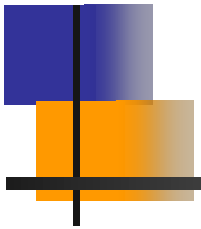
Do you have to reproduce the bug on the first replay attempt?

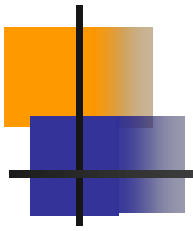


PRES: Probabilistic Replay with Execution Sketching on Multiprocessors

Soyeon Park, Yuanyuan Zhou
University of California, San Diego

Weiwei Xiong, Zuoning Yin, Rini Kaushik, Kyu H. Lee, Shan Lu
University of Illinois at Urbana Champaign

OPERA

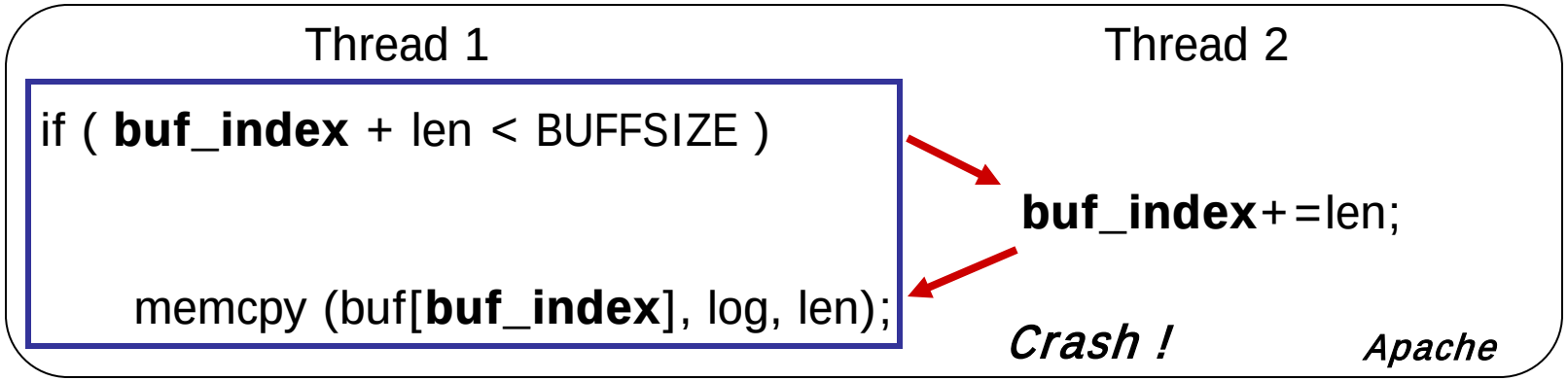


Concurrency bugs are important

- Writing concurrent program is difficult
  - Programmers are used to sequential thinking
  - → Concurrent programs are prone to bugs
- Concurrency bugs cause severe real-world problems
  - Therac-25, Northeast blackout
- Multi-core trend worsens the problem

Characteristics of Concurrency Bugs

- A concurrency bug may need a special thread interleaving to manifest

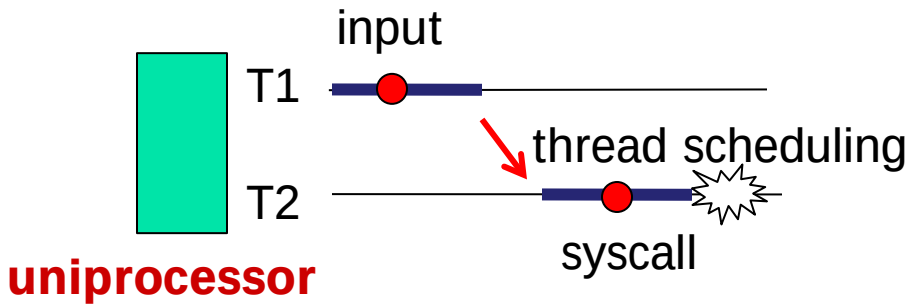


Two implications :

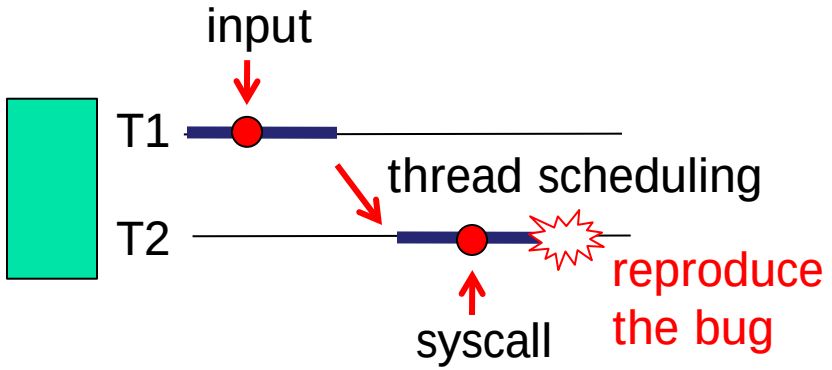
- Hard to expose a concurrency bug during testing
- Difficult to reproduce a concurrency bug for diagnosis

