Outline

- Introduction
- State-of-the-art Forensic Methods
  - OS level
  - Hypervisor level
- Hardware-based Workload Forensics
  - Process Reconstruction
- Experimental Results
  - Setup
  - Result & Overhead
- Summary

Introduction

- Motivation
  - Vast amount of sensitive information is stored, processed and communicated in electronic form
  - Intensified malicious efforts
    → unauthorized access
  - Retroactive investigation needed

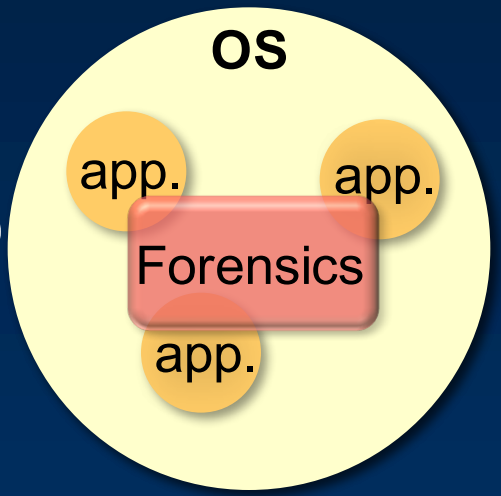
- Workload Forensics
  - Collect data related to past execution of computer programs
  - Analyze data to understand and/or reconstruct corresponding events

Outline

- Introduction
- State-of-the-art Forensic Methods
  - OS level
  - Hypervisor level
- Hardware-based Workload Forensics
  - Process Reconstruction
- Experimental Results
  - Setup
  - Result & Overhead
- Summary

OS-level Forensic Methods

- Forensic module resides at the same level with applications/OS kernel
- Signature comparison
  - Memory image
  - Commercial products (i.e. EnCase, FTK, etc.)
- Program behavior modeling
  - System call pattern
  - Involve machine learning/statistics

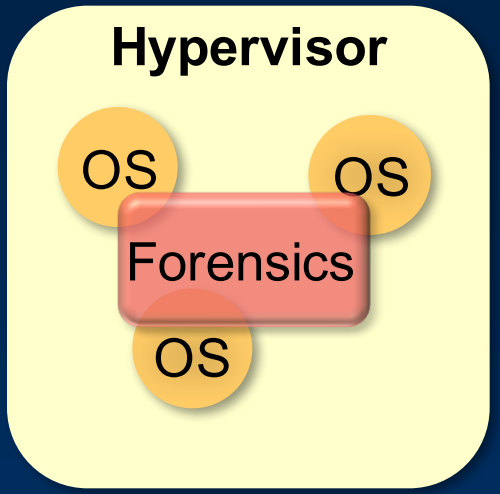


While OS-level Forensic methods benefit from semantic-rich information, they are vulnerable to software attacks at the same level!

C. Kolbitsch, et al. "Effective and efficient malware detection at the end," USENIX, 2009

Hypervisor-level Forensic Methods

- Forensic module resides at Hypervisor level
- Hypervisor
  - Virtualization for OS
  - Isolated management core provides better security
- Bridge semantic gap
  - Process → dedicated addr. space & page table
  - Page table base addr. (CR3 in x86) → process
- Similar methods as at OS-level can be performed



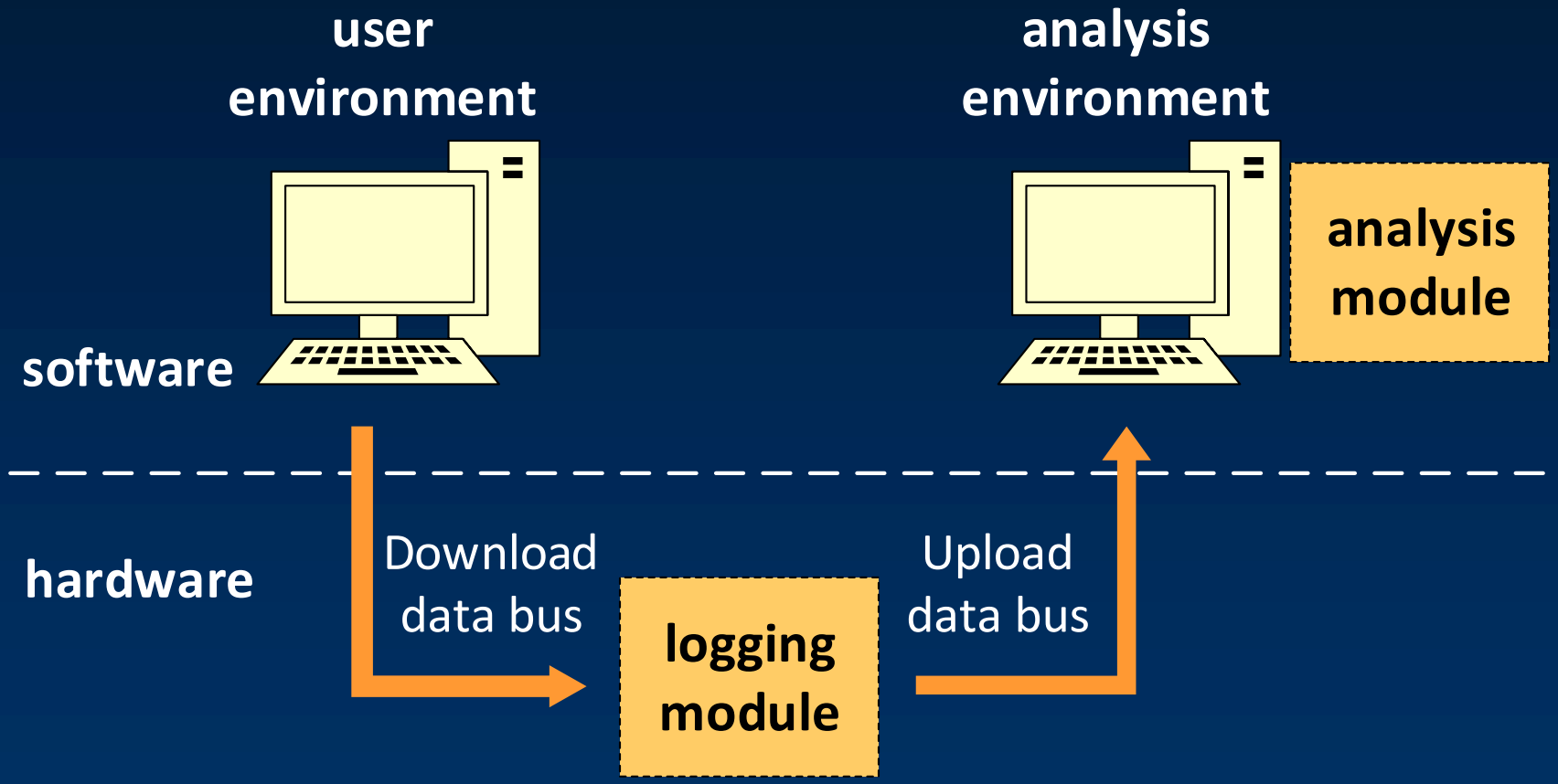
Hypervisor-level Forensic methods are immune to OS-level attacks. Unfortunately, the hypervisor itself can be the attack surface!

