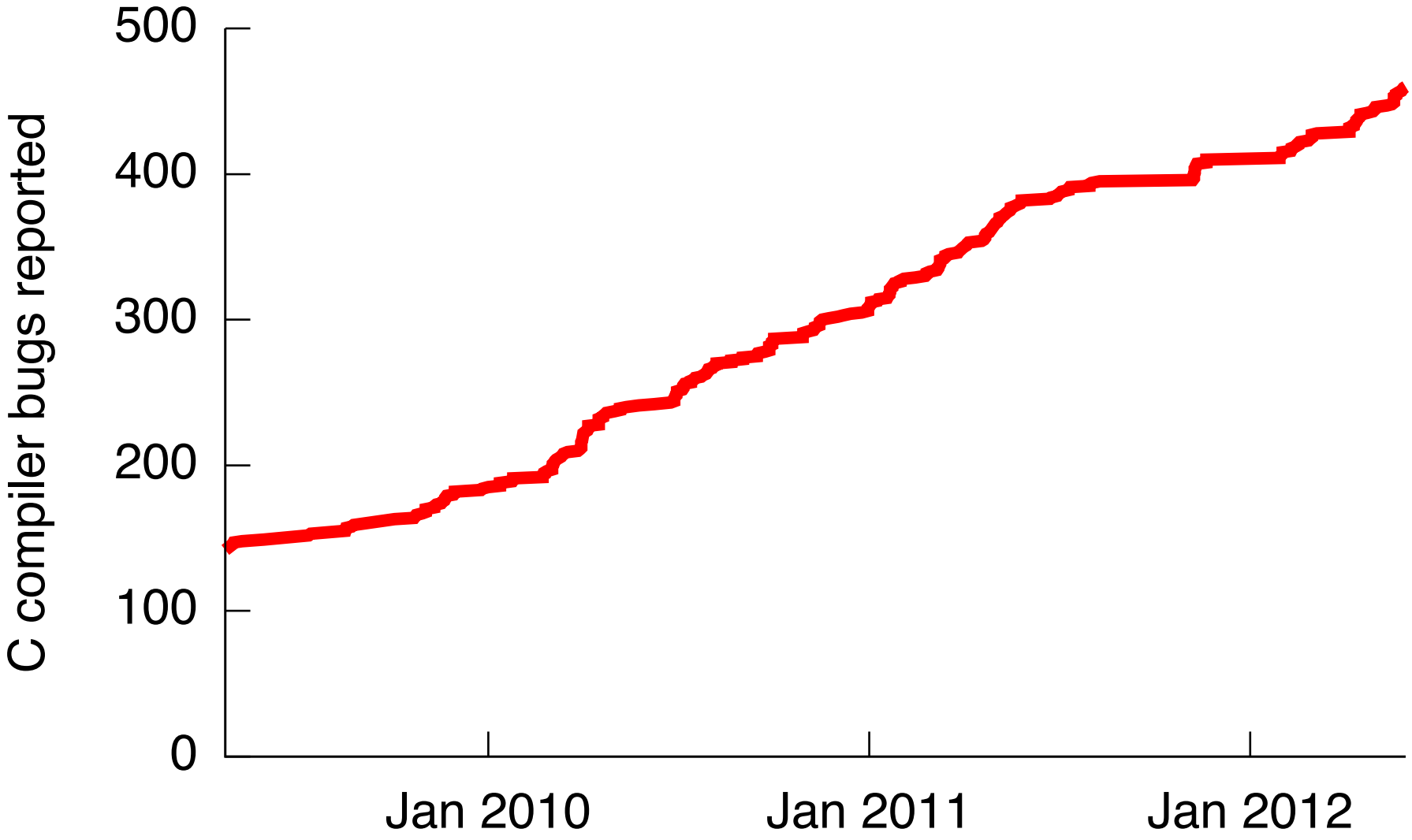
Test-Case Reduction for C Compiler Bugs

John Regehr, Yang Chen, Pascal Cuoq,
Eric Eide, Chucky Ellison, Xuejun Yang

Background: Csmith [PLDI 2011]



- **Csmith's bug-finding power is maximized when programs are ~80 KB**
  - **But 80 KB test cases make bad bug reports**

