FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Fehlertoleranz in verteilten Systemen

Vortrag im Rahmen des Masterseminars „Fehlertolerante Systeme“

Betreuer: Dr.Volkmar Sieh

Autor: Gona Fatah

Erlangen, 16.12.2014

Agenda

- Verteilte Systeme
  - Definition
  - Nutzen
  - Wünschenswerte Eigenschaften
  - Beispiele
- Fehlertoleranz in verteilten Systemen
  - Verlässliche Systeme
  - Störungen in einem System
  - Fehlermodell
  - Protokolle
    - TCP
    - Commit-Protokoll
      - Beschreibung
      - 2-Phasen Commit
      - 3-Phasen Commit
  - Leader Election Service
    - Nutzungsmotivation
    - Quality of Service
    - Leader Election Module
      - Fehlerdetektor
      - Link Quality Estimator
      - Scheduler
    - Architektur
  - Leader Election Service -Evaluation

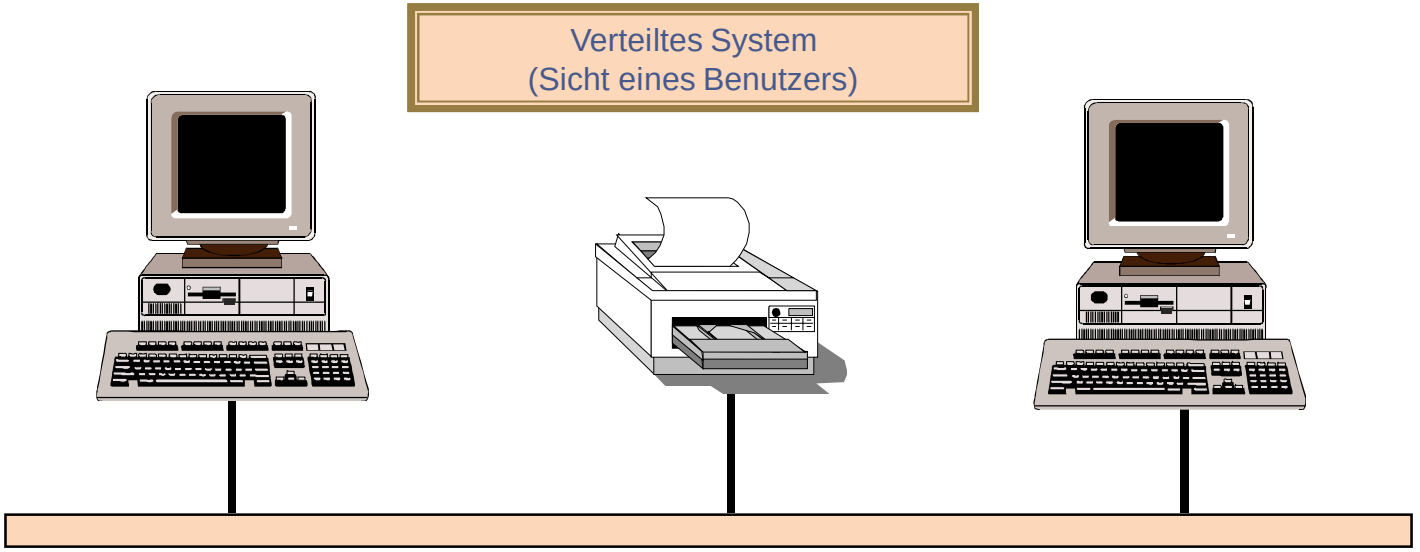
- verteilte Systeme
  - Definition
  - Nutzen
  - wünschenswerte Eigenschaften
  - Beispiele

Was ist ein verteiltes System?

Eine *praxisorientierte* Beschreibung:

Ein verteiltes System besteht aus einer Menge eigenständiger Rechner

- die durch ein Computernetzwerk miteinander verbunden sind
- und mit einer Software zur Koordination ausgestattet sind

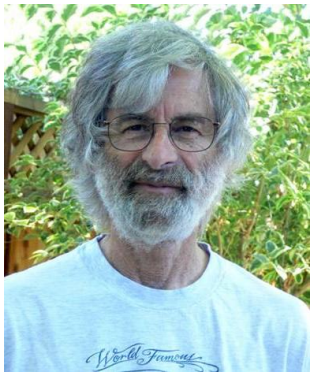


Was ist ein verteiltes System?

Beschreibung aus der Fachliteratur:

„Ein verteiltes System ist ein System, mit dem ich nicht arbeiten kann, weil irgendein Rechner abgestürzt ist, von dem ich nicht einmal weiß, dass es ihn überhaupt gibt.“

Leslie Lamport



„Ein verteiltes System ist eine Kollektion unabhängiger Computer, die den Benutzern als ein Einzelcomputer erscheinen.“

Andrew S. Tanenbaum

Wozu wird ein verteiltes System gebraucht?

- Kommunikations-Verbund

(Übertragung von Daten, insbesondere Nachrichten, an verschiedene, räumlich getrennte Stellen; z.B. E-Mail)

- Informations-Verbund

(Verbreiten von Information an interessierte Personen/Systeme; z.B. WWW)

- Daten-Verbund

(Speicherung von Daten an verschiedenen Stellen: bessere Speicherauslastung, erhöhte Verfügbarkeit, erhöhte Sicherheit)

- Last-Verbund

(Aufteilung stoßweise anfallender Lasten auf verschiedene Rechner: gleichmäßige Auslastung verschiedener Ressourcen)

- Leistungs-Verbund

(Aufteilung einer Aufgabe in Teilaufgaben:Verringerte Antwortzeiten)

- Kapazitäts-Verbund

(Ausnutzung sämtlicher zur Verfügung stehender Rechenkapazität)

Wünschenswerte Eigenschaften in verteilten Systemen

- Gemeinsame Ressourcennutzung
- Nebenläufigkeit
- Skalierbarkeit
- Sicherheit
- Transparenz
- Offenheit
- Fehlertoleranz

Gemeinsame Ressourcennutzung

- Beispiele für gemeinsame Ressourcen:
  - Hardware:
    - Drucker, Festplatten, etc.
  - Daten:
    - Datenbankobjekte, Dateien, etc.
  - Client-Server Model:
    - Server verwaltet Ressourcen, die Clients nutzen
  - Verteilte Services
    - komplexer Dienstleistungen, die über Netz aufrufbar sind, z.B. Bezahldienste, Flugbuchung, etc.
- Problematik:
  - Regelung nebenläufiger (paralleler) Zugriffe
  - Fragen der Konsistenz und der Fehlertoleranz

Nebenläufigkeit (Concurrency)

- Nebenläufigkeit– erlaubt, dass mehrere Prozesse gleichzeitig mit denselben gemeinsam genutzten Ressourcen arbeiten, ohne sich gegenseitig zu stören

- Nebenläufigkeit kann es z.B. bei
  - Clients (Anwendungsprogramme, z.B.Videokonferenz)
  - Servern (Zugriff auf Ressourcen, z.B. Datei) geben.

- Wichtige Aspekte:
  - Synchronisation der Aktivitäten,
    - z.B. um Daten konsistent zu halten.
  - Verbesserung des Durchsatzes und Performance durch Parallelisierung

Skalierbarkeit

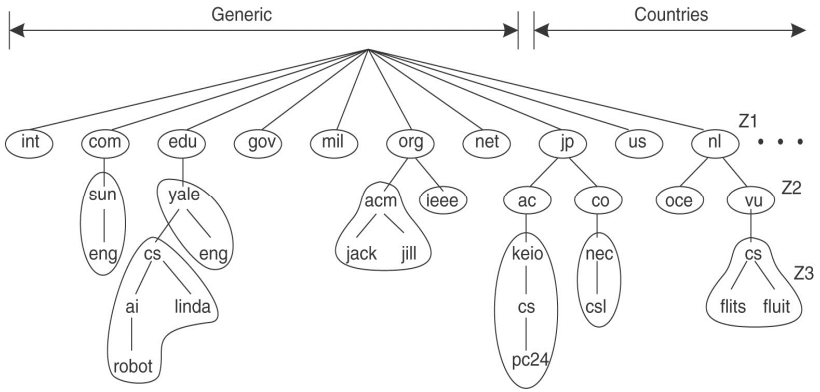
- Skalierung – erlaubt, dass sich das System und

die Applikationen vergrößern können, ohne dass die Systemstruktur oder die Applikationsalgorithmen geändert werden müssen.

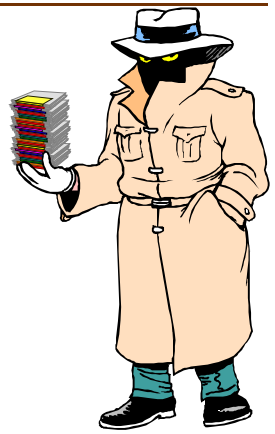
- Vergrößerung von Softwaresystemen bedeutet:
  - Hinzufügen weiterer Hardware
  - Durchsatz (Zugriffe) des Systems
  - Anzahl paralleler Nutzeranforderungen erhöht werden kann

- Beispiel:

Verteilung des DNS-Namensraum



Sicherheit



- Vertraulichkeit:

Daten können nur von dem gewünschten Empfänger gelesen werden.

- Authentizität:

Die Daten wurden tatsächlich von der Person gesendet, die behauptet, der Sender zu sein.

- Integrität:

Die Daten wurden während der Übertragung nicht verändert.

- Verfügbarkeit:

Ein Dienst darf durch eine (Denial of Service) Attacke nicht außer Kraft gesetzt werden.

Transparenz

- **Transparenz** :

Anwender sieht das System als Ganzes und nicht als Sammlung voneinander unabhängiger Komponenten.

