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

Organisatorisches

FAIL* Shootout – EAN Aufgabe

`https://www4.cs.fau.de/Lehre/SS19/V_VEZS/Uebung/dashboard/fail.html`

☞ **Deadline Aufgabe: 18.06.2019**

☞ **Deadline Shootout: 18.07.2019**

- ☞ **Dienstag 11.06.2019 "Bergfrei"**
- ☞ Ersatztermin: **Donnerstag 13.06.2019 12:15 Rechnerübung** (freiwillig)
  - ☞ nächster Übungstermin: 27.06.
- ☞ Vorlesung am 17.06. entfällt
  - ☞ nächster Vorlesungstermin: 24.06.

Überblick

1. Abfangen von Integer-Fehlern

2. Testen
   - Vollständig Testen
   - Testfallintegration mit CMake
   - Peer-Review
   - Codeüberdeckung
   - Instrumentierung – Sanitizer
   - Statische Programmanalyse
   - Testfallgenerierung – Fuzzing

3. Übungsaufgabe

Überblick

Umgang mit Ganzzahlfehlern

- C bietet viele subtile Fehlermöglichkeiten
- Im C-Quiz haben wir einige kennengelernt
- Was uns noch fehlt:
  - *Wie verhält man sich als Programmierer richtig?*

  ⇝ Heute ein paar Beispiele

Addition

Was soll da schon schiefgehen...

```
1 unsigned int func(unsigned int a, unsigned int b) {
2   return a + b;
3 }
```

Addition

Was soll da schon schiefgehen...

```
1 unsigned int func(unsigned int a, unsigned int b) {
2   return a + b;
3 }
```

Vorbedingungstest

```
1 #include <limits.h>
2 unsigned int func(unsigned int a, unsigned int b) {
3   if (UINT_MAX - a < b) { raise("wraparound"); }
4   return a + b;
5 }
```

Nachbedingungstest

```
1 unsigned int func(unsigned int a, unsigned int b) {
2   unsigned int ret = a + b;
3   if (ret < a) { raise("wraparound"); }
4   return ret;
5 }
```

Multiplikation

Was soll da schon schiefgehen...

```
1 unsigned int func(unsigned int a, unsigned int b) {
2   return a * b;
3 }
```

Multiplikation

Was soll da schon schiefgehen...

```
1 unsigned int func(unsigned int a, unsigned int b) {
2   return a * b;
3 }
```

Vorbedingungstest

```
1 #include <limits.h>
2 unsigned int func(unsigned int a, unsigned int b) {
3   if (a == 0 or b == 0) { return 0; }
4   if (UINT_MAX / a < b) { raise("wraparound"); }
5   return a * b;
6 }
```

Addition

Was soll da schon schiefgehen...

```
1 signed int func(signed int a, signed int b) {
2   return a + b;
3 }
```

Addition

Was soll da schon schiefgehen...

```
1 signed int func(signed int a, signed int b) {
2   return a + b;
3 }
```

Vorbedingungstest

```
1 #include <iso646.h>
2 #include <limits.h>
3 signed int func(signed int a, signed int b) {
4   if ((b > 0 and a > INT_MAX - b)
5    or (b < 0 and a < (INT_MIN - b))) { raise("overflow"); }
6   return a + b;
7 }
```

Division

Was soll da schon schiefgehen...

```
1 signed long func(signed long a, signed long b) {
2   return a / b;
3 }
```

Division

Was soll da schon schiefgehen...

```
1 signed long func(signed long a, signed long b) {
2   return a / b;
3 }
```

Vorbedingungstest

```
1 #include <iso646.h>
2 #include <limits.h>
3 signed long func(signed long a, signed long b) {
4   if (b == 0) { raise("division by 0"); }
5   return a / b;
6 }
```

Division

- Reicht das schon?

Was soll da schon schiefgehen...

```
1 signed long func(signed long a, signed long b) {
2   if (b == 0) { raise("division by 0"); }
3   return a / b;
4 }
```

Division

- Reicht das schon?

Was soll da schon schiefgehen...

```
1 signed long func(signed long a, signed long b) {
2   if (b == 0) { raise("division by 0"); }
3   return a / b;
4 }
```

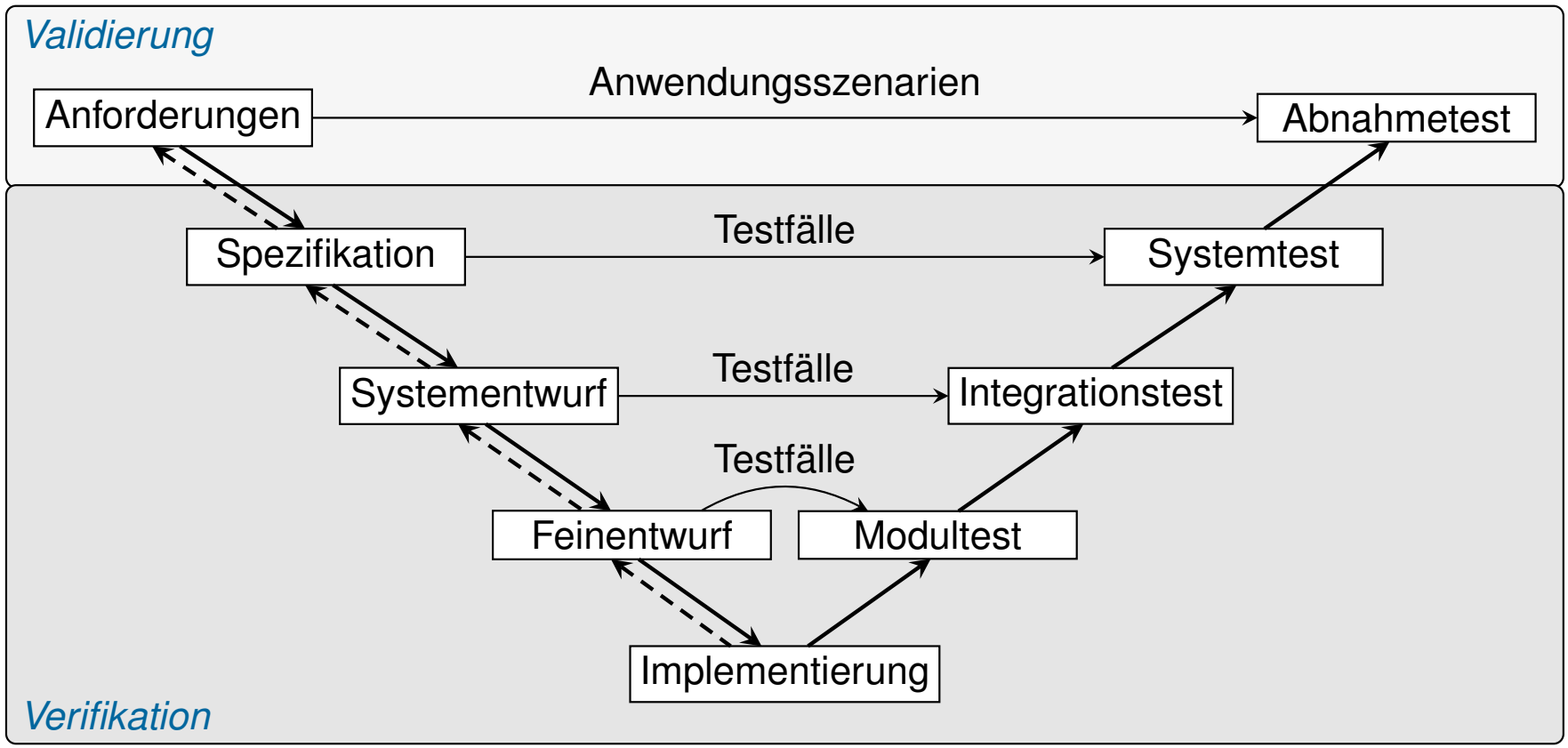
Vorbedingungstest

```
1 #include <iso646.h>
2 #include <limits.h>
3 signed long func(signed long a, signed long b) {
4   if (b == 0) { raise("division by zero"); }
5   if (a == LONG_MIN and b == -1) { raise("overflow"); }
6   return a / b;
7 }
```

