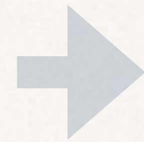
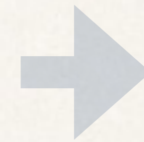
Test Generator



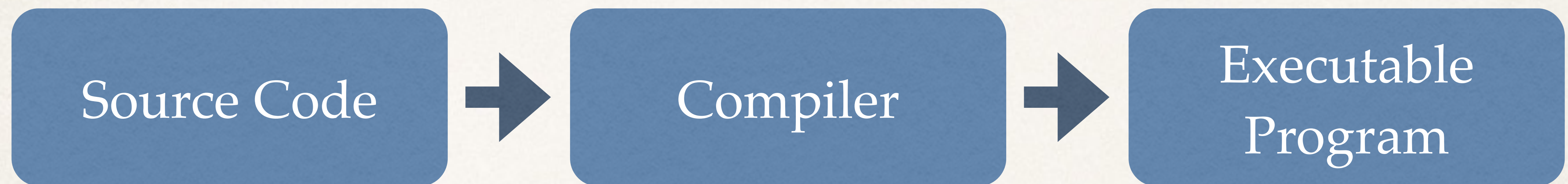
```
int

}
```



Compiler

Search-Based System Testing

Andreas Zeller • Saarland University

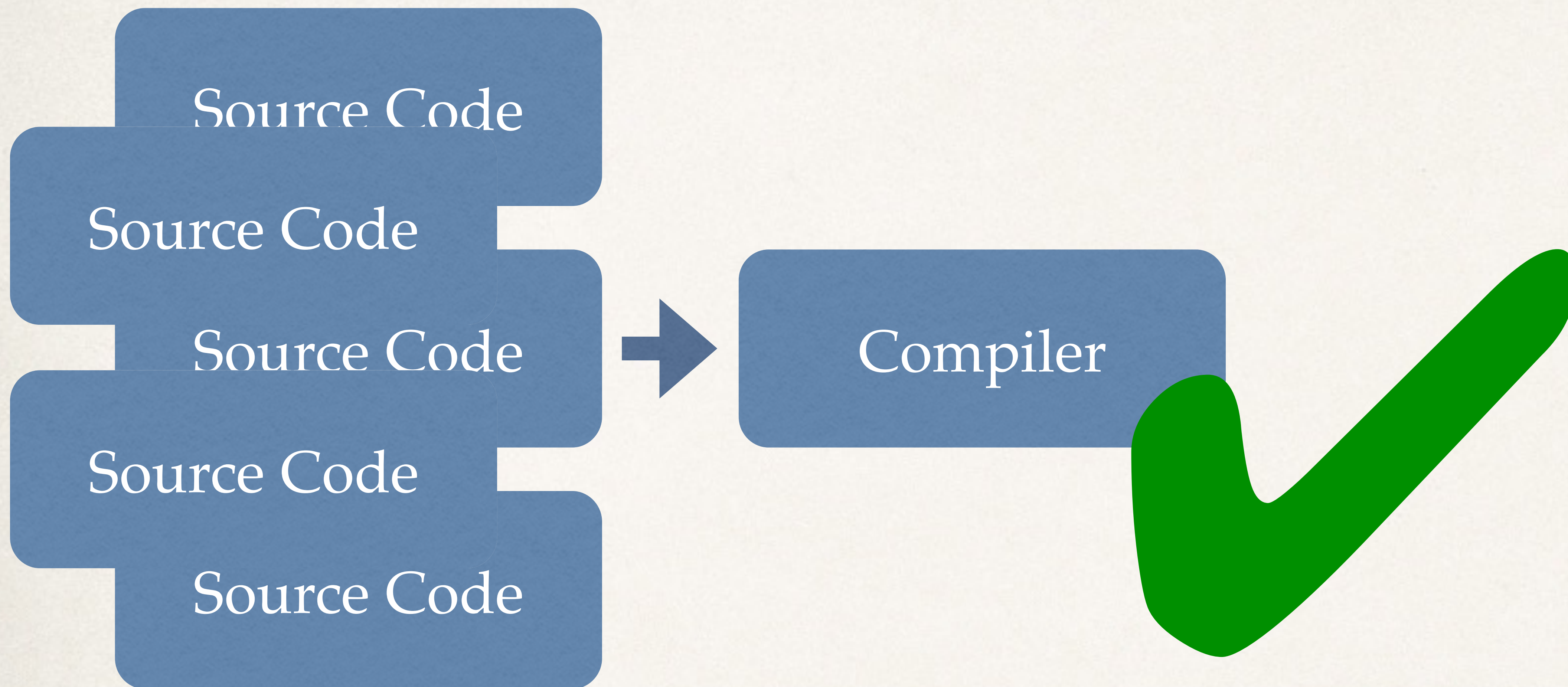


Compiler

Every source must be compiled

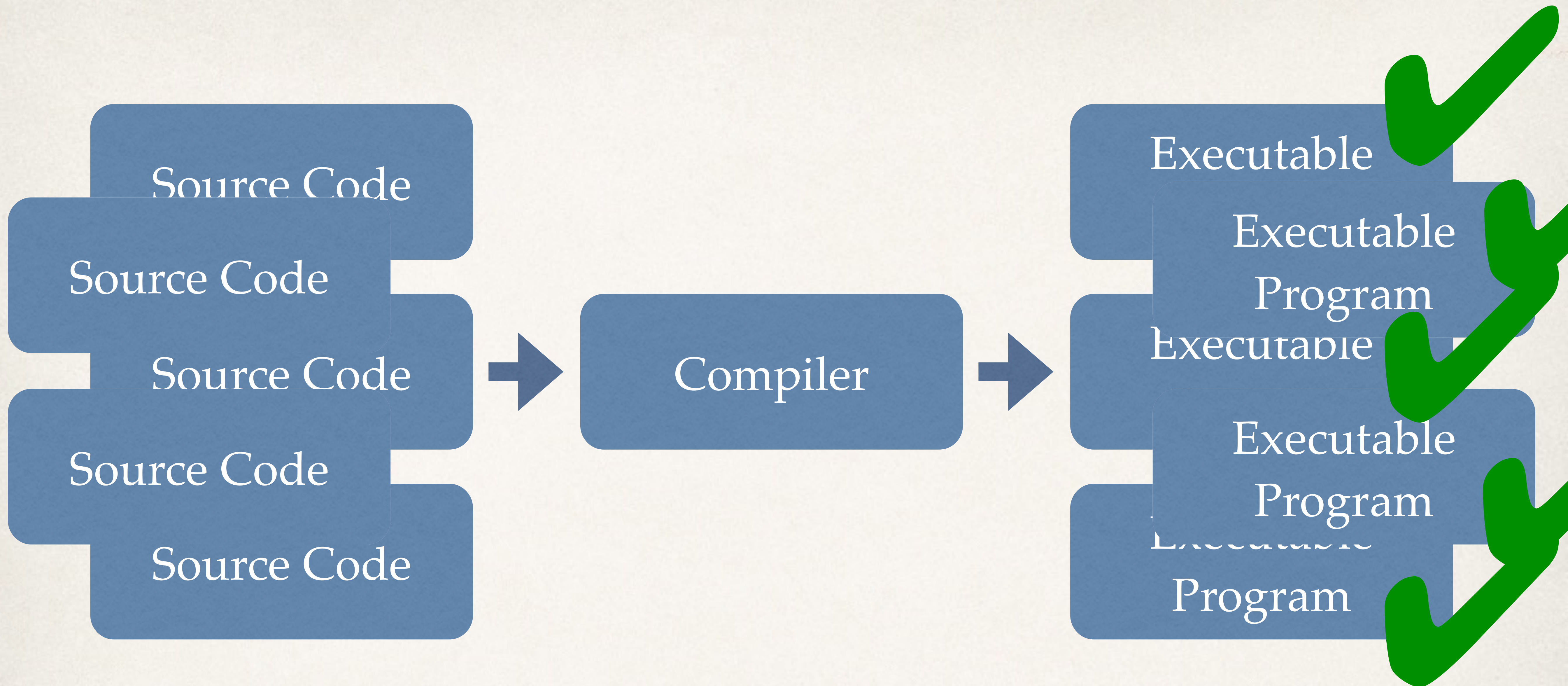
Quality Assurance

How do we know the compiler works?



