Verteilte Systeme

Vorlesung 9
Sebastian Iwanowski
FH Wedel

Client-zentrierte Synchronisation

Client-basierte Konsistenz

Anforderung:

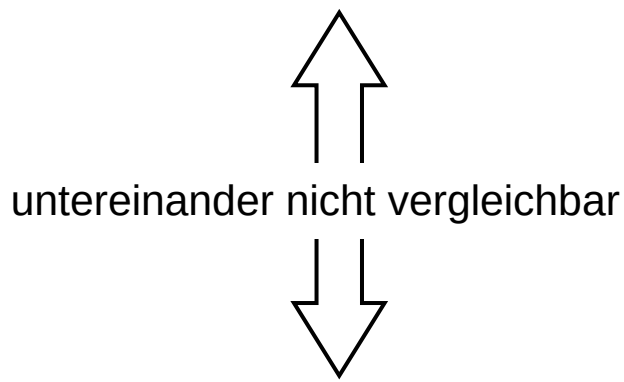
Client darf nur Daten sehen, die er selbst nicht als inkonsistent beweisen kann.

Erschwerendes Problem:

Clients arbeiten im Laufe der Zeit mit unterschiedlichen Kopien.

Konsistenzforderungen:

- **monotones Lesen**
- **monotones Schreiben**
- **Read your Writes**
- **Write follows Read**



Monotones Lesen

Definition:

Client bekommt beim wiederholten Lesen desselben Datensatzes mindestens genau so aktuelle Werte zurück wie beim letzten Mal

Lösung: Verwaltung von Mengen von Schreib-IDs

- **Lesemenge: Menge aller Schreib-IDs, die für diesen Lesevorgang relevant sind**
- **Schreibmenge: Menge aller Schreib-IDs, die von diesem Client durchgeführt worden sind**

Jeder Server, von dem gelesen werden soll, bekommt mit der Leseanforderung die aktuelle Lesemenge.
Vor Rückgabe des Wertes überprüft der Server eventuell erforderliche Aktualisierungen und führt diese durch.

Reicht es aus, wenn die Lesemenge eine Teilmenge der Schreibmenge ist ?

Weitere Konsistenzforderungen

Monotones Schreiben:

Client schließt jede Schreiboperation auf einem Datensatz ab, bevor er eine neue auf demselben Datensatz beginnt.

Read your Writes:

Jede Änderung, die durch denselben Client durchgeführt worden ist, wird in nachfolgenden Leseoperationen berücksichtigt.

Write follows Read:

Das Schreiben eines Datensatzes erfolgt nur auf Kopien, die mindestens so aktuell sind wie alle, die zuvor gelesen wurden.

Wer initiiert das Anlegen von Datenkopien ?

- **automatisch: Permanente Kopien**

- **Server ergreift Initiative: push-Caches**

  aus Kapazitätsgründen

  Heranschieben der Daten an häufig fragende Clients

  Lösung: Regelungstechnischer Ansatz

  gibt Schwellen vor für:
  - ***Löschen einer Datenkopie***
  - ***Erstellen einer Datenkopie***

- **Client ergreift Initiative: Caches**

  zum Verbessern der Antwortzeiten

Was wird bei Aktualisierungen weitergegeben ?

1) Ein aktualisierter Datensatz

2) Benachrichtigung über einer Aktualisierung

3) Eine Aktualisierungsoperation, mit der man einen alten Datensatz auf den neuesten Stand bringen kann

Wie wird eine Aktualisierung weitergegeben ?

- **push**

  für permanente Kopien

  für Server-initiierte Kopien

  für Client-initiierte Kopien bei Verwendung großer Caches durch mehrere Clients

- **pull**

  für Client-initiierte Kopien bei kleinen individuellen Caches

- **Mischform *lease*:**

  Client bitten Server um Versorgung mit Aktualisierungen (pull)

  Server verspricht und übernimmt Aktualisierungen für bestimmte Zeit (push)

