Diagnosing Production-Run Concurrency-Bug Failures

Shan Lu

University of Wisconsin, Madison

Outline

- Myself and my group
- Production-run failure diagnosis
  - What is this problem
  - What are our solutions
    - CCI [OOPSLA'10]
    - PBI [ASPLOS'13]
    - LXR [ASPLOS'14]
- Conclusions

A little bit about myself

Shan 山　Lu 卢





The most exciting thing …





Software bugs

- How many of you have been bothered by bugs?

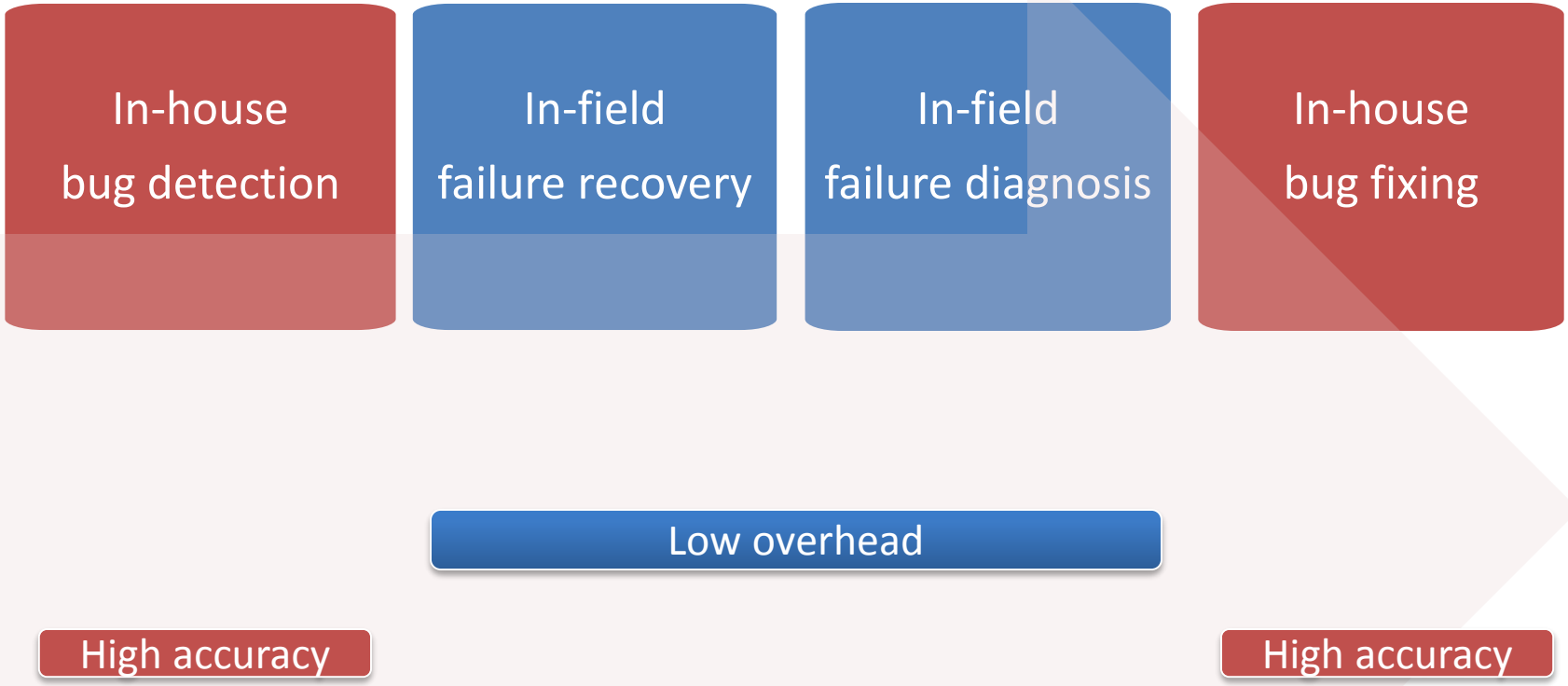


Fighting software bugs is crucial

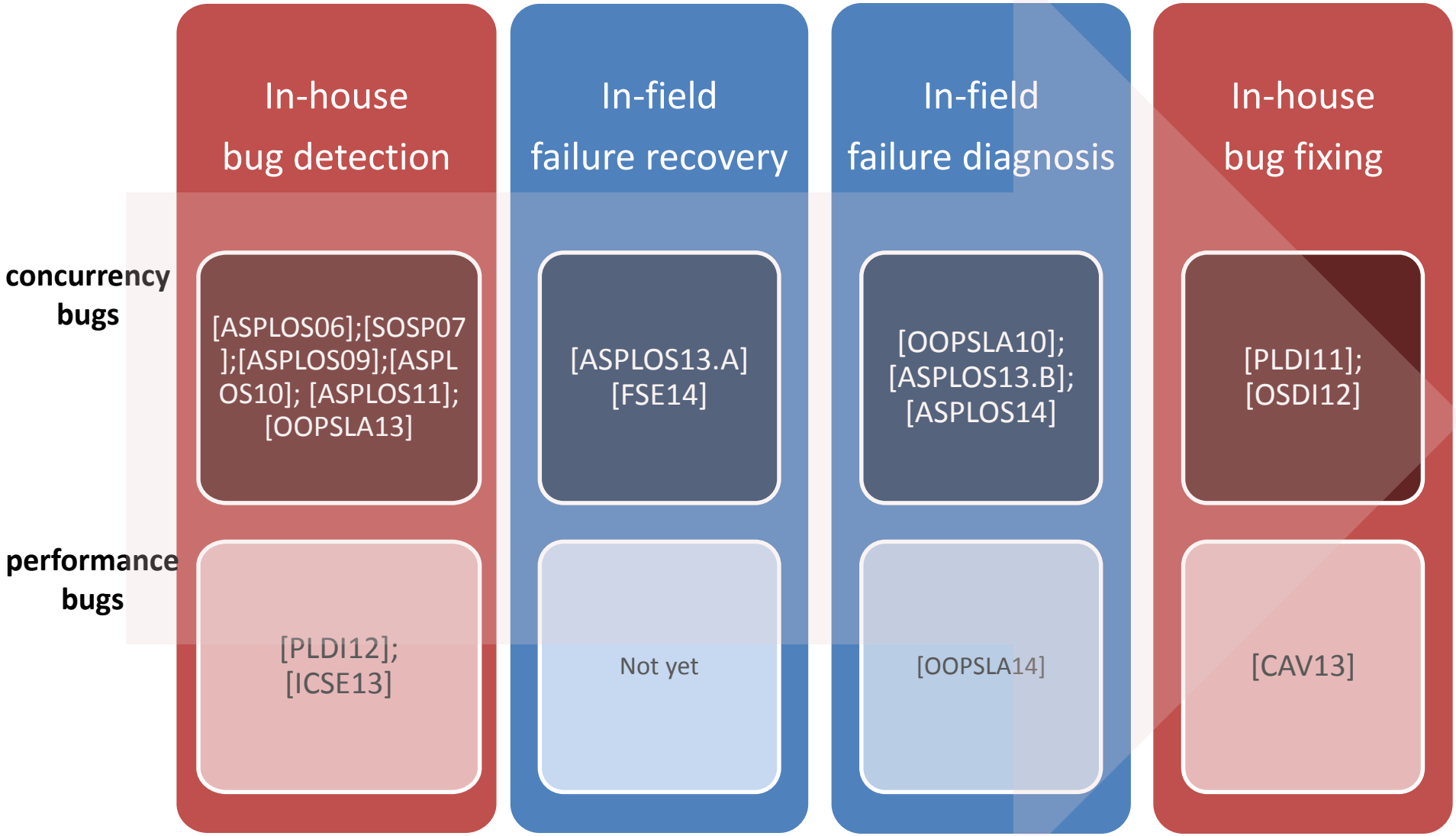
- Software is everywhere
  - http://en.wikipedia.org/wiki/List_of_software_bugs
- Software bugs are widespread and costly
  - Lead to 40% system down time [Blueprints 2000]
  - Cost 312 Billion lost per year [Cambridge 2013]



Different aspects of fighting bugs

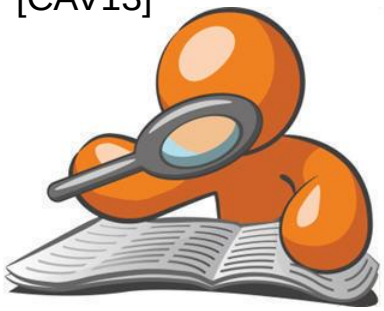


Work from my group



Our high-level approach

[SOSP07];[ASPLOS11];[OOPSLA10];
[PLDI11];[PLDI12];[OSDI12];[ASPLOS13.A];
[CAV13]



[ASPLOS06];[SOSP07];[ASPLOS09];
[OOPSLA10];[ASPLOS10];[ASPLOS11];
[ASPLOS13.B]; [ICSE13]; [OOPSLA13]



fault

failure

[ASPLOS08]
[PLDI12]



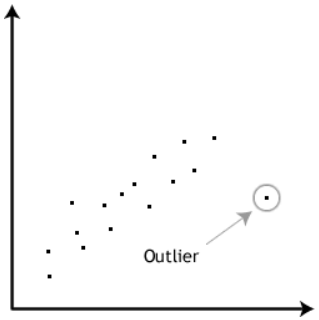


[ASPLOS10]
[ASPLOS11]

error

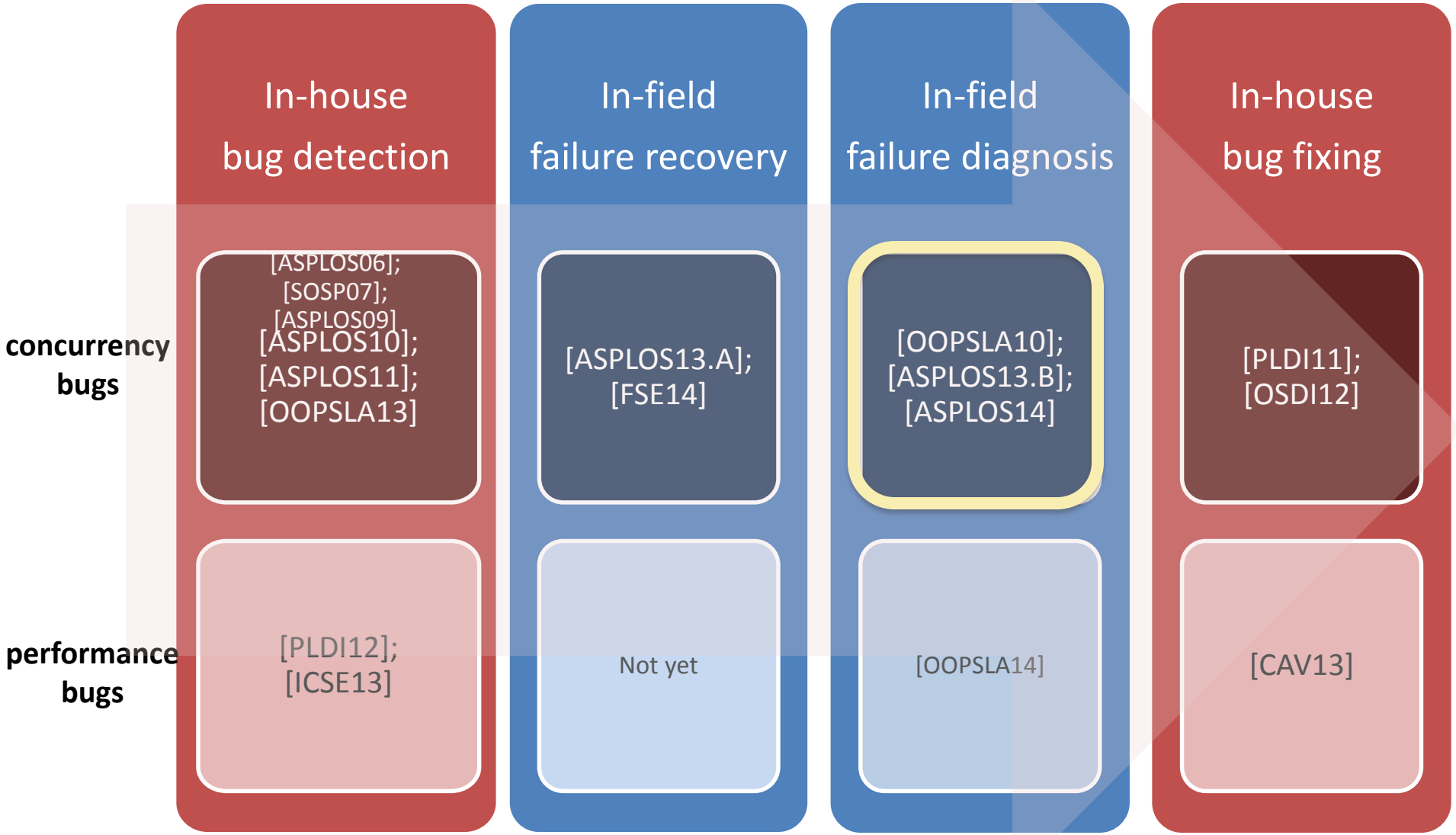


[ASPLOS06];[MICRO06];
[ASPLOS13.B];[ASPLOS14]



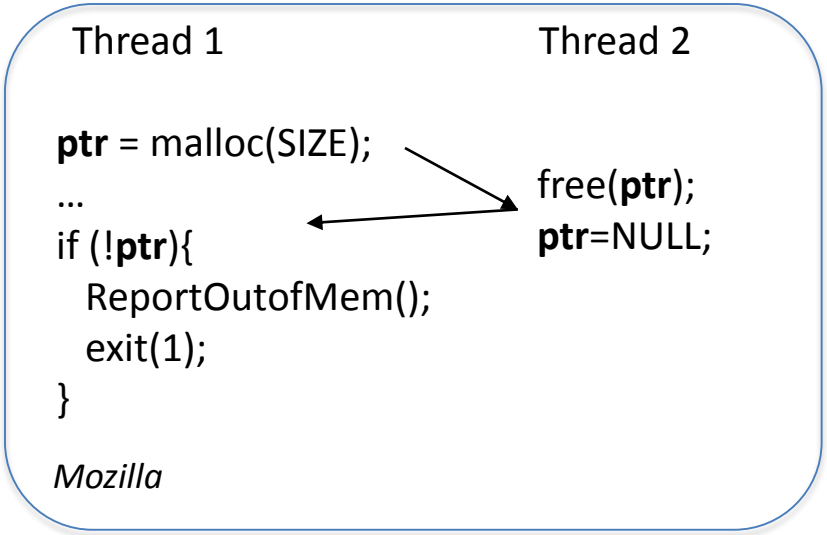
[ASPLOS06];[SOSP07];[OOPSLA10];
[ASPLOS13.B]; [ASPLOS14];[OOPSLA14]

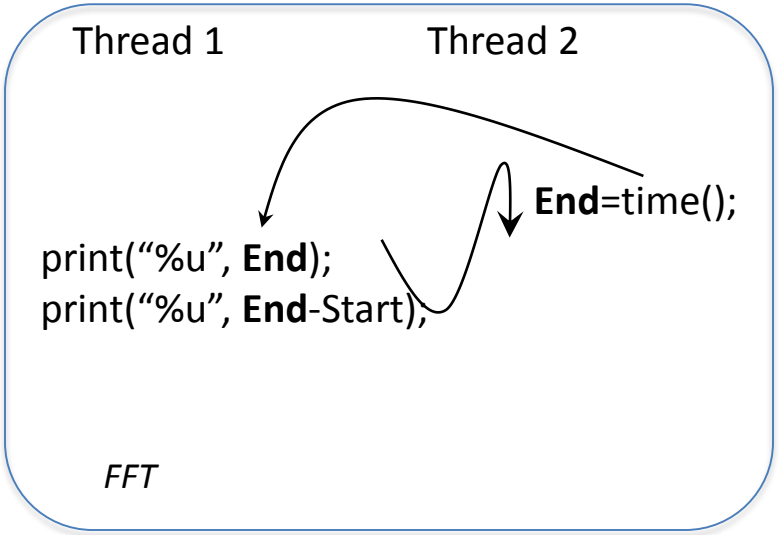
Focus of this talk



What are concurrency bugs?

