PerfGuard: Binary-Centric Application Performance Monitoring in Production Environments

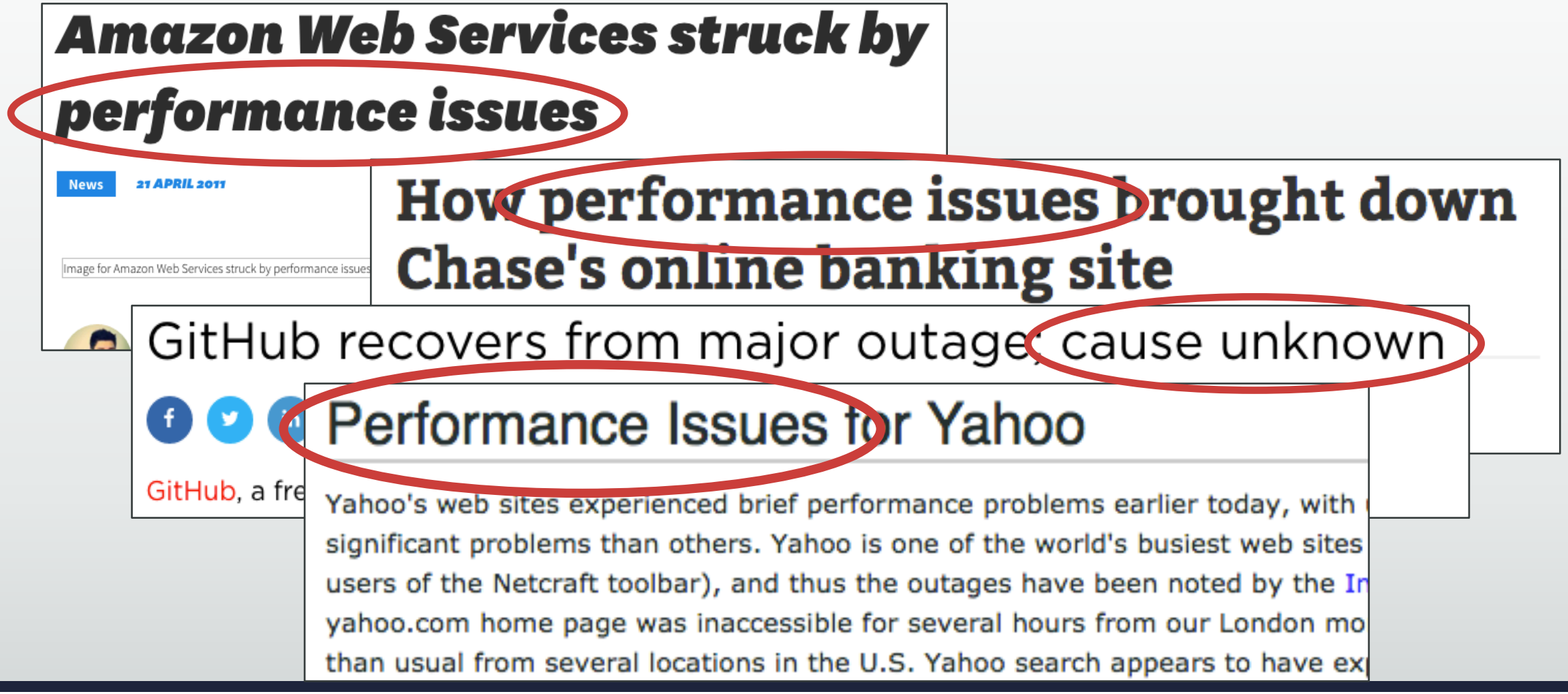
Chung Hwan Kim, Junghwan Rhee[*], Kyu Hyung Lee[+], Xiangyu Zhang, Dongyan Xu

