AI for Detecting Advanced Cyberattacks

January 19th 2026

Tristan Bilot, Ph.D

Who am I?

Tristan Bilot
tristan.bilot@universite-paris-saclay.fr

- **EPITA** (promo 2022)
- PhD **Paris-Saclay** (2025)
- Research Intern at **AWS** (2025)
- Incoming Postdoc at the **University of British Columbia** (UBC)

Outline

1. Detecting Traditional Attacks
2. Detecting Unknown Attacks
3. Limitation of Current Detection Systems
4. Detecting APTs at the Node Level
5. Building More Practical Systems
6. Implementing an IDS From Scratch
7. A Framework for Designing Neural Network-based IDSs

Detecting Traditional Attacks

Some Well-detected Attacks

- **DoS/DDoS**
- **Brute force login**
- **Reusing existing exploits/tools**
- **Unobfuscated Malware**

Traditional Detection Approaches

- **Signature-based detection**: compares patterns with signature hashes of existing attacks

```
IF (MD5(file) == 44d88612fea8a8f36de82e1278abb02f)
    THEN report “Malicious File Detected"
```

- **Rule-based detection**: compares patterns with manually defined rules that describe attacks

```
IF (file_extension == ".exe")
    AND (source == "email")
        THEN report "Suspicious File Detected"
```

Traditional Detection Approaches

- **Signature-based detection**: compares patterns with signature hashes of existing attacks

```
IF (MD5(file) == 44d88612fea8a8f36de82e1278abb02f)
    THEN report “Malicious File Detected"
```

- **Rule-based detection**: compares patterns with manually defined rules that describe attacks

```
IF (file_extension == ".exe")
    AND (source == "email")
        THEN report "Suspicious File Detected"
```

Traditional Detection Approaches

Traditional detection methods are the core of most **SIEMs** and **XDRs**:

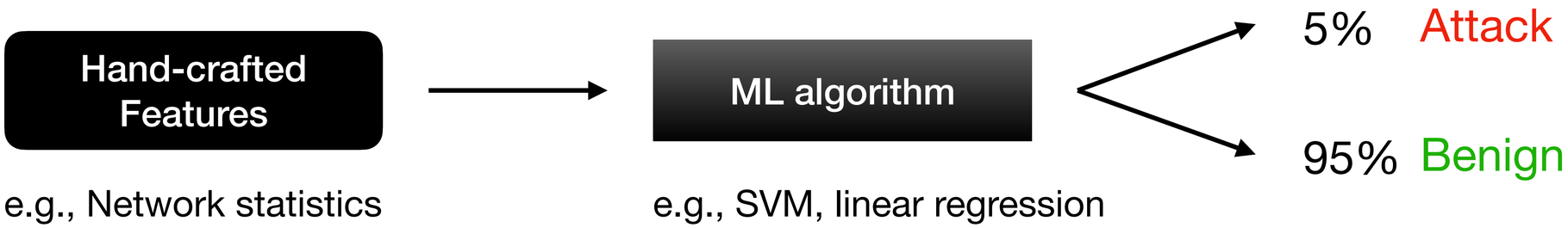
- **Microsoft Sentinel**
- **Splunk Enterprise Security**
- **CrowdStrike Falcon…**

Even in the cloud:

- **AWS GuardDuty**
- **Azure Sentinel…**

Supervised Machine Learning Approaches

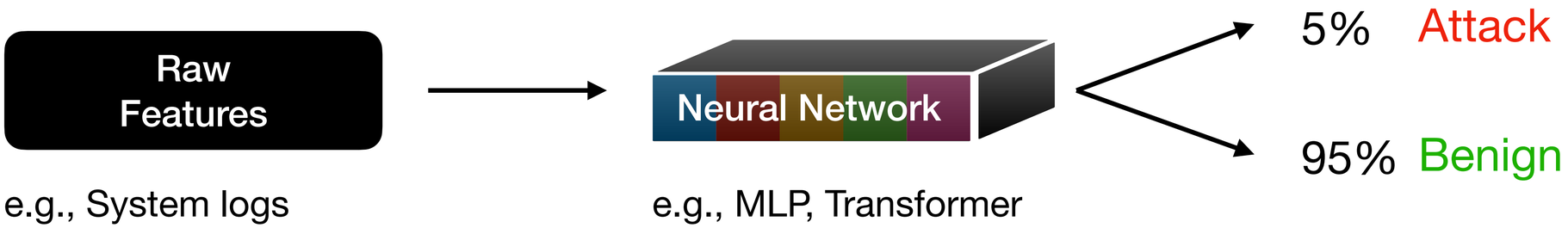
Machine Learning-based detection



Goal: let the model learn the rules by itself → more flexible.

Supervised Deep Learning Approaches

Deep Learning-based detection



Goal: let the model learn the most useful features by itself.

Limitations of Current Approaches

- **Traditional approaches** → *crafted* from known attacks
- **Supervised ML Approaches** → *learned* from known attacks

What if the attack is not known?

Limitations of Current Approaches

Current Detection Approaches

- Signature not found
- Does not trigger any rule
- Example never learned by the ML model

Advanced Persistent Threats (APTs)

- **Prolonged** and **targeted** attacks
  - From hacker groups or **nation-state actors**
  - Targeting **governments** or **large organizations**
- **74%** increase in 2024 [4]
- Detected in **25%** of organizations [4]

[4] Kaspersky Report (2025), https://securitybrief.asia/story/advanced-persistent-threats-rise-by-74-in-2024-report.

Advanced Persistent Threats (APTs)

- **60%** are attributed to states[6]
- **89%** are associated to espionage[5]

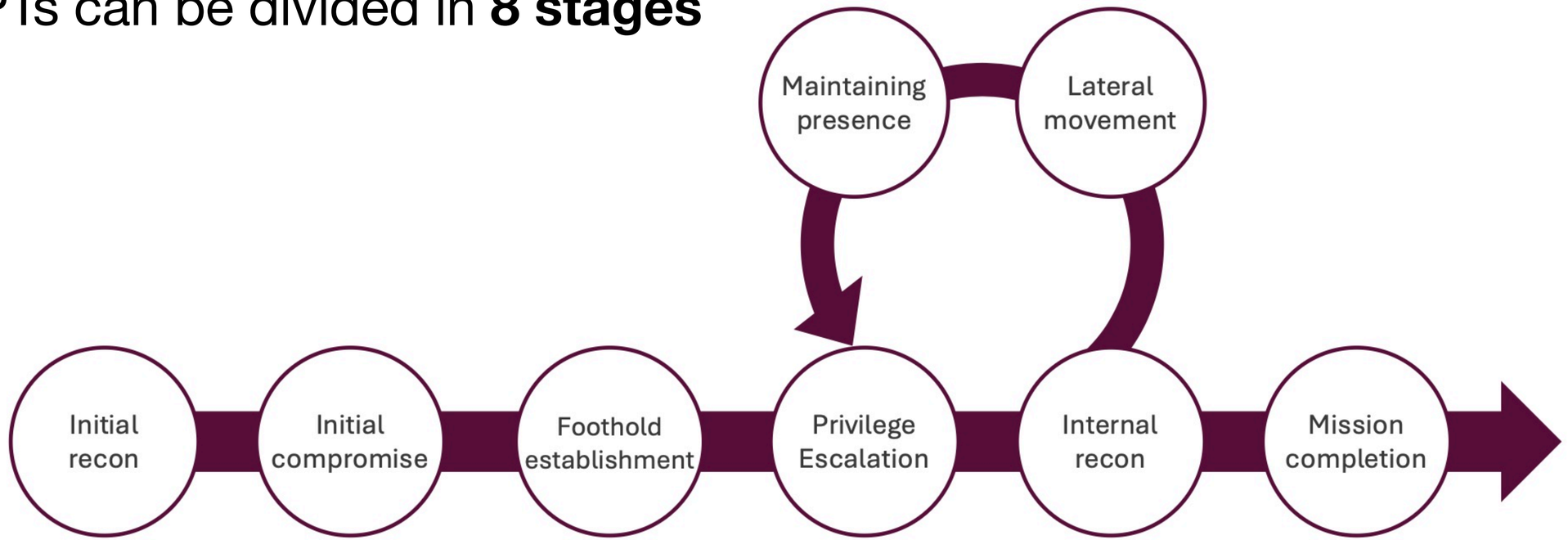
Famous APTs

- 🇷🇺 **APT28**: Fancy Bear → data leaks during U.S. elections (2016)
- 🇰🇵 **APT38**: Lazarus Group → WannaCry ransomware (2017)

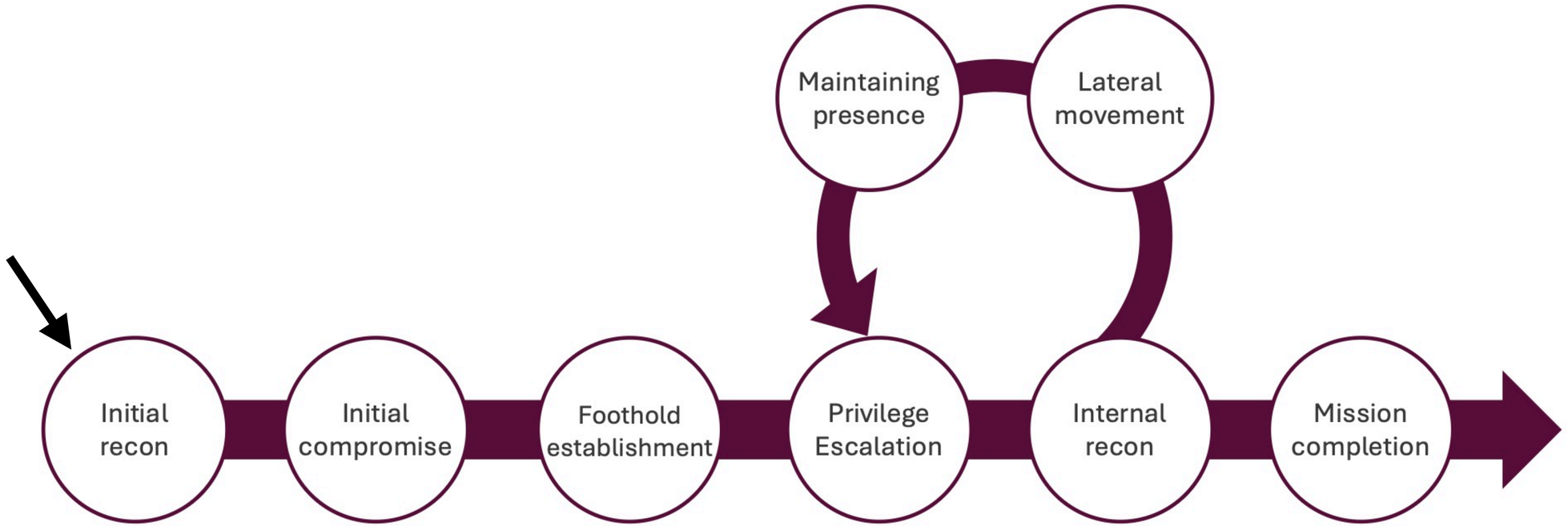
[5] Vectra AI (2025), https://fr.vectra.ai/topics/advanced-persistent-threat.
[6] CrowdStrike Global Threat Report (2025), https://go.crowdstrike.com/2025-global-threat-report.html.

Advanced Persistent Threats (APTs)

APTs can be divided in **8 stages**

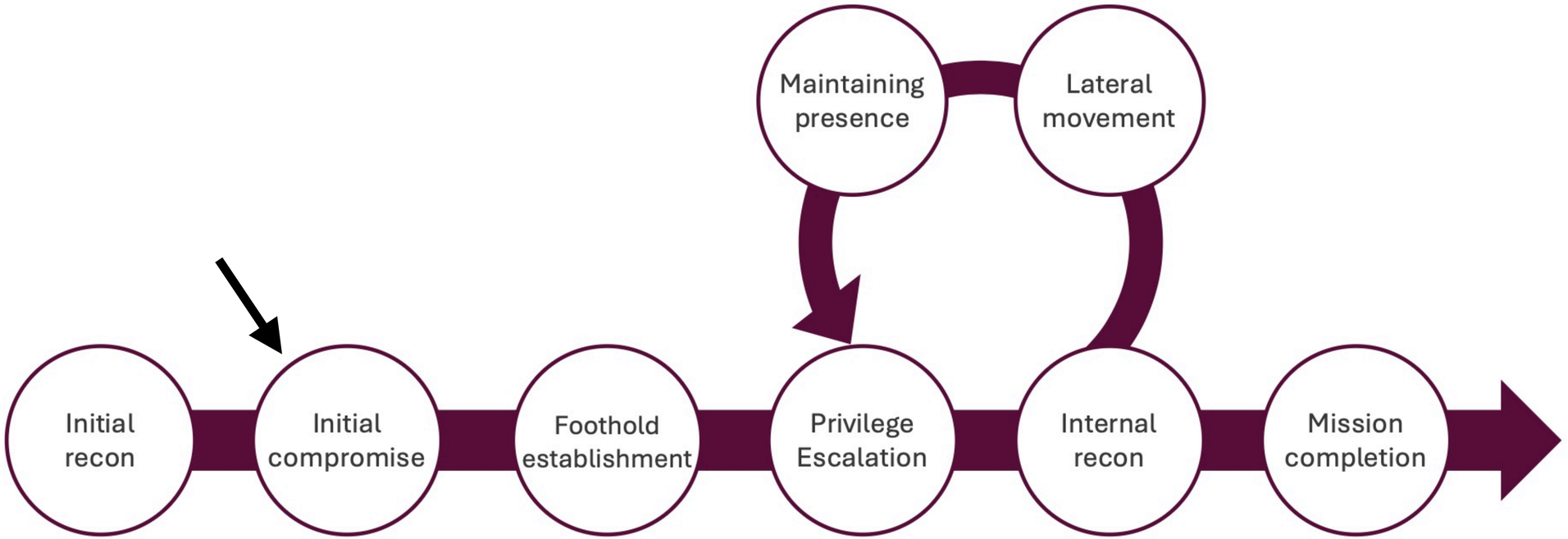


Advanced Persistent Threats (APTs)



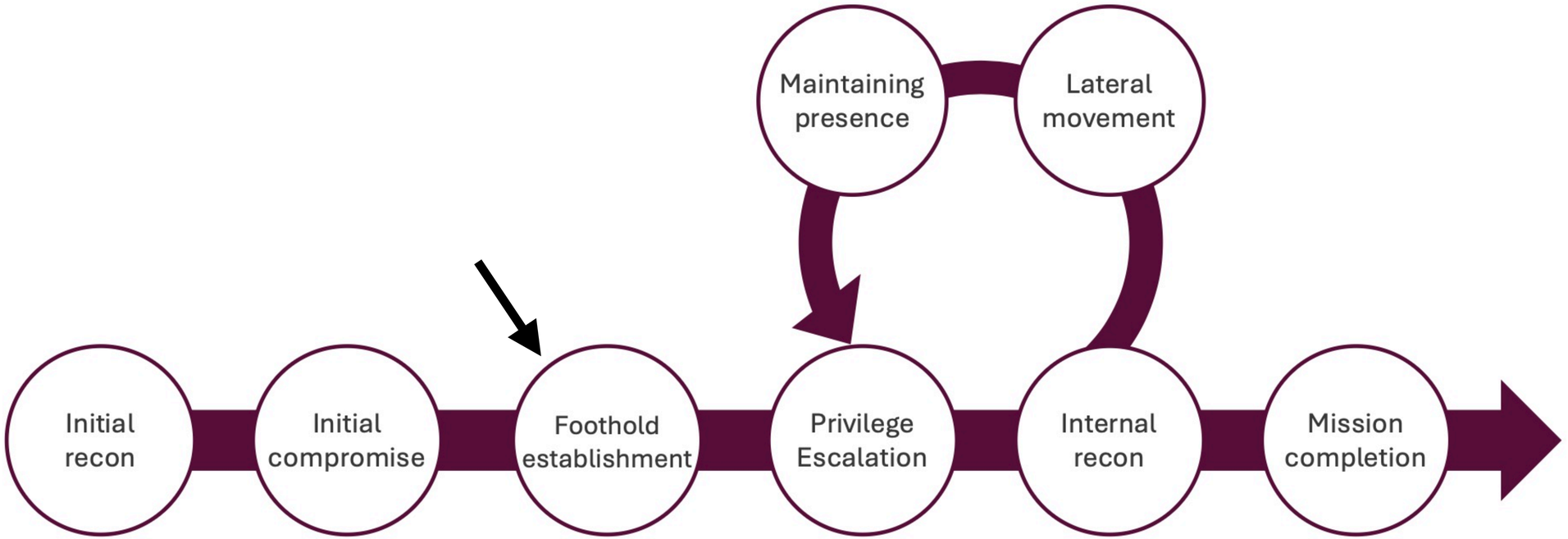
1. **Initial Reconnaissance**: The attacker studies the target's systems and employees to plan the intrusion, often identifying remote access points or social engineering targets.