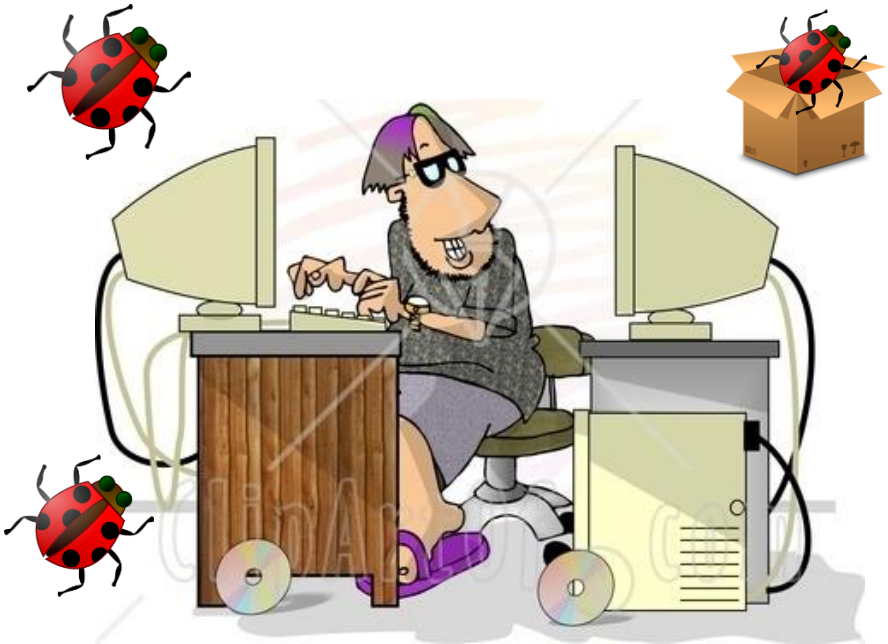
- Untimely accesses among threads (buggy interleavings)



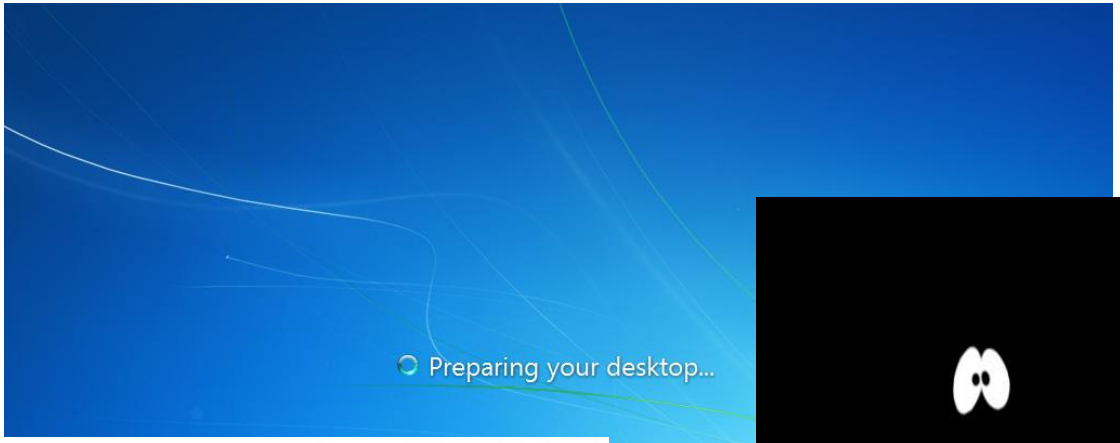


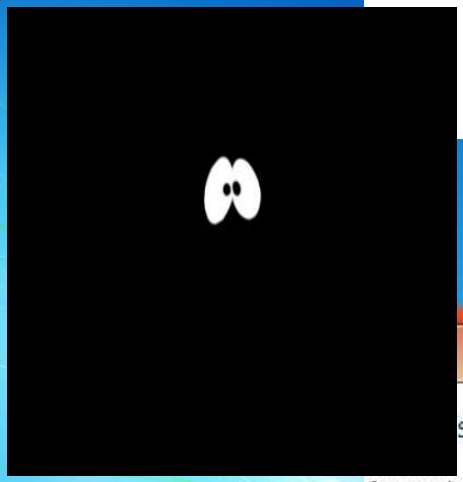
Con. bugs are common





Con. bugs manifest in the field







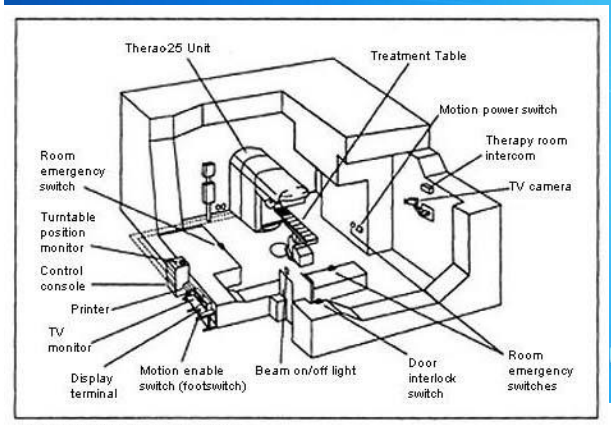


Figure 1. Typical Therac-25 facility



These failures need to be diagnosed before they can be fixed !

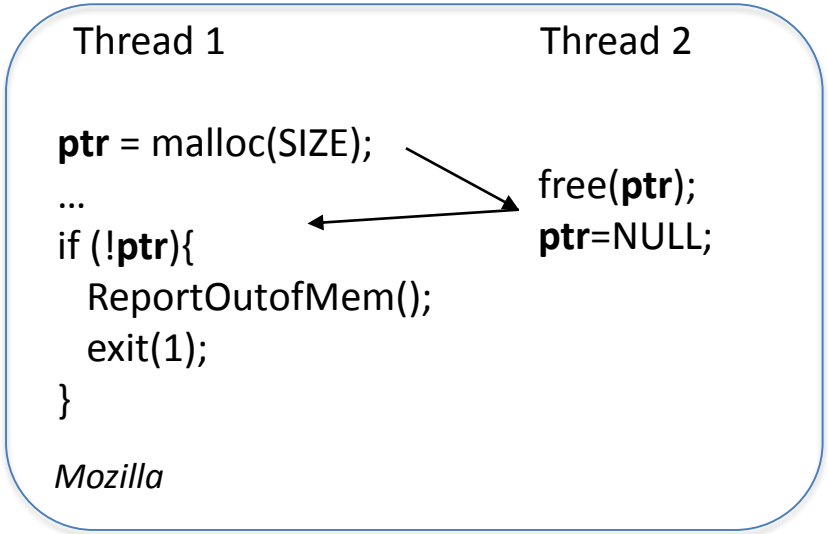
Failure diagnosis is challenging

- Limited information
- Failures are difficult to repeat
- Root causes are difficult to reason about





Example


Thread 1
Thread 2
ptr = malloc(SIZE);
…
if (!ptr){
ReportOutofMem();
exit(1);
}
free(ptr);
ptr=NULL;
Mozilla

Example

```
InitState(...){
    table = New();

    if (table == NULL) {
        ReportOutOfMemory();
        return JS_FALSE;
    }
}
```

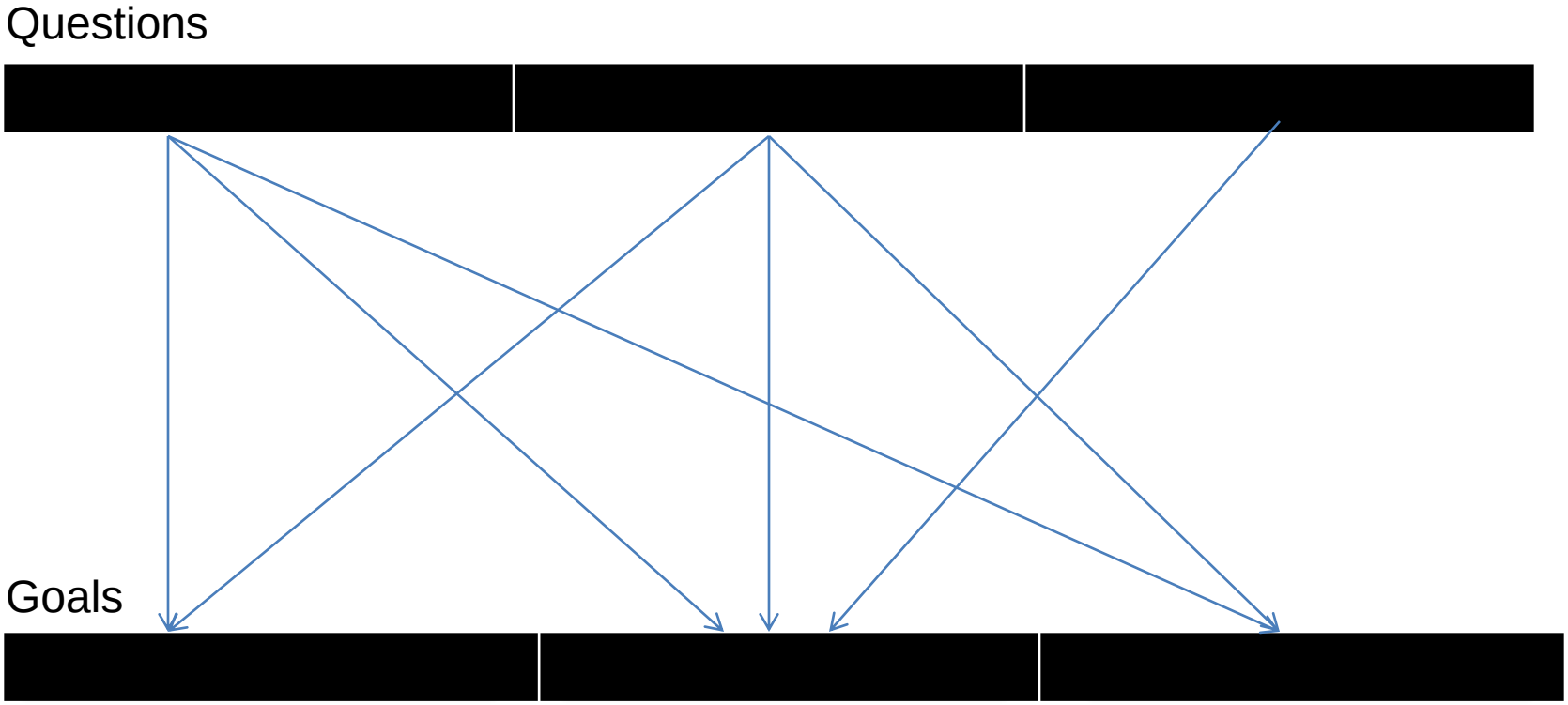
```
ReportOutOfMemory(){
  error("out of memory");
}
```

****.js*
out of memory

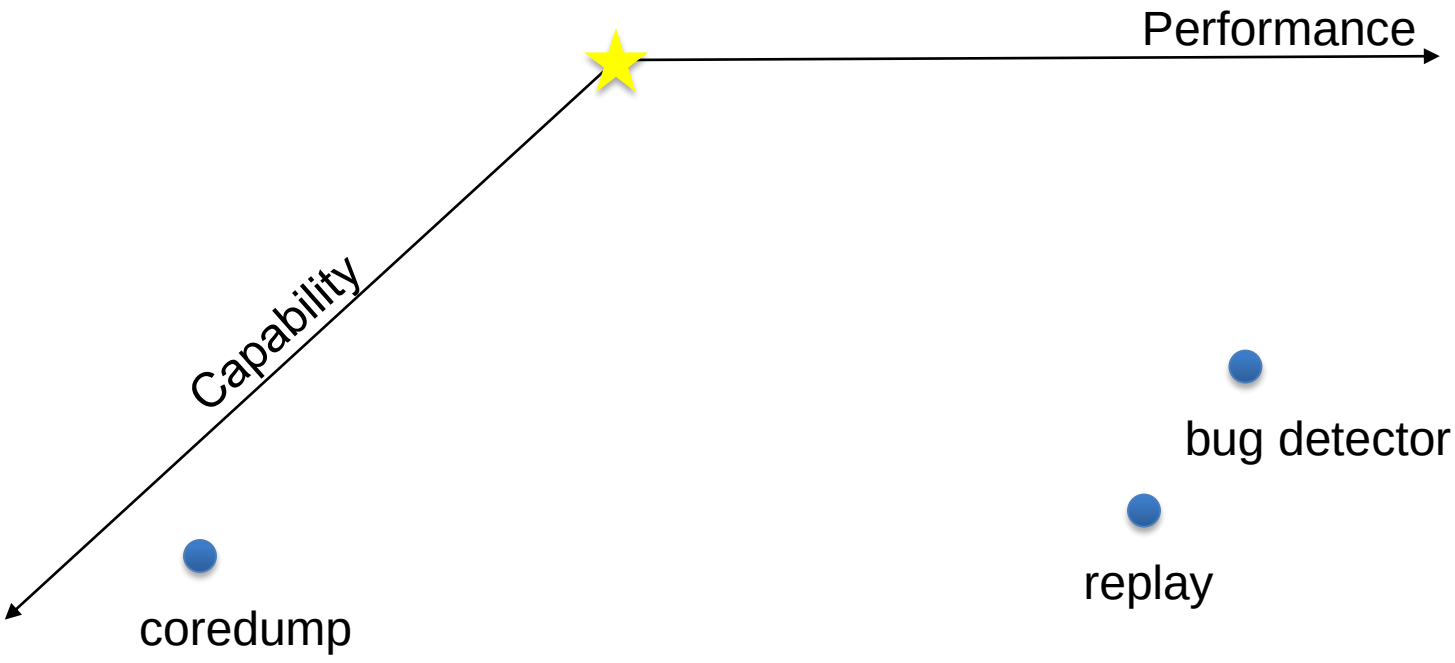


CALL STACK
ReportOutofMemory()
InitState()
…
main()

