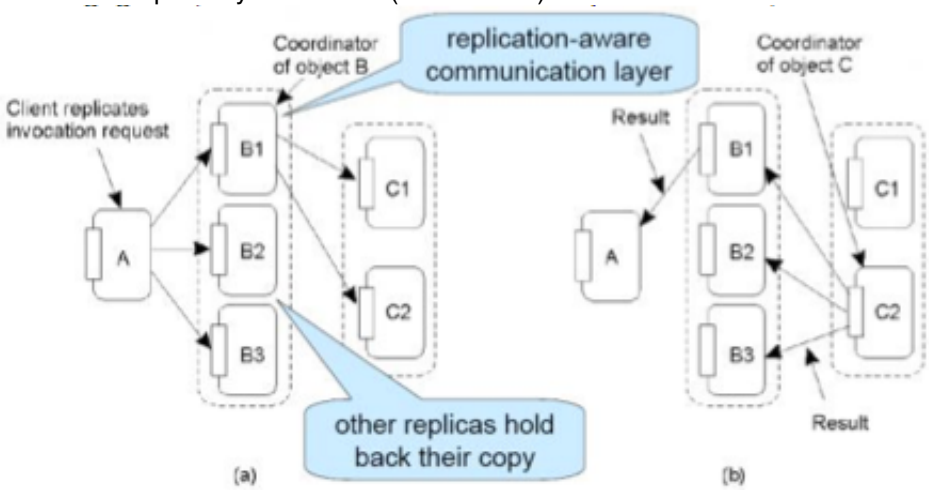
- Primary based: Koordinator leitet alles weiter
- Totally ordered multicast & timestamps

Problem bei nested objects

Ein call wird auf ein Unterobject mehrmals durchgeführt



Replication Transparency hilft mittels (Coordinator)



43. Was sind Epidemic Protocols. Welche Vor- und Nachteile haben diese.

Aus der Medizin: man steckt sich mit neuen Informationen an (Wichtig: Death Certifictes)
Skalieren sehr gut!

Anti Entrophy Modell

Server wählen zufällig neue Server aus und tauschen die Informationen aus (push, pull möglich)

Gossiping

Server mit neuen Informationen wollen diese mit zufälligen Servern teilen, wenn diese die Informationen schon haben, ist es bald uninteressant weiter zu teilen. (Achtung es kann nicht garantiert werden, das alle Server aktualisiert wurden!)

44. Erläutern Sie die grundlegenden Begriffe der Fehlertoleranz. Was ist der Unterschied zwischen Availability und Reliability? Wie ist der Zusammenhang zwischen Failure, Error und Fault?

Dependability

Fähigkeit eines Systems Dienste anzubieten, welche das liefern, was der Nutzer braucht. So kann ein System, welches die Spezifikation erfüllt trotzdem Fehlerhat sein!

Availability

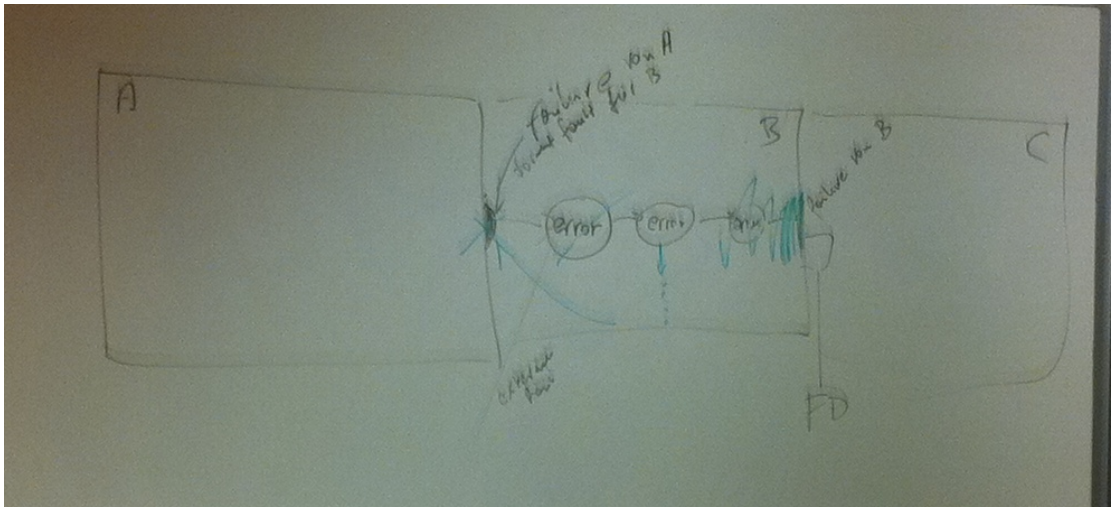
Die Wahrscheinlichkeit, dass ein System zu einem Zeitpunkt korrekt arbeitet

