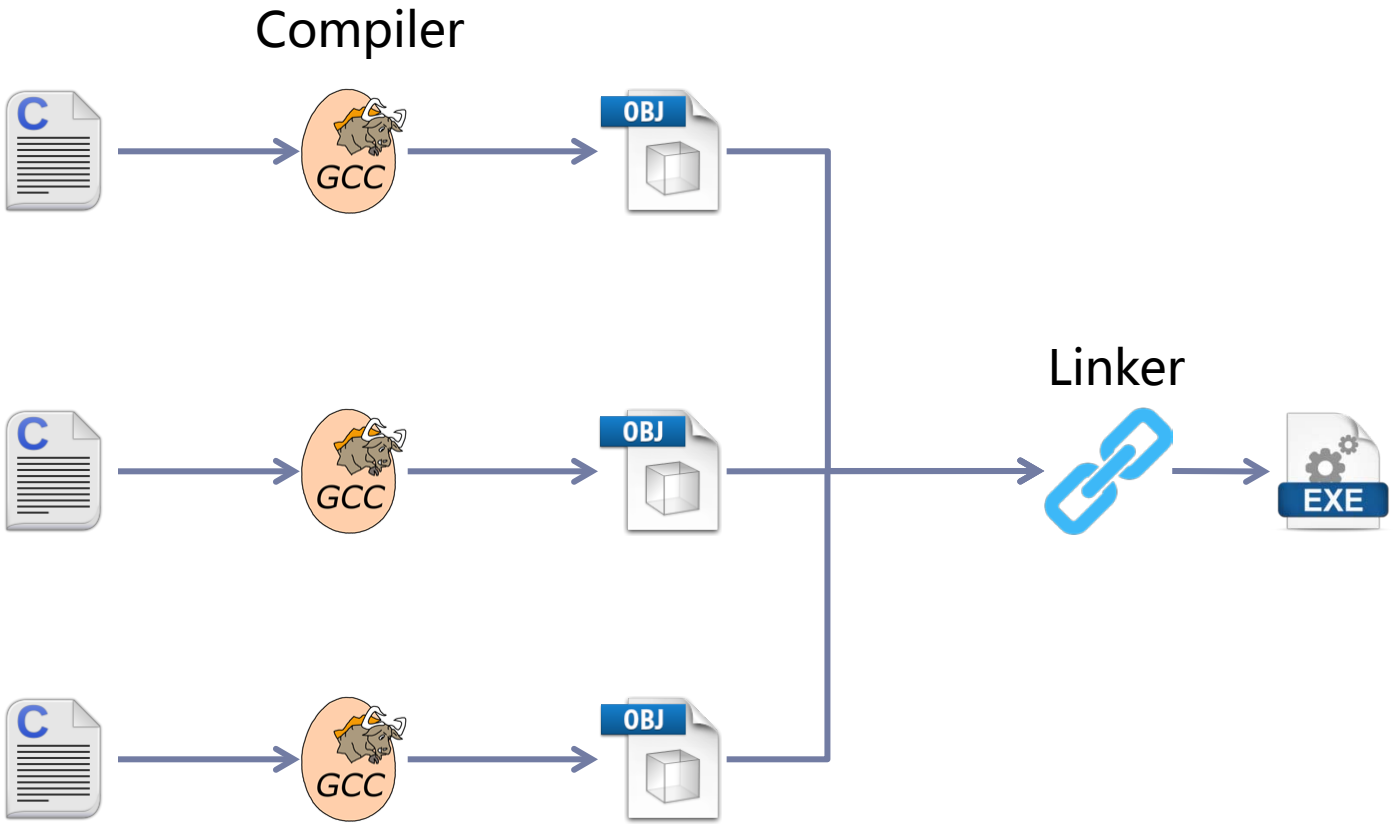
Randomized Stress-Testing of Link-Time Optimizers

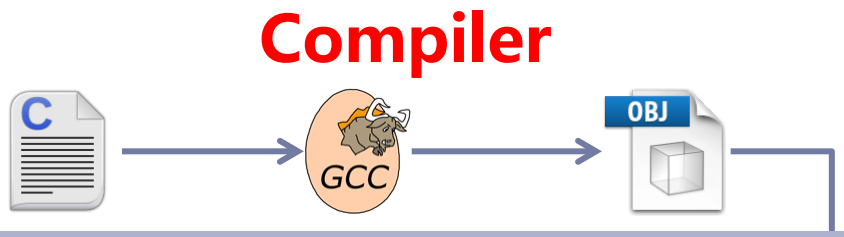
Vu Le, **Chengnian Sun**, Zhendong Su

University of California, Davis

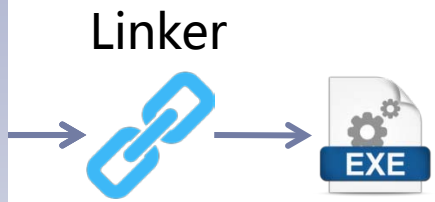
General Software Build Process



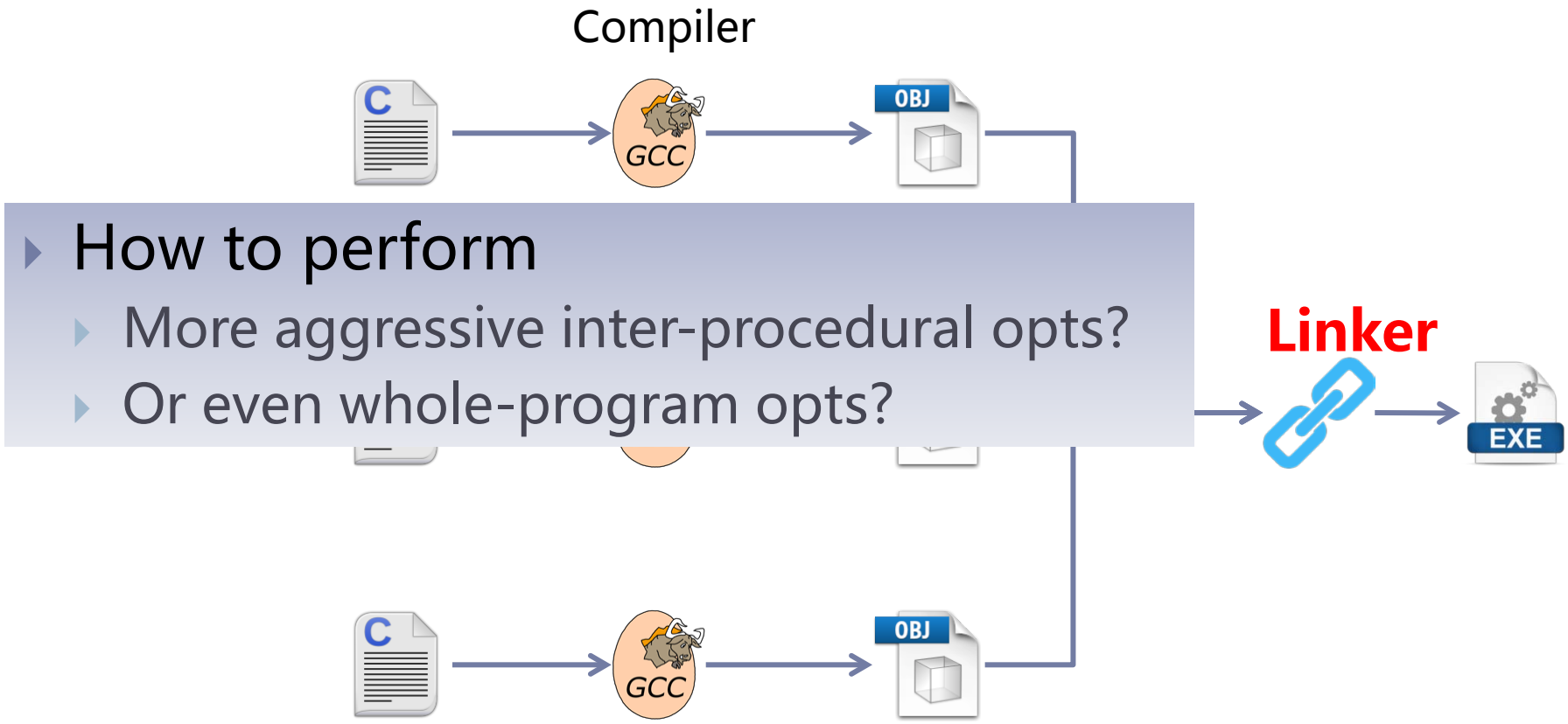
General Software Build Process



- Compiler Optimizations
  - Intra-procedural, within a function
  - Inter-procedural, across functions
  - Whole-program, over all the functions
- Optimizing a translation unit (*.c),
  - Intra-procedural
  - Inter-procedural? Limited to the unit
  - Whole-program? Usually **NO**.

