Verlässliche Echtzeitsysteme

Übungen zur Vorlesung

Florian Franzmann, Martin Hoffmann

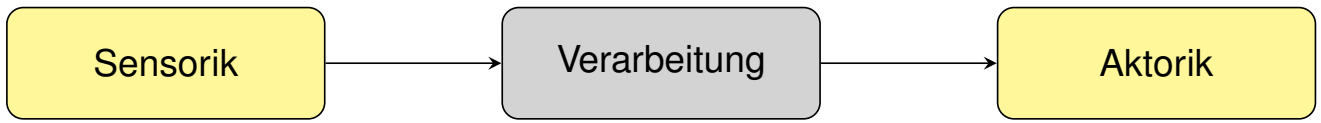
Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
www4.informatik.uni-erlangen.de

18. Juni 2012

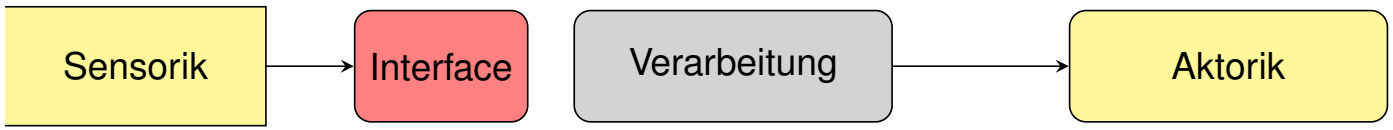
Überblick

1. Wiederholung Software-TMR
2. Eliminierung von Bruchstellen in TMR
3. Aufgabenstellung

Klassische “Triple Modular Redundancy” (TMR)

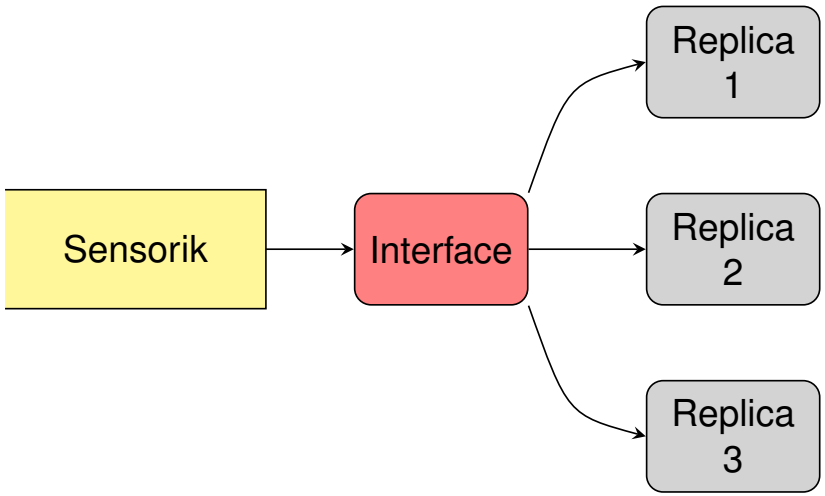


Klassische “Triple Modular Redundancy” (TMR)



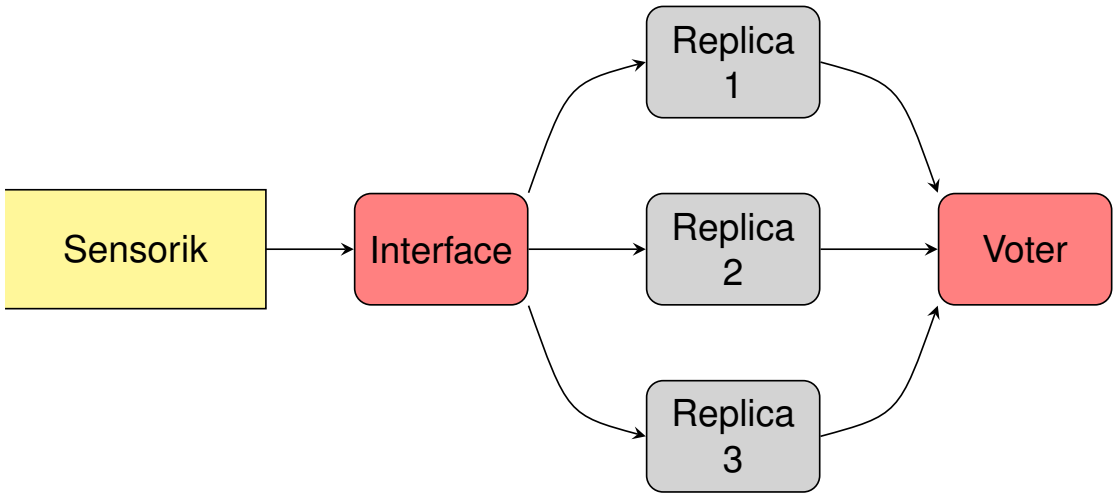
- Schnittstelle sammelt Eingangsdaten (Replikationsdeterminismus)

Klassische “Triple Modular Redundancy” (TMR)



