Accelerating Dynamic Software Analyses

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

February 20, 2012

Software Errors Abound

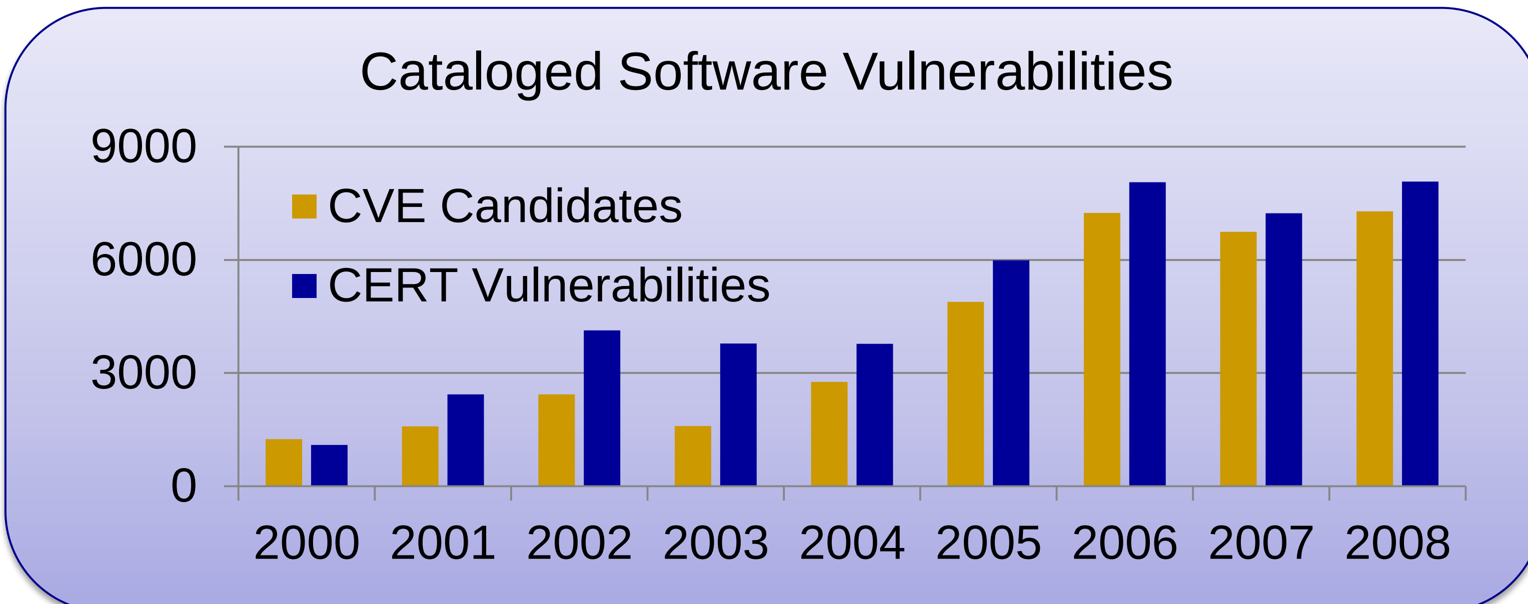
- NIST: SW errors cost U.S. ~$60 billion/year as of 2002

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.



Example of Modern Bug

Nov. 2010 OpenSSL Security Flaw

Example of 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 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

Ø

Example of 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)

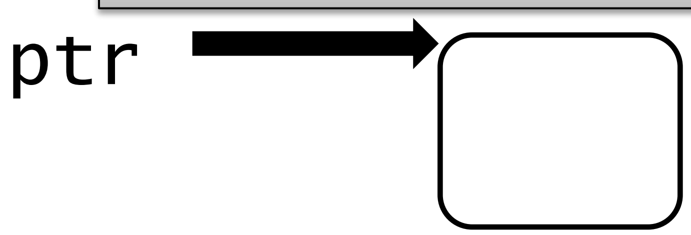
ptr

∅

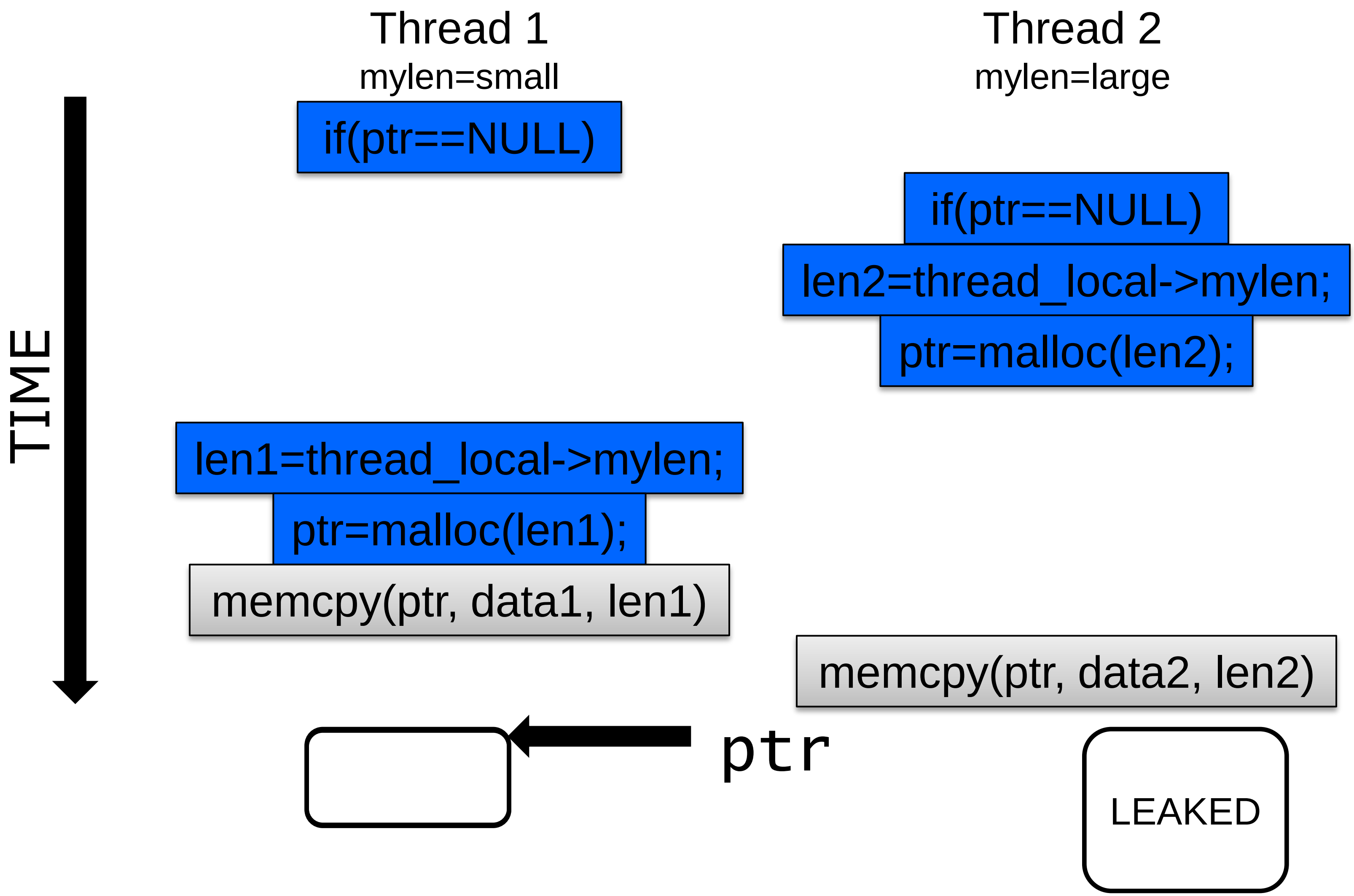
Example of Modern Bug

TIME

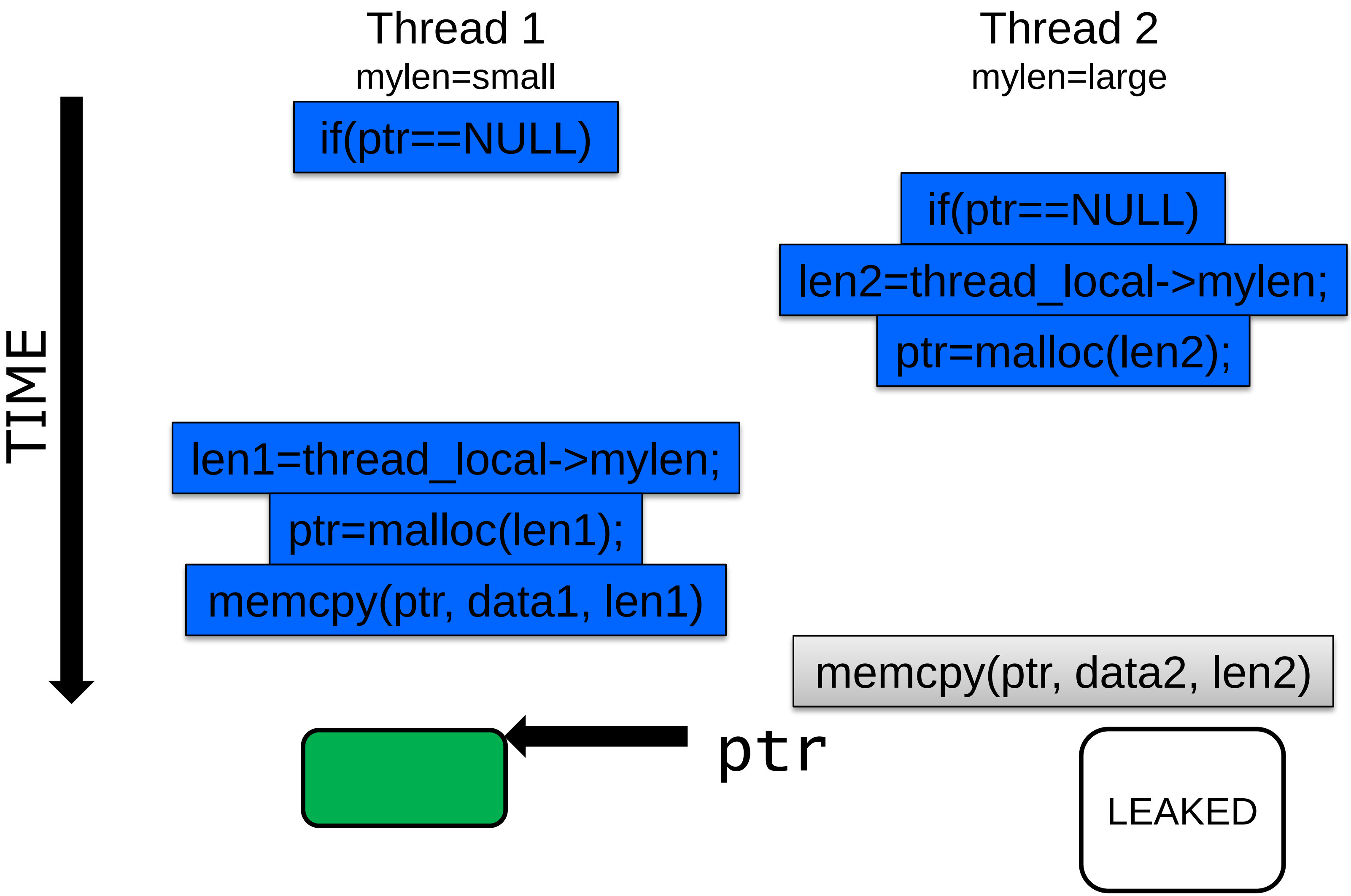
Thread 1 (mylen=small)	Thread 2 (mylen=large)
if(ptr==NULL)	
	if(ptr==NULL)
	len2=thread_local->mylen;
	ptr=malloc(len2);
len1=thread_local->mylen;	
ptr=malloc(len1);	
memcpy(ptr, data1, len1)	
	memcpy(ptr, data2, len2)



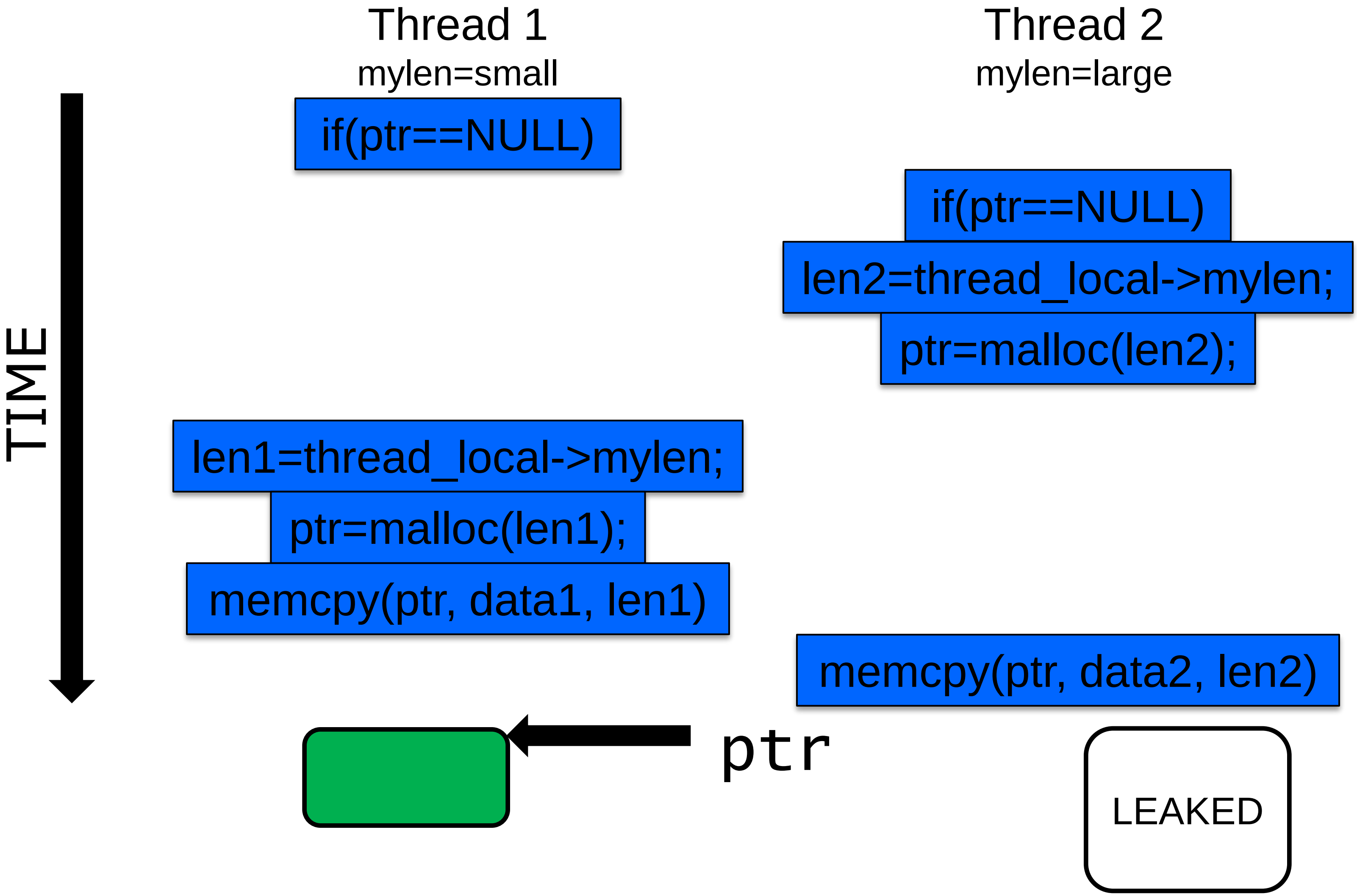
Example of Modern Bug



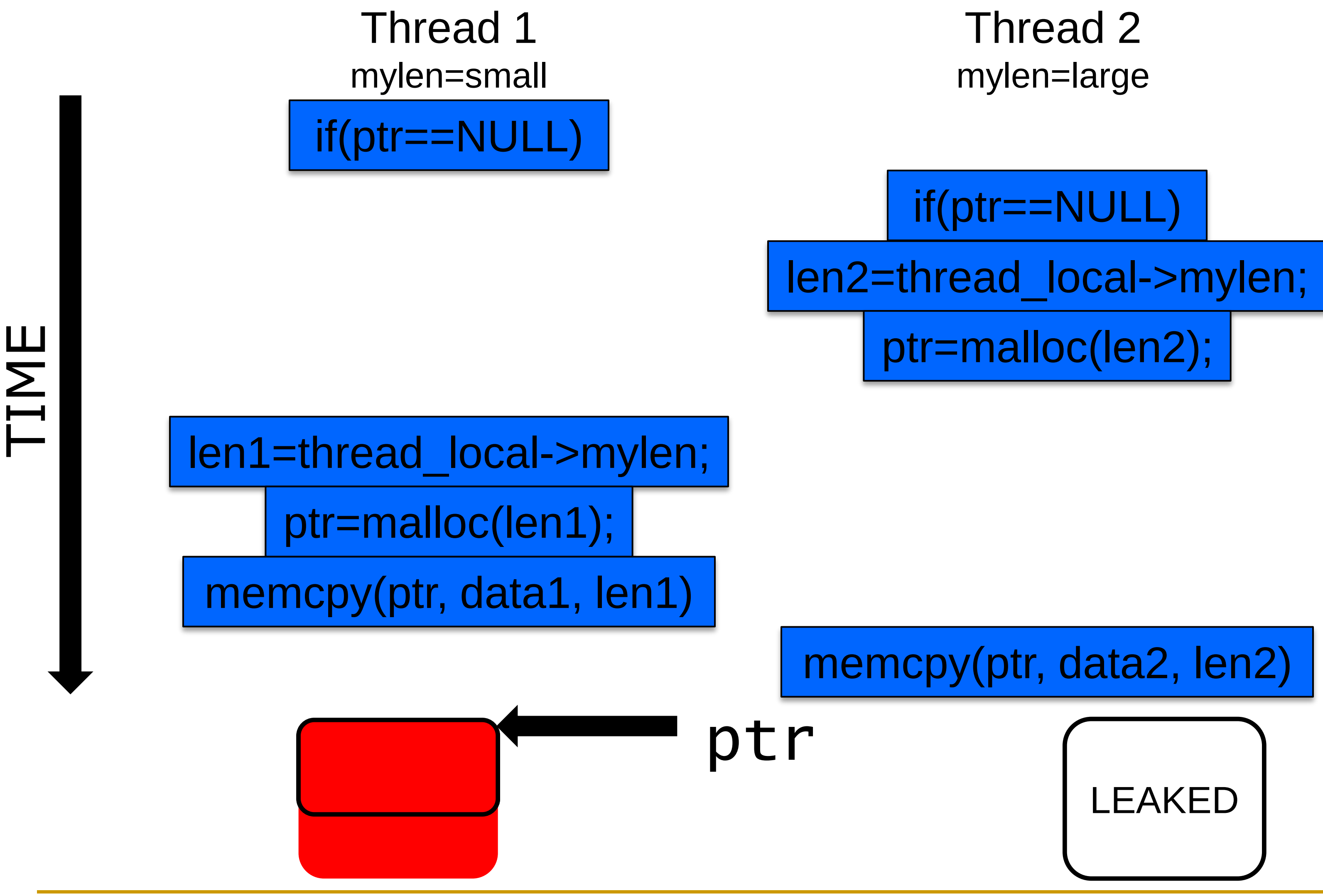
Example of Modern Bug



Example of Modern Bug

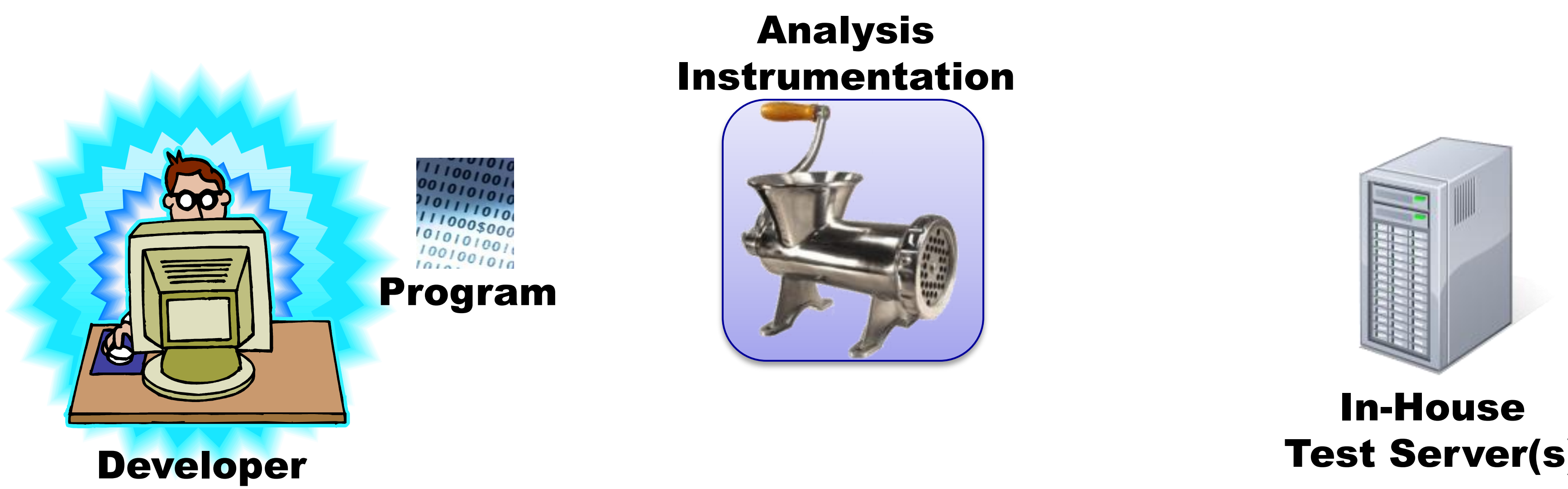


Example of Modern Bug



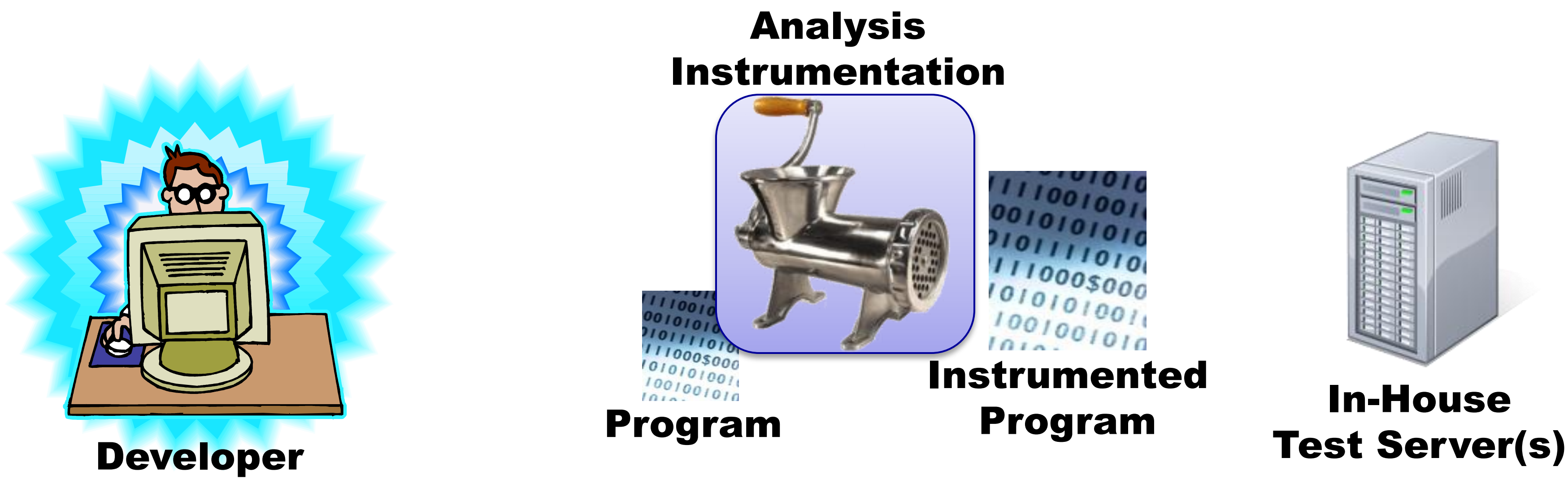
Dynamic Software Analysis

- 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

Analysis Instrumentation

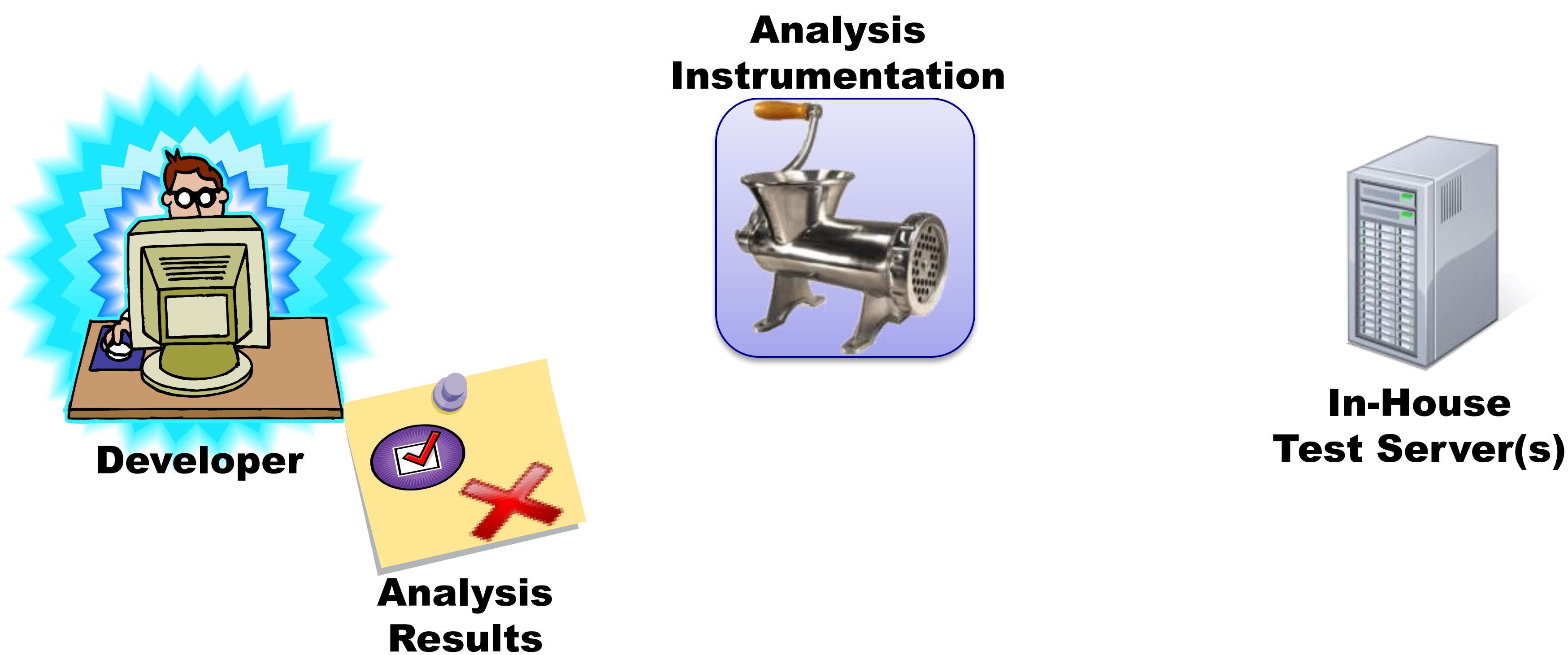
Developer

In-House Test Server(s)

LONG run time

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

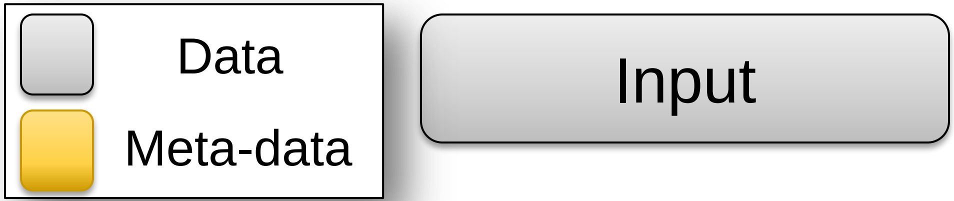
Outline

- Problem Statement
- Background Information
  - Demand-Driven Dynamic Dataflow Analysis
- Proposed Solutions
  - Demand-Driven Data Race Detection
  - Sampling to Cap Maximum Overheads

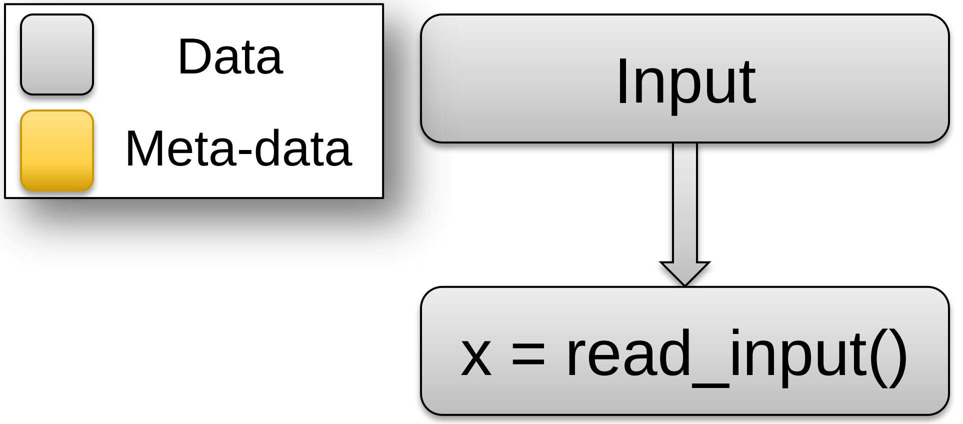
Dynamic Dataflow Analysis

- **Associate** meta-data with program values
- **Propagate/Clear** meta-data while executing
- **Check** meta-data for safety & correctness
- Forms dataflows of meta/shadow information

