Testing The Tests

Mutation Testing For C++11/14

Seph De Busser

Why Test?

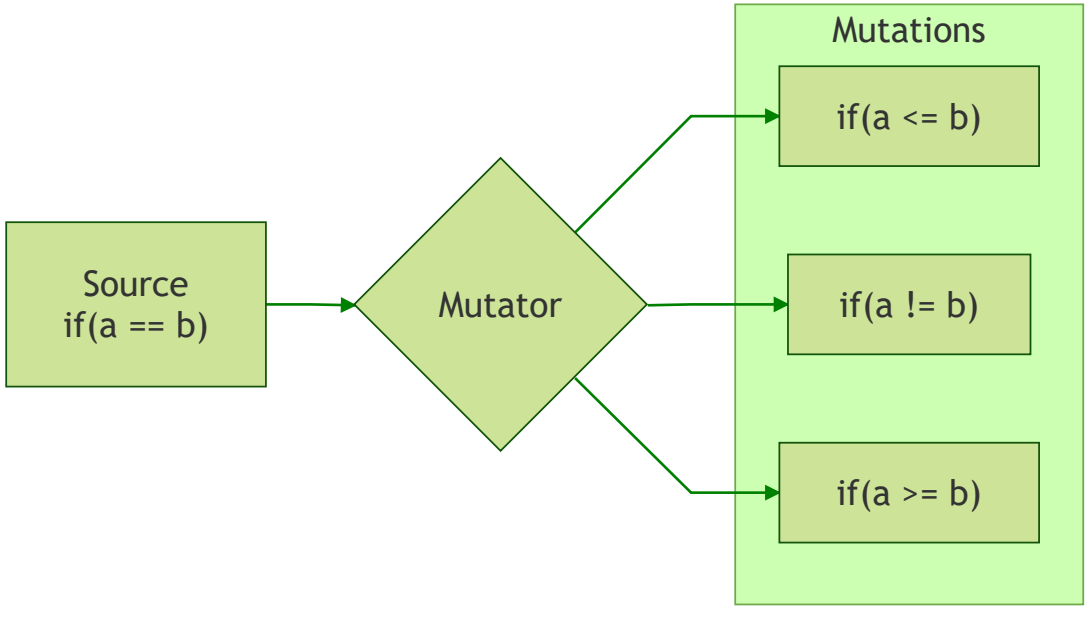
Matthias Breddin
@lunetics

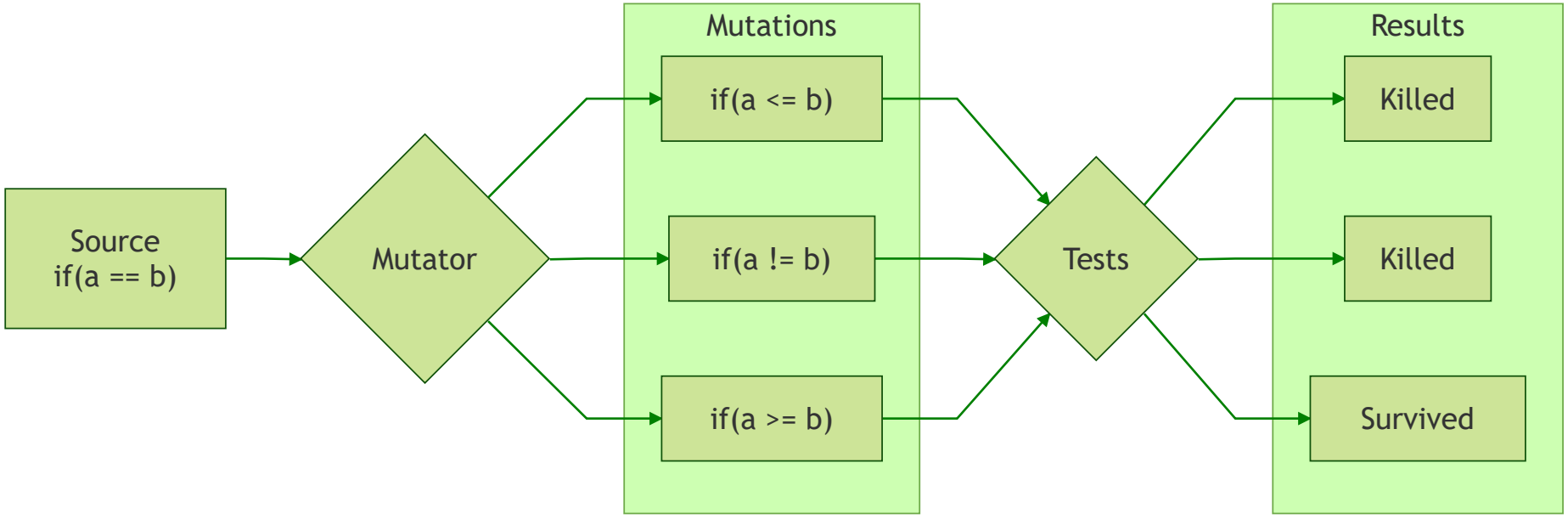
Just received a snipped from an employee of a client

