Sampling Dynamic Dataflow Analyses

Joseph L. Greathouse
Advanced Computer Architecture Laboratory
University of Michigan

UNIVERSITY OF MICHIGAN

University of British Columbia
June 10, 2011

Software Errors Abound

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

A problem has been detected and windows has been shut down to prevent damage
to your computer.

The problem seems to be caused by the following file: SPCMDCON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any windows updates you might need.

If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

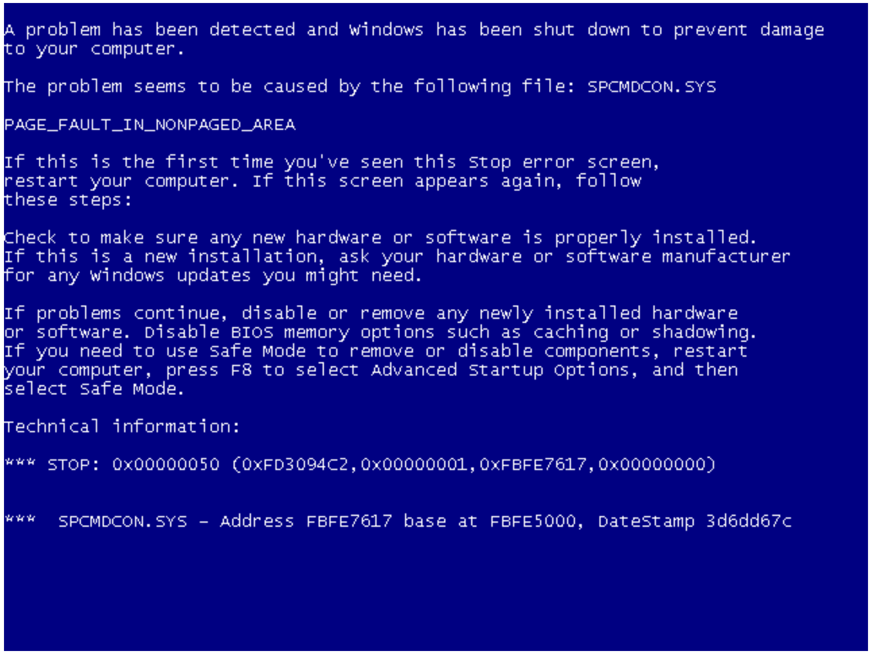
Technical information:

*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)

*** SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c

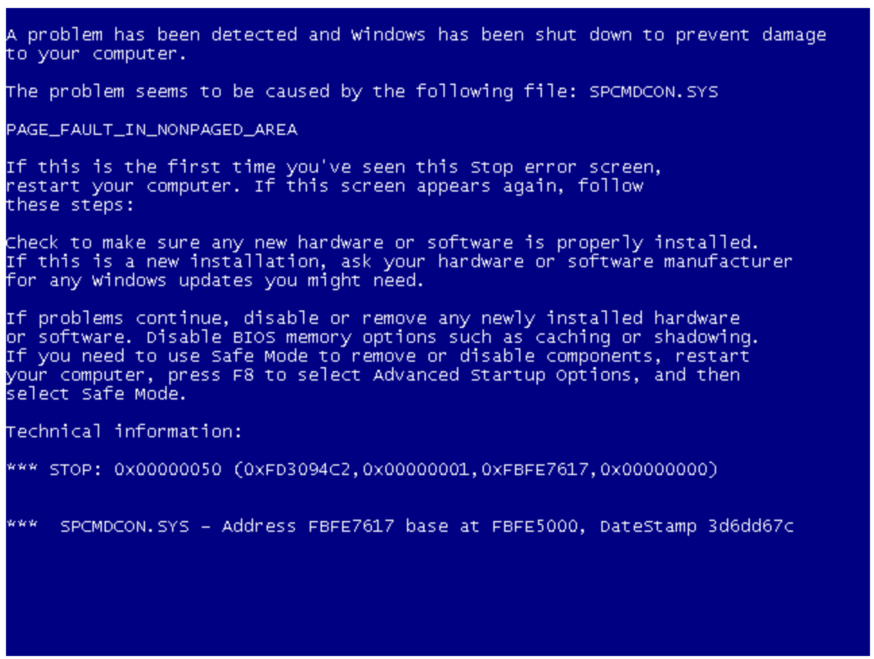
Software Errors Abound

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



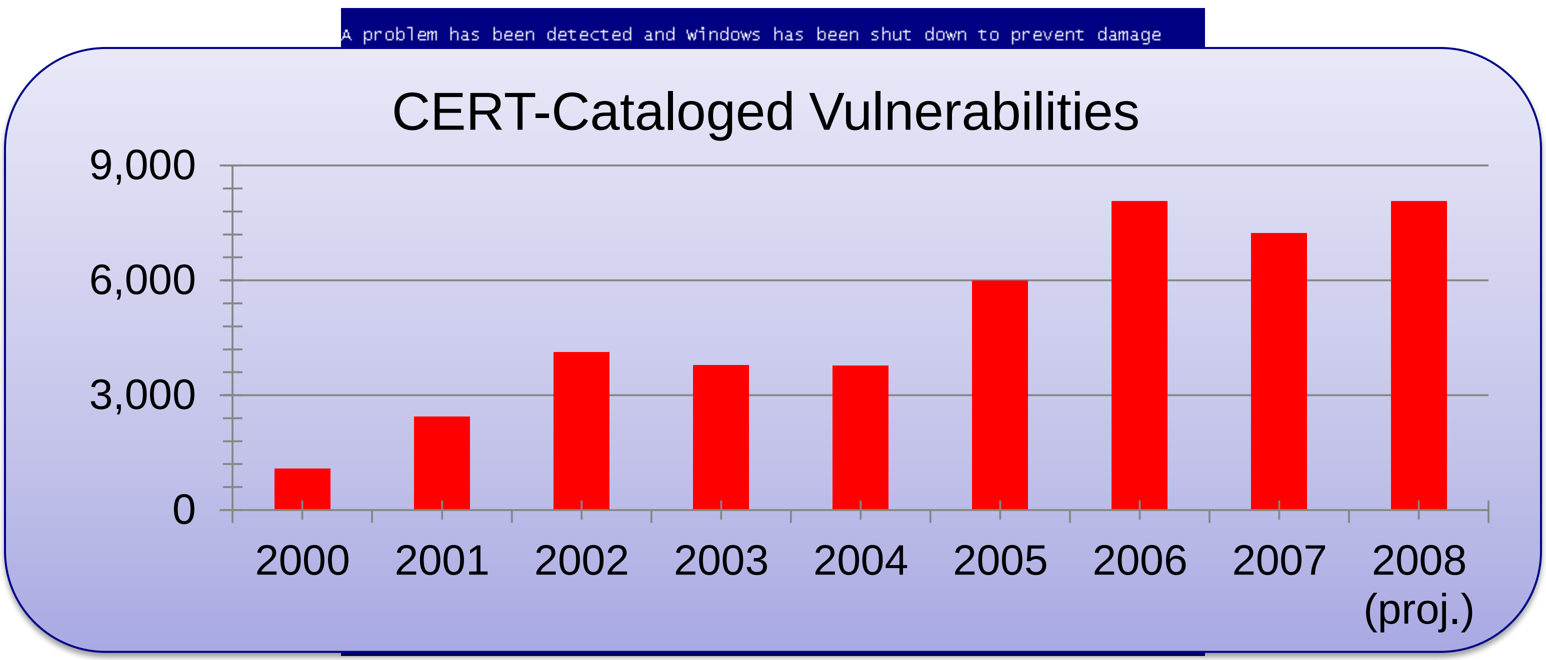
Software Errors Abound

- NIST: SW errors cost U.S. ~$60 billion/year as of 2002
- FBI CCS: Security Issues $67 billion/year as of 2005
  - $>\frac{1}{3}$ from viruses, network intrusion, etc.



Software Errors Abound

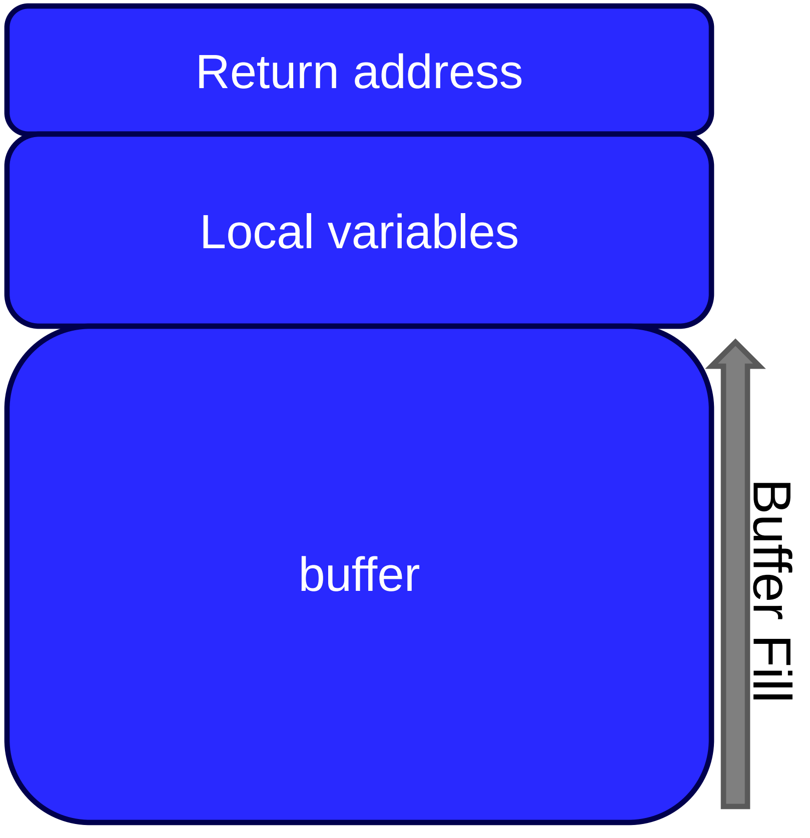
- NIST: SW errors cost U.S. ~$60 billion/year as of 2002
- FBI CCS: Security Issues $67 billion/year as of 2005
  - >⅓ from viruses, network intrusion, etc.



Security Vulnerability Example

- Buffer overflows a large class of security vulnerabilities

```
void foo()
{
   int local_variables;
   int buffer[256];
   …
   buffer = read_input();
   …
   return;
}
```

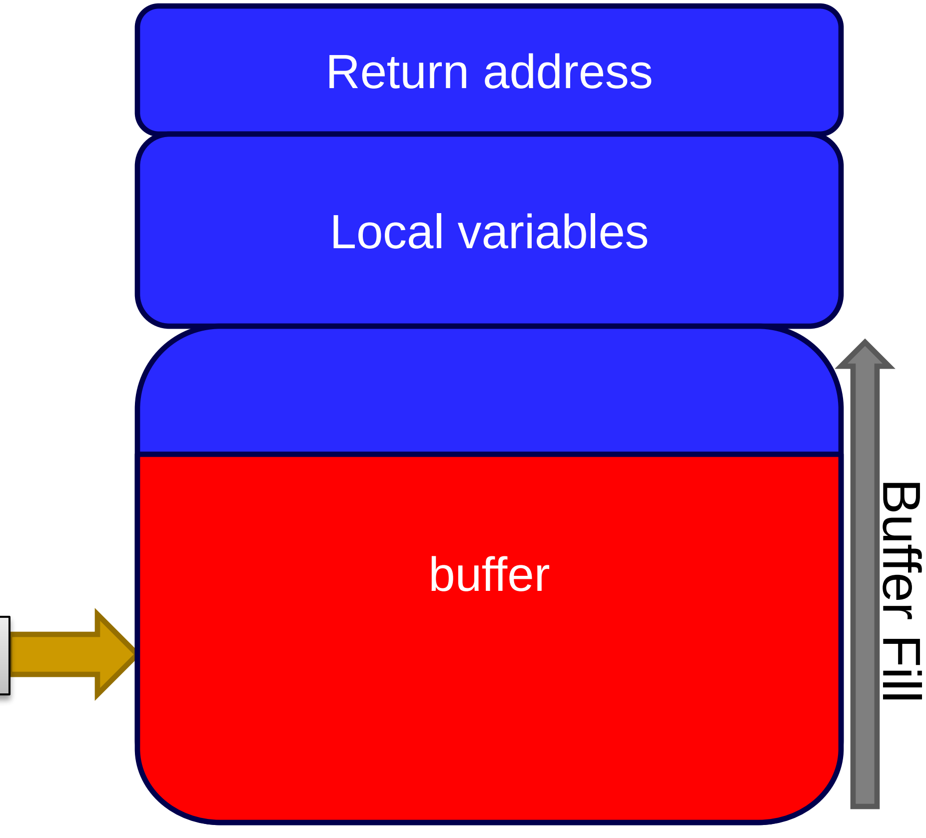


Security Vulnerability Example

- Buffer overflows a large class of security vulnerabilities

```
void foo()
{
   int local_variables;
   int buffer[256];
   …
   buffer = read_input();
   …
   return;
}
```

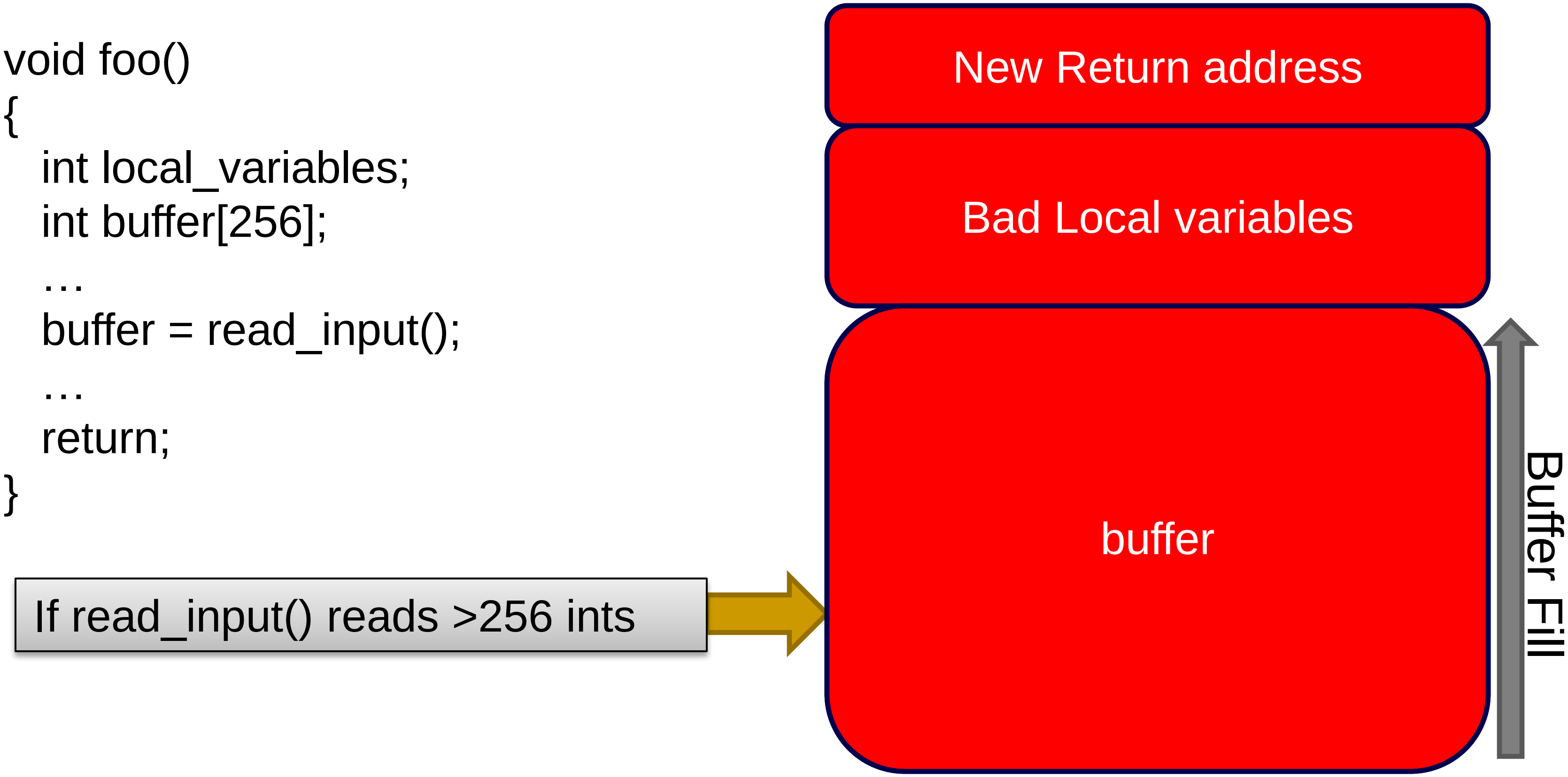
If read_input() reads 200 ints



Security Vulnerability Example

- Buffer overflows a large class of security vulnerabilities

```
void foo()
{
   int local_variables;
   int buffer[256];
   …
   buffer = read_input();
   …
   return;
}
```



Concurrency Bugs Also Matter

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	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

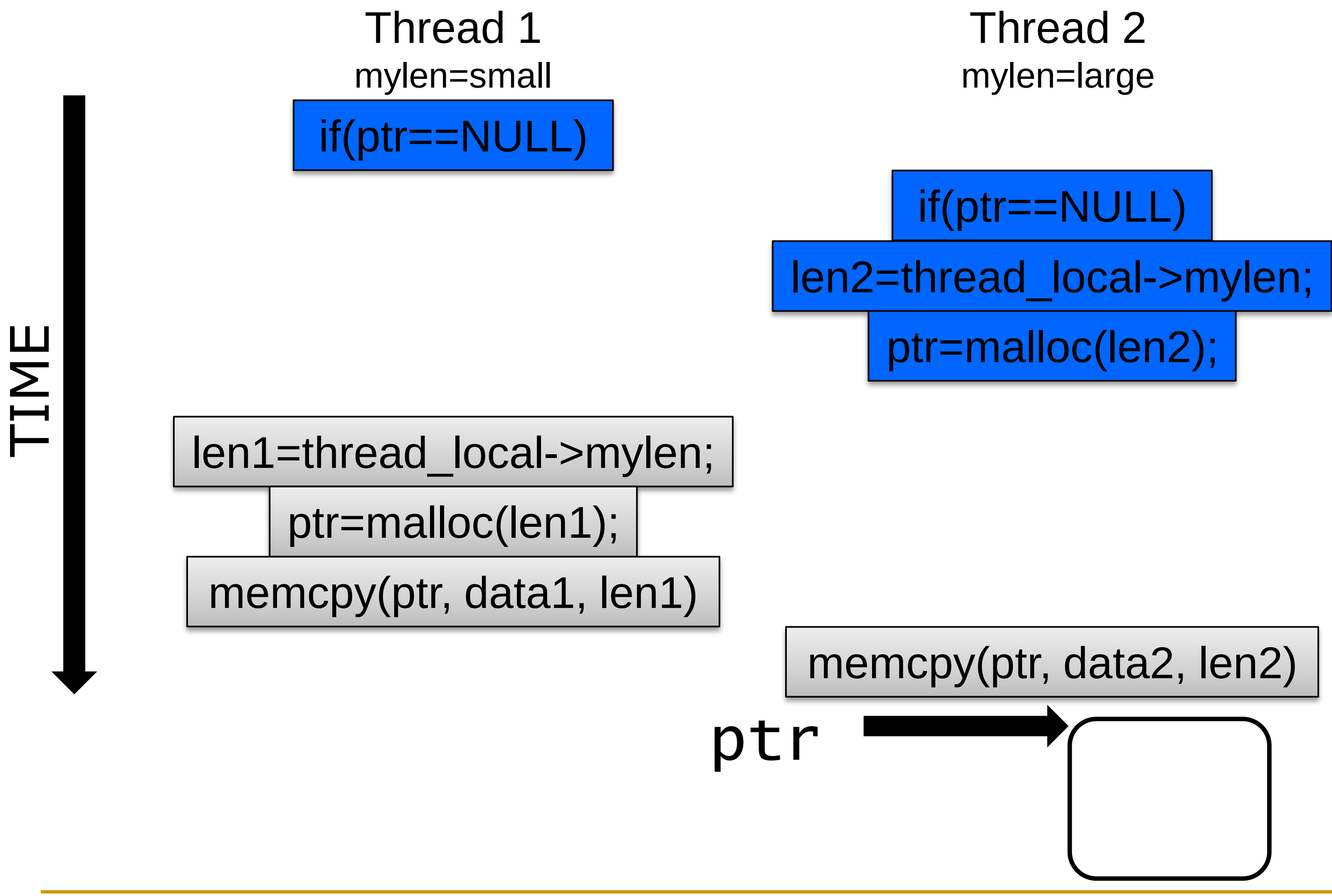
TIME

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)

