# LLVM-based mutation testing for C and C++

Alex Denisov, Virtual LLVM Dev Meeting 2020

# What is Mutation Testing?

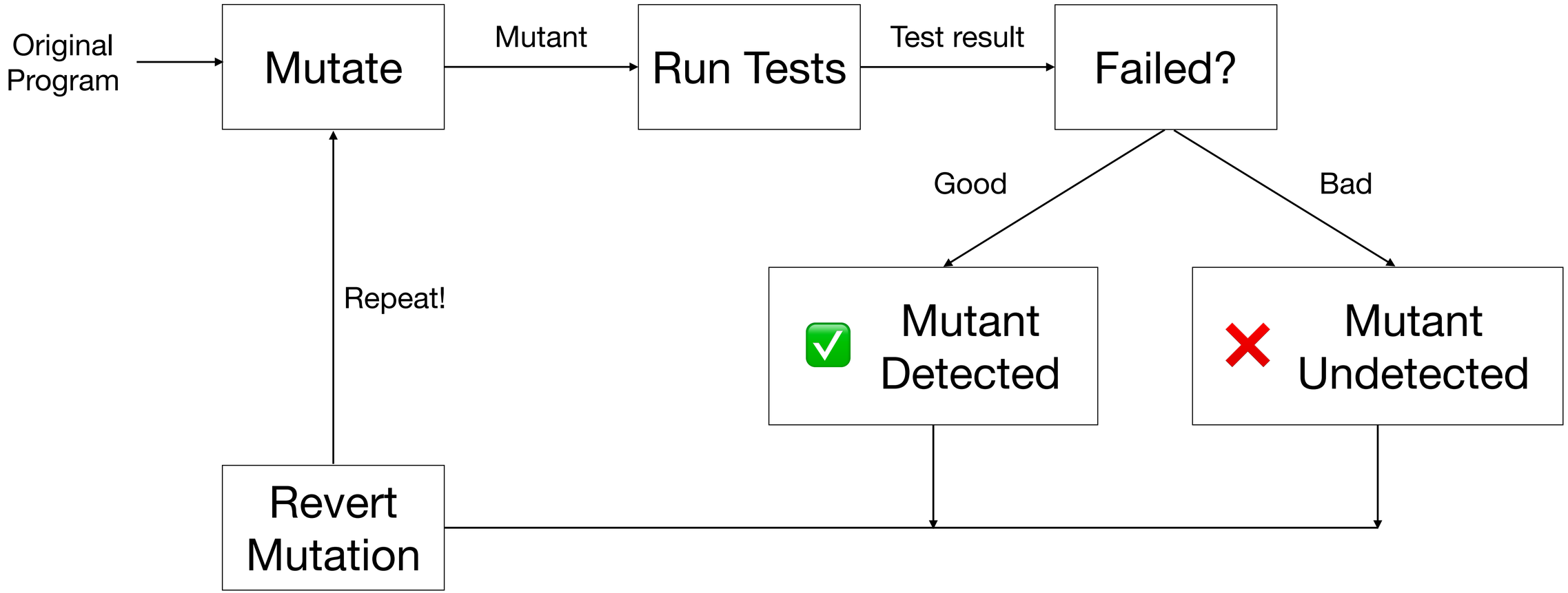
```c
#include <assert.h>

int sum(int a, int b) {
  return a + b;
}

int main() {
  // Associative: a + b = b + a
  assert(sum(5, 10) == sum(10, 5));

  // Commutative: a + (b + c) = (a + b) + c
  assert(sum(3, sum(4, 5)) == sum(sum(3, 4), 5));

  return 0;
}
```

# What is Mutation Testing?

```c
#include <assert.h>

int sum(int a, int b) {
  return a + b;
}

int main() {
  // Associative: a + b = b + a
  assert(sum(5, 10) == sum(10, 5));

  // Commutative: a + (b + c) = (a + b) + c
  assert(sum(3, sum(4, 5)) == sum(sum(3, 4), 5));

  return 0;
}
```

```c
#include <assert.h>

int sum(int a, int b) {
  return a * b;
}

int main() {
  // Associative: a + b = b + a
  assert(sum(5, 10) == sum(10, 5));

  // Commutative: a + (b + c) = (a + b) + c
  assert(sum(3, sum(4, 5)) == sum(sum(3, 4), 5));

  return 0;
}
```

