SWE3002-42: Introduction to Software Engineering

Lecture 10 – Automatic Program Repair

Sooyoung Cha

Department of Computer Science and Engineering

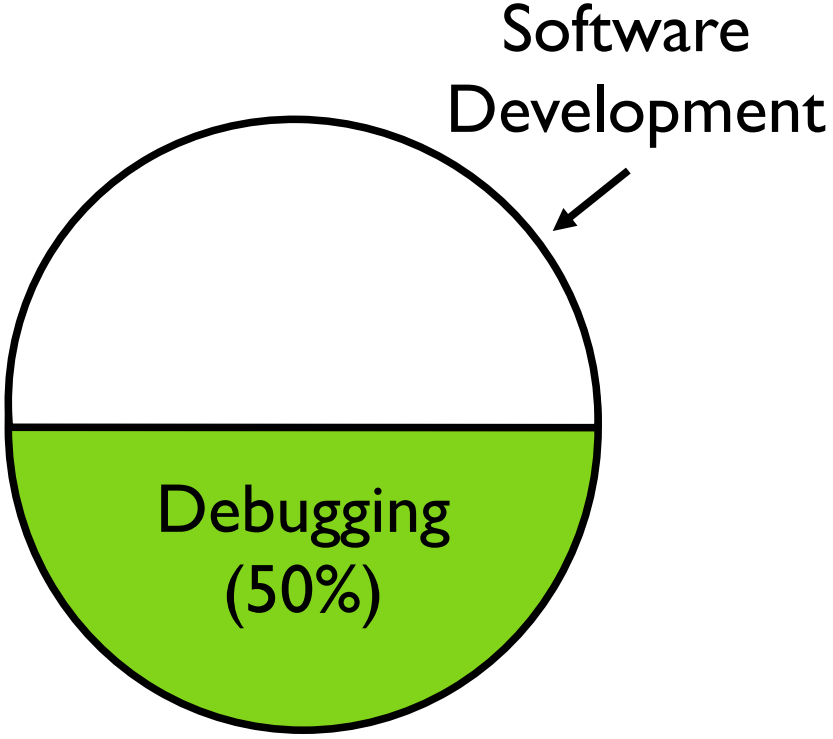
Today's Lecture

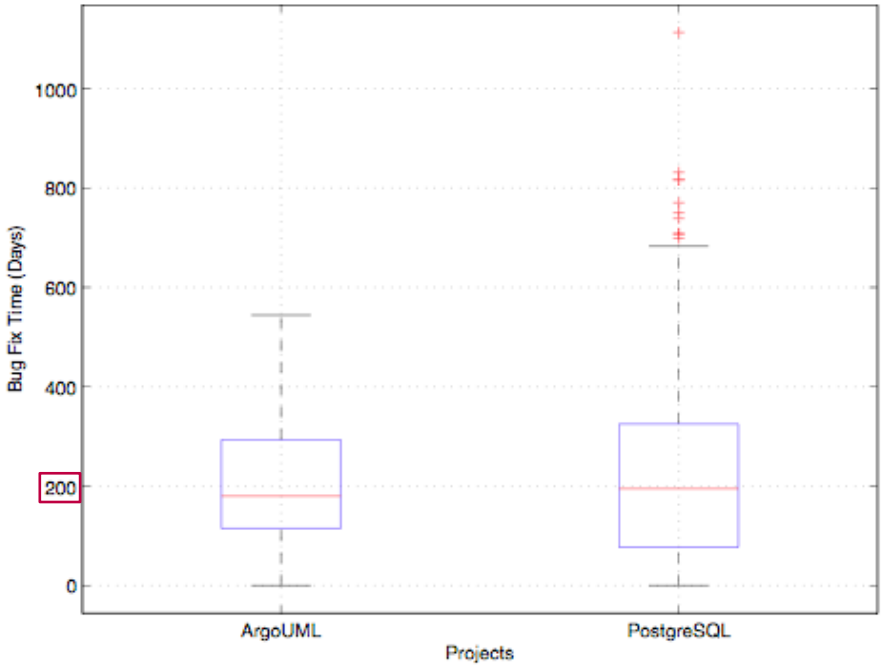
- Motivation:
  - Why do we need automatic program repair techniques?
- Automatic Program Repair (APR)
  - General-Purpose Program Repair
  - Special-Purpose Program Repair
    - A state-of-the art technique for automatically fixing memory deallocation errors.

      (MemFix(FSE'18), LeakFix(ICSE'15), Footpatch(ICSE'18), SAVER(ICSE'20))

Why do we need APR ?

- Debugging takes up half of the time in SW development.
  - Bug-fix time: 200 days[1] (for commercial software)

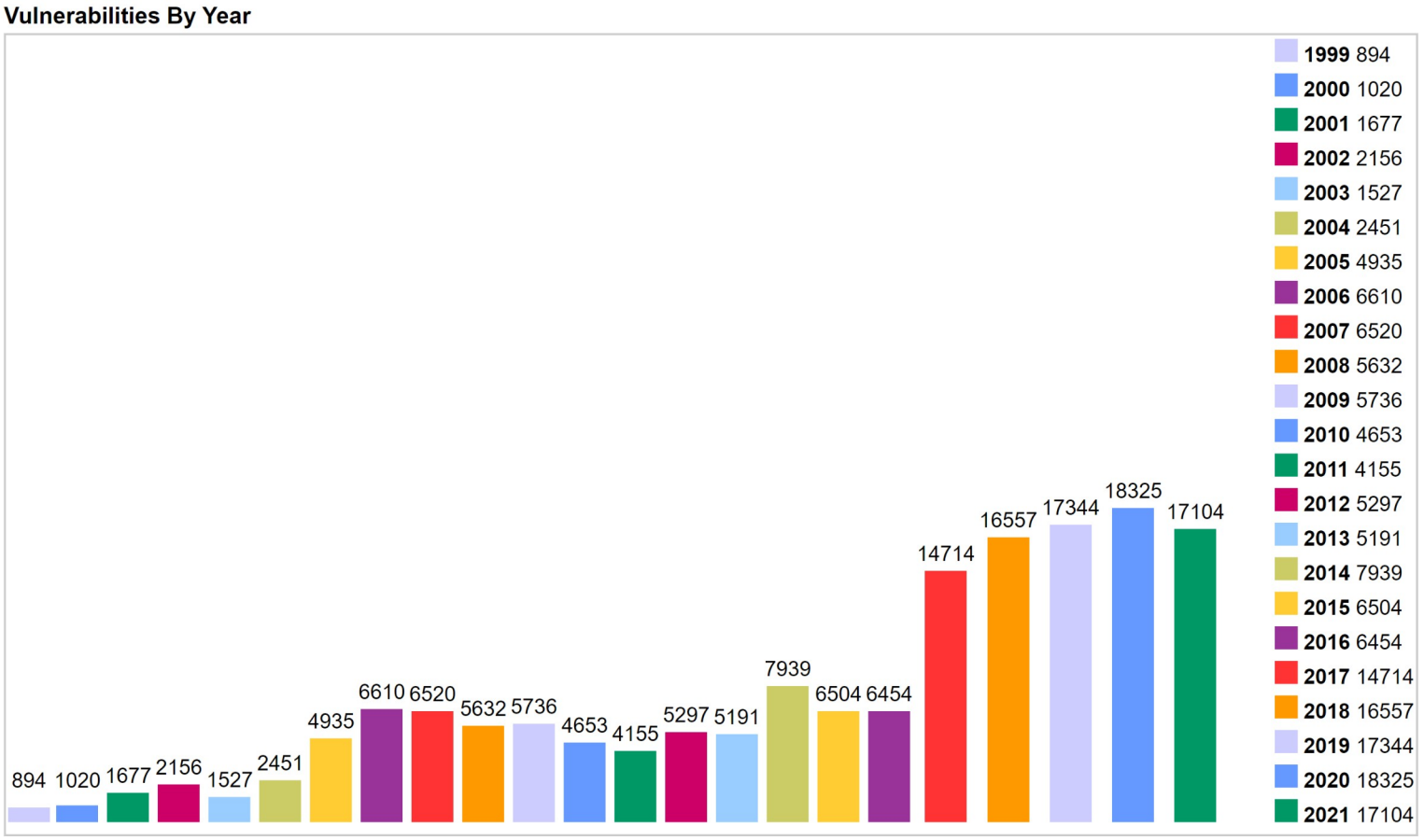




1. How Long Did It Take To Fix Bugs?

Why do we need APR ?

- The number of software bugs increases every year.
  - # of CVE[1]: 4,600 ('2010) → 6,500 ('2015) → 18,000 ('2020)



1. https://www.cvedetails.com/browse-by-date.php

Why do we need APR ?

- Manually fixing bugs is time-consuming and error-prone.
  - Hard to be sure whether the bug is fixed.
  - Introduce a new bug during the process of fixing.

Why do we need APR ?

- Manually fixing bugs is time-consuming and error-prone.
  - Hard to be sure whether the bug is fixed.
  - Introduce a new bug during the process of fixing.

```
in = malloc(1);
out = malloc(1);
... // use in, out
free(out);
free(in);

in = malloc(2);
if (in == NULL) {

  goto err;
}

out = malloc(2);
if (out == NULL) {
  free(in);

  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

Real bug from Linux Kernel[1]
(3 patches, 1 year, 3 developers)

```
in = malloc(1);
out = malloc(1);
... // use in, out
//-
free(in);

in = malloc(2);
if (in == NULL) {
   out = NULL;
  goto err;
}
free(out);
out = malloc(2);
if (out == NULL) {
  free(in);
  in = NULL;
  goto err;
}
... // use in, out
err:
  free(in);
  free(out);
  return;
```

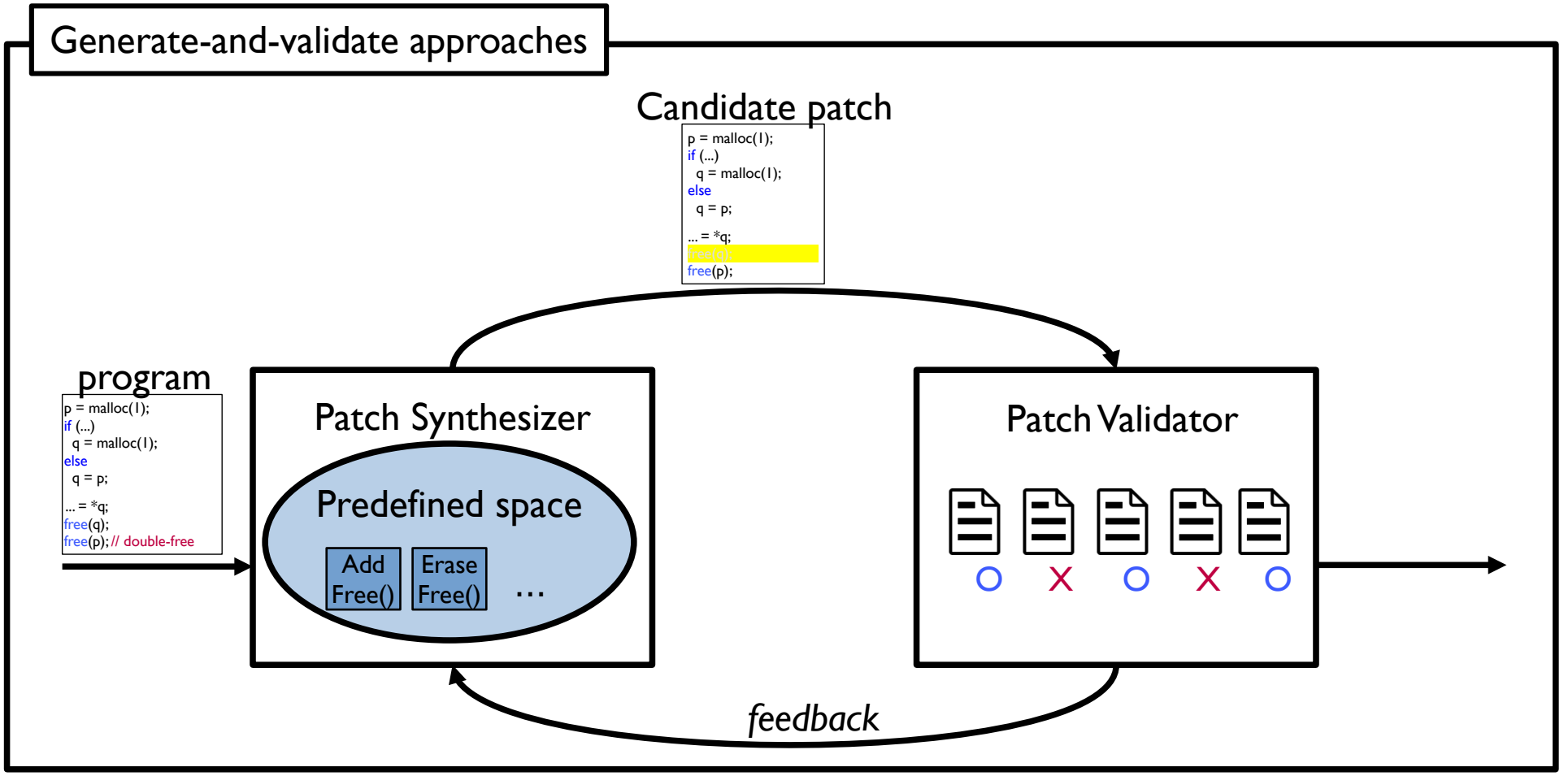
1. https://github.com/torvalds/linux/commit/df3e1ab

Automatic Program Repair

- Can be classified into two APR techniques.
  - General-purpose program repair
    - (+) fix any kinds of errors.
    - (-) Low performance for fixing a specific error.
  - Special-purpose program repair
    - (+) High performance for fixing a specific error.
    - (-) fix specific kinds of errors.

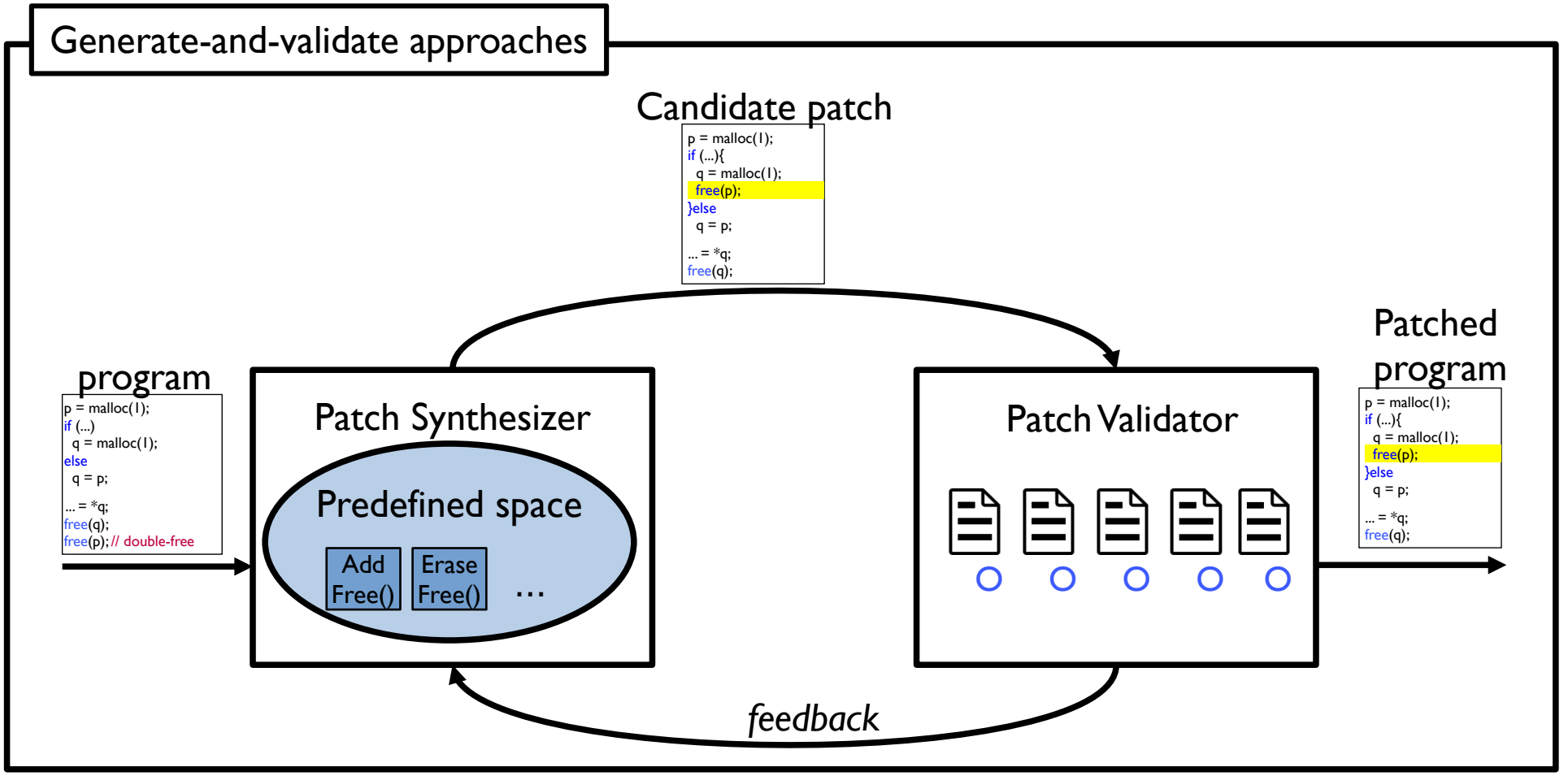
Automatic Program Repair

- General-purpose program repair techniques
  - Usually rely on test cases to validate patches



Automatic Program Repair

- General-purpose program repair techniques
  - Usually rely on test cases to validate patches



Automatic Program Repair

- Special-purpose program repair techniques
  - Focus on fixing specific classes of errors.
    - Memory deallocation errors: MemFix[1], SAVER[2], LeakFix[3]
    - Null dereferences: Vfix[4], SapFix[5]
    - Buffer/integer overflows: IntPTI[6], IntRepair[7]
    - Concurrency errors: Pfix[8]

      ...

1. MemFix: Static Analysisbased Repair of Memory Deallocation Errors for C.
2. SAVER: Scalable, Precise, and Safe Memory-Error Repair
3. Safe Memory-leak Fixing for C Programs
4. VFix: Value-flowguided Precise Program Repair for Null Pointer Dereferences.
5. SapFix: Automated End-to-end Repair at Scale.
6. IntPTI: Automatic Integer Error Repair with Proper-type Inference
7. IntRepair: Informed Repairing of Integer Overflows
8. PFix: Fixing Concurrency Bugs Based on Memory Access Patterns.

Memory Deallocation Errors

- Memory-leak (CWE-401): Free memory too late.
- Use-after-free (CWE-416): Free memory too quickly.
- Double-free (CWE-415): Free the memory multiple times.

Memory-leak

```
p = malloc(1);
…
return;
```

Use-after-free

```
p = malloc(1);
…
free(p);
...
use(p);
```

Double-free

```
p = malloc(1);
…
free(p);
...
free(p);
```

Memory Deallocation Errors

- One of the most troubling errors in open-source programs[1]

Repo.	#commits	ML	DF	UAF	Total	*-overflow
linux	721,119	3,740	821	1,986	**6,363**	5,092
php	105,613	1,129	148	197	**1,449**	649
git	49,475	350	19	95	**442**	258
openssl	21,009	220	36	12	**264**	61

CVE-2021-36145	The Device Model in ACRN through 2.5 has a devicemodel/core/mem.c use-after-free for a fre
CVE-2021-36144	The polling timer handler in ACRN before 2.5 has a use-after-free for a freed virtio device, rela
CVE-2021-36088	Fluent Bit (aka fluent-bit) 1.7.0 through 1.7,4 has a double free in flb_free (called from flb_pa
CVE-2021-36086	The CIL compiler in SELinux 3.2 has a use-after-free in cil_reset_classpermission (called from
CVE-2021-36085	The CIL compiler in SELinux 3.2 has a use-after-free in __cil_verify_classperms (called from _
CVE-2021-36084	The CIL compiler in SELinux 3.2 has a use-after-free in __cil_verify_classperms (called from _
CVE-2021-36081	Tesseract OCR 5.0.0-alpha-20201231 has a one_ell_conflict use-after-free during a strpbrk ca
CVE-2021-36080	GNU LibreDWG 0.12.3.4163 through 0.12.3.4191 has a double-free in bit_chain_free (called fi
CVE-2021-36055	XMP Toolkit SDK versions 2020.1 (and earlier) are affected by a use-after-free vulnerability th
CVE-2021-36008	Adobe Illustrator version 25.2.3 (and earlier) is affected by an Use-after-free vulnerability whe user interaction in that a victim must open a malicious file.

1. MemFix: Static Analysisbased Repair of Memory Deallocation Errors for C.

Memory Deallocation Errors

- What memory deallocation errors exist in this toy code?
- Can you explain exactly how the error is occurring?
- How can we fix that error?

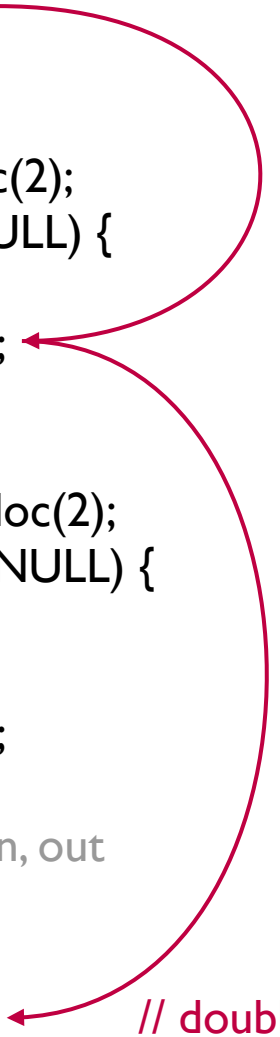
```
1    p = malloc(1);
2    if(…) {
3      q = malloc(2);
4
5    }
6    else
7      q = p;
8    … // use q
9    free(p);
10   free(q);
```

Real Errors from Linux Kernel

```
1   out = malloc(1);
2   in = malloc(1);
3   … // use in, out
4   free(out);
5   free(in);
6
7   in = malloc(2);
8   if(in == NULL) {
9
10    goto err;
11  }
12
13  out = malloc(2);
14  if(out == NULL) {
15    free(in);
16
17    goto err;
18  }
19  … // use in, out
20 err:
21   free(in);
22   free(out);
```

Real Errors from Linux Kernel

```
1   out = malloc(1);
2   in = malloc(1);
3   … // use in, out
4   free(out);
5   free(in);
6
7   in = malloc(2);
8   if(in == NULL) {
9
10    goto err;
11  }
12
13  out = malloc(2);
14  if(out == NULL) {
15    free(in);
16
17    goto err;
18  }
19  … // use in, out
20 err:
21   free(in);
22   free(out);        // double-free
```



Real Errors from Linux Kernel

```
1   out = malloc(1);
2   in = malloc(1);
3   … // use in, out
4   free(out);
5   free(in);
6
7   in = malloc(2);
8   if(in == NULL) {
9
10    goto err;
11  }
12
13  out = malloc(2);
14  if(out == NULL) {
15    free(in);
16
17    goto err;
18  }
19  … // use in, out
20 err:
21   free(in);          // double-free
22   free(out);
```

Real Errors from Linux Kernel

```
1   out = malloc(1);
2   in = malloc(1);
3   … // use in, out
4   free(out);
5   free(in);
6
7   in = malloc(2);
8   if(in == NULL) {
9     out = NULL;
10    goto err;
11  }
12
13  out = malloc(2);
14  if(out == NULL) {
15    free(in);
16    in = NULL;
17    goto err;
18  }
19  … // use in, out
20 err:
21   free(in);
22   free(out);
```

1. First Patch

USB: fix double frees in error code paths of ipaq driver

```
the error code paths can be enter with buffers to freed buffers.
Serial core would do a kfree() on memory already freed.