Überblick

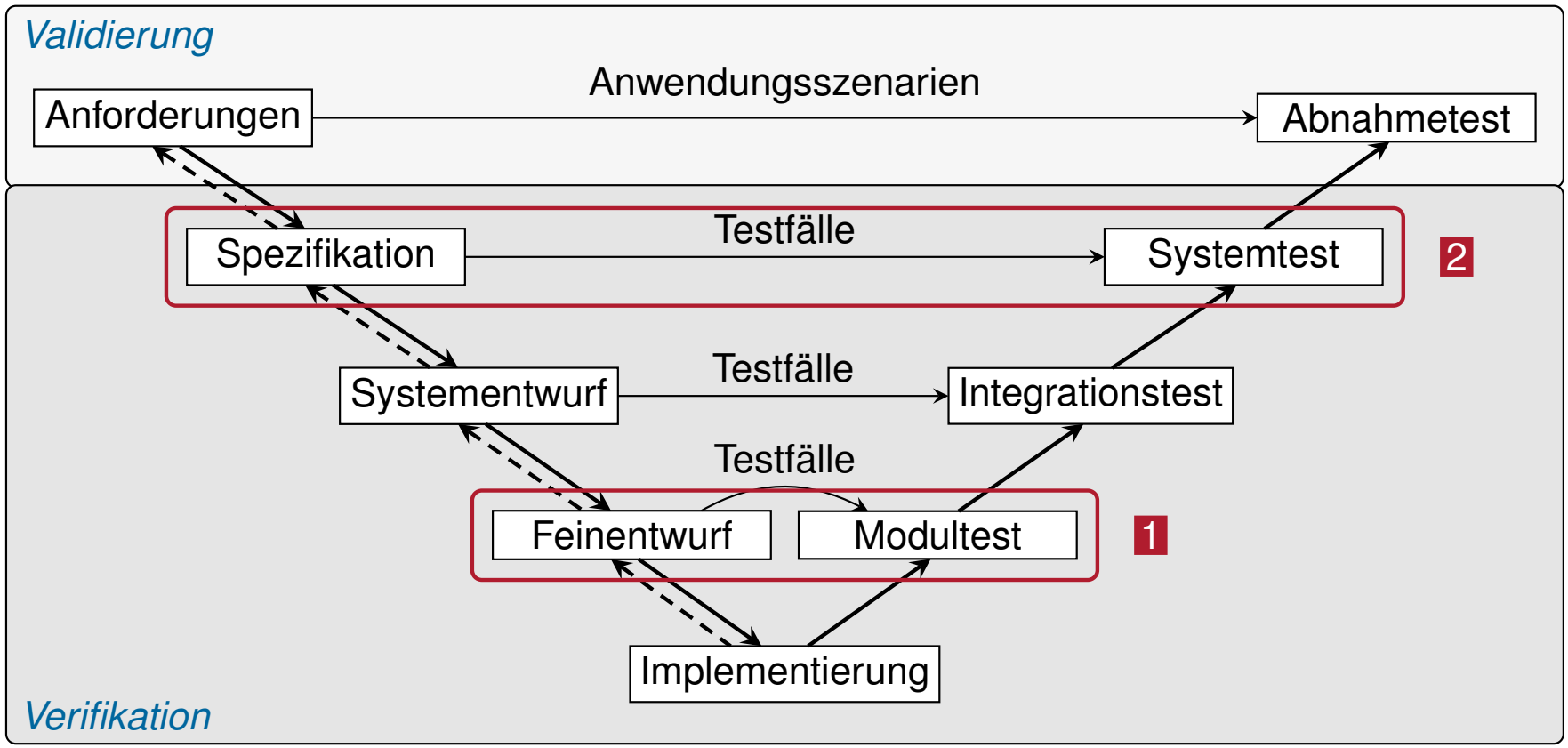
Einordnung in den Entwicklungsprozess

Softwareentwicklung nach dem V-Modell wird zugrunde gelegt



Einordnung in den Entwicklungsprozess

Softwareentwicklung nach dem V-Modell wird zugrunde gelegt



Testen

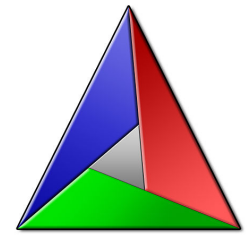
- Erste Grundregeln:
  - Testbarkeit von vornherein einplanen
    - ⇝ Feingranulare Testfälle
    - ⇝ *Ein Testfall für jede einzelne Funktion!*
  - Teste Datentypen an ihren Wertebereichsgrenzen
    - → `INT16_MAX, INT16_MIN, ...`
  - *Minimale Testabdeckung*: erreichbarer Code/Zeilenüberdeckung
- Hilfsmittel:
  - Automatisierte Testinfrastruktur
  - Code-Coverage-Analysewerkzeug

Vorsicht!

- Testfälle können nur die Anwesenheit von Fehlern zeigen
- Nicht deren Abwesenheit! (→ vgl. formale Verifikation)

⇝ Alle *Randfälle* erkennen und abdecken

Testfallintegration mit CMake



- Integration von Tests im Softwareprojekt
- Automatisierte Ausführung und Auswertung von Testläufen
- Konfigurationsdatei: `tests/CMakeLists.txt`
  - Ausführbares Target:
    `add_executable(plus_test plus_test.c)`
  - Hinzubinden der zu testenden Bibliothek:
    `target_link_libraries(plus_test mathe)`
  - Bekanntmachen als Testfall:
    `add_test(MatheTest_PLUS plus_test)`
- Ausführung der Tests: `make && make test`
- Automatische Testauswertung:
  - Anhand Rückgabewert (0 → OK, -1 → Fehler)
  - Notfalls auch Parsen von Ausgaben
- Ausgaben der Tests (`(f)printf`) protokolliert in Datei `Testing/Temporary/LastTest.log`

Testfallintegration mit CMake

- Tests sind Programme im Unterverzeichnis `tests`

```
tests
|-- CMakeLists.txt
|-- priority_queue_test1.c
|-- priority_queue_test1.c
`-- priority_queue_test_malloc.c
```

- Die Datei `tests/CMakeLists.txt` definiert drei Gruppen von Testfällen:

```
############ CONFIGURATION SECTION, add your testcases below
# Generelle Testfälle, sowohl für die eigene wie auch die Fremde
    Implementierung
set(EZS_PQ_GENERAL_TESTS priority_queue_test1
                         priority_queue_test2)

# Mit dem Address-sanetizer inkompatible Tests
set(EZS_PQ_MALLOC_TESTS priority_queue_test_malloc)

# Testfälle ausschließlich für die eigene Implementirung
set(EZS_PQ_OWN_ONLY_TESTS "")
```

- Aktivieren eigener Tests: Eintrag in die entsprechende Liste

Peer-Review und Quality-Assurance

1. ✉ Betreff: “Euer QA-Team wartet”

From: i4ezsmux+projectX-qa@i4.cs.fau.de
To: i4ezsmux+projectX-dev@i4.cs.fau.de
Subject: Euer QA-Team wartet

Bitte sendet eure erste Implementierung bis zum Termin an obige Adresse

Peer-Review und Quality-Assurance

1. ✉ Betreff: “Euer QA-Team wartet”
2. `make doxy`: Dokumentation lesen und implementieren
3. `make pack-review`

Peer-Review und Quality-Assurance

1. ✉ Betreff: “Euer QA-Team wartet”
2. `make doxy`: Dokumentation lesen und implementieren
3. `make pack-review`
4. ✉ `./build/review.tar.xz`: “Unsere Lösung” (Fester Termin!)

From: i4ezsmux+projectX-dev@i4.cs.fau.de
To: i4ezsmux+projectX-qa@i4.cs.fau.de
Subject: Initiale Lösung

Hallo liebes QA-Team,

anbei unsere libpriority_queue_alien.a aus make pack-review.
Wir freuen uns schon auf eure Anmerkungen
Beste Grüße, **ANONYM**

Anlage: review.tar.xz

Peer-Review und Quality-Assurance

1. ✉ Betreff: “Euer QA-Team wartet”
2. `make doxy`: Dokumentation lesen und implementieren
3. `make pack-review`
4. ✉ `./build/review.tar.xz`: “Unsere Lösung” (Fester Termin!)
5. Entpacken nach `./review/libpriority_queue_alien.a`
6. `cmake -DBUILD_ALIEN_TEST=ON -DBUILD_WITH_COVERAGE=OFF`
7. `make && make tests`, Ergebnis und Ausgabe prüfen

Peer-Review und Quality-Assurance

1. ✉ Betreff: “Euer QA-Team wartet”
2. `make doxy`: Dokumentation lesen und implementieren
3. `make pack-review`
4. ✉ `./build/review.tar.xz`: “Unsere Lösung” (Fester Termin!)
5. Entpacken nach `./review/libpriority_queue_alien.a`

To: i4ezsmux+projectX-dev@i4.cs.fau.de
Subject: Mögliche Fehler

Hallo liebes Dev-Team,

bei uns schlagen die folgenden Tests mit euerer Implementierung fehl:
- unordered_insert:
  Der Test fügt in inverser Reihenfolge ein und prüft den folgenden Satz der Spezifikation “...”
  Ausgabe: ...
- ...

