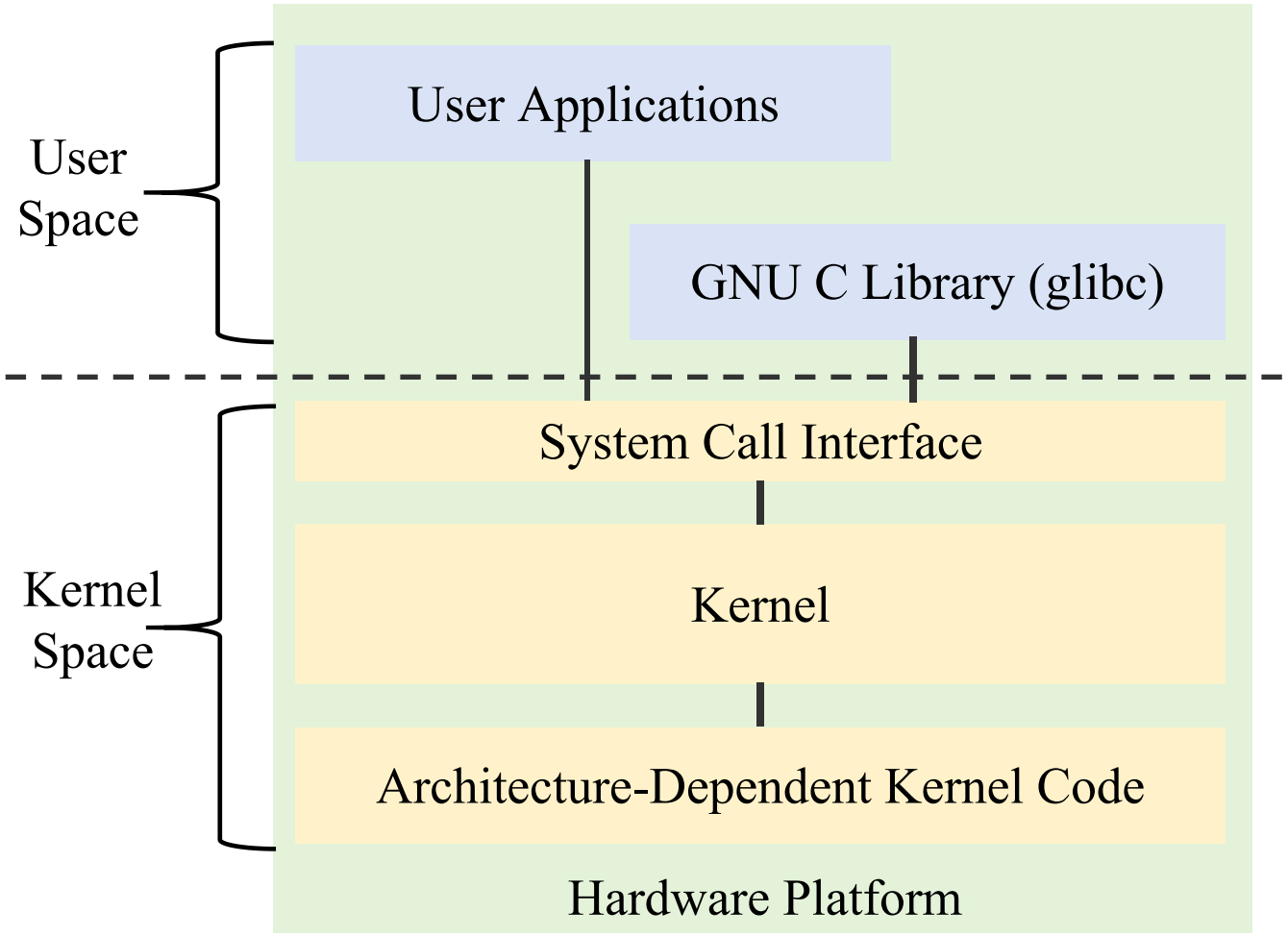
Optimizing Input Minimization in Kernel Fuzzing

Hui Guo, Hao Sun, Shan Huang, Ting Su, Geguang Pu, Shaohua Li

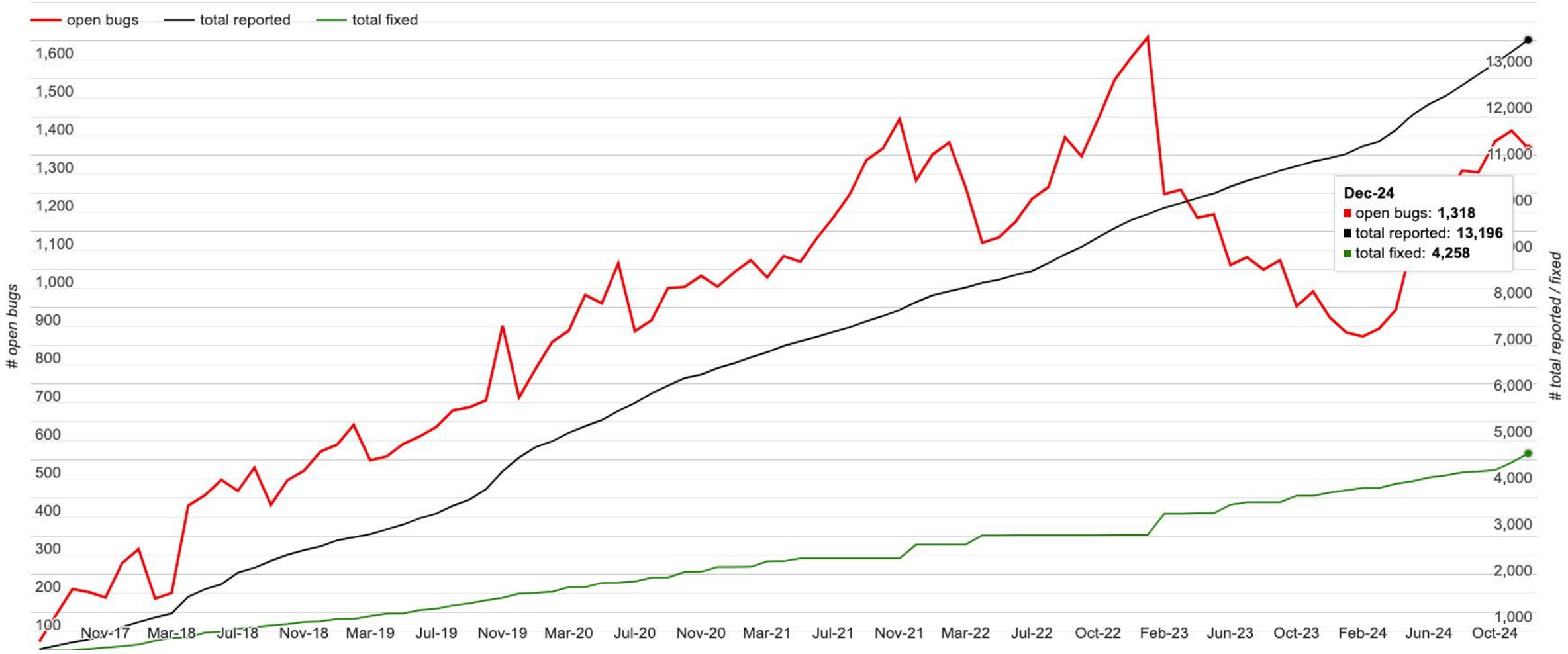
Importance of Kernel Security



GNU/Linux (Ubuntu, Fedora, CentOS, Debian)

Kernel Vulnerabilities:
System-Wide Domino Effects

Kernel Fuzzing: An Effective Technique for Kernel Security

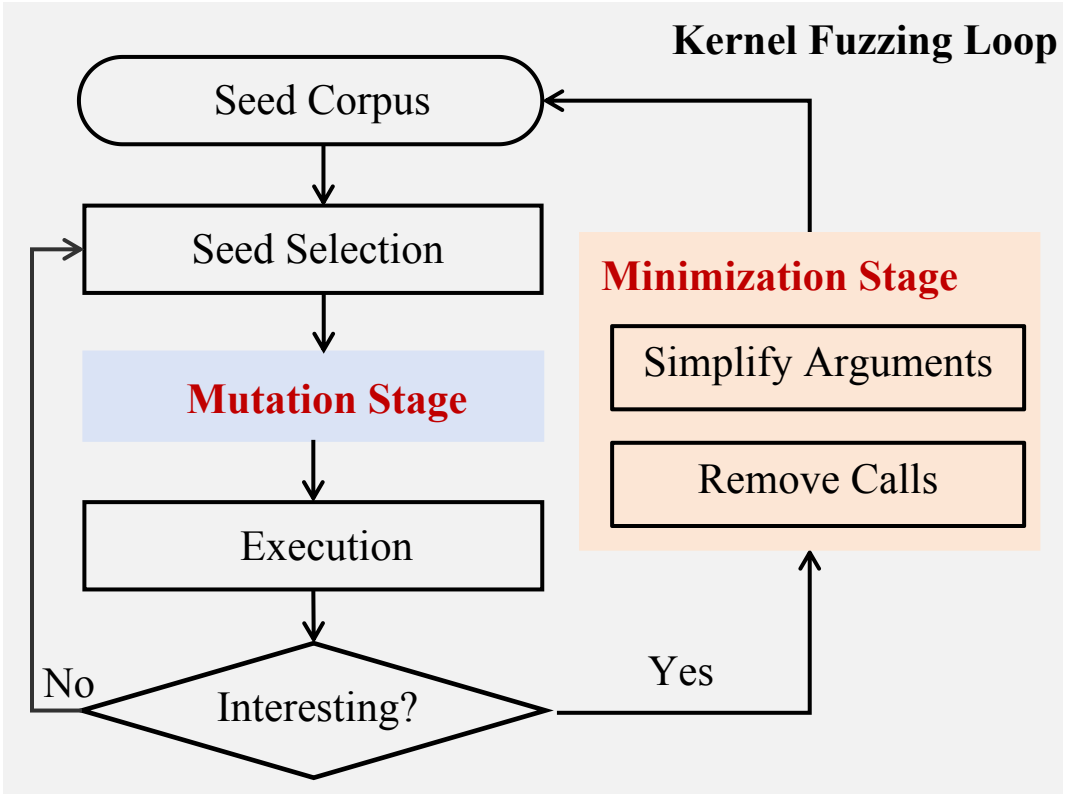


Syzkaller: a coveriage-guided kernel fuzzer has uncovered over 5000 bugs in Linux kernel

Workflow of Coverage-guided Kernel Fuzzing

Mutate seeds to find interesting programs

Mutation Stage



Overall Workflow of Kernel Fuzzing

Remove coverage-irrelevant calls/args to improve mutation efficacy

Minimization Stage

Syzkaller: One-by-one Minimization Strategy

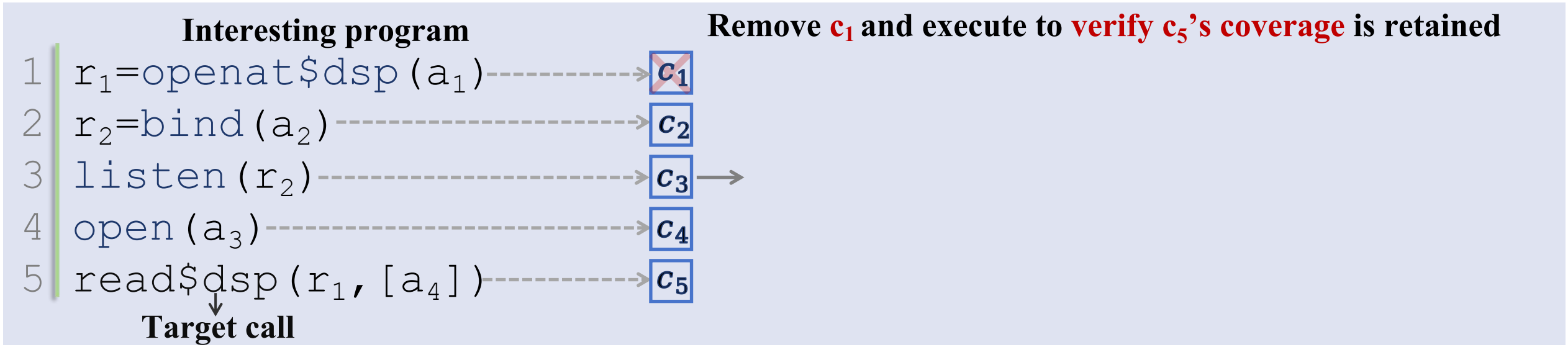
Interesting program

```
1 r1=openat$dsp(a1)
2 r2=bind(a2)
3 listen(r2)
4 open(a3)
5 read$dsp(r1,[a4])
```

