UNICORN

Runtime Provenance-Based Detector for Advanced Persistent Threats

Xueyuan Han, James Mickens
Harvard University

Thomas Pasquier
University of Bristol

Adam Bates
University of Illinois at Urbana-Champaign

Margo Seltzer
University of British Columbia

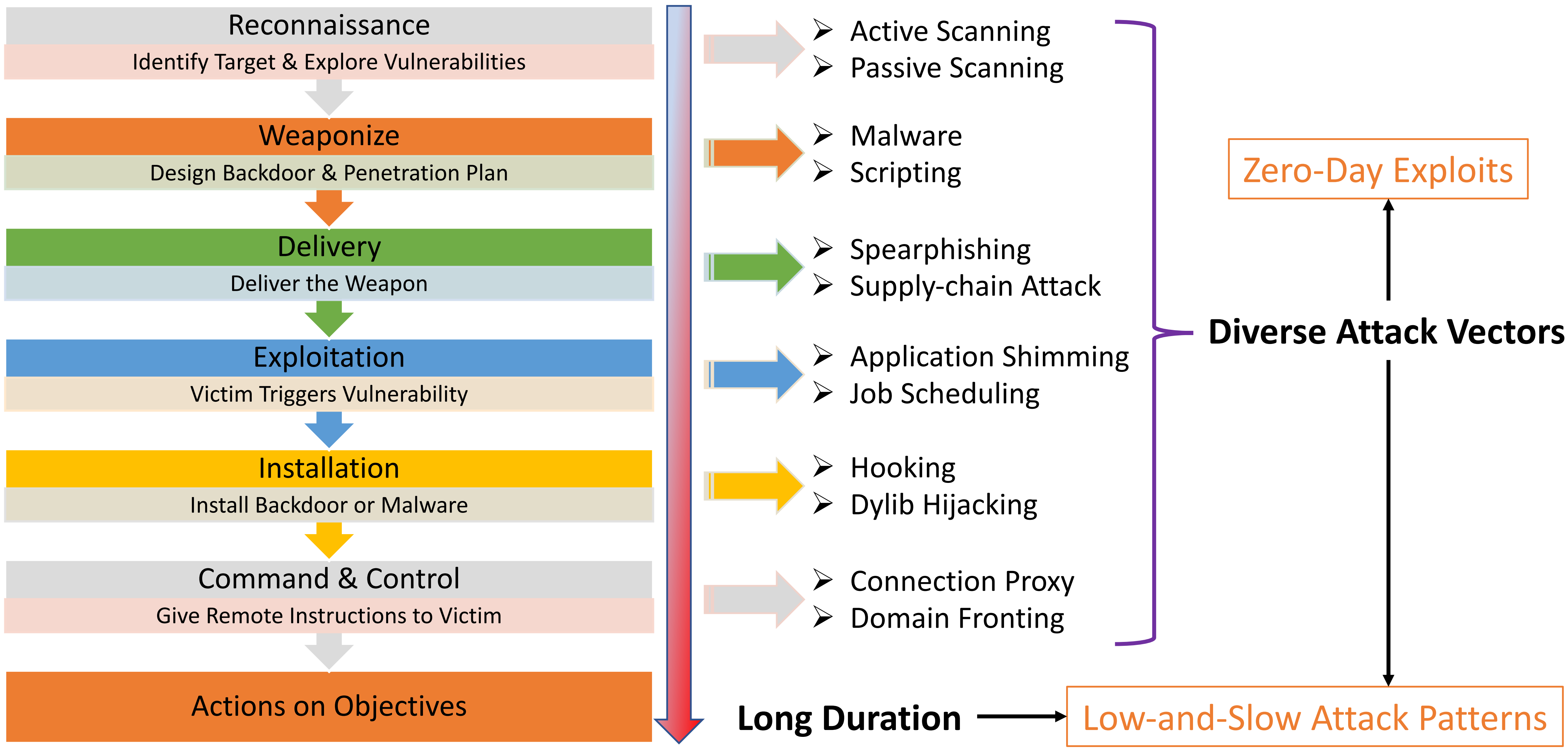
HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

University of BRISTOL

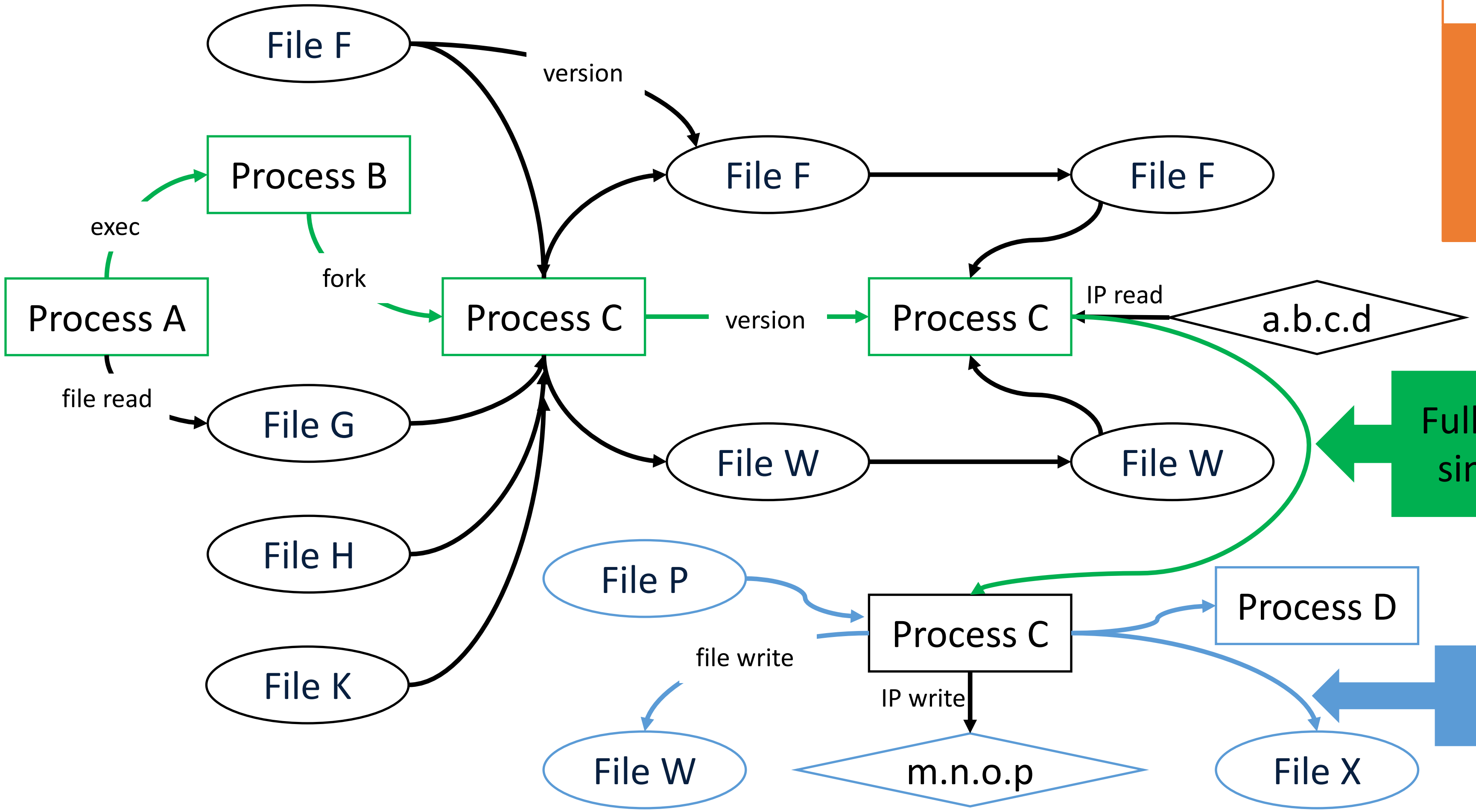
ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