- ***Automated test case reduction* is needed**

```
 template< class A0 , class A1> __attribute__((always_inline)) typename boost::dispatch::meta::call<tag::shift_right_( A0 const& , A1 const& )>::type shift_right ( A0
const& a0 , A1 const& a1 ) { typename boost::dispatch::make_functor<tag::shift_right_, A0>::type callee; return callee( a0 , a1);; }
 template< class A0 , class A1> __attribute__((always_inline)) typename boost::dispatch::meta::call<tag::shift_right_( A0 const& , A1 const& )>::type shr
( A0 const& a0 , A1 const& a1 ) { typename boost::dispatch::make_functor<tag::shift_right_, A0>::type callee; return callee( a0 , a1);; }
} }
# 5 "/home/gaunard/build/may_alias/include/boost/simd/toolbox/operator/include/functions/shift_right.hpp" 2
# 1 "/home/gaunard/dev/may_alias/modules/boost/simd/operator/include/boost/simd/toolbox/operator/functions/scalar/shift_right.hpp" 1
# 14 "/h                                                                                                                    lbox/operator/functions/scalar/shift_right.hpp"
namespa
{
 } } } nam                                                                                                  __attribute__((always_inline)) boost :: simd :: ext ::
impleme                                                                                                     tag::cpu_ > dispatching( boost::simd::tag::shift_right_,
tag::cpu_                                                                                                   per() ) { boost :: simd :: ext :: implement<
boost::si                                                                                                   that; return that; } } } } namespace boost { namespace simd
{ namesp                                                                                                                                scalar_< floating_<A0> > , scalar_<
integer_
 {
  typede
  inline
  {
   typed
   retur
  }
 };
} } }
namespace boost { namespace si
 } } } namespace boost { namespa                                                                                                                                                simd :: ext ::
implement< boost::simd::tag::shi                                                                                                                                                ::shift_right_,
tag::cpu_ , scalar_< integer_<A0>
boost::simd::tag::shift_right_( sca                                                                                                                                             { namespace simd
{ namespace ext { template< class A0 , class A1 , class D                                                                                                                       > , scalar_<
integer_<A1> > ) , tag::cpu_, Dummy >
 {
  typedef A0 result_type;
  inline result_type operator()( A0 const& a0 , A1 const& a1 ) const { return a0 >> a1; }
 };
} } }
# 6 "/home/gaunard/build/may_alias/include/boost/simd/toolbox/operator/include/functions/shift_right.hpp" 2
# 1 "/home/gaunard/dev/may_alias/modules/boost/simd/operator/include/boost/simd/toolbox/operator/functions/simd/common/shift_right.hpp" 1
# 20 "/home/gaunard/dev/may_alias/modules/boost/simd/operator/include/boost/simd/toolbox/operator/functions/simd/common/shift_right.hpp"
namespace boost { namespace simd { namespace ext
{
```

From GCC PR 50800:
"Testcase is [here]
(couldn
bugzilla

Next comment:
"That you couldn't
attach
somet

Next comment:
203 KB reduced test
case attached

- **Our goal: "Beautiful" test cases for compiler bugs**
- **A beautiful test case is:**
  - **Small**
  - **Obviously well-defined**

```
int printf (const char *, …);

char f[] = { -9L };

int main (void) {
  printf ("%d\n", 255 | f[0]);
}
```

Intel CC 12.0.5 for x86-64 is wrong at "-fast -ipo"

```
int printf (const char *, ...);

const union {
  short f1;
  int f2 : 13;
} a = { 30155 };

int main (void) {
  printf ("%d\n", a.f1);
  printf ("%d\n", a.f2);
  return 0;
}
```

GCC 4.4.3 from Ubuntu 10.04 LTS for x86-64 is wrong at -O1

- **These test cases were produced automatically by our tool**
  - **They are (I claim) pretty close to minimal**
  - **Previous tools can't produce them**