- ISO (*I*nternational *S*tandards *Organisation*) identifizieren einige **Formen der Transparenz**:
  - **Zugriffstransparenz** Zugriff auf lokale und entfernte Ressourcen mit identischer Operationen.
  - **Positionstransparenz** Zugriff auf Ressourcen, ohne ihren tatsächlichen Ort kennen zu müssen.
  - **Nebenläufigkeitstransparenz** mehrere Prozesse nutzen gleichzeitig die selben/gemeinsamen Ressourcen, ohne sich gegenseitig zu stören.
  - **Fehlertransparenz** erlaubt das Verbergen von Fehlern, so dass Benutzer ihre Aufgaben erledigen können, auch wenn Hardware- oder Softwarekomponenten ausgefallen sind.
  - Weitere Transparenz-Formen:
    - Skalierungstransparenz
    - Replikationstransparenz
    - Mobilitätstransparenz
    - Leistungstransparenz

Offenheit

- Offenheit:

Ein offenes verteiltes System bietet Dienste nach Standardregeln an. Bzgl.:

  - Nutzung durch andere Systeme
  - Nutzung anderer
    - Systeme
    - Sprachen
    - Betriebssystemen
  - Verwendung standardisierter Schnittstellen bei
    - Kommunikationsmechanismen für den Zugriff auf gemeinsame Ressourcen
    - Datenformattechnologien

Fehlertoleranz

- Fehler erkennen:

z.B. durch Prüfsummen. Nicht erkennbar ist z.B. Absturz eines entfernten Servers

- Fehler maskieren:

Erkannte Fehler verbergen oder abschwächen, z.B. Wiederholung von Nachrichten

- Fehler tolerieren:

z.B. durch Redundanz, Timeout

- Wiederherstellung nach Fehlern:

z.B. Rückkehr in einen sicheren Zustand, wenn ein Fehler entdeckt wird

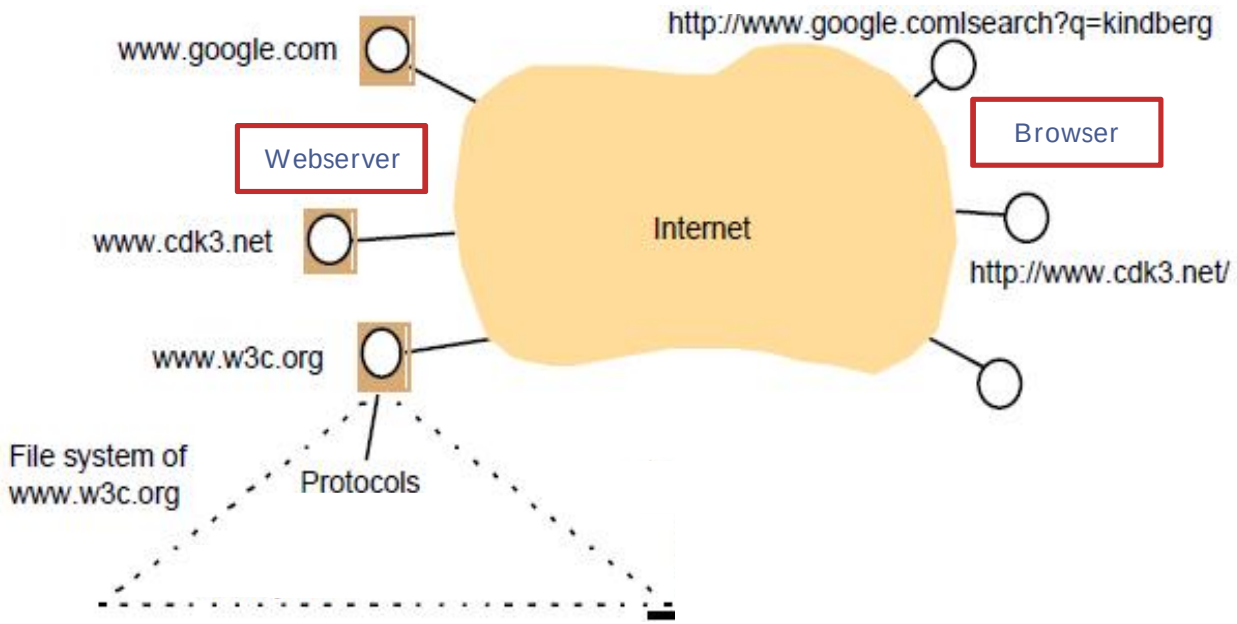
- *Redundanz*:

Fehlertoleranz durch redundante Komponenten

  - doppelte Komponenten
  - doppelte Dienste

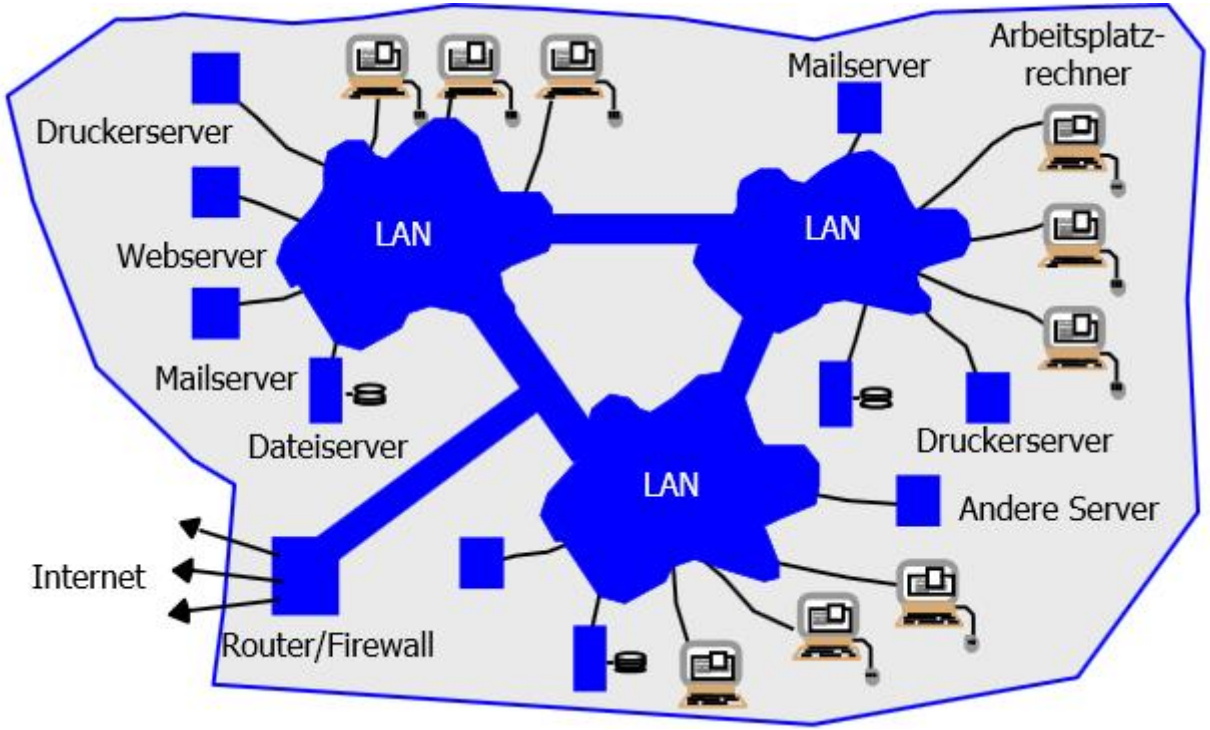
Beispiel 1: Das World Wide Web

- Bekanntester Beispiel für verteilte Systeme:
  - Internet als verteiltes System



Quelle: Prof. Thai

Beispiel 2: Intranet



Quelle: Institut für Informatik - Universität Paderborn

Agenda

- Verteilte Systeme
  - Definition
  - Nutzen
  - Wünschenswerte Eigenschaften
  - Beispiele
- Fehlertoleranz in verteilten Systemen
  - Verlässliche Systeme
  - Störungen in einem System
  - Fehlermodell
  - Protokolle
    - TCP
    - Commit-Protokoll
      - Beschreibung
      - 2-Phasen Commit
      - 3-Phasen Commit
  - Leader Election Service
    - Nutzungsmotivation
    - Quality of Service
    - Leader Election Module
      - Fehlerdetektor
      - Link Quality Estimator
      - Scheduler
    - Architektur
  - Leader Election Service -Evaluation

Fehlertoleranz in verteilten Systemen

- Verlässliche Systeme
- Störungen in einem System
- Fehlermodell
- Protokolle
  - TCP
  - Commit-Protokoll
    - Beschreibung
    - 2-Phasen Commit
    - 3-Phasen Commit

Wann gilt ein verteiltes System als „verlässlich“?

Wenn die folgenden Anforderungen erfüllt sind:

- **Verfügbarkeit** (Availability)

  Wahrscheinlichkeit (in Prozent), dass ein System korrekt arbeitet

- **Zuverlässigkeit** (Reliability)

  System läuft fortlaufend ausfallfrei (in einem Zeitintervall)

- **Funktionssicherheit** (safety)

  Es passiert nichts schlimmes, wenn ein System kurzzeitig nicht korrekt arbeitet. (Negativ-Beispiel: Leitsysteme in Kernkraftwerken oder in der Raumfahrt)

- **Wartbarkeit** (Maintainability)

  Wie leicht kann ein ausgefallenes System repariert werden?

→ *Systeme mit diesen Eigenschaften gelten als „**verlässlich**“*

Störungen in einem System

- **Systeme fallen aus**, wenn sie ihre Zusagen nicht mehr einhalten
  - Ursache dafür wird als **Störung (Fehler)** bezeichnet

- **Aufbau verlässlicher Systeme**
  - Kontrolle der Störungen
  - Möglichkeit, Störungen zu tolerieren bzw. zu maskieren

- Arten von Störungen
  - **Vorübergehend**: Einmaliger Vorfall (z. B. Stromausfall)
  - **Wiederkehrend:** Unberechenbares Wiederauftreten (z. B. Knotenausfall)
  - **Permanent:** Fehler bleibt bestehen, bis die fehlerhafte Komponente ersetzt wird (z. B. fehlerhafte HW)

Fehlermodell

- Das Fehlermodell in VS dient der Klassifikation von Fehlermöglichkeiten und der Abstraktion ihre Ursachen

Klassifikation von Fehlern	
Absturzausfall	Server steht, ständiger Dienstausfall
Zeitbedingter Ausfall	Antwortzeit eines Servers liegt außerhalb des festgelegten Zeitintervalls
Ausfall korrekter Antwort	Antwort eines Servers ist falsch (weil ein falscher Wert geliefert wird, oder ein falscher Programmablauf genutzt wird)
Byzantinischer oder zufälliger Fehler	Ein Server erstellt zufällige Antworten zu zufälligen Zeiten
…..	……

