Demand-Driven Software Race Detection using Hardware Performance Counters

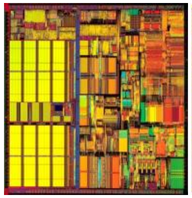
Joseph L. Greathouse[†], Zhiqiang Ma[‡], Matthew I. Frank[‡]
Ramesh Peri[‡], Todd Austin[†]

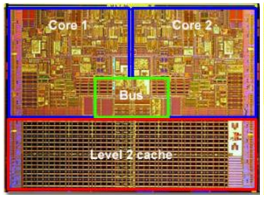
[†]University of Michigan

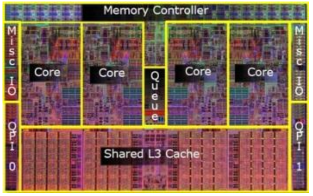
[‡]Intel Corporation

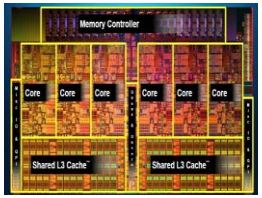
ISCA-38
June 6, 2011

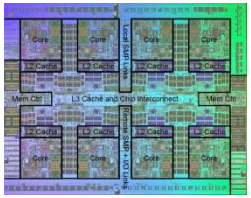
Concurrency Bugs Still Matter

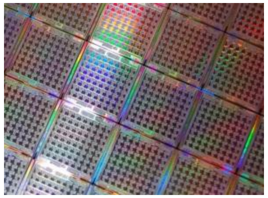


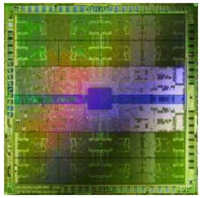




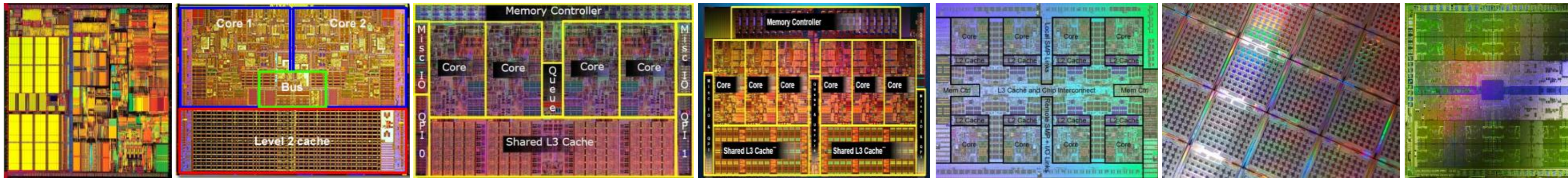








Concurrency Bugs Still Matter



In spite of proposed hardware solutions

Hardware Data Race Recording

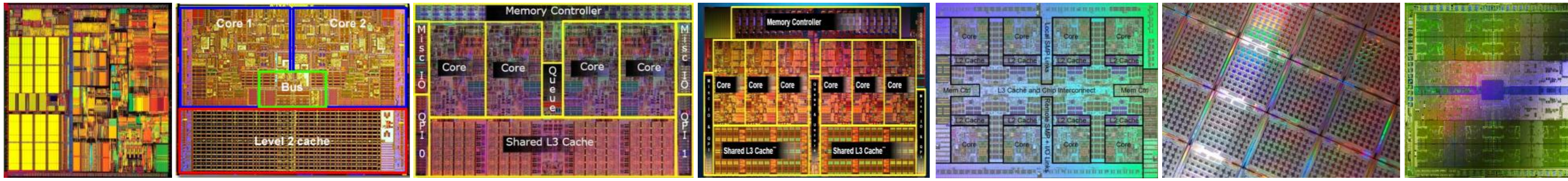
Bulk Memory Commits

Deterministic Execution/Replay

Bug-Free Memory Models

Atomicity Violation Detectors

Concurrency Bugs Still Matter



In spite of proposed hardware solutions

Hardware Data Race Recording

Bulk Memory Commits

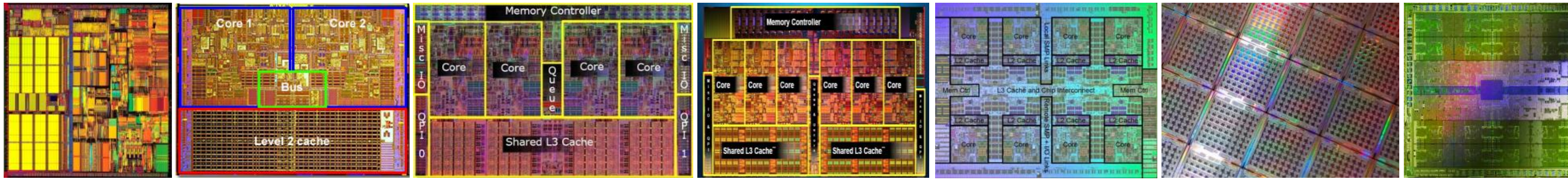
Deterministic Execution/Replay

Bu
Memory

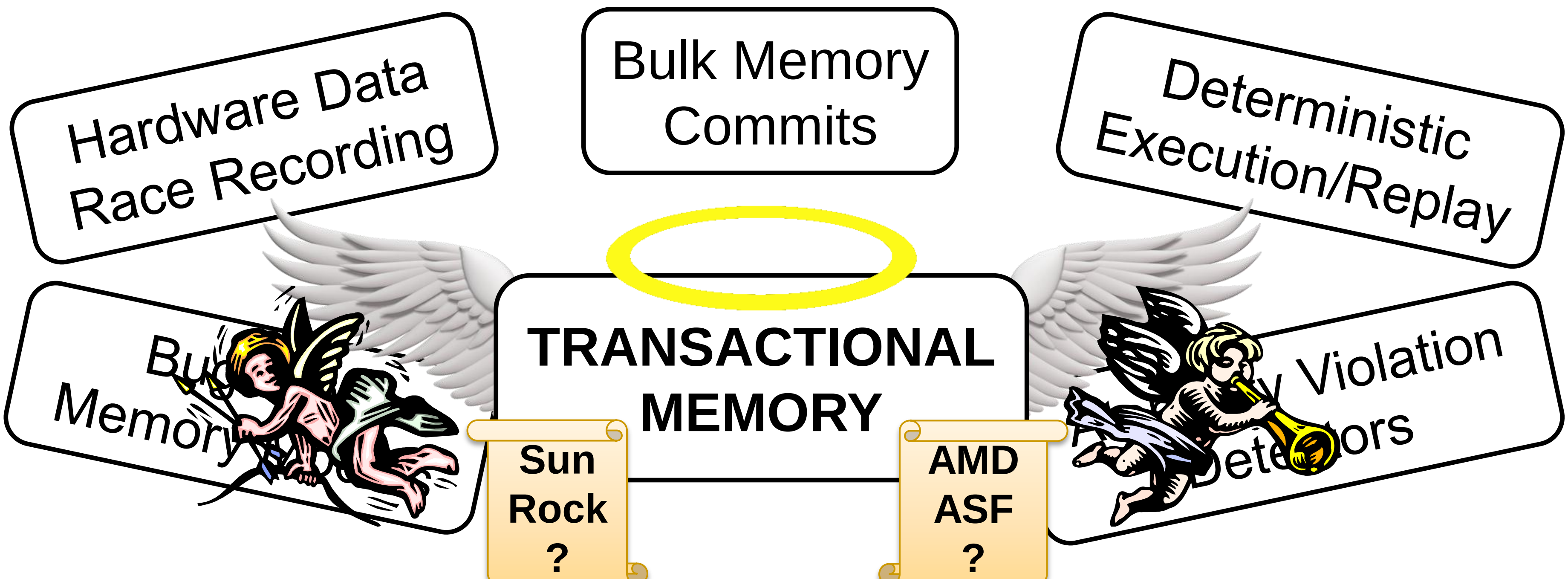
TRANSACTIONAL MEMORY

y Violation
ete ors

Concurrency Bugs Still Matter



In spite of proposed hardware solutions



Concurrency Bugs Matter NOW

Nov. 2010 OpenSSL Security Flaw

Concurrency Bugs Matter NOW

TIME

Thread 1 mylen=small	Thread 2 mylen=large
if(ptr==NULL)	
	if(ptr==NULL)
	len2=thread_local->mylen;
	ptr=malloc(len2);
len1=thread_local->mylen;	
ptr=malloc(len1);	
memcpy(ptr, data1, len1)	
	memcpy(ptr, data2, len2)

ptr

∅

Concurrency Bugs Matter NOW

TIME

Thread 1 mylen=small	Thread 2 mylen=large
if(ptr==NULL)	
	if(ptr==NULL)
	len2=thread_local->mylen;
	ptr=malloc(len2);
len1=thread_local->mylen;	
ptr=malloc(len1);	
memcpy(ptr, data1, len1)	
	memcpy(ptr, data2, len2)

ptr

∅

Concurrency Bugs Matter NOW

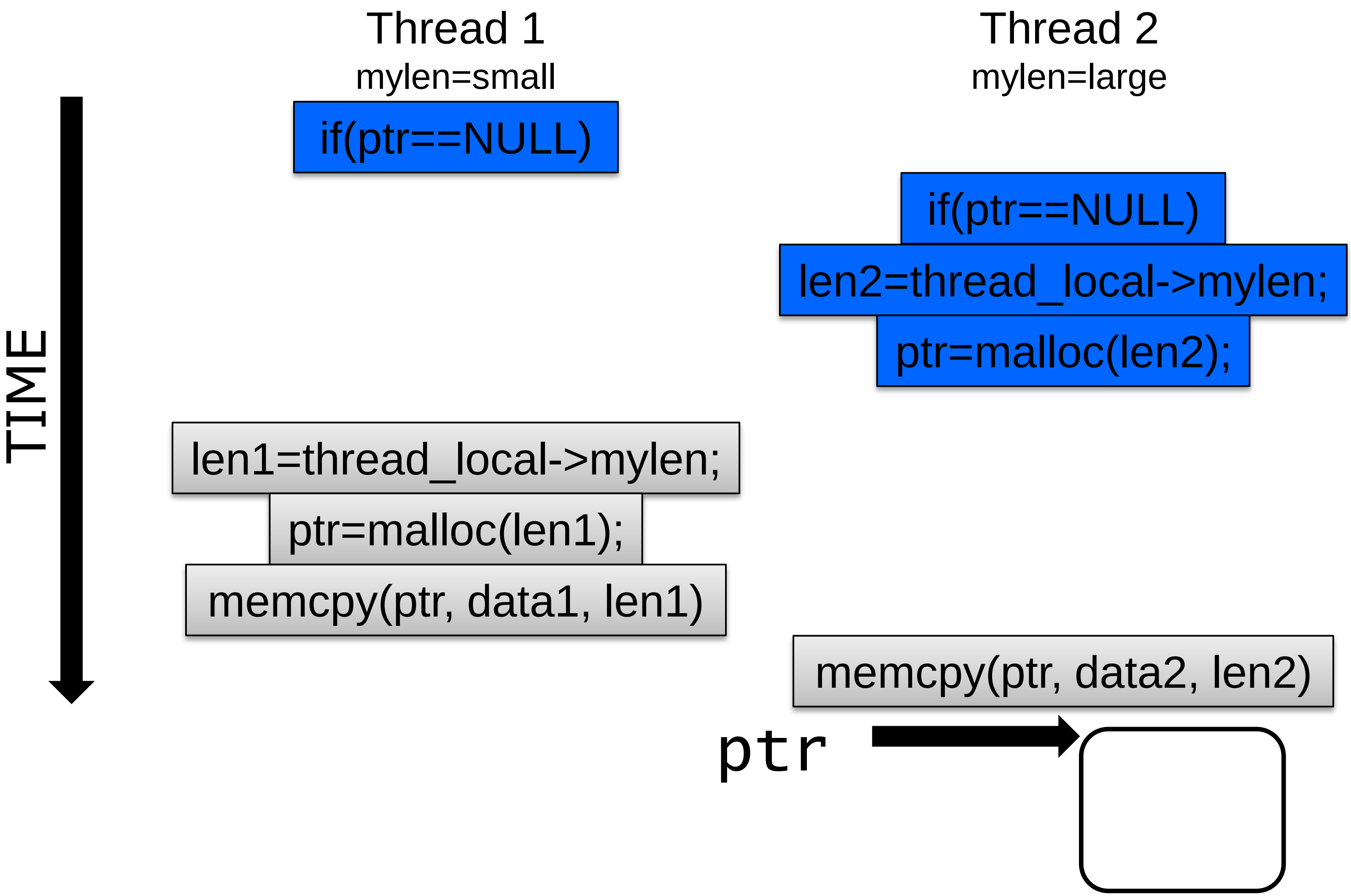
TIME

Thread 1 mylen=small	Thread 2 mylen=large
if(ptr==NULL)	
	if(ptr==NULL)
	len2=thread_local->mylen;
	ptr=malloc(len2);
len1=thread_local->mylen;	
ptr=malloc(len1);	
memcpy(ptr, data1, len1)	
	memcpy(ptr, data2, len2)

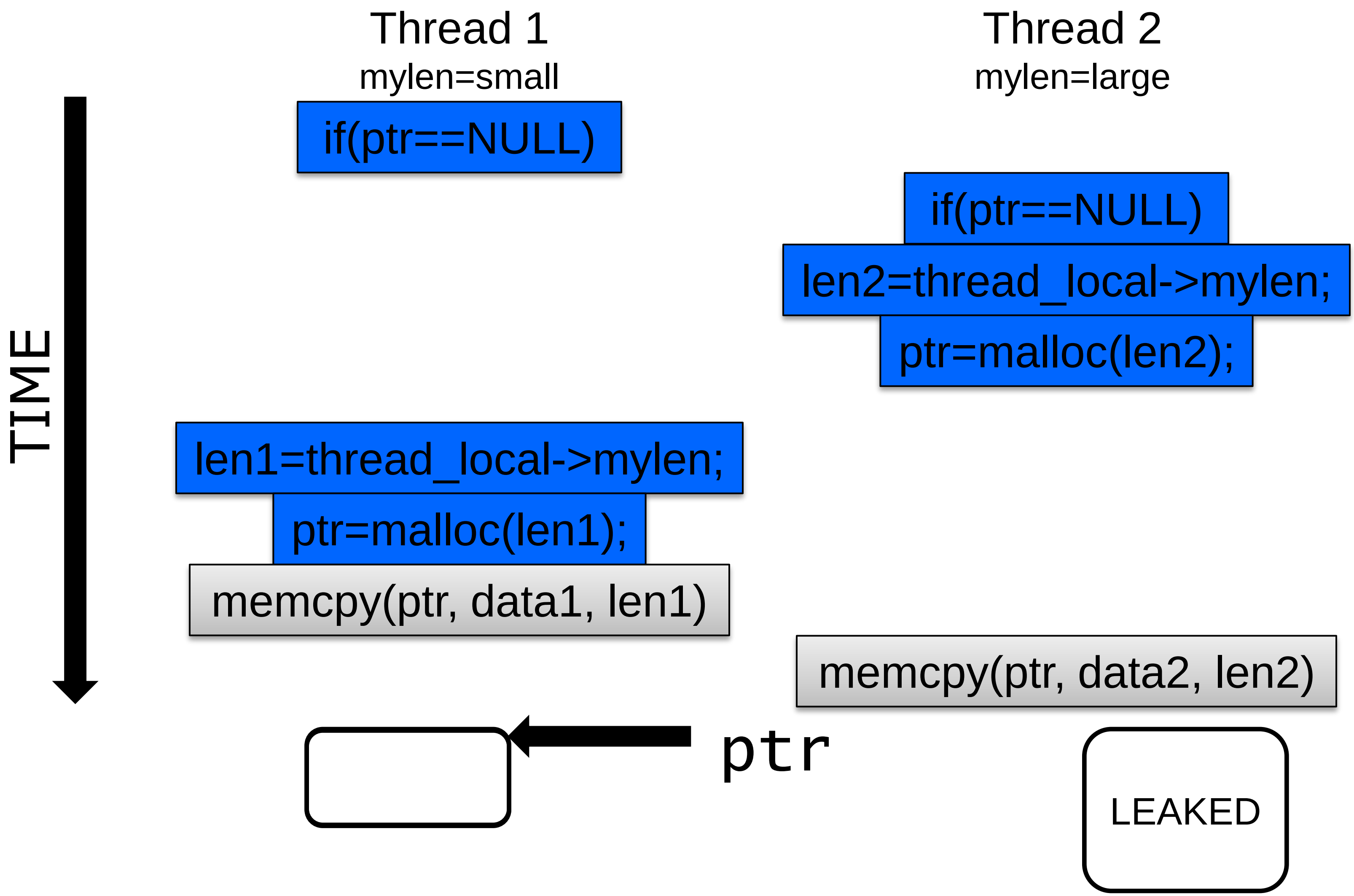
ptr

∅

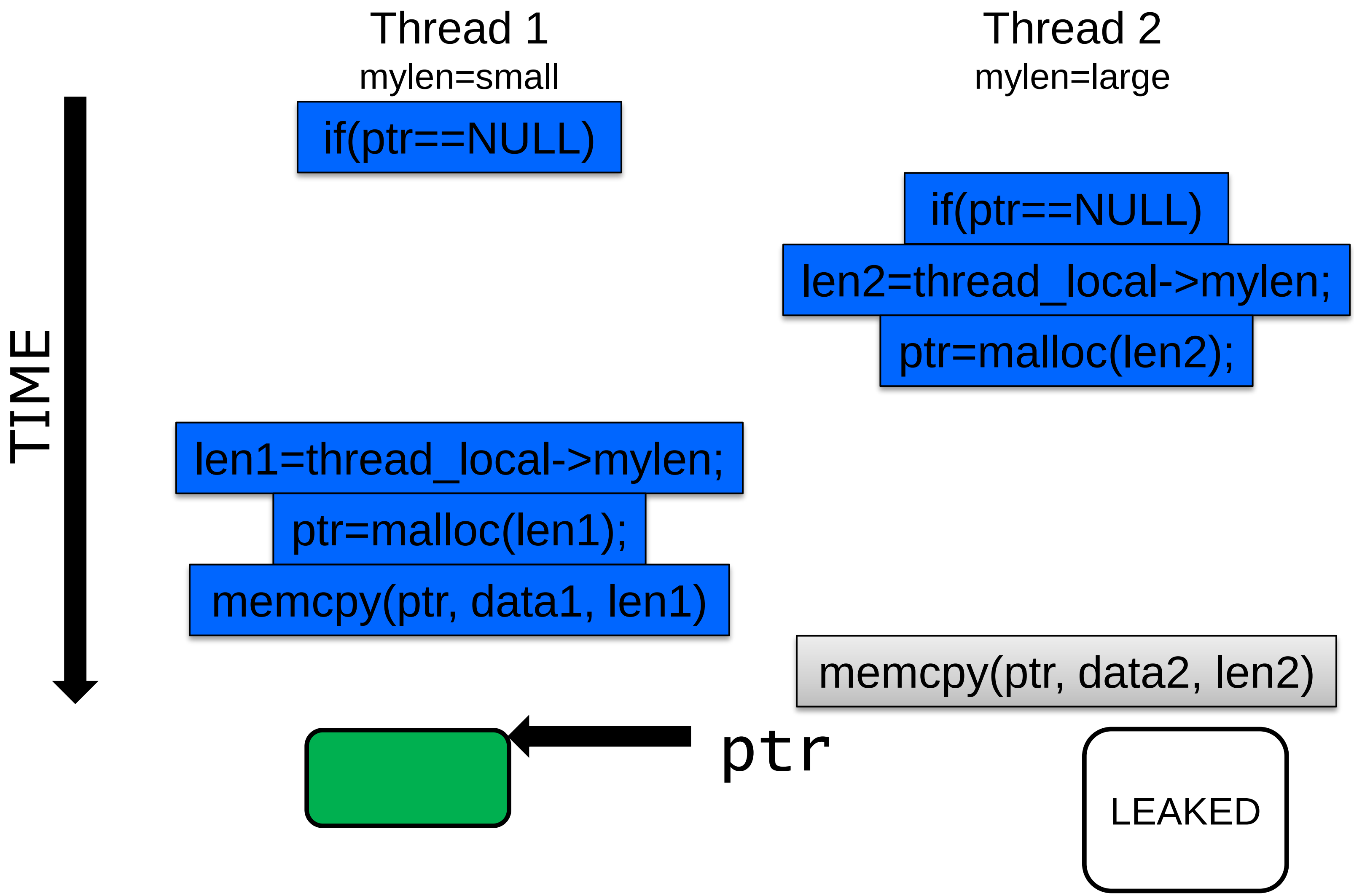
Concurrency Bugs Matter NOW



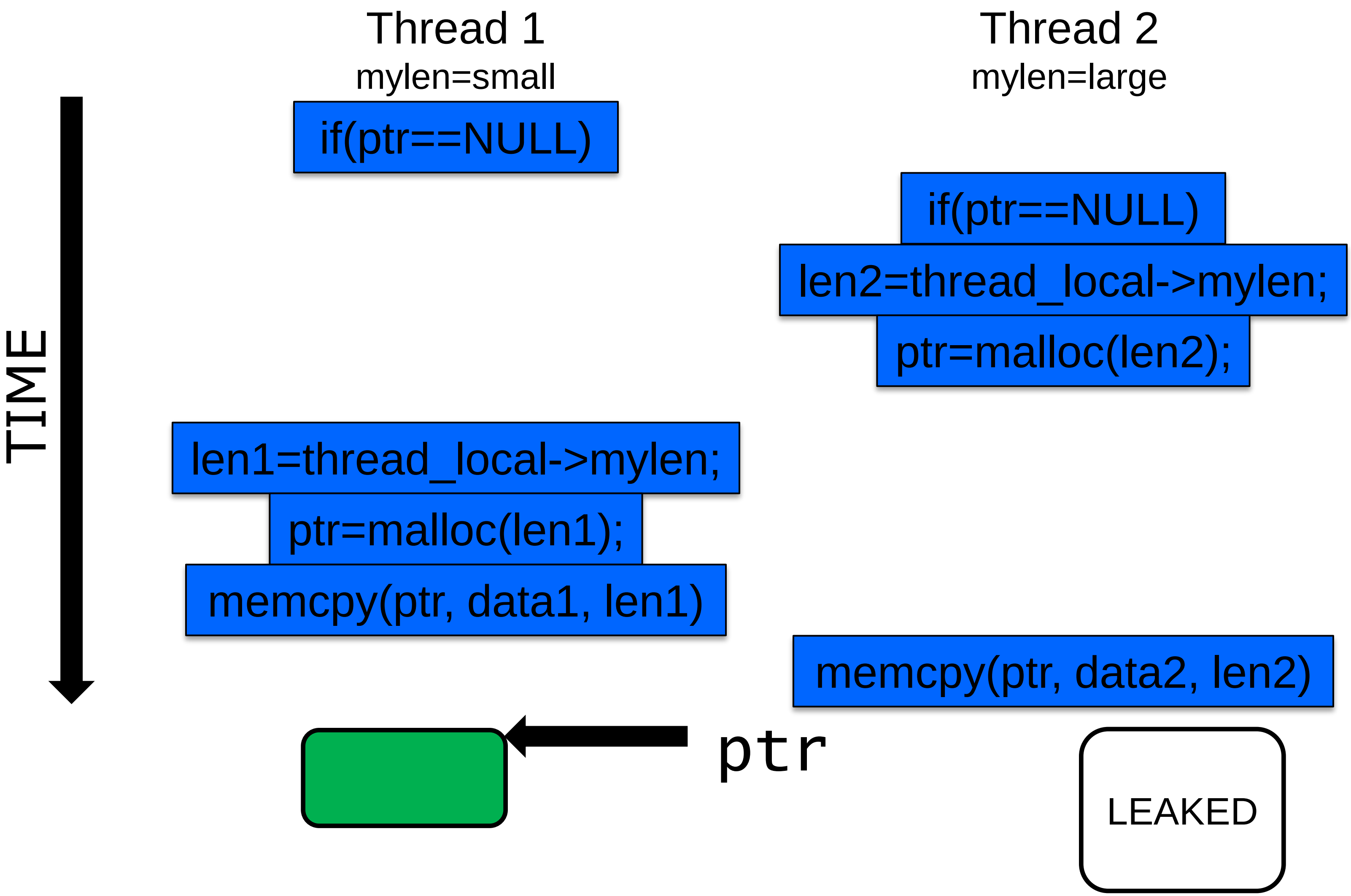
Concurrency Bugs Matter NOW



Concurrency Bugs Matter NOW



Concurrency Bugs Matter NOW



Concurrency Bugs Matter NOW

TIME

Thread 1
mylen=small

```
if(ptr==NULL)
```

Thread 2
mylen=large

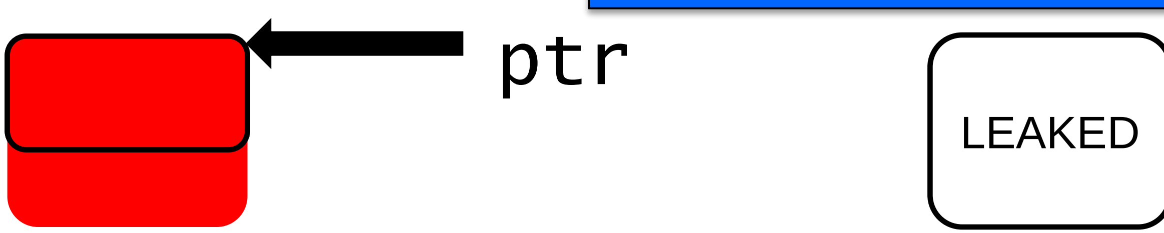
```
if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
```

Thread 1

```
len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)
```

Thread 2

```
memcpy(ptr, data2, len2)
```



This Talk in One Sentence

Speed up software race detection with existing hardware support.

Software Data Race Detection

- Add checks around every memory access
- Find inter-thread sharing events
- Synchronization between write-shared accesses?
    - No? Data race.

