`make test-zesti`

# IMPROVING REGRESSION TESTING USING SYMBOLIC EXECUTION

**Paul Marinescu**
Cristian Cadar

Department of Computing
Imperial College London

# Regression testing

# +

# Symbolic execution

```
make test

make test-valgrind

make test-zesti
```

# Outline

- **Motivation**
- Technique
- Evaluation

# Regression Testing

- Strength:

  manually encoded knowledge about the system

- Shortcoming:

  usually considers only the ‘common scenario’

# 100% Coverage

```
int f(int x) {
  int v[100], r;
  if (x > 99) {
    x = 0;
  }
  r = v[x];
  return r;
}
```

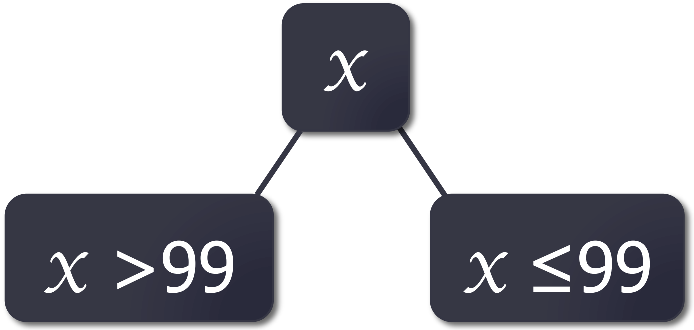
Test 1
x=20

Test 2
x=220

100% coverage ≠ bug-free

# Symbolic Execution

```
int f(int x) {
  int v[100], r;
  if (x > 99) {
    x = 0;
  }
  r = v[x];
  return r;
}
```



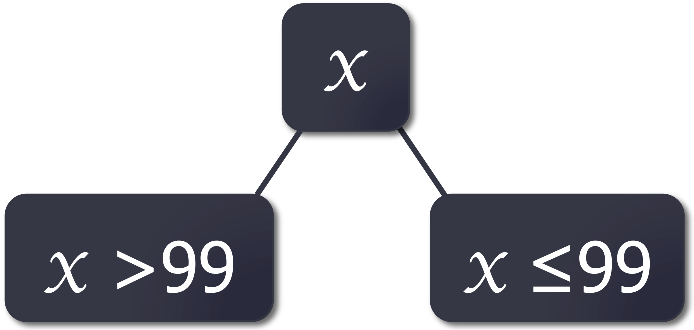
$x$

$x > 99$

$x \le 99$

# Symbolic Execution

```
int f(int x) {
  int v[100], r;
  if (x > 99) {
    x = 0;
  }
  r = v[x];
  return r;
}
```



$x \le 99 \Rightarrow 0 \le x \le 99$

Constraint solver

false, -1

# Symbolic Execution

- Strength:

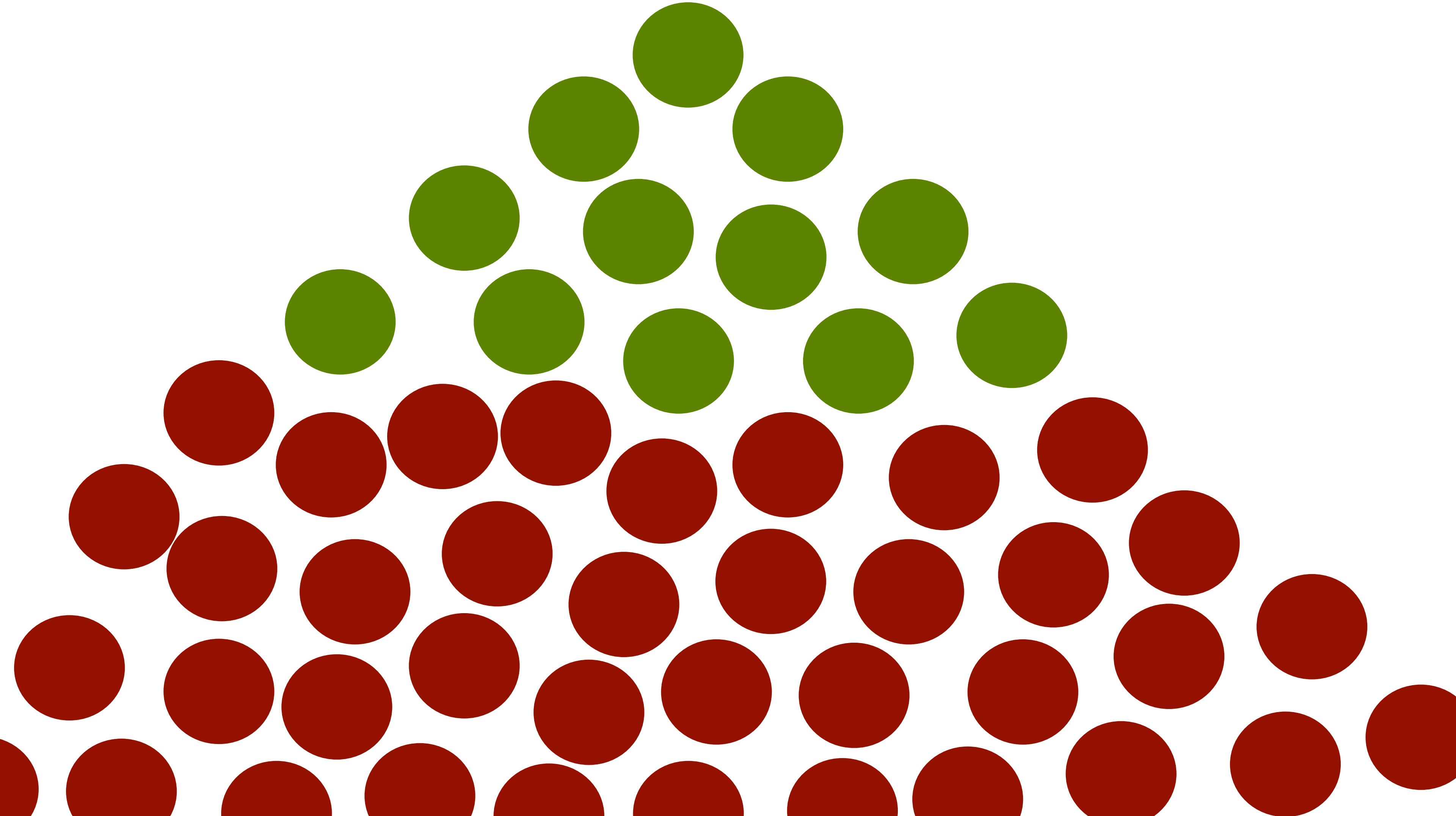
  systematic & exhaustive program exploration

