Rechnerstrukturen

Vorlesung im Sommersemester 2009

Prof. Dr. Wolfgang Karl

Universität Karlsruhe (TH)

Fakultät für Informatik

Institut für Technische Informatik

- **Gastvorlesung**
  - **Termin: 22.7.2009, 9:45-11:15 Uhr**
  - **Carl Mayer, IBM Labor Böblingen**
  - **Thema: RAS mit System z**

- **Kapitel 5: Fehlertoleranz, Zuverlässigkeit**

5.1: Grundlagen

Zuverlässigkeit und Fehlertoleranz

- 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

Zuverlässigkeit und Fehlertoleranz

- 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.

Zuverlässigkeit und Fehlertoleranz

- 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

Zuverlässigkeit und Fehlertoleranz

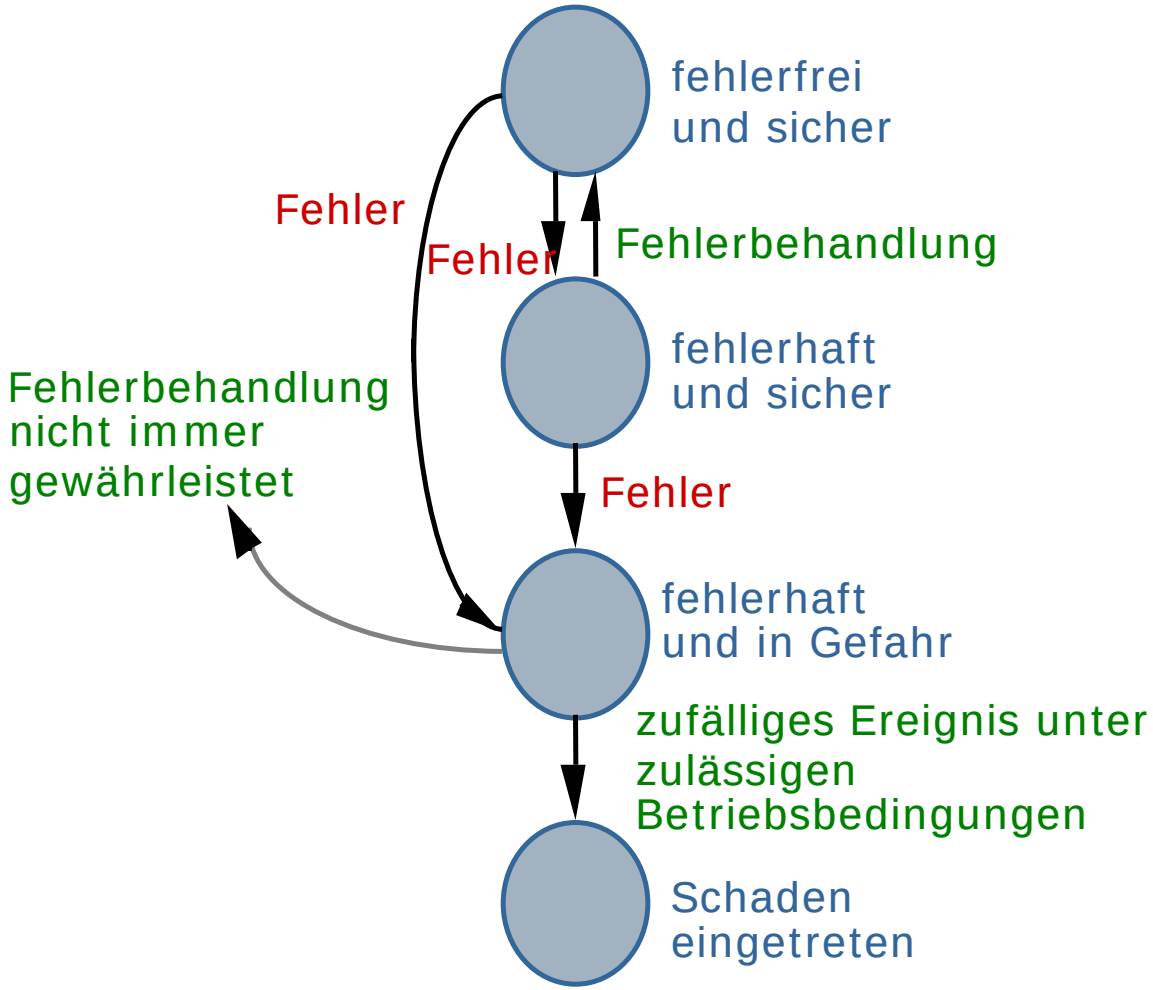
- Begriffsbildung
  - Ausfall
    - Hardwarekomponenten (Chips, Hintergrundspeicher)
    - Software, durch Programmfehler
    - Menschliche Eingriffe
  - Sicherheitsrelevante Anwendungen erfordern eine hohe Verfügbarkeit

Zuverlässigkeit und Fehlertoleranz

- 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?

Zuverlässigkeit und Fehlertoleranz

- Fehler



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

Zuverlässigkeit und Fehlertoleranz

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

Zuverlässigkeit und Fehlertoleranz

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

Zuverlässigkeit und Fehlertoleranz

- 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

Zuverlässigkeit und Fehlertoleranz

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

Zuverlässigkeit und Fehlertoleranz

- 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

Zuverlässigkeit und Fehlertoleranz

- 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

Zuverlässigkeit und Fehlertoleranz

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

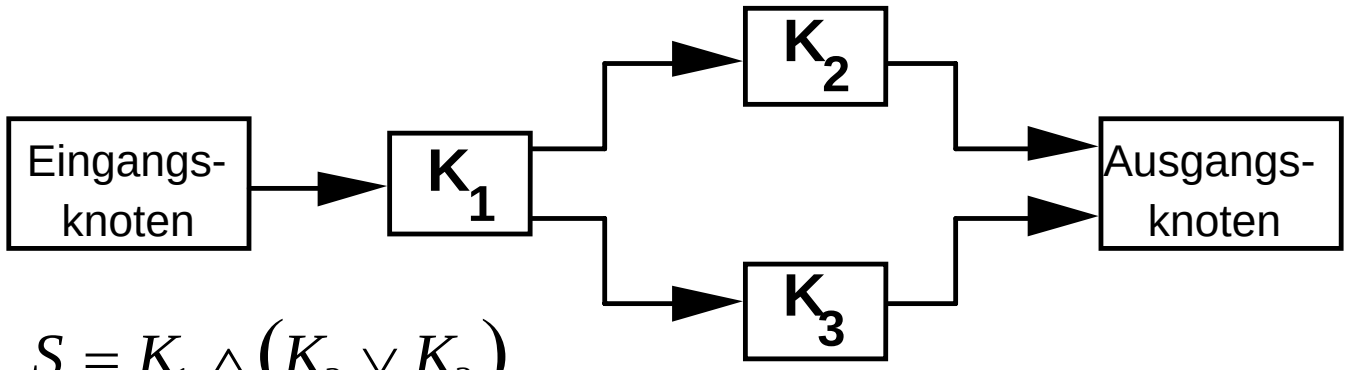
Zuverlässigkeit und Fehlertoleranz

- 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 zu tolerierenden Fehler wird bezüglich einer Fehlerannahme formuliert
  - Festlegung:
    - wie viele Einzelfehlerbereiche können gleichzeitig fehlerhaft werden
    - Welche Fehlfunktionen sind zu behandeln

Zuverlässigkeit und Fehlertoleranz

- 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



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

Zuverlässigkeit und Fehlertoleranz

- 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* $f_D(t)$

Zuverlässigkeit und Fehlertoleranz

- Zuverlässigkeitskenngrößen
  - Zuverlässigkeit, Sicherheit einer Rechensystems
    - Verteilungsfunktionen

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

– mit x = L, B, D

Zuverlässigkeit und Fehlertoleranz

- 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 $\lim_{t\to\infty} R(t) = 0$

Zuverlässigkeit und Fehlertoleranz

- 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

Zuverlässigkeit und Fehlertoleranz

- 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.

Zuverlässigkeit und Fehlertoleranz

- 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

Zuverlässigkeit und Fehlertoleranz

- Zuverlässigkeitskenngrößen
  - Funktionswahrscheinlichkeit der Komponenten und Systemfunktionen

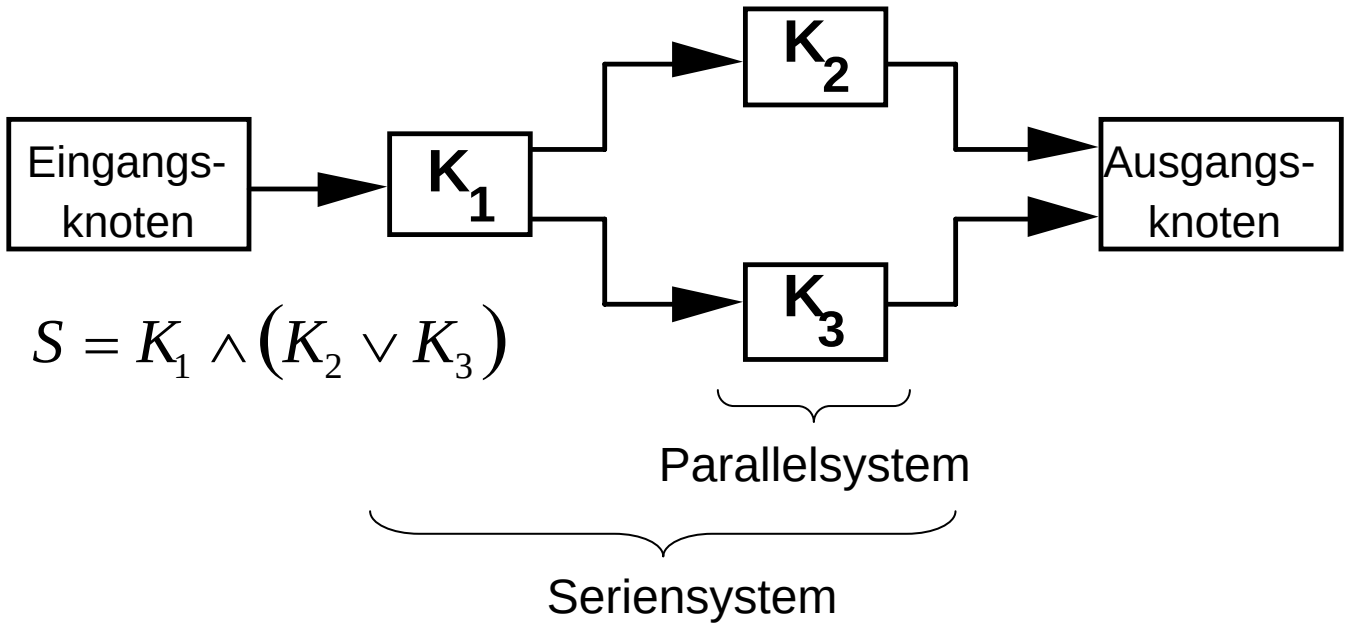
$$\varphi(S) = \sum_{(K_1,\ldots,K_n)\in f^{-1}(wahr)} \varphi(\wedge_{i=1}^{n} K_i)$$

