Applying Hardware Transactional Memory for Concurrency-Bug Failure Recovery in Production Runs

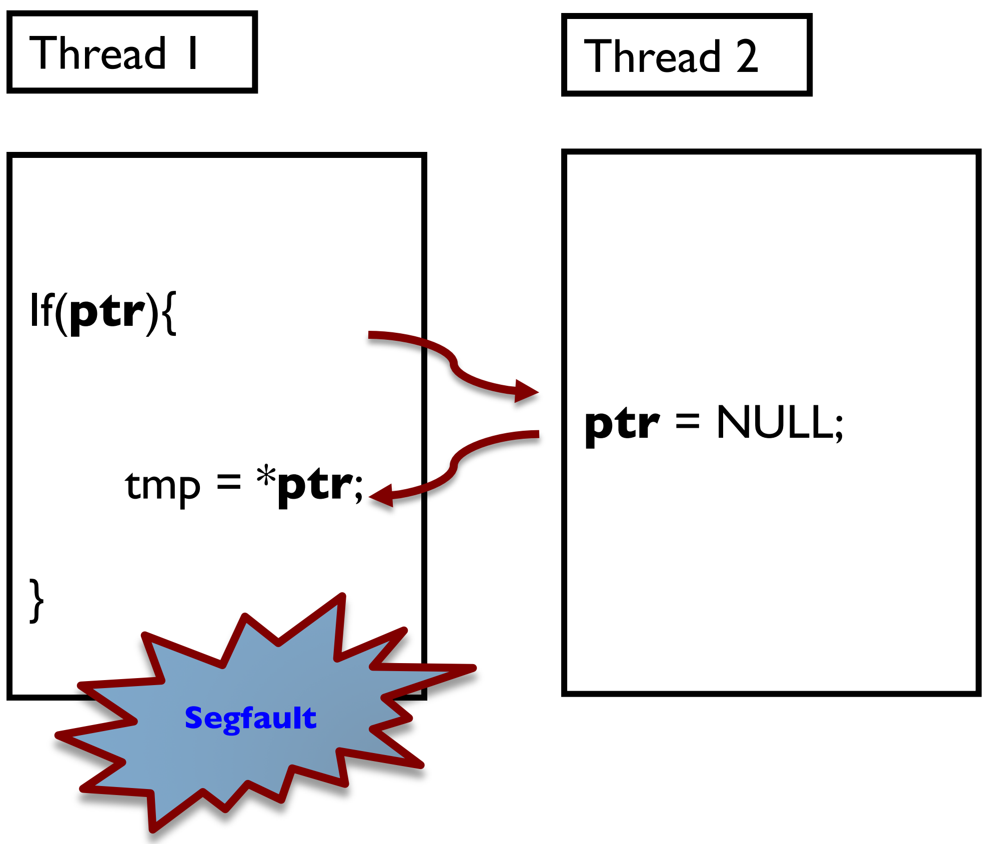
Yuxi Chen, Shu Wang, Shan Lu,
and Karthikeyan Sankaralingam*

THE UNIVERSITY OF CHICAGO

*WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

What are the concurrency bugs?

- Synchronization mistakes in multithreaded programs



Concurrency bugs are problematic

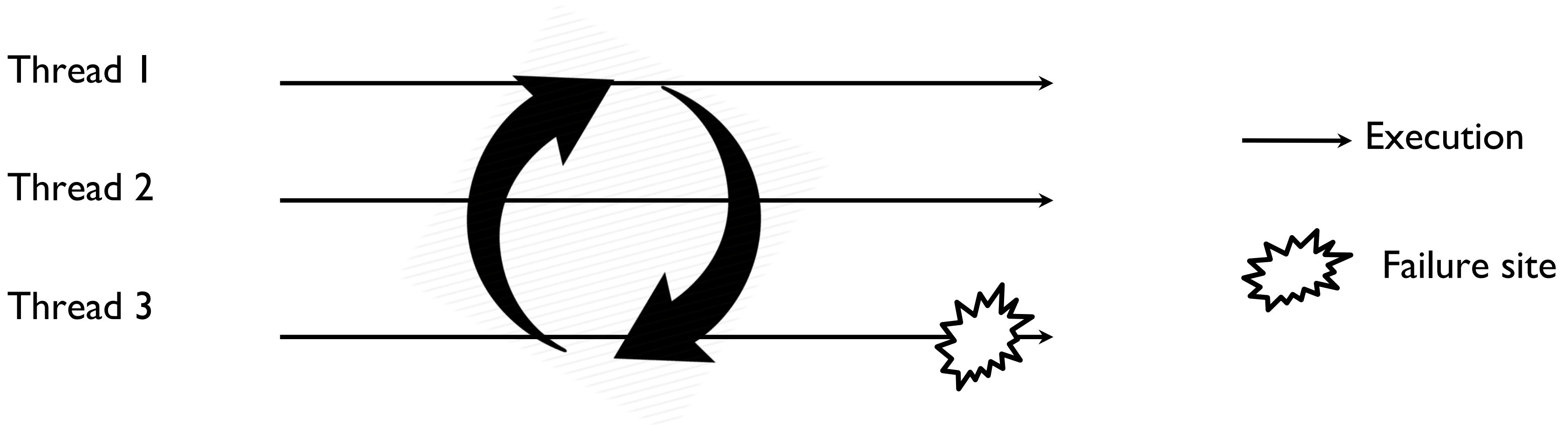
- ❑ Common
- ❑ Hard to diagnose and fix correctly
- ❑ Disasters in **production runs**

Failure recovery in production run

- ❑ Semantic correctness
- ❑ Performance
  - ❖ Low recovery latency
  - ❖ Low overhead

So what's
the solution?

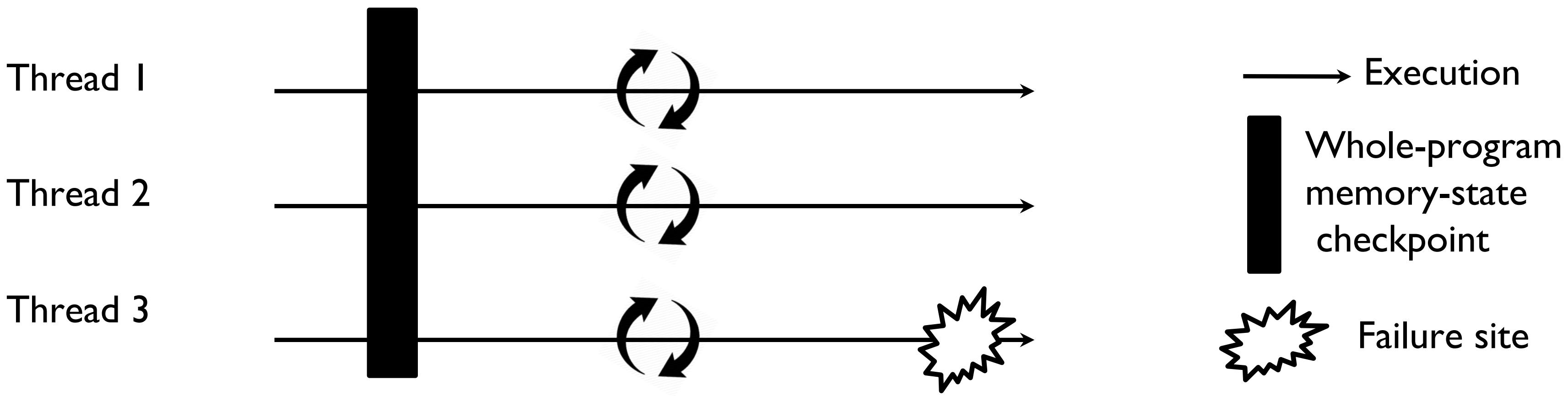
Rollback and reexecution can help recovery

