1678 Verteilte Systeme Zusammenfassung

KE 1 Grundlagen Verteilter Systeme

1. Definition Verteiltes System

- Ein verteiltes System ist eine Ansammlung unabhängiger Computer, die den Benutzer wie **ein einzelnes kohärentes System** erscheinen.
- Ein verteiltes System ist ein System, in dem sich HW- od. SW-Komponenten auf vernetzten Computern befinden und nur über den **Austausch von Nachrichten kommunizieren** und ihre Aktionen koordinieren. Aus dieser Definition ergeben sich folgende Konsequenzen:
  - Nebenläufigkeit der Programmausführung.
  - Keine globale Uhr -> es gibt kein globales Konzept einer genauen Uhr (da Kommunikation ausschließlich über das Senden von Nachrichten erfolgt).
  - Unabhängige Ausfälle -> jede Komponente des Systems kann unabhängig von den anderen ausfallen, während die anderen weiterhin funktionieren (und womöglich lange od. überhaupt nie etwas davon merken).
- Ziel ist die gemeinsame Nutzung von Ressourcen (HW u. SW)

2. Beispiele für Verteilte Anwendungen/Systeme

- Banksysteme
- Flug/Reise-Buchungssysteme
- verteilte Datenbanken
- High Availability – Cluster

3. Vorteile Verteilter Systeme gegenüber Großrechnern

- Wirtschaftlichkeit
- Geschwindigkeit
- Verteiltheit - dadurch sind u.U. natürlichere Problemlösungen möglich ?
- Zuverlässigkeit durch Redundanz
- Skalierbarkeit

4. Nachteile von Verteilten Systemen

- Komplexere Software
- Kommunikationsprobleme sind möglich - dadurch schlechtere Performance, Verlust von Nachrichten
- Nur schwache Schutzvorkehrungen - Ambivalenz zwischen Sicherheit und Einfachheit des Zugriffs auf Ressourcen

5. Eigenschaften Verteilter Systeme

- Rechnertyp und der Kommunikationsarten dieser Rechner
- Interner Aufbau
- Konsistenz und einheitliche Arbeit mit dem Verteilten System
- Skalierbarkeit
- Ausfallsicherheit

6. Middleware

- Ein verteiltes System wird auch manchmal als Middleware bezeichnet. Die Middleware ist als Schicht zu sehen, welchem dem Anwender/der Anwendung verbirgt, dass es sich um ein, auf mehreren Rechnern, laufendes System handelt.

7. Ziele eines Verteilten Systems

- Zugriff auf lokale und entfernte Ressourcen zu vereinfachen
- Verteilungstransparenz (dem Benutzer/Anwendung soll das System wie ein Einprozessorsystem vorkommen / Ein System, dass seine Verteiltheit vollständig vor dem Benutzer verbirgt und wie ein Einprozessorsystem erscheint, heißt transparent, da die Implementierung für den Benutzer durchsichtig - im Sinne von unsichtbar - ist.)
  Arten von Transparenz:
  - *Ortstransparenz* – erlaubt den Zugriff auf die Ressourcen, ohne dass man ihre Position/Ort kennt.
  - *Fehlertransparenz* – erlaubt das Verbergen von Fehlern, sodass Benutzer ihre Aufgaben erledigen können, auch wenn HW- od. SW-Komponenten ausgefallen sind.
  - *Zugriffstransparenz* – ermöglicht den Zugriff auf lokale und entfernte Ressourcen unter Verwendung identischer Operatoren.
  - *Nebenläufigkeitstransparenz* – erlaubt, dass mehrere Prozesse gleichzeitig mit denselben gemeinsam genutzten Ressourcen arbeiten, ohne sich gegenseitig zu stören.
  - *Replikationstransparenz* – erlaubt, dass mehrere Instanzen von Ressourcen verwendet werden, um die Zuverlässigkeit und die Leistung zu verbessern, ohne dass die Benutzer wissen, dass Repliken verwendet werden.
  - *Migrationstransparenz* – erlaubt das Verschieben von Ressourcen und Clients innerhalb eines Systems (Name der Ressource bleibt gleich, aber **geringe** Ausfallzeit)
  - *Relokationstransparenz* – Verbirgt, dass eine Ressource an einen anderen Ort verschoben werden kann, während sie benutzt wird (Name der Ressource bleibt gleich, aber **keine** Ausfallzeit)

Nicht im Buch:

- *Leistungstransparenz* – erlaubt, dass das System neu konfiguriert wird, um die Leistung zu verbessern, wenn die Last variiert.
- *Skalierungstransparenz* – erlaubt, dass sich das System und Applikationen vergrößern, ohne dass die Systemstruktur oder die Applikationsalgorithmen geändert werden müssen.

- Offenheit (ein offenes Verteiltes System bietet Dienste nach Standardregeln an)

  Ziele:
  - Spezifizierung der Dienste durch Schnittstellen (**I**nterface **D**efinition **L**anguage)
  - Spezifizierung sollte
    - Vollständig und Neutral sein
    - Interoperabilität (Zusammenspiel von Komponenten verschiedener Hersteller)
    - Portabilität (Anwendung von einem Verteiltem System zum anderen übertragen ohne Änderungen)
    - Konfiguration verschiedener Komponenten
    - Hinzufügen von Komponenten
    - Ersetzen von Komponenten
    - Flexibilität (kein monolithischer Ansatz)

- Skalierbarkeit
  - Größe
  - Geografische Ausdehnung
  - Administration
  - Skalierungsprobleme:
    - Zentralisierte Daten (z.B. einzige DB)
    - Zentralisierte Dienste (z.B. einziger Server)
    - Zentralisierte Algorithmen (z.B. Routing (Dijkstra-Alg.))
    - Lösung durch Verwendung von dezentralisierten Algorithmen

8. Dezentralisierte Algorithmen

- Kein Computer hat vollständige Informationen über den Systemstatus.
- Computer entscheiden nur aufgrund lokaler Information.
- Der Ausfall eines Computers schädigt nicht den Algorithmus.
- Es wird nicht implizit angenommen, dass es eine globale Uhr gibt.

9. Skalierungstechniken

Meistens handelt es sich um Leistungsprobleme, die man mit Hilfe von Skalierung umgehen möchte.

- Verbergen der Latenzzeiten der Kommunikation

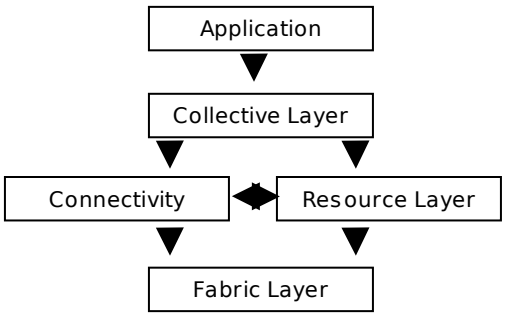
- Replikation
- Verteilung (z.B. DNS)

10. Klassen von Verteilten Systemen

- Cluster-Computer (Homogenität / z.B. MS SQL Cluster)
  - Hardware aus ähnlichen Rechnern oder Servern
  - Gleiches Betriebssystem
- Grid-Computer (Heterogenität / z.B. WWW)
  - Sehr unterschiedliche Hardware, Software und Netzwerktechnik

11. Grid-Computer Architekturvorschlag

- Vier Schichtenmodell von Foster:
- Fabric Layer (Strukturschicht) →Schnittstellen zu lokalen Ressourcen eines Standortes
- Connectivity Layer (Verbindungsschicht)→Kommunikationsprotokolle zur Unterstützung von Grid-Transaktionen, die die Verwendung mehrerer Ressourcen umfassen, z.B. Daten zwischen Ressourcen austauschen
- Resource Layer (Ressourcenschicht)→Verwaltung einzelner Ressourcen, z.B. Abruf von Konfigurationsinformationen einer Resource
- Collective Layer (gemeinsame Schicht)→Dienste zur Suche nach Ressourcen, Zuweisung und Zeitplannung von Aufträgen
- Application Layer (Anwendungsschicht)→Anwendungen welche in virtuellen Organisationen laufen und die Grid-Umgebung nutzen
- Connectivity Layer + Resource Layer = Grid- Middelware



12.

Eigenschaften von Transaktionen

- **A**tomic: die Transaktion wird ganz oder gar nicht ausgeführt
- **C**onsistent: keine Verletzung von Systeminvarianten, z.B. Gelderhaltung bei Banküberweisung
- **I**solated: parallele Transaktionen beeinflussen sich nicht
- **D**urable: die Änderungen nach einem erfolgreichem Abarbeiten (commit) der Transaktion sind dauerhaft

13. Verteilte Pervasive Systeme

- engl. *pervasive* – durchdringend, um sich greifend
- ein *verteiltes pervasives System ist Teil unserer Umgebung*
- *bestehend aus mobilen kleinen mit Funkverbindung ausgestatteten Geräten*
- *keine Administration*
- *Instabilität u.a. wegen Ortsänderung*
- *3 Anforderungen an pervasive Anwendungen:*
  - Erfassung kontextueller Änderung (Ortsänderung)
  - Unterstützung von Ad-hoc-Zusammensetzung (Veränderung des Anwendungsangebots abhängig vom Ort)
  - Gemeinsame Nutzung von Informationen (auch Bereitstellung) als Standard
- *Beispiele:*
  - Haus- Und Multimediasysteme
  - *Informationssyteme im Gesundheitswesen z.B. BAN (Body Area Network)*
  - *Sensornetze*

14. Softwarearchitektur

- *Definition:* Betrachtung der logischen Anordnung der Softwarekomponenten
- Komponente:
  - *Definition*: modulare Einheit mit wohldefinierten erforderlichen und bereitgestellten Schnittstellen, welche in Ihrer Umgebung einsetzbar ist
- Architekturstil:
  - Komponenten
  - Art der Verbindung der Komponenten
  - Daten welche zwischen den Komponenten ausgetauscht werden
  - wie die Komponenten zu einem System konfiguriert werden
- Konnektor:
  - *Definition:* Mechanismus, welcher die Kommunikation, Koordination oder Kooperation zwischen Komponenten vermittelt

15. Softwarearchitekturstile

- Geschichtete Architektur, z.B. Netzwerkprotokollstapel; Abfragen werden nach unten weitergeleitet und Anworten nach oben
- Objektbasierte Architektur, z.B. Client-Server-Architektur Kommunikation der einzelnen Objekte/Komponenten über RPC (Remote Procedure Call)
- Datenzentrierte Architektur, Prozesse kommunizieren über Daten, z.B. über Dateien auf einem gemeinsamen Repository (engl. Ablage, Behälter)
- Ereignisbasierte Architektur, z.B. Publish/Subscribe-Systeme, Prozesse kommunizieren über Ereignisse

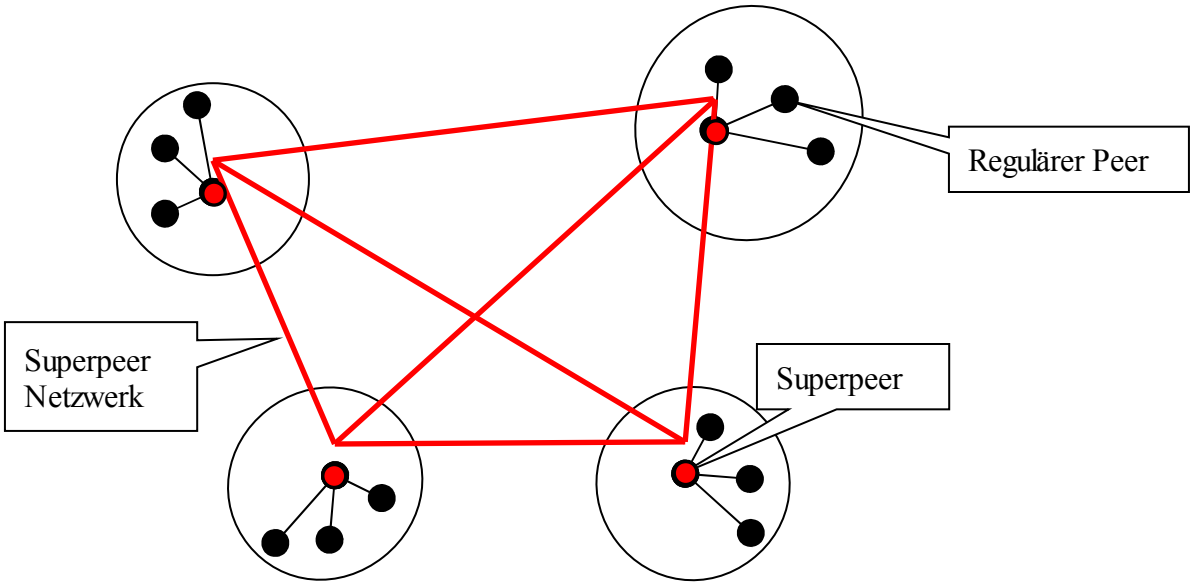
16. Zentralisierte Systemarchitekturen

- Client-Server Modell
  - Request/Reply Verhalten
  - Server bietet Dienste an, z.B. DNS (Domain Name System)
  - Client fordert Dienste auf Server an z.B. will er www.ard.de in IP umwandeln lassen und fordert hierzu den DNS-Dienst auf dem Server an
  - Ablauf:
    - Client sendet Anfrage
    - Server nimmt diese entgegen
    - Server verarbeitet die Anfrage
    - Server sendet Antwort
    - Client empfängt Antwort
  - Idempotent: Anfragen, welche beliebig oft gesendet werden können, wie in unserem Beispiel.
  - Genau einmal-Semantik
  - Mindestens einmal-Semantik (= Idempotent)
  - Höchstens einmal-Semantik
  - C/S ist oft als Schichtarchitektur:
    - Benutzerschnittstellenebene (Darstellung der Anwendung)
    - Verarbeitungsebene (Anwendung)
    - Datenebene (z.B. DB)
- Multitier-Architekturen
  - *Definition (tier)*: Stufe, Ebene
  - 2-Tier-Architekturen:
    - Fat Clients (Darstellung+Verarbeitung←→Daten/Datenbank)
    - Thin Clients (Darstellung←→Verarbeitung+Daten/Datenbank)
  - 3-Tier-Architekturen:
    - Darstellungsrechner
    - Anwendungsserver
    - Datenbankserver