    - (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



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

Zuverlässigkeit und Fehlertoleranz

- Zuverlässigkeitskenngrößen
  - Funktionswahrscheinlichkeit eines Seriensystems

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

  - Funktionswahrscheinlichkeit eines Parallelsystems

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

Zuverlässigkeit und Fehlertoleranz

- 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:

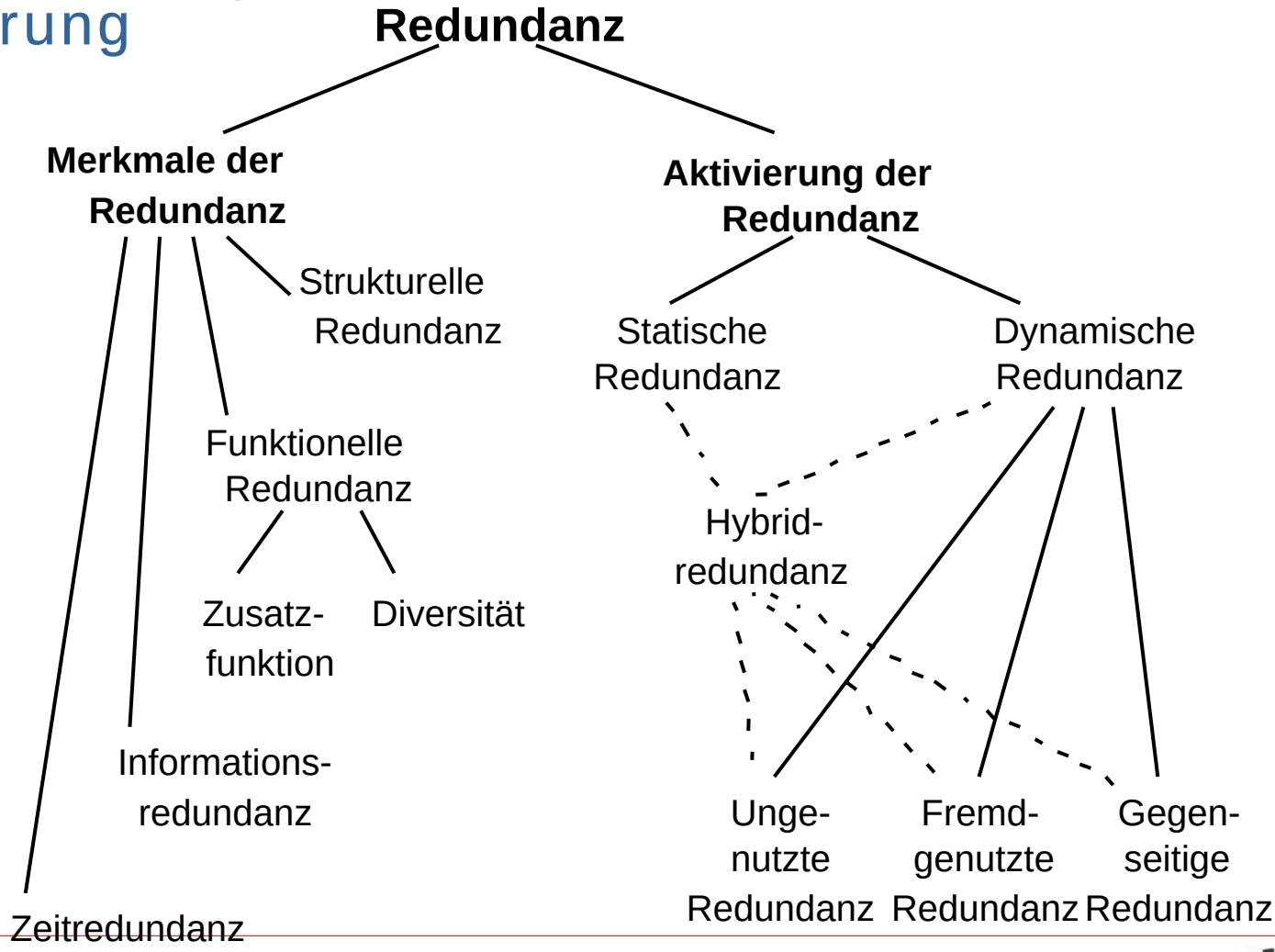
$$\varphi(S) = \varphi(K) \cdot \varphi(S|K) + \varphi(\neg K) \cdot \varphi(S|\neg K)$$

Verbesserungsfaktor:

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

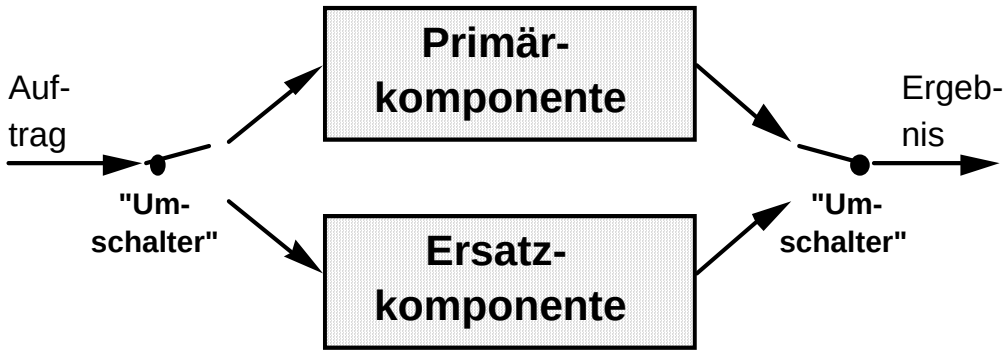
Zuverlässigkeit und Fehlertoleranz

- Redundanz
  - Unterscheidung nach ihren Merkmalen und ihrer Aktivierung



Zuverlässigkeit und Fehlertoleranz

- 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

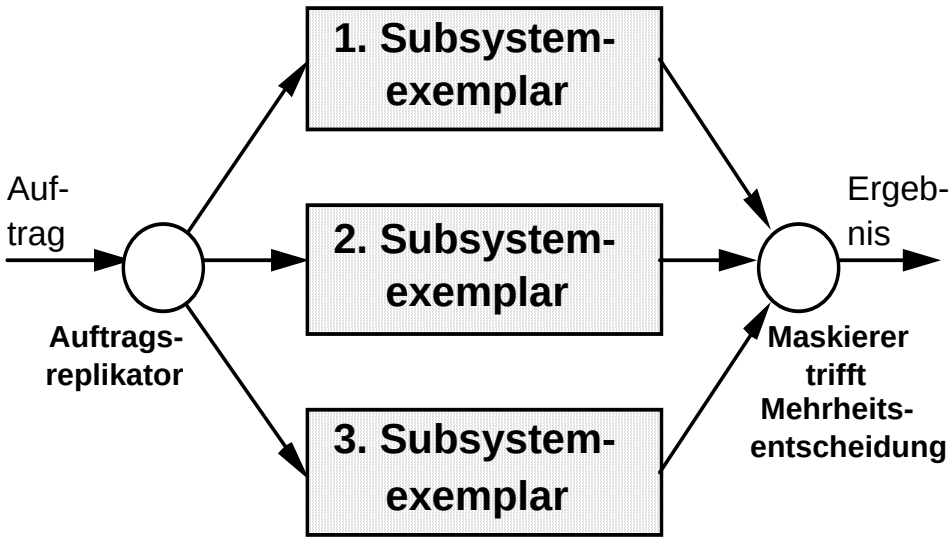


Zuverlässigkeit und Fehlertoleranz

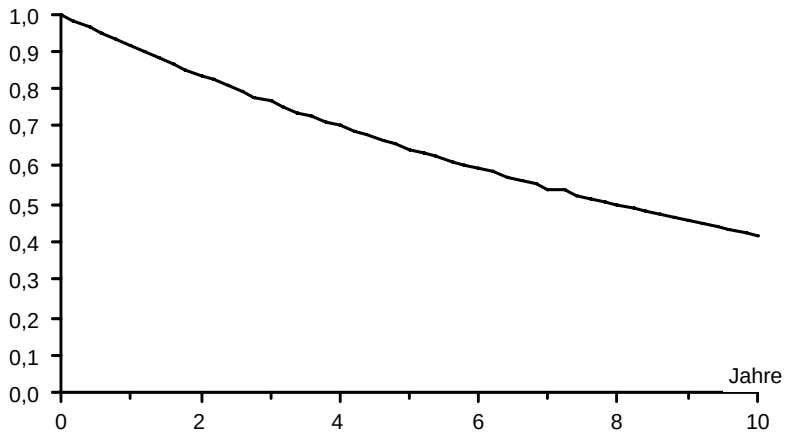
- Dynamische Redundanz (dynamic redundancy)
  - Bevor Ersatzkomponenten aktiviert werden, lassen diese sich auf eine der folgenden Arten verwenden:
  - Ungenutzte Redundanz
    - Ersatzkomponenten führen keine sonstigen Funktionen aus und bleiben bis zur fehlerbedingten Aktivierung passiv.
  - fremdgenutzte 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).

Fehlertoleranz

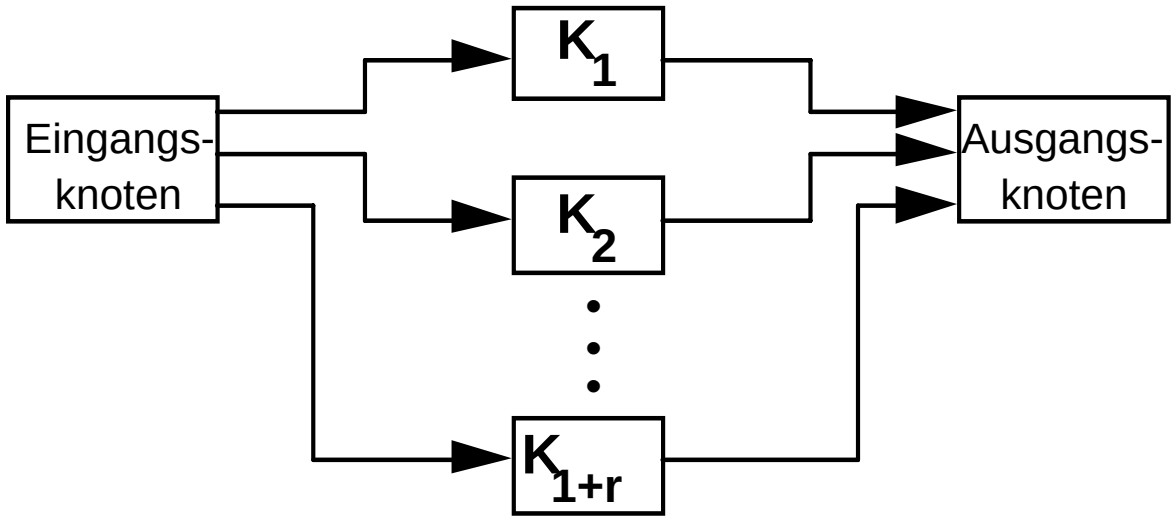
- Statische Redundanz (*static redundancy*)
  - bezeichnet das Vorhandensein von redundanten Mitteln, die während des gesamten Einsatzzeitraums die gleiche Nutzfunktion erbringen.
  - Beispiel der statischen strukturellen Redundanz: *n*-von-*m*-System
  - 2-von-3-System:



- 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 $\lambda=10^{-5/h}$



Parallelsystem (Einfachsystem mit ungenutzter oder fremdgenutzter Redundanz)



Systemfunktion

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

Funktionswahrscheinlichkeit

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

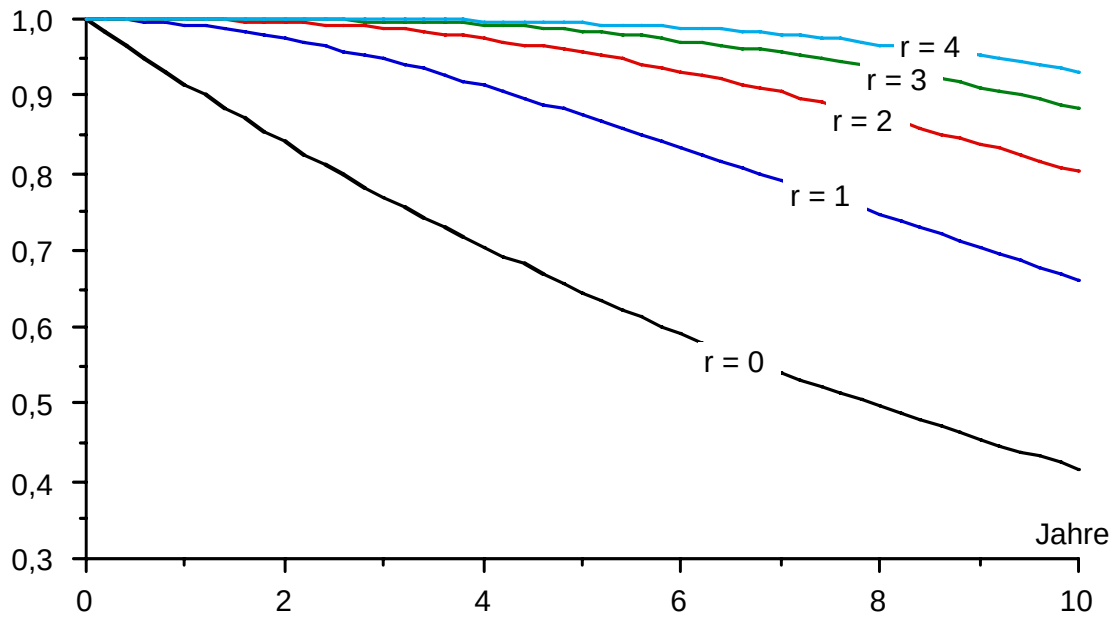
gleiche konstante Ausfallrate λ

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

Zuverlässigkeitsverbesserung

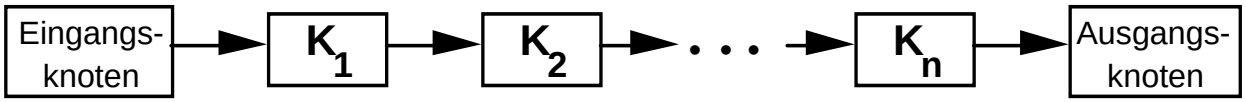
$$\Phi_{S1 \to S1+r} = (1 - e^{-\lambda \cdot t})^{-r}.$$

Funktionswahrscheinlichkeit für Parallelsystem



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

Seriensystem (Nichtredundantes Mehrfachsystem)



Seriensystem

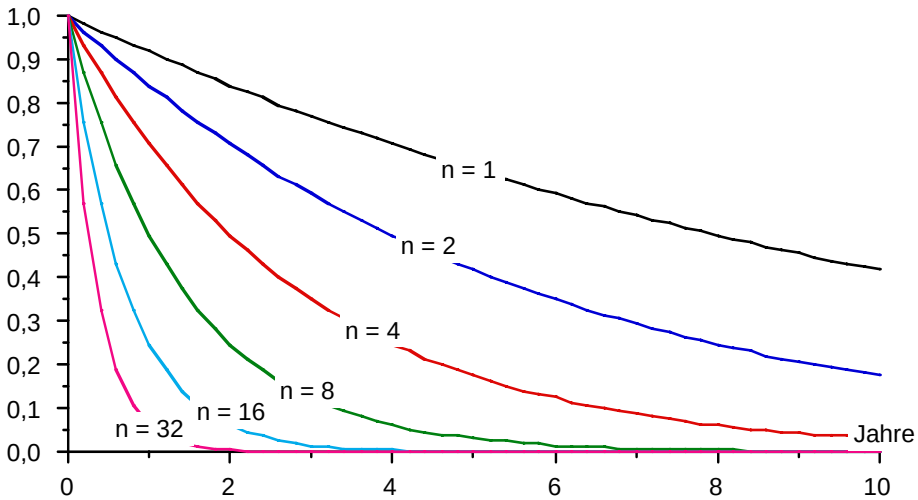
$$S_n = K_1 \wedge ... \wedge K_n$$

Zuverlässigkeit

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

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

für $\lambda = 10^{-5}/h$



Statisch redundantes System

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.

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

Bewertung der Zuverlässigkeitsverbesserung

- **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**

- **Kapitel 4: Vektorverarbeitung**

4.1: Grundlagen

- Wie arbeiten Vektorprozessoren?
  - Ein Beispiel:
    - Y = a * X + Y,
      - wobei X und Y Vektoren sind und a eine Konstante ist.
      - Bildet die innere Schleife des Linpack-Benchmarks und wird auch als SAXPY (Single Precision a x X plus Y) bzw. DAXPY (Double Precision a x X plus Y) bezeichnet.
      - Für alle i, i = Anzahl der Elemente des Vektors X und des Vektors Y, ergibt sich der neue Wert für Y[i] aus der Multiplikation von a mit X[i] und der Addition des Zwischenergebnisses mit Y[i]

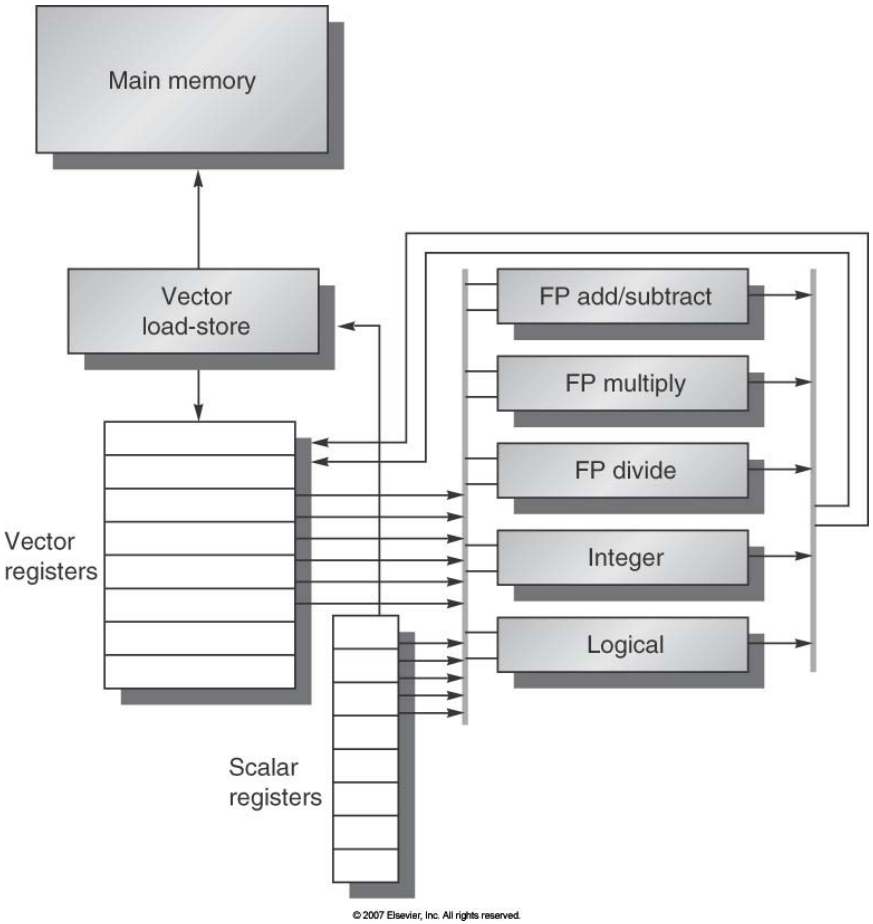
- Wie arbeiten Vektorprozessoren?
  - Ein Beispiel:
    - Y = a * X + Y,
  - In MIPS-Notation

```
                                          ;in Rx bzw. Ry Startadresse von X,Y
        L.D       F0,a                    ;load scalar a
        DADDIU    R4,Rx,#512              ;last address to load
Loop:   L.D       F2,0(Rx)                ;load X(i)
        MUL.D     F2,F2,F0                ;a × X(i)
        L.D       F4,0(Ry)                ;load Y(i)
        ADD.D     F4,F4,F2                ;a × X(i) + Y(i)
        S.D       0(Ry),F4                ;store into Y(i)
        DADDIU    Rx,Rx,#8                ;increment index to X
        DADDIU    Ry,Ry,#8                ;increment index to Y
        DSUBU     R20,R4,Rx               ;compute bound
        BNEZ      R20,Loop                ;check if done
```

- Wie arbeiten Vektorprozessoren?
  - Ein Beispiel:
    - Y = a * X + Y
  - Analyse:
    - Die Schleife wird ungefähr 600 Mal durchlaufen.
    - Hoher Aufwand:
      - In einem Schleifendurchlauf werden jeweils die Elemente von X und Y addiert und nach der Berechnung wird das Ergebnis gespeichert. Die Adressen werden aktualisiert und das Abbruchkriterium wird geprüft.
      - Beachtung von Konflikten aufgrund von Datenabhängigkeiten
      - Beachtung von Multizyklus-Operationen
      - Pipeline-Stalls können durch Compiler-Optimierungen wie Loop-Unrolling und Software-Pipelining reduziert werden

- Wie arbeiten Vektorprozessoren?
  - Ein Beispiel:
    - Y = a * X + Y
  - Idee der Vektor-Prozessoren:
    - SIMD-Verarbeitung
      - Verarbeitung von Vektoren in einem Rechenwerk mit Pipeline-artig aufgebauten Funktionseinheiten
      - Bereitstellung von Vektoroperationen

- Wie arbeiten Vektorprozessoren?
  - Idee der Vektor-Prozessoren:

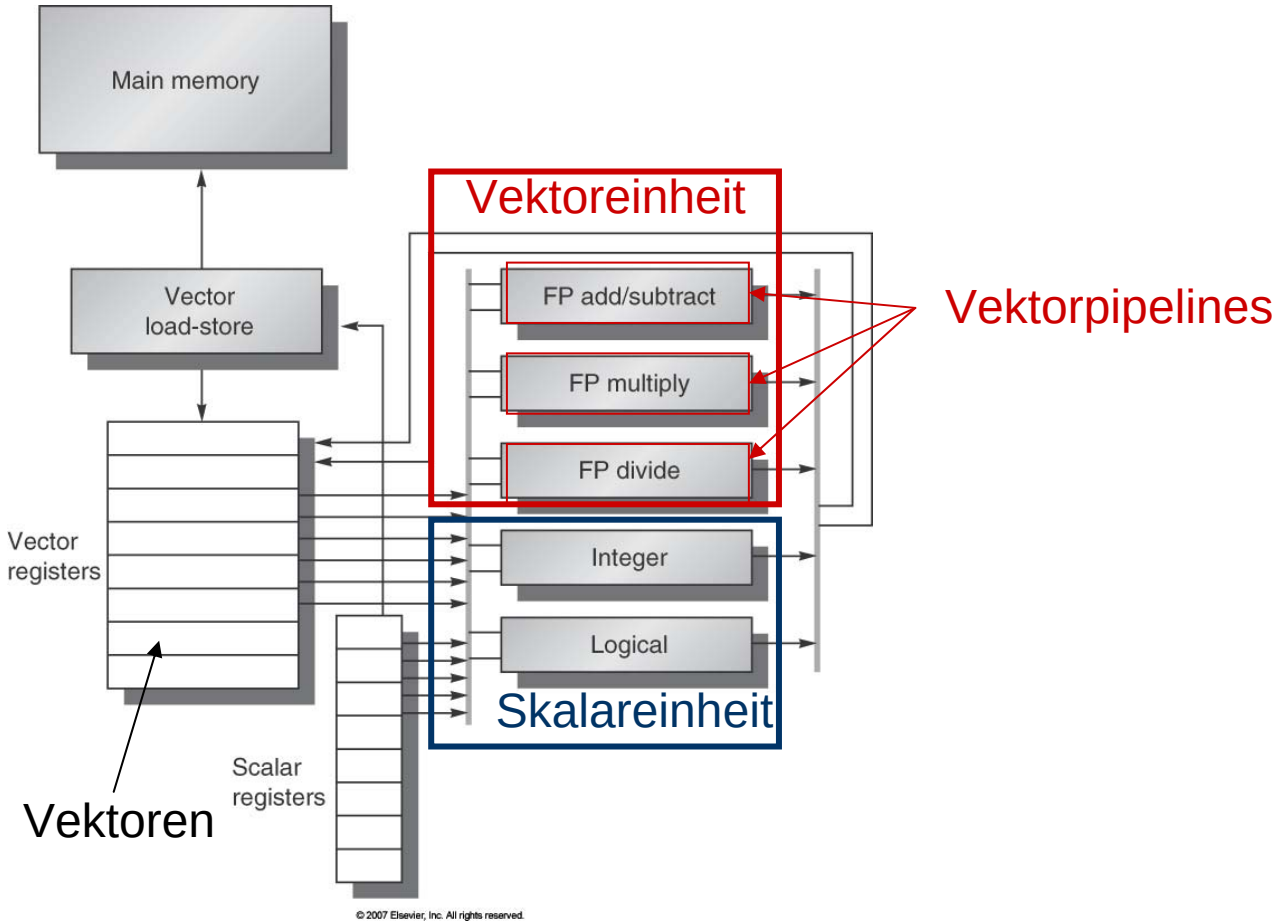


Beispielprogramm mit Vektoroperationen:

```
L.D      F0,a       ;load scalar a
LV       V1,Rx      ;load vector X
MULVS.D  V2,V1,F0   ;vector-scalar multiply
LV       V3,Ry      ;load vector Y
ADDV.D   V4,V2,V3   ;add
SV       Ry,V4      ;store the result
```

- Vektorprozessor:
  - Unter einem Vektorprozessor (Vektorrechner) versteht man einen Rechner mit pipelineartig aufgebautem/n Rechenwerk/en zur Verarbeitung von Arrays von Gleitpunktzahlen.
  - Vektor = Array (Feld) von Gleitpunktzahlen
  - Jeder Vektorrechner besitzt in seinem Rechenwerk einen Satz von Vektorpipelines. Dieses wird als Vektoreinheit bezeichnet.
  - Im Gegensatz zur Vektorverarbeitung wird die Verknüpfung einzelner Operanden als Skalarverarbeitung bezeichnet.
  - Ein Vektorrechner enthält neben der Vektoreinheit auch noch eine oder mehrere Skalareinheiten. Dort werden die skalaren Befehle ausgeführt, d.h. Befehle, die nicht auf ganze Vektoren angewendet werden sollen.
  - Die Vektoreinheit und die Skalareinheit(en) können parallel zueinander arbeiten, d.h. Vektorbefehle und Skalarbefehle können parallel ausgeführt werden.

- Vektorprozessoren
  - Beispiel



- Vektorprozessoren
  - Die Pipeline-Verarbeitung wird mit einem Vektorbefehl für zwei Felder von Gleitpunktzahlen durchgeführt.
  - Die bei den Gleitpunkteinheiten skalarer Prozessoren nötigen Adressrechnungen entfallen.

Beispielprogramm mit Vektoroperationen:

```
L.D       F0,a       ;load scalar a
LV        V1,Rx      ;load vector X
MULVS.D   V2,V1,F0   ;vector-scalar multiply
LV        V3,Ry      ;load vector Y
ADDV.D    V4,V2,V3   ;add
SV        Ry,V4      ;store the result
```

- Vektorprozessoren
  - Vektorbefehle

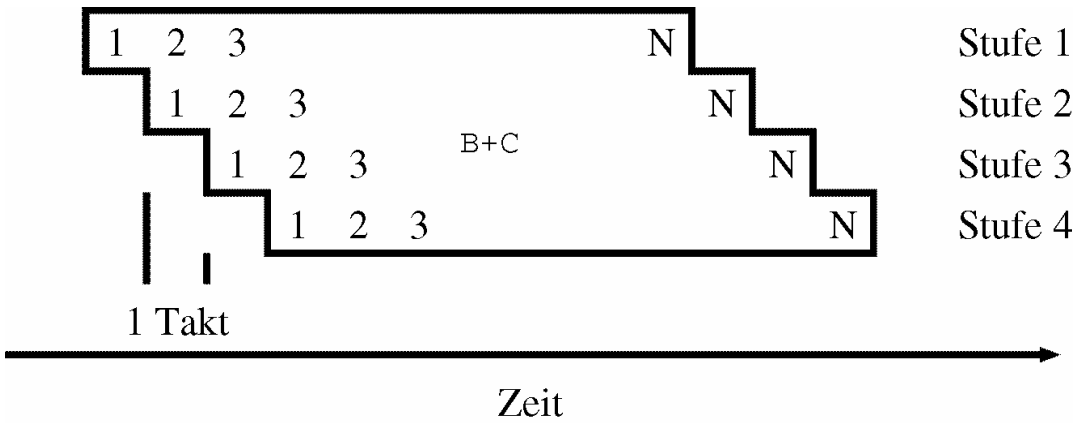
Instruktion	Operanden	Funktion
ADDV.D ADDVS.D	V1,V2,V3 V1,V2,F0	Add elements of V2 and V3, then put each result in V1. Add F0 to each element of V2 , then put each result in V1.
SUBV.D SUBVS.D SUBSV.D	V1,V2,V3 V1,V2,F0 V1,F0,V2	Subtract elements of V3 from V2, then put each result in V1. Subtract F0 from elements of V2, then put each result in V1. Subtract elements of V2 from F0, then put each result in V1.
MULV.D MULVS.D	V1,V2,V3 V1,V2,F0	Multiply elements of V2 and V3, then put each result in V1. Multiply each element of V2 by F0, then put each result in V1.
DIVV.D DIVVS.D DIVSV.D	V1,V2,V3 V1,V2,F0 V1,F0,V2	Divide elements of V2 by V3, then put each result in V1. Divide elements of V2 by F0, then put each result in V1. Divide F0 by elements of V2, then put each result in V1.
LV	V1,R1	Load vector register V1 from memory starting at address R1.
SV	R1,V1	Store vector register V1 into memory starting at address R1.
LVWS	V1,(R1,R2)	Load V1 from address at R1 with stride in R2, i.e., R1+i × R2.
SVWS	(R1,R2),V1	Store V1 from address at R1 with stride in R2, i.e., R1+i × R2.
LVI	V1,(R1+V2)	Load V1 with vector whose elements are at R1+V2(i), i.e., V2 is an index.

- Vektorprozessoren
  - Vektorbefehle

Instruktion	Operanden	Funktion
SVI	(R1+V2),V1	Store V1 to vector whose elements are at R1+V2(i), i.e., V2 is an index.
CVI	V1,R1	Create an index vector by storing the values 0, 1 × R1, 2 × R1,...,63 × R1 into V1.
S--V.D S--VS.D	V1,V2 V1,F0	Compare the elements (EQ, NE, GT, LT, GE, LE) in V1 and V2. If condition is true, put a 1 in the corresponding bit vector; otherwise put 0. Put resulting bit vector in vector-mask register (VM). The instruction S--VS.D performs the same compare but using a scalar value as one operand.
POP	R1,VM	Count the 1s in the vector-mask register and store count in R1.
CVM		Set the vector-mask register to all 1s.
MTC1 MFC1	VLR,R1 R1,VLR	Move contents of R1 to the vector-length register. Move the contents of the vector-length register to R1.
MVTM MVFM	VM,F0 F0,VM	Move contents of F0 to the vector-mask register. Move contents of vector-mask register to F0.