PURDUE UNIVERSITY

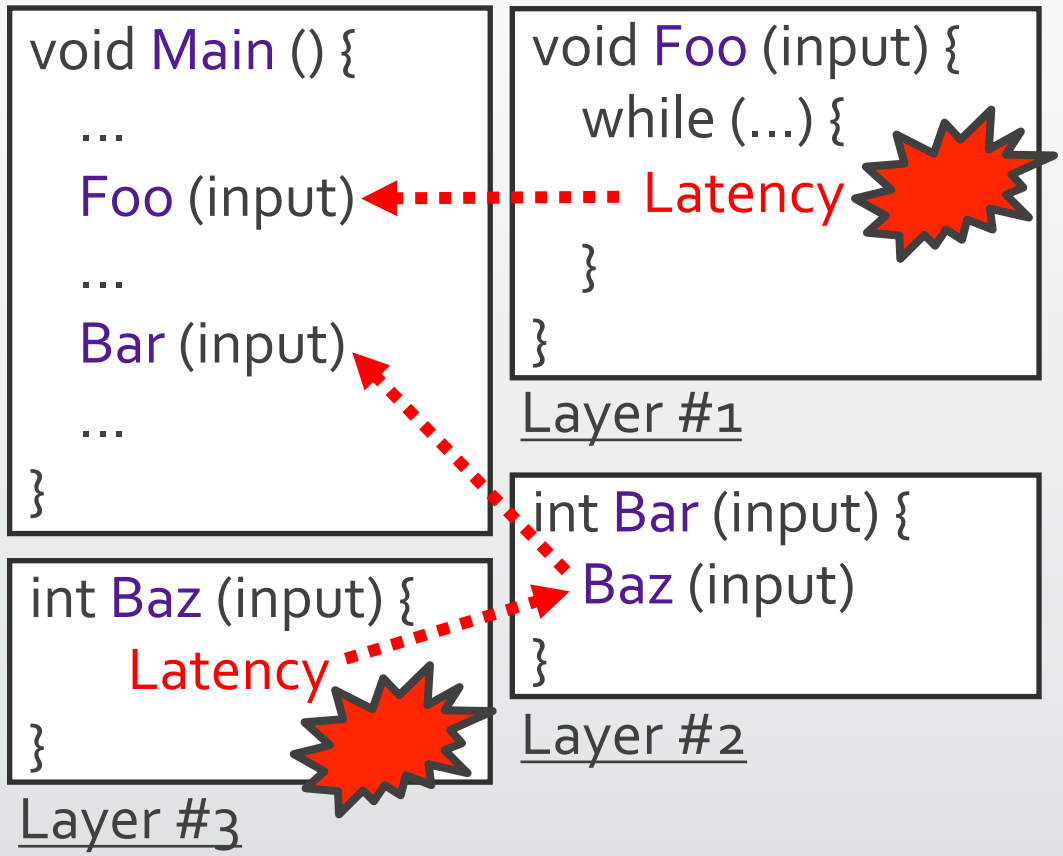
[*] NEC Laboratories America
Relentless passion for innovation

[+] The University of Georgia
1785

Performance Problems



Performance Diagnosis During Development



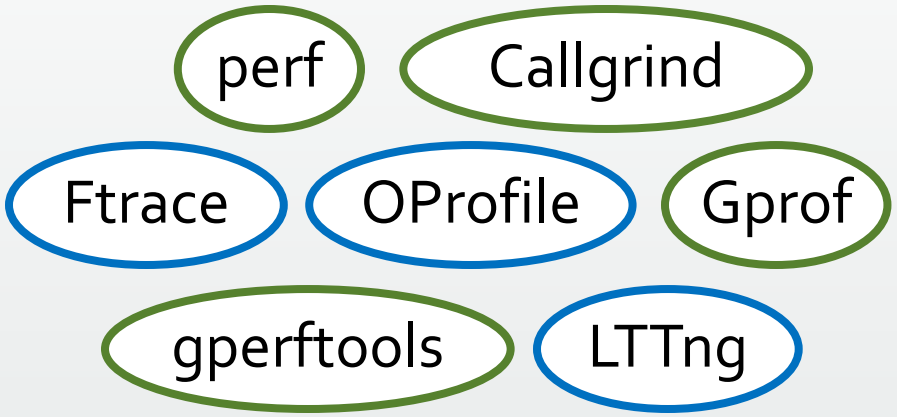
- Complex dependencies and layers [PLDI '12]
- Various usage scenarios
- Limited testing environments

Performance diagnosis during production?

Performance Diagnosis in Production

- Software users do not have:
  - Source code
  - Development knowledge
- But, desire to analyze performance problems [SIGMETRICS '14]
  - Many users are service providers
  - 3rd-party components

- Profilers and tracers:



- CPU usage sampling
  - Constant overhead
  - Blind to program semantics
  - Sampling frequency