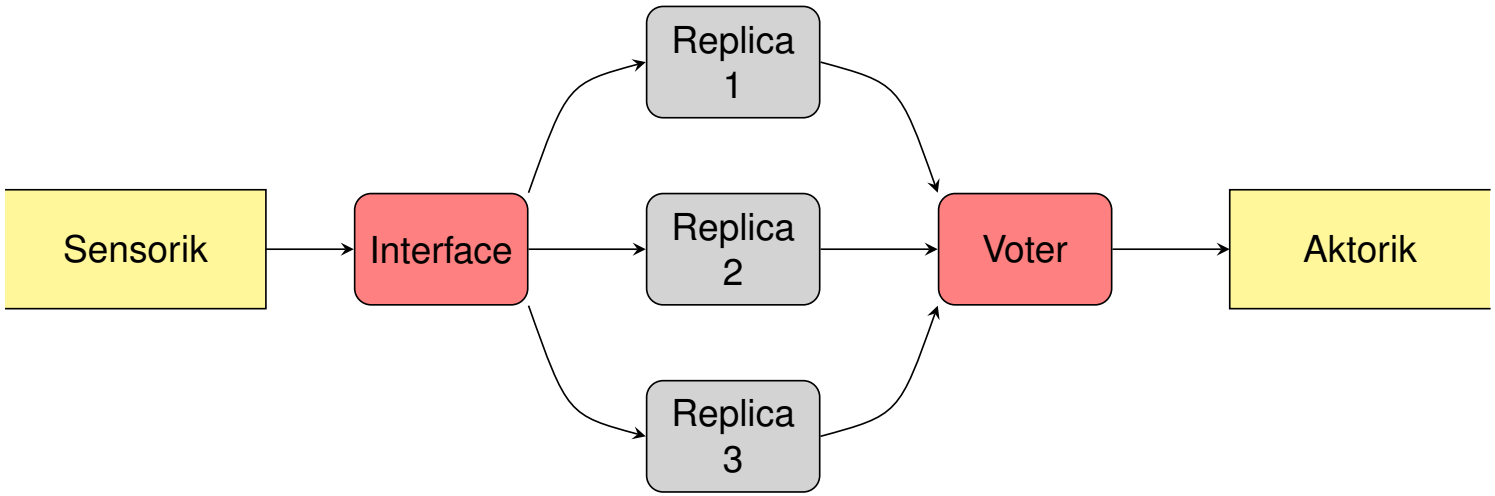
- Schnittstelle sammelt Eingangsdaten (Replikationsdeterminismus)
- Verteilt Daten und aktiviert Replikate

Klassische “Triple Modular Redundancy” (TMR)



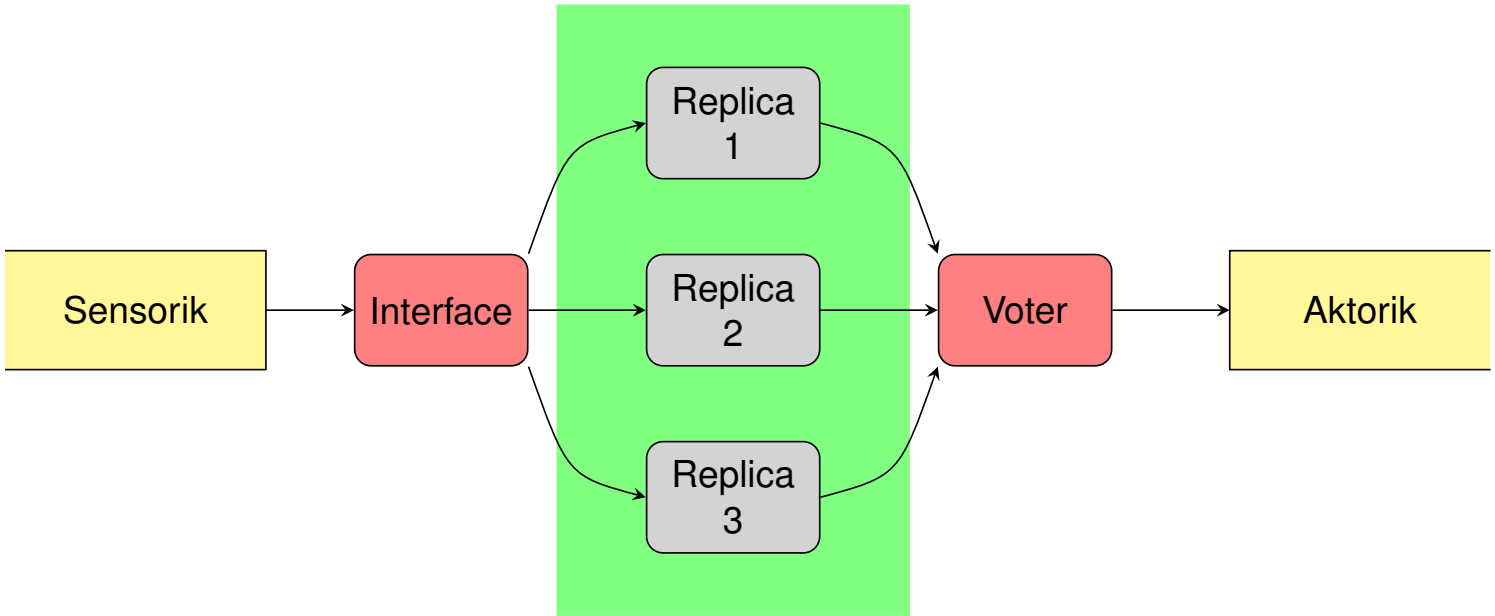
- Schnittstelle sammelt Eingangsdaten (Replikationsdeterminismus)
- Verteilt Daten und aktiviert Replikate
- Mehrheitsentscheider (Voter) wählt Ergebnis

Klassische “Triple Modular Redundancy” (TMR)



- Schnittstelle sammelt Eingangsdaten (Replikationsdeterminismus)
- Verteilt Daten und aktiviert Replikate
- Mehrheitsentscheider (Voter) wählt Ergebnis
- Ergebnis wird an Aktuator versendet

