Evaluating the Effects of Compiler Optimizations on Mutation Testing at the Compiler IR Level

ISSRE'16 – Ottawa, Canada
Oct 25th, 2016

Farah Hariri †, August Shi †, Hayes Converse ‡, Sarfraz Khurshid ‡, Darko Marinov †
†University of Illinois at Urbana-Champaign, ‡The University of Texas at Austin

CNS-1239498, CCF-1319688, CCF-1409423, CCF-1421503, CCF-1438982, and CCF-1439957

Background

MUTATION TESTING

Background: Mutation Testing

- Definition: Mutation testing is a technique for evaluating the quality of test suites

Mutation Testing As A Black Box

- Definition: Mutation testing is a technique for evaluating the quality of test suites
- As a black box, it works as follows:

Mutation Testing As A Black Box

- Definition: Mutation testing is a technique for evaluating the quality of test suites
- As a black box, it works as follows:

- Two key concerns:
  - Interpretation of the mutation score
  - Speed of computing the mutation score

Steps of Mutation Testing

- Mutation testing proceeds in two steps:
- Step 1: Mutant Generation

Steps of Mutation Testing

- Mutation testing proceeds in two steps:
- Step 1: Mutant Generation
  - Mutant: modified version of the original program obtained by applying a mutation operator

Steps of Mutation Testing

- Mutation testing proceeds in two steps:
- Step 1: Mutant Generation
  - Mutant: modified version of the original program obtained by applying a mutation operator
  - Mutation Operator: program transformation that introduces a small syntactic change to the original program

Steps of Mutation Testing

- Mutation testing proceeds in two steps:
- Step 1: Mutant Generation
  - Mutant: modified version of the original program obtained by applying a mutation operator
  - Mutation Operator: program transformation that introduces a small syntactic change to the original program

Example!

Mutant Generation Example

- **Example from the program ‘cut’**

Original code

```
%r12 = and i8 %r11, 1
%r13 = icmp eq i8 %r12, 0
```

Mutant Generation Example

- **Example from the program ‘cut’**

Original code

```
%r12 = and i8 %r11, 1
%r13 = icmp eq i8 %r12, 0
```

ROR

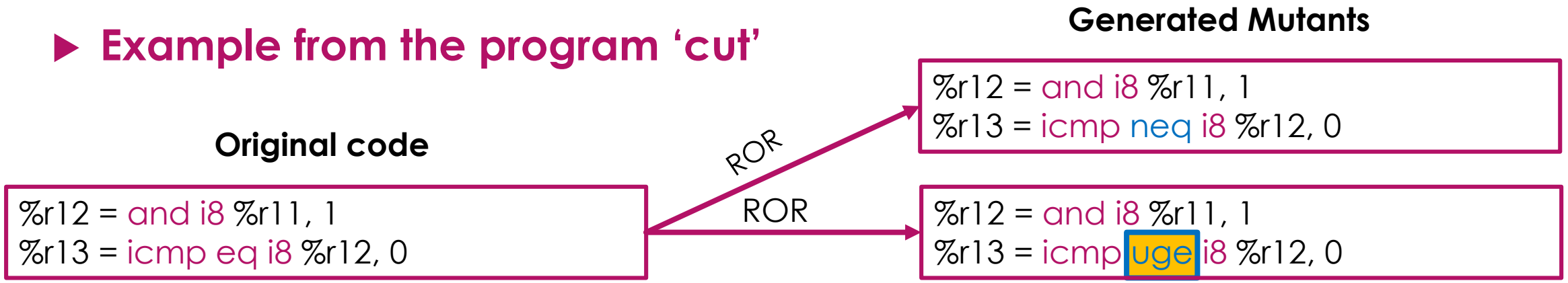
Generated Mutants

```
%r12 = and i8 %r11, 1
%r13 = icmp neq i8 %r12, 0
```