THE UNIVERSITY OF BRITISH COLUMBIA

Advanced Persistent Threats



Whole-System Data Provenance



Low-and-Slow Attack Patterns

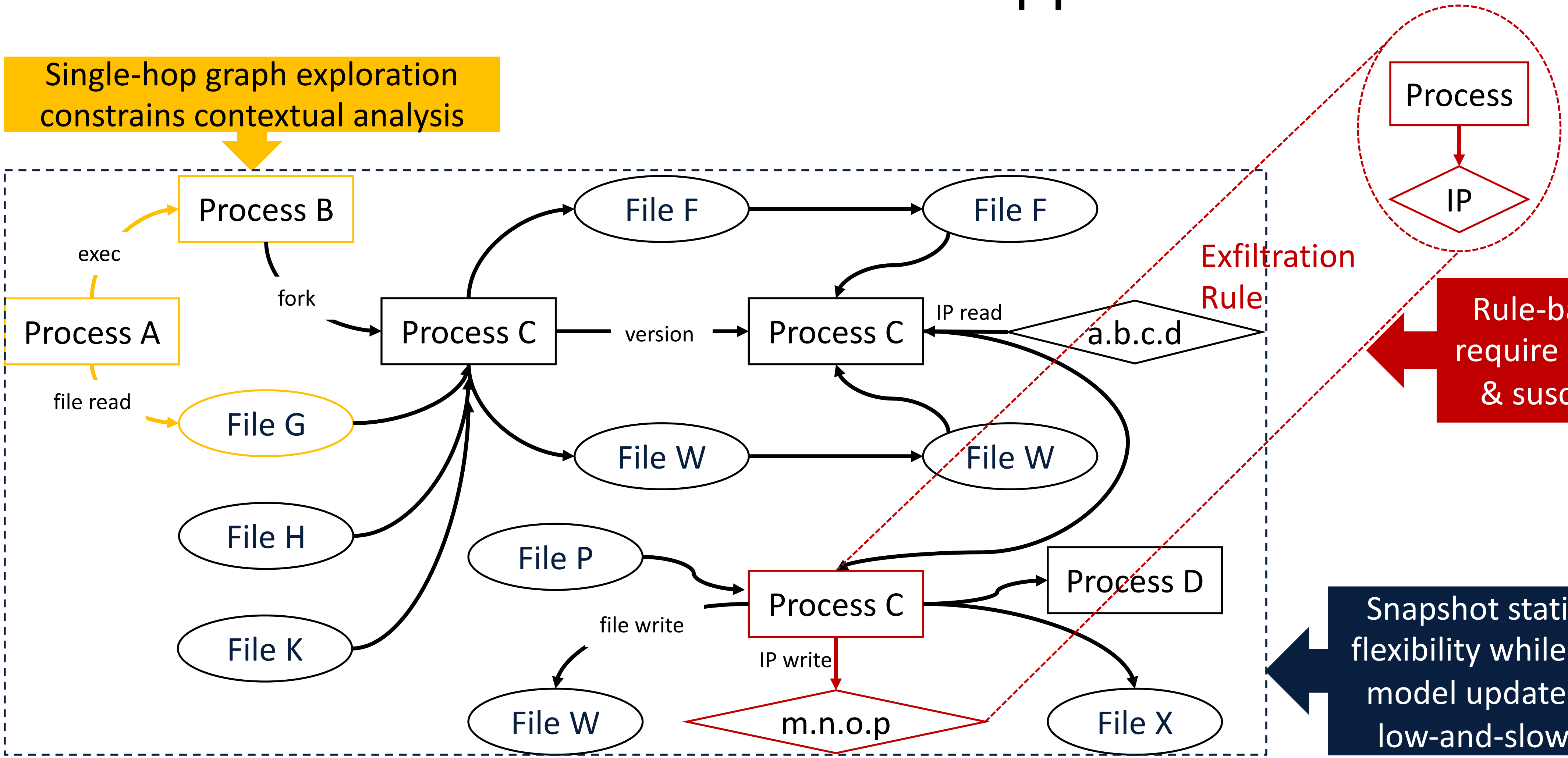
We use whole-system data provenance instead of traditional system call or log-adjacent system event analysis.

