# Verteilte Systeme

Oktober - November 2023

8. Vorlesung – 16.11.2023


Kurs: TINF21AI1

Dozent: Tobias Schmitt, M.Eng.

Kontakt: d228143@
student.dhbw-mannheim.de

# Wiederholungsfragen

- Was verstehen Sie unter serverinitiierte und clientinitiierte Replikation?

- Nennen Sie ein Beispiel für die Anwendung von serverinitiierte Replikation.
- Nennen Sie ein Beispiel für die Anwendung von clientinitiierte Replikation.

- Was verstehen Sie unter Leases? Wofür werden diese verwendet?



- An welche Konsistenzprotokolle können Sie sich erinnern?

# Themenüberblick

- **Fehlertoleranz**
  - **Grundbegriffe**
  - Prozess-Resilienz
  - Client-Server-Kommunikation
  - Gruppenkommunikation

# Fehlertoleranz - Einstiegsfragen

1) Was verstehen Sie unter Fehlertoleranz?

2) Welche anderen Umgangsformen hinsichtlich Fehlern sind noch denkbar?

3) Was ist der Unterschied zwischen einer Störung (fault), einem Fehler (error) und einem Ausfall (failure)?

4) Welche Arten von Ausfällen kann man unterscheiden?

5) Welche Grundansätze hinsichtlich der Fehlertoleranz in verteilten Systemen sind denkbar?

# Fehlertoleranz - Grundbegriffe

- Fehlertoleranz
  - Tolerieren von Ausfallfehlern
  - Starke Verwandtschaft mit Verlässlichkeit
- Anforderungen an ein verlässliches System
  - Verfügbarkeit (Availablilty)
    - Wahrscheinlichkeit, dass das System zu einem bestimmten Zeitpunkt korrekt arbeitet (vgl. hochverfügbare Systeme)

# Fehlertoleranz - Grundbegriffe

- Anforderungen an ein verlässliches System
  - Zuverlässigkeit (Reliability)
    - Aussage über fortlaufende Ausfallfreiheit bezogen auf Zeitintervall
    - z.B. Ausfall jede Stunde für 1ms (hochverfügbar, aber unzuverlässig)
    - z.B. keine Abstürze, aber 1 Monat Wartung (nicht hochverfügbar, aber zuverlässig)

# Fehlertoleranz - Grundbegriffe

- Anforderungen an ein verlässliches System
  - Funktionssicherheit (Safety)
    - Aussage über Auswirkungen im Fehlerfall (Fehler sollen nicht zur Katastrophe führen.)
  - Wartbarkeit (Maintainability)
    - Aussage, wie leicht ein ausgefallenes System repariert werden kann
    - Aussage, über Aufwand bei Änderungen / Aktualisierungen am System

# Fehlertoleranz - Grundbegriffe

- Umgangsformen hinsichtlich Fehlern
  - Fehlervermeidung (fault prevention)
    - Verhinderung des Auftretens von Fehlern
    - z.B. Schulung der Programmierer, Testen, etc.
  - Fehlertoleranz (fault tolerance)
    - Komponenten entwerfen, um das Auftreten von Fehlern zu verbergen
  - Fehlerbehebung (fault removal)
    - Reduzierung der Fehler hinsichtlich der Existenz, der Anzahl, des Schweregrades
  - Fehlervorhersage (fault forecasting)
    - Abschätzung der aktuellen Existenz, zukünftiger Vorfälle und hinsichtlich der Fehlerkonsequenzen

# Fehlertoleranz - Grundbegriffe

- Umgangsformen hinsichtlich Fehlern
  - Fehlervermeidung (fault prevention)
    - Verhinderung des Auftretens von Fehlern
    - z.B. Schulung der Programmierer, Testen, etc.
  - Fehlertoleranz (fault tolerance)
    - Komponenten entwerfen, um das Auftreten von Fehlern zu verbergen

# Fehlertoleranz - Grundbegriffe

- Umgangsformen hinsichtlich Fehlern
  - Fehlerbehebung (fault removal)
    - Reduzierung der Fehler hinsichtlich der Existenz, der Anzahl, des Schweregrades
  - Fehlervorhersage (fault forecasting)
    - Abschätzung der aktuellen Existenz, zukünftiger Vorfälle und hinsichtlich der Fehlerkonsequenzen
    - z.B. bei Kenntnis von Fehlerquellen → Abschätzung hinsichtlich des wirtschaftlichen Schadens beim Auftreten

# Fehlertoleranz - Grundbegriffe

- Systemausfall (Failure)
  - Zusagen können nicht eingehalten werden
- Fehler (Error)
  - Teil des Systemzustandes, der zum Ausfall führen kann
- Störung (Faul

# Fehlertoleranz - Grundbegriffe

- Systemausfall (Failure)
  - Zusagen können nicht eingehalten werden
- Fehler (Error)
  - Teil des Systemzustandes, der zum Ausfall führen kann

# Fehlertoleranz - Grundbegriffe

- Störung (Fault)
  - Ursache eines Ausfalls
  - Herangehensweise an Störungen
    - Verhindern, Beheben, Vorhersagen von Fehlern
  - Fehlertoleranz: Trotz vorliegen bestimmter Störungen kann ein System seine Dienste bereitstellen.

# Fehlertoleranz - Grundbegriffe

- Störung (Fault)
  - Unterscheidung:
    - Vorübergehende Störungen (Transient Faults) (z.B. Vogel durch Strahl eines Mikrowellensender)
    - Wiederkehrende Störungen (z.B. Wackelkontakt, …)
    - (siehe https://www.pcwelt.de/news/Wegen-altem-TV-Internet-Ausfaelle-im-ganzen-Ort-10887532.html)
    - Permanente Störungen (z.B. Soft- oder Hardwarefehler)

# Fehlertoleranz - Fehlermodelle

- Was verstehen Sie unter den folgenden Ausfallarten?
  - Absturzausfall
  - Dienstausfall
  - Zeitbedingter Ausfall
  - Ausfall der korrekten Antwort
  - Byzantinischer oder zufälliger Ausfall

