## Überblick







## Literatur


- V. Garg: Elements of Distributed Computing. Wiley, 2002; Concurrent and Distributed Computing in Java, Wiley, 2004
- N. Lynch: Distributed Algorithms. Morgan Kauffmann, 1996
- P. Veríssimo, L. Rodrigues: Distributed Systems for System Architects. Kluwer Academic Publisher Group, 2000
- F. Mattern: Verteilte Basisalgorithmen. Springer-Verlag, 1989
- M. Singhal, N. G. Shivaratri: Advanced Concepts in Operating Systems. McGraw-Hill, 1994
- K. P. Birman: Building Secure and Reliable Network Applications, 1996; und Reliable Distributed Systems, 2005
- R. Guerraoui, L. Rodrigues: Introduction to Reliable Distributed Programming, Nov. 2005






## Verteilte Systeme: Definition (V. Garg) *(Wh.)*

Aus Sicht der Fehlertoleranz:

### Keine gemeinsame Uhr

- Uhrensynchronisation schwierig aufgrund nicht bekannter Verzögerungszeiten bei der Kommunikation, physikalische Uhren werden daher nur selten zur Koordinierung verwendet

### Kein gemeinsamer Speicher

- Kein einzelner Knoten kennt den vollständigen globalen Zustand. Es ist daher schwierig, globale Eigenschaften des Systems zu beobachten

### Keine akkurate Ausfallerkennung

- Es ist in einem asynchronen verteilten System (d.h. keine obere Schranke für Kommunikationszeiten bekannt) unmöglich, langsame von ausgefallenen Prozessen zu unterscheiden





## Bezeichnungen *(Wh.)*

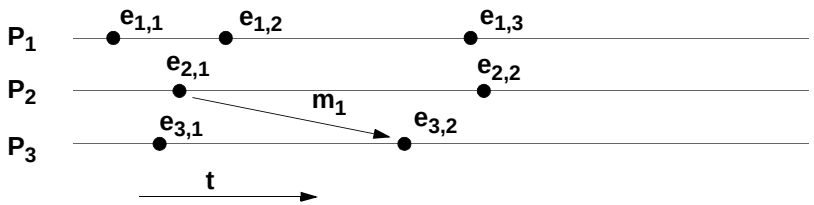
- *Knoten, Rechner, Prozessor, Prozess:*
  Meist synonym gebraucht; bezeichnen eine einzelne aktive Instanz im verteilten System, die mit den anderen Instanzen durch Nachrichtenaustausch interagiert

- *Globaler Zustand:*
  Verteilter Zustand des Systems zu einem Zeitpunkt der realen Zeit, ausgedrückt durch einen Vektor $S = [S_1, ..., S_n]$ aus den Zuständen $S_i$ aller n im System vorhandenen Knoten

- *Schritt:*
  Atomarer Übergang von einem globalen Zustand in einen Folgezustand

## Zeit-Raum Diagramm

- Graphische Darstellung von lokalen Ereignisse und Interaktionen aller Prozessoren




### Konventionen für Bezeichner

- Prozesse: Großbuchstaben ($A, B, C, \ldots$ oder $P_1, P_2, P_3, \ldots$)
- Ereignisse: Kleinbuchstaben ($a, b, c, \ldots$ oder $e_1, e_2, e_3, \ldots$)
- Nachrichten: Kleinbuchstaben ($m_1, m_2, \ldots$)
  Übertragungsrichtungen als (Sender, Empfänger) (z.B. $m_1^{2,3}$)





## Modellierung von Systemeigenschaften

- Punkt-zu-Punkt-Verbindungen
  - Zuverlässigkeit
  - Nachrichtenreihenfolge
- Multicast-Kommunikation
  - Nachrichtenreihenfolge
- Synchrone und asynchrone Systeme





## Punkt-zu-Punkt Verbindung — Zuverlässigkeit

Eigenschaften von zuverlässigen Kommunikationsverbindungen („*perfect link*“):