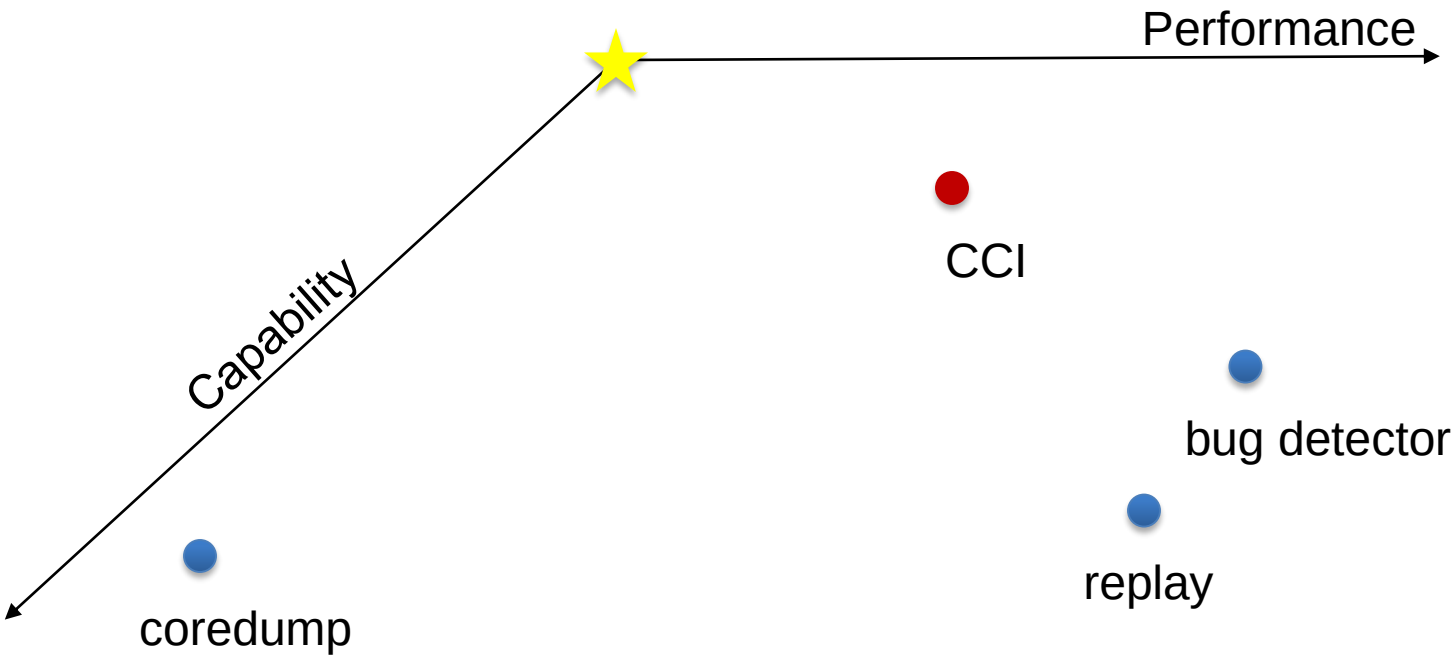
Design space



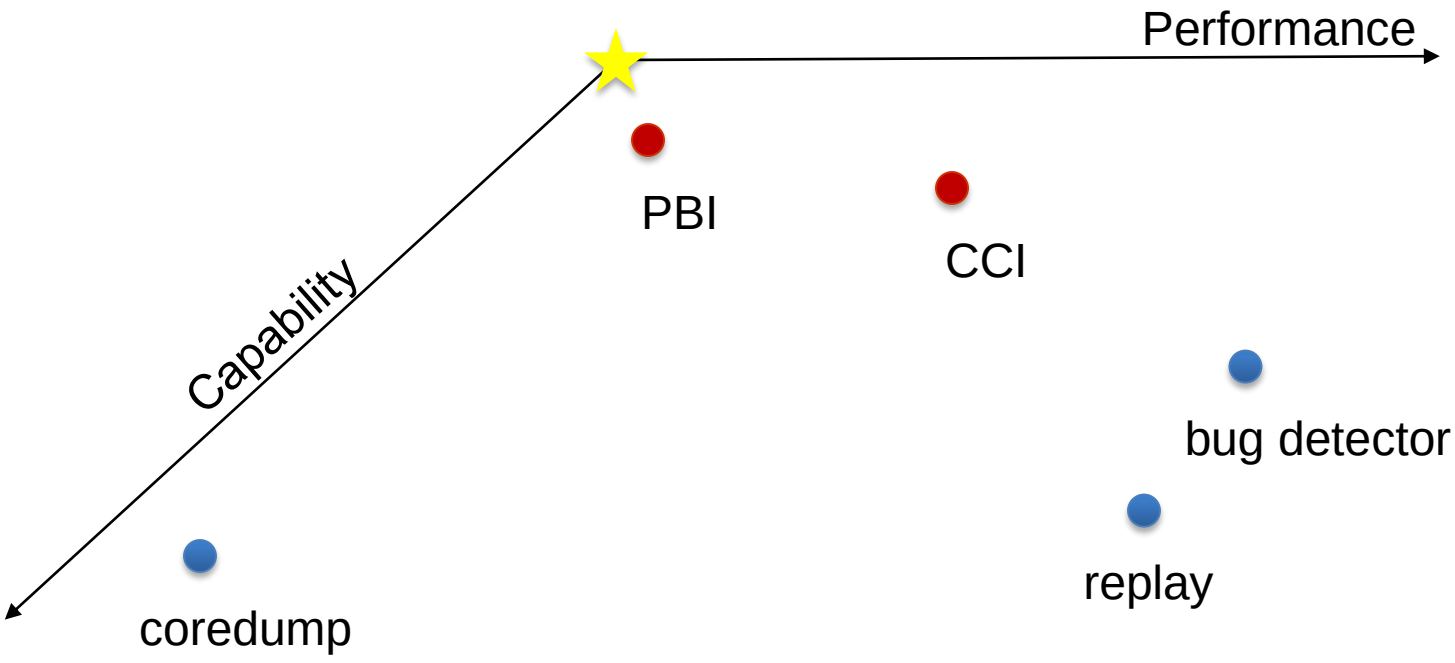
Previous work



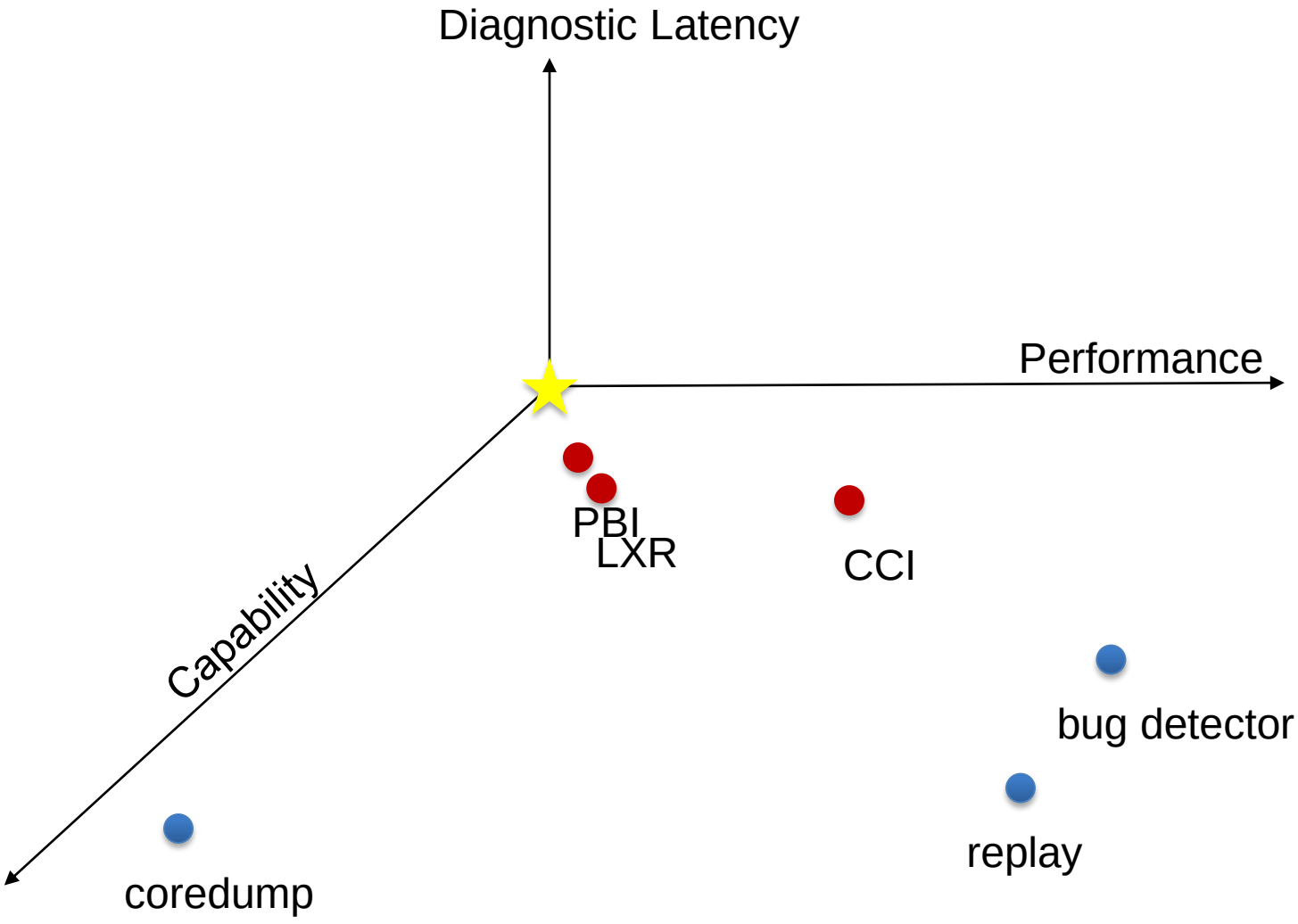
Our work



Our work

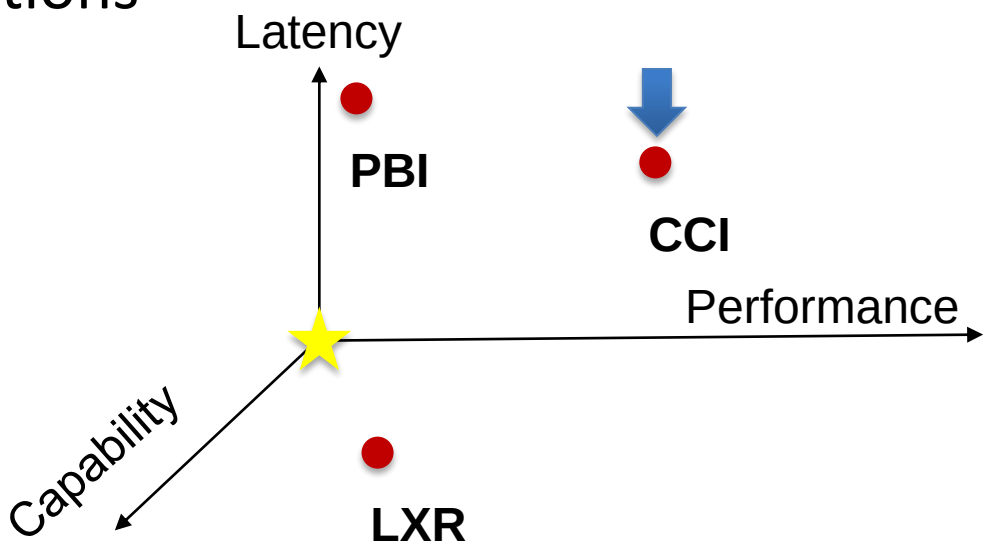

Performance
Capability
PBI
CCI
bug detector
replay
coredump

Our work


Diagnostic Latency
Performance
Capability
PBI
LXR
CCI
bug detector
replay
coredump

Outline

- Myself & my group
- Production-run failure diagnosis
  - What is the problem
  - What are our solutions



- Conclusion

How to do better than state-of-art?

What to collect	How to collect	How to use the collected

Performance	Capability	Latency

How to do better than state-of-art?

What to collect	How to collect	How to use the collected
	Sampling	

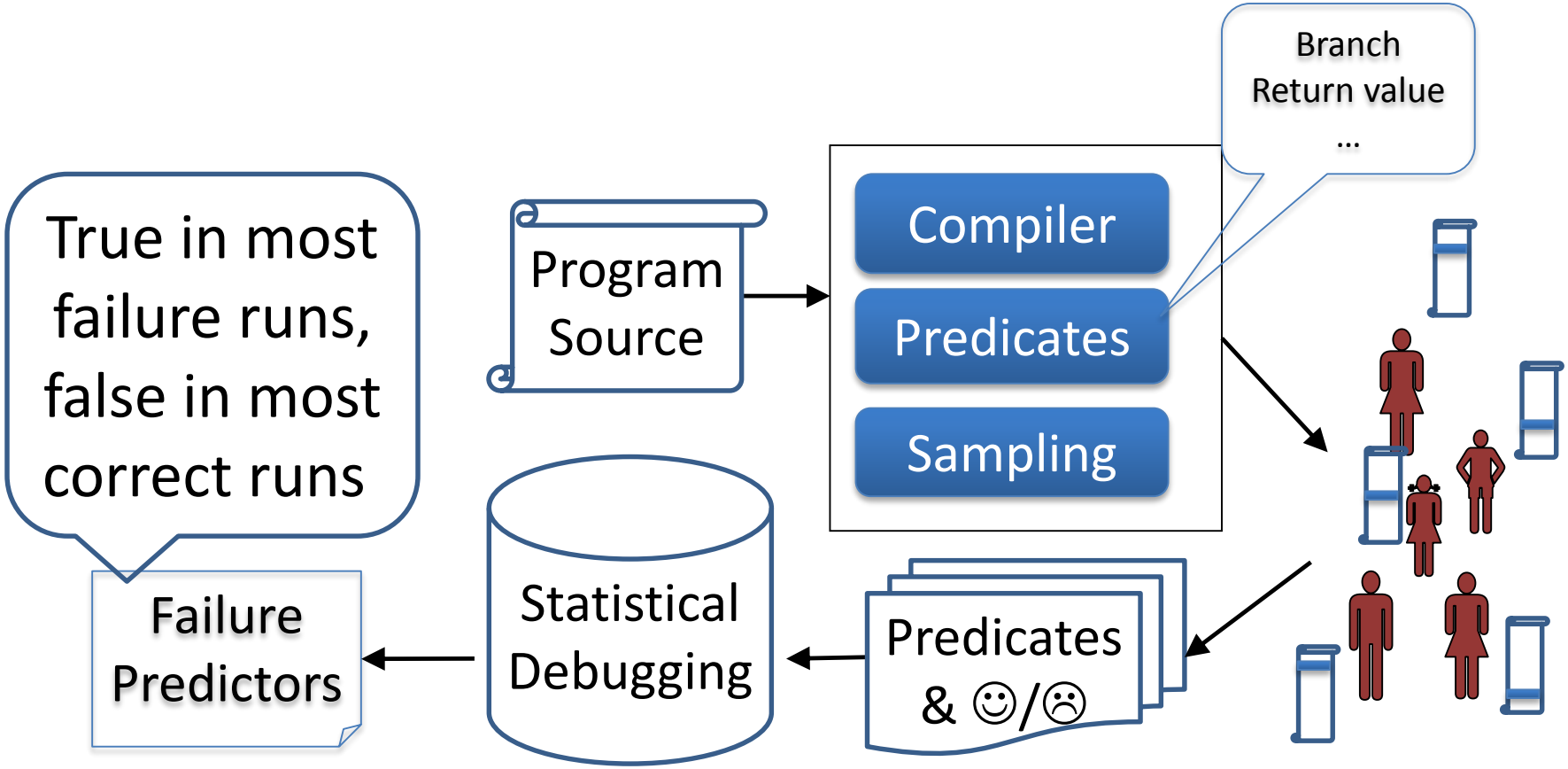
Performance	Capability	Latency

How to do better than state-of-art?

