Understanding and Generating High Quality Patches for Concurrency Bugs

Haopeng Liu, Yuxi Chen and Shan Lu

THE UNIVERSITY OF CHICAGO

What are concurrency bugs

- Synchronization mistakes in multi-threaded programs

What are concurrency bugs

- Synchronization mistakes in multi-threaded programs

```
//child thread                    //parent thread
if(…){
  unlock(f->mut);
  return;
}                                 f->mut = NULL;
```

What are concurrency bugs

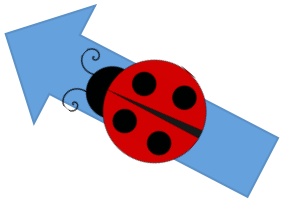
- Synchronization mistakes in multi-threaded programs

```
//child thread                    //parent thread
if(…){
  unlock(f->mut);
  return;
}                                 f->mut = NULL;
```

What are concurrency bugs

- Synchronization mistakes in multi-threaded programs

```
//child thread                    //parent thread
if(…){
  unlock(f->mut);
  return;
}                                 f->mut = NULL;
```



Concurrency bugs need to be fixed

- Prevalence
  - Widely exist in multi-threaded programs
- Seriousness
  - Cause real world disasters

- Difficulty
  - Time-consuming to fix [LuASPLOS'08]
  - Patches are the most error-prone [YinFSE'11]
    - 2X times more often to be buggy than memory/semantic bugs

Concurrency bug automated fixing

- Observation:
  - Program is correct in most timing interleavings.
- Solution:
  - Only need to remove some bad timing.
    - add lock-related synchronization [AFix PLDI'11] [CFix OSDI'12] [Grail FSE'14].

An automated fixing example

```
  //child thread
  if(…){
    unlock(f->mut);
+   lock(L);
+   signal(con);
+   unlock(L);
    return;
  }
+ lock(L);
+ signal(con);
+ unlock(L);
```

```
  //parent thread
+ lock(L)
+ while(…){
+   wait(con, L);
+ }
+ unlock (L)
  f->mut = NULL;
```

- Overall:
  - 9 new synchronizations
  - 1 new while loop

Reality is even worse!

The manual patch

```
//child thread
if(…){
  unlock(f->mut);
  return;
}
```

```
  //parent thread
+ thread_join(…);
  f->mut = NULL;
```

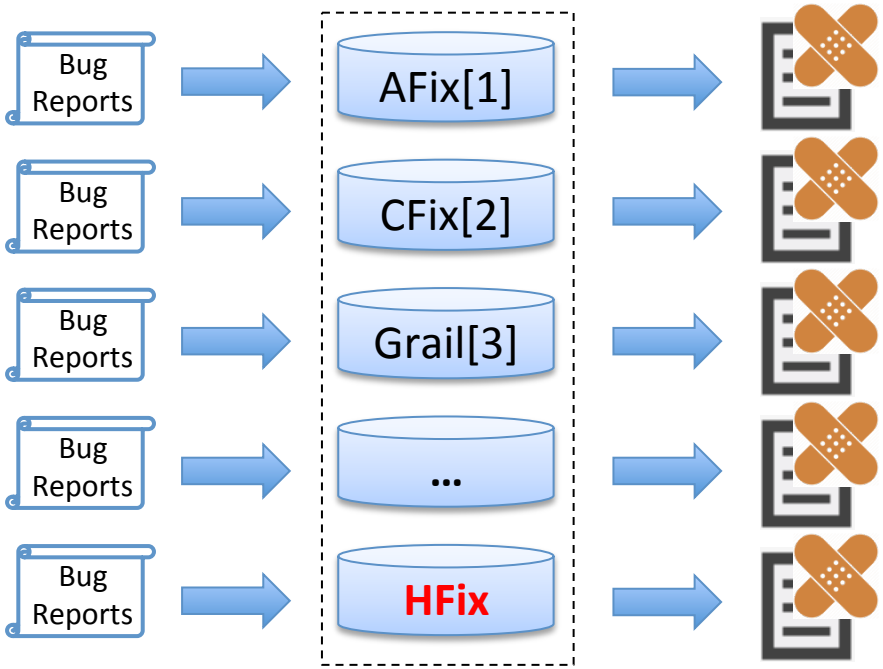
- Overall:
  - New sycn.: 1 VS. 9
  - New while loop: 0 VS. 1

Simpler patch → Easier to read and maintain

Can we automatically generate such simple patches?

Contributions

- Manual patch study
  - 77 real-world concurrency bugs
  - The limitation of existing techniques
  - Guidance to design new tools
- HFix



[1] Guoliang Jin etc. AFix. In PLDI'11
[2] Guoliang Jin etc. CFix. In OSDI'12
[3] Peng Liu etc. Grail. In FSE'14

Outline

- Motivation
- Manual patch study
- HFix
- Evaluation
- Conclusion

Research Questions

- Q1: Do manual patches mostly use lock-related synchronizations?
- Q2: Do manual patches mostly disable buggy timing by adding synchronization ops?

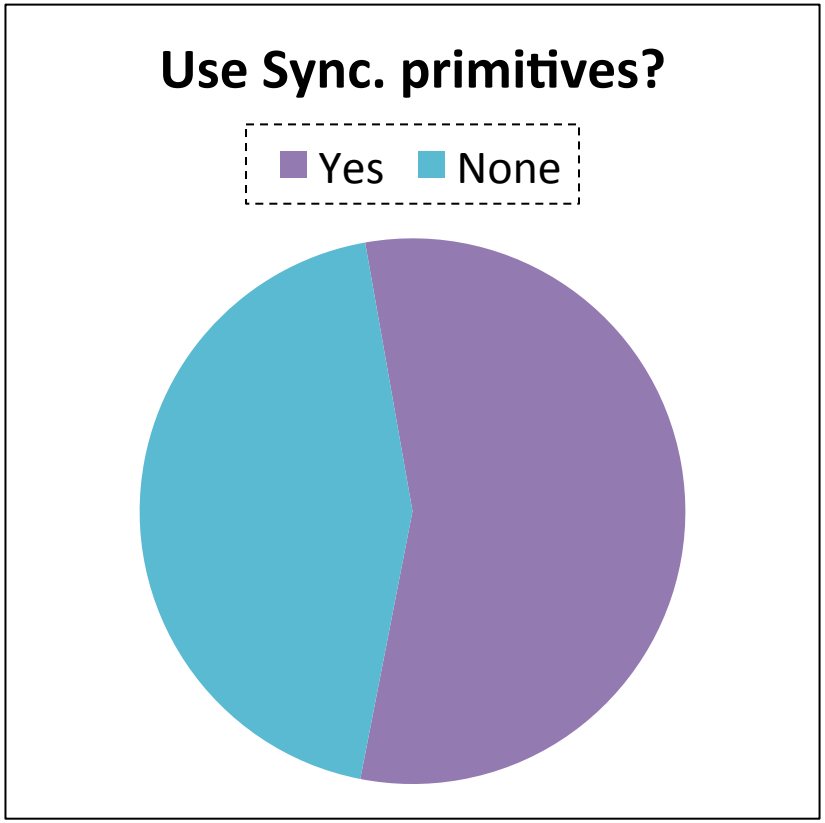
Manual patch study methodology[1]

App.	Source	Description	#. Bugs
Apache	Lu ASPLOS'08	Web Server	13
Mozilla		Browser Suite	41
MySQL		Database Server	12
OpenOffice		Office Suite	5
Misc	Recent papers[2-4]		6
Total			77

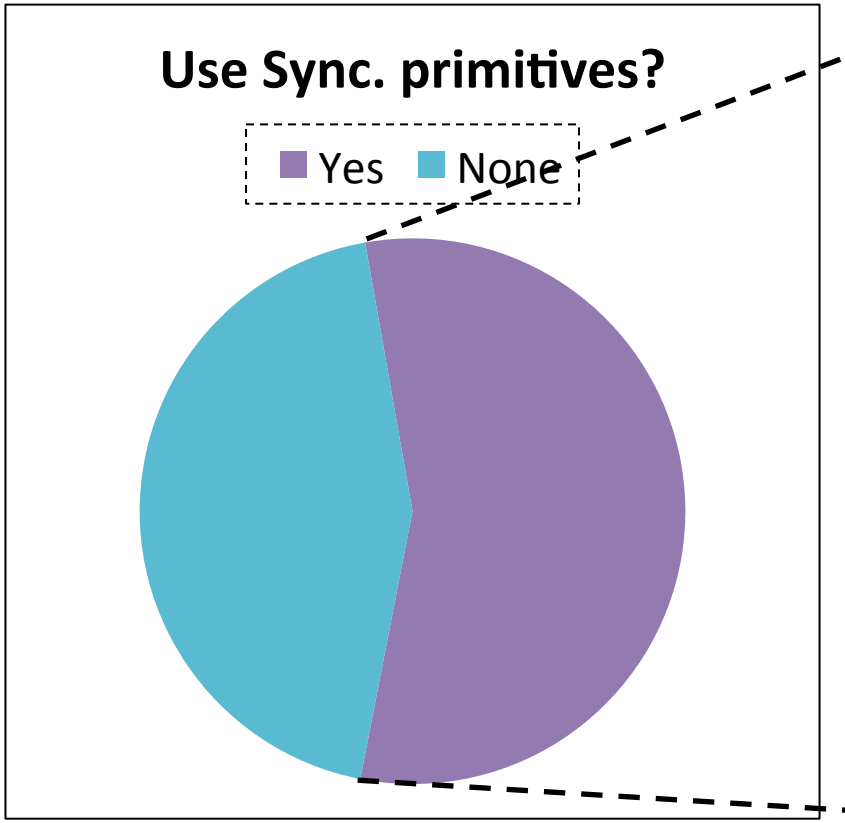
- Study developer patches & on-line discussion

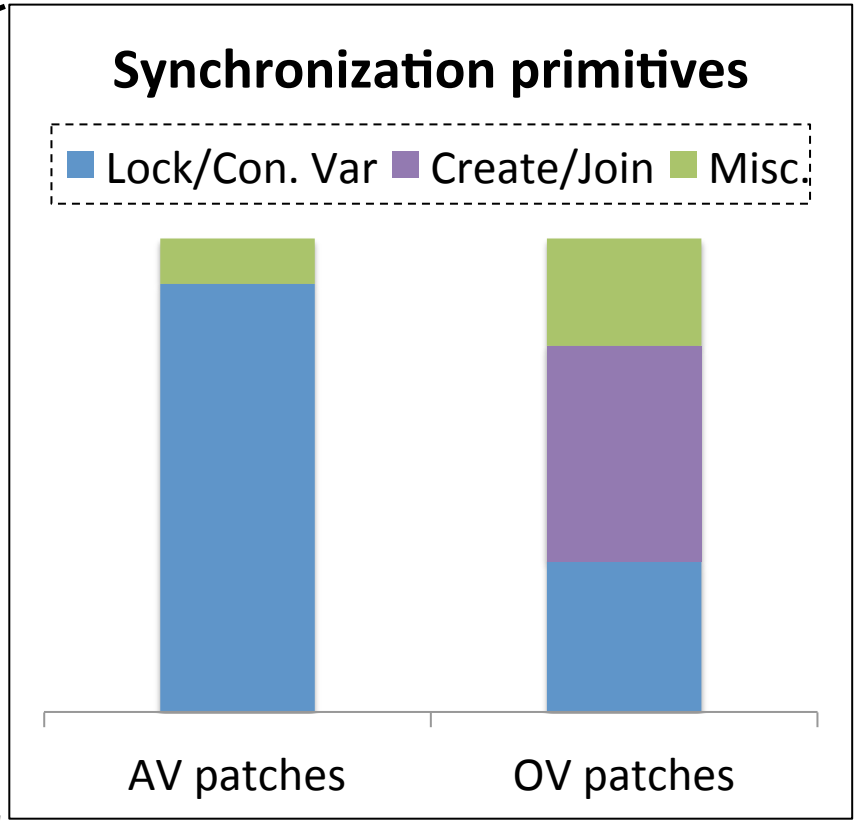
[1] Threads to validity are listed in paper.
[2] Guoliang Jin etc. AFix. In PLDI'11
[3] Guoliang Jin etc. CFix. In OSDI'12
[4] Wei Zhang etc. ConSeq. In ASPLOS'11

Q1: How many patches use lock?