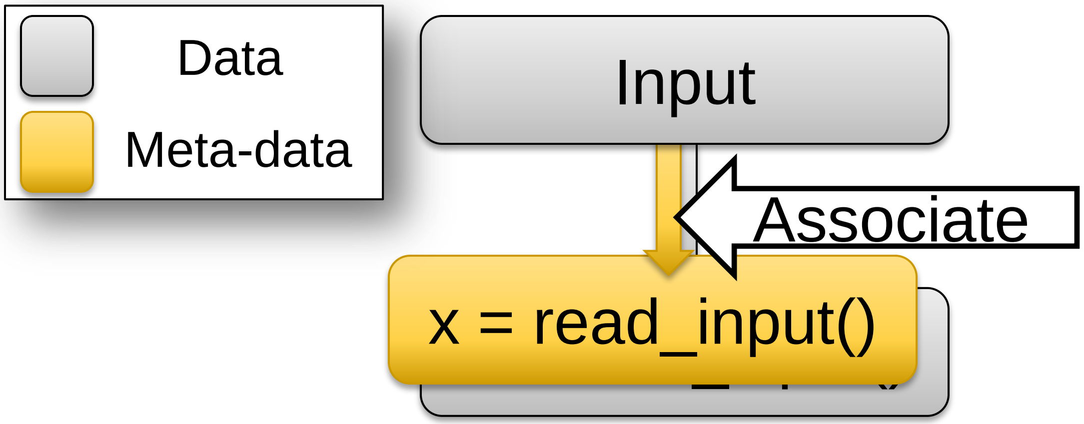
Example: Taint Analysis



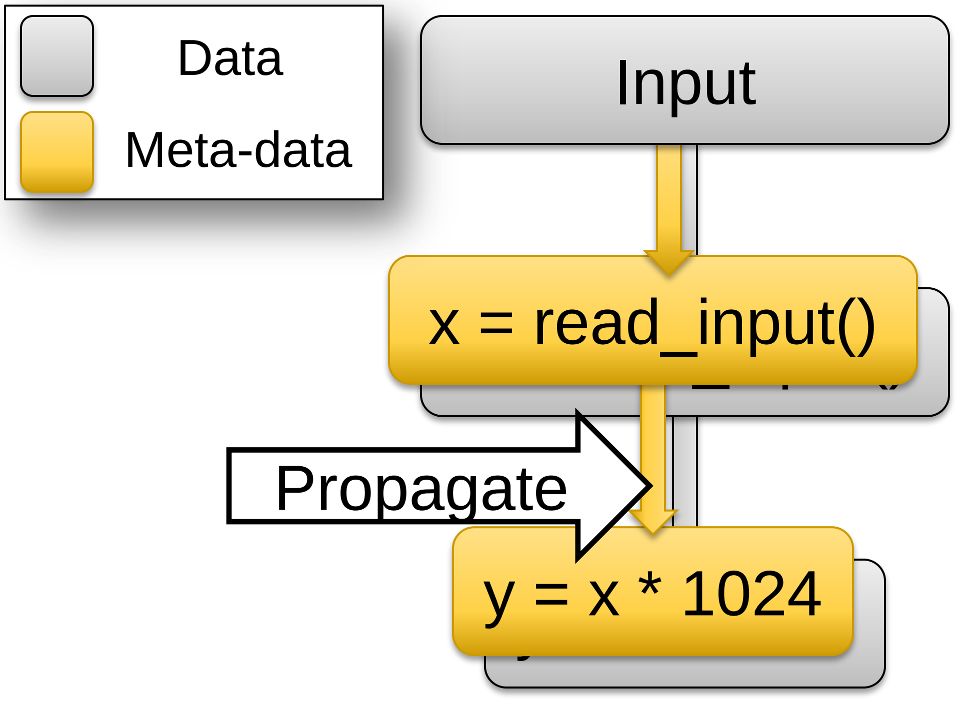
Example: Taint Analysis



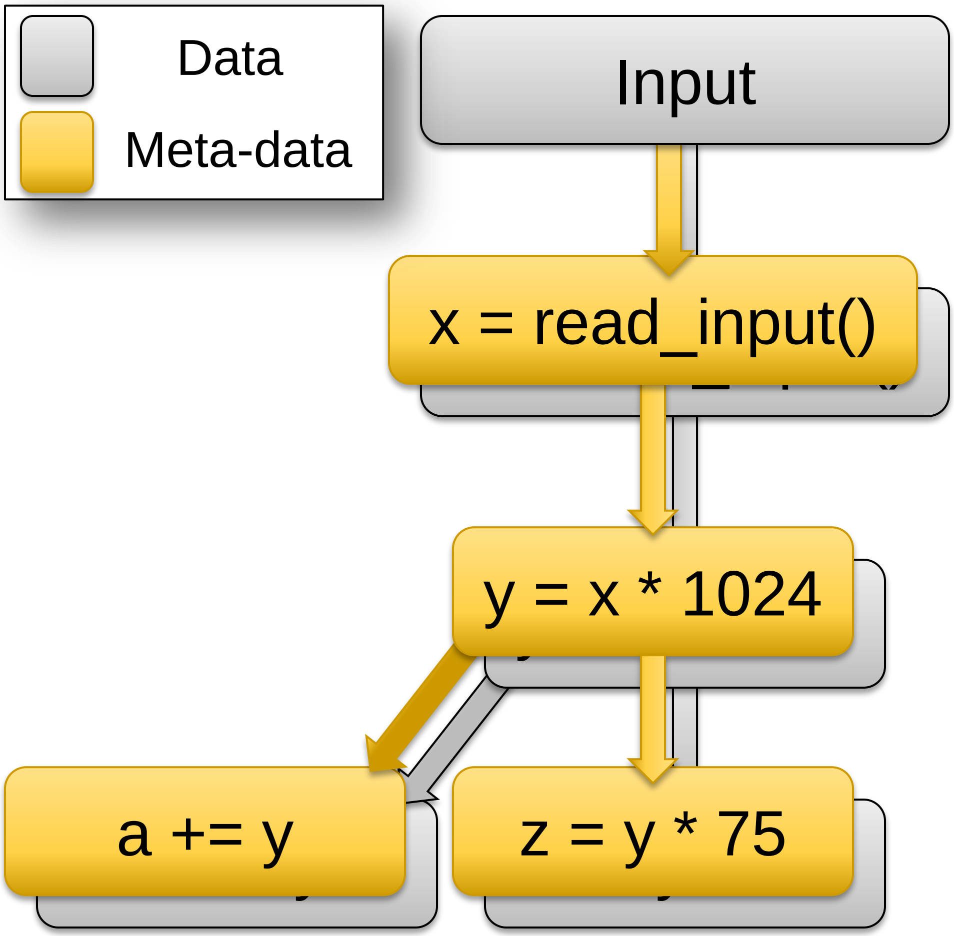
Example: Taint Analysis



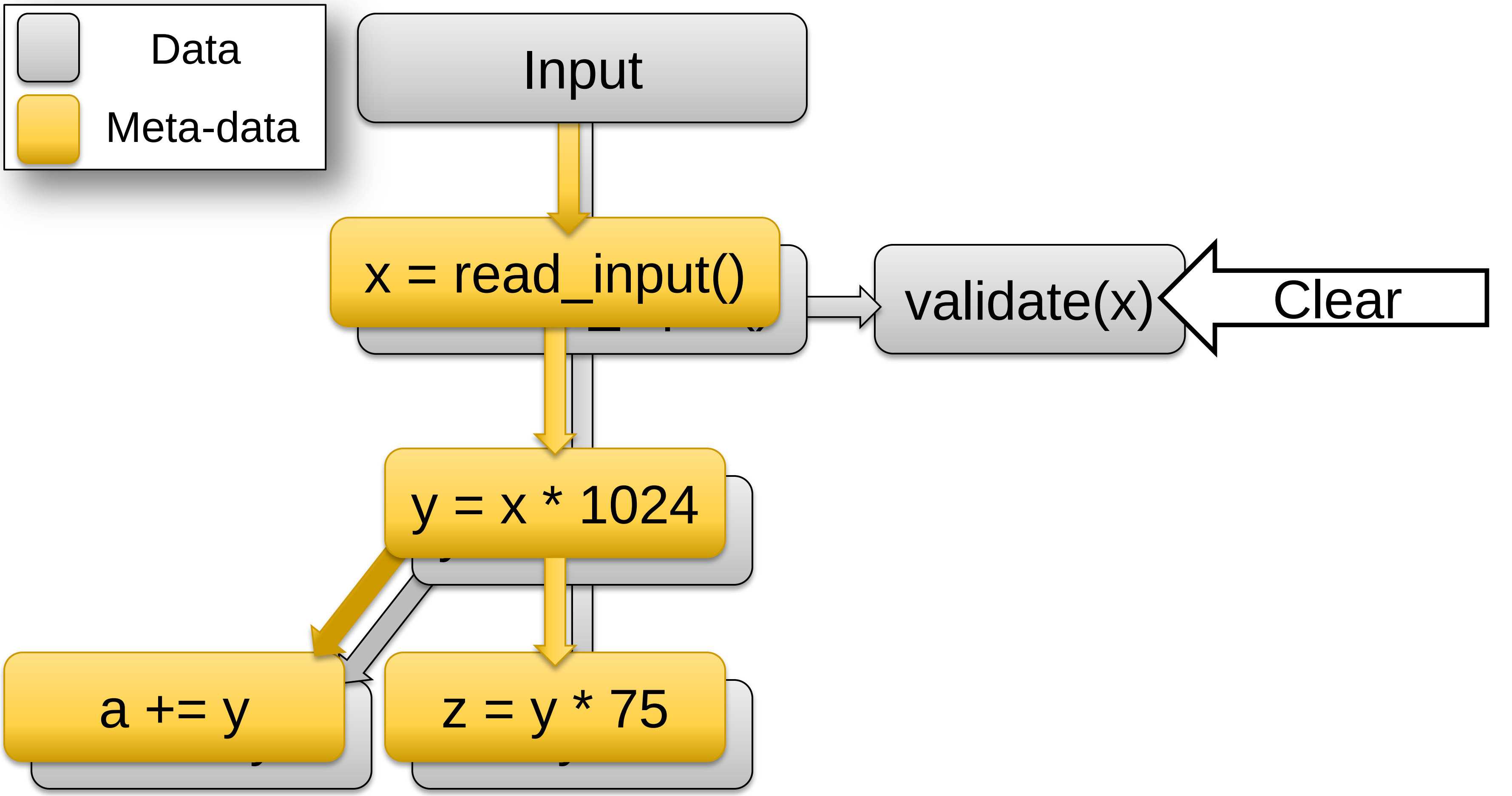
Example: Taint Analysis



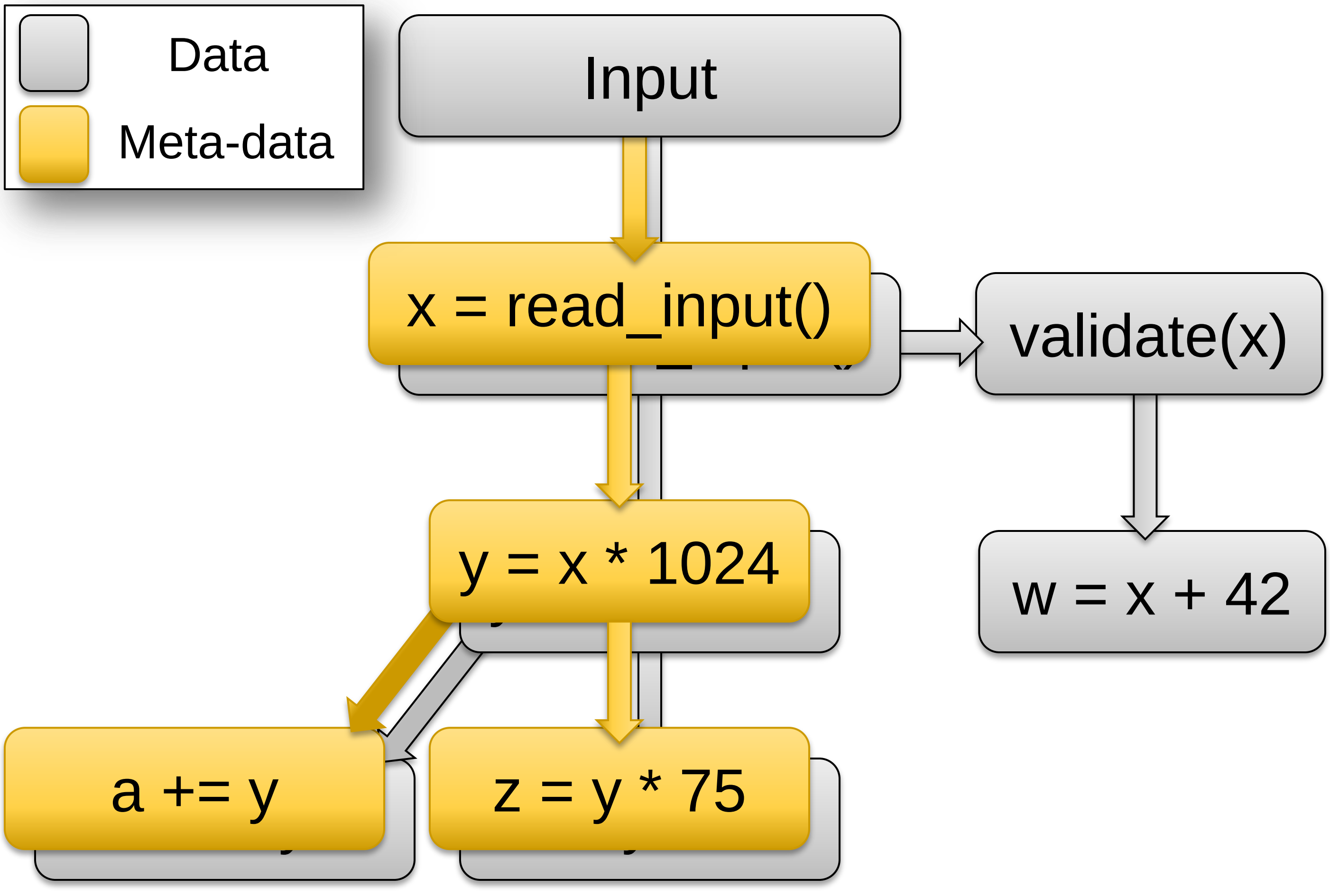
Example: Taint Analysis



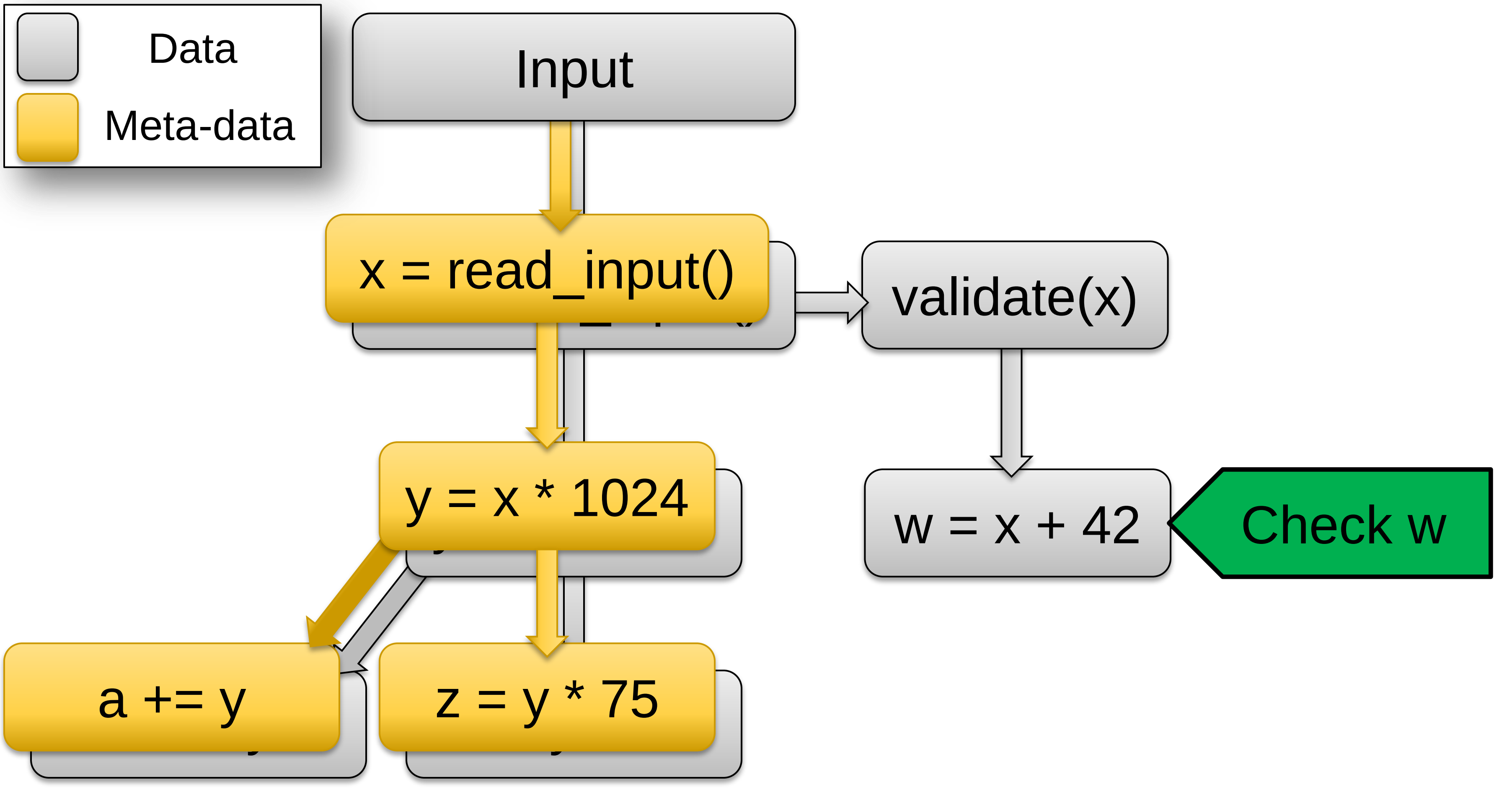
Example: Taint Analysis



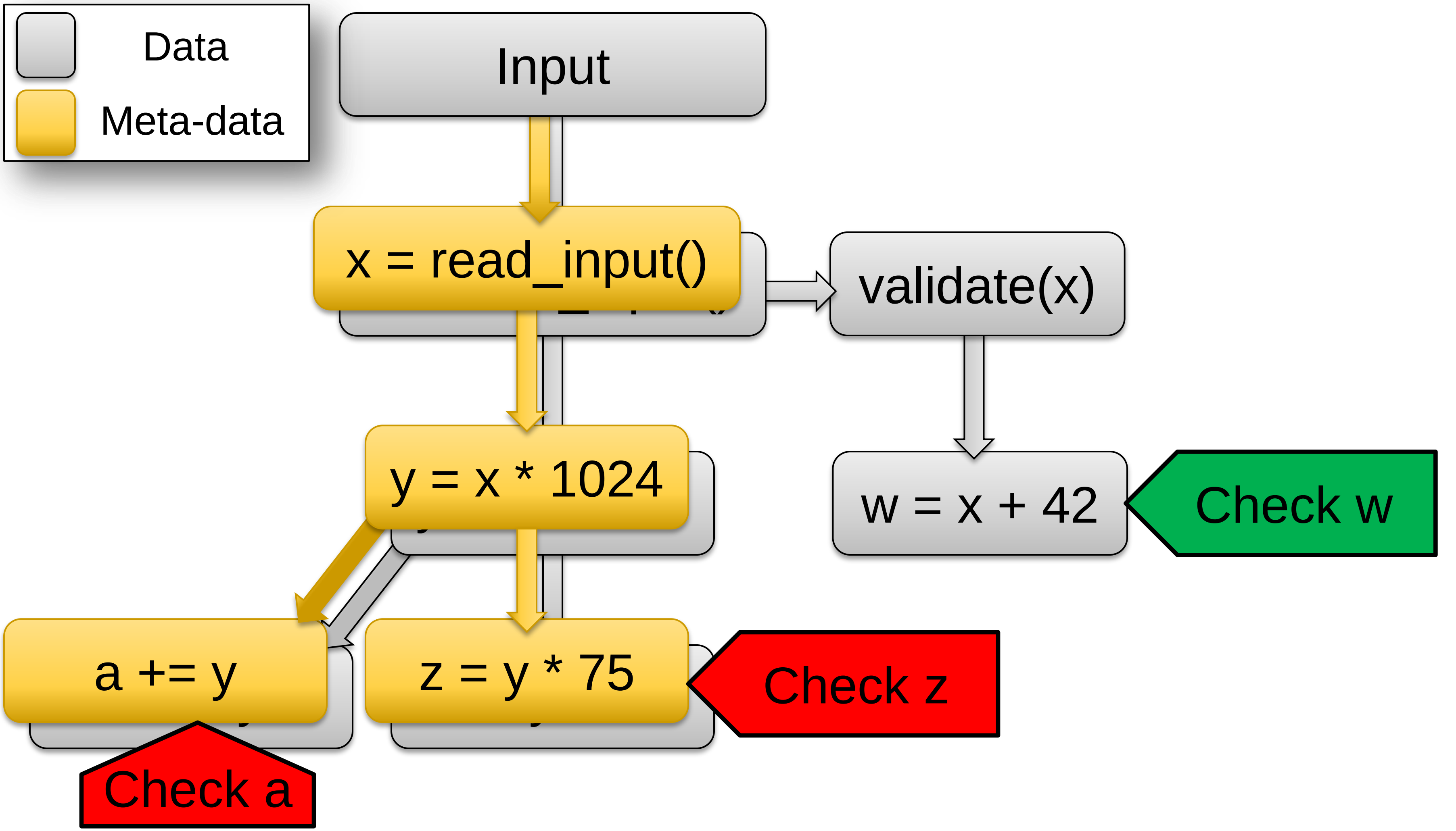
Example: Taint Analysis



Example: Taint Analysis

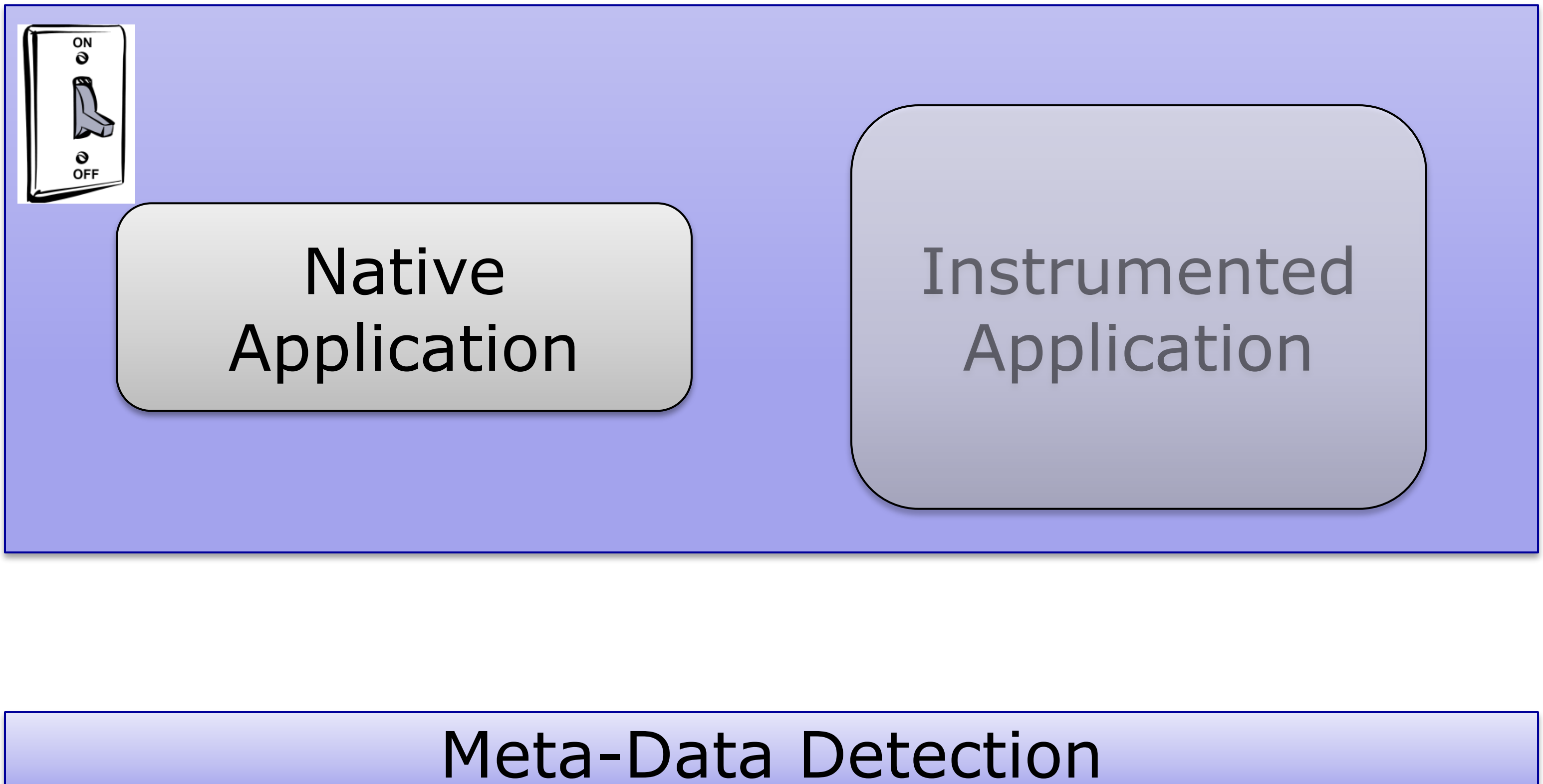


Example: Taint Analysis



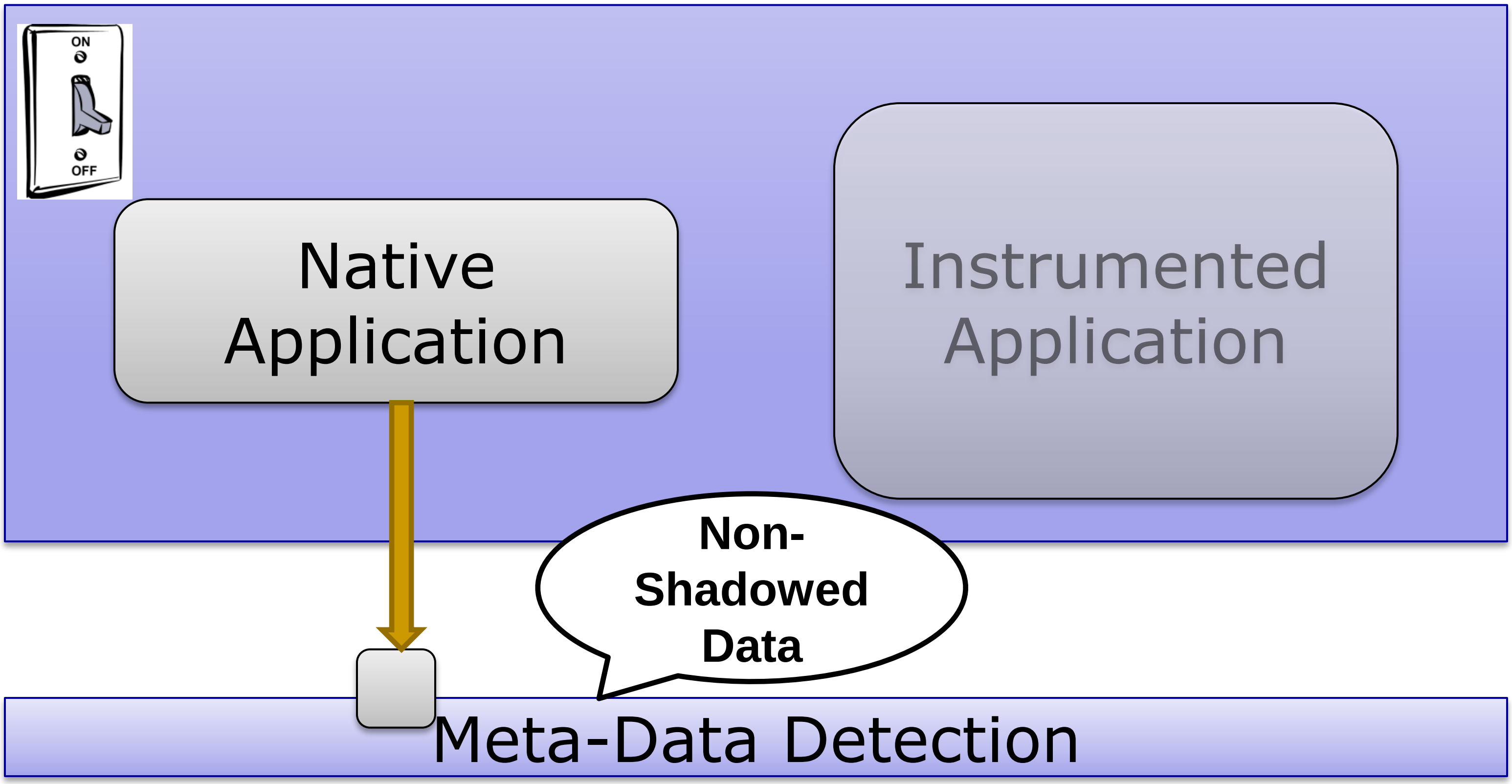
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data



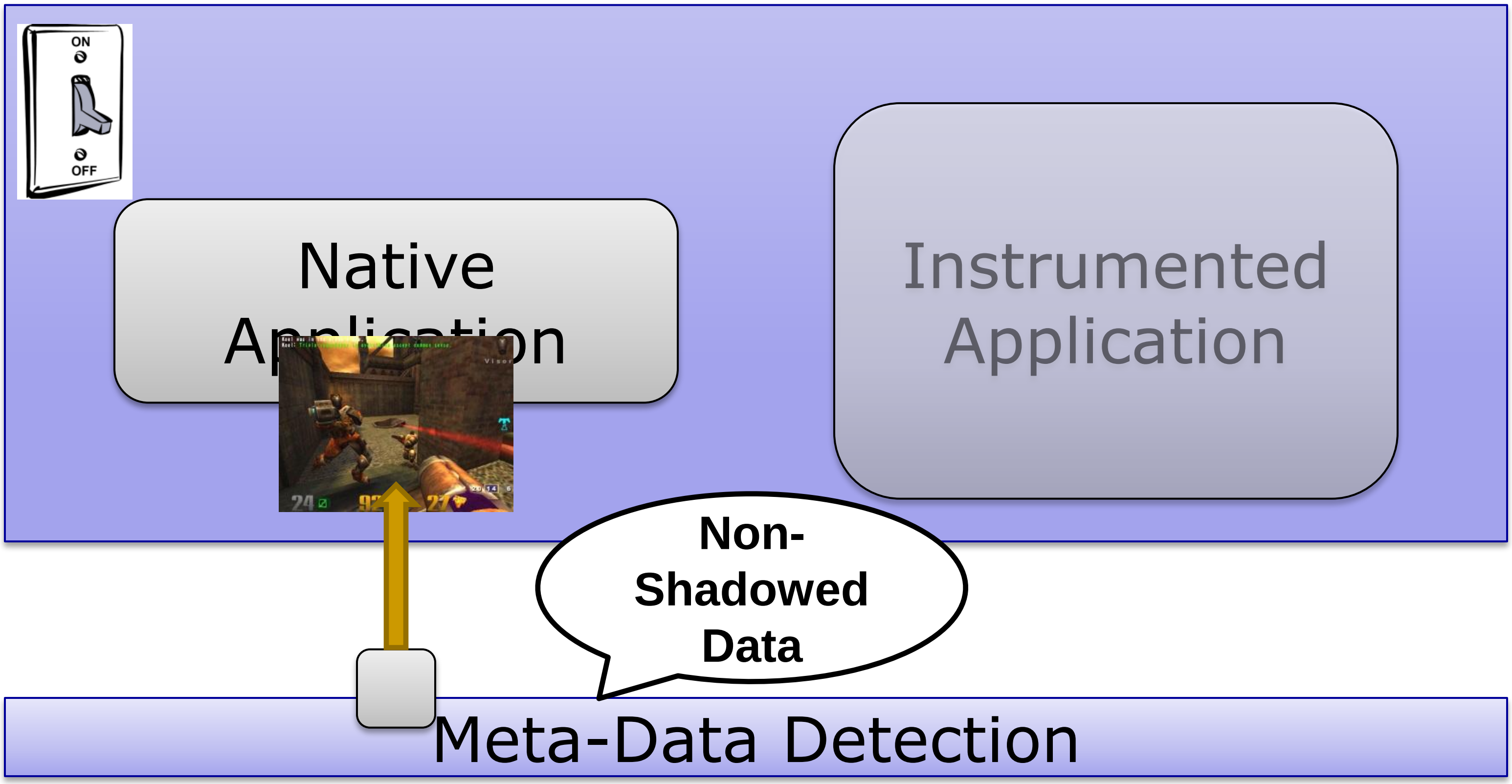
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data



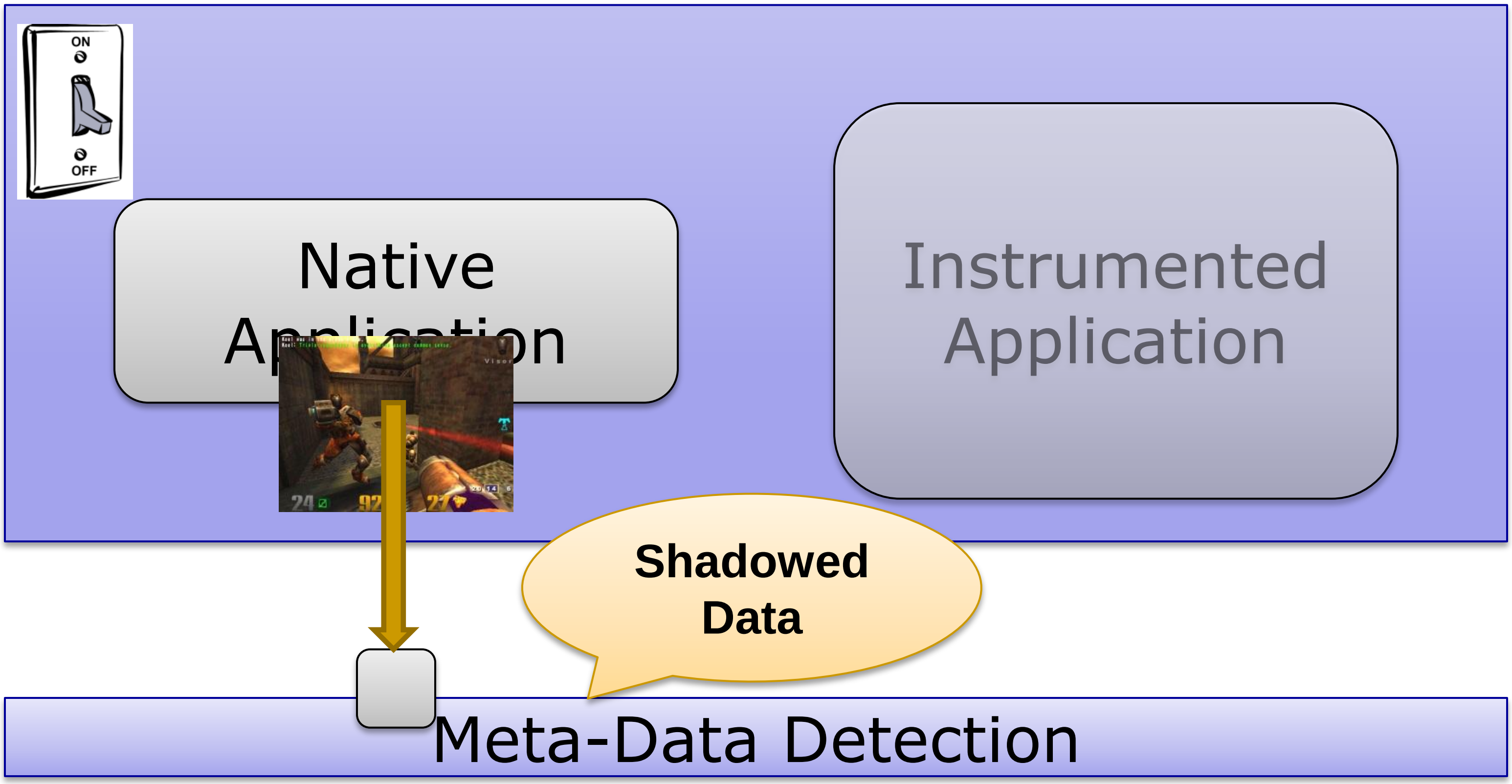
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data



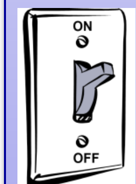
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data



Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data



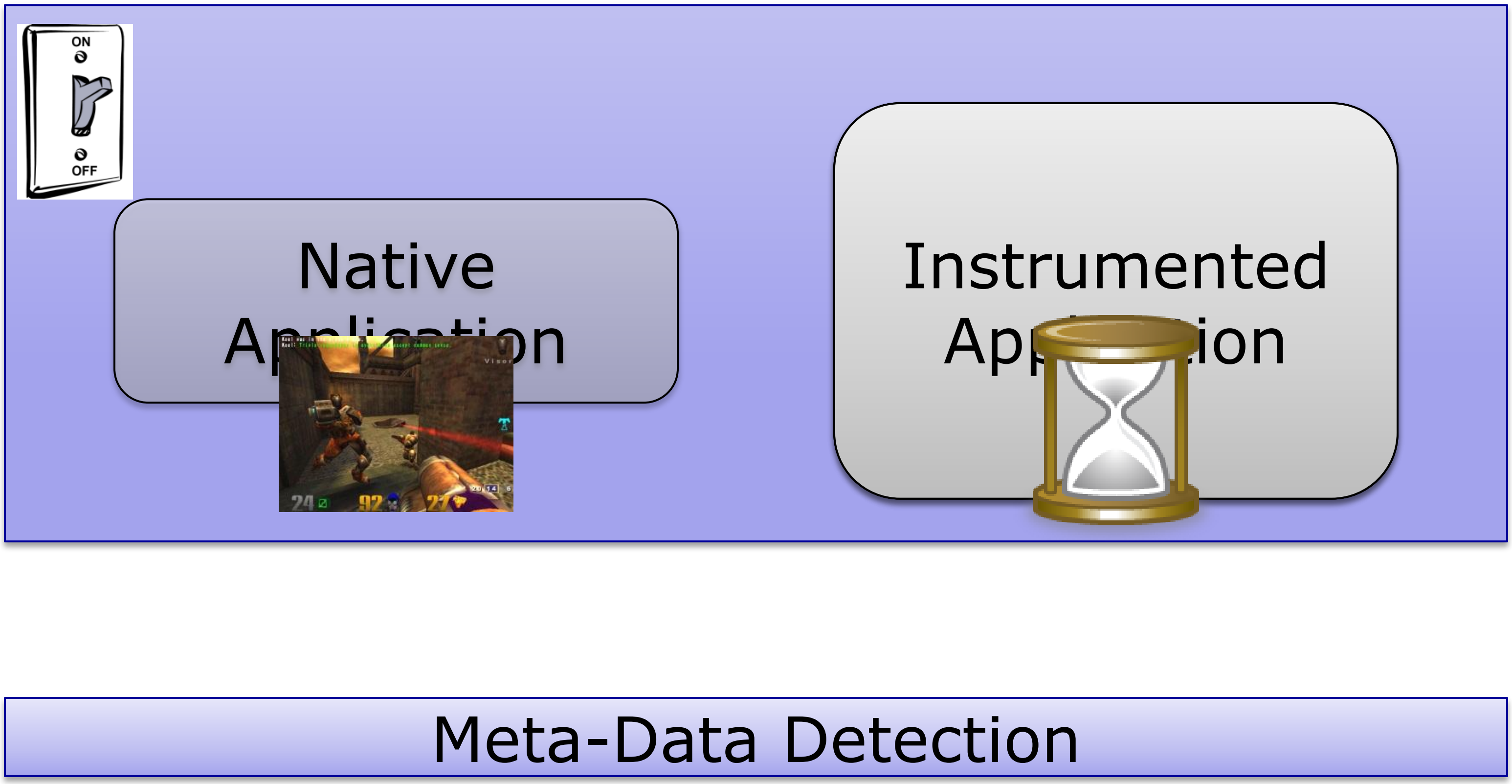
Native Application

Instrumented Application

Meta-Data Detection

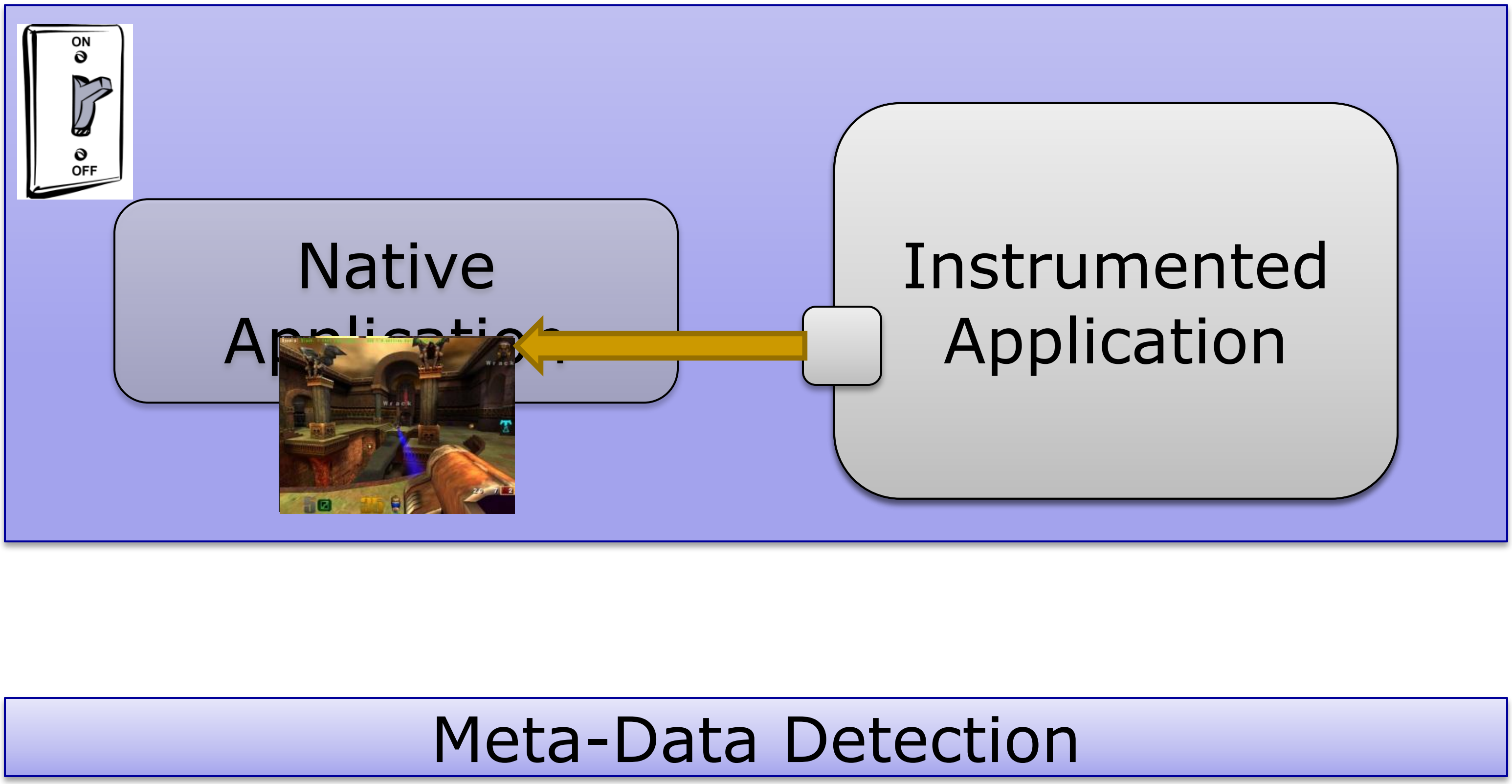
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data



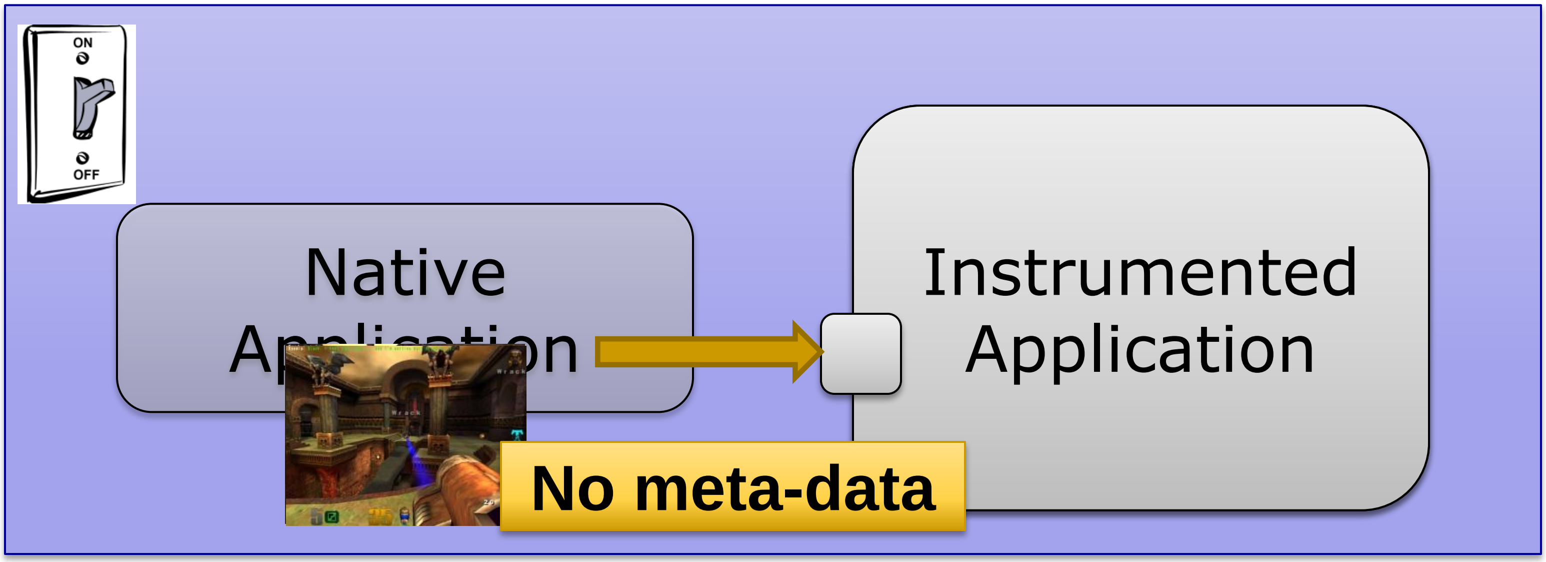
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data



Demand-Driven Dataflow Analysis

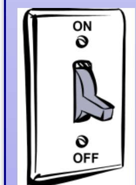
- Only Analyze Shadowed Data



Meta-Data Detection

Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data



Native Application

Instrumented Application

Meta-Data Detection

Finding Meta-Data

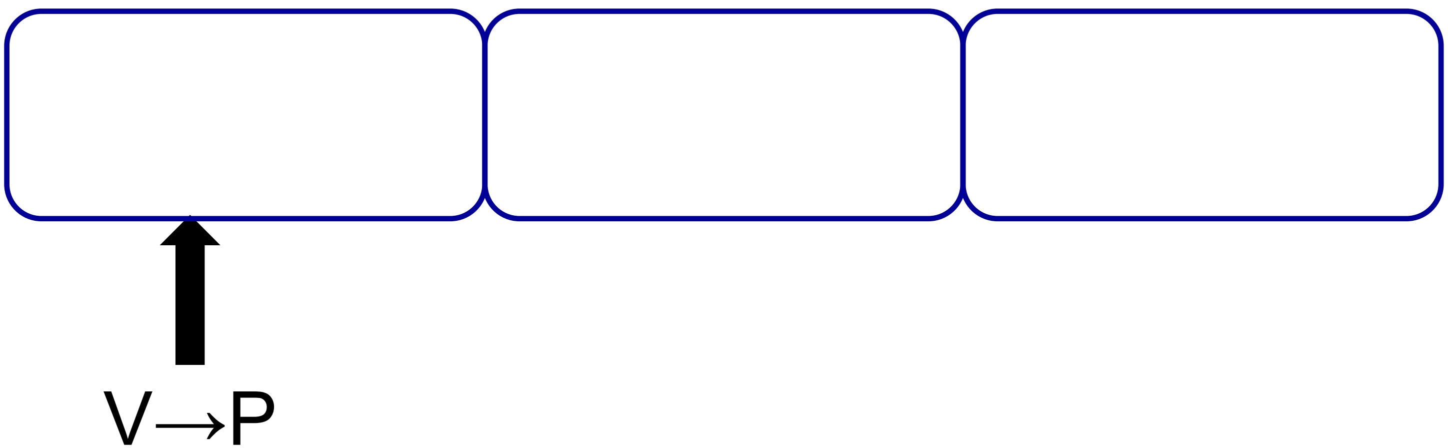
- No additional overhead when no meta-data
  - Needs hardware support
- Take a fault when touching shadowed data

Finding Meta-Data

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

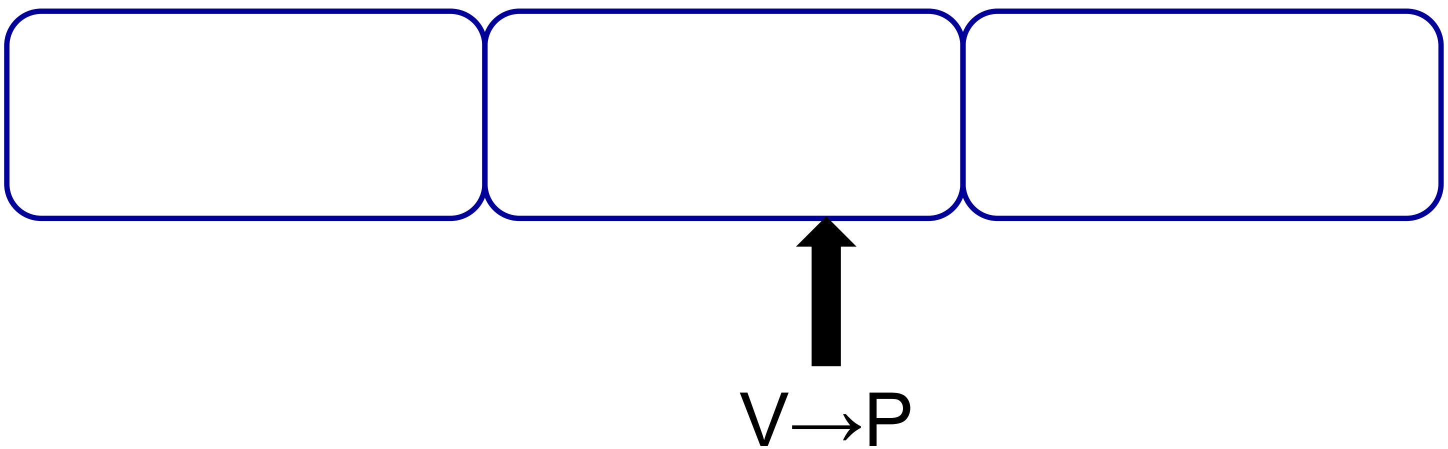
Finding Meta-Data

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



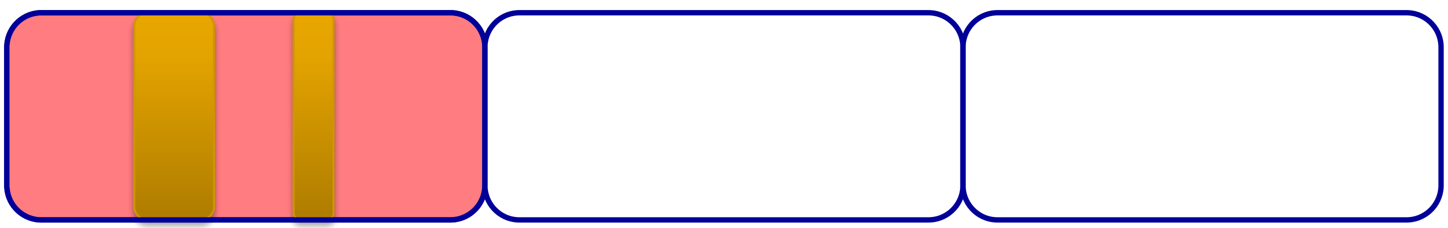
Finding Meta-Data

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



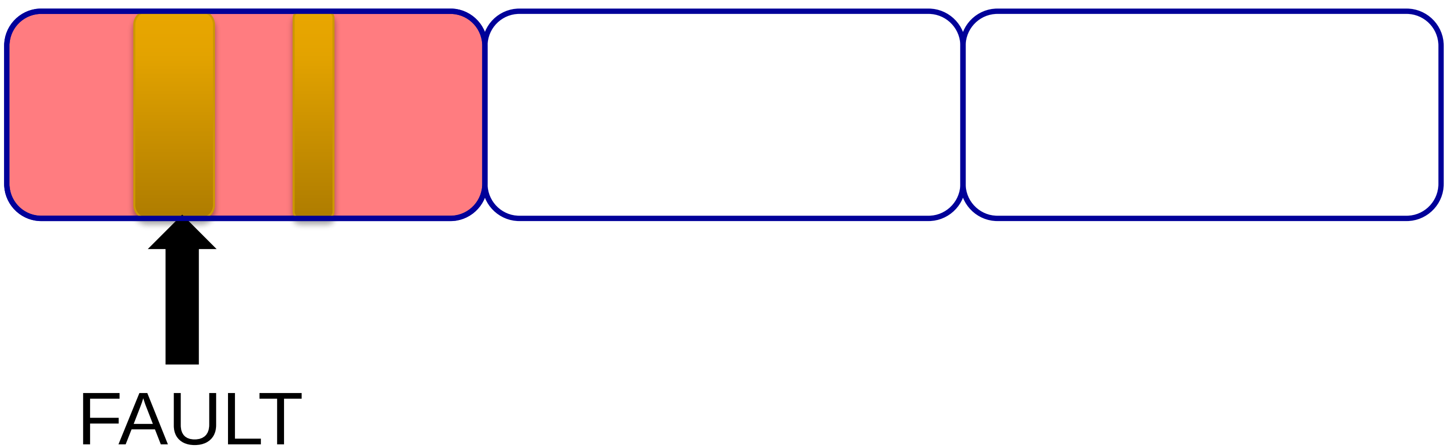
Finding Meta-Data

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



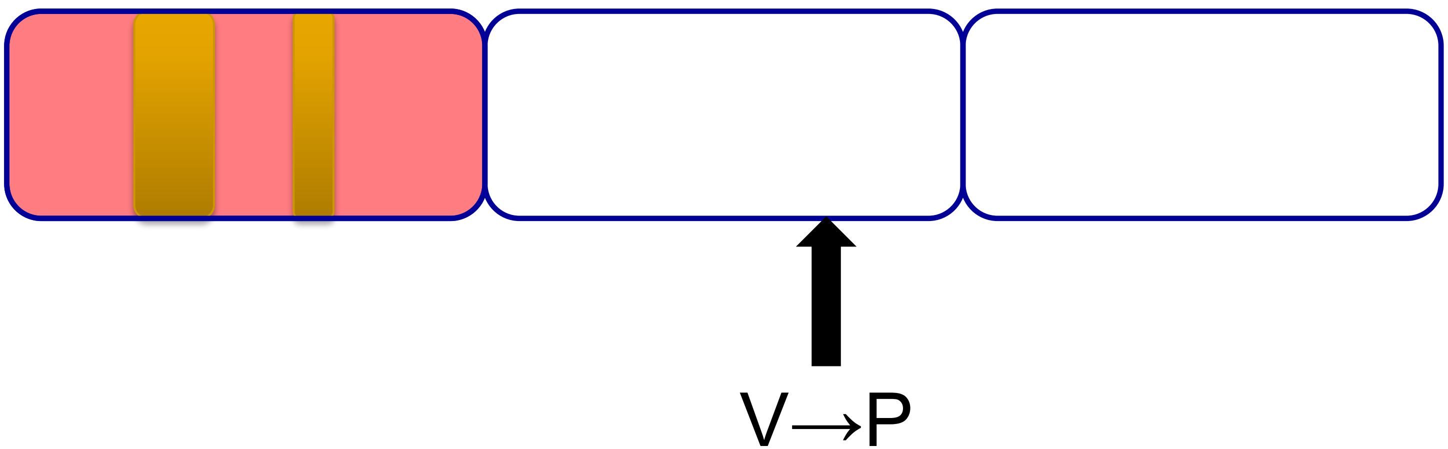
Finding Meta-Data

- 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

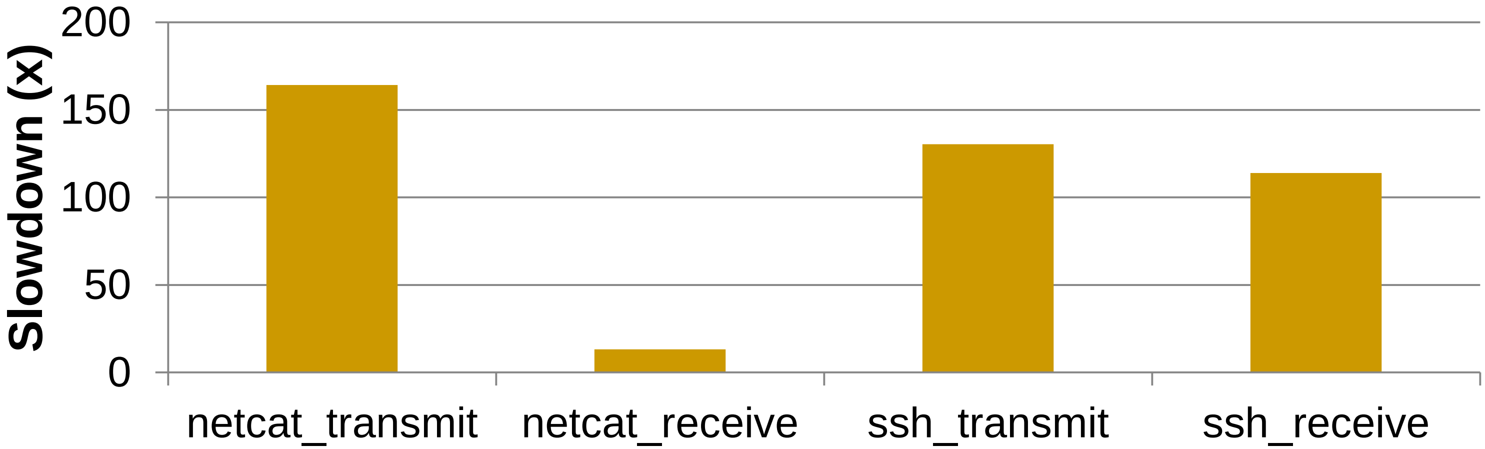


Results by Ho et al.