Signed-off-by: Oliver Neukum <oneukum@suse.de>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>
```

master v4.15-rc1 ... v2.6.24-rc1

Oliver Neukum committed with **gregkh** on 18 Sep 2007 1 par

Real Errors from Linux Kernel

```
1   out = malloc(1);
2   in = malloc(1);
3   … // use in, out
4   free(out);
5   free(in);
6
7   in = malloc(2);
8   if(in == NULL) {
9     out = NULL;
10    goto err;
11  }
12
13  out = malloc(2);
14  if(out == NULL) {
15    free(in);
16    in = NULL;
17    goto err;
18  }
19  … // use in, out
20 err:
21   free(in);          // safe
22   free(out);
```

1. First Patch

USB: fix double frees in error code paths of ipaq driver

```
the error code paths can be enter with buffers to freed buffers.
Serial core would do a kfree() on memory already freed.

Signed-off-by: Oliver Neukum <oneukum@suse.de>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>
```

master v4.15-rc1 … v2.6.24-rc1

Oliver Neukum committed with **gregkh** on 18 Sep 2007 1 par

Real Errors from Linux Kernel

```
1   out = malloc(1);
2   in = malloc(1);
3   … // use in, out
4   free(out);
5   free(in);
6
7   in = malloc(2);
8   if(in == NULL) {
9     out = NULL;
10    goto err;
11  }
12
13  out = malloc(2);
14  if(out == NULL) {
15    free(in);
16    in = NULL;
17    goto err;
18  }
19  … // use in, out
20 err:
21   free(in);          // safe
22   free(out);
```

1. First Patch

USB: fix double frees in error code paths of ipaq driver

```
the error code paths can be enter with buffers to freed buffers.
Serial core would do a kfree() on memory already freed.

Signed-off-by: Oliver Neukum <oneukum@suse.de>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>
```

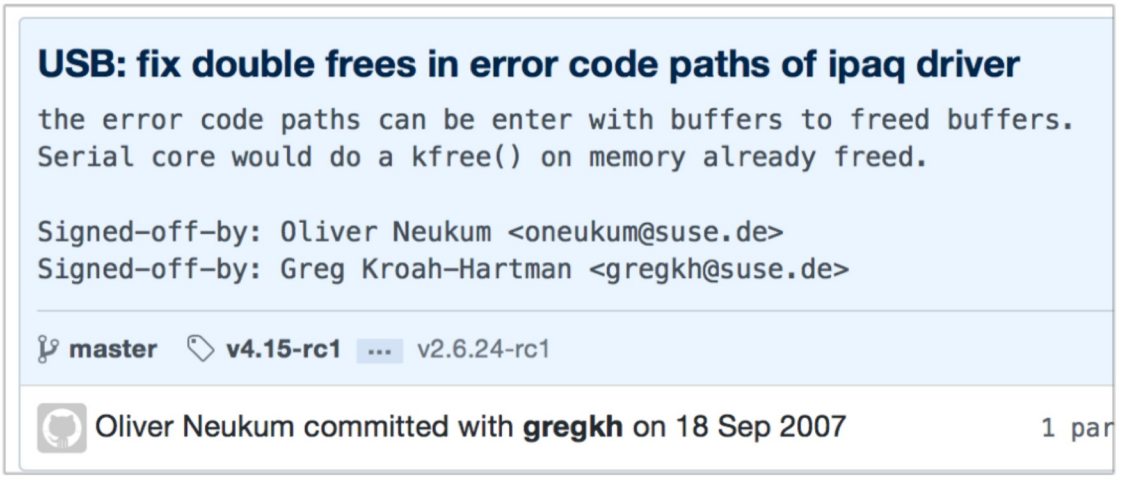
master v4.15-rc1 … v2.6.24-rc1

Oliver Neukum committed with **gregkh** on 18 Sep 2007 1 par

Real Errors from Linux Kernel

```
1   out = malloc(1);
2   in = malloc(1);
3   … // use in, out
4   free(out);
5   free(in);
6
7   in = malloc(2);
8   if(in == NULL) {
9     out = NULL;
10    goto err;
11  }
12
13  out = malloc(2);
14  if(out == NULL) {
15    free(in);
16    in = NULL;
17    goto err;
18  }
19  … // use in, out
20 err:
21   free(in);        // safe
22   free(out);
```

1. First Patch



The first problem of manual debugging:
- Hard to be sure if the error has been fixed

Real Errors from Linux Kernel

```
1   out = malloc(1);
2   in = malloc(1);
3   … // use in, out
4   free(out);
5   free(in);
6
7   in = malloc(2);
8   if(in == NULL) {
9     out = NULL;
10    goto err;
11  }
12  free(out);
13  out = malloc(2);
14  if(out == NULL) {
15    free(in);
16    in = NULL;
17    goto err;
18  }
19  … // use in, out
20 err:
21   free(in);
22   free(out);
```

2. Second Patch

USB: fix double kfree in ipaq in error case

```
in the error case the ipaq driver leaves a dangling pointer to already
freed memory that will be freed again.

Signed-off-by: Oliver Neukum <oneukum@suse.de>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>
```

master v4.19-rc4 … v2.6.27-rc1

Oliver Neukum authored and **gregkh** committed on Jun 30, 2008

Real Errors from Linux Kernel

```
1   out = malloc(1);
2   in = malloc(1);
3   … // use in, out
4   free(out);
5   free(in);
6
7   in = malloc(2);
8   if(in == NULL) {
9     out = NULL;      // memory-leak
10    goto err;
11  }
12  free(out);
13  out = malloc(2);
14  if(out == NULL) {
15    free(in);
16    in = NULL;
17    goto err;
18  }
19  … // use in, out
20 err:
21   free(in);
22   free(out);
```

2. Second Patch

USB: fix double kfree in ipaq in error case

```
in the error case the ipaq driver leaves a dangling pointer to already
freed memory that will be freed again.

Signed-off-by: Oliver Neukum <oneukum@suse.de>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>
```

master v4.19-rc4 … v2.6.27-rc1

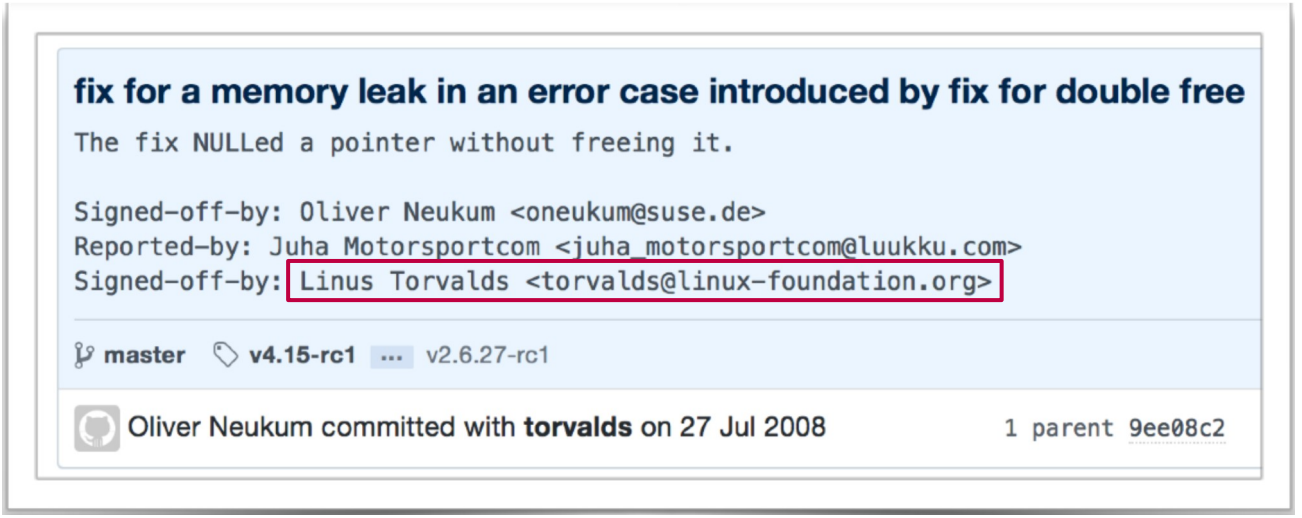
Oliver Neukum authored and **gregkh** committed on Jun 30, 2008

The second problem of manual debugging:
- new error is introduced during the fix process

Real Errors from Linux Kernel

```
1   out = malloc(1);
2   in = malloc(1);
3   … // use in, out
4   free(out);
5   free(in);
6   out = NULL;
7   in = malloc(2);
8   if(in == NULL) {
9     out = NULL;
10    goto err
11  }
12  free(out);
13  out = malloc(2);
14  if(out == NULL) {
15    free(in);
16    in = NULL;
17    goto err;
18  }
19  … // use in, out
20 err:
21   free(in);
22   free(out);
```

3. Third Patch



- The program was fixed during about one year.
- A total of 3 patches commited.
- The program was reviewed by 3 other developers.

MemFix

- Automatically repair memory deallocation errors
  - memory-leak, double-free and use-after-free
- Key features
  - Generated patch is guaranteed to be correct.
  - No new errors are introduced.
- Key Ideas
  - Static Analysis + Exact Cover Problem

FSE'18

MemFix

```
1   out = malloc(1);
2   in = malloc(1);
3   … // use in, out
4   free(out);
5   free(in);
6
7   in = malloc(2);
8   if(in == NULL) {
9
10    goto err;
11  }
12
13  out = malloc(2);
14  if(out == NULL) {
15    free(in);
16
17    goto err;
18  }
19  … // use in, out
20 err:
21   free(in);
22   free(out);
```

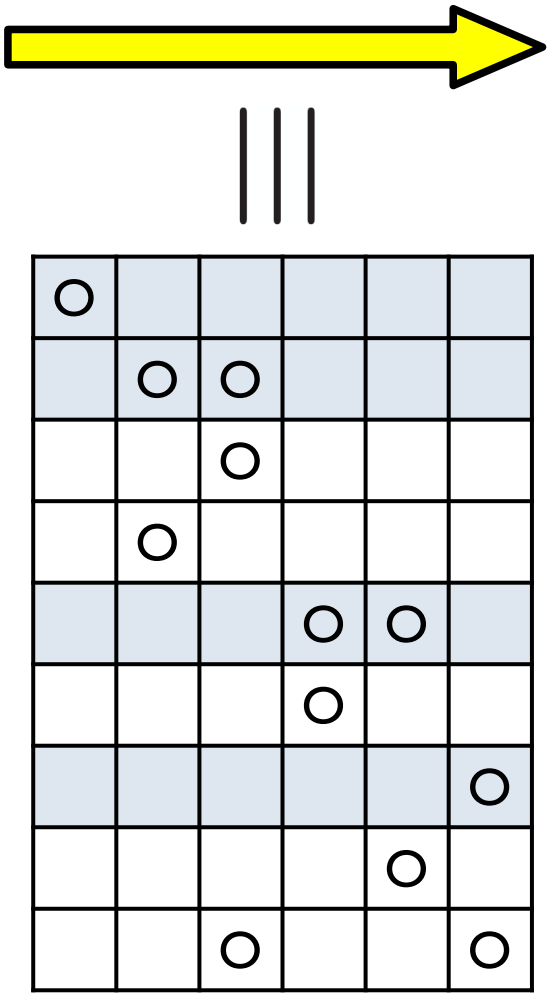

```
1   out = malloc(1);
2   in = malloc(1);
3   … // use in, out
4   free(out);
5   free(in);
6
7   in = malloc(2);
8   if(in == NULL) {
9
10    goto err;
11  }
12  free(out);
13  out = malloc(2);
14  if(out == NULL) {
15    free(in);
16
17    goto err;
18  }
19  … // use in, out
20 err:
21   free(in);
22   free(out);
```

FSE'18

Key Idea of MemFix

```
1   out = malloc(1);
2   in = malloc(1);
3   … // use in, out
4   free(out);
5   free(in);
6
7   in = malloc(2);
8   if(in == NULL) {
9
10    goto err;
11  }
12
13  out = malloc(2);
14  if(out == NULL) {
15    free(in);
16
17    goto err;
18  }
19  … // use in, out
20 err:
21   free(in);
22   free(out);
```

Find a set of free-statements



Find an Exact Cover

```
1   out = malloc(1);
2   in = malloc(1);
3   … // use in, out
4   free(out);
5   free(in);
6
7   in = malloc(2);
8   if(in == NULL) {
9
10    goto err;
11  }
12  free(out);
13  out = malloc(2);
14  if(out == NULL) {
15    free(in);
16
17    goto err;
18  }
19  … // use in, out
20 err:
21   free(in);
22   free(out);
```

FSE'18

Exact Cover Problem

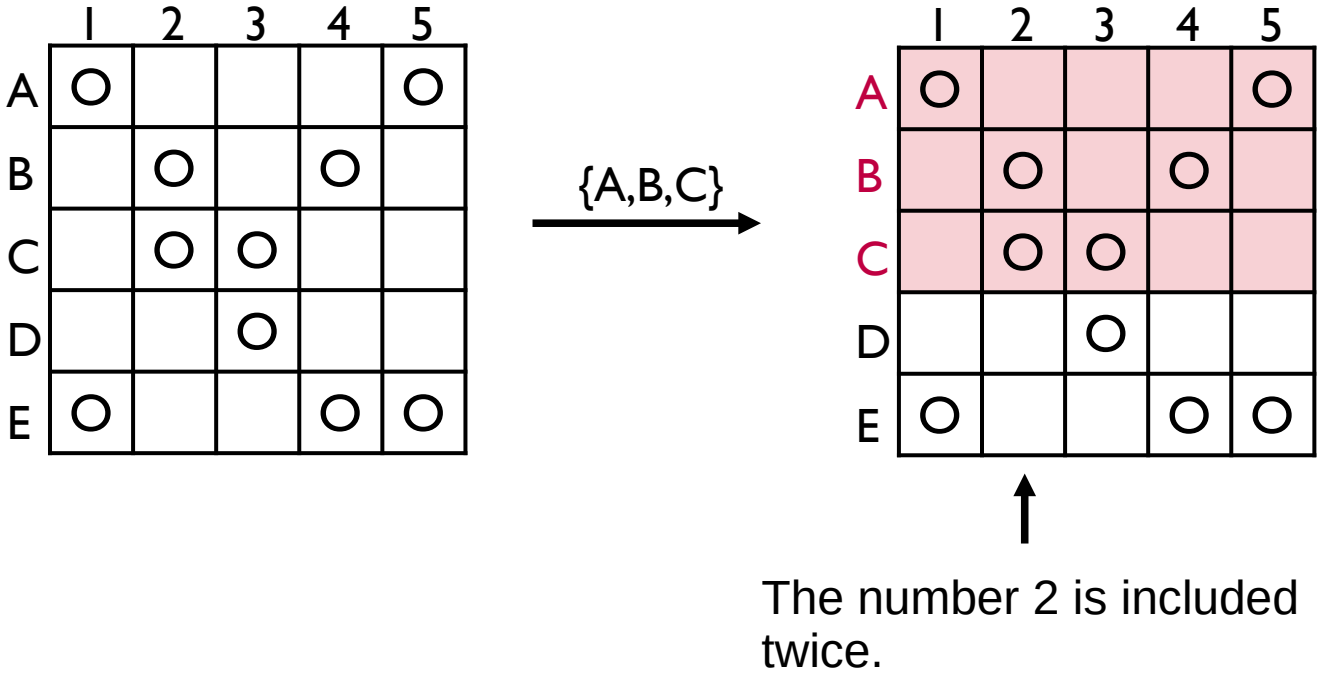
- One of the NP-complete problems
  - ex) Find a subset of the set {A, B, C, D, E}, where the subset contains the numbers 1, 2, 3, 4, 5 only once.

	1	2	3	4	5
A	O				O
B		O		O	
C		O	O		
D			O		
E	O			O	O

FSE'18

Exact Cover Problem

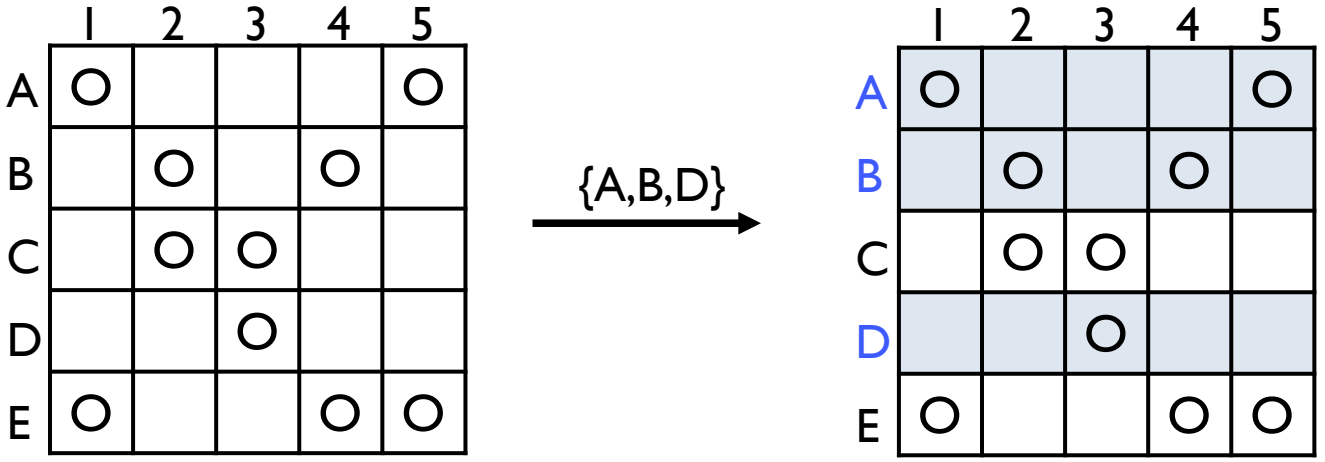
- One of the NP-complete problems
  - ex) Find a subset of the set {A, B, C, D, E}, where the subset contains the numbers 1, 2, 3, 4, 5 only once.



FSE'18

Exact Cover Problem

- One of the NP-complete problems
  - ex) Find a subset of the set {A, B, C, D, E}, where the subset contains the numbers 1, 2, 3, 4, 5 only once.



Exact Cover!

How Does MemFix Work?

- Goal: Automatically fixing the double-free error.

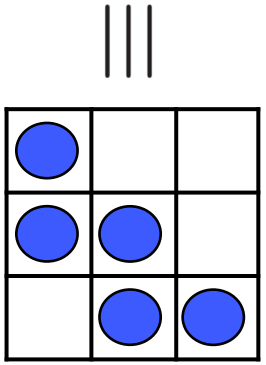
```
1   p = malloc(1);
2   if(…) {
3     q = malloc(2);
4
5   }
6   else
7     q = p;
8   … // use q
9   free(p);
10  free(q);
```

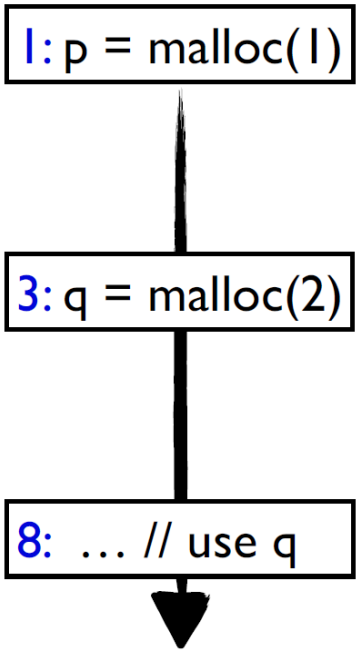
FSE'18

How Does MemFix Work?

- Enumerate all object traces.