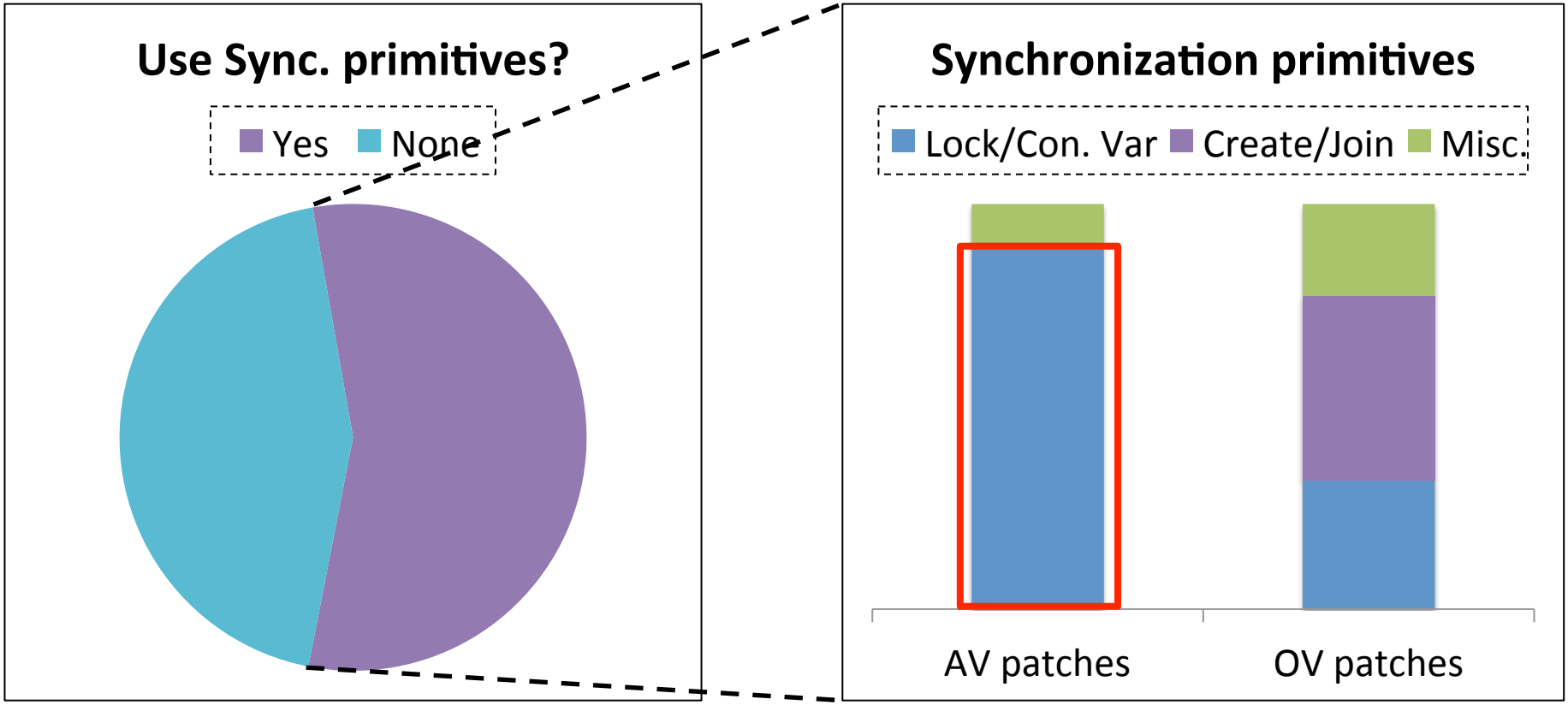


Q1: How many patches use lock?

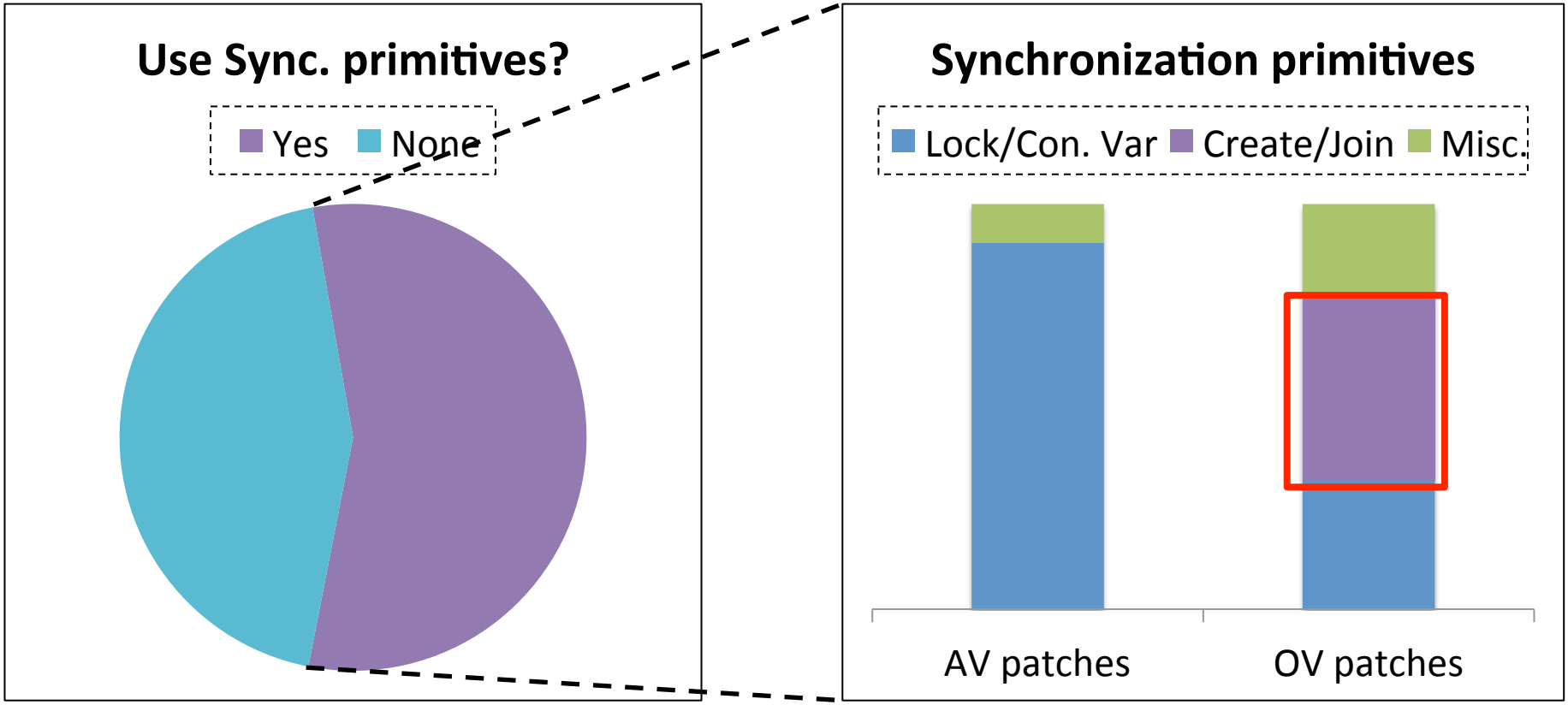




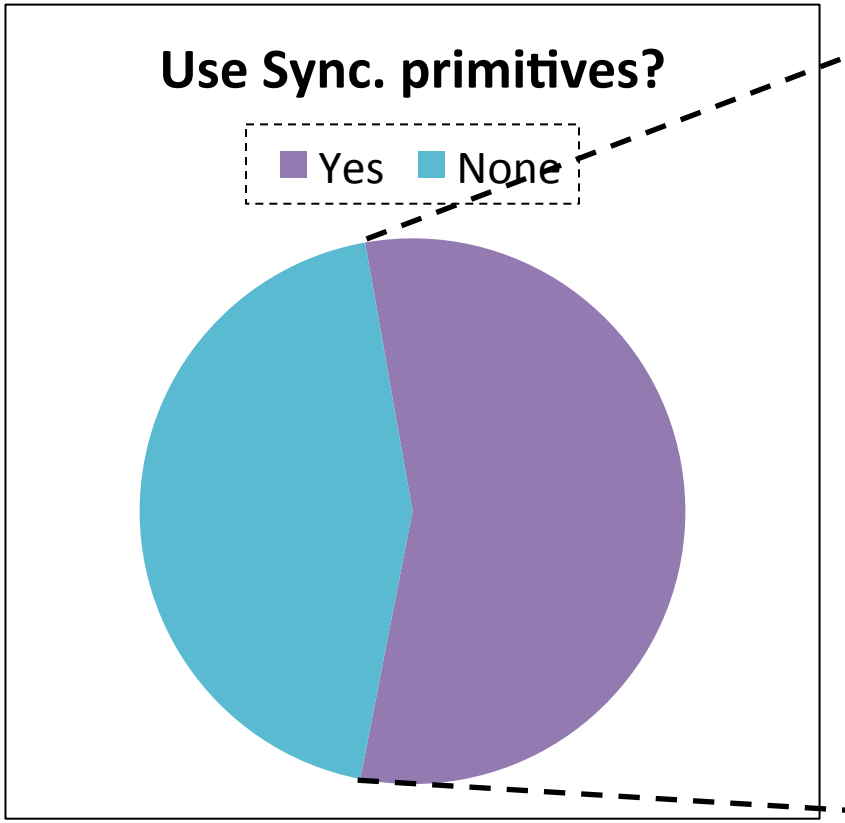
Q1: How many patches use lock?

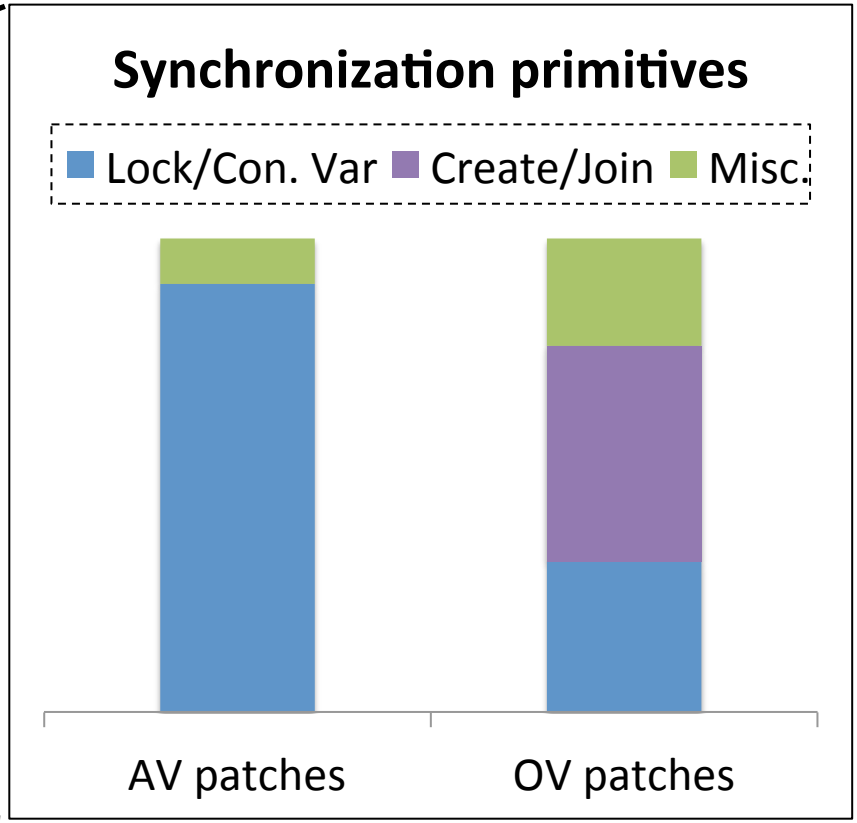


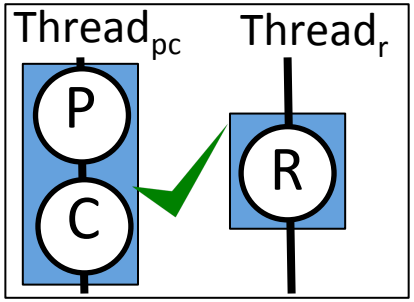
Q1: How many patches use lock?



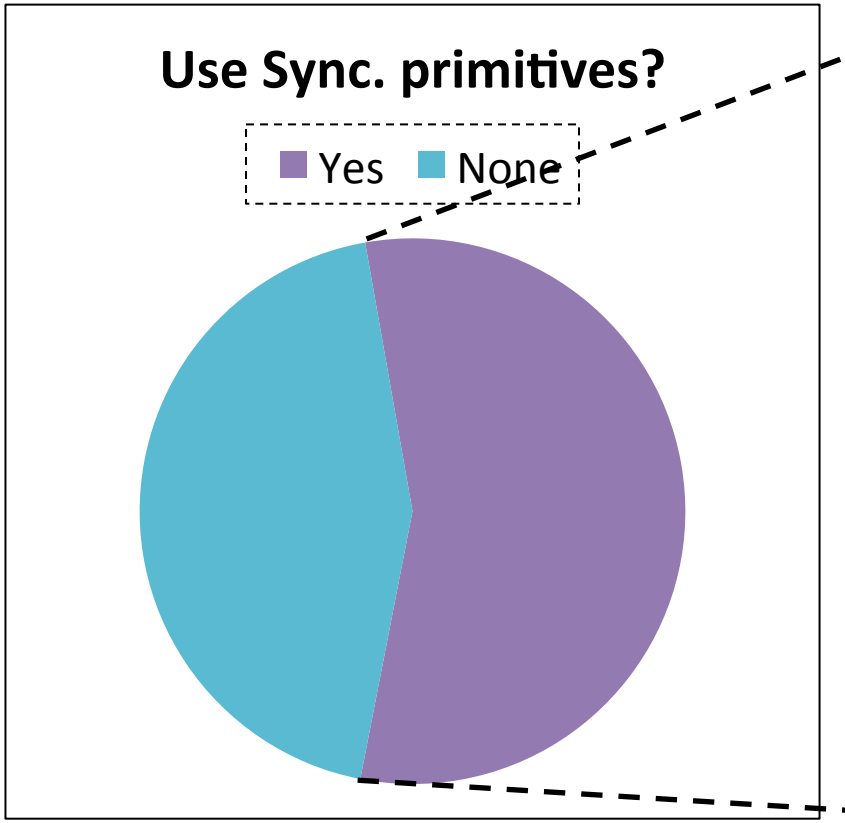
Q1: How many patches use lock?