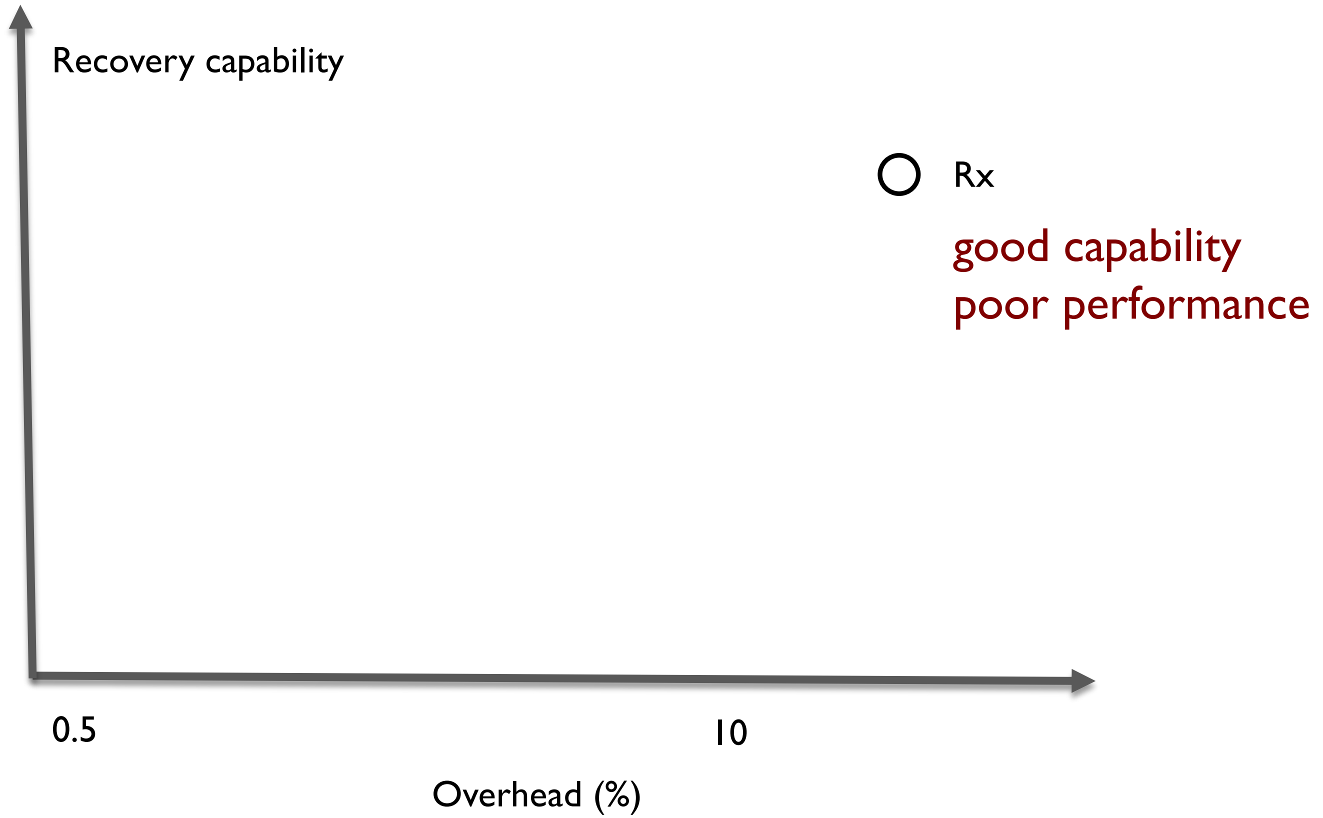


How to rollback and reexecute correctly with low overhead?