# What is Mutation Testing?



# Mutation Operators

```c
#include <assert.h>

int sum(int a, int b) {
  return a + b;
}

int main() {
  // Associative: a + b = b + a
  assert(sum(5, 10) == sum(10, 5));

  // Commutative: a + (b + c) = (a + b) + c
  assert(sum(3, sum(4, 5)) == sum(sum(3, 4), 5));

  return 0;
}
```

More mutations?

```c
int sum(int a, int b) {
  return a * b;
}
```

```c
int sum(int a, int b) {
  return a;
}
```

```c
int sum(int a, int b) {
  return a - b;
}
```

```c
int sum(int a, int b) {
  return b;
}
```

```c
int sum(int a, int b) {
  return a / b;
}
```

```c
int sum(int a, int b) {
  return 0;
}
```

```c
int sum(int a, int b) {
  return a % b;
}
```

```c
int sum(int a, int b) {
  return 42;
}
```

# Mull

## Practical mutation testing tool for C and C++

- Built with large projects in mind
- Transparent
- Deterministic*
- Cross-platform (Linux, macOS, ~~FreeBSD~~)
- Open Source
  https://github.com/mull-project/mull

# Why Mutation Testing?

- Evaluates quality of a test suite
- ???

# Example #1

## Code coverage report

```
    Matrix4D::Matrix4D(double* arr) {
 1      int k = 0;
10      for(int i = 0 ; i<4; i++) {
40          for(int j = 0; j<4; j++) {
16              r[i][j]=arr[k];
16              k++;
16          }
 4      }
 1  }

    Matrix4D::Matrix4D(const Matrix4D& other) {
10      for(int i = 0 ; i<4; i++) {
40          for(int j = 0; j<4; j++) {
16              r[i][j]=other.r[i][j];
16          }
 4      }
 1  }
```

```
    void MATRIX4D_CONSTRUCTOR() {
 1      double matVals[] = {0.12, 3.45, 6.78, 9.01,
                            2.34, 5.67, 8.90, 1.23,
                            4.56, 7.89, 0.12, 3.45,
                            6.78, 9.01, 2.34, 5.67};
 1      Matrix4D mat1(matVals);
 1      Matrix4D mat2(mat1);

 7      CPPUNIT_ASSERT(compareDouble(0.12, mat2.r[0][0]));
 7      CPPUNIT_ASSERT(compareDouble(3.45, mat2.r[1][0]));
 7      CPPUNIT_ASSERT(compareDouble(6.78, mat2.r[2][0]));
 7      CPPUNIT_ASSERT(compareDouble(9.01, mat2.r[3][0]));
 7      CPPUNIT_ASSERT(compareDouble(2.34, mat2.r[0][1]));
 7      CPPUNIT_ASSERT(compareDouble(5.67, mat2.r[1][1]));
 7      CPPUNIT_ASSERT(compareDouble(8.90, mat2.r[2][1]));
 7      CPPUNIT_ASSERT(compareDouble(1.23, mat2.r[3][1]));
 7      CPPUNIT_ASSERT(compareDouble(4.56, mat2.r[0][2]));
 7      CPPUNIT_ASSERT(compareDouble(7.89, mat2.r[1][2]));
 7      CPPUNIT_ASSERT(compareDouble(0.12, mat2.r[2][2]));
 7      CPPUNIT_ASSERT(compareDouble(3.45, mat2.r[3][2]));
 7      CPPUNIT_ASSERT(compareDouble(6.78, mat2.r[0][3]));
 7      CPPUNIT_ASSERT(compareDouble(9.01, mat2.r[1][3]));
 7      CPPUNIT_ASSERT(compareDouble(2.34, mat2.r[2][3]));
 7      CPPUNIT_ASSERT(compareDouble(5.67, mat2.r[3][3]));
 1  }
```