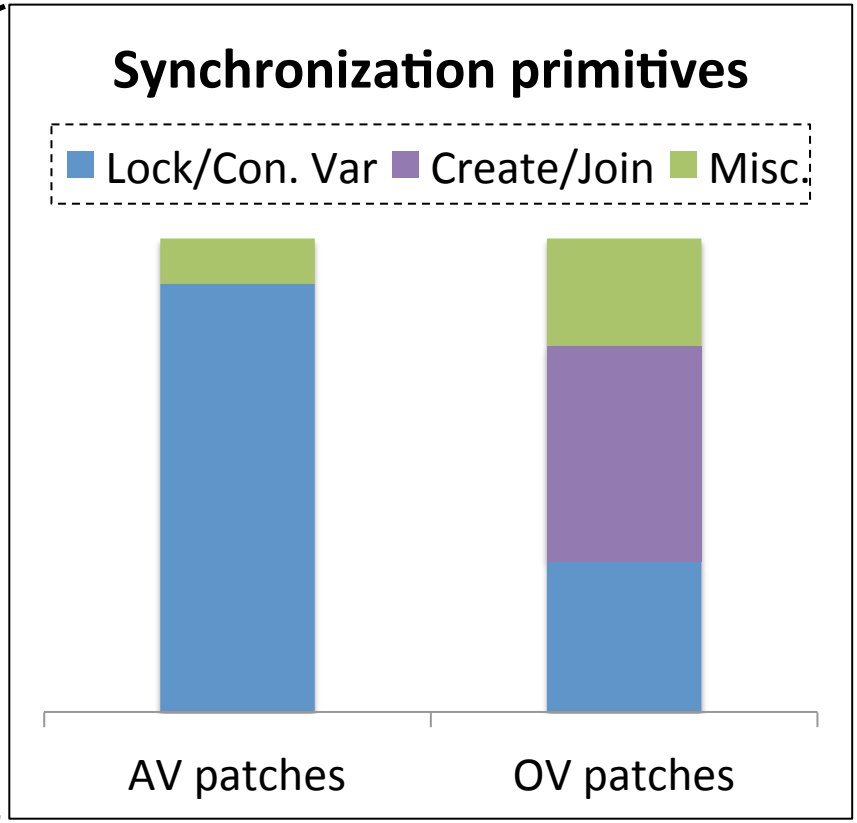


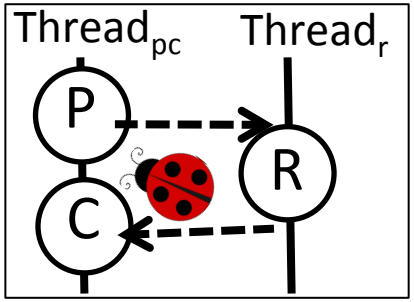




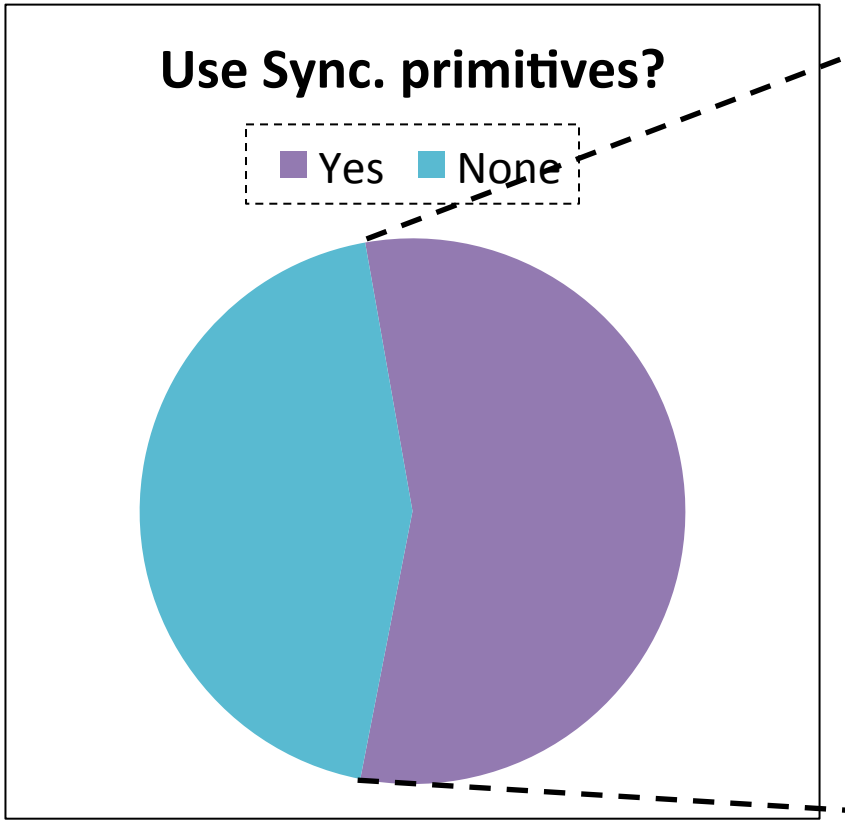
Q1: How many patches use lock?

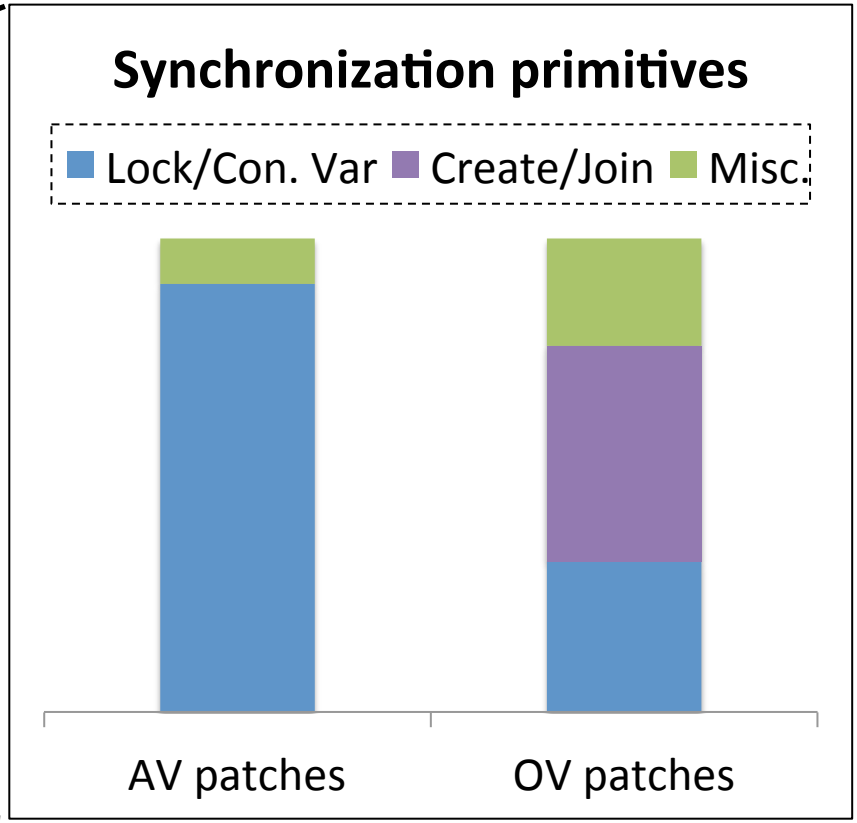


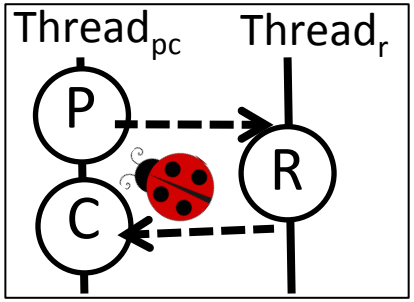


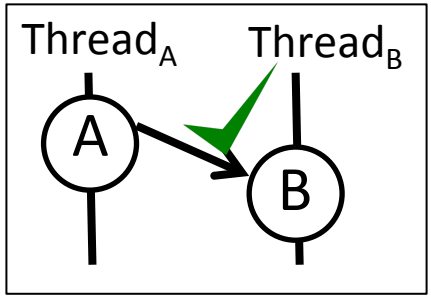


Q1: How many patches use lock?

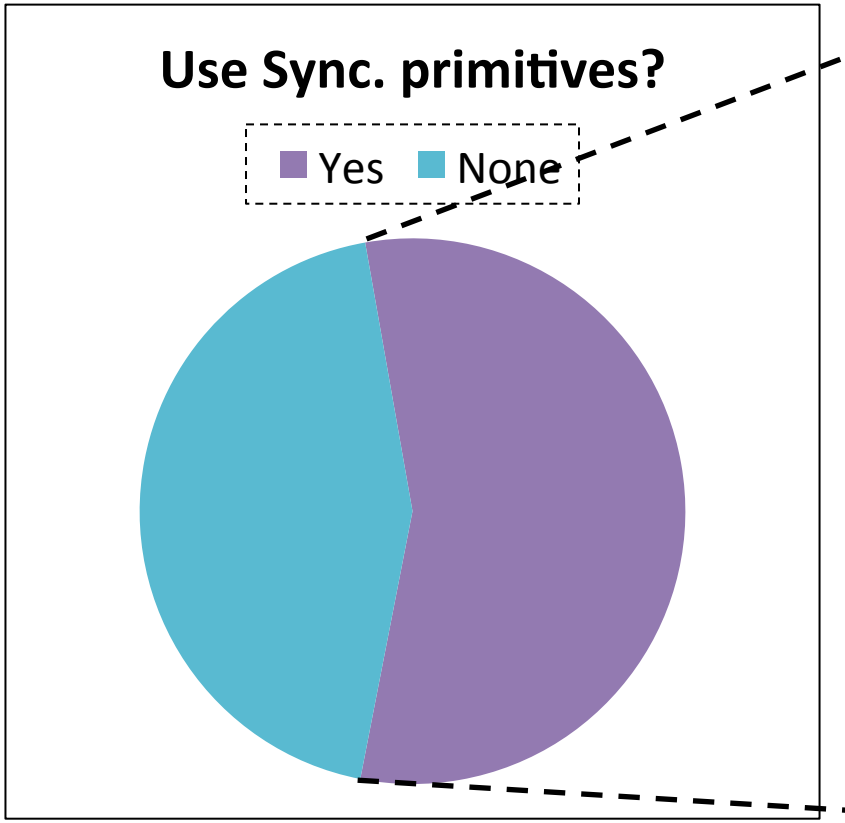


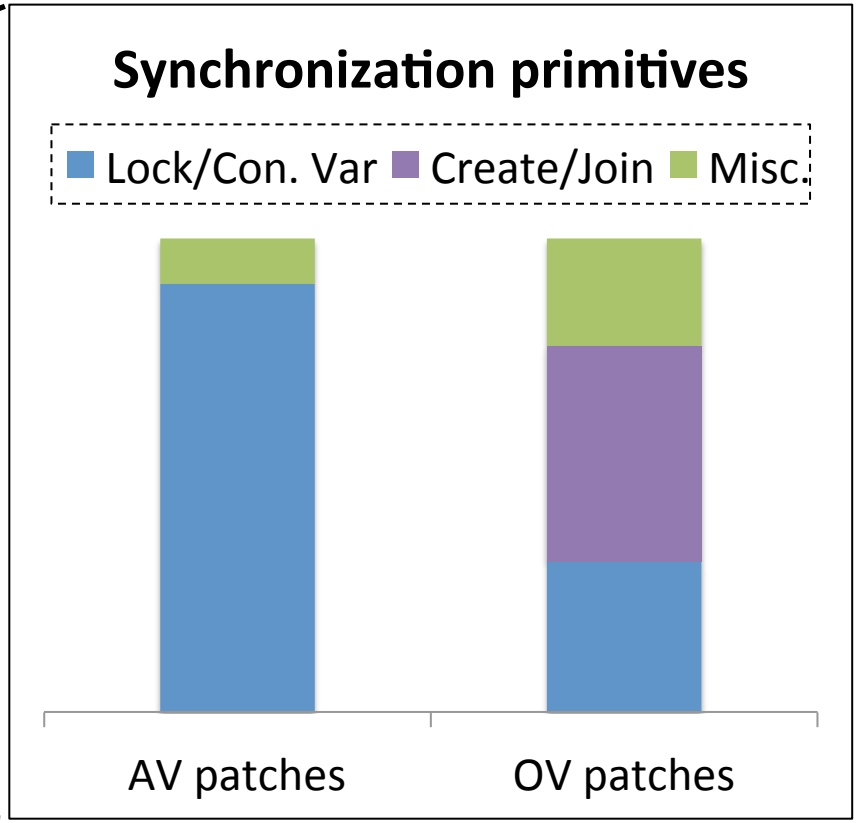


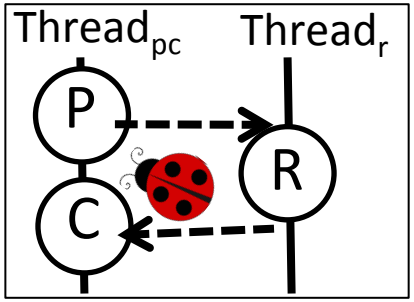


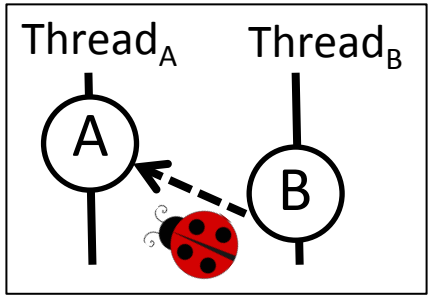


Q1: How many patches use lock?