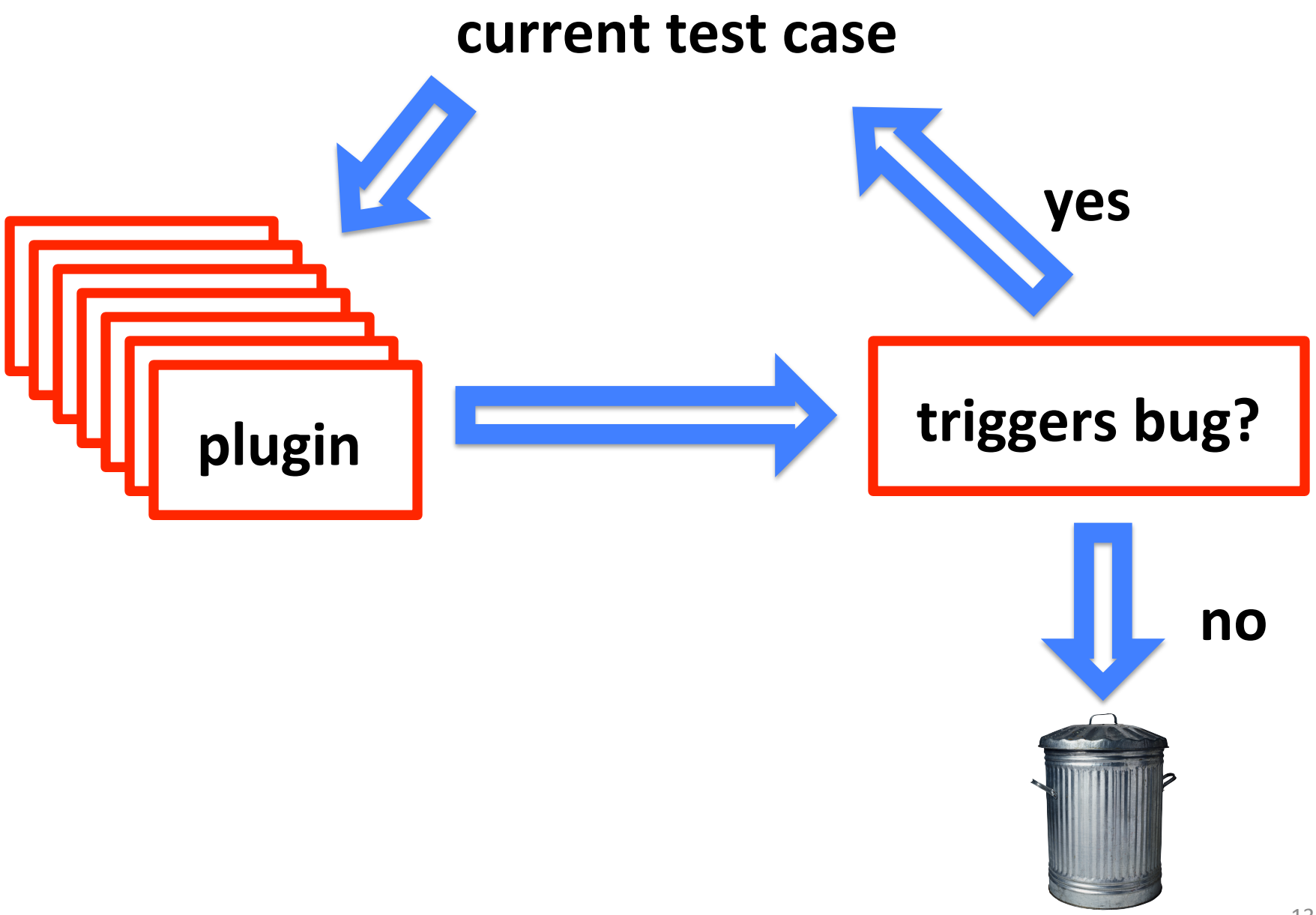
- **Prior art: Delta Debugging**
  - **Greedy search for smaller test cases**
  - **Deletes contiguous chunks of the input**
- **"Delta" tool from UC Berkeley**
  - **Implements Delta Debugging**
  - **Operates at line granularity**
  - **Commonly used by compiler developers**

- **Delta has problems reducing C/C++**
  - **Delta makes localized changes**
  - **But escaping local minima requires coordinated changes**
  - **Consequently, Delta gets stuck at (large) local minima**

- Our goal: "Beautiful" test cases for compiler bugs
- A beautiful test case is:
  - Small
  - Obviously well-defined
- We created 3 new reducers
  - I'll talk about one of them: C-Reduce

C-Reduce:

- Based on "generalized Delta Debugging"
- Transformations implemented by plugins
- Terminates when fixpoint is found


current test case
yes
plugin
triggers bug?
no

```
typedef volatile int vint;
vint **depth;
int *b;
vint **get_depth (void) {
  return depth;
}
int fn1 (int inc) {
  int tmp = 0;
  if (get_depth() == &b)
    tmp = inc + **depth;
  return tmp;
}
```

GCC 4.3.0 for x86-64 crashes at -O3

```
typedef volatile int vint;
vint **depth;
int *b;
vint **get_depth (void) {
  return depth;
}
int fn1 (int inc) {
  int tmp = 0;
  if (get_depth() == &b)
    tmp = inc + **depth;
  return tmp;
}
```

```
typedef volatile int vint;
vint **depth;
int *b;

int fn1 (int inc) {
  int tmp = 0;
  if (depth          == &b)
    tmp = inc + **depth;
  return tmp;
}
```

```
typedef volatile int vint;
vint **depth;
int *b;



int fn1 (int inc) {
  int tmp = 0;
  if (depth        == &b)
    tmp = inc + **depth;
  return tmp;
}
```

```
typedef volatile int vint;
vint **depth;
int *b;
```

```
int fn1 (int inc) {
   int tmp = 0;
   if (depth        == &b)
      tmp = inc + **depth;
   return tmp;
}
```

```
typedef volatile int vint;
vint **depth;
int *b;
```

```
void fn1 (int inc) {
   int tmp = 0;
   if (depth        == &b)
      tmp = inc + **depth;

}
```

```
typedef volatile int vint;
vint **depth;
int *b;



void fn1 (int inc) {
  int tmp = 0;
  if (depth        == &b)
    tmp = inc + **depth;

}
```

```
typedef volatile int vint;
vint **depth;
int *b;



void fn1 (int inc) {
   int tmp = 0;
   if (depth        == &b)
      tmp = inc + **depth;

}
```

```
typedef volatile int vint;
vint **depth;
int *b;


int inc;
void fn1 (          ) {
   int tmp = 0;
   if (depth          == &b)
      tmp = inc + **depth;

}
```

```
typedef volatile int vint;
vint **depth;
int *b;


int inc;
void fn1 (        ) {
  int tmp = 0;
  if (depth        == &b)
    tmp = inc + **depth;

}
```

```
typedef volatile int vint;
vint **depth;
int *b;


int inc;
void fn1 (        ) {
  int tmp = 0;
  if (depth        == &b)
    tmp = inc + **depth;

}
```

```
volatile int **depth;
int *b;


int inc;
void fn1 (        ) {
  int tmp = 0;
  if (depth        == &b)
    tmp = inc + **depth;

}
```

```
volatile int **depth;
int *b;


int inc;
void fn1 (        ) {
  int tmp = 0;
  if (depth        == &b)
    tmp = inc + **depth;

}
```

```
volatile int **depth;
int *b;


int inc;
void fn1 (        ) {
  int tmp = 0;
  if (depth        == &b)
    tmp = inc + **depth;

}
```

```
volatile int **depth;
int *b;


int inc;
void fn1 (        ) {
  int tmp = 0;
  if (depth        == &b)
    [              ]**depth;

}
```

```
volatile int **depth;
int *b;


int inc;
void fn1 (        ) {
  int tmp = 0;
  if (depth        == &b)
                 **depth;

}
```

```
volatile int **depth;
int *b;


int inc;
void fn1 (          ) {
   int tmp = 0;
   if (depth          == &b)
                    **depth;

}
```

```
volatile int **a      ;
int *b;


int inc;
void fn1 (         ) {
   int tmp = 0;
   if (a          == &b)
                 **a      ;

}
```

```
volatile int **a     ;
int *b;


int inc;
void fn1 (        ) {
  int tmp = 0;
  if (a             == &b)
                **a     ;

}
```