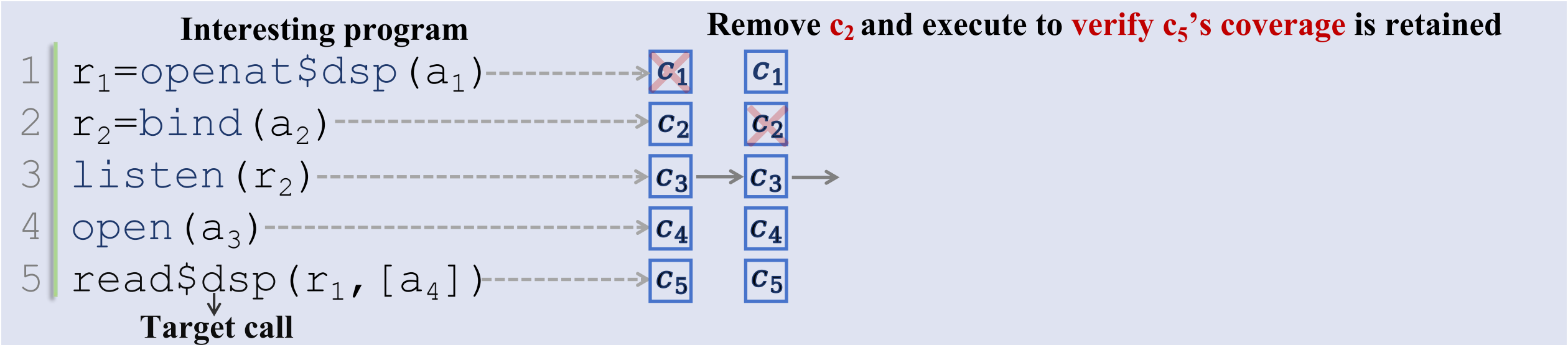
Target call

The goal is to remove irrelevant calls while preserving the new coverage of target call

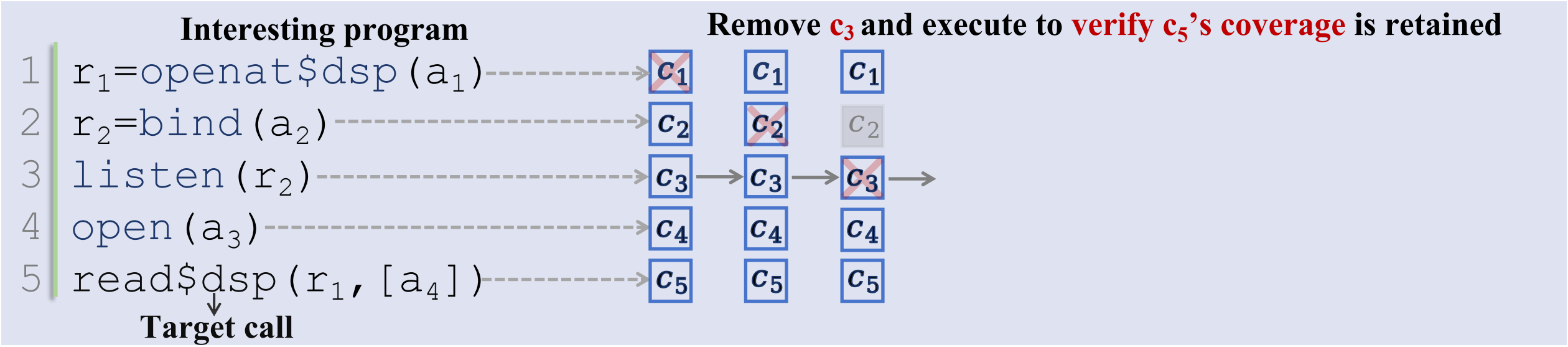
Syzkaller: One-by-one Minimization Strategy



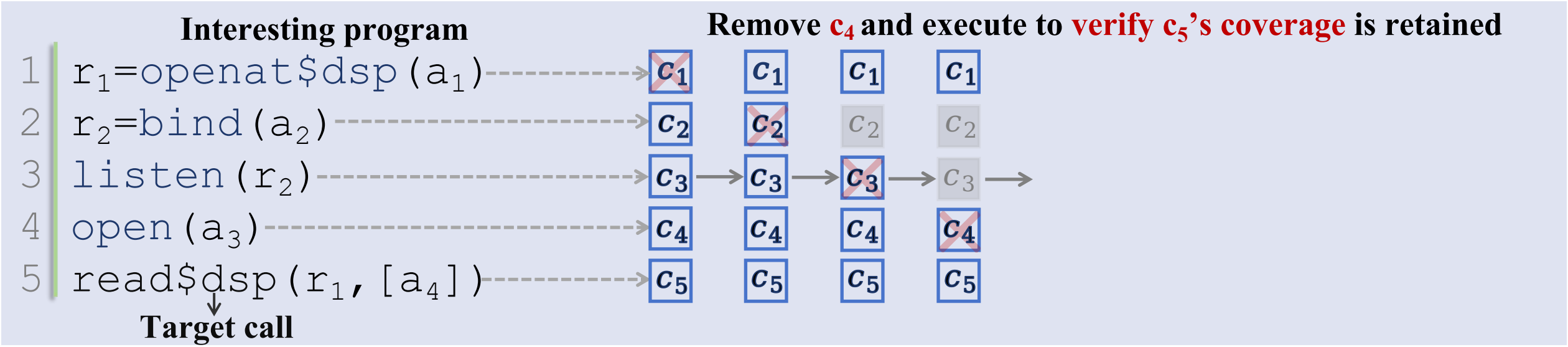
Syzkaller: One-by-one Minimization Strategy



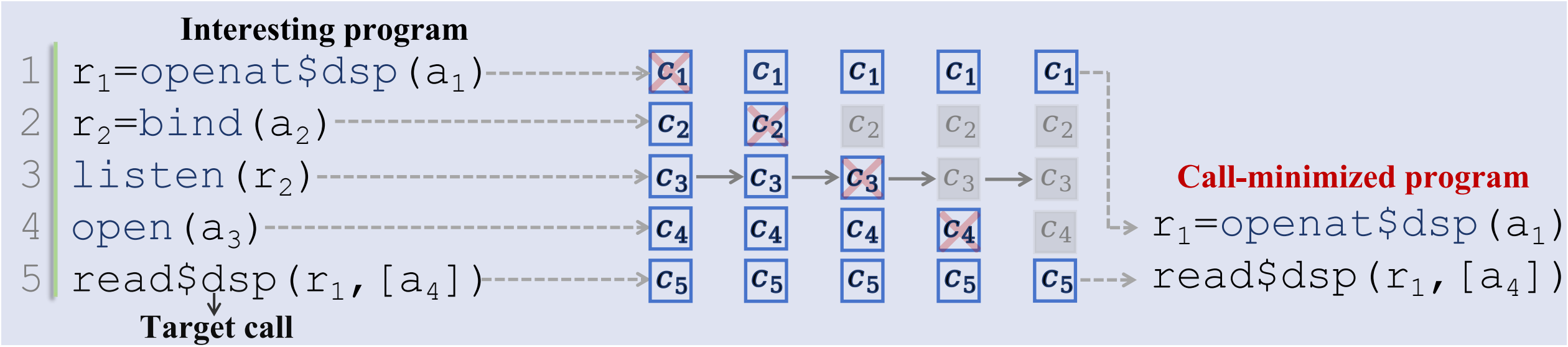
Syzkaller: One-by-one Minimization Strategy



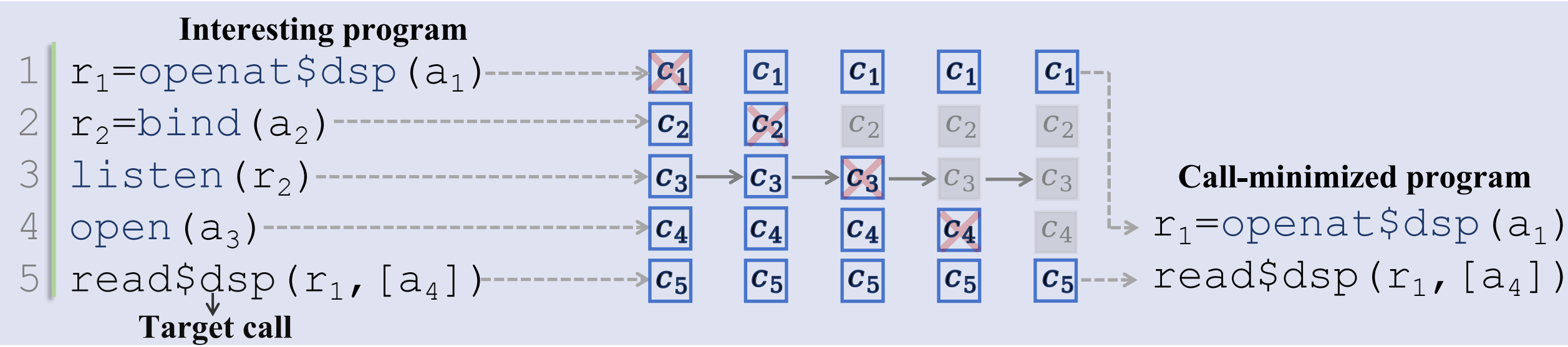
Syzkaller: One-by-one Minimization Strategy



Syzkaller: One-by-one Minimization Strategy

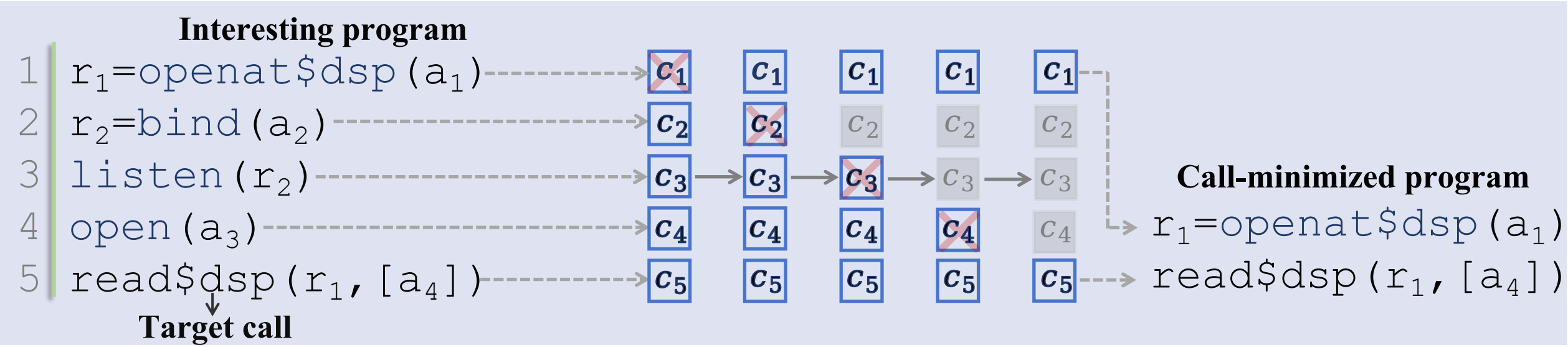


Syzkaller: One-by-one Minimization Strategy



One-by-one call removal consumes four executions

Syzkaller: One-by-one Minimization Strategy



One-by-one call removal consumes four executions

Next step is to simplify arguments

Syzkaller: One-by-one Minimization Strategy

Call-minimized program

```
r1=openat$dsp(a1)
read$dsp(r1,[a4])
```

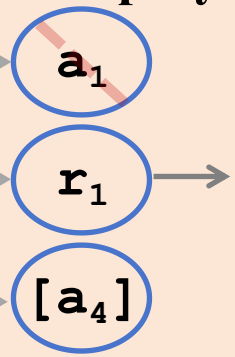
The goal is to simplify each argument while preserving the new coverage of target call

