# Verlässliche Echtzeitsysteme

## Übungen zur Vorlesung

Phillip Raffeck, Florian Schmaus, Simon Schuster

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
https://www4.cs.fau.de

Sommersemester 2019

# Überblick

1. Fehlerinjektion mit FAIL*
2. Wiederholung Software-TMR
3. Eliminierung von Bruchstellen in TMR
4. Aufgabenstellung

# Überblick



# Fehlerinjektion mit FAIL*

- https://www4.cs.fau.de/Lehre/SS19/V_VEZS/Uebung/Folien/Fail.pdf

# FAIL* – Marker

```
1  #ifndef _include_fail_h
2  #define _include_fail_h
3
4  //Mark start of injection
5  void __attribute__((noinline)) fail_start_trace(void);
6  //Mark end of injection
7  void __attribute__((noinline)) fail_stop_trace(void);
8
9  //everything ok: valid code and right values
10 void __attribute__((noinline)) fail_marker_positive(void);
11 //everything ok: valid code but wrong values
12 void __attribute__((noinline)) fail_marker_negative(void);
13 //invalid code
14 void __attribute__((noinline)) fail_marker_detected(void);
15
16 #endif /* _include_fail_h */
```

- Markierung Start/Ende für „Golden Run”
- Ziel: Minimierung Auftreten von `fail_marker_negative`

# FAIL* – Beispiel: Verwendung Marker

```
1  void cyg_user_start() {
2    ...
3    fail_trace_start() // Start Fehlerinjektion
4    dostuff()
5    vote()             // (codierter) Mehrheitsentscheid
6    fail_trace_stop()  // Ende Fehlerinjektion
7
8    actuate()          // Signaturen entfernen, Fehler überprüfen
9
10   ...
11 }
```

```
1  void actuate() {
2    ...
3    if (voted_output == expected_output){
4      fail_marker_positive() // SDC behandelt :-)
5    }else{
6      fail_marker_negative() // undetektierte SDC :-(
7    }
8  }
9
```

# FAIL* – Datenbank einrichten

- Datei anlegen: `~/.my.cnf`
- Folgende Daten eintragen:

```
[client]
user=vezsXX
password=XXXXXXXXXXXXXXX
host=i4jenkins.informatik.uni-erlangen.de
database=vezsXX
```

- Zugangsdaten wurden in der Übung verteilt

**Achtung**

Die Datenbank ist zwischen den Gruppenmitgliedern geteilt!

