Finding Missed Optimizations

Through the Lens of Dead Code Elimination

AST.ethz.ch
ADVANCED SOFTWARE
TECHNOLOGIES

ETH zürich

The “Perfect” Compiler

The “Perfect” Compiler

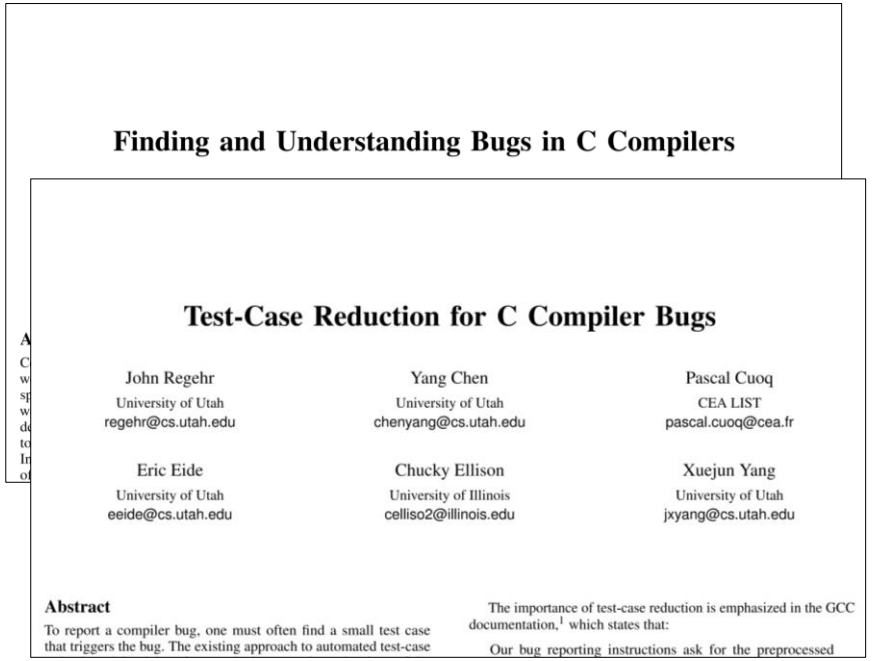
Correct

Robust

The “Perfect” Compiler

Correct

Robust



Finding and Understanding Bugs in C Compilers

Test-Case Reduction for C Compiler Bugs

John Regehr
University of Utah
regehr@cs.utah.edu

Yang Chen
University of Utah
chenyang@cs.utah.edu

Pascal Cuoq
CEA LIST
pascal.cuoq@cea.fr

Eric Eide
University of Utah
eeide@cs.utah.edu

Chucky Ellison
University of Illinois
celliso2@illinois.edu

Xuejun Yang
University of Utah
jxyang@cs.utah.edu

Abstract

To report a compiler bug, one must often find a small test case that triggers the bug. The existing approach to automated test-case

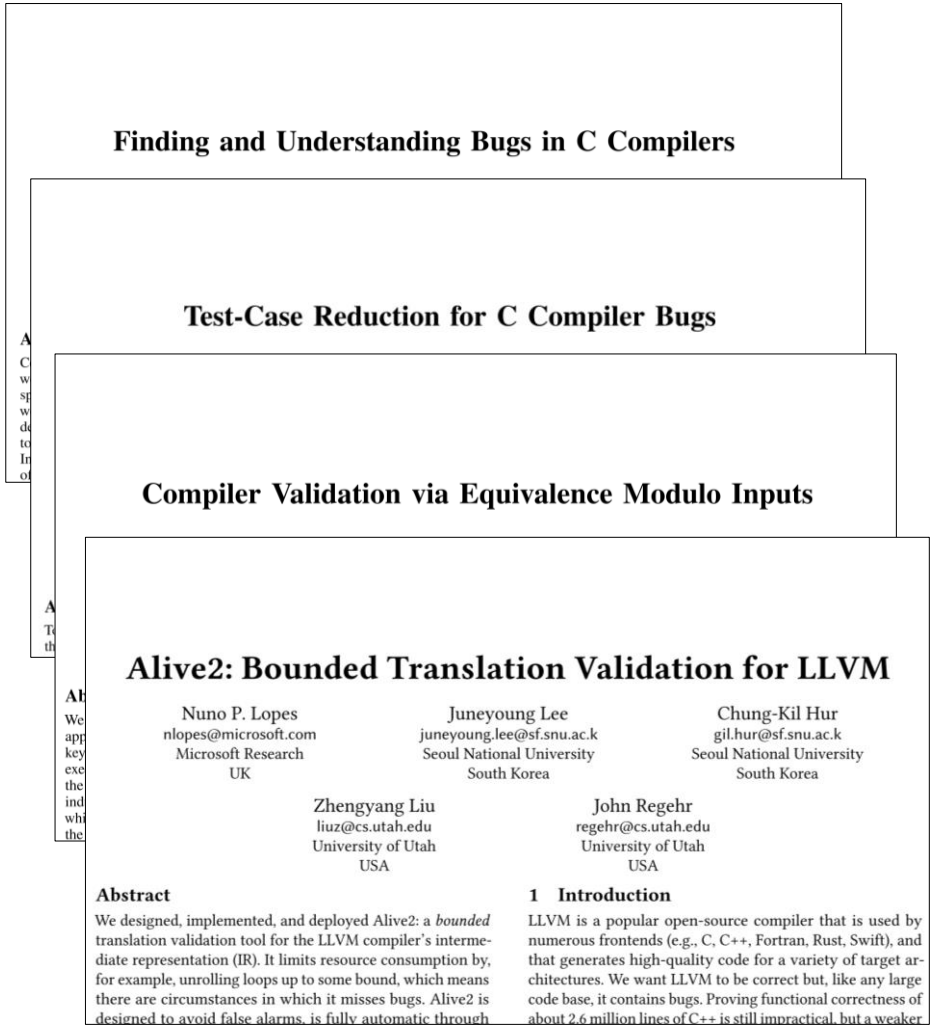
The importance of test-case reduction is emphasized in the GCC documentation,[1] which states that:

Our bug reporting instructions ask for the preprocessed

The “Perfect” Compiler

Correct

Robust



Finding and Understanding Bugs in C Compilers

Test-Case Reduction for C Compiler Bugs

Compiler Validation via Equivalence Modulo Inputs

Alive2: Bounded Translation Validation for LLVM

Nuno P. Lopes
nlopes@microsoft.com
Microsoft Research
UK

Juneyoung Lee
juneyoung.lee@sf.snu.ac.k
Seoul National University
South Korea

Chung-Kil Hur
gil.hur@sf.snu.ac.k
Seoul National University
South Korea

Zhengyang Liu
liuz@cs.utah.edu
University of Utah
USA

John Regehr
regehr@cs.utah.edu
University of Utah
USA

Abstract

We designed, implemented, and deployed Alive2: a *bounded* translation validation tool for the LLVM compiler's intermediate representation (IR). It limits resource consumption by, for example, unrolling loops up to some bound, which means there are circumstances in which it misses bugs. Alive2 is designed to avoid false alarms, is fully automatic through

1 Introduction

LLVM is a popular open-source compiler that is used by numerous frontends (e.g., C, C++, Fortran, Rust, Swift), and that generates high-quality code for a variety of target architectures. We want LLVM to be correct but, like any large code base, it contains bugs. Proving functional correctness of about 2.6 million lines of C++ is still impractical, but a weaker

The “Perfect” Compiler

Correct

Performance

Robust

The “Perfect” Compiler

Correct

Performance

Robust

Domain Specific

The “Perfect” Compiler

Correct

Performance

Robust

Domain Specific

General Optimizations

The “Perfect” Compiler

Correct

Performance

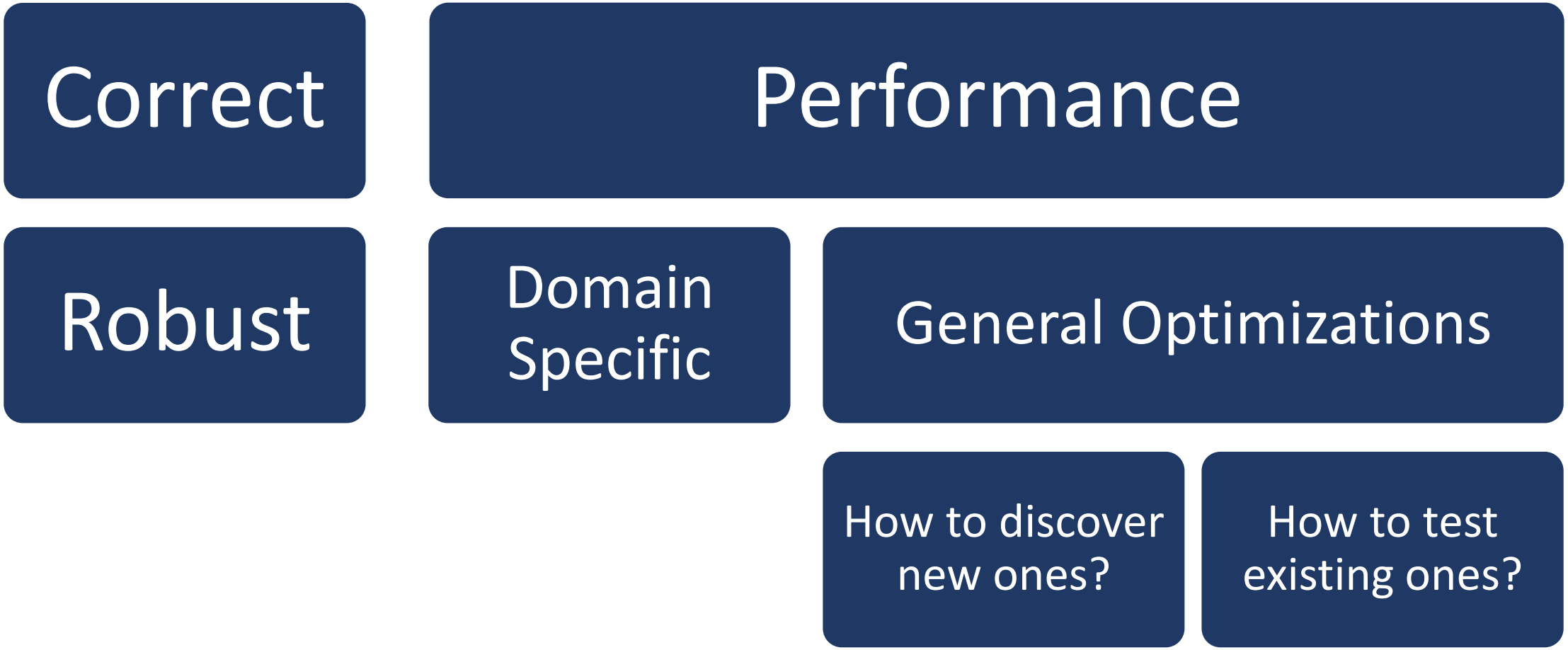
Robust

Domain Specific

General Optimizations

How to discover new ones?

The “Perfect” Compiler



The “Perfect” Compiler

Correct

Per

Robust

Domain Specific

General Optimizations

How to discover new ones?

Finding Missed Optimizations through the Lens of Dead Code Elimination

Theodoros Theodoridis
theodoros.theodoridis@inf.ethz.ch
ETH Zurich
Switzerland

Manuel Rigger
manuel.rigger@inf.ethz.ch
ETH Zurich
Switzerland

Zhendong Su
zhendong.su@inf.ethz.ch
ETH Zurich
Switzerland

ABSTRACT

Compilers are foundational software development tools and incorporate increasingly sophisticated optimizations. Due to their complexity, it is difficult to systematically identify opportunities for improving them. Indeed, the automatic discovery of missed optimizations has been an important and significant challenge. The few existing approaches either cannot accurately pinpoint missed optimizations or target only specific analyses. This paper tackles this challenge by introducing a novel, effective approach that — in a simple and general manner — automatically identifies a wide range

ACM Reference Format:
Theodoros Theodoridis, Manuel Rigger, and Zhendong Su. 2022. Finding Missed Optimizations through the Lens of Dead Code Elimination. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '22), February 28 – March 4, 2022, Lausanne, Switzerland.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3503222.3507764

1 INTRODUCTION

Both industry and academia have invested decades of effort to

How to test existing ones?

```c
static int a = 0;

int main () {
  if (a != 0) {
    return 1;
  }
  a = 1;
  return 0;
}
```

```c
static int a = 0;

int main () {
  if (a != 0) {
    return 1;
  }
  a = 1;
  return 0;
}
```

```asm
main:
 xorl %eax, %eax
 retq
```

C source #1

A Save/Load + Add new... Vim C

```c
static int a = 0;

int main () {
  if (a != 0) {
    return 1;
  }
  a = 1;
  return 0;
}
```

clang 14.0.0

x86-64 clang 14.0.0 -O3

A Output... Filter... Libraries + Add new... Add tool...

```
main:                                   # @main
        movl    $1, %eax
        cmpb    $0, a(%rip)
        jne     .LBB0_2
        movb    $1, a(%rip)
        xorl    %eax, %eax
.LBB0_2:
        retq
```

Output (0/0) x86-64 clang 14.0.0 - cached (7549B) ~158 lines filtered

gcc 11.3

x86-64 gcc 11.3 -O3

A Output... Filter... Libraries + Add new... Add tool...

```
main:
        movl    a(%rip), %eax
        testl   %eax, %eax
        jne     .L3
        movl    $1, a(%rip)
        ret
.L3:
        movl    $1, %eax
        ret
```

Output (0/0) x86-64 gcc 11.3 - cached (3047B) ~182 lines filtered

COMPILER EXPLORER Add... More Sponsors Backtrace intel. Solid Sands Share Policies Other

C source #1

A Save/Load + Add new... Vim C

```c
static int a = 0;

int main () {
  if (a != 0) {
    return 1;
  }
  a = 0;
  return 0;
}
```

(13, 3)

clang 14.0.0

x86-64 clang 14.0.0 -O3

A Output... Filter... Libraries + Add new... Add tool...

```
main:                                   # @main
        xorl    %eax, %eax
        retq
```

Output (0/0) x86-64 clang 14.0.0 - 545ms (7023B) ~139 lines filtered

gcc 11.3

x86-64 gcc 11.3 -O3

A Output... Filter... Libraries + Add new... Add tool...

```
main:
        movl    a(%rip), %edx
        xorl    %eax, %eax
        testl   %edx, %edx
        setne   %al
        ret
```

Output (0/0) x86-64 gcc 11.3 - 507ms (2837B) ~176 lines filtered

```c
static int a = 0;

int main () {
  if (a != 0) {
    return 1;
  }
  a = 1;
  return 0;
}
```

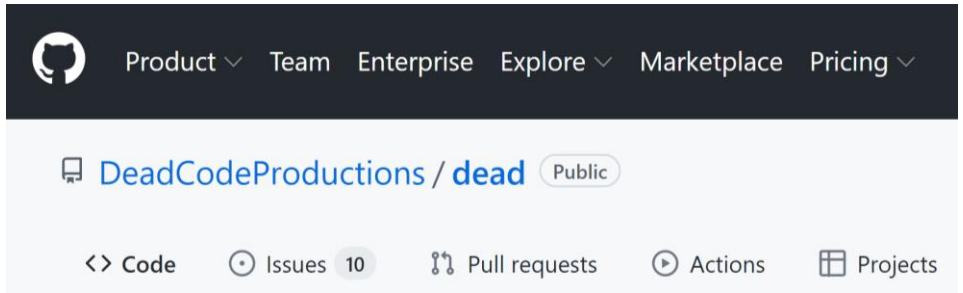
x86-64 clang 14.0.0 | -O3

```asm
main:                                   # @main
        movl    $1, %eax
        cmpb    $0, a(%rip)
        jne     .LBB0_2
        movb    $1, a(%rip)
        xorl    %eax, %eax
.LBB0_2:
        retq
```

x86-64 gcc 11.3 | -O3

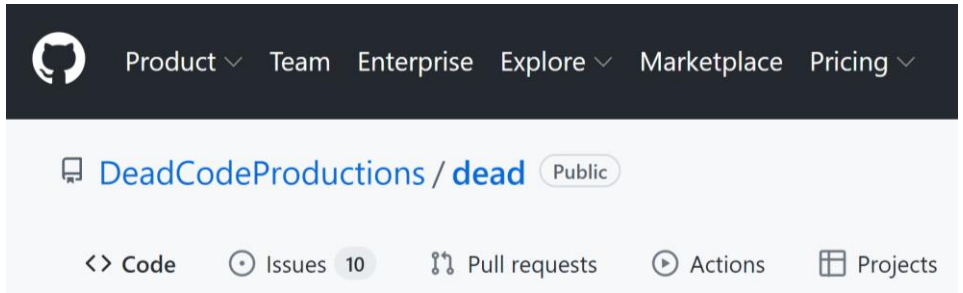
```asm
main:
        movl    a(%rip), %eax
        testl   %eax, %eax
        jne     .L3
        movl    $1, a(%rip)
        ret
.L3:
        movl    $1, %eax
        ret
```