# Example #1

## Mutation coverage report

```
1  #include "matlib.h"
2
3  Matrix4D::Matrix4D(double* arr) {
4      int k = 0;
5      for(int i = 0 ; i<4; i++) {
6          for(int j = 0; j<4; j++) {
7              r[i][j]=arr[k];
8              k++;
9          }
10     }
11 }
12
13 Matrix4D::Matrix4D(const Matrix4D& other) {
14     for(int i = 0 ; i<4; i++) {
15         for(int j = 0; j<4; j++) {
16             r[i][j]=other.r[i][j];
17         }
18     }
19 }
20
```

# Example #1

## The problem

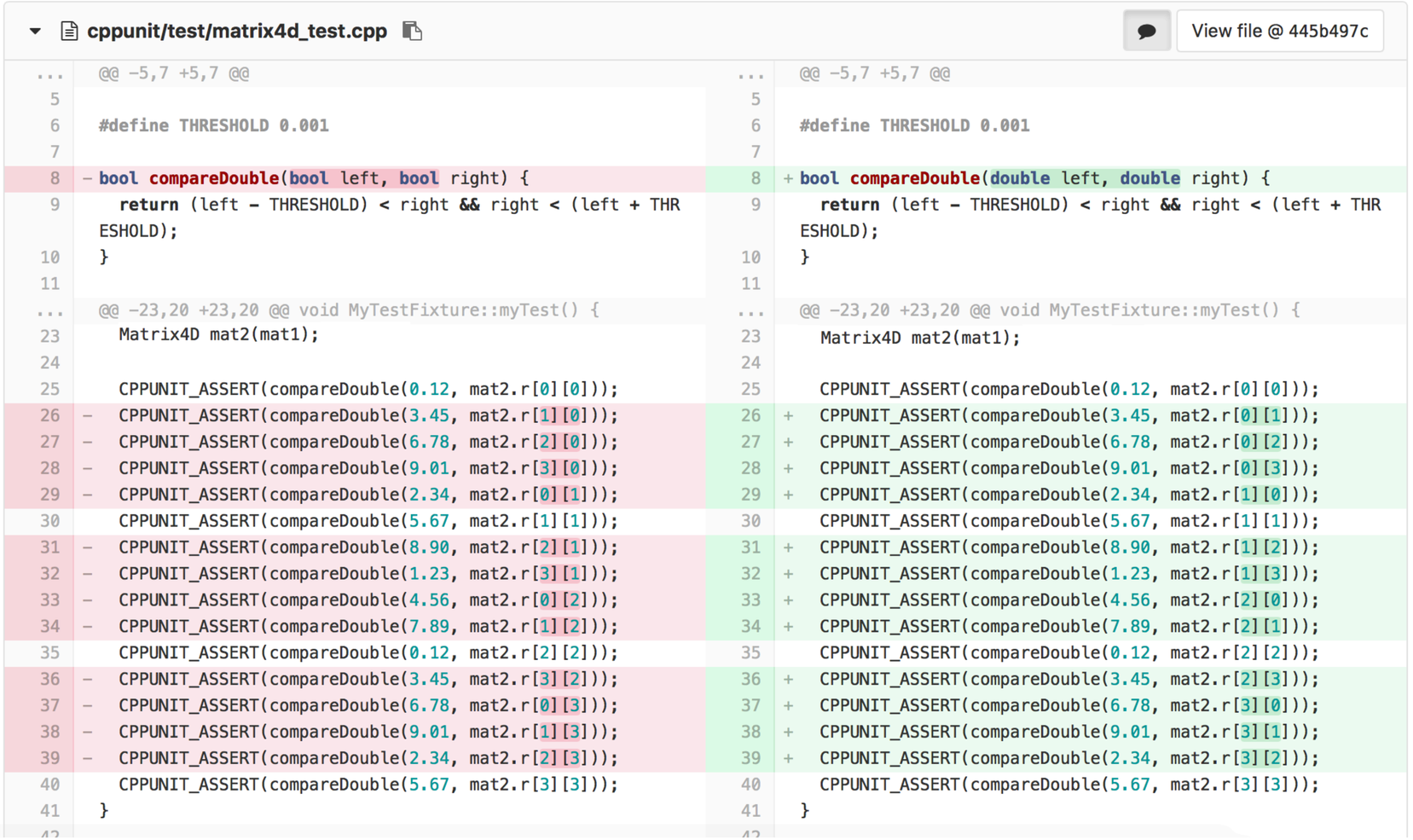
```
bool compareDouble(bool left, bool right) {
  return (left - THRESHOLD) < right && right < (left + THRESHOLD);
}

// compareDouble(0.12, 1)   -> true
// compareDouble(0.12, 122) -> true
// compareDouble(1000, 500) -> true
// compareDouble(0, 0)      -> true
// compareDouble(0, 100)    -> false
// compareDouble(100, 0)    -> false
```

