Rechnerstrukturen

Vorlesung im Sommersemester 2006

Prof. Dr. Wolfgang Karl

Universität Karlsruhe (TH)

Fakultät für Informatik

Institut für Technische Informatik

- **Kapitel 3: Multiprozessoren – Parallelismus auf Prozess/Thread-Ebene**

3.5: Multiprozessoren mit gemeinsamem Speicher

Multiprozessor mit gemeinsamem Speicher

- Speicherkonsistenzmodelle
  - Cache-Kohärenz
    - sichert, dass mehrere Prozessoren eine konsistente Sicht auf den Speicher haben
  - Wichtige Frage:
    - Wann muss ein Prozessor den Wert sehen, den ein anderer Prozessor aktualisiert hat?
    - Oder:
      - In welcher Reihenfolge muss ein Prozessor die Schreiboperationen eines anderen Prozessors beobachten?
    - Oder
      - Welche Bedingungen zwischen Lese- und Schreiboperationen auf verschiedene Speicherstellen durch verschiedene Prozessoren müssen gelten?

- Speicherkonsistenzmodelle
  - Problem: Was kann passieren?

```
P1:     A=0;                          P2:     B=0;
        …                                     …
        A=1;                                  B=1;
L1:     if (B==0)                     L2:     if (A==0)
                …führe Aktion a2 aus                  …führe Aktion a1 aus
```

Mögliche Fälle:
a1 wird ausgeführt und a2 nicht
a2 wird ausgeführt und a1 nicht
a1 und a2 werden beide nicht ausgeführt
a1 und a2 werden beide ausgeführt

Ursache für dieses Verhalten:
- Verzögerungen der Schreiboperationen im Schreibpuffer
- Verzögerungen im Netzwerk

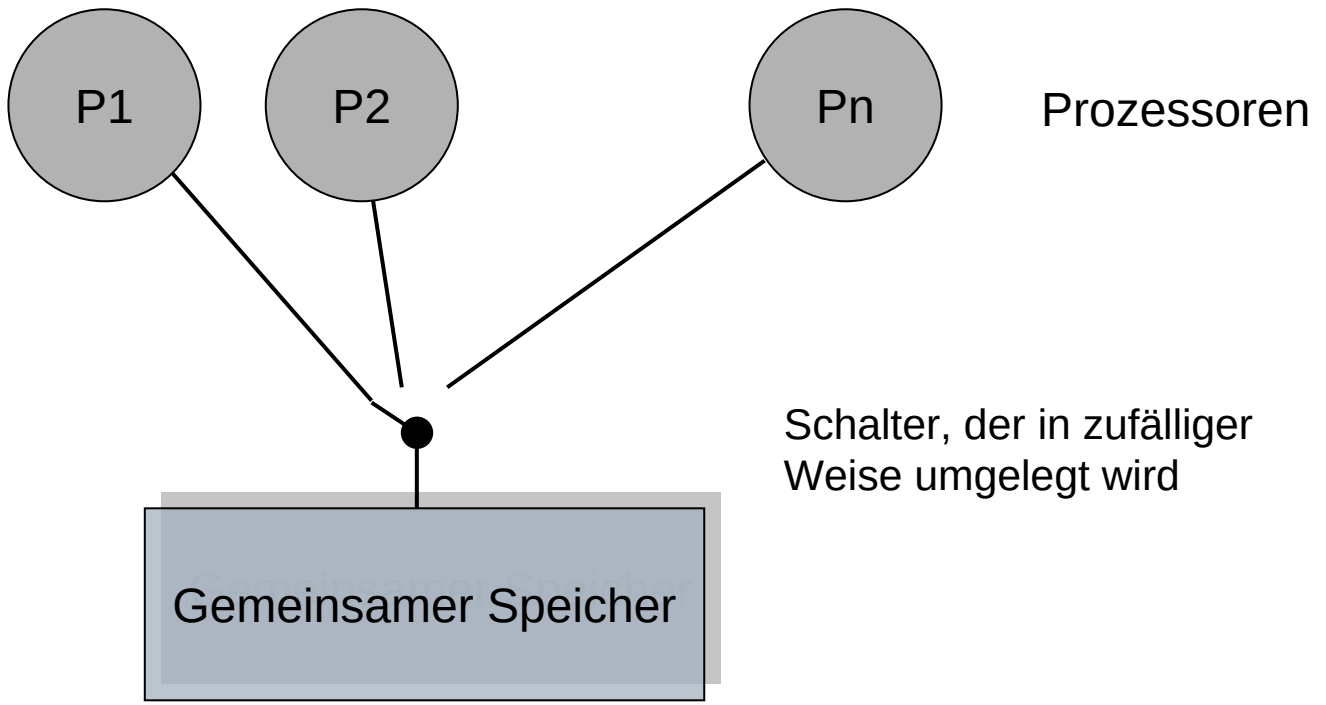
Soll dieses Verhalten erlaubt sein, und wenn ja, unter welchen Bedingungen?

Multiprozessor mit gemeinsamem Speicher

- Speicherkonsistenzmodelle
  - Spezifizieren die Reihenfolge, in der Speicherzugriffe eines Prozesses von anderen Prozessen gesehen werden
  - Sequentielle Konsistenz
    - Ein Multiprozessorsystem heißt sequentiell konsistent, wenn das Ergebnis einer beliebigen Berechnung dasselbe ist, als wenn die Operationen aller Prozessoren auf einem Einprozessorsystem in einer sequentiellen Ordnung ausgeführt würden. Dabei ist die Ordnung der Operationen der Prozessoren die des jeweiligen Programms.
    - Alle Lese- und Schreibzugriffe werden in einer beliebigen sequentiellen Reihenfolge, die jedoch mit den jeweiligen Programmordnungen konform ist, am Speicher wirksam.
    - Entspricht einer überlappenden sequentiellen Ausführung sequentieller Operationsfolgen anstelle einer parallelen Ausführung

Multiprozessor mit gemeinsamem Speicher

- Speicherkonsistenzmodelle
  - Sequentielle Konsistenz
    - Veranschaulichung der sequentiellen Konsistenz



- Speicherkonsistenzmodelle
  - Sequentielle Konsistenz
    - Programmierer geht von sequentieller Konsistenz aus
    - Führt aber zu sehr starken Einbußen bzgl. Implementierung und damit der Leistung
      - Verbietet vorgezogene Ladeoperationen, nichtblockierende Caches
  - Abgeschwächte Konsistenzmodelle
    - Konsistenz nur zum Zeitpunkt einer Synchronisationsoperation
      - Lese- und Schreiboperationen der parallel arbeitenden Prozessoren auf den gemeinsamen Speicher zwischen den Synchronisationszeitpunkten können in beliebiger geschehen.

Multiprozessor mit gemeinsamem Speicher

- Speicherkonsistenzmodelle
  - Definitionen
    - Ein Lesezugriff durch Prozessor Pi heißt zu einem bestimmten Zeitpunkt bezüglich Pk ausgeführt, wenn ein Schreibzugriff durch Prozessor Pk den Wert, den Pi durch den Lesezugriff auf dieselbe Adresse erhält, nicht mehr beeinflussen kann
    - Ein Schreibzugriff durch Pi heißt zu einem bestimmten Zeitpunkt bezüglich Pk ausgeführt, wenn ein Lesezugriff durch Pk auf dieselbe Adresse den Wert liefert, der von Pi geschrieben worden ist
    - Ein Zugriff gilt als ausgeführt, wenn er bezüglich aller Prozessoren im System ausgeführt ist
    - Ein Lesezugriff heißt global ausgeführt, wenn sowohl er als auch der Schreibzugriff, der den gelesenen Wert erzeugt, ausgeführt worden ist

Multiprozessor mit gemeinsamem Speicher

- Speicherkonsistenzmodelle
  - Definitionen
    - Unterschied zwischen ausgeführten und global ausgeführten Lesezugriff nur in Systemen, in denen der Schreibzugriff kein atomarer Vorgang ist, d.h. wenn der geschriebene Wert für alle Prozessoren des Systems nicht gleich lesbar ist
  - Sequentielle Konsistenz:
    - Hinreichende Bedingung:
      - Bevor ein Lese- oder Schreibzugriff bezüglich eines anderen Prozessors ausgeführt werden darf, müssen alle vorhergehenden Lesezugriffe global ausgeführt und alle vorhergehenden Schreibzugriffe ausgeführt sein
    - Reihenfolge der Operationen auf einem Prozessor wird beibehalten, und für alle anderen Prozessoren ist dieselbe Reihenfolge sichtbar.
    - Jeder Lese- und Schreibzugriff muss allen anderen Prozessoren vor einem nachfolgenden Lese- oder Aschreibzugriff bekannt gemacht werden
    - Wenig Spielraum für Optimierungen der Speicherzugriffe
      - Z.B. Puffern der Schreibzugriffe

