*The 46th International Conference on Software Engineering (ICSE 2024)*

# Beyond a Joke: Dead Code Elimination Can Delete Live Code

**Haoxin Tu**, Lingxiao Jiang, Debin Gao (Singapore Management University)
He Jiang (Dalian University of Technology)

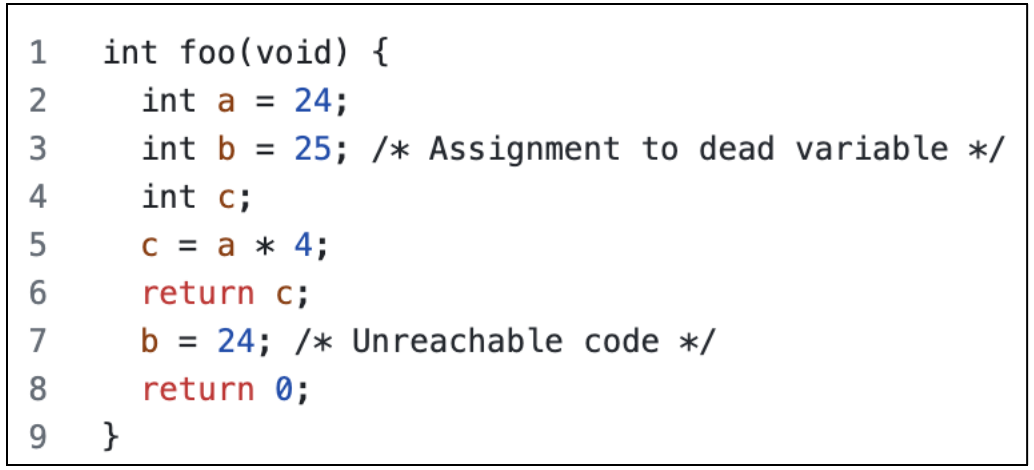
18/04/2024, Lisbon

## Background: Dead Code Elimination

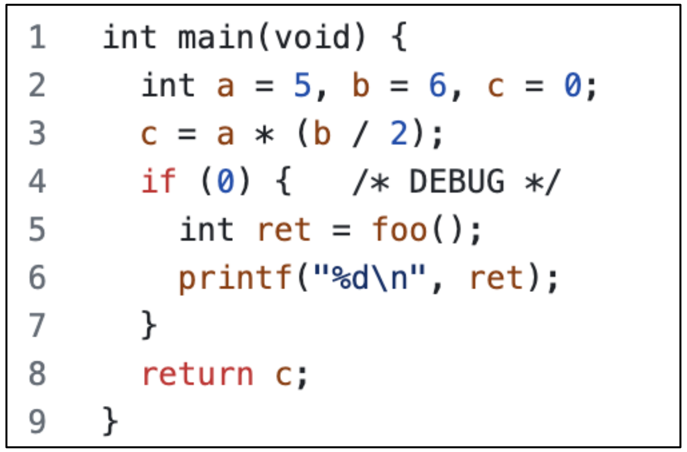
- **What is Dead Code Elimination (DCE)?**
  - A fundamental compiler optimization technique that removes dead code (e.g., unreachable or reachable but whose results are unused) in the program

## Background: Dead Code Elimination

- **What is Dead Code Elimination (DCE)?**
  - A fundamental compiler optimization technique that removes dead code (e.g., unreachable or reachable but whose results are unused) in the program



```c
int foo(void) {
  int a = 24;
  int b = 25; /* Assignment to dead variable */
  int c;
  c = a * 4;
  return c;
  b = 24; /* Unreachable code */
  return 0;
}
```

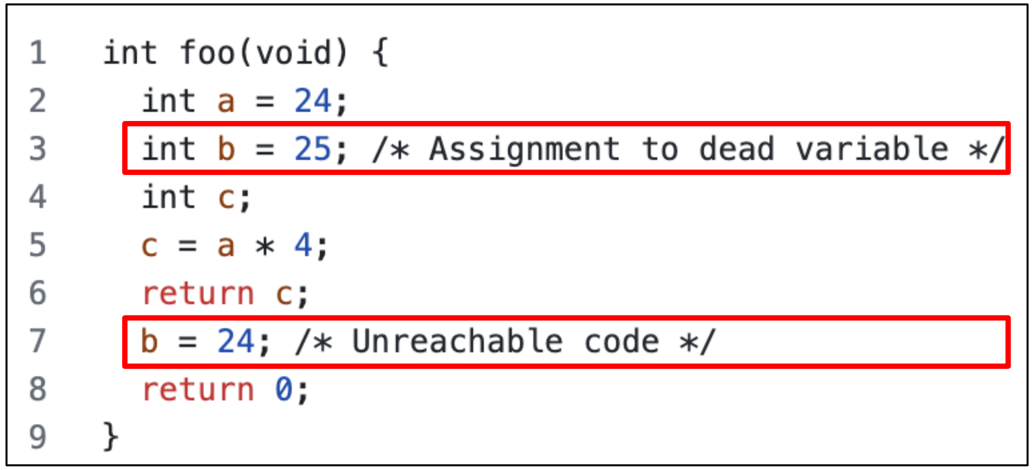


```c
int main(void) {
  int a = 5, b = 6, c = 0;
  c = a * (b / 2);
  if (0) {   /* DEBUG */
    int ret = foo();
    printf("%d\n", ret);
  }
  return c;
}
```

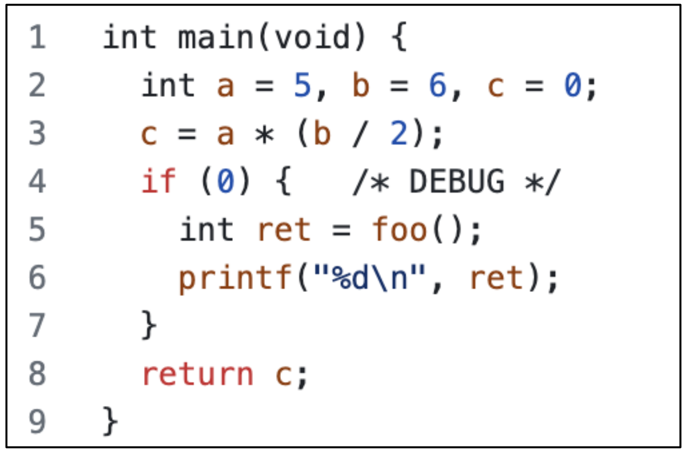
Examples are cited from https://en.wikipedia.org/wiki/Dead-code_elimination#Examples

## Background: Dead Code Elimination

- **What is Dead Code Elimination (DCE)?**
  - A fundamental compiler optimization technique that removes dead code (e.g., unreachable or reachable but whose results are unused) in the program



```c
int foo(void) {
  int a = 24;
  int b = 25; /* Assignment to dead variable */
  int c;
  c = a * 4;
  return c;
  b = 24; /* Unreachable code */
  return 0;
}
```