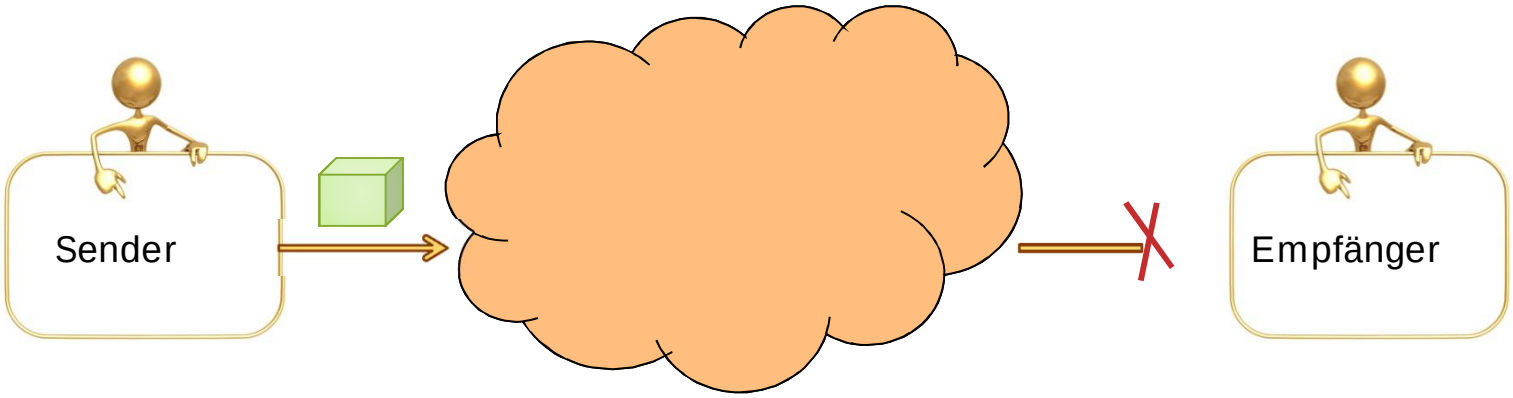
Fehlermodell

Im Rahmen dieser Arbeit relevanten Fehlerklassen:

Klassifikation von Fehlern	
Fehler auf der Bitübertragungsschicht (Physical Layer)	
Nachrichtenverlust	Nachrichten gehen verloren
doppelte Nachrichten	Eine Nachricht kommt mehrmals beim Empfänger an
verfälschte Nachrichten	Eine Nachricht wird verfälscht an ein Empfänger gesendet
Fehler auf der Anwendungsschicht (Application Layer)	
Nachricht kommt irgendwann an	Eine Nachricht kommt verzögert bei Empfänger an
Knotenausfall	Ein Prozess sendet und empfängt ab einer gewissen Zeit keine Nachrichten mehr

Fehlermodell (Klassifizierung von Fehlern)

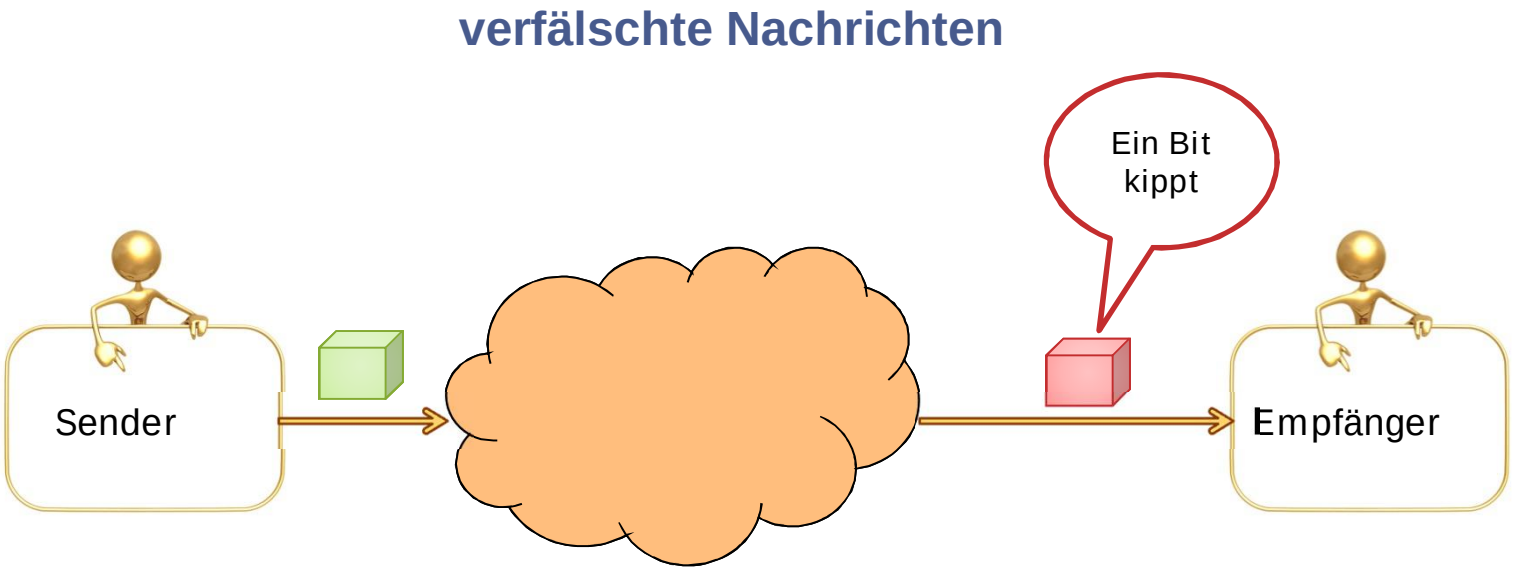
verlorene Nachricht



→Datenverlust wegen Verbindungsunterbrechung

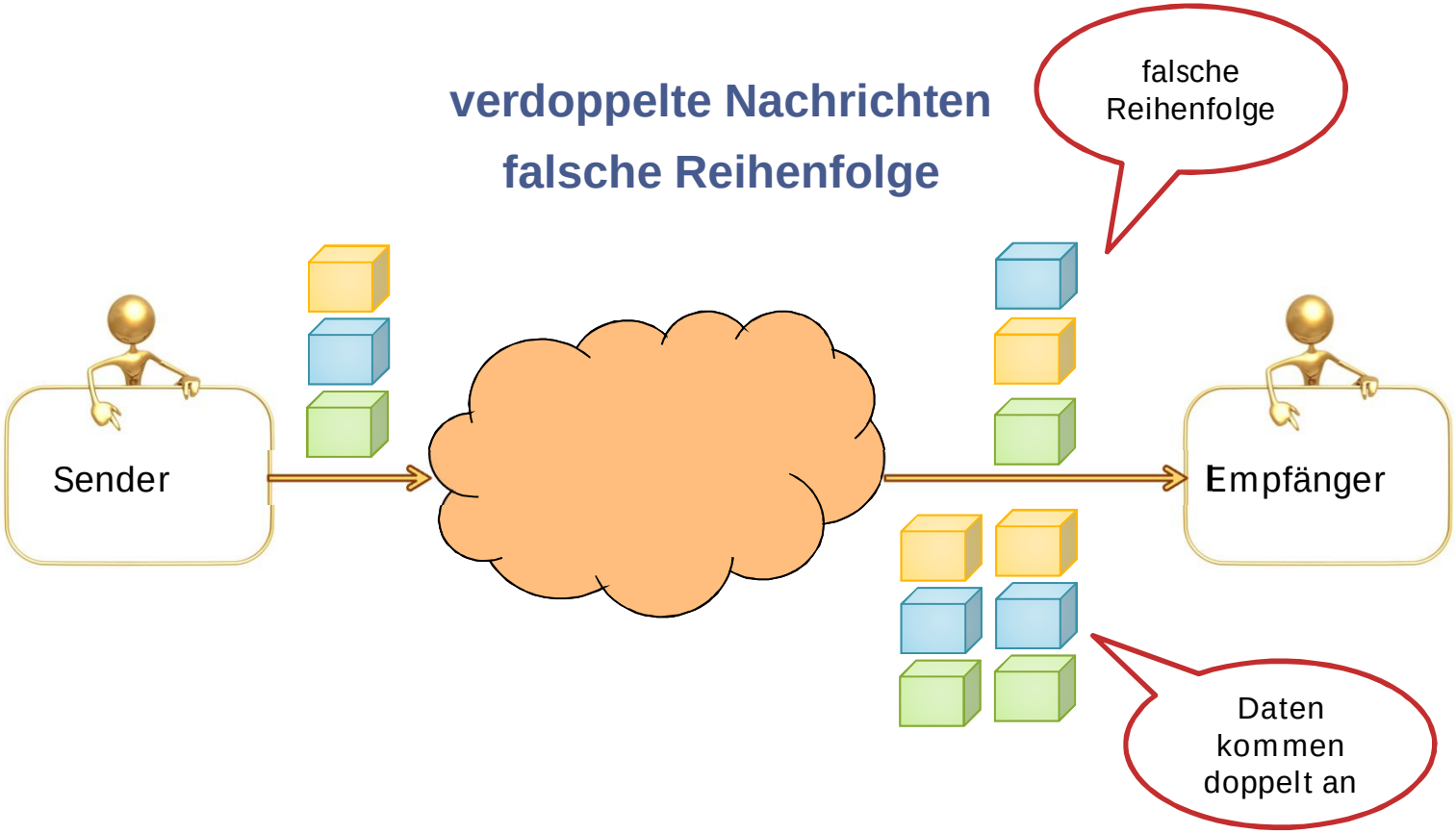
→ **Fehler beheben:** Nach ausbleibendem ACK erfolgt eine zeitgesteuerte Sendewiederholung

Fehlermodell (Klassifizierung von Fehlern)



→ Fehler beheben: **Prüfsummen**vergleich

Fehlermodell (Klassifizierung von Fehlern)



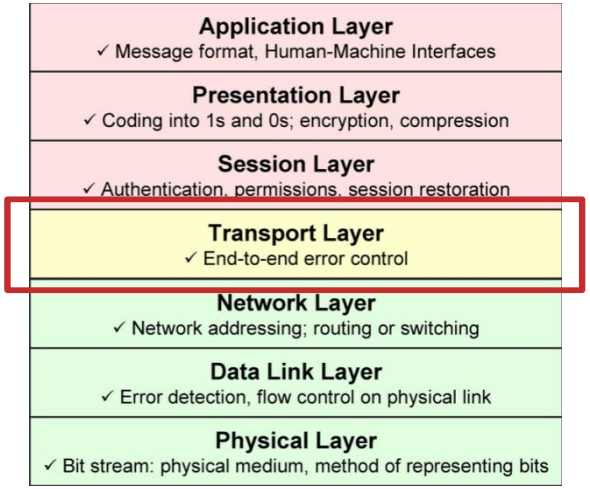
→ Fehler beheben: Nach **Sequenznummer** ordnen und ggf. doppelte Pakete verwerfen