```
1   p = malloc(1); //o1
2   if(…) {
3     q = malloc(2); //o2
4
5   }
6   else
7     q = p;
8   … // use q
9   free(p);
10  free(q);
```





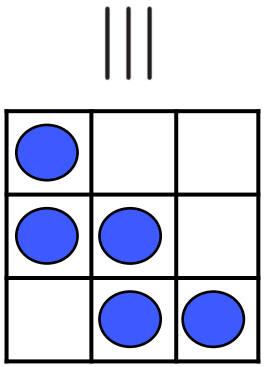
Object traces

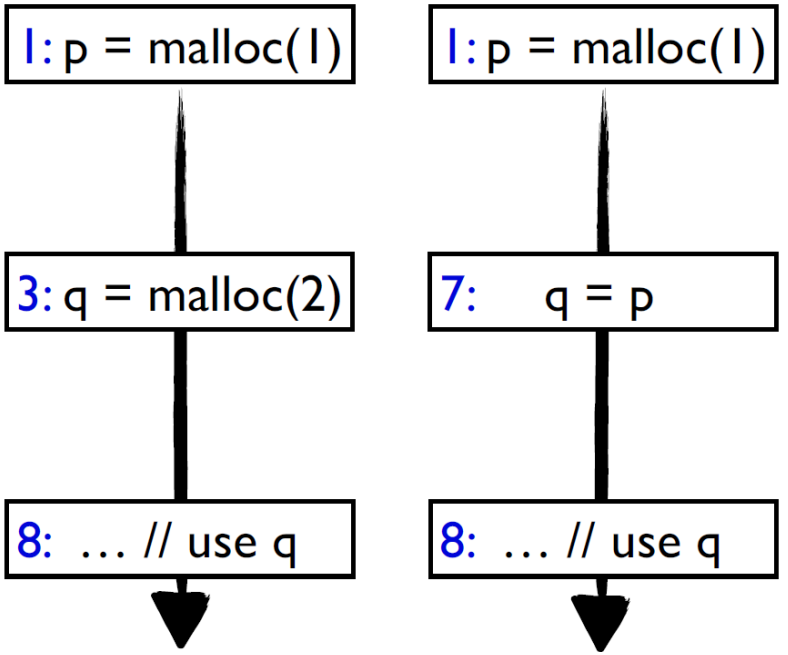
FSE'18

How Does MemFix Work?

- Enumerate all object traces.

```
1   p = malloc(1); //o1
2   if(…) {
3     q = malloc(2); //o2
4
5   }
6   else
7     q = p;
8   … // use q
9   free(p);
10  free(q);
```





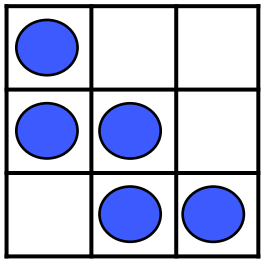
Object traces

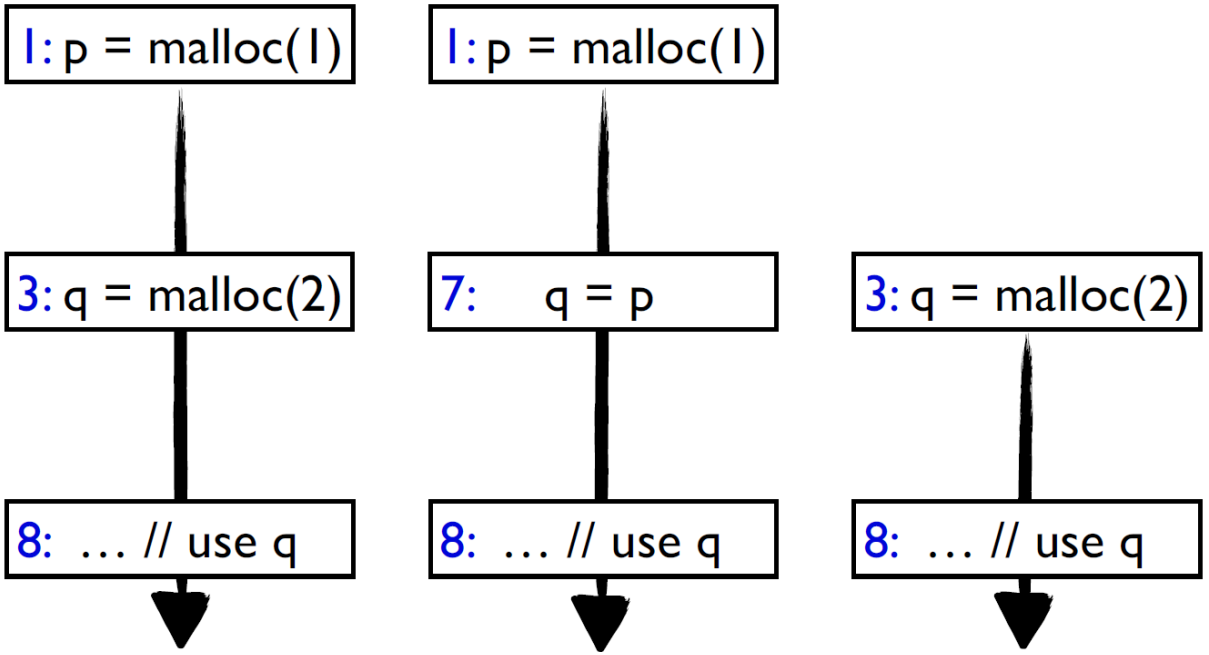
FSE'18

How Does MemFix Work?

- Enumerate all object traces.

```
1   p = malloc(1); //o1
2   if(…) {
3     q = malloc(2); //o2
4
5   }
6   else
7     q = p;
8   … // use q
9   free(p);
10  free(q);
```





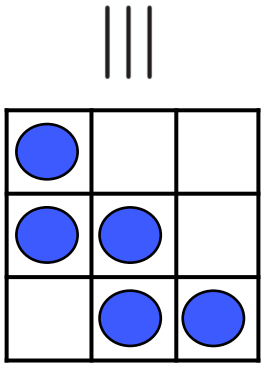
Object traces

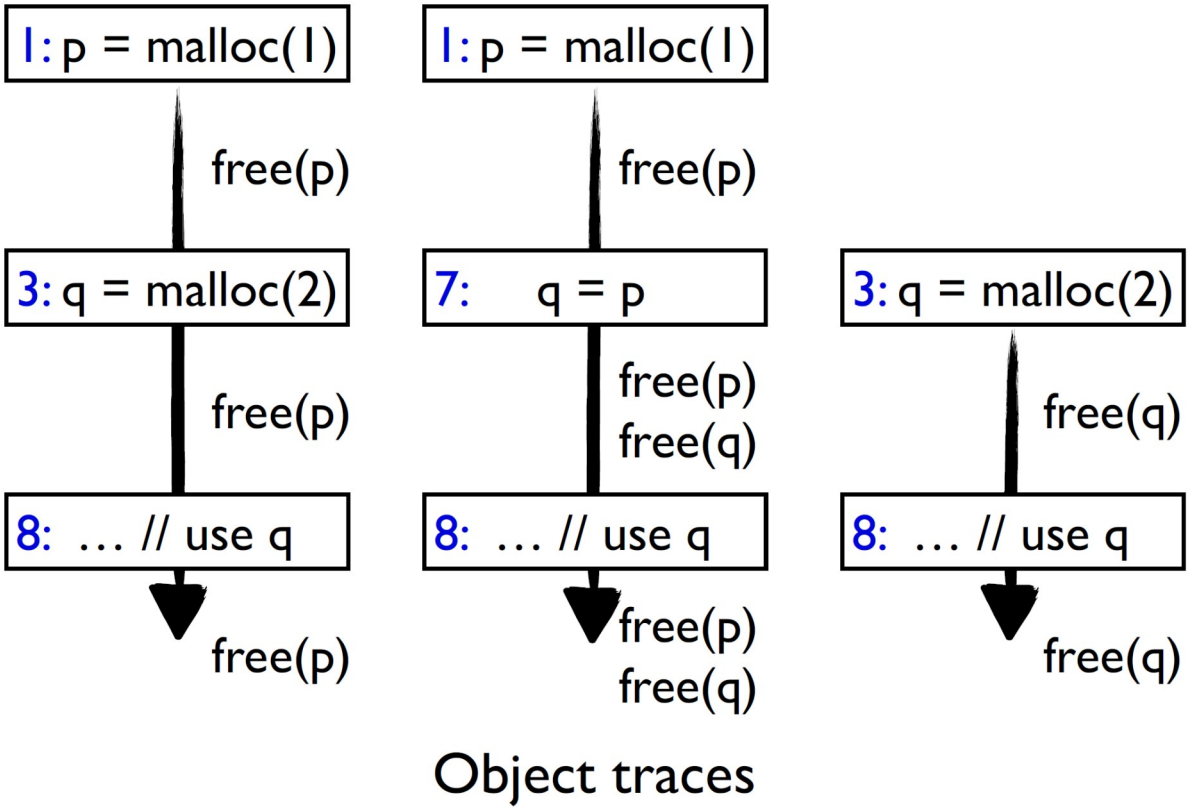
FSE'18

How Does MemFix Work?

- Enumerate all candidate patches for each trace.

```
1   p = malloc(1); //o1
2   if(…) {
3     q = malloc(2); //o2
4
5   }
6   else
7     q = p;
8   … // use q
9   free(p);
10  free(q);
```





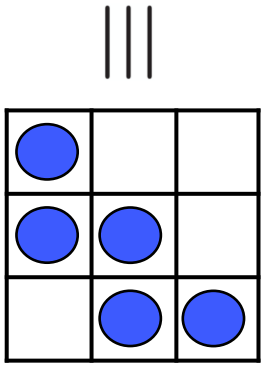
Object traces

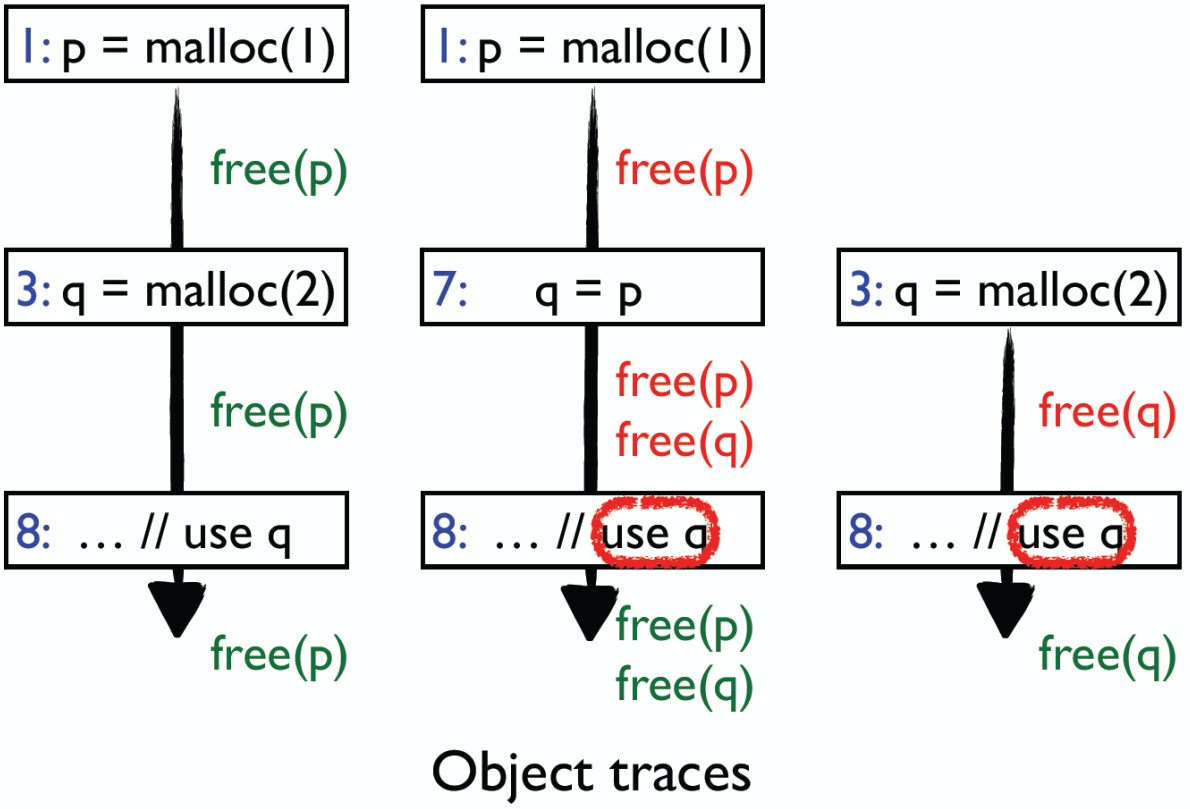
FSE'18

How Does MemFix Work?

- Find safe patches for each trace.

```
1   p = malloc(1); //o1
2   if(…) {
3     q = malloc(2); //o2
4
5   }
6   else
7     q = p;
8   … // use q
9   free(p);
10  free(q);
```





Object traces

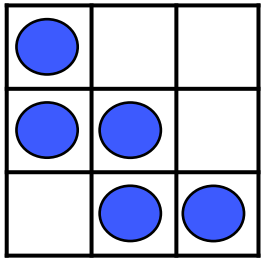
FSE'18

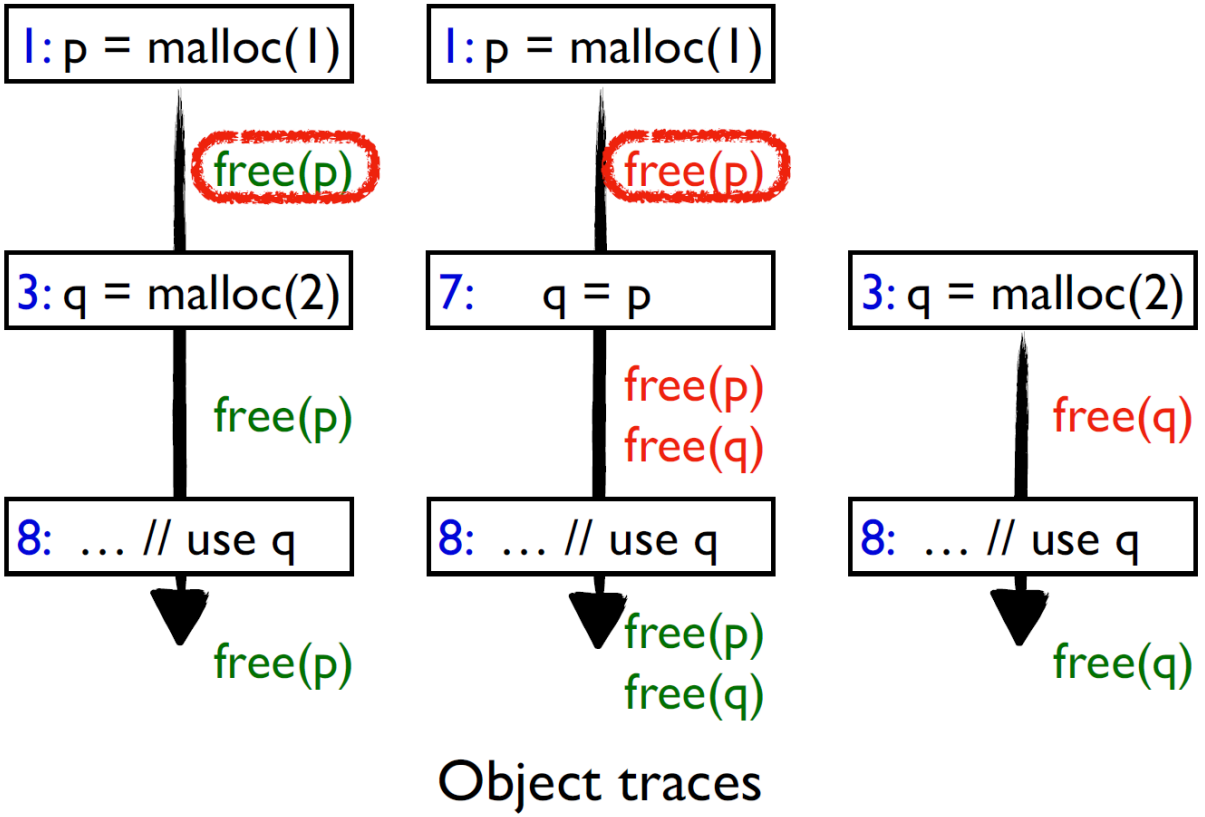
How Does MemFix Work?

- Find safe patches for each trace.

```
1   p = malloc(1); //o1
2   if(…) {
3     q = malloc(2); //o2
4
5   }
6   else
7     q = p;
8   … // use q
9   free(p);
10  free(q);
```

|||





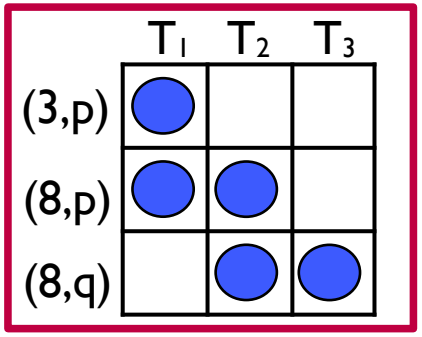
Object traces

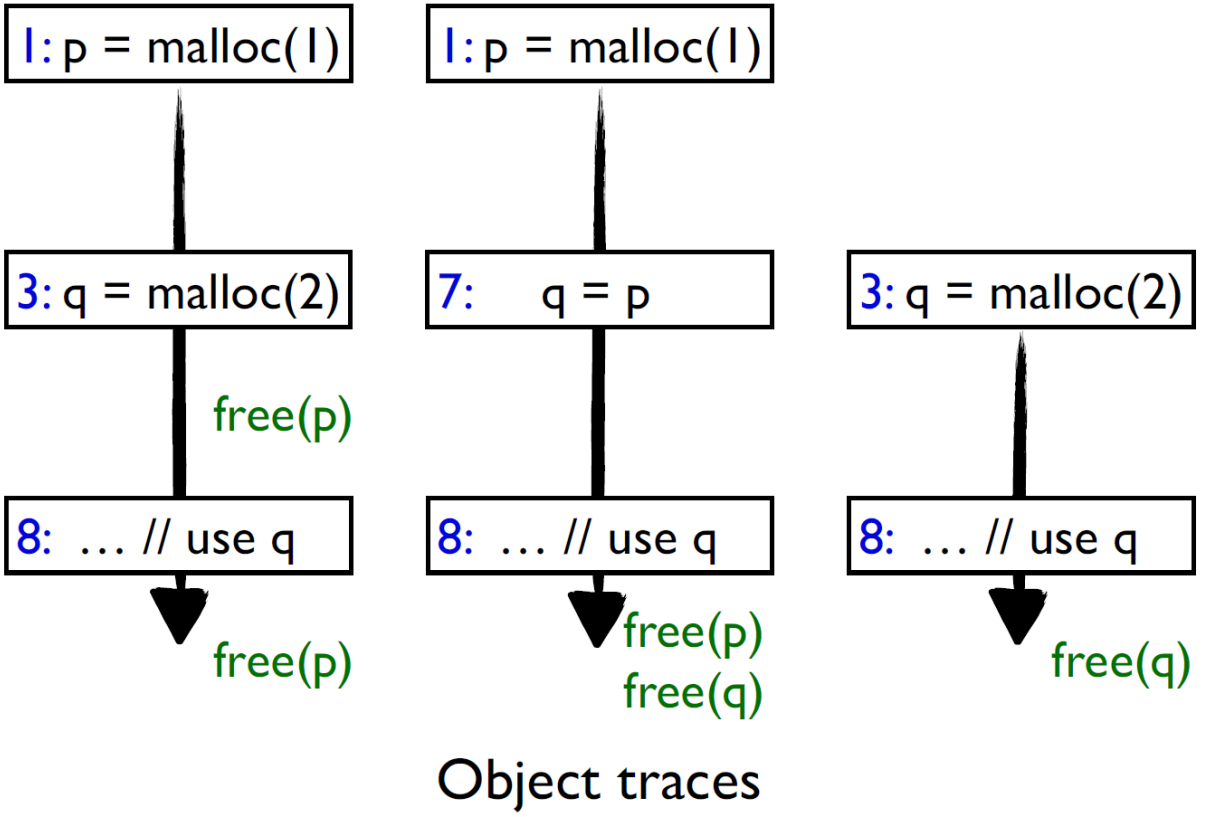
FSE'18

How Does MemFix Work?

- Find safe patches for each trace.