ROR: Relational Operator Replacement

Mutant Generation Example

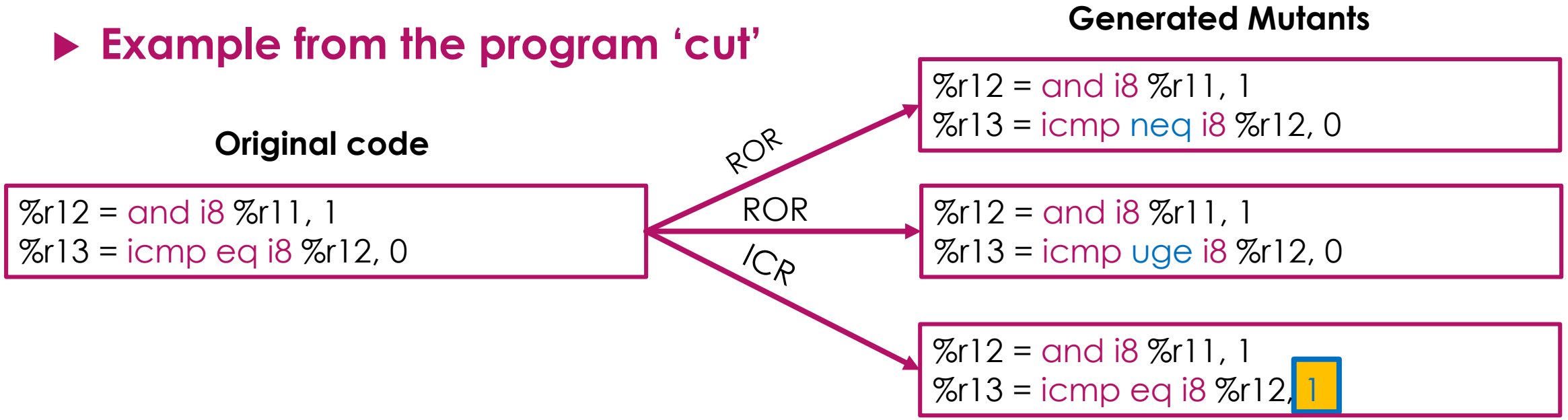
- **Example from the program ‘cut’**



ROR: Relational Operator Replacement

Mutant Generation Example

- **Example from the program 'cut'**

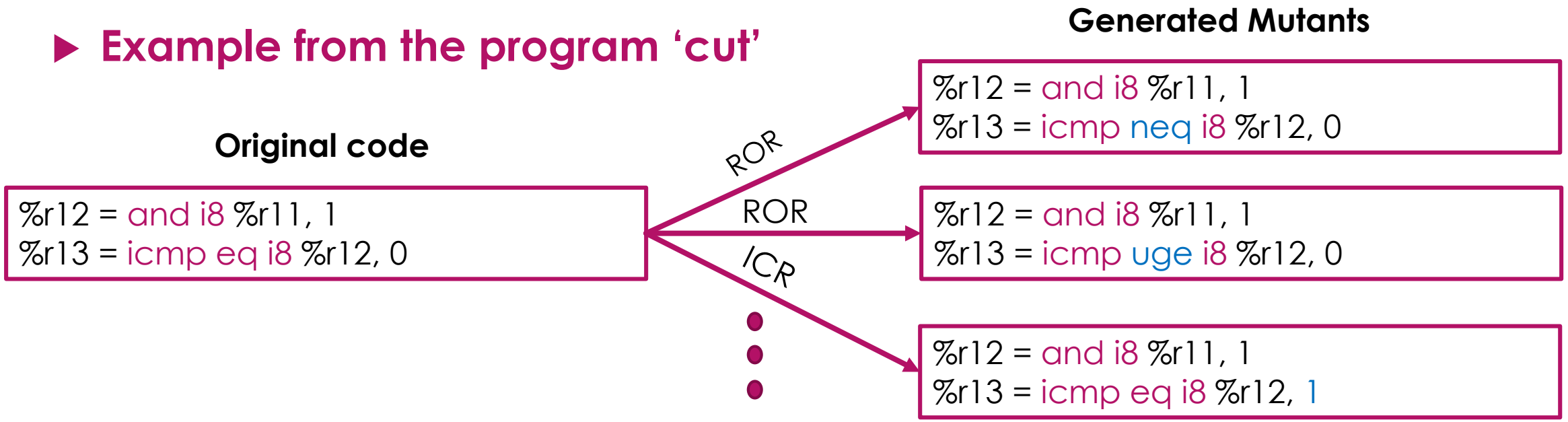


ROR: Relational Operator Replacement

ICR : Integer Constant Replacement

Mutant Generation Example

- **Example from the program 'cut'**



ROR: Relational Operator Replacement

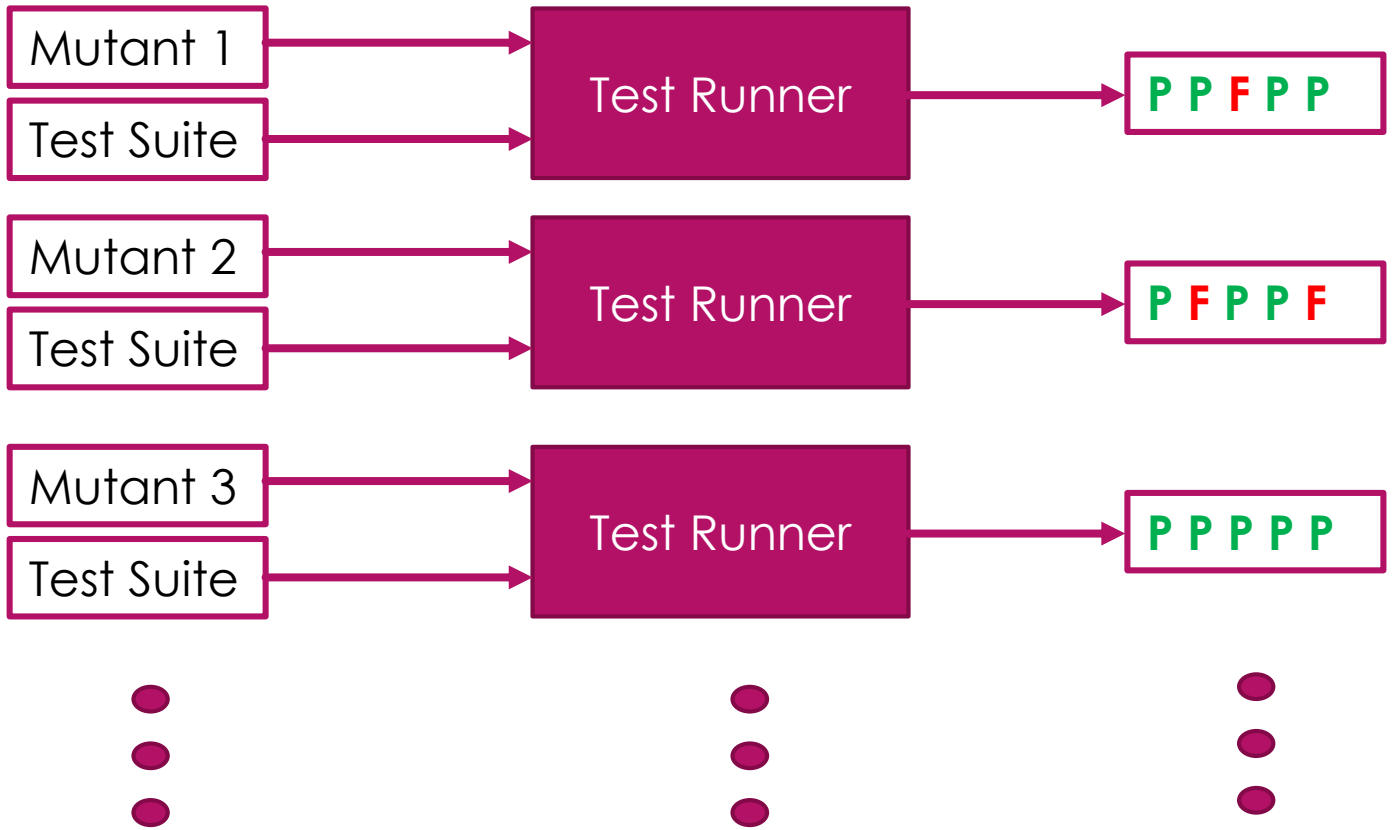
ICR : Integer Constant Replacement

Steps of Mutation Testing

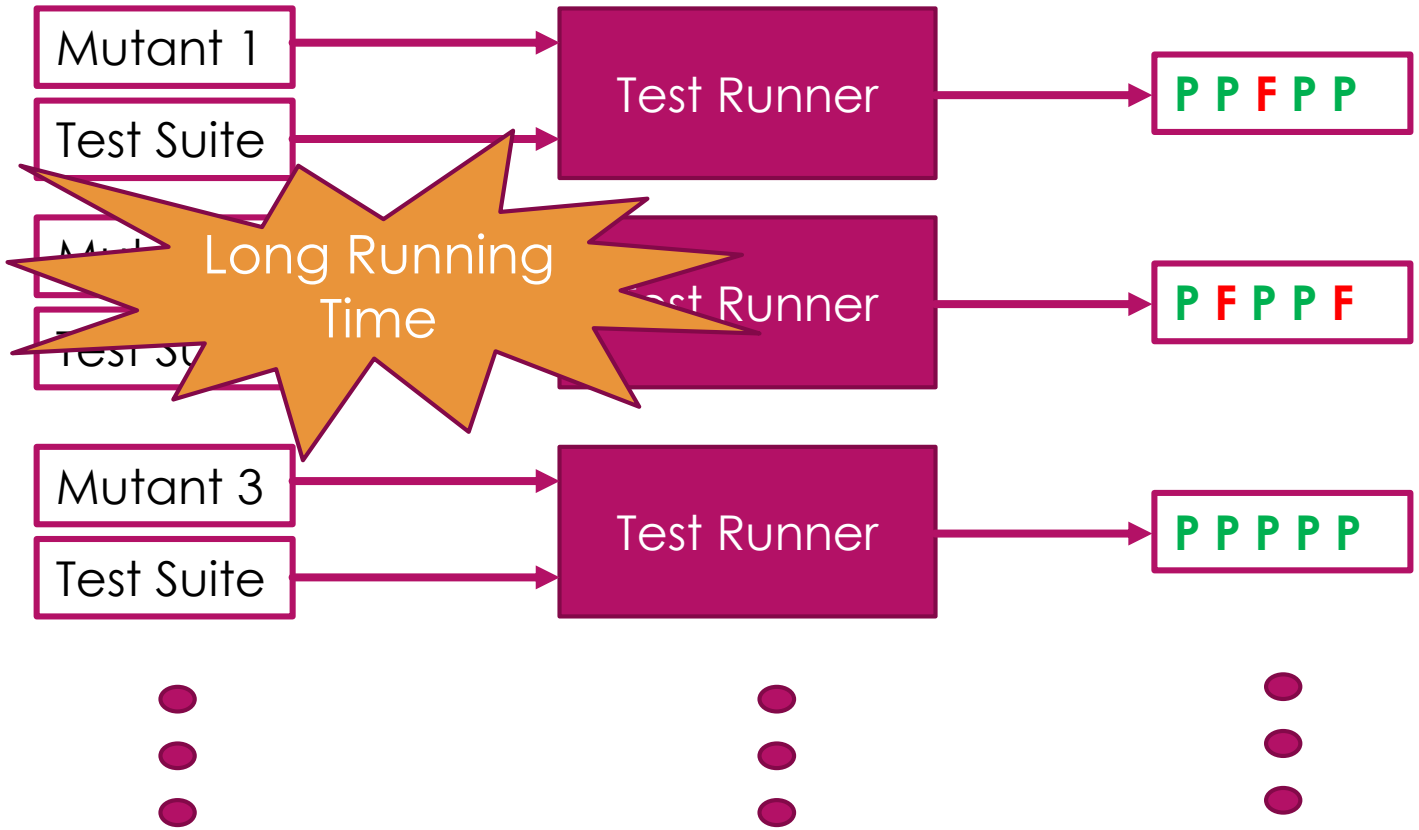
- Mutation testing proceeds in two steps:
- Step 1: Mutant Generation
  - Mutant: modified version of the program obtained by applying a mutation operator
  - Mutation Operator: program transformation that introduces a small syntactic change to the original code
- **Step 2: Running the test suite for every single mutant**

Running The Test Suite Example

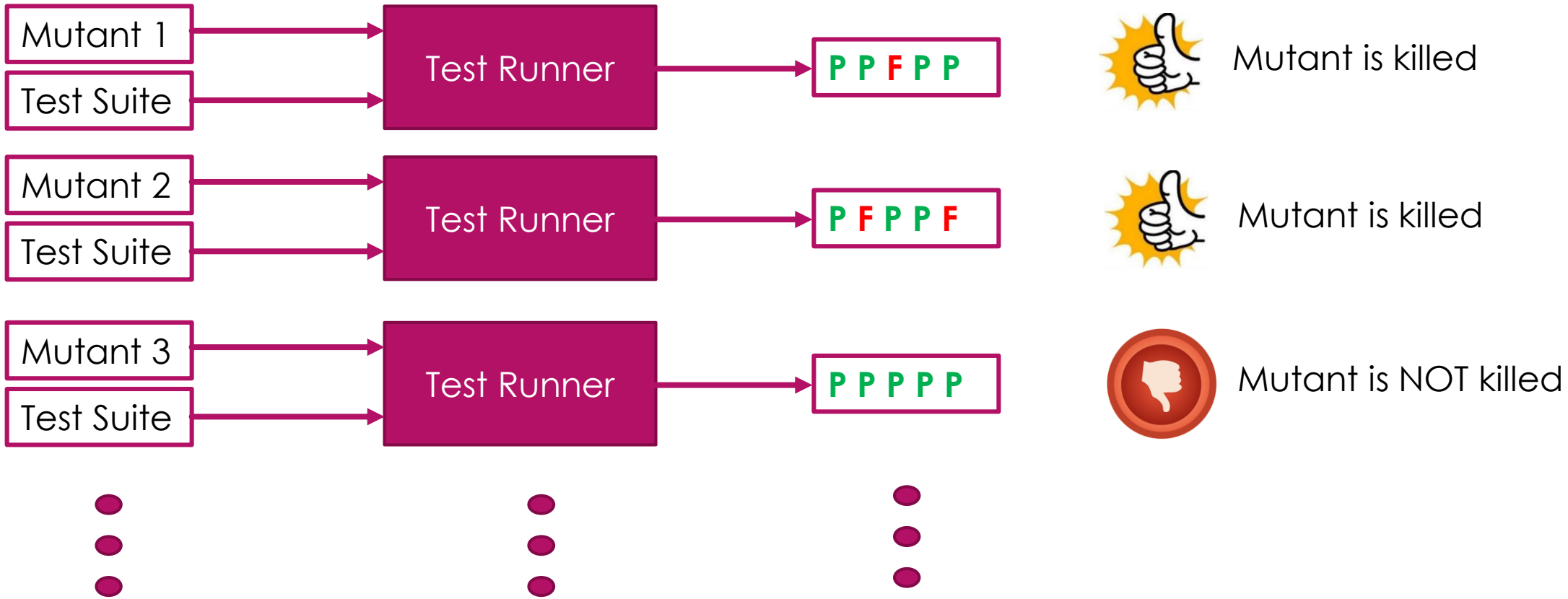
Running The Test Suite Example



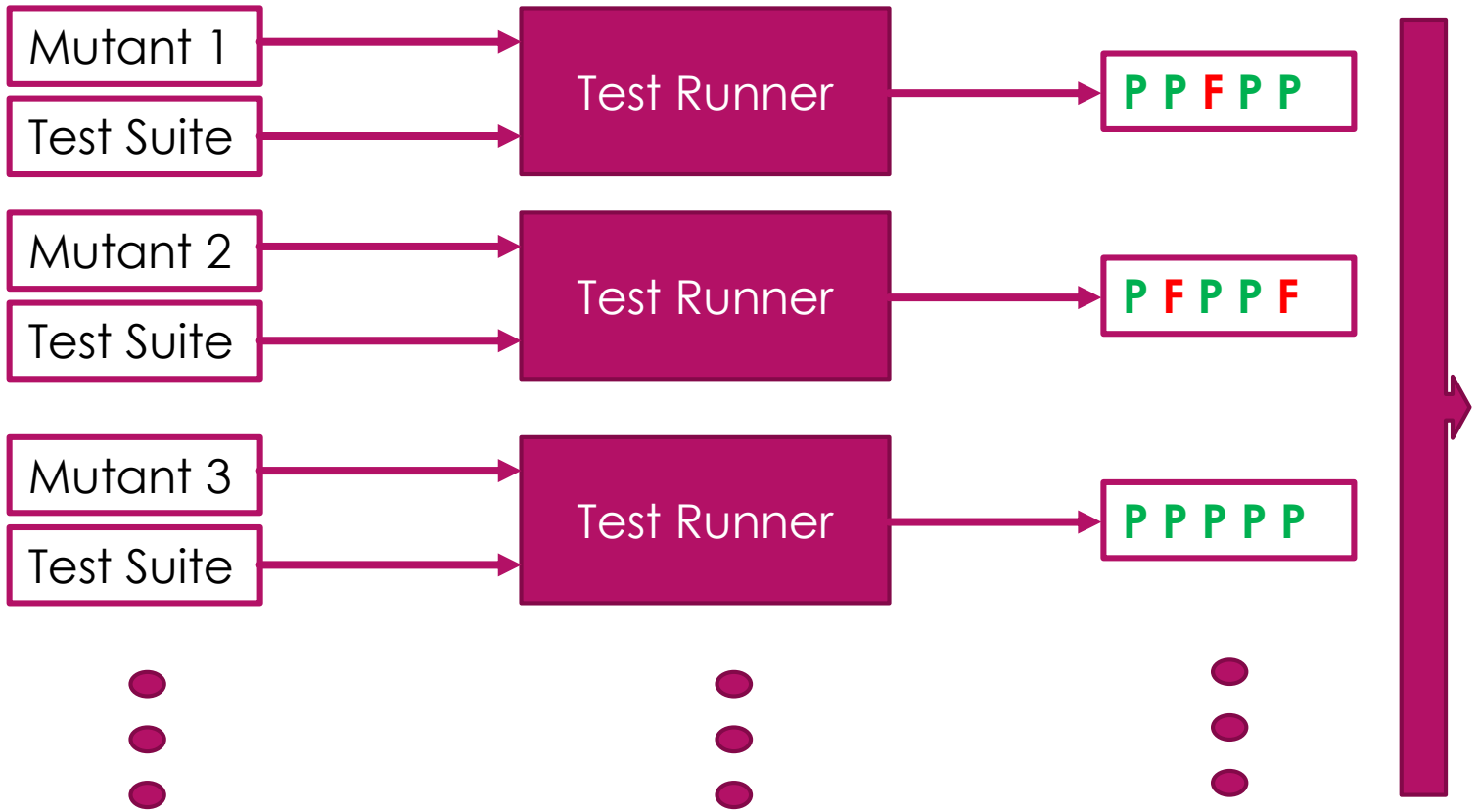
Running The Test Suite Example



Running The Test Suite Example



Mutation Score



$$mutation\ score(\%) = \frac{killed\ mutants}{all\ generated\ mutants}$$

Challenges of Mutation Testing

Challenges of Mutation Testing

- Equivalent and Duplicated Mutants
- Long Running Time

Challenges of Mutation Testing

- Equivalent and Duplicated Mutants
  - Equivalent Mutant: is a mutant that is syntactically different but semantically equivalent to the original program