What to collect	How to collect	How to use the collected
	Sampling	Cooperative statistical analysis

Performance	Capability	Latency

Cooperative Bug Isolation (CBI)



Performance	Capability
Good	??

Does it work for concurrency bugs?



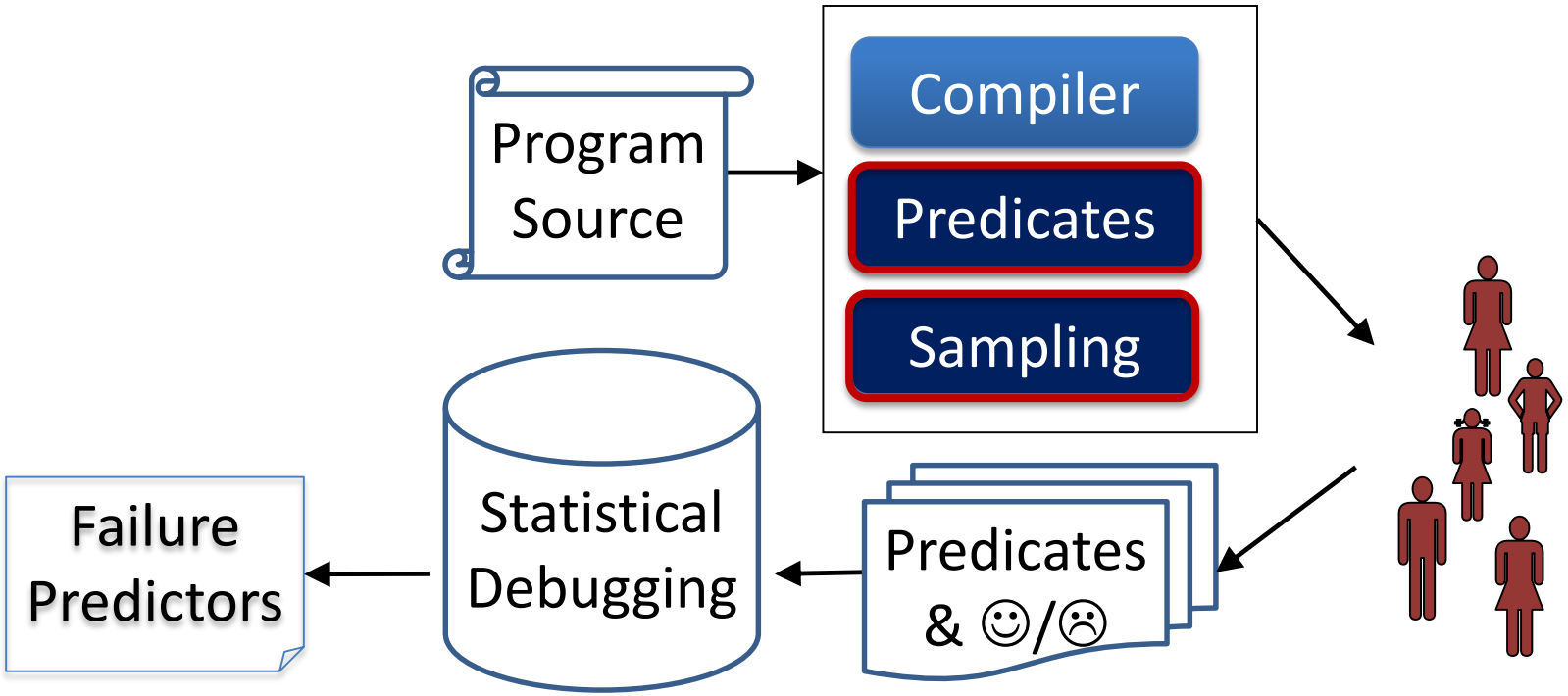
```
Thread 1                          Thread 2

ptr = malloc(SIZE);
…                                 free(ptr);
if (!ptr){ //b                    ptr=NULL;
   ReportOutofMem();
   exit(1);
}
```

Predicate
…
$taken_b$
$!taken_b$
…

Why does CBI not work?

Cooperative Con-Bug Isolation (CCI)



Performance	Capability
Mixed	Good

Instrumentation and Sampling Strategies for Cooperative Concurrency Bug Isolation, OOPSLA'10

What to collect? (predicate design)

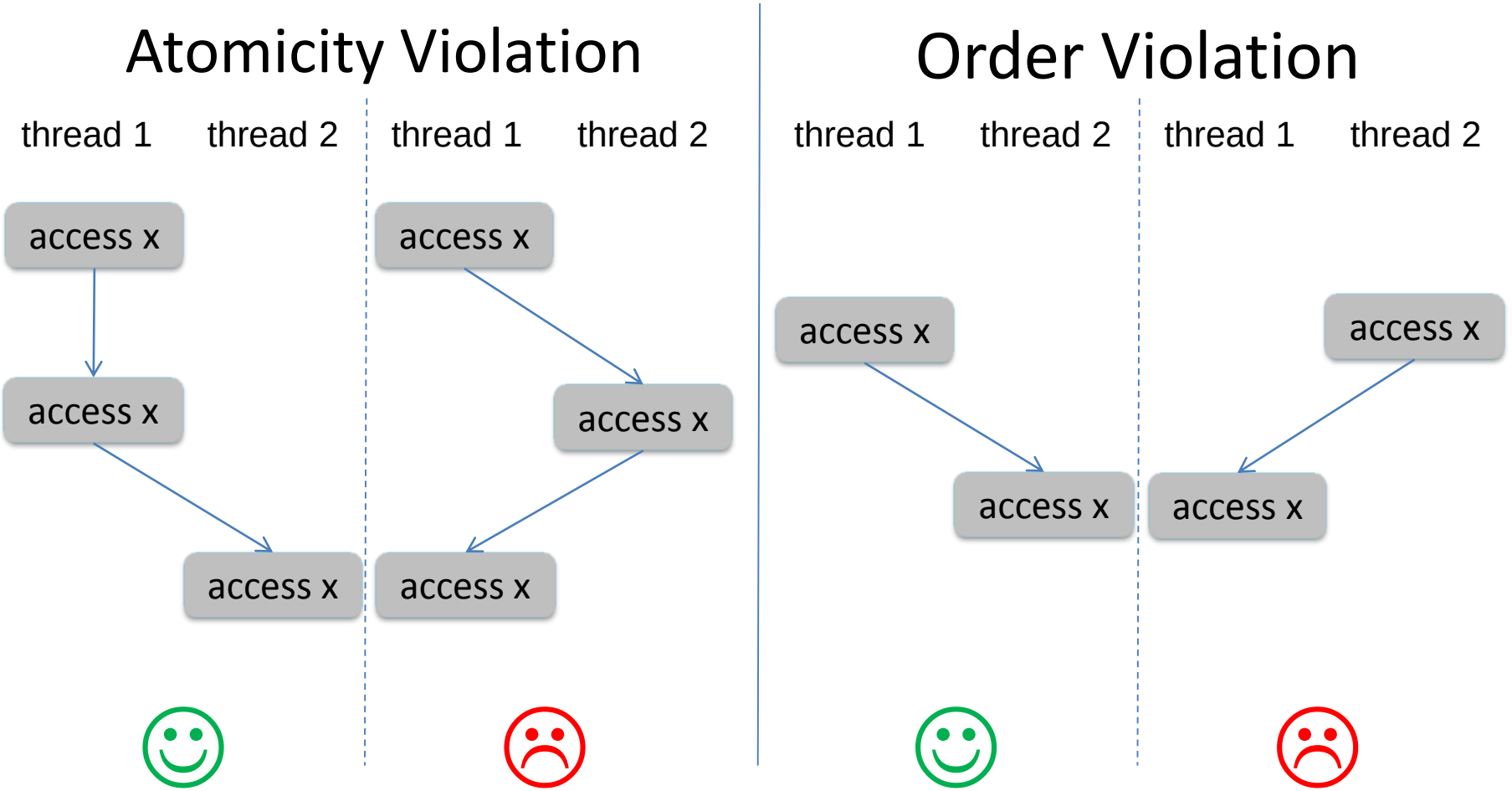
Concurrency bug root cause patterns

Atomicity Violation

Order Violation

Learning from Mistakes --- A Comprehensive Study on Real World Concurrency Bug Characteristics, ASPLOS'08

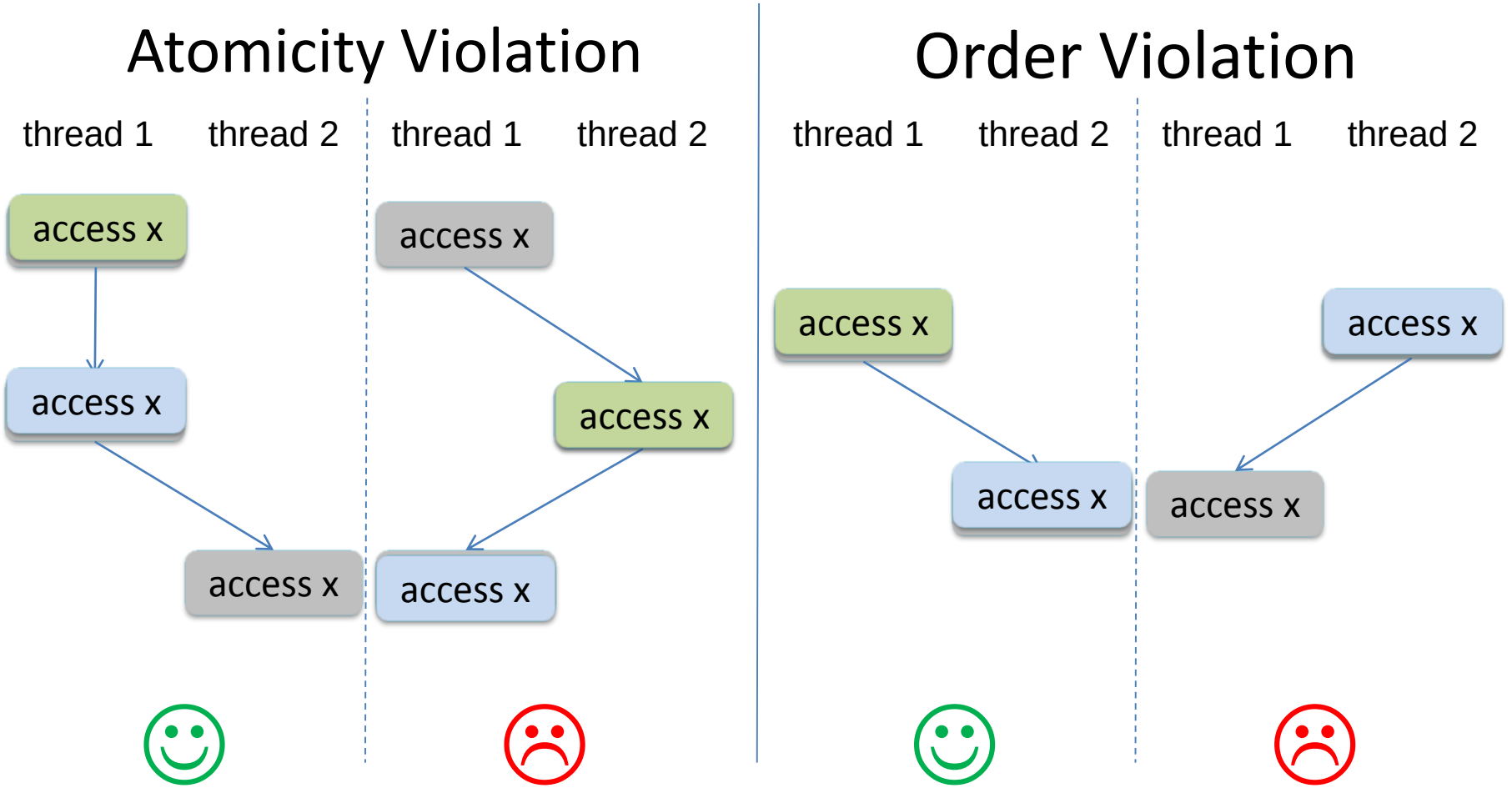
Concurrency bug root cause patterns



CCI-Prev predicate

Whether two successive accesses
to a memory location were by
two distinct threads
or one thread

CCI-Prev can reflect root causes



Is CCI-Prev useful? (Example)

Thread 1	Thread 2

ptr = malloc(SIZE);
…
if (!**ptr**){
  ReportOutofMem();
  exit(1);
}
	free(**ptr**);
	ptr=NULL;

Mozilla

Example (correct runs)

thread 1

```
…
ptr = malloc (SIZE);
…
l  if (!ptr) {
     ReportOutofMem();
     exit(1);
   }
```

thread 2

```
…
free (ptr);
ptr=NULL;
…
```



Predicate	☺	☹
…		
$remote_l$	0	0
$local_l$	1	0
…		



Example (failure run)

thread 1

…
ptr = malloc (SIZE);

thread 2

…
free (**ptr**);
ptr=NULL;
…

Predicate	☺	☹
…		
$remote_l$	0	1
$local_l$	1	0
…		

…
l if (!**ptr**) {
ReportOutofMem();
exit(1);
}



How to evaluate?