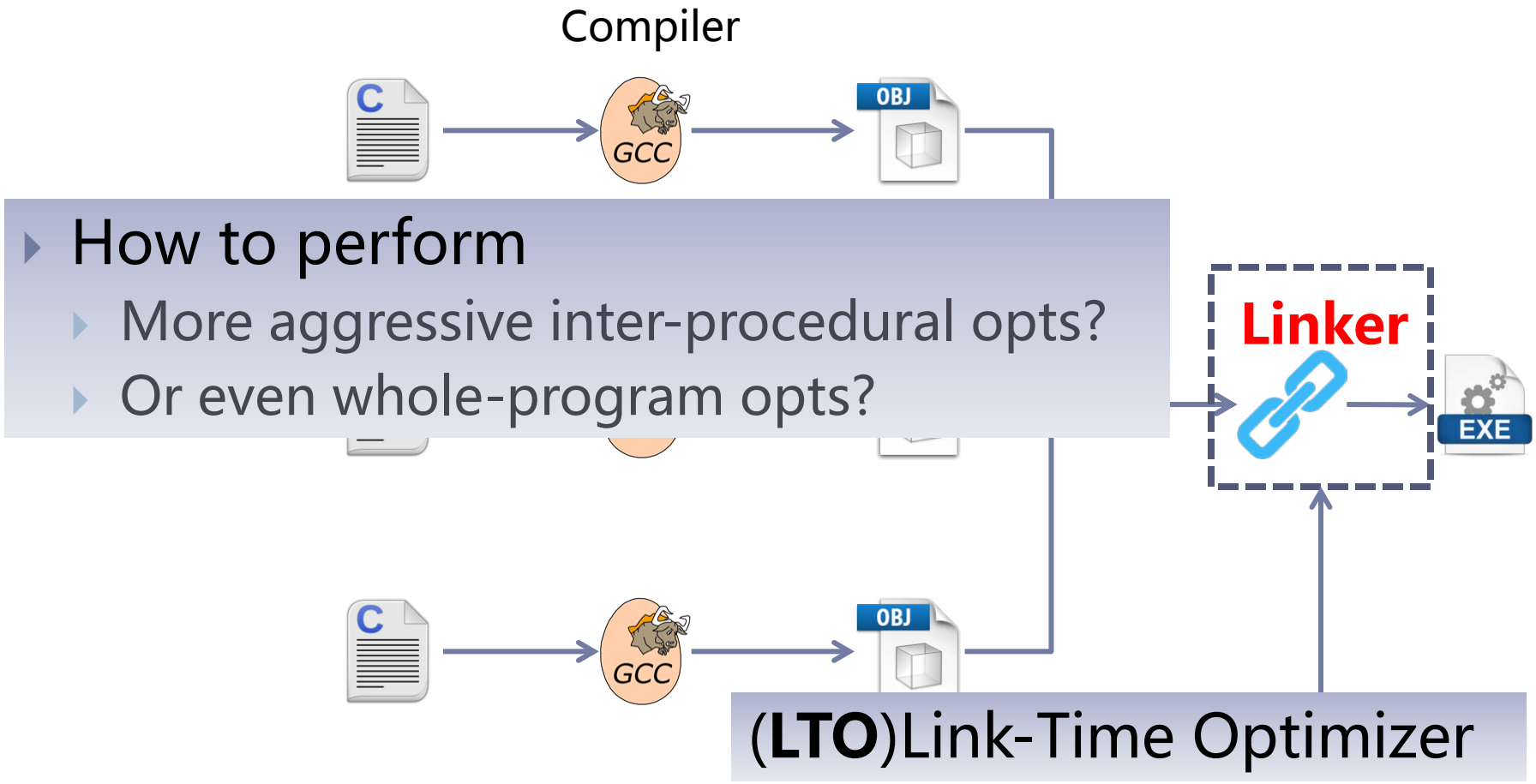


General Software Build Process



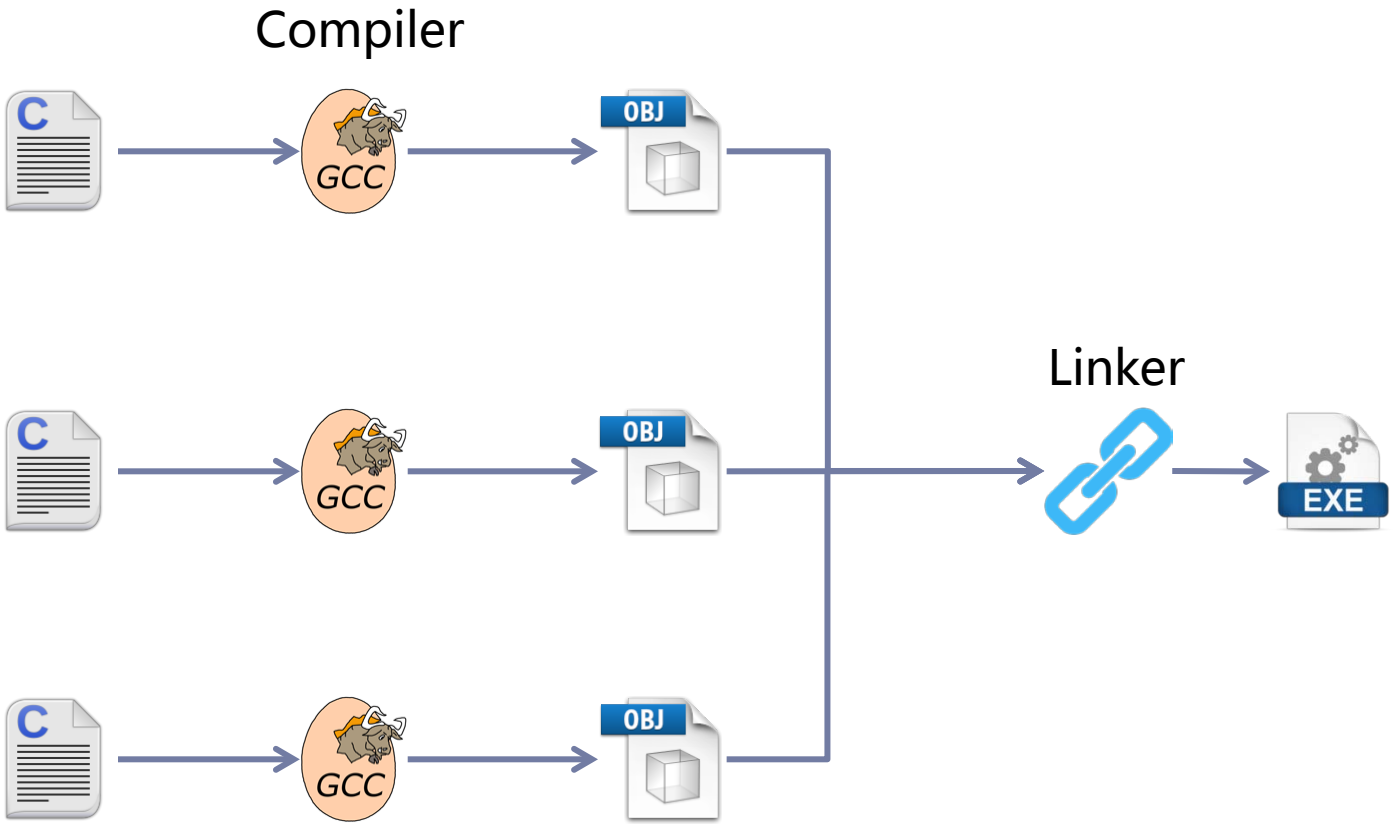
- How to perform
  - More aggressive inter-procedural opts?
  - Or even whole-program opts?