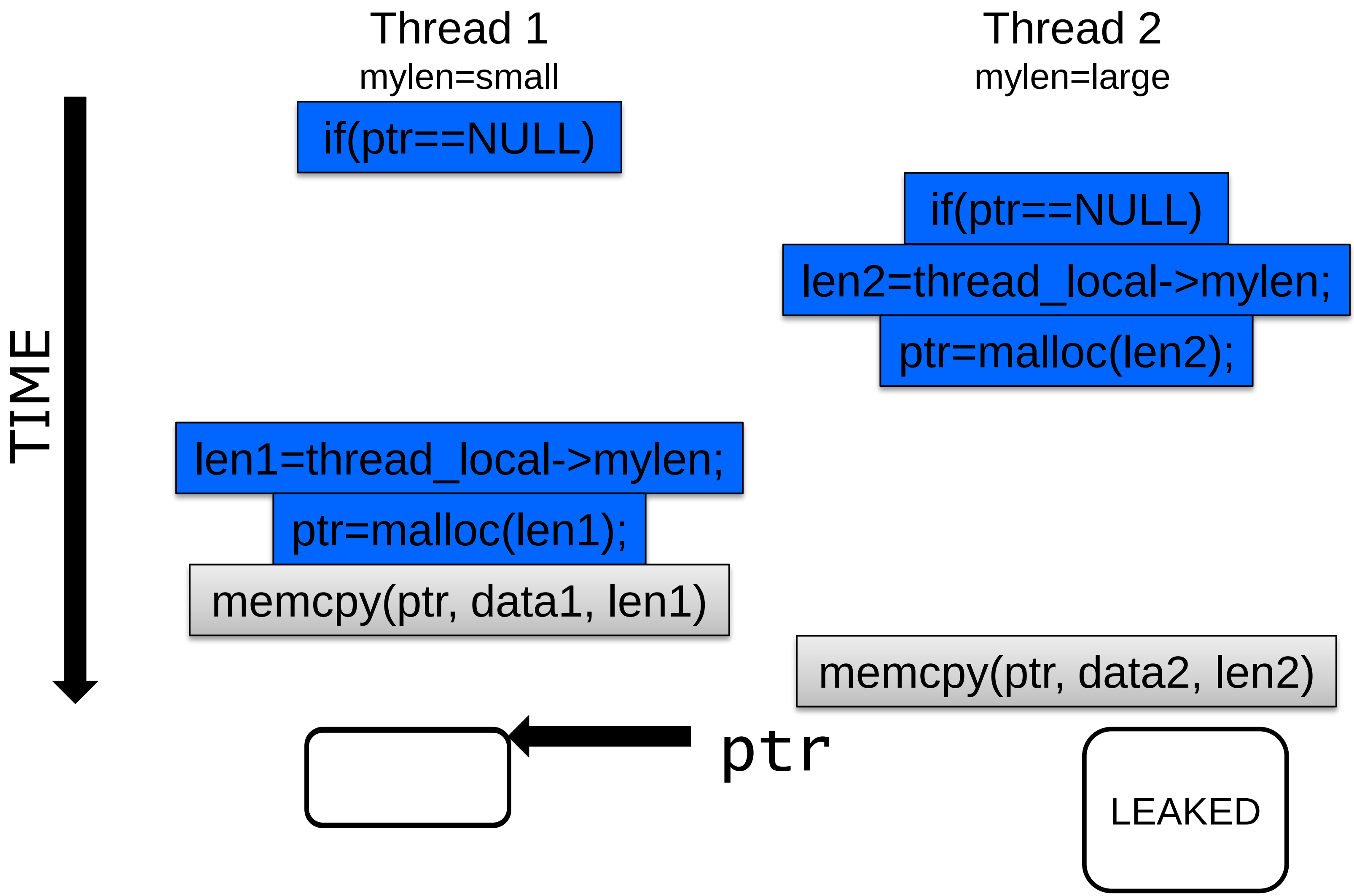
`ptr`

∅

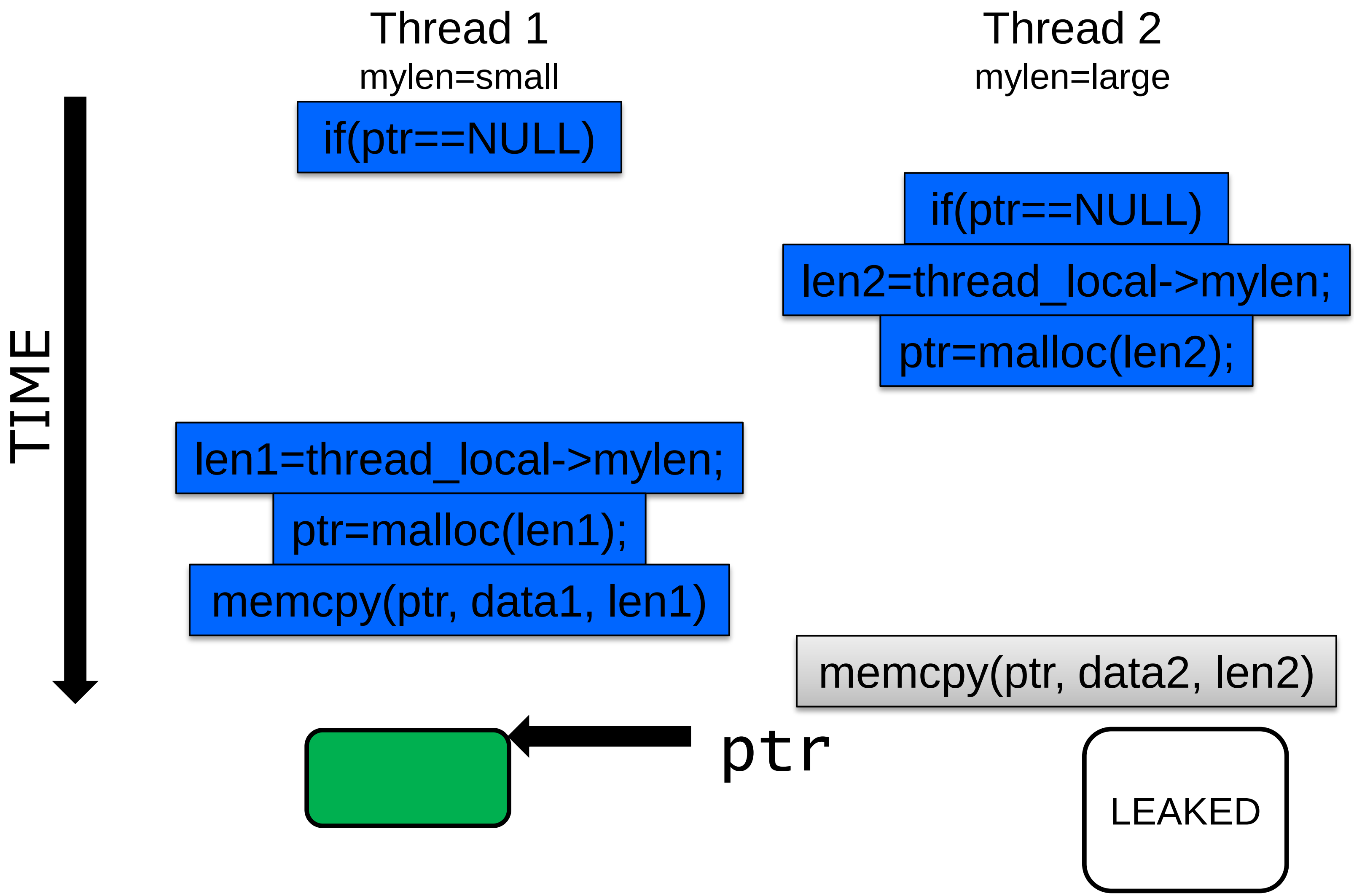
Concurrency Bugs Matter NOW



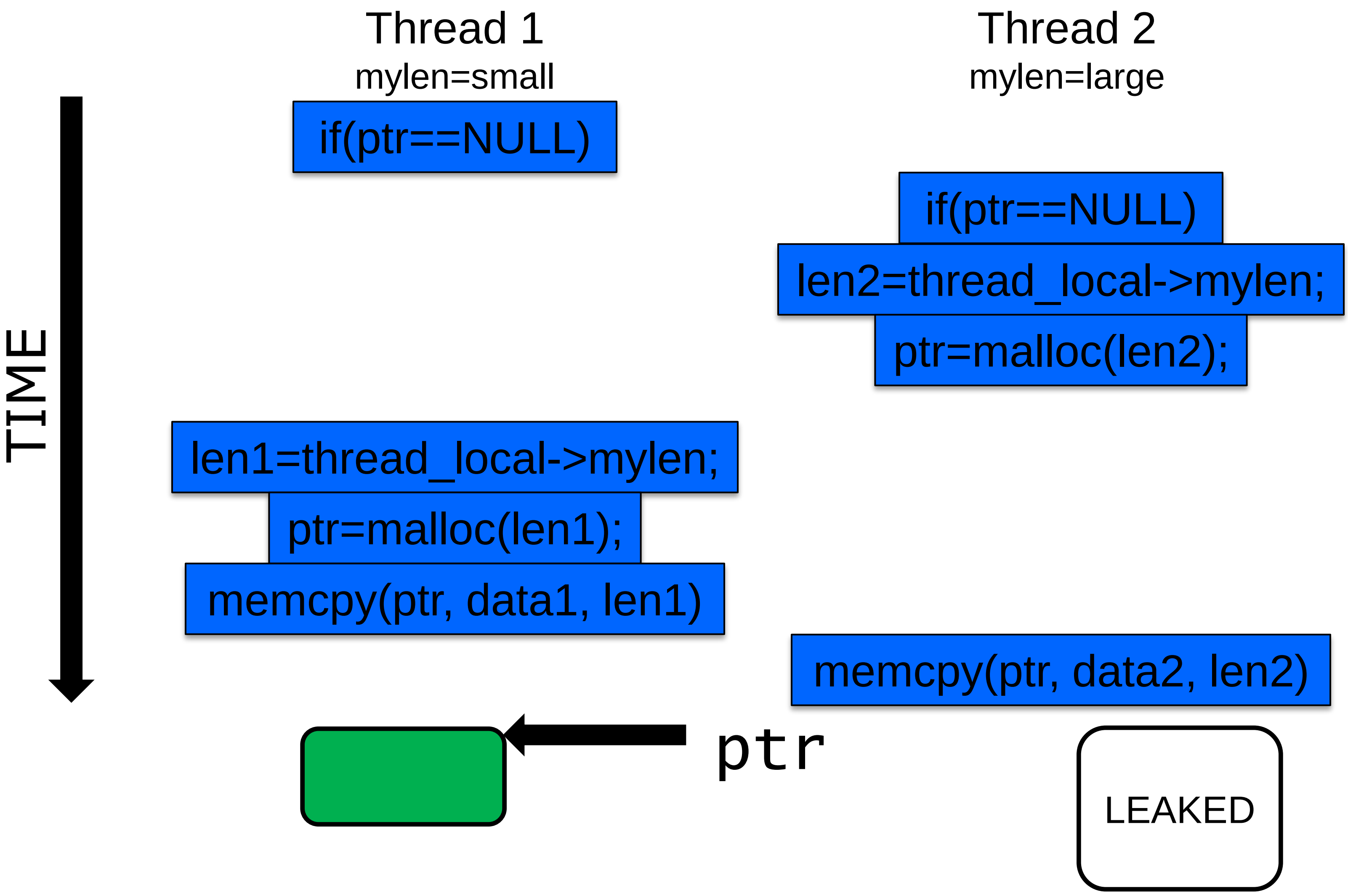
Concurrency Bugs Matter NOW



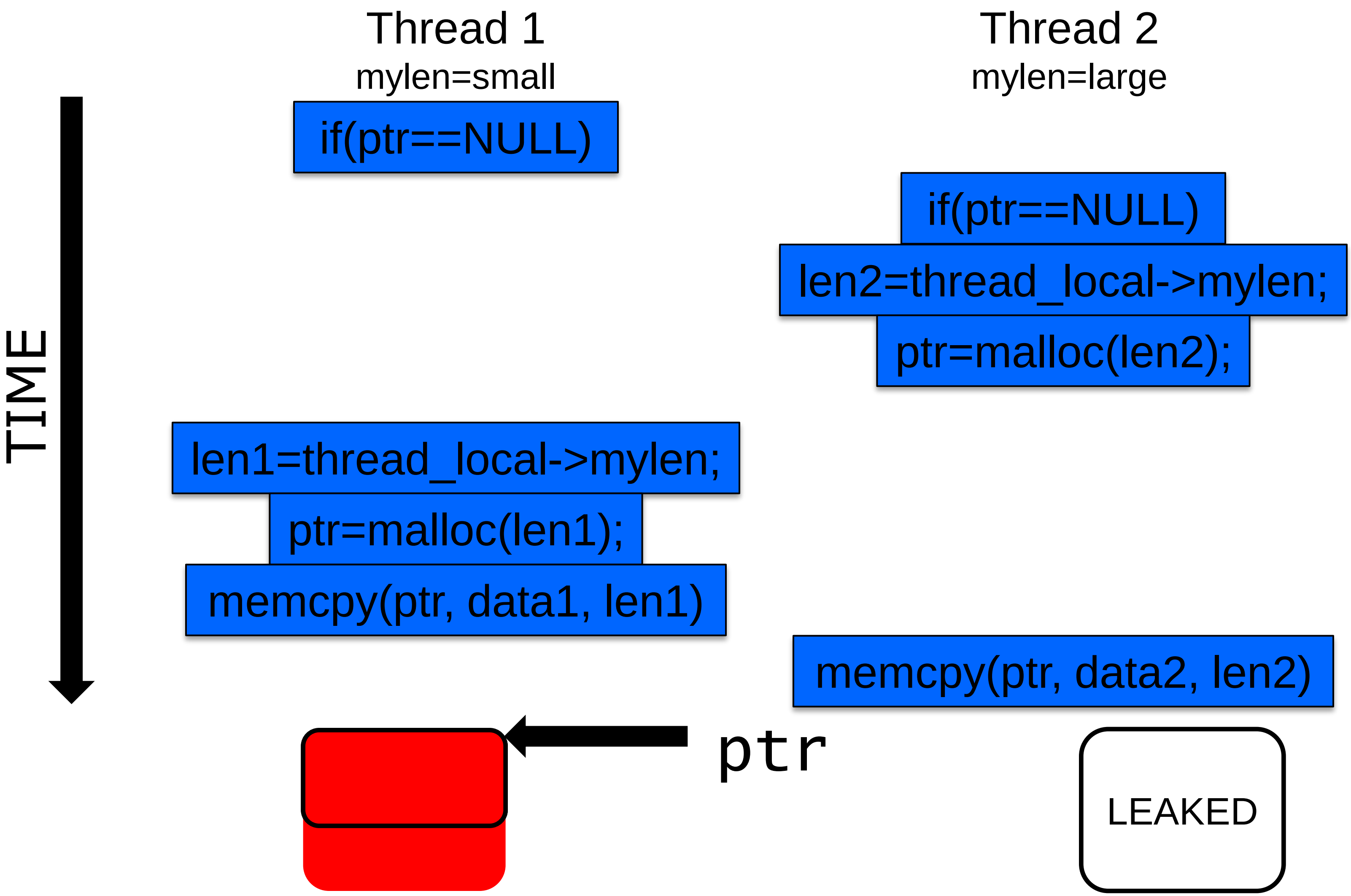
Concurrency Bugs Matter NOW



Concurrency Bugs Matter NOW



Concurrency Bugs Matter NOW



One Layer of a Solution

- High quality dynamic software analysis
  - Find **difficult bugs** that other analyses miss
- **Distribute Tests** to Large Populations
  - Low overhead or users get angry
- Accomplished by **sampling the analyses**
  - Each user only tests part of the program