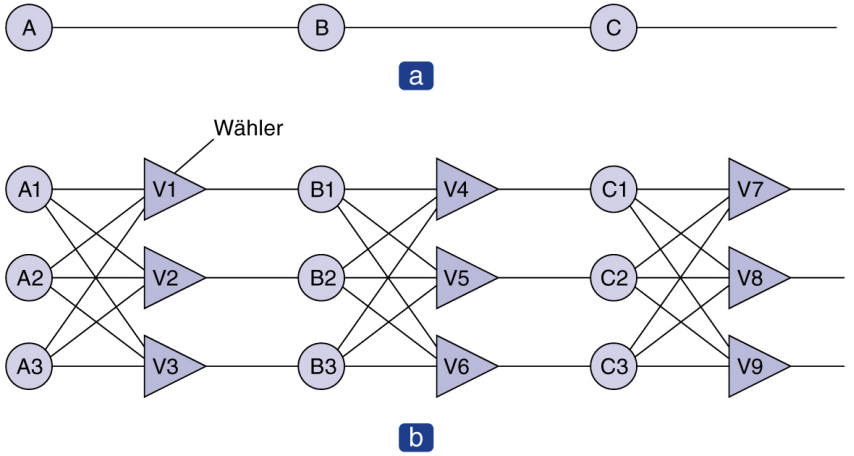
# Fehlertoleranz - Fehlermodelle

| Ausfallart | Beschreibung |
|---|---|
| **Absturzausfall** (Crash Failure) | Ein Server steht, hat aber bis dahin richtig gearbeitet. Der angebotene Dienst bleibt beständig aus (ständiger Dienstausfall). |
| **Dienstausfall** (Omission Failure)<br>*Empfangsauslassung*<br>*Sendeauslassung* | Ein Server antwortet nicht auf eingehende Anforderungen.<br>Ein Server erhält keine eingehenden Anforderungen.<br>Ein Server sendet keine Nachrichten. |
| **Zeitbedingter Ausfall** (Timing Failure) | Die Antwortzeit eines Servers liegt außerhalb des festgelegten Zeitintervalls. |
| **Ausfall korrekter Antwort** (Response Failure)<br>Ausfall durch *Wertfehler (Value Failure)*<br>Ausfall durch *Zustandsübergangsfehler (State Transition Failure)* | Die Antwort eines Servers ist falsch. Dieser Ausfall wird oft auch kurz Antwortfehler genannt.<br>Der Wert der Antwort ist falsch.<br><br>Der Server weicht vom richtigen Programmablauf ab. |
| **Byzantinischer oder zufälliger Ausfall** (Arbitrary oder Byzantine Failure) | Ein Server erstellt zufällige Antworten zu zufälligen Zeiten. |

Fail-stop Failure (Ausfall-Stopp) – Dienst hört einfach auf
Fail-silent Failure (Ausfall durch Verschweigen) – andere Prozesse vermuten Absturz
...

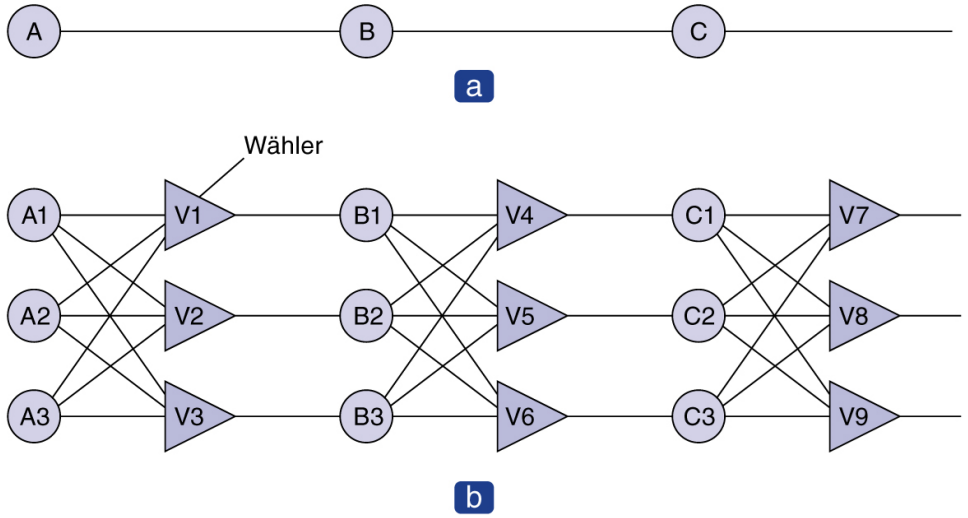
# Fehlertoleranz

- Maskierung des Ausfalls durch Redundanz
  - Informationsredundanz
    - Zusätzliche Bits zwecks Wiederherstellung
    - z.B. Hamming-Code (siehe https://www.youtube.com/watch?v=X8jsijhllIA)
  - Zeitliche Redundanz
    - Aktion wird ggf. wiederholt
    - z.B. bei verübergehende oder wiederkehrende Störungen



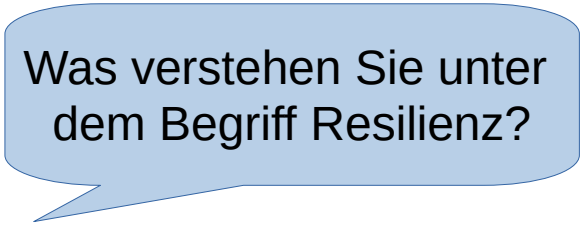
# Fehlertoleranz

- Maskierung des Ausfalls durch Redundanz
  - Technische Redundanz
    - Zusätzliche Ausrüstung oder Prozesse zwecks Kompensation von ausgefallener oder fehlerhafter Komponenten
    - Realisierung via Hardware oder Software
    - Beispiel: Dreifache modulare Redundanz



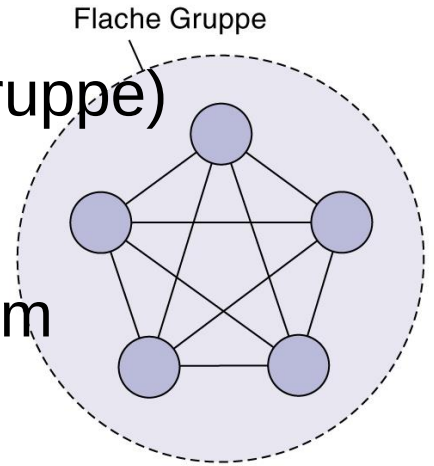
# Themenüberblick

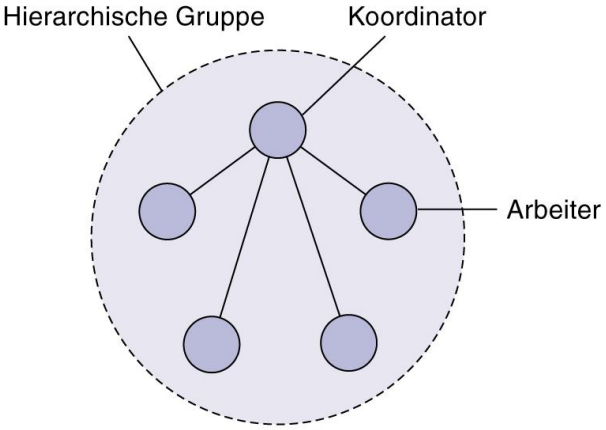
- **Fehlertoleranz**
  - Grundbegriffe
  - **Prozess-Resilienz**
  - Client-Server-Kommunikation
  - Gruppenkommunikation