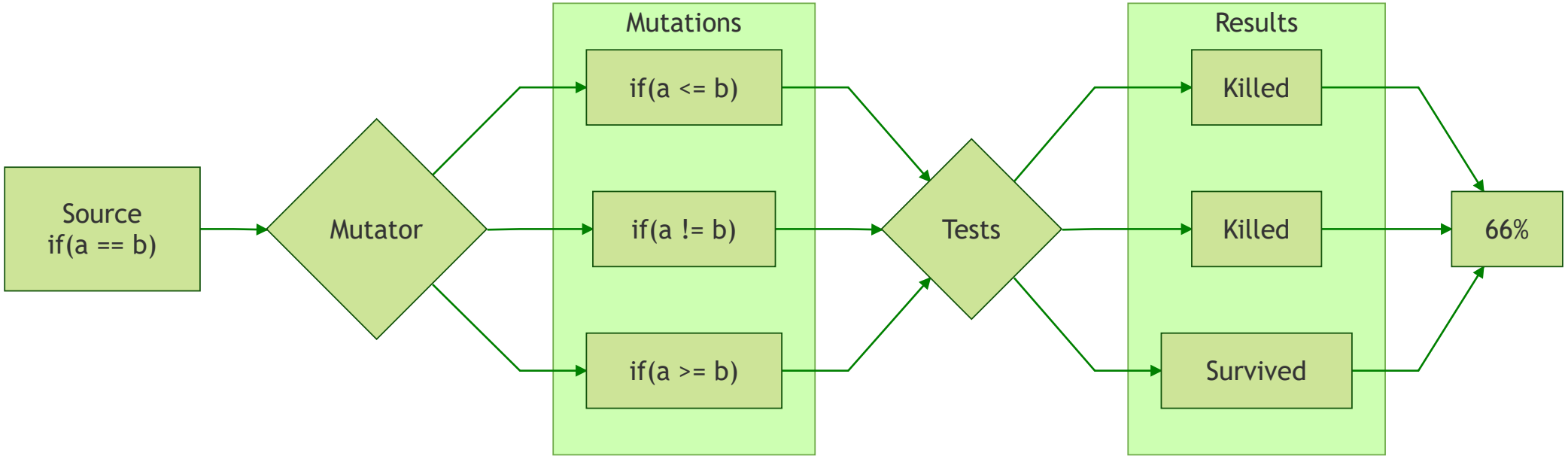
```php
<?php
declare(strict_types=1);

class coverageIncreaser
{
    public function increaseOverallCoverage()
    {
        $a = 1;
        $a++;
        $a++;
        // …
        // add some lines here if your
        // coverage dropped below the allowed level
        // …
        $a++;
        $a++;
        return $a;
    }
}
```

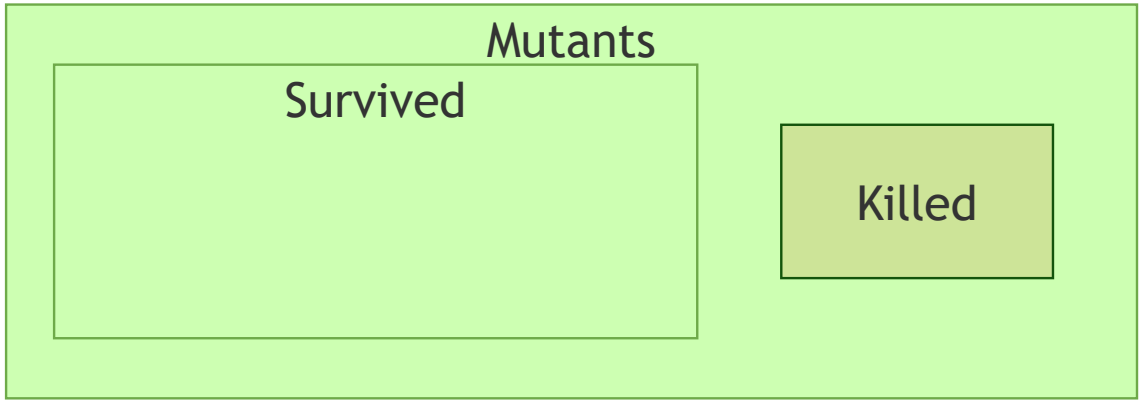
1,558 1:56 PM - Jul 29, 2019

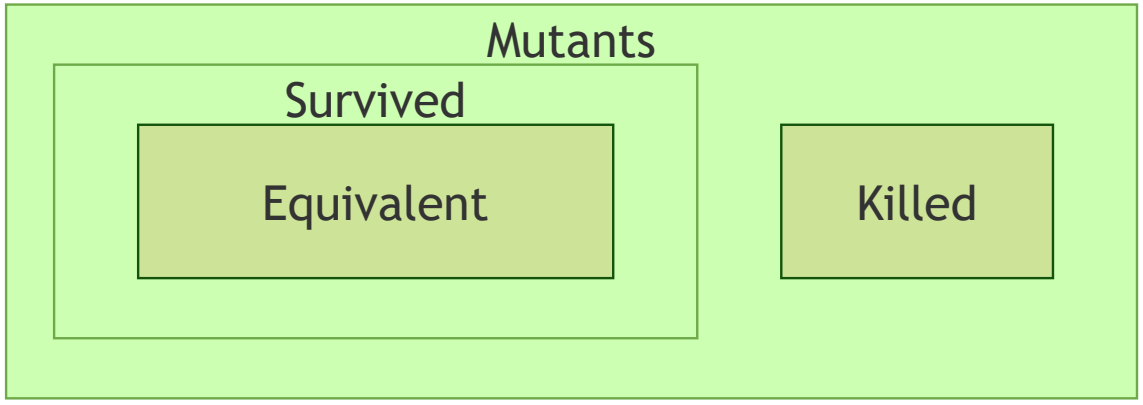
Test Confidence


Source
if(a == b)
Mutator
Mutations
if(a <= b)
if(a != b)
if(a >= b)


Source
if(a == b)
Mutator
Mutations
if(a <= b)
if(a != b)
if(a >= b)
Tests
Results
Killed
Killed
Survived


Source
if(a == b)
Mutator
Mutations
if(a <= b)
if(a != b)
if(a >= b)
Tests
Results
Killed
Killed
Survived
66%