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

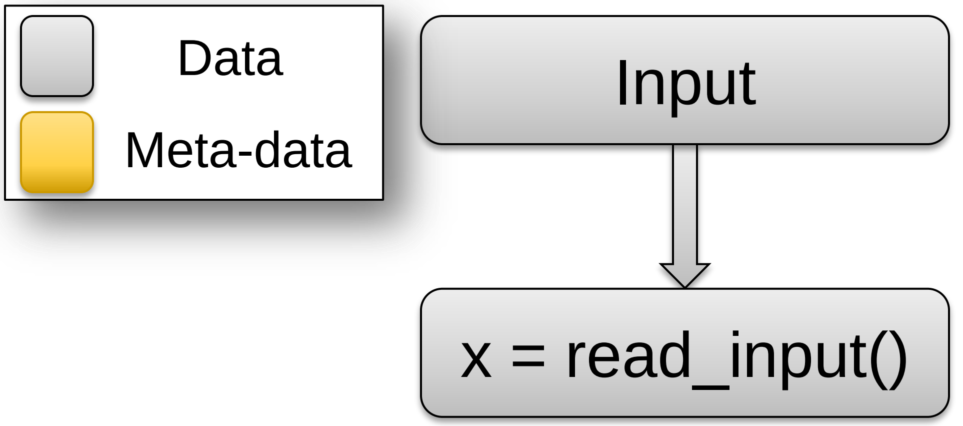
Example Dynamic Dataflow Analysis

Data

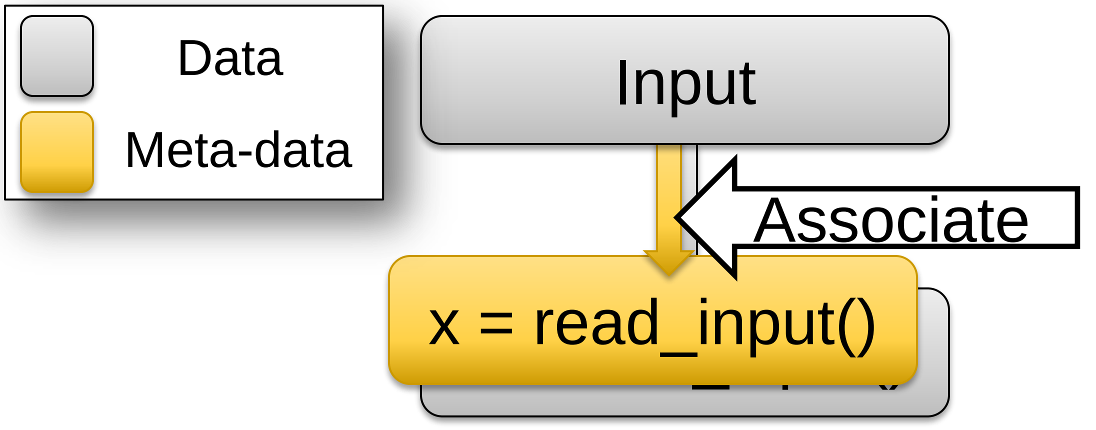
Meta-data

Input

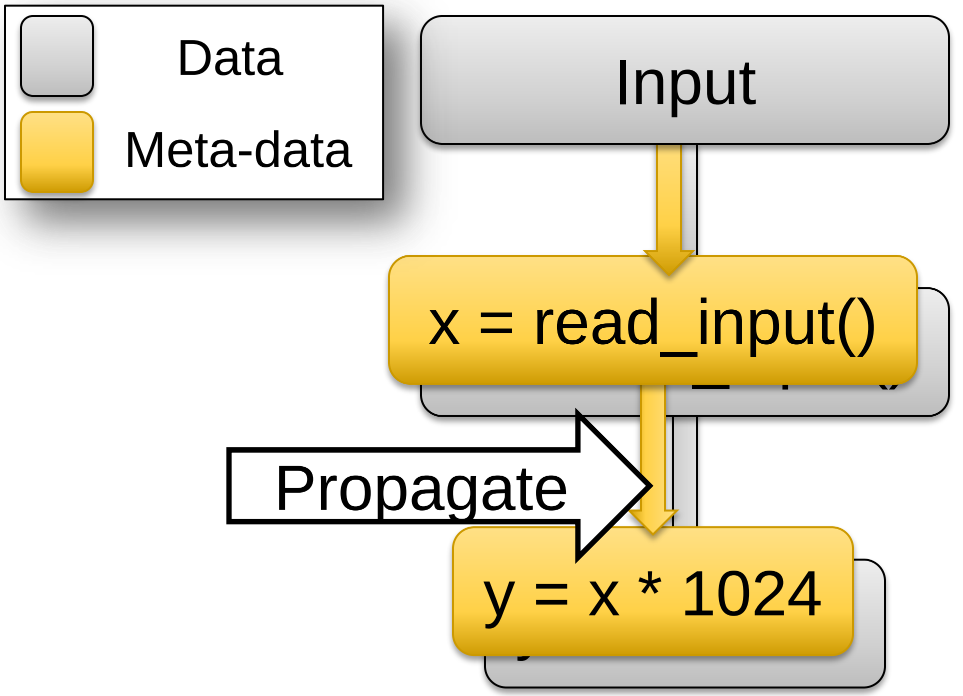
Example Dynamic Dataflow Analysis



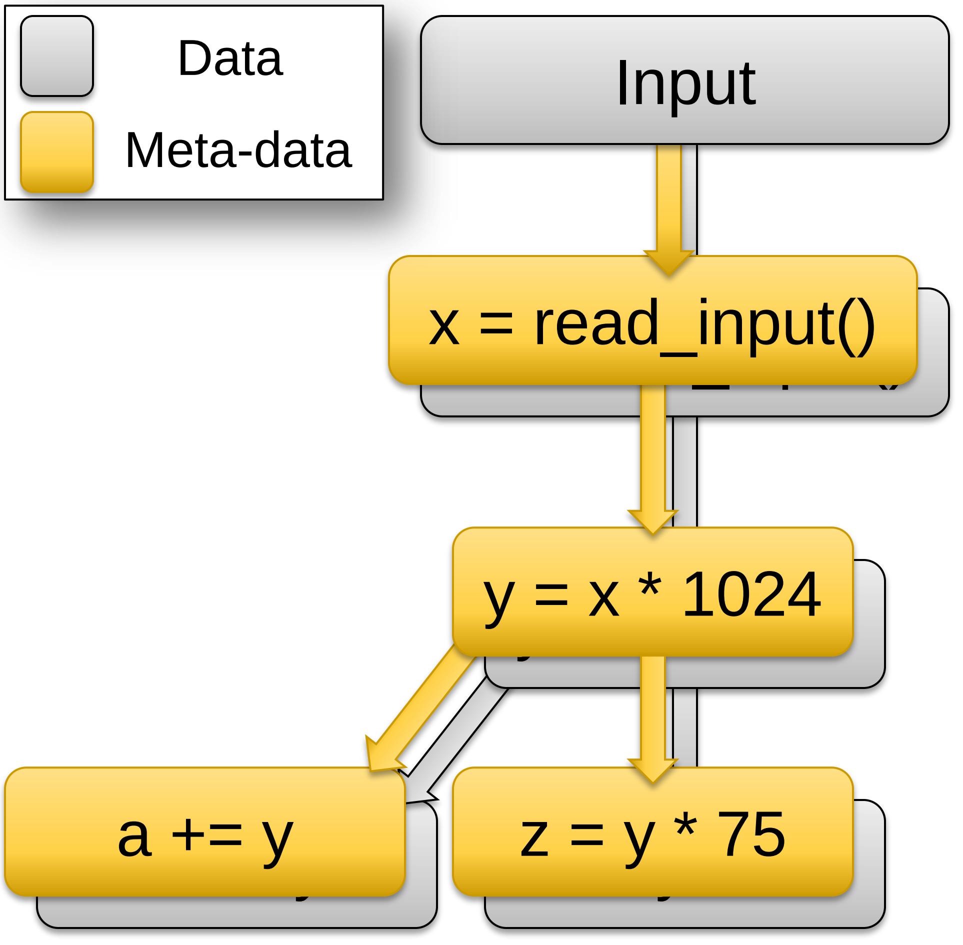
Example Dynamic Dataflow Analysis



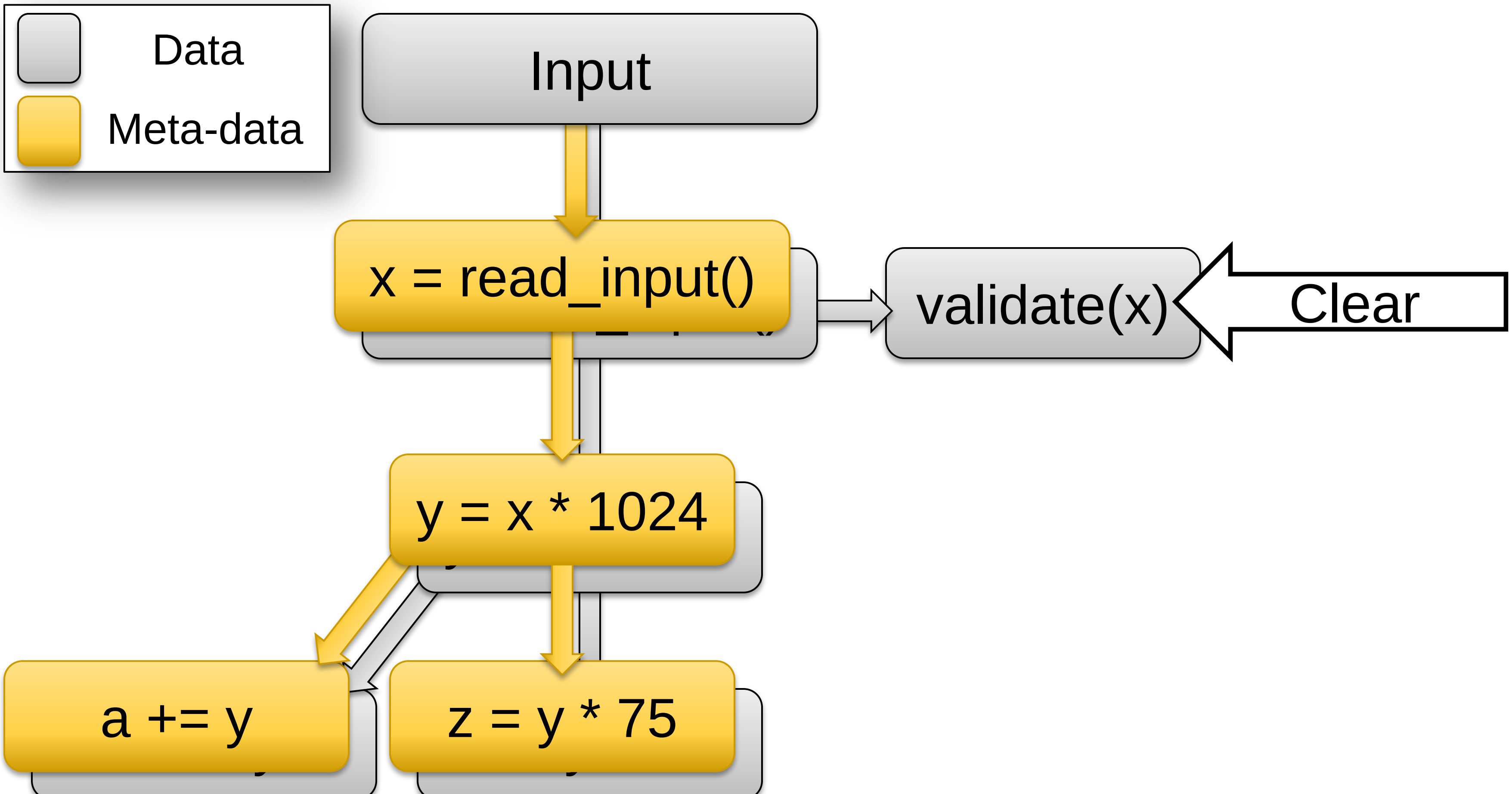
Example Dynamic Dataflow Analysis



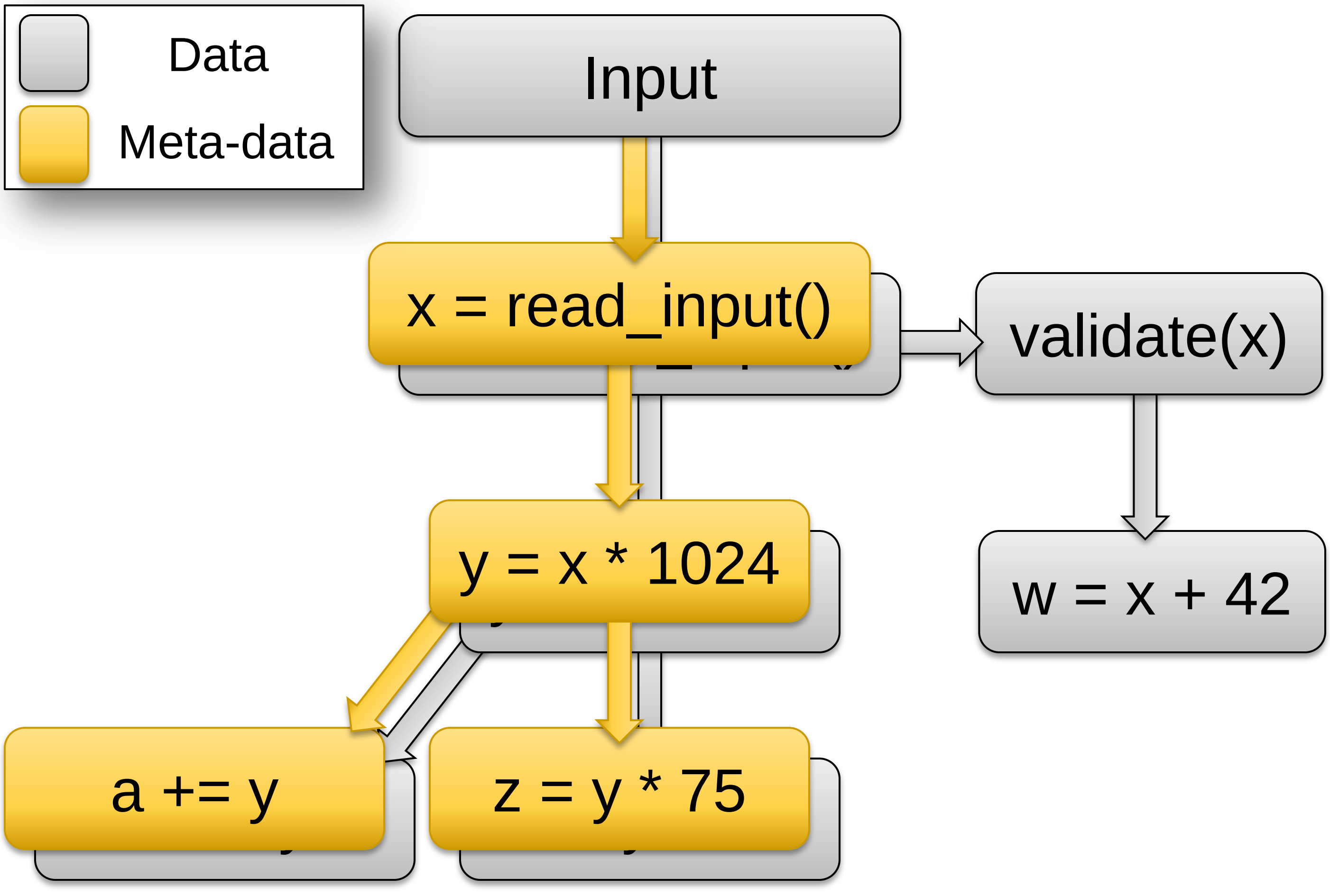
Example Dynamic Dataflow Analysis



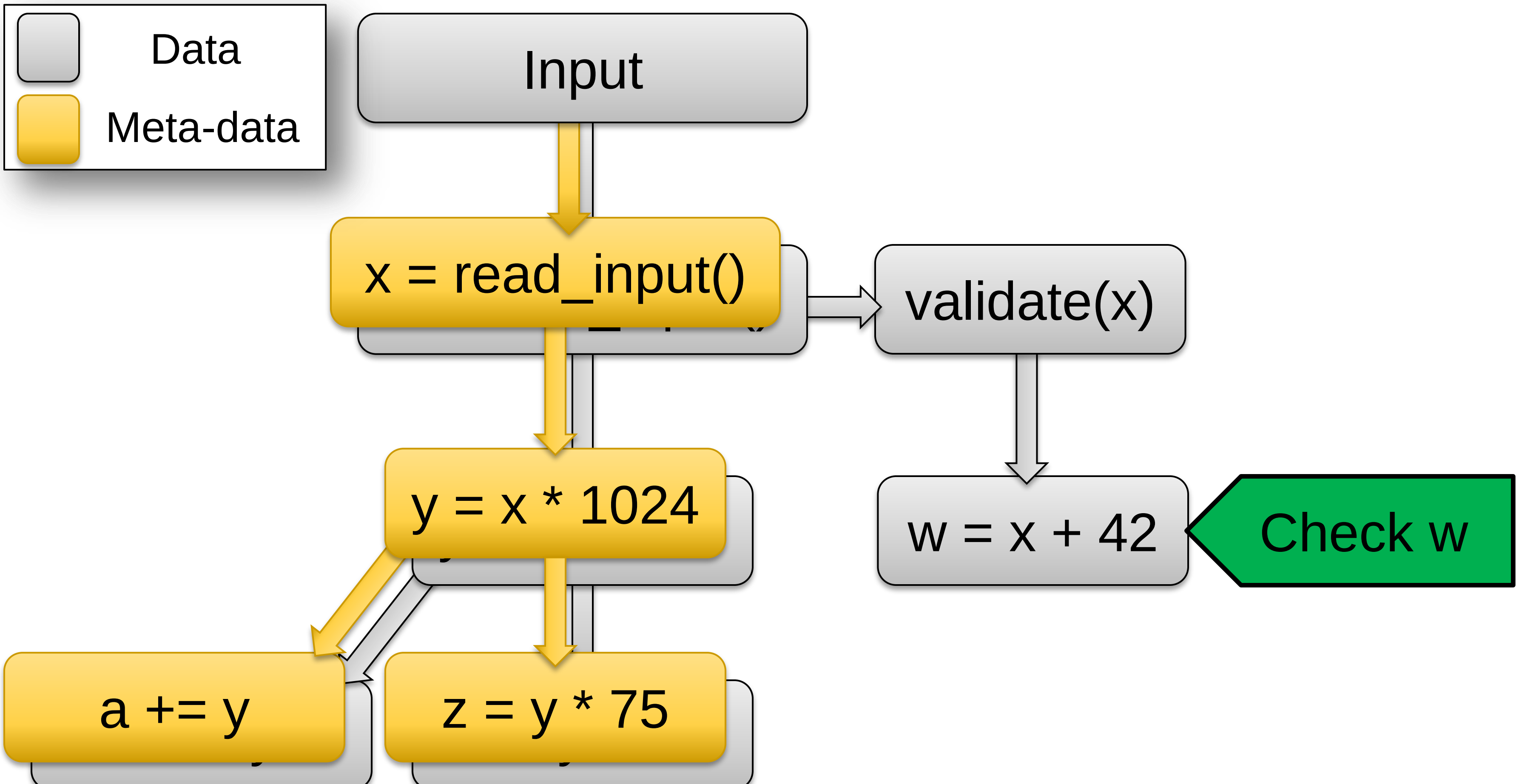
Example Dynamic Dataflow Analysis



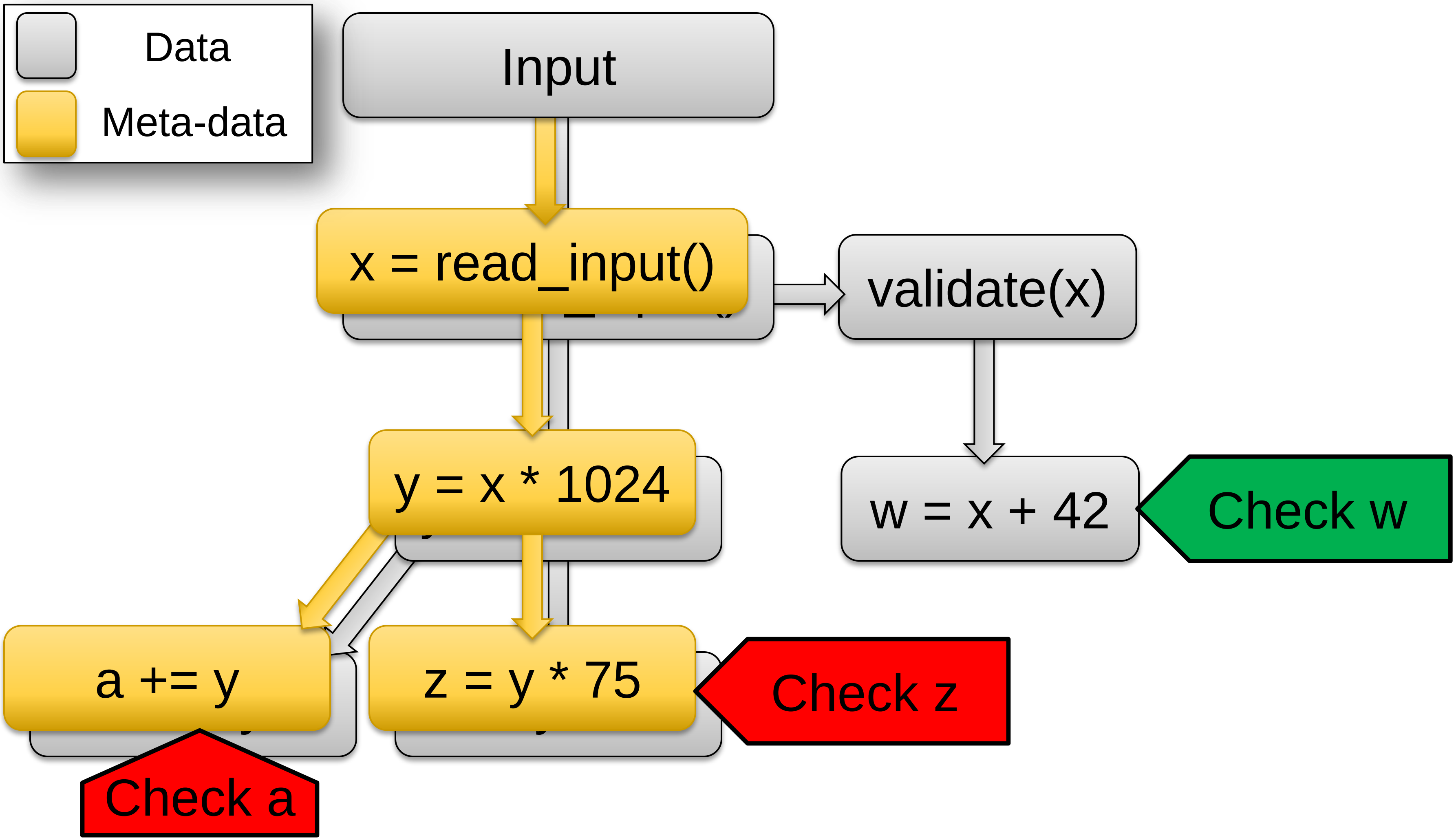
Example Dynamic Dataflow Analysis



Example Dynamic Dataflow Analysis

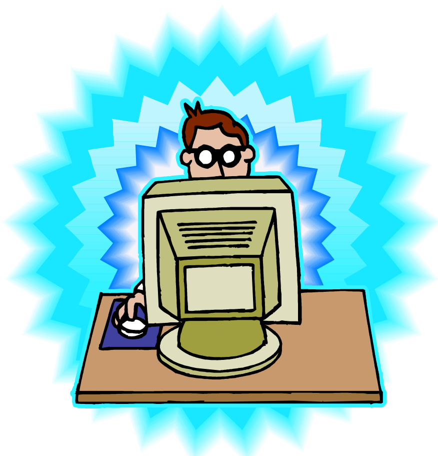


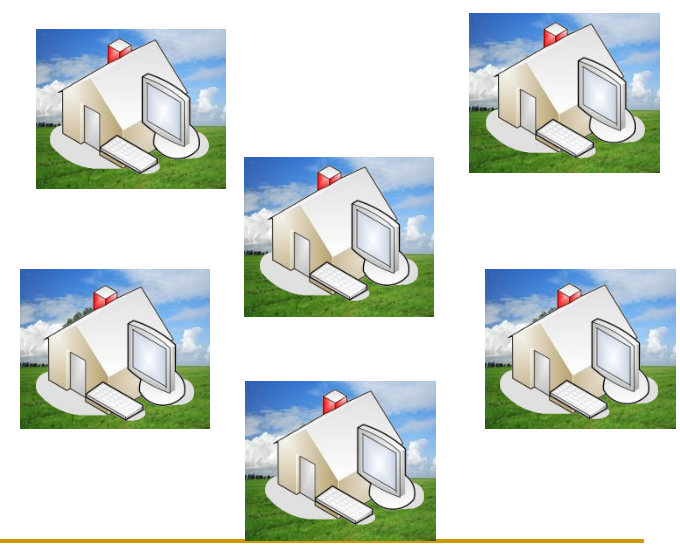
Example Dynamic Dataflow Analysis



Distributed Dynamic Dataflow Analysis

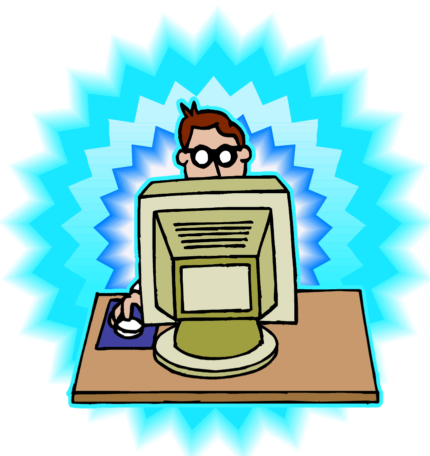
- Split analysis across large populations
  - Observe more runtime states
  - Report problems developer never thought to test



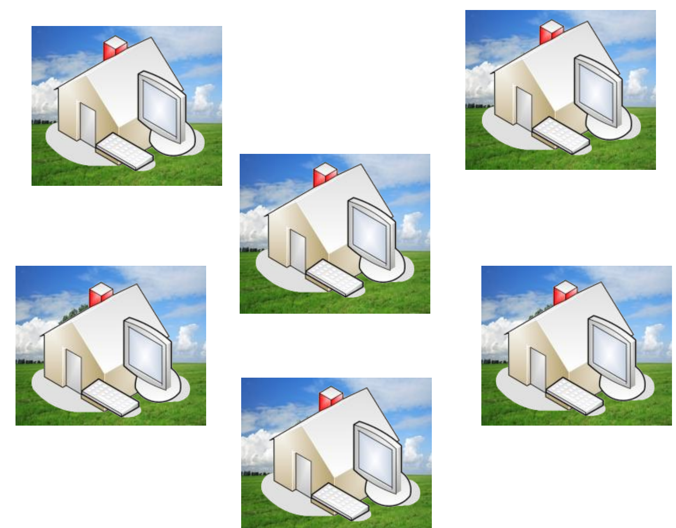


Distributed Dynamic Dataflow Analysis

- Split analysis across large populations
  - Observe more runtime states
  - Report problems developer never thought to test

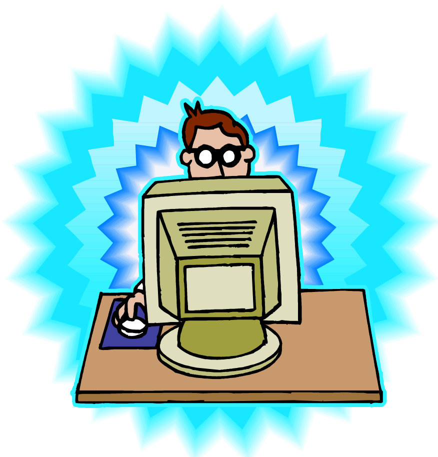


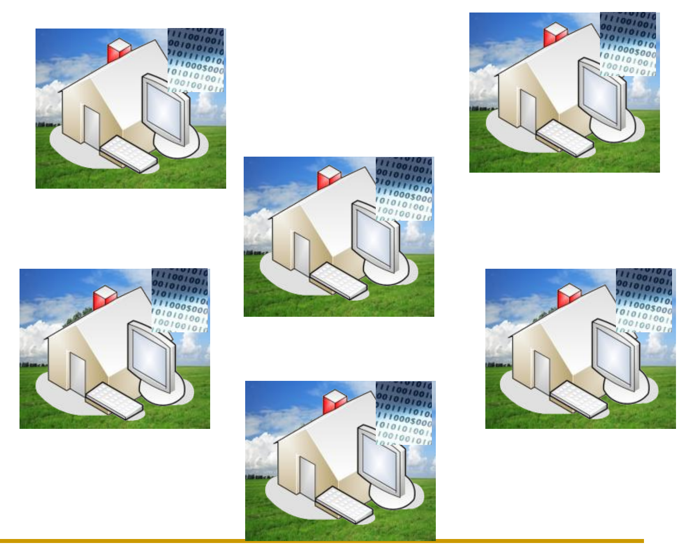
Instrumented Program



Distributed Dynamic Dataflow Analysis

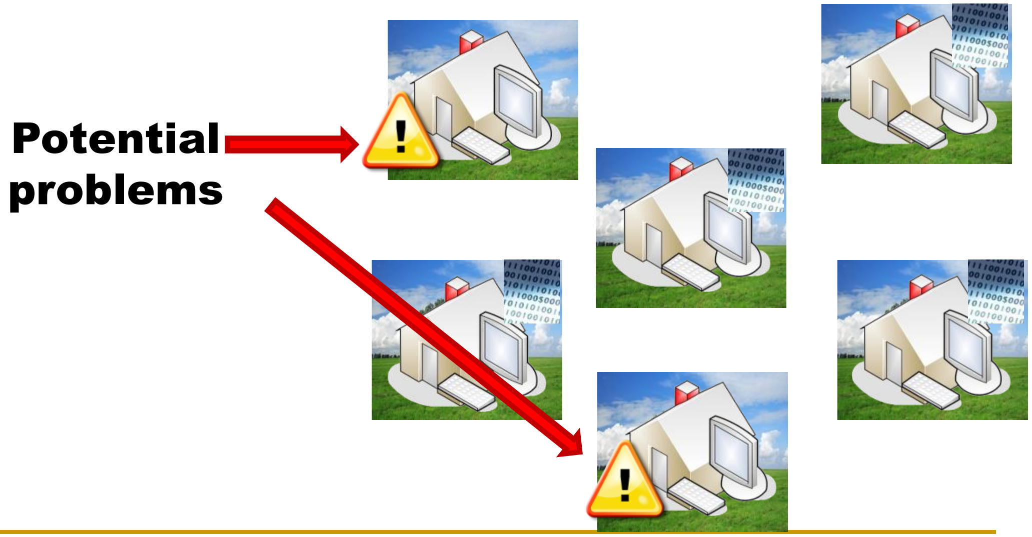
- Split analysis across large populations
  - Observe more runtime states
  - Report problems developer never thought to test





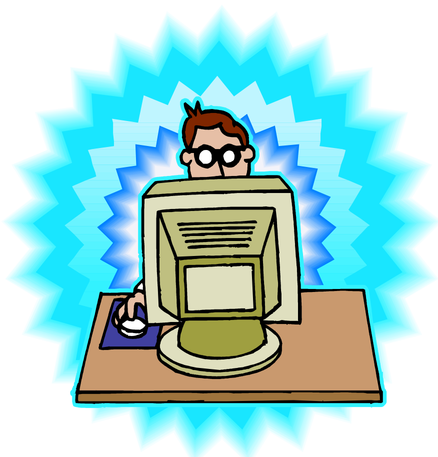
Distributed Dynamic Dataflow Analysis

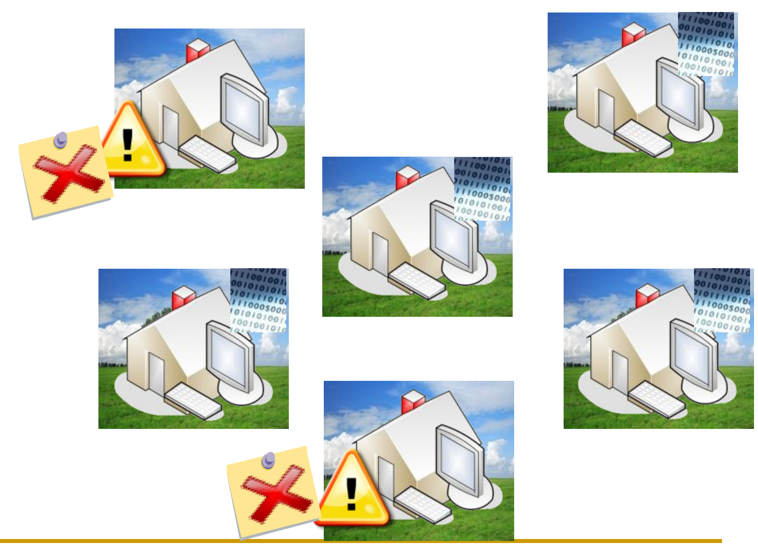
- Split analysis across large populations
  - Observe more runtime states
  - Report problems developer never thought to test



Distributed Dynamic Dataflow Analysis

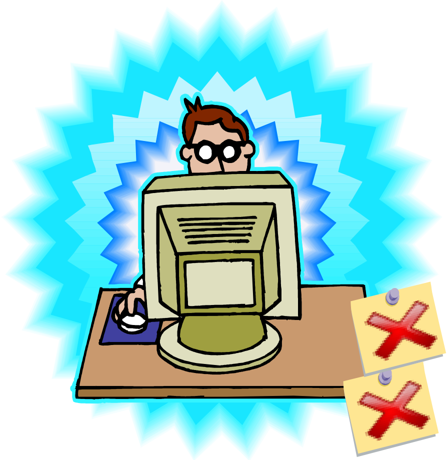
- Split analysis across large populations
  - Observe more runtime states
  - Report problems developer never thought to test

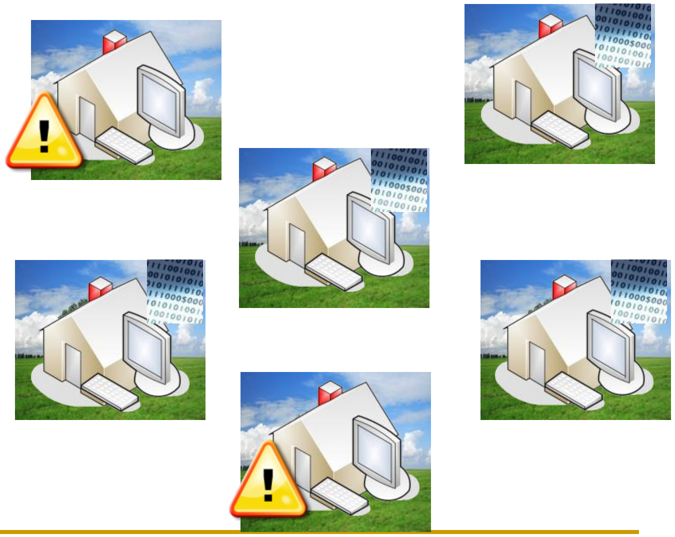




Distributed Dynamic Dataflow Analysis

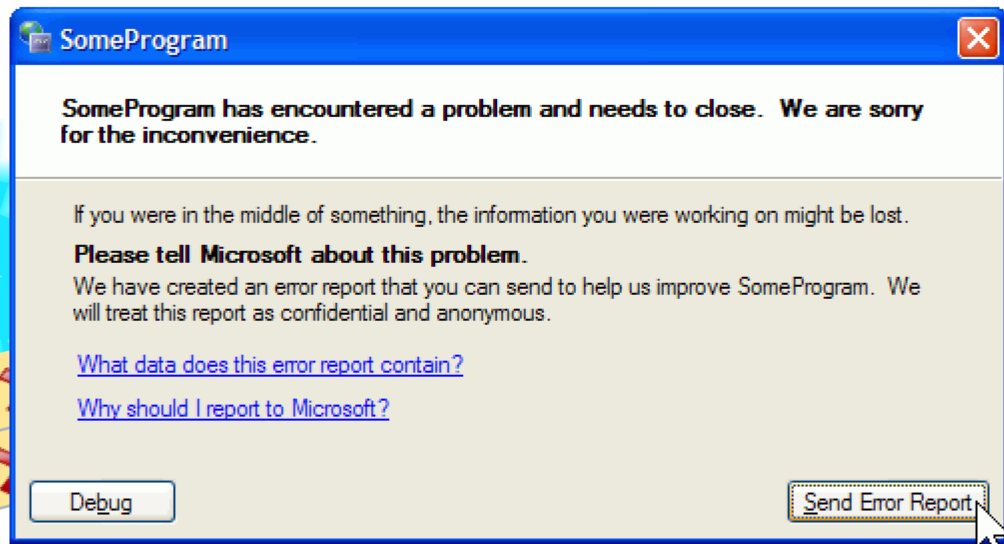
- Split analysis across large populations
  - Observe more runtime states
  - Report problems developer never thought to test

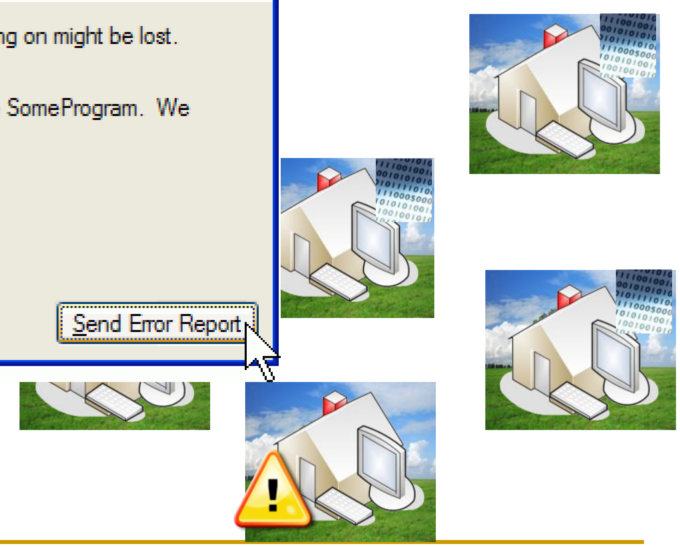




Distributed Dynamic Dataflow Analysis

- Split analysis across large populations
  - Observe more runtime states
  - Report problems developer never thought to test



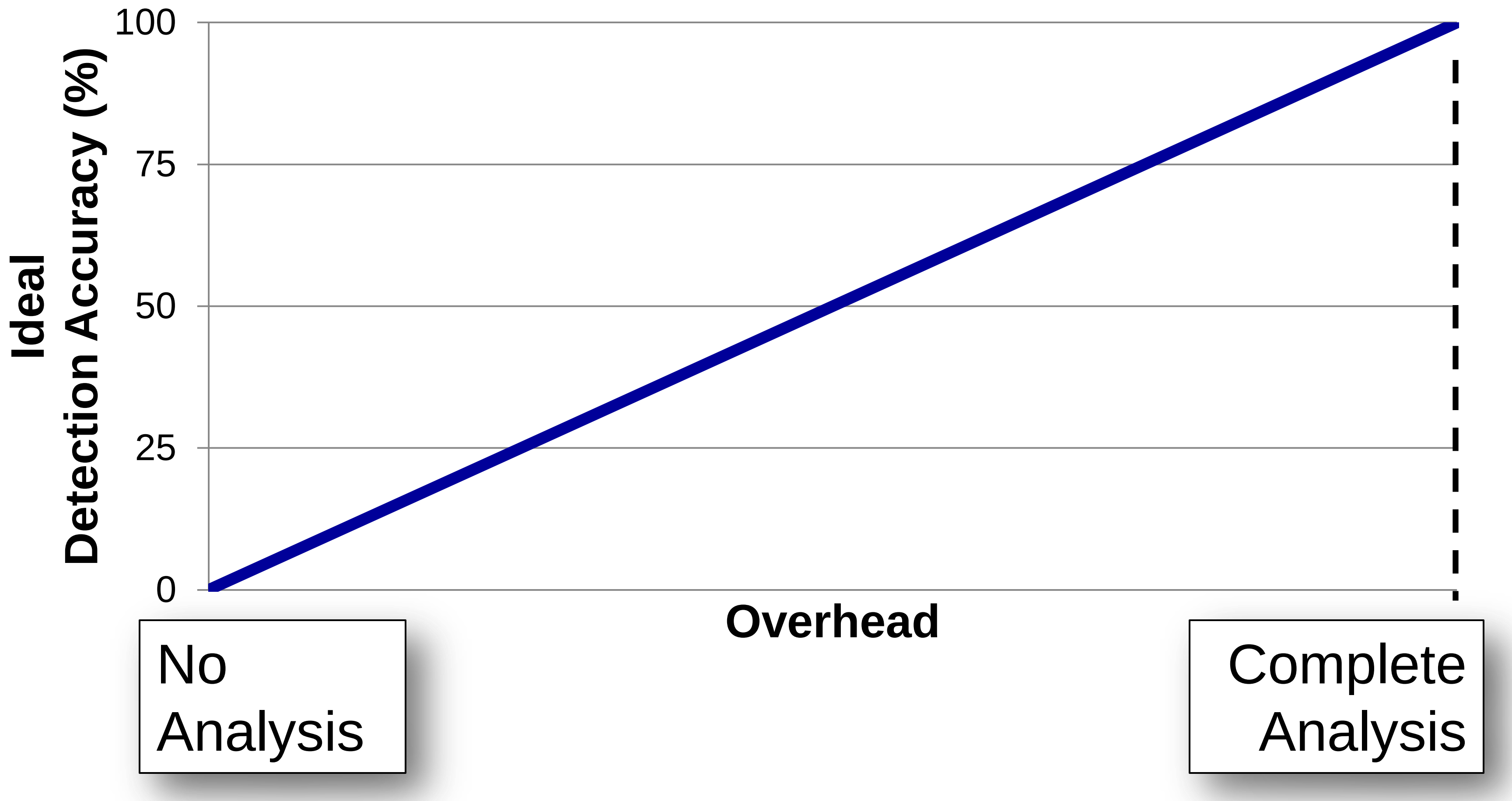


Problem: DDAs are Slow

- Symbolic Execution
  10-200x
- Data Race Detection
  (e.g. Helgrind)
  2-300x
- Memory Checking
  (e.g. Dr. Memory)
  5-50x
- Taint Analysis
  (e.g.TaintCheck)
  2-200x
- Dynamic Bounds Checking
  10-80x
- FP Accuracy Verification
  100-500x