Finding Missed Optimization Opportunities Automatically



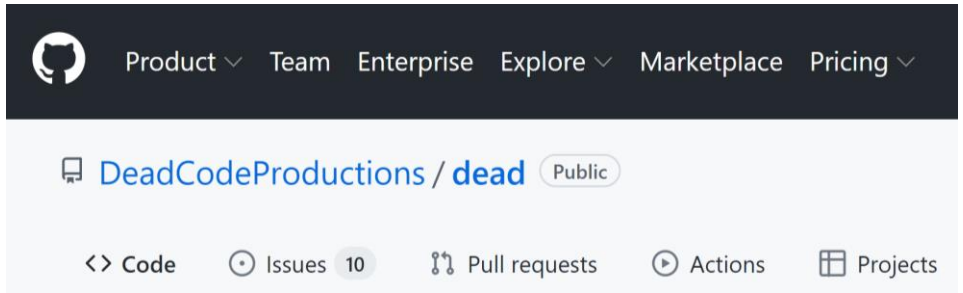
Finding Missed Optimization Opportunities Automatically

Through the Lens of Dead Code Elimination



Finding Missed Optimization Opportunities Automatically

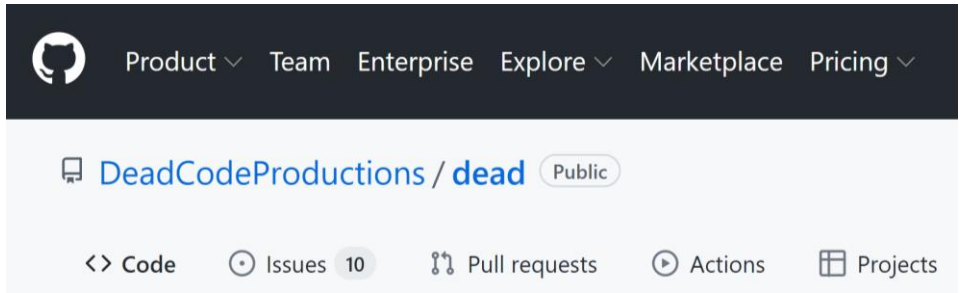
Through the Lens of Dead Code Elimination