TCP-Protokoll

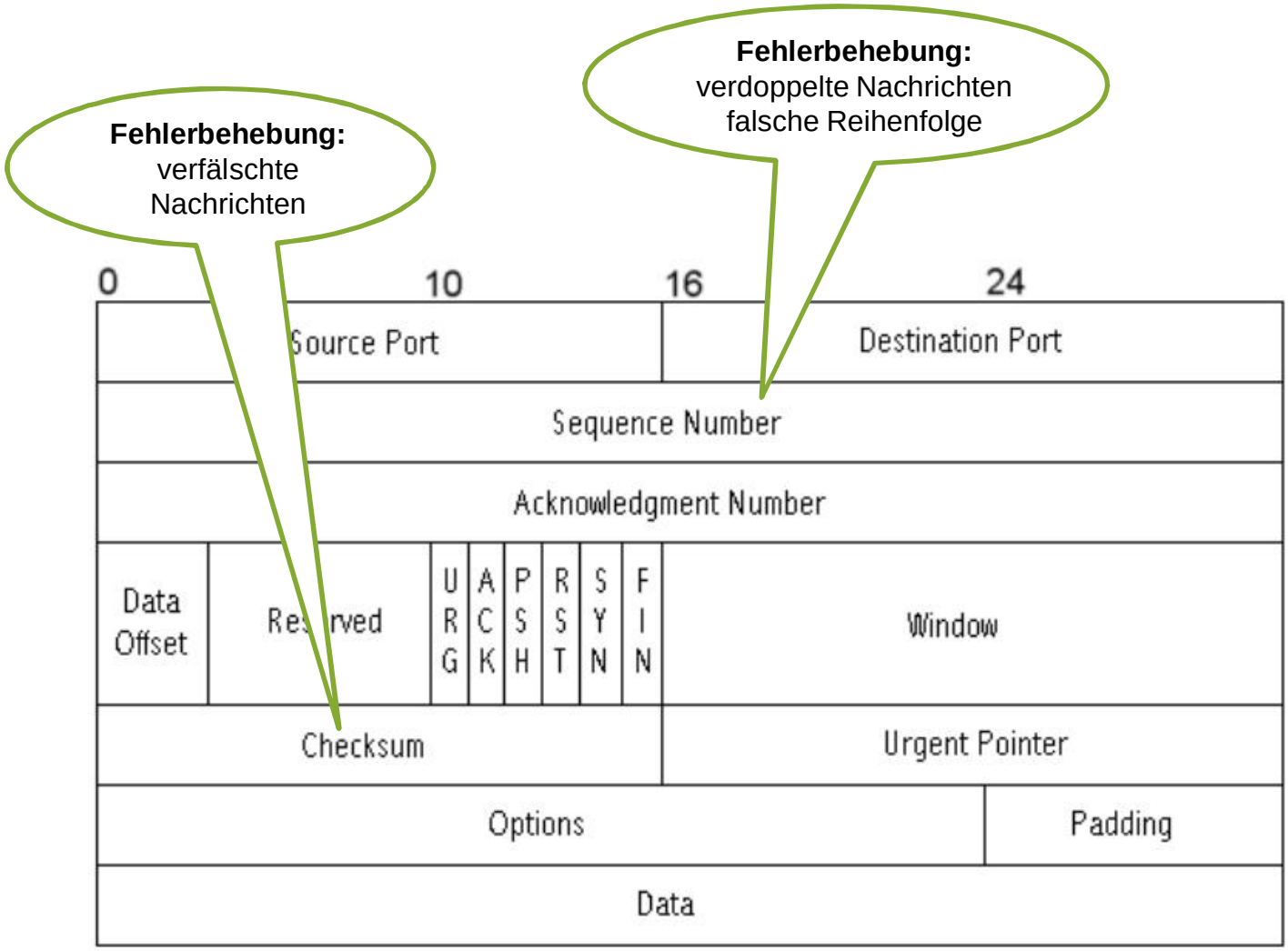
- Kommunikation bedarf der **Einhaltung von Regeln**, diese werden in **Protokollen** festgehalten.

- Ein Kommunikationsprotokoll ist für den Austausch von Daten zwischen Computern bzw. Prozessen zuständig, die in einem Rechnernetz miteinander verbunden sind.

- **TCP**-Protokoll (*Transmission Control Protocol*) gehört zu den wichtigsten Protokollen der Transportschicht.
- Die Aufgabe von TCP besteht in der Bereitstellung eines sicheren und zuverlässigen Ende-zu-Ende-Transports von Daten durch ein Netzwerk.



TCP-Protokoll – TCP Header



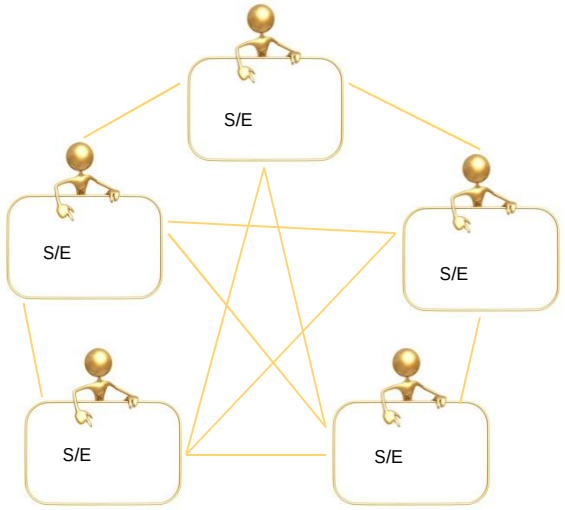
Fehlermodell (Klassifizierung von Fehlern)

→ Fehler beheben: **Link Quality Estimator - Modul**

Gruppenorganisation in VS

Flache Gruppe

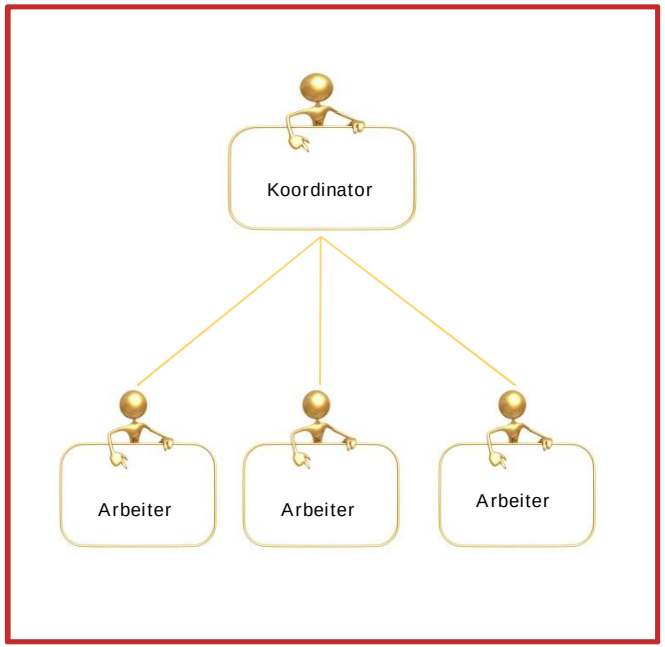
- → Kein Single-Point of Failure
- → Jede Entscheidung erfordert Abstimmung



Hierarchische Gruppe

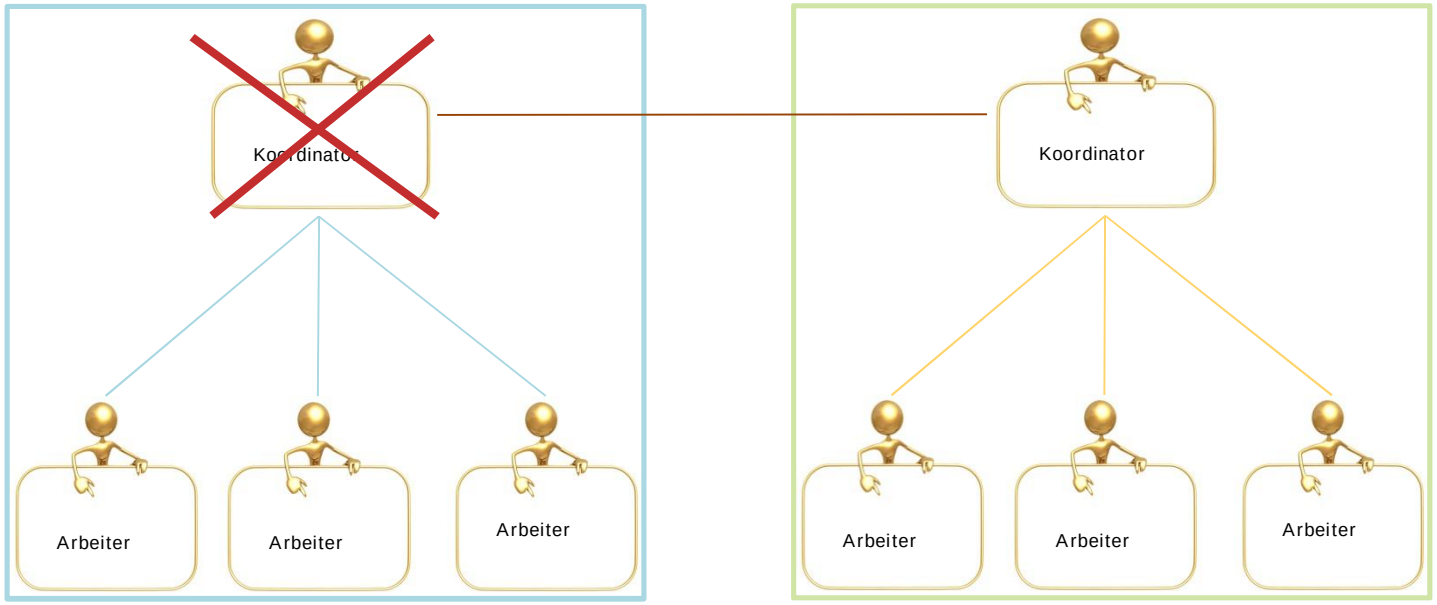
→Schneller im Normalbetrieb

→Koordinator-Ausfall



Fehlermodell (Klassifizierung von Fehlern)

Knotenausfall



→ Fehler beheben: **Commit-Protokoll**

Commit-Protokoll

- **Commit-Protokolle** regeln die Festschreibung (Commit) von Daten, die durch eine (verteilte) Transaktion beispielsweise in einem Datenbankmanagementsystem verändert werden sollen.