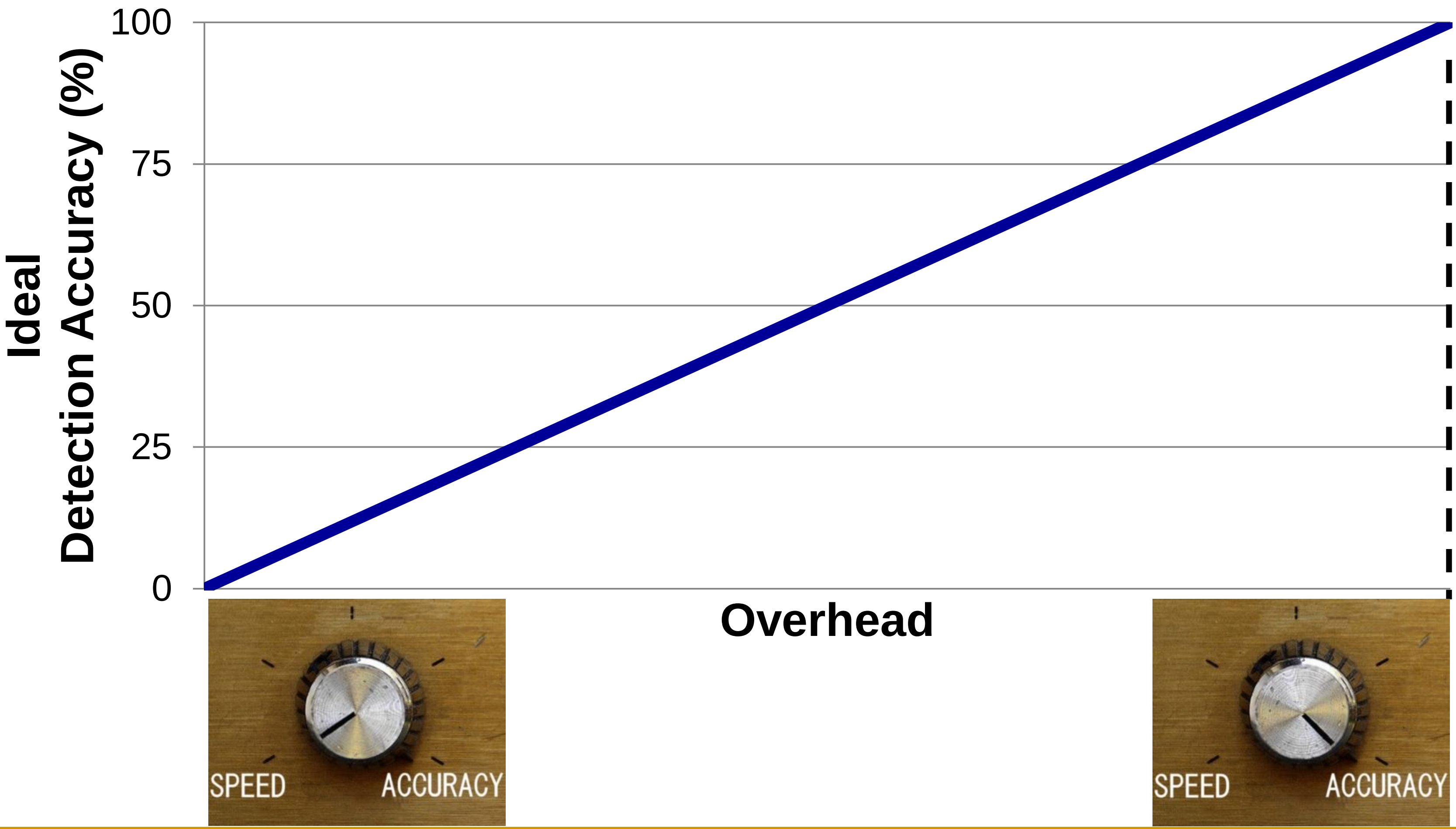
Our Solution: Sampling

- Lower overheads by skipping some analyses

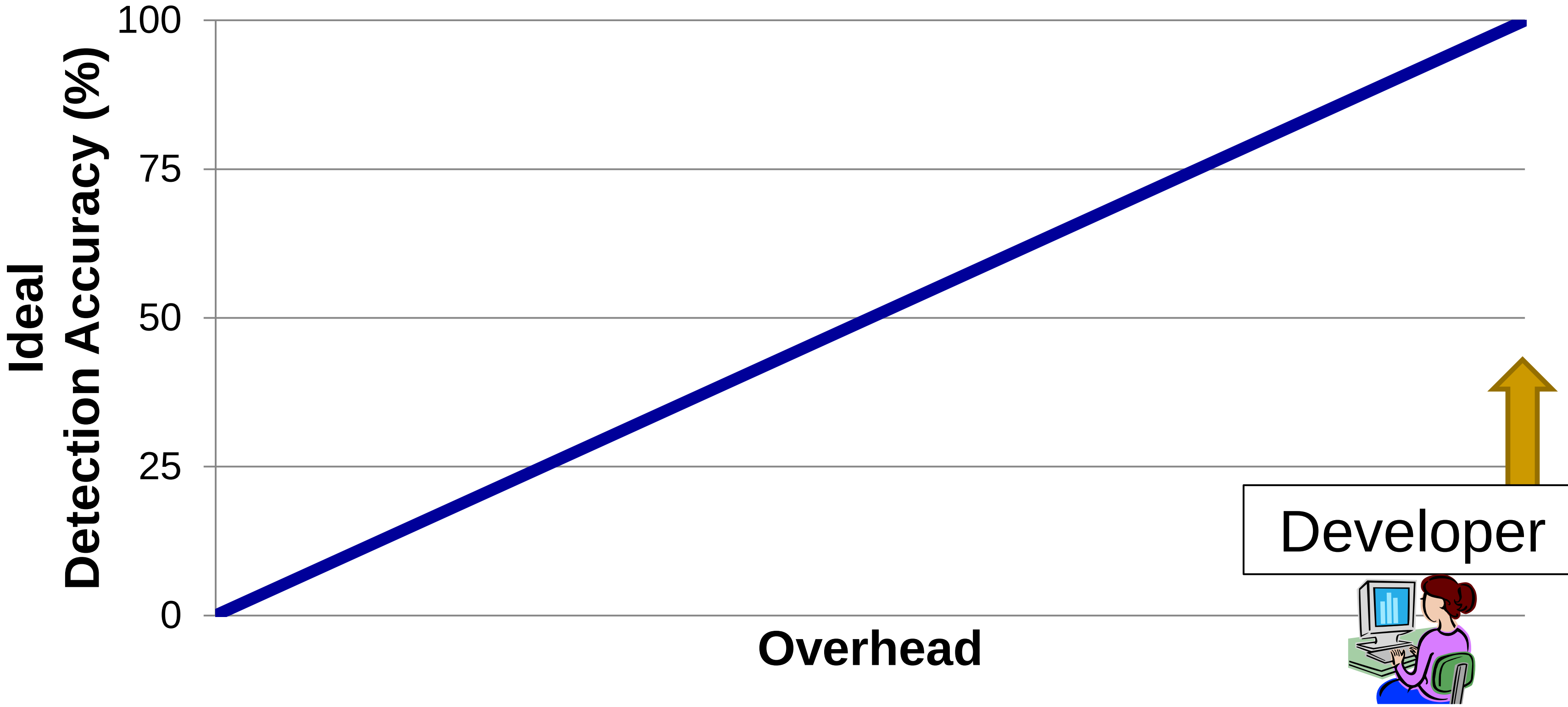


Our Solution: Sampling

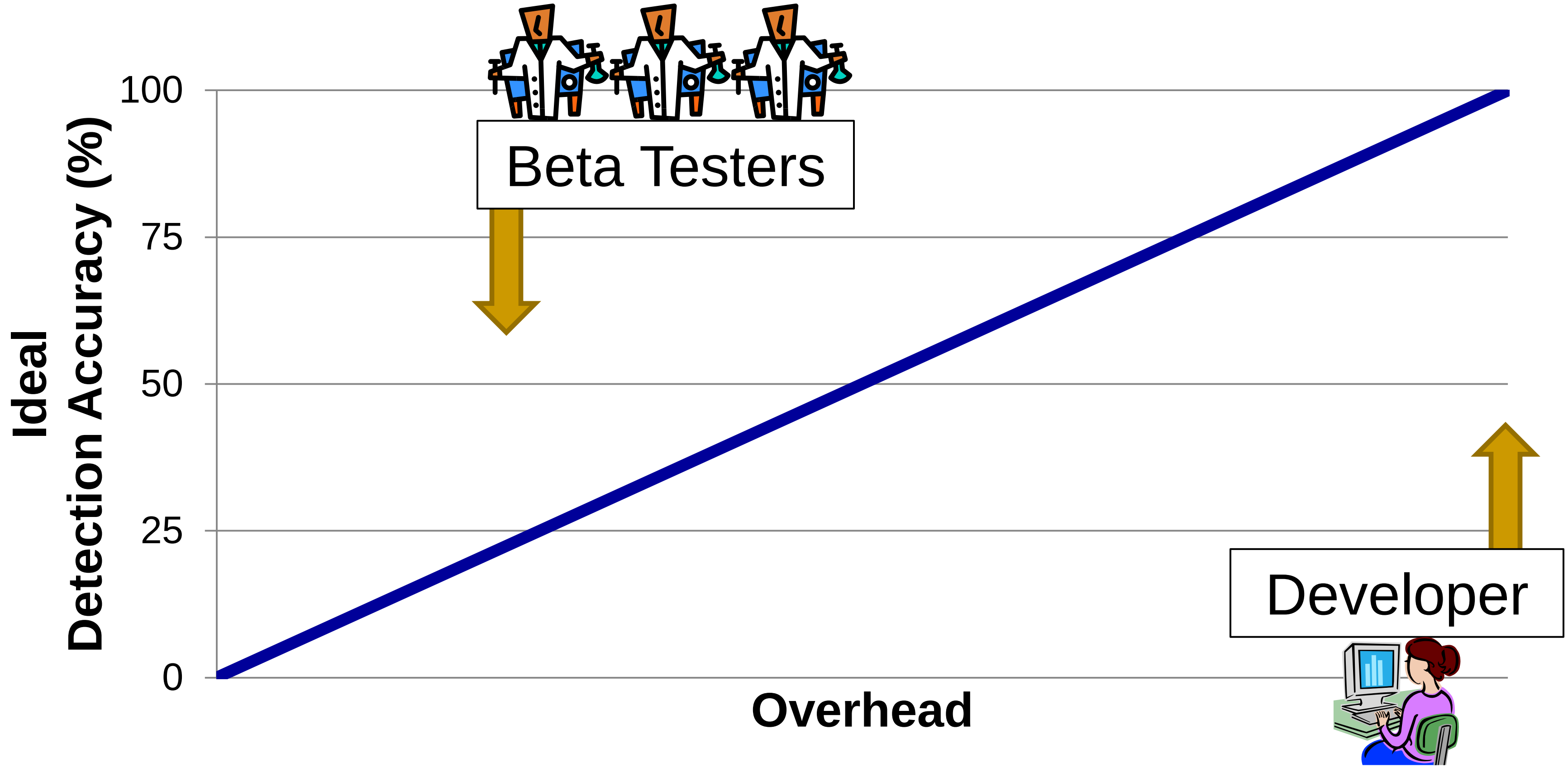
- Lower overheads by skipping some analyses