Mutants
Survived
Killed


Mutants
Survived
Equivalent
Killed

Results

Results

- Overall test suite coverage

Results

- Overall test suite coverage
- Individual test coverage

Results

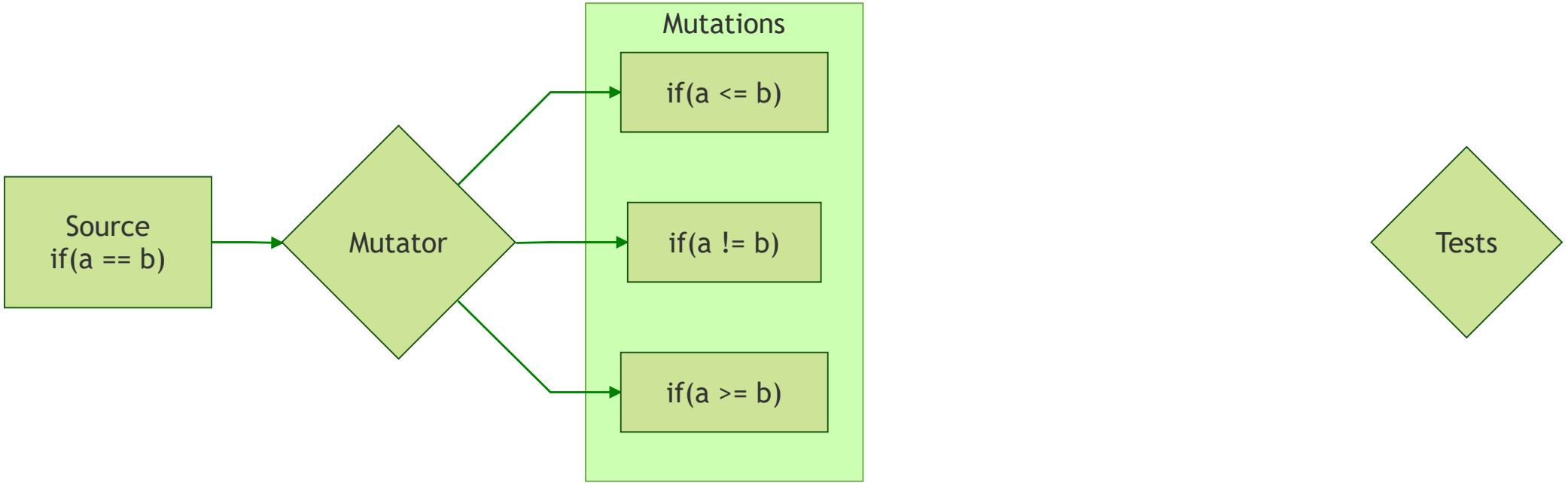
- Overall test suite coverage
- Individual test coverage
- Redundant tests

Results

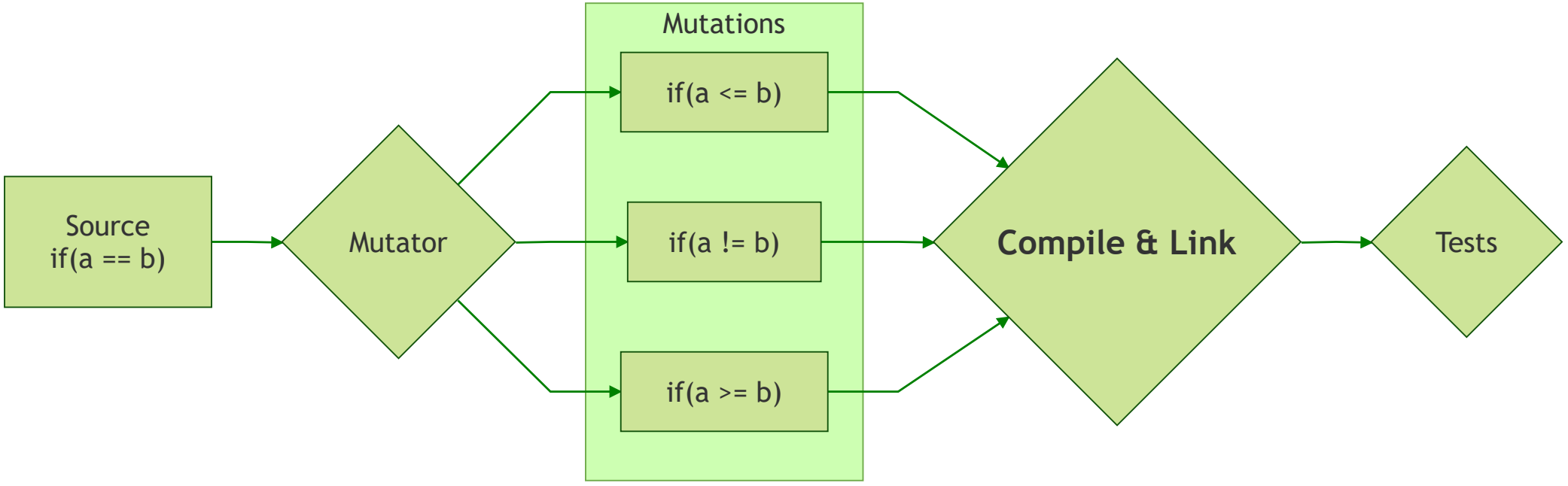
- Overall test suite coverage
- Individual test coverage
- Redundant tests
- Functionally dead code