	LLVM
Reported	47
Confirmed	35
Fixed	15

Finding Missed Optimization Opportunities Automatically

Through the Lens of Dead Code Elimination

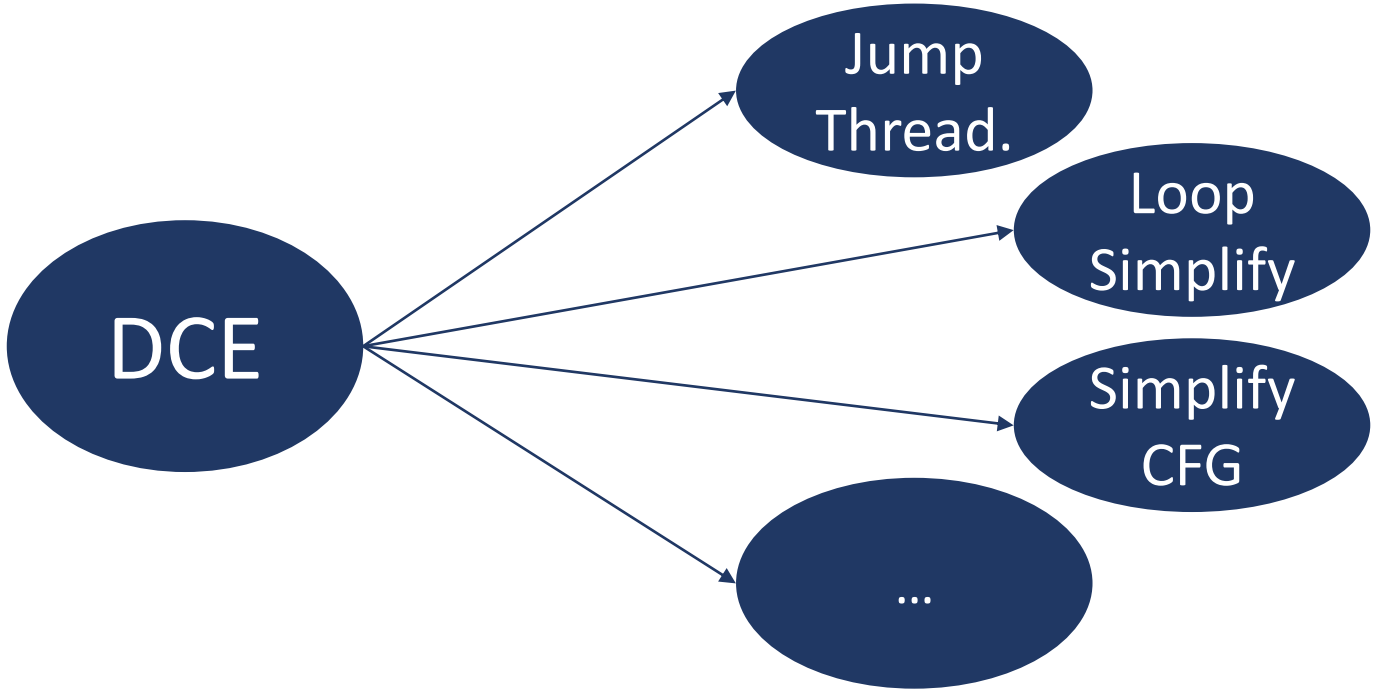


	LLVM	GCC
Reported	47	55
Confirmed	35	46
Fixed	15	15

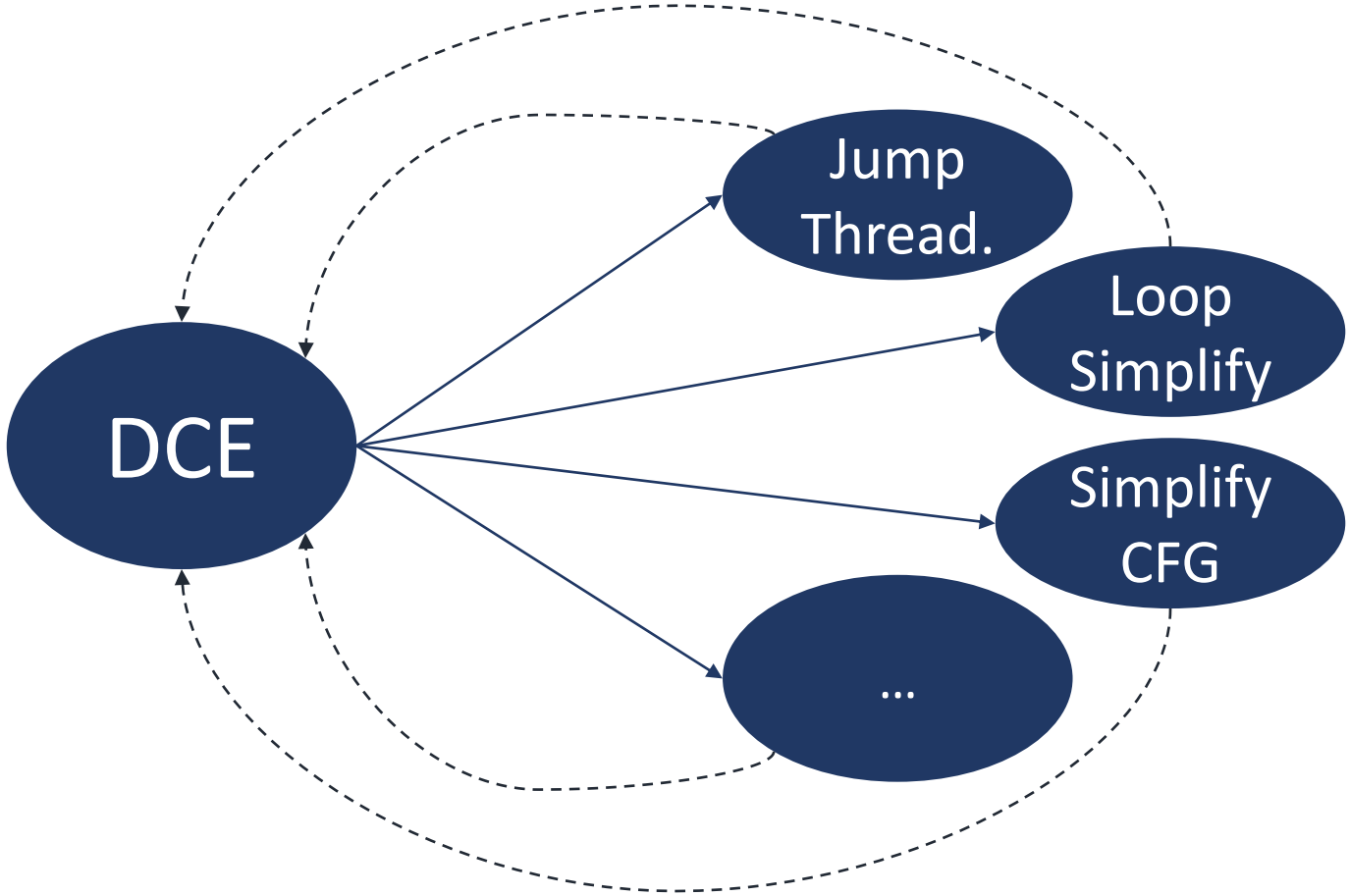
Lens of Dead Code Elimination

Dead Code Elimination Is Important

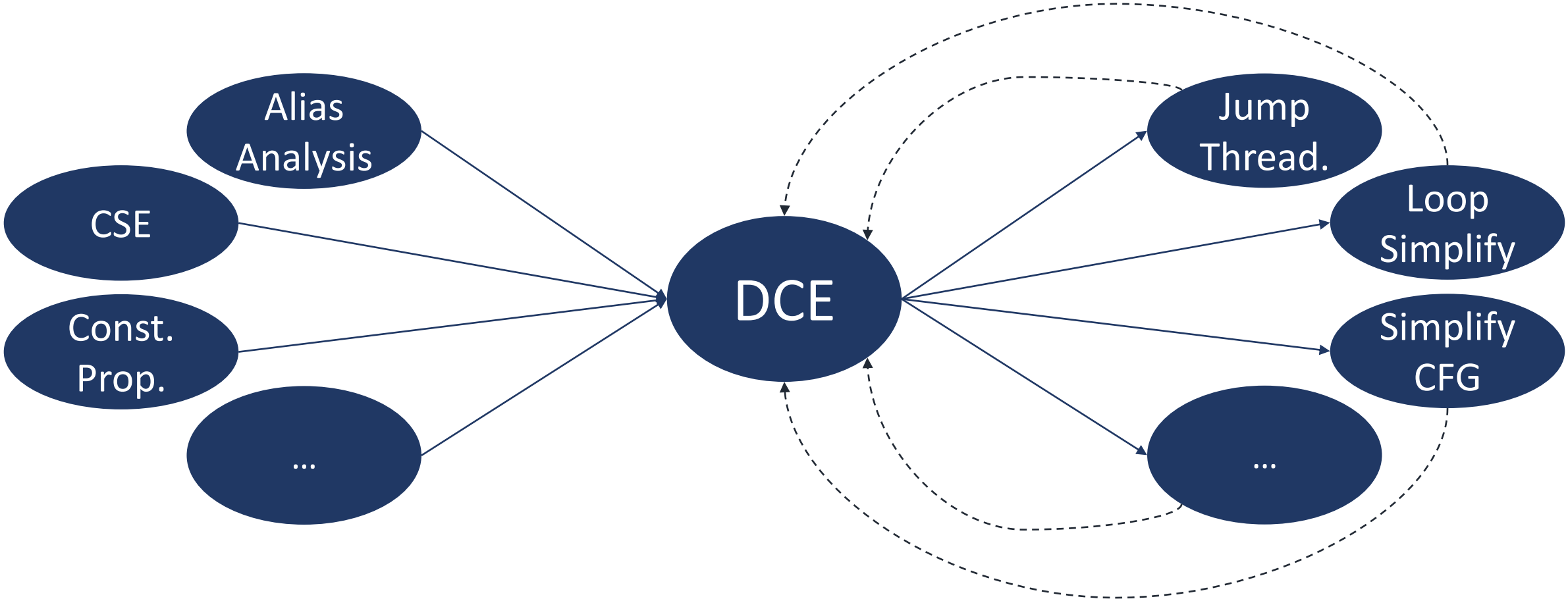
Dead Code Elimination Is Important



Dead Code Elimination Is Important



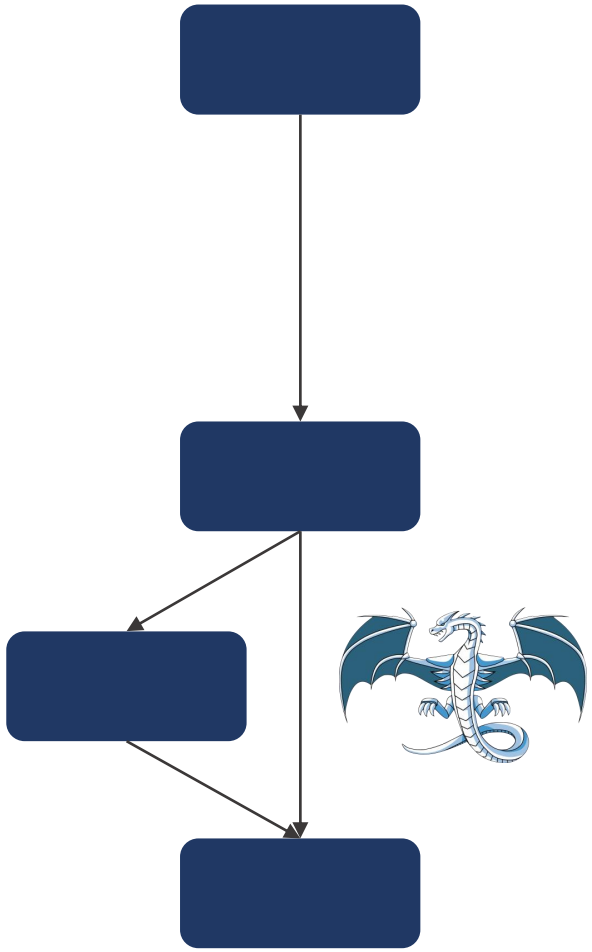
Dead Code Elimination Is Important

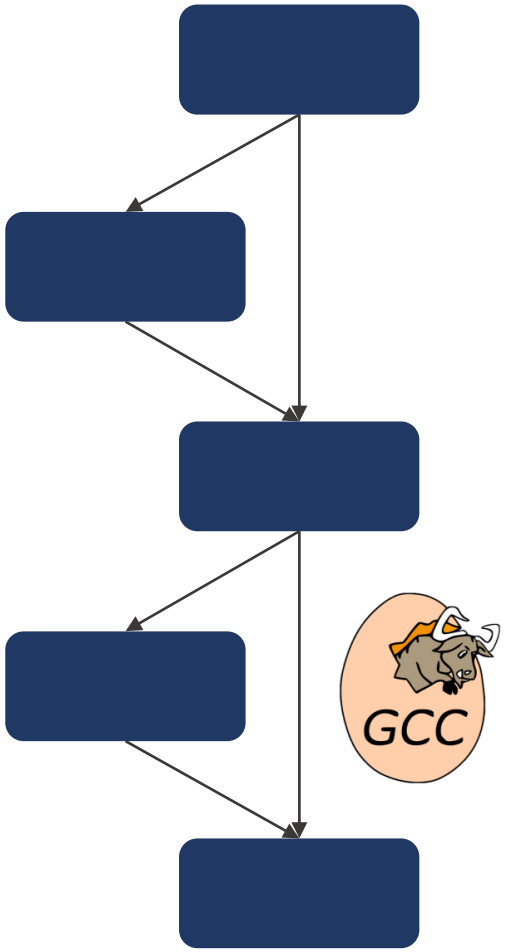


Different Compilers Eliminate Different Parts

Different Compilers Eliminate Different Parts

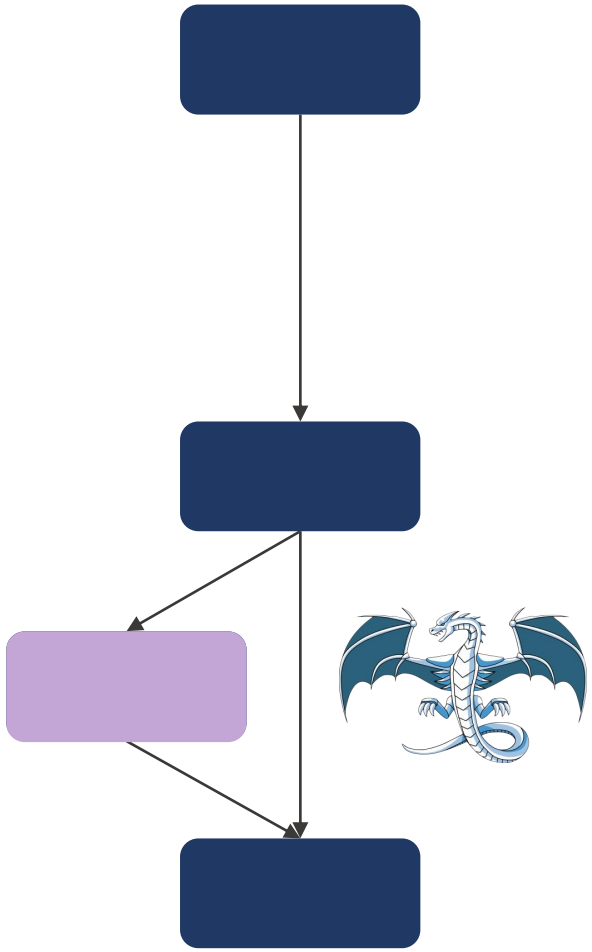
```
int a = 0;
static int b[2] = {0,0}, c = 0;

int main() {
  if (b[a]) {
    return 1;
  }
  if (c) {
    return 2;
  }
  c = 1;
  return 0;
}
```

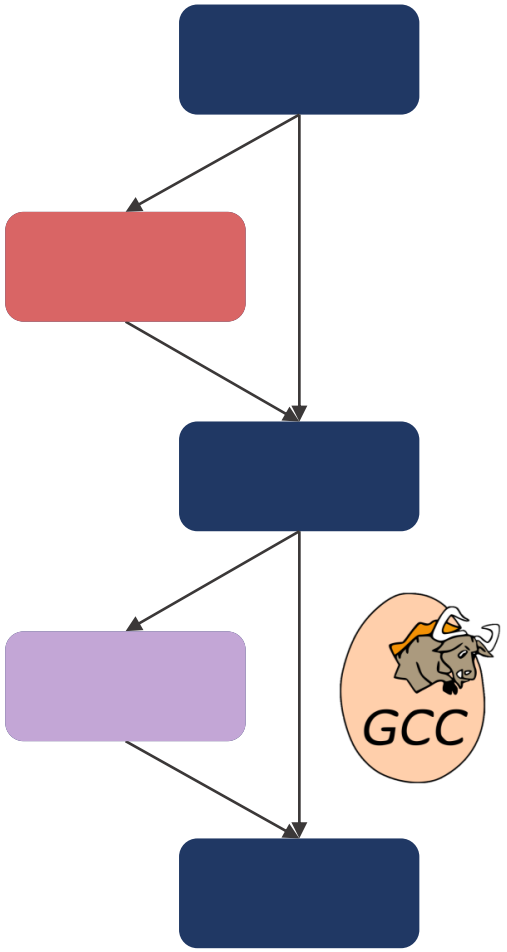




Different Compilers Eliminate Different Parts

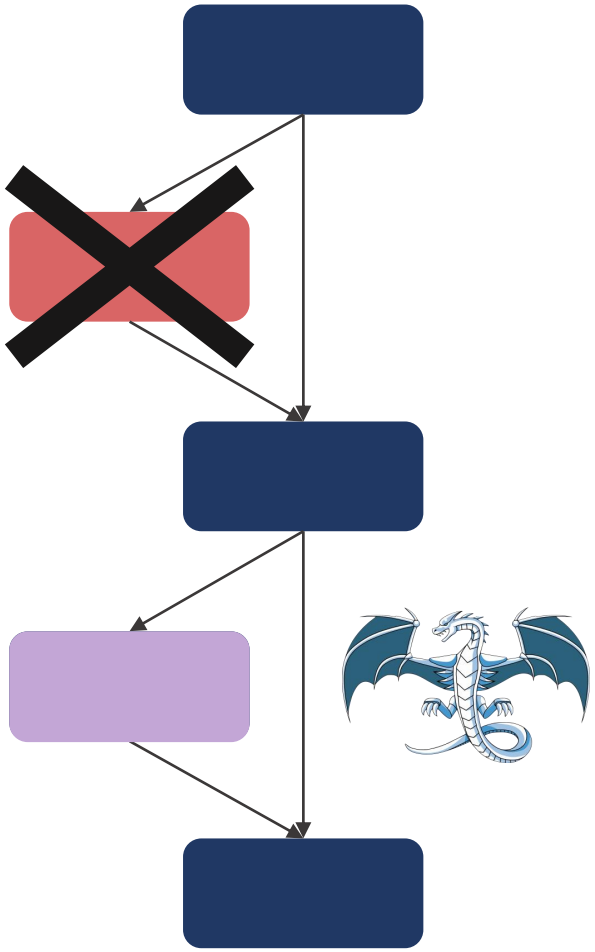
```
int a = 0;
static int b[2] = {0,0}, c = 0;

int main() {
  if (b[a]) {
    return 1;
  }
  if (c) {
    return 2;
  }
  c = 1;
  return 0;
}
```

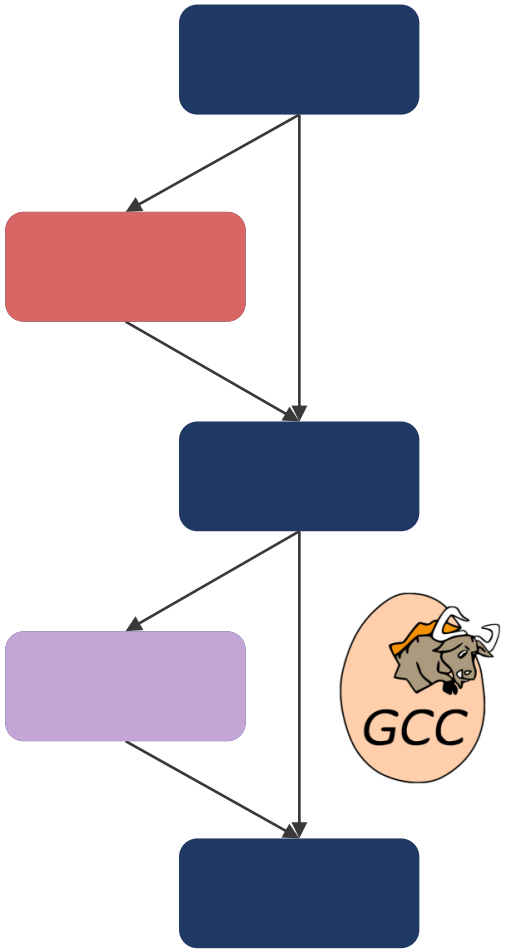




Different Compilers Eliminate Different Parts

```
int a = 0;
static int b[2] = {0,0}, c = 0;

int main() {
  if (b[a]) {
    return 1;
  }
  if (c) {
    return 2;
  }
  c = 1;
  return 0;
}
```





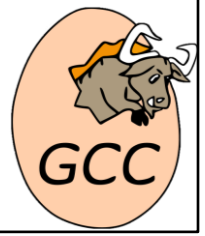
Missed Dead Code Elimination Detection

Missed Dead Code Elimination Detection

```
int a = 0;
static int b[2] = {0,0}, c = 0;

int main() {
  if (b[a]) {
    return 1;
  }
  if (c) {
    return 2;
  }
  c = 1;
  return 0;
}
```

```
main:
 movl  $2, %eax
 cmpb  $0, c(%rip)
 jne  .LBB0_2
 movb  $1, c(%rip)
 xorl  %eax, %eax
.LBB0_2:
 retq
```

```
main:
 movslq a(%rip), %rdx
 movl   $1, %eax
 movl   b(,%rdx,4),%edx
 testl  %edx, %edx
 jne    .L1
 movl   c(%rip), %eax
 testl  %eax, %eax
 jne    .L4
 movl   $1, c(%rip)
 ret
.L4:
 movl   $2, %eax
.L1:
 ret
```



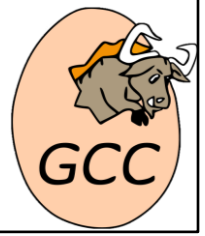
Missed Dead Code Elimination Detection

Differential Testing!

```
int a = 0;
static int b[2] = {0,0}, c = 0;

int main() {
  if (b[a]) {
    return 1;
  }
  if (c) {
    return 2;
  }
  c = 1;
  return 0;
}
```

```
main:
 movl  $2, %eax
 cmpb  $0, c(%rip)
 jne   .LBB0_2
 movb  $1, c(%rip)
 xorl  %eax, %eax
.LBB0_2:
 retq
```

```
main:
 movslq a(%rip), %rdx
 movl   $1, %eax
 movl   b(,%rdx,4),%edx
 testl  %edx, %edx
 jne    .L1
 movl   c(%rip), %eax
 testl  %eax, %eax
 jne    .L4
 movl   $1, c(%rip)
 ret
.L4:
 movl   $2, %eax
.L1:
 ret
```



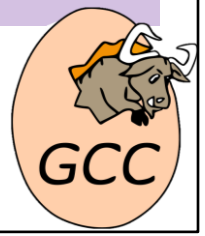
Missed Dead Code Elimination Detection

Differential Testing!

```
int a = 0;
static int b[2] = {0,0}, c = 0;

int main() {
  if (b[a]) {
    return 1;
  }
  if (c) {
    return 2;
  }
  c = 1;
  return 0;
}
```

```
main:
 movl   $2, %eax
 cmpb   $0, c(%rip)
 jne   .LBB0_2
 movb   $1, c(%rip)
 xorl   %eax, %eax
.LBB0_2:
 retq
```

```
main:
 movslq a(%rip), %rdx
 movl   $1, %eax
 movl   b(,%rdx,4),%edx
 testl  %edx, %edx
 jne    .L1
 movl   c(%rip), %eax
 testl  %eax, %eax
 jne    .L4
 movl   $1, c(%rip)
 ret
.L4:
 movl   $2, %eax
.L1:
 ret
```



Missed Dead Code Elimination: Markers

Missed Dead Code Elimination: Markers

```
int a = 0;
static int b[2] = {0,0}, c = 0;

int main() {
  if (b[a]) {
    return 1;
  }
  if (c) {
    return 2;
  }
  c = 1;
  return 0;
}
```

Missed Dead Code Elimination: Markers

```
int a = 0;
static int b[2] = {0,0}, c = 0;
int main() {
  if (b[a]) {
    DCEMarker1();
    return 1;
  }
  if (c) {
    DCEMarker2();
    return 2;
  }
  c = 1;
  return 0;
}
```

Missed Dead Code Elimination: Markers

```
int a = 0;
static int b[2] = {0,0}, c = 0;
int main() {
  if (b[a]) {
    DCEMarker1();
    return 1;
  }
  if (c) {
    DCEMarker2();
    return 2;
  }
  c = 1;
  return 0;
}
```

```
main:
 subq    $8, %rsp
 movslq  a(%rip), %rax
 movl    b(,%rax,4),%eax
 testl   %eax, %eax
 jne     .L7
 …
.L7:
 call    DCEMarker1
 movl    $1, %eax
 jmp     .L1
.L8:
 call    DCEMarker2
 movl    $2, %eax
 jmp     .L1
```

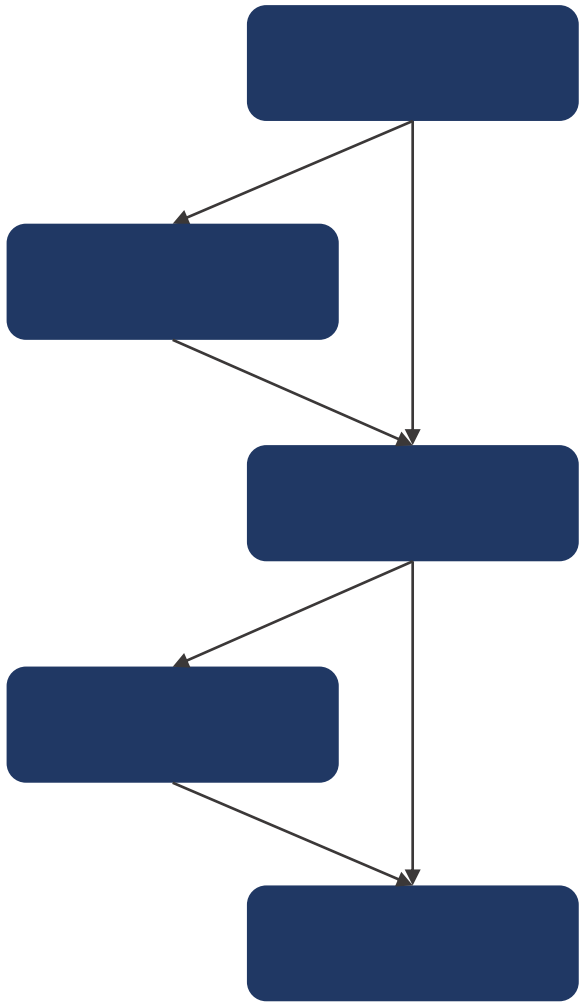
Missed Dead Code Elimination: Markers

```
int a = 0;
static int b[2] = {0,0}, c = 0;
int main() {
  if (b[a]) {
    DCEMarker1();
    return 1;
  }
  if (c) {
    DCEMarker2();
    return 2;
  }
  c = 1;
  return 0;
}
```

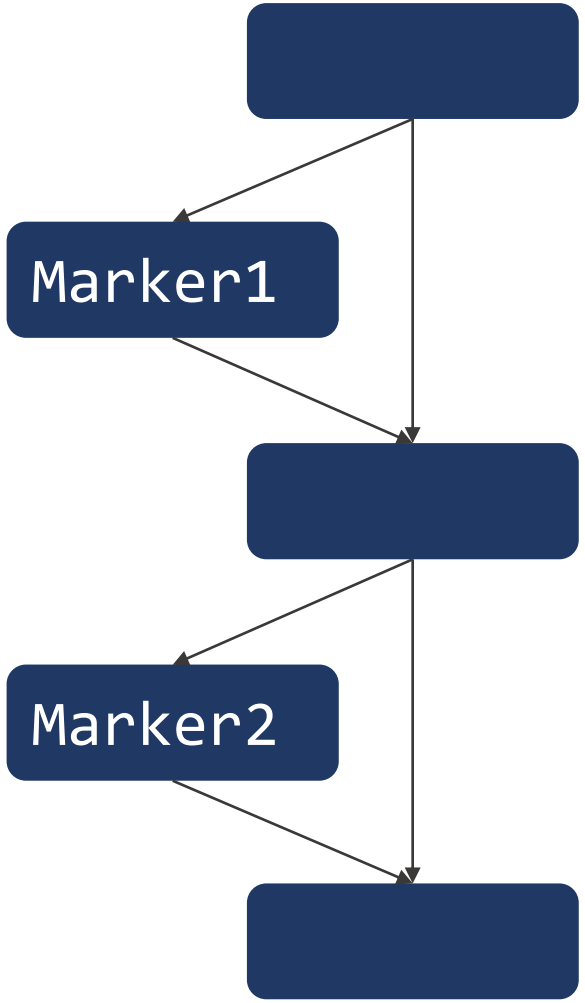
```
main:
  pushq   %rax
  cmpb    $1, c(%rip)
  jne     .LBB0_2
  callq   DCEMarker2
  movl    $2, %eax
  popq    %rcx
  retq
.LBB0_2:
  movb    $1, c(%rip)
  xorl    %eax, %eax
  popq    %rcx
  retq
```

```
main:
  subq    $8, %rsp
  movslq  a(%rip), %rax
  movl    b(,%rax,4),%eax
  testl   %eax, %eax
  jne     .L7
  …
.L7:
  call    DCEMarker1
  movl    $1, %eax
  jmp     .L1
.L8:
  call    DCEMarker2
  movl    $2, %eax
  jmp     .L1
```