Synchronisation von Daten: Zusammenfassung

Typen von Konsistenzforderungen

- **datenzentriert**
  - **strenge Konsistenz**
  - **sequentielle Konsistenz**
    - primärbasiertes Schreiben
      - entferntes Schreiben
      - lokales Schreiben
    - mehrfaches Schreiben
  - **kausale Konsistenz**
    - FIFO
  - **schwache Konsistenz**
- **clientzentriert**
  - **monotones Lesen**
  - **monotones Schreiben**
  - **read your writes**
  - **write follows read**

Synchronisation von Daten: Zusammenfassung

Typen von Konsistenzlösungen

Wer ?	*Wie ?*
• Server-getrieben (push)	• eifrig (eager, greedy)
• Client-getrieben (pull)	• träge (lazy)
• Mischform (lease)	

Verteilte Systeme

1. Innovative Beispiele aus der Praxis
2. Allgemeine Anforderungen und Techniken verteilter Systeme
3. Die Client-Server-Beziehung und daraus entstehende Fragestellungen
4. Nebenläufigkeitstechniken in Java
5. Entfernte Aufrufe
6. Objektmigration
7. Agententechnologie
8. Dienstevermittlung
9. Synchronisation von Daten
10. Konzepte zur Erzielung von Fehlertoleranz
11. Web Services

Fehlertypen

Server-Fehler:

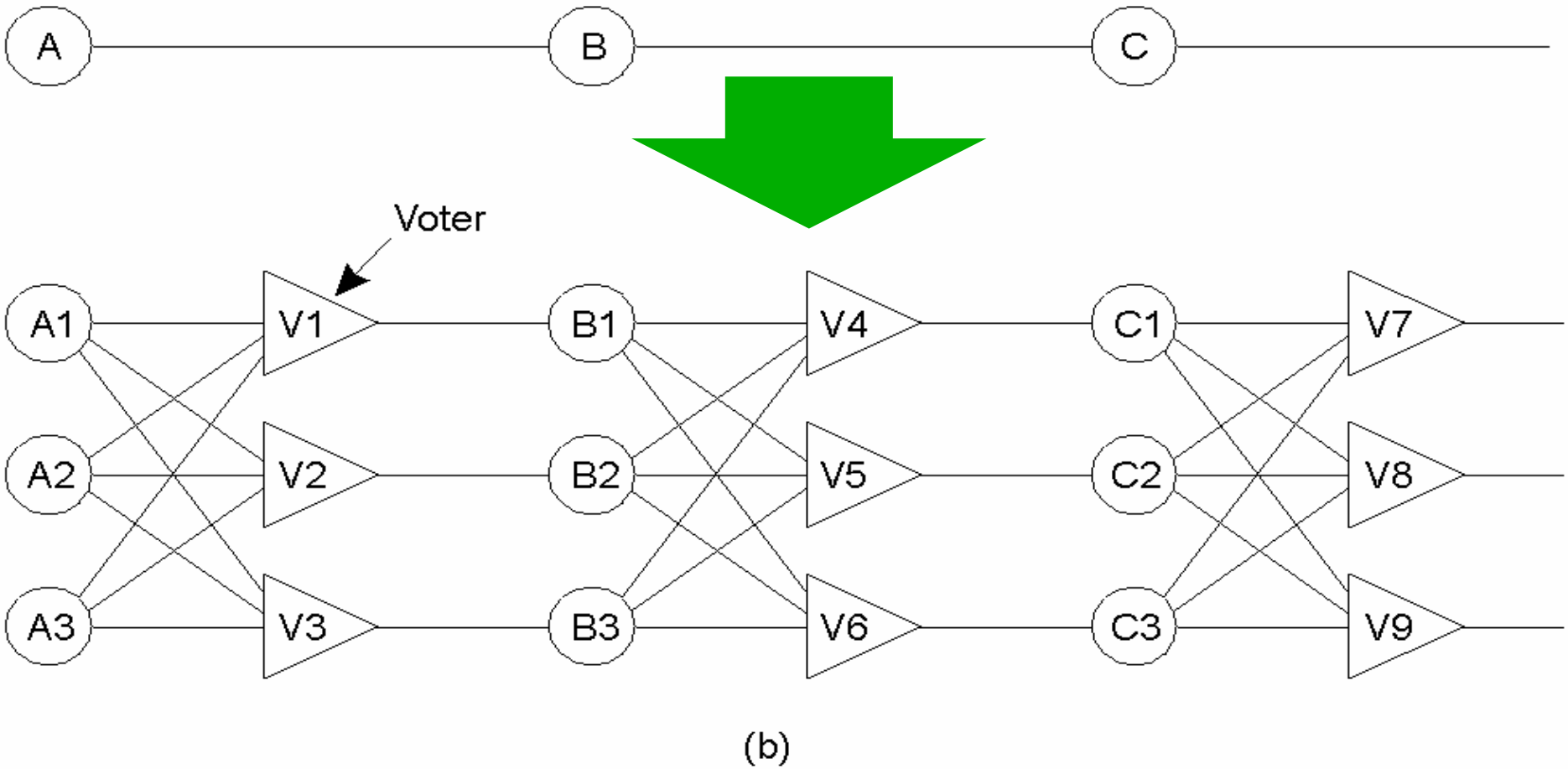
1) Totalausfall
2) Auslassungsfehler
3) Verzögerungsfehler
4) Verfälschungsfehler
5) Zufällige Fehler

