Fehlertoleranz in eingebetteten Systemen

Thomas Klöber

Ausgewählte Kapitel eingebetteter Systeme (AKES)

19.07.2006

Überblick

Fehler – Definition

„Ein Fehler ist eine Abweichung von einem optimalen oder normierten Zustand oder Verfahren in einem bezüglich seinen Funktionen determinierten System.“

(Wikipedia)

Fault – Failure - Error

„Fehler“ = ...?

„Fault“ : Fehlfunktion, Störung

„Error“ : Irrtum, Abweg

„Failure“ : Ausfall, Versagen

Fehlertoleranz = fault-tolerance

Fehlerklassen

Hardware-Fehler:

- Versagen oder Fehlfunktion einer Komponente

Software-Fehler:

- Fehlerhaft programmierte Anwendung

Kommunikations-Fehler:

- Fehlerhafte Übertragung von Daten

Überblick

Tolerieren von Hardware-Fehlern

Prinzip:

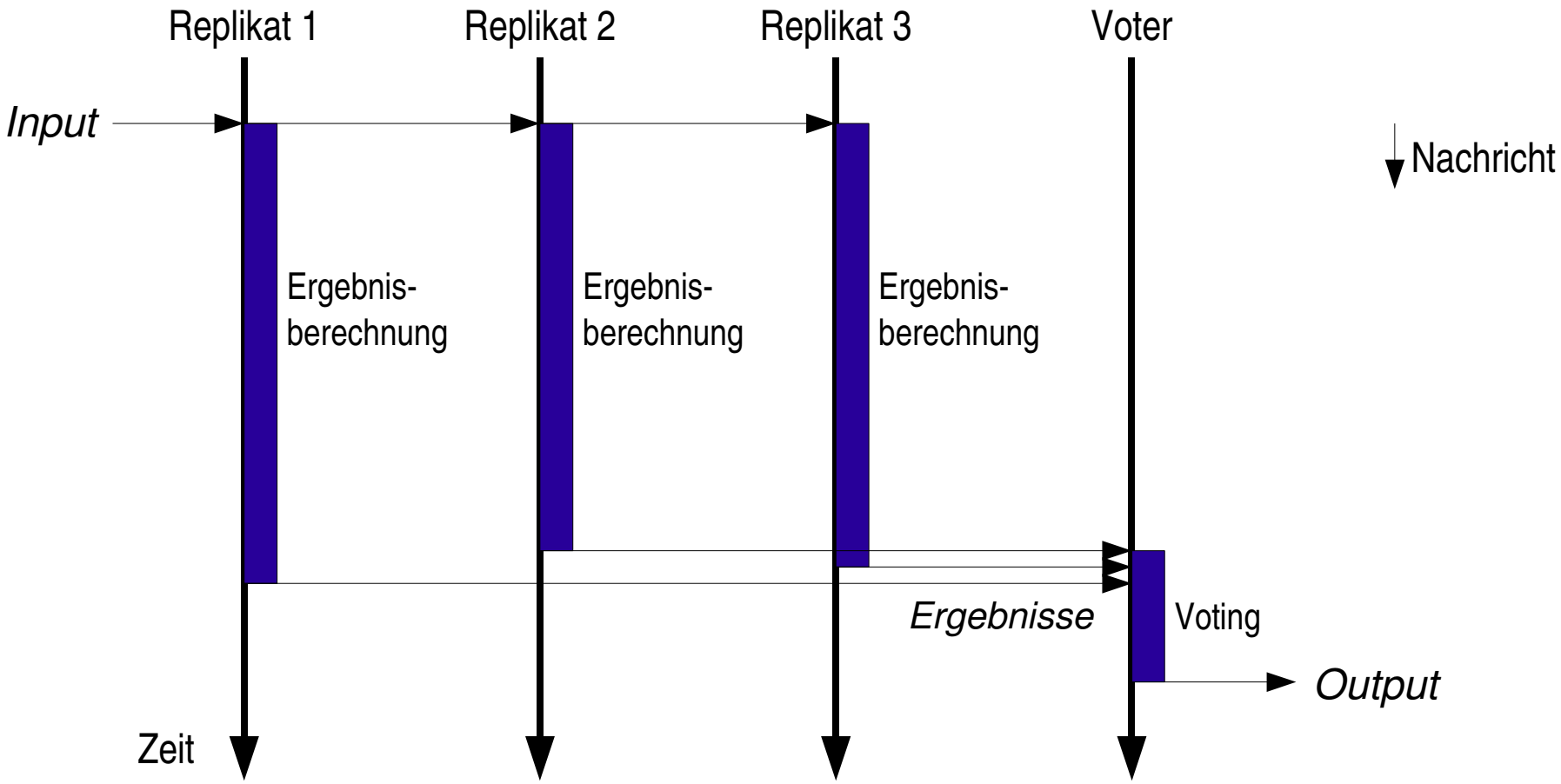
- Prozess wird repliziert
- Hardware mehrfach vorhanden
- Replikate werden verteilt