N.B. clang gives a warning, gcc does not

```
implicit conversion from 'double' to 'bool' changes value from 0.12 to true
```

# Example #1

## The solution



cppunit/test/matrix4d_test.cpp

View file @ 445b497c

```
...  @@ -5,7 +5,7 @@
5
6    #define THRESHOLD 0.001
7
8  - bool compareDouble(bool left, bool right) {
9      return (left - THRESHOLD) < right && right < (left + THRESHOLD);
10   }
11
...  @@ -23,20 +23,20 @@ void MyTestFixture::myTest() {
23     Matrix4D mat2(mat1);
24
25     CPPUNIT_ASSERT(compareDouble(0.12, mat2.r[0][0]));
26 -   CPPUNIT_ASSERT(compareDouble(3.45, mat2.r[1][0]));
27 -   CPPUNIT_ASSERT(compareDouble(6.78, mat2.r[2][0]));
28 -   CPPUNIT_ASSERT(compareDouble(9.01, mat2.r[3][0]));
29 -   CPPUNIT_ASSERT(compareDouble(2.34, mat2.r[0][1]));
30     CPPUNIT_ASSERT(compareDouble(5.67, mat2.r[1][1]));
31 -   CPPUNIT_ASSERT(compareDouble(8.90, mat2.r[2][1]));
32 -   CPPUNIT_ASSERT(compareDouble(1.23, mat2.r[3][1]));
33 -   CPPUNIT_ASSERT(compareDouble(4.56, mat2.r[0][2]));
34 -   CPPUNIT_ASSERT(compareDouble(7.89, mat2.r[1][2]));
35     CPPUNIT_ASSERT(compareDouble(0.12, mat2.r[2][2]));
36 -   CPPUNIT_ASSERT(compareDouble(3.45, mat2.r[3][2]));
37 -   CPPUNIT_ASSERT(compareDouble(6.78, mat2.r[0][3]));
38 -   CPPUNIT_ASSERT(compareDouble(9.01, mat2.r[1][3]));
39 -   CPPUNIT_ASSERT(compareDouble(2.34, mat2.r[2][3]));
40     CPPUNIT_ASSERT(compareDouble(5.67, mat2.r[3][3]));
41   }
```

```
...  @@ -5,7 +5,7 @@
5
6    #define THRESHOLD 0.001
7
8  + bool compareDouble(double left, double right) {
9      return (left - THRESHOLD) < right && right < (left + THRESHOLD);
10   }
11
...  @@ -23,20 +23,20 @@ void MyTestFixture::myTest() {
23     Matrix4D mat2(mat1);
24
25     CPPUNIT_ASSERT(compareDouble(0.12, mat2.r[0][0]));
26 +   CPPUNIT_ASSERT(compareDouble(3.45, mat2.r[0][1]));
27 +   CPPUNIT_ASSERT(compareDouble(6.78, mat2.r[0][2]));
28 +   CPPUNIT_ASSERT(compareDouble(9.01, mat2.r[0][3]));
29 +   CPPUNIT_ASSERT(compareDouble(2.34, mat2.r[1][0]));
30     CPPUNIT_ASSERT(compareDouble(5.67, mat2.r[1][1]));
31 +   CPPUNIT_ASSERT(compareDouble(8.90, mat2.r[1][2]));
32 +   CPPUNIT_ASSERT(compareDouble(1.23, mat2.r[1][3]));
33 +   CPPUNIT_ASSERT(compareDouble(4.56, mat2.r[2][0]));
34 +   CPPUNIT_ASSERT(compareDouble(7.89, mat2.r[2][1]));
35     CPPUNIT_ASSERT(compareDouble(0.12, mat2.r[2][2]));
36 +   CPPUNIT_ASSERT(compareDouble(3.45, mat2.r[2][3]));
37 +   CPPUNIT_ASSERT(compareDouble(6.78, mat2.r[3][0]));
38 +   CPPUNIT_ASSERT(compareDouble(9.01, mat2.r[3][1]));
39 +   CPPUNIT_ASSERT(compareDouble(2.34, mat2.r[3][2]));
40     CPPUNIT_ASSERT(compareDouble(5.67, mat2.r[3][3]));
41   }
```

