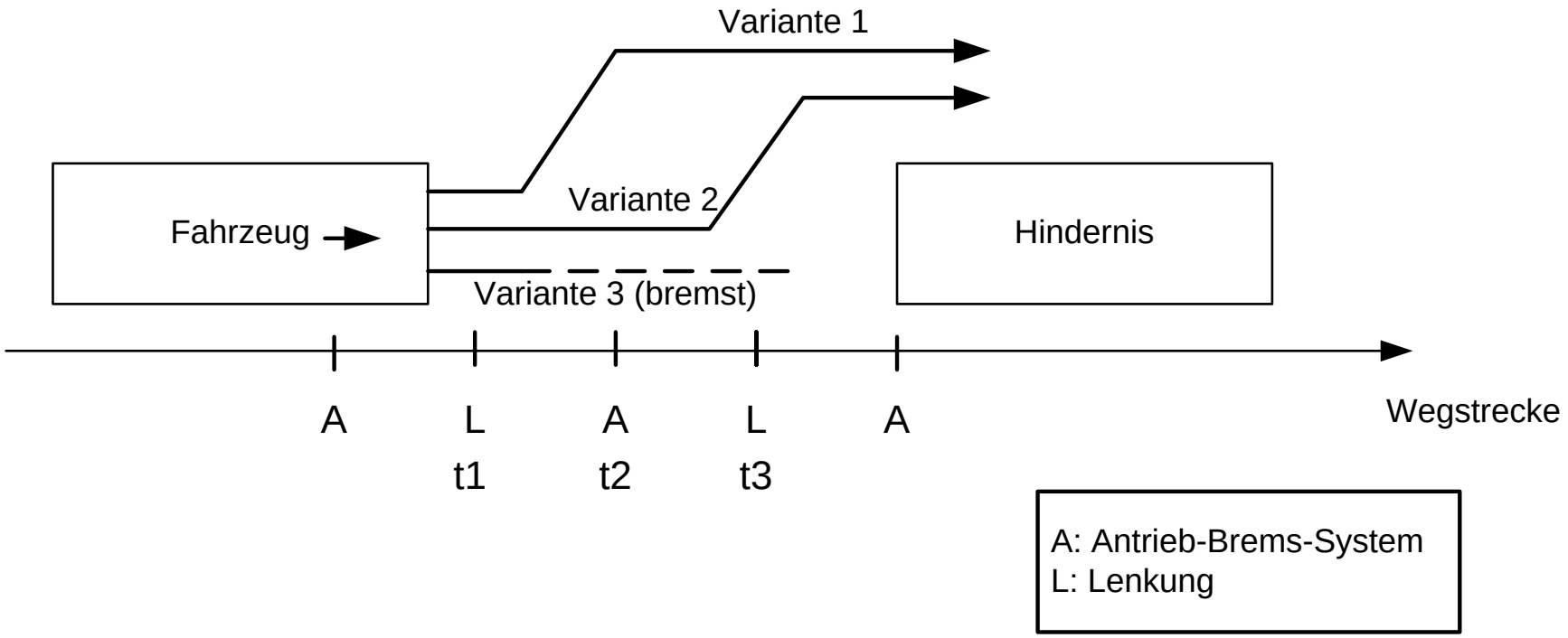
Vor- und Nachteile der Fehlermaskierung

- Fehlermaskierung reicht als **einziges** Fehlertoleranz-Verfahren aus.
- Maskierer lassen sich vergleichsweise einfach implementieren.
- Wiederholungsbetrieb entfällt, dadurch ist die Fehlerbehandlung schneller.
- 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.

  $\Rightarrow$ 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)

  $\Rightarrow$ 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.
- 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.

Fehlertoleranz

Zusammenfassung

Lerninhalt Fehlertoleranz

- Wesentliche Elemente / Phasen der Fehlertoleranz
  - Fehlererkennung
  - Fehlerdiagnose
  - Fehlerbehandlung / Fehlerbehebung (Reparatur)
- Die Auswahl und Realisierung der Fehlertoleranzmechanismen basiert immer auf der Fehlerhypothese (definiert Menge und Art der zu tolerierenden Fehler)
- Kenntnis der Mechanismen und Vergleich in Bezug auf zu tolerierende Fehler und Echtzeitfähigkeit

Ausblick

Ausblick

- Seminare:
  - Modellbasierte Entwicklung eingebetteter Systeme
  - Sensornetzwerke
  - Software Engineering für eingebettete Multicore-Systeme
- Praktikum:
  - Echtzeitsysteme
- Studienarbeiten (SEP, BA, MA, DA, Guided Research):
  - Sensornetzwerke
  - Mikroprozessorprogrammierung
  - Elektrofahrzeug
  - Fehlertolerante Systeme
- Studentische Hilfskräfte / Promotion (Lehrstuhl bzw. fortiss)