Performance Diagnosis in Production

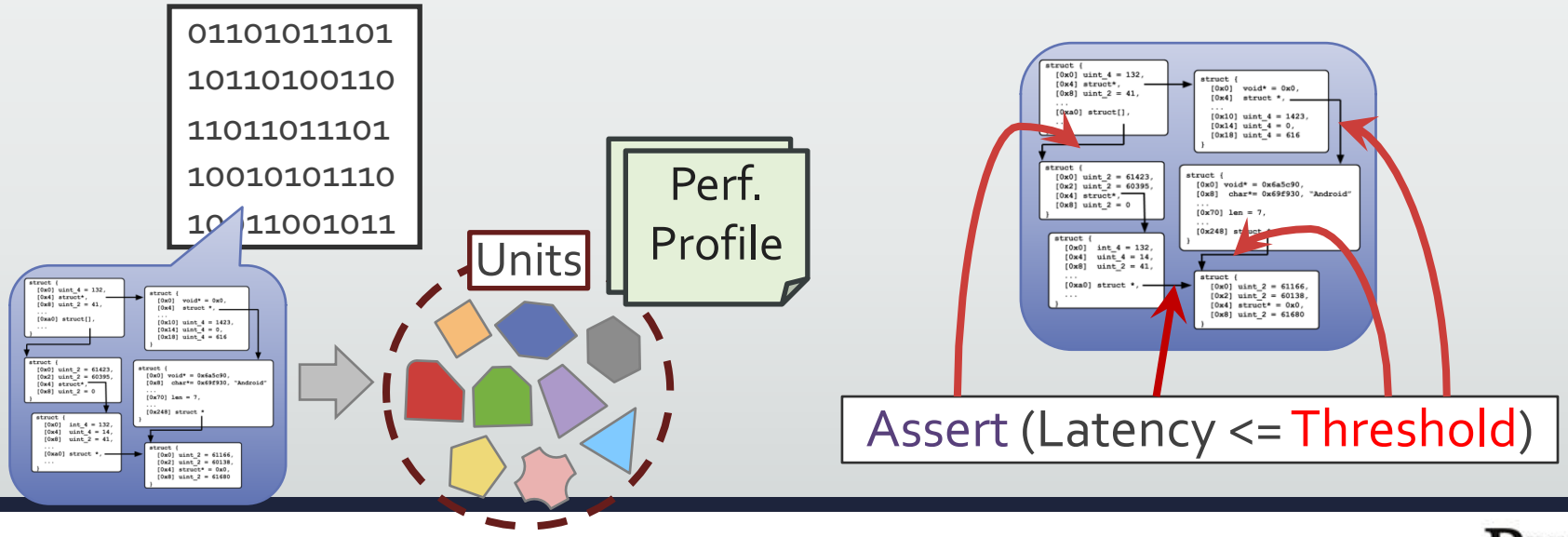
- PerfTrack: Microsoft products only
- Application Performance Management (APM)
  - Limited # of pre-instrumented programs
  - Manual instrumentation with APIs
- Required: Source code and development knowledge

Automated Perf. Diagnosis in Production?

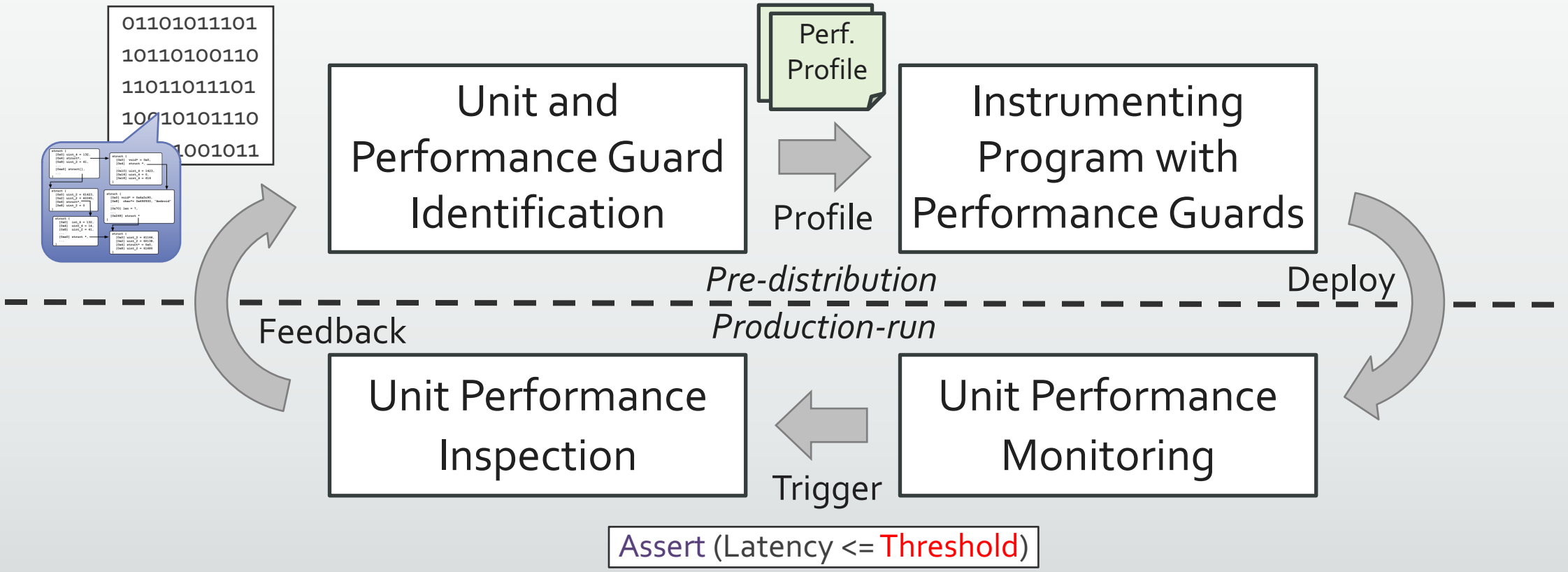
- Performance diagnosis without source code and development knowledge?
- At what granularity should we measure performance?
- When and where should we check performance?
- How can we determine if program is too slow?