Leitungsfehler:

1) Totalausfall
2) Sporadischer Ausfall

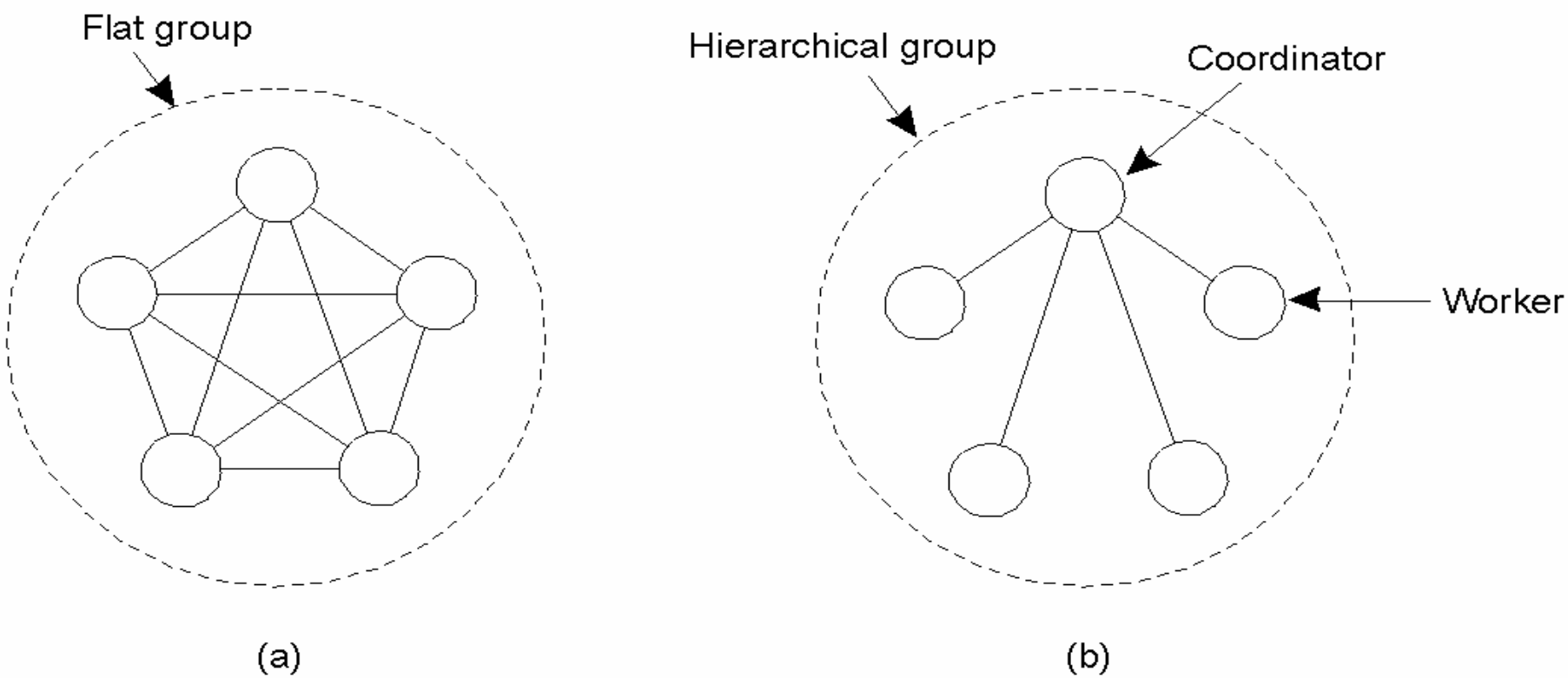
Fehlertoleranz durch Redundanz

Prinzip:



Koordination der redundanten Server

Flache vs. hierarchische Koordination:



Fehlermaskierungsverfahren (Server)

Gegeben:

- **n redundante Server**
- **Jeder Server sammelt Informationen**
- **Die Server tauschen periodisch die Informationen aus mit dem Ziel, dass alle den gleichen Informationsstand haben**
- **Einige Server können fehlerhaft arbeiten !**

Gesucht:

- **Verfahren, das erreicht, dass die korrekten Server am Ende die richtigen Informationen haben**

Fehlermaskierungsverfahren (Server)

Flache Koordination:

- **Die Informationen aller Server werden als gleichwertig angesehen und verarbeitet**

Hierarchische Koordination:

- **Alle Informationen gehen beim Koordinator ein und werden von dort weiter verteilt.**

Gesucht:

- **Verfahren, das erreicht, dass die korrekten Server am Ende die richtigen Informationen haben**

Fehlermaskierungsverfahren (Server)

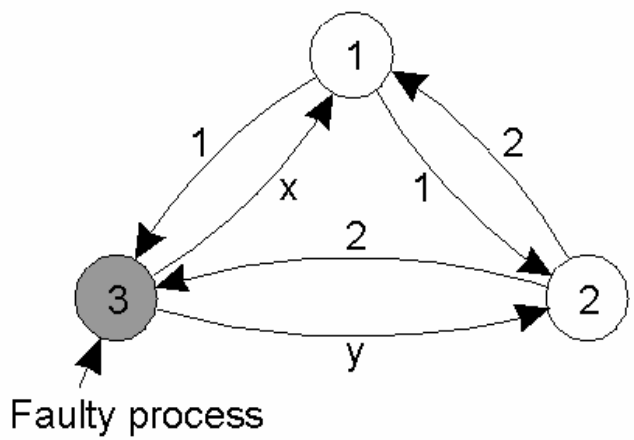
Lösungsverfahren für die flache Koordination: Mehrheitsentscheid

a) **Die Server tauschen sich paarweise aus.**

b) **Jeder Server bildet Vektoren aus den Werten, die er von den anderen Servern erhalten hat.**

c) **Die Server tauschen paarweise diese Vektoren aus.**

d) **Als gültige Werte werden dann die Mehrheiten an den jeweiligen Positionen angenommen**



(a)

Beispiel mit 3 Servern:

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

(b)

1 Got	2 Got
(1, 2, y)	(1, 2, x)
(a, b, c)	(d, e, f)

(c)

Fehlermaskierungsverfahren (Server)

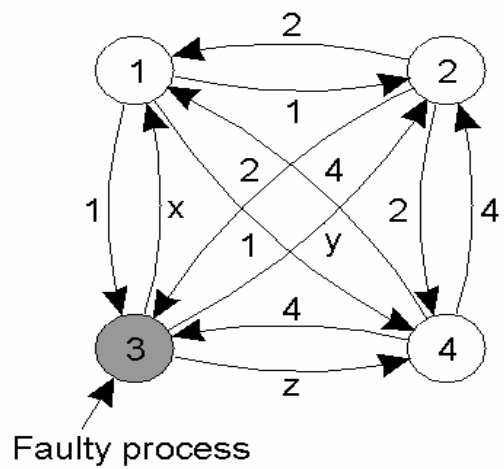
Lösungsverfahren für die flache Koordination: Mehrheitsentscheid

a) **Die Server tauschen sich paarweise aus.**

b) **Jeder Server bildet Vektoren aus den Werten, die er von den anderen Servern erhalten hat.**

c) **Die Server tauschen paarweise diese Vektoren aus.**

d) **Als gültige Werte werden dann die Mehrheiten an den jeweiligen Positionen angenommen**



(a)

Beispiel mit 4 Servern:

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)

Fehlermaskierungsverfahren (Server)

3 anscheinend verschiedene Problemstellungen:

1) **Übereinstimmung (Consensus):**
   Vor.: Alle Server erhalten gleiche Werte.
   Ziel: Die korrekten Server haben den richtigen Wert.