Tool Choices

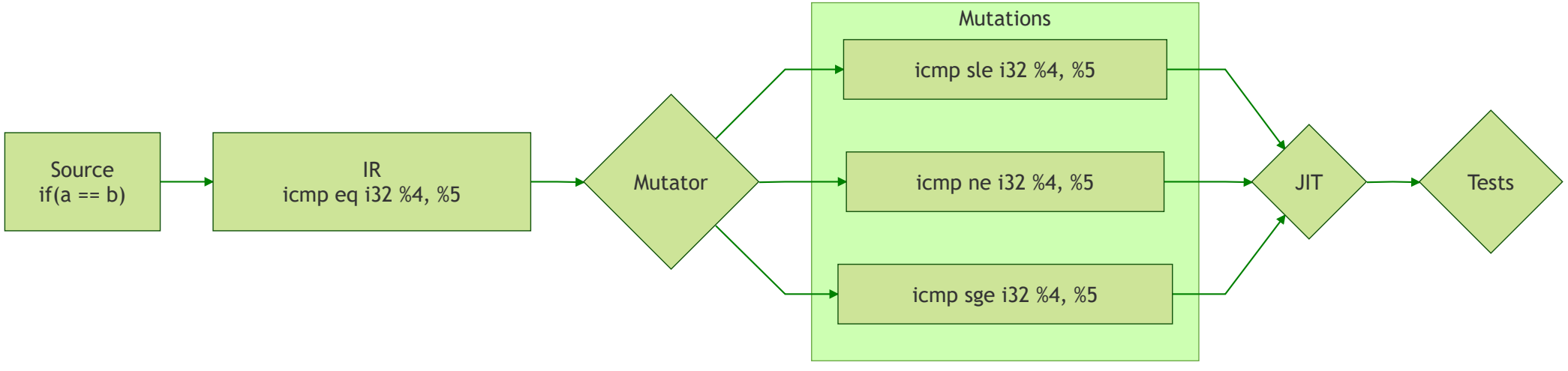
Source



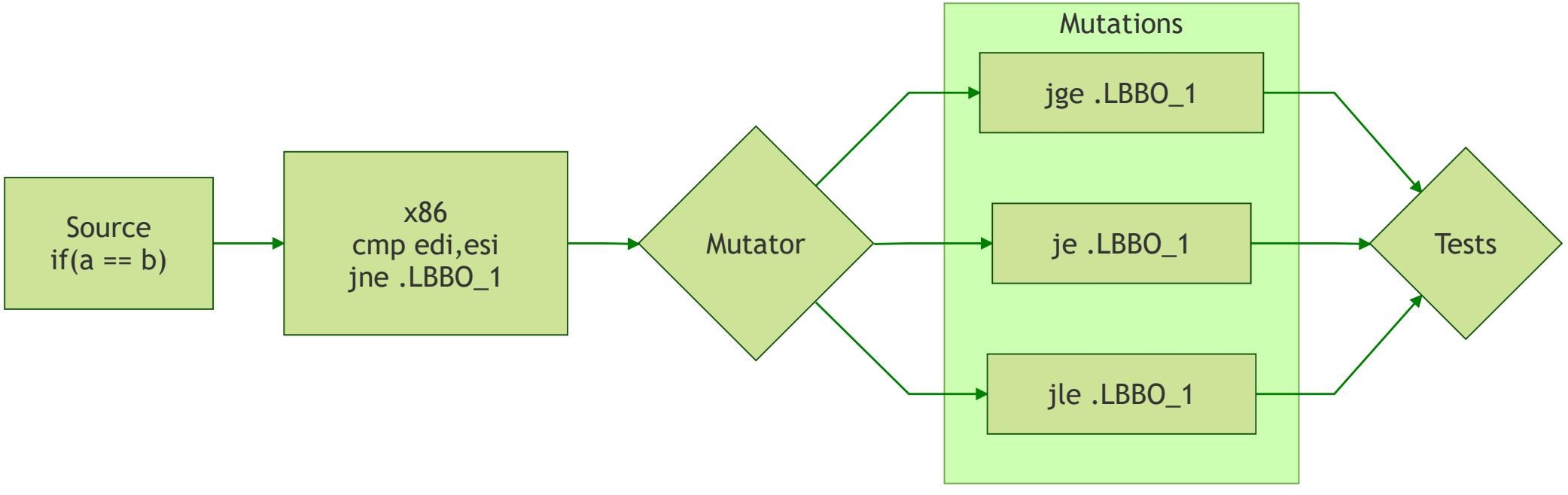
Source

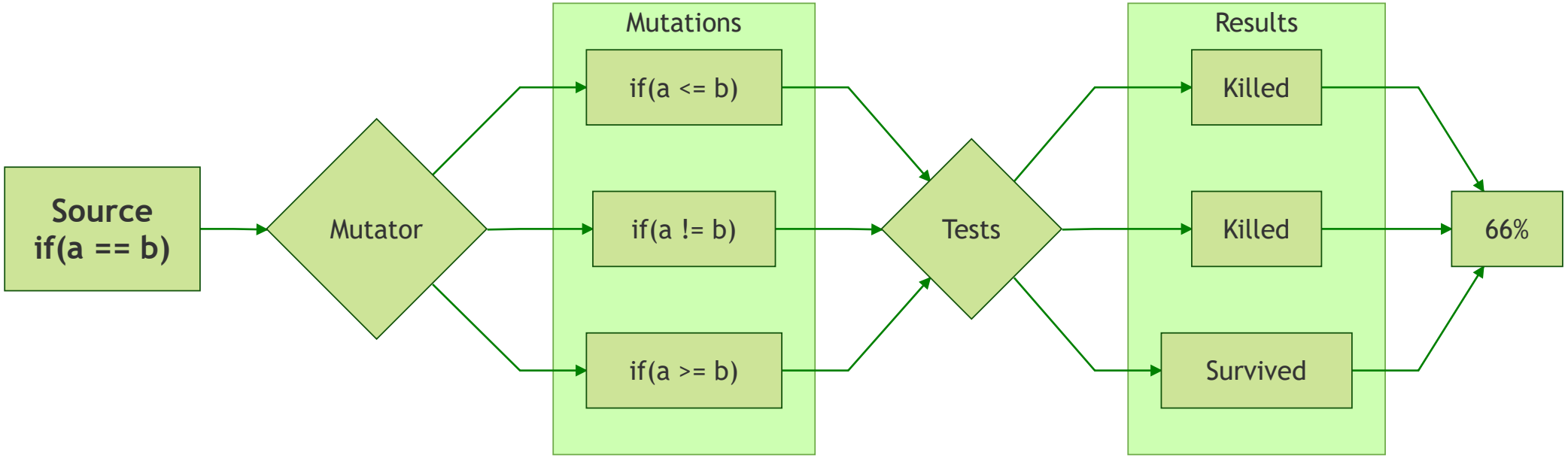

Source
if(a == b)
Mutator
Mutations
if(a <= b)
if(a != b)
if(a >= b)
Compile & Link
Tests

IR


Source
if(a == b)
IR
icmp eq i32 %4, %5
Mutator
Mutations
icmp sle i32 %4, %5
icmp ne i32 %4, %5
icmp sge i32 %4, %5
JIT
Tests

Binary


Source
if(a == b)
x86
cmp edi,esi
jne .LBBO_1
Mutator
Mutations
jge .LBBO_1
je .LBBO_1
jle .LBBO_1
Tests


Source
if(a == b)
Mutator
Mutations
if(a <= b)
if(a != b)
if(a >= b)
Tests
Results
Killed
Killed
Survived
66%