Key Ideas

- Extract "hints" from program binaries through dynamic analysis
- Use the hints to identify individual operations (*units*)
- Generate and inject performance checks



PerfGuard: Binary-Centric Performance Check Creation and Injection



Unit Identification

- *Unit* := One iteration of event processing loop [NDSS '13]

```
UiThread (...) {
  ...
  while (...) {
    e = GetEvent (...)
    DistpatchEvent (e, callback)
  } // end while
  ...
} // end UiThread
```

- Type I: UI programs

```
ListenerThread (...) {
} // end ListenerThread
```

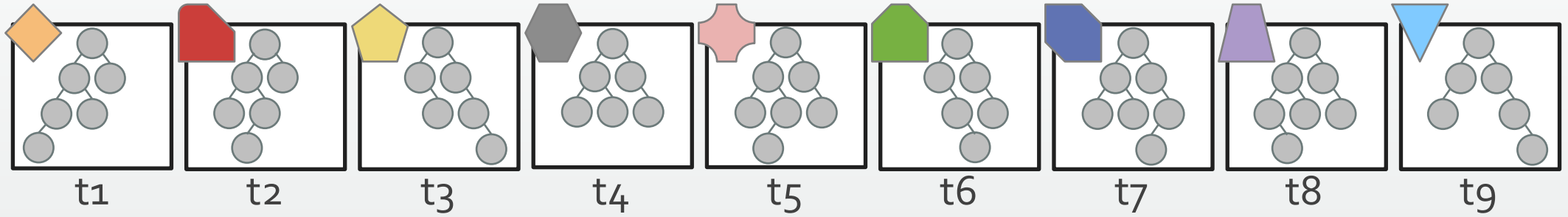
```
} // end WorkerThread
```

1) Most large-scale apps are event-driven
2) Small number of event processing loops

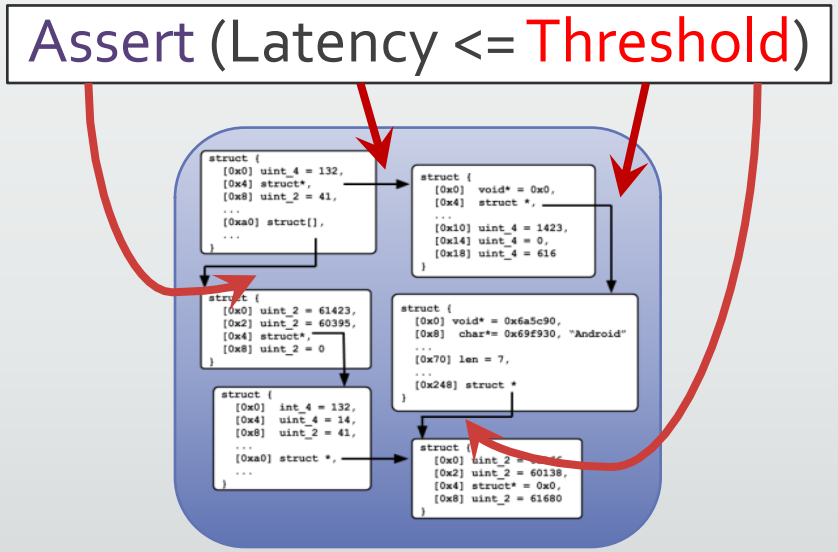
- Type II: Server programs

Unit Classification Based on Control Flow

- Units with different call trees have distinct performance



- Threshold estimation: based on time samples of unit groups
  - Average of 11% deviation in top 10 costly unit groups