# Prozess-Resilienz

- Lösungsansatz: Replizierung von Prozessen
  - Gruppierung von Prozessen (fehlertolerante Gruppe)
- Lineare Gruppe
  - Kein Chef und Entscheidungen stets gemeinsam
  - Vorteil: keinen einzelnen Ausfallpunkt
  - Nachteil: Entscheidungsfindung braucht Zeit

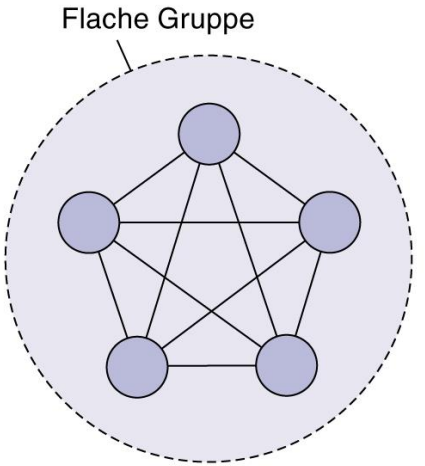


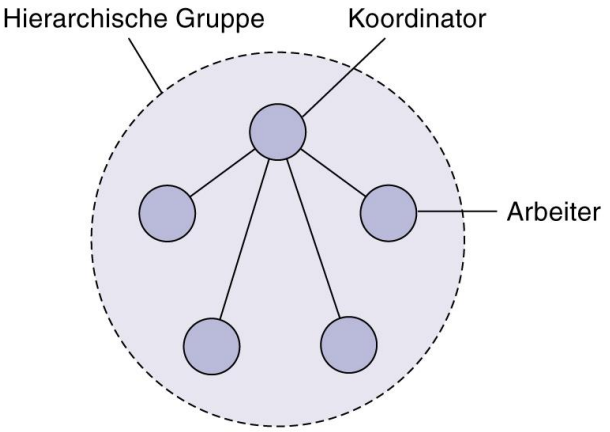




*Resilienz: psychische Widerstandskraft; Fähigkeit,
schwierige Lebenssituationen ohne anhaltende Beeinträchtigung zu überstehen

# Prozess-Resilienz

- Hierarchische Gruppe
  - Existenz eines Koordinators
  - z.B. Anforderung wird an besten Arbeiter weitergeleitet
  - Vorteil: schneller als lineare Gruppe
  - Nachteil: einzelner Ausfallpunkt (Koordinator)




*Resilienz: psychische Widerstandskraft; Fähigkeit, schwierige Lebenssituationen ohne anhaltende Beeinträchtigung zu überstehen

# Prozessgruppen
# Gruppenverwaltung

- Gruppenserver
  - Verwaltung der Gruppen und der Mitgliedschaften
  - Vorteil: leicht zu implementieren
  - Nachteil: einzelner Ausfallpunkt
- Alternative: Verteilte Verwaltung
  - Bedingt Existenz (zuverlässigen) Multicastings
- funktioniert)

# Prozessgruppen
# Gruppenverwaltung

- Betrachtungsaspekte:
  - Eintritt und Austritt einzelner Prozesse
    - Austritt mit oder ohne Ankündigung
    - Senden und Empfangen von Nachrichten synchron zum Ein- und Austritt
  - Kritische Größe einer Gruppe (Absturz mehrere Computer, so dass Gruppe nicht mehr funktioniert)

# Prozessgruppen Designfragen

- Replikation
  - Urbildbasierte Protokolle
    - Hierarchische Strukturierung, Urbild koordiniert Schreibvorgänge
    - Absturz des Urbildes: Wahl unter den Backups
  - Protokolle für replizierte Schreibvorgänge oder quorumbasierte Schreibvorgänge
    - Anwendung in linearen Gruppen

# Prozessgruppen Designfragen

- Anzahl an Replikationen
  - Bezeichnung: k-fehlertolerant
    - Kein Ausfall trotz Fehler in k Komponenten
  - k+1 Replikationen für k-Fehlertoleranz
    - Bei Absturz-, Dienst- und zeitlichem Ausfall
  - Mindestens 2k+1 Replikationen für k-Fehlertoleranz
    - Bei byzantinischen Ausfall oder Ausfall korrekter Antworten
- Bedingung: Anforderungen auf allen Servern in derselben Reihenfolge
  - Realisierung durch atomares Multicasting

# Interludium
# Problem der byzantinischen Generäle

- Referenz auf byzantinisches Reich (330-1453)
  - „… einem Ort (Balkan und die heutige Türkei), wo endlose Verschwörungen, Intrigen und Lügen in Herrscherkreisen als üblich galten.“
    (Tanenbaum und van Steen, Verteilte Systeme, 2. Auflage, S. 359)

- Problemstellung
  - Mehrere Divisionen geografisch verstreut (mit je einem General) belagern feindliches Lager
  - Übereinstimmung zwecks Angriff – Kommunikation zwischen Generälen via Boten
  - Übereinstimmung wichtig, da Angriff einiger Divisionen zur Niederlage führt
  - Verhinderung einer Übereinstimmung durch
    - Boten können vom Feind gefangen genommen werden (unzuverlässige Kommunikation) … aber in dem Fall ist das Problem ist nicht lösbar.
    - Unter den Generälen können Verräter sein (einstreuen irreführender Informationen)



(Byzantische Fehlertoleranz in Blockchains: https://www.youtube.com/watch?v=VWG9xcwjxUg)

# Einigungsalgorithmen

- Beispiel für Relevanz von Einigung
  - Auswahl eines Koordinators
  - Entscheidung, ob eine Transaktion mit Commit festgeschrieben wird
  - Aufteilung von Aufgaben
  - ...
- Annahme: Prozess arbeiten nicht zusammen, um ein falsches Ergebnis zu produzieren.

- Ziel verteilter Einigungsalgorithmen:
  - Alle nicht fehlerbehafteten Prozesse sind sich einig über einen Aspekt
  - Erreichen der Einigkeit in endlicher Anzahl von Schritten

# Einigungsalgorithmen

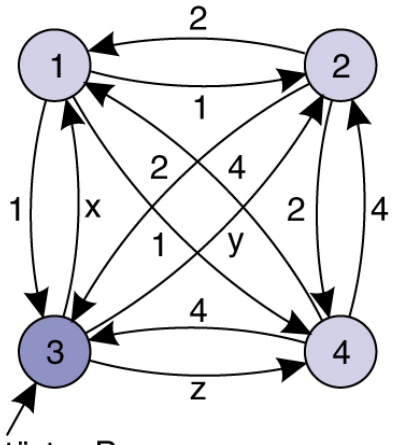
- Gruppenarbeit:
  - Überlegen Sie sich einen möglichen Einigungsalgorithmus!
  - Machen Sie sich Gedanken zur Fehlererkennung.
    Wie könnten Sie dies in verteilten Systemen bewerkstelligen?