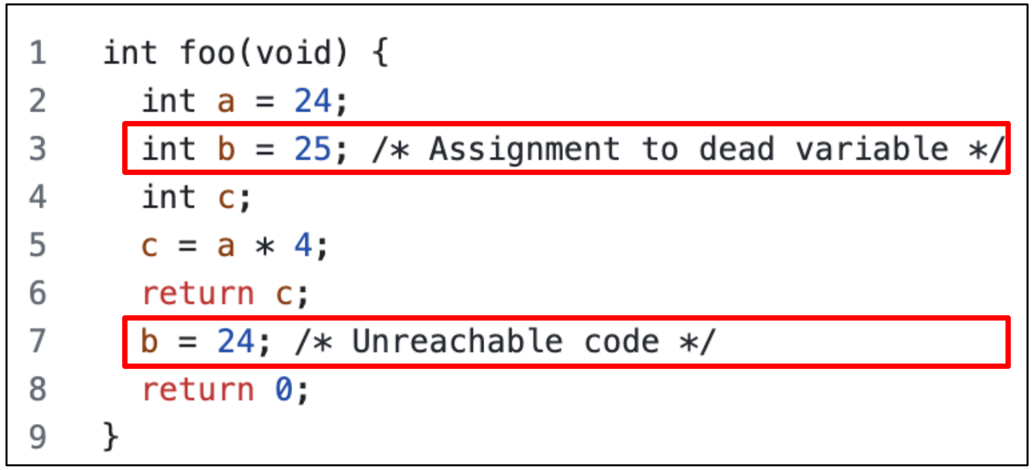
```c
int main(void) {
  int a = 5, b = 6, c = 0;
  c = a * (b / 2);
  if (0) {   /* DEBUG */
    int ret = foo();
    printf("%d\n", ret);
  }
  return c;
}
```

Examples are cited from https://en.wikipedia.org/wiki/Dead-code_elimination#Examples

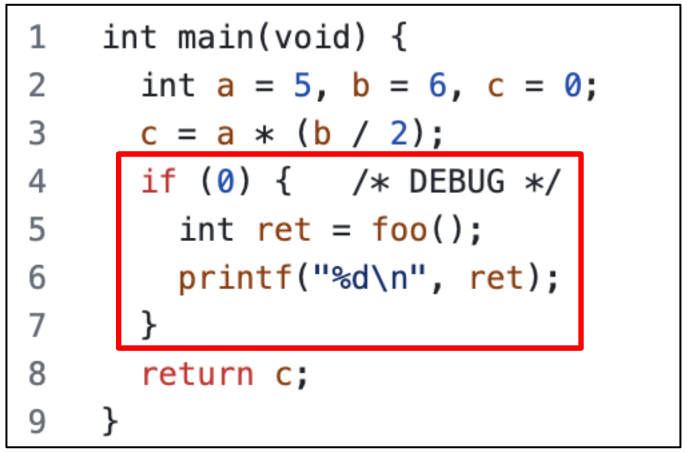
# Background: Dead Code Elimination

## ❑ What is Dead Code Elimination (DCE)?

➢ A fundamental compiler optimization technique that removes dead code (e.g., unreachable or reachable but whose results are unused) in the program



```c
int foo(void) {
  int a = 24;
  int b = 25; /* Assignment to dead variable */
  int c;
  c = a * 4;
  return c;
  b = 24; /* Unreachable code */
  return 0;
}
```



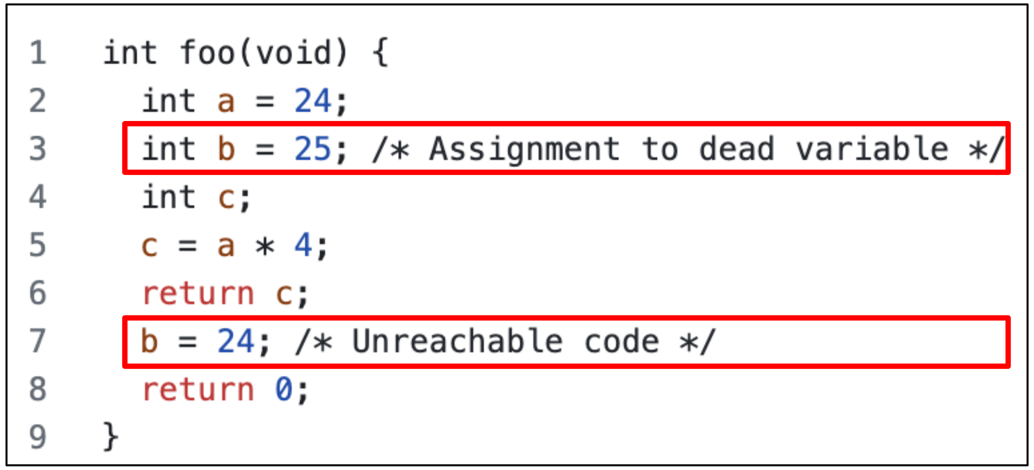
```c
int main(void) {
  int a = 5, b = 6, c = 0;
  c = a * (b / 2);
  if (0) {   /* DEBUG */
    int ret = foo();
    printf("%d\n", ret);
  }
  return c;
}
```

Examples are cited from https://en.wikipedia.org/wiki/Dead-code_elimination#Examples

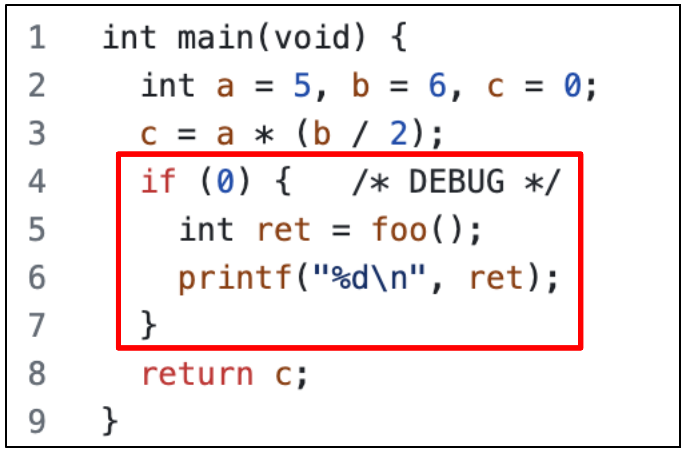
# Background: Dead Code Elimination

## ❑ What is Dead Code Elimination (DCE)?

- ➢ A fundamental compiler optimization technique that removes dead code (e.g., unreachable or reachable but whose results are unused) in the program



```c
int foo(void) {
  int a = 24;
  int b = 25; /* Assignment to dead variable */
  int c;
  c = a * 4;
  return c;
  b = 24; /* Unreachable code */
  return 0;
}
```



```c
int main(void) {
  int a = 5, b = 6, c = 0;
  c = a * (b / 2);
  if (0) {    /* DEBUG */
    int ret = foo();
    printf("%d\n", ret);
  }
  return c;
}
```

- ➢ Benefits of DCE: produce ***smaller*** or ***faster*** executables
  - • Many other applications and languages (Java, Go, and Rust, etc.)