**D. Perez-Botero et al. "Characterizing hypervisor vulnerabilities in cloud computing servers," SCC, 2013*

Outline

- Introduction
- State-of-the-art Forensic Methods
  - OS level
  - Hypervisor level
- Hardware-based Workload Forensics
  - Process Reconstruction
- Experimental Results
  - Setup
  - Result & Overhead
- Summary

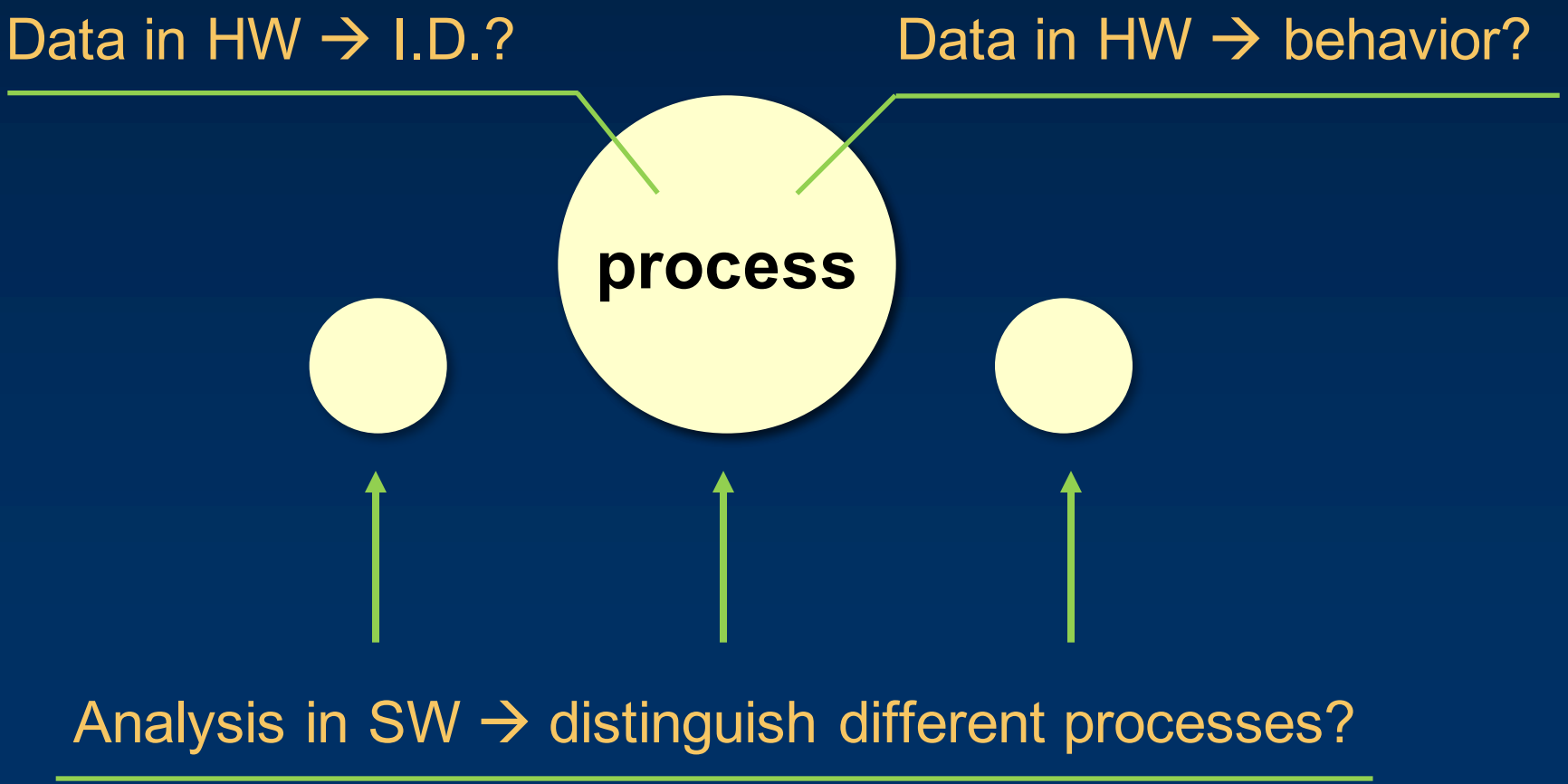
Why Hardware-based Forensics?



A logging module at hardware level is expected to be immune to software-based tampering!

Process Reconstruction – Challenge

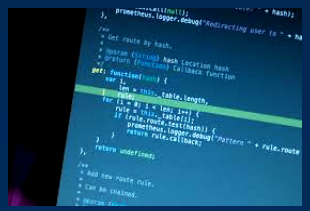
- Three main questions:



Logging Module – Logging Object

CR3 value

Identifier of the process



Phase 1

Phase 2

Phase 3

Phase 2

Phase 3

Phase 1

User-space instruction raising iTLB miss

Behavior of the process

- Why TLB profile?
  - CR3 change → TLB flush
  - Accurate association between TLB events and CR3 value
  - Mitigation of the effect of different program execution order