Example of Data Race Detection

TIME

Thread 1
mylen=small

```
if(ptr==NULL)
len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)
```

Thread 2
mylen=large

```
if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2)
```

Example of Data Race Detection

TIME

Thread 1
mylen=small

```
if(pt   ULL)
len1=thread_lo  l->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)
```

Thread 2
mylen=large

```
if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2)
```

Example of Data Race Detection

TIME

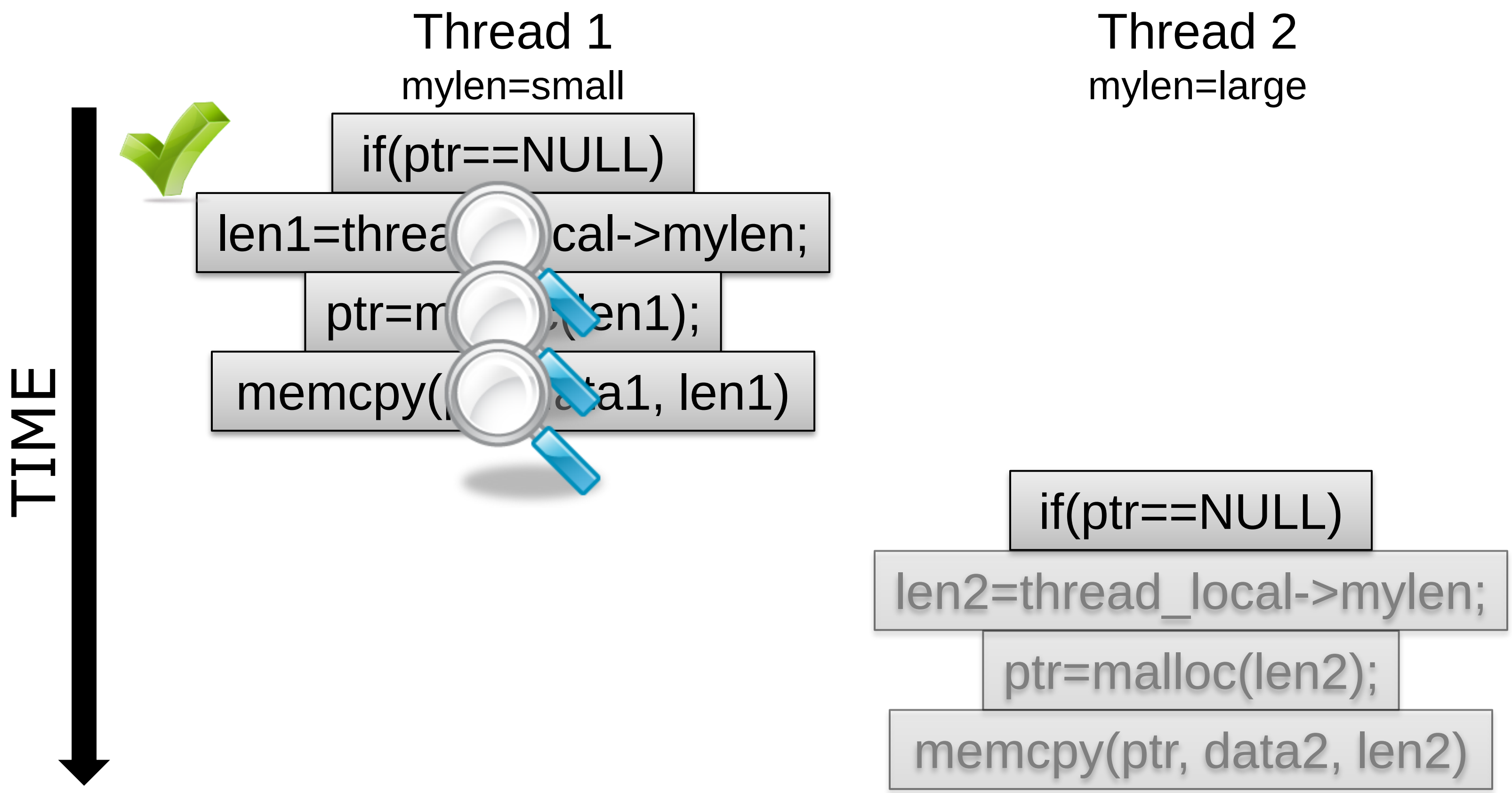
Thread 1
mylen=small

```
if(ptr==NULL)
len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)
```

Thread 2
mylen=large

```
if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2)
```

Example of Data Race Detection



Example of Data Race Detection

TIME

Thread 1

mylen=small

```
if(ptr==NULL)
len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)
```

Thread 2

mylen=large

```
if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2)
```

Example of Data Race Detection

TIME

Thread 1
mylen=small

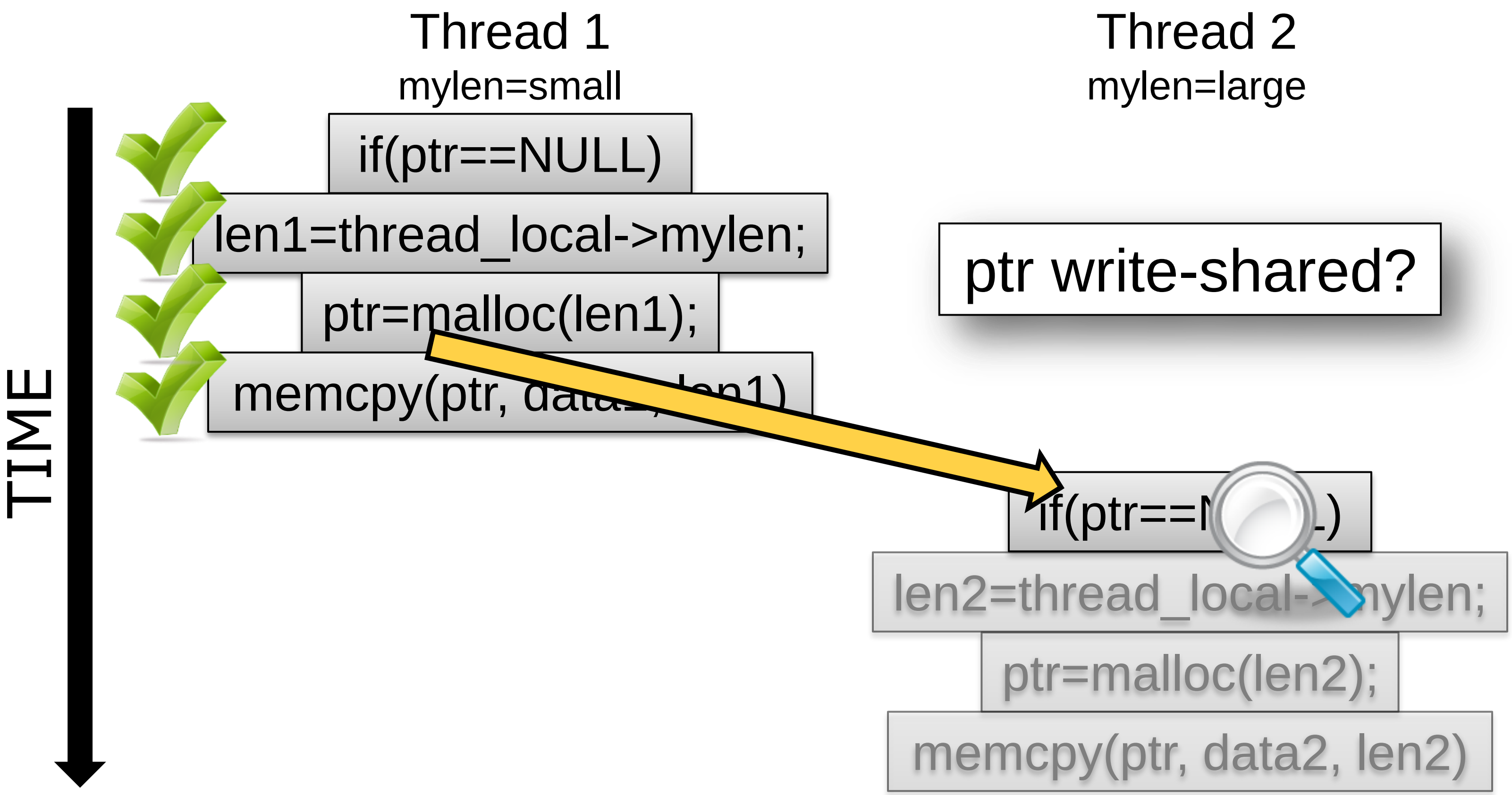
```
if(ptr==NULL)
len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)
```

Thread 2
mylen=large

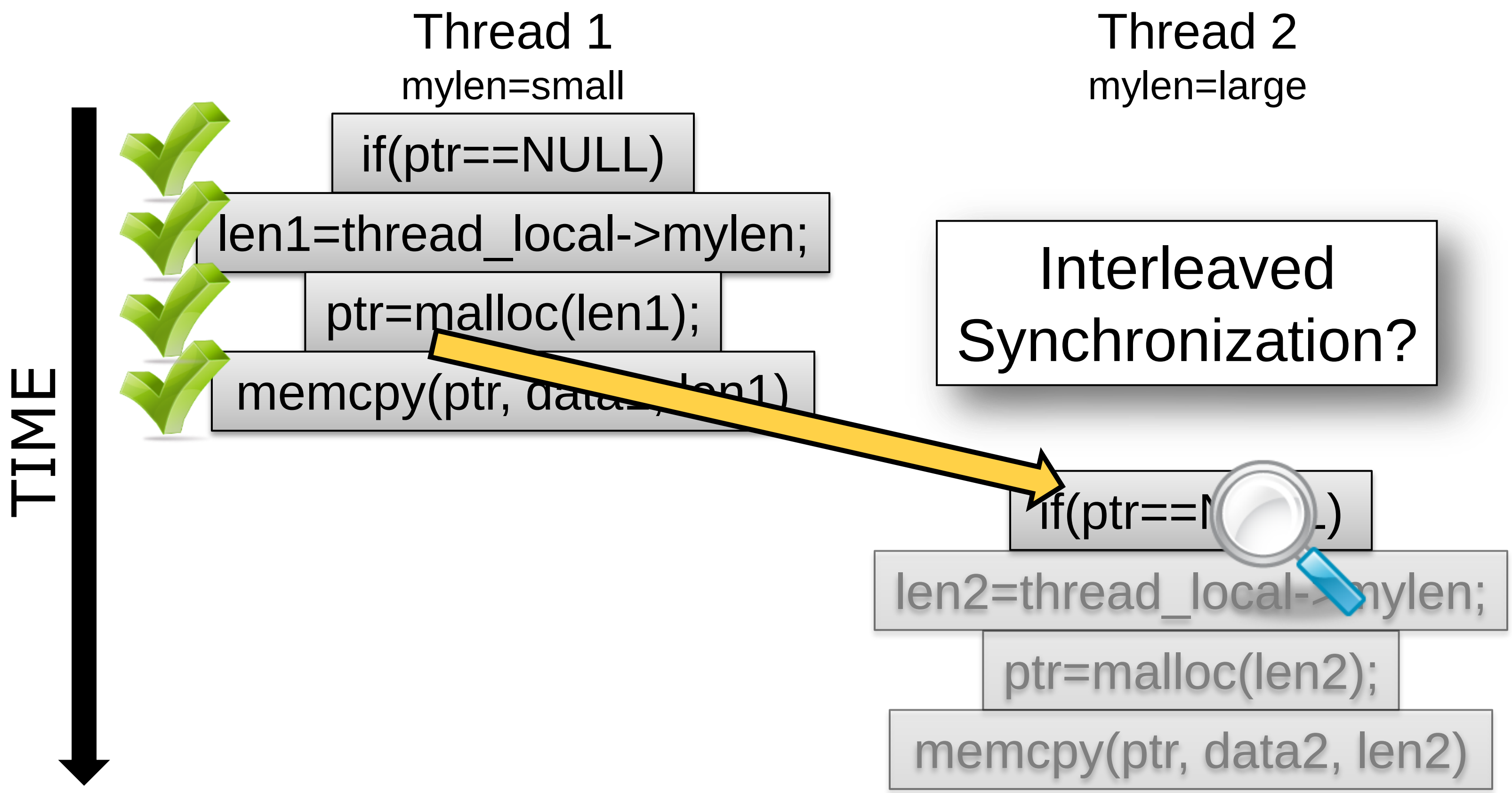
ptr write-shared?

```
if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2)
```

Example of Data Race Detection



Example of Data Race Detection



Example of Data Race Detection

TIME

Thread 1

mylen=small

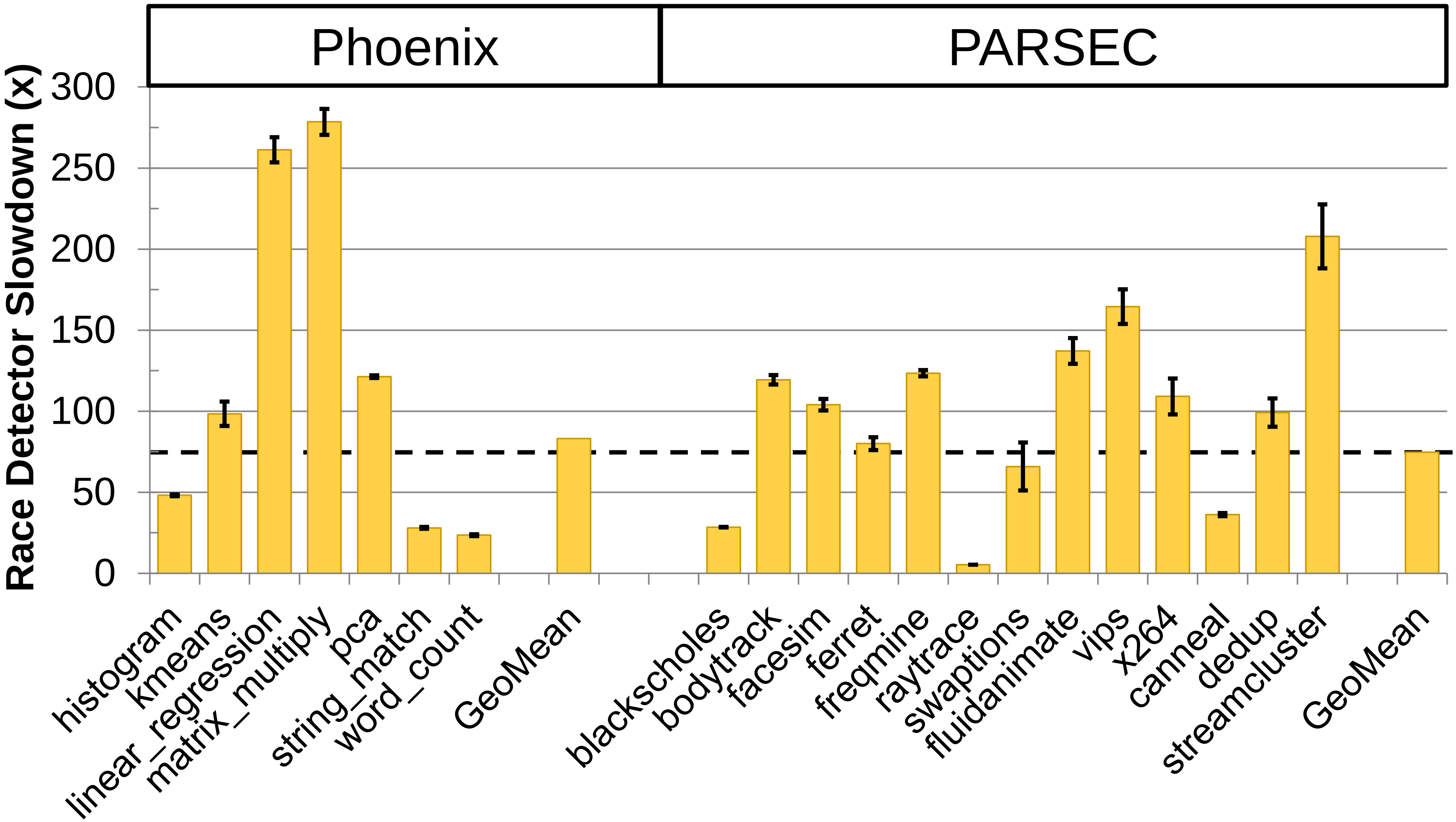
```
if(ptr==NULL)
len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)
```

Thread 2

mylen=large

```
if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2)
```

SW Race Detection is Slow



Goal of this Work

Accelerate Software Data Race Detection

Technique #1: *Making it Fast*

Demand-Driven Data Race Detection

Technique #2: *Keeping it Real*

Find sharing events with existing HW

Inter-thread Sharing is What's Important

"Data races ... are failures in programs that **access and update shared data** in critical sections" – Netzer & Miller, 1992

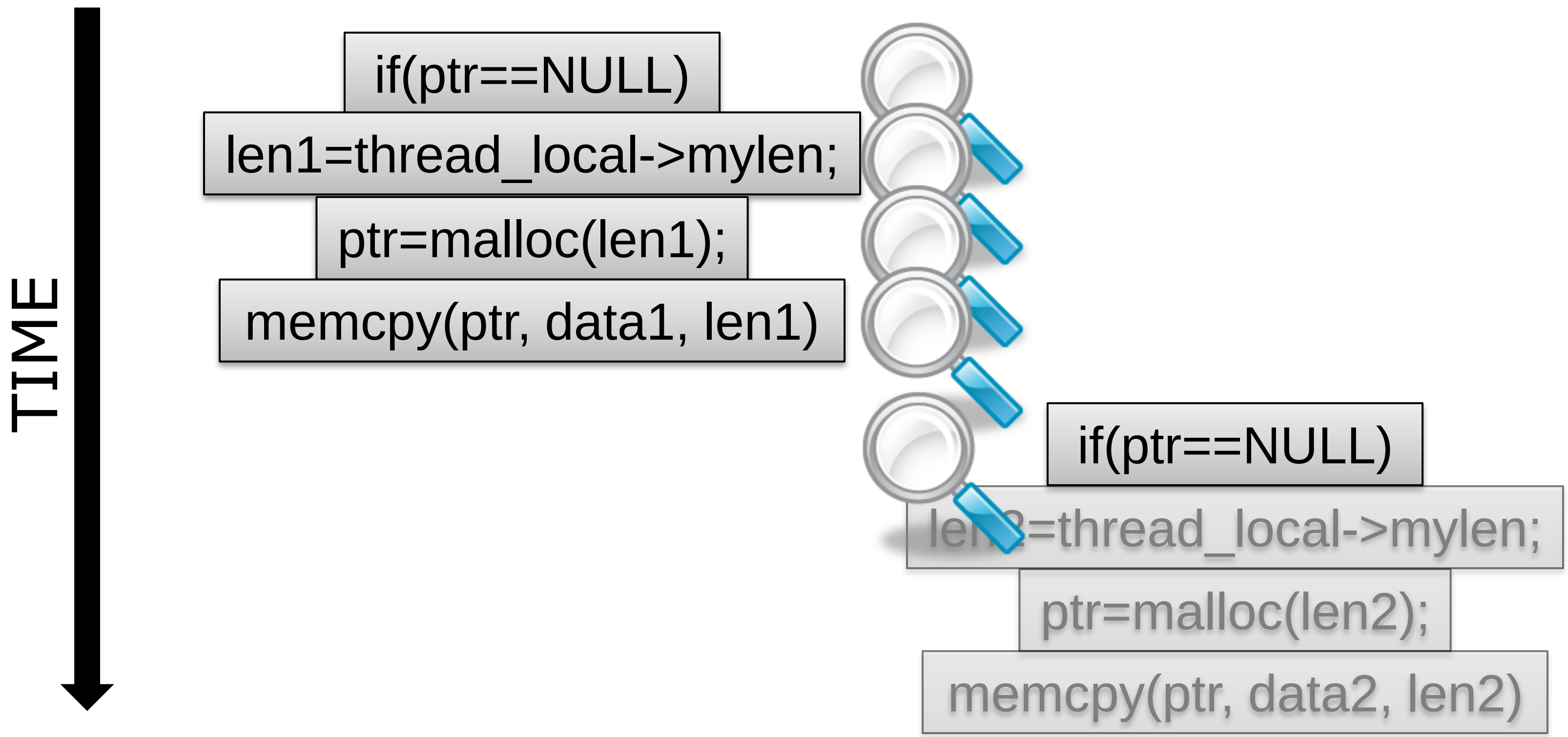
TIME

```
if(ptr==NULL)
len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)
```

```
if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2)
```

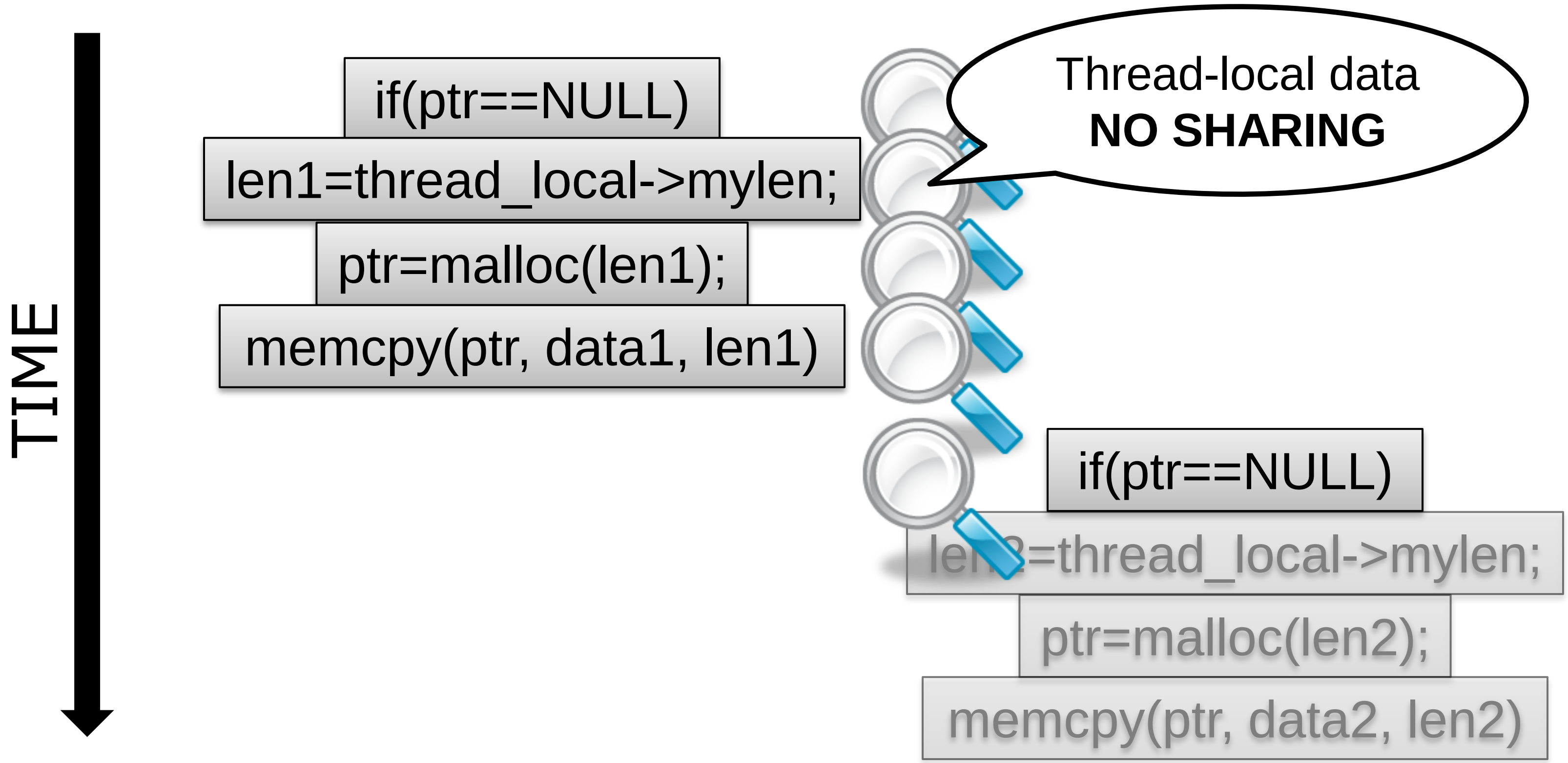
Inter-thread Sharing is What's Important

"Data races ... are failures in programs that **access and update shared data** in critical sections" – Netzer & Miller, 1992



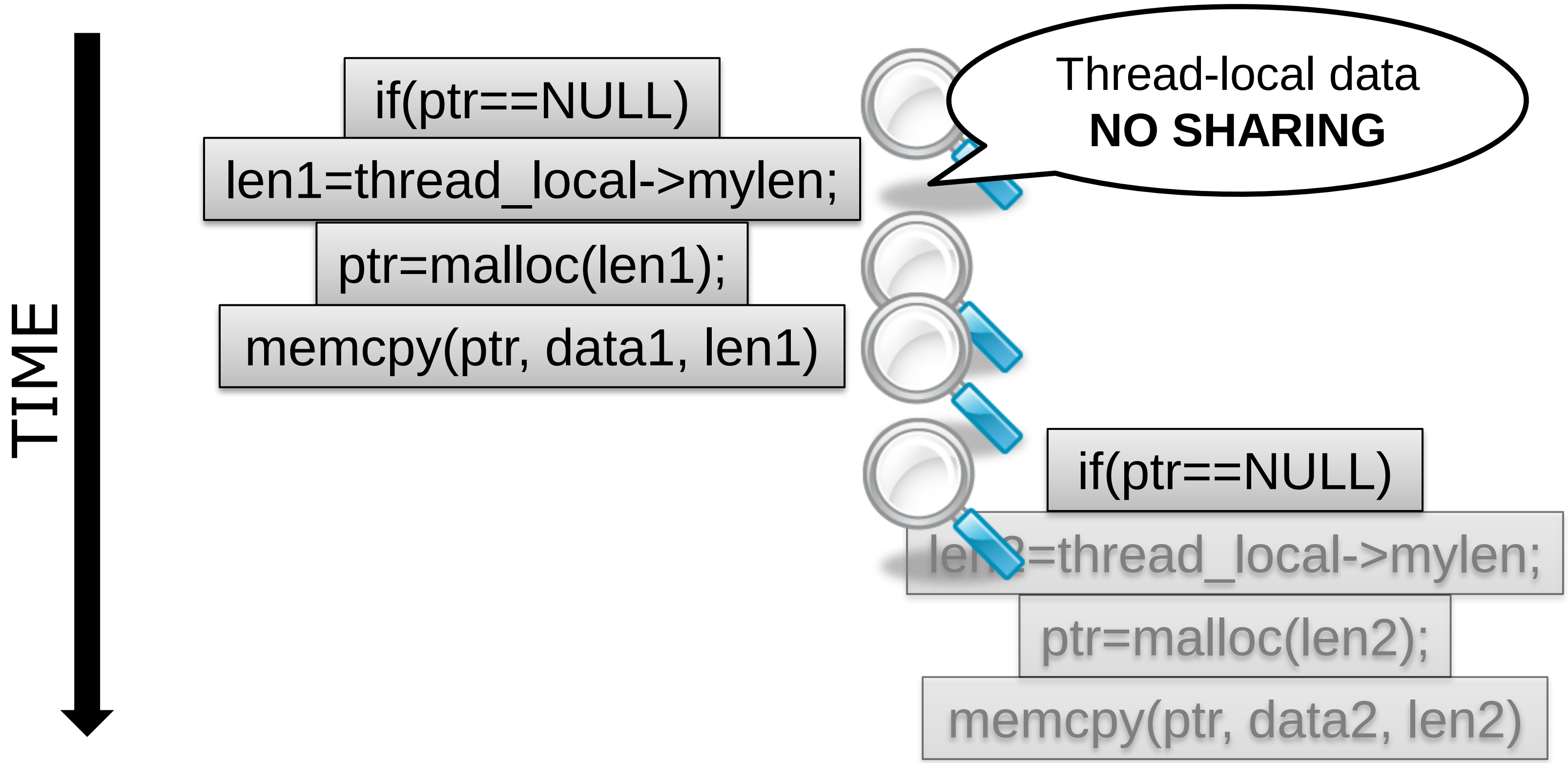
Inter-thread Sharing is What's Important