Full historical context of a system from a single, connected whole-system graph

Causal relationships among system subjects (e.g., process) and objects

Previous Provenance-Based Approaches

Single-hop graph exploration constrains contextual analysis



Rule-based approaches require expert knowledge & susceptible to 0-day

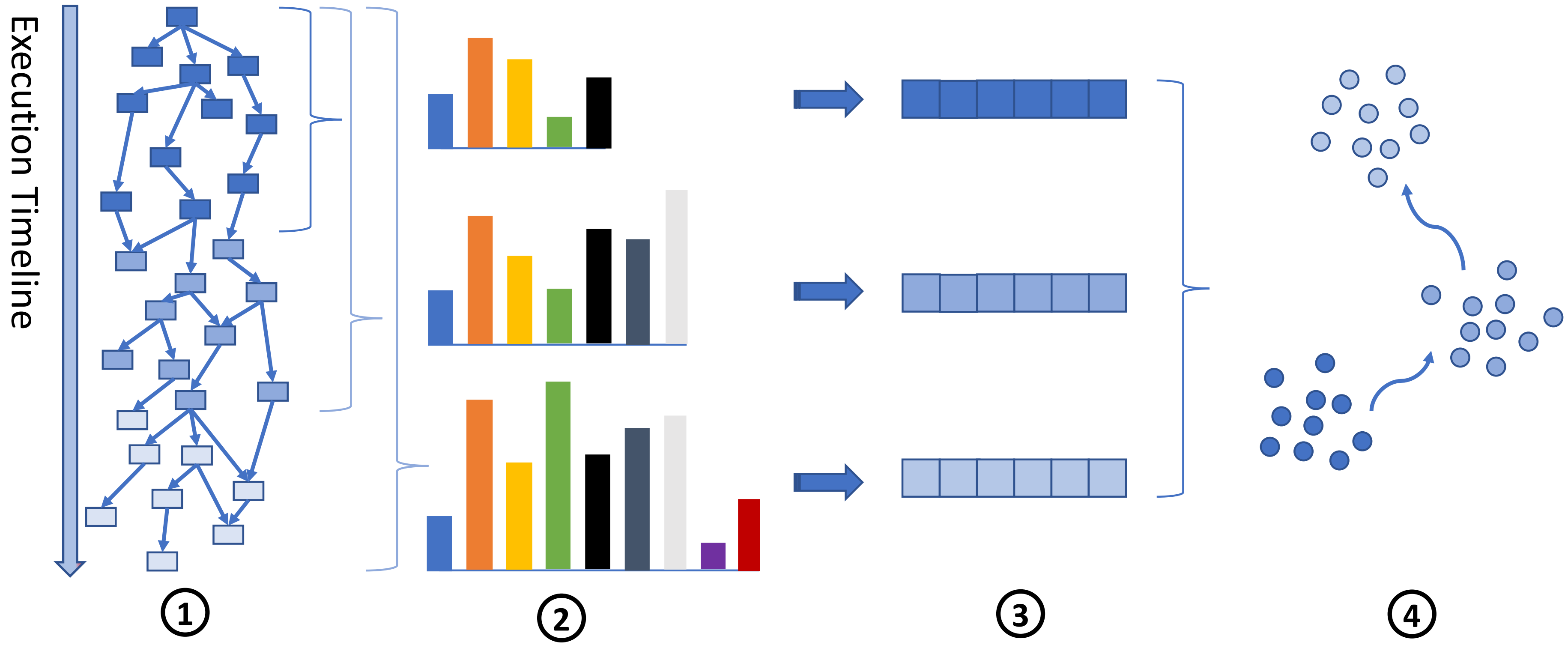
Snapshot static modeling lacks flexibility while runtime dynamic model update is unsuitable for low-and-slow attack patterns

UNICORN Goals

We formalize system-wide intrusion detection problem in APT campaigns as a *real-time, graph-based anomaly detection problem* on large, *attributed, streaming* whole-system provenance graphs.

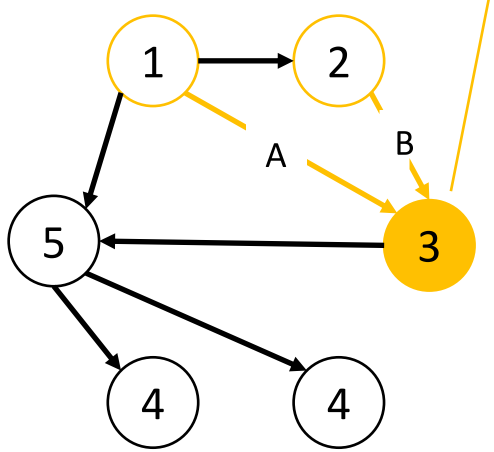
- Continuously analyze provenance graph with space and time efficiency while leveraging its rich historical context and system-wide causality relationships
- Consider the entire duration of system execution without making assumptions of attack behavior
- Learn only normal system behavior changes but not those directed by the attackers

UNICORN Overview



1. Takes as input a labeled, streaming provenance graph
2. Builds at runtime an in-memory graph histogram
3. Computes a fixed-size graph sketch periodically
4. Clusters sketches into a system model

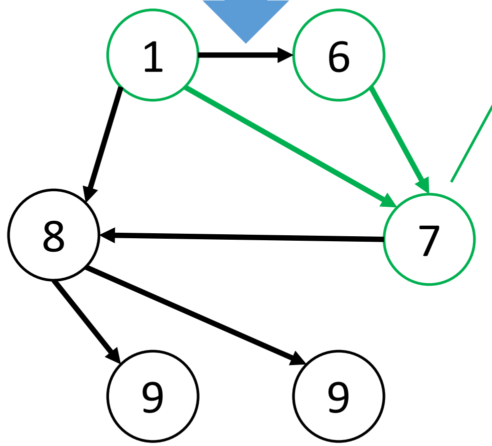
Graph Histogram



Iterative, vertex-centric,
Weisfeiler-Lehman label update:

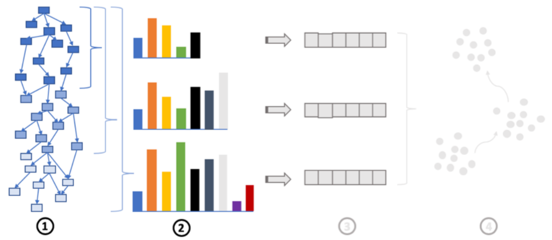
```
new_label = Hash(3, 1A2B)
histogram[new_label] += 1
```

Within the same iteration, every vertex is updated in parallel



In the next iteration, each vertex is updated again, exploring larger neighborhood:

```
new_label = Hash(7, 16)
histogram[new_label] += 1
```



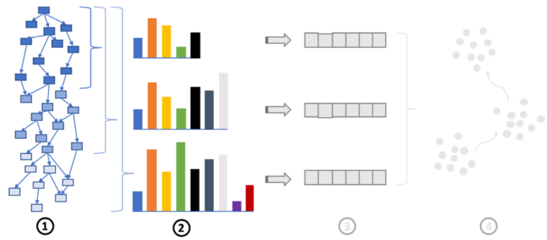
After `R` iterations:

- Each vertex explored R-hop neighborhood
  - Rich execution context
- `histogram` contains entire graph statistics
  - Full historical context

Efficient streaming variant:

- Leverage partial ordering guarantee from the provenance capture system

Discount Histogram for Concept Drift



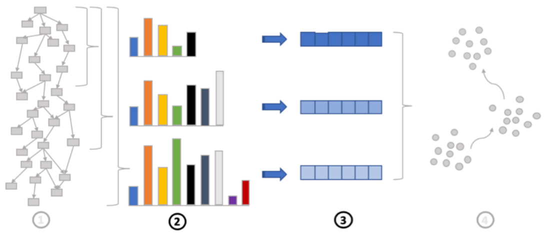
We model and monitor long-term system behavior, which often *changes over time*.