17. Dezentralisierte Systemarchitekturen (Peer-to-Peer Architekturen)

- Vertikale Verteilung
  - Logisch unterschiedliche Komponenten auf unterschiedlichen Rechnern
- Horizontale Verteilung
  - Aufteilung von Server und Clients
  - Client und Server physisch in funktional gleiche Teile zerlegt, d.h. Client und Server laufen auf einer Hardware, aber arbeiten nur mit einer Teilmenge an Daten oder Dienste des gesamten Systems
- Peer-to-Peer unterstützt horizontale Verteilung
  - Anzahl der möglichen Kommunikationspaare bei n beteiligten Rechner ist n(n-1)/2
  - Atomisches P2P (Peer-to-Peer)
    - Keine zentrale Verwaltung des Netzes
    - Bei Beitritt gibt es 2 Möglichkeiten
      - Broadcastanfrage an alle Clients, um zu erfahren, welche Dienst vorhanden sind
        - Beispiel: Datei- und Druckerfreigabe von Microsoft
      - Verbindung zu bekanntem Host (aus Liste) und Erfragung der Topologie und der Dienste
  - Benutzerzentriertes P2P
    - Einem Atomischen P2P wird ein für die Vermittlung von Benutzer, egal vom welchem Rechner, zuständiger Server hinzugefügt
    - Der Vermittlungsserver enthält eine Liste von Benutzern und deren Dienste (Freigabe einer MP3 Datei)
    - Betritt ein Benutzer dieses P2P meldet er sich mit seinen Diensten am Vermittlungsserver an
  - Datenzentriertes P2P
    - Vermittlungsserver merkt sich, wo Daten und Dienste liegen; sonst ähnlich wie bei benutzerzentrierten P2P
- Overlay-Netzwerk überlagert LAN/MAN/WAN und bildet somit ein eigenes Netzwerk
  - Peers sind Knoten des Netzwerks, welche durch Prozesse gebildet werden
  - Verbindungen sind die möglichen Kommunikationskanäle zwischen den Peers
  - Strukturierte
  - Unstrukturiere
- Strukturierte Peer-to-Peer Architekturen (Overlay-Netzwerke):
  - Konstruktion des Netzwerks durch deterministisches Verfahren
  - Verteilte Hash-Tabelle (Distributed Hash-Table, DHT), jeder Knoten

erhält zufällige Zahl aus dem Bezeichnerraum
  - Chord-System, Anordnung der Knoten als logischer Ring
  - CAN (Content Adressable Network), d-dimensionales kartesisches Koordinatensystem wird vollständig zwischen allen Koten auftgeteilt
- Unstrukturierte Peer-to-Peer Architekturen (Overlay-Netzwerke):
  - Knotenbildung durch Zufallsalgorithmen
- Topologieverwaltung in Overlay-Nertzwerken
  - Realisierung der Anordnung der Knoten im Raum /Topologie durch eine Kombination aus strukturierten und unstrukturierten Peer-to-Peer Systemen in 2 Schichten
- Superpeer-Konzepte
  - Superpeer enthalten Index (für Daten oder Dienste) oder fungieren als Makler
  - Superpeers sind auch als Peer-to-Peer System angeordnet
  - Regulären Peer hat meistens eine festen Superpeer an welchen er sich bindet



18. Hybridarchitekturen

- Kombination von Client-Server und dezentralisierter Architektur
- Beispiele:
  - Superpeer
  - Edge-Server
    - Edge (engl. Kante) zwischen Internet und Unternehmensnetzwerk oder ISP eines Endanwenders
    - Verbindung von Client zum Edge als Server, Edge als Peer-to-Peer zu Internetdiensten verbunden
  - Kollaborative Systeme
    - Knoten tritt dem System an definierter Stelle bei, danach benutzt er ein dezentralisiertes Verfahren zur Zusammenarbeit mit anderen Knoten
    - BitTorrent-System
      - Globales Verzeichnis von .torrent Dateien (strukturiert)
      - Laden der Daten von aktiven anderen Knoten (unstrukturiert)

19. Fehlertoleranz

- Anforderungen an Verlässliche Systeme (solch ein System ist fehlertolerant):
  - Verfügbarkeit:
    - Wahrscheinlichkeit, dass das System zu einer bestimmt Zeitpunkt korrekt arbeitet und seine Dienste bereitstellt
  - Zuverlässigkeit:
    - Mit hoher Wahrscheinlichkeit lange Zeit ohne Unterbrechung arbeiten kann
  - Funktionssicherheit:
    - Bezeichnet den Umstand, dass es zu keiner Katastrophe kommt, wenn das System zeitweise unkorrekt arbeitet
  - Wartbarkeit:
    - Wie leicht kann ein ausgefallenes System repariert werden
- Ein *Fehler* ist ein Teil eines Systemzustandes welcher zum Ausfall führen kann. Ein Ausfall beinhaltet das teilweise oder komplette Versagen von Diensten, welche es anbietet.
- Die Ursache eines Ausfalls wird als *Störung* bezeichnet.
  - Vorübergehende Störung
  - Wiederkehrende Störung
  - Permanente Störung
- Kontrolle von Störungen wird unterschieden in
  - Verhinderung von Störungen
  - Behebung von Störungen
  - Vorhersage von Störungen

- Ziel ist es Systeme zu konstruieren, welche selbst bei Störungen noch Ihre Dienste anbieten können. In diesem Fall ist das System Fehlertolerant.

20. Fehlermodelle

- Kommt es zu einer Störung, kann es folgende Ursachen haben:
  - Server hat Störung
  - Kommunikationskanal hat Störung
  - Beides hat eine Störung
- Ausfallarten:
  - Absturzausfall:
    - Server steht; ständiger Ausfall vom allen Diensten
  - Dienstausfall:
    - Server antwortet nicht auf Anfragen
    - Server erhält keine Anfragen von Clients
    - Server sendet keine Nachrichten
  - Zeitbedingter Ausfall:
    - Antwortzeit liegt außerhalb des definierten Wertes
  - *Ausfall korrekter Antworten*:
    - Falscher Wert wird geantwortet
    - Server weicht vom Programmablauf ab
  - Byzantinischer/Zufälliger Ausfall:
    - Server liefert zufällige Antworten zu unbestimmten Zeitpunkten
    - Server liefert absichtlich falsche Antworten (Byzantinischer Fehler)
    - Server liefert gar keine Antworten mehr(Ausfall-Stopp)
    - Server liefert sinnlose für den Client als Unsinn zu erkennende Antworten(Fail Safe/ausfallsicher, da vom Client erkannt)

21. Maskierung von Störungen durch Redundanz

- Schlüsseltechnik Redundanz
  - Informationsredundanz
    - z.B. CRC (Cyclic Redundancy Check), hinzufügen von Bits zum Datenstrom, um im Fehlerfall einige zu korrigieren
  - Zeitliche Redundanz
    - Einfach später nochmals versuchen ein Aktion auszuführen
  - Technische Redundanz:
    - Zusätzliche Prozesse oder Ausrüstung
    - *Gebräuchlichste Technik zu Realisierung von Fehlertoleranz*

KE 2 Kommunikation Verteilter Systeme

22. Netzwerkschichten

- Aufgrund des Fehlens von Shared Memory (mehrere Prozesse teilen sich einen Speicher) erfolgt die gesamte Kommunikation in Verteilten Systemen über **das Senden und Empfangen von Nachrichten.**
- Die Ansammlung in einem System verwendeten Protokolle wird als Protokollstapel bezeichnet.
- OSI (Open System Interconnection)-Modell:
  - Bestehend aus sieben Schichten:
    - Application Layer (Anwendungsschicht)
      - Aufgabe:
        - Unterstützung von Netzwerkanwendungen
        - Schnittstelle zu Anwendungen
      - Beispiele:
        - SIP Internettelefonie
        - HTTP Webseiten
        - FTP Datenübertragung
        - Telnet
        - SMTP E-Mail
    - Presentation Layer (Darstellungsschicht)
      - Aufgabe:
        - Interpretation der Semantik der übertragenen Bits in strukturierten Informationen wie z.B. Name, Adresse oder Geldbetrag sowie deren Erstellung.
        - Formatumwandlung unterschiedlicher Datenrepräsentationen (ASCII→EBCDIC→Unicode oder big-/little- Endianformat)
    - Session Layer (Sitzungsschicht, Kommunikationssteuerungsschicht)
      - Aufgabe:
        - Verbesserte Version der Transportschicht. Stellt Dialogkontrolle zur Verfügung, überwacht welcher Teilnehmer gerade kommuniziert und bietet Synchronisationsmöglichkeiten an. Wird nur selten unterstützt!
    - Transportation Layer (Transportschicht)
      - Kommunikation zwischen *Prozess auf Quellknoten←→ Prozess auf Zielknoten*
      - Aufgabe:
        - Stellt Daten anderen Prozessen zu
        - Ggf. Fehlerkorrektur
        - Ggf. Einhaltung der Reihenfolge von Paketen
      - Beispiele:

        - TCP (Transmission Control Protocol, zuverlässig, verbindungsorientiert)
        - UDP (User Datagram Protocol, unzuverlässig, verbindungslos)
- Network Layer (Vermittlungsschicht)
    - Kommunikation zwischen *Quellknoten←→ Zielknoten*
    - Aufgabe:
        - Routing in WANs mit dem Ziel, die kürzesten Segmentlaufzeiten zu erreichen. Segment ist die Dateneinheit dieser Schicht.
    - Beispiele:
        - X.25: verbindungsorientiertes Protokoll, beste Route wird während des Verbindungsaufbaus bestimmt.
        - IP: verbindungsloses Protokoll. Jedes IP-Paket wird unabhängig von allen anderen verschickt und geroutet.
- Data Link Layer (Sicherungsschicht)
    - Kommunikation zwischen *aktuellen Knoten←→ benachbarten nächsten Knoten* auf dem Weg zu Zielknoten
    - Aufgabe:
        - Senden von Bits in Frames (Rahmen)
        - Fehlerkorrektur, z.B. CRC
        - Flusskontrolle
            - Sliding Window Protokoll
                - Ermöglich Bestätigungen von Frames auf der Sicherungsschicht
                - Selective Repeat: einzelne Sendungswiederholung
                - Go Back n: Übertragung aller Frames ab dem verlorenem
    - Beispiele:
        - Frame Relay
        - Ethernet
        - ATM (Asynchrone Transfer Mode)
- Physical Layer (Bitübertragungsschicht)
    - Kommunikation zwischen *aktuellen Knoten←→ nächsten Knoten* auf dem Weg zu Zielknoten
    - Aufgabe:
        - Übertragung von beliebigen Bitfolgen
        - Bestimmung der physikalischen Darstellung für 0- und 1-Bits
    - Beispiele:

- RS 232
- V.24

23. Middleware-Protokolle

- Eine Anwendung, welche in der Anwendungsschicht logisch angesiedelt ist und viele Protokolle für allgemeine Zwecke bereitstellt nennt man *Middleware*.
- Unterscheidung zwischen:
  - Kommunikationsprotokolle auf hoher Ebene
  - Protokolle zur Bereitstellung von Middleware-Diensten
- *Middlewareschicht liegt zwischen Anwendungs- und Transportschicht*
- Middleware-Kommunikationsdienste:
  - RPC (Remote Procedure Call)
  - Warteschlangensysteme
  - Streams
  - Multicasting

24. Arten der Kommunikation

- Persistente (dauerhafte) Kommunikation:
  - Nachricht wird solange von der Kommunikations-Middleware gespeichert, bis sie zugestellt werden kann
- Transiente (flüchtige) Kommunikation:
  - Nachricht wird verworfen, wenn sie nicht ausgeliefert werden kann, z.B. bei Verbindungsstörung
- Asynchrone Kommunikation:
  - Sender fährt sofort fort, sobald er eine Nachricht abgesetzt hat
- Synchrone Kommunikation:
  - Sender wartet bis seine Anfrage akzeptiert wurde, erst dann fährt er fort und sendet z.B. eine weitere Anfrage.
  - Diese Synchronisation kann an 3 Stellen erfolgen:
    - Sender bleibt gesperrt bis die Middleware seine Anfrage übernimmt
    - Sender wartet bis die Anfrage an den Empfänger zugestellt wurde
    - Sender wartet bis seine Anfrage bei Empfänger vollständig verarbeitet wurde und er eine Nachricht vom Empfänger über den Erfolg oder Misserfolg erhält

25. RPC Remote Procedure Call

- Will ein Prozess auf Rechner A eine Prozedur auf Rechner B ausführen und evtl. ein Ergebnis erhalten, so nennt man dieses Verfahren RPC (dt. entfernter Prozeduraufruf).