Examples are cited from https://en.wikipedia.org/wiki/Dead-code_elimination#Examples

## Motivation

## Motivation

## Motivation

**Question: Can DCE happen to erroneously delete live code?**

## Motivation

**Question: Can DCE happen to erroneously delete live code?**

## Motivation

**Question: Can DCE happen to erroneously delete live code?**

- **Motivating example**

## Motivation

**Question: Can DCE happen to erroneously delete live code?**

- **Motivating example**

```
int idx = 0;
int a = 0;
void __attribute__((noinline)) marker_2(){ ++idx; }
static void c() { marker_2(); }
void d(int j) { for (;;) ; } // infinite loop
void e() { for (int i = 0; i < 100; m++)  d(i); }
void f() {
  e(); // live code here is erroneously deleted
  c();
}
void g() { if (a == 0x99)     f();   }

int main (int argc, char* argv[]) {
  // when a = 0x99, the bug triggers
  a = strtol(argv[1], NULL, 16);
  g();
  printf("\%d", idx);
  return 0;
}
```

**A *miscompilation* bug detected by our approach**

## Motivation

**Question: Can DCE happen to erroneously delete live code?**

- **Motivating example**

```
int idx = 0;
int a = 0;
void __attribute__((noinline)) marker_2(){ ++idx; }
static void c() { marker_2(); }
void d(int j) { for (;;) ; } // infinite loop
void e() { for (int i = 0; i < 100; m++)  d(i); }
void f() {
  e(); // live code here is erroneously deleted
  c();
}
void g() { if (a == 0x99)     f();   }

int main (int argc, char* argv[]) {
  // when a = 0x99, the bug triggers
  a = strtol(argv[1], NULL, 16);
  g();
  printf("\%d", idx);
  return 0;
}
```

**A *miscompilation* bug detected by our approach**

## Motivation

SMU SINGAPORE MANAGEMENT UNIVERSITY | School of Computing and Information Systems

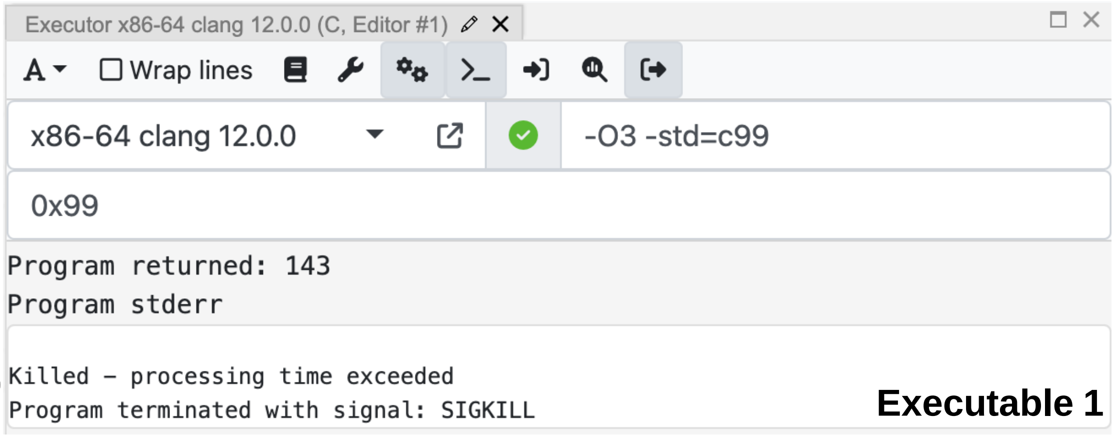
**Question: Can DCE happen to erroneously delete live code?**

➢ **Motivating example**

```
int idx = 0;
int a = 0;
void __attribute__((noinline)) marker_2(){ ++idx; }
static void c() { marker_2(); }
void d(int j) { for (;;) ; } // infinite loop
void e() { for (int i = 0; i < 100; m++)  d(i); }
void f() {
  e(); // live code here is erroneously deleted
  c();
}
void g() { if (a == 0x99)     f();   }

int main (int argc, char* argv[]) {
  // when a = 0x99, the bug triggers
  a = strtol(argv[1], NULL, 16);
  g();
  printf("\%d", idx);
  return 0;
}
```

**A *miscompilation* bug detected by our approach**



x86-64 clang 12.0.0 -O3 -std=c99

**(Input)** 0x99

```
Program returned: 143
Program stderr
```

**(Output)**

```
Killed - processing time exceeded
Program terminated with signal: SIGKILL
```

**Executable 1**

https://godbolt.org/z/z7zxexfr1

# Motivation

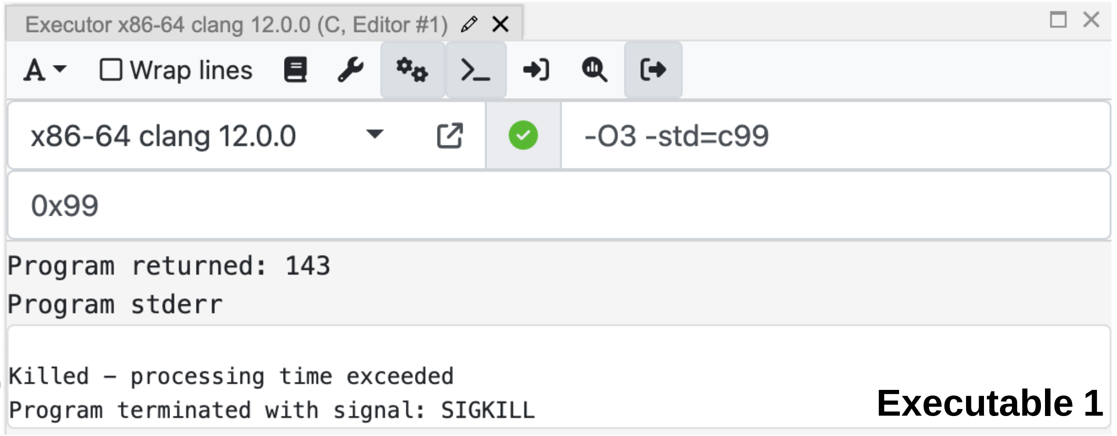
**Question: Can DCE happen to erroneously delete live code?**