T. Sherwood et al. “Discovering and exploiting program phases,” IEEE Micro, 2003

Logging Module – Feature Extraction

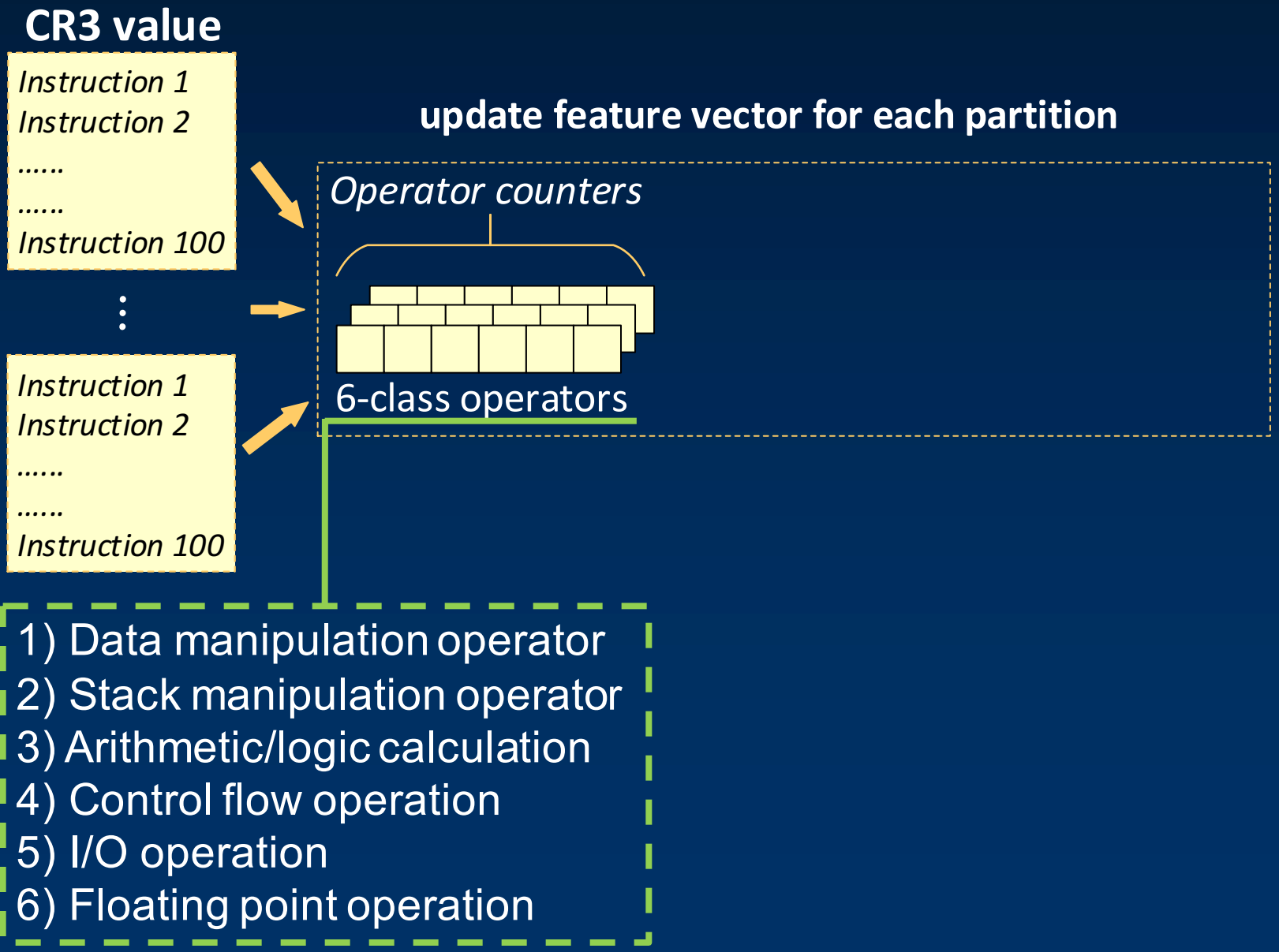
CR3 value

Instruction 1
Instruction 2
……
……
Instruction 100

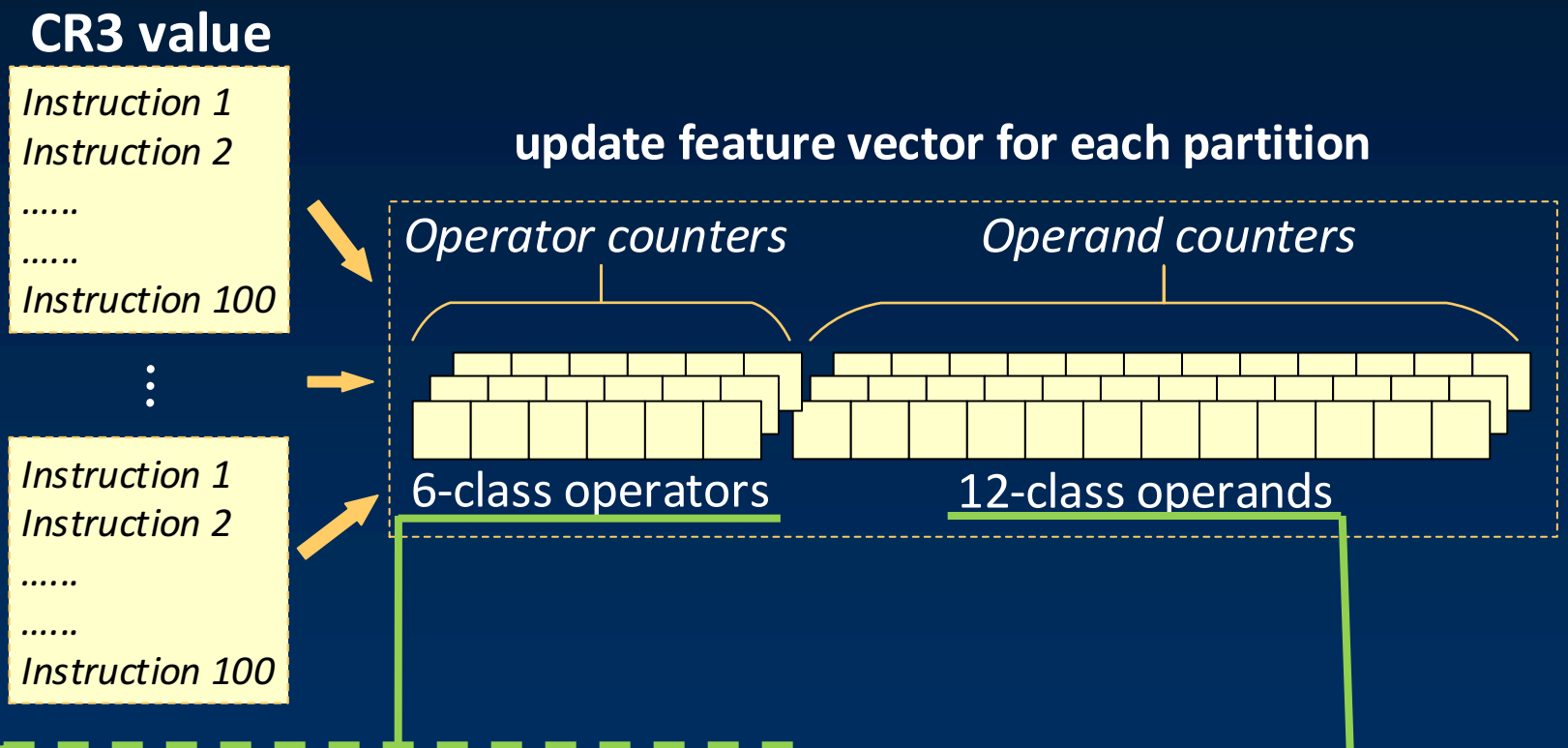
⋮

Instruction 1
Instruction 2
……
……
Instruction 100

Logging Module – Feature Extraction



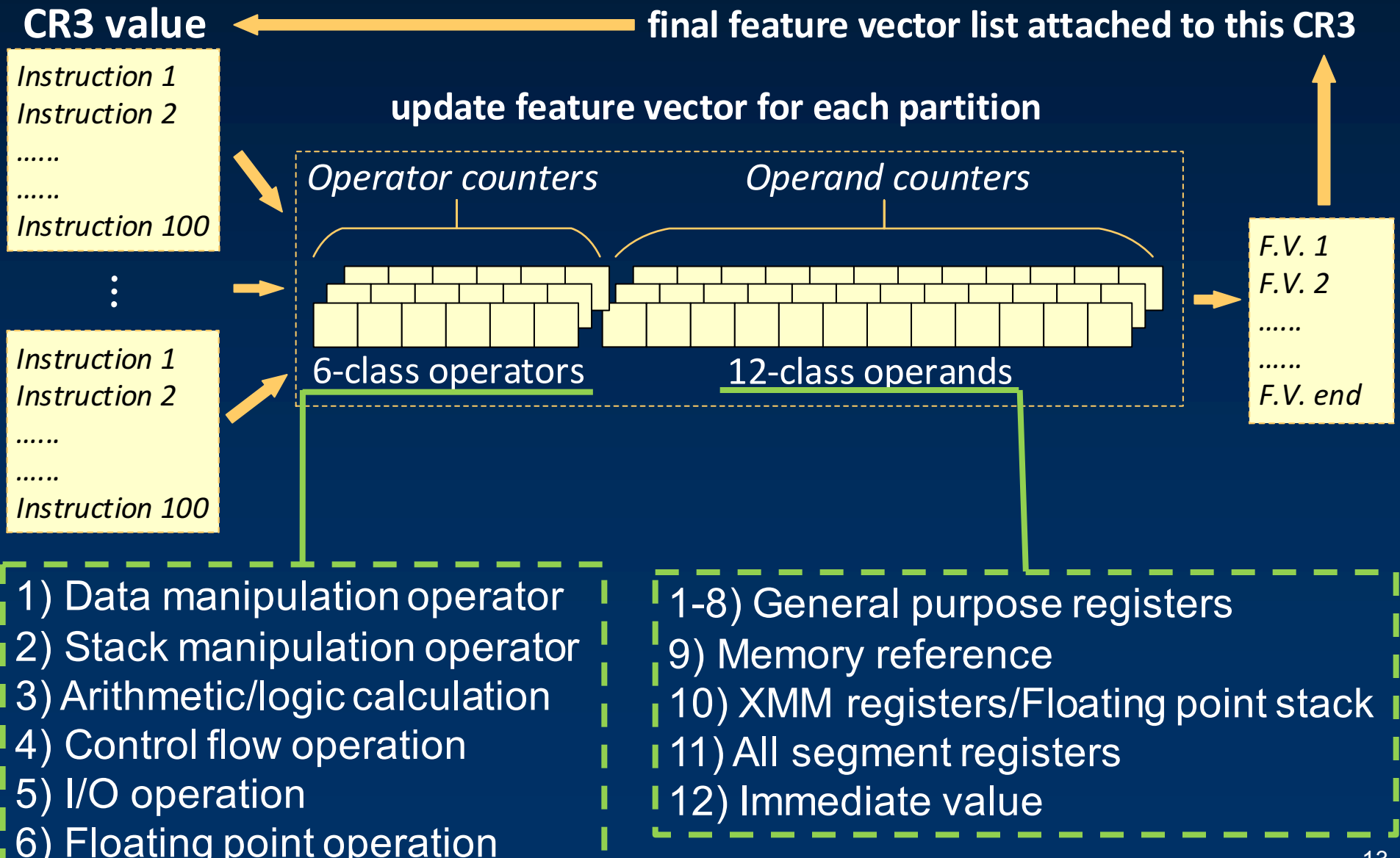
Logging Module – Feature Extraction



1) Data manipulation operator
2) Stack manipulation operator
3) Arithmetic/logic calculation
4) Control flow operation
5) I/O operation
6) Floating point operation

1-8) General purpose registers
9) Memory reference
10) XMM registers/Floating point stack