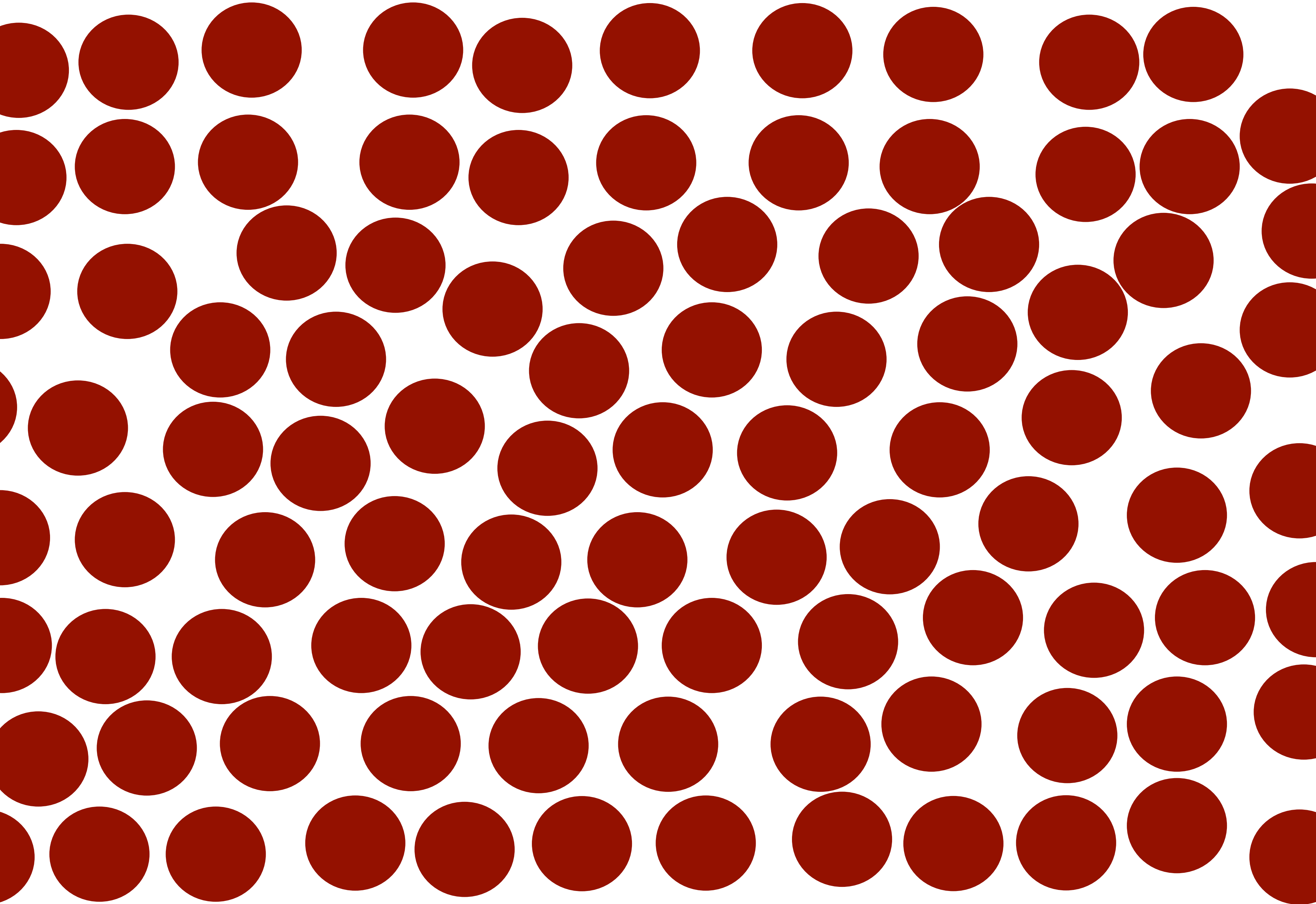
- Shortcomings:

  needs input structure

  path explosion

# Challenge: Path Explosion





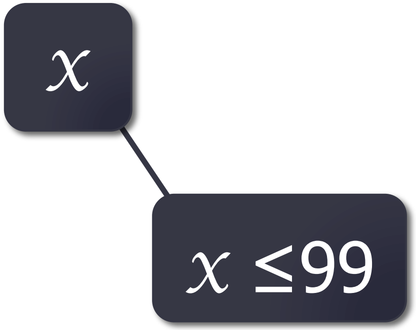
# Outline

- Motivation
- **Technique**
- Evaluation

# Basic Technique

```
int f(int x) {
  int v[100], r;
  if (x > 99) {
    x = 0;
  }
  r = v[x];
  return r;
}
```

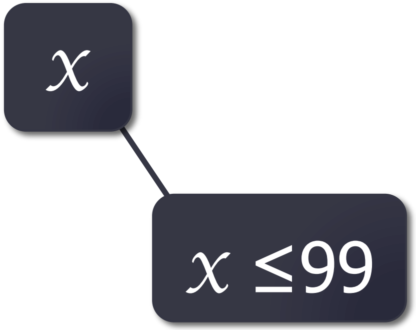
Test 1
x = 20



# Basic Technique

```
int f(int x) {
  int v[100], r;
  if (x > 99) {
    x = 0;
  }
  r = v[x];
  return r;
}
```

Test 1
x = 20



$x \leq 99 \Rightarrow 0 \leq x \leq 99$

Constraint solver

false, -1

# Bug Types Detected

- Invalid memory access
- Division by zero
- Assertion failure

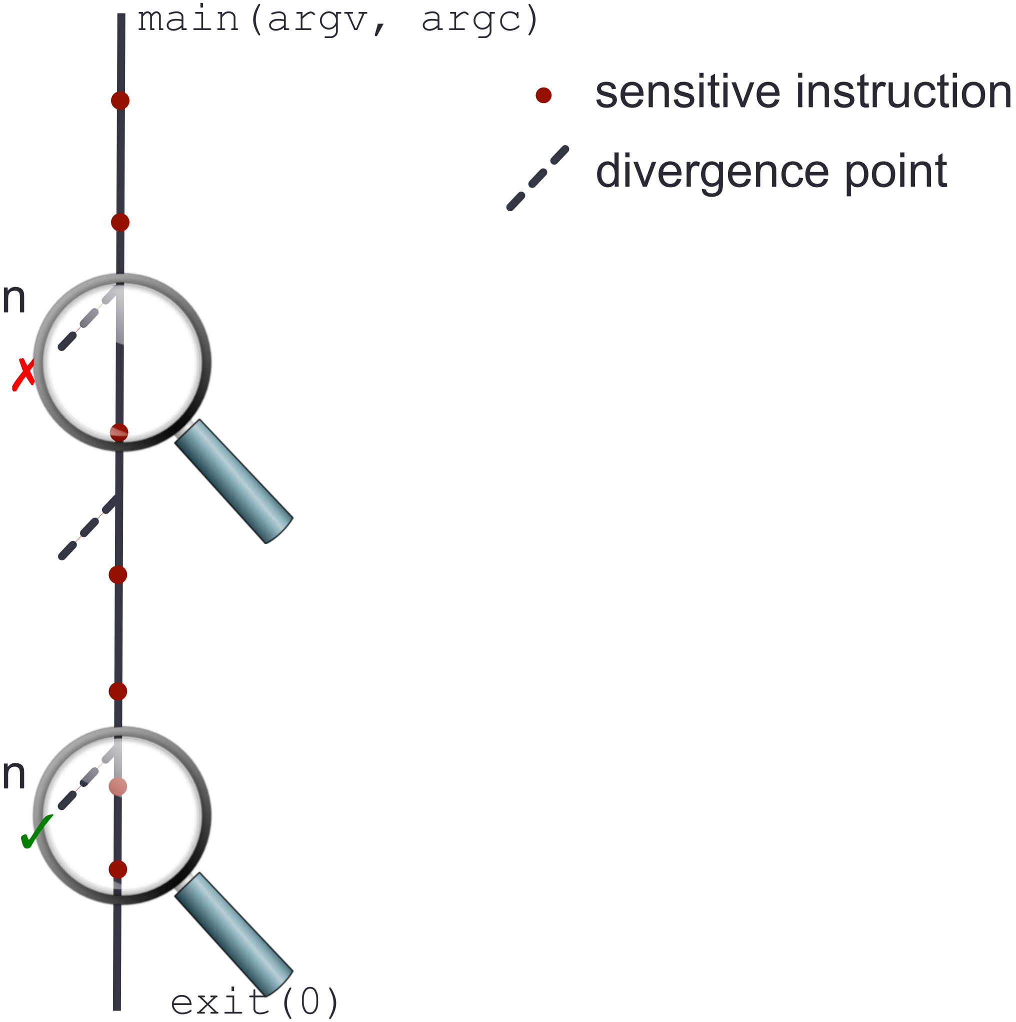
# Basic Technique Limitation

```
int f(int x) {
   int v[100], r;
   if (x > 99) {
      x = 0;
   }
   r = v[x];
   return r;
}
```