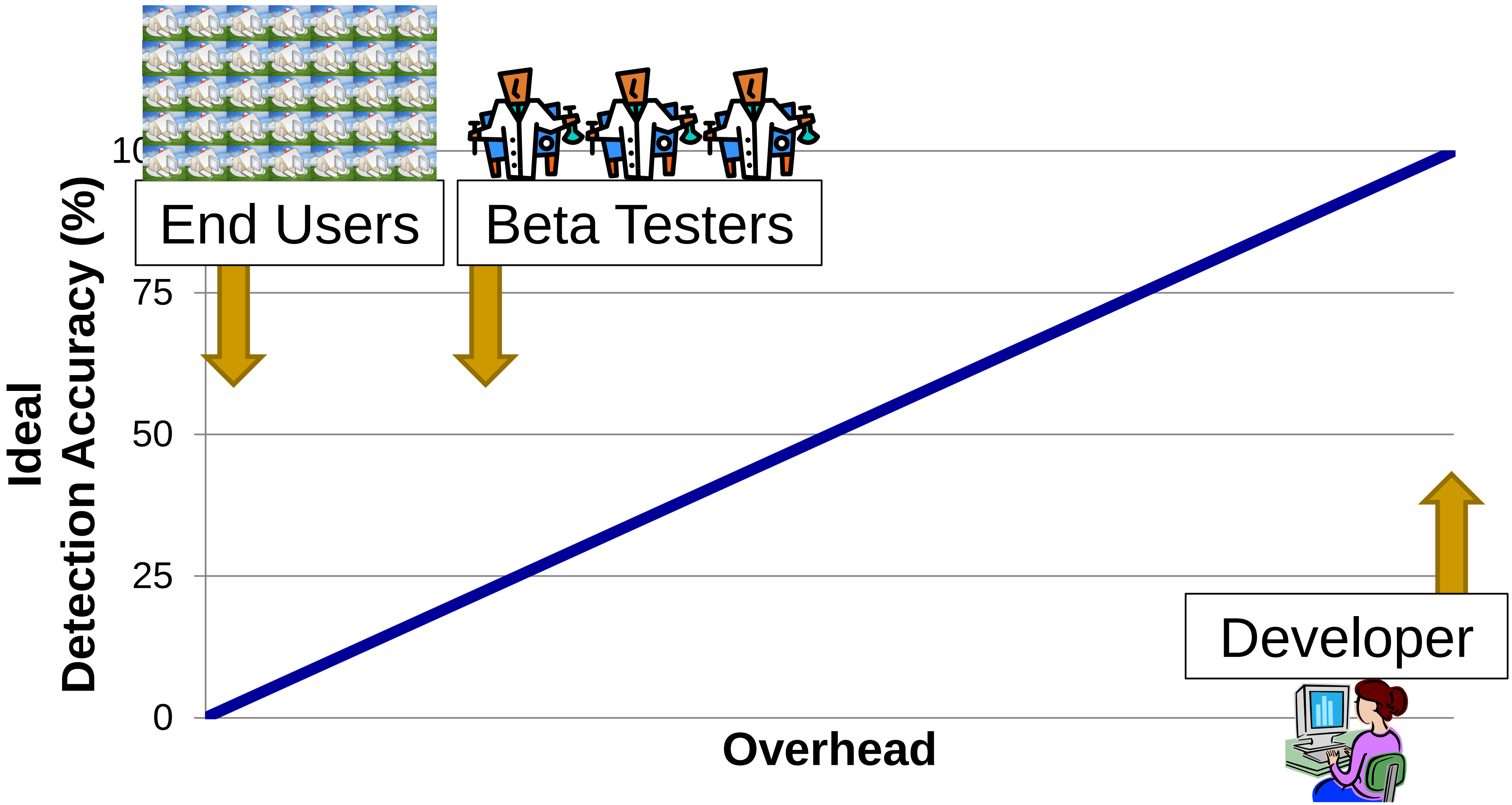
Sampling Allows Distribution



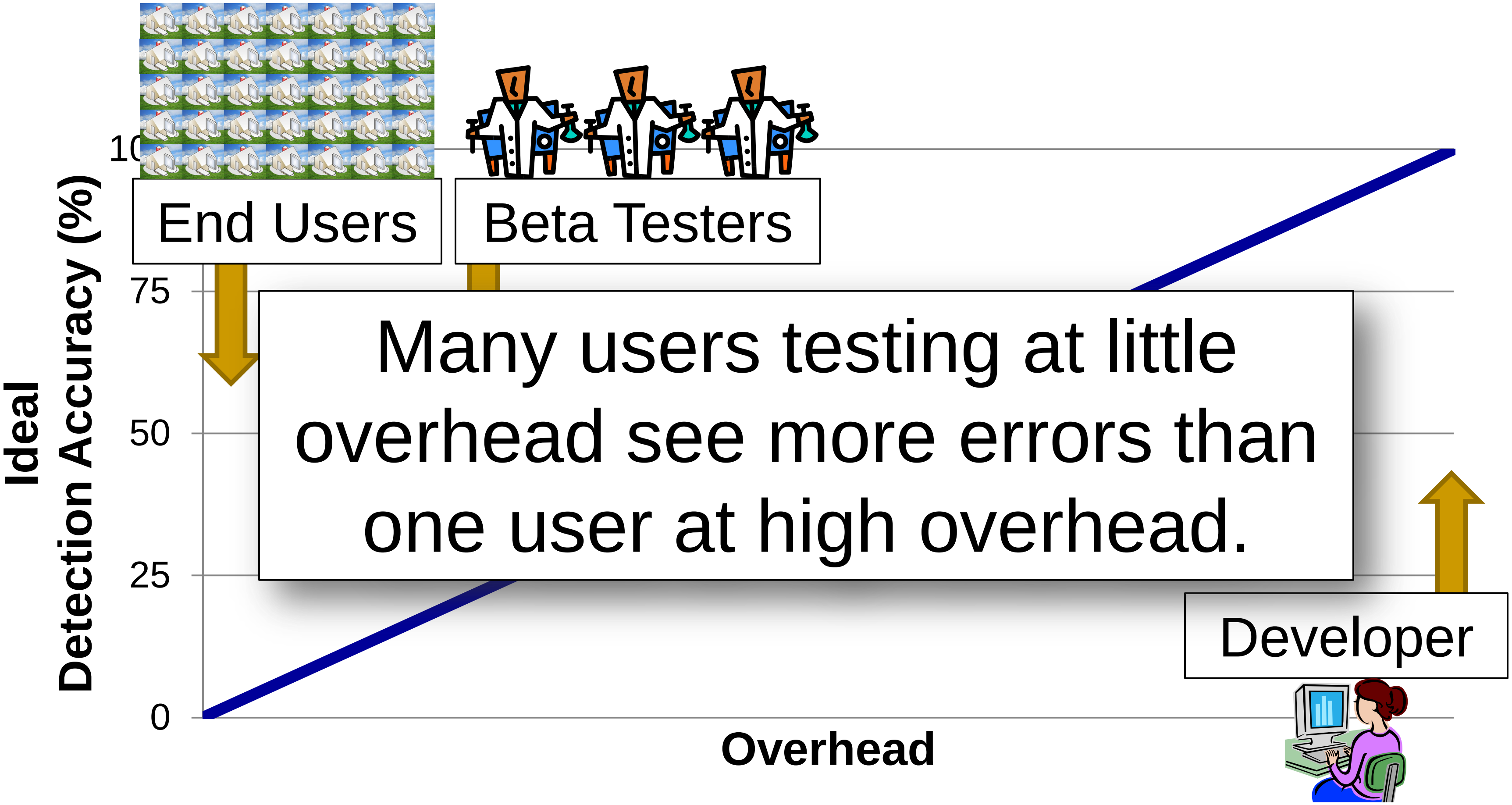
Sampling Allows Distribution



Sampling Allows Distribution



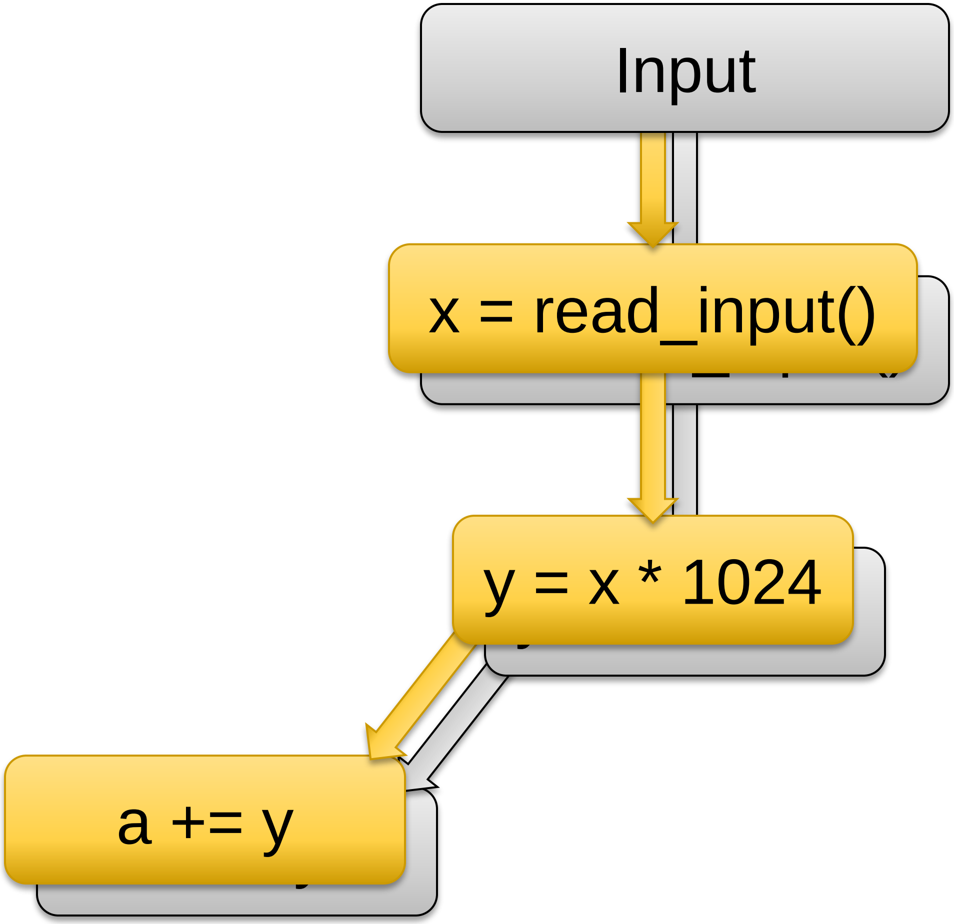
Sampling Allows Distribution



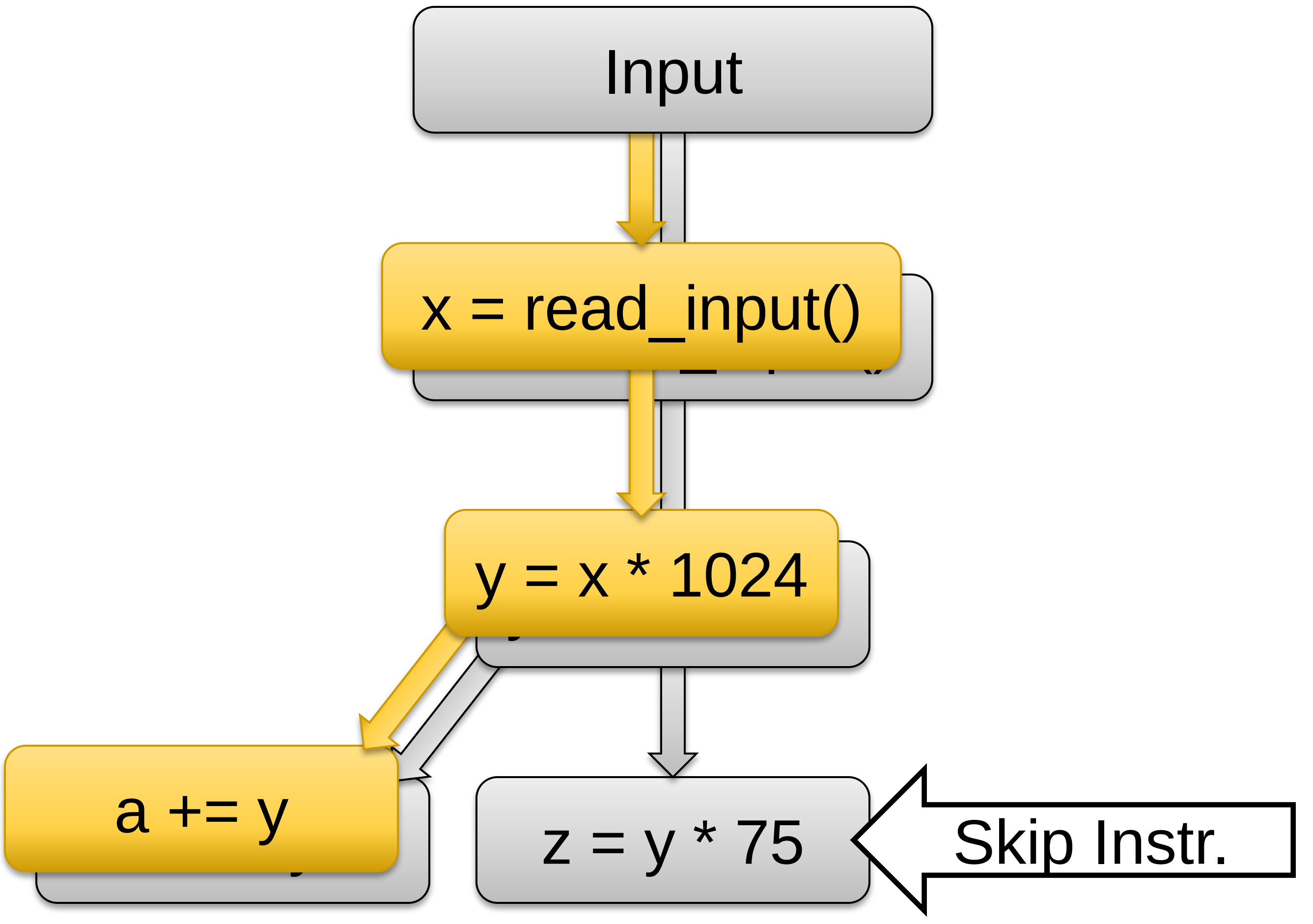
Cannot Naïvely Sample Code

Input

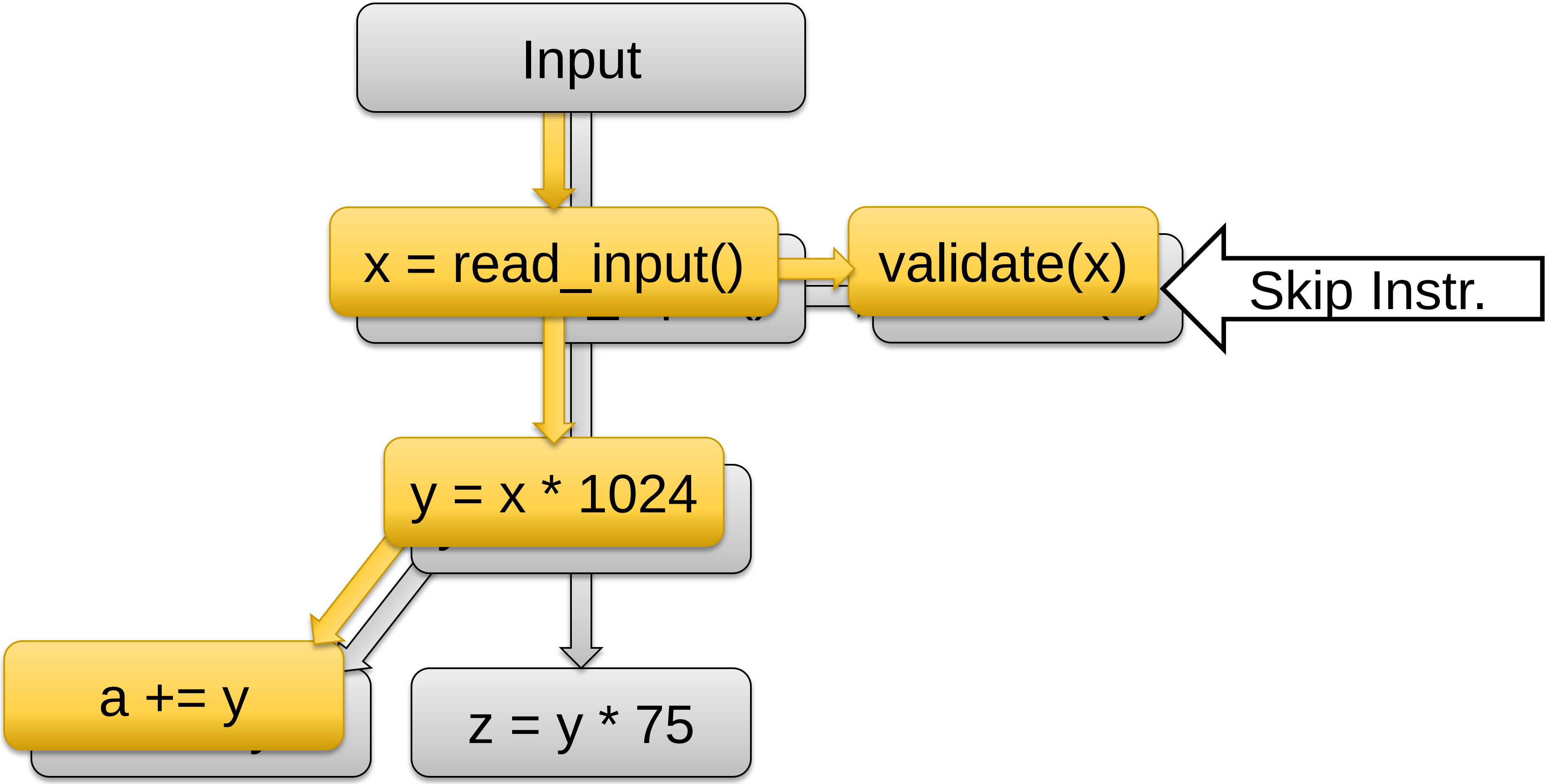
Cannot Naïvely Sample Code



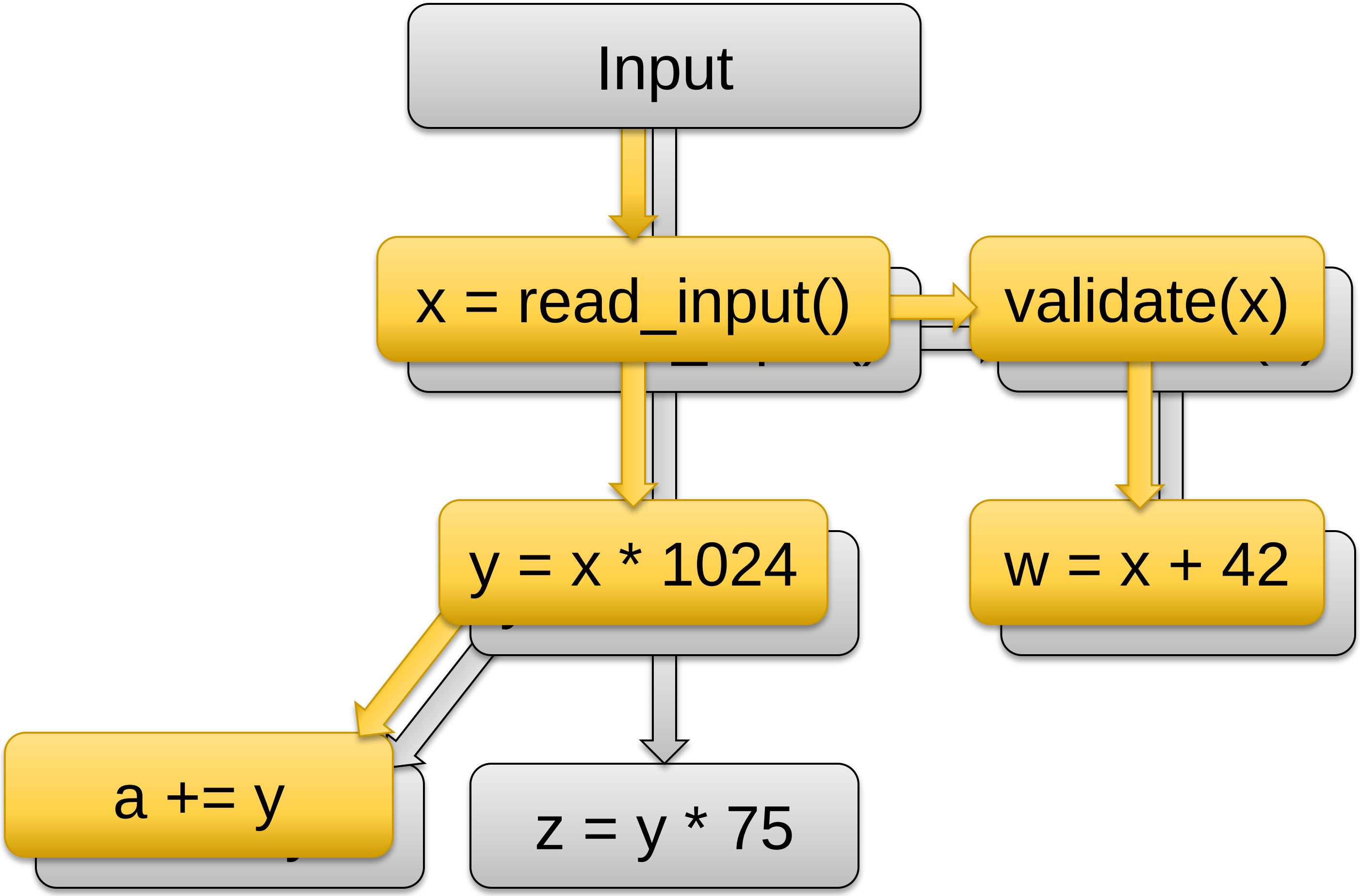
Cannot Naïvely Sample Code



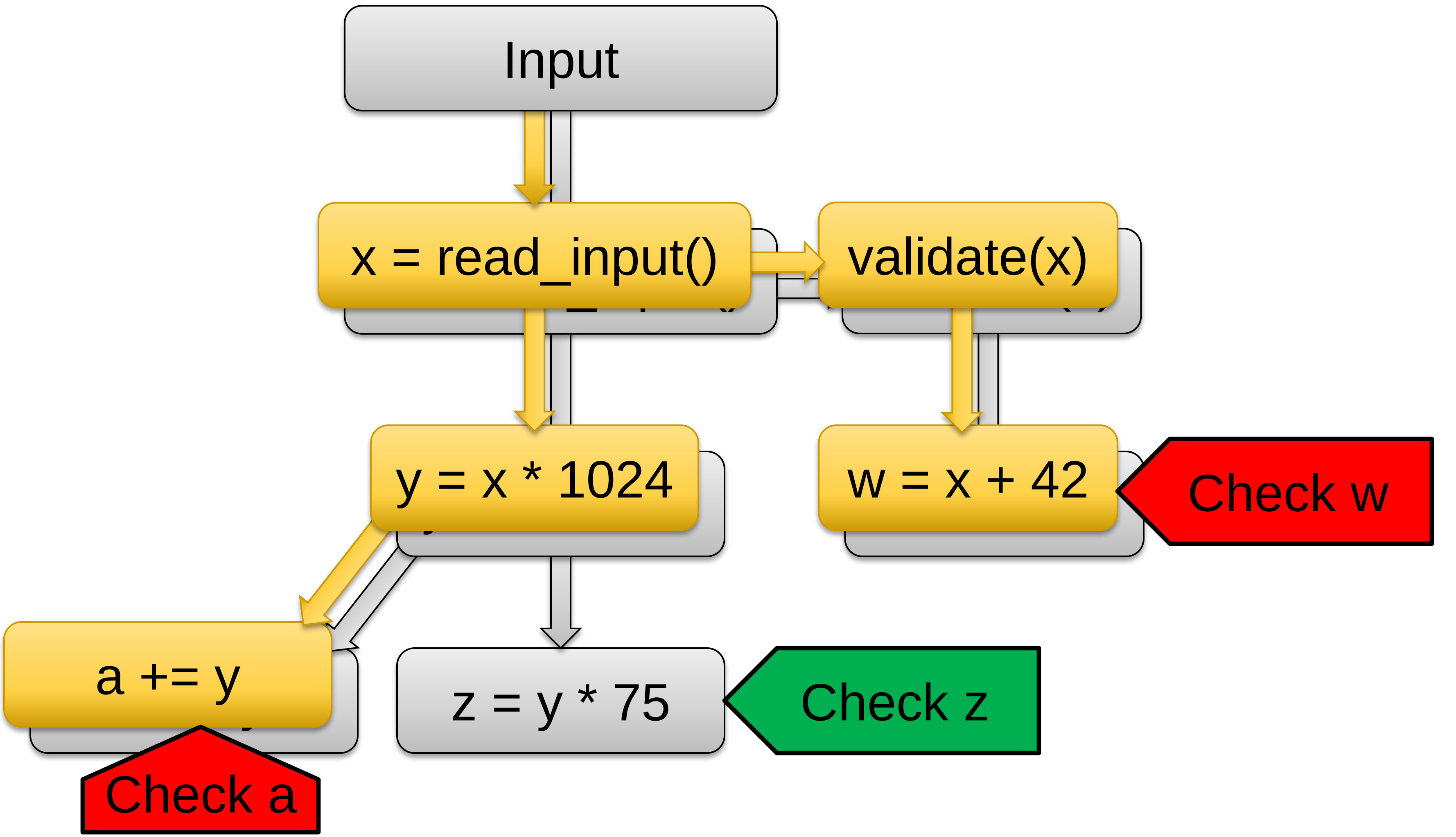
Cannot Naïvely Sample Code



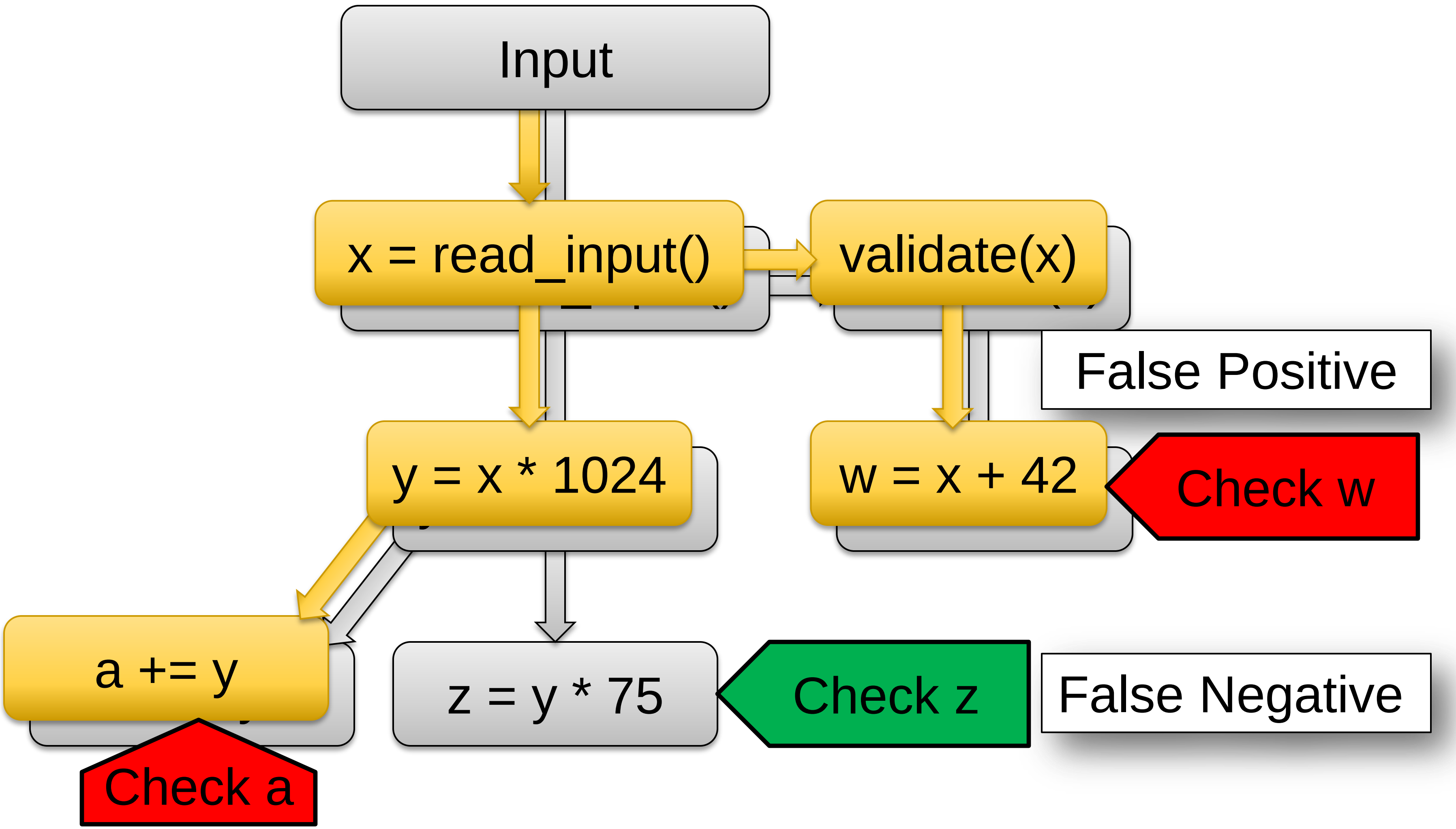
Cannot Naïvely Sample Code



Cannot Naïvely Sample Code

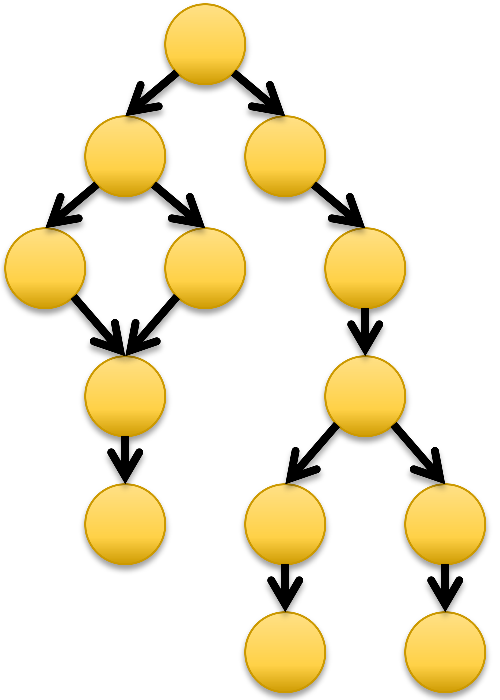


Cannot Naïvely Sample Code



Our Solution: Sample Data, not Code

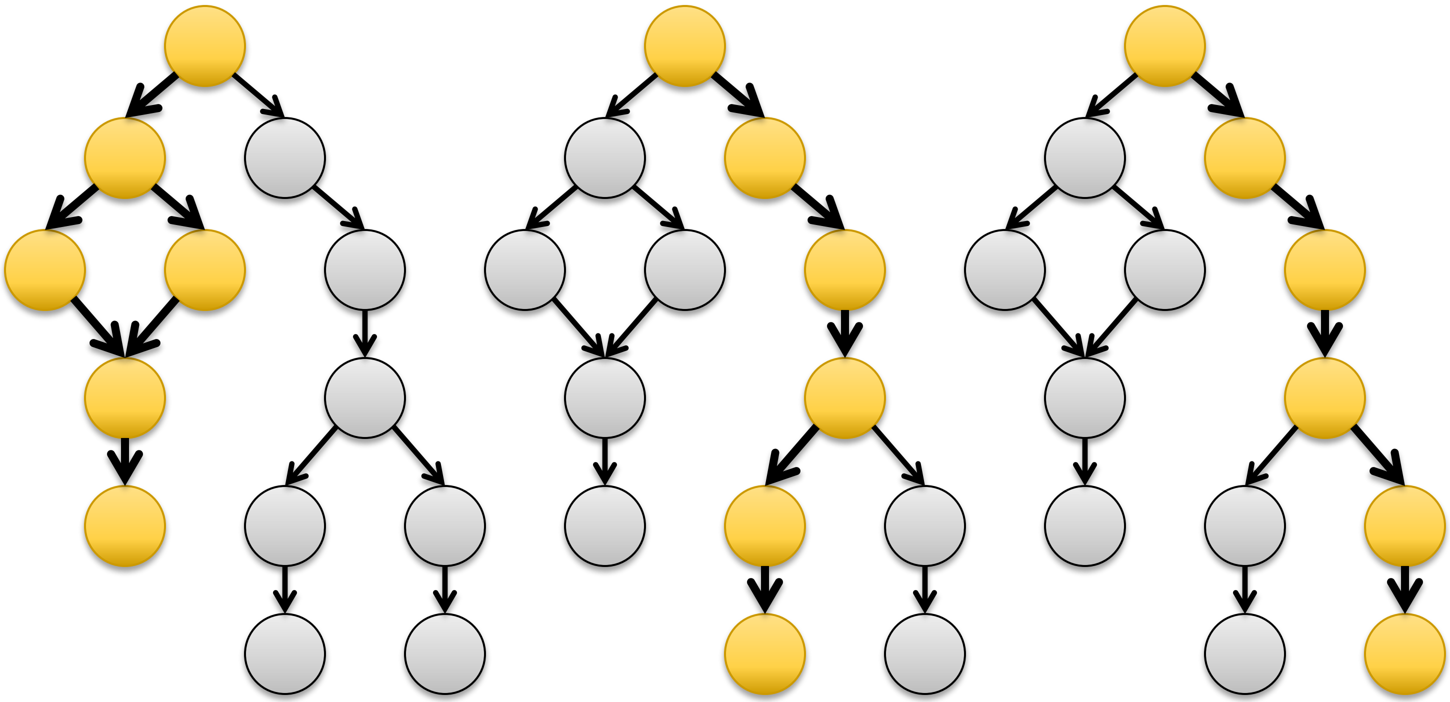
- Sampling must be aware of meta-data



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

Our Solution: Sample Data, not Code

- Sampling must be aware of meta-data

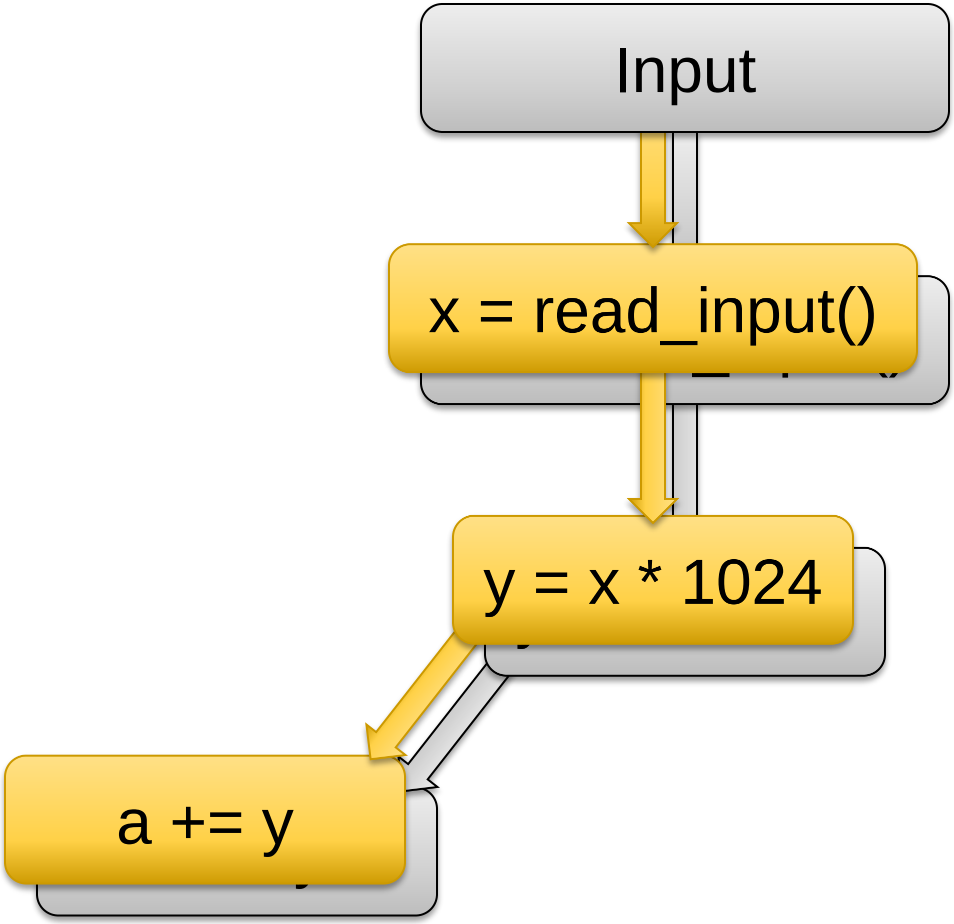