thread 1

…
ptr = malloc (SIZE);

thread 2

…
free (**ptr**);
ptr=NULL;
…

Predicate	☺	☹
…		
$remote_l$	0	1
$local_l$	1	0
…		

lock(glock);
remote = test_and_insert(& ***ptr***, curTid);
record(l, remote);
l temp = ***ptr***;
unlock(glock);

```
if (!temp) {
  ReportOutofMem();
  exit(1);
}
```

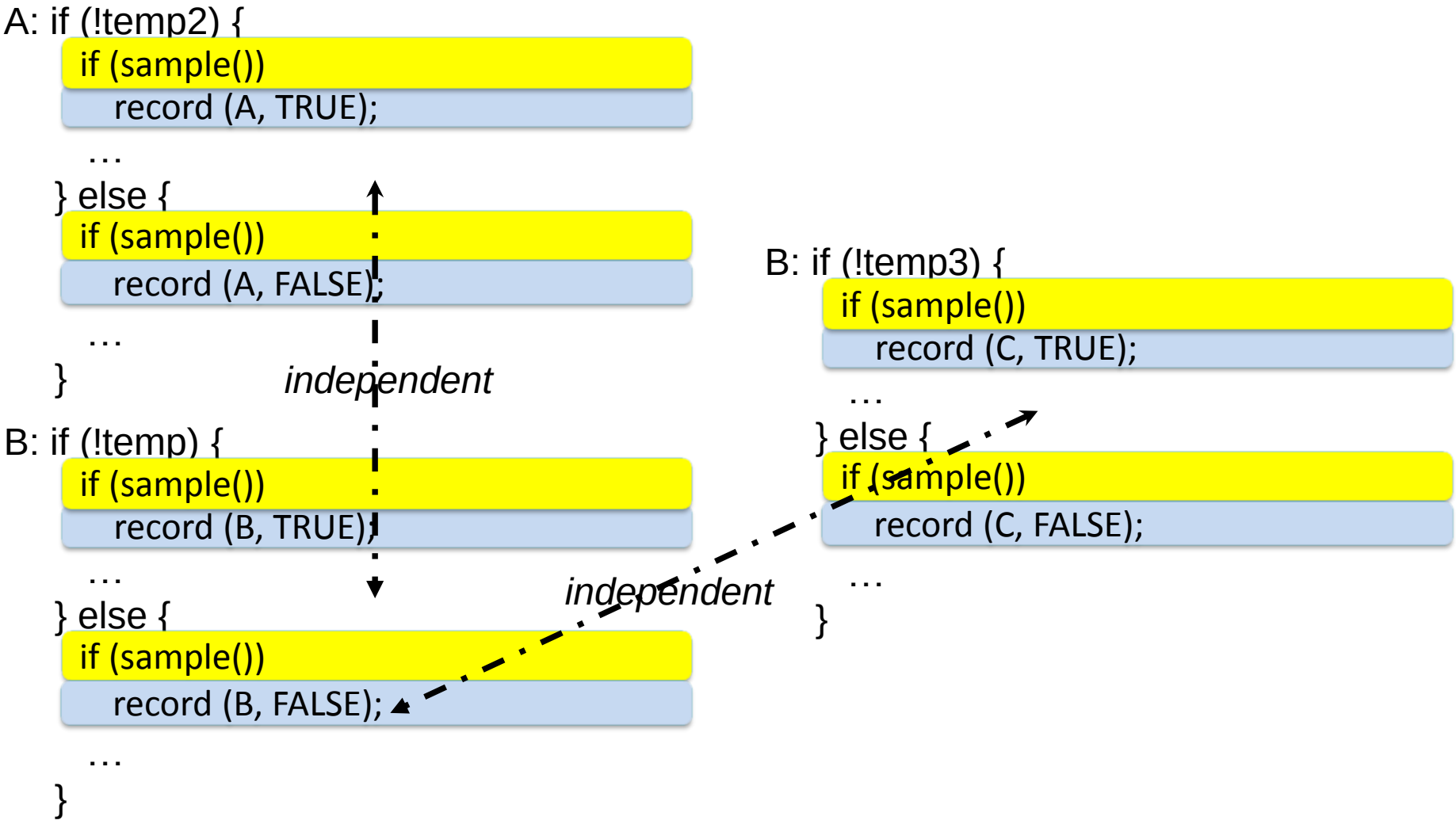
a global hash table



address	ThreadID
…	…
& ***ptr***	1
…	…

How to sample?

How to sample branch predicates?



How to sample CCI-Prev?

thread 1

```
…
ptr = malloc (SIZE);
```

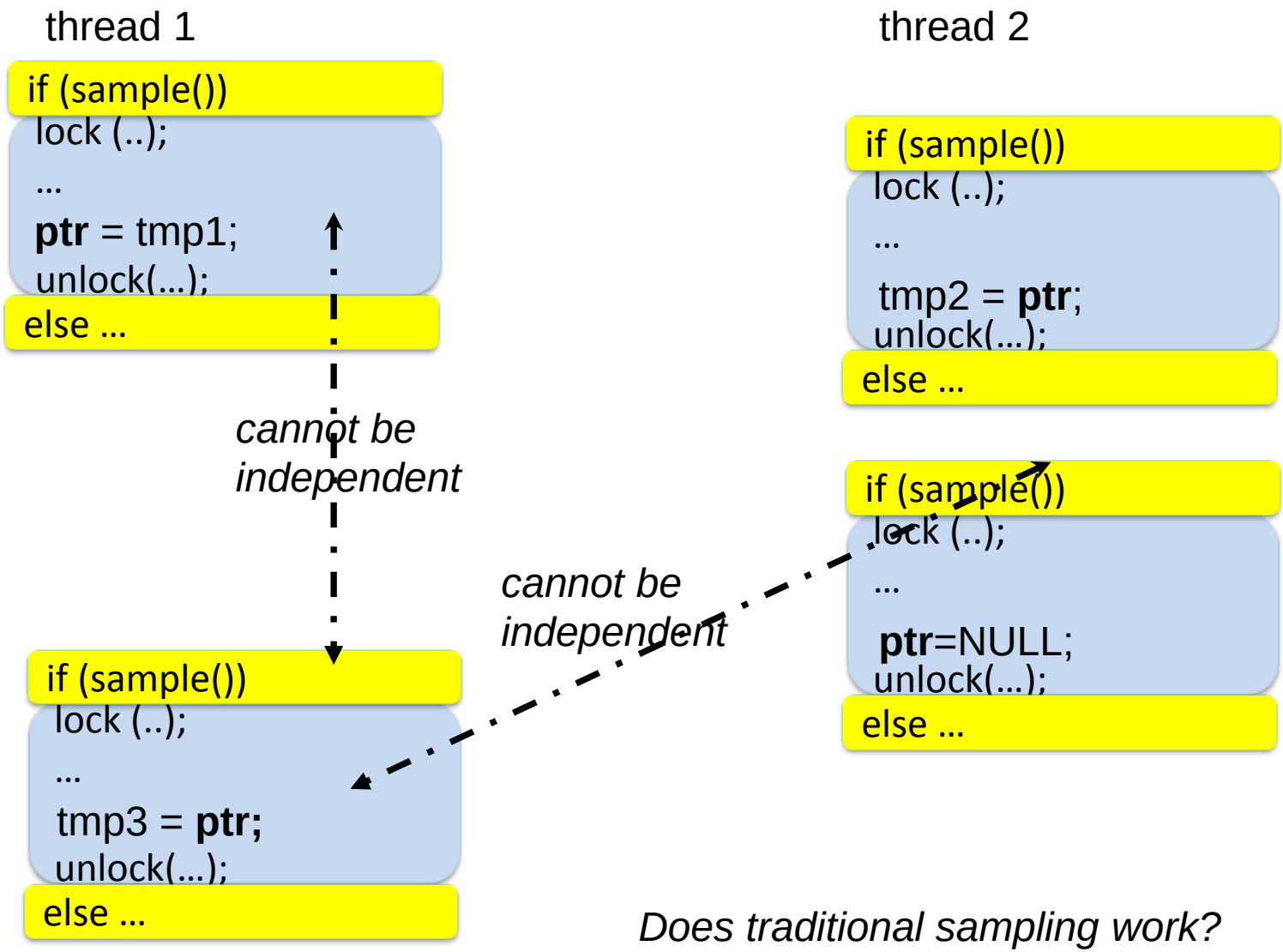
thread 2

```
…
free (ptr);
ptr=NULL;
…
```

```
…
if (!ptr) {
   ReportOutofMem();
   exit(1);
}
```

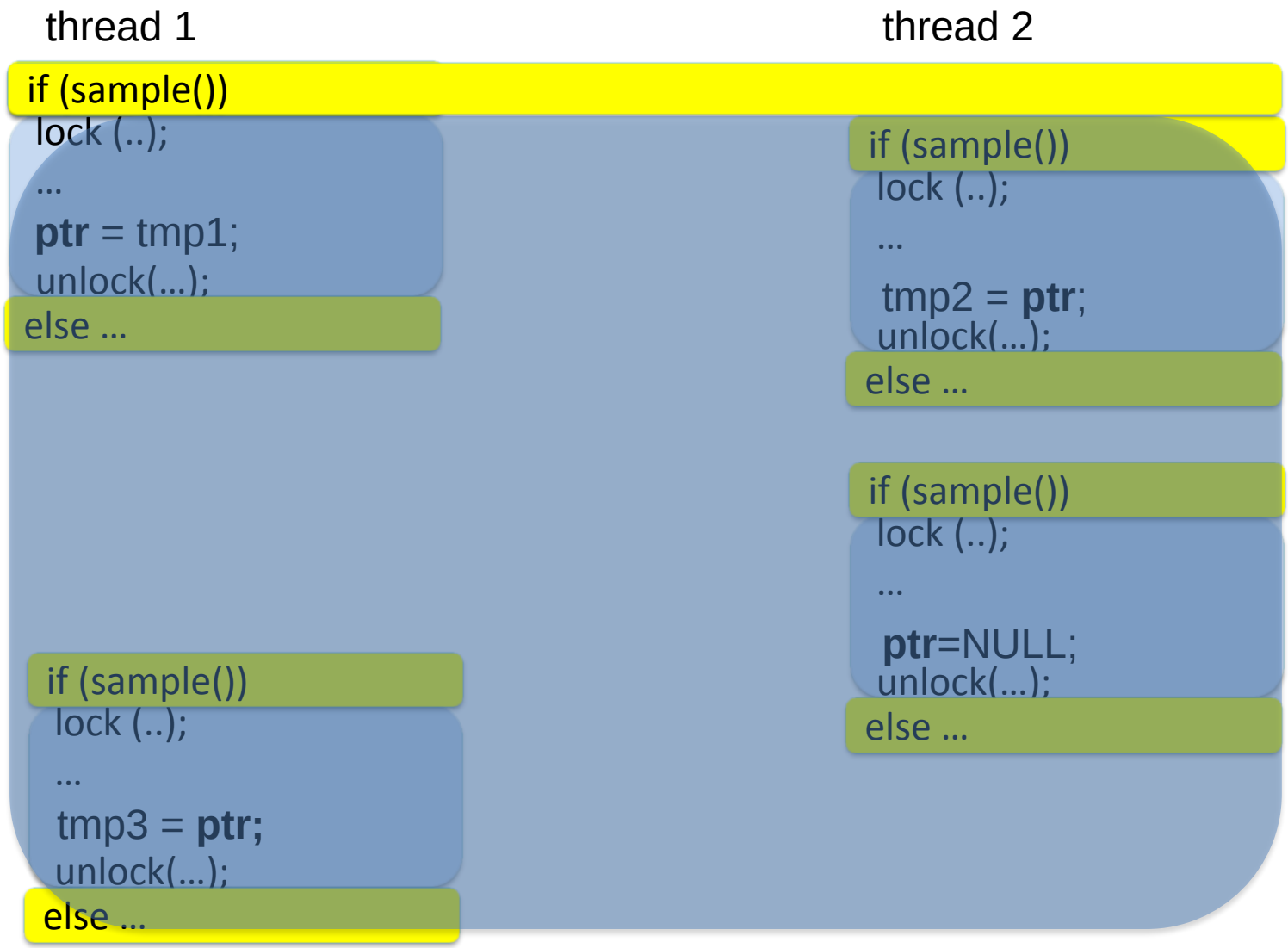
Does traditional sampling work?

How to sample CCI-Prev?