Challenges of Mutation Testing

- Equivalent and Duplicated Mutants
  - Equivalent Mutant: is a mutant that is syntactically different but semantically equivalent to the original program
  - Duplicated Mutants: are two or more mutants that are syntactically different but semantically equivalent to each another (but not to the original program)

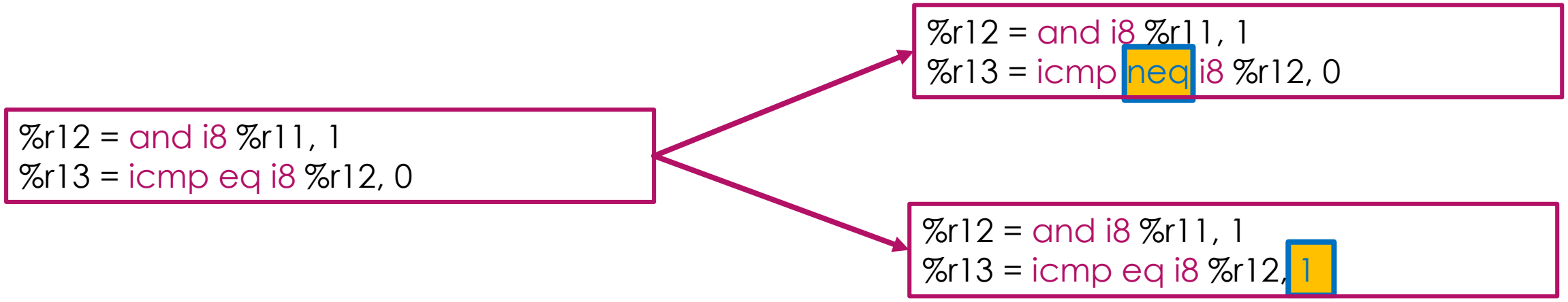
Challenges of Mutation Testing

- Equivalent and Duplicated Mutants
  - Equivalent Mutant: is a mutant that is syntactically different but semantically equivalent to the original program
  - Duplicated Mutants: are two or more mutants that are syntactically different but semantically equivalent to each another (but not to the original program)

Example!

Duplicated Mutants Example

- Checking that a boolean is not equal to zero is semantically equivalent to checking that a boolean is equal to one

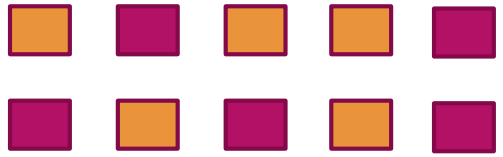


Challenges of Mutation Testing

- Equivalent and Duplicated Mutants
  - Equivalent Mutant: is a mutant that is syntactically different but semantically equivalent to the original program
  - Duplicated Mutants: are two or more mutants that are syntactically different but semantically equivalent to each another (but not to the original program)
  - **Equivalent and duplicated mutants can skew the mutation score and increase the running time**

Skewing Mutation Score Example

- 10 mutants
- Original mutation score = 5/10 = 50 %



Killed

Not Killed

Scenario	Mutation Score	Running Time

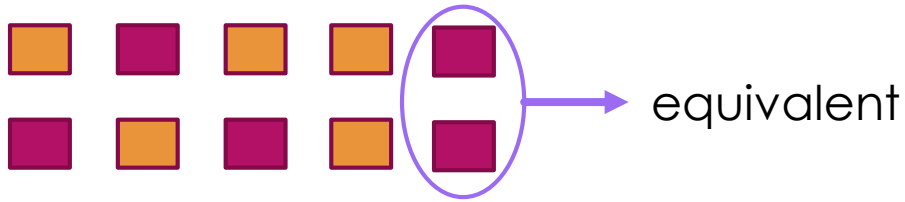
Skewing Mutation Score Example

- 10 mutants
- Original mutation score = 5/10 = 50 %

Killed

Not Killed

Mutation score = 5/8 = 62.5%



Scenario	Mutation Score	Running Time

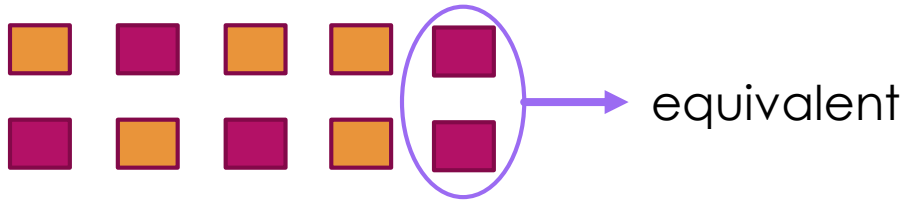
Skewing Mutation Score Example

- 10 mutants
- Original mutation score = 5/10 = 50 %

Killed

Not Killed

- **Mutation score = 5/8 = 62.5%**



Scenario	Mutation Score	Running Time
Equivalent Mutants	Deflate	Increase

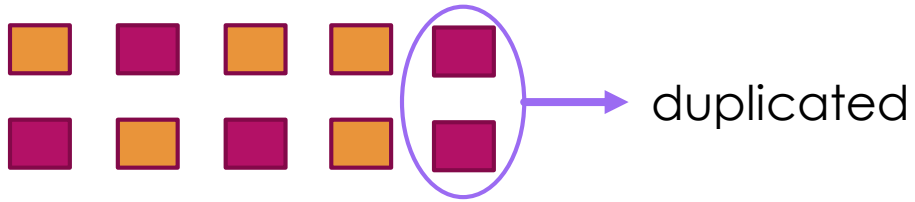
Skewing Mutation Score Example

- 10 mutants
- Original mutation score = 5/10 = 50 %

Killed

Not Killed

- **Mutation score = 5/9 = 55.5 %**



Scenario	Mutation Score	Running Time
Equivalent Mutants	Deflate	Increase

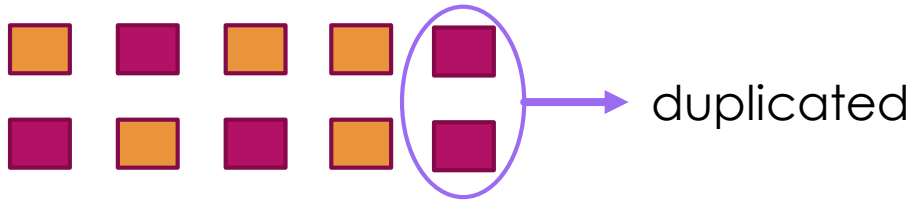
Skewing Mutation Score Example

- 10 mutants
- Original mutation score = 5/10 = 50 %

Killed

Not Killed

- **Mutation score = 5/9 = 55.5 %**



Scenario	Mutation Score	Running Time
Equivalent Mutants	Deflate	Increase
Duplicated Mutants (not killed)	Deflate	Increase

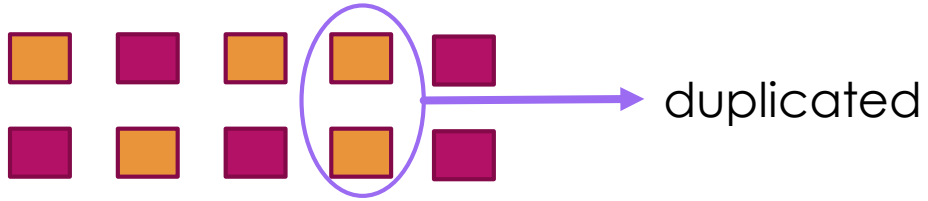
Skewing Mutation Score Example

- 10 mutants
- Original mutation score = 5/10 = 50 %

Killed

Not Killed

Mutation score = 4/9 = 44.4 %



Scenario	Mutation Score	Running Time
Equivalent Mutants	Deflate	Increase
Duplicated Mutants (not killed)	Deflate	Increase

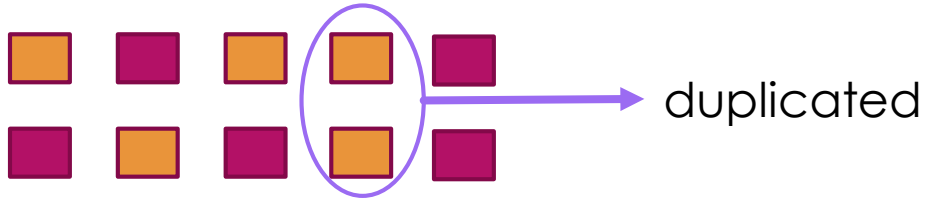
Skewing Mutation Score Example

- 10 mutants
- Original mutation score = 5/10 = 50 %

Killed

Not Killed

Mutation score = 4/9 = 44.4 %



Scenario	Mutation Score	Running Time
Equivalent Mutants	Deflate	Increase
Duplicated Mutants (not killed)	Deflate	Increase
Duplicated Mutants (killed)	Inflate	Increase

Challenges of Mutation Testing

- Equivalent and Duplicated Mutants
  - Equivalent Mutant: is a mutant that is syntactically different but semantically equivalent to the original program
  - Duplicated Mutants: are two or more mutants that are syntactically different but semantically equivalent to each another (but not to the original program)
  - Equivalent and duplicated mutants can skew the mutation score and increase the running time
- **Long Running Time**

Mutation Testing And Compiler Optimizations

Mutation Testing at the IR Level

- Traditionally mutant generation (step 1) is applied on the source code

Mutation Testing at the IR Level

- Traditionally mutant generation (step 1) is applied on the source code
- Compiler Optimizations: Semantic preserving transformation applied automatically by the compiler to improve performance

Mutation Testing at the IR Level

- Traditionally mutant generation (step 1) is applied on the source code
- Compiler Optimizations: Semantic preserving transformation applied automatically by the compiler to improve performance
- Mutant generation has been recently applied on the compiler IR level

(Schulte, PhD thesis'14; Sousa et al., CODES+ISSS'12)

  - Advantage: One tool for many source languages
  - E.g., LLVM supports C, C++, Objective-C, CUDA, …

Problem Statement

Transformation	Semantic preserving	Effect of other compiler optimizations
Compiler Optimizations	Yes	
Mutant Generation	No	

Problem Statement

Transformation	Semantic preserving	Effect of other compiler optimizations
Compiler Optimizations	Yes	
Mutant Generation	No	

Problem Statement

- **What are the effects of compiler optimizations on mutation testing at the IR level?**

Problem Statement

- **What are the effects of compiler optimizations on mutation testing at the IR level?**
- Effects can be on :
  - Number of mutants generated (and therefore speed of computing the mutation score)
  - Mutation Score

Problem Statement

- **What are the effects of compiler optimizations on mutation testing at the IR level?**
- Effects can be on :
  - Number of mutants generated (and therefore speed of computing the mutation score)
  - Mutation Score

Example!