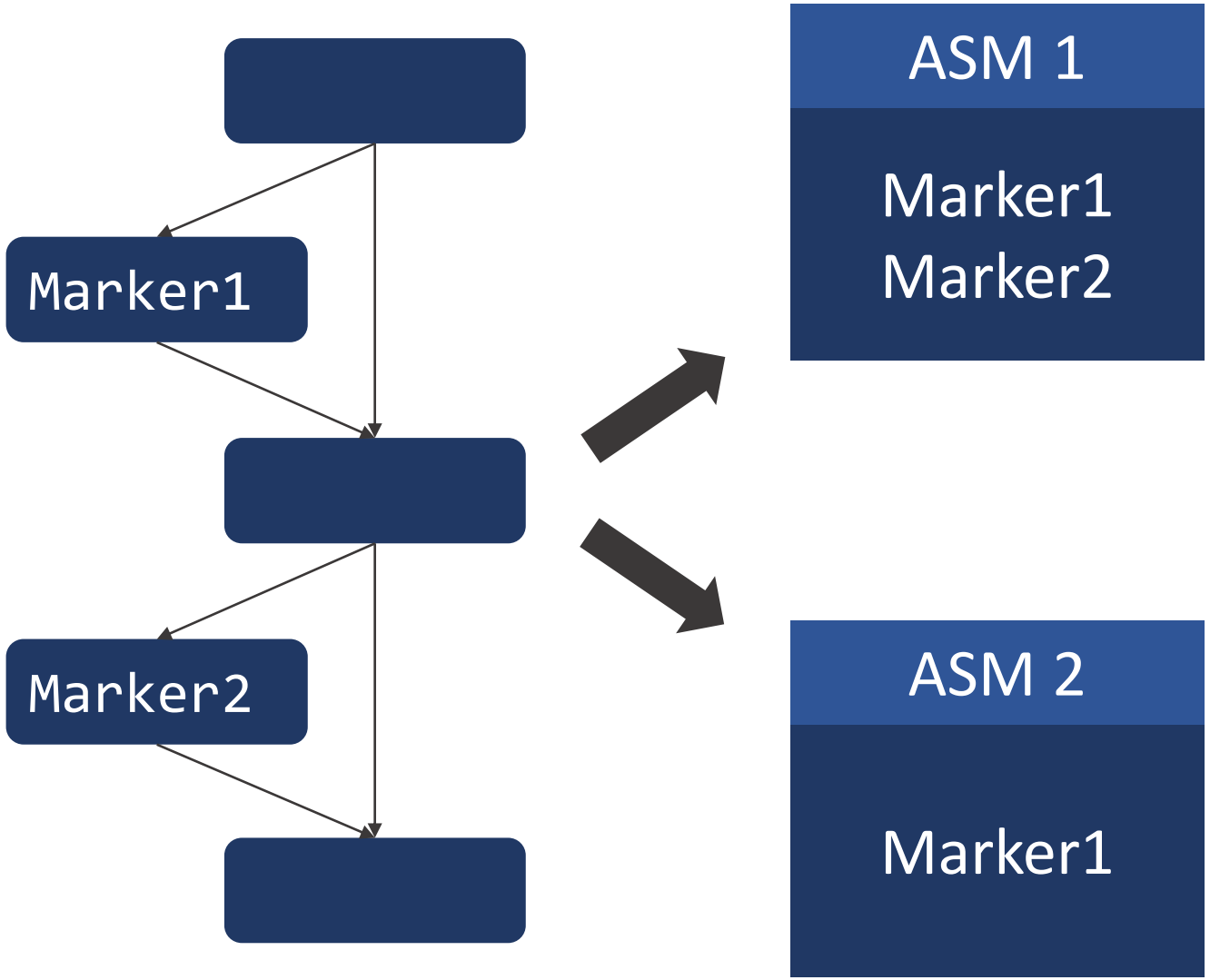
The Lens of Dead Code Elimination



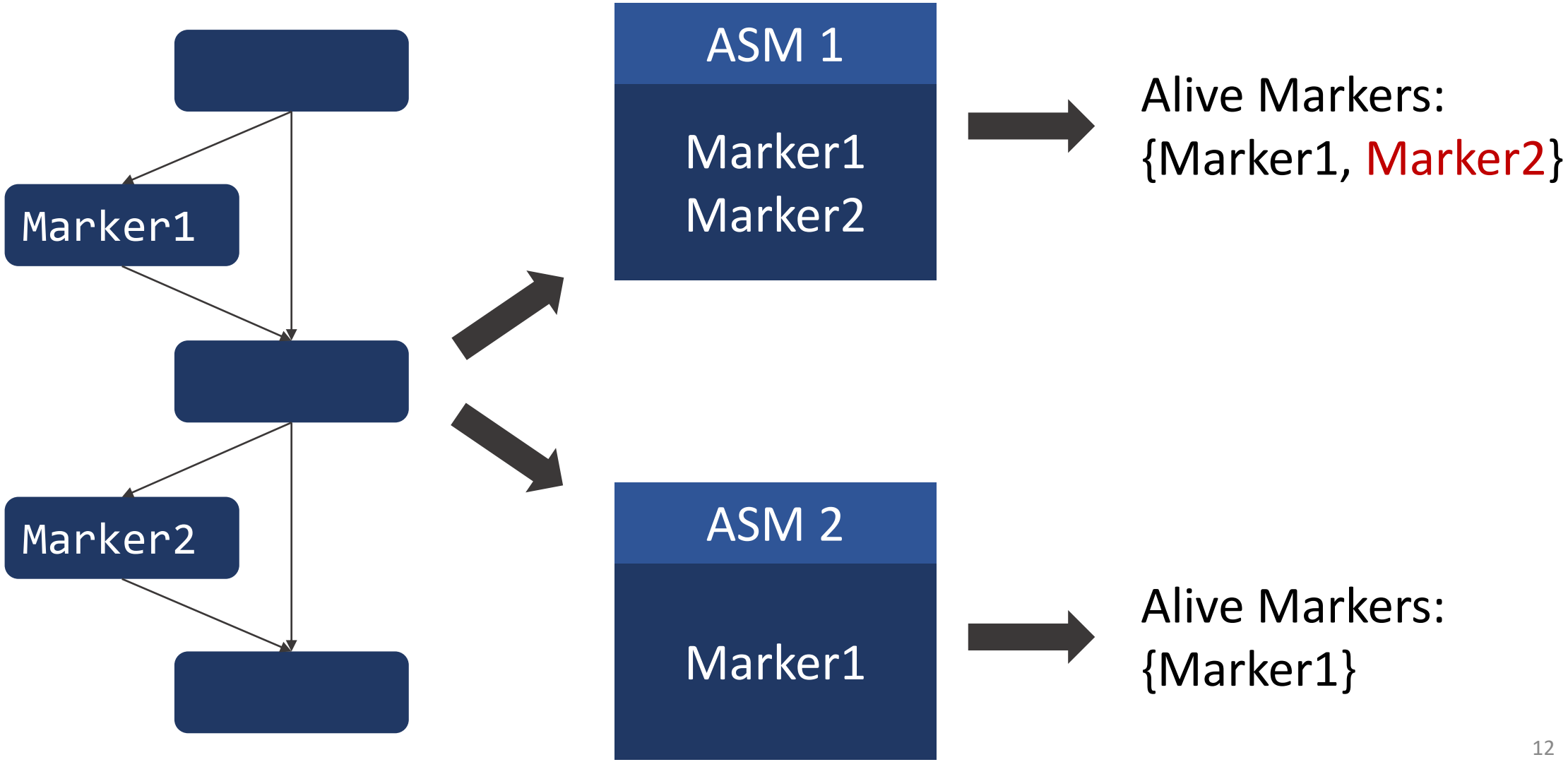
The Lens of Dead Code Elimination



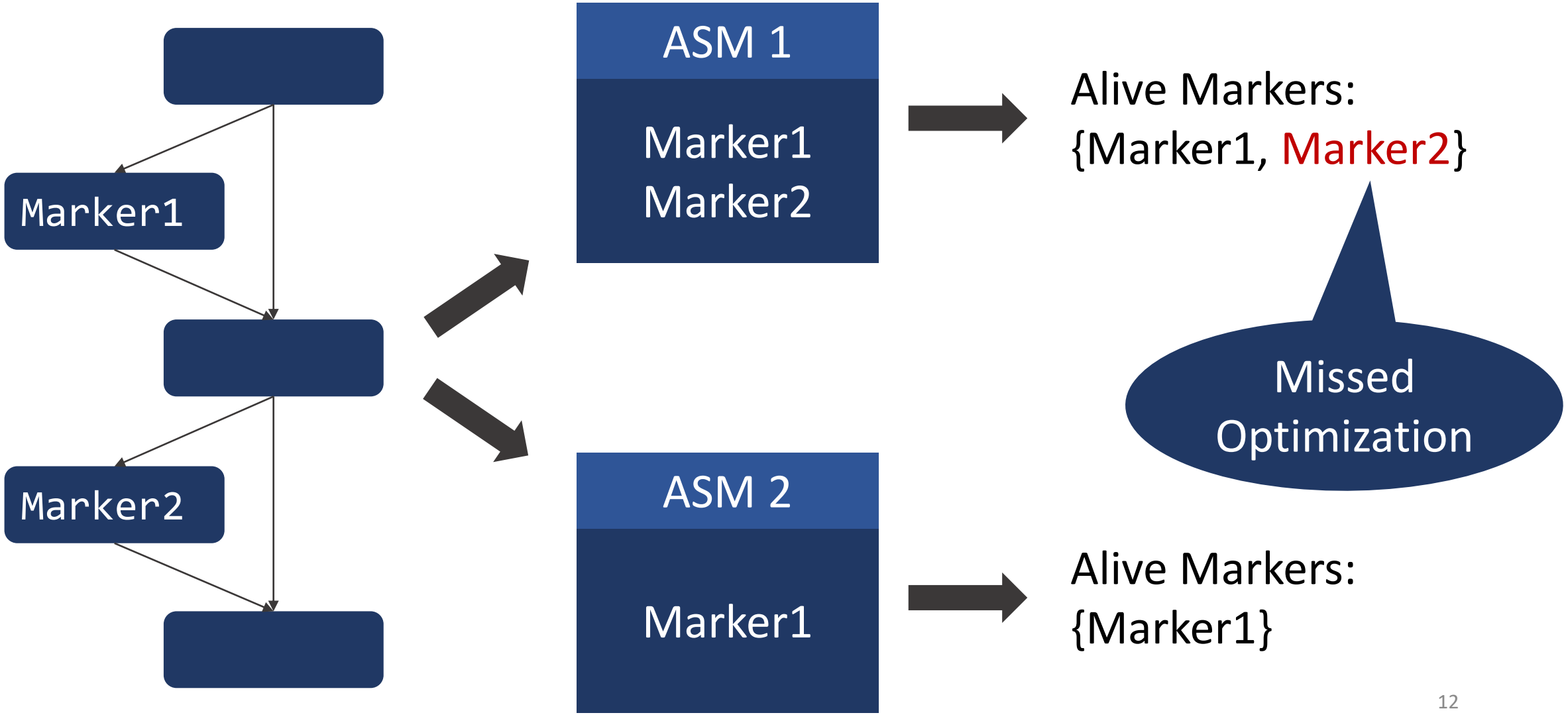
The Lens of Dead Code Elimination



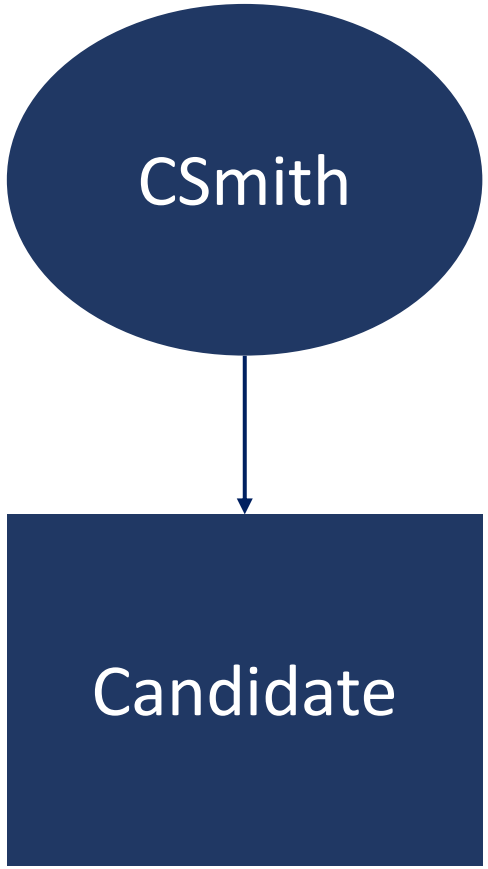
The Lens of Dead Code Elimination

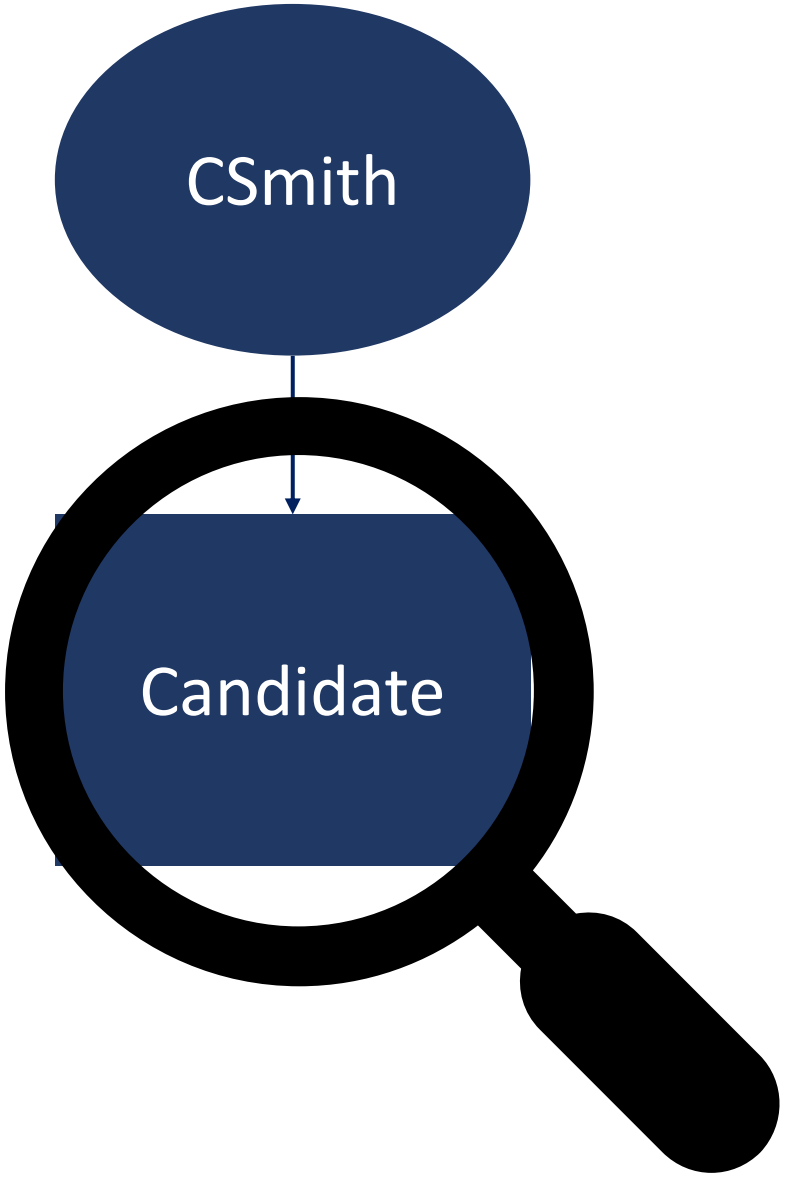


The Lens of Dead Code Elimination



End-to-End Automated Testing

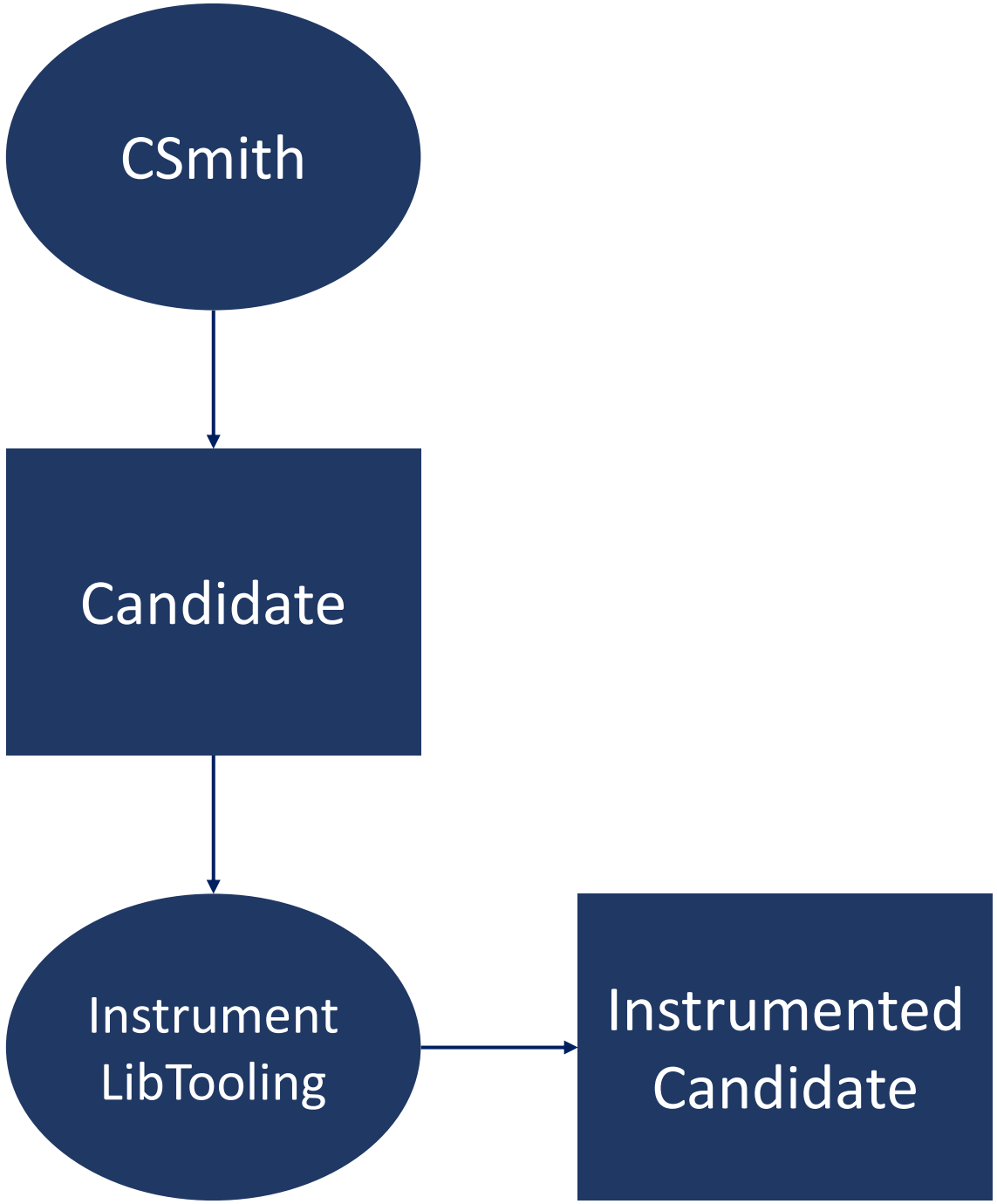

CSmith
Candidate




```
struct S1 l_74 = {1U, 1, 0U, 5, 4294967292U};
struct S0 l_76 = {7, 0, 0x57};
for (p_28 = 0; (p_28 > (-13)); --p_28) {
    g_75 = l_74;
}
g_71 = l_76;
for (p_28 = (-29); (p_28 == (-1)); p_28++) {
  int32_t l_79 = 0;
  uint32_t l_80 = 0xB5F43E01;
  if (p_28) {
    break;
  }
  --l_80;
  }
```

```
struct S1 l_74 = {1U, 1, 0U, 5, 4294967292U};
struct S0 l_76 = {7, 0, 0x57};
for (p_28 = 0; (p_28 > (-13)); --p_28) {
    g_75 = l_74;
}
g_71 = l_76;