- lmbench Best Case Results:

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

- Results when everything is tainted:



Outline

- Problem Statement
- Background Information
  - Demand-Driven Dynamic Dataflow Analysis
- Proposed Solutions
  - Demand-Driven Data Race Detection
  - Sampling to Cap Maximum Overheads

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

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)
```

TIME

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

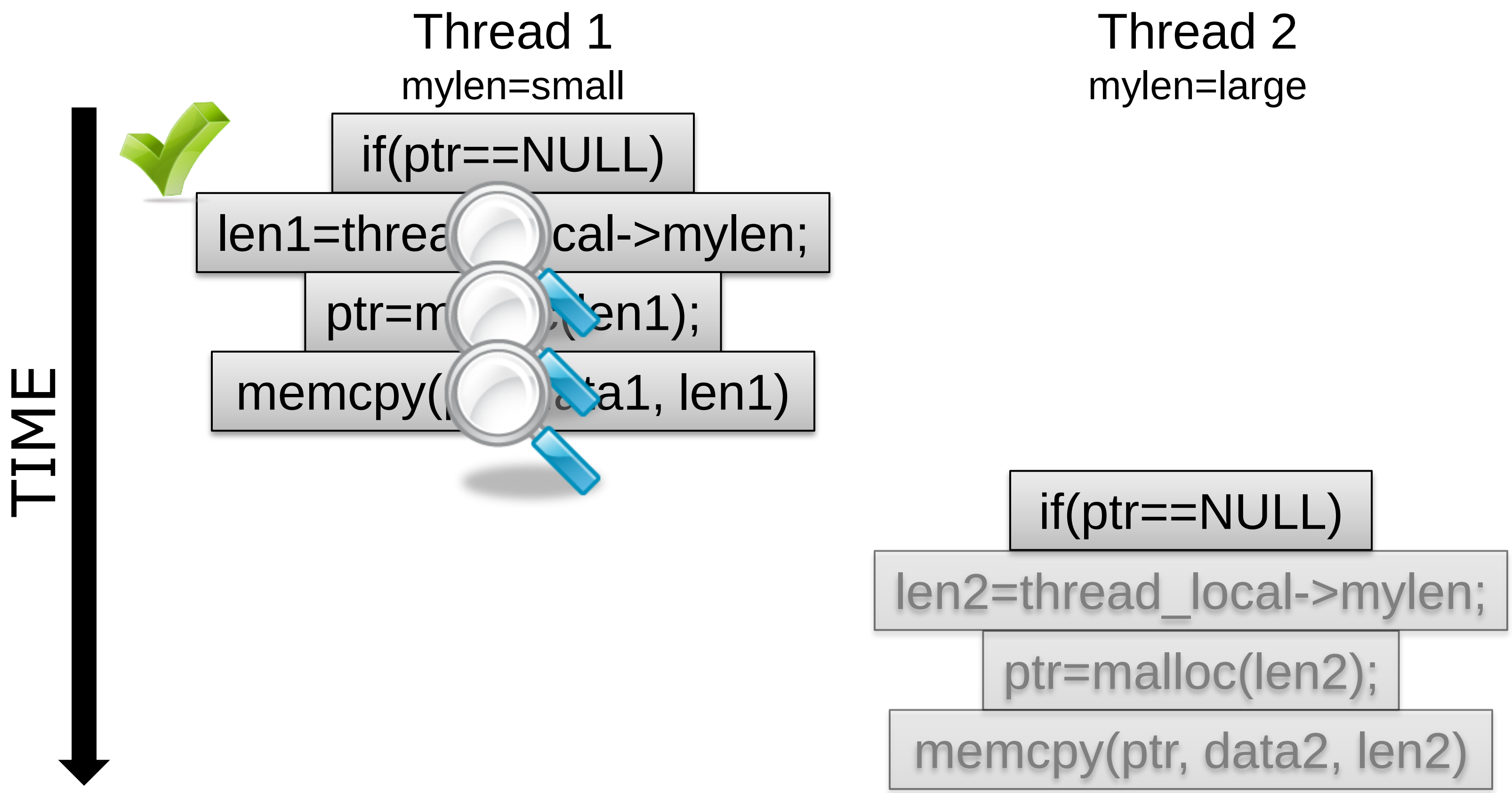
Thread 1
mylen=small

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

Thread 2
mylen=large

```
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

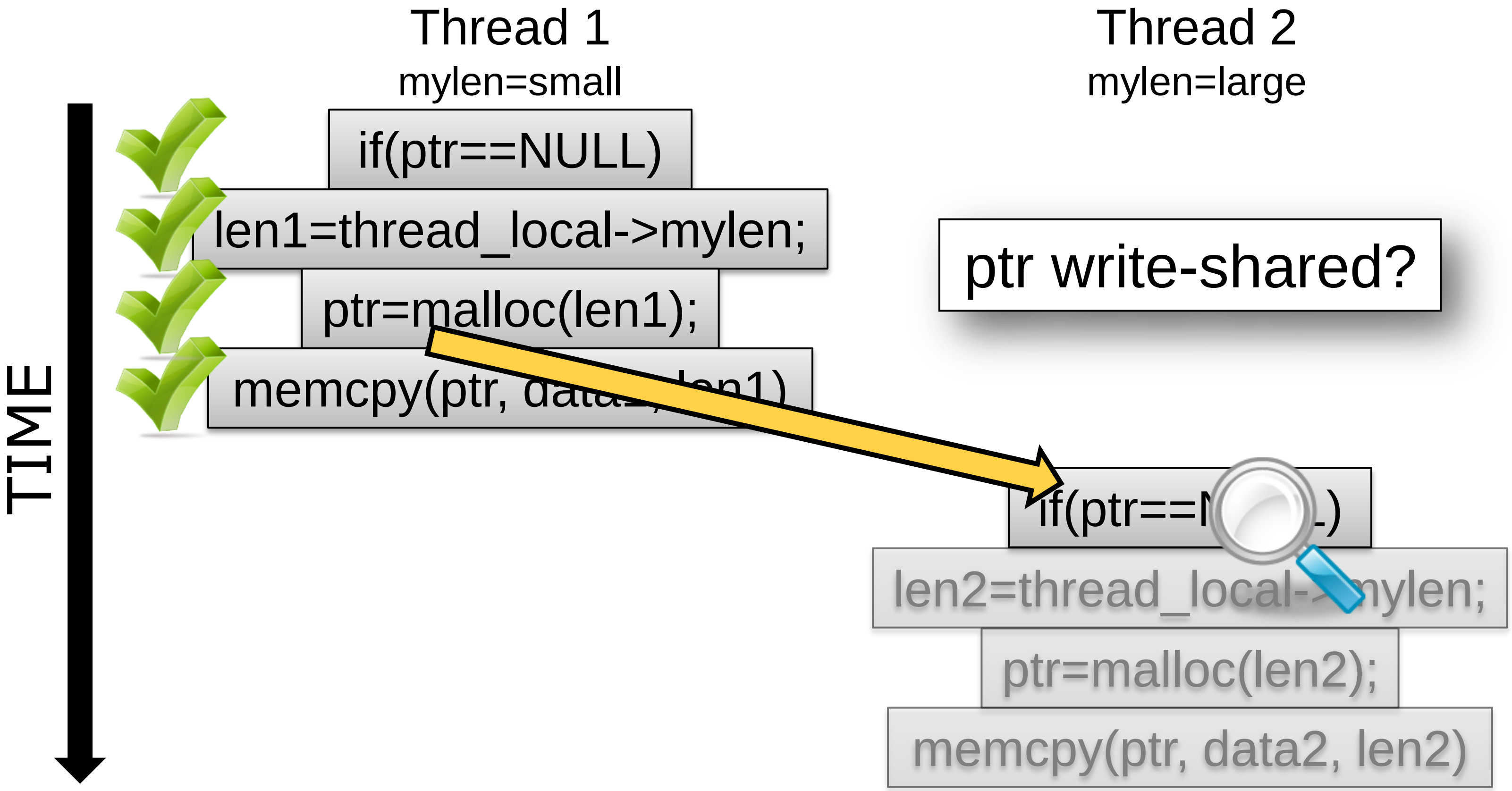
```
if(ptr==NULL)
len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)
```