11) All segment registers
12) Immediate value

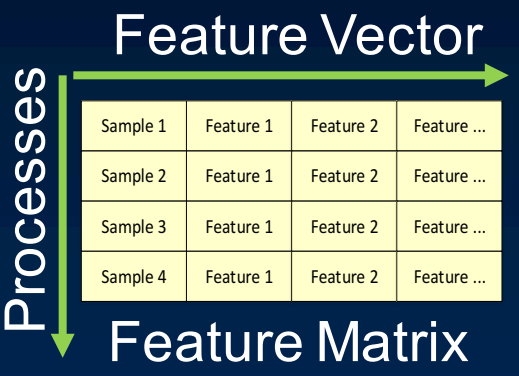
Logging Module – Feature Extraction



1) Data manipulation operator
2) Stack manipulation operator
3) Arithmetic/logic calculation
4) Control flow operation
5) I/O operation
6) Floating point operation

1-8) General purpose registers
9) Memory reference
10) XMM registers/Floating point stack
11) All segment registers
12) Immediate value

Analysis Module



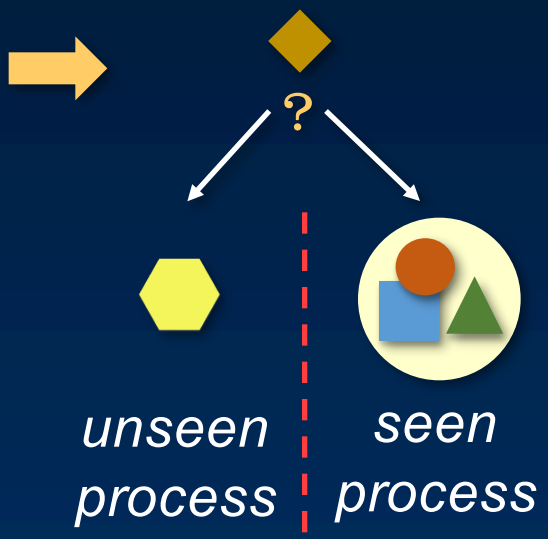
Analysis Module

Feature Vector

Processes

Sample 1	Feature 1	Feature 2	Feature ...
Sample 2	Feature 1	Feature 2	Feature ...
Sample 3	Feature 1	Feature 2	Feature ...
Sample 4	Feature 1	Feature 2	Feature ...