- **Motivating example**

```
int idx = 0;
int a = 0;
void __attribute__((noinline)) marker_2(){ ++idx; }
static void c() { marker_2(); }
void d(int j) { for (;;) ; } // infinite loop
void e() { for (int i = 0; i < 100; m++)  d(i); }
void f() {
  e(); // live code here is erroneously deleted
  c();
}
void g() { if (a == 0x99)     f();   }

int main (int argc, char* argv[]) {
  // when a = 0x99, the bug triggers
  a = strtol(argv[1], NULL, 16);
  g();
  printf("\%d", idx);
  return 0;
}
```

**A *miscompilation* bug detected by our approach**



Executor x86-64 clang 12.0.0 (C, Editor #1)

x86-64 clang 12.0.0 -O3 -std=c99

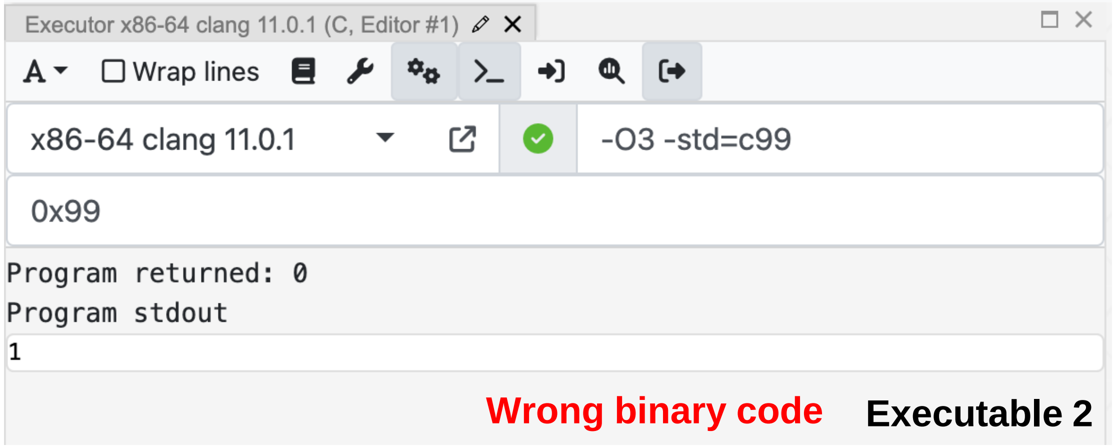
**(Input)** 0x99

**(Output)**

```
Program returned: 143
Program stderr

Killed - processing time exceeded
Program terminated with signal: SIGKILL
```

**Executable 1**



Executor x86-64 clang 11.0.1 (C, Editor #1)

x86-64 clang 11.0.1 -O3 -std=c99

**(Input)** 0x99

**(Output)**

```
Program returned: 0
Program stdout
1
```

**Wrong binary code** **Executable 2**

https://godbolt.org/z/z7zxexfr1

## Motivation

SMU SINGAPORE MANAGEMENT UNIVERSITY | School of Computing and Information Systems

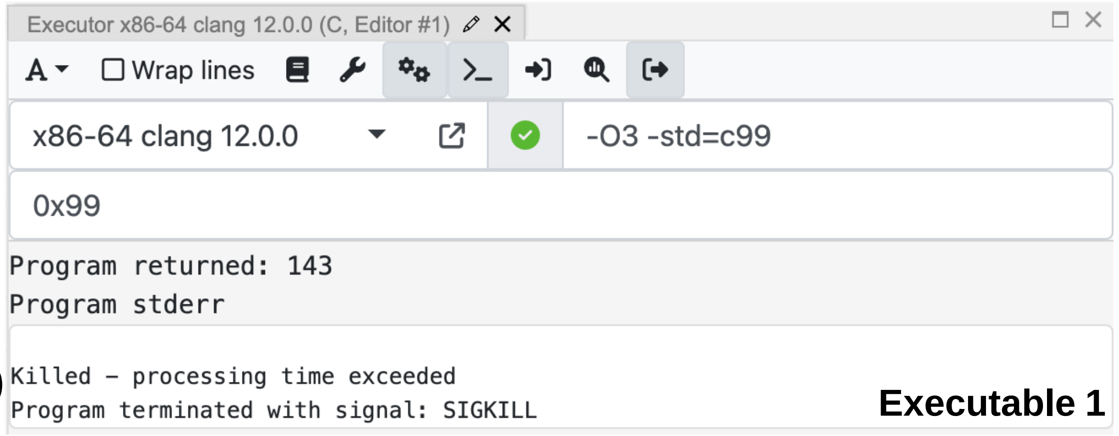
**Question: Can DCE happen to erroneously delete live code?**

➢ **Motivating example**

```
int idx = 0;
int a = 0;
void __attribute__((noinline)) marker_2(){ ++idx; }
static void c() { marker_2(); }
void d(int j) { for (;;) ; } // infinite loop
void e() { for (int i = 0; i < 100; m++)  d(i); }
void f() {
  e(); // live code here is erroneously deleted
  c();
}
void g() { if (a == 0x99)   f();   }

int main (int argc, char* argv[]) {
  // when a = 0x99, the bug triggers
  a = strtol(argv[1], NULL, 16);
  g();
  printf("\%d", idx);
  return 0;
}
```

**A *miscompilation* bug detected by our approach**

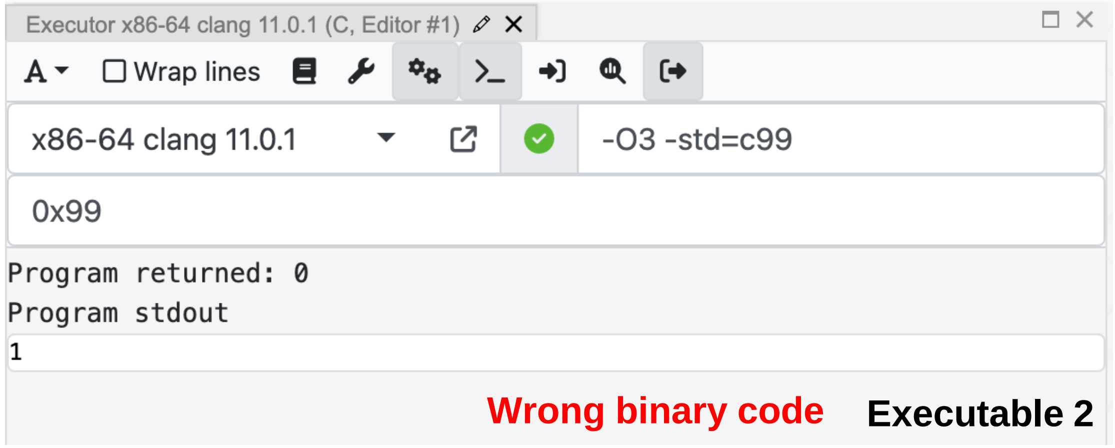


(Input)

(Output)

```
x86-64 clang 12.0.0    -O3 -std=c99
0x99
Program returned: 143
Program stderr
Killed - processing time exceeded
Program terminated with signal: SIGKILL
```