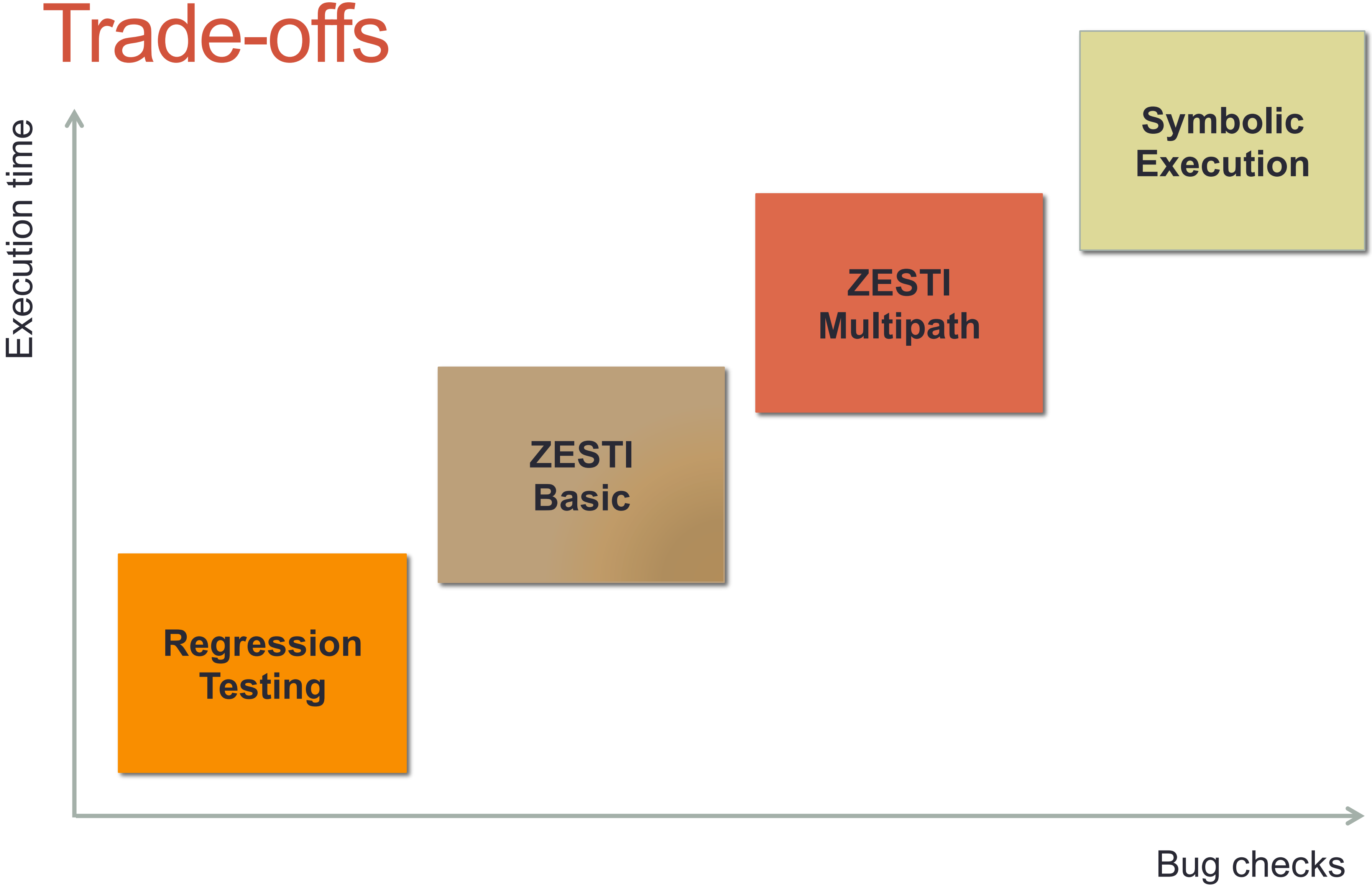
~~Test 1~~
~~x=20~~

Test 2
x=220

# Multipath Analysis



# Trade-offs


Execution time
Symbolic Execution
ZESTI Multipath
ZESTI Basic
Regression Testing
Bug checks

# Outline

- Motivation
- Technique
- **Evaluation**

# Evaluation

- **Coreutils 6.10**
  - <12,293 ELOC
  - basic input processing
  - **8 bugs (2 previously unknown)**

- **libdwarf 2011/06/12**
  - 13,585 ELOC
  - complex input processing
  - **40 bugs previously unknown**