Loop Peeling Example

Unoptimized Code

```
<label>:2
... loop header ...
<label>:5
%r12 = and i8 %r11, 1
%r13 = icmp eq i8 %r12, 0
br label %2
```

Loop Peeling Example

Unoptimized Code

```
<label>:2
… loop header …
<label>:5
%r12 = and i8 %r11, 1
%r13 = icmp eq i8 %r12, 0
br label %2
```

Optimized Code

```
%r12 = and i8 %r11, 1
%r13 = icmp eq i8 %r12, 0

<label>:2
… loop header mod …
<label>:5
%r12 = and i8 %r11, 1
%r13 = icmp eq i8 %r12, 0
br label %2
```

Loop Peeling Example

Unoptimized Code

```
<label>:2
... loop header ...
<label>:5
%r12 = and i8 %r11, 1
%r13 = icmp eq i8 %r12, 0
br label %2
```

Optimized Code

```
%r12 = and i8 %r11, 1
%r13 = icmp eq i8 %r12, 0

<label>:2
... loop header mod ...
<label>:5
%r12 = and i8 %r11, 1
%r13 = icmp eq i8 %r12, 0
br label %2
```

Loop Peeling Example

Unoptimized Code

```
<label>:2
... loop header ...
<label>:5
%r12 = and i8 %r11, 1
%r13 = icmp eq i8 %r12, 0
br label %2
```

Optimized Code

```
%r12 = and i8 %r11, 1
%r13 = icmp eq i8 %r12, 0

<label>:2
... loop header mod ...
<label>:5
%r12 = and i8 %r11, 1
%r13 = icmp eq i8 %r12, 0
br label %2
```

Research questions

- RQ1. How do compiler optimizations affect the number of generated mutants?
- RQ2. How do compiler optimizations affect the number of equivalent and duplicated mutants?
- RQ3. How do compiler optimizations affect the mutation score?
- RQ4. How do these effects vary with the class of mutation operators applied?

Experimental Setup

Evaluated Mutation Operators

- **AOR** replaces every arithmetic operator with another arithmetic operator {add, sub, mul, udiv, …}

Evaluated Mutation Operators

- **AOR** replaces every arithmetic operator with another arithmetic operator {add, sub, mul, udiv, …}
- **LCR** replaces every logical connector with another logical connector {and, or, xor}

Evaluated Mutation Operators

- **AOR** replaces every arithmetic operator with another arithmetic operator {add, sub, mul, udiv, …}
- **LCR** replaces every logical connector with another logical connector {and, or, xor}
- **ROR** replaces every relational operator with another relational operator {eq, neq, uge, ult, sgt, …}

Evaluated Mutation Operators

- **AOR** replaces every arithmetic operator with another arithmetic operator {add, sub, mul, udiv, …}
- **LCR** replaces every logical connector with another logical connector {and, or, xor}
- **ROR** replaces every relational operator with another relational operator {eq, neq, uge, ult, sgt, …}
- **ICR** replaces every integer constant c with a different value from the set
  {−1, 0, 1, c − 1, c + 1}

Subjects and Optimization Levels

- Subjects: 18 programs from the *Coreutils* library
  - E.g., chmod, chown, cut, head, rm, rmdir, tr, wc, …

Subjects and Optimization Levels

- Subjects: 18 programs from the *Coreutils* library
  - E.g., chmod, chown, cut, head, rm, rmdir, tr, wc, …
- Programs come with their individual test suites

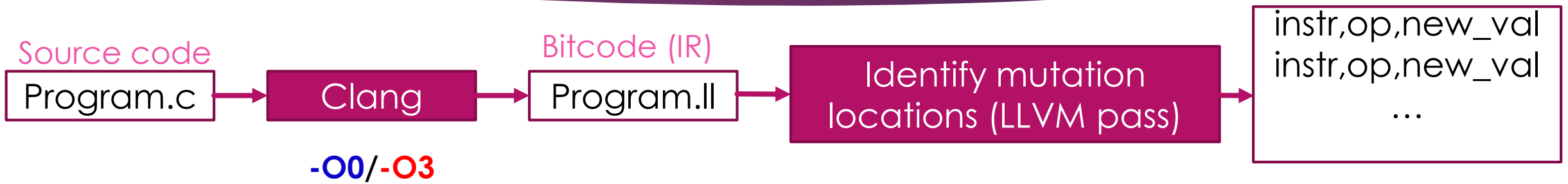
Subjects and Optimization Levels

- Subjects: 18 programs from the *Coreutils* library
  - E.g., chmod, chown, cut, head, rm, rmdir, tr, wc, …
- Programs come with their individual test suites
- LLVM 3.8.1
  - -O0: without optimizations
  - -O3: with optimizations