Multiprozessor mit gemeinsamem Speicher

- Speicherkonsistenzmodelle
  - Prozessorkonsistenz
    - Definition (nach Goodman, 1989)
      - Ein Prozessor ist prozessorkonsistent, wenn das Ergebnis irgendeiner Ausführung dasselbe ist, als wenn die Operation eines jeden Prozessors in der sequentiellen Reihenfolge, die durch sein Programm bestimmt wird, erscheinen
    - Unterschied zur sequentiellen Konsistenz
      - Bei der Prozessorkonsistenz muss es nicht mehr für alle Prozessoren eine einheitliche Reihenfolge der Speicherzugriffe geben
      - Schreibzugriffe zweier Prozessoren können von einem dritten Prozessor in einer anderen Reihenfolge gesehen werden, als von den beiden schreibenden Prozessoren
      - Die Schreibzugriffe eines Prozessors werden jedoch von allen Prozessoren in der in seinem Programm angegebenen Reihenfolge gesehen

- Speicherkonsistenzmodelle
  - Prozessorkonsistenz
    - Definition (nach Gharachorloo, 1990)
      - Bedingung der globalen Ausführung der Lesezugriffe wird fallen gelassen
      - Es gilt:
        - » Bevor ein Lesezugriff bezüglich eines anderen Prozessors ausgeführt werden darf, müssen alle vorhergehenden Lesezugriffe ausgeführt worden sein
        - » Bevor ein Schreibzugriff irgendeines anderen Prozessors ausgeführt werden darf, müssen alle vorhergehenden Schreibzugriffe ausgeführt worden sein

- Speicherkonsistenzmodelle
  - Prozessorkonsistenz
    - Definition (nach Gharachorloo, 1990)
      - Konsequenz
        - » Prozessorkonsistenz führt nicht mehr unbedingt zur korrekten Sequentialisierung der Speicherzugriffe, da das Lesen schon erlaubt ist, bevor die vorhergehenden Schreibzugriffe alle ausgeführt worden sind.
        - » Lesezugriff kann zwar von den anderen Prozessoren nicht beobachtet werden, der Prozessor selbst sieht jedoch eine Reihenfolge der Speicheroperationen, die nicht seiner Programmordnung entspricht
        - » Schreibender Prozessor muss nicht auf die Ausführung des Schreibzugriffs bezüglich aller Prozessoren warten, bevor er eine weitere Schreiboperation veranlassen kann
        - » Puffern von Schreibzugriffen ist wieder erlaubt

- Speicherkonsistenzmodelle
  - Schwache Konsistenz
    - Bisherige Konsistenzmodelle lassen Synchronisation paralleler Threads außer Acht
    - Konkurrierende Zugriffe auf gemeinsame Daten werden durch geeignete Synchronisationen geschützt

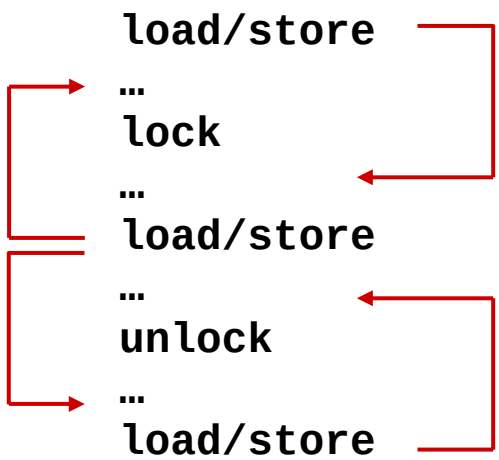
```
mutex m;
…
lock(m)
y=0;
x=0;
unlock(m)
…
P1:     lock(m)
        A=1;
        if (B==0)
                …führe Aktion a2 aus
        unlock(m)

P2:     lock(m)
        B=1;
        if (A==0)
                …führe Aktion a1 aus
        unlock(m)
```

- Speicherkonsistenzmodelle
  - Schwache Konsistenz (weak consistency)
    - Idee
      - Die Konsistenz des Speicherzugriffs wird nicht mehr zu allen Zeiten gewährleistet, sondern zu bestimmten, vom Programmierer in das Programm eingesetzten Synchronisationspunkten
      - Einführung von kritischen Bereichen
        - » Innerhalb dieser Bereich wird die Inkonsistenz der gemeinsamen Daten zugelassen
        - » Voraussetzung: konkurrierende Lese-/Schreibzugriffe sind durch den kritischen Bereich unterbunden

- Speicherkonsistenzmodelle
  - Schwache Konsistenz (weak consistency)
    - Bedingungen
      - Bevor ein Schreib- oder Lesezugriff bezüglich irgendeines anderen Prozessors ausgeführt werden darf, müssen alle vorhergehenden Synchronisationspunkte erreicht worden sein
      - Bevor eine Synchronisation bezüglich irgendeines anderen Prozessors ausgeführt werden darf, müssen alle vorhergehenden Schreib- oder Lesezugriffe ausgeführt worden sein.
      - Synchronisationspunkte müssen sequentiell konsistent sein
        - » *bevor* und *vorübergehend* beziehen sich auf die Programmordnung

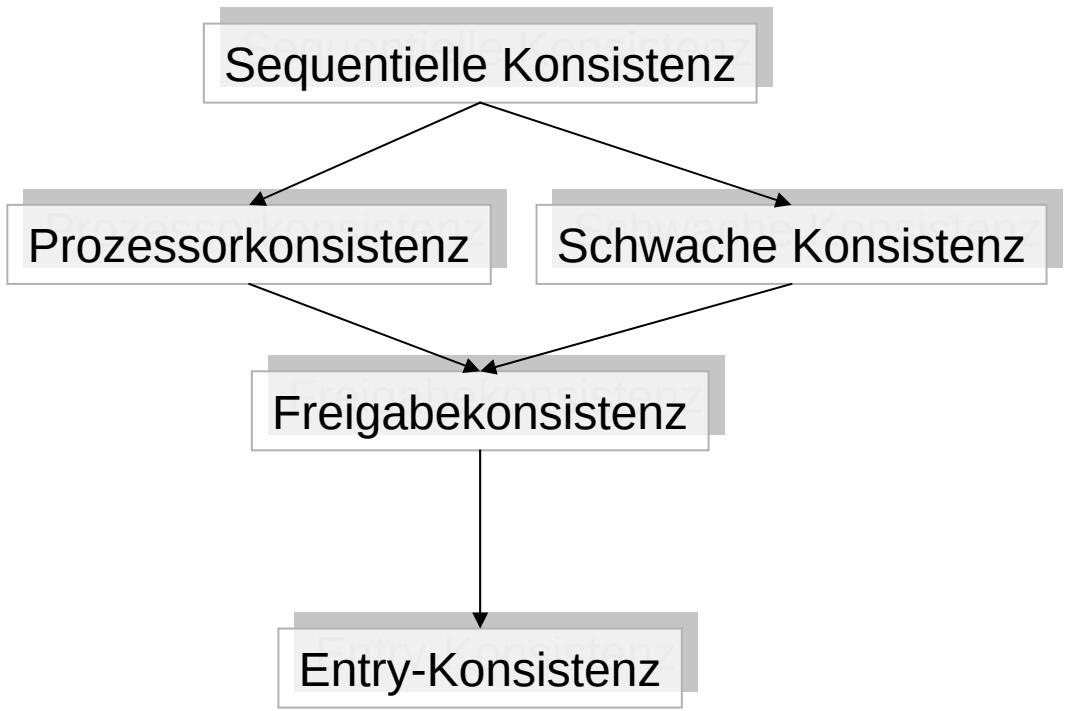
- Speicherkonsistenzmodelle
  - Schwache Konsistenz (weak consistency)
    - Auswirkung
      - Synchronisationsbefehle stellen Hürden dar, die von keinem Lese- oder Schreibzugriff übersprungen werden