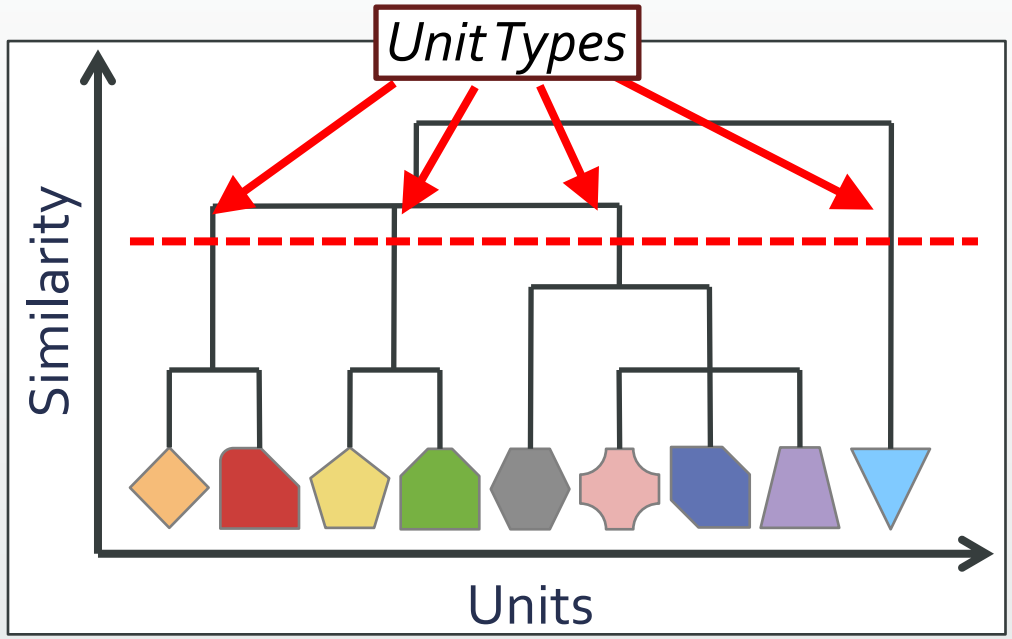


Unit Clustering

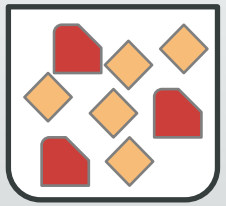
- Hierarchical clustering
  - Unit distance:

$$\frac{\max(|p_i|,|p_j|)-|LCS(p_i,p_j)|}{\max(|p_i|,|p_j|)}$$

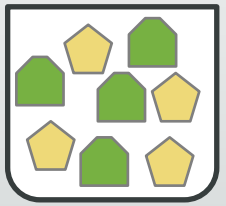
  - *Unit type*:
    Set of clustered units