Feature Matrix



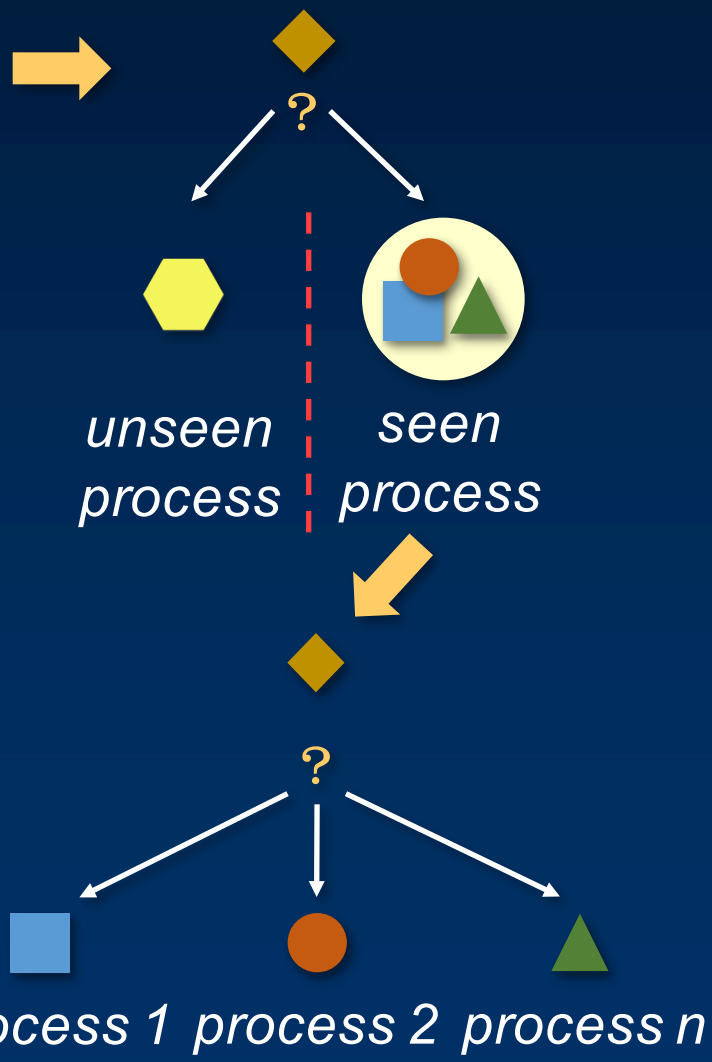
Analysis Module

Feature Vector

Processes

Sample 1	Feature 1	Feature 2	Feature ...
Sample 2	Feature 1	Feature 2	Feature ...
Sample 3	Feature 1	Feature 2	Feature ...
Sample 4	Feature 1	Feature 2	Feature ...

Feature Matrix



Analysis Module

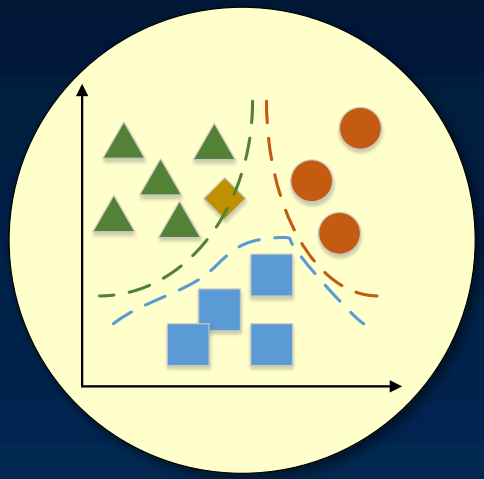
Feature Vector

Processes

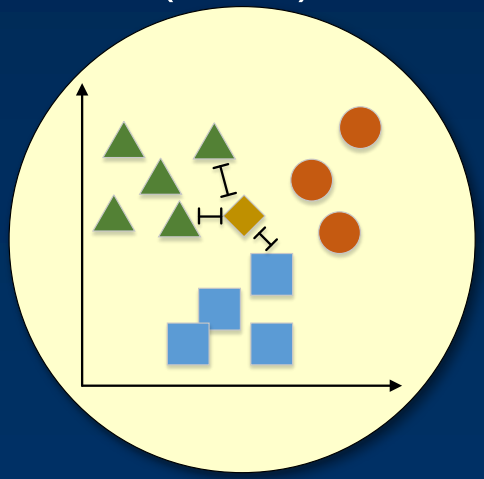
Sample 1	Feature 1	Feature 2	Feature ...
Sample 2	Feature 1	Feature 2	Feature ...
Sample 3	Feature 1	Feature 2	Feature ...
Sample 4	Feature 1	Feature 2	Feature ...

Feature Matrix

Prob. estimates



Support Vector Machine (SVM)



k-Nearest Neighbors (kNN)