Past solution I

Traditional Roll-back Recovery

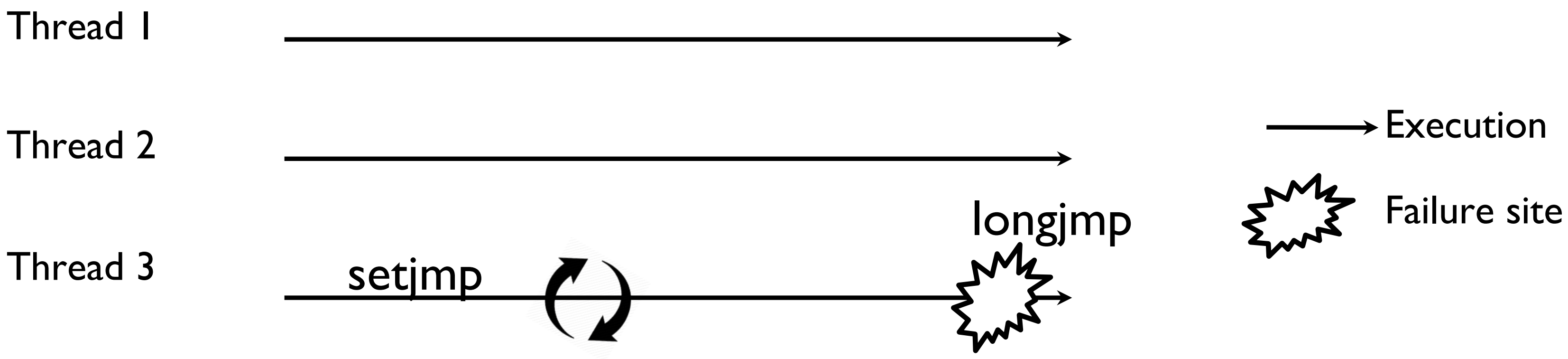


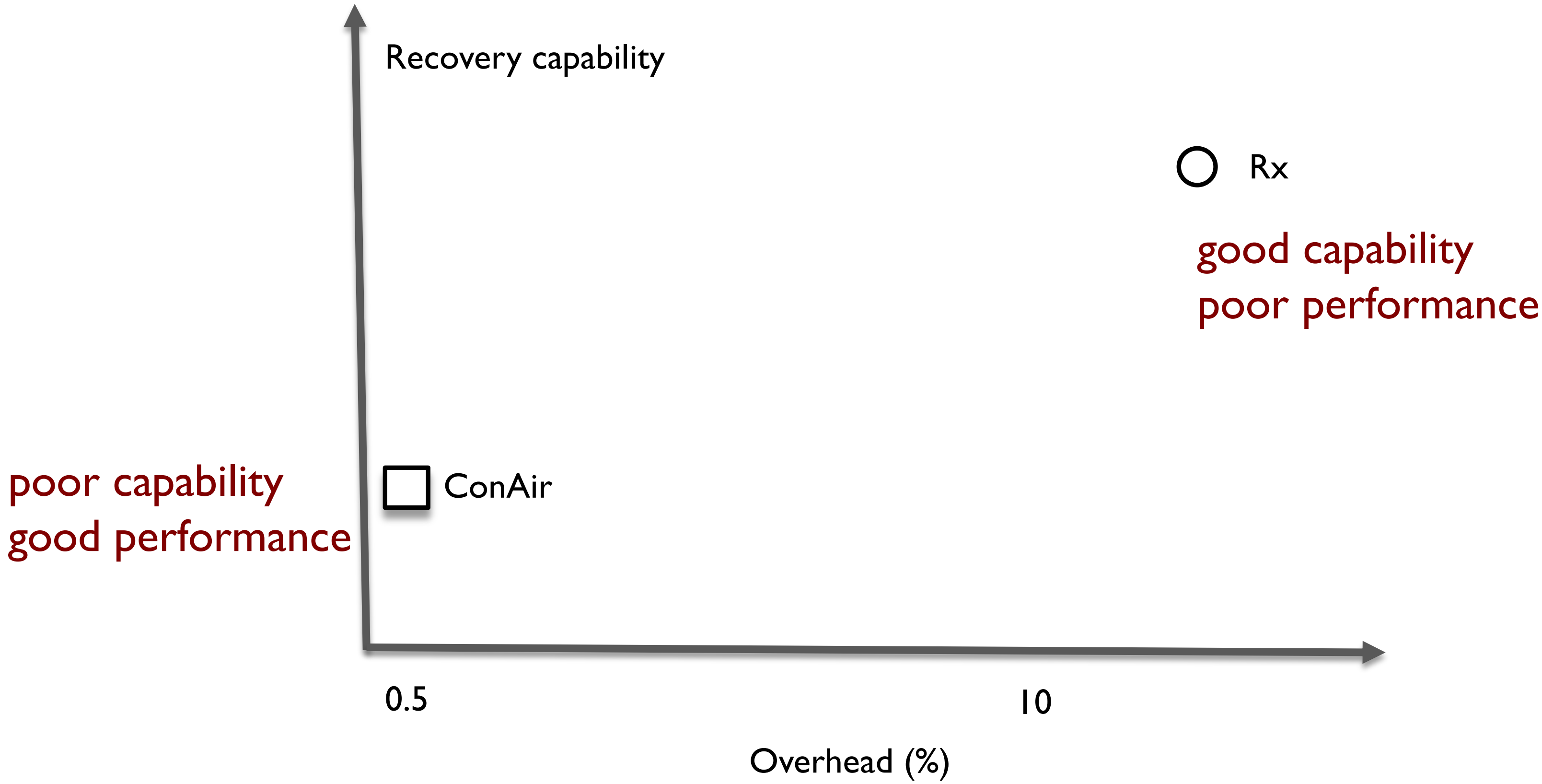


[1]Rx: Treating bugs as allergies – a safe method to survive software failure, SOSP'05