```
1   p = malloc(1); //o1
2   if(…) {
3     q = malloc(2); //o2
4
5   }
6   else
7     q = p;
8   … // use q
9   free(p);
10  free(q);
```





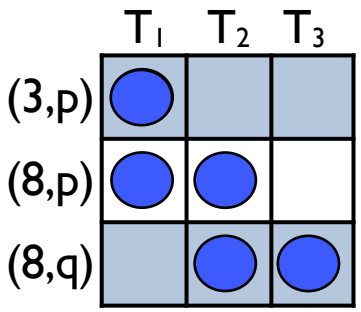
Object traces

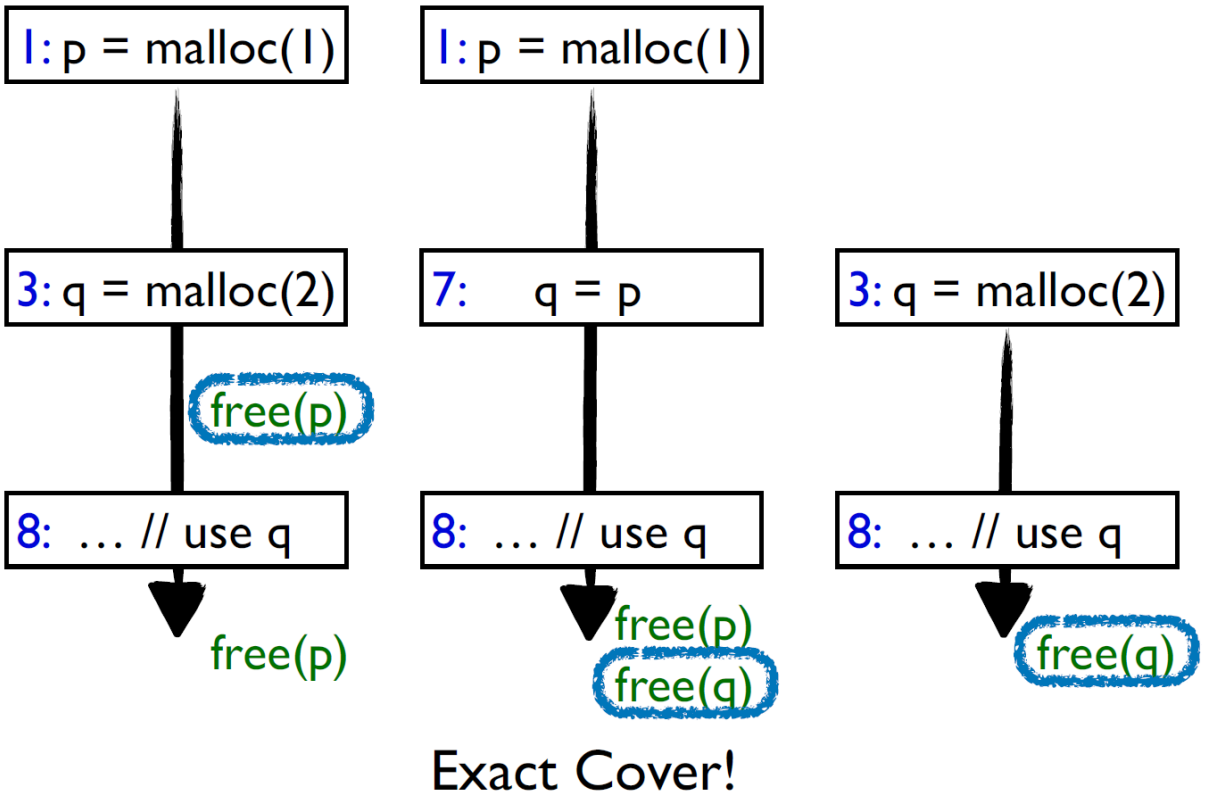
FSE'18

How Does MemFix Work?

- Find an exact cover.

```
1   p = malloc(1); //o1
2   if(…) {
3     q = malloc(2); //o2
4
5   }
6   else
7     q = p;
8   … // use q
9   free(p);
10  free(q);
```





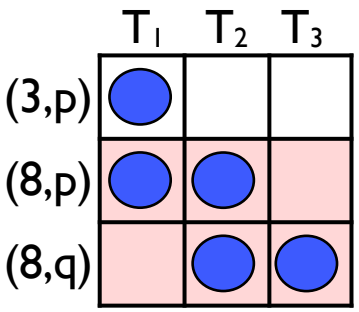
Exact Cover!

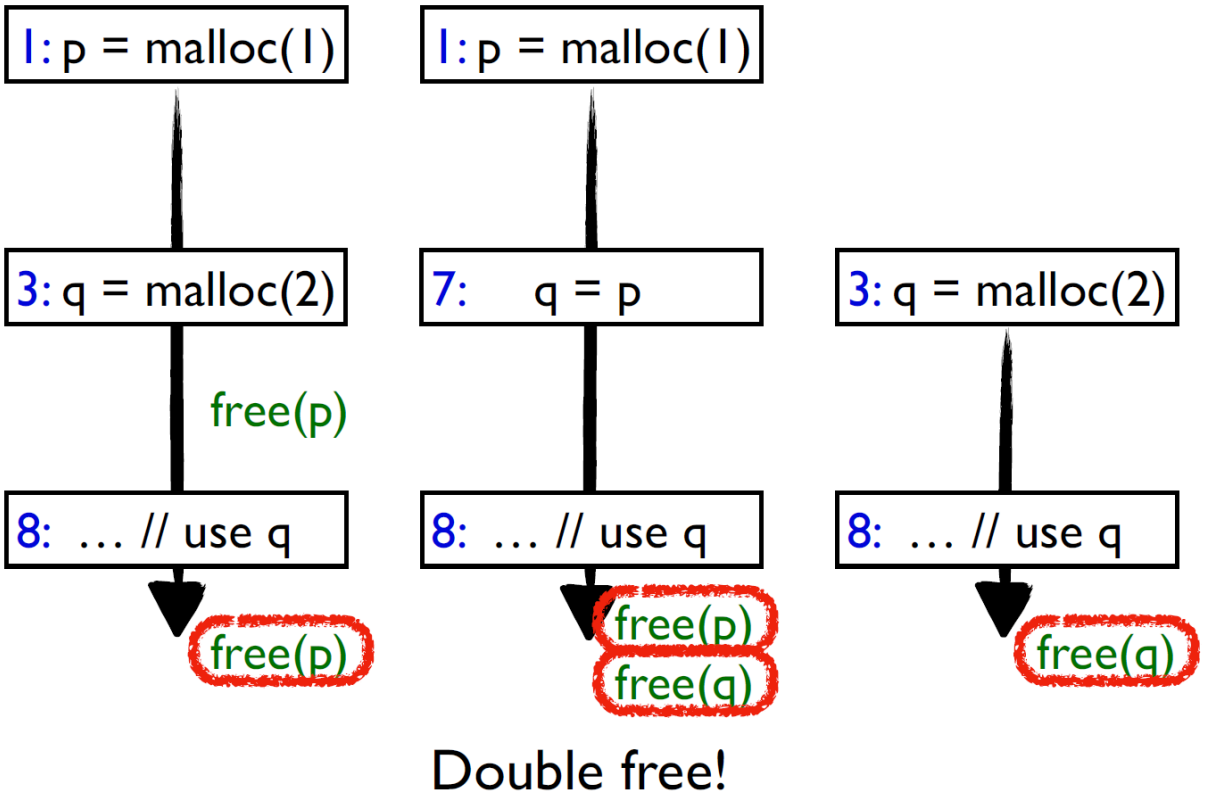
FSE'18

How Does MemFix Work?

- Find an exact cover.

```
1   p = malloc(1); //o1
2   if(…) {
3     q = malloc(2); //o2
4
5   }
6   else
7     q = p;
8   … // use q
9   free(p);
10  free(q);
```





FSE'18

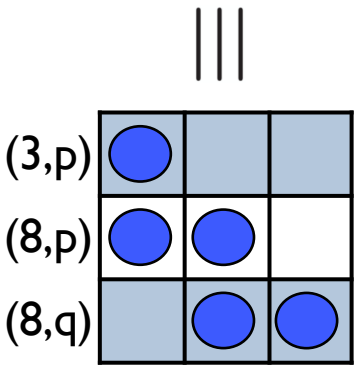
How Does MemFix Work?

- Apply the patch (3, p), (8, q)

```
1   p = malloc(1);
2   if(…) {
3     q = malloc(2);
4
5   }
6   else
7     q = p;
8   … // use q
9   free(p);
10  free(q);
```

```
1   p = malloc(1);
2   if(…) {
3     q = malloc(2);
4     free(p);
5   }
6   else
7     q = p;
8   … // use q
9   free(p);
10  free(q);
```

Correct Patch!



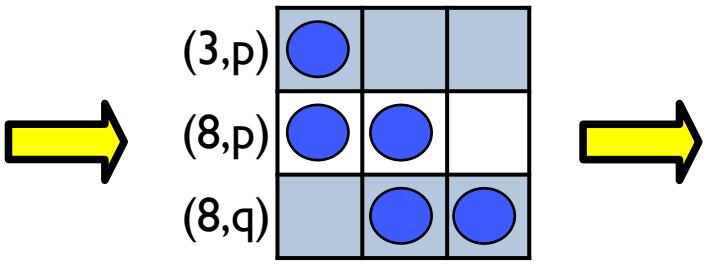
FSE'18

How Does MemFix Work?

- Key Idea: Static Analysis + Exact Cover Problem

```
1   p = malloc(1);
2   if(…) {
3     q = malloc(2);
4
5   }
6   else
7     q = p;
8   … // use q
9   free(p);
10  free(q);
```

Buggy Program



Exact Cover Problem