Rote Pfeile: verboten

- Speicherkonsistenzmodelle
  - Schwache Konsistenz (weak consistency)
    - Auswirkung
      - Voraussetzung für die Implementierung der schwachen Konsistenz ist die hardware- und softwaremäßige Unterscheidung der Synchronisationsbefehle von den Lade- und Speicherbefehlen und eine sequentiell konsistente Implementierung der Synchronisationsbefehle
      - Das Puffern von Schreibzugriffen ist erlaubt, das von Synchronisationsbefehlen nicht!
      - „Hürdeneigenschaft“ der Synchronisationsbefehle
        - » In heutigen Mikroprozessoren mit Hilfe von Spezialbefehlen implementiert
      - Die Ordnung der Speicherbefehle wird durch die sehr viel losere Ordnung der Synchronisationsbefehle ersetzt

- Speicherkonsistenzmodelle
  - Weitere Konsistenzmodelle
    - Zusammenhang



- Speicherkonsistenzmodelle
  - Weitere Konsistenzmodelle
    - definieren theoretisch weitere Abschwächungen oder
    - leiten sich aus Hardware-Implementierungen spezieller Prozessoren oder Multiprozessorsysteme ab
      - SPARC-Prozessor-spezifisches schwaches Konsistenzmodell TSO (total store order) oder das hiervon abgeleitete PSO (patial total store)

- **Kapitel 4: Fehlertoleranz, Zuverlässigkeit**

4.1: Grundlagen

- Begriffsbildung
  - Zuverlässigkeit (dependability)
    - bezeichnet die Fähigkeit eines Systems, während einer vorgegebenen Zeitdauer bei zulässigen Betriebsbedingungen die spezifizierte Funktion zu erbringen.
    - Ziel
  - Fehlertoleranz (fault tolerance)
    - bezeichnet die Fähigkeit eines Systems, auch mit einer begrenzten Anzahl fehlerhafter Subsysteme die spezifizierte Funktion (bzw. den geforderten Dienst) zu erbringen.
    - Technik

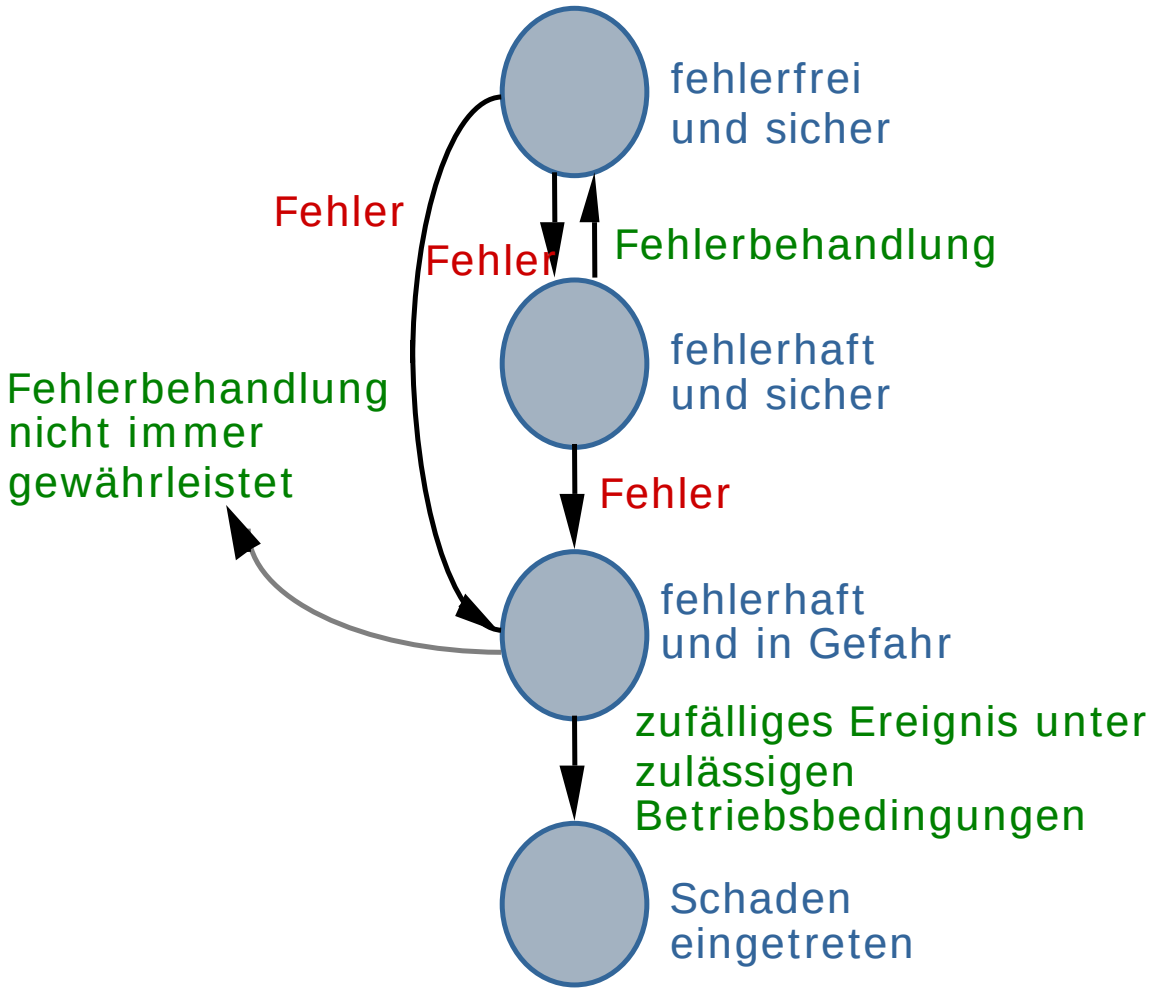
- Begriffsbildung
  - Sicherheit (safety)
    - bezeichnet das Nichtvorhandensein einer Gefahr für Menschen oder Sachwerte. Unter einer Gefahr ist ein Zustand zu verstehen, in dem (unter anzunehmenden Betriebsbedingungen) ein Schaden zwangsläufig oder zufällig entstehen kann, ohne dass ausreichende Gegenmaßnahmen gewährleistet sind.
  - Vertraulichkeit (security)
    - betrifft Datenschutz, Zugangssicherheit.

- Begriffsbildung
  - Zuverlässigkeitskenngrößen:
    - Verfügbarkeit
    - Überlebenswahrscheinlichkeit
    - Ausfallsicherheit
  - Wartungsfreundlichkeit
    - System: Instandhaltung notwendig?
    - Komponenten: Ersatz?
    - Datenbestände: langfristig lesbar?
  - Nutzungsdauer eines Rechners
    - Mindestens 5 Jahre oder 40000 h, Dauerbetrieb notwendig?
    - Sehr kleine Ausfallraten der Komponenten ($< 10^{-9}h^{-1}$)
    - Probleme: Nachweisbarkeit, Testmöglichkeiten

- Begriffsbildung
  - Ausfall
    - Hardwarekomponenten (Chips, Hintergrundspeicher)
    - Software, durch Programmfehler
    - Menschliche Eingriffe
  - Sicherheitsrelevante Anwendungen erfordern eine hohe Verfügbarkeit

- Fragen:
  - Wie zuverlässig sind heutige Rechensysteme?
  - Nutzen redundante, also fehlertolerante Strukturen zur Verbesserung der Zuverlässigkeit?
  - Welche Verbesserung der Zuverlässigkeit lässt sich durch derartige Fehlertoleranzmaßnahmen überhaupt erreichen?

