USENIX / ACM / IFIP 10th International Middleware Conference

How To Keep Your Head Above Water While Detecting Errors

Ignacio Laguna, Fahad A. Arshad, David M. Grothe, Saurabh Bagchi

Dependable Computing System Lab
School of Electrical and Computer Engineering
Purdue University

Impact of Failures in Internet Services

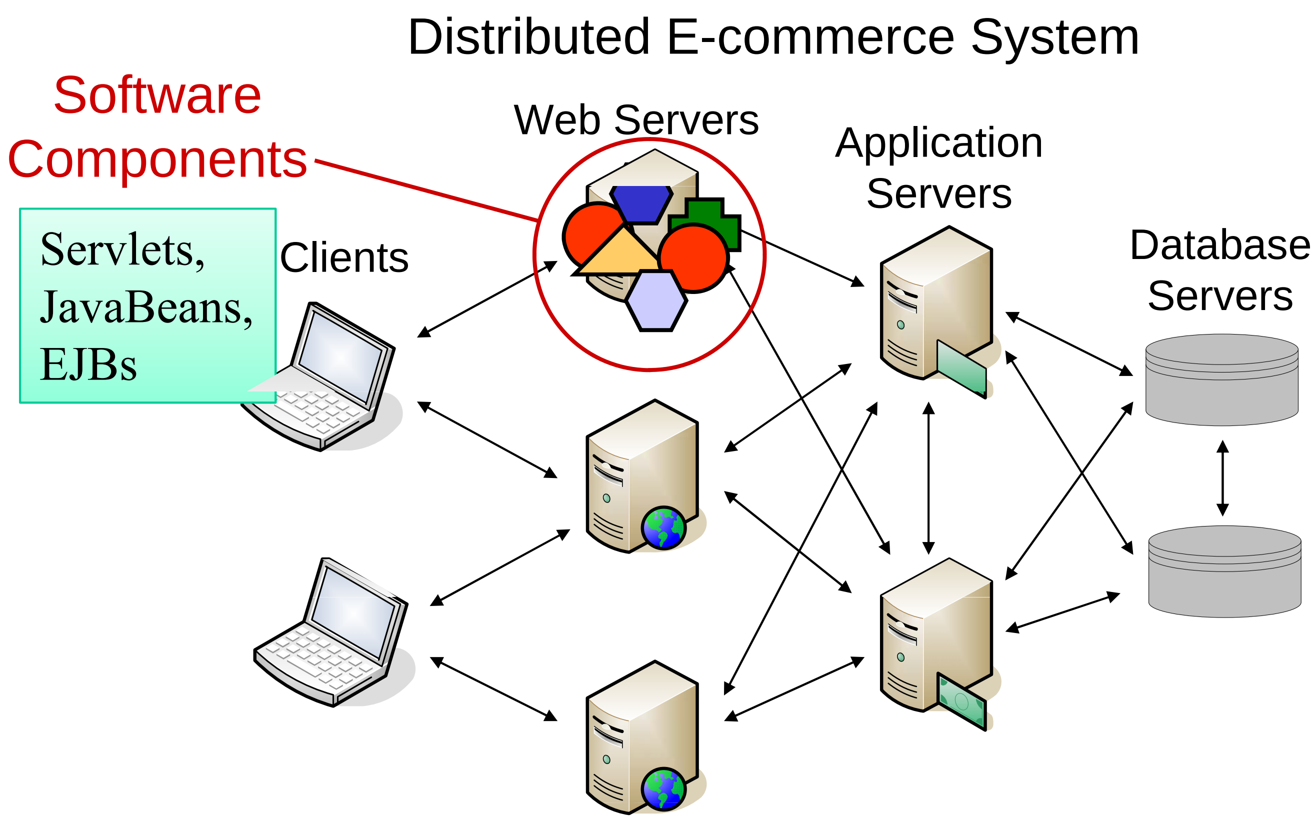
- Internet services are expected to be running 24/7
  - System downtime can cost **$1 million** / hour

    (Source: Meta Group, 2002)

- *Service degradation* is the most frequent problem
  - Service is slower than usual; almost unavailable
  - Can be difficult to detect and diagnose

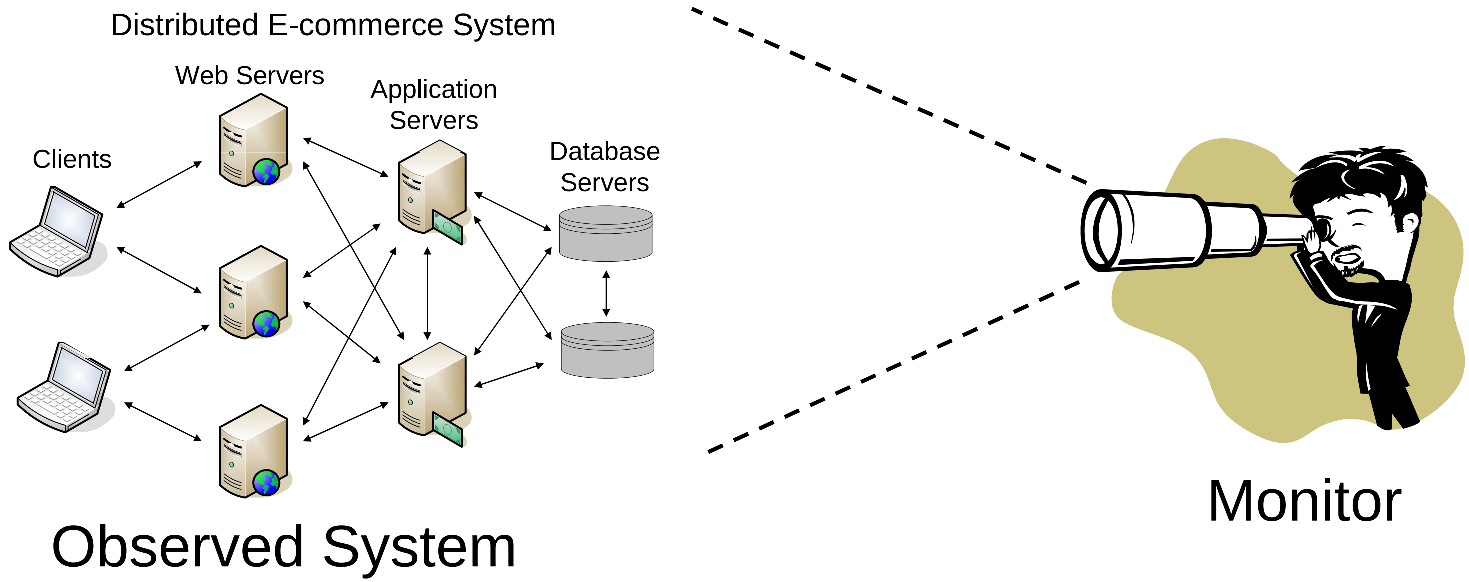
- Internet-based applications are very large and dynamic
  - Complexity increases as new components are added