Thread 2
mylen=large

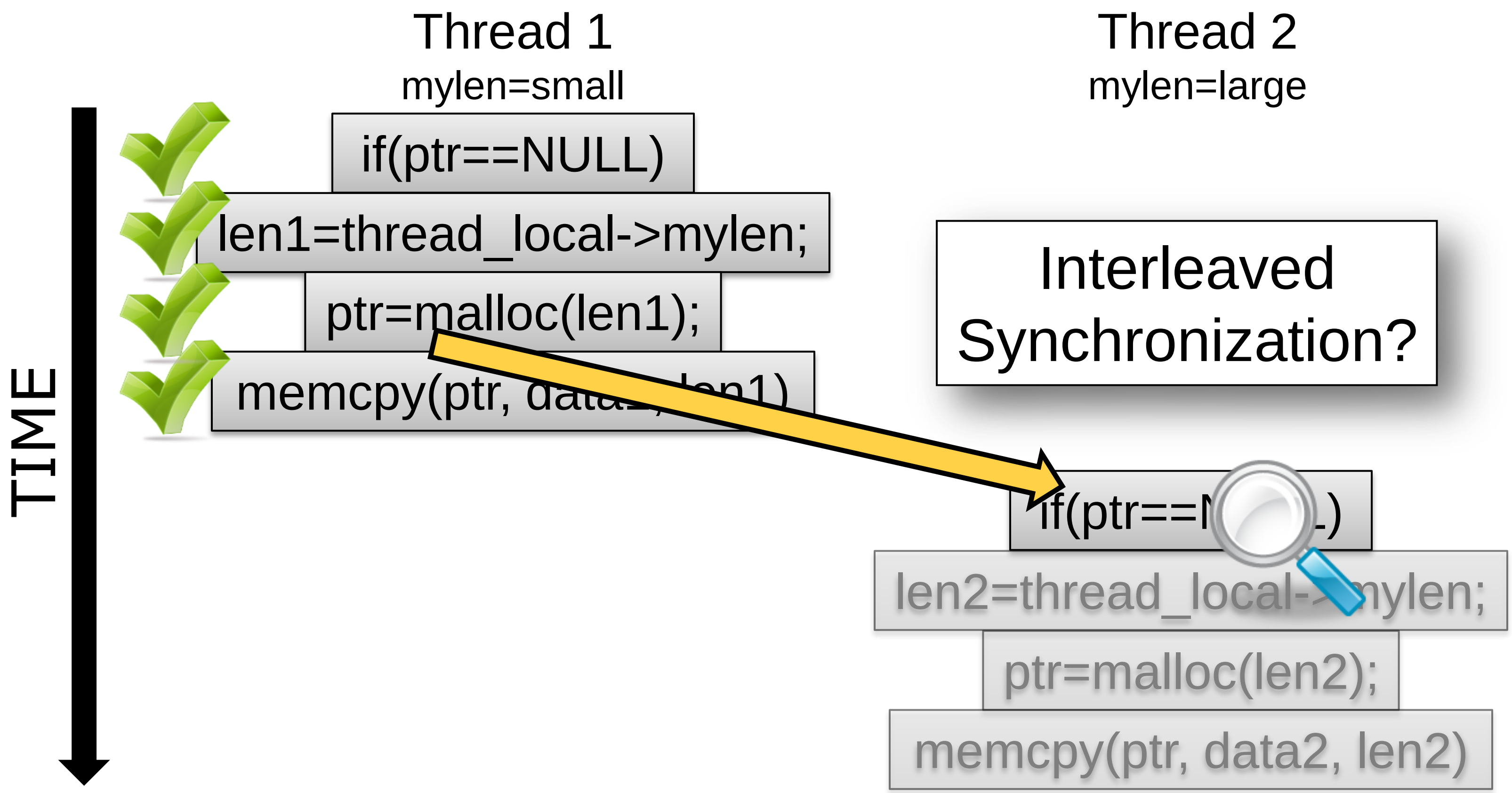
ptr write-shared?

```
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

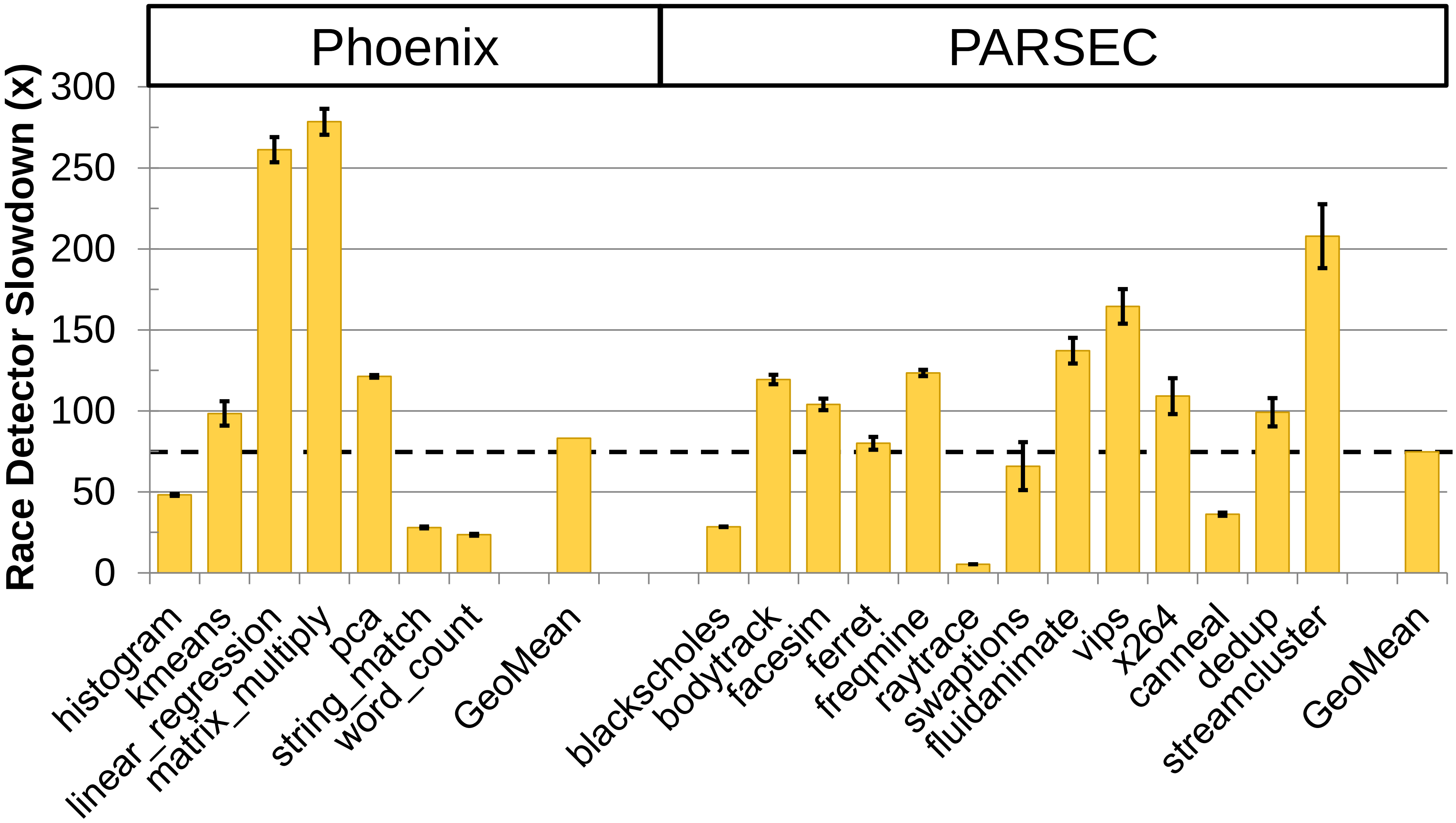
```
if(ptr==NULL)
len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)
```

Thread 2
mylen=large

Interleaved Synchronization?

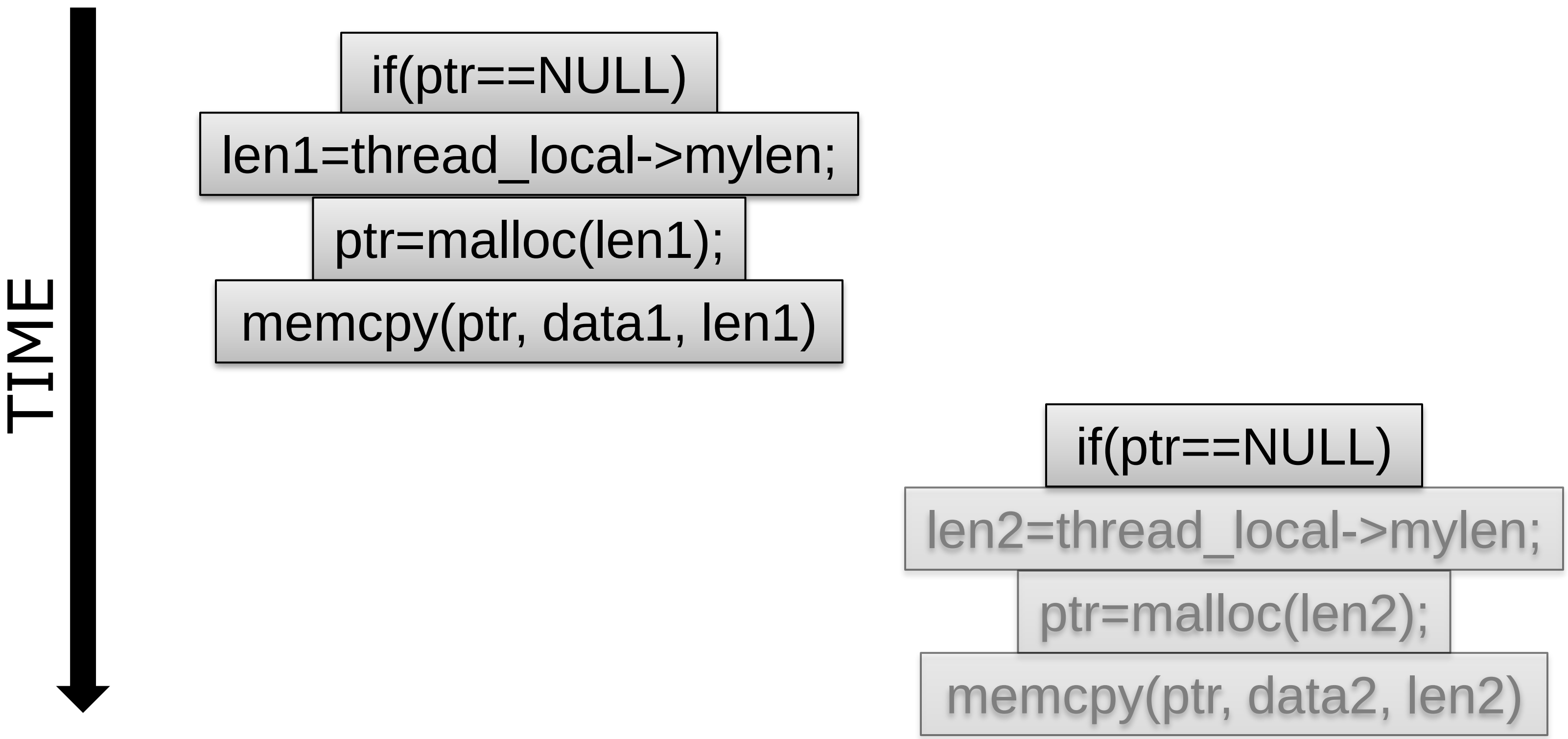
```
if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2)
```

SW Race Detection is Slow



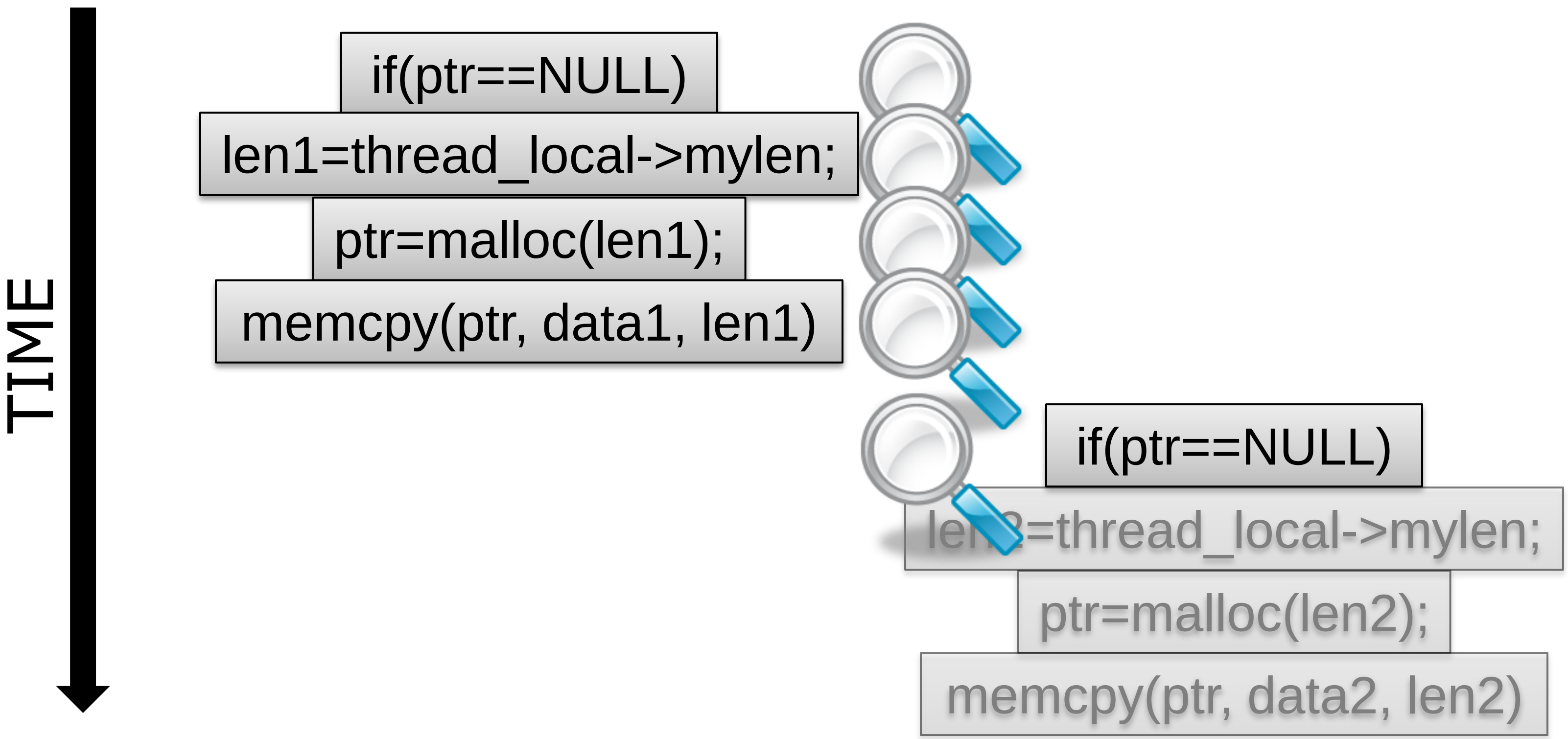
Inter-thread Sharing is What's Important

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



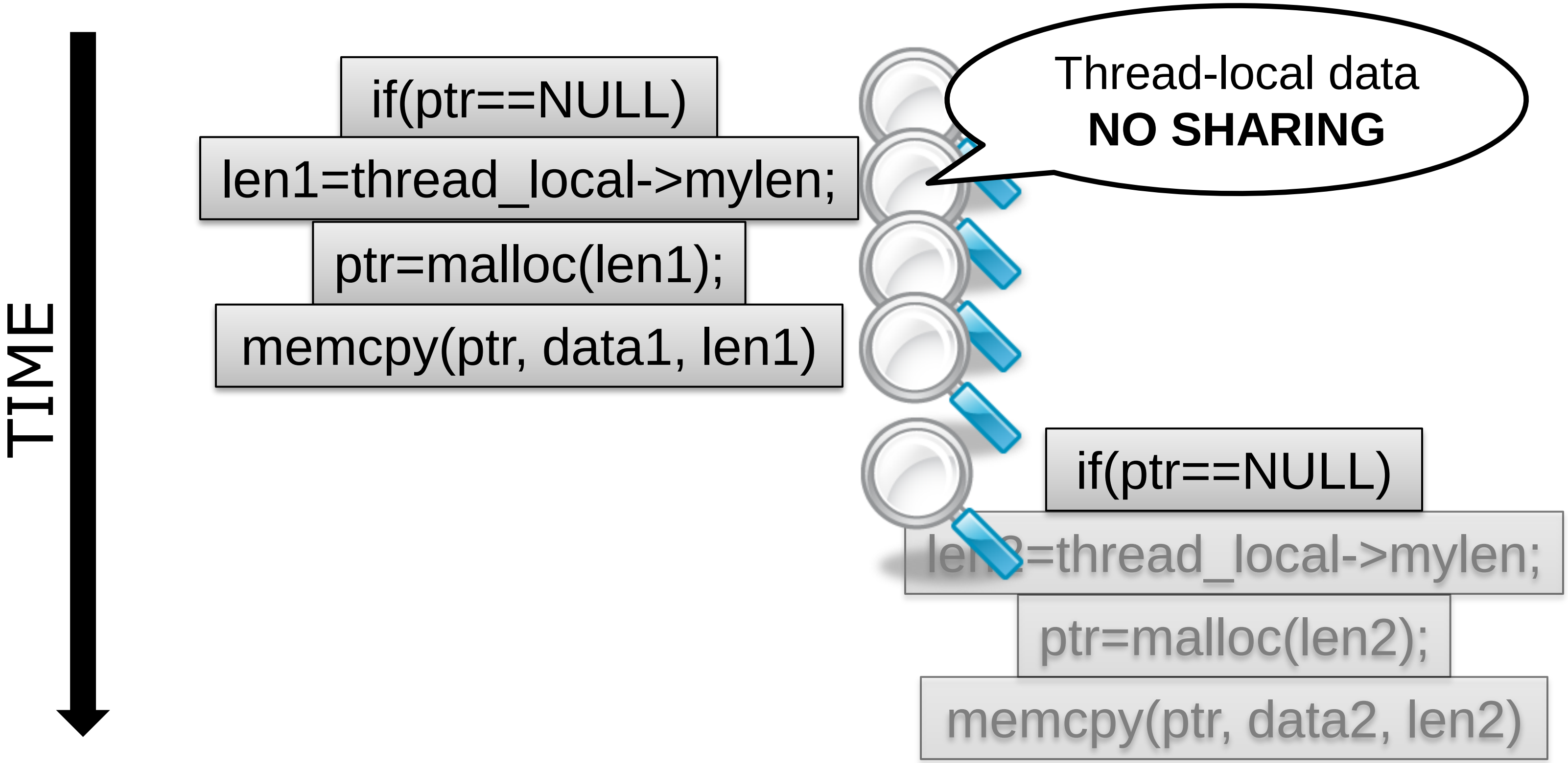
Inter-thread Sharing is What's Important

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



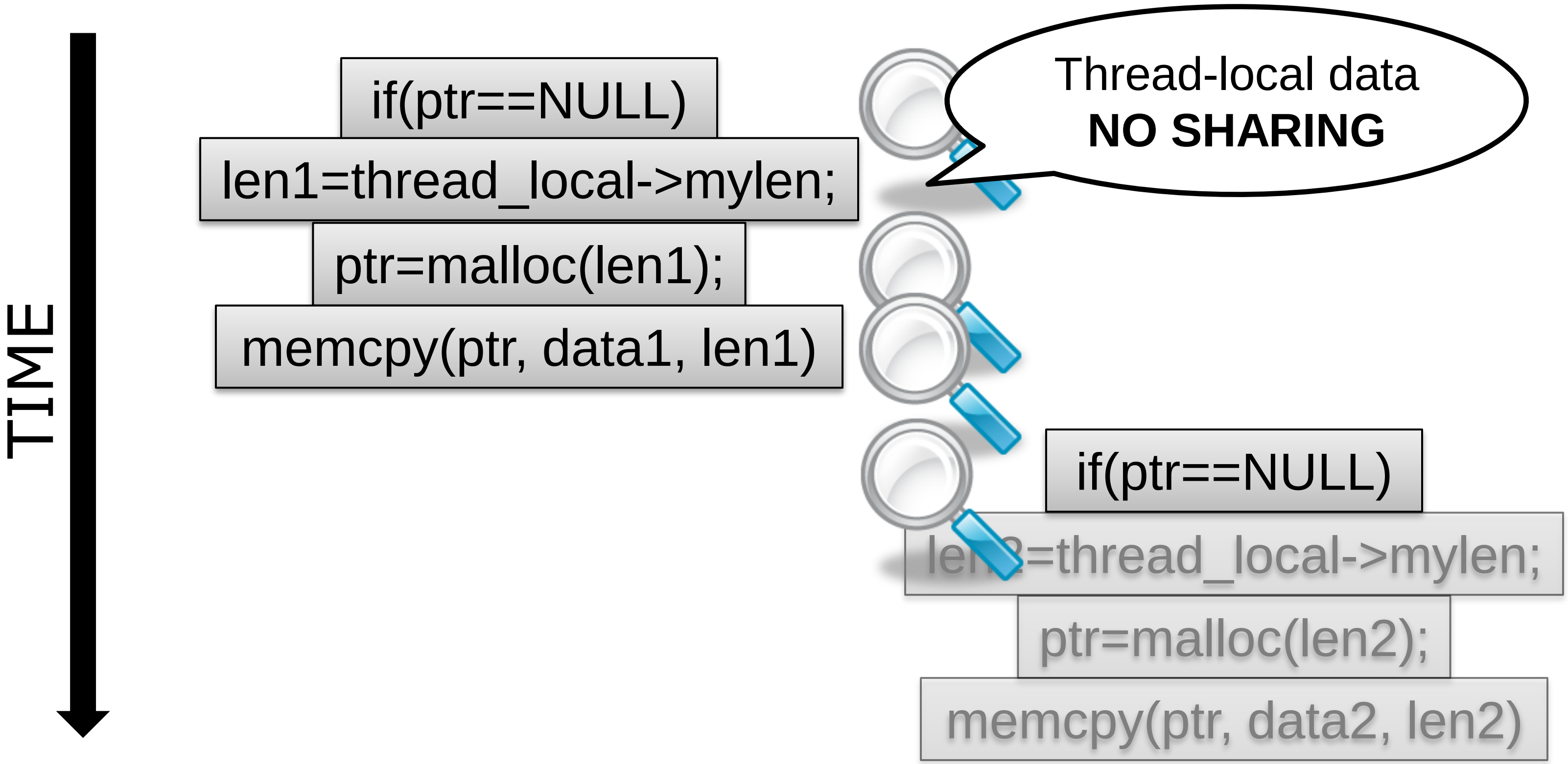
Inter-thread Sharing is What's Important

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



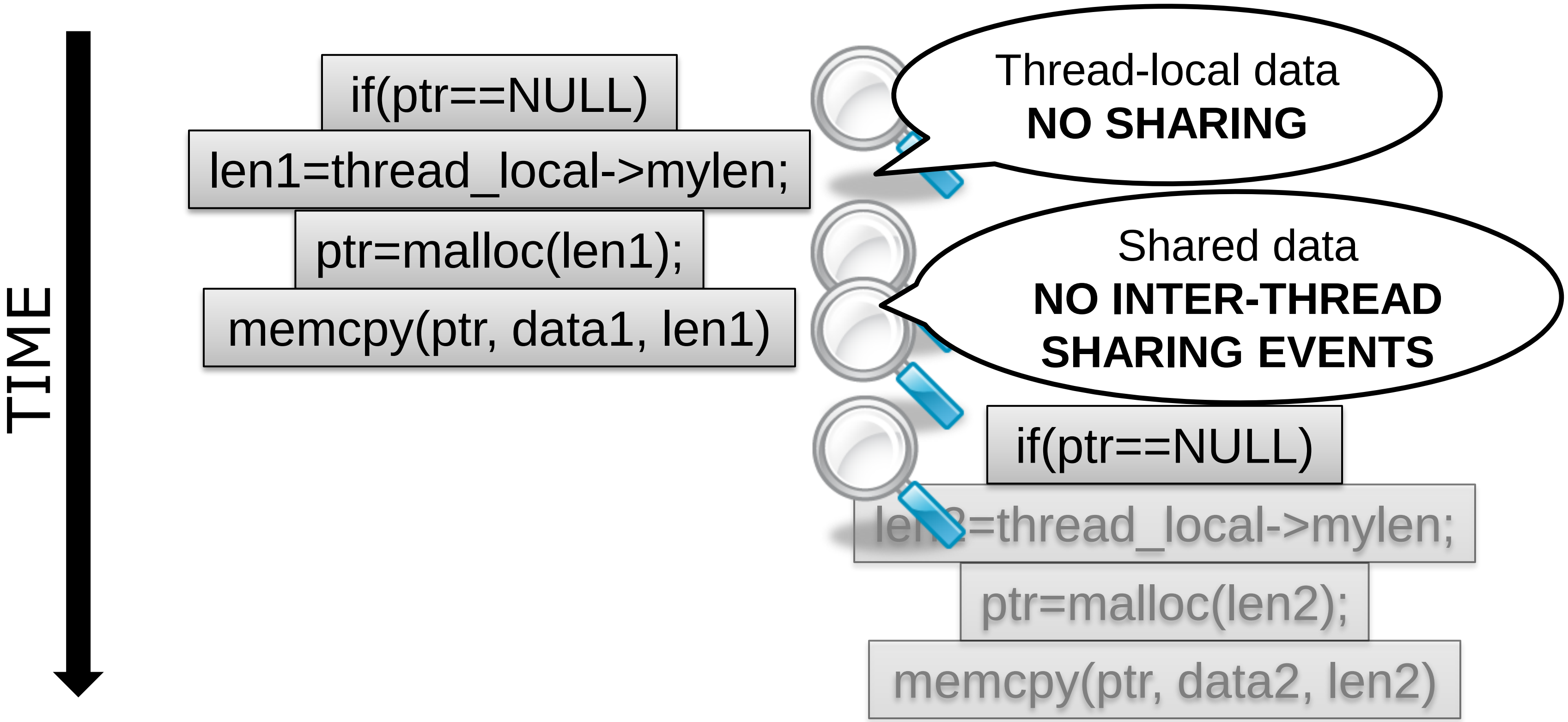
Inter-thread Sharing is What's Important

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



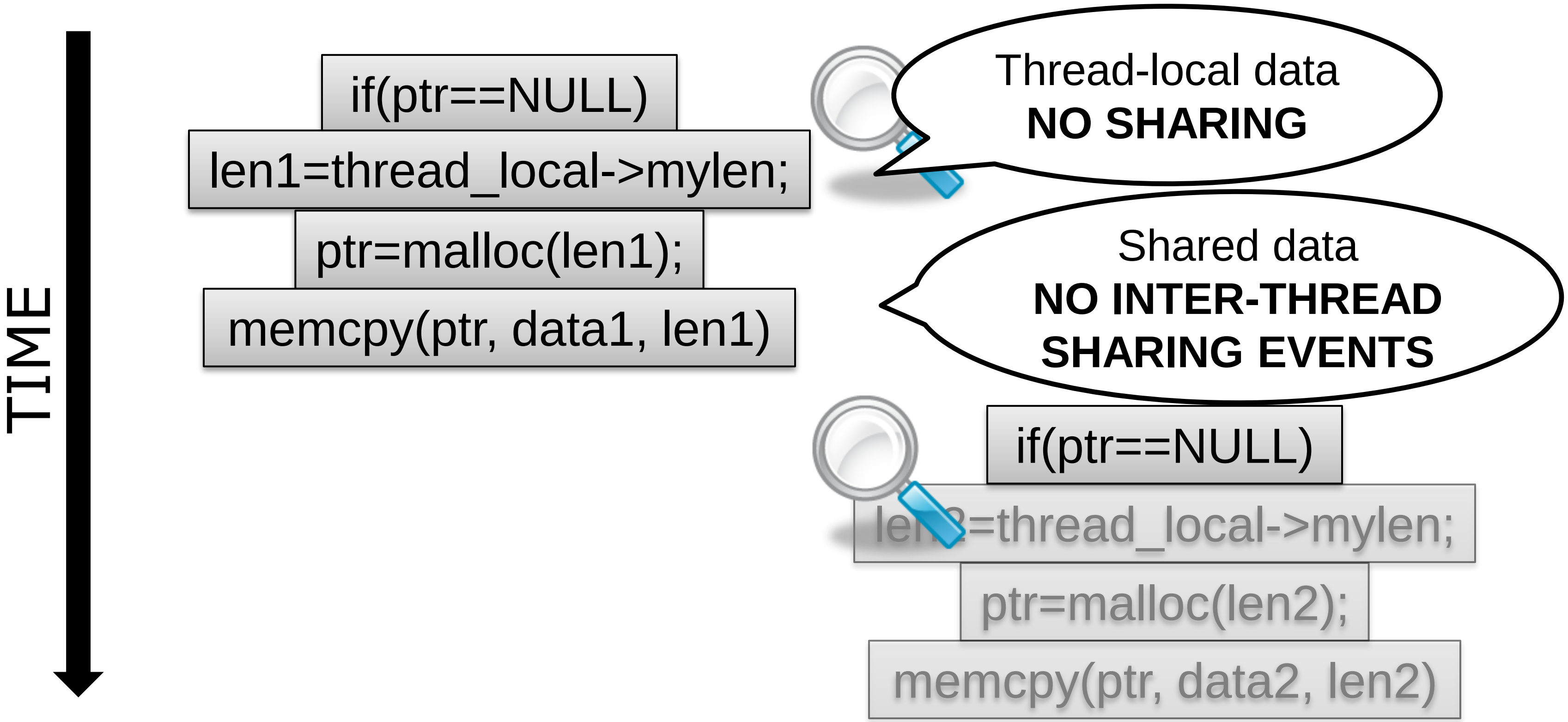
Inter-thread Sharing is What's Important

"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



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

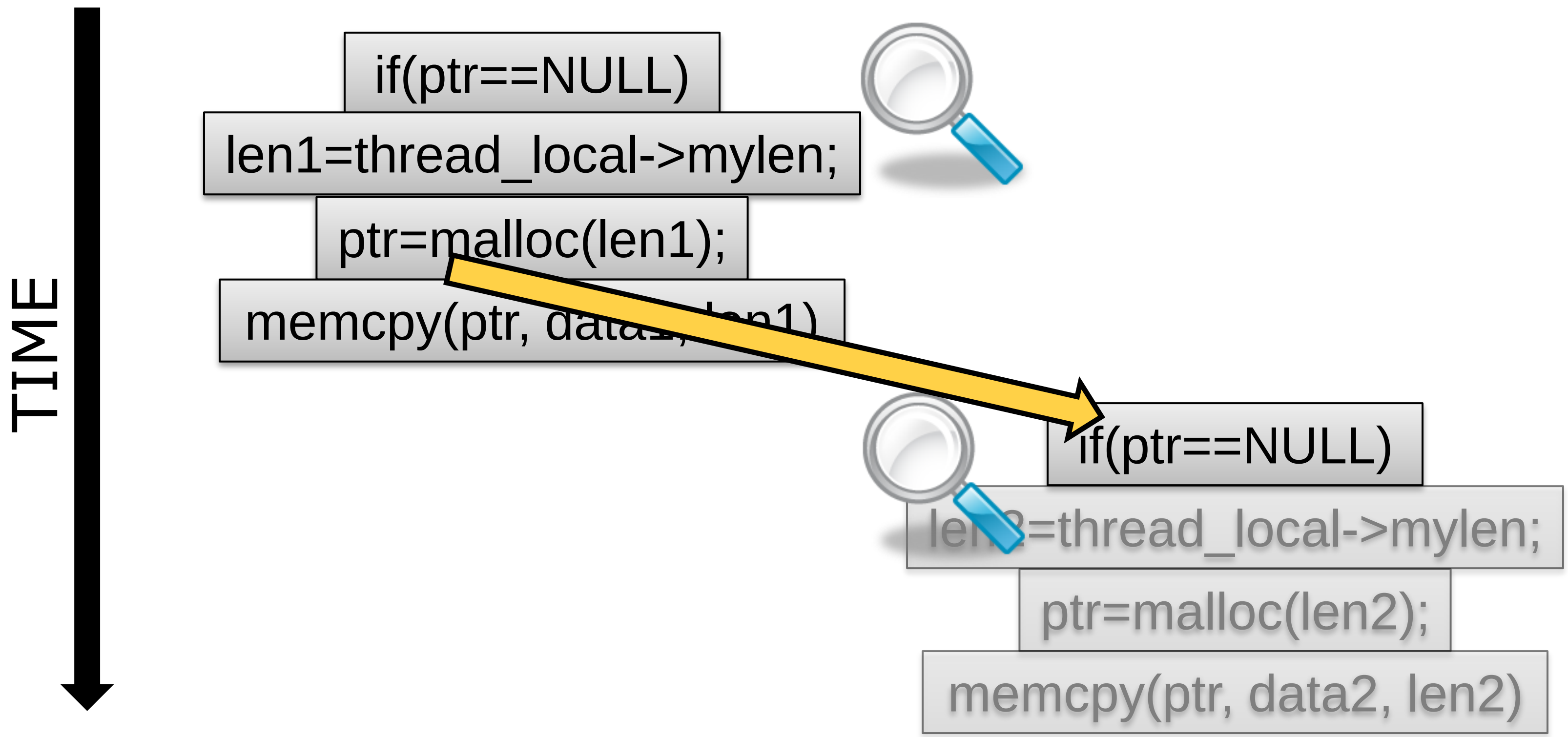
TIME

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

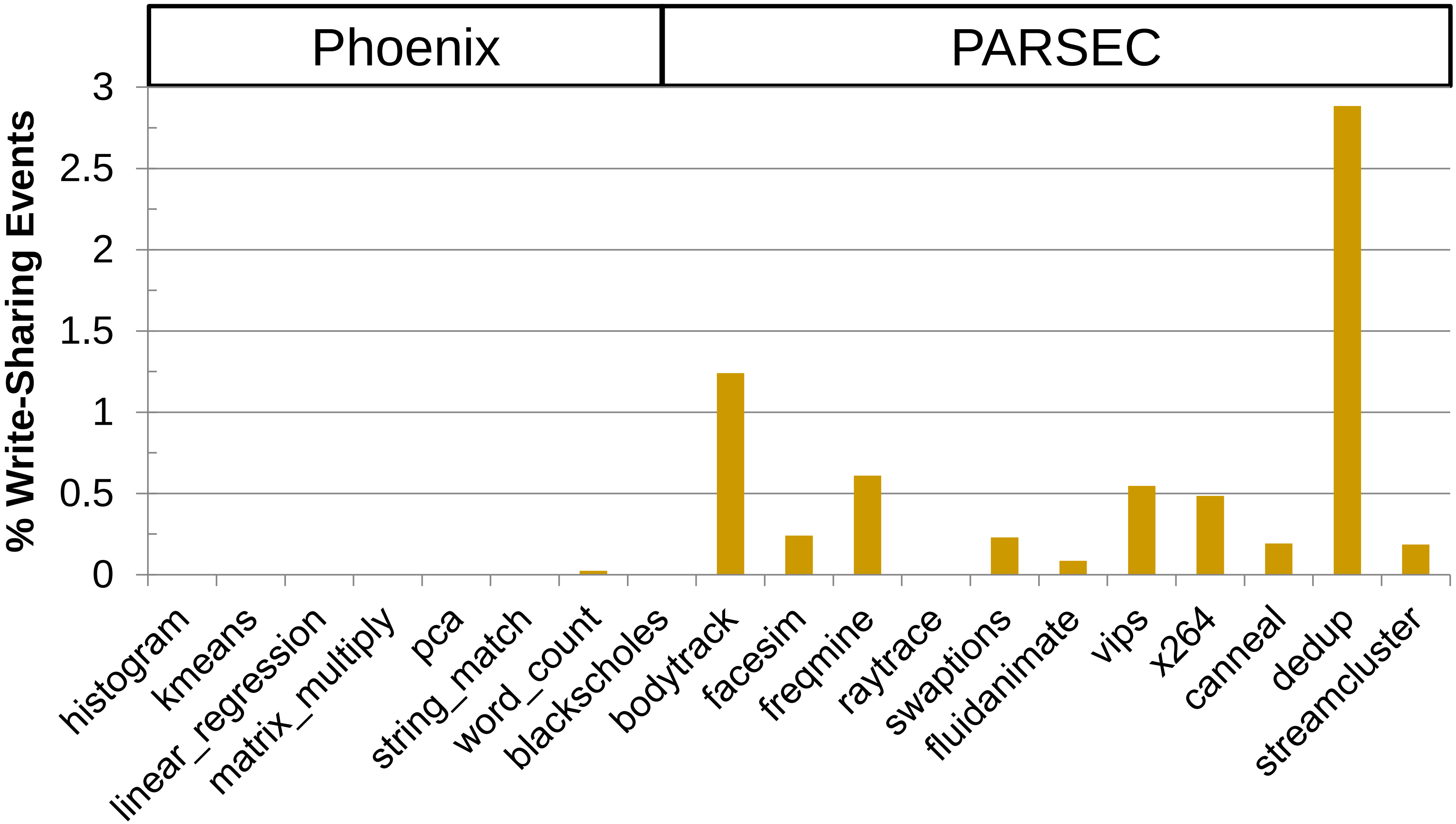
```
if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2)
```

Inter-thread Sharing is What's Important

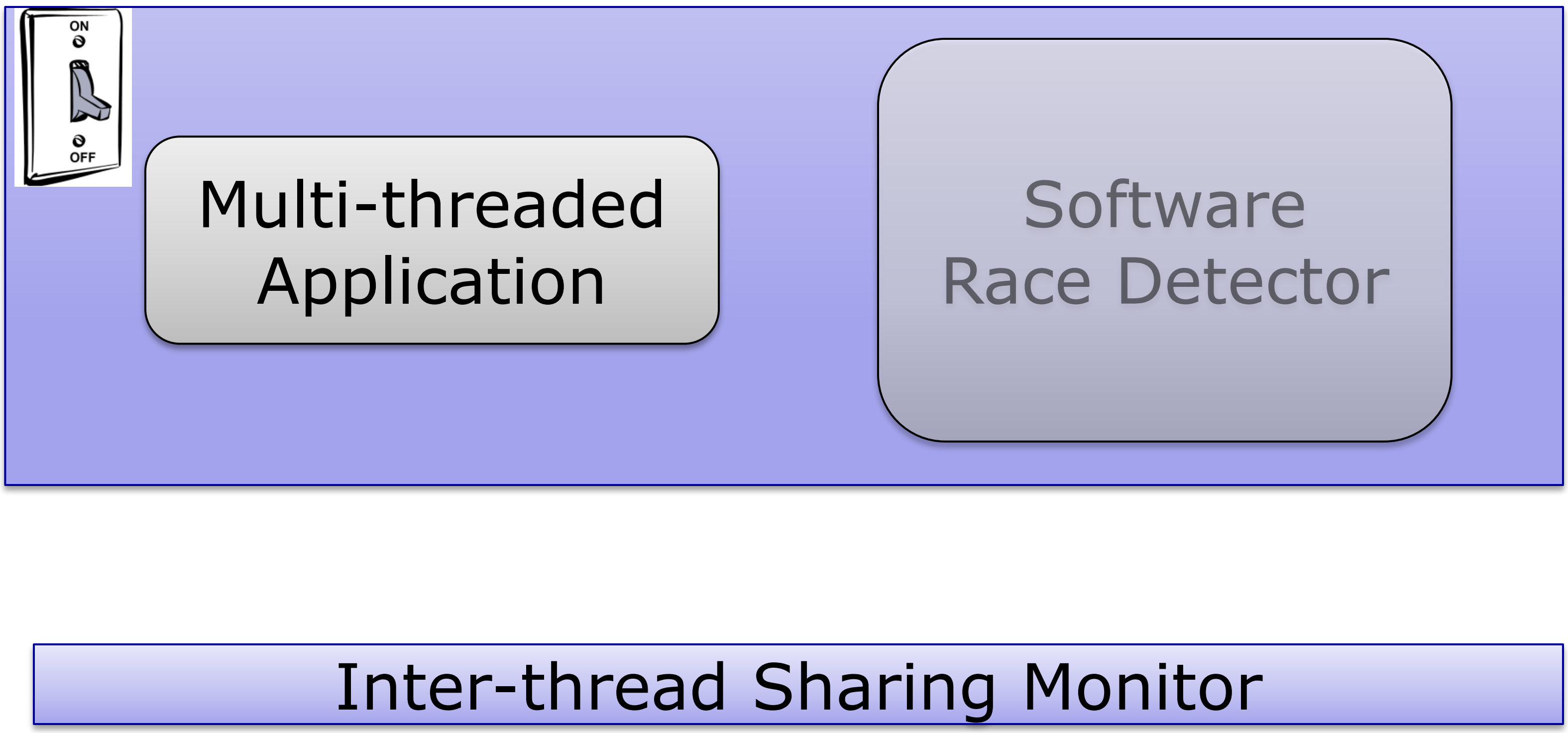
"Data races ... are failures in programs that **access and update shared data** in critical sections" – Netzer & Miller, 1992



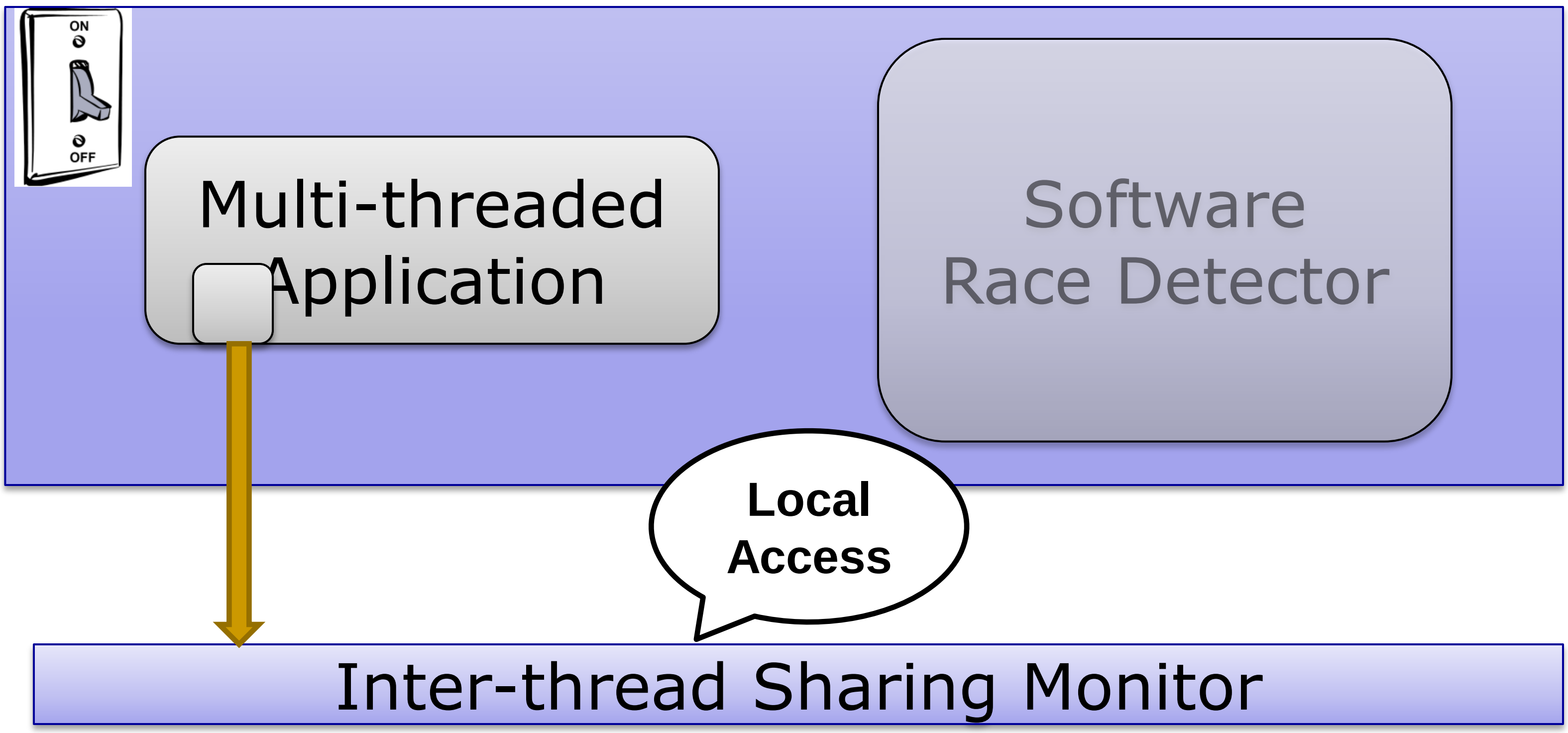
Very Little Inter-Thread Sharing



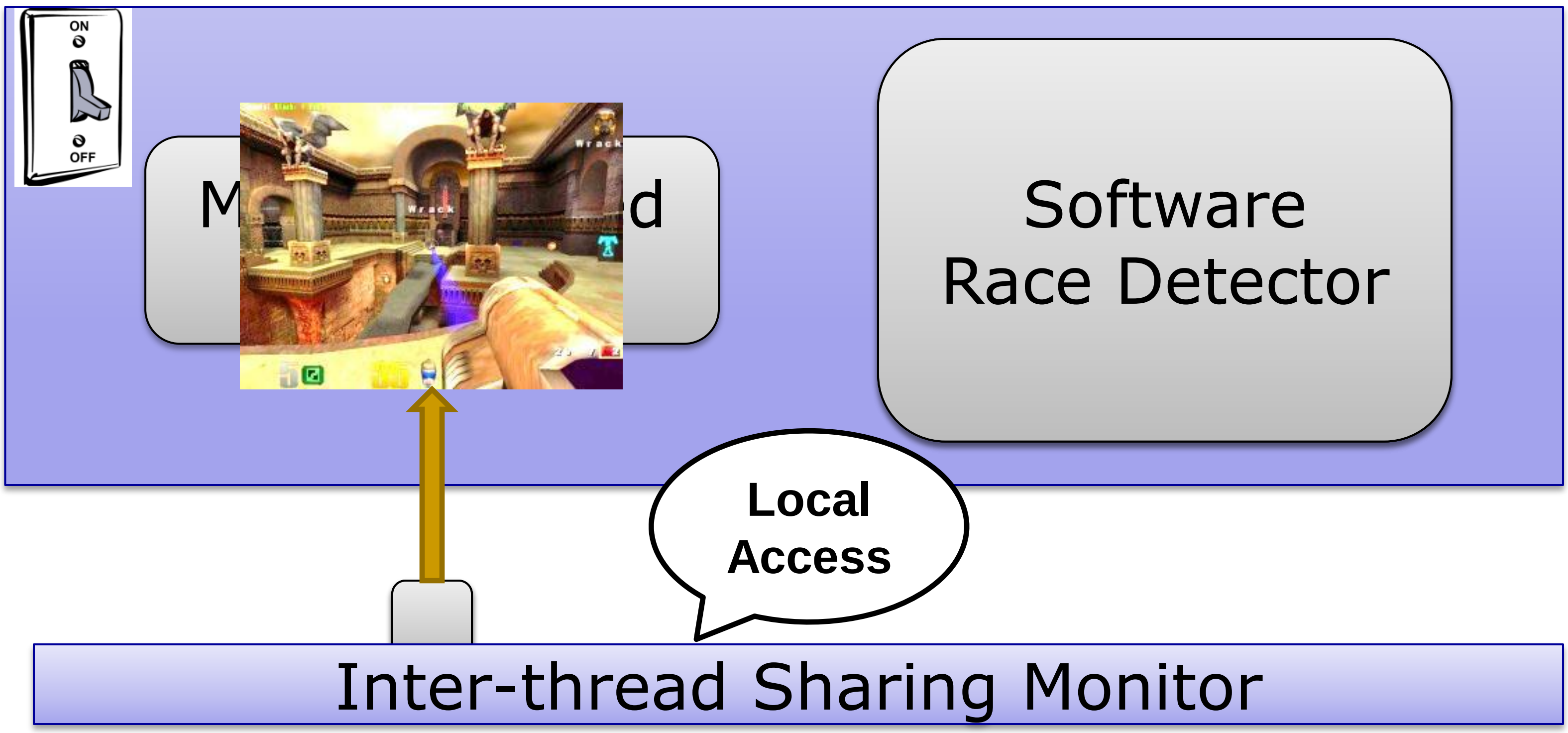
Use Demand-Driven Analysis!



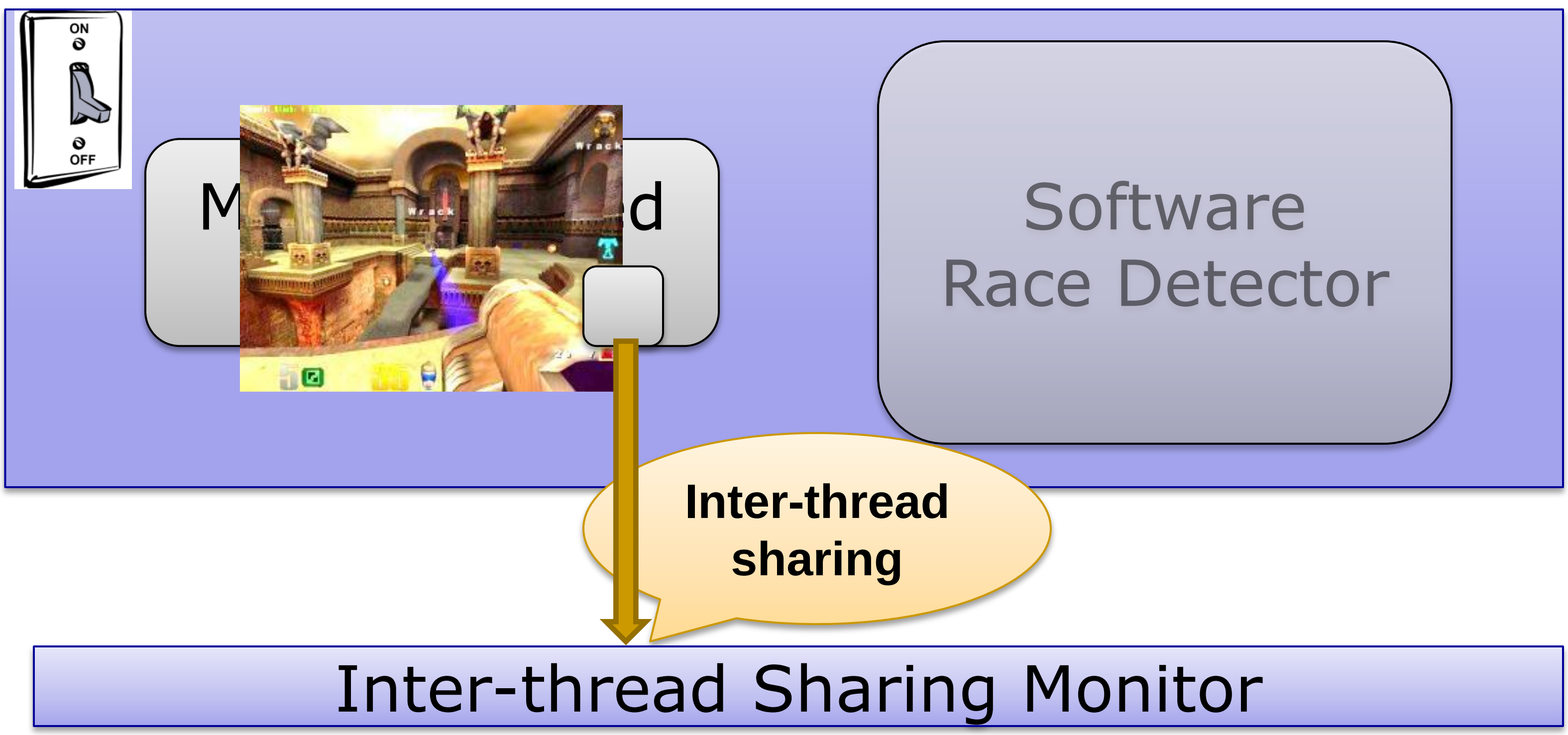
Use Demand-Driven Analysis!



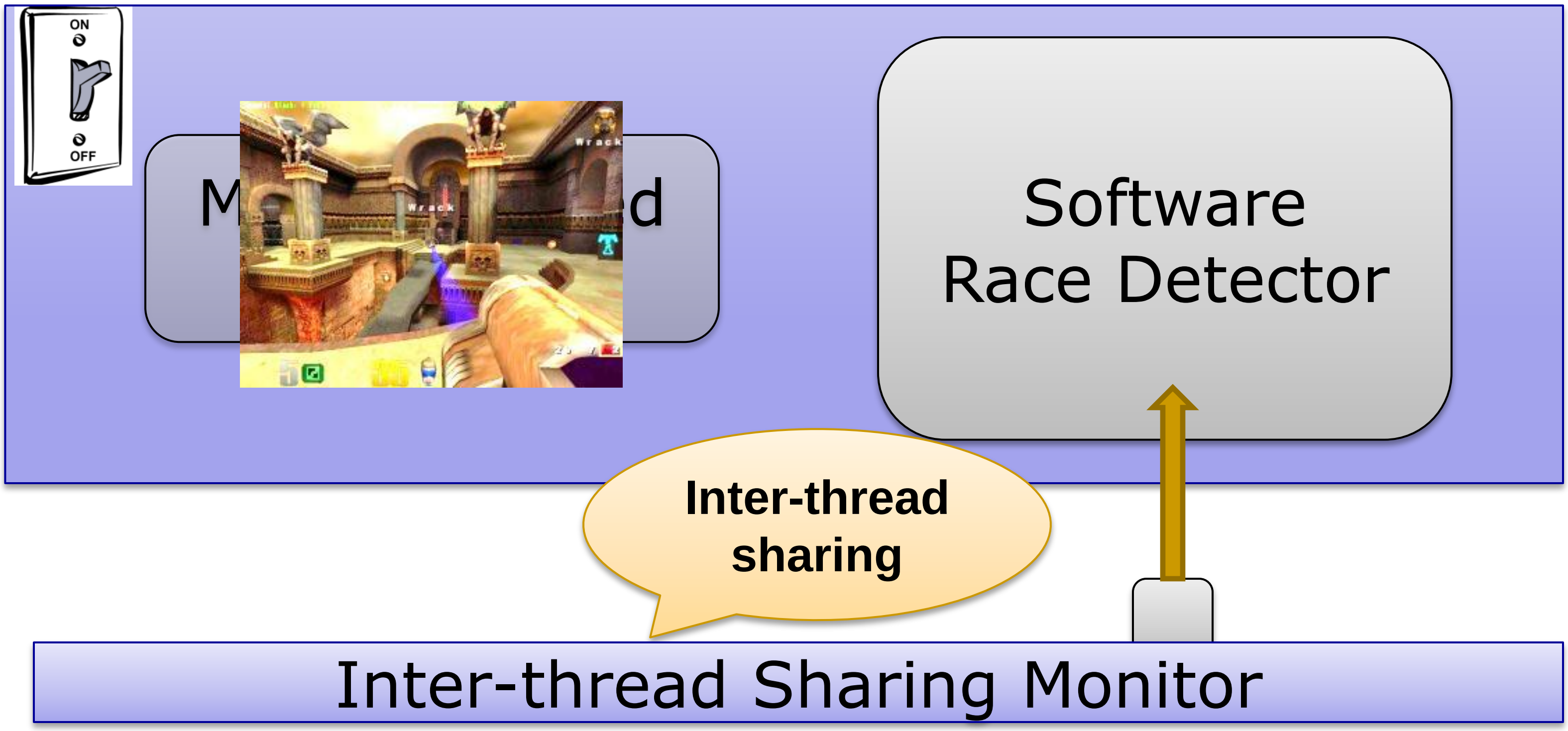
Use Demand-Driven Analysis!



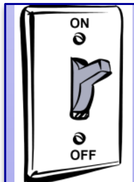
Use Demand-Driven Analysis!



Use Demand-Driven Analysis!

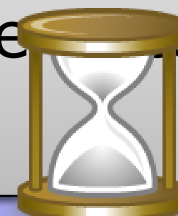