"Data races ... are failures in programs that **access and update shared data** in critical sections" – Netzer & Miller, 1992



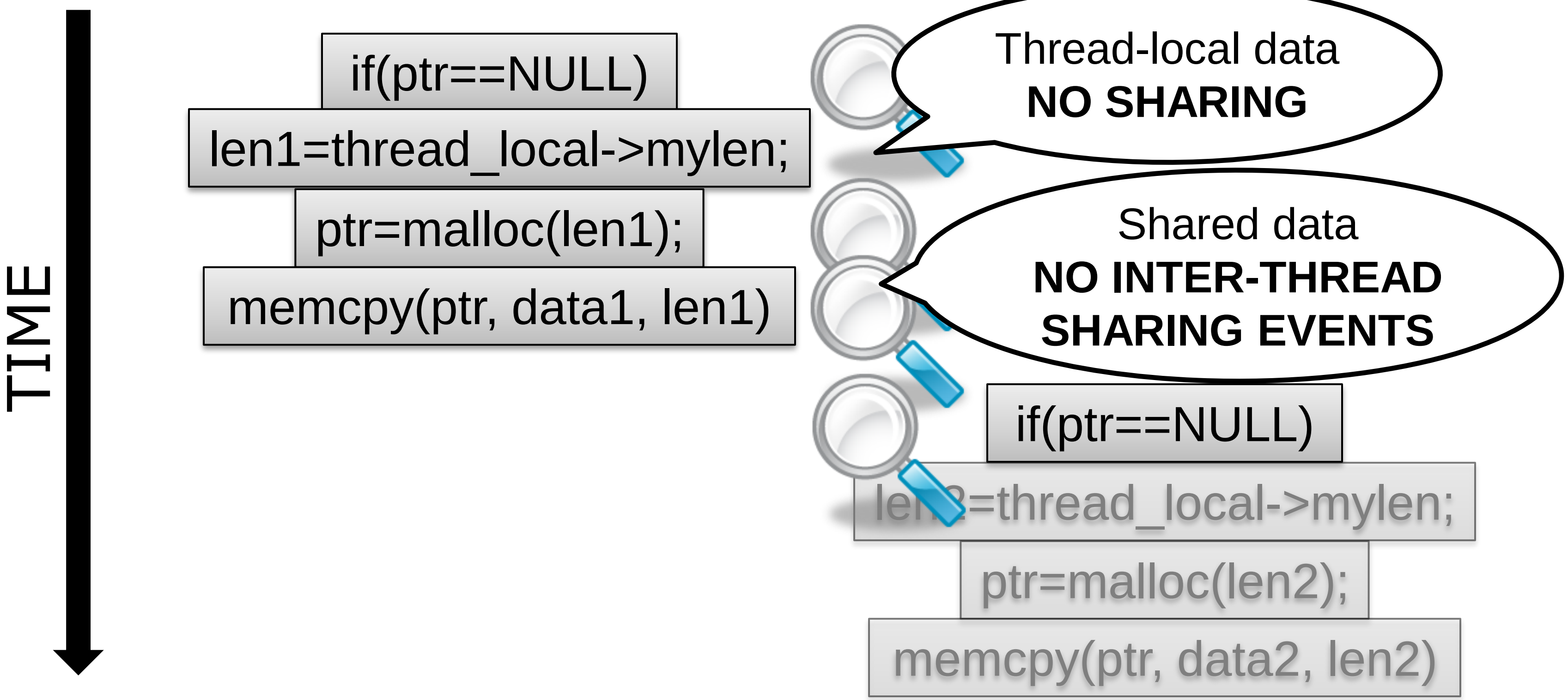
Inter-thread Sharing is What's Important

"Data races ... are failures in programs that **access and update shared data** in critical sections" – Netzer & Miller, 1992



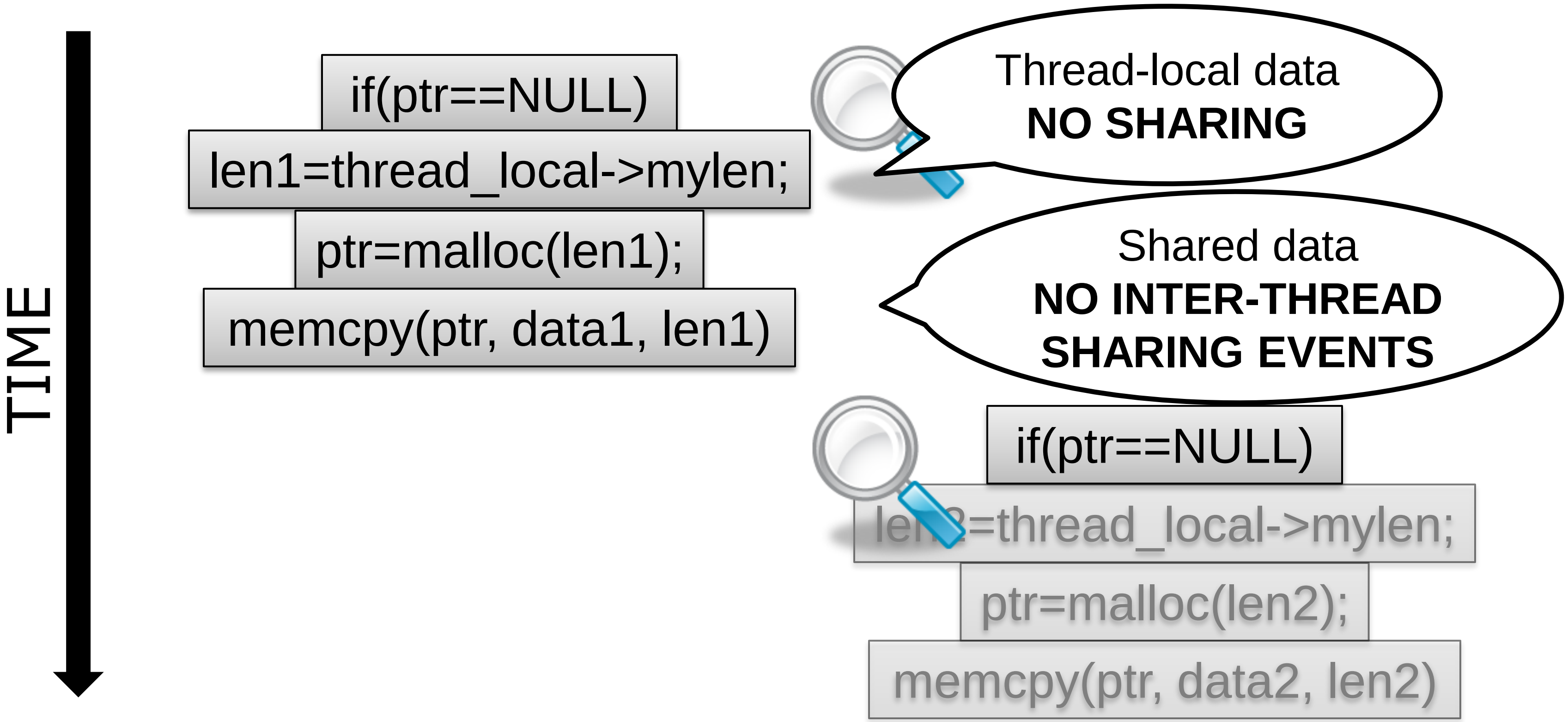
Inter-thread Sharing is What's Important

"Data races ... are failures in programs that **access and update shared data** in critical sections" – Netzer & Miller, 1992



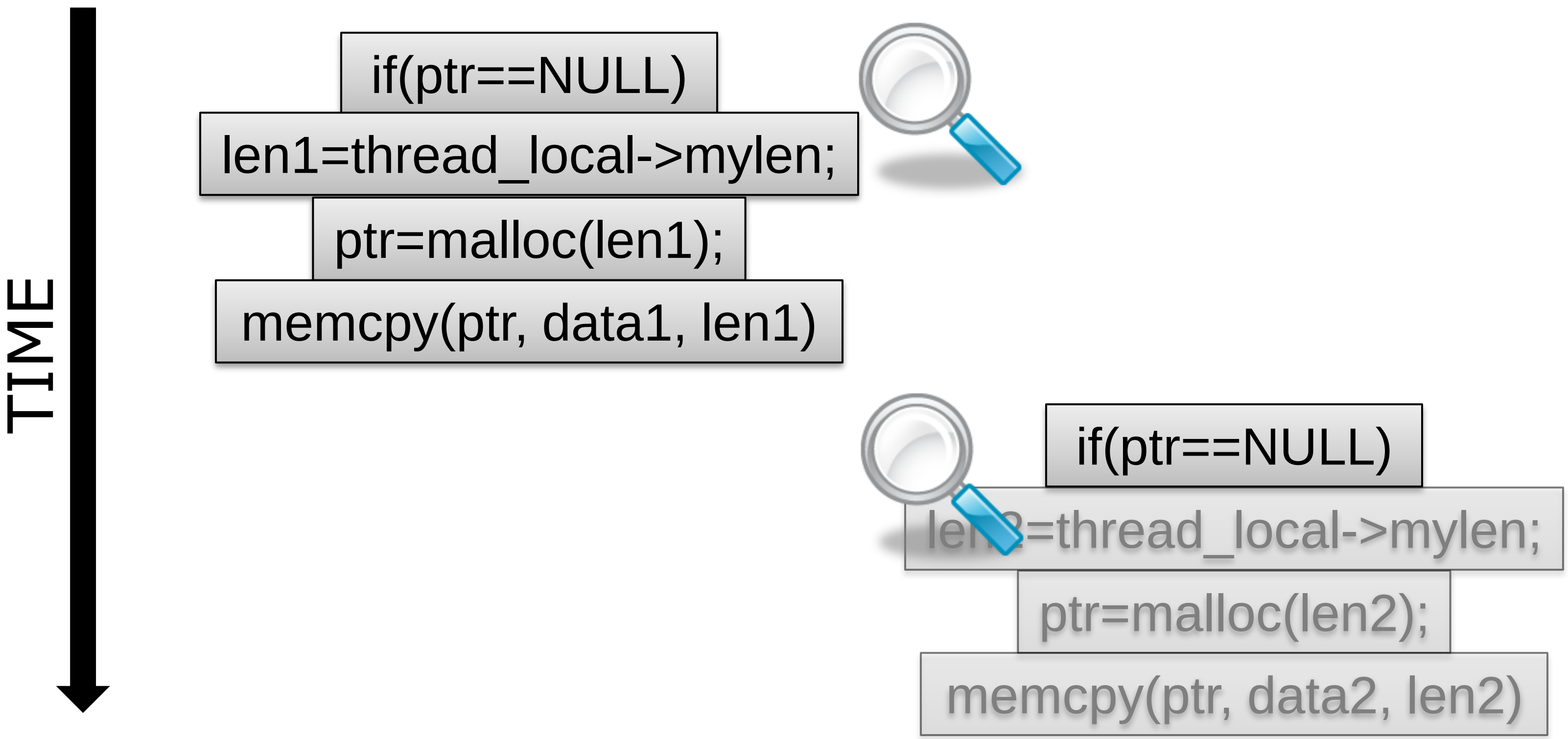
Inter-thread Sharing is What's Important

"Data races ... are failures in programs that **access and update shared data** in critical sections" – Netzer & Miller, 1992



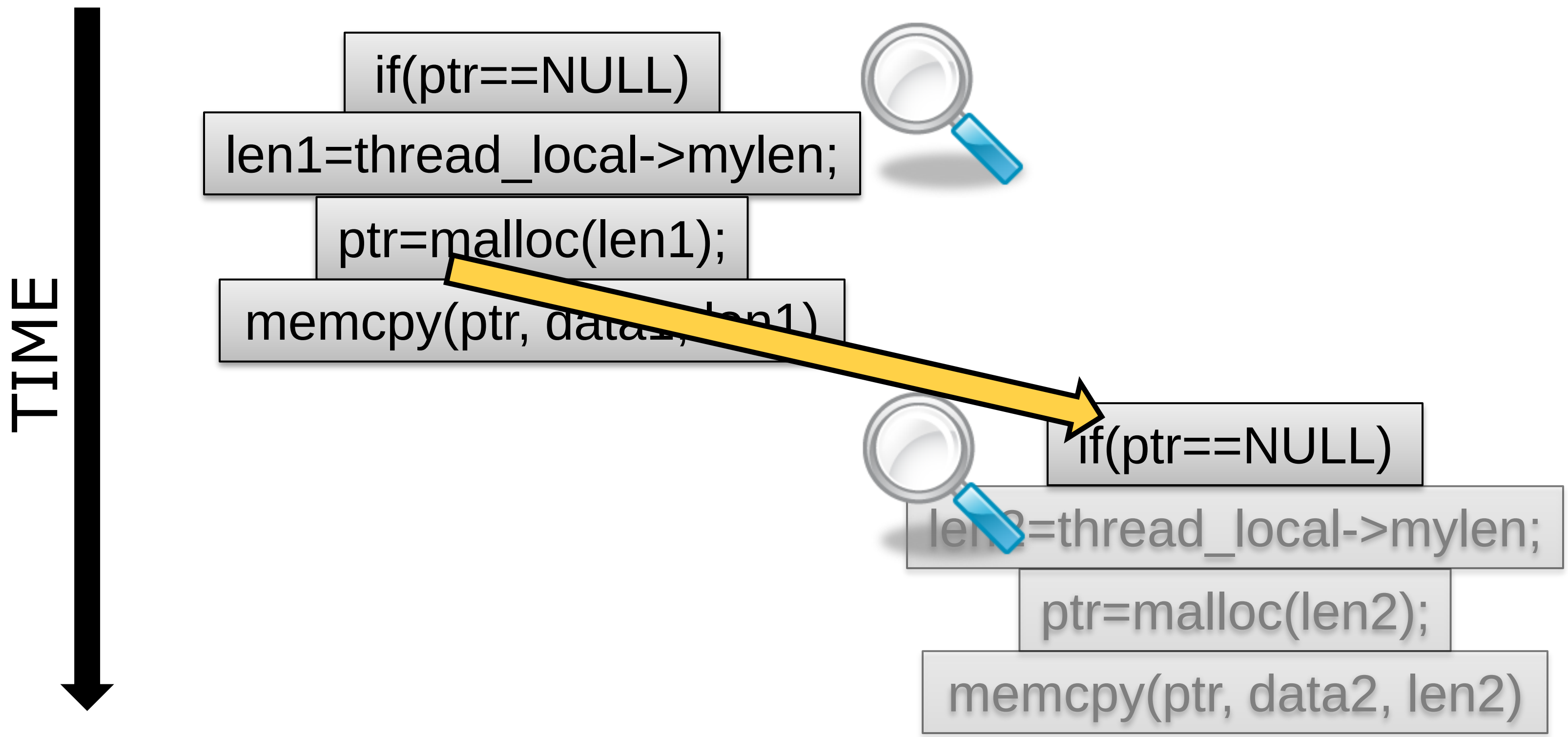
Inter-thread Sharing is What’s Important

“Data races ... are failures in programs that **access and update shared data** in critical sections” – Netzer & Miller, 1992

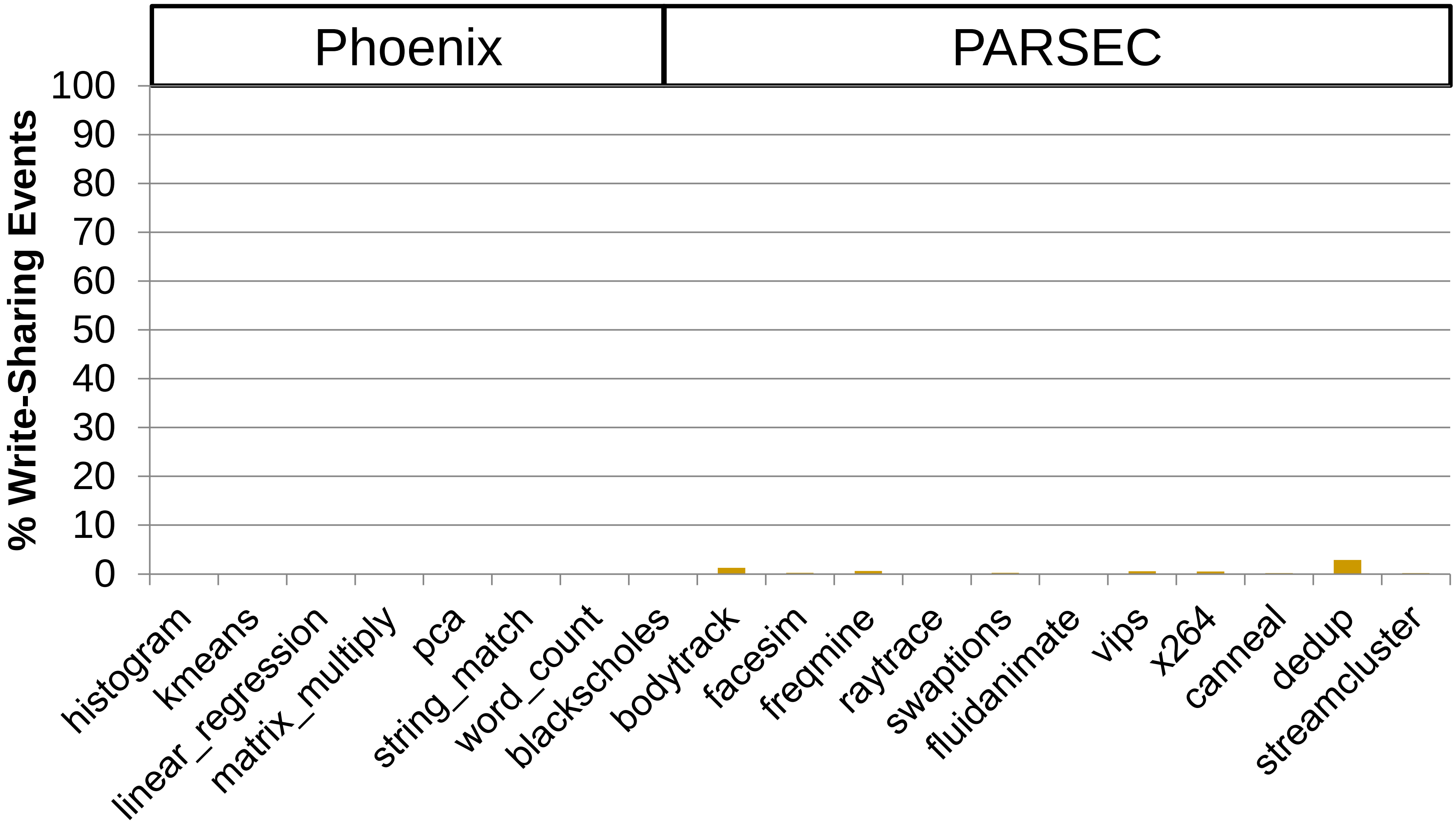


Inter-thread Sharing is What's Important

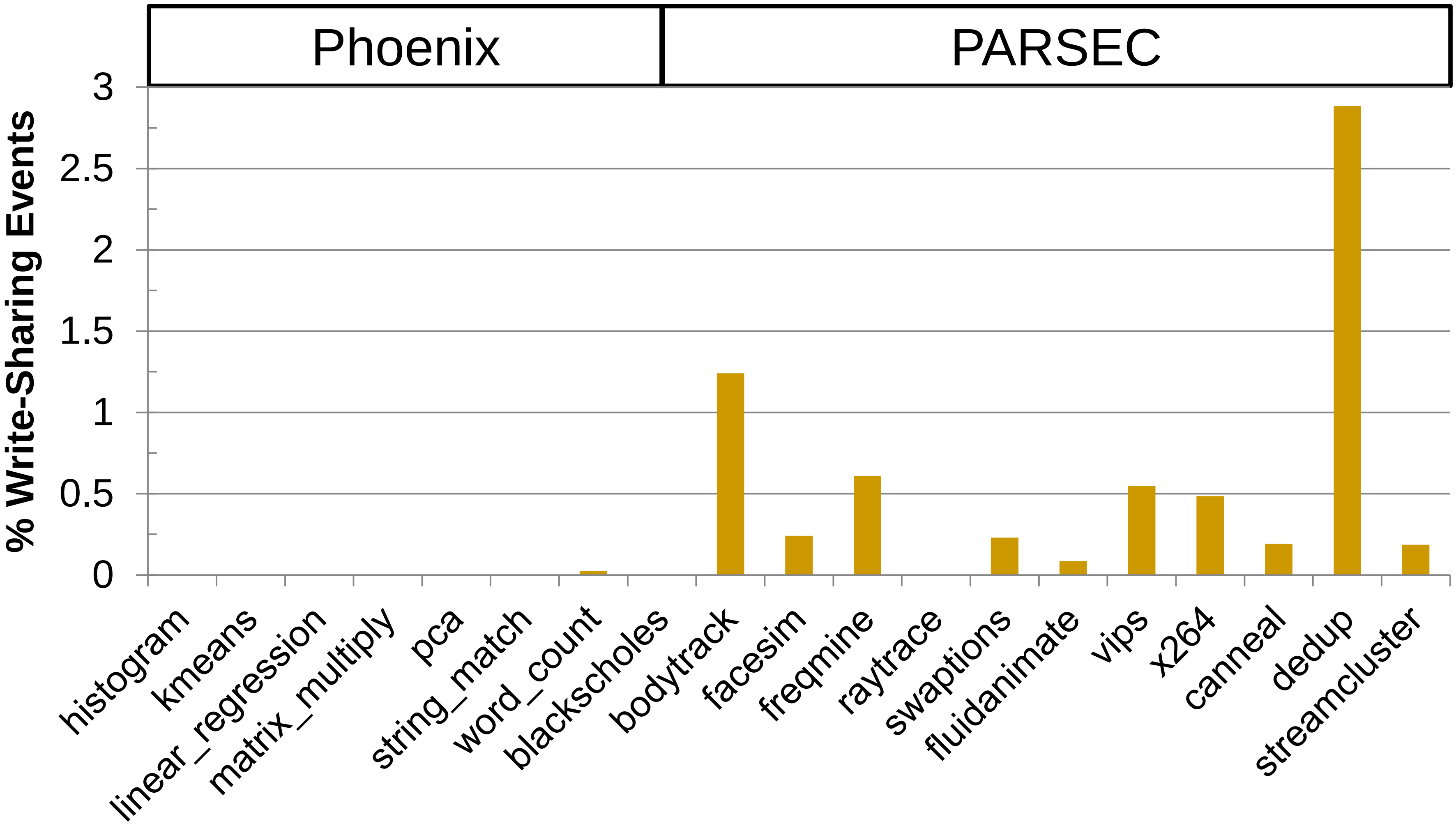
"Data races ... are failures in programs that **access and update shared data** in critical sections" – Netzer & Miller, 1992



