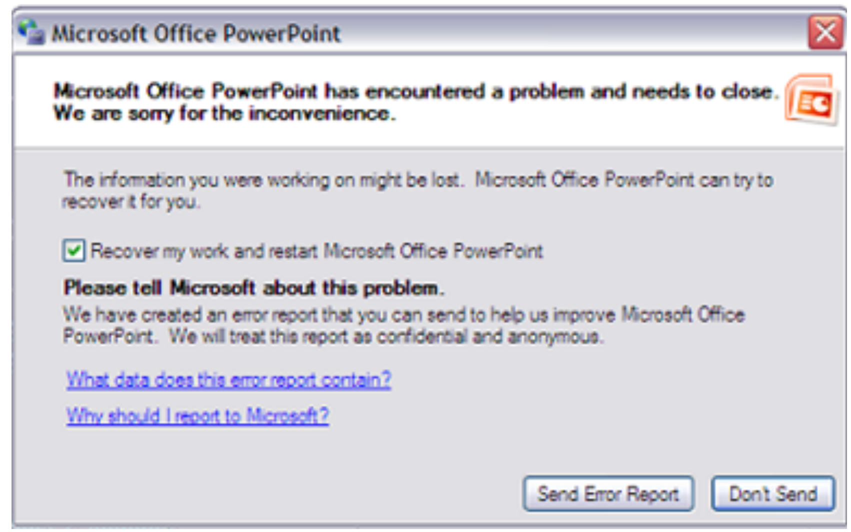
Leveraging the Short-Term Memory of Hardware to Diagnose Production-Run Software Failures

Joy Arulraj, Guoliang Jin and Shan Lu

THE UNIVERSITY of WISCONSIN MADISON

Production-Run Failure Diagnosis



- Goal
  - Figure out root cause of failure on client machines
  - Fix them quickly

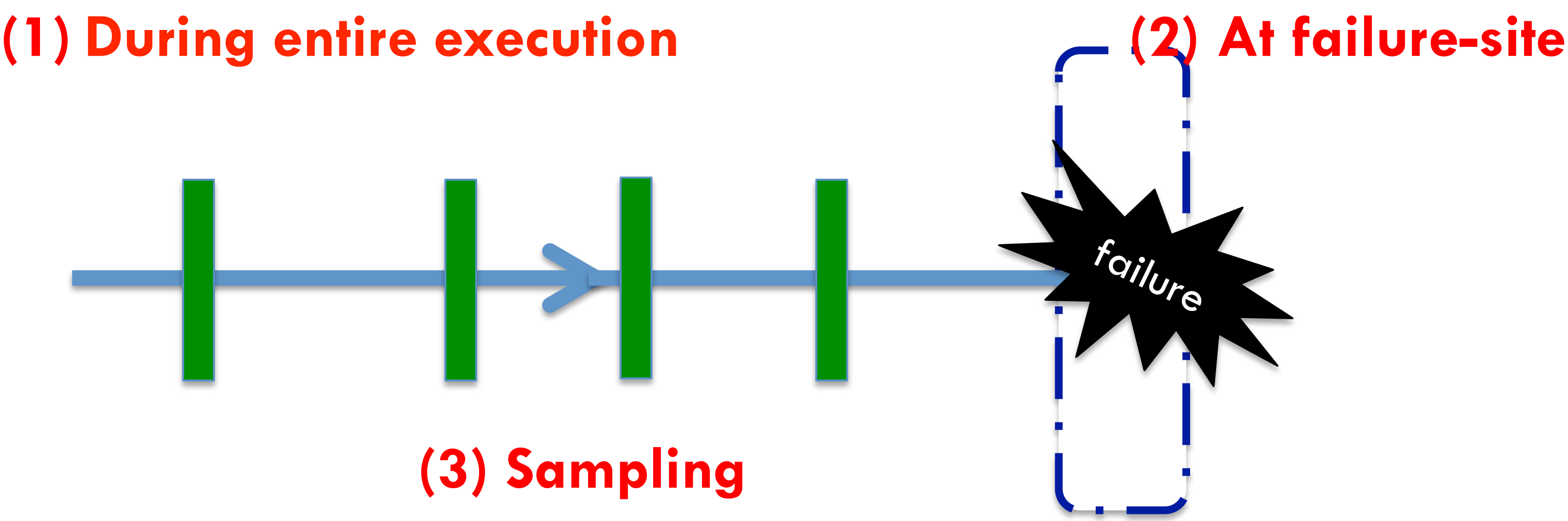
Importance

- Social and financial impact
  - Toyota Prius software glitch
  - NASDAQ Facebook IPO glitch