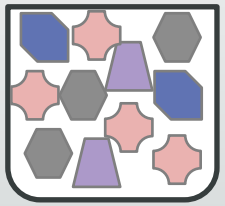
Unit Type X



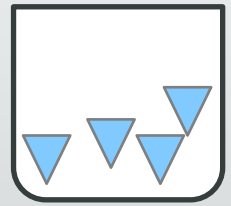
Unit Type Y



Unit Type W

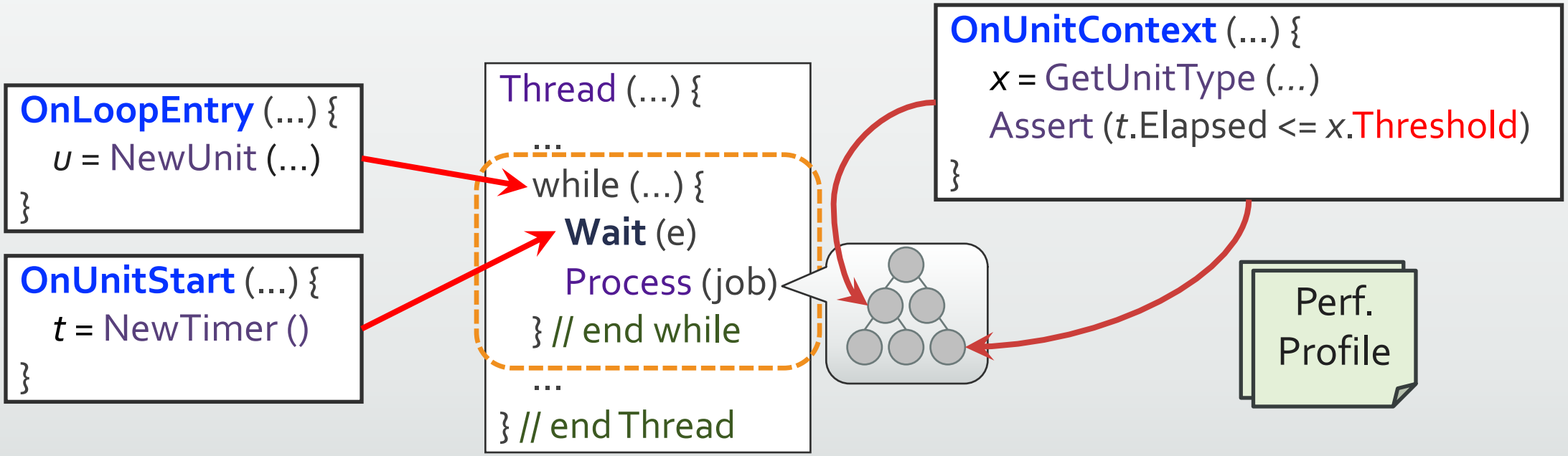


Unit Type Z



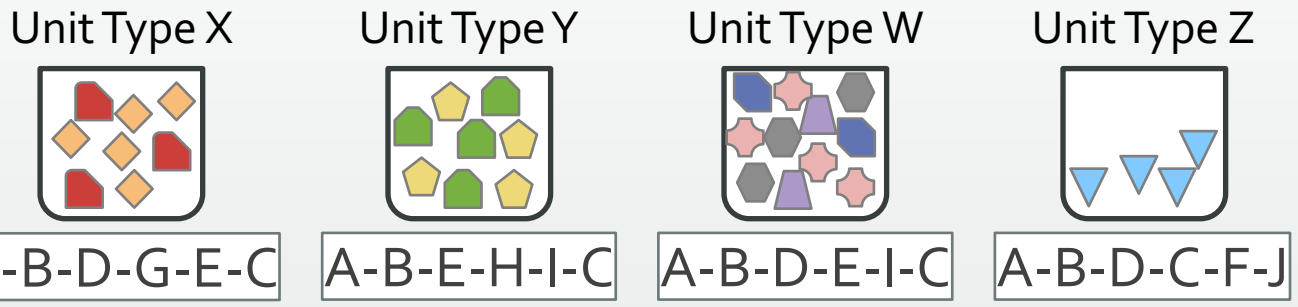
Performance Guard Generation

- 3 shared library functions
- Input: unit performance profile



How to Recognize Unit Types at Run-Time

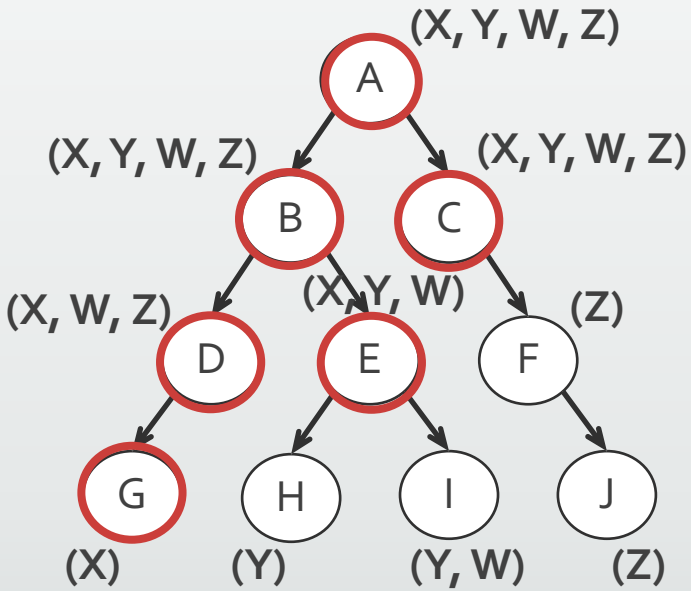
- Unit type election: Mark # of total occurrences



- Example: Unit Type X

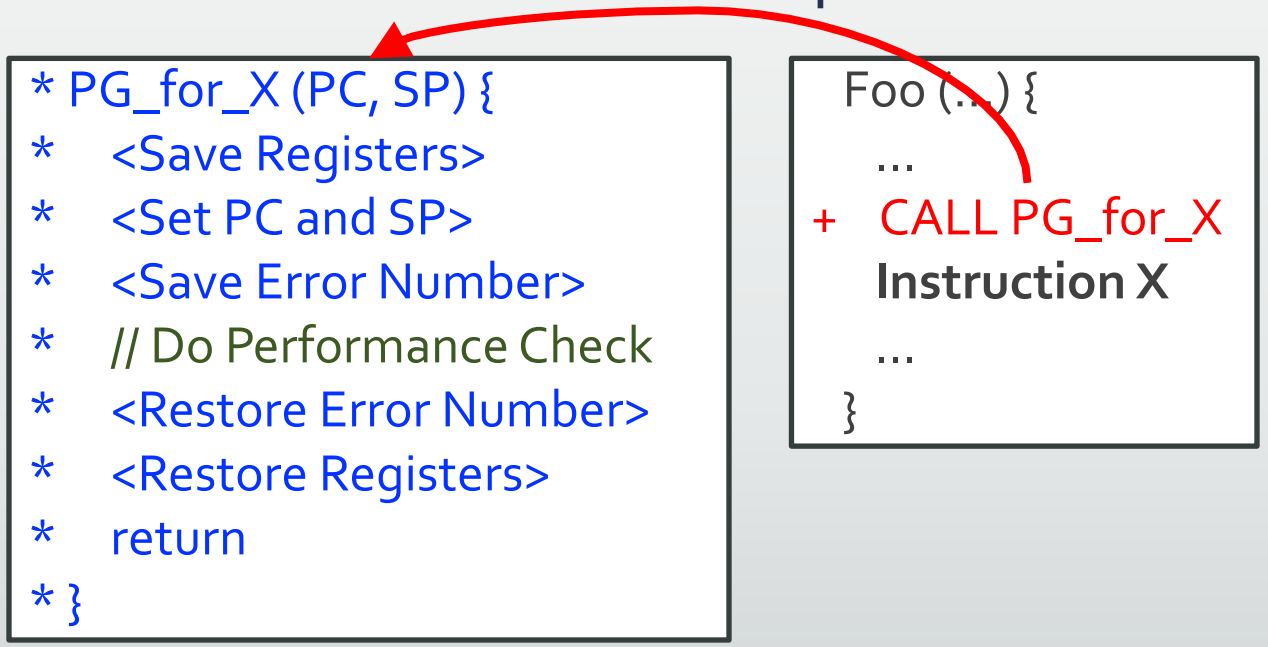
	A	B	D	G	E	C
X:	**1**	**2**	**3**	**4**	**5**	**6**
Y:	**1**	**2**	2	2	3	4
W:	**1**	**2**	**3**	3	4	5
Z:	**1**	**2**	**3**	3	3	4
Unit Type Candidates	(X, Y, W, Z)	(X, Y, W, Z)	(X, W, Z)	(X)	(X)	(X)