- Ziel ist es, den RPC wie einen lokalen Prozeduraufruf ausschauen zu lassen; er soll transparent sein.
- Probleme:
  - Übertragung der Parameter
  - Übertragung des Ergebnisses
  - Unterschiedliche Datendarstellungen z.B. ASCII←→ Unicode; Little Endian←→ Big Endian, Boolsche Werte, Fließkommadarstellung
  - Call by Reference muß simuliert werden (Arrays oder Speicherbereiche müssen mit übertragen werden).
  - Unterschiedliche Adressräume
- Stub:
  - Ermöglicht den Aufruf von Prozeduren auf anderen Rechnern. Für den Entwickler sieht ein solcher Aufruf wie ein gewöhnlicher lokaler Aufruf aus. (Zugriffstransparenz)
- Ablauf von RPC:
  - Client ruft Client-Stub auf
  - Client-Stub stellt Nachricht zusammen und ruft das lokale OS auf
  - Das lokale OS sendet Nachricht an das entfernte OS
  - Entferntes OS übergibt Nachricht an Server-Stub
  - Server-Stub entpackt Paramter und ruft den Server auf
  - Server verarbeitet den Aufruf und übergibt Ergebnis an Server-Stub
  - Server-Stub verpackt es in eine Nachricht und ruft das lokale OS auf
  - Lokales OS sendet Nachricht an Client OS
  - Client OS übergibt Nachricht an Client-Stub
  - Client-Stub entpackt Nachricht und gibt das Ergebnis an den Client weiter
- Synchrone RPC
- Asynchrone RPC
  - Kein Rückgabewert vorhanden, z.B. Geld überweisen Eintrage in DB hinzufügen (persistente Kommunikation)
- IDL (Interface Definition Language)
  - Definiert die Schnittstelle zwischen dem Client-Stub und den Anwendungen
  - Definiert Sammlung von Prozeduren, welche vom Client aufgerufen werden können und vom Server implementiert werden
- Beispiel DCE-RPC

26.

Parameterübergabe bei RPC

- Marshalling:
  - Das Verpacken von Parametern/Datenstrukturen in eine Nachricht in einem Zwischenformat (z.B. XML inclusive DTD Document Type Definition oder XML Schema)
  - Gründe für Marshalling:
    - Unterschiedliche Datendarstellungen
- Unmarshalling:
  - Das Entpacken einer Nachricht aus einem Zwischenformat

27. Nachrichtenbasierte Kommunikation

- Gründe für Nachrichten:
  - RPC innewohnende synchrone Natur nicht immer wünschenswert
  - Server wird nicht ausgeführt, während Anforderung vom Client ankommt
- Transiente nachrichtenorientierte Kommunikation
  - Berkeley Sockets:
    - Socket: Kommunikationsendpunkt
    - Server implementiert:
      - Socket: neuen Kommunikationsendpunkt erstellen
      - Bind: lokale Adresse einem Socket zuordnen
      - Listen: Bereitschaft zum Empfangen von Verbindungen
      - Accept: Aufrufer blockieren bis Verbindungsanfrage eingeht, dann ist eine Verbindung vorhanden
    - Client implementiert:
      - Connect: aktiver Versuch eine Verbindung aufzubauen
      - Send: Daten über Verbindung senden
      - Receive: Daten über Verbindung empfangen
      - Close: Verbindung freigeben
  - MPI (Message-Passing Interface)
    - 2 Gründe für die Unzulänglichkeit von Sockets:
      - Sie unterstützen nur Senden und Empfangen
      - Sockets wurden entworfen um mit Allzweckprotokollstapeln wie TCP/IP über das Netzwerk zu kommunizieren. Sie eignen sich z.B. nicht für Hochgeschwindigkeitsnetze in Serverclustern, da diese erweiterte Funktionen wie Synchronisierung oder Pufferung bedarf.
    - MPI nutzt das zugrundeliegende Netzwerk unmittelbar
    - Unterstützt synchrone und asynchrone Kommunikation
      - Beipiel:
        - MSI_bsend asynchron
        - MSI_send synchron

        - MPI wurde für Hochleistungsparallelanwendungen entworfen
- Persistente nachrichtenorientierte Kommunikation
    - MQS Message-Queuing Systems (Warteschlangensysteme) ODER
    - MOM Message-Oriented Middleware (nachrichtenorientierte Middleware)
        - Gedacht für Kommunikation welche Minuten anstatt Sekunden/ Millisekunden benötigen darf
    - Anwendungen kommunizieren, indem Sie Nachrichten in Warteschlangen einfügen
    - Nachrichten bleiben erhalten, egal ob Sender, Empfänger oder beide aktiv oder passiv sind (falls gewünscht)
    - Adressierung über systemweiten eindeutigen Namen der Zielwarteschlange
    - Primitive:
        - Put: Nachricht an Warteschlange anhängen
        - Get: blockieren bis Warteschlange nicht mehr leer ist, dann erste (am längsten wartende) Nachricht holen
        - Poll: die erste (am längsten wartende) Nachricht holen, blockiert nicht
        - Notify: Handler installieren, welcher bei Einstellung einer Nachricht in die Warteschlange, aufgerufen wird
    - Aufbau/Aufgaben eines Warteschlangensystems:
        - Quellwarteschlange
        - Zielwarteschlange
        - Bereitstellung von Warteschlangen für Sender und Empfänger
        - Warteschlangen werden von Warteschlangenmanagern verwaltet
            - Normalerweise kommuniziert der Message-Queue Manager mit der Anwendung
            - Warteschlangenmanager können aber auch als Relay oder Router arbeiten, um Nachrichten an andere Warteschlangen zuzustellen, Nachrichten zu protokollieren (Ziele: Sicherheit, Fehlertoleranz etc.) oder skalierbare Warteschlangensysteme aufzubauen
    - Nachrichtenbroker (Nachrichtenvermittler)
        - Aufgabe ist die Umwandlung von Nachrichtenformaten, falls Sender und Empfänger verschiedene Formate benutzen
        - Ist nicht zwingend fester Bestandteil eines MQS
    - Wichtigster Anwendungsbereich ist Integration einer Sammlung von Datenbanken und Anwendungen in ein verbundenes Informationssystem
    - Beispiel: WebSphere von IBM, EntireX, JMS

28. Objektbasierte Nachrichtenübermittlung

- Beispiele:
  - CORBA - Common Object Request Broker Architecture
  - SOAP - Simple Object Access Protocol
  - Java RMI - Remote Method Invocation
  - Globe
    - OID Object Identifier werden als Objektreferenzen benutzt
    - Wo das Objekt liegt, muss an einem Locationserver erfragt werden. Dieser gibt eine Kontaktadresse zurück:
      - Adressbezeichnung des Knotens + Adressbezeichnung des Objekts
- Das Verpacken von Parametern/Datenstrukturen in eine Nachricht in einem Zwischenformat (z.B. XML inclusive DTD Document Type Definiton oder XML Schema)
- Synchronisierung
  - Verhinderung, dass mehrere Prozesse gleichzeitig auf ein Objekt oder eine Methode zugreifen.
  - Kann zu Problemen führen, da der aufrufende Prozess keine Kenntnis über die entfernten Objekte hat. Diese Objekte können auch Ihrerseits wiederum Verweise auf andere entfernte Objekte haben. Hierdurch kann es zur einer Sperrenkaskade (MEGA Deadlock) kommen.

29. RMI Remote Method Invocation, CORBA und SOAP

- Im Wesentlichen das gleiche wie RPC, nur dass hierbei Methoden an Objekten aufgerufen werden. Z.B. Java RMI
- CORBA Common Object Request Broker Architecture
  - Komponenten:
    - ORB Object Request Broker
      - Sorgt für die Vermittlung von Anfragen an das richtige verteilte Objekt.
    - IDL Interface Definition Language
      - Damit Objekte unterschiedlicher Programmiersprachen miteinander kommunizieren können, müssen Objekte mit IDL definiert werden, z.B. Kommunikation zwischen Delphi- und Java-Objekte
    - IIOP Internet Inter-Orb Protokoll
      - Regelt die Kommunikation über Nachrichten zwischen den ORB’s auf den jeweiligen Rechnern.
    - Interface Repository
      - Jeder ORB verfügt über sein eigenes Interface Repository. Hier werden für alle Objekte, welche der ORB verwaltet, IDLs abgelegt.
    - IOR Interoperable Object Reference

            - Sprachunabhängige Objektreferenz; Objektreferenzen werden in diesem Format zwischen ORB's ausgetauscht
    - Ablauf eines CORBA Methodenaufrufs :
        - Rechner A erhält von ORB Referenz auf ein entferntes Objekt
        - Rechner A sendet nun eine Nachricht an Ihre Referenz
        - ORB von Rechner A verpackt die Nachricht(Marshalling)
        - ORB von Rechner A verschickt die Nachricht an Rechner B's ORB
        - ORB von Rechner B entpackt die Nachricht(Unmarshalling)
        - Rechner B verarbeitet die Nachricht und schickt Ergebnis an lokalen ORB
        - ORB von Rechner B verpackt das Ergebnis
        - ORB von Rechner B sendet Ergebnis an ORB von Rechner A
        - ORB von Rechner A entpackt das Ergebnis
        - ORB übergibt Ergebnis an Rechner A
    - Asynchrone Methodenaufrufe von CORBA:
        - Call-Back
            - Eine Schnittstelle/Methode wird durch 2 ersetzt (am Client)
                - z.B. int add(in int i, in int j);
            - Die 1. enthält nur die Übergabeparameter und Aufrufnamen, sie wird von Client aufgerufen
                - z.B. void sendCB_add(in int i, in int j);
            - Die 2. wird vom Server, nach erfolgreicher Arbeit, aufgerufen
                - z.B. void replyCB_add(in int returnvalue);
        - Abfragemodell
            - Client fragt nach, ob Ergebnis schon da ist
                - z.B. int add(in int i, in int j); wird zu:
                    - 1) void sendPOLL_add(in int i, in int j);
                    - 2) void replyPOLL_add(out int returnvalue);
    - Stärken
        - Sprachunabhängigkeit
        - Unterstützung zum Auffinden von entfernten Objekten
        - Kontrolle von konkurrierenden Zugriffen auf Objekte
        - Einfache Transaktionsmechanismen
- SOAP Simple Object Acces Protokoll
    - Lösung von 3 allgemeinen Problemen:
        - Abbildung der Dateirepräsentation auf ein plattformunabhängiges Zwischenformat(Marshalling).
        - Unterstützung der Adressierung der Kommunikationspartner (Port und Host) und eventuelles Verbergen vor Programmierer
        - Zugangspunkte für die Kommunikation auch bei Systemen mit Firewall

- Realisierung mithilfe von HTTP
  - Zusätzlicher Headereintrag SOAPAction
  - Nutzlasten als SOAP-XML im SOAP-Envelope (Umschlag)
- HTTP Port 80 zur Überbrückung von Firewalls
- XML als Plattformunabhängiges weit verbreitetes Format

30. Konsistenz und Replikation in objektbasierten Verteilten Systemen

- Konsistenz der Replikate:
  - Änderungen nur an Masterkopie auf Server:
    - Vorteil von Replikation bei Schreibvorgängen aufgehoben
  - Änderungen nur an Masterkopie auf einem Client:
    - Zu jedem Zeitpunkt ist genau ein Client im Besitz einer solchen Masterkopie.
    - 2 Möglichkeiten der Veränderung an Objekten
      - Anfrage eines Clients an den Client mit Masterkopie
      - Erhalt der Masterkopie, z.B. durch Mastertoken
  - Änderung an jedem Objekt erlaubt
    - Probleme:
      - Keine mehrfachen Aufrufe desselben Objekts zur gleichen Zeit darf erfolgen
      - Keine unabhängigen Methodenaufruf an 2 Replikaten eines Objekt zu gleichen Zeit dürfen erfolgen
    - Lösung:
      - Vollständig geordnetes Multicasting (zeitlicher Ablauf ist komplett geordnet bei allen am verteilten System beteiligten Prozessen/Threads)
      - Alle Threads, die sich eine Sperre teilen müssen bei allen Replikaten in der selben Reihenfolge abgearbeitet werden
  - Beispiel für replizierte Datenhaltung:
    - Web-Browser mit Pull-Modell Aktualisierung (Browser holt sich Änderungen)
    - Replikationsframework mit Interzeptoren (Unterbrecher bei gewissen Ereignissen, welche für die Konsistenz der Replikate erforderlich sind) für Java-Beans (EJB) auf J2EE Replikation
- Replizierte Aufrufe:
  - Es muss unterbunden werden, dass eine Methode, welche auf mehreren Replikaten durchgeführt wird und diese ein anderes einzelnes Objekt manipulieren, mehrfach manipulieren.
  - Beispiel:
    - Banküberweisung wird an einem replizierten KontoSteuerungsObjekt durchgeführt. Dieses hebt nun vom tatsächlichen Quellkontoobjekt ein Geldbetrag X ab. Dieser Vorgang darf nur 1mal erfolgen, das Abheben der anderen