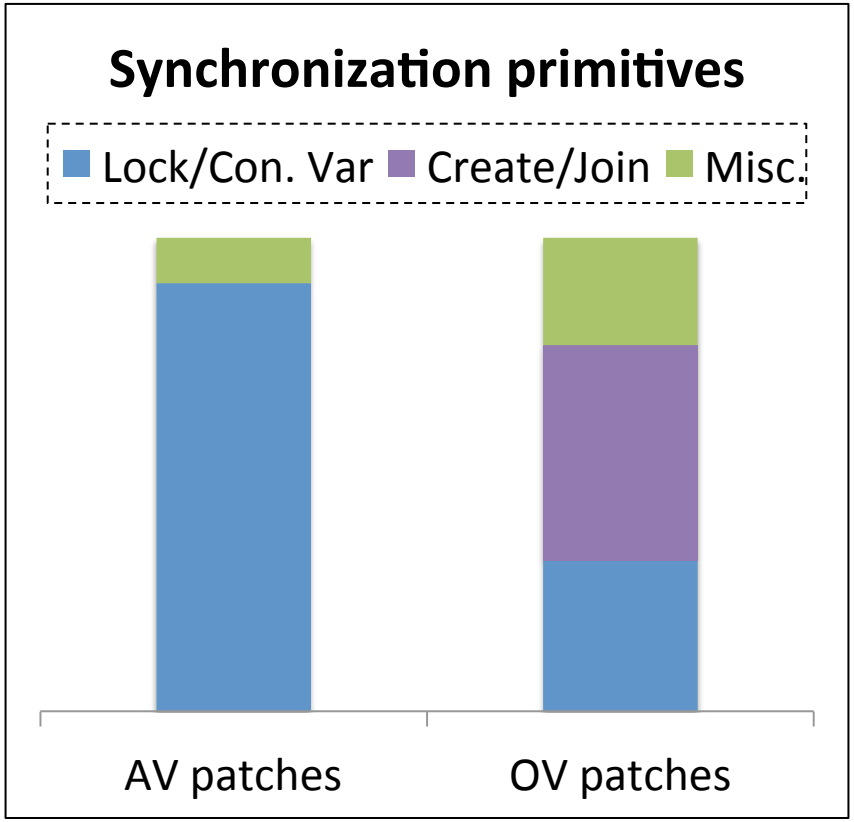






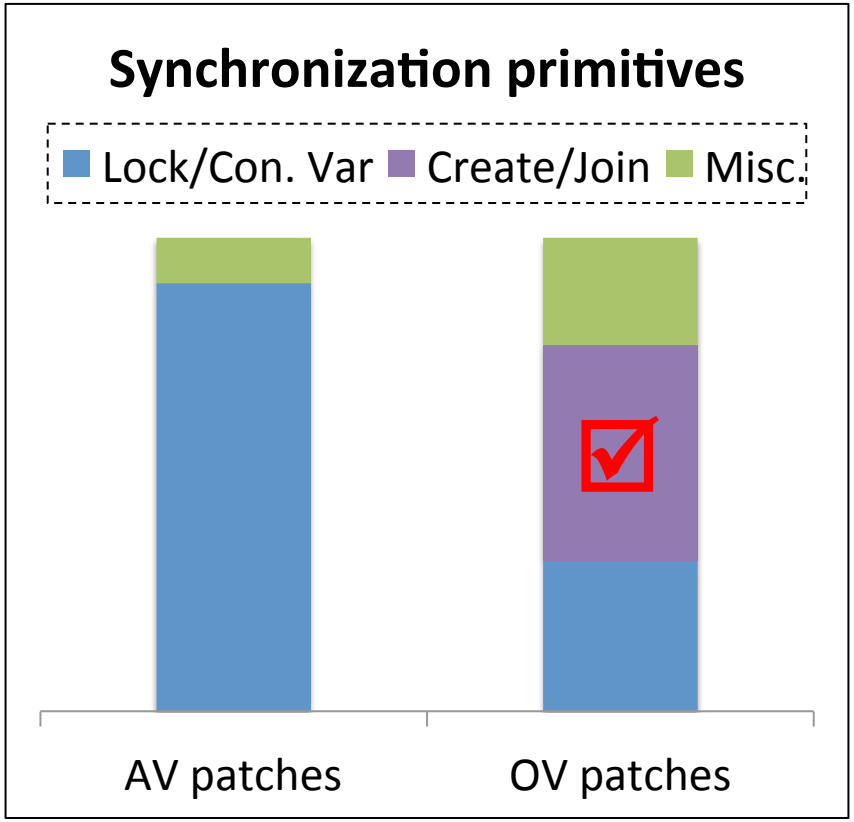
Q1 implication

- Use non-lock synchronization to fix OV.

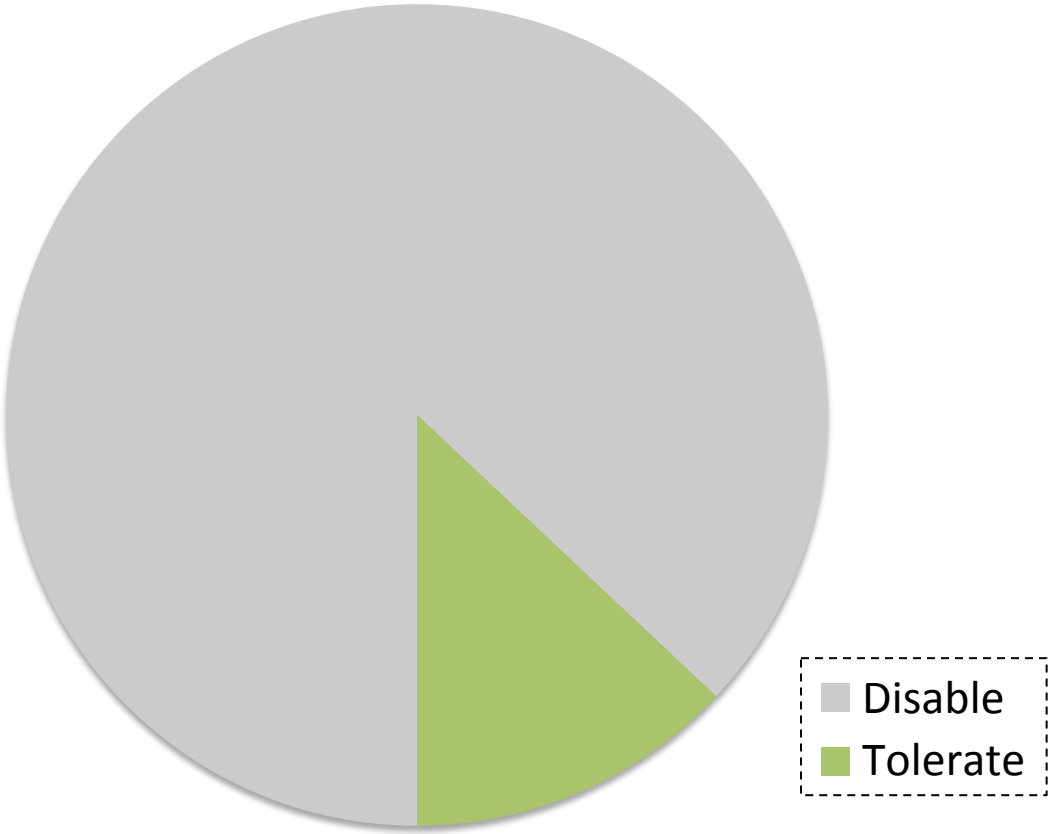


Q1 implication

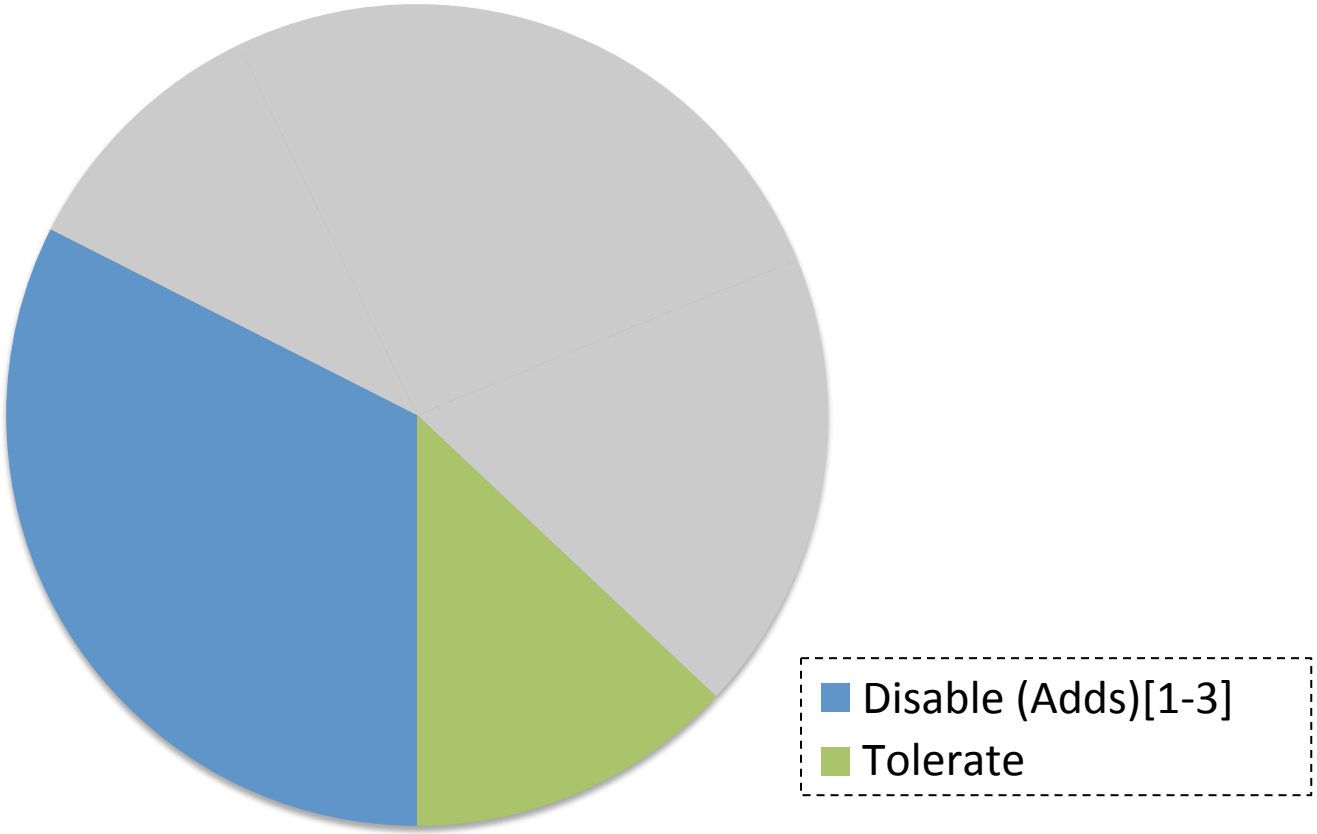
- Use non-lock synchronization to fix OV.