Syzkaller: One-by-one Minimization Strategy

Simplify a_1 and execute to verify c_5's coverage is retained

Call-minimized program

```
r1=openat$dsp(a1)
read$dsp(r1,[a4])
```

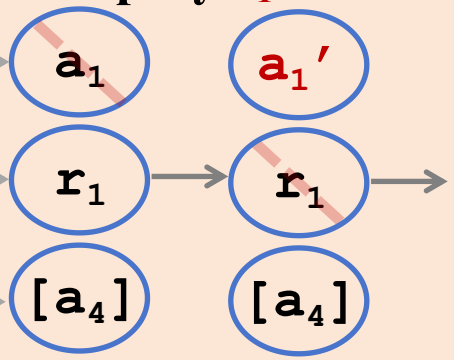


Syzkaller: One-by-one Minimization Strategy

Simplify r_1 and execute to verify c_5's coverage is retained

Call-minimized program

```
r1=openat$dsp(a1)
read$dsp(r1,[a4])
```

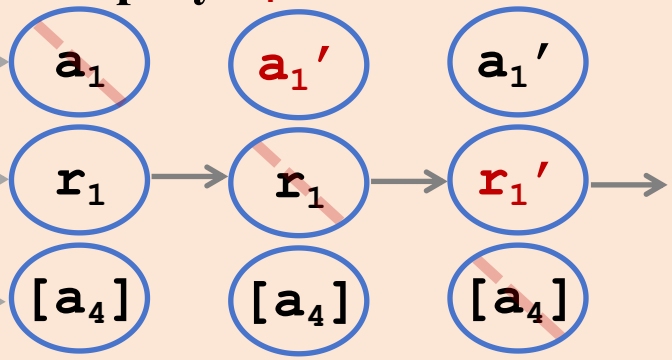


Syzkaller: One-by-one Minimization Strategy

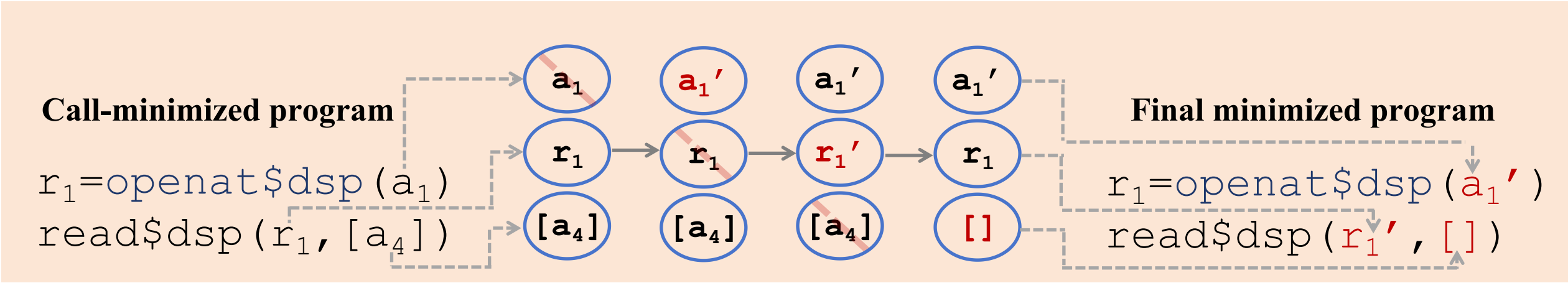
Simplify a_4 and execute to verify c_5's coverage is retained

Call-minimized program

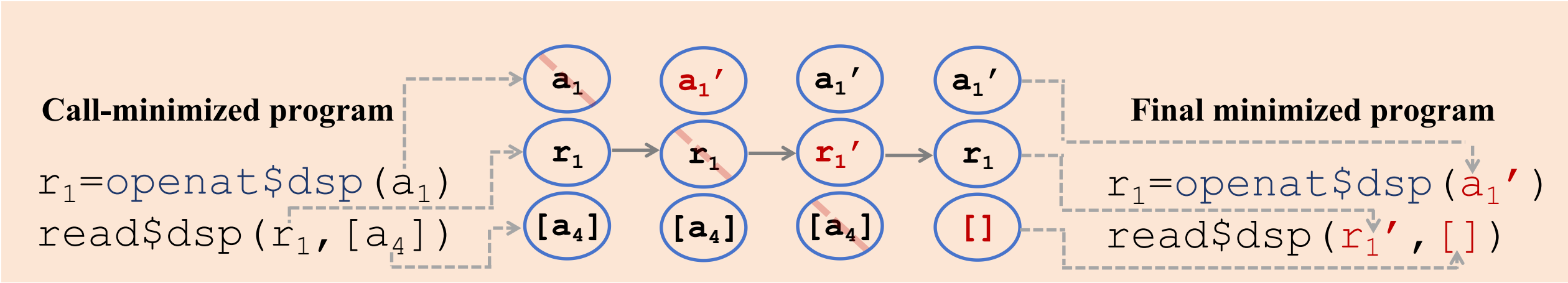
```
r1=openat$dsp(a1)
read$dsp(r1,[a4])
```



Syzkaller: One-by-one Minimization Strategy

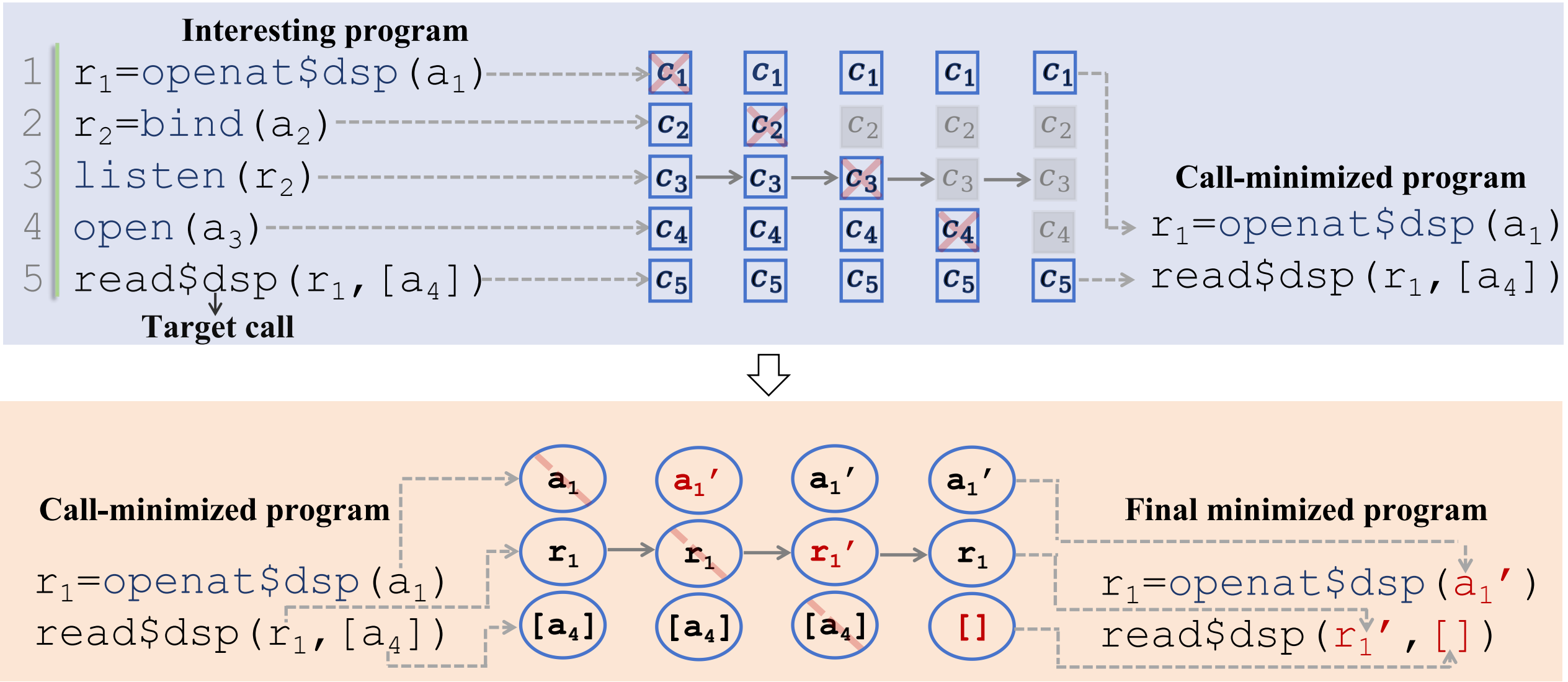


Syzkaller: One-by-one Minimization Strategy



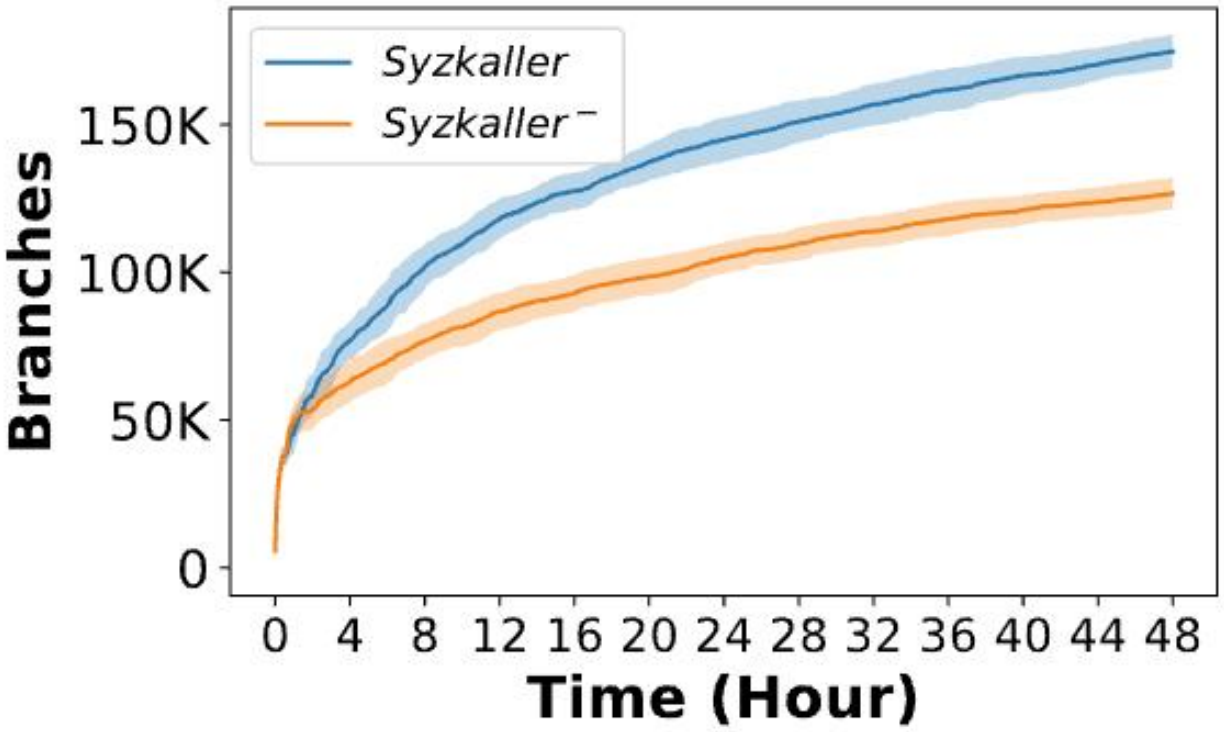
One-by-one arg simplification consumes three executions