for (p_28 = (-29); (p_28 == (-1)); p_28++) {
  int32_t l_79 = 0;
  uint32_t l_80 = 0xB5F43E01;
  if (p_28) {
    break;
  }
  --l_80;
  }
```

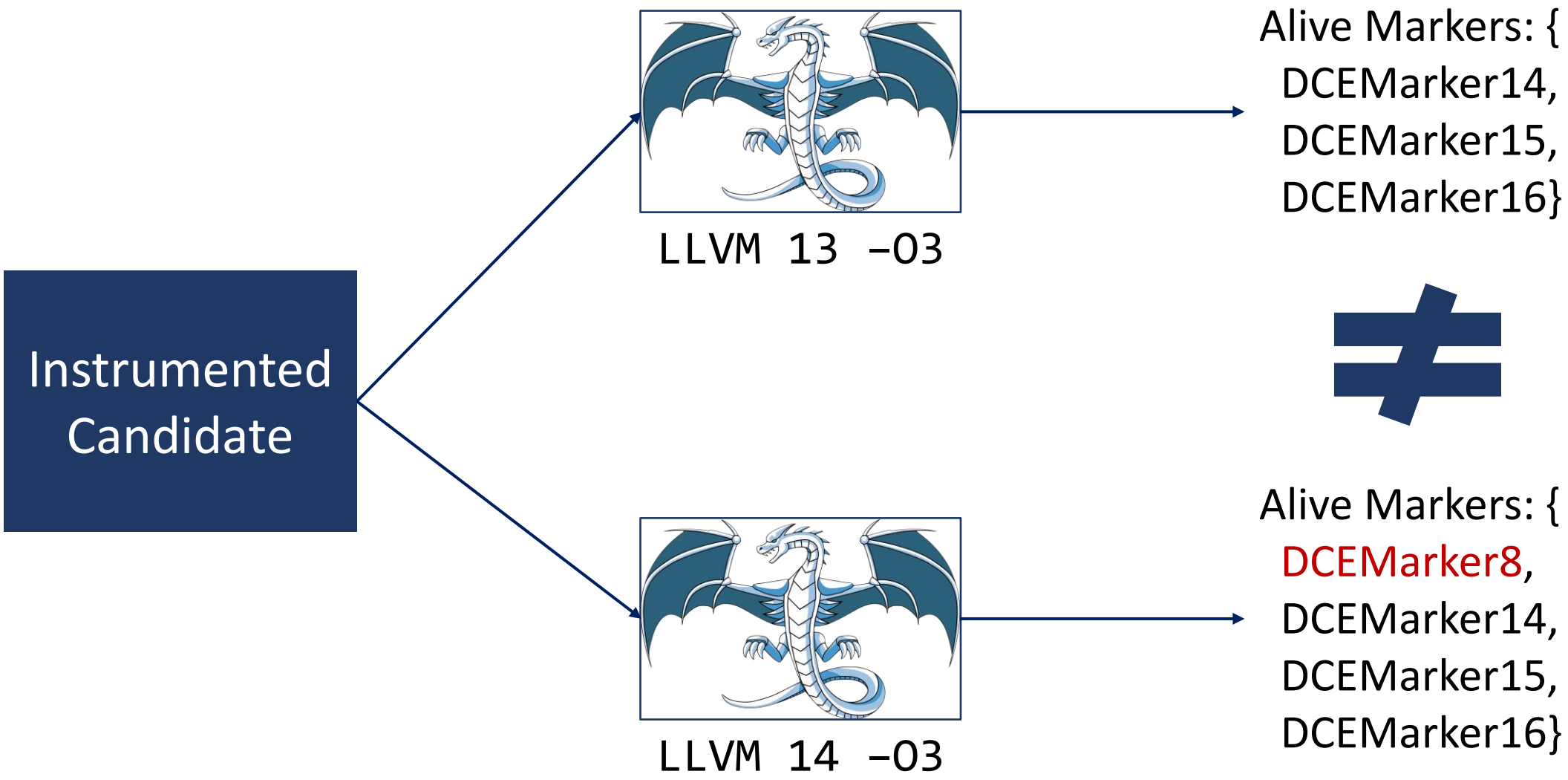
```
struct S1 l_74 = {1U, 1, 0U, 5, 4294967292U};
struct S0 l_76 = {7, 0, 0x57};
for (p_28 = 0; (p_28 > (-13)); --p_28) {
    DCEMarker14();
    g_75 = l_74;
}
g_71 = l_76;
for (p_28 = (-29); (p_28 == (-1)); p_28++) {
  DCEMarker15();
  int32_t l_79 = 0;
  uint32_t l_80 = 0xB5F43E01;
  if (p_28) {
    DCEMarker16();
    break;
  }
  --l_80;
  }
```

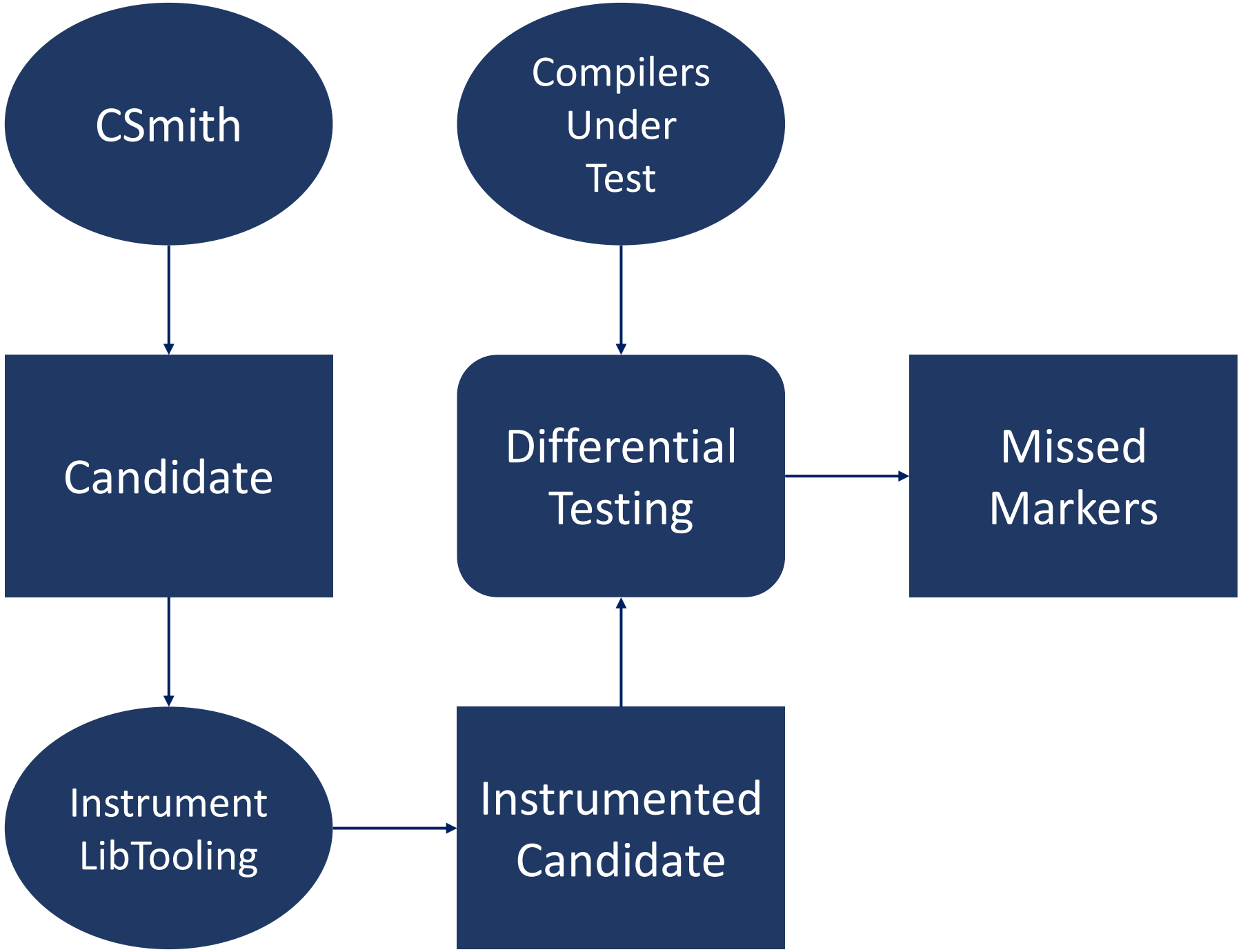

CSmith
Candidate
Instrument
LibTooling
Instrumented
Candidate

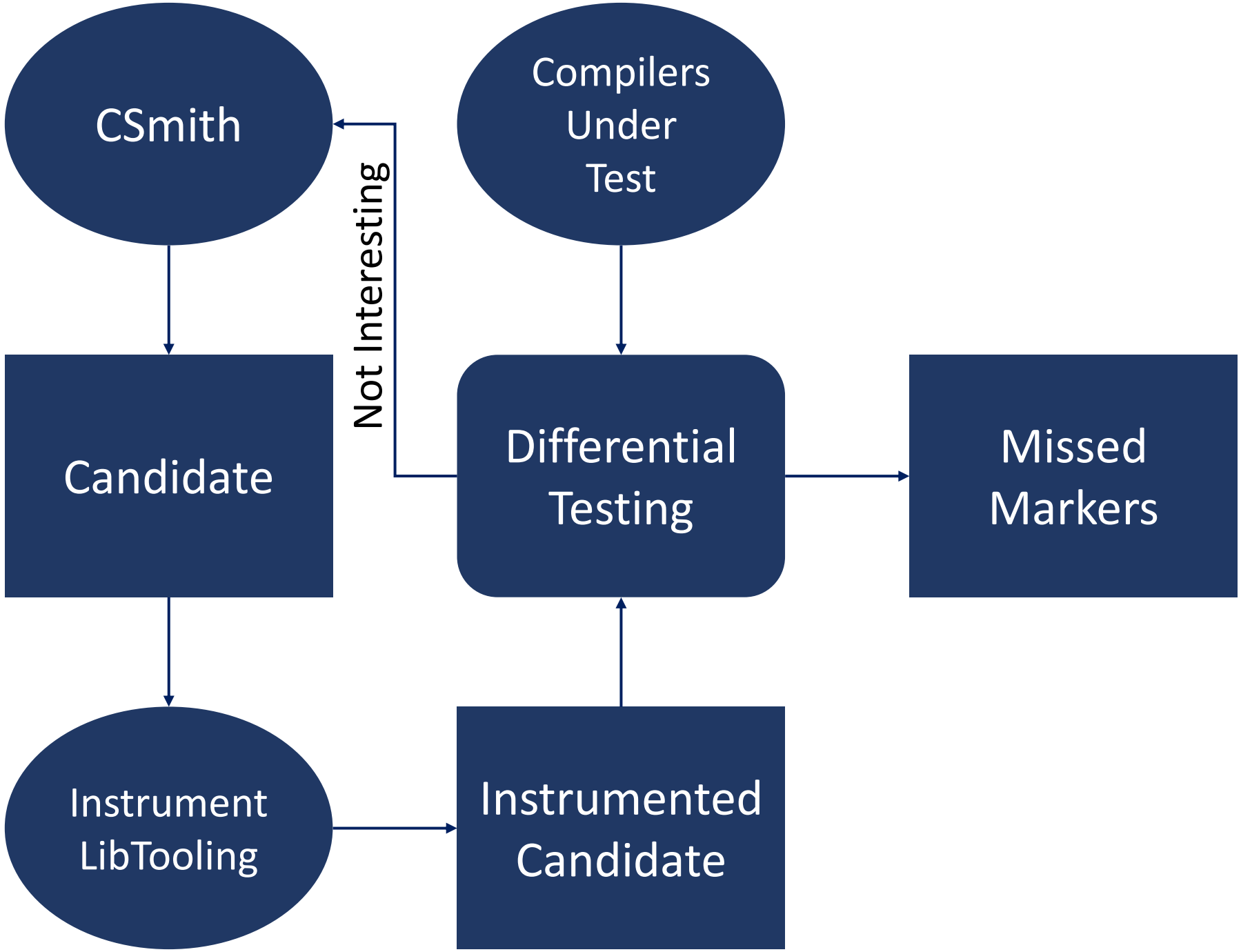
Instrumented Candidate

LLVM 13 -O3

Alive Markers: {
DCEMarker14,
DCEMarker15,
DCEMarker16}


Instrumented Candidate
LLVM 13 -O3
Alive Markers: {
DCEMarker14,
DCEMarker15,
DCEMarker16}
≠
LLVM 14 -O3
Alive Markers: {
DCEMarker8,
DCEMarker14,
DCEMarker15,
DCEMarker16}


