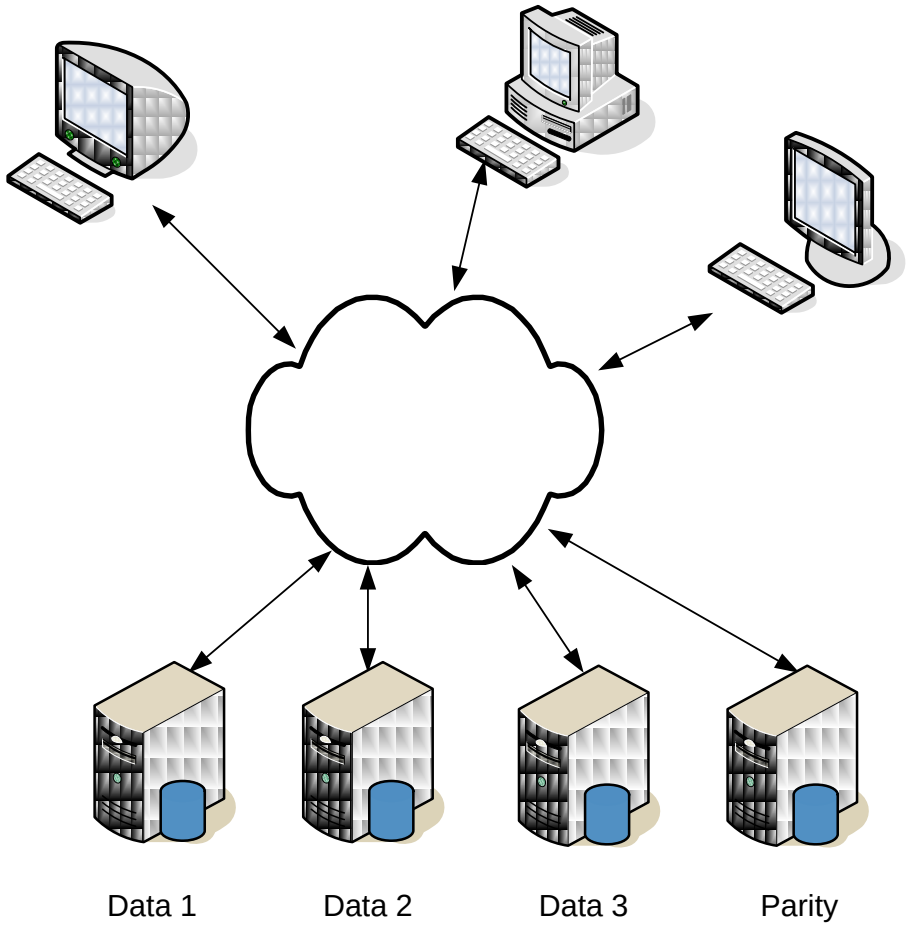
Strukturelle Redundanz

- Strukturelle Redundanz bezeichnet die Erweiterung eines Systems um zusätzliche (gleich- oder andersartige) für den Nutzbetrieb entbehrliche Komponenten.
- Beispiele:
  - 2-von-3-Rechnersysteme
  - redundante Kommunikationskanäle (siehe TTP, Flexray)
  - mehrfache Kopien einer Datei
  - unterschiedliche Sensoren

Anwendungsbeispiel:

- Verteilte Dateisysteme
  NetRAID (Lübeck),
  xFs (Berkeley)

- Ziel:
  - skalierbare Speichergröße
  - hohe Zugriffsraten
  - Ausfallstoleranz



Funktionelle Redundanz

- Funktionelle Redundanz bezeichnet die Erweiterung eines Systems um zusätzlich für den Nutzbetrieb entbehrliche Funktionen.
- Beispiele:
  - Testfunktionen
  - Funktionen zur Rekonfiguration im Mehrrechnerbetrieb
  - Erzeugung eines Paritätsbits
  - N-Versions-Programmierung

Diversität (N-Versions-Programmierung)

- Diversität bezeichnet die Erfüllung der Spezifikation einer Nutzbetriebs-Funktion durch mehrere verschiedenartig implementierte Funktionen.
- Um Entwurfsfehler in Hard- und / oder Softwaresystemen tolerieren zu können, ist der Einsatz von Diversität zwingend. Diversität verbessert die Zuverlässigkeit aber nicht unbegrenzt. Die Verbesserungsgrenze ist insbesondere von der Schwierigkeit des zu lösenden Systems vorgegeben.
- Um Diversität zu realisieren, muss der Entwurfsspielraum für die verschiedenen Varianten genutzt werden.
- Ansätze:
  - Unabhängiger Entwurf
  - Gegensätzlicher Entwurf