Sagt Bescheid, wenn ihr mehr Infos benötigt.

Peer-Review und Quality-Assurance

1. ✉ Betreff: “Euer QA-Team wartet”
2. `make doxy`: Dokumentation lesen und implementieren
3. `make pack-review`
4. ✉ `./build/review.tar.xz`: “Unsere Lösung” (Fester Termin!)
5. Entpacken nach `./review/libpriority_queue_alien.a`
6. `cmake -DBUILD_ALIEN_TEST=ON -DBUILD_WITH_COVERAGE=OFF`
7. `make && make tests`, Ergebnis und Ausgabe prüfen
8. ✉ Test-Report: "Fehler in libpriority_queue"
9. Fehler beheben & Neu Packen
10. ✉ `./build/review.tar.xz`: “Update/Fixes”
11. `...` (Wiederholen solange nötig)

Peer-Review und Quality-Assurance

1. ✉ Betreff: “Euer QA-Team wartet”
2. `make doxy`: Dokumentation lesen und implementieren
3. `make pack-review`
4. ✉ `./build/review.tar.xz`: “Unsere Lösung” (Fester Termin!)
5. Entpacken nach `./review/libpriority_queue_alien.a`
6. `cmake -DBUILD_ALIEN_TEST=ON -DBUILD_WITH_COVERAGE=OFF`
7. `make && make tests`, Ergebnis und Ausgabe prüfen
8. ✉ Test-Report: "Fehler in libpriority_queue"
9. Fehler beheben & Neu Packen
10. ✉ `./build/review.tar.xz`: “Update/Fixes”
11. `...` (Wiederholen solange nötig)

∞ ✉ ✓ “All clear”

Codeüberdeckung: gcov/lcov

LCOV - code coverage report

				Hit	Total	Coverage
Current view:	top level		**Lines:**	49	114	43.0 %
Test:	coverage.lcov		**Functions:**	6	14	42.9 %
Date:	2018-05-26 00:38:08		**Branches:**	19	72	26.4 %
Legend:	Rating:	low: < 75 % / medium: >= 75 % / high: >= 90 %				

Directory	Line Coverage		Functions		Branches	
src	43.0 %	49 / 114	42.9 %	6 / 14	26.4 %	19 / 72

Generated by: LCOV version 1.13-14-ga5dd952

- Werkzeug aus der gcc-Toolchain
- Instrumentierung des Binärcodes $\rightsquigarrow$ *Laufzeitkosten*
- Protokollieren der Programmausführung
  - Wie oft wird jede Codezeile ausgeführt?
  - Welche Zeilen werden überhaupt ausgeführt?
  - Welche Verzweigungen wurden genommen?
- HTML Ausgabe: lcov

$\rightarrow$ Tests solange erweitern, bis *vollständige Verzweigungsüberdeckung* erreicht!

Aufdecken von Laufzeitfehlern – AddressSanitizer

- *„Im besten Fall kracht es bei Speicherzugriffsfehlern!"*
- In Übungen: Verwendung von Clang AddressSanitizer [1][1]
- Checks zur Laufzeit
  - falsche Verwendung von Zeigern
  - nicht-definierte Integer-Operationen
  - Lesen uninitialisierten Speichers
  - Integer-Überlauf
  - ...

Entdeckt Fehler . . .

. . . nur, wenn die verwendeten Testfälle diese auslösen.

☞ **zur Laufzeit**

- Laufzeitkosten: $\approx 2\,\mathrm{x}$

[1] `http://clang.llvm.org/docs/AddressSanitizer.html`

Clang AddressSanitizer – Verwendung

```
1  // program.cpp
2  int main(int argc, char **argv) {
3    int *array = new int[100];
4    delete[] array;
5    return array[argc];  // BOOM
6  }
```

```
$ clang++ -O1 -g -fsanitize=address program.cpp
```

```
$ ./a.out
```

```
ERROR: AddressSanitizer: heap-use-after-free on address 0x602e0001fc64 at pc ...
```

- Wird von cmake-Skripten automatisch verwendet, wenn
  - Debugging aktiviert ist
  - und `clang` als Compiler verwendet wird
  - siehe `cmake/sanitizer.cmake`
- Aufruf von cmake

↝ `CC=clang CXX=clang++ cmake -DCMAKE_BUILD_TYPE=Debug ..`

Statische Programmanalyse – Clang Static Analyzer

- Analyse des Quellcodes (C, C++, Objective-C)
- Keine Ausführung des Codes auf Hardware $\leadsto$ „statische Analyse”
- Eingabewerte als *symbolisch* angenommen
  $\rightarrow$ *symbolische Ausführung/Erreichbarkeitsanalyse*
- Verfügbare Checks[2]
  - Wertebereichsanalysen: Division mit Null
  - Verwendung uninitialisierter Variablen
  - ...
- Analyse ist *nicht fehlerfrei* (engl. sound)
  - Nicht möglich alle Fehler zu finden (engl. false negatives)
- Analyse ist *nicht präzise* (engl. precise)
  - Falsche positive Befunde sind möglich (engl. false positives)

[2] `http://clang-analyzer.llvm.org/available_checks.html`

Clang Static Analyzer – Verwendung

```
1  void test() {
2    int i, a[10];
       1  'i' declared without an initial value →
3    int x = a[i]; // warn: array subscript is undefined
       2  ← Array subscript is undefined
4  }
```

- Einzelne Datei überprüfen: `scan-build clang -c program.c`
- Übung: Aufruf von `scan-build` mit `cmake` als Argument

  ⇝ `CC=clang CXX=clang++ scan-build cmake ..`

  ⇝ `scan-build make`
- Fehler/Warnungen gefunden ⟶ Ausgabe von HTML Dateien
- Aufruf von `scan-view` wie in Ausgabe beschrieben

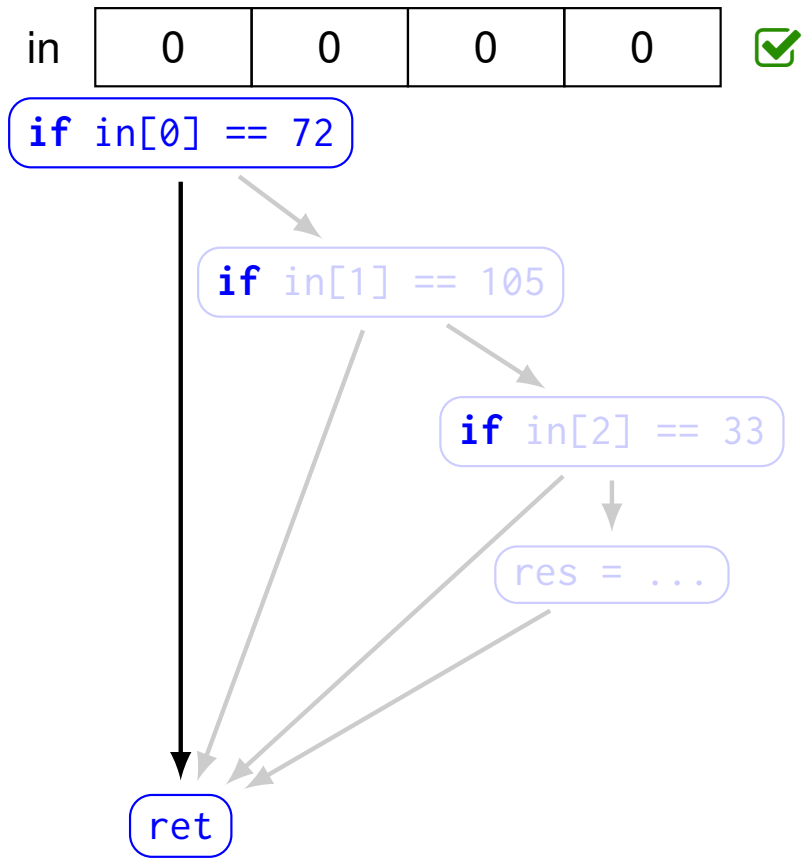
Überdeckungsgesteuertes Wuscheln (engl. *Fuzzing*)

- Raten möglicher Eingaben
- Strukturiertes Vorgehen:
  möglichst alle Blöcke gleichmäßig abdecken