unseen process

seen process

process 1 *process 2* *process n*

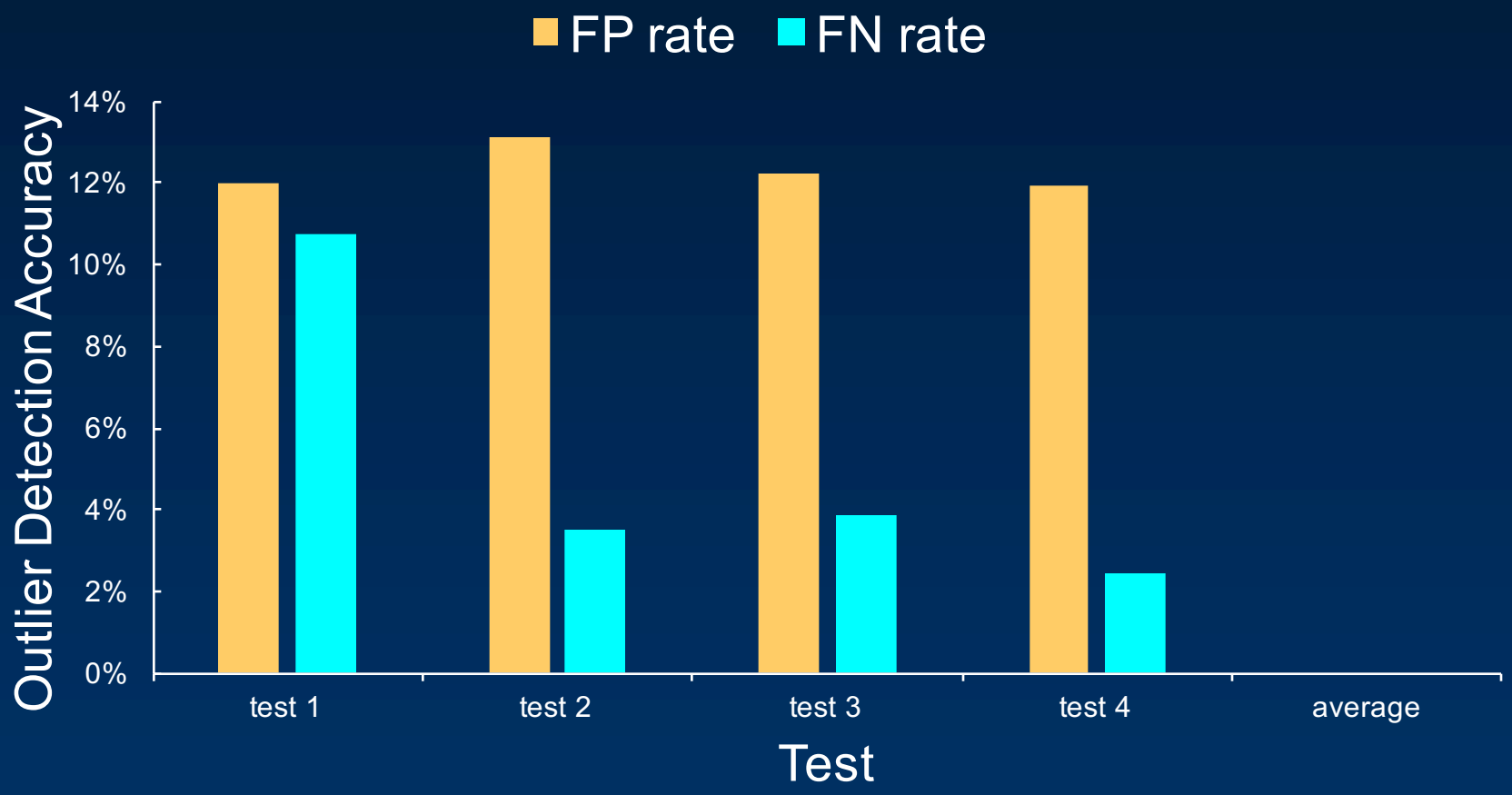
Outline

- Introduction
- State-of-the-art Forensic Methods
  - OS level
  - Hypervisor level
- Hardware-based Workload Forensics
  - Process Reconstruction
- Experimental Results
  - Setup
  - Result & Overhead
- Summary

Experimental Setup

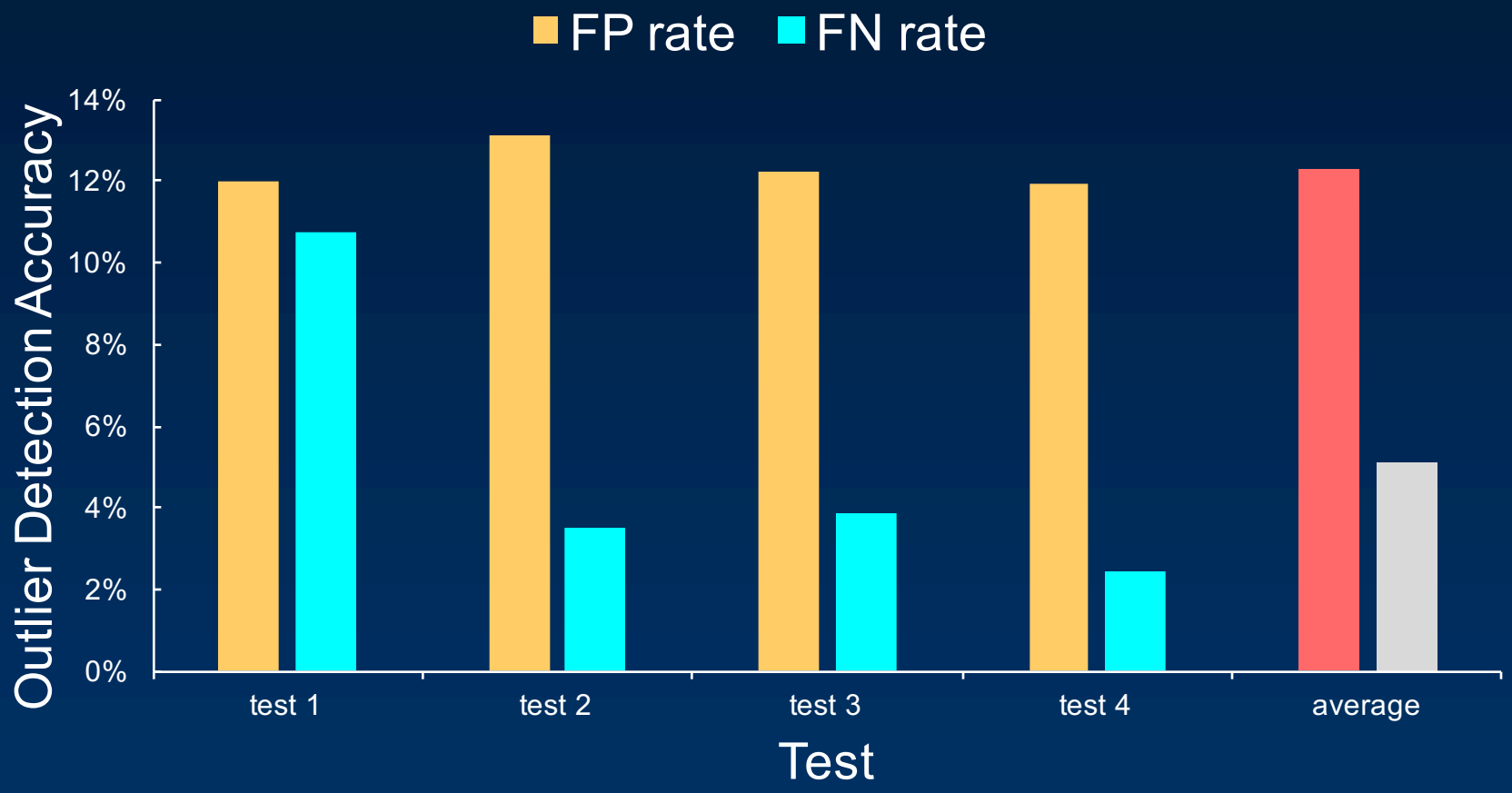
- Simulator
  - Simics 4.86
- Target Platform
  - 32-bit x86 with single Intel Pentium 4 core, 2Ghz
  - 4GB RAM
- Simulated Operating System (OS)
  - Minimum installation Ubuntu server (Linux 2.6 kernel)
- Workload Benchmark
  - Mibench
  - 50% training, 50% validation
- Analysis Software
  - Matlab