Beispiel aus dem Alltag: Arztbesuch

- Typisches Beispiel für das N-Versions-Konzept: Konsultation von verschiedenen Ärzten:
  - Unterschiedliche Spezialisierungen
  - Unterschiedliche Untersuchungen
  - Unterschiedliche Behandlungsmethoden

Informationsredundanz

- Informationsredundanz bezeichnet zusätzliche Informationen neben der Nutzinformation.
- Beispiele:
  - Fehlerkorrigierende Codes
  - Doppelt verkettete Listen
  - Paritätsbits
  - CRC: cyclic redundancy check
- Voraussetzung: Fehler dürfen sich nur auf einen beschränkten Teil der gesamten Information auswirken (z.B. Fehlfunktions-Annahme)

Cyclic Redundancy Check

- Nachrichten werden als Polynome interpretiert, korrekte Nachrichten müssen ein Vielfaches vom Generatorpolynom G(u) sein.
- Beispiel zur Berechnung der Kontrollstellen: $k = 3$, $G(u) = u^3 + u^1 + 1$, $m = 4$ Nachrichtenstellen, $n = k + m$ Gesamtstellen.
- Seien als Nachrichtenstellen gewählt: (**1001**). Also Codewort: (**1001???**).
- Polynomdivision:

$$\overbrace{(u^6\ u^5\ u^4\ u^3\ u^2\ u^1\ u^0)}^{X(u)} \ / \ \overbrace{(u^3\ u^2\ u^1\ u^0)}^{G(u)} = \overbrace{(u^3\ u^2\ u^1\ u^0)}^{Q(u)}$$

$$1\ 0\ 0\ 1\ 0\ 0\ 0 \ / \ 1\ 0\ 1\ 1 = 1\ 0\ 1\ 0$$

$$\begin{array}{r}
1\ 0\ 1\ 1 \\
\hline
0\ 1\ 0\ 0 \\
0\ 0\ 0\ 0 \\
\hline
1\ 0\ 0\ 0 \\
1\ 0\ 1\ 1 \\
\hline
0\ 1\ 1\ 0 \\
0\ 0\ 0\ 0 \\
\hline
1\ 1\ 0 \quad \longleftarrow \text{Rest der Division}
\end{array}$$

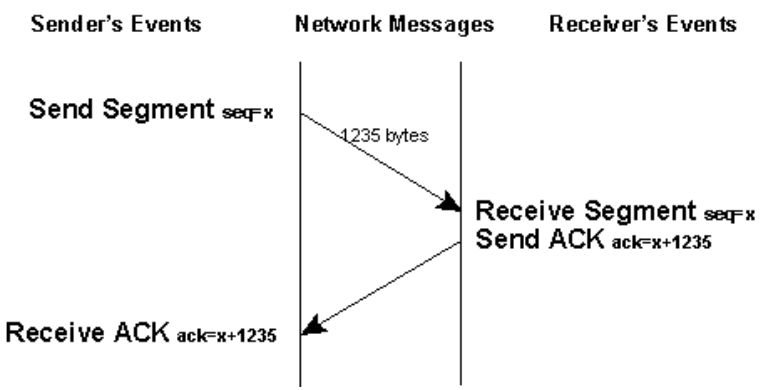
- Also Codewort: (**1001000**) – (**0000110**) = (**1001110**)

