On-Demand Dynamic Software Analysis

Joseph L. Greathouse
Ph.D. Candidate
Advanced Computer Architecture Laboratory
University of Michigan

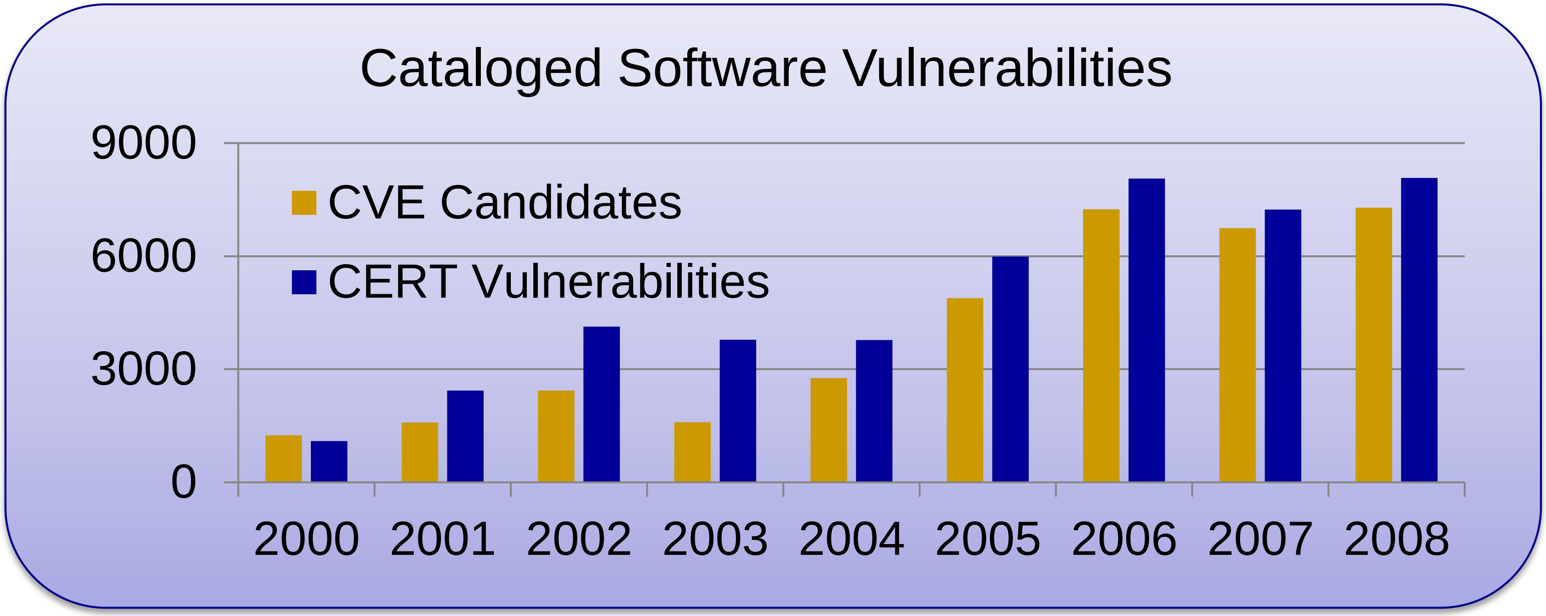
December 12, 2011

Software Errors Abound

- NIST: SW errors cost U.S. ~$60 billion/year as of 2002
- FBI CCS: Security Issues $67 billion/year as of 2005
  - >⅓ from viruses, network intrusion, etc.

Software Errors Abound

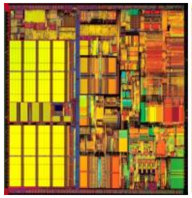
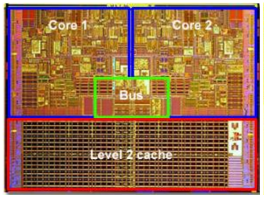
- NIST: SW errors cost U.S. ~$60 billion/year as of 2002
- FBI CCS: Security Issues $67 billion/year as of 2005
  - >⅓ from viruses, network intrusion, etc.

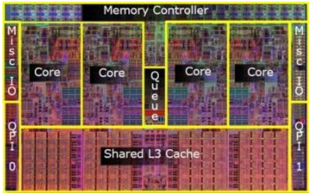


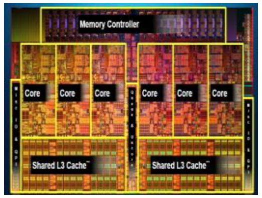
Software Errors Abound

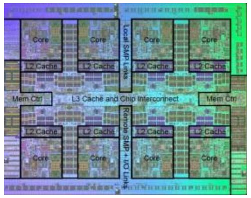
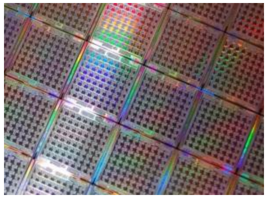
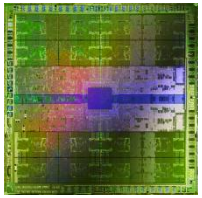
- NIST: SW errors cost U.S. ~$60 billion/year as of 2002
- FBI CCS: Security Issues $67 billion/year as of 2005
  - >⅓ from viruses, network intrusion, etc.

Hardware Plays a Role

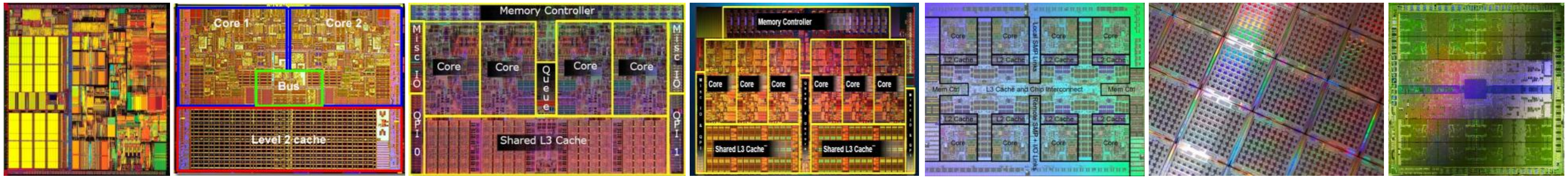







Hardware Plays a Role



In spite of proposed solutions

Hardware Data Race Recording

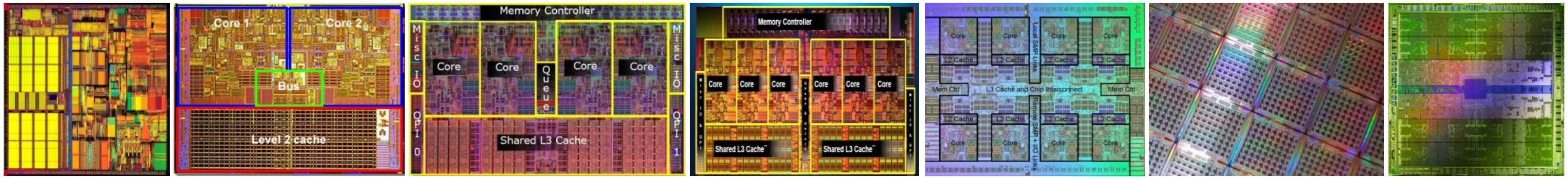
Bulk Memory Commits

Deterministic Execution/Replay

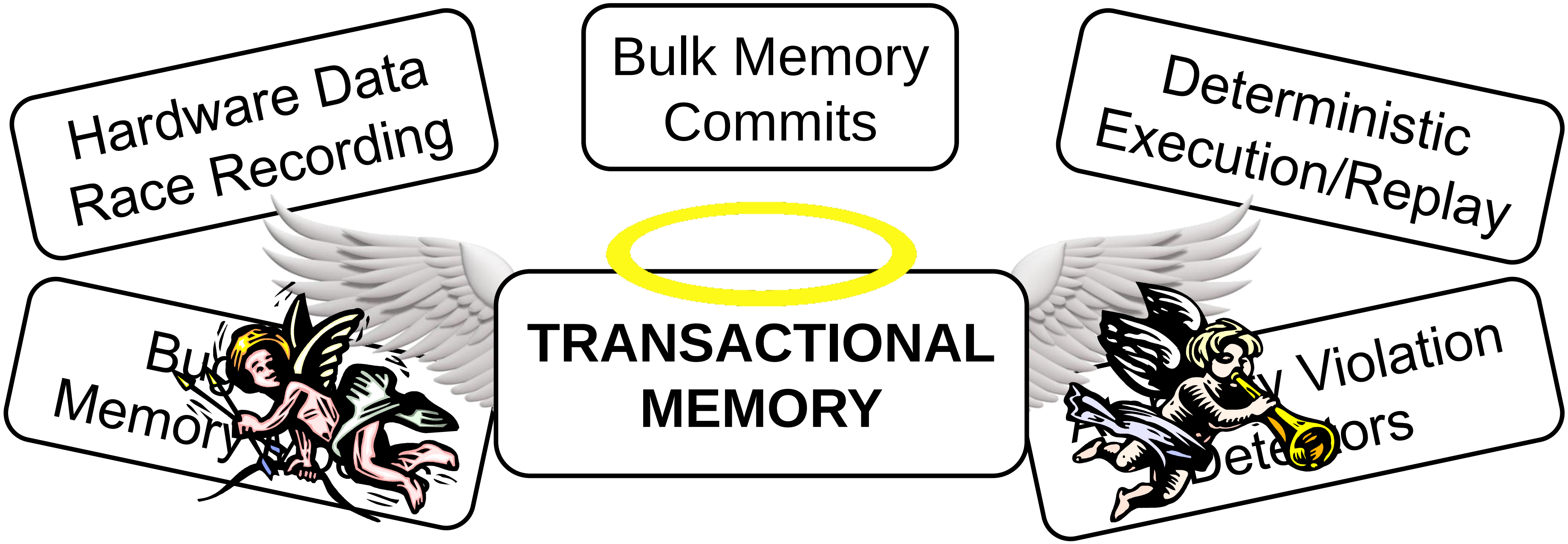
Bug-Free Memory Models

Atomicity Violation Detectors

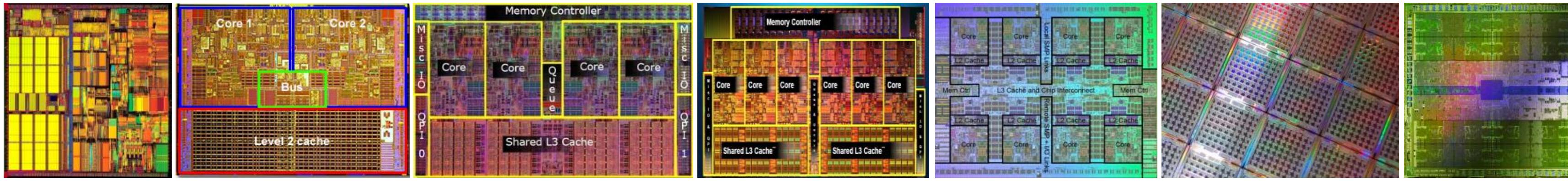
Hardware Plays a Role



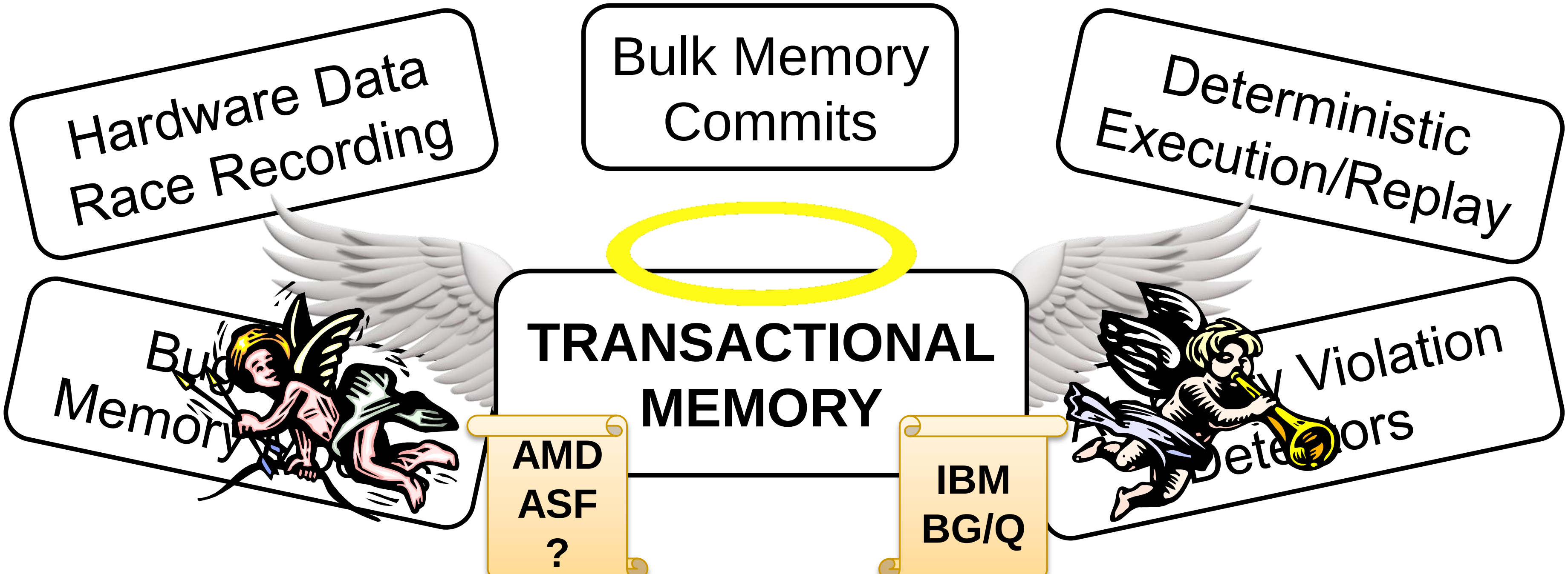
In spite of proposed solutions



Hardware Plays a Role

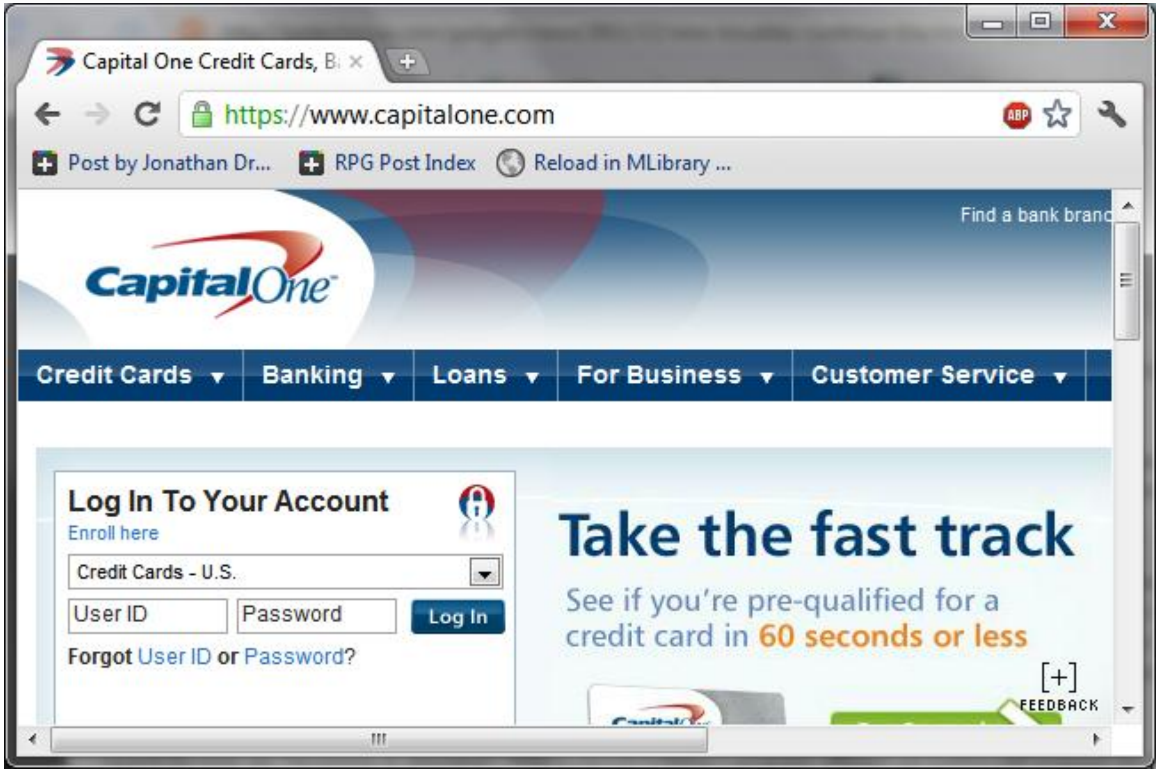


In spite of proposed solutions



Example of a Modern Bug

Nov. 2010 OpenSSL Security Flaw

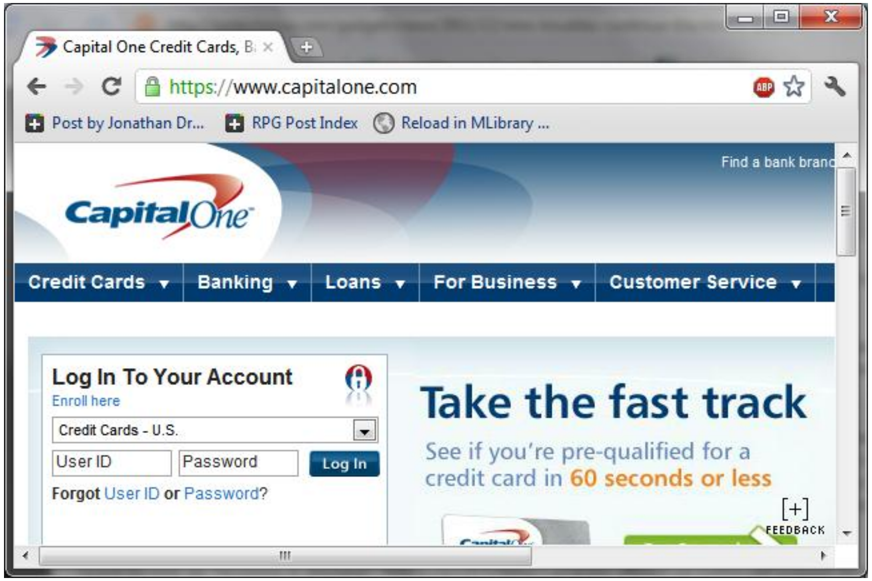


Example of a Modern Bug

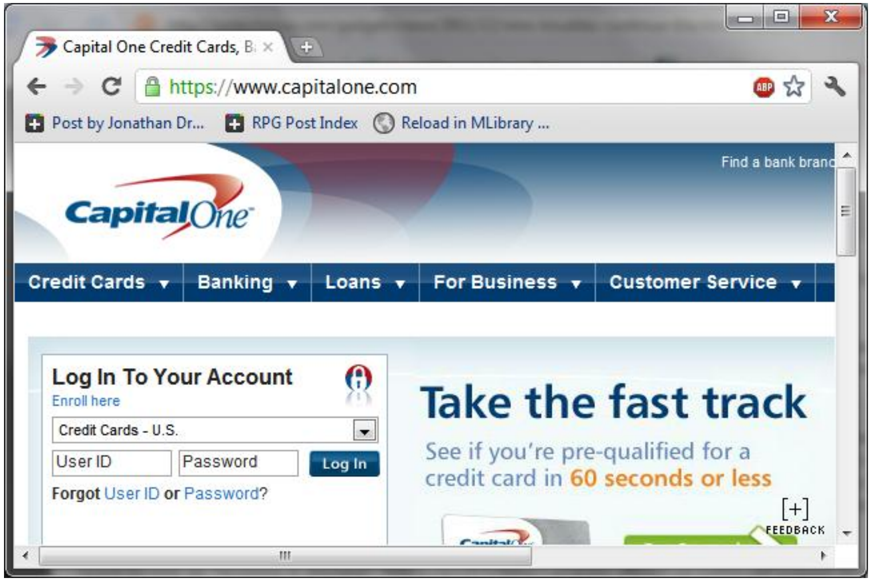
```
if(ptr == NULL) {
   len=thread_local->mylen;
   ptr=malloc(len);
   memcpy(ptr, data, len);
}
```