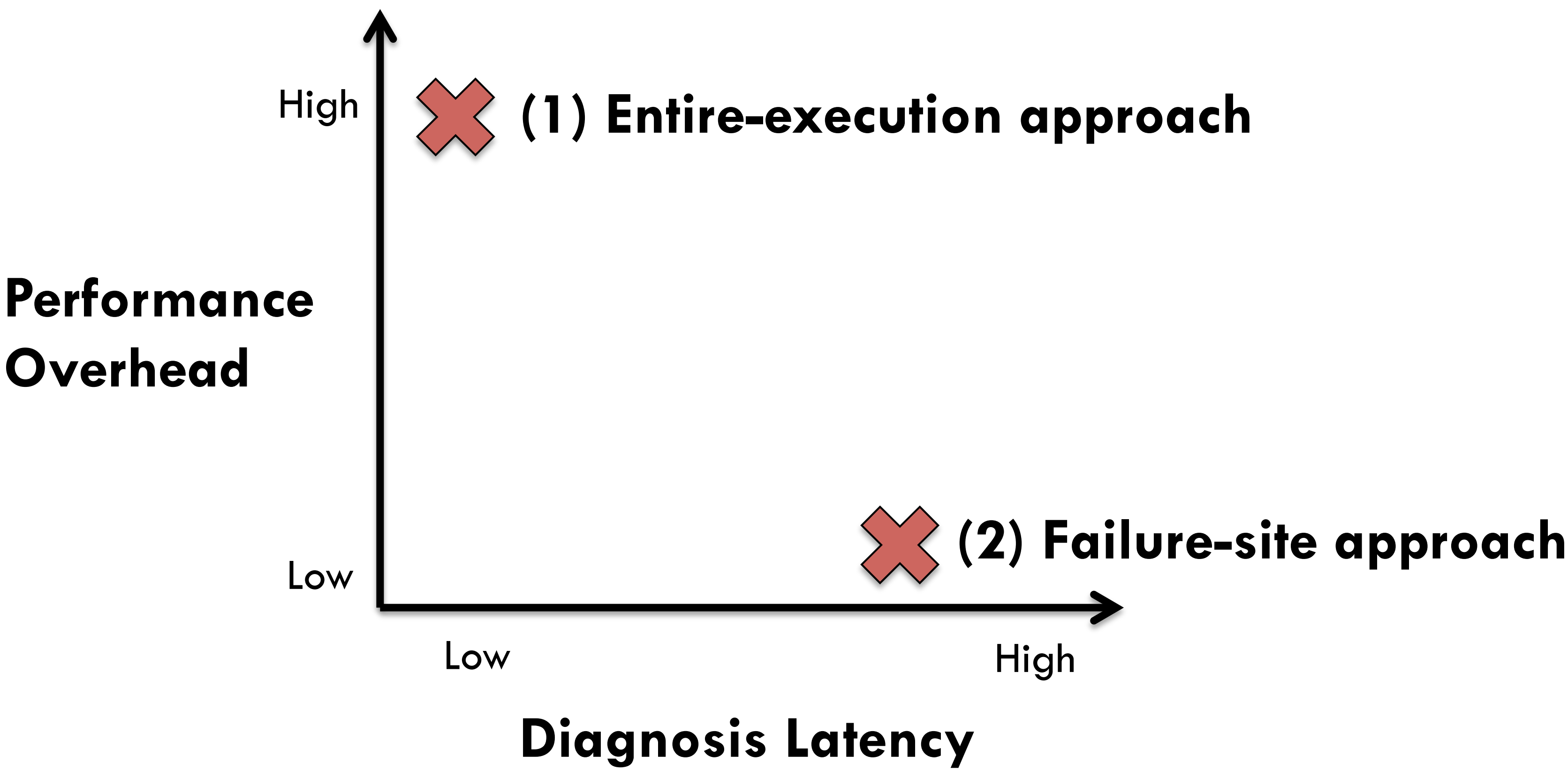
Challenges

- Limited program execution information
  - Performance and privacy reasons
- Complicated root cause
  - Sequential bugs
  - Concurrency bugs
- Need to diagnose and fix quickly