Deterministic Replay of Uniprocessor

- Recording non-deterministic factors and re-execution
  - Inputs (keyboards, networks, files, etc)
  - Thread scheduling
  - Return values of system calls



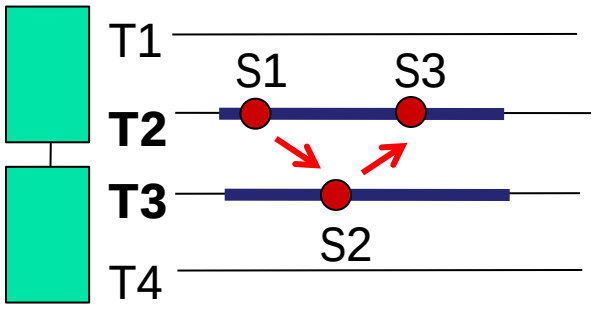
< Production run >



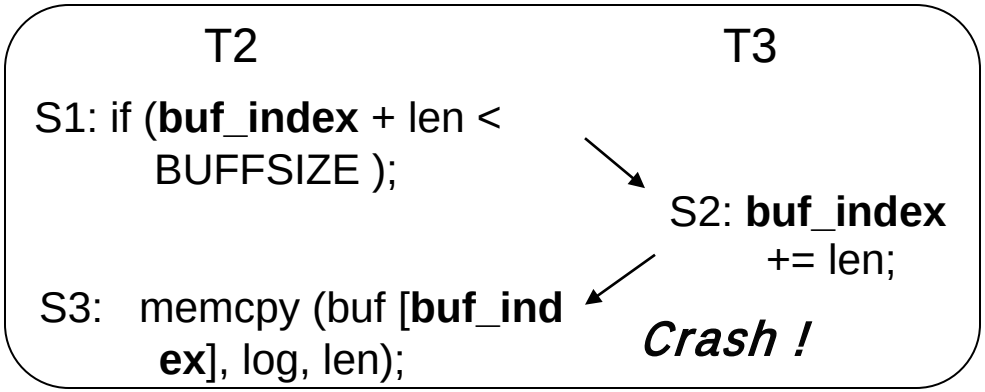
< Replay run >

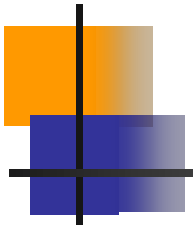
Deterministic Replay for Multiprocessors

- Much more difficult
  - Multi-threads execute simultaneously on different processors
- Extra source of non-determinism:
  - Interleaving of shared memory accesses



multiprocessor





State of the Art on Multiprocessor Replay

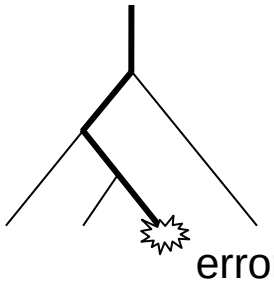
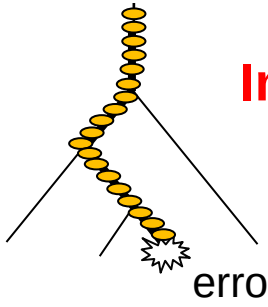
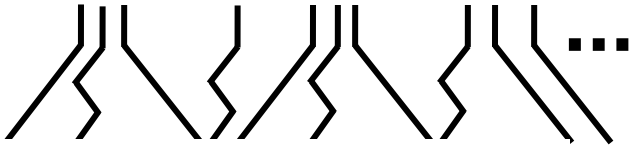
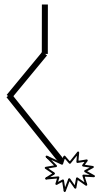
- Hardware-assisted approach
  - Recording all thread interactions with new hardware extension
    - ex) Flight Data Recorder, BugNet, Strata, RTR, DMP, Rerun, etc.

  ⇨ None of them exists in reality !

- Software-only approach
  - High production-run overhead (> 10-100X) ⇨ Not practical !
    - due to capturing the global order of shared memory accesses
    - ex) InstantReplay, Strata/s, etc.