$\rightsquigarrow$ Keine gleichzeitige Nutzung möglich

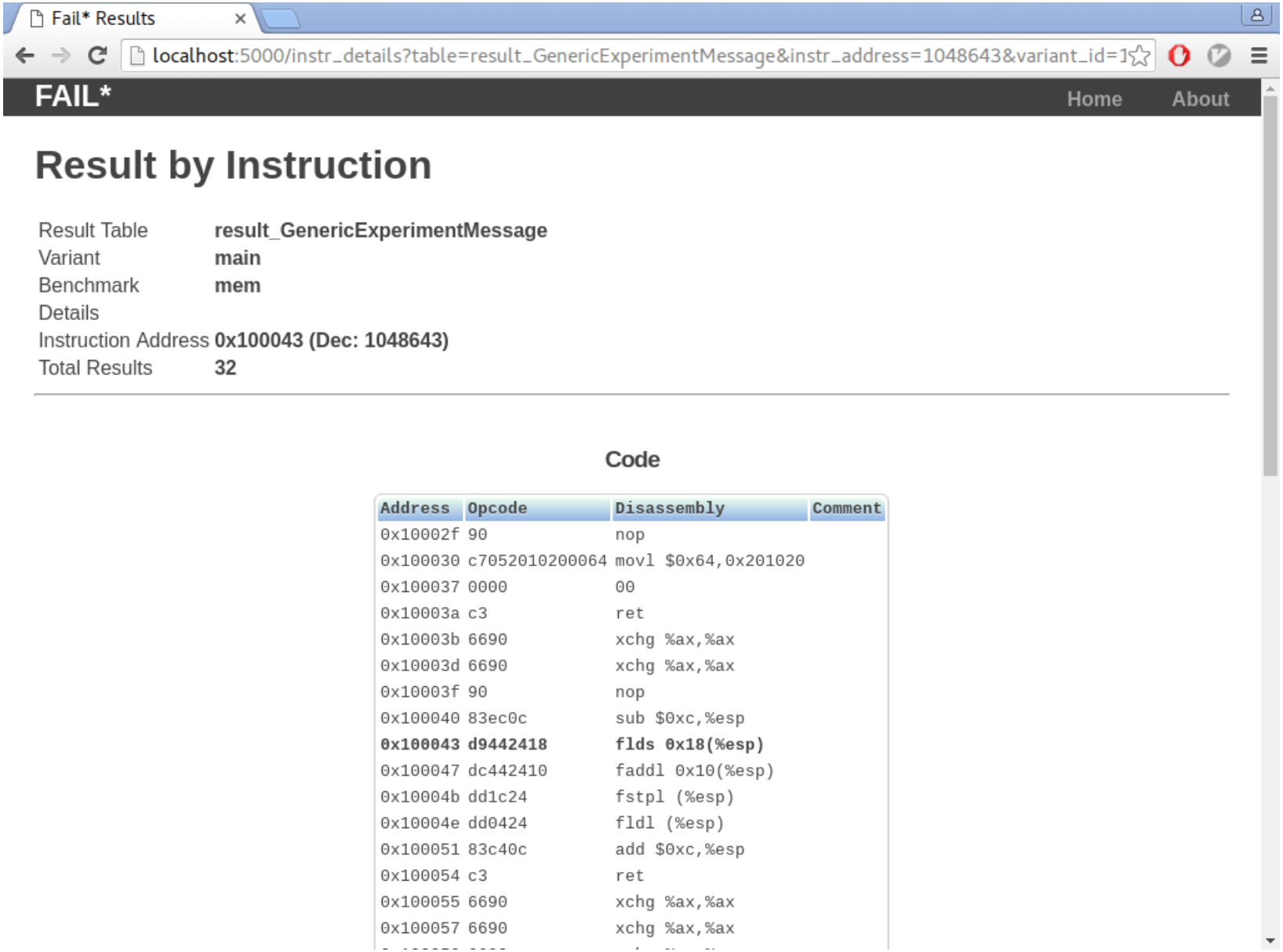
# FAIL* – Workflow

1. `make trace`
   - Erstellung des „Golden Runs”
   - Speicherung der Experimente in Datenbank
   - Faltung des Fehlerraums
2. `make inject`
   - Durchführung Fehlerinjektion
   - Überblick der Ergebnisse in lokal angelegter Datei: `ean_result.csv`

     ☞ Abschätzung des zeitlichen Aufwands
3. `make resultbrowser`
   - Startet lokalen Webserver
   - Ergebnisse per Webbrowser einsehbar

