КАК ГЕНЕРАТОР ТЕСТОВ ПОМОГ СТАБИЛИЗИРОВАТЬ КОМПИЛЯТОР В ZING VM

Макс Казанцев

Нина Ринская

Azul Systems

max.kazantsev@azul.com

О ЧЁМ ЭТОТ ДОКЛАД?

- Проблема поиска функциональных багов (на примере JIT-компилятора Java)
- Подходы к решению
- Генераторы тестов как один из подходов
- Устройство и использование Fuzzer

ВИДЫ БАГОВ

- Краши
- Зависания/неприемлемо долгое время работы
- Некорректные результаты

ОБЩАЯ ПРОБЛЕМА

- Мы делаем продукт, который быстро развивается
- Постоянно выходят новые версии
- Хотим, чтобы между версиями не было функциональных регрессий
- Хотим находить и фиксить баги до того, как их найдёт пользователь

ПОДХОД ПРИМЕНИМ НЕ ТОЛЬКО К VM

- Я делаю конвертер json в xml
- У меня есть reference-имплементация
- А теперь я хочу переписать её с Java на Scala
- Как проверить, что новая имплементация работает?
- Как проверить, что каждая очередная версия не хуже предыдущей?

ПОДХОДЫ К ПОИСКУ БАГОВ

- Статические тесты (unit, regression, реальные приложения и т.п.)
- Измерение покрытия и добавление недостающих тестов на его основе
- Fuzzing входных данных

ЧТО ТАКОЕ FUZZING

- От англ. Fuzzy – “нечёткий”, “пушистый”
- Заключается в автоматизированной подаче на вход случайных данных
- Широко применяется там, где требуется высокая степень надёжности
- Fuzzing для компилятора – генератор тестов

НЕМНОГО МАТЕМАТИКИ

- Работает не баллистика, а статистика
- Пусть “хороший” тест находит баг с вероятностью 0.000001
- Тогда с вероятностью 0.999999 такой тест не найдёт бага
- А если тест не один, а 5 миллионов?
- $0.999999^{5000000} < 0.007$
- Вероятность, что ни один из них не найдёт баг, менее 1%

ОБЗОР ФАЗЗЕРОВ

- LibFuzzer: часть проекта LLVM (https://llvm.org/docs/LibFuzzer.html)
- American Fuzzy Lop (http://lcamtuf.coredump.cx/afl/)
- Mull mutation (https://github.com/mull-project/mull)
- JitTester (by Oracle, WIP)
- Java Fuzzer for Android (https://github.com/android-art-intel/Fuzzer)
- Java Fuzzer (https://github.com/AzulSystems/JavaFuzzer)

ЧТО ТАКОЕ ZING VM И FALCON

- Zing VM – виртуальная машина языка Java
- LLVM – фреймворк для построения компиляторов
- Falcon – JIT-компилятор Zing VM, основанный на LLVM

ПОЧЕМУ БАГИ ТАК СЛОЖНО ИСКАТЬ?

- Объём кода компилятора
  - >80 MB исходников
- Код быстро изменяется
  - Порядка 70к строк изменений в неделю
- JIT-компиляция зависит от рантайма => меньше детерминизма
- Каждый метод компилируется несколько раз

А ЕЩЁ БАГИ МОГУТ ХОРОШО ПРЯТАТЬСЯ

```
int foo() {
    int a = ... // Correct
    int b = ... // Buggy value
    if (rnd.nextInt(100) == 42)
        return b;
    return a;
}
```

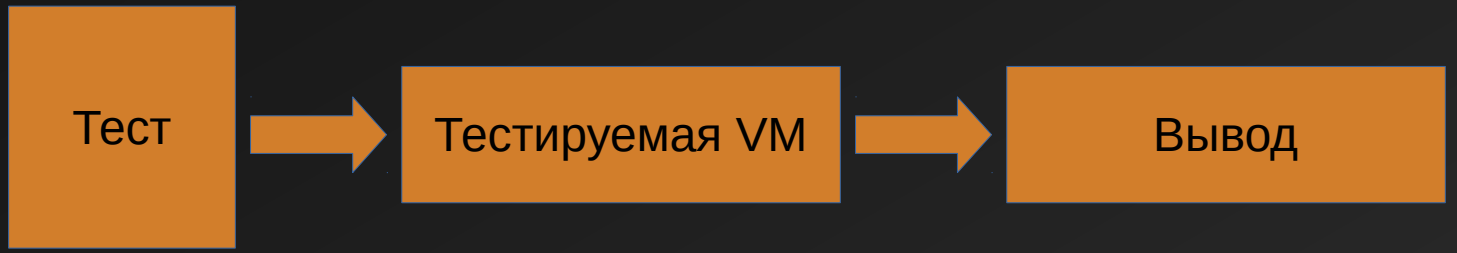
ИСТОРИЯ JAVA FUZZER

- Создан в Intel как средство тестирования Dalvik и ART VM
- Язык программирования – Ruby
- Код открыт: https://github.com/android-art-intel/Fuzzer
- Выпущена версия для сервера/десктопа: https://github.com/AzulSystems/JavaFuzzer
- Активно используется для стабилизации Falcon

ЧТО ТАКОЕ ХОРОШИЙ ТЕСТ ДЛЯ КОМПИЛЯТОРА?