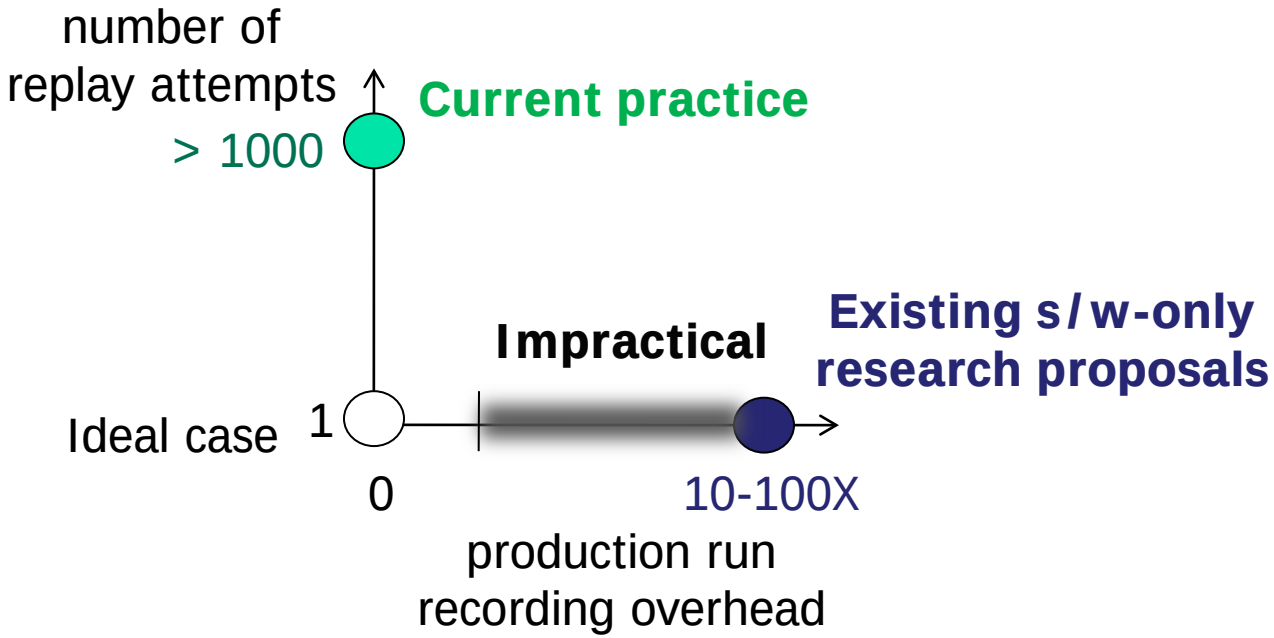
  - Recent work: SMP-Revirt
    - use page protection mechanism to optimize memory monitoring
    - >10X production-run overhead on 2 or 4 processors
    - has false sharing and page contention issues (scalability)

Contrast between Common Practice & Existing Research Proposals

	Common practice	Existing research proposals
Production run	 **0 %** overhead	 **10-100 X** slowdown
Diagnosis phase	 **>1000** replay attempts*	 the **1st** replay attempt

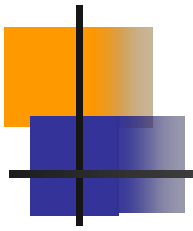
* : according to our experimental results

Observations



1) Production run performance is more critical than replay time

2) We do NOT need to reproduce a bug on the 1st replay attempt



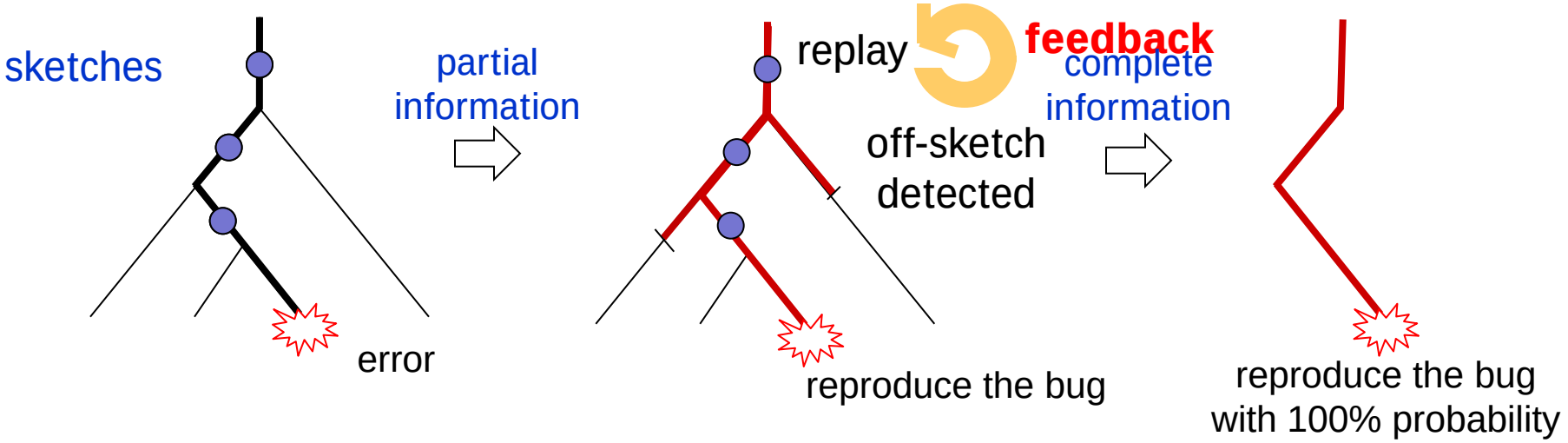
Our Idea

Probabilistic Replay with Execution Sketching (PRES)