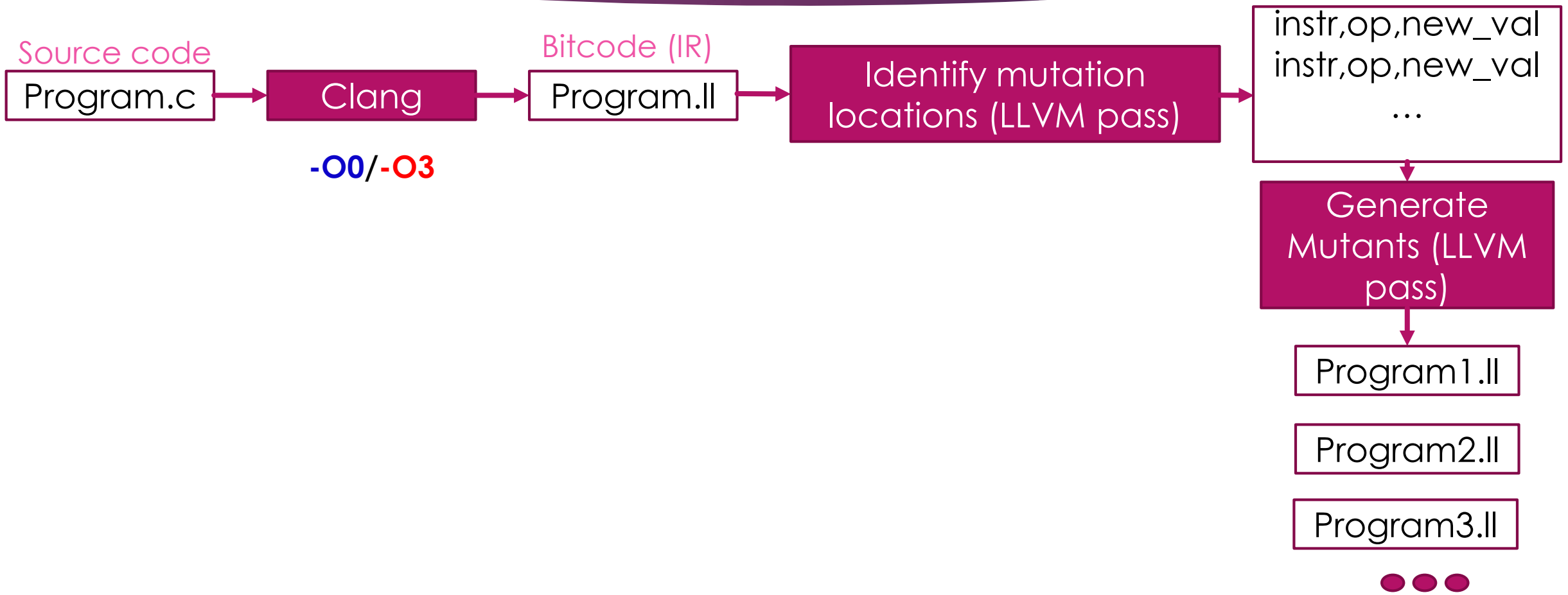
Implementation Pipeline

Implementation

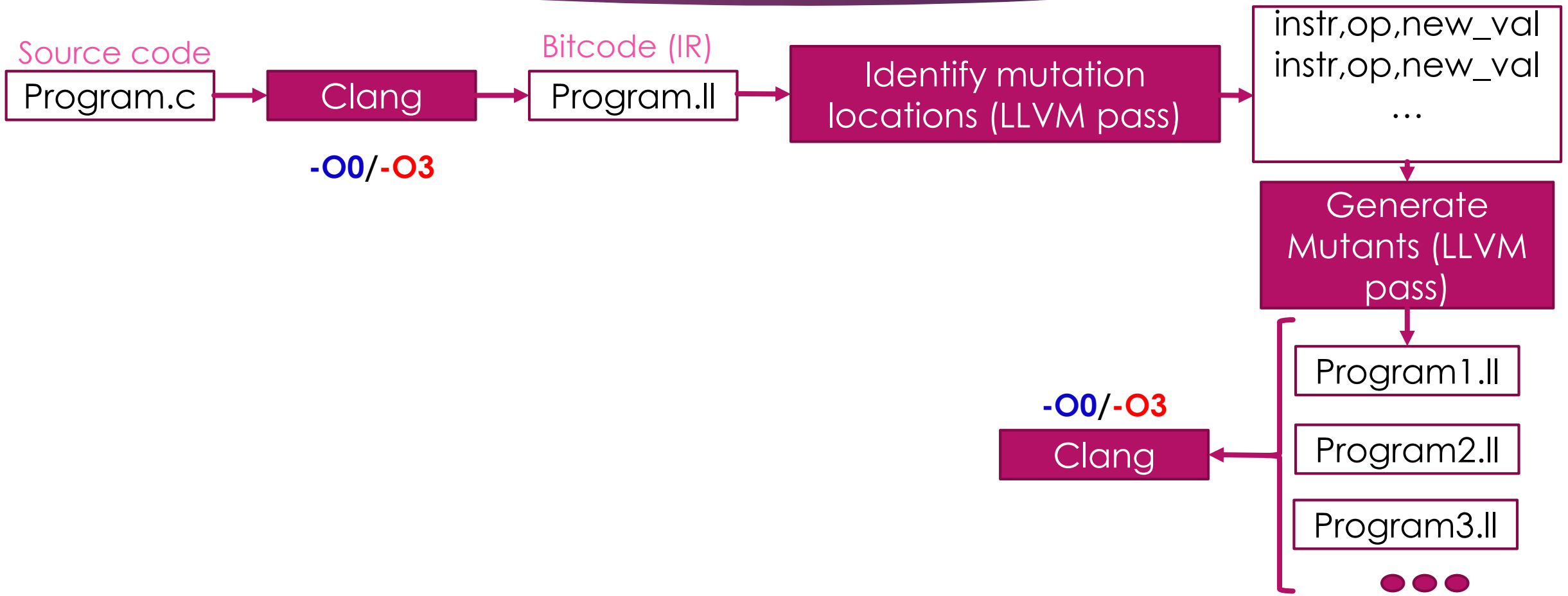
Implementation



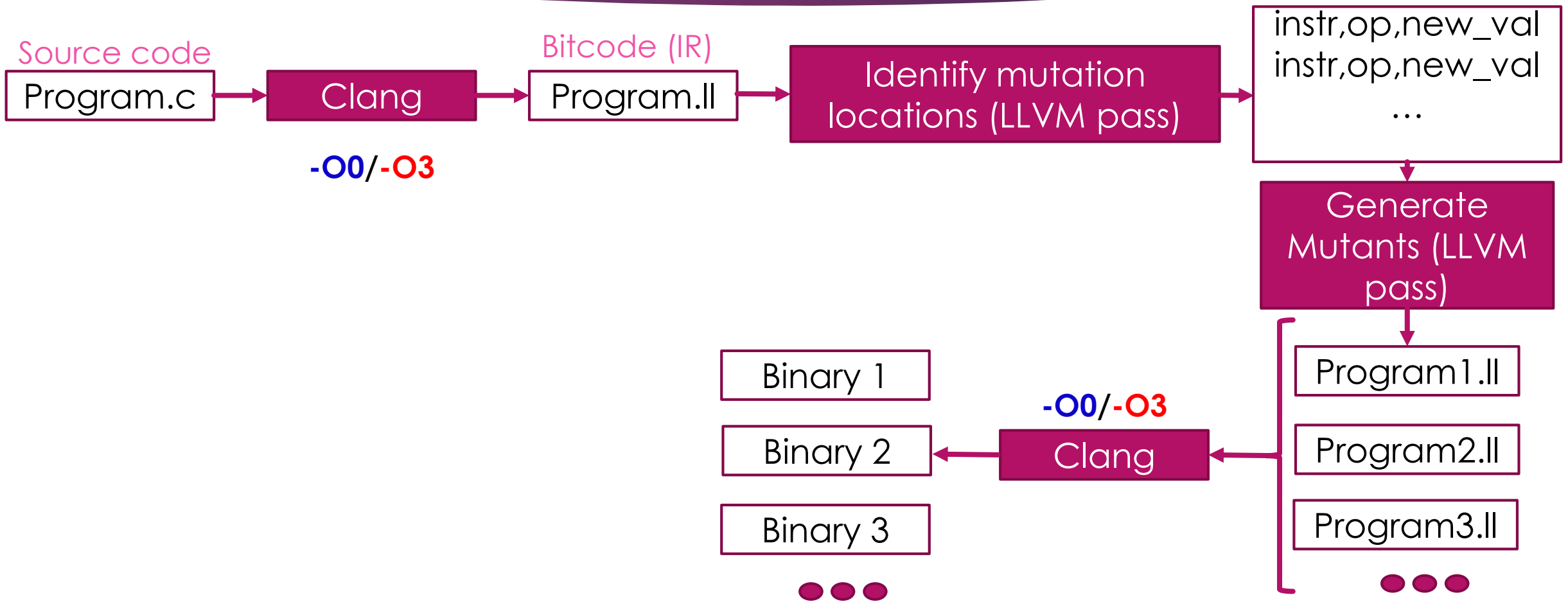
Implementation



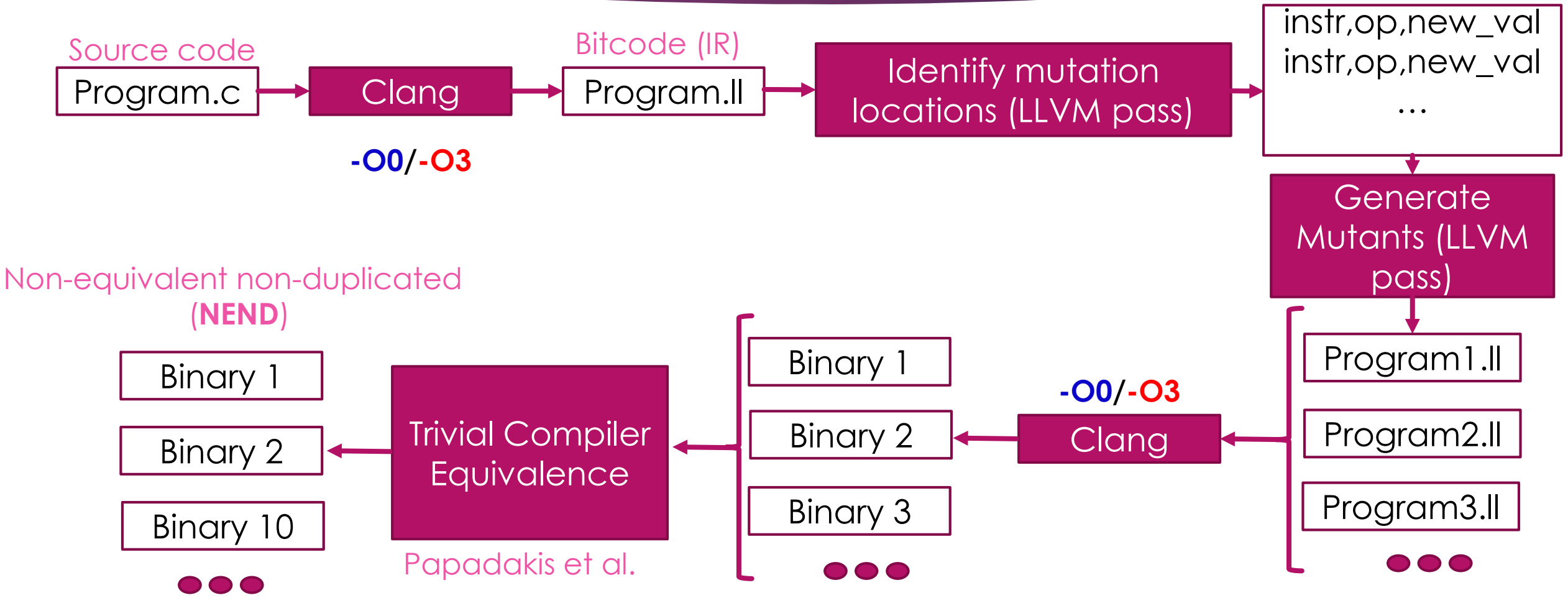
Implementation



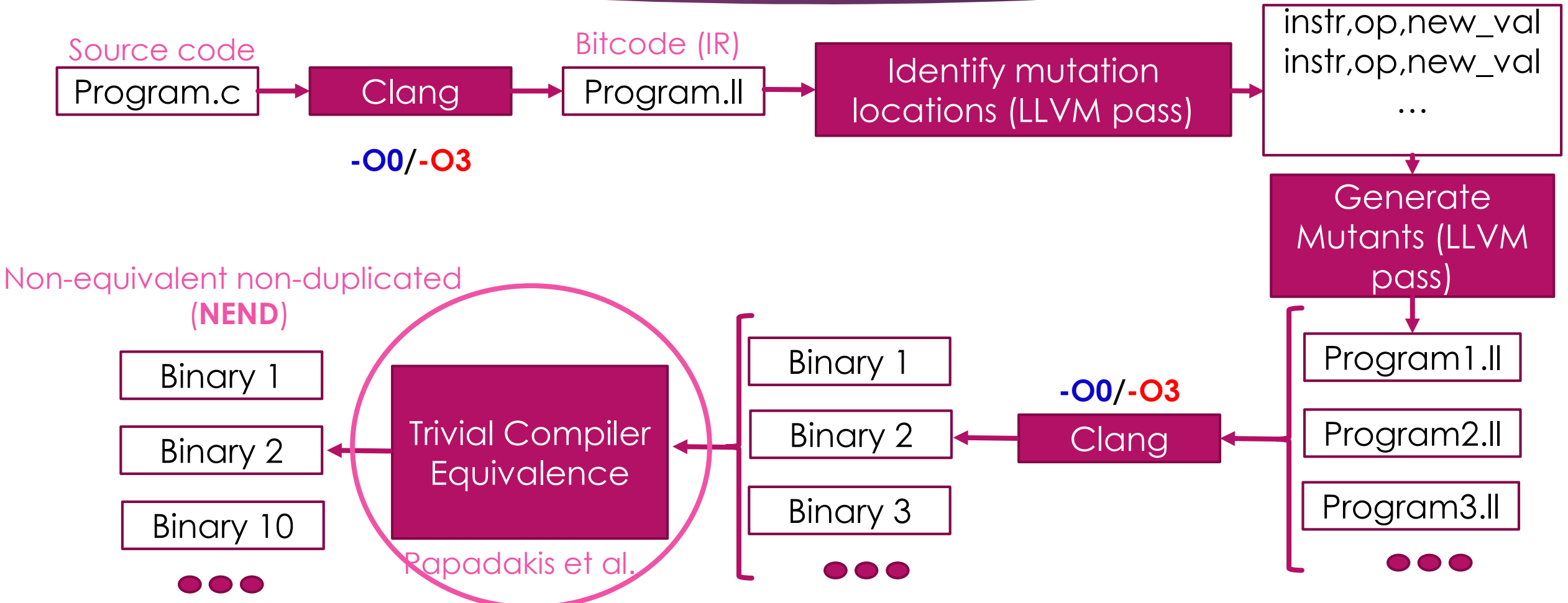
Implementation



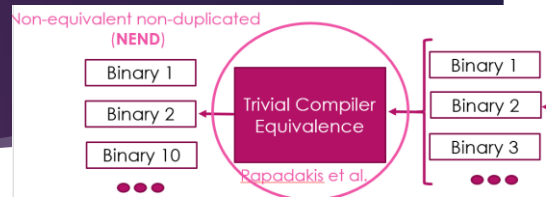
Implementation

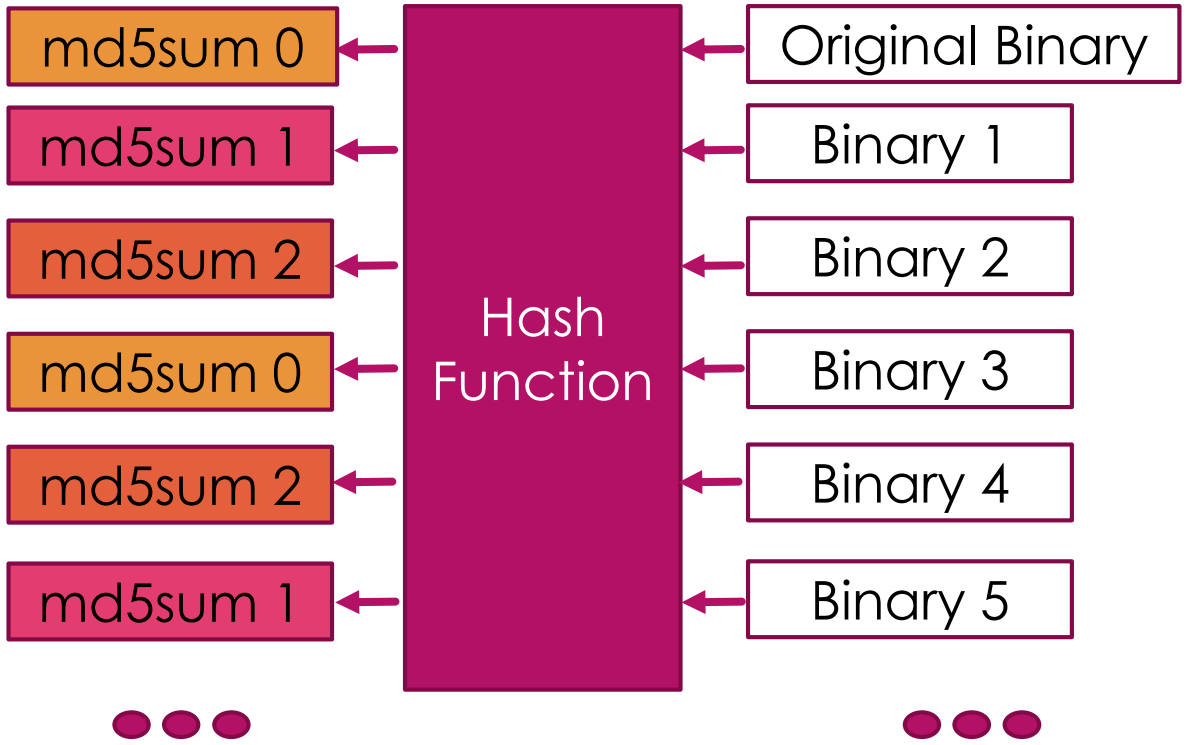


Implementation

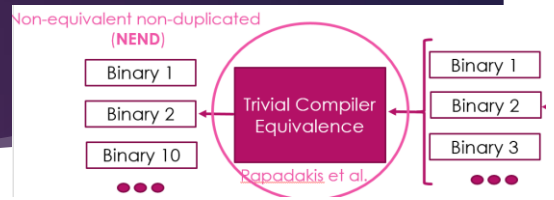