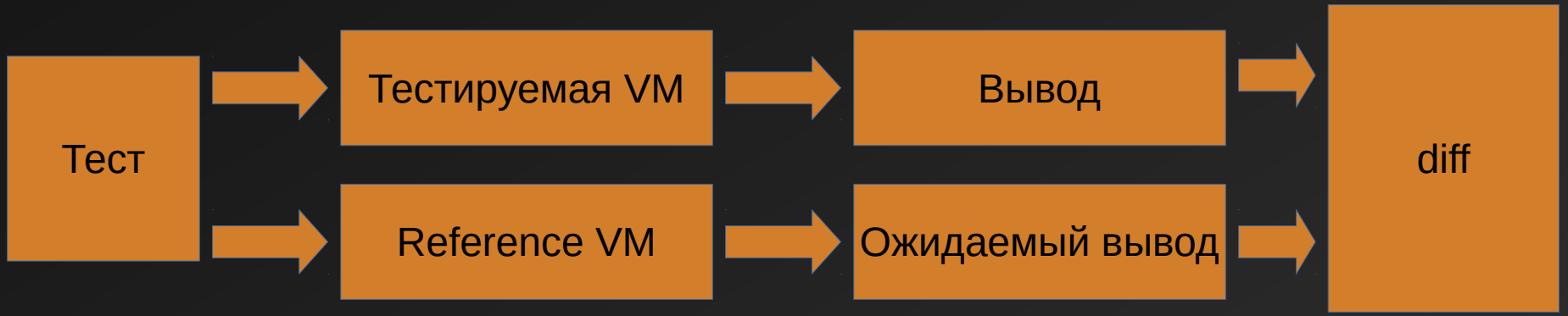
- Доступен в исходном коде
- Всегда выдаёт один и тот же результат
- Достаточно быстро исполняется
- Все переменные участвуют в ответе
- Находит баг с “хорошей” вероятностью

И ЧТО ДЕЛАТЬ С ХОРОШИМ ТЕСТОМ?

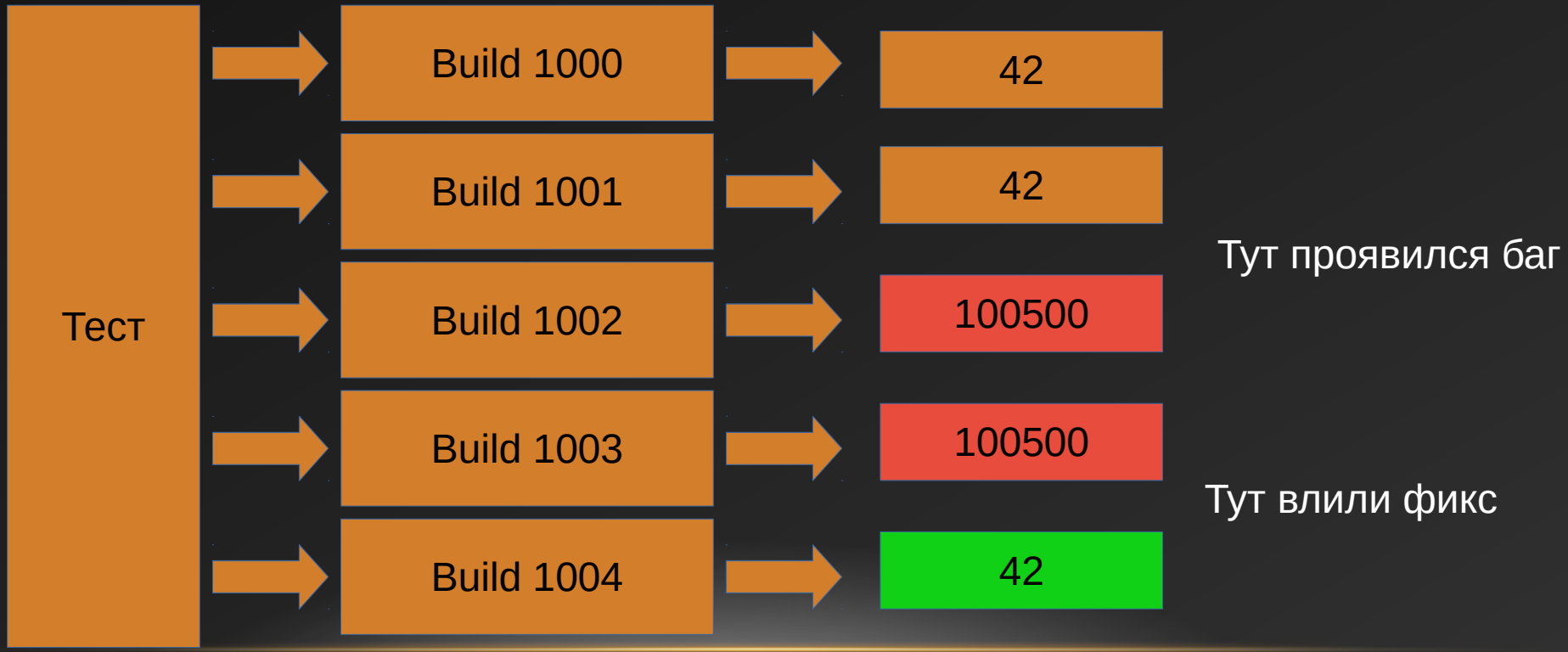


Как понять, он правильный или нет?