   ⇝ Zugriff über `http://localhost:5000`

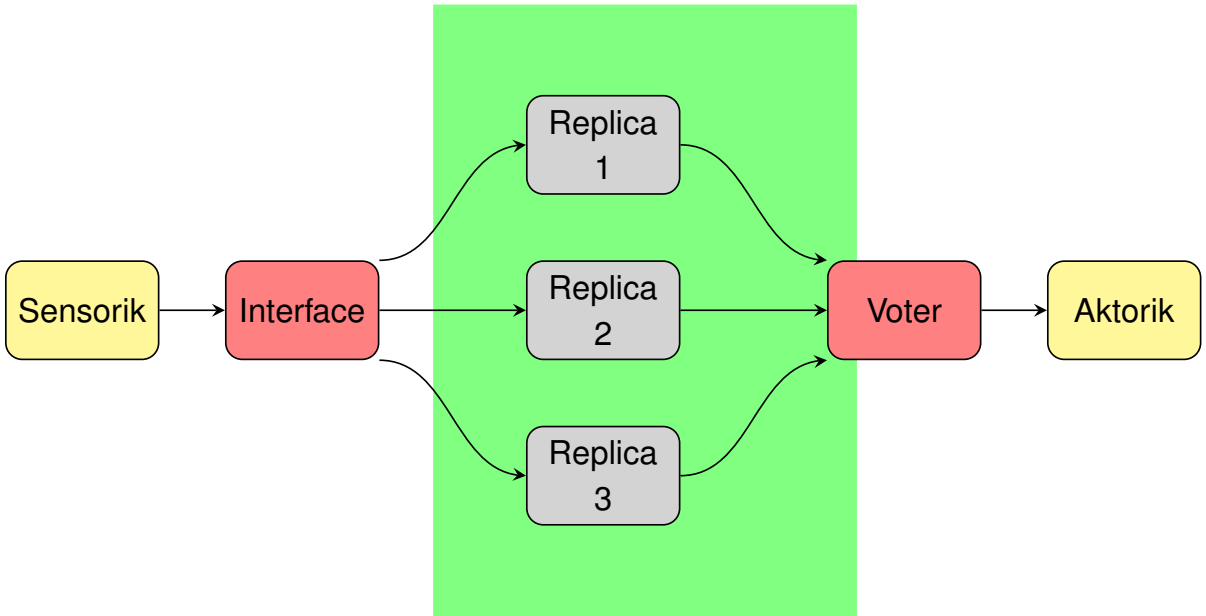
# FAIL* – Resultbrowser




# Überblick

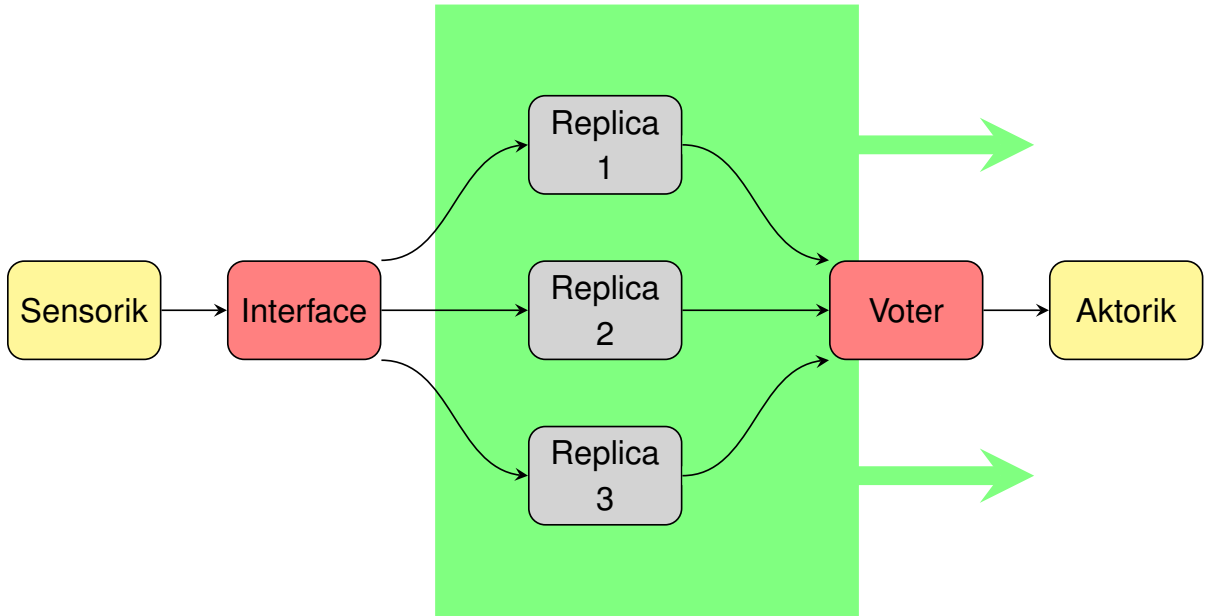


# Klassische “Triple Modular Redundancy” (TMR)




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

# Klassische “Triple Modular Redundancy” (TMR)




## Redundanzbereich

Ausschließlich Replikatausführung

- Erweiterung der Ausgangsseite mit Informationsredundanz
- Mehrheitsentscheid über codierte Prüfsumme