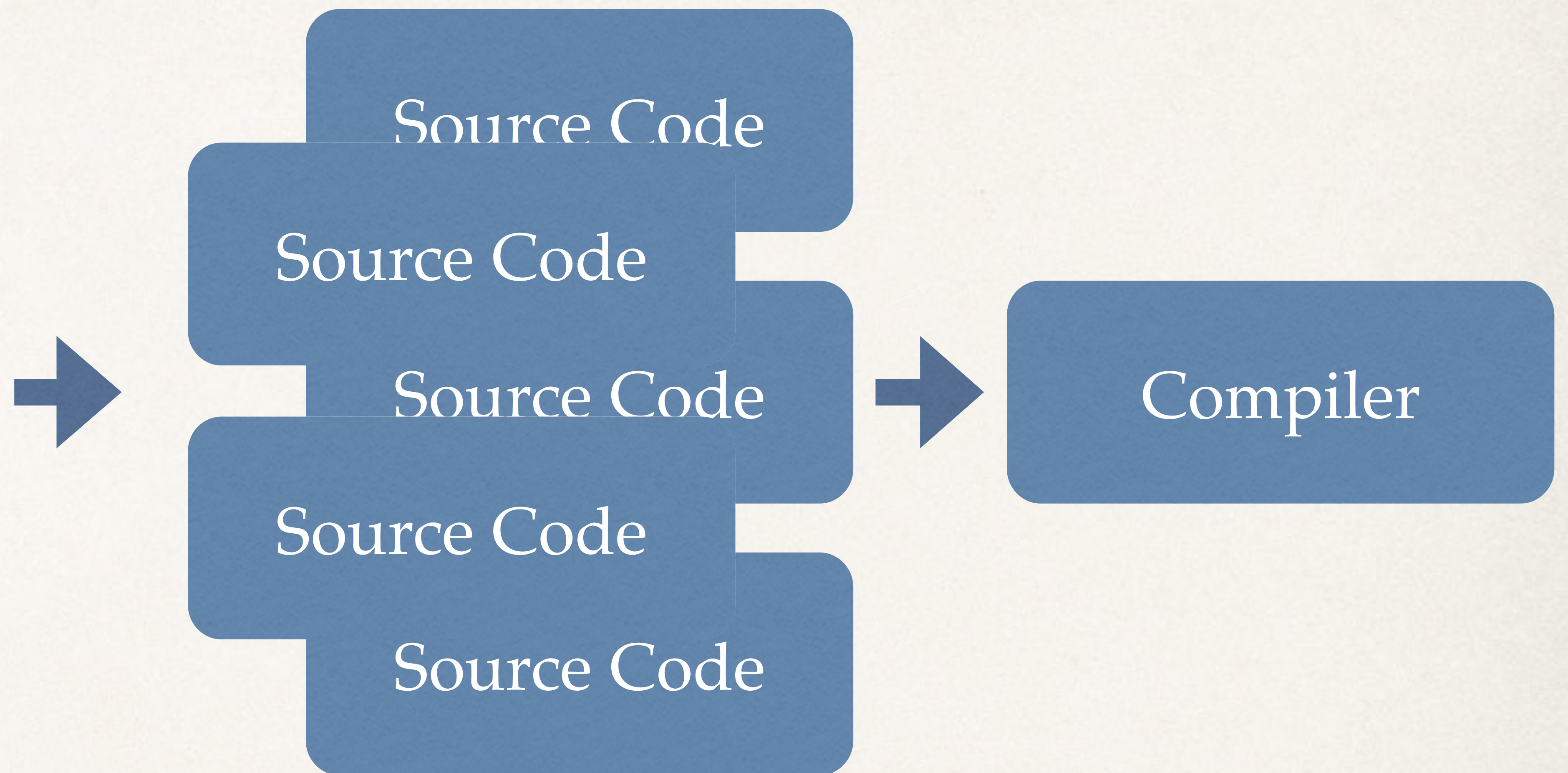
Verification

A 100% correct compiler – under specific circumstances



Testing

Assess the compiler across all stages and customizations



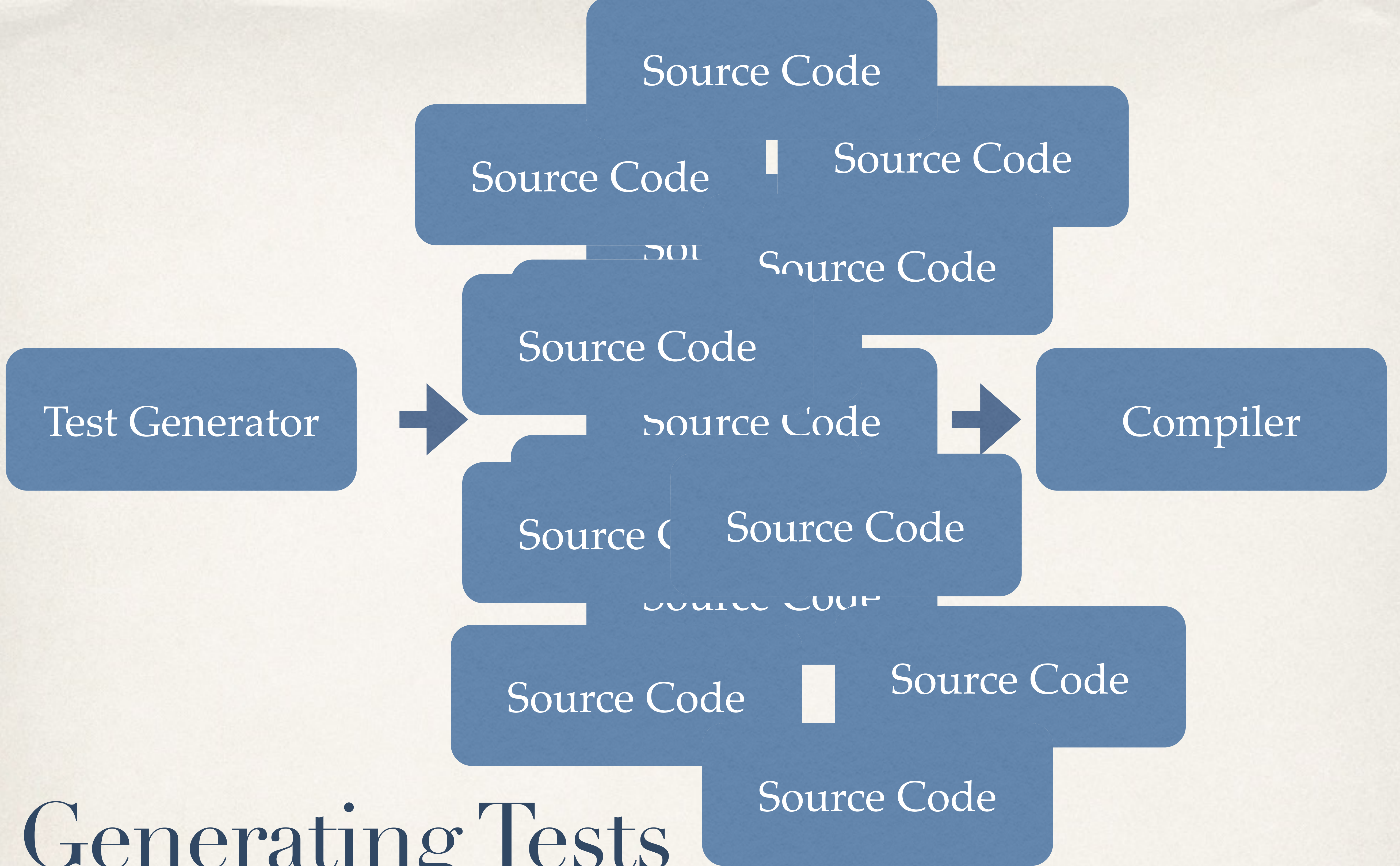
Testing

But who should write all these tests?

Test Generator

Generating Tests

A *Test Generator* automatically generates C inputs



Generating Tests

A *Test Generator* automatically generates C inputs

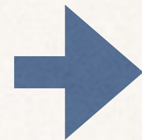
```
%token IDENTIFIER CONSTANT STRING_LITERAL SIZEOF
%token PTR_OP INC_OP DEC_OP LEFT_OP RIGHT_OP LE_OP GE_OP EQ_OP NE_OP
%token AND_OP OR_OP MUL_ASSIGN DIV_ASSIGN MOD_ASSIGN ADD_ASSIGN
%token SUB_ASSIGN LEFT_ASSIGN RIGHT_ASSIGN AND_ASSIGN
%token XOR_ASSIGN OR_ASSIGN TYPE_NAME

%token TYPEDEF EXTERN STATIC AUTO REGISTER
%token CHAR SHORT INT LONG SIGNED UNSIGNED FLOAT DOUBLE CONST VOLATILE VOID
%token STRUCT UNION ENUM ELLIPSIS

%token CASE DEFAULT IF ELSE SWITCH WHILE DO FOR GOTO CONTINUE BREAK RETURN

%start translation_unit
%%

primary_expression
	: IDENTIFIER
	| CONSTANT
	| STRING_LITERAL
	| '(' expression ')'
	;

postfix_expression
	: primary_expression
	| postfix_expression '[' expression ']'
	| postfix_expression '(' ')'
	| postfix_expression '(' argument_expression_list ')'
	| postfix_expression '.' IDENTIFIER
	| postfix_expression PTR_OP IDENTIFIER
	| postfix_expression INC_OP
	| postfix_expression DEC_OP
	;
```

Test Generator

Generating Tests

Use a grammar as base

Test Generator



```
int foo (void) {
    signed char x = 1;
    unsigned char y = 255;
    return x > y;
}
```

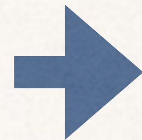
correct result is 0

returns 1 with GCC on Ubuntu Linux 8.04

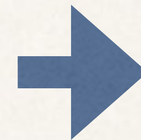
CSmith

The *CSmith* Test Generator found 325 new errors in C compilers

Test Generator



```
int foo (void) {
    signed char x = 1;
    unsigned char y = 255;
    return x > y;
}
```

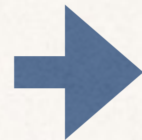


Compiler

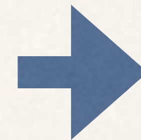
CSmith

Such inputs would be generated by the thousands

Test Generator



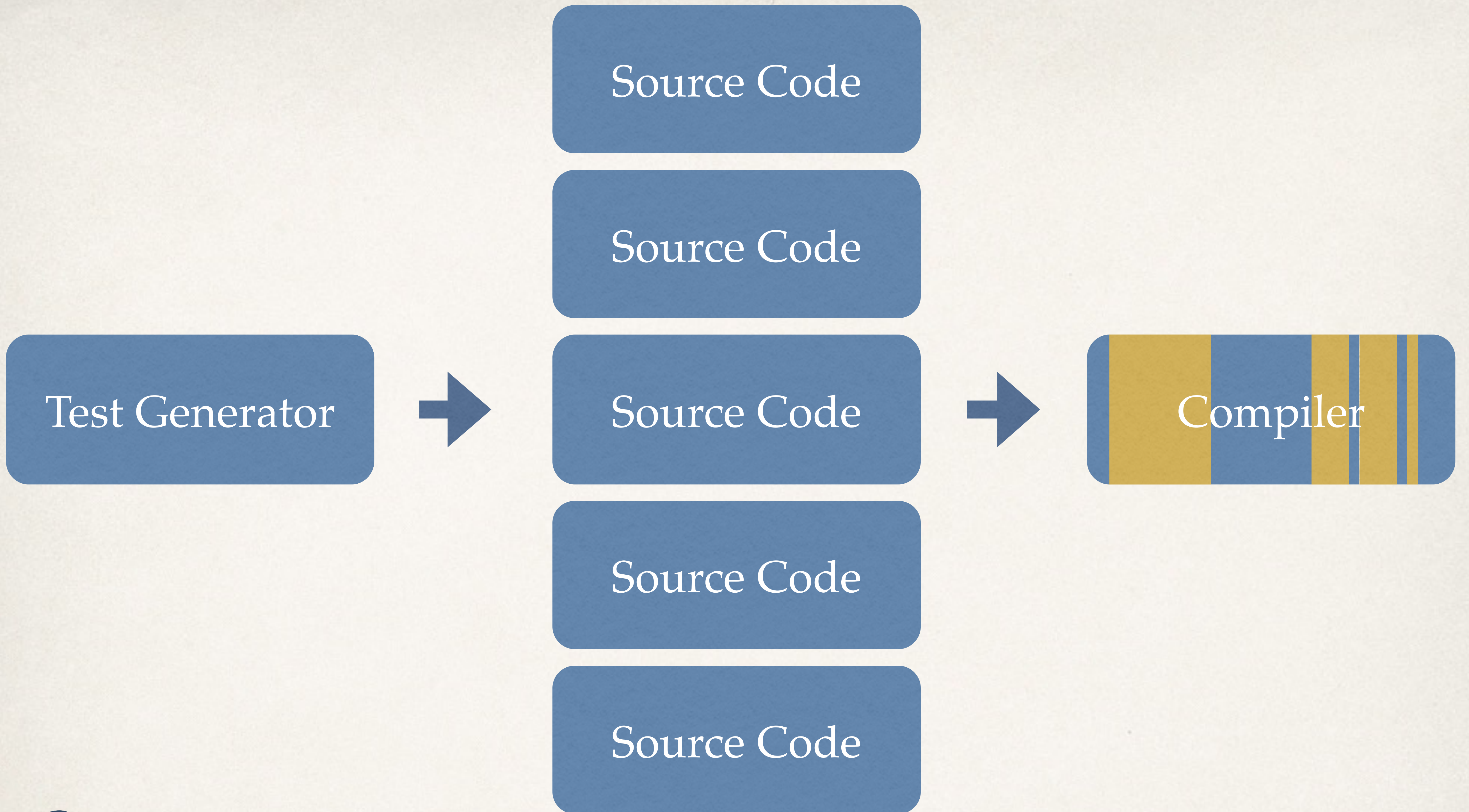
```c
int foo (void) {
    signed char x = 1;
    unsigned char y = 255;
    return x > y;
}
```