Zeitredundanz

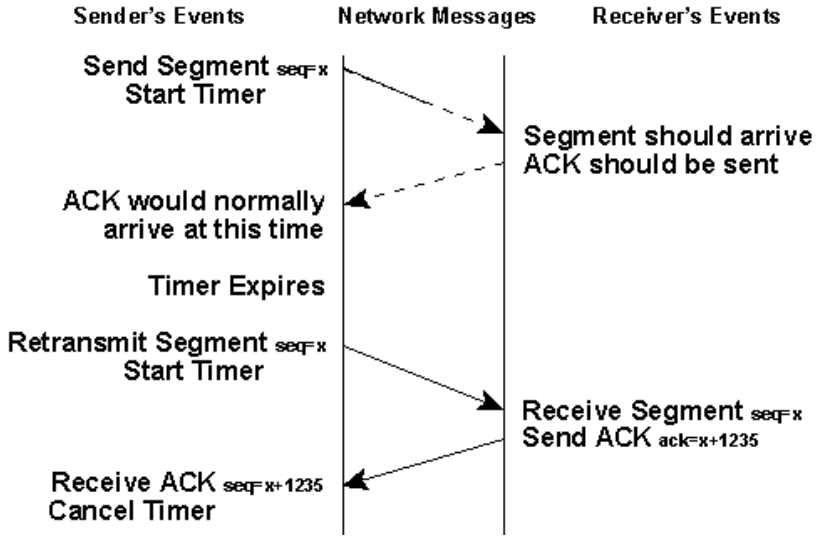
- Zeitredundanz bezeichnet über den Zeitbedarf des Normalbetriebs hinausgehende zusätzliche Zeit, die einem funktionell redundantem System zur Funktionsausführung zur Verfügung steht.
- Beispiele:
  - Wiederholungsbetrieb
  - Zeitbedarf für Konsistenzmechanismen in verteilten Dateisystemen

Beispiel: TCP

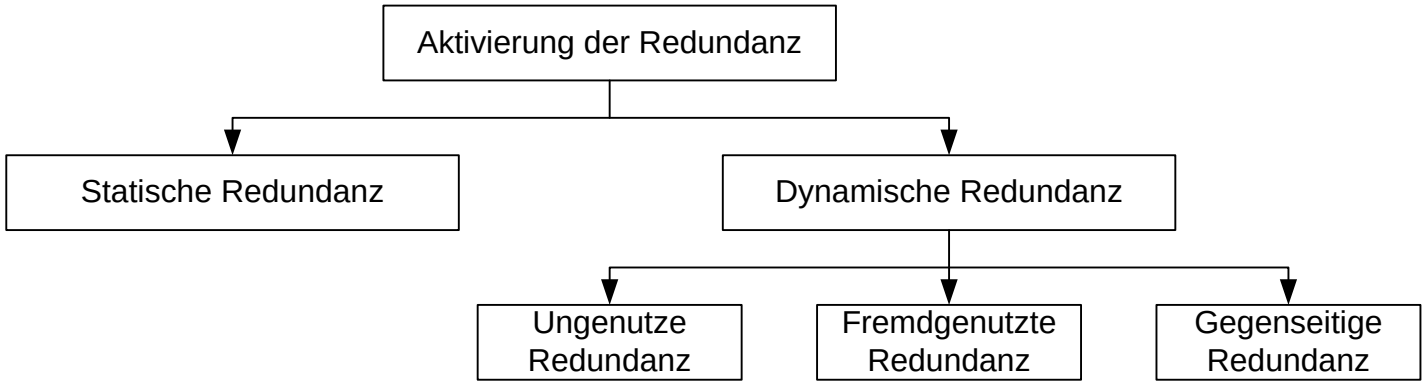
Fehlerfrei



Verlust einer Nachricht



Aktivierung der Redundanz



Statische Redundanz

- Statische Redundanz bezeichnet das Vorhandensein von redundanten Mitteln, die während des gesamten Einsatzzeitraums aktiv zu den zu unterstützenden Funktionen beitragen.
- Ausprägungen:
  - Statische strukturelle Redundanz: z.B. n-von-m System
  - Statische funktionelle Redundanz (Zusatzfunktionen): z.B. doppeltes Senden von Nachrichten auf unterschiedlichen Wegen
  - Statische funktionelle Redundanz (Diversität): N-Versions-Programmierung
  - Statische Informationsredundanz: fehlerkorrigierende Codes
  - Statische Zeitredundanz: statische Mehrfachausführung einer Funktion

Dynamische Redundanz

- Dynamische Redundanz bezeichnet das Vorhandensein von redundanten Mitteln, die erst im Ausnahmebetrieb (d.h. nach Auftreten eines Fehlers) aktiviert werden, um zu den zu unterstützenden Funktionen beizutragen.
- Typisch für dynamisch strukturelle Redundanz ist die Unterscheidung in **Primärkomponenten** und **Ersatzkomponenten**. Die Dauer der Umschaltung hängt im Wesentlichen von den ggf. erforderlichen Vorbereitungsmaßnahmen der Ersatzkomponenten ab. Hier wird zwischen heißer Reserve (**hot stand-by**) und kalter Reserve (**cold stand-by**) unterschieden.
- Die Definition verlangt kein vollkommen passives Verhalten. Folgende Szenarien sind möglich:
  - ungenutzte Redundanz: Ersatzkomponenten sind bis zur fehlerbedingten Aktivierung passiv
  - fremdgenutzte Redundanz: Ersatzkomponenten erbringen nur Funktionen, die von den zu unterstützenden Funktionen verschieden sind und im Fehlerfall storniert werden
  - gegenseitige Redundanz: Komponenten stehen sich gegenseitig als Reserve zur Verfügung. Im Fehlerfall übernimmt eine Komponente die Funktionen der anderen zusätzlich zu den eigenen.