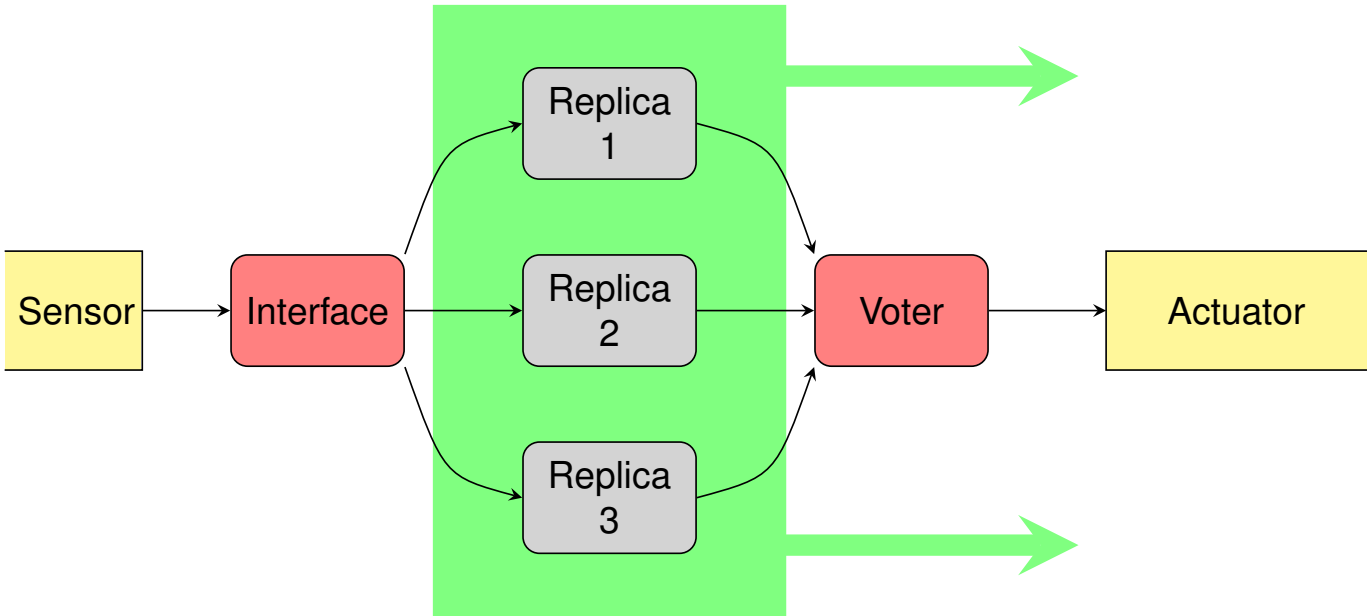
Klassische “Triple Modular Redundancy” (TMR)



Redundanzbereich

Ausschließlich Replikatausführung.

Erweiterung I – kodierte Ausgangswerte



- Erweiterung der Ausgangsseite mit Informationsredundanz
- Mehrheitsentscheid über kodierte Prüfsumme

Table of Contents

1 Wiederholung Software-TMR

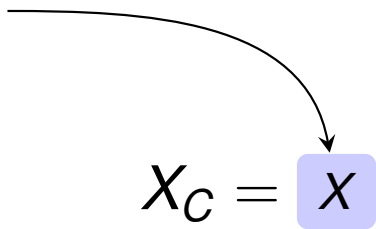
2 Eliminierung von Bruchstellen in TMR

3 Aufgabenstellung

Erweiterte arithmetische Kodierung

nach Forin 1989: “Vital coded microprocessor principles and application for various transit systems” [**?**]

- Arithmetisch kodierter Wert X_C
- Ausgangswert



Erweiterte arithmetische Kodierung

nach Forin 1989: “Vital coded microprocessor principles and application for various transit systems” [**?**]

- Arithmetisch kodierter Wert X_C
- Ausgangswert
- Primzahl

$$X_C = X * A$$

Bitfehlererkennung
(Restfehlerwahrscheinlichkeit
$P = 1/A$)

Erweiterte arithmetische Kodierung

nach Forin 1989: “Vital coded microprocessor principles and application for various transit systems” [**?**]

- Arithmetisch kodierter Wert X_C
- Ausgangswert
- Primzahl
- Variablenspezifische Signatur

$$X_C = X * A + B_X$$

Adressierungsfehlererkennung

Erweiterte arithmetische Kodierung

nach Forin 1989: “Vital coded microprocessor principles and application for various transit systems” [**?**]

- Arithmetisch kodierter Wert X_C
- Ausgangswert
- Primzahl
- Variablenspezifische Signatur
- Zeitstempel