# Example #2

```
int32_t File::read(char *buf, int32_t len) {
  if (filePtr->is_open()) {
    // actual reading
  }
  return -1;
}

int32_t File::getString(char *buf, int32_t max) {
  int32_t len = read(buf, max - 1);
  buf[len] = '\0';
  return len;
}
```

```
void ReadEmptyFileTest() {
  File f;
  f.open("an_empty_file");

  char buf[4] = { 0 };

  f.getString(buf, 4);

  CPPUNIT_ASSERT(buf[0], '\0');
  CPPUNIT_ASSERT(buf[1], '\0');
  CPPUNIT_ASSERT(buf[2], '\0');
  CPPUNIT_ASSERT(buf[3], '\0');
}
```

# Example #2

```
int32_t File::read(char *buf, int32_t len) {
  if (filePtr->is_open()) {
    // actual reading
  }
  return -1;
}

int32_t File::getString(char *buf, int32_t max) {
  int32_t len = read(buf, max - 1); // len = -1
  buf[len] = '\0';                  // buf[-1] = 0
  return len;
}
```

```
void ReadEmptyFileTest() {
  File f;
  // f.open("an_empty_file");

  char buf[4] = { 0 };

  f.getString(buf, 4);

  CPPUNIT_ASSERT(buf[0], '\0');
  CPPUNIT_ASSERT(buf[1], '\0');
  CPPUNIT_ASSERT(buf[2], '\0');
  CPPUNIT_ASSERT(buf[3], '\0');
}
```

# Example #3

```
int32_t bitstuff(int32_t number) {
  number |= (1 << 7);
  number |= (1 << 15);
  number |= (1 << 23);
  number |= (1 << 31);
  return number;
}
```

```
int32_t bitstuff(int32_t number) {
  number |= (1 << 7);
  number |= (1 << 15);
  number |= (1 << 23);
  number &= (1 << 31);
  return number;
}
```

```
int32_t bitstuff(int32_t number) {
  number |= (1 << 7);
  number |= (1 << 15);
  number |= (1 << 0);
  number |= (1 << 31);
  return number;
}
```

# Example #3

```
int32_t bitstuff(int32_t number) {
  number |= (1 << 7);
  number |= (1 << 15);
  number |= (1 << 23);
  number |= (1 << 31);
  return number;
}
```

```
int16_t fixed(int16_t number) {
  number |= (1 << 7);
  number |= (1 << 15);
  return number;
}
```

```
int32_t bitstuff(int32_t number) {
  number |= (1 << 7);
  number |= (1 << 15);
  number |= (1 << 23);
  number &= (1 << 31);
  return number;
}
```

```
int32_t bitstuff(int32_t number) {
  number |= (1 << 7);
  number |= (1 << 15);
  number |= (1 << 0);
  number |= (1 << 31);
  return number;
}
```

# Why Mutation Testing?

- Evaluates quality of a test suite
- ???
  - Incorrect test
  - Potential Vulnerability
  - Dead code

# Why Mutation Testing?

- ~~Evaluates quality of a test suite~~
- Shows semantic gaps between the test suite and the software
  - Incorrect test
  - Potential Vulnerability
  - Dead code
  - Many more things

# Brief history of mutation testing

- Invented by Richard Lipton in 1971
- Implemented by Timothy Budd in 1980
- …
- …
- …
- Still niche/academia topic in 2020 (though it’s slowly changing)