- Remove meta-data from skipped dataflows
  - Prevents false positives

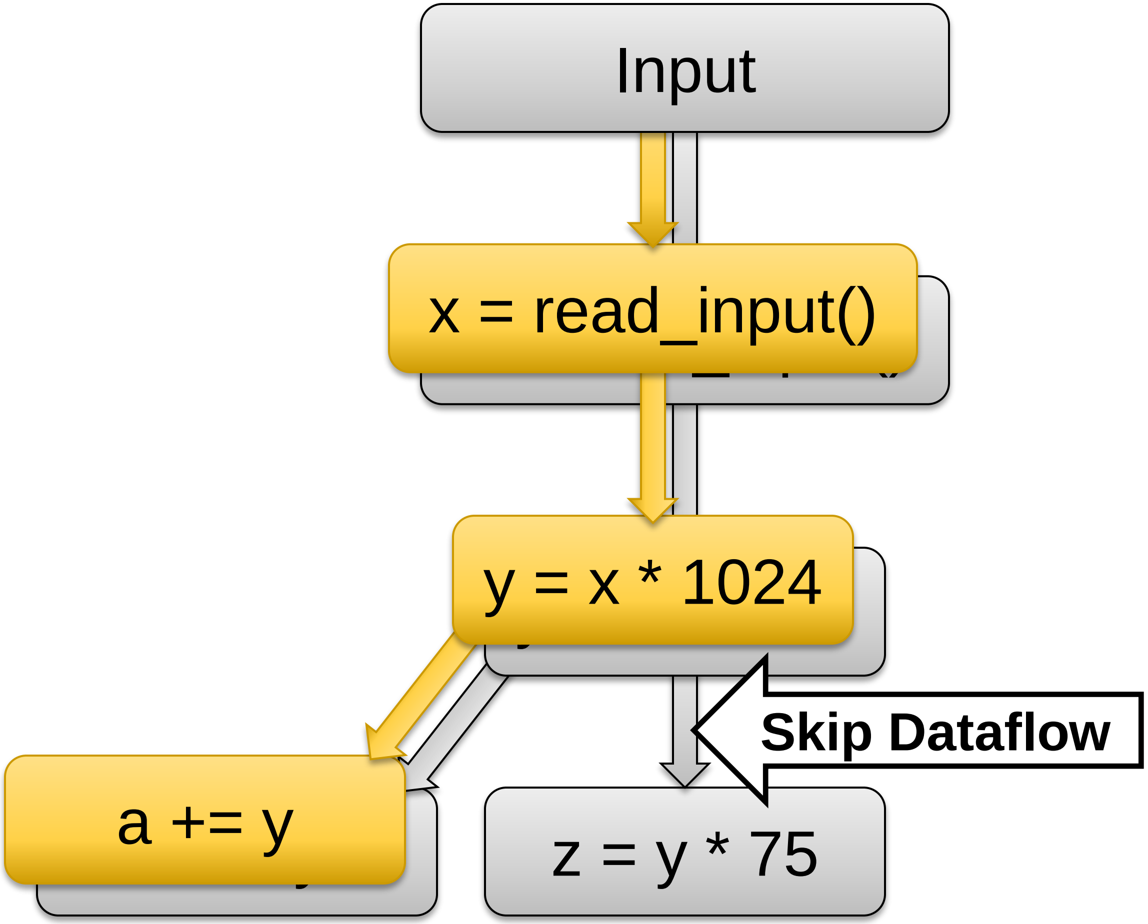
Dataflow Sampling Example

Input

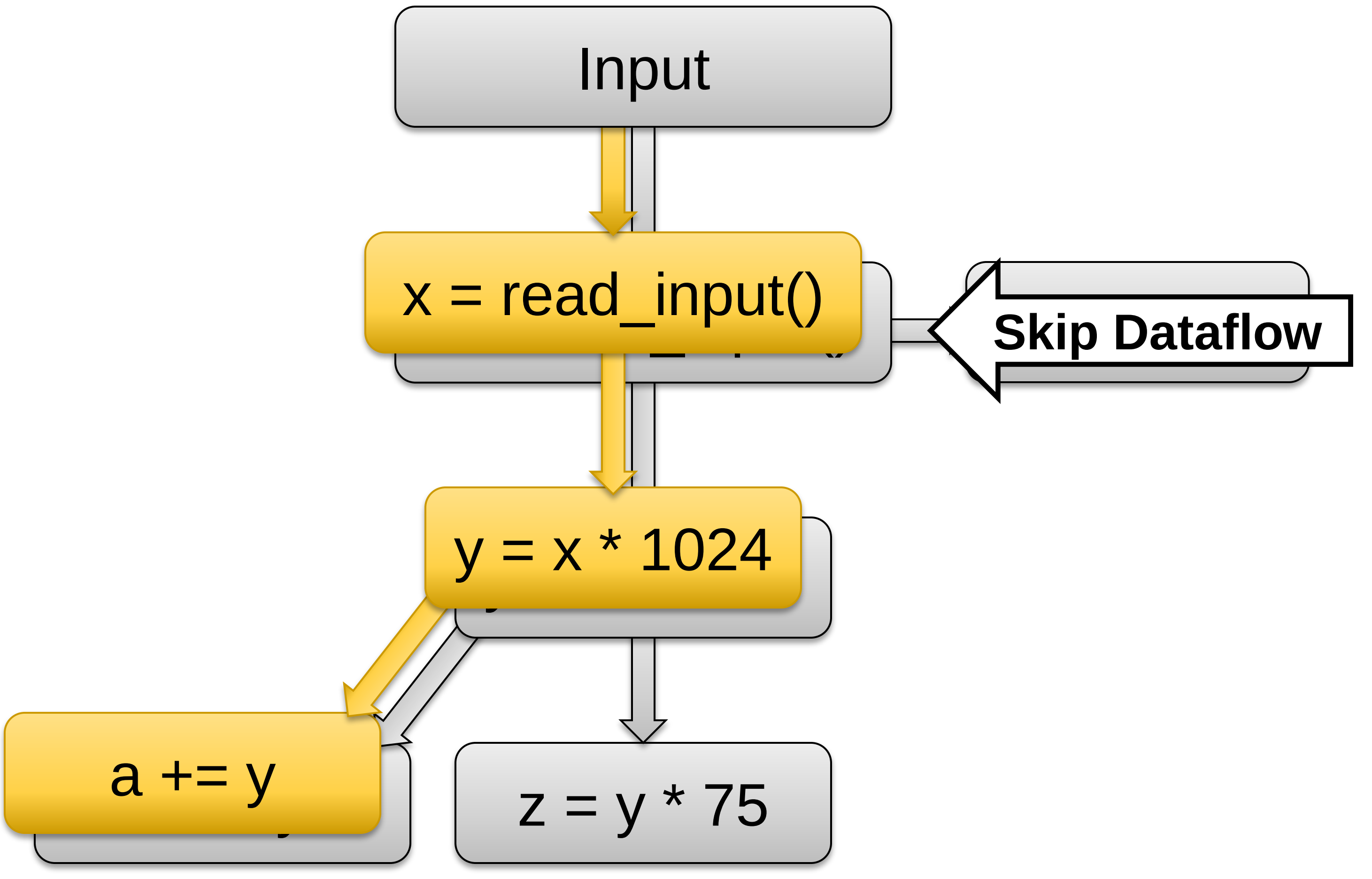
Dataflow Sampling Example



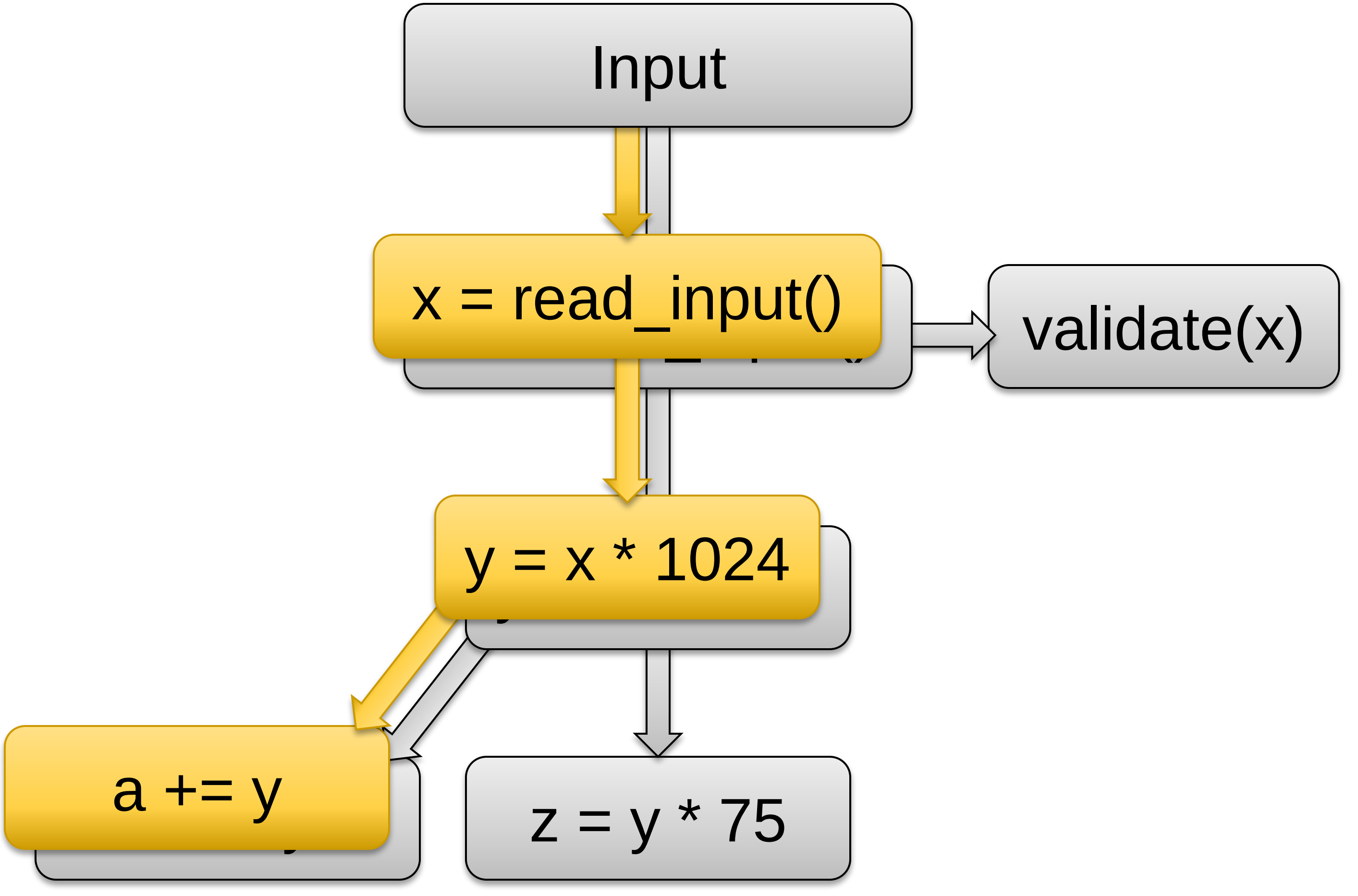
Dataflow Sampling Example



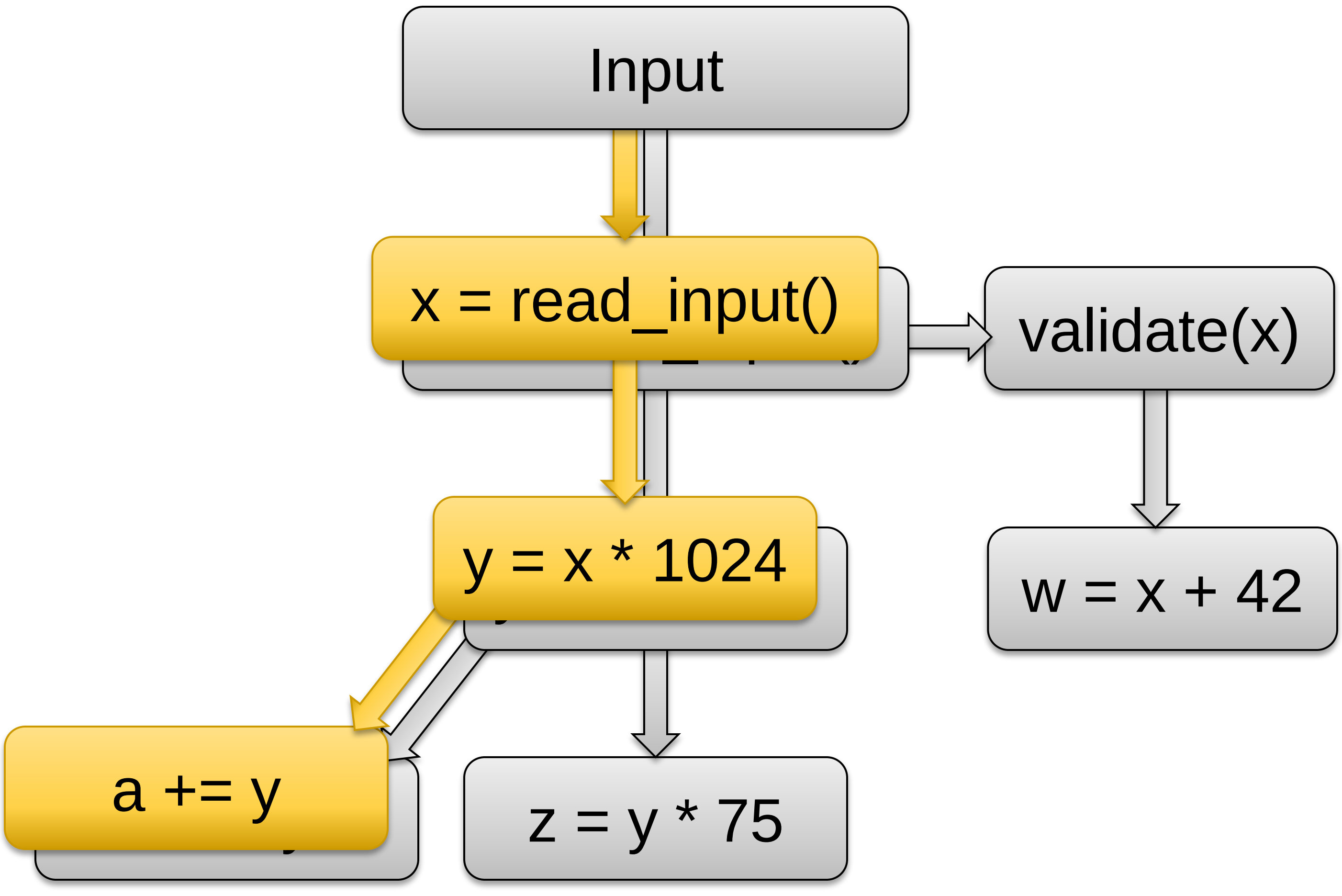
Dataflow Sampling Example



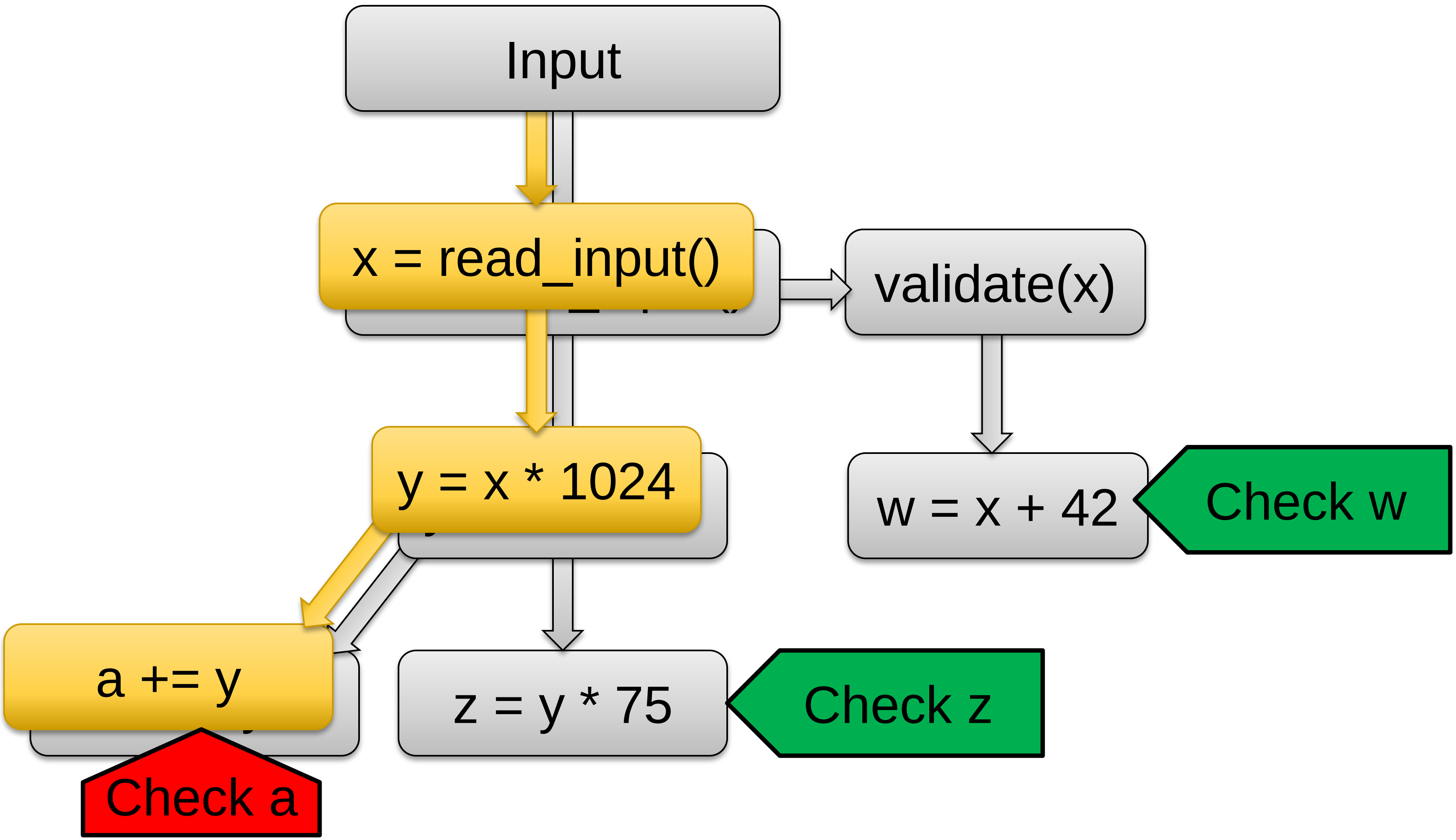
Dataflow Sampling Example



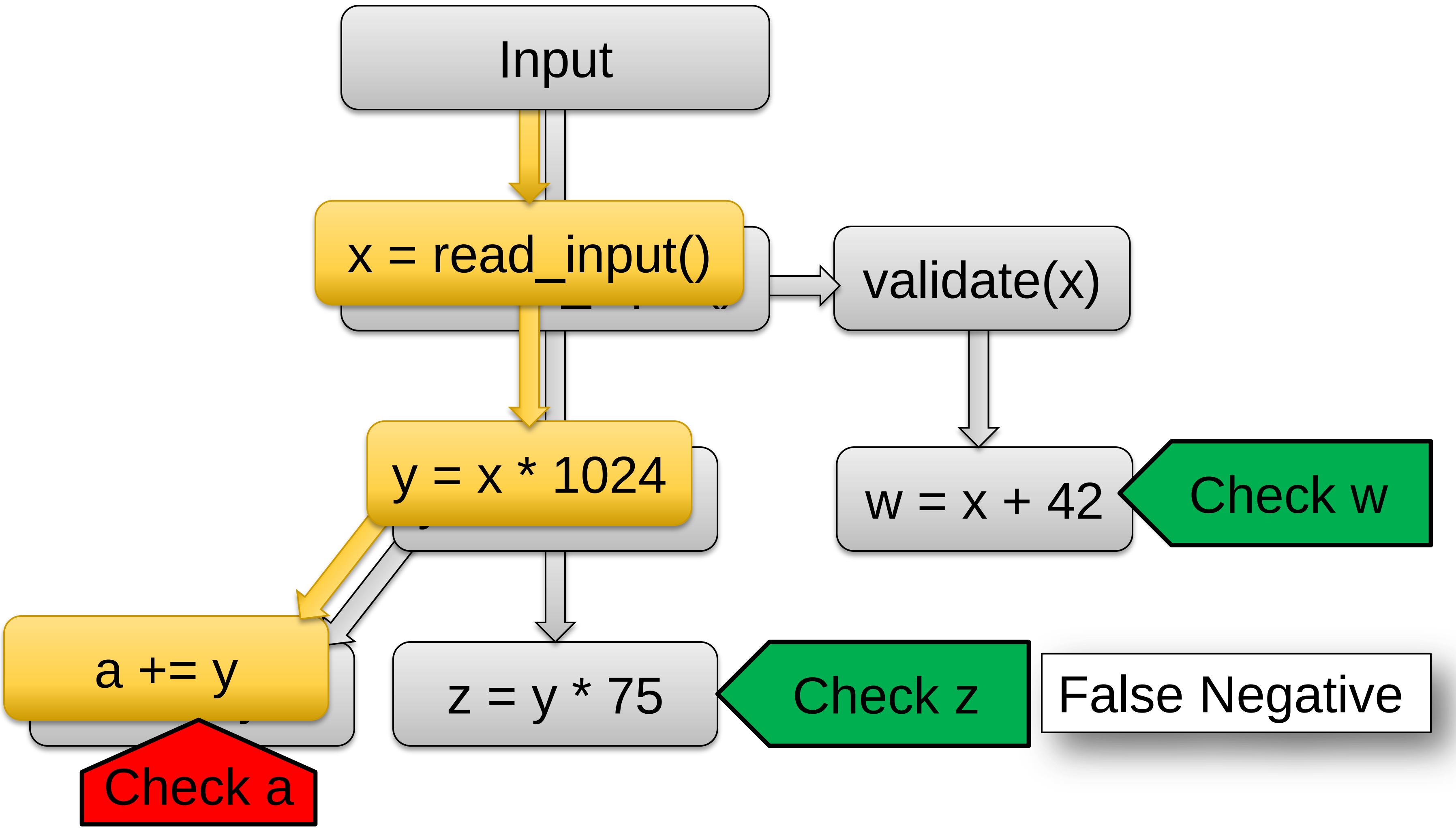
Dataflow Sampling Example



Dataflow Sampling Example

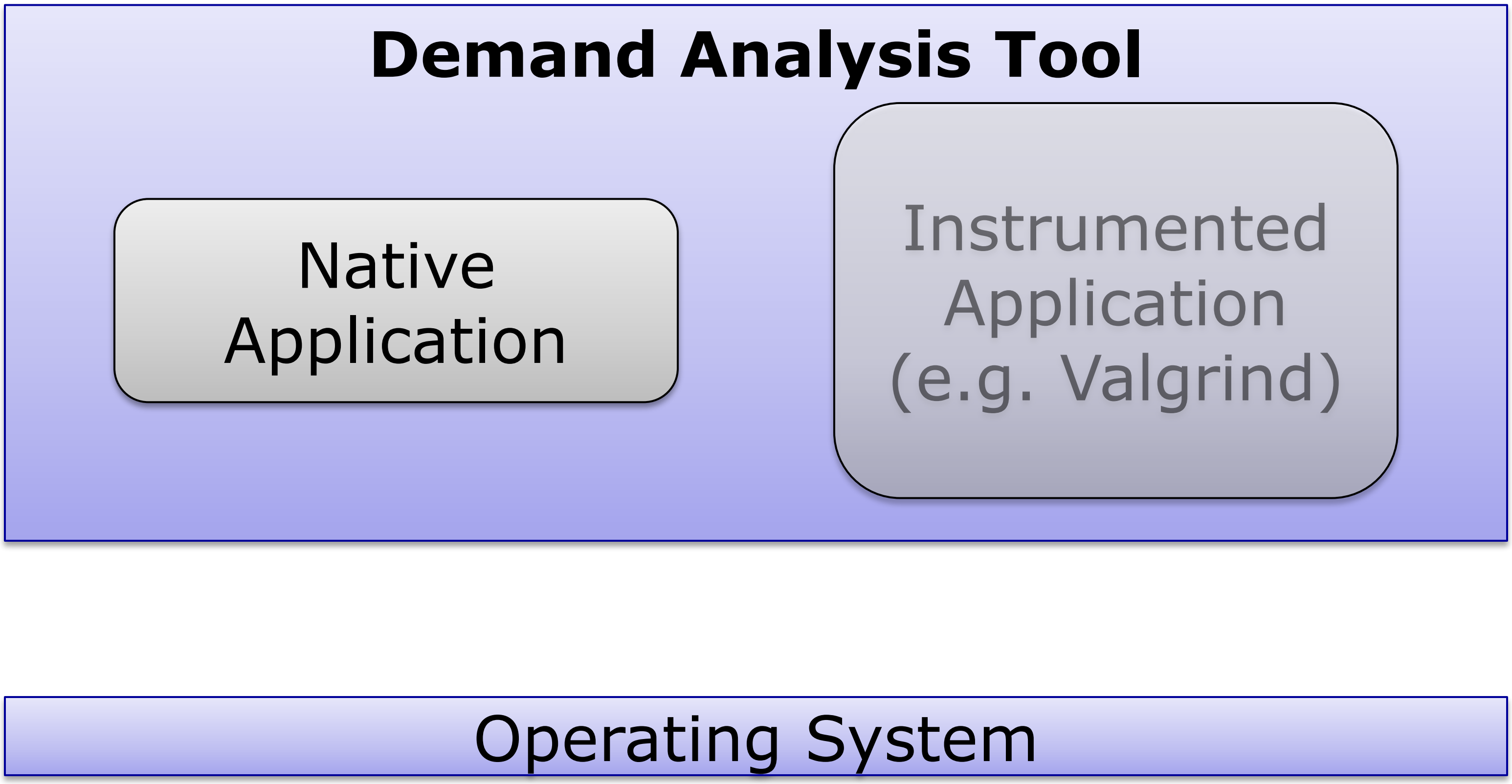


Dataflow Sampling Example



Mechanisms for Dataflow Sampling (1)

- Start with demand analysis



Mechanisms for Dataflow Sampling (1)

- Start with demand analysis

Demand Analysis Tool

Native Application

Instrumented Application (e.g. Valgrind)

Operating System

Mechanisms for Dataflow Sampling (1)

- Start with demand analysis

Demand Analysis Tool

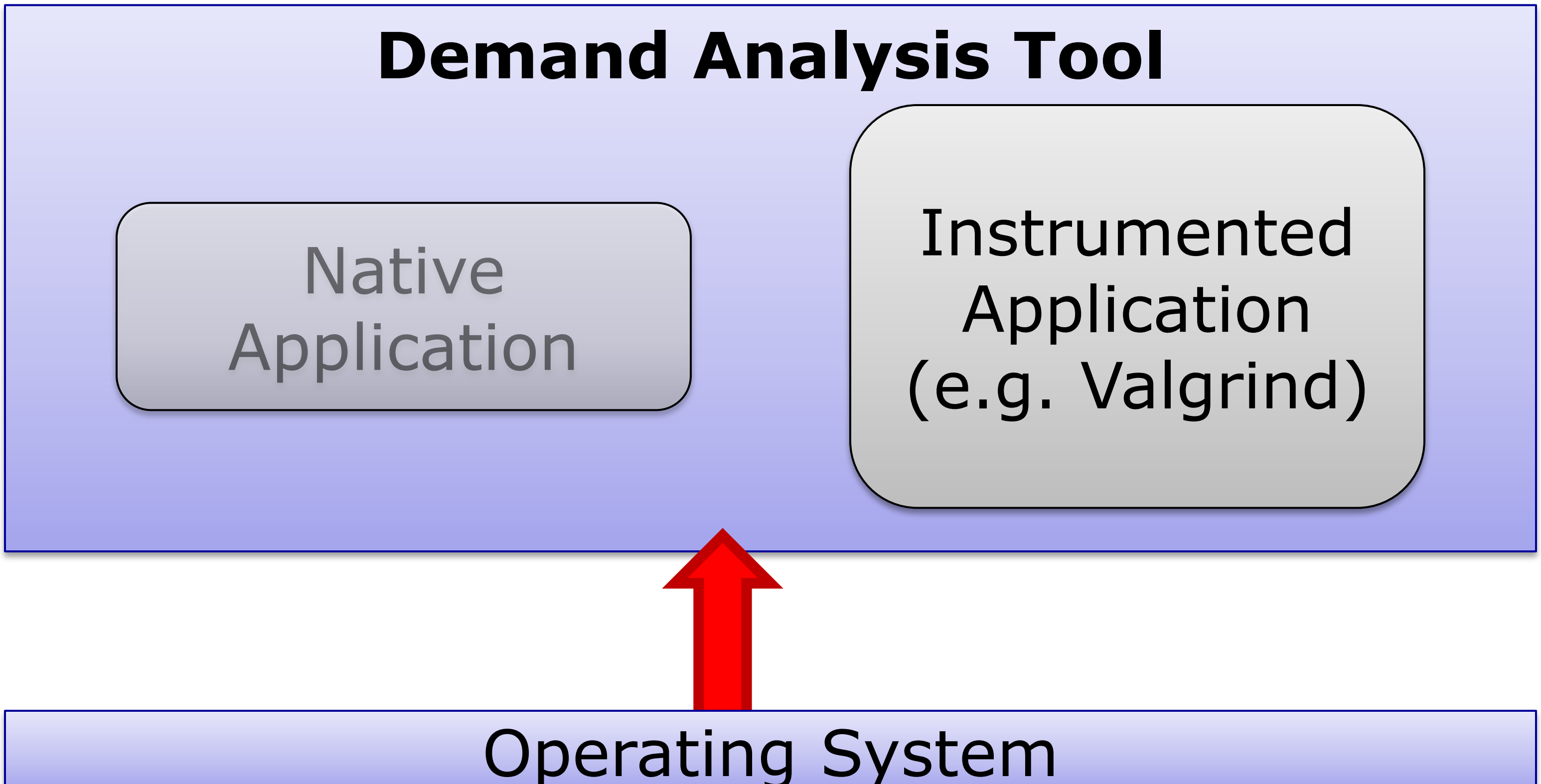
Native Application

Instrumented Application (e.g. Valgrind)