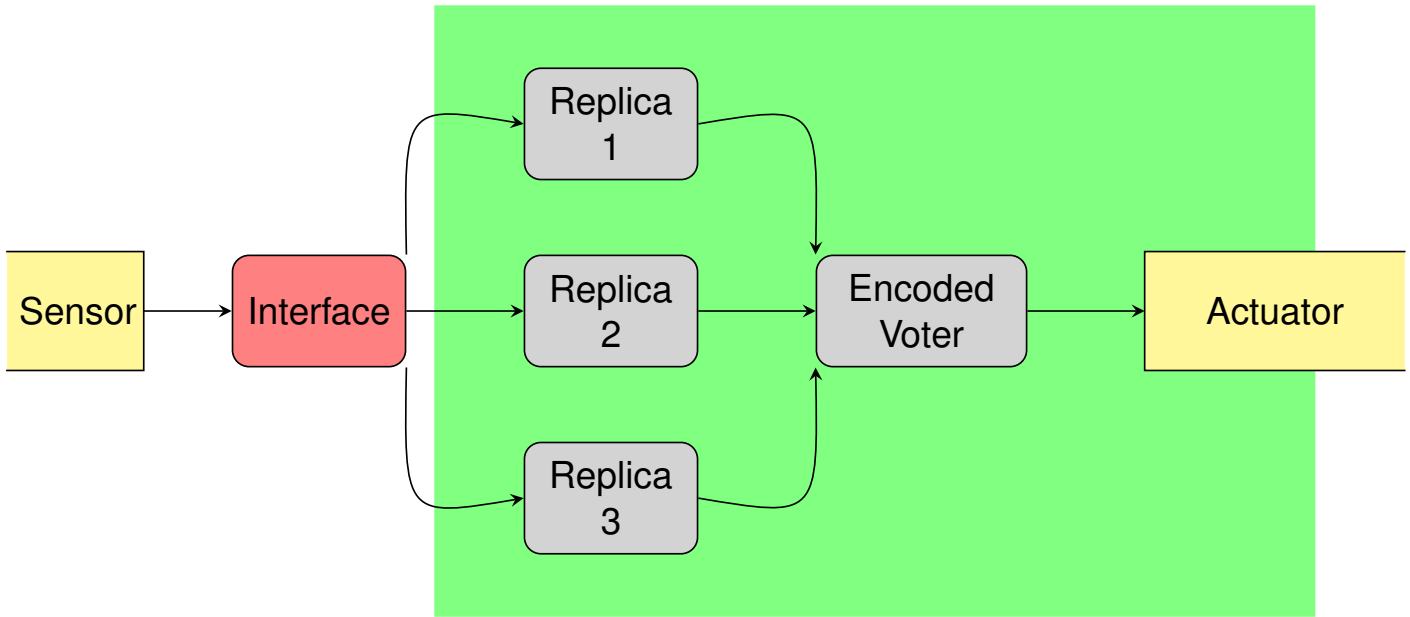
$$X_C = X * A + B_X + T$$

Erkennung veralteter Daten

Wertebereichseinschränkungen

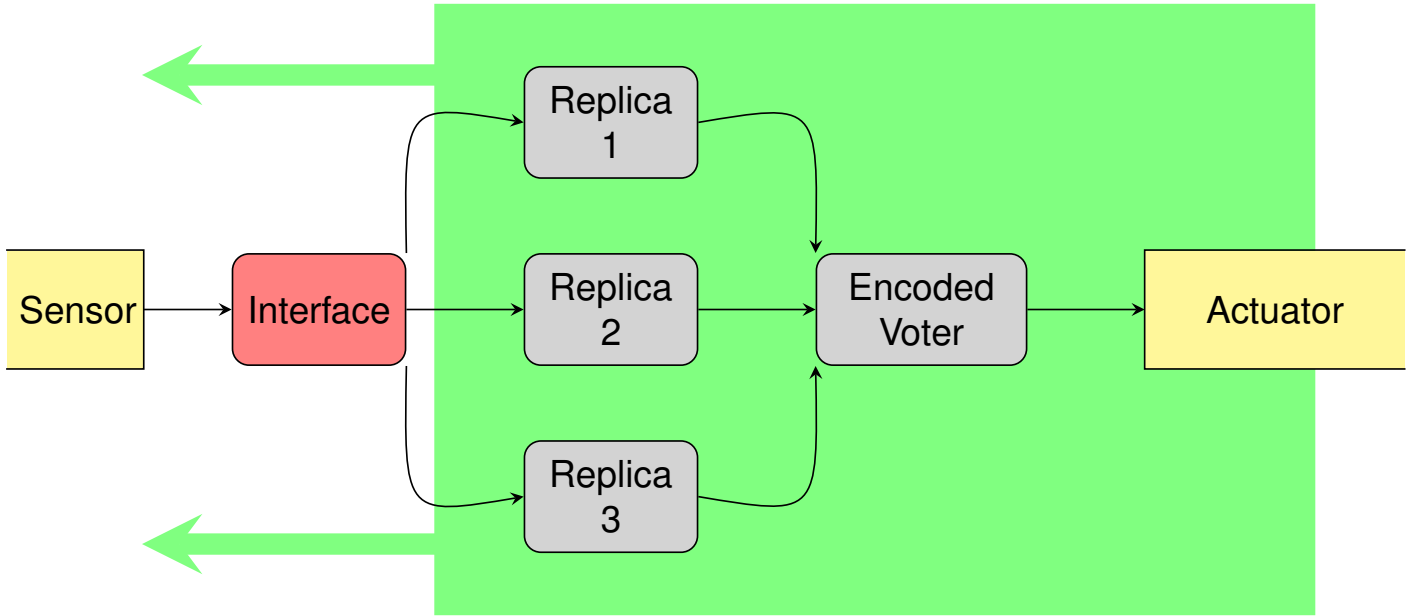
- Primzahl A sollte so groß wie möglich sein:
  - $\rightsquigarrow$ Möglichst geringe Restfehlerwahrscheinlichkeit ($P = 1/A$)
- Wertebereich des dynamischen Zeitstempels
  - $T = \{x | x \in \mathbb{N}_0 \wedge x \leq D_{max}\}$
  - Zeitstempel darf überlaufen: $D_{max} + 1 = 0$
- Für jede Signatur B_* muss dann gelten
  - $B_* + D_{max} < A$
  - Die minimale Distanz zwischen jeweils zwei Signaturen im System muss kleiner D_{max} sein: $\forall i, j : |B_i - B_j| < D_{max}$

