Verteilte Systeme Testvorbereitung

Foliensatz 1

Was ist ein verteiltes System?
Eine Sammlung von autonomen Rechensystemen, die dem User als ein koheränes System erscheinen (= Distribution Transparency).

Transparency	Description
Access	Hide differences in data representation and how an object is accessed
Location	Hide where an object is located
Relocation	Hide that an object may be moved to another location while in use
Migration	Hide that an object may move to another location
Replication	Hide that an object is replicated
Concurrency	Hide that an object may be shared by several independent users
Failure	Hide the failure and recovery of an object

Ziele:
- Offenheit
- Skalierbarkeit
- Distributed Transparency
- Ressource sharing

Schwierigkeiten: Synchronisation + Koordination; Nodes müssen miteinander kommunizieren

Lösung: **Overlay-Netz** = Rechnernetz auf einem bestehenden Netz (Underlay).
- (logisches) Netz oberhalb existierender Infrastruktur
- oftmals eigener Adressraum mit eigener Adressierung (unabhängig vom Underlay)
- ggf. Einsatz eigener Wegewahlverfahren

Das Overlay-Netz wird daher zum Aufbau einer zusätzlichen Topologie (physikalisch, logisch, strukturell …) genutzt. (z.B.: Peer-to-peer system; tree, ring,…)

Was ist Middleware?
Middleware = anwendungsneutrale Programme, vermittelt zwischen Anwendungen; Verteilungsplattform oder Protokollbündel auf einer höheren Schicht. Im Gegensatz zu niveautieferen Netzwerkdiensten, welche die einfache Kommunikation zwischen Rechnern handhaben, unterstützt Middleware die Kommunikation zwischen *Prozessen*.

Die Middleware ist eine Softwareschicht, die auf Basis standardisierter Schnittstellen und Protokolle Dienste für eine transparente Kommunikation verteilter Anwendungen bereitstellt. Middleware Dienste stellen eine Infrastruktur für die Integration von Anwendungen und Daten in einem heterogenen und verteilten Umfeld zur Verfügung (Bengel, 2014).
Ein Interceptor (-Pattern) ermöglicht das transparente Hinzufügen von Diensten zu einer Middleware (oder auch Framework) und das automatische Auslösen, wenn bestimmte Ereignisse auftreten. Solche Services könnten unter anderem eine Authentifizierung oder Logging sein (Dr.-Ing Eichberg).

Full distribution transparency is impossible
- Completely hiding failures of networks and nodes is impossible
- Full transparency will cost performance (replicas, flushing write operations)

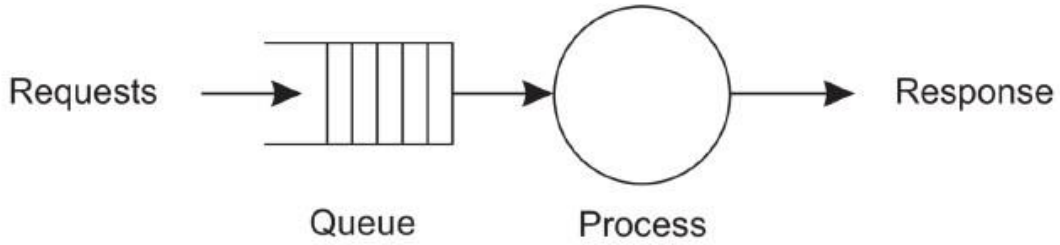
Exposing distribution may be good
- Making use of location-based services
- When dealing with users in different time zones
- When it makes it easier for a user to understand what's going on

Offenheit
Systems should:
- conform to well-defined interfaces (**Kompatibilität**)
- easily interoperate (**Kooperation**)
- support portability of applications (**Portabilität**)
- be easily extensible (**Erweiterbarkeit**)

Balance seperation – Komplexität vs. Flexiblität (es ist eine adäquate Balance zu finden).

Skalierbarkeit
- Size (Anzahl User, Prozesse) → Rechenleistung, Storage, Netzwerk
- geographisch (Distanz zwischen Nodes)
- administrativ (Anzahl der Domains)

Size → Lösung: Queuing System



geographische Skalierbarkeit: Probleme •
LAN auf WAN nicht einfach möglich
- Synchronizität! Latenz!
- WAN „links" sind unzuverlässig
- Lack of multipoint communication! Search broadcast? Develop naming and directory services!

administrative Skalierbarkeit: Probleme
- Conflicting policies (usage, management, and security)
- Ausnahmen: manche peer-to-peer networks
- File-sharing systems (based, e.g., on BitTorrent)
- Peer-to-peer telephony (Skype)
- Peer-assisted audio streaming (Spotify)
- End users collaborate and not administrative entities.

Techniken für Skalierung:
- Asynchrone Kommunikation (UDP?)
- Berechnung passiert beim client, nur Result wird versendet
- Dezentralisierung auf viele Rechner (DNS, WWW)
- Replizierung und Caching (Mirroring Websites, replicated DB + File Servers, file caching)

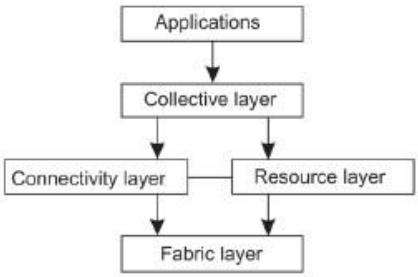
Probleme mit Replizierzung → Inkonsitenzen
Konsistente Daten verlangen globale Synchronization.

Pitfalls beim Entwickeln von verteilten Systemen (falsche Annahmen):
- The network is reliable
- The network is secure
- The network is homogeneous
- The topology does not change
- Latency is zero
- Bandwidth is infinite
- Transport cost is zero
- There is one administrator

3 Typen verteilter Systeme:
- High Performance (Parallel Computing)
- Informationssysteme
- Rechnerdurchdringung (Allgegenwärtigkeit; z.B.: IoT)

Cluster Computing – Gruppe von homogenen Computern verbunden via LAN (selbes OS und ähnliche Specs). Einzelne Managing Node.

Grid Computing – heterogene PC's von überall verbinden sich. Virtuelle Organisationen (group of users) um Allokation von Daten zu managen.



The layers

- Fabric
  interfaces to local resources (querying state and capabilities, locking, etc.)
- Connectivity
  Communication/transaction protocols, e.g., for moving data between resources, authentication protocols
- Resource
  Manages a single resource, such as creating processes or reading data.
- Collective
  Handles access to multiple resources: discovery, scheduling, replication.

Cloud Computing

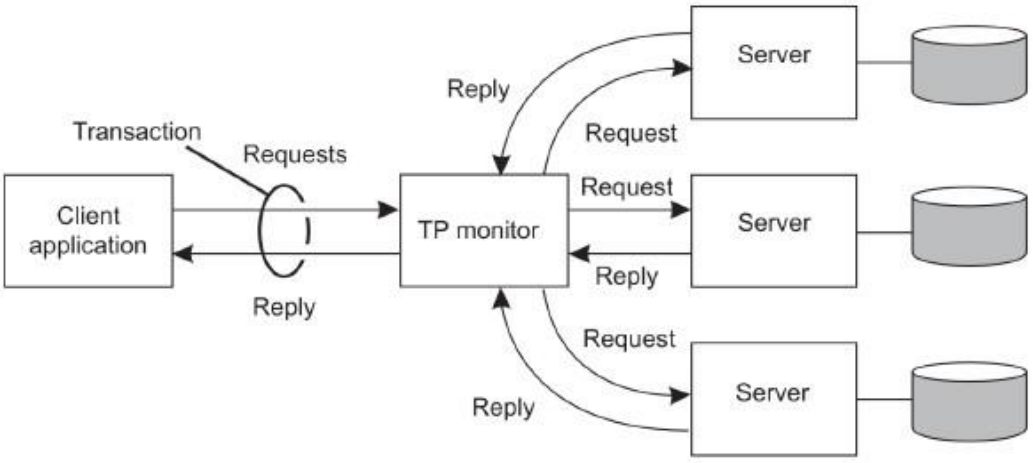
- **Infrastructure as a Service (IaaS)** - CPU, Memory, Disk, Bandwidth, VM's, Netzwerk
- **Platform as a Service (PaaS) -** Application Services, Databases, Pay for reserved resources
- **Functions as a Service (FaaS)** – AWS Lambda, Azure Functions, Logic Apps (Pay per use)
- **Software as a Service (SaaS)** - (YouTube, Flickr, Office 365, Azure DevOps)

Batch Computing = Stapelverarbeitung (Tasks werden geplant, z.B.: Disney rendering in der Nacht)

EAI = Enterprise Application Integration; Geschäftsabwicklung durch ein Netzwerk unternehmensinterner Applikationen verschiedener Generationen und Architekturen. Alle Apps sollen gemeinsam einen Geschäftsprozess unterstützen und ausführen, obwohl sie heterogen sind.

A **nested transaction** is a database transaction that is started by an instruction within the scope of an already started transaction.

TPM (Transaction Processing Monitor) – kontrolliert Transaktionen, wenn mehrere Terninals (Server) involviert sind.



Middleware & EAI

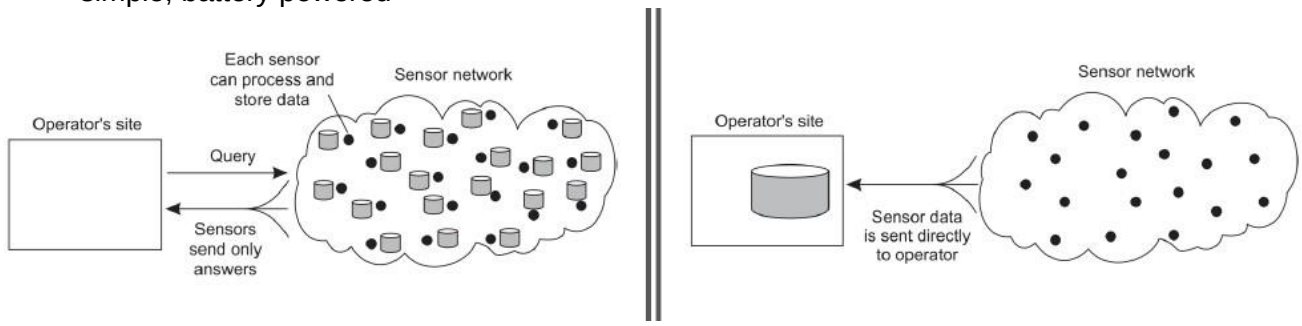
- file transfer
- messaging
- Remote Procedure Call
- shared DB

Remote Procedure Call (RPC) – Procedure Call wird ausgeführt, als wäre er lokal, allerdings wird er weitergeleitet (auf ein anderes System/ Adressraum), ohne dass dies explizit im Code verankert ist.

Message oriented Middleware (MOM) – Middleware provides distributed communication layer for sending messages across multiple platforms. Easier than developing for heterogenous Systems (different OS & Protocols)

Next generation of distributed systems: small, mobile nodes (often embedded) 3 Subtypes:

- **Ubiquitous computing systems** computing anytime and everywhere using any device, in any location, and in any format. A user interacts with the computer, which can exist in many different forms, including laptop computers, tablets and terminals in everyday objects such as a refrigerator or a pair of glasses.
- **Mobile computing systems**
- **Sensor (and actuator) networks** sensing of the environment. Nodes are many, simple, battery powered



Ubiquitous Systems Core Elements:

- **Distribution** – Transparenz der Devices
- **Interaction** - zwischen user und device ist dezent und unauffällig
- **Context awareness** – des Users
- **Autonomy** – Device agiert autonom und weitestgehend ohne menschliches Zutun
- **Intelligence** – reagiert smart auf veränderte Umstände; situationsbezogen

Foliensatz 2

Software vs. System Architektur

Monolitische Applikation
- Single tiered (einstufig)
- User Interface und Datenzugriff sind kombiniert
- unabhängig
- nicht modular

N-tier Applikation
- layered architecture
- oft bei IaaS
- modular
- migirert bereits existierende Applikationen
- OSI Modell
- traditionell 3-layered:
    1. User Interface Layer
    2. Business Layer
    3. Data Layer

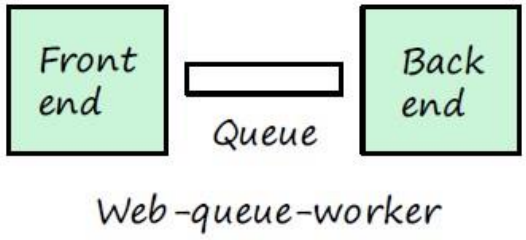
Component-based Architecture (Object based)
- objekte viel lockerer verbunden als layers
- everything (processes, files, I/O operations, etc.) is represented as an object
- Objects encapsulate data without revealing the internal implementation.
- Service-oriented architecture (SOA)
- RPC, DCOM wird verwendet, damit object interagieren

Event-based Architecture

Systemarchitekturen

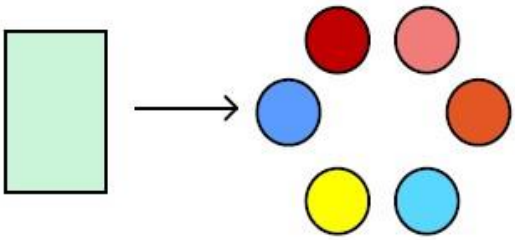
- Centralized
- Client-Server
- Application layering (logical software layering)
- Multi-tiered (system architecture) **- Decentralized**
- Structured peer-to-peer
- Unstructured peer-to-peer
- Super-peers
- Hybrid architectures
- Edge-server systems
- Collaborative distributed systems

Web Queue Worker
Web-Front-End → stellt Clientanforderungen bereit → **Worker** erledigt ressourcenintensive Aufgaben. Web-Front End kommuniziert über eine Queue mit dem Worker. Für einfache Domains.