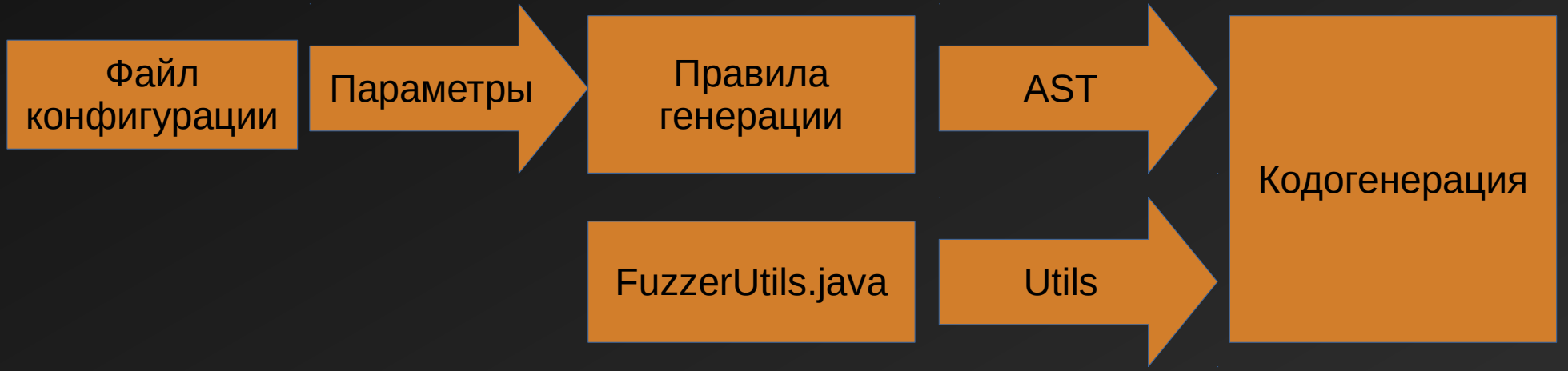
НУЖЕН REFERENCE!



ОТЛОВ РЕГРЕССИЙ



УСТРОЙСТВО FUZZER



КАКИЕ КОНСТРУКЦИИ FUZZER УМЕЕТ ГЕНЕРИРОВАТЬ

- Циклы (for, while, do-while, enhanced for; break, continue)
- Ветвления (if, switch)
- Вызовы методов (рекурсия запрещена)
- Арифметика (целочисленная, floating-point, логические операции)
- Исключения, try-catch-finally
- Массивы (одномерные, многомерные)
- Классы и объекты
- Потоки (в ограниченном режиме, только краши)
- Переопределение методов (в ограниченном режиме, нет abstract)

СКОРОСТЬ ГЕНЕРАЦИИ

- Среднее время генерации теста – 0.2с
- Соотношение времени генерации ко времени исполнения – 1:100

ФАЙЛ КОНФИГУРАЦИИ

- Ограничения (min/max)
- Допустимые типы и операции
- Вероятности появления конструкций
- Нужно пострессить фичу? Делаем конфиг под неё!

```
115  #       class          weight  weight-in-loop   scale-down-factor
116  statements: {
117      ForLoopStmt:     [48,         24,         1.5 ],
118      WhileDoStmt:     [16,          8,         1.5 ],
119      EnhancedForStmt: [ 4,          2,         1.5 ],
120      ContinueStmt:    [ 0,          1,           1 ],
121      BreakStmt:       [ 0,          1,           1 ],
122      IfStmt:          [ 3,          4,         1.5 ],
123      SwitchStmt:      [ 1,          1,           1 ],
124      AssignmentStmt:  [ 1,         80,           1 ],
125      IntDivStmt:      [ 0,          1,           1 ],
126      ReturnStmt:      [ 0,          1,           1 ],
127      TryStmt:         [ 1,          1,           2 ],
128      ExcStmt:         [ 1,          1,           2 ],
129      VectStmt:        [ 0,          8,           1 ],
130      InvocationStmt:  [ 1,          1,           1 ],
131      CondInvocStmt:   [ 1,          1,           2 ],
132      SmallMethStmt:   [ 10,         1,           2 ],
133      NewThreadStmt:   [ 40,         1,           2 ]
```

ПРАВИЛА ГЕНЕРАЦИИ

<Stmt> → <ForLoopStmt> | <WhileLoopStmt> | … | <SetStmt>

<SetStmt> → <Variable> = <Expr>;

<Expr> → <Terminal> | <UnOp> (<Expr>) | (<Expr>) <BinOp> (<Expr>) | <Call>

<Terminal> → <Constant> | <Variable>

<BinOp> → * | + | … | && | ...

