

HeapMD: Identifying Heap-based Bugs using Anomaly Detection

Trishul M. Chilimbi

Microsoft Research

Redmond, WA

trishulc@microsoft.com

Vinod Ganapathy

University of Wisconsin

Madison, WI

vg@cs.wisc.edu

ASPLOS XII, October 2006

San Jose, California

A motivating example

```
pNewAsset = Initialize(pAssetParams);
```

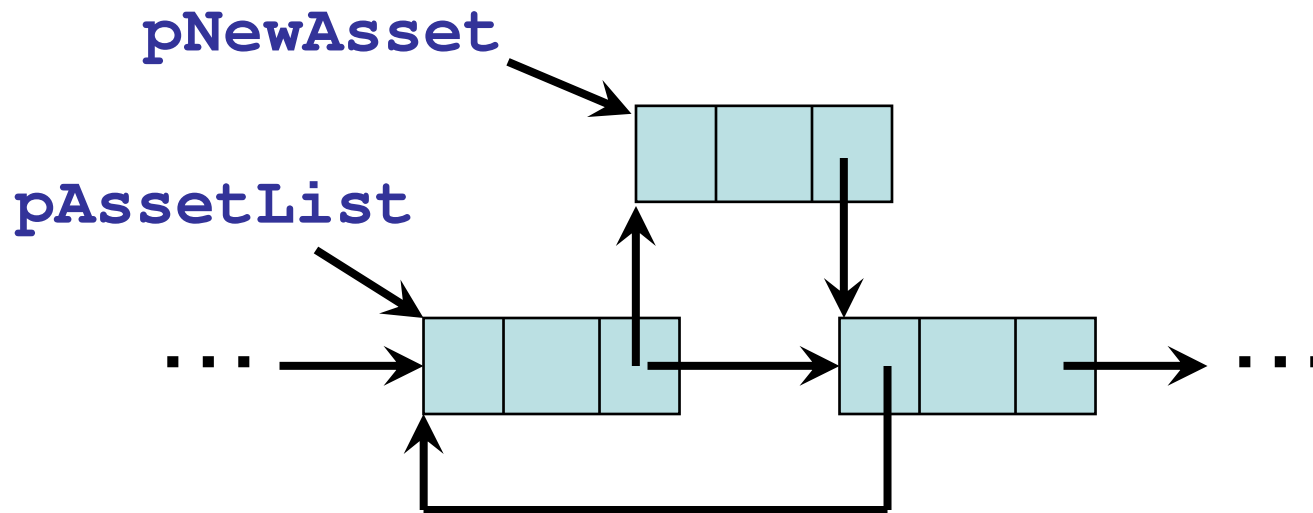
```
...
```

```
if (pAssetList->next != NULL) {
```

```
    pNewAsset->next = pAssetList->next
```

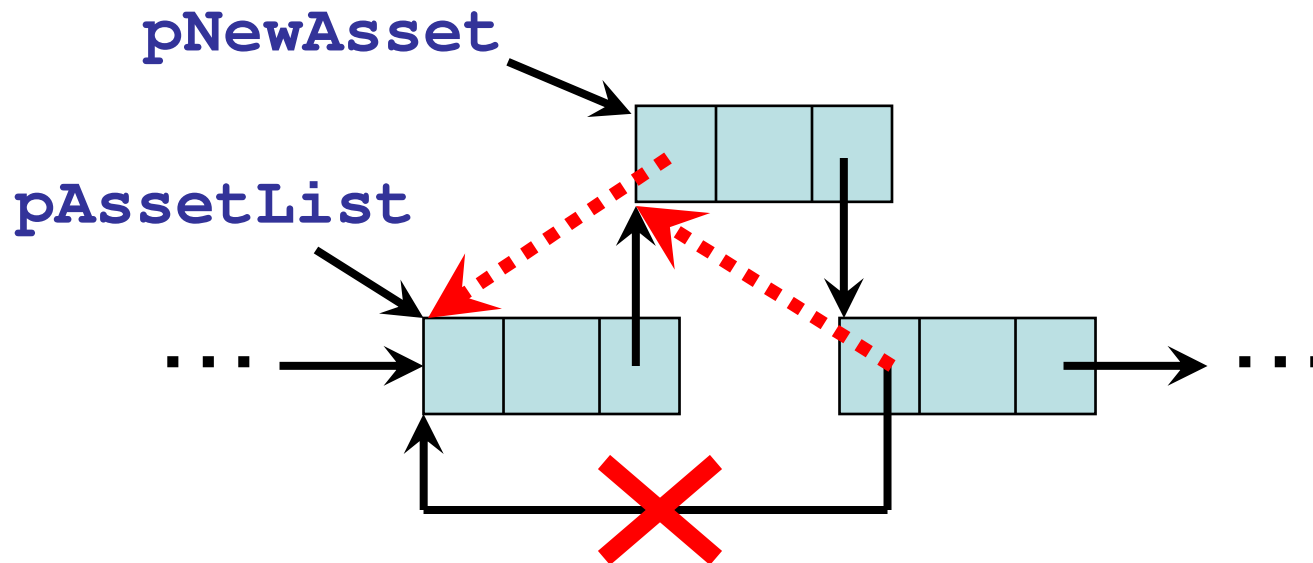
```
    pAssetList->next = pNewAsset
```

```
}
```