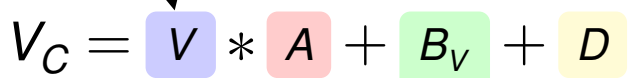
# Überblick



# Erweiterte arithmetische Codierung

nach Forin 1989: “Vital coded microprocessor principles and application for various transit systems” [1]

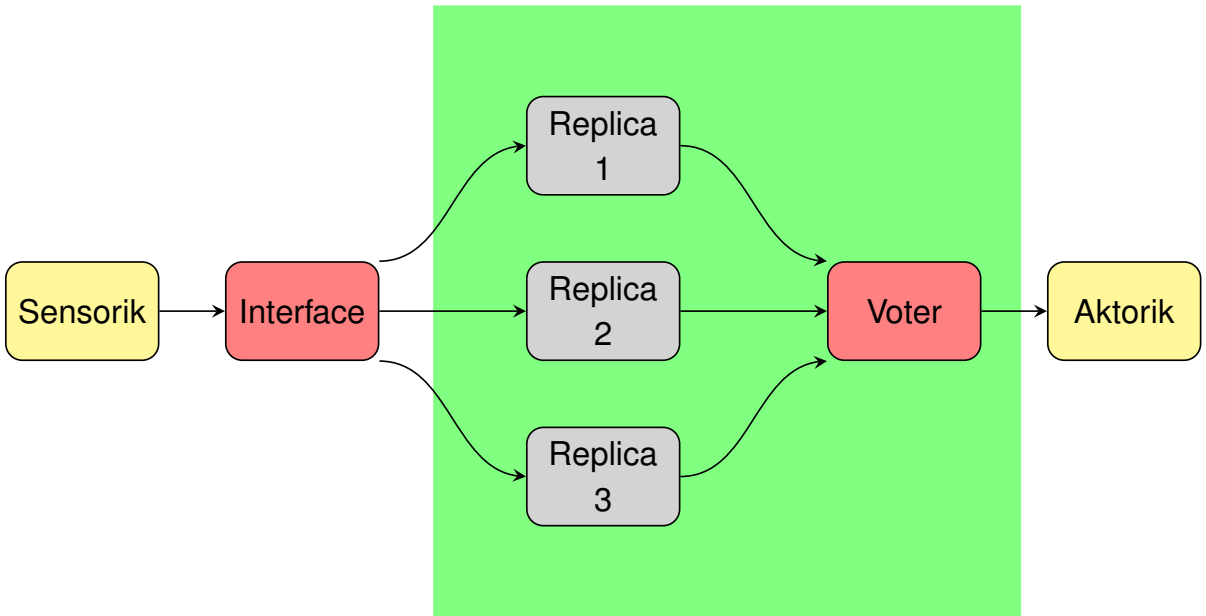
- Arithmetisch codierter Wert $V_C$
- Ausgangswert
- Schlüssel
- Variablenspezifische Signatur
- Zeitstempel




# Wertebereichseinschränkungen

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

# Erweiterung I – codierte Ausgangswerte




- Replikate liefern arithmetisch codierte Ergebnisse
- Mehrheitsentscheid auf codierten Prüfsummen
- Übertragung codierter Ergebnisse

# Vereinfachung für diese Übung

## Für diese Übungsaufgabe:

- Kein Zeitstempel
- Nur Absicherung der Ausgangsseite!