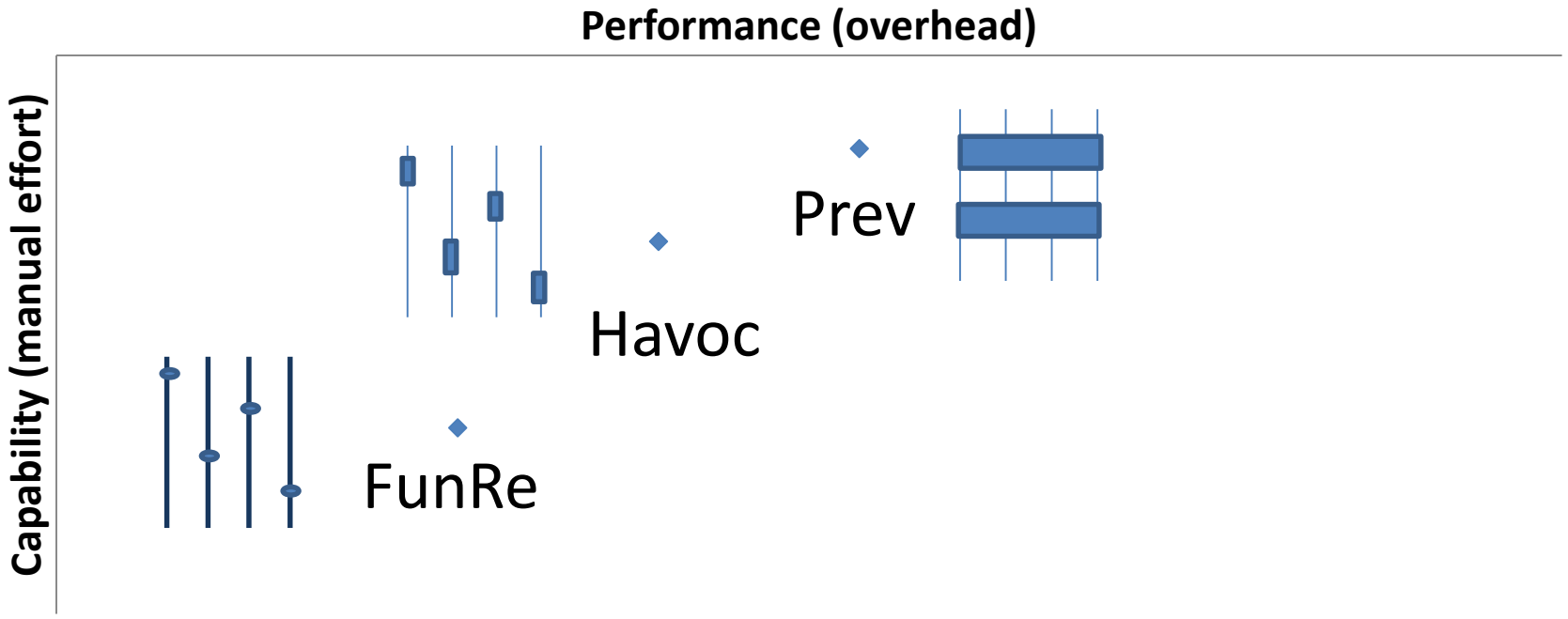


Does traditional sampling work? **NO!**

Thread-coordinated, bursty sampling



Other predicates



Evaluation methodology

Program
Apache-1
Apache-2
Cherokee
FFT
LU
Mozilla-JS-1
Mozilla-JS-2
Mozilla-JS-3
PBZIP2

CIL-based static code instrumentor

1/100 sampling rate, ~3000 runs in total (failure:success~1:1)

Diagnosis capability (w/ sampling)

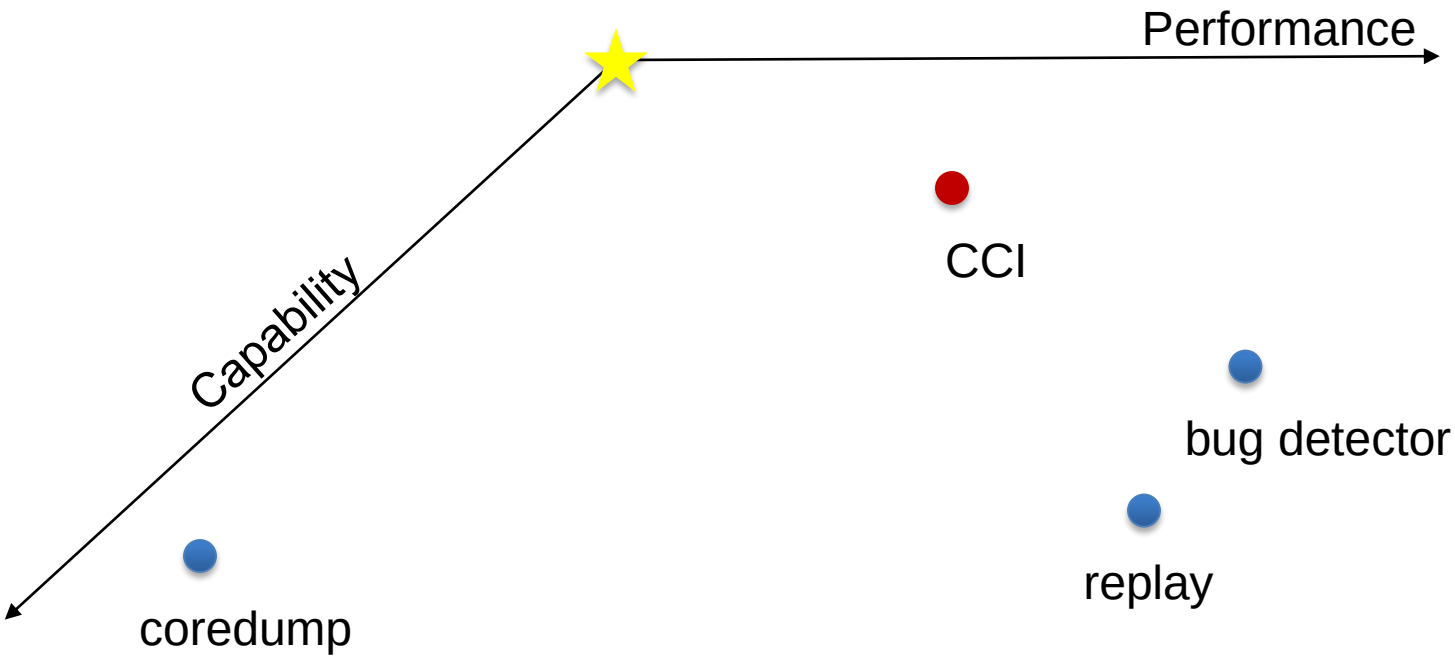
Program	CCI-Prev
Apache-1	✓ top1
Apache-2	✓ top1
Cherokee	✗
FFT	✓ top1
LU	✓ top1
Mozilla-JS-1	✗
Mozilla-JS-2	✓ top1
Mozilla-JS-3	✓ top2
PBZIP2	✓ top1

1/1000 sampling rate, ~3000 runs in total (failure:success~1:1)

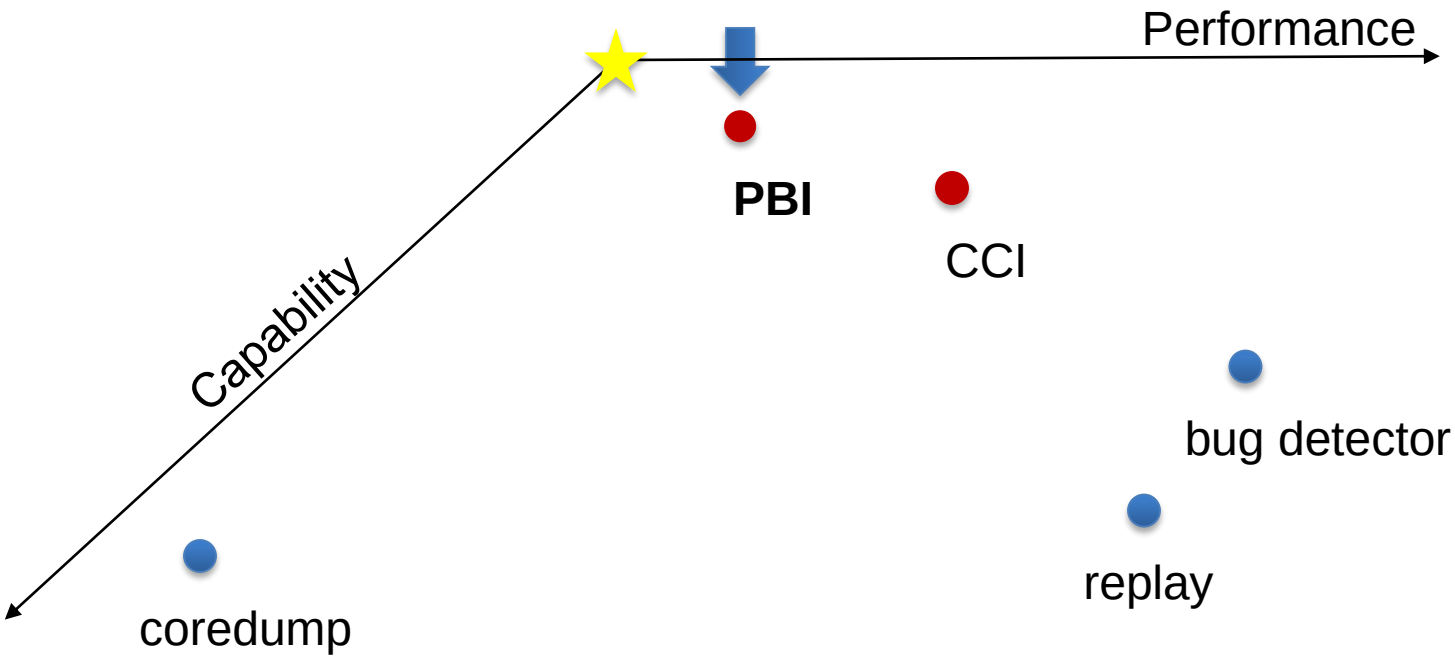
Diagnosis performance (overhead)

	Prev	
	No Sampling	Sampling
Apache-1	62.6%	**1.9%**
Apache-2	**8.4%**	**0.5%**
Cherokee	19.1%	**0.3%**
FFT	169 %	24.0%
LU	57857 %	949 %
Mozilla-JS	11311 %	606 %
PBZIP2	**0.2%**	**0.2%**

Are we done?



Outline


Performance
PBI
CCI
Capability
bug detector
replay
coredump

How to do better than CCI?

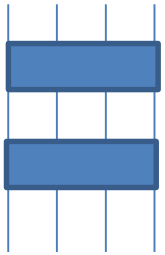
What to collect	How to collect	How to use the collected
CCI-Prev …	Sampling	Cooperative statistical analysis

Performance	Capability	Latency

How to do better than CCI?

What to collect	How to collect	How to use the collected
	Sampling	

Slow sampling infrastructure

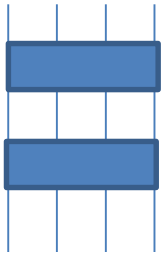


Performance	Capability	Latency

How to do better than CCI?

What to collect	How to collect	How to use the collected
	Sampling	

Slow sampling infrastructure
Inaccurate evaluation

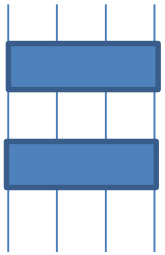


Performance	Capability	Latency

How to do better than CCI?

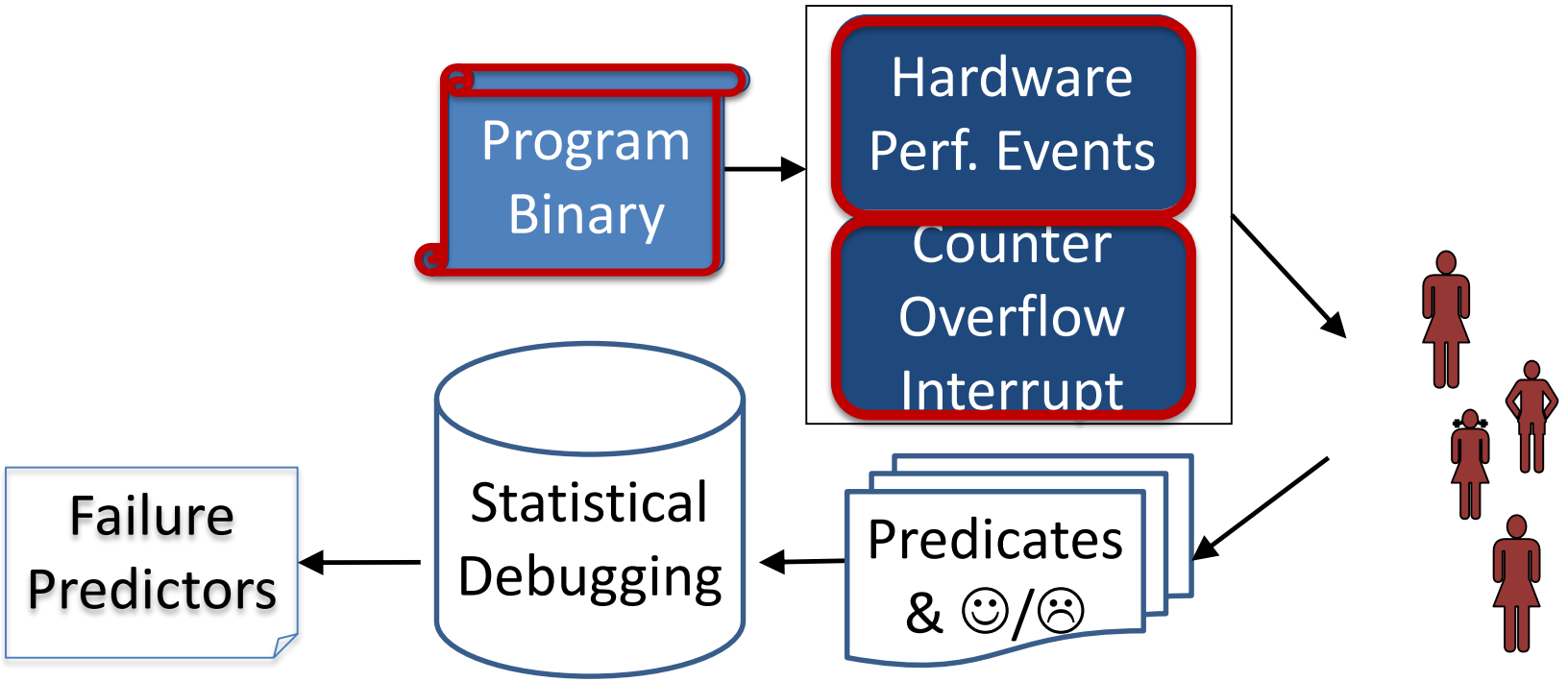
What to collect	How to collect	How to use the collected
	Hardware-based evaluation & sampling	

~~Slow sampling infrastructure~~
~~Inaccurate evaluation~~



Performance	Capability	Latency

PerfCnt-based Bug Isolation (PBI)



Performance	Capability	Code Size	Change Hardware?
Good (<5% overhead)	Good	No Change	NO!

Production-Run Software Failure Diagnosis via Hardware Performance Counters, ASPLOS'13

Hardware Performance Counters

- Registers monitor hardware performance events
  - 1—8 registers per core
  - Each register can contain an event count
  - Large collection of hardware events
    - Instructions retired, TLB misses, cache misses, etc.
- Traditional usage
  - Hardware testing/profiling

How can this help diagnose software failures?

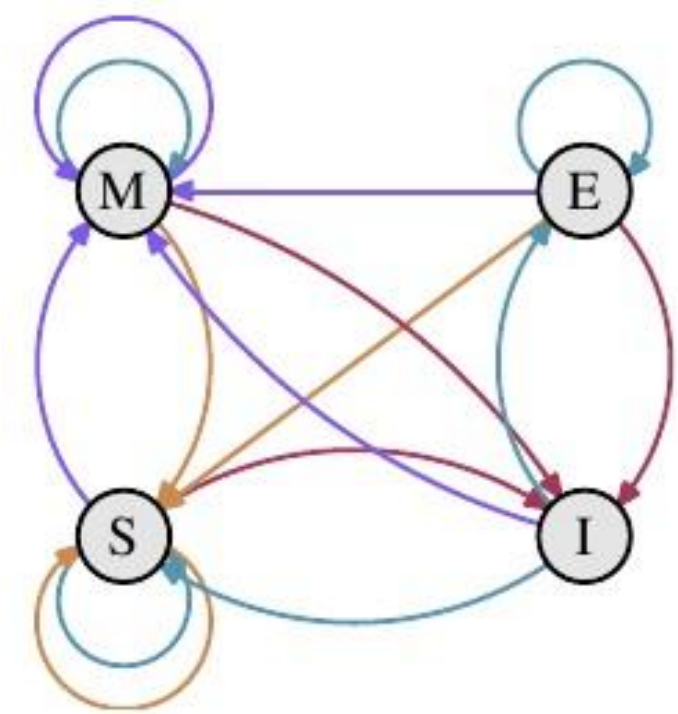
What to collect?