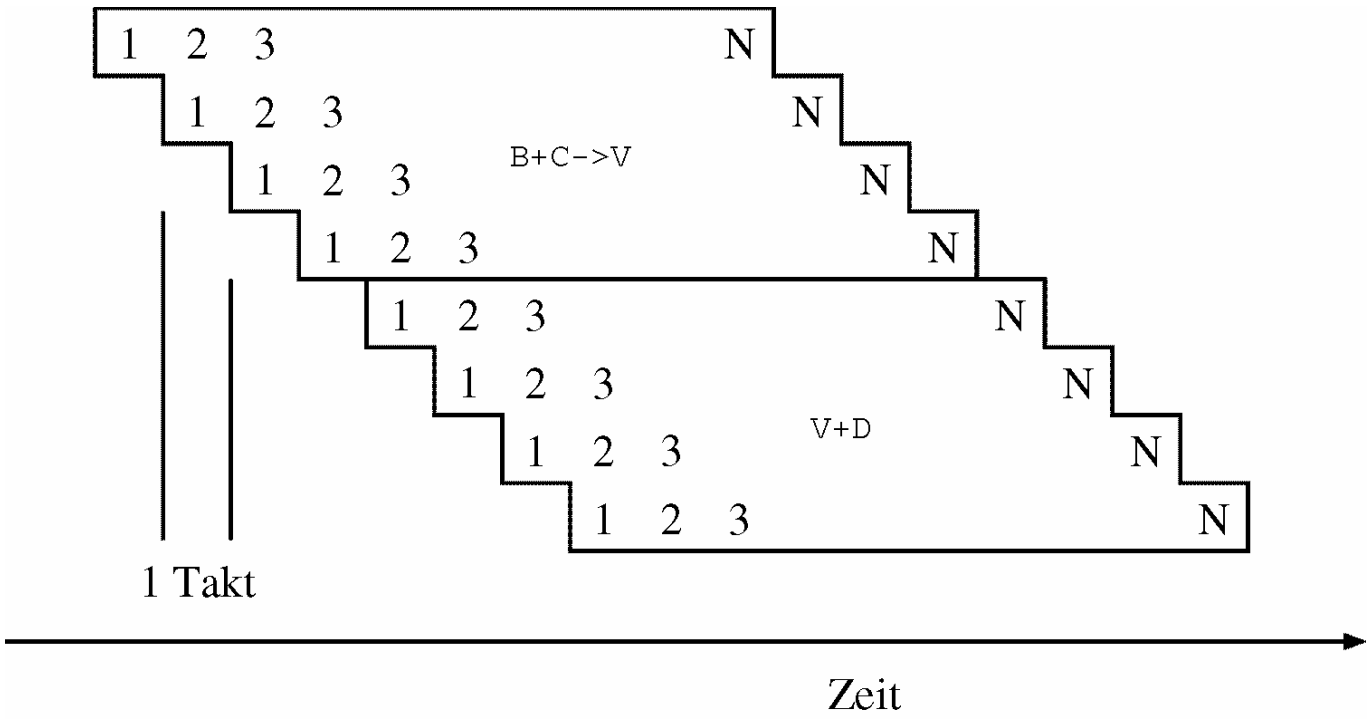
- Vektorprozessoren
  - Pipelining
    - Bei ununterbrochener Arbeit in der Pipeline kann man nach einer gewissen Einschwingzeit bzw. Füllzeit, die man braucht, um die Pipeline zu füllen, mit jedem Pipeline-Takt ein Ergebnis erwarten.
    - Dabei ist die Taktdauer durch die Dauer der längsten Teilverarbeitungszeit zuzüglich der Stufentransferzeit gegeben.
    - Beispiel Gleitkomma-Operationen:
      - Laden eines Paars von Gleitpunktzahlen aus Vektorregister
      - Vergleichen der Exponenten und Verschieben einer Mantisse
      - Addition der ausgerichteten Mantissen
      - Normalisieren des Ergebnisses und Schreiben in Zielregister

- Vektorprozessoren
  - Pipelining
    - Beispiel:
      - B[i] + C[i] mit i= 1,2,...,N

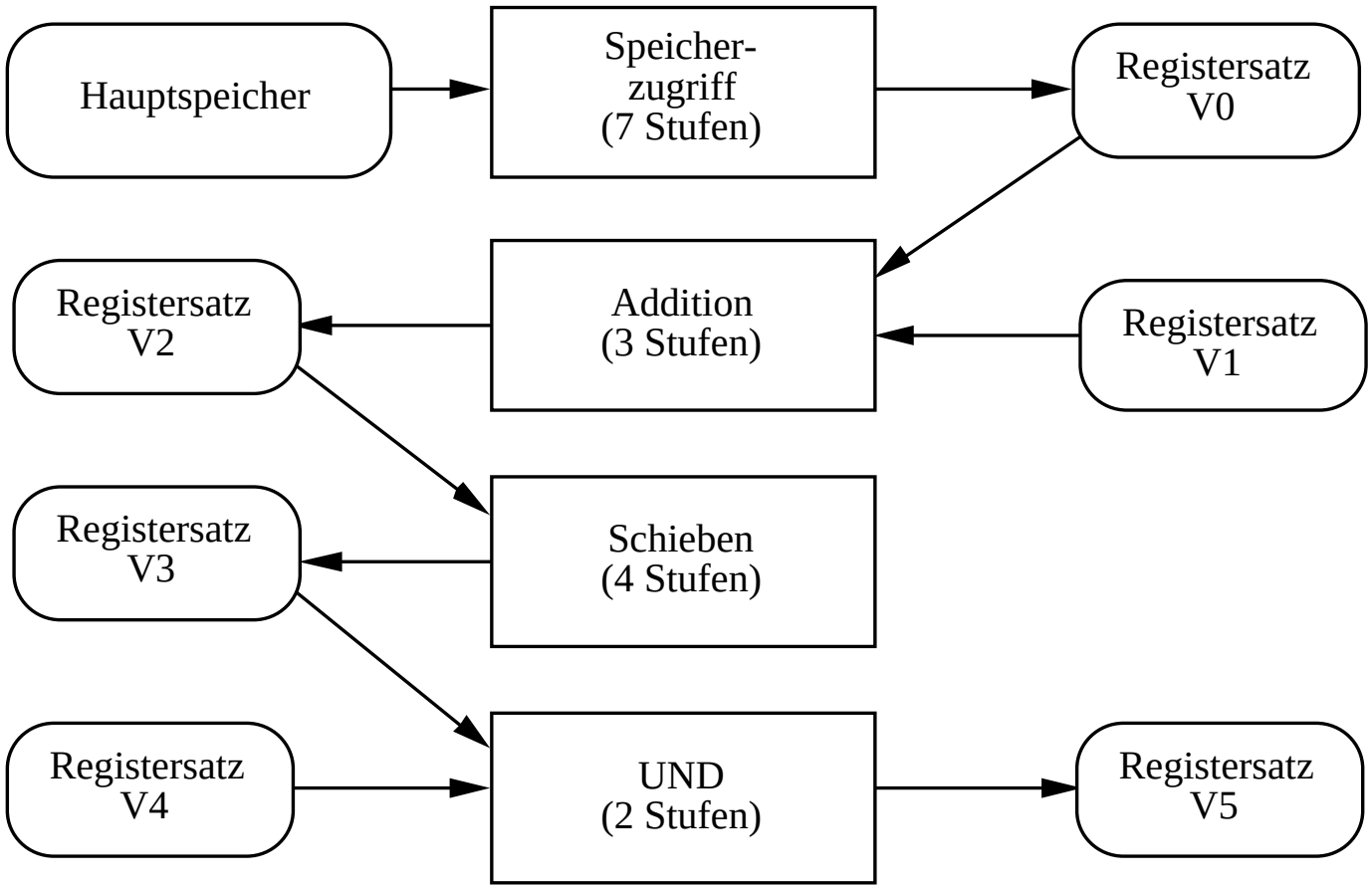


- Vektorprozessoren
  - Verkettung
    - Das Pipeline-Prinzip kann auch auf eine Folge von Vektoroperationen erweitert werden.
    - Zu diesem Zweck werden die (spezialisierten) Pipelines miteinander verkettet,
    - d.h. die Ergebnisse einer Pipeline werden sofort der nächsten Pipeline zur Verfügung gestellt.

- Vektorprozessoren
  - Verkettung
    - Beispiel:
      - B[i] * C[i] + D[i] mit i= 1,2,...,N



- Vektorprozessoren
  - Verkettung von 4 Pipelines



Aus: T. Ungerer: Parallelrechner und parallele Programmierung. Spektrum Akademischer-Verlag, Heidelberg, 1997

- Vektorprozessoren
  - Multifunktions- oder spezialisierte Pipelines
    - Zur Realisierung der arithmetisch-logischen Verknüpfung von Vektoren verwendet man entweder so genannte Multifunktions-Pipelines oder eine Anzahl von spezialisierten Pipelines.
    - Spezialisierte Pipelines werden zur Durchführung von speziellen Funktionen benutzt.
    - Hardware und Steuerung sind relativ einfach.
    - Man benötigt mehrere unabhängige Pipelines, um alle wichtigen Verknüpfungen durchführen zu können.

- Vektorprozessoren
  - Multifunktions- oder spezialisierte Pipelines
    - Zur Realisierung der arithmetisch-logischen Verknüpfung von Vektoren verwendet man entweder so genannte Multifunktions-Pipelines oder eine Anzahl von spezialisierten Pipelines.
    - Multifunktions-Pipelines
      - Der Aufbau einer Multifunktions-Pipeline erfordert eine höhere Stufenzahl, als sie zur Durchführung einer Verknüpfungsoperation notwendig wäre. Für die gerade aktuelle Operation werden alle nicht benötigten Stufen der Pipeline übersprungen.
    - Spezialisierte Pipelines
      - Durchführung von speziellen Funktionen benutzt.
      - Hardware und Steuerung sind relativ einfach.
      - Man benötigt mehrere unabhängige Pipelines, um alle wichtigen Verknüpfungen durchführen zu können.

- Vektorprozessoren
  - Parallelarbeit in einem Vektorrechner
    - Vektor-Pipeline-Parallelität:
      - durch die Stufenzahl der betrachteten Vektor-Pipeline gegeben
    - Mehrere Vektor-Pipelines in einer Vektoreinheit:
      - Vorhandensein mehrerer, meist funktional verschiedener Vektor-Pipelines in einer Vektoreinheit, durch Verkettung hintereinander geschaltet
    - Vervielfachung der Pipelines:
      - Vektor-Pipeline vervielfachen, so dass, bei Ausführung eines Vektorbefehl pro Takt nicht nur ein Paar von Operanden in eine Pipeline, sondern jeweils ein Operandenpaar in zwei oder mehr parallel arbeitende gleichartige Pipelines eingespeist werden.
    - mehrere Vektoreinheiten,
      - die parallel zueinander nach Art eines speichergekoppelten Multiprozessors arbeiten.

- Vektorprozessoren
  - Parallelitätsebenen in der Software
    - Vektor-Pipeline-Parallelität wird durch die Vektorisierung der innersten Schleife mittels eines vektorisierenden Compilers genutzt.
    - Mehrere Vektor-Pipelines in einer Vektoreinheit können durch Verkettung von Vektorbefehlen oder durch einen Vektor-Verbundbefehl (beispielsweise Vector-Multiply-Add) genutzt werden.
    - Bei der Vervielfachung der Pipelines (beispielsweise vier Vektoradditions-Pipelines, die mittels eines VADD-Befehls gleichzeitig aktiviert werden) geschieht die Vektorisierung wieder über die innerste Schleife

- Vektorprozessoren
  - Parallelitätsebenen in der Software
    - Das Vorhandensein mehrerer Vektoreinheiten wird durch ähnliche Parallelisierungsmechanismen wie für speichergekoppelte Multiprozessoren genutzt
    - Beispiel:
      - Aufteilung der Iterationen einer äußeren Schleife auf verschiedene Vektoreinheiten, welche die innere Schleife vektorisiert.