**Executable 1**



(Input)

(Output)

```
x86-64 clang 11.0.1    -O3 -std=c99
0x99
Program returned: 0
Program stdout
1
```

**Wrong binary code** **Executable 2**

https://godbolt.org/z/z7zxexfr1

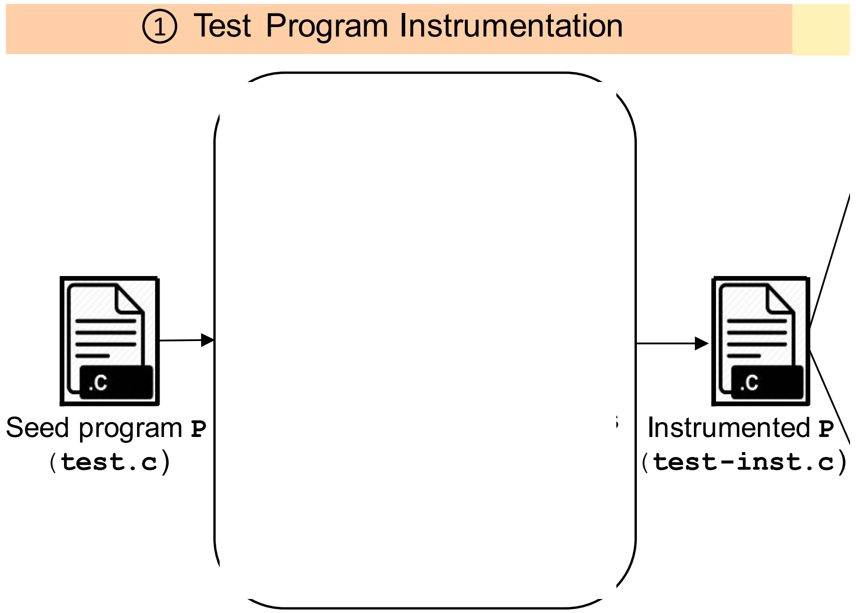
## Solution: Xdead

## Solution: Xdead

- **Key insights: divergence indicates DCE bugs**
  - Identify the divergent portions in binary first
  - leverage symbolic execution to reveal the divergent portion

# Solution: Xdead

- **Key insights: divergence indicates DCE bugs**
  - Identify the divergent portions in binary first
  - leverage symbolic execution to reveal the divergent portion

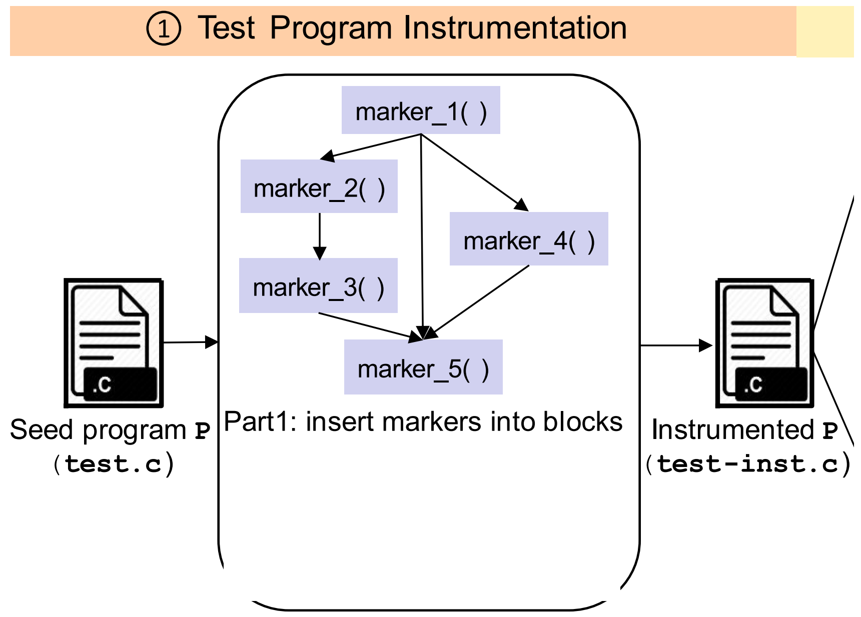


# Solution: Xdead

➢ **Key insights: divergence indicates DCE bugs**

– Identify the divergent portions in binary first

– leverage symbolic execution to reveal the divergent portion

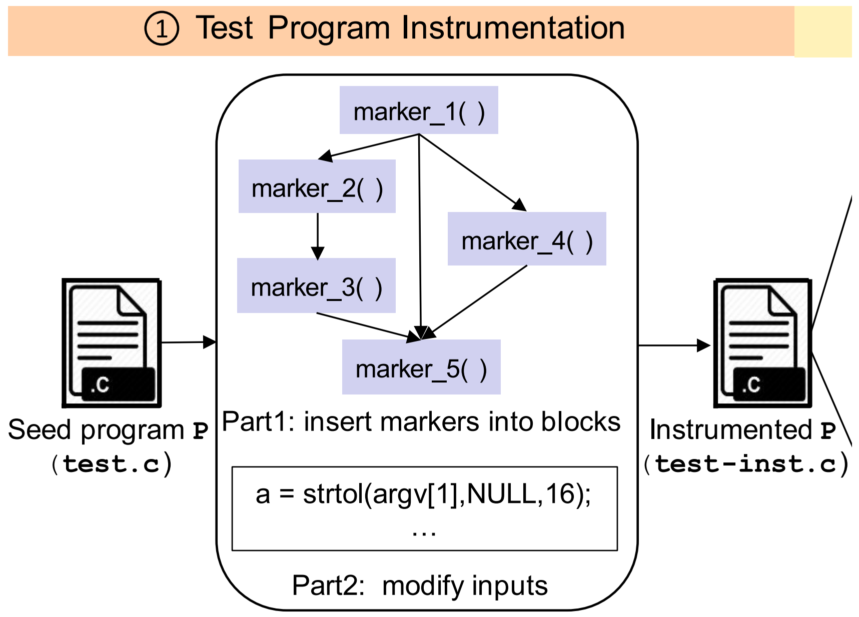


**The marker function:**

```
void __attribute__((noinline)) marker_i() { ++idx; };
```

# Solution: Xdead

- **Key insights: divergence indicates DCE bugs**
  - Identify the divergent portions in binary first
  - leverage symbolic execution to reveal the divergent portion