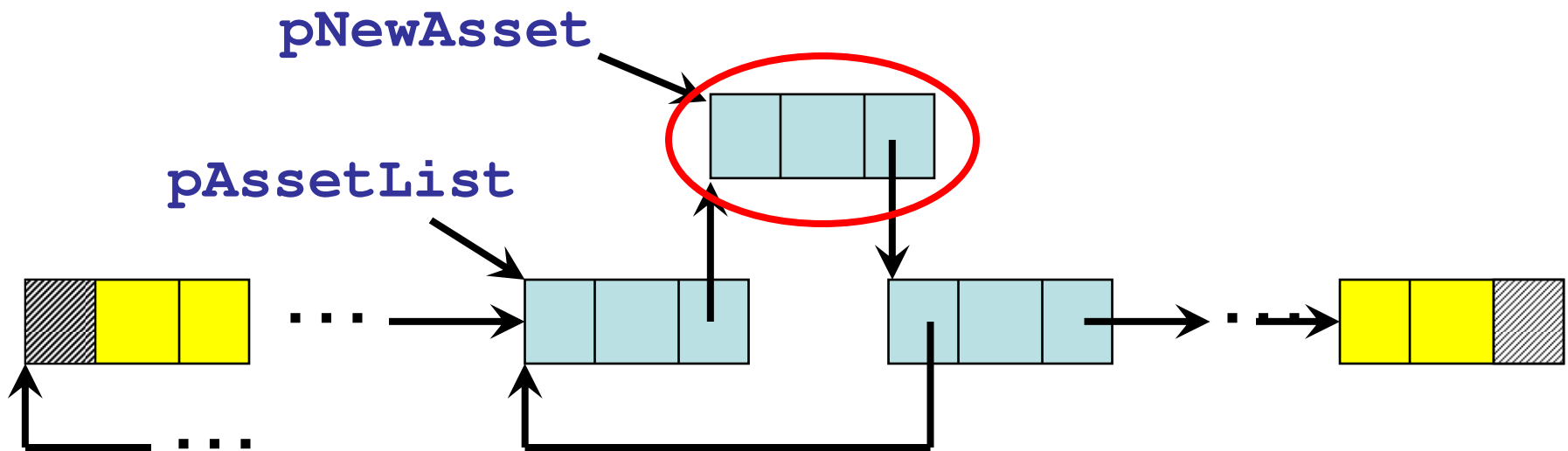
A motivating example

```
pNewAsset = Initialize(pAssetParams);  
...  
if (pAssetList->next != NULL) {  
    pNewAsset->next = pAssetList->next  
    pAssetList->next = pNewAsset  
}
```



Noteworthy points

- Violation of implicit data-structure invariant
 - Only extremal nodes must have indegree=1
- Malformed, but pointer-correct structure
 - Bug does not necessarily result in a crash



Key challenges

- Programmers rarely write down invariants for heap data structures
- Bugs may not be immediately apparent

Can we infer invariants for the heap and use them for bug detection?

Presenting HeapMD

- A tool to monitor the **health of the heap**

Highlights of results

**HeapMD found 40 bugs (31 new)
in 5 large commercial applications**

Talk outline

- Motivation
- Stability of the heap
- Application to bug-finding
- Related work and conclusion