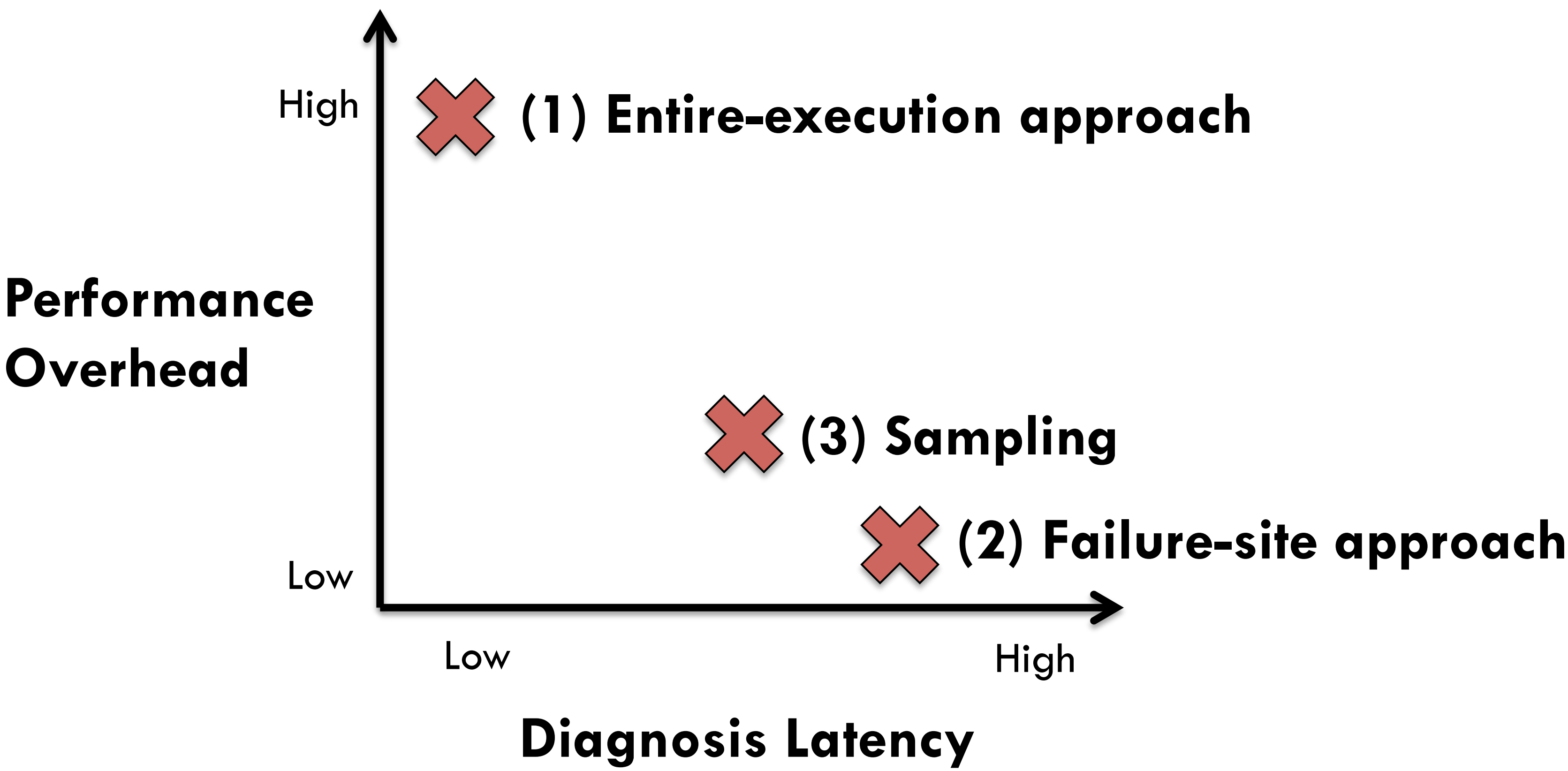
Existing Tools



Limitations Of Existing Tools



Limitations Of Existing Tools



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

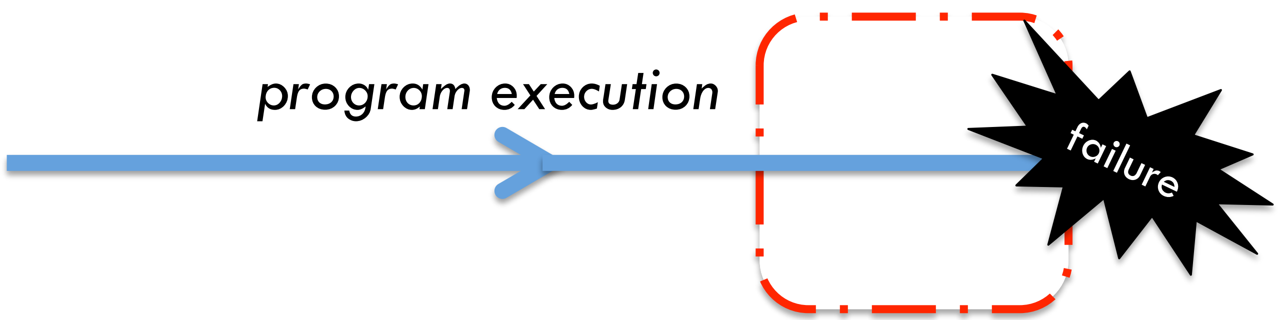
Challenge

- Low performance overhead
  - – Collect *little* execution information
- Low diagnosis latency
  - – Collect *root-cause* related information

Which part of the program execution is most likely to contain root-cause information?