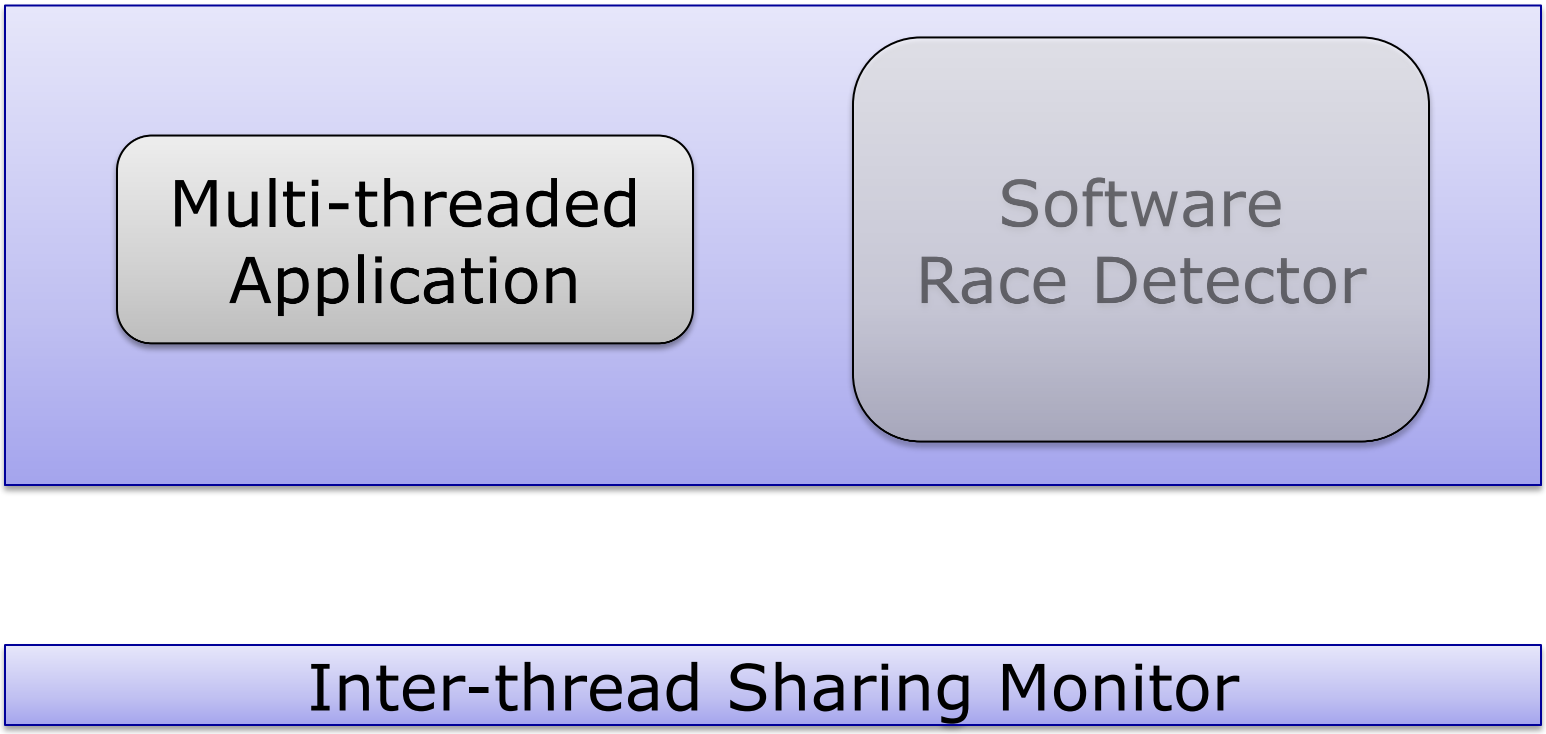
Very Little Inter-Thread Sharing



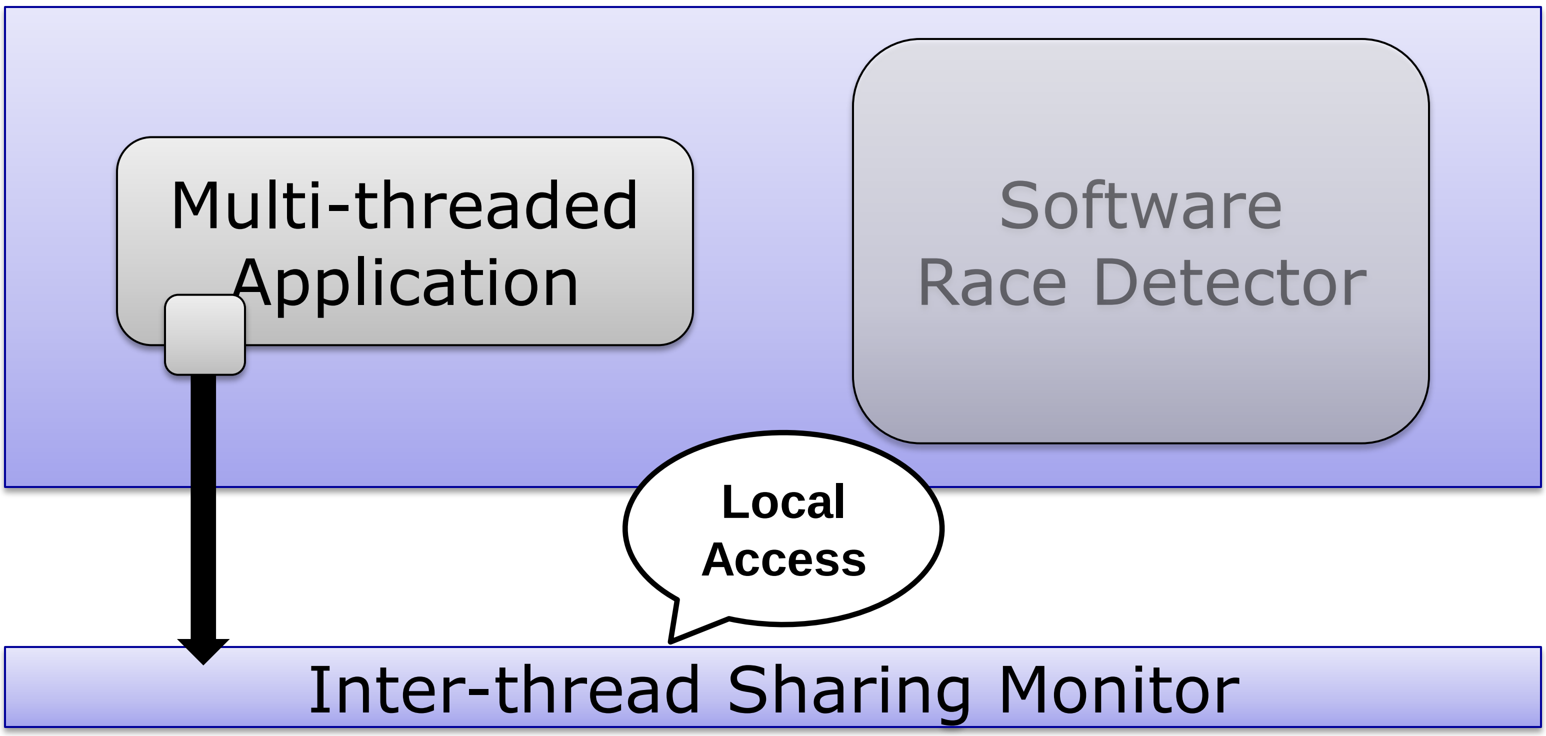
Very Little Inter-Thread Sharing



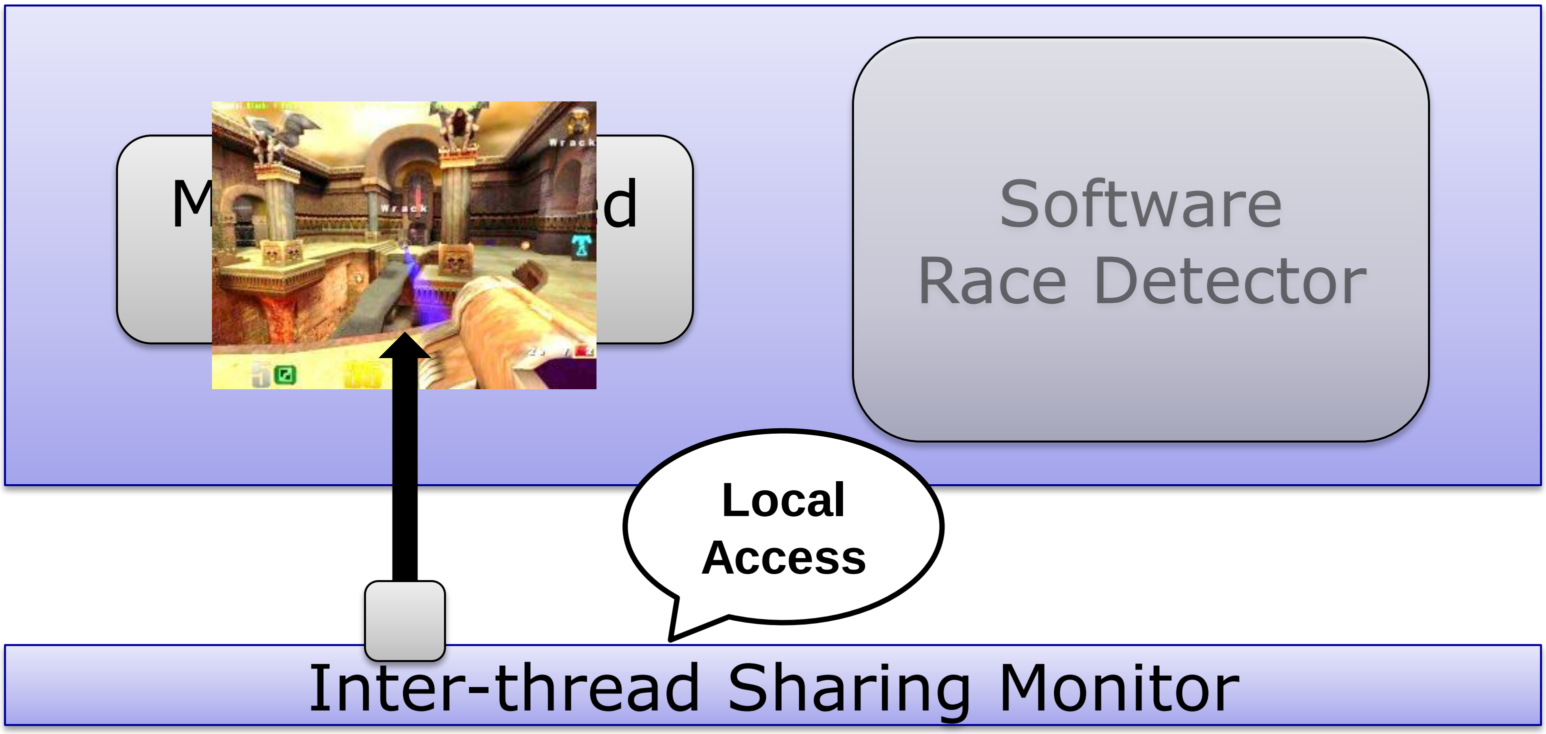
Technique 1: Demand-Driven Analysis



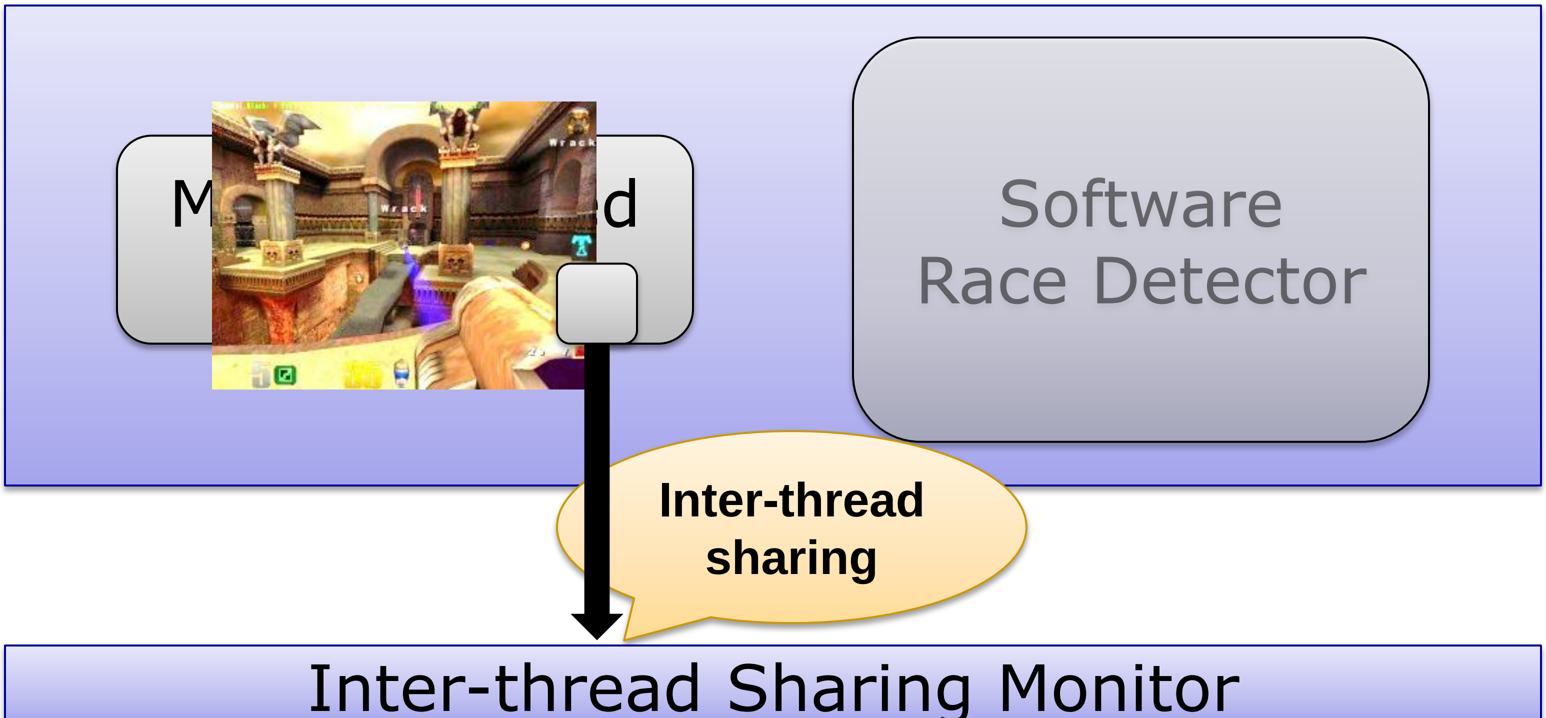
Technique 1: Demand-Driven Analysis



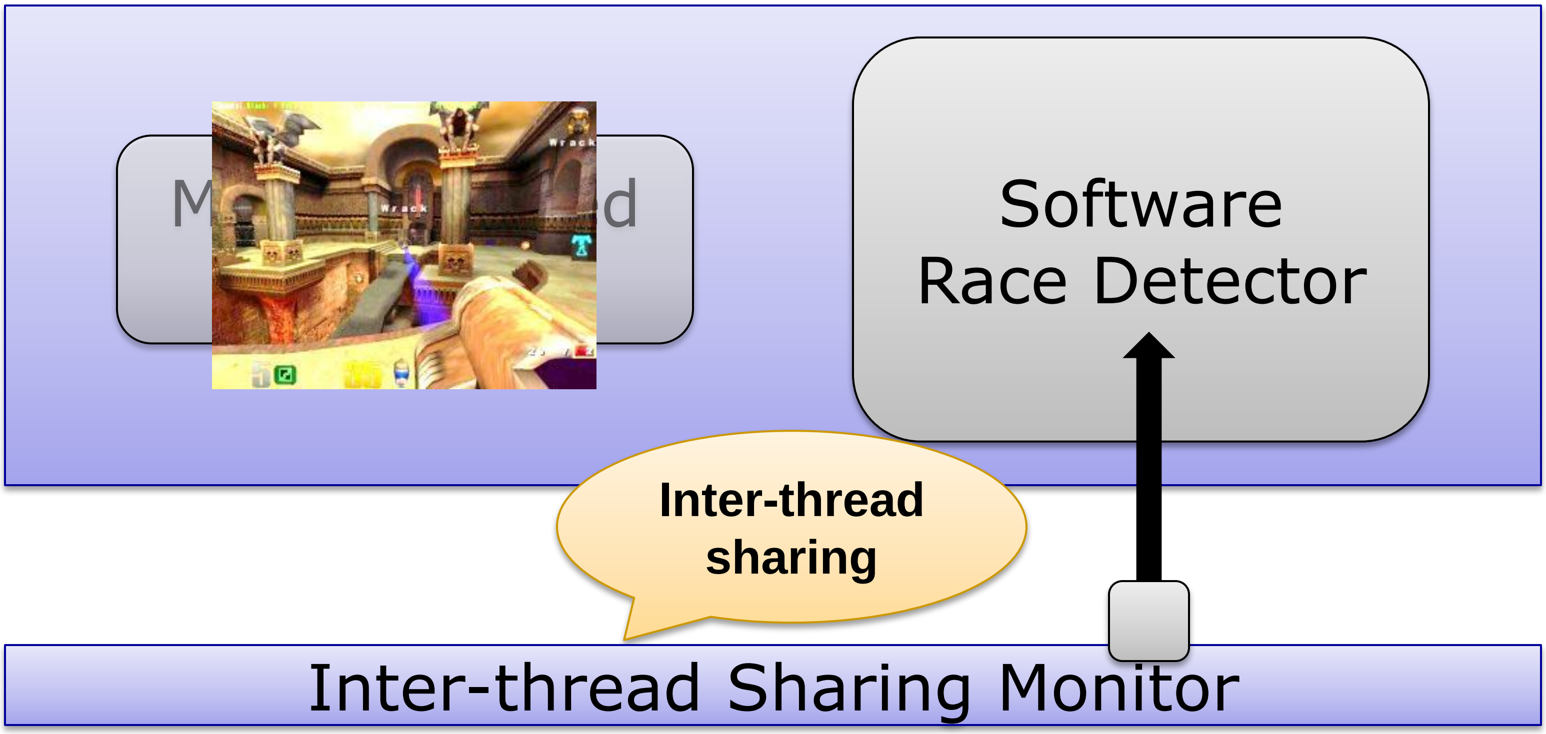
Technique 1: Demand-Driven Analysis



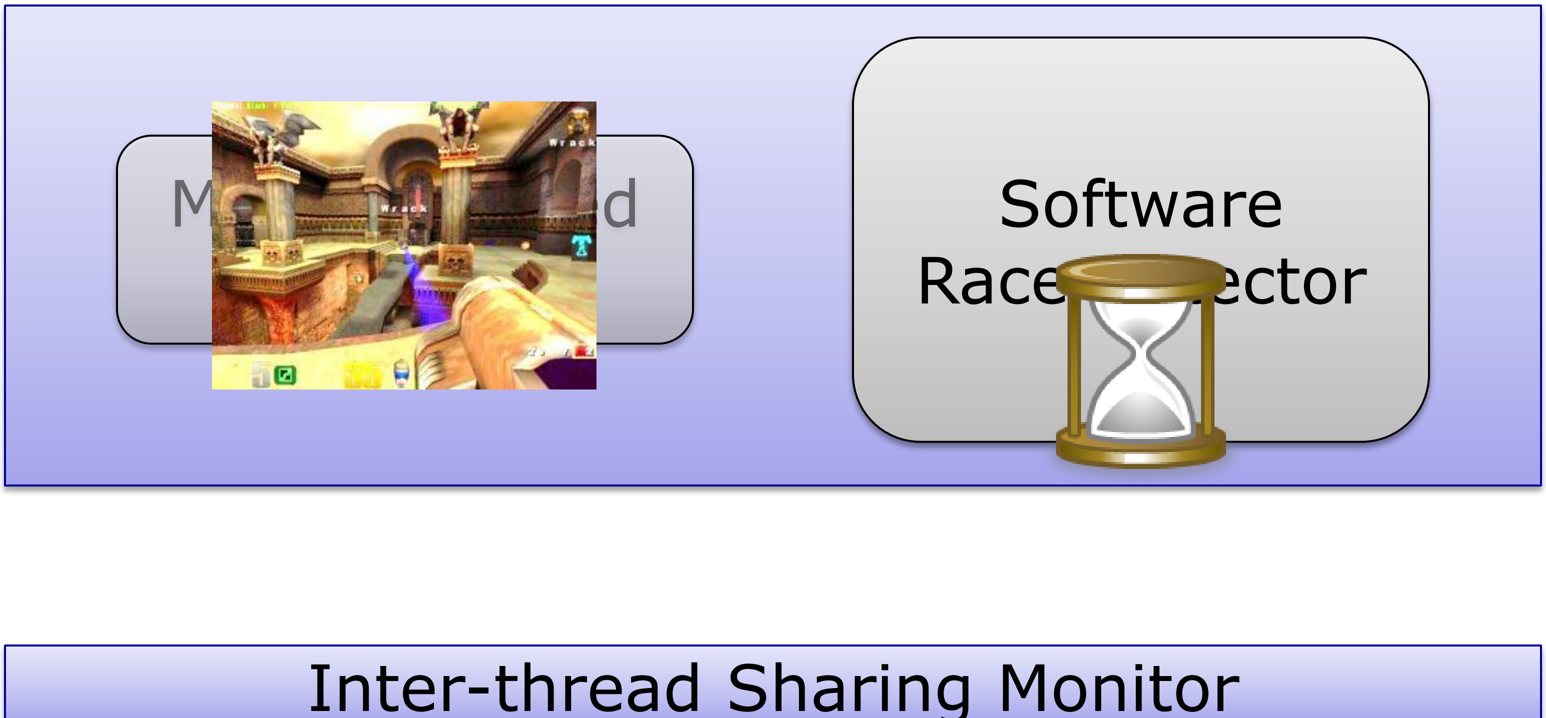
Technique 1: Demand-Driven Analysis



Technique 1: Demand-Driven Analysis



Technique 1: Demand-Driven Analysis



Technique 1: Demand-Driven Analysis

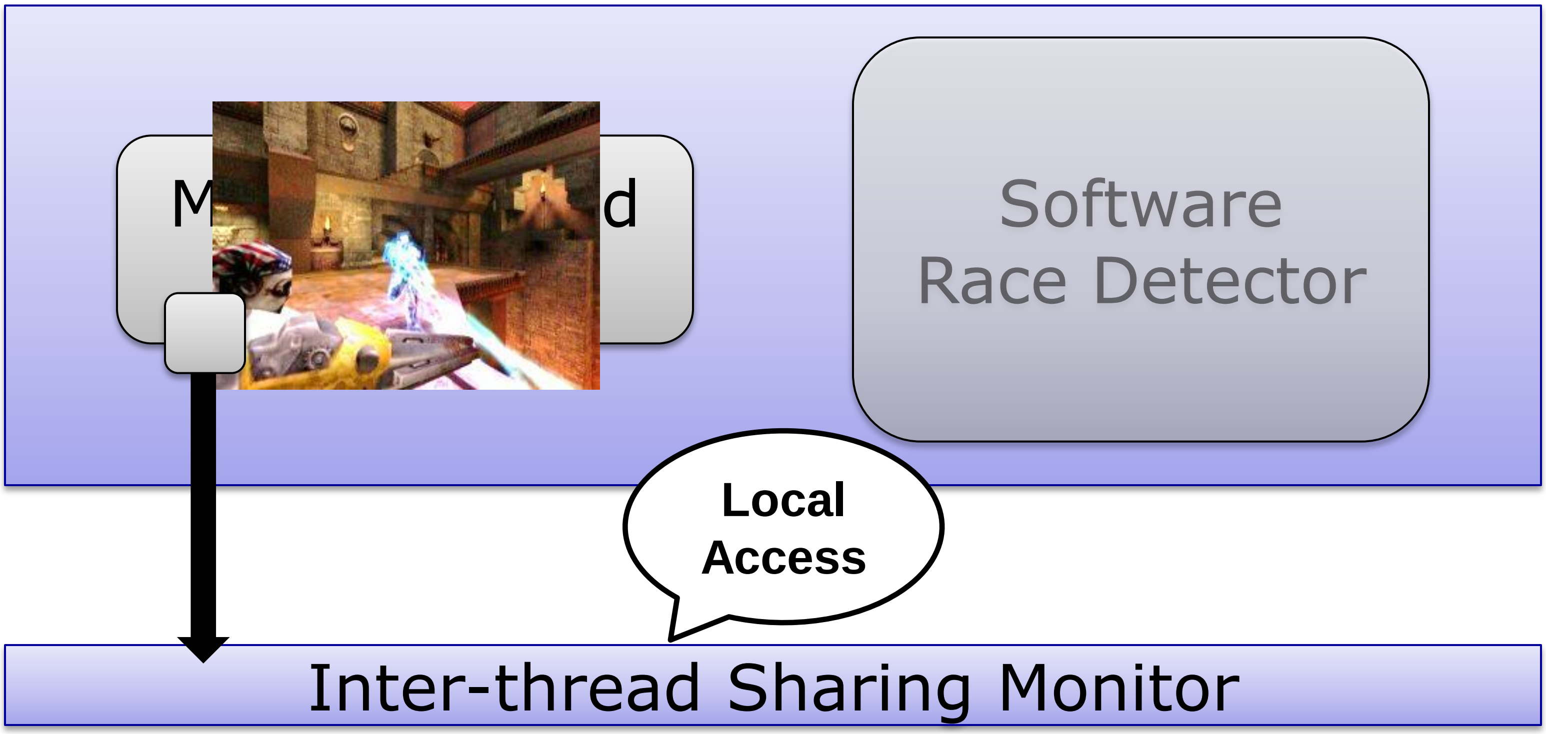
M

d

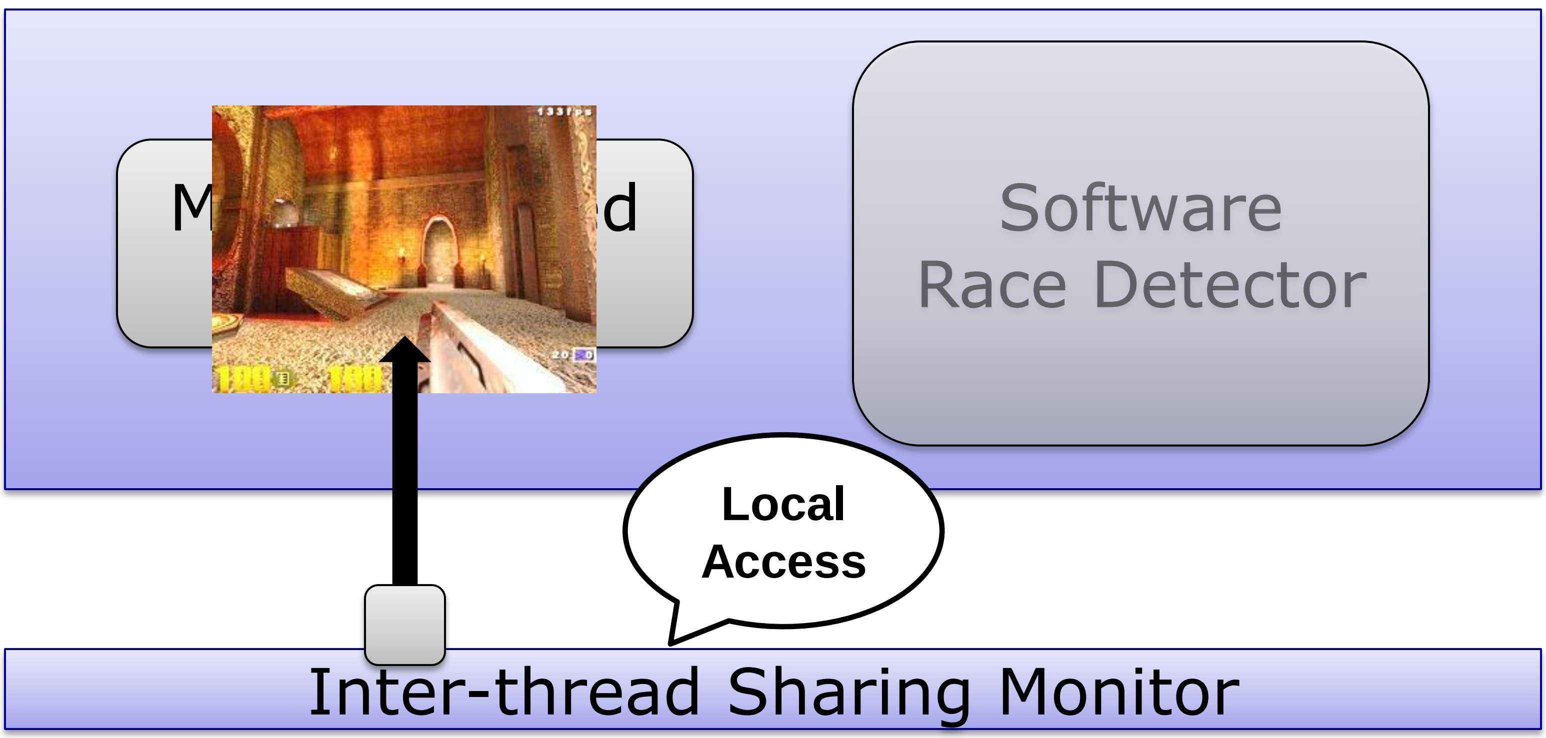
Software
Race Detector

Inter-thread Sharing Monitor

Technique 1: Demand-Driven Analysis



Technique 1: Demand-Driven Analysis



Technique 1: Demand-Driven Analysis

Software Race Detector

Inter-thread Sharing Monitor

Inter-thread Sharing Monitor

- Check each memory op. for write-sharing

- Signal software race detector on sharing

- Possible to do in software
  + Can be built now with instrumentation
  – Slow. May take as long as race detection