Heap data

If this were true in practice, building large software would be untenable

1. Invisible

- No tangible representation

2. Can be arbitrarily modified

- Akin to self-modifying code

3. Can have arbitrary structure

- Akin to programming with gotos

Heap data

In practice, the heap has a simple, stable structure

1. Invisible

- No tangible representation

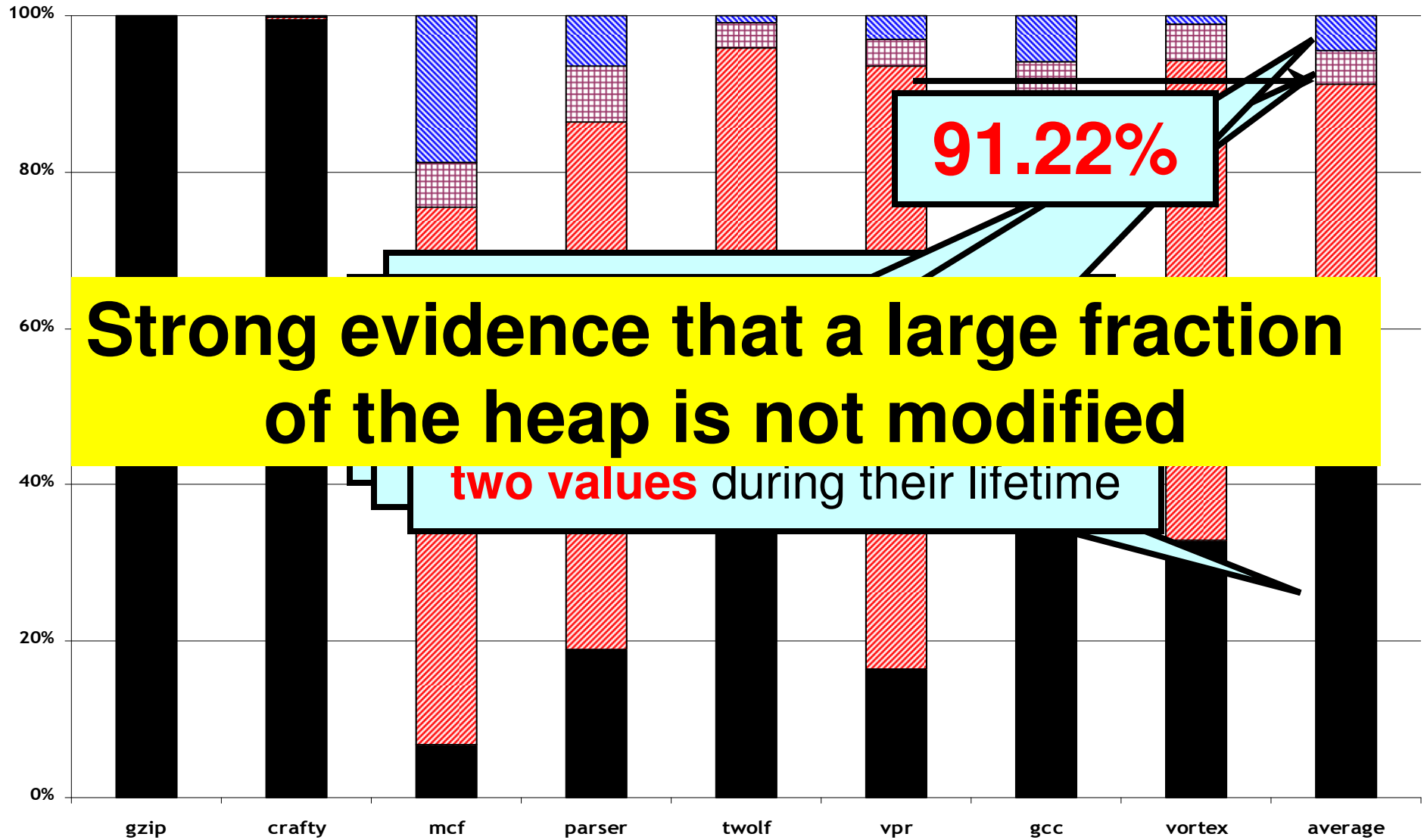
2. Can be arbitrarily modified

- Often only a small fraction is modified

3. Can have arbitrary structure

- In practice, structure is simple

Stability of pointer-valued locations



Strong evidence that a large fraction of the heap is not modified

two values during their lifetime

Simple structure of the heap

- Most data structures in large programs have low in- and out-degrees
 - Linked lists
 - Trees
 - Hash tables

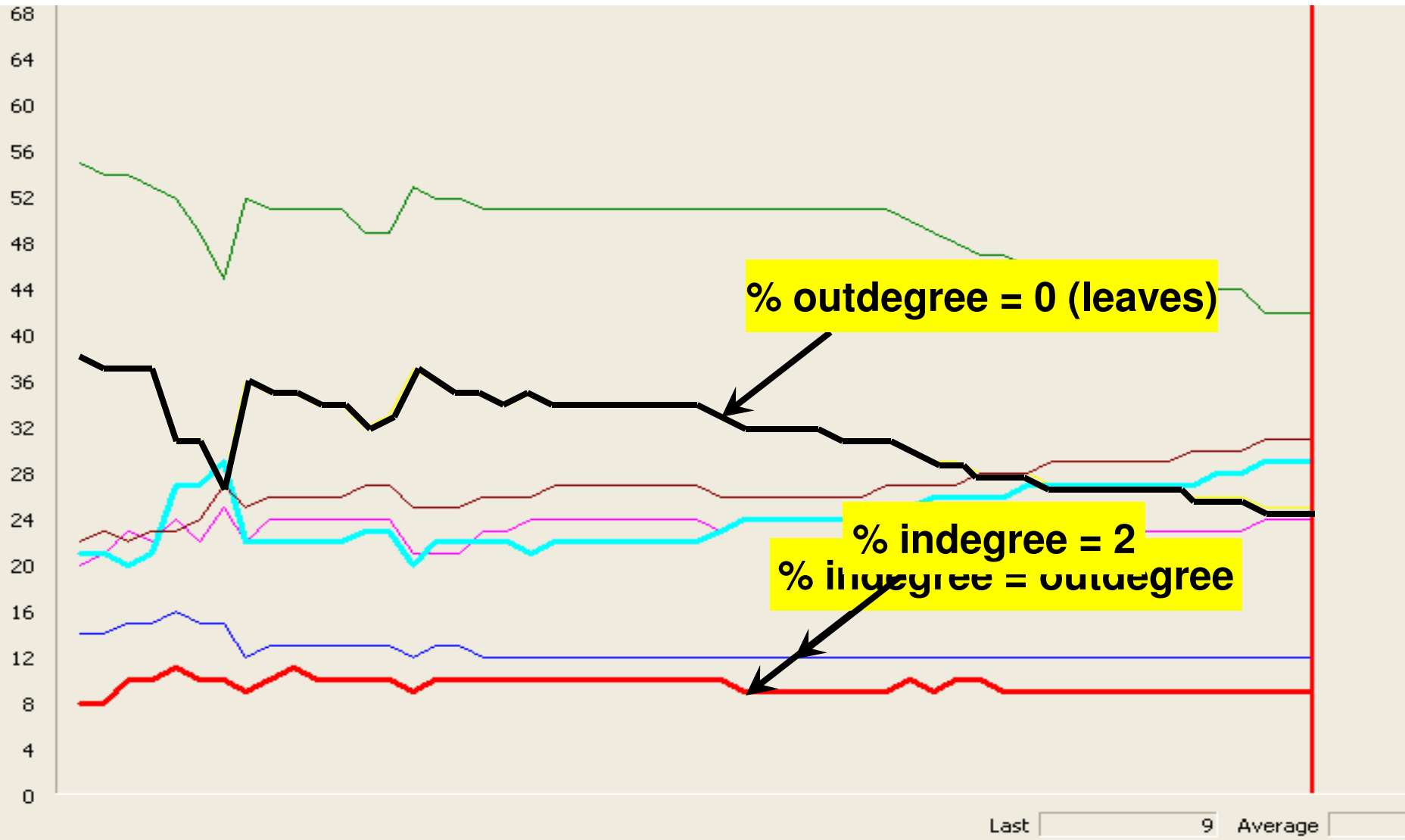
Can we quantify the simplicity and stability of the heap?