- Fehler


fehlerfrei
und sicher
Fehler
Fehler
Fehlerbehandlung
fehlerhaft
und sicher
Fehlerbehandlung
nicht immer
gewährleistet
Fehler
fehlerhaft
und in Gefahr
zufälliges Ereignis unter
zulässigen
Betriebsbedingungen
Schaden
eingetreten

Zuverlässigkeit und Fehlertoleranz

- Fehler:
  - Fehlzustände oder Funktionsausfälle
    - Der Ausfall einer Komponente erzeugt einen Fehlzustand
    - Ausfall eines Systems: Versagen
  - Wirkungskette
    - Fehler → Fehlzustand → Ausfall
    - Fehlerausbreitung verhindern!
- Ziel der Fehlertoleranz:
  - Tolerierung der Fehlzustände von Teilsystemen (Komponenten)
  - Erhöhung der Zuverlässigkeit
  - Behebung der Fehlzustände vor dem Ausfall des Systems

- Fehler
  - Ursachen
    - Fehler beim Entwurf
      - Spezifikationsfehler
      - Implementierungsfehler
      - Dokumentationsfehler
      - Herstellungsfehler
    - Betriebsfehler
      - Störungsbedingte Fehler
      - Verschleißfehler
      - Zufällige physikalische Fehler
    - Bedienungsfehler
    - Wartungsfehler

- Fehler
  - Dauer und Ort
    - Fehlerentstehungsort
      - Hardware oder Software
    - Fehlerdauer
      - Permanenter Fehler
      - Temporärer Fehler

- Ausfallverhalten
  - Teilausfall
    - Von einer fehlerhaften Komponente fallen eine oder mehrere, aber nicht alle Funktionen aus
  - Unterlassungsausfall
    - Eine fehlerhafte Komponente gibt eine Zeit keine Ergebnisse aus. Wenn jedoch ein Ergebnis ausgegeben wird, dann ist dieses korrekt
  - Anhalteausfall
    - Eine fehlerhafte Komponente gibt nie mehr ein Ergebnis aus

- Ausfallverhalten
  - Haftausfall
    - Eine fehlerhafte Komponente gibt ständig den gleichen Ergebniswert aus
  - Binärstellenausfall
    - Ein Fehler verfälscht eine oder mehrere Binärstellen des Ergebnisses

- Systemausfallverhalten
  - Fail-stop-System
    - Ein System, dessen Ausfälle nur Anhalteausfälle sind
  - Fail-silent-System
    - Ein System, dessen Ausfälle nur Unterlassungsausfälle sind
  - Fail-safe-System
    - Ein System, dessen Ausfälle nur unkritische Ausfälle sind

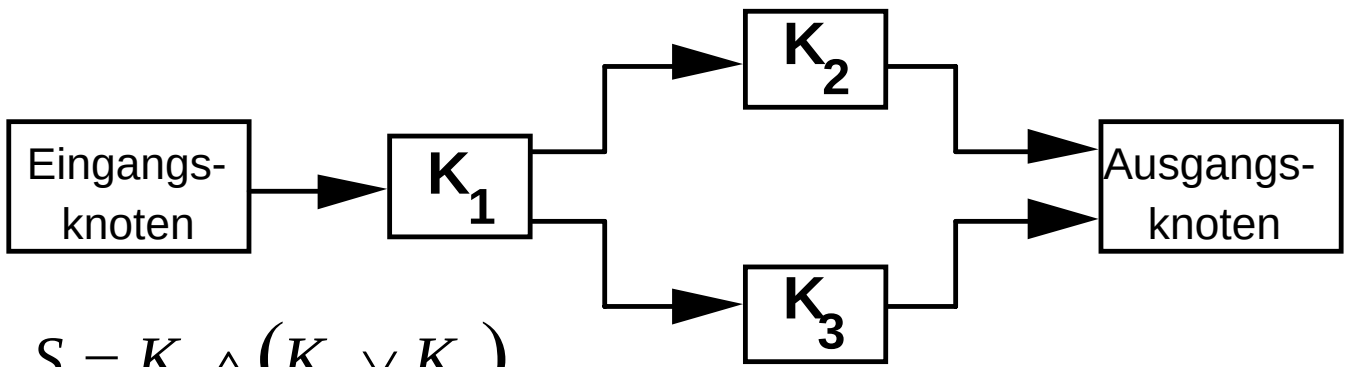
- Anforderungen
  - Hohe Überlebenswahrscheinlichkeit
    - kurzzeitige Mission (z.B. 10-stündiger Flug)
  - Hohe mittlere Lebensdauer
    - z.B. bei begrenzten Reparaturmöglichkeiten in unzugänglichen Rechensystemen
  - Hohe Verfügbarkeit
    - z.B. im interaktiven Rechenzentrums- oder Nutzerbetrieb
  - Hohe Sicherheitswahrscheinlichkeit
    - Schutz von Menschen, Maschinen, daten
  - Hohe Sicherheitsdauer

- Vorgehensweise
  - Höchste Präferenz: Fehlervermeidung
    - Perfektionierung, Verwendung von zuverlässigen Komponenten, sorgfältiger Entwurf
  - Fehlertoleranz
    - Erfordert Redundanz und damit Zusatzaufwand

- Weitere Gesichtspunkte
  - Fehlervorgabe
    - Nicht alle Fehler sind tolerierbar
  - Menge der zu tolerierenden Fehler
    - Gibt an, welche im Fehlermodell vorgesehenen Fehler tolerierbar sind
  - Fehlerbereichsannahme
    - Die Menge der zutolerierenden Fehler wird bezüglich einer Fehlerannahme formuliert
  - Festlegung:
    - wie viele Einzelfehlerbereiche können gleichzeitig fehlerhaft werden
    - Welche Fehlfunktionen sind zu behandeln

- Zusätzliche Anforderungen
  - Nachweis der Fehlertoleranz
    - Verifikation, Validierung, Durchführung einer Anfälligkeitsanalyse
  - Geringer Betriebsmittelbedarf (geringe Kosten)
  - Schnelle Ausführung von Fehlertoleranzverfahren (Leistung)
  - Unabhängigkeit von der Anwendungssoftware (Transparenz)
  - Unabhängigkeit vom Rechensystem

- Ausgangspunkt
  - Zuverlässigkeitsblockdiagramm
    - Die Systemfunktion lässt sich grafisch durch ein Zuverlässigkeitsblockdiagramm darstellen:
      - Gerichteter Graph mit einem Eingangs- und einem Ausgangsknoten



$$S = K_1 \wedge (K_2 \vee K_3)$$

- Zuverlässigkeitskenngrößen
  - Zuverlässigkeit, Sicherheit einer Rechensystems
    - Quantifizierbar mittels stochastischer Modelle
    - Man betrachtet die kontinuierliche Variable Zeit zwischen dem Zeitpunkt, ab dem die Zuverlässigkeitsbetrachtung beginnen soll (Zeitpunkt Null), bis zum Auftreten eines betrachteten Effekts
    - Nichtnegative Zufallsvariablen:
      - Lebensdauer L – besitzt die Dichte $f_L(t)$
      - Fehlerbehandlungsdauer B – besitzt die Dichte $f_B(t)$
      - Sicherheitsdauer D – *besitzt die Dichte* $f_D(t)$

- Zuverlässigkeitskenngrößen
  - Zuverlässigkeit, Sicherheit einer Rechensystems
    - Verteilungsfunktionen

$$F_x(t) := \int_0^t f_x(s)ds$$

– mit x = L, B, D

- Zuverlässigkeitskenngrößen
  - Fehlerwahrscheinlichkeit $F_L(t)$
    - Bezeichnet die Wahrscheinlichkeit, dass ein zu Beginn fehlerfreies System im Zeitintervall [0,t] fehlerhaft wird
  - Überlebenswahrscheinlichkeit

$$R(t) := 1 - F(t)$$

    - System ist von t=0 bis zum Zeitpunkt t ununterbrochen fehlerfrei.
    - $F_L(t)=0$ und $\lim_{t\to\infty} F_L(t) = 1$
      - damit folgt: R(0)=1,
      - und R ist in t monoton fallend $\lim_{t\to\infty} R(t) = 0$

- Zuverlässigkeitskenngrößen
  - Mittlere Lebensdauer

$$E(L) = \int_0^\infty R(t)dt$$

    - bezeichnet für ein zu Beginn fehlerfreies System den Erwartungswert der Zeitdauer bis zum eintreffen des ersten Fehlers
  - Ausfallrate

$$z(t) := \frac{f_L(t)}{R(t)}$$

    - bezeichnet den Anteil der in einer Zeiteinheit ausfallenden Komponenten bezogen auf den Anteil der noch fehlerfreien Komponenten

- Zuverlässigkeitskenngrößen
  - Verfügbarkeit

$$v := \frac{E(L)}{E(L) + E(B)}$$

    - Wahrscheinlichkeit, ein System zu einem beliebigen Zeitpunkt fehlerfrei anzutreffen
    - D.h., der zeitliche Anteil der Benutzbarkeit des Systems an der Summe der Erwartungswerte von Lebensdauer L und Behandlungsdauer B, wenn während B das System repariert und wieder funktionsfähig wird.