# Einigungsalgorithmen

- Lösung für das byzantinische Übereinstimmungsproblem
- Annahme: 4 Prozesse, darunter 1 fehlerhafter Prozess

- Schritt 1: Senden der Infos an alle anderen (a)
- Schritt 2: Sammeln aller erhaltenen Infos (b)
- Schritt 3: Ergebnisvektor an alle anderen (c)
- Schritt 4: Vergleich der erhaltenen Ergebnisvektoren

*Ergebnis: (1,2,unbekannt,4)*




a

1 Got(1, 2, x, 4)
2 Got(1, 2, y, 4)
3 Got(1, 2, 3, 4)
4 Got(1, 2, z, 4)

b

| 1 Got | 2 Got | 4 Got |
|---|---|---|
| (1, 2, y, 4) | (1, 2, x, 4) | (1, 2, x, 4) |
| (a, b, c, d) | (e, f, g, h) | (1, 2, y, 4) |
| (1, 2, z, 4) | (1, 2, z, 4) | (i, j, k, l) |

c

# Erkennung von Ausfällen (Failure Detection)

- Ansatz 1:
  - Zustandsanfragen an andere Prozesse („Lebst Du noch?“)
  - Abwarten einer Rückantwort
  - Zeitüberschreitungen indiziert Absturz eines Prozesses
  - Problem: falsche Positivmeldungen möglich
- Ansatz 2:
  - Regelmäßiges Senden der Dienstverfügbarkeit an Nachbarn (Senden eines „Heartbeats“)

- Achtung: Unterscheidung zwischen Netzwerkausfällen und Knotenausfällen
  - Entscheidung über Ausfall nicht alleinig durch einzelnen Prozess

# Erkennung von Ausfällen (Failure Detection)

- Szenario:
  - Rechner C erhält keine Nachricht eines anderen Rechners C*
  - Problem: Ist C* ausgefallen?
  - Aussage hinsichtlich: Absturzausfall, Dienstausfall oder zeitbedingter Ausfall
- Unterscheidung
  - Asynchrone Systeme
    - Keine Aussage über Ausführungs- oder Auslieferungsgeschwindigkeiten
      → Detektion von Absturzausfällen nicht möglich
  - Synchrone Systeme
    - Ausführungs- oder Auslieferungsgeschwindigkeiten innerhalb vorgeschriebener Grenzen
      → Zuverlässige Detektion von Dienstausfall oder zeitbedingter Ausfall
  - Teilweise synchrone Systeme (partially synchronous systems)
    - Annahme über System: weitestgehend synchrones System auch wenn keine Zeitgrenzenvorgaben
      → Normalerweise - Detektion von Absturzausfällen möglich

# Themenüberblick

- **Fehlertoleranz**
  - Grundbegriffe
  - Prozess-Resilienz
  - **Client-Server-Kommunikation**
  - Gruppenkommunikation

# Client-Server-Kommunikation

- Fokus: Kommunikationsfehler
  - Absturz-, Auslassungs-, Timing- und zufällige Fehler
  - Beispiel für zufällige Fehler: Doppeltes Senden einer Nachricht

- Punkt-zu-Punkt-Kommunikation
  - Verwendung eines zuverlässigen Transportprotokolls (z.B. TCP)
  - TCP maskiert Auslassungsfehler
  - Keine Maskierung von Absturzfehlern

# Client-Server-Kommunikation

- RPC-Semantik bei Fehlern
  - Erinnerung: RPC (Remote Procedure Call)
    - Verbergen der Kommunikation
    - Entfernte Prozeduraufrufe genau wie lokale Prozeduraufrufe
  - Fehlerklassen

    1) Client kann Server nicht finden

    2) Anforderungsnachricht vom Client an Server geht verloren

    3) Serverabsturz nach Erhalt der Anforderung

    4) Antwortnachricht vom Server an Client geht verloren

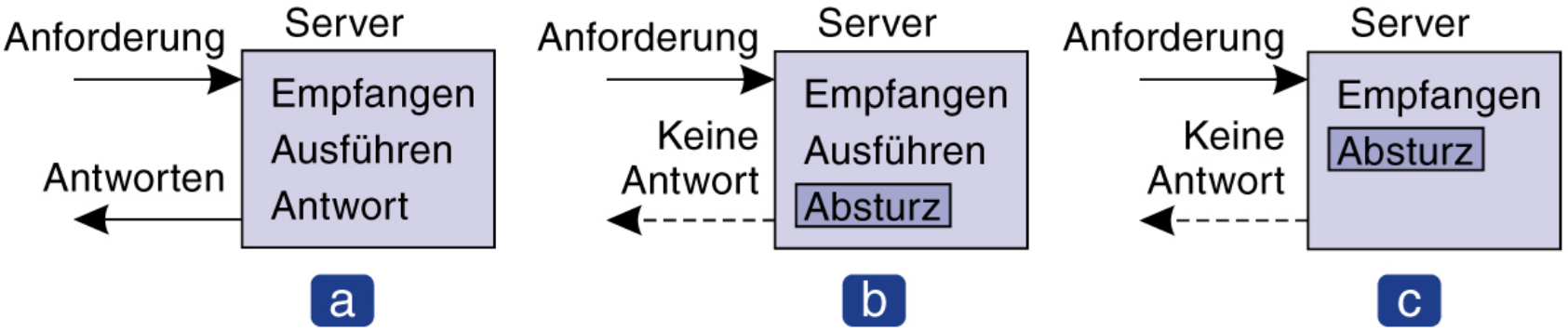
    5) Clientabsturz nach dem Senden der Anforderungsnachricht

# Client-Server-Kommunikation
## RPC-Semantik bei Fehlern

- Client kann Server nicht finden
  - Ursache: Server läuft nicht oder veraltete Version des Client-Stubs
  - Umgang durch Nutzung von Ausnahmen oder Signale
  - Maskierung schwer realisierbar
  - Resultat: „Kann den Server nicht finden.“
- Anforderungsnachricht vom Client an Server geht verloren
  - Mit Senden → Start eines Timers
  - Ablauf des Timers → Erneutes Versenden
  - Ggf. Resultat: „Kann den Server nicht finden.“