Replikate muss verhindert oder erkannt werden.
    - Lösung sind z.B. Koordinatoren an den Objekten
- Replizierende Systeme:
    - Unterscheidung durch 2 Punkte von lokalen:
        - Aktualität der Replikate
        - Konsistenz von Änderungen, nachdem 2 Clients Replikate geändert haben, müssen die Replikate den gleichen Zustand aufweisen
    - Arten der Verteilung von Veränderungen:
        - Verteilung von Benachrichtigung
            - Alle nicht geänderten Replikate werden als invalid markiert
            - Komplette Übertragung des Objektzustandes bei Bedarf (Lese- oder Schreibzugriff)
            - Mittlere Bandbreitenauslastung
            - Gut bei großer Anzahl von Änderungen und kleiner an Lesevorgängen
        - Verteilung von veränderten Daten
            - Alle Änderungen der Daten werden sofort an alle Systeme, auf welchem Replikate liegen verschickt, kompletter Objektzustand
            - Senden der Veränderungen, egal ob Client diese noch braucht oder nicht
            - Hohe Bandbreitenauslastung
            - Gut bei großer Anzahl von Lesevorgängen und kleinerer Anzahl an Schreibvorgängen
        - Verteilung von Updateinformationen
            - Mitteilung wie und was sich an einem Objekt geändert hat
            - Client kann hierdurch selber entscheiden, ob er jetzt oder später ein Update durchführen will oder muss, je nach Schreib oder Lesezugriff oder nicht von beiden
            - Nur Updateinformationen werden versendet
            - Geringe Bandbreitenauslastung
            - Nachteil ist ein höherer Rechenbedarf an den Clients
                - Veränderungsnachrichten können
                - Per Push-Modell gesendet werden oder
                - Per Pull-Modell geholt werden

31. Fehlertoleranz und Sicherheit in objektbasierten Verteilten Systemen

- Fehlertoleranz wird u.A. durch technische Replikation erreicht
  - Java verwendet aktive Replikation
    - Aktive Replikation:
      - Keinem Replikat werden Datenveränderungen übertragen, sondern nur mitgeteilt welche Änderungsoperation (plus nötige Parameter dafür) es ausführen muß, um in den aktuellen Zustand zu kommen.
      - Voraussetzung ist hierfür, dass jedes Replikat durch einen Prozess repräsentiert wird, welcher die Änderungssoperationen verarbeiten kann und sich aktiv auf dem aktuellen Stand hält.
        Prozesse müssen zwingend als deterministische endliche Automaten arbeiten!
    - Kann wegen Nativem Code, Threads und verschiedenen Fehlerausgaben der JVM (Java Virtual Machine) nicht deterministisch sein
- Sicherheit wird wesentlich durch Standardauthentifizierung und Autorisierung erreicht

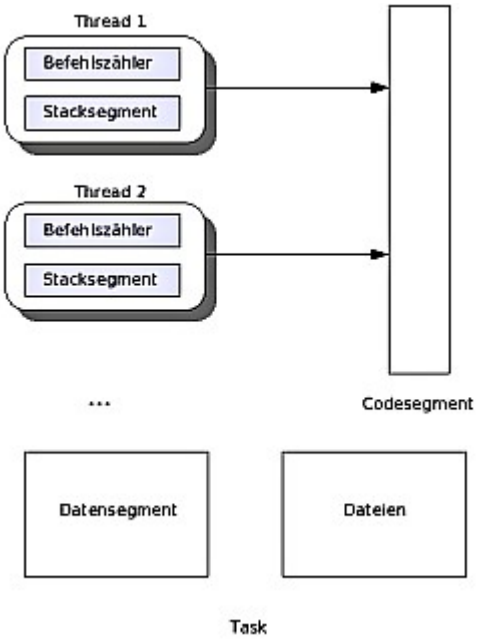
KE 3 Prozesse und Zeit

32. Threads

- Gründe für Threads in Verteilten Systemen:
  - Mehrere Steuerthreads erleichtern den Aufbau
  - Führen zu höherer Leistung
- Prozess:
  - Programm in Ausführung
  - Programmcode + Daten + Prozesskontext
    - Prozesskontext besteht aus:
      - Registerinhalten bestehend u.A. aus
      - Befehlszähler
      - Grenzen der Adressraums
      - Prozessnummer
      - Anderen Informationen wie geöffnete Dateien, Elternprozess etc.
  - Prozesswechsel:
    - Unabhängigen Adressraum anlegen
    - Datensegmente mit 0 initialisieren
    - Prozesskontext sichern
    - Memory Mapped Unit(Überstetzung virtuelle auf physikalische

Adressen) und Translation Lookaside Buffer(Puffer für die übersetzten Adressen) ändern
- Bei E/A Operation wird gesamter Prozess blockiert

- Threads:
  - Threads teilen sich Programmcode, Adressraum und Dateien wie ihr Erzeuger
  - Programmcode + Daten + Threadkontext
    - Threadkontext besteht aus:
      - Befehlszähler
      - Eigenem Stack



  - Gruppe von Threads nennt man Task, diese sind einem Prozess zugeordnet

- Implementierung von Threads:
  - User Threads:
    - Vorteile:
      - Einfaches erstellen und zerstören
      - Einfacher Kontextwechsel
      - Leistungsgewinn gegenüber Prozessen
    - Nachteile:
      - Legt den gesamten Prozess, in welchem er läuft, und somit den Task bei E/A lahm
      - Fehlender Speicherschutz

- Kernel Threads:
  - Vorteil:
    - Keine Blockierung des Task, da er vom Kernel wie ein Prozess behandelt wird
  - Nachteil:
    - Leistungsverlust, da Systemcall welcher die Leistungsvorteile komplett zunichte machen kann
- Lösung:
  - Leichgewichtige Prozesse LWP
    - Läuft auch im Kontext eines Prozesses
    - Kontextwechsel wird bei E/O durch den Userspace(Threadbibiliothek) geregelt
    - Synchronisierung (Schutz) wird auch durch den Userspace implementiert

33. Threads in Verteilten Systemen

- Lassen Systemaufrufe zu, ohne den gesamten Prozess lahmzulegen
- Multithread-Client am Beispiel eines Browsers:
  - Kann parallel Verbindungen zu verschiedenen Server aufnehmen, während man schon ein Dokument lesen kann. Bilder und andere Elemente werden während der Betrachtung nachgeladen.
  - Allgemein können verschiedene Threads parallel unterschiedliche Aufgaben übernehmen, ohne sich gegenseitig zu blockieren
- Multithreading wird in Verteilten Systemen hauptsächlich serverseitig genutzt
- Multithread-Server:
  - Vereinfacht Servercode durch Schichtung und Weiterleitung von Aufgaben an andere Threads
    - Beispiel: Ein Thread in einem Dateiserver nimmt Anfragen entgegen und startet für jede einen eigenen Antwortthread.
  - Vereinfacht Serverentwicklung, die die Parallelverarbeitung zur Leistungsoptimierung ausnutzen

33. Virtualisierung

- Definition:
  - Bestehende Schnittstelle wird so erweitert oder ersetzt, dass sie das Verhalten eines anderen Systems nachahmen kann
- Durch rasche Umschaltung von Threads und Prozess wird die Illusion erzeugt, man könne parallel Verarbeiten
- Diese Idee Übertragen auf Ressourcen führt zur Ressourcenvirtualisierung.
- Hardware ändert sich schneller als die Software (Anwendungen und Middleware)

- Anwendungen laufen auf Ihren eigenen Virtuellen Maschinen und ermöglichen somit einfach Portabilität, Flexibilität und Replikation
- Architektur Virtueller Maschinen

34. Prozessmigration

2 Typen der Prozessmigration (Preemptiv und Non-Preemptiv).
Non-Preemptiv:

1. Neustart des gleichen Prozesses auf einer anderen Rechner.

<u>Preemptiv (Migration während der Prozess läuft)</u>

Folgende Schritte sind dafür notwendig:

- Einfrieren des Prozesses auf dem Quellrechner
- Erzeugung eines neuen eingeforenen Prozesses auf dem Zielrechner
- Übertragung des Prozesszustands (Register, Scheduling-Tabellen, Seitentabellen, EA-Zustände, Zugriffsrechte, Prozess-ID)
- Übertragung des Arbeitsspeichers des Prozesses
- Aktivierung des Prozesses auf dem Zielrechners

Wir unterscheiden zwischen zwei Arten der Arbeitsspeichermigration:

1. Übertragung nach dem Einfrieren des Prozesses
   - Übertragung des gesamten Arbeitsspeichern en block
   - Prozess steht während der gesamten Migration nicht zur Verfügung
2. Übertragung vor dem Einfrieren des Prozesses
   - Übertragung wird vor dem Einfrieren angestossen
   - Geänderte Speicherseiten werden neu übertragen
   - Sobald nur noch sehr wenige Speicherseiten übertragen werden müssen, wird der Prozess eingefroren und die Migration abgeschlossen (Übertragung der noch fehlenden Speicherseiten, Prozesskontextübertragung, Prozessstart etc.)

Vor- bzw. Nachteile der zwei Speicherübertragungsverfahren:

“Übertragung nach”:	einfacheres Verfahren **Ausfallzeit des Prozesses ist größer**
“Übertragung vor”:	komplexer **Prozessausfallzeit sehr klein** Prozessmigration dauert insgesamt länger

35. Zustandslose/behaftet iterativ/nebenläufige Server

36. Uhrzeitsynchronisation

- Warum braucht man Zeitsychronisation?
  - Zeitstempel für eCommerce
  - Konsistenz verteilter Daten
  - Prozesssynchronisation
  - Authentizitätsüberprüfung (Kerberos)
  - make-Prozess
- Möglichkeiten:
  - Synchronisation mit autorativen ext. Uhren (UTC-Zeit; DCF77; Atomuhren)
  - interne Synchronisation mit bekanntem Maß an Genauigkeit (muss nicht mit realer Zeit übereinstimmen)

Achtung: Uhrzeiten dürfen nicht zurückgestellt werden. Bei Abweichungen müssen sie entweder verlangsamt oder beschleunigt werden.
Algorithmen:

- Logischen Uhren:
- Lamport-Algorithmus (kann keine Kausalität abbilden): ein Ereignis b, das nach a auftritt, bedeutet nicht zwingend, dass b aufgrund von a aufgetreten ist
Basis: happens-before Relation a --> b, d.h. Ereignis a war vor b.
  - Nachrichtenversand erfolgt mit Angabe der lokalen Zeit des Senderprozesses

Falls Empfänger t > Sender t ist alles o.k.

  - Falls Empfänger t < Sender t => Empfänger t = Sender t + 1

Falls die Zeitangabe um Prozessnummer ergänzt wird ([t,pid]), wird ein vollständig geordnetes Multicasting erreicht. Bsp.: [1,1] --> [1,2]

Vektoruhr (von Mattern und Fidge):

Vorteile:

– komplette Kausalitätskette

Nachteile:

– sehr hoher Speicherverbrauch
– Nutzlast steigt proportional zur Anzahl der Prozesse

Funktion: Jeder Prozess besitzt einen eigenen Zähler und merkt sich für alle anderen beteiligten Prozesse deren ihm zuletzt bekannte Zeit/Zähler in einem Vektor. Bei einem Ereignis wird der eigene Zeitstempel erhöht und im Vektor vermerkt. Der Vektor enthält ebenfalls die ihm zuletzt bekannten Zeitstempel der anderen Prozesse. Diese Zeitstempel werden ihm mit dem Empfang einer Nachricht mitgeteilt (ebenfalls als Vektor).

Beispiel: Gegeben seien 3 Prozesse.
Prozess P1 sendet Prozess P2 eine Nachricht; hier wird nun ein neuer Zeitstempel berechnet.
V1 = (2,0,0)
V2 = (0,3,0)
Hieraus ergibt sich der neue Zeitstempel:
V2_neu = (2,3+1,0)

Es existiert noch ein nebenläufiger Prozess P3.

Physikalische Uhren: synchronisieren idealerweise mit einer Atomuhr.