Advanced Persistent Threats (APTs)



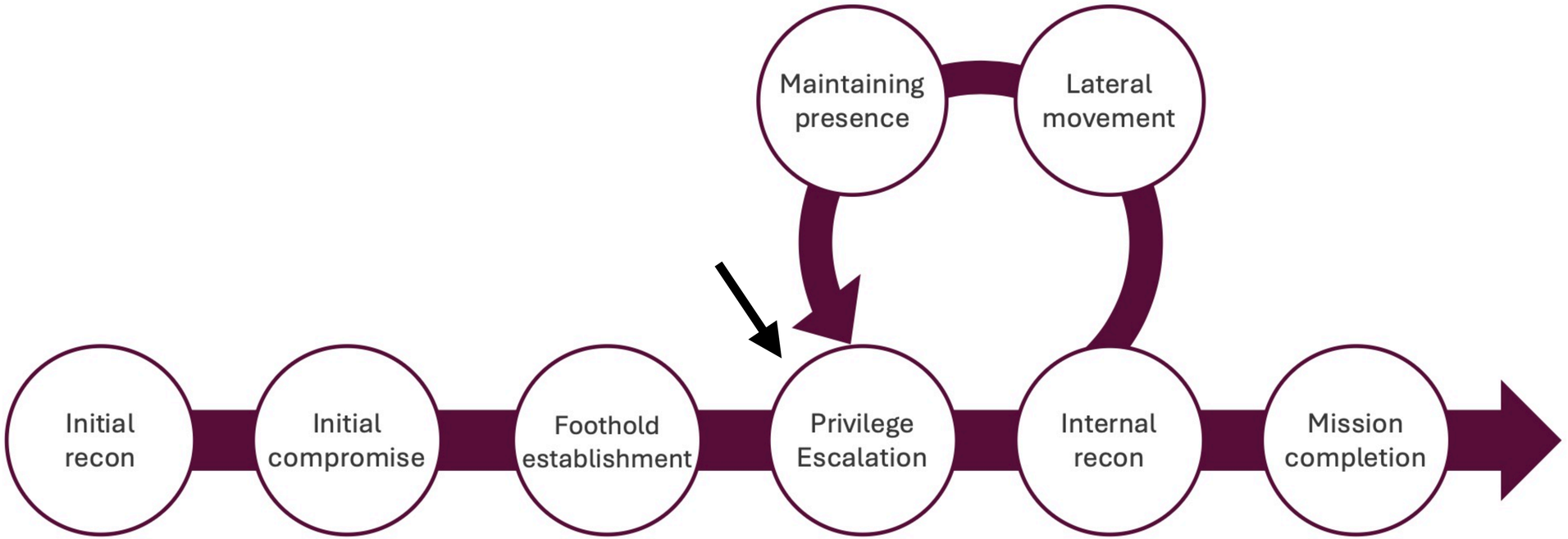
2. **Initial Compromise**: Malicious code is executed on one or more systems, typically via social engineering or exploiting internet-facing vulnerabilities.

Advanced Persistent Threats (APTs)



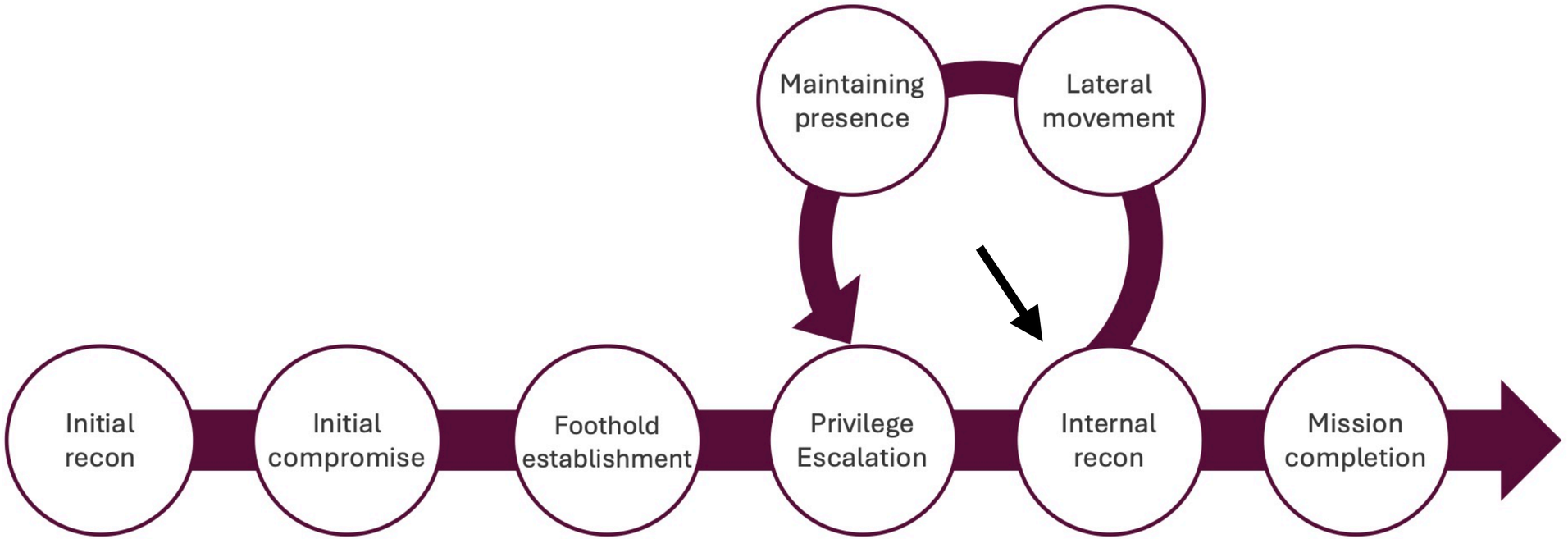
3. **Foothold Establishment**: The attacker maintains control by installing persistent backdoors or downloading tools on the compromised system.

Advanced Persistent Threats (APTs)



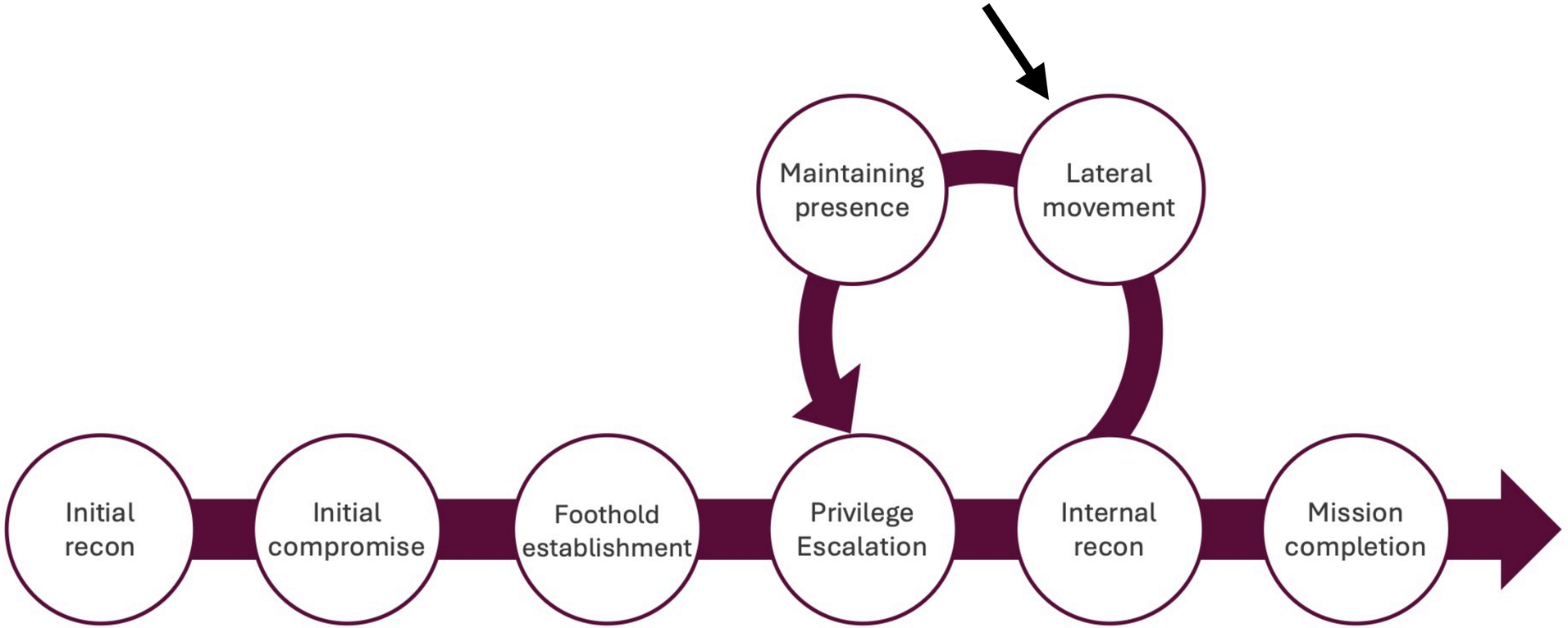
4. **Privilege Escalation**: The attacker gains higher-level access, often by harvesting credentials, logging keystrokes, or bypassing authentication.

Advanced Persistent Threats (APTs)



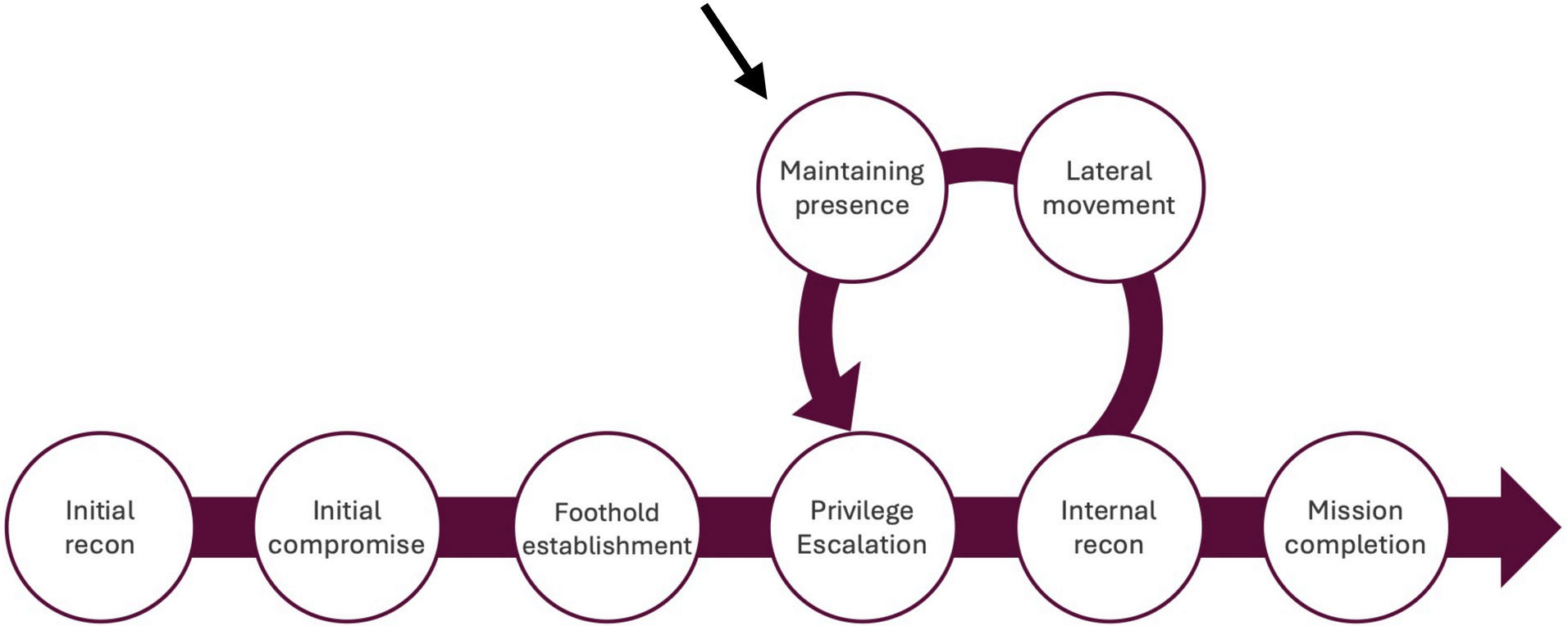
5. **Internal Reconnaissance**: The attacker scans the internal network to locate valuable data and understand system roles.

Advanced Persistent Threats (APTs)



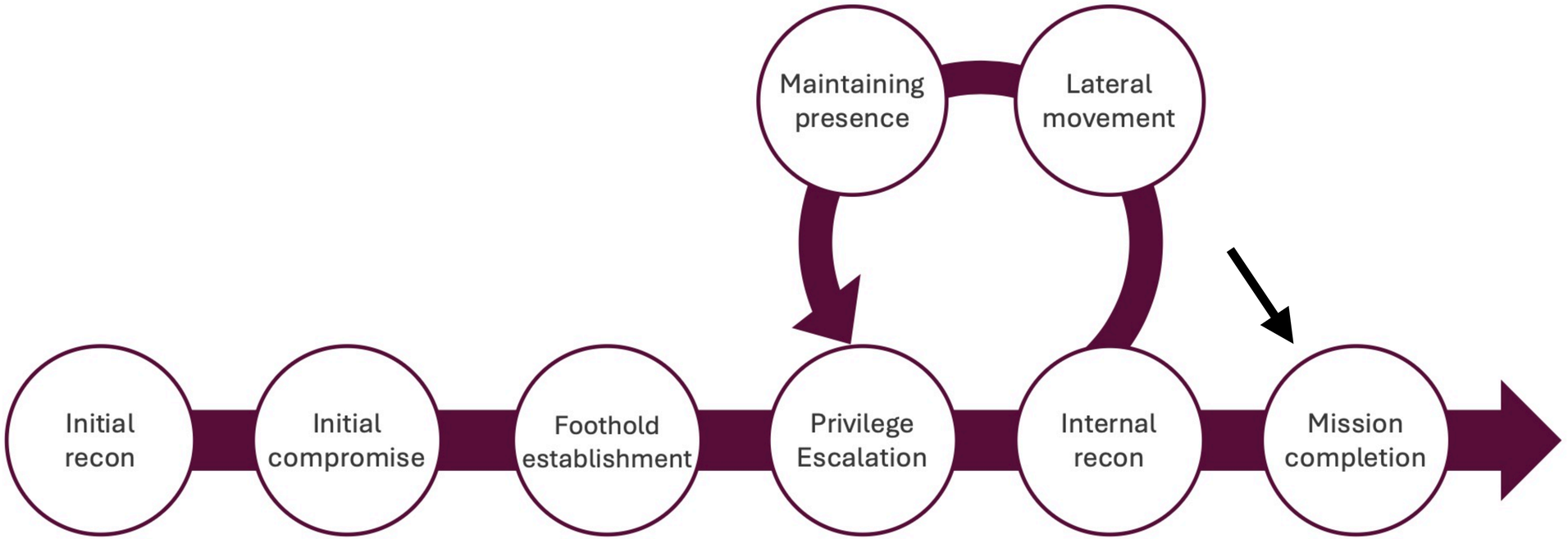
6. **Lateral Movement**: Using stolen credentials, the attacker moves to other systems via file shares, remote commands, or Secure Shell (SSH) access.