PL1: Zuverlässige Übermittlung: $P_a$ sendet Nachricht $m$ an $P_b$. Falls weder $P_a$ noch $P_b$ ausfällt, erhält $P_b$ $m$ in endlicher Zeit

PL2: Keine Verdopplung: Keine Nachricht wird einem Prozess mehr als einmal zugestellt

PL3: Keine Erfindung: Erhält $P_b$ eine Nachricht $m$, so wurde $m$ zuvor von einem Prozess $P_a$ gesendet

Bei den meisten betrachteten Algorithmen werden wir von zuverlässiger Kommunikation gemäß dieser Definition ausgehen!





## Punkt-zu-Punkt Verbindung — Zuverlässigkeit

### Modellierung von unzuverlässigen Verbindungen

Keine „beliebige“ Unzuverlässigkeit (sonst keine sinnvolle Validierung von Algorithmen möglich), sondern realistische Annahmen (*fair-loss* link):

FLL1: Faire Verluste: Eine von $P_a$ unendlich oft gesendete Nachricht $m$ kommt bei $P_b$ auch unendlich oft an.

FLL2: Endliche Verdopplung: Wenn $m$ endlich oft von $P_a$ nach $P_b$ gesendet wird, kommt $m$ nicht unendlich oft bei $P_b$ an.

FLL3: Keine Erfindung: Erhält $P_b$ eine Nachricht $m$, so wurde $m$ zuvor von einem Prozess $P_a$ gesendet.

## Punkt-zu-Punkt Verbindung Nachrichtenreihenfolge

PL und FLL machen keine Aussage zur Reihenfolge, in der Nachrichten ankommen

- **Ungeordnet**
  Keinerlei Aussage über Empfangsreihenfolge von Nachrichten
- **FIFO-Ordnung** (Sequentielle Ordnung, Senderordnung)
  Nachrichten, die von einem Knoten in einer bestimmten Reihenfolge ausgesendet werden, werden von anderen Knoten in genau dieser Reihenfolge empfangen

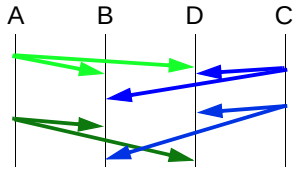




## Multicast-Kommunikation

Ordnung bezüglich eines Senders

- **ungeordnet**
- **FIFO-Ordnung**
  Nachrichten, die von einem Knoten in einer bestimmten Reihenfolge erzeugt wurden, werden von anderen Knoten in genau dieser Reihenfolge empfangen/bearbeitet Nachrichten von unterschiedlichen Knoten können in unterschiedlicher Reihenfolge empfangen werden!








## Multicast-Kommunikation

Knotenübergreifenden Ordnung

- **Kausale Ordnung** (logische Ordnung, Präzedenzordnung)
  Falls Nachricht $b$ von Nachricht $a$ beeinflusst werden konnte, trifft $a$ bei allen Knoten vor $b$ ein (Notation: $a \rightarrow b$)
- **Konsistente totale Ordnung**
  Es gibt eine systemweite totale Ordnung auf allen Nachrichten
- **Kausale totale Ordnung**
  Konsistente totale Ordnung + Kausale Ordnung
- **Physikalische Ordnung** (Ordnung nach Realzeit)
  Alle Nachrichten werden nach dem physikalischen Sendezeitpunkt geordnet empfangen





## „*Synchron*“ und „*Asynchron*“

Oft sind hier unterschiedliche Dinge damit gemeint:

- Entfernte Methodenaufrufe:
  - synchron: Aufrufer wartet (blockierend) auf Ergebnis
  - asynchron: Aurufer wartet nicht (nicht-blockierend)
- Ausführung von verteilten Aktivitäten:
  - synchron: Synonym zu „gleichzeitig“: Aktivitäten werden mit Synchronisierung ausgeführt (durch gemeinsame Uhren oder andere Synchronisierungsmechanismen gesteuert)
  - asynchron: Keine Synchronisierung vorhanden
- Koordinierung
  - „Synchronisierung“ als Synonym gebraucht; „nicht gleichzeitig“