2) **Byzantinische Generäle:**
   Vor.: Alle Server erhalten gleichen Wert von Koordinator.
   Ziel: Die korrekten Server haben den richtigen Wert.

3) **Interaktive Übereinstimmung:**
   Vor.: Alle Server erhalten einen beliebigen Wert.
   Ziel: Die korrekten Server haben die Werte aller anderen korrekten Server.

Satz: Die Probleme sind äquivalent

Beweis: siehe Coulouris, S. 454

Fehlermaskierungsverfahren (Server)

Satz von Lamport (1982):

Im Problem der Byzantinischen Generäle braucht man mindestens 3f+1 Server, um f Fehler maskieren zu können.

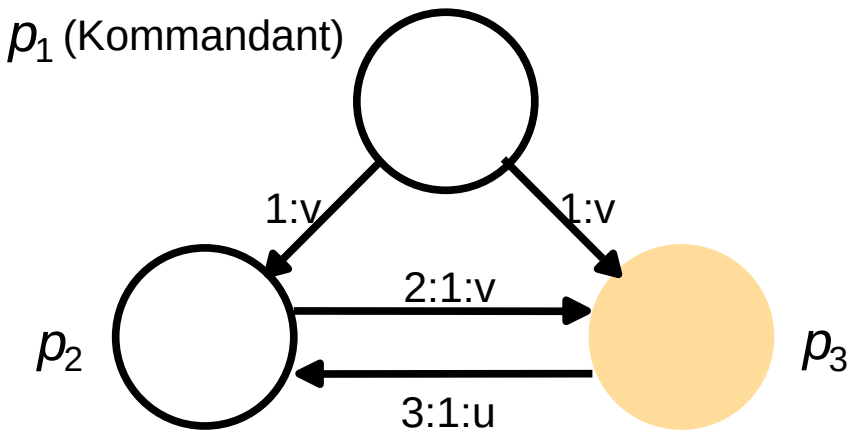
Beweisskizze: siehe Coulouris, S. 457

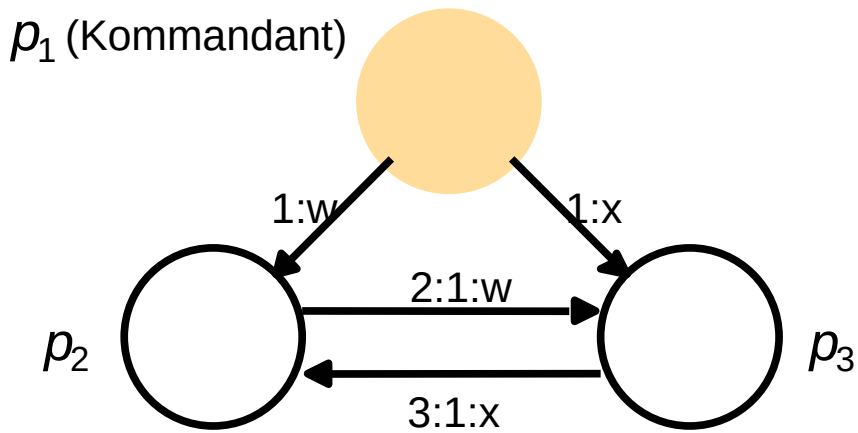
Fehlermaskierungsverfahren (Server)

Satz von Lamport (1982):

Im Problem der Byzantinischen Generäle braucht man mindestens 3f+1 Server, um f Fehler maskieren zu können.