Mutation locations

Mutation locations

- All code

Mutation locations

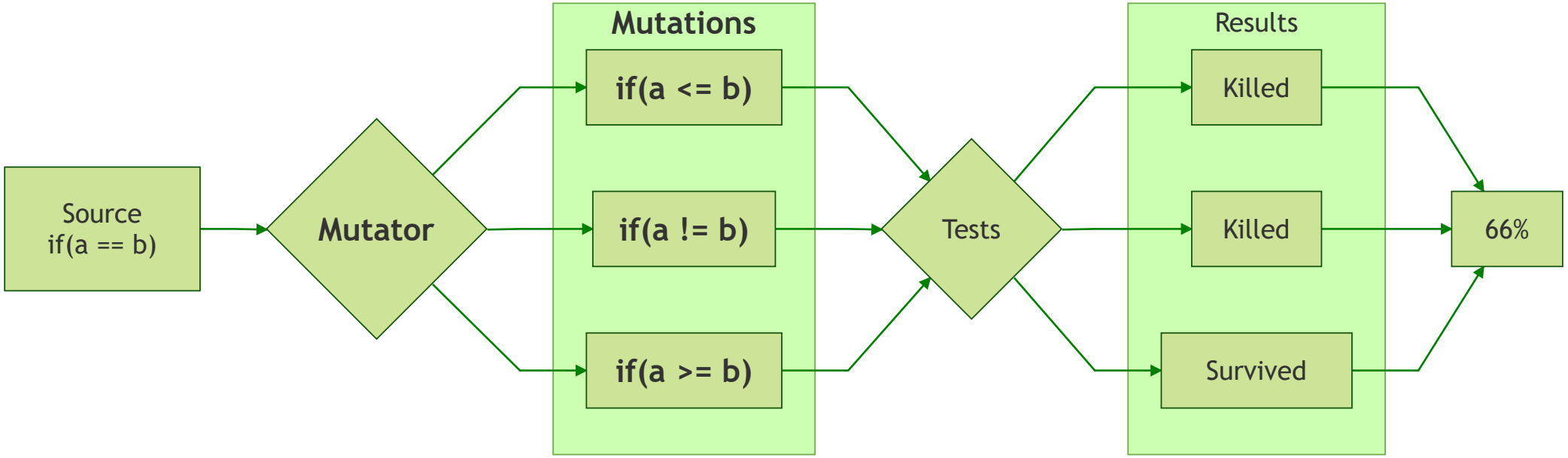
- All code
- Covered code

Mutation locations

- All code
- Covered code
- Directly covered code

Mutation locations

- All code
- Covered code
- Directly covered code
- Interesting modules


Source
if(a == b)
Mutator
Mutations
if(a <= b)
if(a != b)
if(a >= b)
Tests
Results
Killed
Killed
Survived
66%

Which mutants

Which mutants

- Operator replacements

```
auto c = a + b
```

Which mutants

- Operator replacements

```
auto c = a + b
```

```
auto c = a - b
```

```
auto c = a * b
```

```
auto c = a / b
```

Which mutants

- Operator replacements
- Statement deletion

```
void hello() {
   std::cout << "Hello World!\n";
}
```

Which mutants

- Operator replacements
- Statement deletion

```
void hello() {
    std::cout << "Hello World!\n";
}
```

```
void hello() {
}
```

Which mutants

- Operator replacements
- Statement deletion
- OOP constructs

```
struct Parent {
   virtual void process() {
      //do stuff
   }
};
struct Child : public Parent {
   void process() override {
      //do other stuff
   }
};
```

Which mutants

- Operator replacements
- Statement deletion
- OOP constructs

```
struct Parent {
   virtual void process() {
      //do stuff
   }
};
struct Child : public Parent {
   void process() override {
      //do other stuff
   }
};
```

```
struct Parent {
   virtual void process() {
      //do stuff
   }
};
struct Child : public Parent {
};
```

Which mutants

- Operator replacements
- Statement deletion
- OOP constructs
- Threading constructs

```
std::lock_guard a(mutex1);
std::lock_guard b(mutex2);
```

Which mutants

- Operator replacements
- Statement deletion
- OOP constructs
- Threading constructs

```
std::lock_guard a(mutex1);
std::lock_guard b(mutex2);
```

```
std::lock_guard b(mutex2);
std::lock_guard a(mutex1);
```