Complexity of Internet-based Applications



- Each tier has multiple components
- Components can be stateful

The *Monitor* Detection System (*TDSC '06*)



- ***Non-intrusive*** **—** observe messages between components
- ***Online*** **—** faster detection than offline approaches
- ***Black-box detection*** **—** components treated as *black boxes*
  - No knowledge of components' internals

Stateful Rule-based Detection

Monitor

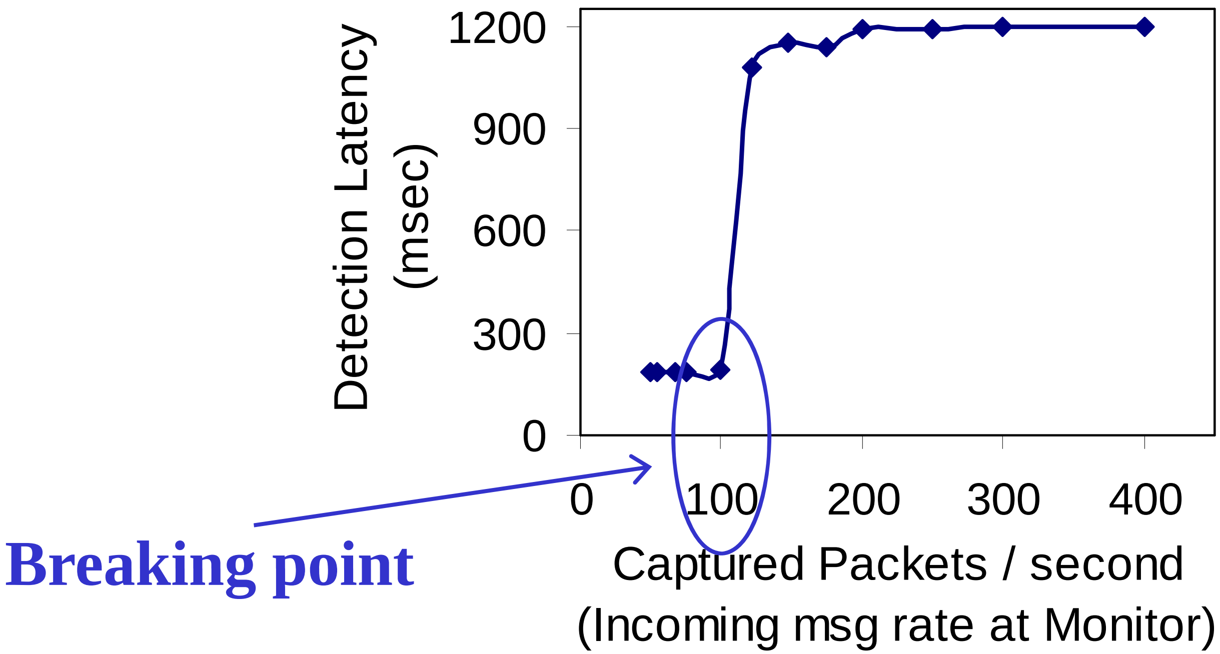
- **Packet Capturer**
- **Finite State Machine** (event-based state transition model)
- **Normal-Behavior Rules**

Detection Process

1. A *message* is captured →
2. Deduce current application *state* →
3. Match rules based on the deduced *state* →
4. If a rule does not satisfy, signal *alarm*

PURDUE UNIVERSITY

The *Breaking Point* in High Rate of Messages



- After breaking point, latency increases sharply
- True alarms rate decreases because packets are dropped
- Breaking point expected in any stateful detection system

Avoiding the Breaking Point: *Random Sampling* (SRDS '07)

- *Processing Load* in Monitor

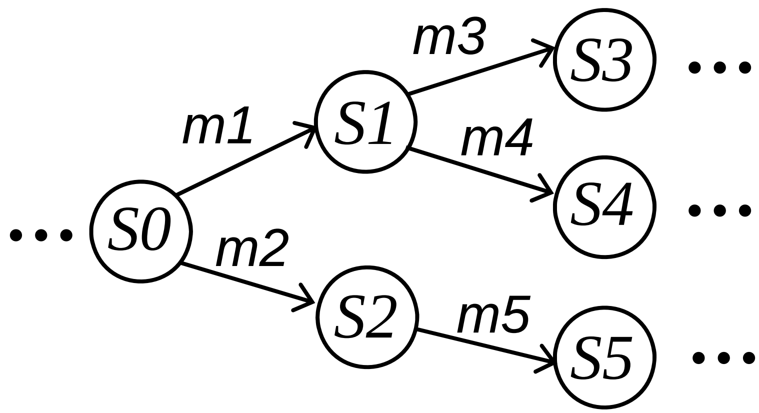
$$\delta = R \times C$$

R: Incoming message rate, C: Processing cost per message

- Processing Load δ is reduced by reducing R
  - Only a portion of incoming messages is processed
  - n out of m messages are *randomly sampled*

- Sampling is activated if $R \geq$ *breaking point*

The *Non-Determinism* Caused by Sampling



Events in Monitor

(1) SV = { *S0* }
(2) A message is dropped
(3) SV = { *S1*, *S2* }
(4) A message is sampled
(5) The message is *m5*
(6) SV = { *S5* }

Definitions:

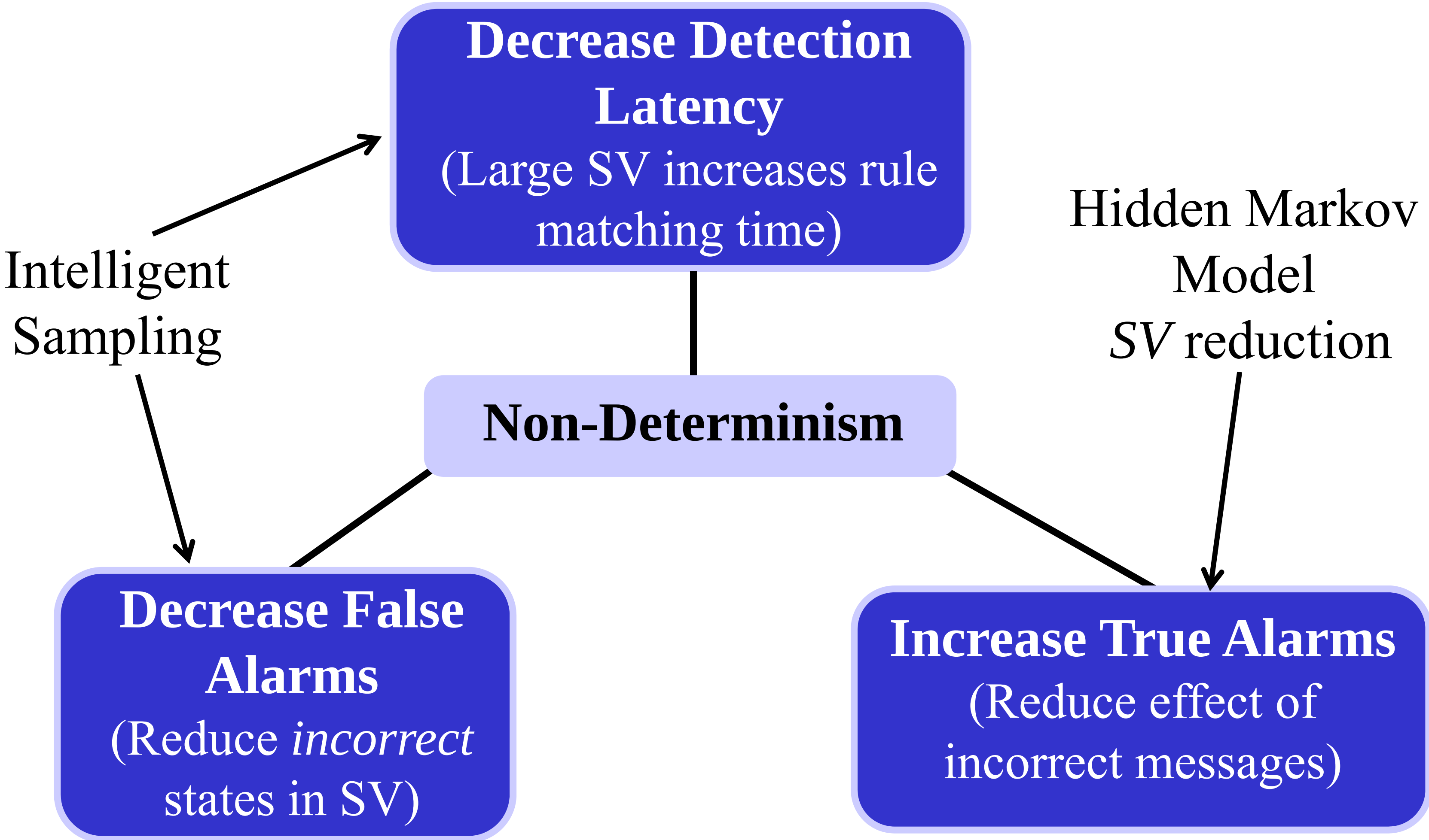
State Vector (SV) — The state(s) of the application from Monitor's point of view (deduced state(s))

Non-Determinism — Monitor is no longer aware of the exact state the application is in (because of dropped messages)

Remaining Agenda

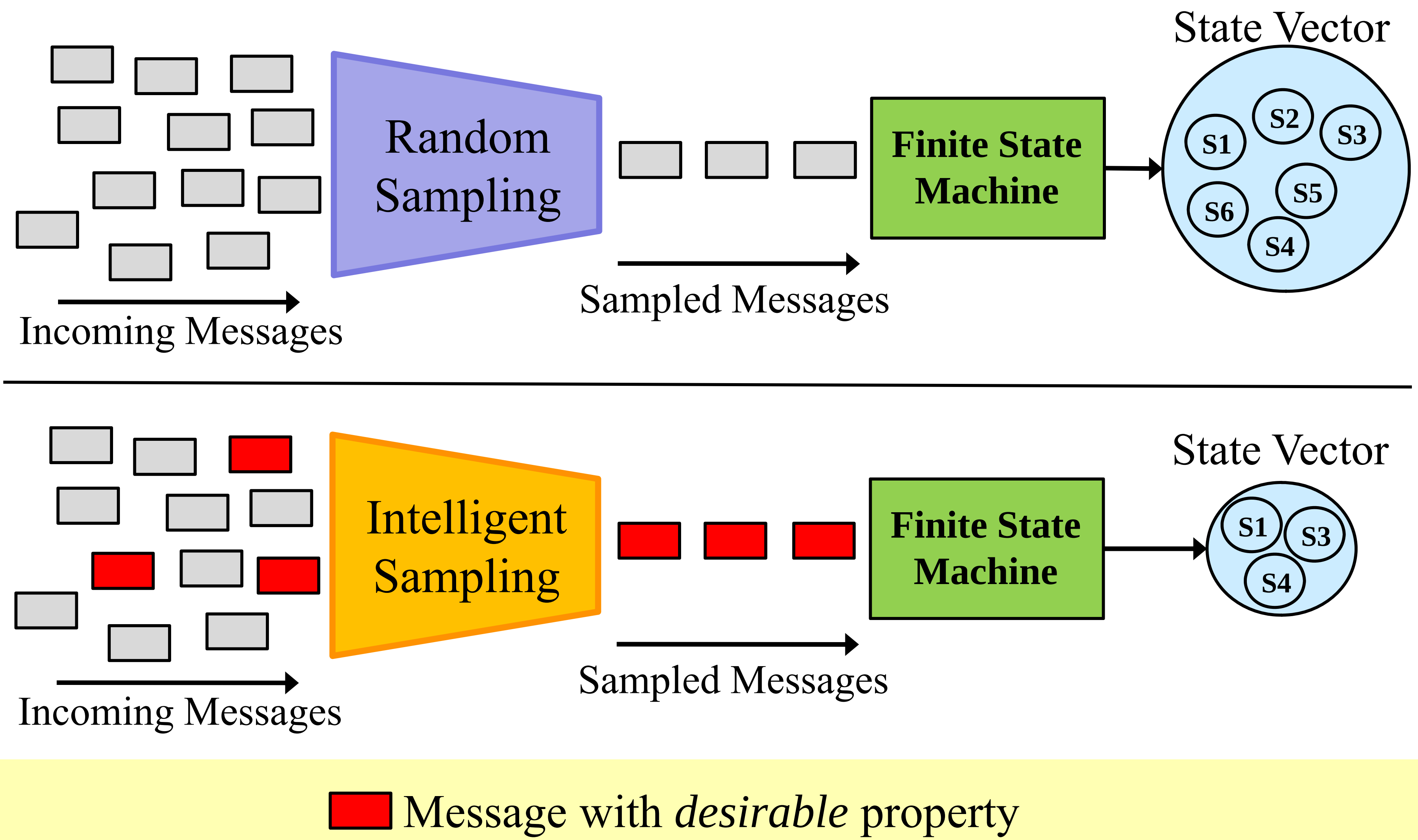
I. Addressing the problem of *non-determinism*
   A. Intelligent Sampling
   B. Hidden Markov Model (HMM) for state determination
II. Experimental Test-bed
III. Performance Results
IV. Efficient Rule Matching and Triggering