Compiler

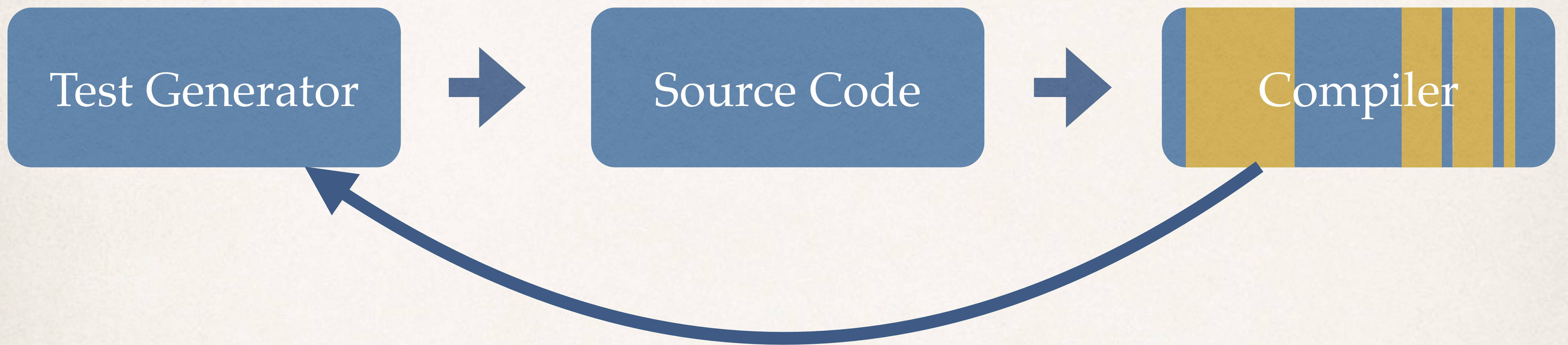
Coverage

We can measure which code has been *covered*



Coverage

But random inputs will leave code *uncovered*

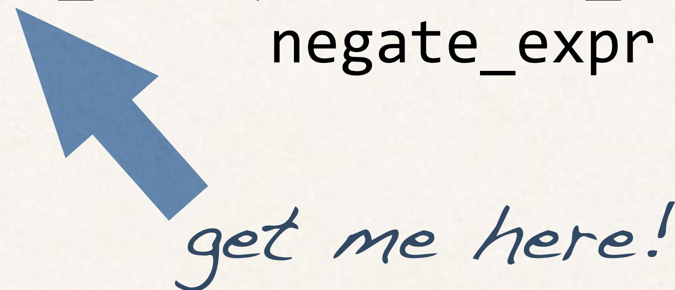


Feedback

Key idea: Generate *targeted inputs* to cover functionality

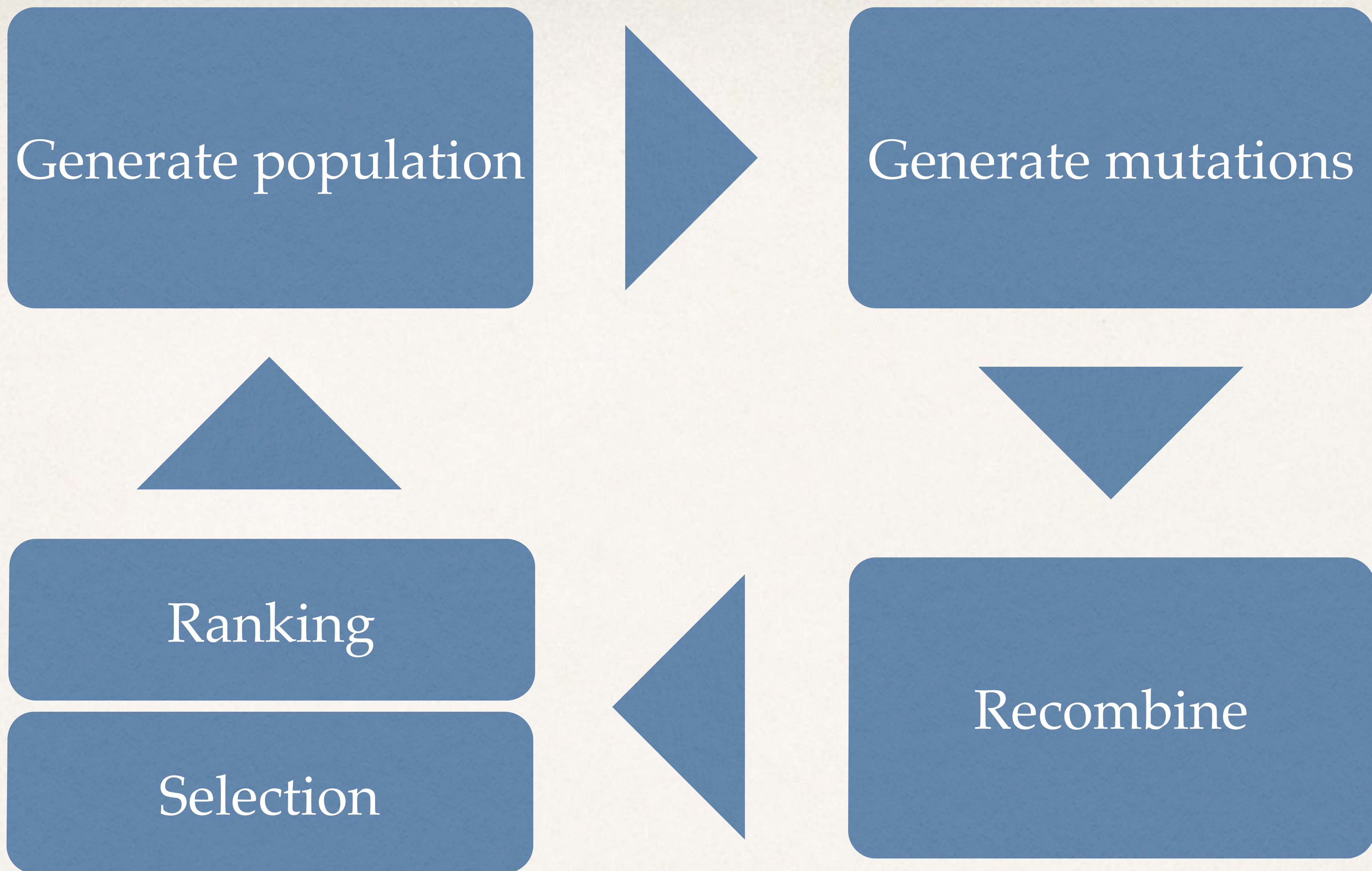
GCC Source Code

```
static tree
fold_negate_expr (location_t loc, tree t)
{
    // 200 Lines...
    tem = TREE_OPERAND (t, 0);
    if ((INTEGRAL_TYPE_P (type)
          && (TREE_CODE (tem) == NEGATE_EXPR
              || (TREE_CODE (tem) == INTEGER_CST
                  && may_negate_without_overflow_p (tem))))
         || !INTEGRAL_TYPE_P (type))
         return fold_build2_loc (loc, TREE_CODE (t), type,
                                 negate_expr (tem), TREE_OPERAND (t, 1));
}
```



Path Conditions

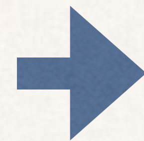
Traditional test generators solve conditions to reach uncovered code