Example of a Modern Bug

Thread 1
mylen=small



Thread 2
mylen=large



`ptr`

∅

Example of a Modern Bug

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
∅
```

Example of a Modern Bug

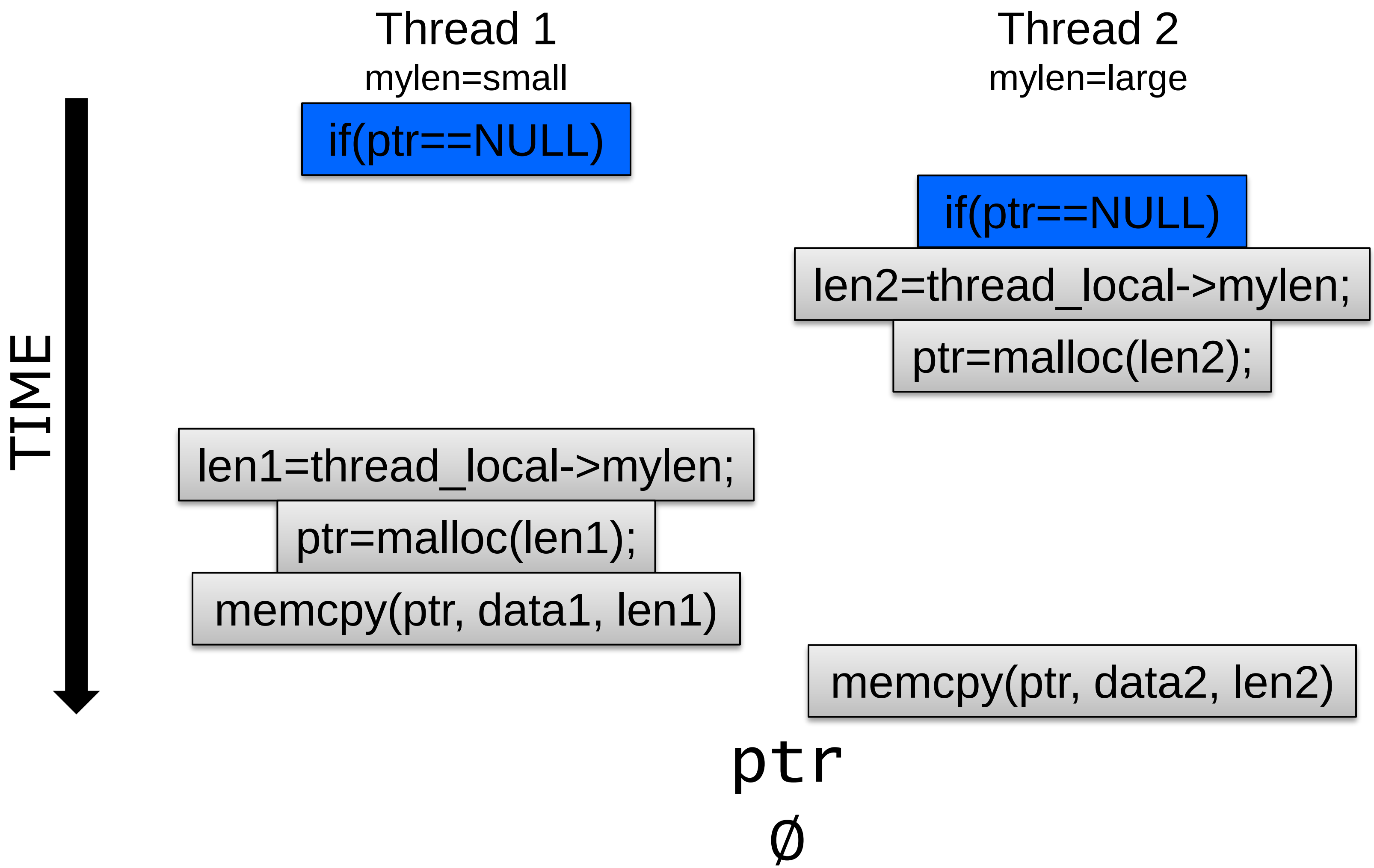
TIME

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`

∅

Example of a Modern Bug



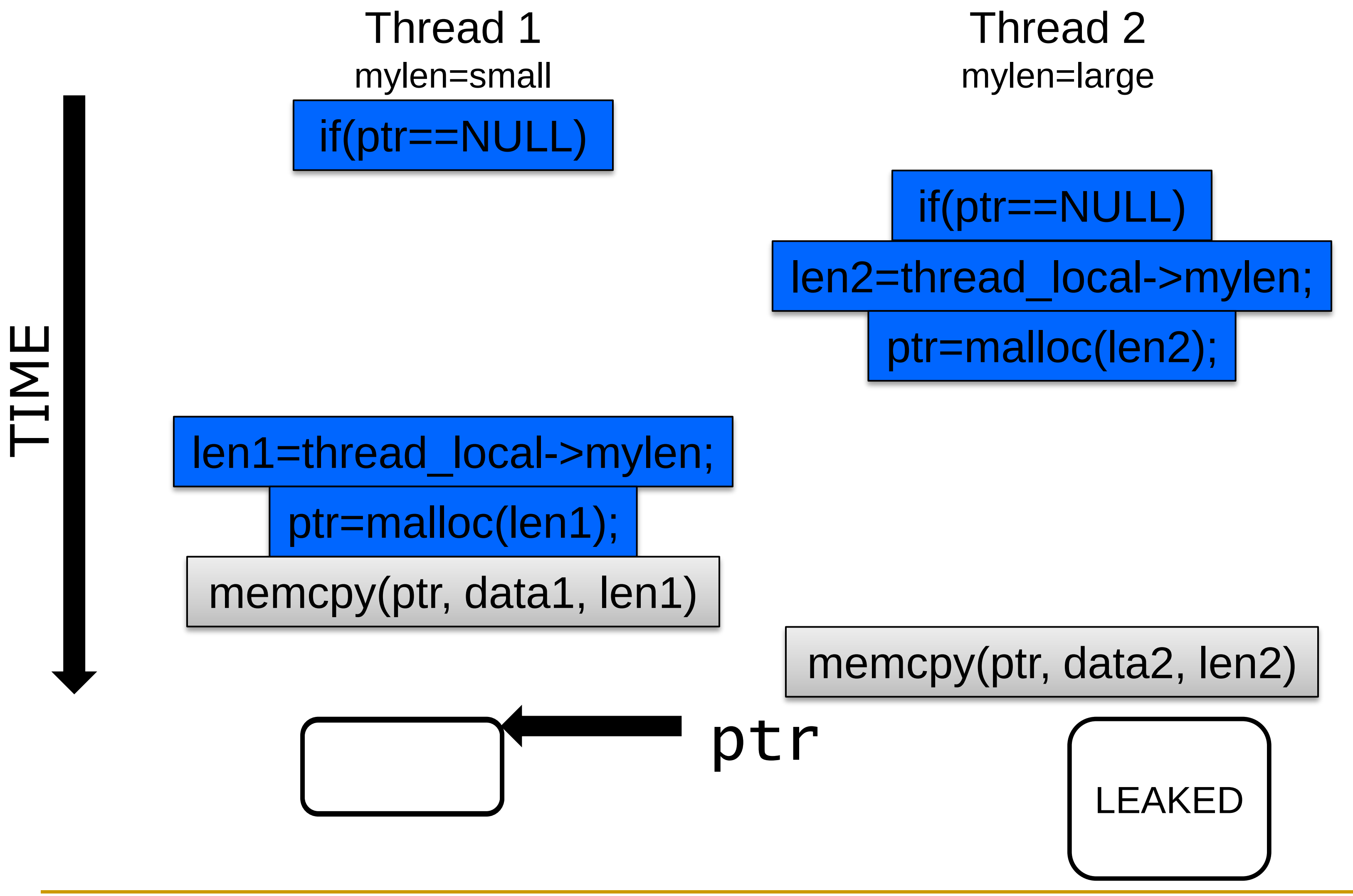
Example of a Modern Bug

TIME

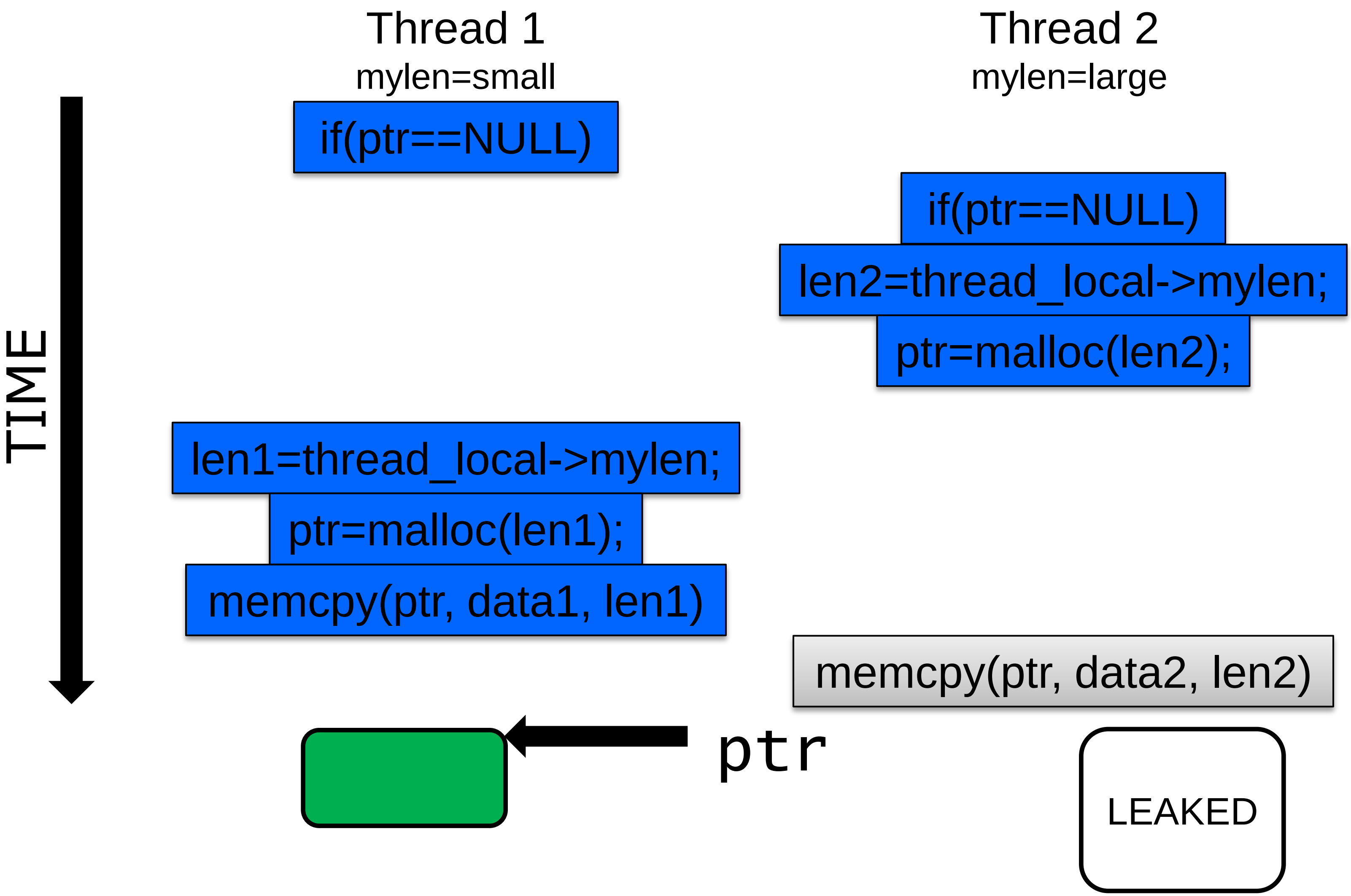
Thread 1 mylen=small	Thread 2 mylen=large
`if(ptr==NULL)`	
	`if(ptr==NULL)`
	`len2=thread_local->mylen;`
	`ptr=malloc(len2);`
`len1=thread_local->mylen;`	
`ptr=malloc(len1);`	
`memcpy(ptr, data1, len1)`	
	`memcpy(ptr, data2, len2)`

ptr

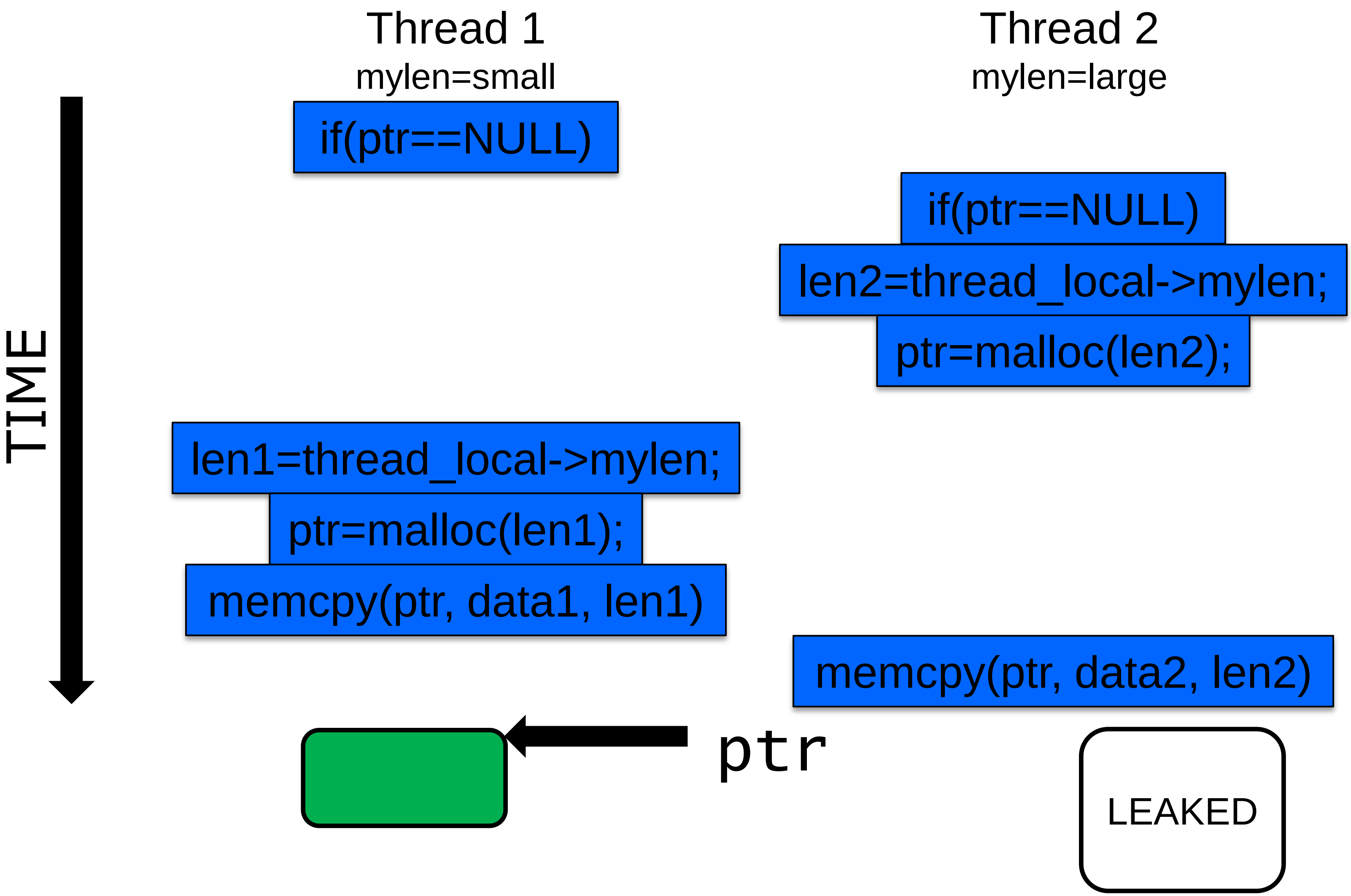
Example of a Modern Bug



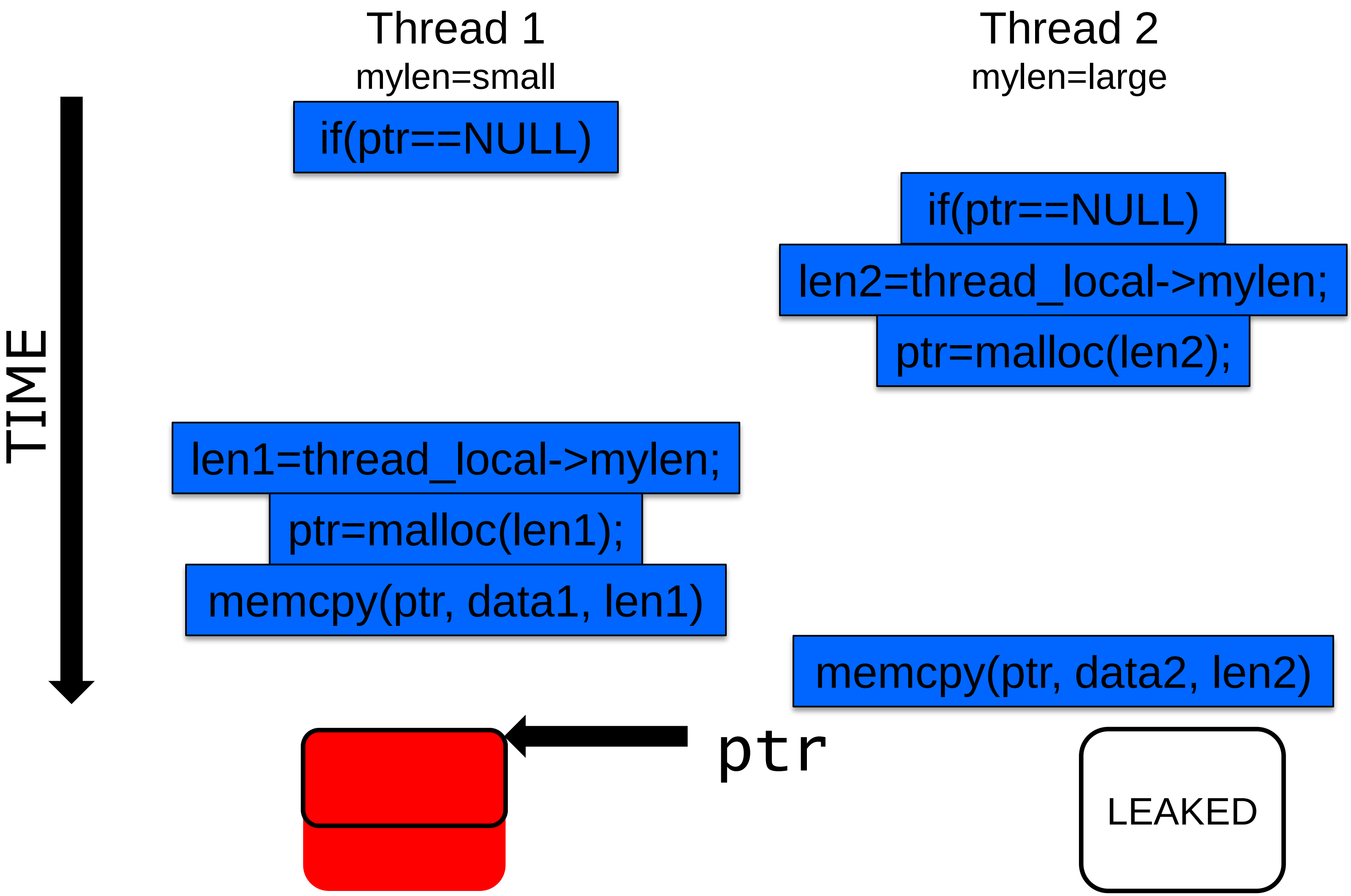
Example of a Modern Bug



Example of a Modern Bug

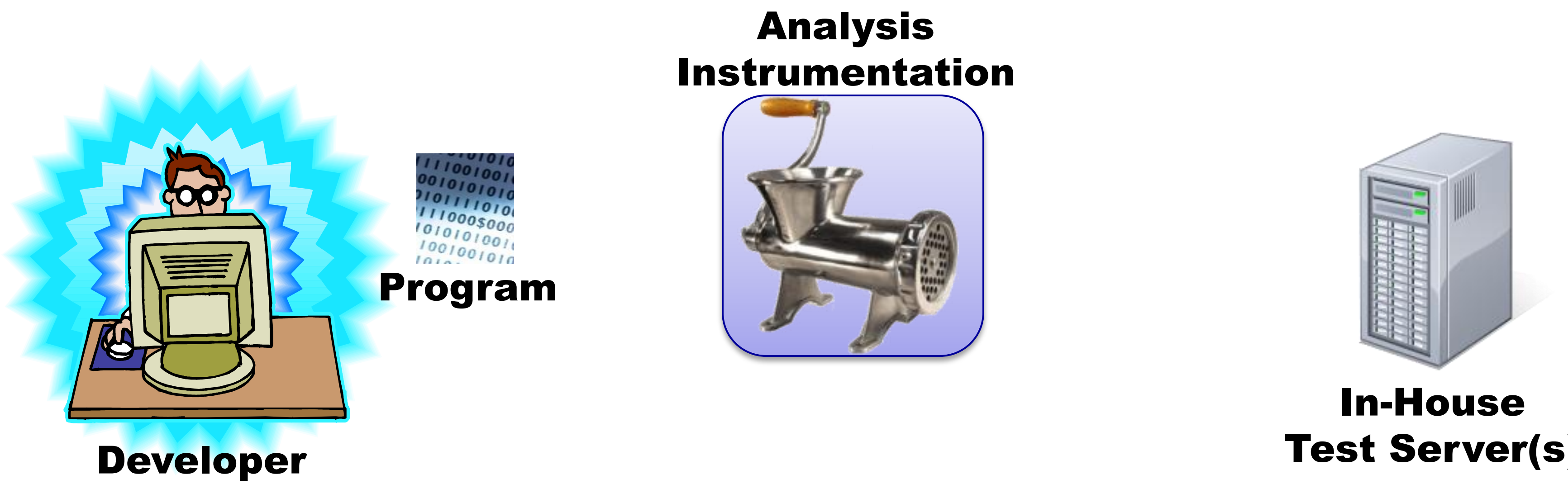


Example of a Modern Bug



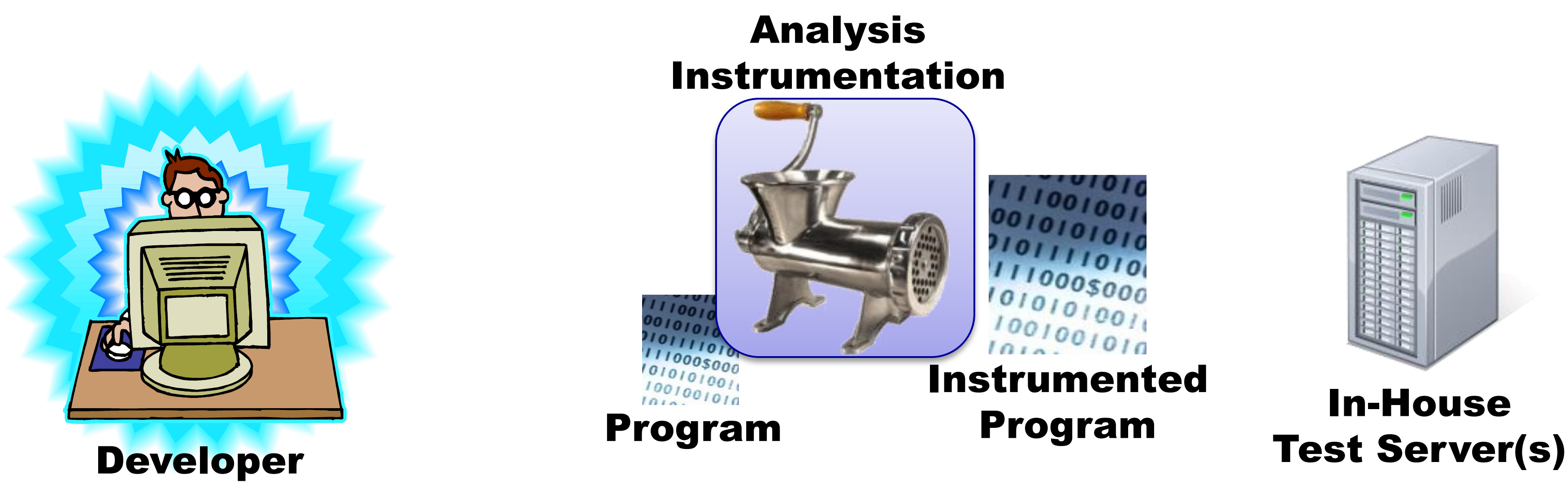
Dynamic Software Analysis

- Analyze the program as it runs
  - \+ System state, find errors on any executed path
  - – LARGE runtime overheads, only test one path



Dynamic Software Analysis

- Analyze the program as it runs
  - \+ System state, find errors on any executed path
  - – LARGE runtime overheads, only test one path



Dynamic Software Analysis

- Analyze the program as it runs
  - \+ System state, find errors on any executed path
  - – LARGE runtime overheads, only test one path

Developer

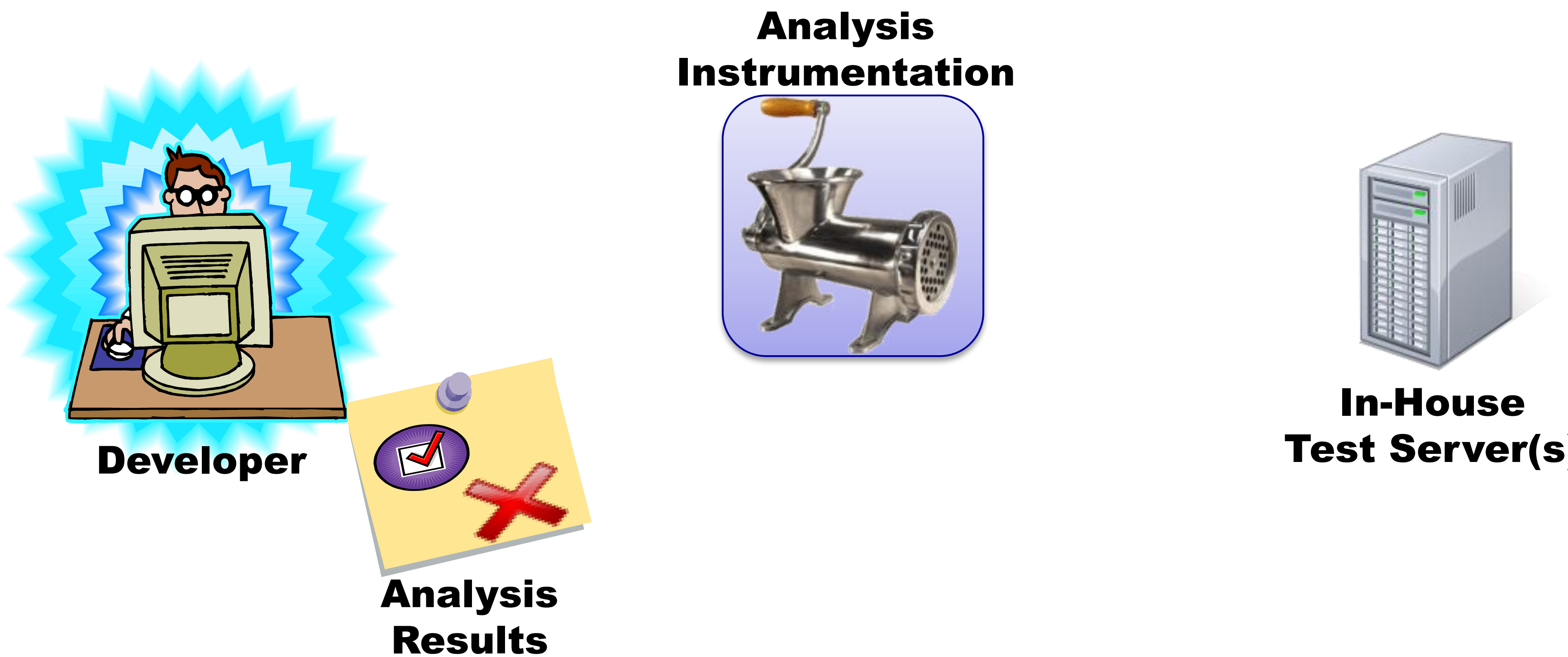
Analysis Instrumentation

LONG run time

In-House Test Server(s)

Dynamic Software Analysis

- Analyze the program as it runs
  - \+ System state, find errors on any executed path
  - – LARGE runtime overheads, only test one path



Runtime Overheads: How Large?

- Data Race Detection
  (e.g. Inspector XE)

  2-300x