- Record only partial information during production run
  ⇨ **Low recording overhead**
- Push the complexity to diagnosis time
- Leverage **feedback** from unsuccessful replays

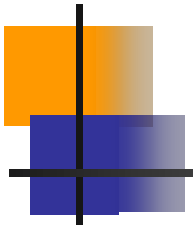
PRES Overview

- Probabilistic Replay via Execution Sketching (PRES)



Sketch recording during production run	Partial-Information based replay (PI-Replay)	Diagnosis phase

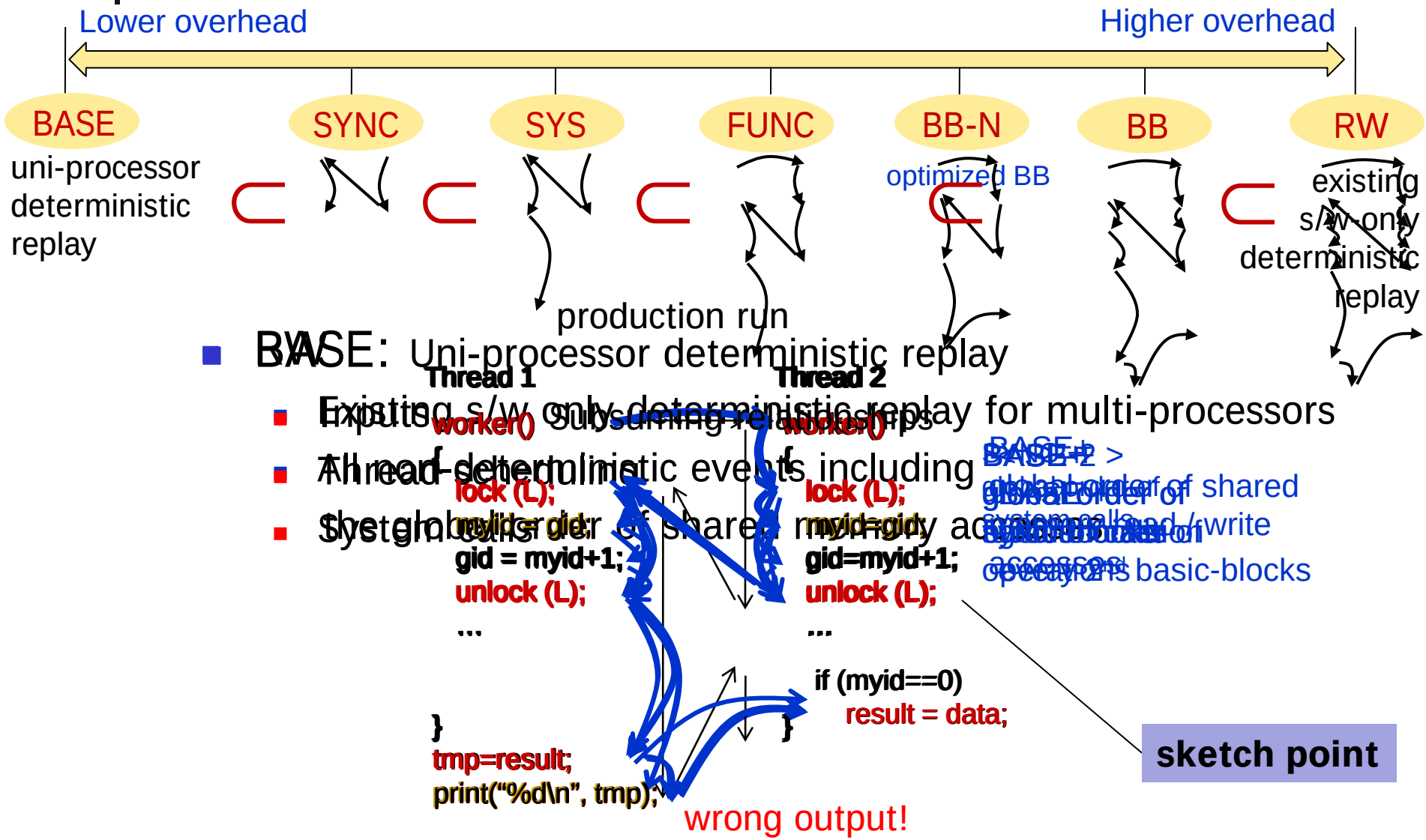
- Recording partial information (sketch) during production run
- Reproducing a bug, not the original execution