Challenges with Non-Determinism



PURDUE UNIVERSITY

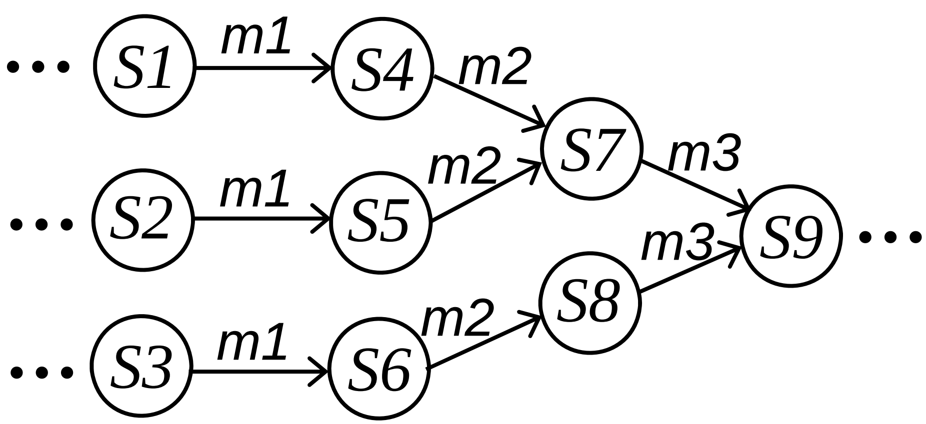
Intelligent Sampling



What is a Desirable Property of a Message?

Suppose, *SV* ={ *S1*, *S2*, *S3* }, *Sampling Rate* = 1/3

A portion of a Finite State Machine



Sampled Message	SV	Discriminative Size
m1	{*S4*, *S5*, *S6*}	3
m2	{ *S7*, *S8* }	2
m3	{ *S9* }	1

Discriminative Size — Number of times a message appears in transitions to different states in the FSM

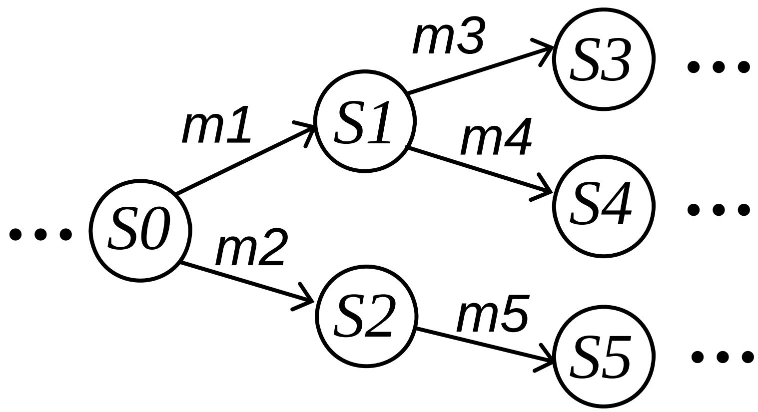
- Desirable property is a small *discriminative size*

Benefits of Intelligent Sampling

Random Sampling	Intelligent Sampling
• *SV* can grow into large size	• *SV* is kept small – Detection latency reduction
• Multiple incorrect states – Increase of false alarms	• Less incorrect states in *SV* – False alarms reduction

The Problem of Sampling an *Incorrect Message*

- What if an *incorrect* message is sampled?
  - The message is *incorrect* in current states, e.g., a message from buggy component



- Suppose *SV* = { *S1*, *S2* } and *m3* is changed to *m5*

  $\Rightarrow$ *SV* = { *S5* } (incorrect *SV*!, it should be { *S3* })

Probabilistic State Vector Reduction: *A Hidden Markov Model Approach*

- Hidden Markov Model (HMM) used to reduce SV
  - An HMM is an extended ***Markov Model*** where states are not observable
  - States are hidden as in the monitored application

- Given an HMM, we can ask:

  The probability of the application being in any state, given a sequence of messages?