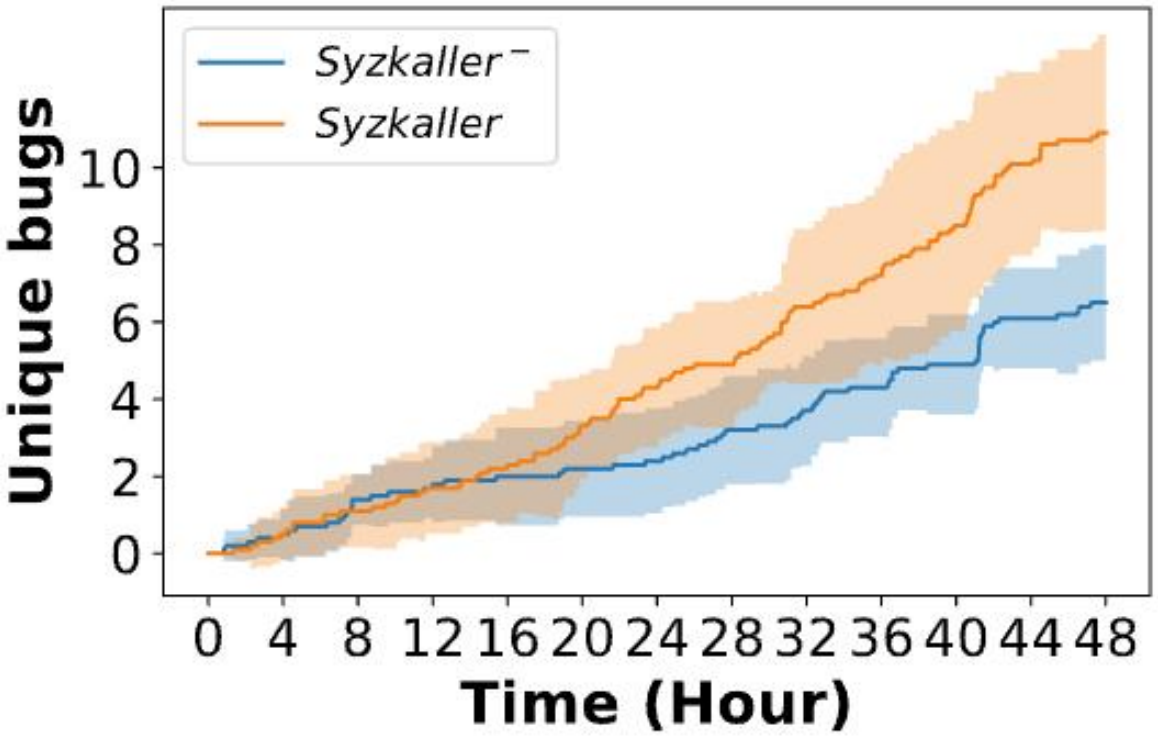
Syzkaller: One-by-one Minimization Strategy



One-by-one minimization strategy consumes a total of **seven** executions

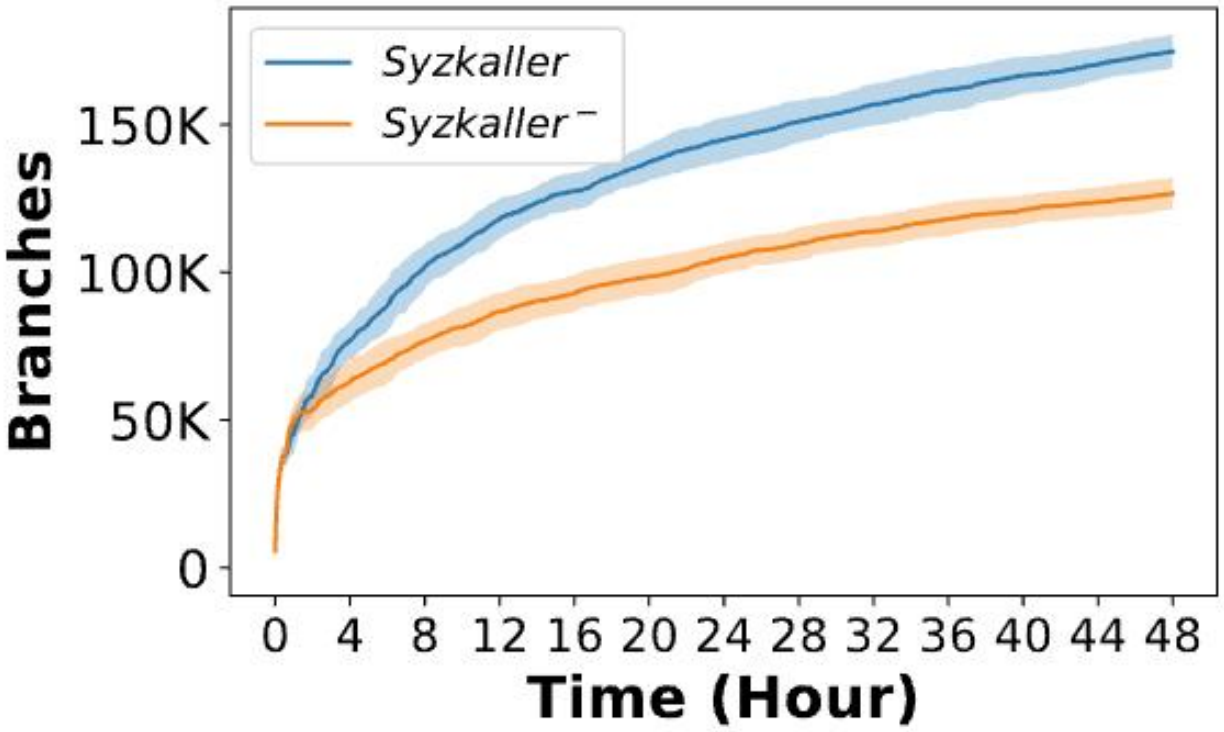
Tension: Benefit and Cost of Minimization Stage

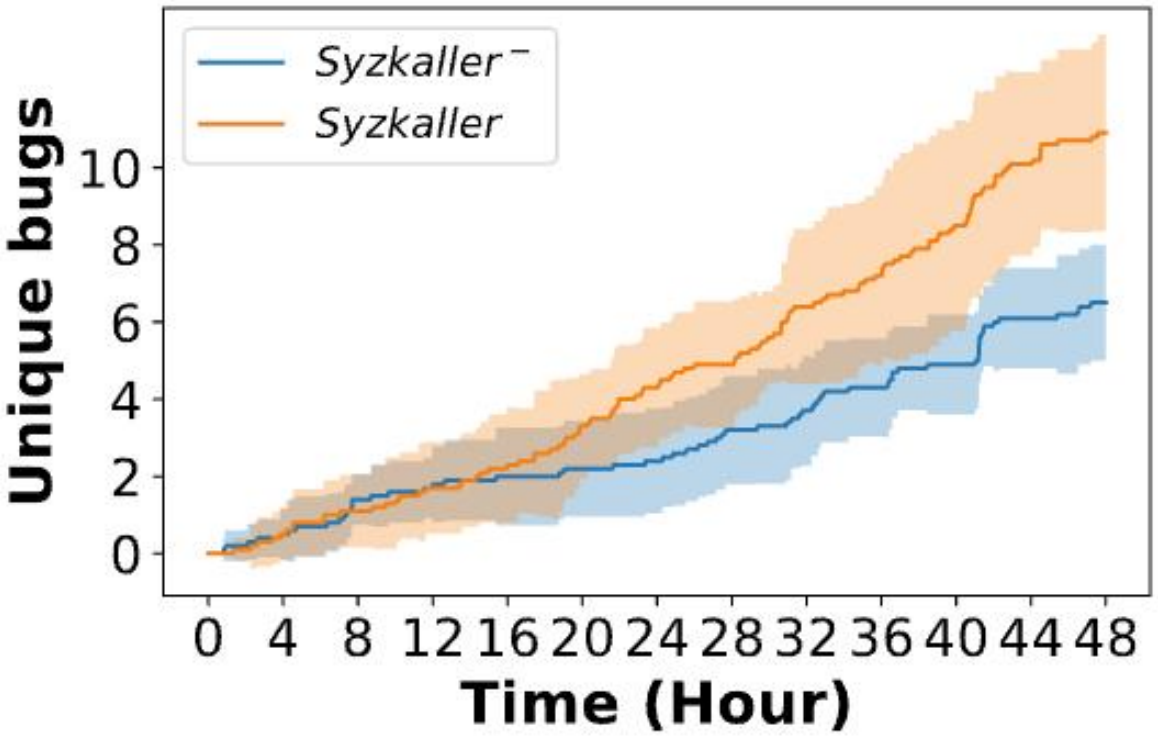




Syzkaller: with minimization stage
Syzkaller⁻ : without minimization stage

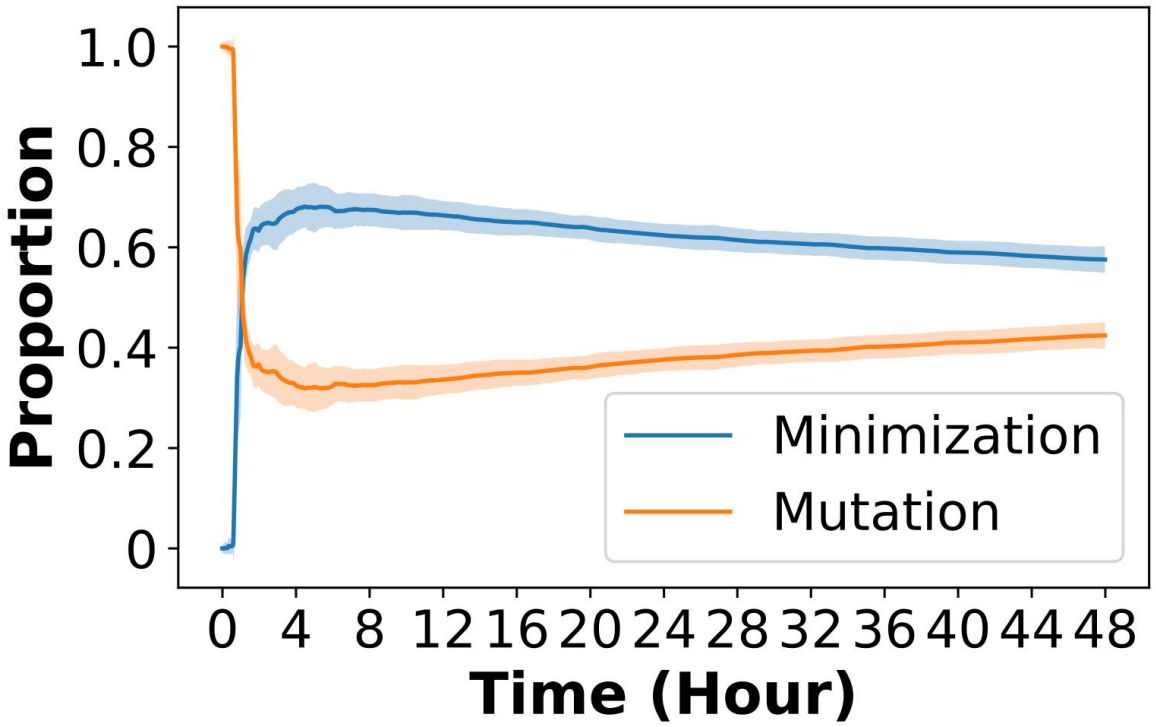
Tension: Benefit and Cost of Minimization Stage