Beispiel für f = 1: **3 Server reichen nicht aus !**





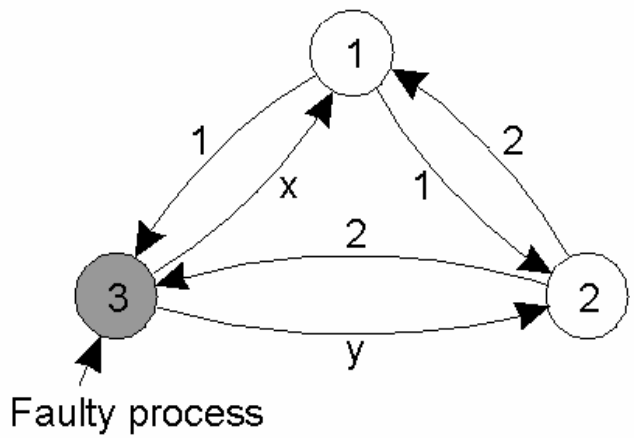
Fehlerhafte Server sind schattiert

Fehlermaskierungsverfahren (Server)

Satz von Lamport (1982):

Im Problem der Byzantinischen Generäle braucht man mindestens 3f+1 Server, um f Fehler maskieren zu können.

<u>Beispiel für f = 1:</u> **Problem *Interaktive Übereinstimmung,***
Beispiel in Tanenbaum, S. 423



(a)

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

(b)

1 Got	2 Got
(1, 2, y)	(1, 2, x)
(a, b, c)	(d, e, f)

(c)

Gegenbeispiel ?

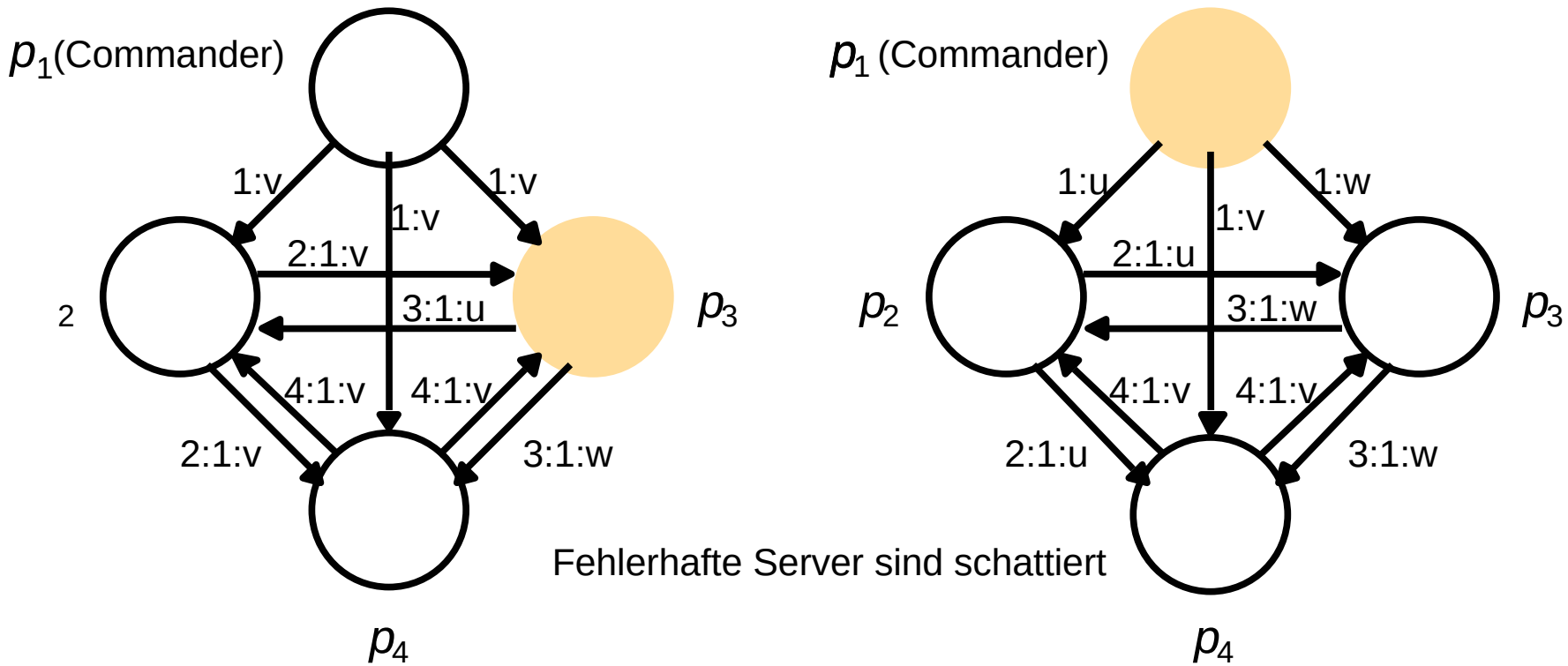
Was hat Tanenbaum nicht bedacht ?