# Performance

```
for_each(mutant)
 1. Add a change
 2. Compile
 3. Link
 4. Run tests
 5. Rollback the change
 6. Repeat (go to step 1)
```

# Performance

```
for_each(mutant)
 1. Add a change
 2. Compile
 3. Link
 4. Run tests
 5. Rollback the change
 6. Repeat (go to step 1)
```

```
execution time(N) =
    N * (time to change)
  + N * (time to compile)
  + N * (time to link)
  + N * (time to run tests)
  + N * (time to rollback)
```

# Performance

```c
// test.c

int S = 1000000;
void test(int a, int b) {
  int x = 0;
  int i = S;
  while (i-- > 0) {
    x = a + b;
    x = a + b;
    x = a + b;
    // 47 lines more
  }
}
```

```c
// mutant_0.c

int S = 1000000;
void test(int a, int b) {
  int x = 0;
  int i = S;
  while (i-- > 0) {
    x = a - b;
    x = a + b;
    x = a + b;
    // 47 lines more
  }
}
```

```c
// mutant_1.c

int S = 1000000;
void test(int a, int b) {
  int x = 0;
  int i = S;
  while (i-- > 0) {
    x = a + b;
    x = a - b;
    x = a + b;
    // 47 lines more
  }
}
```

# Performance

baseline = compile(0.03s) + link(0.04s) + run(0.01s) = **~0.08s**

naïve execution time = mutate(0.02s) +

  50 * compile(0.03s) +

  50 * link(0.04s) +

  50 * run(0.01s) = **~4.02s**

naïve slowdown = ~4.02s / 0.08s = **~50x**

# Performance

```
> clang -g -fembed-bitcode test.c -o test
> mull-cxx -test-framework=CustomTest -mutators=cxx_add_to_sub test
[info] Extracting bitcode from executable (threads: 1)
       [####################################] 1/1. Finished in 4ms
[info] Loading bitcode files (threads: 1)
       [####################################] 1/1. Finished in 11ms
[info] Compiling instrumented code (threads: 1)
       [####################################] 1/1. Finished in 12ms
...
/tmp/sc-MkpAK7Yit/test.c:54:11: warning: Survived: Replaced + with - [cxx_add_to_sub]
    x = a + b;
          ^
/tmp/sc-MkpAK7Yit/test.c:55:11: warning: Survived: Replaced + with - [cxx_add_to_sub]
    x = a + b;
          ^
[info] Mutation score: 1%
[info] Total execution time: 526ms
```

# Performance

baseline = compile(0.03s) + link(0.04s) + run(0.01s) = **~0.08s**

naïve execution time = mutate(0.02s) +

  50 * compile(0.03s) +

  50 * link(0.04s) +

  50 * run(0.01s) = **~4.02s**

naïve slowdown = ~4.02s / 0.08s = **~50x**

mull execution time = **~0.5s**

mull slowdown = ~0.5s / 0.08s = **~6x**

# Performance

baseline = compile(0.03s) + link(0.04s) + run(0.01s) = **~0.08s**

naïve execution time = mutate(0.02s) +

  50 * compile(0.03s) +

  50 * link(0.04s) +

  50 * run(0.01s) = **~4.02s**

naïve slowdown = ~4.02s / 0.08s = **~50x**

mull execution time = **~0.5s**

mull slowdown = ~0.5s / 0.08s = **~6x**

Applying Mutation Analysis On Kernel Test Suites: An Experience Report
https://ieeexplore.ieee.org/document/7899043/

~3500 hours!

# Mull Algorithm

```c
#include <assert.h>

int sum(int a, int b) {
  return a + b;
}

int main() {
  // Associative: a + b = b + a
  assert(sum(5, 10) == sum(10, 5));

  // Commutative: a + (b + c) = (a + b) + c
  assert(sum(3, sum(4, 5)) == sum(sum(3, 4), 5));

  return 0;
}
```