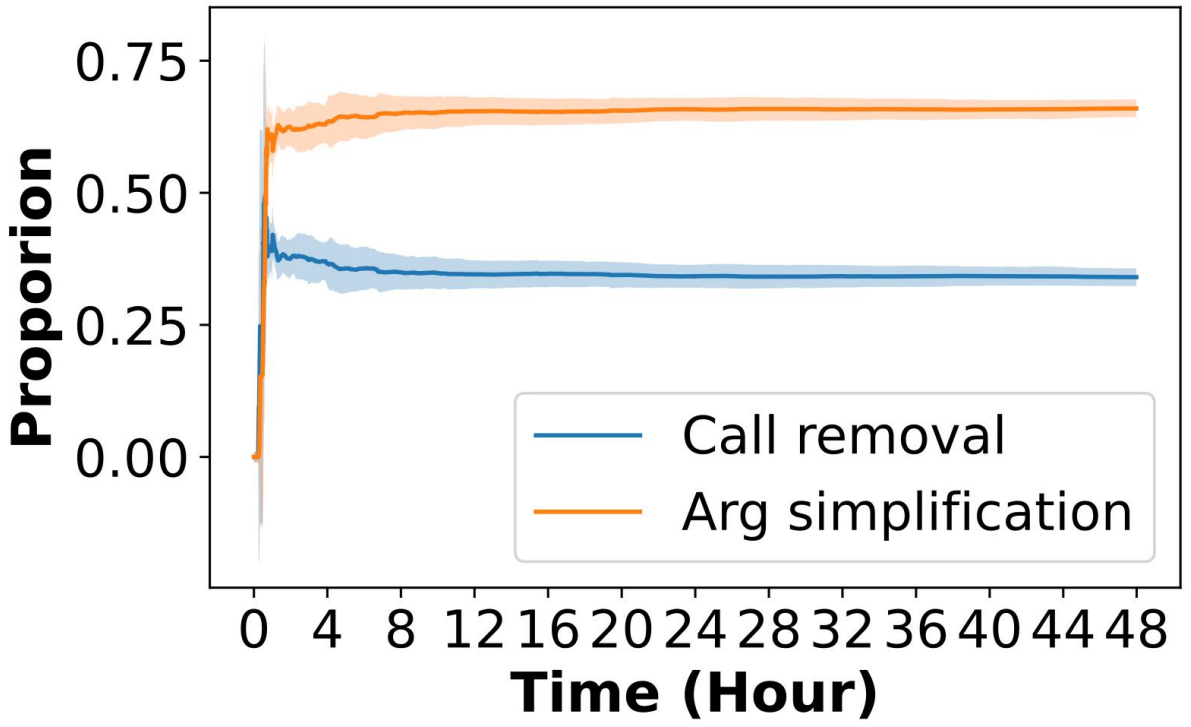




Without minimization stage,
coverage and number of found bugs respectively decrease by 27.5% and 40.4%.

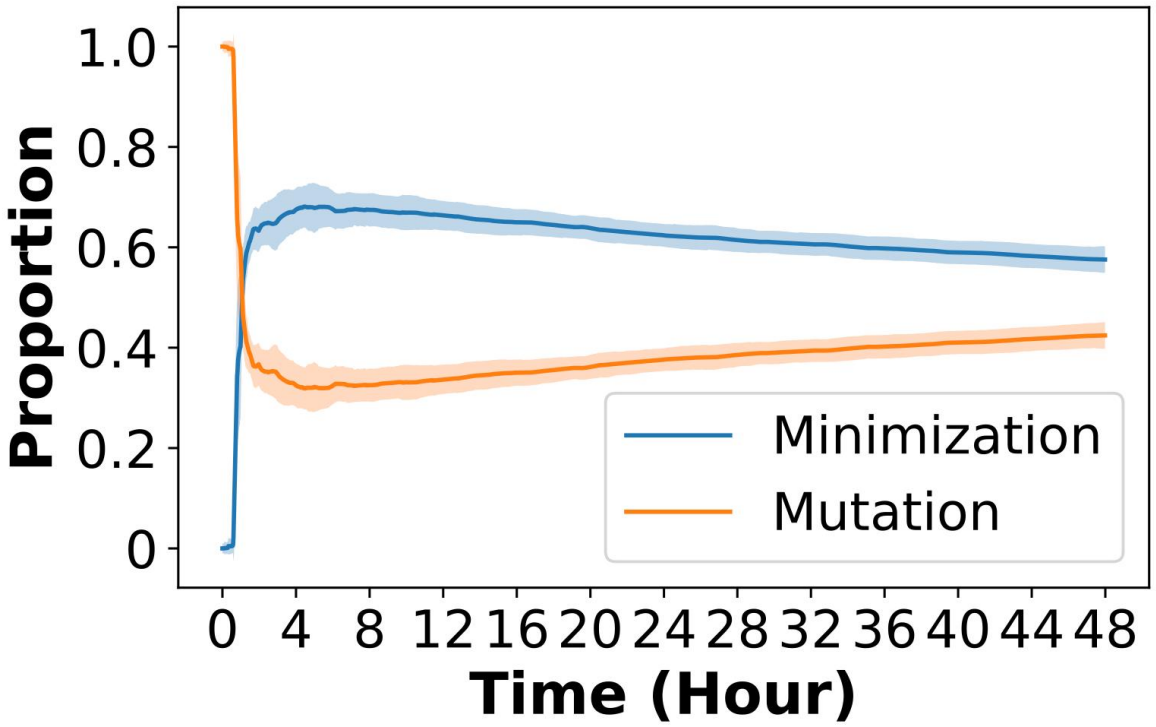
Tension: Benefit and Cost of Minimization Stage

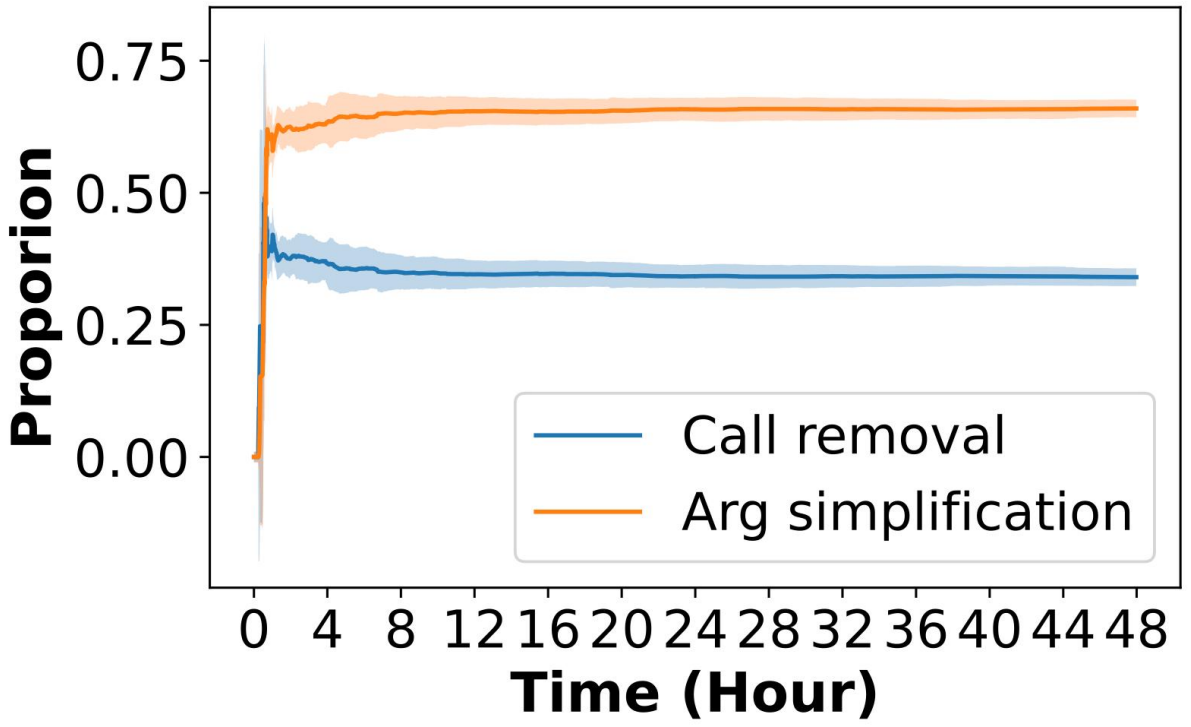




Count the proportion of program executions as the cost of minimization and mutation, because each mutate or minimize require one program execution.

Tension: Benefit and Cost of Minimization Stage





Proportion of program executions during minimization stage accounts for 57.5%, leading to significant cost.

Tension: Benefit and Cost of Minimization

Benefit: Improve the effectiveness of kernel fuzzing

Cost: Consume over half of the fuzzing resources

Challenge to mitigate tension

How to reduce the minimization cost while keeping the minimization benefit?

Our Key Idea & Contribution

- Our idea:
  Reduce the number of program executions for verifying whether new coverage is preserved—the fewer the program executions are, the lower the cost is

Our Key Idea & Contribution

- Our idea:
  Reduce the number of program executions for verifying whether new coverage is preserved—the fewer the program executions are, the lower the cost is

- Our optmization strategy:
  - Influence-guided call removal strategy
  - Type-informed argument simplification strategy

Our Key Idea & Contribution

- Our idea:
  Reduce the number of program executions for verifying whether new coverage is preserved—the fewer the program executions are, the lower the cost is

- Our optmization strategy:
  - Influence-guided call removal strategy
  - Type-informed argument simplification strategy

- Result:
  Reduce the minimization cost of Syzkaller by 60.7%

Influence-guided Call Removal Strategy

Insight:

Irrelevant calls can be **removed at once rather than one by one**, as they does not affect new achieved coverage

Influence-guided Call Removal Strategy

Insight:

Irrelevant calls can be **removed at once rather than one by one**, as they does not affect new achieved coverage

Interesting program

```
1 r1=openat$dsp(a1)
2 r2=bind(a2)
3 listen(r2)
4 open(a3)
5 read$dsp(r1,[a4])
```

Target call

Influence-guided Call Removal Strategy

Insight:

Irrelevant calls can be **removed at once rather than one by one**, as they does not affect new achieved coverage

Interesting program

```
1 r1=openat$dsp(a1)
2 r2=bind(a2)
3 listen(r2)
4 open(a3)
5 read$dsp(r1,[a4])
```