Past solution II

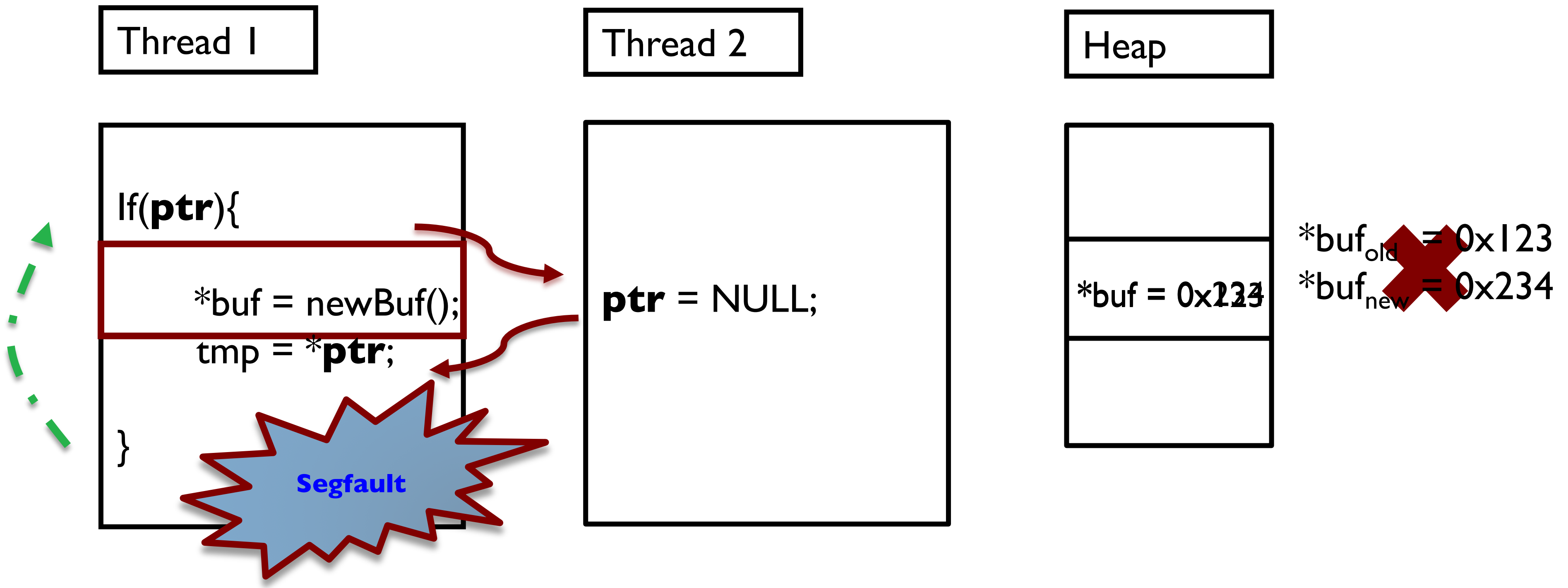
ConAir Roll-back Recovery



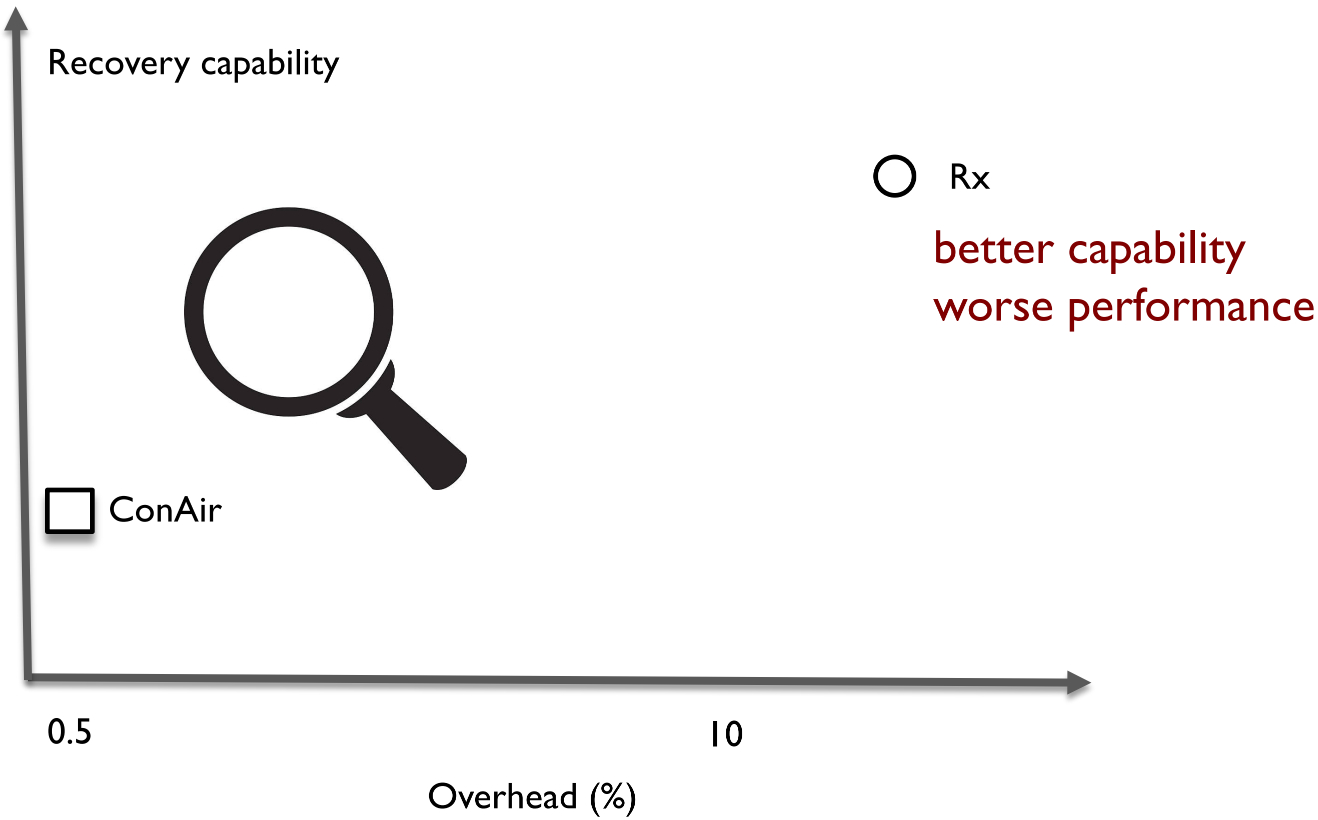

Recovery capability
Rx
good capability
poor performance
poor capability
good performance
ConAir
0.5
10
Overhead (%)

ConAir's recovery capability limitation

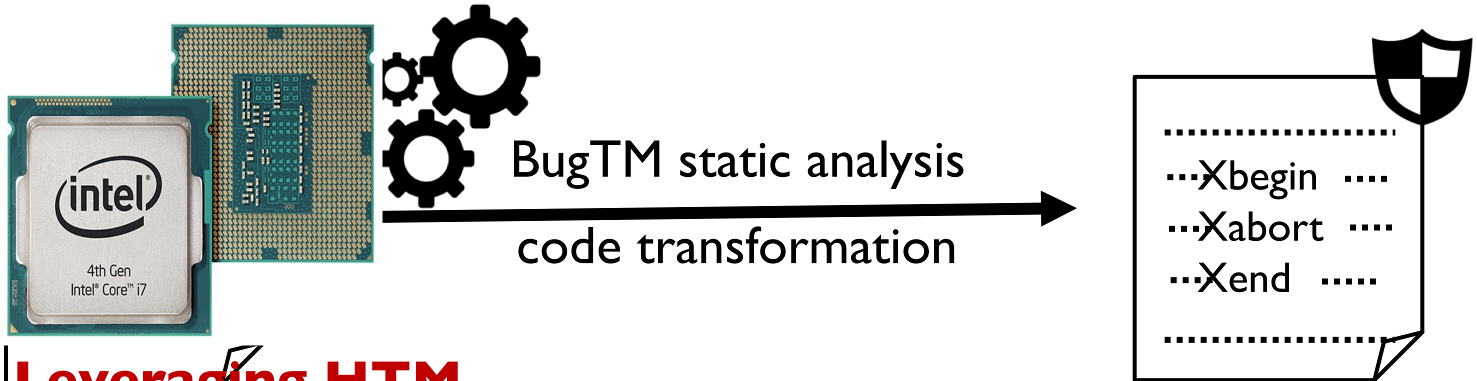
- ConAir cannot reexecute **shared-variable writes**



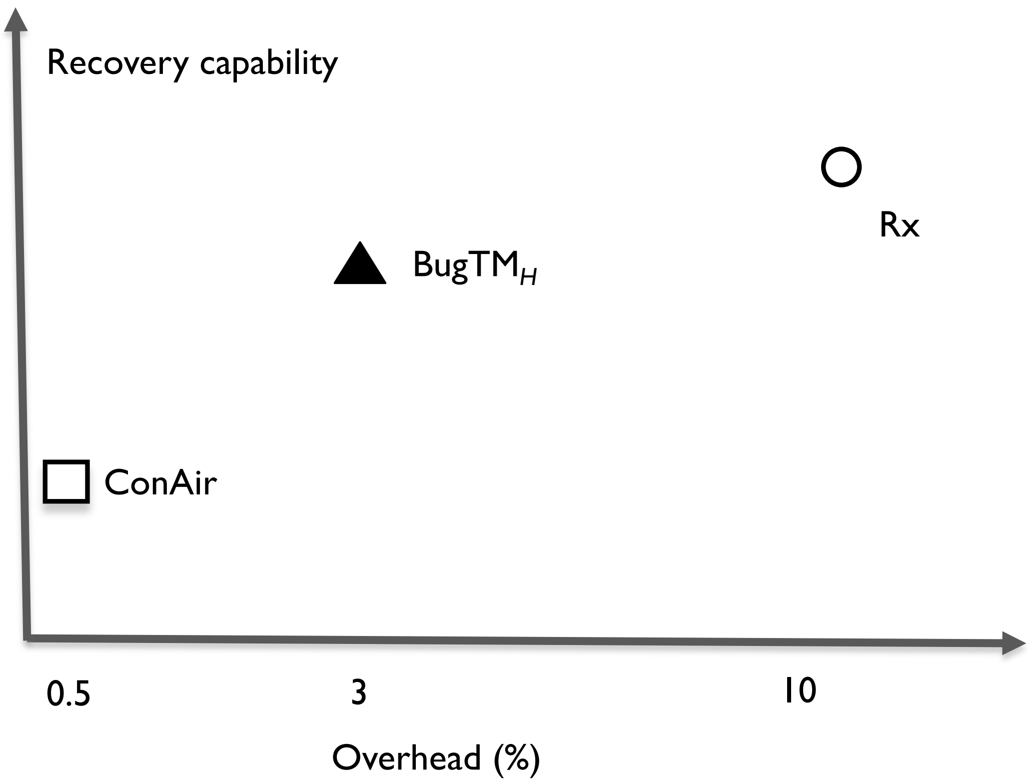
How can we do better?