Erweiterung I – kodierte Ausgangswerte

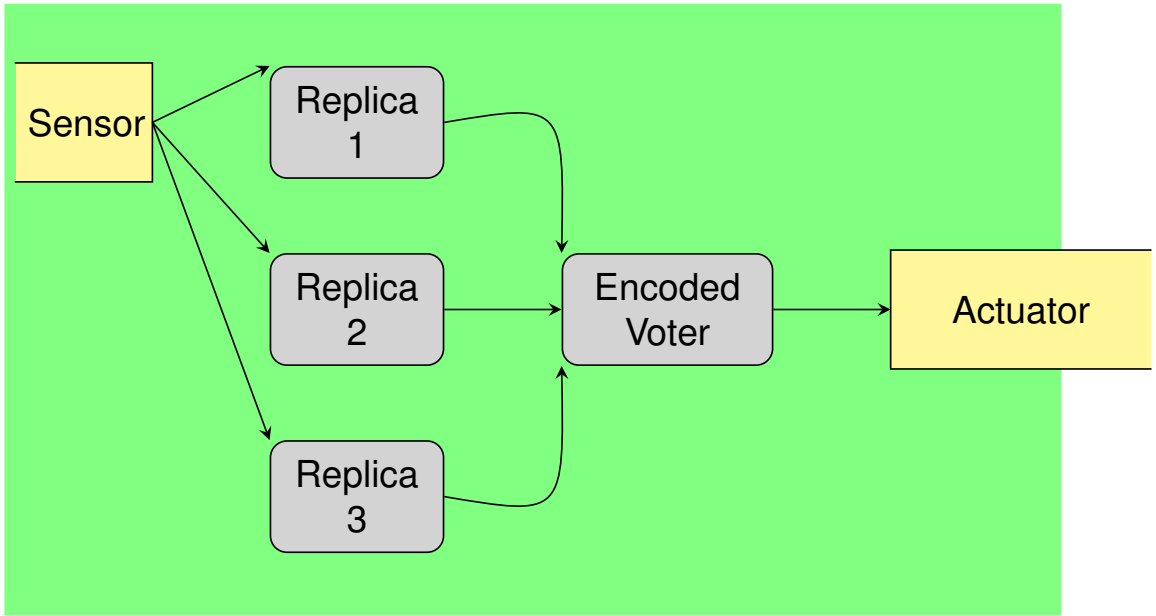


- Replikate liefern arithmetisch kodierte Ergebnisse
- Mehrheitsentscheid auf kodierten Prüfsummen
- Übertragung kodierter Ergebnisse