General Software Build Process

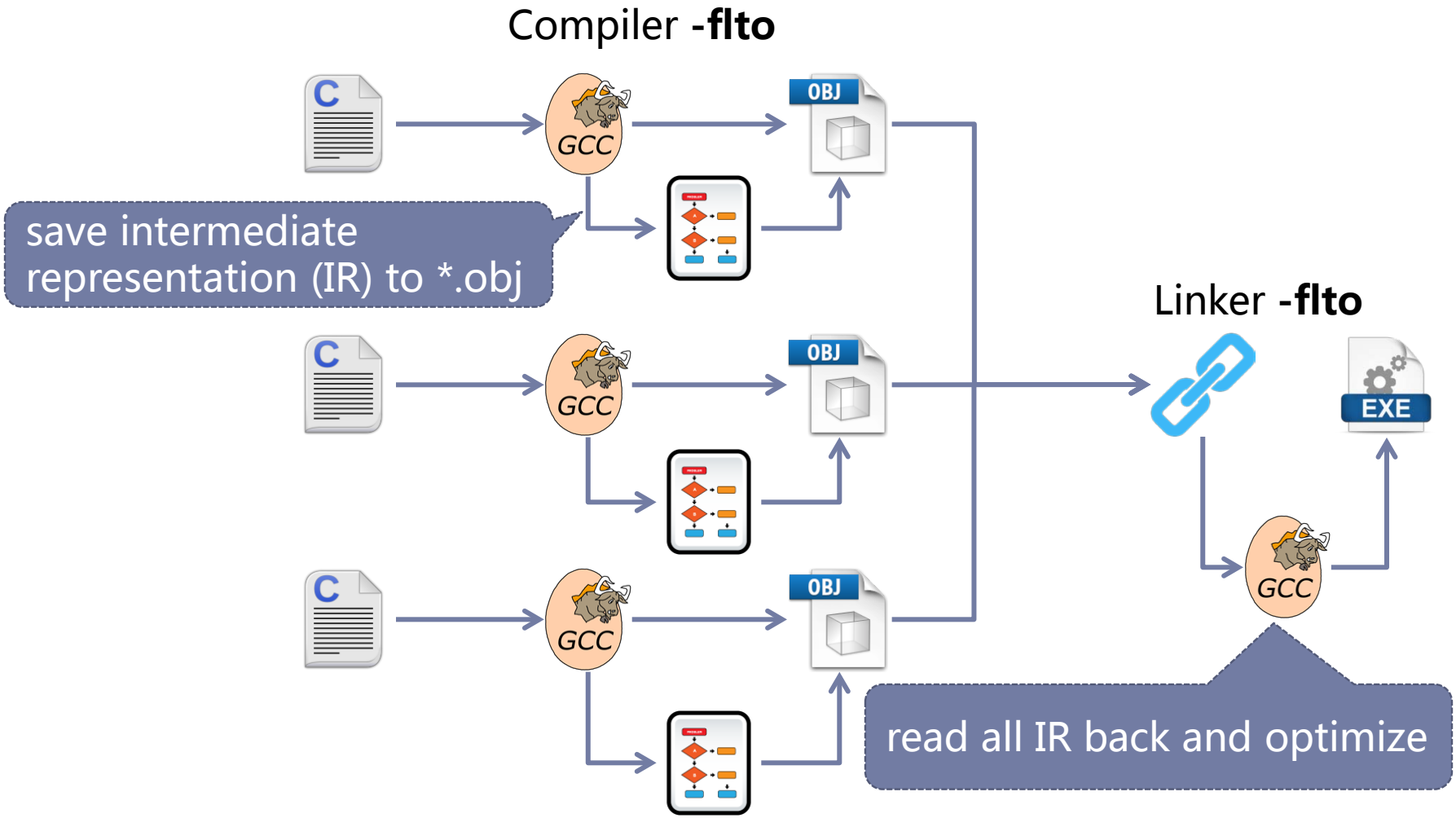


- How to perform
  - More aggressive inter-procedural opts?
  - Or even whole-program opts?

General Software Build Process



Software Build Process with LTO (**-flto**)



Motivation – Stress Testing LTO

- LTO is increasingly important [1,2]
  - Reduce code size by 15-20%
  - Increase speed by 5-15%

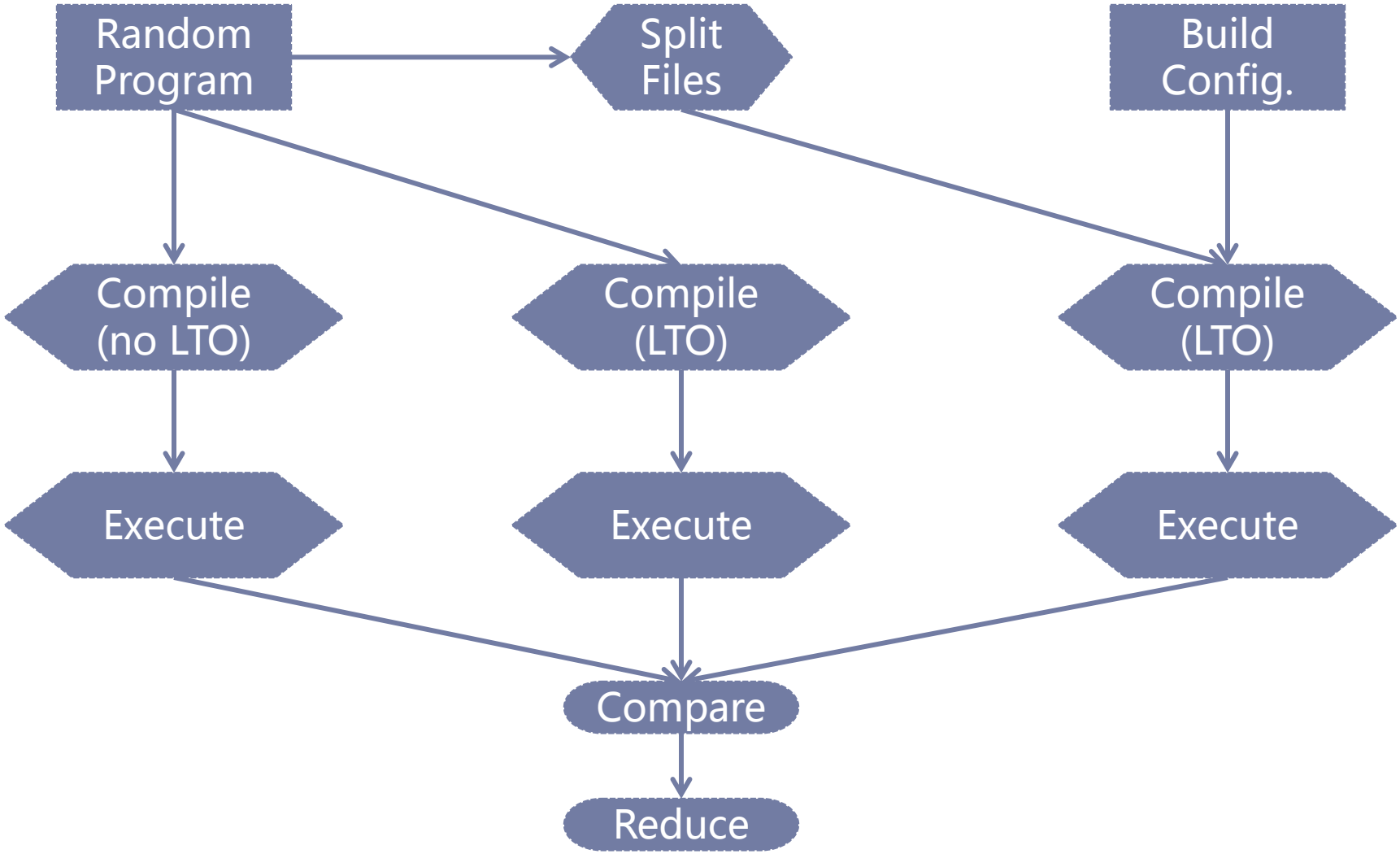
- No effort yet on stress testing LTO
  - Csmith [3] and Orion [4] focus on classical optimizers