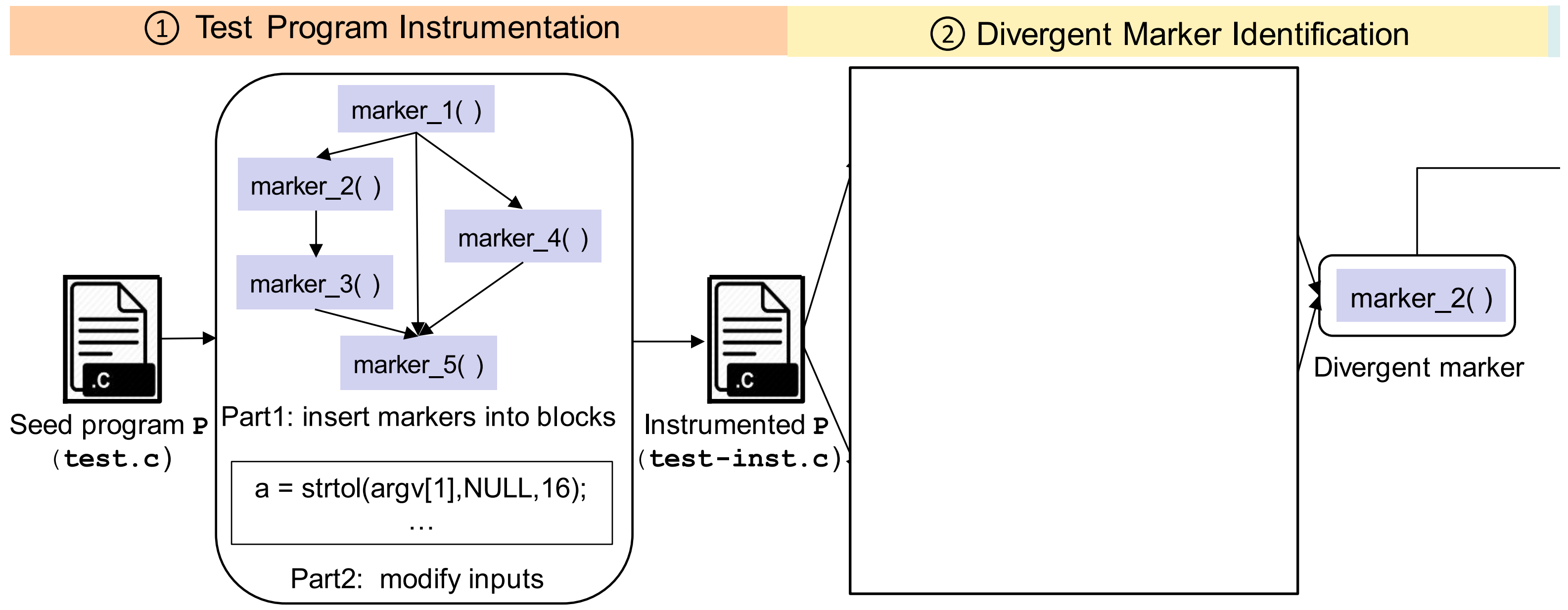
① Test Program Instrumentation
marker_1( )
marker_2( )
marker_3( )
marker_4( )
marker_5( )
.C
Seed program P (test.c)
Part1: insert markers into blocks
a = strtol(argv[1],NULL,16);
…
Part2: modify inputs
Instrumented P (test-inst.c)

**The marker function:**

```
void __attribute__((noinline)) marker_i() { ++idx; };
```

# Solution: Xdead

- **Key insights: divergence indicates DCE bugs**
  - Identify the divergent portions in binary first
  - leverage symbolic execution to reveal the divergent portion

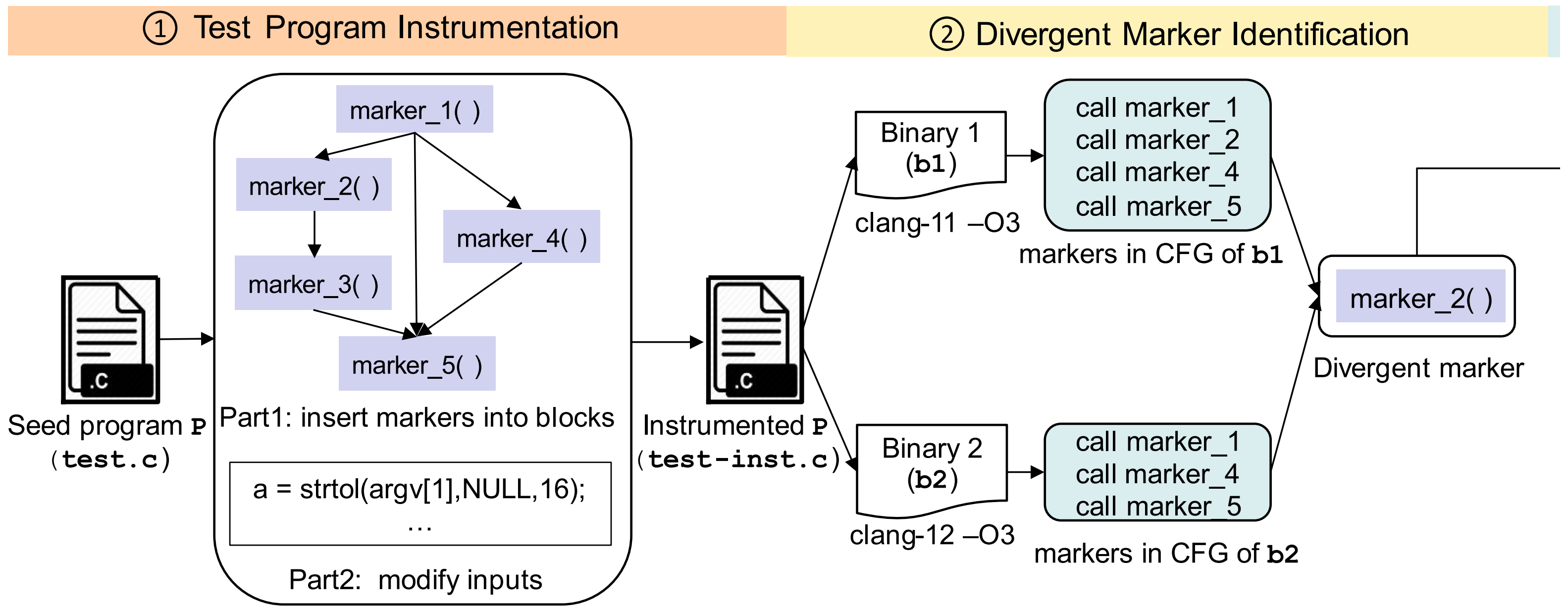


**The marker function:**

```
void __attribute__((noinline)) marker_i() { ++idx; };
```

## Solution: Xdead

- **Key insights: divergence indicates DCE bugs**
  - Identify the divergent portions in binary first
  - leverage symbolic execution to reveal the divergent portion