Erweiterung II – Datendiversität

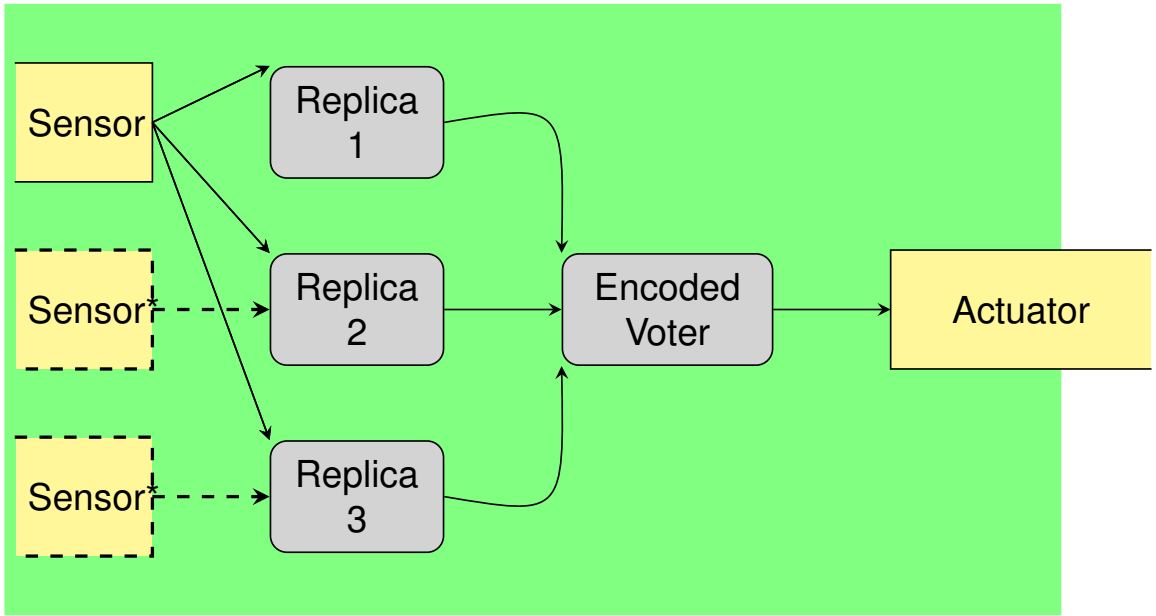


Erweiterung II – Datendiversität



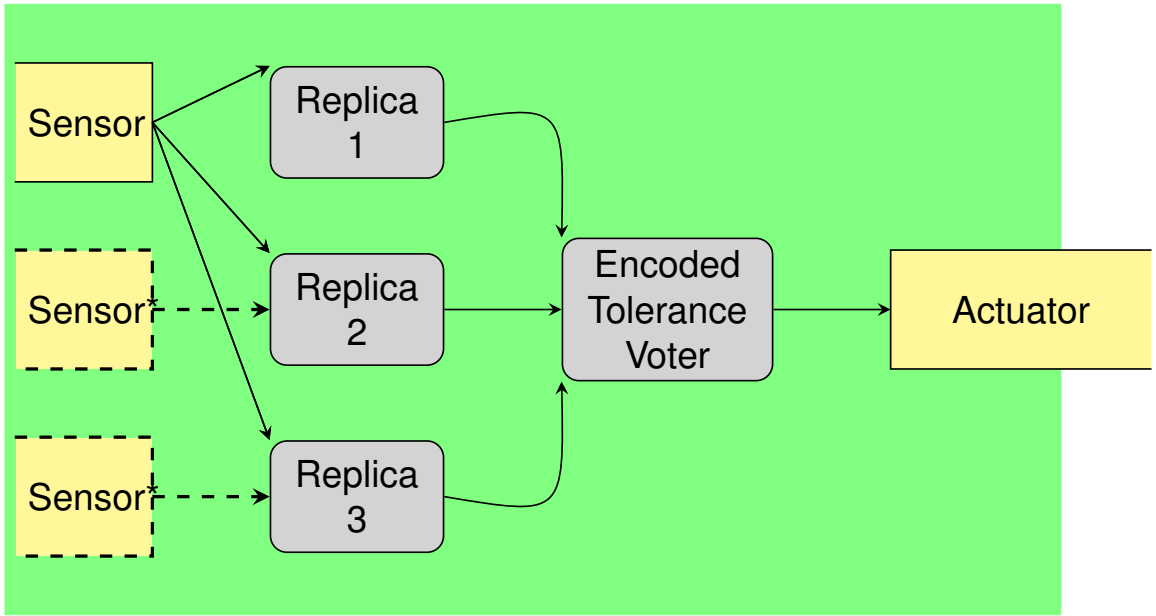
- Replikate ermitteln Eingangsdaten selbständig
- Diversitäre Eingangsdaten
  - Unterschiedliche Messzeitpunkte (zeitliche Redundanz)

Erweiterung II – Datendiversität



- Replikate ermitteln Eingangsdaten selbständig
- Diversitäre Eingangsdaten
  - Unterschiedliche Messzeitpunkte (zeitliche Redundanz)
  - Redundante Sensoren (physikalische Redundanz)

Erweiterung II – Datendiversität



- Replikate ermitteln Eingangsdaten selbständig
- Diversitäre Eingangsdaten
  - Unterschiedliche Messzeitpunkte (zeitliche Redundanz)
  - Redundante Sensoren (physikalische Redundanz)
- Mehrheitsentscheid mittels Toleranzbereich (Tolerance Voter)

Vereinfachung für diese Übung

Für diese Übungsaufgabe:

- Keine Datendiversität am Eingang
- Nur Absicherung der Ausgangsseite!

EAN Vergleichsoperator

- Voting basiert auf kodierter Vergleichsoperation:

$\rightsquigarrow X_C = Y_C \Rightarrow X * A + B_X + T_X = Y * A + B_Y + T_Y$

- Im fehlerfreien Fall gilt:

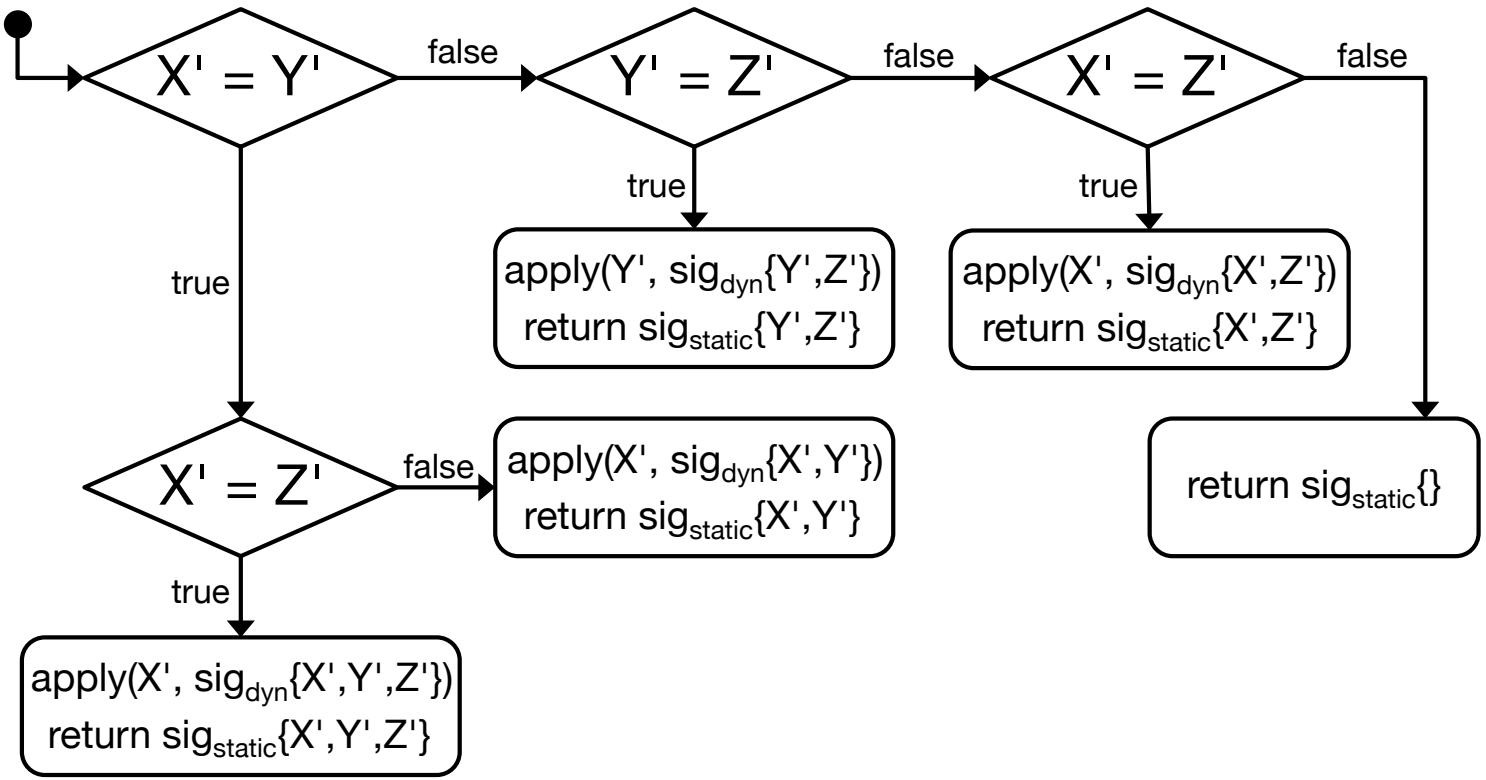
$X = Y$, $T_X = T_Y$, $A = A$ aber $B_X \neq B_Y$!