[1] B. Anckaert, F. Vandeputte, B. Bus, B. Sutter, and K. Bosschere. Link-Time Optimization of IA64 Binaries. In M. Danelutto, M. Vanneschi, and D. Laforenza, editors, Euro-Par 2004 Parallel Processing
[2] B. De Sutter, L. Van Put, D. Chanet, B. De Bus, and K. De Bosschere. Link-Time Compaction and Optimization of Arm Executables. ACM Trans. Embed. Comput. Syst 2007
[3] X. Yang, Y. Chen, E. Eide, and J. Regehr. Finding and understanding bugs in C compilers. PLDI 2011
[4] V. Le, M. Afshari, and Z. Su. Compiler validation via equivalence modulo inputs. PLDI 2014

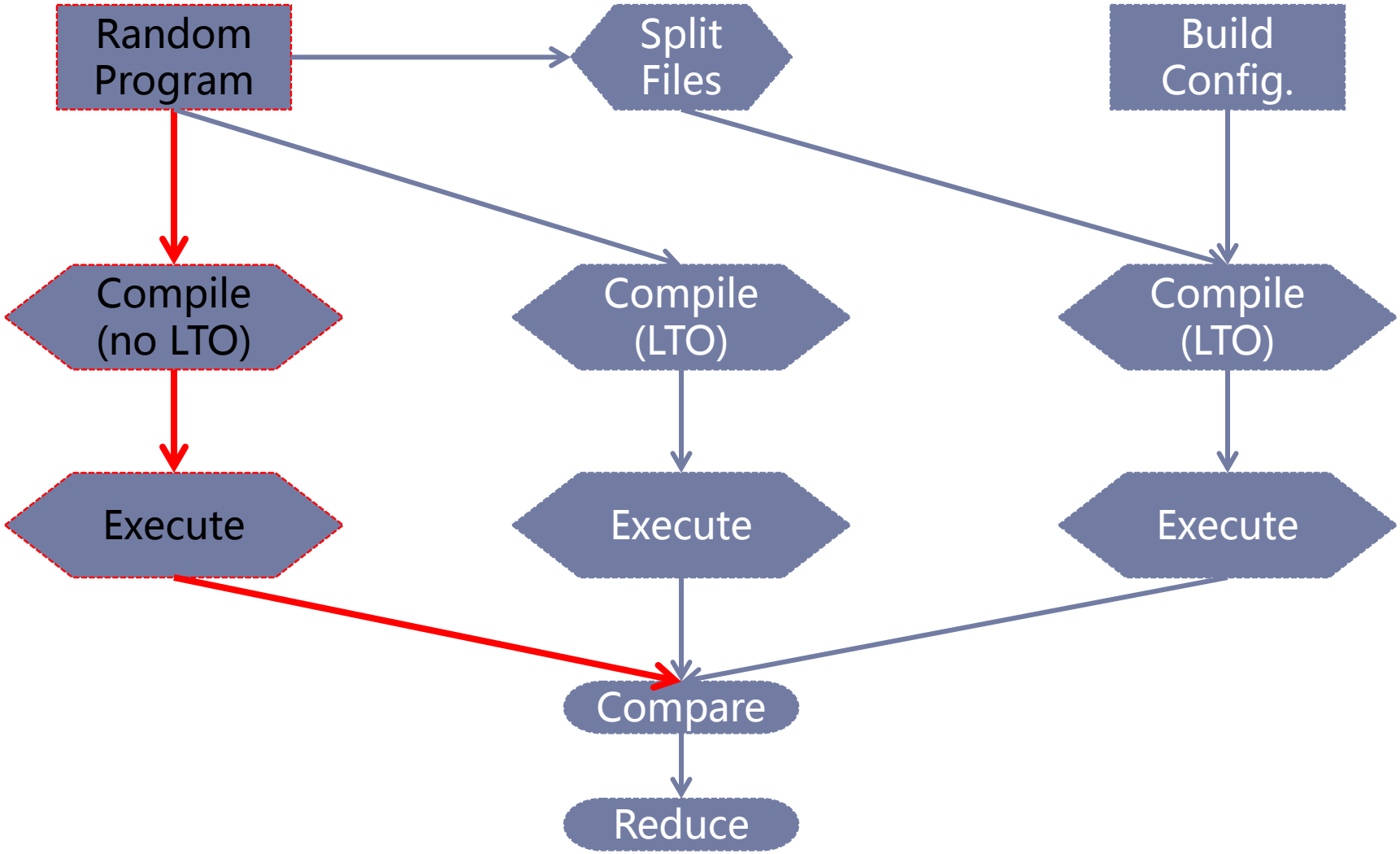
Challenges

- How to generate LTO-relevant test programs?
  - Csmith and Orion generate single-file test programs
- How to reduce bug-triggering test programs?
  - Delta and Creduce, designed for single-file tests