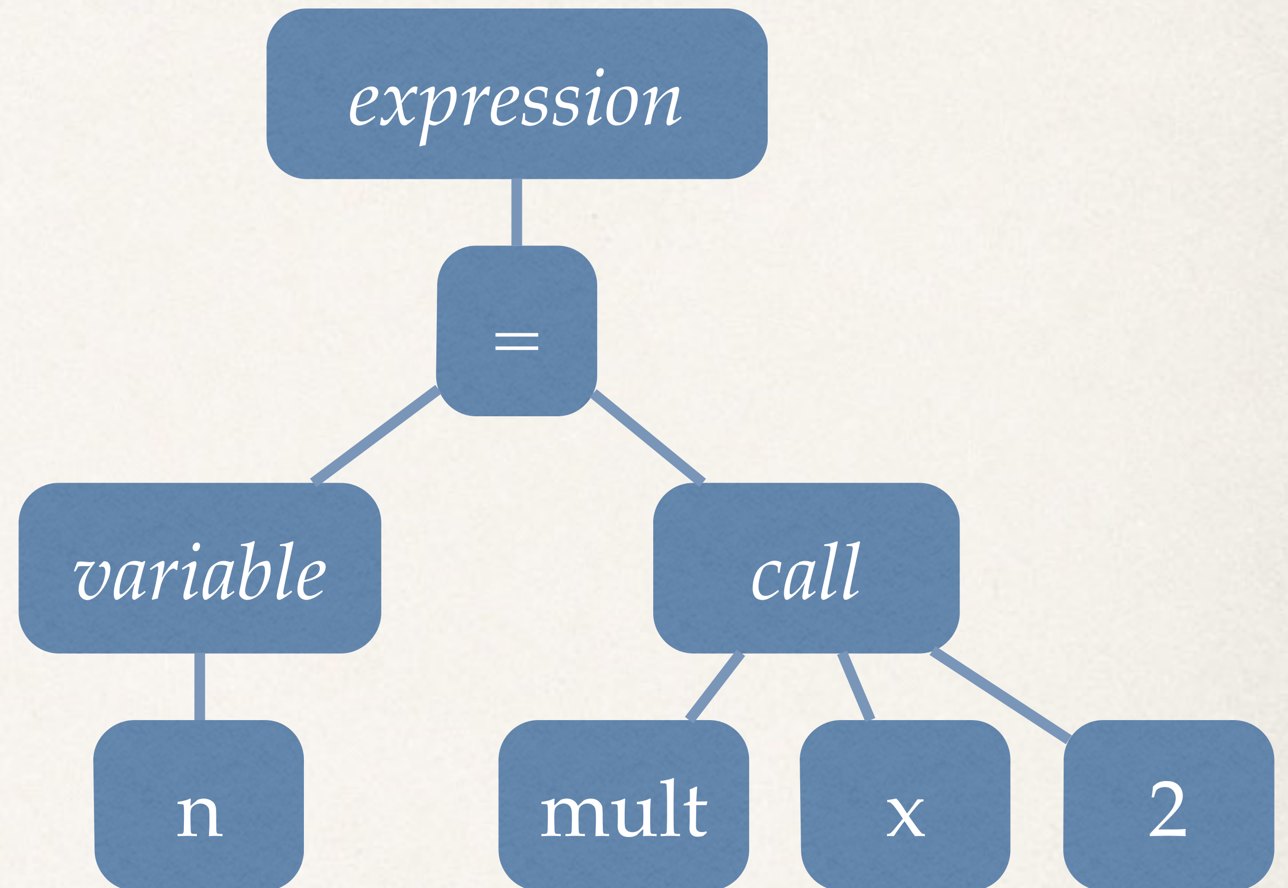


Evolutionary Algorithms

Select inputs according to given goal

```
n = mult(x, 2);
```

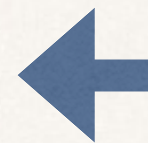


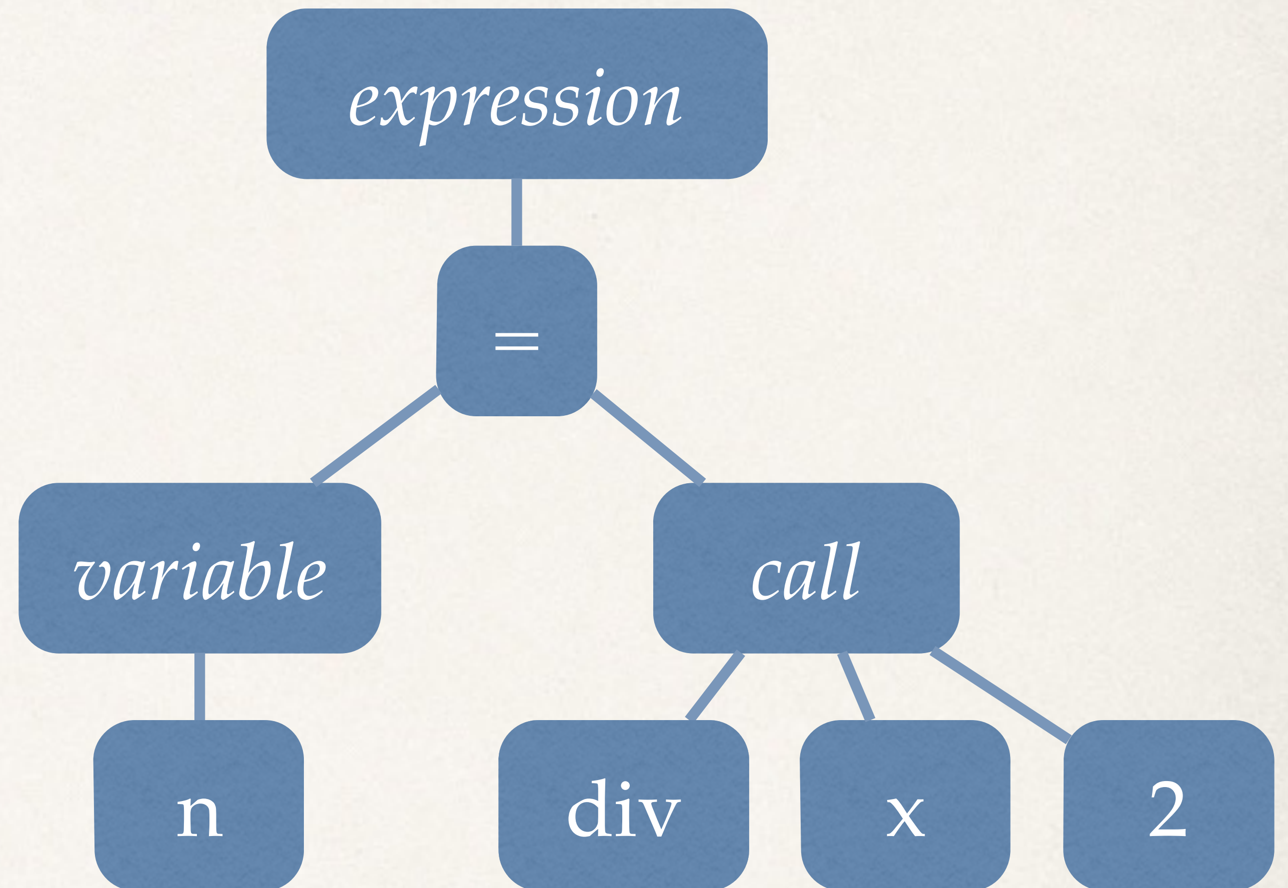


A C Program

We parse a given C program into an abstract syntax tree

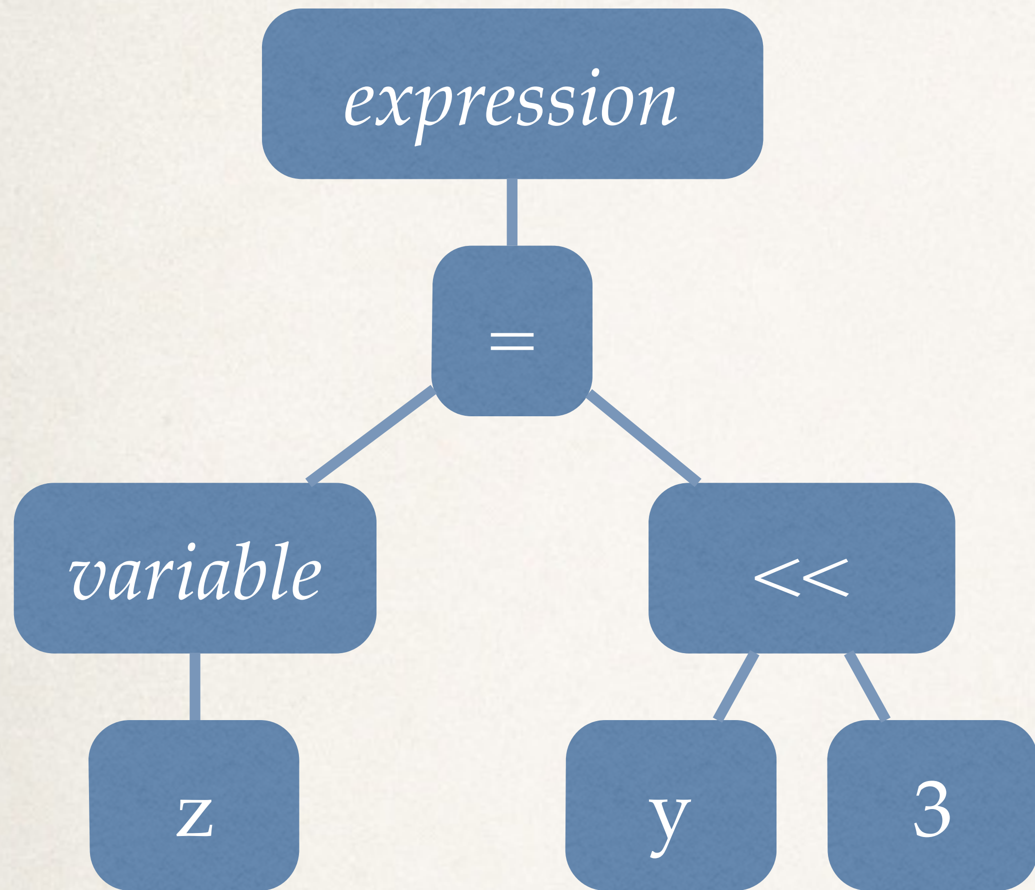
```