  ⇝ Aus Modifikation von Eingaben die zum Vorgänger führten

  ⇝ *Mutation*: Zufällige Veränderung
  Crossover: Kombination existierender Eingaben
- Beispiel:

```
if (size < 4) return 0;
if(in[0] == 72)
  if(in[1] == 105)
    if(in[2] == 33)
      res = 42 / in[3];
return res;
```



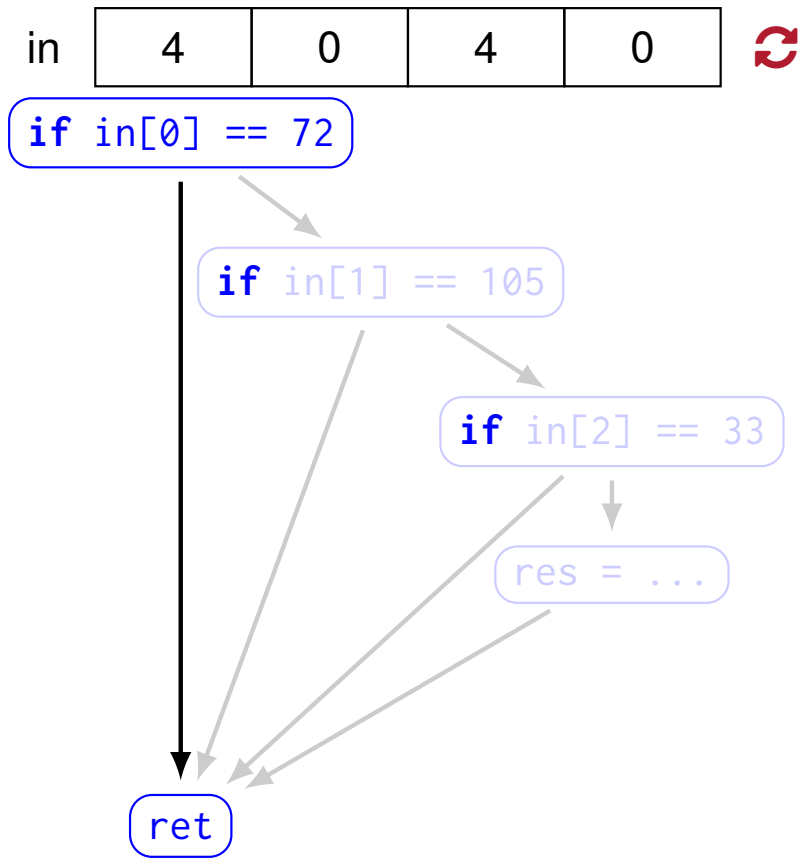
Überdeckungsgesteuertes Wuscheln (engl. *Fuzzing*)

- Raten möglicher Eingaben
- Strukturiertes Vorgehen:
  möglichst alle Blöcke gleichmäßig abdecken

  ⇝ Aus Modifikation von Eingaben die zum Vorgänger führten

  ⇝ *Mutation*: Zufällige Veränderung
  Crossover: Kombination existierender Eingaben
- Beispiel:

```
if (size < 4) return 0;
if(in[0] == 72)
  if(in[1] == 105)
    if(in[2] == 33)
      res = 42 / in[3];
return res;
```



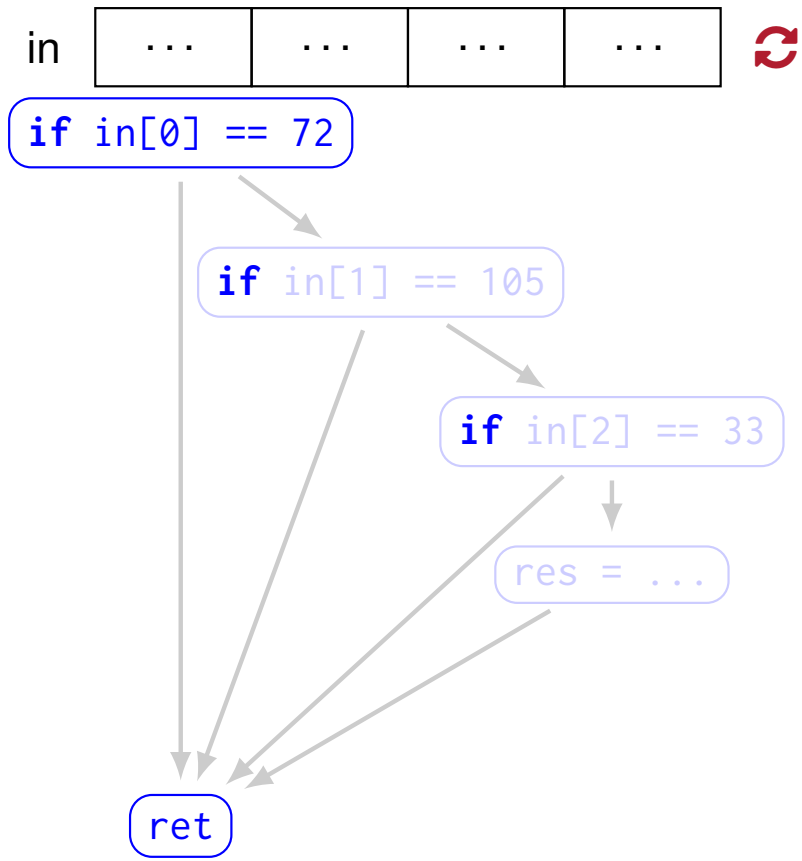
Überdeckungsgesteuertes Wuscheln (engl. *Fuzzing*)

- Raten möglicher Eingaben
- Strukturiertes Vorgehen:
  möglichst alle Blöcke gleichmäßig abdecken

  ⇝ Aus Modifikation von Eingaben die zum Vorgänger führten

  ⇝ *Mutation*: Zufällige Veränderung
  Crossover: Kombination existierender Eingaben
- Beispiel:

```
if (size < 4) return 0;
if(in[0] == 72)
  if(in[1] == 105)
    if(in[2] == 33)
      res = 42 / in[3];
return res;
```



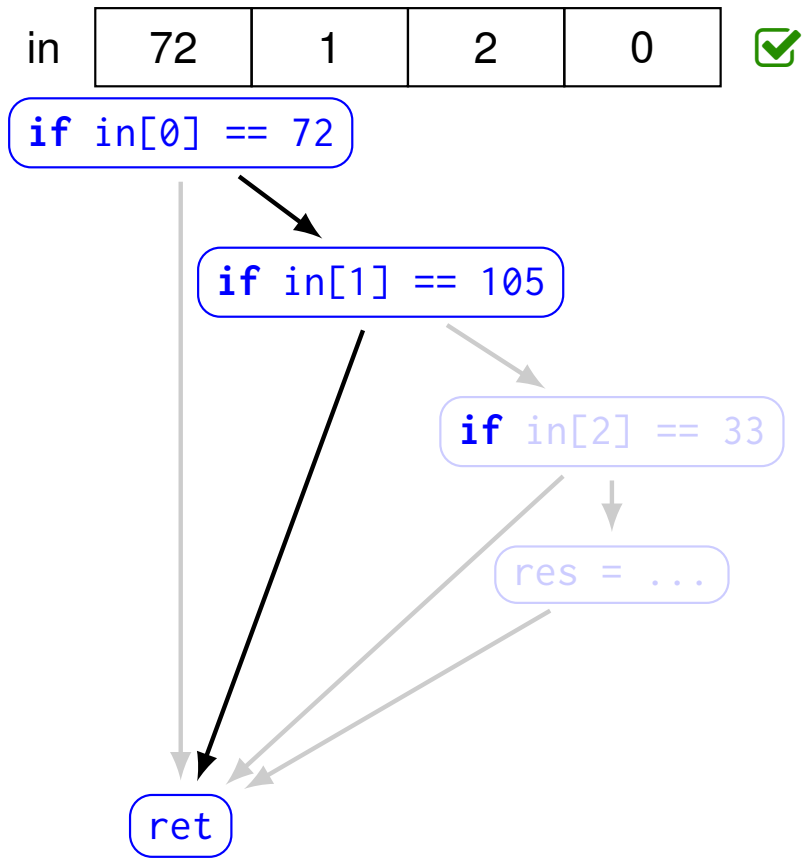
Überdeckungsgesteuertes Wuscheln (engl. *Fuzzing*)

- Raten möglicher Eingaben
- Strukturiertes Vorgehen:
  möglichst alle Blöcke gleichmäßig abdecken

  ⇝ Aus Modifikation von Eingaben die zum Vorgänger führten

  ⇝ *Mutation*: Zufällige Veränderung
  Crossover: Kombination existierender Eingaben
- Beispiel:

```
if (size < 4) return 0;
if(in[0] == 72)
  if(in[1] == 105)
    if(in[2] == 33)
      res = 42 / in[3];
return res;
```