Advanced Persistent Threats (APTs)



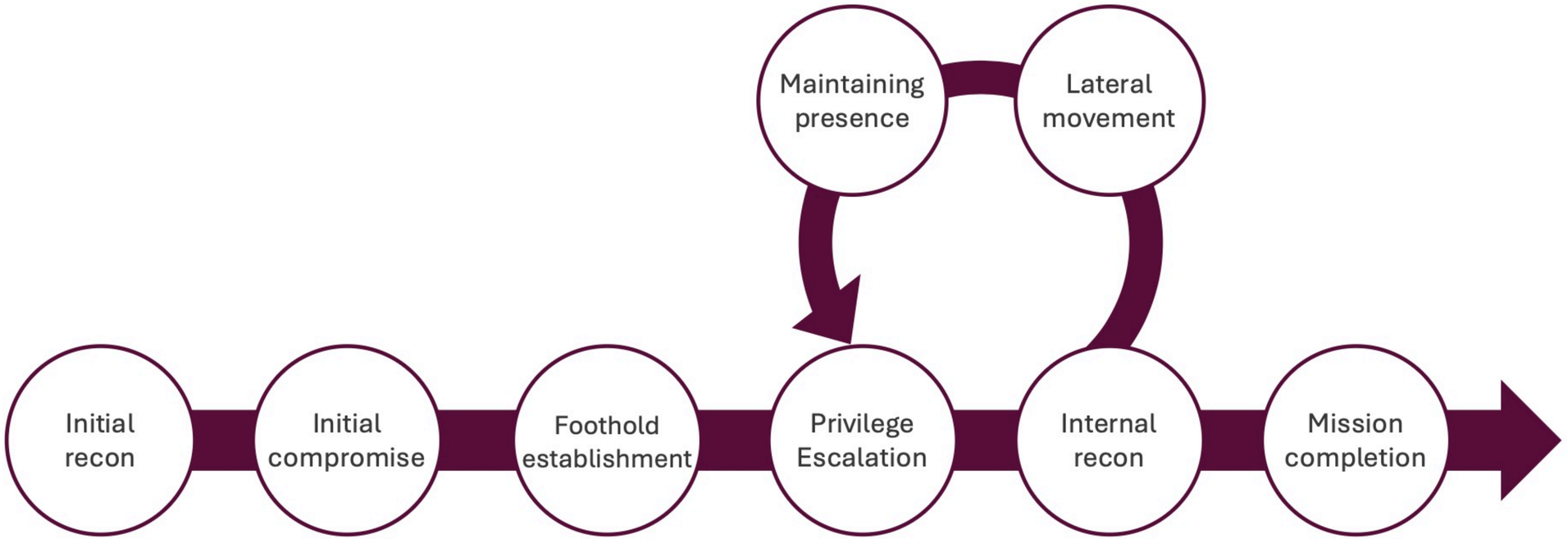
7. **Maintaining Presence**: Continued access is secured through multiple backdoors or remote access services like Virtual Private Networks (VPNs).

Advanced Persistent Threats (APTs)



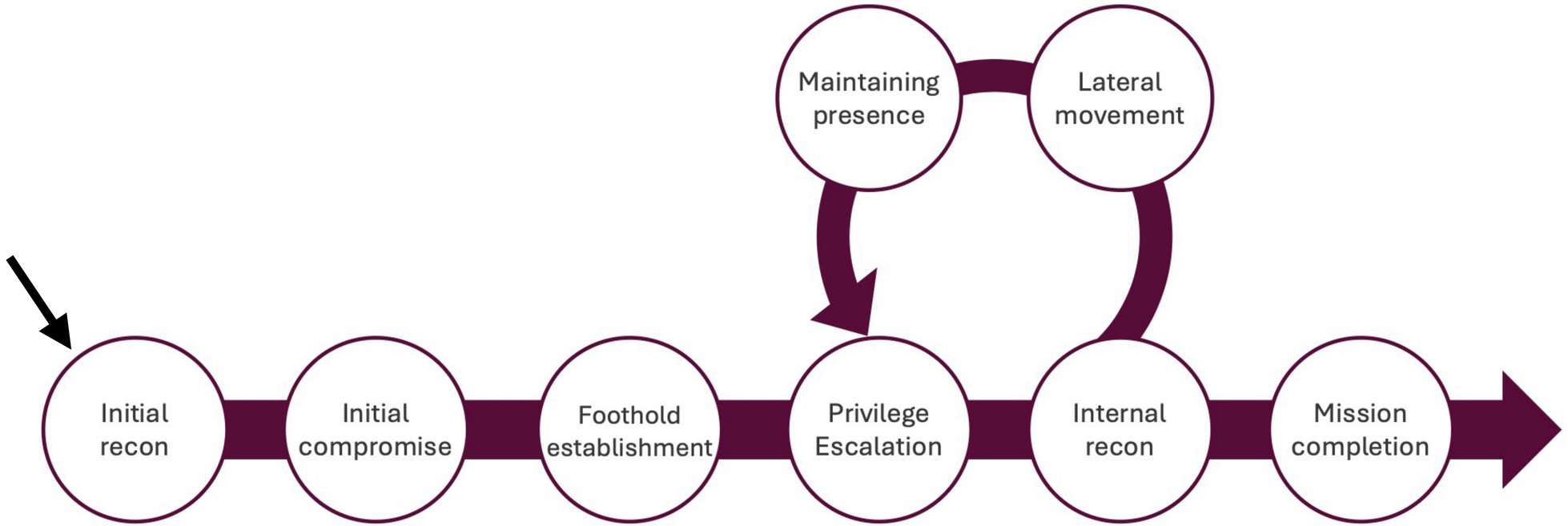
8. **Mission Completion**: The attacker completes objectives such as stealing data, disrupting systems, or destroying assets.

Case Study: APT1



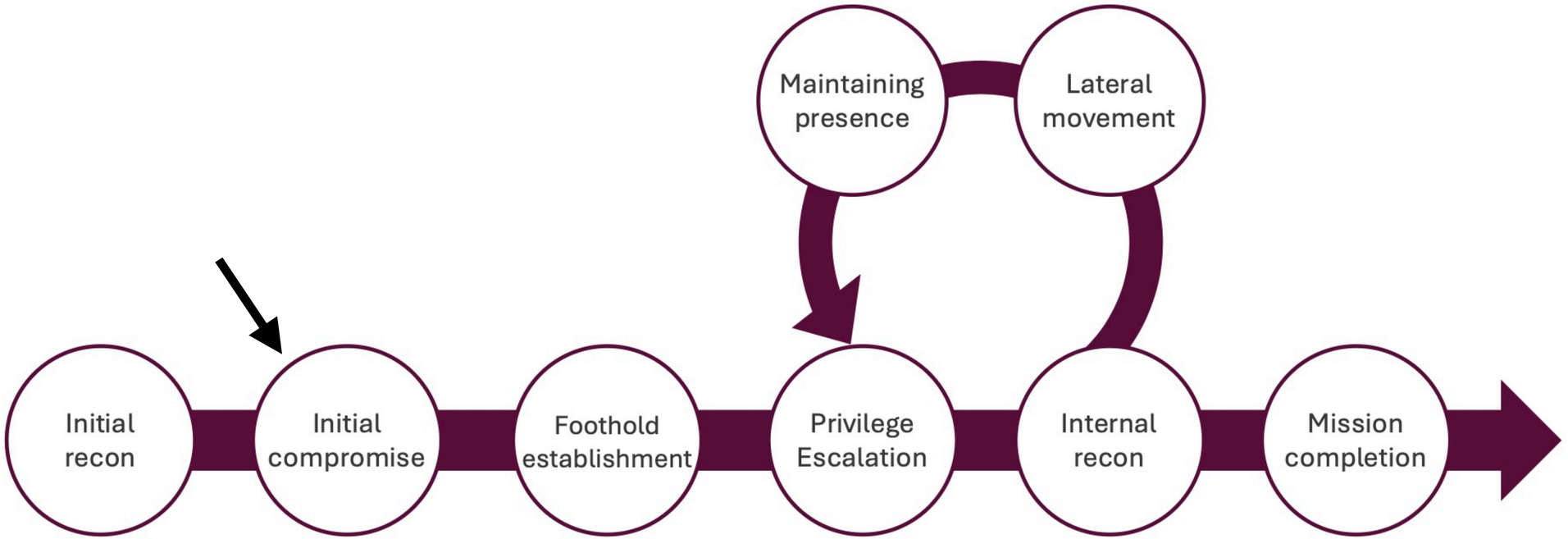
🇨🇳 **APT1** (Unit 61398): attributed to the Chinese People's Liberation Army, first exposed in **2013**

Case Study: APT1



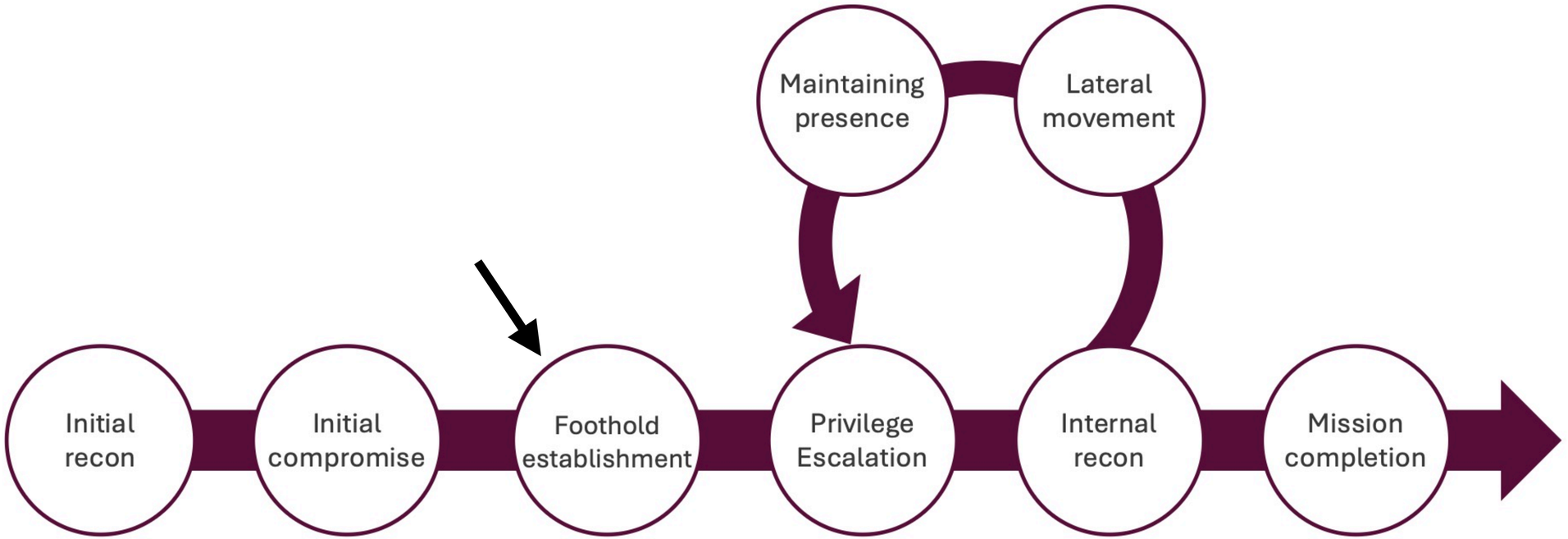
1. **Initial Reconnaissance**: APT1 conducted extensive open source intelligence (OSINT) by gathering employee emails, organizational charts, and publicly exposed systems to prepare highly targeted spear-phishing campaigns.

Case Study: APT1



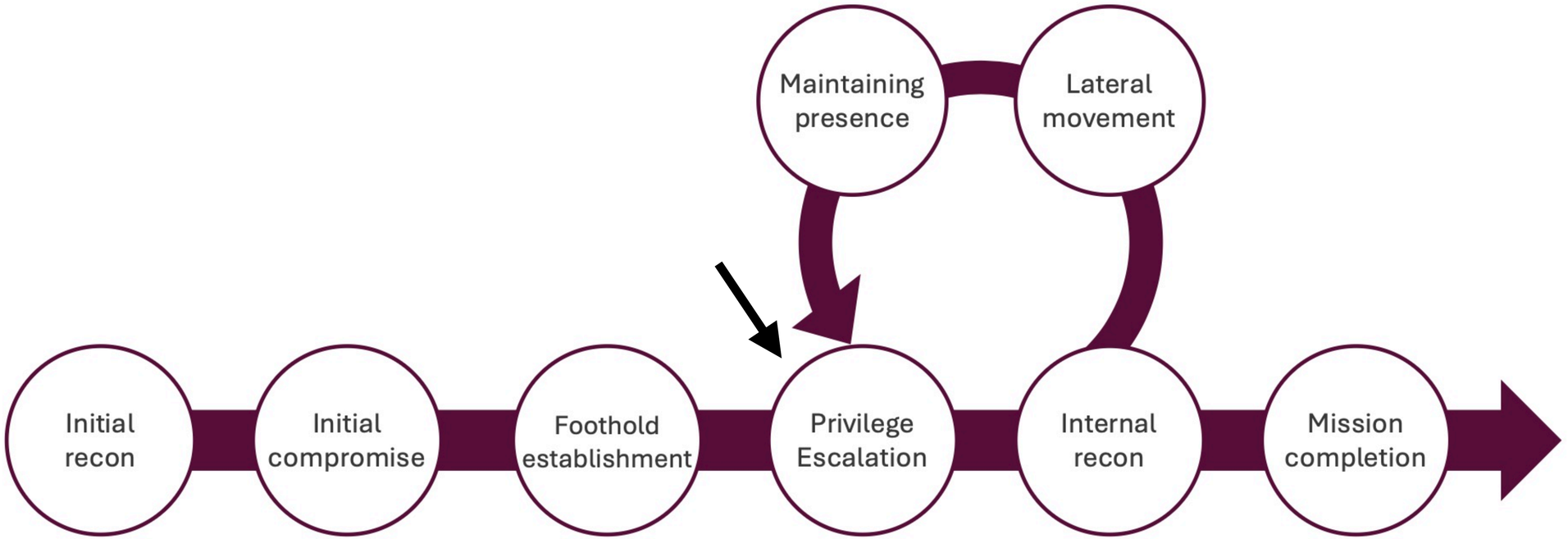
2. **Initial Compromise**: Their primary entry method was spear-phishing emails with plausible subject lines and attachments, which, when opened, deployed custom backdoors like WEBC2.

Case Study: APT1



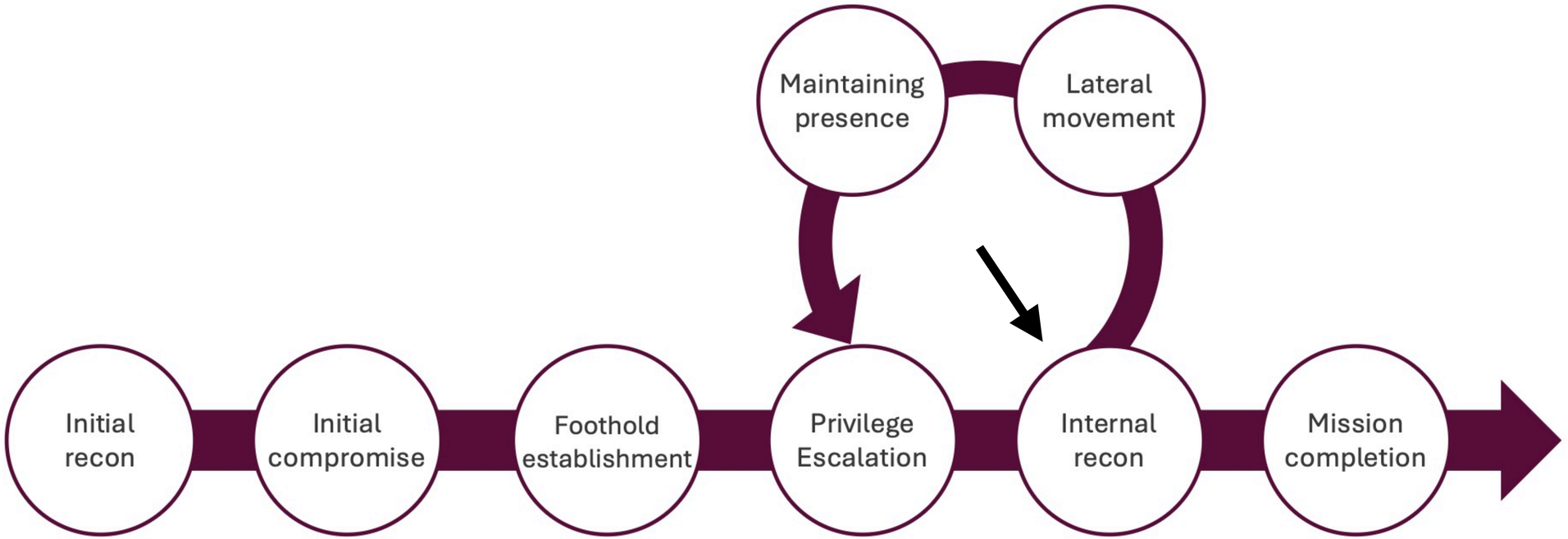
3. **Foothold Establishment**: After delivering malware, APT1 installed remote access Trojans (RATs) or custom implants to maintain persistence, establishing outbound Command and Control (C2) connections.