Optimizations

Optimizations

- Mutant schemata

```
auto c = a + b
```

Optimizations

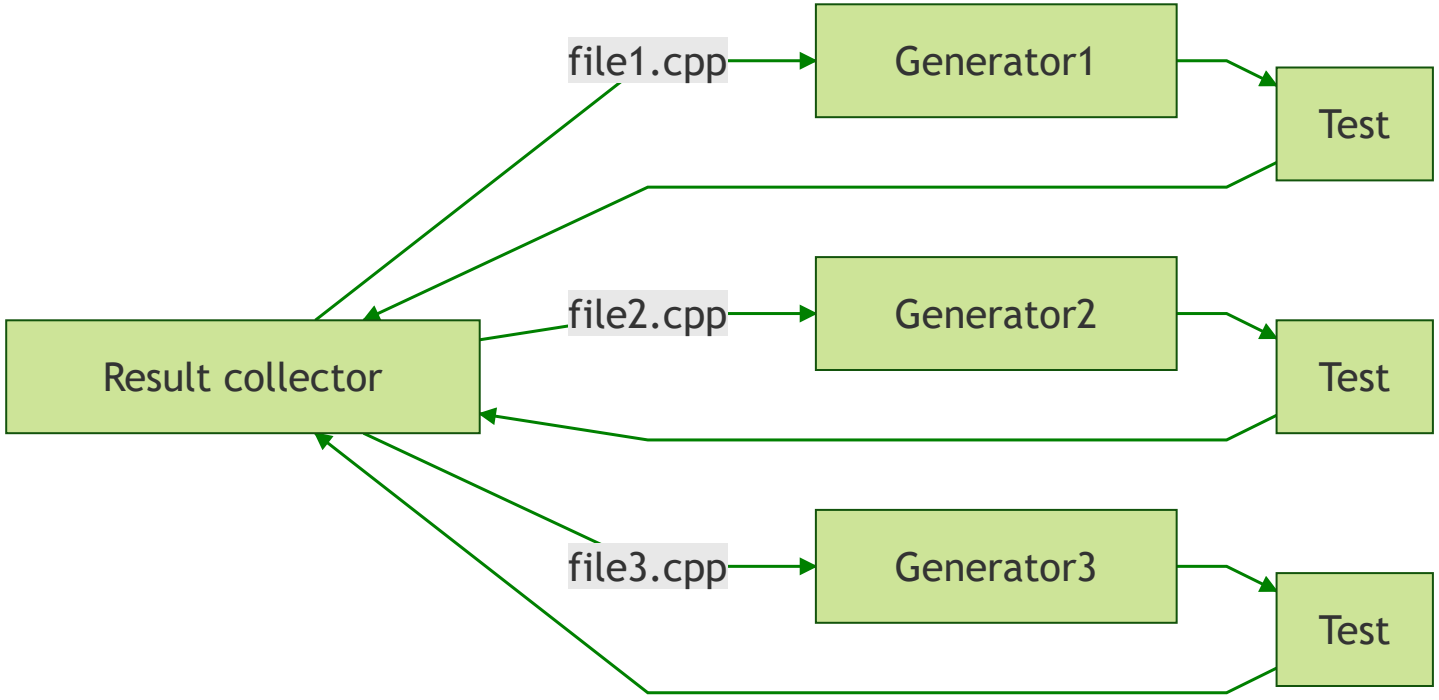
- Mutant schemata

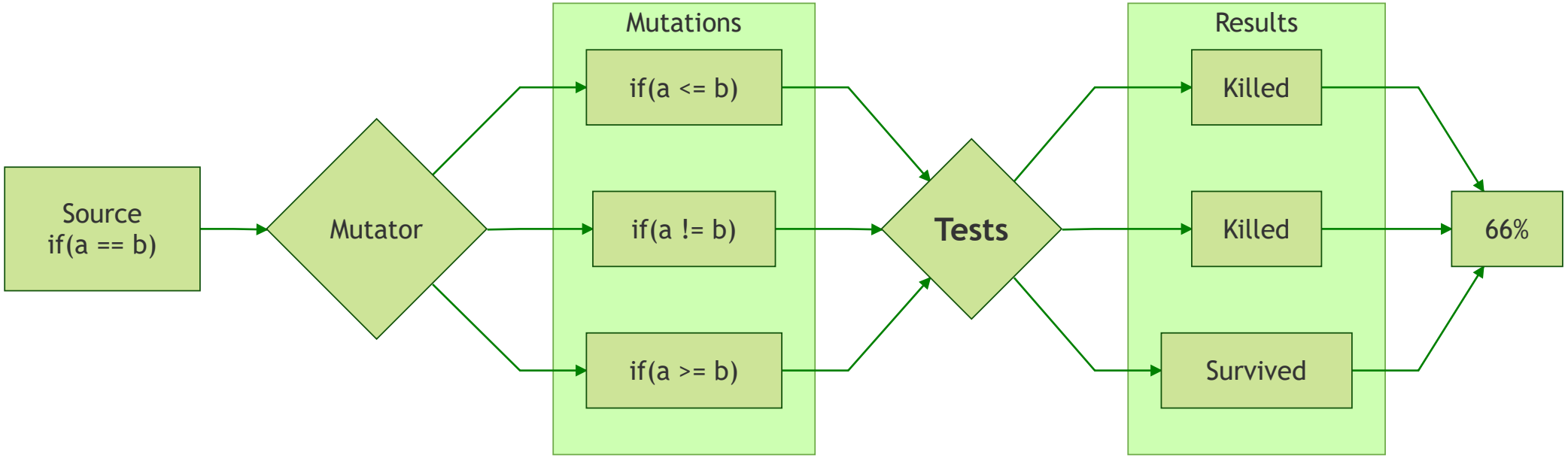
```
auto c = a + b
```

```
auto c = [&]{
   switch(mutation_selector) {
      case 0:
         return a - b;
      case 1:
         return a * b;
      case 2:
         return a / b;
   }
}();
```

Optimizations

- Mutant schemata
- Distributed generation




Source
if(a == b)
Mutator
Mutations
if(a <= b)
if(a != b)
if(a >= b)
Tests
Results
Killed
Killed
Survived
66%

Running tests

Running tests

- All tests

Running tests

- All tests
- Coverage-based

Running tests

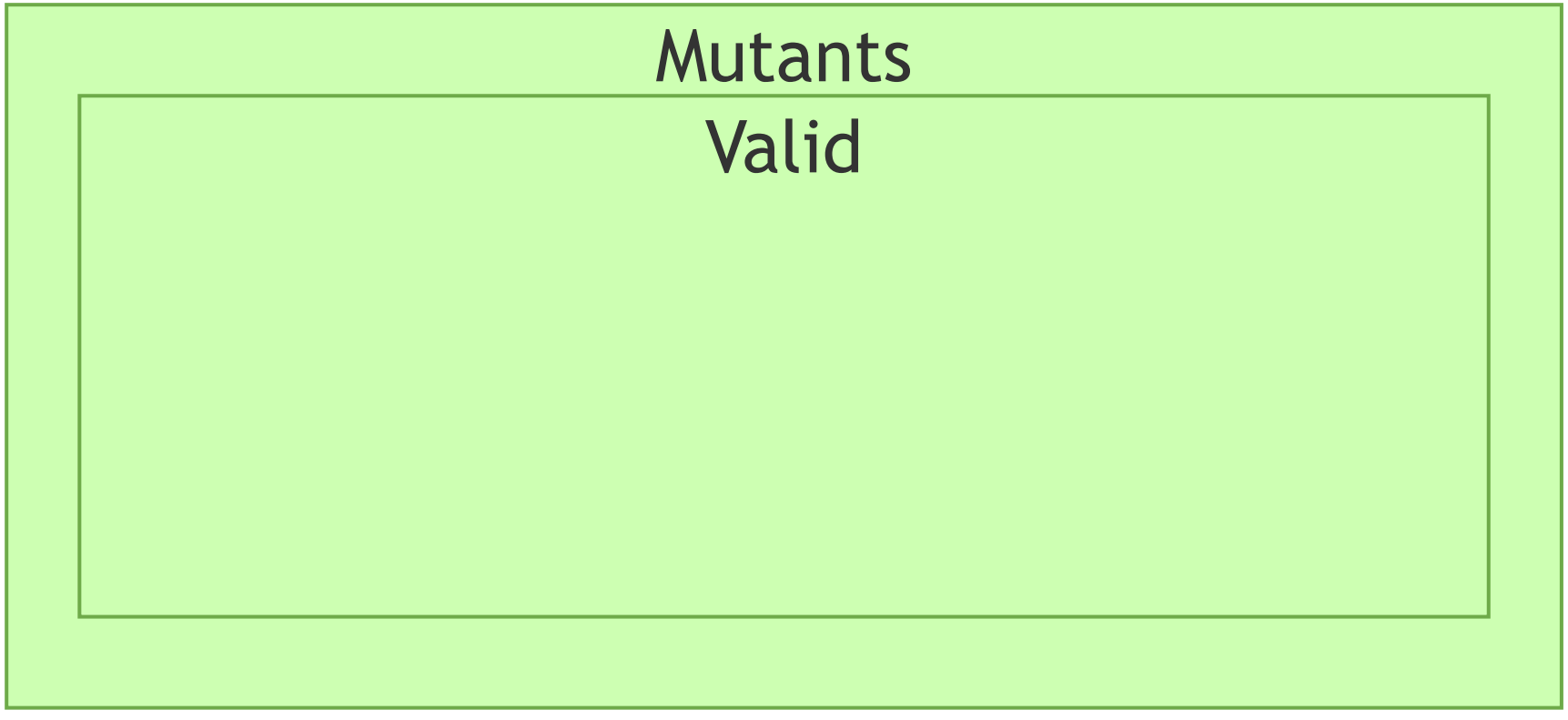
- All tests
- Coverage-based
- Test order