n = div(x, 2);
```

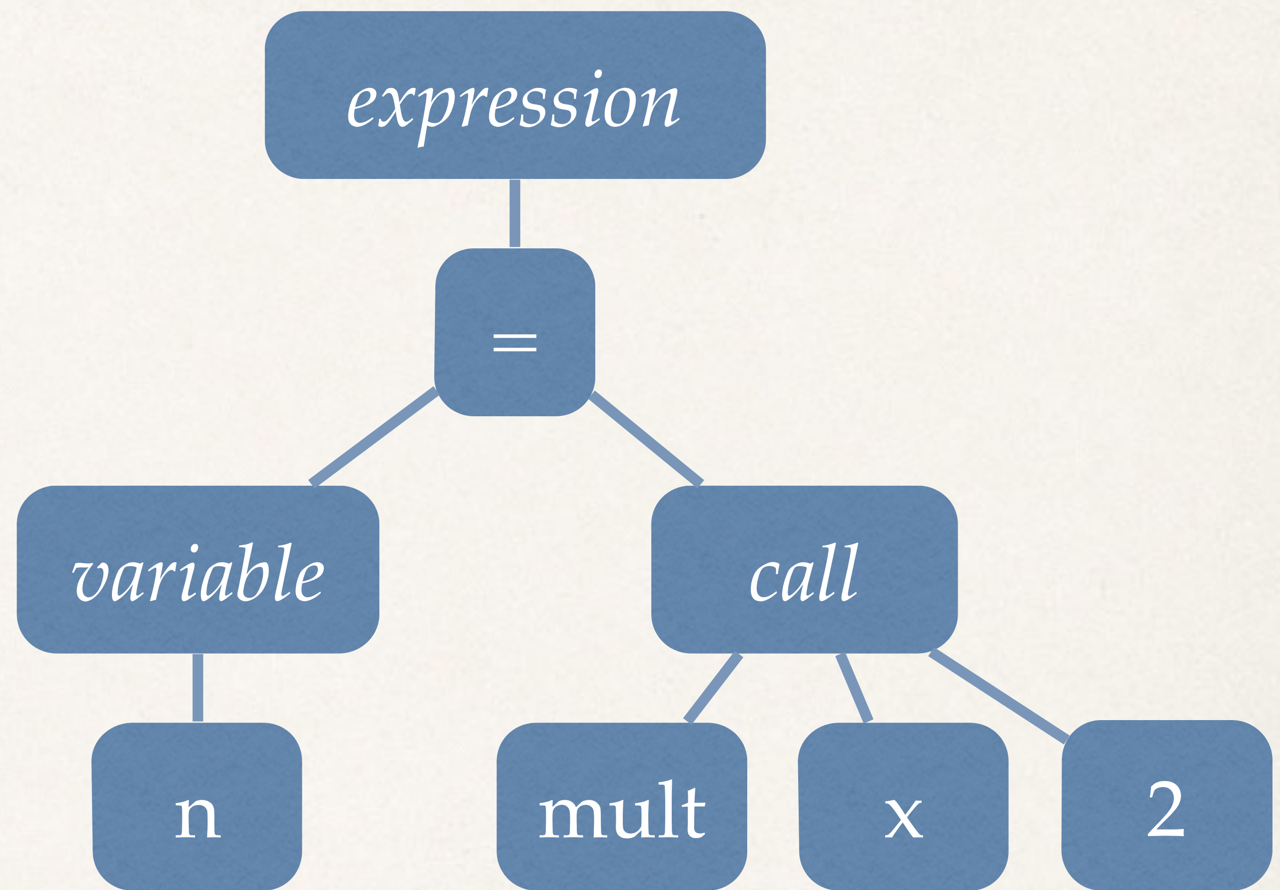




Mutation

For mutation, we replace parts of the abstract syntax tree

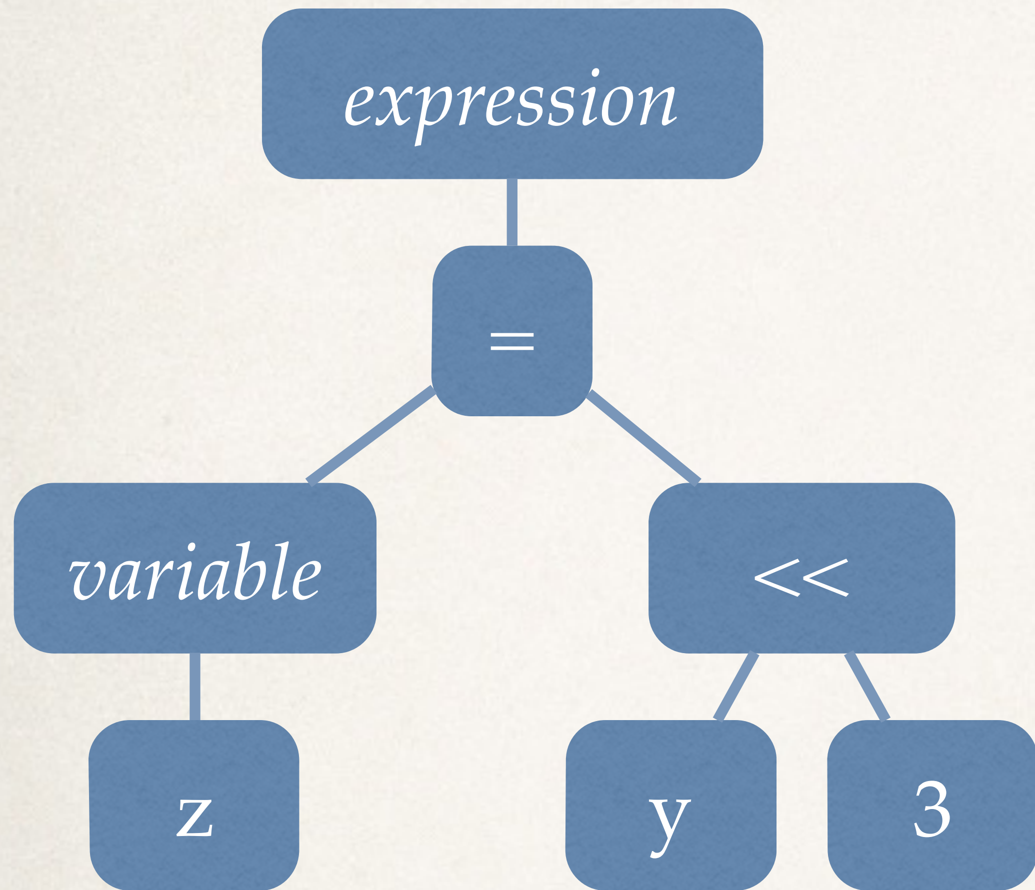


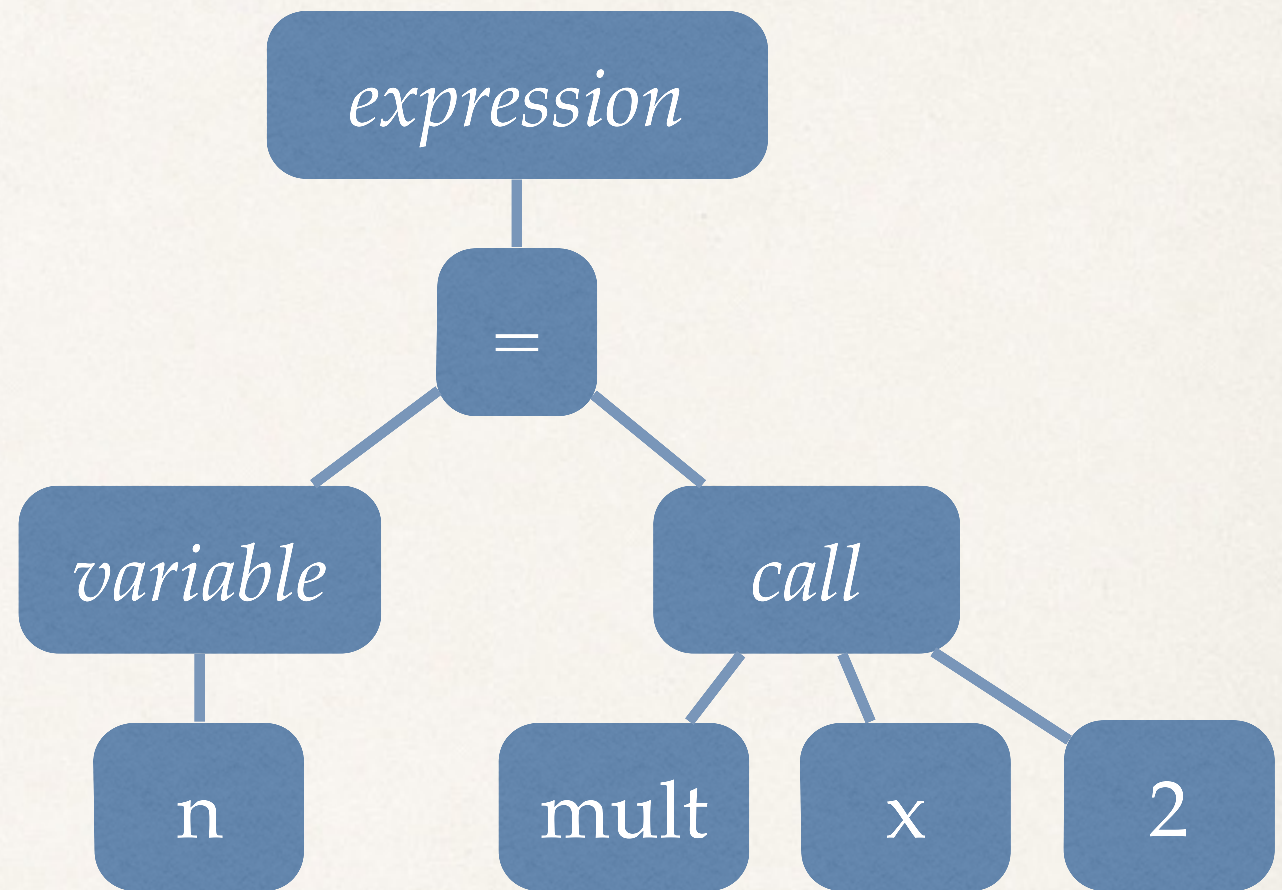


Recombination

We generate new programs by recombining fragments

```
z = y << 3;                n = mult(x, 2);
```