```
1   p = malloc(1);
2   if(…) {
3     q = malloc(2);
4     free(p);
5   }
6   else
7     q = p;
8   … // use q
9   free(p);
10  free(q);
```

Patched Program

Effectiveness of MemFix

- 24 Programs in GNU Coreutils
  - Use the programs with free statements removed
- MemFix vs LeakFix (ICSE'15)

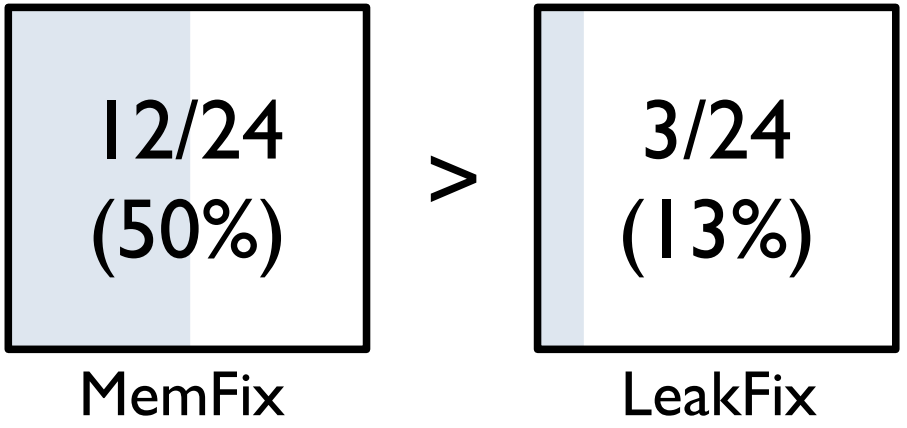
FSE'18

Effectiveness of MemFix

- 24 Programs in GNU Coreutils (553 ~ 3,475 LoC)
  - Use the programs with free statements removed

- MemFix vs LeakFix (ICSE'15)

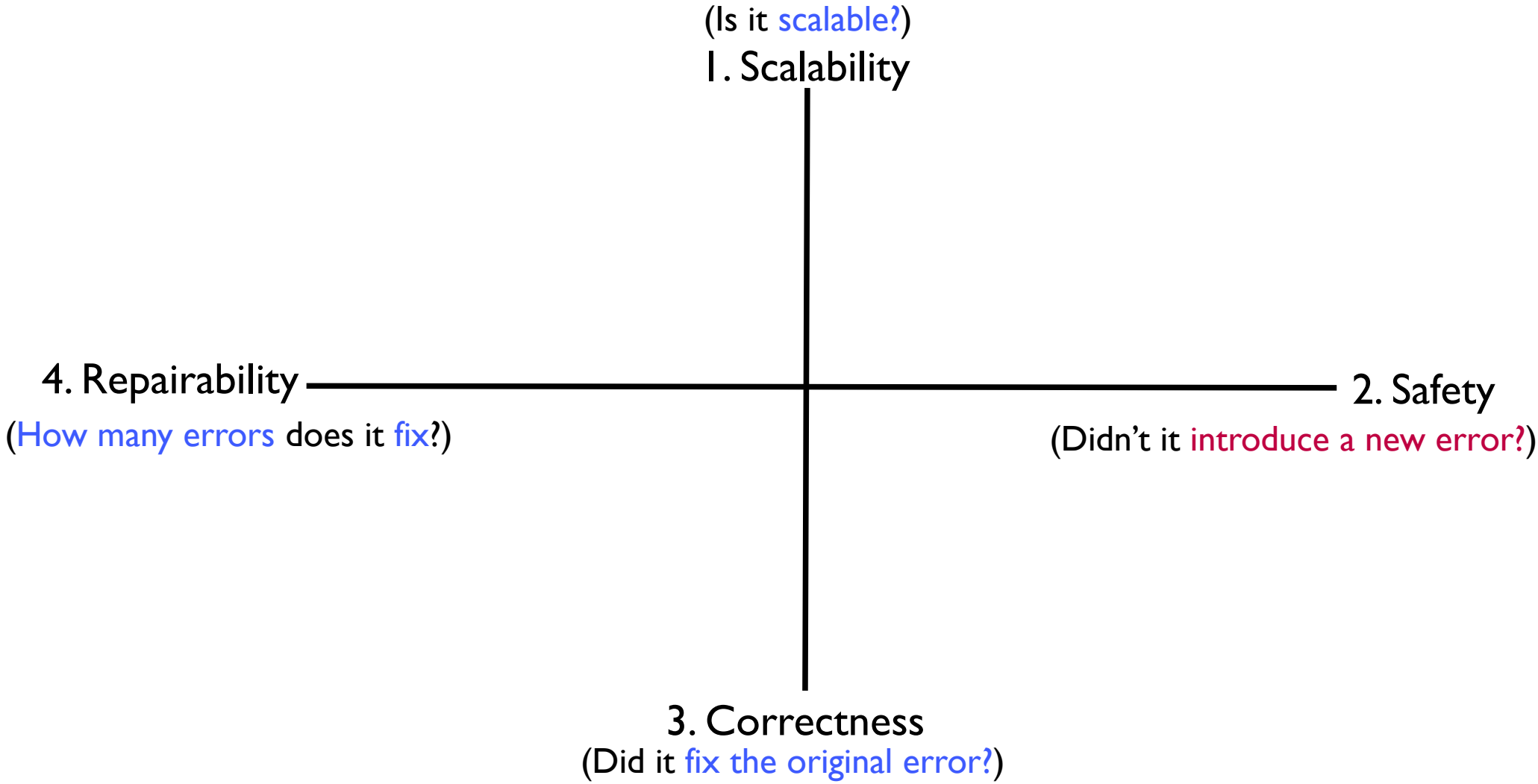
The number of correct patches



Programs	LoC	#Al.	MemFIx		LeakFix	
			#Ins.	sec	#Ins.	sec
yes	553	1	1	< 1.0	✗	< 1.0
users	577	1	1	< 1.0	✗	< 1.0
unexpand	707	1	1	< 1.0	✗	< 1.0
tee	779	1	1	< 1.0	1	< 1.0
mktemp	794	4	✗	1.3	✗	< 1.0
tsort	920	3	✗	1.4	✗	< 1.0
paste	982	3	3	2.4	△/3	< 1.0
date	1,054	1	1	3.5	✗	< 1.0
cut	1,056	1	✗	2.0	✗	< 1.0
nl	1,063	4	4	4.0	✗	< 1.0
pinky	1,120	3	4	5.2	✗	< 1.0
cat	1,209	3	✗	9.3	✗	< 1.0
ln	1,258	2	✗	5.2	✗	< 1.0
printf	1,288	1	1	3.0	✗	< 1.0
stdbuf	1,605	3	3	1.3	✗	< 1.0
wc	1,669	1	1	7.3	△/2	< 1.0
shred	1,822	5	✗	31.1	✗	< 1.0
cp	1,926	8	✗	430.7	✗	< 1.0
install	2,076	1	✗	13.4	✗	< 1.0
who	2,156	8	✗	36.8	✗	< 1.0
tr	2,304	10	✗	20.0	✗	< 1.0
expr	2,378	9	✗	13.0	✗	< 1.0
stat	2,439	10	6	130.3	✗	< 1.0
dd	3,475	2	✗	52.2	✗	< 1.0

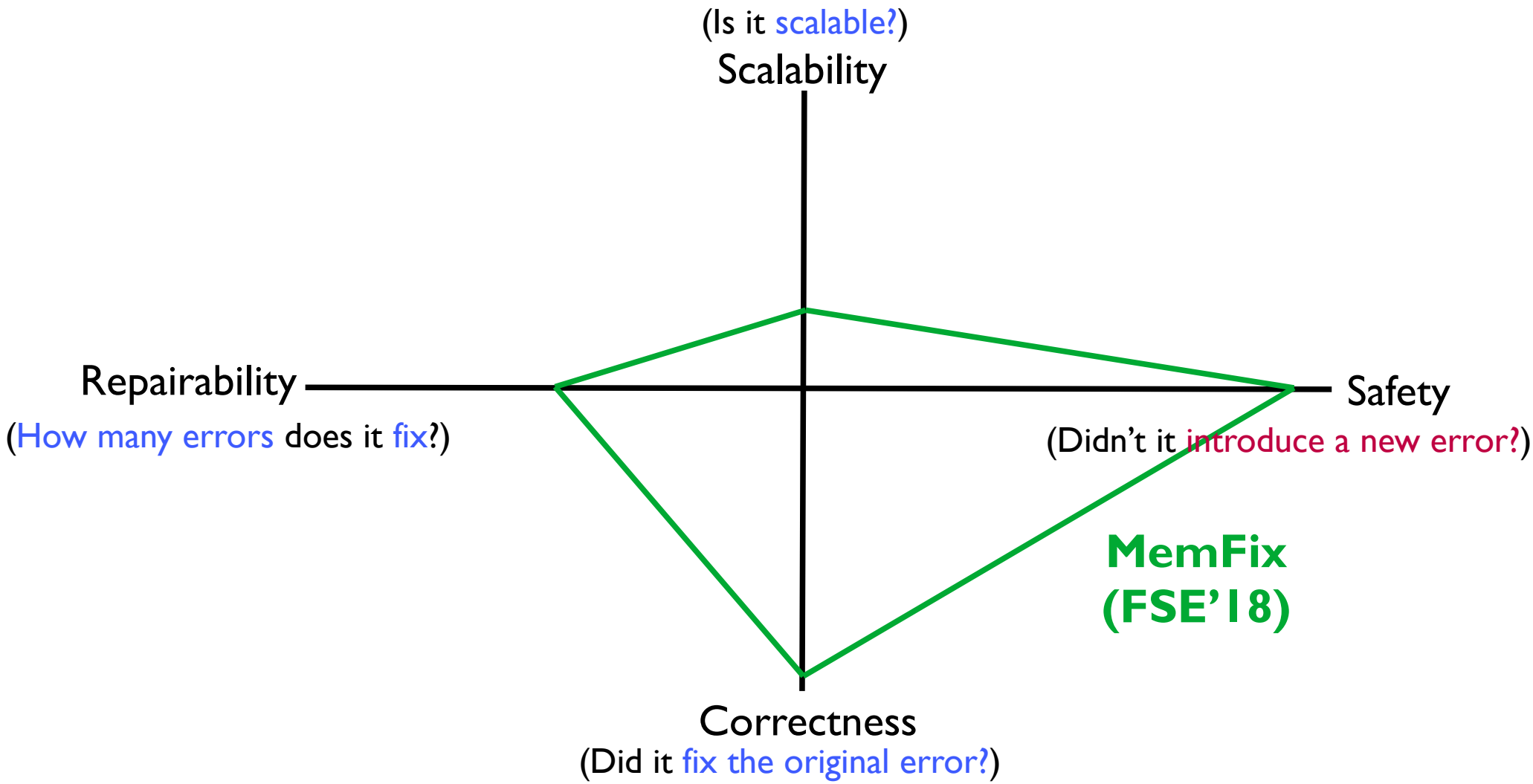
Limitations

- The limitations for state-of-the-art APR techniques.



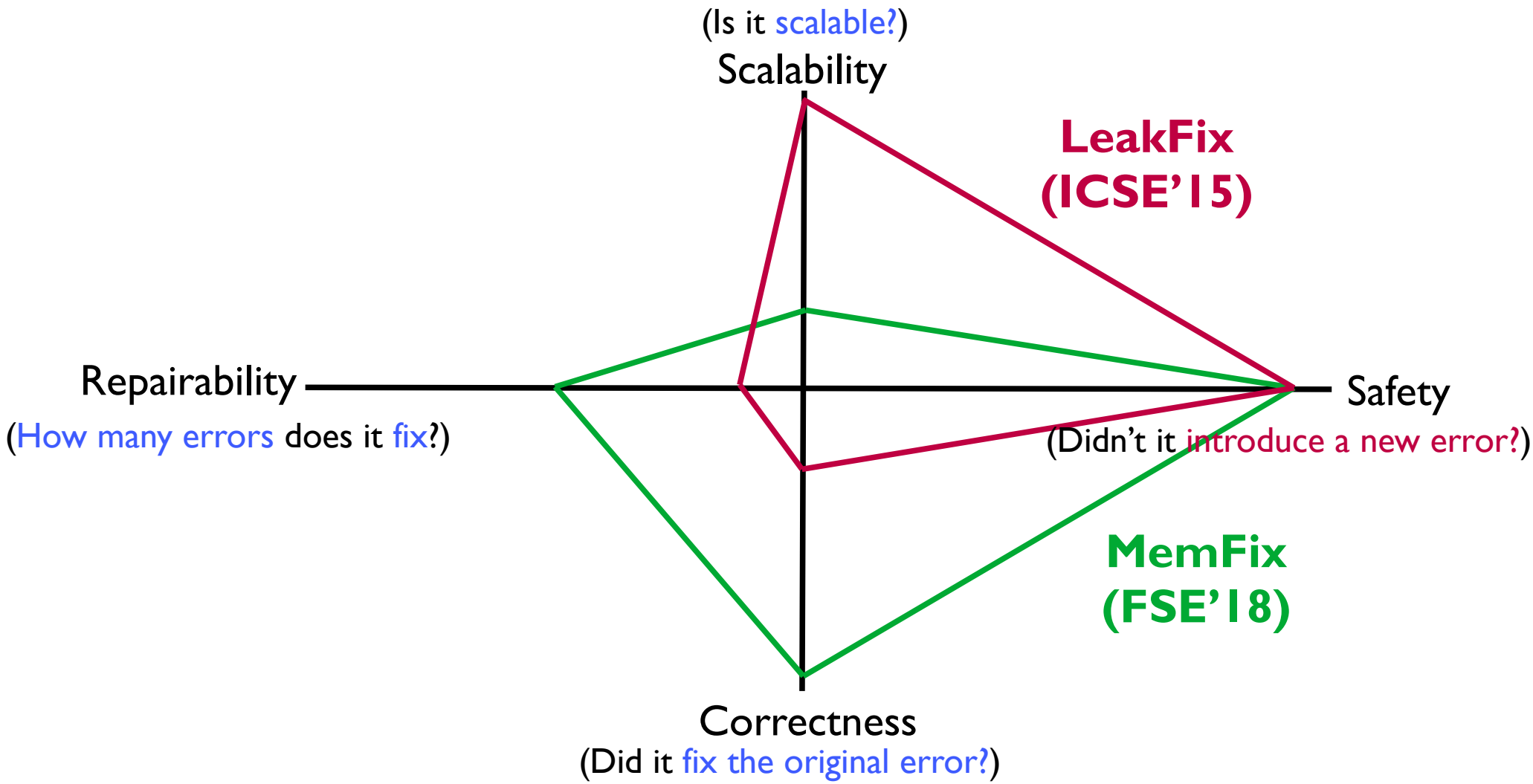
Limitations

- The limitations for state-of-the-art APR techniques.



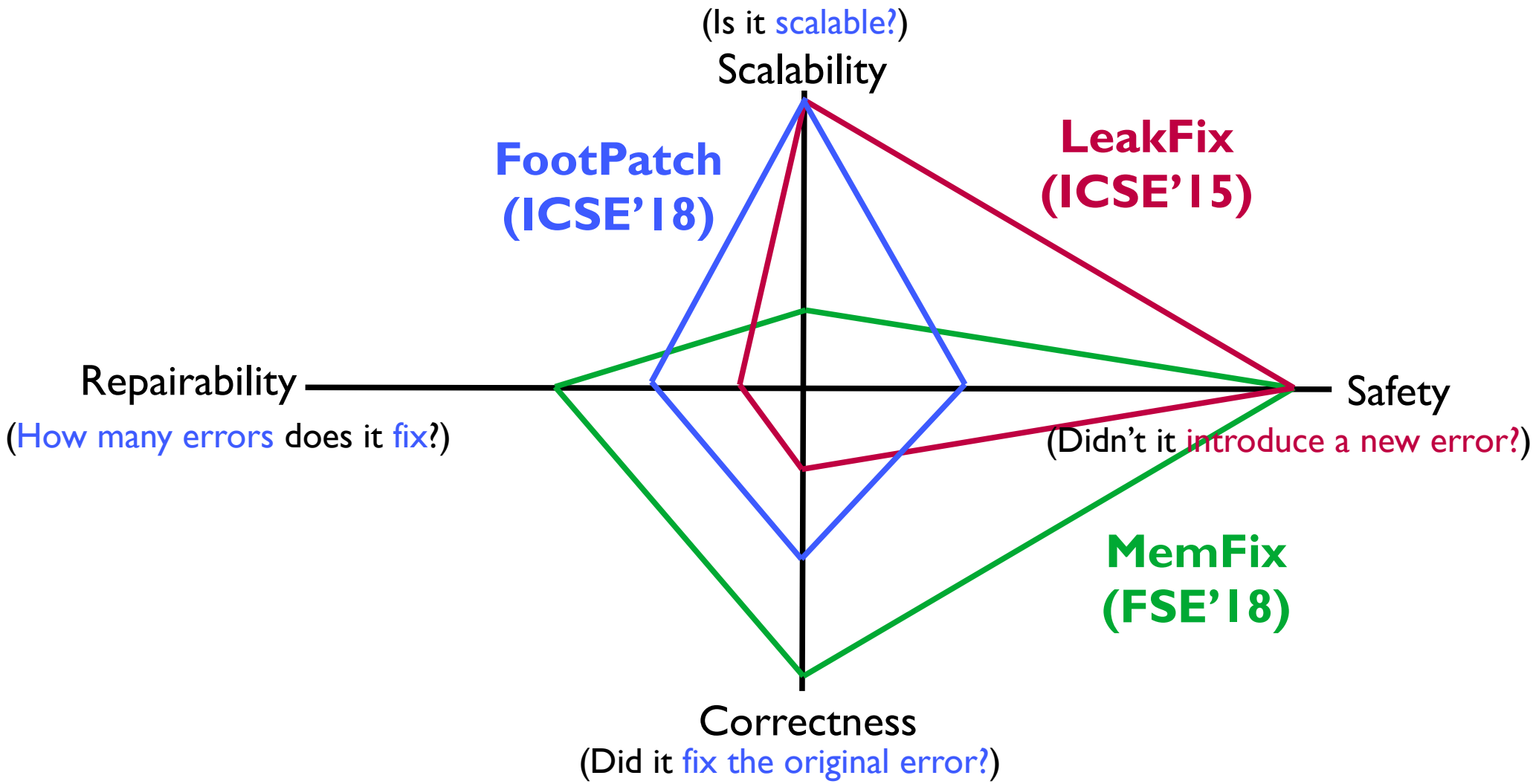
Limitations

- The limitations for state-of-the-art APR techniques.



Limitations

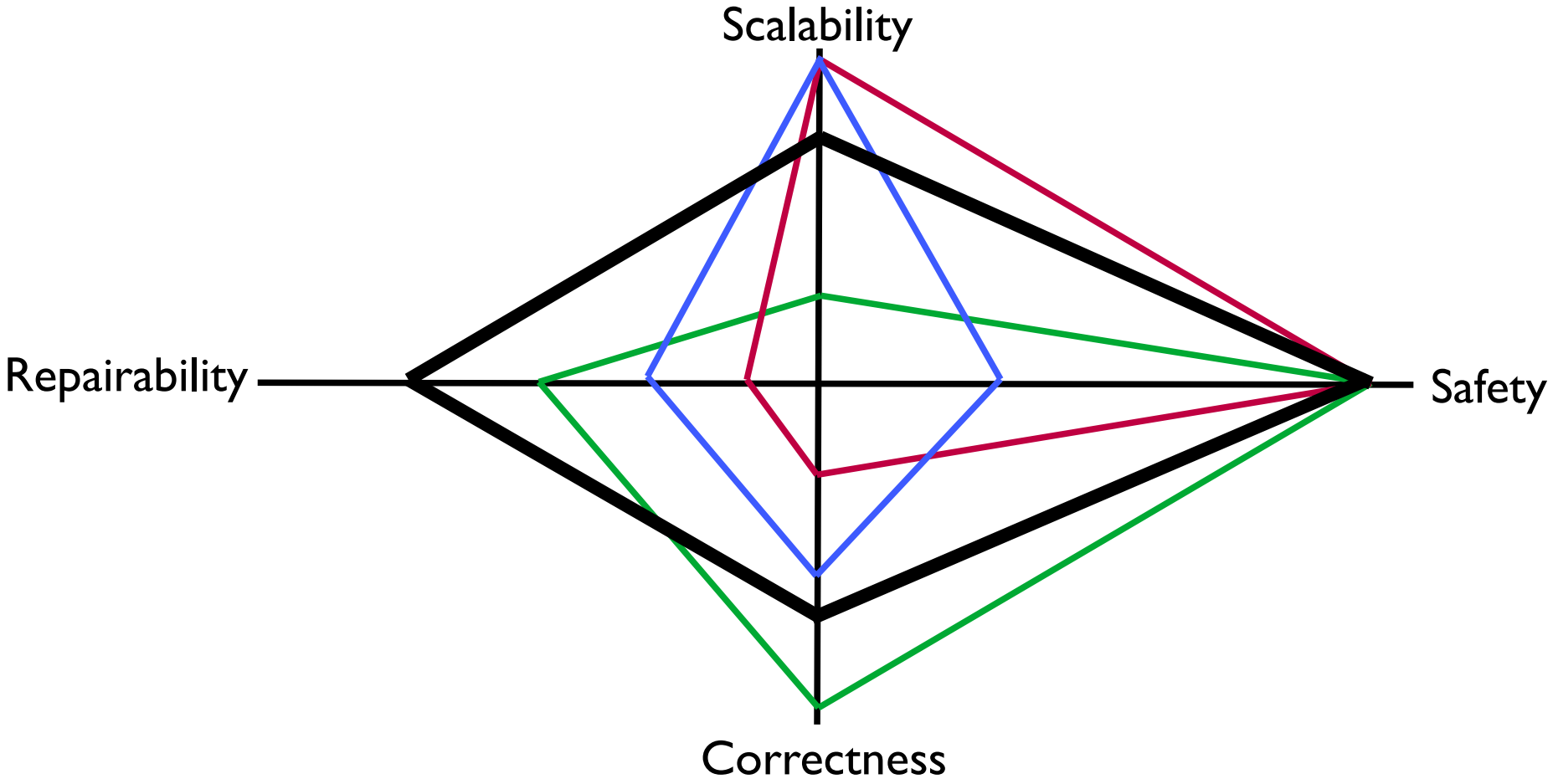
- The limitations for state-of-the-art APR techniques.



ICSE'20

SAVER

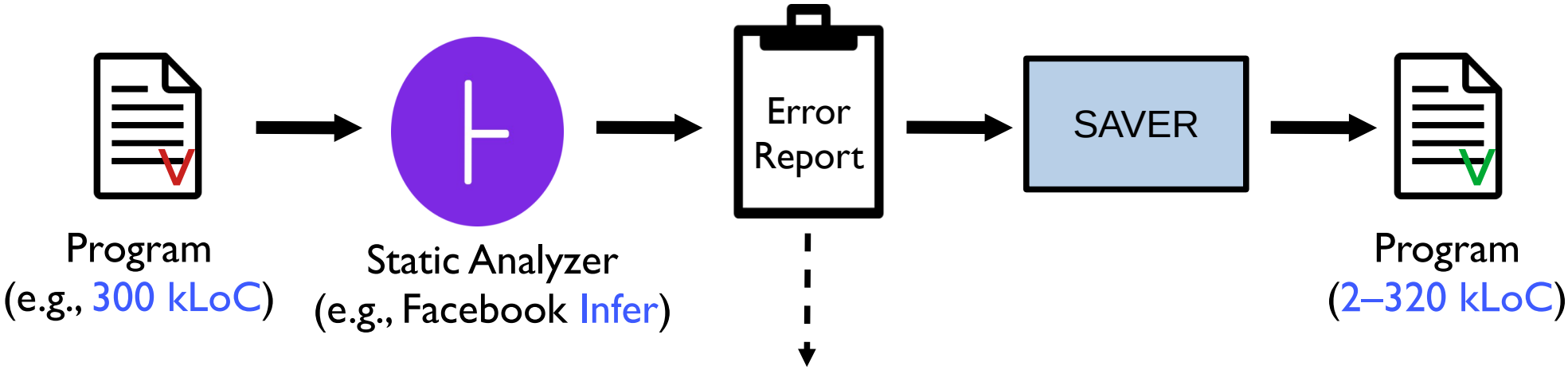
- SAVER: Scalable, Precise, and Safe Memory-Error Repair



ICSE'20

SAVER

- Usage Scenario



How Does SAVER Work?

- How can we fix the toy program?

```
1  p = malloc(); // o1
2  if (C)
3    q = p;
4  else
5    q = malloc(); // o2
6  *p = 1;
7  free(q);
```

How Does SAVER Work?

- How can we fix the toy program?

```
1  p = malloc(); // o1
2  if (C)
3    q = p;
4  else
5    q = malloc(); // o2
6  *p = 1;
7  free(q);
```

← Memory leak

An object allocated at line 1 is unreachable after line 7

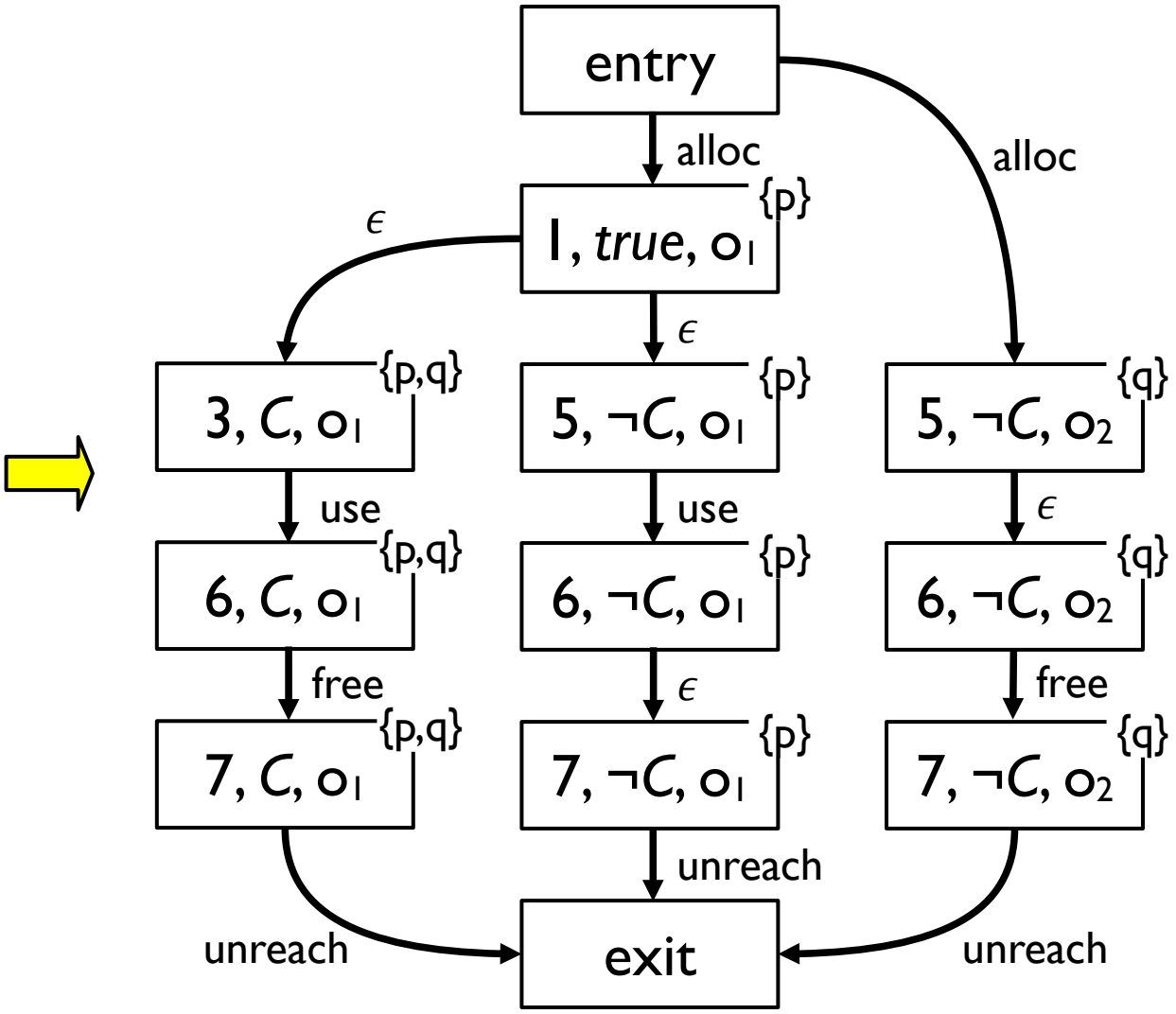
ICSE'20

How Does SAVER Work?

- Step1: Construct an object flow graph (*OFG*).

```
1  p = malloc(); // o1
2  if (C)
3    q = p;
4  else
5    q = malloc(); // o2
6  *p = 1;
7  free(q);
```

An object allocated at line 1 is unreachable after line 7



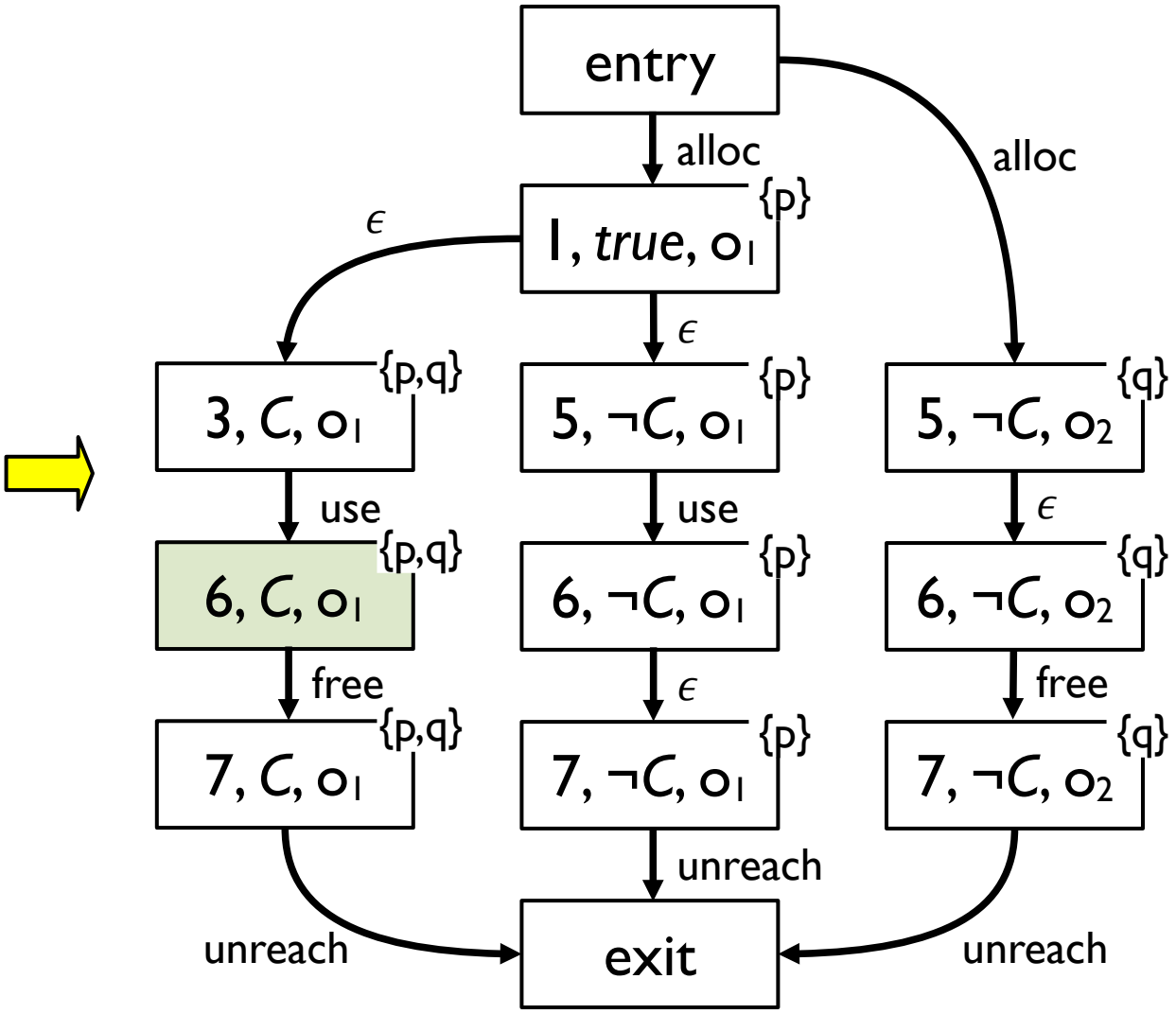
ICSE'20

How Does SAVER Work?

- Step1: Construct an object flow graph (*OFG*).