Use Demand-Driven Analysis!



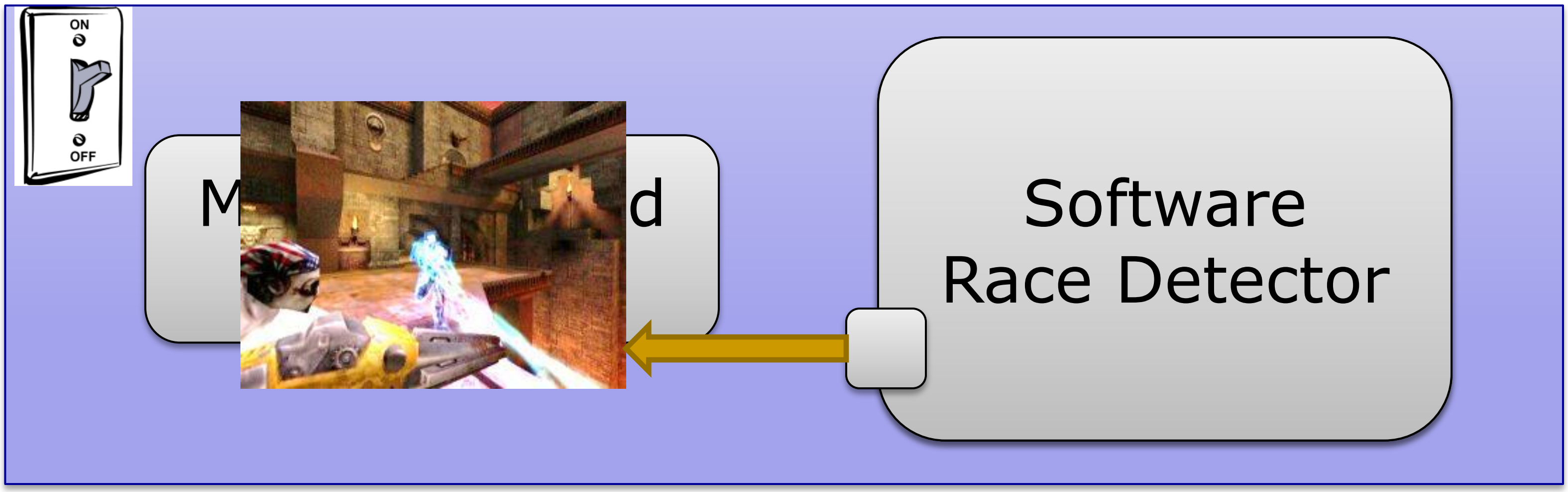
M d

Software
Race ector



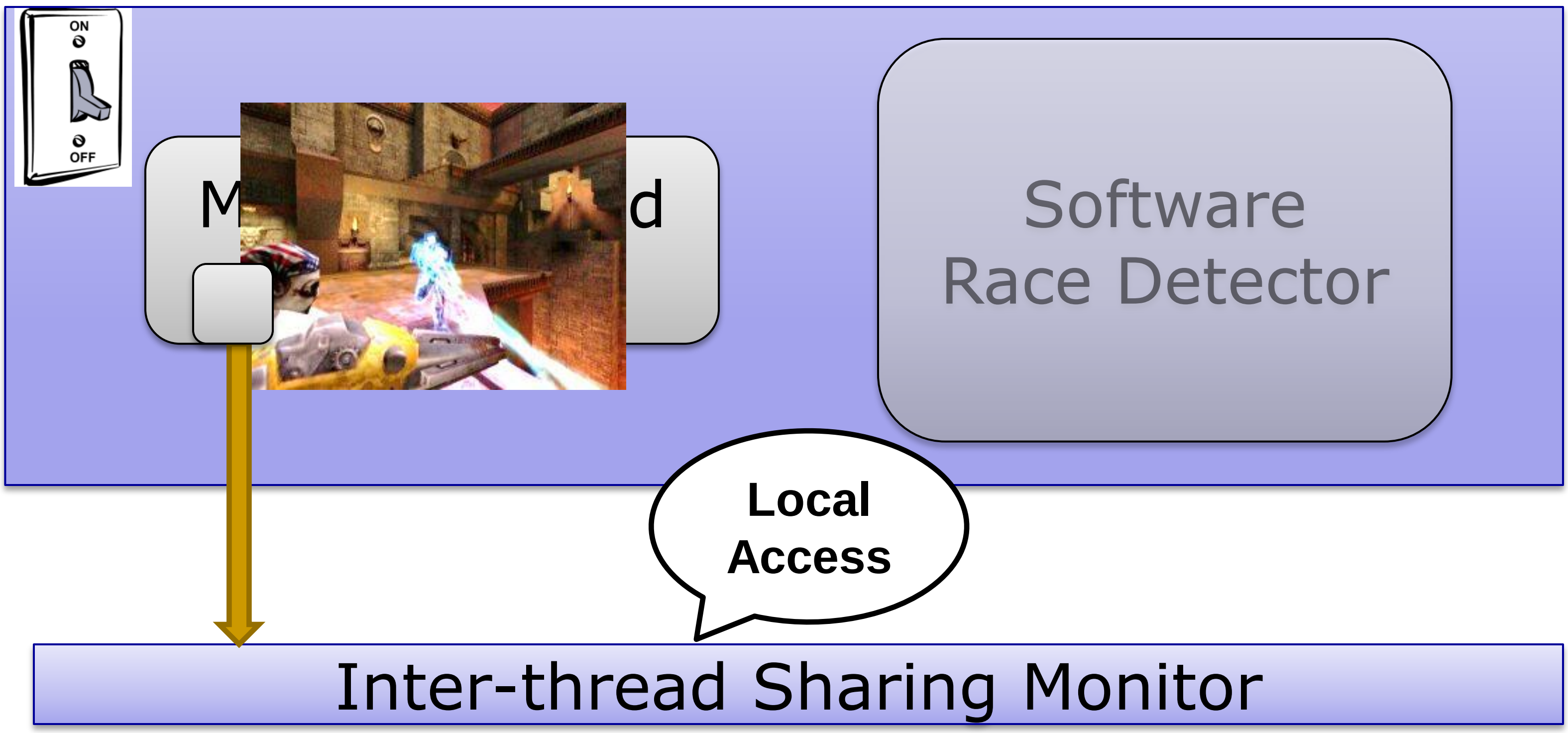
Inter-thread Sharing Monitor

Use Demand-Driven Analysis!

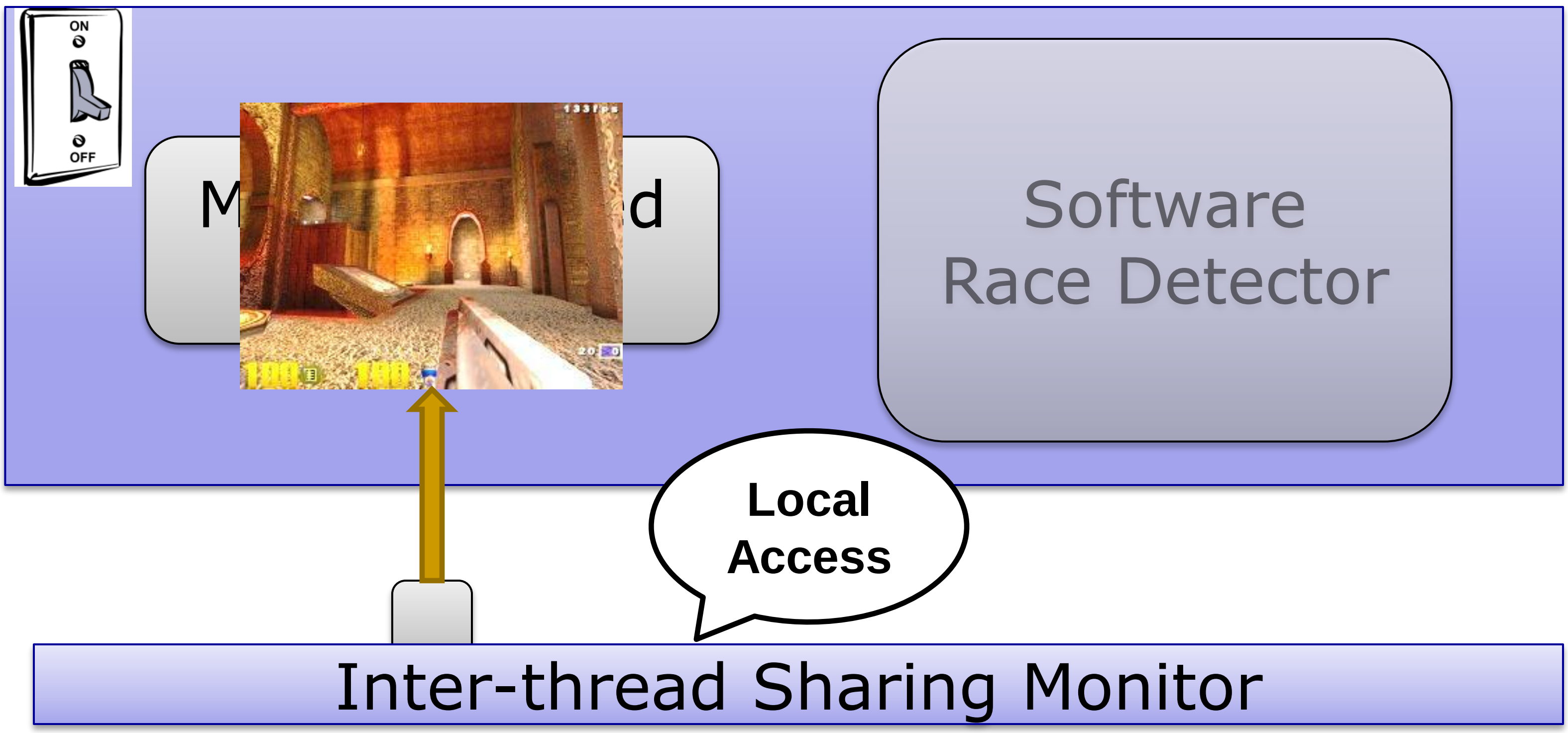


Inter-thread Sharing Monitor

Use Demand-Driven Analysis!



Use Demand-Driven Analysis!

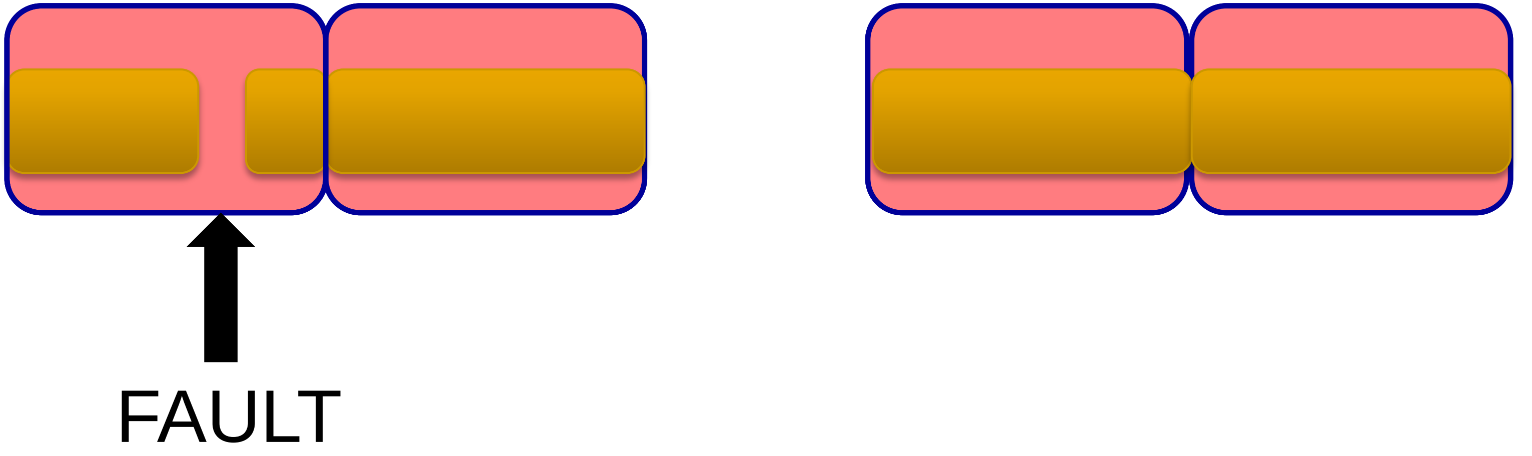


Finding Inter-thread Sharing

- Virtual Memory Watchpoints?

Finding Inter-thread Sharing

- Virtual Memory Watchpoints?

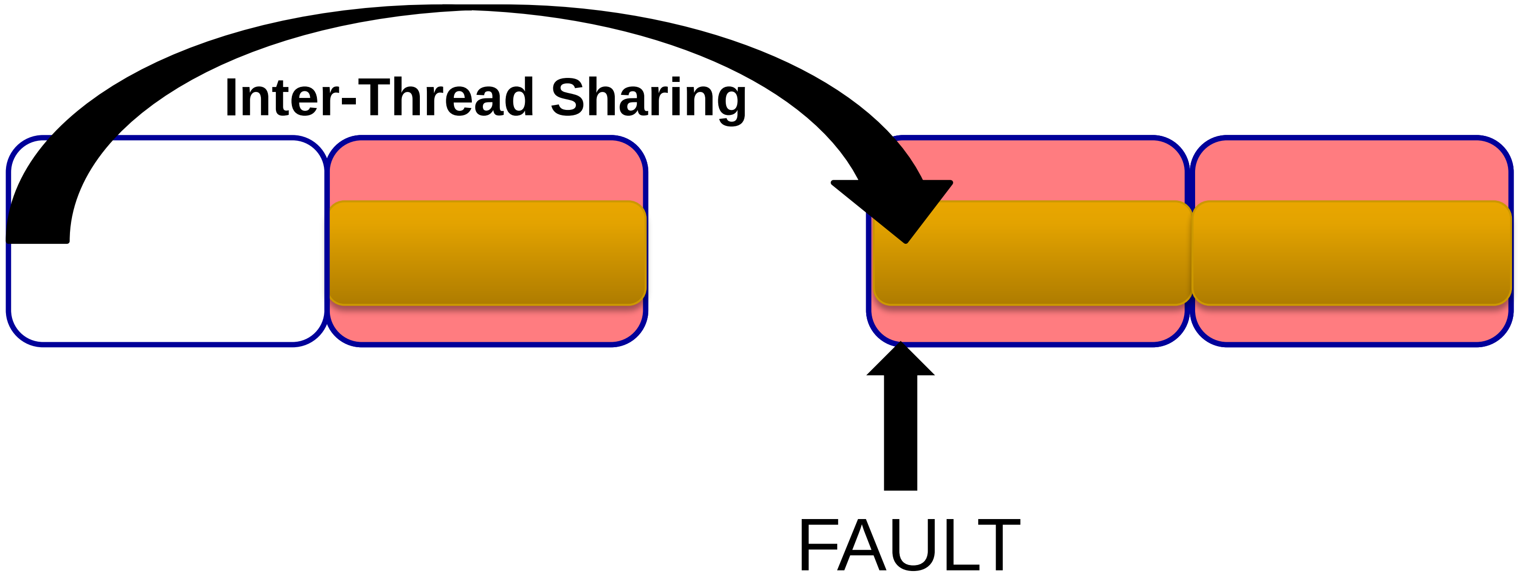


Finding Inter-thread Sharing

- 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

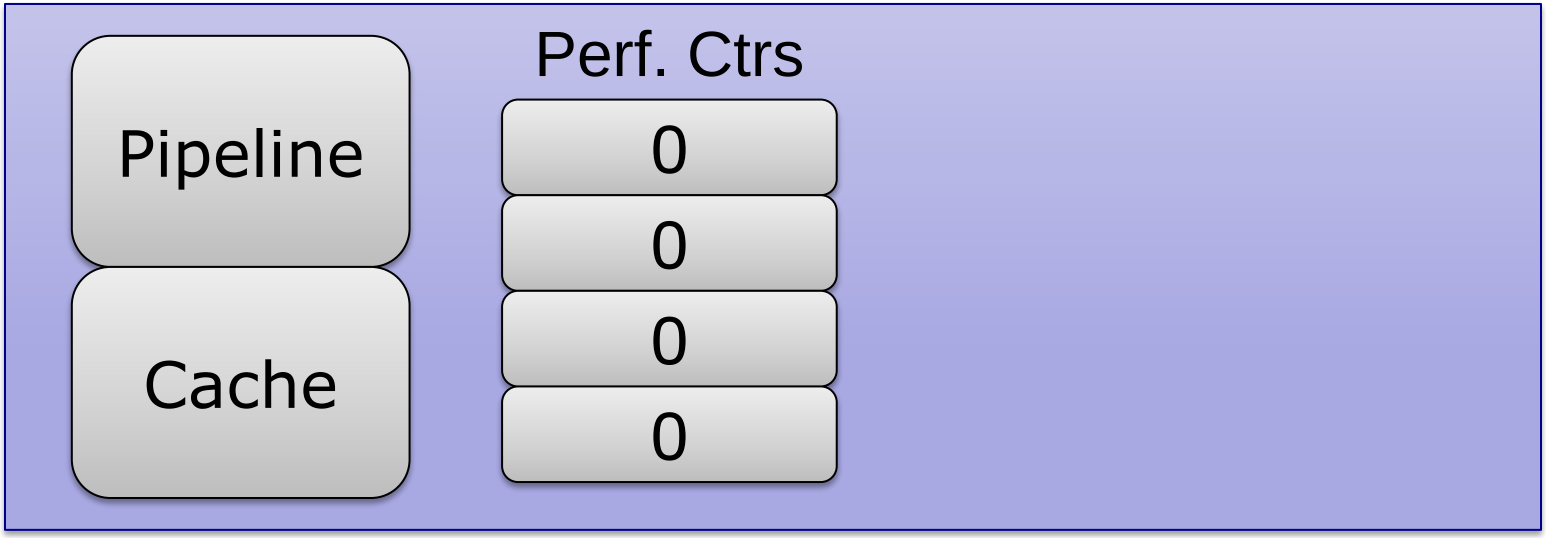
Finding Inter-thread Sharing

- 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

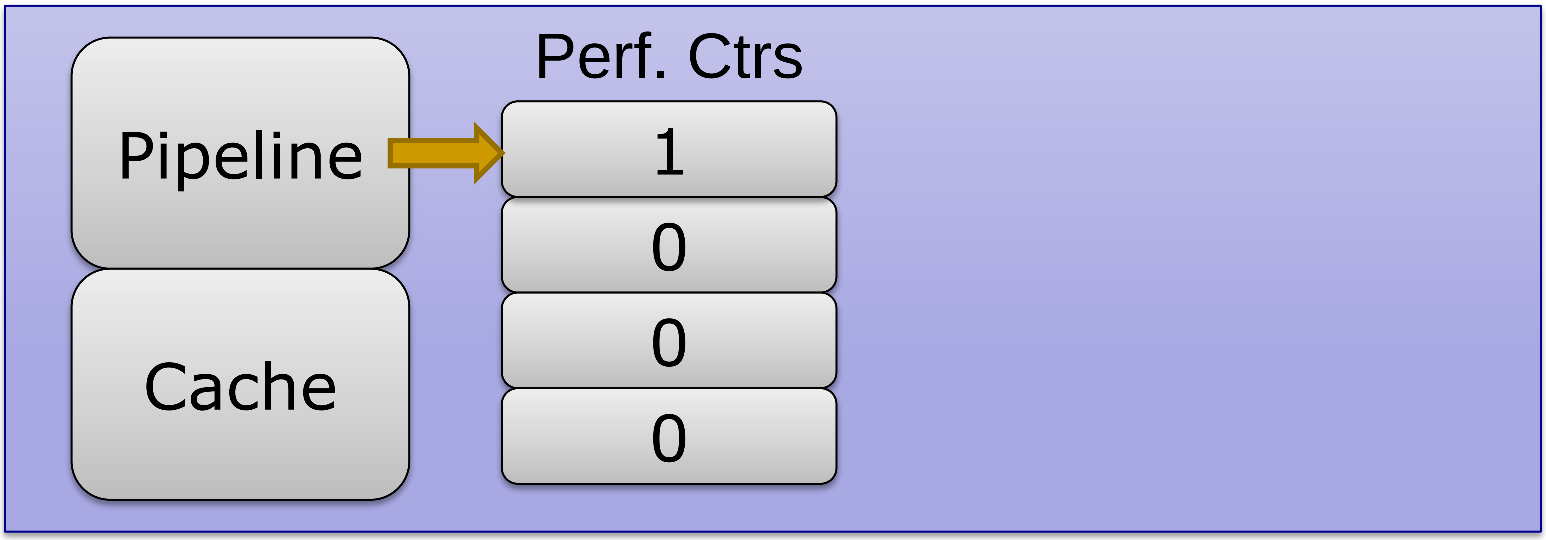
Hardware Sharing Detector

- Hardware Performance Counters



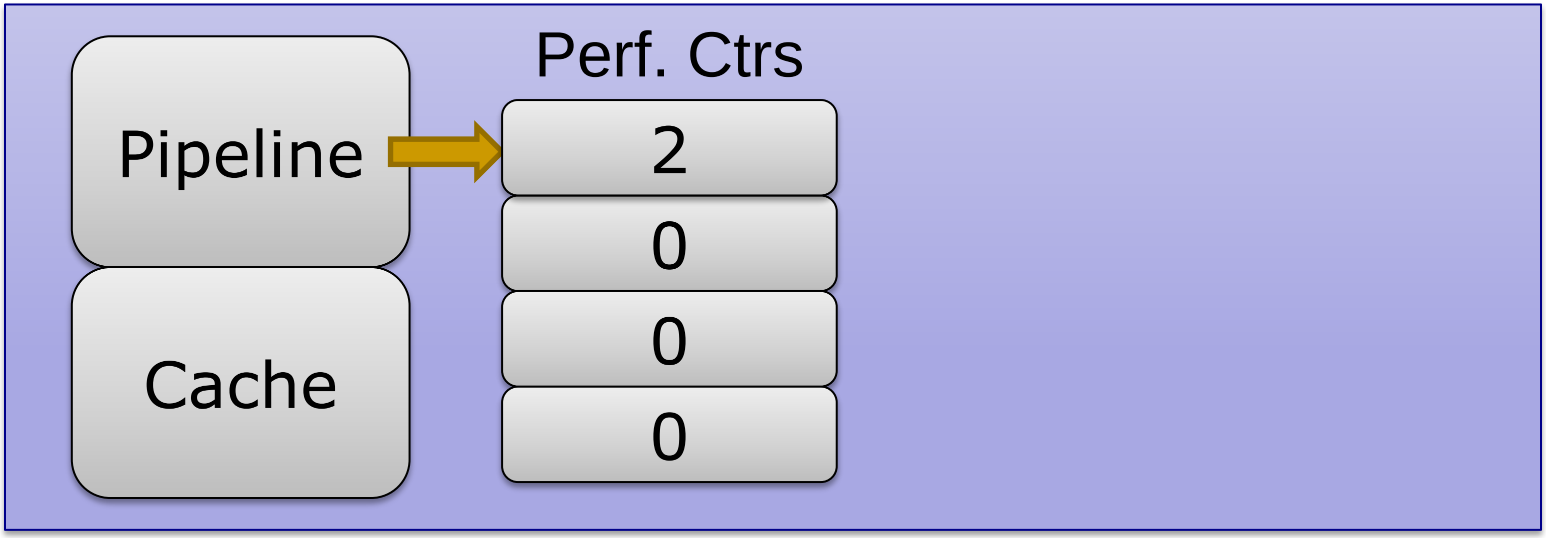
Hardware Sharing Detector

- Hardware Performance Counters



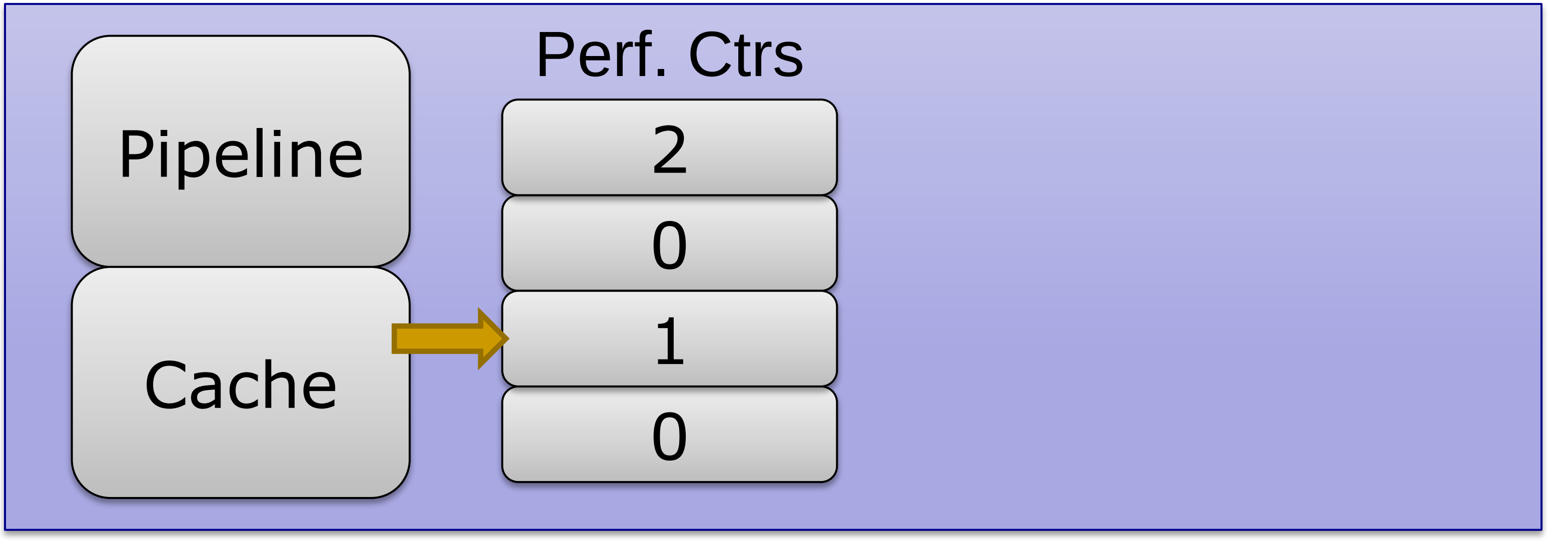
Hardware Sharing Detector

- Hardware Performance Counters



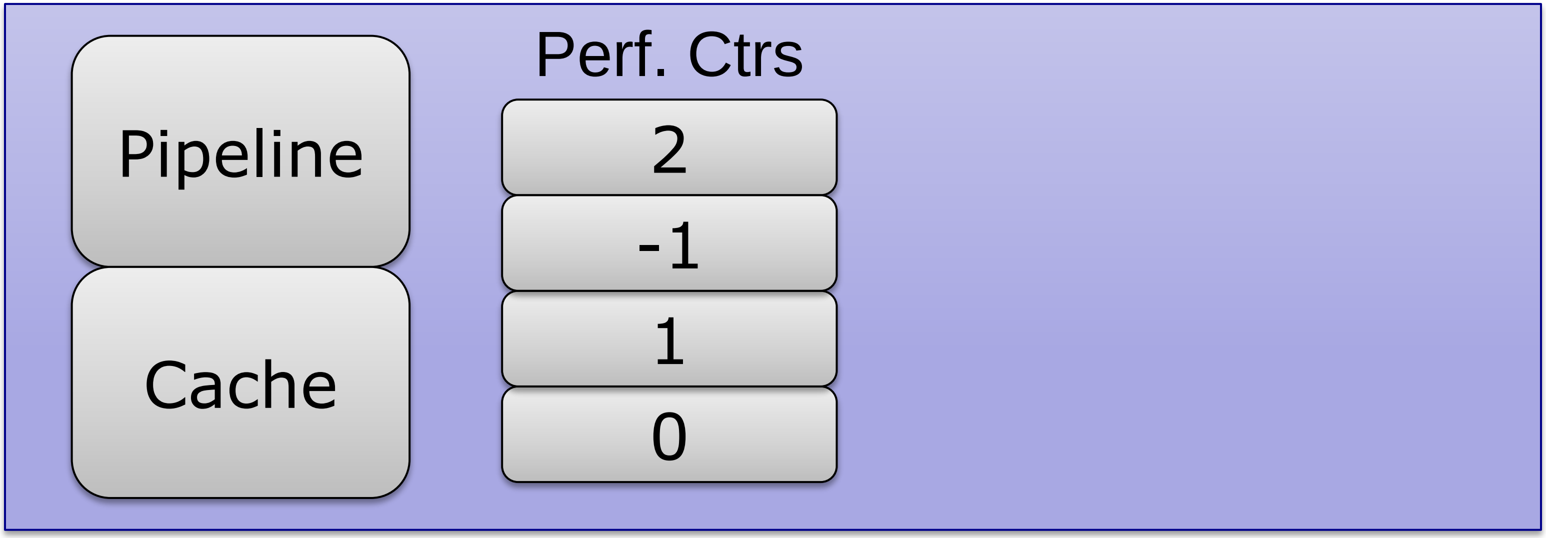
Hardware Sharing Detector

- Hardware Performance Counters



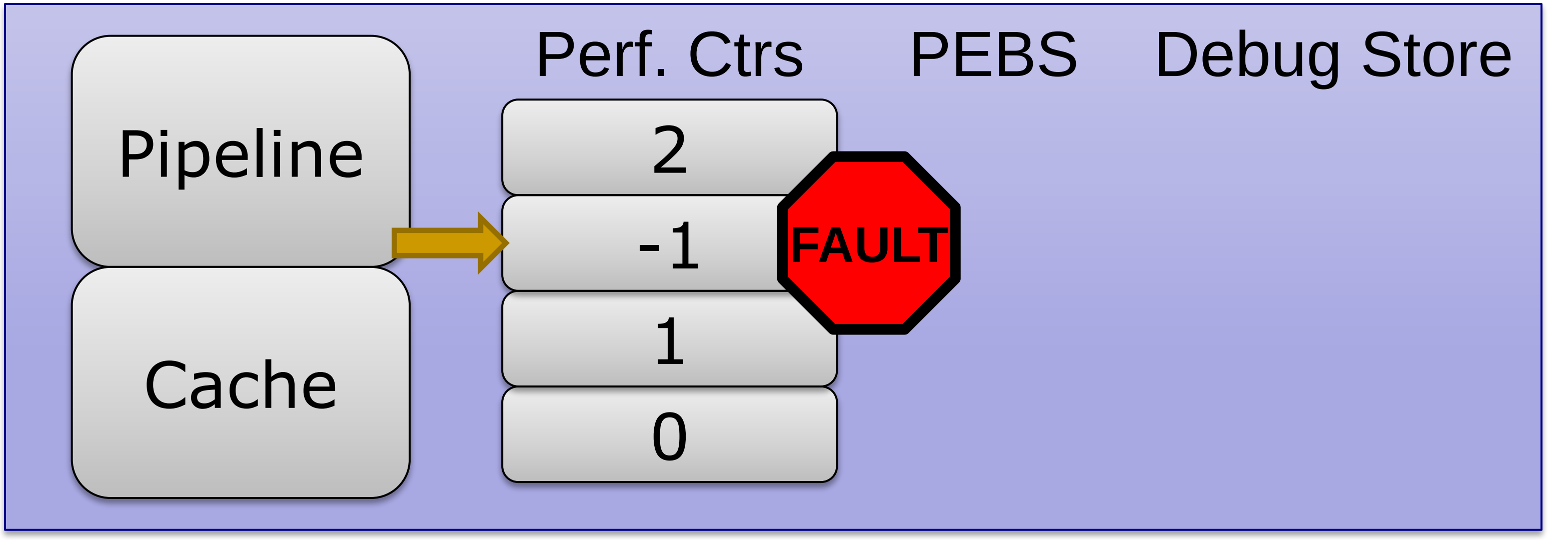
Hardware Sharing Detector

- Hardware Performance Counters



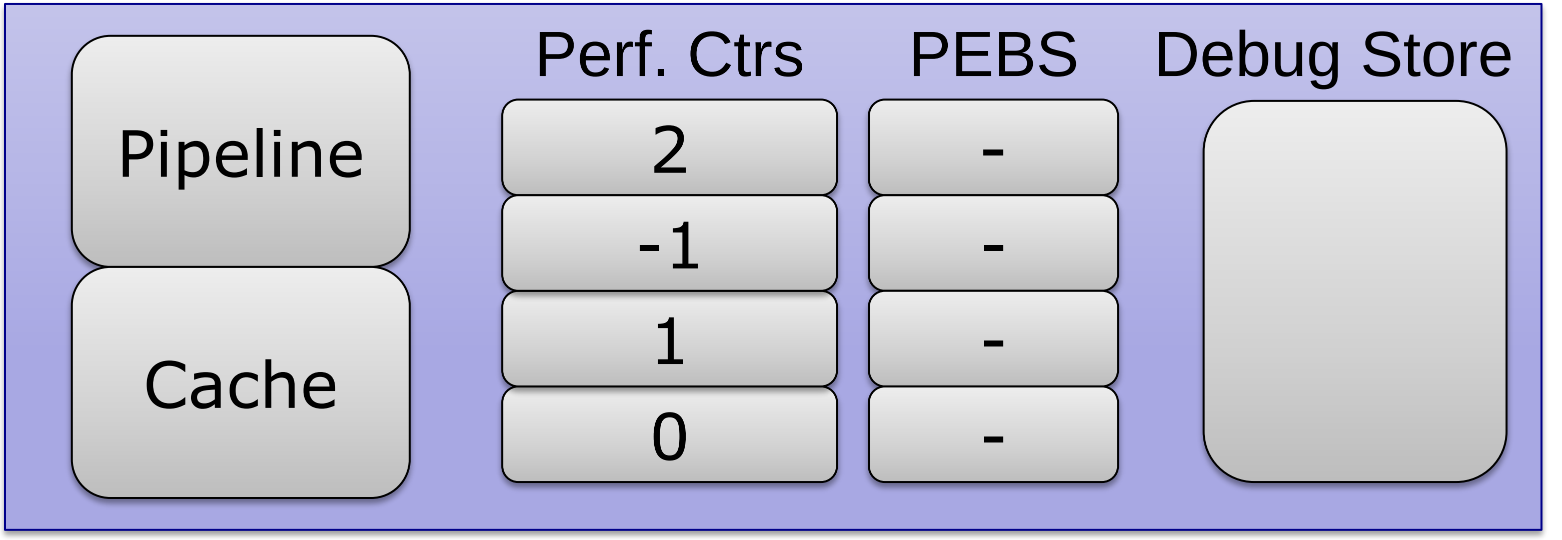
Hardware Sharing Detector

- Hardware Performance Counters



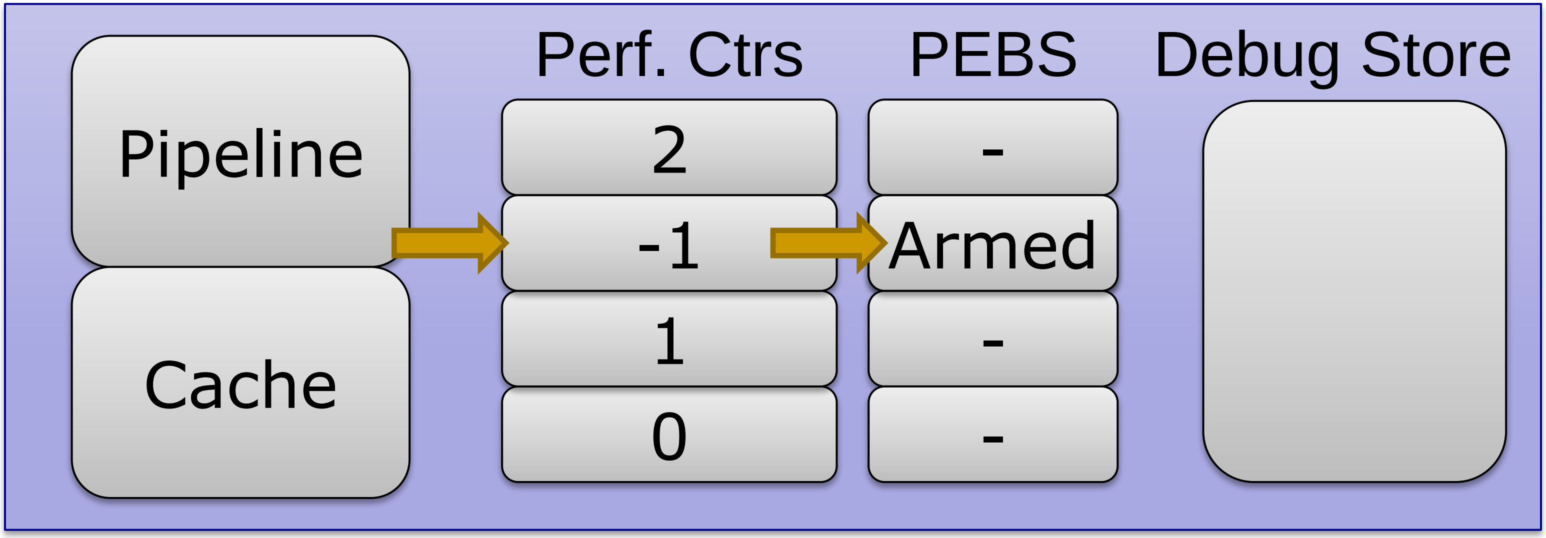
Hardware Sharing Detector

- Hardware Performance Counters



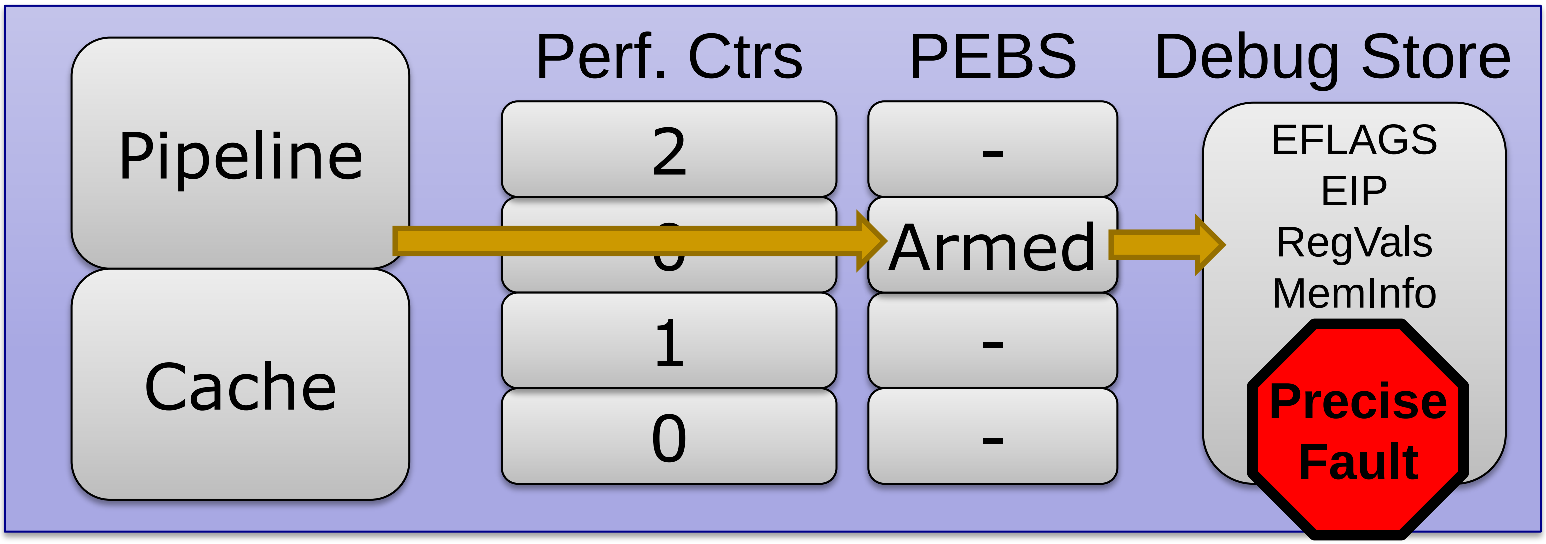
Hardware Sharing Detector

- Hardware Performance Counters



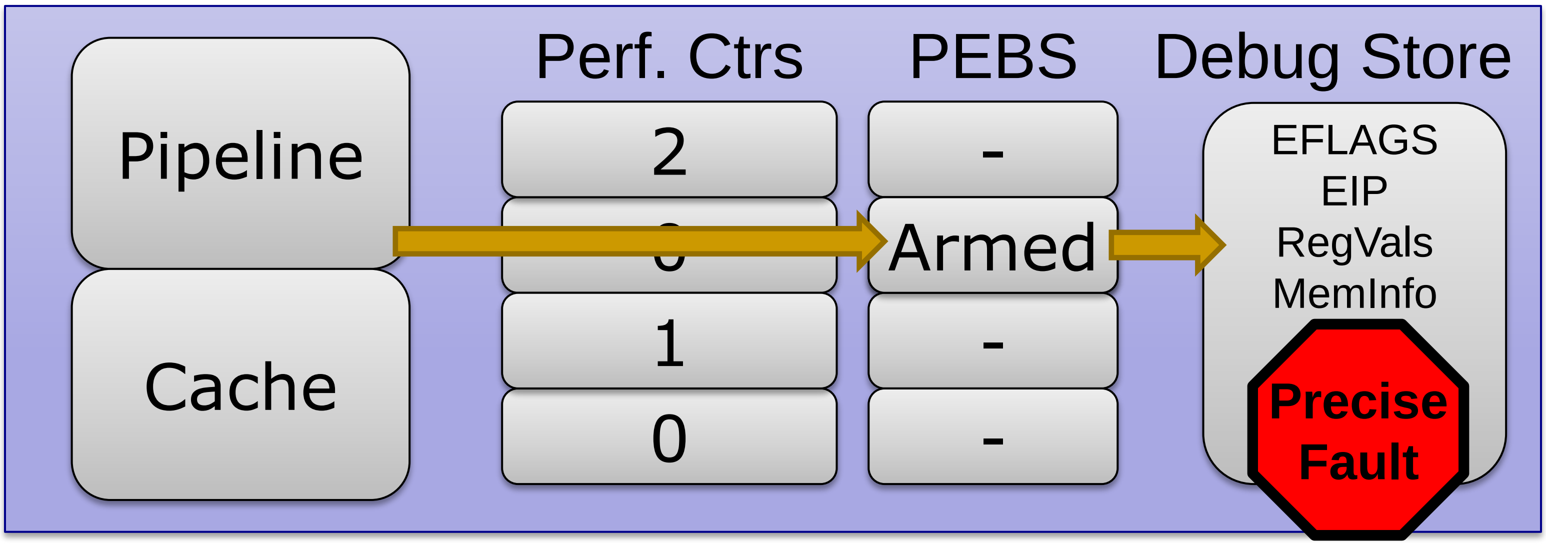
Hardware Sharing Detector

- Hardware Performance Counters

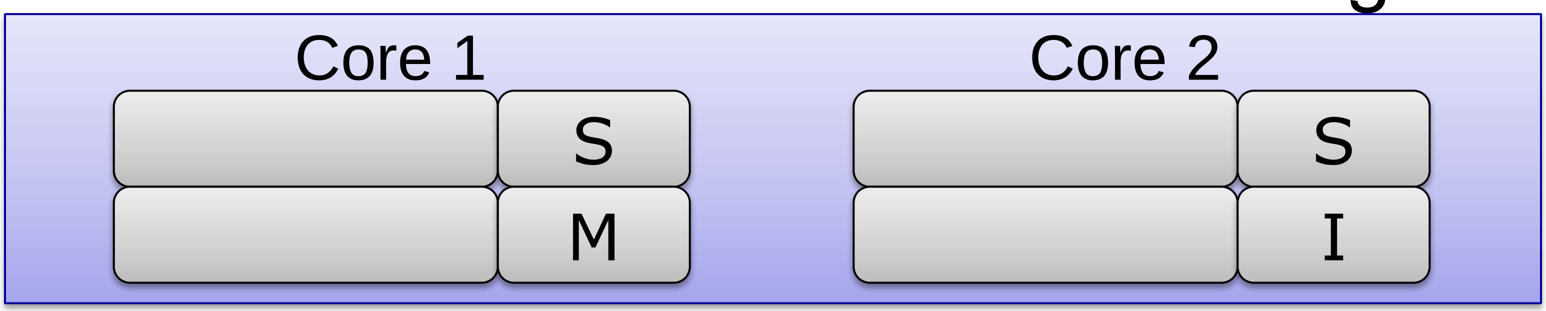


Hardware Sharing Detector

- Hardware Performance Counters

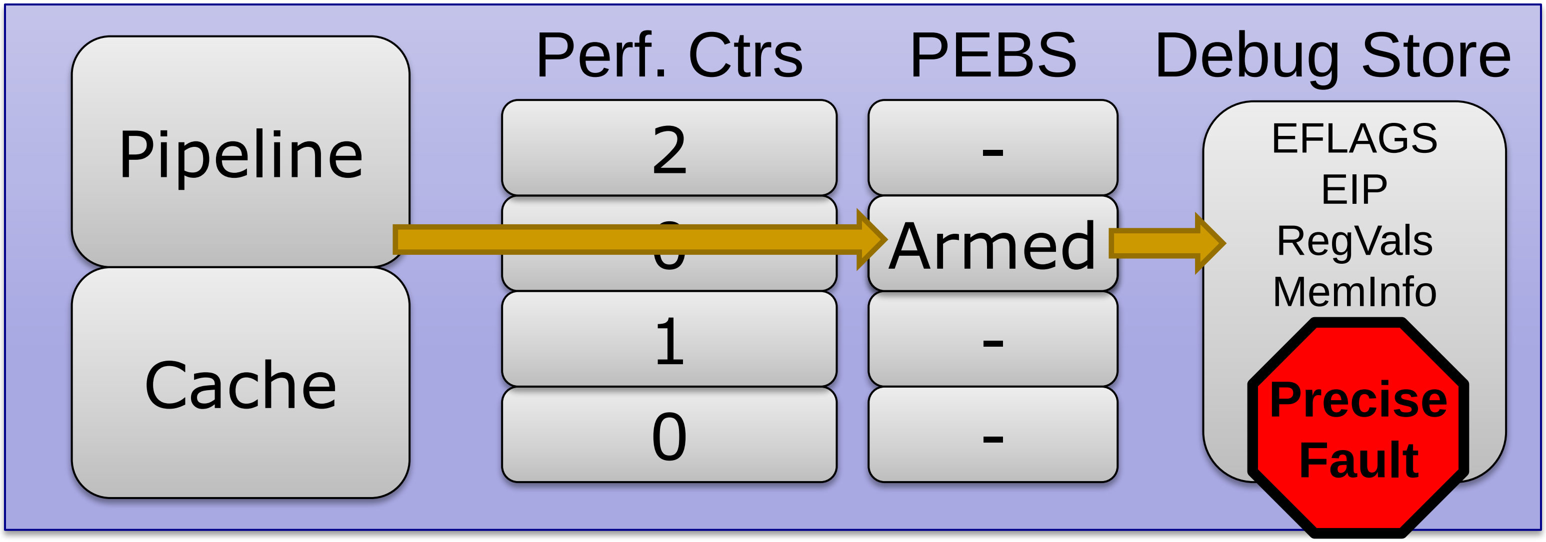


- Intel's HITM event: W→R Data Sharing

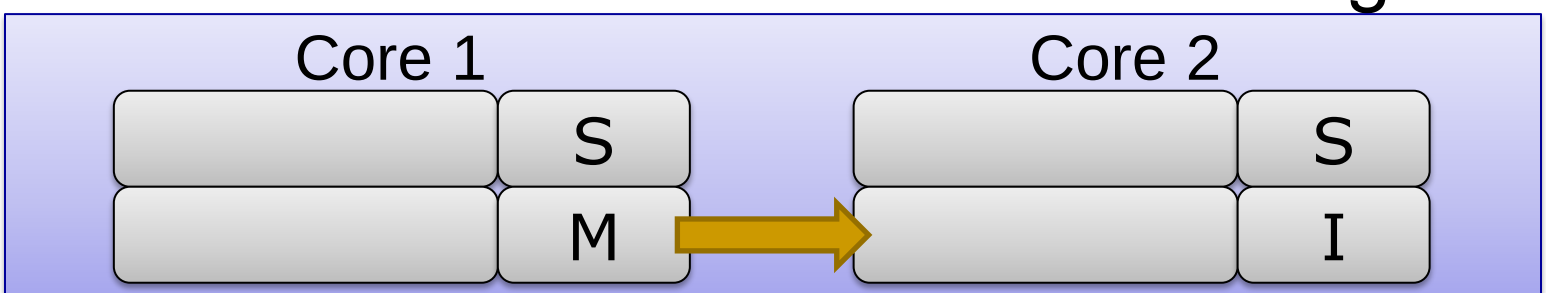


Hardware Sharing Detector

- Hardware Performance Counters

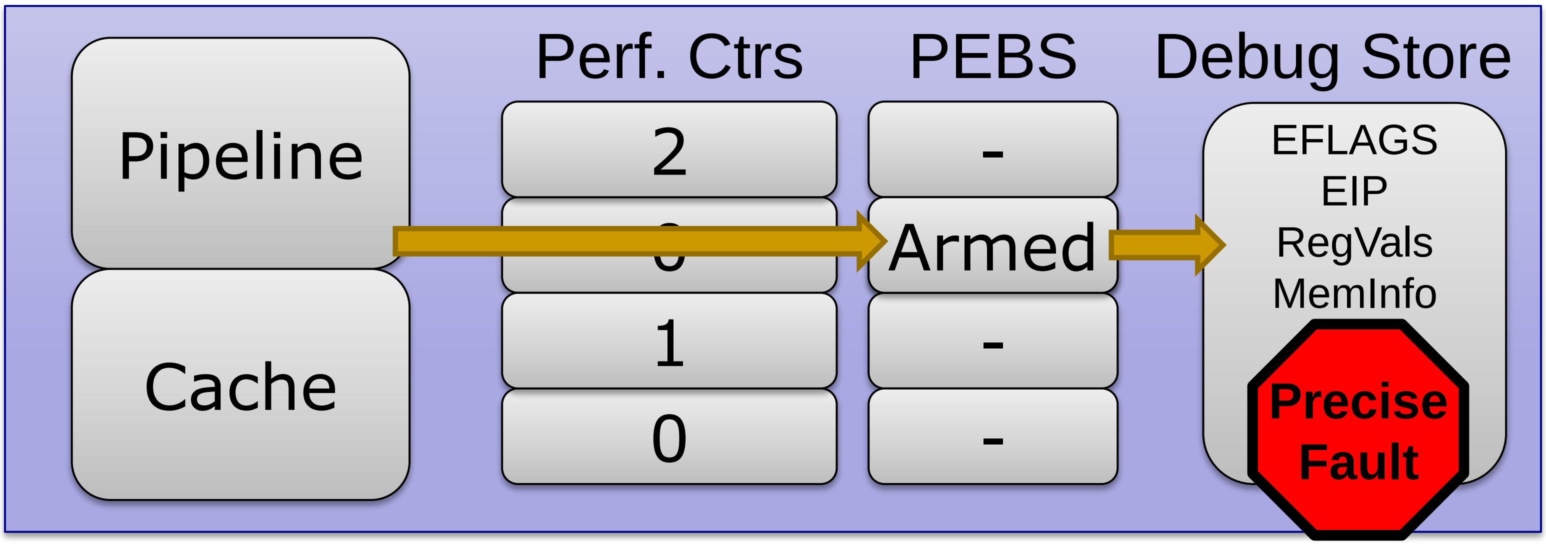