## „*Synchron*“ und „*Asynchron*“

### Im Kontext von verteilten Algorithmen übliche Bedeutung

asynchron Unabhängigkeit von physikalischer Zeit

synchron Alle Komponenten des System halten zeitliche Schranken ein

### Systemeigenschaften eines synchronen Systems

SYN1 *Synchrone Verarbeitung*: Für alle Verarbeitungsschritte gibt es eine obere Zeitschranke

SYN2 *Synchrone Kommunikation*: Für alle Nachrichtenlaufzeiten existiert eine obere Schranke





## „*Synchron*“ und „*Asynchron*“

### Eigenschaften und Vorteile synchroner Systeme

- Ablauf in Runden: Im synchronen Modell ist es möglich, Algorithmen verteilt in Runden ablaufen zu lassen
- Ausfallerkennung durch Zeitschranken ist möglich
- Zeitbasierte Koordination (Leases) kann verwendet werden
- Eine Synchronisation physikalischer Uhren kann mit bestimmbarer Genauigkeit durchgeführt werden
- Performanz-Analyse ist möglich

### Nachteil eines synchronen Systemmodells

- In vielen Situation (z.B. internetbasierte, verteilte Systeme) realitätsfern





## Partielle Synchronität

### Problem

- synchron ⇒ Einfache Algorithmen, aber realitätsfern
- asynchron ⇒ Deckt Realität ab, aber komplexe bzw. nicht existente Algorithmen

### Versuch eines Ausweg: Partielle Synchronität

Meistens werden die Eigenschaften eines synchronen Systems erfüllt. Manchmal kann es aber Perioden geben, in denen sich das System beliebig (asynchron) verhält.

### Abstrakte Modellierung

- Es gibt einen (unbekannten) Zeitpunkt, ab dem sich das System synchron verhält





## Was ist ein Fehler? *(Wh.)*

### Fault (Fehlerusache)

- unerwünschter Zustand, der zu einem Fehler führen kann

### Error (Fehler)

- Systemzustand, der nicht den Spezifikationen entspricht

### Failure (Funktionsausfall)

- Diensterbrinung ist nicht mehr möglich

**Singal/Shivaratri**

An error is a manifestation of a fault in a system, which could lead to system failure.

## Klassifikation von Fehlern — Gutmütige Fehler *(Wh.)*

- **Crash Stop:** Ein Knoten fällt komplett aus
  - **Fail Stop**: Jeder korrekte Knoten erfährt innerhalb endlicher Zeit vom Ausfall eines Knoten
  - **Fail Silent**: Keine perfekte Ausfallerkennung möglich (asynchrones oder partiell synchrones Modell)
- **Crash Recovery**: Ein korrekter Knoten kann endlich oft ausfallen und wiederanlaufen. Ein Knoten wird nur dann als fehlerhaft betrachtet, wenn er dauerhaft ausfällt oder unendlich oft ausfällt und wiederanläuft.
  - Schwierigkeit: Stabiler Speicher, um Zustand von korrekten Knoten über Ausfälle hinweg zu erhalten
  - Zwei Spezialfälle
    - **Omission Failure:** Bei Ausfall eines korrekten Knotens gehen nur Nachrichten verloren, kompletter Zustand bleibt erhalten
    - **Timing Failure:** Es gehen weder Nachrichten noch Zustand verloren, bei einem Ausfall kann aber die spezifikationsgerechte Ausführung verzögert werden





## Klassifikation von Fehlern — Bösartige Fehler *(Wh.)*

### Bösartige Fehler (malicious faults)

- Häufig als **byzantinische** Fehler bezeichnet
- Fehlerhafte Prozesse können beliebige Aktionen ausführen, und dabei auch untereinander kooperieren
- Häufig auch etwas eingeschränktes Modell
  z.B. fehlerhafte Prozesse können beliebige mit polynomialer Komplexität berechenbare Aktionen ausführen (also u.a. keine digitalen Signaturen von korrekten Knoten fälschen!)
- Modell, das alle beliebigen Arten von Fehler umfasst, z.B. auch gezielte Angriffe von außen auf das System

Viele realen Systeme gehen von ausschließlich gutmütigen Fehlern aus. Dies muss aber nicht immer realistisch sein!