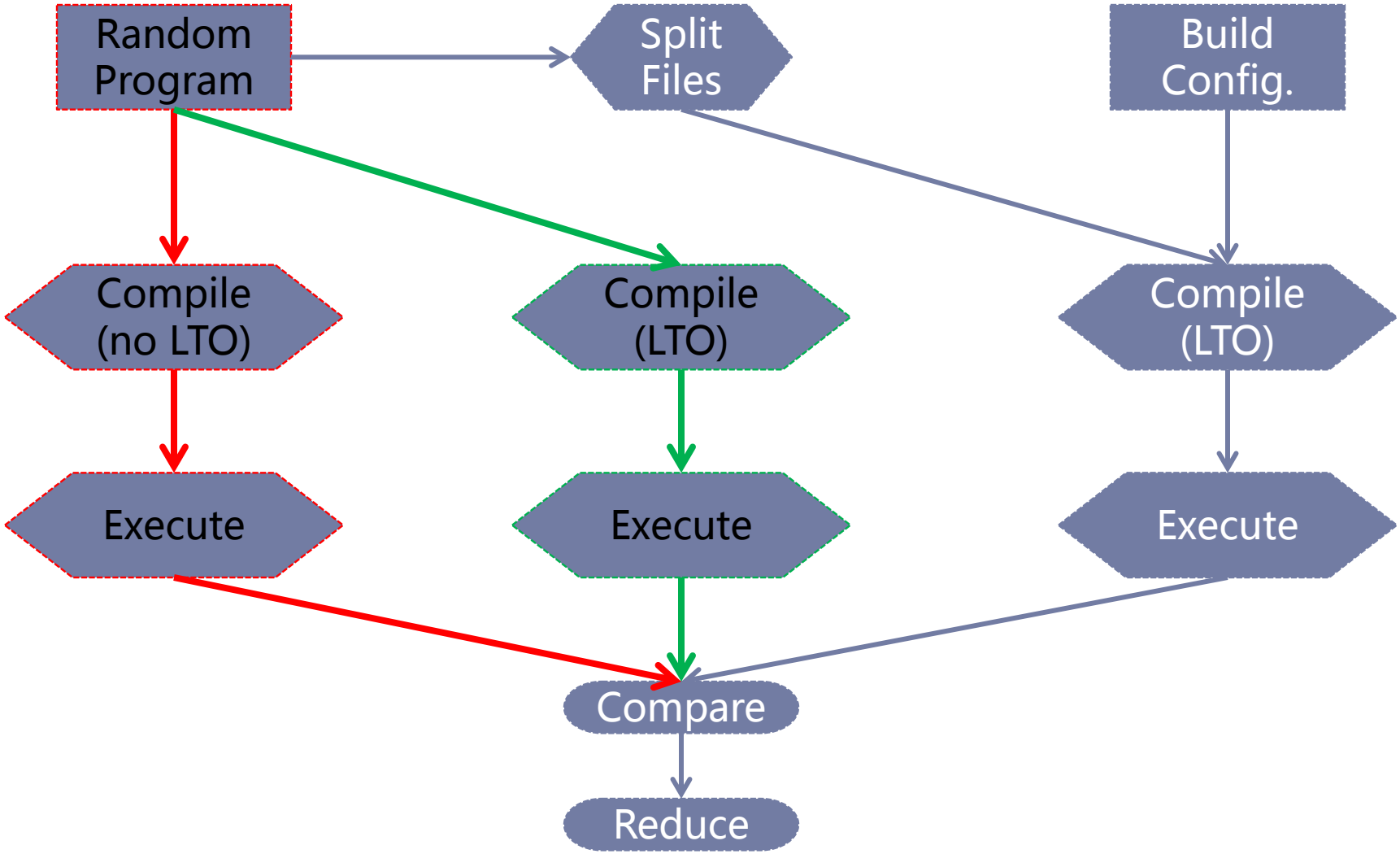
Overall Framework – Differential Testing



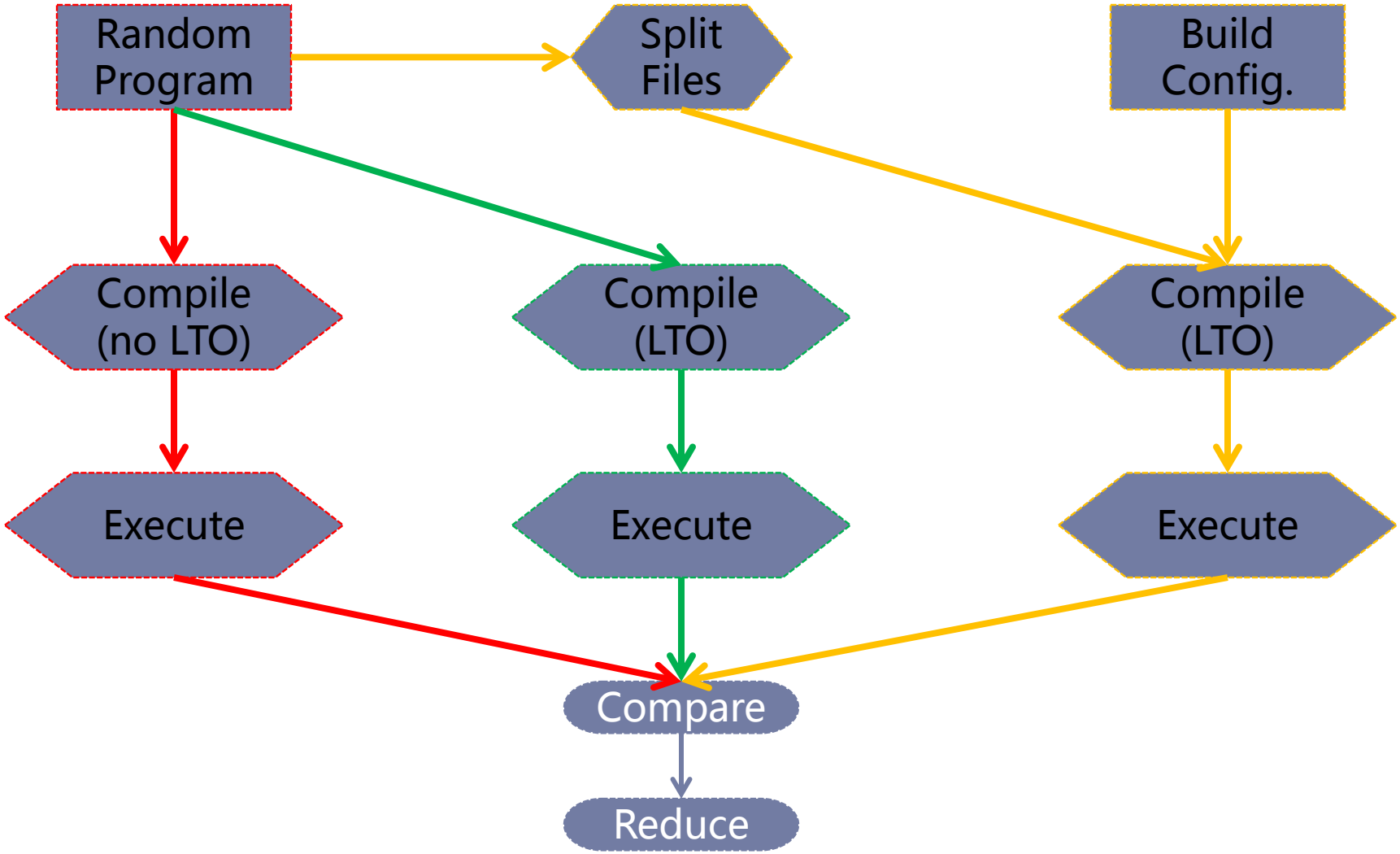
Overall Framework – Differential Testing