Ideal Hardware Sharing Detector

- Follow read/write sets of threads

Thread 1
WRITE Y

Sharing Monitor	
T1	T2
R: Ø	R: Ø
W: Ø	W: Ø

Thread 2
READ Y

Ideal Hardware Sharing Detector

- Follow read/write sets of threads

Thread 1
WRITE Y

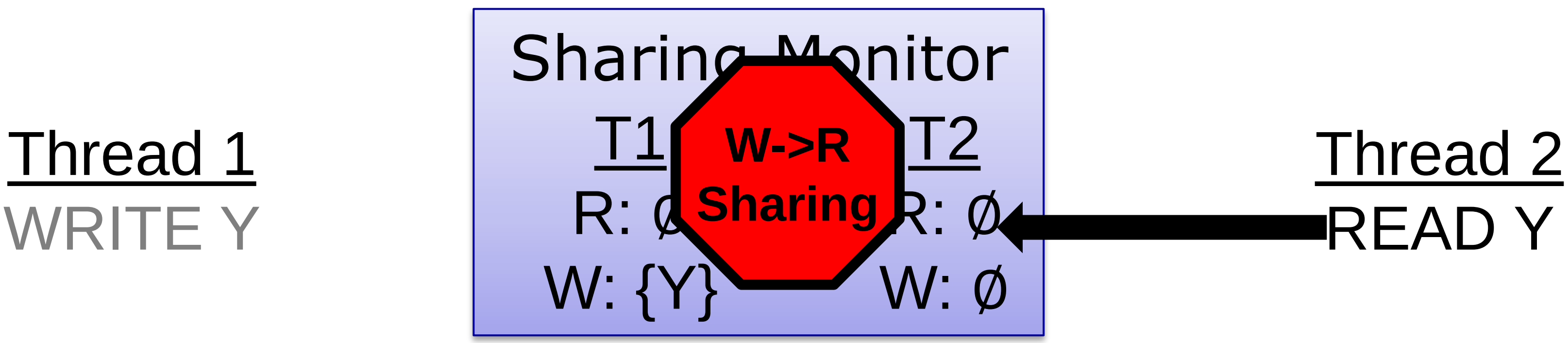
Sharing Monitor

T1	T2
R: Ø	R: Ø
W: {Y}	W: Ø

Thread 2
READ Y

Ideal Hardware Sharing Detector

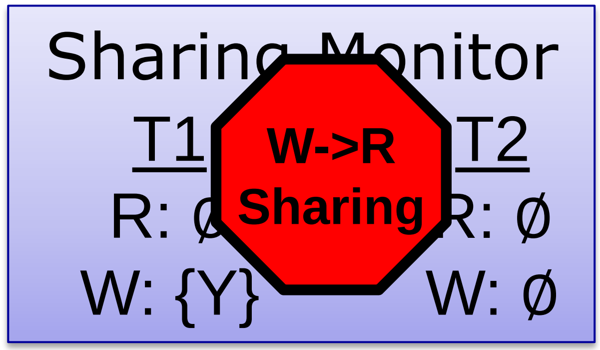
- Follow read/write sets of threads



Ideal Hardware Sharing Detector

- Follow read/write sets of threads

Thread 1
WRITE Y



Thread 2
READ Y

- Fast user-level faults

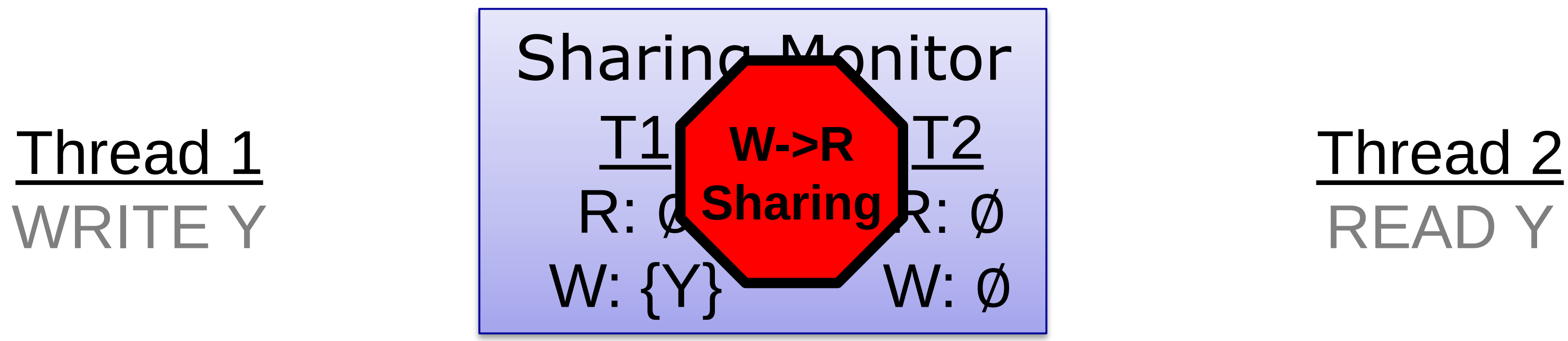
Multi-threaded Application

Software Race Detector

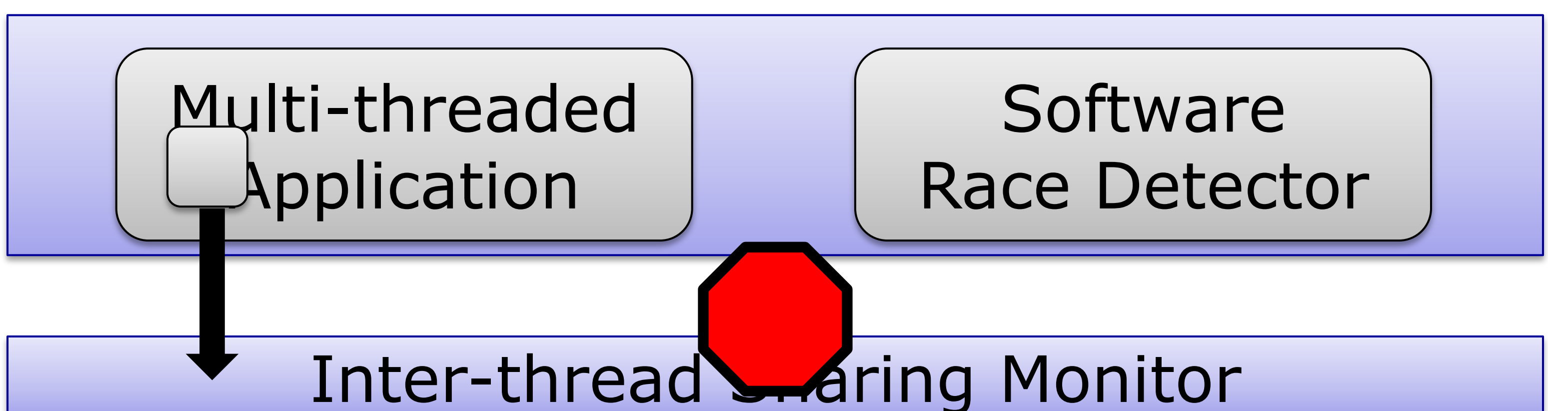
Inter-thread Sharing Monitor

Ideal Hardware Sharing Detector

- Follow read/write sets of threads

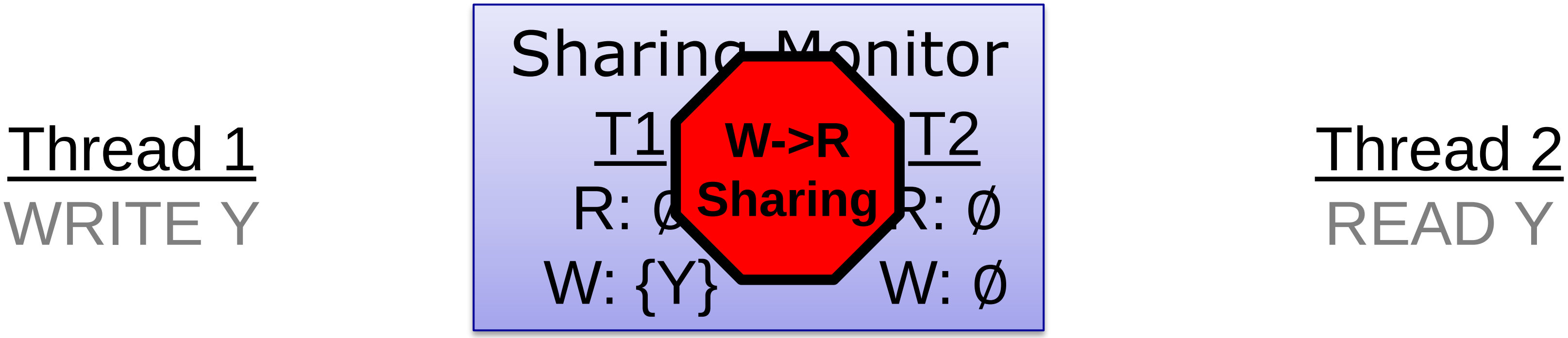


- Fast user-level faults

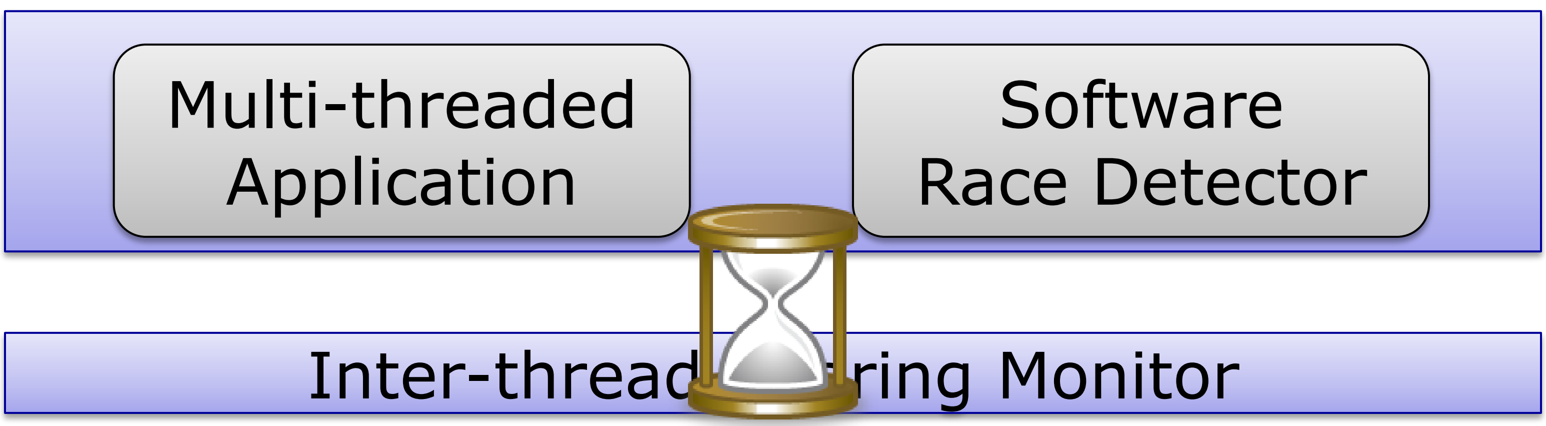


Ideal Hardware Sharing Detector

- Follow read/write sets of threads

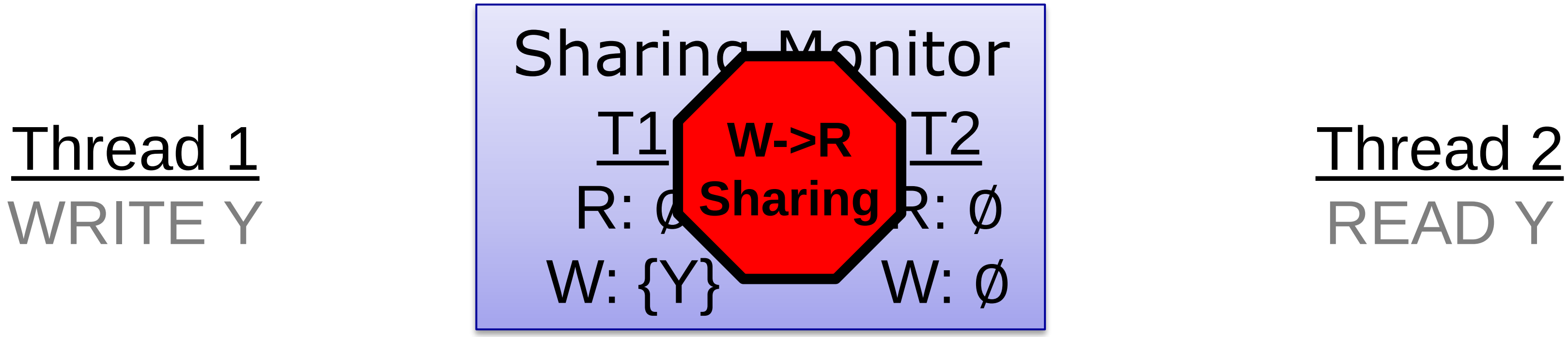


- Fast user-level faults

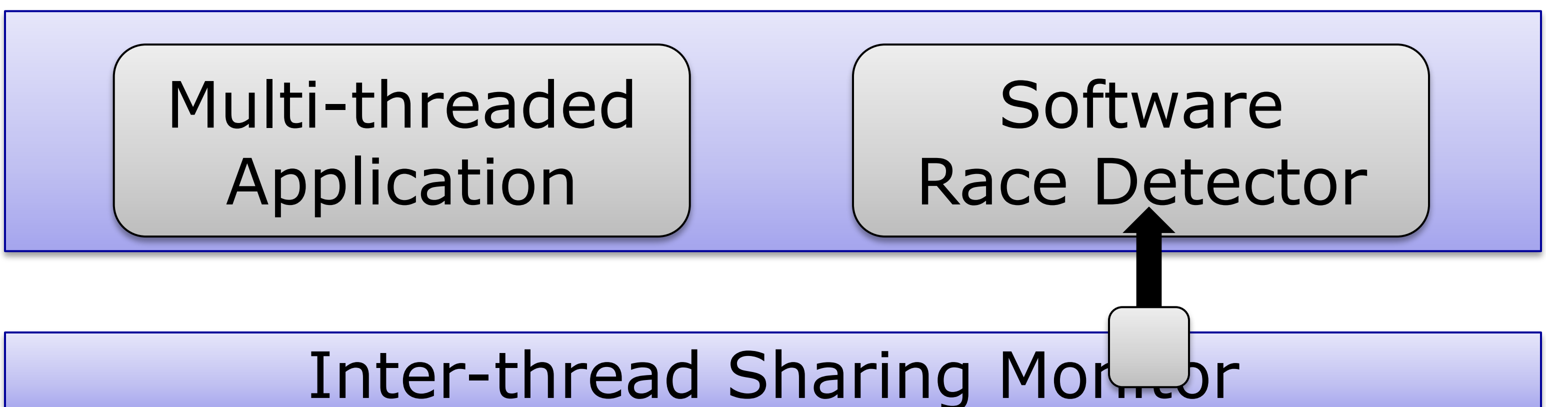


Ideal Hardware Sharing Detector

- Follow read/write sets of threads

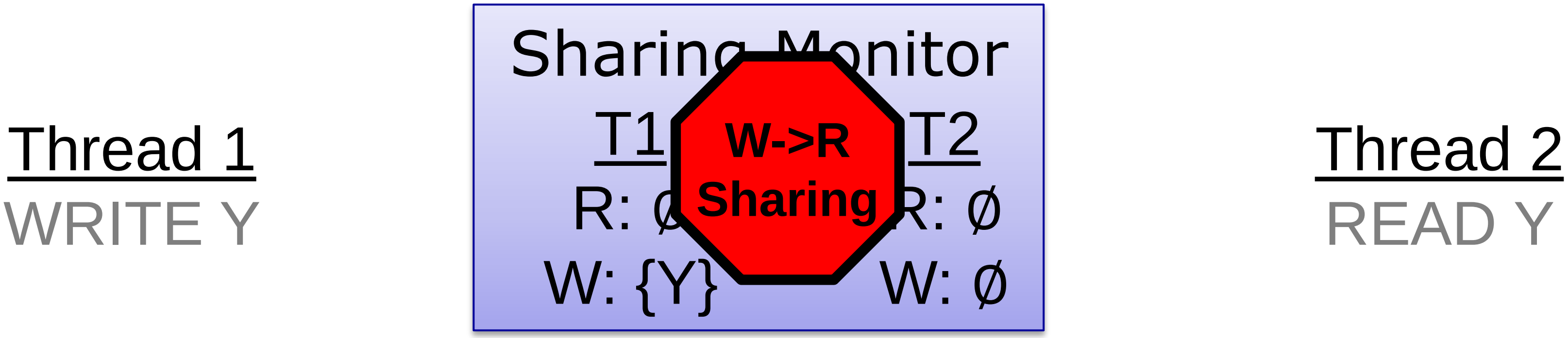


- Fast user-level faults

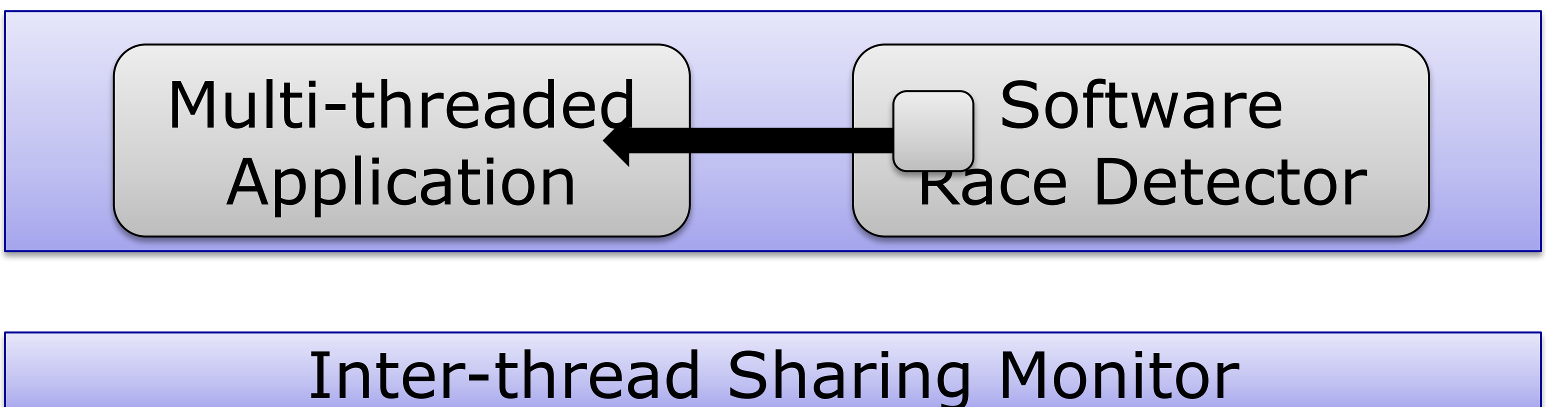


Ideal Hardware Sharing Detector

- Follow read/write sets of threads

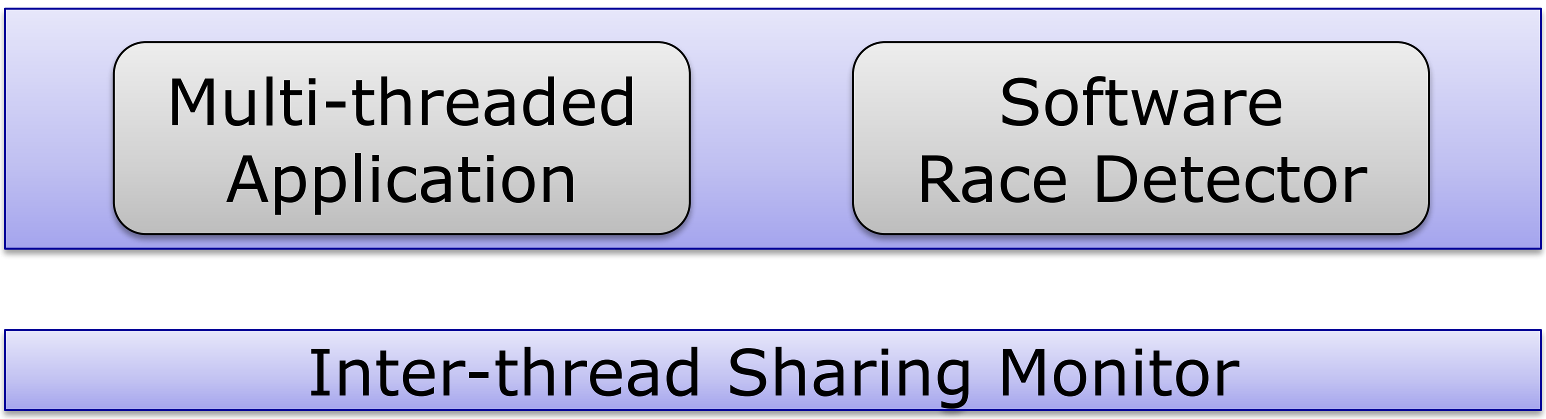


- Fast user-level faults



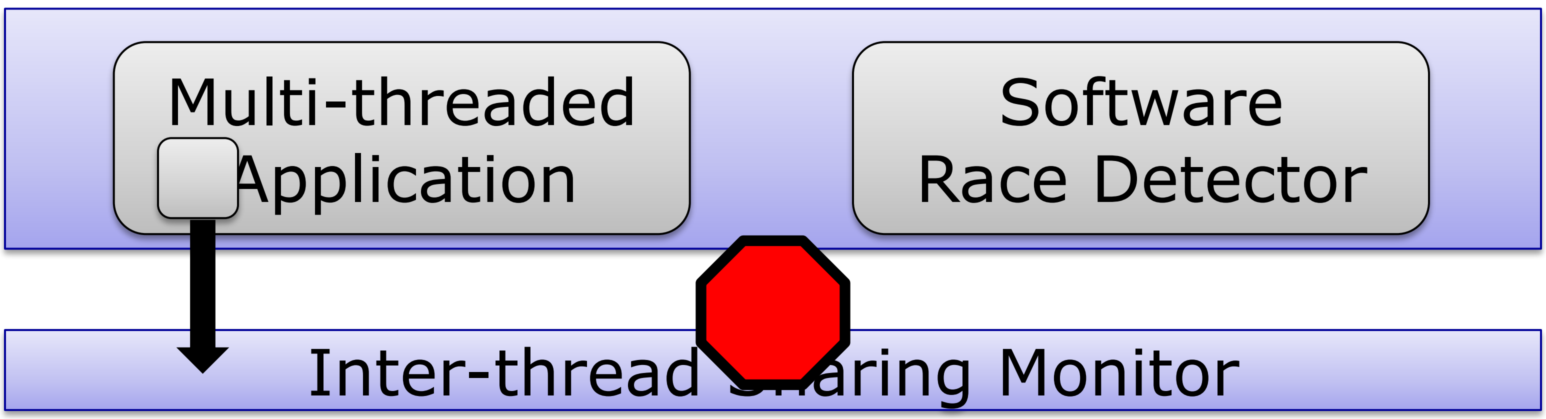
Limitations of Existing Hardware

- Fast faults
  - Solution: Enable detector for long periods of time



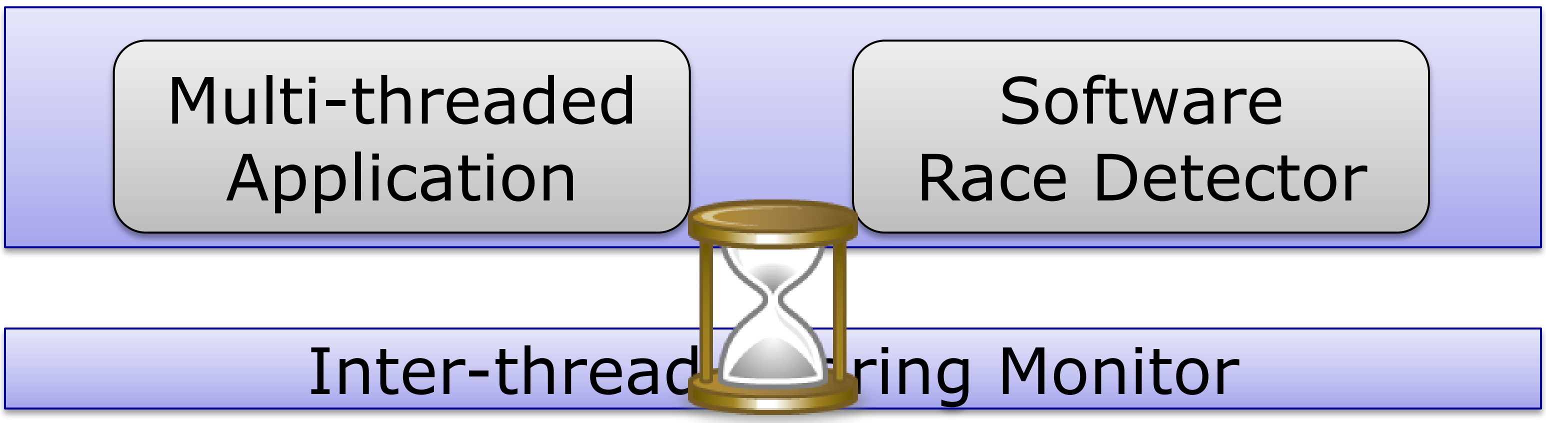
Limitations of Existing Hardware

- Fast faults
  - Solution: Enable detector for long periods of time



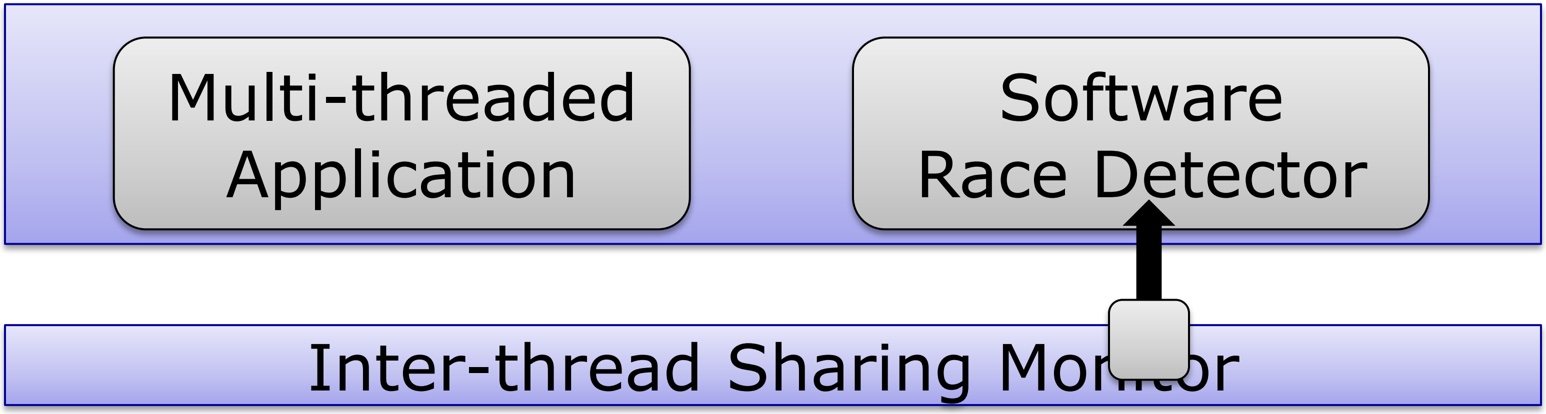
Limitations of Existing Hardware

- Fast faults