# EAN Vergleichsoperator

- Voting basiert auf codierter Vergleichsoperation:

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

- Im fehlerfreien Fall gilt:

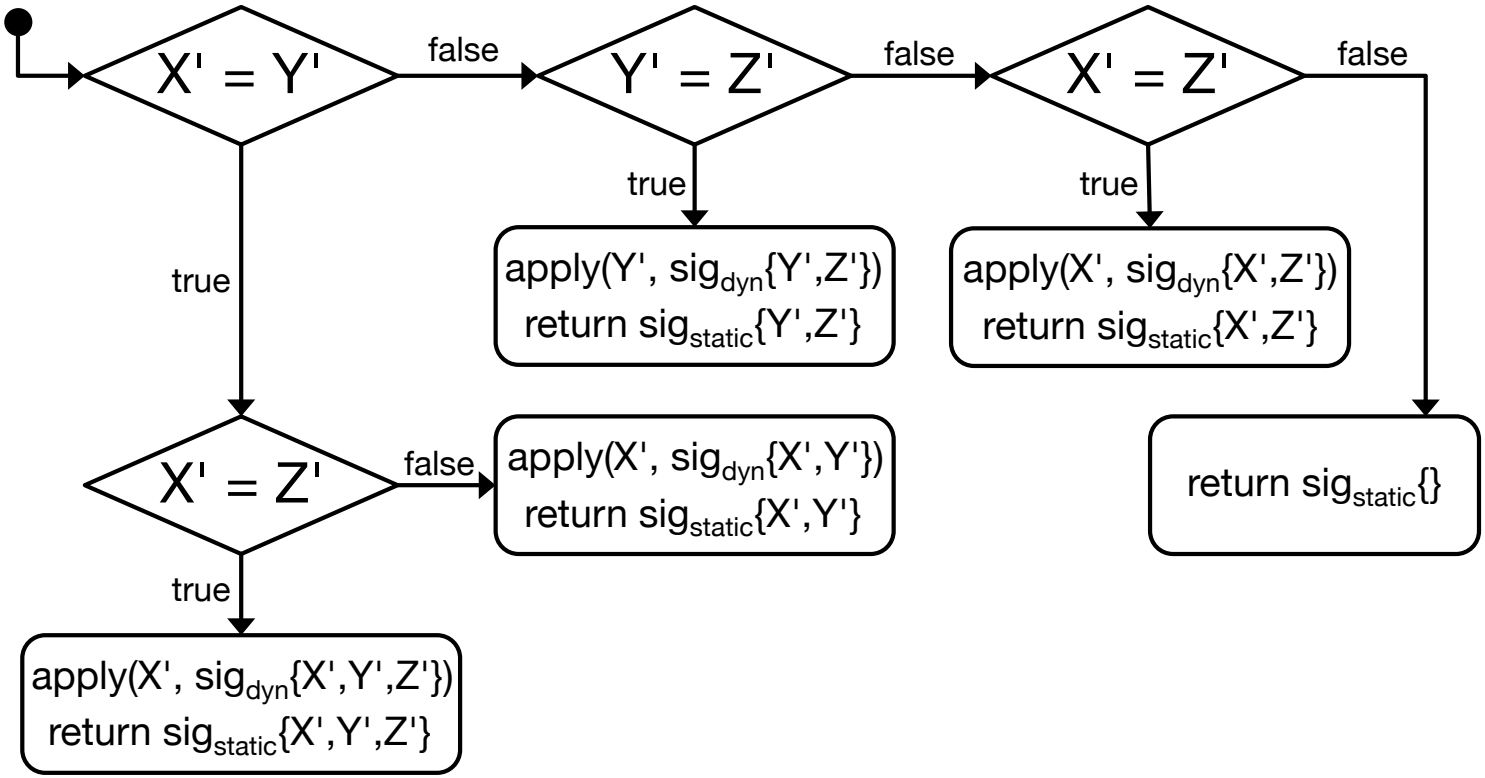
$X = Y$ , $A = A$ aber $B_X \neq B_Y$ !