- Zuverlässigkeitskenngrößen
  - Sicherheit einer Rechensystems
    - Geht man Ausfällen aus, die die Sicherheit beeinträchtigen, dann ergeben sich analog zu den bisher betrachteten Größen solche mit Sicherheitsrelevanz
    - Gefährdungswahrscheinlichkeit $F_D(t)$
      - Wahrscheinlichkeit, dass ein zu Beginn sicheres System im Zeitintervall [0,t] in einen gefährlichen Zustand gerät
    - Sicherheitswahrscheinlichkeit $S(t) := 1 - F_D(t)$
      - Wahrscheinlichkeit, dass ein zu Beginn sicheres System bis zum Zeitpunkt t ununterbrochen in einem sicheren Zustand bleibt
    - Mittlere Sicherheitsdauer $E(D) = \int_0^\infty t \cdot f_D(t)dt = \int_0^\infty S(t)dt$
      - Erwartungswert der Zeitdauer, bis ein gefährlicher Zustand auftritt

- Zuverlässigkeitskenngrößen
  - Funktionswahrscheinlichkeit der Komponenten und Systemfunktionen

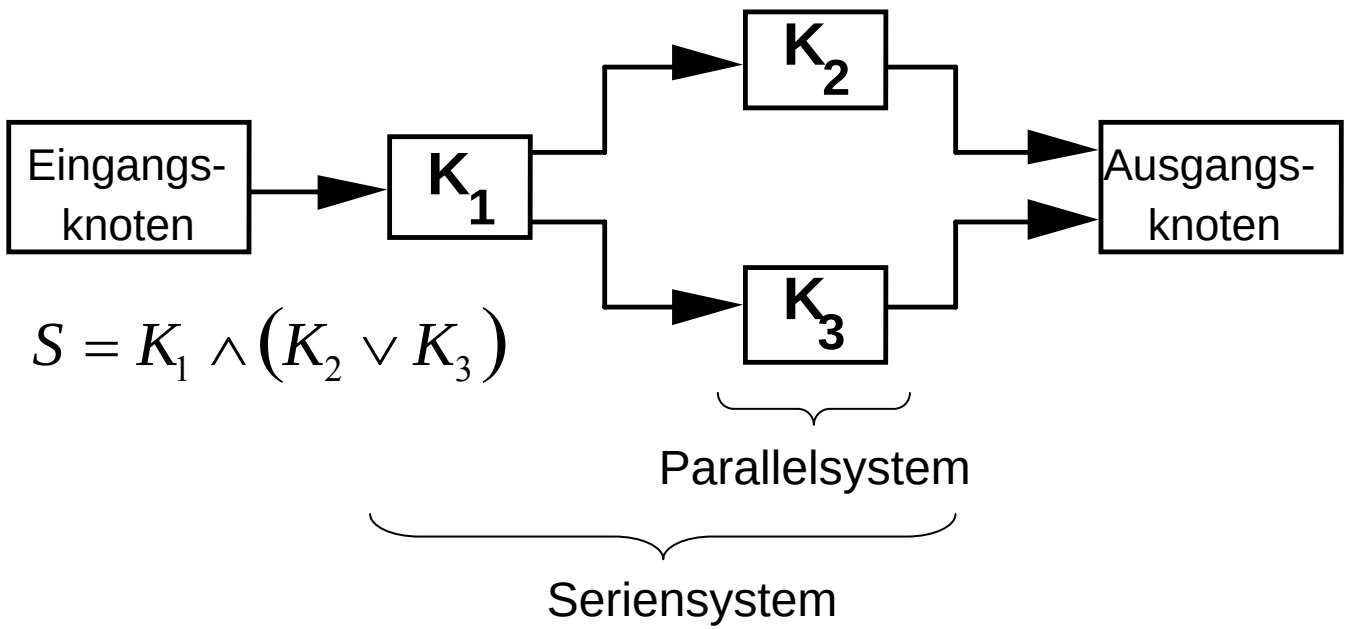
$$\varphi(S) = \sum_{(K_1,\ldots,K_n)\in f^{-1}(wahr)} \varphi(\wedge_{i=1}^{n} K_i)$$

    - (als Oberbegriff für Überlebenswahrscheinlichkeit und Verfügbarkeit)

  - Nichtfunktionswahrscheinlichkeit

$$\varphi(\neg K) = 1 - \varphi(K)$$

- Ausgangspunkt
  - Zuverlässigkeitsblockdiagramm
    - Die Systemfunktion lässt sich grafisch durch ein Zuverlässigkeitsblockdiagramm darstellen:
      - Gerichteter Graph mit einem Eingangs- und einem Ausgangsknoten



$$S = K_1 \wedge (K_2 \vee K_3)$$

- Zuverlässigkeitskenngrößen
  - Funktionswahrscheinlichkeit eines Seriensystems

$$\varphi(\wedge_{K\in\Lambda}) = \prod_{K\in\Lambda} \varphi(K)$$

  - Funktionswahrscheinlichkeit eines Parallelsystems

$$\varphi(\vee_{K\in\Lambda}) = \sum_{\varnothing\neq A\in\Lambda} (-1)^{1+\#A} \cdot \varphi(\wedge_{K\in A} K)$$

- Zuverlässigkeitsverbesserung
  - Für ein System $S = K_1 \vee K_2$
    - Gilt demnach:

$$\varphi(S) = \varphi(K_1 \vee K_2) = \varphi(K_1) + \varphi(K_2) - \varphi(K_1 \wedge K_2)$$

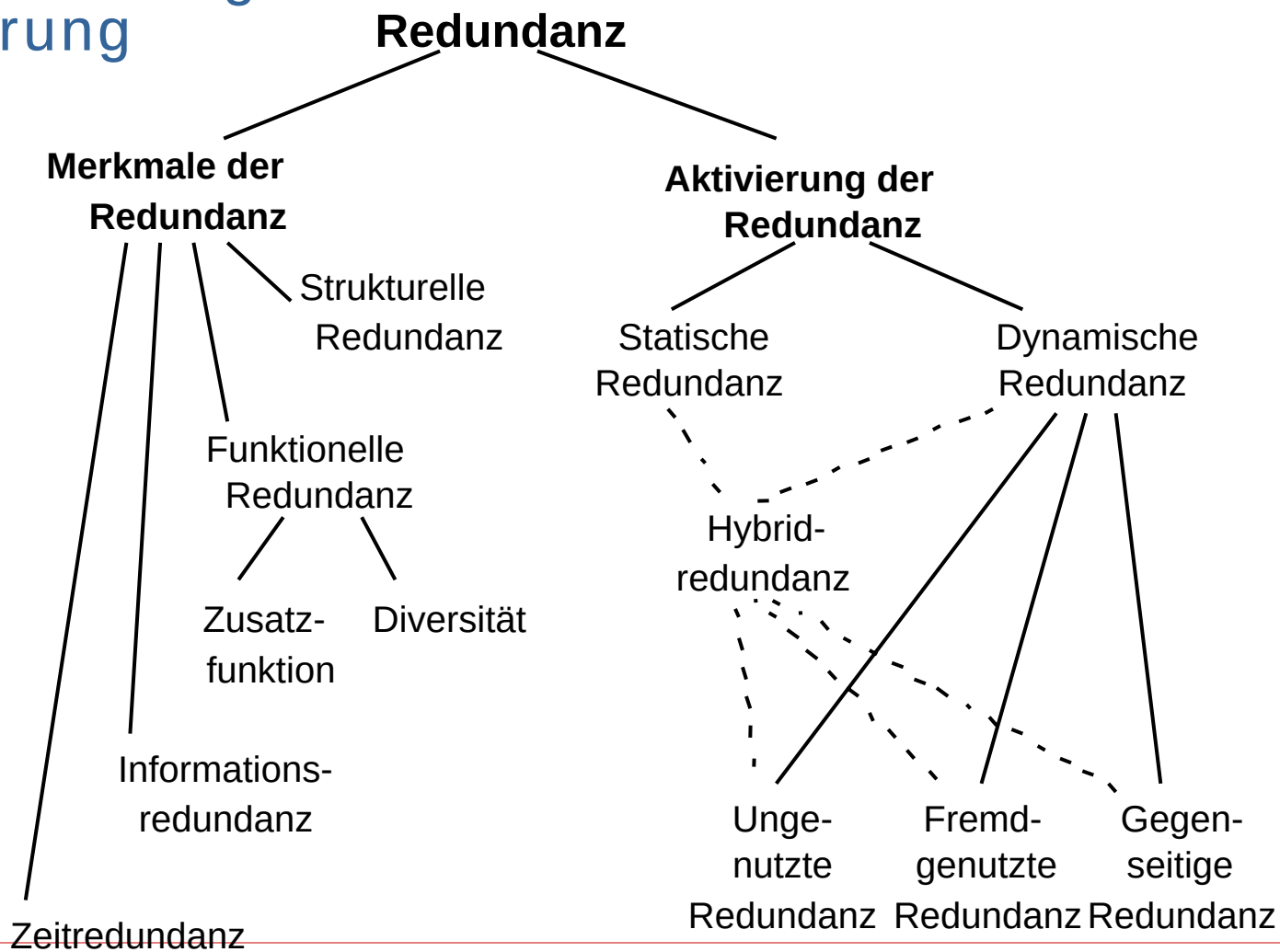
    - Mit bedingten Wahrscheinlichkeiten erhält man:

$$\varphi(S) = \varphi(K) \cdot \varphi(S \mid K) + \varphi(\neg K) \cdot \varphi(S \mid \neg K)$$