- Sie legen fest:
  - wie die an einer Transaktion teilnehmenden Prozesse (sog. „***agents***“) über einen Koordinator (sog. „**Leader**“) miteinander kommunizieren müssen?
  - wie Informationen protokolliert (geloggt) werden?
  - wie schließlich die betroffenen Daten festgeschrieben werden?

→ Dabei werden verschiedene **Fehlersituationen** durch das Protokoll **abgefangen**, wie z. B. ein Absturz des Koordinators während einer Phase.

Commit-Protokoll – 2-Phasen-Commit

1. Phase (PREPARE)

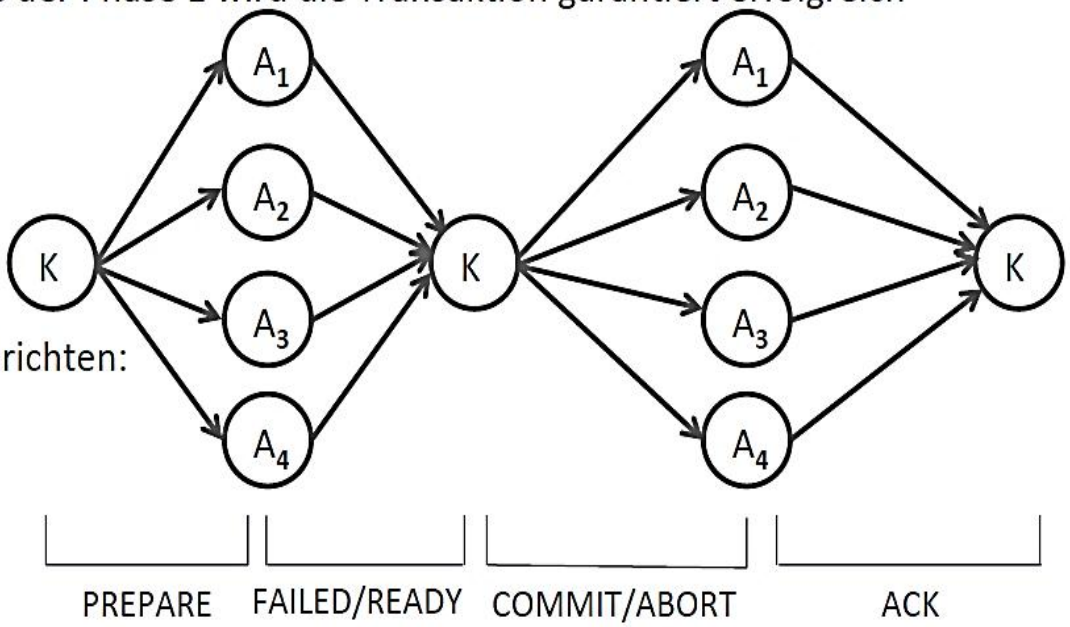
- Zusichern der Wiederholbarkeit der Transaktion
- Änderungen protokollieren und Commit-Satz schreiben

2. Phase (COMMIT)

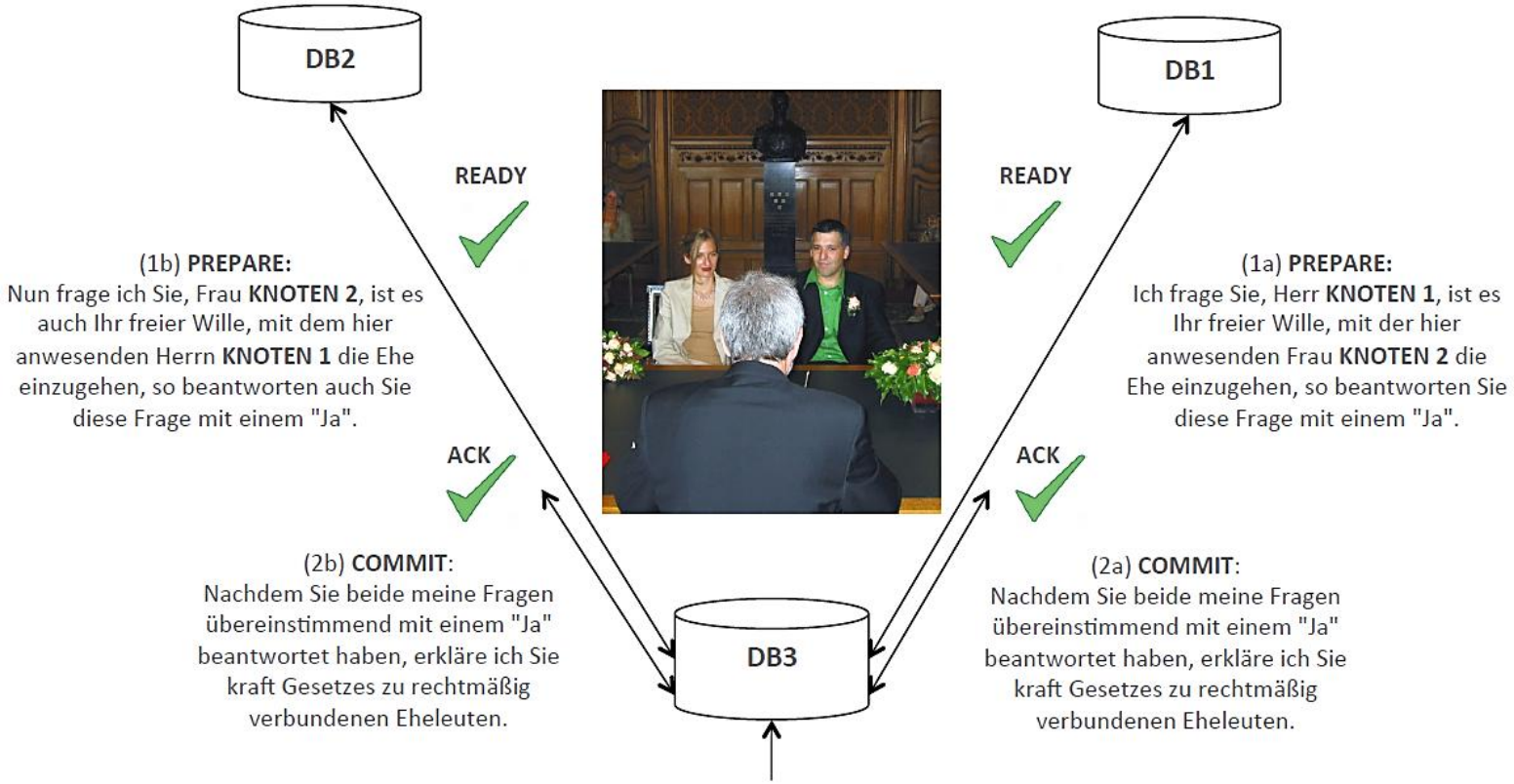
- Änderungen sichtbar machen (Sperren aufheben) ggf. hinfällige Protokollinformation (UNDO) löschen
- Merke: Nach Abschluss der Phase 1 wird die Transaktion garantiert erfolgreich abgeschlossen!

Beispiel

- n=5 Teilnehmer
- Aufwand
  Pro Teilnehmer 4 Nachrichten:
  4* (n-1) Nachrichten



2-Phasen-Commit- Beispiel



2-Phasen-Commit- Beispiel

*K schickt allen Agenten eine **PREPARE**-Nachricht, um herauszufinden, ob sie Transaktionen festschreiben können*

*Jeder Agent A_i empfängt **PREPARE**-Nachricht und schickt eine von zwei möglichen Nachrichten an K*

- **READY**, falls A_i in der Lage ist, die Transaktion T lokal festzuschreiben
- **FAILED**, falls A_i kein commit durchführen kann (wegen Fehler, Inkonsistenz etc.)

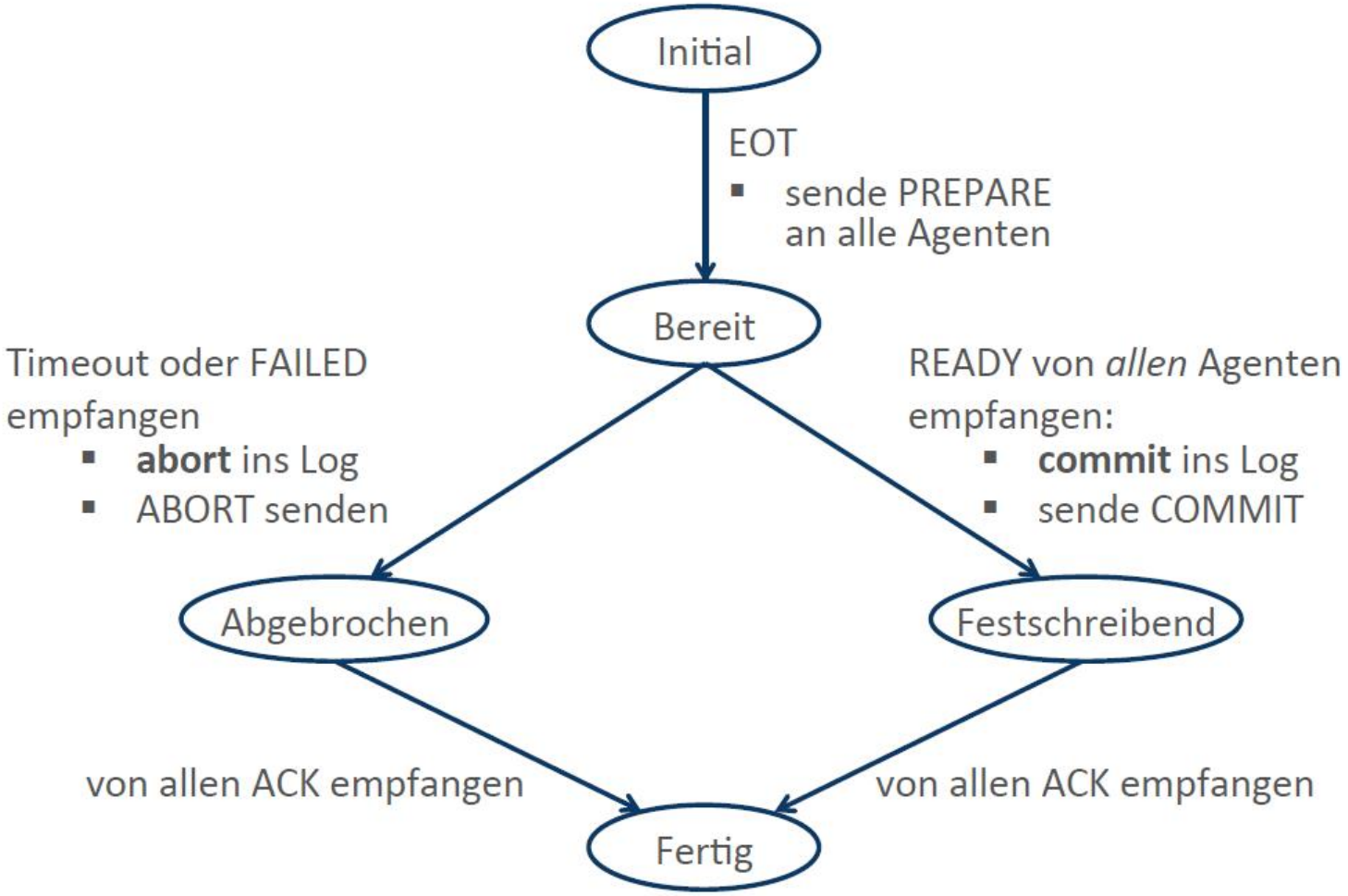
*Hat K von allen n Agenten $A_1,\ldots,A_n$ ein **READY** erhalten*