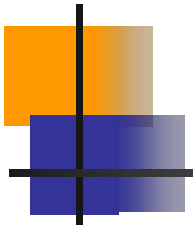


Contents

- Introduction
- Our approach
- Overview of PRES
- **Sketch recording**
- Bug reproduction
  - Partial-Information based replayer
  - Monitor
  - Feedback generator
- Evaluation
- Conclusion

Sketch Recording



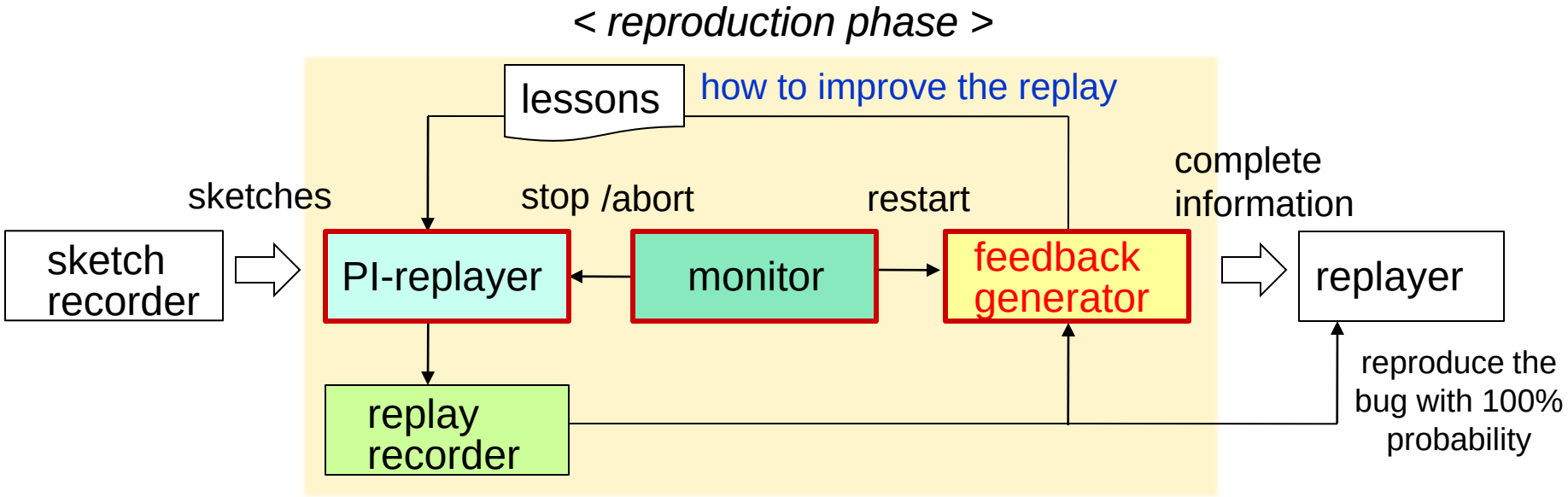


Contents

- Introduction
- Our approach
- Overview of PRES
- Sketch recording
- **Bug reproduction**
  - Partial-Information based replayer (PI-Replayer)
  - Monitor
  - Feedback generator
- Evaluation
- Conclusion

Partial Information-based Replay

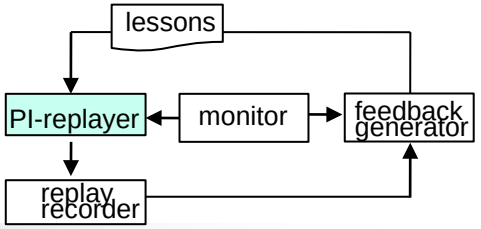
- Process of bug reproduction phase



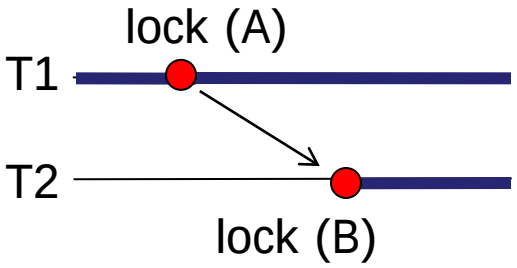
Monitor is used for:

- Detecting successful bug reproduction
- Detecting off-sketch path: deviates from sketches

PI-replayer

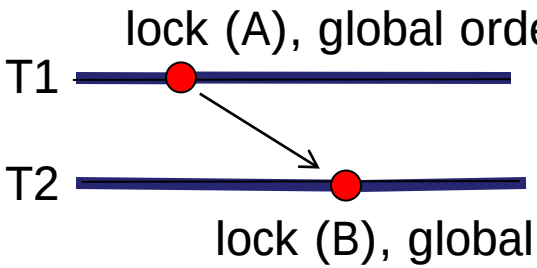


- Partial-Information based replayer
  - Consults the execution sketch to enforce observed global orders
  - Right before re-executing a sketch point, make sure that all prior points from other threads have been executed