ГЕНЕРАЦИЯ МЕТОДА (ШАГ 1)

```
<MethodType> <MethodName> (<MethodParams>) {
    <MethodBody>
}
```

ГЕНЕРАЦИЯ МЕТОДА (ШАГ 2)

```
int iMeth4 (int i1, float f1, int i2) {
    <Decls>
    <Stmt>
    <Stmt>
    ...
    <Stmt>
    <ControlSum>
}
```

ГЕНЕРАЦИЯ МЕТОДА (ШАГ 3)

```
int iMeth4 (int i1, float f1, int i2) {
    int i3 = 24;
    float f2 = -24.34f,
    for (i3 = 5; i3 < 10004; i3++) {
        <Stmt>
        <Stmt>
    }
    if (i3 > i2 && f2 < (4 * i3 + 9 * i2 - 24.54f))
        i2++;
    <CtrlSum>
}
```

ГЕНЕРАЦИЯ МЕТОДА (ШАГ 4)

```
int iMeth4 (int i1, float f1, int i2) {
    int i3 = 24;
    float f2 = -24.34f,
    int[] iArr1 = new int[200];
    FuzzerUtrils.init(iArr1);
    for (i3 = 5; i3 < 10004; i3++) {
        iArr1[i3 + 1] = (int) (f1 + f2 + i3 * 2);
        f2 += iArr1[i3] * iArr2[i3 - 1];
    }
    if (i3 > i2 && f2 < (4 * i3 + 9 * i2 - 24.54f))
        i2++;
    <CtrlSum>
}
```

ГЕНЕРАЦИЯ МЕТОДА (ШАГ 5)

```
int iMeth4 (int i1, float f1, int i2) {
    int i3 = 24;
    float f2 = -24.34f;
    int[] iArr1 = new int[200];
    FuzzerUtrils.init(iArr1);
    for (i3 = 5; i3 < 10004; i3++) {
        iArr1[i3 + 1] = (int) (f1 + f2 + i3 * 2);
        f2 += iArr1[i3] * iArr2[i3 - 1];
    }
    if (i3 > i2 && f2 < (4 * i3 + 9 * i2 - 24.54f))
        i2++;
    iMeth4CtrlSum += i1 + i2 + i3 + Float.floatToIntBits(f1) +
        Float.floattoIntBits(f1) + FuzzerUtils.checkSum(iArr1);
    return iMeth4CtrlSum;
}
```

ТИПИЧНЫЙ ТЕСТ

```
public static void vMeth1() {

    int i7=54703, i8=-14, i18=-32644, i19=4, i20=8, i21=-1, iArr1[]=new int[N];
    short sArr[]=new short[N];
    long lArr1[]=new long[N];

    FuzzerUtils.init(sArr, (short)28174);
    FuzzerUtils.init(iArr1, 5);
    FuzzerUtils.init(lArr1, -50290L);

    for (i7 = 11667; i7 > 225; i7--) {
        switch ((i7 % 3) + 74) {
        case 74:
            for (i18 = 4; i18 < 175; i18++) {
                for (i20 = i7; i20 < 2; ++i20) {
                    double d1=-20.76874;
                    sArr[i18] = (short)(((long)(d1 / (i7 | 1)) >> iArr1[i7]) * i20);
                    i19 = (int)8563533763630855434L;
                    Cls1.fFld += ((lArr1[i7 + 1] - (++i8)) + i18);
                    i19 += (int)(-Cls.instanceCount);
                    if (i7 != 0) {
                        vMeth1_check_sum += i7 + i8 + i18 + i19 + i20 + i21 + FuzzerUtils.checkSum(sArr) +
                            FuzzerUtils.checkSum(iArr1) + FuzzerUtils.checkSum(lArr1);
                        return;
                    }
                    try {
                        iArr1[i20] = (240 / Test.iFld1);
                        iArr1[i20 - 1] = (-44739 % i21);
                        i19 = (Test.iFld1 / -34797);
                    } catch (ArithmeticException a_e) {}
                    i8 += (i20 | Test.instanceCount);
                    Test.iFld1 = (i21--);
                }
            }
            break;
        case 75:
            try {
                Test.iFld1 = (5441 / i18);
                iArr1[i7] = (i20 % -49571);
                i19 = (i8 % 130);
            } catch (ArithmeticException a_e) {}
        case 76:
            i8 |= (int)((-Cls.instanceCount) - (iArr1[i7]--));
            break;
        default:
            i8 -= i18;
        }
    }
    vMeth1_check_sum += i7 + i8 + i18 + i19 + i20 + i21 + FuzzerUtils.checkSum(sArr) + FuzzerUtils.checkSum(iArr1)
        + FuzzerUtils.checkSum(lArr1);
}
```

ОН ЖЕ ПОСЛЕ СОКРАЩЕНИЯ

```
1   public static void vMeth1() {
2       int i7=54703, iArr1[] = new int[N];
3       for (i7 = 228; i7 > 225; i7--) {
4           switch ((i7 % 3)) {
5           case 0:
6               for (int i18 = 4; i18 < 175; i18++) {
7                   for (int i20 = i7; i20 < 2; ++i20) {
8                       if (i7 != 0) {
9                           return;
10                      }
11                      iArr1[i20 - 1] = 1;
12                  }
13              }
14          default:
15          }
16      }
17      vMeth1_check_sum += i7;
18  }
```