- Taint Analysis
  (e.g.TaintCheck)

  2-200x

- Memory Checking
  (e.g. MemCheck)

  5-50x

- Dynamic Bounds Checking

  10-80x

- Symbolic Execution

  10-200x

Could use Hardware

- Data Race Detection: HARD, CORD, etc.
- Taint Analysis: Raksha, FlexiTaint, etc.
- Bounds Checking: HardBound

– None Currently Exist; Bugs Are Here Now
– Single-Use Specialization
  – Won't be built due to HW, power, verification costs
  – Unchangeable algorithms locked in HW

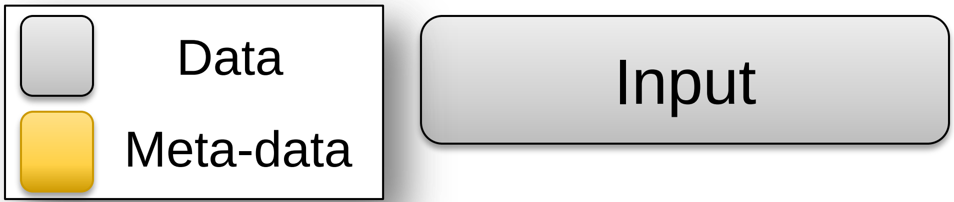
Goals of this Talk

- Accelerate SW Analyses Using Existing HW
- Run Tests **On Demand**: Only When Needed
- Explore Future **Generic HW Additions**

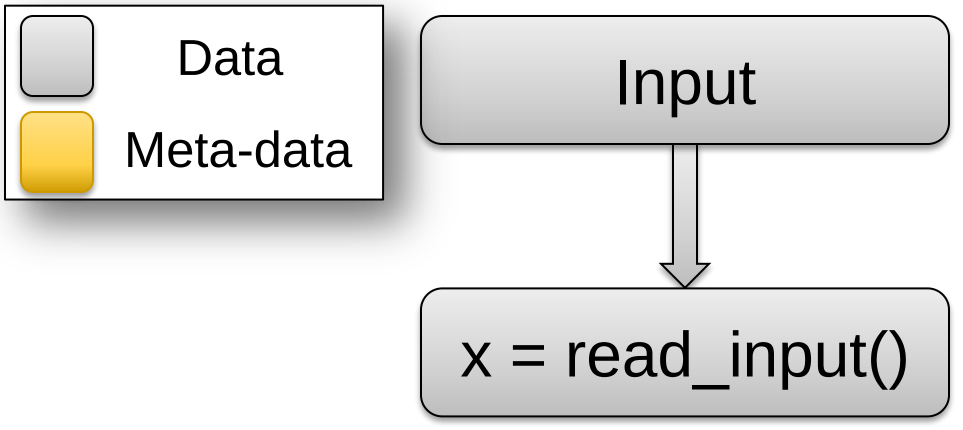
Outline

- Problem Statement
- Background Information
  - Demand-Driven Dynamic Dataflow Analysis
- Proposed Solutions
  - Demand-Driven Data Race Detection
  - Unlimited Hardware Watchpoints

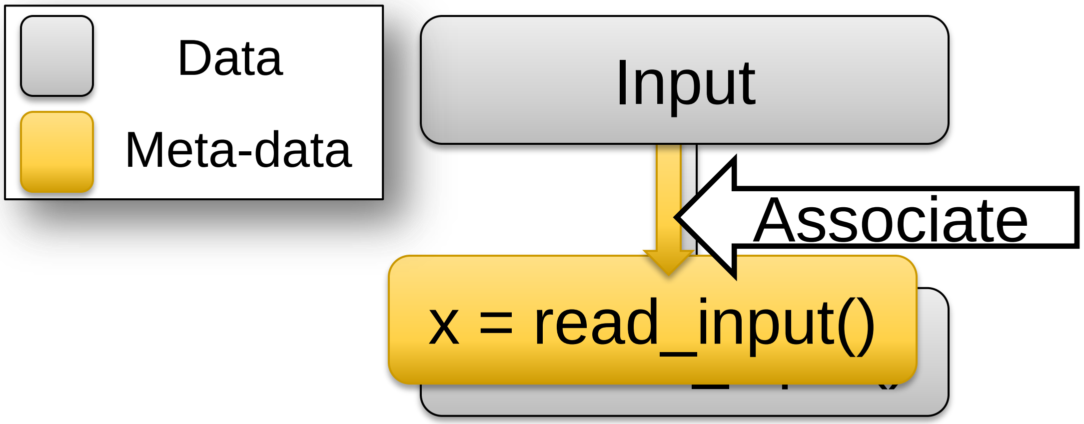
Example Dynamic Dataflow Analysis



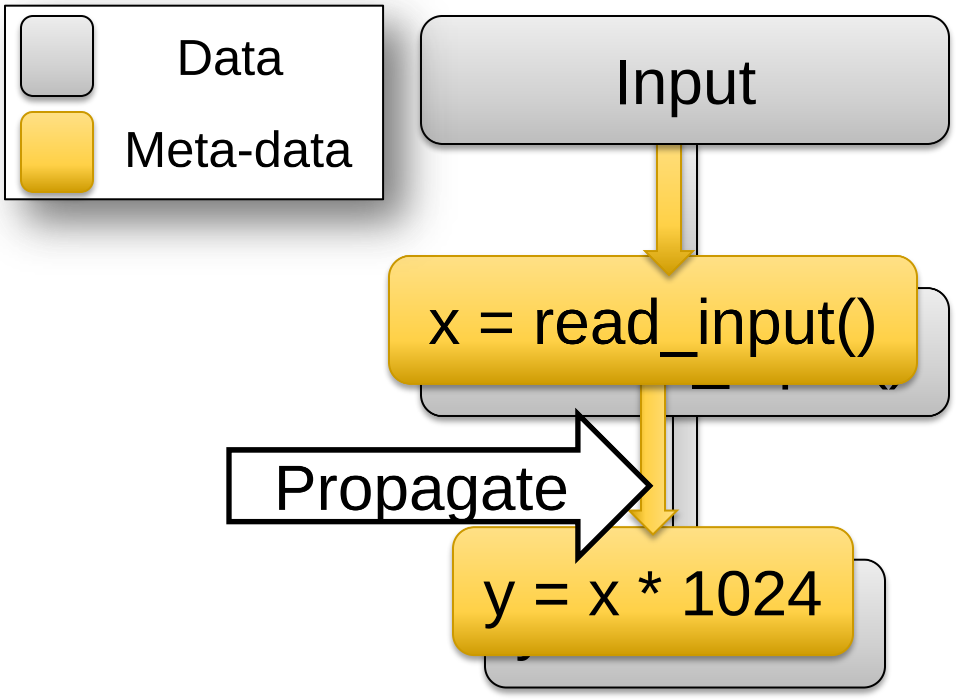
Example Dynamic Dataflow Analysis



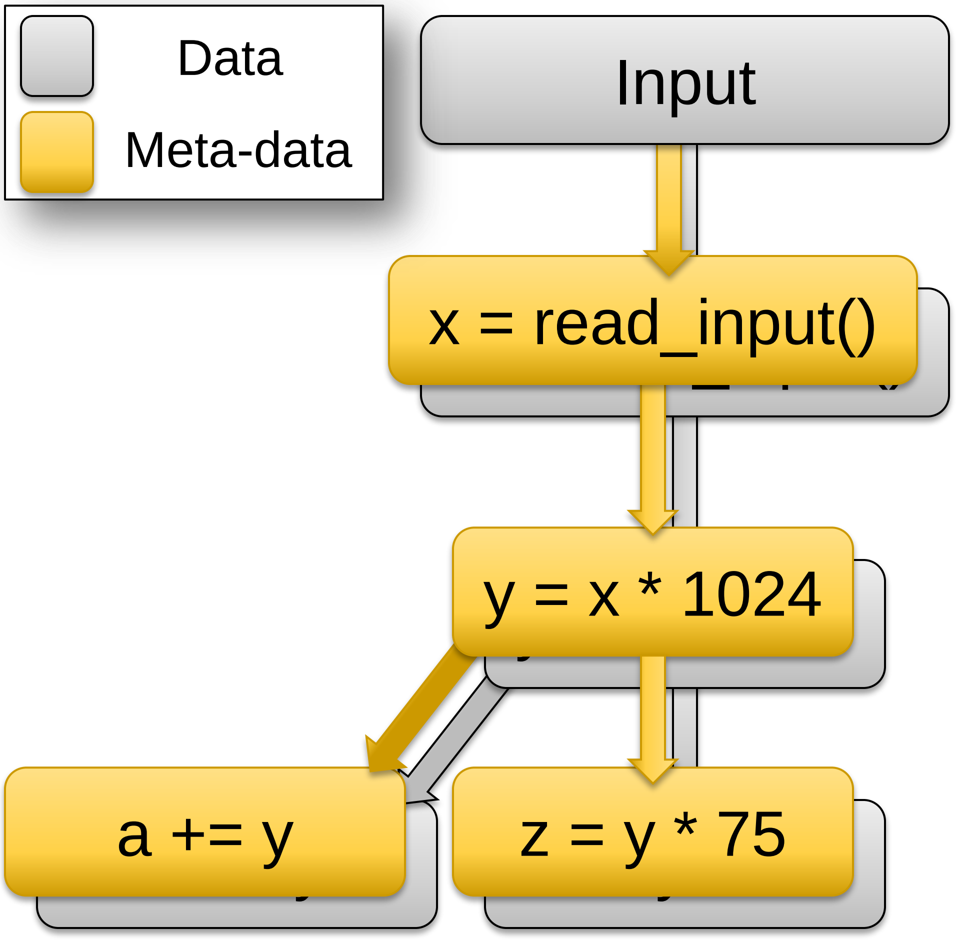
Example Dynamic Dataflow Analysis



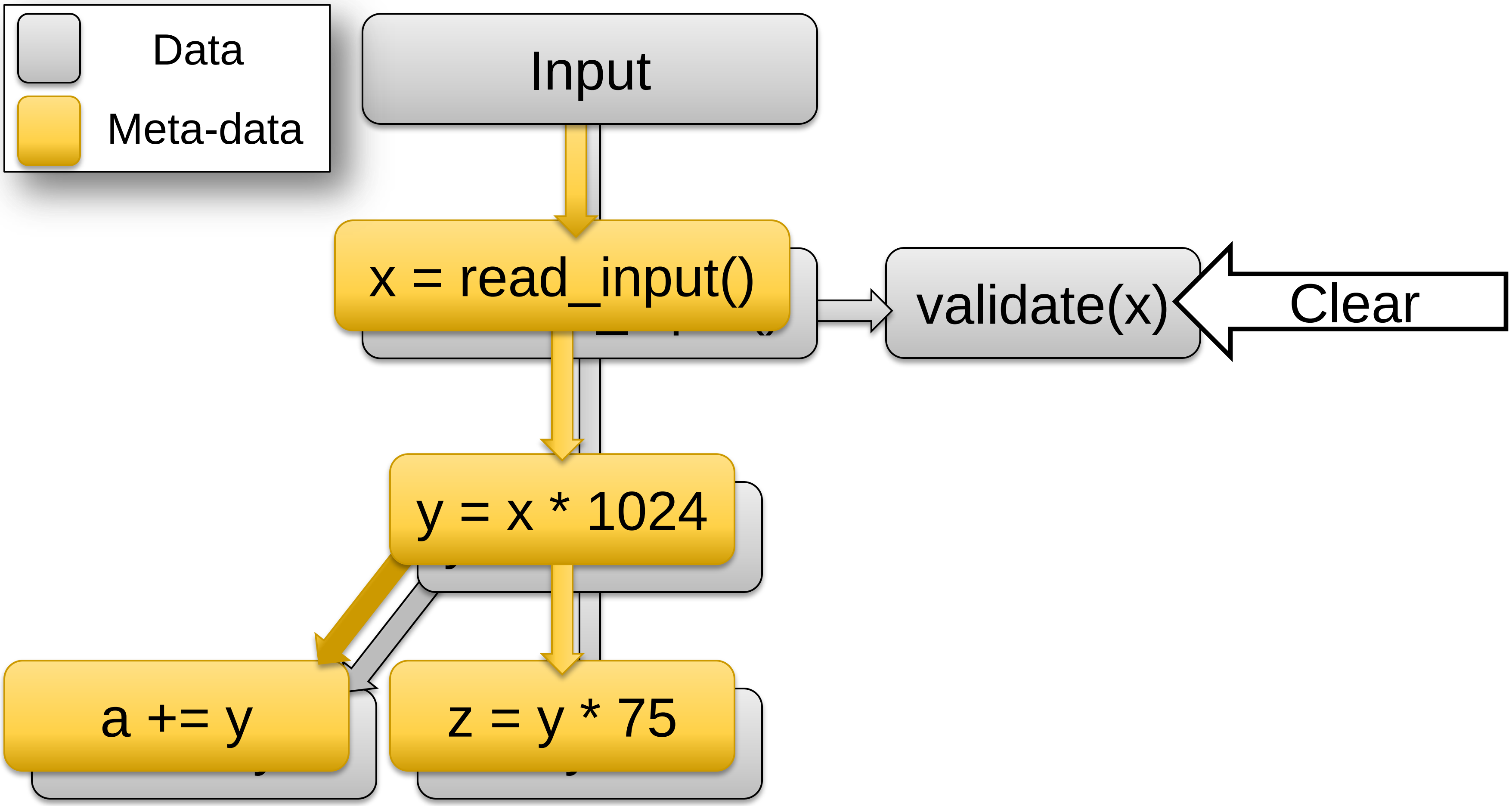
Example Dynamic Dataflow Analysis



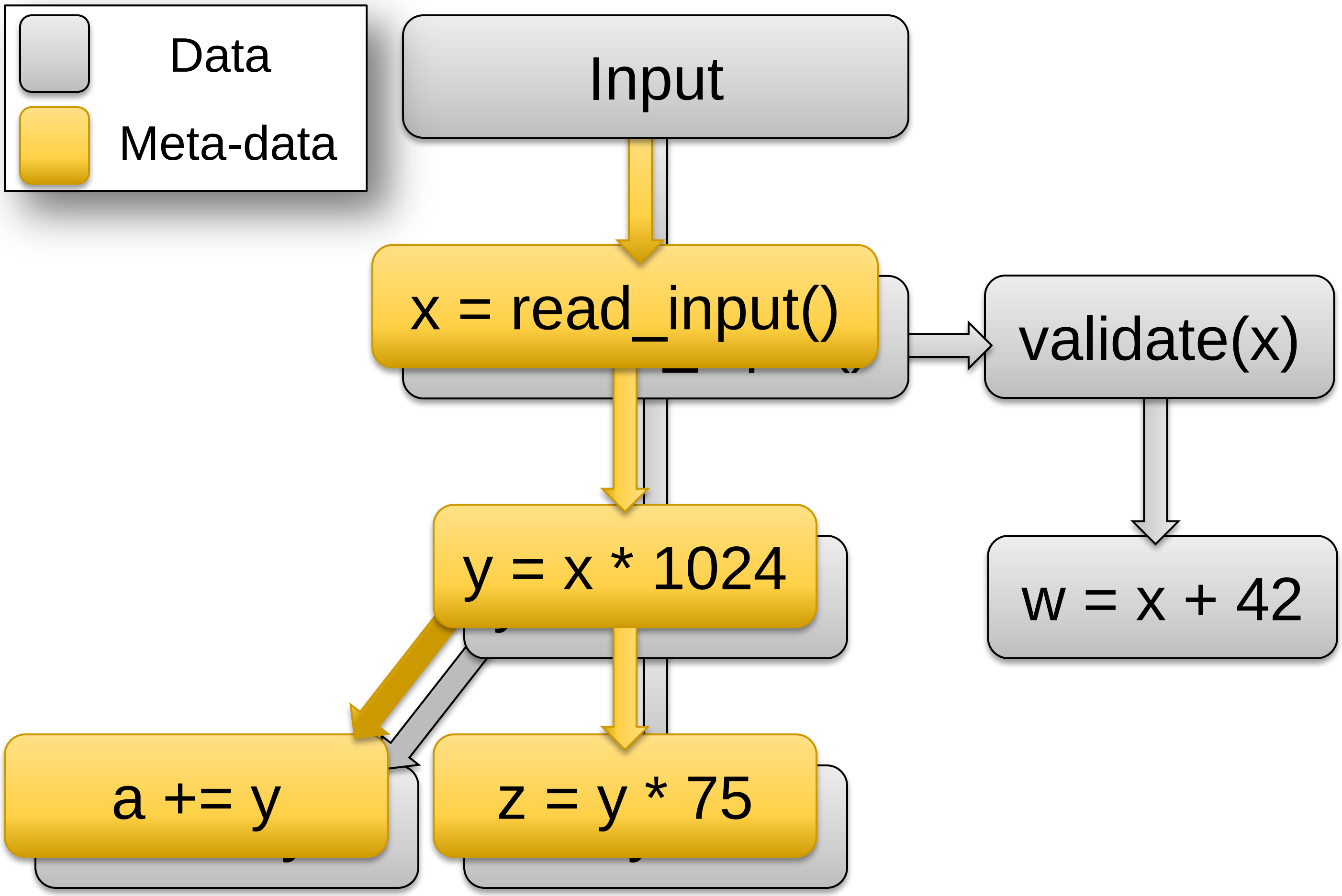
Example Dynamic Dataflow Analysis



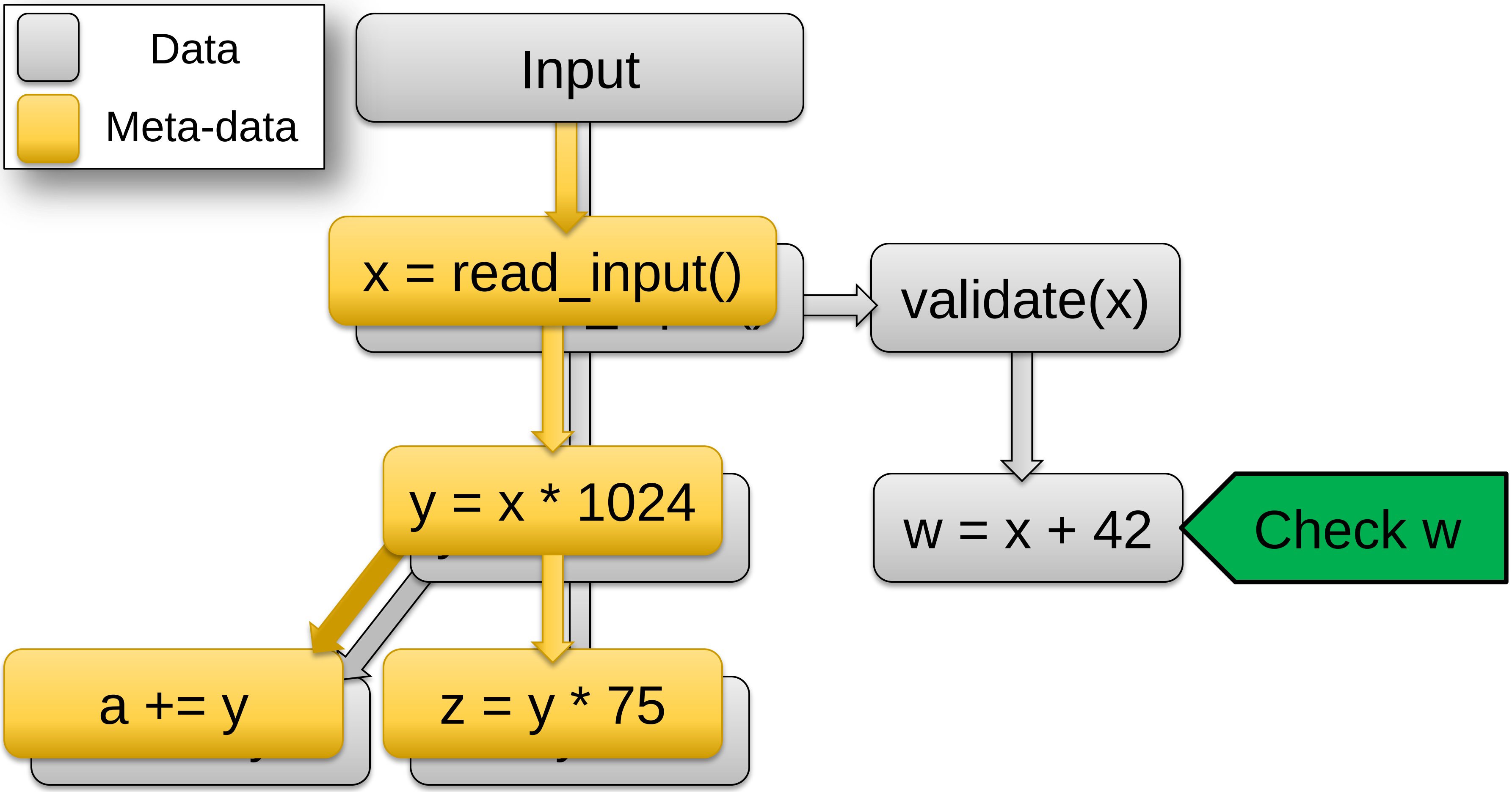
Example Dynamic Dataflow Analysis



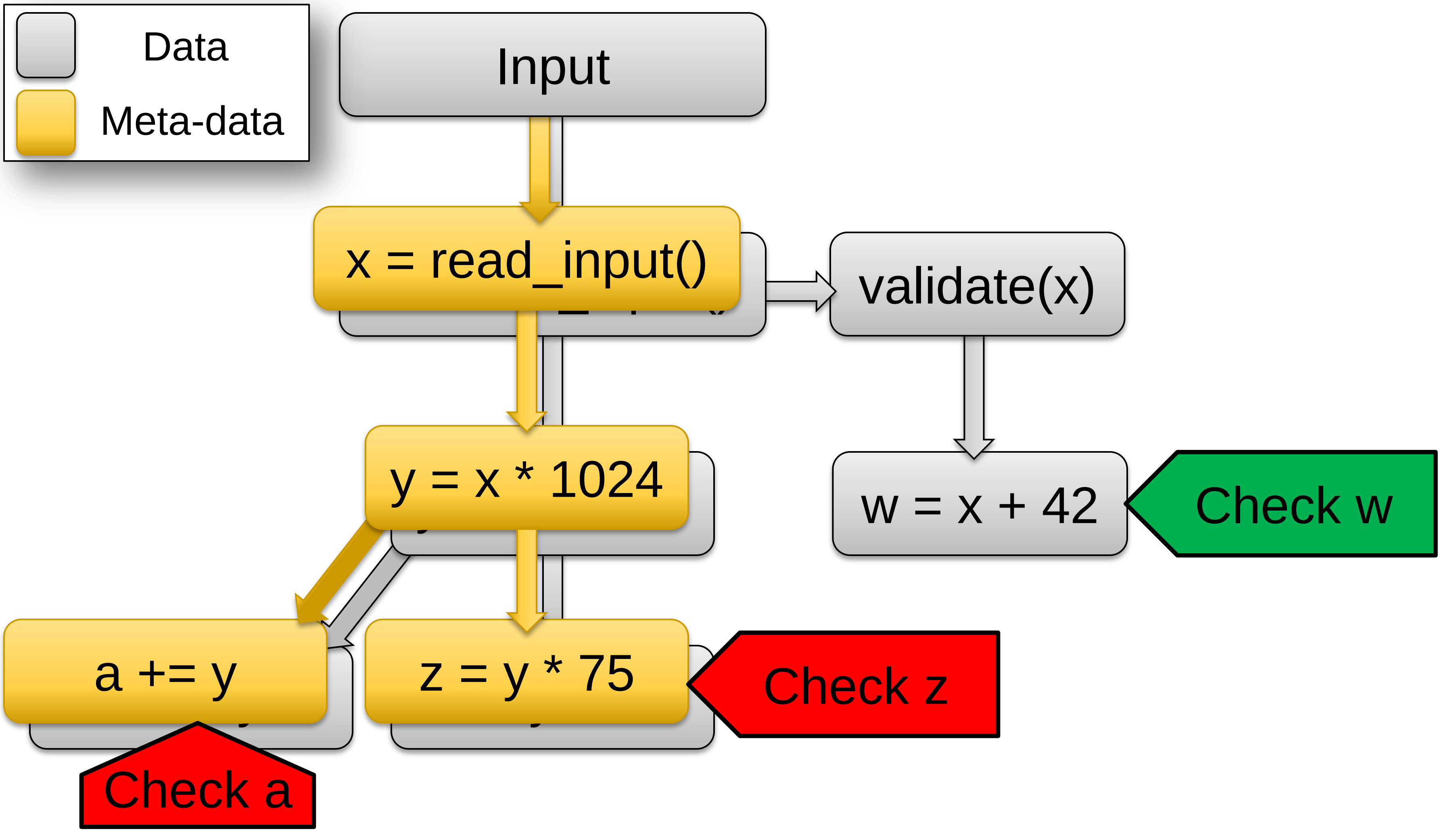
Example Dynamic Dataflow Analysis



Example Dynamic Dataflow Analysis

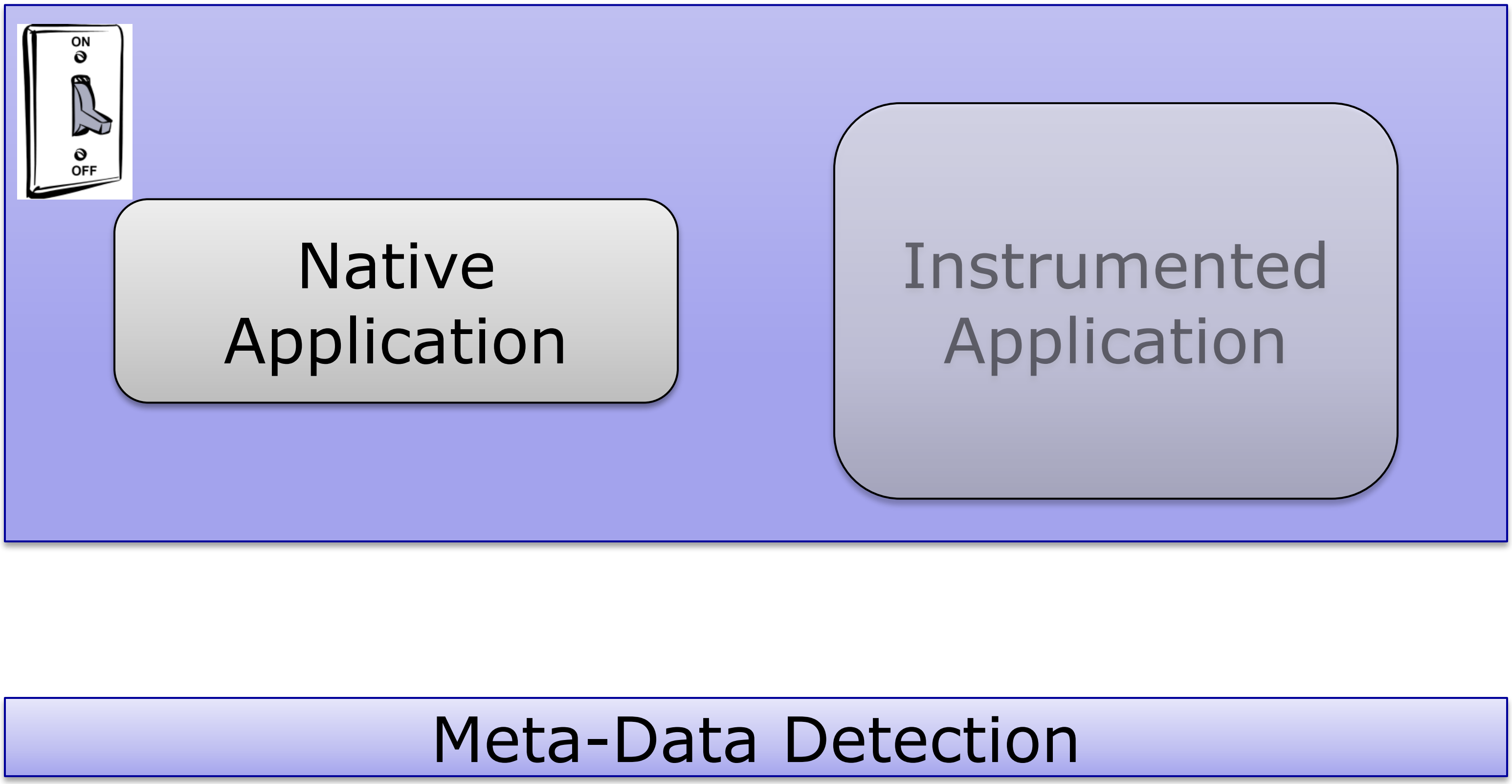


Example Dynamic Dataflow Analysis



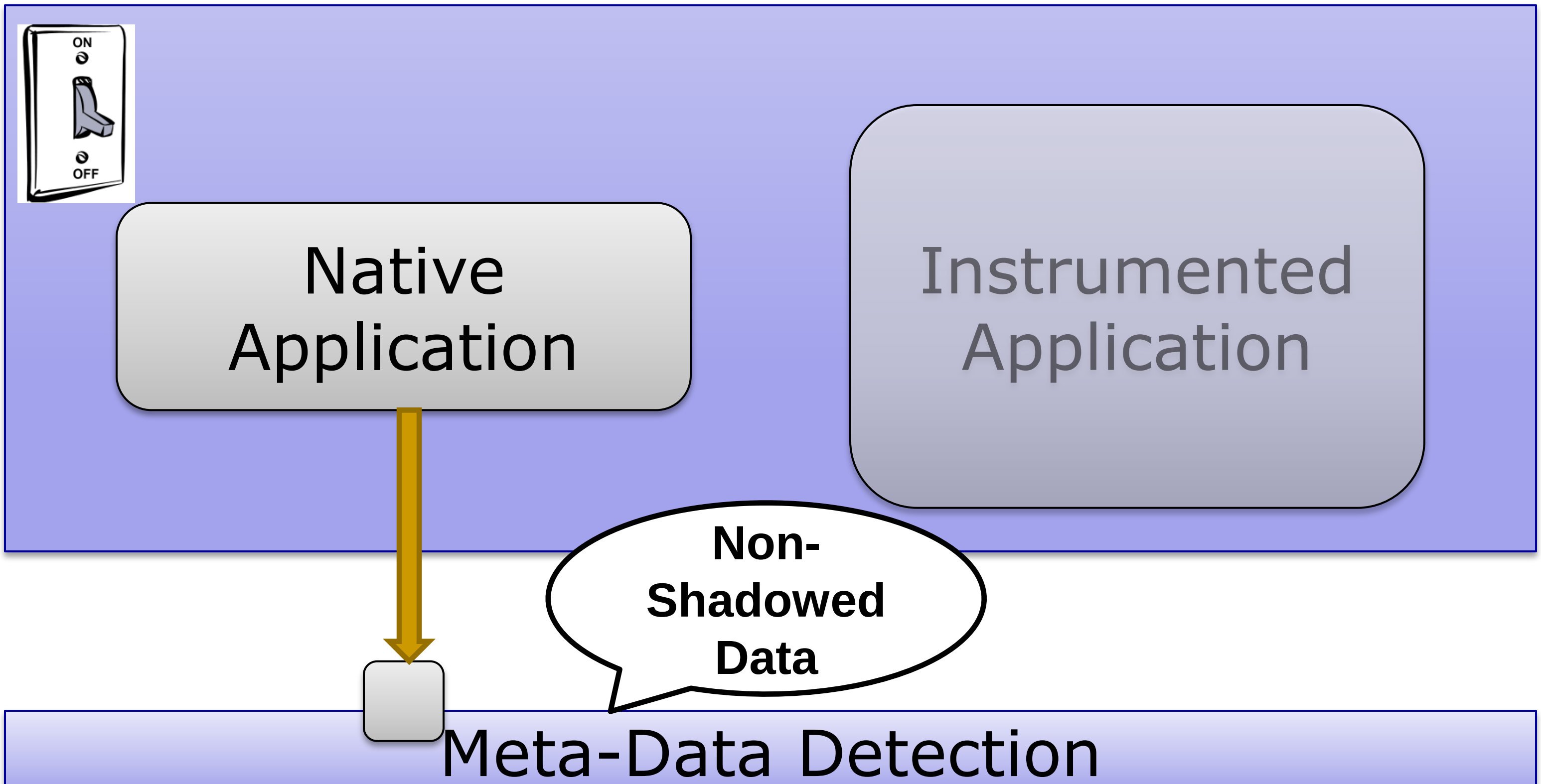
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data



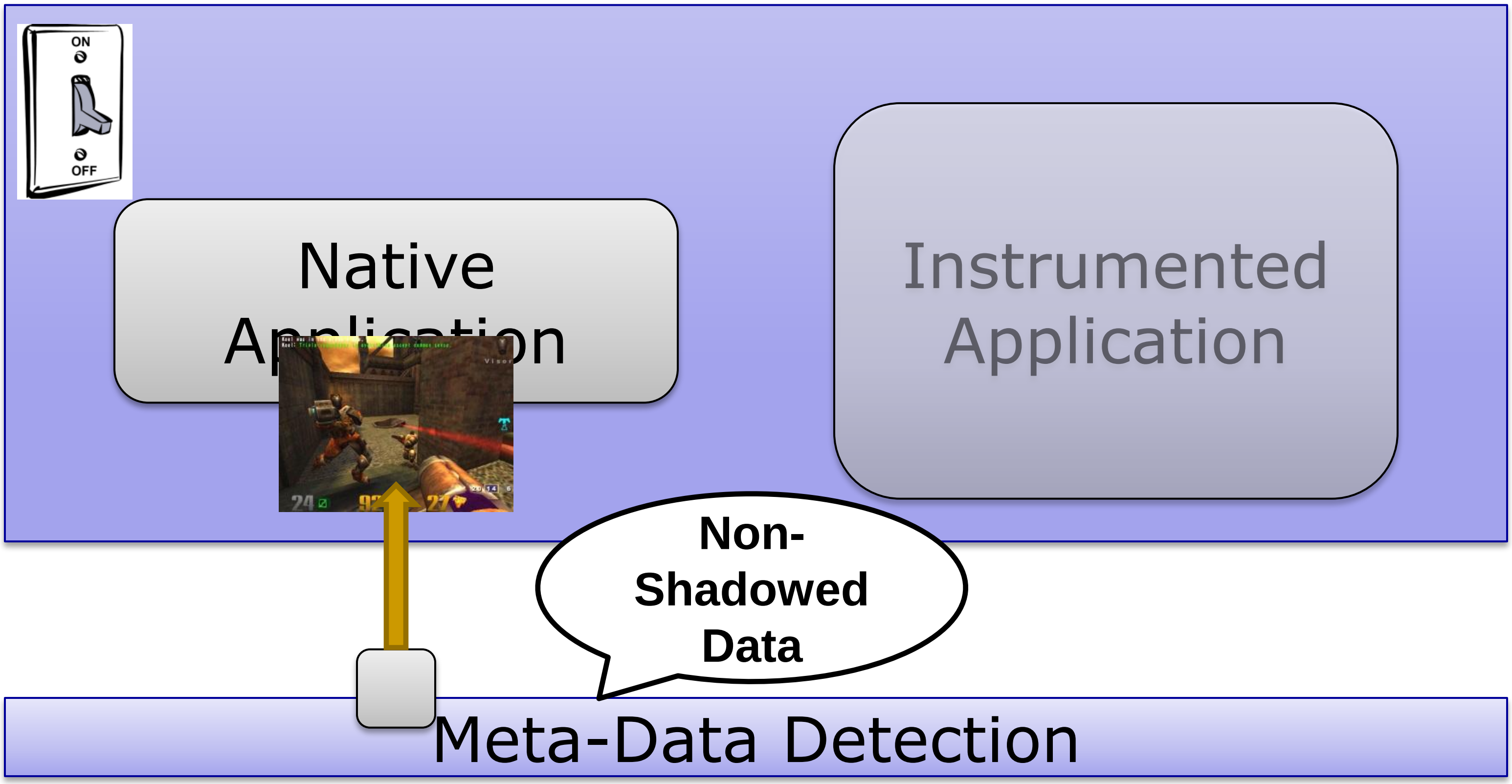
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data



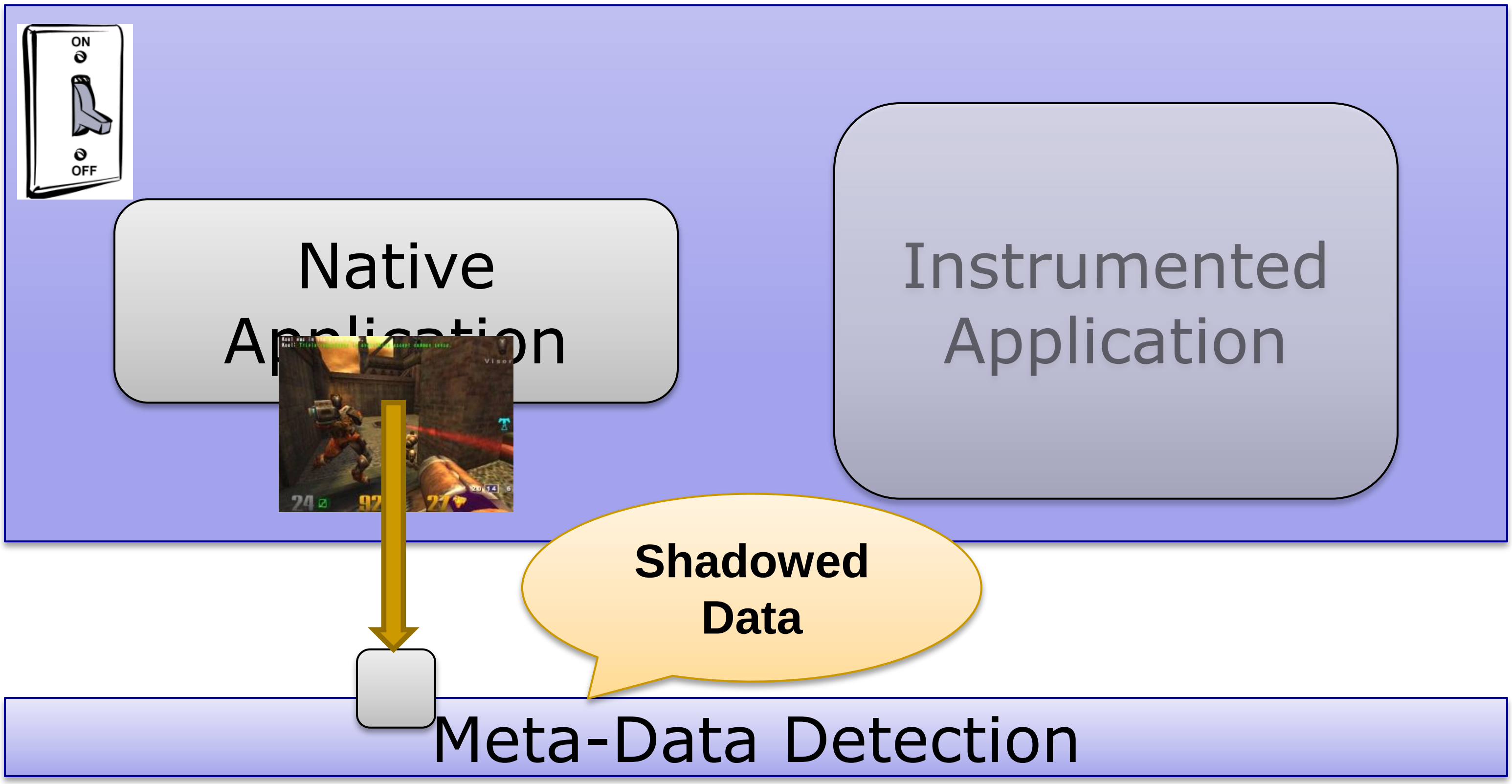