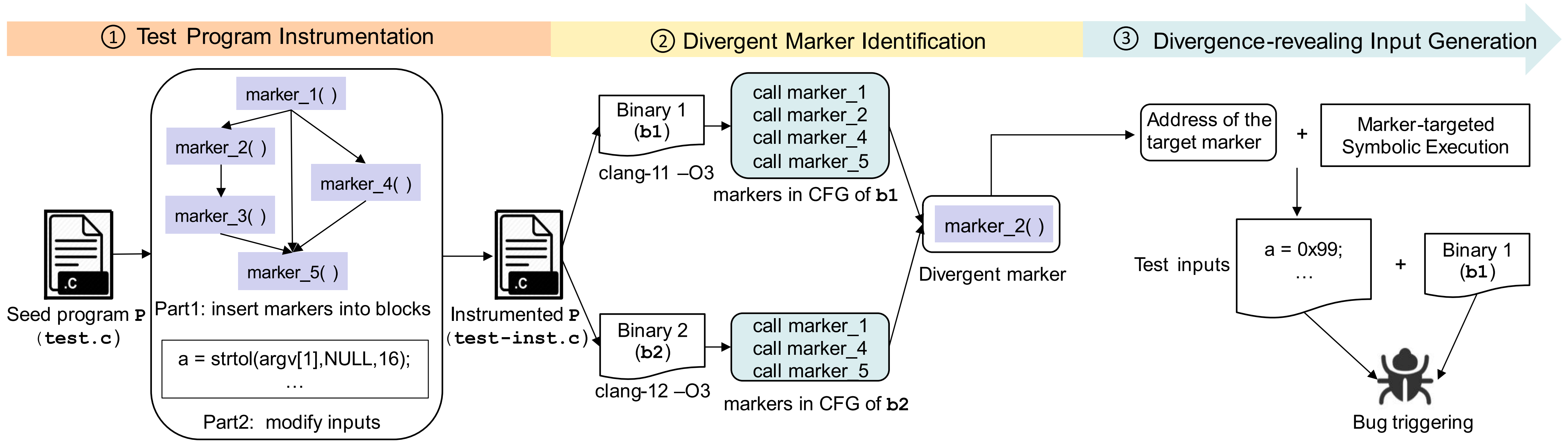


**The marker function:**

```
void __attribute__((noinline)) marker_i() { ++idx; };
```

# Solution: Xdead

- **Key insights: divergence indicates DCE bugs**
  - Identify the divergent portions in binary first
  - leverage symbolic execution to reveal the divergent portion



**The marker function:**

```
void __attribute__((noinline)) marker_i() { ++idx; };
```

## Preliminary results

## Preliminary results

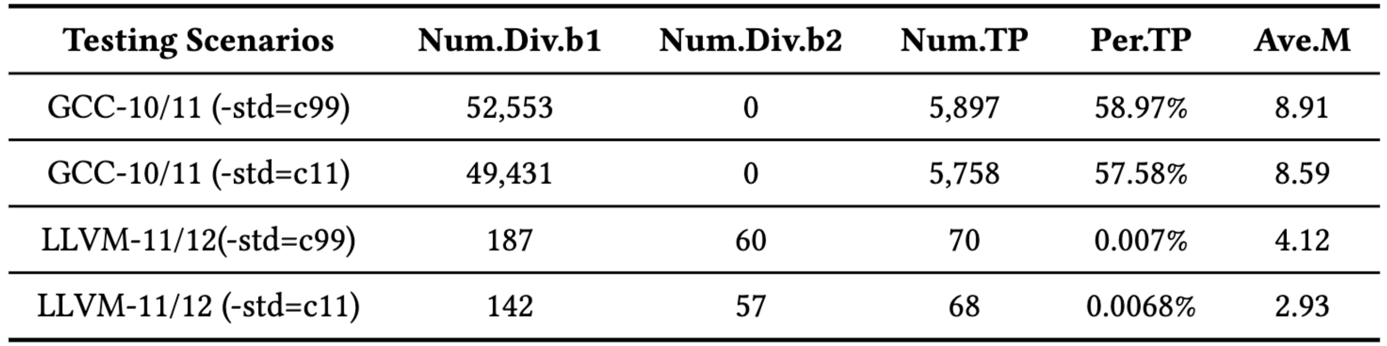
- **Evaluation setup**
  - Benchmark
    - **10,000** seed program from Csmith
  - Subjects
    - GCC and LLVM
  - Running setting
    - four scenarios under “–O3”

- **Metric**
  - Number of divergent markers
  - Number of bugs

# Preliminary results

- **Evaluation setup**
  - Benchmark
    - **10,000** seed program from Csmith
  - Subjects
    - GCC and LLVM
  - Running setting
    - four scenarios under "–O3"

**Table 1: Statistics of divergent markers and test programs**



| Testing Scenarios | Num.Div.b1 | Num.Div.b2 | Num.TP | Per.TP | Ave.M |
|---|---|---|---|---|---|
| GCC-10/11 (-std=c99) | 52,553 | 0 | 5,897 | 58.97% | 8.91 |
| GCC-10/11 (-std=c11) | 49,431 | 0 | 5,758 | 57.58% | 8.59 |
| LLVM-11/12(-std=c99) | 187 | 60 | 70 | 0.007% | 4.12 |
| LLVM-11/12 (-std=c11) | 142 | 57 | 68 | 0.0068% | 2.93 |

- **Metric**
  - Number of divergent markers
  - Number of bugs

## Preliminary results

- **Evaluation setup**
  - Benchmark
    - **10,000** seed program from Csmith
  - Subjects
    - GCC and LLVM
  - Running setting
    - four scenarios under “–O3”