CSmith
Compilers Under Test
Candidate
Differential Testing
Missed Markers
Instrument LibTooling
Instrumented Candidate

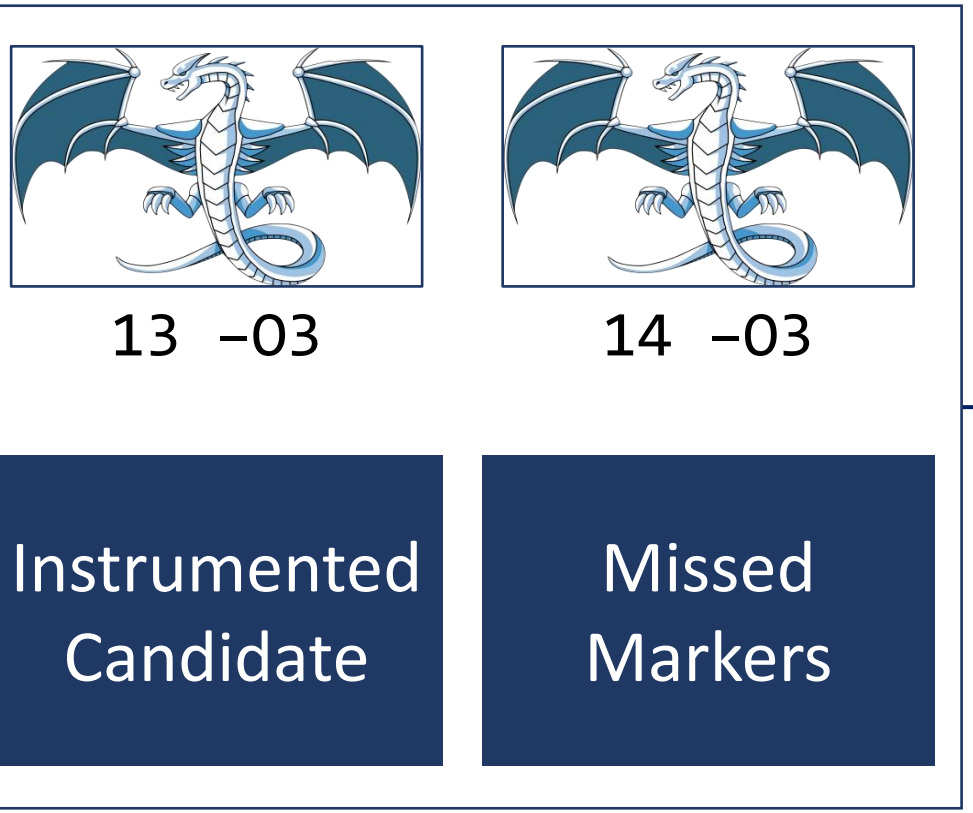

CSmith
Compilers Under Test
Not Interesting
Candidate
Differential Testing
Missed Markers
Instrument LibTooling
Instrumented Candidate

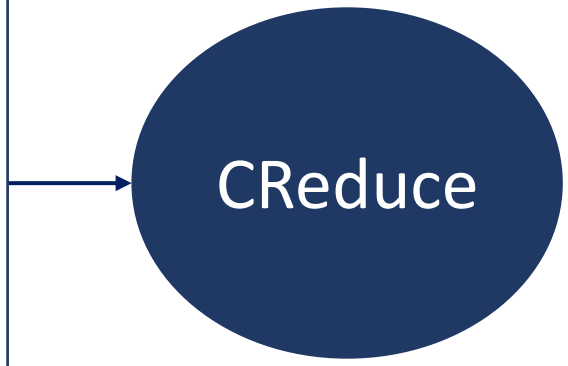
13 –03

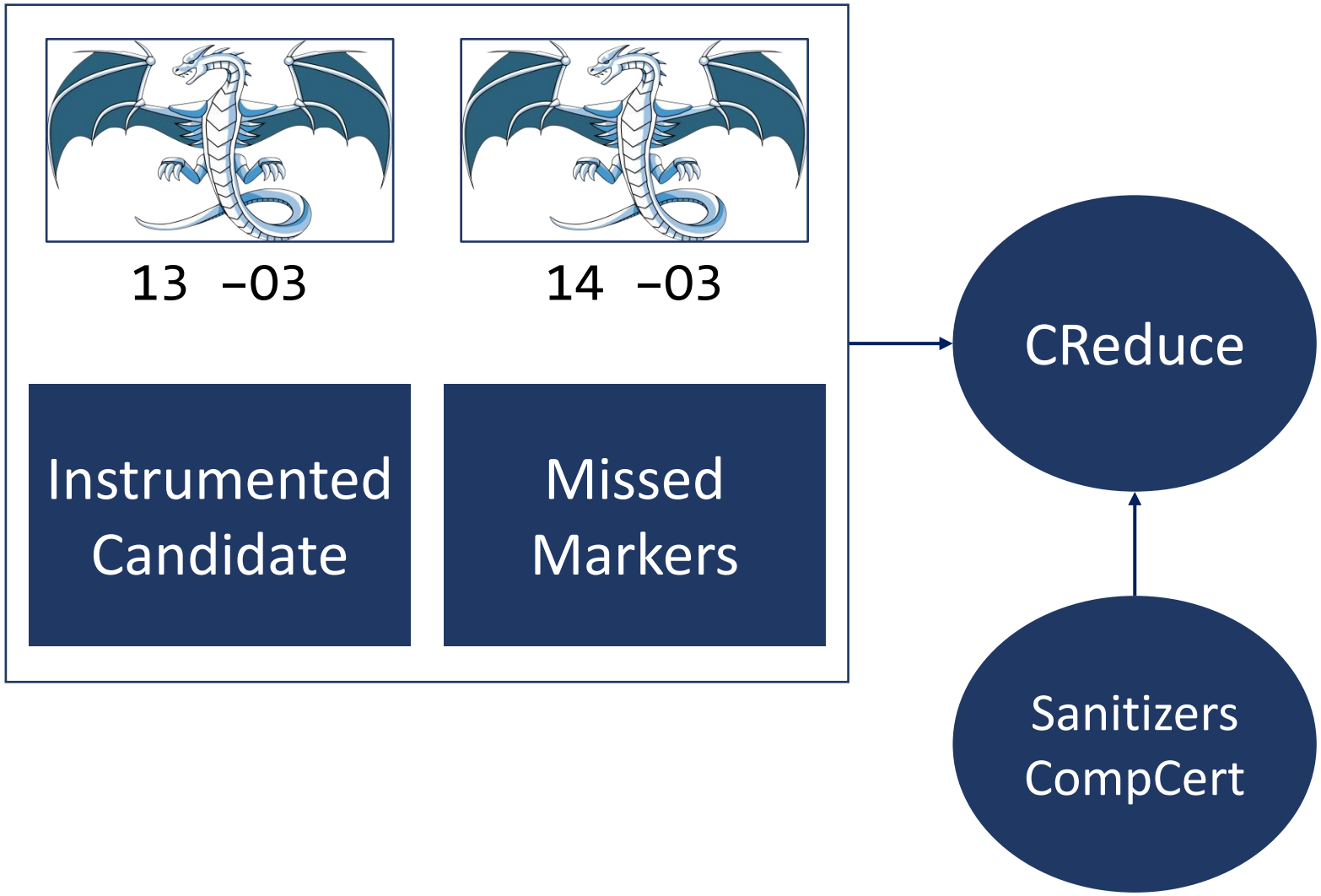
14 –03

Instrumented Candidate

Missed Markers


13 –03
14 –03
Instrumented Candidate
Missed Markers


CReduce


13 -03
14 -03
Instrumented Candidate
Missed Markers
CReduce
Sanitizers CompCert

13 –03

14 –03

Instrumented Candidate

Missed Markers

CReduce

Sanitizers
CompCert

```
void DCEMarker8(void);
void DCEMarker14(void);
static short a = 1, b;

int main() {
  a = -2;
  DCEMarker14();
  short c = -a;
  b = c % a;
  if (b)
    DCEMarker8();
}
```

```
void DCEMarker8(void);
void DCEMarker14(void);
static short a = 1, b;

int main() {
  a = -2;
  DCEMarker14();
  short c = -a;
  b = c % a;
  if (b)
    DCEMarker8();
}
```

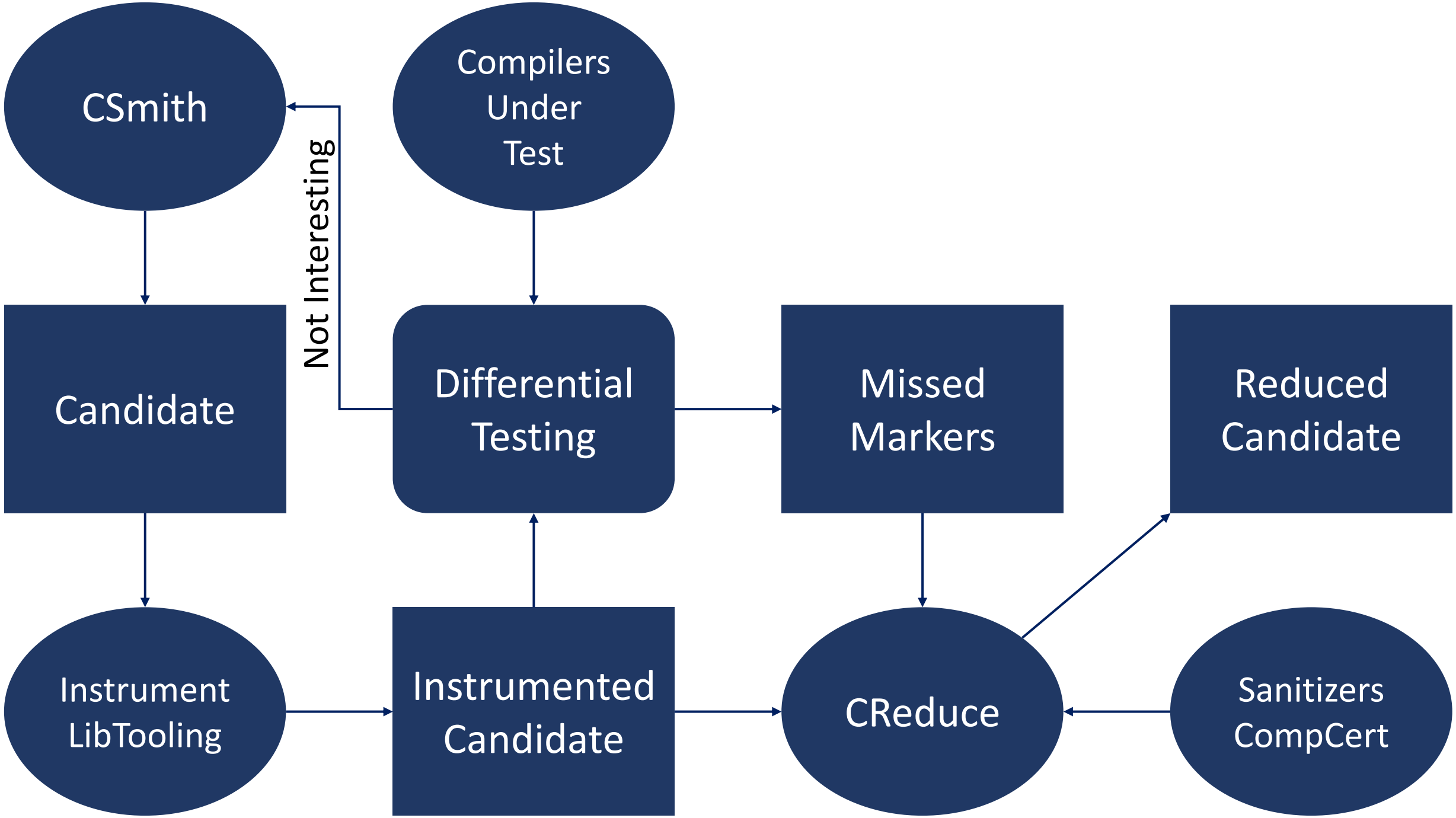
LLVM 13 -O3

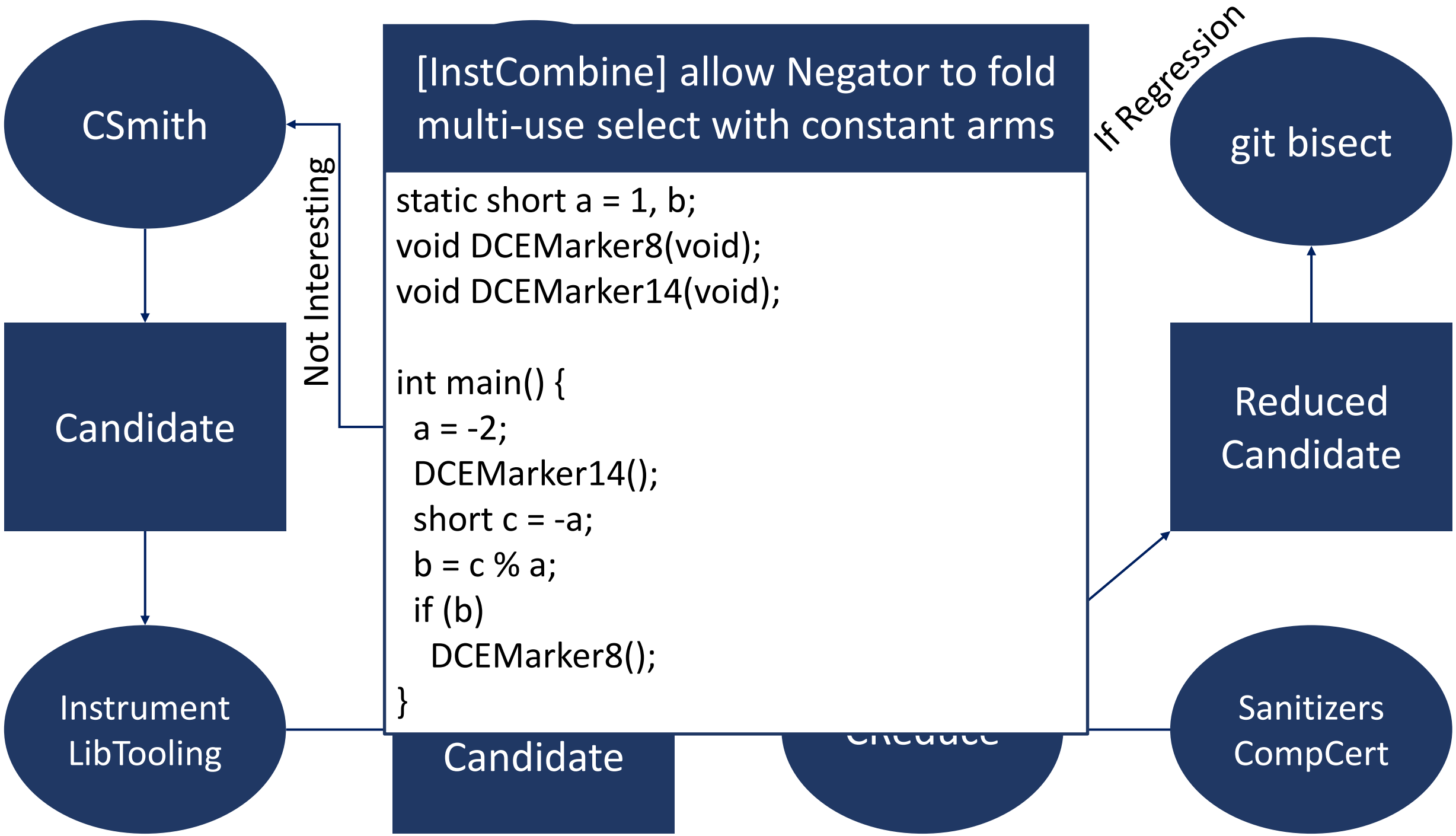
```
main:
  pushq   %rax
  callq   DCEMarker14
  xorl    %eax, %eax
  popq    %rcx
  retq
```