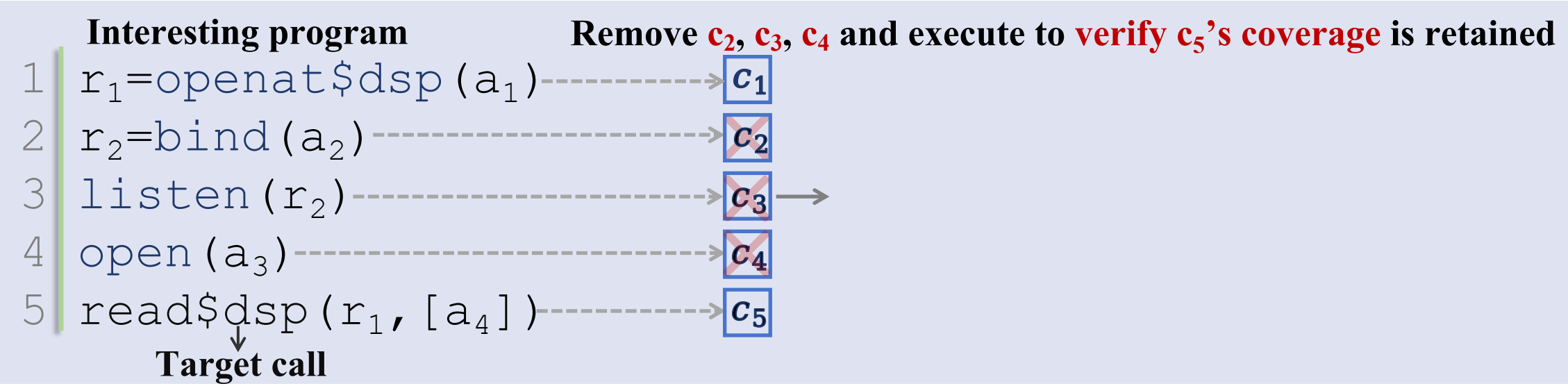
Target call

Assuming c_2, c_3, c_4 does not affect c_5

Influence-guided Call Removal Strategy

Insight:

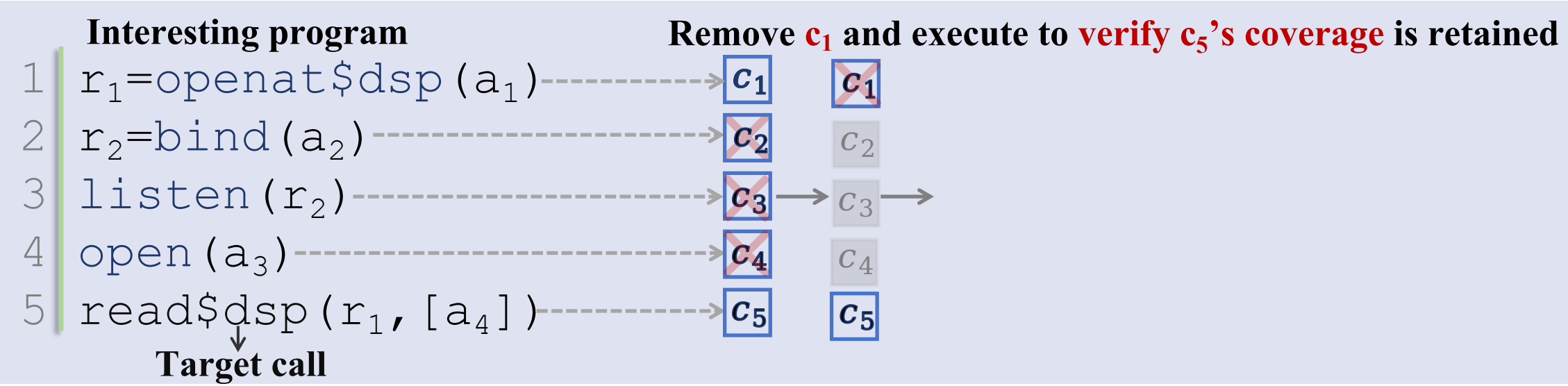
Irrelevant calls can be **removed at once rather than one by one**, as they does not affect new achieved coverage



Influence-guided Call Removal Strategy

Insight:

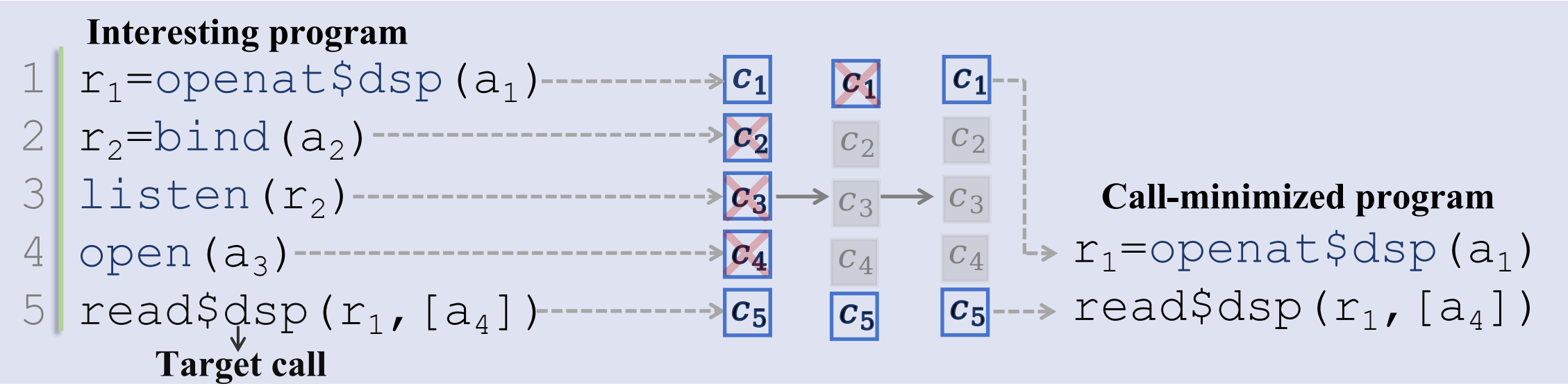
Irrelevant calls can be **removed at once rather than one by one**, as they does not affect new achieved coverage



Influence-guided Call Removal Strategy

Insight:

Irrelevant calls can be **removed at once rather than one by one**, as they does not affect new achieved coverage



Influence-guided call removal strategy consumes two executions

Influence-guided Call Removal Strategy

Insight:

Irrelevant calls can be **removed at once rather than one by one**, as they does not affect new achieved coverage

Strategy	The number of Required Executions
One-by-one strategy	**4**
Our influence-guided strategy	**2**

Reduce the program execution by 50% compared to one-by-one call removal.

Type-informed Argument Simplification Strategy

Insight:

Only need to simplify the **redundant arguments** or their sub-fields that **truly matter for subsequent mutations**

Type-informed Argument Simplification Strategy

Insight:

Only need to simplify the **redundant arguments** or their sub-fields that **truly matter for subsequent mutations**

Call-minimized program

```
r1=openat$dsp(a1)
read$dsp(r1,[a4])
```

Type-informed Argument Simplification Strategy

Insight:

Only need to simplify the **redundant arguments** or their sub-fields that **truly matter for subsequent mutations**

Call-minimized program

```
r1=openat$dsp(a1)
read$dsp(r1,[a4])
```

a_1 is integer type, r_1 is resource type, a_4 is an element of array type

Type-informed Argument Simplification Strategy

Insight:

Only need to simplify the **redundant arguments** or their sub-fields that **truly matter for subsequent mutations**

Call-minimized program

```
r1=openat$dsp(a1)
read$dsp(r1,[a4])
```

Skip simplification of integer a_1 and resource argument r_1, as simplify them does not reduce subsequent mutation space

Type-informed Argument Simplification Strategy

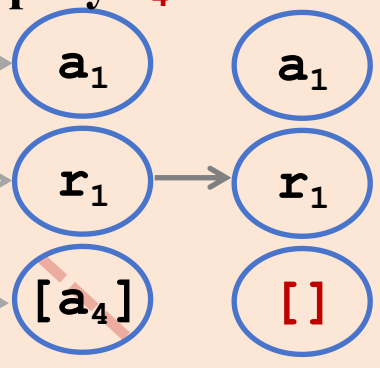
Insight:

Only need to simplify the **redundant arguments** or their sub-fields that **truly matter for subsequent mutations**

Simplify a_4 and execute to verify c_5's coverage is retained

Call-minimized program

```
r1=openat$dsp(a1)
read$dsp(r1,[a4])
```

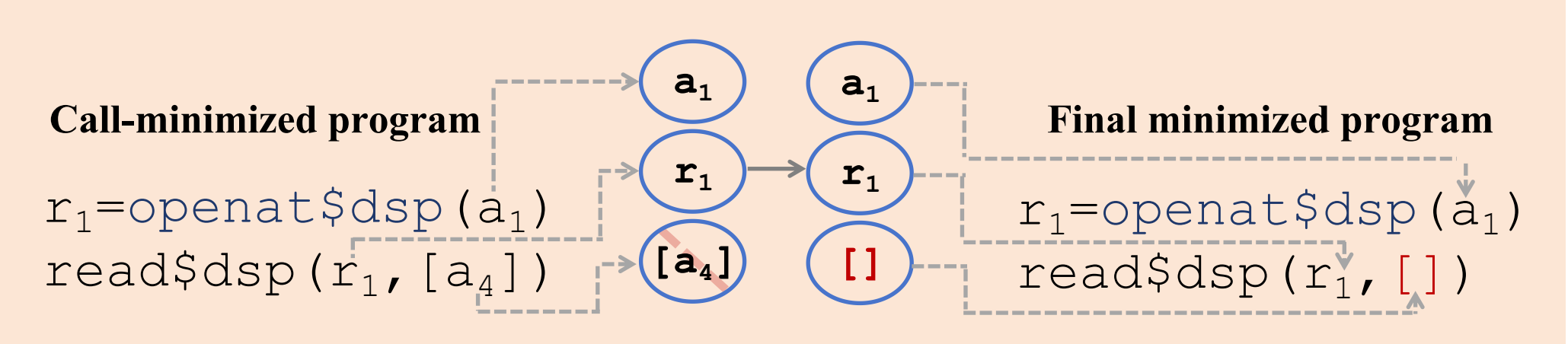


Skip simplification of integer a_1 and resource argument r_1, as simplify them does not reduce subsequent mutation space

Type-informed Argument Simplification Strategy

Insight:

Only need to simplify the **redundant arguments** or their sub-fields that **truly matter for subsequent mutations**



Type-informed argument simplification strategy only consume one execution

Type-informed Argument Simplification Strategy

Insight:

Only need to simplify the **redundant arguments** or their sub-fields that **truly matter for subsequent mutations**

Strategy	The number of Required Executions
One-by-one strategy	3
Our type-informed strategy	1

Reduce the program execution by 66% compared to one-by-one argument simplification.