- Kann K ein **COMMIT** an alle Agenten schicken mit der Aufforderung, die Änderungen von T lokal festzuschreiben

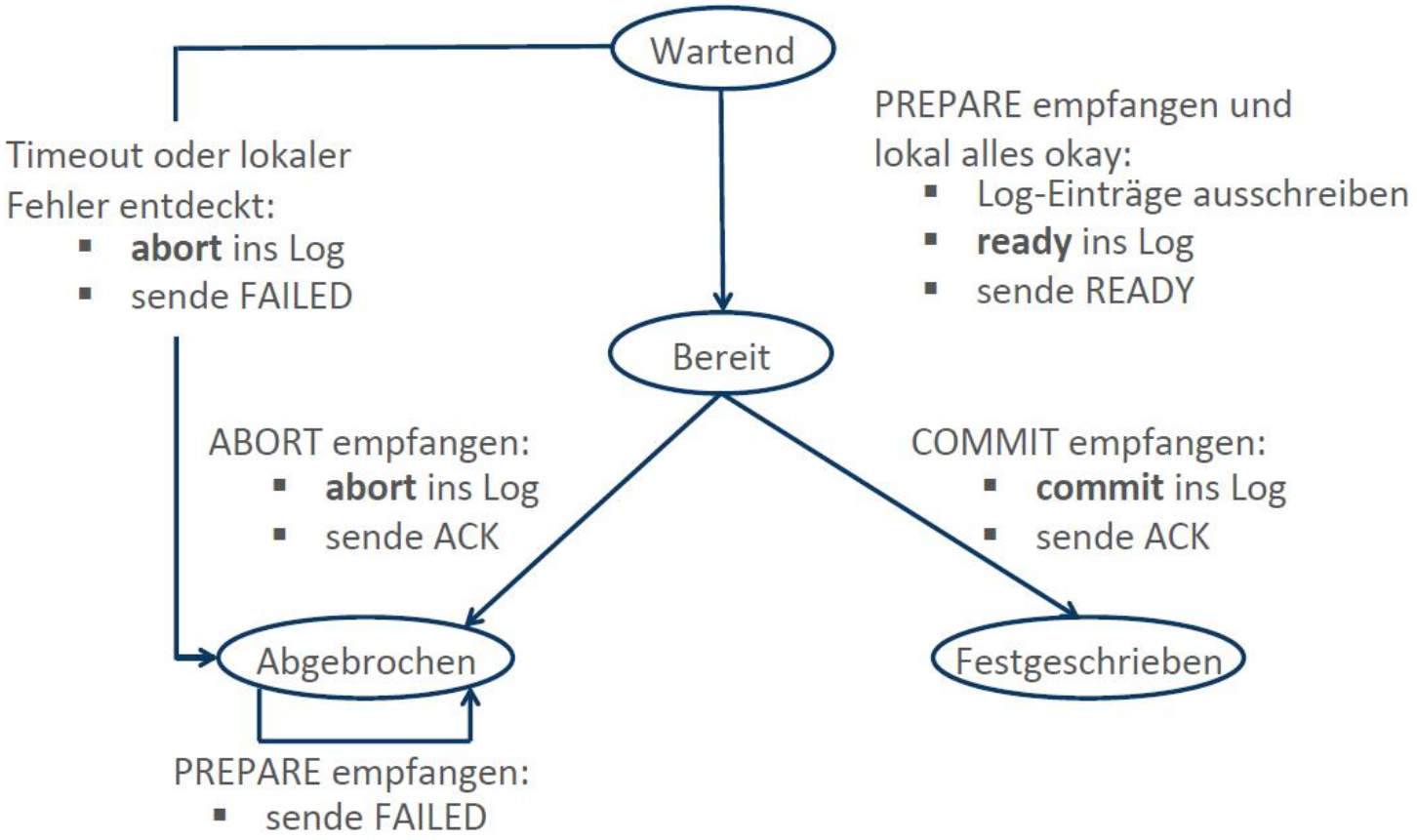
*Antwortet einer der Agenten mit **FAILED** oder gar nicht innerhalb einer bestimmten Zeit (**timeout**)*

- Schickt K ein **ABORT** an alle Agenten und diese machen die Änderungen der Transaktion rückgängig haben die Agenten ihre lokale EOT-Behandlung abgeschlossen, schicken sie eine **ACK**-Nachricht (=acknowledgement) an den Koordinator

2-Phasen-Commit- Zustandsübergang-Koordinator



2-Phasen-Commit- Zustandsübergang-Agenten



Warum 3-Phasen-Commit?

Absturz nachdem Knoten ein READY mitgeteilt haben

- Teilnehmer hat lokale Entscheidung dem Koordinator mitgeteilt und wartet auf die globale Entscheidung
- **Blockierung der Knoten**
- Hauptproblem des 2PC-Protokolls beim Absturz des Koordinators
- Verfügbarkeit der Knoten bzgl. anderer globaler und lokaler Transkationen wird eingeschränkt

Problem Blockierung

- Ein Teilnehmer der mit ready gestimmt hat und das globale Ergebnis noch nicht kennt, befindet sich in der Unsicherheitsphase
- Ressourcen werden von Teiltransaktionen in der Unsicherheitsphase ggf. unnötig lange gesperrt
- Lösung: Dreiphasen-Commit-Protokoll → Verhinderung der Blockierung

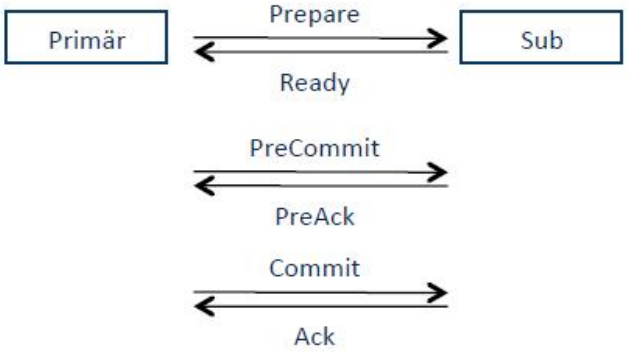
Commit-Protokoll - 3-Phasen-Commit

Motivation

- 2PC: ggf. lange Blockierung (Abhängigkeit vom Koordinator)
  - Ausfall des Koordinators bevor Teilnehmer Global-Commit / Global-Abort erhalten
  - Ausfall eines Teilnehmers während PREPARED kann zu langen Blockierungen führen
- Performance-Erhöhung durch „nicht-blockierende“ Commit-Protokoll (maximal k<n Rechner gleichzeitig ausfallen)

Konzept: ***Eliminierung der Unsicherheitsphase!***

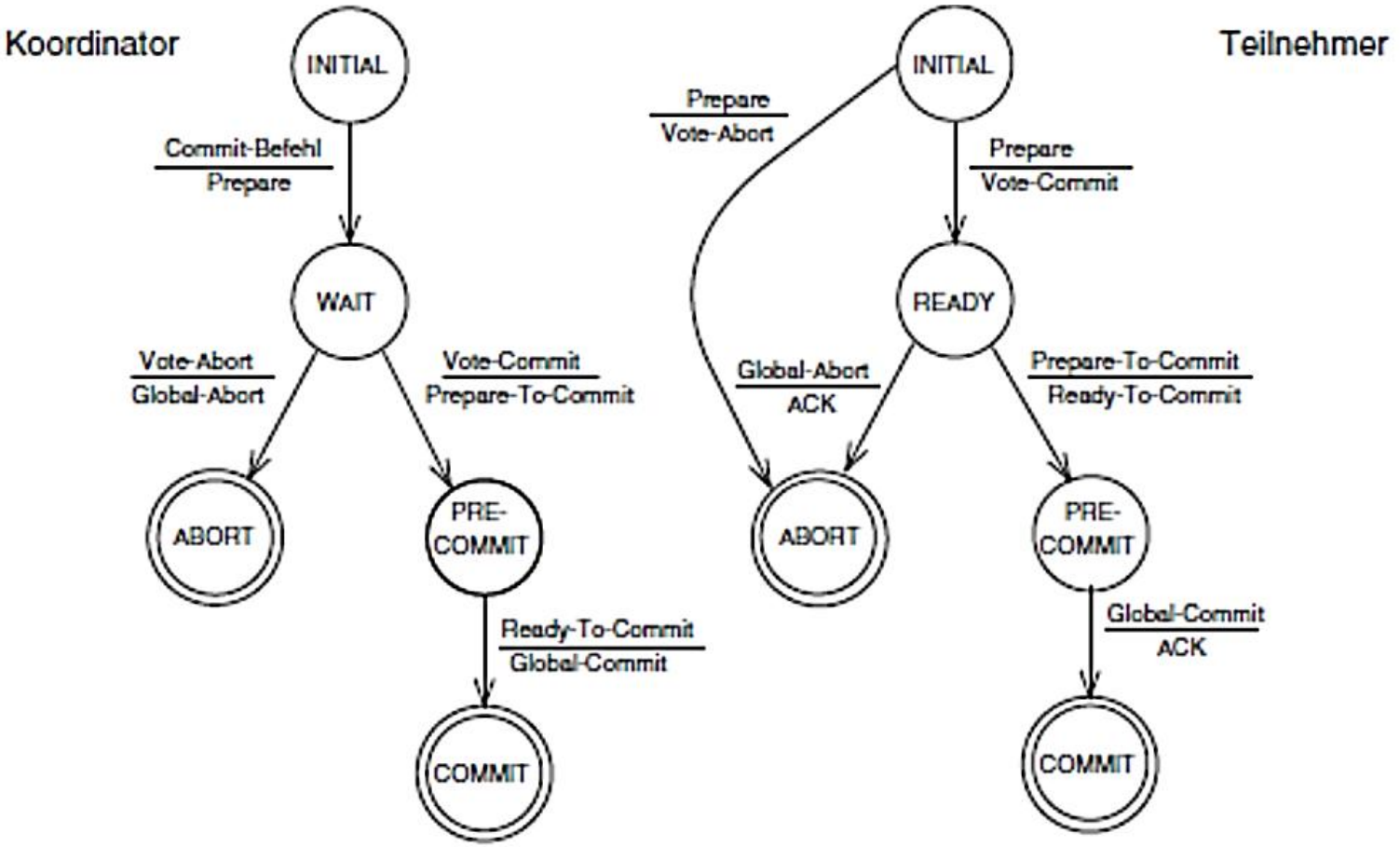
- 3PC-Protokoll: zusätzliche **PreCommit-Phase (1. Phase)**
- Teilnehmer, die PreCommit empfangen haben, wissen, dass nur Commit kommen kann
- Koordinator sendet erst Commit, wenn k Teilnehmer PreCommit mit PreAck bestätigt haben