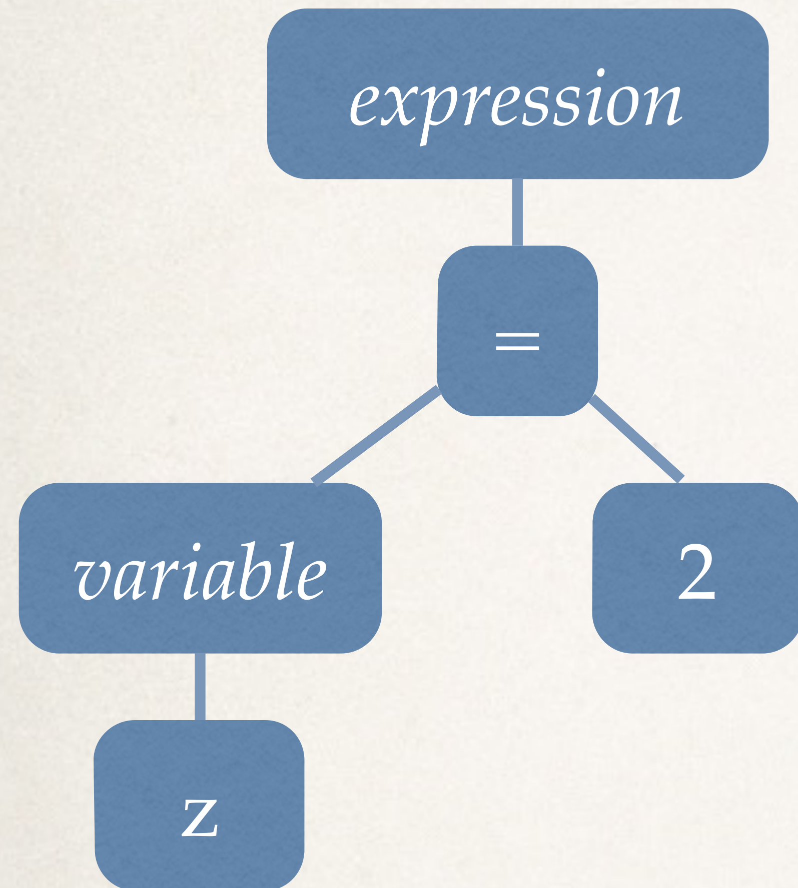




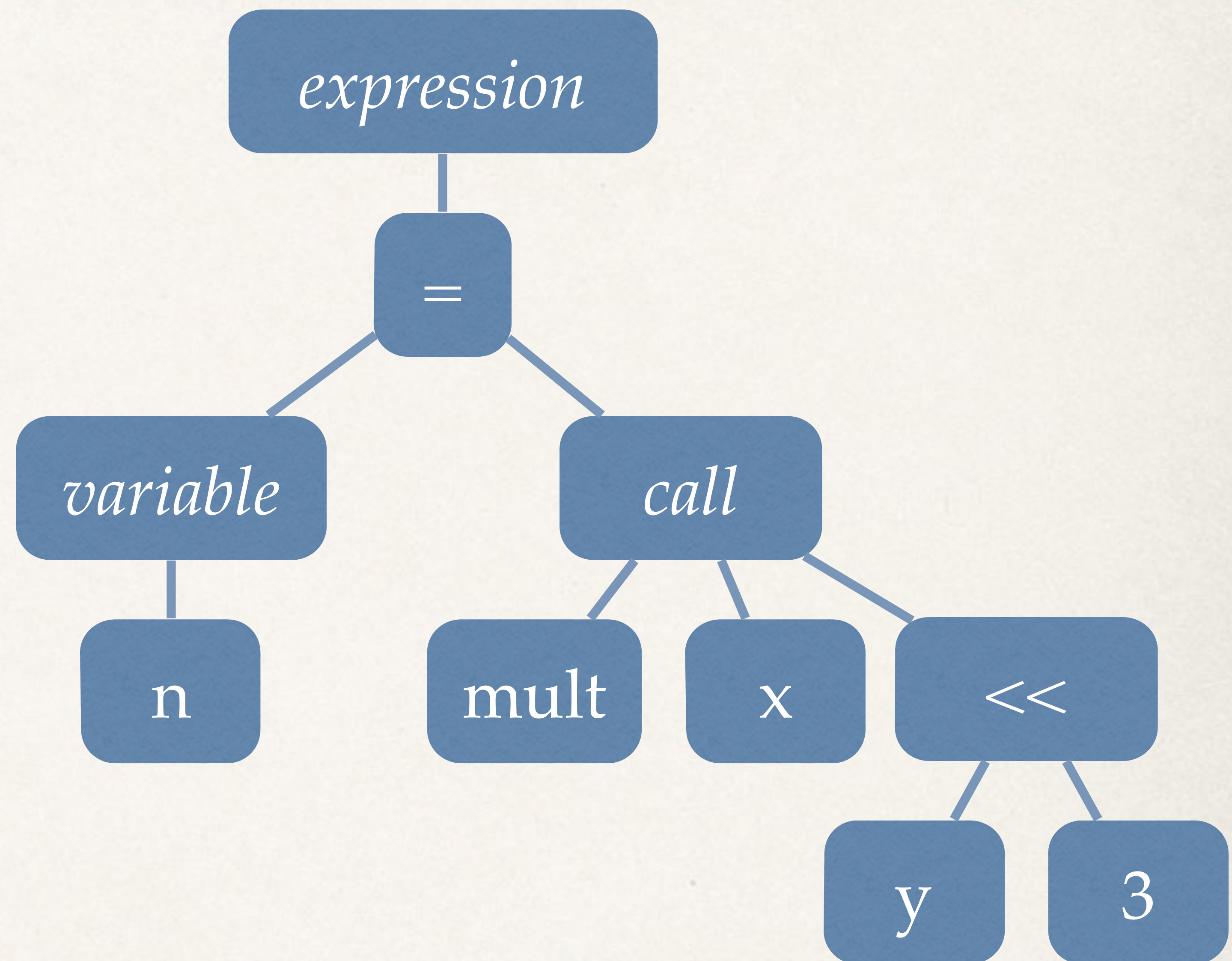
Recombination

We generate new programs by recombining fragments

```
z = 2;
```

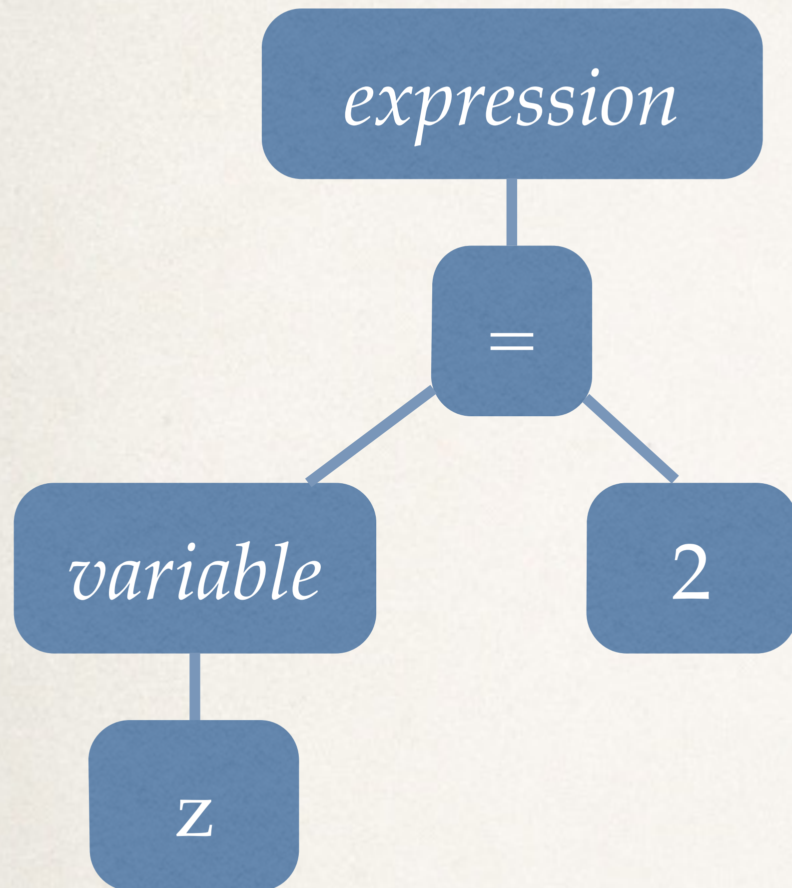


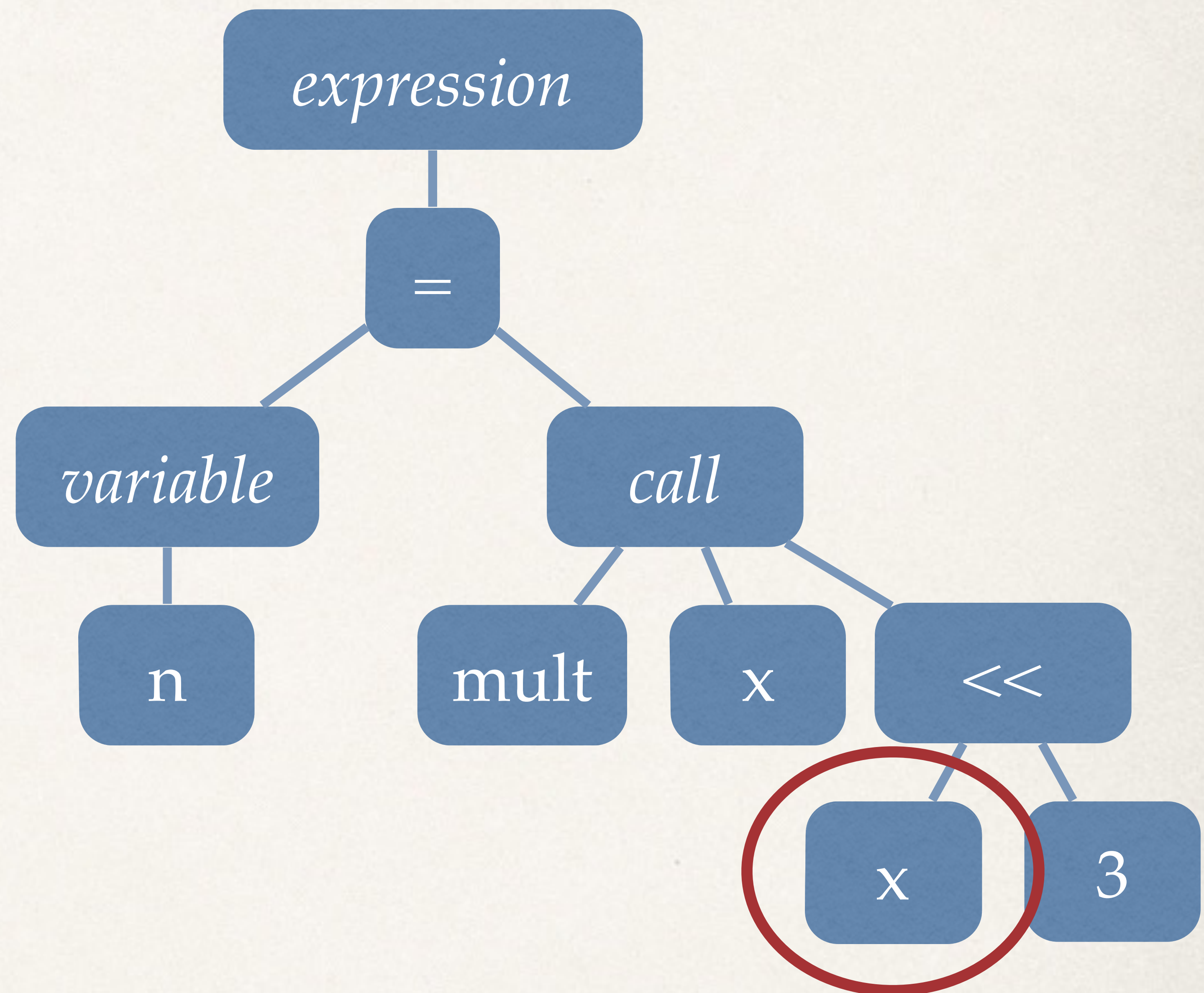

```
n = mult(x, y << 3);
```



Recombination

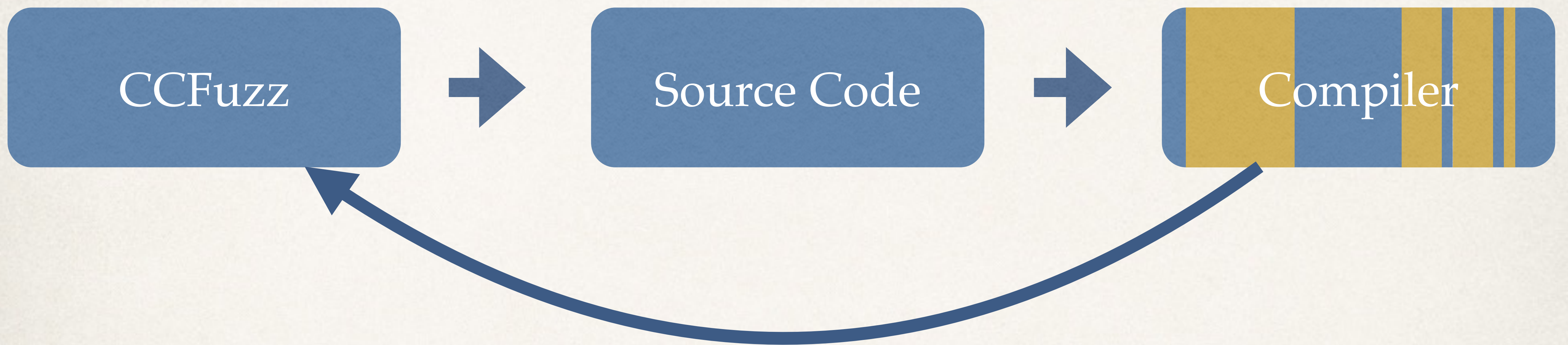
We generate new programs by recombining fragments





Recombination

Mutation and recombination must preserve language semantics



CCFuzz

An automatic, evolutionary test generator for C compilers

```c
int main() {
    double var0 = -474411016.5983948;
    double var1 = var0;
    var1 /= (var0 + 842286918.0743376 * var0) + 1757571303008385195L;
    unsigned long var2 = 7608230537870159477UL;
    var2 *= var2 * var2;
    int var3 = -1522999176;
    for (; var3 <= -1522999137; ++var3) {
            var1 = var1 / -3315559715349753910LL /
                 (var3 & 7709094732231684709ULL) == -1854837480 / 2075989423U;
            unsigned long long var4 = 17274394483946351480ULL;
            int var5 = 1690262972;
            var5 += (4 / var4 % var2 & 8151340956687106979ULL) >= 36423 / -24124 % 120;
    }
    int var4 = 1979470929;
    var4 |= var3 - -4708;
    signed char var5 = 22;
    unsigned char var6 = 7;
    var5 = var5 + var6;
    // Output variables ...
}
```

C Input generated by *CCFuzz*

Current focus: data type conversions

```
static void fast_mix(struct fast_pool *f, __u32 input[4])
{
    __u32       w;
    unsigned input_rotate = f->rotate;

    w = rol32(input[0], input_rotate) ^ f->pool[0] ^ f->pool[3];
    f->pool[0] = (w >> 3) ^ twist_table[w & 7];
    input_rotate = (input_rotate + 14) & 31;
    w = rol32(input[1], input_rotate) ^ f->pool[1] ^ f->pool[0];
    f->pool[1] = (w >> 3) ^ twist_table[w & 7];
    input_rotate = (input_rotate + 7) & 31;
    w = rol32(input[2], input_rotate) ^ f->pool[2] ^ f->pool[1];
    f->pool[2] = (w >> 3) ^ twist_table[w & 7];
    input_rotate = (input_rotate + 7) & 31;
    w = rol32(input[3], input_rotate) ^ f->pool[3] ^ f->pool[2];
    f->pool[3] = (w >> 3) ^ twist_table[w & 7];
    input_rotate = (input_rotate + 7) & 31;

    f->rotate = input_rotate;
    f->count++;
}
```

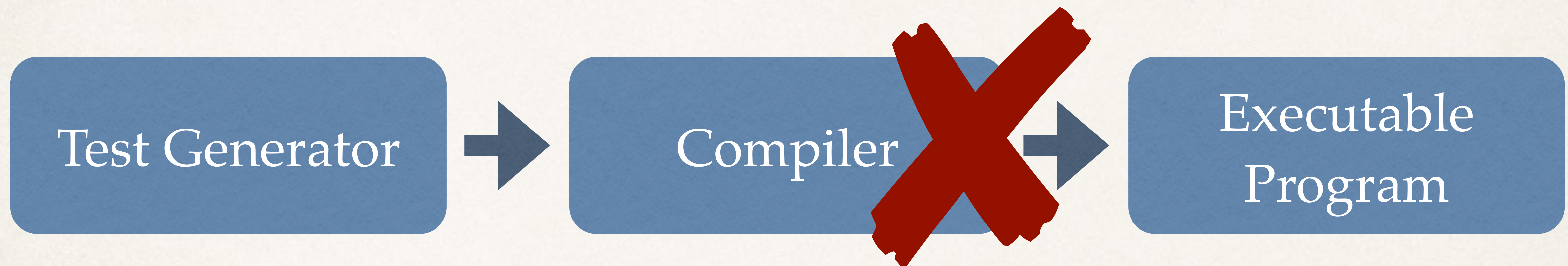
Mutation and Recombination

CCFuzz uses existing code to reuse its focus

```
static void fast_mix(struct fast_pool *f, __u32 input[4])
{
	__u32		w;
	unsigned input_rotate = f->rotate;

	w = rol32(input[0], input_rotate) ^ f->pool[0] ^ f->pool[3];
	f->pool[0] = (w >> 3) ^ twist_table[w << (ENTROPY_SHIFT + 3)];
	input_rotate = (input_rotate + 14) & 31;
	w = rol32(input[1], input_rotate) ^ f->pool[1] ^ f->pool[0];
	f->pool[1] = (w >> 3) ^ twist_table[w << (ENTROPY_SHIFT + 3)];
	input_rotate = (input_rotate + 7) & 31;
	w = rol32(input[2], input_rotate) ^ f->pool[2] ^ f->pool[1];
	f->pool[2] = (w >> 3) ^ twist_table[w << (ENTROPY_SHIFT + 3)];
	input_rotate = (input_rotate + 7) & 31;
	w = rol32(input[3], input_rotate) ^ f->pool[3] ^ f->pool[2];
	f->pool[3] = (w >> 3) ^ twist_table[w << (ENTROPY_SHIFT + 3)];
	input_rotate = (input_rotate + 7) & 31;

	f->rotate = input_rotate;
	f->count++;
}
```