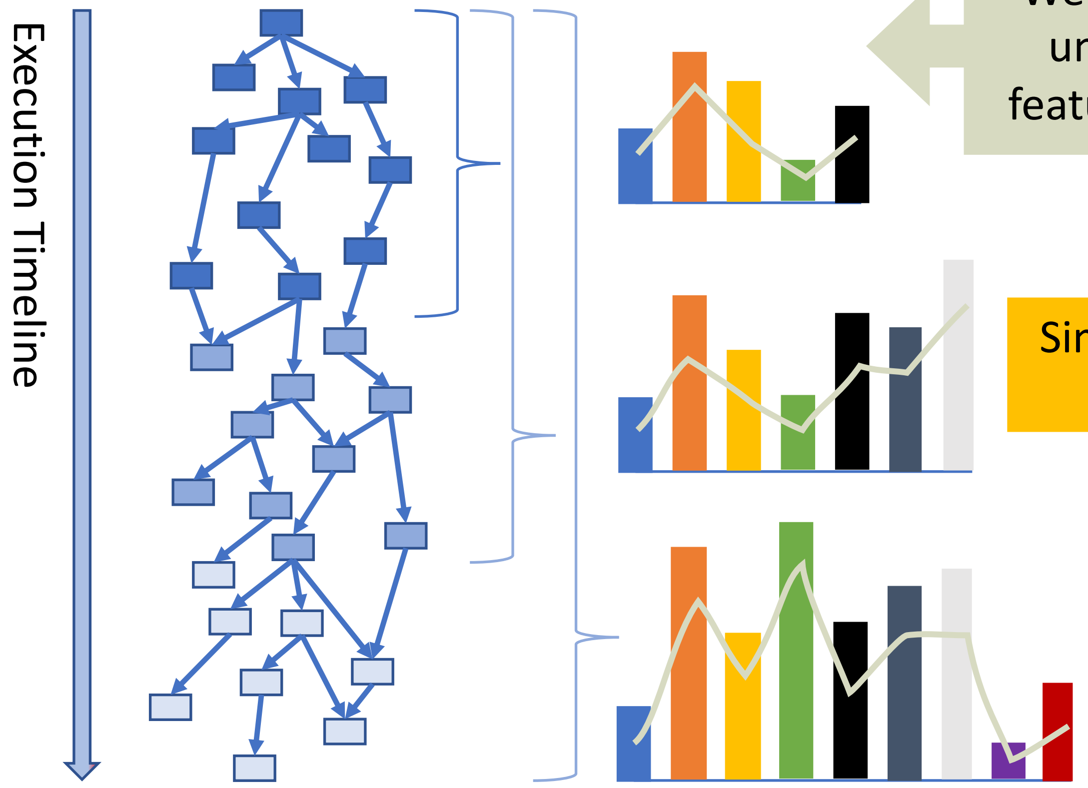
- Such changes result in changes in the underlying statistical properties of the histogram. This phenomenon is called **concept drift**.
- We use *exponential weight decay* to gradually forget outdated data.
  - Unicorn focuses on current system execution as well as elements that are *causally related* to current execution **even if they are temporally distant**.
  - Unicorn maintains fading "memory" of the past.

$$L_h = \sum_t w_t 1_{x_t = h}$$

Exponential decay:
$w_t = e^{-\lambda \Delta t}$
λ (decay factor) controls the rate of forgetting

Graph Sketch





We want to measure based on the underlying *distribution* of graph features, instead of absolute counts

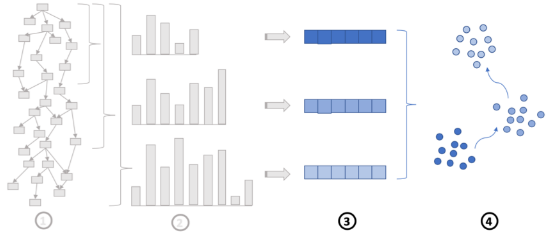
Similarity-Preserving Data Sketching

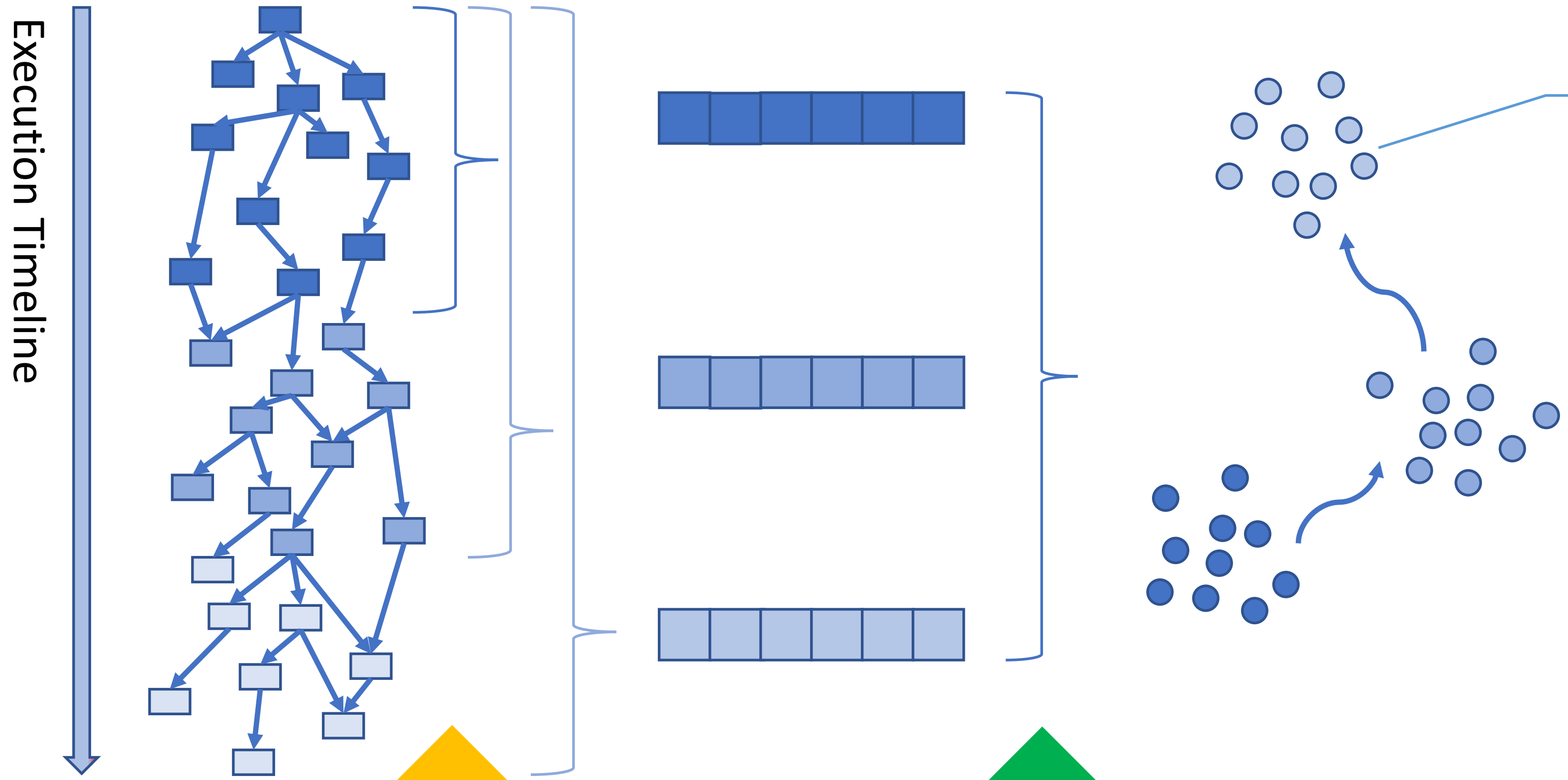
We employs `HistoSketch`:

- Hash histograms to compact, fixed-size sketch vectors
- Approximate histograms based on *normalized Jaccard similarity*
- Constant time algorithm to support real-time streaming
- Sketch size |S| controls tradeoffs between information loss and computation efficiency

In a streaming setting, # of histogram elements changes continuously

Evolutionary Model





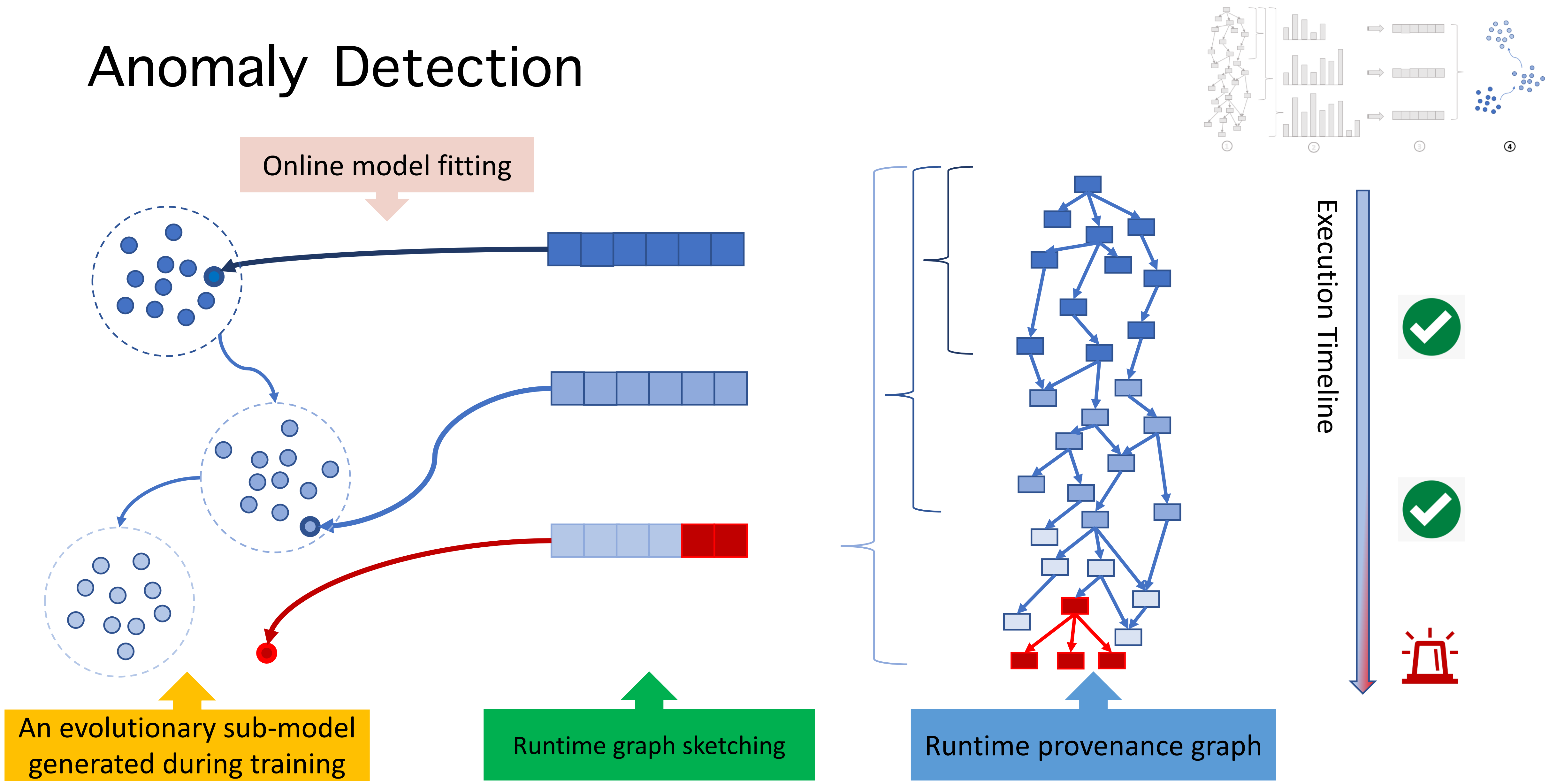
Periodic data sketching during model building

Clustering temporally-ordered sketches based on Jaccard similarity

Each cluster represents a ”meta-state” of system execution. We use those clusters and their statistics (e.g., diameter) to construct evolutionary model.

- With evolutionary modeling, UNICORN learns system behavior at *many points in time* during a single training execution trace.
- With gradually forgetting scheme, UNICORN focuses on the most relevant activities at each time point.

Anomaly Detection



Evaluation Datasets

- StreamSpot dataset: We compare UNICORN against a state-of-the-art provenance-based anomaly detection system StreamSpot using its published dataset
  - Can UNICORN outperform StreamSpot? If so, what are the factors?
- DARPA TC dataset: Data obtained during a red-team vs blue-team adversarial engagement with various provenance capture systems
  - Can UNICORN accurately detect anomalies in long-running systems?
  - Is the algorithm generalizable to different capture systems?
- Simulated supply-chain (SC) attack dataset: Our own controlled dataset using CamFlow whole-system provenance capture system
  - How do UNICORN's different design decisions affect APT detection?