Aktive Replikation (1)

Prinzip:

- alle Replikate gleichzeitig aktiv
- Ergebnisse werden simultan erzeugt
- Voter leitet mehrheitlich erzieltes Ergebnis weiter

Aktive Replikation (2)



Aktive Replikation (3)

Fehlererkennung:

- Ergebnis stimmt nicht mit Mehrheit überein
- Ergebnis wird zu spät erzeugt
- Ergebnis wird gar nicht erzeugt

Aktive Replikation (4)

Bedingungen:

- Eingabe-Konsistenz
- Replikations-Determinismus

Aktive Replikation (5)

Replikationsgrad:

- 2f+1 Knoten zur Tolerierung von f Fehlern
- Standardfall: f = 1 (triple modular redundancy)

„fail-silent“ Knoten

Besonderheiten:

- Sendung fehlerhafter Ergebnisse wird verhindert
- kein Voter notwendig
- nur f+1 Knoten zur Tolerierung von f Fehlern nötig

Passive Replikation (1)

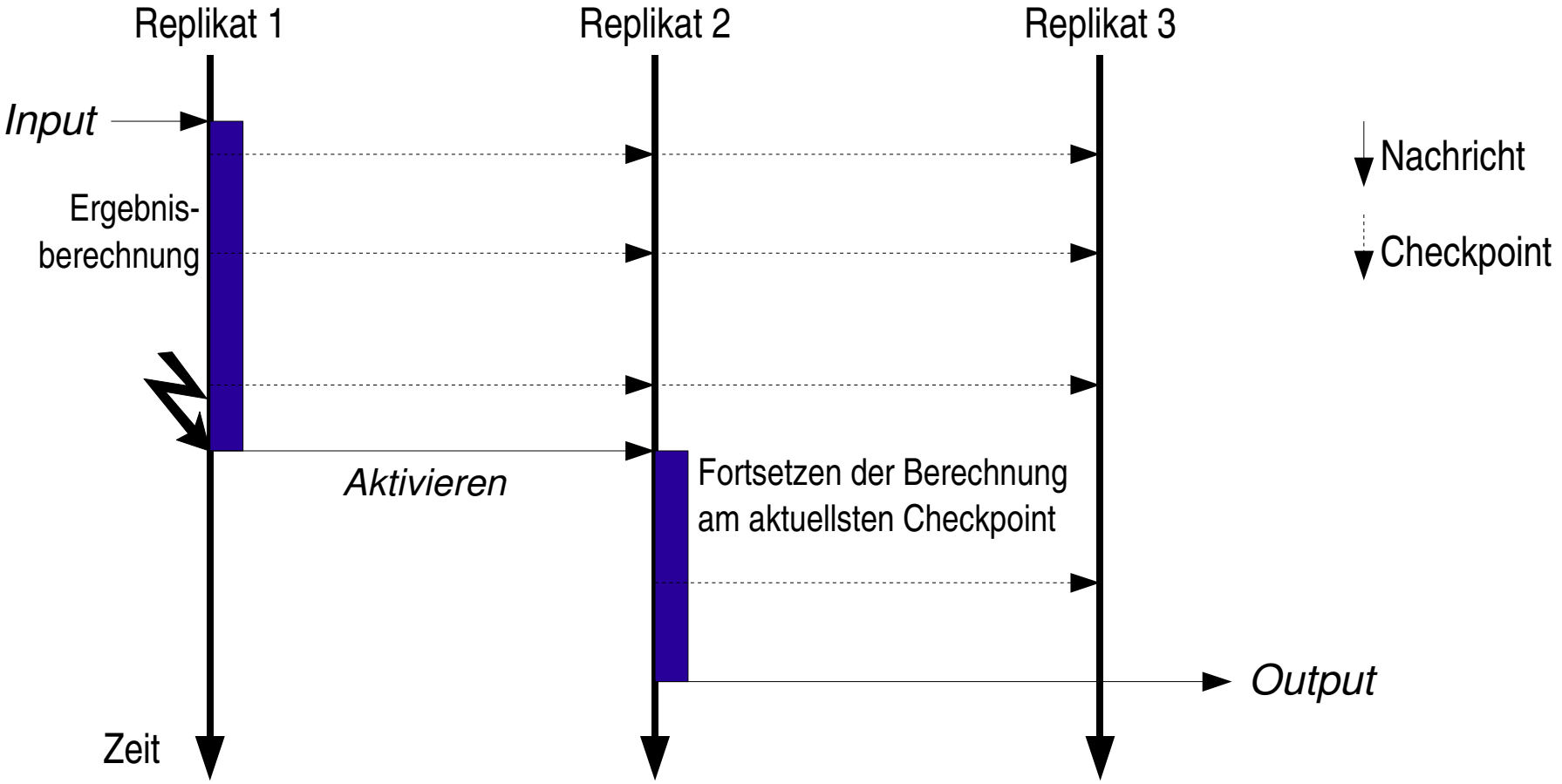
Prinzip:

- nur ein Knoten aktiv
- Zustand wird zu bestimmten Zeiten (Checkpoints) auf die Replikate übertragen
- im Fehlerfall Aktivieren eines anderen Knoten

Voraussetzung:

- alle Knoten sind „fail-silent“

Passive Replikation (2)



Passive Replikation (3)

mögliches Problem:

- doppeltes Versenden von Nachrichten durch Rollback

Gegenmaßnahmen:

- systematische Checkpoints
- periodische Checkpoints

Passive Replikation (4)

Systematische Checkpoints:

- Checkpoint nach jedem Versenden einer Nachricht
- Rollback erfordert niemals erneutes Senden

Passive Replikation (5)

Periodische Checkpoints:

- weniger Checkpoints, z.B. alle n Nachrichten
- nach Rollback jede zu Sendende Nachricht erst mit Log vergleichen

Voraussetzungen:

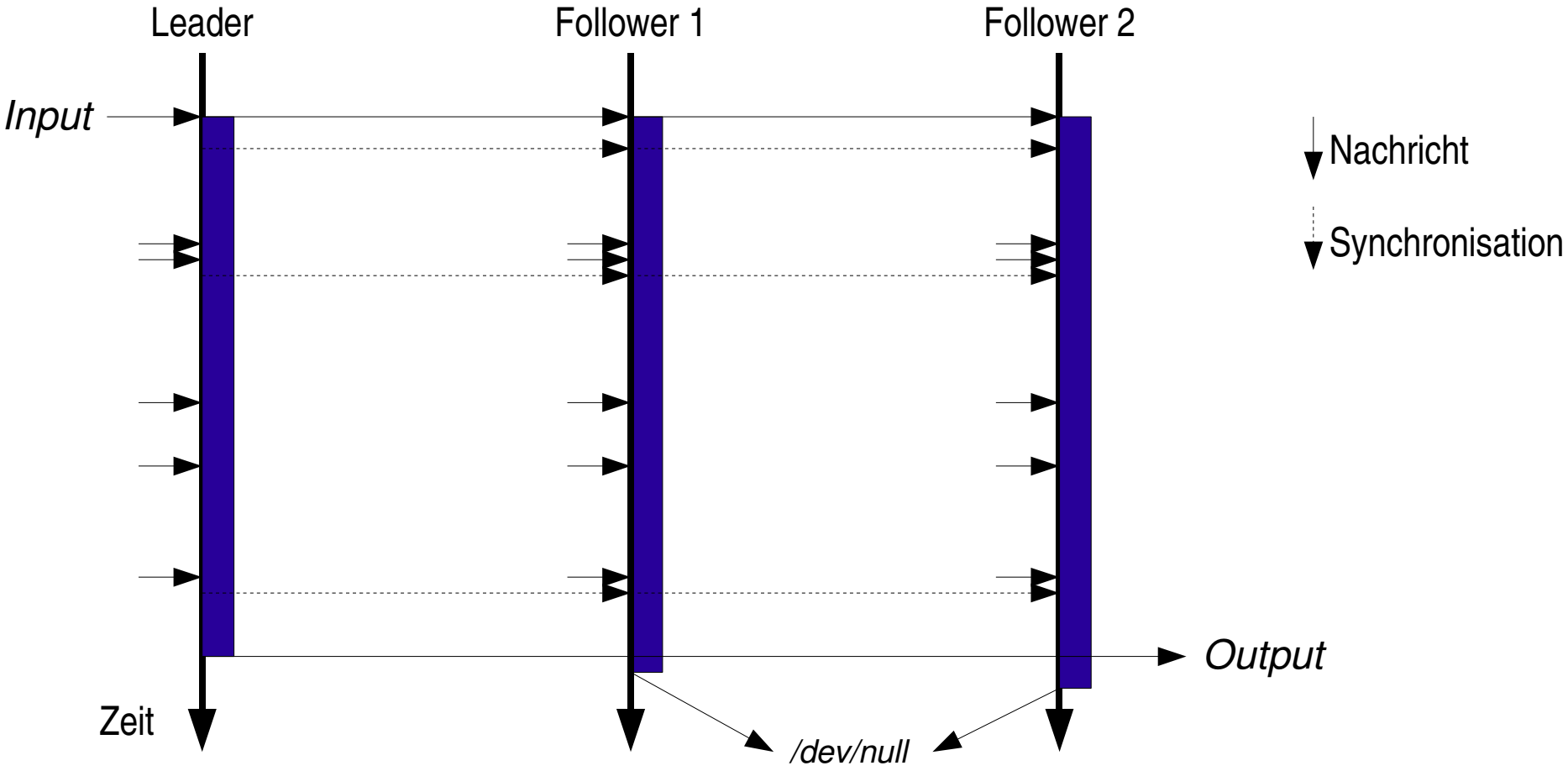
- Replikations-Determinismus
- Eingabe-Konsistenz