Results – Outlier Detection



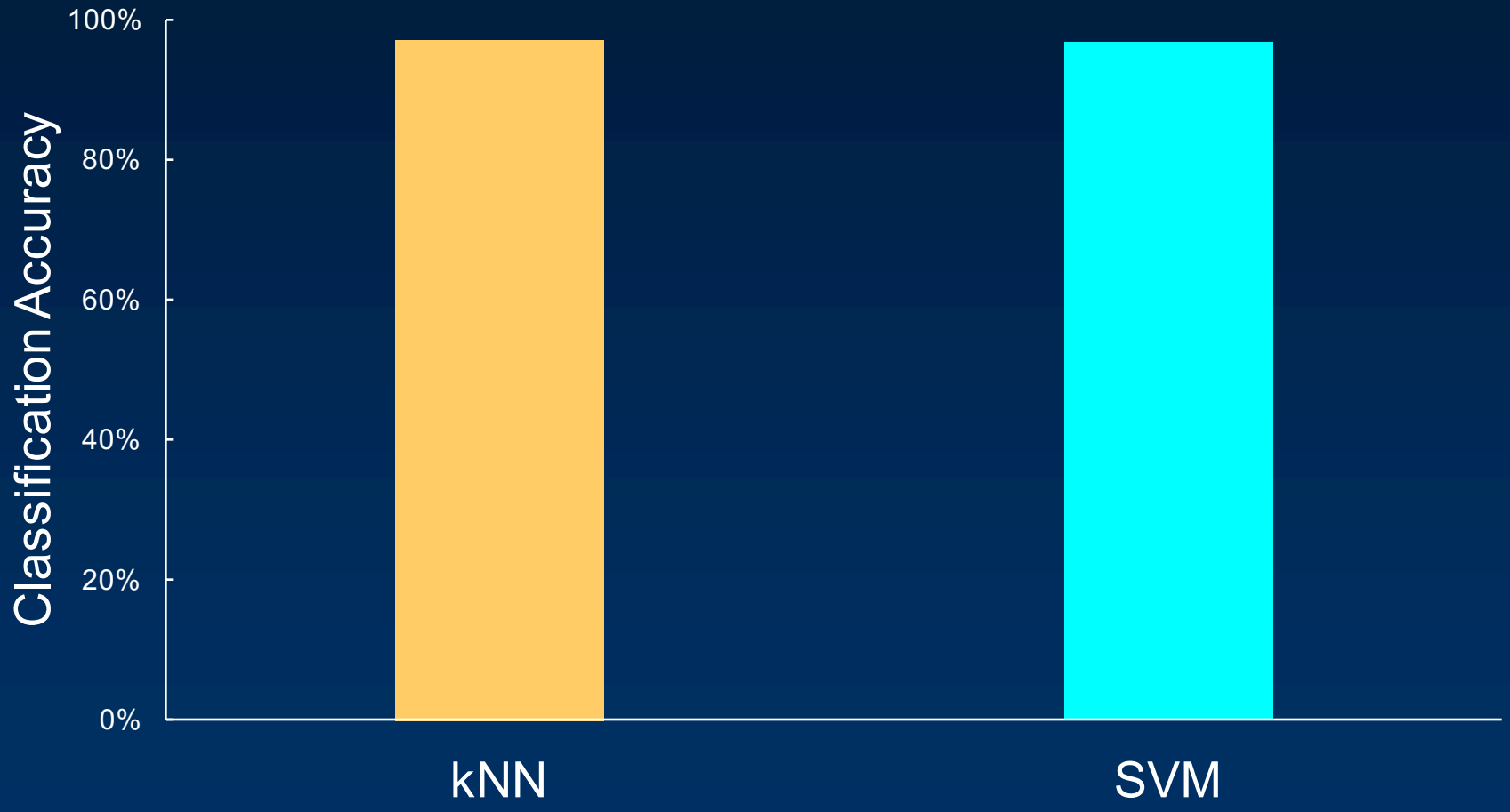
- FP: seen process classified as unseen
- FN: unseen process classified as seen

Results – Outlier Detection



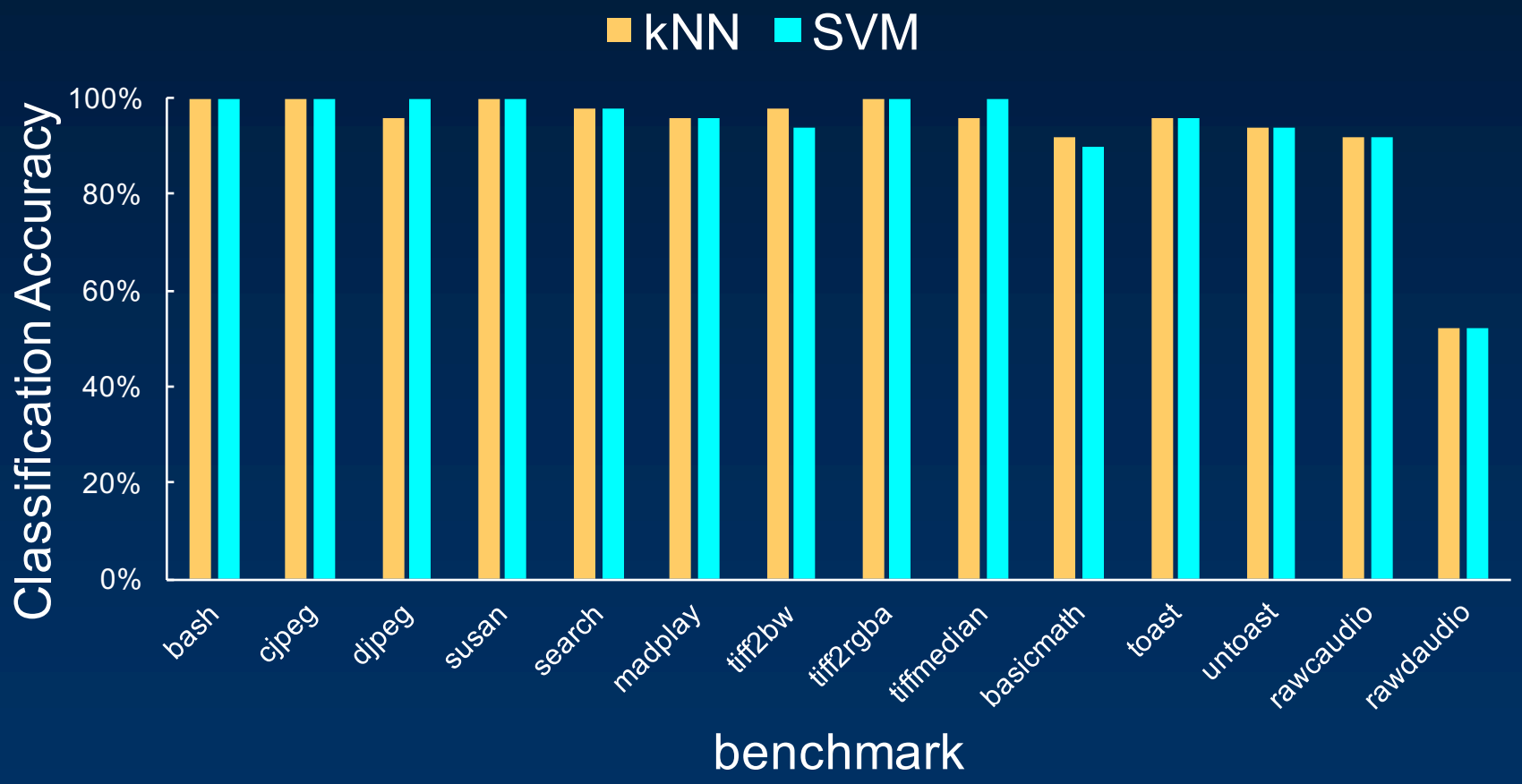
- FP: seen process classified as unseen
- FN: unseen process classified as seen
- Average FP rate: 12.31%; average FN rate: 5.13%