# Mull Algorithm

```c
#include <assert.h>

int sum(int a, int b) {
  return a + b;
}

int main() {
  // Associative: a + b = b + a
  assert(sum(5, 10) == sum(10, 5));

  // Commutative: a + (b + c) = (a + b) + c
  assert(sum(3, sum(4, 5)) == sum(sum(3, 4), 5));

  return 0;
}
```

```c
int sum_original(int a, int b) {
  return a + b;
}

int sum_mutant_0(int a, int b) {
  return a - b;
}

int sum_mutant_1(int a, int b) {
  return a * b;
}

int sum_mutant_2(int a, int b) {
  return a / b;
}
```

# Mull Algorithm

```c
#include <assert.h>

int (*sum_ptr)(int, int) = sum_original;

int sum(int a, int b) {
  return a + b;
}

int main() {
  // Associative: a + b = b + a
  assert(sum(5, 10) == sum(10, 5));

  // Commutative: a + (b + c) = (a + b) + c
  assert(sum(3, sum(4, 5)) == sum(sum(3, 4), 5));

  return 0;
}
```

```c
int sum_original(int a, int b) {
  return a + b;
}

int sum_mutant_0(int a, int b) {
  return a - b;
}

int sum_mutant_1(int a, int b) {
  return a * b;
}

int sum_mutant_2(int a, int b) {
  return a / b;
}
```

# Mull Algorithm

```c
#include <assert.h>

int (*sum_ptr)(int, int) = sum_original;

int sum(int a, int b) {
  return sum_ptr(a, b);
}

int main() {
  // Associative: a + b = b + a
  assert(sum(5, 10) == sum(10, 5));

  // Commutative: a + (b + c) = (a + b) + c
  assert(sum(3, sum(4, 5)) == sum(sum(3, 4), 5));

  return 0;
}
```

```c
int sum_original(int a, int b) {
  return a + b;
}

int sum_mutant_0(int a, int b) {
  return a - b;
}

int sum_mutant_1(int a, int b) {
  return a * b;
}

int sum_mutant_2(int a, int b) {
  return a / b;
}
```

# Mull Algorithm

- Load program's Bitcode into memory
- Scan each function to find instructions to mutate
- Generate mutants
- Lower bitcode into machine code
- Execute each mutant via JIT engine (in a forked/isolated process)
- Report results

Mull it over: mutation testing based on LLVM

https://arxiv.org/abs/1908.01540

# Mull Algorithm

## Find instructions to mutate

```
for (auto &module : allModules) {
  for (auto &function : module) {
    for (auto &instruction : function) {
      if (canMutate(instruction)) {
        mutate(instruction);
      }
    }
  }
}
```

# Mull Algorithm

## Find instructions to mutate

```
for (auto &module : allModules) {
  for (auto &function : module) {
    for (auto &instruction : function) {
      if (canMutate(instruction)) {
        mutate(instruction);
      }
    }
  }
}
```

```
auto coveredFunctions =
                runTestsWithCoverage();
for (auto &function : coveredFunctions) {
  for (auto &instruction : function) {
    if (canMutate(instruction)) {
      mutate(instruction);
    }
  }
}
```

# Mull Algorithm

## Generate mutants

### Source Code

```
a + b
```

### C/C++

```
entry:
  %2 = add i64 %0, %1
  ret i64 %2
```

### Swift (and Rust)

```
entry:
  %2 = tail call { i64, i1 }
    @llvm.sadd.with.overflow.i64(i64 %0, i64 %1)
  %3 = extractvalue { i64, i1 } %2, 1
  br i1 %3, label %6, label %4

ok:
  %5 = extractvalue { i64, i1 } %2, 0
  ret i64 %5

err:
  tail call void asm sideeffect "", "n"(i32 0)
  tail call void @llvm.trap()
  unreachable
```

# Mull Algorithm

## Mutation Operators

| Operator Name | Operator Semantics |
|---|---|
| cxx_add_assign_to_sub_assign | Replaces += with -= |
| cxx_add_to_sub | Replaces + with - |
| cxx_sub_assign_to_add_assign | Replaces -= with += |
| cxx_sub_to_add | Replaces - with + |
| cxx_xor_to_or | Replaces ^ with \| |
| cxx_and_to_or | Replaces & with \| |
| cxx_or_to_and | Replaces \| with & |
| cxx_le_to_gt | Replaces <= with > |
| cxx_eq_to_ne | Replaces == with != |
| remove_void_function_mutator | Removes calls to a function returning void |