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data



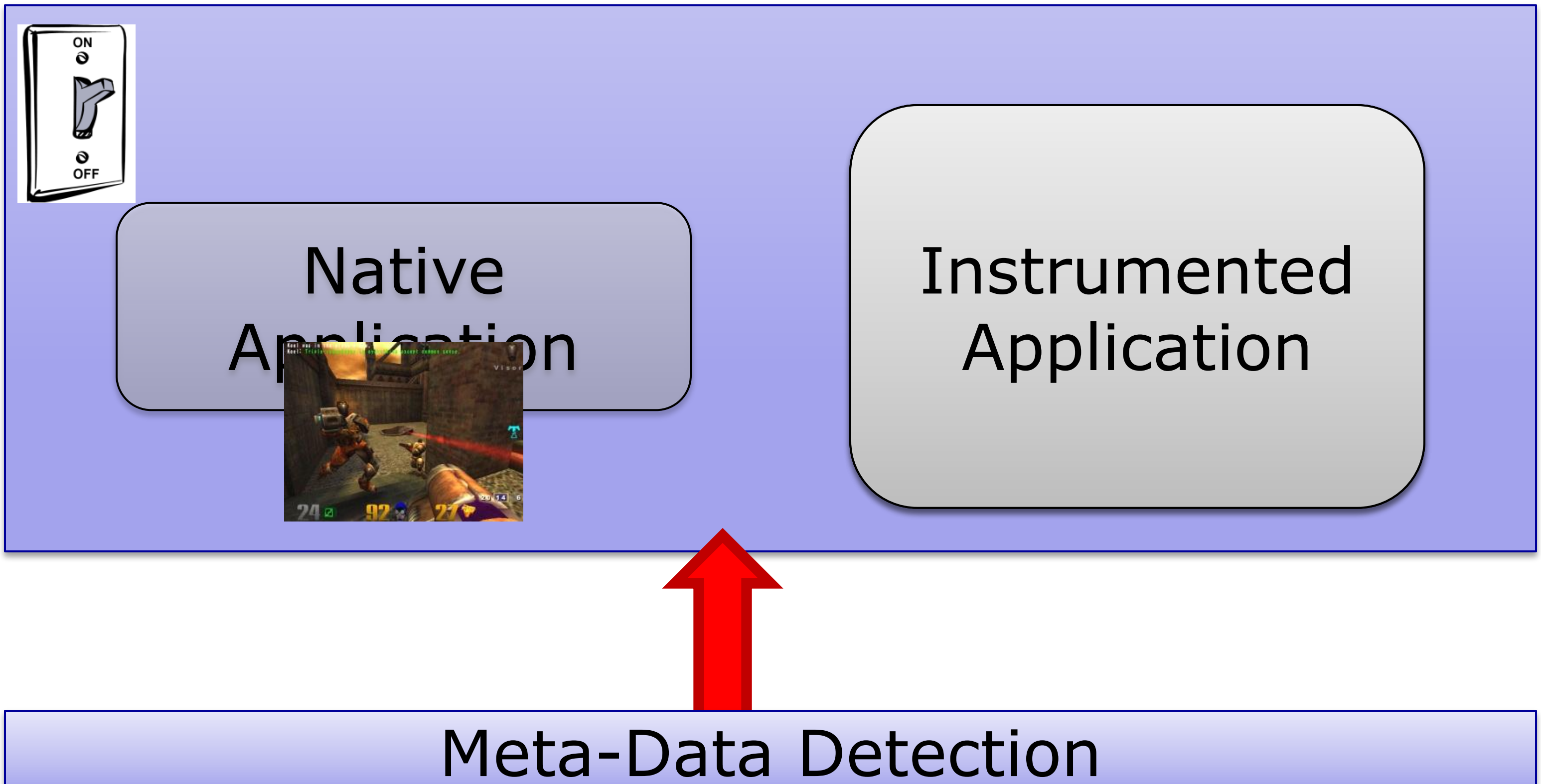
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data



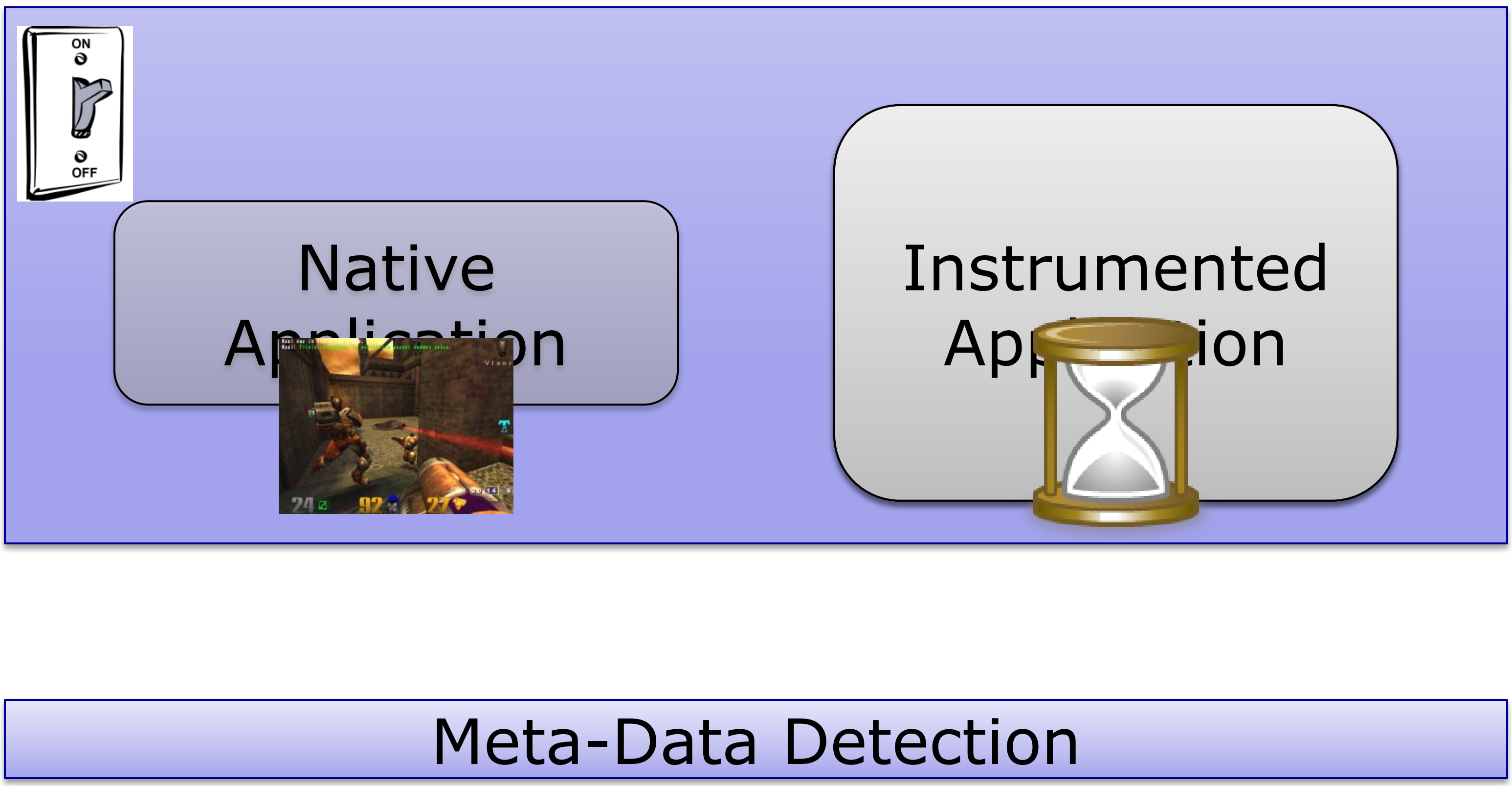
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data



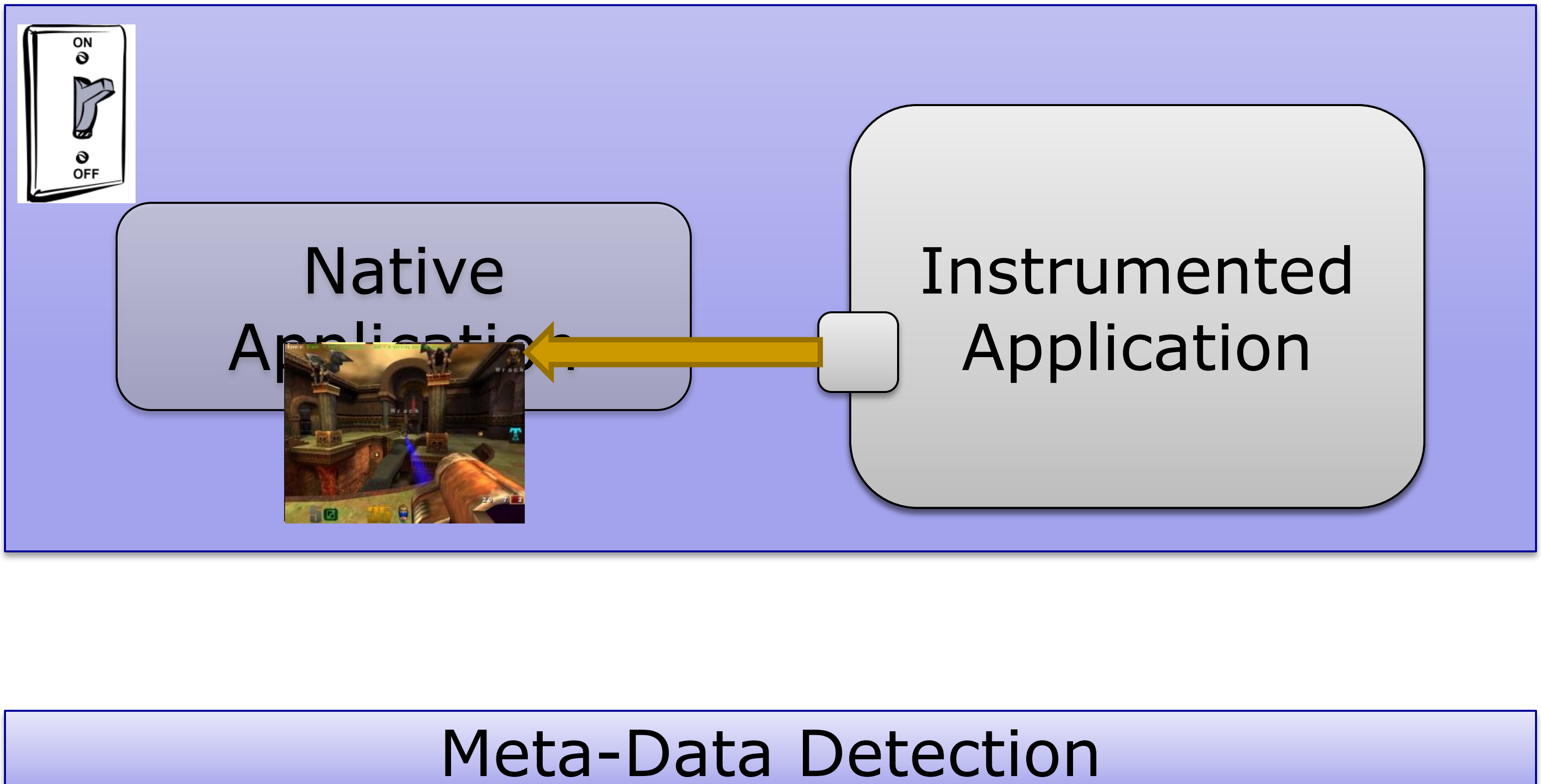
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data



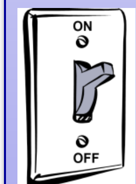
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data



Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data



Native Application

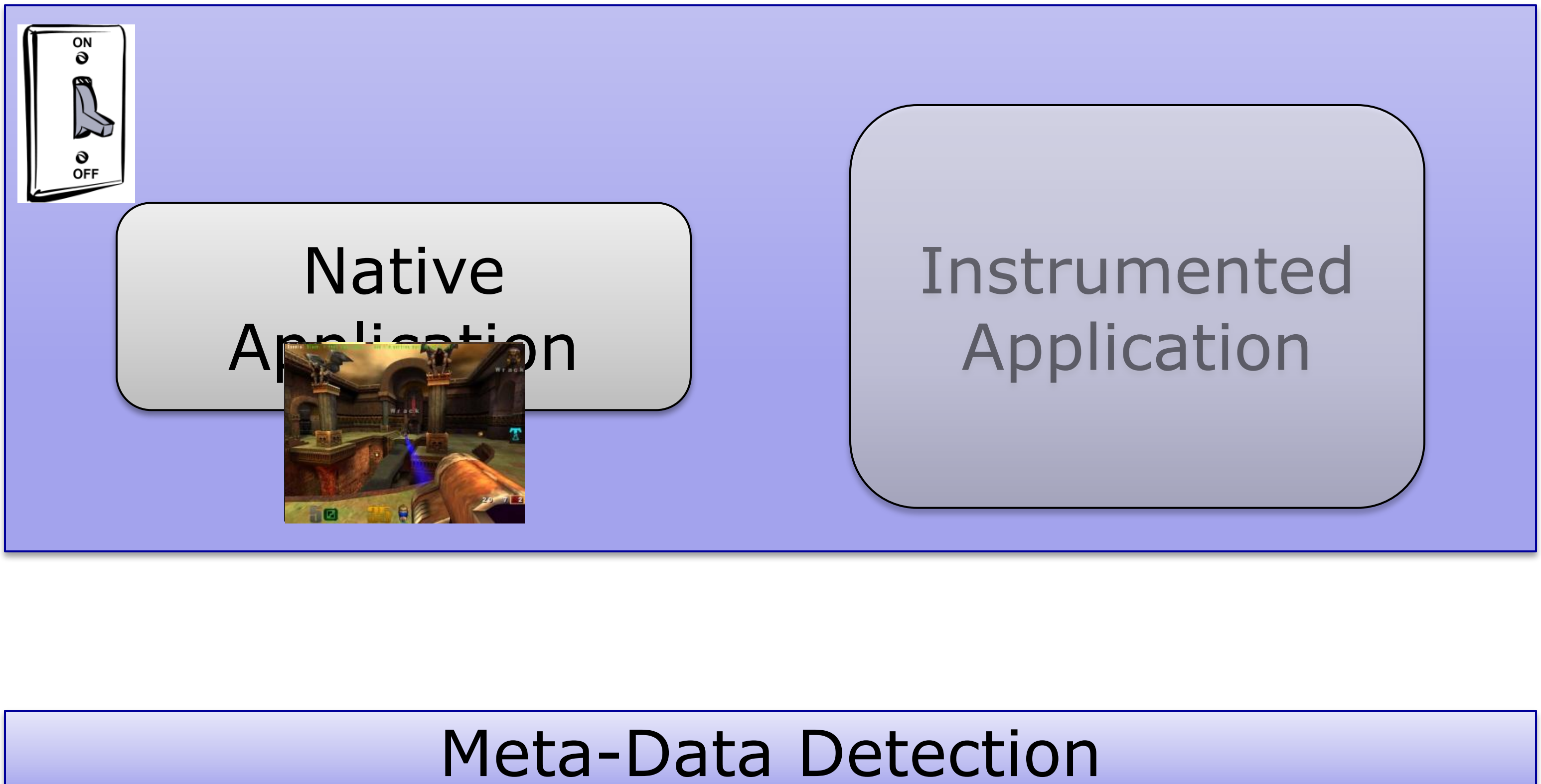
Instrumented Application

No meta-data

Meta-Data Detection

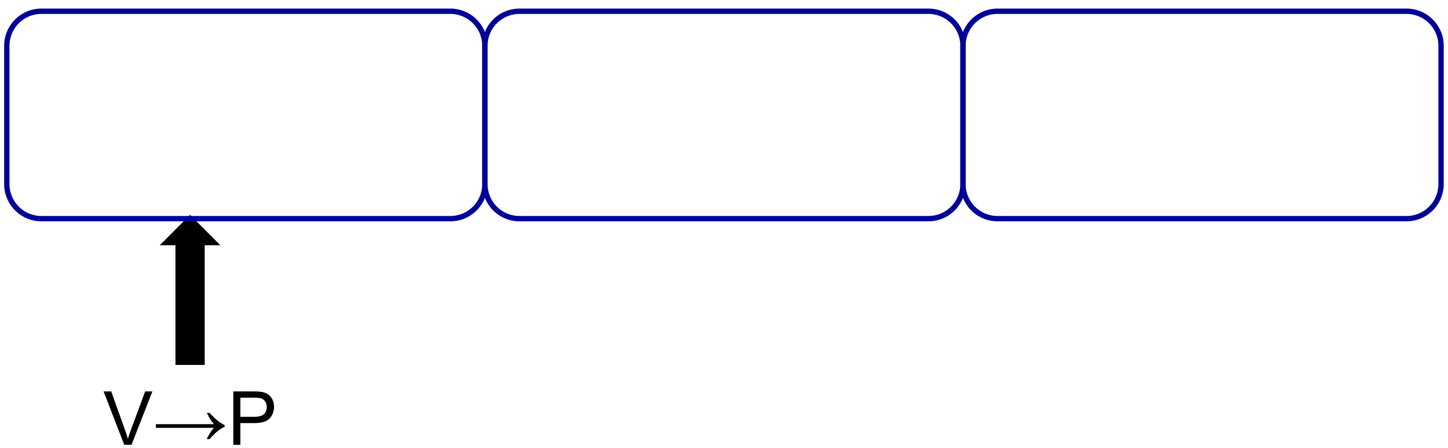
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data



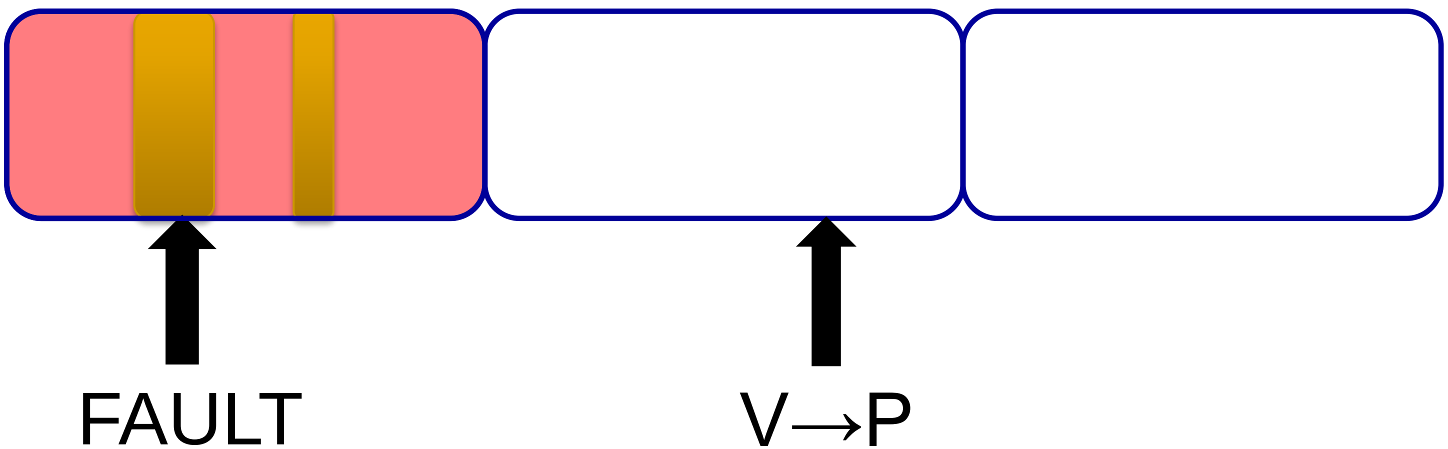
Finding Meta-Data

- No additional overhead when no meta-data
  - Needs hardware support
- Take a fault when touching shadowed data
- Solution: Virtual Memory Watchpoints



Finding Meta-Data

- No additional overhead when no meta-data
  - Needs hardware support
- Take a fault when touching shadowed data
- Solution: Virtual Memory Watchpoints



Results by Ho et al.

- From “Practical Taint-Based Protection using Demand Emulation”

System	Slowdown (normalized)
Taint Analysis	101.7x
On-Demand Taint Analysis	1.98x

Outline

- Problem Statement
- Background Information
  - Demand-Driven Dynamic Dataflow Analysis
- Proposed Solutions
  - Demand-Driven Data Race Detection
  - Unlimited Hardware Watchpoints

Software Data Race Detection

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

Data Race Detection

TIME

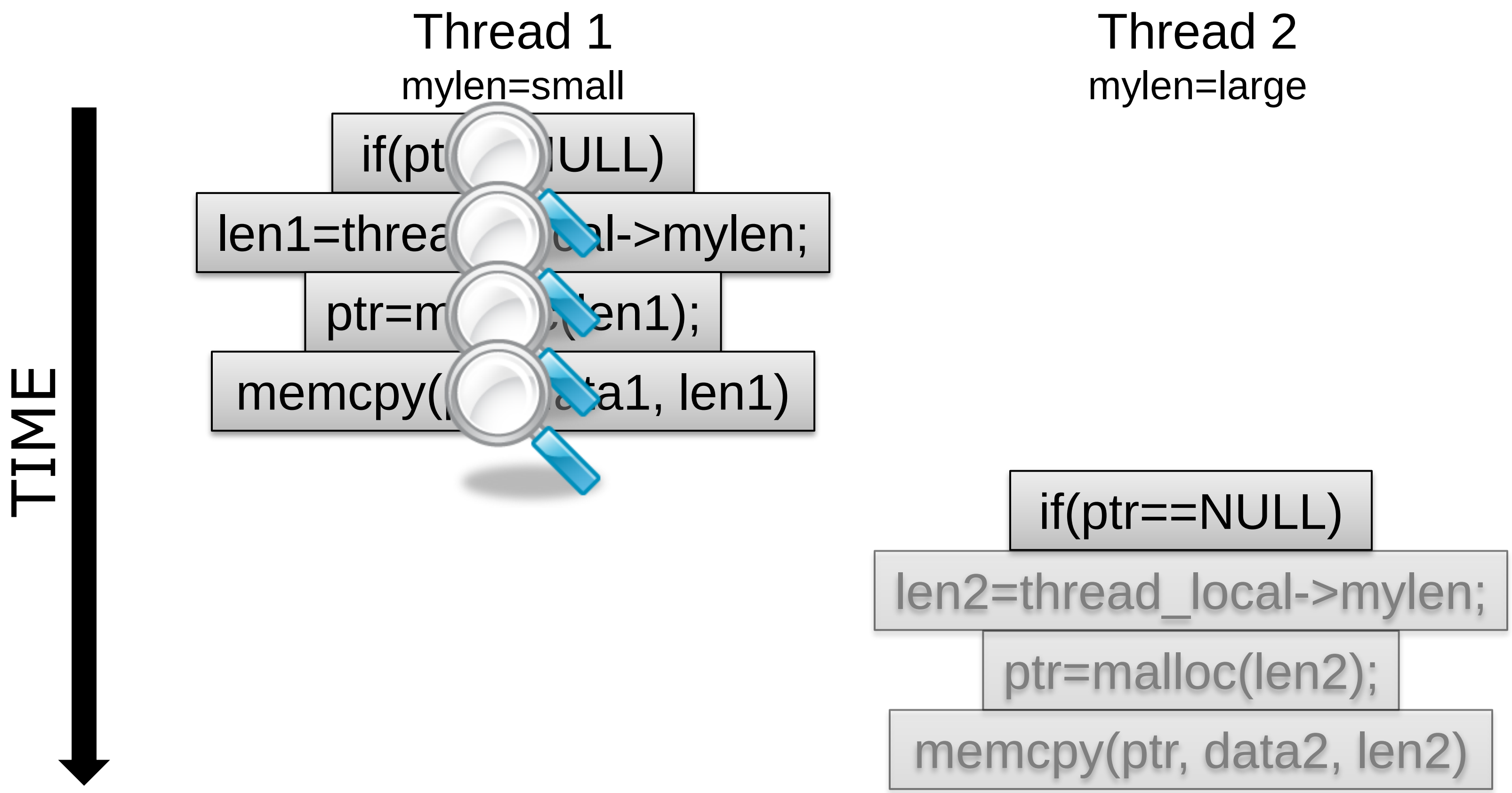
Thread 1
mylen=small

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

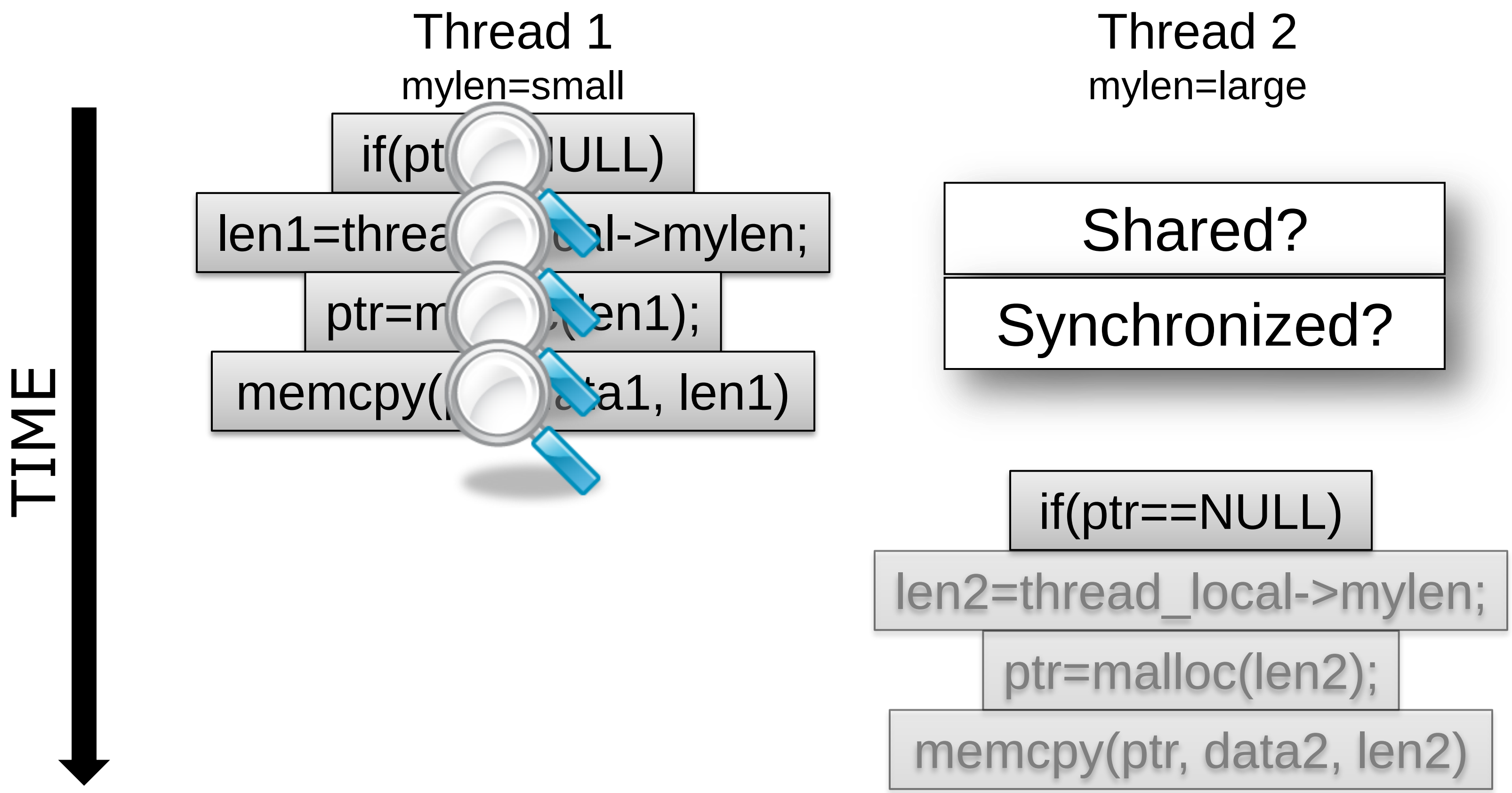
Thread 2
mylen=large

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