Op **Meta-data** ystem

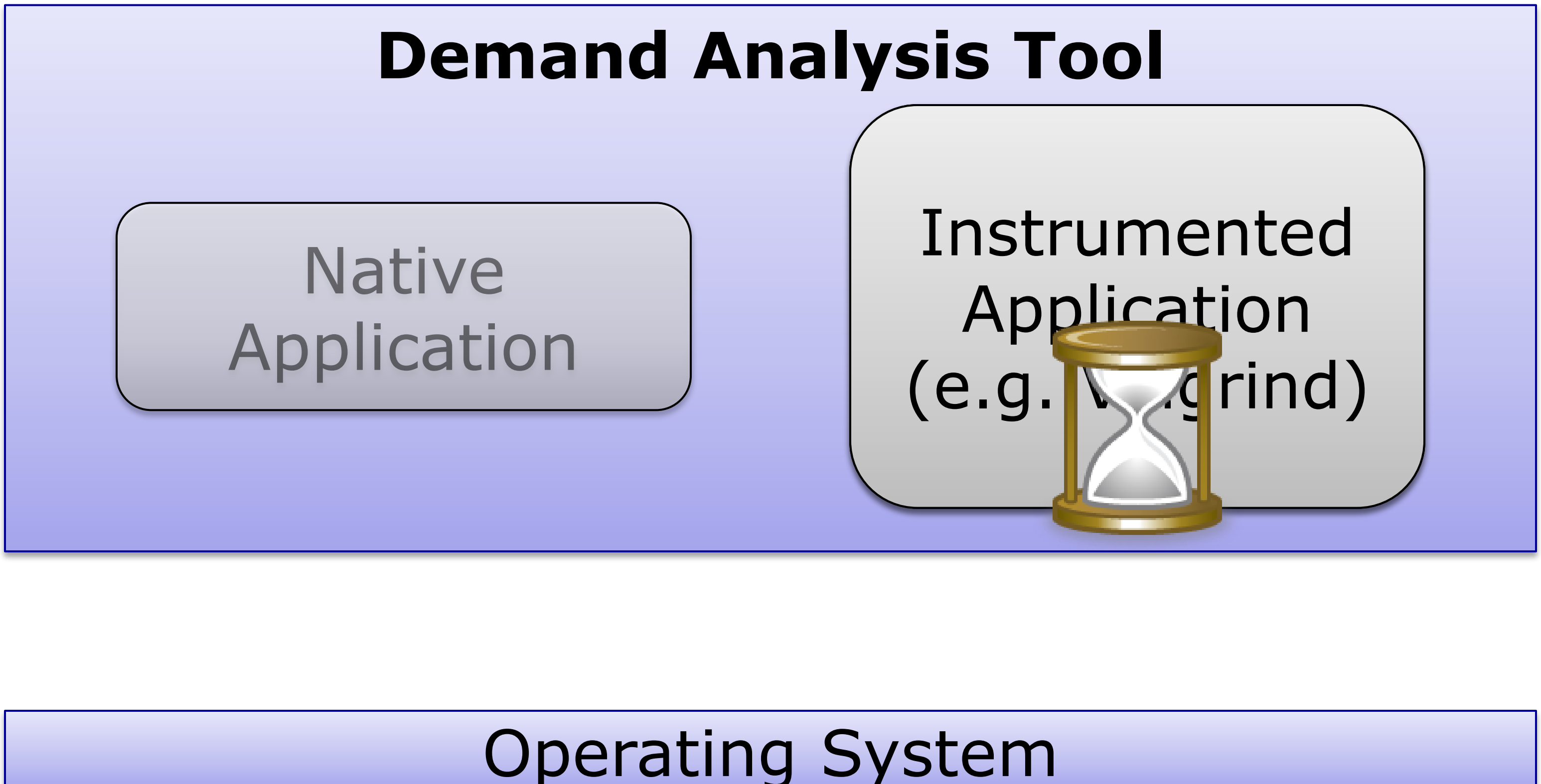
Mechanisms for Dataflow Sampling (1)

- Start with demand analysis



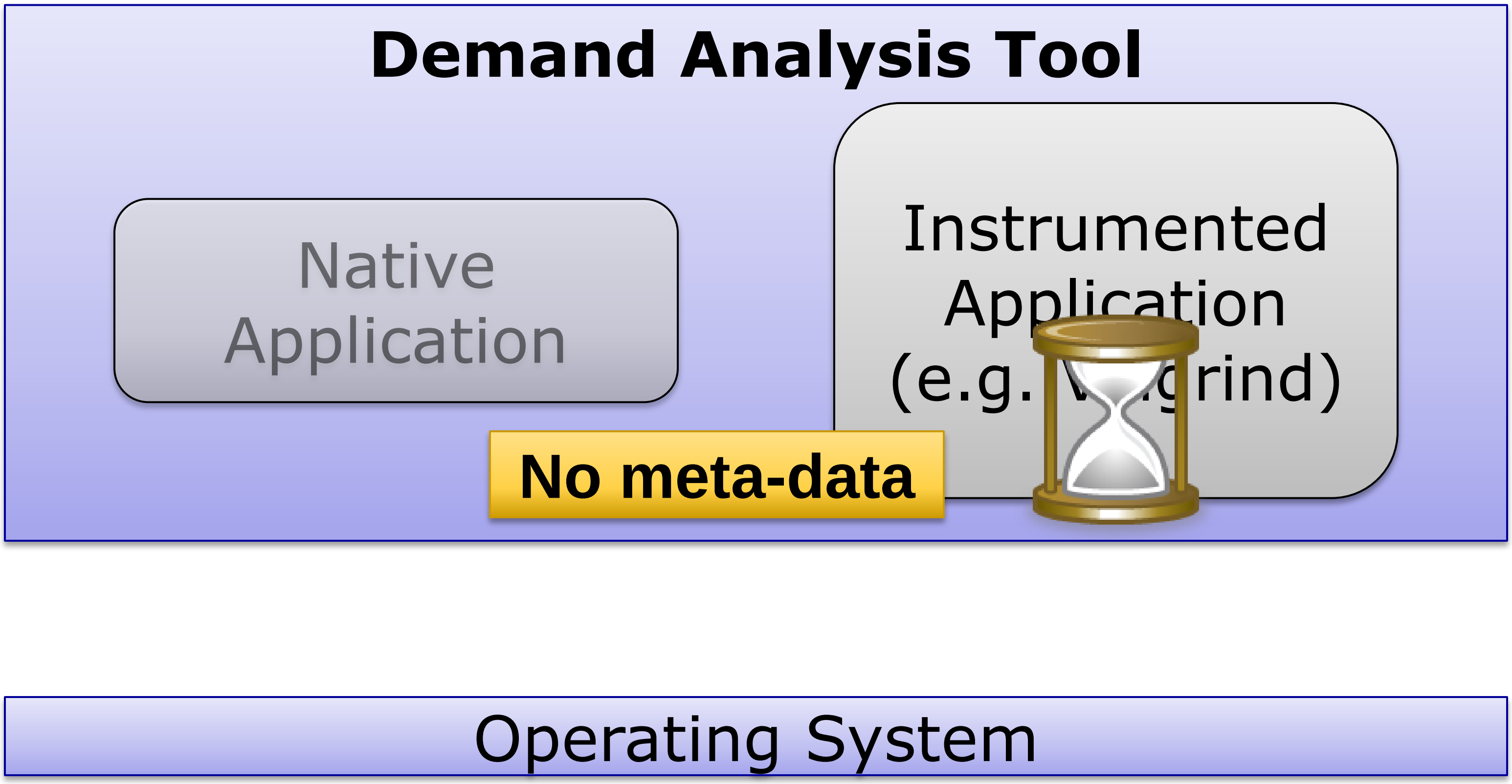
Mechanisms for Dataflow Sampling (1)

- Start with demand analysis



Mechanisms for Dataflow Sampling (1)

- Start with demand analysis



Mechanisms for Dataflow Sampling (1)

- Start with demand analysis

Demand Analysis Tool

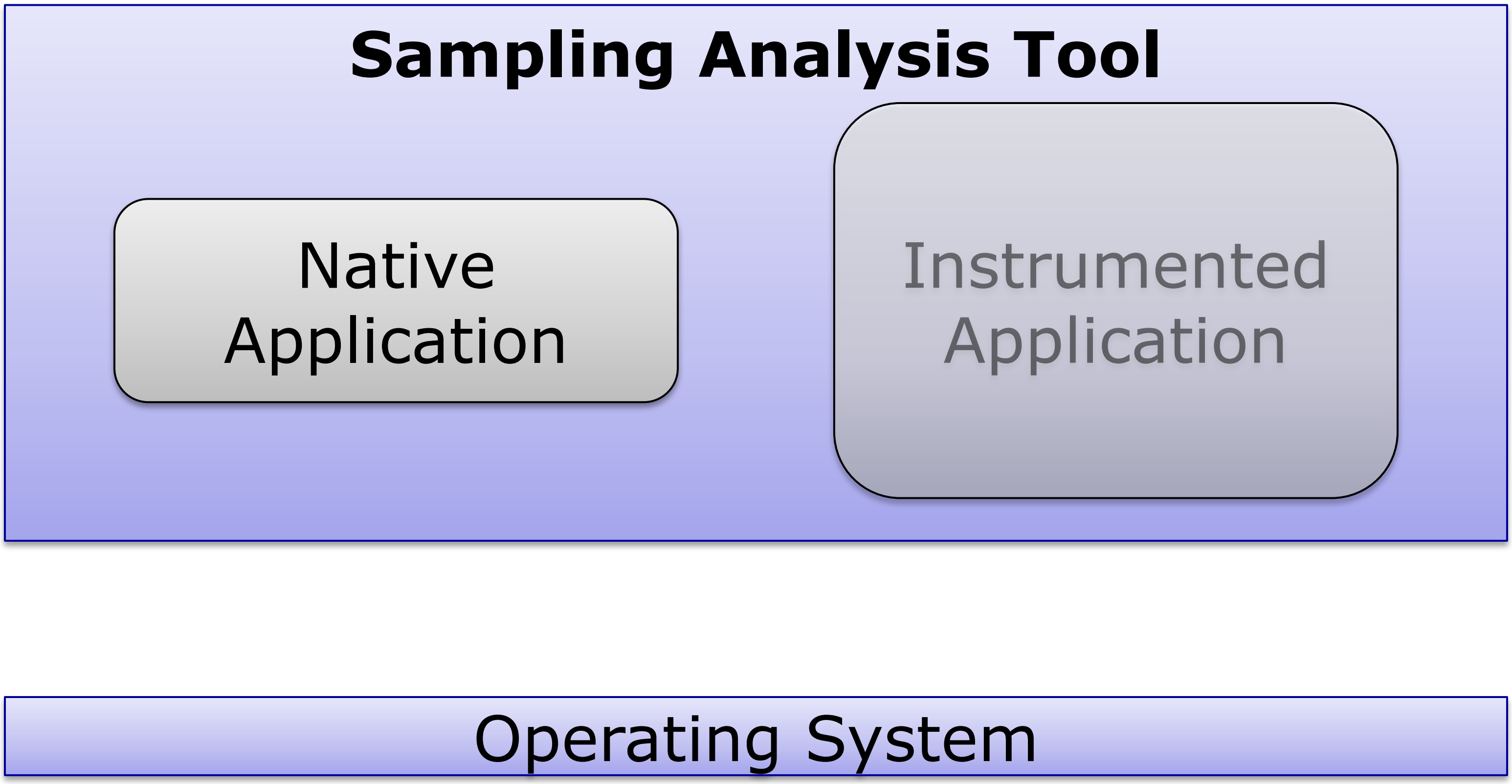
Native Application

Instrumented Application (e.g. Valgrind)

Operating System

Mechanisms for Dataflow Sampling (2)

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



Mechanisms for Dataflow Sampling (2)

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

Sampling Analysis Tool

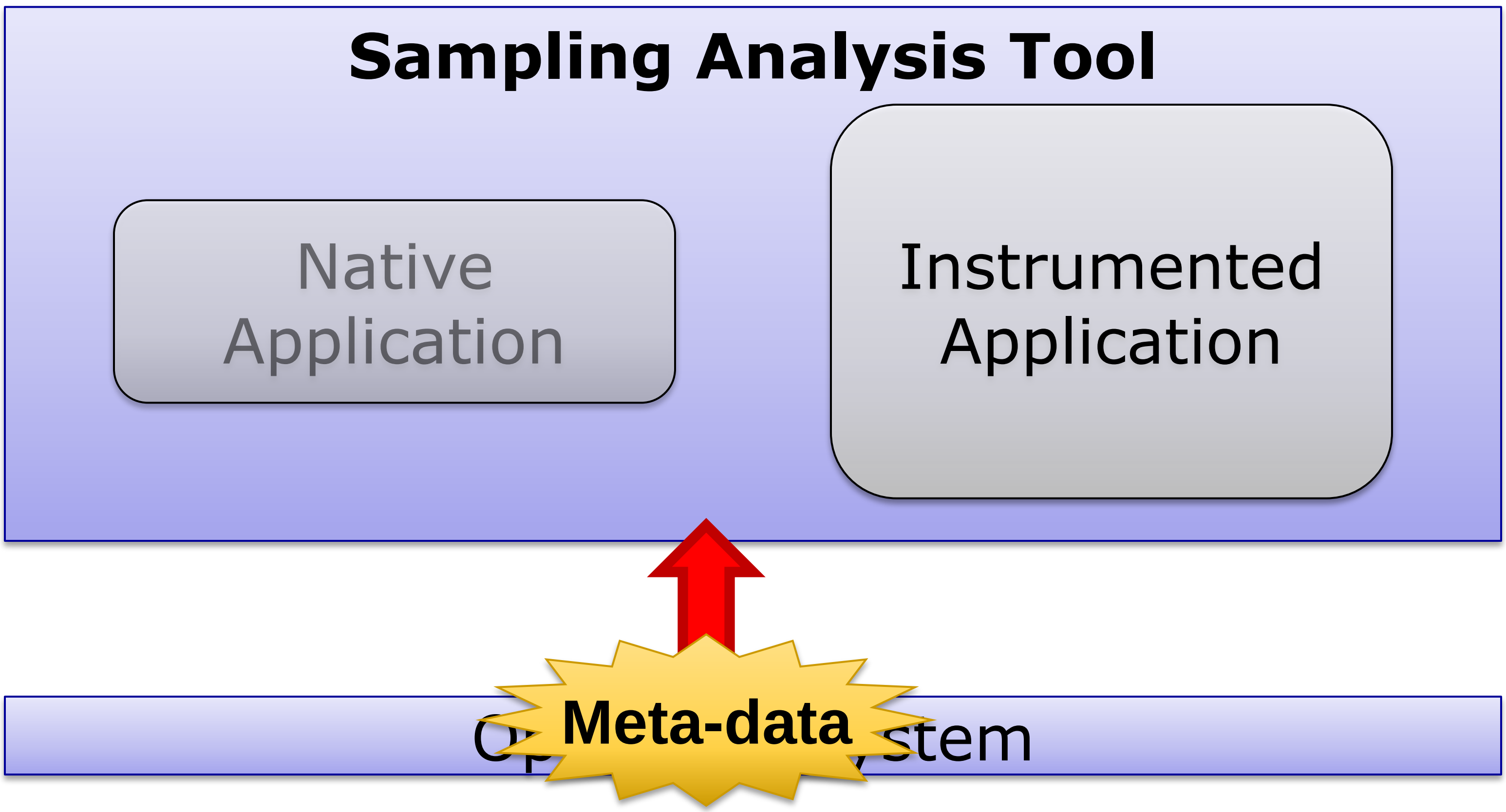
Native Application

Instrumented Application

Operating System

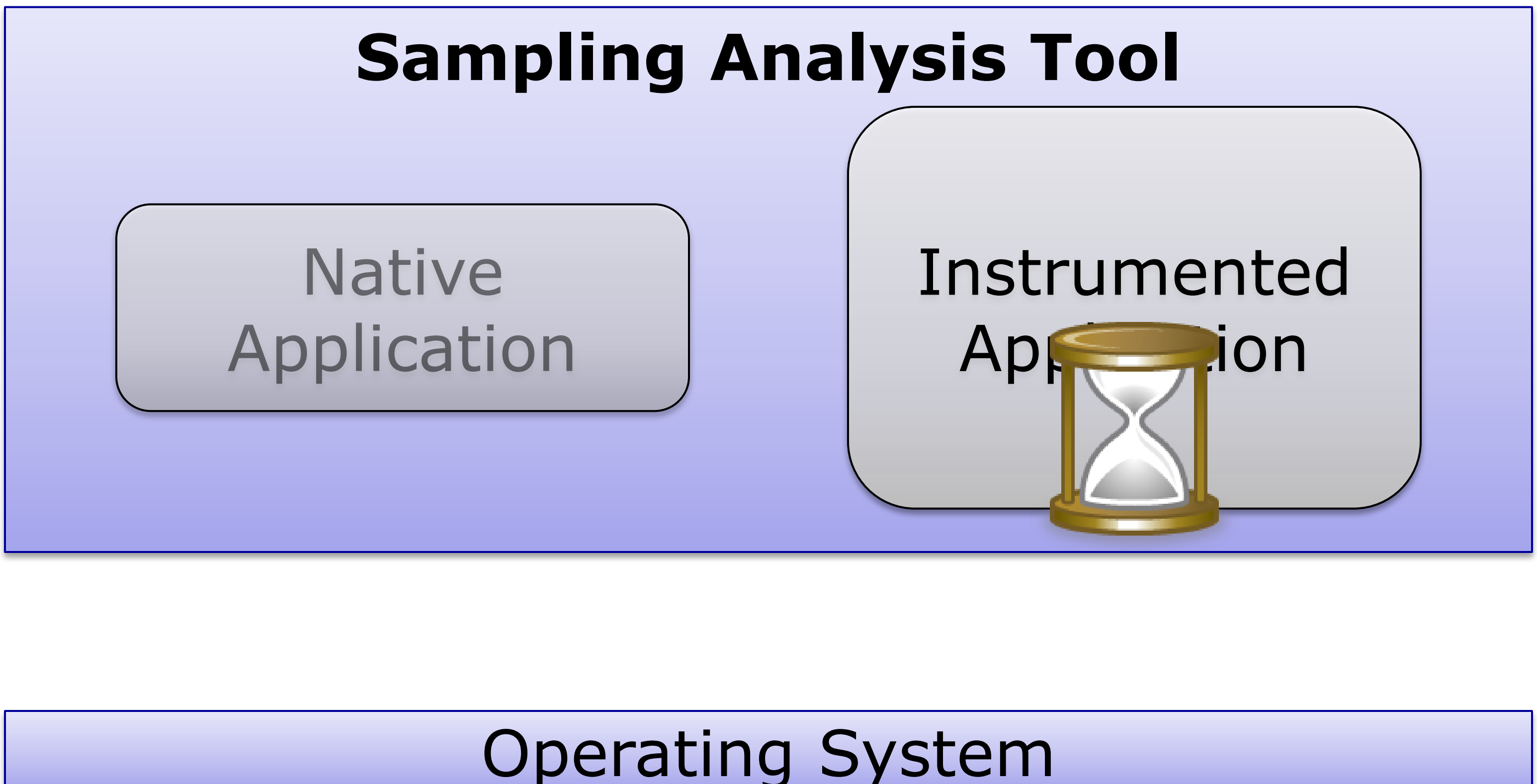
Mechanisms for Dataflow Sampling (2)

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



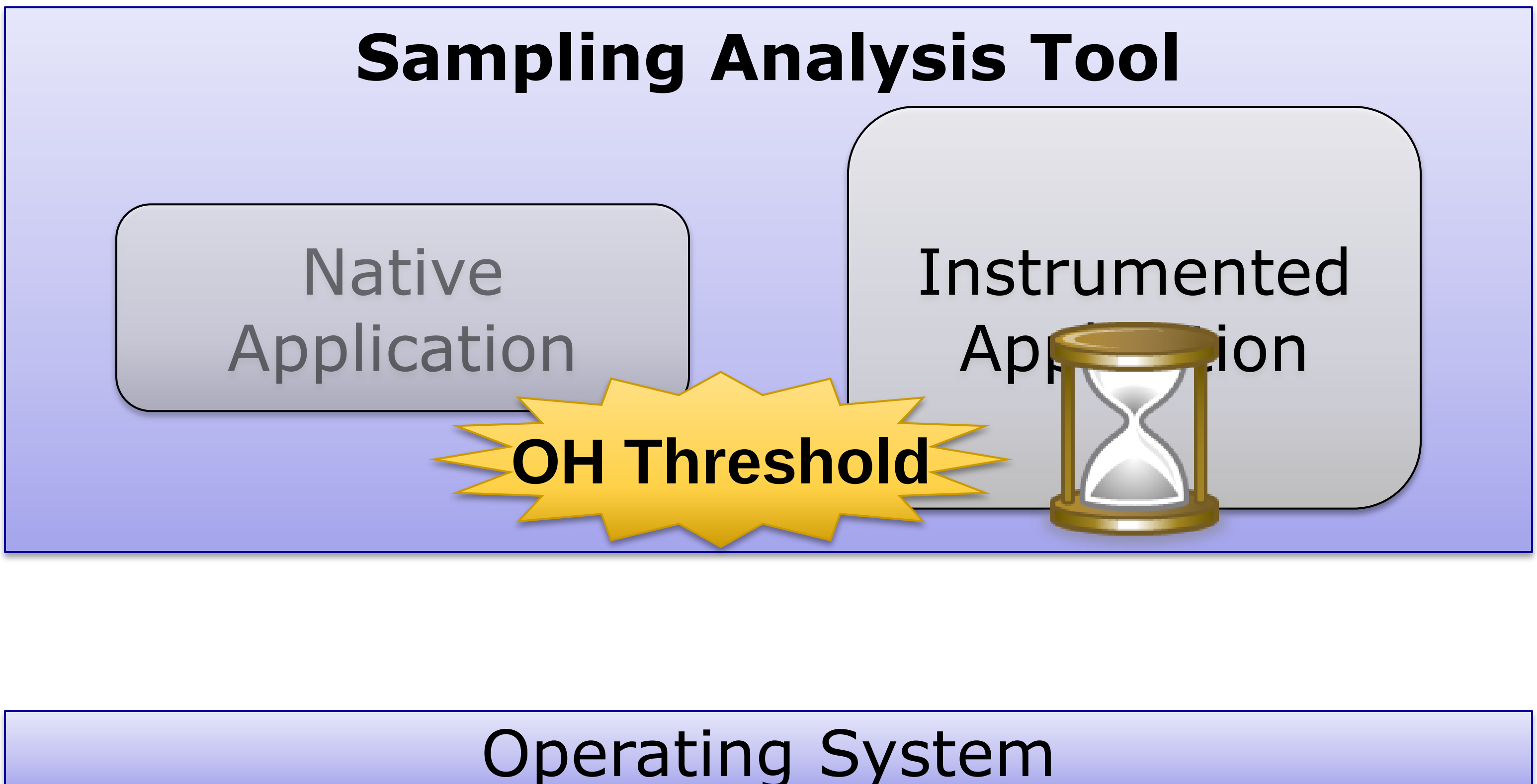
Mechanisms for Dataflow Sampling (2)

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



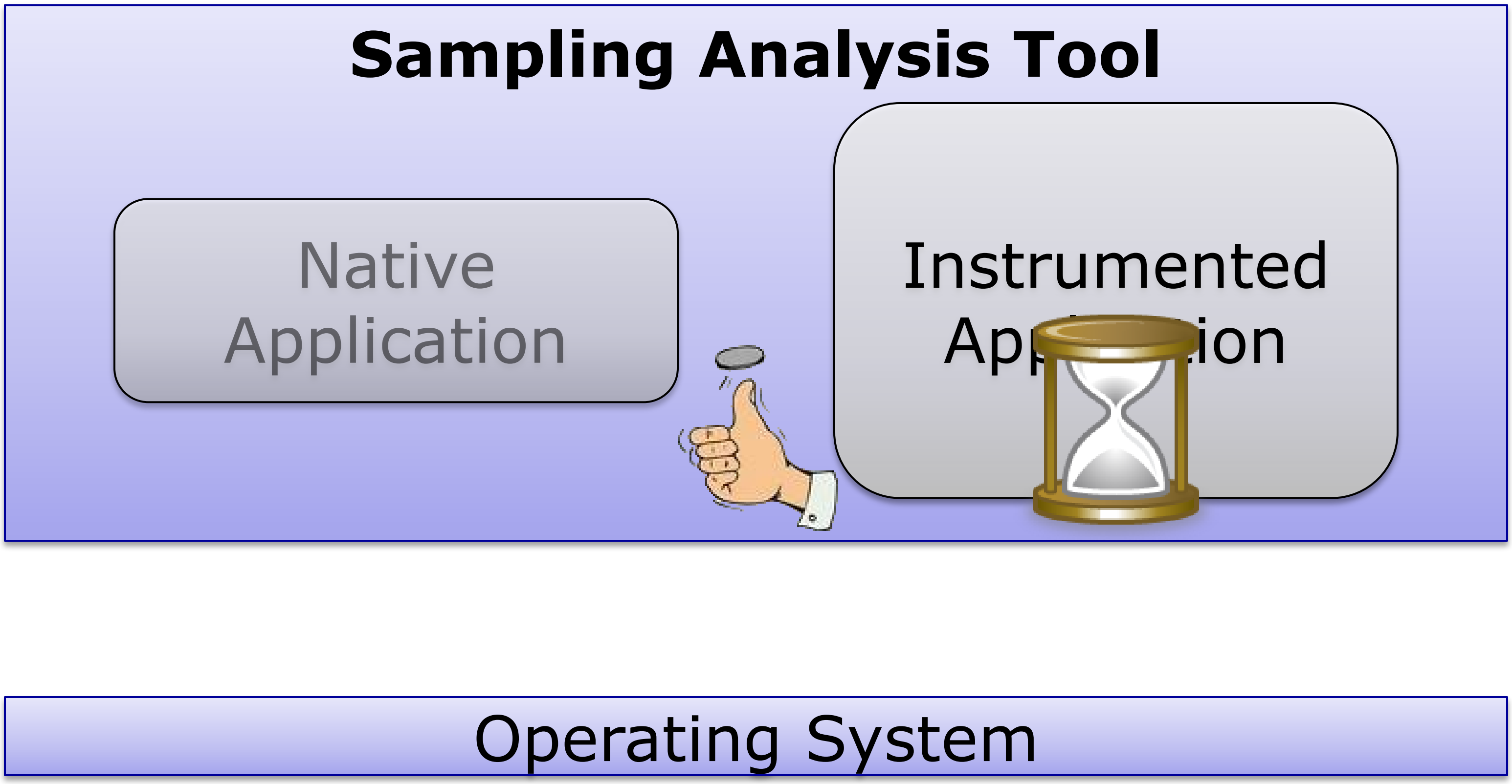
Mechanisms for Dataflow Sampling (2)