Q2: What are developers' fix strategy?

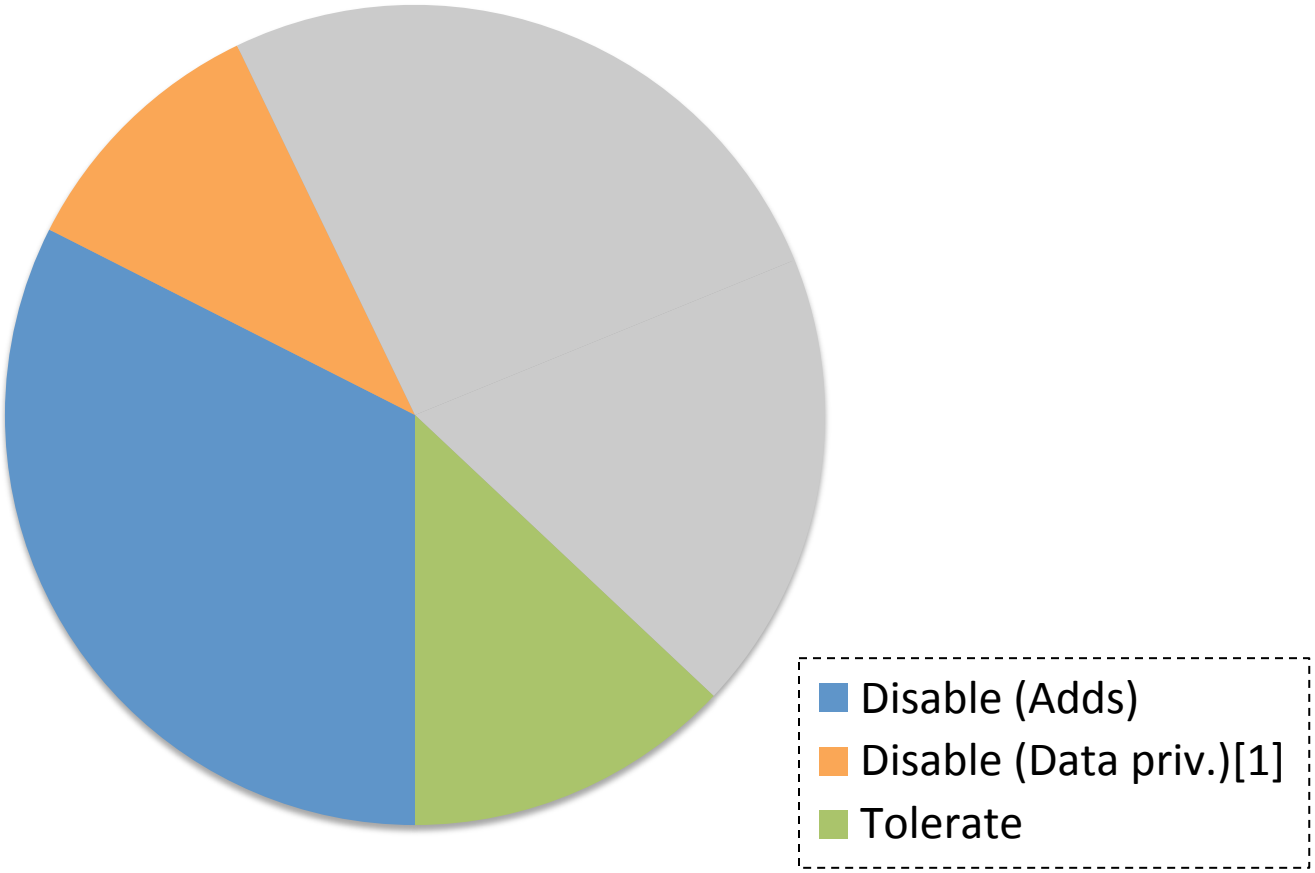


Q2: What are developers' fix strategy?



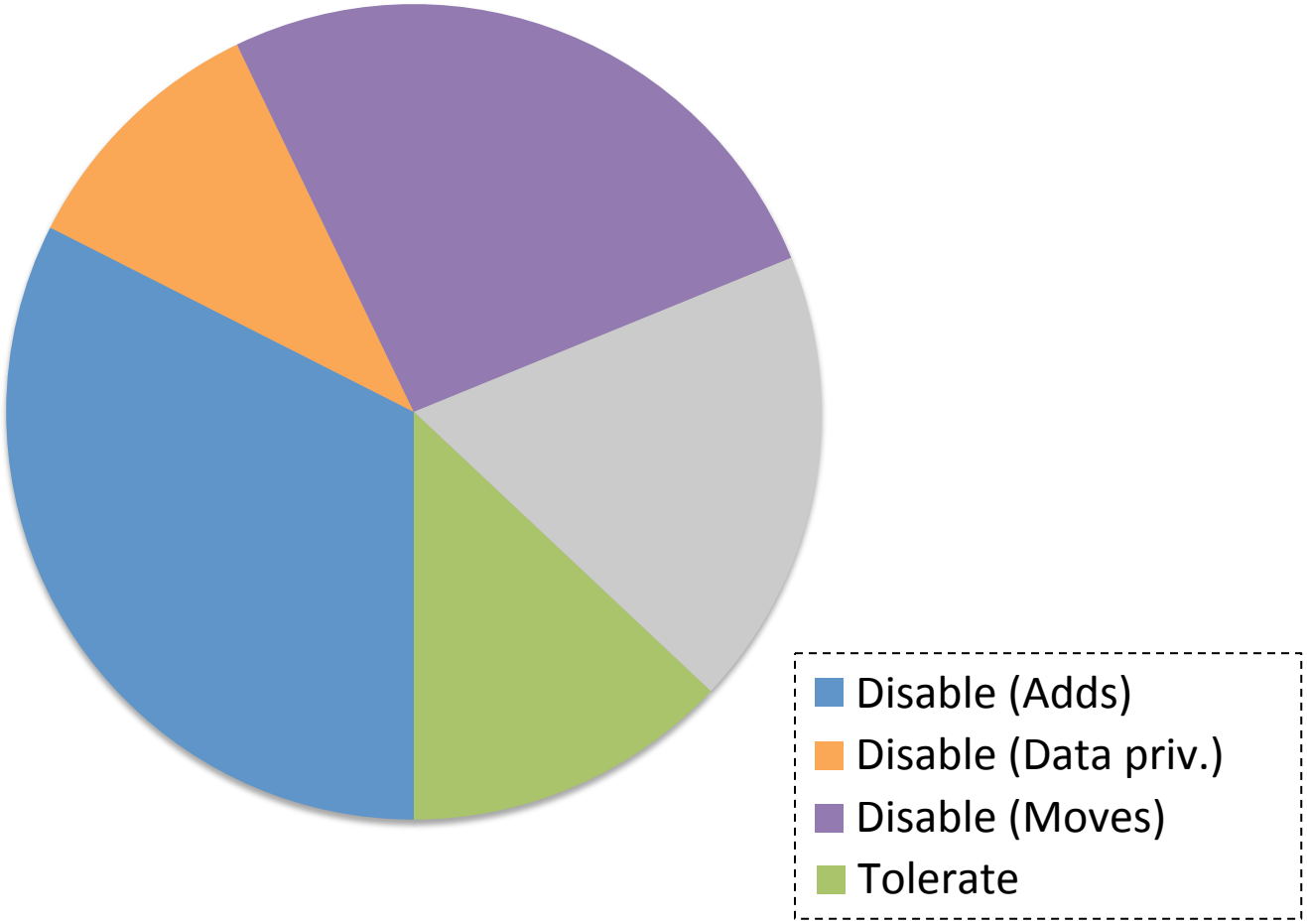
[1] Guoliang Jin etc. AFix. In PLDI'11
[2] Guoliang Jin etc. CFix. In OSDI'12
[3] Peng Liu etc. Grail. In FSE'14

Q2: What are developers' fix strategy?

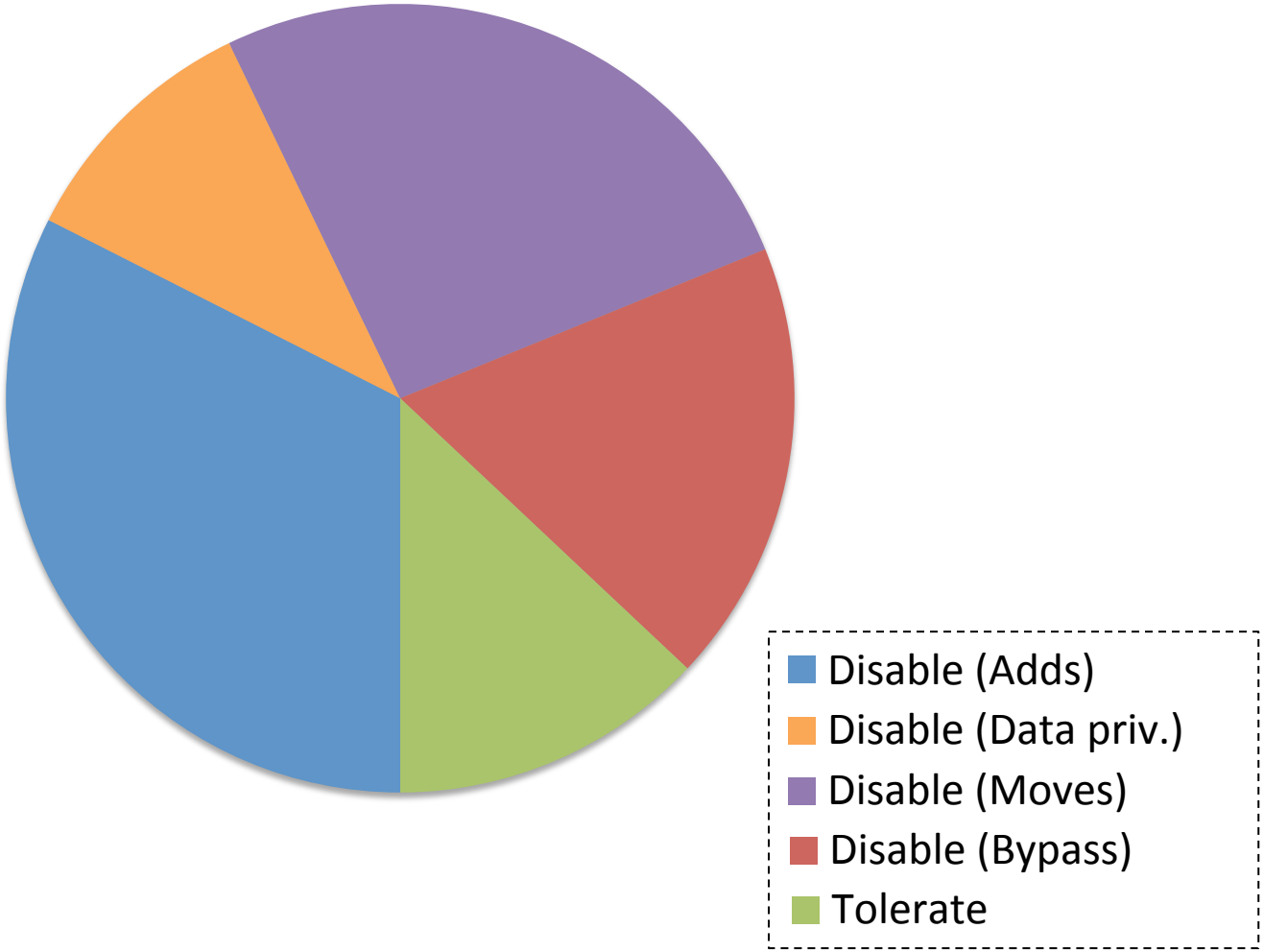


[1] Jeff Huang etc. Execution Privatization of Scheduler-Oblivious Concurrent Programs. In OOPSLA'11

Q2: What are developers' fix strategy?

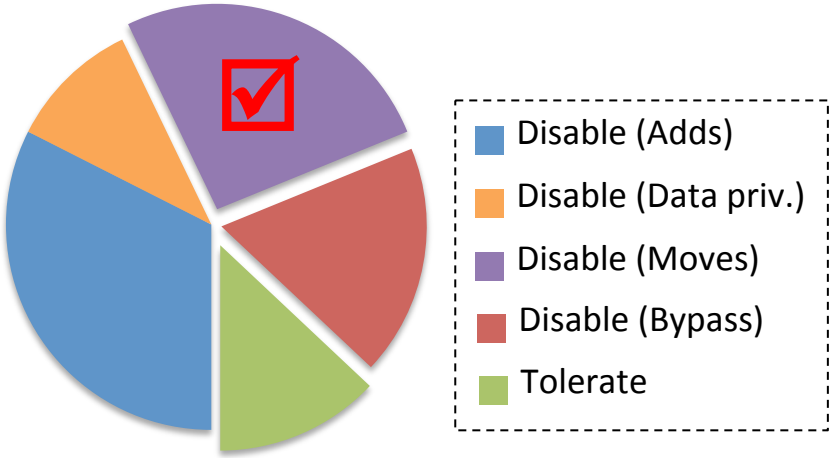


Q2: What are developers' fix strategy?