Leader/Follower-Replikation (1)

Prinzip:

- alle Knoten aktiv
- ein Knoten Leader, alle anderen Follower
- kein Voting, Leader übermittelt Ergebnis
- Leader trifft alle Entscheidungen und gibt sie via Synchronisations-Nachricht an Follower weiter
- defekter Leader wird durch einen Follower ersetzt

Leader/Follower-Replikation (2)



Leader/Follower-Replikation (3)

Synchronisations-Nachrichten:

- Input-Synchronisation: in welcher Reihenfolge sind Nachrichten zu konsumieren

  (⇒ Eingabe-Konsistenz gewährleistet)
- Preemption-Synchronisation: zu welchem Zeitpunkt darf der Prozess unterbrochen werden

  (⇒ Replikations-Determinismus gewährleistet)

Vergleich der Techniken (1)

Aktive Replikation:

- Vorteile: keine Unterbrechung im Fehlerfall, falls Knoten „fail-silent“ keine Verzögerung und kein Voter nötig
- Nachteile: atomares Multicasting muss unterstützt werden, Prozesse müssen replikations-deterministisch sein, Unterbrechungen sehr schwer zu handhaben, hoher Rechenaufwand

Vergleich der Techniken (2)

Passive Replikation:

- Vorteile: Unterbrechungen problemlos, einfache Kommunikation, Prozesse müssen nicht deterministisch sein, kein Voter nötig, geringer Rechenaufwand
- Nachteile: Knoten zwingend „fail-silent“, Verzögerung im Fehlerfall, Kommunikations-Overhead durch Checkpoint-Erstellung

Vergleich der Techniken (3)

Leader/Follower-Replikation:

- Vorteile: kein Voter nötig, weniger Overhead als passive Replikation, Unterbrechungen kein Problem, nicht-deterministische Prozesse möglich
- Nachteile: Verzögerung beim Ausfall des Leaders, Knoten zwingend „fail-silent“, hoher Rechenaufwand trotz zusätzlicher Kommunikation