  - *Rohwerte* sind identisch
  - Schlüssel ist per Definition identisch
  - Signaturen sind unterschiedlich (aber konstant!)

**Bestimmung der Gleichheit durch Differenzbildung:**

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

# Codierter Mehrheitsentscheid



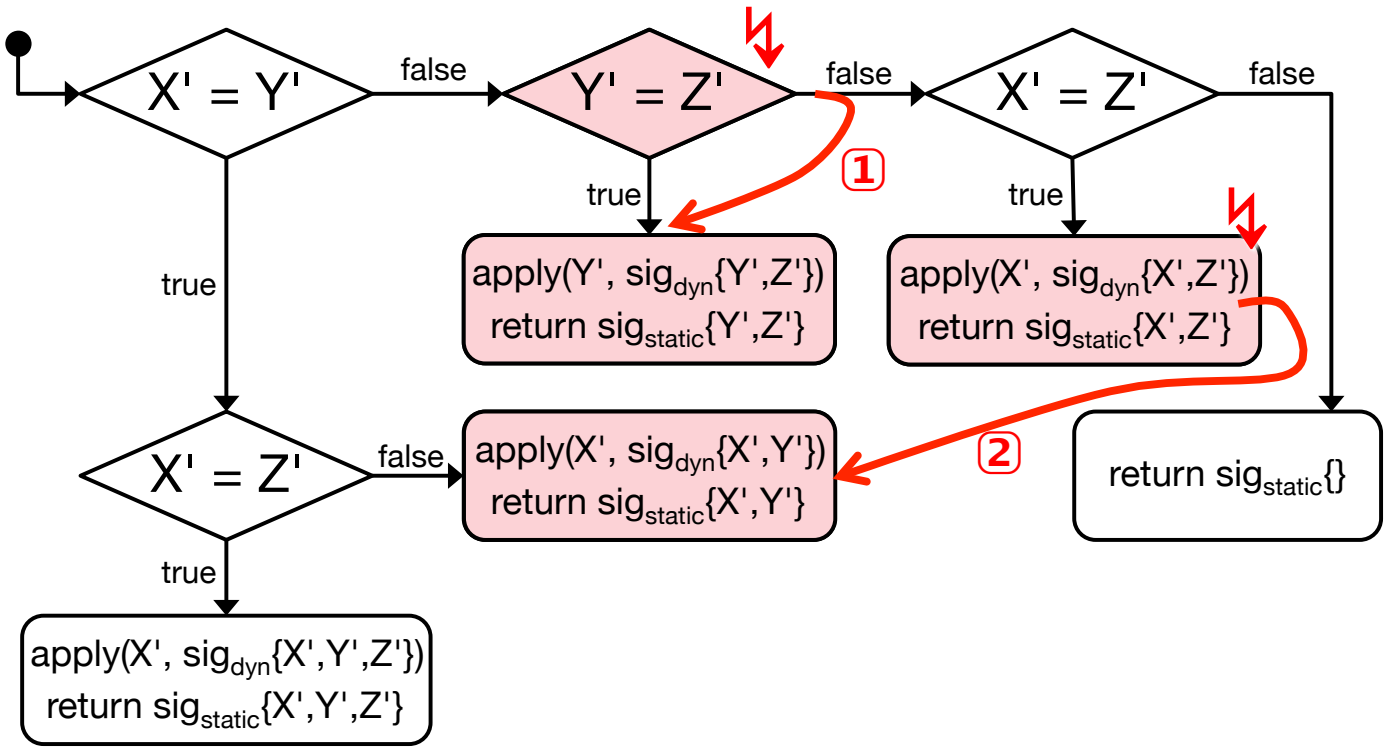

- Bestimmung von dynamischer und statischer Signatur:
  - → $sig_{dyn}(X', Y') : X' = Y' \Rightarrow X' - Y'$
  - → $sig_{static}(X', Y') : X' = Y' \Rightarrow B_X - B_Y$
- Für die Signaturen muss gelten: $B_X > B_Y > B_Z$