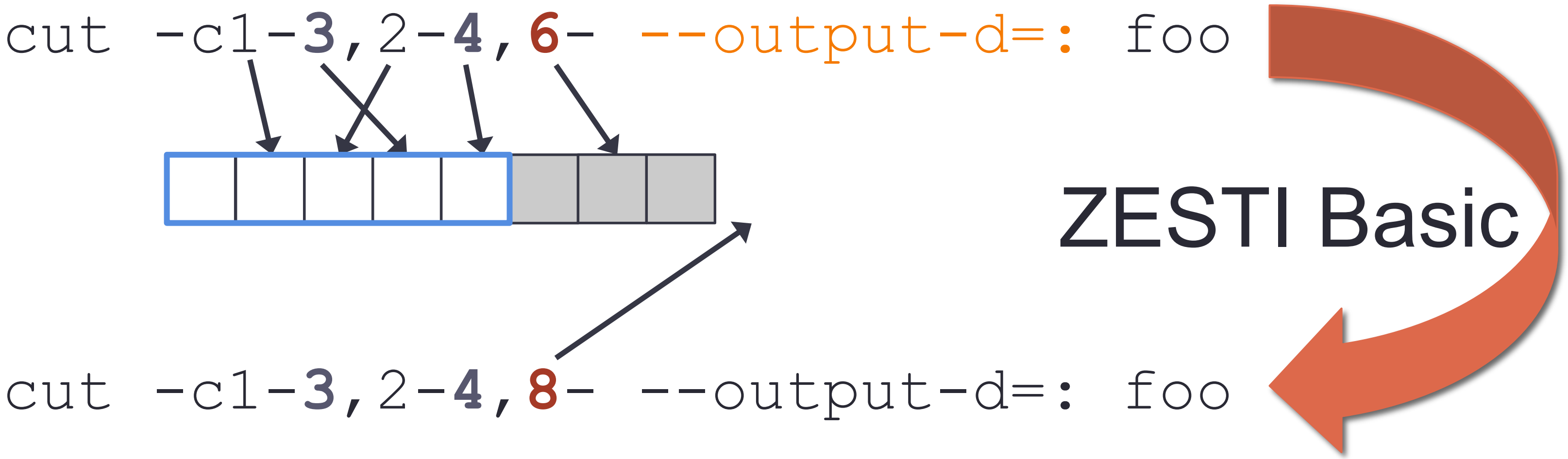
- **readelf 2.21.53**
  - 9,938 ELOC
  - complex input processing
  - **10 bugs previously unknown**

Coreutils

| Bug | ZESTI Basic | ZESTI Multipath | KLEE |
|---|---|---|---|
| cut | ✔ | ✔ | |
| printf | | ✔ | |
| seq | | ✔ | ✔ |
| paste | | ✔ | ✔ |
| mkdir | | ✔ | ✔ |
| mknod | | ✔ | ✔ |
| mkfifo | | ✔ | ✔ |
| md5sum | | ✔ | ✔ |
| tac | ✔* | ✔* | ✔ |
| ptx (1) | | ✔* | ✔ |
| ptx (2) | | | ✔ |
| pr | | | ✔ |
| libdwarf | ✔ | ✔ | blows up |
| readelf | ✔ | ✔ | blows up |

*with additional regression test

# ZESTI Case Study – cut Bug



```
cut -c1-3,2-4,6- --output-d=: foo
```

ZESTI Basic

```
cut -c1-3,2-4,8- --output-d=: foo
```

```
printable_field = malloc(max_range_endpoint/CHAR_BIT+1);
...
if (output_delimiter_specified
      && eol_range_start
      && !check (printable_field [eol_range_start]))
{ ... }
```

# ZESTI Case Study – printf Bug

```
printf %d 0
```

ZESTI Multipath

```
printf %d '
```

```
if (*s == '\"' || *s == '\'')
{
    unsigned char ch = *++s
    val = ch;
    if (*++s != 0)
        error (0, 0, _(cfcc_msg), s);
}
```