# Client-Server-Kommunikation
## RPC-Semantik bei Fehlern

- Serverabsturz nach Erhalt der Anforderung
  - Mögliche Abläufe
    - (a) der Normalfall
    - (b) Absturz nach der Ausführung
    - (c) Absturz vor der Ausführung
  - Achtung: Behandlungsfall bei (b) und (c) unterschiedlich



# Client-Server-Kommunikation
## RPC-Semantik bei Fehlern

- Serverabsturz nach Erhalt der Anforderung – Teil 2
  - Clientansatz 1: Mindestens-einmal-Semantik
    - Warte bis Server neu gebootet ist oder wähle anderen Server und versuche es erneut
  - Clientansatz 2: Höchstens-einmal-Semantik
    - Aufgabe und Fehlerrückgabe
  - Clientansatz 3: Keine Garantien → Versuche es 0 bis n-Mal
  - Wunsch: Genau-einmal-Semantik (aber nicht realisierbar)

# Client-Server-Kommunikation
## RPC-Semantik bei Fehlern

- Serverabsturz nach Erhalt der Anforderung – Teil 2
  - Ansatz über Unterstützung des Servers
    - Server sendet Fertigstellungsnachricht vor oder nach der Bearbeitung
    - Client hat verschiedene Wiederholungsstrategien
      - niemals neue Anforderung
      - immer neue Anforderung
      - neue Anforderung, wenn kein Erhalt einer Fertigungsstellungsnachricht
      - neue Anforderung, wenn ein Erhalt einer Fertigstellungsnachricht

# Client-Server-Kommunikation
## RPC-Semantik bei Fehlern

- Serverabsturz nach Erhalt der Anforderung – Teil 3
  - Mögliche Ergebnisse:

| Client | Server | | | | | |
|---|---|---|---|---|---|---|
| | Strategie M → PS | | | Strategie P → M | | |
| Wiederholungsstrategie | MPC | MC(P) | C(MP) | PMC | PC(M) | C(PM) |
| Immer | DUP | OK | OK | DUP | DUP | OK |
| Nie | OK | ZERO | ZERO | OK | OK | ZERO |
| Nur wenn bestätigt | DUP | OK | ZERO | DUP | OK | ZERO |
| Nur wenn nicht bestätigt | OK | ZERO | OK | OK | DUP | OK |

OK = Text wird einmal gedruckt
DUP = Text wird zweimal gedruckt
ZERO = Text wird gar nicht gedruckt

M → Senden der Fertigungsstellungsnachricht
P → Ausführung (z.B. Drucken eines Textes)
C → Absturz

# Client-Server-Kommunikation
## RPC-Semantik bei Fehlern

- Antwortnachricht vom Server an Client geht verloren
  - Ansatzmöglichkeiten:
    - Idempotente Operationen – Operationen sind beliebig oft wiederholbar ohne eine Änderung des Systemzustandes
      - z.B. Anforderung der ersten 1024 Bytes einer Datei
      - Negativbeispiel: Überweisung auf ein Bankkonto
    - Client weist Anforderungen Folgenummer zu
      - Server unterscheidet zwischen Originalübertragung und wiederholter Übertragung
      - Nachteil: Server muss Zustandsinfos des Clients pflegen
    - Bits im Nachrichtenheader hinsichtlich Originalanforderung oder wiederholter Übertragung
      - Ausführung der Orginalanforderung immer sicher
      - Besondere Sorgfalt bei wiederholten Übertragungen

# Client-Server-Kommunikation
## RPC-Semantik bei Fehlern

- Clientabsturz nach dem Senden der Anforderungsnachricht
  - Resultat: Berechnung ist aktiv, aber niemand wartet darauf
  - Unerwünschte Berechnung = Waise (Orphan)
  - Problemmöglichkeit: Client rebootet und empfängt unerwartete Antwort
  - *Welche Umgangsmöglichkeiten gibt es hier?*
  -

# Client-Server-Kommunikation
## RPC-Semantik bei Fehlern

- Clientabsturz nach dem Senden der Anforderungsnachricht
  - Resultat: Berechnung ist aktiv, aber niemand wartet darauf
  - Unerwünschte Berechnung = Waise (Orphan)
  - Problemmöglichkeit: Client rebootet und empfängt unerwartete Antwort
  - Umgangsmöglichkeiten (4 Stück)
    - Exterminierung von Waisen
      - Client-Stub vermerkt Nachricht vor Absenden auf Festplatte
      - Nach Reboot: Infos auf Festplatte vorhanden → ggf. explizites Beenden der Waisen
    - Reinkarnation
      - Aufteilung der Zeit in aufeinanderfolgende nummerierte Zyklen
      - Nach Reboot: Multicast eines neuen Zyklus
      - Prozesse zu Client mit alter Zyklusnummer werden gelöscht

# Client-Server-Kommunikation
## RPC-Semantik bei Fehlern

- Clientabsturz nach dem Senden der Anforderungsnachricht – Teil 2
  - Umgangsmöglichkeiten (4 Stück)
    - Verfall
      - Zuweisung einer Zeitspanne T für Durchführung eines Auftrages
      - Falls kein Abschluss in T: Server fordert neue Zeitspanne an
      - Nach Reboot (nach Warten einer Zeitspanne T): Waisen sind hoffentlich verschwunden
  - Aber
    - Löschen eines Waisen ggf. problematisch,
      - falls Sperren angelegt wurden
      - falls Einträge in entfernte Warteschlangen erstellt wurden
      - ...

# Client-Server-Kommunikation
## RPC-Semantik bei Fehlern

- Clientabsturz nach dem Senden der Anforderungsnachricht – Teil 2
  - Umgangsmöglichkeiten (4 Stück)
    - Freundliche Reinkarnation
      - Mit Zyklusnachricht: Prüfung aller entfernten Berechnungen
      - Wenn Eigentümer nicht auffindbar: Löschung der Berechnung

# Themenüberblick

- **Fehlertoleranz**
  - Grundbegriffe
  - Prozess-Resilienz
  - Client-Server-Kommunikation
  - **Gruppenkommunikation**

# Gruppenkommunikation

- Zielsetzung: Zuverlässiges Multicasting
- Gründe:
  - Einzelne Punkt-zu-Punkt-Kommunikation vergeudet Bandbreite
- Probleme:
  - Ein Prozess stürzt während der Kommunikation ab.
  - Ein Prozess kommt während der Kommunikation hinzu.
- Zuverlässigkeit, wenn garantiert werden kann, dass alle nicht fehlerhaften Gruppenmitglieder die Nachricht erhalten.
- Fragestellung:
  - Wer gehört alles zur Gruppe?
  - Wie sieht es hinsichtlich der Reihenfolge aus?