Fehlermaskierungsverfahren (Server)

Satz von Lamport (1982):

Im Problem der Byzantinischen Generäle braucht man mindestens 3f+1 Server, um f Fehler maskieren zu können.

Beispiel für f = 1: **4 Server reichen aus !** **Wirklich ?**

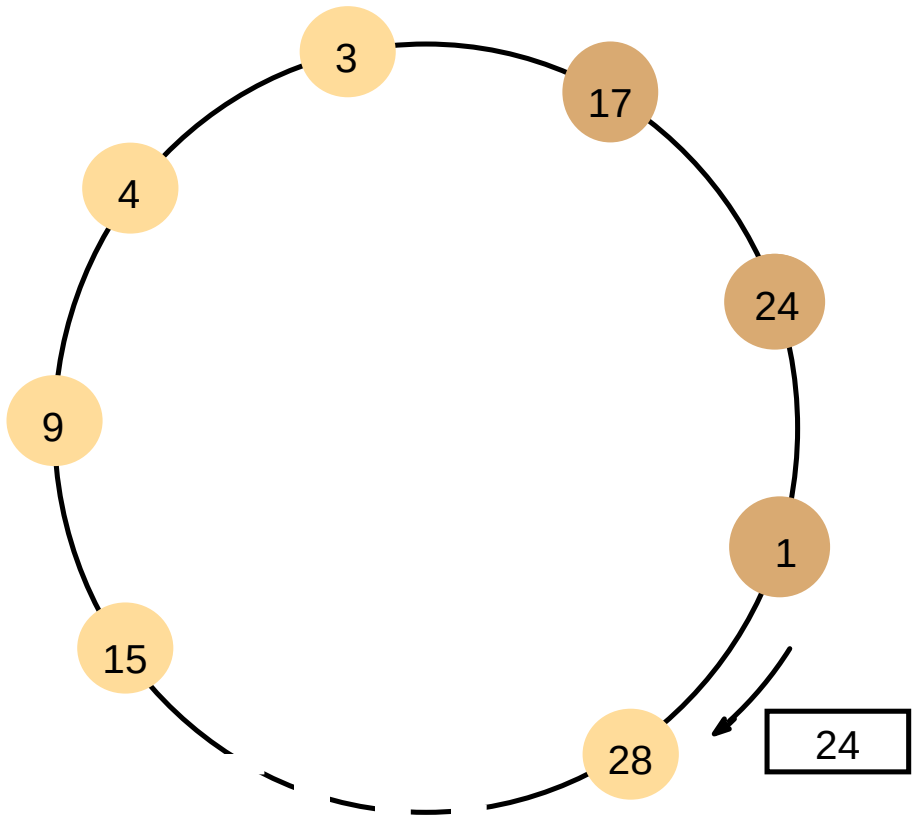


Fehlerhafte Server sind schattiert

Verteilte Auswahl eines Koordinators

Verfahren für ringförmig verbundene Server (Chang / Roberts 1979):

1) **Am Anfang ist jeder Server nichtmarkiert.**
2) **Irgendein Server beginnt, markiert sich und sendet seine ID dem Nachbarn.**
3) **Jeder Server, der eine ID empfängt, vergleicht diese mit der eigenen:**
   a) **Fremde ID > eigene ID => Server markiert sich und sendet fremde ID weiter.**
   b) **(Fremde ID < eigene ID) und (Server noch nicht markiert) => Server markiert sich und sendet fremde ID weiter.**
   c) **(Fremde ID < eigene ID) und (Server bereits markiert) => nichts**
   d) **Fremde ID = eigene ID => Dieser Server wird der Koordinator !**