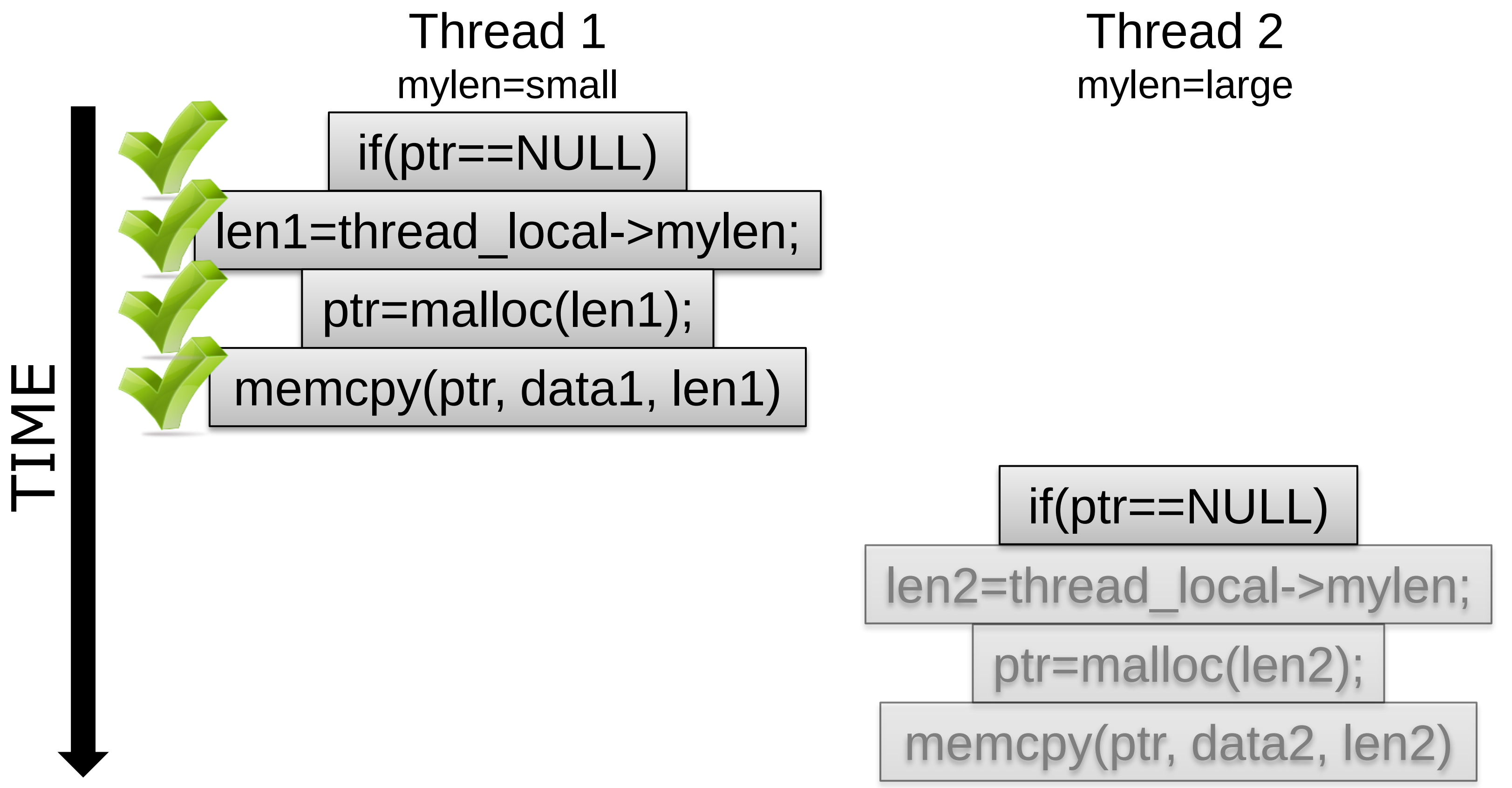
Data Race Detection



Data Race Detection



Data Race Detection

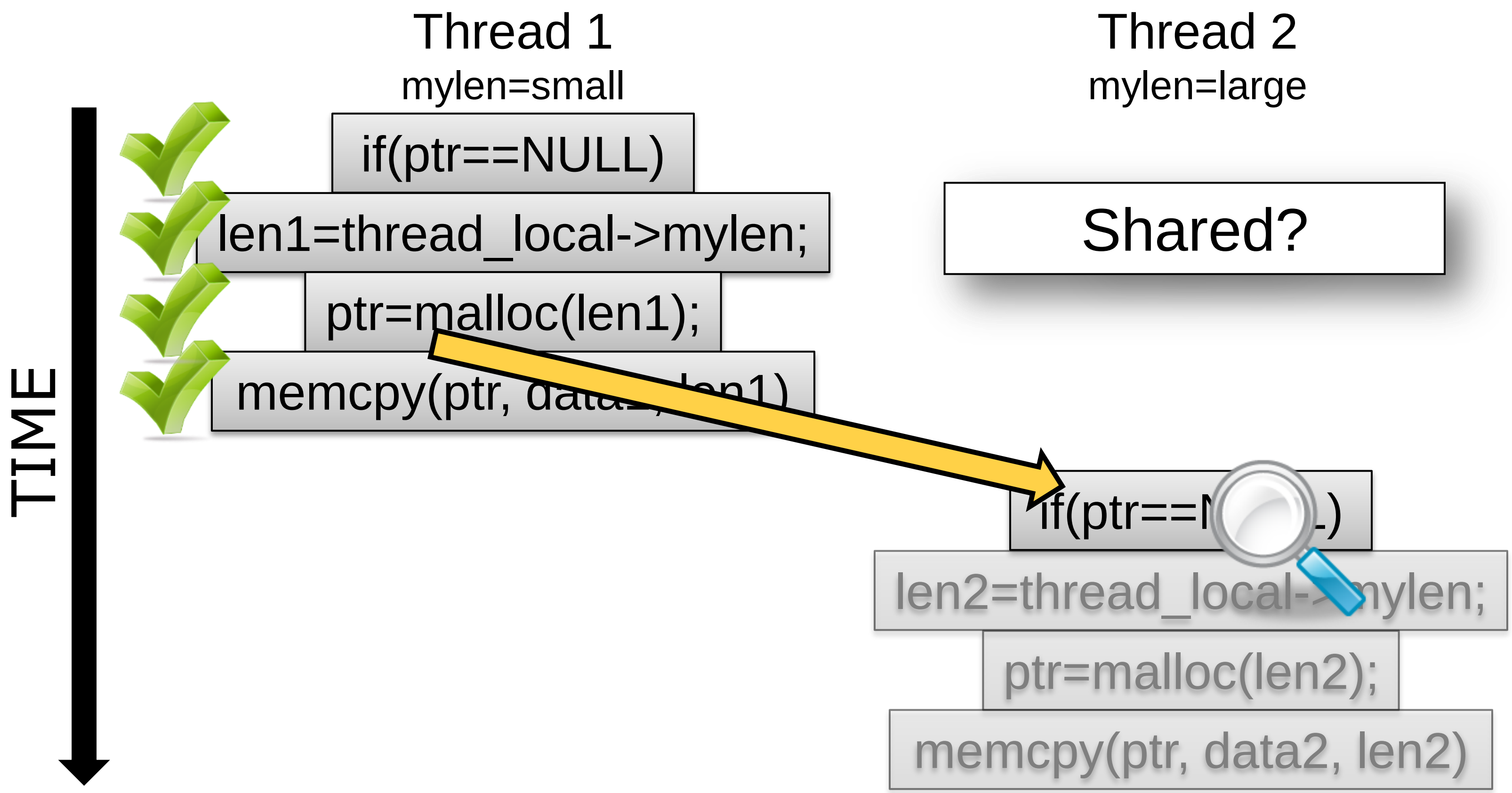


Example of Data Race Detection

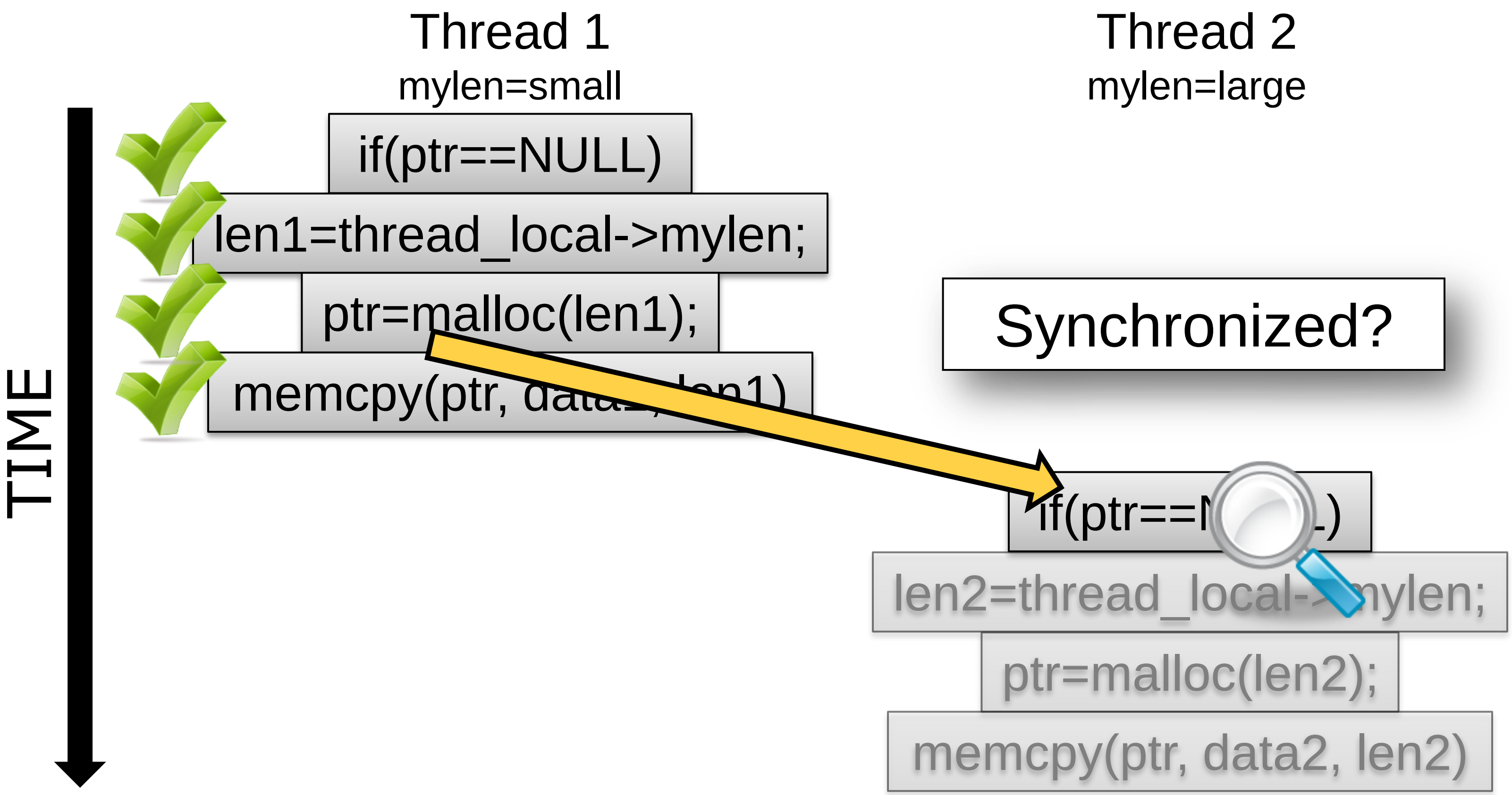
TIME

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

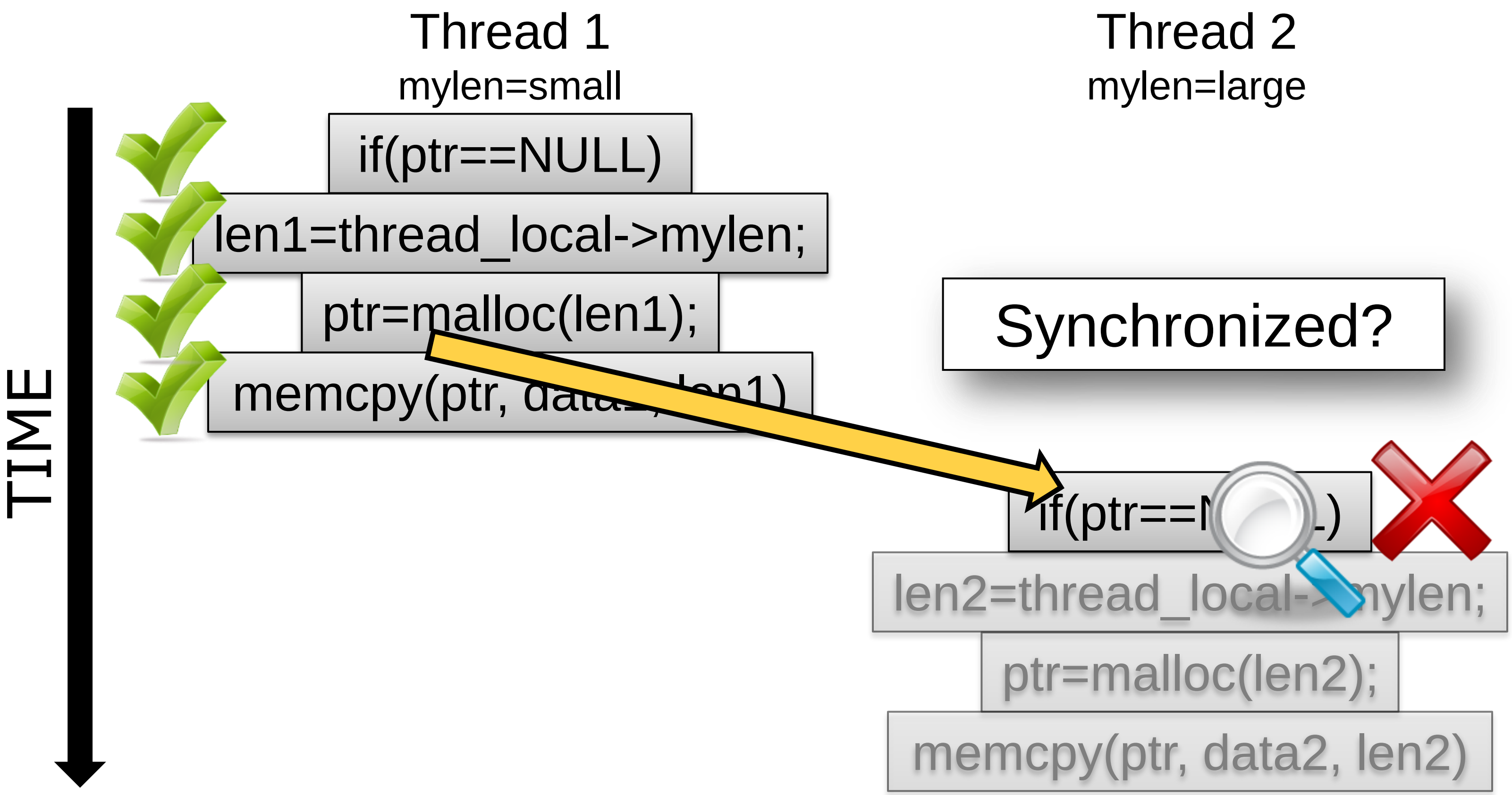
Example of Data Race Detection



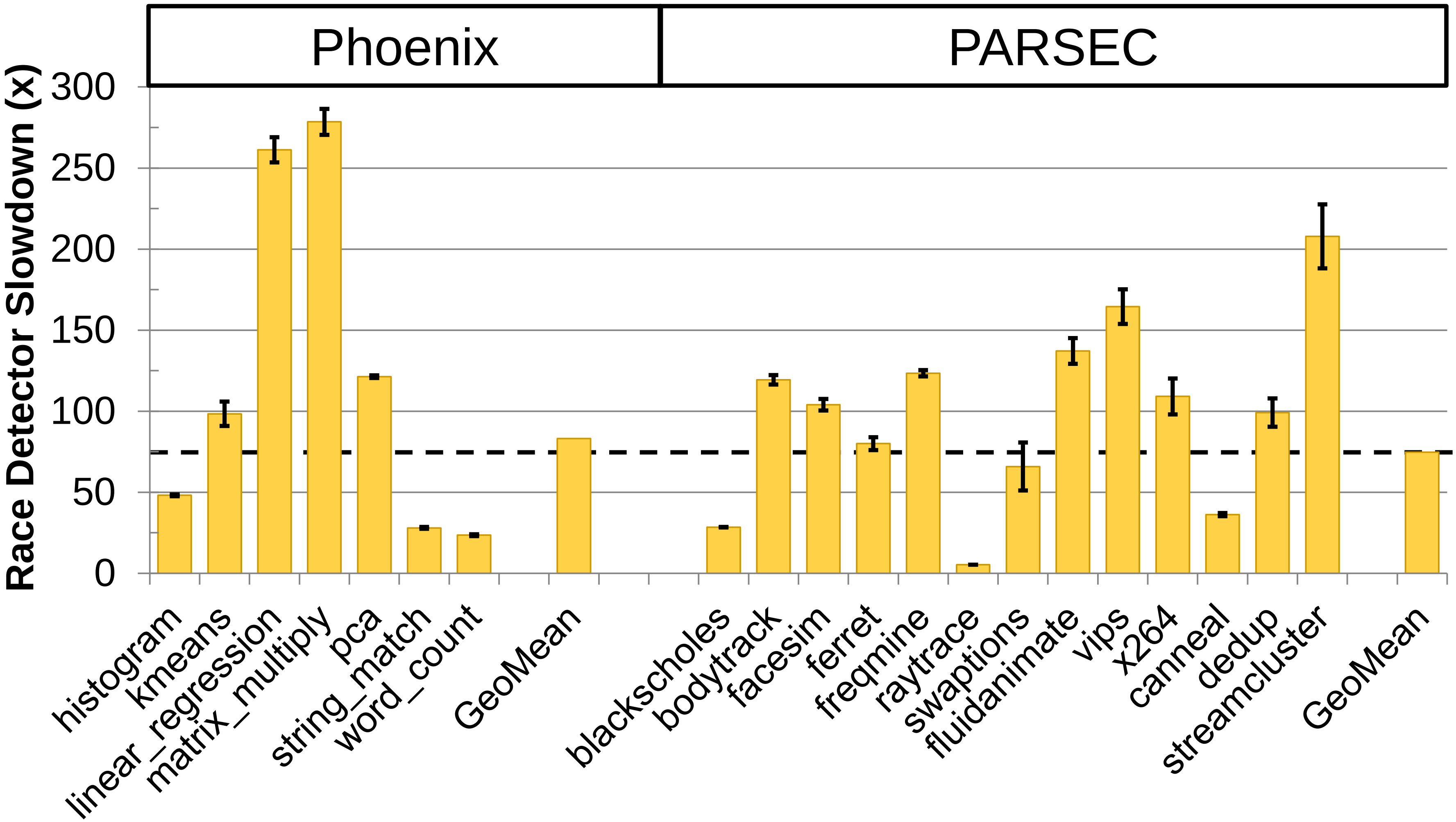
Example of Data Race Detection



Example of Data Race Detection

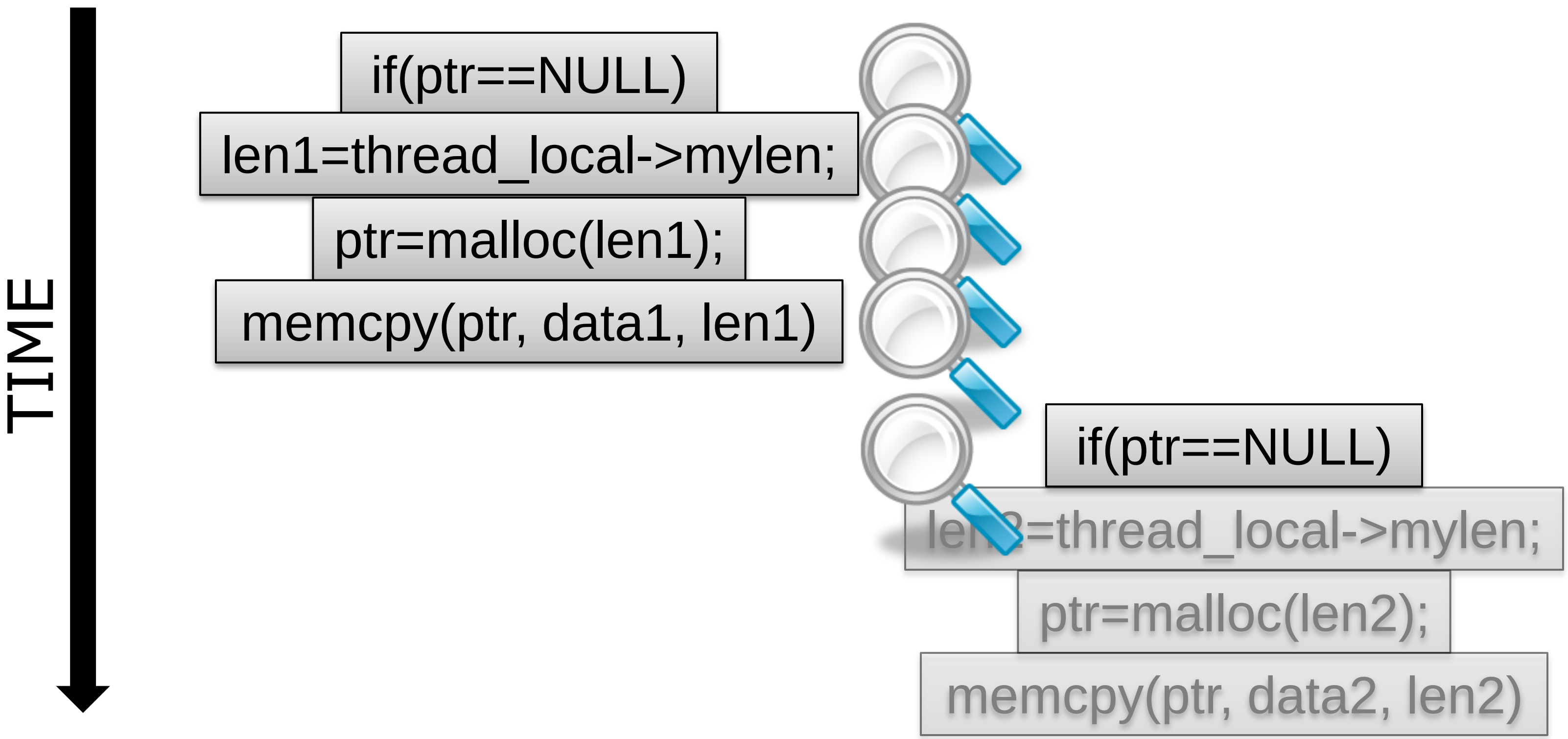


SW Race Detection is Slow



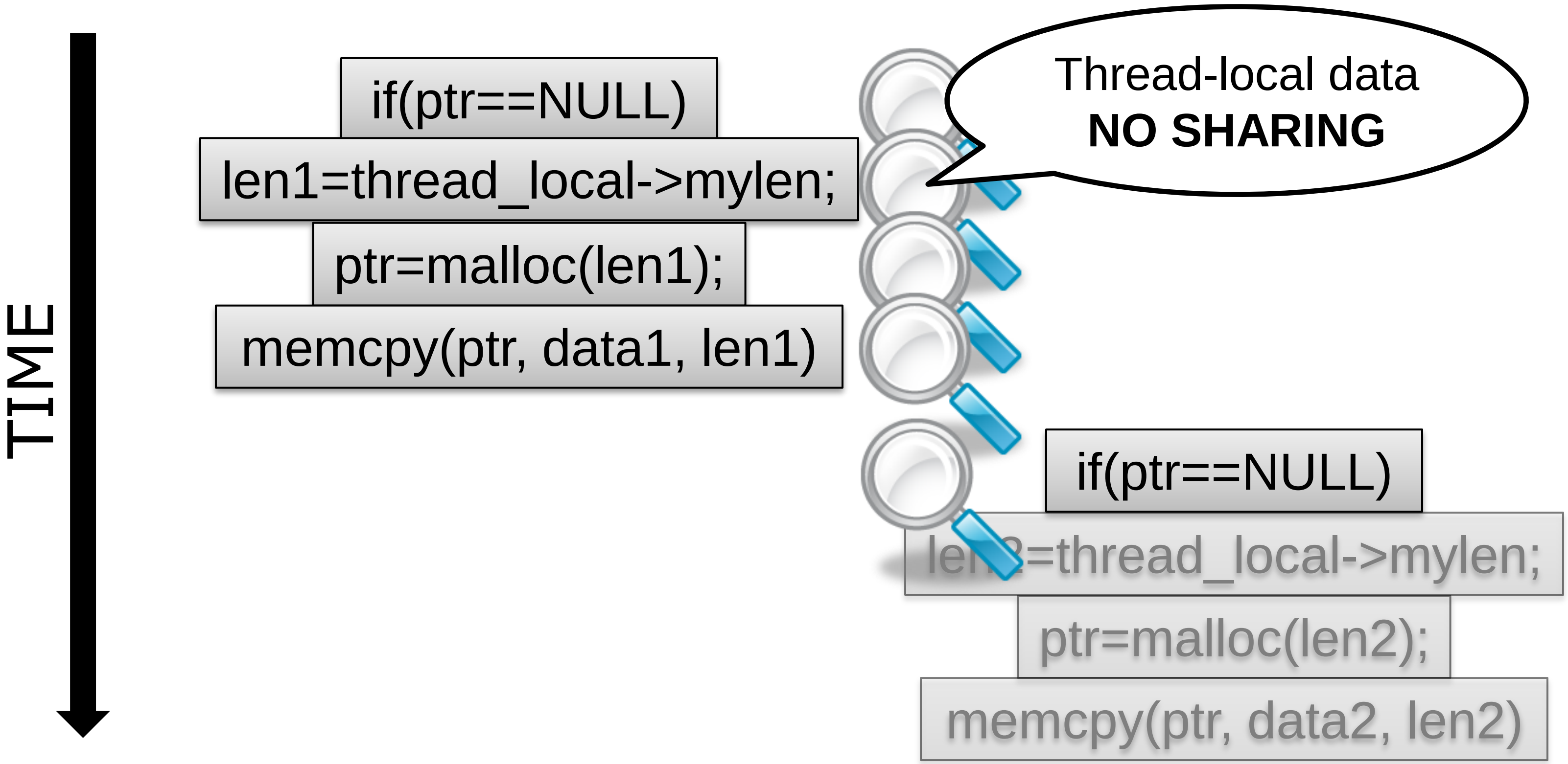
Inter-thread Sharing is What's Important

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



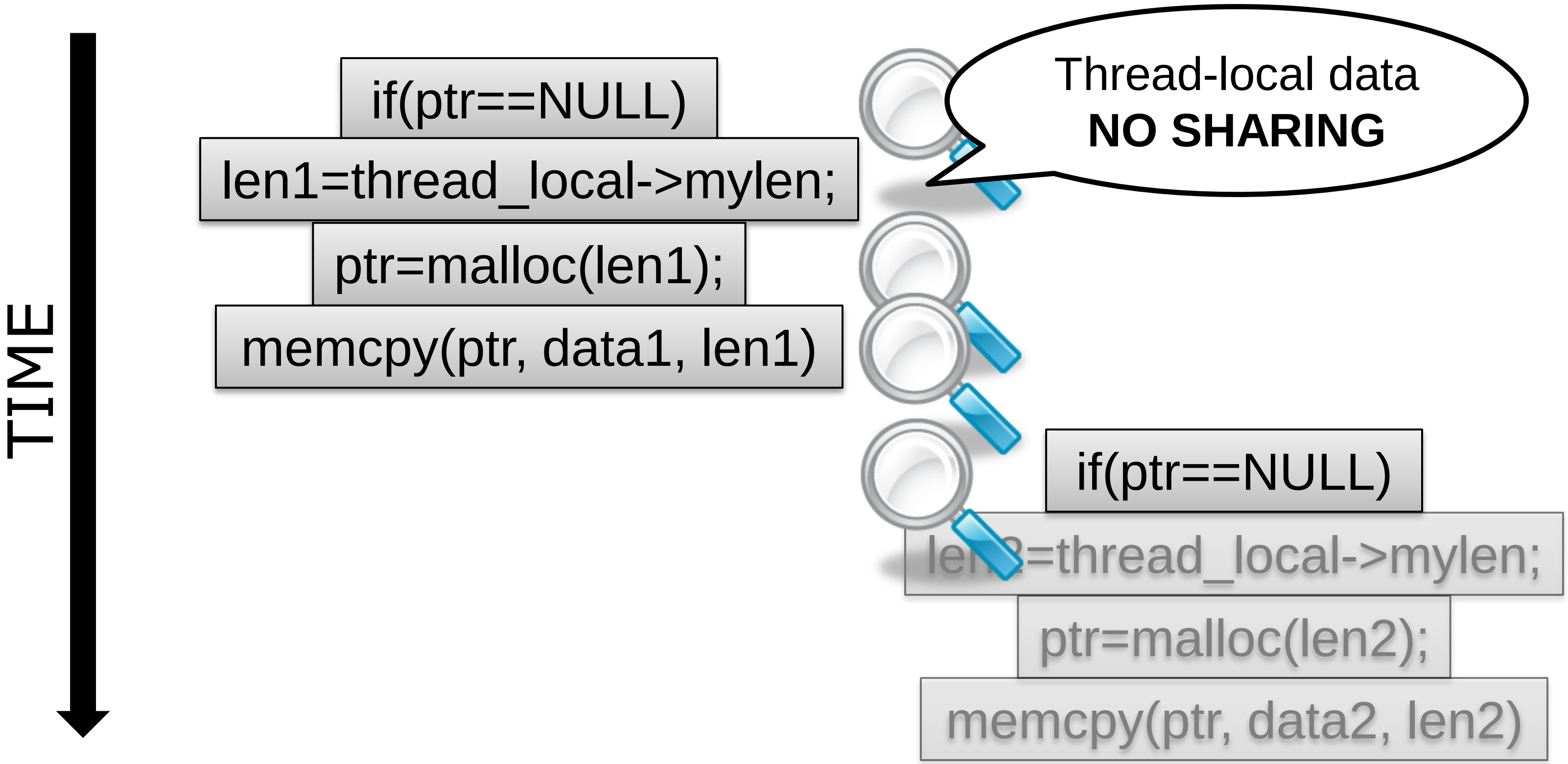
Inter-thread Sharing is What's Important

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



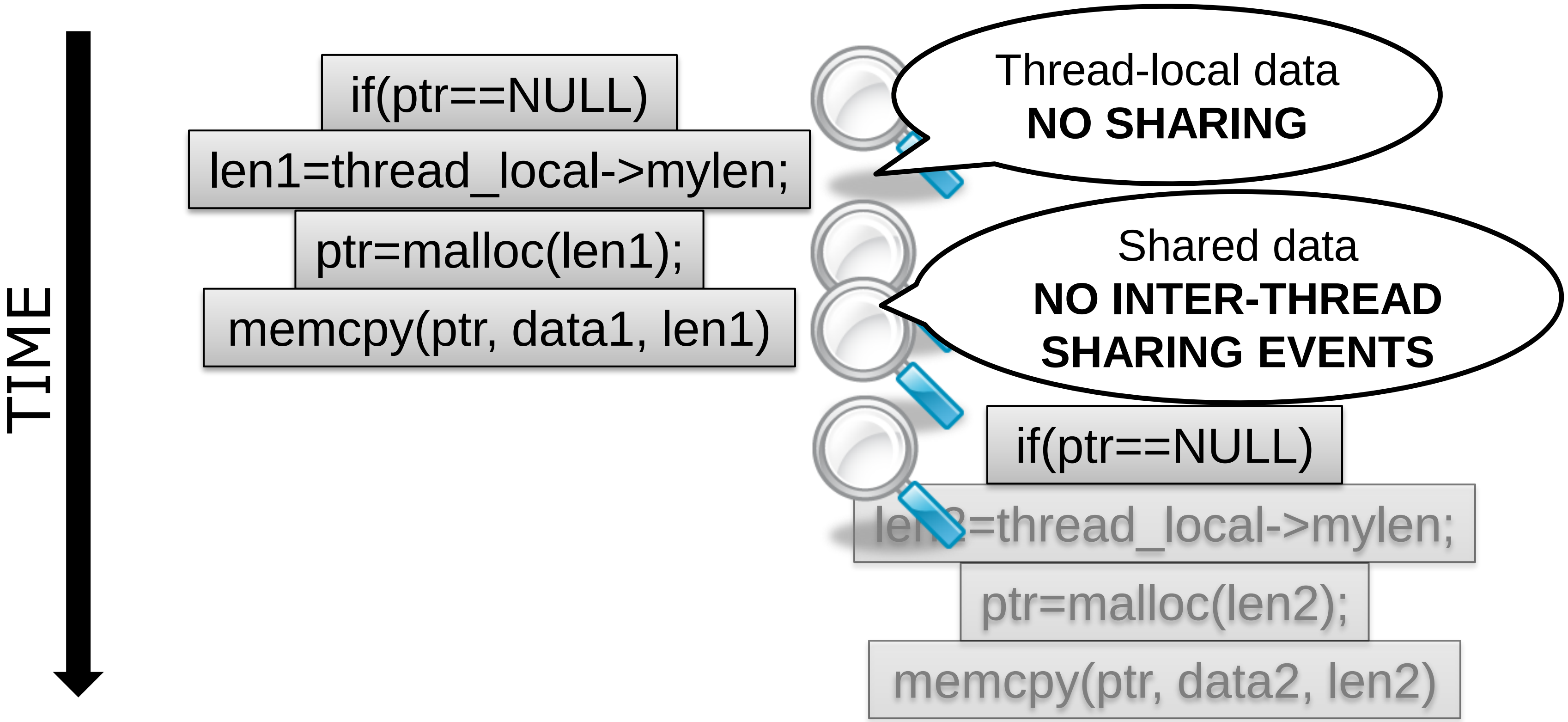
Inter-thread Sharing is What’s Important

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



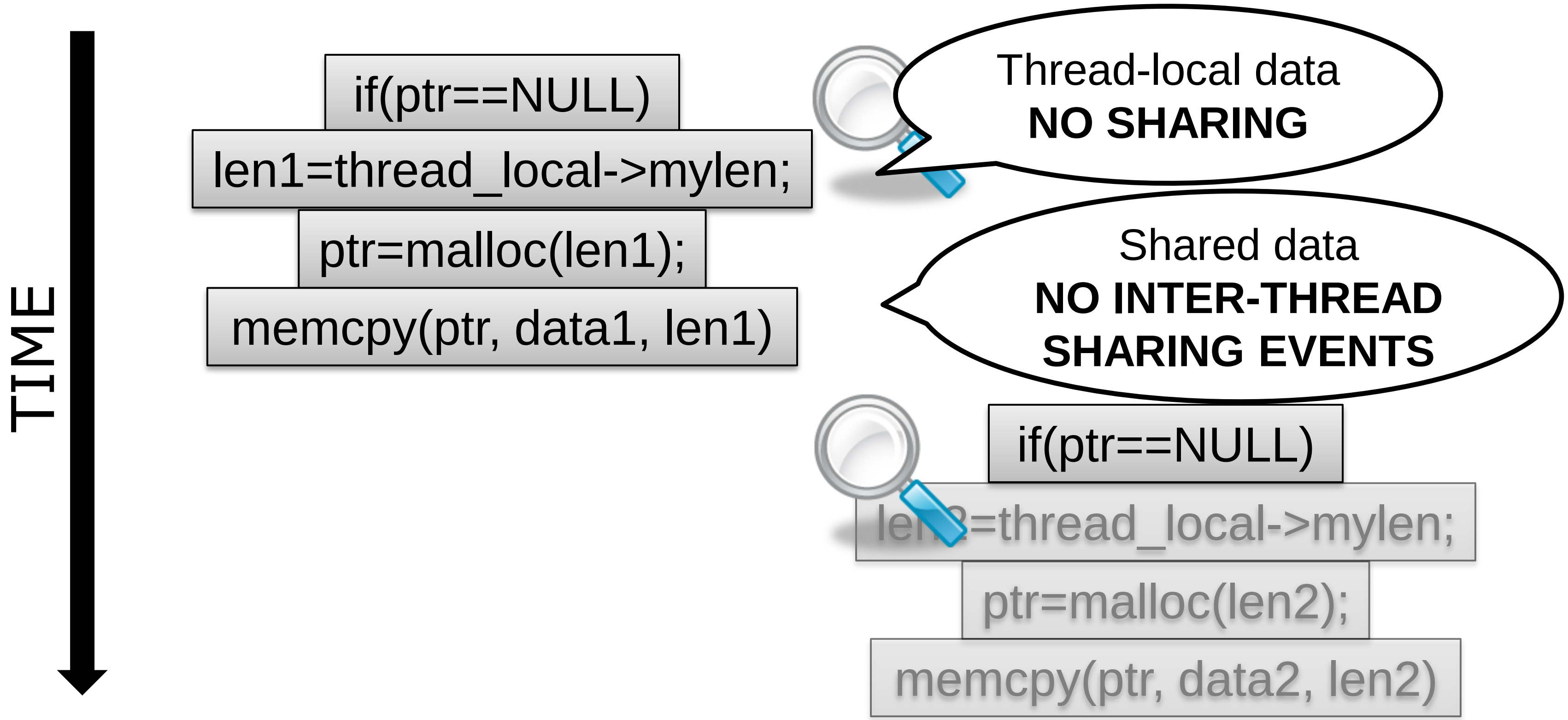
Inter-thread Sharing is What's Important

"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

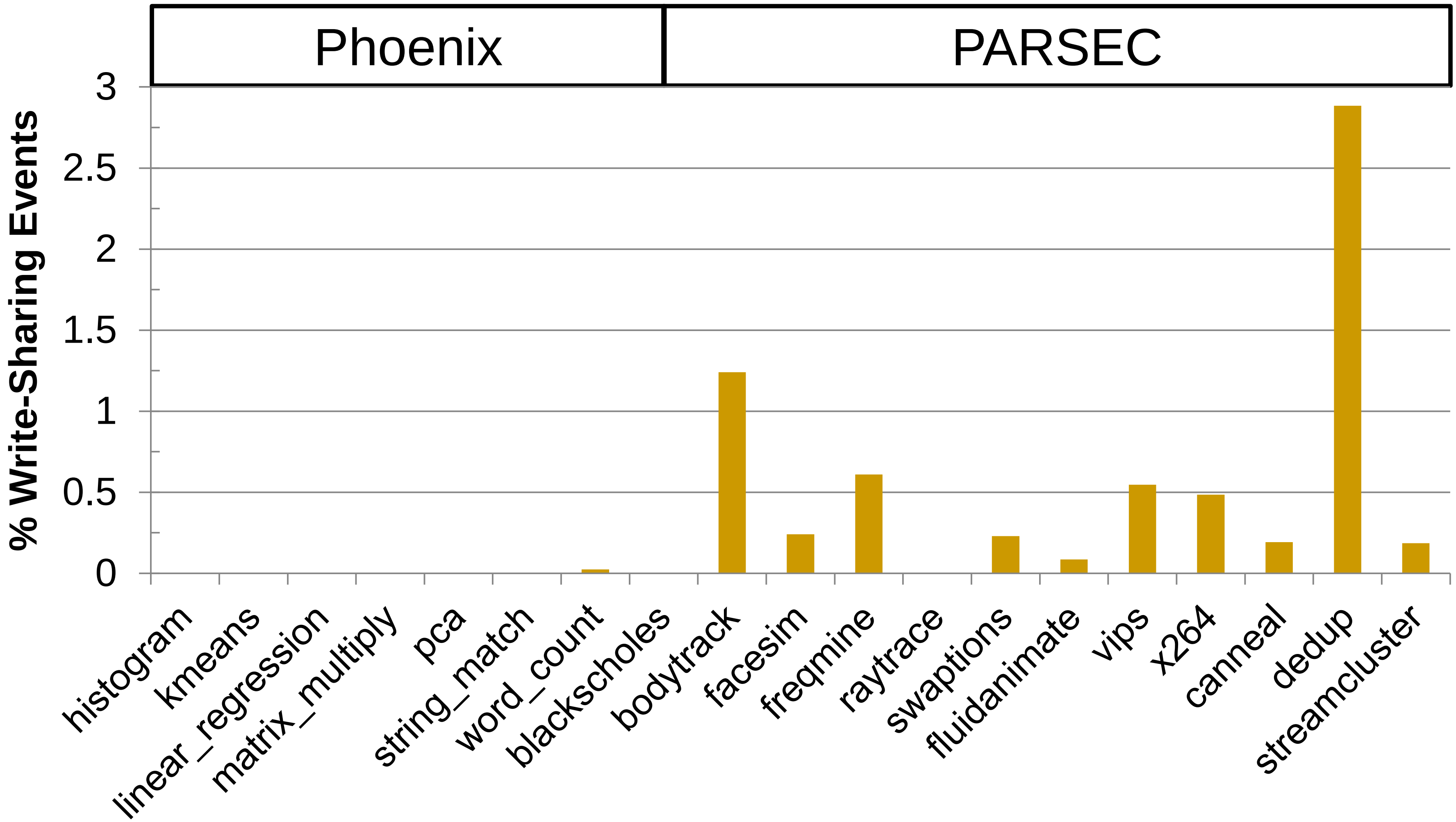
"Data races ... are failures in programs that **access and update shared data** in critical sections" – Netzer & Miller, 1992

TIME

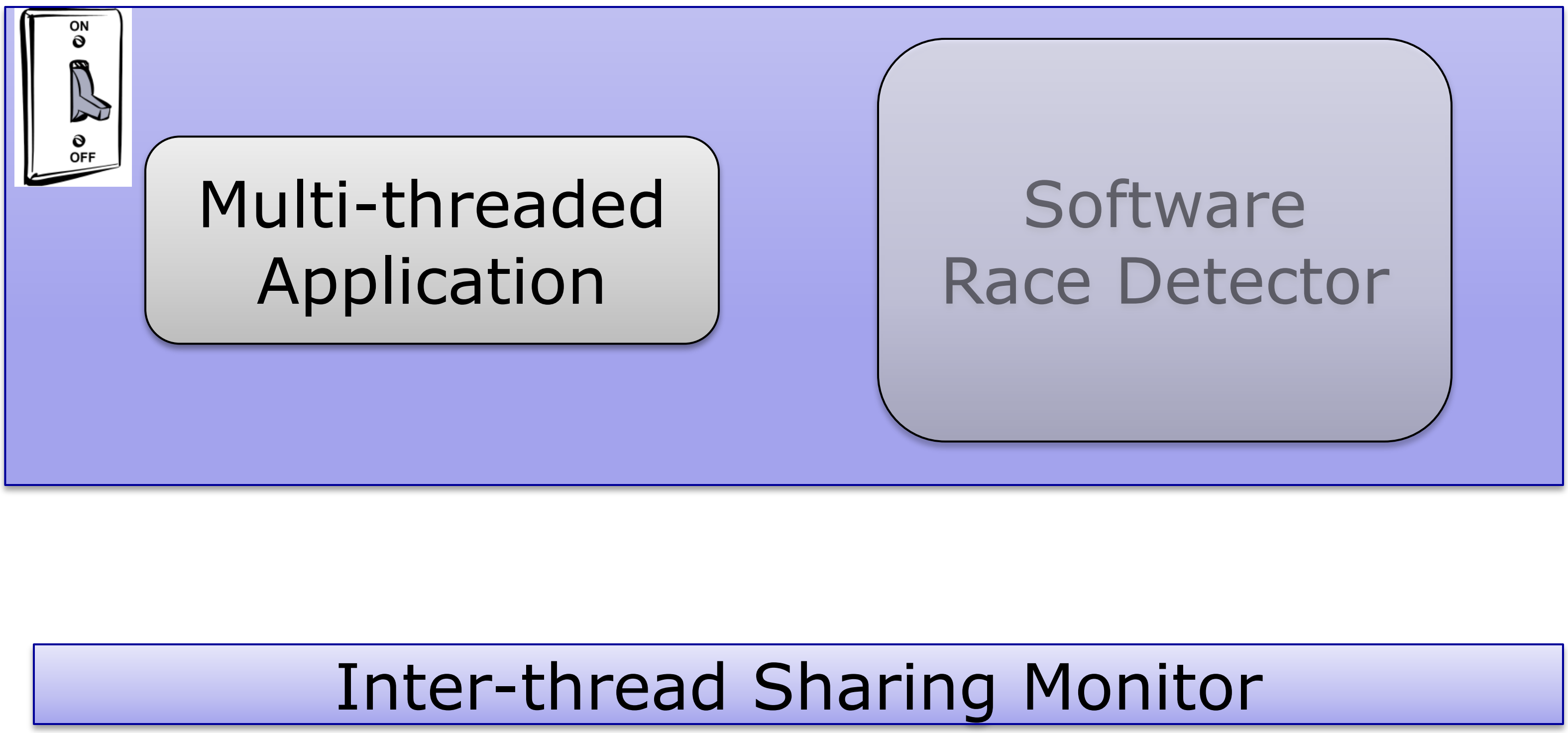
```
if(ptr==NULL)
len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)
```

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

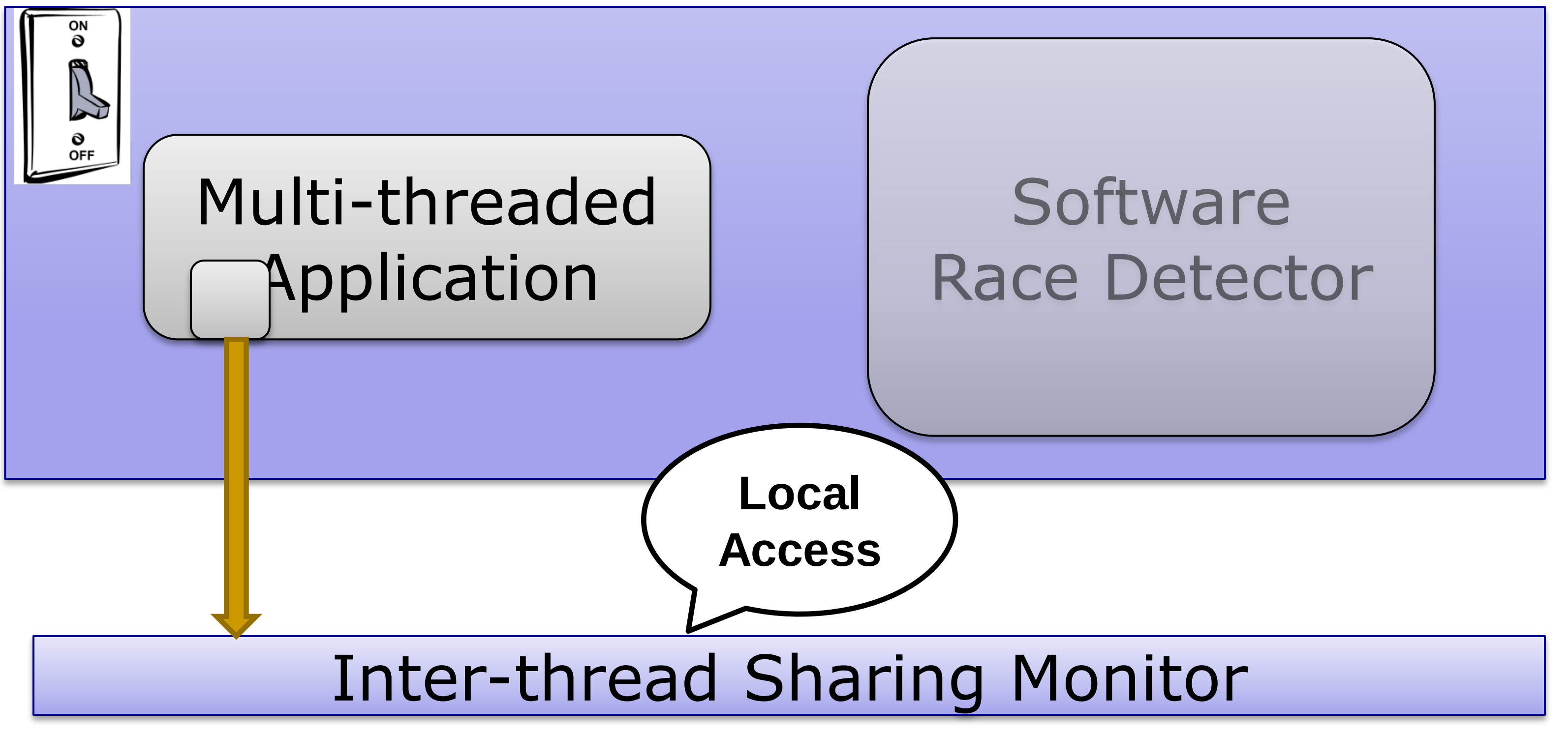
Very Little Inter-Thread Sharing



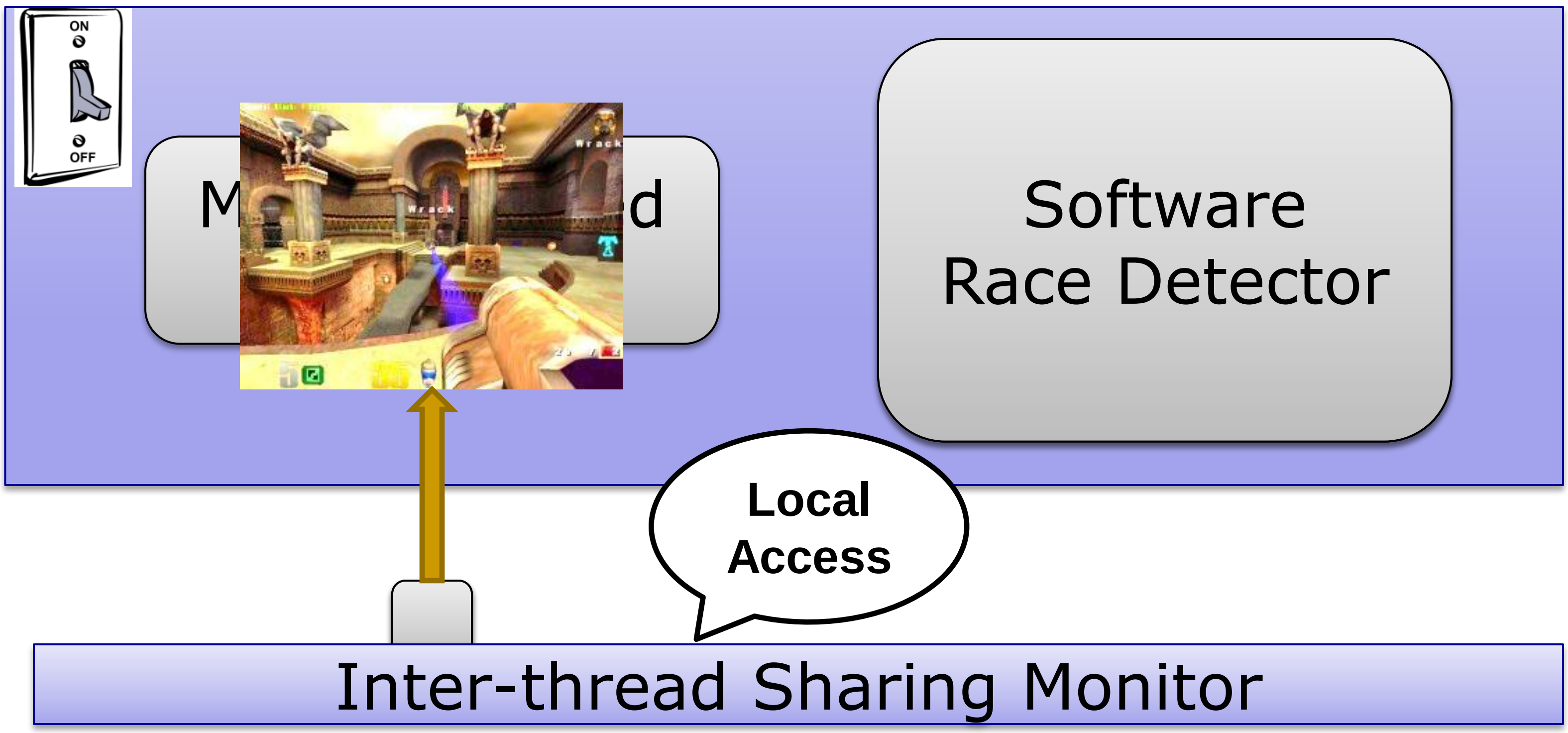
Use Demand-Driven Analysis!



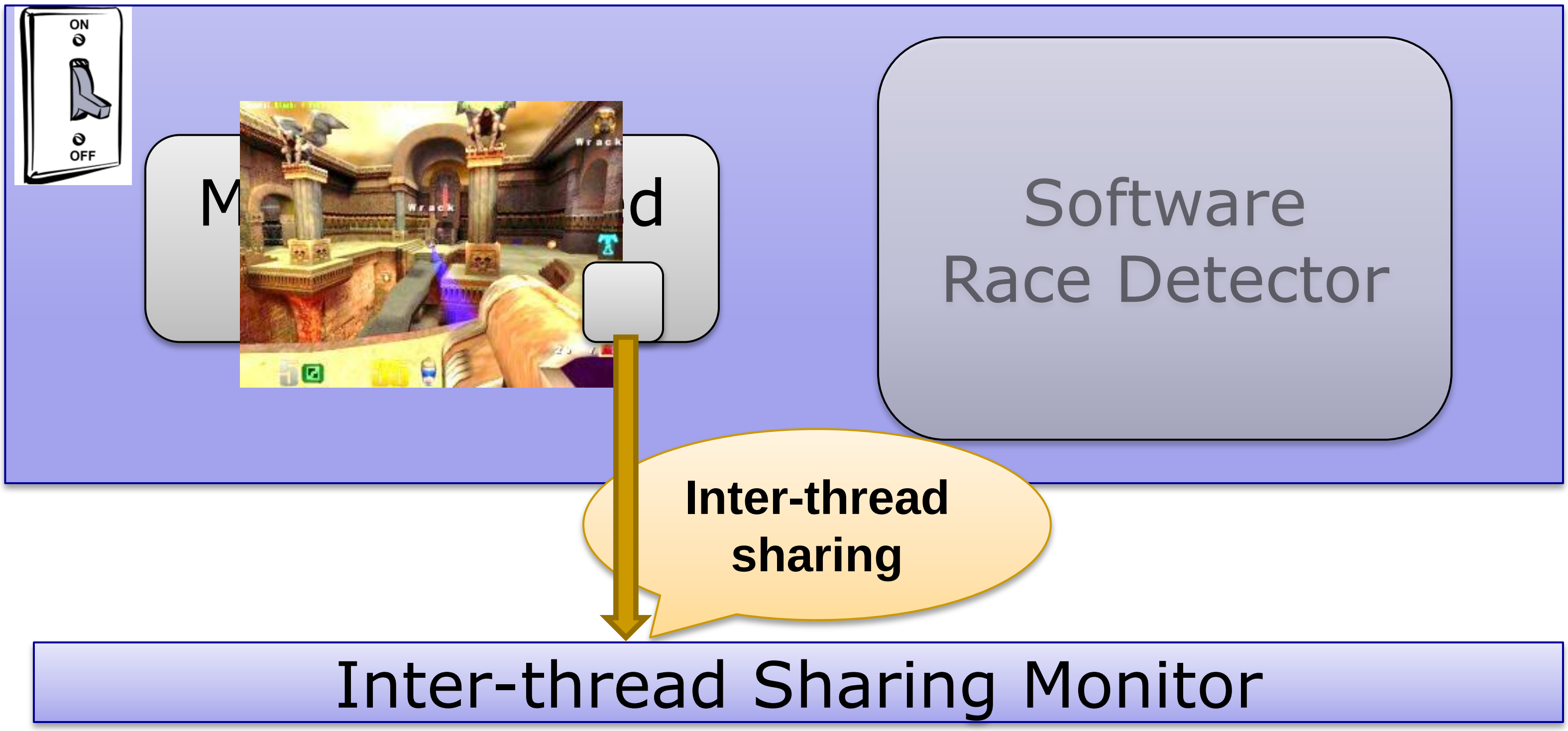
Use Demand-Driven Analysis!



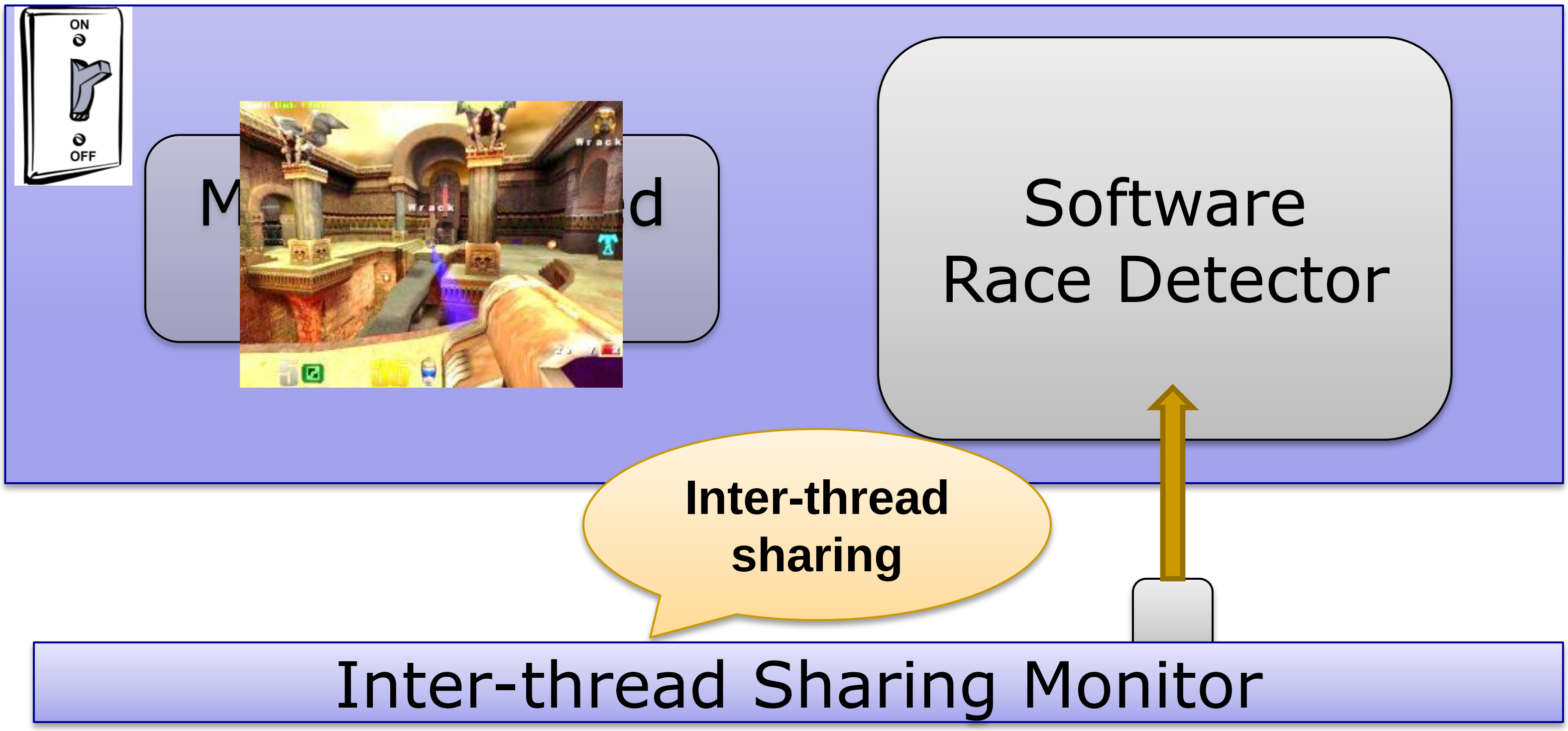
Use Demand-Driven Analysis!



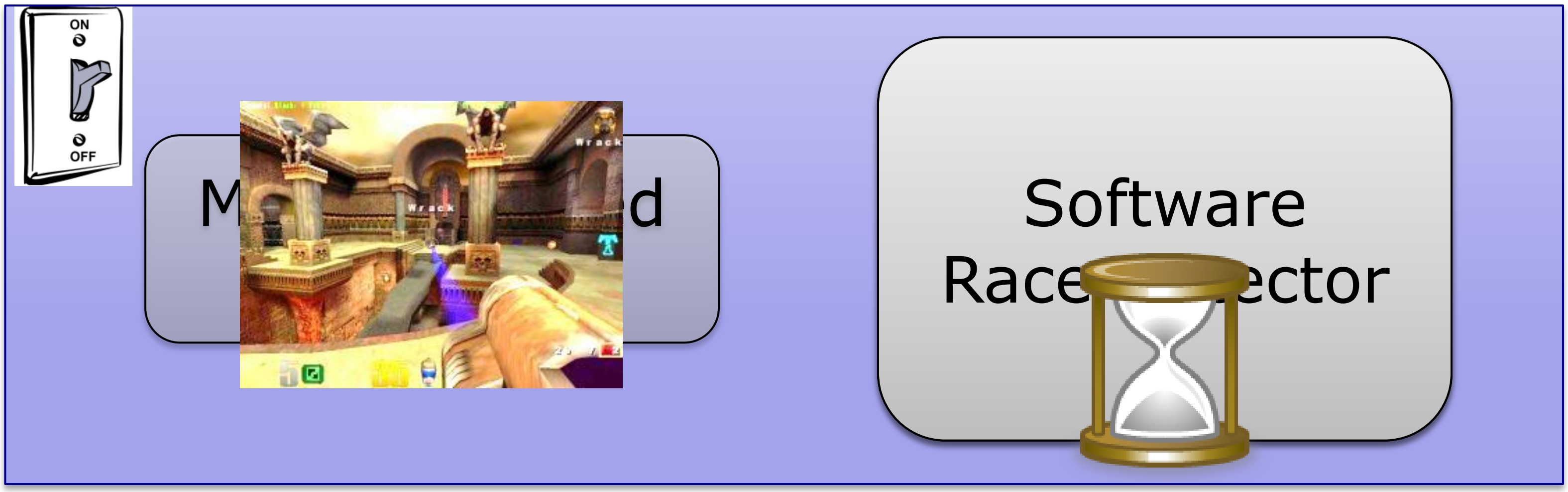