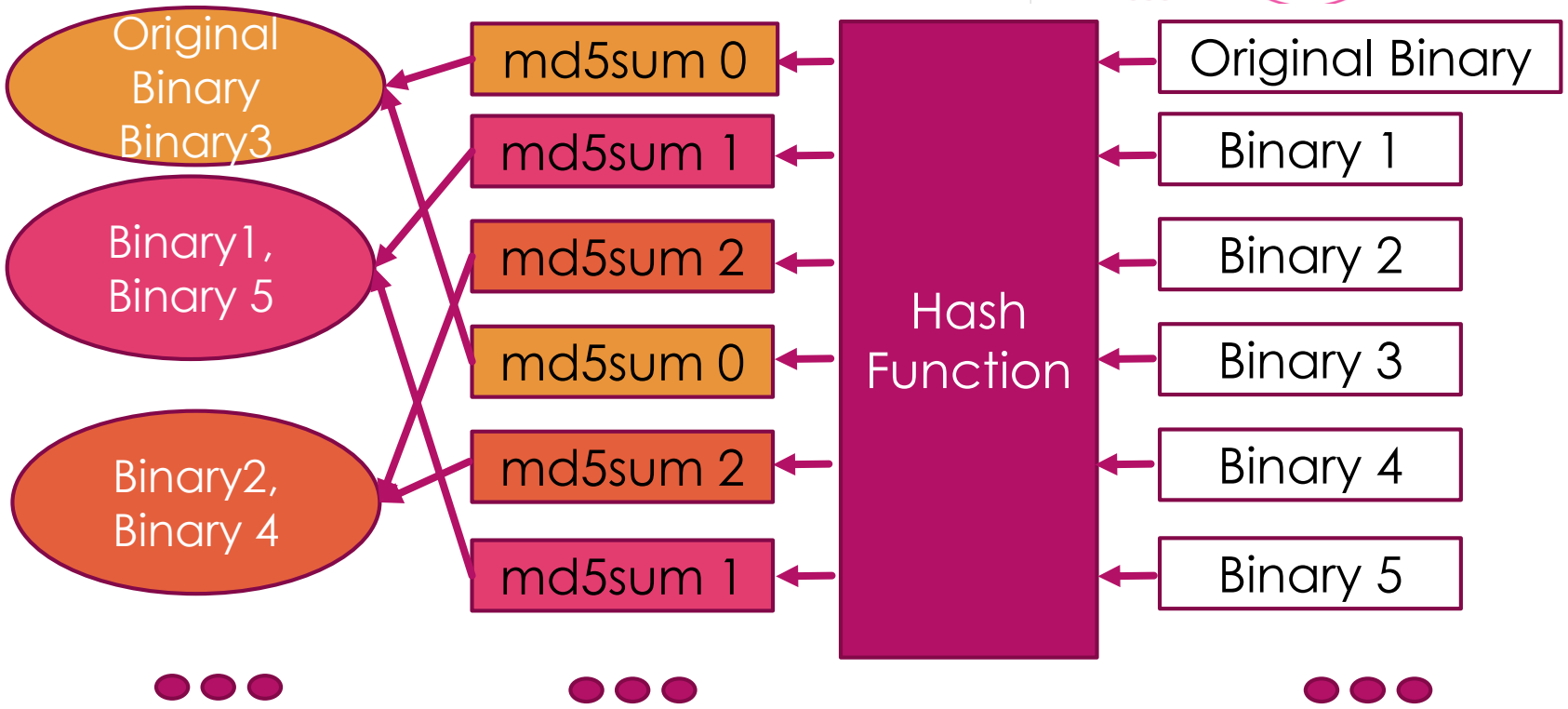
Trivial Compiler Equivalence





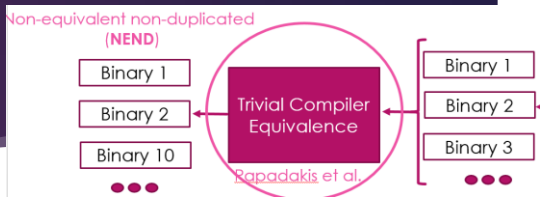
Trivial Compiler Equivalence

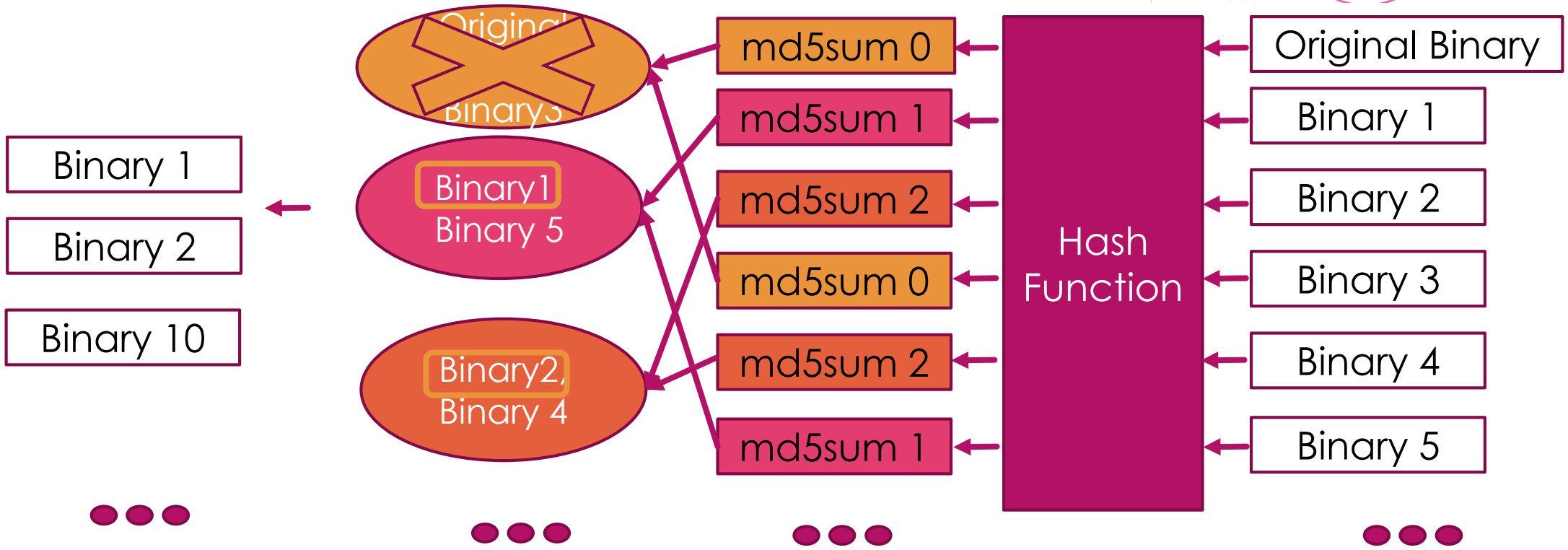




Equivalence Classes

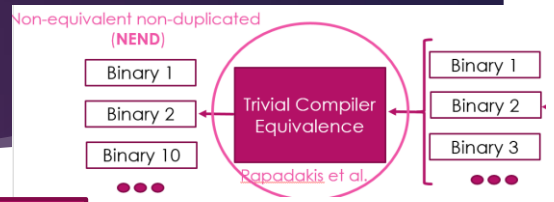
Trivial Compiler Equivalence





Equivalence Classes

Trivial Compiler Equivalence



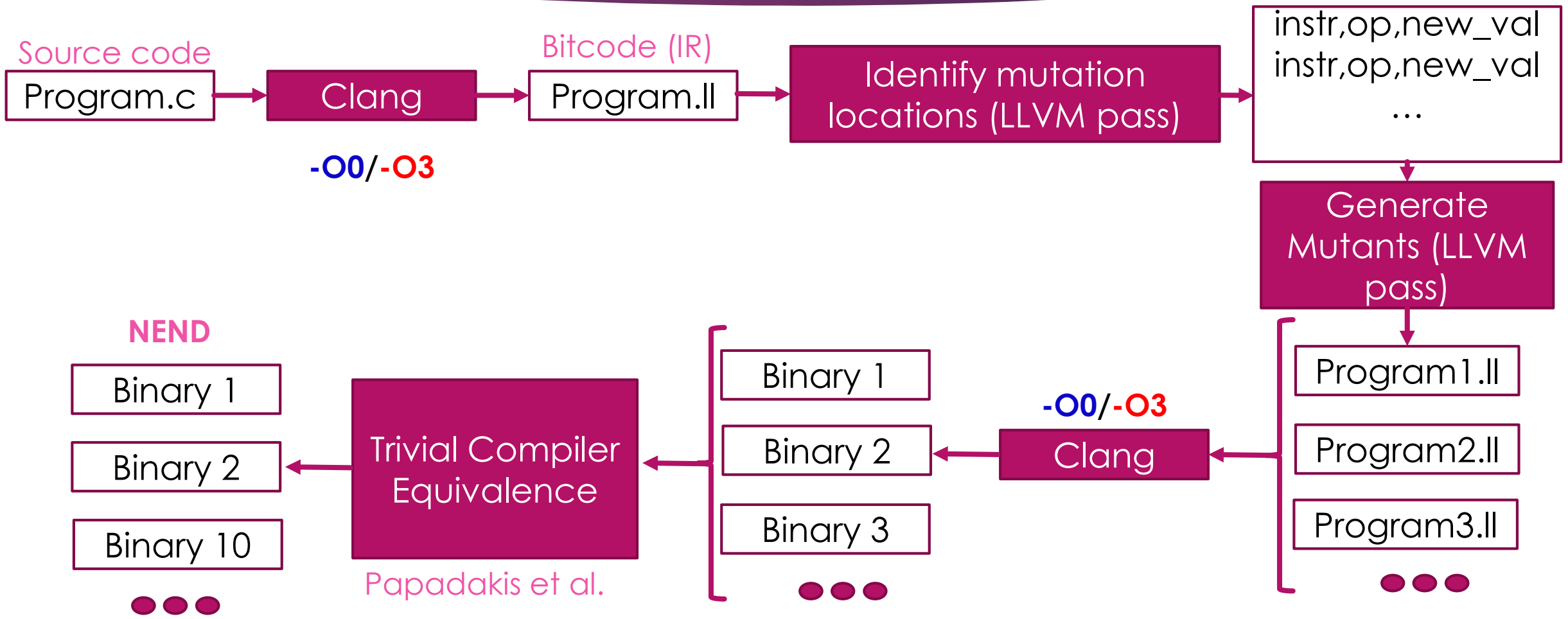
Non-equivalent non-duplicated
(**NEND**)

- Binary 1
- Binary 2
- Binary 10
- ...