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



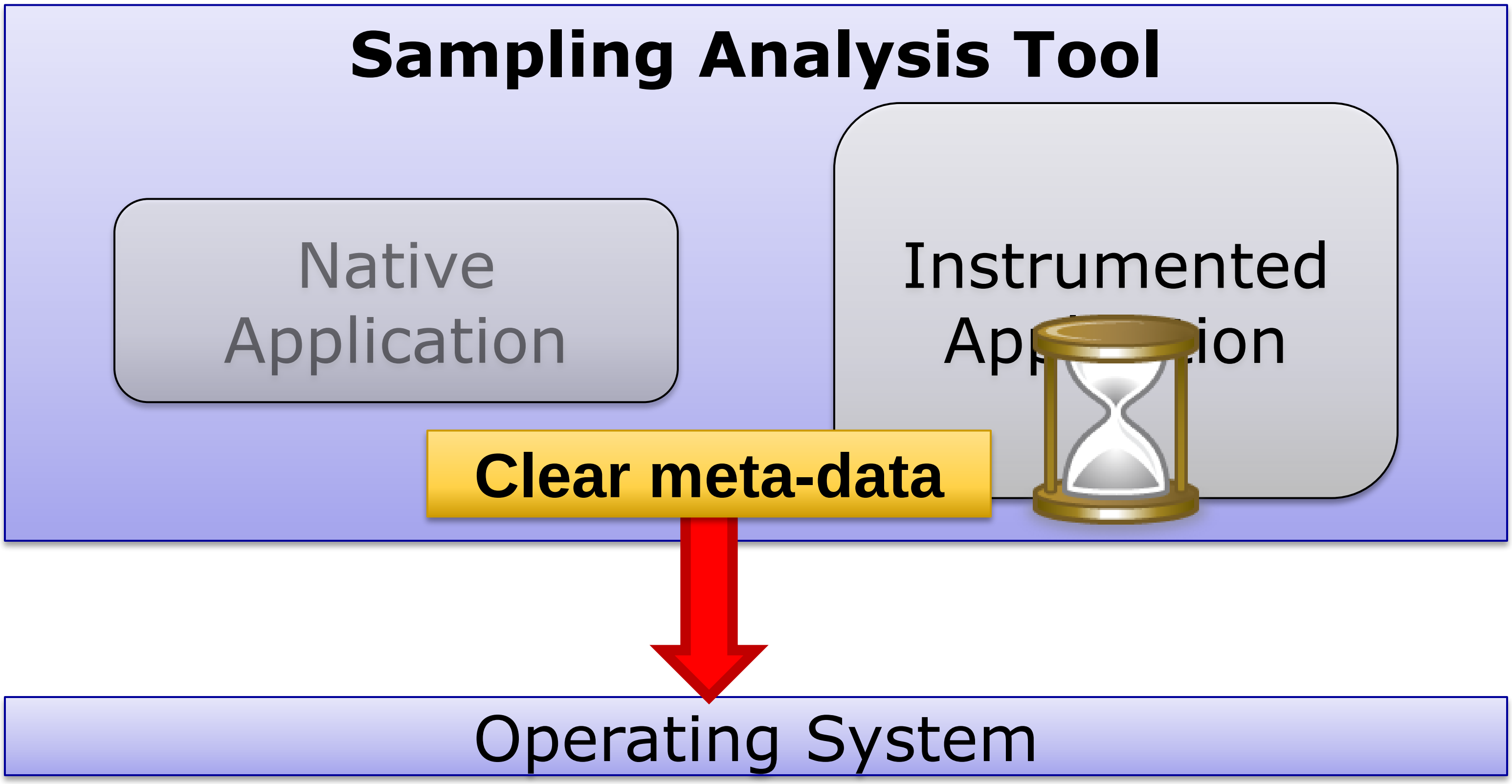
Mechanisms for Dataflow Sampling (2)

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



Mechanisms for Dataflow Sampling (2)

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



Mechanisms for Dataflow Sampling (2)

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

Sampling Analysis Tool

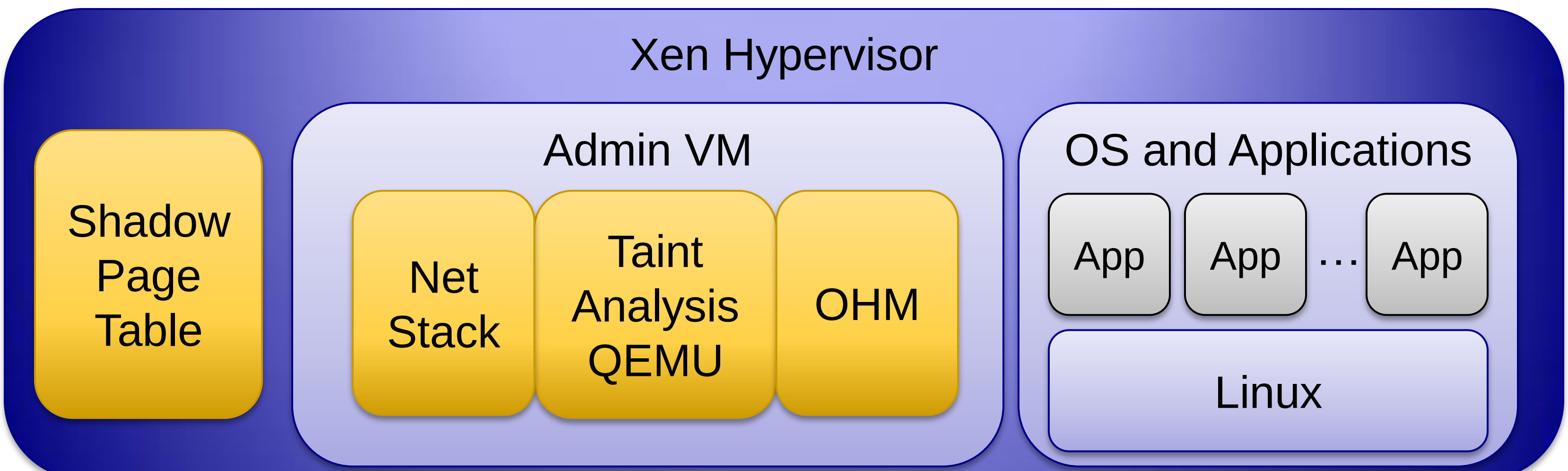
Native Application

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

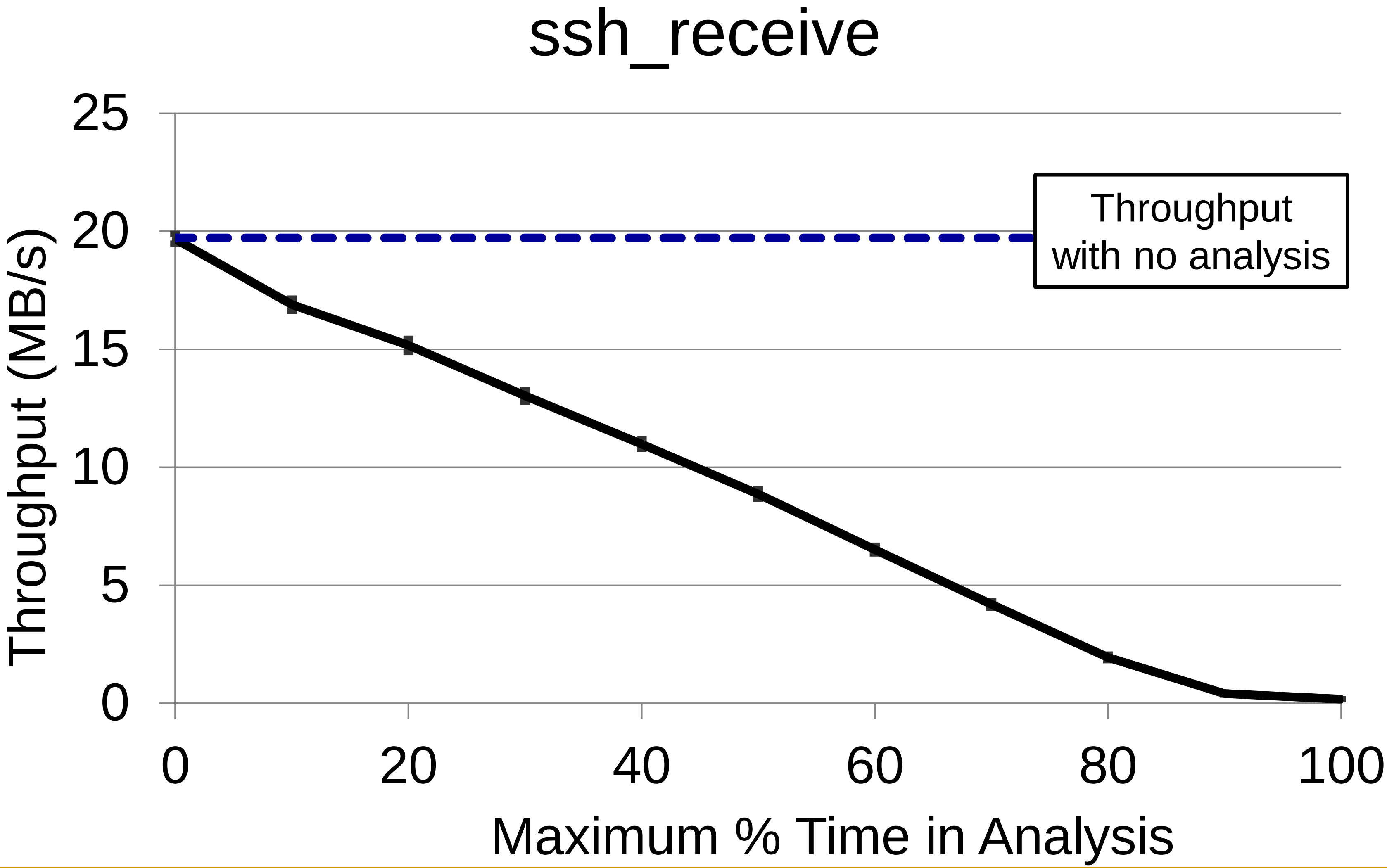


Benchmarks

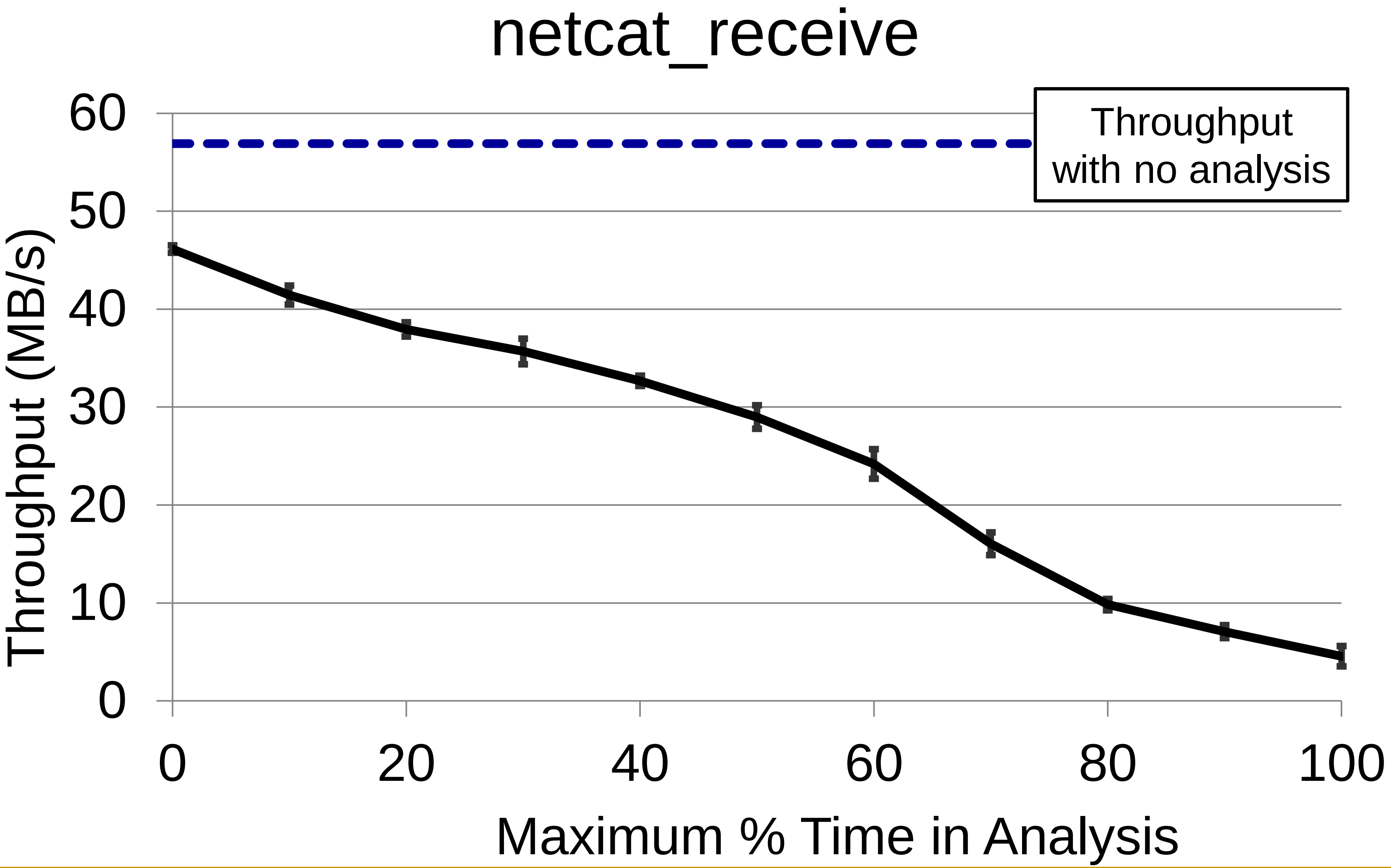
- Performance – Network Throughput
  - *Example:* ***ssh_receive***
- Accuracy of Sampling Analysis
  - Real-world Security Exploits

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

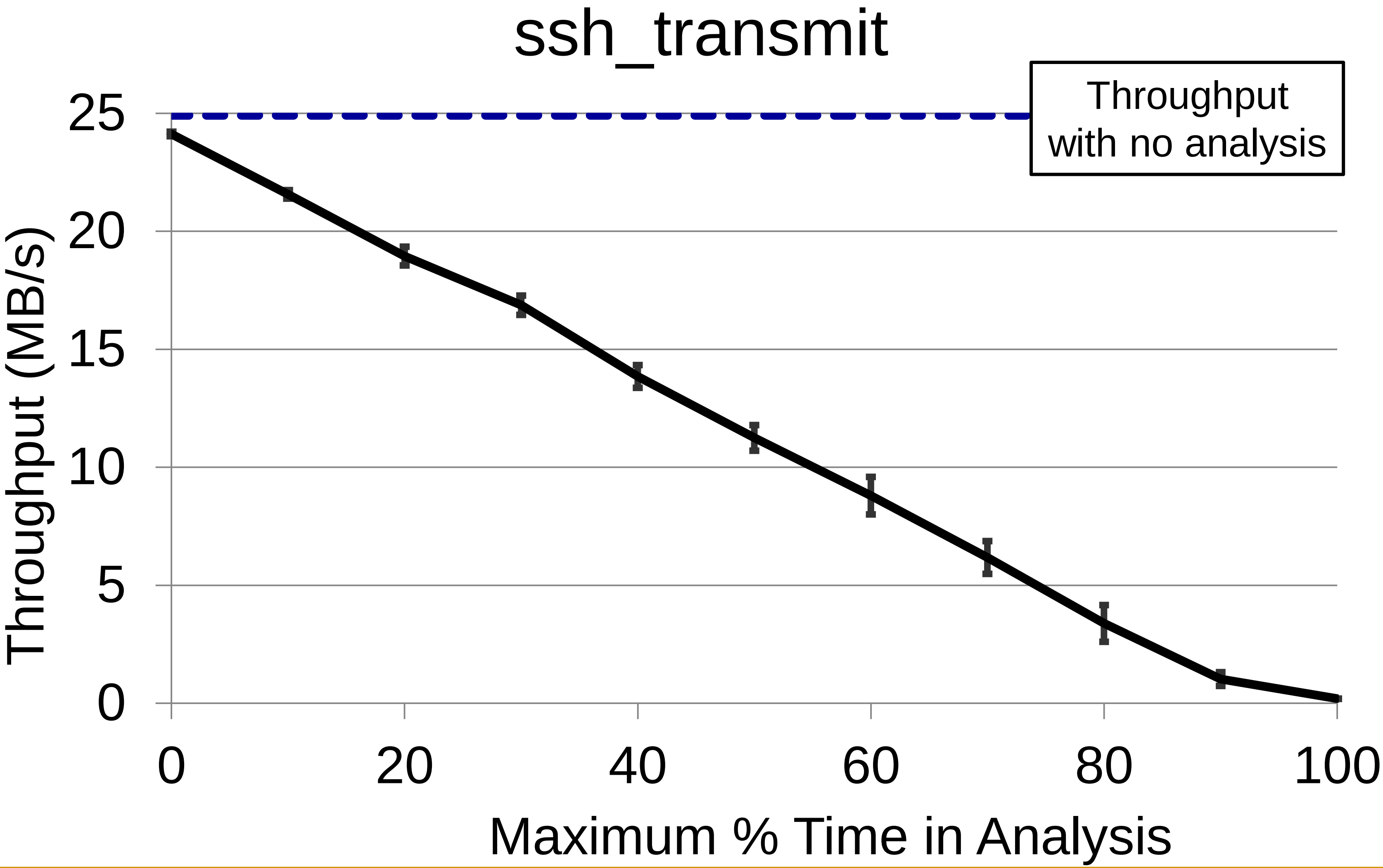
Performance of Dataflow Sampling (1)



Performance of Dataflow Sampling (2)



Performance of Dataflow Sampling (3)



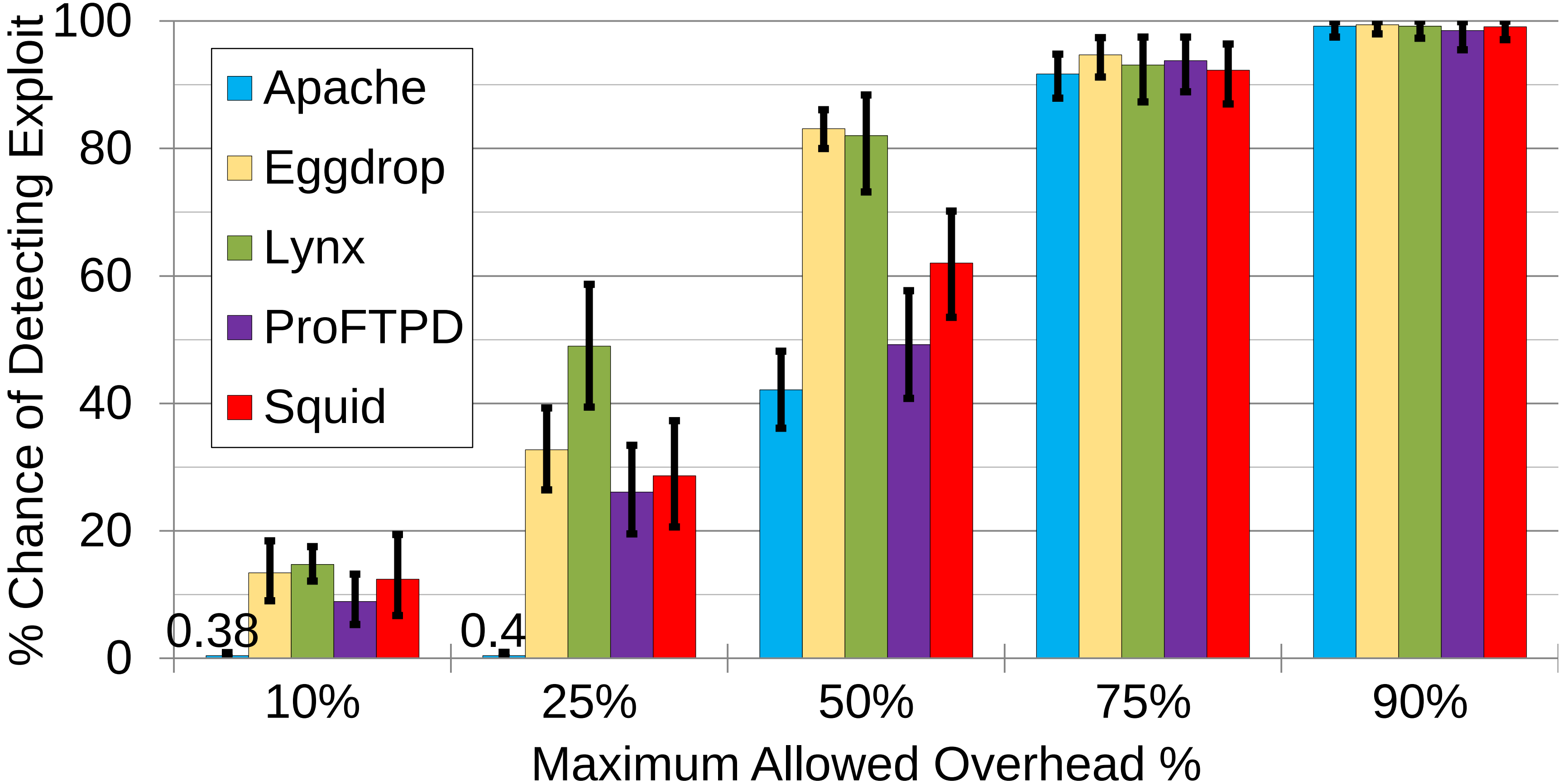
Accuracy at Very Low Overhead

- Max time in analysis: 1% every 10 seconds
- Always stop analysis after threshold
  - Lowest probability of detecting exploits

Name	Chance of Detecting Exploit
Apache	100%
Eggdrop	100%
Lynx	100%
ProFTPD	100%
Squid	100%

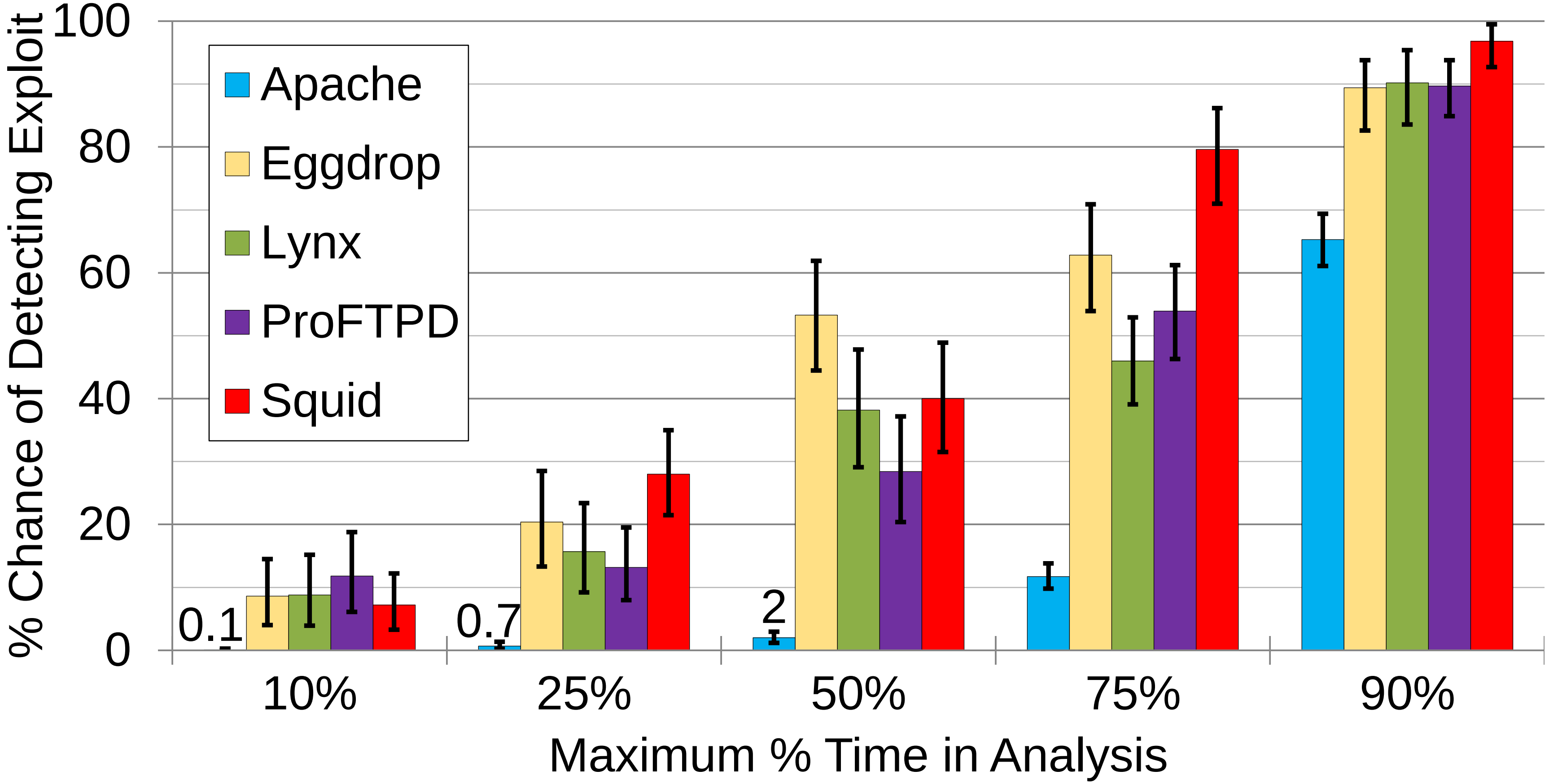
Accuracy with Background Tasks

netcat_receive running with benchmark



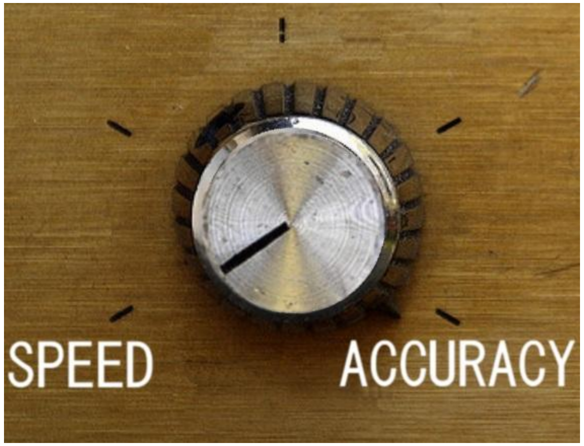
Accuracy with Background Tasks

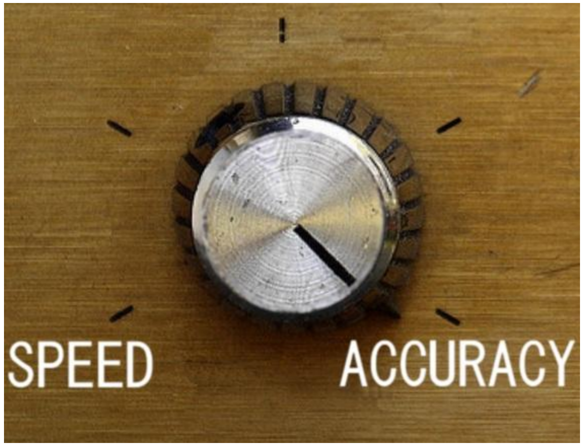
ssh_receive running in background



Conclusion & Future Work

Dynamic dataflow sampling gives users a knob to control accuracy vs. performance





- Better methods of sample choices
- Combine static information
- New types of sampling analysis

Conclusion & Future Work

Dynamic dataflow sampling gives users a knob to control accuracy vs. performance

- Better methods of sample choices
- Combine static information
- New types of sampling analysis

BACKUP SLIDES

Outline

- Software Errors and Security
- Dynamic Dataflow Analysis
- Sampling and Distributed Analysis
- Prototype System
- Performance and Accuracy

Width Test