  - Solution: Enable detector for long periods of time



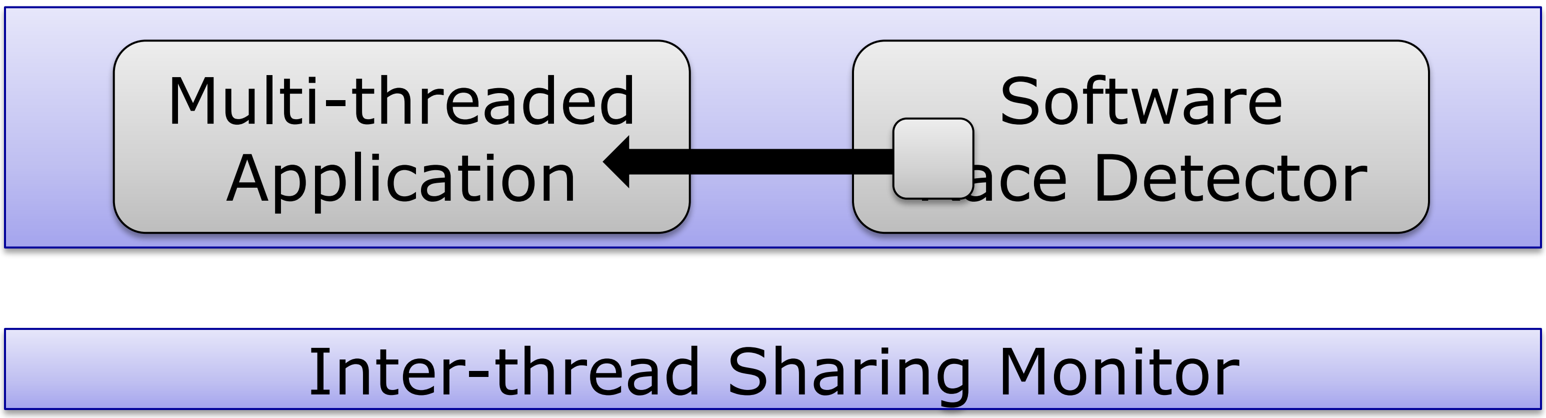
Limitations of Existing Hardware

- Fast faults
  - Solution: Enable detector for long periods of time



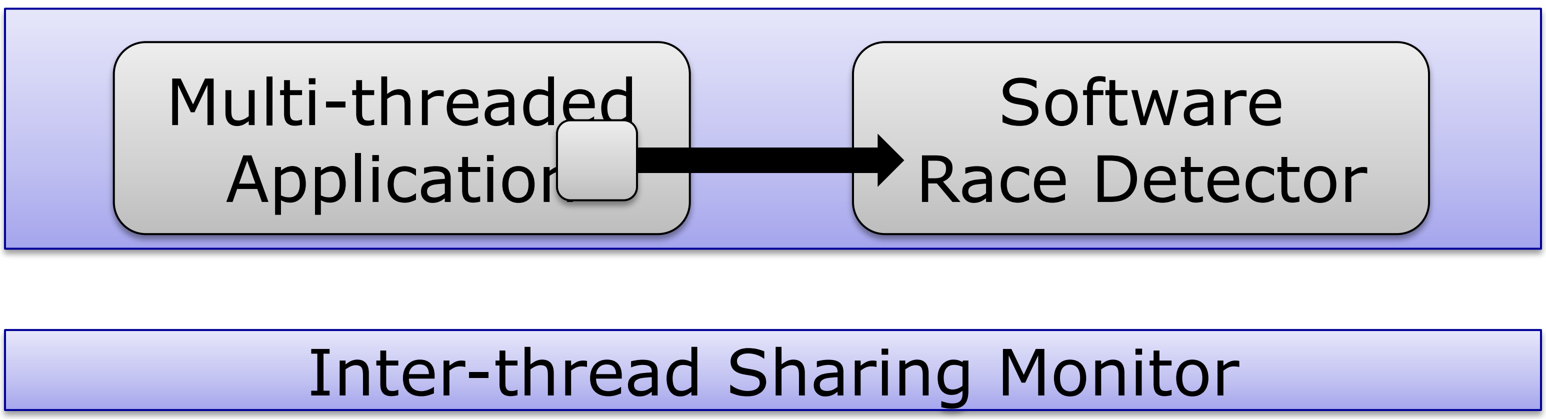
Limitations of Existing Hardware

- Fast faults
  - Solution: Enable detector for long periods of time



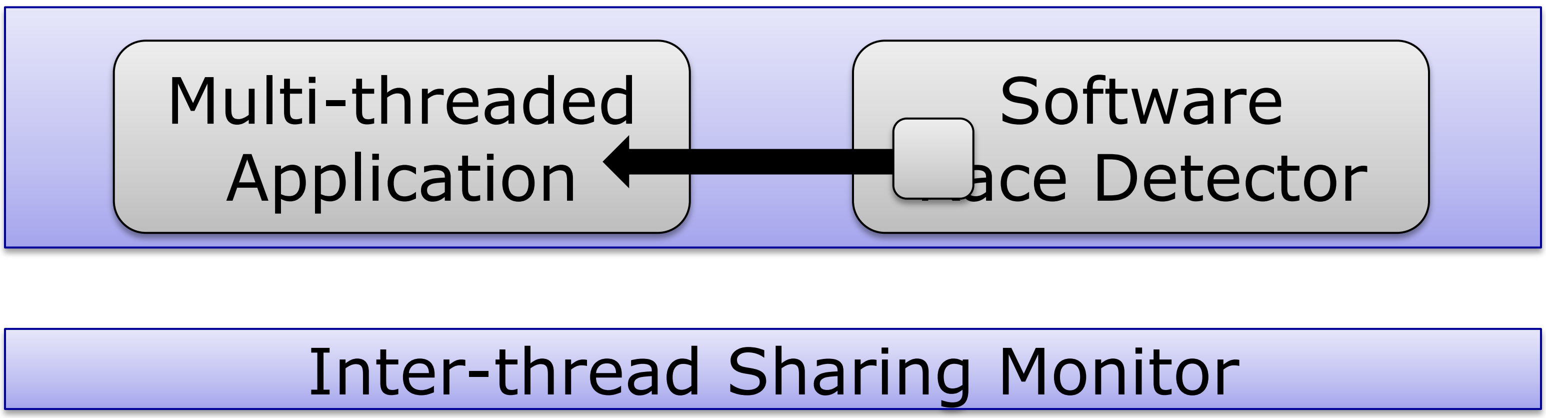
Limitations of Existing Hardware

- Fast faults
  - Solution: Enable detector for long periods of time



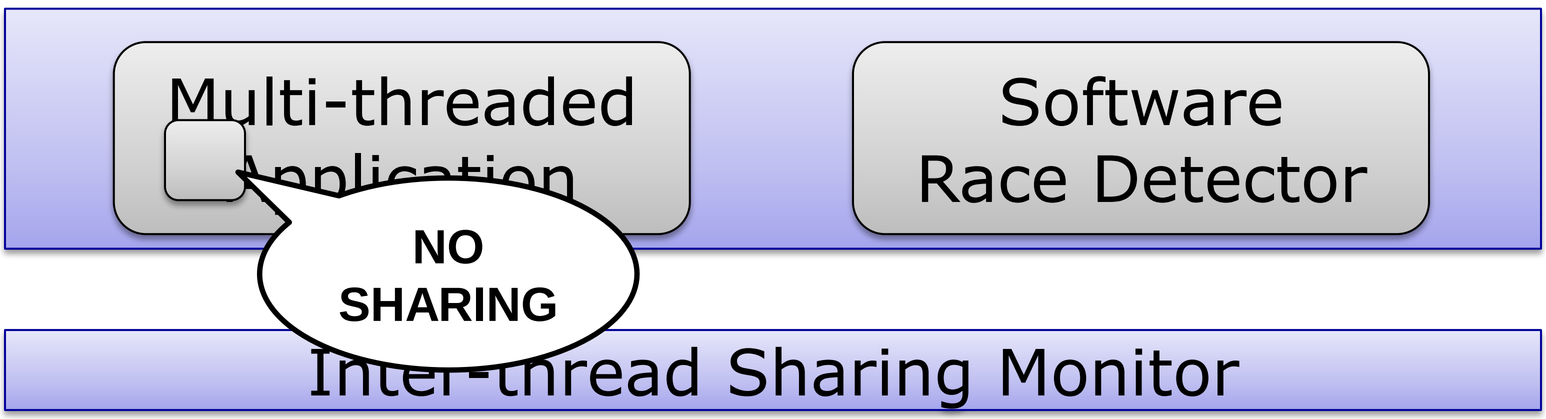
Limitations of Existing Hardware

- Fast faults
  - Solution: Enable detector for long periods of time



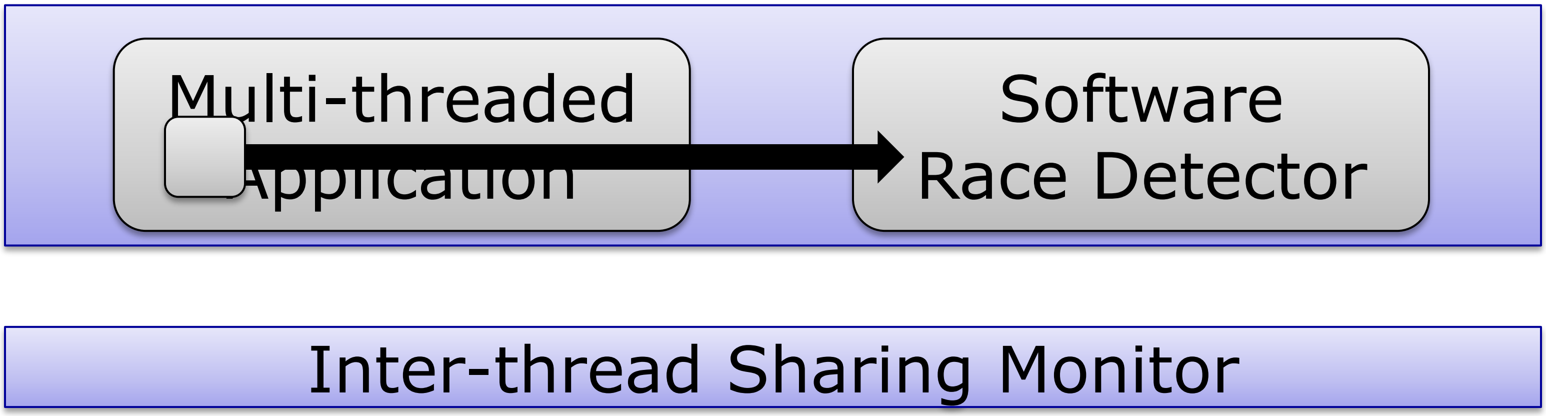
Limitations of Existing Hardware

- Fast faults
  - Solution: Enable detector for long periods of time



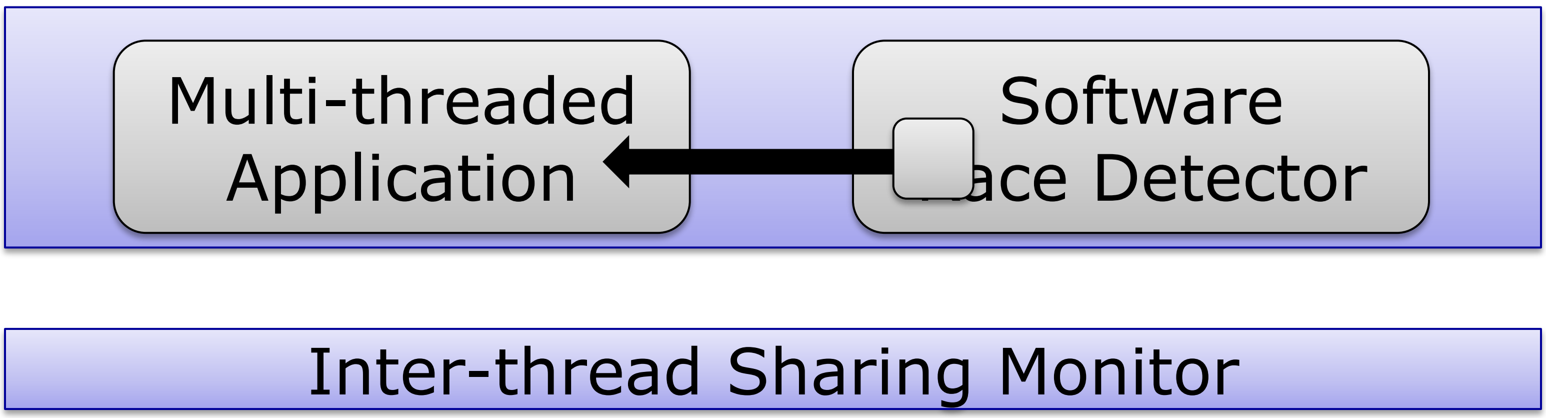
Limitations of Existing Hardware

- Fast faults
  - Solution: Enable detector for long periods of time



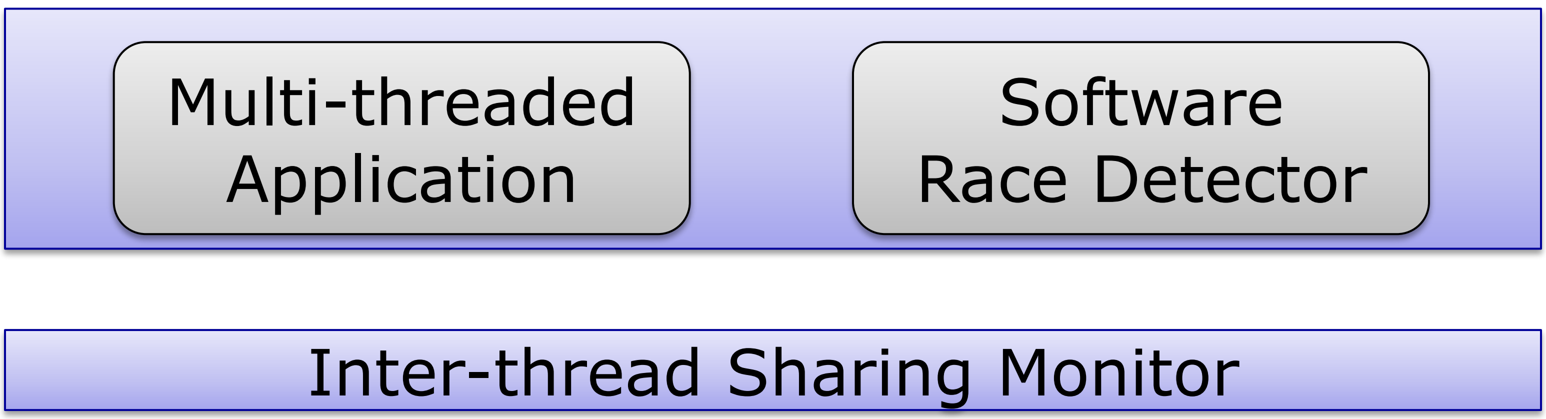
Limitations of Existing Hardware

- Fast faults
  - Solution: Enable detector for long periods of time



Limitations of Existing Hardware

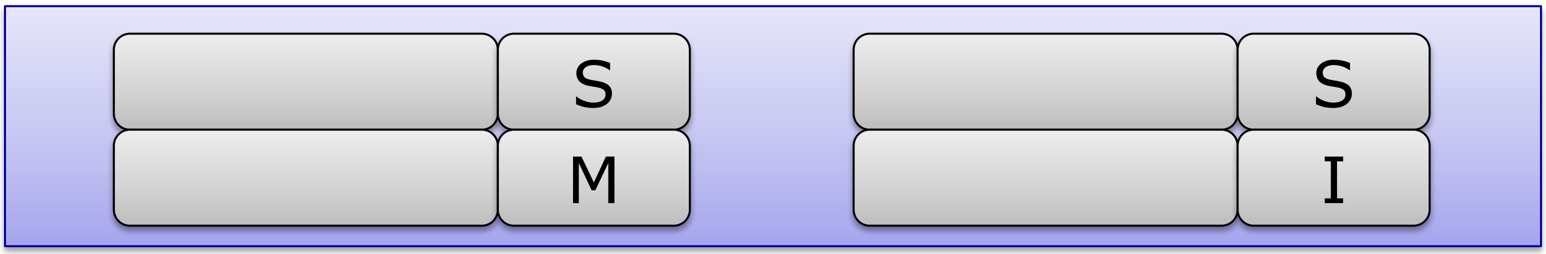
- Fast faults
  - Solution: Enable detector for long periods of time



- Read/write sets
  - Solution:

Technique 2: Hardware Sharing Detector

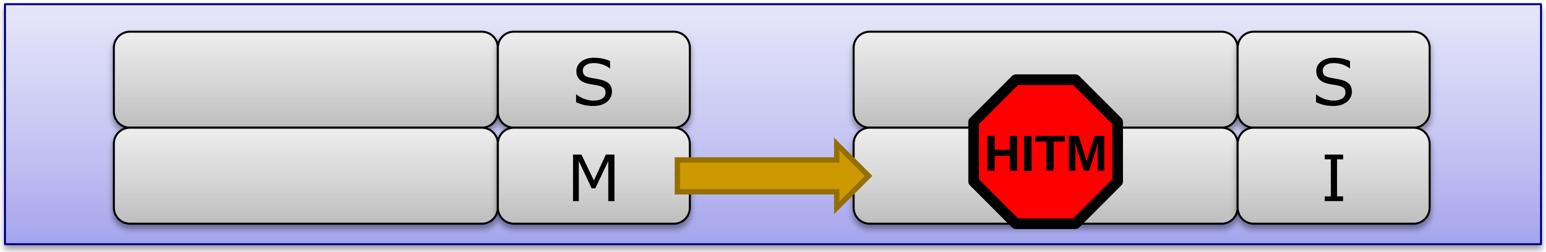
- Hardware Performance Counters
  - Interrupt on cache coherency events
  - Intel's HITM event: W→R Data Sharing



- Limitations of this method:
  - SMT sharing can't be counted
  - Cache eviction
  - Others in paper

Technique 2: Hardware Sharing Detector

- Hardware Performance Counters
  - Interrupt on cache coherency events
  - Intel's HITM event: W→R Data Sharing



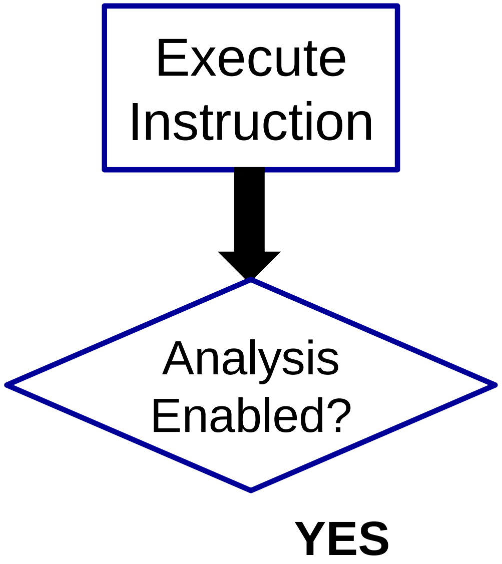
- Limitations of this method:
  - SMT sharing can't be counted
  - Cache eviction
  - Others in paper