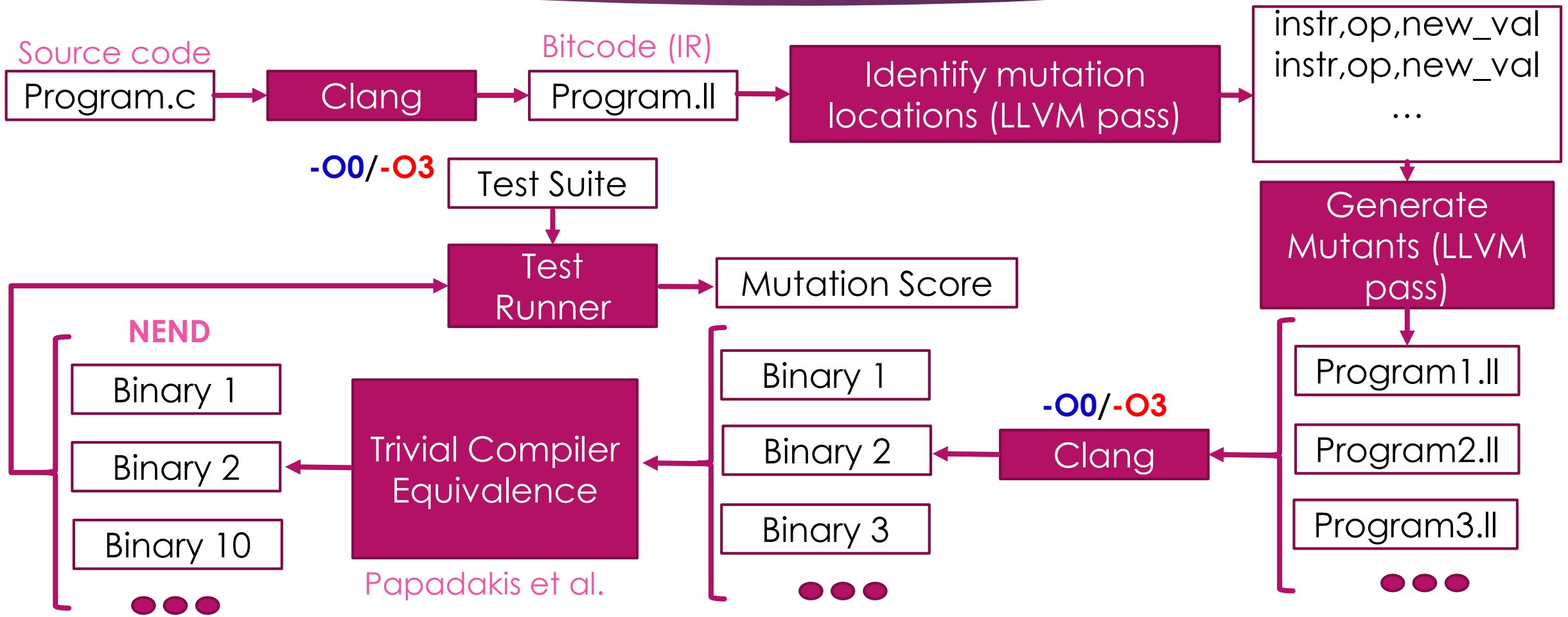
Trivial Compiler Equivalence

- Original Binary
- Binary 1
- Binary 2
- Binary 3
- Binary 4
- Binary 5
- ...

Implementation



Implementation



Results

RQ1:

HOW DO COMPILER OPTIMIZATIONS AFFECT THE NUMBER OF GENERATED MUTANTS?

No. of Instructions

Program	O0	O3
	Total LLVM Instructions	Total LLVM Instructions
chmod	675	403
chown	292	233
cut	1274	1110
dd	2436	2080
du	1199	835
head	1744	994
join	2088	1958
mkdir	251	159
mv	604	372
readlink	140	95
rm	322	200
rmdir	349	199
tac	871	581
tail	3030	2175
test	1895	1711
touch	606	413
tr	3116	2219
wc	1172	842
Overall	22064	16579

No. of Instructions

Program	O0	O3
	Total LLVM Instructions	Total LLVM Instructions
chmod	675	403
chown	292	233
cut	1274	1110
dd	2436	2080
du	1199	835
head	1744	994
join	2088	1958
mkdir	251	159
mv	604	372
readlink	140	95
rm	322	200
rmdir	349	199
tac	871	581
tail	3030	2175
test	1895	1711
touch	606	413
tr	3116	2219
wc	1172	842
Overall	22064	16579

No. of Instructions

Program	O0	O3
	Total LLVM	Total LLVM
touch	606	413
tr	3116	2219
wc	1172	842
Overall	22064	16579

The total number of instructions at O3 (16K instructions) is **smaller** than at O0 (22K instructions)

No. of Generated Mutants

Program	O0				O3			
	#M	E%	D%	#NEND	#M	E%	D%	#NEND
chmod	1069	3.8	0.8	1019	952	9.4	13.7	731
chown	467	3.2	0	452	453	14.7	9	345
cut	1958	3.3	0.5	1881	2547	4.3	14.1	2076
dd	4208	3.1	0.5	4055	4721	6.2	16.8	3627
du	1723	4.2	0.6	1638	1682	8.6	10.5	1358
head	2699	4	1	2562	2513	9.9	12.1	1957
join	2902	3.8	0.8	2766	3980	8.5	12.4	3144
mkdir	368	6.2	0.8	342	253	5.9	5.9	223
mv	907	3.5	0.5	870	792	10.3	12.3	612
readlink	192	3.6	0	185	140	3.5	8.5	123
rm	543	2.2	0	531	458	6.1	9.1	388
rmdir	479	5.2	2.3	443	417	3.8	11.5	353
tac	1151	5	1	1081	1120	5.9	9.9	942
tail	4673	3.3	0.7	4480	5409	9.2	12.5	4231
test	3077	2.4	2.1	2936	4717	5.2	15	3759
touch	1083	4.8	1.5	1014	983	12.7	10.2	757
tr	4280	3.7	1	4075	4624	4.5	13.1	3802
wc	1780	3.8	1	1694	1729	6.2	13.7	1384
Overall	33559	3.6	0.9	32024	37490	7.2	13.2	29812

No. of Generated Mutants

Program	O0				O3			
	#M	E%	D%	#NEND	#M	E%	D%	#NEND
chmod	1069	3.8	0.8	1019	952	9.4	13.7	731
chown	467	3.2	0	452	453	14.7	9	345
cut	1958	3.3	0.5	1881	2547	4.3	14.1	2076
dd	4208	3.1	0.5	4055	4721	6.2	16.8	3627
du	1723	4.2	0.6	1638	1682	8.6	10.5	1358
head	2699	4	1	2562	2513	9.9	12.1	1957
join	2902	3.8	0.8	2766	3980	8.5	12.4	3144
mkdir	368	6.2	0.8	342	253	5.9	5.9	223
mv	907	3.5	0.5	870	792	10.3	12.3	612
readlink	192	3.6	0	185	140	3.5	8.5	123
rm	543	2.2	0	531	458	6.1	9.1	388
rmdir	479	5.2	2.3	443	417	3.8	11.5	353
tac	1151	5	1	1081	1120	5.9	9.9	942
tail	4673	3.3	0.7	4480	5409	9.2	12.5	4231
test	3077	2.4	2.1	2936	4717	5.2	15	3759
touch	1083	4.8	1.5	1014	983	12.7	10.2	757
tr	4280	3.7	1	4075	4624	4.5	13.1	3802
wc	1780	3.8	1	1694	1729	6.2	13.7	1384
Overall	33559	3.6	0.9	32024	37490	7.2	13.2	29812

No. of Generated Mutants

Program	O0				O3			
	#M	E%	D%	#NEND	#M	E%	D%	#NEND
chmod	1069	3.8	0.8	1019	952	9.4	13.7	731
tr	4280	3.7	1	4075	4624	4.5	13.1	3802
wc	1780	3.8	1	1694	1729	6.2	13.7	1384
Overall	33559	3.6	0.9	32024	37490	7.2	13.2	29812

The total number of generated mutants at O3 (37K mutants) is **larger** than at O0 (33K mutants)

No. of Generated Mutants

Program	O0				O3			
	#M	E%	D%	#NEND	#M	E%	D%	#NEND
chmod	1069	3.8	0.8	1019	952	9.4	13.7	731
tr	4280	3.7	1	4075	4624	4.5	13.1	3802
wc	1780	3.8	1	1694	1729	6.2	13.7	1384
Overall	33559	3.6	0.9	32024	37490	7.2	13.2	29812

Answer 1:
Even though O3 has **less instructions** than O0, it has **more** generated mutants in total

RQ2:

HOW DO COMPILER OPTIMIZATIONS AFFECT THE NUMBER OF EQUIVALENT AND DUPLICATED MUTANTS?

Equivalent Mutants

Program	O0				O3			
	#M	E%	D%	#NEND	#M	E%	D%	#NEND
chmod	1069	3.8	0.8	1019	952	9.4	13.7	731
chown	467	3.2	0	452	453	14.7	9	345
cut	1958	3.3	0.5	1881	2547	4.3	14.1	2076
dd	4208	3.1	0.5	4055	4721	6.2	16.8	3627
du	1723	4.2	0.6	1638	1682	8.6	10.5	1358
head	2699	4	1	2562	2513	9.9	12.1	1957
join	2902	3.8	0.8	2766	3980	8.5	12.4	3144
mkdir	368	6.2	0.8	342	253	5.9	5.9	223
mv	907	3.5	0.5	870	792	10.3	12.3	612
readlink	192	3.6	0	185	140	3.5	8.5	123
rm	543	2.2	0	531	458	6.1	9.1	388
rmdir	479	5.2	2.3	443	417	3.8	11.5	353
tac	1151	5	1	1081	1120	5.9	9.9	942
tail	4673	3.3	0.7	4480	5409	9.2	12.5	4231
test	3077	2.4	2.1	2936	4717	5.2	15	3759
touch	1083	4.8	1.5	1014	983	12.7	10.2	757
tr	4280	3.7	1	4075	4624	4.5	13.1	3802
wc	1780	3.8	1	1694	1729	6.2	13.7	1384
Overall	33559	3.6	0.9	32024	37490	7.2	13.2	29812

Duplicated Mutants

Program	O0				O3			
	#M	E%	D%	#NEND	#M	E%	D%	#NEND
chmod	1069	3.8	0.8	1019	952	9.4	13.7	731
chown	467	3.2	0	452	453	14.7	9	345
cut	1958	3.3	0.5	1881	2547	4.3	14.1	2076
dd	4208	3.1	0.5	4055	4721	6.2	16.8	3627
du	1723	4.2	0.6	1638	1682	8.6	10.5	1358
head	2699	4	1	2562	2513	9.9	12.1	1957
join	2902	3.8	0.8	2766	3980	8.5	12.4	3144
mkdir	368	6.2	0.8	342	253	5.9	5.9	223
mv	907	3.5	0.5	870	792	10.3	12.3	612
readlink	192	3.6	0	185	140	3.5	8.5	123
rm	543	2.2	0	531	458	6.1	9.1	388
rmdir	479	5.2	2.3	443	417	3.8	11.5	353
tac	1151	5	1	1081	1120	5.9	9.9	942
tail	4673	3.3	0.7	4480	5409	9.2	12.5	4231
test	3077	2.4	2.1	2936	4717	5.2	15	3759
touch	1083	4.8	1.5	1014	983	12.7	10.2	757
tr	4280	3.7	1	4075	4624	4.5	13.1	3802
wc	1780	3.8	1	1694	1729	6.2	13.7	1384
Overall	33559	3.6	0.9	32024	37490	7.2	13.2	29812

No. of NEND Mutants

Program	O0				O3			
	#M	E%	D%	#NEND	#M	E%	D%	#NEND
chmod	1069	3.8	0.8	1019	952	9.4	13.7	731
chown	467	3.2	0	452	453	14.7	9	345
cut	1958	3.3	0.5	1881	2547	4.3	14.1	2076
dd	4208	3.1	0.5	4055	4721	6.2	16.8	3627
du	1723	4.2	0.6	1638	1682	8.6	10.5	1358
head	2699	4	1	2562	2513	9.9	12.1	1957
join	2902	3.8	0.8	2766	3980	8.5	12.4	3144
mkdir	368	6.2	0.8	342	253	5.9	5.9	223
mv	907	3.5	0.5	870	792	10.3	12.3	612
readlink	192	3.6	0	185	140	3.5	8.5	123
rm	543	2.2	0	531	458	6.1	9.1	388
rmdir	479	5.2	2.3	443	417	3.8	11.5	353
tac	1151	5	1	1081	1120	5.9	9.9	942
tail	4673	3.3	0.7	4480	5409	9.2	12.5	4231
test	3077	2.4	2.1	2936	4717	5.2	15	3759
touch	1083	4.8	1.5	1014	983	12.7	10.2	757
tr	4280	3.7	1	4075	4624	4.5	13.1	3802
wc	1780	3.8	1	1694	1729	6.2	13.7	1384
Overall	33559	3.6	0.9	32024	37490	7.2	13.2	29812

No. of NEND Mutants

Program	O0				O3			
	#M	E%	D%	#NEND	#M	E%	D%	#NEND
chmod	1069	3.8	0.8	1019	952	9.4	13.7	731
tr	4280	3.7	1	4075	4624	4.5	13.1	3802
wc	1780	3.8	1	1694	1729	6.2	13.7	1384
Overall	33559	3.6	0.9	32024	37490	7.2	13.2	29812

Answer 2:
Number of NEND mutants at O3 (29K mutants) is **smaller** than at O0 (32K mutants)

No. of NEND Mutants

Program	O0				O3			
	#M	E%	D%	#NEND	#M	E%	D%	#NEND
chmod	1069	3.8	0.8	1019	952	9.4	13.7	731
tr	4280	3.7	1	4075	4624	4.5	13.1	
wc	1780	3.8	1	1694	1729	6.2	13.7	
Overall	33559	3.6	0.9	32024	37490	7.2	13.2	

Answer 2:
Number of NEND mutants at O3 (29K mutants) is **small** (32K mutants)

Faster with optimizations

RQ3:

HOW DO COMPILER OPTIMIZATIONS AFFECT THE MUTATION SCORE?