Q2 implication

- Add-Sync widely applies, but often not best
- Need to automate:
  - Move synchronization
  - Bypass
  - Tolerate

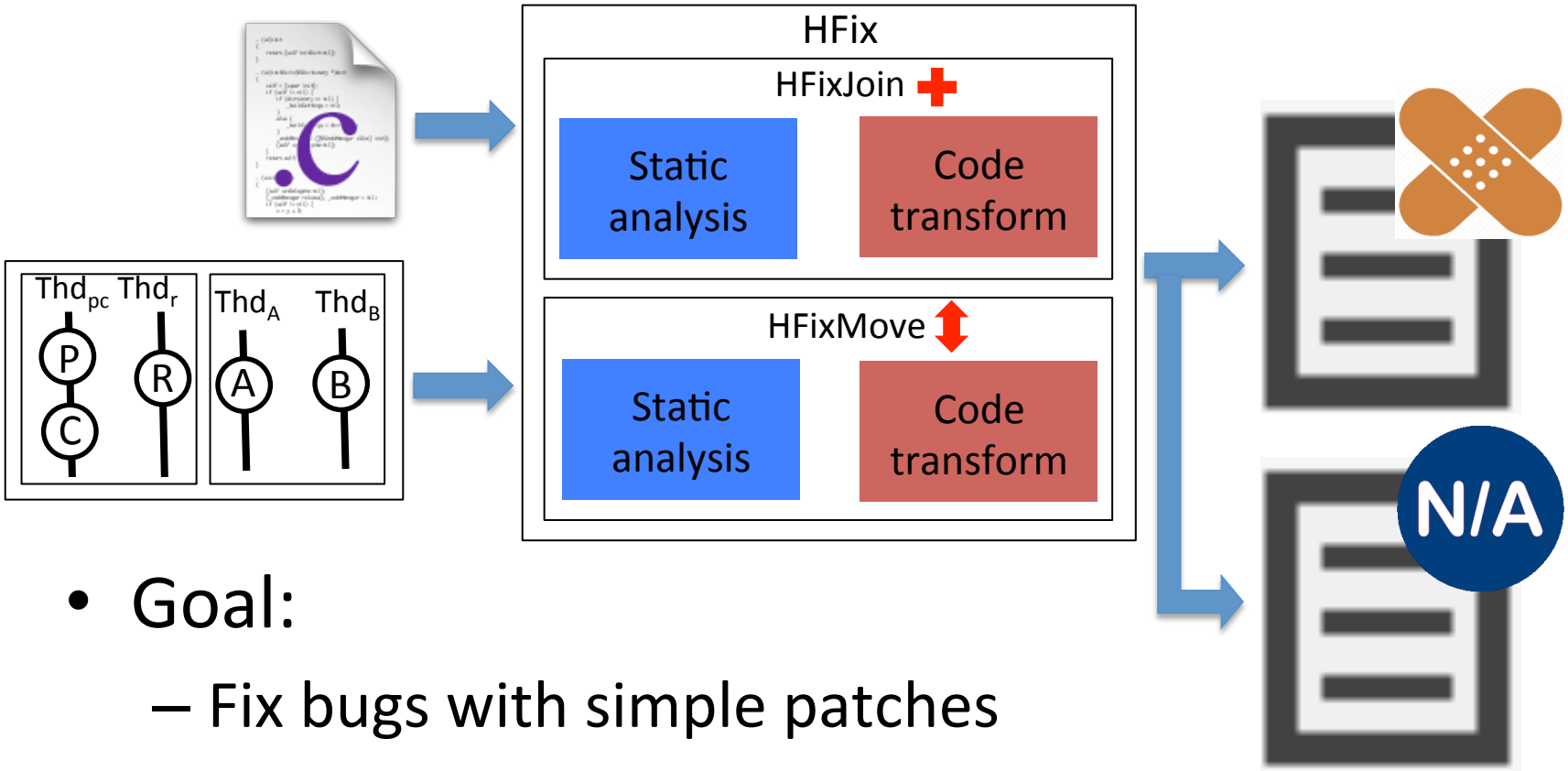


- Different bugs are suitable different strategies

Outline

- Motivation
- Manual patch studying
- HFix
  - HFixJoin
  - HFixmove
- Evaluation
- Conclusion

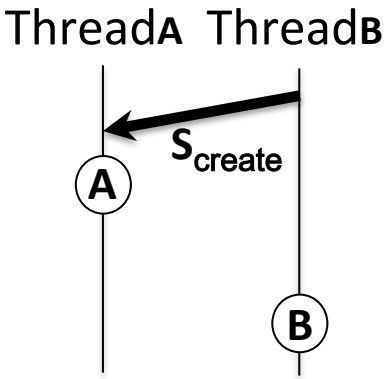
HFix overview



- Goal:
  - Fix bugs with simple patches
  - Do not introduce new bugs

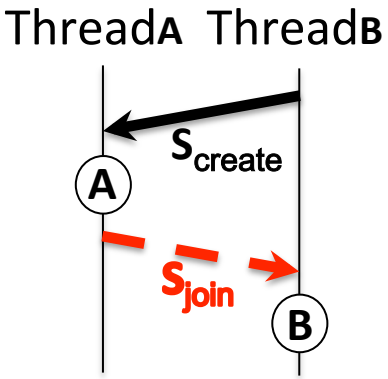
HFixjoin Intuition

- Which kind of bugs can be fixed by HFixjoin?



HFixjoin Intuition

- Which kind of bugs can be fixed by HFixjoin?

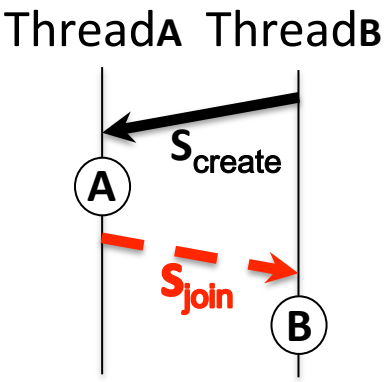


HFixjoin

- Step1: Suitability checking
  - Goal: do not introduce new bugs.
- Step2: Insert join synchronization
  - Goal: fix the bug

Suitability checking (static analysis)

- Is Thread**A** a joinable child thread of Thread**B**?
- Only one Thread**B**?
- Is Thread**A** already joined?
- Will there be deadlock/severe performance slow-down?



Suitability checking (static analysis)

- Is Thread**A** a joinable child thread of Thread**B**?
- Only one Thread**B**?
- Is Thread**A** already joined?
- Will there be deadlock/severe performance slow-down?

```
  //parent thread
+ thread_join(…);
  fp = NULL;
  signal(condV);
```

```
//child thread
fputs(fp,…);
wait(condV);
```

Insert join synchronization

- Insert join before operation B
- Patch Example

```
//child thread                //parent thread
if(…){                        pthread_create(&tid,…)
  unlock(f->mut);
  return;                     ...
}
                              f->mut = NULL;
```

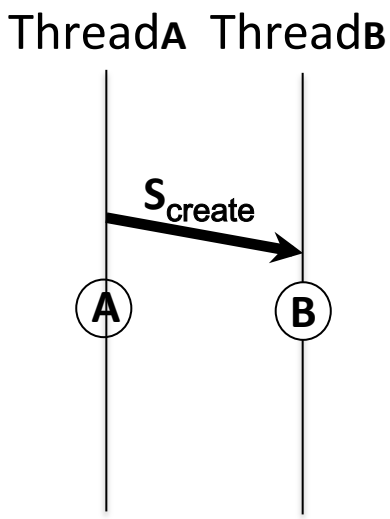
Insert join synchronization

- Insert join before operation B
- Patch Example

```
//child thread                //parent thread
if(…){                        pthread_create(&tid,…)
  unlock(f->mut);             tmp = tid;
  return;                     ...
}                             pthread_join(tmp);
                              f->mut = NULL;
```

HFixmove Intuition

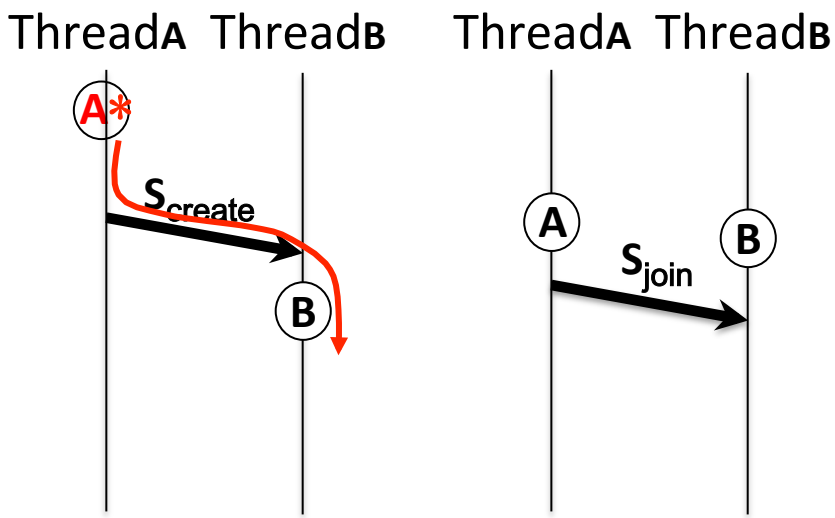
- What kind of bugs can be fixed by HFixmove?



(a) **Move**$_{\textbf{create}}$ fixes OV bug
move-up A or move-down S

HFixmove Intuition

- What kind of bugs can be fixed by HFixmove?

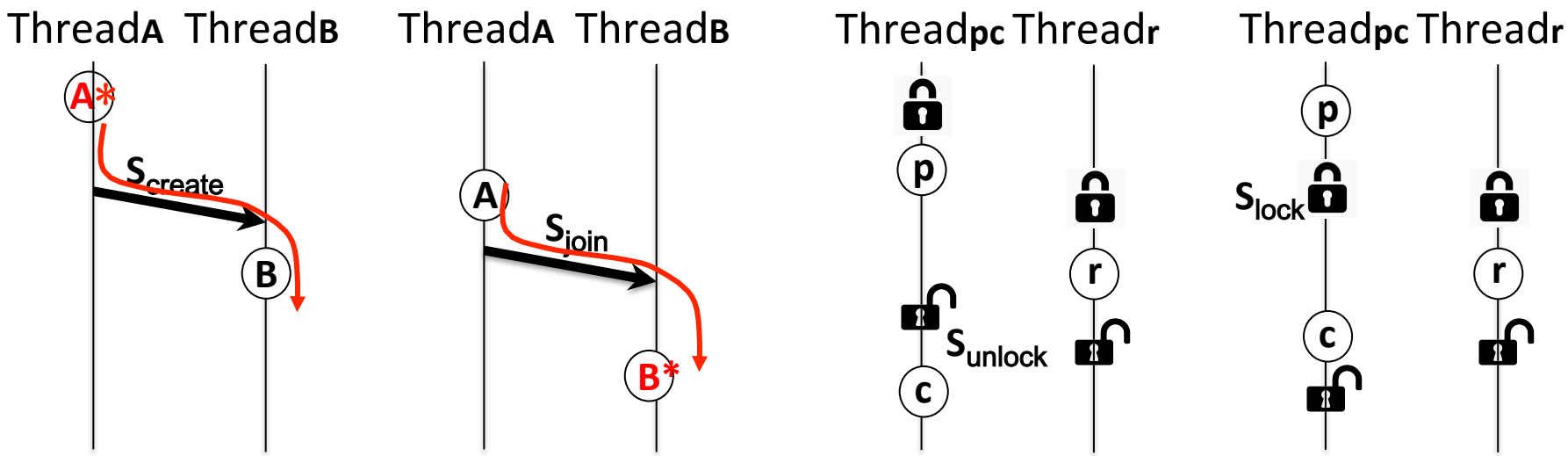


(a) **Move**$_{\textbf{create}}$ fixes OV bug
move-up A or move-down S

(b) **Move**$_{\textbf{join}}$ fixes OV bug
move-up S or move-down B

HFixmove Intuition

- What kind of bugs can be fixed by HFixmove?