Reliability

Die Zeitspanne, in welchem das System korrekt arbeitet

Beispiel: Ein System welches 2 Wochen im Jahr gewartet wird, sonst jedoch nicht ausfällt, hat eine hohe Reliability aber nur eine Availability von 96%

Failure, Error, Fault



Möglichkeit noch eine Schicht um das System zu ziehen, um Failures an Endpunkten zu erkennen (Beispiel: Ariane Rackete)

45. Geben Sie verschiedene Fehlermodelle an und diskutieren Sie diese. Wozu benötigt man überhaupt Fehlermodelle? Inwiefern ist es u.U. heikel zu spezifizieren, daß ein System "k-fault-tolerant" sein soll?

Fehlermodelle werden benötigt, um die Fehler besser zu unterteilen und ihre Auswirkungen. einschätzen zu können.

- Crash Failure: Server stürzt ab
- Ommision Failure: Dienstausfall (Server antwortet nicht; receive/send Ommision)
- Timing Response Failure: Antwort dauert zu lange
- Response Failure: Falsche Antwort (Falsche Antwort bzw. Falsche Aktion)
- Inconsistent Failure: Der Server gibt zu gleichen Anfragen unterschiedliche Antworten

- Transient Fault: treten zufällig aber einmalig auf
- Intermittet fault: tretten immer wieder zufällig auf
- Permanent Faul: tretten auf, bis sie behoben sind

Crash Failure	Ist einfach weg	k+1 Nodes
Meldet sich ab		K+1 Nodes
Babbling Idiot		2k+1
Byzantine Failure	Von extern würde 2k+1 reichen	3k+1

Problematisch

Problematisch ist es, weil oft nicht bedacht wird, wass passiert wenn k+1 Nodes ausfallen! Oft ist es auch sinnvoll, Fehlererkennung in die Applikation einzubauen damit z.B.: byzantinische Fehler nicht mehr auftreten können!

46. Wieso benötigt man Redundanz zur Maskierung von Fehlern? Welche Arten von Redundanz gibt es?

Es braucht also irgendeine Art von BackUp um den Ausfall zu Maskieren (z.B.: einen Replica)

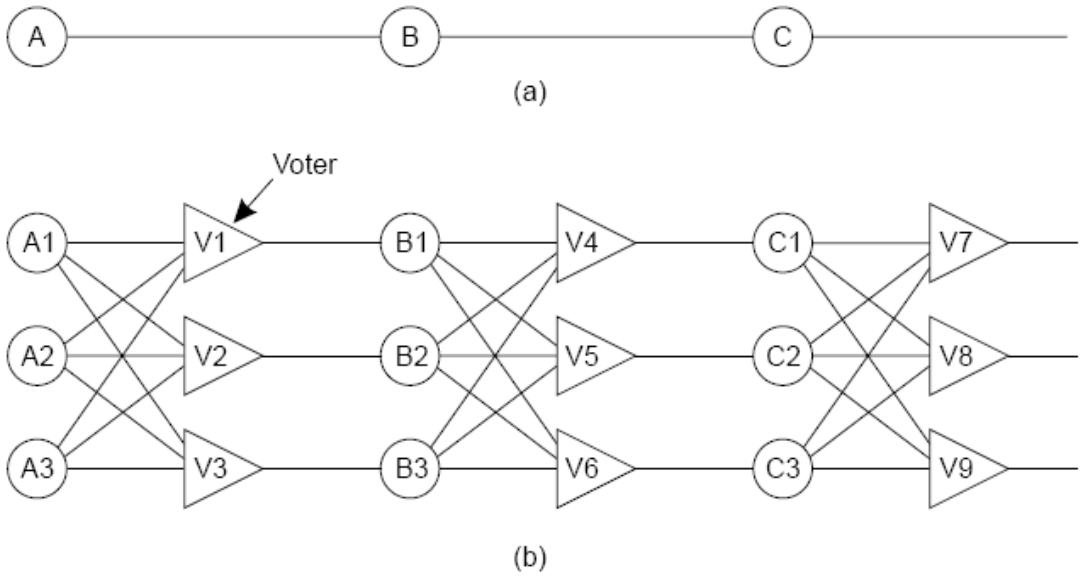


Figure 7-2. Triple modular redundancy.