< Production run >

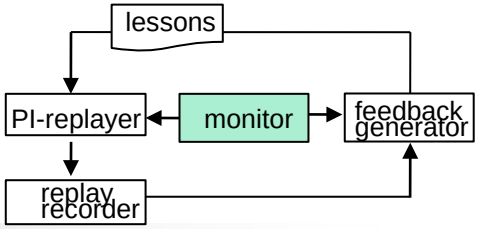
SYNC sketches
T1 : lock A, global order 1
T2 : lock B, global order 2



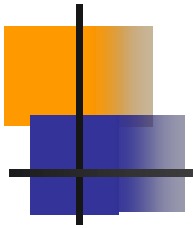
wait for T1 to execute lock A first

< Replay run >

Monitor



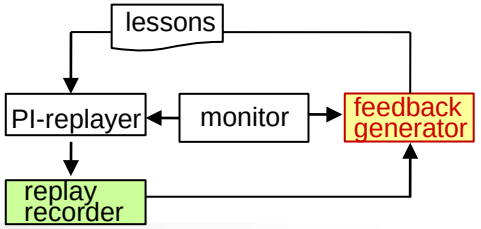
- Detect successful bug reproduction
  - Crash failure - PRES can catch exceptions
  - Deadlock - a periodic timer to check for progress
  - Incorrect results - programmer needs to provide conditions for checking
  - Can leverage testing oracles and existing bug detection tools
- Detect unsuccessful replay
  - Compare against the execution sketch from the original execution
  - Prevent from giving useless replay efforts on a wrong path



What if a replay attempt fails?

- Replay it again!
  - Restart from the beginning or the previous checkpoint
- Shall we do something different next time?
  - Random approach: just leave it to fate
  - Systematic approach
    - Actively learn from previous mistakes

Feedback Generator (1/2)



- Why previous replays cannot reproduce a bug?
  - Some un-recorded data races execute in different orders

Production run

```
Thread 1                    Thread 2
worker()                    worker()
{                           {
   …                           …
                               if (myid==0)
                                 result = data;
}                           }
tmp = result;
printf("%d\n", tmp);
```

1st replay attempt

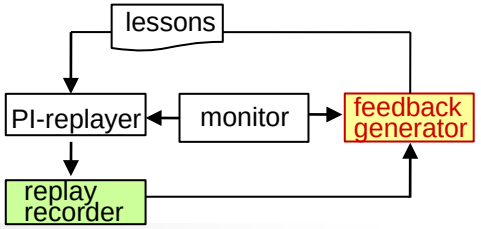
```
Thread 1                    Thread 2
worker()                    worker()
{                           {
   …                           …
}
tmp = result;               if (myid==0)
printf("%d\n", tmp);          result = data;
                            }
```

fail to reproduce the bug!

< FUNC sketches >

This original order is not recorded in the sketch

Feedback Generator (2/2)



- Steps

replay recorder

⇩ R/W traces

Identifying dynamic race pairs

⇩ initial candidates

Filtering order-determined races

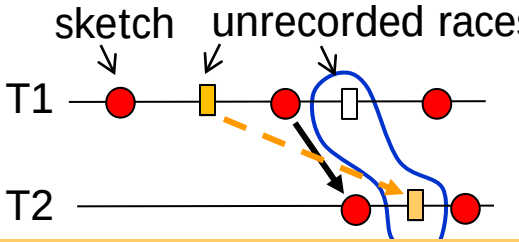
⇩ suspect races

Selecting suspect

⇩ one race

Starting next replay

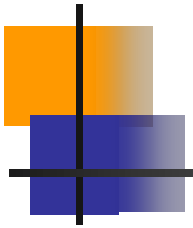
- use happens-before race detector



The race order is already implied by the order of sketch points

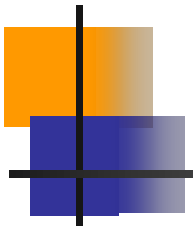
- close-to-failure-first, depth-first

- deterministically execute until the suspect race pair
- flip the race order



Contents

- Introduction
- Our approach
- Overview of PRES
- Sketch recording
- Bug reproduction
  - Partial-Information based replayer
  - Monitor
  - Feedback generator
- **Evaluation**
- Conclusion



Methodology

- Implement PRES using Pin
- 8-core Xeon machine
- 11 evaluated applications
  - 4 server applications (Apache, MySQL, Cherokee, OpenLDAP)
  - 3 desktop applications (Mozilla, PBzip2, Transmission)
  - 4 scientific computing applications (Barnes, Radiosity, FMM, LU)
- 13 real-world concurrency bugs
  - 6 Atomicity violation bugs (single- and multi-variable bugs)
  - 4 Order violation bugs
  - 3 Deadlock bugs

Recording Overhead (1/2)