Berkeley: Zeitserver fragt Clients nach Ihrer Zeit:
1. Er schickt seine Zeit an die Clients
2. Die Clients antworten mit der Differenz
3. Der Server ermittelt arithmetisches Mittel
4. Der Server schickt eine Änderungsinformation an alle Clients (+/- x Minuten)

NTP, basierend auf Algorithmus von Cristian:
– zentraler Zeitserver, der die genaue Zeit kennt (vielleicht durch DCF77)
1. Client fragt Server nach Zeit, zum Zeitpunkt t1
2. Nachricht trifft zu Zeitpunkt t2 beim Server ein
3. Nachricht wird zum Zeitpunkt t3 vom Server beantwortet (in Antwort steckt Zeit t2 drin)
4. Zum Zeitpunkt t4 kommt Antwort beim Client an
5. Client kann Abweichung der Zeit vom Zeitserver berechnen:
   $$\text{Delta} = \frac{(t2 - t1) + (t3 - t4)}{2}$$

Falls delta < 0, ist der Client zu schnell und muss seine Uhr verlangsamen

Bisheriger Ablauf ist der Algorithmus von Chrisitan, zusätzlich wird bei NTP eine Abschätzung der Verzögerung (Latenz) hinzu:

$$\text{klein delta} = \frac{(t2 - t1) + (t4 - t3)}{2}$$ (ist Gesamtlaufzeit)

NTP merkt sich 8 Tupel (Delta, klein delta). Das Tupel mit der geringsten Verzögerung, ist qualitativ die beste Zeitabweichung und wird deshalb verwendet.

NTP baut auf Stratum-Ebenen auf. Je geringer der Stratum-Wert, desto genauer ist

die Uhr. (Stratum 0 ist meistens eine Atom-Uhr, Stratum 1 der NTP-Server der nächsten Ebene, usw.). Die Stratumebenen bilden eine Hierarchie.
http://en.wikipedia.org/wiki/Clock_synchronization
http://en.wikipedia.org/wiki/Network_Time_Protocol

KE 4 – Sicherheit

Wie kann man Kommunikation sichern/schützen?
Datenübertragung verschlüsseln

Warum macht man das? Warum Verschlüsselung

- Schutz vor Angriffen, Spionage o. Sabotage…
- Sicherstellung von folgenden Eigenschaften bei der Übertragung:
  - Vertraulichkeit: Nur die Leute können Inhalt lesen, die es auch was angeht.
  - Integrität: Manipulationen können erkannt und korrigiert o. verhindert werden (z.B. Gebotshöhe bei Onlineauktionen)
  - Authentifizierung: Benutzer authentifiziert sich gegenüber System: z.B. es wird nicht das Passwort sondern nur ein Hashwert übertragen, der verglichen wird.
  - Digitale Signatur (Nicht Anfechtbarkeit): Rechtsgültige Unterschrift von elektronischen Dokumenten bzw. Kontrolle, ob Absender wirklich derjenige ist, der er vorgibt zu sein.
  - Anonymität: eCommerce (durch Mithören)
  - Verhinderung von Verzögerungen: wichtig bei Auktionen -> Angriffsmöglichkeit über Netzwerkknoten

Welche Methoden zur Verschlüsselung existieren

- Symmetrische Verfahren (secret key Verfahren):
  Mit demselben geheimen Schlüssel wird sowohl ver- als auch entschlüsselt
  - Vorteil: schnell
  - Nachteil: sicherer Schlüsselaustausch zwischen den Kommunikationspartnern
  - Beispiele: Verschiebechiffre, Substitutionschiffre, Vigenère-Chiffre, Permutationschiffre, DES (Data Encryption Standard; Feistel), 3DES, IDEA (International Data Encryption Algorithm; Feistel), Blowfish (Feistel), CAST-128 (Feistel), RC5 (Ron Rivest Algorithmen Familie), AES (Advanced Encryption Standard; 128, 192, 256 Bit Schlüssellänge; Substitution)
- Asymmetrische Verfahren (public key Verfahren):
  Es gibt einen öffentlichen Schlüssel mit dem verschlüsselt wird und einen geheimen privaten Schlüssel mit dem entschlüsselt oder signiert wird
  - Vorteil: leichter Schlüsselaustausch, da der öffentliche Schlüssel der ganzen Welt bekannt sein darf
  - Nachteil: langsam, da Entschlüsselung sehr aufwendig ist, deswegen für Sessionverschlüsselung nicht gut geeignet
  - Beispiele: El-Gamal (Diffie-Hellmann; Problem Berechnung von diskreten Logarithmen über endlichen Körpern), RSA (Rivest Shamir Adleman; Problem Primfaktorzerlegung großer Zahlen)

Welche klassischen Verschlüsselungsverfahren kennen Sie? Angriffsmöglichkeiten?

(1) **Verschiebechiffre** (monoalphabetische Verschlüsselung, d.h. ein und derselbe Buchstabe wird im Text immer mit demselben Zeichen codiert)
   a) Cäsar-Chiffre: Verschiebung um drei Stellen nach links: z.B. A -> D
      Formel: $e(x) = (x + 3) \bmod 26$; $d(y) = (y - 3) \bmod 26$ (bei 26 buchstabigem Alphabet)
   b) ROT13: Verschiebung um 13 Stellen nach links
      Formel: $e(x) = (x + 13) \bmod 26$; $d(y) = (y - 13) \bmod 26$

   Angriff: Häufigkeitsanalyse, wenn Sprache des Textes und somit seine Normverteilung bekannt ist (häufigster Buchstabe im deutschen z.B. „e“)

(2) **Substitutionschiffre** (monoalphabetische Verschlüsselung)
   Formel: Schlüsselumfang K ist alle Permutationen über das Alphabet.
   $\pi : \mathbb{Z} \rightarrow \mathbb{Z}$ mit $\pi(i) = j$ für $i,j \in \mathbb{Z}$.
   $e_\pi(i) = \pi(i) = j$ für jedes $i \in \mathbb{Z}$; $d_\pi(j) = \pi^{-1}(j) = i$ für jedes $j \in \mathbb{Z}$; $\pi^{-1}(j)$ ist die inverse Permutation von π.
   Angriff: Häufigkeitsanalyse, wenn Sprache des Textes bekannt ist (häufigster Buchstabe im deutschen z.B. „e“)

(3) **Vigenère-Chiffre** (polyalphabetische Verschlüsselung, d.h. ein und derselbe Buchstabe wird im Text unterschiedlich kodiert)

Verfahren Verschlüsseln:

1. Wähle ein Schlüsselwort „k“ der Länge l
2. Teile Klartext in Teiltexte der Länge l
3. Verschlüssle Teiltexte wie folgt:
   $e(x_0,\ldots,x_{l-1}) = (x_0+k_0,\ldots, x_{l-1} + k_{l-1})$
4. Setze verschlüsselte Teiltexte zu einem Text zusammen.

Verfahren Entschlüsseln:

1. Teile Chiffrat in Teiltexte der Länge l
2. Entschlüssle Teiltexte wie folgt:
   $d(y_0,\ldots,y_{l-1}) = (y_0-k_0,\ldots, y_{l-1} - k_{l-1})$
3. Setze entschlüsselte Teiltexte zu einem Text zusammen.

Angriff: Häufigkeitsanalyse mit Hilfe des Koinzidenzindex, wenn Sprache bekannt ist.

Anmerkung: Koinzidenzindex = Wahrscheinlichkeit, dass zwei zufällig aus einem Text gewählten Zeichen dieselben sind (z.B. bei deutschen Text ca. 0,074)

Verfahren:

- Berechne den Koinzidenzindex für verschiedene Schlüssellängen z.B. von 1 – 20 und wähle die Länge, die am nächsten am Koinzidenzindex der Sprache des verschlüsselten Textes liegt.
- Teile Chiffrat in Teiltexte der gewählten Länge
- Führe auf den Teiltexten eine Häufigkeitsanalyse durch, um das Schlüsselwort zu ermitteln (1. Teiltext liefert erstes Zeichen des Schlüsselworts, 2. das Zweite usw.) Wichtig ist hier zu wissen, dass die Teiltexte mit Verschiebechiffre verschlüsselt werden.

Verbesserung der Sicherheit: Schlüsselwort im Idealfall so lange wie den Klartext selbst wählen, weil dann zu wenig statistisches Material für die Bestimmung der Schlüssellänge zur Verfügung steht.

Anmerkung: Laut Vorlesung der Tübinger Uni knackt man den Vigenère einfacher, wenn man statt der reinen Koinzidenzindexmethode (Friedman-Test) das ganze mit dem Kasiski-Test kombiniert.
http://de.wikipedia.org/wiki/Kasiski-Test + http://timms.uni-tuebingen.de/List/List01.aspx?rpattern=UT_200[67]_____00[12]_krypto_000_ (8. Stunde)

Welche modernen Verfahren kennen Sie?

- Symmetrische Verfahren
  - DES (Data Encryption Standard; Feistel)
  - 3DES
  - IDEA (International Data Encryption Algorithm; Feistel)
  - Blowfish (Feistel)
  - CAST-128 (Feistel)
  - RC5 (Ron Rivest Algorithmen Familie)
  - AES (Advanced Encryption Standard; 128, 192, 256 Bit Schlüssellänge; Substitution)
- Asymmetrische Verfahren
  - El-Gamal (Diffie-Hellmann; Problem Berechnung von diskreten Logarithmen über endlichen Körpern),
  - RSA (Rivest Shamir Adleman; Problem Primfaktorzerlegung großer Zahlen)
  - DSA (Digital Signature Algorthm; orientiert an El-Gamal)

Wie funktioniert das DES Verfahren?

DES (Data Encryption Standard) ist ein Stromchiffre, d.h. eine Kombination aus Permutations- und Substitutionschiffre. Er basiert auf dem Feistel-Alorithmus.

Funktion: Ver- u. Entschlüsselung mit 56 Bit-Schlüssel aus einer S-BOX (Substitutionsbox mit geheimem Schlüssel)

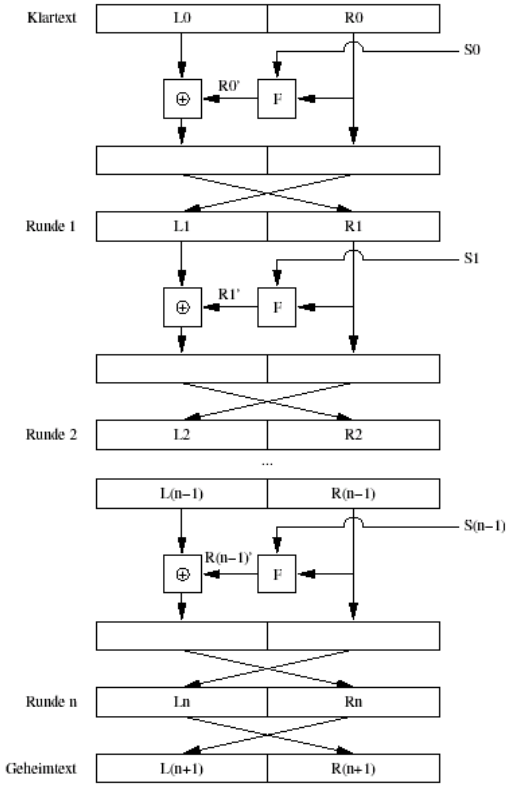
Verschlüsselung:

- Teile Klartext in 64 Bit Blöcke (weiteres Verfahren für je einen Block erklärt)
- Teile den 64 Bit Block in zwei Hälften L_0 und R_0

- Verschlüsselung erfolgt in 16 Runden mit folgendem Vorgehen:
  - $L_i=R_{i-1}$
  - $R_i=L_{i-1}$ EXOR $f(R_{i-1},S_i)$
- Nach der 16. Runde ist L_iR_i der verschlüsselte 64 Bit Block

Entschlüsselung:

- Teile Chiffretext in in 64 Bit Blöcke (weiteres Verfahren für je einen Block erklärt)
- Teile 64 Bit Block in zwei Hälften L_0 und R_0
- Verschlüsselung erfolgt in 16 Runden mit folgendem Vorgehen:
  - $R_{i-1} = L_i$
  - Li-1= Ri EXOR f(Li,Si)
- Nach 16 Runden ist L_iR_i der entschlüsselte Datenblock



Zusammenfassung:

- Funktion F: Kombination aus XOR und festen Ersetzungsboxen (S-BOX)
- Teilschlüssel: Generierung von 48 Bit Teilschlüssel durch Shift und Permutation.
- Runden: 16
- Blockgröße: 64 Bit
- Schlüssellänge: 56 Bit

Auf welchem mathematischen Problem beruht RSA und wie funktioniert RSA?

Problem: Primfaktorzerlegung großer ganzer Zahlen dauert sehr lange, da man bis alle Primzahlen bis $\sqrt{n}$ probieren müsste. Deswegen p und q möglichst gleich groß wählen.