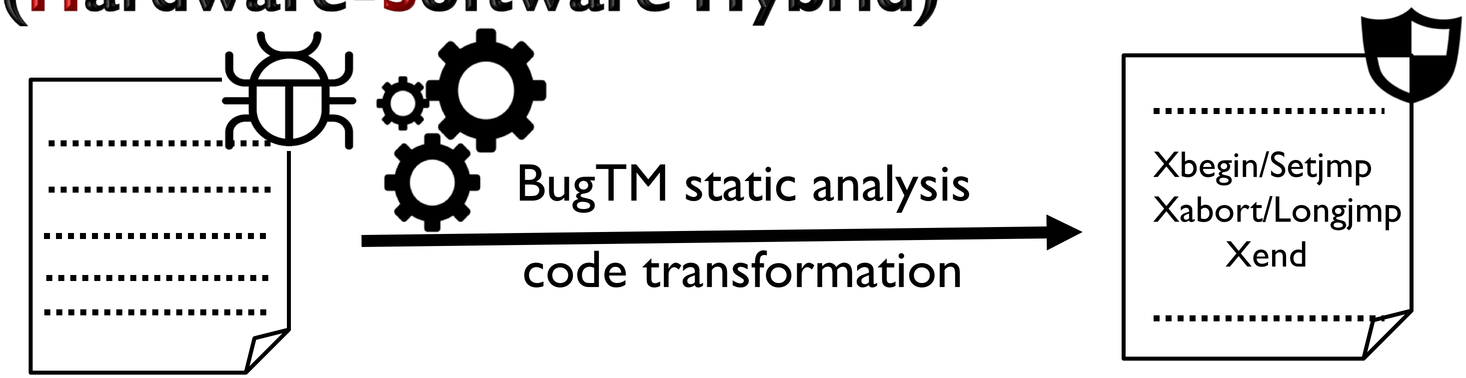
Our contribution: BugTM$_H$ (Hardware)



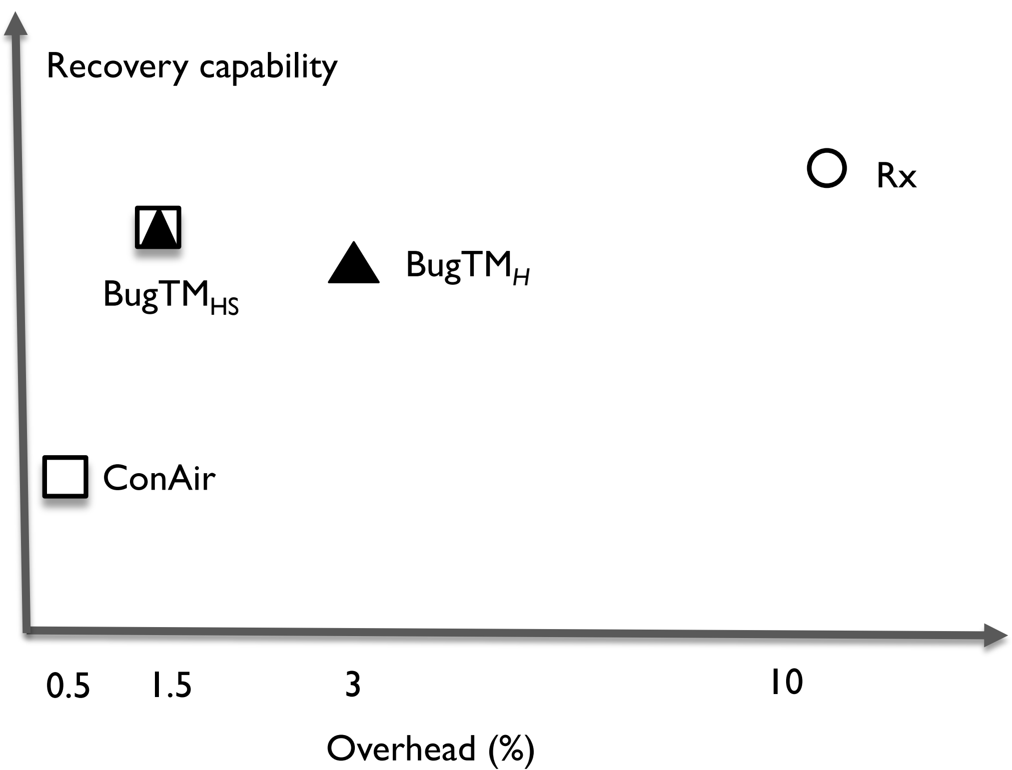
Leveraging HTM



Our contribution: BugTM_{HS} (Hardware-Software Hybrid)



FREE LUNCH!