```
1  p = malloc(); // o1
2  if (C)
3    q = p;
4  else
5    q = malloc(); // o2
6  *p = 1;
7  free(q);
```

An object allocated at line 1 is unreachable after line 7



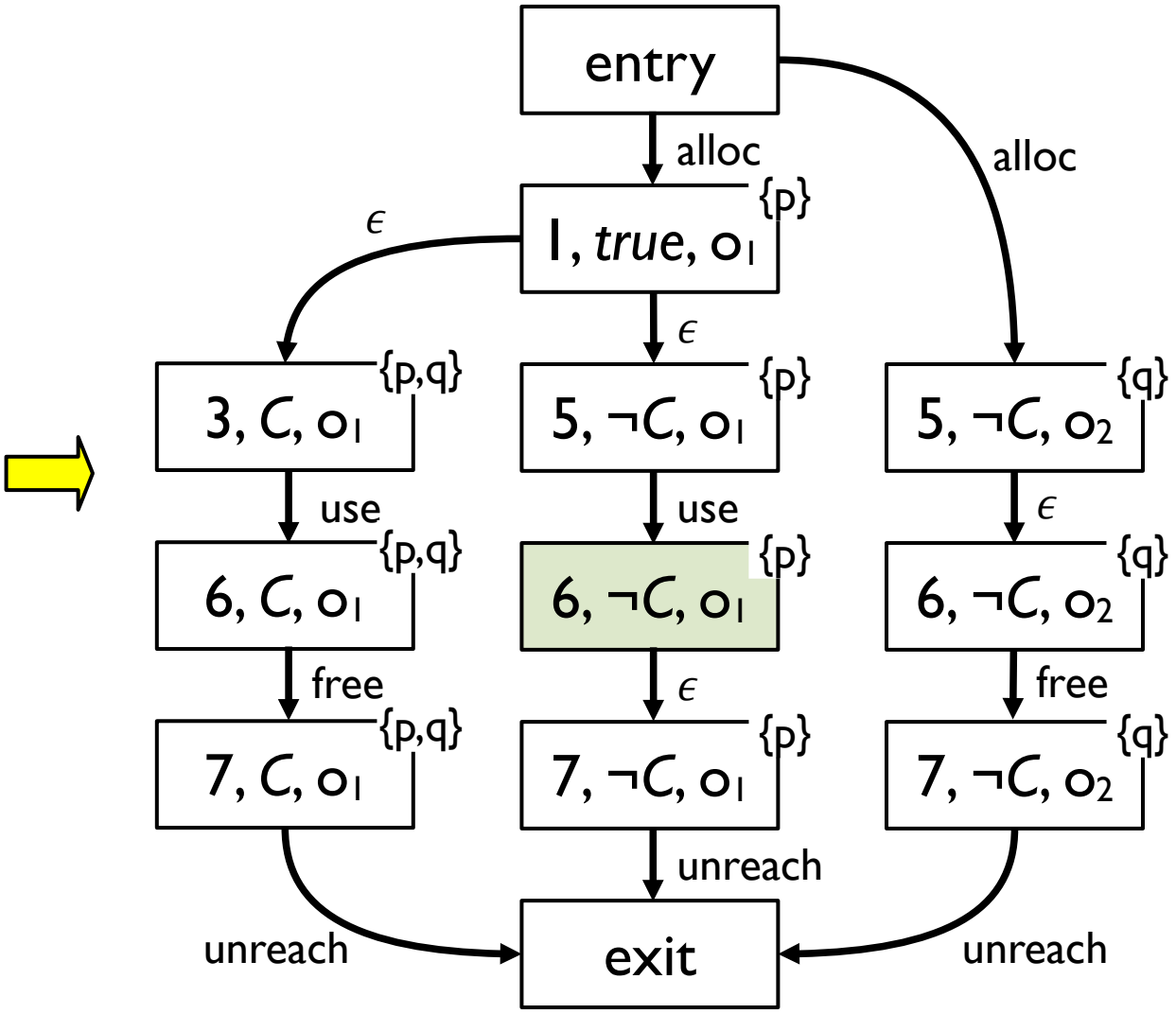
ICSE'20

How Does SAVER Work?

- Step1: Construct an object flow graph (*OFG*).

```
1  p = malloc(); // o1
2  if (C)
3    q = p;
4  else
5    q = malloc(); // o2
6  *p = 1;
7  free(q);
```

An object allocated at line 1 is unreachable after line 7



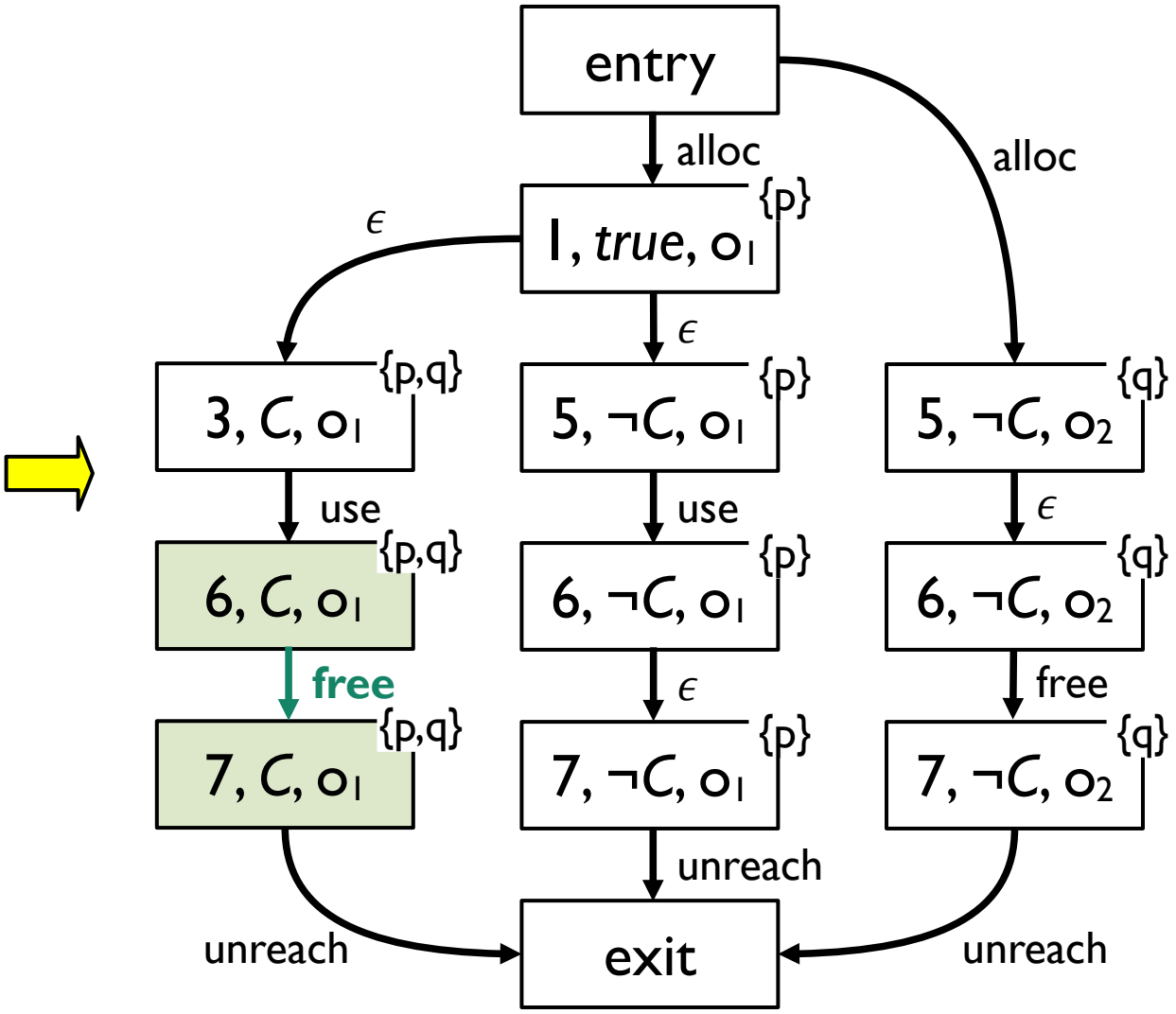
ICSE'20

How Does SAVER Work?

- Step1: Construct an object flow graph (*OFG*).

```
1  p = malloc(); // o1
2  if (C)
3    q = p;
4  else
5    q = malloc(); // o2
6  *p = 1;
7  free(q);
```

An object allocated at line 1 is unreachable after line 7



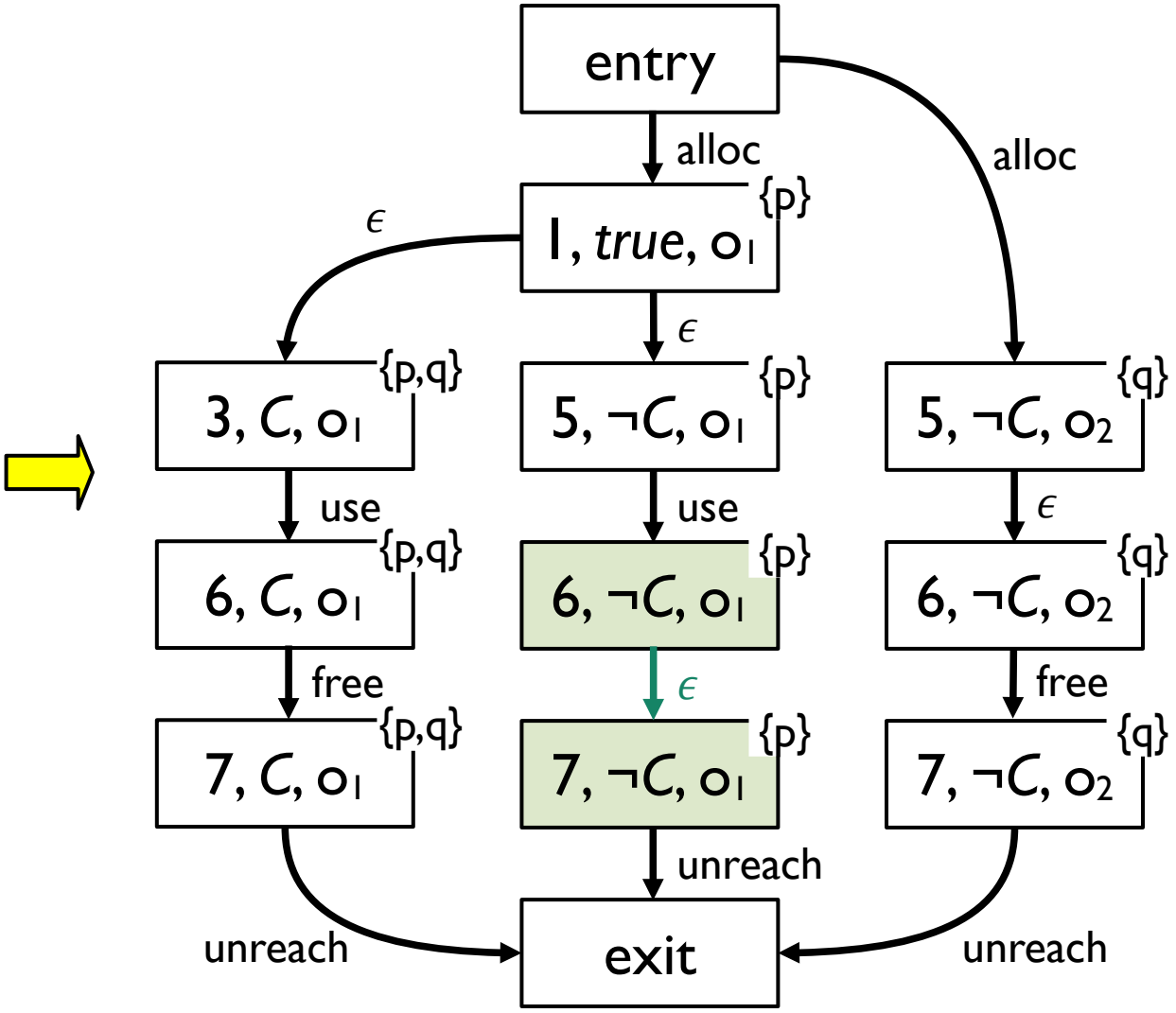
ICSE'20

How Does SAVER Work?

- Step1: Construct an object flow graph (*OFG*).

```
1  p = malloc(); // o1
2  if (C)
3    q = p;
4  else
5    q = malloc(); // o2
6  *p = 1;
7  free(q);
```

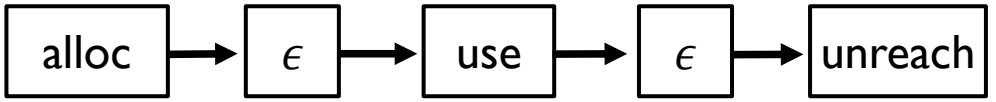
An object allocated at line 1 is unreachable after line 7



ICSE'20

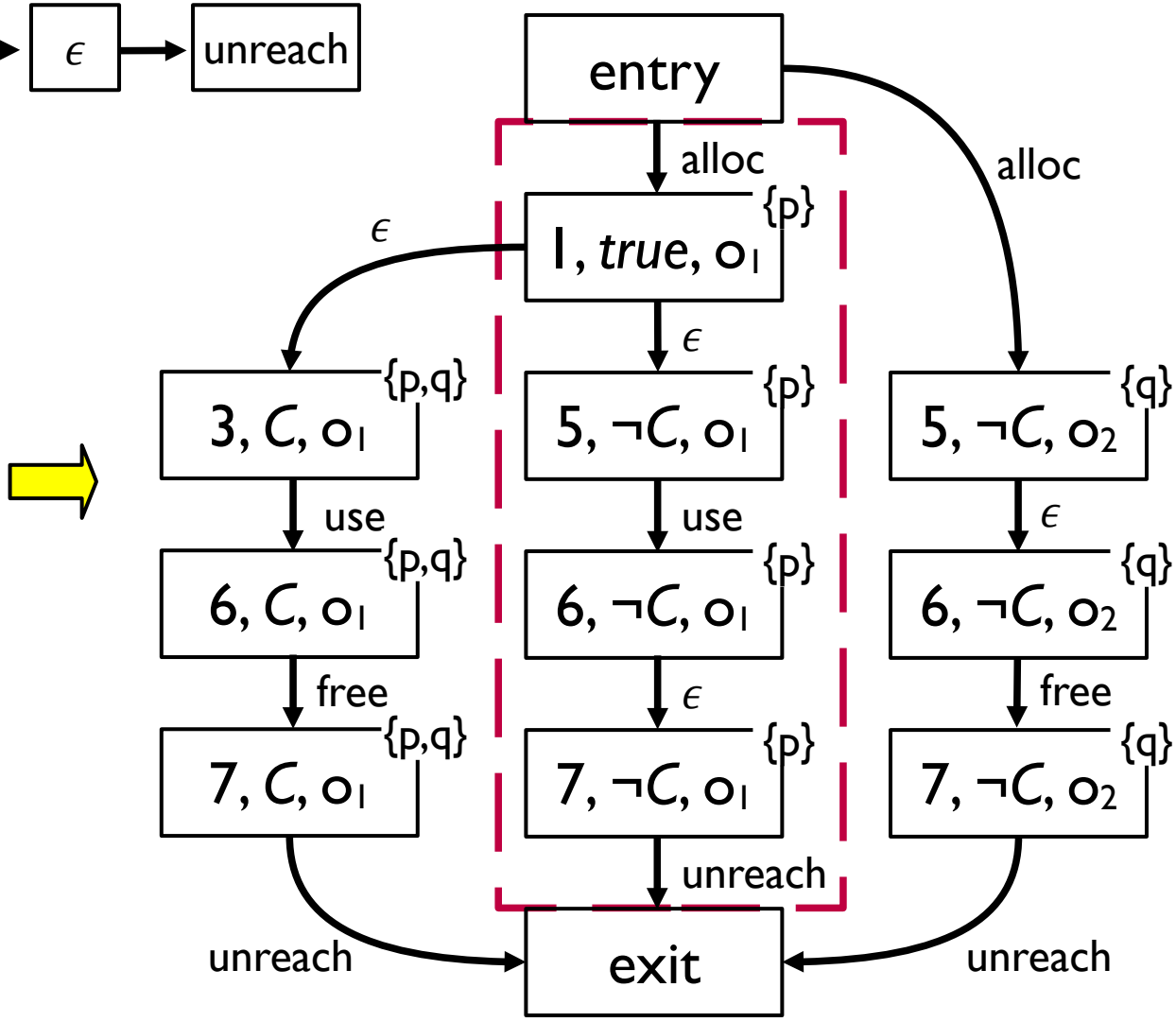
How Does SAVER Work?

- Step 2: Identify error paths.




```
1  p = malloc(); // o1
2  if (C)
3    q = p;
4  else
5    q = malloc(); // o2
6  *p = 1;
7  free(q);
```

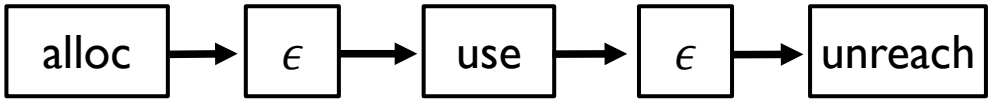
An object allocated at line 1 is unreachable after line 7



ICSE'20

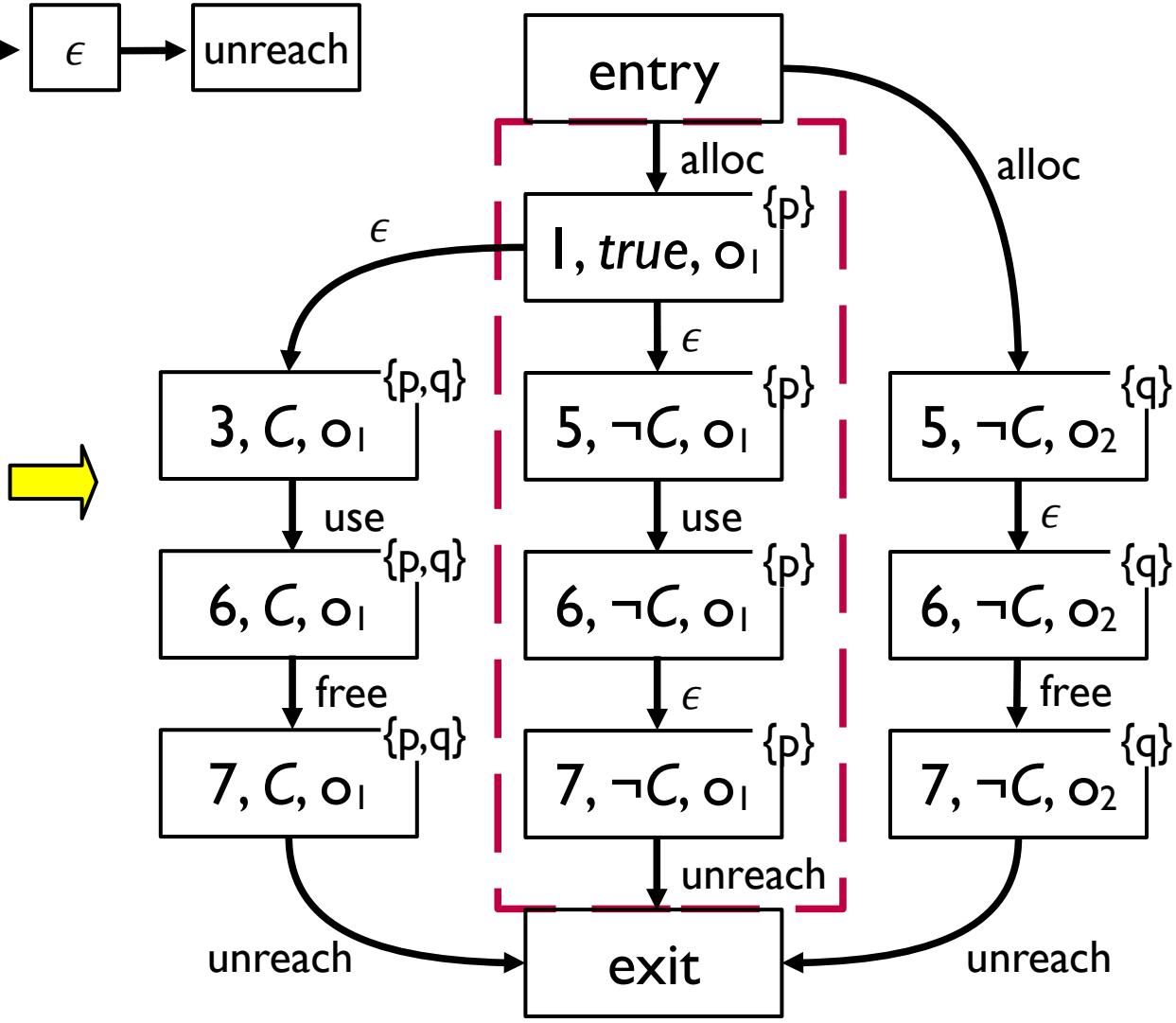
How Does SAVER Work?

- Step 3: Relabel the OFG.