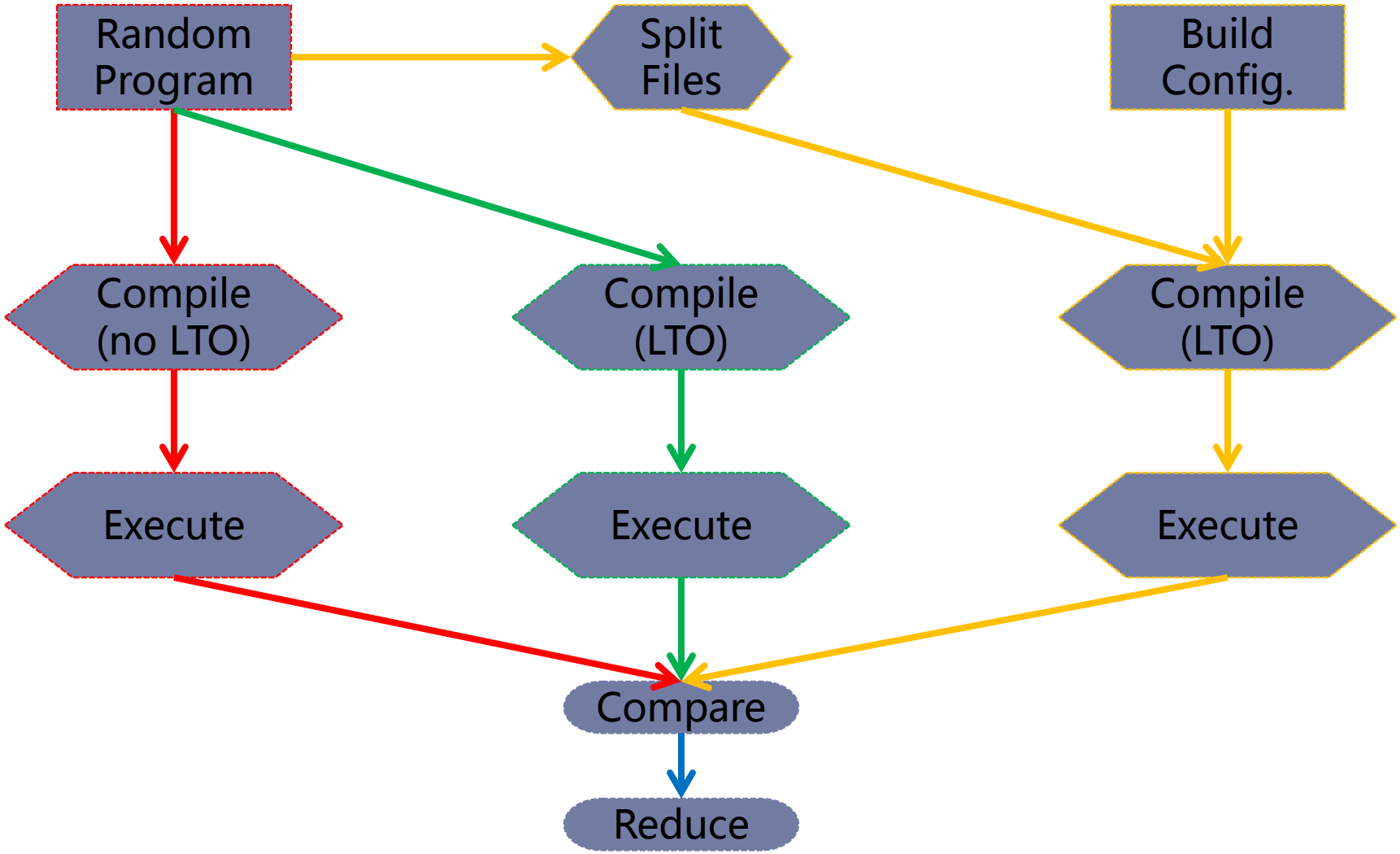
Overall Framework – Differential Testing



Overall Framework – Differential Testing



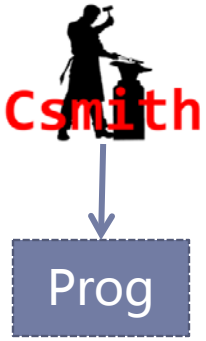
Overall Framework – Differential Testing



Challenge I – Program Generation

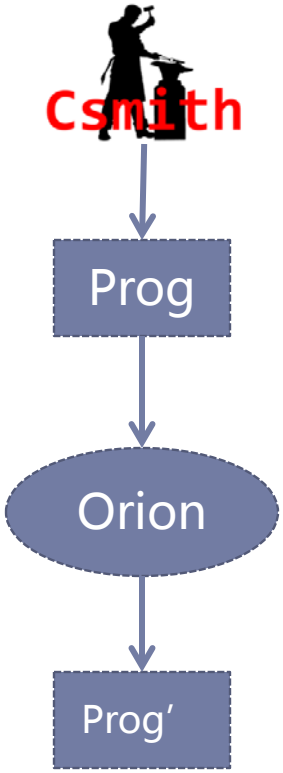
- Leverage existing program generators
- Convert a ***single***-file test to ***multiple*** files
- Maximize the dependencies between source files

Challenge I – Program Generation (1)



Csmith: Generate a random **single-file** program with Csmith

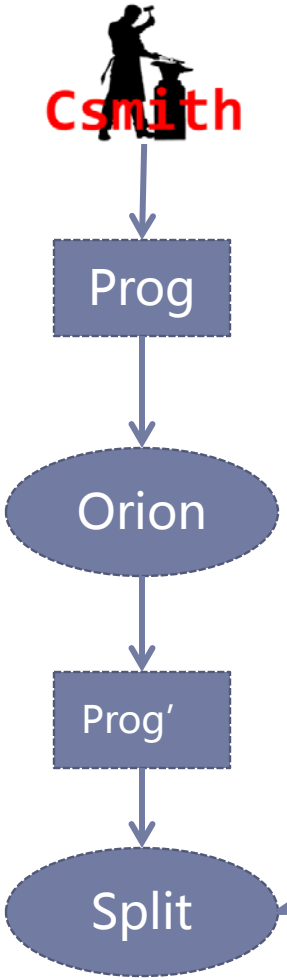
Challenge I – Program Generation (2)



Csmith: Generate a random **single-file** program with Csmith

Orion: Inject arbitrary function calls into dead code regions, to complicate inter-dependencies

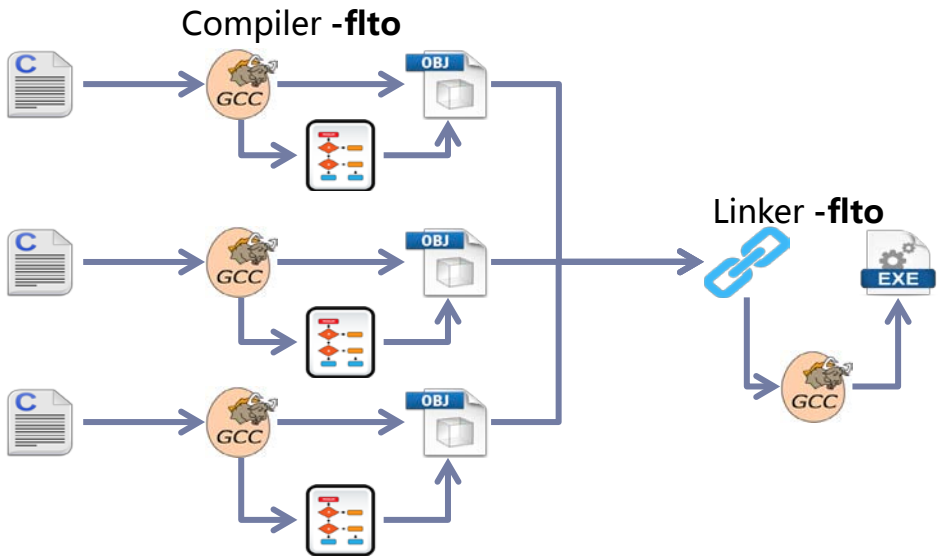
Challenge I – Program Generation (3)



Csmith: Generate a random **single-file** program with Csmith

Orion: Inject arbitrary function calls into dead code regions, to complicate inter-dependencies

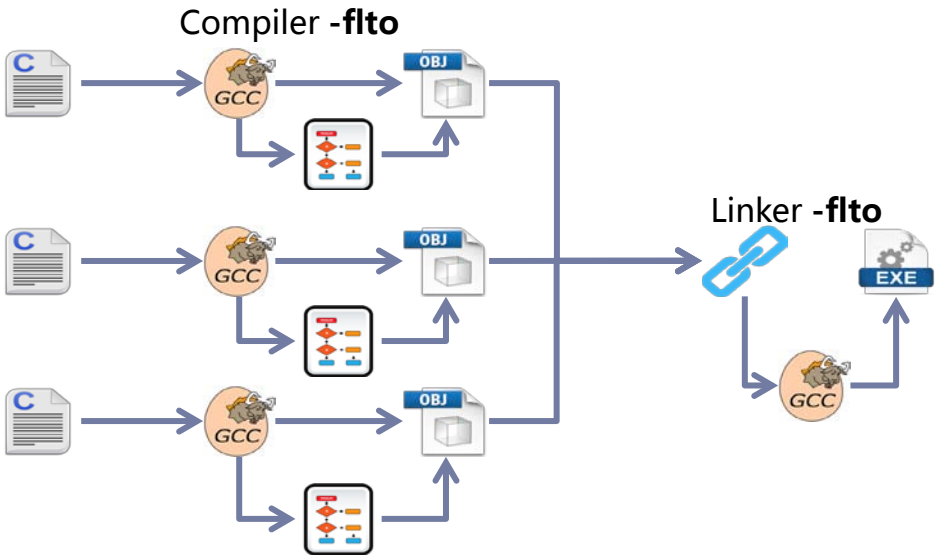
Split: Split the single-file program into multiple files, each file containing one function