Web-queue-worker

Microservices
Komplexe Anwendungssoftware ist aus unabhängigen Prozessen komponiert, welche über sprachunabhängige Programmierschnittstellen kommunizieren. Dienste weitgehend entkoppelt → jeder erledig eine kleine Aufgabe → modular



Microservices

- complex domains
- harder to manage and build
- faster innovation

CQRS – Command Query Responsibility Segregation
"CQRS" (Command and Query Responsibility Segregation) bedeutet die Trennung von Befehls- und Abfrageverantwortung und trennt die Lese- und Aktualisierungsprozesse für einen Datenspeicher.
Die Vorteile der Implementierung von „CQRS“ sind höhere Leistung, Skalierbarkeit und maximale Sicherheit. Diese Implementierung ermöglicht es einem System, sich besser zu entwickeln und zu verhindern, dass Konflikte auf Domänenebene zusammengeführt werden. Es wird hauptsächlich z.B. mit vielen parallelen Zugriffen auf dieselben Daten.

Peer-to-peer System
- Alle Peers sind gleichwertig
- Node can send messages via communication channels

Strukturiertes peer-to-peer system
- efficiently look up data
- using ring, binary tree, grid…
- data items are identified by **unique key**
- find system with **hash of key**

Unstrukturiertes peer-to-peer system
- nodes maintain ad-hoc (in diesem Augenblick) list of neighbors
- random graph
- flooding, random walks

Foliensatz 3

Ein Prozess besteht aus mehreren Threads.
Alle Thread teilen sich „process state“

Hierarchie:

- Prozess (Task) > OS
  - Thread (Kernel Thread) > OS
    - Fiber (User Thread) > Anwendung/Prozess

Wieso Threads?

- Parallelisierung
- Blocken vermeiden
- process switching vermeiden

Fibers

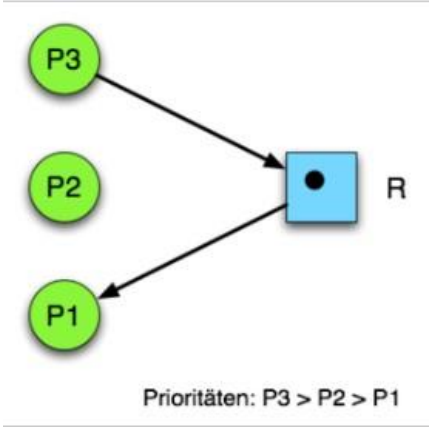
- werden NIE vom OS gescheduled
- …sondern von der Applikation (kooperatives scheduling)

Unterscheidung zwischen Kernel / User Threads

1:1 kernel-level threading	Linux, Solaris, MacOS
N:1 user-level threading	GNU portable threads
M:N hybrid threading	ab Win7

Preemptive Multitasking – Das OS bricht Porzesse ggf ab
Kooperatives Multitasking – Wenn ein Programm blockiert, muss neu gebootet werden

Prioritäten können gesetzt werden.
Priority Inversion – Endlosschleife, Thread ist für immer in wait, weil ein anderer locked.
Bekommt einen **Priority boost**

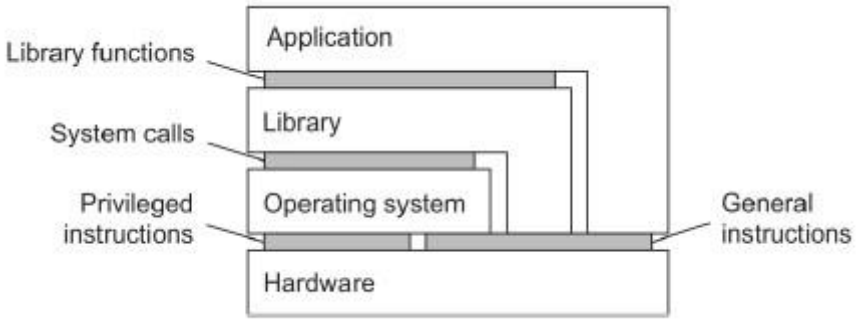


P2 verdrängt P1, daher kommt P3 nie dran, obwohl die höchste Priorität.
P1 → Lock
P3 → wait

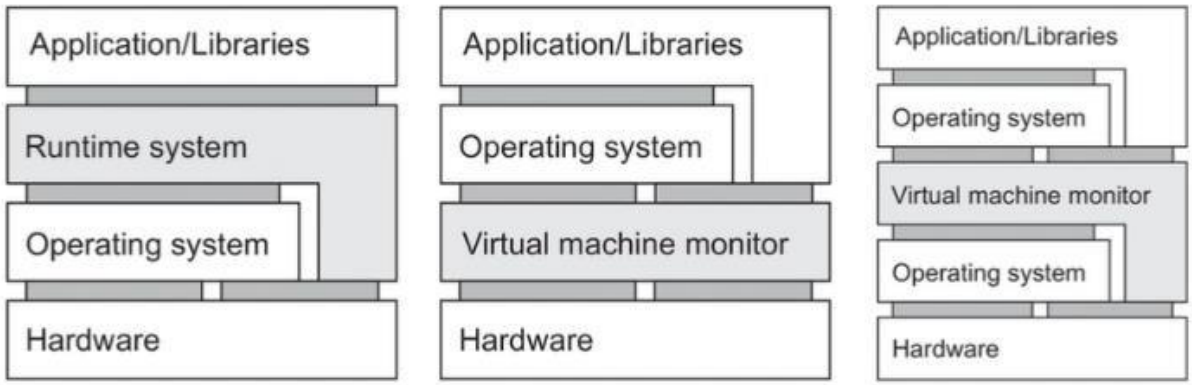
Probleme bei Threads:

- **Deadlock**
- **Race Condition** (zwei Systeme greifen gleichzeitig zu, für die Korrekte ausführung muss aber eine gewisse Reihenfolge eingehalten werden.

OS



Virtual Machine



- **Process-VM**
  Separate set of instructions, interpreter/emulator/runtime running on top of OS
- **Native-VMM**
  low-level instructions, minimal operating system
- **Hosted VMM**
  low-level instructions, delegating most work to full-fledged OS

VM's und die Cloud • **IaaS**
Basic Infrastructure, Virtual Machines

- **PaaS**
  System-level Services, VM behind the scenes but cannot be fully managed
- **FaaS**
  Not paying for the VM, Pay per use
- **SaaS**
  Actual applications

chroot = change root bei Linux. Man kann / als root directory ablösen

Docker generell

Docker ist eine Container-Virtualisierungs-Technologie. Einen Docker Container kann man sich wie eine leichtgewichtige virtuelle Maschine (Anderson, 2015) oder auch als ein leichtes Äquivalent einer virtuellen Maschine vorstellen. Den Kern von Docker bilden Linux Containers und LXC, ein User-Space-Control Paket für Linux-Container.
LXC verwendet Namespaces auf Kernelebene, um den Container vom Host zu isolieren:

- Der User-Namespace trennt die Benutzerdatenbank des Containers und des Hosts und stellt so sicher, dass der Root-Benutzer des Containers keine Root-Berechtigungen für den Host hat.
- Der Prozess-Namespace ist dafür verantwortlich, nur Prozesse anzuzeigen und zu verwalten, die im Container ausgeführt werden, nicht den Host.
- Der Netzwerk-Namespace stellt dem Container ein eigenes Netzwerkgerät und eine virtuelle IP-Adresse zur Verfügung.

Eine weitere von LXC bereitgestellte Komponente von Docker sind Kontrollgruppen (cgroups). Kontrollgruppen implementieren die Ressourcenabrechnung und -begrenzung. Während Docker die von einem Container verbrauchten Ressourcen wie Speicher, Speicherplatz und E / A einschränken kann, gibt cgroups auch viele Metriken zu diesen Ressourcen aus. Mit diesen Metriken kann Docker den Ressourcenverbrauch der verschiedenen Prozesse in den Containern überwachen und sicherstellen, dass jeder nur seinen angemessenen Anteil an den verfügbaren Ressourcen erhält (Merkel, 2014).

- Package of software and dependencies
- Run inside virtual user space **Docker images**
- Images are **immutable** (=state cannot be modified after it is created) *deployment units* to package **code and dependencies**. **Docker registry**
- The Docker Hub contains predefined images **Docker container**
- Docker containers running on a single machine *share the machines resources*

Docker File

Contains commands to build a Docker Image

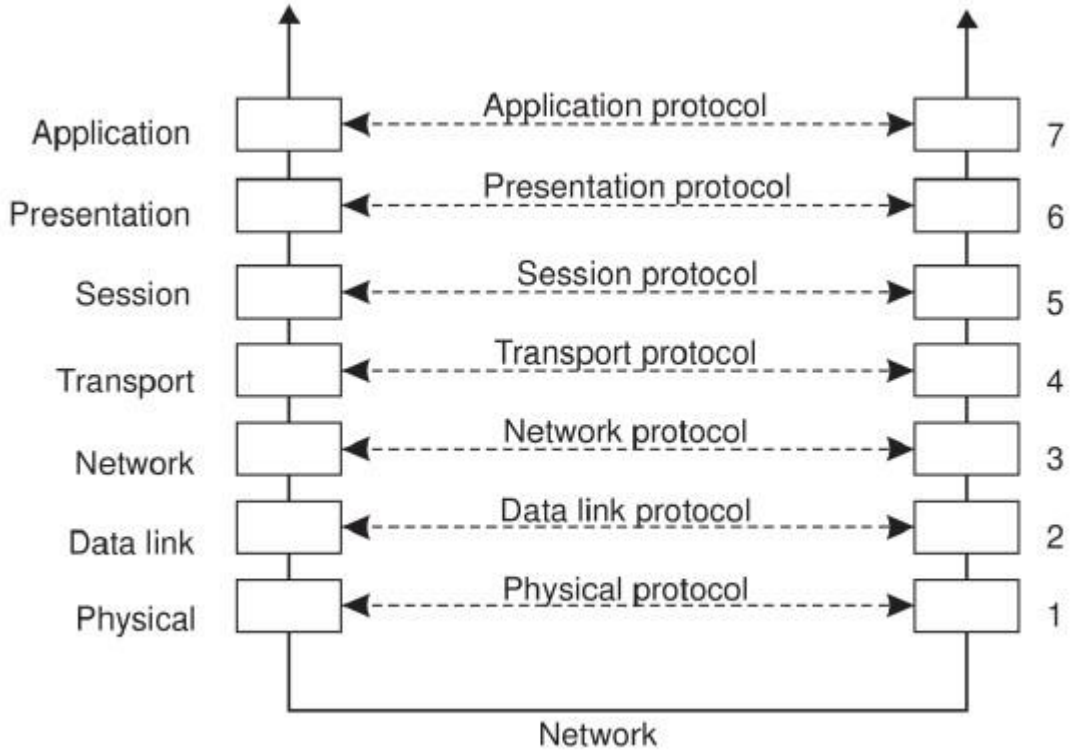
```
3  FROM mcr.microsoft.com/dotnet/core/aspnet:3.1-buster-slim AS base
4  WORKDIR /app
5  EXPOSE 80
6  EXPOSE 443
7
8  FROM mcr.microsoft.com/dotnet/core/sdk:3.1-buster AS build
9  WORKDIR /src
10 COPY ["WebAppWithDocker/WebAppWithDocker.csproj", "WebAppWithDocker/"]
11 RUN dotnet restore "WebAppWithDocker/WebAppWithDocker.csproj"
12 COPY . .
13 WORKDIR "/src/WebAppWithDocker"
14 RUN dotnet build "WebAppWithDocker.csproj" -c Release -o /app/build
15
```

Container werden auf Betriebssystemebene virtualisiert, während Hypervisor-basierte Lösungen auf Hardwareebene virtualisiert werden. Obwohl der Effekt ähnlich ist, sind die Unterschiede wichtig und signifikant.

Container stellen geschützte Teile des Betriebssystems zur Verfügung - sie virtualisieren das Betriebssystem effektiv. Zwei Container, die auf demselben Betriebssystem ausgeführt werden, wissen nicht, dass sie Ressourcen gemeinsam nutzen, da jeder über eine eigene abstrahierte Netzwerkschicht, Prozesse usw. verfügt.

Foliensatz 4

OSI MODELL



Transport Layer:

TCP
- connection-oriented
- reliable
- stream-oriented communication

UDP
- connectionless
- unreliable (best effort)
- datagram communication
- um einiges kleiner als TCP Paket

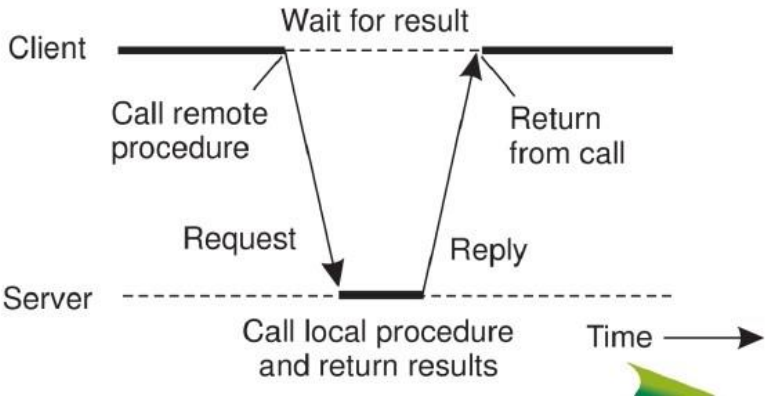
Types of communication:

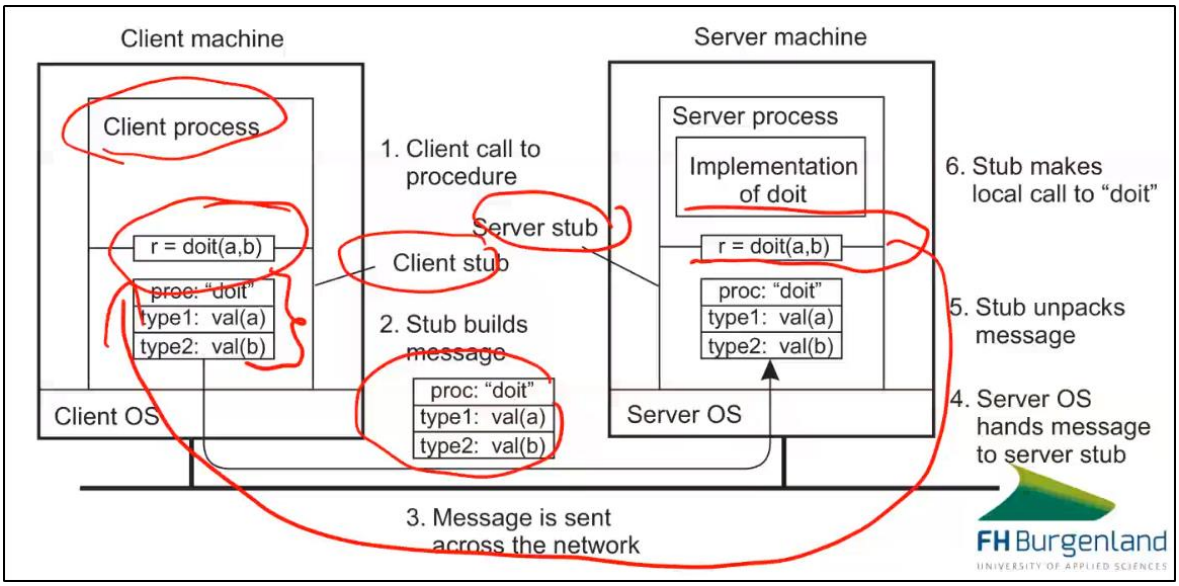
Transient vs. **persistent communication**
weggeschmissen gespeichert

Asynchronous vs. **synchronous communication**
es gibt versch. Punkte, welche die Synchonisation auslösen können.