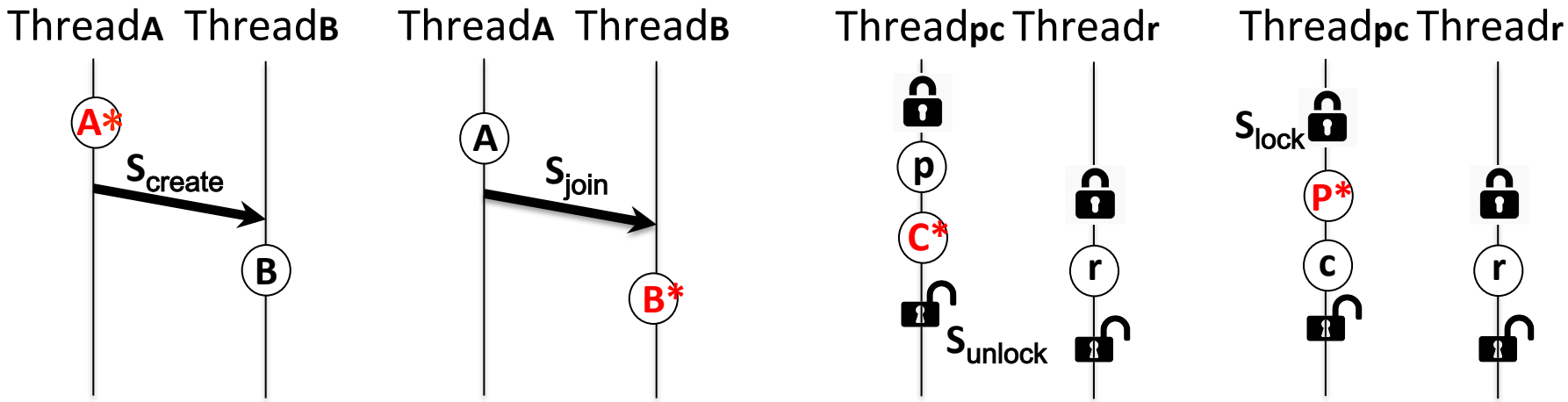
(a) **Move**$_{create}$ fixes OV bug
move-up A or move-down S

(b) **Move**$_{join}$ fixes OV bug
move-up S or move-down B

(c) **Move**$_{unlock}$ fixes AV bug
move-up c or move-down S

(d) **Move**$_{lock}$ fixes AV bug
move-up S or move-down p

Step1: identify two operations to move



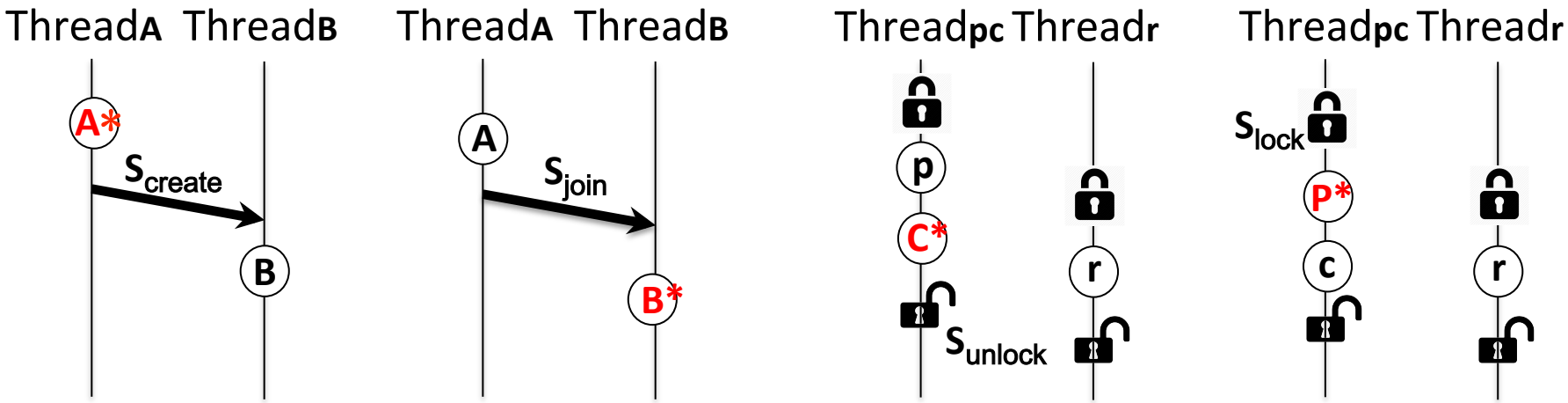
(a) **Move**$_{create}$ fixes OV bug
move-up A or move-down S

(b) **Move**$_{join}$ fixes OV bug
move-up S or move-down B

(c) **Move**$_{unlock}$ fixes AV bug
move-up c or move-down S

(d) **Move**$_{lock}$ fixes AV bug
move-up S or move-down p

Step1: identify two operations to move

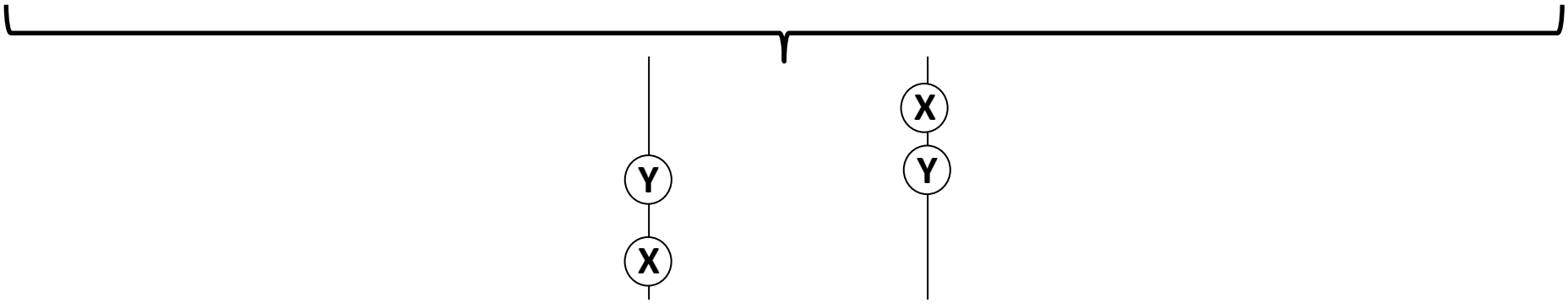


(a) **Move**$_{create}$ fixes OV bug
move-up A or move-down S

(b) **Move**$_{join}$ fixes OV bug
move-up S or move-down B

(c) **Move**$_{unlock}$ fixes AV bug
move-up c or move-down S

(d) **Move**$_{lock}$ fixes AV bug
move-up S or move-down p



Step 2: decide the moving destination X*

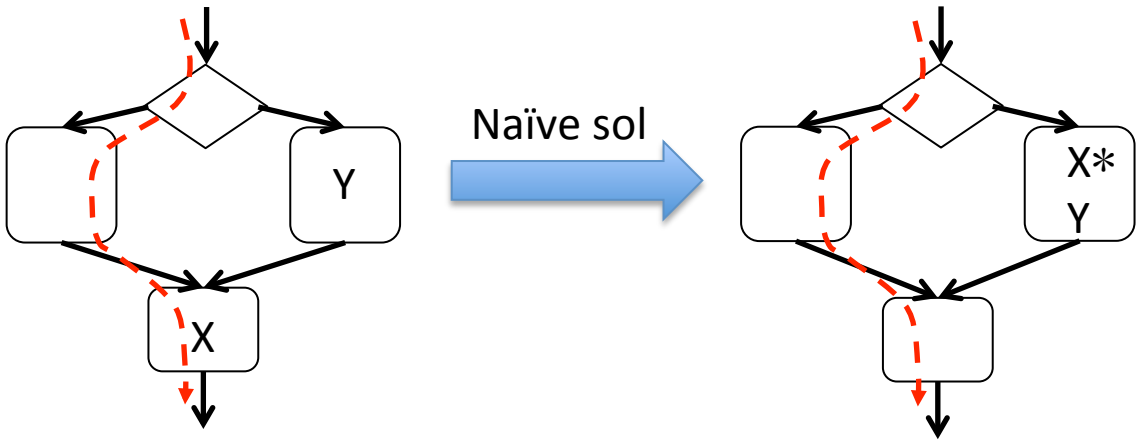
- Goal

  # of executions of X == # of executions of X*

Strawman solution doesn't work

of executions of X ≠ # of executions of X*

$$1 \neq 0$$



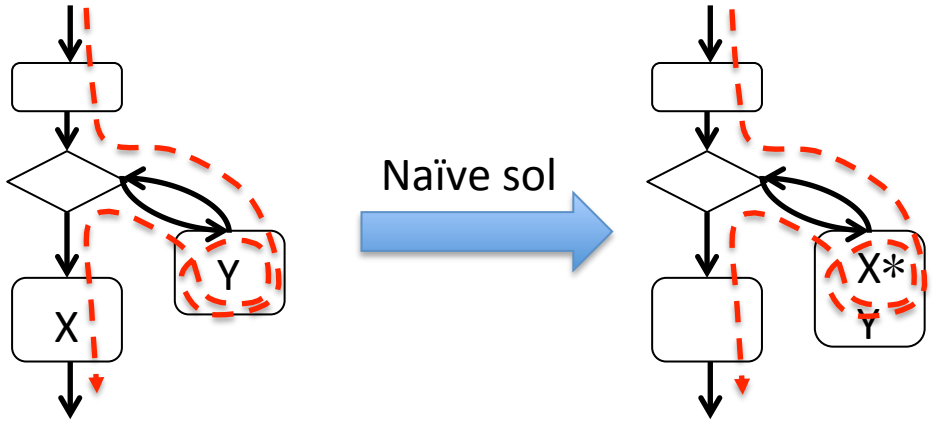
Originally, X executes once.

After moving, X* does not execute.

Strawman solution doesn't work

of executions of X ≠ # of executions of X*

$$1 \neq N$$

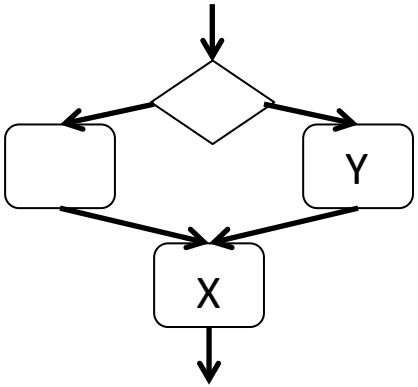


Originally, X executes once.

After moving, X* executes more than once.

HFixmove demonstration

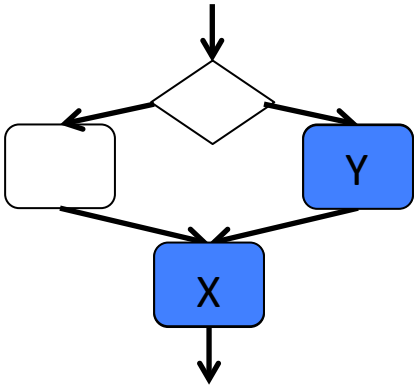
- Guarantee: X* will execute exactly once before Y.



* X and Y are inside the same function; neither is in loop

HFixmove demonstration

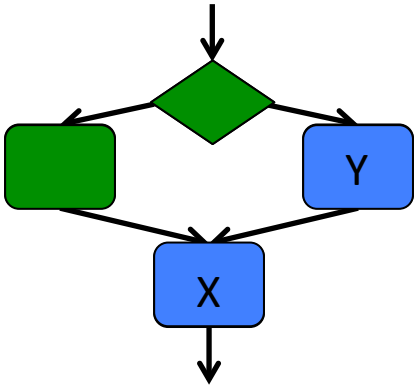
- Guarantee: X* will execute exactly once before Y.



* X and Y are inside the same function; neither is in loop

HFixmove demonstration

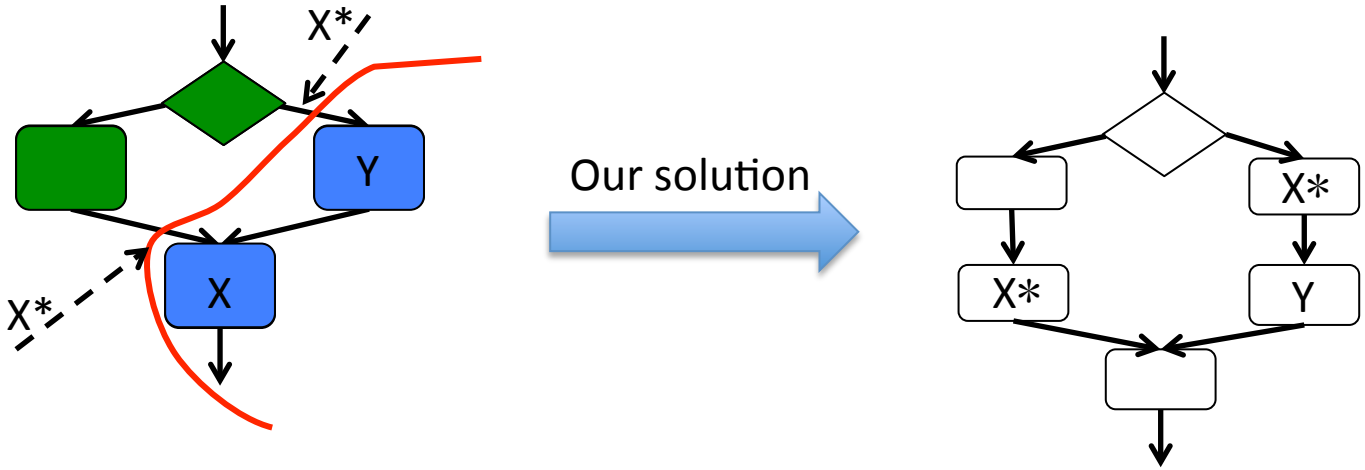
- Guarantee: X* will execute exactly once before Y.