Entwicklung sicherheitskritischer Systeme

Fehlertoleranzmechanismen

Grundlage: Fehlererkennung

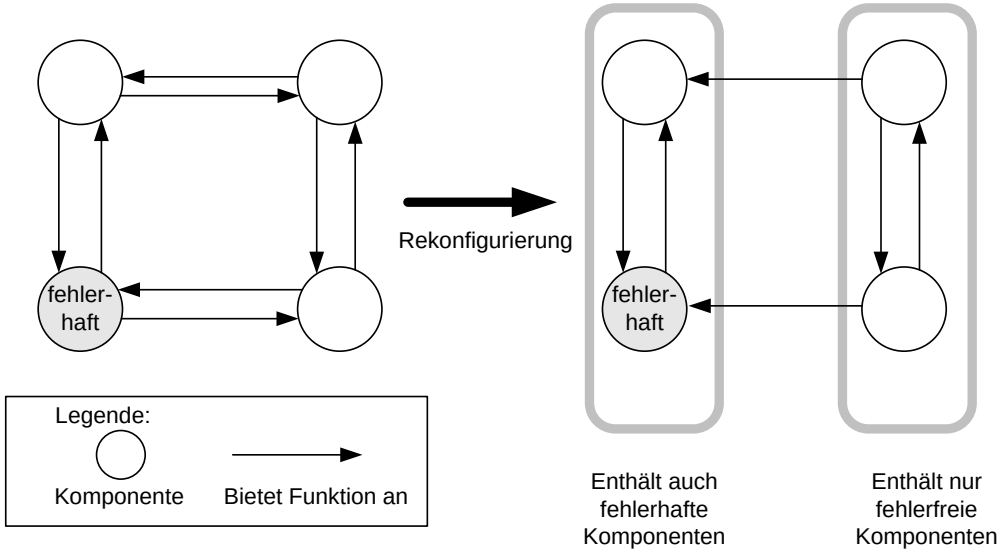
- Grundlage der Fehlertoleranzmechanismen ist die Fehlerdiagnose.
- Ziele der Fehlerdiagnose ist:
  - das Erkennen von Fehlern (im Nutzbetrieb)
  - die Lokalisierung von Fehlern (zumeist im Ausnahmebetrieb)
  - die Bestimmung des Behandlungsbereichs (zumeist im Ausnahmebetrieb)

Fehlererkennung

- Möglichkeiten zur Fehlererkennung:
  - Zeitschrankenüberwachung
  - Absoluttests: getestet wird direkt das Ergebnis (z.B. Anzahl der Elemente muss nach Sortieren gleich der eingegebenen Anzahl sein)
  - Relativtests: Vergleich von mehreren Ergebnissen redundanter Prozesse
    - bei deterministischen Prozessen
    - bei nicht deterministischen Prozessen
  - Nutzung von Informationsredundanz (z.B. CRC)

Rekonfigurierung

- Durch Rekonfigurierung werden fehlerhafte Komponenten ausgegrenzt und bestehende Funktionszuordnungen zwischen fehlerhaften und fehlerfreien Komponenten aufgelöst.
- Nach einer Rekonfigurierung ist das System in zwei Komponenten-Teilmengen partitioniert: eine enthält nur fehlerfreie, die andere auch fehlerhafte Komponenten.



Rekonfigurierung: Beitrag zur Fehlertoleranz

- Rekonfigurierung dient zur Behandlung von Funktionsausfällen, nicht aber der Behebung von Fehlzuständen.

  → nicht ausreichend für erfolgreiche Fehlerbehandlung

  → Verfahren zur Fehlerbehebung (Rückwärts-, Vorwärtsbehebung) oder Fehlerkompensierung (Fehlermaskierung, Fehlerkorrektur) müssen hinzukommen.