Client ↔ Server = transient synchronous
Messaging = persistent asynchronous

Remote Procedure Call – RPC
(Dem User ist verborgen, wo die Methode abgerufen wird)





RPC-based Technologies

- Distributed Computing Environment (**DCE/RPC**)
- commissioned by the Open Software Foundation (**OSF**), now The Open Group
- Distributed Component Object Model (**DCOM**)
- Microsoft
- Based on **DCE/RPC**
- Security-focused changes since Windows XP SP2
- Common Object Request Broker Architecture (**CORBA**)
- Object Management Group (**OMG**)

Simple Object Access Protocol (SOAP) v1.1

- **RPC**/Encoded
- Invoke methods via a proxy

SOAP v1.2

- SOAP = Protocol
- Document/Literal

- Send messages defined by **XML Schema** (Document)
- Feature: Versioning
- WCF = implementierung von SOAP

REST API's

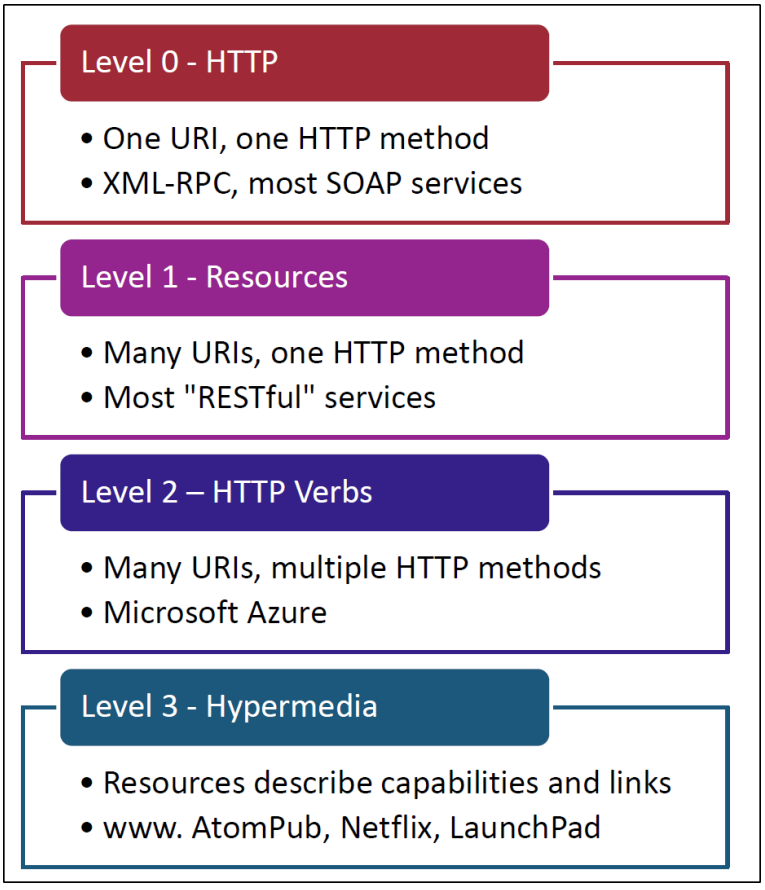
- REST = Guideline
- Client-Server
- Statelessness (soap = „stateful“)
  Client kann jedes mal einen anderen Server aufrufen → sehr skalierbar
  bei Statefull muss Client immer denselben Server aufrufen.
- Cachability
- Layered System
- Code on demand (optional)
- Uniform interface

Uniform Interface:

- Resource identification in requests
- Resource manipulation through representations
- Self-descriptive messages
- Hypermedia as the engine of application state (HATEOAS)

REST Levels:

Nicht immer alle features von REST APIs notwendig, daher verschiedene Levels:



Meisten SOAP Services oder auch RPC erfüllen Level 0

Level 1 kann man mehrere Ressources unterscheiden, da gibt es icht mehr „einen“ link zu einem service (für jedes Ressource)

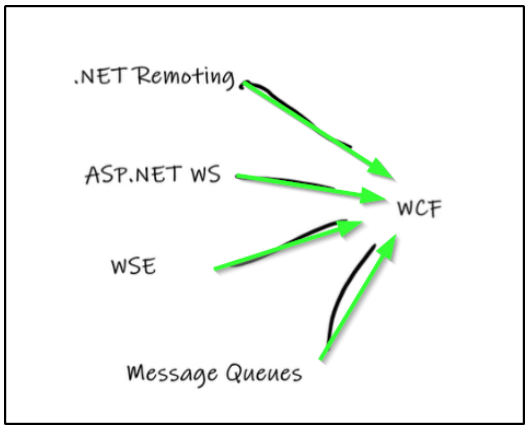
Meisten modernen APIs unterstützen Level 2. (DELETE request, PUT request, …)

Level 3 kommt HATEOAS dazu: (hypermedia) welche Daten und Links verschicke ich mit.

gRPC

- Binary communication between services
- Protobuf wire transfer

mit WCF musste man nicht mehr jede API lernen, WCF war „die eine Technologie für alles“.
Nachteil: WCF war sehr komplex, man musste alles konfigurieren



Für binäre Kommunikation hat sich gRPC etabliert. (bei cloudservices wo man für Rechenleistung zahlt können Kosten steigen wenn man mit json arbeitet 26 → binäre Kommunikation > kostet weniger Rechenleistung und somit weniger Geld)

Bei gRPC kann man eine Verbindung offenhalten und direkt vom Server zum Client kommunizieren (Plattform unabhängig)

gRPC setzt auf HTTP2 auf.

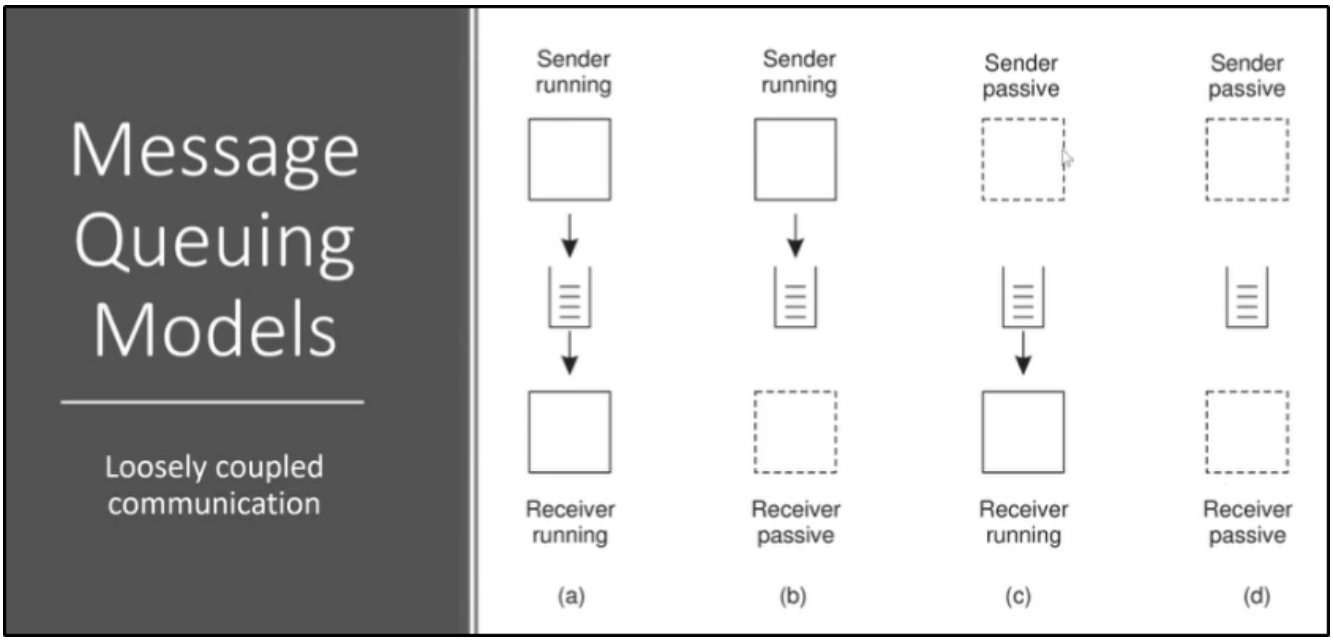
Websockets:

früher hat der Client „ständig“ beim Server nachgefragt, ob es „etwas Neues“ gibt. → polling, um durch eine Firewall durchzukommen.

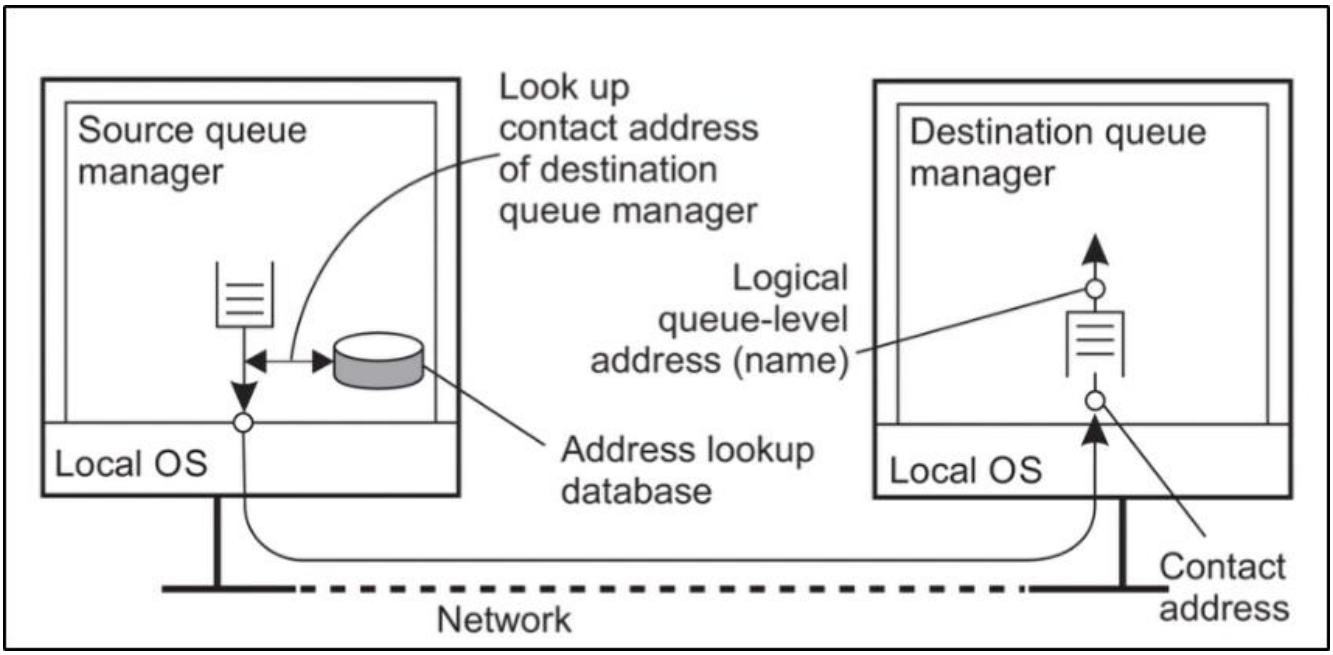
Bei Websockets bleibt, im Gegensatz zu http, die Verbindung offen und Server kann laufend Informationen zurückschicken. Websockets = „eine Lüge“, dass wir durch die Firewall kommen.

Message Queuing

Sender und Receiver sind disconnected, Sender schickt eine Nachricht aber der Receiver muss nicht sofort „online“ sein. Kann „später“ bearbeiten.



Local and Remote Queues:



zB bei Windows: Clientapplication will eine Queue schicken, ohne dass das Zielsystem erreichbar ist. System speichert Queue lokal. Beim nächsten Booten kann dann die Queue ans Ziel geschickt werden, ohne dass die Applikation laufen muss.

5. Consistency & Replication

5.1 Reasons:

- Reliability
- Perfomance

5.2 Degree of consistency (Continous Consistency)

Numerische Abweichung (numeric deviation))

- absolute numerische Abweichung (z. B. 0,02 USD)
- relative numerische Abweichung (z. B. 0,5%)
- Anzahl der Aktualisierungen, die auf ein bestimmtes Replikat angewendet wurden und von anderen noch nicht gesehen wurden (Gewicht)

Staleness deviation:

- Das letzte Mal waren Daten Aktualisierungen

Geordnete Abweichung (ordered deviation):

Aktualisierungen (updates) werden vorläufig auf eine lokale Kopie angewendet und warten auf die globale Zustimmung der Replikate (global agreement of replicas).

5.3 Sequential consistency, causual consistency, eventual consistency

Sequentielle Konsistenz

- Operationen werden in einer Reihenfolge angezeigt, die vom Programm „bestimmt“ wird.