Simple metrics suffice

- % nodes with indegree = 0 (root nodes)
- % nodes with outdegree = 0 (leaf nodes)
- % nodes with indegree = 1
- % nodes with indegree = 2
- % nodes with outdegree = 1
- % nodes with outdegree = 2
- % nodes with indegree = outdegree

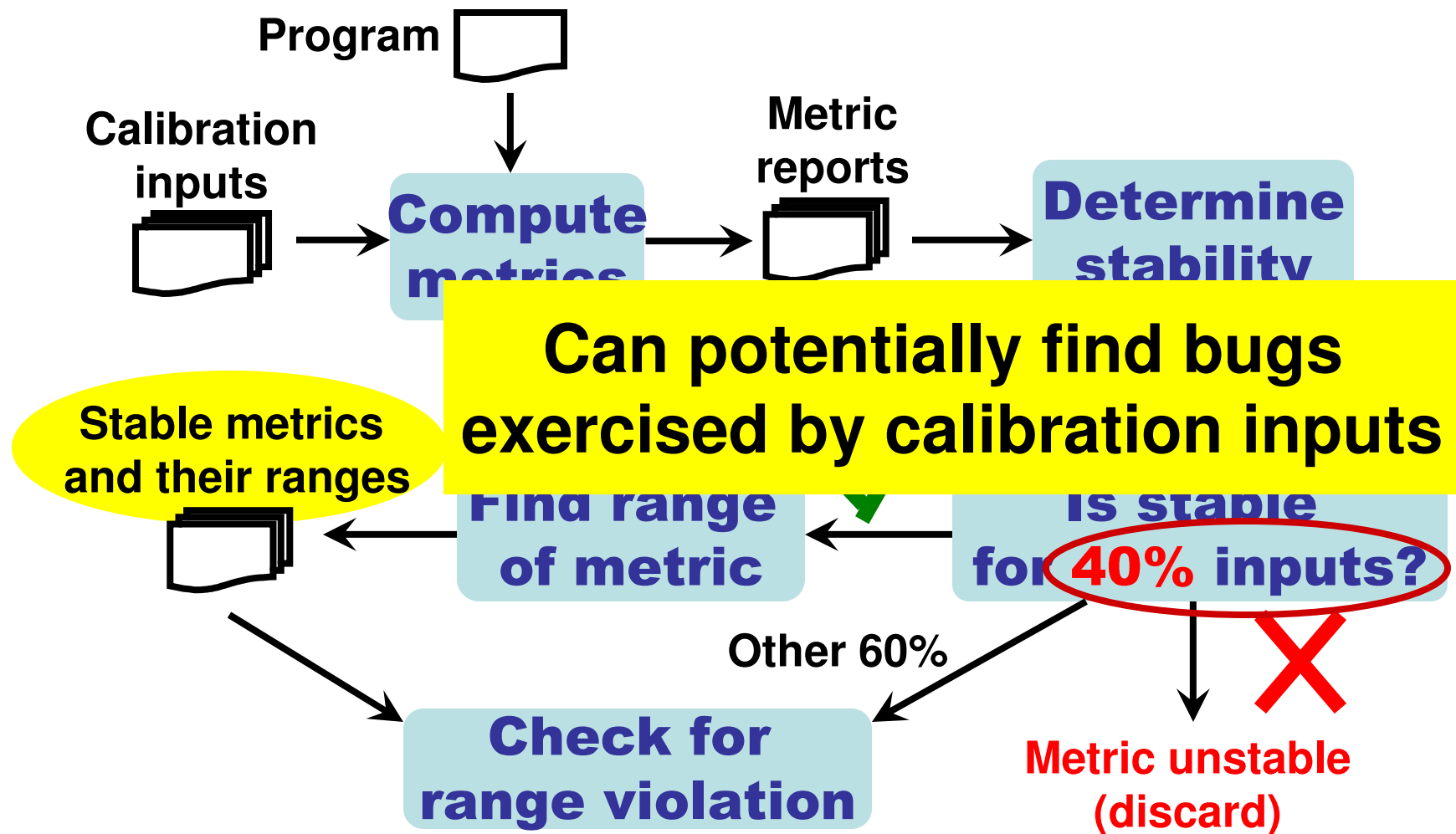
Gathering metrics: Setup

- Instrument program (x86) to track metrics
- Execute instrumented program on a set of inputs
- Gather at metric-computation points
 - Function entry points
 - Frequency is tunable: currently **1/100,000**



Color	Scale	Counter	Instance	Parent	Object	Computer
Red	1.000	% InDegEqualsOutDeg	Heap Co...	---	HeapC...	\\TECHFEST...
Green	1.000	% InDegree 1	Heap Co...	---	HeapC...	\\TECHFEST...
Blue	1.000	% InDegree 2	Heap Co...	---	HeapC...	\\TECHFEST...
Yellow	1.000	% Leafs	Heap Co...	---	HeapC...	\\TECHFEST...
Magenta	1.000	% OutDegree 1	Heap Co...	---	HeapC...	\\TECHFEST...
Cyan	1.000	% OutDegree 2	Heap Co...	---	HeapC...	\\TECHFEST...
Brown	1.000	% Roots	Heap Co...	---	HeapC...	\\TECHFEST...

Algorithm to find stable metrics



Stable metrics exist: SPEC

SPEC b/m	# Inputs	# Stable
twolf	3	6
% of vertices with Indegree = Outdegree Range = [14.2%, 17.2%]		
vpi	6	1
vortex	5	1
gzip	100	2
parser	100	3
gcc	100	2

Stable metrics exist: Commercial

Benchmark	# Inputs	# Stable
Multimedia	50	2
Interactive web-app.	50	2
PC-game (simulation)	50	2
PC-game (action)	50	1
Productivity	50	2

Extending the observation

Several degree-based metrics of the heap-graph remain stable as the heap evolves

In fact, we observed that ...