StreamSpot dataset

Can UNICORN outperform StreamSpot? If so, what are the factors?

Experiment	Precision	Recall	Accuracy	F-Score
StreamSpot (baseline)	0.74	N/A	0.66	N/A
$R = 1$	0.51	1.0	0.60	0.68
$R = 3$	0.98	0.93	0.96	0.94

UNICORN's larger neighborhood exploration (R) improves precision/recall and outperforms StreamSpot.

StreamSpot creates snapshot-based static model and dynamically updates the model at runtime.

- Results in a significant number of false alarms, creating an opportune time window for attackers
- Persistent attackers can manipulate the model to gradually and slowly change system behavior to avoid detection
- UNICORN's evolutionary model reduces false positives (see paper) and prevents model manipulation

TC dataset

Can UNICORN accurately detect anomalies in long-running systems? Is the algorithm generalizable to different capture systems?

- DARPA'S 2-week long third adversarial engagement with datasets collected from a network of hosts running different audit systems
- Benign background activity generated from the red team allows us to model normal system behavior

Experiment	Precision	Recall	Accuracy	F-Score
DARPA CADETS	0.98	1.0	0.99	0.99
DARPA ClearScope	0.98	1.0	0.98	0.99
DARPA THEIA	1.0	1.0	1.0	1.0

High detection performance that accurately detects anomalies in long-running systems without prior attack knowledge

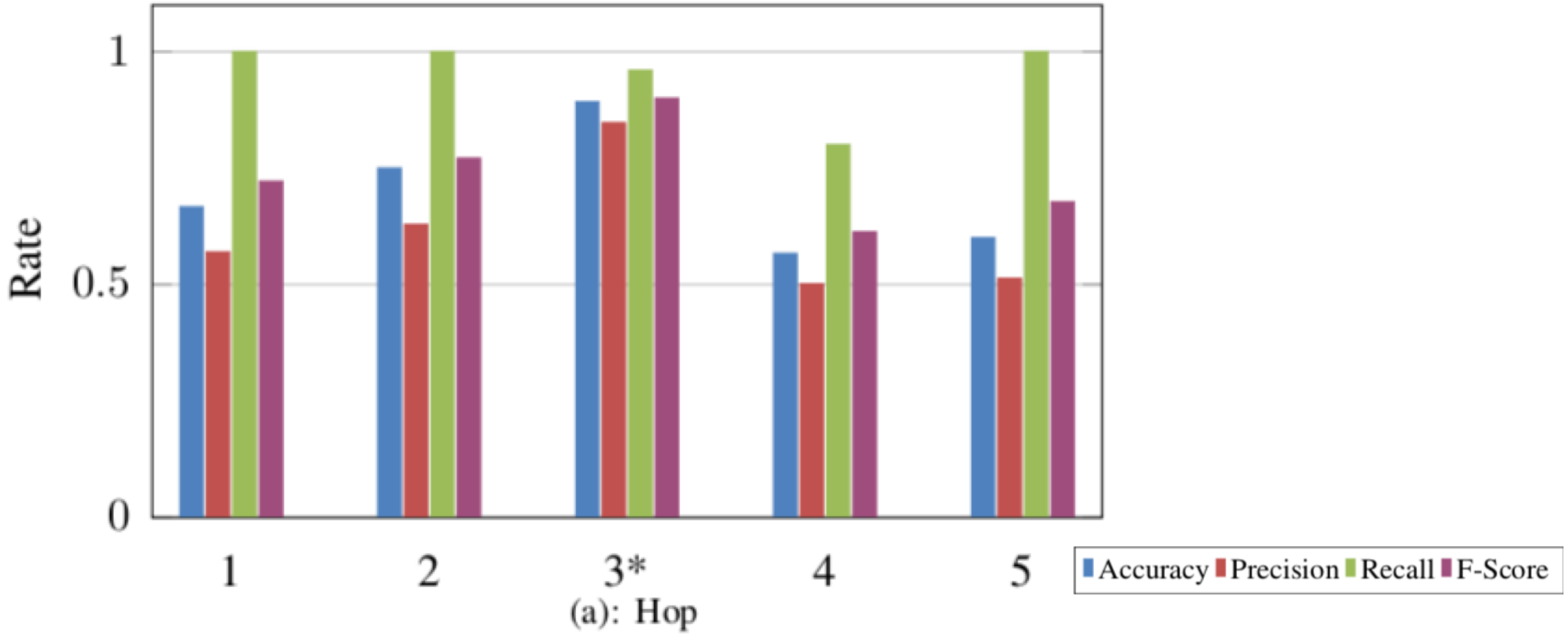
UNICORN's analytics framework generalizes to different capture systems and various graph structures.

SC attack dataset: Detection Performance

How do UNICORN's different design decisions affect APT detection?

We identify four important parameters that can affect detection performance:

- ❖ Hop count (R): size of neighborhood exploration
- ❖ Sketch size (|S|): size of fixed-size graph sketches
- ❖ Interval of sketch generation: how often we construct new graph sketches as the provenance graph grows during system execution
- ❖ Decay factor (λ): the rate at which we forget the past and focus on present execution