Results – Workload Classification



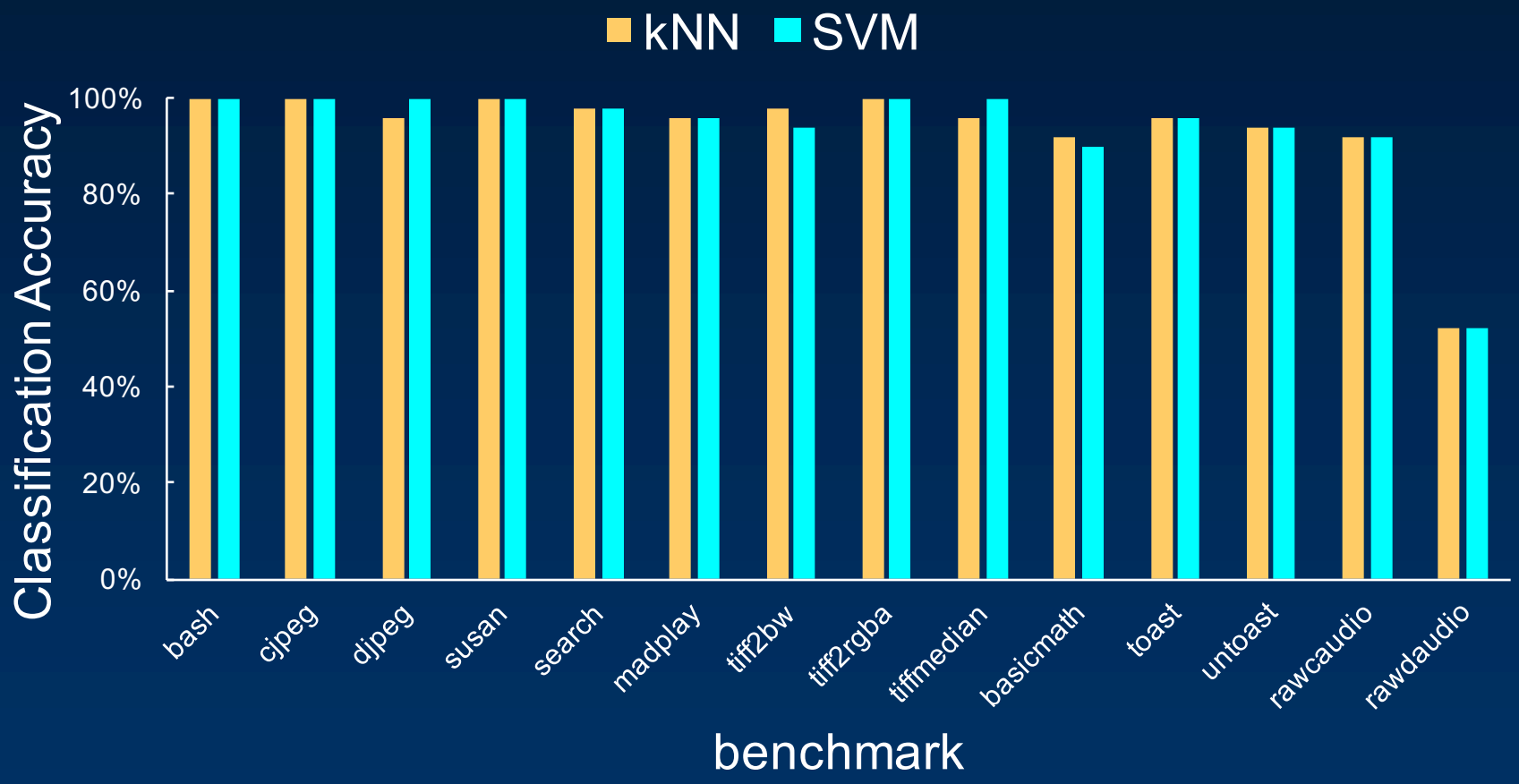
- Average classification accuracy: 96.97% for kNN and 96.93% for SVM

Results – Workload Classification



- Classification accuracy for some classes reaches 100%

Results – Workload Classification

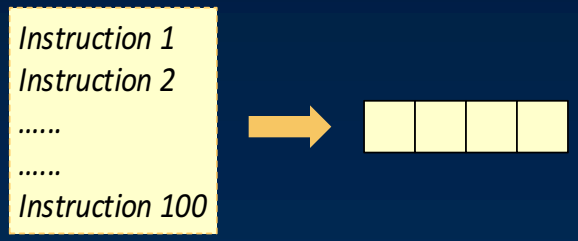


- Classification accuracy for some classes reaches 100%
- *rawcaudio* (ADPCM encoding algorithm) ⊇ *rawdaudio* (decoding algorithm) → reduced classification efficiency due to similarity

Logging Overhead

- Steps to compute logging overhead:

$$Feature\ Vector\ Size = 18 \times \lceil \log_2 partition_{\downarrow} size \rceil$$



$$Partition\ generation\ rate = \frac{iTLB\ miss\ rate}{partition_{\downarrow} size}$$

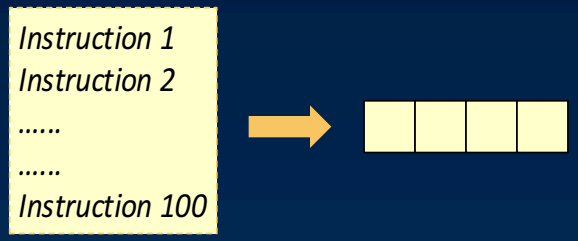
$$bits\ per\ instruction = Feature\ Vector\ size \times Partition\ generation\ rate$$

$$esitimated\ logging\ rate(bits/second) = \frac{bits\ per\ instruction \times clock\ frequency}{CPI\ (assumed = 1)}$$

Logging Overhead

- Steps to compute logging overhead:

$$Feature\ Vector\ Size = 18 \times \lceil \log_2 partition_{\downarrow} size \rceil$$



$$Partition\ generation\ rate = \frac{iTLB\ miss\ rate}{partition_{\downarrow} size}$$

$$bits\ per\ instruction = Feature\ Vector\ size \times Partition\ generation\ rate$$

$$esitimated\ logging\ rate(bits/second) = \frac{bits\ per\ instruction \times clock\ frequency}{CPI\ (assumed = 1)}$$

- Computation result:
  - Average iTLB miss rate for user space instructions is 0.0016%
  - This leads to an estimated logging rate of only 5.17 KB/s

Outline

- Introduction
- State-of-the-art Forensic Methods
  - OS level
  - Hypervisor level
- Hardware-based Workload Forensics
  - Process Reconstruction
- Experimental Results
  - Setup
  - Result & Overhead
- Summary

Summary

- Contributions
  - First hardware-based method for workload forensics analysis
  - Addresses the weakness of OS-level/hypervisor-level methods
  - Demonstrates process reconstruction feasibility via TLB profiling
- Implementation
  - Complete Hardware-to-Software logging-analysis flow
- Results
  - High workload-classification accuracy
  - Low logging overhead
- Future Work
  - Investigate information theoretic content of other features
  - Experiment with more advanced machine learning models