Stable heap metrics exist even across different development versions of a program

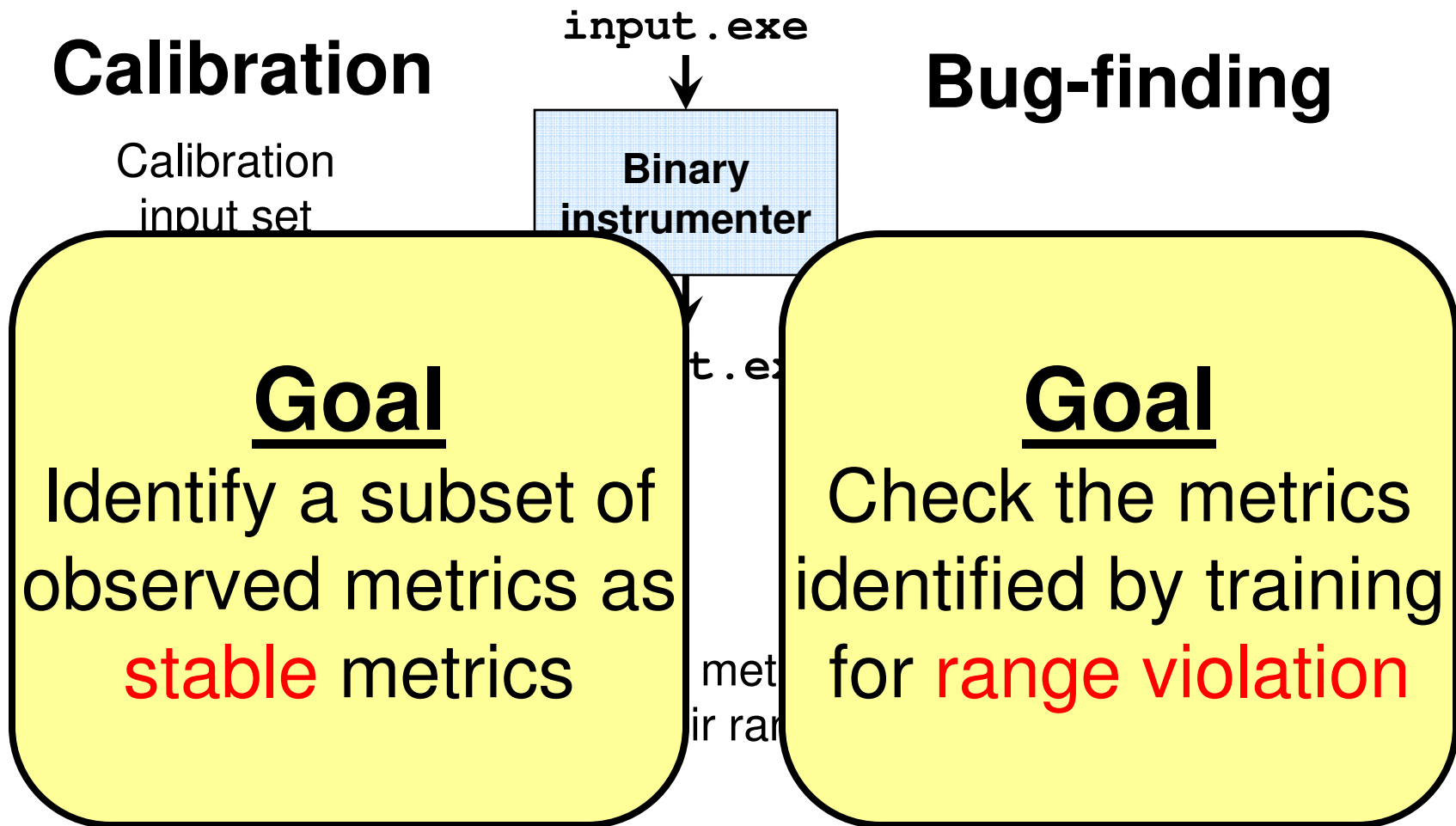
Metric stability across versions

Benchmark	# Versions	# Inputs	# Stable
Multimedia	5	10	2
Interactive web-app.	5	10	2
PC-game (simulation)	5	10	2
PC-game (action)	5	10	1
Productivity	5	10	2

Talk outline

- Motivation
- Stability of the heap
- **Application to bug-finding**
- **Related work and conclusion**