# Codierter Mehrheitsentscheid (Forts.)

1. Vergleichsoperation wird durchgeführt (z. B. $X' = Y' \wedge X' = Z'$)
   - Berechnung von $sig_{dyn}$
   - Vergleich mit $sig_{static}$
2. Verzweigungsentscheidung wird nachberechnet:
   - Wiederholte (redundante) Berechnung von $sig_{dyn}$
   - Erneuter Vergleich: $sig_{dyn} = sig_{static}$
   - Addiere $sig_{dyn}$ (*apply*) zum gewählten Ergebnis
3. Konstante Signatur des durchlaufenen Zweiges identifiziert Gewinner (Rückgabewert: $sig_{static}$)
   - Aktor wählt entsprechendes Replikatergebnis
   - Führt inverse Operation zu *apply* durch

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

# Codierter Mehrheitsentscheid - Fehlerfall




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

# Überblick



# Aufgabenstellung

## Aufgabe

- Absichern des Voters per EAN mittels CoRed-Ansatz
- Absichern des Akzeptanzmaskierers am Eingang

- Jedes Replikat hat genau einen Ausgabewert (integer $\mapsto$ enc_t): eine codierte Prüfsumme des Ergebnisses

  $\rightsquigarrow$ Festlegung für jeden der drei Ausgabewerte $(X', Y', Z')$ jeweils *unterschiedliche* aber *konstante* Signatur $(SIG_X, SIG_Y, SIG_Z)$

- Nutzung des nächstgrößeren Datentyps $X$ für den ursprünglichen Wert $X'$

  $\rightsquigarrow$ Wahl einer Zahl $A$ mit möglichst großem Hamming-Abstand unter *Vermeidung* möglicher *Überläufe bei der Codierung*

# Hinweise

## Berechnungen mit codierten Werten

- Berücksichtigung der (statischen) Signaturen

⇝ Eigene Operationen mit konstanten Signaturwerten notwendig

⇝ bspw. eigenes `equals` anstatt ==

- Bestenliste wird automatisch auf Webseite veröffentlicht
- VEZS-Shootout Wettbewerb
  - Wer hat den robustesten Code?
  - Vergleich der Repositories
- Wie werden Resultate generiert?

# Von Zuhause entwickeln/injizieren I

- Auf leistungsfähigen Rechnern arbeiten, bspw. `faui00a-y` / `faui02a-y`

  ⇝ Rechnerliste: `wwwcip.cs.fau.de/cipPools/roomIndex.en.html`
- Entfernt arbeiten per ssh:
  - Per ssh anmelden: `ssh nutzer@fauiXXx.cs.fau.de`
  - Auf andere Nutzer prüfen: `who`
  - Im Hintergrund arbeiten lassen: TMUX
    1. Starten: `tmux`

    ⇝ Injektion mit `nice` starten (verringert Prozesspriorität):
    `nice make trace && nice make inject`
    2. Abkoppeln: `Strg + b` dann `d`
    3. Wieder verbinden: tmux attach
    4. Mehr Informationen:

    + `man tmux`
    + `https://www.hamvocke.com/blog/a-quick-and-easy-guide-to-tmux/`
    + `https://robots.thoughtbot.com/a-tmux-crash-course`
  - `tmux` ab und an schließen: Prozesschecker im CIP beendet langlaufende Prozesse automatisch