Case Study: APT1



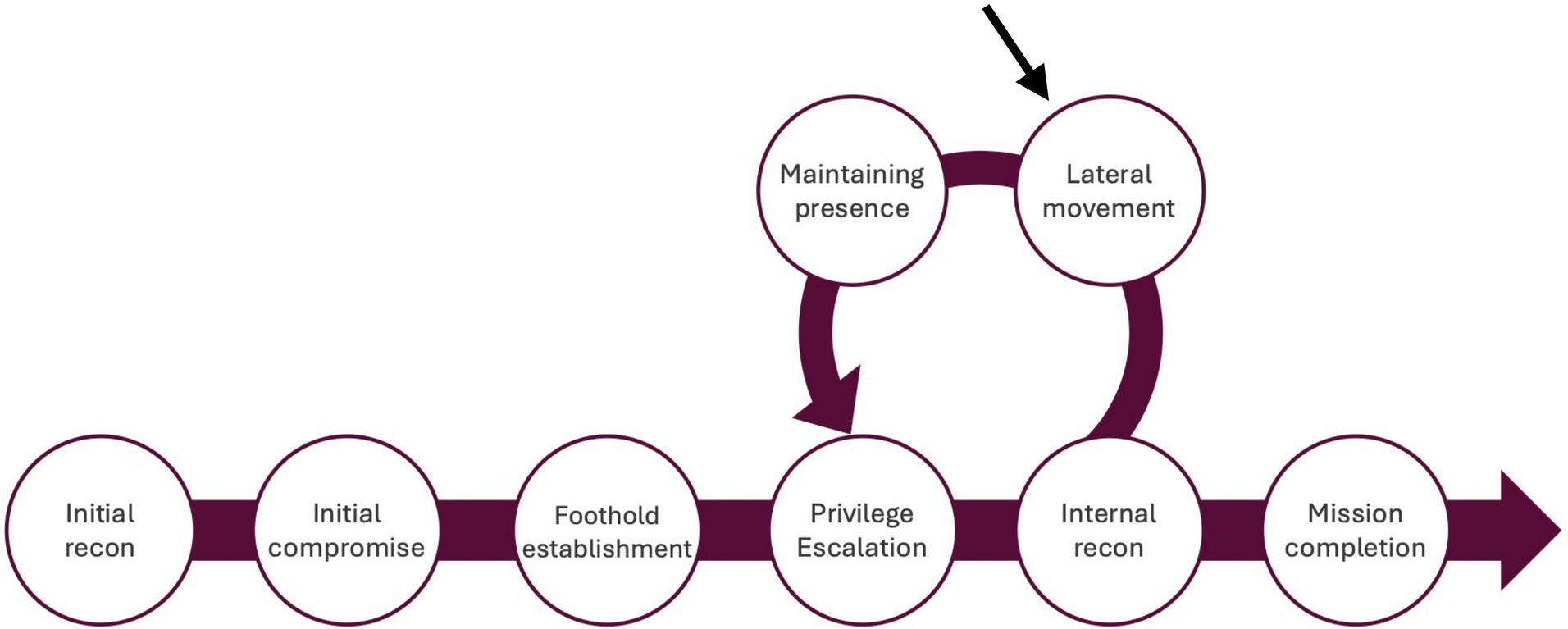
4. **Privilege Escalation**: They harvested credentials using tools like Mimikatz, enabling administrator access.

Case Study: APT1



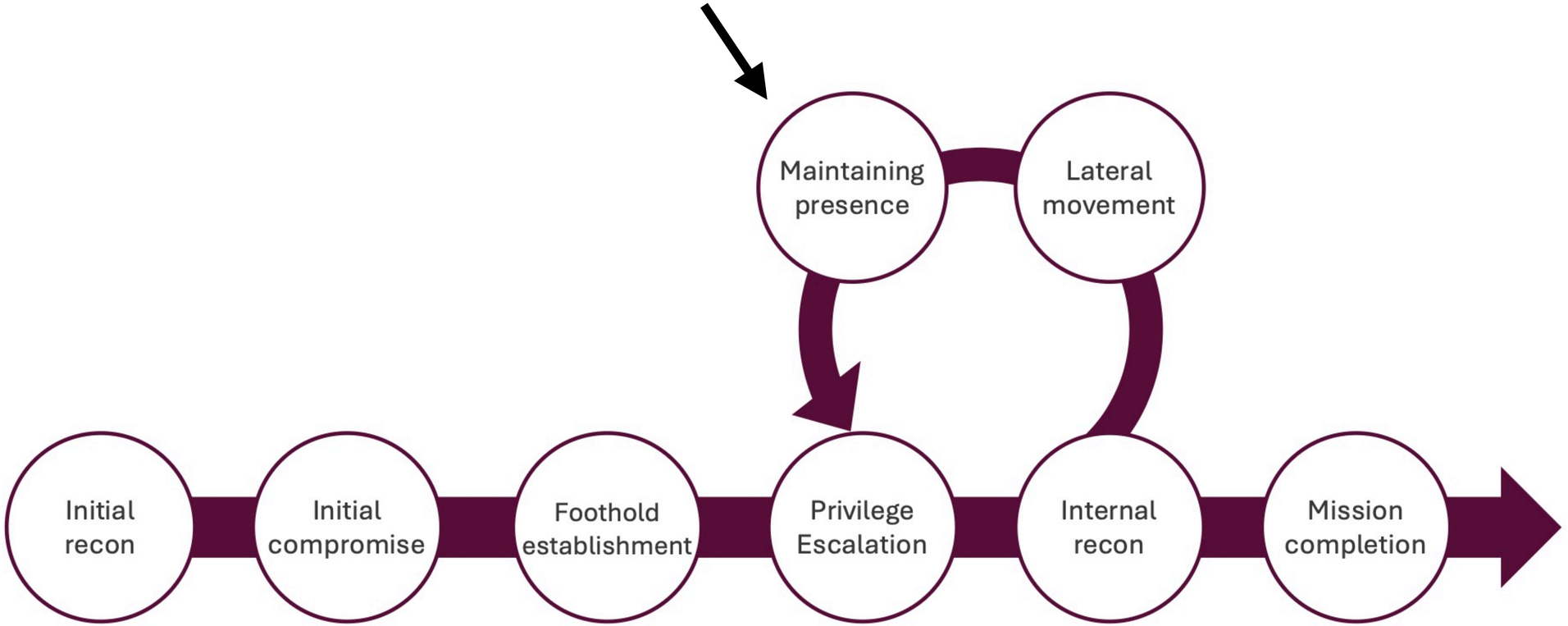
5. **Internal Reconnaissance**: Once on a host, APT1 scanned the internal network, identifying file servers, domain controllers, and other critical assets.

Case Study: APT1



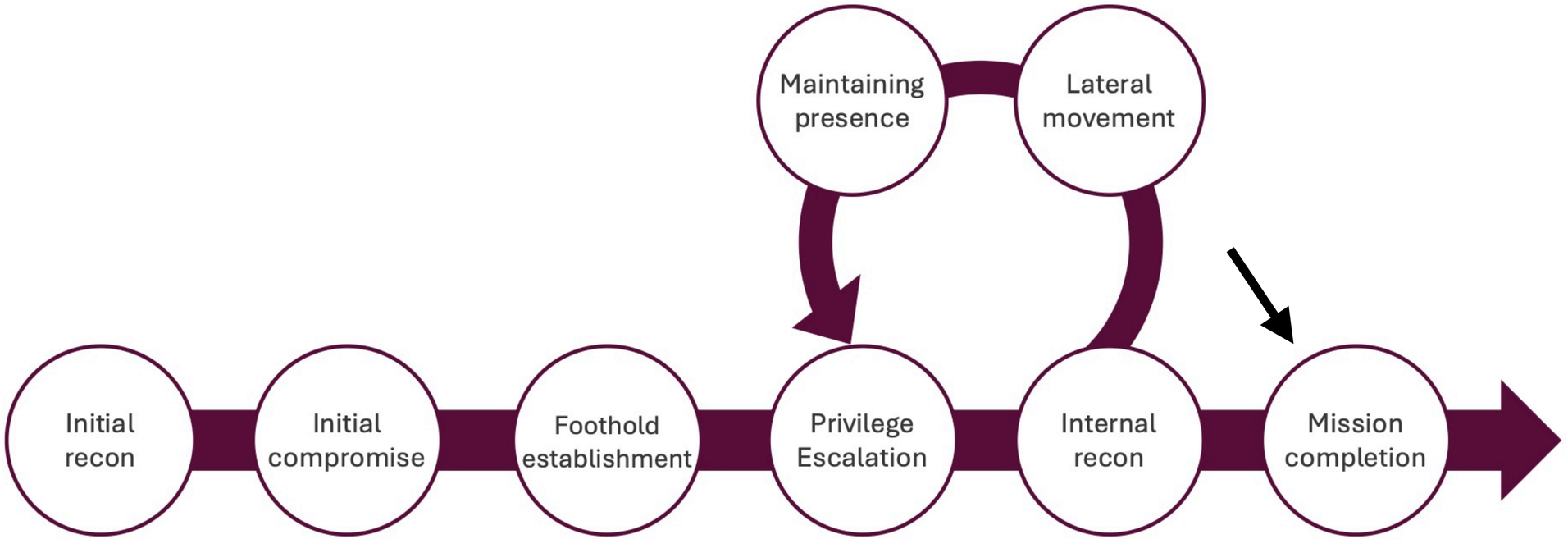
6. **Lateral Movement**: They leveraged stolen credentials and tools like PsExec and remote desktop protocol (RDP) to move laterally across the network.

Case Study: APT1



7. **Maintaining Presence**: Redundant backdoors and compromised user accounts ensured APT1 could remain within the network even if one access point was discovered.

Case Study: APT1

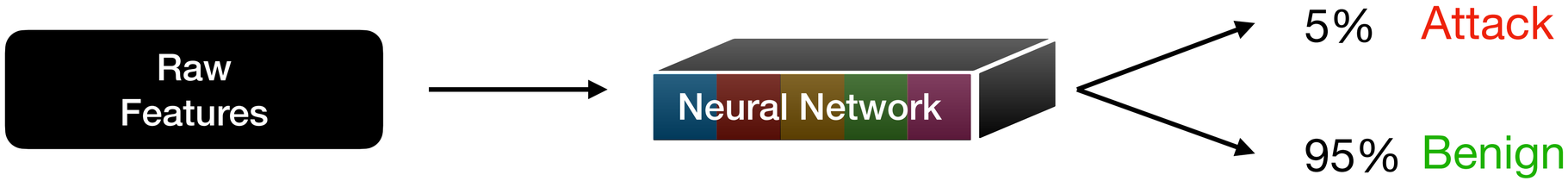


8. **Mission Completion**: Data exfiltration was executed such that files were compressed, encrypted, and sent back to C2 servers in China, often spanning months or even years of undetected presence.

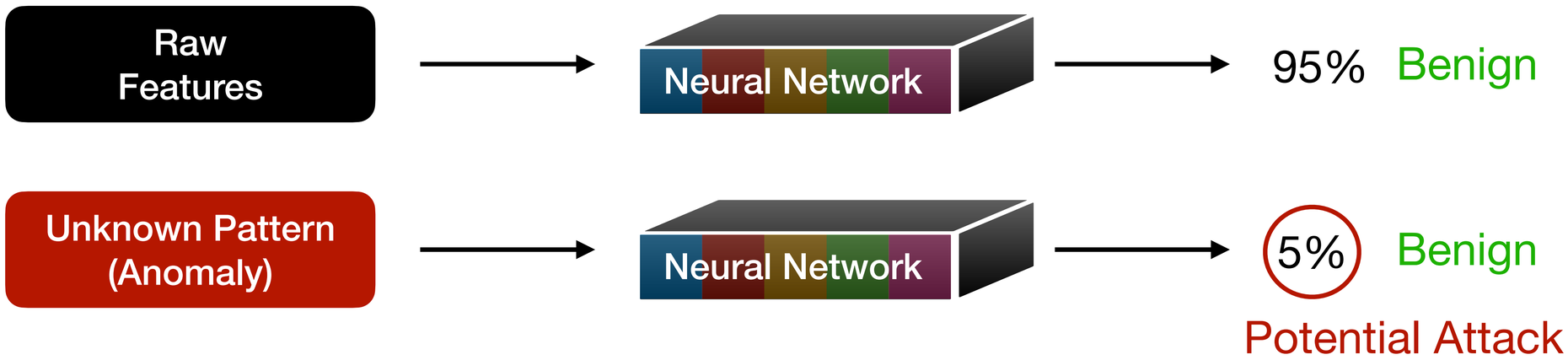
Detecting Unknown Attacks

Detecting Unknown Attack Patterns

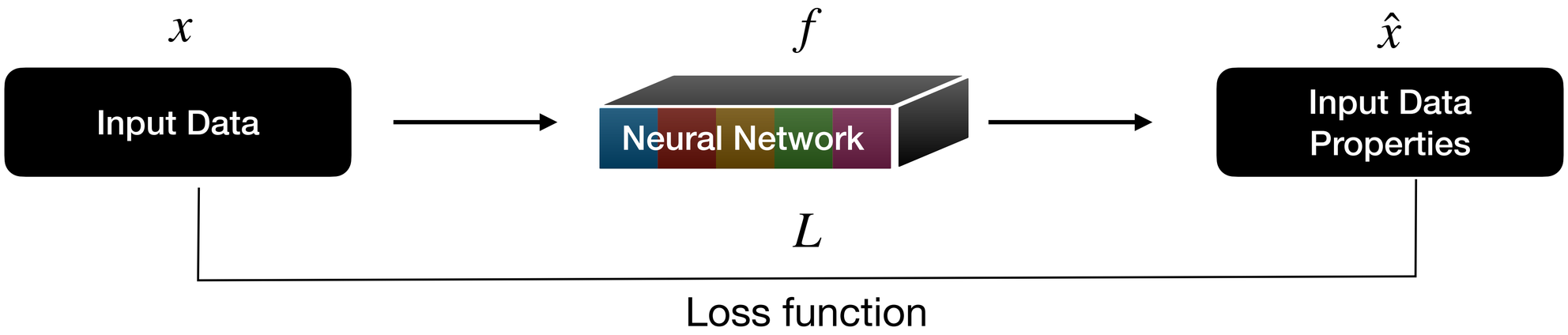
Learning from known attack data



Learning from benign data



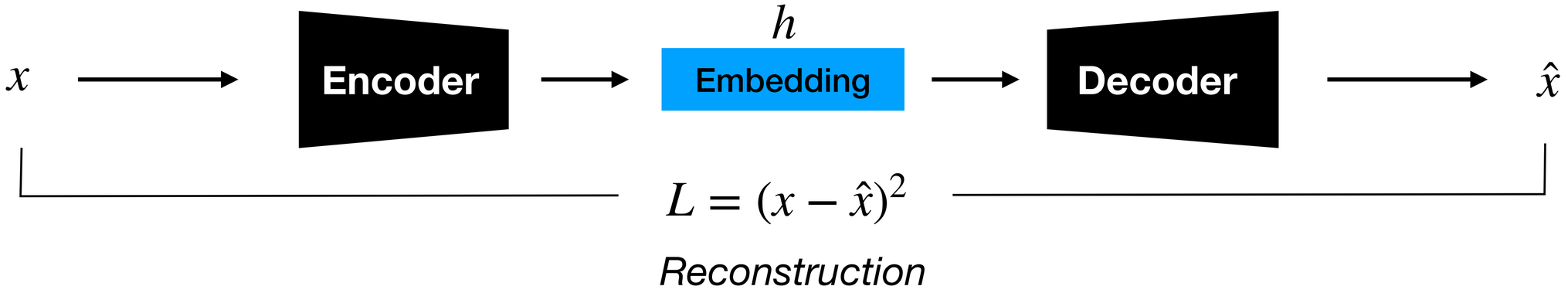
Self-Supervised Learning



*When trained on benign data, the loss serves as **anomaly score***

Self-Supervised Learning

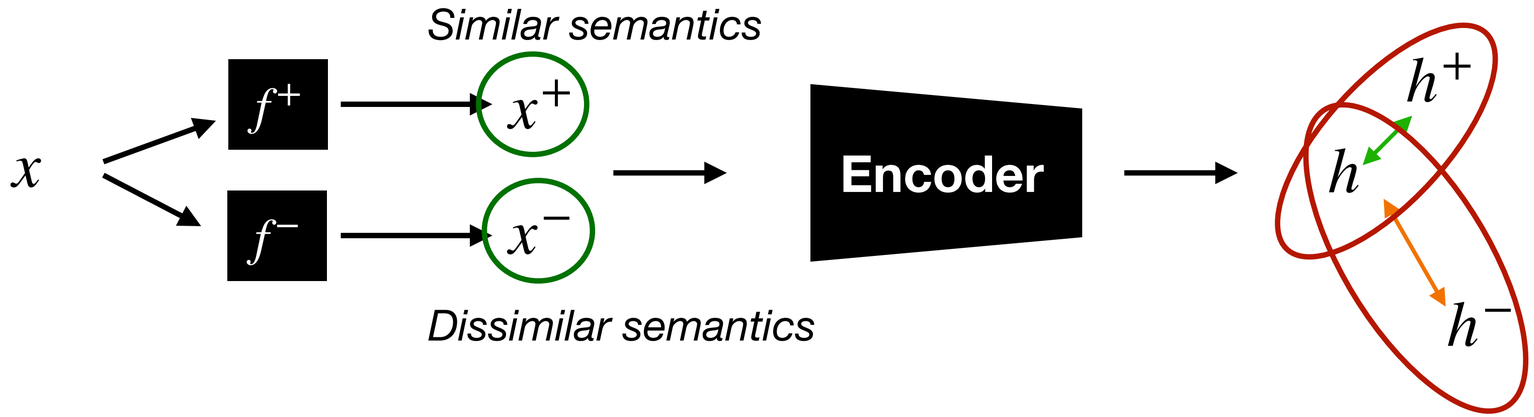
Autoencoder [7]



[7] Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." *science* 313.5786 (2006): 504-507.