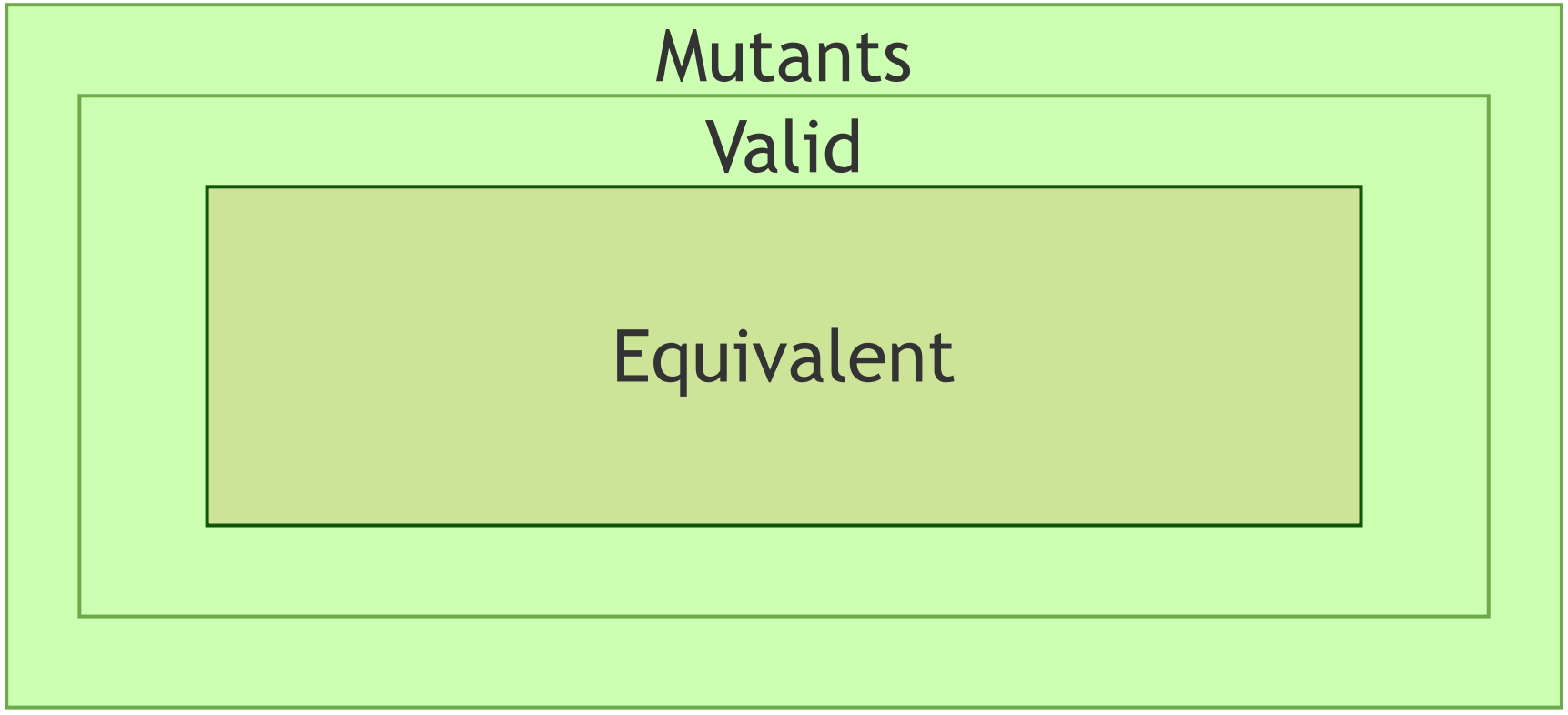
Running tests

- All tests
- Coverage-based
- Test order
- Timing out infinite loops

Mutation Design

Mutants


Mutants
Valid


Mutants
Valid
Equivalent

Equivalence is Hard

```
bool sums_to_zero(vector<int> v) {
    int sum = 0;
    for(int i : v) {
        sum += i;
    }
    return sum == 0;
}
```

Equivalence is Hard

```
bool sums_to_zero(vector<int> v) {
    int sum = 0;
    for(int i : v) {
        sum += i;
    }
    return sum == 0;
}
```

```
bool sums_to_zero(vector<int> v) {
    int sum = 0;
    for(int i : v) {
        sum -= i;
    }
    return sum == 0;
}
```

Initializer-list construction

```
vector<string> names{"Seph", "Cynthia"};
```

Initializer-list construction

```
vector<string> names{"Seph", "Cynthia"};
```

```
vector<string> duplicate_names(20,"John");
```

Initializer-list construction

```
vector<string> names{"Seph", "Cynthia"};
```

```
vector<string> duplicate_names(20,"John");
```

```
const int size = 5;
vector<int> v(size,42);
```

Initializer-list construction

```
vector<string> names{"Seph", "Cynthia"};
```

```
vector<string> duplicate_names(20,"John");
```

```
const int size = 5;
vector<int> v(size,42);
```

```
iota(v.data(),v.data()+size);
```

Initializer-list construction

```
vector<string> names{"Seph", "Cynthia"};
```

```
vector<string> duplicate_names(20,"John");
```

```
const int size = 5;
vector<int> v(size,42);
```

```
iota(v.data(),v.data()+size);
```

```
0,1,2,3,4
```