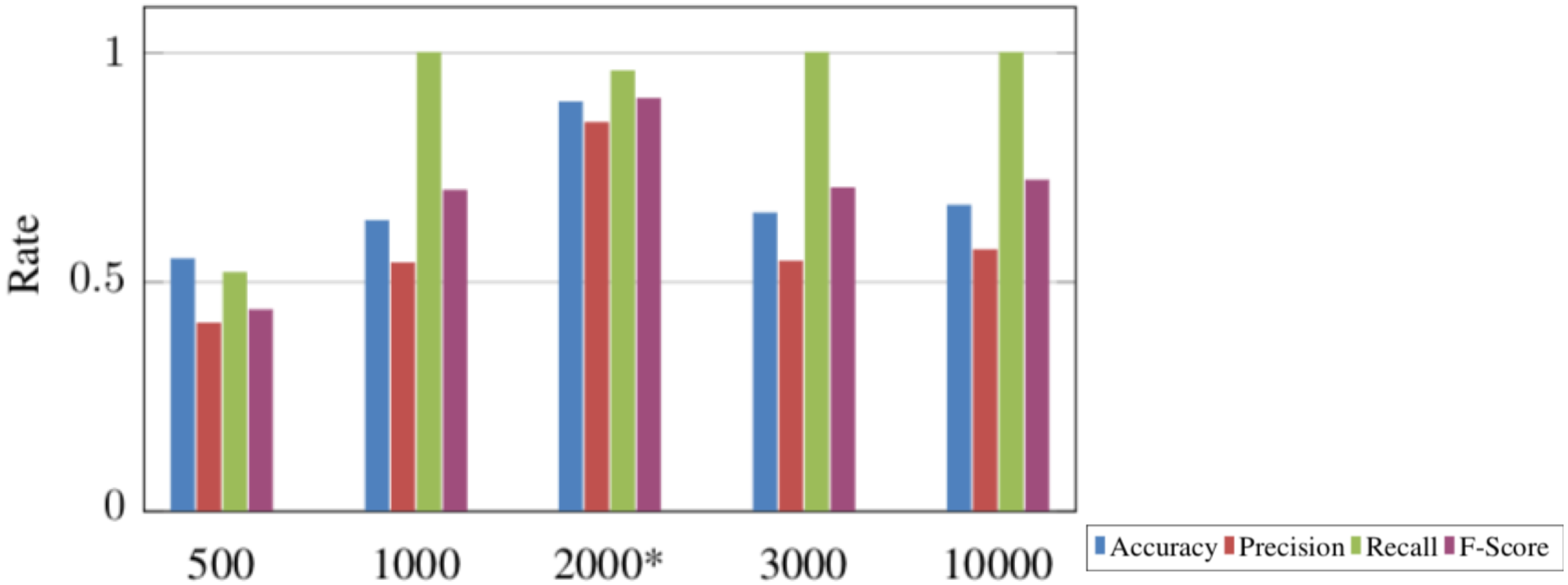


SC attack dataset: Detection Performance

How do UNICORN's different design decisions affect APT detection?

We identify four important parameters that can affect detection performance:

- Hop count (R): size of neighborhood exploration
- Sketch size (|S|): size of fixed-size graph sketches
- Interval of sketch generation: how often we construct new graph sketches as the provenance graph grows during system execution
- Decay factor (λ): the rate at which we forget the past and focus on present execution

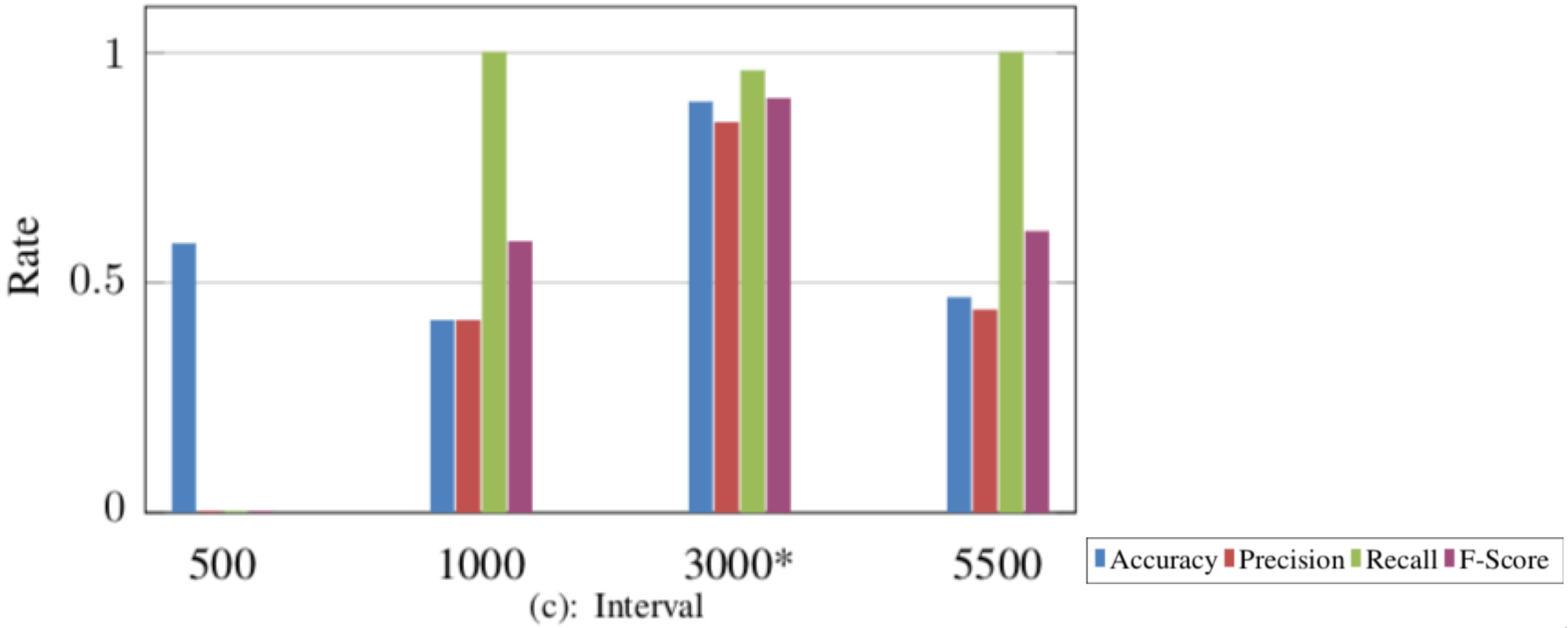


(b): Sketch

SC attack dataset: Detection Performance

How do UNICORN's different design decisions affect APT detection?

We identify four important parameters that can affect detection performance:
- ❖ Hop count (R): size of neighborhood exploration
- ❖ Sketch size (|S|): size of fixed-size graph sketches
- ❖ Interval of sketch generation: how often we construct new graph sketches as the provenance graph grows during system execution
- ❖ Decay factor (λ): the rate at which we forget the past and focus on present execution