Kausale Konsistenz

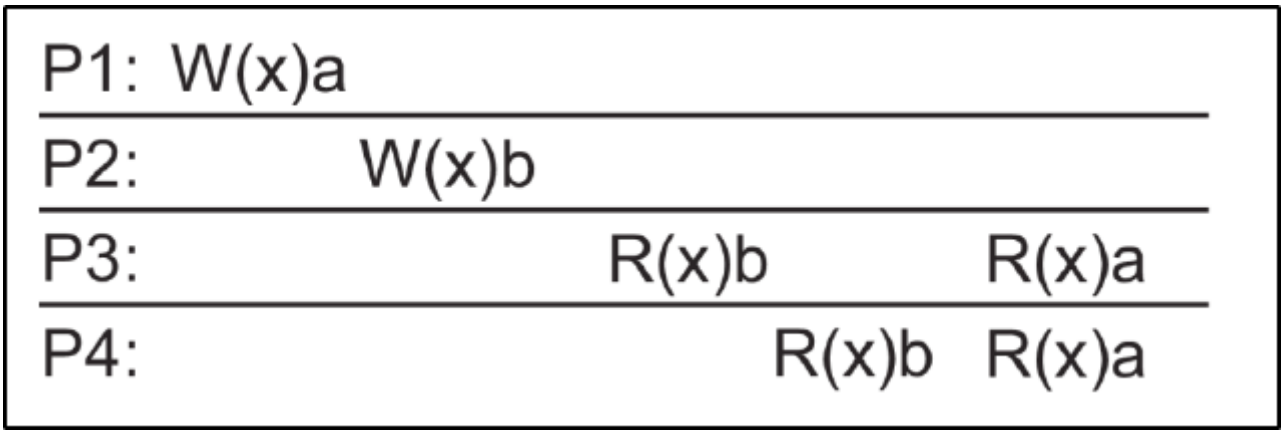
- Schwächung der sequentiellen Konsistenz
- Schriften, die kausal zusammenhängen, müssen von allen Prozessen in derselben Reihenfolge gesehen werden
- Gleichzeitige Schreibvorgänge werden möglicherweise in einer anderen Reihenfolge angezeigt

Eventuelle Konsistenz

- Was sind gleichzeitige/konkurierende Bedürfnisse?
- Datenbank mit seltenen Update-Vorgängen?
- Wie schnell müssen Updates für Lesevorgänge verfügbar gemacht werden?

- Bei den meisten Clients, die auf dasselbe replica umgeleitet werden, treten niemals Inkonsistenzen auf
- Beispiele: Webproxys, DNS

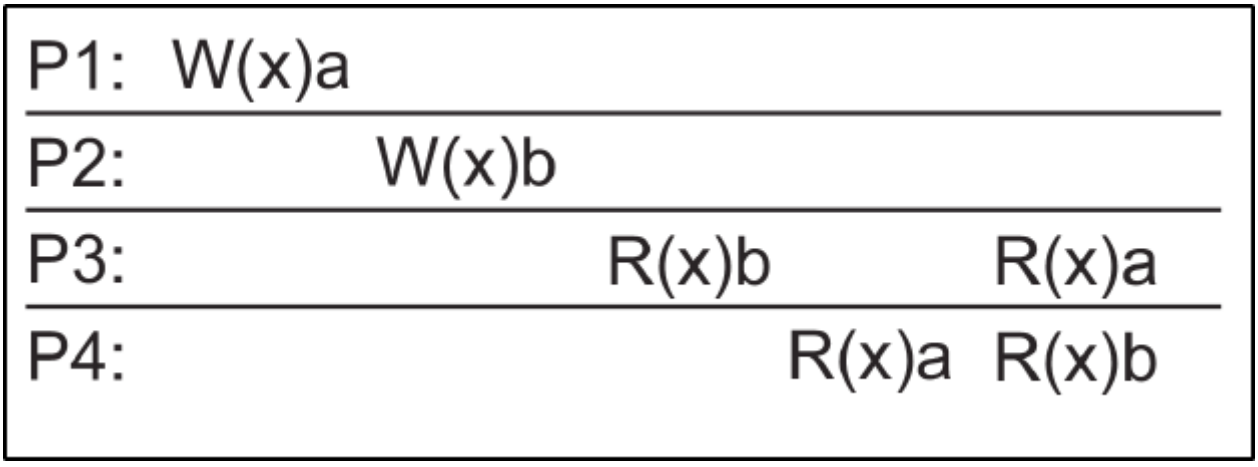
Sequentielle Konsistenz:



Prozess 1 schreibt a in x (W(x)a), danach schreibt Prozess 2 b in x (W(x)b) → b = aktuellstes Datum). Demzufolge gibt die Middleware vor, dass Prozess 3 und 4 zuerst b und danach a lesen sollen, um **sequentiell konsistent** zu sein

Oder: Es ist dann sequentiell Konsistent, wenn alle „Reader“ gleich lesen (zB zuerst b dann a, oder zuerst a dann b). Es kann auch sein, dass P1 zuerst begonnen hat a zu schreiben, jedoch P2 früher damit fertig war b zu schreiben und deshalb b zuerst gelesen wird. Schlussendlich kommt es darauf an, dass alle gleich gelesen werden.

Nicht-sequentielle Konsistenz:



5.4

5.4 Client-centric consistency models

Eine **starke Konsistenz** (strong consistency) kann möglicherweise nur garantiert werden, wenn Prozesse Transaktionen oder Synchronisationsvariablen verwenden

Monotone Lesekonsistenz (monotonic-read):

Wenn ein Prozess den Wert von Element x liest, gibt jede aufeinanderfolgende Leseoperation für x immer den gleichen Wert oder einen neueren Wert zurück

Monotone Schreibkonsistenz (monotonic-write):

Eine Schreiboperation durch einen Prozess auf Datenelement x wird vor jeder aufeinanderfolgenden Schreiboperation auf x durch denselben Prozess abgeschlossen

Lese die Schreibvorgangs Konsistenz (read your writes):

Eine Schreiboperation durch einen Prozess auf Datenelement x wird immer durch eine aufeinanderfolgende Leseoperation auf x durch denselben Prozess gesehen

Schreibvorgänge folgen Lesekonsistenz (writes follow reads):

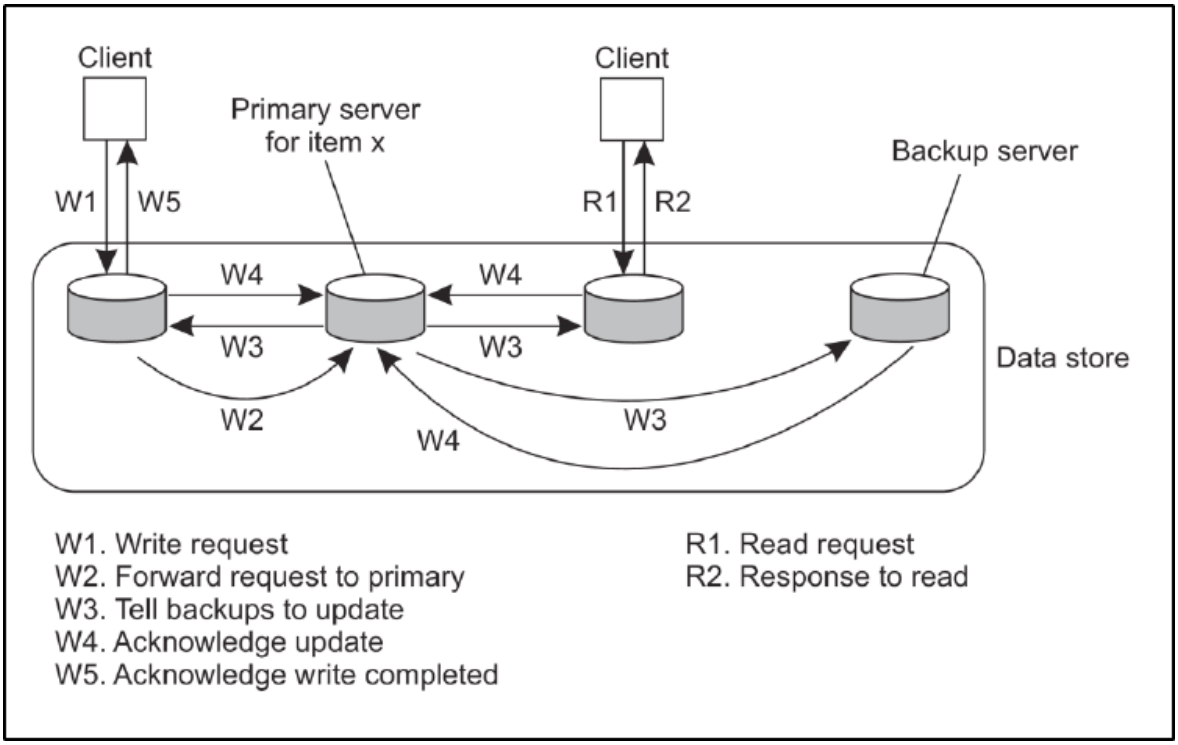
Aktualisierungen werden als Ergebnis früherer Lesevorgänge weitergegeben

5.5 Primary-backup consistency protocol

Schreibvorgänge an den Primärserver weiterleiten.

Es dauert lange, bis das Update fortgesetzt werden kann (blocking)

Der Client weiß nicht genau, ob Daten von mehreren Servern gesichert werden (non-blocking).

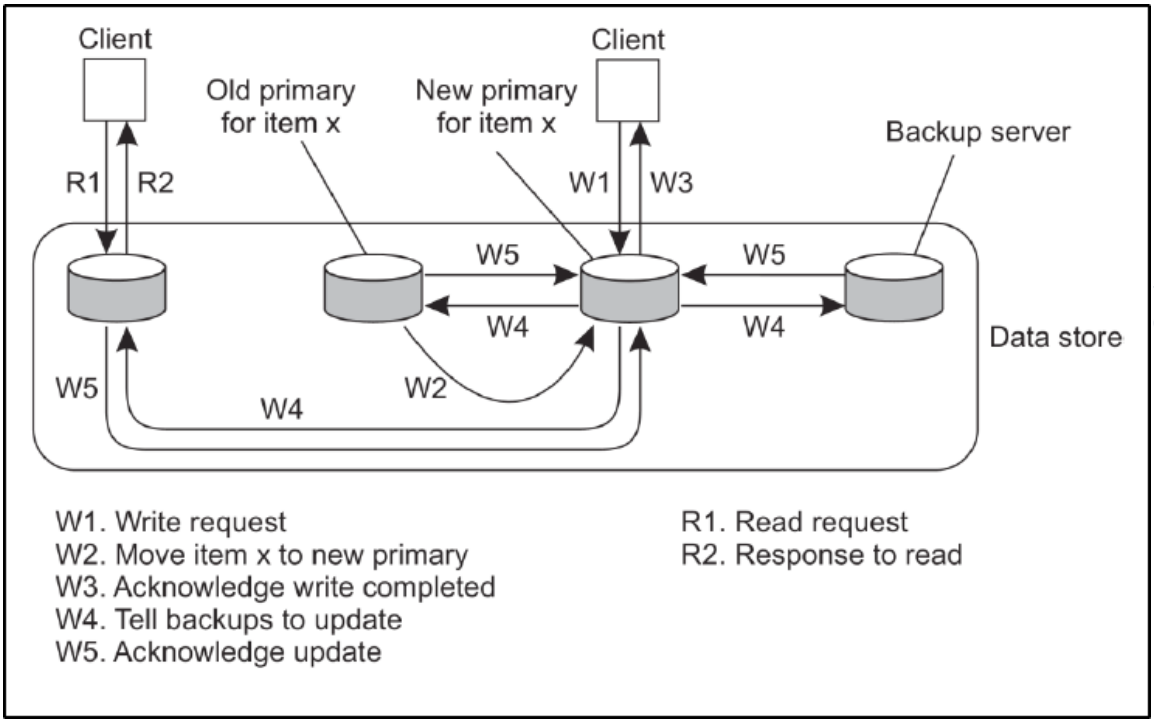


5.6 Local-write consistency protocol

Die primäre Kopie wird zwischen Prozessen migriert, um einen Schreibvorgang durchzuführen

Aufeinanderfolgende Schreibvorgänge können lokal ausgeführt werden

Kann für mobile Computer im getrennten Modus verwendet werden

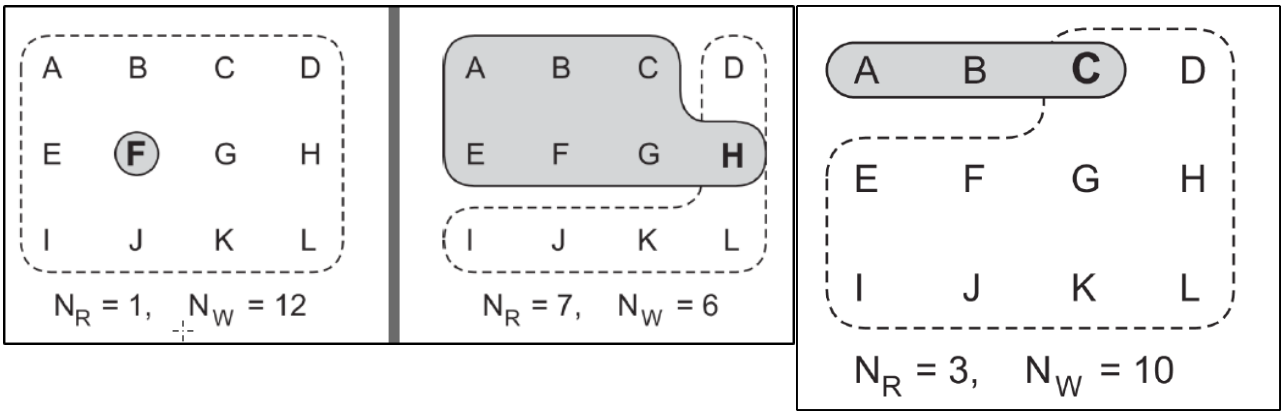


5.7 Quorum protocol

Requirements

- Bei Quorums geht es quasi immer um „Mehrheiten“ (jeder Buchstabe stellt zB einen Server dar, ein paar wurden als read-server und andere als write-server definiert)