Architecture of HeapMD



Finding bugs using HeapMD

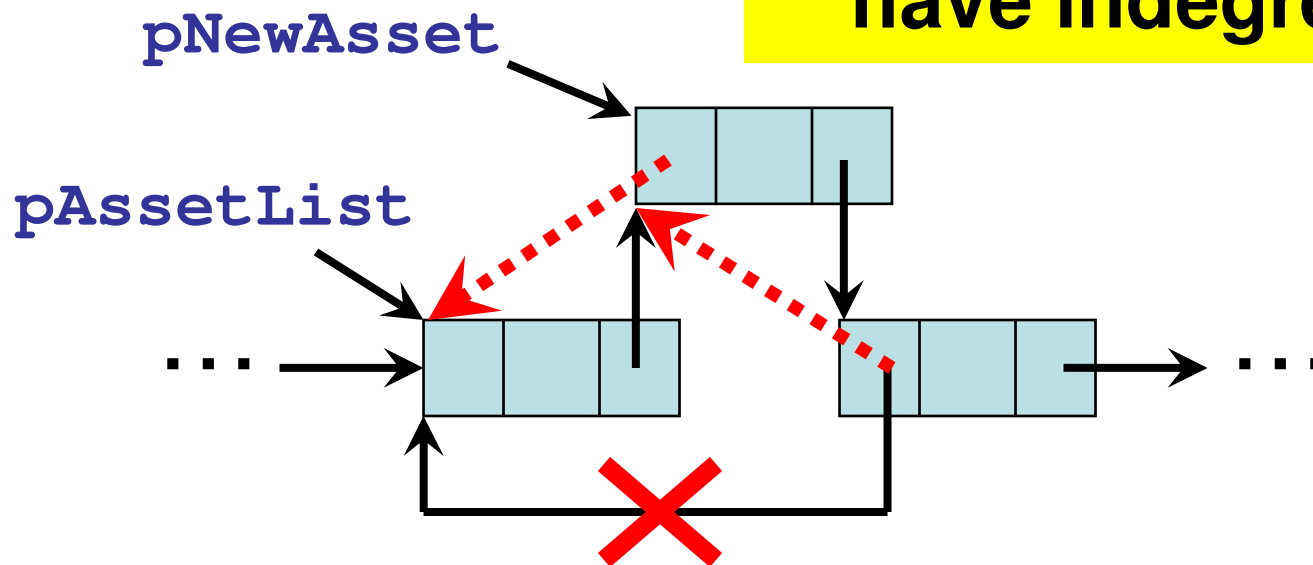
Key intuition
Range violation → Likely bug



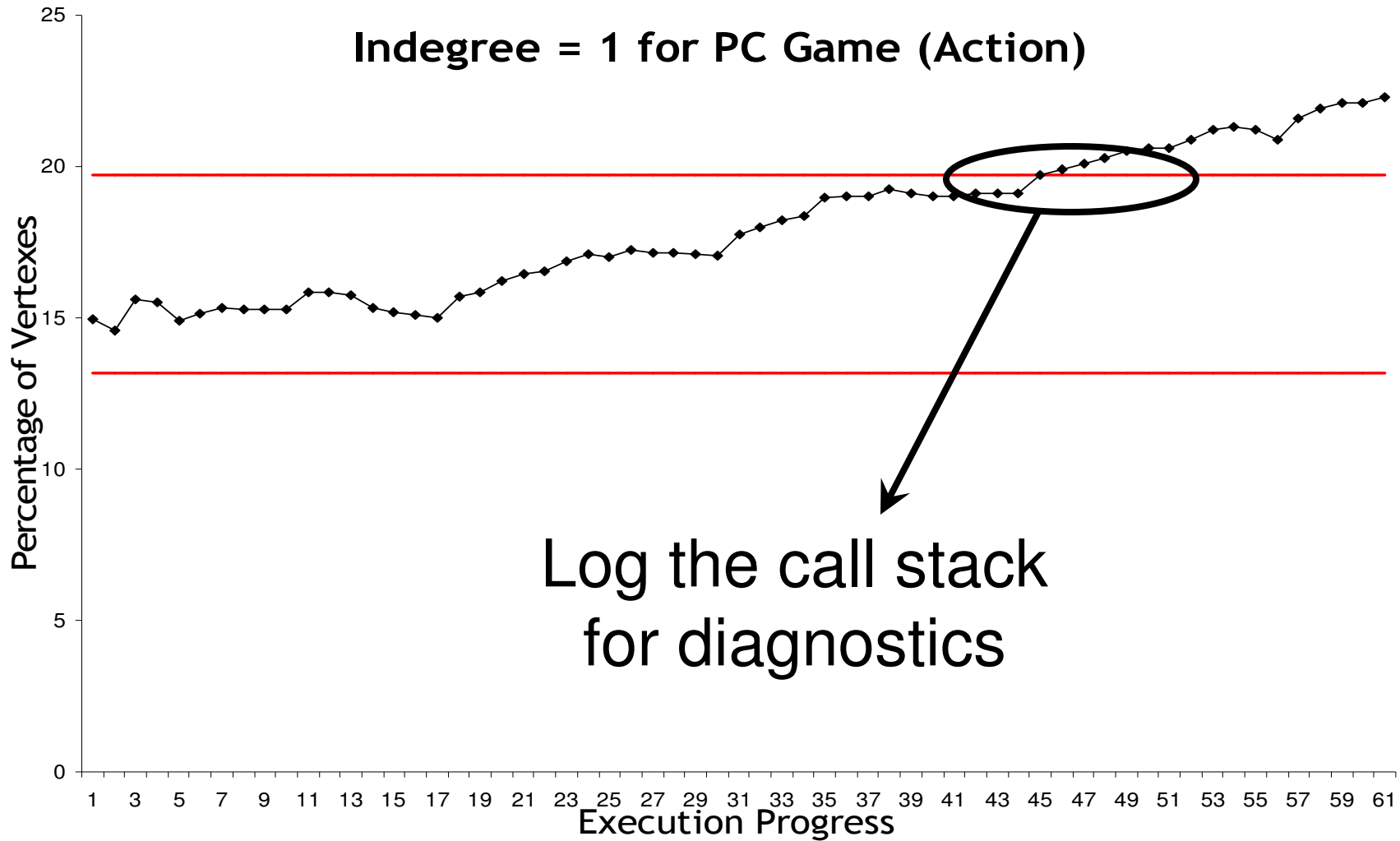
Recall our motivating example

```
pNewAsset = Initialize(pAssetParams);  
...  
if (pAssetList->next != NULL) {  
    pNewAsset->next  
    pAssetList->nex  
}
```