```
1  p = malloc(); // o1
2  if (C)
3    q = p;
4  else
5    q = malloc(); // o2
6  *p = 1;
7  free(q);
```

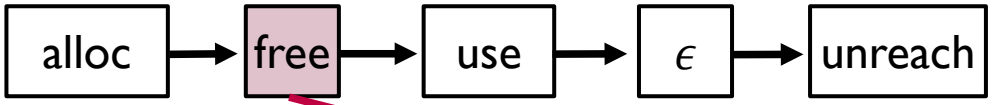
An object allocated at line 1 is unreachable after line 7



ICSE'20

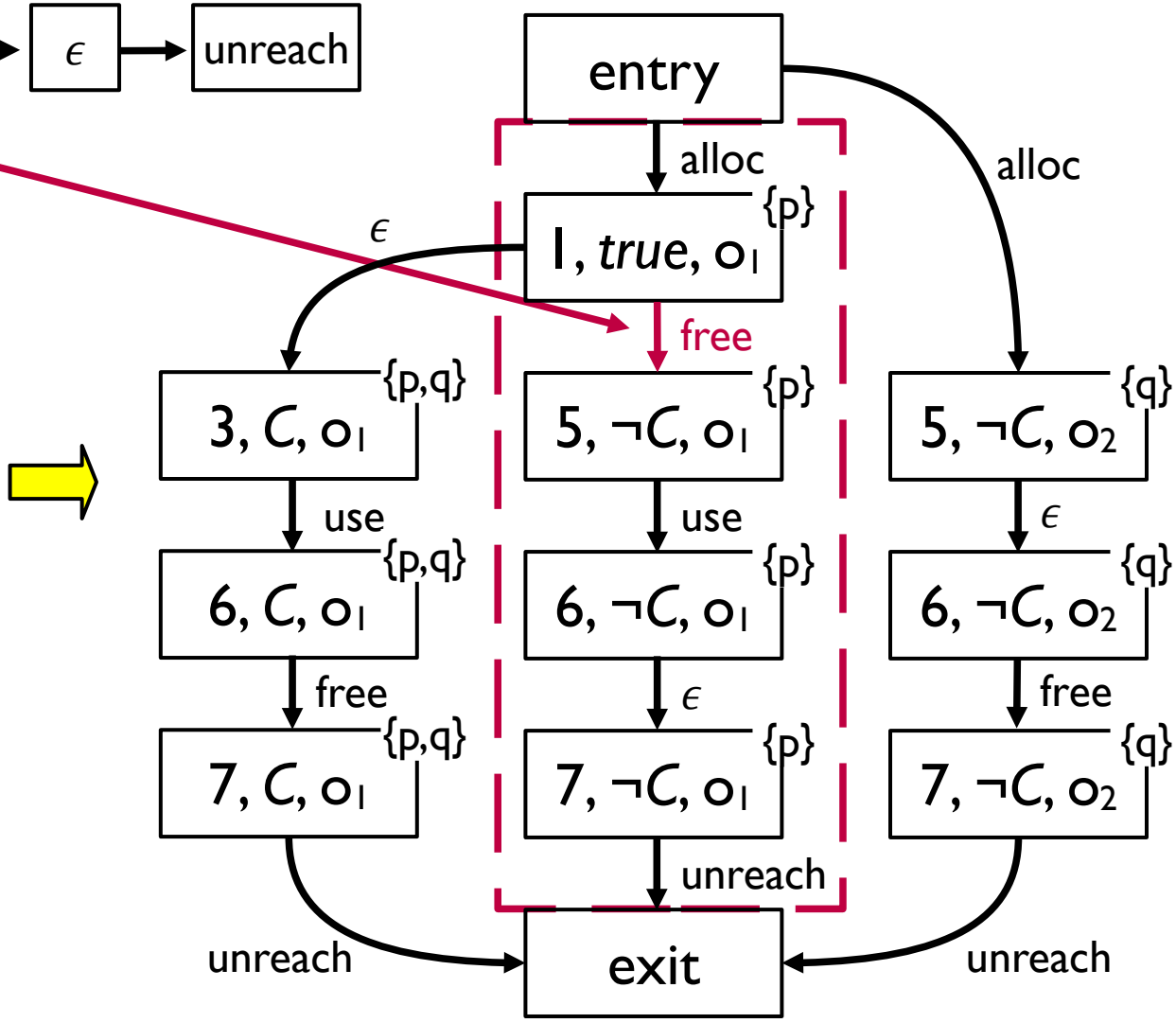
How Does SAVER Work?

- Step 3: Relabel the object flow graph.




```
1  p = malloc(); // o1
2  if (C)
3    q = p;
4  else
5    q = malloc(); // o2
6  *p = 1;
7  free(q);
```

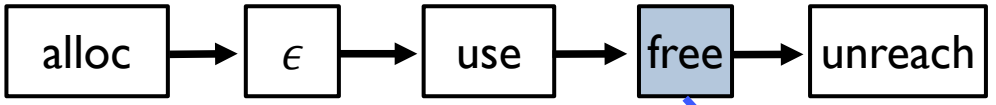
An object allocated at line 1 is unreachable after line 7



ICSE'20

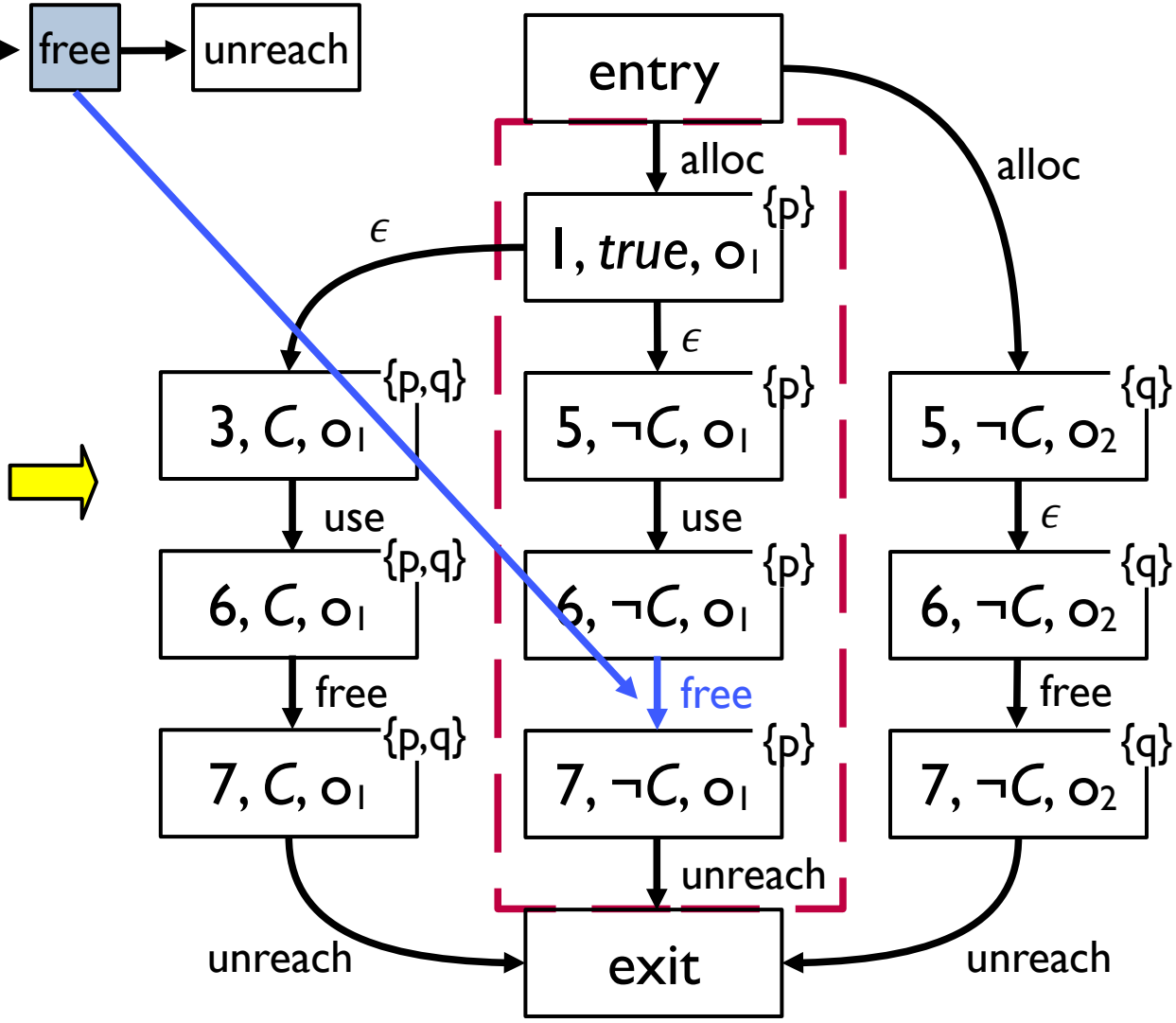
How Does SAVER Work?

- Step 3: Relabel the object flow graph.




```
1  p = malloc(); // o1
2  if (C)
3    q = p;
4  else
5    q = malloc(); // o2
6  *p = 1;
7  free(q);
```

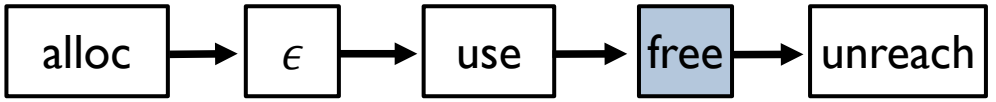
An object allocated at line 1 is unreachable after line 7



ICSE'20

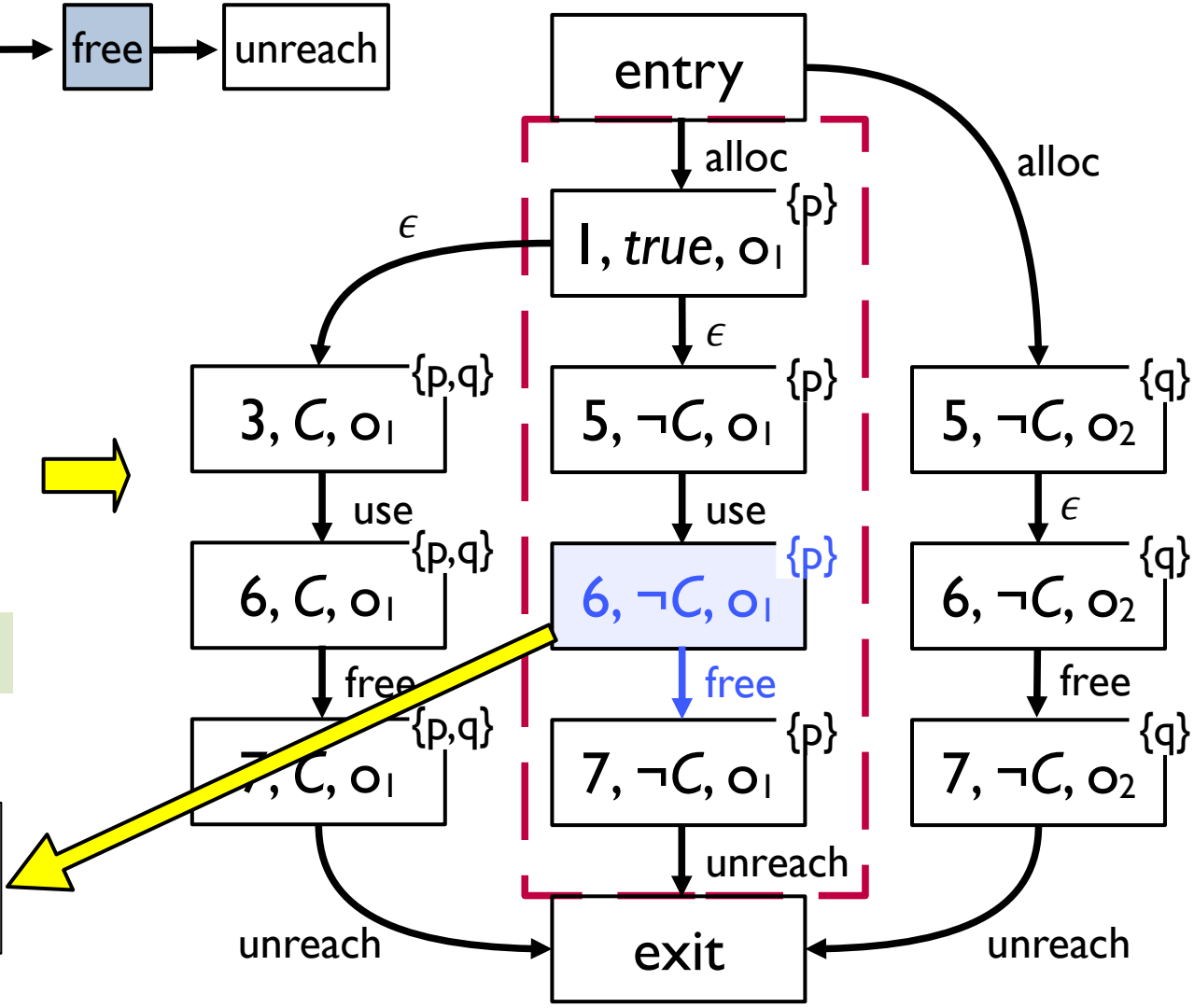
How Does SAVER Work?

- Step 4: Convert Labeling to a Patch.