Self-Supervised Learning

Contrastive Learning

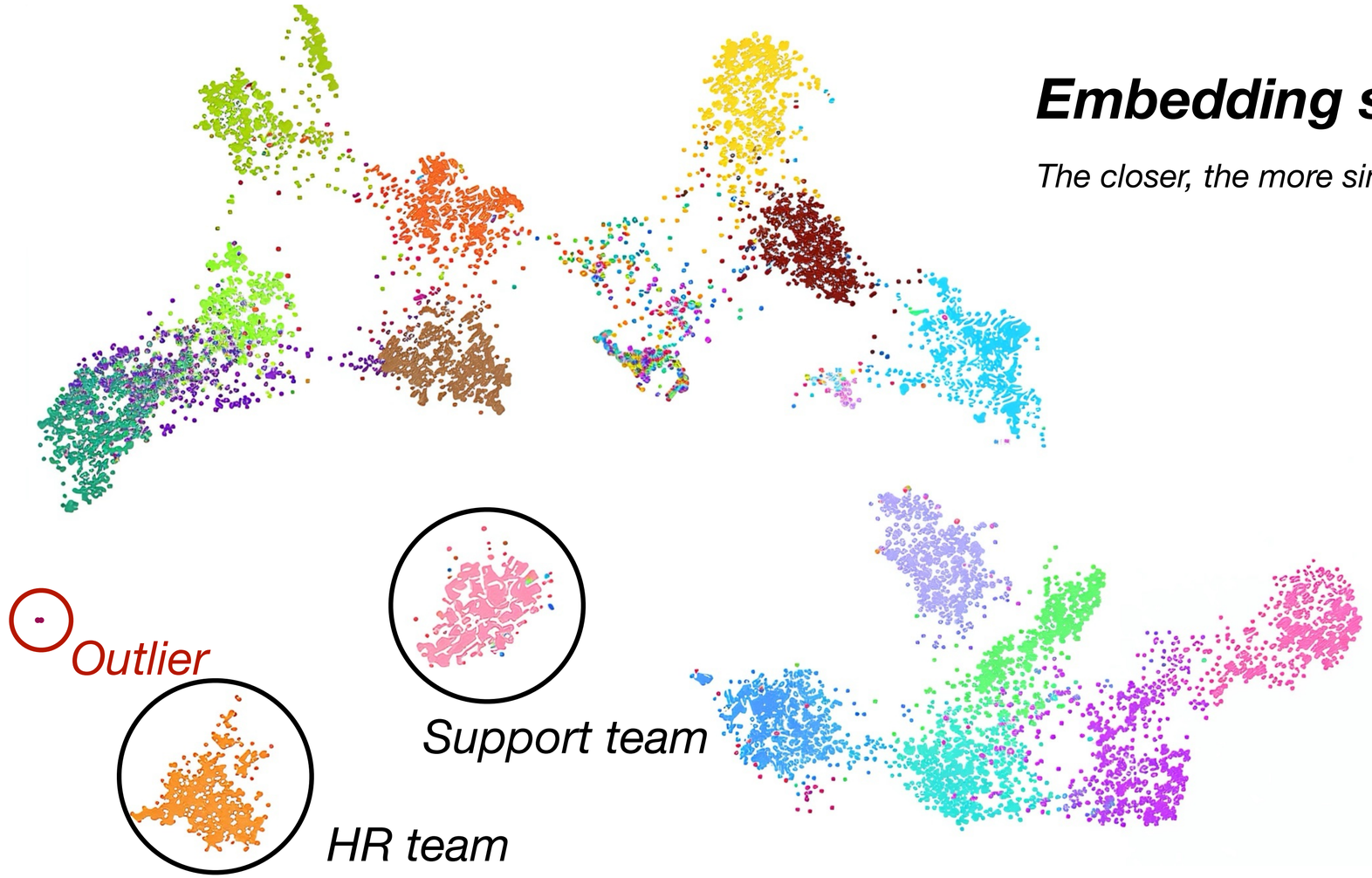


$$L_{\text{InfoNCE}} \overset{[8]}{=} \log \frac{\exp(\text{sim}(h, h^+)/\tau)}{\exp(\text{sim}(h, h^+)/\tau) + \sum_{k=1}^{N} \exp(\text{sim}(h, h_k^-)/\tau)}$$

$$\text{sim}(h_i, h_j) = \frac{h_i^\top h_j}{\|h_i\| \|h_j\|}$$

[8] Oord, Aaron van den, Yazhe Li, and Oriol Vinyals. "Representation learning with contrastive predictive coding." *arXiv preprint arXiv:1807.03748* (2018).

Self-Supervised Learning

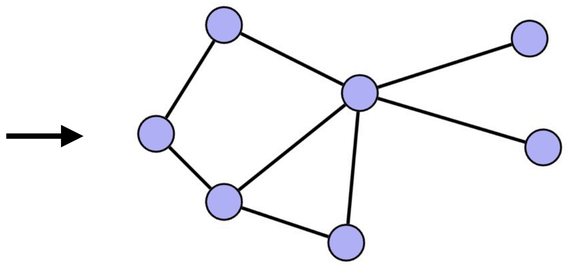


Embedding space

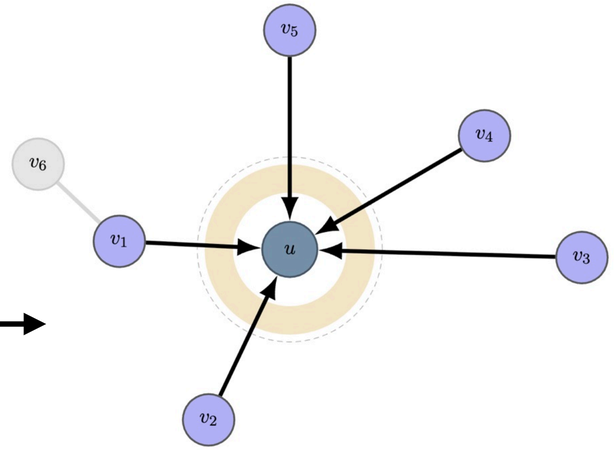
The closer, the more similar

Graph Learning

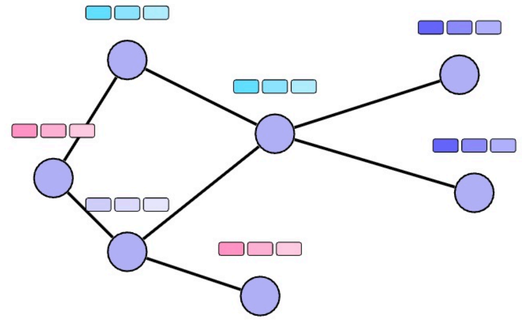

```
{
  "timestamp": "2025-11-05T21:03:00Z",
  "event_id": "evt_12345",
  "activity": "write",
  "entity": {
    "type": "file",
    "name": "/var/log/syslog",
    "inode": "12345"
  },
  "subject": {
    "type": "process",
    "id": "54321",
    "name": "logrotate",
    "user": "root"
  },
  "details": {
    "bytes_written": "1024",
    "signature": "..."
  },
  "wasAssociatedWith": "user_admin"
}
```

$G = (V, E)$



$h_u = f\big(h_u, \{h_v \mid v \in \mathcal{N}(u)\}$



$\{h_u, \quad \forall u \in V\}$

Flat Data

E.g., textual logs

Graph

E.g., attributed directed graph

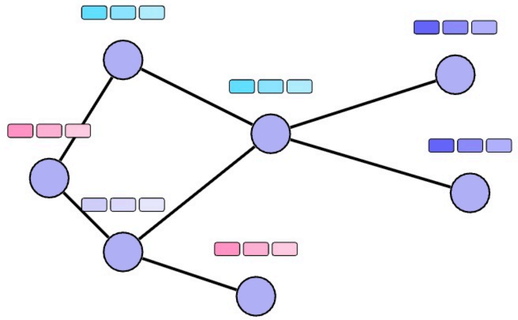
Graph Encoder

E.g., Graph Neural Network (GNN)

Node Embeddings

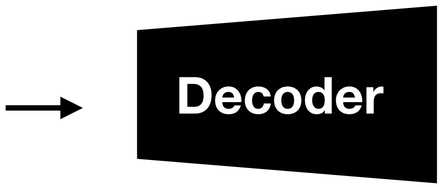
Or edge embeddings

Self-Supervised Graph Learning



$\{h_u, \quad \forall u \in V\}$

Node Embeddings



e.g., MLP

SSL objective

Reconstruction Loss

$$L_{\text{gen}} = \frac{1}{|V|} \sum_{u \in V} \left(\mathbf{x}_u - \hat{\mathbf{x}}_u \right)^2$$

Contrastive Loss

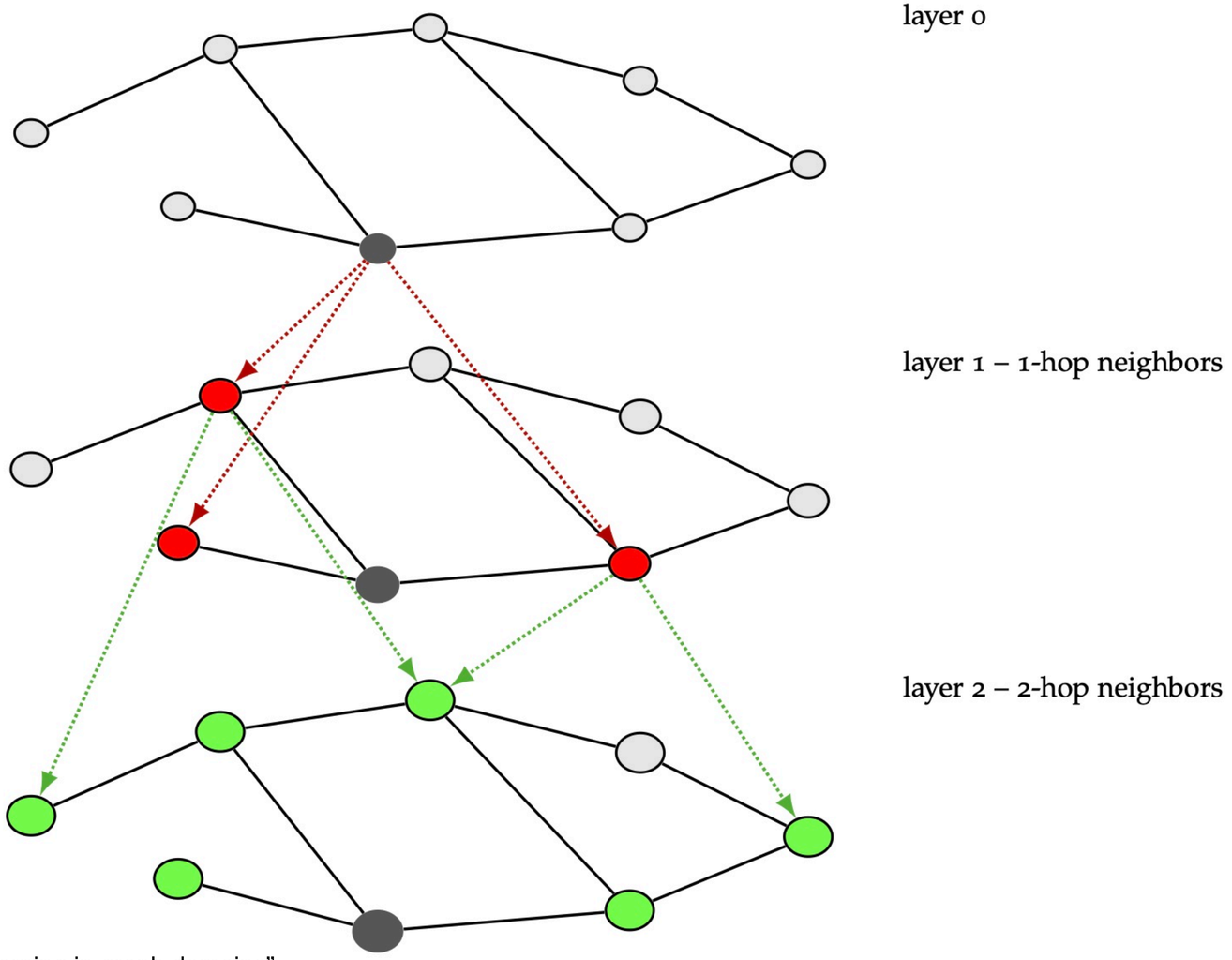
$$L_{\text{con}} = -\frac{1}{|V|} \sum_{u \in V} \log \left(\hat{y}_u \right) - \frac{1}{|\widetilde{V}|} \sum_{u \in \widetilde{V}} \log \left(1 - \hat{y}_u \right)$$

Predictive Loss

$$L_{\text{pre}} = -\frac{1}{|V|} \sum_{u \in V} \sum_{k=1}^{K} y_{u,k} \log(\hat{y}_{u,k})$$

Graph Neural Networks (GNNs)

- Exist since 2005 [42]
- Became famous in 2017 with the rise of deep learning
- Capture/learn relationships between entities
- Aggregate features from neighbors then multiply with learnable weights
- N GNN layers = N-hop neighbors



[42] Gori, Marco, Gabriele Monfardini, and Franco Scarselli (2005). “A new model for learning in graph domains”. In: Proceedings. 2005 IEEE international joint conference on neural networks 2.2005, pp. 729–734 (cit. on p. 17).