Demand-Driven Analysis on Real HW

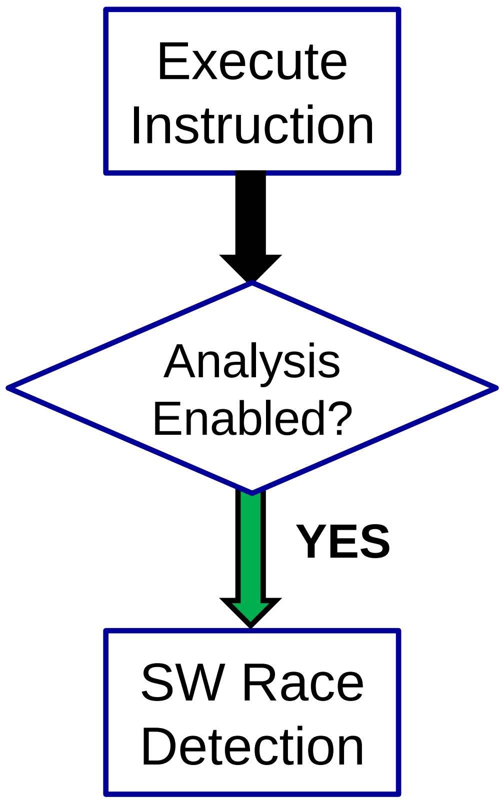
Demand-Driven Analysis on Real HW

Execute Instruction

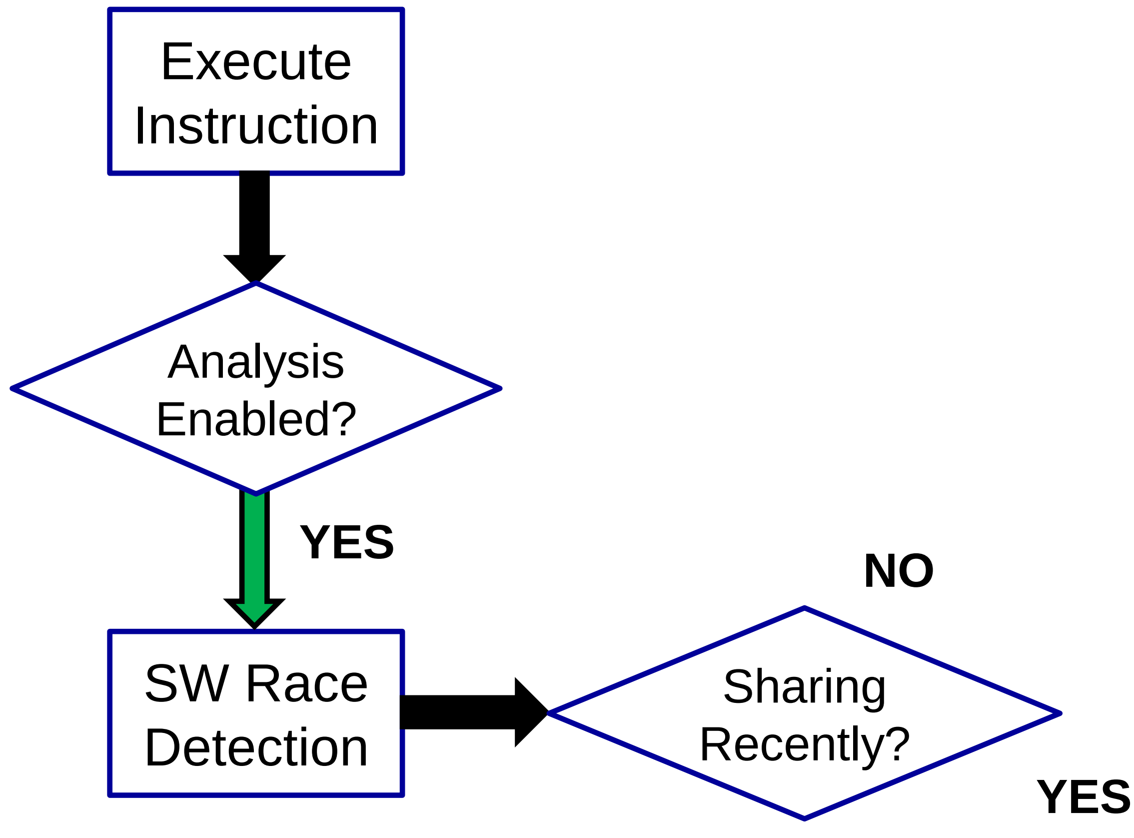
Demand-Driven Analysis on Real HW



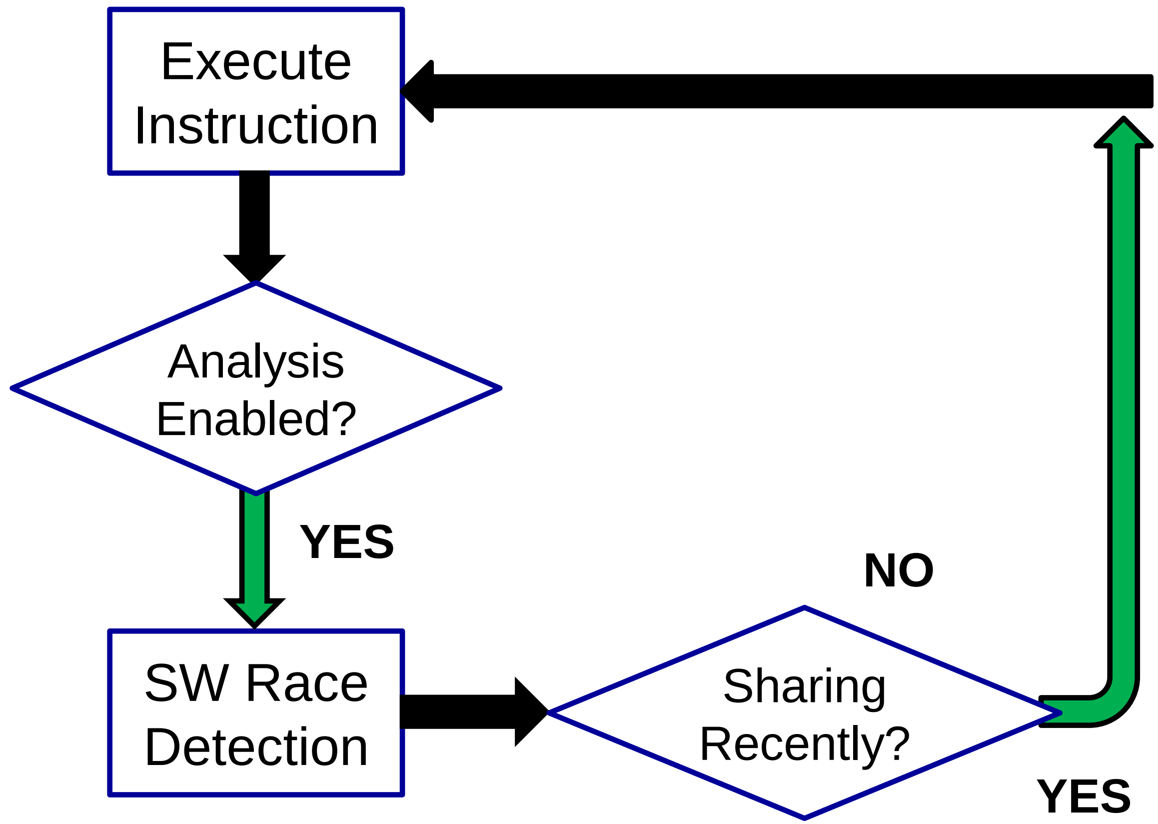
Demand-Driven Analysis on Real HW



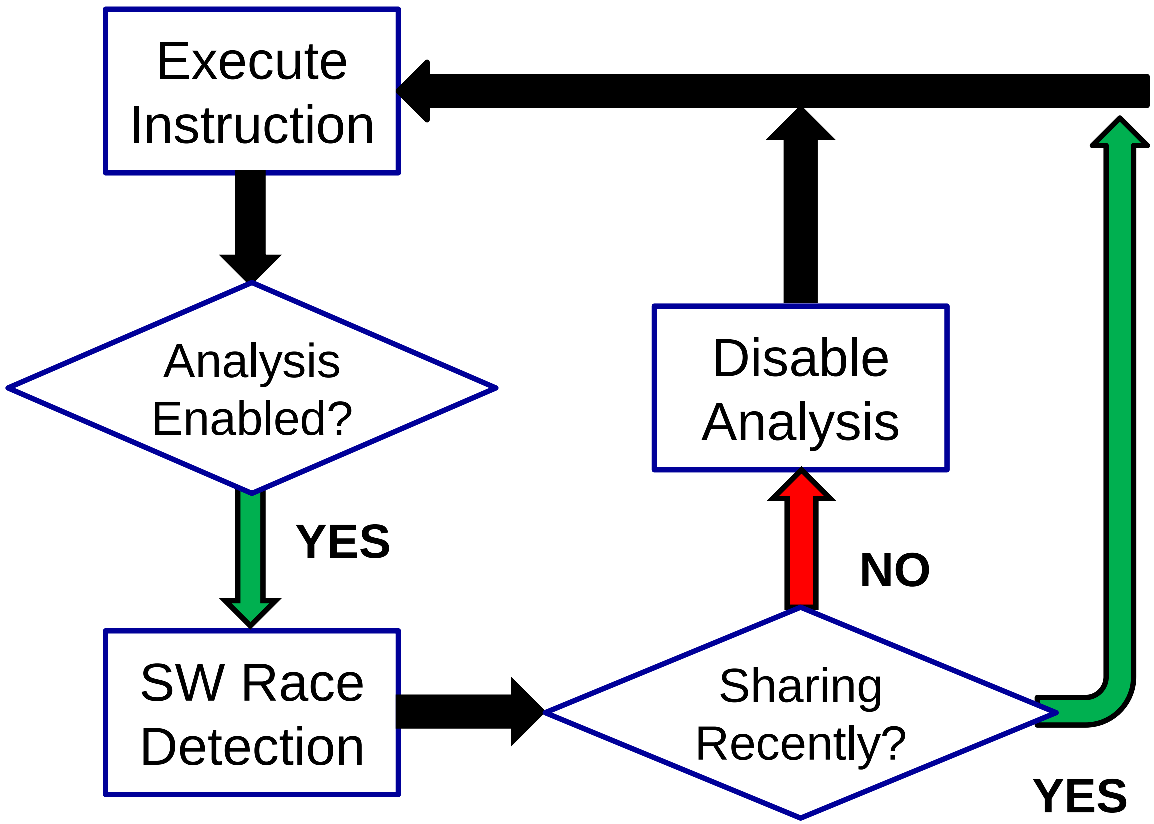
Demand-Driven Analysis on Real HW



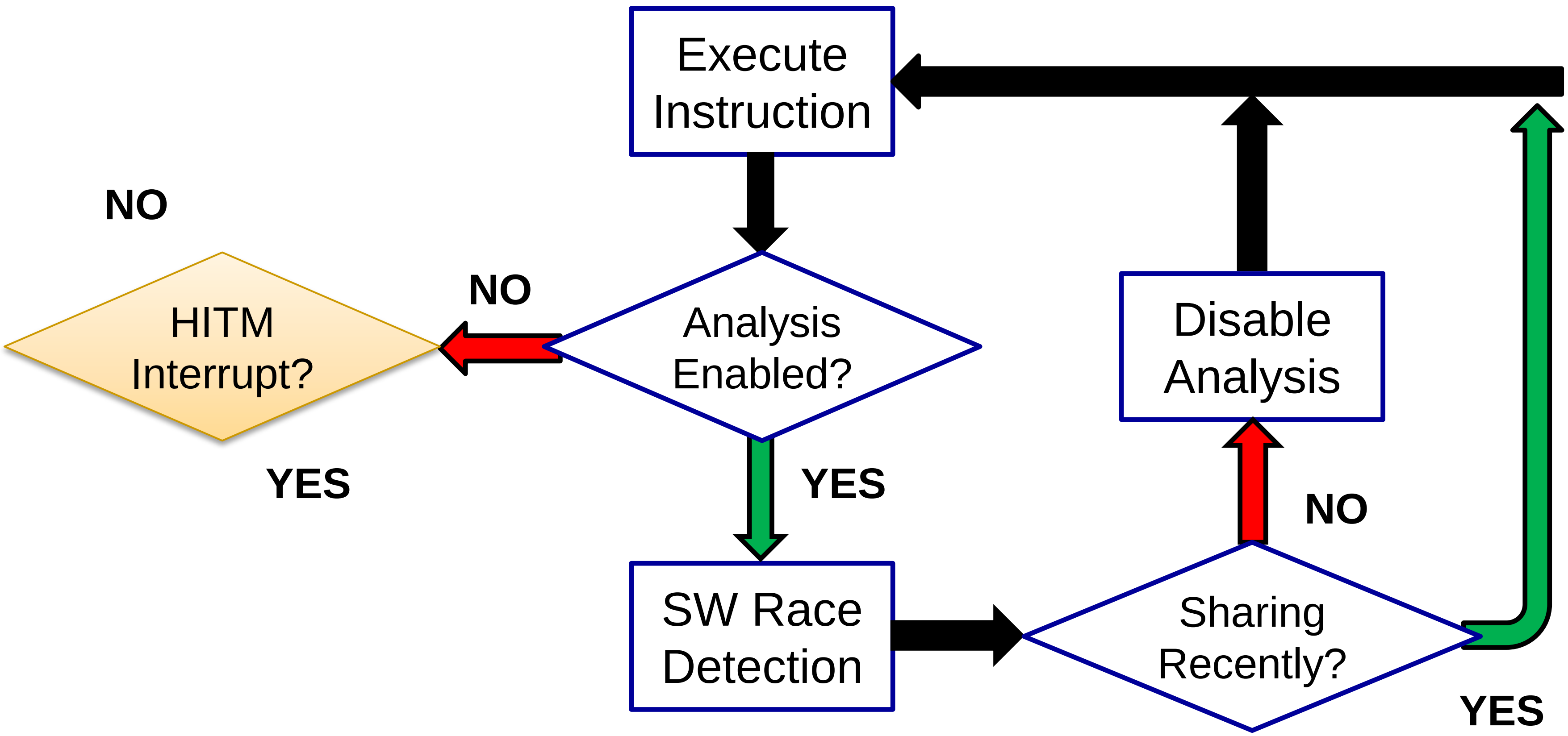
Demand-Driven Analysis on Real HW



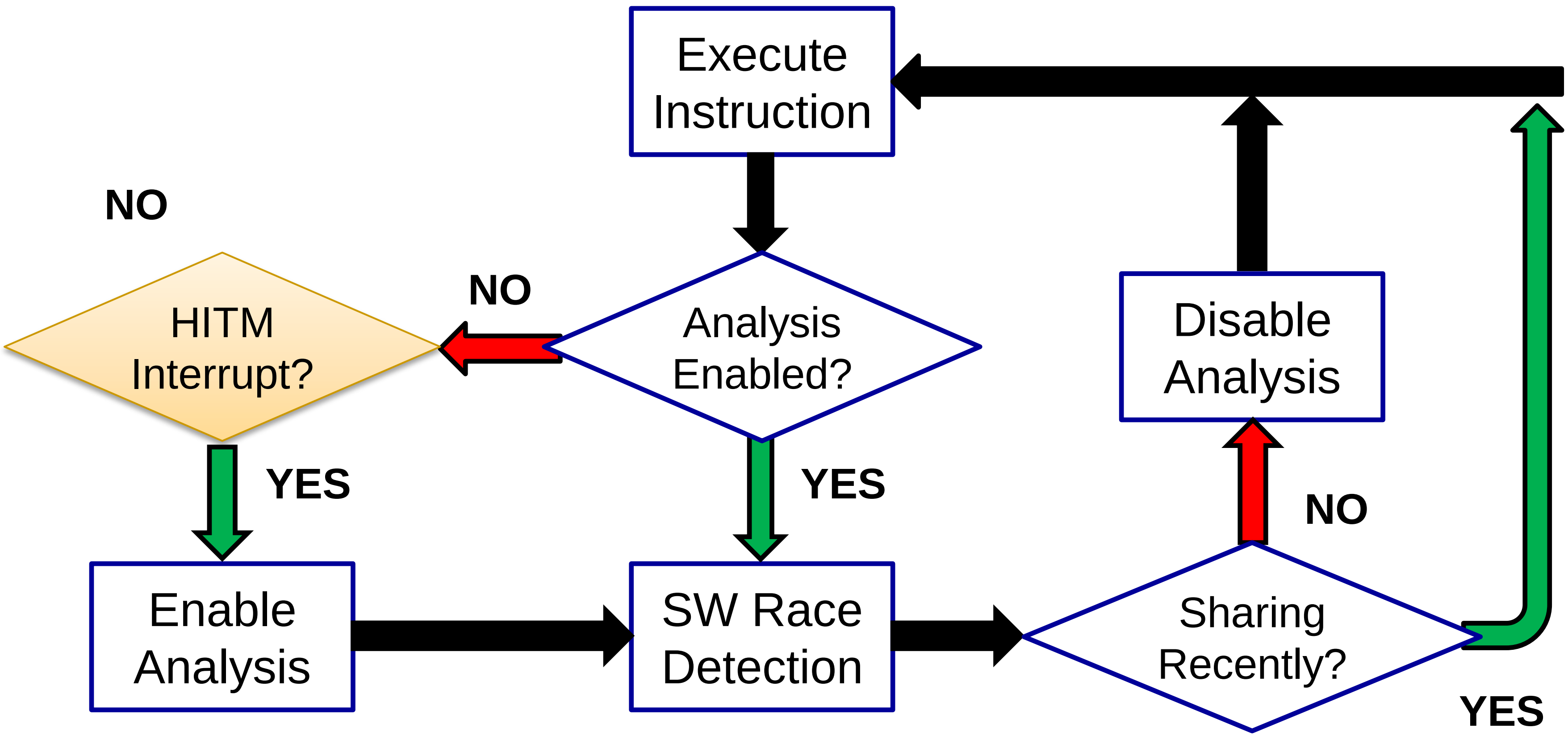
Demand-Driven Analysis on Real HW



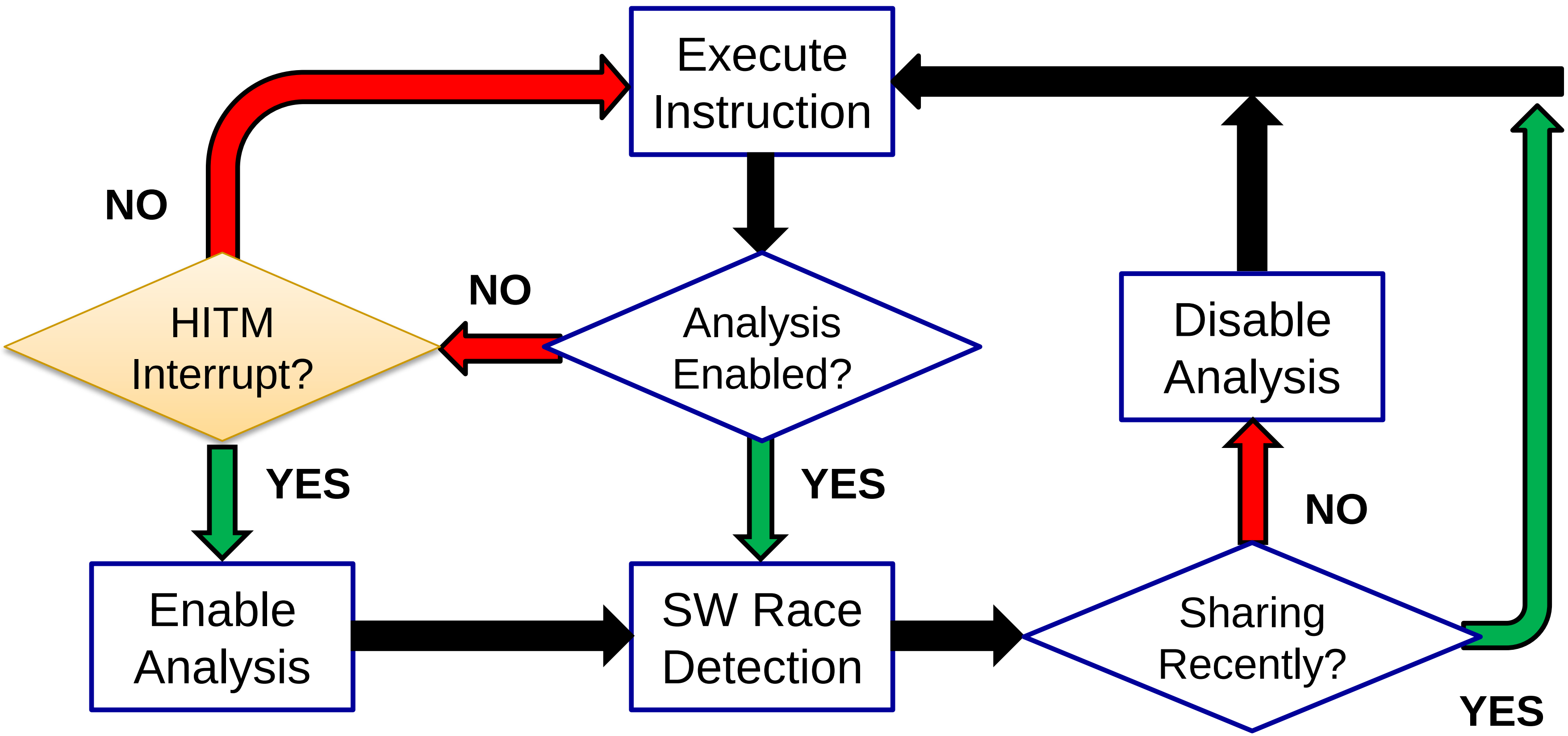
Demand-Driven Analysis on Real HW



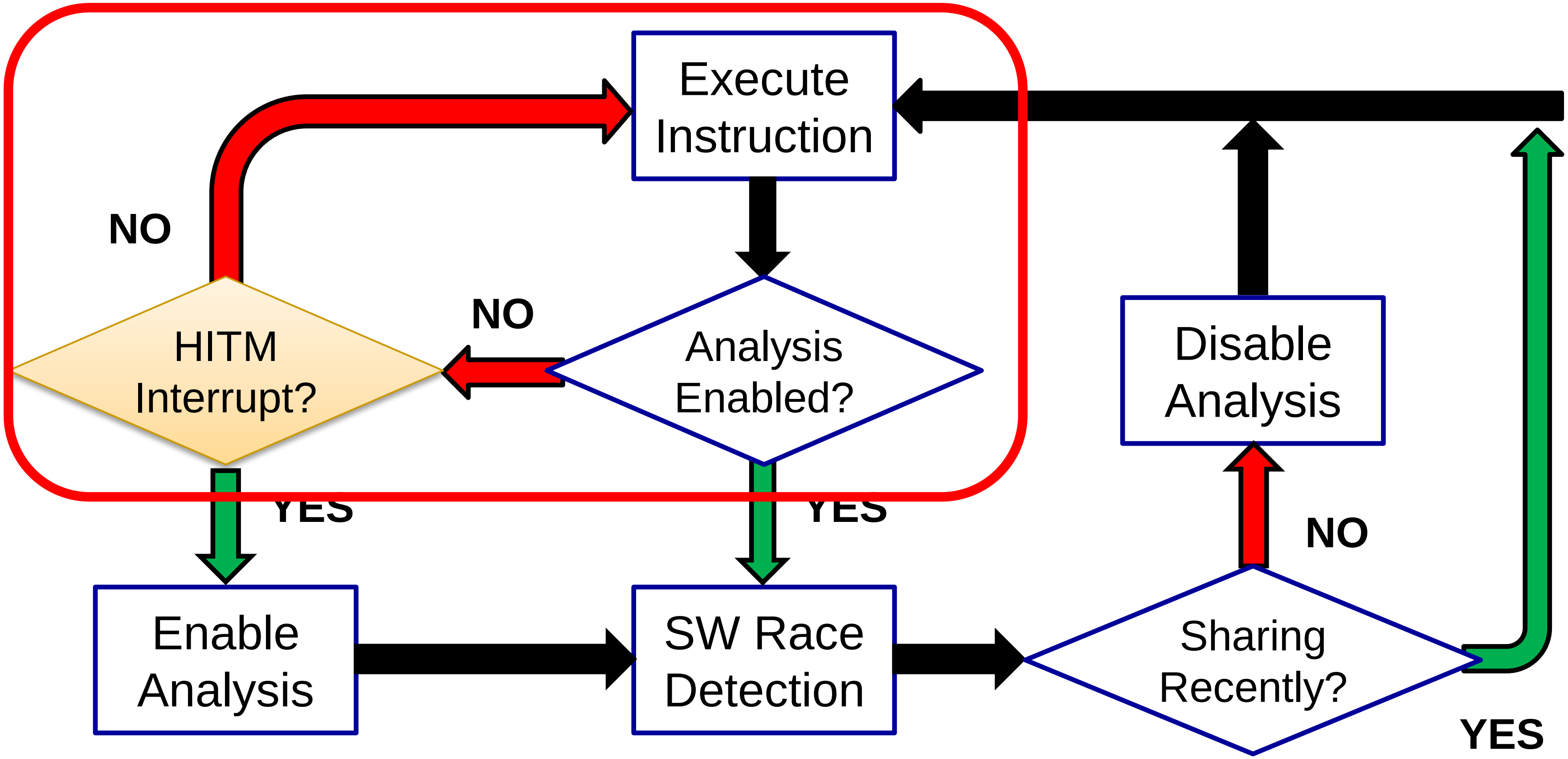
Demand-Driven Analysis on Real HW



Demand-Driven Analysis on Real HW



Demand-Driven Analysis on Real HW



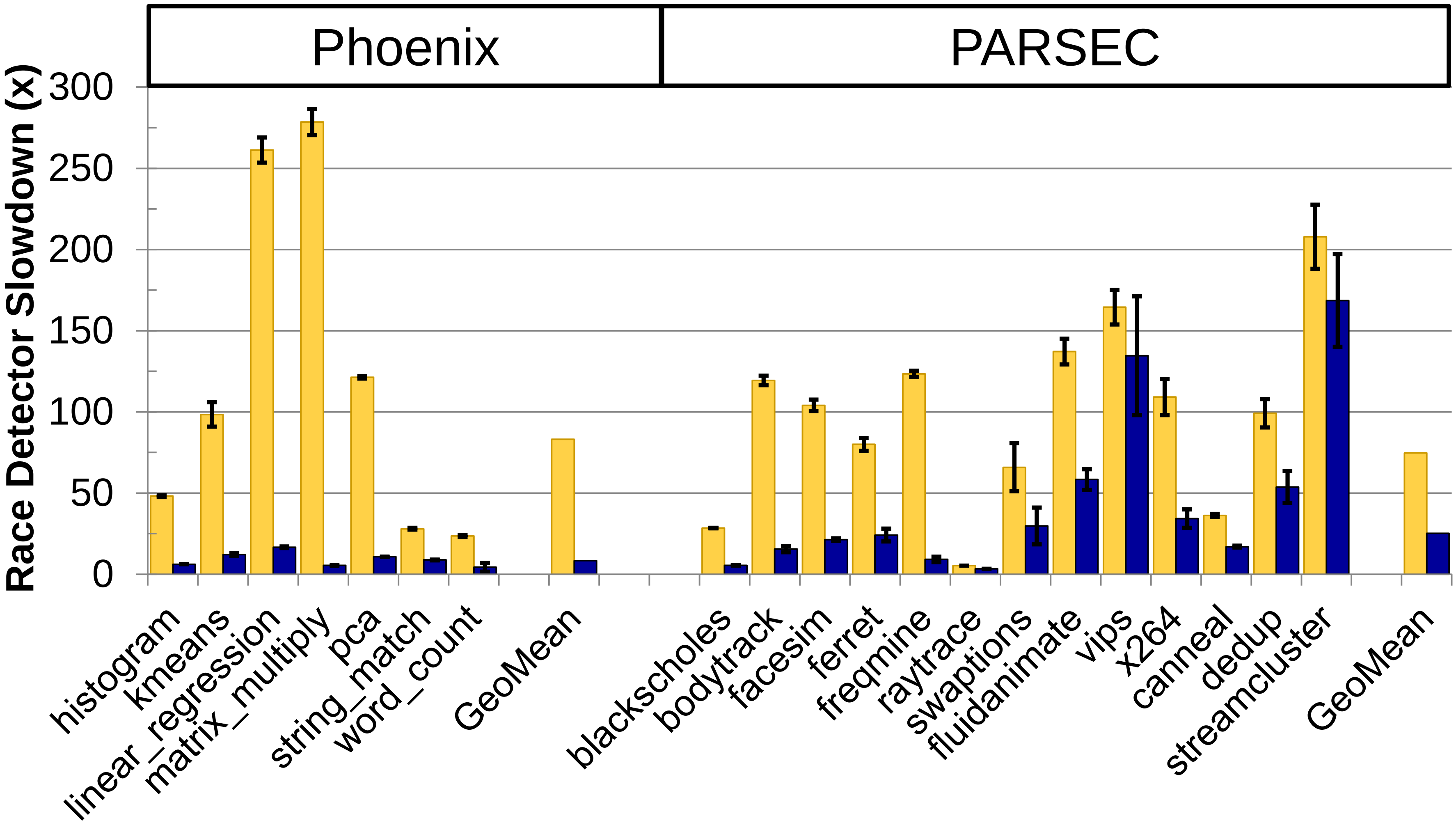
Experimental Evaluation

- Modified Intel Inspector XE Race Detector
- Linux on 4-core Core i7, no Hyper-Threading