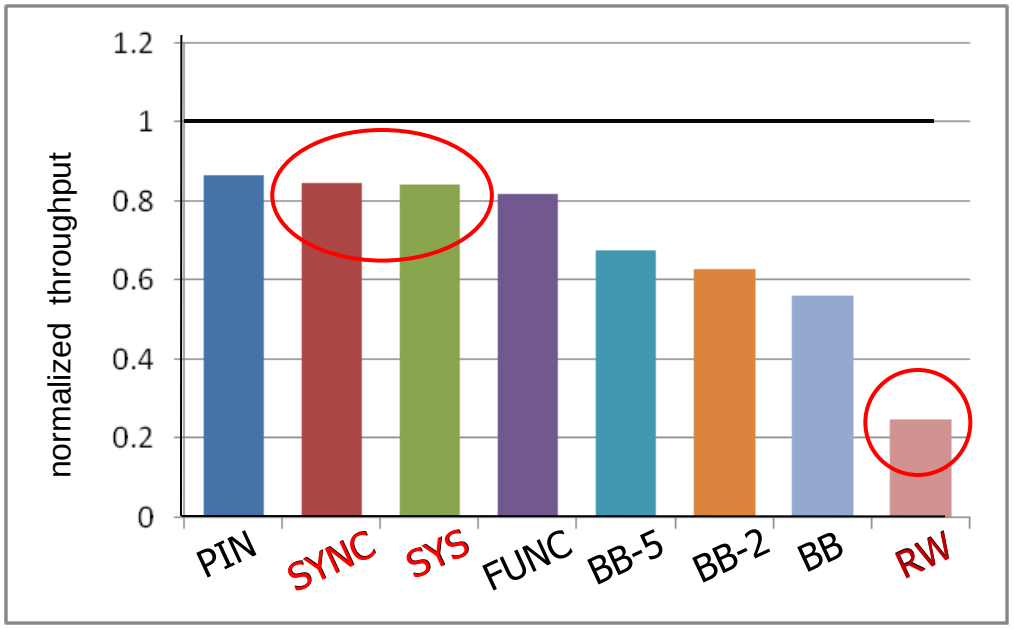
- Non-server applications

Application	PIN	SYNC	SYS	FUNC	BB-5	BB-2	BB	RW
			Desktop Applications (overhead %)					
Mozilla	32.5	59.5	60.6	83.8	598.0	858.8	1213.9	3093.5
PBZip2	17.4	**18.0**	**18.0**	18.4	595.4	1066.6	1977.9	**27009.7**
Transmission	5.9	14.5	21.3	32.7	30.3	33.9	41.9	71.8
			Scientific Applications (overhead %)					
Barnes	2.5	6.5	6.8	427.7	424.2	1122.3	2351.0	28702.2
Radiosity	16.0	16.1	16.1	779.0	480.3	1181.7	2425.5	27209.6

- RW : up to around 280 times slowdown
- SYNC, SYS : Good for performance critical applications
  - 6-60% overhead

Recording Overhead (2/2)

- Server application



MySQL

- SYNC, SYS : 3 times higher than RW throughput

Number of Replay Attempts

NO : not reproduced within 1000 tries

Appli-cation	Bug type	Base	SYNC	SYS	FUNC	BB-5	BB-2	BB	RW
			Server Applications						
Apache	Atom.	NO	96	28	7	8	7	1	1
MySQL	Atom.	NO	3	3	2	3	2	1	1
Cherokee	Atom.	NO	33	22	24	25	8	7	1
OpenLDAP	Deadlock	NO	1	1	1	1	1	1	1
			Desktop Applications						
Mozilla	Multi-v	NO	3	3	3	4	4	3	1
PBZip2	Atom.	NO	3	3	2	4	3	3	1
Transm.	Order	NO	2	2	2	2	2	2	1
			Scientific Applications						
Barnes	Order	NO	10	10	1	74	19	1	1
Radiosity	Order	NO	**NO**	**NO**	152	5	1	1	1

reproduce 12 out of 13 bugs mostly within 10 attempts

Reproduce all 13 bugs mostly within 5 attempts

Benefit of Feedback Generation

NO : not reproduced within 1000 tries

Application		SYNC	SYS	FUNC	BB-5	BB-2	BB
Apache	**w/**	96	28	7	8	7	1
	w/o	**NO**	**NO**	**NO**	**NO**	754	1
PBZip2	**w/**	3	3	2	4	3	3
	w/o	**NO**	**NO**	**NO**	**NO**	**NO**	**NO**
Barnes	**w/**	10	10	1	74	19	1
	w/o	**NO**	**NO**	**NO**	**NO**	**NO**	**NO**

- Low-overhead sketches (SYNC and SYS) can be used to reproduce concurrency bugs only because of our PI-replayer that leverages feedback

- The random approach does NOT work!