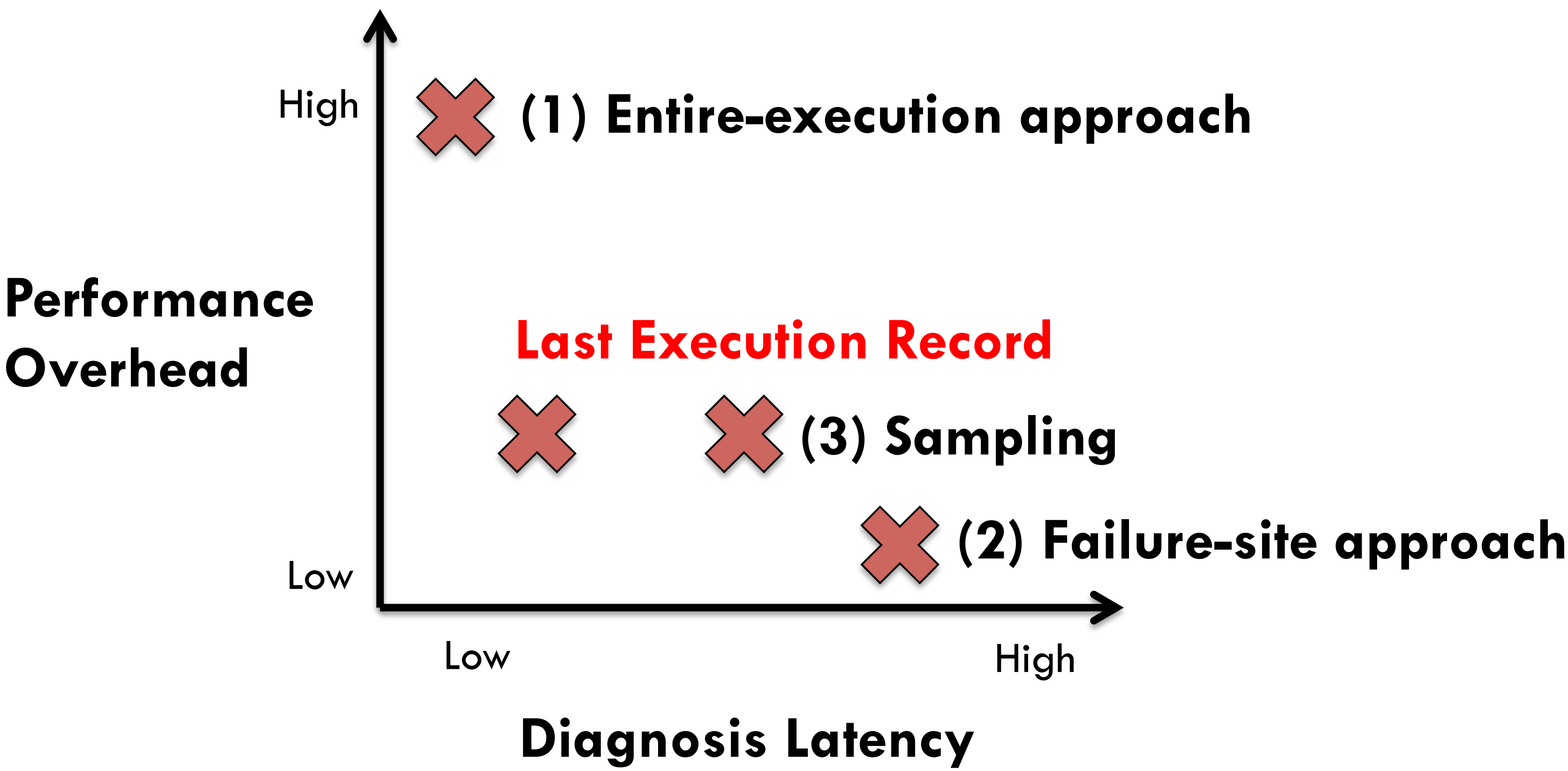
Our Solution: Last Execution Record

- Execution right before failure
  - Last Execution Record (LXR)
- **How** to collect this information efficiently?
  - Leverage simple hardware support



Last Execution Record (LXR)

Last Execution Record (LXR)



Outline

- LXR Design
- Failure diagnosis using LXR
- Evaluation

LXR Design Questions & Principles

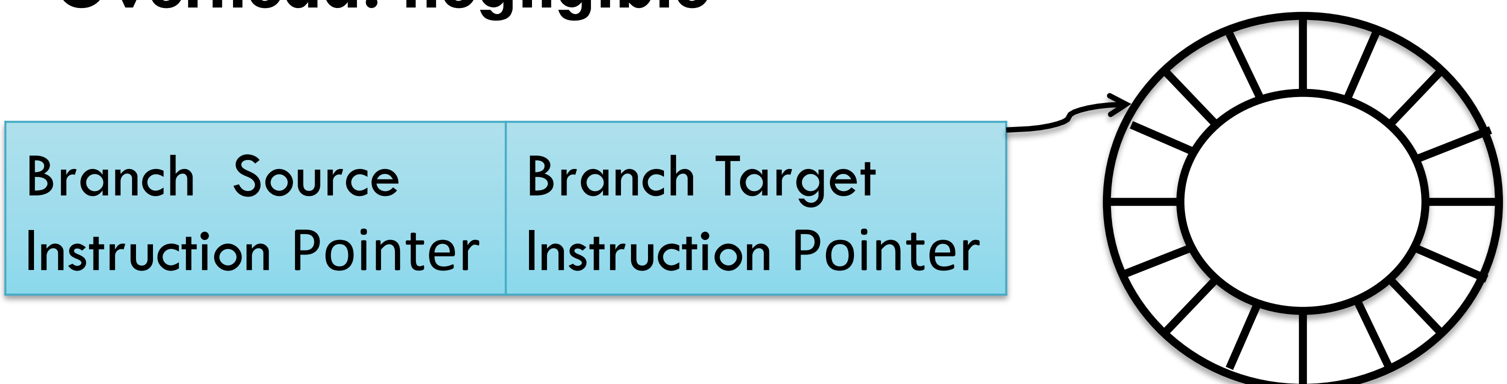
- **What** should we collect in LXR?
  - **Useful** for failure diagnosis
- **How** to collect LXR?
  - **Lightweight** to collect