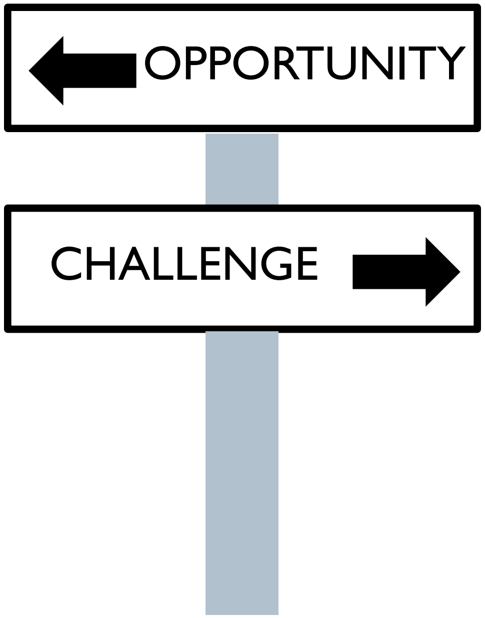


Outline:

1. BugTM_{H}
2. BugTM_{HS}
3. Evaluation Methodology
4. Experiment Results
5. Conclusion

HTM

- Implicit checkpoint
- Rollback-reexecution



Challenges

- Performance challenges
  - High frequency of transaction uses
  - Unsuitable content of transactions (eg. trapping instructions)
  - Nesting && Loops

Challenges

- Performance challenges
- Correctness challenges
  - Unpaired transaction-start and transaction-commit
  - Deterministic aborts (eg. trapping instruction aborts)

Challenges

- ❑ Performance challenges
- ❑ Correctness challenges
- ❑ Failure recovery challenges
  - ❖ Surround the buggy codes when failures happen
  - ❖ HTM-abort handlers

☑ 1. Carefully design HTM start, commit, and abort routines

☑ 2. Selectively insert HTM start, commit, and abort routines

Intel hardware transaction memory --TSX

- ❑ **Xbegin**
- ❑ **Xend**
- ❑ **Xtest**
- ❑ **Xabort**

Thread 1

```
+Xbegin();

+        Xabort();
         assertion();

+if(Xtest())
   Xend();
+    Xend();
```

Routines Design

```
mXbegin(){
        if(!Xtest())
                Xbegin();
}
```

No nested TM!

```
mXend(){
        if(Xtest())
                Xend();
}
```

No unpaired Xbegin and Xend during run time!

Where to put mXabort

Principle: put mXabort before where failures might happen

```
If(ptr){
    *buf = newBuf();
    if(ptr == NULL){
      +mXabort();
      *buf = newBuf();
      __assert_fail();
    }else{
      tmp = *ptr;
        tmp = *ptr;
    }
}
```

Where to put mXbegin

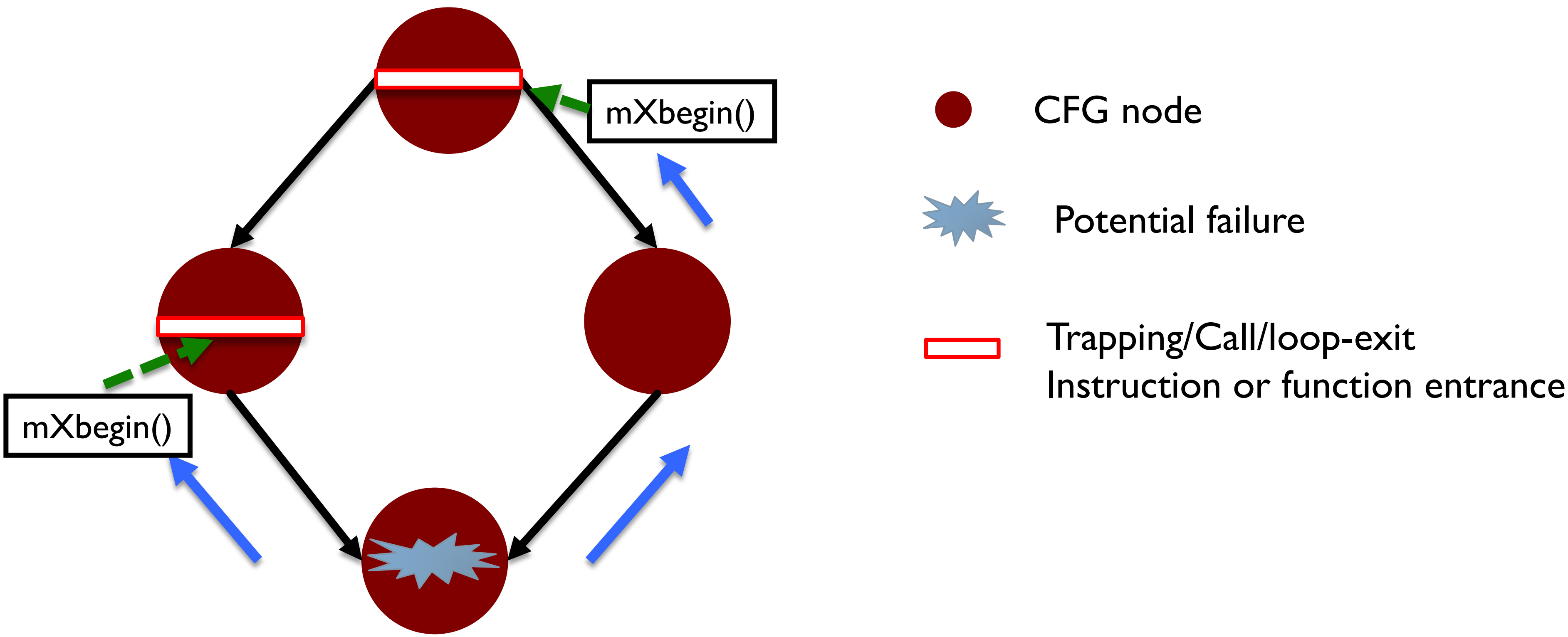
Principle:

- ❑ Avoid trapping instructions abort
- ❑ Minimize capacity abort

```
time();
+ mXbegin();
If(ptr){
        *buf = newBuf();
        if(Ptr==NULL){
+           mXabort();
            __assert_fail;
        }else{
            tmp = *ptr;
        }

}
```