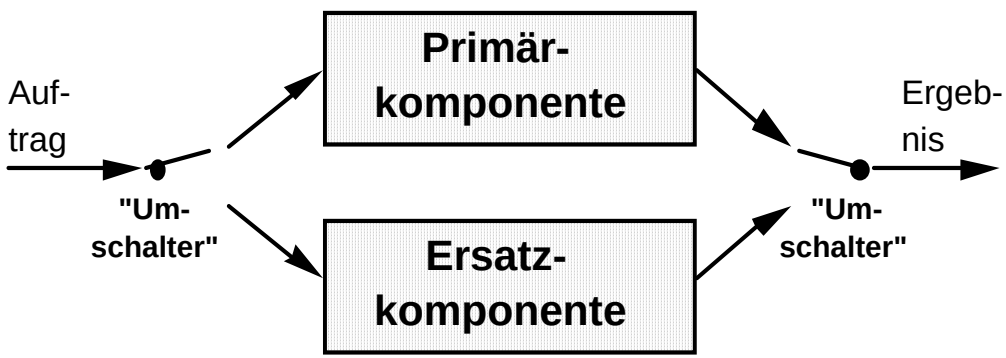
Verbesserungsfaktor:

$$\Phi_{S_1 \to S_2} = \frac{\varphi(\neg S_1)}{\varphi(\neg S_2)} = \frac{1 - \varphi(S_1)}{1 - \varphi(S_2)}$$

- Redundanz
  - Unterscheidung nach ihren Merkmalen und ihrer Aktivierung

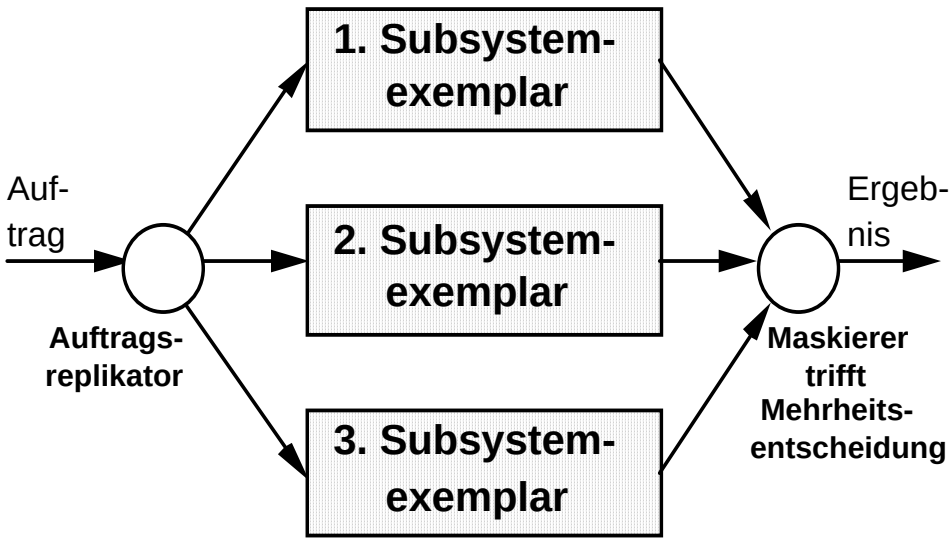


- Dynamische Redundanz (dynamic redundancy)
  - bezeichnet das Vorhandensein von redundanten Mitteln, die erst nach Auftreten eines Fehlers aktiviert werden, um eine ausgefallene Nutzfunktion zu erbringen.
  - Typisch für dynamische strukturelle Redundanz ist die Unterscheidung in Primär- und Ersatzkomponenten (bzw. Sekundär- oder Reservekomponenten).
  - Grundstruktur eines dynamisch strukturell redundanten Systems

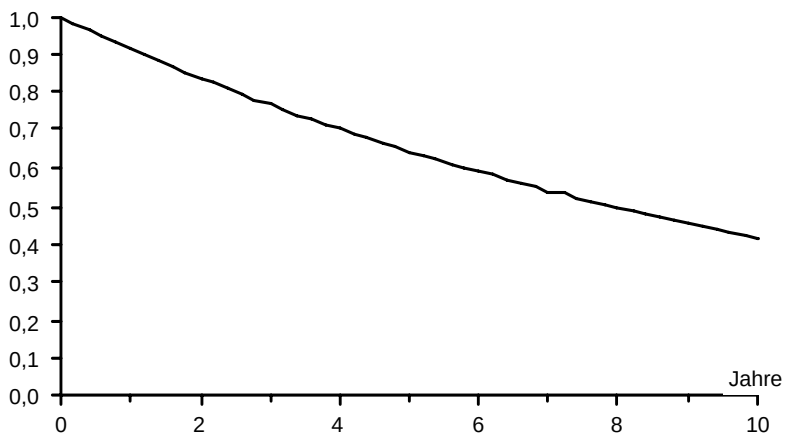


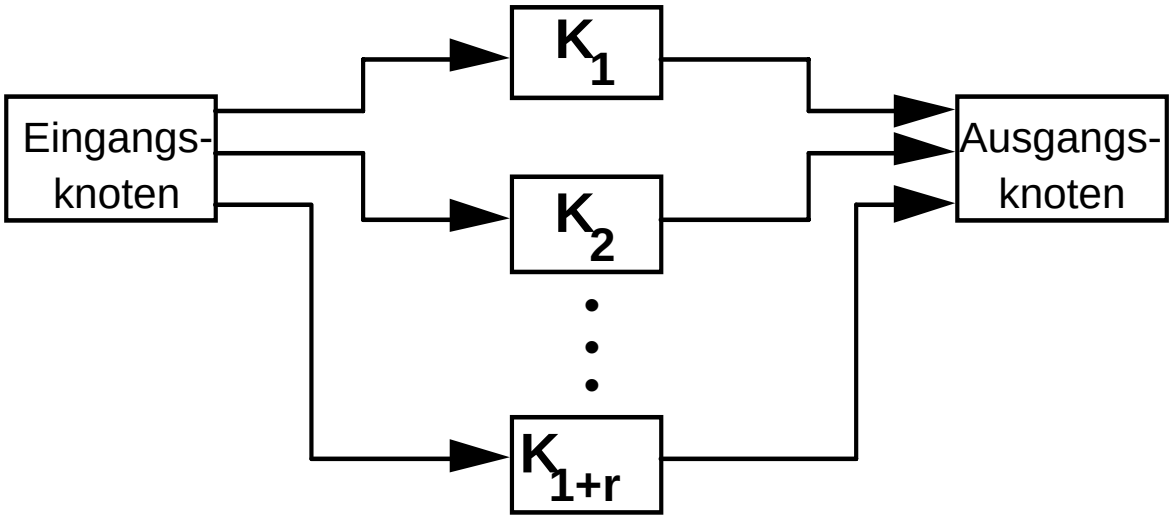
- Dynamische Redundanz (dynamic redundancy)
  - Bevor Ersatzkomponenten aktiviert werden, lassen diese sich auf eine der folgenden Arten verwenden:
  - Ungenutzte Redundanz
    - Ersatzkomponenten führen keine sonstigen Funktionen aus und bleiben bis zur fehlerbedingten Aktivierung passiv.
  - fremdgenutzte Redundanz:
    - Ersatzkomponenten erbringen nur Funktionen, die nicht zum betreffenden Subsystem gehören und im Fehlerfall bei niedrigerer Priorisierung ggf. verdrängt werden.
  - gegenseitige Redundanz:
    - Ersatzkomponenten erbringen die von einer anderen Komponente zu unterstützenden Funktionen, die Komponenten stehen sich gegenseitig als Reserve zur Verfügung.
      Dies ermöglicht einen abgestuften Leistungsabfall (graceful degradation).