Challenge I – Build Configurations

- Describe at which optimization level
  - a translation unit should be compiled
  - all object files should be linked

- Random configurations can further exercise LTO
  - Opt as obfuscators



Challenge I – An Example

```c
/*** small.c ***/
#include <stdio.h>
int a[1] = { 0 }, b = 0;

void fn1 (int p) { }
void fn2 (int p) {
  b = p++;
  fn1 (p);
}
int main () {
  fn2 (0);
  printf ("%d\n", a[b]);
  return 0;
}
```

expected output: 0

Challenge I – An Example

```
/*** small.c ***/
#include <stdio.h>
int a[1] = { 0 }, b = 0;

void fn1 (int p) { }
void fn2 (int p) {
  b = p++;
  fn1 (p);
}
int main () {
  fn2 (0);
  printf ("%d\n", a[b]);
  return 0;
}
```

```
/*** small.h ***/
#include <stdio.h>
int a[1], b;
void fn1 (int p);
void fn2 (int p);
```

```
/*** small.c ***/
#include "small.h"
int a[1] = { 0 }, b = 0;
```

expected output: 0

Challenge I – An Example

```
/*** small.c ***/
#include <stdio.h>
int a[1] = { 0 }, b = 0;

void fn1 (int p) { }
void fn2 (int p) {
  b = p++;
  fn1 (p);
}
int main () {
  fn2 (0);
  printf ("%d\n", a[b]);
  return 0;
}
```

```
/*** small.h ***/
#include <stdio.h>
int a[1], b;
void fn1 (int p);
void fn2 (int p);
```

```
/*** fn1.c ***/
#include "small.h"
void fn1 (int p) { }
```

```
/*** small.c ***/
#include "small.h"
int a[1] = { 0 }, b = 0;
```

expected output: 0

Challenge I – An Example

```c
/*** small.c ***/
#include <stdio.h>
int a[1] = { 0 }, b = 0;

void fn1 (int p) { }
void fn2 (int p) {
  b = p++;
  fn1 (p);
}
int main () {
  fn2 (0);
  printf ("%d\n", a[b]);
  return 0;
}
```

```c
/*** small.h ***/
#include <stdio.h>
int a[1], b;
void fn1 (int p);
void fn2 (int p);
```

```c
/*** fn1.c ***/
#include "small.h"
void fn1 (int p) { }
```

```c
/*** small.c ***/
#include "small.h"
int a[1] = { 0 }, b = 0;
```

```c
/*** fn2.c ***/
#include "small.h"
void fn2 (int p) {
  b = p++;
  fn1 (p);
}
```

expected output: 0

Challenge I – An Example

```
/*** small.c ***/
#include <stdio.h>
int a[1] = { 0 }, b = 0;

void fn1 (int p) { }
void fn2 (int p) {
  b = p++;
  fn1 (p);
}
int main () {
  fn2 (0);
  printf ("%d\n", a[b]);
  return 0;
}
```

```
/*** small.h ***/
#include <stdio.h>
int a[1], b;
void fn1 (int p);
void fn2 (int p);
```

```
/*** fn1.c ***/
#include "small.h"
void fn1 (int p) { }
```

```
/*** main.c ***/
#include "small.h"
int main () {
  fn2 (0);
  printf ("%d\n", a[b]);
  return 0;
}
```

```
/*** small.c ***/
#include "small.h"
int a[1] = { 0 }, b = 0;
```

```
/*** fn2.c ***/
#include "small.h"
void fn2 (int p) {
  b = p++;
  fn1 (p);
}
```

expected output: 0

Challenge I – An Example

```
/*** small.c ***/
#include <stdio.h>
int a[1] = { 0 }, b = 0;

void fn1 (int p) { }
void fn2 (int p) {
  b = p++;
  fn1 (p);
}
```

```
int main () {
  fn2 (0);
  printf ("%d\n", a[b]);
  return 0;
}
```

expected output: 0
real output : 1

```
/*** small.h ***/
#include <stdio.h>
int a[1], b;
void fn1 (int p);
void fn2 (int p);
```

```
/*** fn1.c ***/
#include "small.h"
void fn1 (int p) { }
```

```
/*** main.c ***/
#include "small.h"
int main () {
  fn2 (0);
```

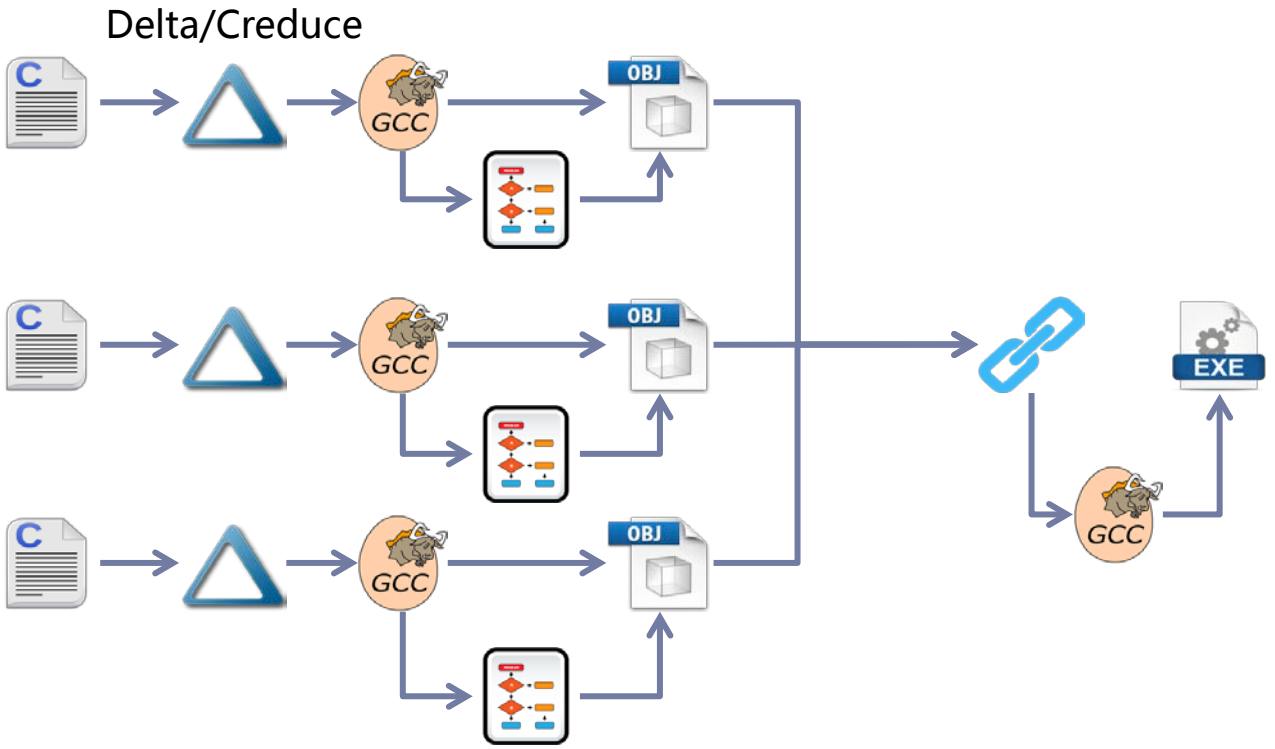
```
/*** small.c ***/
#include "small.h"
int a[1] = { 0 }, b = 0;
```

```
/*** fn2.c ***/
#include "small.h"
void fn2 (int p) {
  b = p++;
  fn1 (p);
}
```

```
/*** configuration ***/
gcc -flto -O1 -c fn1.c
gcc -flto -O1 -c fn2.c
gcc -flto -O1 -c main.c
gcc -flto -O1 -c t.c
gcc -flto -O0 fn1.o fn2.o main.o t.o
```