Our Two Optimization Strategy vs One-by-one Strategy

Insight of influence-guided call removal strategy:

Irrelevant calls can be removed at once rather than one by one, as they does not affect new achieved coverage

Insight of arg-informed argument simplification strategy:

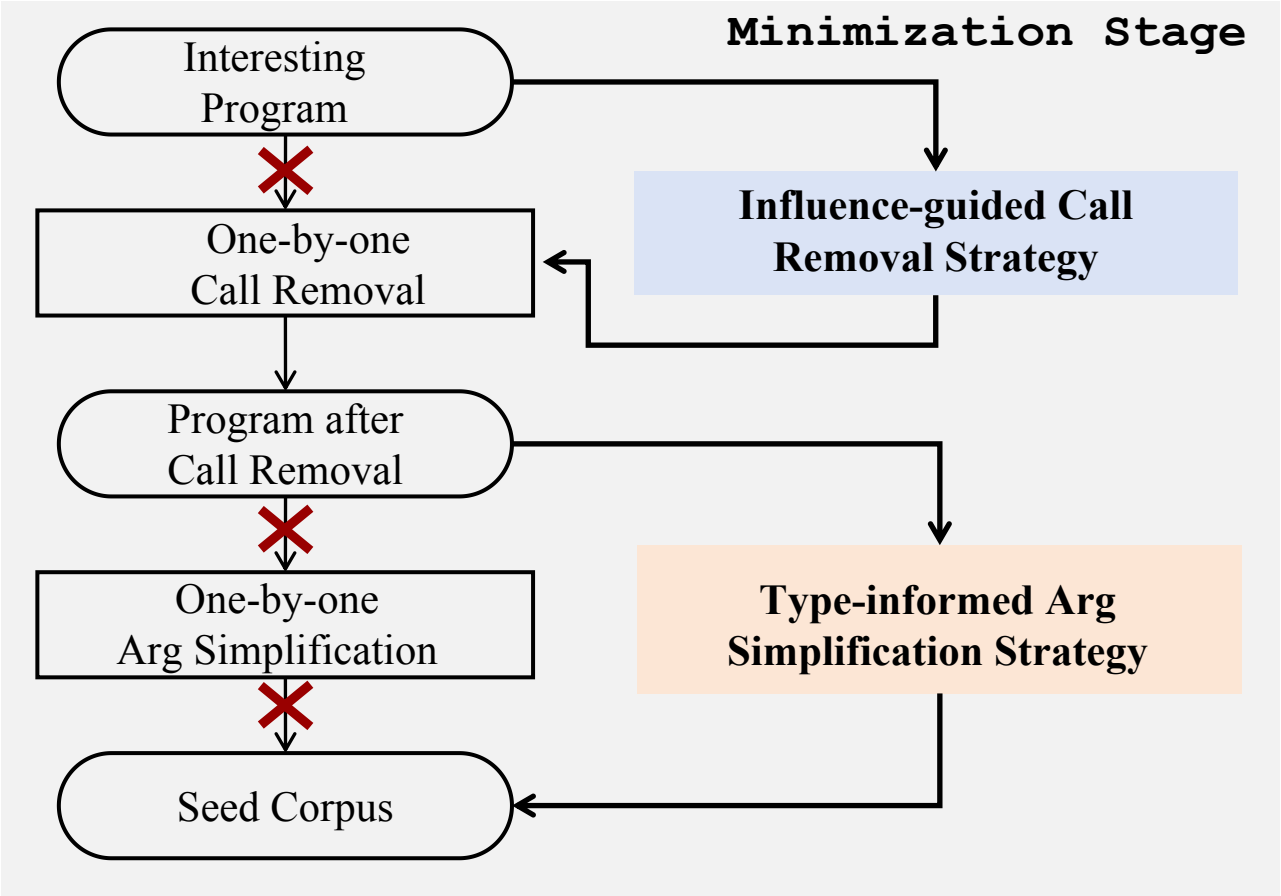
Only need to simplify the redundant arguments or their sub-fields that **truly matter for** subsequent mutations

Strategy	The number of Required Executions
One-by-one strategy	**7**
Our two strategy	**3**

Reduce the number of program execution by 57.1% compared to one-by-one strategy

Our Implementation: SyzMini

- The two strategies was implemented in Syzkaller



Our Implementation: SyzMini

- The two strategies was implemented in Syzkaller

- Collect influence relation to infer irrelevant call

  Influence Relation: defined as the effect of two calls on execution path

  - Collect 44,966 static influence relations by static analysis
  - Collect 29,899 dynamic influence relations by dynamic analysis

Our Implementation: SyzMini

- The two strategies was implemented in Syzkaller

- Collect influence relation to infer irrelevant call

  Influence Relation: defined as the effect of two calls on execution path

  - Collect 44,966 static influence relations by static analysis
  - Collect 29,899 dynamic influence relations by dynamic analysis

- Use the default branch coverage to record the execution information of each call

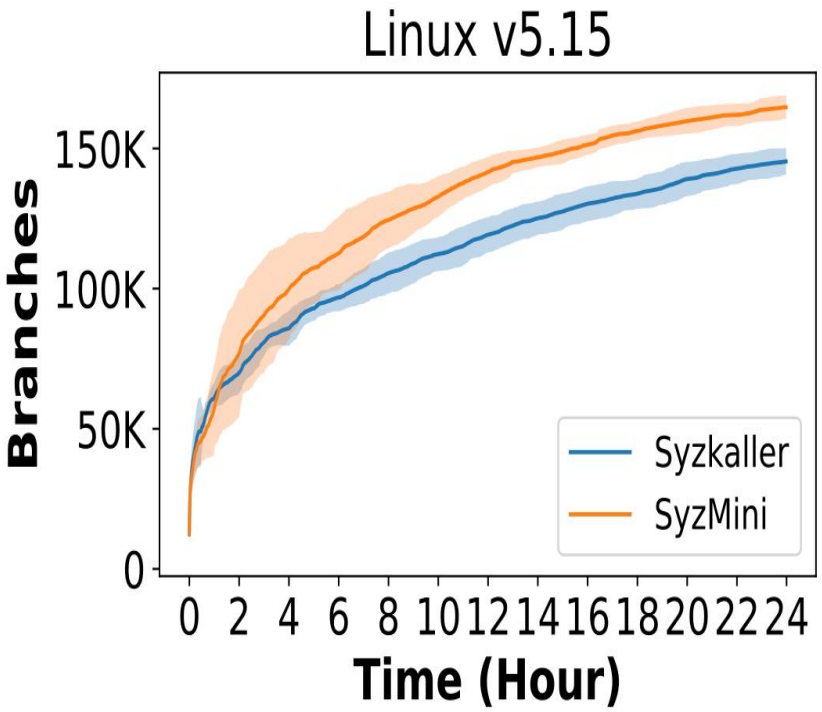
Evaluation

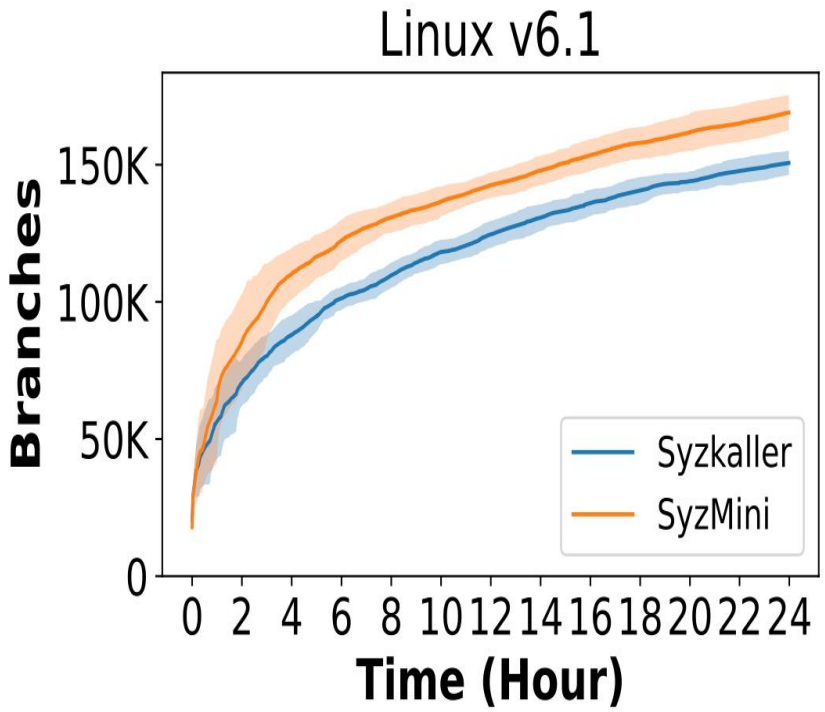
- **RQ1: Effectiveness of SyzMini**
  - Achieved code coverage
  - Number of found bugs
  - Hitting-round and Time-to-exposure (TTE)

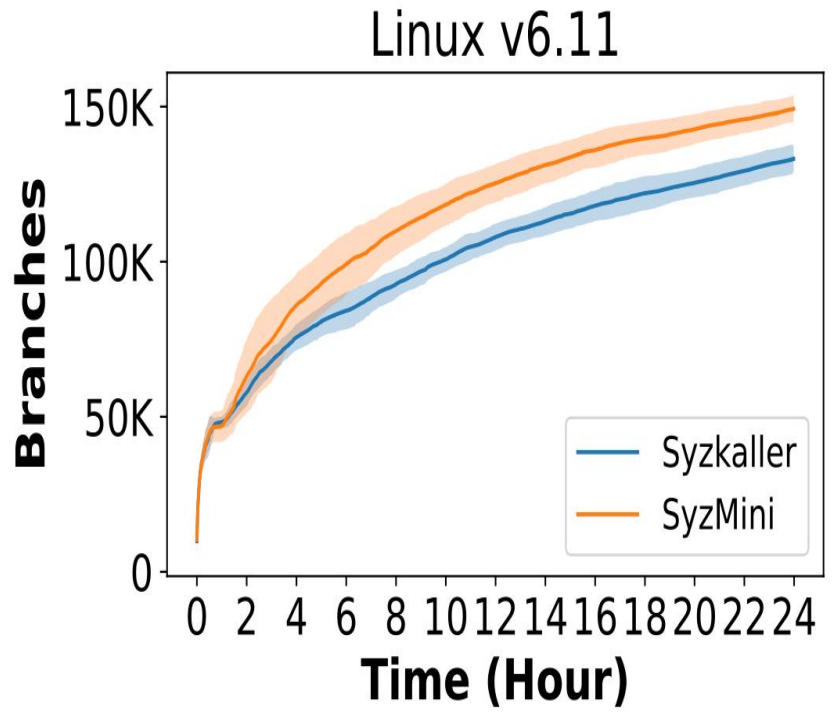
Setup of RQ1

- Compare SyzMini to Syzkaller on Linux v5.15, v6.1 and v6.11
- Each tool run 24-hours and repeates 10 rounds to avoid randomness
- Count achieved code coverage and the number of unique bugs
- Measure hitting-round and time-to-exposure (TTE) for each bug

RQ1 Result: Code Coverage

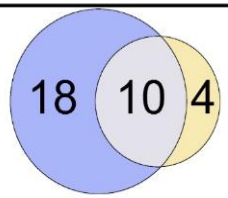
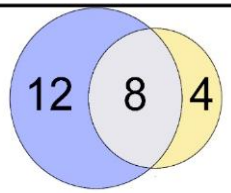
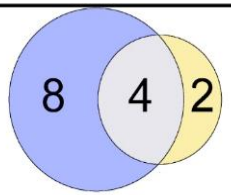
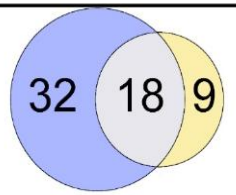






SyzMini achieve 13.3%, 12.2% and 12.1% coverage improvement on three kernel versions

RQ1 Result: Number of Found Unique Bugs

Fuzzer	#Unique Bugs				Bug Distribution			
	v5.15	v6.1	v6.11	Total	v5.15	v6.1	v6.11	Total
SYZKALLER	14	12	6	27				
SYZMINI	28	20	12	50				

[1] The blue and yellow regions in the venn diagrams denote #bugs found by SyzMini and Syzkaller, respectively; the gray regions denote #bugs found by both tools.