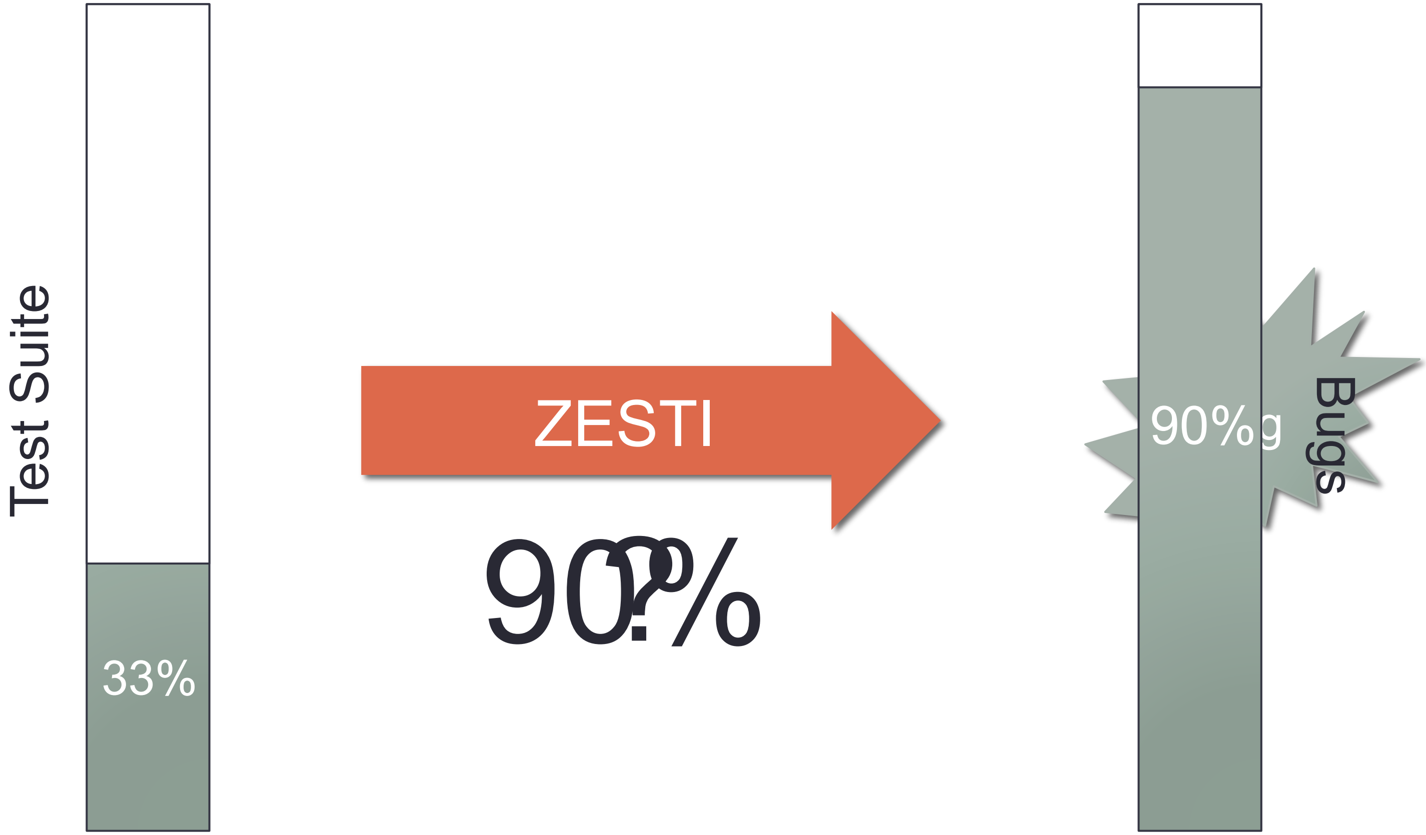
# dwarfdump Bug

ZESTI

| Offset | Original File | Bug-Revealing File |
|---|---|---|
| 0000 | 7F 45 4C 46 | 7F 45 4C 46 |
| ... | | |
| 1070 | 00 00 00 **04** | 00 00 00 **00** |
| ... | | |
| 2024 | 69 74 00 | 69 74 00 |

```
entry_size = 2*address_size+segment_size
if (section_size % entry_size != 0)
  // report error
```

# Test Suite Quality Impact



⅓ randomly picked tests typically give 90% probability of finding the bugs in Coreutils

# Related Work

- Systematic Dynamic Test Generation
  - CUTE, SAGE, PEX, MACE, and more
- Test Suite Augmentation
  - Augmenting the test suite as the system evolves

# Zero-Effort Symbolic Test Improvement

- ZESTI transparently improves regression testing

  ⇒ `make test-zesti`

- ZESTI found over 50 new bugs in 3 mature systems

http://srg.doc.ic.ac.uk/projects/zesti