LXR Design For Sequential Bugs

- **What** should we collect in LXR?
  – Recently taken branches
- **How** to collect LXR?
  – Use existing hardware support -- LBR

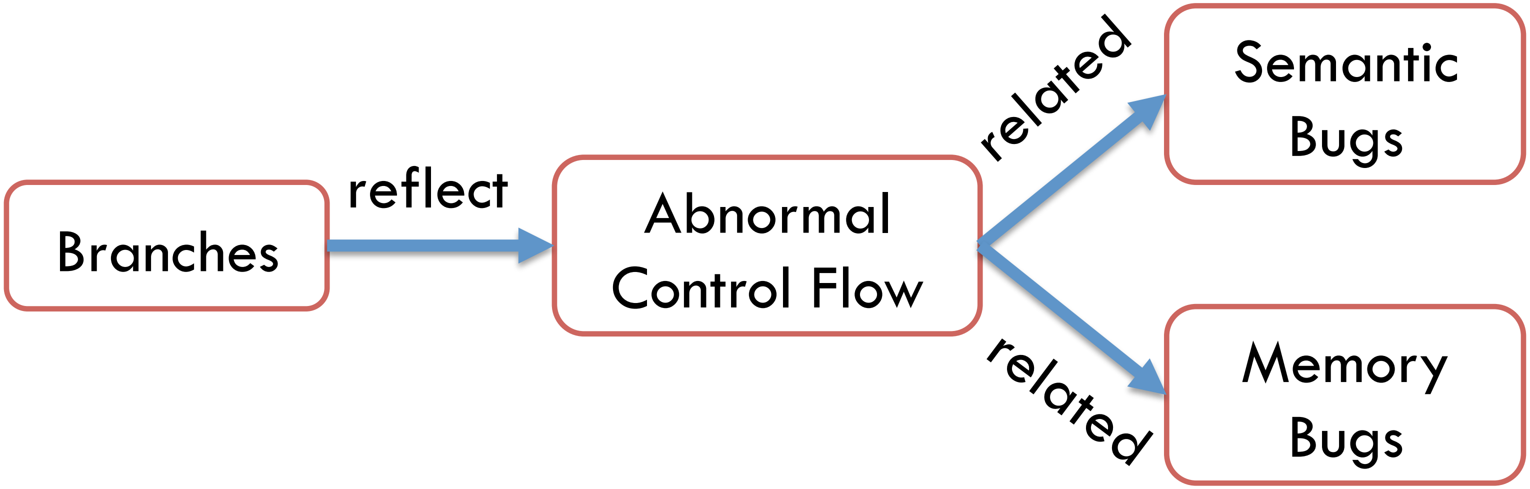
Last Branch Record (LBR)

- **Existing hardware feature**
  - Set of recently taken branches
  - Circular buffer with 16 entries (Intel Nehalem)
  - **Overhead: negligible**



Lightweight

Is LBR useful ?



- Root-cause of many types of sequential bugs [PLDI.2005]
- Error-propagation distances tend to be short [DSN.2003]

Useful

Sequential Bug Example

- Coreutils: sort -m -o file1 file1

```
// SORT.C

void merge (...) {
  ...
  open_input_files(...);
}
```

```
int open_input_files (...) {
  if (files[i].pid != 0) /* child process */
    table→bucket = val;
  else ...
}
```

failure

CALL STACK
open_input_files()
merge()
...
main()

Is LBR useful ?

- Coreutils: sort -m -o file1 file1

```
// SORT.C

void merge (...) {
  avoid_trashing_input(…);
  …
  open_input_files(...);
}
```

```
int avoid_trashing_input (...) {
  if(…) {
    int num_merged = 0;
    while (i + num_merged < nfiles) {
      num_merged += mergefiles(...);
      memmove(&files[i], &files[i+num_merged],);
    }
  }
}
```

```
int open_input_files (...) {
  if (files[i].pid != 0)
    table→bucket = val;
  else …
}
```

failure

LBR
if (files[i].pid != 0)
...
while (i+num_merged < nfiles)

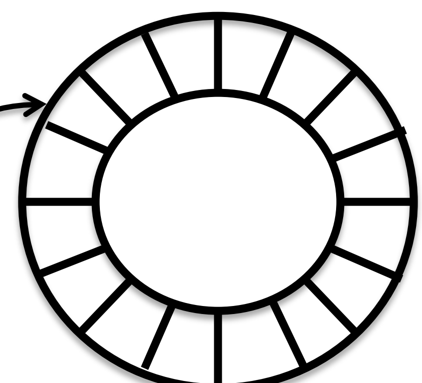
LXR Design For Concurrency Bugs

- **What** should we collect in LXR?
  - Recently executed cache-access instructions
  - Cache-coherence state observed (M/E/S/I)
- **How** to maintain and collect LXR?
  - Key hardware feature already exists
  - Propose a simple hardware extension to use that

Last Cache-coherence Record (LCR)