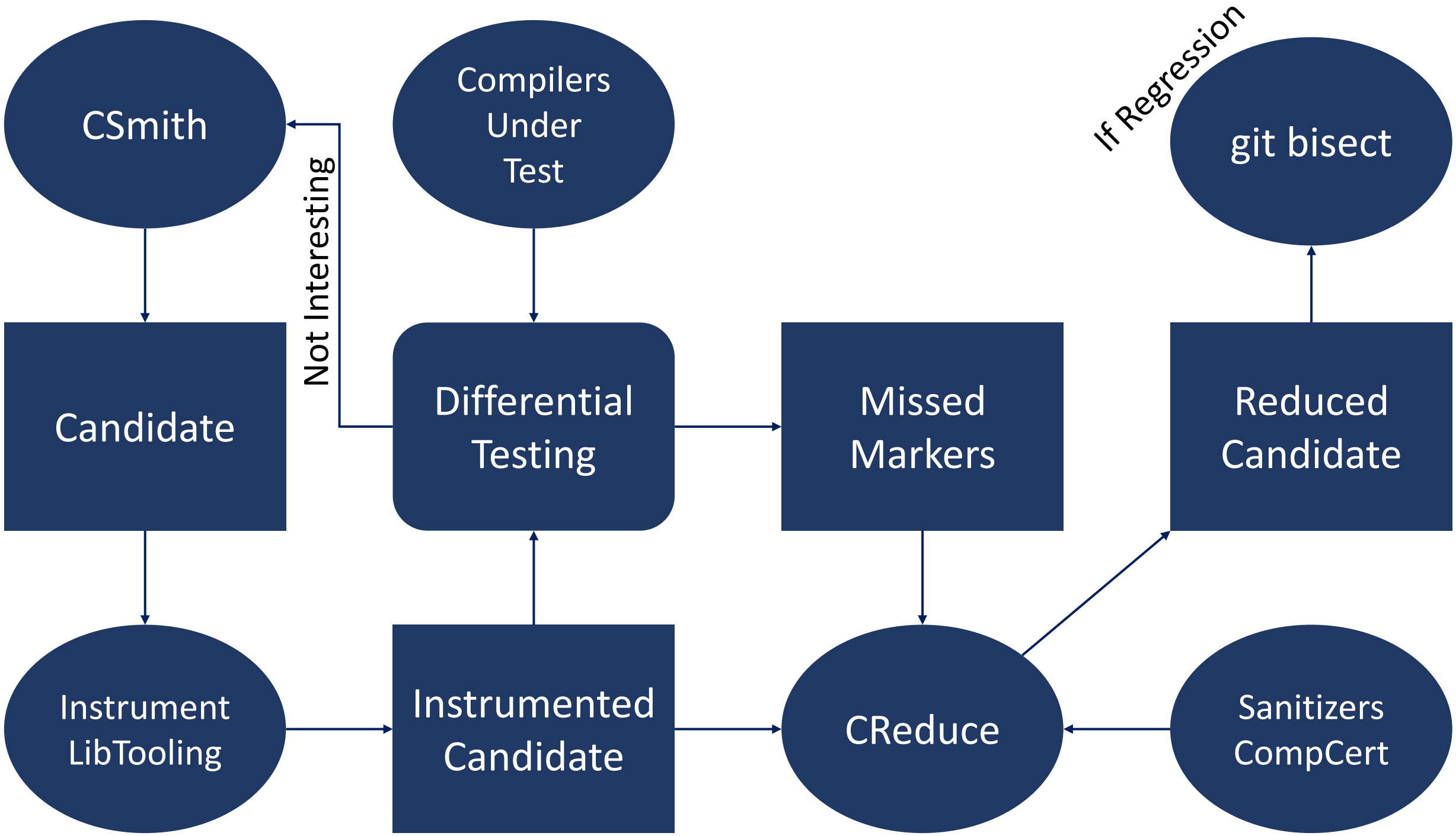
```
void DCEMarker8(void);
void DCEMarker14(void);
static short a = 1, b;

int main() {
  a = -2;
  DCEMarker14();
  short c = -a;
  b = c % a;
  if (b)
    DCEMarker8();
}
```

LLVM 14 -O3

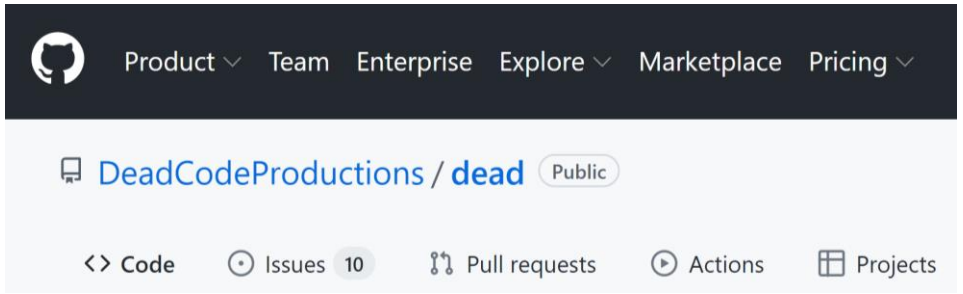
```
main:
 pushq   %rax
 movb    $1, a(%rip)
 callq   DCEMarker14
 cmpb    $0, a(%rip)
 movl    $2, %eax
 movl    $255, %ecx
 cmovnel %eax, %ecx
 movl    $254, %eax
 movl    $1, %edx
 cmovnel %eax, %edx
 movsbl  %cl, %eax
 idivb   %dl
 movsbl  %ah, %eax
 testb   %al, %al
 je      .LBB0_2
 callq   DCEMarker8
.LBB0_2:
 xorl    %eax, %eax
 popq    %rcx
 retq
```


CSmith
Compilers Under Test
Not Interesting
Candidate
Differential Testing
Missed Markers
Reduced Candidate
Instrument LibTooling
Instrumented Candidate
CReduce
Sanitizers CompCert

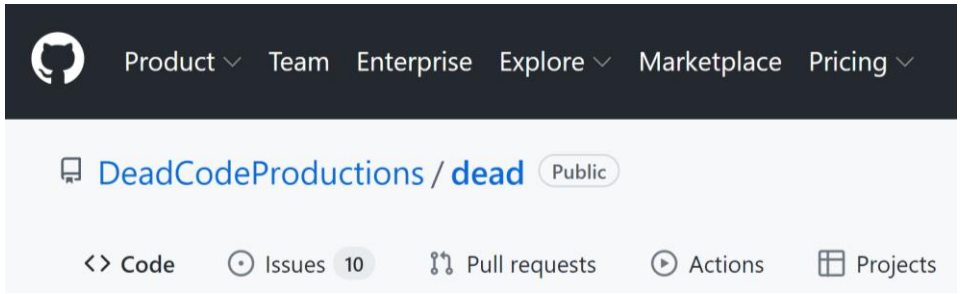

CSmith
Candidate
Instrument
LibTooling
Not Interesting
[InstCombine] allow Negator to fold multi-use select with constant arms
static short a = 1, b;
void DCEMarker8(void);
void DCEMarker14(void);
int main() {
a = -2;
DCEMarker14();
short c = -a;
b = c % a;
if (b)
DCEMarker8();
}
Candidate
If Regression
git bisect
Reduced Candidate
Sanitizers
CompCert


CSmith
Compilers Under Test
If Regression
git bisect
Not Interesting
Candidate
Differential Testing
Missed Markers
Reduced Candidate
Instrument LibTooling
Instrumented Candidate
CReduce
Sanitizers CompCert

DEAD: **D**ead Code **E**limination based **A**utomatic **D**ifferential Testing



DEAD: **D**ead Code **E**limination based **A**utomatic **D**ifferential Testing



	LLVM	GCC
Reported	47	55
Confirmed	35	46
Fixed	15	15

Studying Compilers with DCE

How good are compilers at DCE?

How good are compilers at DCE?

Corpus of 10,000 test programs:

- Generated with Csmith
- 3,109,167 dead blocks

How good are compilers at DCE?

Corpus of 10,000 test programs:

- Generated with Csmith
- 3,109,167 dead blocks

Optimization Level	% of dead blocks that are missed	
	GCC	LLVM
O0	85.2%	83.2%
O1	8.2%	5.2%
Os	6.0%	4.8%
O2	5.7%	4.4%
O3	5.6%	4.3%

Is DCE-based Differential Testing Feasible?

Is DCE-based Differential Testing Feasible?

Across Compilers

Eliminated blocks missed by the other compiler	
GCC	4,749
LLVM	39,723

Is DCE-based Differential Testing Feasible?

Across Compilers

Eliminated blocks missed by the other compiler	
GCC	4,749
LLVM	39,723

Across Optimization Levels

Missed dead blocks at –O3 but eliminated at –O1 / –O2	
GCC	308
LLVM	456

LLVM’s Evolution

Version	4	5	6	7	8	9	10	11	12	13	Trunk

LLVM’s Evolution

Version	4	5	6	7	8	9	10	11	12	13	Trunk
Eliminated Blocks	67%	68%	80%	82%	83%	85%	85%	89%	92%	94%	96%

LLVM's Evolution

Version	4	5	6	7	8	9	10	11	12	13	Trunk
Eliminated Blocks	67%	68%	80%	82%	83%	85%	85%	89%	92%	94%	96%
Regressions	0%	1%	4%	4%	4%	3%	4%	4%	4%	4%	4%

LLVM's Evolution

Version	4	5	6	7	8	9	10	11	12	13	Trunk
Eliminated Blocks	67%	68%	80%	82%	83%	85%	85%	89%	92%	94%	96%
Regressions	0%	1%	4%	4%	4%	3%	4%	4%	4%	4%	4%

1.4% of regressions
13 → Trunk

Examples

```
static int a[2], b, *c[2];

int main() {
  for (b = 0; b < 2; b++) {
    c[b] = &a[1];
  }
  if (!c[0]){
    DCEMarker();
  }
  return 0;
}
```

c[0] points to a non-zero address

-O1

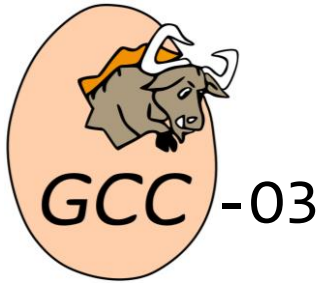
-O3

```
static int a[2], b, *c[2];

int main() {
  for (b = 0; b < 2; b++) {
    c[b] = &a[1];
  }
  if (!c[0]){
    DCEMarker();
  }
  return 0;
}
```

Vectorized at –O3

c[0] points to a non-zero address

-O1



-O3

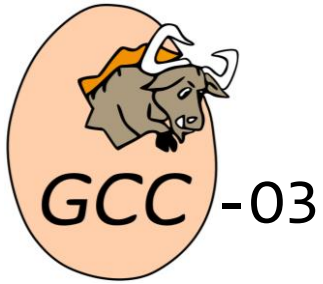
```
static int a[2], b, *c[2];

int main() {
  for (b = 0; b < 2; b++) {
    c[b] = &a[1];
  }
  if (!c[0]){
    DCEMarker();
  }
  return 0;
}
```

Pointer data vectorized as unsigned int

Vectorized at –O3

c[0] points to a non-zero address

-O1



-O3

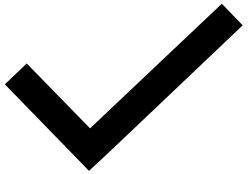
```
static long a = 78240;
static int b, d;
static short e;
static short c(short f, short h) {
  return h == 0 ||
   (f && h == 1) ? 0 : f % h; }
int main() {
  short g = a;
  for (b = 0; b < 1; b++) {
    e = a;
    d = c((e == a) ^ g, a);
  }

  if (d) {
    DCEMarker();
    for (; a; a++);
  }
}
```

LLVM 13 -O1

LLVM 13 -O3

```
static long a = 78240;
static int b, d;
static short e;
static short c(short f, short h) {
  return h == 0 ||
   (f && h == 1) ? 0 : f % h; }
int main() {
  short g = a;
  for (b = 0; b < 1; b++) {
    e = a;
    d = c((e == a) ^ g, a);
  }

  if (d) {
    DCEMarker();
    for (; a; a++);
  }
}
```

LLVM 13 -O1



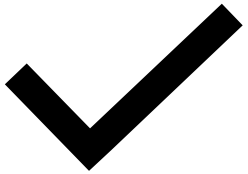
Modulo on constant ranges: [X,X+1) % [X,X+1) not simplified

LLVM 13 -O3

```
static int b = -1, e = 1;
static short c = 0, d = 0;
short a(unsigned short f, int g) {
  return f >> g;
}

int main() {
  c++;
  d  = a(4294967295 + (c > 0),1);
  e ^= (short)(d * 3) /(unsigned)b;
  if (!e)
    DCEMarker();
}
```

e != 0

LLVM 13
-O3



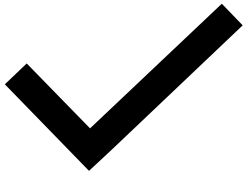
LLVM dev
-O3

```
static int b = -1, e = 1;
static short c = 0, d = 0;
short a(unsigned short f, int g) {
  return f >> g;
}

int main() {
  c++;
  d  = a(4294967295 + (c > 0),1);
  e ^= (short)(d * 3) /(unsigned)b;
  if (!e)
    DCEMarker();
}
```

Regression on shift peephole optimization

e != 0

LLVM 13
-O3



LLVM dev
-O3

```
static char a = 1, b = 0, c = 1;

int main() {
  for (b = 6; b < 1; b = 0)
    c = 0;
  if (1 != 0 ^ 0 < a)
    DCEMarker();
  a = c;
  return 0;
}
```