Funktion: Voraussetzung ist die Verschlüsselung von Zahlen zwischen 0 und r-1 mit $r \in \mathbb{N}$ und Klar- und Chiffretext sind $P = C = \mathbb{Z}_r$

1. Wähle zwei Primzahlen p und q mit n = p*q >= r.
2. Berechne Eulersche Funktion φ(n) = (p-1) * (q-1)
3. $k_e * k_d$ mod => k_e und k_d sind multiplikativ invers zu einander φ(n)

4. Wähle zu φ(n) teilerfremdes k_e mit $1 < k_e < \varphi(n)$
5. Berechne mittels erweitertem Euklidischen Algorithmus das multiplikativ Inverse zu k_e und φ(n)
   a. Ausgangsmatrix: $\begin{pmatrix} g & r \\ x & u \\ y & v \end{pmatrix} := \begin{pmatrix} \varphi(n) & k_e \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$
   b. Berechnungsmatrix: $\begin{pmatrix} g & r \\ x & u \\ y & v \end{pmatrix} := \begin{pmatrix} r & g \bmod r \\ u & x - u \cdot (g \; div \; r) \\ v & y - v \cdot (g \; div \; r) \end{pmatrix}$
   c. Endmatrix: $\begin{pmatrix} gdc(\varphi(n), k_e) = 1 & 0 \\ x & -k_e \\ MultiplikativInverses & -\varphi(n) \end{pmatrix}$
6. k_d = Muliplikativ Iverses, falls $k_d > 0$, sonst $k_d = (k_d + \varphi(n)) \bmod \varphi(n)$

Angriff: Schlüssel < 1024 Bit gelten mittlerweile als unsicher, da die derzeitige Rechenleistung ausreicht, da Schlüssel bis zu einer Länge 769 Bit (232 Dezimalstellen) inzwischen schnell geknackt werden.

- **Kleiner Exponent-Angriff** (low exponent attack)
  Prinzip basiert auf der Tatsache, dass oft mit k_e=3 verschlüsselt wird (hohe Geschwindigkeit). Sendet man eine Nachricht an drei unterschiedliche Empfänger mit k_e=3, so kann man Kongruenzen mittels Chinesischem Restwertsatz berechnen (nicht im Skript erklärt)
- **Gemeinsamer-Modulus-Angriff** (common modulus attack)
  Schwierig immer wieder neue große Primzahlen zu verwenden. Zwei Benutzern dasselbe n zu geben, führt hier zum Fehler:
  http://www-alt.physik.uni-muenchen.de/kurs/Computing/javaoo_07/source/RSA.pdf
  http://members.tripod.com/irish_ronan/rsa/attacks.html
- **Protokoll-Fehler-Angriff** (protocol failure)
  Wenn die Klartextzeichen einzeln verschlüsselt werden, kann der Klartext leicht ermittelt werden (EA 4.3)
- **Rotations-Angriff** (cycling attack)
  Wenn φ (φ(n)) nicht zu groß ist, wiederholtes potenzieren von x mit k_e. Das macht man i Mal bis man wieder den verschlüsselten Text hat. Klartext = x^k_e^i-1

Angriffsmethoden auf Verschlüsselung (Kryptoanalyse)?

- **Chiffretext-Angriff**
  Voraussetzung: nur Chiffretext ist bekannt
  Ziel: Klartext u. Schlüssel ermitteln
  Möglichkeiten:
  - Brute-Force-Attacke (vollständiges Durchsuchen des Schlüsselraumes)
  - Statistische Methoden (z.B. Häufigkeitsanalyse)
  - Ausprobieren „wahrscheinlicher Schlüssel“
  - Strukturanalyse des Verfahrens: Ermittlung einer Funktion als Ersatz der Entschlüsselungsfunktion, die ohne Schlüssel auskommt => Kryptosystem dauerhaft geknackt.

- **Bekannter-Klartext-Angriff**
  Voraussetzung: Klartext u. Chiffretext sind bekannt
  Ziel: Schlüssel ermitteln
  Möglichkeiten:
  - Brute-Force-Attacke (vollständiges Durchsuchen des Schlüsselraumes)

  Beispiel: WW II: Verschlüsselte Übertragung öffentlicher Reden ins U-Boot, dadurch konnten die Briten den Schlüssel ermitteln

- **Gewählter-Klartext-Angriff**
  Voraussetzung: Verschlüsselungfunktion u. Chiffretext ist bekannt, Schlüssel nicht
  Ziel: Schlüssel ermitteln
  Möglichkeiten:
  - Beliebig viele Nachrichten verschlüsseln und wenn als Ergebnis der Chiffretext entsteht, wurde der Schlüssel ermittelt (Kommt einer Brute-Force-Attacke in Umkehrung gleich)

- **Gewählter-Chiffretext-Angriff**
  Voraussetzung: Entschlüsselungsfunktion bekannt
  Ziel: Schlüssel ermitteln
  Möglichkeiten: ??

Wie funktioniert das public key Verfahren (im Mailaustausch)?

(1) Schlüsselpaar generieren
(2) Öffentlichen Schlüssel an potentiellen Sender schicken
(3) Sender verschlüsselt E-Mail mit öffentlichem Schlüssel
(4) Empfänger entschlüsselt mit privatem Schlüssel

Wie signiere ich eine Mail digital? Wie funktioniert das mit der elektronischen Unterschrift?

(1) Signatur erstellen
   a) Hashwert der E-Mail über kryptographische Funktion (z.B. MD5) berechnen.
   b) Verschlüsselung des Hashwerts mit privaten Schlüssel (z.B. von RSA-Verfahren)
- Signatur überprüfen
  - Hashwert der E-Mail über kryptographische Funktion (z.B. MD5) berechnen.
  - Entschlüsselung des verschlüsselten originalen Hashwerts mit dem öffentlichen Schlüssel
  - Vergleich der beiden Hashwerte auf Gleichheit

Kann man eine Mail signieren und gleichzeitig verschlüsseln?

Ja, das funktioniert wie folgt:

- Senderseite
  - Signatur der unverschlüsselten Mail wie oben erstellen
  - E-Mail samt Signatur mit öffentlichem Schlüssel des Empfängers verschlüsseln
- Empfängerseite
  - E-Mail mit privatem Schlüssel entschlüsseln
  - Signatur wie oben überprüfen

Welche Sicherheitsprotokolle gibt es im Internet?

- **IPSec**: symmetrische Verschlüsselung auf IP-Schicht
- **VPN** (Virtual Private Network): Baut auf IPSec auf und verschlüsselt mittels dieser Funktionalität die Datenkommunikation zwischen zwei oder mehr Netzwerkknoten (Stichwort: VPN-Tunnel)
- **SSL** (Secure Socket Layer): Verschlüsselung auf der Transportschicht. Somit kann eine Anwendung den Datenstrom verschlüsseln lassen (https). Das Verfahren setzt auf den Schlüsselaustausch von Diffie-Hellmann auf, mittels diesem zwei Kommunikationspartner sich auf einen Sitzungsschlüssel einigen können.
- **SSH** (Secure Shell): Protokoll zur gesicherten authentifizierten Kommunikation. Mittels SSH kann man inzwischen z.B. Kopieren von Dateien, X11 tunneln, Portweiterleitungen z.B. an einen Mailserver, Dateisysteme montieren (SSHFS) und vermutlich noch ein wenig mehr.
- **PGP** u. **S/MIME**: Mechanismen zum Verschlüsseln o. signieren von E-Mails, die meist auf DES und RSA aufsetzen. Als Signaturalogrithmus wird meist SHA1 (Secure Hash Alogorithm) oder MD5 verwendet.
  Anmerkung: Laut Wikipedia gilt SHA1 nicht mehr als richtig sicher (Kollisionsfreiheit)

Wie funktioniert Kerberos?

Vorab: Was mir nicht klar ist, wie der geheime Schlüssel des Clients zum Kerberos-Server gelangt. Wenn ich den RFC3244 richtig verstanden habe, wird ein Initialpasswort für den User vergeben und im AS hinterlegt. Mit diesem kann man dann schon verschlüsseln, weil der Benutzer das Passwort kennt und sich damit am System einloggen kann. Im Normalfall wird man beim ersten Mal anmelden aufgefordert, sein Passwort zu ändern. Mit dem alten Passwort kann nun eine verschlüsselte Übertragung des neuen Passworts durchgeführt werden. Das Passwort muss mind. einmal vom Client zum AS übertragen werden, da nur dort ggf. eine Passwortpolicy überprüft werden kann. Wenn das Passwort dann getauscht wurde, muss es nicht mehr übertragen werden. Somit senkt man die Anzahl der möglichen Angriffe auf die Anzahl der notwendigen Passwortänderungen über ein Jahr.

- Basiert auf dem Needham-Schroeder-Authentifizierungsprotokoll
- Folgende Abkürzungen erleichtern die Erklärung:
  a) AS = Authentication Server
  b) TGS = Ticket Granting Server
  c) SS = Service Server
  d) TGT = Ticket Granting Ticket

Funktion: Abgeschrieben von der Wikipedia, da mir der Anfang einfach nicht einleuchtet.

User Client-based Logon Steps:

1. A user enters a username and password on the client machine.
2. The client performs a one-way function (hash mostly) on the entered password, and this becomes the secret key of the client/user.

Client Authentication Steps:

1. The client sends a cleartext message to the AS requesting services on behalf of the user. Sample message: "User XYZ would like to request services".
   Note: Neither the secret key nor the password is sent to the AS.
2. The AS checks to see if the client is in its database. If it is, the AS sends back the following two messages to the client:
   - Message A: Client/TGS Session Key encrypted using the secret key of the client/user.
   - Message B: Ticket-Granting Ticket (which includes the client ID, client network address, ticket validity period, and the client/TGS session key) encrypted using the secret key of the TGS.
3. Once the client receives messages A and B, it decrypts message A to obtain the Client/TGS Session Key. This session key is used for further communications with TGS.
   Note: The client cannot decrypt Message B, as it is encrypted using TGS's secret key. At this point, the client has enough information to authenticate itself to the TGS.

Client Service Authorization Steps:

- When requesting services, the client sends the following two messages to the TGS:
  - Message C: Composed of the Ticket-Granting Ticket from message B and the ID of the requested service.
  - Message D: Authenticator (which is composed of the client ID and the timestamp), encrypted using the Client/TGS Session Key.
- Upon receiving messages C and D, the TGS retrieves message B out of message C. It decrypts message B using the TGS secret key. This gives it the "client/TGS session key". Using this key, the TGS decrypts message D (Authenticator) and sends the following two messages to the client:
  - Message E: Client-to-server ticket (which includes the client ID, client network address, validity period and Client/Server Session Key) encrypted using the service's secret key.
  - Message F: Client/server session key encrypted with the Client/TGS Session Key.

Client Service Request Steps:

1. Upon receiving messages E and F from TGS, the client has enough information to authenticate itself to the SS. The client connects to the SS and sends the following two messages:
   - Message E from the previous step (the client-to-server ticket, encrypted using service's secret key).
   - Message G: a new Authenticator, which includes the client ID, timestamp and is encrypted using

client/server session key.

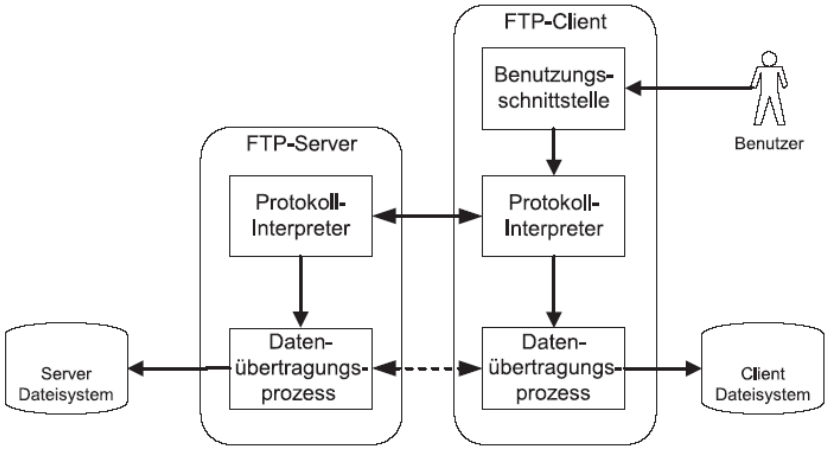
2. The SS decrypts the ticket using its own secret key to retrieve the Client/Server Session Key. Using the sessions key, SS decrypts the Authenticator and sends the following message to the client to confirm its true identity and willingness to serve the client:
   - Message H: the timestamp found in client's Authenticator plus 1, encrypted using the Client/Server Session Key.
3. The client decrypts the confirmation using the Client/Server Session Key and checks whether the timestamp is correctly updated. If so, then the client can trust the server and can start issuing service requests to the server.
4. The server provides the requested services to the client.

Nachteile:

- Single point of failure: It requires continuous availability of a central server. When the Kerberos server is down, no one can log in. This can be mitigated by using multiple Kerberos servers and fallback authentication mechanisms.
- Kerberos requires the clocks of the involved hosts to be synchronized. The tickets have a time availability period and if the host clock is not synchronized with the Kerberos server clock, the authentication will fail. The default configuration requires that clock times are no more than 10 minutes apart. In practice Network Time Protocol daemons are usually used to keep the host clocks synchronized.
- The administration protocol is not standardized and differs between server implementations. Password changes are described in RFC 3244.
- Since the secret keys for all users are stored on the central server, a compromise of that server will compromise all users' secret keys.
- A compromised client will compromise the user's password

KE 5 – Verteilte Dateisysteme

FTP als Protokoll: wie funktioniert es, welcher Verbindungsaufbau findet statt?