RANGE CHECK

```
arr[index] = 40;
if (index < 0 || index >= arr.length)
    throw new ArrayIndexOutOfBoundsException(…);
store(arr, index, 40);
```

RANGE CHECK

```
arr[index] = 40;
if (index >=u arr.length)
    throw new ArrayIndexOutOfBoundsException(…);
store(arr, index, 40);
```

INDUCTIVE RANGE CHECK

```
for (i = 0; i < N; i += step) {
    index = i + offset;
    if (index  >=u arr.length)
        throw new ArrayIndexOutOfBoundsException(...);
    store(arr, index, 40);
}
```

INDUCTIVE RANGE CHECK ELIMINATION

```
for (i = 0; i < Start; i += step) {
    <Range check>
}
for (; i < End; i += step) {
    index = i + offset;
    store(arr, index, 40);
}
for (; i < N; i += step) {
    <Range check>
}
```

КАК НАЙТИ [START; END)?

- i изменяется от 0 до N
- index = i + offset
- 0 <= index < arr.length – безопасные значения
- Следовательно, -offset <= i < arr.length – offset
- Start = max(0, -offset)
- End = min(arr.length – offset, N)

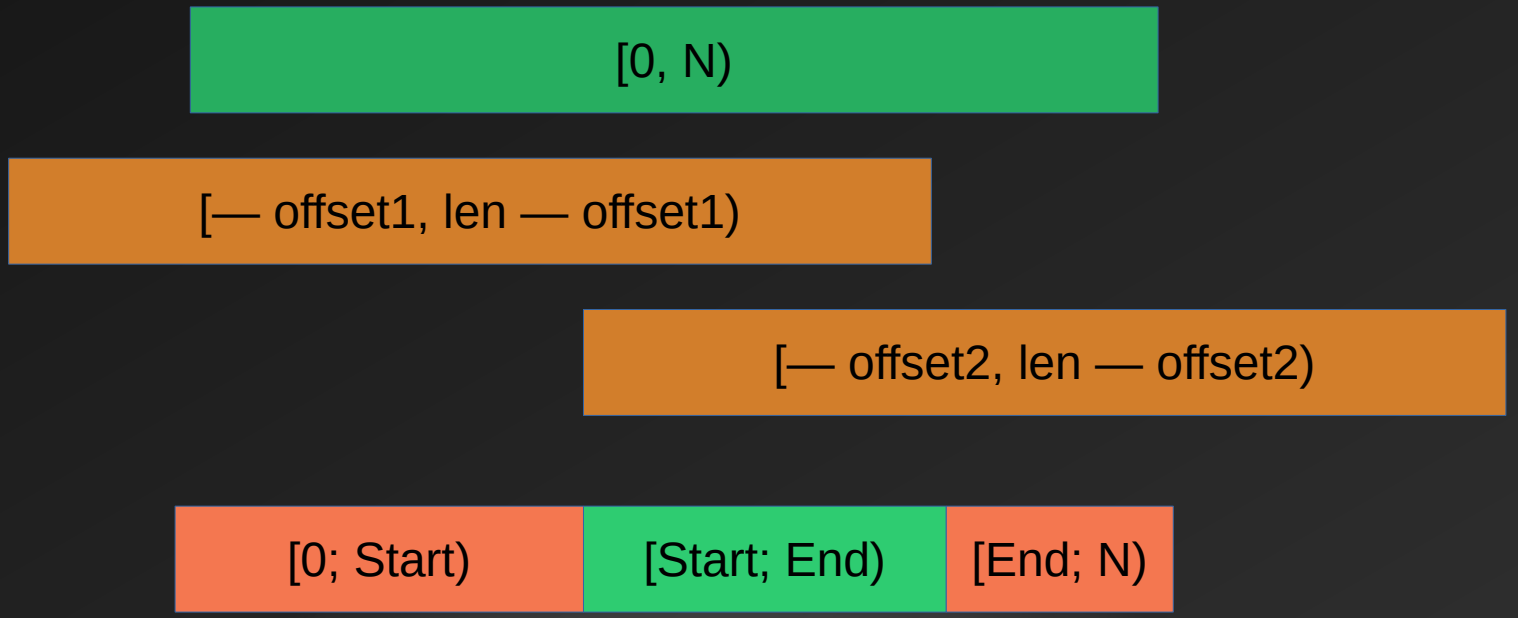
А ЕСЛИ RANGE CHECK’ОВ МНОГО?

```
for (i = 0; i < N; i += step) {
    arr[i + offset1] = …;
    arr[i + offset2] = …;
    …
    arr[i + offsetK] = …;
}
```

Start = max(0, -offset1, -offset2, …, -offsetK)

End = min(N, length – offset1, length – offset2, …, length - offsetK)

КАК НАЙТИ [START; END)?



ТАК В ЧЁМ ЖЕ ПРОБЛЕМА?