Which event can reflect root causes?

- L1 data cache cache-coherence events

It tracks which cache-coherence state (M/E/S/I) an instruction observes

Modified
Exclusive
Shared
Invalid



Local read
Local write
Remote read
Remote write

Is cache-coherence event useful?

Thread 1	Thread 2

ptr = malloc(SIZE);
…
if (!**ptr**){
  ReportOutofMem();
  exit(1);
}
Mozilla
	free(**ptr**);
	ptr=NULL;

Example (correct runs)

thread 1 (core 1)

Modified

```
…
ptr = malloc (SIZE);
…
I: if (!ptr) {
     ReportOutofMem();
     exit(1);
   }
```

thread 2 (core 2)

Invalid

```
…
free (ptr);
ptr=NULL;
…
```

Predicate	☺	☹
…		
M_I	1	0
E_I	0	0
S_I	0	0
I_I	0	0
…		



Example (failure run)

thread 1 (core 1)

Invalid

ptr = malloc (SIZE);

thread 2 (core 2)

Modified

…
free (**ptr**);
ptr=NULL;
…

Predicate	☺	☹
…		
M_I	1	0
E_I	0	0
S_I	0	0
I_I	0	1
…		



…
if (!**ptr**) {
ReportOutofMem();
exit(1);
}



Concurrency Bug from Apache HTTP Server

Useful for Atomicity Violations

Bug Type	FAILURE PREDICTOR
WWR Violation	INVALID
RWR Violation	INVALID
RWW Violation	INVALID
WRW Violation	SHARED

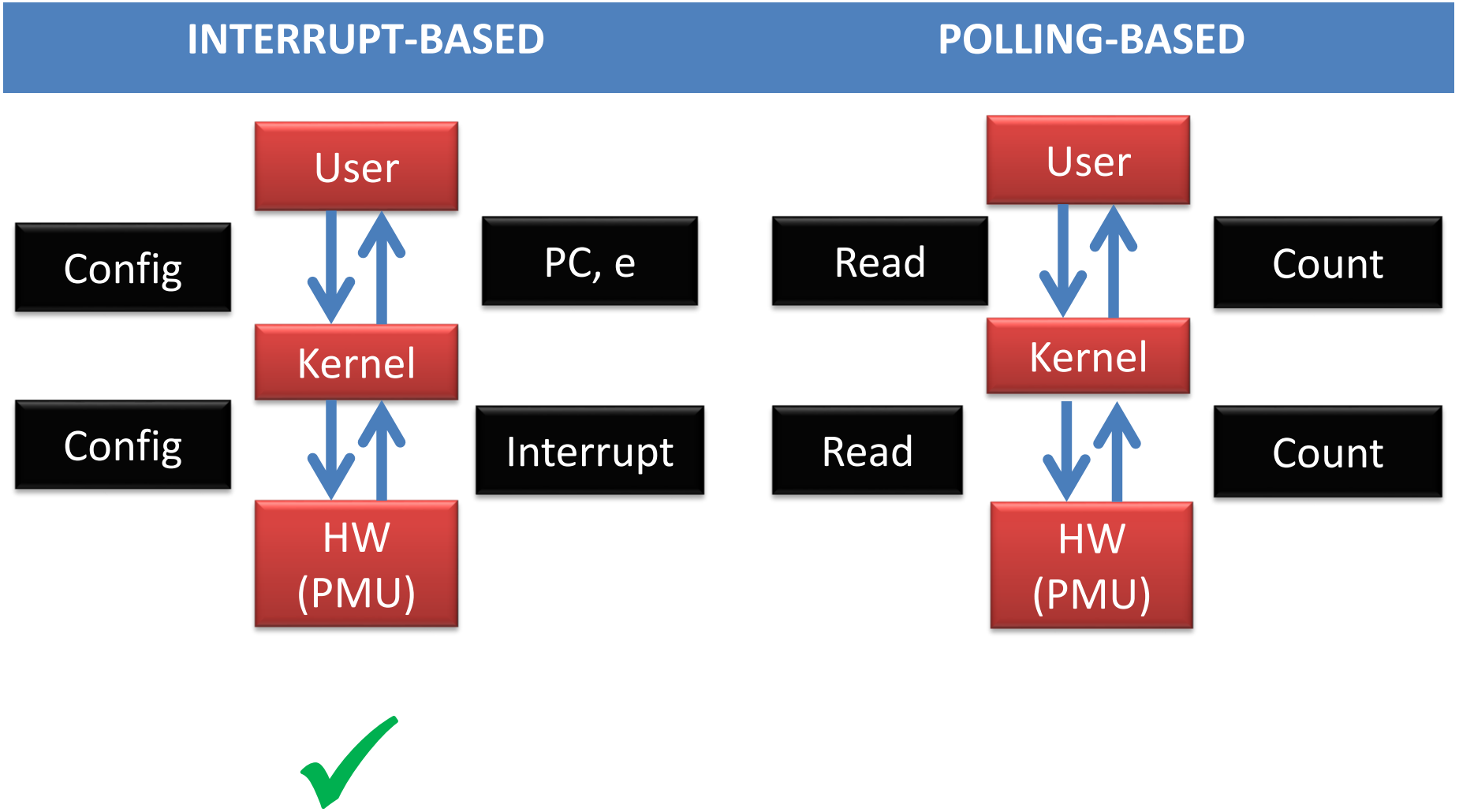
Useful for order violations

Bug Type	FAILURE PREDICTOR
Read-too-early	EXCLUSIVE (!INVALID)
Read-too-late	INVALID

How to evaluate & sample?

Which performance events occur at a specific instruction?

Accessing performance counters



More details of counter access

```
perf record –event=<code> -c <sampling_rate>
            <program monitored>
```

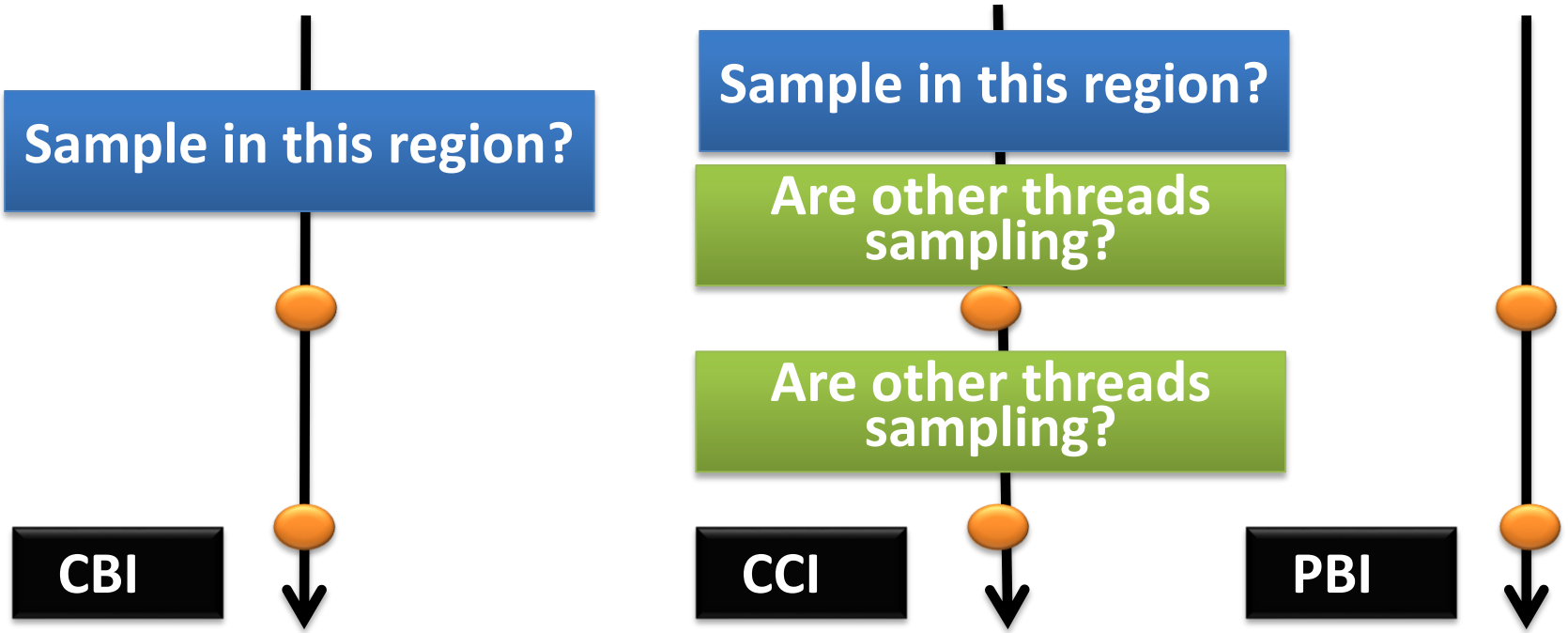
Log Id	APP	Core	Performance Event	Instruction	Function
1	Httpd	2	0x140 (Invalid)	401c3b	decrement _refcnt

Beyond concurrency bugs

- Which event?
  - Branch taken/non-taken event
- How to evaluate & sample?
  - Performance counter overflow interrupt

PBI vs. CBI/CCI (Qualitative)

- Performance



- Diagnostic capability
  - Discontinuous monitoring (CCI/CBI)
  - Continuous monitoring (PBI)
  - PBI differentiates interleaving reads from writes

Evaluation methodology

Program
Apache-1
Apache-2
Cherokee
FFT
LU
Mozilla-JS-1
Mozilla-JS-2
Mozilla-JS-3
MySQL-1
MySQL-2
PBZIP2

1/100 sampling rate, ~1000 runs in total (failure:success~1:1)

Diagnosis capability (w/ sampling)

Program	CCI-Prev
Apache-1	✓ top1
Apache-2	✓ top1
Cherokee	✗
FFT	✓ top1
LU	✓ top1
Mozilla-JS-1	✗
Mozilla-JS-2	✓ top1
Mozilla-JS-3	✓ top2
MySQL-1	-
MySQL-2	-
PBZIP2	✓ top1

Diagnosis capability (w/ sampling)

Program	CCI-Prev	PBI
Apache-1	✓ top1	✓ top1
Apache-2	✓ top1	✓ top1
Cherokee	✗	✓ top1
FFT	✓ top1	✓ top1
LU	✓ top1	✓ top1
Mozilla-JS-1	✗	✓ top1
Mozilla-JS-2	✓ top1	✓ top1
Mozilla-JS-3	✓ top2	✓ top1
MySQL-1	-	✓ top1
MySQL-2	-	✓ top1
PBZIP2	✓ top1	✓ top1

Diagnosis capability (w/ sampling)

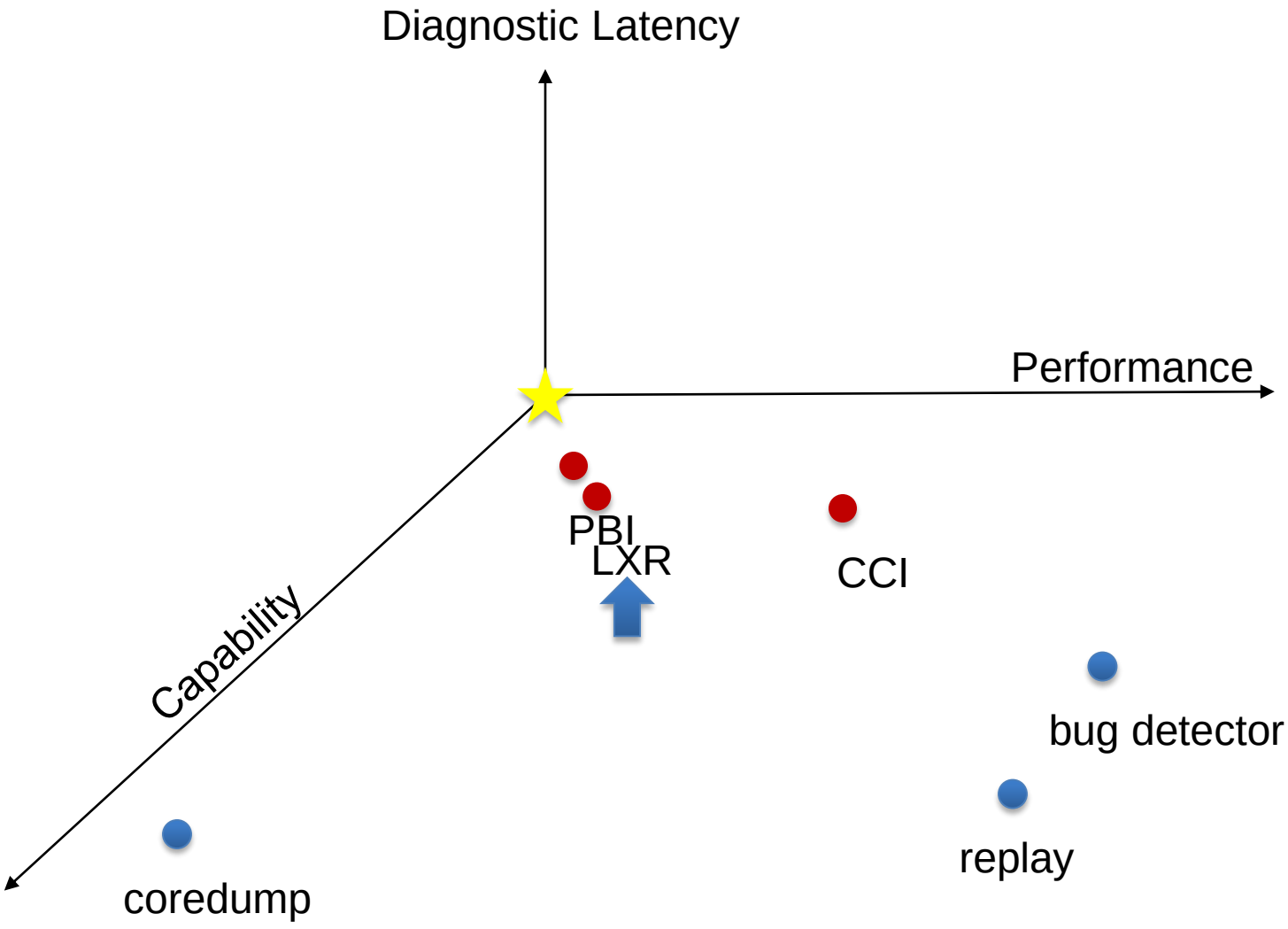
Program	CCI-Prev	PBI
Apache-1	✓ top1	✓ top1-I
Apache-2	✓ top1	✓ top1-I
Cherokee	✗	✓ top1-I
FFT	✓ top1	✓ top1-E
LU	✓ top1	✓ top1-E
Mozilla-JS-1	✗	✓ top1-I
Mozilla-JS-2	✓ top1	✓ top1-I
Mozilla-JS-3	✓ top2	✓ top1-I
MySQL-1	-	✓ top1-S
MySQL-2	-	✓ top1-S
PBZIP2	✓ top1	✓ top1-I

Diagnosis performance (overhead)

Program	CCI-Prev	PBI
Apache-1	1.90%	0.40%
Apache-2	0.40%	0.40%
Cherokee	0.00%	0.50%
FFT	121%	1.00%
LU	285%	0.80%
Mozilla-JS-1	800%	1.50%
Mozilla-JS-2	432%	1.20%
Mozilla-JS-3	969%	0.60%
MySQL-1	-	3.80%
MySQL-2	-	1.20%
PBZIP2	1.40%	8.40%

Sequential-bug failure diagnosis results are also good!