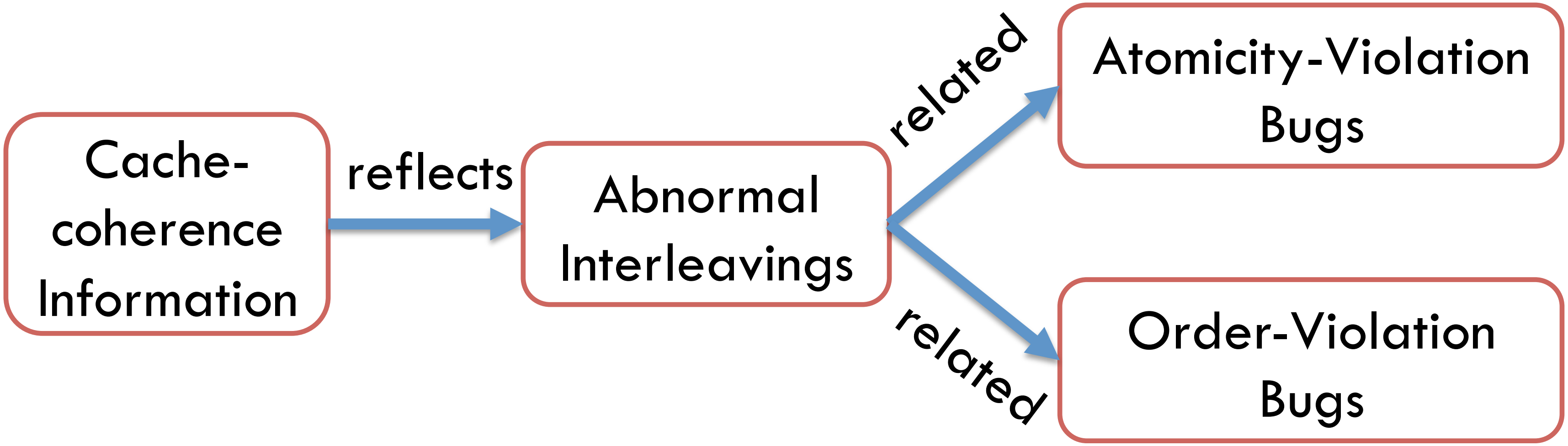
- **Existing hardware feature**
  - Configurable cache-coherence event counting
- **Extension:**
  - Buffer to collect this information
  - Set of recent L1 data cache access instructions
- **Overhead: not perceivable**

Cache-access Instruction Pointer	Cache-coherence State (M/E/S/I)



Lightweight

Is LCR Useful ?



- Related to concurrency bug root-causes [ASPLOS.2013]
- Error-propagation distances tend to be short [ASPLOS.2011]

Useful

Is LCR useful ?

- Mozilla JavaScript Engine

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


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

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

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

Is LCR useful ?

- Mozilla JavaScript Engine: Success Run ✔

Thread 1 **Modified**

W

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


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

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

R

Modified

Thread 2 **Modified**

```
FreeState(...){
        Destroy(table);

        table = NULL;
}
```

R

W

Is LCR useful ?

- Mozilla JavaScript Engine: Failure Run ✖

Thread 1 | **Invalid**

W

R

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

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

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

failure

Thread 2 | **Modified**

```
FreeState(...){
    Destroy(table);

    table = NULL;
}
```

R

W

Invalid

LCR	
table == NULL	Invalid
table = New()	Invalid
…	…

Outline

- LXR Design
- Failure diagnosis using LXR
- Evaluation

Manual failure diagnosis

- Enhance logging by collecting LXR
  - Existing failure logging functions
  - Signal handler

```
// failure logging function
error_wrapper(args){
      DISABLE_LXR();
      PROFILE_LXR();
      error (args);
}
```

```
// signal handler
void handler(int signo)
{
    DISABLE_LXR();
    PROFILE_LXR();
    …
}
```

Automated failure diagnosis

- Collect LXR in both failure and success runs
- Statistical analysis
  - Automatically identify failure predictors

LCR	
table == NULL	Invalid
table = New()	Invalid
…	…

✖

LCR	
table == NULL	Modified
table = New()	Invalid
…	…

✔

LCR AUTOMATED	Score
table == NULL	0.91
table = New()	0.56
…	…

Implementation details

- LBR exposed via Linux kernel module
  - Enable, configure and access using our interface
  - Reducing LBR pollution from irrelevant branches
  - Details in paper
- LCR simulated using PIN infrastructure
  - L1 data cache with MESI coherence protocol
  - Interface similar to LBR
  - Details in paper

Outline

- LXR Design
- Failure diagnosis using LXR
- Evaluation

Methodology

- 31 real-world failures
  - In open-source server, client, utility programs
  - 20 sequential and 11 concurrency bugs
- Compared against CBI/CCI
  - State-of-the-art **software-based** tools
  - Diagnose sequential and concurrency bugs
  - Perform **sampling to lower overhead**

Does LBR help locate root cause ?