Use Demand-Driven Analysis!



Use Demand-Driven Analysis!

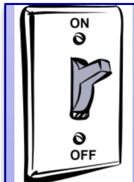


Use Demand-Driven Analysis!



Inter-thread Sharing Monitor

Use Demand-Driven Analysis!

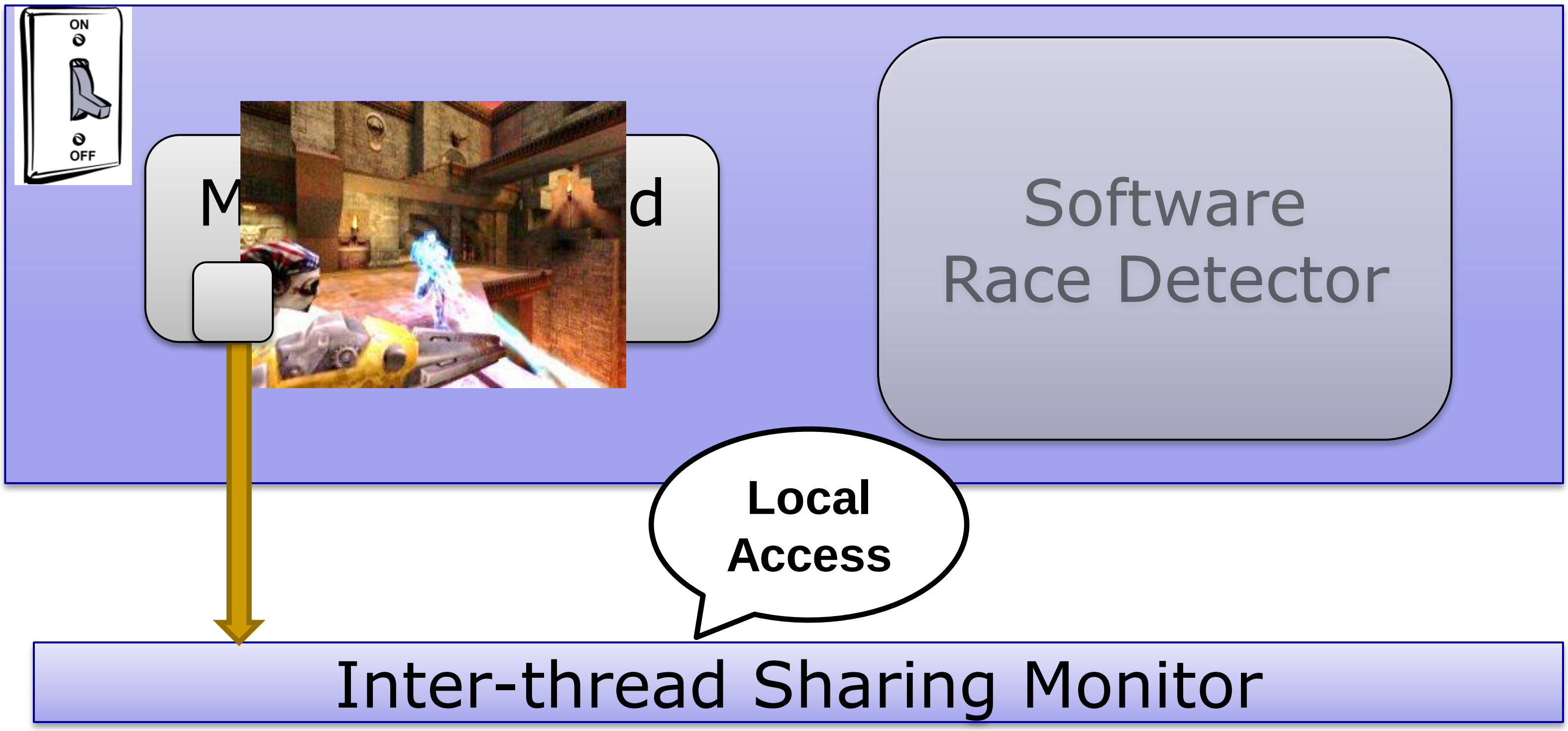


M d

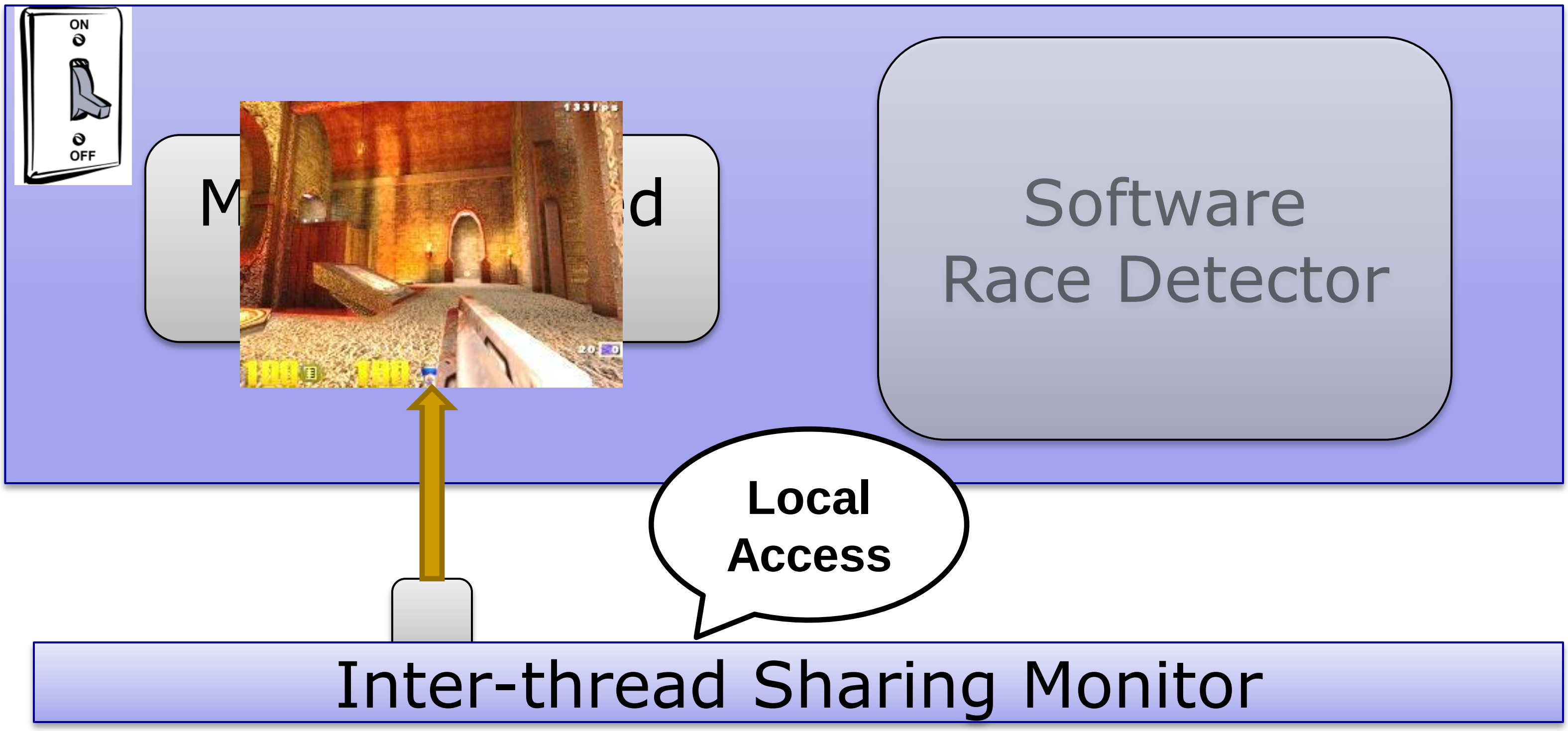
Software
Race Detector

Inter-thread Sharing Monitor

Use Demand-Driven Analysis!



Use Demand-Driven Analysis!

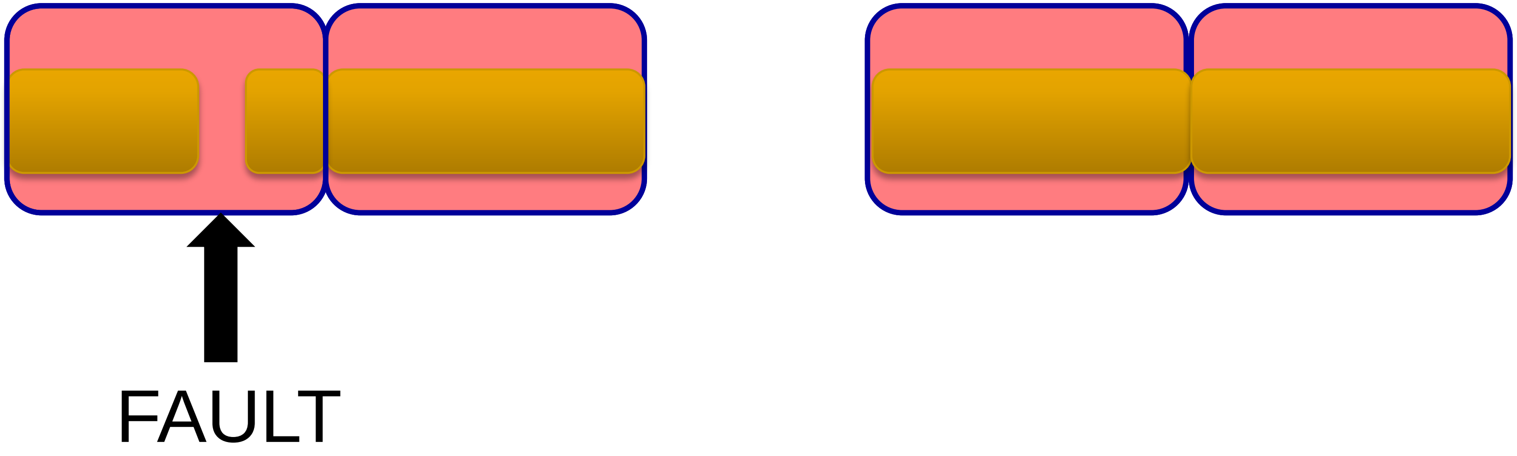


Finding Inter-thread Sharing

- Virtual Memory Watchpoints?

Finding Inter-thread Sharing

- Virtual Memory Watchpoints?

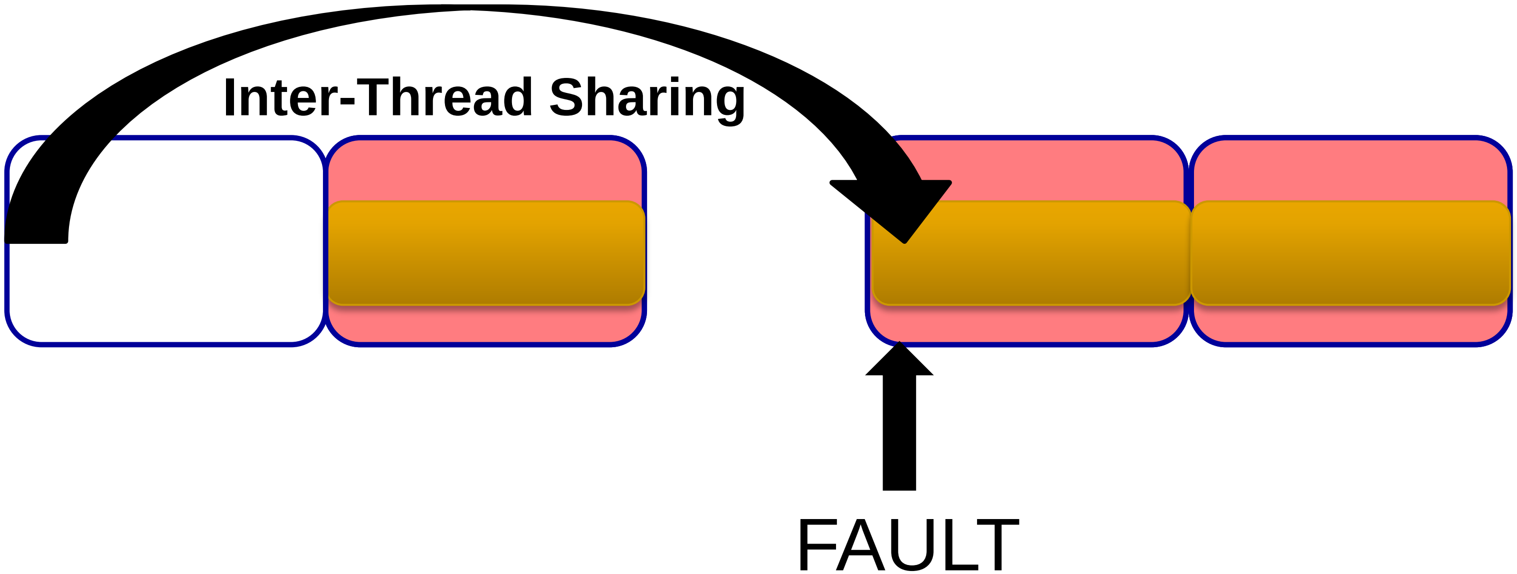


Finding Inter-thread Sharing

- Virtual Memory Watchpoints?

Finding Inter-thread Sharing

- Virtual Memory Watchpoints?



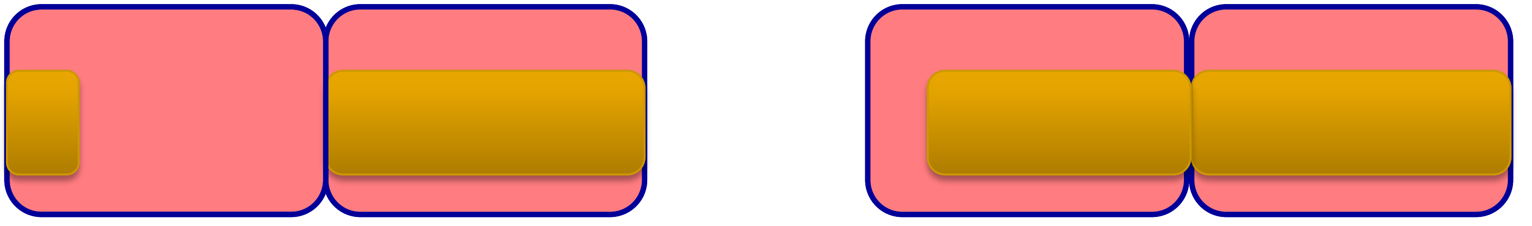
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?

  – ~100% of accesses cause page faults

Finding Inter-thread Sharing

- Virtual Memory Watchpoints?



  – ~100% of accesses cause page faults
- Granularity Gap

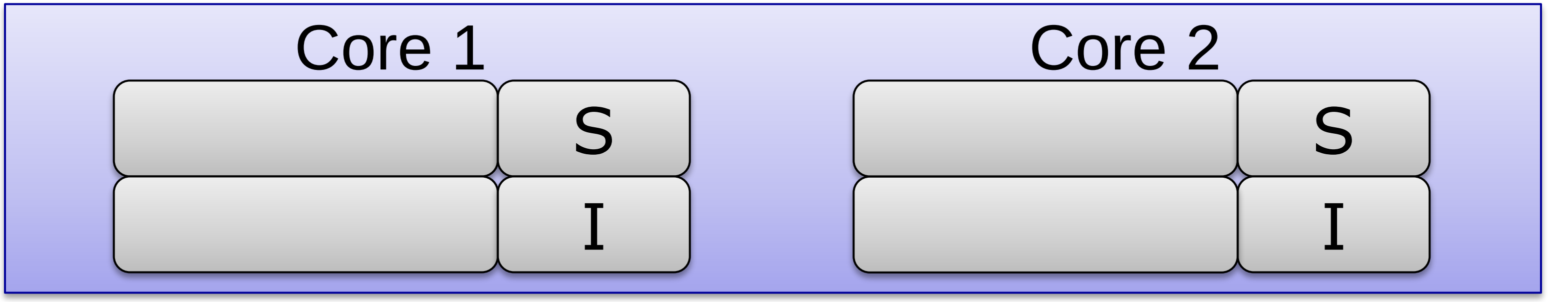
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?

  - ~100% of accesses cause page faults
- Granularity Gap
- Per-process not per-thread
- Must go through the kernel on faults
- Syscalls for setting/removing meta-data

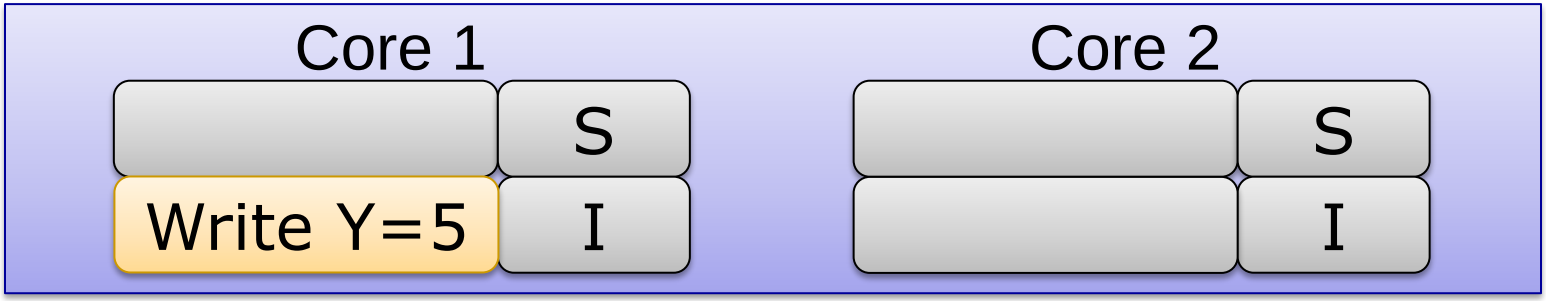
Hardware Sharing Detector

- HITM in Cache: W→R Data Sharing



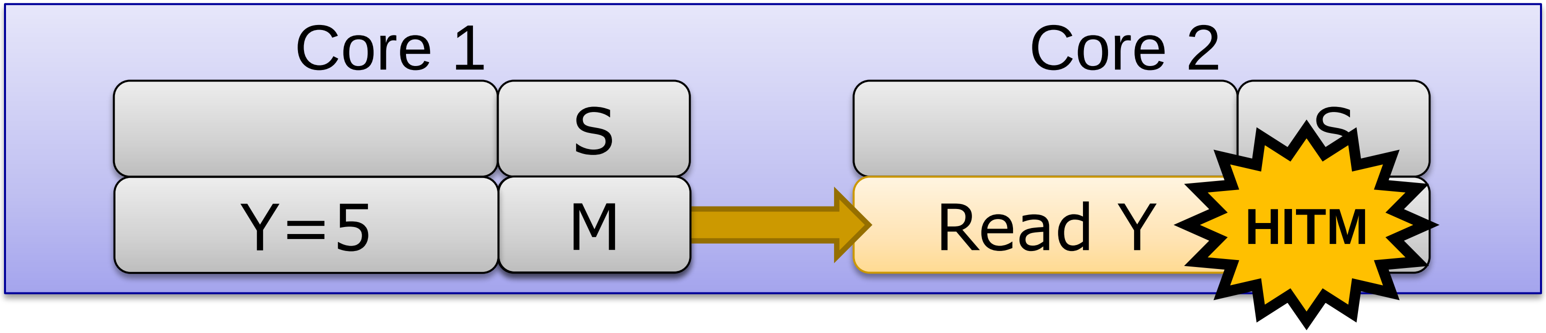
Hardware Sharing Detector

- HITM in Cache: W→R Data Sharing



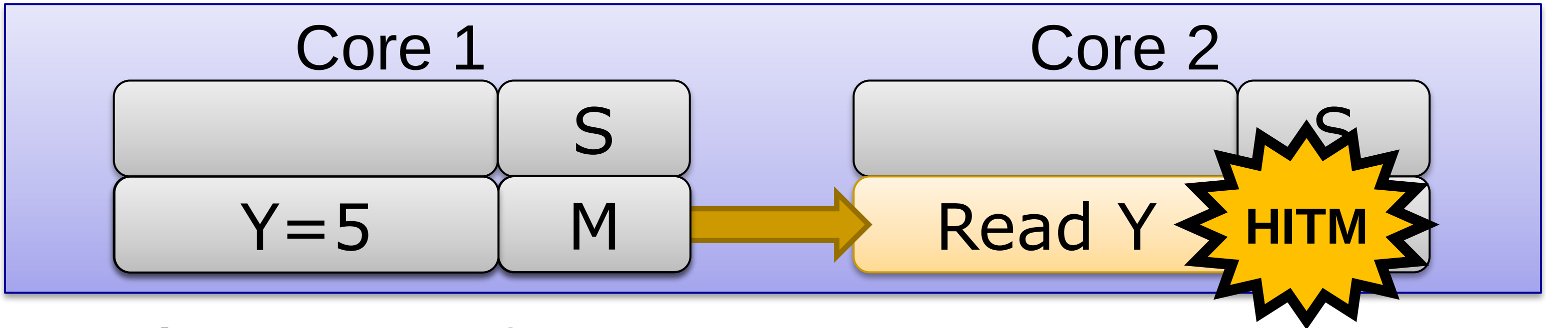
Hardware Sharing Detector

- HITM in Cache: W→R Data Sharing

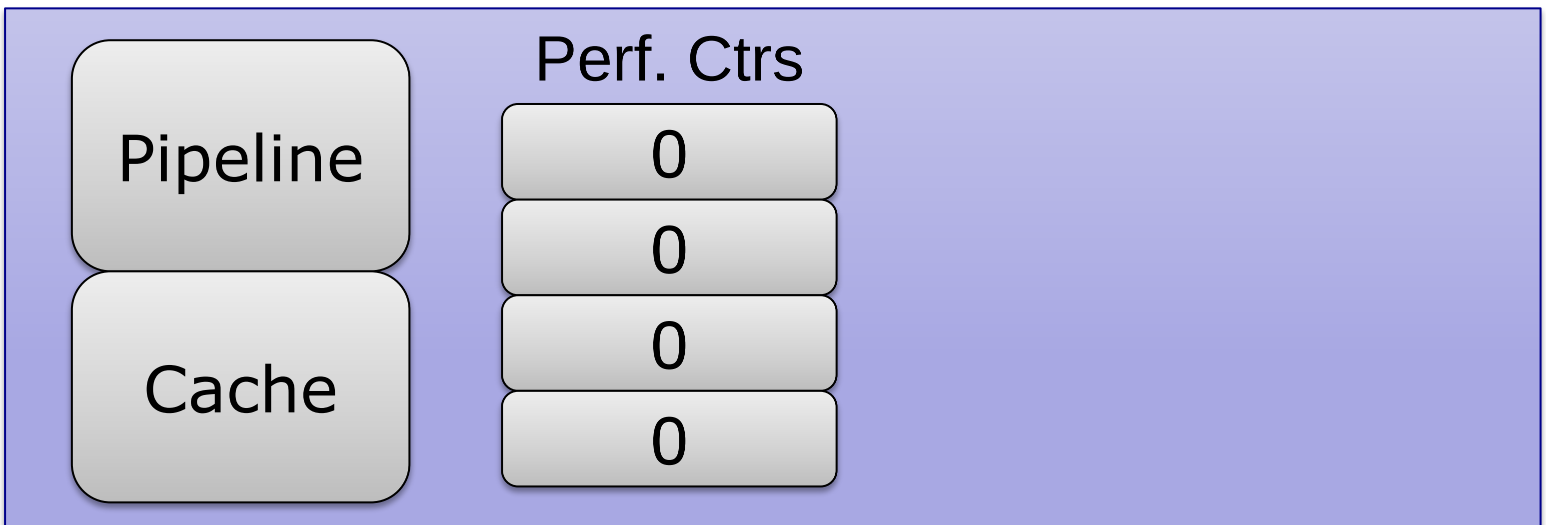


Hardware Sharing Detector

- HITM in Cache: W→R Data Sharing

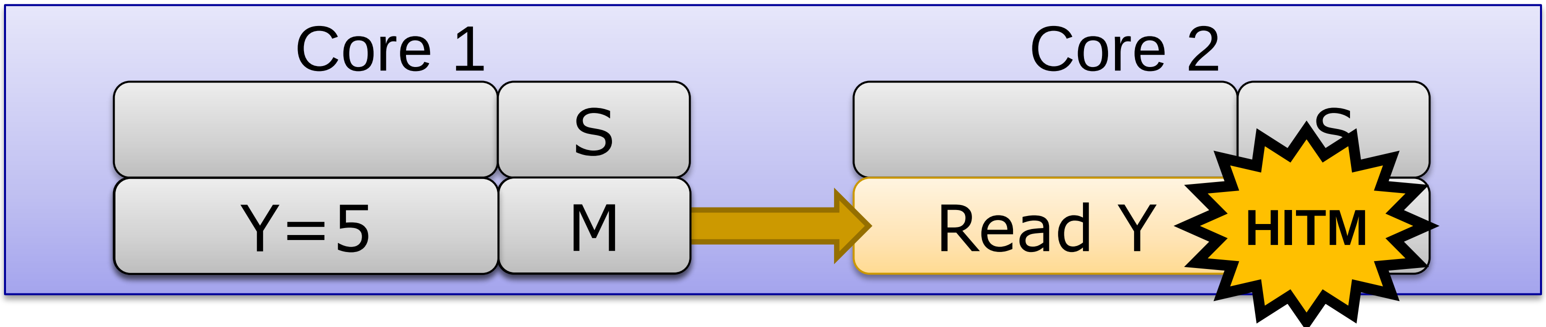


- Hardware Performance Counters

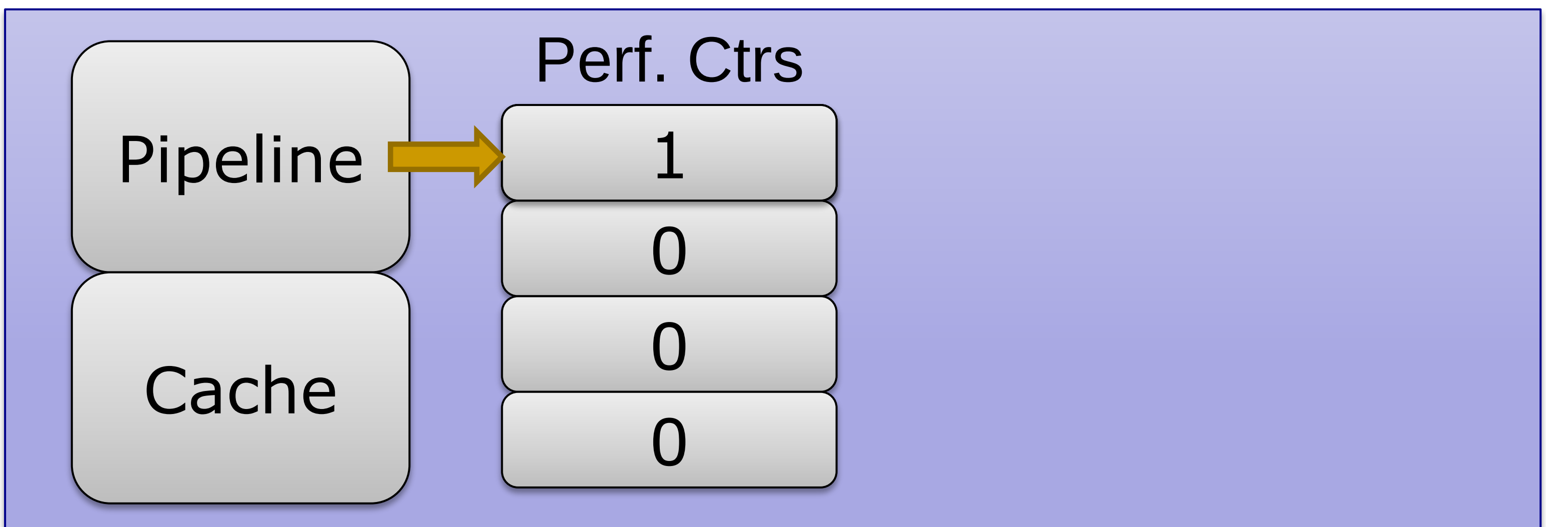


Hardware Sharing Detector