# Zuverlässiges Multicasting

- Überlegen Sie sich Möglichkeiten um zuverlässiges Multicasting zu realisieren.

- Siehe z.B. Buch „Verteilte Systeme: Prinzipien und Paradigmen“ von Andrew S. Tanenbaum und Maarten van Steen (2te aktualisierte Auflage) – Seite 376 bis 380

# Zuverlässiges Multicasting

- Realisierung einer schwachen Form des zuverlässigen Multicastings
  - Annahme:
    - 1 Sender und endliche Anzahl an Empfängern
    - zugrundeliegendes Kommunikationssystem unterstützt nur unzuverlässiges Multicasting
      - Nachricht ggf. nicht an alle Gruppenmitglieder aufgrund des Verlustes einer Nachricht

# Zuverlässiges Multicasting

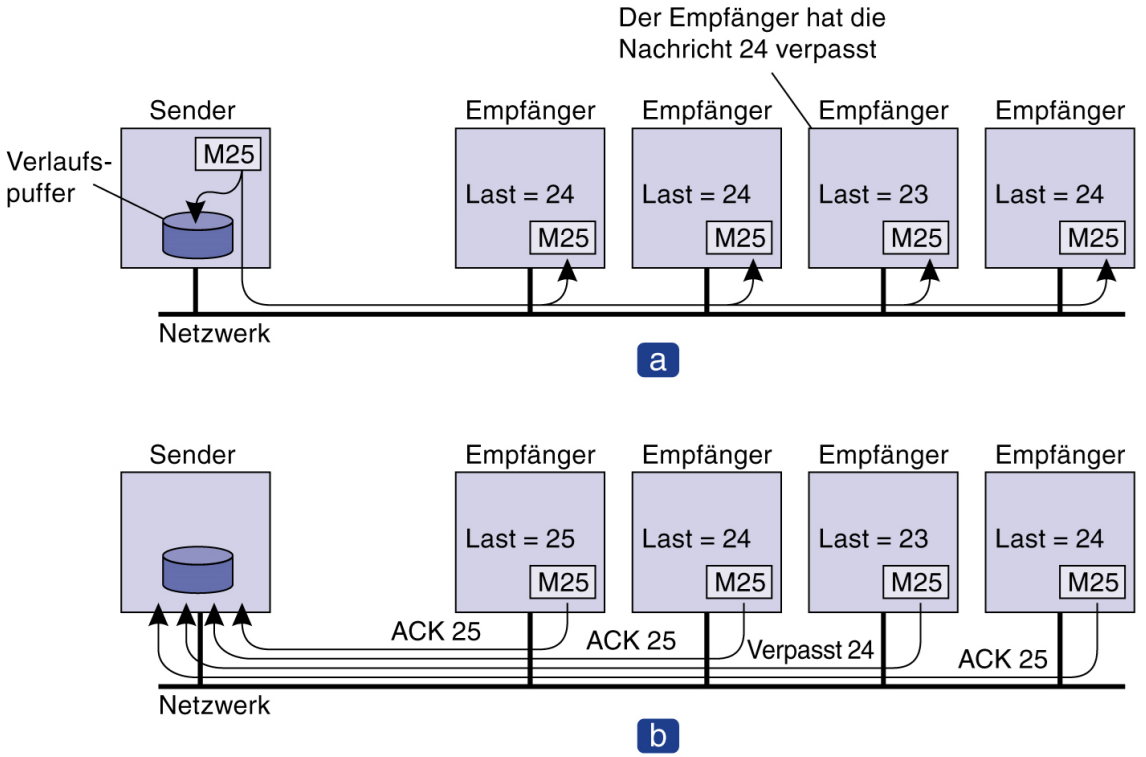
- Realisierung einer schwachen Form des zuverlässigen Multicastings
  - Ansatz:
    - Sender weist jeder Nachricht Folgenummer zu
    - Empfänger bestätigen Erhalt
    - Empfänger geben ggf. Rückmeldung, wenn Nachricht fehlt
    - Bedingung: Sender speichert Nachricht im Verlaufspuffer bis alle Empfänger Bestätigung gesendet haben

# Zuverlässiges Multicasting

- Realisierung einer schwachen Form des zuverlässigen Multicastings
  - Modifikation:
    - Erneutes Senden der Nachricht, wenn nicht Rückmeldung von allen innerhalb gewisser Zeitspanne
    - Bestätigungsnachrichten im Zusammenhang mit anderen Nachrichten
    - Unicast bei erneutem Senden

# Zuverlässiges Multicasting

- Realisierung einer schwachen Form des zuverlässigen Multicastings – Teil 2
  - Visualisierung:



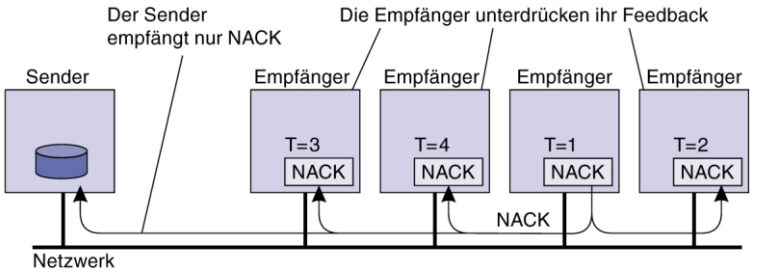

  - Probleme:
    - Rückmeldungsimplosion
      → Skalierbarkeit
    - Ggf. Nachrichten ewig im Verlaufspuffer des Senders

# Zuverlässiges Multicasting

- SRM-Protokoll (Scalable Reliable Multicasting)
  - Kernidee: Rückmeldungsunterdrückung
  - Konzept einer nichthierarchischen Rückkopplungssteuerung
  - Realisierung:
    - Erkennen einer fehlenden Nachricht (Achtung: anwendungsabhängig)
    - Rückmelder warten eine zufällige Zeit
    - Rückmeldung via Multicast
    - Keine Rückmeldung nötig, wenn anderer Prozess bereits Rückmeldung gesendet hat

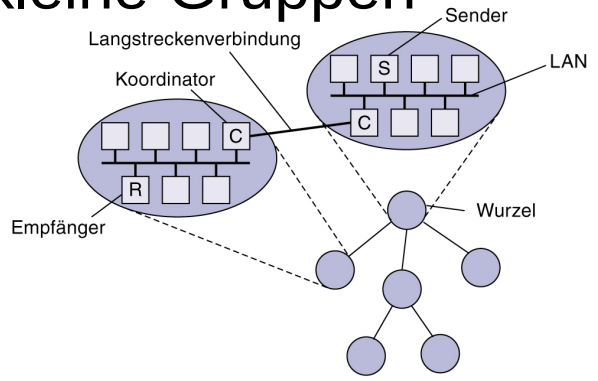
# Zuverlässiges Multicasting

- SRM-Protokoll (Scalable Reliable Multicasting)
  - Nachteil
    - Last auf alle Empfänger, die Nachricht erhalten haben
  - Modifikationen
    - Einschränkung des Multicastings durch Untergruppenbildung
    - Empfänger, dem erfolgreich Nachricht zugesandt wurde, kann bei Rückmeldung Nachricht selber via Multicast senden