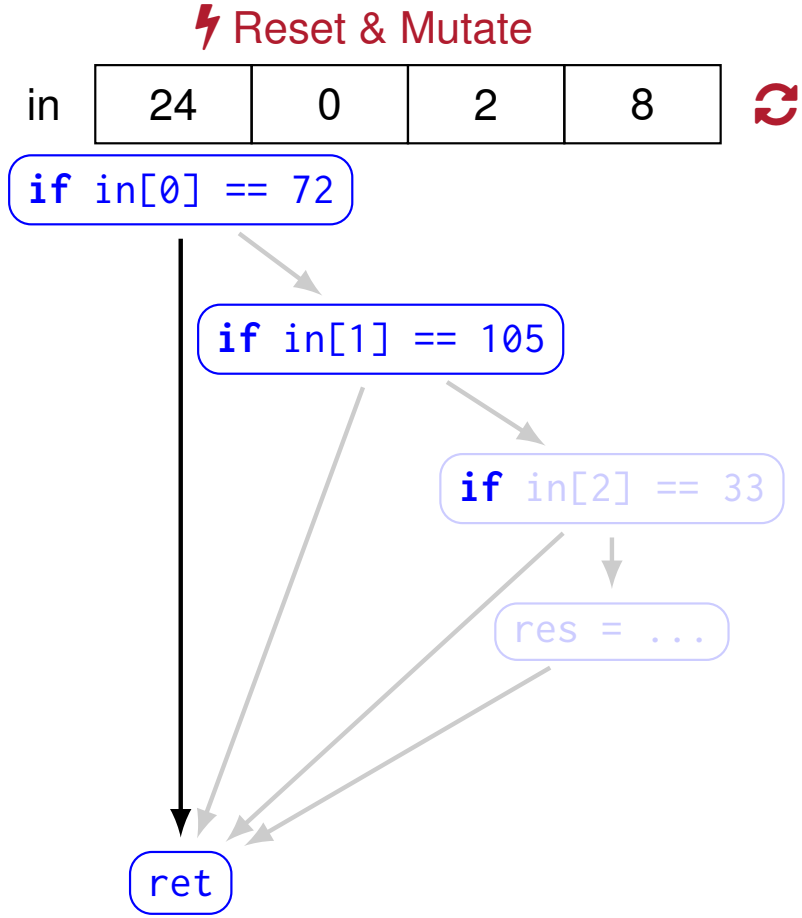
Überdeckungsgesteuertes Wuscheln (engl. *Fuzzing*)

- Raten möglicher Eingaben
- Strukturiertes Vorgehen: möglichst alle Blöcke gleichmäßig abdecken

  ⇝ Aus Modifikation von Eingaben die zum Vorgänger führten

  ⇝ *Mutation*: Zufällige Veränderung
  Crossover: Kombination existierender Eingaben
- Beispiel:

```
if (size < 4) return 0;
if(in[0] == 72)
  if(in[1] == 105)
    if(in[2] == 33)
      res = 42 / in[3];
return res;
```



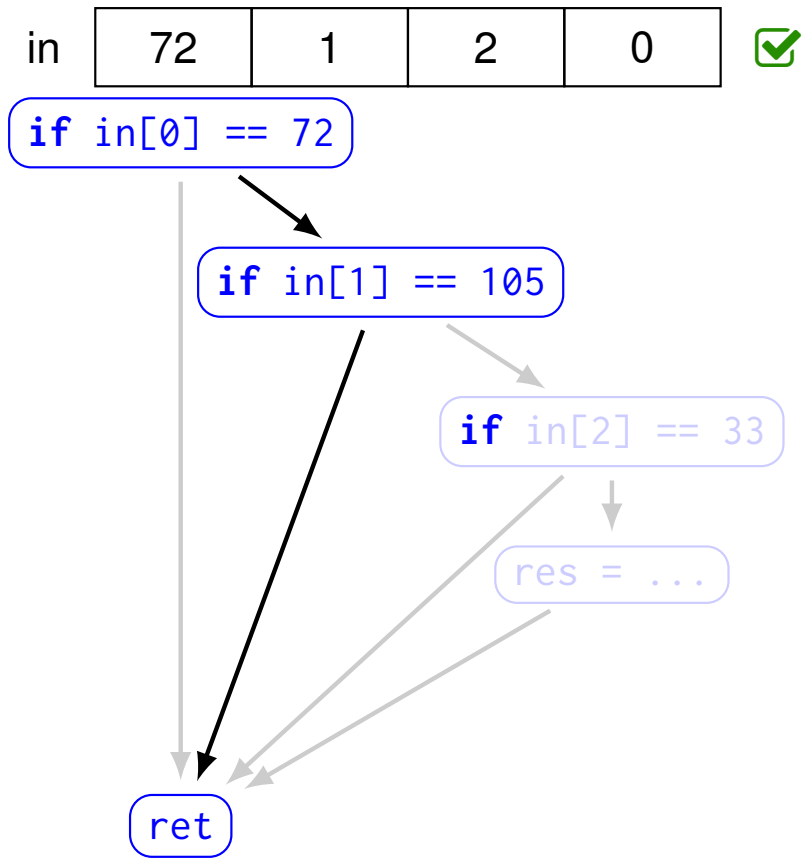
Überdeckungsgesteuertes Wuscheln (engl. *Fuzzing*)

- Raten möglicher Eingaben
- Strukturiertes Vorgehen:
  möglichst alle Blöcke gleichmäßig abdecken

  ⇝ Aus Modifikation von Eingaben die zum Vorgänger führten

  ⇝ *Mutation*: Zufällige Veränderung
  Crossover: Kombination existierender Eingaben
- Beispiel:

```
if (size < 4) return 0;
if(in[0] == 72)
  if(in[1] == 105)
    if(in[2] == 33)
      res = 42 / in[3];
return res;
```



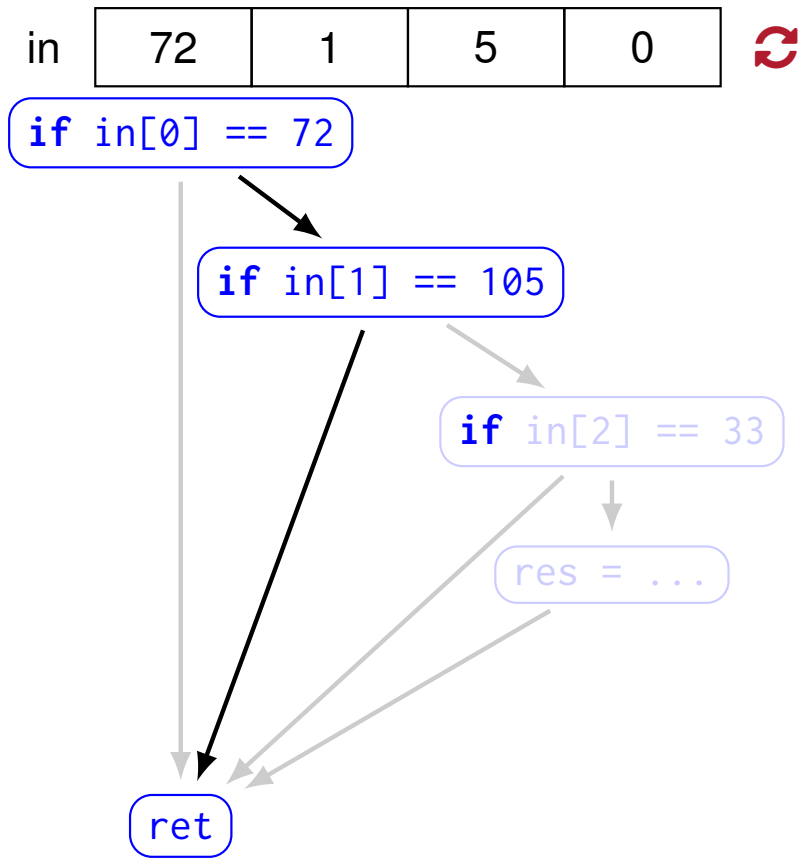
Überdeckungsgesteuertes Wuscheln (engl. *Fuzzing*)

- Raten möglicher Eingaben
- Strukturiertes Vorgehen:
  möglichst alle Blöcke gleichmäßig abdecken

  ⇝ Aus Modifikation von Eingaben die zum Vorgänger führten

  ⇝ *Mutation*: Zufällige Veränderung
  Crossover: Kombination existierender Eingaben
- Beispiel:

```
if (size < 4) return 0;
if(in[0] == 72)
  if(in[1] == 105)
    if(in[2] == 33)
      res = 42 / in[3];
return res;
```