Mutation Score

Program	O0		O3	
	All	NEND	All	NEND
chmod	35.7	36.9	33.7	35.7
chown	32.7	33.8	29.8	33.3
cut	55.7	57.6	55	55.3
dd	32.8	33.9	31.4	31.5
du	41.6	43.4	36.2	38
head	17.8	18.7	15.7	17.6
join	54.5	56.6	40	42.1
mkdir	52.9	56.4	55.3	57.8
mv	53.8	55.5	50.6	51.4
readlink	39.5	41	40.7	39.8
rm	42.7	43.6	35.3	37.1
rmdir	29.4	31.3	28.3	29.7
tac	41.3	43.5	37	38.4
tail	31.3	32.4	24.5	26.5
test	37.6	38.7	37.3	38.8
touch	39.4	41.6	38.2	41
tr	59.2	61.5	59.2	59.8
wc	32.4	33.8	26.7	26
Overall	40.4	41.9	37	38.5

Mutation Score

Program	O0		O3	
	All	NEND	All	NEND
chmod	35.7	36.9	33.7	35.7
chown	32.7	33.8	29.8	33.3
cut	55.7	57.6	55	55.3
dd	32.8	33.9	31.4	31.5
du	41.6	43.4	36.2	38
head	17.8	18.7	15.7	17.6
join	54.5	56.6	40	42.1
mkdir	52.9	56.4	55.3	57.8
mv	53.8	55.5	50.6	51.4
readlink	39.5	41	40.7	39.8
rm	42.7	43.6	35.3	37.1
rmdir	29.4	31.3	28.3	29.7
tac	41.3	43.5	37	38.4
tail	31.3	32.4	24.5	26.5
test	37.6	38.7	37.3	38.8
touch	39.4	41.6	38.2	41
tr	59.2	61.5	59.2	59.8
wc	32.4	33.8	26.7	26
Overall	40.4	41.9	37	38.5

Mutation Score

Program	O0		O3	
	All	NEND	All	NEND
tr	59.2	61.5	59.2	59.8
wc	32.4	33.8	26.7	26
Overall	40.4	41.9	37	38.5

Answer 3:
The mutation score values are **lower** at the optimized -O3 level than at the non optimized -O0 level both for all mutants and for only NEND mutants

RQ4:

HOW DO THESE EFFECTS VARY WITH THE CLASS OF MUTATION OPERATORS APPLIED?

Across Operator Analysis

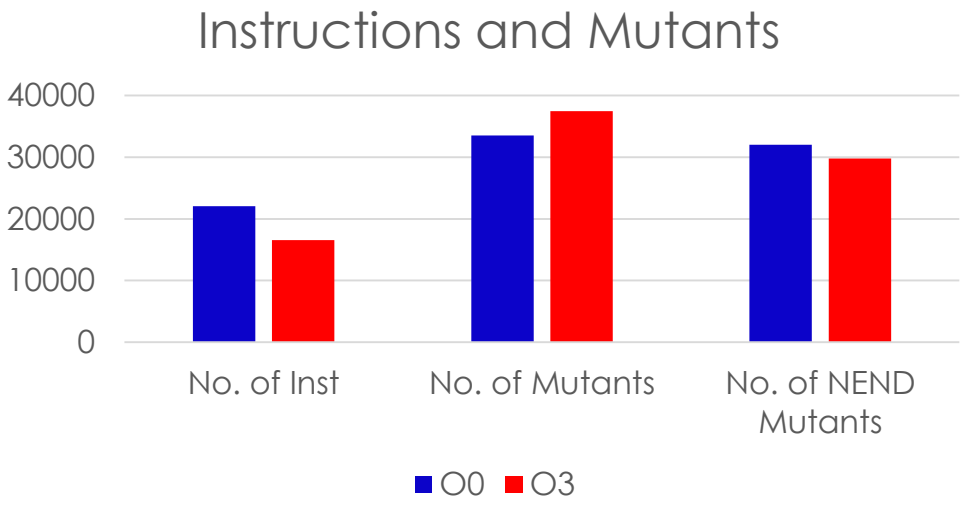
Relationship	Operators				
	All	AOR	LCR	ROR	ICR
For all generated mutants, overall #M at –O3 > #M at –O0	Yes	Yes	Yes	Yes	No
For only NEND mutants, overall #NEND at –O3 < #NEND at –O0	Yes	Yes	No	No	Yes
Overall E% at –O3 ≥ E% at –O0	Yes	Yes	Yes	Yes	Yes
Overall D% at –O3 ≥ D% at –O0	Yes	Yes	Yes	Yes	Yes
For all generated mutants, overall mutation score at –O3 < mutation score at –O0	Yes	Yes	Yes	Yes	Yes
For only NEND mutants, overall mutation score at –O3 < mutation score at –O0	Yes	Yes	Yes	Yes	Yes

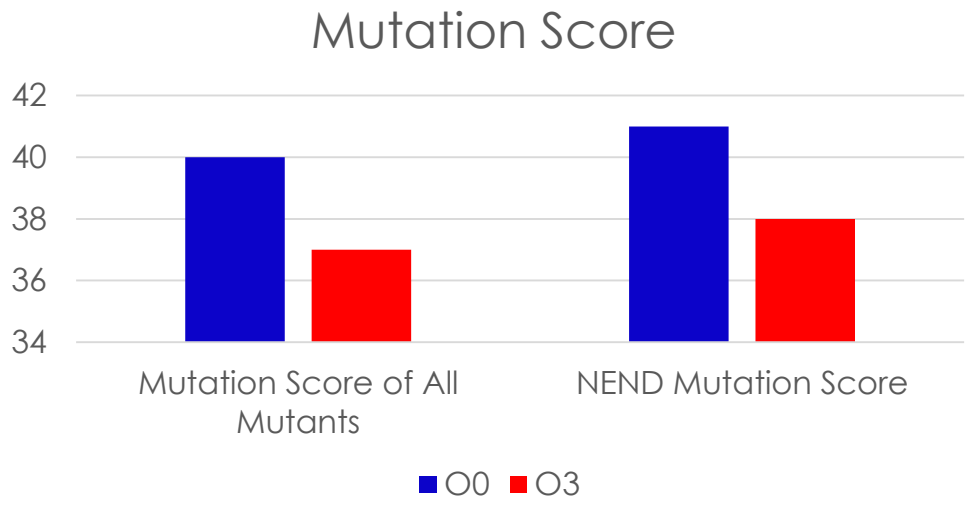
Across Operator Analysis

Answer 4:
The effects of -O0 and -O3 levels on mutation testing are most likely due to compiler optimizations and not due to specific mutation operators

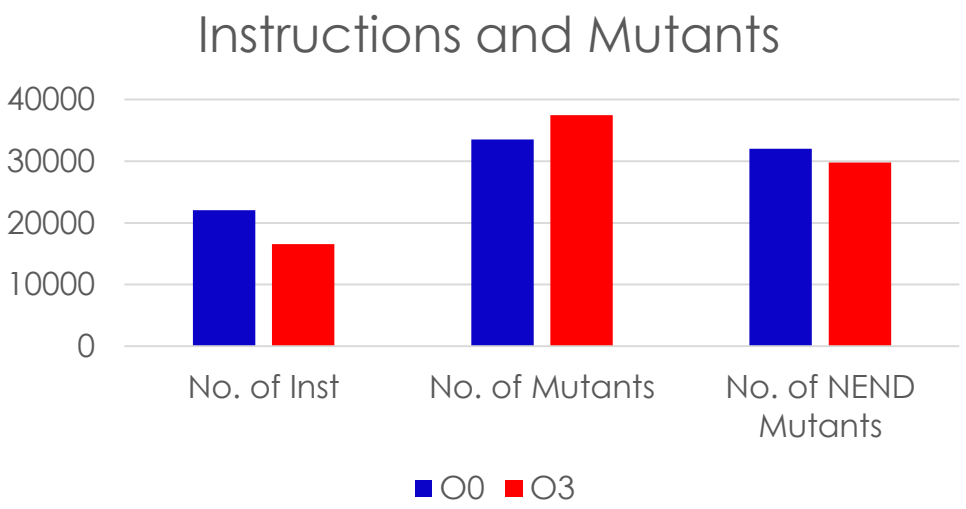
For al					R
For or					o
Overa					es
Overa					es
For al score					es
For only NEND mutants, overall mutation score at –O3 < mutation score at –O0	Yes	Yes	Yes	Yes	Yes

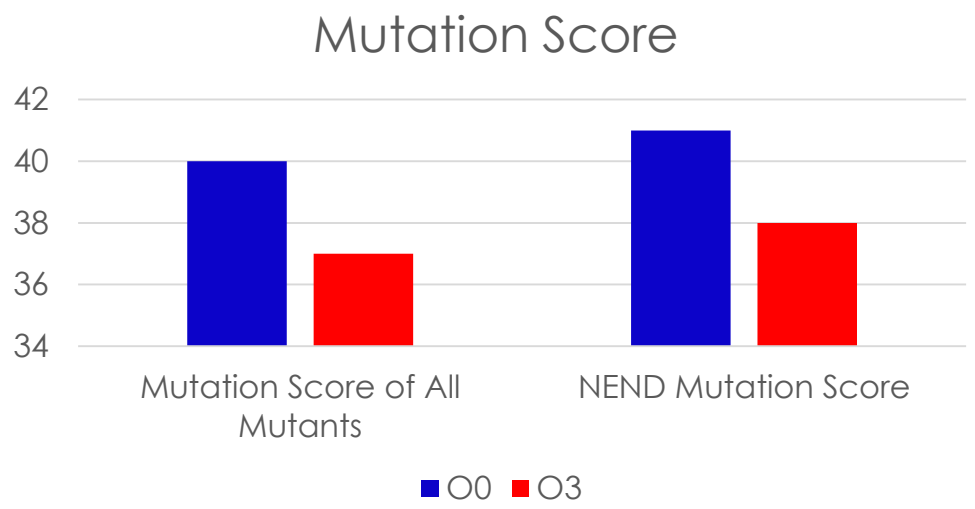
Summary


Instructions and Mutants
40000
30000
20000
10000
0
No. of Inst
No. of Mutants
No. of NEND Mutants
O0
O3


Mutation Score
42
40
38
36
34
Mutation Score of All Mutants
NEND Mutation Score
O0
O3

Summary





Takeaway: Apply mutation testing on optimized code