```
1  p = malloc(); // o1
2  if (C)
3    q = p;
4  else
5    q = malloc(); // o2
6  *p = 1;
     if (!C) free(p);
7  free(q);
```

Insert:
If (!C) free(p) after line 6



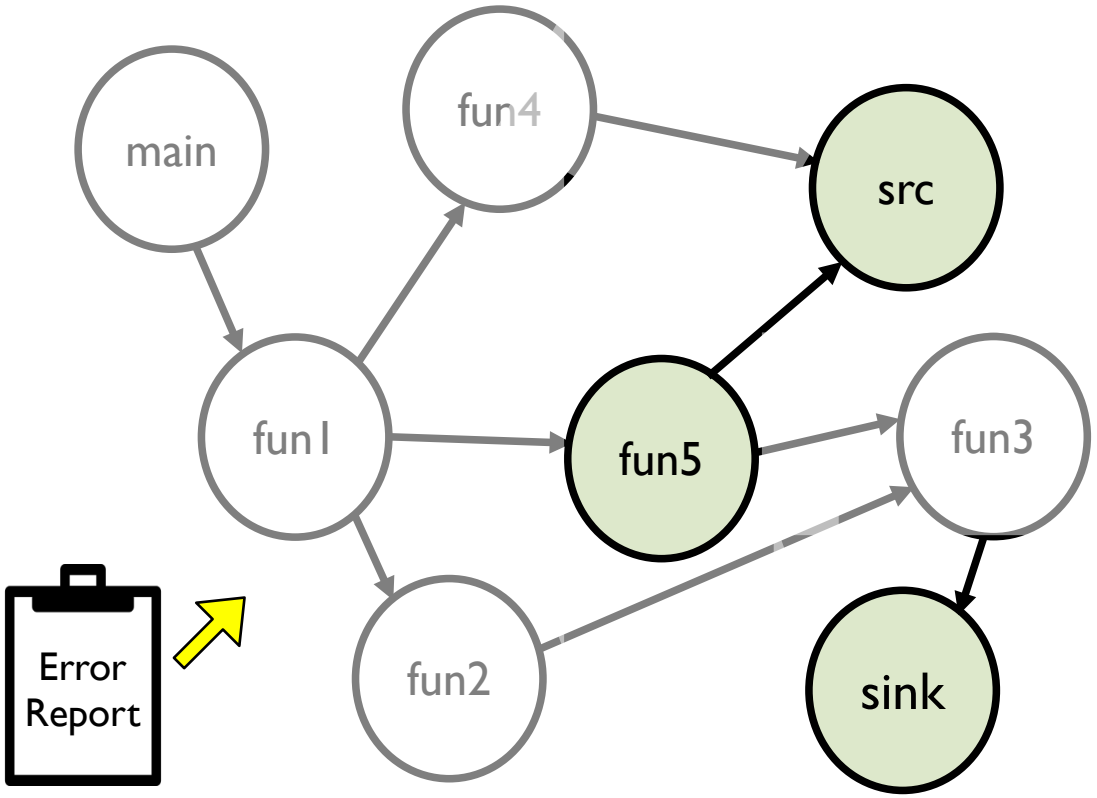
Challenge & Key Ideas

- Constructing OFG for real-world programs is costly.

ICSE'20

Challenge & Key Ideas

- Constructing OFG for real-world programs is costly.
- Two key Ideas
  1) Program-slicing heuristic

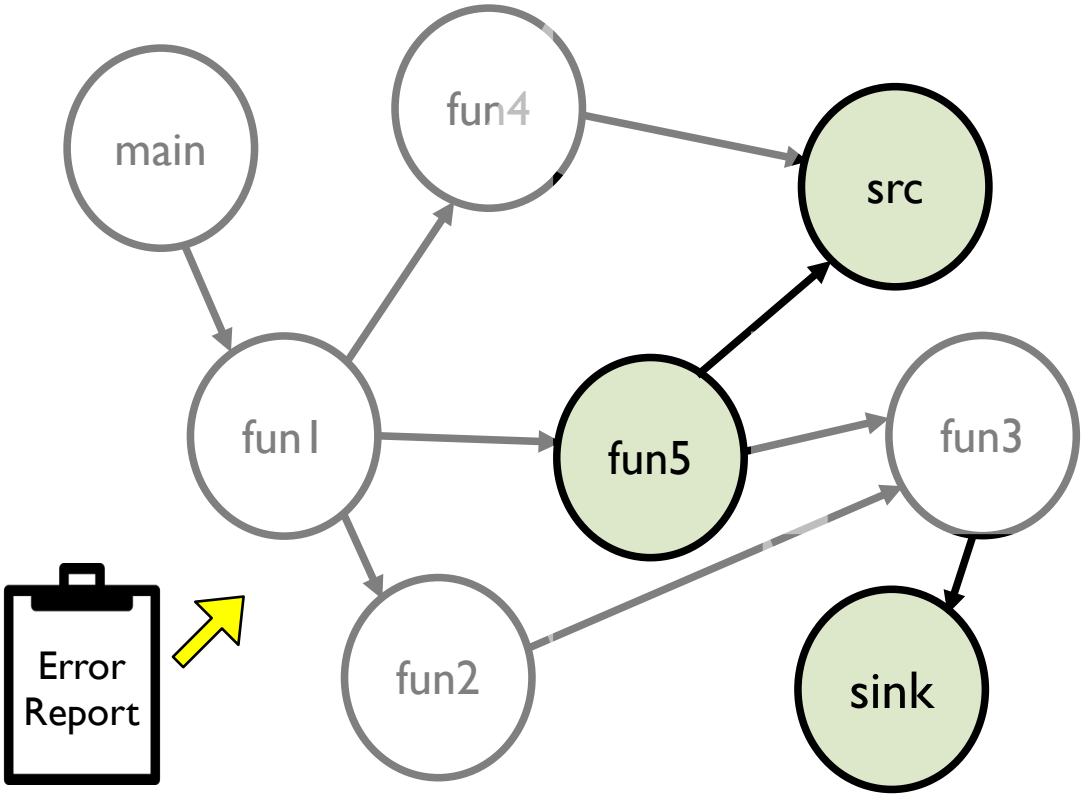


ICSE'20

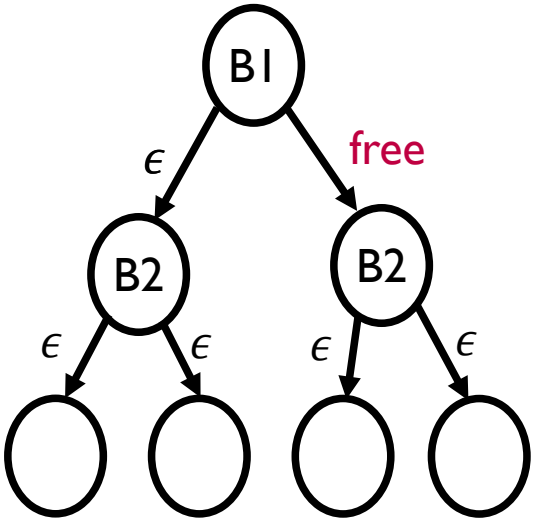
Challenge & Key Ideas

- Constructing OFG for real-world programs is costly.
- Two key Ideas

1) Program-slicing heuristic



2) Path-merging heuristic

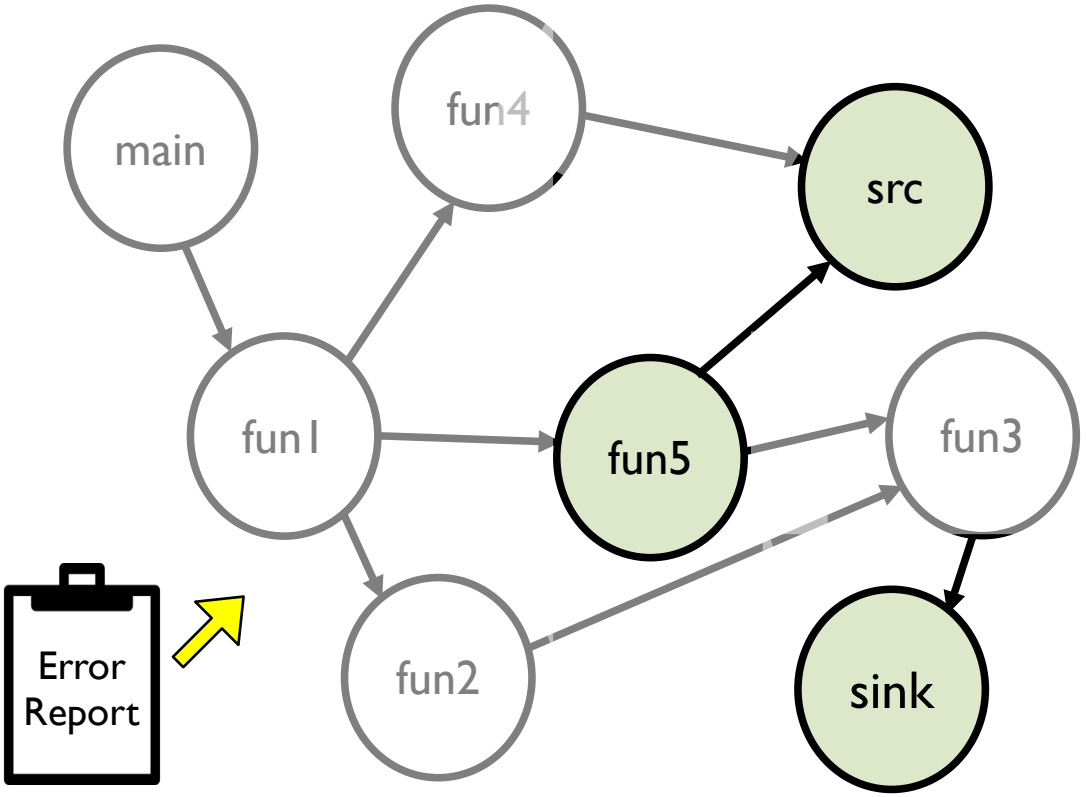


ICSE'20

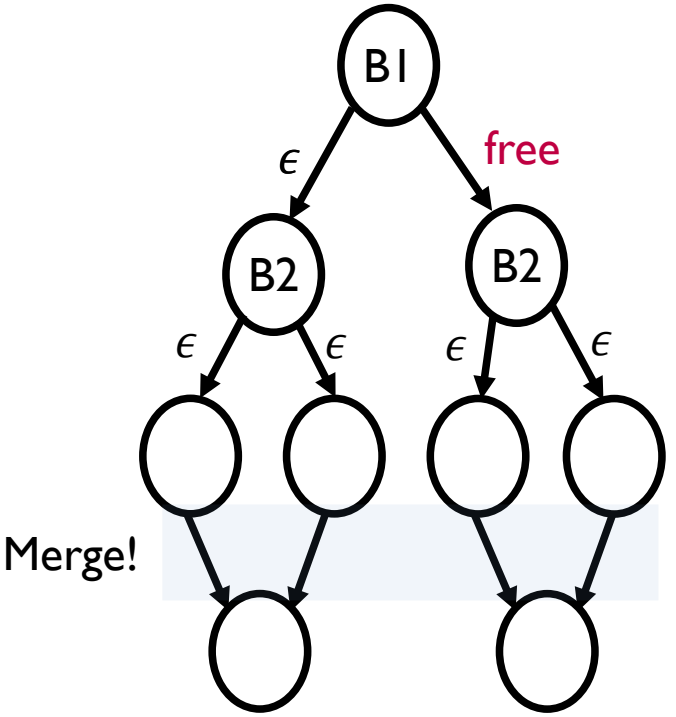
Challenge & Key Ideas

- Constructing OFG for real-world programs is costly.
- Two key Ideas

1) Program-slicing heuristic



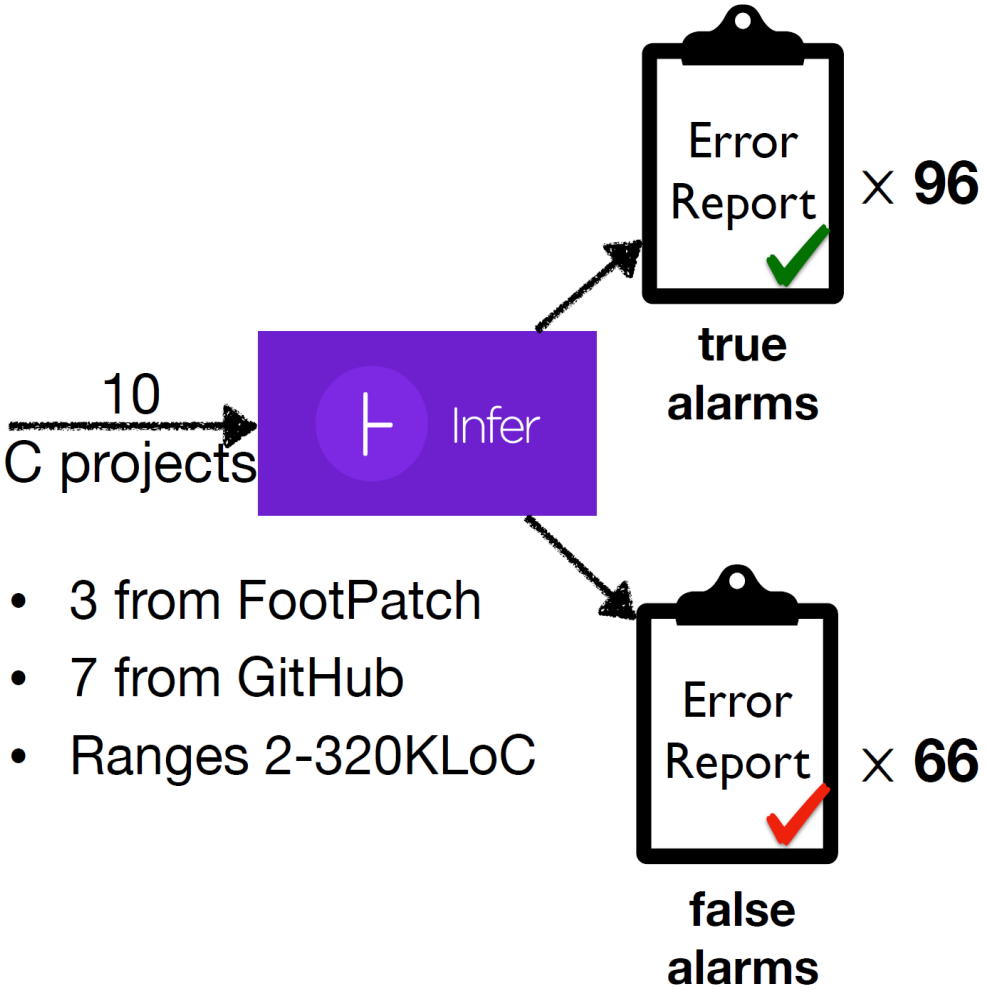
2) Path-merging heuristic



ICSE'20

Effectiveness of SAVER

- Evaluation on Memory Leak (vs FootPatch)

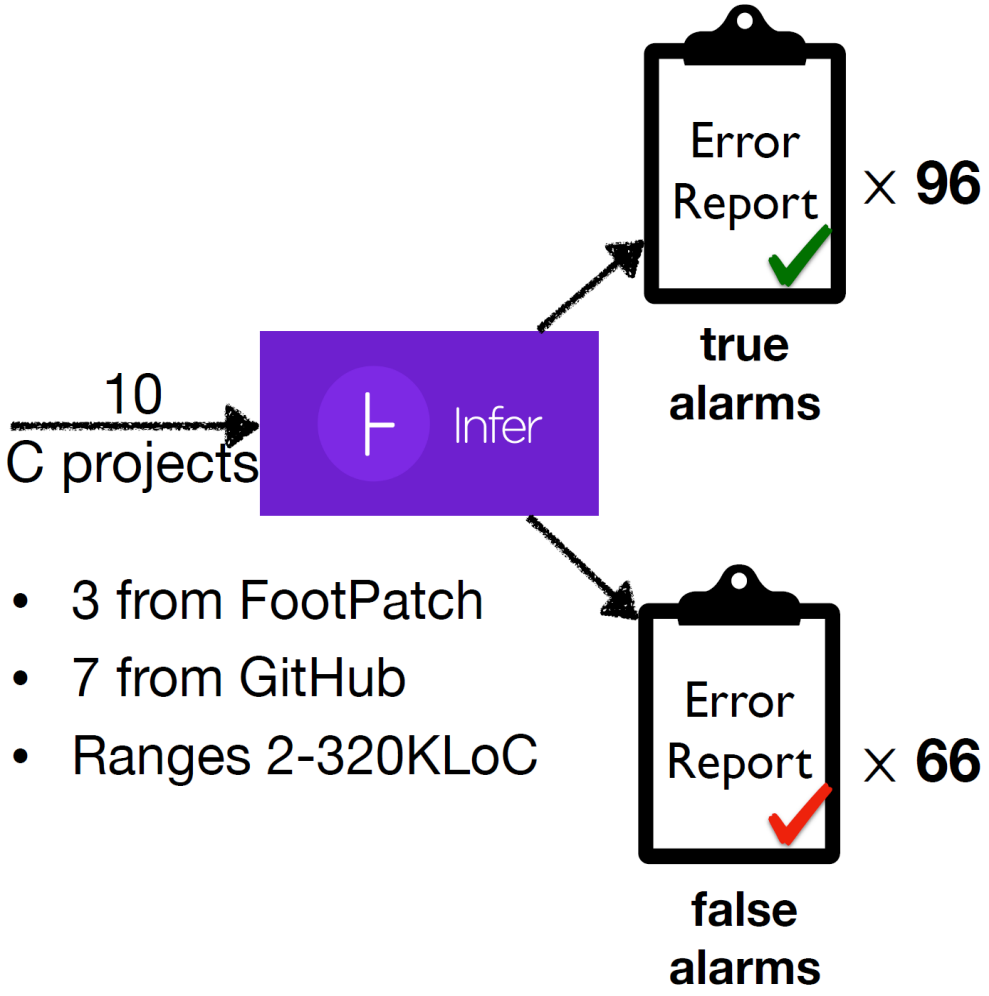


- 3 from FootPatch
- 7 from GitHub
- Ranges 2-320KLoC

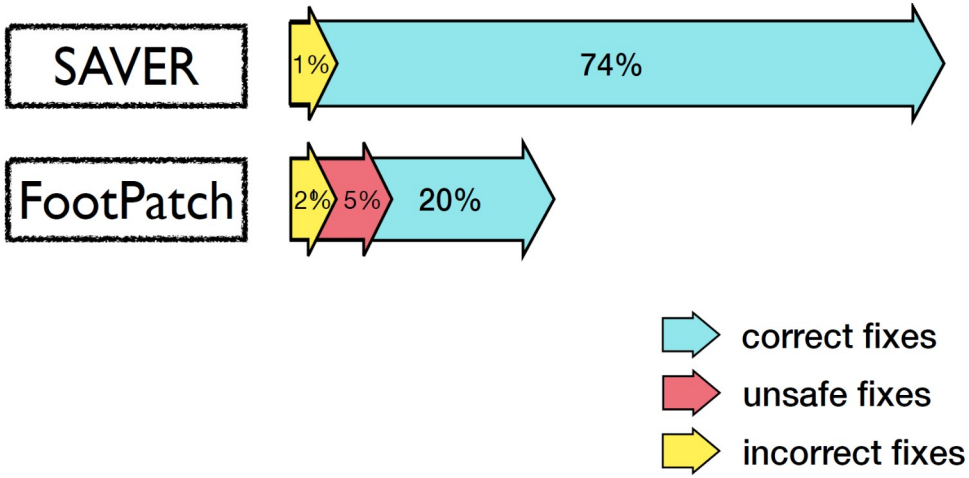
ICSE'20

Effectiveness of SAVER

- Evaluation on Memory Leak (vs FootPatch)



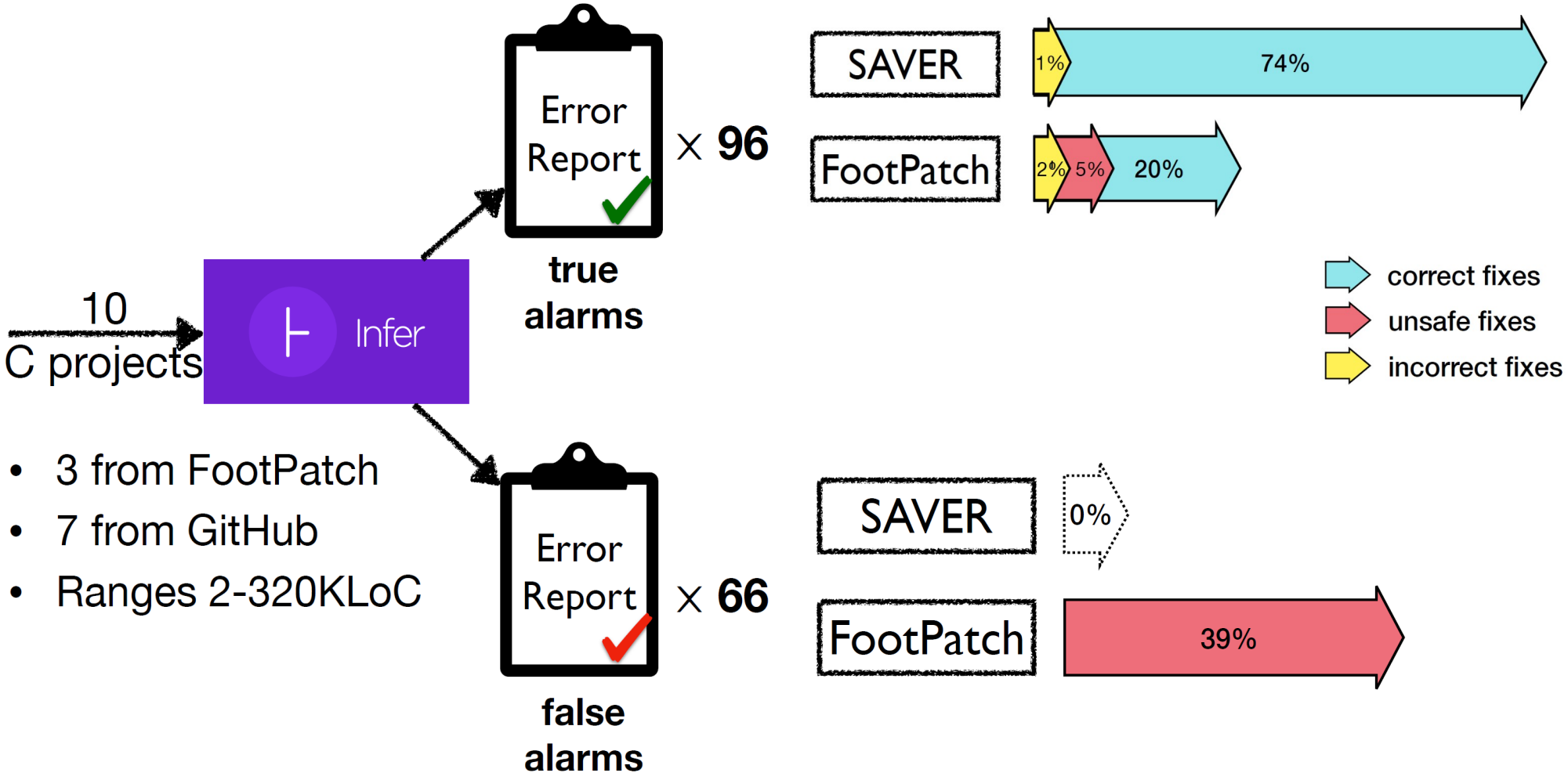
- 3 from FootPatch
- 7 from GitHub
- Ranges 2-320KLoC



ICSE'20

Effectiveness of SAVER

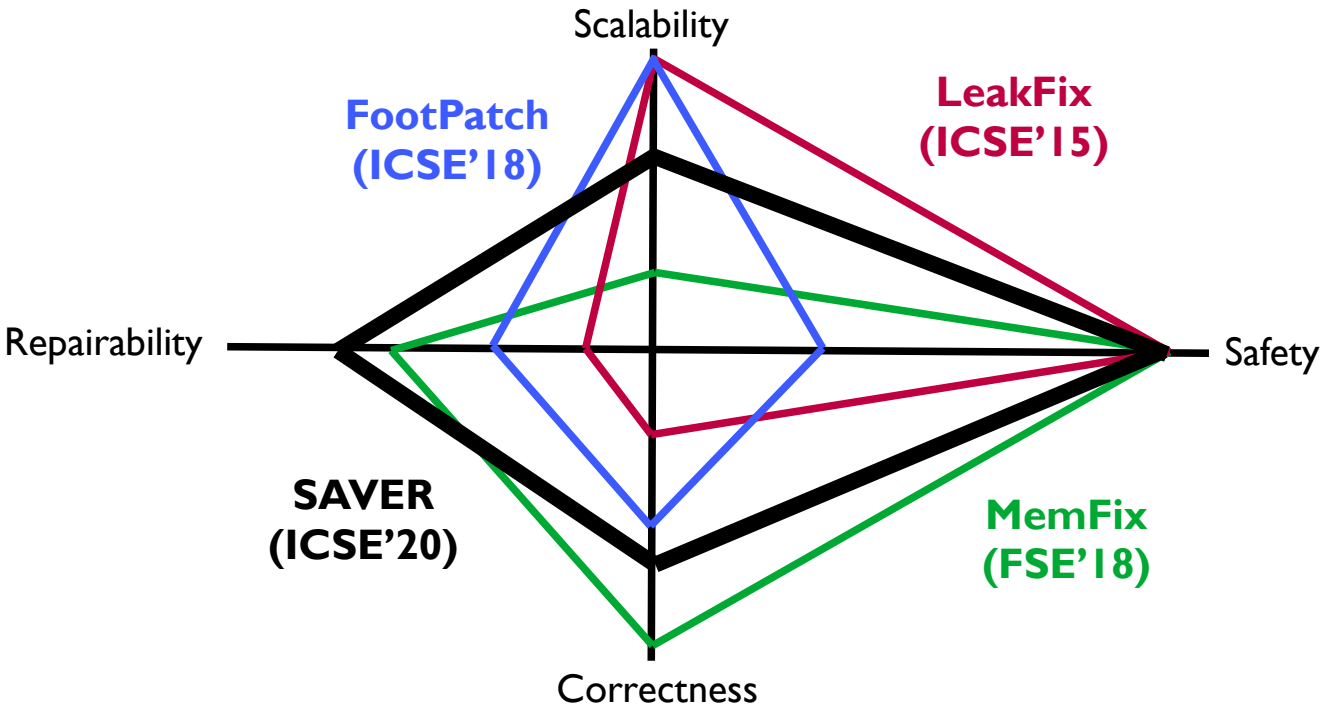
- Evaluation on Memory Leak (vs FootPatch)



- 3 from FootPatch
- 7 from GitHub
- Ranges 2-320KLoC

Summary

- Automatic Program Repair (APR)
  - Why do we need APR techniques?
  - Special-Purpose Program Repair



Thank You