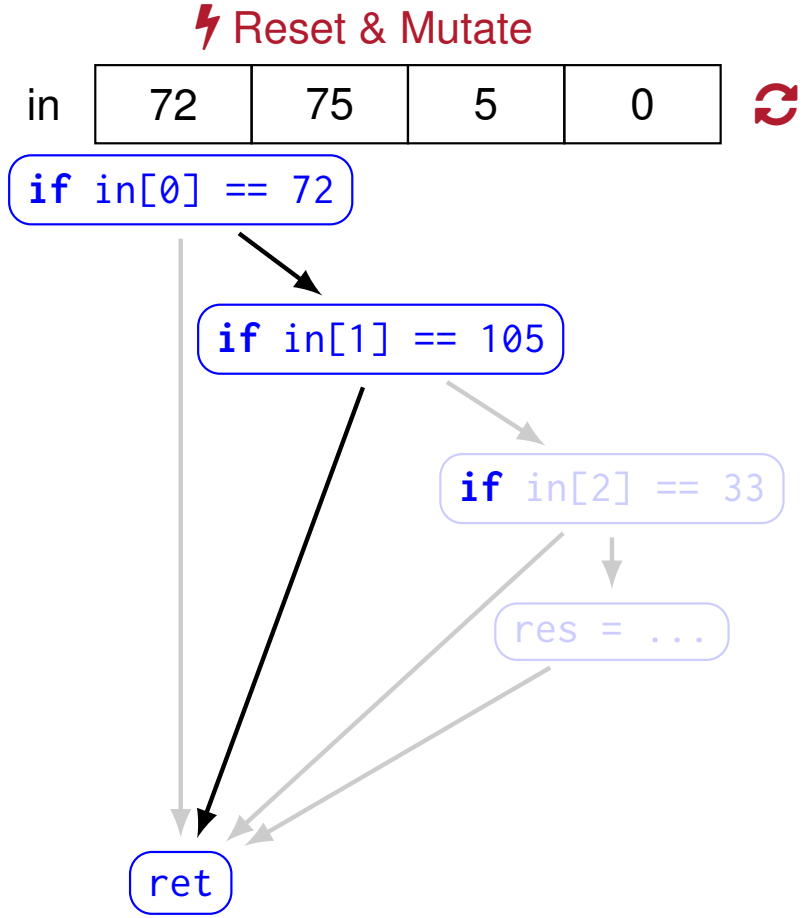
Überdeckungsgesteuertes Wuscheln (engl. *Fuzzing*)

- Raten möglicher Eingaben
- Strukturiertes Vorgehen: möglichst alle Blöcke gleichmäßig abdecken

  ⇝ Aus Modifikation von Eingaben die zum Vorgänger führten

  ⇝ *Mutation*: Zufällige Veränderung
  Crossover: Kombination existierender Eingaben
- Beispiel:

```
if (size < 4) return 0;
if(in[0] == 72)
  if(in[1] == 105)
    if(in[2] == 33)
      res = 42 / in[3];
return res;
```



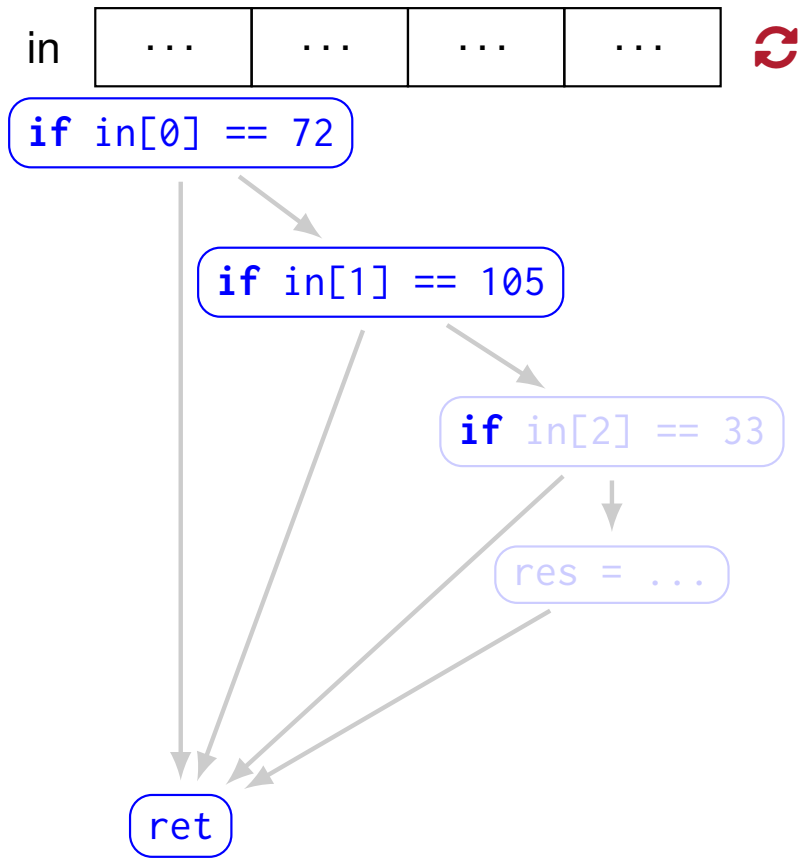
Überdeckungsgesteuertes Wuscheln (engl. *Fuzzing*)

- Raten möglicher Eingaben
- Strukturiertes Vorgehen:
  möglichst alle Blöcke gleichmäßig abdecken

  ⇝ Aus Modifikation von Eingaben die zum Vorgänger führten

  ⇝ *Mutation*: Zufällige Veränderung
  Crossover: Kombination existierender Eingaben
- Beispiel:

```
if (size < 4) return 0;
if(in[0] == 72)
  if(in[1] == 105)
    if(in[2] == 33)
      res = 42 / in[3];
return res;
```



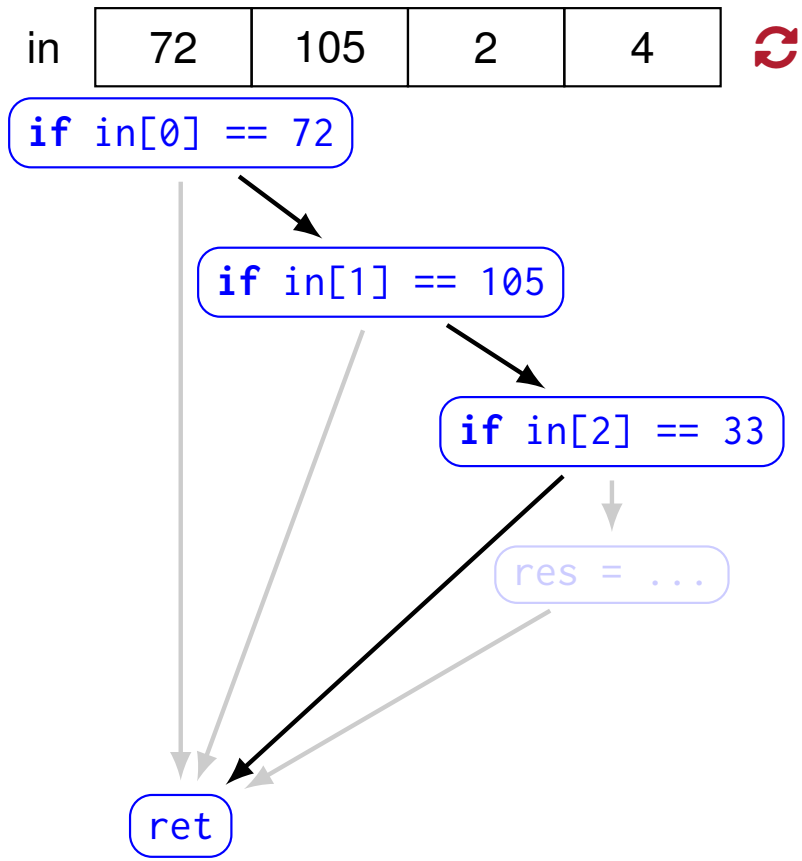
Überdeckungsgesteuertes Wuscheln (engl. *Fuzzing*)

- Raten möglicher Eingaben
- Strukturiertes Vorgehen:
  möglichst alle Blöcke gleichmäßig abdecken

  ⇝ Aus Modifikation von Eingaben die zum Vorgänger führten

  ⇝ *Mutation*: Zufällige Veränderung
  Crossover: Kombination existierender Eingaben
- Beispiel:

```
if (size < 4) return 0;
if(in[0] == 72)
  if(in[1] == 105)
    if(in[2] == 33)
      res = 42 / in[3];
return res;
```