- HITM in Cache: W→R Data Sharing

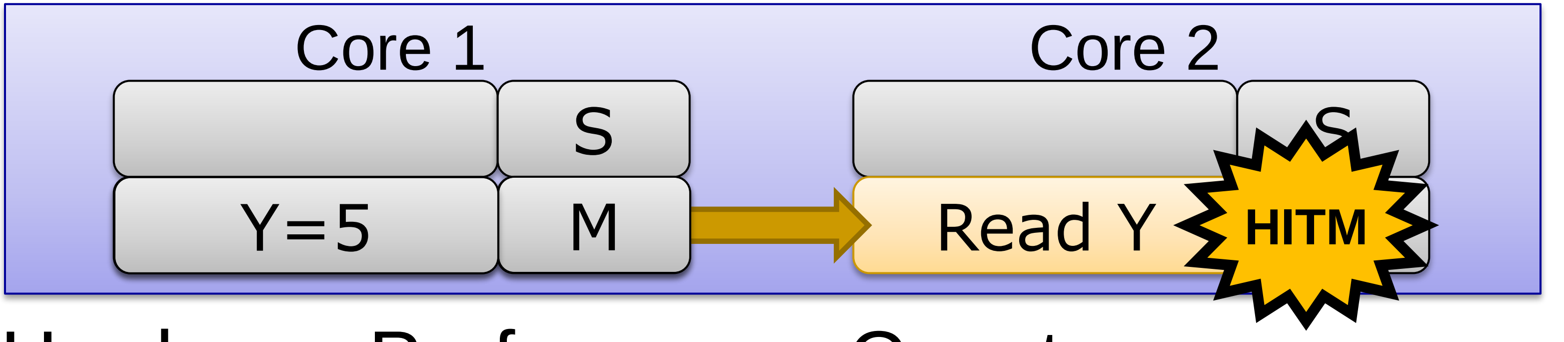


- Hardware Performance Counters

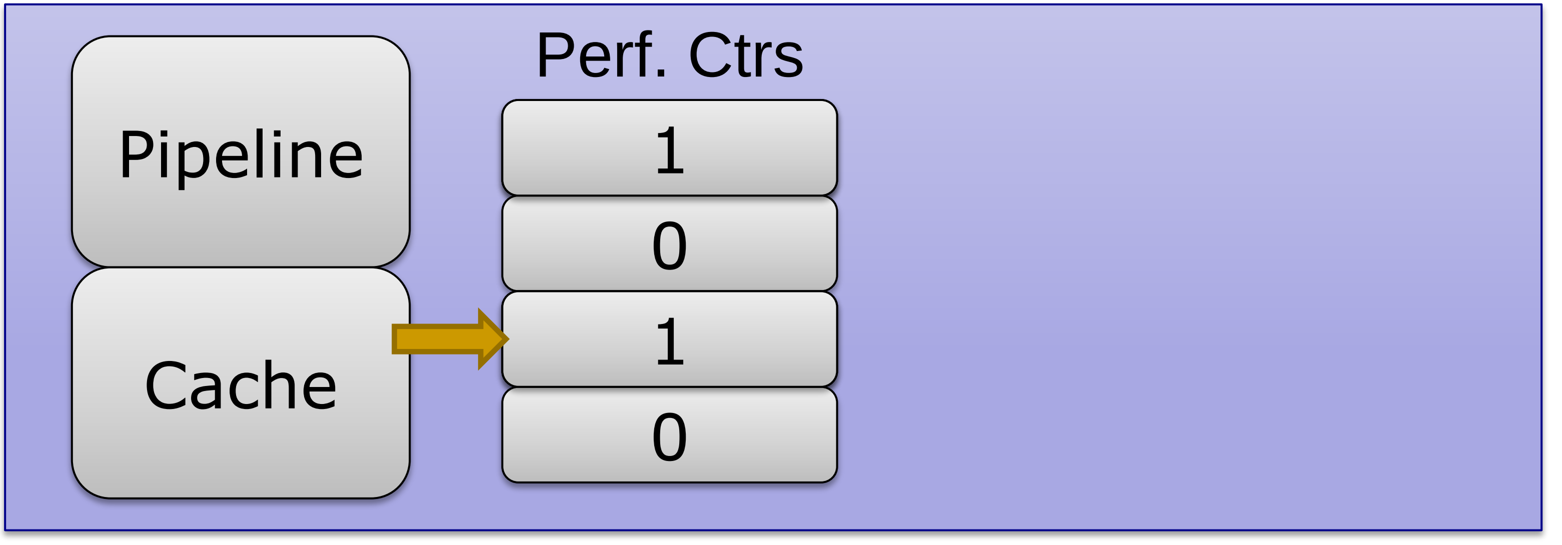


Hardware Sharing Detector

- HITM in Cache: W→R Data Sharing

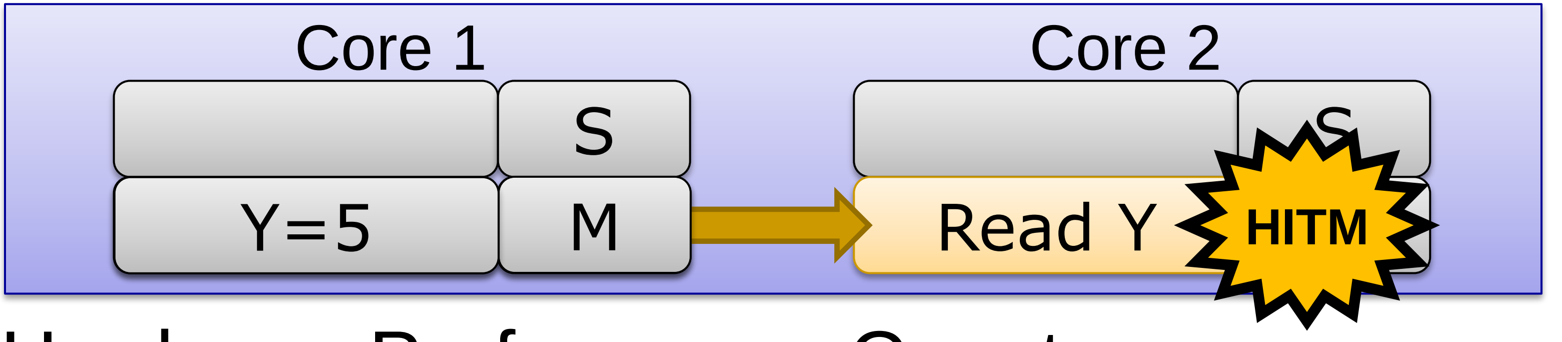


- Hardware Performance Counters

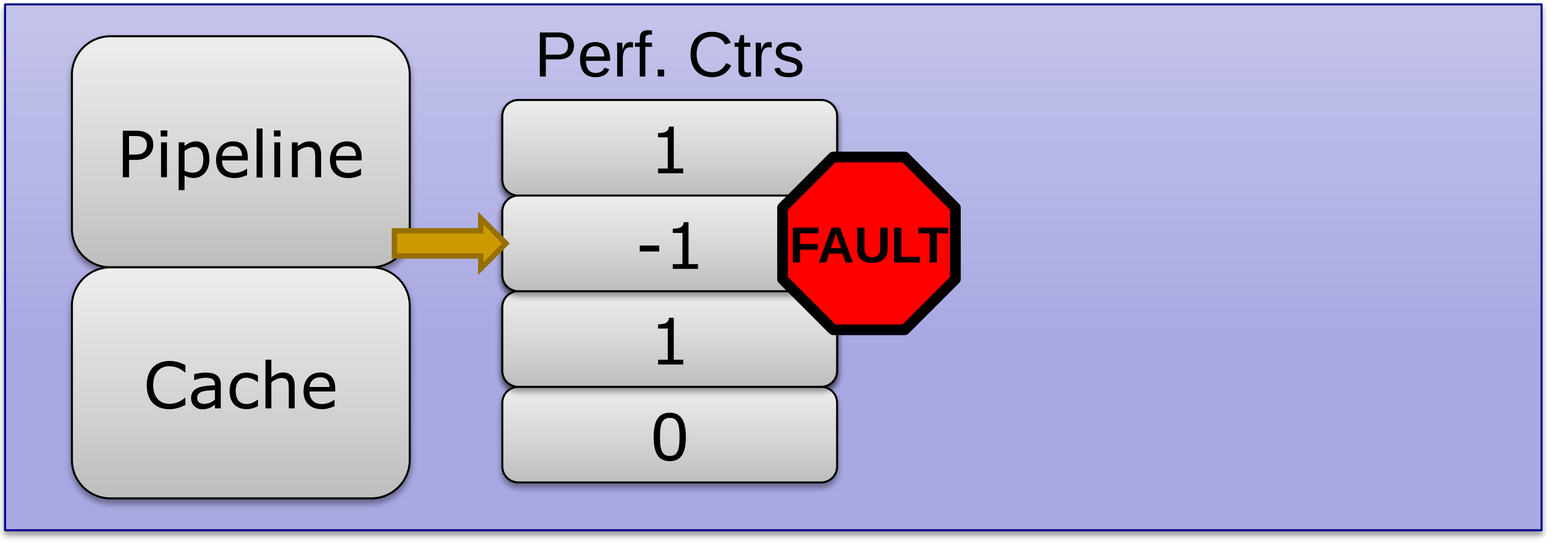


Hardware Sharing Detector

- HITM in Cache: W→R Data Sharing

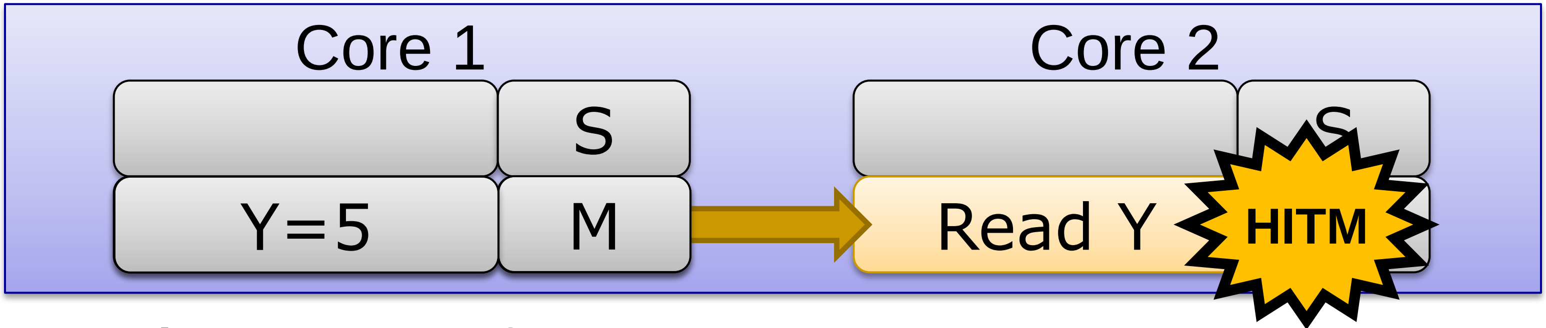


- Hardware Performance Counters

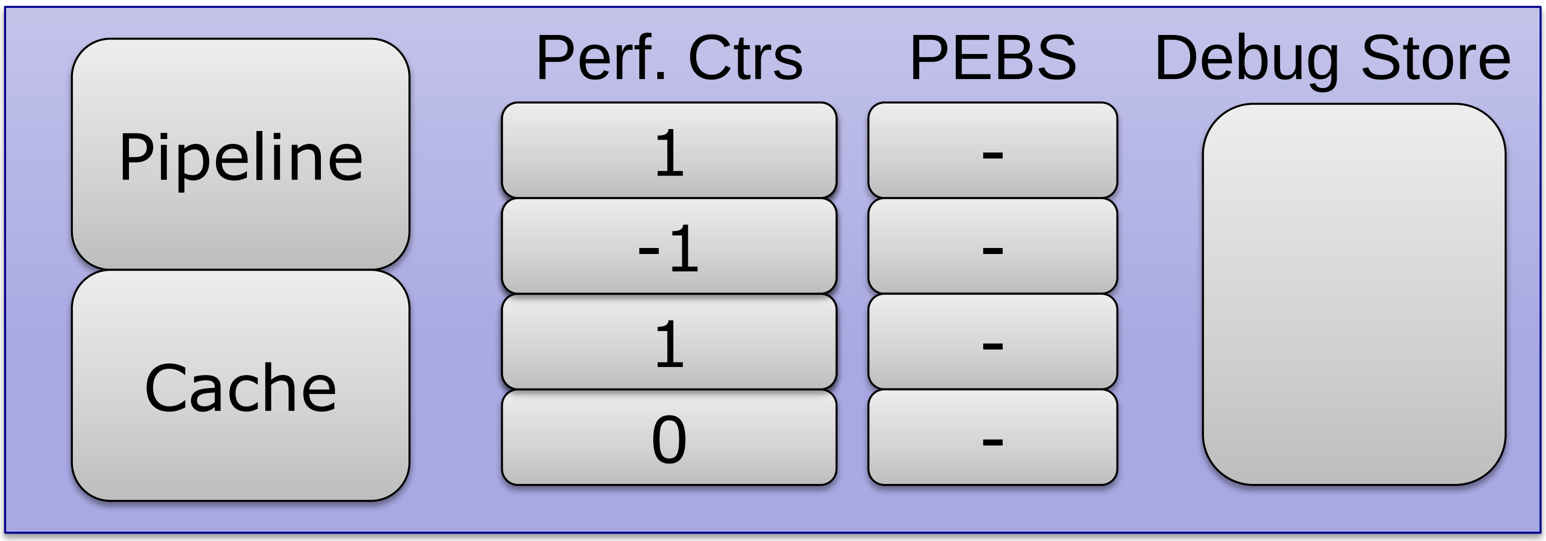


Hardware Sharing Detector

- HITM in Cache: W→R Data Sharing

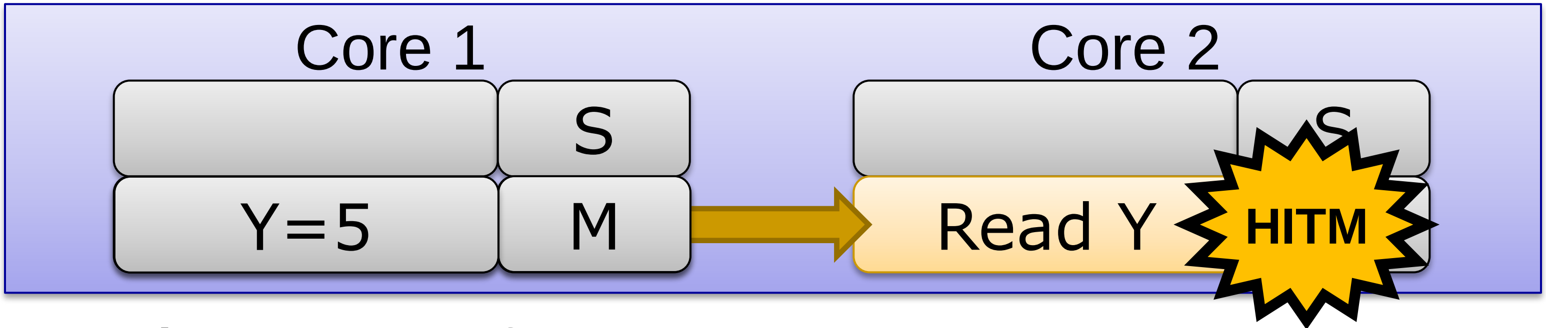


- Hardware Performance Counters

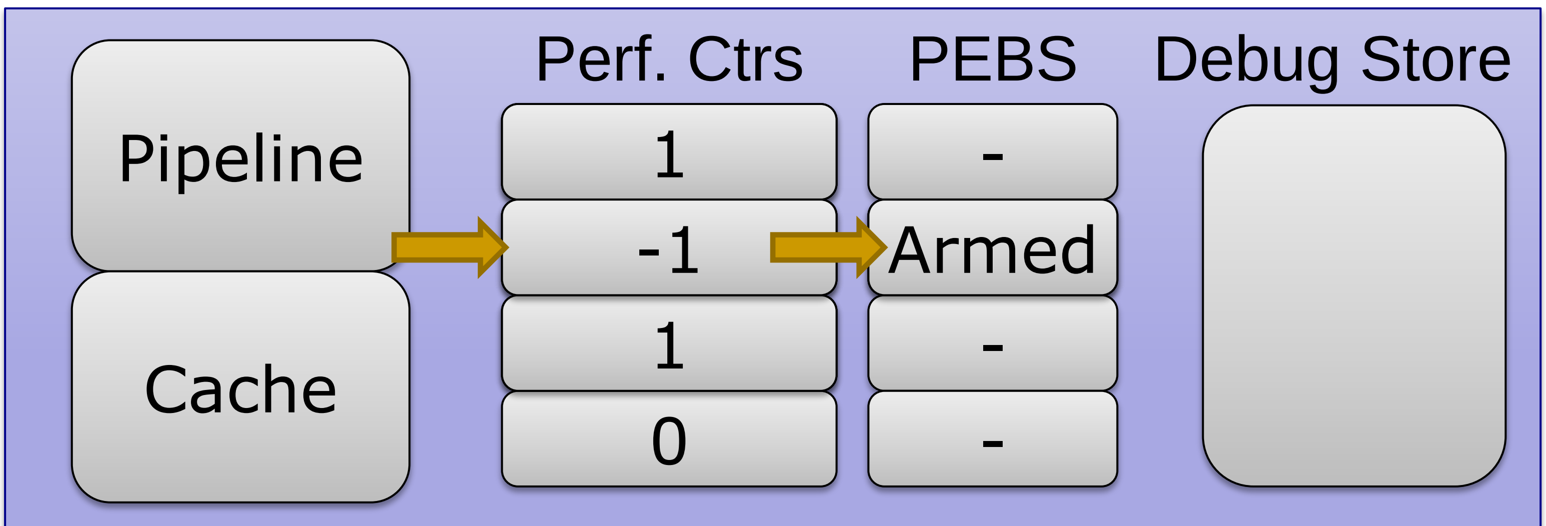


Hardware Sharing Detector

- HITM in Cache: W→R Data Sharing

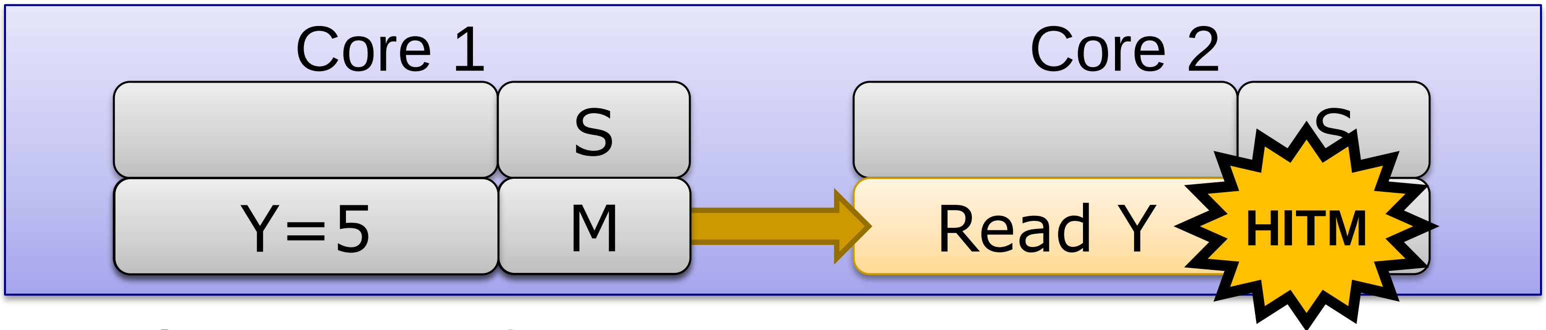


- Hardware Performance Counters

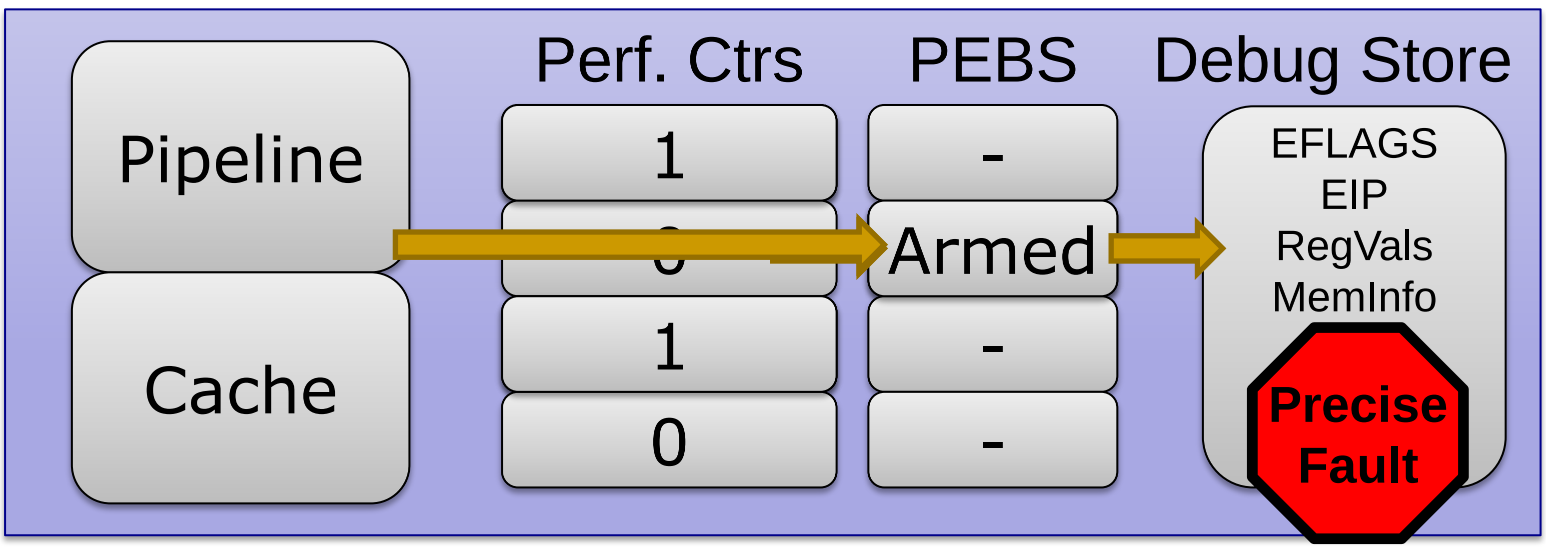


Hardware Sharing Detector

- HITM in Cache: W→R Data Sharing



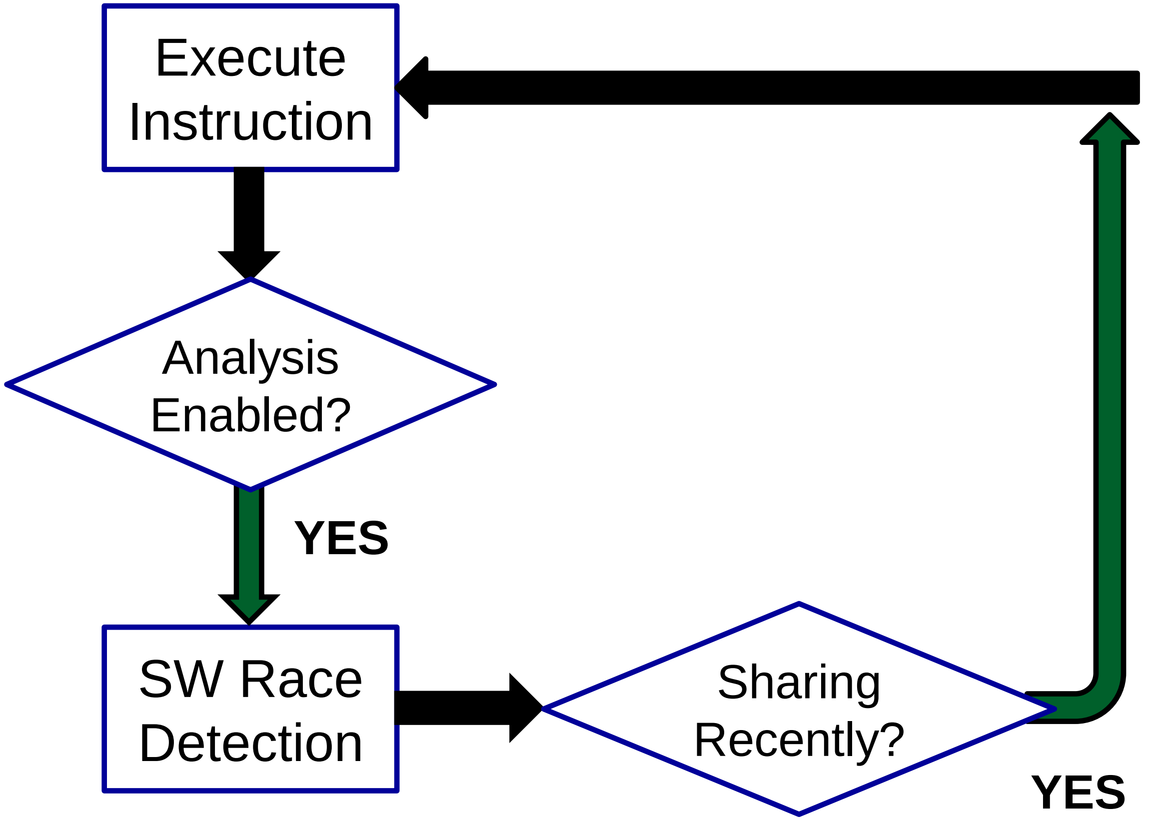
- Hardware Performance Counters



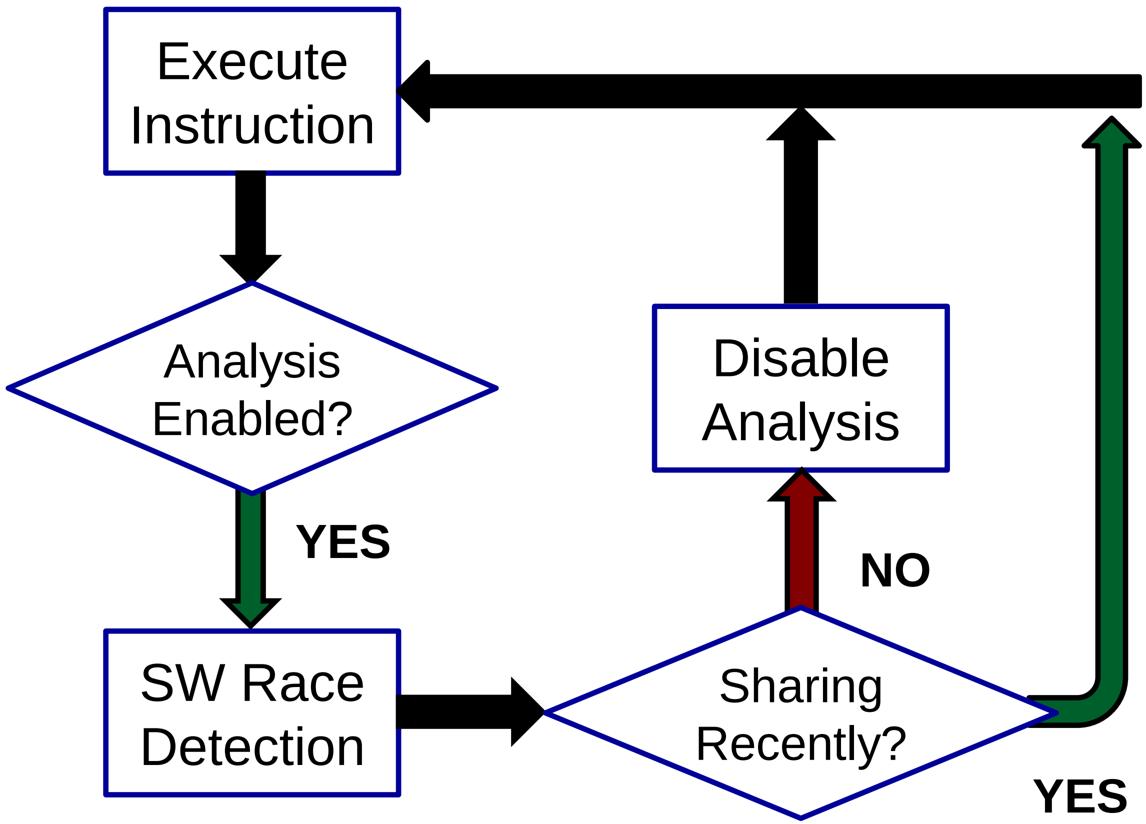
Potential Accuracy & Perf. Problems

- Limitations of Performance Counters
  - HITM only finds W→R Data Sharing
  - Hardware prefetcher events aren’t counted
- Limitations of Cache Events
  - SMT sharing can’t be counted
  - Cache eviction causes missed events
  - False sharing, etc…
- PEBS events still go through the kernel

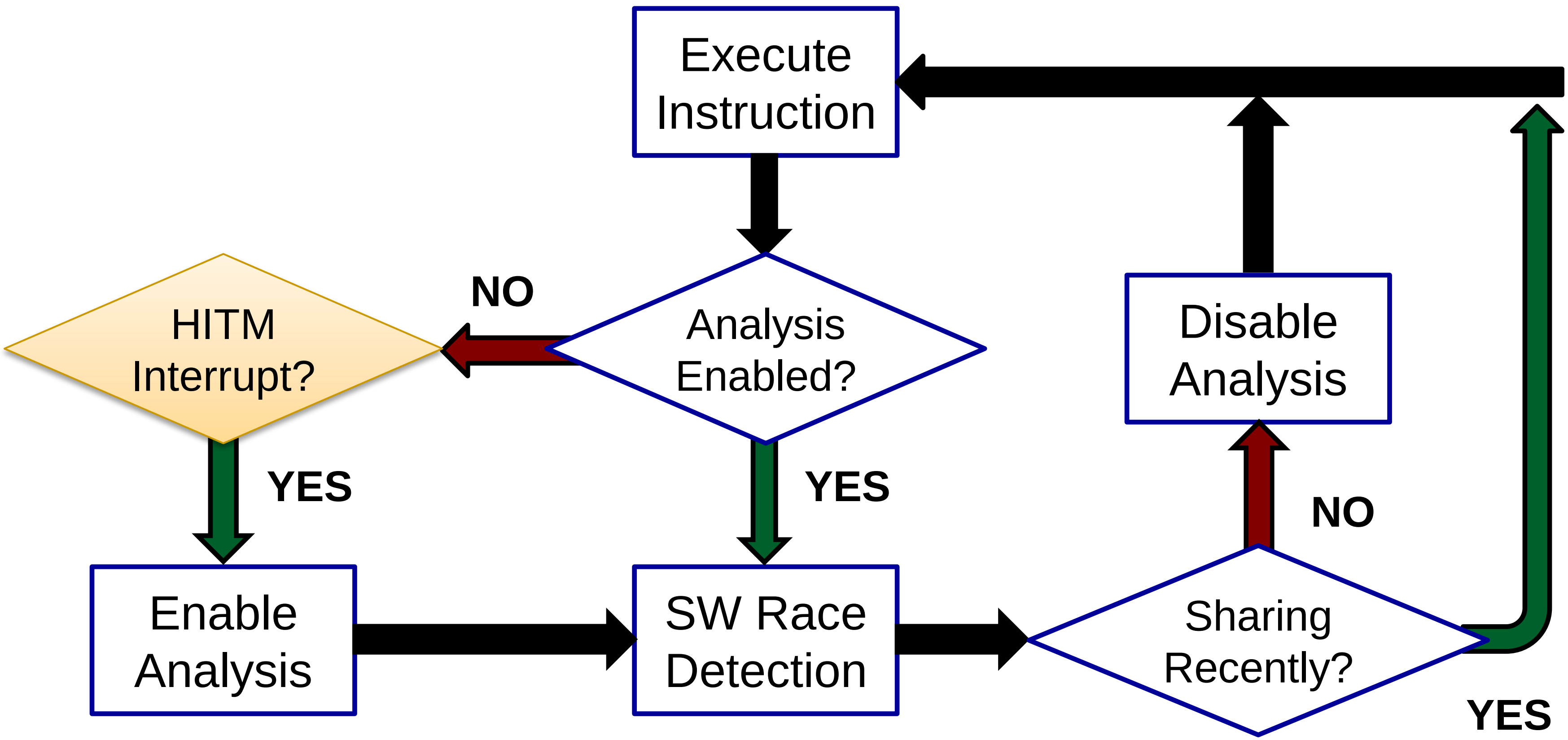
On-Demand Analysis on Real HW



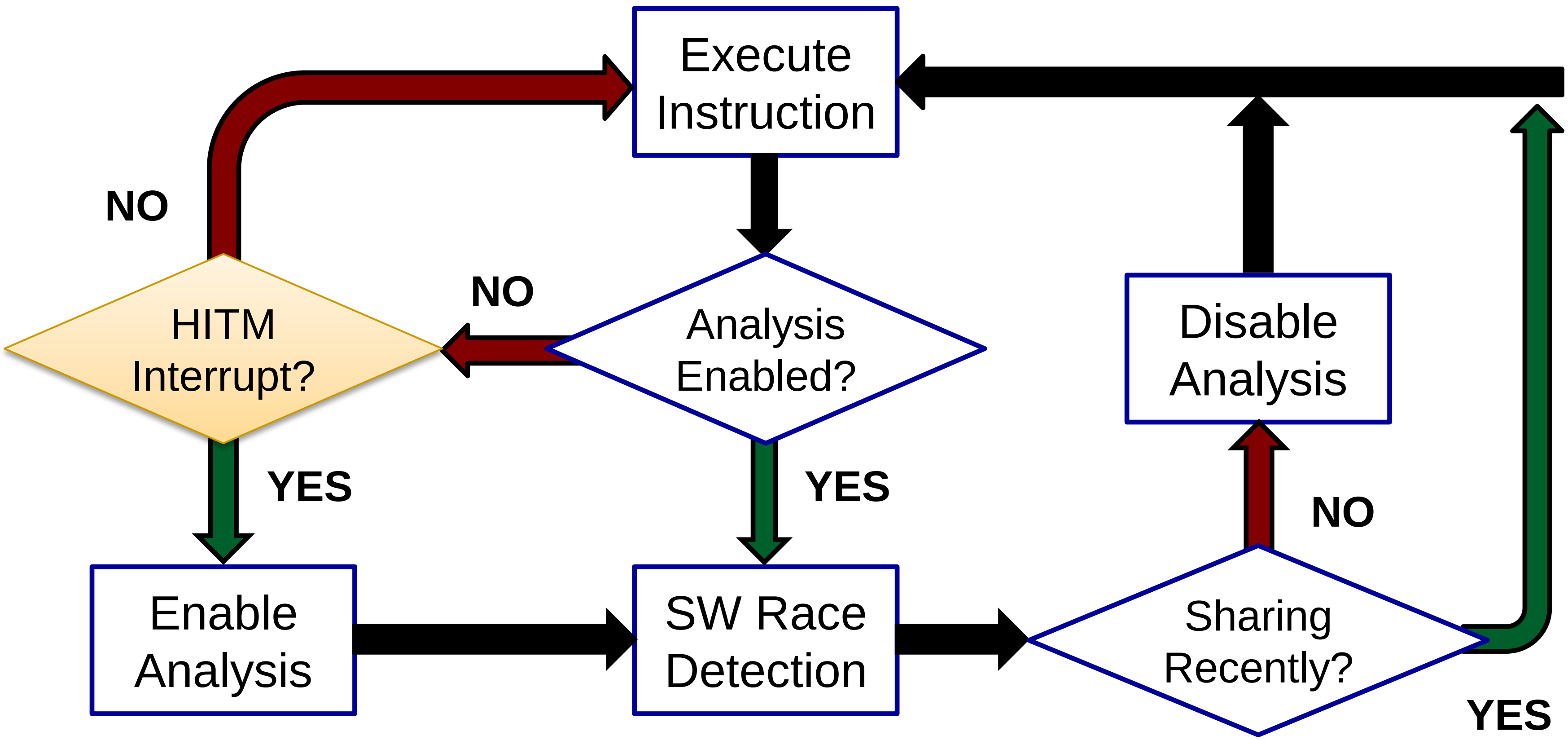
On-Demand Analysis on Real HW



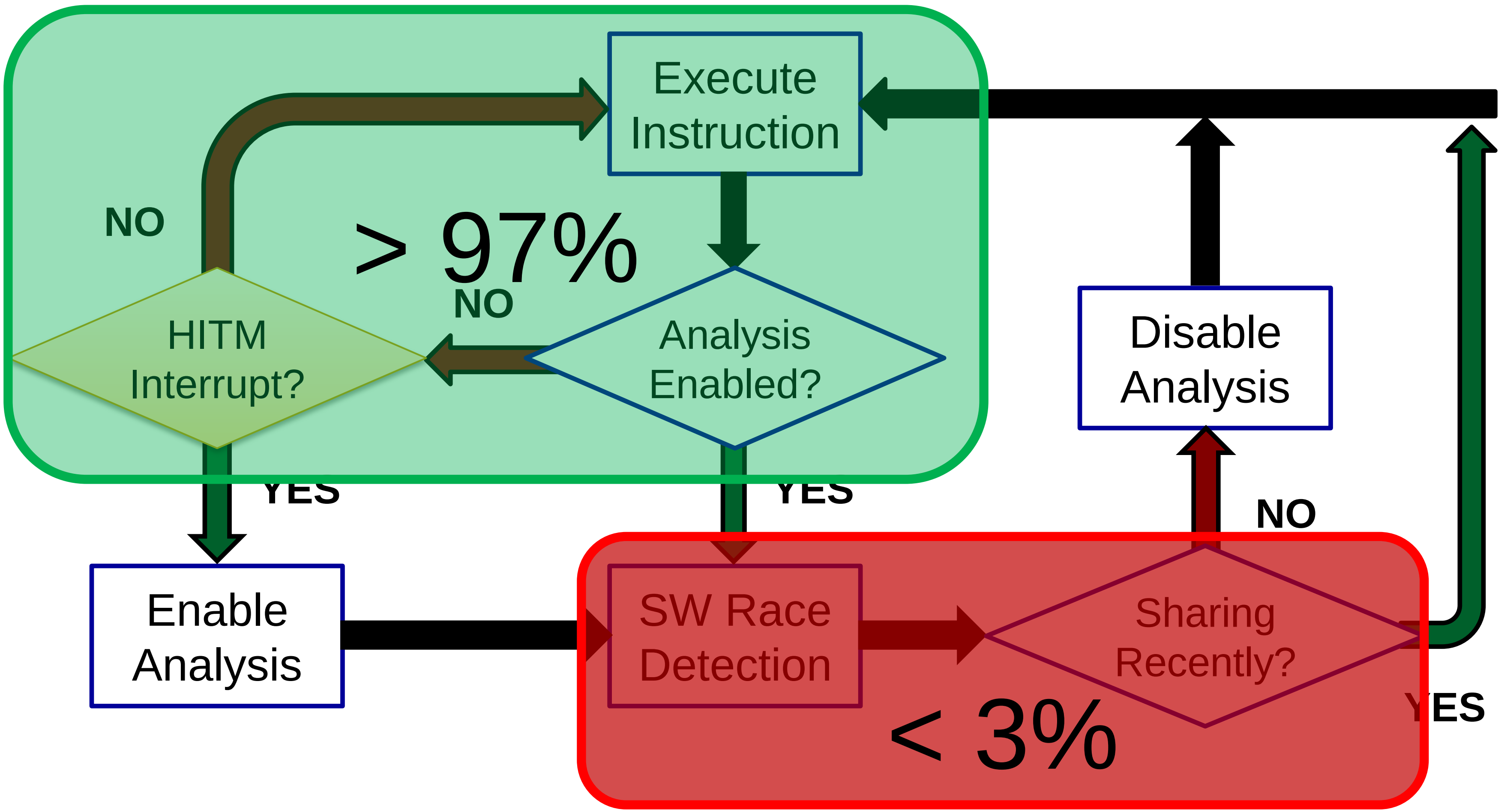
On-Demand Analysis on Real HW



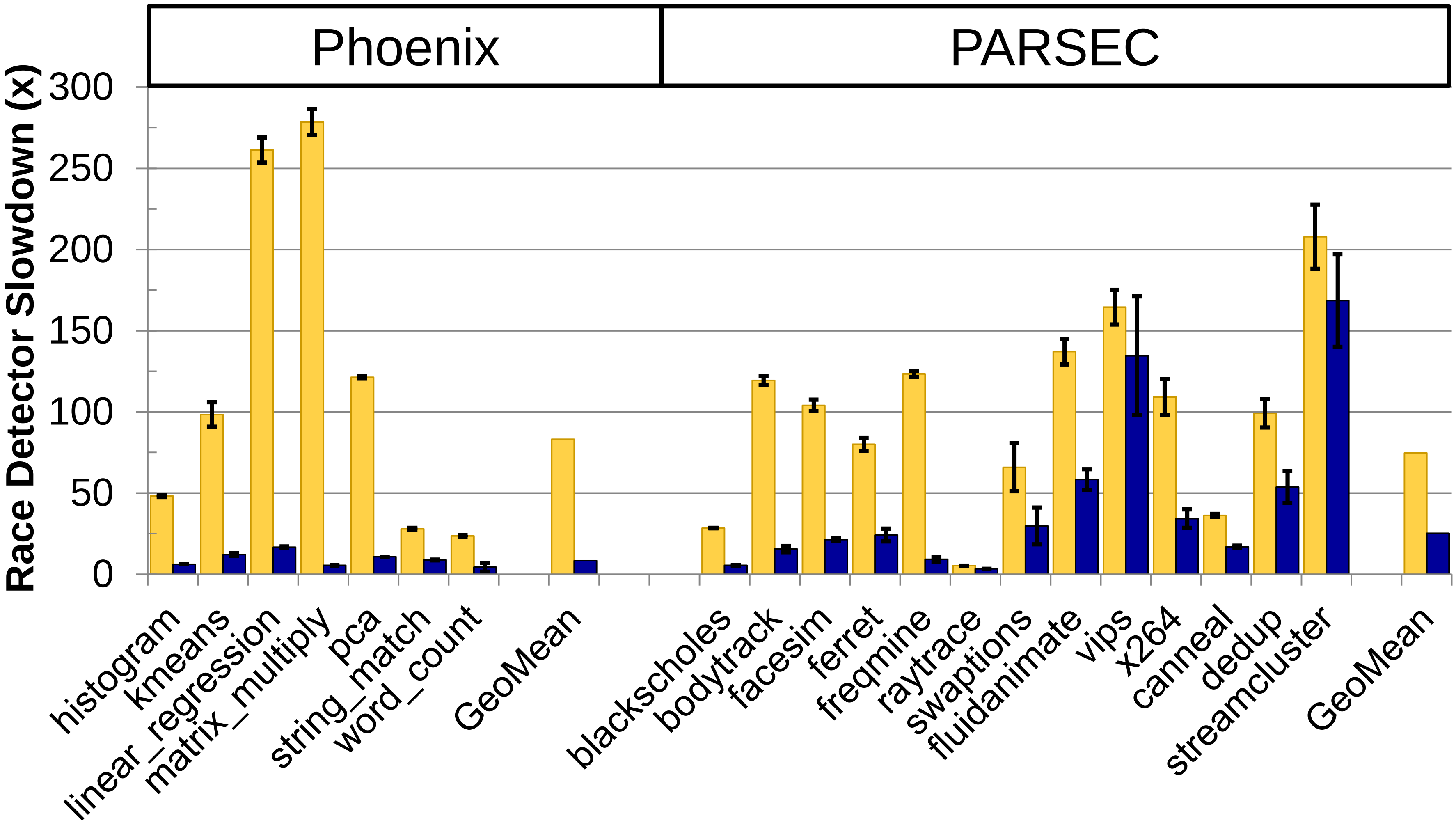
On-Demand Analysis on Real HW



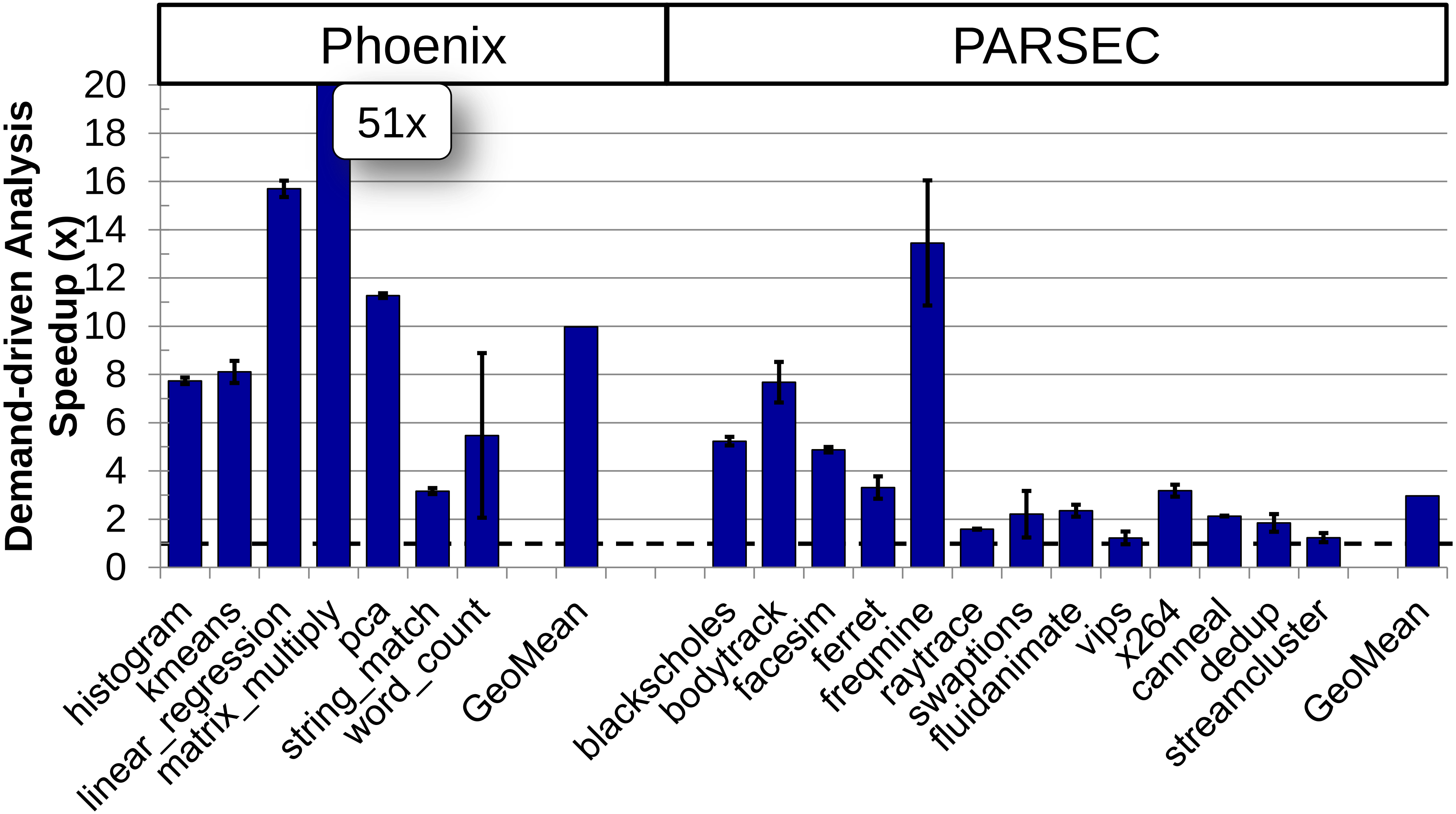
On-Demand Analysis on Real HW



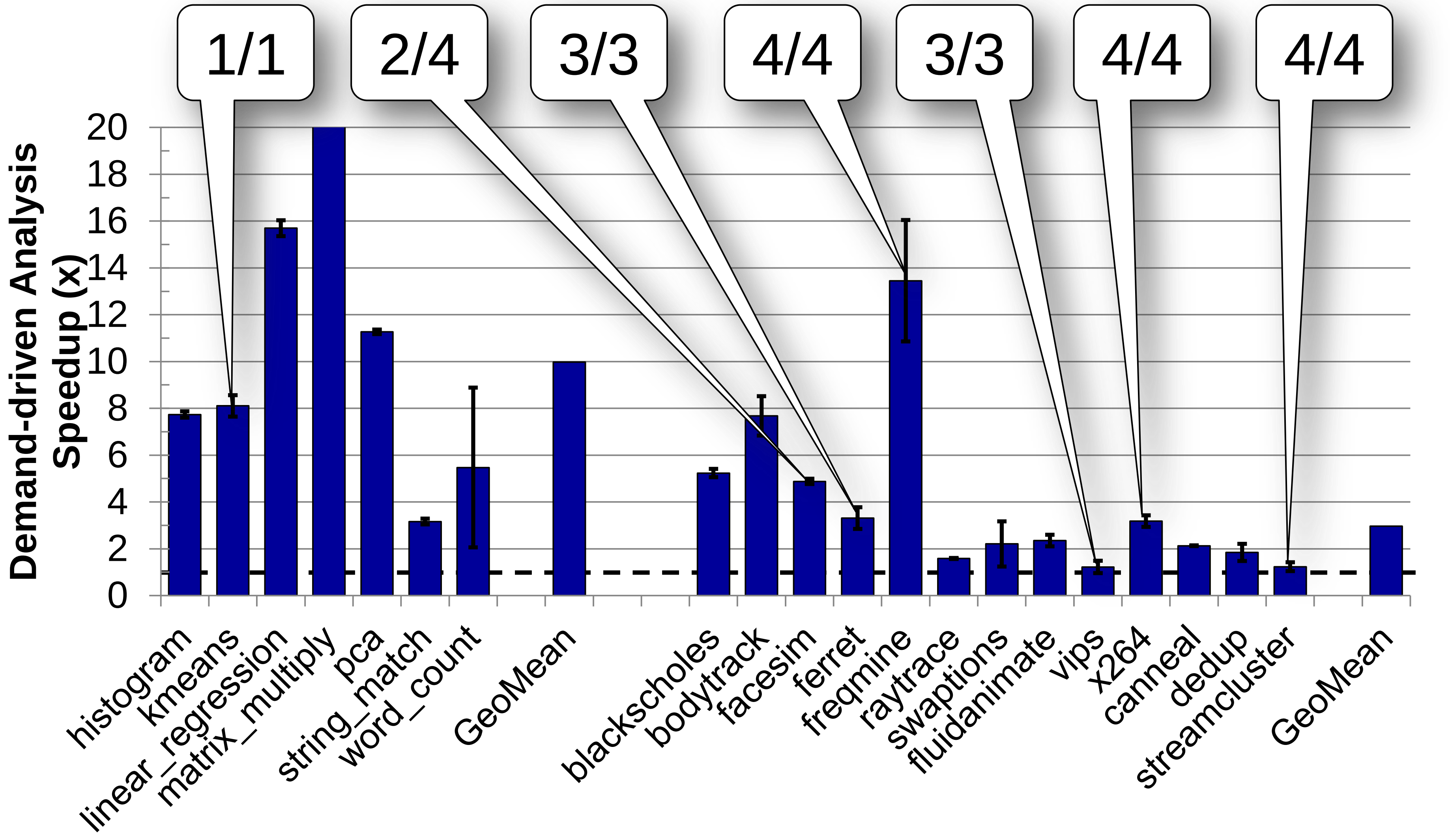
Performance Difference