- Kommunikation über Steuerkanal (Port 21) und Datenkanal (Port 20).
- Die Verwendung eines Steuerkanals nennt man „out-of-band“
- FTP ist zustandsbasiert, deswegen kann man sofern es die Dateien verkraften, eine FTP-Übertragung nach einer Unterbrechung wieder an der Stelle aufsetzen, an der man war.
- 2 Prozesse sowohl auf dem Server als auch auf dem Client
  - Datentransferprozess: über den laufen die Daten und wird vom Server bei jeder Übertragung auf- und abgebaut
  - Protokoll-Interpreter: überträgt die Kommandos im Klartext und bleibt während der Sitzung geöffnet.
- Verbindungsaufbau: Client öffnet Steuerkanal zum Server und meldet sich mittels der Kommandos „USER“ und „PASS“ an. Anonymer FTP-Zugang mit Benutzername „anonymous“.
- Nach Anmeldung kann sich der Benutzer im FTP-Dateisystem bewegen und Verzeichnisse/Dateien löschen/anlegen bzw. up-/downloaden usw.
- Vor Übertragung kann es sinnvoll sein, eine Übertragungscodierung (ASCII, binär, EBCDIC) und ggf. einen Datenverbindungsport anzugeben
- Quittierung durch Server erfolgt durch dreistelligen Zahlencode

FTP: Welche Authentifizierung gibt es, warum ist FTP unsicher, welche Gründe sprechen dafür es trotzdem zu verwenden? Wie funktioniert anonymous FTP?

- Authenfizierung erfolgt über Benutzernamen und Passwort
- FTP ist unsicher, weil die Zugangsdaten unverschlüsselt übertragen werden.
- Für FTP spricht Plattformunabhängigkeit und sehr weite Verbreitung
- Anonymer FTP erfolgt über Eingabe des Benutzernamens „anonymous“, dies muss jedoch vom FTP-Server unterstützt sein.

HTTP als Protokoll: welche Möglichkeiten hat man, welche Daten kann man sich damit anschauen?

- HTTP nutzt TCP
- Objekt wird mit URL (Uniform Resource Locator) angegeben
  <protocol-id><protocol specific address>
- Verbindungen der Version 1.0 sind nicht persistent, die der Version 1.1 falls gewünscht.
- Zustandloses pull-Protokoll, welches jedes Datei einzeln überträgt
- Steuerinformationen werden in-Band gesendet
- HTTP-Nachricht besteht aus Kommandozeile (POST, GET), Nachrichtenheader und Nachrichtenkörper.

- Meist beschränkte Möglichkeiten, weil Befehle wie PUT o. DELETE meist nicht unterstützt werden. => kein echtes verteiltes Dateisystem im herkömmlichen Sinne
  - Dateien können nicht geändert werden
  - Verzeichnisse können nicht aufgelistet werden
  - Keine Besitzer von Dateien, Zugriff ja nein, Schutz nur von Verzeichnissen
  - Beschränkt sich auf den Download von Dateien

Caching-Strategien:

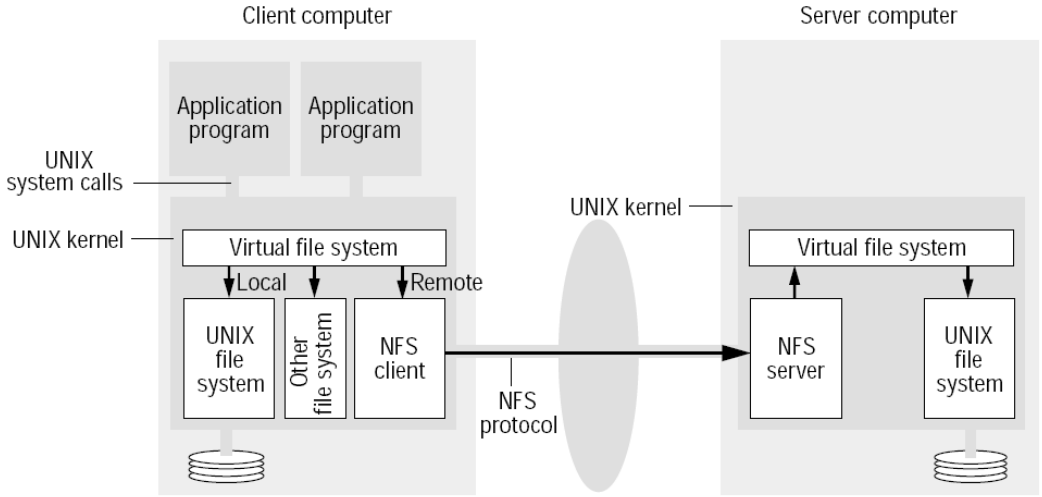
- Server-Cache, wenn sehr viele Clients immer wieder die gleichen Daten anfordern
- Cache in Form eines Proxy in Intranets, um den externen Verkehr zu reduzieren
- Client-Cache der vom Browser verwaltet wird
- Expires-Field in den Dateien führen zu Konsistenzwahrung in den letzten beiden Fällen

Was ist NFS?

Sun Network File System. Dient dazu, ein entferntes Dateisystem im lokalen Dateisystem einzubinden (virtuelles Dateisystem) um für den User eine Zugriffstransparenz auf entfernte Dateien zu realisieren.

Wie funktioniert NFS?

Figure 8.8 NFS architecture



NFS <= v3 zustandslos, ab v4 zustandsorientiert

- Die NFS Server- und Client-Module befinden sich im Kernel des OS (Unix, Linux).
- Anforderungen an Dateien im entfernten Dateisystem werden vom NFS-Client in NFS-Operationen übersetzt und mittels RPC an den NFS-Server weitergegeben
- NFS kann entweder über TCP o. UDP kommunizieren.
- Ein Port-Mapper-Dienst ermöglicht den Clients sich dem Namen nach zu Diensten in einem Host zu binden.
- NFS bietet *Zugriffstransparenz*, d.h. lokale Programme u. Benutzer können auf entfernte Dateien wie auf Dateien im lokalen Dateisystem zugreifen. Dazu wird über alle vorhandenen Dateisysteme ein virtuelles Dateisystem gestellt. Zum Zugriff auf Dateien werden Datei-IDs verwendet
- Ein *Automounter* (z.B. *autofs*) ermöglicht es, dass automatisch das entsprechende entfernte Verzeichnis gemountet wird, wenn ein Client auf einen leeren Mountpunkt verweist.
- NFS nutzt *Caching* zur Optimierung des Kommunikationsaufwandes.
  - Beim Server wird wie beim konventionellen Dateisystem das *Read-ahead* Caching (voraus lesen) eingesetzt.
  - Das *Delayed-write* Caching kann allerdings z.B. bei Serverabsturz Konsistenzprobleme erzeugen, da NFS bis Version 3 zustandslos operiert, ab Version 4 arbeitet NFS

zustandsorientiert.
- Daher gibt es seit NFS 3.0 zwei Optionen für die *write*-Operation:
  - *Write-through* Caching (geschriebenen Daten werden in Cache und auf HD geschrieben)
  - Daten in write-Operationen werden nur im Cache gehalten und erst durch eine *commit*-Operation des Clients auf HD übertragen. Der Client erhält erst dann eine Antwort auf das commit, wenn die Daten wirklich auf der HD sind.
- Clientseitig werden die Ergebnisse von read, write, getattr, lookup und readdir in den Cache gestellt. Die Clients müssen aber bei den Servern die Aktualität ihrer Daten im Cache sicherstellen.

Was hat der Administrator vor dem Einsatz von NFS zu tun?
- NFS-Server-Software installieren
- Lokale Dateisysteme für entferntes Mounten freigeben (üblicherweise über /etc/exports)
- Zugriffsliste erstellen, welche entfernten Hosts welche Dateisysteme mounten dürfen
- NFS-Server starten

Wie findet die Authentifizierung bei NFS statt (Sicherheitsaspekt)?
<u>NFS bis v3:</u>
- Authentifizierung über Benutzer- und Gruppen-ID, die bei jedem Zugriff mit übertragen werden und mit den Zugriffsrechten der Datei überprüft werden
- Übertragung erfolgt unverschlüsselt, was leicht zu einem Angriff mittels IP-Klau führen kann
- Einsatz von SecureRPC (kaum verbreitet; DES-Verschlüsselung des RPC-Protokolls) schafft hier Abhilfe

<u>NFS ab v4:</u>
- Authentifizierung kann mittels Kerberos erfolgen
- Speicherung der Benutzer-Id + Client-Adresse mit den Mount-Informationen
- Bei jedem Dateizugriff werden dann die gespeicherten Daten vom Mounten überprüft und bei Übereinstimmung Zugriff gewährt.

Wie genau geschieht das Mounten? Wann wird gemountet?
(2) Mount-Dienst auf NFS-Server unterstützt mounten entfernter Dateisysteme (Anfragen von Clients).
(3) Client nutzt mount-Kommando und gibt dabei NFS-Server Hostnamen und Pfad des entfernten Dateisystems an und wo es im lokalen Dateisystem eingehangt werden soll (z.B. mount smg1:/nim/lpp /mnt)
(4) Mount-Befehl kommuniziert mittels RPC über das Mount-Protokoll mit dem Mount-Dienst auf dem NFS-Server, was gemountet werden soll
(5) Zurückgelieferter Dateihandle wird über die VFS-Schicht an den NFS-Client übergeben, welcher dann die Kommunikation mit dem NFS-Server und den Mount durchführt.
Dabei gibt es zwei Arten
  f. Hard-Mount: Prozess wird unterbrochen bis Mount erfolgt ist, (auch wenn der NFS-Server nicht erreichbar ist)
  g. Soft-Mount: NFS-Client gibt Fehlermeldung an den Prozess zurück, wenn der NFS-Server nicht erreichbar ist.

<u>Es gibt drei Zeitpunkte, wann gemountet wird:</u>
- Wenn der Benutzer das Mount-Kommando ausführt
- Beim Systemstart, falls ein NFS-Mount in der „/etc/fstab“ steht
- Durch den Automounter, sobald auf ein Verzeichnis zugegriffen wird, welches gemäß Konfiguration ein NFS-Mount ist und durch den Automounter bedient werden soll.

Wie ist die Arbeitsweise von CVS? Was genau ist CVS?
- CVS (Concurrent Version System) ist ein Versionsverwaltungssystem
- Erfüllt neben Datenkonsistenz auch Anforderungen, die durch die Gruppenarbeit entstehen:
  - Isolation: Benutzer können isoliert von einander eine Datei ändern
  - Integration: Änderungen können wieder in die Gruppe integriert werden

- Aktualität: einfach aktuelle Dateiinhalte zu erhalten
- Nachvollziehbarkeit: von Änderungen
- Gruppenwahrnehmung: Benutzer werden über Aktivitäten anderer Benutzer informiert

Funktion:

- Ablage aller Versionen in einer Datei in gemeinsamen Repository
- Unterstützung der Benutzer bei Auswahl, Erzeugung und Manipulation von Versionen
- Änderungen an einer bestehenden Version sind nicht möglich, eine Änderung führt immer zu einer neuen Version => Nachvollziehbarkeit der Änderungen
- 3 Phasen der Arbeit:
  - check out (CVS update) => Aktualität
  - Bearbeitung der Dateien => Isolation
  - check in (CVS commit) => Integration

Was passiert, wenn wir die gleiche Datei bearbeiten, kann es zu Problemen kommen?

- Wenn zwei Benutzer zur selben Zeit an derselben Datei arbeiten, führt dies dazu, dass zwei Versionen ins Repository eingecheckt werden. Daraus resultiert ein **Versionsbaum**.
- Will man zu viele Verzweigungen vermeiden, kann man vor dem check in ein Update durchführen, um zu sehen, ob sich an der Datei inzwischen etwas geändert hat.
  - Falls nein, ist alles i.O. und die Datei kann eingecheckt werden.
  - Falls ja, führt der Update zu einem lokalen Merge der Version im Repository und der lokalen Datei

  Mergekonflikt:
  Kann auftreten, wenn z.B. derselbe Bereich in der Datei verändert wird oder semantische Abhänigkeiten auftreten. Dann muss der Benutzer eigenverantwortlich aus den beiden Dateien eine sinnvolle erstellen und ins Repository einchecken.

Was ist beim Merge durch das 2-Wegevergleichsverfahren nicht abgedeckt?
Es ist unmöglich, aus einer gemeinsamen Ursprungsversion eine gelöschte Zeile zu erkennen. Deshalb Erweiterung zum 3-Wegevergleich wo auch die Vorversion in den Vergleich miteinbezogen wird. Es werden dabei für die beiden parallelen Texte die Unterschiede (hinzugefügte, gelöschte Zeilen) zu ihren Vorversionen ermittelt.

In welchem Fall führen parallele Änderungen nicht zu Problemen?
Wenn es keine semantischen Abhängigkeiten gibt bzw. wenn nicht dieselben Dokumentbereiche geändert wurden.

Wie wird die Gruppenwahrnehmung in CVS realisiert?
In CVS gibt es dafür einen allgemeinen Mechanismus, der im Anschluss an commit-Aktionen ausgeführt werden kann. Dieser lässt sich z.B. so konfigurieren, dass beim Erstellen einer neuen Version durch einen Benutzer automatisch interessierte Benutzer über E-Mail darüber informiert werden. Die Benutzer abonnieren dazu den Projektarbeitsbereich indem sie ihre E-Mail Adresse in eine Benachrichtigungsliste eintragen. Das System verschickt dann nach jeder Änderung Informationen über die betroffene Datei und den Versionskommentar an die Abonnenten.