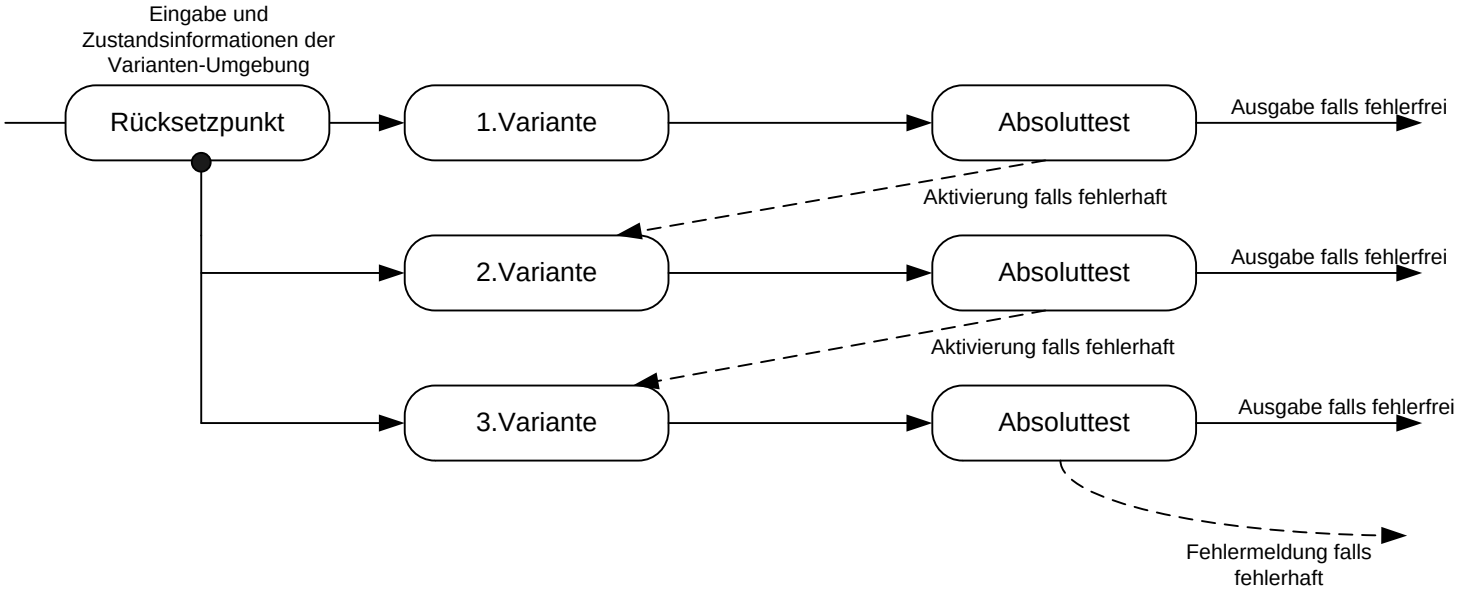
Rückwärtsbehebung (backward recovery)

- Rückwärtsbehebung versetzt Komponenten in einen Zustand, den sie bereits in der Vergangenheit angenommen hatten oder als konsistenten Zustand hätten annehmen können.
- „Konsistent“ bedeutet, dass die lokalen Komponentenzustände und die aktuellen Interaktionen mit anderen Komponenten die (Protokoll- bzw. Dienst-) Spezifikation nicht verletzen.
- Rückwärtsbehebung ist bei intermittierenden Fehlern ausreichend, bei permanenten Fehlern ist sie als Ergänzung zur Rekonfigurierung zu sehen.
- Bei reiner Rückwärtsbehebung kann die Fehlererkennung nur über Absoluttests (da keine redundanten Ergebnisse vorhanden) erfolgen. Diese Tests werden periodisch oder ereignisabhängig durchgeführt.

Rücksetzpunkte (recovery points)

- Nach Auftreten eines Fehlers lassen sich Zustandsinformationen aus der Zeit vor Auftreten eines Fehlers nur gewinnen, wenn die Informationen zuvor kopiert wurden und an einem getrennten Ort zwischengespeichert wurden. Die abgespeicherte Zustandsinformation wird als Rücksetzpunkt bezeichnet.
- Rücksetzpunkte werden periodisch oder ereignisbasiert erstellt und verursachen also schon im Normalbetrieb einen Extrazeitaufwand.
- Zumeist finden vor der Rücksetzpunkterstellung Absoluttests statt.
- Auch Rücksetzpunkte können fehlerhaft sein (Auftreten eines Fehlers direkt nach Absoluttest bzw. beim Speichern des Rücksetzpunktes, Fehlererkennung mit einer Wahrscheinlichkeit <1). Deshalb muss das System eventuell mehrfach zurückgesetzt werden.