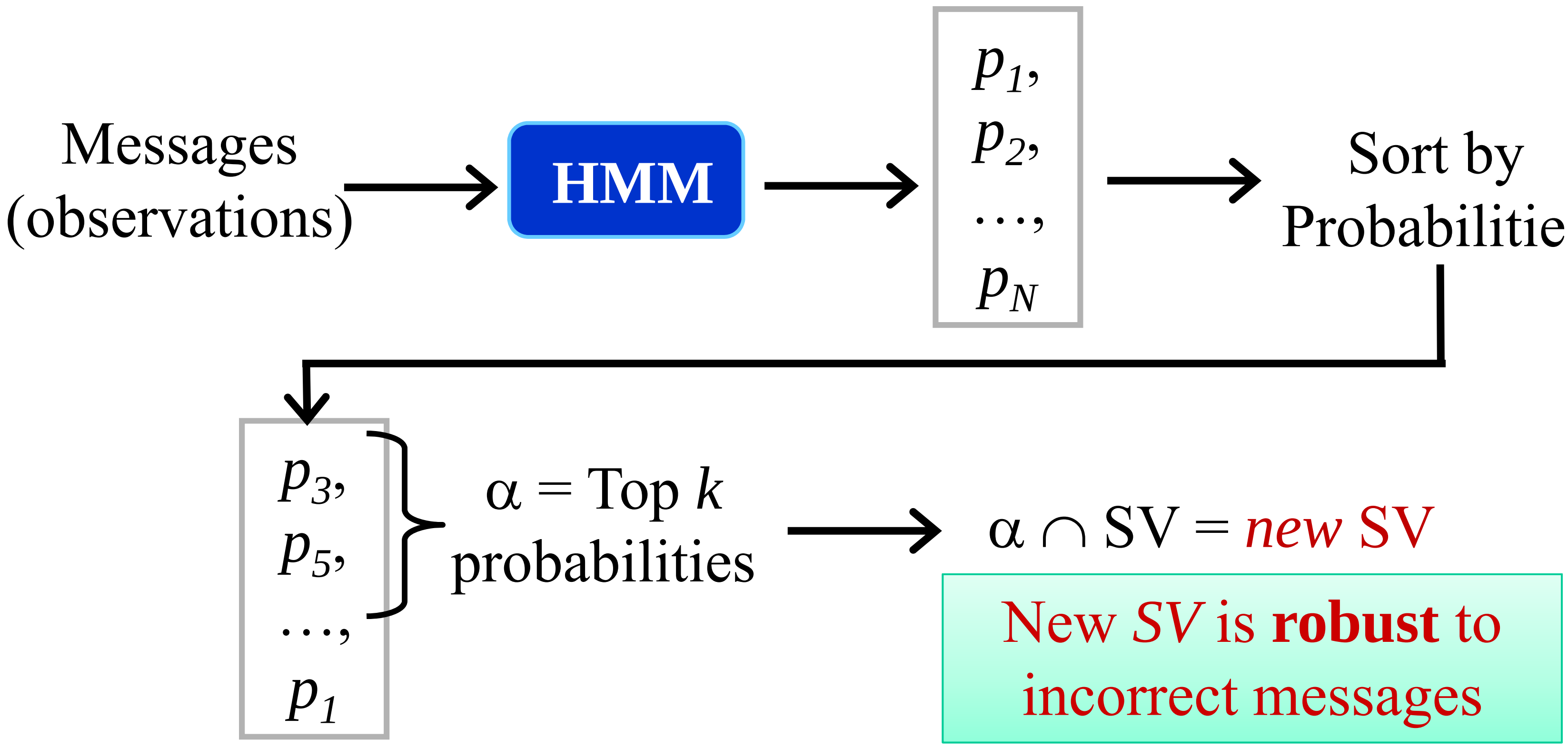
  Cost is **O(N^3L)**, ***N***: number of states ***L***: sequence length

- The HMM is trained (offline) with application traces

State Vector Reduction with the HMM

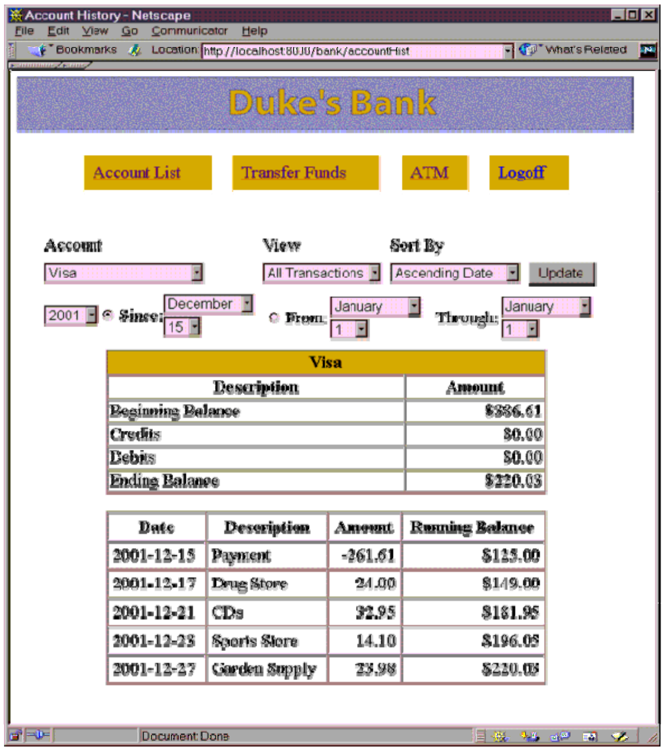
- Monitor asks the HMM: $\{p_1, p_2, \ldots, p_N\}$