GNNs are Everywhere

DeepMind’s AlphaFold leverages GNNs for **protein folding prediction**

Pinterest uses GNNs for **content recommendation**

Alibaba uses GNNs for **product recommendation**

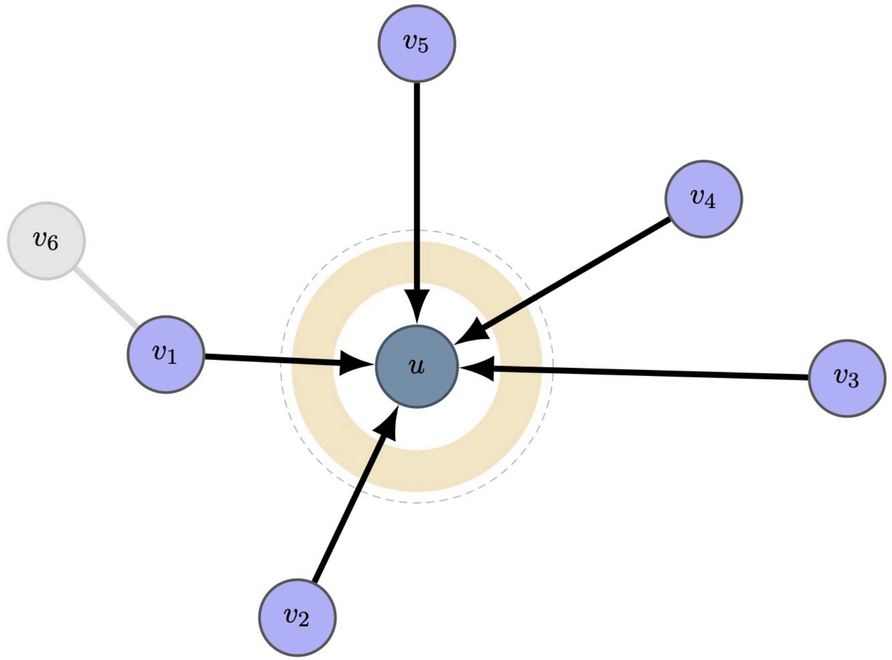
Uber uses GNNs for **matching drivers and clients**

Paypal uses GNNs for **fraud detection**

Graph Convolutional Network (GCN)

Each node receives a degree-normalized aggregation of its neighbors' features.

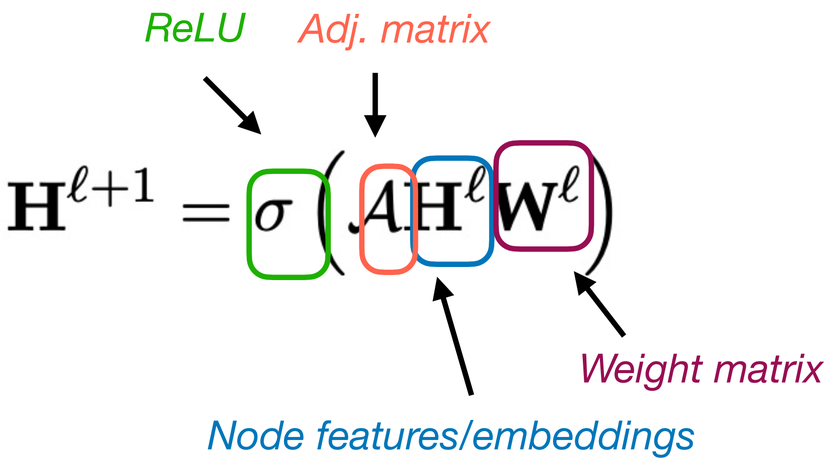
$$\mathbf{H}^{\ell+1} = \sigma\left(\mathcal{A}\mathbf{H}^{\ell}\mathbf{W}^{\ell}\right)$$

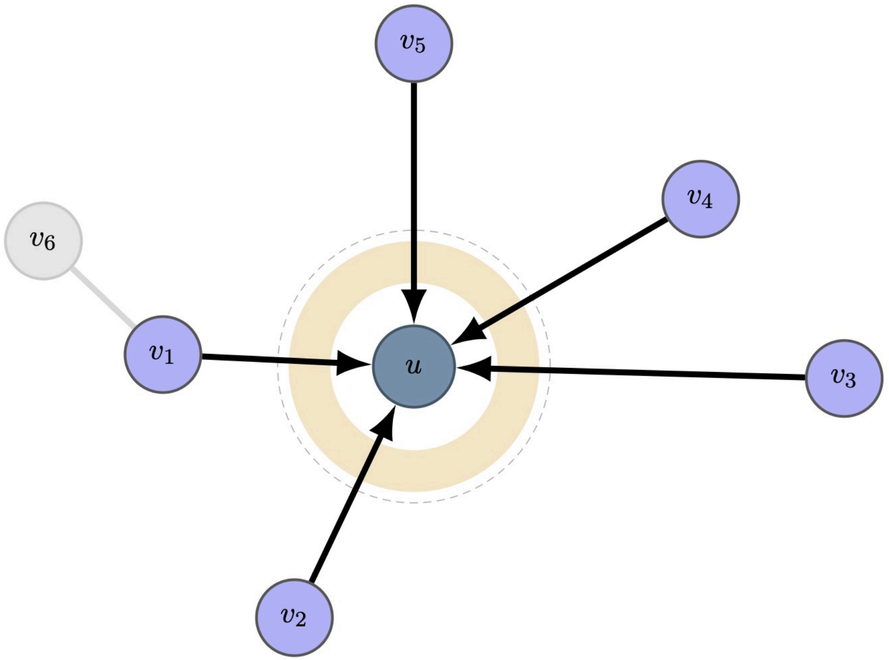


Kipf, T. N. "Semi-supervised classification with graph convolutional networks." *arXiv preprint arXiv:1609.02907* (2016).

Graph Convolutional Network (GCN)

Each node receives a degree-normalized aggregation of its neighbors' features.



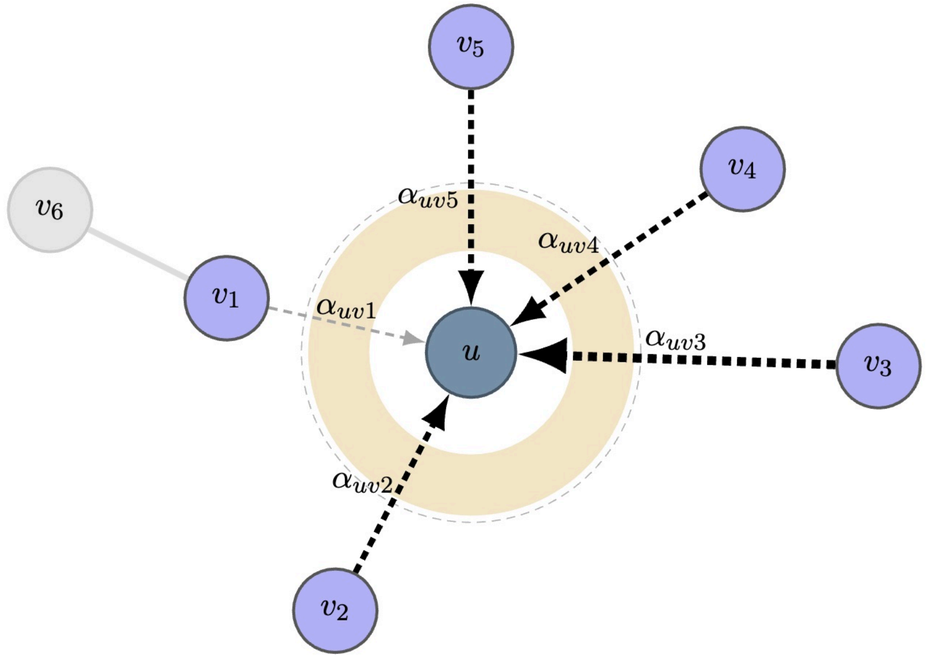


This operation is highly efficient on GPU

Graph Attention Network (GAT)

Each node receives an attention-weighted aggregation of its neighbors' features.

$$\mathbf{h}_u^{\ell+1} = \sigma\left(\sum_{v \in \mathcal{N}(u)} \alpha_{uv}^{\ell} \mathbf{W}^{\ell} \mathbf{h}_v^{\ell}\right)$$



Veličković, Petar, et al. "Graph attention networks." *arXiv preprint arXiv:1710.10903* (2017).

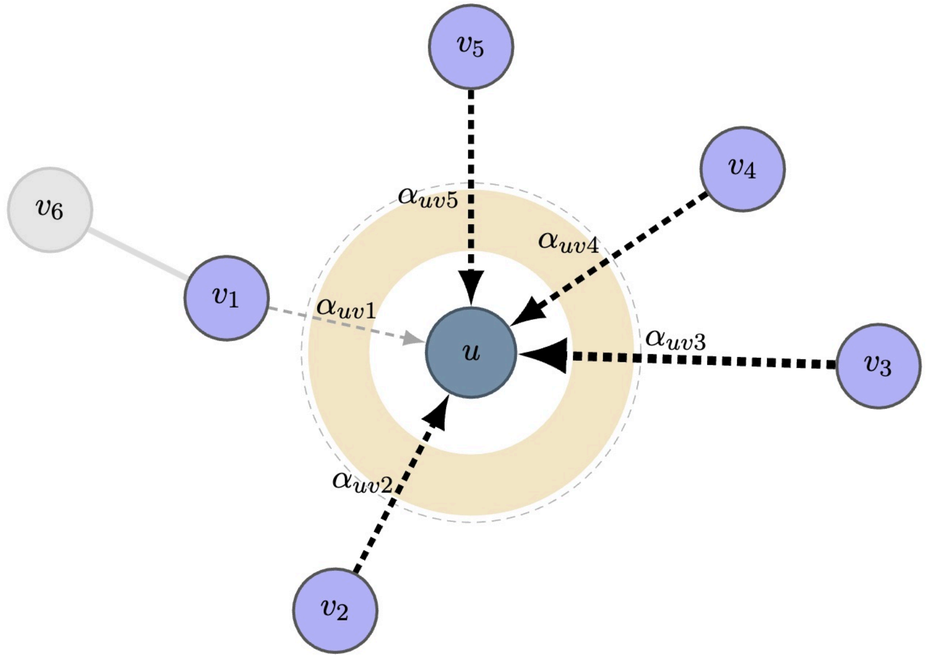
Graph Attention Network (GAT)

Each node receives an attention-weighted aggregation of its neighbors' features.

1. Aggregation weighted by attention coefficient

Attention coefficient

$$\mathbf{h}_u^{\ell+1} = \sigma\left(\sum_{v \in \mathcal{N}(u)} \alpha_{uv}^{\ell} \mathbf{W}^{\ell} \mathbf{h}_v^{\ell}\right)$$



Graph Attention Network (GAT)

Each node receives an attention-weighted aggregation of its neighbors' features.

2. Edge-level attention coefficients

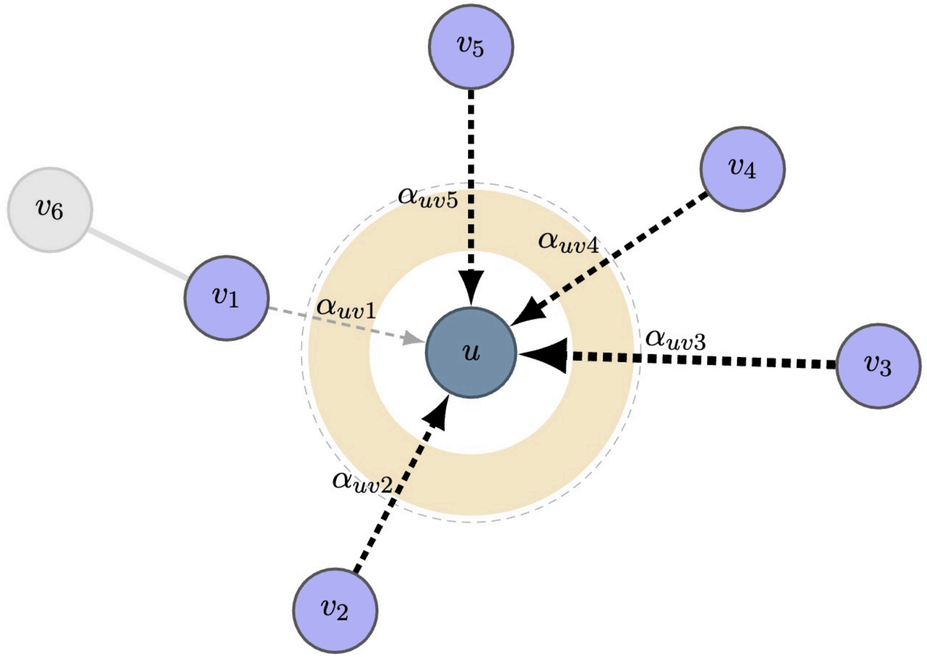
Src-dst node concatenation

Attention vector

LeakyReLU

$$\alpha_{uv} = \frac{\exp\left(\sigma\left(\mathbf{a}^T\left[\mathbf{W}\mathbf{h}_u, \mathbf{W}\mathbf{h}_v\right]\right)\right)}{\sum_{k\in\mathcal{N}(u)}\exp\left(\sigma\left(\mathbf{a}^T\left[\mathbf{W}\mathbf{h}_u, \mathbf{W}\mathbf{h}_k\right]\right)\right)},$$

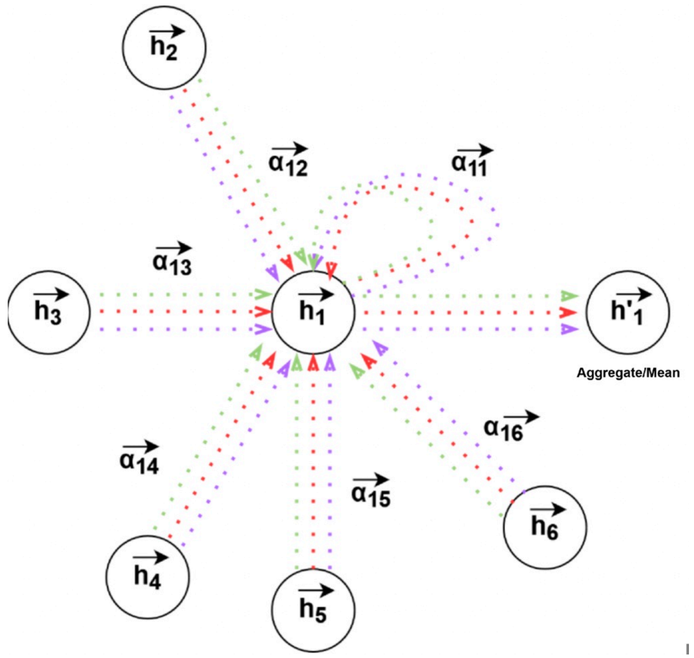
Sum of all u's neighbors



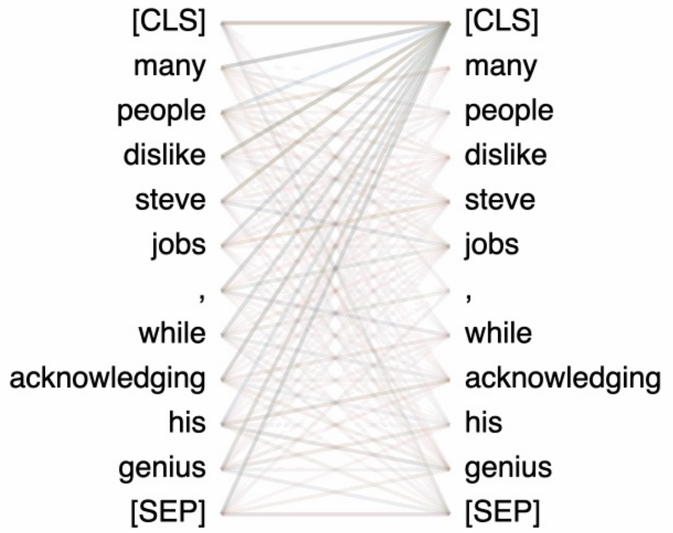
GAT vs Transformer

A **transformer** can be seen as a GAT on a **fully connected** graph, given a sentence modelled as a graph where nodes are tokens.