- **Kapitel 4: Vektorrechner**

4.2: Eigenschaften

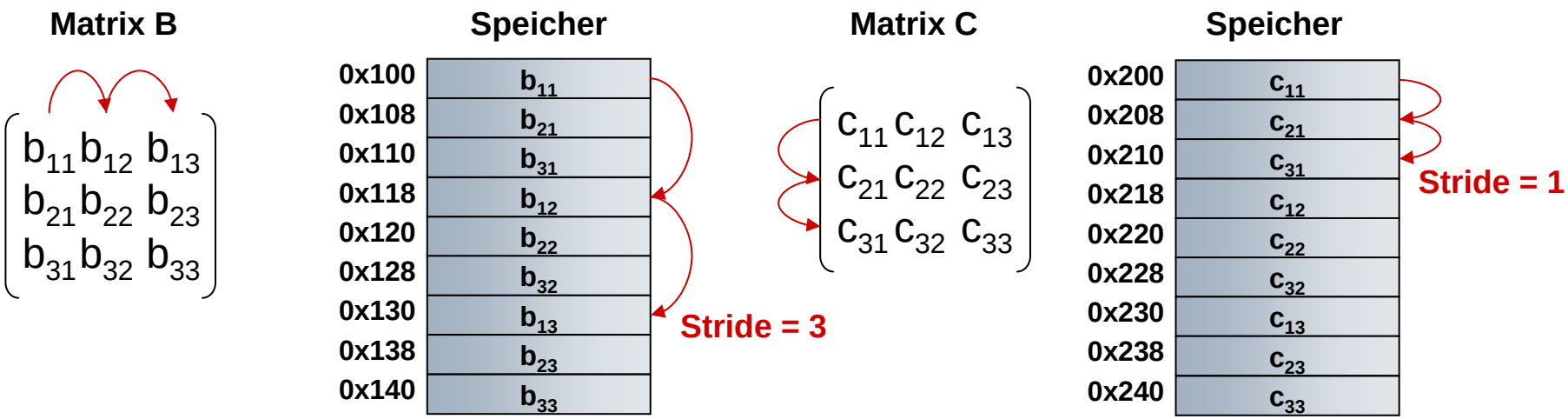
- Vektorprozessoren
  - Vektor Stride
    - Problem: die Elemente eines Vektors liegen nicht in aufeinander folgenden Speicherzellen
    - Beispiel: Matrix-Multiplikation

```
      do 10 i = 1,100
        do 10 J = 1,100
              A(i,j) = 0.0
              do 10 k = 1,100
10                  A(i,j) = A(i,j)+B(i,k)*C(k,j)
```

Anweisung mit der Marke 10: Vektorisierung der Multiplikation der Elemente jeder Reihe von B mit den Elementen der Spalten von C.
Beachte dabei Adressierung der benachbarten Elemente in B und der benachbarten Elemente in C

Eigenschaften

- Vektorprozessoren
  - Vektor Stride
    - Problem: die Elemente eines Vektors liegen nicht in aufeinander folgenden Speicherzellen
    - Beispiel: Matrix-Multiplikation (k=3)



Berechnung: $b_{11}* c_{11}+ b_{12}* c_{21}+b_{13}*c_{31}$ …

Eigenschaften

- Vektorprozessoren
  - Vektor Stride
    - Abstand zwischen Elementen, die in einem Register abgelegt werden müssen
    - Beispiel:
      - bei spaltenweiser Ablage der Daten im Speicher ist die der Stride für die Matrix C gleich 1 (oder 1 Doppelwort) und für die Matrix B gleich 3 (oder 3 Doppelwörter)
    - Vektorprozessor mit Vektorregister können Strides größer 1 verarbeiten:
      - Vektorlade- und Vektrospeicherbefehle mit „Stride-Capability“
        - » Zugriff auf nicht sequentielle Speicherzellen und Umformen in dichte Struktur

Eigenschaften

- Vektorprozessoren
  - Vektor Stride
    - Problem:
      - Stride-Wert ist erst zur Laufzeit bekannt oder kann sich ändern
    - Lösung
      - Ablegen des Stride-Wertes in ein Allzweckregister
      - Vektorspeicherzugriffsbefehle greifen auf den Wert zu
    - Problem:
      - Zugriff auf eine Speicherbank erfolgt häufiger als es die Zugriffszeit der Bank erlaubt
      - Anhalten des Zugriffs, wenn:

$$\frac{\text{Anzahl der Speicherbänke}}{\text{kleinste gemeinsame Vielfache (Stride, Anzahl der Bänke)}} < \text{busy time der Bank}$$

    - In heutigen Vektorrechnern:
      - Verteilen der Zugriffe von jedem Prozessor über mehrere Hundert von Speicherbänken
      - Da Speicherbankkonflikte bei „nonunit strides“ grundsätzlich auftreten können, bevorzugen Programmierer „unit strides“

Eigenschaften

- Vektorprozessoren
  - Bedingt ausgeführte Anweisungen
    - Problem:
      - Programme mit if-Anweisungen in Schleifen können nicht vektorisiert werden:
        - » Kontrollflussabhängigkeiten!
    - Beispiel:

```
      do 100 i=1, 64
        if (A(i).ne. 0) then
              A(i) = A(i) –B(i)
         endif
100 continue
```

    - Lösung
      - Bedingt ausgeführte Anweisungen
      - Umwandlung von Kontrollflussabhängigkeiten in Datenabhängigkeiten

Eigenschaften

- Vektorprozessoren
  - Bedingt ausgeführte Anweisungen
    - Vektor-Maskierungssteuerung
      - verwendet einen Boole‘schen Vektor der Länge der festgelegten MVL (maximale Vektorlänge), um die Ausführung eines Vektorbefehls zu steuern, in ähnlicher Weise wie bedingt ausgeführte Befehle eine Boole‘sche Bedingung verwenden, um zu bestimmen, ob eine Instruktion ein gültiges Ergebnis liefert oder nicht
      - Vektor-Mask-Register
        - » jede ausgeführte Vektorinstruktion arbeitet nur auf den Vektorelementen, deren Einträge eine 1 haben. Die Einträge im Zielvektorregister, die eine 0 im entsprechenden Feld des VM Registers haben, werden nicht verändert

Eigenschaften

- Vektorprozessoren
  - Bedingt ausgeführte Anweisungen
    - Vektor-Maskierungssteuerung
      - Beispiel:

```
LV       V1,Ra     ;load vector A into V1
LV       V2,Rb     ;load vector B
L.D      F0,#0     ;load FP zero into F0
SNEVS.D  V1,F0     ;sets VM(i) to 1 if V1(i)!=F0
SUBV.D   V1,V1,V2  ;subtract under vector mask
CVM                ;set the vector mask to all 1s
SV Ra,V1           ;store the result in A
```

Eigenschaften

- Vektorprozessoren
  - Dünn besetzte Matrizen
    - Elemente eines Vektors werden in einer komprimierten Form im Speicher abgelegt
    - Beispiel:

```
      do   100 i = 1,n
100        A(K(i)) = A(K(i)) + C(M(i))
```

» implementiert die Summe der dünn besetzten Felder A und C mit Hilfe der Indexvektoren K und M. K und M zeigen jeweils die Elemente von A und C an, die nicht 0 sind.

» Alternative Darstellung ist die Verwendung von Bit-Vektoren

Eigenschaften

- Vektorprozessoren
  - Dünn besetzte Matrizen
    - SCATTER-GATHER Operationen mit Index-Vektoren
      - unterstützen den Transport zwischen der gepackten Darstellung und der normalen Darstellung dünn-besetzter Matrizen
      - GATHER-Operation
        - » verwendet Index-Vektor und holt den Vektor, dessen Elemente an den Adressen liegen, die durch Addition einer Basisadresse und den Offsets im Index-Register berechnet werden
        - » Nicht gepackte Darstellung im Vektorregister
      - SCATTER-Operation
        - » Speichern der gepackten Darstellung

Eigenschaften

- Vektorprozessoren
  - Dünn besetzte Matrizen
    - SCATTER-GATHER Operationen mit Index-Vektoren

```
LV        Vk,Rk          ;load K
LVI       Va,(Ra+Vk)     ;load A(K(I))
LV        Vm,Rm          ;load M
LVI       Vc,(Rc+Vm)     ;load C(M(I))
ADDV.D    Va,Va,Vc       ;add them
SVI       (Ra+Vk),Va     ;store A(K(I))
```

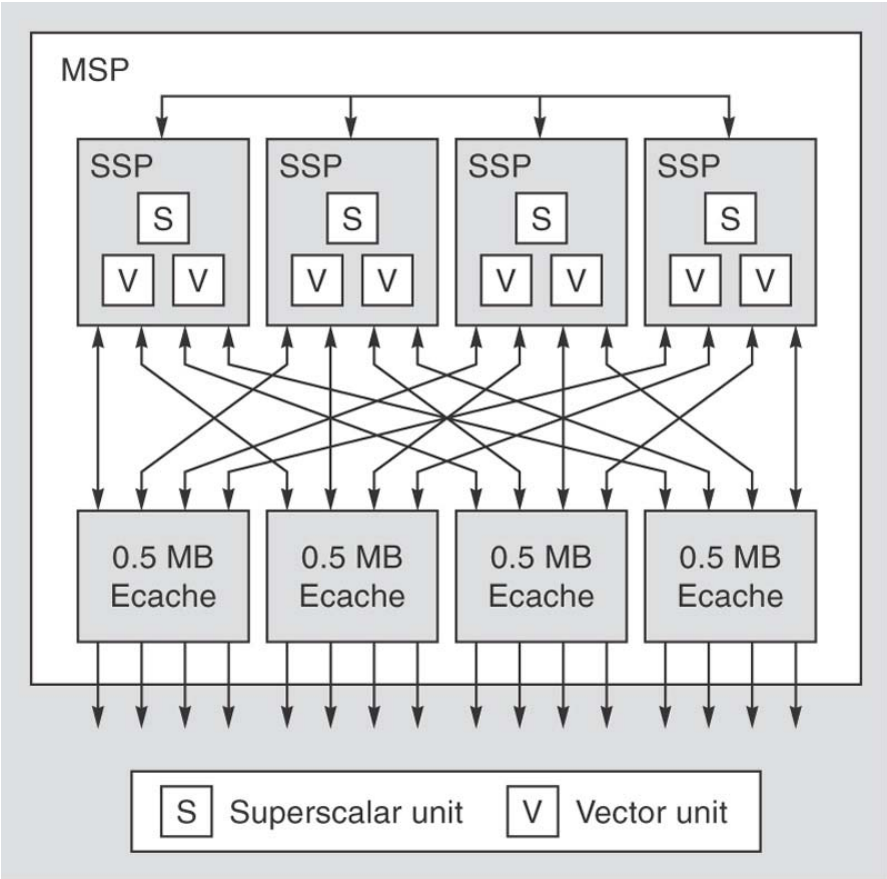
    - Problem für vektorisierenden Compiler: konservative Annahmen wegen Speicherreferenzen
      - Verwendung einer Software-Hash Tabelle, ähnlich der ALAT im Intanium
        - » erkennt, wenn zwei Elemente innerhalb einer Interation auf dieselbe Adresse zeigen

Eigenschaften

- Vektorprozessoren
  - Beispiel: CRAY X1 (2002)
    - ausbaubar auf mehrere tausend Vektorprozessoren
    - Shared-Memory Architektur
    - Prozessorarchitektur: Multi-Streaming Prozessor (MSP) mit
      - 4 Single-Streaming Prozessoren (SSP)
        - » Jeder SSP ist ein Vektorprozessor mit einer skalaren Einheit.