Verteilte Auswahl eines Koordinators

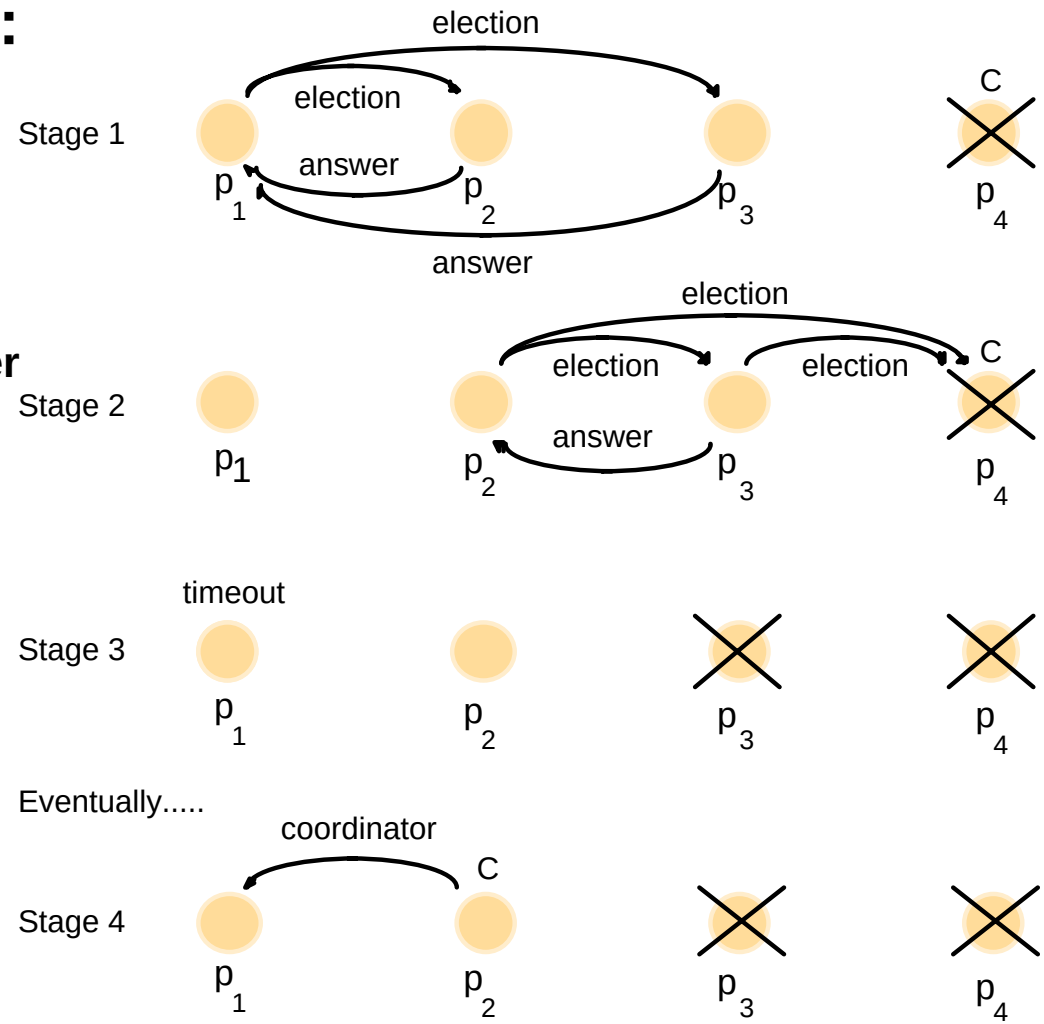
Verfahren für Server, welche alle höherwertigen kennen

(Bully-Algorithmus, Garcia-Molina 1982):

1) Jeder Server testet, ob ein höherrangiger antwortet:

a) Wenn der höherrangige antwortet, unternimmt der Server nichts weiter

b) Wenn der höherrangige nach timeout nicht geantwortet hat, erklärt sich der Server zum Koordinator und teilt das allen mit, die ihn gefragt haben.



Verteilte Auswahl eines Koordinators

Vergleich Ring-Algorithmus vs. Bully-Algorithmus ?

Beim nächsten Mal:

Graphentheoretische Verfahren
für die Erzielung von Fehlertoleranz

WebServices