  $p_i = P(S_i \mid O)$, S_i: application state i, O: observation's sequence



PURDUE UNIVERSITY

Experimental Testbed: Java Duke's Bank Application

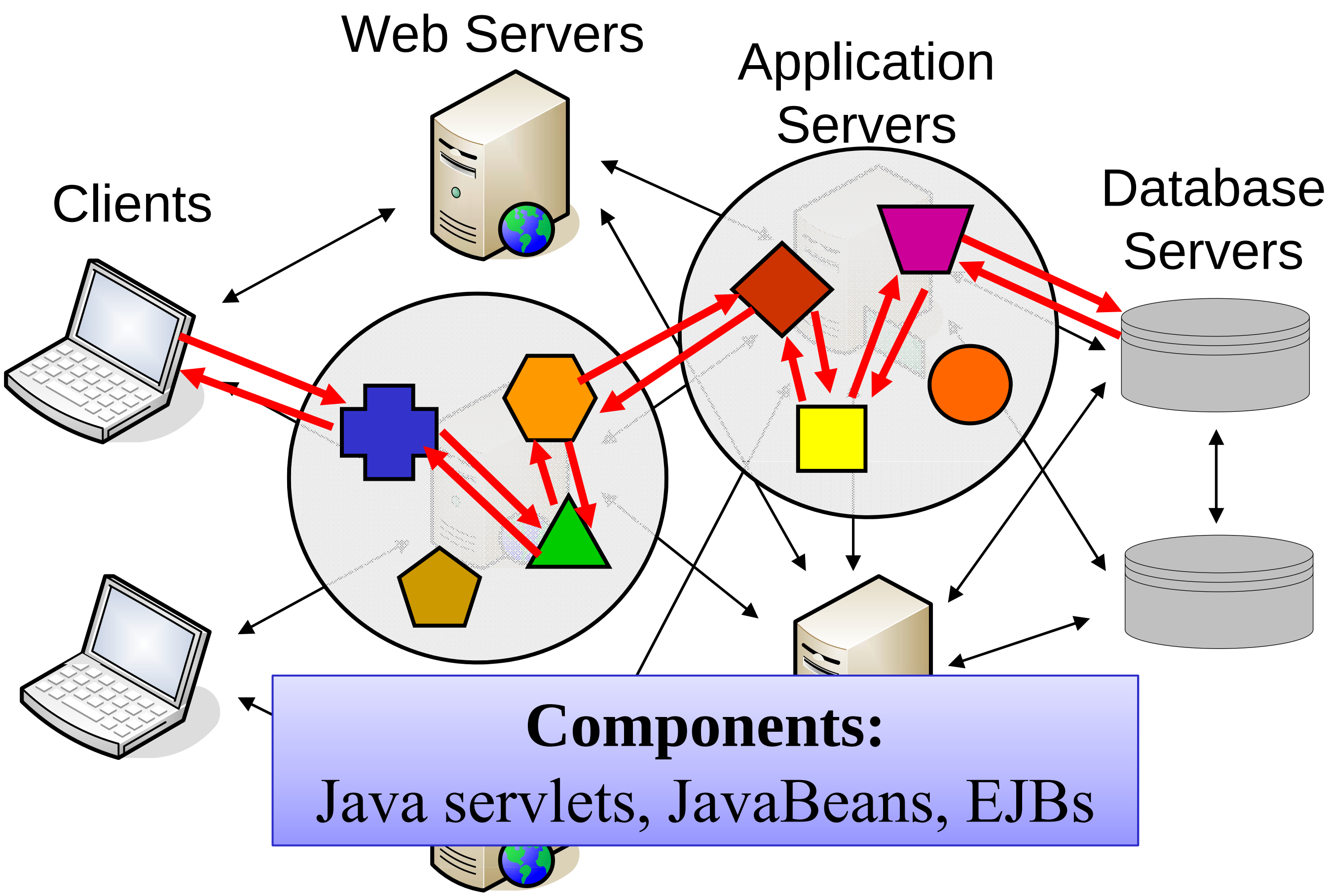


- Simulates multi-tier online banking system
- User transactions:
  - Access account information, transfer money, etc.
- Application stressed with different workloads
  - Incoming message rate at Monitor varies with user load

PURDUE UNIVERSITY

Web Interaction: A Sequence of *calls* and *returns*

Distributed E-commerce System



Error Injection Types

Error Type	Description
Response Delays	a response delay in a method call
Null calls	a call to a component that is never executed
Unhandled Exceptions	exception thrown by execution that is never caught by the program
Incorrect Message Sequences	change randomly the web interaction structure

- Errors are injected in components touched by web interactions
  - A web interaction is faulty if at least one of its components is faulty

Performance Metrics Used in Experiments

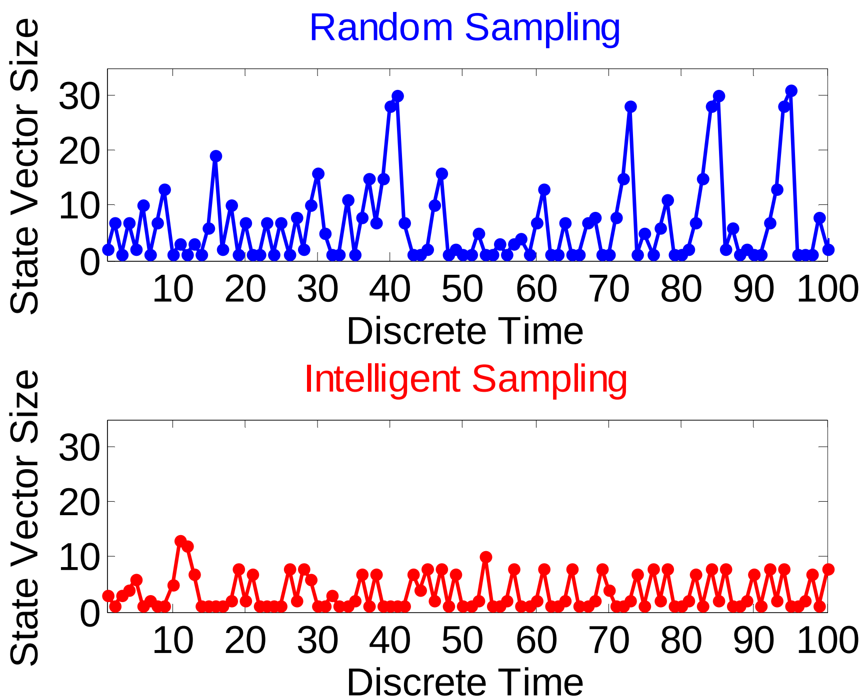
Accuracy (True Alarms)	% of *true* detections out of web interactions where errors were injected
Precision (False Alarms)	% of *true* detections out of the total number of detections
Detection Latency	time elapsed between the error injection and its detection