- В LLVM нет min/max нет!
- Есть umin/umax, smin/smax
- Отрицательные числа в беззнаковых сравнениях – очень большие числа

ЧТО БЫЛО

- IRCE поддерживал только знаковые сравнения
- Везде использовался smin/smax
- Все были счастливы, но...

ВЫХОД ИЗ ЦИКЛА ПО UNSIGNED

```
for (int i = 0; i <s N; i += step) {
    int index = i + offset;
    if (index >=u arr.length)
        throw new ArrayIndexOutOfBoundsException(...);
    store(arr, index, 40);
}
```

```
for (int i = 0; i <u N; i += step) {
    int index = i + offset;
    if (index >=u arr.length)
        throw new ArrayIndexOutOfBoundsException(...);
    store(arr, index, 40);
}
```

КАКОЙ MIN/MAX ВЫБРАТЬ?

- Индукционную переменную можно рассматривать как беззнаковую, если условие выхода из цикла беззнаковое
- Если выхода из цикла => umin/umax, иначе smin/smax
- Вроде, логично… Полетели!

ПРОШЛО 2 НЕДЕЛИ

```
public void mainTestReduced(String[] strArr1) {
    int i6=-5, i7=42788, i8=-35635, i9=46643, i10=-11, i11=-22389, i12=-24142;
    long lArr[]=new long[N];
    double dArr[]=new double[N];

    i7 = TestReduced.UnknownZero ;

    for (i8 = 2; i8 < 79; i8++) {
        TestReduced.instanceCount = ((TestReduced.instanceCount++) - ((TestReduced.instanceCount + -170) - (-1815486561L & (~(long)(-147
            * (TestReduced.fFld + i6))))));
        for (double d : dArr) {
            TestReduced.fFld -= (-TestReduced.iArrFld[i8 - 1]);
            d = ((++TestReduced.fFld) + (i9 += (int)(TestReduced.dFld * d)));
            for (i10 = 1; i10 < i8; i10 += 3) {
                TestReduced.instanceCount += i10;
                TestReduced.iFld += (-9582 + (i10 * i10));
                TestReduced.iArrFld[i10] >>= TestReduced.iFld;
                byArrFld[i8 - 1] >>= (byte)Math.min((++i9) - i7, (int)((i9 - 1L) - (i8 - i9)));

                try {
                    TestReduced.instanceCount += (((i10 * TestReduced.instanceCount) + i9) - TestReduced.fFld);
                    TestReduced.instanceCount = ((i7++) - ((i11 - i9) * (TestReduced.iArrFld[i8]++)));
                }
                catch (NegativeArraySizeException exc) {
                    lArr[i10 - 1] -= i11;
                }
                finally {
                    byArrFld[i8] = (byte)TestReduced.iArrFld[i8 + 1];
                    TestReduced.fFld = i7;
                    TestReduced.instanceCount %= ((TestReduced.instanceCount + TestReduced.iArrFld[i10]) | 1);
                }

                i11 = (i11--);
            }
        }
    }

}
```

НАШЛИ, ПОЧИНИЛИ

- В одном месте неправильно берётся max
- Поправили
- Всё починилось, тесты ходят, “на глазок” ошибок не нашли
- Летим дальше...

И СНОВА ЗДРАВСТВУЙТЕ

```
public static void vMeth() {
    int i31=-45, i32=19462, i33=-9, i34=-43566, i35=-225, iArr4[]=new int[N], iArr5[]=new int[N];
    float f4=-72.285F;
    short s3=14412;
    TestReduced.instanceCount *= Integer.reverseBytes(-173);
    try {
        for (int i30 : iArr4) {
            iArr4[0] += Integer.reverseBytes(Math.min(i30, i30) - (i30 += 11));
            TestReduced.iFld1 += (int)(TestReduced.lArrFld[(TestReduced.iFld1 >>> 1) % N]++);
            for (i31 = 1; i31 < (4 + 400); i31++) {
                for (i33 = 1; i33 < (1 + 400); ++i33) {
                    TestReduced.iFld1 -= i30;
                    TestReduced.bFld = (i30 > ((i34 >>= i34) + Math.max(i33, i30)));
                }
            }
            iArr5[0] -= i34;
        }
        i35 = 2;
        while (++i35 < 207) {
            TestReduced.lArrFld[i35 - 1] = ((--i32) * ((iArr5[i35 - 1]--) - i31));
            iArr5[i34++] = (int)((Math.max(i34, i33)) + ((TestReduced.iFld1 + i34) - (19379 + (i33 - i31))));
        }
    }
    catch (ArrayIndexOutOfBoundsException exc3) {
        f4 = ((s3--) * ((TestReduced.instanceCount++) + iArr5[(TestReduced.iFld1 >>> 1) % N]));
    }
    vMeth_check_sum += i31 + i32 + i33 + i34 + i35 + Float.floatToIntBits(f4) + s3 ;
}
```

ПОХОЖЕ, МЫ ЧЕГО-ТО НЕ ПОНИМАЕМ

- Выключили unsigned comparison
- Стали изучать проблему
- Вскоре обнаружилось страшное...