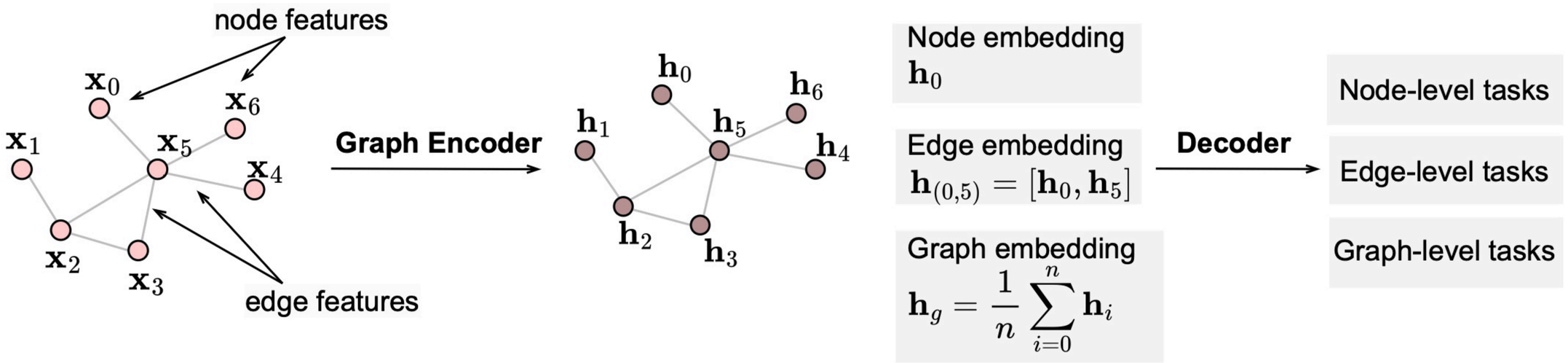
GAT
Each node attends its direct neighbors



Transformer
Each token attends to all other tokens
=> fully-connected graph



Encoder-Decoder Architecture



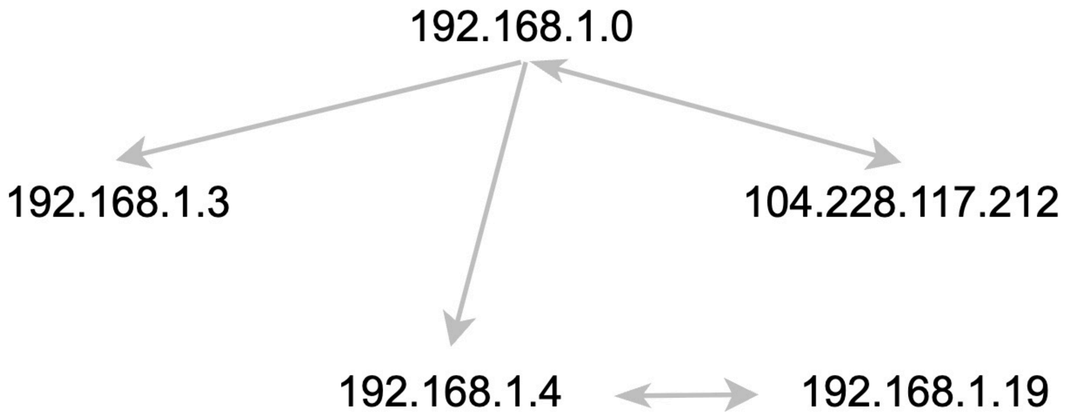
This type of architecture enables attack detection at the node, edge or graph levels

Representing Systems as Graphs

Common Graphs: Network Graph

- **Nodes**: IP addresses
- **Edges**: Network flows

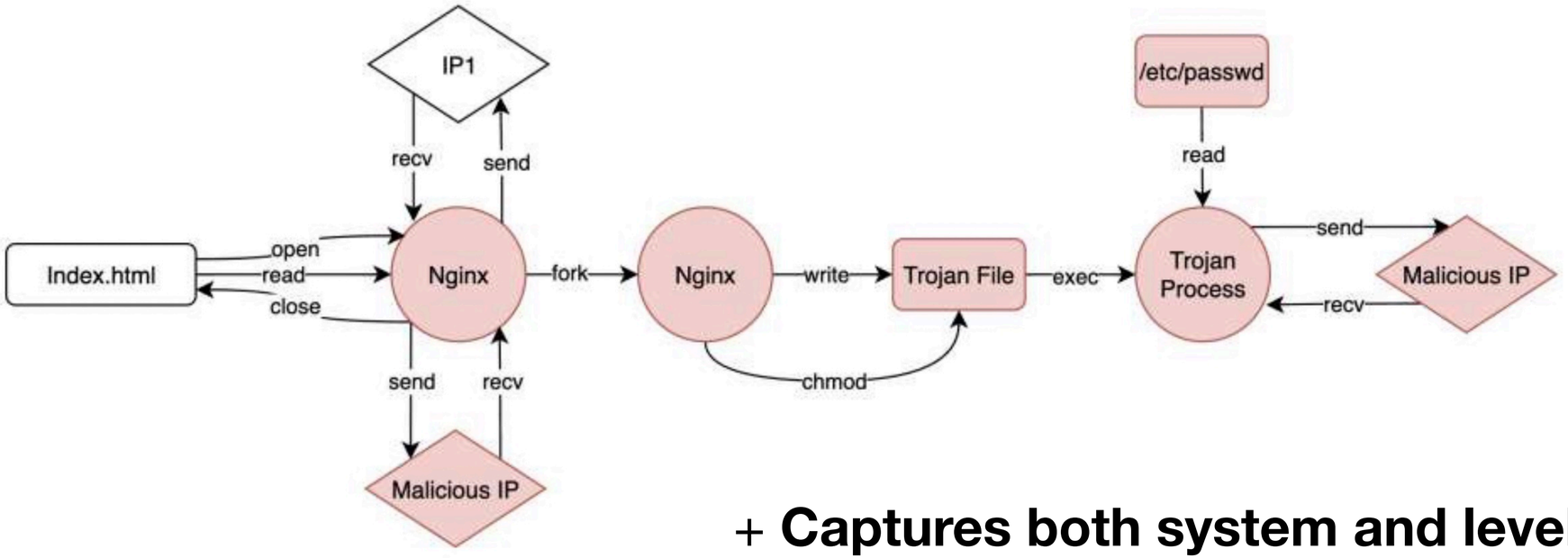
- **Node Features**: One-hot vector
- **Edge Features**: Network stats



Common Graphs: Provenance Graph

- **Nodes**: Operating system entities (e.g., file, process, socket)
- **Edges**: Events, (e.g., system calls)

- **Node Features**: Path, cmd line, IP address
- **Edge Features**: System call type, timestamp

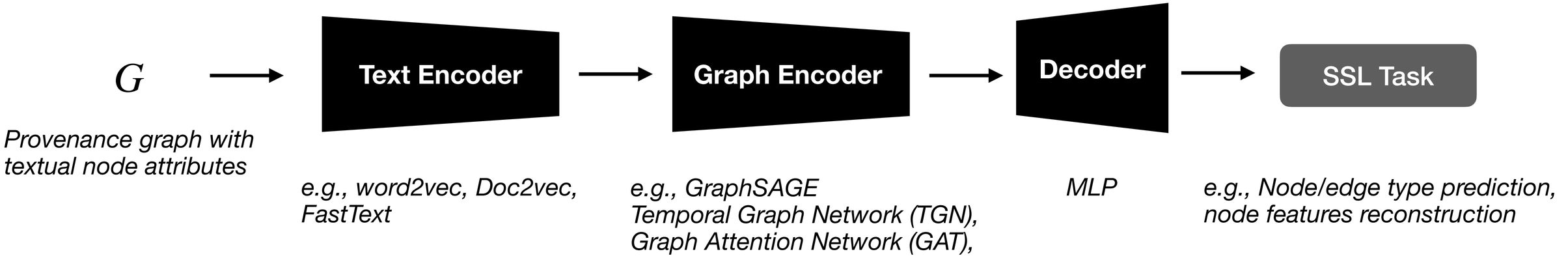


+ **Captures both system and level activity!**

Provenance-based Intrusion Detection Systems (PIDSs)

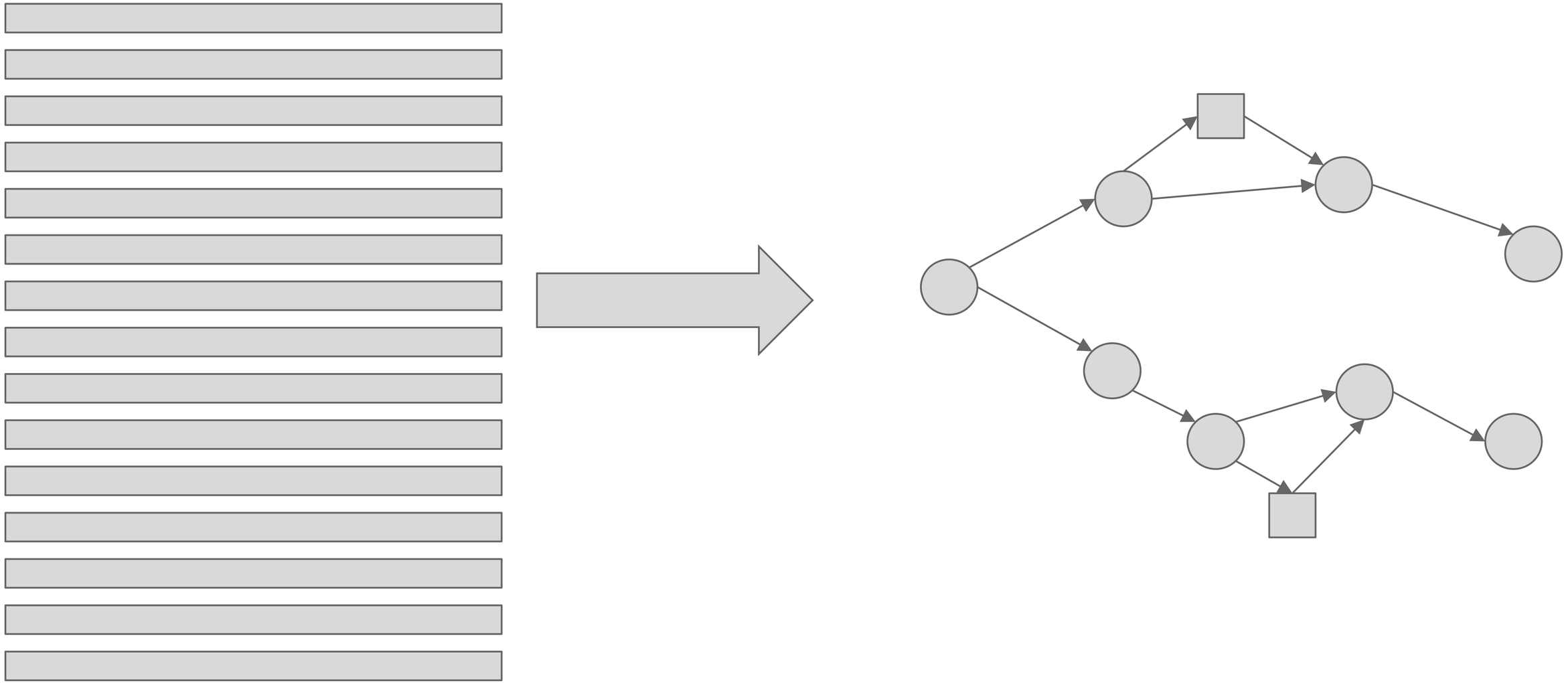
Recent PIDSs are:
Kairos [18], **ThreaTrace** [19], **NodLink** [20], **Magic** [21], **Flash** [22], **R-Caid** [23], **SIGL** [24]

They share a **common architecture**:

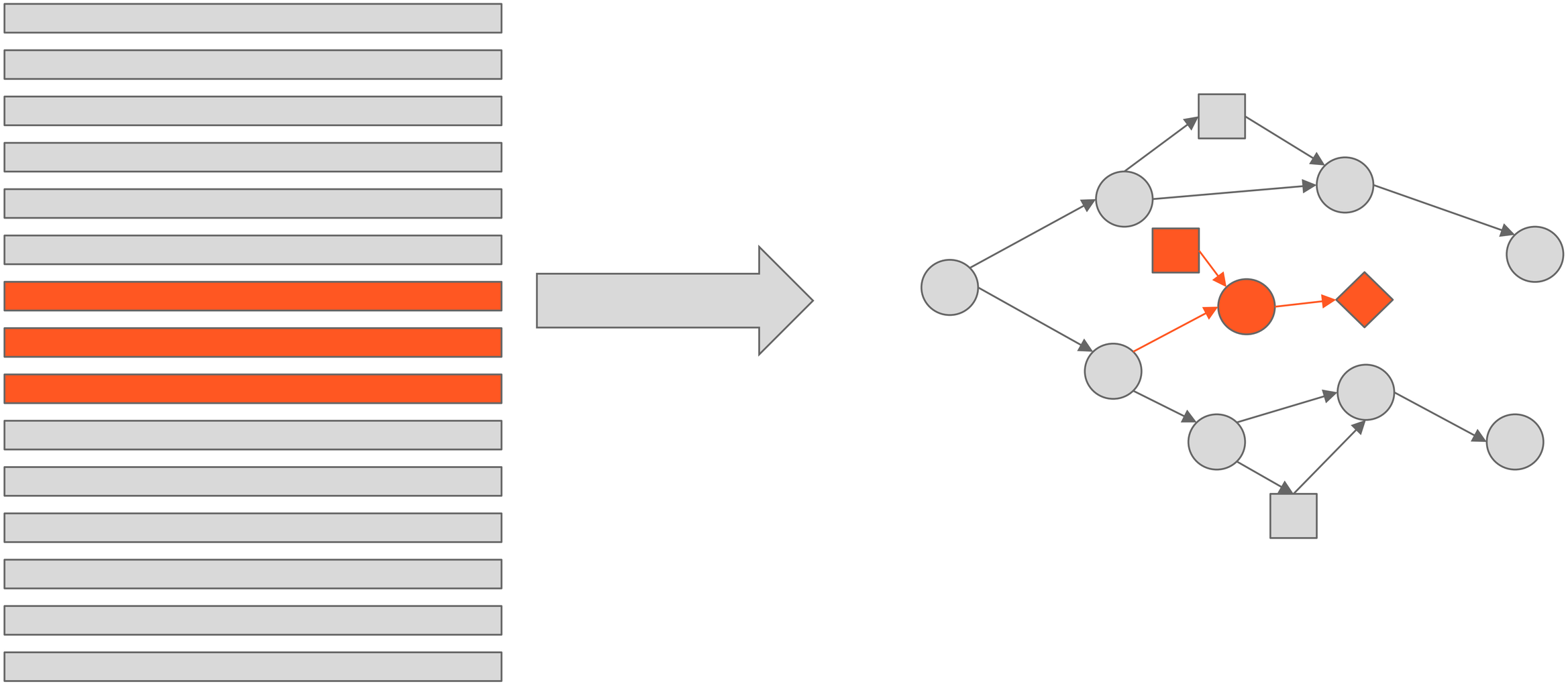


[18] Cheng, Zijun, et al. "Kairos: Practical intrusion detection and investigation using whole-system provenance." *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024.
[19] Wang, Su, et al. "Threatrace: Detecting and tracing host-based threats in node level through provenance graph learning." *IEEE Transactions on Information Forensics and Security* 17 (2022): 3972-3987.
[20] Li, Shaofei, et al. "Nodlink: An online system for fine-grained apt attack detection and investigation." *arXiv preprint arXiv:2311.02331* (2023).
[21] Jia, Zian, et al. "{MAGIC}: Detecting advanced persistent threats via masked graph representation learning." *33rd USENIX Security Symposium (USENIX Security 24)*. 2024.
[22] Rehman, Mati Ur et al. "Flash: A comprehensive approach to intrusion detection via provenance graph representation learning." *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024.
[23] Goyal, Akul, Gang Wang, and Adam Bates. "R-caid: Embedding root cause analysis within provenance-based intrusion detection." *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024.
[24] Han, Xueyuan, et al. "{SIGL}: Securing software installations through deep graph learning." *30th USENIX Security Symposium (USENIX Security 21)*. 2021.