Problem

- Signifikante Zunahme an Systemaufwand (6*(n-1) Nachrichten)

3-Phasen-Commit - Zustandsübergang



Agenda

- Verteilte Systeme
  - Definition
  - Nutzen
  - Wünschenswerte Eigenschaften
  - Beispiele
- Fehlertoleranz in verteilten Systemen
  - Verlässliche Systeme
  - Störungen in einem System
  - Fehlermodell
  - Protokolle
    - TCP
    - Commit-Protokoll
      - Beschreibung
      - 2-Phasen Commit
      - 3-Phasen Commit
  - Leader Election Service
    - Nutzungsmotivation
    - Quality of Service
    - Leader Election Module
      - Fehlerdetektor
      - Link Quality Estimator
      - Scheduler
    - Architektur
  - Leader Election Service -Evaluation

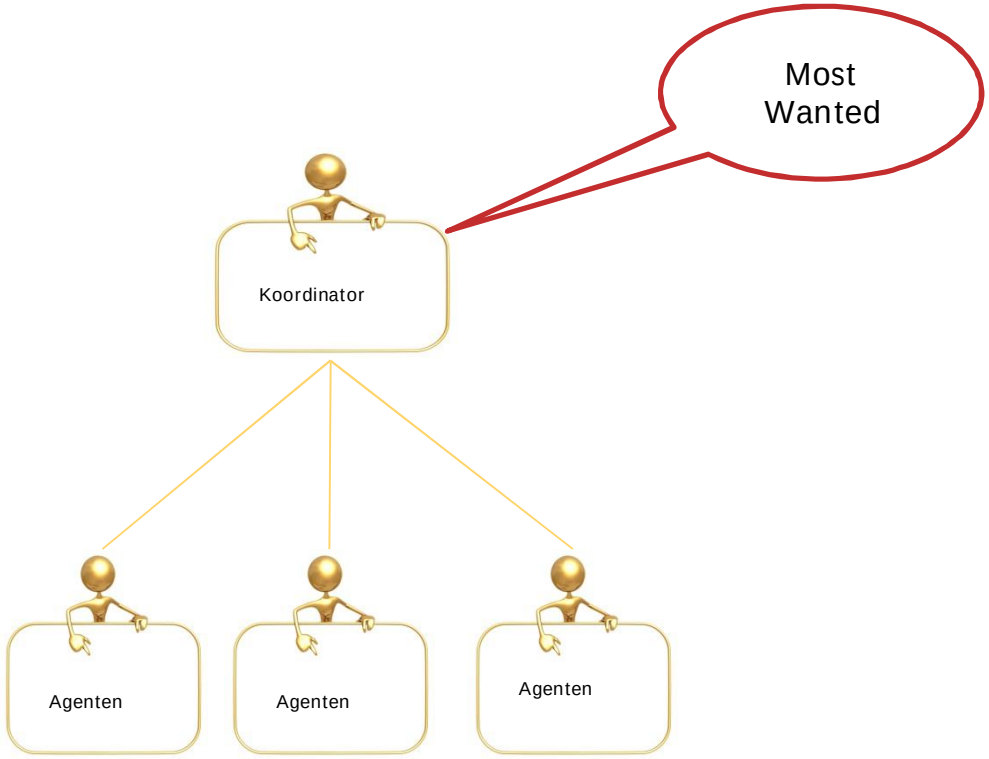
Leader Election Service

- Nutzungsmotivation
- Quality of Service
- Leader Election Module
  - Fehlerdetektor
  - Link Quality Estimator
  - Scheduler
- Architektur

Leader Election Service

Leader Election Service (Koordinator-Auswahlverfahren) ist ein Prozess zur Auszeichnung eines einzelnen Prozesses, der als Koordinator (Leader) für andere Prozesse, die in verteilten Systemen sind und eine gemeinsame Gruppe angehören, agiert.

→ Das bedeutet:



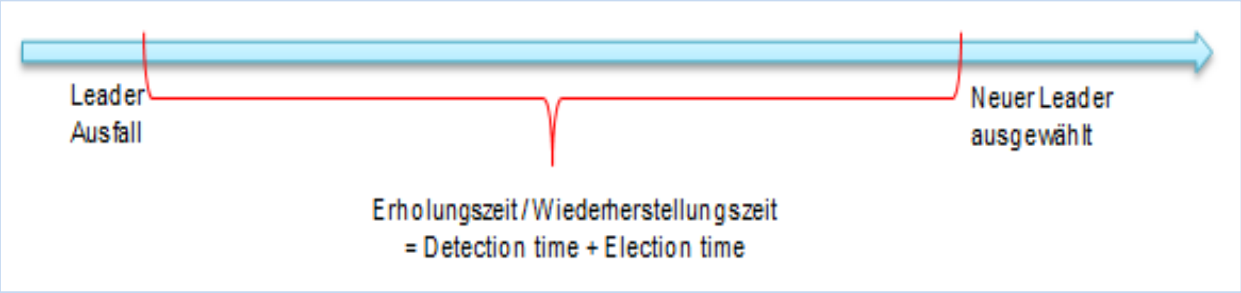
Leader Election Service – Quality of Service

- **Quality of Service (QoS):**

Eine Menge von Qualitätsanforderungen an das gemeinsame Verhalten bzw. Zusammenspiel von mehreren Prozessen.

In LE-Services relevante **QoS-Metriken**:

- **Geschwindigkeitsmetrik:** Es wird die Zeit gemessen, die ein Service braucht, um sich von einem Ausfall (Crash) des aktuellen Leaders zu **erholen**.



- **Durchschnittliche Fehlerrate:** Es ist der Anteil an Fehlern, die ein Service macht in dem es einen voll funktionsfähigen Leader zurückstuft (= Entlassen aus der Rolle als Gruppen-Leader) .

Leader Election Service–Module

- Im Kern jedes Leader Election Services existieren sog. **Fehlerdetektoren** (FD)-Modul. Sie helfen dabei:
  - aufzuspüren, ob ein aktueller Leader fehlgeschlagen ist.
  - festzustellen, welche von den Kandidaten, die den fehlgeschlagenen Leader ersetzen könnten, aktuell betriebs-/einsatzbereit sind.

- **Link Quality Estimator-Module**

In regelmäßigen Abständen wird die Qualität der Verbindung zwischen Prozess gemessen. Das Messen der „Qualität“ wird anhand von 3- Größen erreicht:

- **P** = Wahrscheinlichkeit, dass Nachrichten verloren gehen
- **E** = Erwartungswert der Nachrichtenverzögerung (geschätzte Verzögerungszeit unter den aktuellen Netzwerkkonditionen)
- **S** = Standardabweichung der Nachrichtenverzögerungen („normale“ Verzögerungszeit unter den gegebenen Netzwerkkonditionen)

- **Scheduler-Module**

Es plant die Versendung der *I’m alive*-Nachrichten, die von Prozessen versendet bzw. empfangen werden.

- Anhand der Häufigkeit (η) , Timeout (δ) und der Zeit in dem ein Prozess p seine letzte *I’m alive*-Meldung von ein Prozess q empfangen hat.

Leader Election Service –Architektur

- Für die Nutzung von Leader Election Service, muss ein Prozess p sich erst registrieren im Service und zwar mit einer **einmaligen und eindeutigen Prozess-Identifier**.

- Prozess p kann nun jederzeit jede Gruppe beitreten bzw. verlassen.

➢ Wie tritt ein Prozess p einer Gruppe bei? Prozess p muss die folgenden 4-Parameter angeben:

  ➢ Gruppe g's Identifier

  ➢ Ob Prozess p ein Kandidat für g's–Führung (Leadership) ist oder nicht

  ➢ Den Weg, dass sich Prozess p wünscht, um den aktuellen Leader von g zu finden.

    ➢ Durch ein „interrupt"/Unterbrechung von Service, falls sich der Leader von g geändert hat

    ➢ Durch „querying"/Fragen vom Service, ob Prozess p es so haben will.

Agenda

- Verteilte Systeme
  - Definition
  - Nutzen
  - Wünschenswerte Eigenschaften
  - Beispiele
- Fehlertoleranz in verteilten Systemen
  - Verlässliche Systeme
  - Störungen in einem System
  - Fehlermodell
  - Protokolle
    - TCP
    - Commit-Protokoll
      - Beschreibung
      - 2-Phasen Commit
      - 3-Phasen Commit
  - Leader Election Service
    - Nutzungsmotivation
    - Quality of Service
    - Leader Election Module
      - Fehlerdetektor
      - Link Quality Estimator
      - Scheduler
    - Architektur
  - Leader Election Service -Evaluation

- Leader Election Service –Evaluation
  - S1
  - S2
  - S3
  - Erkenntnisse aus der Evaluierung

Leader Election Service – Evaluation

- Experimentenreihe von *Nicolas Schiper* und *Sam Toueg. Evaluiert und verglichen wurden:*
  - 3 verschiedene LE-Services (S1, S2, S3)
  - CPU-Auslastung und Netzwerkbandbreitenkapazität
- Beschreibung der Systemparameter:

Systemparameter	Beschreibung
Netzwerk	5 Netzwerke, LAN, 12 Workstations
Workstation	- P4 3.2 GHz mit 512 MB RAM - Betriebssystem: Suse Linux 9.2
Anzahl der Applikationen	12 Prozesse, einer pro Workstation
Erholungszeit nach einem Leader-Ausfall	Jedes Prozess fällt alle 10 min. aus 5 Sek.
Kommunikations-verbindung	verlustbehaftete Verbindung: Kommunikationsverbindungen mit zufälligen Nachrichten-verluste bzw. -verzögerungen Ausfallanfällige Verbindung: Kommunikationsverbindungen, die Gegenstand von zufälligen Ausfällen & Wiederherstellung sind.