СНОВА СТАРЫЕ ЗНАКОМЫЕ

```
for (i = 0; i < N; i += step) {
    arr[i + offset1] = …;
    arr[i + offset2] = …;
    …
    arr[i + offsetK] = …;
}
```

Start = s/umax(0, -offset1, -offset2, …, -offsetK)

End = s/umin(N, length – offset1, length – offset2, …, length - offsetK)

ПРОБУЕМ SMIN/SMAX

- Пусть offset1 = 5, offset2 = -5, len = SINT_MAX - 2
- Start = smax(0, 5, -5) = 5;
- End = smin(N, len + 5, len – 5) = len + 5!

ПРОБУЕМ SINT/SMAX

```
len = SINT_MAX - 2;

for (i = 0; i <s 5; I += step) {
    <Range check>
}
for (; i <s len + 5; I += step) {
    index = i + offset;
    store(arr, index, 40)
}
for (; i <s N; I += step) {
    <Range check>
}
```

ЛУЧШЕ ПОПРОБУЕМ UMIN/UMAX

- Пусть offset1 = 5, offset2 = -5, len = SINT_MAX - 2
- Start = umax(0, 5, -5) = -5;
- End = umin(N, len + 5, len – 5) = umin(N, len - 5)

ПРОБУЕМ UINT/UMAX

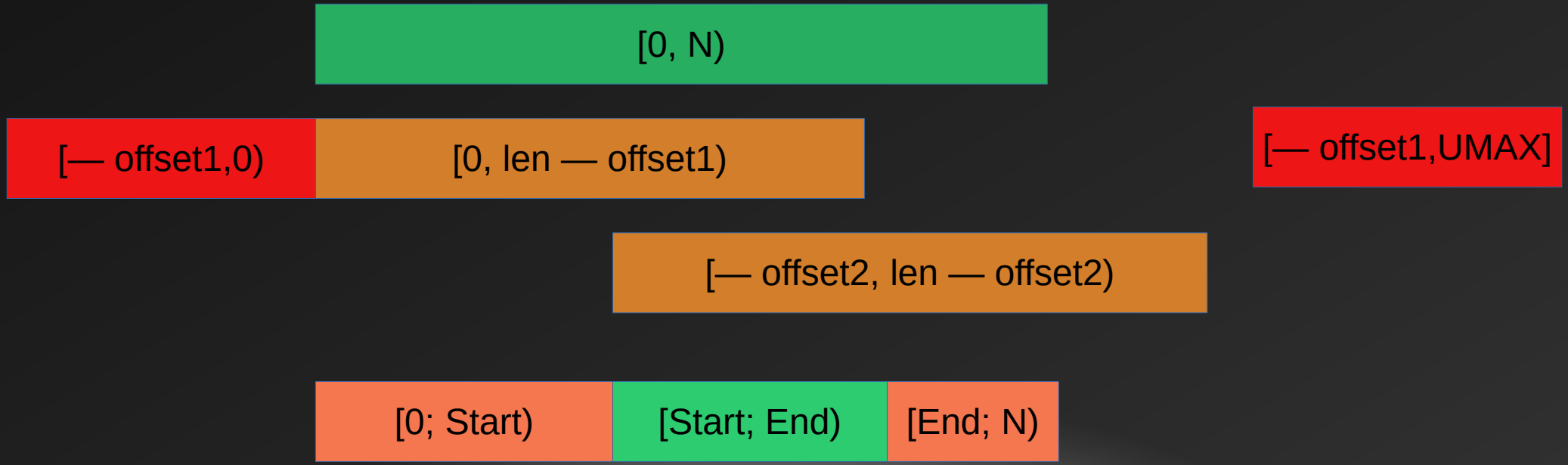
```
len = SINT_MAX - 2;

for (i = 0; i <u -5; I += step) {
    <Range check>
}
for (; i <u umin(N, len - 5); I += step) {
    index = i + offset;
    store(arr, index, 40)
}
for (; i <u N; I += step) {
    <Range check>
}
```