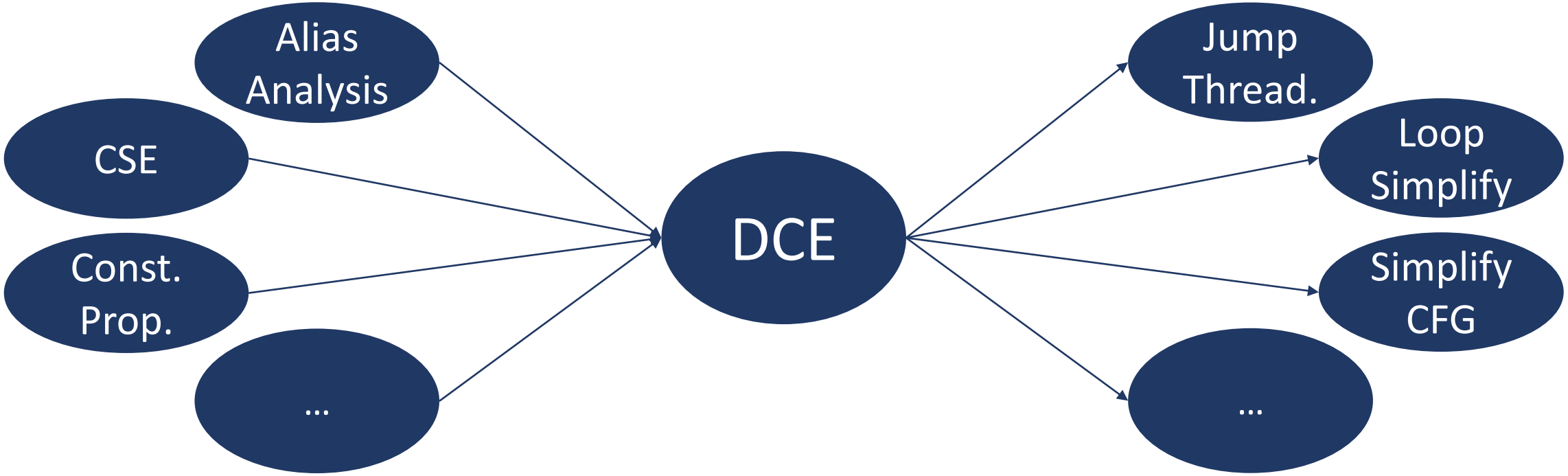
LLVM 4
-O3

LLVM main
-O3

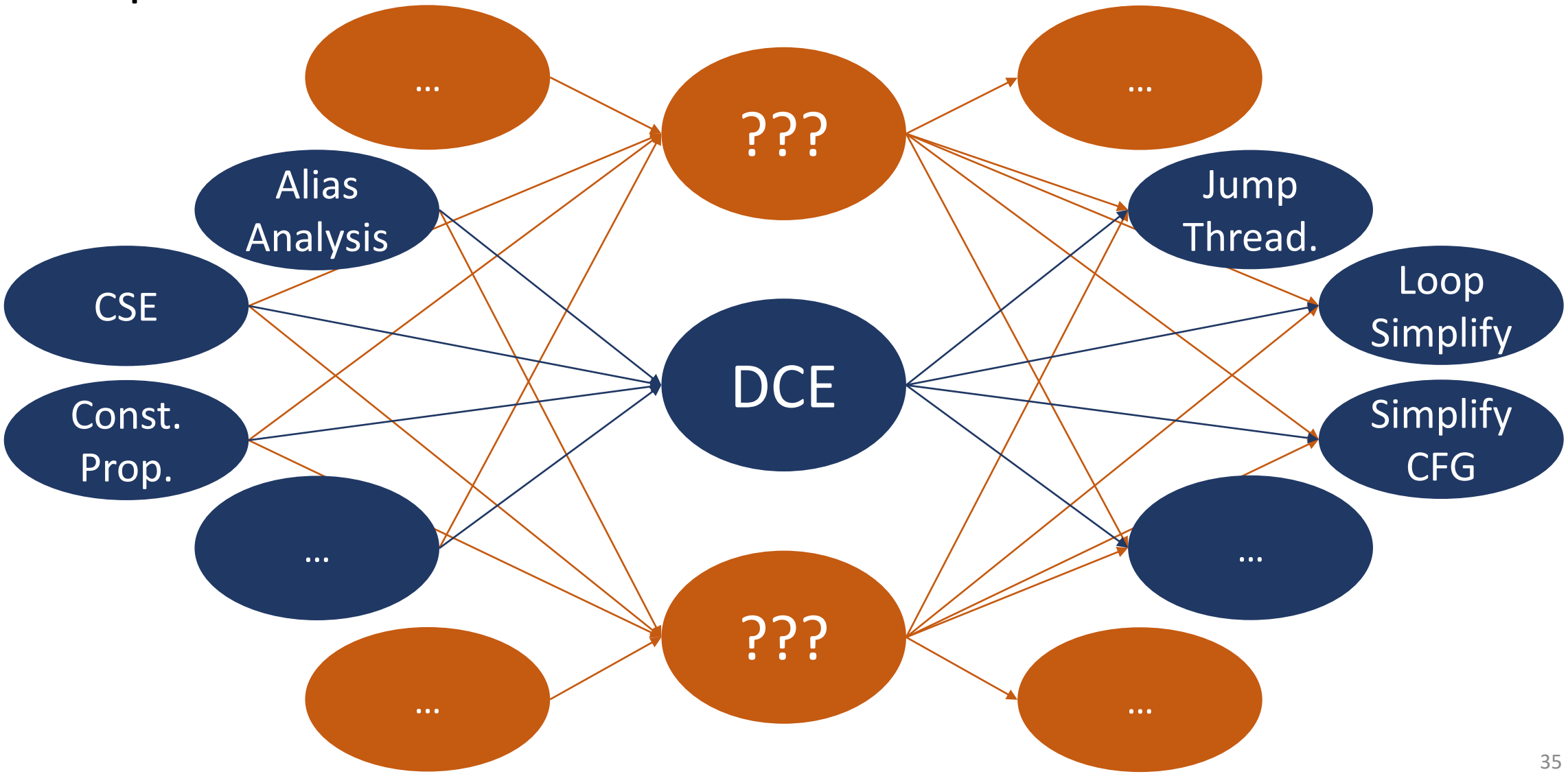
[SimplifyCFG] don't sink common insts too soon (PR34603)

This should solve:
https://bugs.llvm.org/show_bug.cgi?id=34603
...by preventing SimplifyCFG from altering redundant instructions before early-cse has a chance to run.

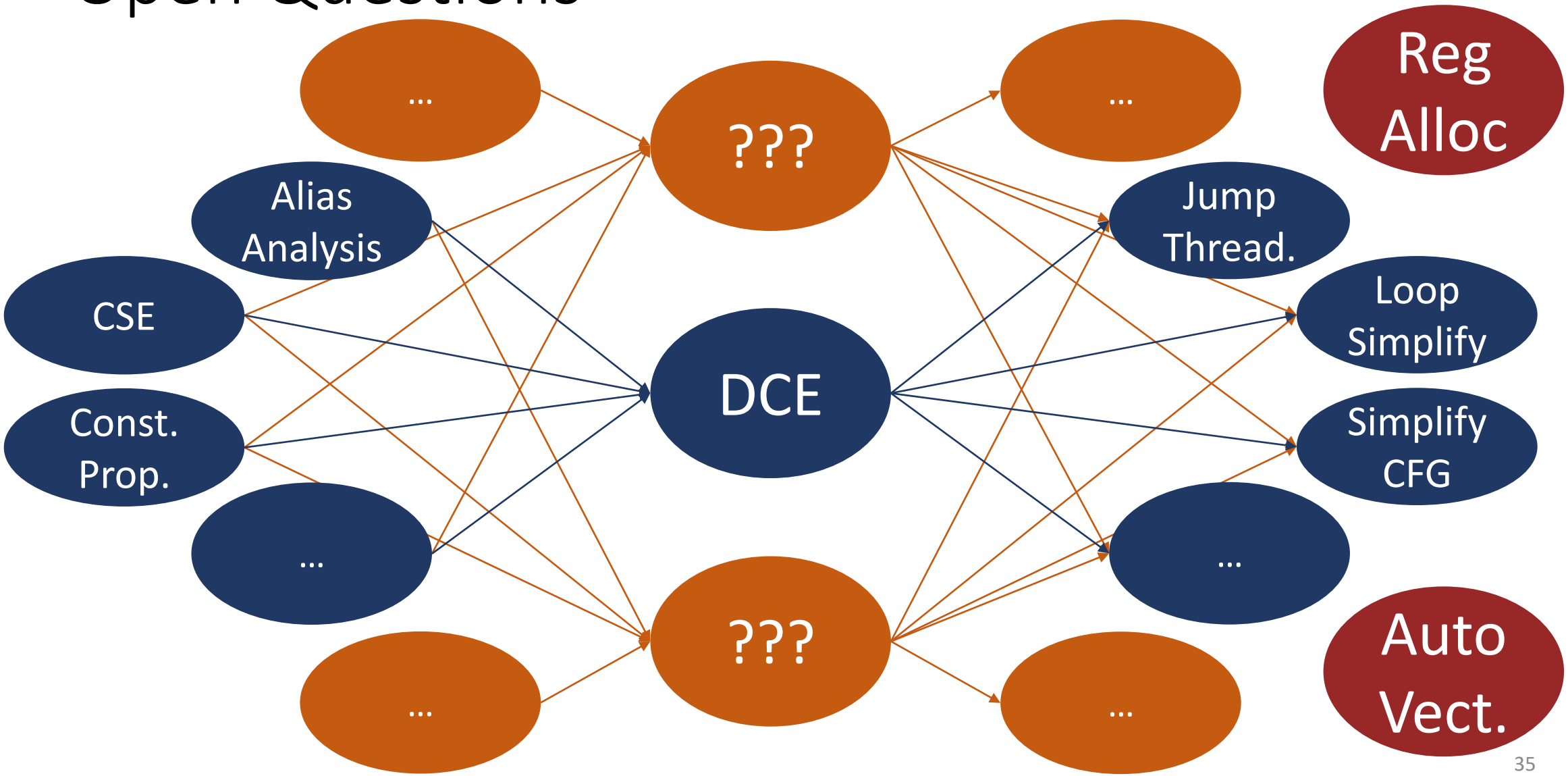
Open Questions

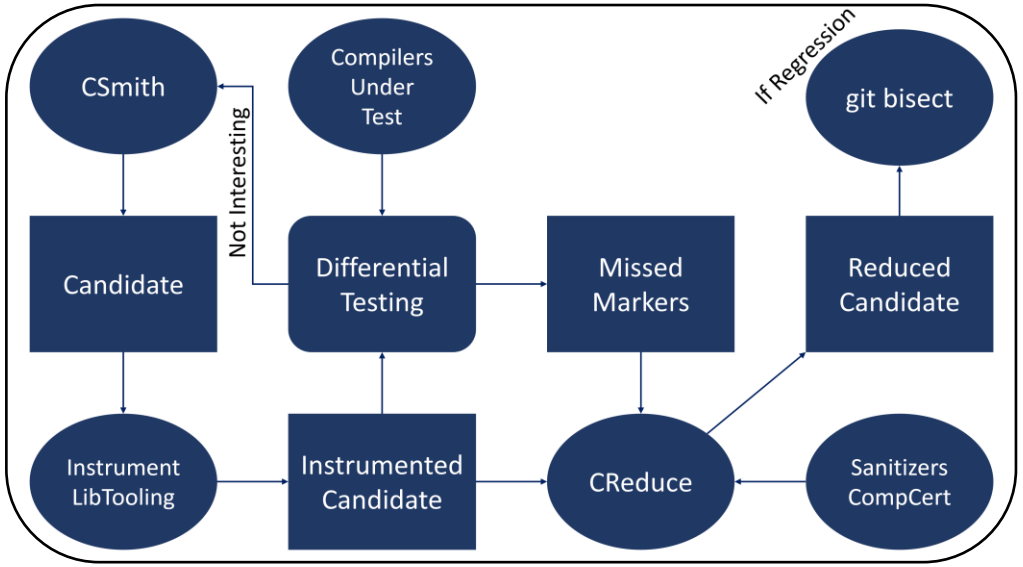


Open Questions

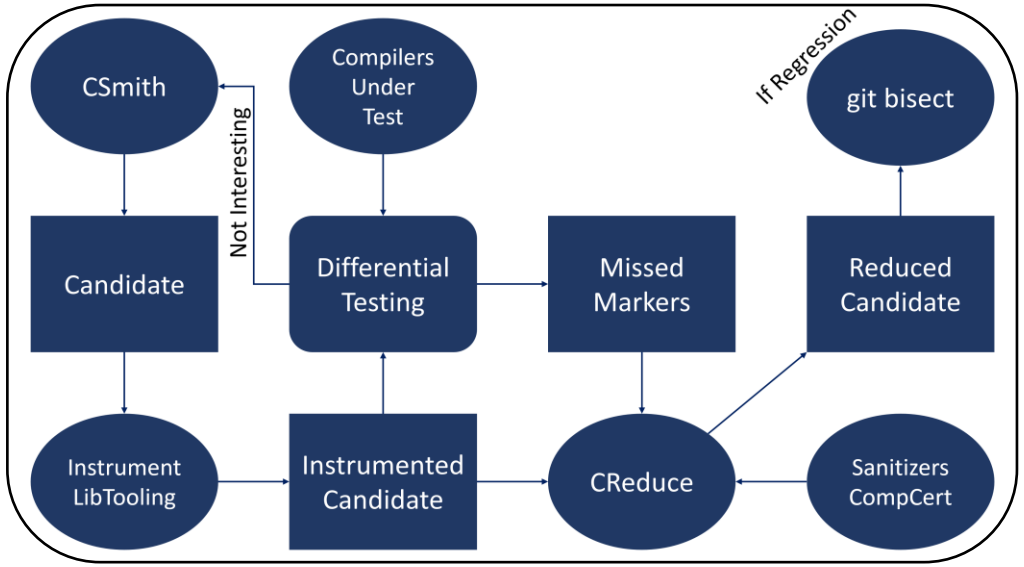


Open Questions

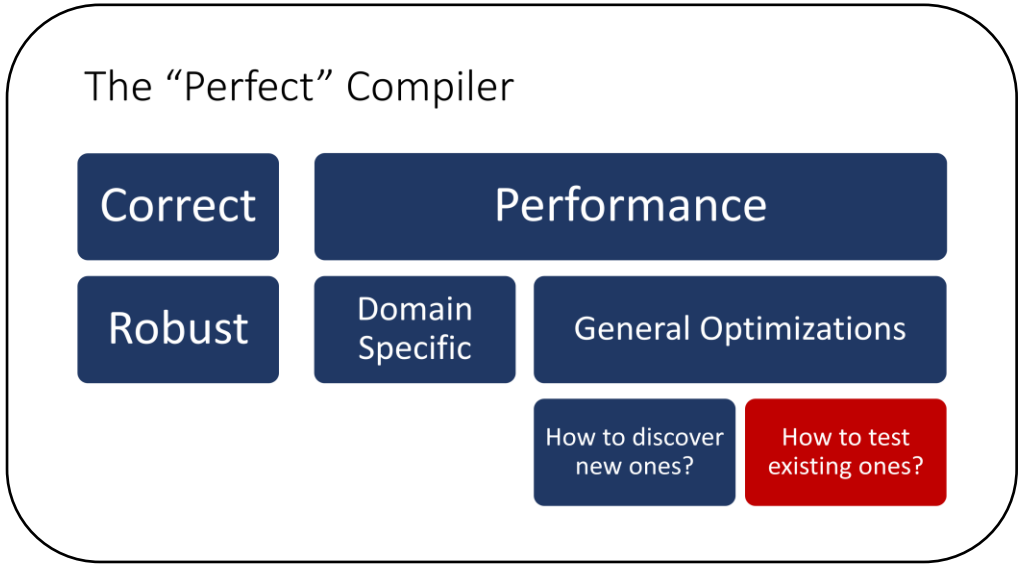


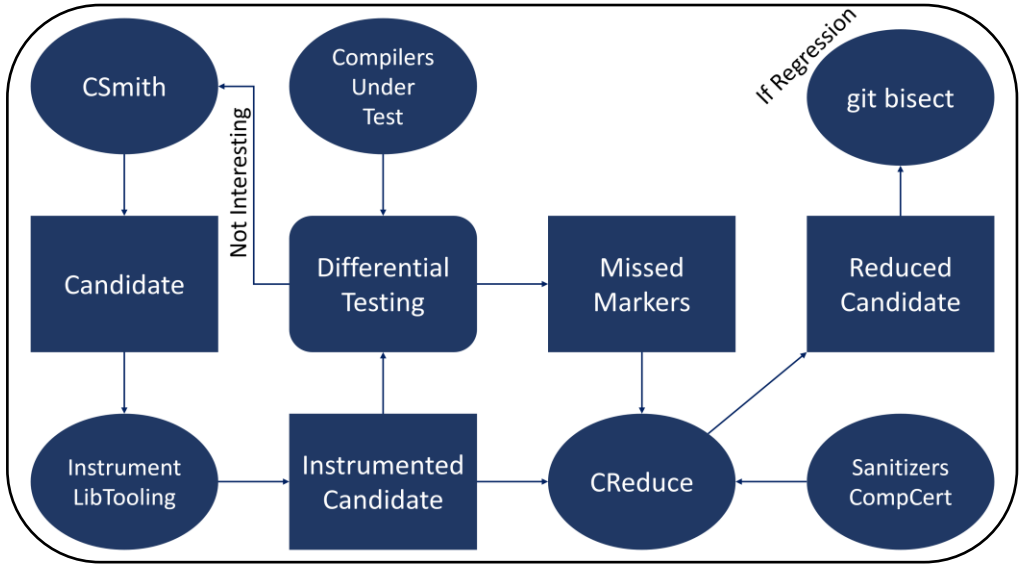


DeadCodeProductions/dead



DeadCodeProductions/dead





DeadCodeProductions/dead

The "Perfect" Compiler

- Correct
- Performance
- Robust
  - Domain Specific
  - General Optimizations
    - How to discover new ones?
    - How to test existing ones?

Open Questions