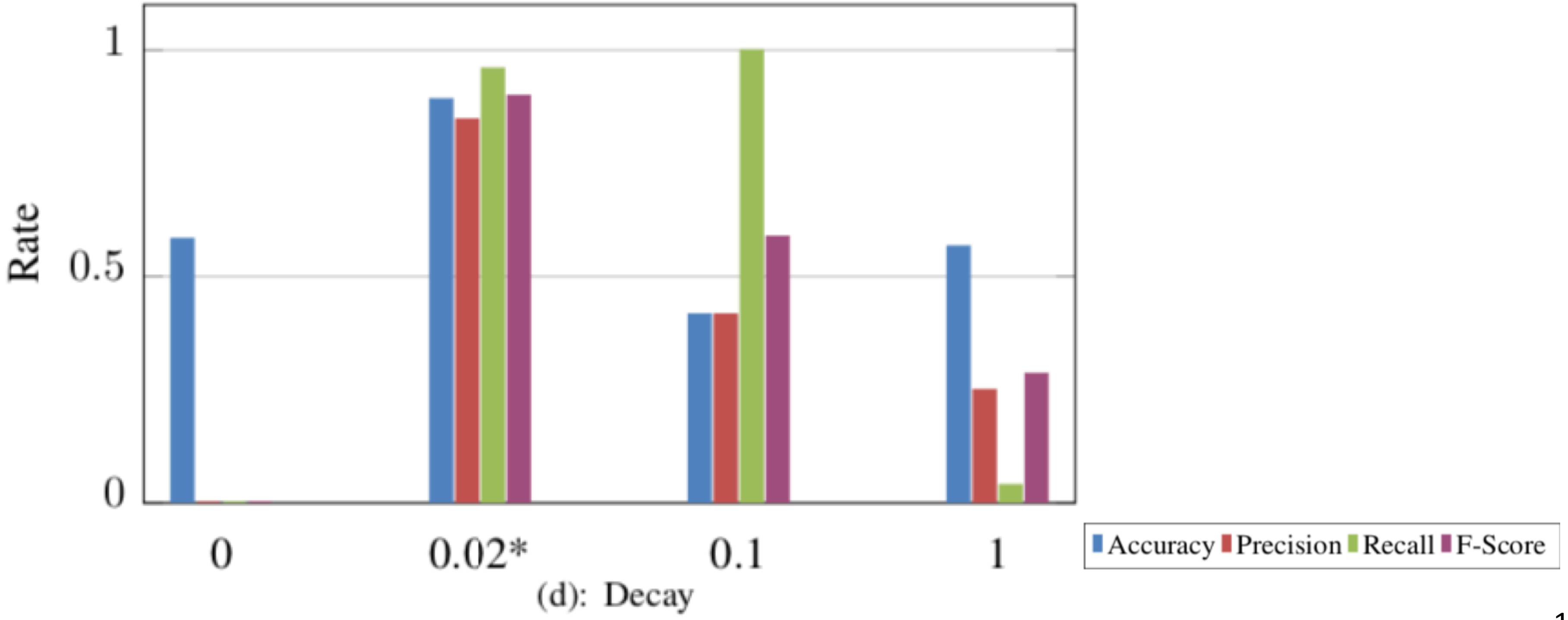


SC attack dataset: Detection Performance

How do UNICORN's different design decisions affect APT detection?

We identify four important parameters that can affect detection performance:

- ❖ Hop count (R): size of neighborhood exploration
- ❖ Sketch size (|S|): size of fixed-size graph sketches
- ❖ Interval of sketch generation: how often we construct new graph sketches as the provenance graph grows during system execution
- ❖ Decay factor (λ): the rate at which we forget the past and focus on present execution

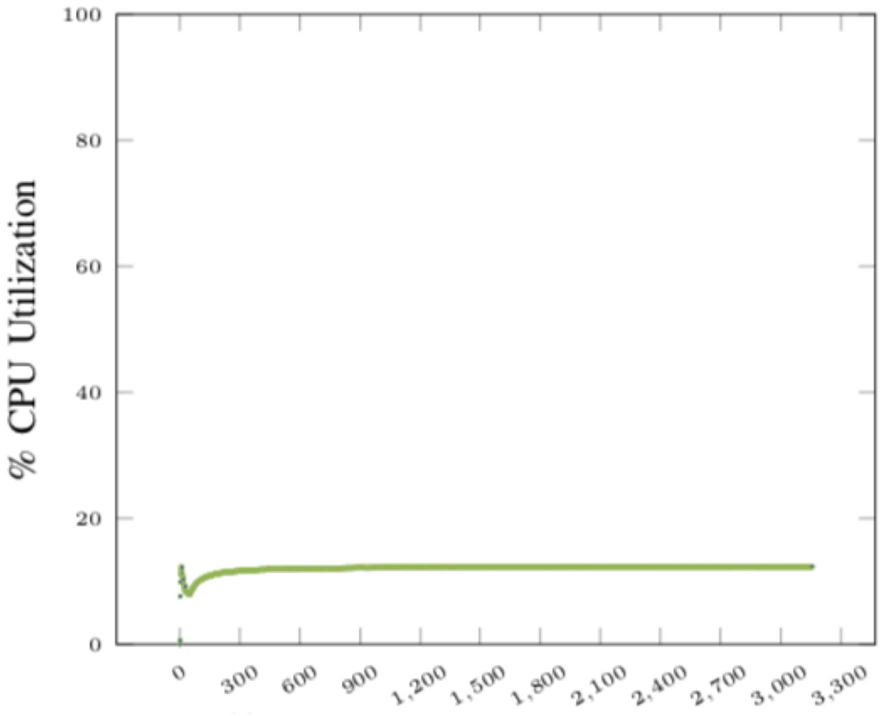


Runtime Performance

Hop count (R), sketch size (|S|), interval of sketch generation, and decay factor (λ) minimally affect UNICORN's ability to process the provenance graph as new edges arrive. We use **batching** to further improve its processing speed. This means UNICORN can perform real-time detection with parameters optimized for detection accuracy.

Configuration Parameter	Parameter Value	Max Memory Usage (MB)
Hop Count	R = 1	562
	R = 2	624
	R = 3	687
	R = 4	749
	R = 5	812
Sketch Size	$\lvert S\rvert$ = 500	312
	$\lvert S\rvert$ = 1,000	437
	$\lvert S\rvert$ = 2,000	687
	$\lvert S\rvert$ = 5,000	1,374
	$\lvert S\rvert$ = 10,000	2,498

Memory usage depends on hop count and sketch size, but empirically large R and |S| are not ideal for detection performance.



Average CPU stabilizes around 12.3% on a single CPU regardless of parameter settings.

Discussion & Conclusion

- UNICORN is a real-time provenance-based anomaly detector that efficiently analyze system-wide data provenance for APT attacks.
- UNICORN leverages graph sketching to build an incrementally updatable, fixed-size, longitudinal graph data structure to enable online, streaming analysis.
- Anomaly-based detection requires a “good” set of benign behavior to learn from, can be susceptible to evasion techniques, and needs human-in-the-loop to verify FPs and update the model.
- Reasoning about anomaly alerts (forensics) can be difficult and requires additional tools.

Q & A

UNICORN: Runtime Provenance-Based Detector for Advanced Persistent Threats

AUTHORS:

Xueyuan Han (presenter), Thomas Pasquier, Adam Bates, James Mickens, and Margo Seltzer

PROJECT REPO:

https://github.com/crimson-unicorn

Thank you for your time and attention!

HARVARD UNIVERSITY