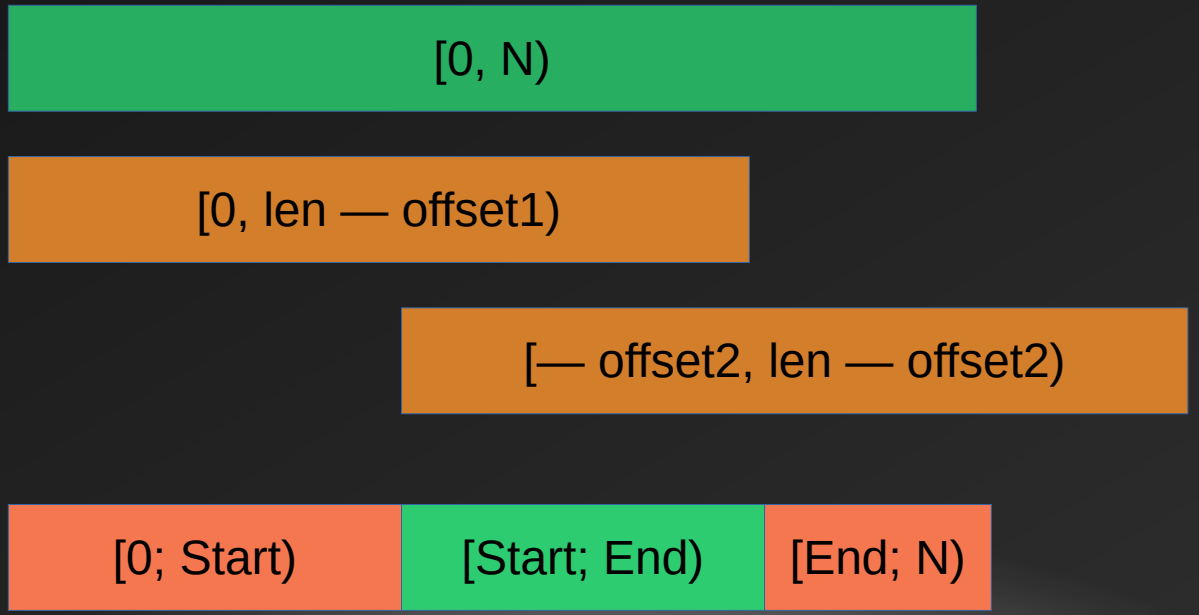
А ЧТО МЫ ХОТИМ?

- Пусть offset1 = 5, offset2 = -5, len = SINT_MAX - 2
- Start = smax(0, 5, -5) = 5;
- End = umin(len, N, len + 5, len – 5) = umin(N, len - 5)

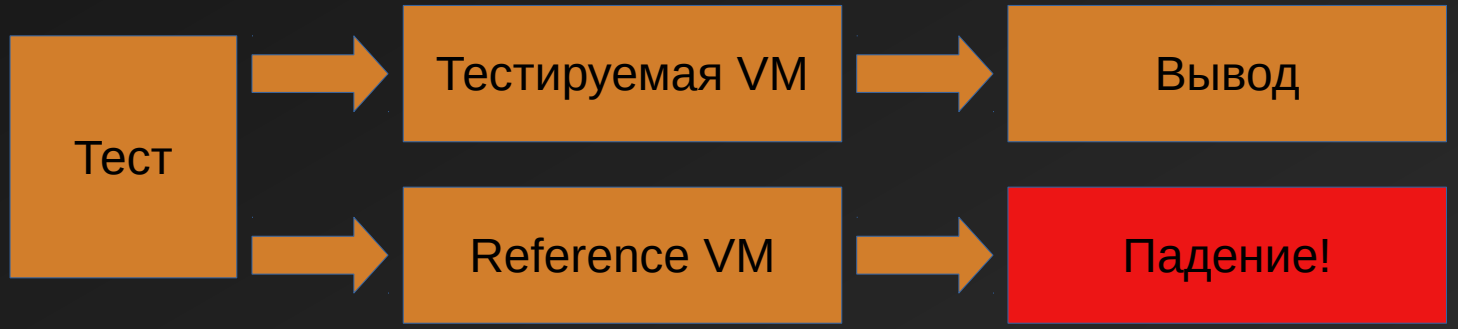
КАК ВСЁ БЫЛО НА САМОМ ДЕЛЕ



КАК ВСЁ БЫЛО НА САМОМ ДЕЛЕ



FUZZER НАХОДИТ БАГИ И В ДРУГИХ JAVA-МАШИНАХ



НАХОДИМ БАГИ, КОТОРЫЕ БЫЛИ В LLVM ГОДАМИ

https://bugs.llvm.org/show_bug.cgi?id=35743

```
char foo(char c) {
    for (int i = 1; i < 193; i++) {
        c &= (char) 1;
        c += (char) 3;
    }
    return c;
}
```

Впервые появился в clang v3.8

Должно быть 3 или 4. Возвращается 0 или 1

ЕЩЁ ОДИН ПРИМЕР: ДОЛГАЯ КОМПИЛЯЦИЯ

https://bugs.llvm.org/show_bug.cgi?id=33494

```
void foo(int *a) {
    while (a[0]){
        a[0]=1;
        a[73]=(a[73]+1);
        while (a[0]){
            a[0]=1;
            a[73]=(a[73]+1);
            while (a[0]){
                a[0]=(a[0]+a[0]);
                a[1]=(a[1]*(a[73]-a[0]));
            }
            a[0]=1;
        }
        a[0]=1;
    }
}
```

ВЫУЧЕННЫЕ УРОКИ

- Баги есть во всех компиляторах, даже в старых и проверенных временем
- Генерация тестов – хороший способ искать баги
- Семантика теста не имеет значения, если есть референс
- Без измерения покрытия можно обойтись

ПОЛЕЗНЫЕ ССЫЛКИ

- Zing VM: https://www.azul.com/products/zing/virtual-machine/
- LLVM Project: http://llvm.org/
- Falcon: https://www.azul.com/press_release/falcon-jit-compiler/
- Java Fuzzer: https://github.com/AzulSystems/JavaFuzzer

БЛАГОДАРНОСТИ

- Нина Ринская
- Иван Крылов
- Falcon Team
- JUG RU GROUP

СПАСИБО ЗА ВНИМАНИЕ!