Violated invariant
Only extremal nodes
have indegree=1



Range violation for this example

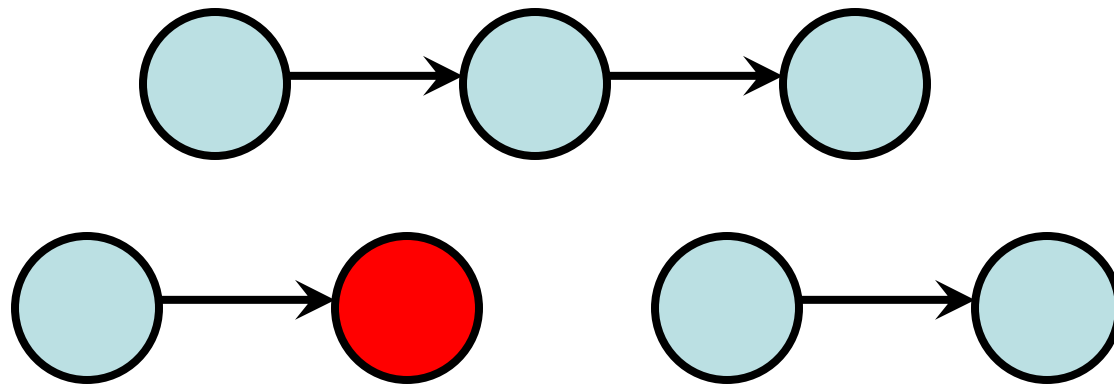


Summary of bugs found

Benchmark	# Bugs	# New bugs	# Invariant violations
Multimedia	8	6	3
Interactive web-app	10	6	5
PC-game (simulation)	9	6	2
PC-game (action)	8	8	3
Productivity	5	5	4
Total	40	31	17

Kinds of bugs found (1)

- Erroneous insertion into linked list



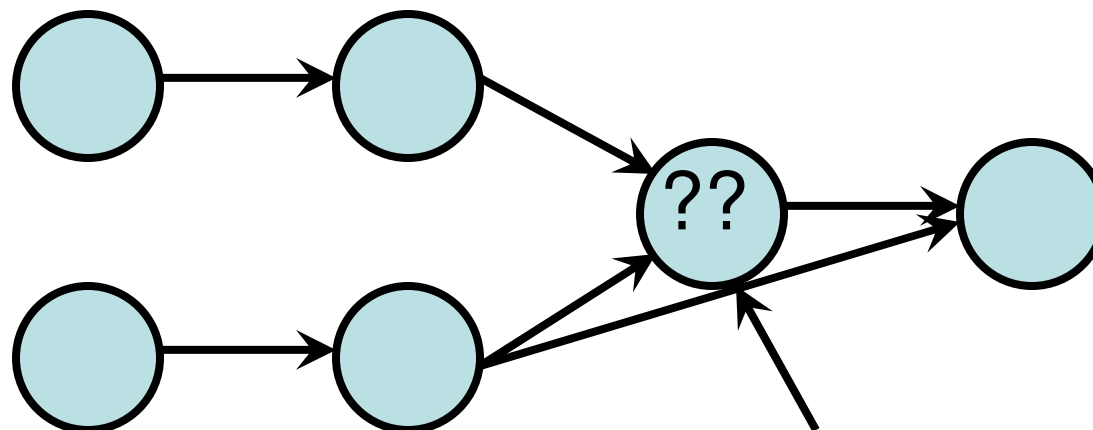
% vertices with indegree = 0, 1

% vertices with outdegree = 0, 1

% vertices with indegree = outdegree

Kinds of bugs found (2)

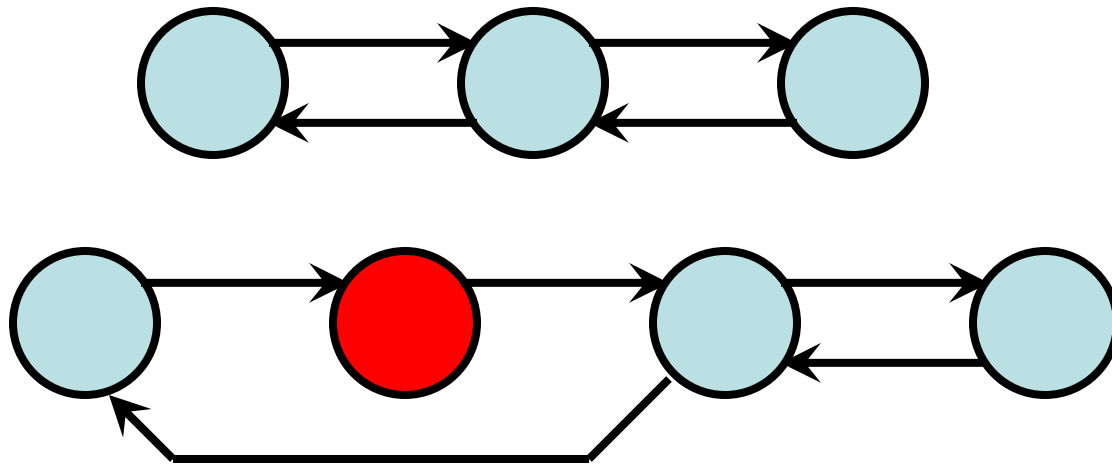
- Shared data structure manipulation errors



Delete
% vertices with outdegree = 0
% vertices with indegree = 2
% vertices with indegree = outdegree

Kinds of bugs found (3)