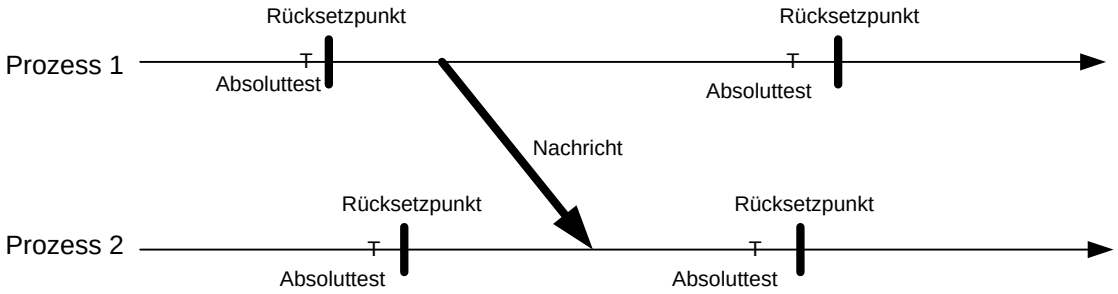
Rückwärtsbehebung diversitärer Systeme



Dieses Verfahren entspricht dem Ausprobieren mehrerer Funktionen. Z.B. das Testen von verschiedenen Browsern (bis Benutzer ein Programm gefällt, die Anzeige einer Internetseite korrekt ist).

Rücksetzlinien bei interagierenden Prozessen

Betrachten wir folgendes Beispiel:



Tritt bei einem der beiden Prozesse zwischen den jeweiligen Rücksetzpunkten ein Fehler auf, so müssen beide Prozesse zurückgesetzt werden.
Warum?

Definition: Die Menge der Rücksetzpunkte, auf die mehrere Prozesse zugleich zurückgesetzt werden können, heißt Rücksetzlinie (recovery line).

Mögliche Probleme: Dominoeffekt

Vor- und Nachteile der Rückwärtsbehebung

- Rückwärtsbehebung verwendet die vorhandenen Betriebsmittel sparsam.
- Im Fehlerfall lässt sich ein Prozess wiederholt zurücksetzen (solange Rücksetzpunkte vorhanden), dadurch erhöht sich die Menge der tolerierenden Fehler.
- Wiederholungsbetrieb erfordert nicht zwangsläufig die gleichen Eingaben wie der zuvor erfolgte Nutzbetrieb (Nicht-Determinismus zulässig).
- Rückwärtsbehebung ist transparent (unabhängig von der Anwendung) implementierbar.
- Nur Absoluttests, keine Relativtests anwendbar.
- Menge der tatsächlichen tolerierten Fehler ist wegen der Abhängigkeit von verschiedenen Absoluttest-Algorithmen kaum formal spezifizierbar.
- Rücksetzpunkterstellung kostet schon im Normalbetrieb.
- Der im Fehlerfall erforderliche Wiederholungsbetrieb kann Zeitredundanz in beträchtlichem Umfang fordern. ➔ schwierigere Abschätzung des Zeitverhaltens und damit eher schlecht für Echtzeitsysteme geeignet

Vorwärtsbehebung

- **Vorwärtsbehebung** bezeichnet Fehlerbehebungs-Verfahren, die keine Zustandsinformationen der Vergangenheit verwenden.
- Basis dieser Verfahren sind Fehlfunktions-Annahmen und anwendungsspezifisches Wissen.
- **Beispiel**: Geht aufgrund eines Fehlers in einem Rechner ein zuvor gelesener Temperaturwert verloren, so kann er durch zweimaliges Einlesen der aktuellen Temperatur und Extrapolation näherungsweise zurückgewonnen werden (Voraussetzung: zeitliche Ableitung der Temperatur ändert sich kaum).