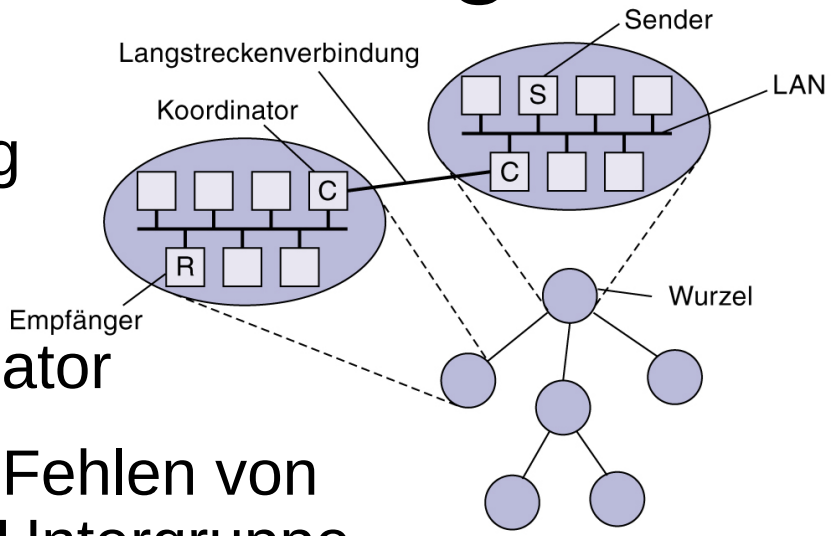


# Zuverlässiges Multicasting

- Hierarchische Rückkopplungssteuerung
  - Annahme: 1 Sender und sehr große Gruppe von Empfängern
  - Multicast-Baum
    - Gruppe von Empfängern in Untergruppen unterteilt
    - Anordnung der Untergruppen als Baum
    - Wurzel des Baumes: Untergruppe mit dem Sender
    - Innerhalb jeder Gruppe: beliebiges Verfahren für zuverlässiges Multicasting für kleine Gruppen



# Zuverlässiges Multicasting




- Hierarchische Rückkopplungssteuerung
  - Koordinator
    - Jede Gruppe hat einen Koordinator
    - Koordinator meldet Erhalt bzw. Fehlen von Nachrichten an übergeordnete Untergruppe
    - Koordinator bearbeitet Anfragen zum erneuten Senden
  - Problematik:
    - Aufbau des Baumes, da meist dynamisch

# Atomares Multicasting

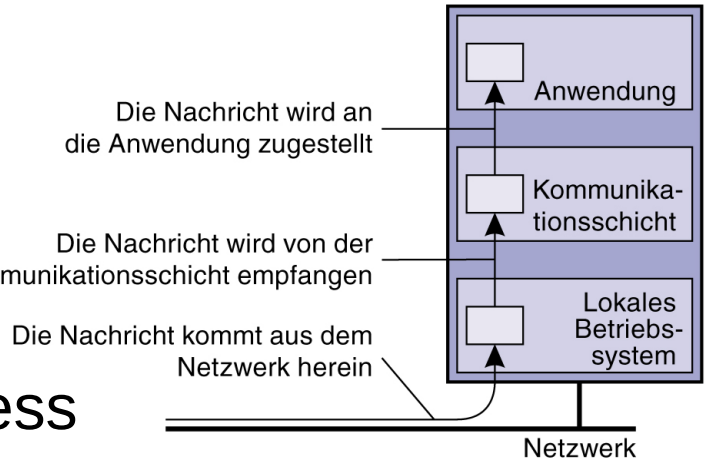
- Begrifflichkeit Atomares Multicasting
  - Zuverlässiges Multicasting bei Vorliegen von Prozessausfällen
  - Garantie: Auslieferung an alle oder keinen
  - Alle Nachrichten in der gleichen Reihenfolge

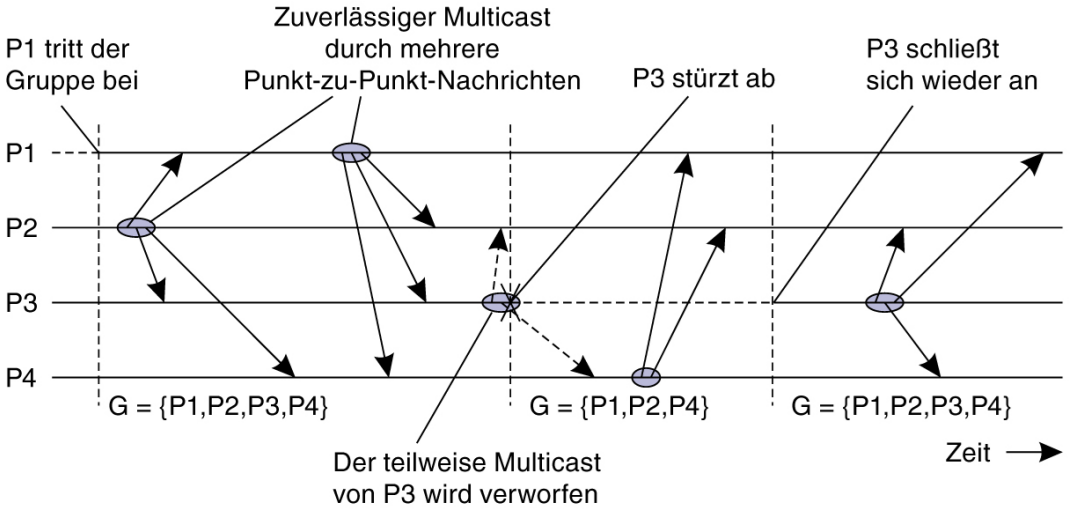
# Atomares Multicasting

- Konzept
  - Klare Entscheidungen bzgl. Gruppenmitgliedschaft
    - Aktualisierung erst, wenn abgestürzte Replik nicht zur Gruppe
    - Neue bzw. rebootete Repliken erhalten erst Aktualisierungen, wenn sie als Mitglied registriert sind
  - Beitritt zu einer Gruppe nur wenn der gleiche Zustand wie alle anderen Gruppenmitglieder
  - Nicht fehlerhafte Prozesse haben konsistente Sicht auf Datengrundlage