time



Binary Code Instrumentation

- Modified x86 Detours [USENIX WinNT '99]
  - Arbitrary instruction instrumentation
- NOP insertion using BISTRO [ESORICS '13]
- Original program state preserved



Evaluation

- Diagnosis of real-world performance bugs

Program Name	Bug ID	Root Cause Binary	Unit Call Trees	Unit Call Paths	Unit Functions	Inserted Unit Perf. Guards	Unit Thres-hold (ms)
Apache	45464	Internal Library	8	17,423	635	138	9,944
MySQL Client	15811	Main Binary	24	255,126	106	13	997
MySQL Server	49491	Main Binary	8	270,454	980	303	2,079
						115	122
						127	109
						15	110
7-Zip File Manager	54	Main Binary	6	26,842	143	120	101
Notepad++	2909745	Main Binary	16	352,831	711	370	6,797
ProcessHacker	3744	Main Binary	1	47,910	86	23	3,104
ProcessHacker	5424	Plug-in	32	62,136	69	19	10

1-32 Distinct Call Trees,
4,000-352,000 Call Paths, and
65-980 Functions

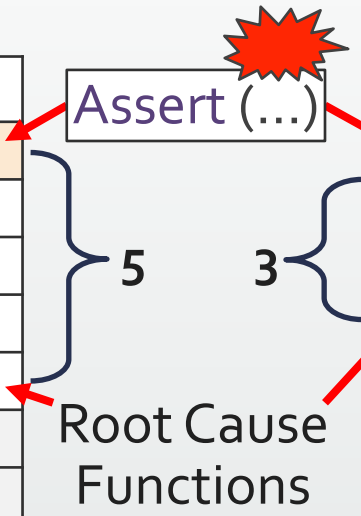
Average of:
124 Insertions, and
2,337 ms
Per Buggy Unit

Use Case: Unit Call Stack Traces

- Case 1: Apache HTTP Server

Unit Call Stack	
T	libapr-1.dll!convert_prot
	libapr-1.dll!more_finfo
	libapr-1.dll!apr_file_info_get
	libapr-1.dll!resolve_ident
R	**libapr-1.dll!apr_stat**
-	mod_dav_fs.so!dav_fs_walker
-	mod_dav_fs.so!dav_fs_internal_walk
-	mod_dav_fs.so!dav_fs_walk
-	…
-	libhttpd.dll!ap_run_process_connection
-	libhttpd.dll!ap_process_connection
-	libhttpd.dll!worker_main