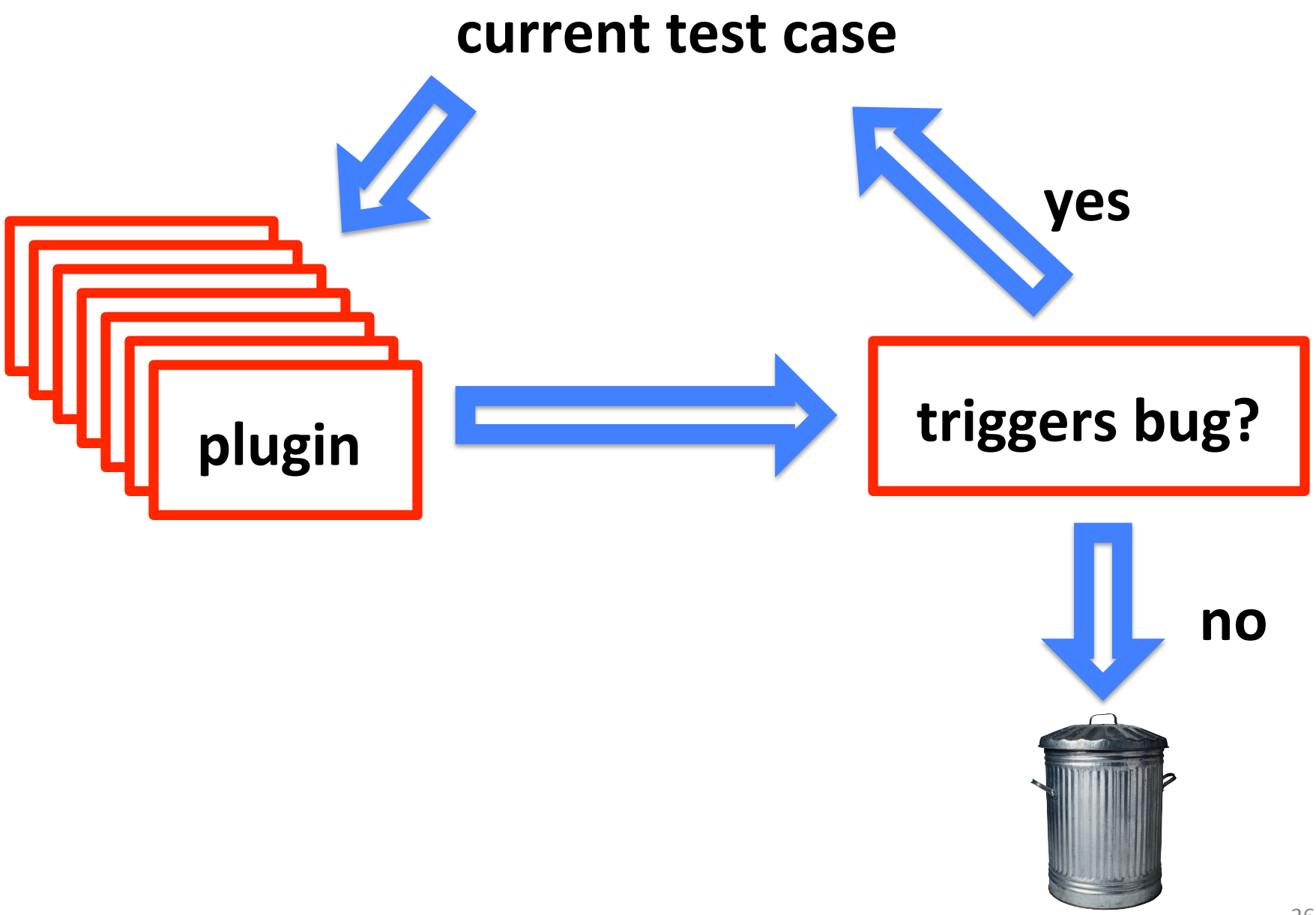
```
volatile int **a;
int *b;
void fn1() {
  if (a == &b) **a;
}
```

GCC 4.3.0 for x86-64 crashes at -O3

65 plugins including…

- C-specific peephole passes:
  - 0xfeedbeefULL → 1
  - x ^= y → x = y
  - (x + 1) → x + 1
  - while (…) → if (…)
  - x ? y : z → y
- Remove chunks of text, like Delta
- Some non-local transformations

- **41 C/C++-specific plugins including:**
  - **Inline a function call**
  - **Scalar replacement of aggregates**
  - **Un-nest nested function calls**
  - **Remove dead arguments**
  - **Make function return void**
  - **Reduce array dimension or pointer level**
  - **Shorten identifier name**
- **Built using Clang (LLVM C frontend)**


current test case
yes
plugin
triggers bug?
no

- **Our goal: "Beautiful" test cases for compiler bugs**
- **A beautiful test case is:**
  - **Small**
  - **Obviously well-defined**