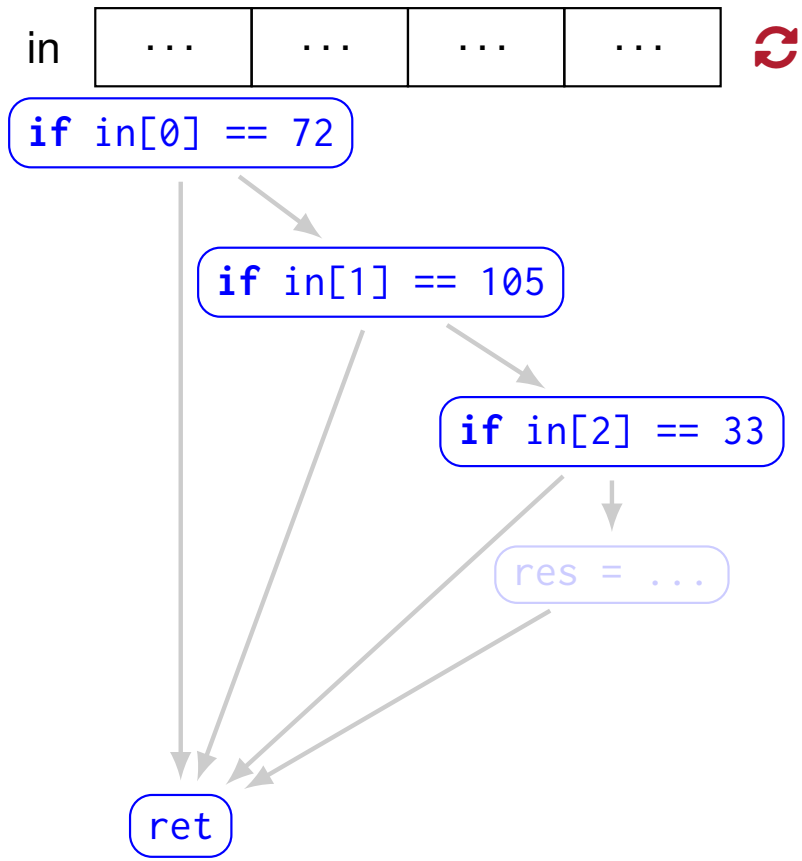
Überdeckungsgesteuertes Wuscheln (engl. *Fuzzing*)

- Raten möglicher Eingaben
- Strukturiertes Vorgehen:
  möglichst alle Blöcke gleichmäßig abdecken

  ⇝ Aus Modifikation von Eingaben die zum Vorgänger führten

  ⇝ *Mutation*: Zufällige Veränderung
  Crossover: Kombination existierender Eingaben
- Beispiel:

```
if (size < 4) return 0;
if(in[0] == 72)
  if(in[1] == 105)
    if(in[2] == 33)
      res = 42 / in[3];
return res;
```



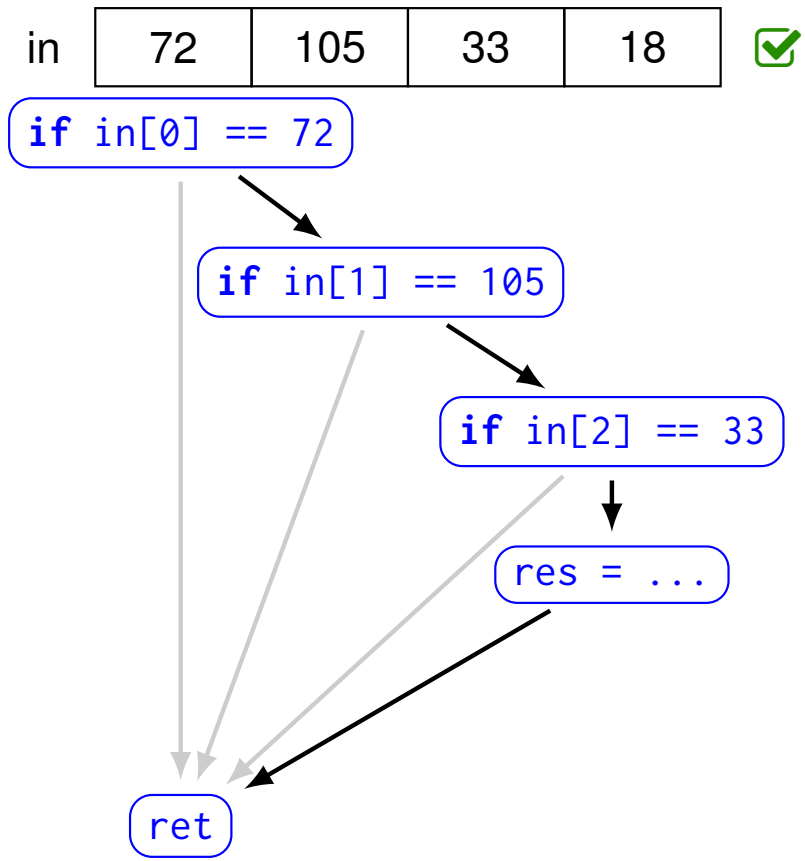
Überdeckungsgesteuertes Wuscheln (engl. *Fuzzing*)

- Raten möglicher Eingaben
- Strukturiertes Vorgehen:
  möglichst alle Blöcke gleichmäßig abdecken

  ⇝ Aus Modifikation von Eingaben die zum Vorgänger führten

  ⇝ *Mutation*: Zufällige Veränderung
  Crossover: Kombination existierender Eingaben
- Beispiel:

```
if (size < 4) return 0;
if(in[0] == 72)
  if(in[1] == 105)
    if(in[2] == 33)
      res = 42 / in[3];
return res;
```



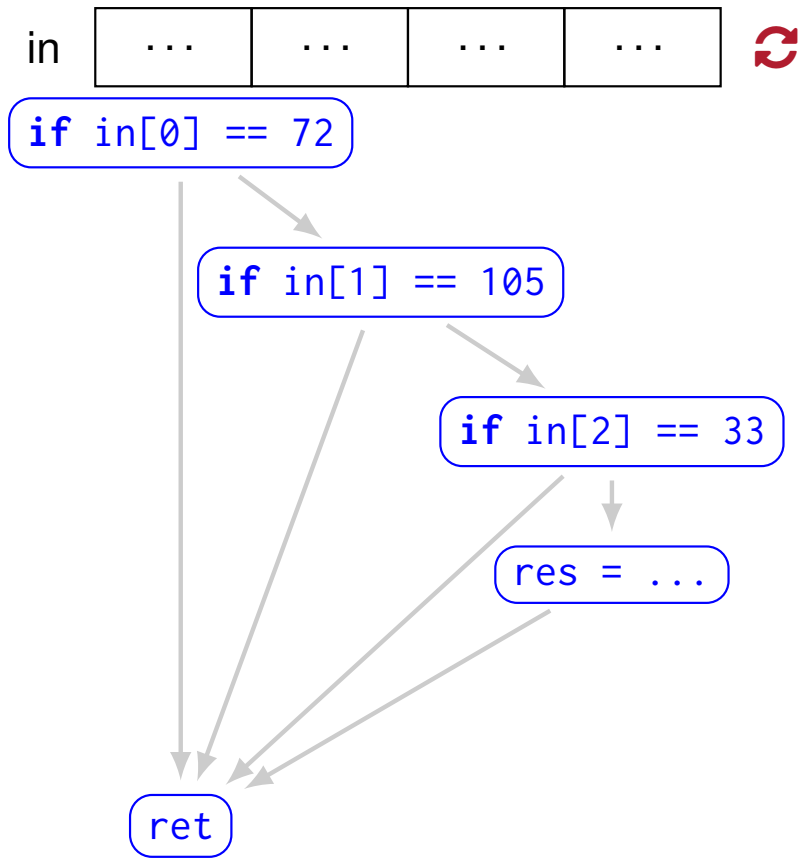
Überdeckungsgesteuertes Wuscheln (engl. *Fuzzing*)

- Raten möglicher Eingaben
- Strukturiertes Vorgehen:
  möglichst alle Blöcke gleichmäßig abdecken

  ⇝ Aus Modifikation von Eingaben die zum Vorgänger führten

  ⇝ *Mutation*: Zufällige Veränderung
  Crossover: Kombination existierender Eingaben
- Beispiel:

```
if (size < 4) return 0;
if(in[0] == 72)
  if(in[1] == 105)
    if(in[2] == 33)
      res = 42 / in[3];
return res;
```