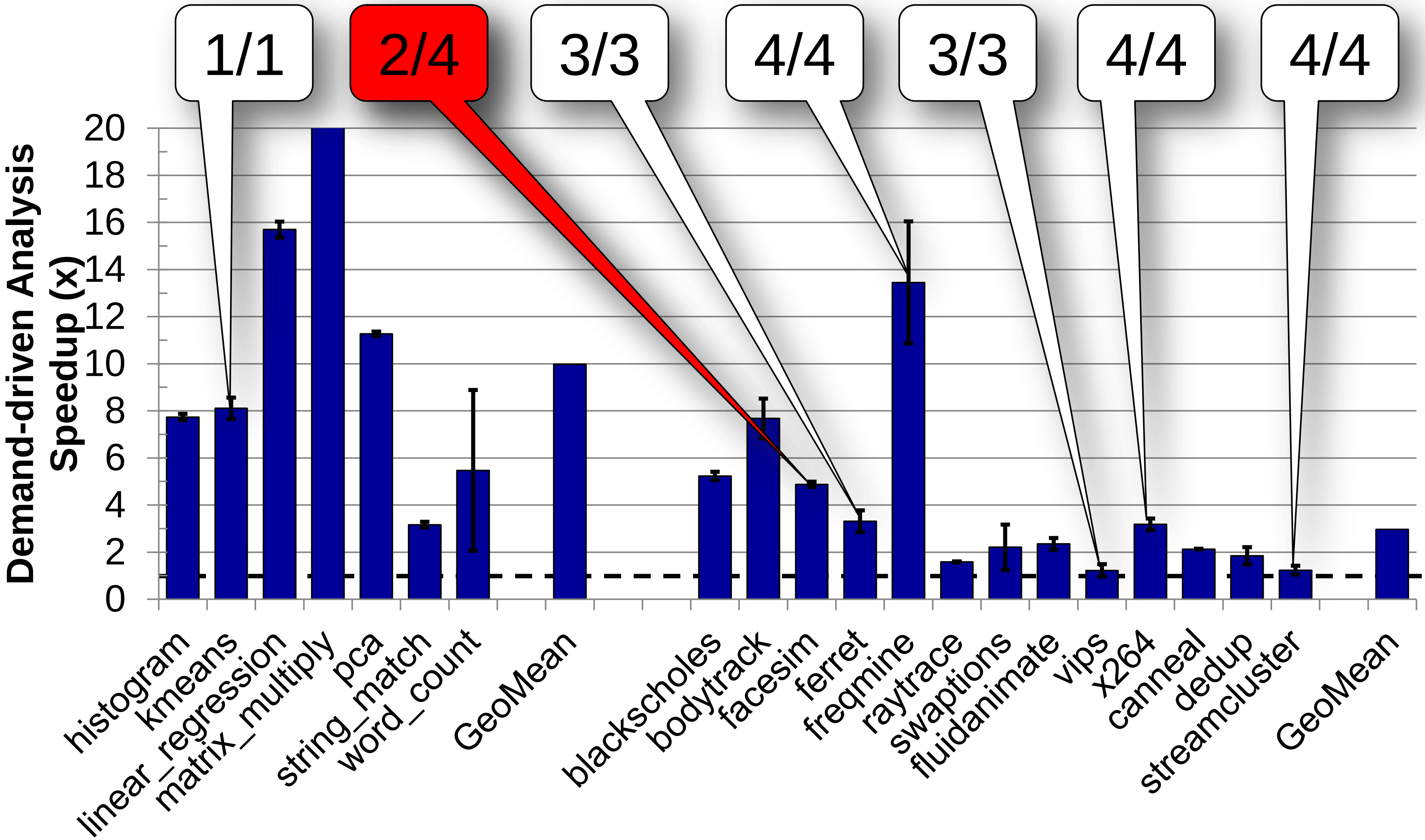
Performance Increases



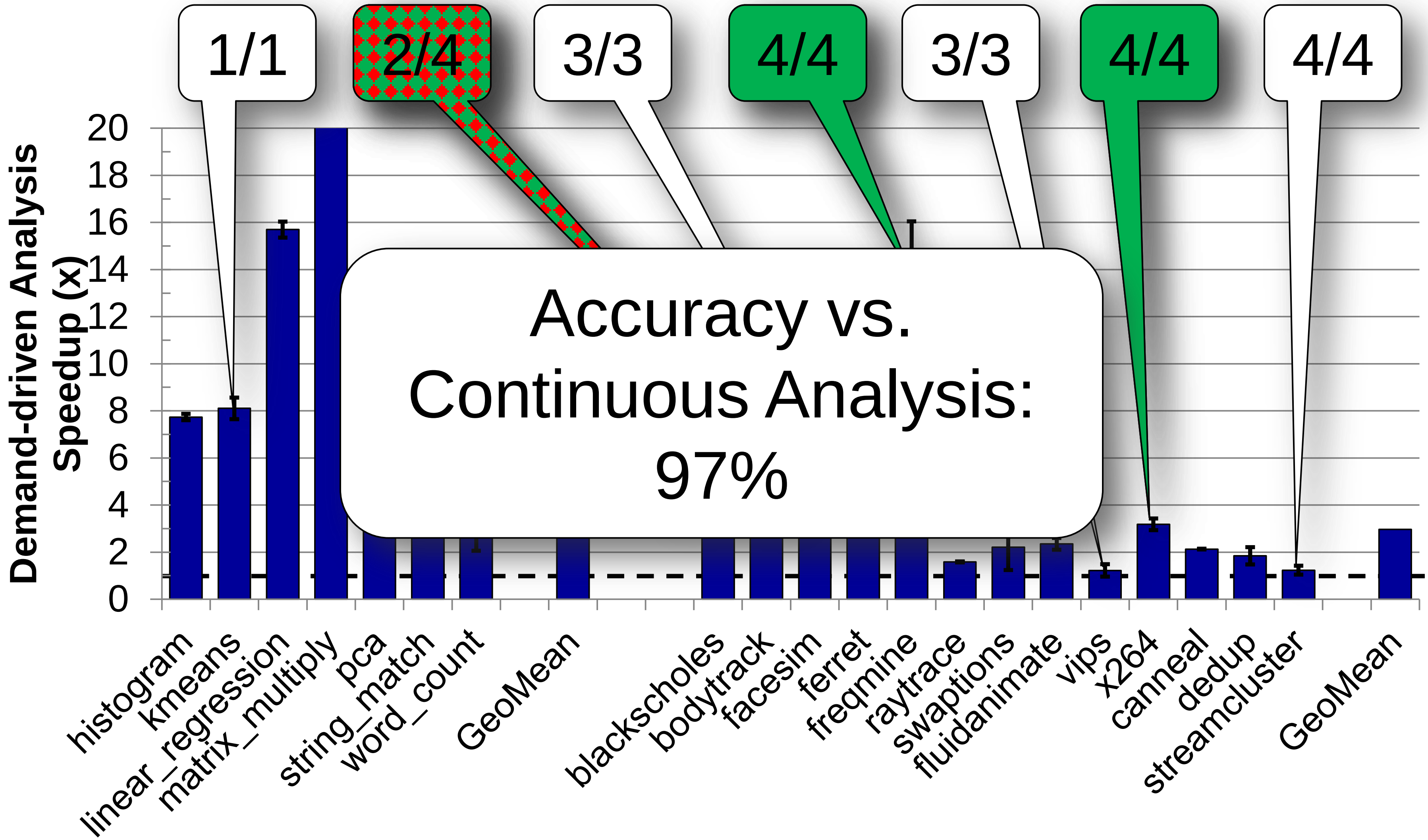
Demand-Driven Analysis Accuracy



Demand-Driven Analysis Accuracy



Demand-Driven Analysis Accuracy



Outline

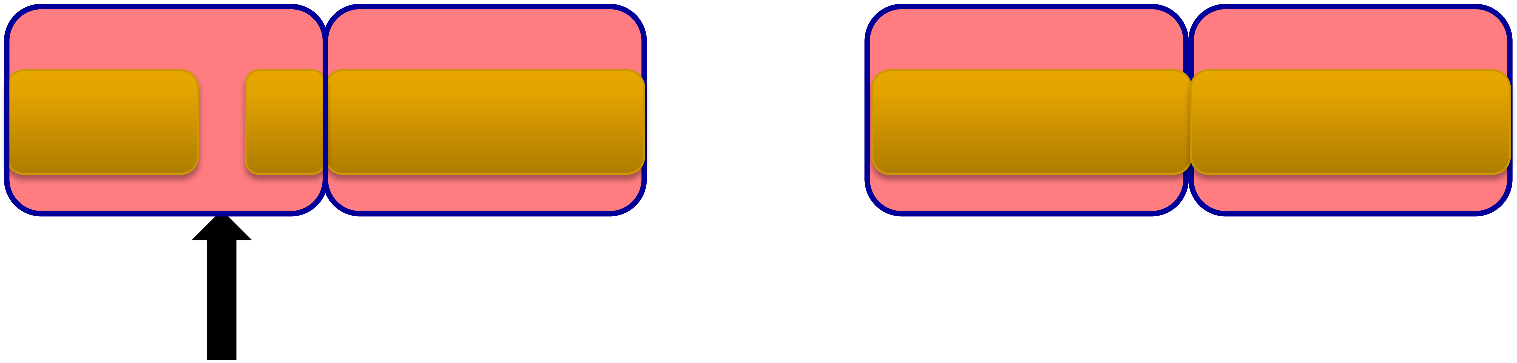
- Problem Statement
- Background Information
  - Demand-Driven Dynamic Dataflow Analysis
- Proposed Solutions
  - Demand-Driven Data Race Detection
  - Unlimited Hardware Watchpoints

Watchpoints Globally Useful

- Byte/Word Accurate and Per-Thread

Watchpoints Globally Useful

- Byte/Word Accurate and Per-Thread

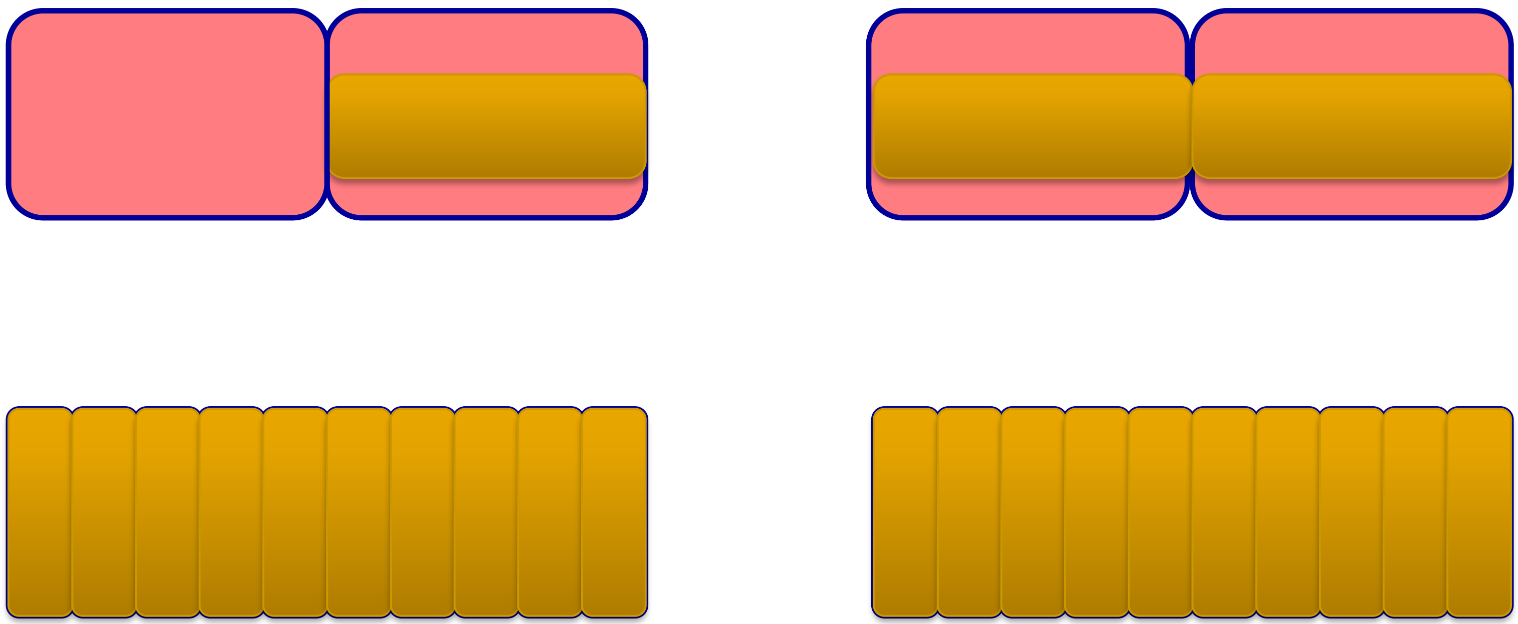


Watchpoints Globally Useful

- Byte/Word Accurate and Per-Thread

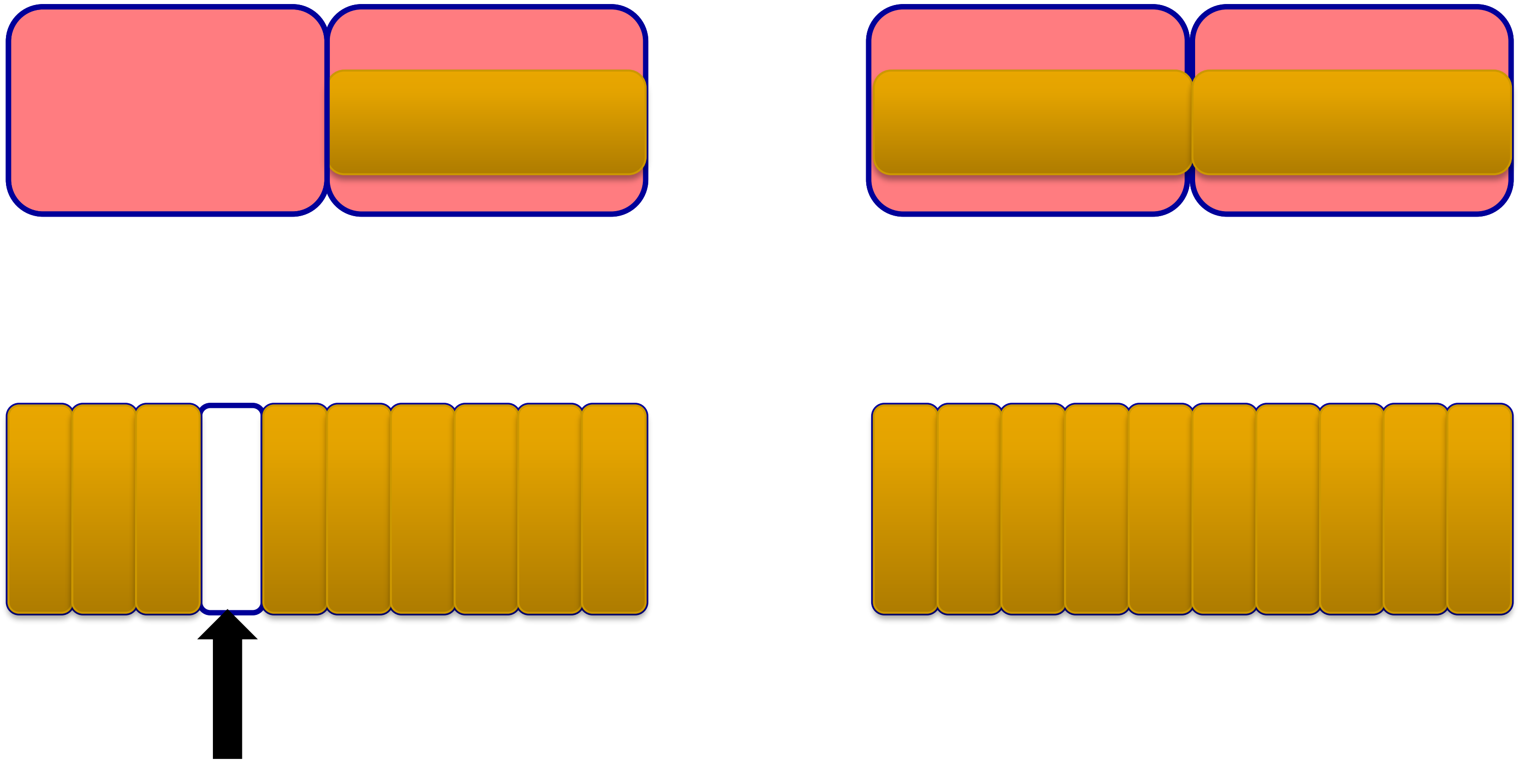
Watchpoints Globally Useful

- Byte/Word Accurate and Per-Thread



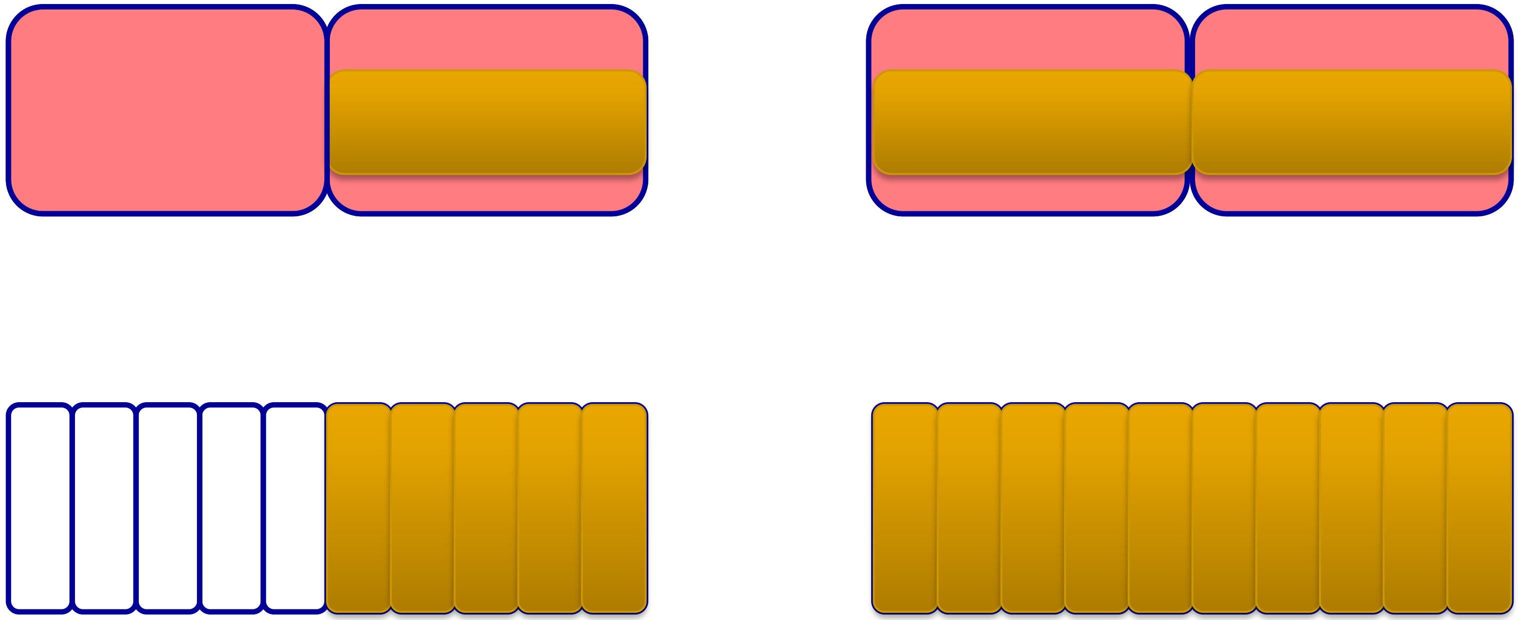
Watchpoints Globally Useful

- Byte/Word Accurate and Per-Thread



Watchpoints Globally Useful

- Byte/Word Accurate and Per-Thread



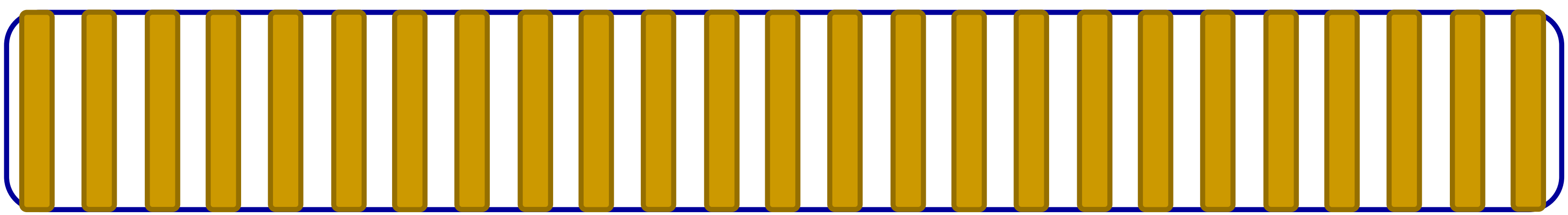
Watchpoint-Based Software Analyses

- Taint Analysis
- Data Race Detection
- Deterministic Execution
- Canary-Based Bounds Checking
- Speculative Program Optimization
- Hybrid Transactional Memory

Challenges

- Some analyses require watchpoint ranges

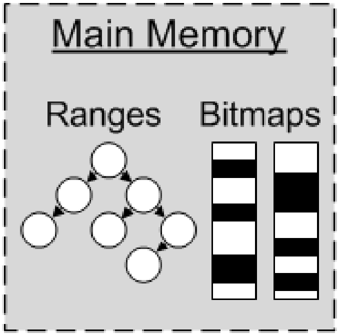
  - Better stored as base + length
- Some need large # of small watchpoints



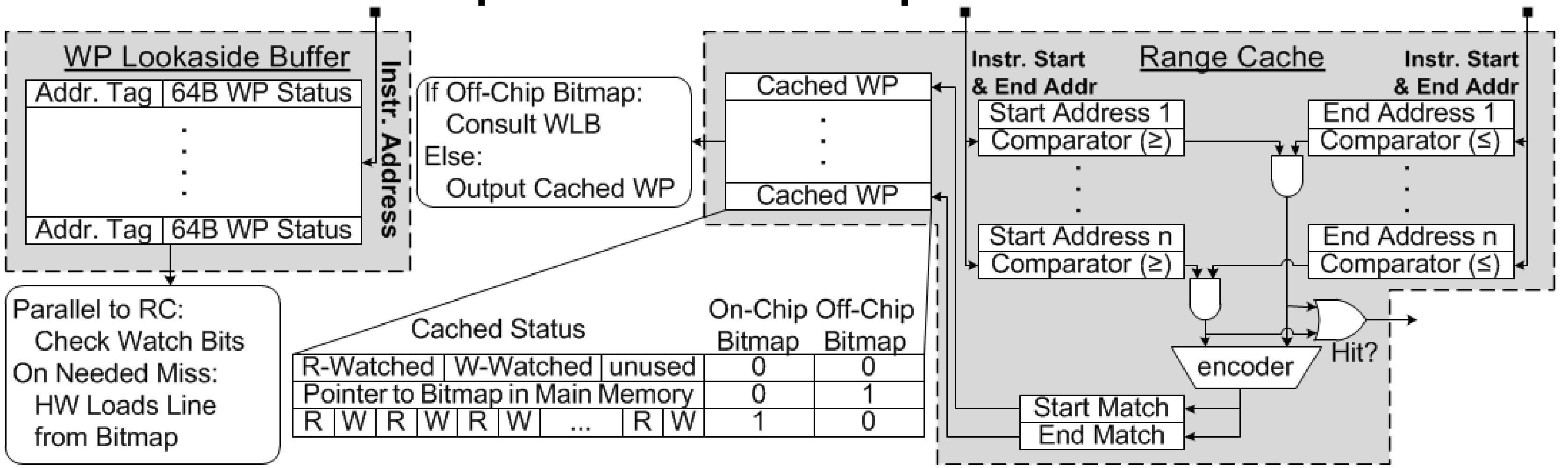
  - Better stored as bitmaps
- Need a large number

The Best of Both Worlds

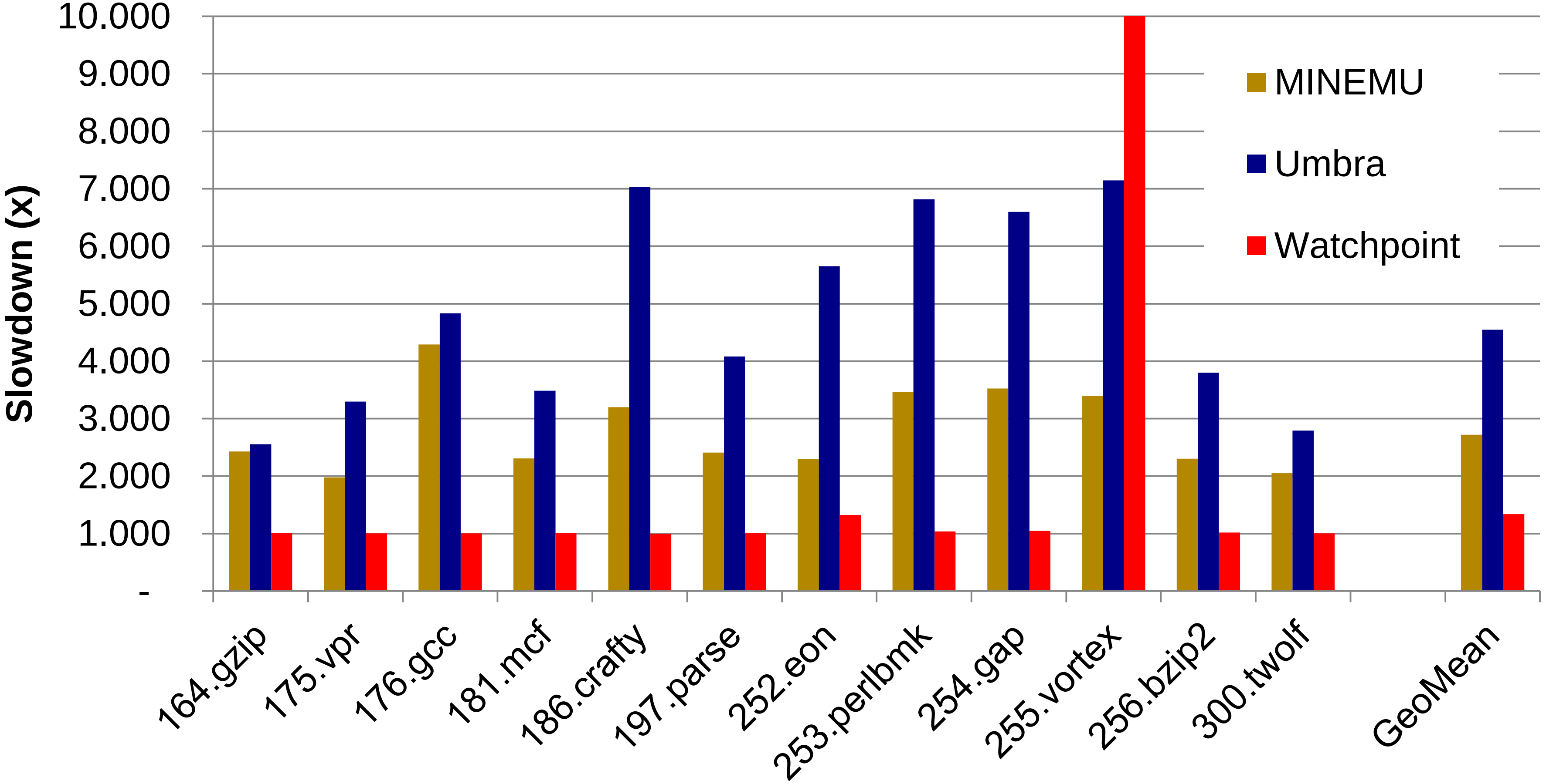
- Store Watchpoints in Main Memory



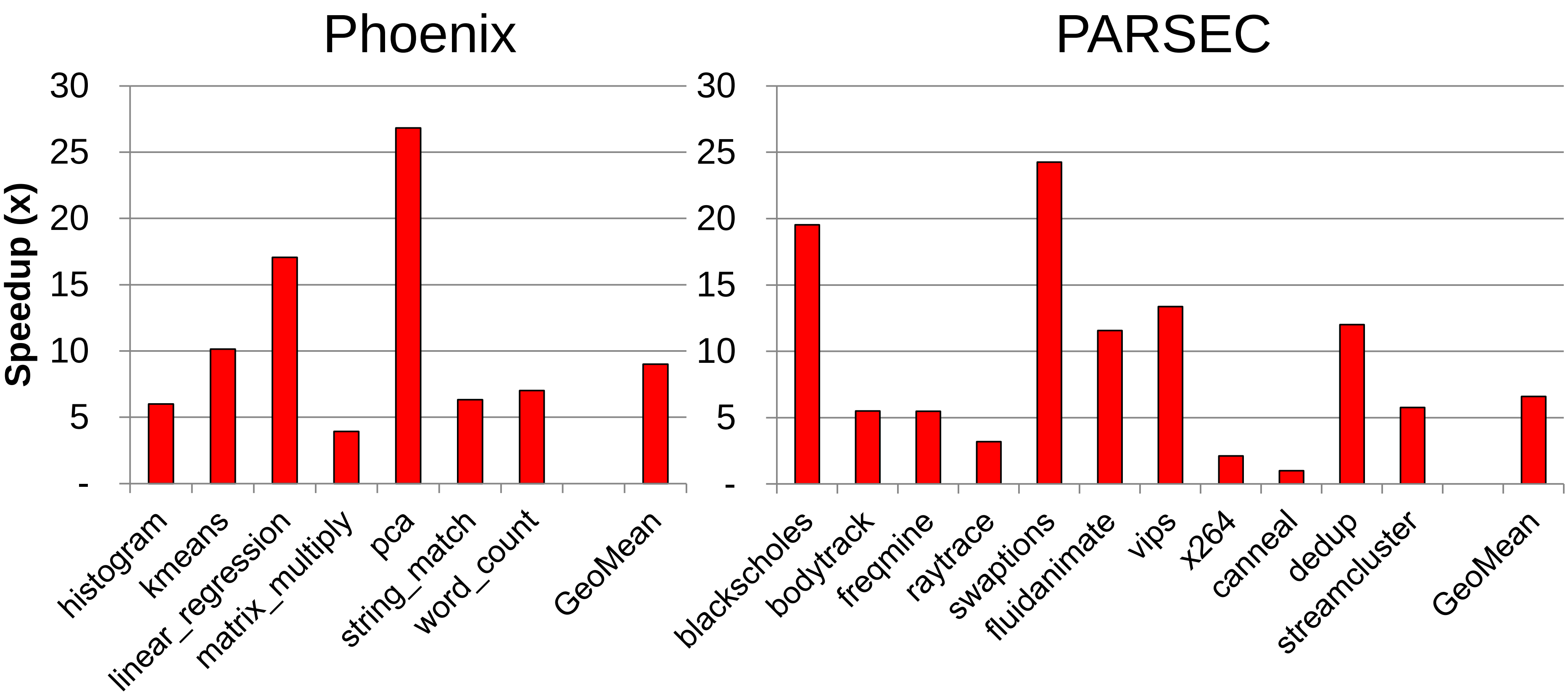
- Cache watchpoints on-chip



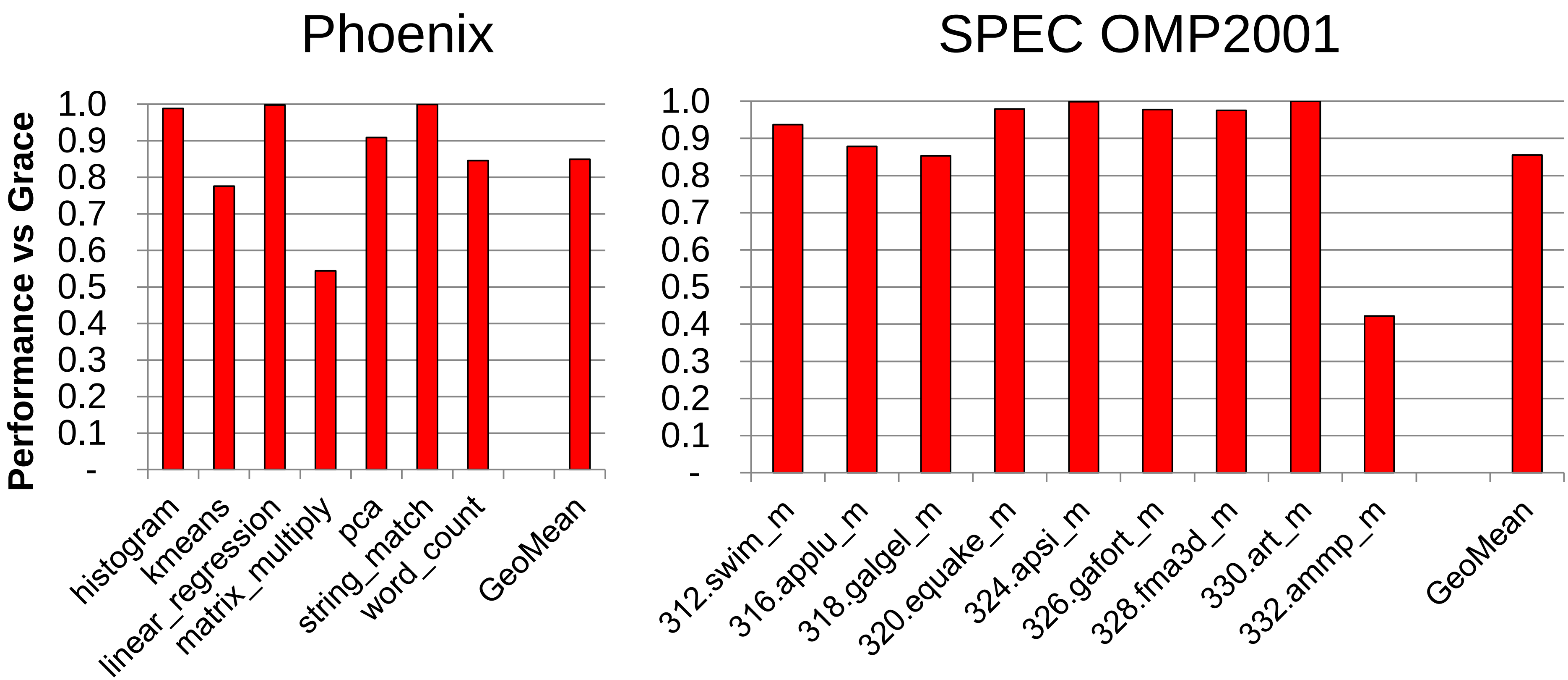
Demand-Driven Taint Analysis



Watchpoint-Based Data Race Detection



Watchpoint Deterministic Execution



BACKUP SLIDES