- Intel's HITM event: W→R Data Sharing

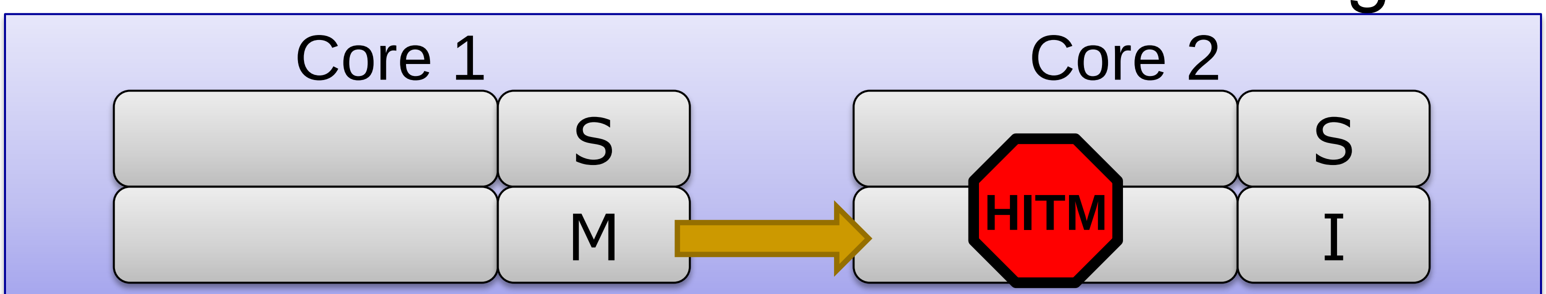


Hardware Sharing Detector

- Hardware Performance Counters



- Intel's HITM event: W→R Data Sharing



Potential Accuracy & Perf. Problems

- Limitations of Performance Counters
  - HITM only finds W→R Data Sharing
  - Hardware prefetcher events aren't counted

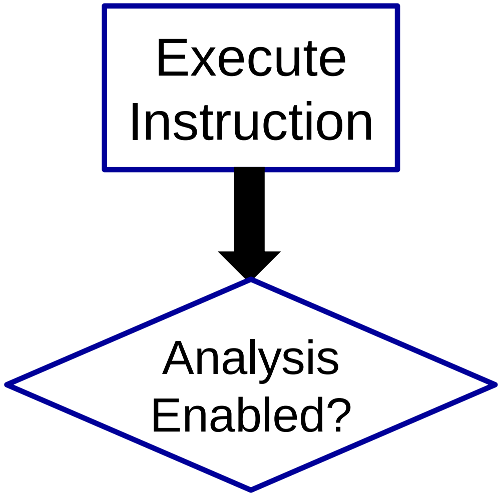
- Limitations of Cache Events
  - SMT sharing can't be counted
  - Cache eviction causes missed events
  - False sharing, etc…

- PEBS events still go through the kernel

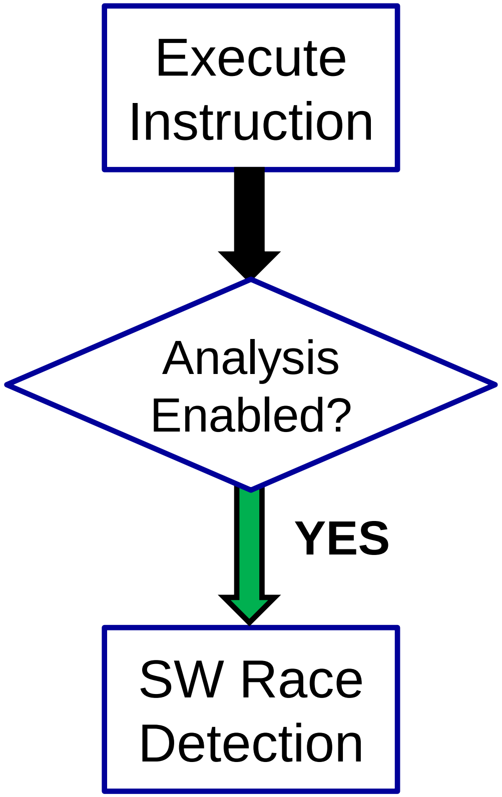
Demand-Driven Analysis on Real HW

Execute
Instruction

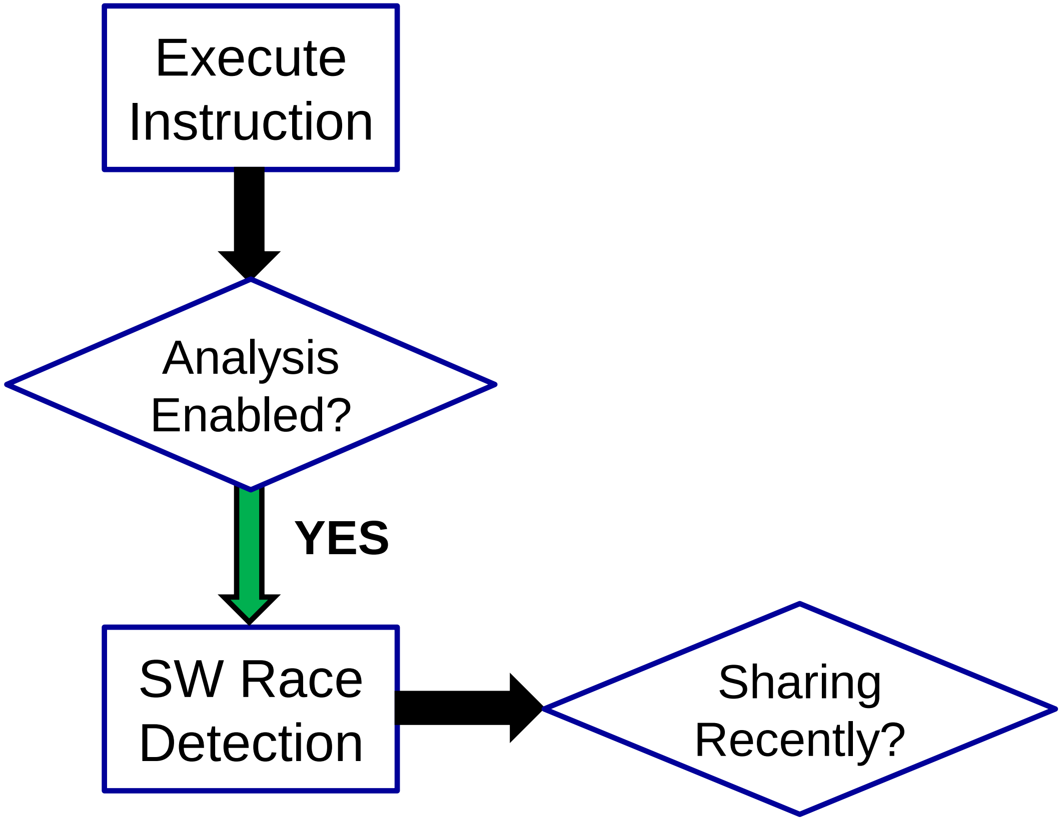
Demand-Driven Analysis on Real HW



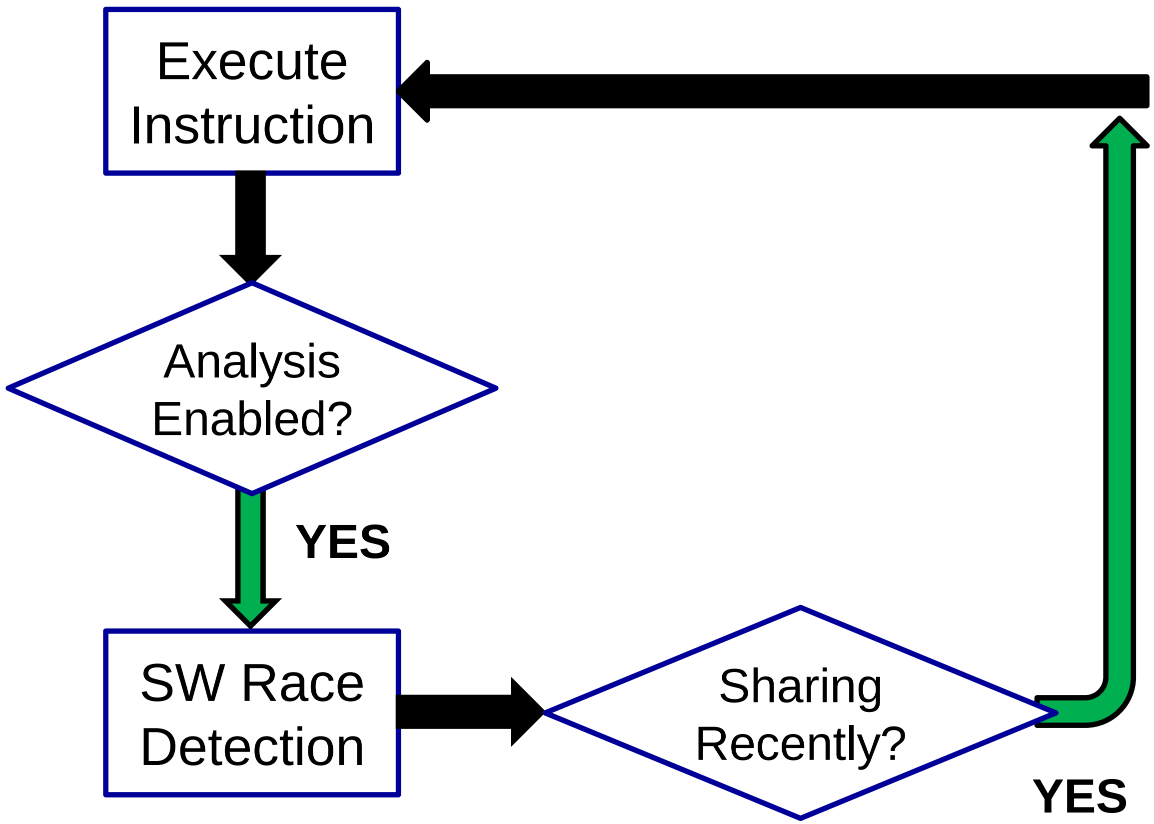
Demand-Driven Analysis on Real HW



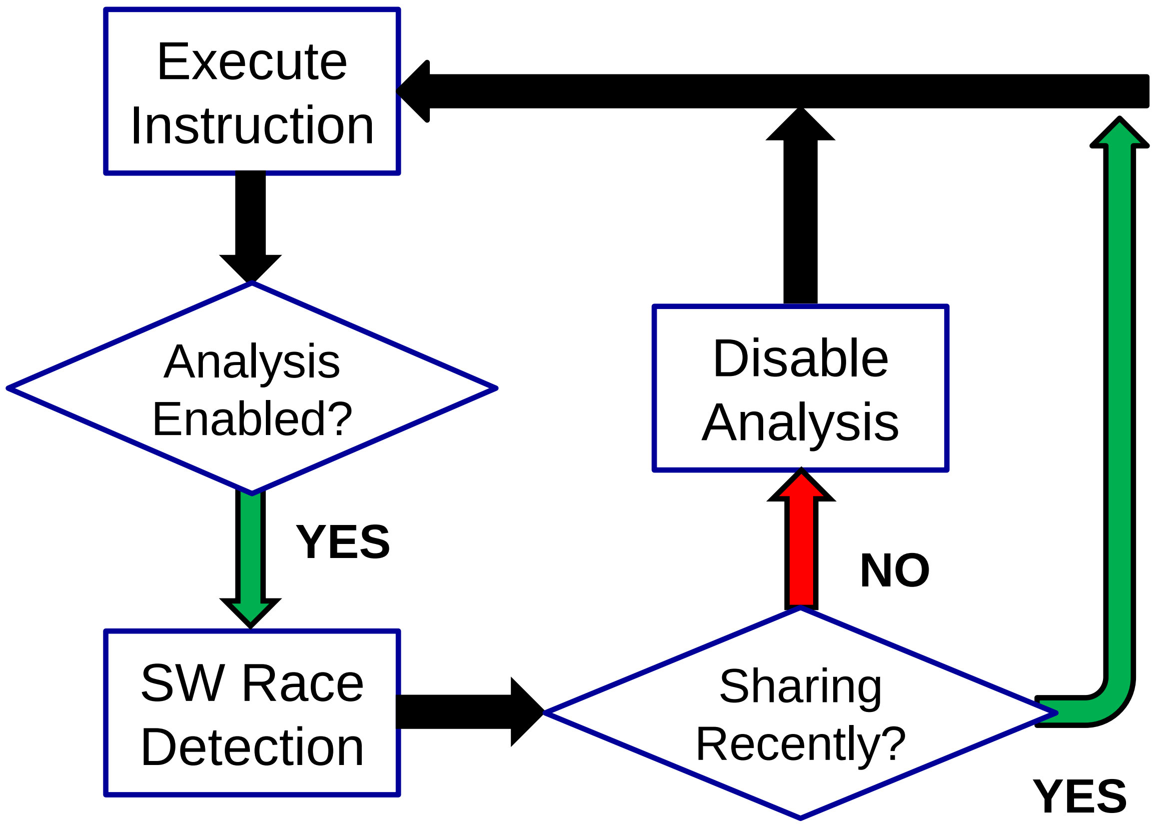
Demand-Driven Analysis on Real HW



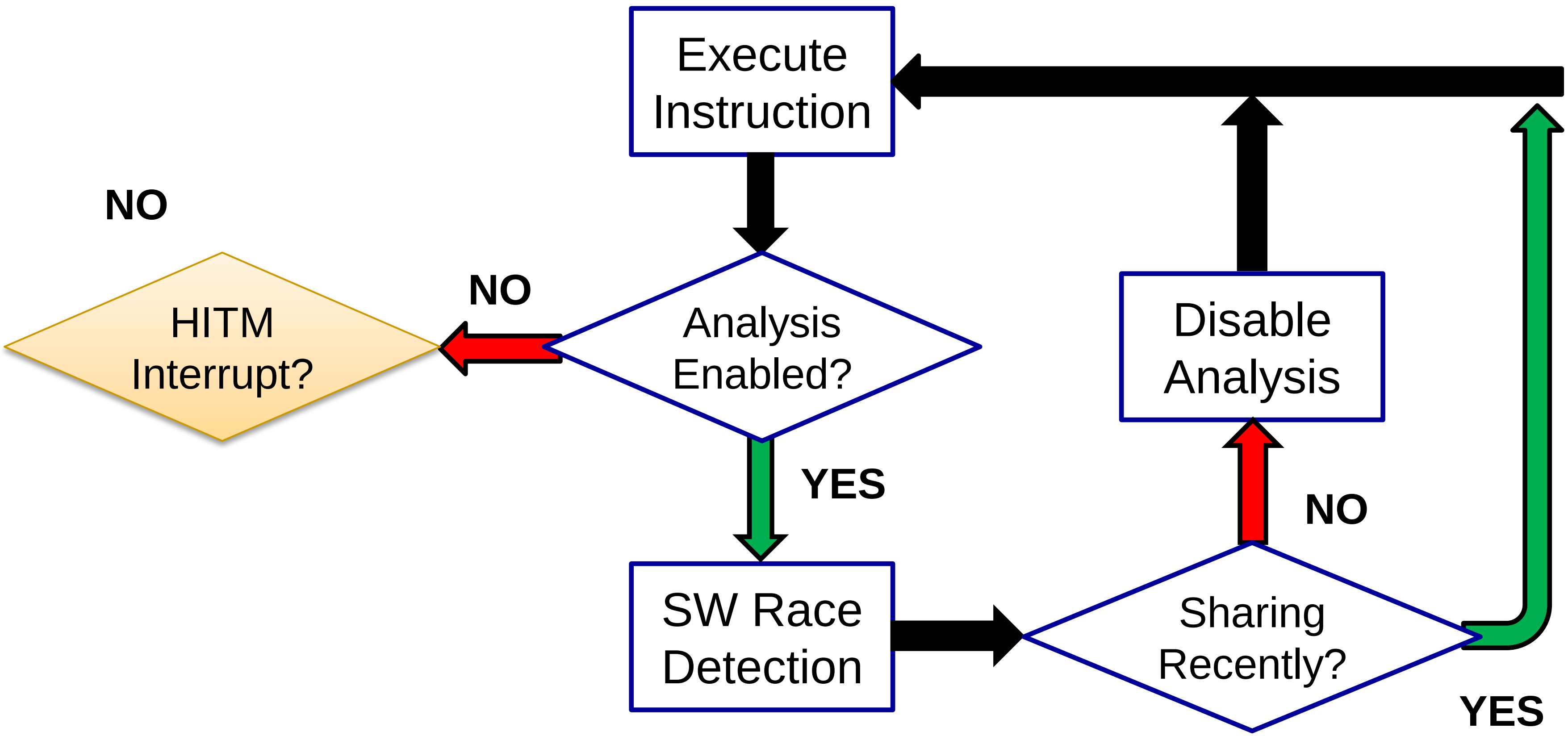
Demand-Driven Analysis on Real HW



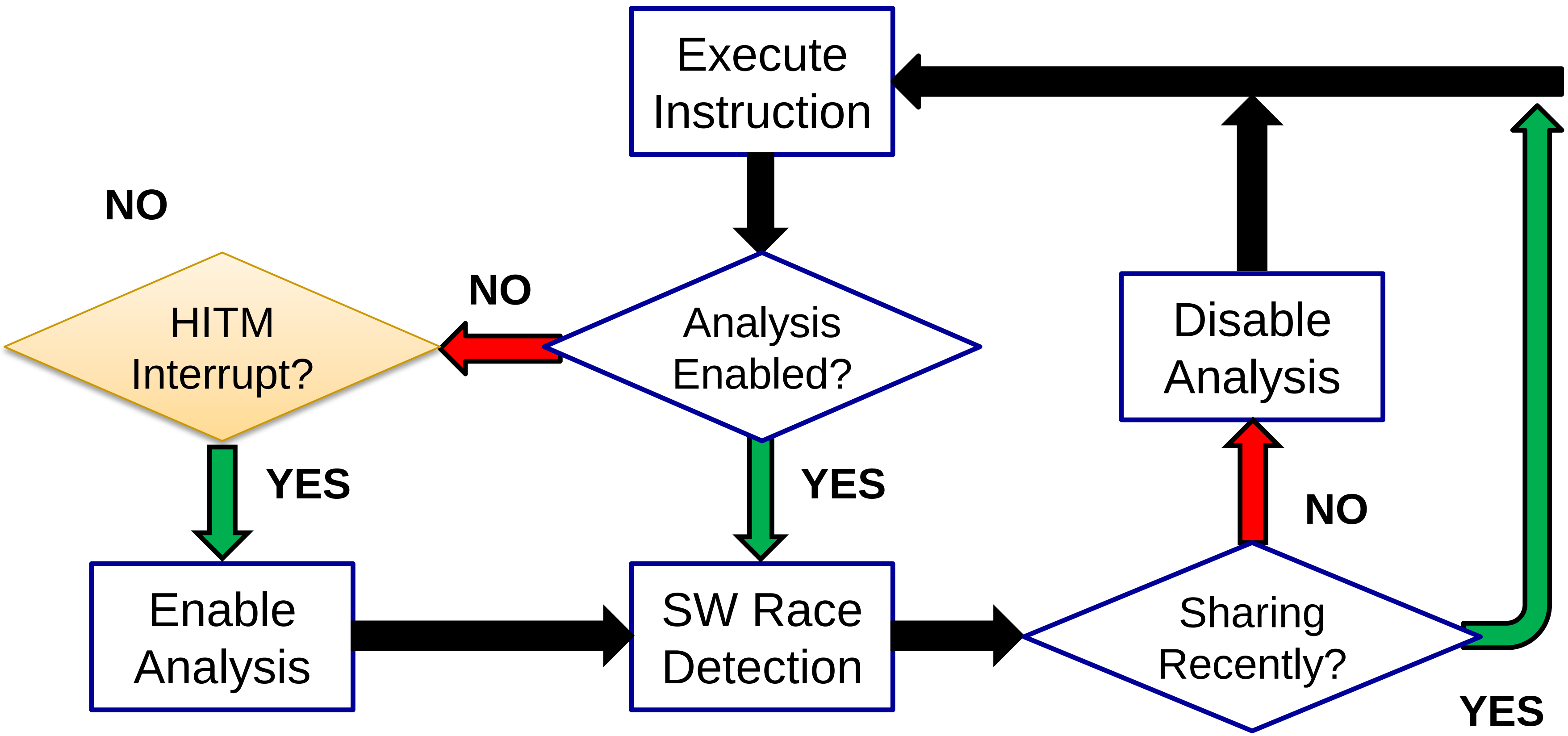
Demand-Driven Analysis on Real HW



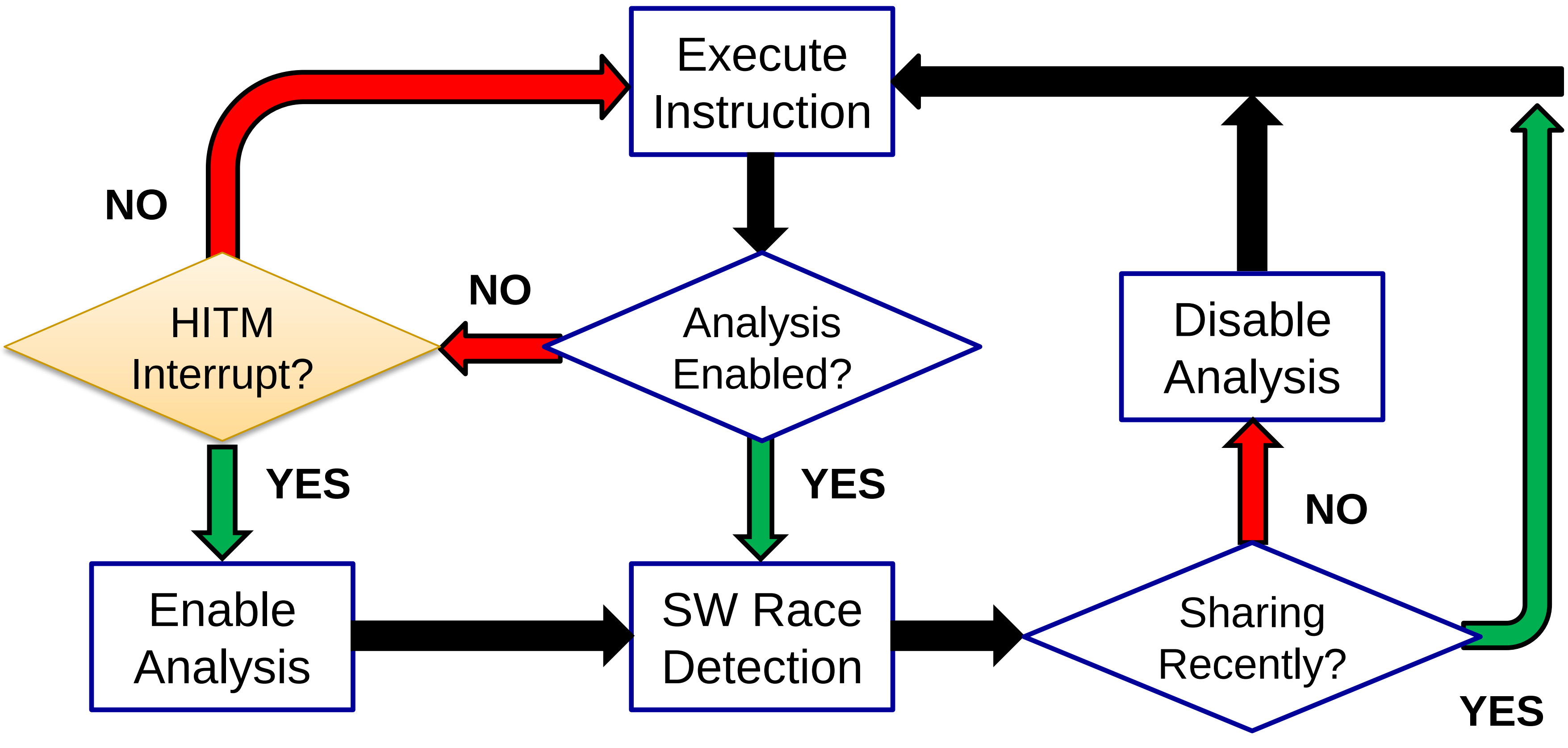
Demand-Driven Analysis on Real HW



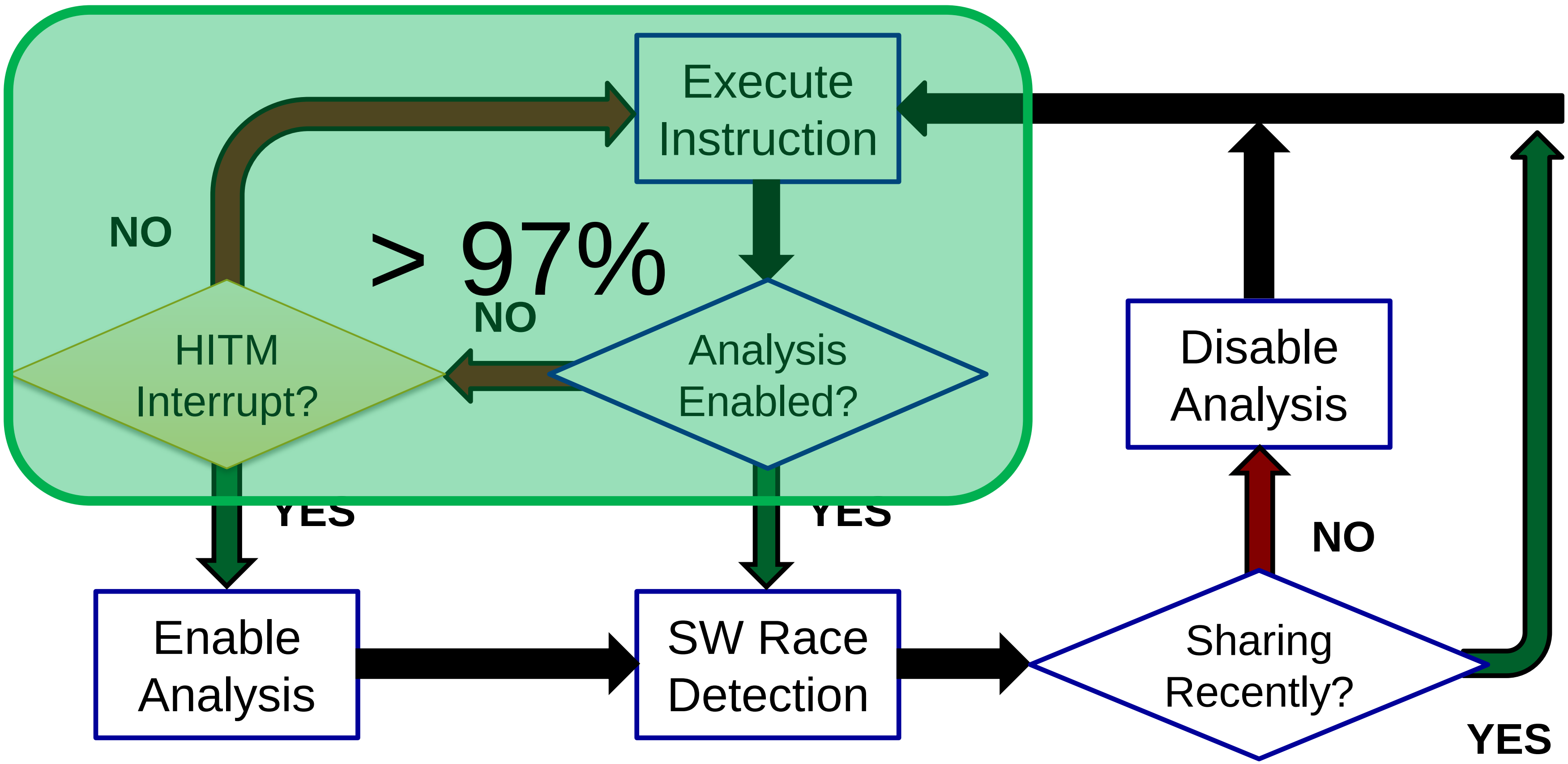
Demand-Driven Analysis on Real HW



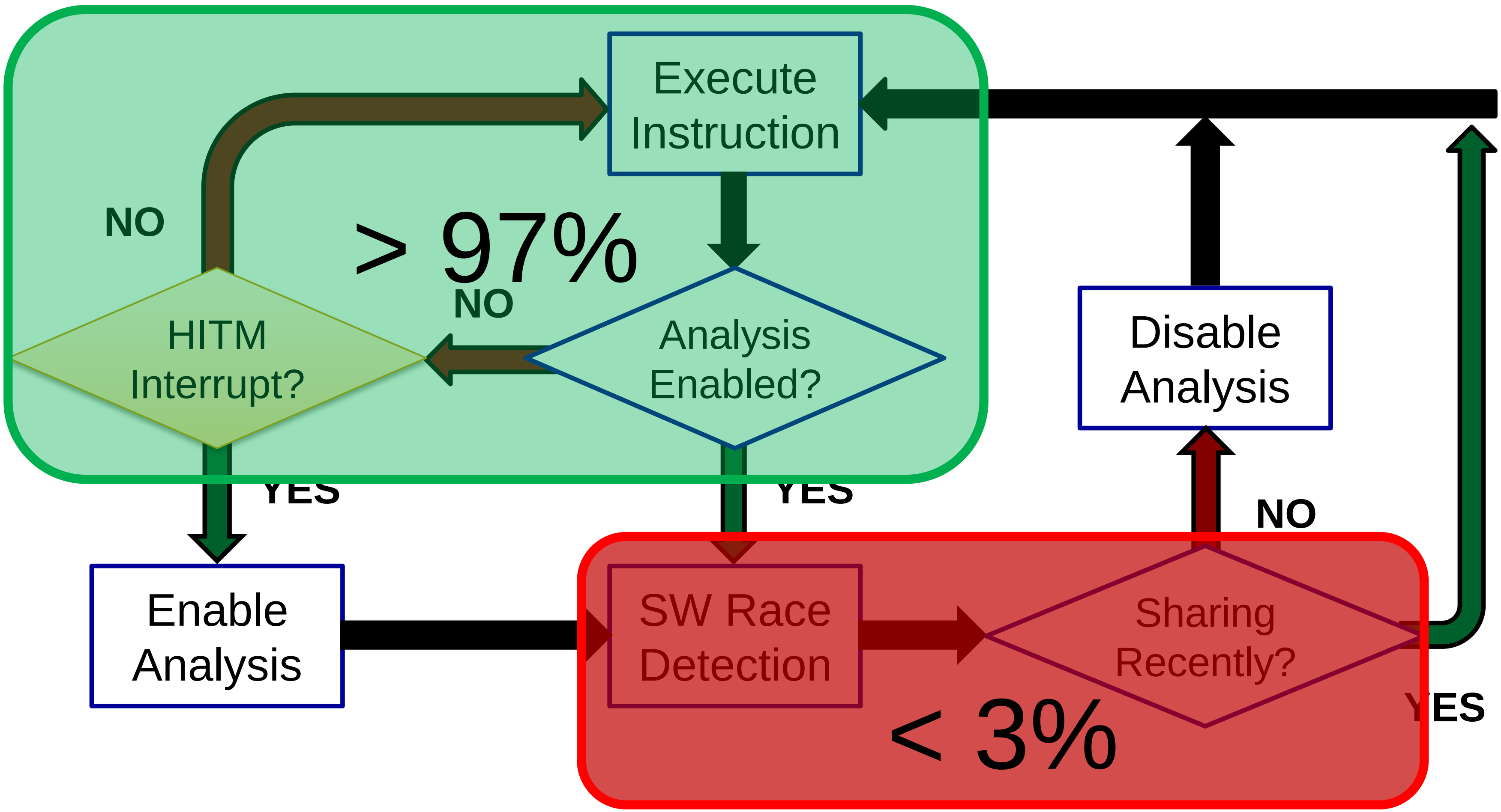
Demand-Driven Analysis on Real HW



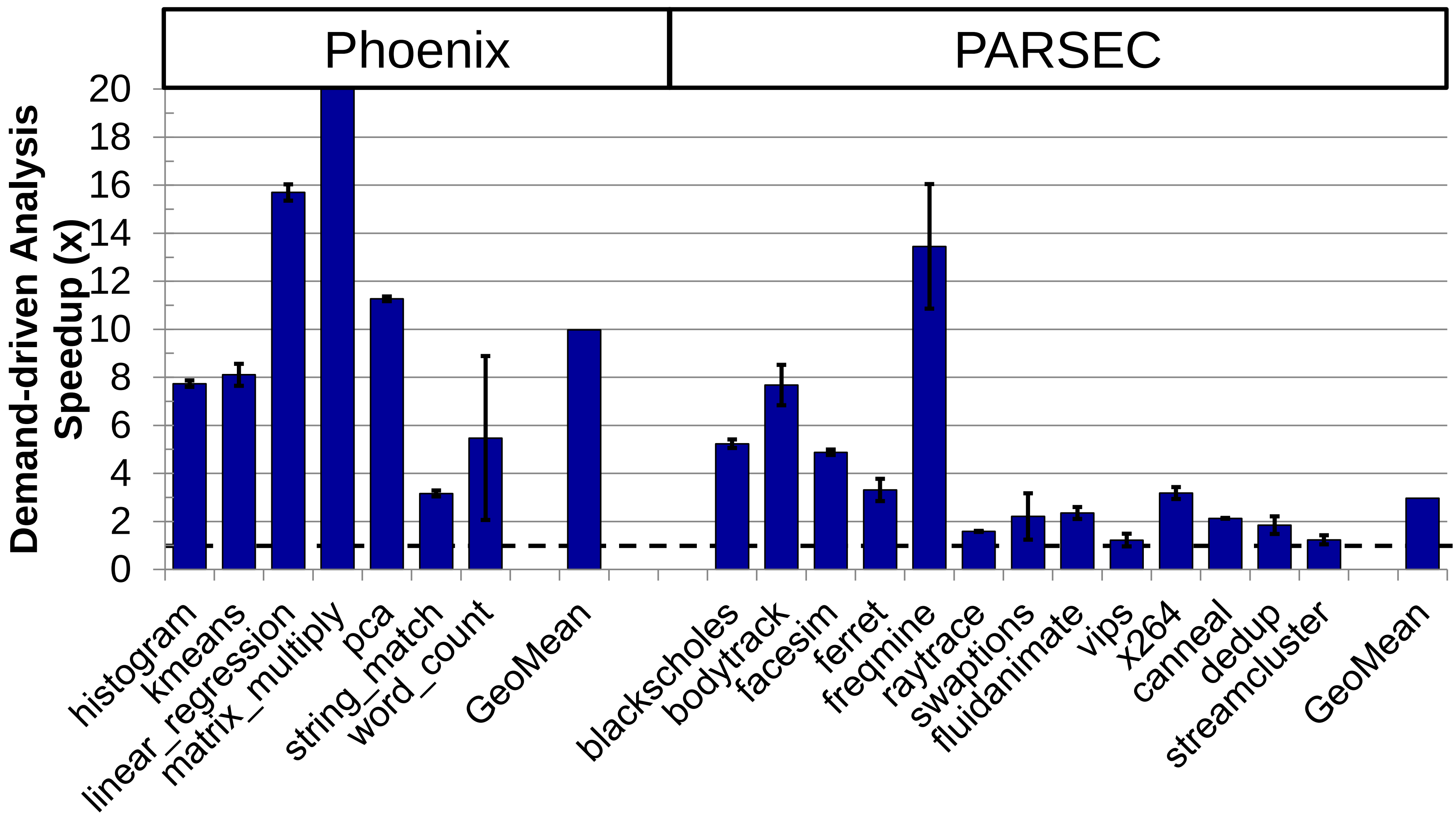
On-Demand Analysis on Real HW



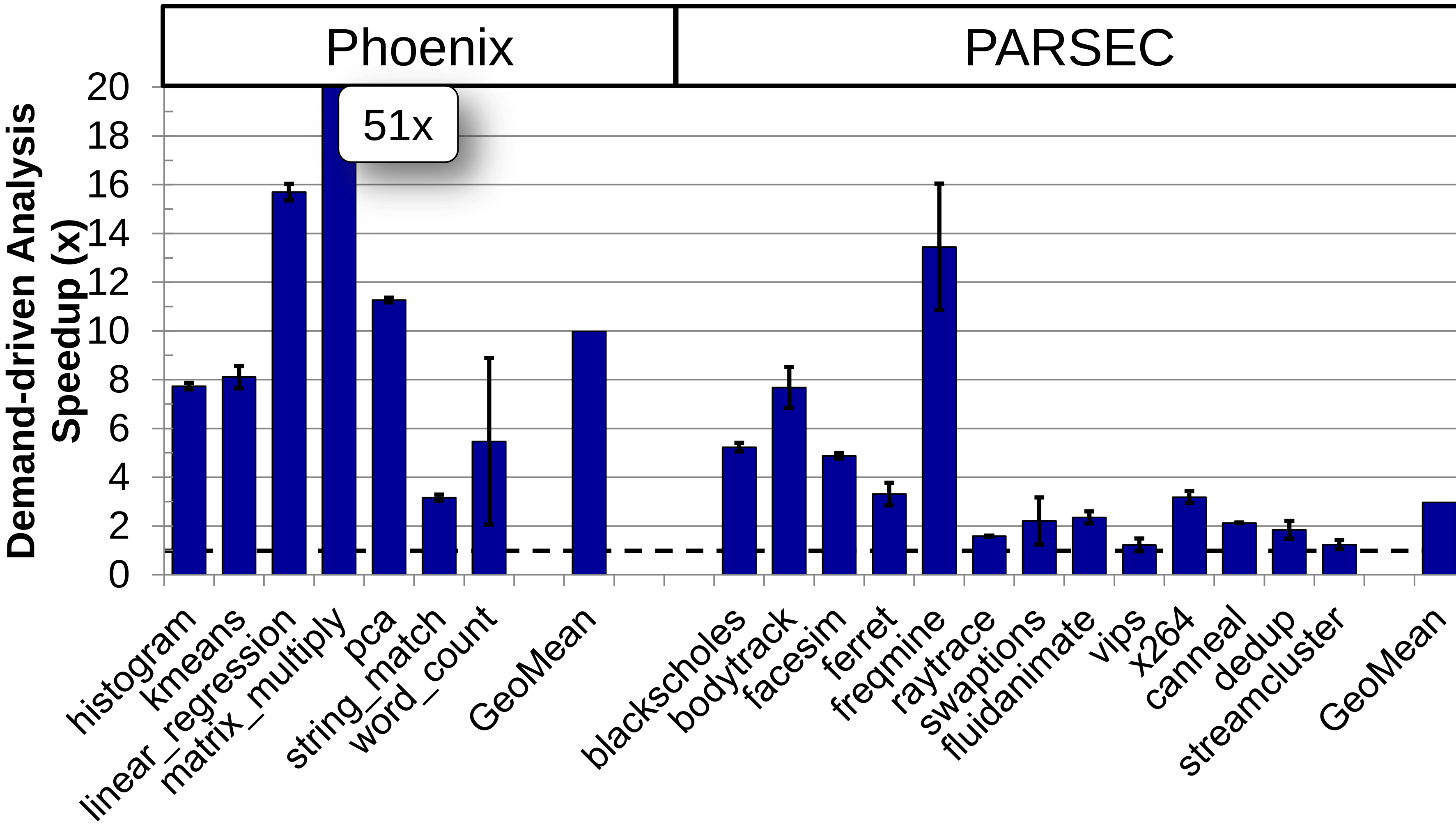
On-Demand Analysis on Real HW



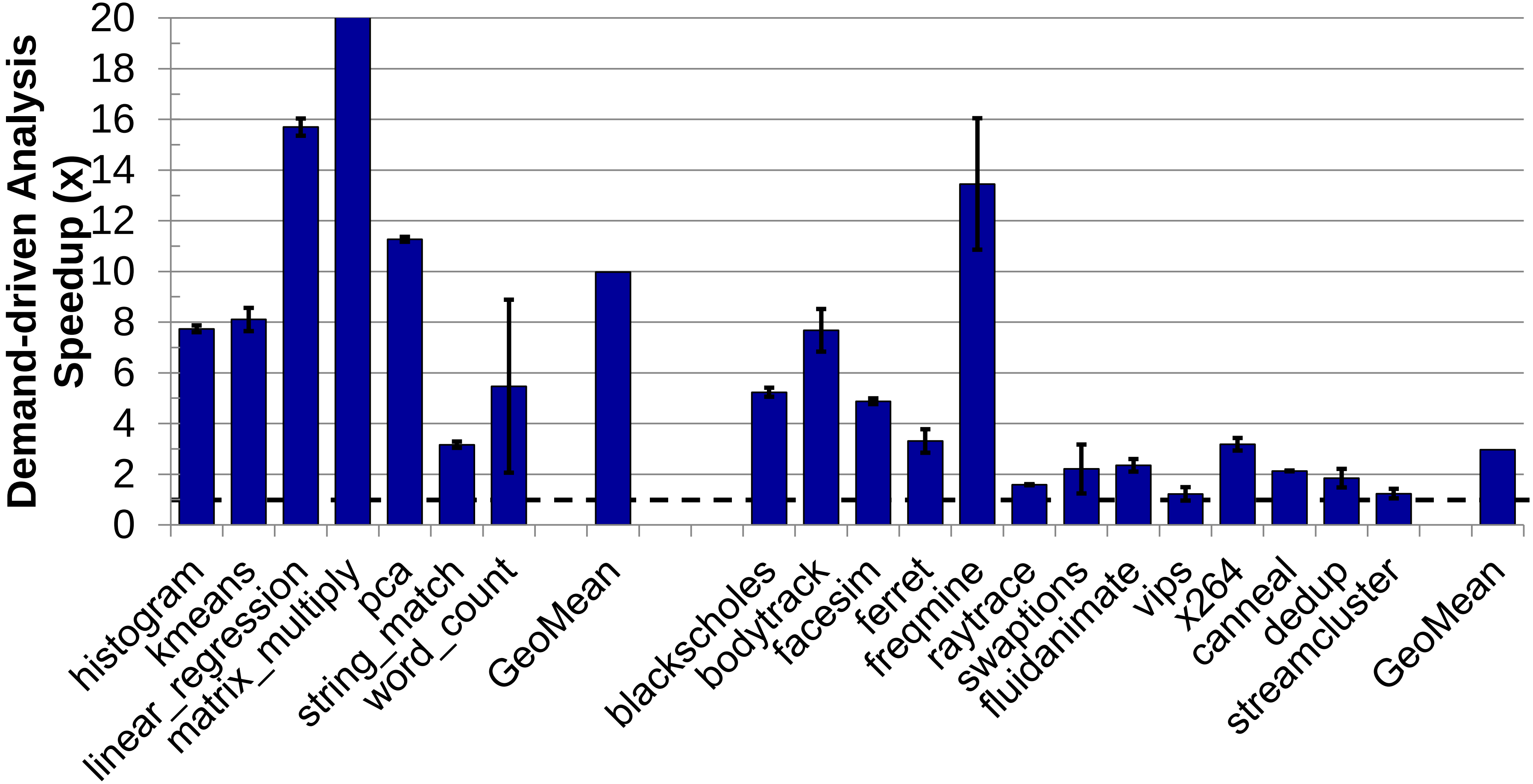
Performance Increases



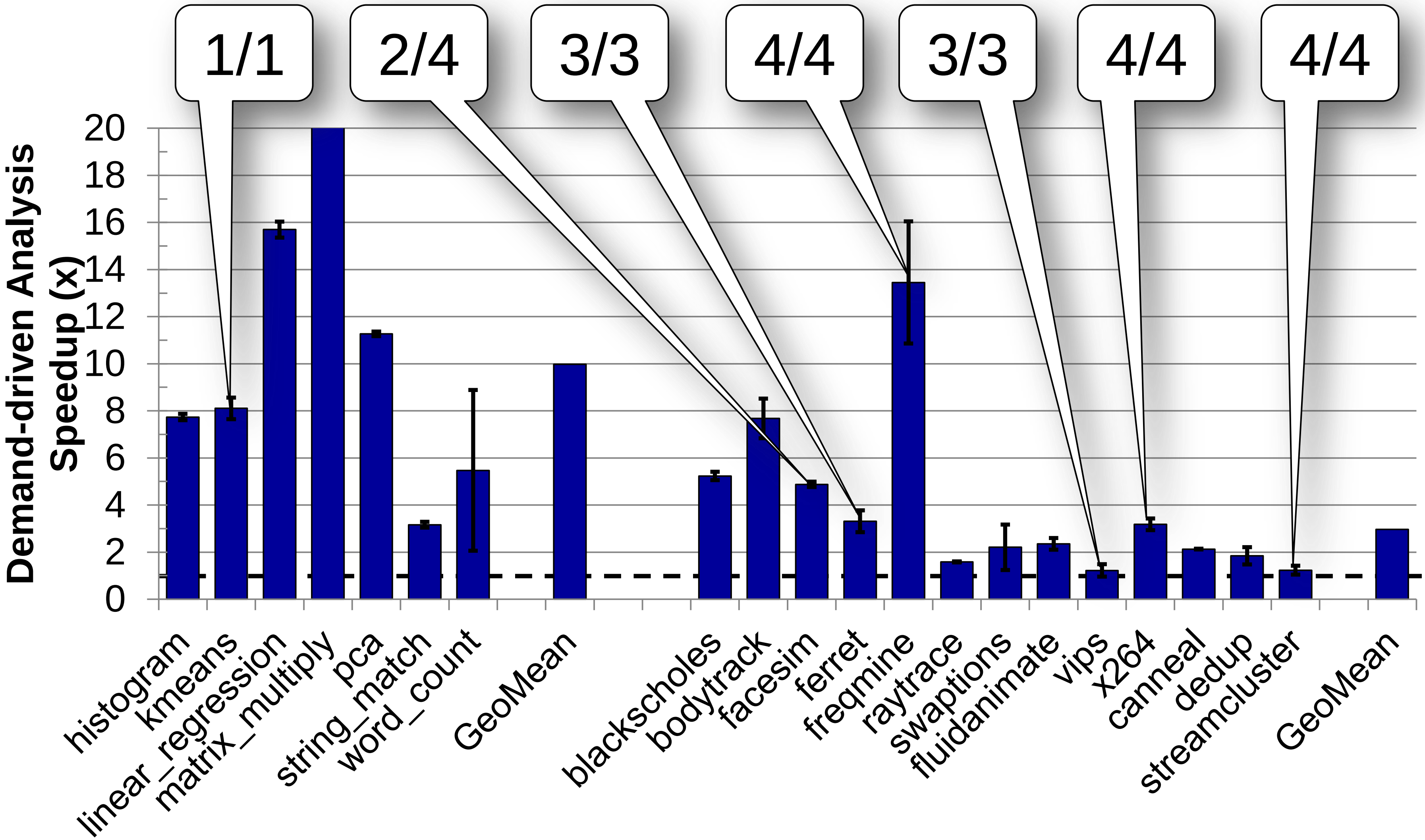
Performance Increases



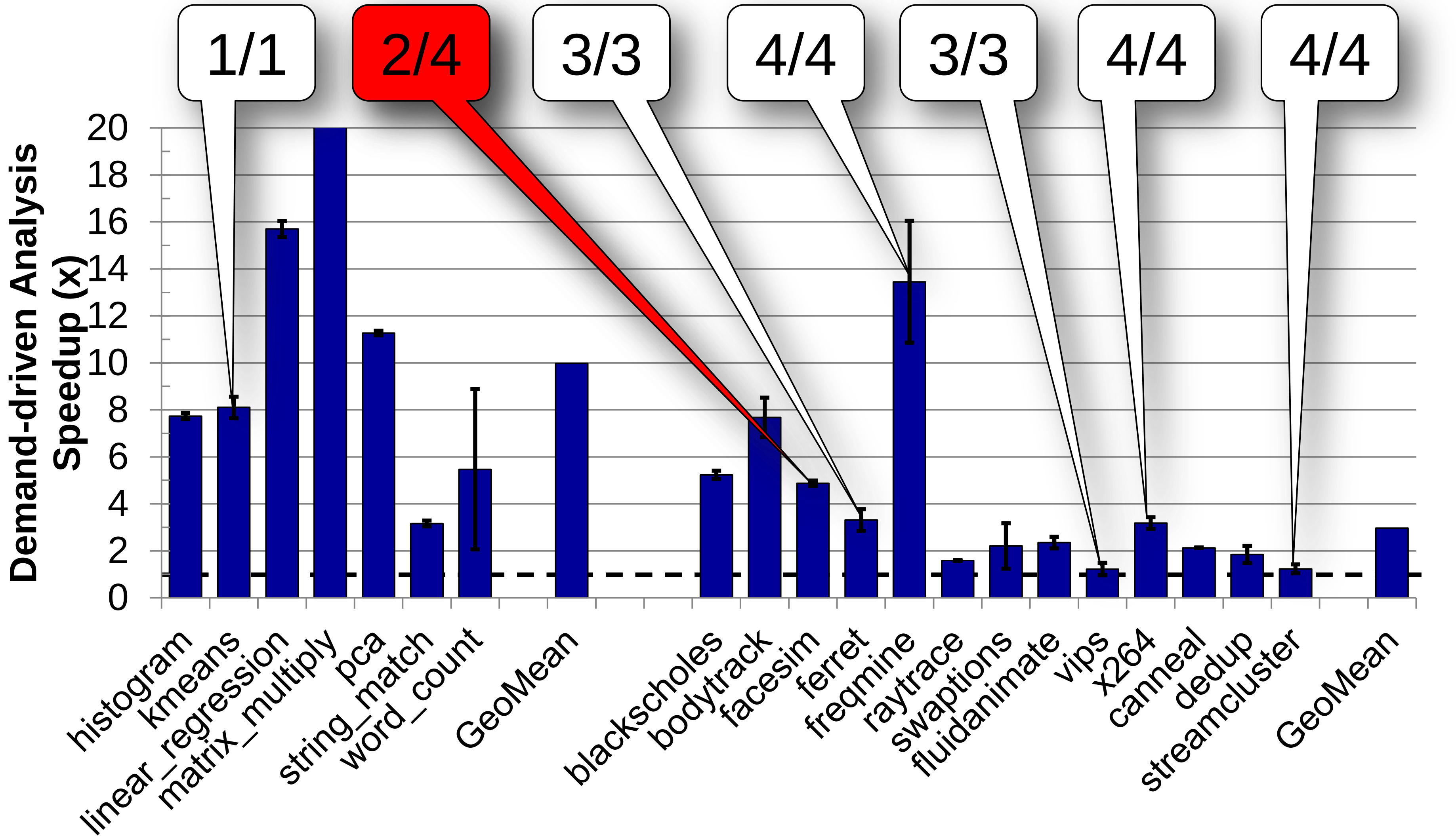
Demand-Driven Analysis Accuracy



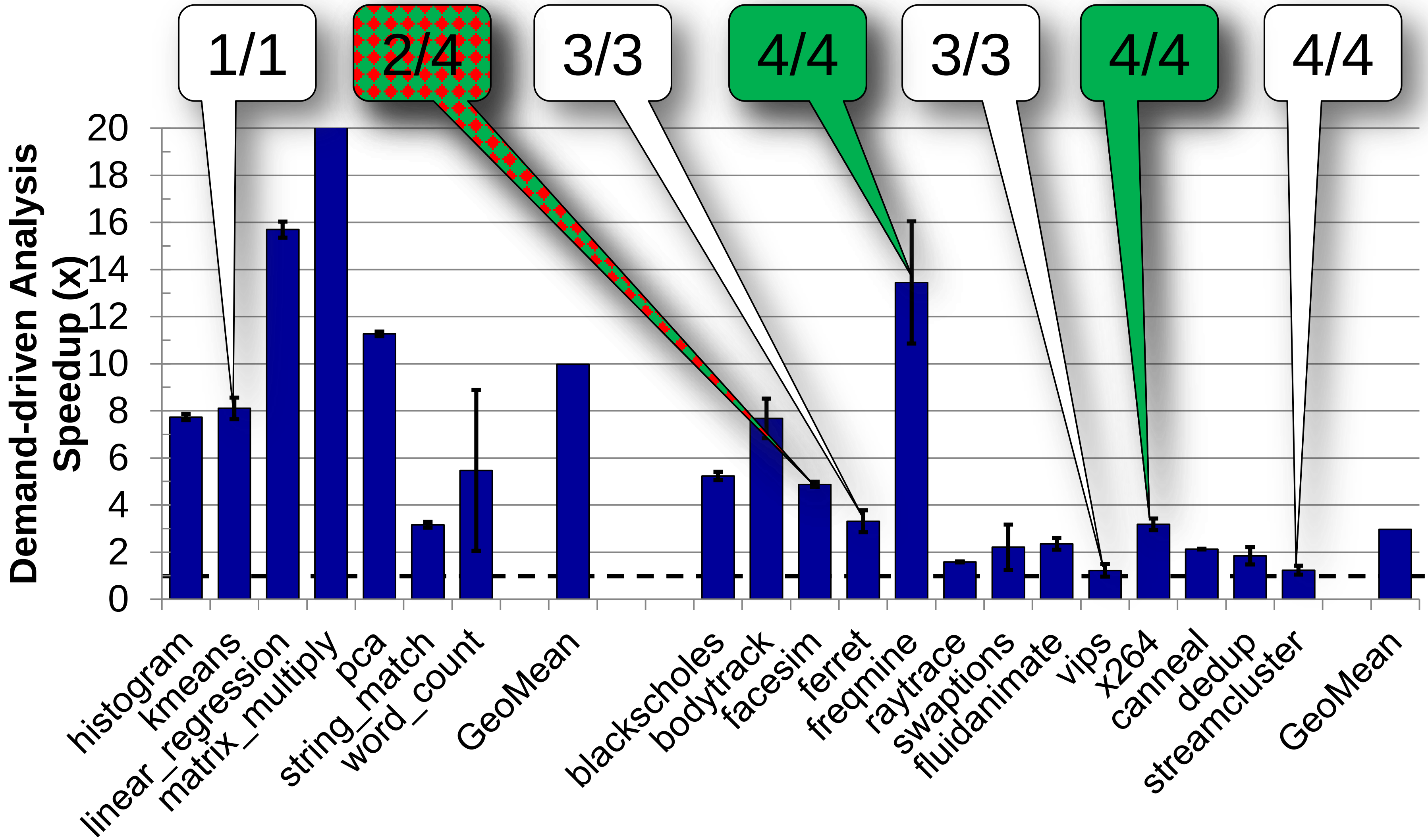
Demand-Driven Analysis Accuracy



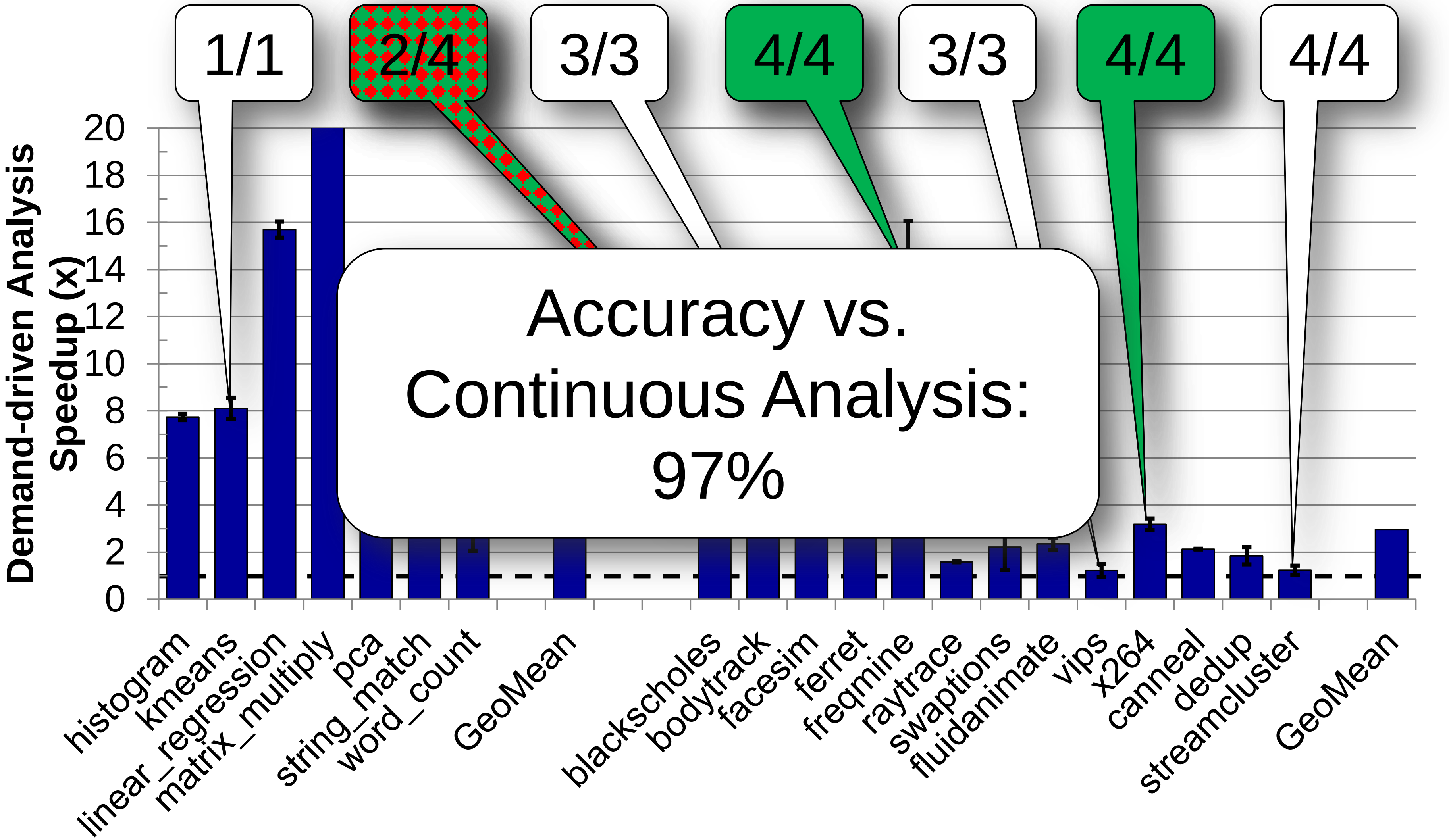
Demand-Driven Analysis Accuracy



Demand-Driven Analysis Accuracy



Demand-Driven Analysis Accuracy

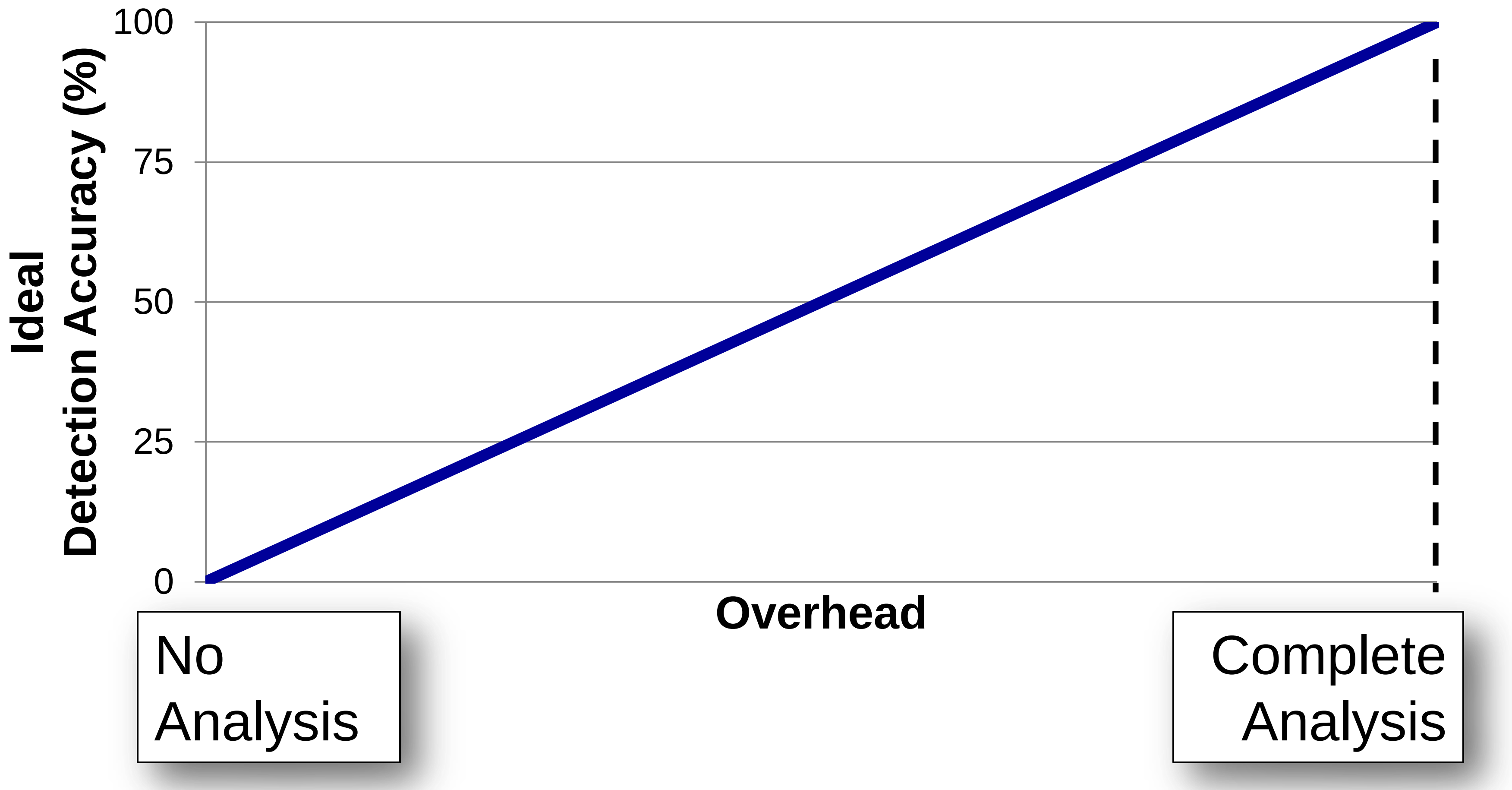


Outline

- Problem Statement
- Background Information
  - Demand-Driven Dynamic Dataflow Analysis
- Proposed Solutions
  - Demand-Driven Data Race Detection