# Atomares Multicasting
# Virtuelle Gleichzeitigkeit

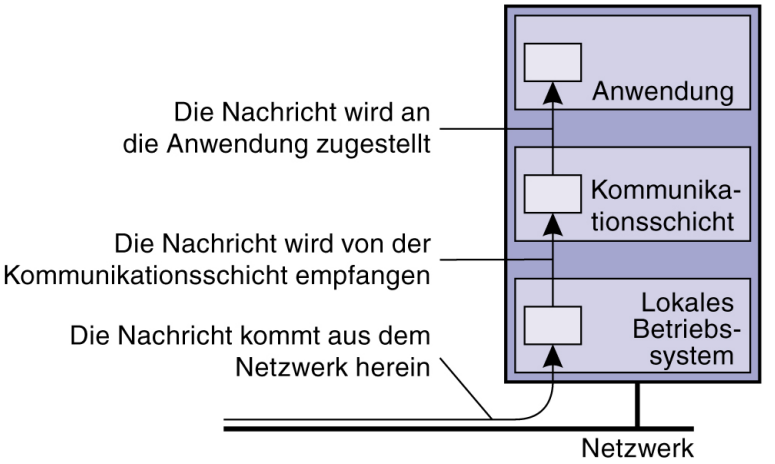
- Realisierung atomaren Multicastings – Architekturkonzepte
  - Unterscheidung zwischen Empfang und Auslieferung
  - Einführung der Gruppensicht
    - Änderung der Gruppensicht meint einen Ein- oder Austritt eines Prozess zu der Gruppe

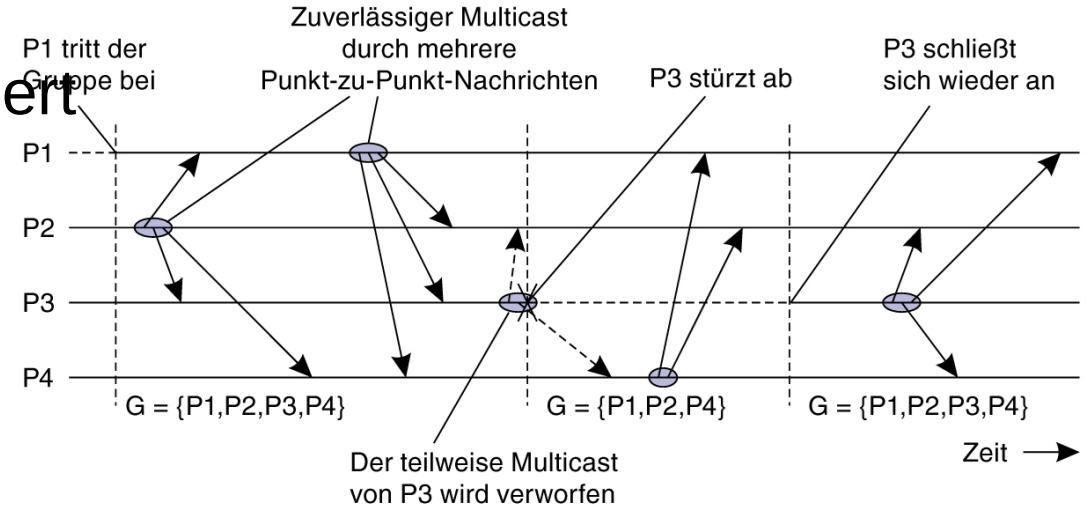


# Atomares Multicasting
# Virtuelle Gleichzeitigkeit

- Realisierung atomaren Multicastings – Architekturkonzepte
  - Virtuelle Gleichzeitigkeit:
    - Problem: Während Senden an Gruppe stürzt Sender ab
    - Resultatmöglichkeiten
      - Dennoch Auslieferung der Nachricht an alle übrigen Prozesse
      - Nachricht wird ignoriert



# Atomares Multicasting
# Anordnung von Nachrichten

- Unterscheidung von Multicasts
  - Nicht geordnete Multicasts
  - FIFO-geordnete Multicasts
  - Kausal geordnete Multicasts
  - Total geordnete Multicasts

| Prozess P1 | Prozess P2 | Prozess P3 |
|---|---|---|
| Sendet m1 | Empfängt m1 | Empfängt m2 |
| Sendet m2 | Empfängt m2 | Empfängt m1 |

| Prozess P1 | Prozess P2 | Prozess P3 | Prozess P4 |
|---|---|---|---|
| Sendet m1 | Empfängt m1 | Empfängt m3 | Sendet m3 |
| Sendet m2 | Empfängt m3 | Empfängt m1 | Sendet m4 |
| | Empfängt m2 | Empfängt m2 | |
| | Empfängt m4 | Empfängt m4 | |

- Nicht geordnete Multicasts
  - Keine Garantien bzgl. Anordnung
- FIFO-geordnete Multicasts
  - Eingehende Nachrichten des gleichen Prozesses überall in der gleichen Reihenfolge

# Atomares Multicasting
# Anordnung von Nachrichten

- Kausal geordnete Multicasts
  - Kausalität zwischen unterschiedlichen Nachrichten bleibt erhalten (Nachrichten ggf. von unterschiedlichen Sendern)
- Total geordnete Multicasts
  - Alle Gruppenmitglieder sehen dieselbe Reihenfolge
  - Keine Angabe über Art der Ordnung

- Atomares Multicasting meint
Virtuell gleichzeitiges zuverlässiges Multicasting mit total geordneter Auslieferung
  - Auslieferung via FIFO oder kausal möglich

# Atomares Multicasting

- Idee einer Realisierung virtueller Gleichzeitigkeit
  - Grunddesign:
    - Verwendung zuverlässiger Punkt-zu-Punkt-Kommunikation (TCP)
    - → Übertragung garantiert erfolgreich,
    - aber Senderabsturz bevor alle Gruppenmitglieder etwas erhalten haben
  - Vorgehen bzgl. Nachrichten:
    - Prozess in G speichert Nachrichten
    - Auslieferung erst, wenn Nachricht stabil = Nachricht von alle Prozesse von G empfangen
    - Sicherstellung der Stabilität – beliebiger (funktionierender) Prozess sendet Nachricht noch einmal an alle anderen Prozesse
  - Vorgehen bei Änderung der Gruppensicht:
    - (a) Erhalt einer Nachricht für Wechsel auf $G_{i+1}$
    - (b) Senden aller instabilen Nachrichten +
    - Senden einer Flush-Nachricht (Leerungsnachricht)
    - (c) Wechsel auf $G_{i+1}$ bei Erhalt aller andere Flush-Nachrichten