# Von Zuhause entwickeln/injizieren II

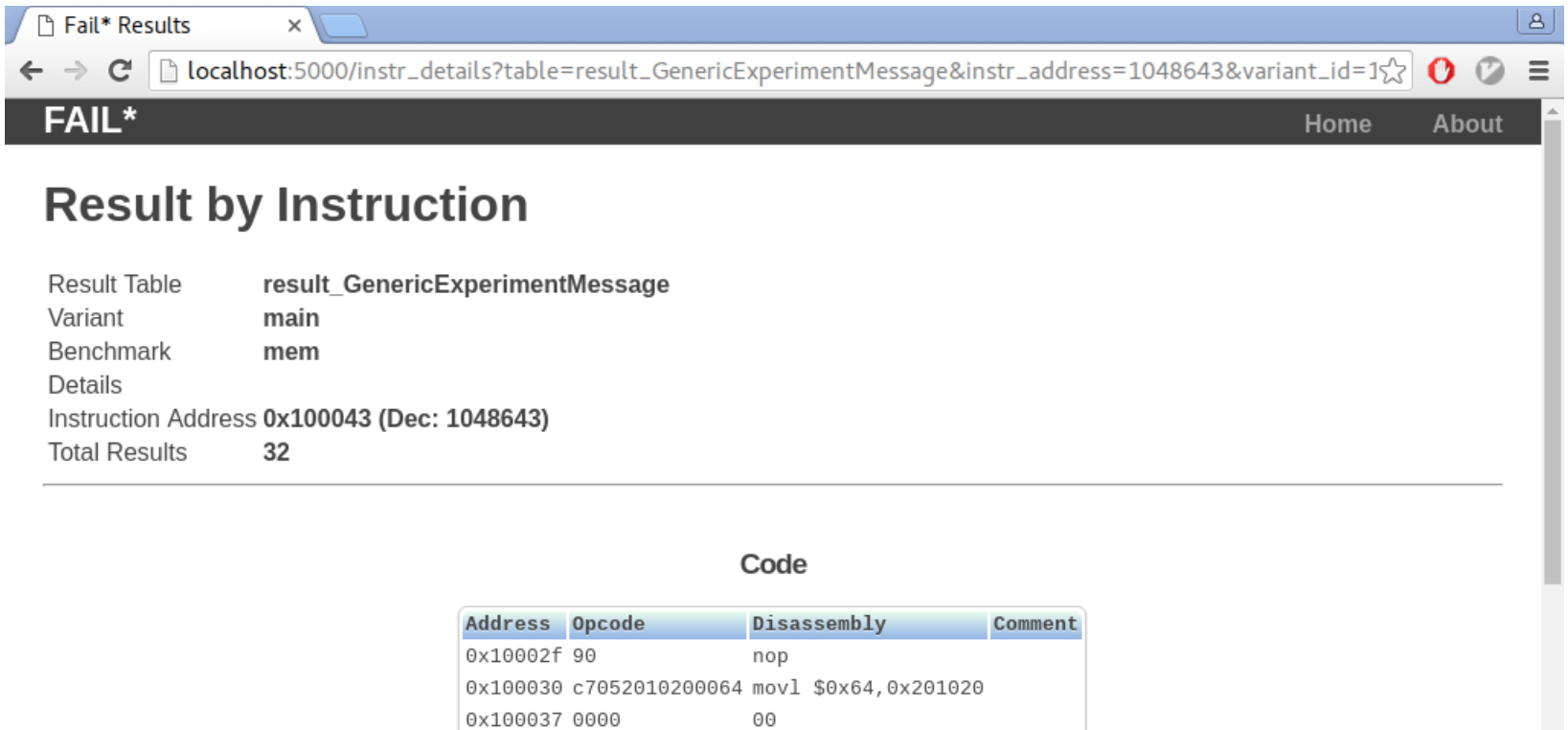


Um auf die Resultbrowseroberfläche zuzugreifen:

```
% ssh -N -L 127.0.0.1:5000:localhost:5000 user@fauiXXx...
```

↝ Leitet Port 5000 des Rechners `fauiXXx` auf den lokalen Rechner um

↝ Zugriff über `http://localhost:5000`




# Literatur I

Forin.
Vital coded microprocessor principles and application for various transit systems.
*IFA-GCCT*, pages 79–84, 1989.