Where to put mXbegin



Where to put mXend

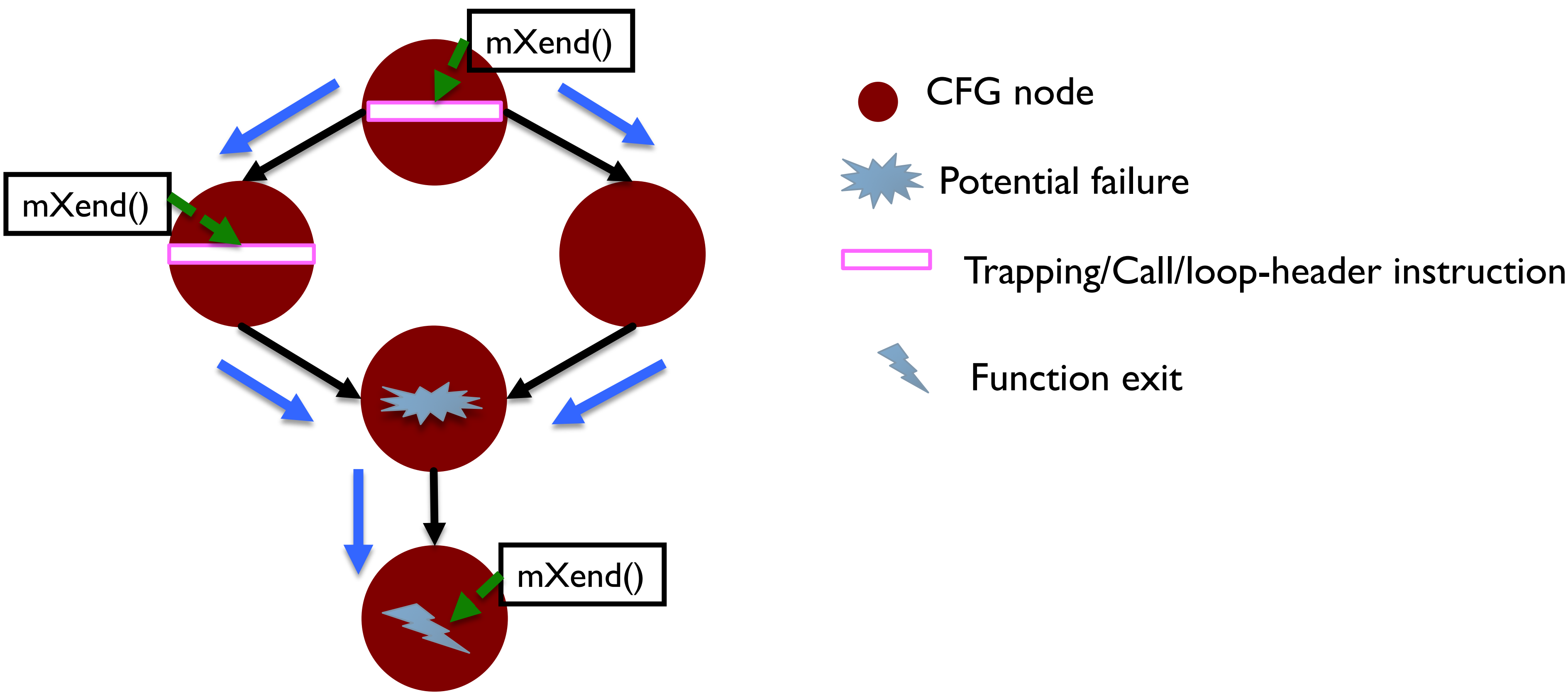
Principles:

- Avoid trapping instructions abort
- Minimize capacity abort
  - End TM before loop entry
  - End TM before function exit

```
time ();
+ mXbegin();
If(ptr){
        *buf = newBuf();
        if(Ptr==NULL){
+           mXabort();
            __assert_fail;
        }else{
            tmp = *ptr;
        }

}
+ mXend();
```

Where to put mXend



Design fallback and retry

Principle: Reexecution only when aborts might be caused by concurrency bugs

- Concurrency bug relevant aborts (reexecute in Tx mode)
  - Data conflict abort && Xabort abort

- Concurrency bug irrelevant aborts (reexecute in non-Tx mode)
  - Capacity abort && Trapping instruction abort

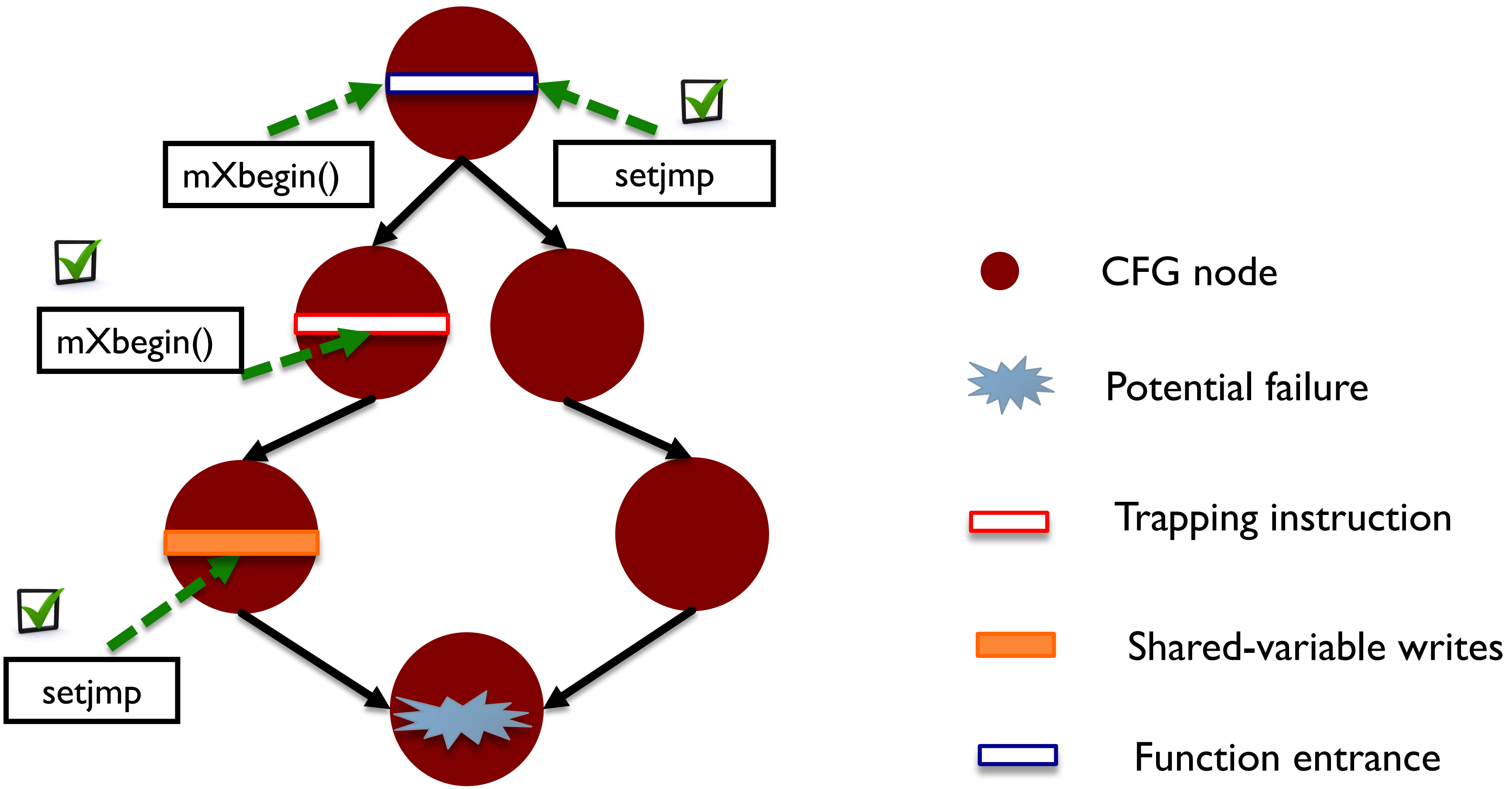
Outline:

1. BugTM_{H}
2. BugTM_{HS}
3. Evaluation Methodology
4. Experiment Results
5. Conclusion

- **Where to put setjmp and mXbegin**

Loc_{setjmp} : locations where ConAir inserts setjmp
$Loc_{mXbegin}$: locations where $BugTM_H$ inserts mXbegin

- ❑ Insert setjmp at every Loc_{setjmp}
- ❑ Insert mXbegin only when $Loc_{mXbegin}$ is farther than Loc_{setjmp}
- ❑ Not insert mXbegin if Loc_{setjmp} and $Loc_{mXbegin}$ are same



- **Where to put mXend**
  - ❑ Exactly same as BugTM$_H$

- **How to retry**

❑ HTM rollback first (under an active transaction)
➢ longer reexecution region

❑ Longjmp rollback (not under an active transaction)
➢ If HTM rollback fails, longjmp rollback can still have a chance

Outline:

1. BugTM_H
2. BugTM_{HS}
3. Evaluation Methodology
4. Experiment Results
5. Conclusion

Evaluation methodology

- Benchmarks (29 bugs[1,2,3,4,5,6])

....