Initializer-list construction

```
vector<string> names{"Seph", "Cynthia"};
```

```
vector<string> duplicate_names(20,"John");
```

```
const int size = 5;
vector<int> v(size,42);
```

```
const int size = 5;
vector<int> v{size,42};
```

```
iota(v.data(),v.data()+size);
```

```
0,1,2,3,4
```

Initializer-list construction

```
vector<string> names{"Seph", "Cynthia"};
```

```
vector<string> duplicate_names(20,"John");
```

```
const int size = 5;
vector<int> v(size,42);
```

```
const int size = 5;
vector<int> v{size,42};
```

```
iota(v.data(),v.data()+size);
```

```
0,1,2,3,4
```

```
SEGFAULT
```

Equivalence

Invalidity

Equivalence

```
std::vector<int>(2,2);
```

```
std::vector<int>{2,2};
```

Invalidity

Equivalence

```
std::vector<int>(2,2);
```

```
std::vector<int>{2,2};
```

Invalidity

```
std::vector<std::string>{"",""};
```

Ranged for loop

```
vector<string> names{"Seph", "Cynthia"};
```

```
for(auto& name : names) {
   name[0] = std::tolower(name[0]);
}
```

Ranged for loop

```
vector<string> names{"Seph", "Cynthia"};
```

```
for(auto& name : names) {
   name[0] = std::tolower(name[0]);
}
```

```
seph, cynthia
```

Ranged for loop

```
vector<string> names{"Seph", "Cynthia"};
```

```
for(auto& name : names) {
   name[0] = std::tolower(name[0]);
}
```

```
for(auto  name : names) {
   name[0] = std::tolower(name[0]);
}
```

```
seph, cynthia
```

Ranged for loop

```
vector<string> names{"Seph", "Cynthia"};
```

```
for(auto& name : names) {
   name[0] = std::tolower(name[0]);
}
```

```
seph, cynthia
```

```
for(auto  name : names) {
   name[0] = std::tolower(name[0]);
}
```

```
Seph, Cynthia
```

Equivalence

Invalidity

Equivalence

```
for(const auto& x : v) {
    ...
}
```

Invalidity

Equivalence

```
for(const auto& x : v) {
    ...
}
```

Invalidity

```
for(NonCopyable& x : v) {
    ...
}
```

Lambda capture - Intro

```
string msprepender(const string& name) {
   return "Ms. " + name;
}
```

Lambda capture - Intro

```
string msprepender(const string& name) {
   return "Ms. " + name;
}
```

```
vector<string> names{"Seph", "Cynthia"};
transform(begin(names),end(names),
   begin(names), msprepender);
```

Lambda capture - Intro

```
string msprepender(const string& name) {
   return "Ms. " + name;
}
```

```
vector<string> names{"Seph", "Cynthia"};
transform(begin(names),end(names),
   begin(names), msprepender);
```

```
Ms. Seph, Ms. Cynthia
```

Lambda capture - Intro

```
string msprepender(const string& name) {
   return "Ms. " + name;
}
```

```
vector<string> names{"Seph", "Cynthia"};
transform(begin(names),end(names),
   begin(names), msprepender);
```

```
Ms. Seph, Ms. Cynthia
```

```
transform(begin(names),end(names),begin(names),
   [](const string& element) {return "Ms. " + element;});
```

Lambda capture - Intro

```
string msprepender(const string& name) {
   return "Ms. " + name;
}
```

```
vector<string> names{"Seph", "Cynthia"};
transform(begin(names),end(names),
   begin(names), msprepender);
```

```
Ms. Seph, Ms. Cynthia
```

```
transform(begin(names),end(names),begin(names),
   [](const string& element) {return "Ms. " + element;});
```

```
string pre = "Ms. ";
transform(begin(names),end(names),begin(names),
   [=](const string& element) {return pre + element;});
```

Lambda capture - Mutation

```
auto prepender(string prepend) {
   return [=](const string& element) {
      return prepend + element;
   };
}
```

```
vector<string> names{"Seph", "Cynthia"};
transform(begin(names),end(names),begin(names),
   prepender("Ms. "));
```

```
Ms. Seph, Ms. Cynthia
```

Lambda capture - Mutation

```
auto prepender(string prepend) {
   return [=](const string& element) {
      return prepend + element;
   };
}
```

```
auto prepender(string prepend) {
   return [&](const string& element) {
      return prepend + element;
   };
}
```

```
vector<string> names{"Seph", "Cynthia"};
transform(begin(names),end(names),begin(names),
   prepender("Ms. "));
```

```
Ms. Seph, Ms. Cynthia
```

Lambda capture - Mutation

```
auto prepender(string prepend) {
   return [=](const string& element) {
      return prepend + element;
   };
}
```

```
auto prepender(string prepend) {
   return [&](const string& element) {
      return prepend + element;
   };
}
```

```
vector<string> names{"Seph", "Cynthia"};
transform(begin(names),end(names),begin(names),
   prepender("Ms. "));
```

```
Ms. Seph, Ms. Cynthia
```

```
- H►}►Seph,H►}►Cynthia
- SEGFAULT
...
```

Equivalence

Invalidity

Equivalence

```
[]{
    ...
}
```

```
[this]{
    ...
}
```

Invalidity

Equivalence

```
[]{
    ...
}
```

```
[this]{
    ...
}
```

```
std::all_of(v.begin(),v.end(),[=](const auto& e) {...});
```

Invalidity

Equivalence

```
[]{
    ...
}
```

```
[this]{
    ...
}
```

```
std::all_of(v.begin(),v.end(),[=](const auto& e) {...});
```

Invalidity

```
-Werror
```

Compilation As Test

Compilation As Test

```
template <typename Range>
constexpr auto sort(Range&& range)
{
    //Exercise for the reader
}

static_assert(sort(std::array{4,3,2,1}) == std::array{1,2,3,4});
```

Q&A

Tools

- Dextool Mutate: http://tiny.cc/s4k6bz
- MuCPP: http://tiny.cc/p2k6bz
- MULL: http://tiny.cc/g1k6bz
- CCMutator: http://tiny.cc/83k6bz
- XEMU: http://tiny.cc/y5k6bz