- Data structure invariants

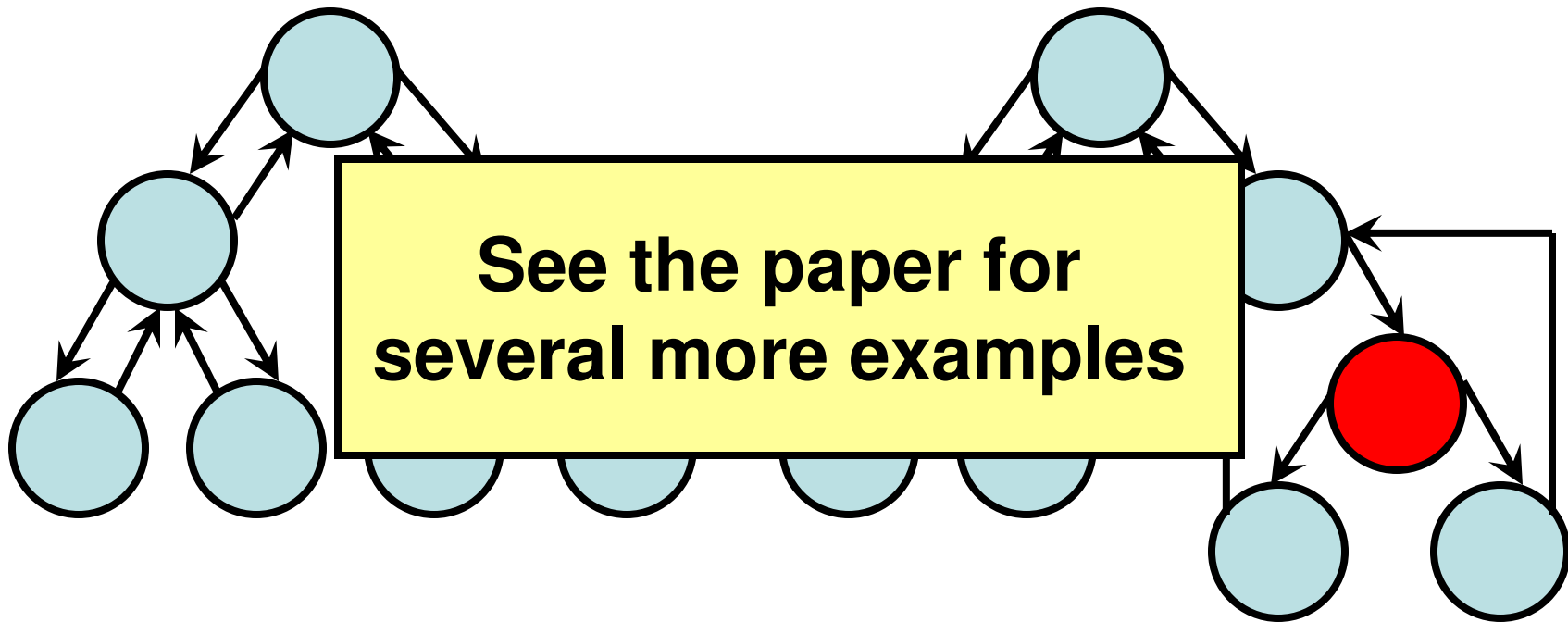


% vertices with outdegree = 1, 2

% vertices with indegree = 1, 2

Kinds of bugs found (3)

- Data structure invariants



% vertices with indegree = outdegree

% vertices with indegree = 1

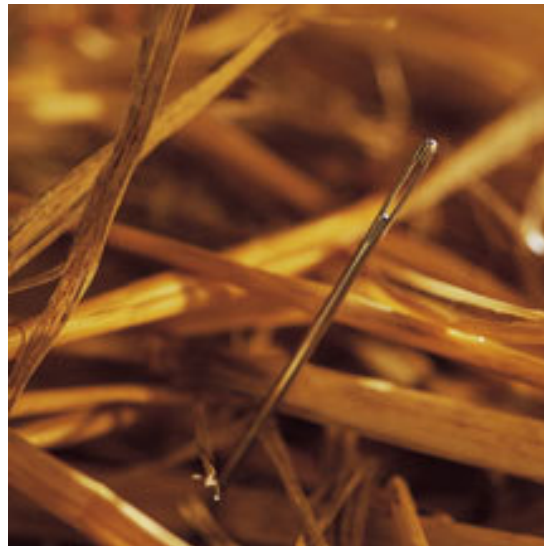
Comparison with SWAT

[Chilimbi and Hauswirth, ASPLOS'04]

Benchmark	SWAT		HeapMD		
	Leaks	#FP	Leaks	#FP	Other bugs
Multimedia					
Interactive web-app.					
PC-Game (simulation)					

Characteristics of bugs found

- **Systemic bugs**: Repeated often enough to affect heap-graph metric
- HeapMD cannot find “one-off” bugs
 - Temporary data structure invariant violations



False positives and negatives

- In our experiments: **0** false positives
 - Metrics are computed for the whole heap rather than per data structure
 - Anomaly detector only looks for range violations, not stability
- **Comes at a cost: False negatives**
 - Bugs with no effect on heap-graph metrics
 - Bugs that affect heap-graph metrics, but do not violate calibrated range

Talk outline

- Motivation
- Stability of the heap
- Application to bug-finding
- **Related work and conclusion**

Related work

- Tools for special classes of bugs
 - Purify, Valgrind, SWAT, ...
 - Not built for detecting invariant violations
- Tools to find invariants
 - Daikon [Ernst], DIDUCE [Hangal and Lam, ICSE 2002]
 - HeapMD complements these in the types of invariants found
- Shape analysis algorithms and tools
 - Can find invariant violations given a correctness specification

Points to take home

**Stable heap-graph metrics exist.
Their ranges serve as invariants**

Range violation → Bug likely exercised

**Can find non-crashing bugs
e.g., invariant violations**

Questions?

HeapMD: Identifying Heap-based Bugs using Anomaly Detection

Trishul M. Chilimbi

Microsoft Research

Redmond, WA

trishulc@microsoft.com

Vinod Ganapathy

University of Wisconsin

Madison, WI

vg@cs.wisc.edu