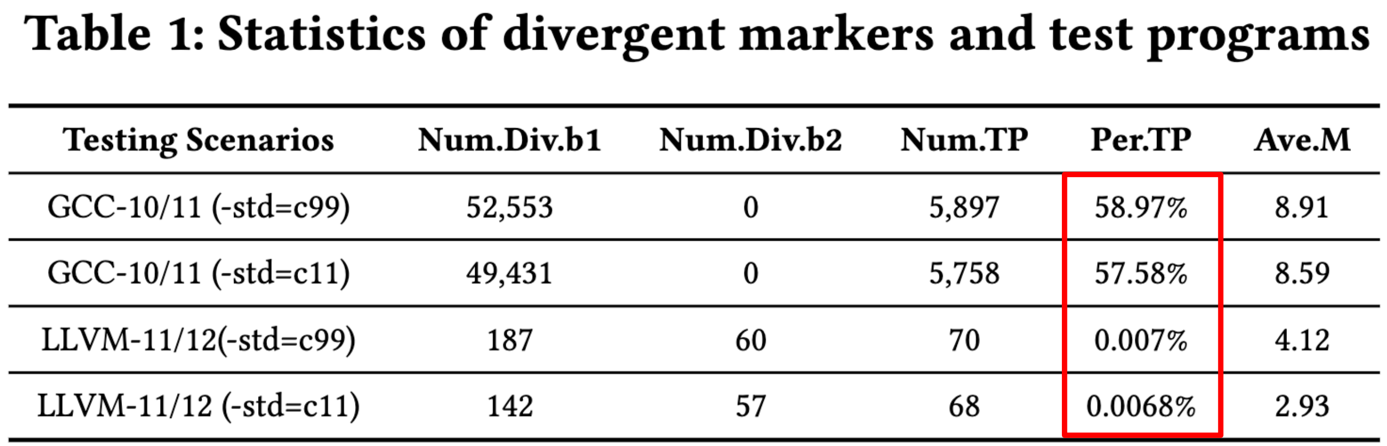


**Table 1: Statistics of divergent markers and test programs**

| Testing Scenarios | Num.Div.b1 | Num.Div.b2 | Num.TP | Per.TP | Ave.M |
|---|---|---|---|---|---|
| GCC-10/11 (-std=c99) | 52,553 | 0 | 5,897 | 58.97% | 8.91 |
| GCC-10/11 (-std=c11) | 49,431 | 0 | 5,758 | 57.58% | 8.59 |
| LLVM-11/12(-std=c99) | 187 | 60 | 70 | 0.007% | 4.12 |
| LLVM-11/12 (-std=c11) | 142 | 57 | 68 | 0.0068% | 2.93 |

- **Metric**
  - Number of divergent markers
  - Number of bugs

- **Summary**
  - Found many divergent portions indicating erroneously deleted live code (i.e., wrong compiler optimization opportunities)
  - Detected ***Two*** miscompilation bugs in LLVM compilers

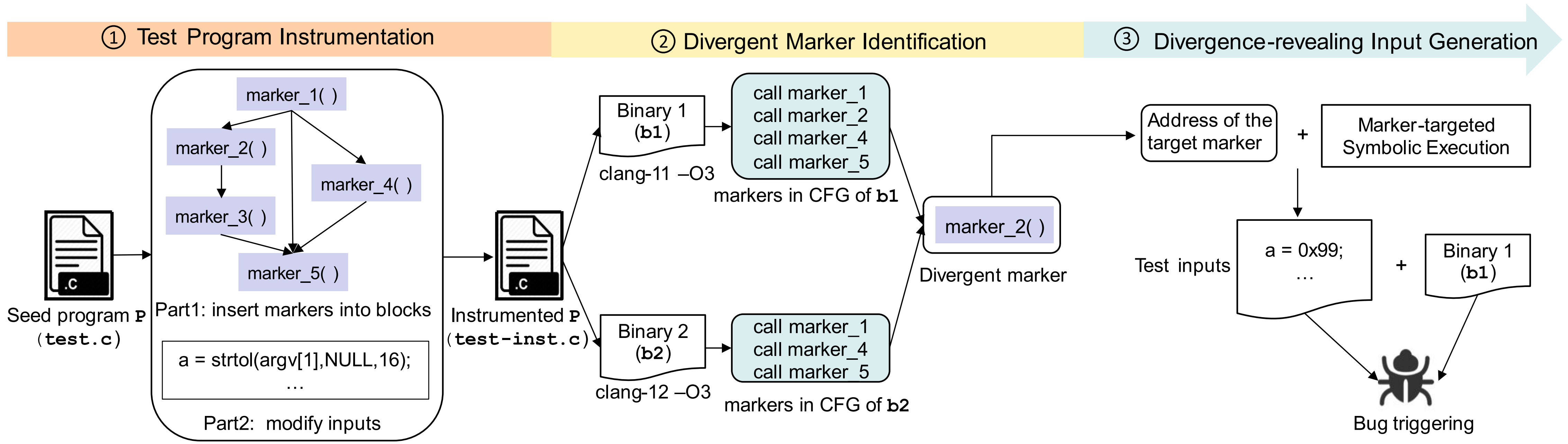
## Conclusion

## Conclusion

**Answer: DCE can erroneously delete live code sometimes (Solution: Xdead)**

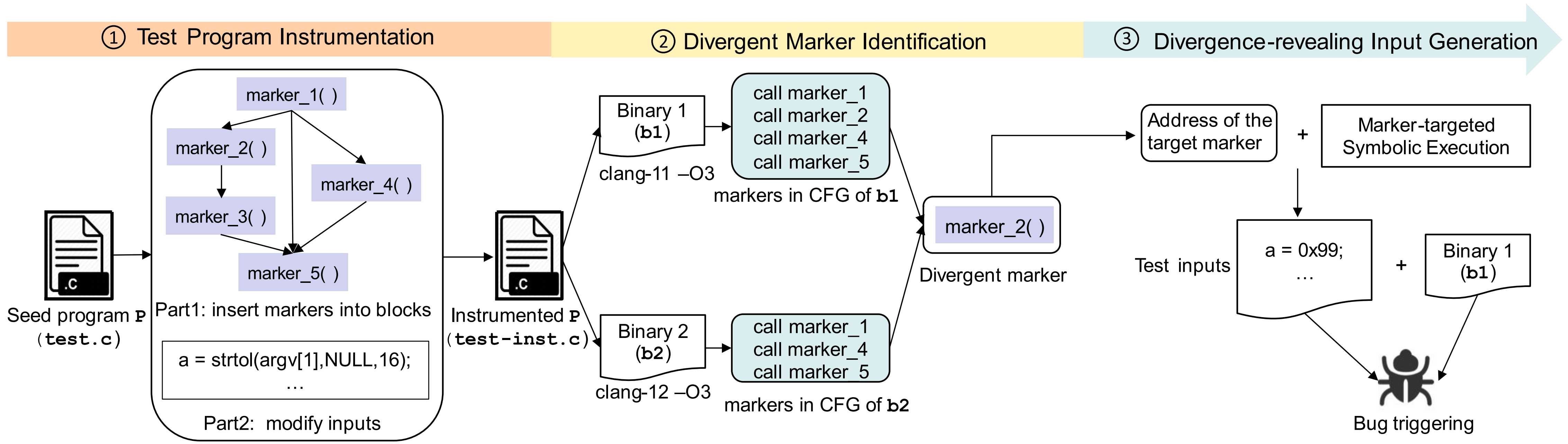
# Conclusion

**Answer: DCE can erroneously delete live code sometimes (Solution: Xdead)**



# Conclusion

**Answer: DCE can erroneously delete live code sometimes (Solution: Xdead)**



- **Future work**
  - Utilize more fine-grained binary analysis to identify fine-grained divergent portions in **Part 1**
  - Improve the efficiency of **Part 3**
    - efficient path exploration

**Paper**

**Code**

*The 46th International Conference on Software Engineering (ICSE 2024)*

# Thank you & Questions?

## Beyond a Joke: Dead Code Elimination Can Delete Live Code

**Haoxin Tu**, Lingxiao Jiang, Debin Gao (Singapore Management University)
He Jiang (Dalian University of Technology)

18/04/2024, Lisbon