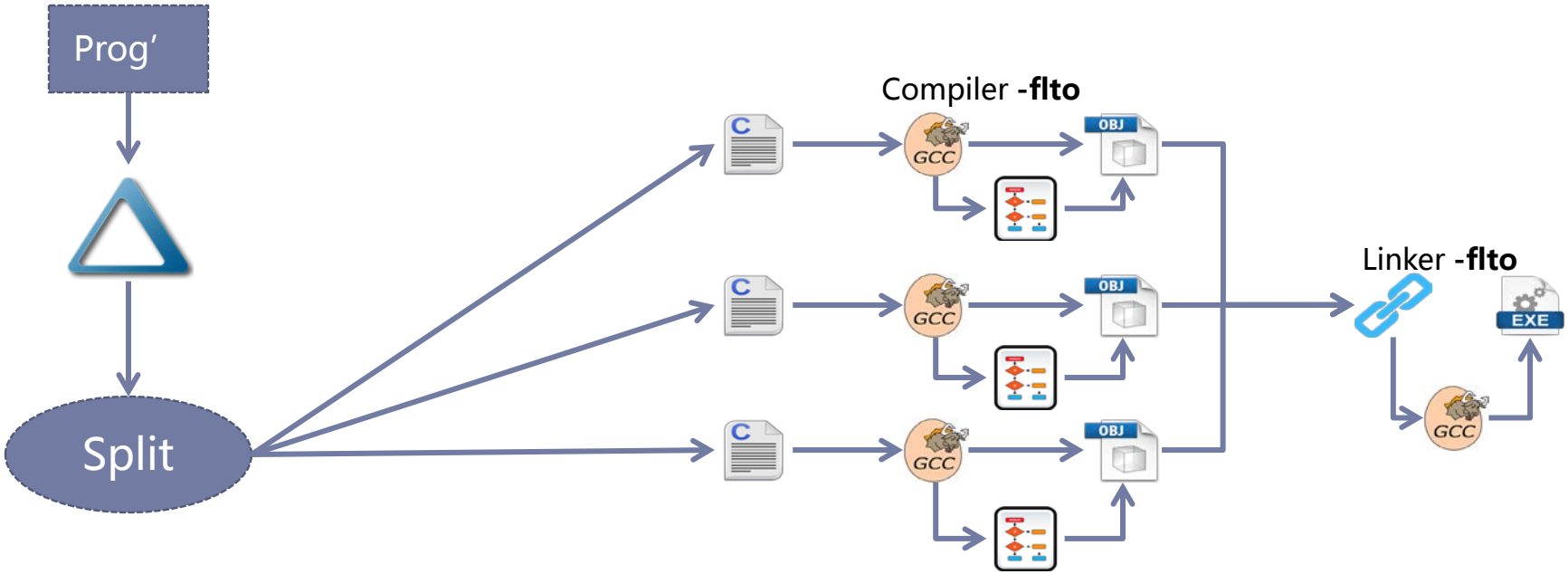
Challenge II – Reducing Test Programs

- Reducing multiple files is challenging
  - Interdependencies between translation units
  - Avoiding undefined behaviors (CompCert)



Challenge II – Reducing Test Programs

- Reducing multiple files is challenging
  - Interdependencies between translation units
  - Avoiding introducing undefined behaviors
- Instead, we reduce the single-file test program



Evaluation

- Two multi-core Ubuntu machines
- February 2014 – Janurary 2015
- 37 valid bug reports to GCC and LLVM (11 fixed)

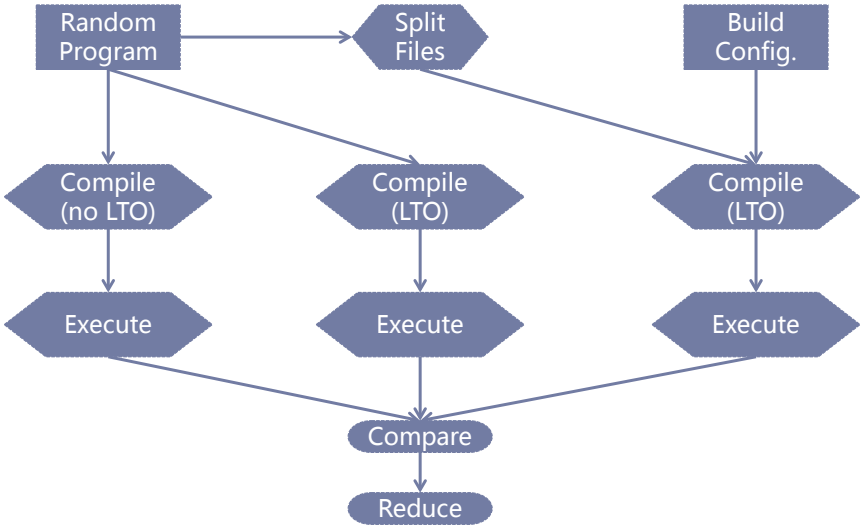
Bug Classification

	GCC	LLVM	Total
Wrong code	6 **(5 fixed)**	22 **(0 fixed)**	28
Crash	5 **(5 fixed)**	0	5
Linker Error	1 **(1 fixed)**	3 **(0 fixed)**	4

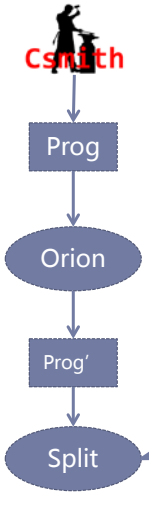
Conclusion

- the first effort to stress-test LTO
- transformation way to generate test programs
- an effective technique to reduce LTO bugs
- 11 months, 37 valid bugs in GCC and LLVM

Overall Framework – Differential Testing



Challenge I – Program Generation (3)



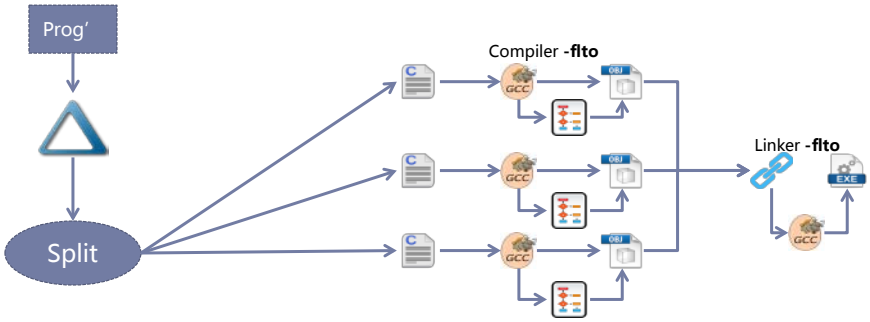
Csmith: Generate a random **single-file** program with Csmith

Orion: Inject arbitrary function calls into dead code regions, to complicate inter-dependencies

Split: Split the single-file program into multiple files, each file containing one function

Challenge II – Reducing Test Programs

- Reducing multiple files is challenging
  - Interdependencies between translation units
  - Avoiding introducing undefined behaviors
- Instead, we reduce the single-file test program



Bug Classification

	GCC	LLVM	Total
Wrong code	6 **(5 fixed)**	22 **(0 fixed)**	28
Crash	5 **(5 fixed)**	0	5
Linker Error	1 **(1 fixed)**	3 **(0 fixed)**	4