  - Sampling to Cap Maximum Overheads

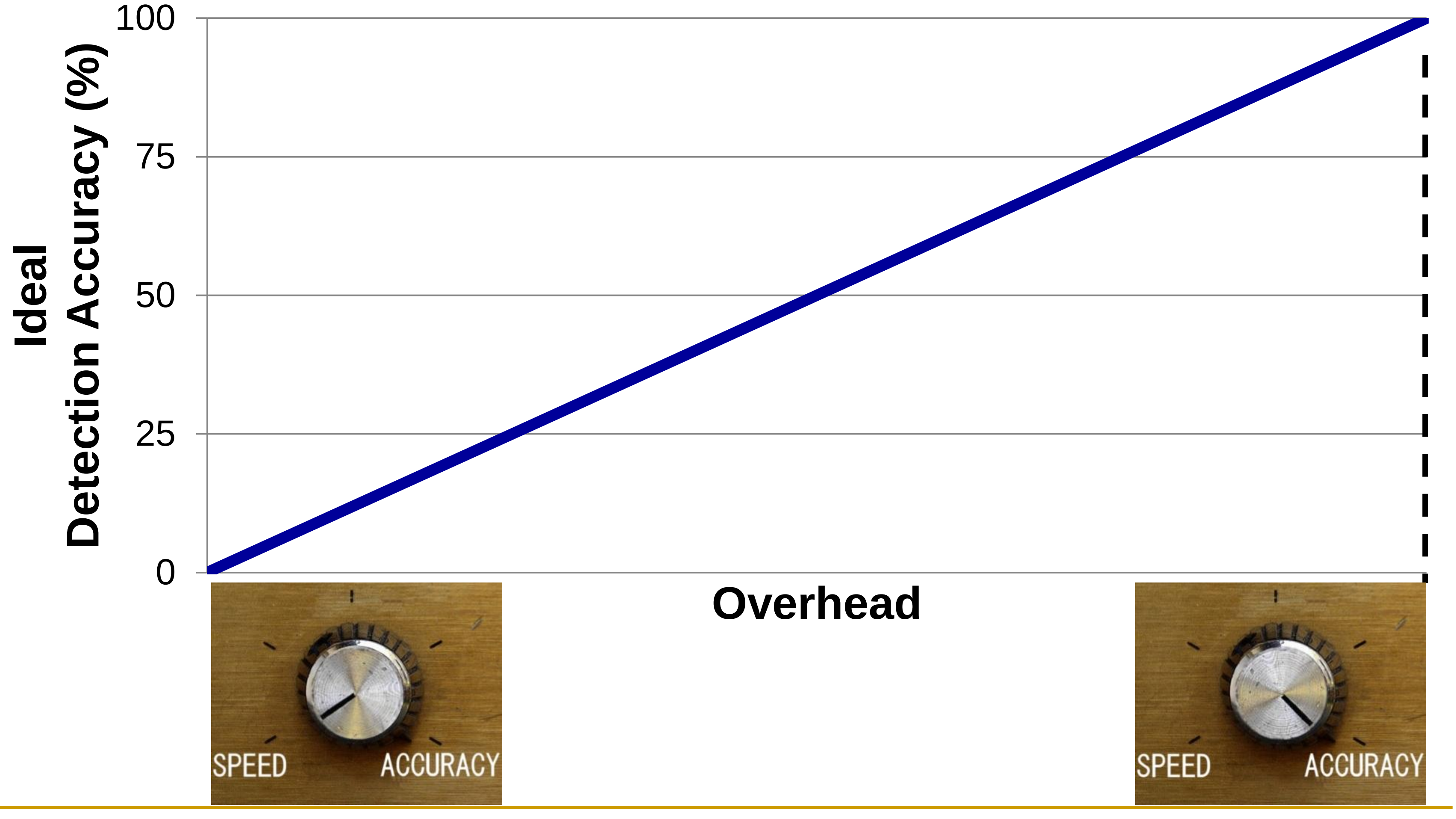
Reducing Overheads Further: Sampling

- Lower overheads by skipping some analyses

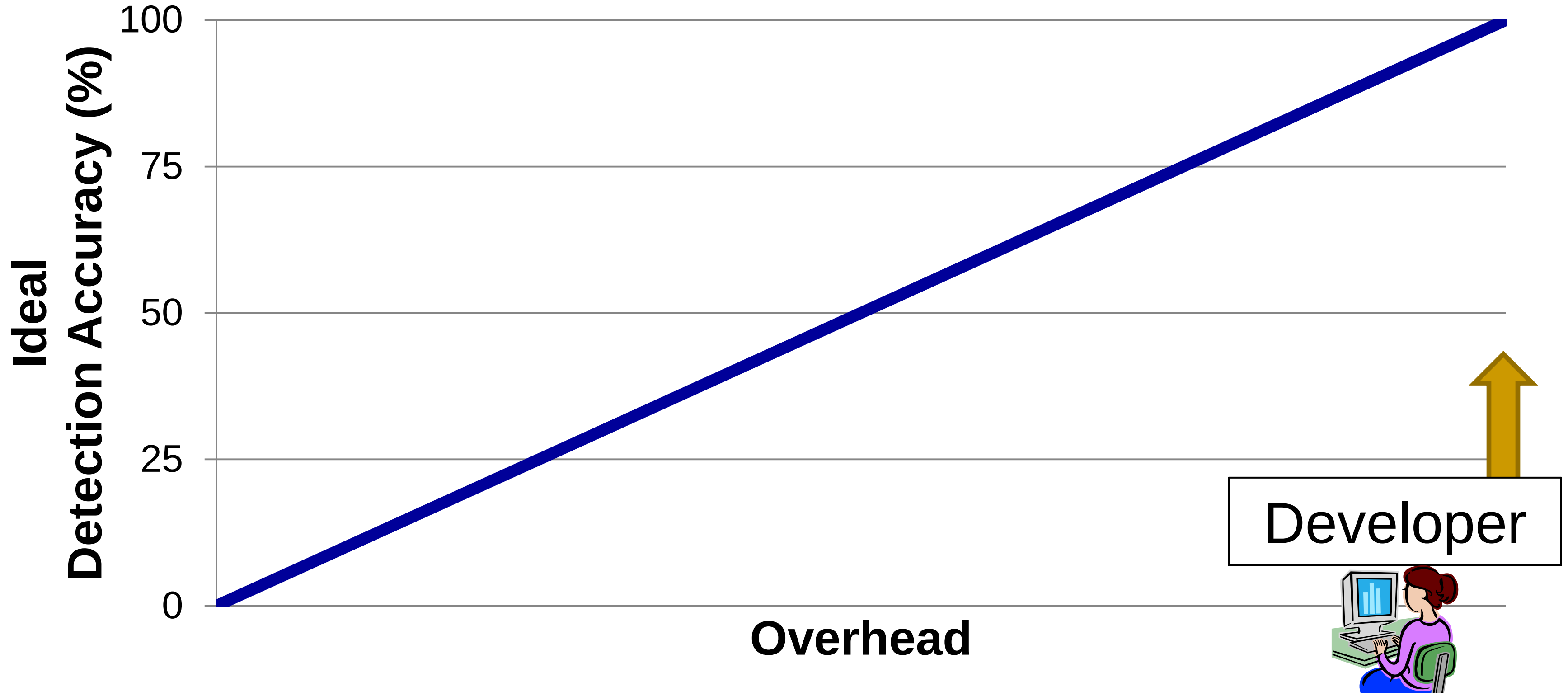


Reducing Overheads Further: Sampling

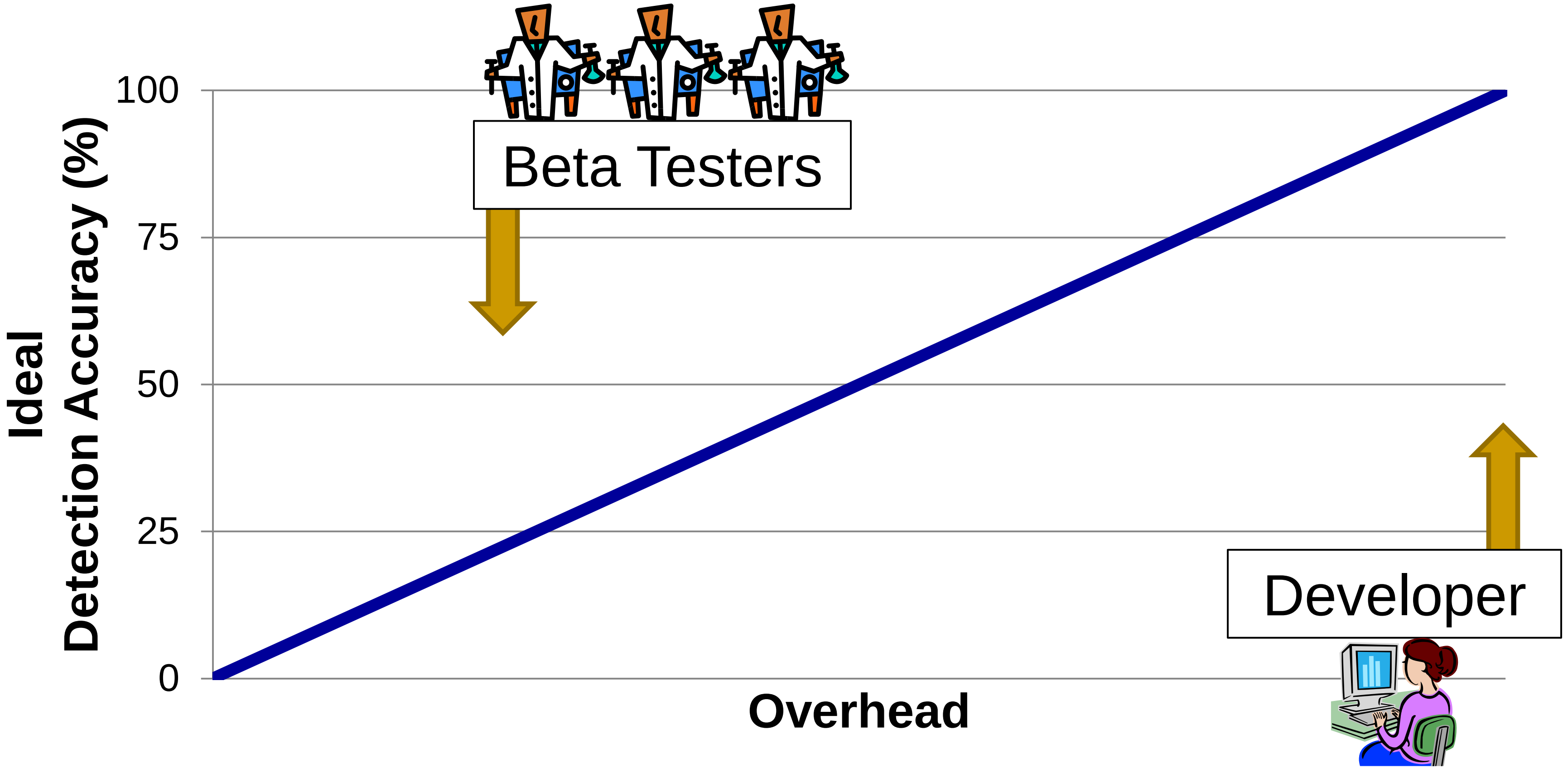
- Lower overheads by skipping some analyses



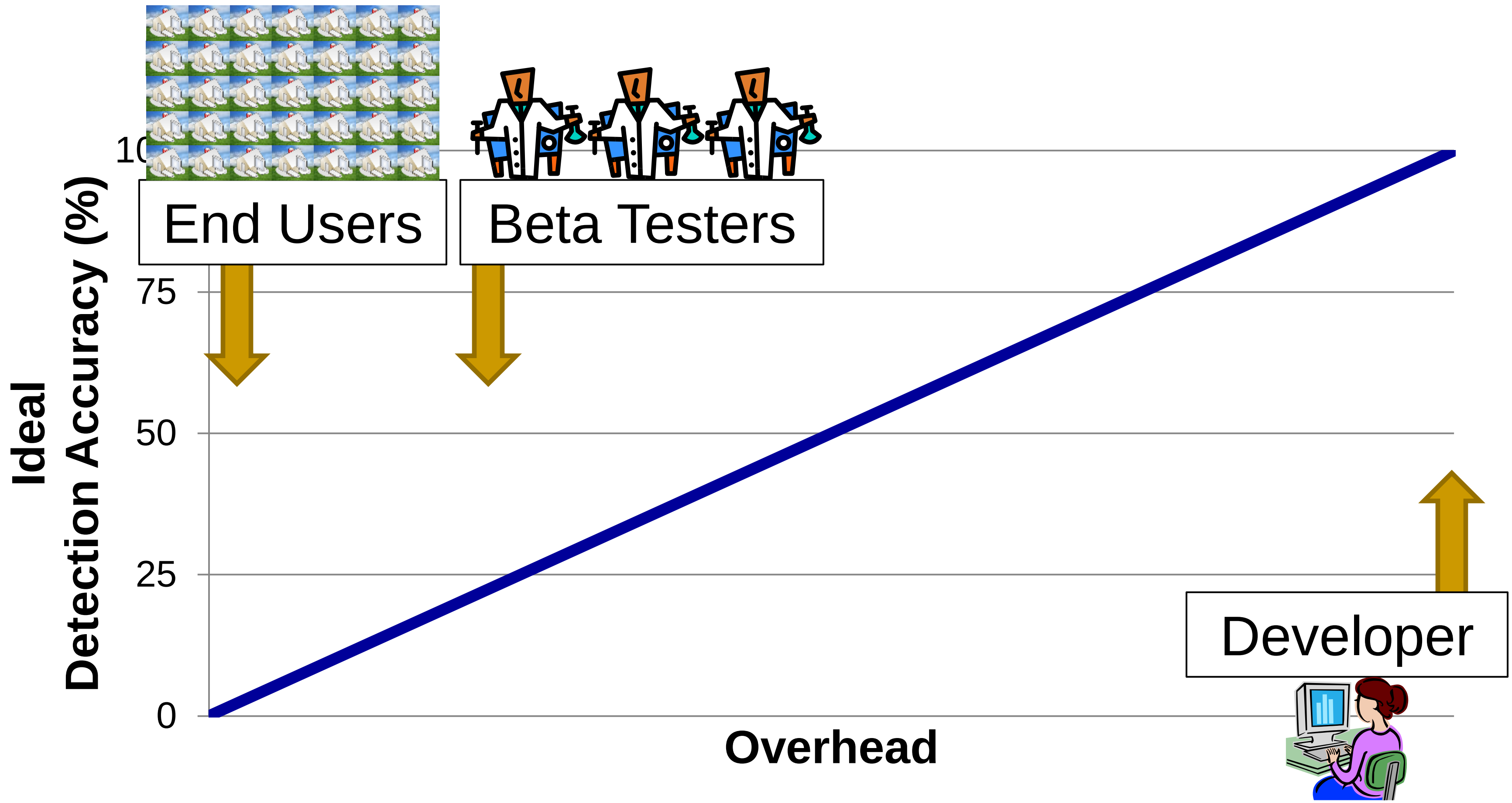
Reducing Overheads Further: Sampling



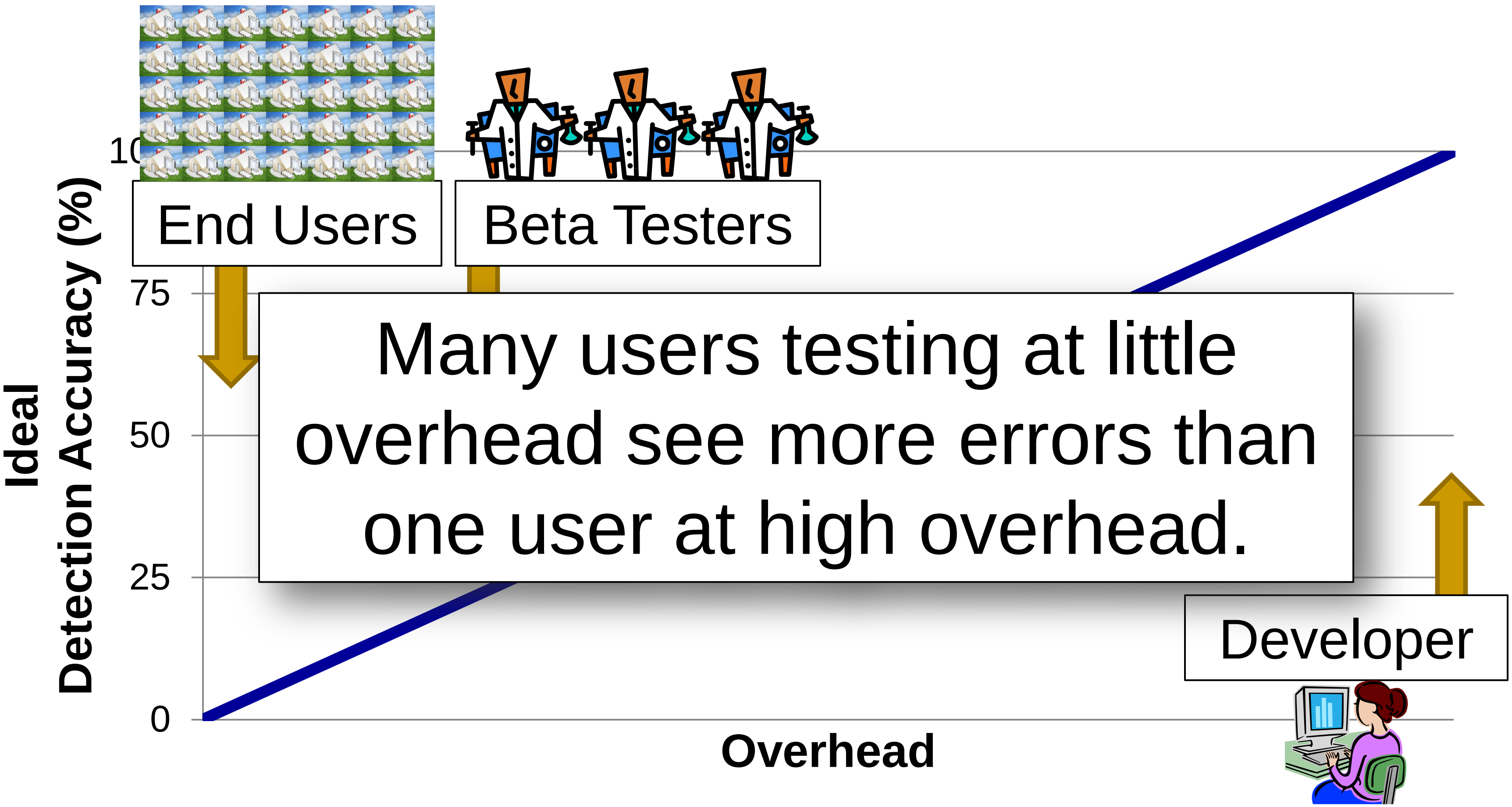
Reducing Overheads Further: Sampling



Sampling Allows Distribution



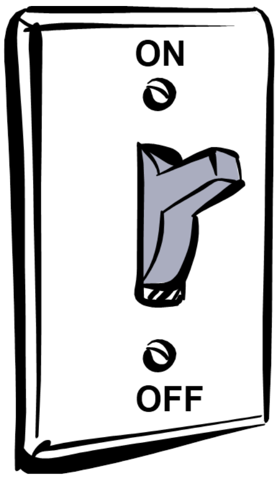
Sampling Allows Distribution



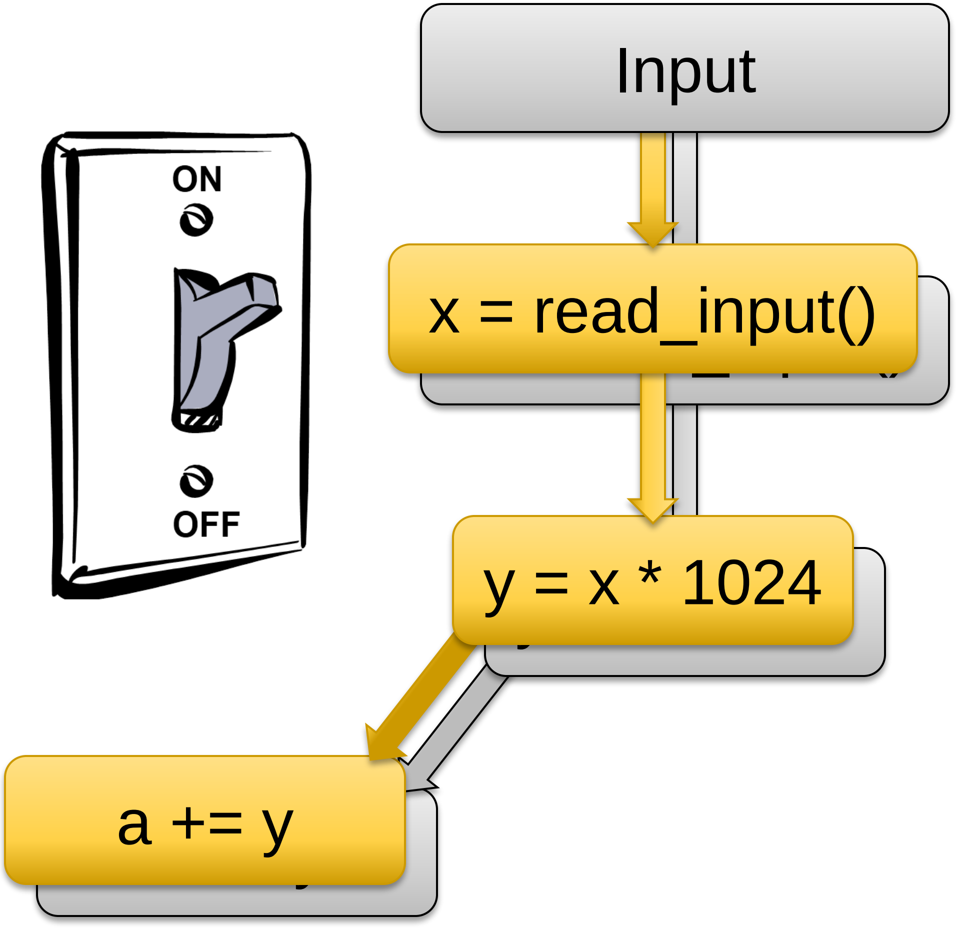
Many users testing at little overhead see more errors than one user at high overhead.

Cannot Naïvely Sample Code

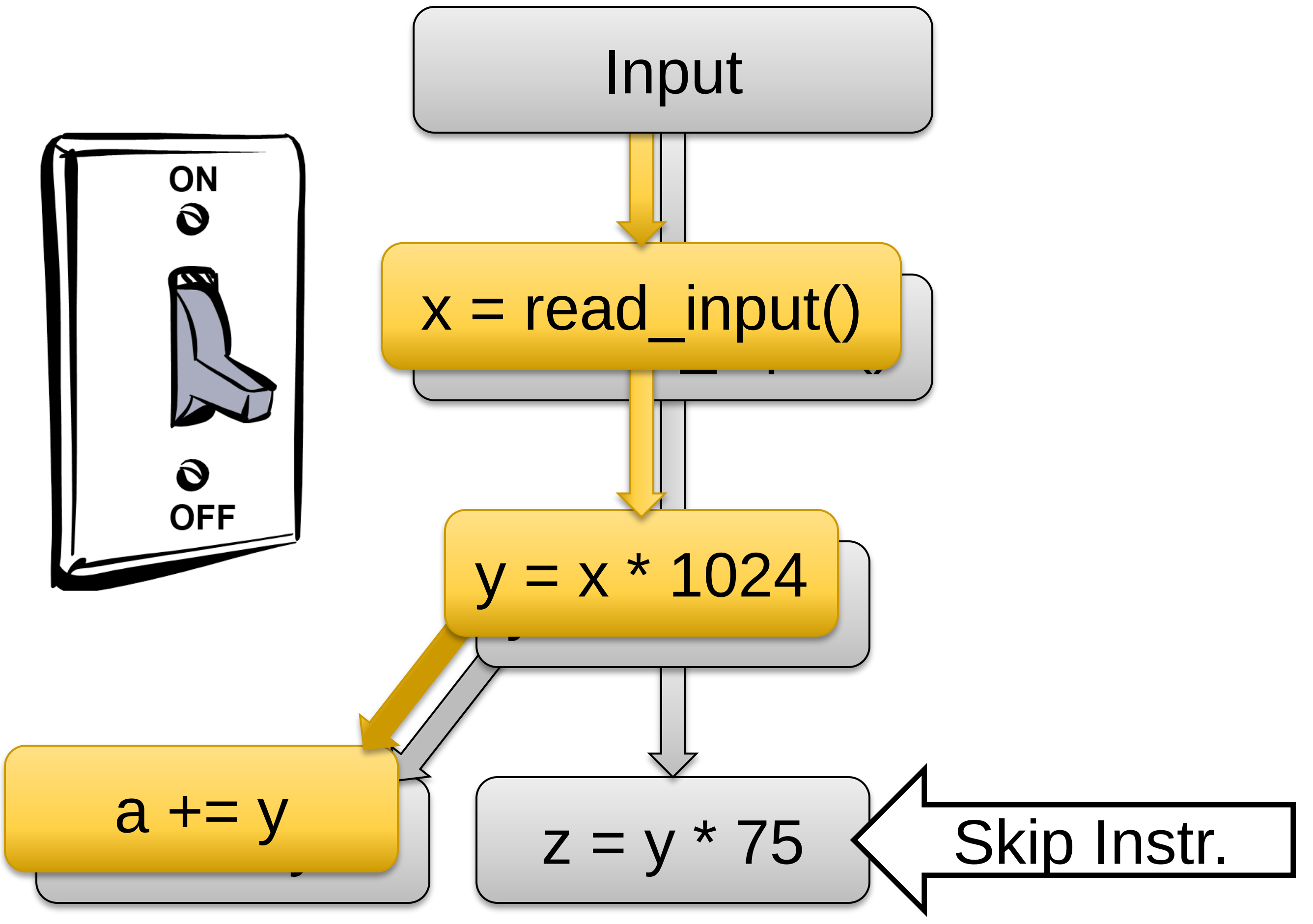
Input



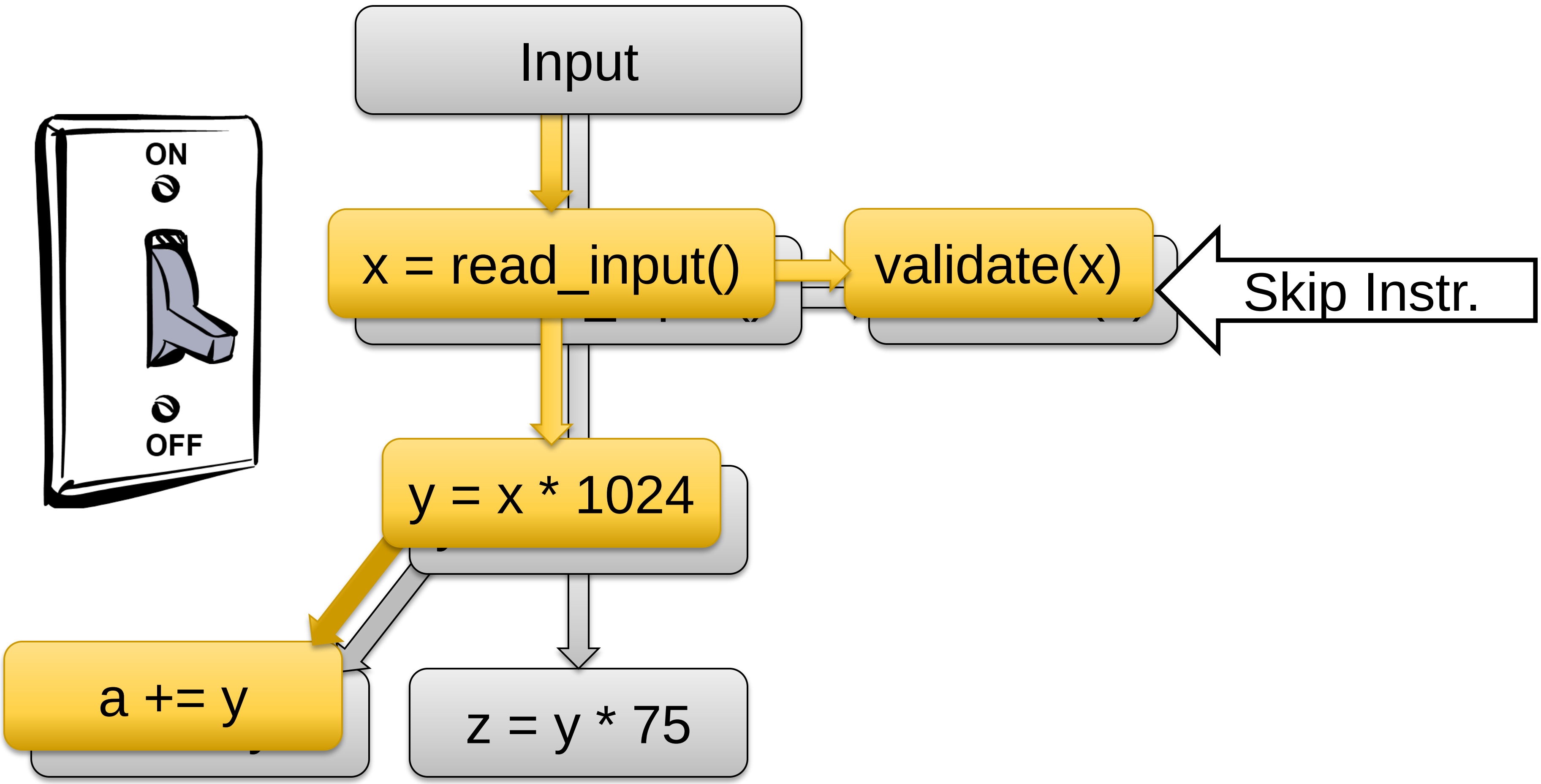
Cannot Naïvely Sample Code



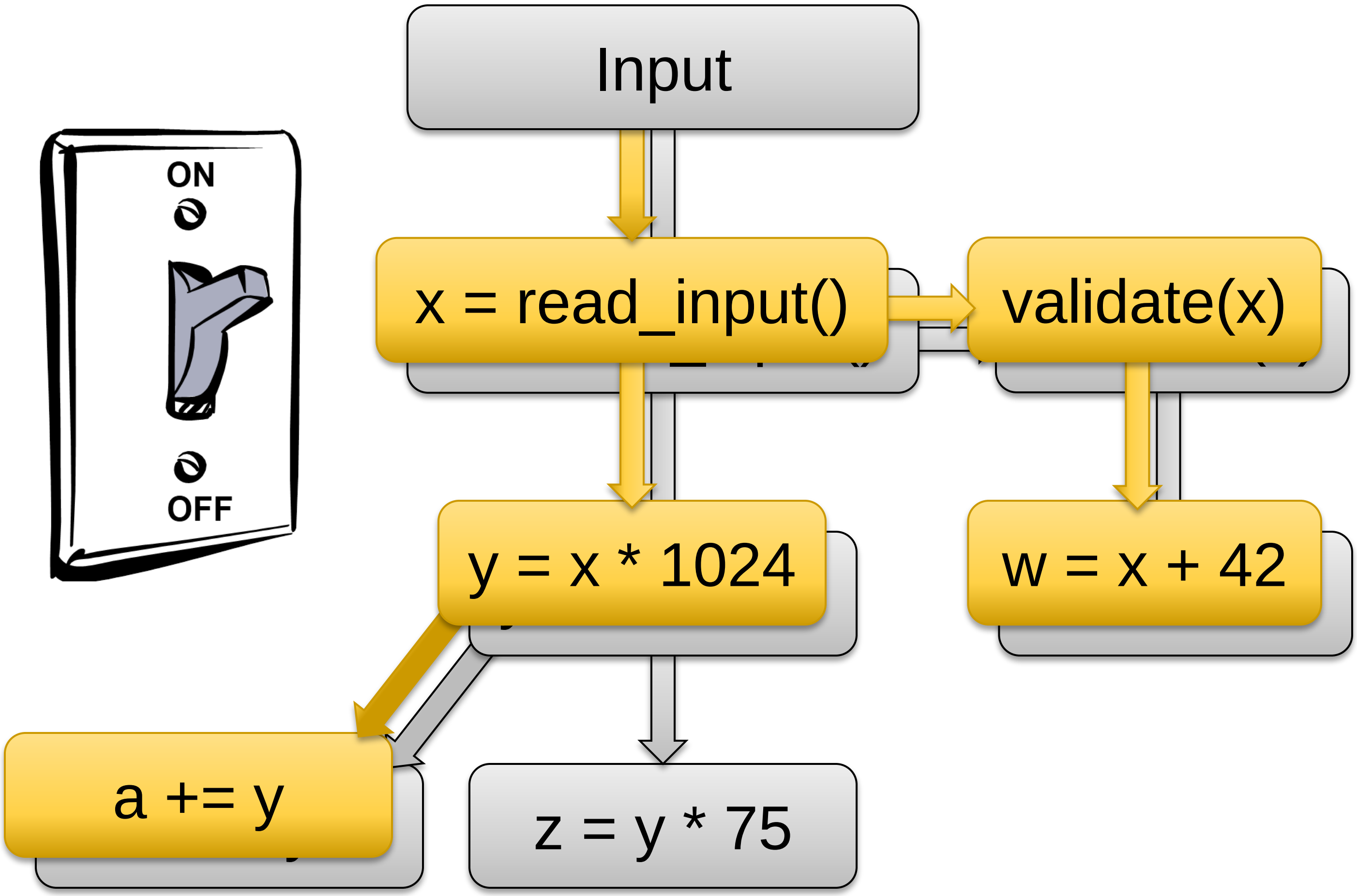
Cannot Naïvely Sample Code



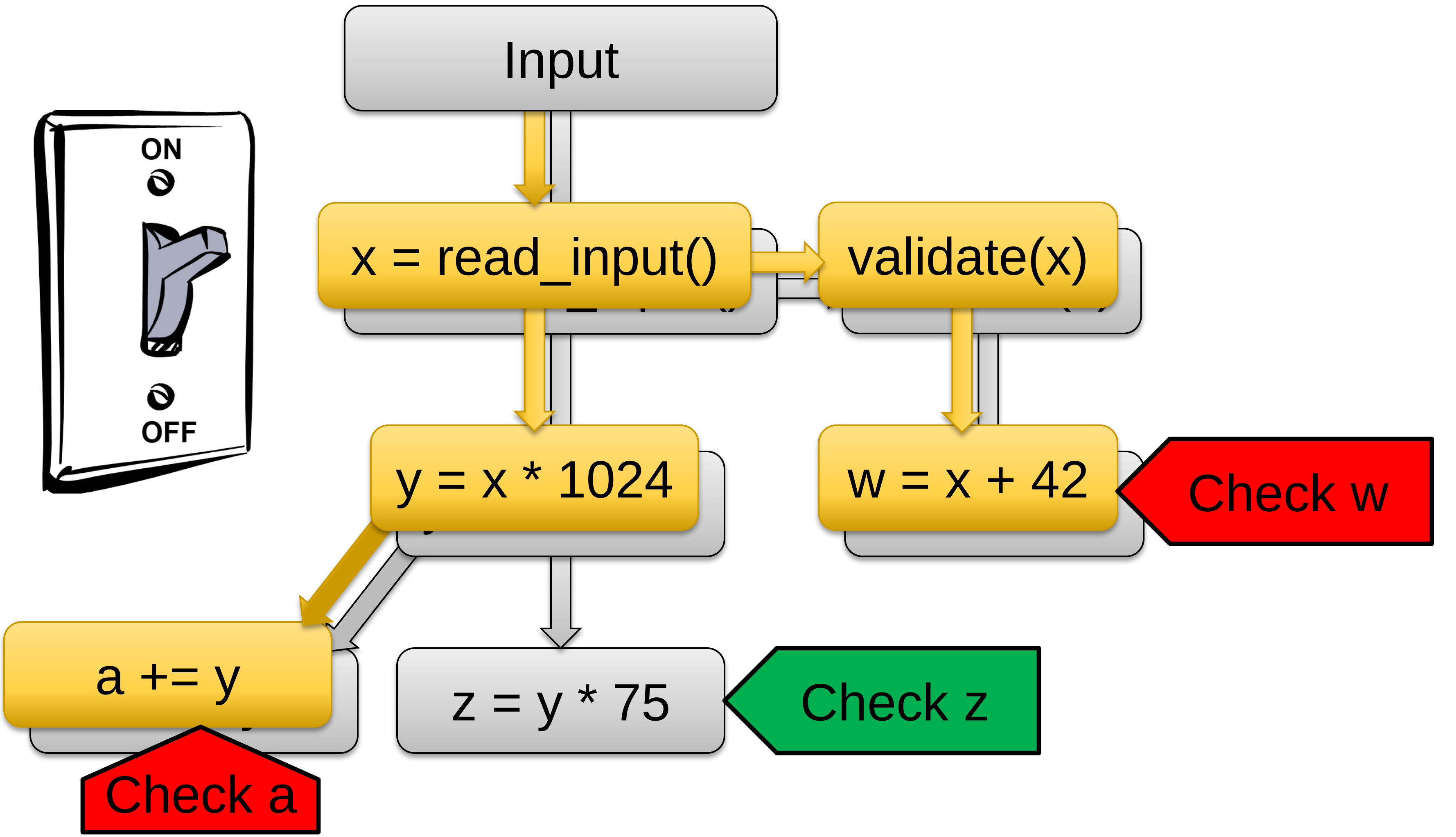
Cannot Naïvely Sample Code



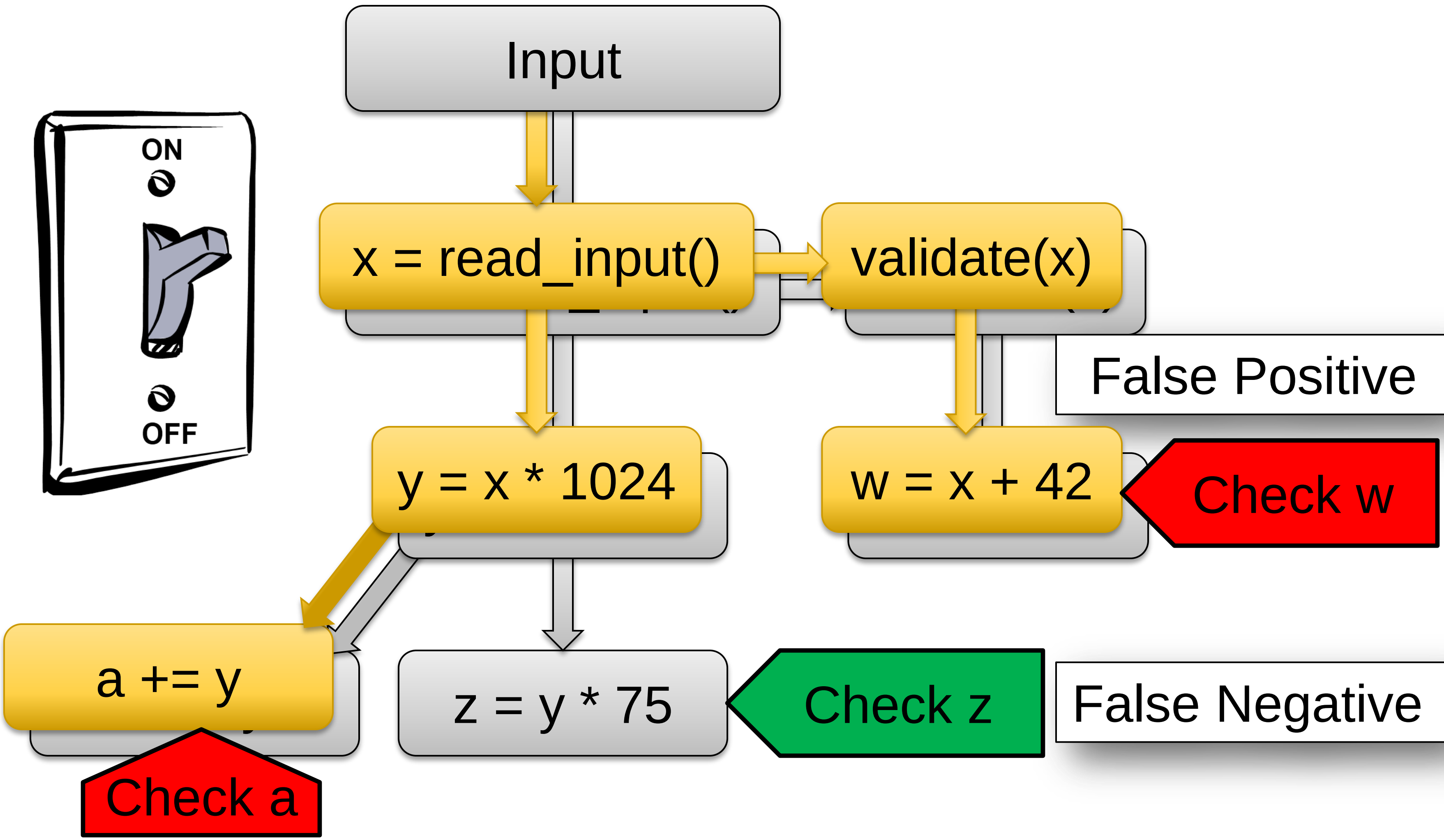
Cannot Naïvely Sample Code



Cannot Naïvely Sample Code

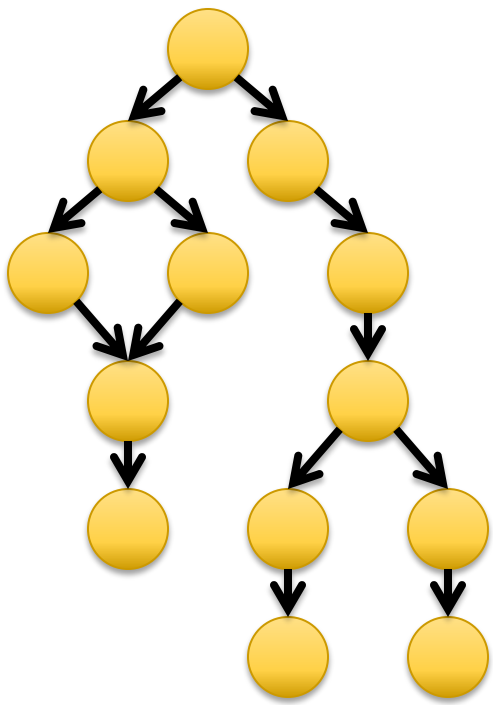


Cannot Naïvely Sample Code



Our Solution: Sample Data, not Code

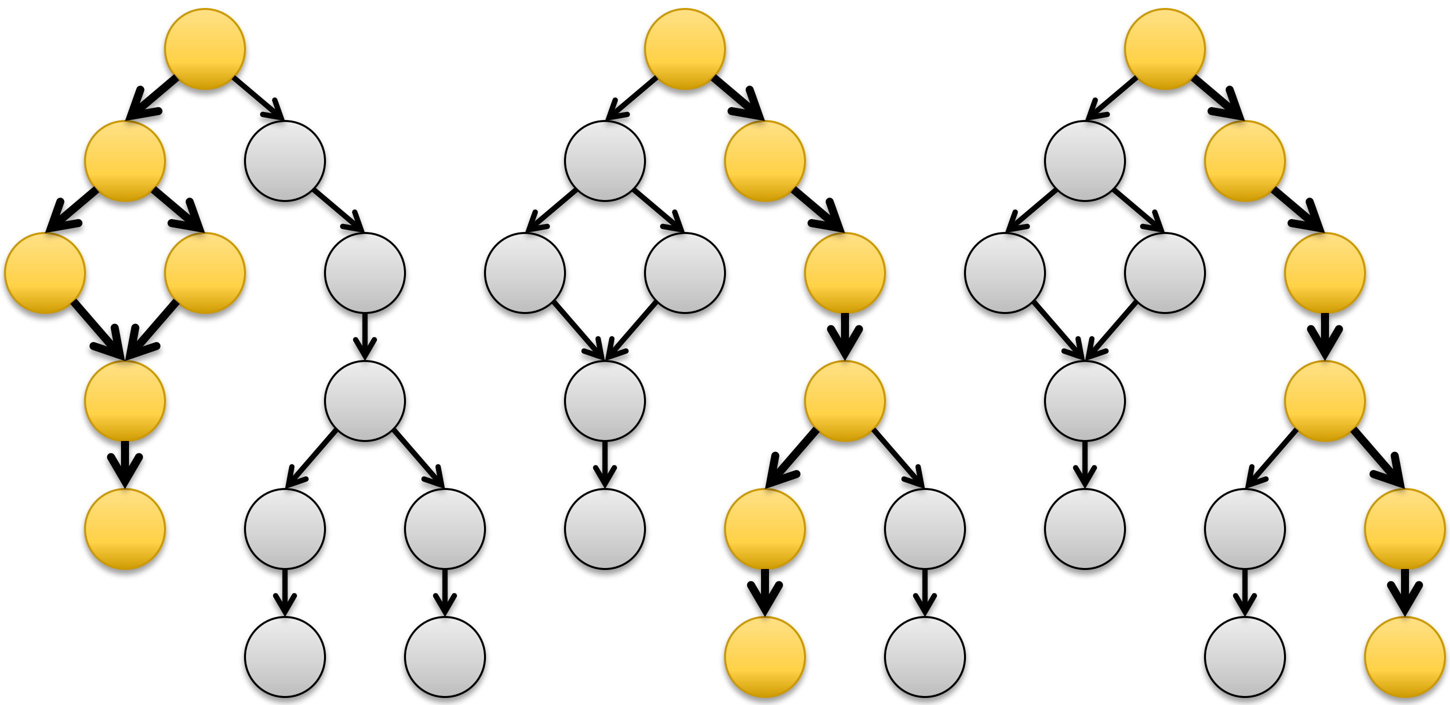
- Sampling must be aware of meta-data



- Remove meta-data from skipped dataflows
  - Prevents false positives

Our Solution: Sample Data, not Code

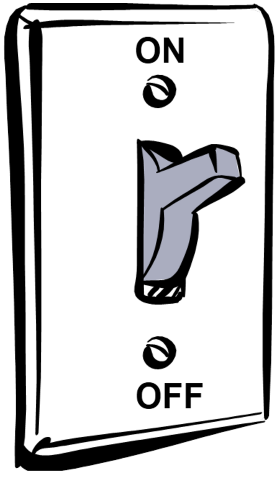
- Sampling must be aware of meta-data



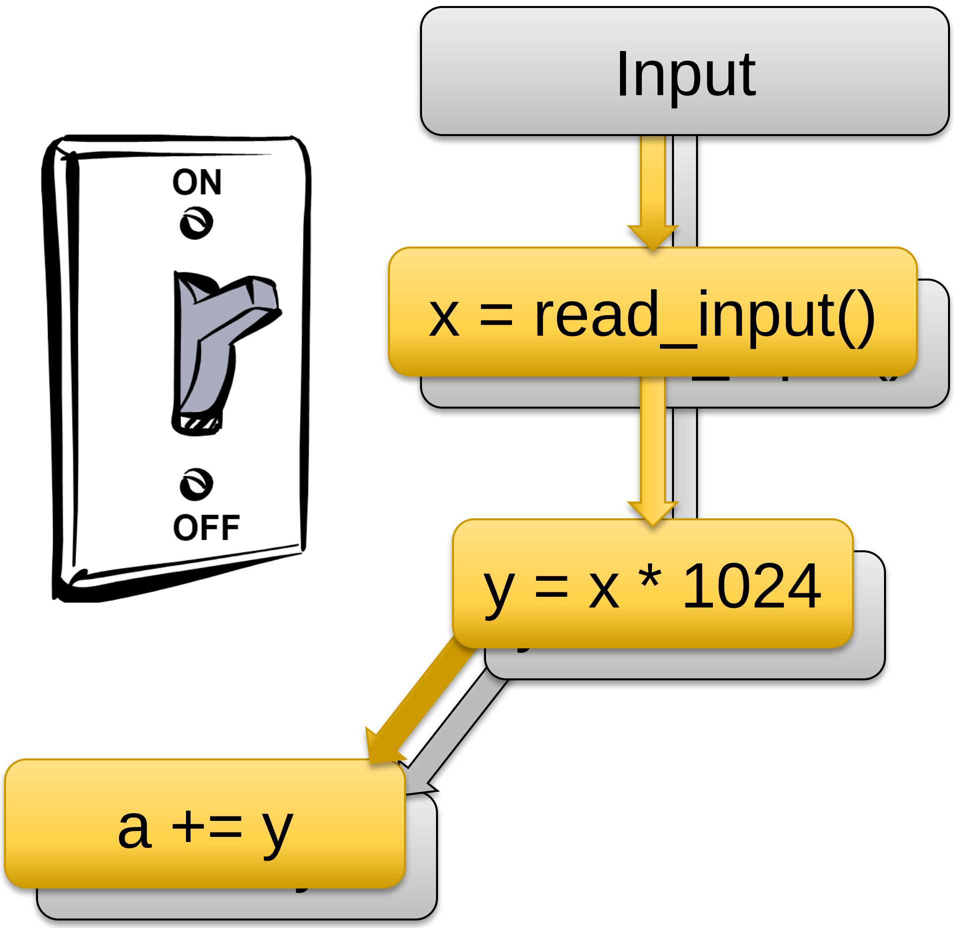
- Remove meta-data from skipped dataflows
  - Prevents false positives

Dataflow Sampling Example

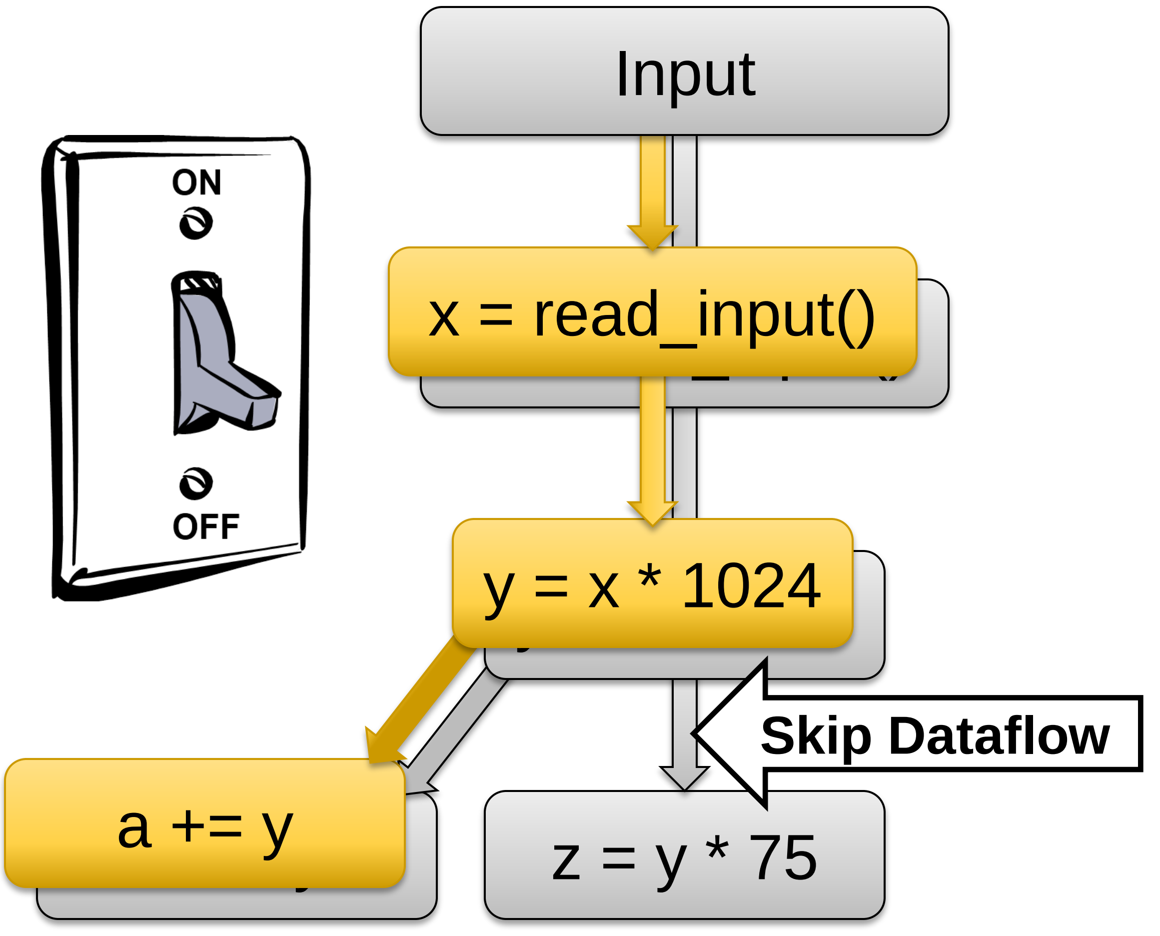
Input



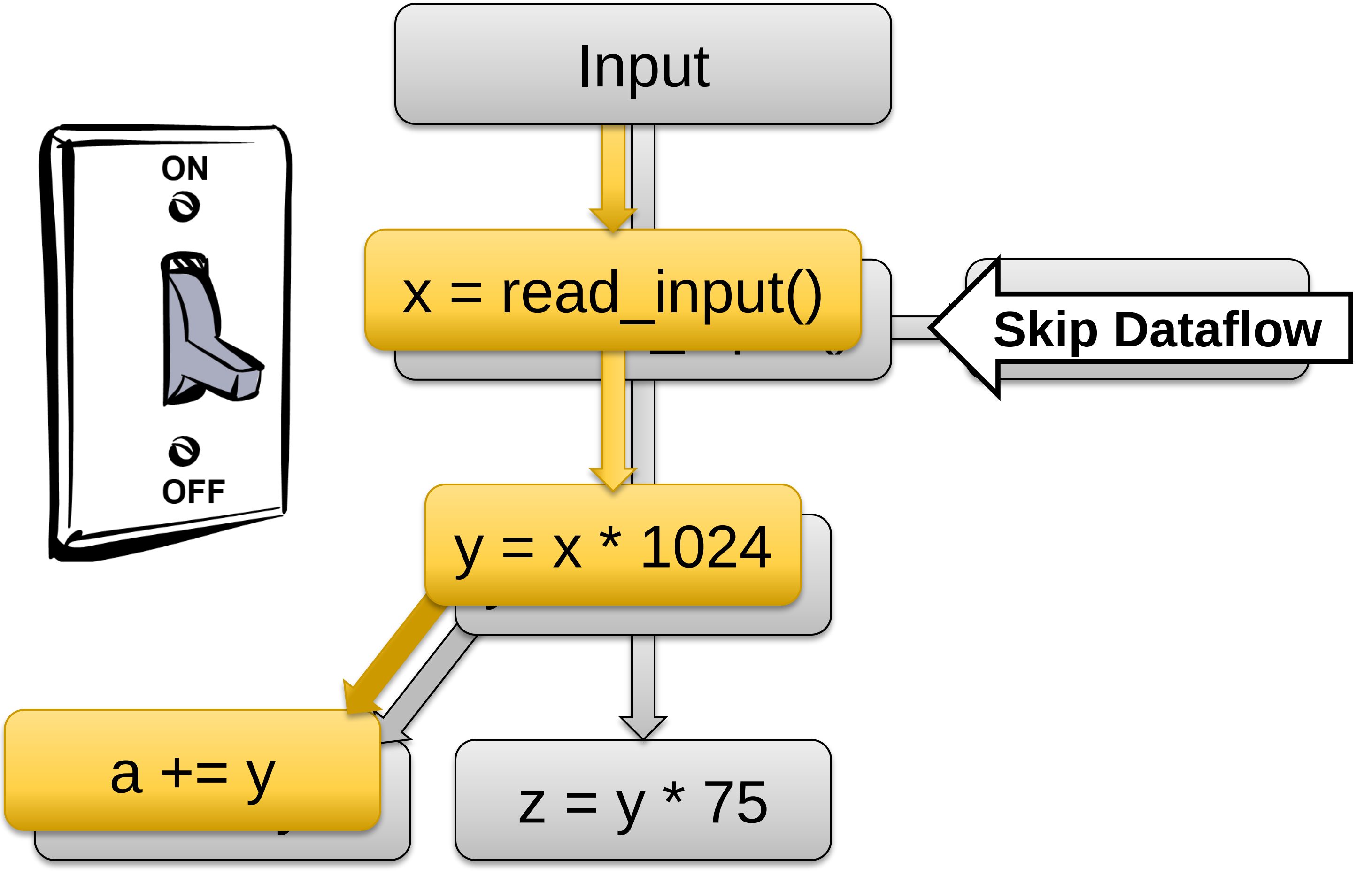
Dataflow Sampling Example



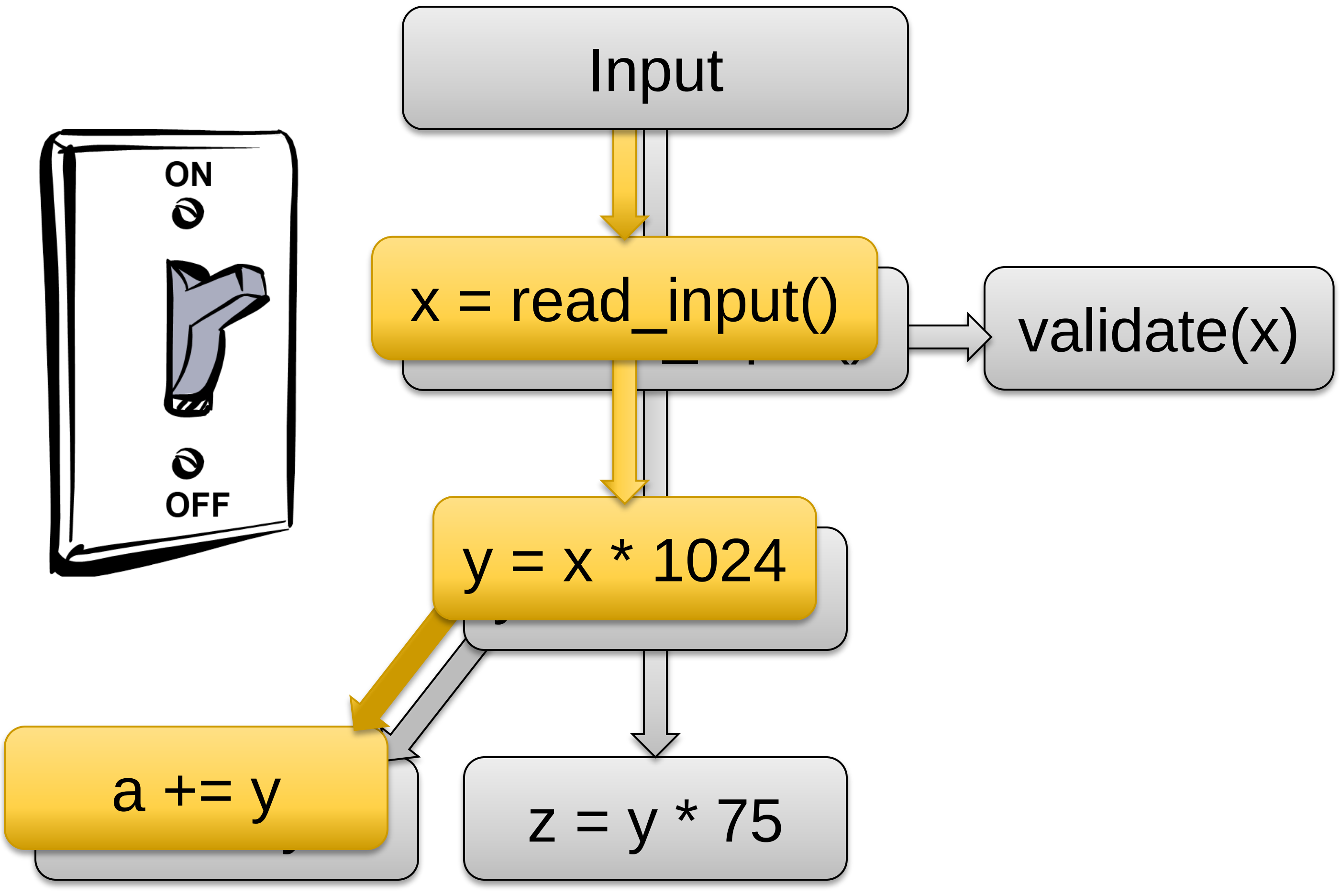
Dataflow Sampling Example



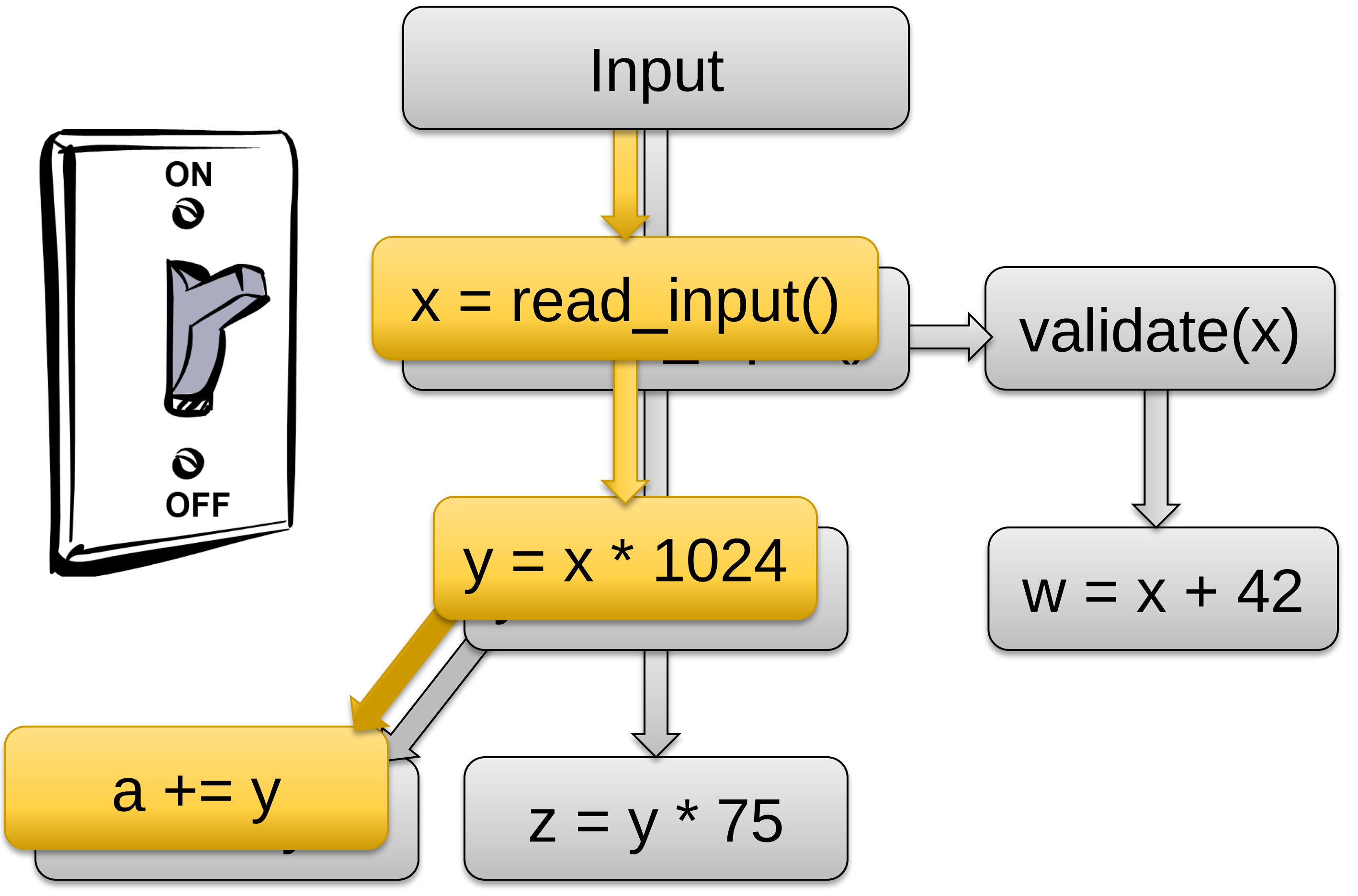
Dataflow Sampling Example



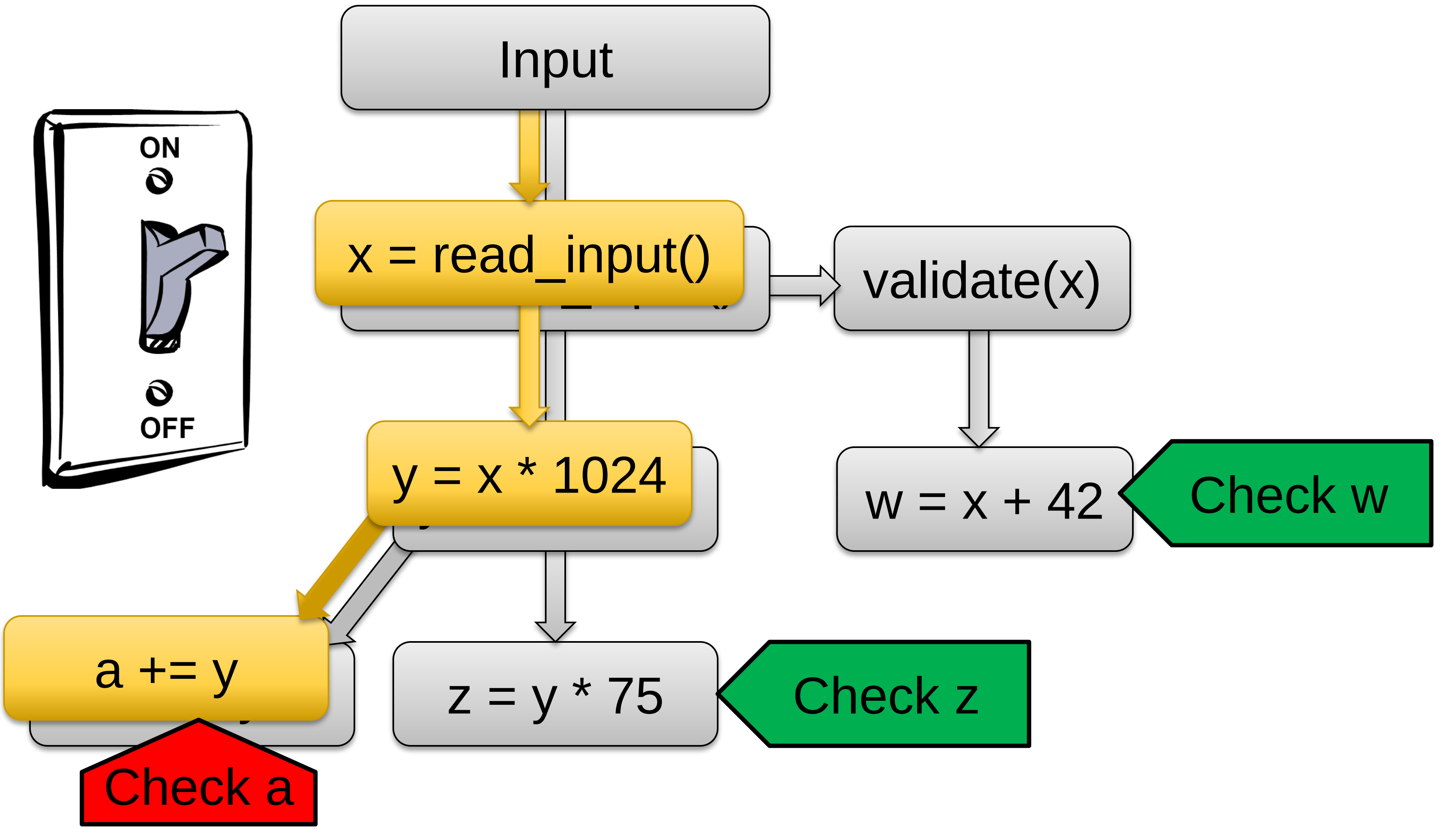
Dataflow Sampling Example



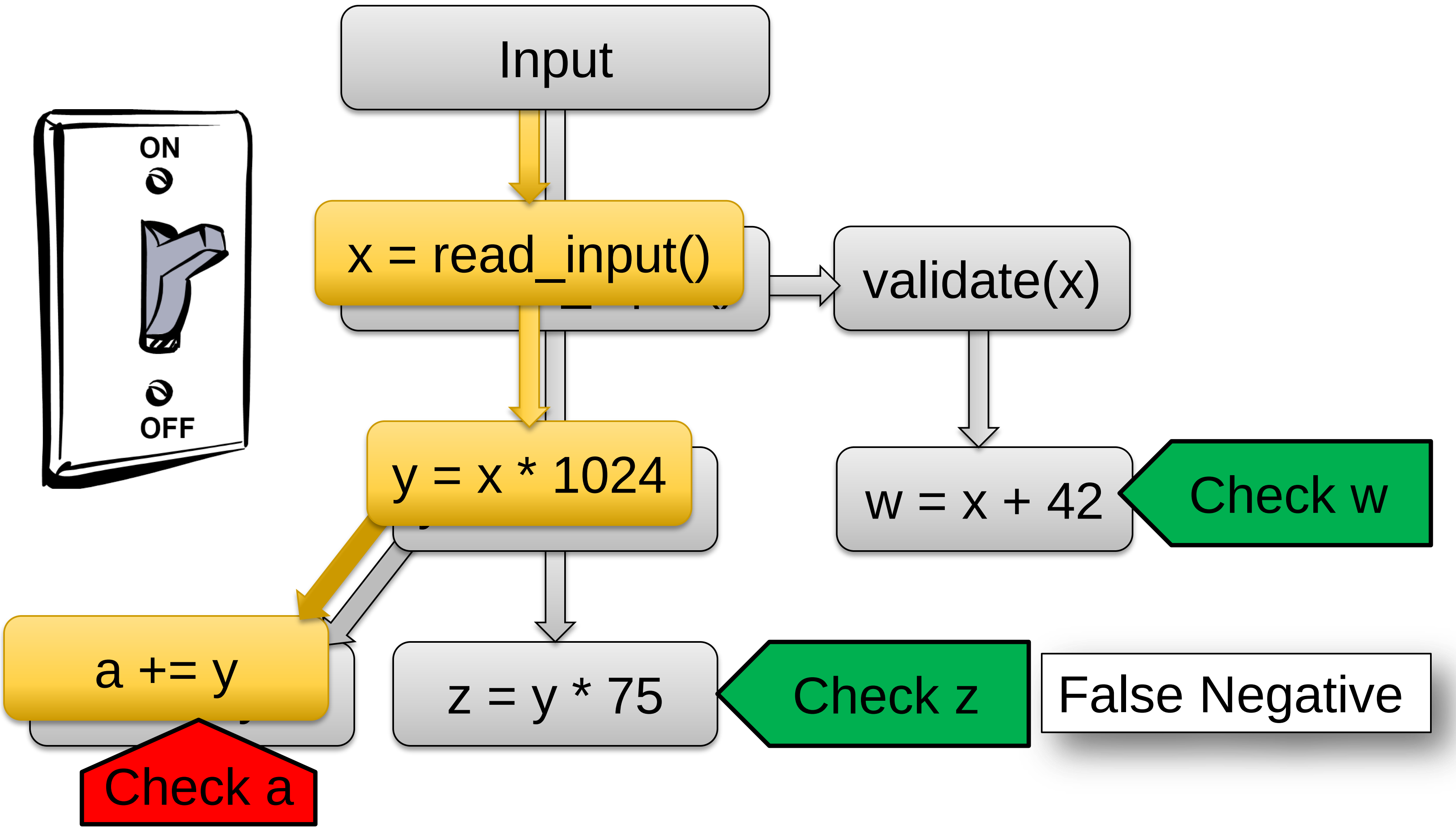
Dataflow Sampling Example