- Statische Redundanz (*static redundancy*)
  - bezeichnet das Vorhandensein von redundanten Mitteln, die während des gesamten Einsatzzeitraums die gleiche Nutzfunktion erbringen.
  - Beispiel der statischen strukturellen Redundanz: *n*-von-*m*-System
  - 2-von-3-System:



- Verbesserung der Zuverlässigkeit durch Redundanz
  - Nichtredundantes Einfachsystem: S1=K1
  - Bei konstanter Ausfallrate beschreibt man die Zeitabhängigkeit der Funktionswahrscheinlichkeit φ (S_1, t) durch eine Exponentialverteilung
    - mit $z(t)=\lambda$, $\varphi(S_1, t) = e^{-\lambda \cdot t}$.
  - Beispiel:
    - Funktionswahrscheinlichkeit φ (S_1, t) mit $\lambda=10^{-5/h}$





Systemfunktion

$$S_{1+r} = K_1 \vee \ldots \vee K_{1+r}$$

Funktionswahrscheinlichkeit

$$\varphi(S_{1+r}, t) = 1 - \prod_{i=1}^{1+r} (1 - \varphi(K_i, t))$$

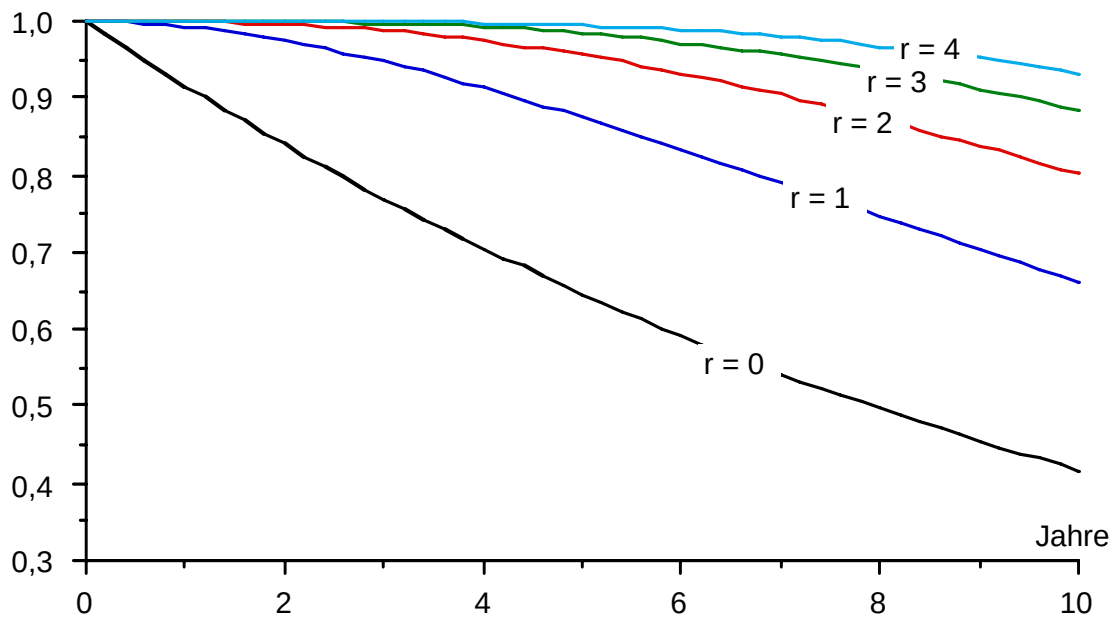
gleiche konstante Ausfallrate λ

$$\varphi(S_{1+r}, t) = 1 - (1 - e^{-\lambda \cdot t})^{1+r}$$

Zuverlässigkeitsverbesserung

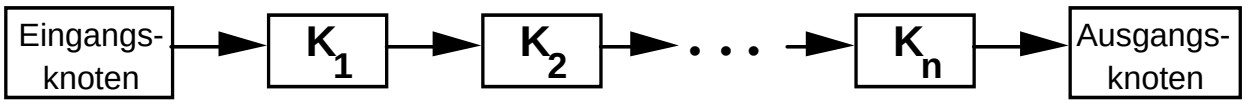
$$\Phi_{S1 \to S1+r} = (1 - e^{-\lambda \cdot t})^{-r}.$$

Funktionswahrscheinlichkeit für Parallelsystem



Annahme einer Komponentenausfallrate von $\lambda = 10^{-5}/\mathrm{h}$

Seriensystem (Nichtredundantes Mehrfachsystem)