Eigenschaften

- Vektorprozessoren
  - Beispiel: CRAY X1 (2002)
    - Multi-Streaming-Prozessor (MSP)

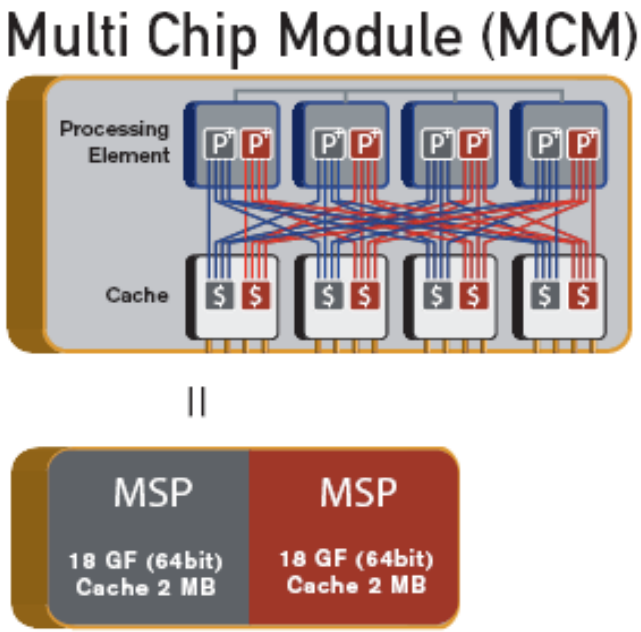


Eigenschaften

- Vektorprozessoren
  - Beispiel: CRAY X1 (2002)
    - Single-Streaming-Prozessor (MSP)
      - skalare Einheit
        - » Superskalarer Prozessor
        - » 16 KB Befehls-Cache und 16 KB Write-through Daten-Cache, zweifach-satz-assoziativ mit 32-Bytes langen Zéilen.
      - Vektoreinheit
        - » Vektorregisterdatei
        - » drei Vektor-Arithmeische Einheiten
        - » eine Vektor-Lade- und Speichereinheit

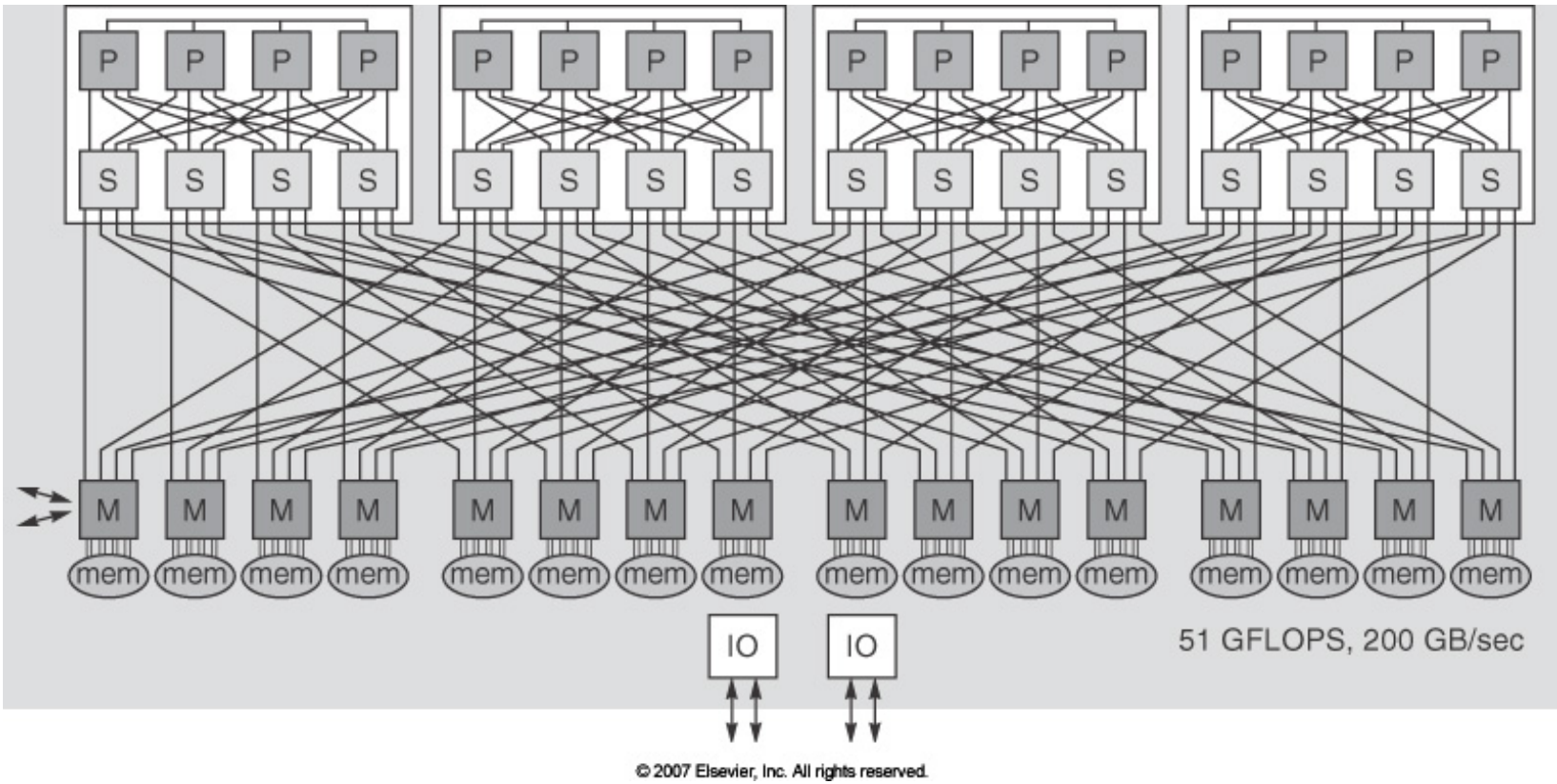
Eigenschaften

- Vektorprozessoren
  - Beispiel: CRAY X1 (2002)
    - Single-Streaming-Prozessor (MSP)



Eigenschaften

- Vektorprozessoren
  - Beispiel: CRAY X1 (2002)
    - Knoten:



Eigenschaften

- Vektorprozessoren
  - Beispiel: CRAY X1 (2002)
    - Konfigurationen

Cray X1E Supercomputer Sample Configurations

	1 AC* Cabinet	1 LC* Cabinet	4 LC Cabinets	8 LC Cabinets
MSPs	32	128	512	1,024
Peak Performance	576 GFLOPS	2.3 TFLOPS	9.2 TFLOPS	18.4 TFLOPS
Maximum Memory	128 GB	512 GB	2 TB	4 TB
Aggregate Peak Memory Bandwidth	800 GB/s	3.2 TB/s	12.8 TB/s	25.6 TB/s

*Air Cooled (AC) Liquid Cooled (LC)

Eigenschaften

- Vektorprozessoren
  - Beispiel: CRAY X1 (2002)
    - Merkmale

CPU	64-bit Cray X1E Multistreaming Processor (MSP); 8 per compute module
	8 vector pipes per MSP
	Scalar operations overlapped with vector operations
	IEEE floating point compatible
	Bit matrix multiply, pop count, and integer add/subtract with carry
Cache	2MB cache per MSP
FLOPS	18 GFLOPS theoretical peak performance per MSP (1.13 GHz vector clock speed)
SMP	4-way SMP node
Main Memory	16 GB or 32 GB RDRAM memory per compute module
Memory Bandwidth	200 GB/s per compute module peak bandwidth; 34 GB/s per processor peak bandwidth

Eigenschaften

- Vektorprozessoren
  - Beispiel: CRAY X1 (2002)
    - Merkmale

Interconnect	Custom high performance interconnect providing distributed shared memory access through entire system
	16 parallel 2D torus networks
	51 GB/s per compute module peak bandwidth
	5 microsecond MPI latency between processors
External I/O	Peak bandwidth of 4.8 GB/s through dedicated I/O channels from each compute module
	Shared memory allows any processor to perform I/O to any I/O channel
	Separate Cray Network Server (CNS) supports network I/O
File System	XFS (direct-attached disk)
	ADIC StorNext File System (SAN-attached disk)

Eigenschaften

- Vektorprozessoren
  - Beispiel: CRAY X1 (2002)
    - Merkmale

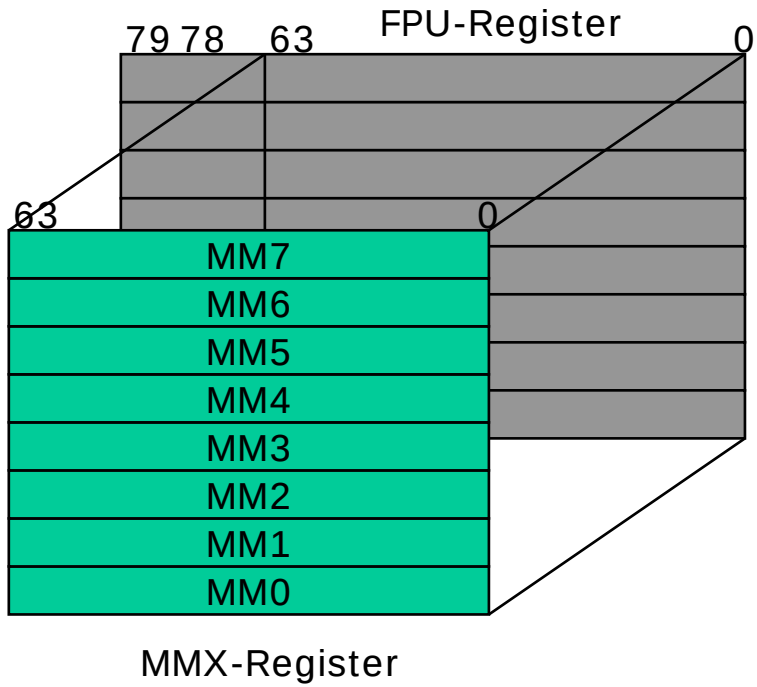
Reliability Features	System is resilient to hardware or software failures in application processors	
	System can be run in degraded mode if necessary due to hardware failures	
Operating System	UNICOS/mp	
Parallel Programming	MPI 1.2, SHMEM, OpenMP, Co-Array Fortran, UPC	
Cray Fortran Compiler	Fully adheres to Fortran 95 standard (ISO/IEC 1539-1:1997 Part 1); supports selected features from proposed Fortran 2003 standard	
Cray C & C++ Compilers	Adhere to industry standards for C (ISO/IEC 9899:1999 (C99)) and C++ (ISO/IEC 14882:1998)	
Cabinets	**Liquid-cooled (LC) cabinet** up to 16 compute modules (128 processors)	**Air-cooled (AC) cabinet** up to 4 compute modules (32 processors)
Maximum Configuration	Up to 64 LC cabinets (8,192 processors)	Up to 4 AC cabinets (128 processors)
Standard Configuration*	Up to 8 LC cabinets (1,024 processors)	1 AC cabinet (32 processors)
Power (cabinet)	65 kW, 200-208 VAC	17 kW, 200-208 VAC
Footprint (cabinet)	50.75 in. x 103 in. (1.3 m x 2.6 m)	35.5 in. x 59.75 in. (.9 m x 1.5 m)
Weight (cabinet)	5,754 lbs. (2,610 kg)	1,973 lbs. (895 kg)

* Configurations exceeding these limits, up to the design parameters of the system, are available by special order.