- *Rohwerte* sind identisch
- Ergebnisse sind aktuell
- Primzahl ist per Definition identisch
- Signaturen sind unterschiedlich (aber konstant!)

Bestimmung der Gleichheit durch Differenzbildung:

$\rightsquigarrow X_C - Y_C = B_X - B_Y = const.$

Kodierter Mehrheitsentscheid



- Bestimmung von dynamischer und statischer Signatur:

$\rightsquigarrow$ $sig_{dyn}(X', Y') : X' = Y' \Rightarrow X' - Y'$

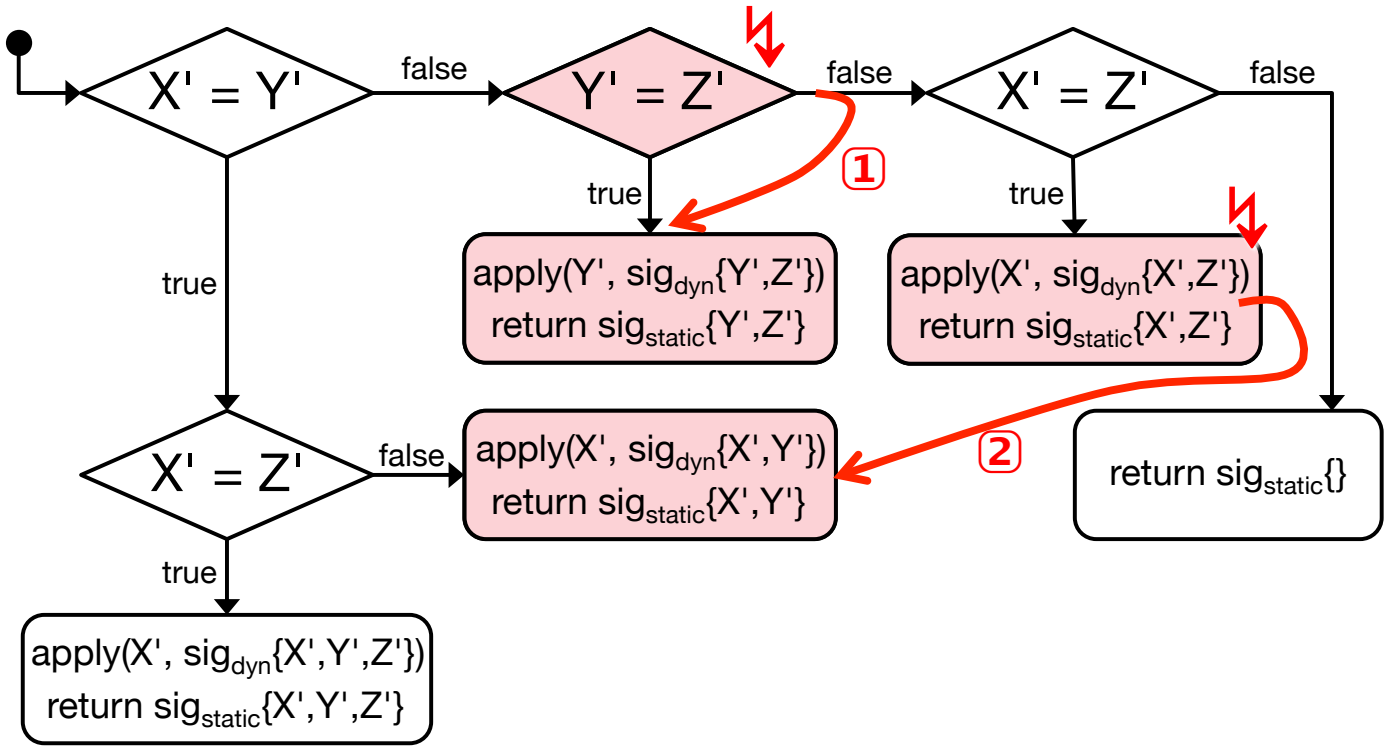
$\rightsquigarrow$ $sig_{static}(X', Y') : X' = Y' \Rightarrow B_X - B_Y$

Kodierter Mehrheitsentscheid (Forts.)

1. Vergleichsoperation wird durchgeführt (z.B.: $X' = Y'$, $X' = Z'$)
   - Berechnung von sig_{dyn}
   - Vergleich mit sig_{static}
2. Verzweigunsentscheidung wird nachberechnet:
   - Wiederholte (redundante) Berechnung von sig_{dyn}
   - Beaufschlagung (*apply*) auf gewähltes Ergebnis
3. Konstante Signatur des durchlaufenen Zweiges identifiziert Gewinner (Rückgabewert: sig_{static})
   - Aktor wählt entsprechendes Replikatergebnisse
   - führt inverse Operation zu *apply* durch