Performance Bug: Apache 45464



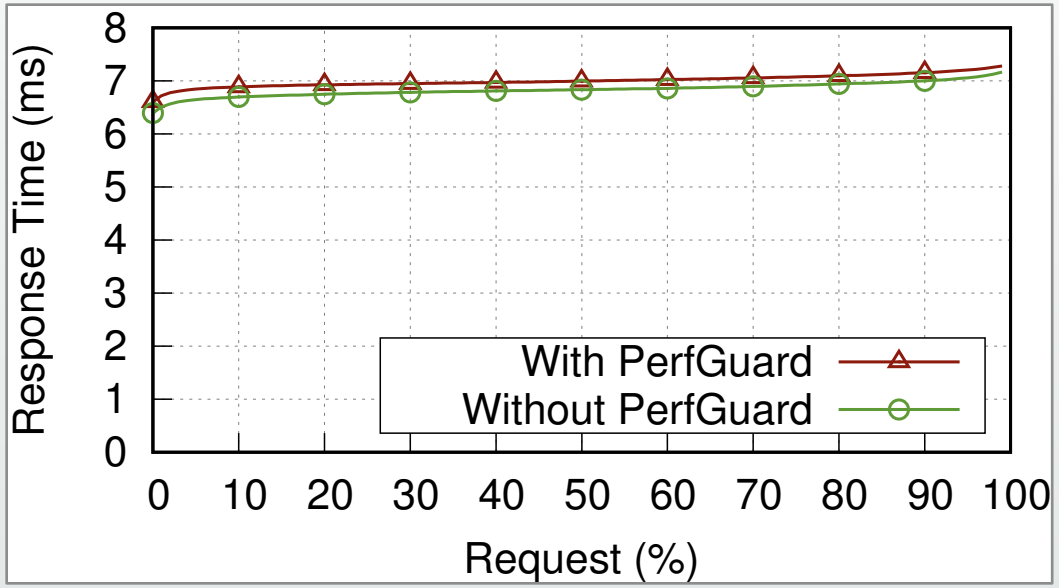
- Case 2: 7-Zip File Manager

Unit Call Stack	
T	7zFM.exe!NWindows::NCOM::MyPropVariantClear
	7zFM.exe!GetItemSize
R	**7zFM.exe!Refresh_StatusBar**
-	7zFM.exe!OnMessage
-	7zFM.exe!NWindows::NControl::WindowProcedure
-	USER32.dll!InternalCallWinProc
-	…
-	USER32.dll!DispatchMessageWorker
-	USER32.dll!DispatchMessageW
-	7zFM.exe!WinMain

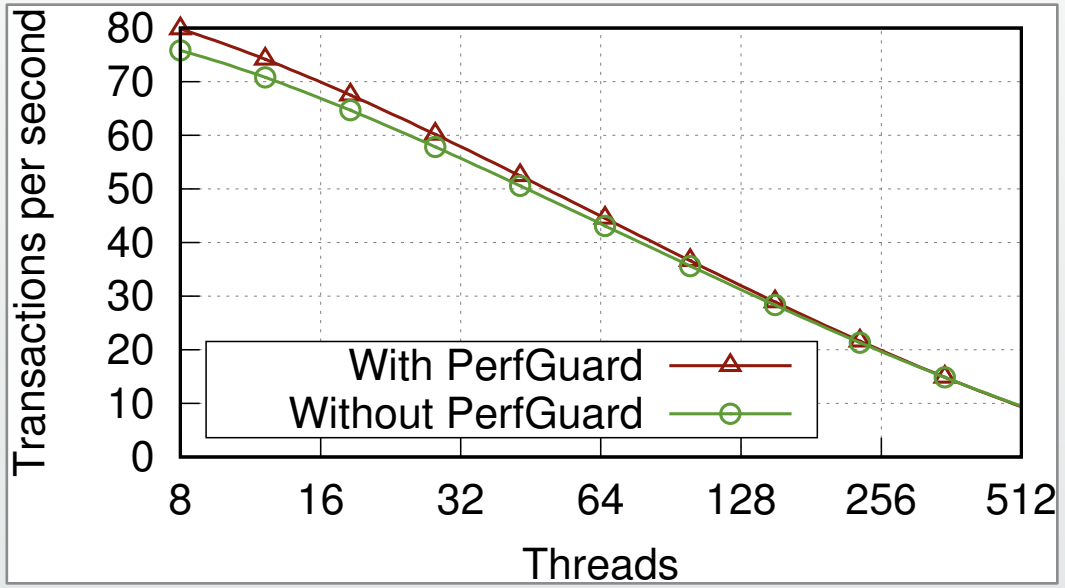
Performance Bug: 7-Zip S3

Performance Overhead

- ApacheBench & SysBench: Overhead **< 3%**



- Apache HTTP Server



- MySQL Server

Related Works

C. H. Kim, J. Rhee, H. Zhang, N. Arora, G. Jiang, X. Zhang, and D. Xu. IntroPerf: Transparent Context-Sensitive Multi-Layer Performance Inference Using System Stack Traces. In Proc. ACM SIGMETRICS 2014.

S. Han, Y. Dang, S. Ge, D. Zhang, and T. Xie. Performance Debugging in the Large via Mining Millions of Stack Traces. In Proc. ICSE 2012.

A. Nistor, P.-C. Chang, C. Radoi, and S. Lu. CARAMEL: Detecting and Fixing Performance Problems That Have Non-intrusive Fixes. In Proc. ICSE 2015.

A. Nistor, L. Song, D. Marinov, and S. Lu. Toddler: Detecting Performance Problems via Similar Memory-access Patterns. In Proc. ICSE 2013.

X. Xiao, S. Han, D. Zhang, and T. Xie. Context-sensitive Delta Inference for Identifying Workload-dependent Performance Bottlenecks. In Proc. ISSTA 2013.

Y. Liu, C. Xu, and S.-C. Cheung. Characterizing and Detecting Performance Bugs for Smartphone Applications. In Proc. ICSE 2014.

L. Ravindranath, J. Padhye, S. Agarwal, R. Mahajan, I. Obermiller, and S. Shayandeh. AppInsight: Mobile App Performance Monitoring in the Wild. In Proc. OSDI 2012.

Conclusion

- PerfGuard enables diagnosis of performance problems **without** source code and development knowledge
- Unit-based performance profiling allows targeting a **general scope** of software
- Automatically detects performance problems with **low run-time overhead** (< 3%)

Thank you!
Questions?

Chung Hwan Kim
chungkim@cs.purdue.edu

PURDUE UNIVERSITY CERIAS