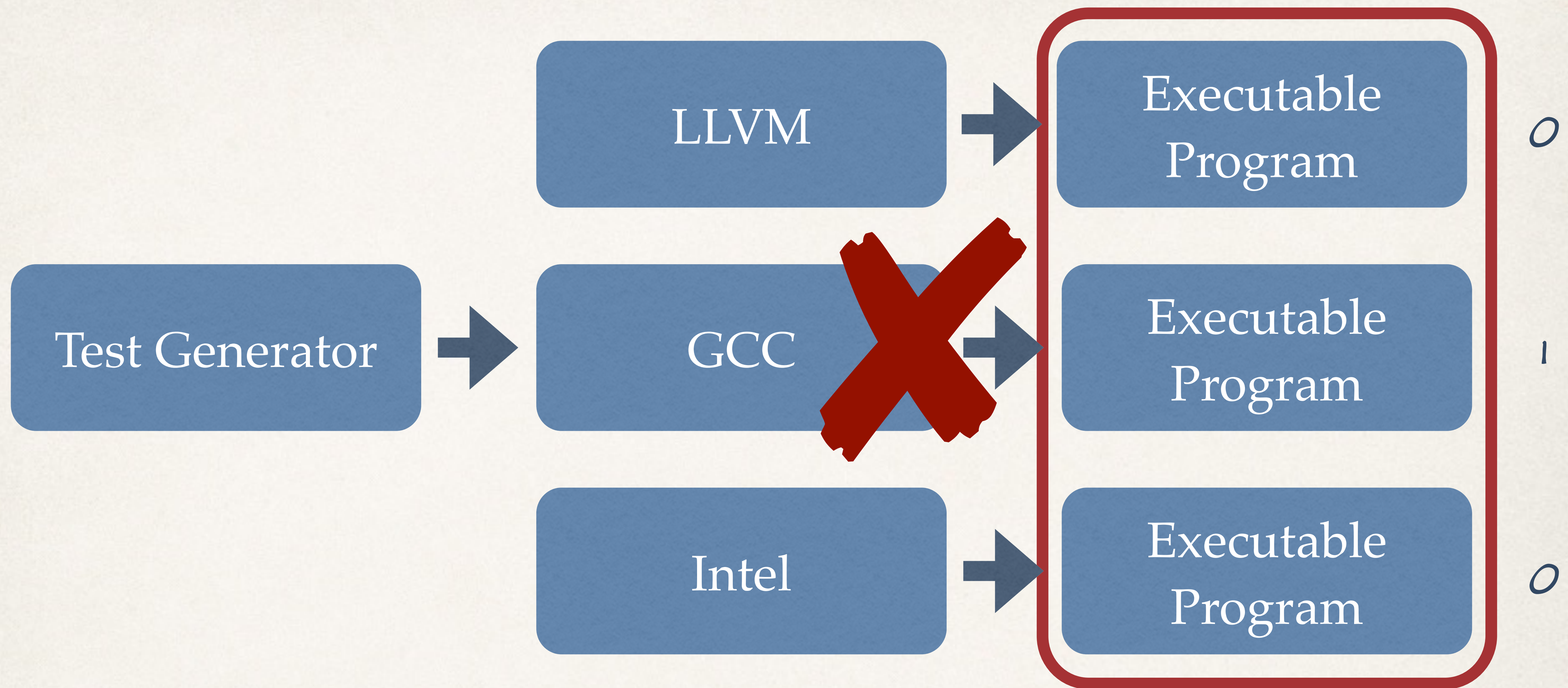
Mutation and Recombination

New code fragments are taken from other parts



Oracle

How do we know whether the result is correct? For one, the compiler might crash…



Oracle

We can also compare the resulting programs against each other

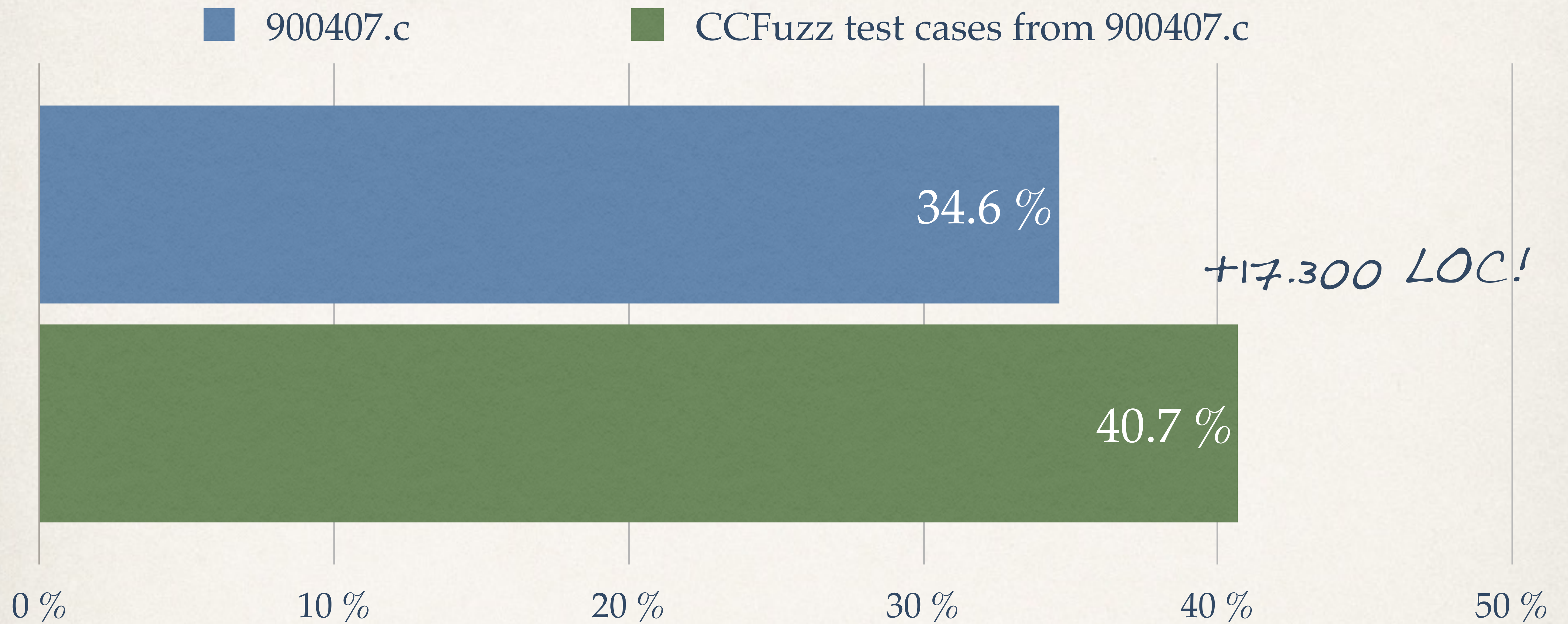
```
foo (a, b, p)
     int *p;
{
  int c;
  p[1] = a + 0x1000;
  c = b + 0xffff0000;
  if ((b + 0xffff0000) == 0)
    c++;
  p[2] = c;
}
```