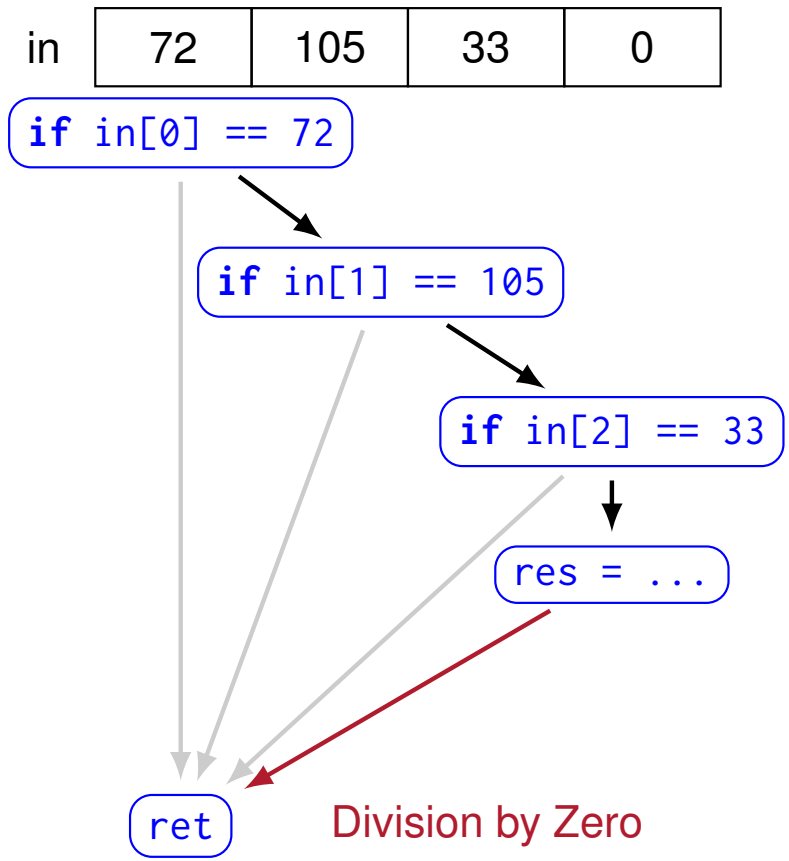
Überdeckungsgesteuertes Wuscheln (engl. *Fuzzing*)

- Raten möglicher Eingaben
- Strukturiertes Vorgehen:
  möglichst alle Blöcke gleichmäßig abdecken

  ⇝ Aus Modifikation von Eingaben die zum Vorgänger führten

  ⇝ *Mutation*: Zufällige Veränderung
  Crossover: Kombination existierender Eingaben
- Beispiel:

```
if (size < 4) return 0;
if(in[0] == 72)
  if(in[1] == 105)
    if(in[2] == 33)
      res = 42 / in[3];
return res;
```



Überblick

Aufgabe 5 – Testen

- Verwendung von GNU/Linux (kein eCos mehr)
- Ziele
  1. Testfokusierter Softwareentwurf
  2. Testfallentwurf
     - Vollständige Pfadüberdeckung
     - Abdecken aller Randfälle
  3. Implementierung von Software und Testfällen
     - Getrennte Implementierung von Software und Testfällen
     - Möglichst durch verschiedene Übungsteilnehmer

     ⇝ Peer-Review
- Implementiert werden soll eine *Prioritätswarteschlange*
- Einfügen, Entfernen, *Iterieren*

⇝ `for (... x = ...; x = ...; ++x) ...!`

- Implementierung?

Iterator – 1. Versuch

- *Datenstruktur als Array* im Header vereinbaren
- Zugriff durch Zeigerarithmetik

```
1 typedef struct Element { ... } Element;
2 Element elements[ELEMENTS_SIZE];
3 ...
4   for (size_t i = 0; i < ELEMENTS_SIZE; ++i)
5     { use(elements[i]); }
```

Iterator – 1. Versuch

- *Datenstruktur als Array* im Header vereinbaren
- Zugriff durch Zeigerarithmetik

```
1 typedef struct Element { ... } Element;
2 Element elements[ELEMENTS_SIZE];
3 ...
4   for (size_t i = 0; i < ELEMENTS_SIZE; ++i)
5     { use(elements[i]); }
```

- *Vorteile:*
  - Einfache Implementierung
  - Für den Compiler leicht zu optimieren
- *Nachteil:* Implementierung offen gelegt

$\rightsquigarrow$ Verpflichtung gegenüber Benutzer

Iterator – 2. Versuch

- *Iterator als Teil des Objekts*
- Header:

```
1 typedef struct Elements Elements;
2 void El_reset_iterator(Elements *self);
3 void El_next(Elements *self);
4 bool El_isAtEnd(Elements *self);
5 int64_t El_iterator_value(Elements *self);
```

- Verwendung:

```
1 El_reset_iterator(dings);
2 while(!El_isAtEnd(dings)) {
3   use(El_iterator_value(dings));
4   El_next(dings);
5 }
```

Iterator – 2. Versuch

- Implementierung:

```
1  typedef struct Element { int64_t value; } Element;
2  struct Elements {
3    Element elements[ELEMENTS_SIZE];
4    Element *it;
5  };
6  void El_reset_iterator(Elements *self)
7    { self->it = &self->elements }
8  void El_next(Elements *self)
9    { self->it = self->it + 1; }
10 bool El_isAtEnd(Elements *self)
11   { return self->it
12       == &(self->elements[ELEMENTS_SIZE]); }
13 int64_t El_iterator_value(Elements *self)
14   { return self->it->value; }
```

Iterator – 2. Versuch

- Implementierung:

```
1  typedef struct Element { int64_t value; } Element;
2  struct Elements {
3    Element elements[ELEMENTS_SIZE];
4    Element *it;
5  };
6  void El_reset_iterator(Elements *self)
7    { self->it = &self->elements }
8  void El_next(Elements *self)
9    { self->it = self->it + 1; }
10 bool El_isAtEnd(Elements *self)
11   { return self->it
12       == &(self->elements[ELEMENTS_SIZE]); }
13 int64_t El_iterator_value(Elements *self)
14   { return self->it->value; }
```

- *Vorteil:* Kapselung sehr gut
- *Nachteile:*
  - Für den Compiler evtl. nicht mehr optimierbar (Schleife ausrollen)
  - So nur ein Iterator gleichzeitig möglich

Iterator – 3. Versuch

- *Iterator als eigenes Objekt*
- Header:

```
1  typedef struct Elements Elements;
2  typedef struct El_Iterator El_Iterator;
3
4  El_Iterator *El_begin(Elements *self);
5  void El_Iterator_destroy(El_Iterator *self);
6  void El_Iterator_next(El_Iterator *self);
7  bool El_Iterator_isAtEnd(El_Iterator *self);
8  int64_t El_Iterator_value(El_Iterator *self);
```

- Verwendung:

```
1  El_Iterator *it;
2  for (it = El_begin(dings);
3       not El_Iterator_isAtEnd(it);
4       El_Iterator_next(it)) {
5    use(El_Iterator_value(it))
6  }
7  El_Iterator_destroy(it);
```

Iterator – 3. Versuch

- Implementierung:

1em

```
1  typedef struct Element { int64_t value; } Element;
2  struct Elements { Element elements[ELEMENTS_SIZE]; };
3  struct El_Iterator {
4    Element *position;
5    Element *end;
6  };
7
8  El_Iterator *El_begin(Elements *self) {
9    El_Iterator *ret = malloc(sizeof(El_Iterator));
10   if (ret == NULL) { return NULL; }
11   ret->position = self->elements;
12   ret->end = &self->elements[ELEMENTS_SIZE];
13   return ret;
14 }
15
16 void El_Iterator_next(El_Iterator *self)
17   { self->position += 1; }
18 bool El_Iterator_isAtEnd(El_Iterator *self) { ... }
19 int64_t El_Iterator_value(El_Iterator *self) { ... }
20 void El_Iterator_destroy(El_Iterator *self) { ... }
```

Iterator – 3. Versuch

- *Vorteile:*
  - Vollständige Kapselung
  - Beliebig viele Iteratoren möglich
- *Nachteil:*
  - Iterator muss nach Gebrauch beseitigt werden
  - Compiler hat evtl. Probleme zu optimieren

Fragen...

Literatur I

Konstantin Serebryany, Derek Bruening, Alexander Potapenko, and Dmitriy Vyukov.
AddressSanitizer: A fast address sanity checker.
In *Proceedings of the USENIX Annual Technical Conference*, pages 309–318, 2012.