- Read Quorum (Nr)
- Write Quorum (Nw)
- Nr + Nw > N (prevent read write conflicts)
- Nw > N / 2 (prevent write write conflicts)



6. Naming

6.1 Name .vs. entity .vs. access point .vs. address

Name:	bezieht sich auf Entität
Entität:	kann alles sein (Hosts, Drucker, Festplatten, Dateien, Prozesse, Benutzer, Postfächer, Webseiten, Nachrichten, Netzwerkverbindungen…)
Acces Point:	ist eine Zugriffsentität (access entity)
Address:	ist ein Name eines access point

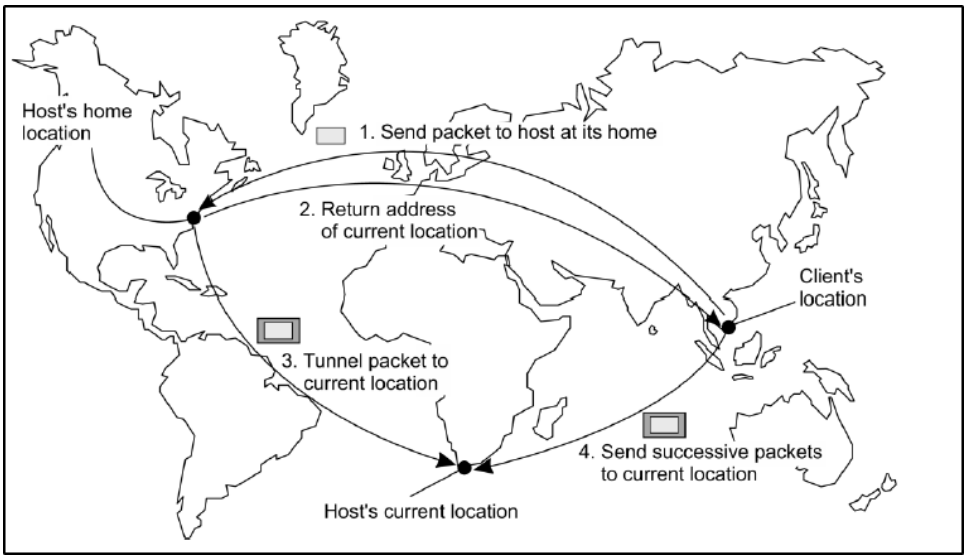
Type of Names: Addresses, Identifiers, Human-friendly names

6.2 True identifiers

- Ein identifier bezieht sich auf höchstens eine Entität
- Jede Entität wird höchstens einem identifier verwiesen
- Ein identifier bezieht sich immer auf dieselbe Entität (er wird niemals wiederverwendet).

6.3 Flat, unstructured naming

- Identifier sind eindeutig dargestellte Entitäten
- In vielen Fällen sind Bezeichner zufällige Bitfolgen
- Einfache Lösungen:
  - Rundfunk
    - Adressauflösungsprotokoll (ARP)
  - Weiterleitungszeiger
    - Die Entität wechselt von A nach B und hinterlässt einen Verweis auf ihre neue Position
- Heimbasierter (home-based) Ansatz



- Verteilte Hash-Tabellen (distributed hash tables - DHT)
- Hierarchischer Ansatz

6.4 Structured naming

Flat naming ist gut für Maschinen, nicht bequem für Menschen

Namenssysteme unterstützen strukturierte Namen

Namespace:

Organisieren von Namen in Namensräumen

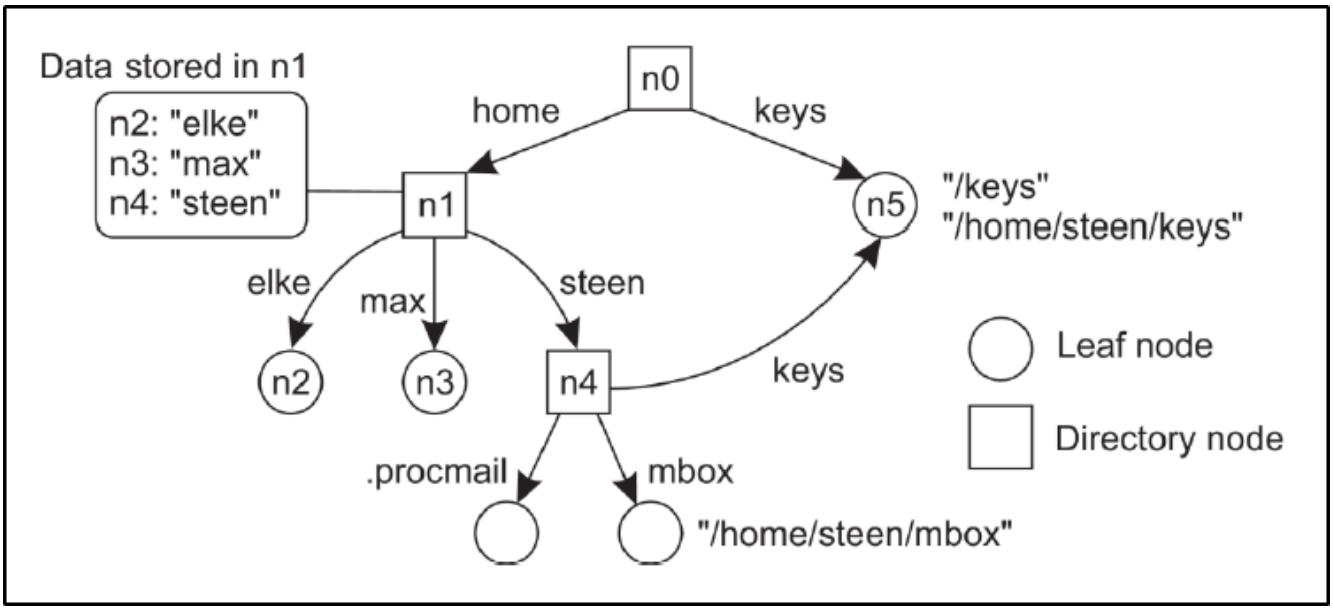
Leaf node:

Informationen über die Entität (z. B. Adresse)

Optional auch der Inhalt

Directory node:

Anzahl der ausgehenden Kanten



6.5 Attribute-based naming

Entitäten werden durch eine collection beschrieben (attribute, value)

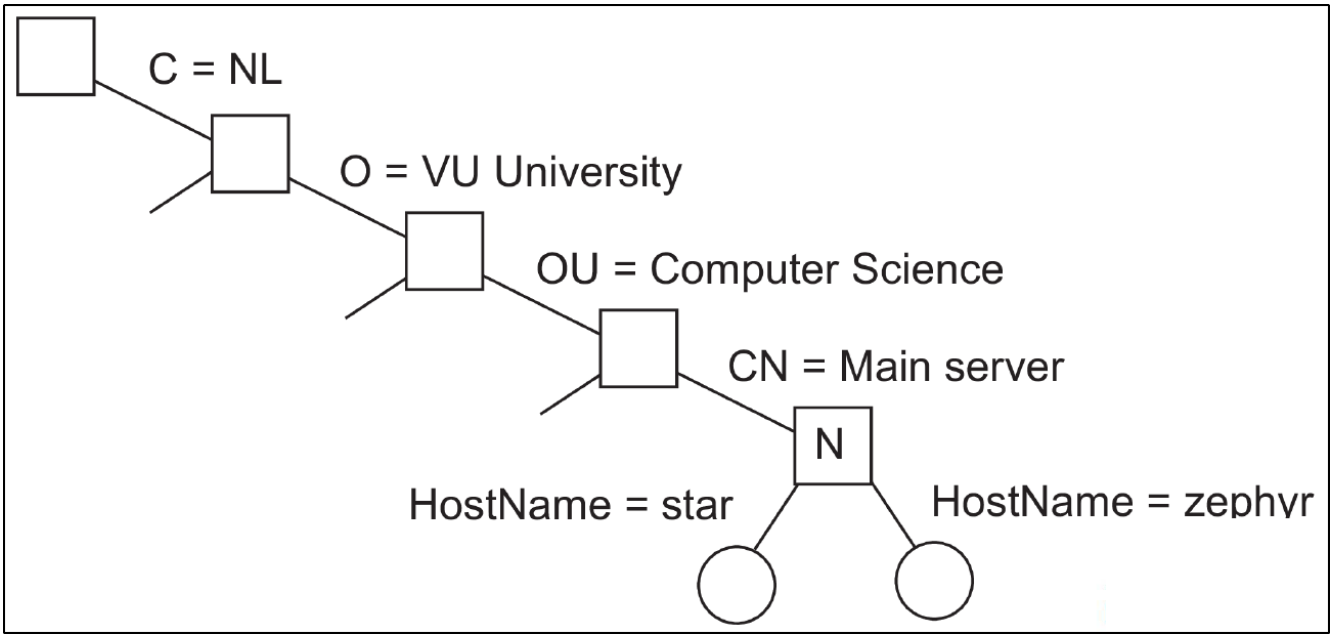
Attributbasierte Benennungssysteme, sogenannte **directory-services** (strukturierte Benennung: naming systems)

Suche nach Attributen

Resource Description Framework (RDF)

- Ressourcen als triplrts (Drillinge) beschrieben
- Subjekt, Prädikat, Objekt
- Person, Name Alice

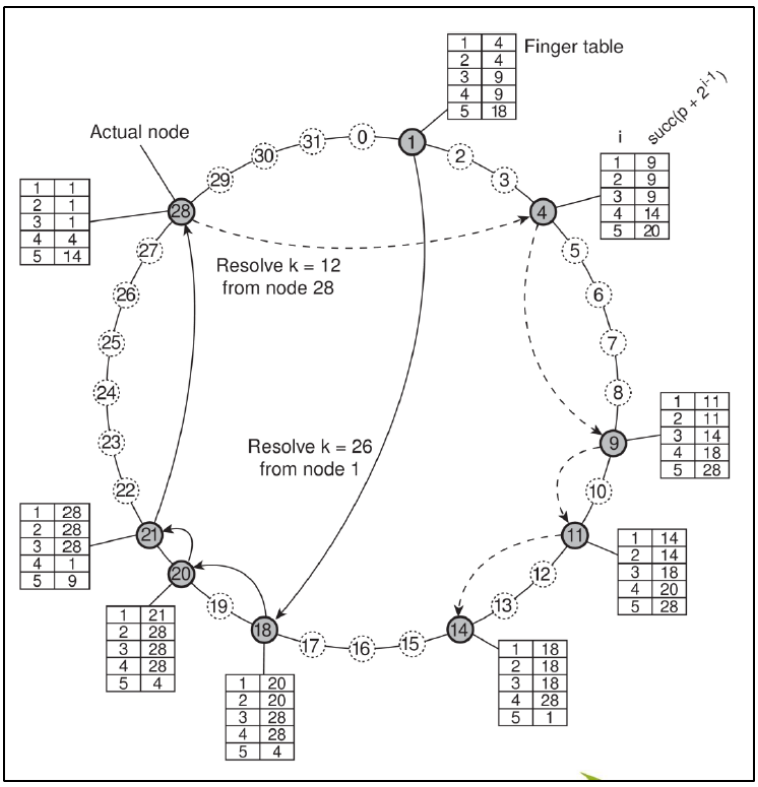
Directory Information Tree:



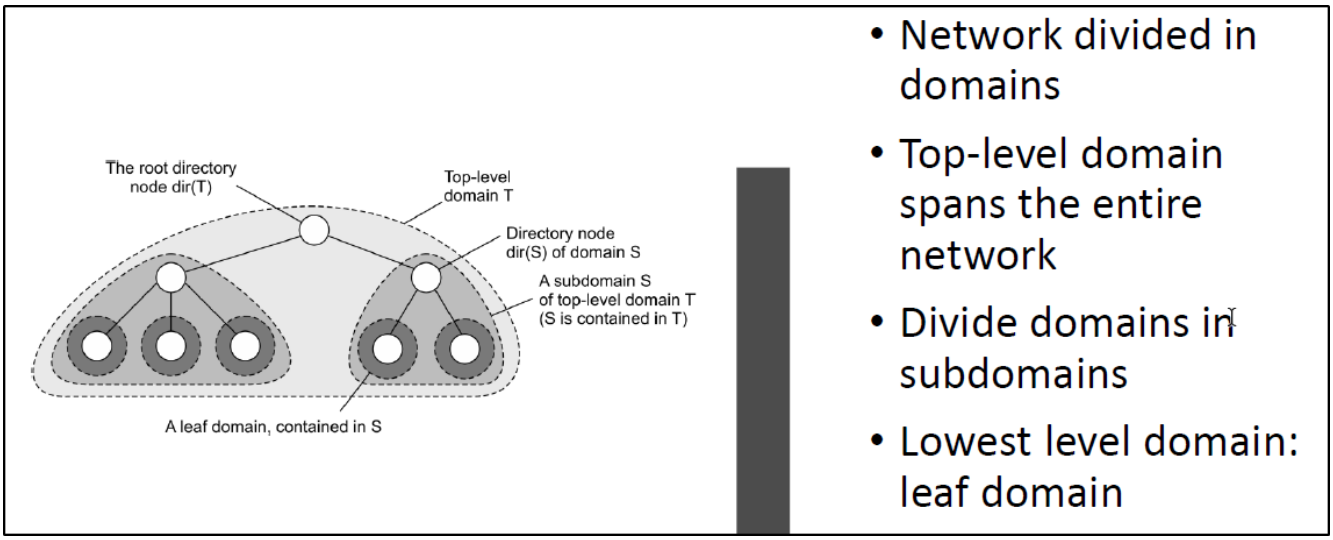
6.6 Distributed hash tables

Den Schlüssel k effizient in die Adresse von succ (k) auflösen

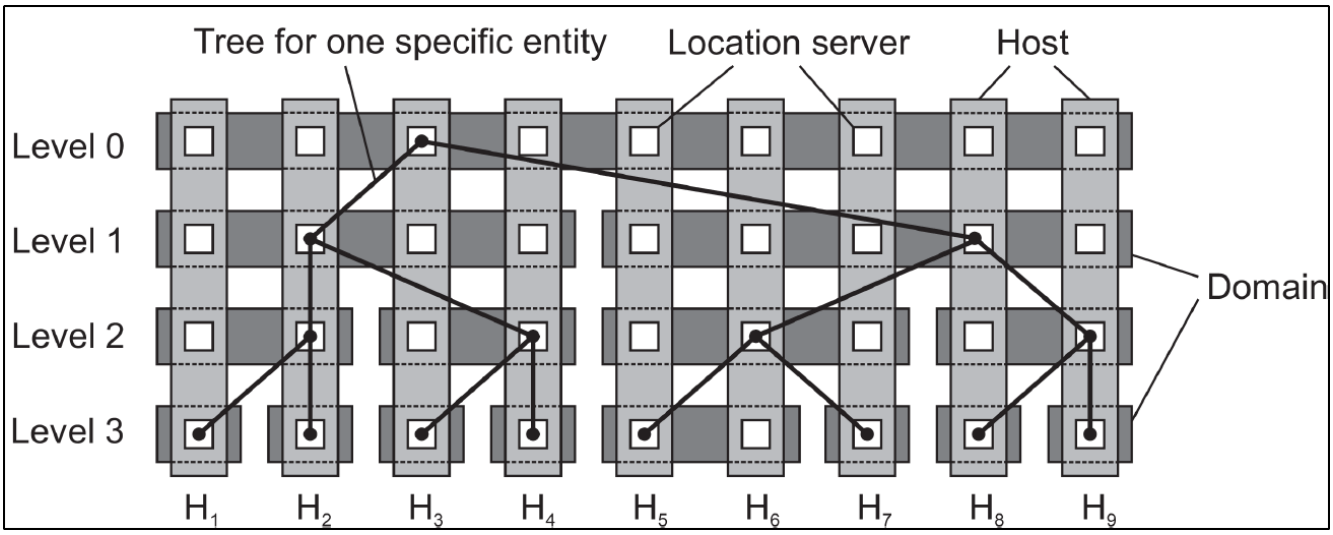
Fingertables, der vom chordsystem verwendet wird



6.7 Hierarchical location service

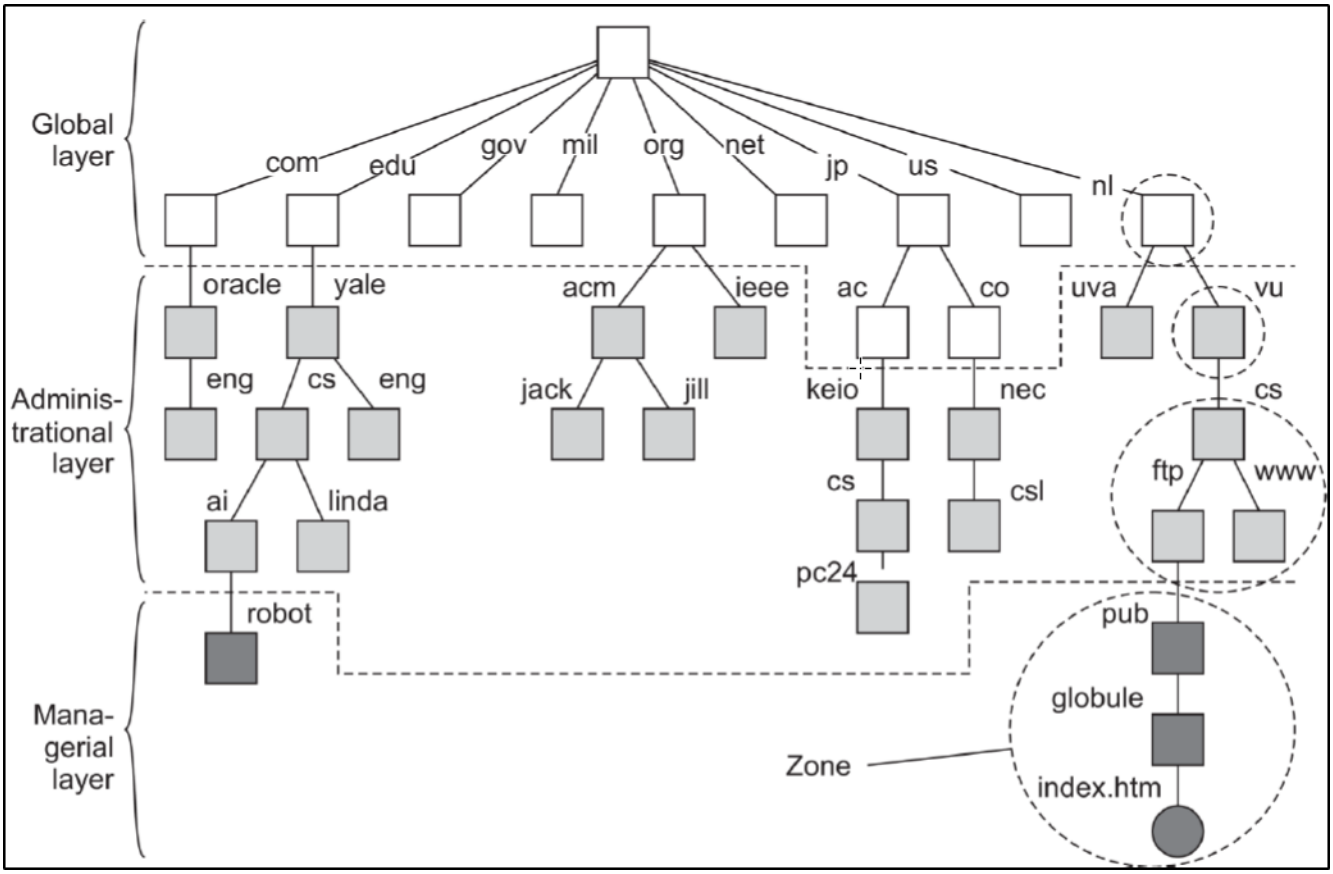


6.8 Scalability



6.9 DNS namespace

DNS Name space:



DNS Name Space – Resource Records:

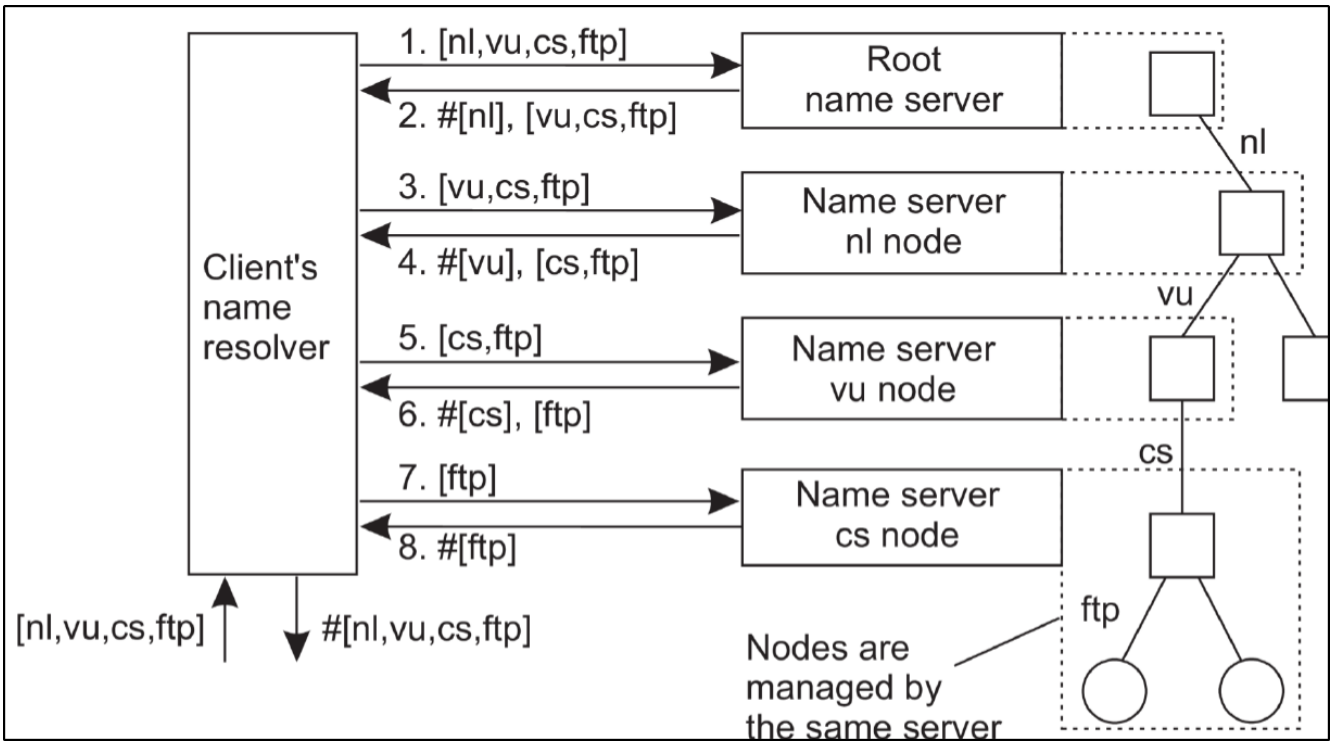
Type	Refers to	Description
SOA	Zone	Holds info on the zone
A	Host	IP Address of host
MX	Domain	Mail server to handle mail for this node
SRV	Domain	Server handling a specific service
NS	Zone	Name server for a zone
CNAME	Node	Symbolic link
PTR	Host	Canonical name of a host
HINFO	Host	Info on this host
TXT	Any kind	Any info

6.10 Layers and issues with naming

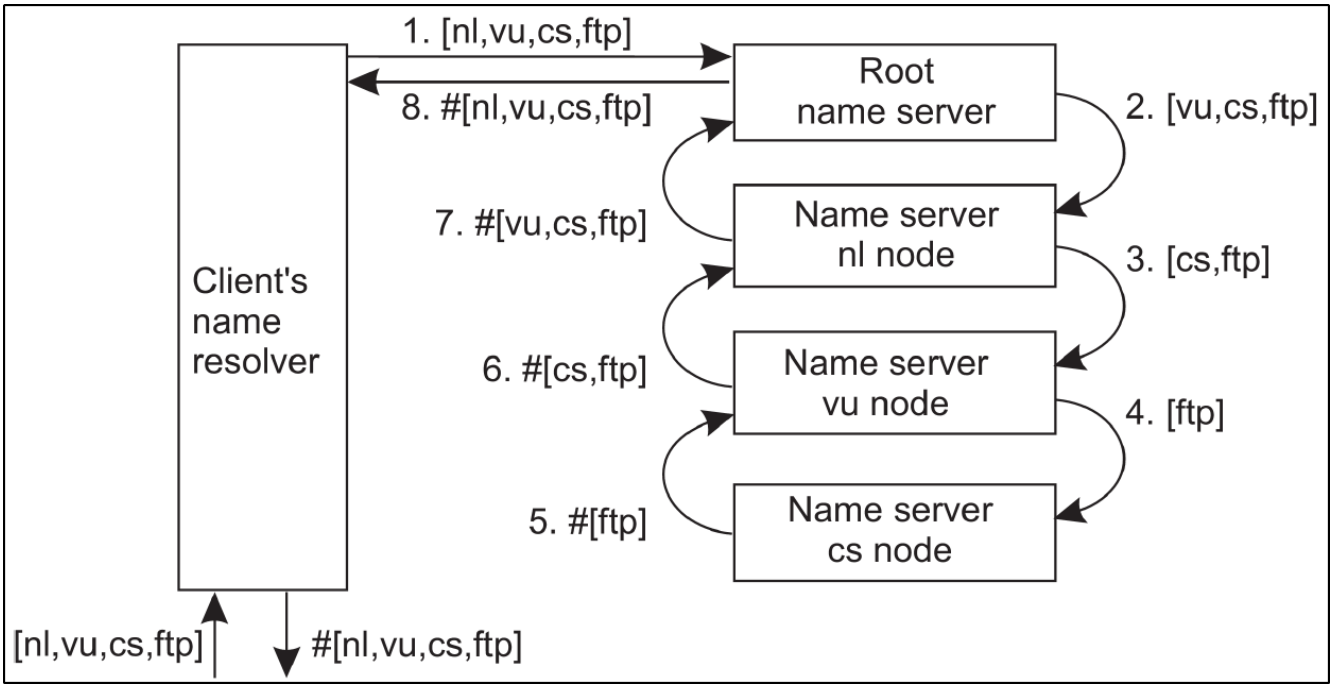
Issue	Global	Administrational	Managerial
Geographical Scale	Worldwide	Organization	Department
Number of nodes	Few	Many	Vast numbers
Responsiveness for lookups	Seconds	Milliseconds	Immediate
Update propagation	Lazy	Immediate	Immediate
Number of replicas	Many	None or few	None
Client-side caching	Yes	Yes	Sometimes

6.11 Iterative .vs. recursive name resolution

Iterative:



Recursive:



7. Fault Tolerance

7.1 Concepts:

Verfügbarkeit (Availability)

– ready to be used

Vertrauenswürdigkeit/Zuverlässigkeit (Reliability)

- can run continuisly without failure

Sicherheit/Zuverlässlichkeit (Safety)

- when temporary failures happen, nothing catastrophic happens

Wartbarkeit (Maintainability)

- easy to repair

7.2 Failure Models:

Crash failure (Absturzfehler):

Der Server wird angehalten, funktioniert ordnungsgemäß bis er angehalten wird

Omission failure (Auslassungsfehler):

Der Server reagiert nicht auf eingehende Anfragen

Timing failure:

Die Antwort liegt außerhalb eines bestimmten Zeitintervalls

Response failure („Antwortfehler“):

Antwort ist falsch

Arbitrary failure (Willkürliches Versagen):

kann jederzeit beliebige Antworten liefern

7.3 Redundancy

Informationsredundanz (Information redundancy):

zusätzliche Bits, um die Wiederherstellung von verstümmelten Bits zu ermöglichen, z.B.: Hamming-Code