GCC PR 51962

```
#include <iostream>
using namespace std;
int r[3], x[3], y[3];
int main() {
  int xa=2,ya=5,xb=4,yb=2,n=3;
  x[0] = 3; x[1] = 5; x[2] = 1;
  y[0] = 1; y[1] = 3; y[2] = 3;
  r[0] = 2; r[1] = 1; r[2] = 2;
  int tcount = 0;
  for (int k=min(xa,xb); k<=max(xa,xb); k++) {
    bool found1,found2 = false;
    for (int j=0; j<n; j++) {
      if (((k-x[j])*(k-x[j])+(y[j]-ya)*(y[j]-ya))<=r[j]*r[j]) { found1 = true; }
      if (((k-x[j])*(k-x[j])+(y[j]-yb)*(y[j]-yb))<=r[j]*r[j]) { found2 = true; }
      if (found1 && found2) break;
    }
    if (!found1) tcount++; if (!found2) tcount++;
  }
  cout << tcount << endl; return 0;  }
```

GCC PR 51962

Bug report says:

"Compile the following simple code without -O3, and run.

Now compile it with -O3 option (for optimization), run again.

Surprisingly 2 different outputs appear."

```
#include <iostream>
using namespace std;
```

```
nd1 = true; }
nd2 = true; }
```

```
  }
  if (!found1) tcount++; if (!found2) tcount++;
 }
 cout << tcount << endl; return 0;  }
```

```
#include <iostream>
using namespace std;
```

GCC PR 51962

Bug report says:

"Compile the following simple code without -O3

Now compi optimizatio

Surprisingly

```
  }
  if (!found1) tco
}
cout << tcount <
```

GCC developer responds:

"You do not initialise found1."