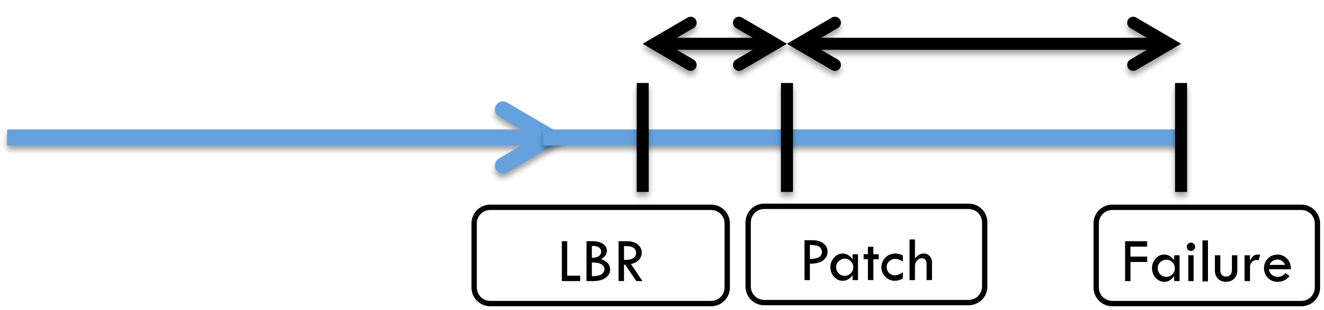
APPLICATION	LBR MANUAL (N^{th} entry)	LBR AUTOMATED (N^{th} entry)	CBI (N^{th} entry)
Apache	3	1	2
Squid	2	1	-
Coreutils	12	1	2
Tar	4	1	1
PBZIP	4	1	-

- Root-cause branch mostly in recent 8 LBR entries
- So, even short-term LBR memory sufficient !

Cross-checking LBR with patches

APPLICATION	FAILURE SITE TO PATCH (LoC)	LBR TO PATCH (LoC)
Apache	Another file	3
Squid	123	2
Coreutils	309	0
Tar	Another file	2
PBZIP	41	1

- LBR entries are much closer to patch for most bugs

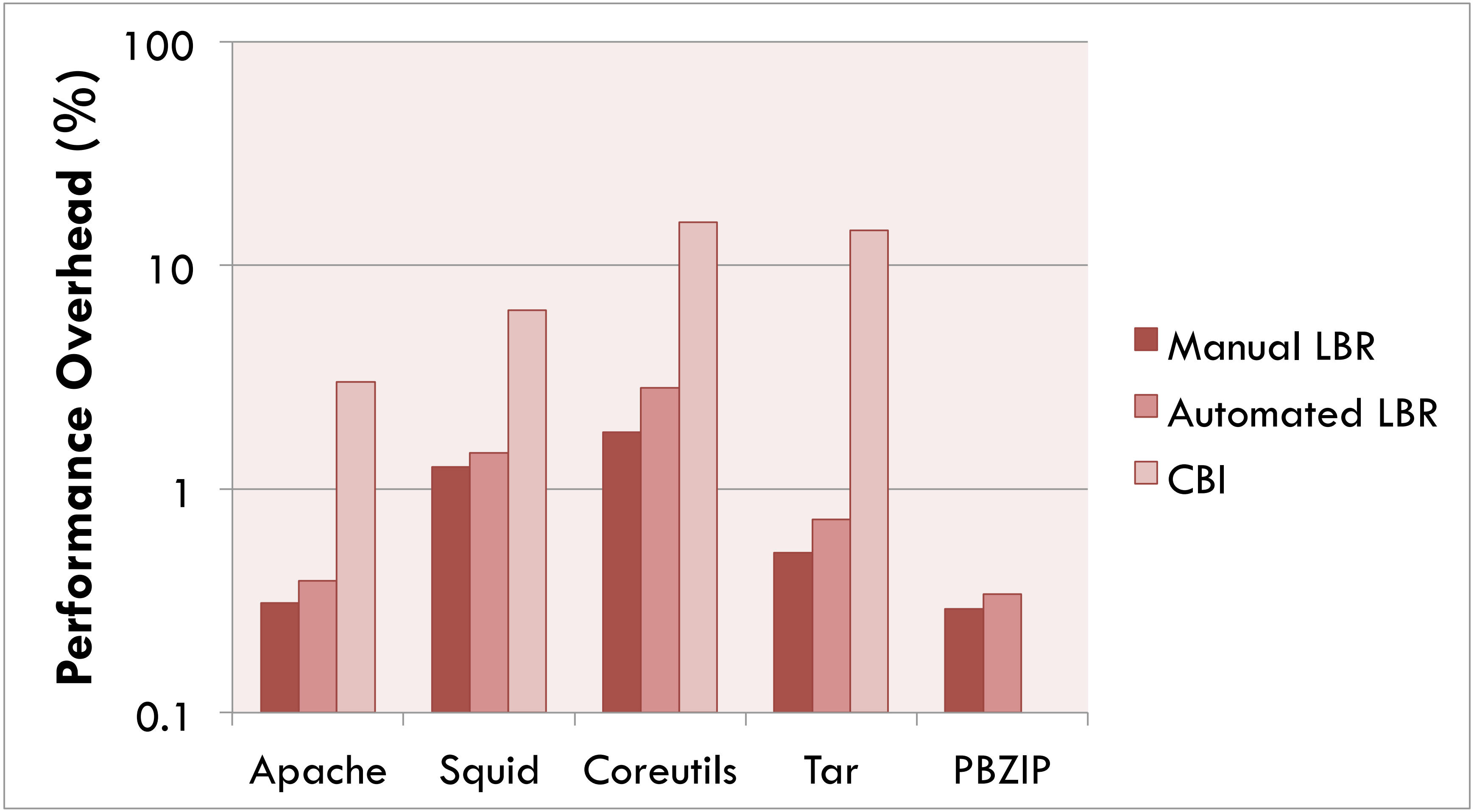


Diagnosis Latency

TOOL	DIAGNOSIS LATENCY	SAMPLING
Manual LBR	1 failure run	No
Automated LBR	10 failure runs	No
CBI	1000 failure runs	Yes