- micro architecture
  Broadwell (4-core Intel Core i7-5775C)
- LLVM

[1] Guoliang Jin, et al, Automated atomicity-violation fixing, PLDI'11
[2] Horatiu Jula, et al, Deadlock immunity: Enabling systems to defend against deadlocks, OSDI'08
[3] Yao Shi, et al, Do I use the wrong definition? DefUse: Definition-use invariants for detecting concurrency and sequential bugs, OOPSLA'10
[4] Wei Zhang, et al, ConAir: Featherweight concurrency bug recovery via single-threaded idempotent execution, ASPLOS'13
[5] Wei Zhang, et al, ConSeq: Detecting concurrency bugs through sequential errors, ASPLOS'11
[6] Wei Zhang, et al, ConMem: Detecting Crash-Triggering Concurrency Bugs through an Effect-Oriented Approach, ACM TOSEM, 2012.

Recovery capability

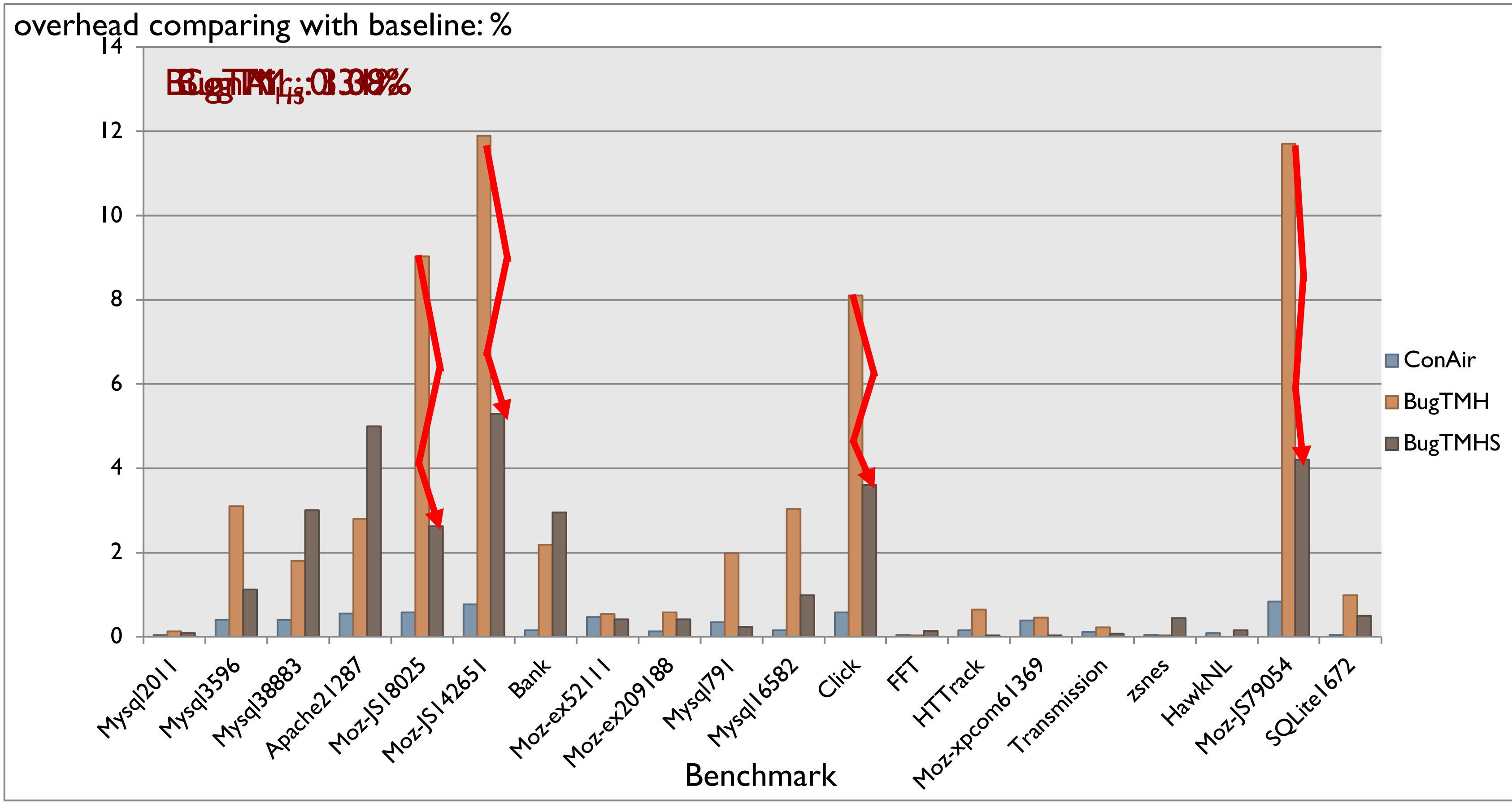
Recovery capability comparison

Benchmarks(ID)	Root Cause	ConAir	BugTM$_H$	BugTM$_{HS}$
Mysql2011	RAR Atomicity Violation	-	✔	✔
Mysql38883	RAR Atomicity Violation	-	✔	✔
Apache21287	RAW Atomicity Violation	-	✔	✔
Moz-JS18025	RAW Atomicity Violation	-	✔	✔
Moz-JS142651	RAW Atomicity Violation	-	✔	✔
Bank	WAR Atomicity Violation	-	✔	✔
Transmission	Order Violation	✔	-	✔
Total		1	6	7

ConAir < BugTM$_H$ < BugTM$_{HS}$

Performance Overhead

ConAir > BugTM$_{HS}$ > BugTM$_{H}$



Outline:

1. BugTM_H
2. BugTM_{HS}
3. Evaluation Methodology
4. Experiment Results
5. Conclusion

Conclusions

- BugTM can help recover all major types of concurrency-bug failures in production run
  - Low run-time overhead
  - Outperform the state of art approach (ConAir)
  - Present novel ways of using HTM techniques (failure recovery)

Come and eat this free lunch!

Thank you !

Q & A