## Ausfallerkennung

### Orakel

- Beliebte Abstraktion für Algorithmen im verteilten System
  - Orakel liefert Aussagen über den Zustand von Rechnern im verteilten System
  - Orakel erfüllt formal (einfach) fassbare Eigenschaften
- Ziele/Vorteile
  - Vereinfachung von Algorithmen durch Modularisierung
  - Algorithmen können ohne der (vom Systemmodell abhängigen) Realisierung der Ausfallerkennung implementiert und verifiziert werden





## Ausfallerkennung

### Definition des „Perfect Failure Detetor“ ($P$)

- Letztendliche Vollständigkeit: Es gibt einen (unbekannten) Zeitpunkt, ab dem jeder fehlerhafte Prozess von jedem korrekten Prozess dauerhaft als ausgefallen erkannt wird
- Genauigkeit: Kein Prozess wird bevor er ausgefallen ist von einem anderen als ausgefallen erkannt

### Realisierung

- Im synchronen Modell: z.B. *Alife*-Nachrichten und Timeouts
- Im asynchronen Modell: ???

## *Letztendliche* perfekte Ausfallerkennung

- Einfacher zu realisierende Variante
- Definition des Eventually Perfect Failure Detector ($\diamond P$):
  - Letztendliche Vollständigkeit: Es gibt einen (unbekannten) Zeitpunkt, ab dem jeder fehlerhafte Prozess von jedem korrekten Prozess dauerhaft als ausgefallen erkannt wird
  - Letztendliche Genauigkeit: Es gibt einen Zeitpunkt, ab dem kein korrekter Prozess von einem anderen korrekten Prozess als ausgefallen betrachtet wird





## Analyse von Algorithmen

### Korrektheit

- **Sicherheit** („Integrität„, „safety„): Der Algorithmus liefert keine falschen Ergebnisse
- **Lebendigkeit** („liveness„): Der Algorithmus verklemmt sich nicht

Das Vorliegen von Sicherheit wird auch als „partielle Korrektheit„ bezeichnet; Von „totaler Korrektheit„ spricht man, wenn sowohl Sicherheit als auch Lebendigkeit erfüllt ist





## Analyse von Algorithmen

### Komplexität (nur ganz oberflächlich)

- O-Notation: Die Komplexität $O(f(n))$ bedeutet, dass für die tatsächliche Komplexität $p(n)$ folgendes gilt:

  Es gibt zwei Konstanten $n_0$, $c$, so dass für alle $n > n_0$ gilt:

$$p(n) \leq c * f(n) \tag{1}$$





## Bewertung von Algorithmen

- Zeitlicher Aufwand
  - mittlere/maximale Zeit bis Terminierung in Sekunden/Runden
- Nachrichtenaufwand
  - mittlere/maximale Zahl an notwendigen Nachrichten
- Verifizierbarkeit
- Implementierungsaufwand

# Zusammenfassung

- Algorithmen für Verteilte Systeme
  - Kooperierende, selbstständige Rechner
  - lose gekoppelt durch Kommunikationsnetz
- Modellierung von Kommunikation
  - Punkt-zu-Punkt: unzuverlässig, zuverlässig, FIFO
  - Multicast: ..., FIFO, kausal, total, kausal+total
- Synchrone und asynchrone Systeme
- Klassifzierung von Fehlern, Ausfallerkennung
- Analyse und Bewertung von Algorithmen