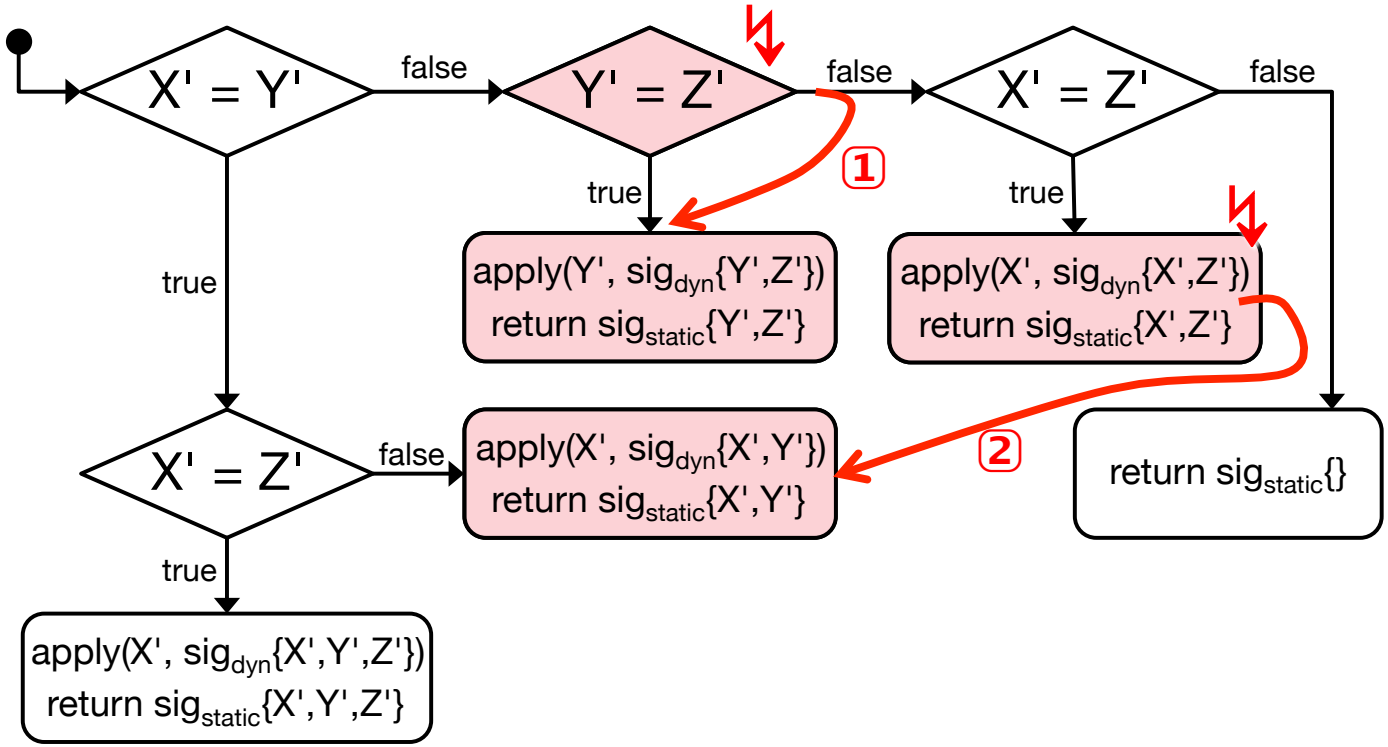
Im Voter wurde die *dynamisch berechnete Signatur der Verzweigungsentscheidung* beaufschlagt. Im Aktor wird mit der entsprechenden *konstanten Signatur zurückgerechnet*.

Kodierter Mehrheitsentscheid - Fehlerfall



1. Falsche Verzweigungsentscheidung: ($Y' \neq Z'$)
   - Y' wird als korrekt angenommen, sig_{dyn} wird erneut berechnet
   - allerdings ist sig_{dyn} tatsächlich $\neq sig_{static}$
   - Fehler wird bei der inversien Operation zu *apply* erkannt

Kodierter Mehrheitsentscheid - Fehlerfall



2. Falscher (plötzlicher) Sprung
   - X' wird als korrekt erkannt, sig_{dyn} wird erneut berechnet
   - Ein fehlerhafter Sprung ein einen anderen Block führt zu einem inkonsistenten Rückgabewert $sig_{static}\{X', Z'\}$
   - $sig_{dyn}\{X', Y'\} \neq sig_{static}\{X', Z'\}$ wird beim dekodieren erkannt

Table of Contents

1 Wiederholung Software-TMR

2 Eliminierung von Bruchstellen in TMR

3 Aufgabenstellung

Aufgabenstellung

Aufgabe

Erweitern Sie Ihre Software-TMR Implementierung um einen EAN kodierten Voter für die (Festkomma)-Filterimplementierung aus der vorherigen TMR Aufgabe.

- Jeder Regelschritt entspricht dabei einer Zeiteinheit
  - ⇝ Nutzen Sie einen globalen Zähler als Zeitgeber für T
    - Begrenzen Sie dabei T auf den Wertebereich $0 \ldots D_{max}$
- Jedes Replikat hat genau einen Ausgabewert (integer)
  - ⇝ Legen Sie für jede der drei Ausgabewerte (X', Y', Z') jeweils *unterschiedliche* aber *konstante* Signaturen (B_X, B_Y, B_Z) fest
- Nutzen Sie für X' den nächstgrößeren Datentyp zu X
  - ⇝ Wählen Sie eine *möglichst große* Primzahl A (> 200), *vermeiden Sie* dabei mögliche *Überläufe bei der Kodierung*

Hinweise

- In dieser Aufgabe betrachten wir nur die Ausgangsseite
- Nach dem Voting kann das Interface das Ergebnis gleich dekodieren/validieren
- Die Eingangsseite bleibt vorerst “ungeschützt”
- Für *jede Operation* zwischen zwei kodierten Werten benötigt man *eine eigene Funktion* mit konstanten Signaturwerten!

Fragen?