SyzMini found 2.0X, 1.7X and 2.0X unique bugs than Syzkaller on three kernel versions

RQ1 Result: Hitting-round and uTTE of Each Common Bug

SK → Syzkaller
SM →SyzMini

ID	*hitting-round* SK/SM	*μTTE* (h) SK/SM	ID	*hitting-round* SK/SM	*μTTE* (h) SK/SM
1	3 / 6	22.2 / 12.2	12	3 / 5	16.8 / 14.0
2	1 / 2	23.9 / 19.7	13	7 / 10	12.5 / 3.5
3	4 / 8	15.8 / 10.8	14	1 / 2	23.5 / 20.9
4	1 / 3	23.1 / 20.9	15	2 / 4	19.4 / 15.9
5	5 / 5	17.0 / 13.8	16	1 / 4	22.3 / 18.9
6	7 / 9	19.4 / 15.8	17	2 / 8	22.4 / 13.1
7	3 / 4	22.8 / 19.4	18	1 / 6	23.4 / 13.3
8	3 / 3	21.7 / 20.7	19	1 / 3	22.3 / 19.5
9	1 / 2	22.3 / 20.1	20	10 / 10	4.2 / 2.0
10	1 / 2	22.0 / 20.8	21	1 / 3	23.9 / 19.4
11	9 / 9	11.9 / 7.5	22	8 / 10	5.2 / 2.8

RQ1 Result: Hitting-round and uTTE of Each Common Bug

SK → Syzkaller
SM →SyzMini

ID	*hitting-round* SK/SM	*μTTE* (h) SK/SM	ID	*hitting-round* SK/SM	*μTTE* (h) SK/SM
1	3 / 6	22.2 / 12.2	12	3 / 5	16.8 / 14.0
2	1 / 2	23.9 / 19.7	13	7 / 10	12.5 / 3.5
3	4 / 8	15.8 / 10.8	14	1 / 2	23.5 / 20.9
4	1 / 3	23.1 / 20.9	15	2 / 4	19.4 / 15.9
5	5 / 5	17.0 / 13.8	16	1 / 4	22.3 / 18.9
6	7 / 9	19.4 / 15.8	17	2 / 8	22.4 / 13.1
7	3 / 4	22.8 / 19.4	18	1 / 6	23.4 / 13.3
8	3 / 3	21.7 / 20.7	19	1 / 3	22.3 / 19.5
9	1 / 2	22.3 / 20.1	20	10 / 10	4.2 / 2.0
10	1 / 2	22.0 / 20.8	21	1 / 3	23.9 / 19.4
11	9 / 9	11.9 / 7.5	22	8 / 10	5.2 / 2.8

Among 22 common bugs, SyzMini found them faster and more frequently than Syzkaller

Evaluation

- **RQ1: Effectiveness of SyzMini**
  - Achieved code coverage
  - Number of found bugs
  - Hitting-round and Time-to-expose (TTE)

- **RQ2: Applicability of our two optimization strategies for different kernel fuzzers**
  - SyzVegas (Usenix security 21)

    A tool that utilizes reinforcement learning to improve kernel fuzzing effectiveness
  - CountDown (CCS 24)

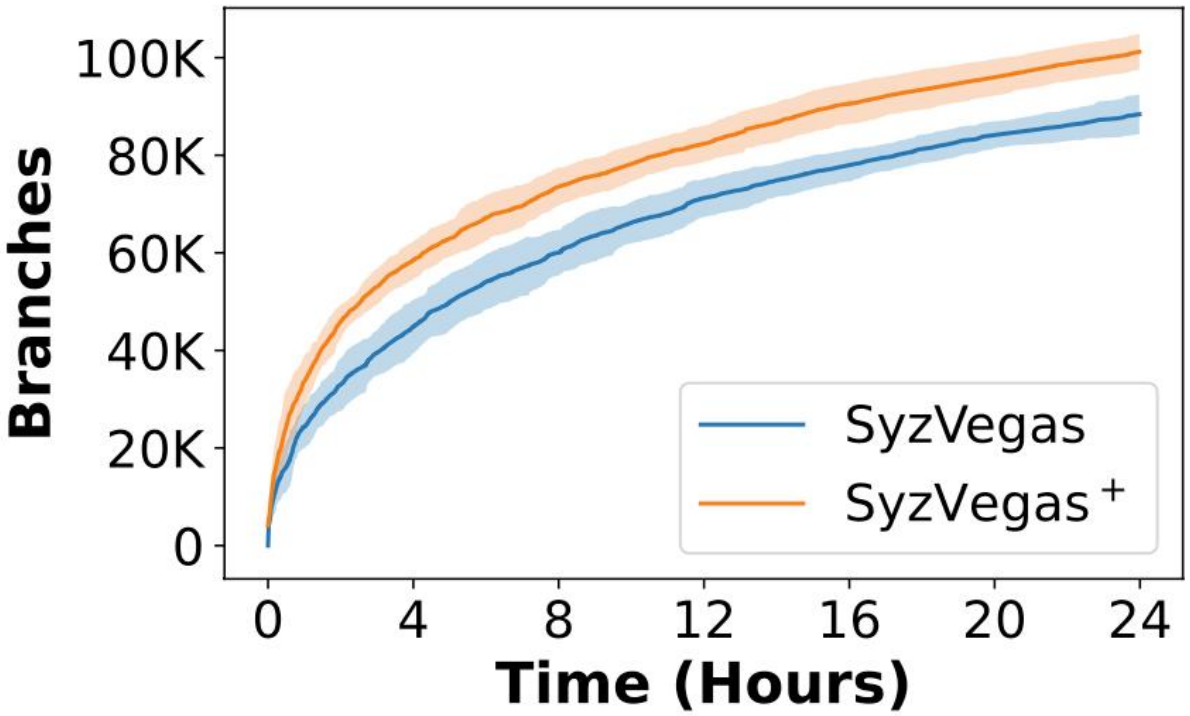
    A tool to uncover memory-related bugs
  - SyzDirect (CCS 23)

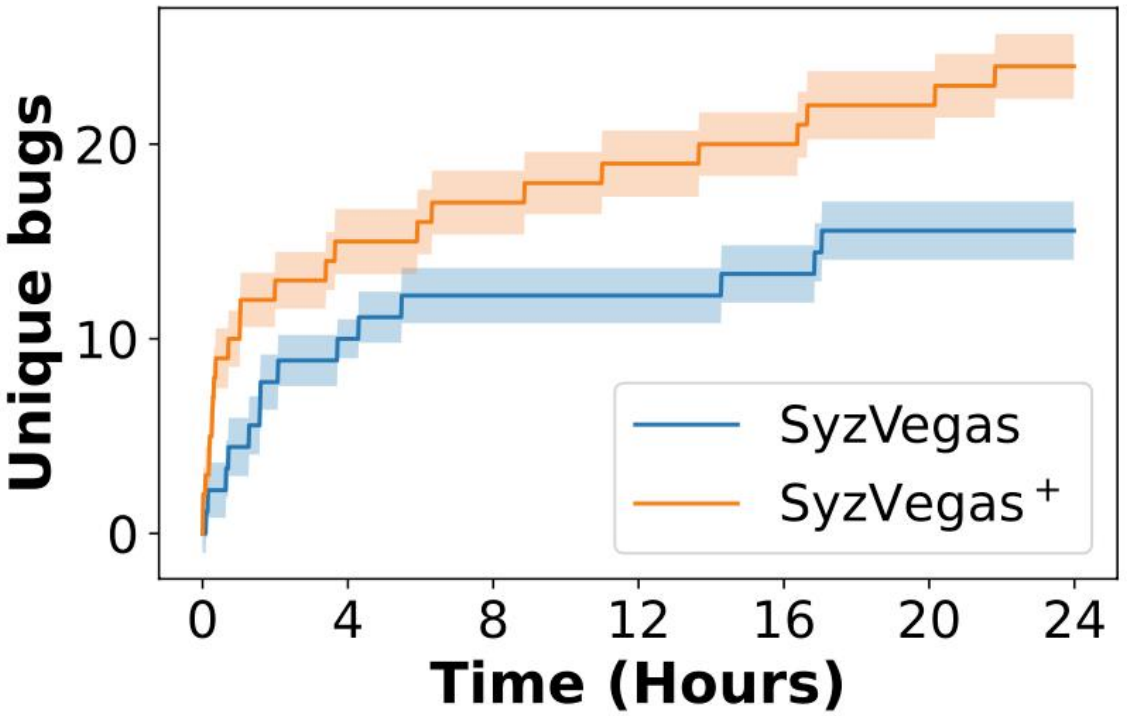
    A directed kernel fuzzing tool

Setup of RQ2

- Implement our strategies in SyzVegas, CountDown and SyzDirect, called SyzVegas$^+$, CountDown$^+$ and SyzDirect$^+$.
- Compare SyzVegas with SyzVegas$^+$ in term of the code coverage and the number unique bugs
- Compare CountDown with CountDown$^+$ in term of the number of memory-related bugs
- Compare SyzDirect with SyzDirect$^+$ in term of μTTE and hitting-round for reproducing each target bug

RQ2 Result: SyzVegas vs SyzVegas+





SyzVegas+ improves branch coverage by 14.5% and found 2.0X unique bugs on average.

RQ2 Result: CountDown vs CountDown+

Tool	All KASAN Bugs			Branch Coverage		
	Min	Max	Total	Min	Max	Avg
CountDown	0	5	27	116.9k	120.7k	119.0k
CountDown+	1	7	43	119.7k	130.1k	124.3k
Improvement	1↑	2↑	66.7%↑	2.4%↑	7.8%↑	4.5%↑

CountDown+ achieves 66.7% improvement in term of KASAN bugs.

RQ2 Result: SyzDirect vs SyzDirect+

Bug ID	Commit ID	SyzDirect *μTTE~hitting-round*	SyzDirect+ *μTTE~hitting-round*
1	3f2db250099f	×	16.0h~4/10
2	c9a2f90f4d6b	21.6h~2/10	20.0h~4/10
3	b1a811633f73	×	21.2h~1/10
4	2fd10bcf0310	21.2h~2/10	10.1h~8/10
5	b648eba4c69e	×	×
6	20aaef52eb08	13.6h~10/10	12.8h~10/10
7	3b0c40612471	18.8h~4/10	17.6h~6/10
8	ffb324e6f874	×	22.1h~1/10
9	01faae5193d6	18.7h~4/10	14.4h~6/10
10	b2a616676839	20.4h~6/10	14.10h~8/10

SyzDirect+ reproduced bugs with less time than SyzDirect.

RQ2 Result: SyzDirect vs SyzDirect+

Bug ID	Commit ID	SyzDirect *μTTE~hitting-round*	SyzDirect+ *μTTE~hitting-round*
1	3f2db250099f	×	16.0h~4/10
2	c9a2f90f4d6b	21.6h~2/10	20.0h~4/10
3	b1a811633f73	×	21.2h~1/10
4	2fd10bcf0310	21.2h~2/10	10.1h~8/10
5	b648eba4c69e	×	×
6	20aaef52eb08	13.6h~10/10	12.8h~10/10
7	3b0c40612471	18.8h~4/10	17.6h~6/10
8	ffb324e6f874	×	22.1h~1/10
9	01faae5193d6	18.7h~4/10	14.4h~6/10
10	b2a616676839	20.4h~6/10	14.10h~8/10

SyzDirect+ reproduced 1.5X bugs than SyzDirect, and find them more frequently.

Summary

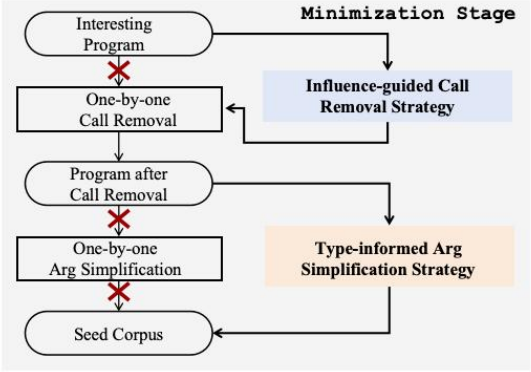
Tension: Benefit and Cost of Minimization

Benefit: Improve the effectiveness of kernel fuzzing

Cost: Consume over half of the fuzzing resources

Our Implementation: SyzMini

- The two strategies was implemented in Syzkaller



Our Two Optimization Strategy vs One-by-one Strategy

Insight of influence-guided call removal strategy:
Irrelevant calls can be removed at once rather than one by one, as they does not affect new achieved coverage

Insight of arg-informed argument simplification strategy:
Only need to simplify the redundant arguments or their sub-fields that **truly matter for** subsequent mutations

Strategy	The number of Required Executions
One-by-one strategy	7
Our two strategy	3

Reduce the number of program execution by 57.1% compared to one-by-one strategy

- Our tool SyzMini has been open-sourced at *https://github.com/ecnusse/SyzMini/*
- The Syzkaller team has already implemented our optimization strategies in its latest version
- Email: guohui.study@gmail.com