- LBR tools need fewer failure runs for diagnosis
- CBI uses sampling which increases latency

Performance Overhead

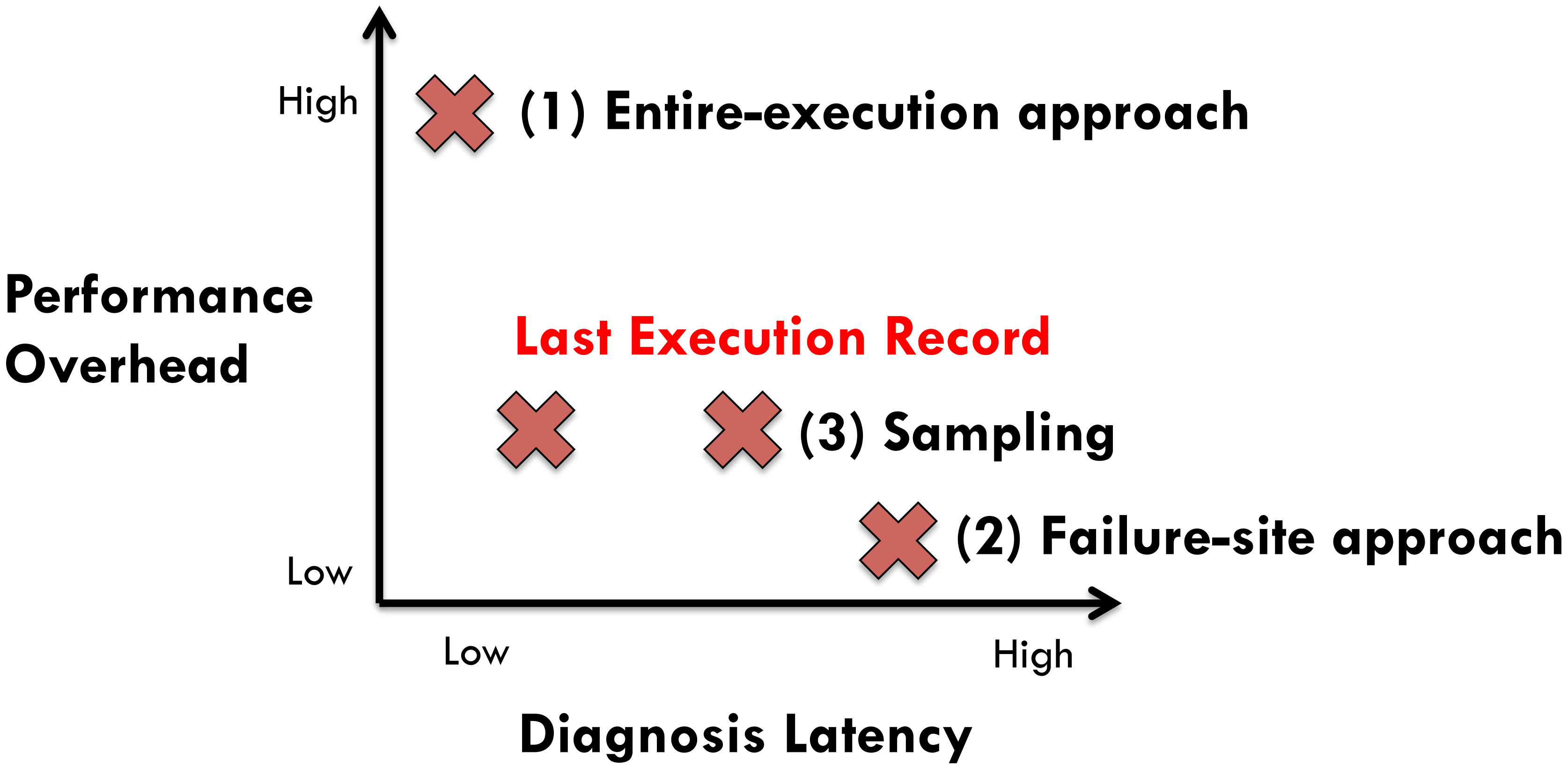


Does LCR help locate root cause ?

APPLICATION	LCR MANUAL (N^{th} entry)	LCR AUTOMATED (N^{th} entry)	CCI (N^{th} entry)
Apache	5	1	1
Cherokee	-	-	-
Mozilla	8	1	1
MySQL	9	1	-
PBZIP	7	1	1

- Locates root-cause in 7 out of 11 failures
- So, short-term LCR memory sufficient

Summary


High
(1) Entire-execution approach
Performance Overhead
Last Execution Record
(3) Sampling
(2) Failure-site approach
Low
Low
High
Diagnosis Latency