Effects of Race Filtering

The number of dynamic benign data races to be explored

Applications	BASE	SYNC	SYS	FUNC	BB-5	BB-2	BB
Apache	54390	1072	274	33	25	25	6
MySQL	39983	2	2	2	2	1	0
Cherokee	133	86	58	16	36	7	3
Mozilla	36258	317	310	14	72	60	42
PBZip2	667	326	318	1	7	4	4
Transmission	225	240	172	6	6	6	4

- Filters out the races whose order can be inferred from sketches
- Significantly shrinks the unrecorded non-deterministic space to be explored

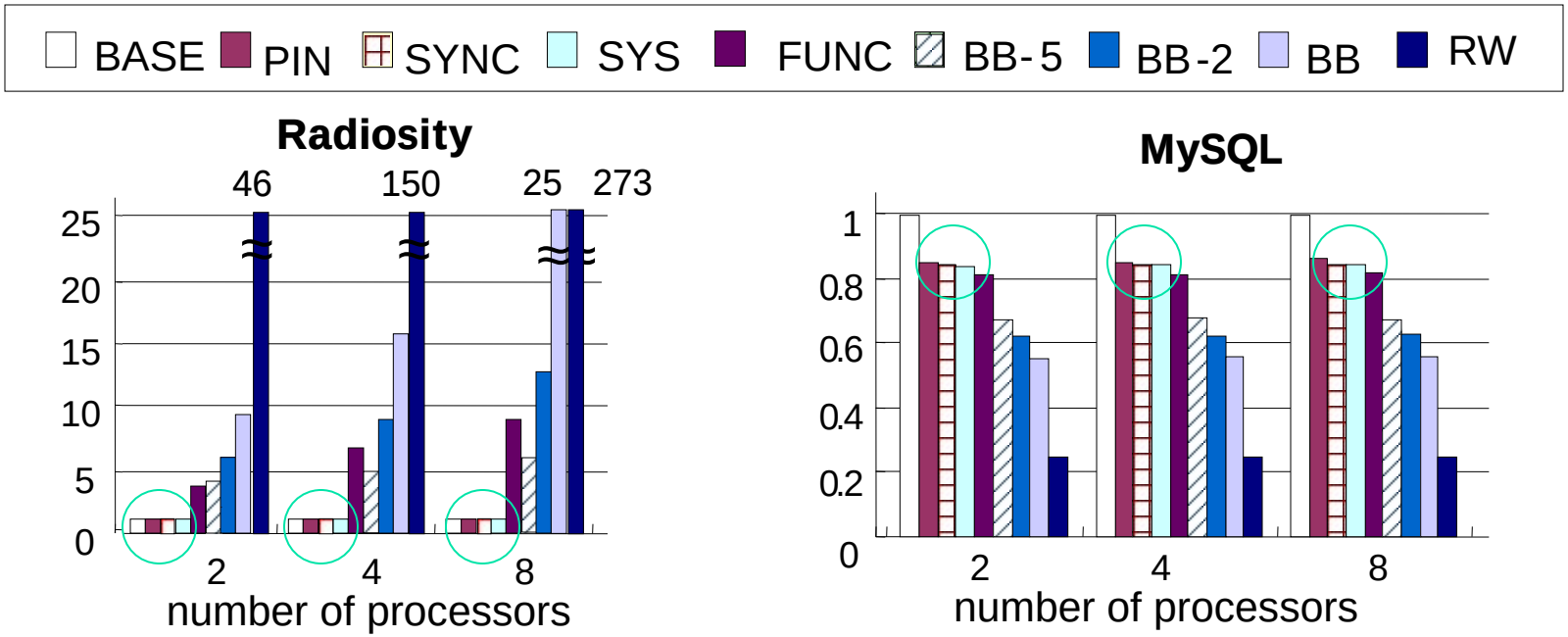
Bugs vs. Execution Reproduction

Application		SYNC	SYS	FUNC	BB-5	BB-2	BB
Mozilla	**BR**	3	3	3	4	4	3
	ER	**16**	**16**	4	4	4	3
PBZip2	**BR**	3	3	2	4	3	3
	ER	5	5	2	4	5	3

the number of replays for bug reproduction (BR)
vs. those for execution reproduction (ER)

- Reproducing the exact same execution path requires 1.6-5 times more attempts with SYS and SYNC

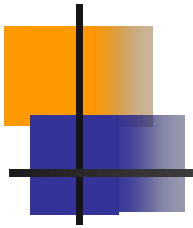
Scalability



Normalized elapse time
(the lower, the better)

Normalized throughput of a server application
(the higher, the better)

2-core : SYNC 6.2%, SYS 11.8%
(770% by SMP-Revirt)



Conclusions

- Software solutions can be practical for reproducing concurrency bugs on multiprocessor
- PRES: Probabilistic Replay via Execution Sketch
  - No need to reproduce the bug at the first attempt
  - Trade replay-time efficiency for lower recording overhead
  - PI-Replayer (that leverages feedback) makes low-overhead sketches (SYS, SYNC) useful for bug reproduction