Vor- und Nachteile der Vorwärtsbehebung

- Aufwand an struktureller Redundanz ist gering: nur Absoluttests und die erst im Ausnahmebetrieb zu aktivierenden Ausnahmebehandler sind hinzuzufügen
- Laufzeitaufwand im Normalbetrieb wird nur von Absoluttests verursacht und ist daher minimal
- Minimale Verzögerung des Systems im Fehlerfall, da Vorwärtsbehebung meistens relativ einfache Algorithmen umfasst ➔ gute Eignung für Echtzeitsysteme
- Vorwärtsbehebung ist nicht transparent implementierbar, sondern anwendungsabhängig
- hoher Entwurfsaufwand
- Gelingen hängt stark vom Schwierigkeitsgrad der Anwendung ab
- Nur durch Absoluttests erkennbare Fehler sind überhaupt tolerierbar
- Oft nur degradierter Betrieb nach Vorwärtsbehebung möglich

Fehlermaskierung

- Das Verfahren der Fehlermaskierung berechnet aus redundant berechneten Ergebnissen ein korrektes Ergebnis zur Weitergabe.
- Typischerweise ist die „Maske“ durch einen Mehrheitsentscheider realisiert, der Ergebnisse durch einen Relativtest vergleicht. Dieses Verfahren toleriert fehlerhafte Ergebnisse solange diese in der Minderheit bleiben.
- Typische Ausprägungen sind 2-von-3-Systeme oder 3-von-5 Systeme.

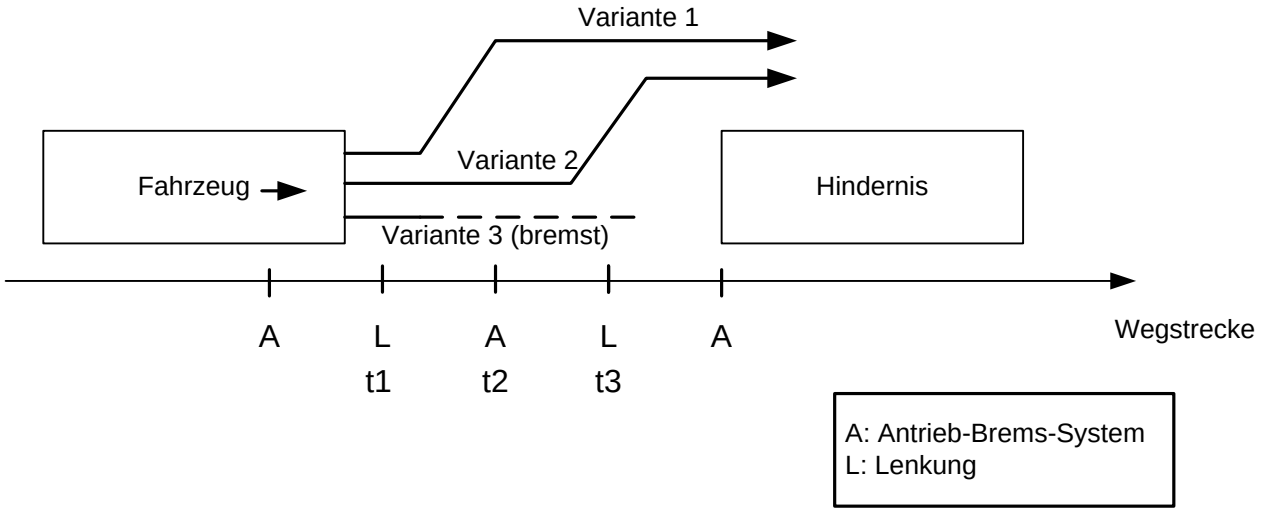
Maskierungsentscheidungen/Votierung

- für **deterministische** Prozesse:
  - Mehrheitsentscheidung: Mehrheit der Gesamtanzahl von Komponenten nötig
  - Paarentscheidung: Annahme, dass fehlerhafte Ergebnisse nie gleich sind zwei übereinstimmende Ergebnisse sind immer korrekt (Reduzierung der Anzahl nötiger Vergleiche)
  - Meiststimmenentscheidung
  - Einstimmigkeitsentscheidung: alle Komponenten müssen im Ergebnis übereinstimmen (Erhöhung der Sicherheit, gleichzeitig wird Zuverlässigkeit des Systems gesenkt)
- für **nicht deterministische** Prozesse:
  - Medianentscheidung: Mittleres Ergebnis wird für die Ausgabe übernommen
  - Intervallentscheidung: Annahme, dass korrekte Ergebnisse in einem Intervall liegen, ein Ergebniswert aus
  - Intervall wird gewählt
  - Kugelentscheidung: wie Intervallentscheidung nur mit mehrdimensionalen Ergebnissen, statt Intervall wird nach kürzesten Abständen gesucht