Full List: https://mull.readthedocs.io/en/latest/SupportedMutations.html

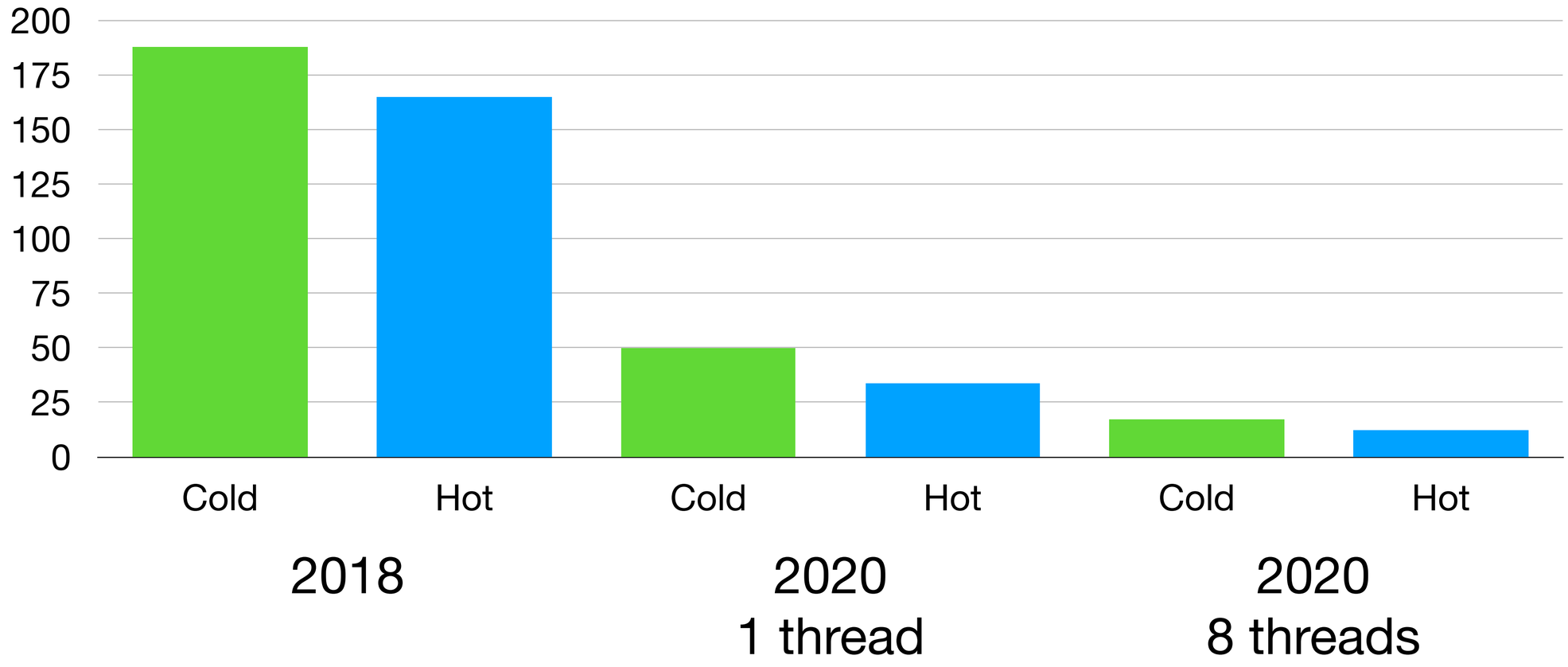
# Mull Algorithm

## Run mutants in forked process

- Clean state for each run
- Sandboxing
  - mutated code can crash
  - mutated code can hit deadlock/infinite loop
  - mutated code can *sometimes* crash

# Real Numbers

## OpenSSL test suite



# Try It

- https://mull.readthedocs.io/en/latest/GettingStarted.html
- http://github.com/mull-project/mull

- Mull it over: mutation testing based on LLVM
  https://arxiv.org/abs/1908.01540
- Building an LLVM-based tool: lessons learned
  https://www.youtube.com/watch?v=Yvj4G9B6pcU

# Questions?