Are we done?



1/100 sampling rate ➔ ~100 failures required for diagnosis

How to do better than PBI?

What to collect	How to collect	How to use the collected
	Sampling	



Performance	Capability	Latency

How to collect sufficient root-cause information in 1 run w/ small overhead?

How to do better than PBI?

What to collect	How to collect	How to use the collected
	Biased sampling	



Performance	Capability	Latency

Collect information @ likely root-cause locations

LXR – Last eXecution Record

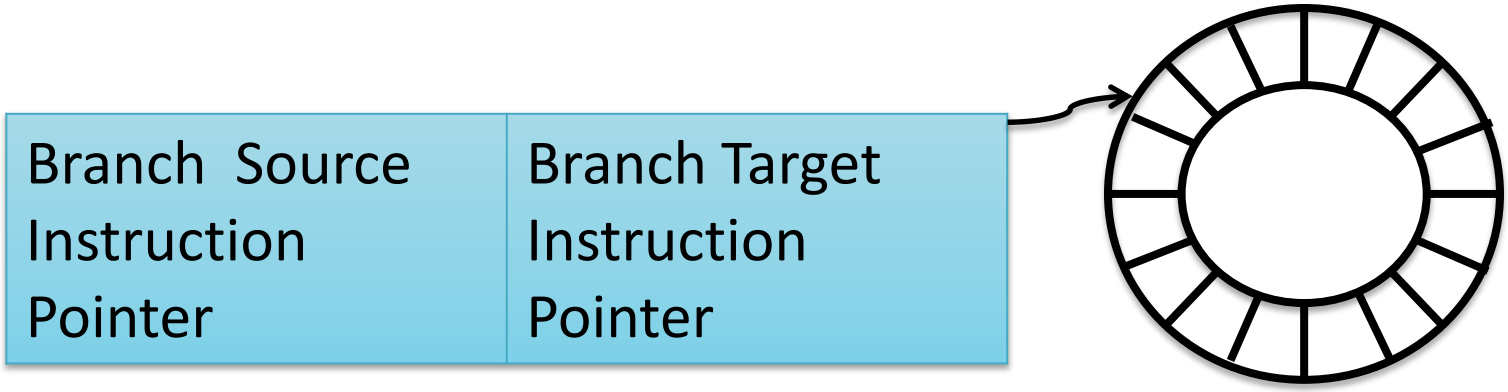
- What to collect?
  - Last few branches right before failure
  - Last few cache-coherence events right before failures
- How to collect/maintain LXR?
  - Existing* hardware support!



Performance	Capability	Code Size	Change Hardware?	Diagnosis Latency
Good (<5% overhead)	Good	Little Change	Simple Extension*	Short

Last Branch Record (LBR)

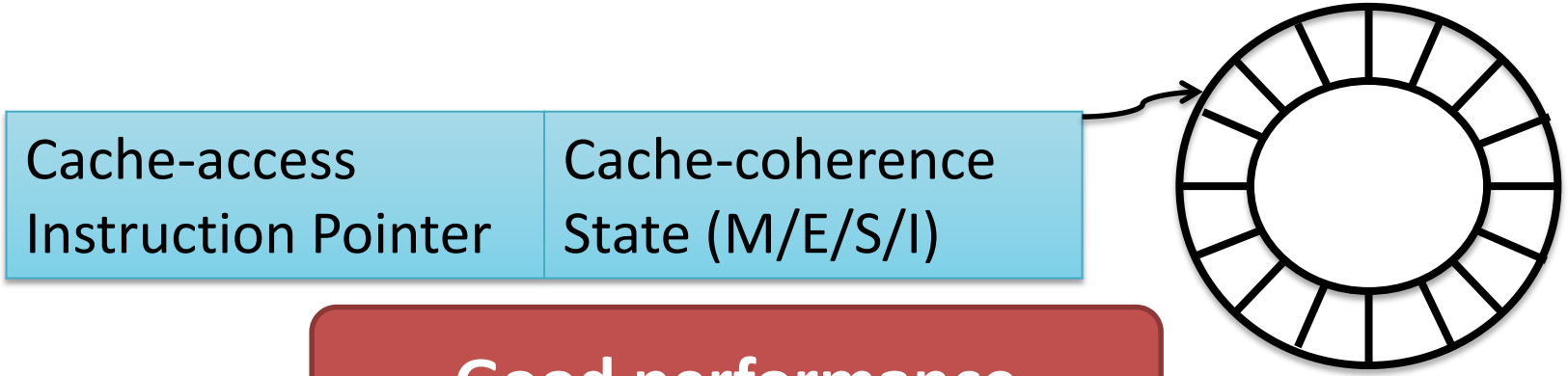
- **Existing** hardware feature
  - Store recently taken branches
  - Circular buffer with 16 entries (Intel Nehalem)
  - **Negligible** overhead



Good performance

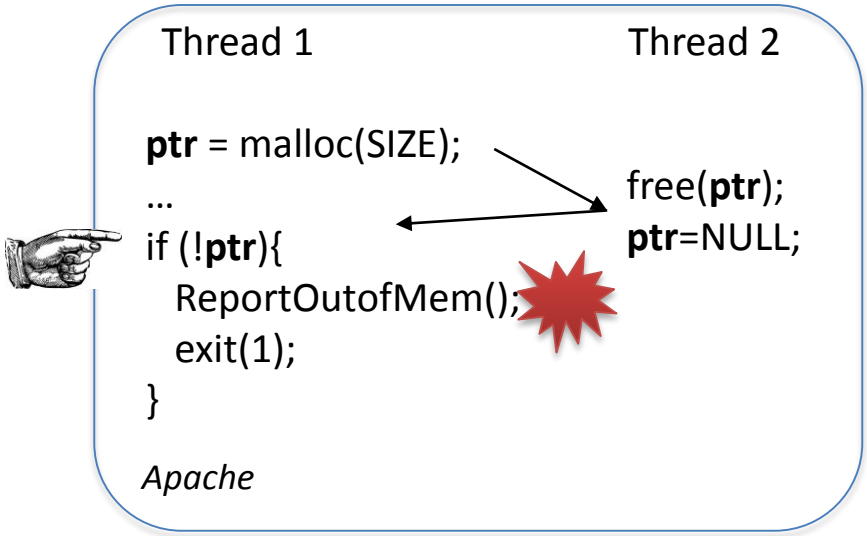
Last Cache-coherence Record (LCR)

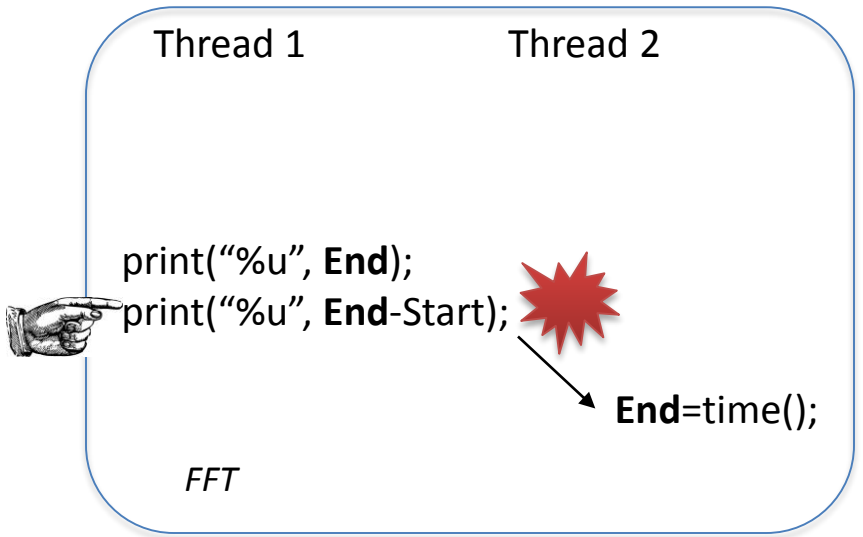
- **Existing** hardware feature
  - Configurable cache-coherence event counting
- Extension
  - Buffer to collect this information
  - Set of recent L1 data cache access instructions
- Negligible overhead (estimated)



Good performance

Is LXR useful?





Bugs have short error-propagation distance → LXR is sufficient for failure diagnosis

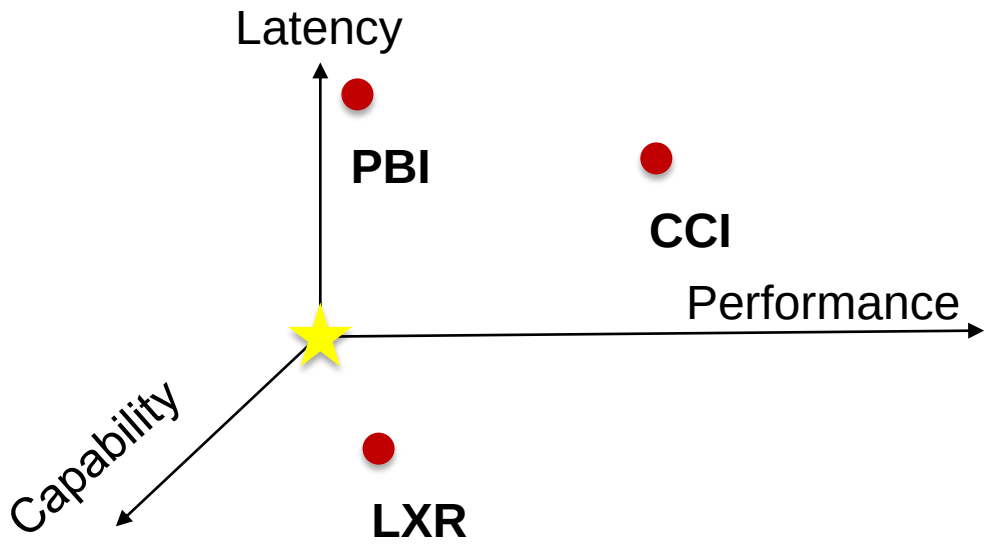
Good diagnosis capability

ConSeq: Detecting Concurrency Bugs through Sequential Errors, ASPLOS'11

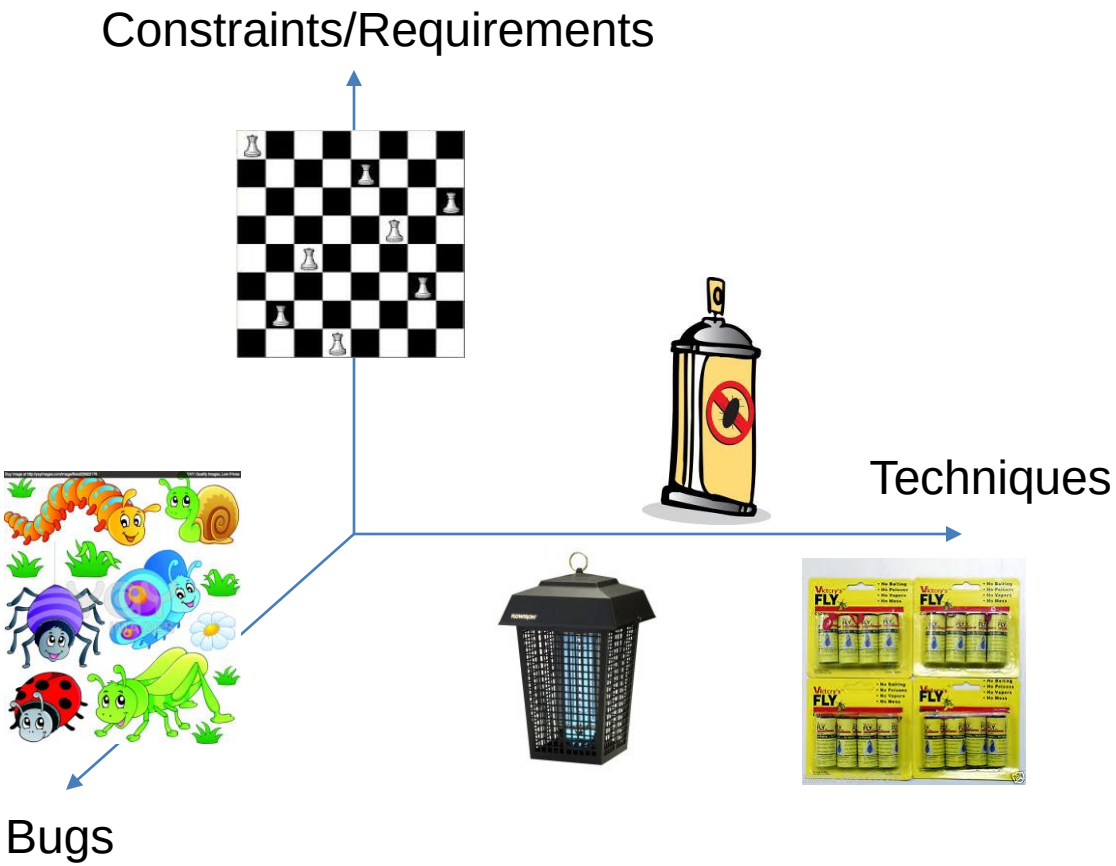
LXR vs PBI vs CBI/CCI

	Performance	Capability	Diagnosis Latency (#-failure-runs)
LXR	<5%	23/31	1~10 failures
PBI	<5%	25/31	1000 failures
CBI/CCI	3% ~ 969%	18/31	1000 failures

Outline



Conclusions & Future Work



Thanks!
Questions?

My collaborators

- Prof. Tom Reps
- Prof. Ben Liblit
- Prof. Michael Swift
- Prof. Karthikeyan Sankaralingam
- Prof. Darko Marinov

My students

- Wei Zhang (IBM Research)
- Guoliang Jin (N. Carolina State Univ.)
- Linhai Song
- Joy Arulraj
- Po-chun Chang