900407.c

C Torture Test

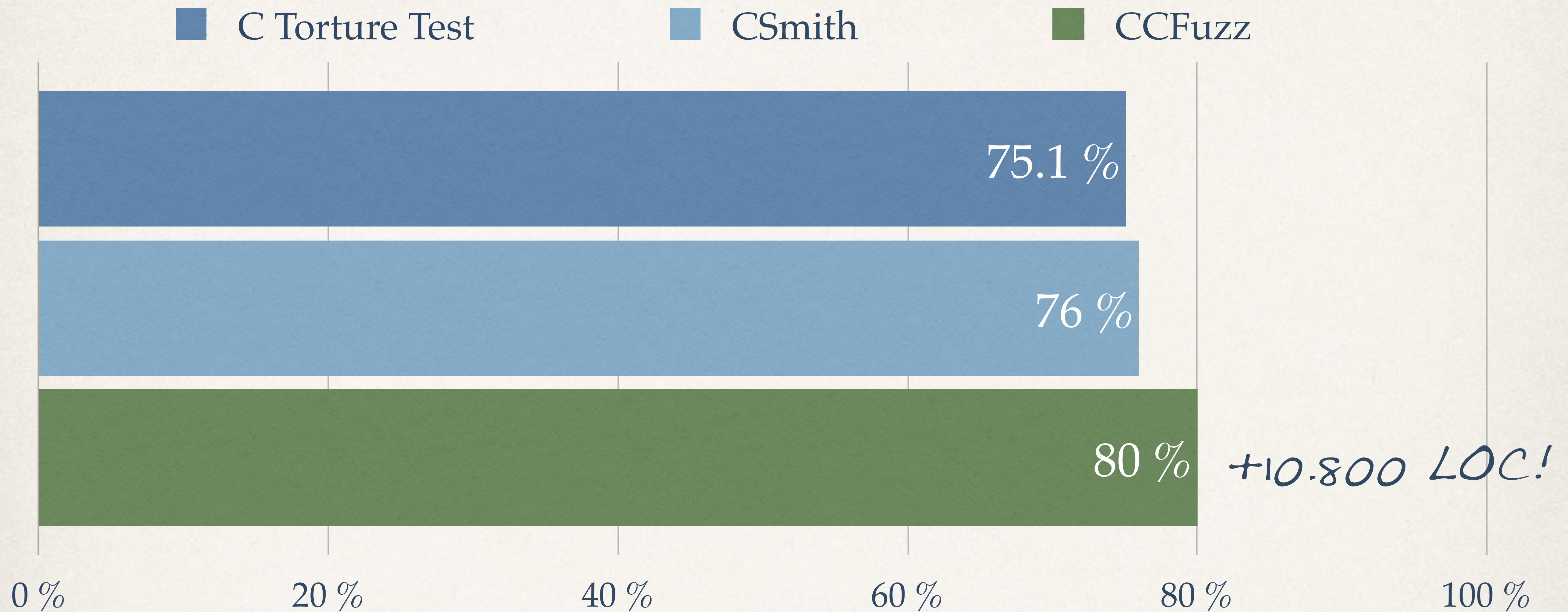
Results

Right from of the trenches



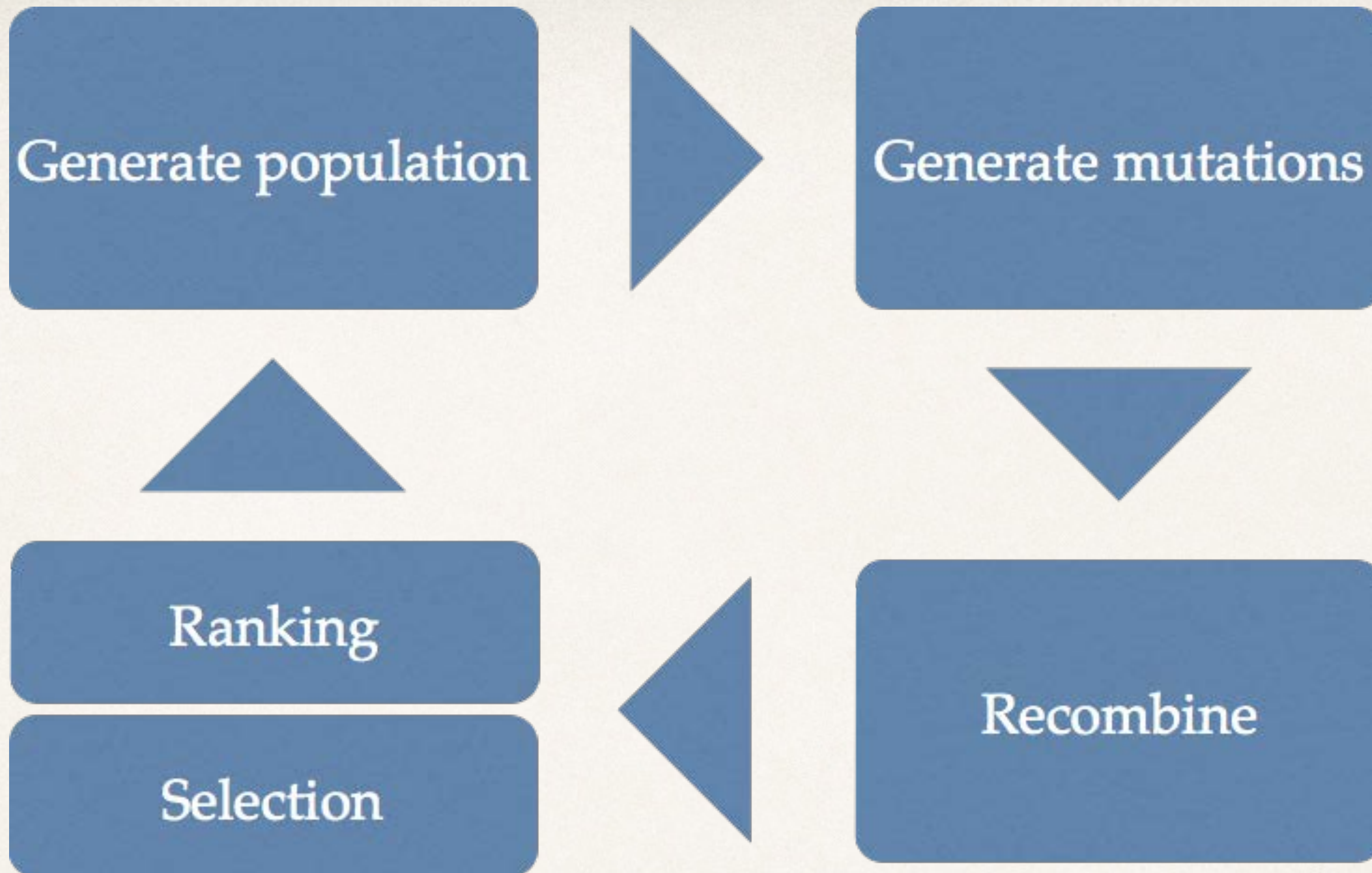
Coverage: A C Program

Right from the trenches



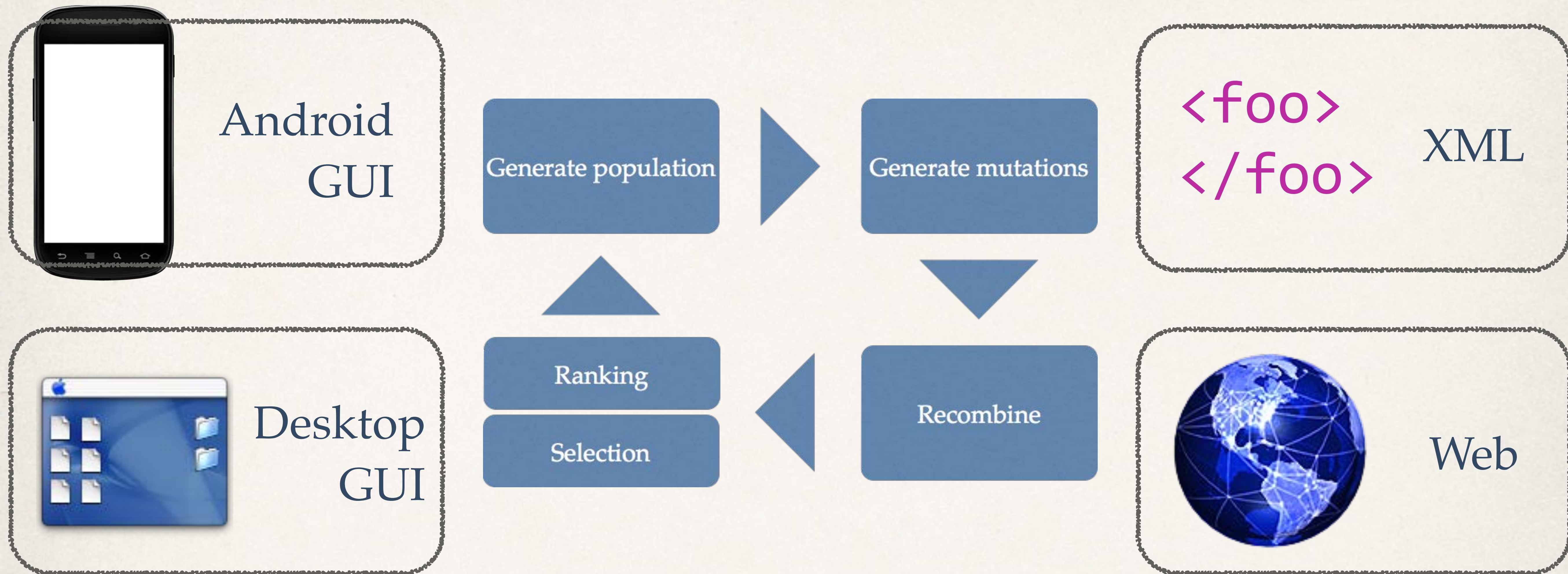
Coverage: A C Test Suite

Right from the trenches



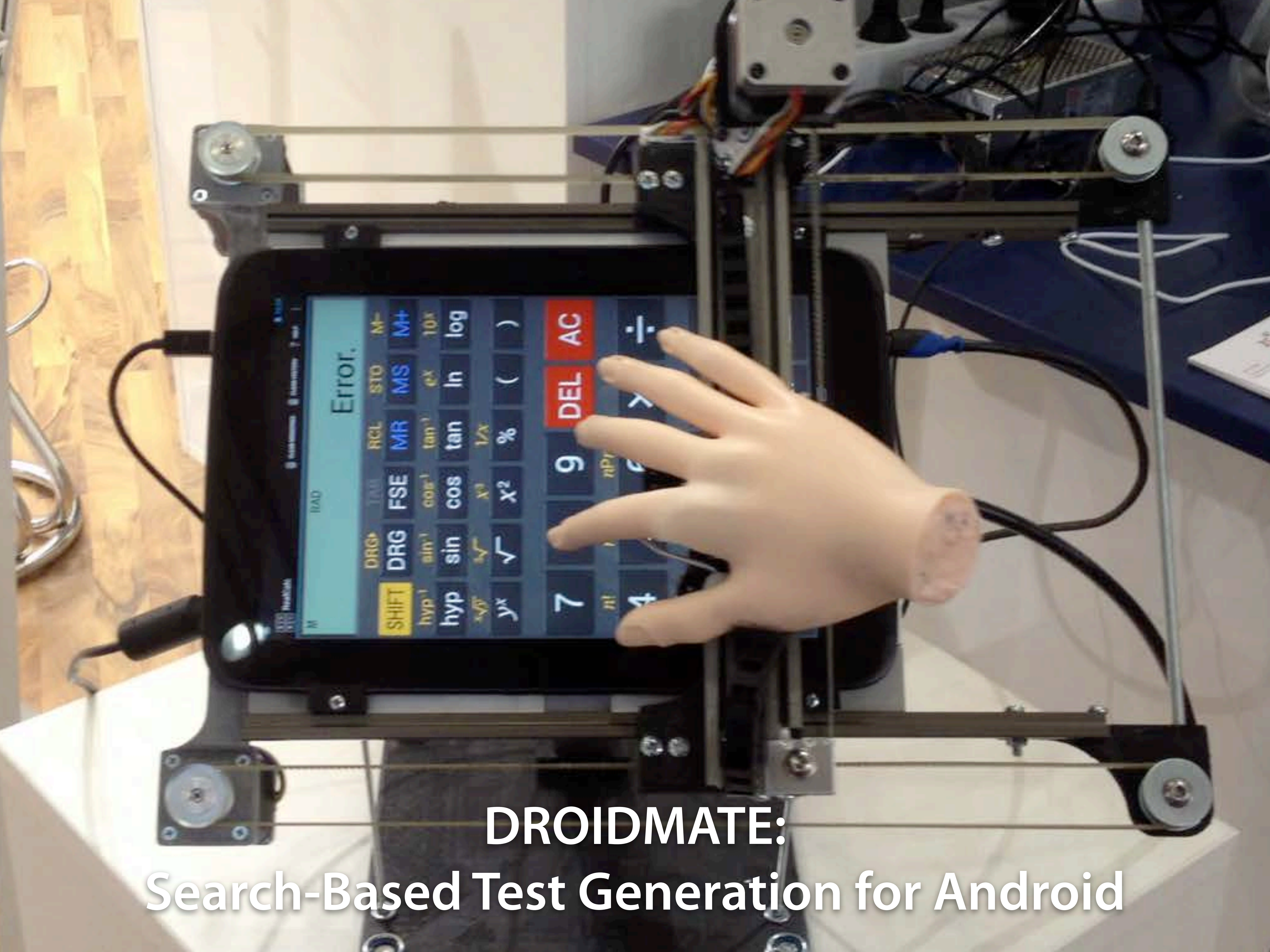
Test Generators

Basic principle: evolutionary testing



Test Generators

We applied this principle on a multitude of platforms


DROIDMATE:
Search-Based Test Generation for Android

Target
Recent Changes

Target
Code Issues

Target
Core Dumps

Cover
Input Features

Cover
Output Features

all by adapting the fitness function

Research Questions

We can adapt the fitness function towards new goals

Automatic Parallelization

Switch between sequential and parallel functions at runtime

```
long performTask(int size1, int size2) {
    list *x = makeList(size1);
    list *y = makeList(size2);
    long hashX = hashList(x);
    long hashY = hashList(y);
    freeList(x);
    freeList(y);
    return hashX * hashY;
}

long hashList(list *x) {
    if (x == 0) return 0;
    return hashElem(x) + 31 * hashList(x->next);
}
```

Parallelize this!

Switch between sequential and parallel functions at runtime

```
odin : demo $
```

SAMBAMBA: A Parallelizing C Compiler

```
odin.cs.uni-saarland.de : 1:top- 2:bash*          97.54% - 0.88 1.00 0.95 - Wed Mar 21, 10:4
```

Target
Recent Changes

Target
Code Issues

Target
Core Dumps

Cover
Input Features

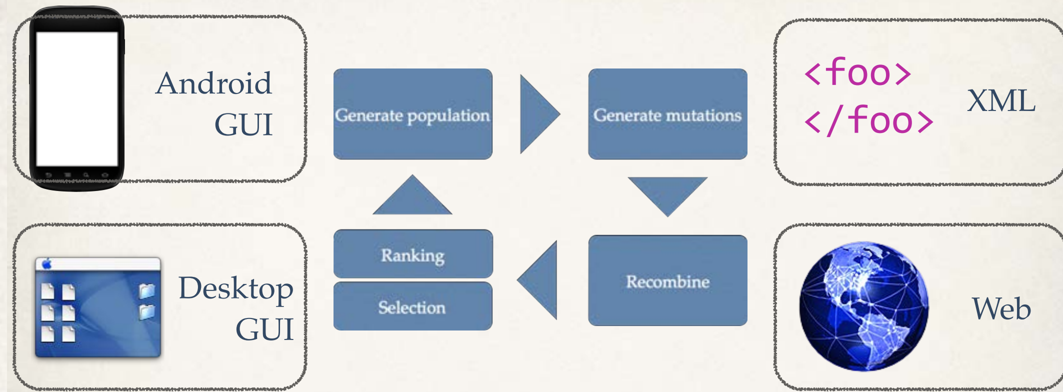
Find
Best Optimizations

Cover
Output Features

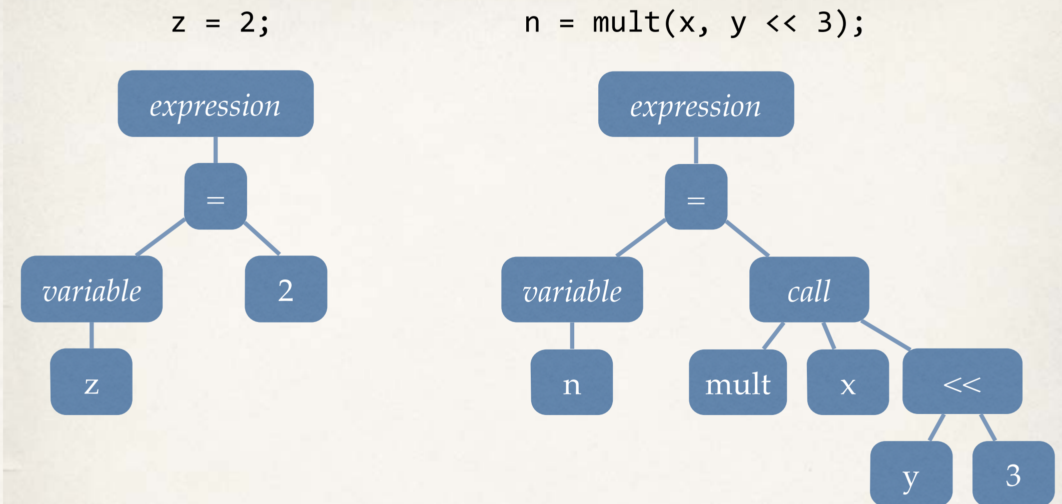
all by adapting the fitness function

Research Questions

We can adapt the fitness function towards new goals



Test Generators



Recombination

Automatic Parallelization

Switch between sequential and parallel functions at runtime

all by adapting the fitness function

Research Questions

We can adapt the fitness function towards new goals

Search-Based System Testing

Andreas Zeller • Saarland University

http://www.st.cs.uni-saarland.de/testing/