Dataflow Sampling Example

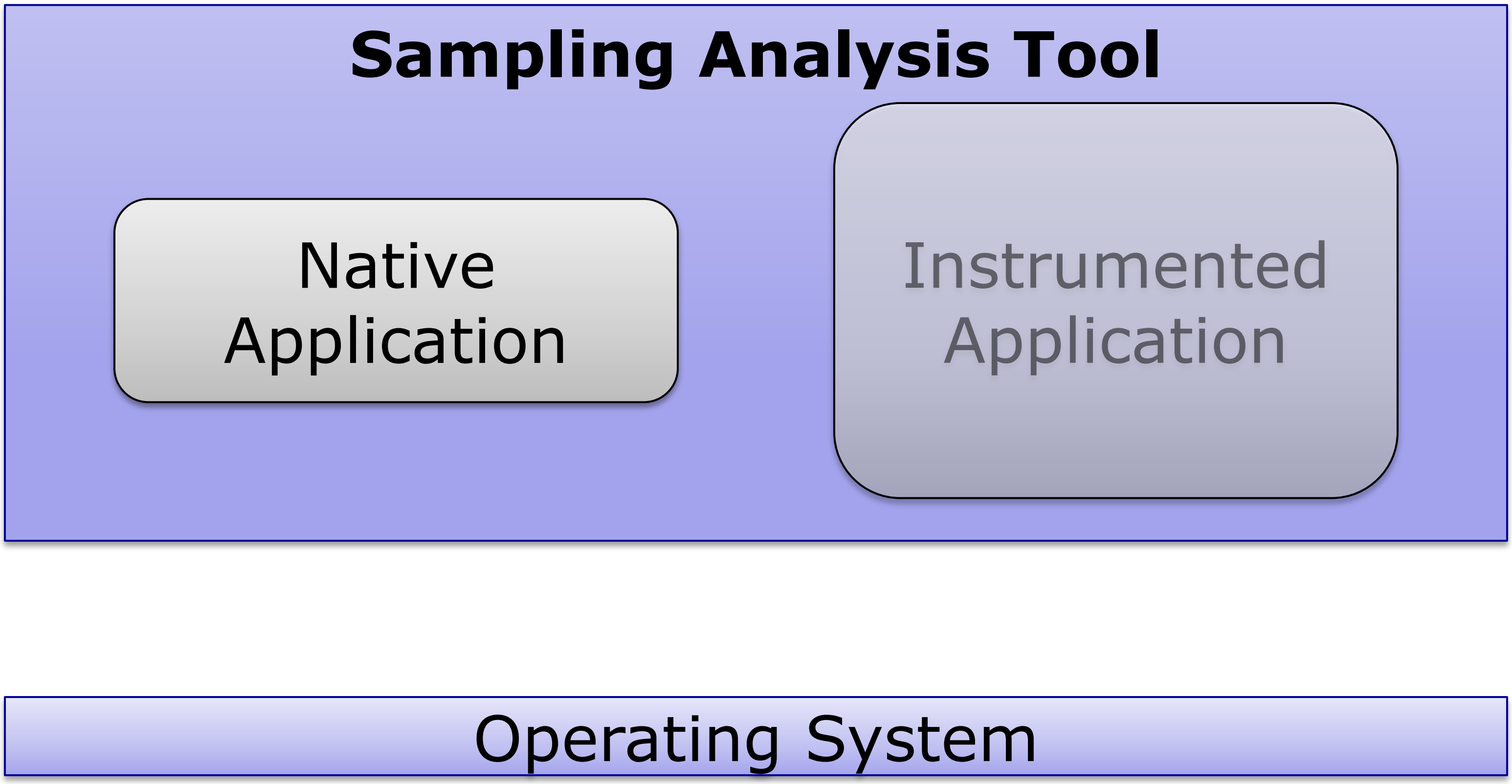


Dataflow Sampling Example



Dataflow Sampling

- **Remove** dataflows if execution is too slow



Dataflow Sampling

- **Remove** dataflows if execution is too slow

Sampling Analysis Tool

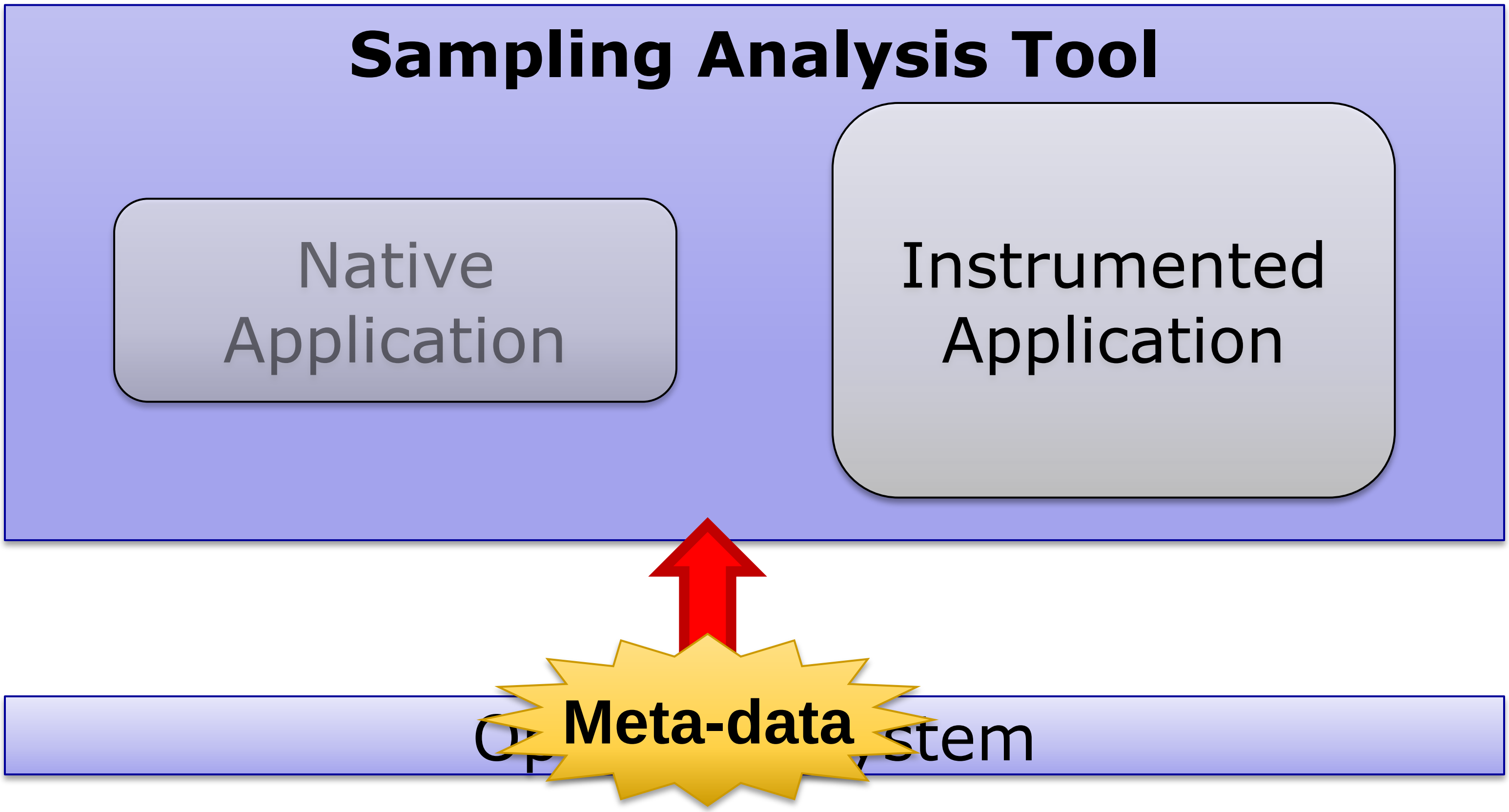
Native Application

Instrumented Application

Operating System

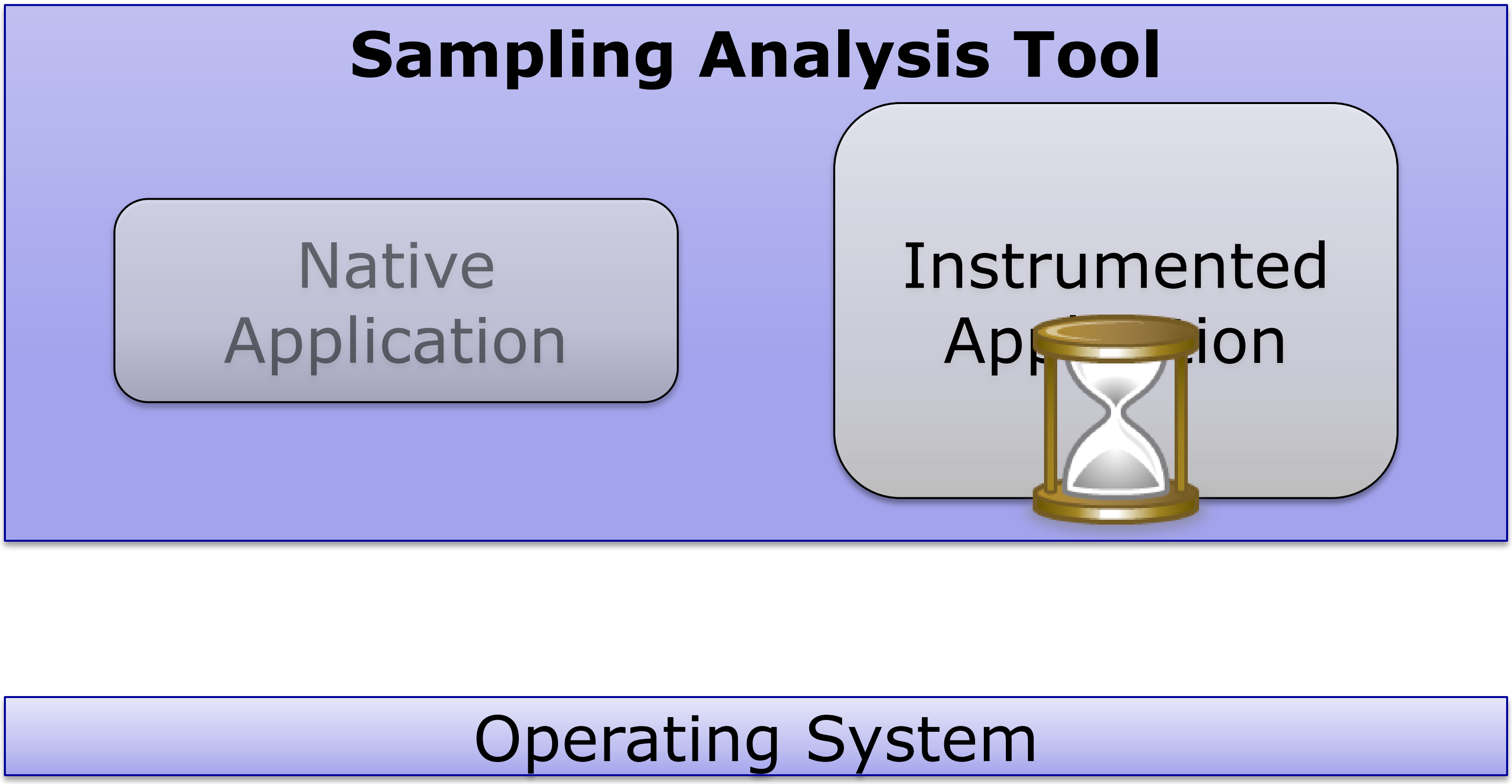
Dataflow Sampling

- **Remove** dataflows if execution is too slow



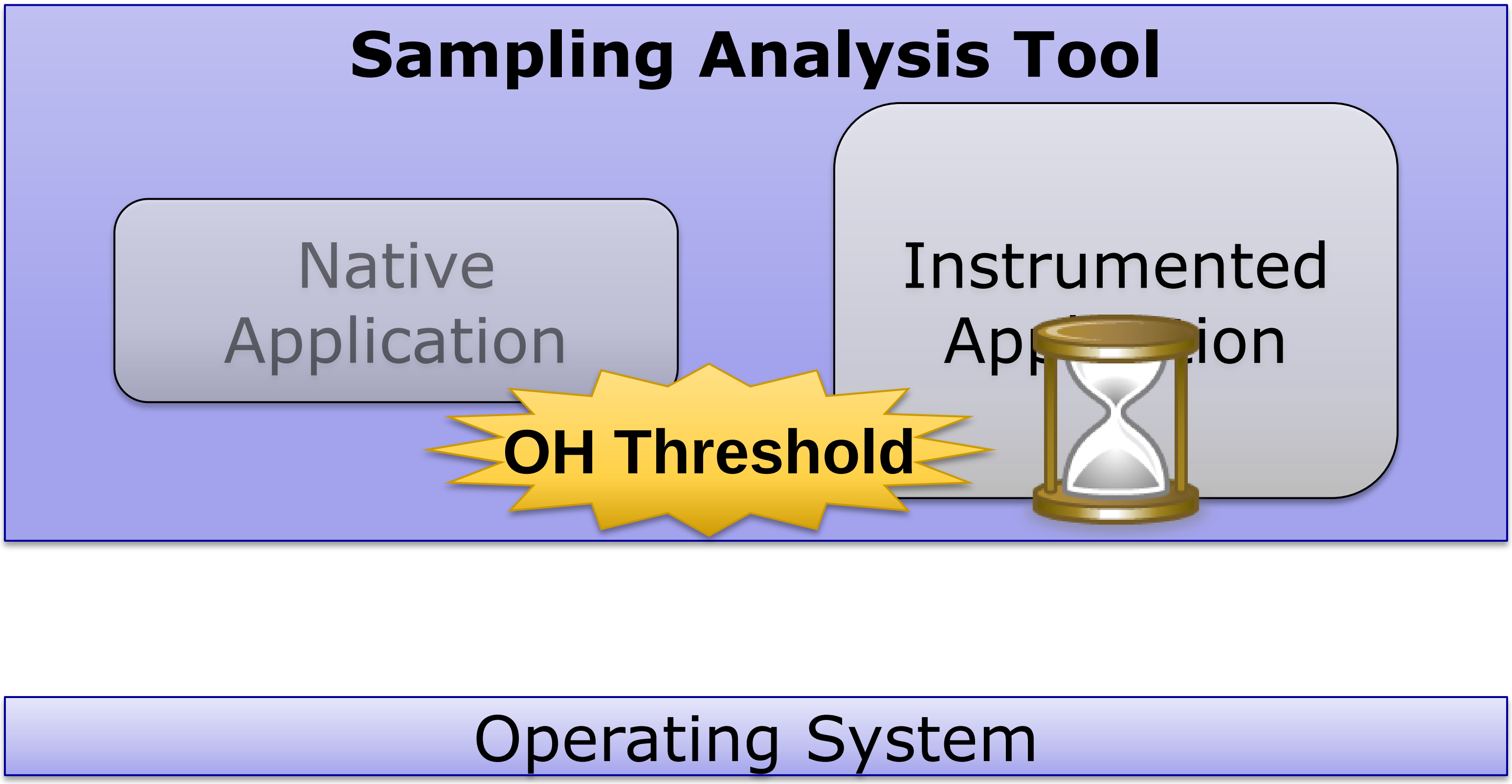
Dataflow Sampling

- **Remove** dataflows if execution is too slow



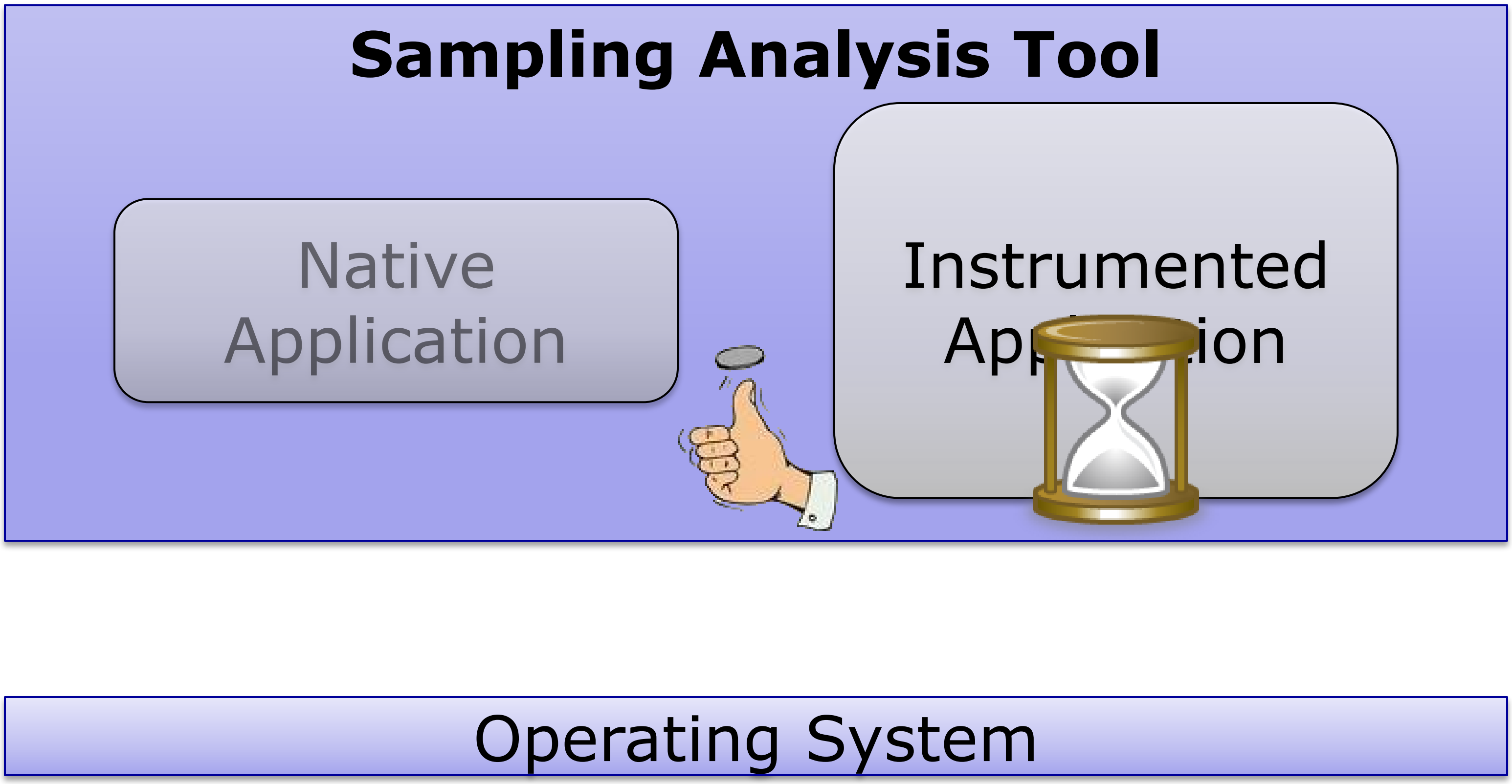
Dataflow Sampling

- **Remove** dataflows if execution is too slow



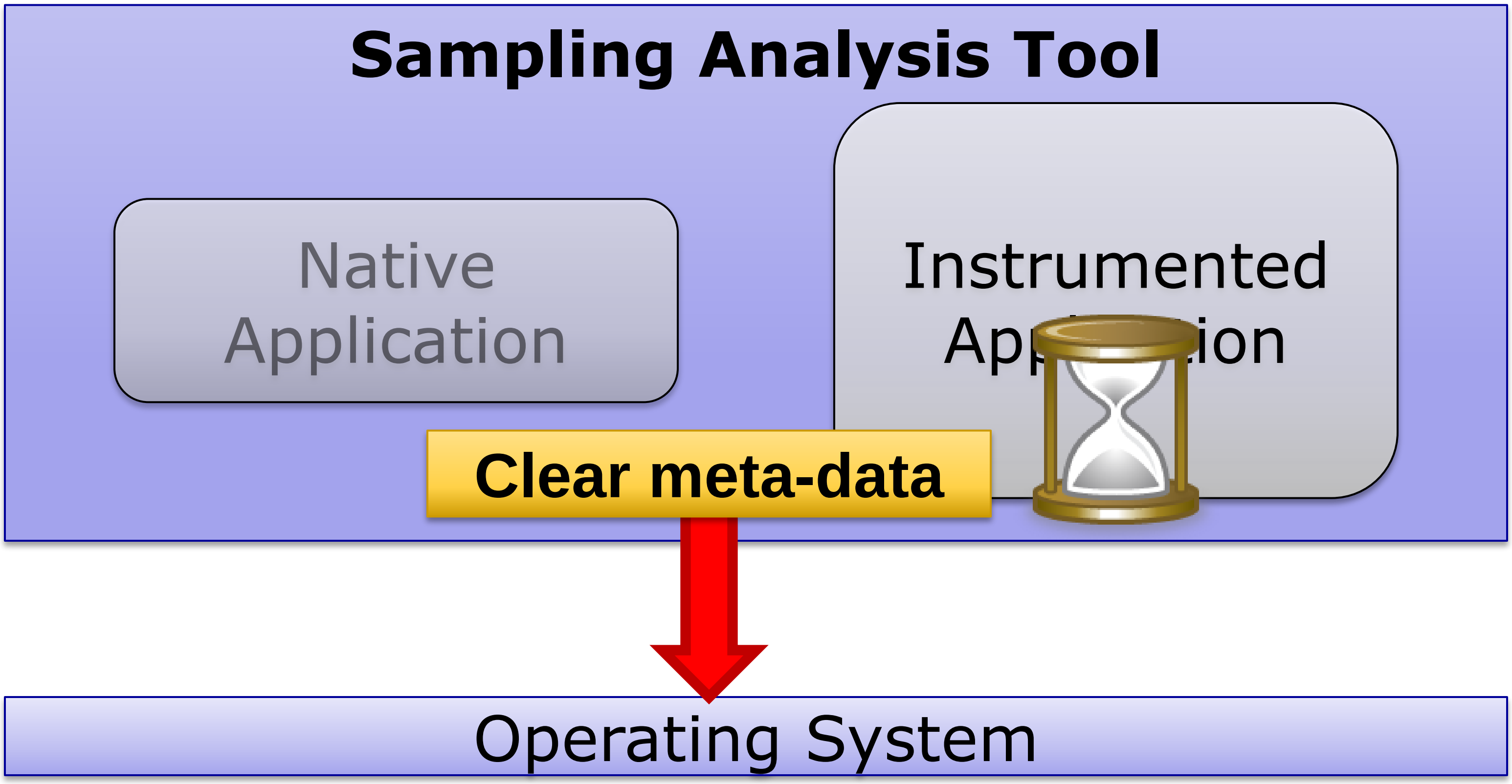
Dataflow Sampling

- **Remove** dataflows if execution is too slow



Dataflow Sampling

- **Remove** dataflows if execution is too slow



Dataflow Sampling

- **Remove** dataflows if execution is too slow

Sampling Analysis Tool

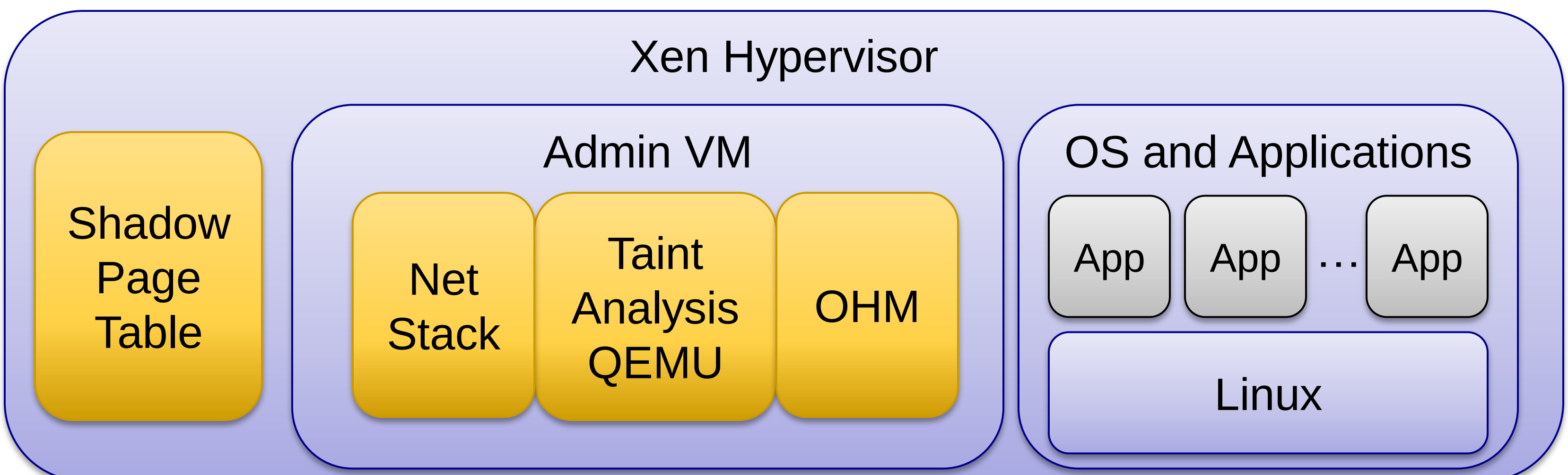
Native Application

Instrumented Application

Operating System

Prototype Setup

- Taint analysis sampling system
  - Network packets untrusted
- Xen-based demand analysis
  - Whole-system analysis with modified QEMU
- Overhead Manager (OHM) is user-controlled

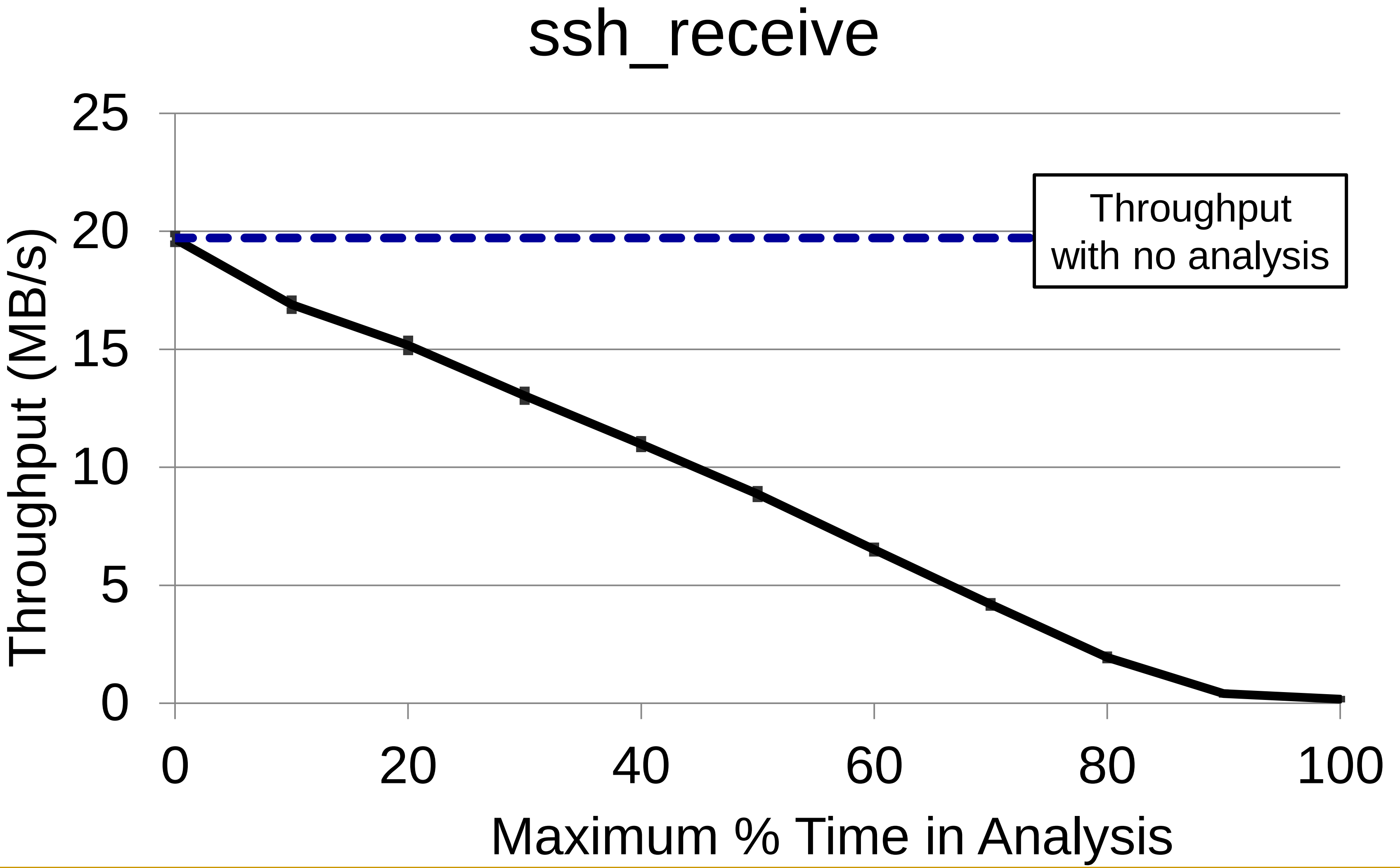


Benchmarks

- Performance – Network Throughput
  - *Example:* ***ssh_receive***
- Accuracy of Sampling Analysis
  - Real-world Security Exploits

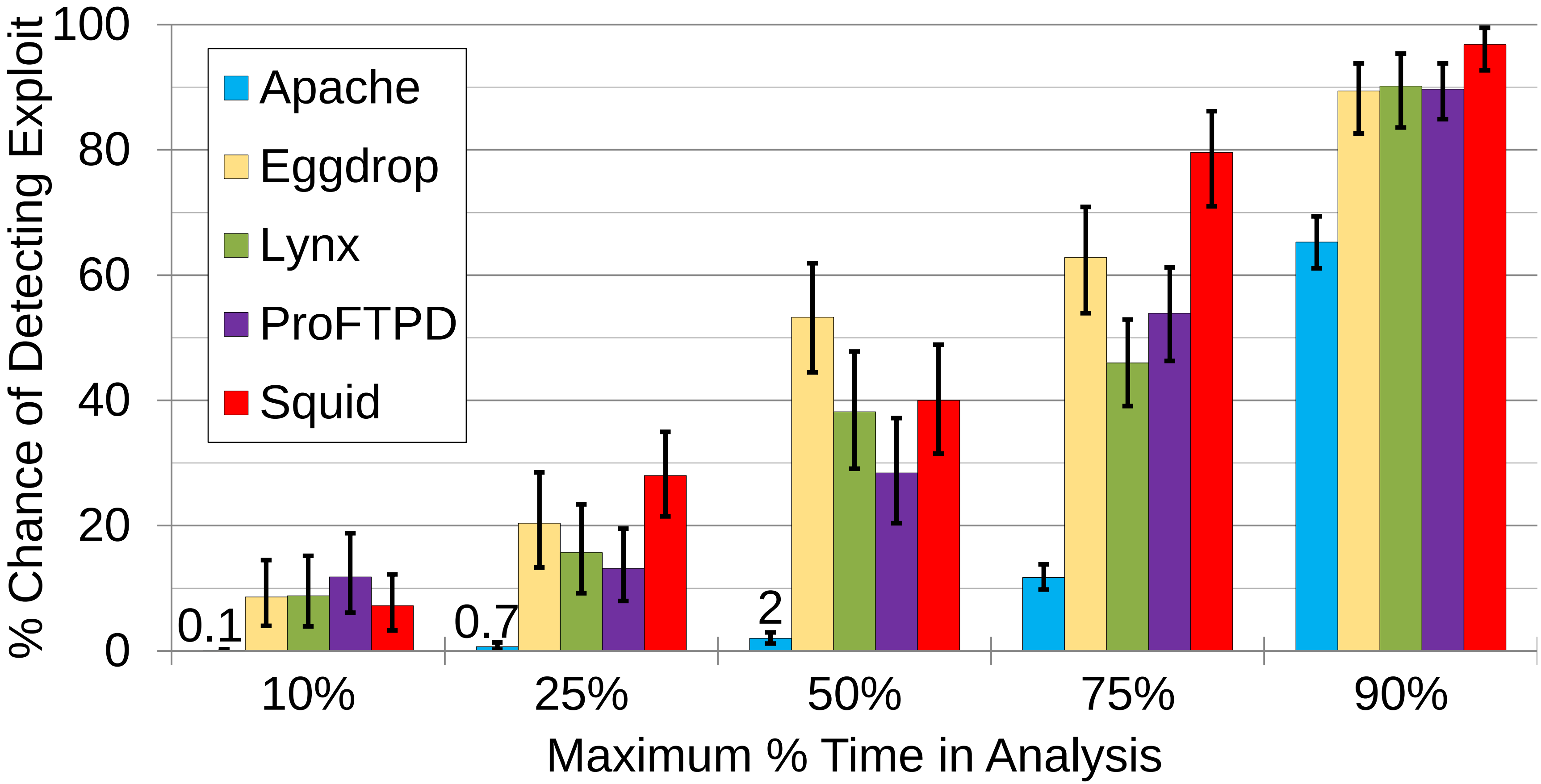
Name	Error Description
Apache	Stack overflow in Apache Tomcat JK Connector
Eggdrop	Stack overflow in Eggdrop IRC bot
Lynx	Stack overflow in Lynx web browser
ProFTPD	Heap smashing attack on ProFTPD Server
Squid	Heap smashing attack on Squid proxy server

Performance of Dataflow Sampling



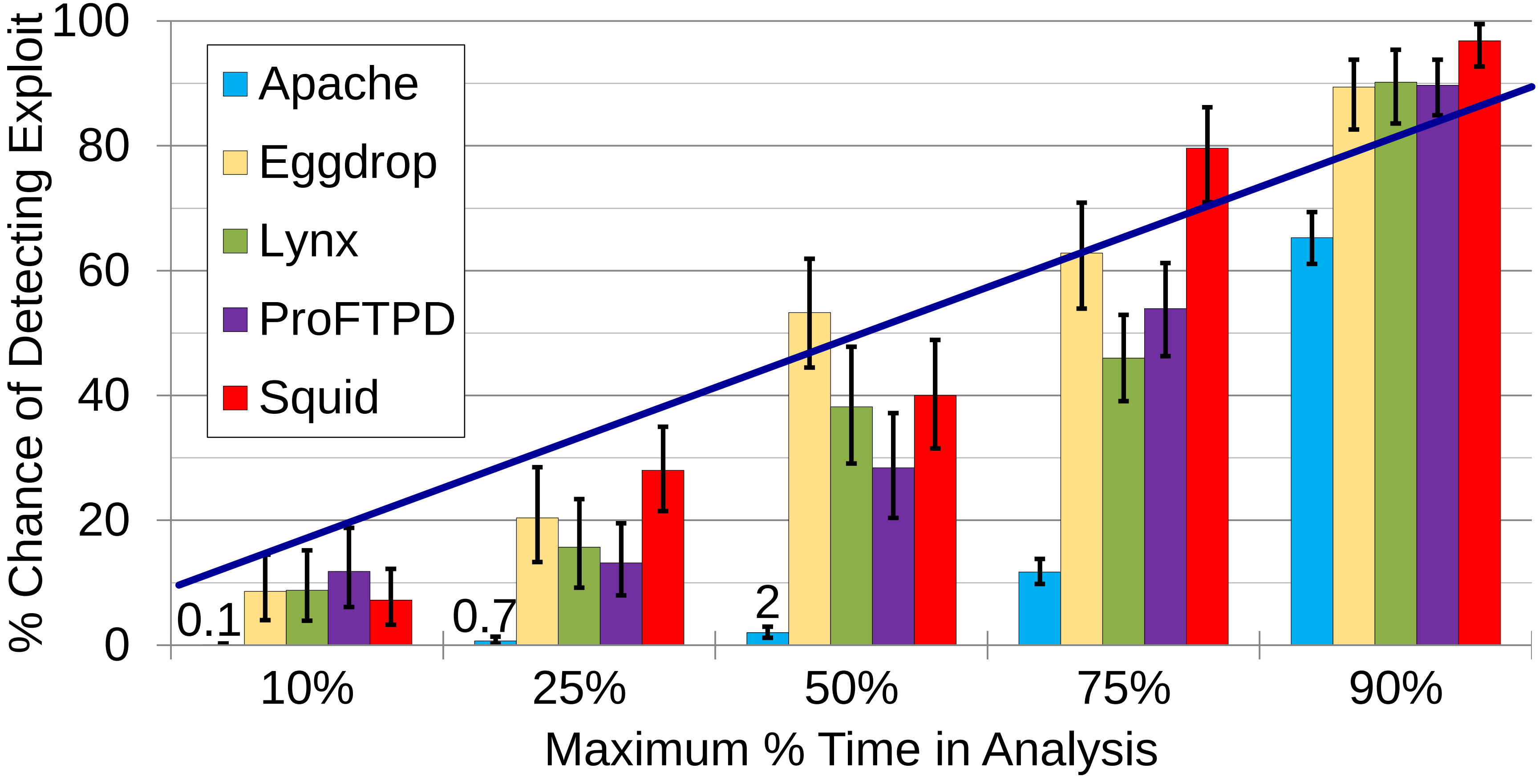
Accuracy with Background Tasks

ssh_receive running in background



Accuracy with Background Tasks

ssh_receive running in background



BACKUP SLIDES

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%)

Width Test