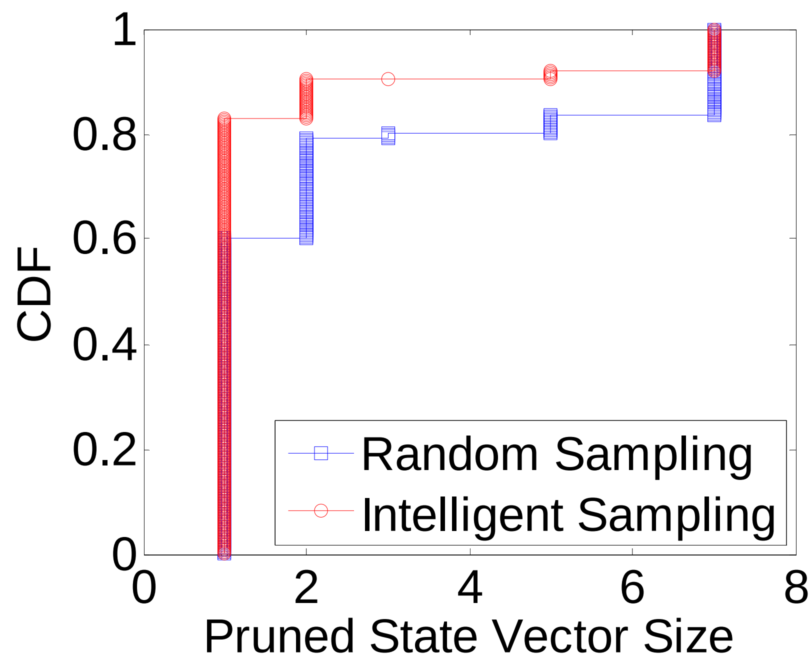
Example:

	Web Interactions				
	1	2	3	4	5
Error Injected		X	X		X
Detection (An alarm is signaled)	X	X		X	X

Accuracy = 2/3 = 0.67 **Precision** = 2/4 = 0.5

Results: State Vector Reduction

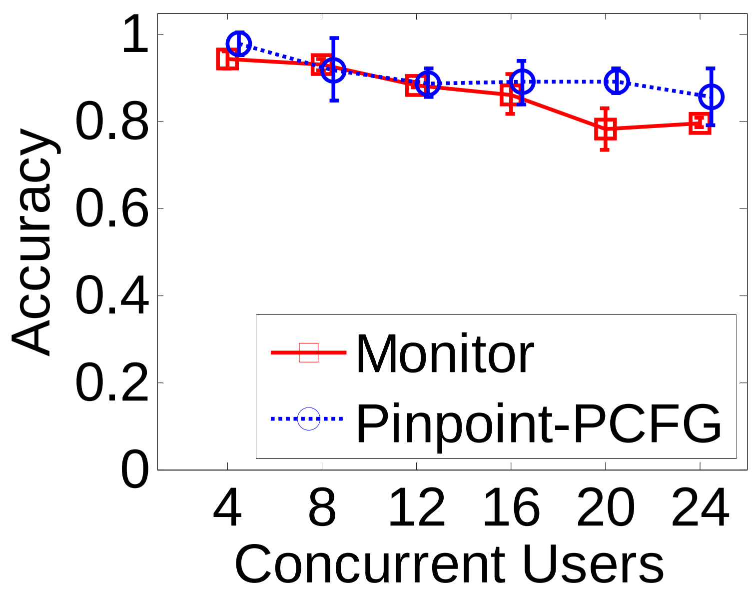


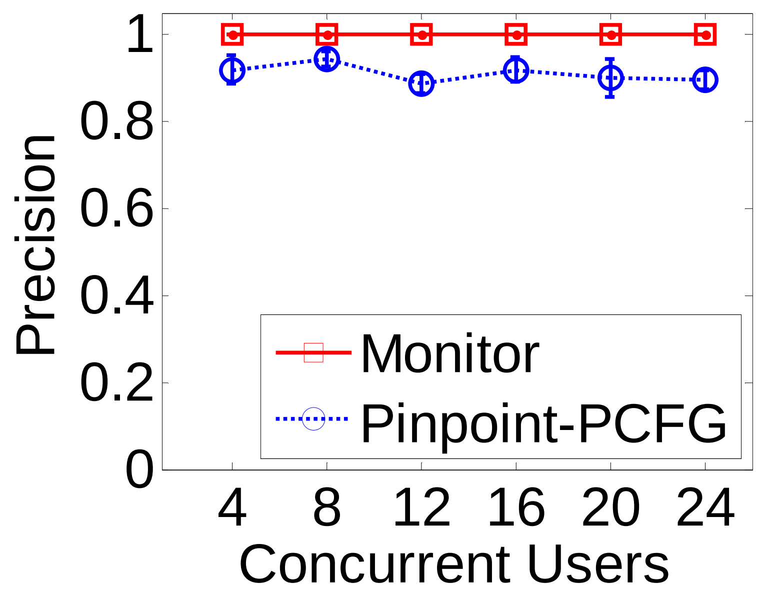


- Peaks are not observed in intelligent sampling (IS)
  - IS capability of selecting messages with small discriminative size
- SV of size 1 is more frequent in IS

Results: Monitor vs. Pinpoint (Accuracy, Precision)

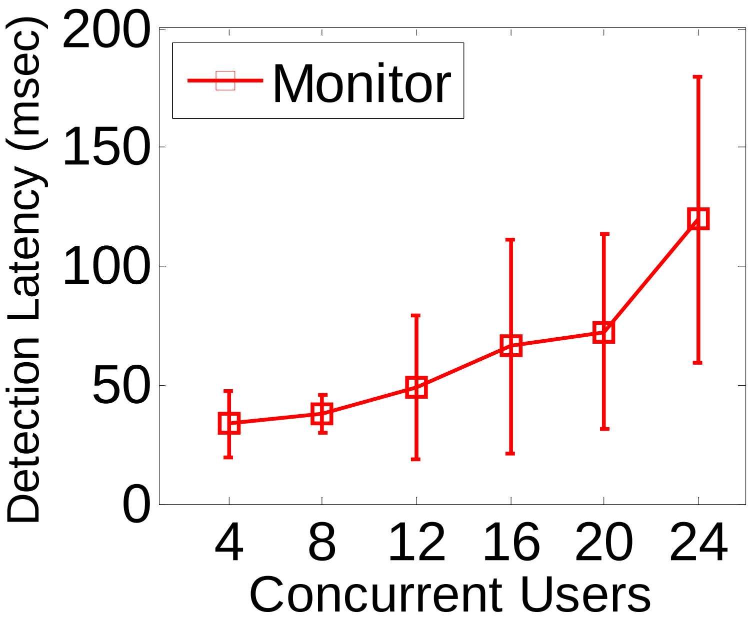
- **Pinpoint** (*NSDI* ‘04), traces paths though multiple components
- Use of **PCFG** to detect abnormal paths