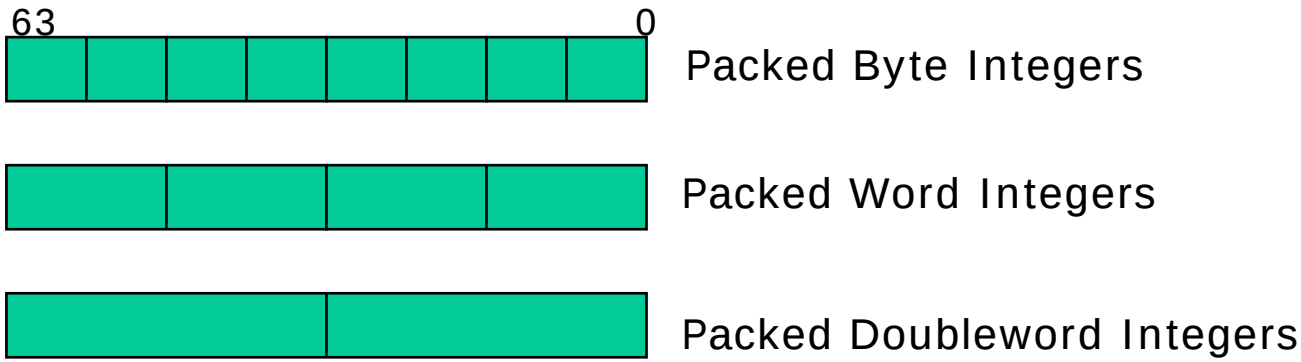
- **Kapitel 4: Vektorverarbeitung**

4.3: SIMD-Verarbeitung in Mikroprozessoren

- Beispiel Intel MMX-Technologie
  - MMX-Register, abgebildet auf FPU-Register

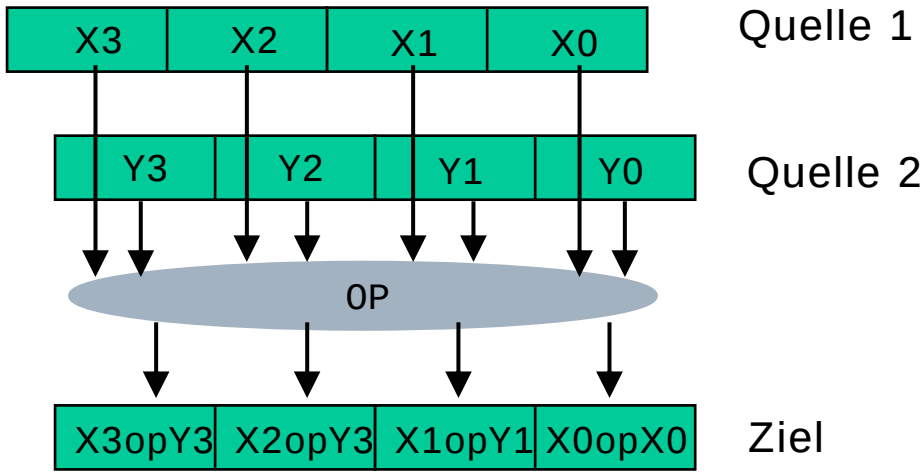


- Beispiel Intel MMX-Technologie
  - MMX-Datentypen und Formate

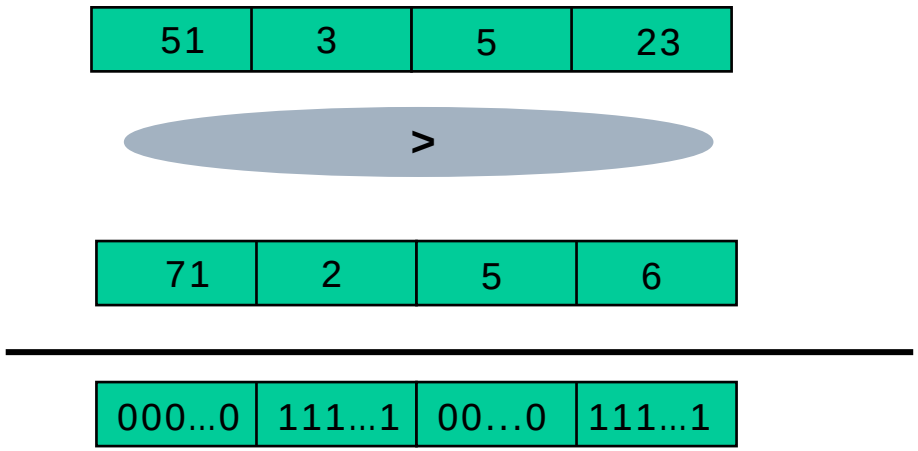


- Beispiel Intel MMX-Technologie
  - MMX-Befehle
    - Datentransport
    - Konvertierung
    - Gepackte arithmetische Befehle
    - Vergleichsbefehle
    - Logische Befehle
    - Zustandsverwaltung

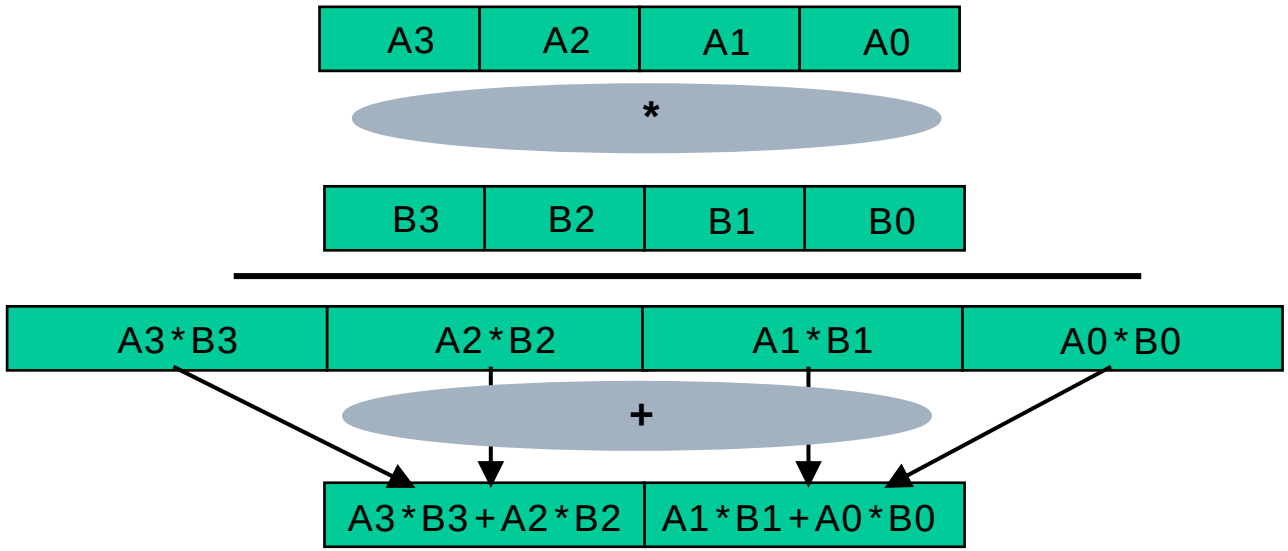
- Beispiel Intel MMX-Technologie
  - SIMD-Verarbeitung



- Beispiel Intel MMX-Technologie
  - SIMD-Verarbeitung
    - **PCMPGTW**-Befehl auf Wortoperanden (Vergleich größer als)



- Beispiel Intel MMX-Technologie
  - SIMD-Verarbeitung
    - **PMADDWD**-Befehl auf Wortoperanden, Ergebnis im Doppelwort (Multiply and add)



- Beispiel Intel MMX-Technologie
  - Saturation Arithmetik
    - Algorithmen in der graphischen Datenverarbeitung vermeiden Überlauf und Unterlauf bei der Addition und Subtraktion von nicht vorzeichenbehafteten Pixeln
    - Übergang zum größten oder kleinsten darstellbaren Wert

```
  10101010
+ 11001100
  11111111
```

    - Keine Überprüfung, ob Über- oder Unterlauf des Zahlenbereichs, keine Ausnahmeverarbeitung

- Intel MMX-Technologie
  - Beispiel: Chroma keying („bluebox“ Technik)
    - Überlagerung des Bildes einer Frau auf ein Bild mit einer Blume mit Hilfe eines blauen Hintergrundes
      - x: Bild der Frau auf dem blauen Hintergrund
      - y: Blumenbild
      - new_image: neues Bild

```
for (i=0; i<image_size; i++) {
    if (x[i]==blue) new_image[i]=y[i]
            else    new_image[i]=x[i]
}
```

- Intel MMX-Technologie
  - Beispiel: Chroma keying („bluebox“ Technik)
    - MMX Code-Folge

```
movq      mm3,mem1      #load 8 pixels from woman's image (addr. mem1)
movq      mm4,mem2      #load 8 pixels from flower's image (addr. mem2)
pcmpeqb   mm1,mm3       #gen. Selection bit mask into mm1 (orig. „blue“)
pand      mm4,mm1       #AND; use the bit mask for cond. select into mm4.
pandn     mm1,mm3       #(NOT mm1) AND mm3; -“- into mm1
por       mm4,mm1       #OR; result into mm4
```

pcmpeqb mm1,mm3

mm1	blue	blue	blue	blue	blue	blue	blue	blue
mm3	x7!=blue	x6!=blue	x5=blue	x4=blue	x3!=blue	x2!=blue	x1=blue	x0=blue
mm1	0x0000	0x0000	0xFFFF	0xFFFF	0x0000	0x0000	0xFFFF	0xFFFF

pand mm4,mm1

mm4	y7	y6	y5	y4	y3	y2	y1	y0
mm1	0x0000	0x0000	0xFFFF	0xFFFF	0x0000	0x0000	0xFFFF	0xFFFF
mm4	0x0000	0x0000	y5	y4	0x0000	0x0000	y1	y0

pandn mm1,mm3

mm1	0x0000	0x0000	0xFFFF	0xFFFF	0x0000	0x0000	0xFFFF	0xFFFF
mm3	x7	x6	x5	x4	x3	x2	x1	x0
mm1	x7	x6	0x0000	0x0000	x3	x2	0x0000	0x0000

por mm4,mm1

mm1	x7	x6	y5	y4	x3	x2	y1	y0