LE-Service – Die Evaluation von S1

- Der **Gruppen-Leader** ist der jeweilige Prozess **mit dem kleinsten Prozess-Identifier**
- Alle Prozesse **senden** in regelmäßigen Abständen (alle η Sekunden) sog. ***I'm alive*-Nachrichten** an alle Prozesse der Gruppe.
- **Timeouts δ**, damit festgestellt werden kann, dass ein Prozess fehlgeschlagen ist (sendet keine *I'm alive*-Nachrichten mehr).

Auswertung von:

- Durchschn. Wiederherstellungszeit eines Koordinators
- Durchschn. Fehlerrate

Auswertungsergebnisse:

➢ **Fehlerrate:** Service S1 ist **nicht stabil**, es macht über 6 Fehler*/Stunde.

** wir erinnern uns, ein Fehler tritt auf, wenn der Service einen Leader „unberechtigt" zurückstuft, obwohl dieser voll funktionsfähig ist. Im Falle von S1, wird diese Fehler getriggert, da ein Prozess der Gruppe beitritt, der einen kleineren Identifier als den des aktuellen Leader hat.*

➢ **Wiederherstellungszeit:** ca. 1 sec.

➢ **FAZIT**: Die Leader-Unstabilität aus S1 ist von Nachteil für die Robustheit eines Systems

LE-Service – Die Evaluation von S2

- Jeder Prozess p behält das letzte Mal im Auge in dem er verdächtigt wurde ausgefallen zu sein (sog. Accusation time (**Beschuldigungszeit**)) und

- **Prozess p wählt seinen Leader** unter eine Menge von Prozessen aus, die wie folgt aufgebaut sind (In 2-Phasen):
  - **Phase 1:** P wählt seinen **lokalen Leader** unter den Prozessen, von den Prozess p eine *I'm alive*-Nachricht empfangen hat)
  - **Phase 2:** P selektiert seinen **globalen Leader** aus der Menge aller lokalen Leader mit der frühsten Beschuldigungszeit

Auswertung von:
- Durchschn. Wiederherstellungszeit eines Koordinators
- Durchschn. Fehlerrate

Auswertungsergebnisse:

- **Fehlerrate:** Service S2 ist sehr **stabil**, denn in jedem der 5 Netzwerke wurde beobachtet, dass es **keine *"unberechtigten"* Zurückstufungen** eines Leader gab.

- **Wiederherstellungszeit:** Minimale Verzögerung aufgrund des o.g. Leader Ermittlungsmechanismus, denn noch beträgt die Leaderverfügbarkeit 99,82%.

- **FAZIT**: S2 ist stabiler als S1, allerdings ist die Anzahl der Nachrichten, die zwischen den Prozessen ausgetauscht werden enorm.

LE-Service – Die Evaluation von S3

- Service S3 basiert auf ein LE-Algorithmus, welches **kommunikations-effizient** ist.

- Nur der ausgewählte Leader einer Gruppe sendet *I'm alive*-Nachrichten an den anderen Prozessen einer Gruppe.

Kommunikationseffizienz wird erreicht, in dem die Menge aller kandidierenden Prozesse reduziert wird. Wie?

- Ein Prozess p berücksichtigt ein Prozess q als Kandidaten, nur wenn Prozess p direkt *I'm alive*-Nachrichten von Prozess q empfängt.

- Wenn Prozess p herausfindet, dass ein kandidierender Prozess q existiert, der eine kleinere Beschuldigungszeit hat, folglich besser als Leader-Kandidat geeignet wäre, tritt in diesem Fall Prozess p freiwillig als Kandidat zurück, in dem er keine *I'm alive*-Nachrichten mehr versendet.

LE-Service – Die Evaluation von S3- Fortsetzung

Auswertung von:

- Durchschn. Wiederherstellungszeit eines Koordinators
- Durchschn. Fehlerrate

Auswertungsergebnisse:

- **Fehlerrate:** S3 ist sehr **stabil**. Wie bereits unter S2 festgestellt, ist der Leader Election Algorithmus (kleinste **Beschuldigungszeit)** sehr robust, da ein funktionierender Leader nie zurückgestuft wird.

- **Wiederherstellungszeit:** Minimale Verzögerung aufgrund des o.g. Leader Ermittlungsmechanismus, denn noch beträgt die Leader-Verfügbarkeit 99,82%. (Wie in S2)

FAZIT: Bei dem Vergleich von S2 und S3 unter den gleichen Voraussetzung/Einstellungen, hat sich herausgestellt, dass S3 Kommunikations-effizienter als S2 ist.

Zusammenfassung

- *Ein **verteiltes System** ist eine Kollektion unabhängiger Computer, die den Benutzern als ein Einzelcomputer erscheinen*

- Wünschenswerte **Eigenschaften** in einem VS sind: Gemeinsame Ressourcennutzung, Nebenläufigkeit, Skalierbarkeit, Sicherheit, Transparenz, Offenheit, Fehlertoleranz

- Ein System gilt als **„verlässlich“** wenn die Anforderungen Verfügbarkeit, Zuverlässigkeit, Funktionssicherheit und Wartbarkeit erfüllt sind

- Das **Fehlermodell** definiert Fehler auf der
  - **Physikalischen Ebene** (Nachrichtenverlust, doppelte -und verfälschte Nachrichten), die vom **TCP**-Protokoll abgefangen wird
  - **Anwendungsebene** (Knotenausfall) wird durch das **Commit**-Protokoll abgefangen
  - **Anwendungsebene** (Nachricht kommt irgendwann an) abschwächen durch das Link Quality Estimator-Modul

- Leader Election Service (Koordinator-Auswahlverfahren) ist ein Prozess zur Auszeichnung eines einzelnen Prozesses, der als Koordinator

- **Quality of Service** (Qualitätsanforderungen an das gemeinsame Verhalten von mehreren Prozessen). Für Leader Election relevante Metriken: **Geschwindigkeitsmetrik, durchschnittliche Fehlerrate**

- Leader Election Service Module: Fehlerdetektor (Fehler und Ausfälle aufspüren), Link Quality Estimator (Schätzung von durchschnittliche Nachrichtenverzögerung) und Scheduler (plant die Versendung der *I'm alive* Nachrichten der Prozesse)

- Leader Election Service Architektur (Wie ein Prozess eine Gruppe beitritt? Wie bemerken Prozesse Leader-Ausfälle?)

- Leader Election Services können verglichen und evaluiert werden

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Vielen Dank für die Aufmerksamkeit

Erkenntnisse aus der Auswertung S1, S2 und S3

- **CPU-Auslastung:**

S2 und S3 sind unmaßgeblich für die CPU-Last. Der schlechteste Wert betrug 0,04% CPU-Auslastung.

- **Netzwerkbandbreite:**

Die schlechteste Messung betrug 6,48KB/Sekunde Nachrichtenverkehr pro Workstation.

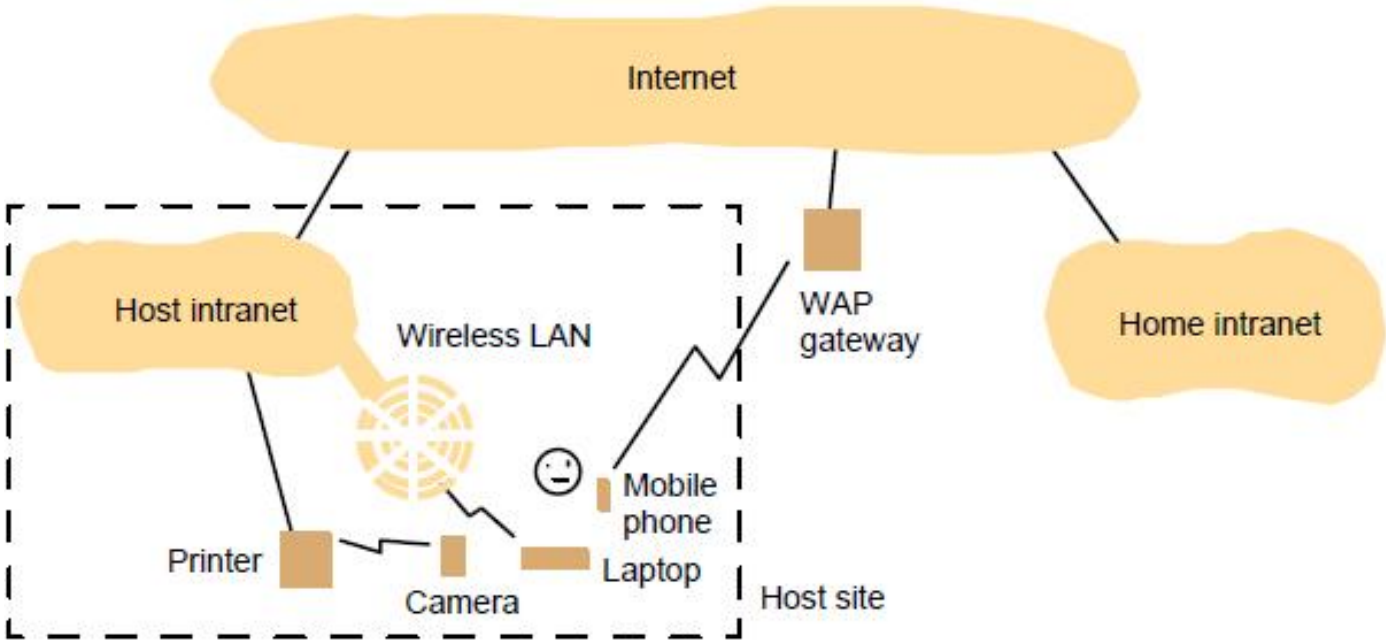
- **Leader Election:**

Leader-Verfügbarkeit ist in S2 und S3 sehr hoch.

- **Wiederherstellungszeit:**

Sehr kurze Wiederherstellungszeit in S1, S2 und S3.

Beispiel 3: Mobile Computing



Quelle: Prof. Thai