PR 51962 is RESOLVED INVALID

And this person may have a hard time getting someone to read his next bug report

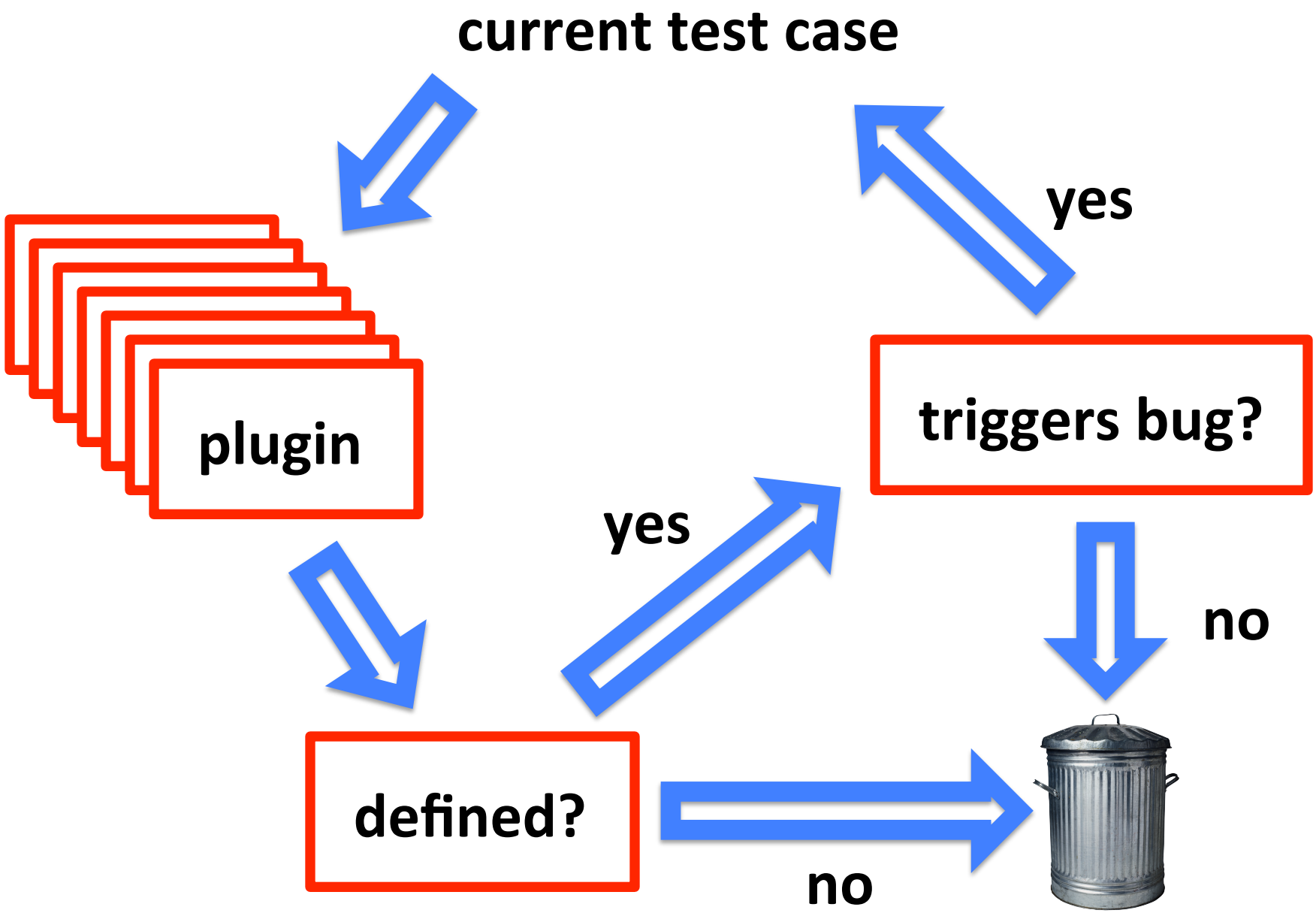
GCC PR 51962

```
#include <iostream>
using namespace std;
int r[3], x[3], y[3];
int main() {
  int xa=2,ya=5,xb=4,yb=2,n=3;
  x[0] = 3; x[1] = 5; x[2] = 1;
  y[0] = 1; y[1] = 3; y[2] = 3;
  r[0] = 2; r[1] = 1; r[2] = 2;
  int tcount = 0;
  for (int k=min(xa,xb); k<=max(xa,xb); k++) {
    bool found1,found2 = false;
    for (int j=0; j<n; j++) {
      if (((k-x[j])*(k-x[j])+(y[j]-ya)*(y[j]-ya))<=r[j]*r[j]) { found1 = true; }
      if (((k-x[j])*(k-x[j])+(y[j]-yb)*(y[j]-yb))<=r[j]*r[j]) { found2 = true; }
      if (found1 && found2) break;
    }
    if (!found1) tcount++; if (!found2) tcount++;
  }
  cout << tcount << endl; return 0;  }
```

- C99 has
  - 191 kinds of undefined behavior
  - 52 kinds of unspecified behavior
- Code in a bug report must not execute these behaviors
  - Though sometimes this rule may be relaxed for compiler crash bugs
- Test-case reducers tend to introduce these behaviors

Solutions:

1. **Teach the reducer to avoid undefined behavior**
   - **Two of our reducers do this by reusing Csmith's logic**
   - **But they can only reduce Csmith output**
2. **Call an external validity checker**
   - **C-Reduce does this**


current test case
plugin
defined?
yes
triggers bug?
yes
no
no

- **Dynamic validity checkers for C**
  - **KCC: executable semantics for C99**
  - **Frama-C: static analyzer that supports an interpreter mode**

- **Result: C-Reduce's output is free from undefined / unspecified behaviors**

Median size output from reducers:

- **57 compiler-crash bugs**
  - Delta: 8,600 bytes
  - C-Reduce: 120 bytes
- **41 wrong-code bugs**
  - Delta: 6,500 bytes
  - C-Reduce: 200 bytes
- **Median reduction times are <10 minutes**

- **What about reducing other languages?**
- **C++ is pretty easy**
  - **We recently added 10 transformations**
    - **Collapse namespace, collapse class hierarchy, …**
  - **Problem: No validity checker for C++**
- **Should be pretty easy to support additional languages**

C-Reduce is…

- Almost as good as the best human test case reducers
- Extensible via plugins
- Open source:

  http://embed.cs.utah.edu/creduce/
- Being used