Kurseinheit 6: Transaktionen

1. Wechselseitiger Ausschluss (Buch 6.3)
2. ACID-Eigenschaften (Buch 1.3.2)
3. Verteilter Commit (Buch 8.5)
4. Deadlocks (Heft 6.2)
5. Recovery (Buch 8.6)
6. Transaktionen in kooperativen Anwendungen insbes. COAST (Heft 6.3.2)

1. Wechselseitiger Ausschluss (Buch 6.3) = Mutual Exclusion

- Prozesse haben den Bedarf gleichzeitig auf die gleiche Ressource zuzugreifen
- damit die Ressource nicht zerstört wird (z.B. Erzeugung von Inkonsistenz) existieren verschiedene Algorithmen um einen wechselseitigen Ausschluss zu gewährleisten

a) Zentralisierter Algorithmus
- ein Prozess fungiert als Koordinator, welcher Zugriff erteilt oder verwehrt. Verwehrte Zugriffe landen in einer Queue.

Vorteile:
- leichte Implementierung, da simpel
- fair, da Zugriffsreihenfolge fest definiert (FIFO)
- no starvation (kein Verhungern)

Nachteile:
- Koordinator ist ein SPF (Single Point of Failure) bei Crash
- Koordinator ist ein Performance Bottleneck

b) Dezentralisierter Algorithmus
- Jede Ressource wird n mal repliziert, es existieren auch n Koordinatoren
- Stimmenbasierte Abstimmung auf Zugriff, es werden mindestens n/2+1 positive Stimmen benötigt, um Zugriff zu erhalten

Vorteile:
- Resistenter gegen Ausfälle als der zentralisierte Algorithmus

Nachteile:
- Wenn eine Ressource sehr oft benötigt wird, kann es sein, dass niemand Zugriff erhält, da niemand die nötigen Stimmen erhält (Starvation)
- Nicht effektiv

c) Verteilter Algorithmus (kein Koordinator, Selbstorganisation)
- basierend auf Ricart und Agrawala (1981)
- Voraussetzung ist eine totale Ordnung aller Ereignisse im System (basierend wiederrum auf Lamports Algorithmus zur Uhrsynchronisierung 1978)
- Prozessanfragen enthalten den Namen der Ressource, die eigene Prozess-ID, sowie einen globalen Timestamp
- Prozessanfragen werden an alle anderen beteiligten Prozesse gesendet, es können folgende Antworten auftreten:
    * OK message, wenn der Empfänger die Ressource nicht benötigt
    * keine message, wenn der Empfänger die Ressource bereits benutzt, Anfrage landet in einer Queue
    * Wenn beide Prozesse die Ressource benutzen wollen, gewinnt diejenige mit dem kleineren Timestamp --> Antworten siehe oben

Vorteile:
- kein SPF (Single Point of Failure)

Nachteile:
- Entweder es wird Multicast für die Anfragen benutzt oder jeder Prozess kennt alle anderen beteiligten Prozesse --> Aufwändig.
- Langsamer, komplizierter, teurer, weniger robust als der zentrale Algorithmus

d) Token Ring Algorithmus

- basierend auf einem Bus-Netzwerk, in dem ein Token pro Ressource ständig zwischen den Prozessen kreist (logischer Ring)
- die Prozessreihenfolge ist nicht wichtig, jedoch muss jeder Prozess seinen Nachfolger im Ring kennen.

Vorteile:
- leichte Implementierung, da simpel
- fair, da Zugriffsreihenfolge fest definiert (FIFO)
- no starvation (kein Verhungern)

Nachteile:
- "Verlieren" des Token, d.h. dieser muss neu generiert werden. Problem ist die Feststellung, dass der Token verloren wurde
- wenn ein Prozess crasht, kann die Kette unterbrochen werden (wenn jeder Prozess alle Prozesse kennt, kann das Problem umgangen werden)

2. ACID-Eigenschaften (Buch 1.3.2)

= Vier Eigenschaften, die jede Transaktion haben muss

Atomic:	jede Transaktion geschieht entweder komplett oder gar nicht
Consistent:	Zustände sind immer konsistent. Teiltransaktionen sind von außen **nicht** sichtbar.
Isolated:	Transaktionen die parallel laufen, führen zum gleichen Ergebnis, als wenn diese nacheinander laufen
Durable:	Ergebnisse abgeschlossener Transaktionen (Commited) sind unveränderbar durch z.B. Failure/Systemcrash etc.

3. Verteilter Commit (Buch 8.5)

Es existieren u.A. 1phasige, 2phasige und 3phasige Commits.
Verteilte Commits werden benötigt, um sicher zu stellen, dass Transaktionen entweder gar nicht oder komplett (**A**tomic) in einem **verteiltem System** (mehrere beteiligte Prozesse) ausgeführt werden.

a) one-phase-commit
- Koordinator-basierend
- Koordinator teilt allen Teilnehmern mit einen Commit auszuführen
- es besteht keine Möglichkeit für einen Prozess den Koordinator zu kontaktieren (z.B. wenn der Prozess den Commit nicht ausführen kann (z.B. durch referential constraint verletzung))
- Koordinator is SPF (Single Point of Failure)

b) two-phase-commit (Gray 1978)
- Koordinator-basierend
- Koordinator frägt an, ob Prozesse Commiten könnten (VOTE_REQUEST)
- Wenn alle Prozesse bestätigten (VOTE_COMMIT), wird ein globaler Commit vom Koordinator ausgelöst (GLOBAL_COMMIT).
- Sollte ein Prozess nicht commiten können (VOTE_ABORT wird zurück gesendet), so wird die Transaktion komplett abgebrochen (GLOBAL_ABORT)
- Koordinator ist SPF (Single Point of Failure)

c) three-phase-commit
- Umgeht das Problem des SPF-Koordinators des two-phase-commits
- Vermeidet blockierende Prozesse (im Gegensatz zum two-phase-commit)
- Stabileres Verhalten wenn Prozesse oder der Koordinator crashed (autom. Rollback etc., Alle Zustände können rekonstruiert werden (besser als beim 2phasigen Commit))

4. Deadlocks (Heft 6.2)
- = Systemverklemmung, gegenseitiges Aussperren, unerwünschter Zustand
- es existieren Verfahren zur Erkennung und Vermeidung (nicht Kursbestandteil)

5. Recovery (Buch 8.6)
- Recovery = Wiederherstellung eines gültigen Zustandes
- backward recovery = Wiederherstellung eines alten/gesicherten Zustandes durch "zurückdrehen" auf einen Checkpoint (Beispiel: erneutes Senden eines verlorengegangenen Paketes = br)
- forward recovery = Erzeugung eines neuen gültigen Zustandes

(Beispiel: Wiederherstellung eines verlorengegangenen Paketes durch erasure correction)
- Checkpoint = sichern des aktuellen Zustandes (konsistenter Zustand) zu Wiederherstellungszwecken (teure Operation)
- message logging = mitloggen von messages (entweder sender-based oder receiver-based)
Mit Hilfe von Checkpoints **und** message logging kann der Aufwand/die Kosten reduziert werden
- Wiederherstellung mit Hilfe von Checkpoints/message logging: Checkpoint-Zustand wiederherstellen, messages replayen
Stable storage: Speicherbereich, der auch größere Failures übersteht, ohne dass Daten verloren gehen (zur Speicherung der Recovery-Daten), z.B. Raid Drive Arrays und/oder Datensicherungen
- distributed checkpoint = checkpointing in einem fault-tolerant distributed system (jeder Prozess erzeugt individuelle lokale Checkpoints = independent checkpointing)
- Bei der Wiederherstellung eines distributed checkpoints muss eine recovery line gefunden werden (wiederherstellbarer konsistenter Zustand des Gesamtsystems)
- coordinated checkpointing: koordiniertes Auslösen der Checkpoint-Generierung. Vorteil: generierter distributed checkpoint ist immer global Konsistent, Domino-Effekt beim Wiederherstellen wird vermieden.
- incremental snapshot: Auslösen der Checkpoint-Generierung nur bei Prozessen, deren Zustand sich geändert hat (die Nachrichten empfangen haben)

6. Transaktionen in kooperativen Anwendungen insbes. COAST (Heft 6.3.2)

- optimistische Nebenläufigkeitskontrolle:
mehrere Befehle werden zusammen als Transaktion behandelt. Das System überwacht Zugriffe auf replizierte Objekte und überwacht Veränderungen in einem Transaktionslog. Clients werden dabei über Veränderungen informiert.
Das Log enthält auch die ursprünglichen Zustände, um diese bei Kollissionen wiederherstellen zu können.
Nebenläufigkeitskontrolle: werden mehrere Transaktionen parallel ausgeführt, so wird festgestellt, ob diese beiden ausgeführt werden können (d.h. sich nicht gegenseitig beeinflussen), oder ob ein Client seine Transaktion rückgängig machen muss.
Einfache Nebenläufigkeit: es darf keine parallelen Schreiboperationen auf einem Objekt geben.
Verbesserte Nebenläufigkeit: Nebenläufig sind dabei Aktionen, die das gleiche Objekt betreffen, sich aber nicht gegenseitig ausschließen, wie z.B. die gleichzeitige Änderung von Farbe und Größe eines 3D-Körpers. Die Erkennung von Nebenläufigkeit wird dabei über eine Kommutativitätstabelle realisiert. Operationen, die kommutativ (miteinander vertauschbar) sind, können als unabhängig von einander betrachtet werden.

Kurseinheit 7: Computer Supported Cooperative Work

CSCW = Computer Supported Cooperative Work = Groupware = rechnergestützte Zusammenarbeit
CSCL = Computer Supported cooperative/collaborative learning (Verteilung von Wissen durch z.B. Vorlesungen)

Beispiele:
* Kooperative Bearbeitung von Übungsaufgaben in der synchronen kooperativen Lernumgebung FUB
* Kooperatives Schreiben eines Buches mit BSCW (Basic Supported Cooperative Work)
* Gruppenarbeit in verteilten Organisationen mit Workware innerhalb einer Firma

FUB:
* Bilden einer Lerngruppe
* Durchführen der kooperativen Lernübung (gemeinsamer Arbeitsbereich)
* gemeinsame Dateien inkl. Versionssystem (alt/neu), unterstützt kein Merge
* Chat-Kommunikation
* Phasenwechsel
* Einreichen der gemeinsamen Lösung

Buch:
* Gemeinsamen Arbeitsbereich einrichten
* Groben Zeitplan festlegen
* Gliederung entwickeln
* Kapitel verteilen und Arbeitsplan erstellen
* Kapitel schreiben
* Kapitel gegenseitig begutachten
* Kapitel verbessern
* Ergebnisse zum ersten Entwurf integrieren

Groupware:
* Integriertes Workflow-Management
* Aufgaben erzeugen/starten
* Aufgaben entlang einer Prozesskette verteilen
* Aufgaben rückmelden
* Aufgaben überwachen

Einflussfaktoren der Gruppenarbeit:
* Gemeinsame Artefakte
* Kommunikation
* Koordination

Phasen bei der Gruppenarbeit:
* Individuelle Arbeit
* Asynchrone Kooperation
* Synchrone Kooperation

Es existieren Rollen in der Gruppe, mit verschiedenen sozialen Funktionen bzw. Rechte und Pflichten.

Abgrenzung zwischen CSCW und CSCL:
* CSCW als Bindeglied zwischen Individualanwendung und Organisationsunterstützung
* CSCW als Konvergenzbereich verschiedener Entwicklungsbereiche, z.B. HCI, MIS, Gruppenpsychologie, Soziologie, etc.
* CSCL benutzt **zusätzlich** Forschungsmethoden und Forschungsergebnisse verschiedener Disziplinen.

Klassifikation von CSCW-Systemen in einer Raum/Zeit-Matrix:

Raum/Zeit	Gleiche Zeit (synchron)	Verschiedene Zeit (asynchron) / vorhersehbar	Verschiedene Zeit (asynchron) / nicht vorhersehbar
Gleicher Ort	Face-to-Face Sitzungsraum	Organisation von Schichtarbeit	Schwarzes Brett
Verschiedener Ort (vorhersehbar)	Videokonferenz	E-Mail	Kooperatives Schreiben
Verschiedener Ort (nicht vorhersehbar)	Mobilfunkkonferenz	Asynchrone rechnergestützte Konferenz	Vorgangsbearbeitung

Fallstudie: DOLPHIN

Basiskonzepte in kooperativen Systemen:
* E-Mail, News
* Chat
* Instant Messaging

* Koordination (Prozesse, Rollenkonzepte, Gruppenkalender, Workflow, gemeinsame Dokumente
* Kooperation (gemeinsame Arbeitsbereiche, Benutzerlisten, Application Sharin g, WYSIWIS)

Mit Hilfe von COAST (ist ein Framework) können CSCW-Systeme entwickelt werden.