Zeitredundanz (Time redundancy):

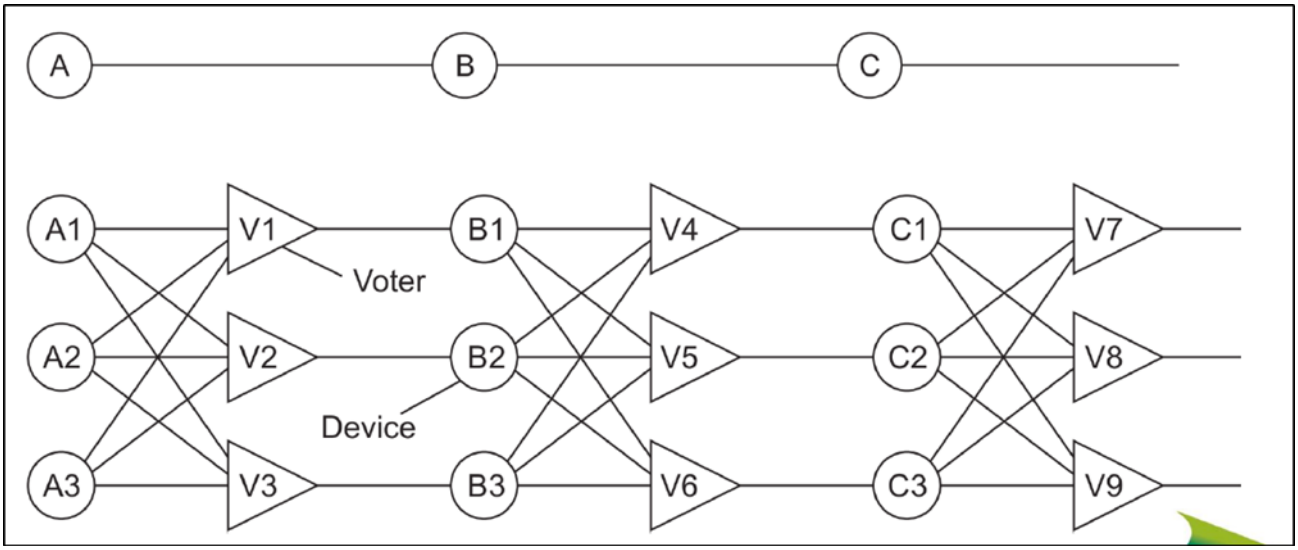
Aktion erneut ausführen, z.B.: Wenn die Transaktion abgebrochen wird, ist das Wiederherstellen nützlich, wenn Fehler vorübergehend oder zeitweise auftreten

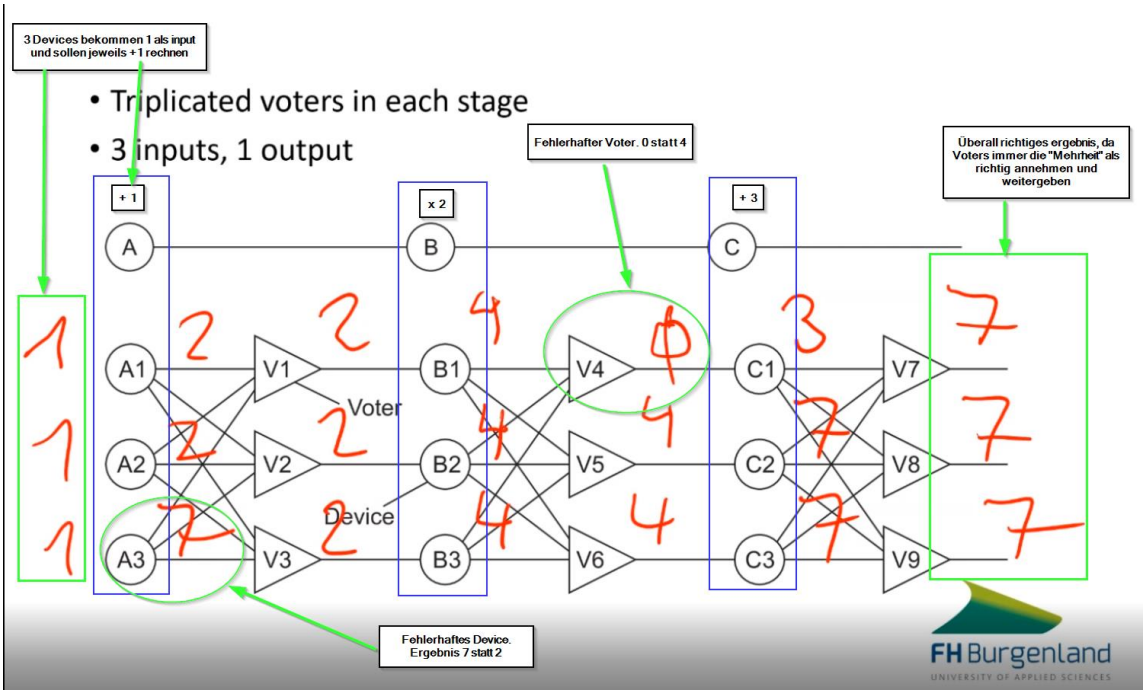
Physische Redundanz (Physical redundancy):

Bekannt für Fehlertoleranzbiologie (Säugetiere), Flugzeuge, Sportschiedsrichter, elektronische Schaltungen
> Azure-Speicher

7.4 Tripple Modular Redundancy

„Dreifache Wähler (voter) in jeder Phase (A, B, C)“ – 3 Inputs > 1 Output





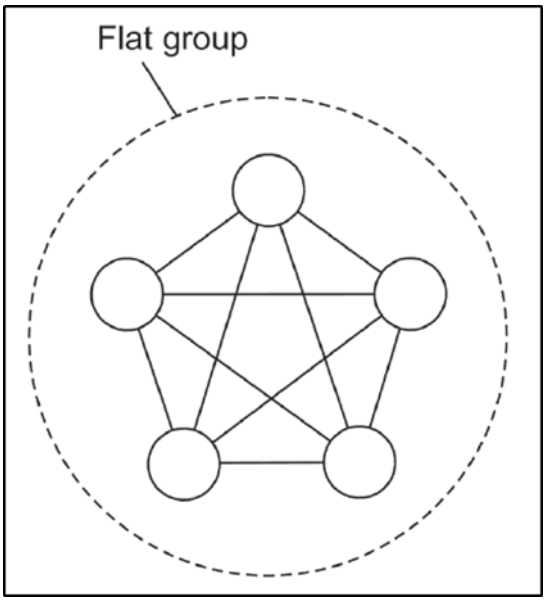
Prozessresilienz (Process resiliency):

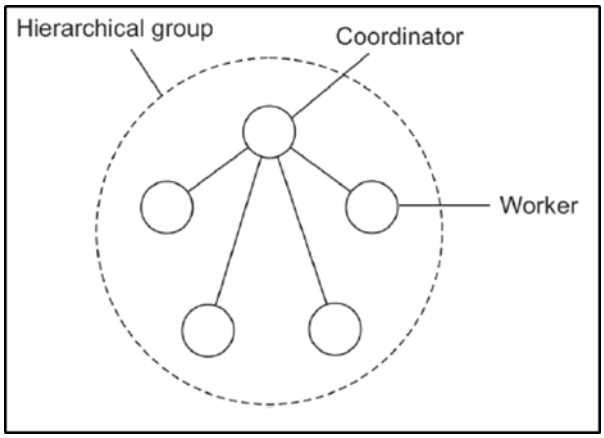
identische Prozesse in einer Gruppe zusammenfassen/organisieren

Eine gruppe von Prozessen als einzelne Abstraktion verwenden

Flache Gruppen

Hierarchische Gruppen





7.4 Paxos

Protokollfamilie, Basis für „state machine replication“

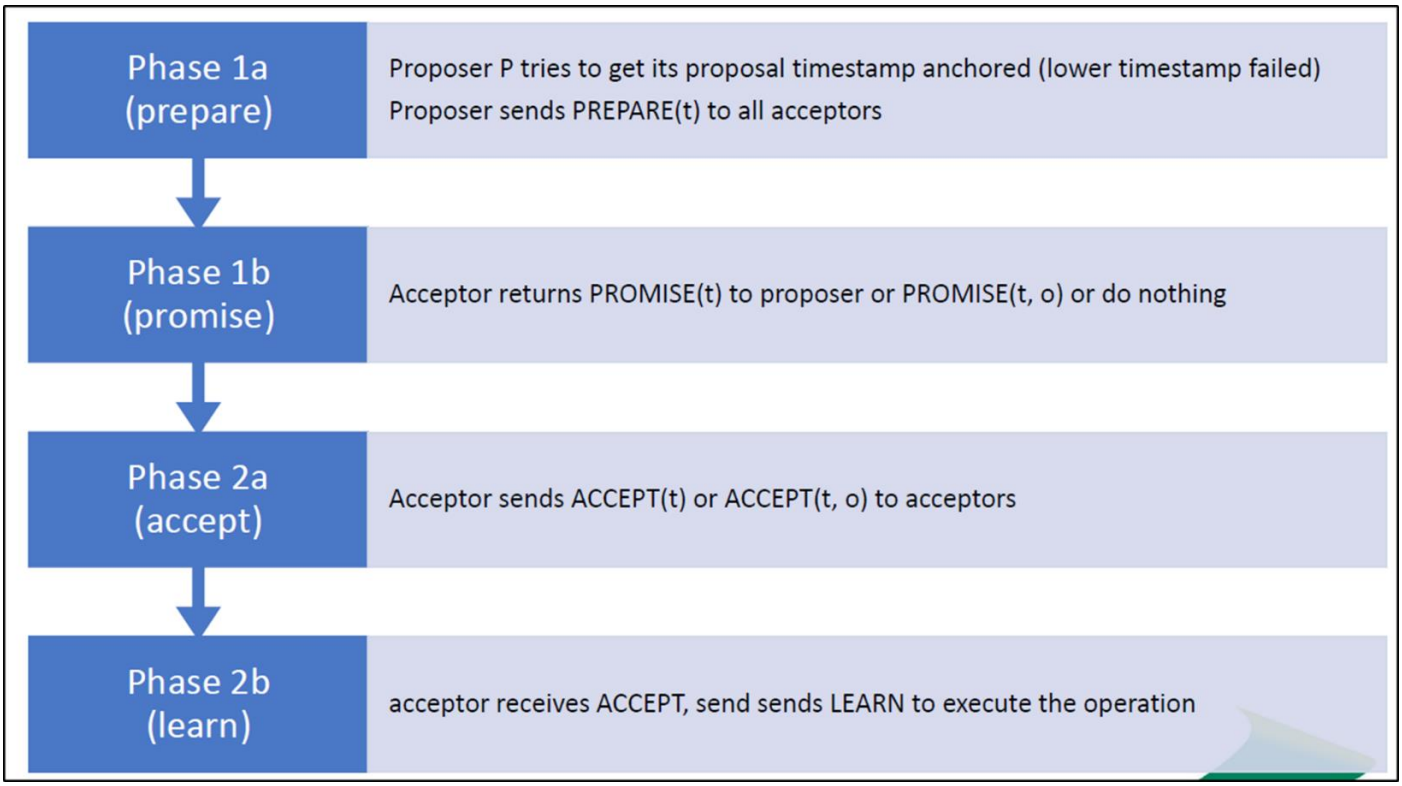
„schwache“ Annahmen:

- Das verteilte System ist teilweise synchron (kann asynchron sein)
- Die Kommunikation zwischen Prozessen kann unzuverlässig sein
- Beschädigte Nachrichten können erkannt werden
- Alle Operationen sind deterministisch: Sobald die Ausführung gestartet ist, ist bekannt, was sie tun wird
- Prozesse können Absturzfehler aufweisen, aber keine willkürlichen Fehler, noch kollidieren Prozesse

Ablauf:

1. Leading proposer empfängt Anfragen vom Client und leitet Anfragen an alle acceptors weiter
2. Der nonleading proposer sendet einen proposal an alle acceptors, um die erforderliche Operation zu akzeptieren
3. Der acceptor sendet eine learn message
4. Der learner empfängt eine Nachricht von einer Mehrheit der acceptors und führt sie aus.

Phasen:



7.5 Recovery

Backward recovery:

Das System aus dem Fehlerzustand zurück in einen zuvor korrekten Zustand zu bringen, verwenden von Checkpoints (Checkpoints können teuer sein). ZB Server crashed und fährt wieder hoch mit den Daten des letzt-gültigen Zustandes, einige Daten (jene dazwischen) könnten verloren gehen.

Forward recovery:

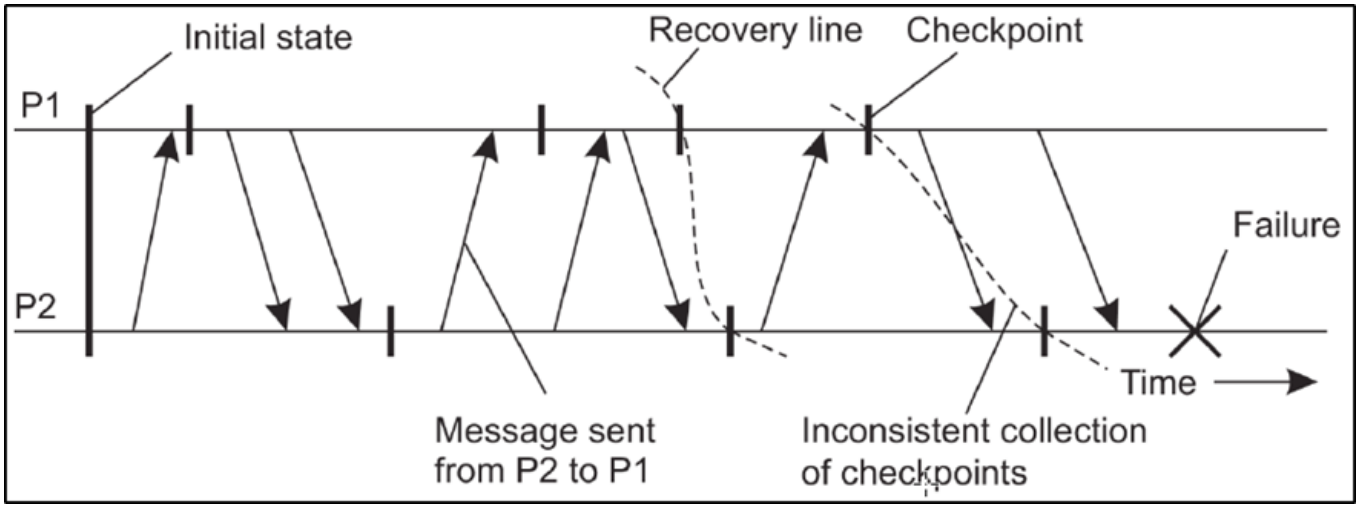
Das System wieder in einen korrekten neuen Zustand zurückbringen. ZB 1 von 3 Servern crashed, bekommt nach dem booten alle Daten von den anderen Servern und ist wieder am aktuellsten Stand.