Überblick

Einführung

Hardware-Fehler

Software-Fehler

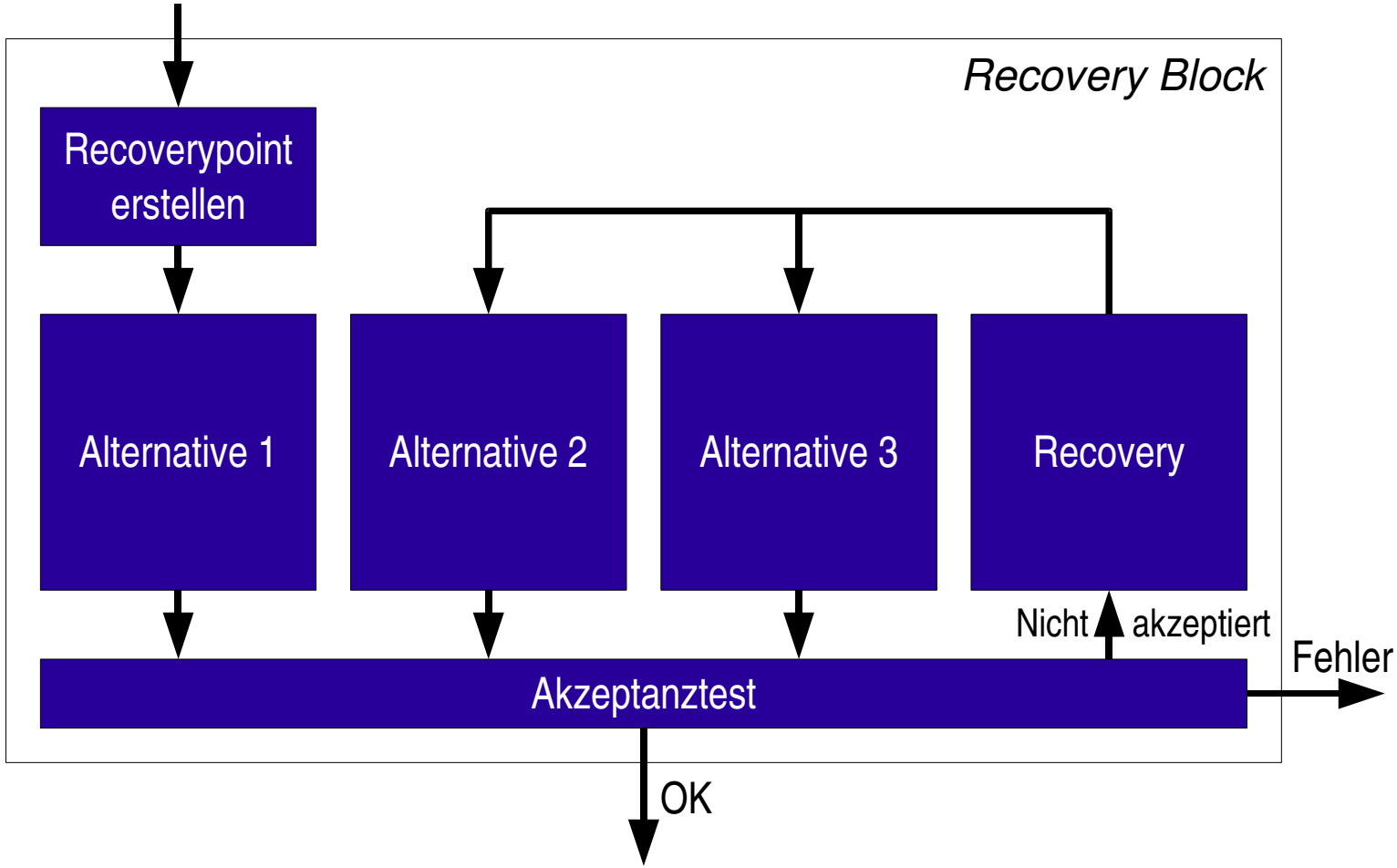
- Recovery-Blocks
- N-Version-Programming

Recovery-Blocks (1)

Prinzip:

- unabhängige Entwicklung mehrerer Alternativen nach gleicher Spezifikation
- Akzeptanztest, der Korrektheit der Ergebnisse prüft
- Ausführen der Alternativen nacheinander, bis Ergebnis Akzeptanztest besteht

Recovery-Blocks (2)



Recovery-Blocks (3)

Problem:

- Alternative, die Nachrichten an andere gesendet hat, schlägt fehl
- verschickte Nachrichten sind ungültig, also alle Konsumenten fehlgeschlagen
- Konsumenten müssen identifiziert werden

Recovery-Blocks (4)

Mögliche Maßnahmen:

- Keine Inter-Prozess-Kommunikation in Recovery-Blocks $\Rightarrow$ zu restriktiv
- Verwendung von Dialogen

Recovery-Blocks (5)

Dialoge:

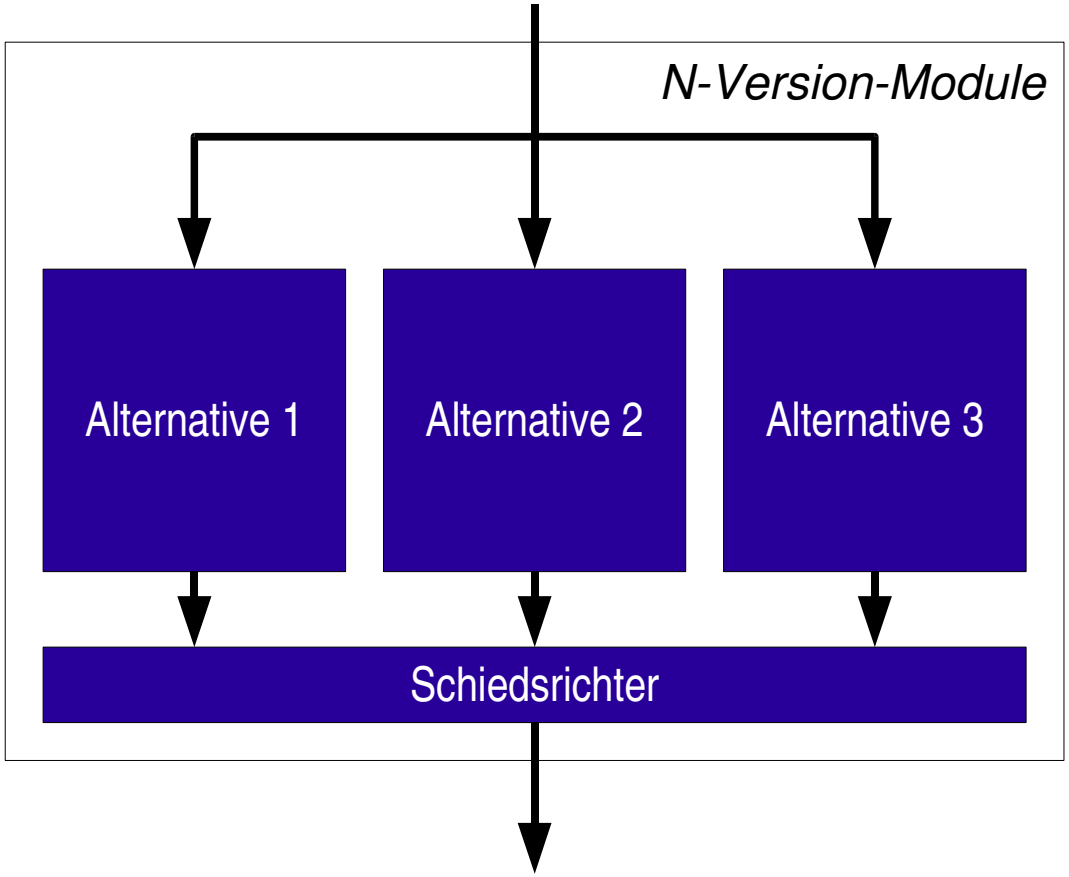
- enthalten Prozesse und Datenstrukturen
- ermöglichen Kommunikation ohne Recovery zu beeinträchtigen
- falls Alternative fehlschlägt werden betroffene Prozesse auch wiederhergestellt
- terminiert, wenn alle enthaltenen Prozesse erfolgreich

N-Version-Programming (1)

Prinzip:

- unabhängige Entwicklung mehrerer Alternativen nach gleicher Spezifikation
- Ausführung aller Alternativen nebenläufig
- Schiedsrichter konstruiert endgültiges Ergebnis aus allen Einzelergebnissen

N-Version-Programming (2)



N-Version-Programming (3)

Problem:

- Verhalten nach Fehlschlagen einer Version
- Falls Modul zustandslos: kein Problem
- sonst: Modul in inkonsistentem Zustand

N-Version-Programming (4)

Maßnahmen:

- Klonen der fehlgeschlagenen Version
- Wiederherstellung des Zustands
  - durch Untersuchung anderer Komponenten
  - durch Abfrage bei anderen Versionen, dann Standard-Repräsentation des Zustands nötig

Gibt es noch Fragen?

Literaturverzeichnis

- http://de.wikipedia.org/wiki/Fehler
- Barret P. A. , Speirs N. A. 1993:
  Towards an integrated approach to fault tolerance in Delta-4,
  IEE Distributed Systems Engineering, Volume 1, Issue 2, pp 59-66,
  IOP Publishing Ltd.
- Kopetz H., Bauer G. 2003:
  The Time-Triggered Architecture *Proceedings of the IEEE, Special Issue on Modeling and Design of Embedded Software,* pp 112-126