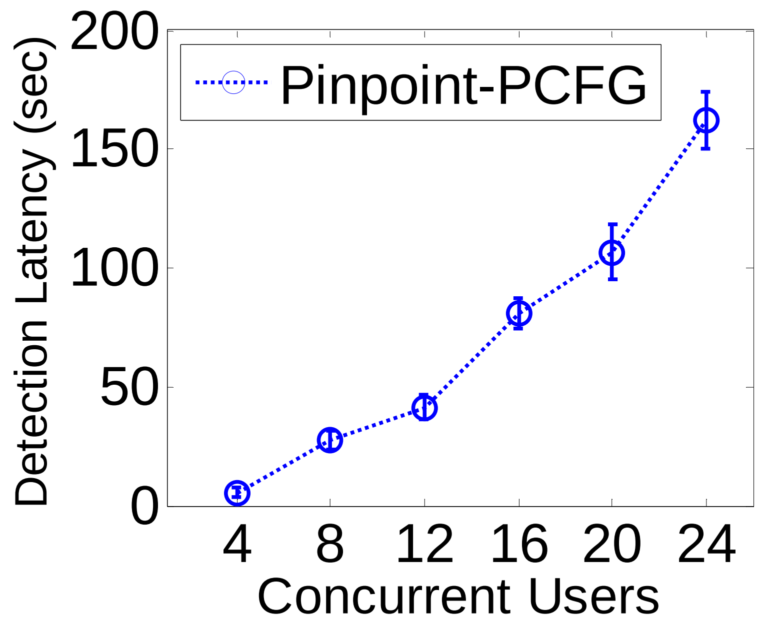




- Monitor and Pinpoint expose similar levels of accuracy
- Precision in Monitor (1.0) is higher than in Pinpoint (0.9)

Results: Monitor vs. Pinpoint (Detection Latency)





- Detection latency in Monitor is in the order of **milliseconds**, while in Pinpoint is in **seconds**
- The PCFG has a high space and time complexity

Results: Memory Consumption (MB)

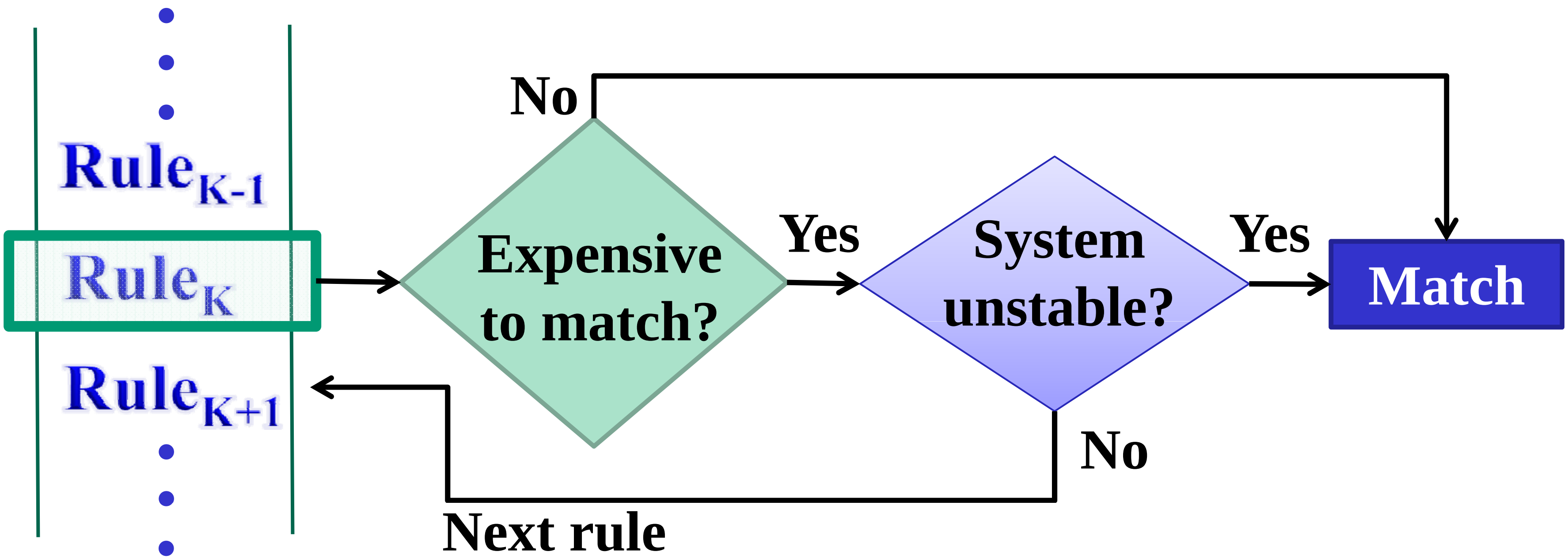
	Virtual Memory	RAM
Monitor	282.27	25.53
Pinpoint-PCFG	933.56	696.06

- Monitor doesn't rely in large data structures
- PCFG in Pinpoint has high **space** and **time** complexity
  - $O(RL^2)$ and $O(L^3)$

  R: number of rules in the grammar

  L: size of a web interaction
- Pinpoint thrashes due to high memory requirements

Efficient Rule Matching



- Selectively match computationally expensive rules
  - Expensive rules don't have to be matched all the time
- Rules are matched only if instability is present

Efficient Rule Matching Example: ***Detecting Memory Leak***

- Efficiently detect memory leak in Apache web server
  - Memory leak injected probabilistically with web requests
- Expensive ARIMA-based rule to detect abnormal memory usage
  - Average matching latency reduced

Rule Matching Criteria	Memory Leak Detected	Average Matching Latency (msec.)
Always matched	yes	19.283
$\sigma \geq 0.5$	yes	7.115
$\sigma \geq 1.0$	no	1.25

Concluding Remarks

- Contributions:
  - **Sampling** used to **scale** stateful detection system under high-rate of messages
  - **Intelligent Sampling** reduces non-determinism caused by sampling
  - **HMM-approach** handles incorrect messages
  - Techniques can be applied to any stateful detection system
  - Monitor performs better than other approaches

- Future Work:
  - **Efficient Rule Matching** technique will be extended
  - Sampling only sequences of messages that lead to errors
  - Automatic generation of rules from traces