Seriensystem

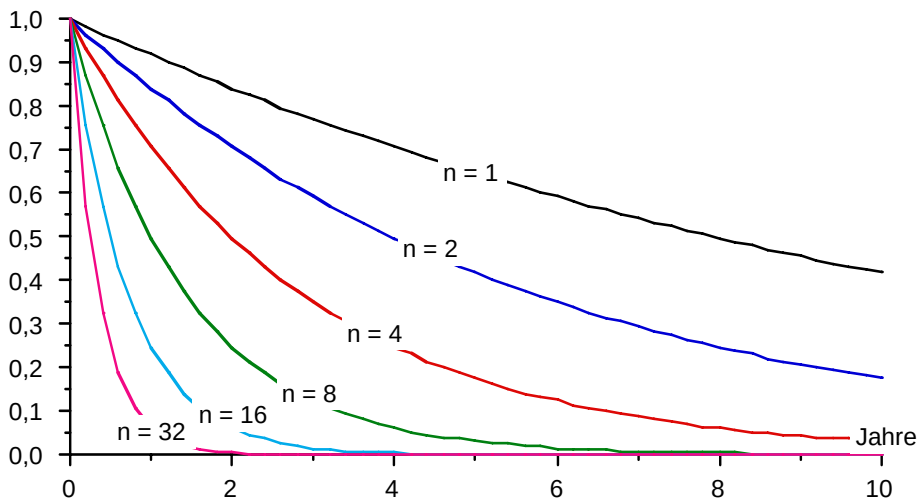
$$S_n = K_1 \wedge \ldots \wedge K_n$$

Zuverlässigkeit

$$\varphi(S_n, t) = \prod_{i=1}^{n} \varphi(K_i, t)$$

Funktionswahrschein-lichkeit $\varphi\,(S_n, t)$

für $\lambda = 10^{-5}/h$



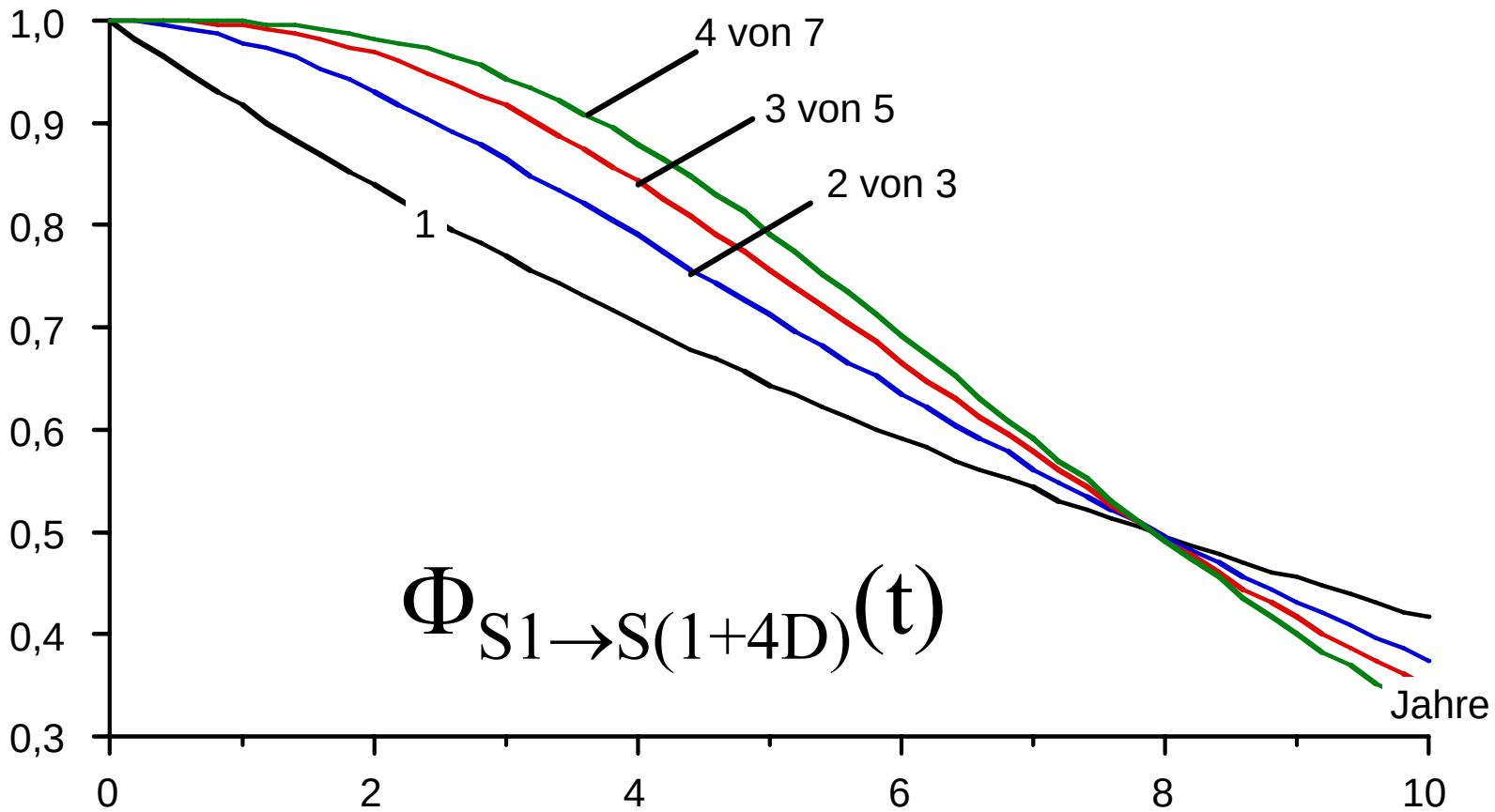
Ist die Fehlererfassung zu gering oder verbieten sich wiederholte Berechnungen wegen den geforderten maximalen Antwortzeiten, so kann statische Redundanz eingesetzt werden.

Dabei führen mehrere Komponenten die gleiche Berechnung aus, um anschließend die errechneten Ergebnisse zu vergleichen und ein mehrheitliches auszuwählen.

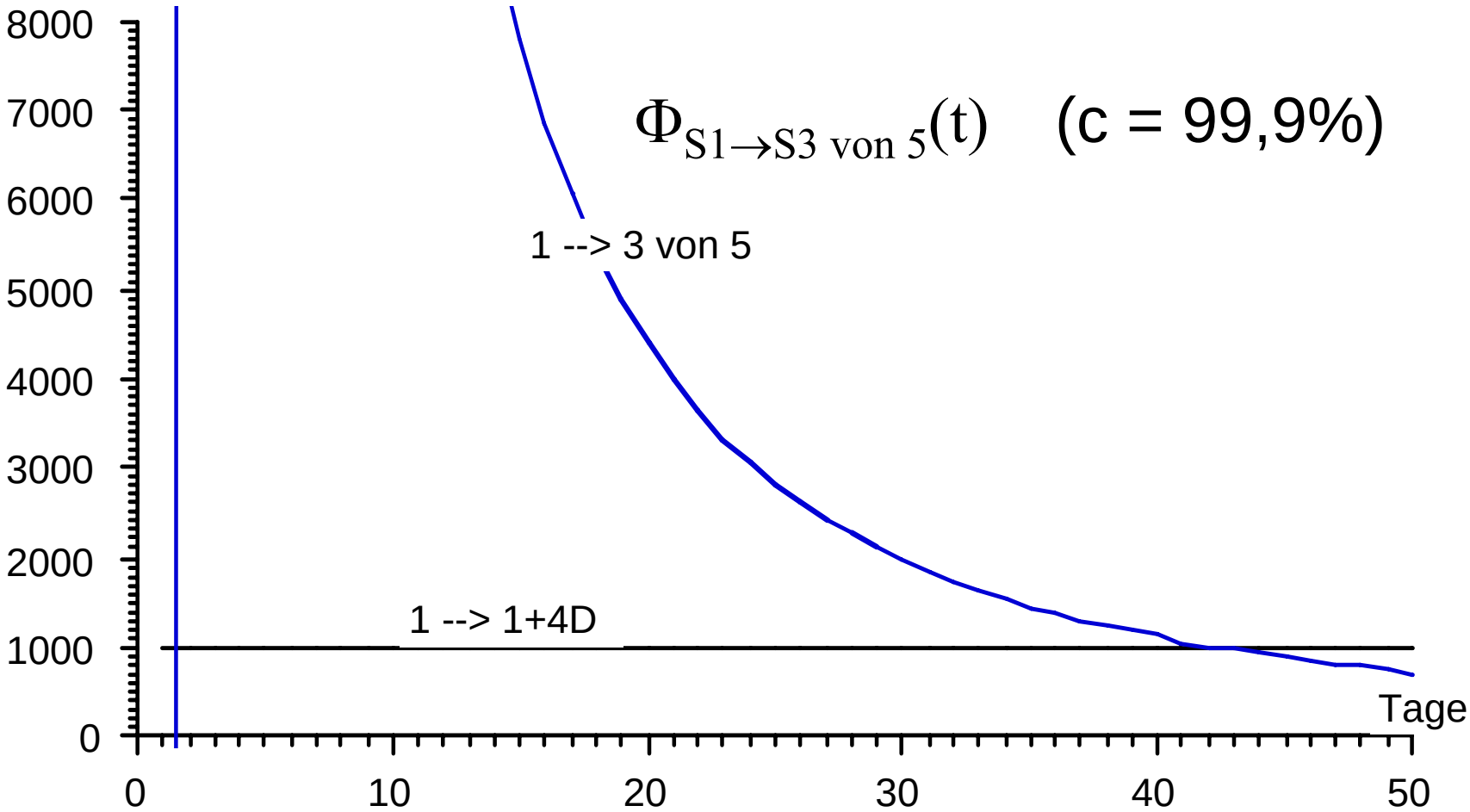
Bis zu f fehlerhafte Komponenten können überstimmt werden, wenn mindestens n=f+1 fehlerfreie, insgesamt also m=2·f+1 Komponenten vorhanden sind.

$$S_{m\,von\,m} = \bigvee_{1 \le i_1 < \ldots < i_n \le m} K_{i_1} \wedge \ldots \wedge K_{i_n}$$

Statisch redundantes System



Statisch redundantes System



Maskierer M trifft in einem statisch redundanten System die Mehrheitsentscheidung. Dies verändert Systemfunktion und Zuverlässigkeit:

$$S_{n\,von\,mM} = M \wedge \left(\bigvee_{1 \le i_1 < \ldots < i_n \le n+r} K_{i_1} \wedge \ldots \wedge K_{i_n} \right)$$

Funktionswahrscheinlichkeit ist durch die des "Zuverlässigkeitsengpasses" M beschränkt:

$$\varphi\,(S_{n\ von\ mM}) \le \varphi\,(M).$$

- **1. Alternative: Verbesserung der Komponenten**
  - **+einfacher Ansatz zur Verbesserung**
  - **+bis zu einer gegebenen Grenze kostengünstig**
  - **–ab dieser Grenze steigen die Kosten überproportional**
  - **–Lösung oft nicht leistungsfähig und zuverlässig zugleich**

- **2. Alternative: Zusätzliche Komponenten**
  - **–bei Verdoppelung oder Vervierfachung hoher Aufwand**
  - **+Prüfzeichen sind ein effizientes Mittel gegen spezielle Fehler**
  - **– Ansatz ist unflexibel, da meist voller Zusatzaufwand notwendig**

- **3. Alternative: Zusätzliche Subsysteme (zusätzliche Rechner)**
  - **+alle Rechner gleich, keine Spezialrechner**
  - **+flexible Lastverteilung (und Umverteilung bei Fehler) möglich**
  - **–überproportionale Kosten**
  - **–Aufwand zur Herstellung der aktuellen Zustandsinformation**
  - **–Aufwand zur Vermeidung von Inkonsistenzen**

Zusammenfassung

- Grundlagen
- Prozessorarchitekturen
- Multiprozessorsysteme
- Fehlertoleranz

- Stoff der Klausur:
  - Vorlesung
  - Übung