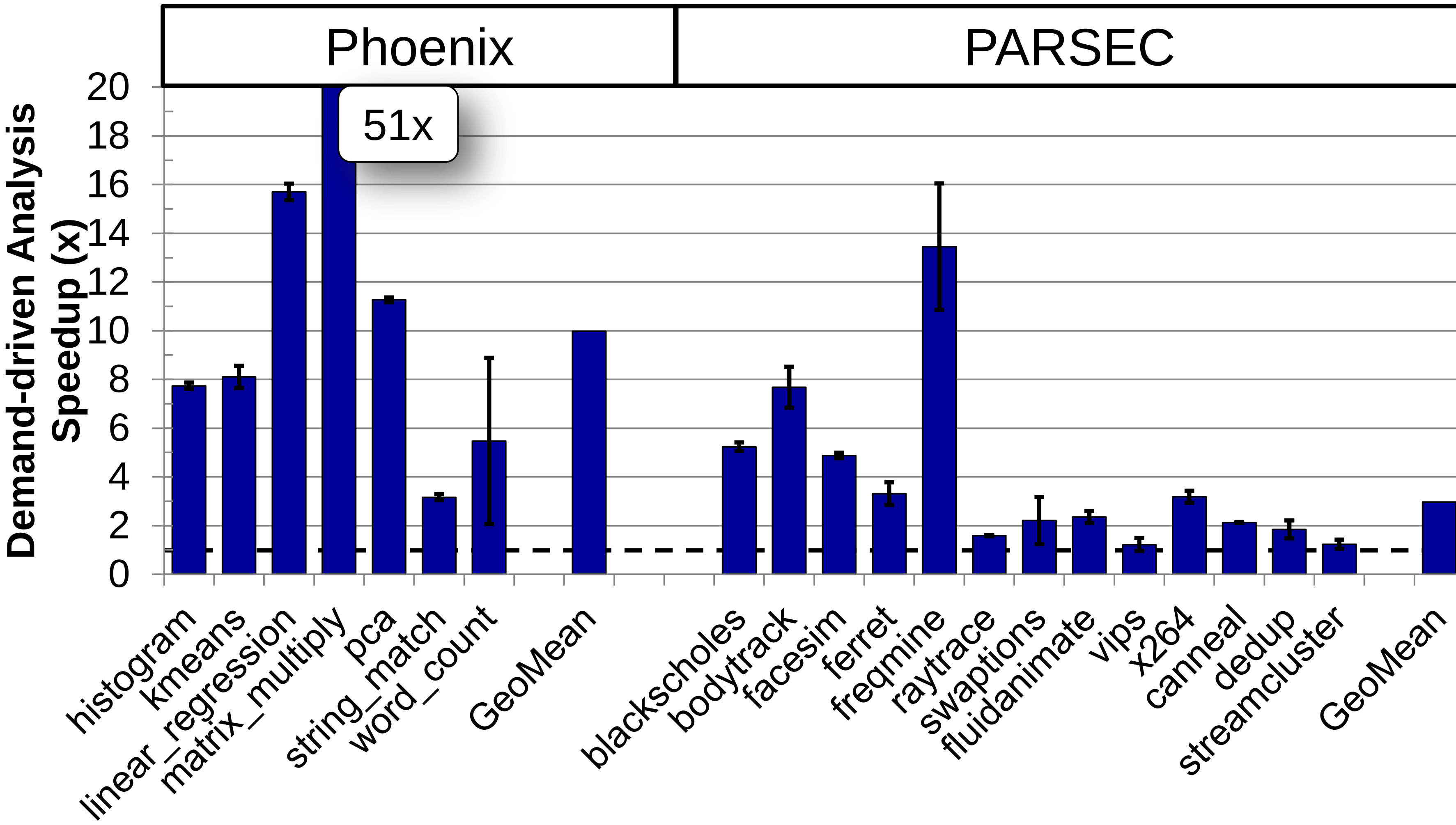
- Performance Tests:
  - Phoenix Suite
  - PARSEC
- Accuracy Tests:
  - Phoenix Suite
  - PARSEC
  - Pre-release version of RADBench

Simulation

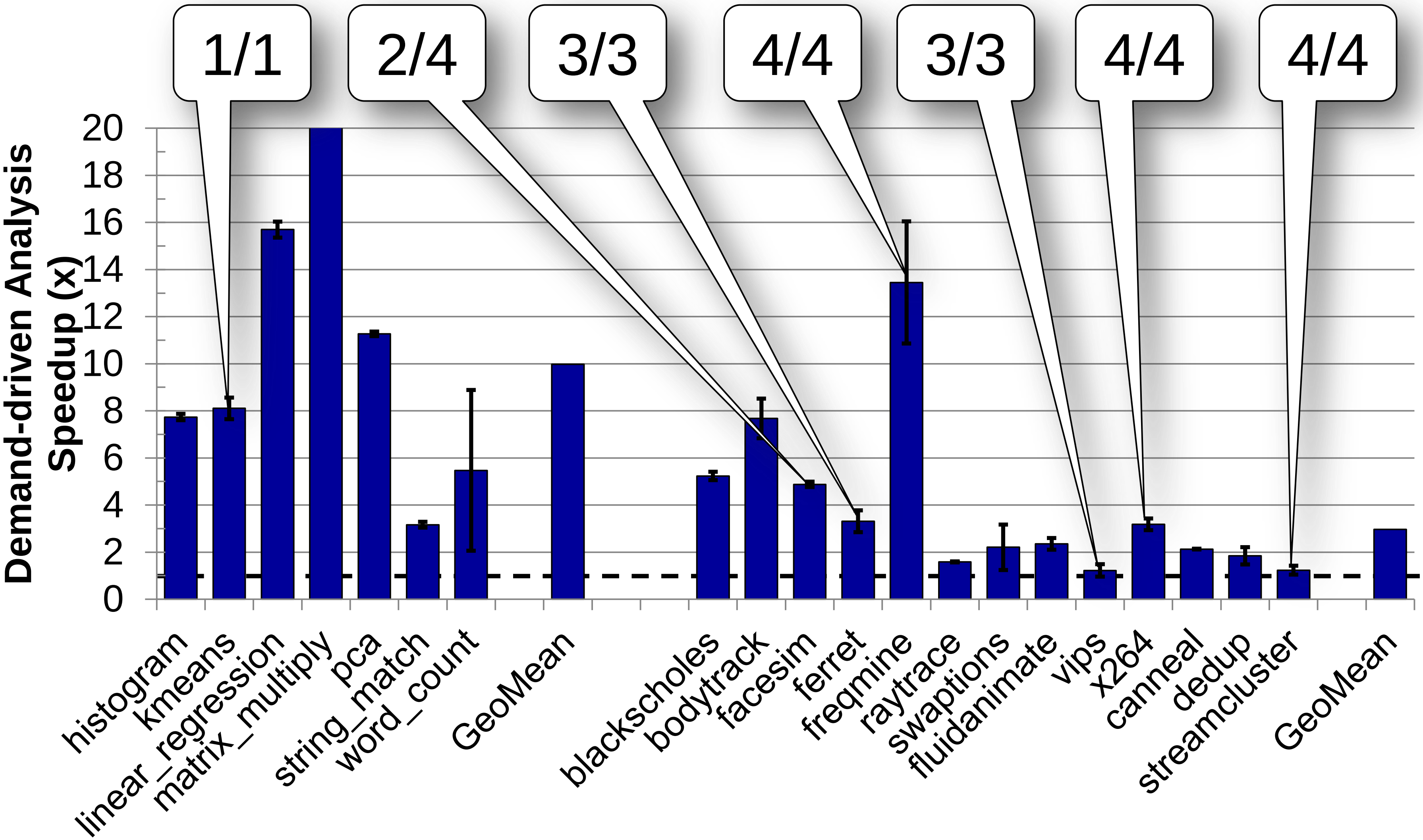
Performance Difference



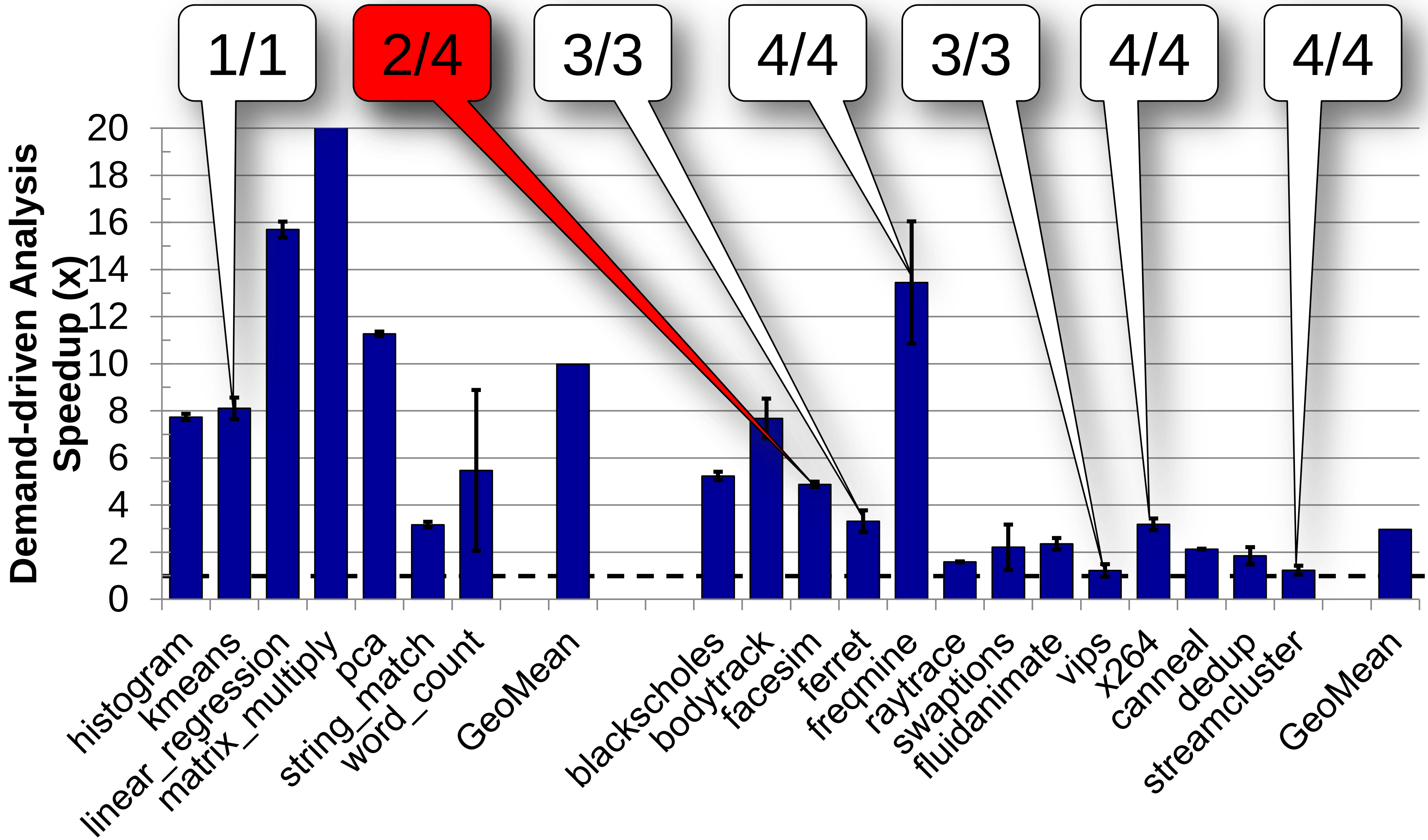
Performance Increases



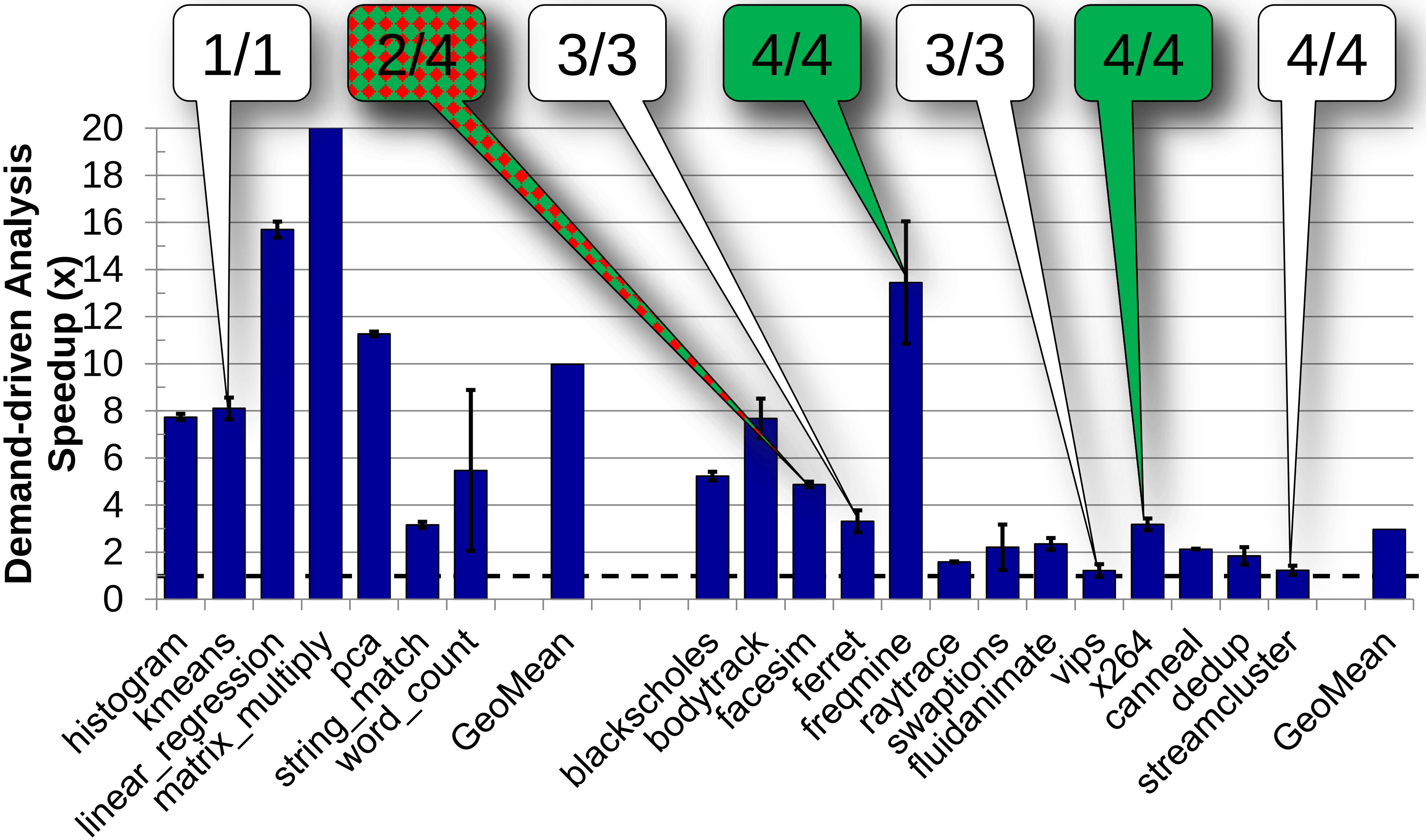
Demand-Driven Analysis Accuracy



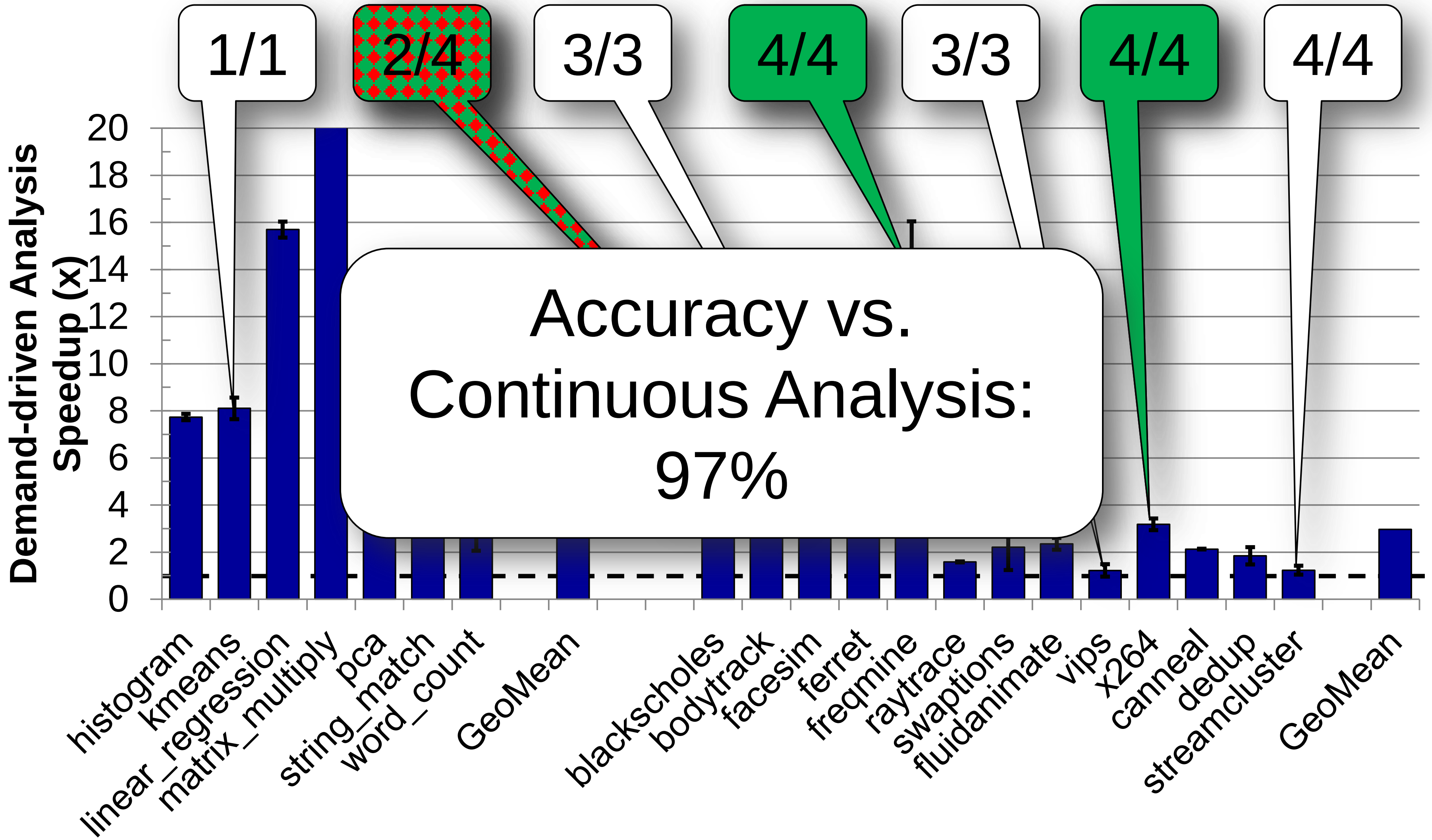
Demand-Driven Analysis Accuracy



Demand-Driven Analysis Accuracy



Demand-Driven Analysis Accuracy



Future Directions

- Better Performance
  - Fast user-level faults
  - Application specific hardware
- More Accuracy
  - Better performance counters
  - Inform SW on cache evictions/misses
- Smooth transition to ideal hardware

- Combine sampling & demand-driven analysis

BACKUP SLIDES

Concurrency Bugs Matter NOW

Thread 1
mylen=small

Thread 2
mylen=large

Nov. 2010 OpenSSL Security Flaw

```
if(ptr == NULL) {
   len=thread_local->mylen;
   ptr=malloc(len);
   memcpy(ptr, data, len);
}
```

Concurrency Bugs Matter NOW

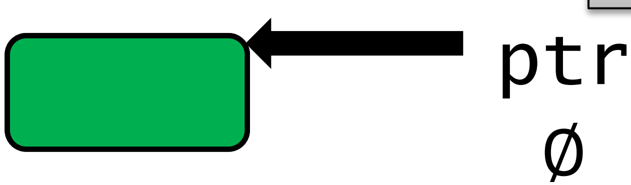
TIME

Thread 1
mylen=small

```
if(ptr==NULL)
len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)
```

Thread 2
mylen=large

```
if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2)
```



Concurrency Bugs Matter <u>NOW</u>

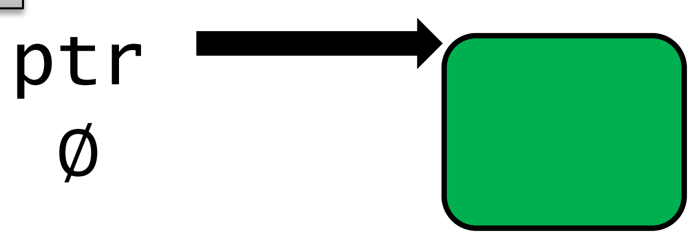
TIME

Thread 2 (mylen=large)

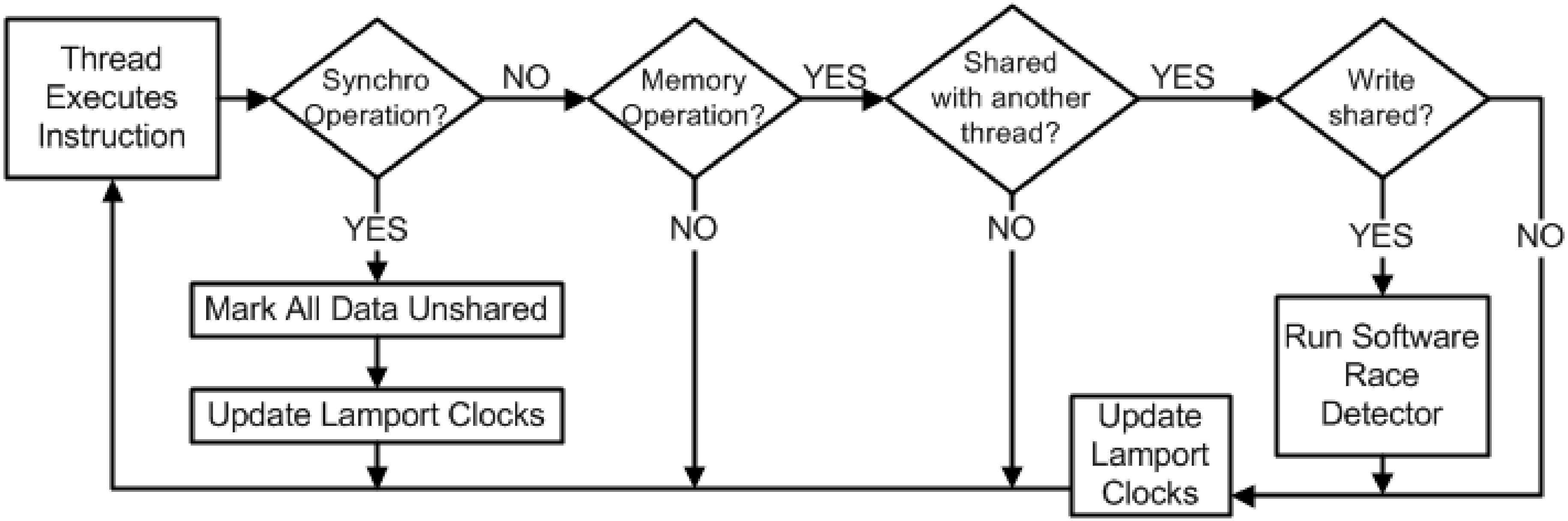
```
if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2)
```

Thread 1 (mylen=small)

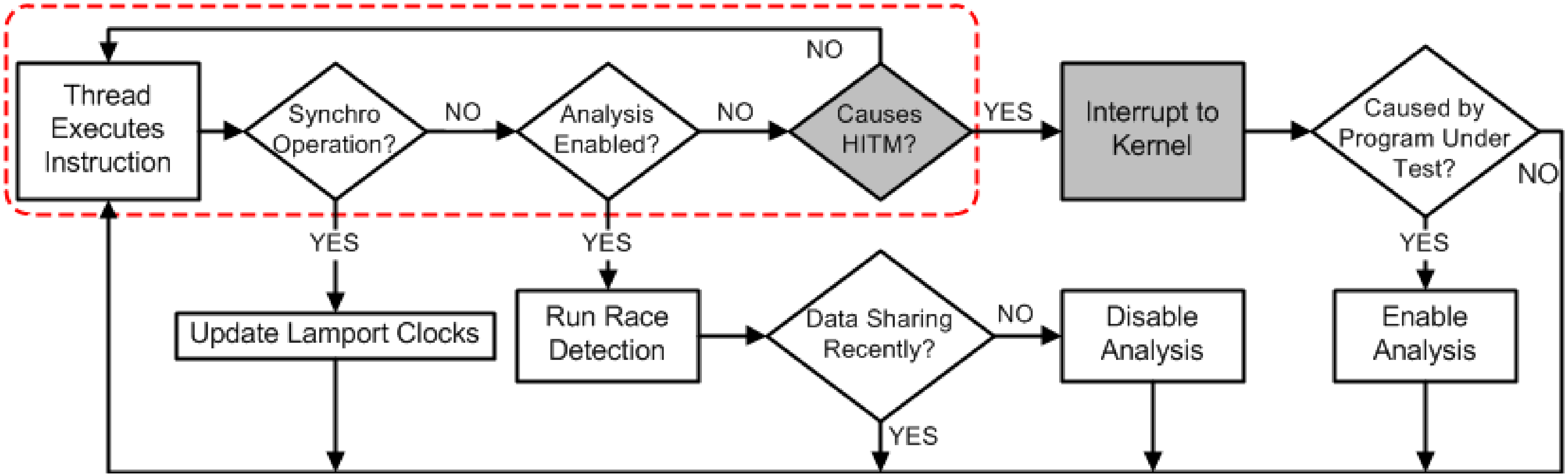
```
if(ptr==NULL)
len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1 ,len1)
```



Demand-Driven Analysis Algorithm



Demand-Driven Analysis on Real HW



Accuracy on Real Hardware

	kmeans	facesim	ferret	freqmine	vips	x264	streamcluster
W→W	1/1 (100%)	0/1 (0%)	-	-	1/1 (100%)	-	1/1 (100%)
R→W	-	0/1 (0%)	2/2 (100%)	2/2 (100%)	1/1 (100%)	3/3 (100%)	1/1 (100%)
W→R	-	2/2 (100%)	1/1 (100%)	2/2 (100%)	1/1 (100%)	3/3/ (100%)	1/1 (100%)

	Spider Monkey-0	Spider Monkey-1	Spider Monkey-2	NSPR-1	Memcached-1	Apache-1
W→W	9/9 (100%)	1/1 (100%)	1/1 (100%)	3/3 (100%)	-	1/1 (100%)
R→W	3/3 (100%)	-	1/1 (100%)	1/1 (100%)	1/1 (100%)	7/7 (100%)
W→R	8/8 (100%)	1/1 (100%)	2/2 (100%)	4/4 (100%)	-	2/2 (100%)