Vor- und Nachteile der Fehlermaskierung

- Fehlermaskierung reicht als **einziges** Fehlertoleranz-Verfahren aus.
- Maskierer lassen sich vergleichsweise einfach implementieren.
- Wiederholungsbetrieb entfällt, dadurch ist die Fehlerbehandlung schneller und Zeitverhalten unterscheidet sich nicht im Fehlerfall vom Normalfall ➔ besonders gut geeignet für harte Echtzeitsysteme
- Fehlerhafte Subsystemexemplare dürfen beliebiges fehlerhaftes Verhalten zeigen, da Relativtests angewandt werden.
- Fehlermaskierung ist transparent zu implementieren.
- Hoher Aufwand durch strukturelle Redundanz

Probleme bei Fehlermaskierung diversitärer Systeme



Erläuterung des Beispiels

- Das System berechnet zu unterschiedlichen Zeitpunkten neue Werte für Antrieb (Bremsen, Beschleunigen, konstant fahren) und Lenkung (gerade, links, rechts).
- Zum Zeitpunkt t1 wird ein neuer Wert für die Lenkung berechnet: alle Varianten entscheiden sich für geradeaus fahren.
- Zum Zeitpunkt t2 wird ein neuer Wert für den Antrieb berechnet: die Mehrheit (Variante 1 und 2) entscheiden sich für konstant fahren.
- Zum Zeitpunkt t3 wird ein neuer Wert für die Lenkung berechnet: die Mehrheit (Variante 1 und 3) entscheidet sich für geradeaus fahren.

  → es kommt zur Kollision
- Forderung: Varianten, die überstimmt wurden, müssen für eine bestimmte Zeit ausgeschlossen werden.

Synchronisation von redundanten Einheiten

- Alternativen:
  - Gesteuerte Synchronisierung: Steuerung von zentraler Stelle
  - Geregelte Synchronisierung: Synchronisation durch Maskierer
  - Implizite Synchronisierung: anwendbar falls die Auftragsrate die Bearbeitungsrate stets unterschreitet oder Ergebnisse verschiedener Aufträge vergleichbar sind

Reparatur und Integration von redundanten Einheiten

- Wie bereits gesehen sinkt die Zuverlässigkeit eines redundanten Systems nach einer bestimmten Zeitdauer immer unter die Zuverlässigkeit eines einfach ausgelegten Systems falls keine Reparaturen möglich sind.
- Zuverlässige Systeme mit langen geplanten Betriebszeiten müssen deshalb Reparaturen unterstützen.
- Ablauf:
  - Erkennen eines Fehlers
  - Ausgliedern der fehlerhaften Einheit
  - Zeitunkritische Durchführung von Fehlerdetektions- und Fehlerbehebungsalgorithmen
  - Wiedereingliederung (Integration)
- Wie kann sich eine Einheit wieder integrieren (state synchronisation) ohne den normalen Betrieb zu stören?

Fehlerkorrektur

- **Fehlerkorrektur** bildet einzelne fehlerhafte Ergebnisse, die genügend Informationsredundanz enthalten auf fehlerfreie ab.
- Basis ist eine Einschränkung in der Fehlfunktionsannahme (zumeist k-Binärstellen-Ausfall)

  → Anwendungsbereich vor allem, wo physikalische Gesetze diese Annahme rechtfertigen:
  - bei der Übertragung und
  - bei der Speicherung von Daten

Vor- und Nachteile der Fehlerkorrektur

- Strukturelle Redundanz und Zeitredundanz werden nur im geringen Umfang benötigt. ➔ gut geeignet für Echtzeitsysteme
- Die erforderliche Informationsredundanz ist im Allgemeinen mit geringen Aufwand zu erzeugen und zu überprüfen, allerdings können die dadurch verbundenen Kosten (z.B. zur Speicherung bzw. Senden) dem Einsatz entgegenstehen:
  - können die Daten etwa durch erneutes Senden wieder hergestellt werden, so wird häufig auf die Fehlerkorrektur verzichtet und ausschließlich Fehlererkennungsalgorithmen eingesetzt (z.B. CRC)
  - ist die Datenwiederherstellung nicht möglich und handelt es sich um wichtige Daten werden Fehlerkorrekturmaßnahmen verwendet (z.B. RAID)
- Bezüglich der Fehlervorgabe weist der Absoluttest eine hohe Fehlererfassung auf.
- Fehlerkorrektur lässt bei der Fehlervorgabe keine beliebigen Ergebnisverfälschungen zu.
- Im Allgemeinen kein geeignetes Mittel um Entwurfsfehler zu korrigieren.