Zeitliche (es wir nochmals gesendet)
Datentechnische (Prüfsummen)
Physikalische (2 Server)

47. Erläutern Sie die Aussage des "two-army" Problems.

Hier soll ein gemeinsamer Konsense gefunden werden

A	=>	B eine Nachricht	1)
A	<=	B antwortet	2)

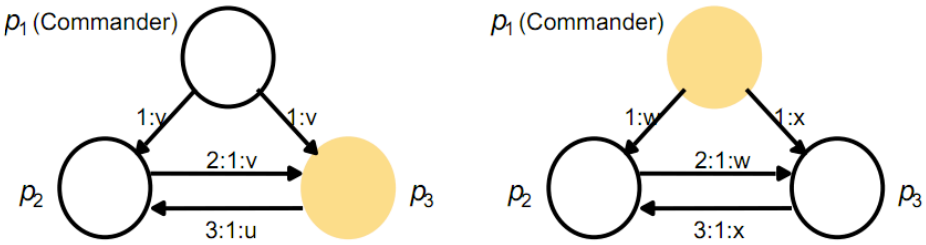
Hier gibt es jedoch einige Probleme

Wenn die Nachricht bei 1 verloren geht => kein Problem
Wenn B die Nachricht erhält, aber A nicht => nur B kommt, aber A nicht

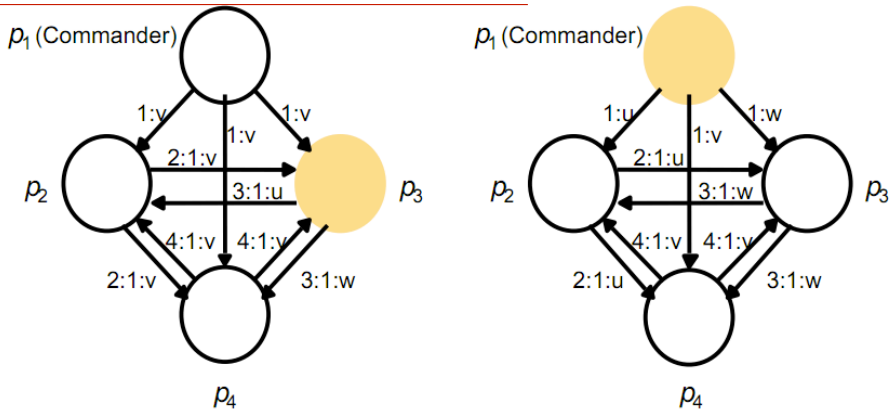
Es kann also nie 100% sicher gesagt werden, dass A und B eine Übereinkunft getroffen haben!

48. Erläutern Sie die Aussage der "Byzantinischen Generäle".

Es werden 3k+1 Nodes benötigt, dann wird immer der Richtige Konsens gefunden, ausser der Kommander lügt.



Faulty processes are shown shaded



Faulty processes are shown shaded

49. Erläutern Sie die Fehlerklassen in RPC-Client/server-Umgebungen. Gehen Sie besonders auf das "lost reply" Problem ein.

Fehlerklassen

- Client kann Server nicht finden
- Anfrage vom Client an den Server geht verloren
- Server stürtzt währen Verarbeitung ab
- Antwort geht verloren
- Client stürzt ab

Lost Reply

Besonders Problematisch ist, wenn der Client nicht weiß, ob eine Aktion durchgeführt wurde, und diese nicht idempotent (z.B.: lesen eines Files) ist. Um dieses Problem zu beheben, können beispielsweise Anfragen IDs genutzt werden (Nochmalige Anfragen für den Server erkennbar)