* X and Y are inside the same function; neither is in loop

HFixmove demonstration

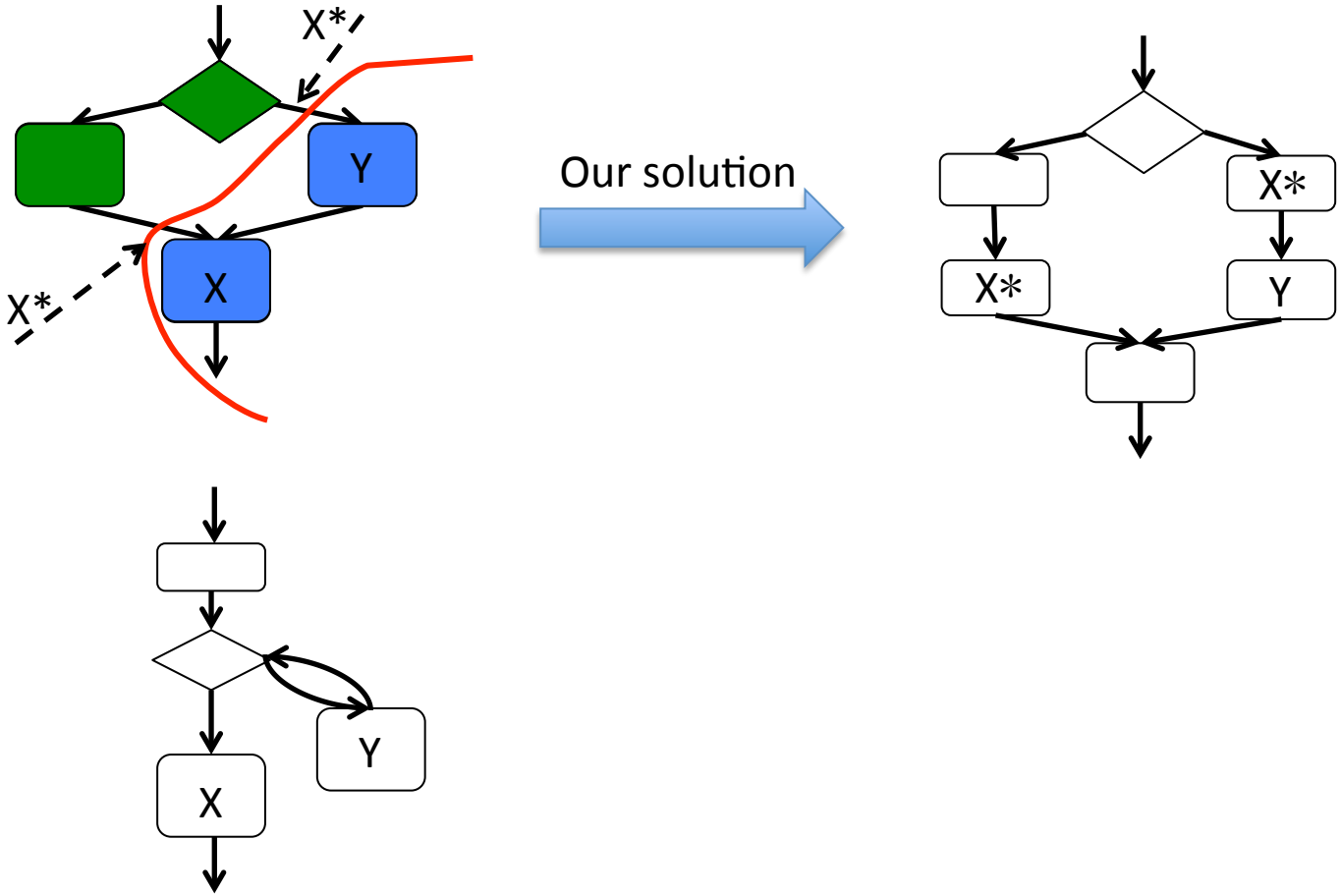
- Guarantee: X* will execute exactly once before Y.



* X and Y are inside the same function; neither is in loop

HFixmove demonstration

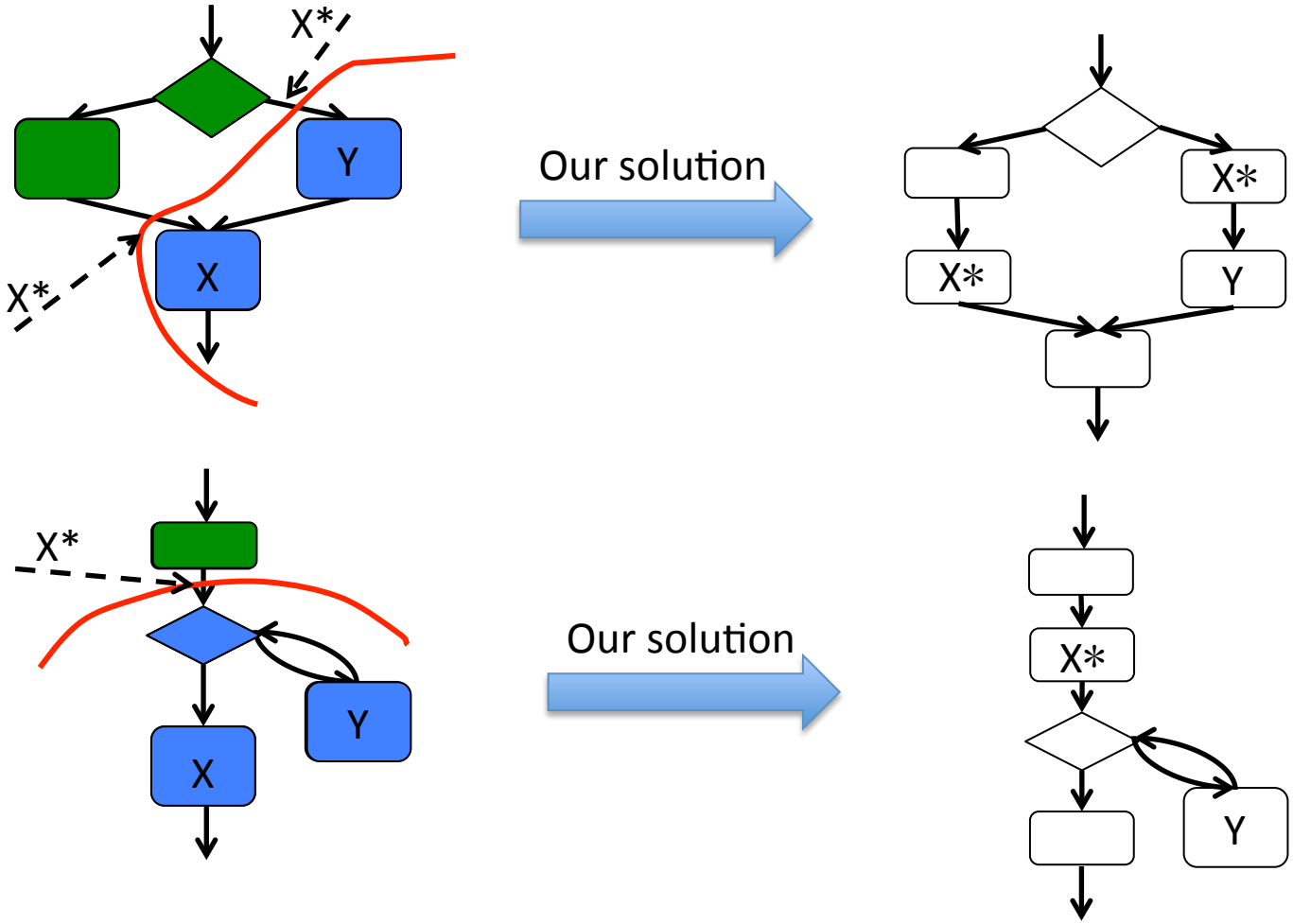
- Guarantee: X* will execute exactly once before Y.



* X and Y are inside the same function; neither is in loop

HFixmove demonstration

- Guarantee: X* will execute exactly once before Y.



* X and Y are inside the same function; neither is in loop

Step 3: suitability checking

- Does *move* change local def-use relationship?

```
+ gX = m;
  s_create(…);
  m = 0;
  gX = m;
```

Step 3: suitability checking

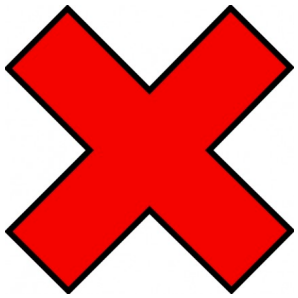
- Does *move* change local def-use relationship?

```
+ gX = m;
  s_create(…);
  m = 0;
  gX = m;
```

Step 3: suitability checking

- Does *move* change local def-use relationship?

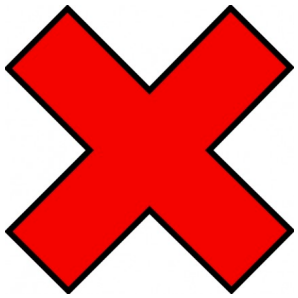
```
+ gX = m;
  s_create(…);
  m = 0;
  gX = m;
```



Step 3: suitability checking

- Does *move* change local def-use relationship?

```
+ gX = m;
  s_create(…);
  m = 0;
```



- Might *move* introduce deadlocks?

HFixmove patch example

```
//child thread                    //parent thread
                                + h->band = bdNew(h);
assert(h->band);                  pthread_create(…)
                                  ...
                                - h->band = bdNew(h);
```

Outline

- Motivation
- Manual patch studying
- HFix
- Evaluation
  - Methodology
  - Results
- Conclusion

Methodology

- Benchmark Suite
  - 13 bugs set up by Cfix [OSDI'12]
  - All the AV bugs in ASPLOS'08 suite that are fixed through moving synchronization by developers.

Results for OV bugs

App.	HFix	VS. Manual
FFT	✚ join	✓
HTTrack	-	✗
Mozilla	-	✗
PBZIP2	✚ join	✓
Transmission	⬍ create	✓
X264	⬍ join	✗
ZSNES	⬍ create	✓

Results for OV bugs

App.	HFix		CFix	
	Strategy	#.Sync	Strategy	#.Sync
FFT	✚ join	1	✚ signal/wait	5
HTTrack	-	-	✚ signal/wait	3
Mozilla	-	-	✚ signal/wait	2
PBZIP2	✚ join	1	✚ signal/wait	5
Transmission	⬍ create	0	✚ signal/wait	2
X264	⬍ join	0	✚ signal/wait	7
ZSNES	⬍ create	0	✚ signal/wait	3

Results for AV bugs

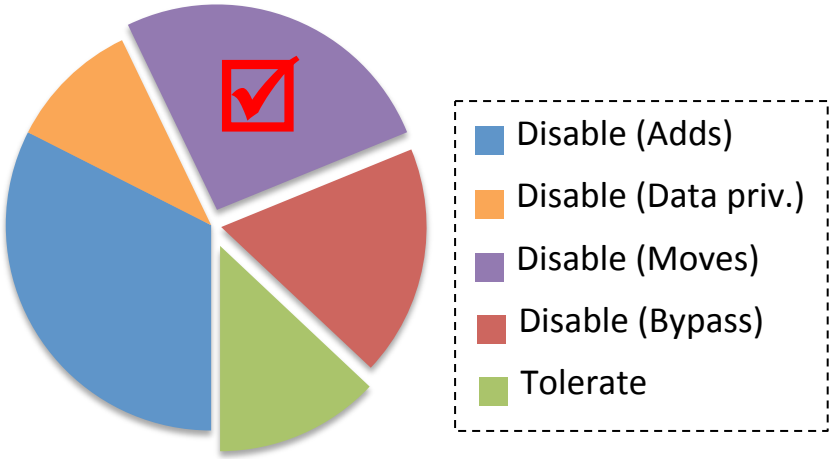
App.	HFix
Apache	✗
Mozilla1	↕ lock
Mozilla2	↕ lock
MySQL1	↕ unlock
MySQL2	↕ unlock
MySQL3	↕ unlock

Outline

- Motivation
- Manual patch studying
- HFix
- Evaluation
- Conclusion

Conclusion

- Manual patch study:
  - Add join for OV (HFix)
  - Move existing sync. (HFix)
  - Bypass & tolerate (Future work)
- HFix
  - A useful complement.
  - Generate much simple patches.



Thank you!
Q&A

THE UNIVERSITY OF CHICAGO