7.5 Erasure correction (Löschkorrektur)

Das fehlende Paket wird aus anderen erfolgreich gelieferten Paketen erstellt. ZB (korrupte/fehlende) „Pakete“ können durch Vorhandensein anderer Pakete wiederhergestellt werden.

7.6 Checkpopinting

Regelmäßige Speicherung des Zustands



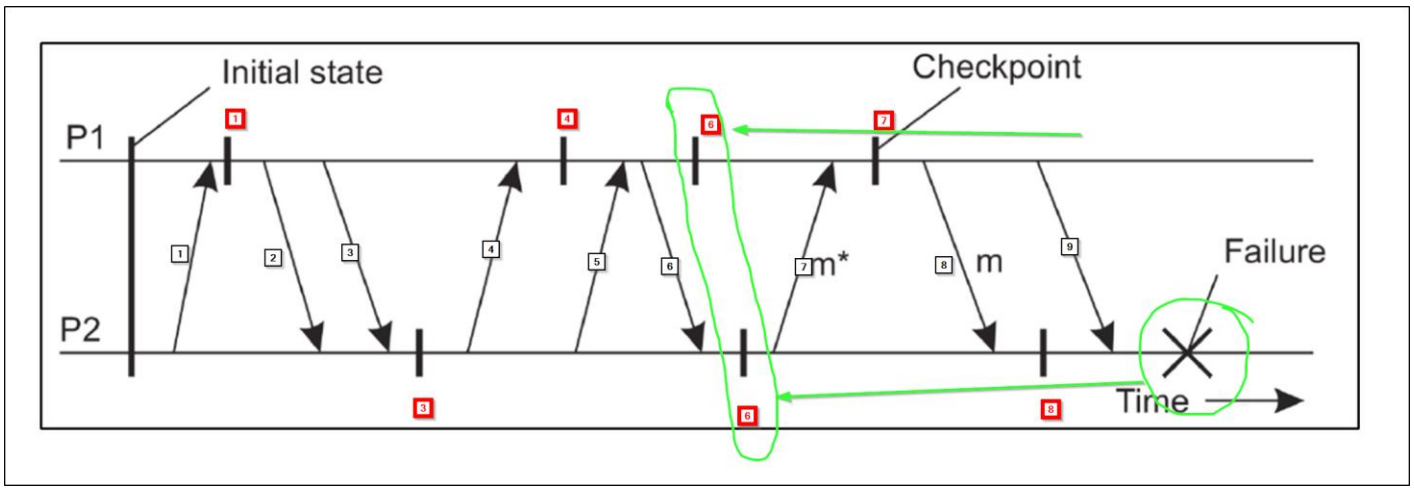
Coordinated Checkpoint:

Alle Prozesse synchronisieren den Status und schreiben ihn gemeinsam in den lokalen Speicher

Zwei Phasen Commit-protocol

Independent Checkpoint:

Kann zu einem Dominoeffekt mit kaskadierendem Rollback für globale Konsistenz führen, damit global konsistent (failure nach 9, rollback wo beide Prozesse gleich sind = CKP 6).

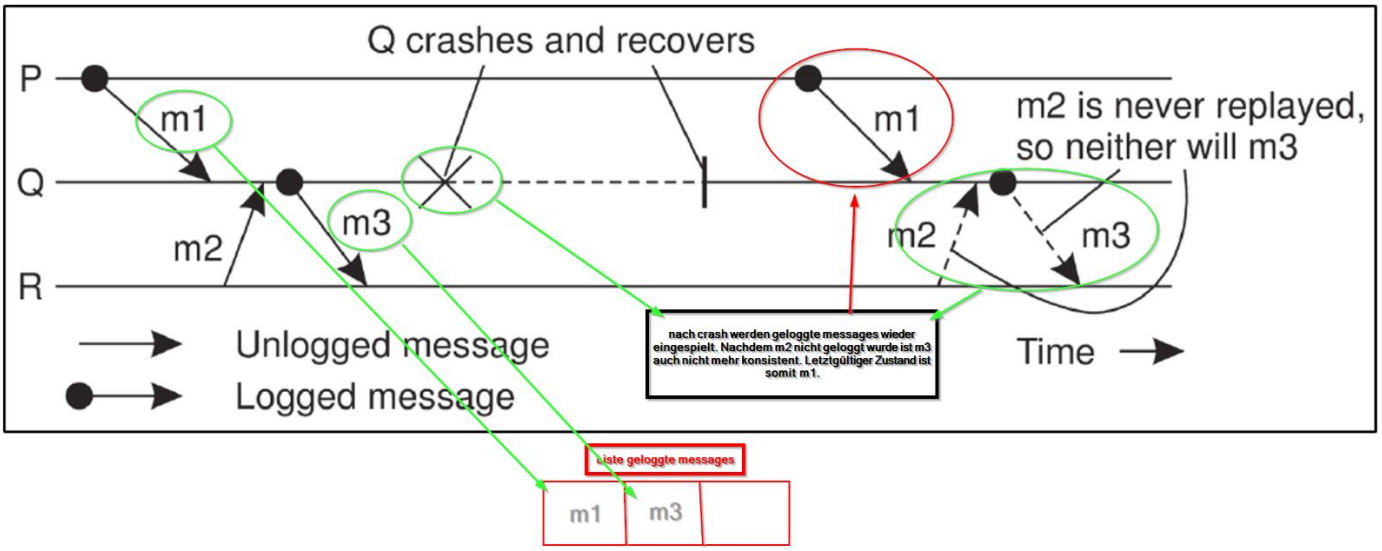


7.8 Message Logging:

Statt Prozess-states (Checkpoints) kann man auch messages schreiben. Reduzieren Checkpoints (teurer Betrieb), Wiederherstellung (recovery) jedoch weiterhin möglich

Transmission messages können wiedergegeben werden

Stückweise deterministisches Ausführungsmodell (piecewise deterministic execution model)

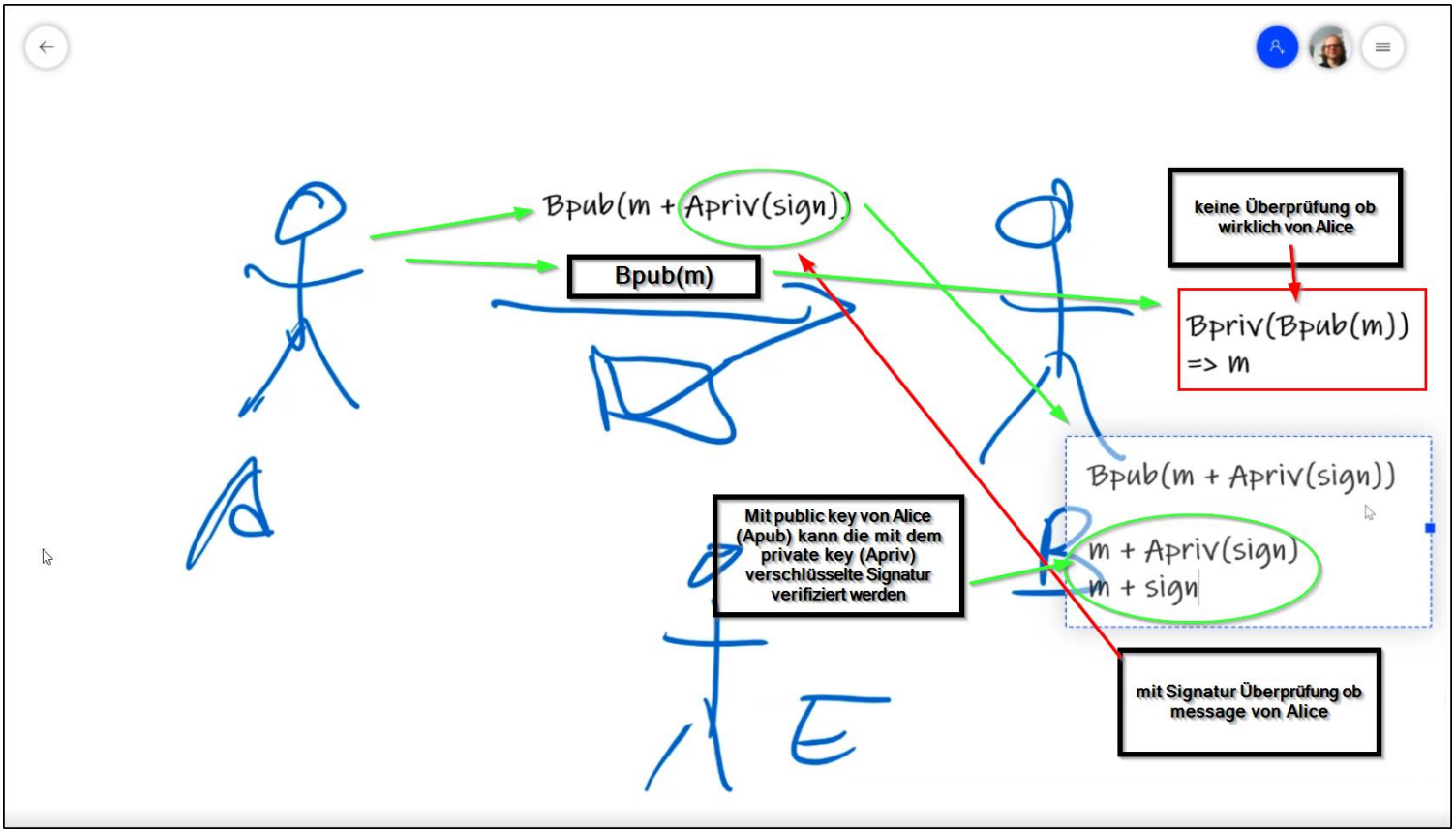


8. Security

Mögliche Attacken für Webapplikationen:

- Encoding
- SQL Injection
- Cross-Site Request Forgery (XSRF)
- Social Attacks

8.1 Encryption / Cryptography



Symmetrische Verschlüsselung:

DES (Data Encryption Standard)

> insecure → 56 Bit Keys

Tripple-DES

> 168 Bit Key (effektiv aber 112)

AES (Advanced Encryption Standard)

> 128, 192, 256 Bit Keys

Rijandel

Asymmetrische Verschlüsselung:

RSA (Rivest, Shamir, Adleman)

> e-commerce protocols

> 1024-4096 Bit Keys (1977)

DSA

EC-Dsa (Elliptic Curve)

Hash Funktionen:

Create fixed-length hash

MD5, SHA1, SHA256, SHA512

8.2 Authentication

Identifizieren einer Person

Single-FA (single factor authentication)

MFA (multi factor authentication)

„You are who you say you are“

8.3 Authorization:

Zugriffsrichtlinien

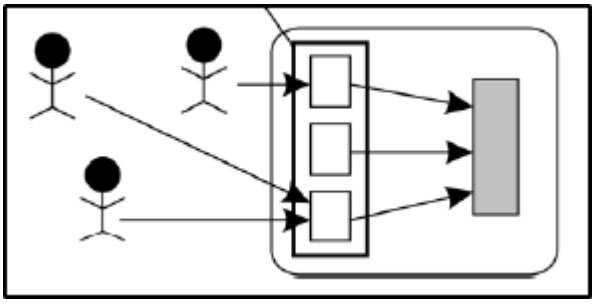
Access tokens: Keys, Zertifikate, tickets

RBAC: Role-based Access Control

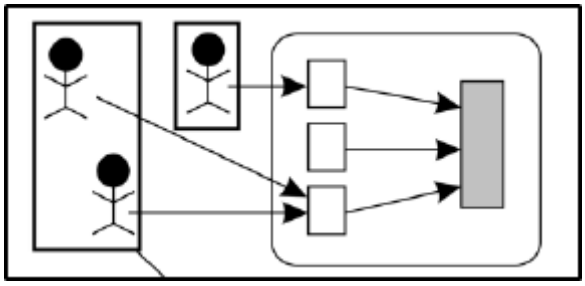
ACL: Access Control List

8.4 Protection against threats:

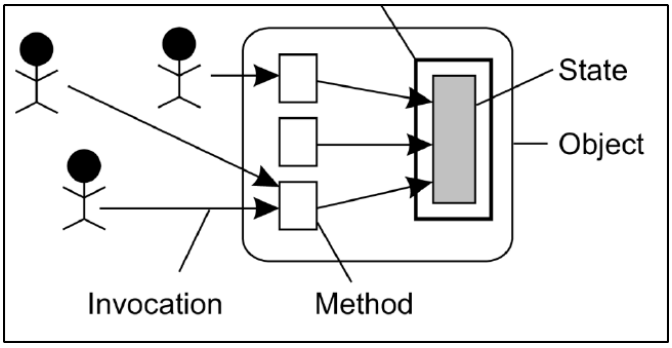
Daten sind vor unbefugten Aufrufen (unauthorized invocations) geschützt



Daten werden durch Überprüfen der Rolle eines Aufrufers (unauthorized users) geschützt .



Daten sind vor falschen oder ungültigen Vorgängen (invalid operations) geschützt.



Fehlt:

- Auditing