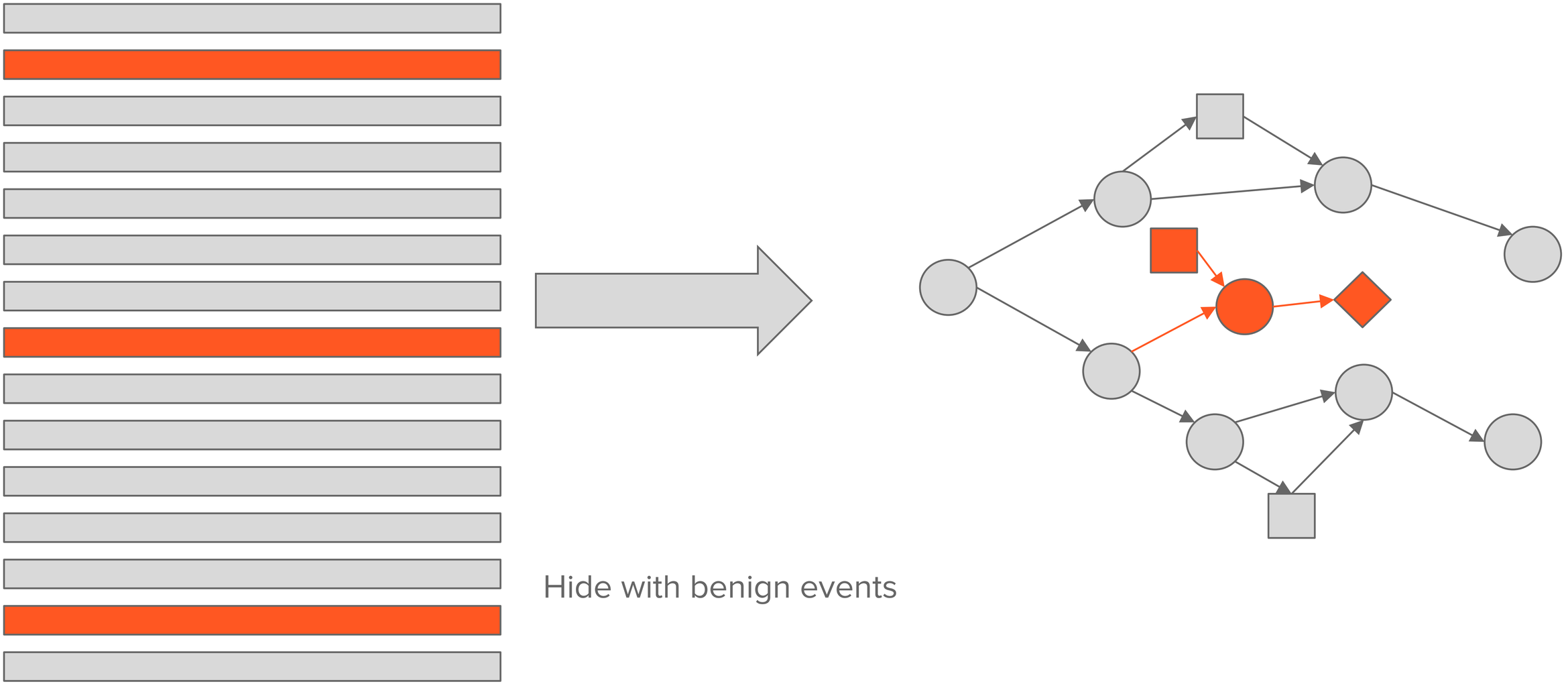
PIDSs are Suited for APT Detection



PIDSs are Suited for APT Detection

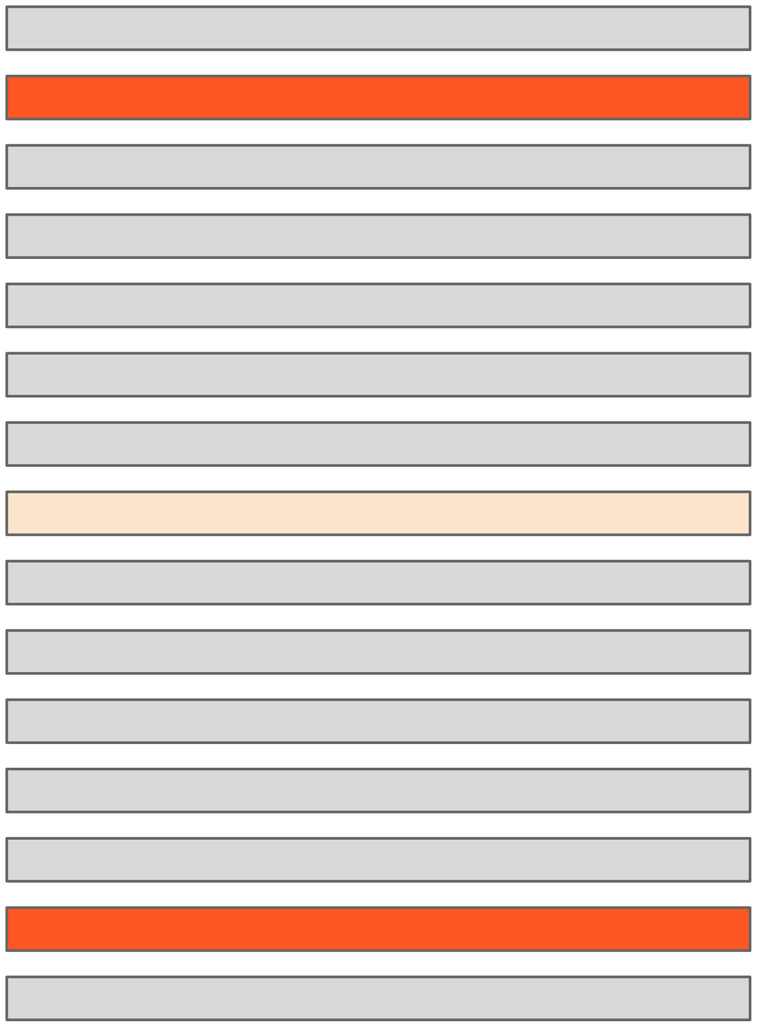


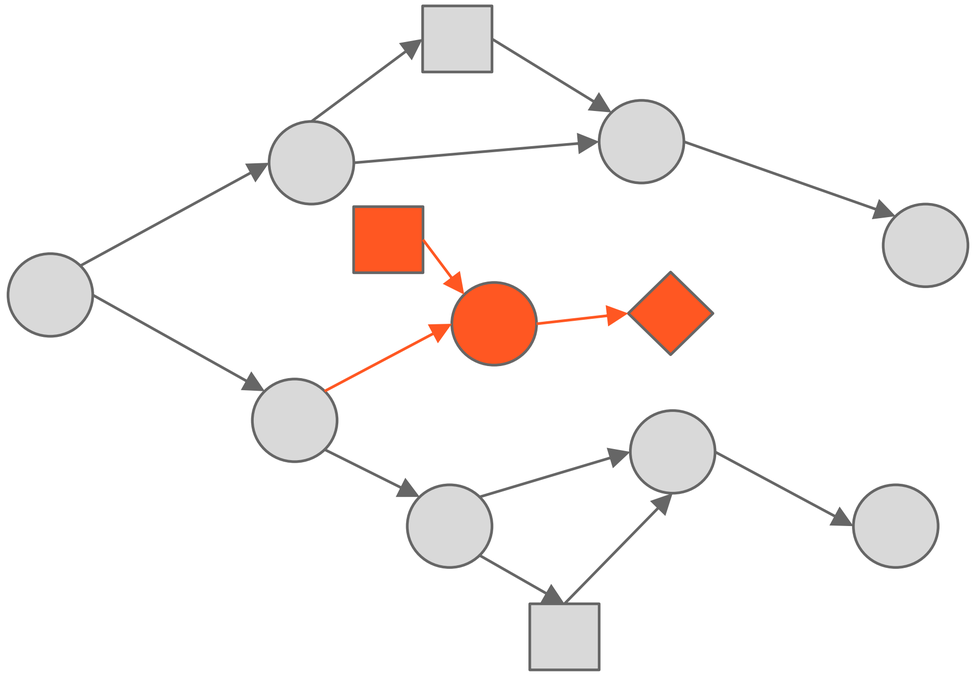
PIDSs are Suited for APT Detection



Hide with benign events

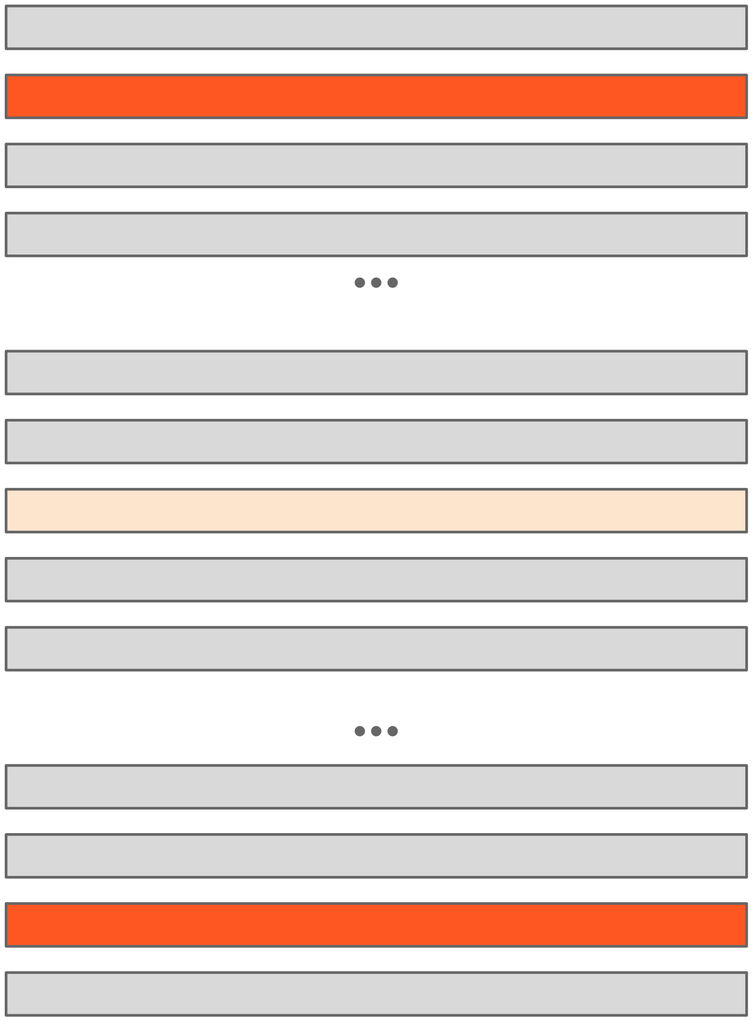
PIDSs are Suited for APT Detection

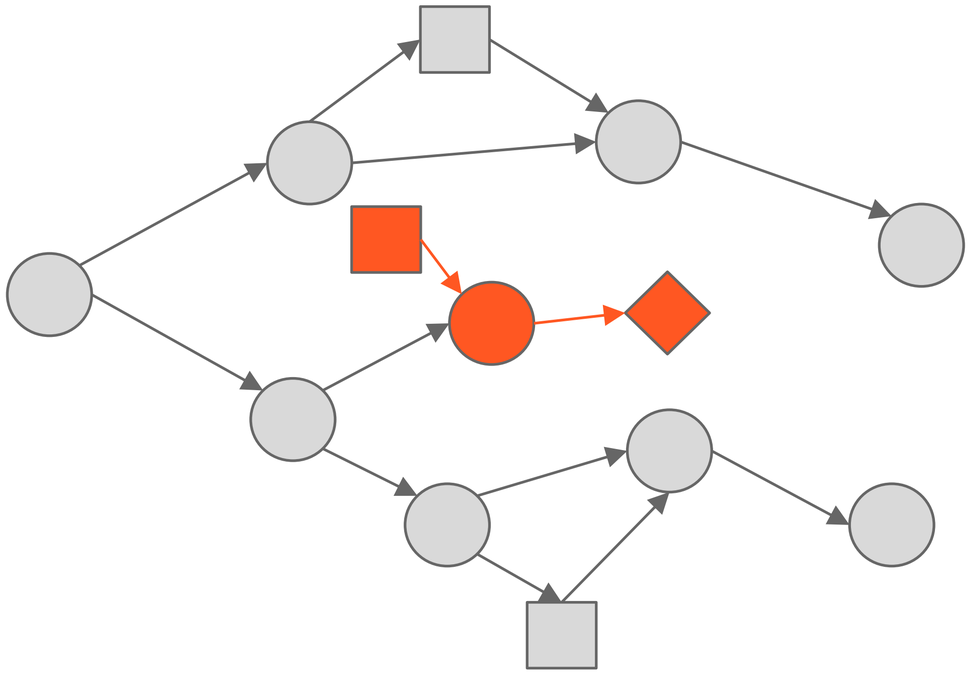




Masquerading as a benign event

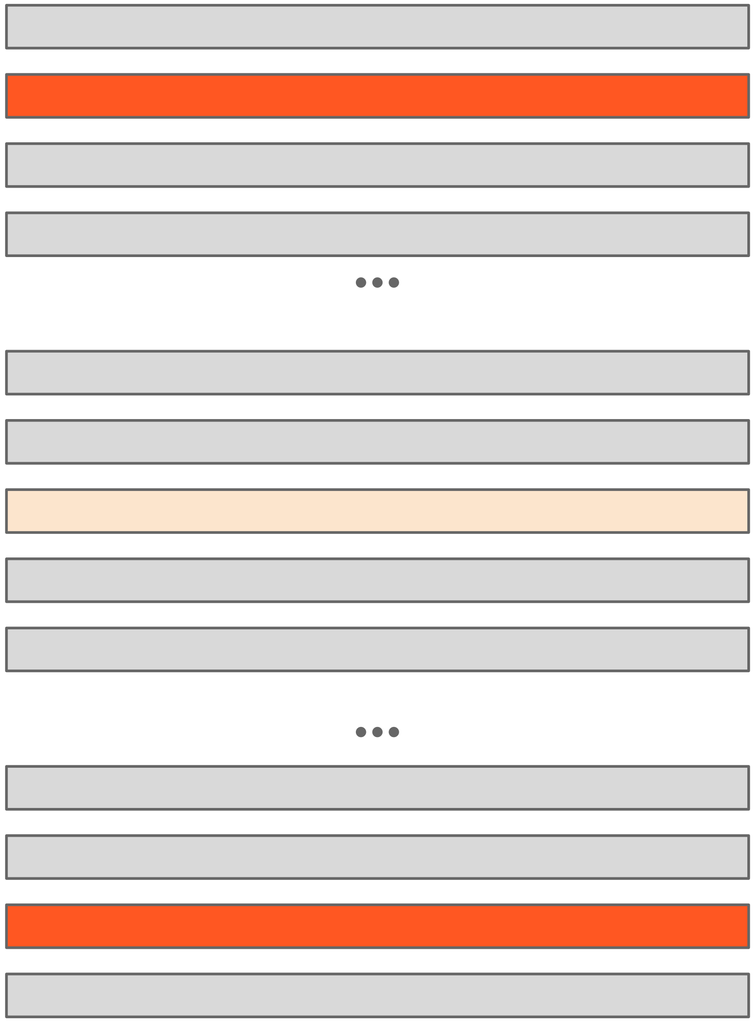
PIDSs are Suited for APT Detection

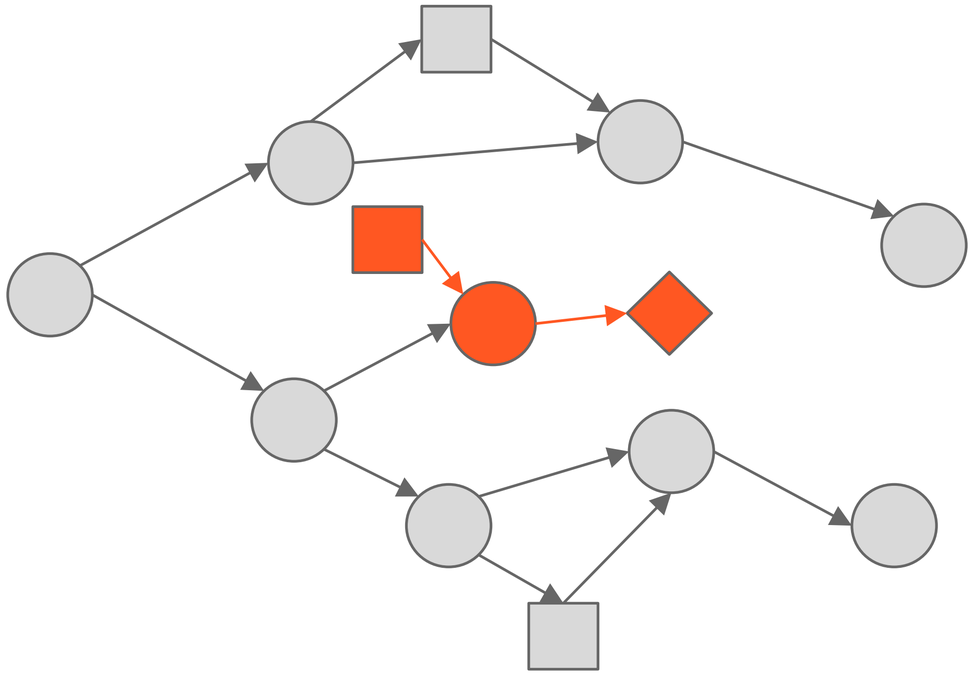




Attacks spaced in time

PIDSs are Suited for APT Detection





Causality relationship is preserved.

Current Research Objectives

Since the last decade, researchers attempt to make APT detection with self-supervised graph learning **practical** and **deployable** in **SOCs** because:

- Results with self-supervised graph learning are **very encouraging**

But,

- These methods are **more uncertain** and involve **more overhead**
- **Recent research** attempts to make these methods **practical**

Limitations of Current PIDSs

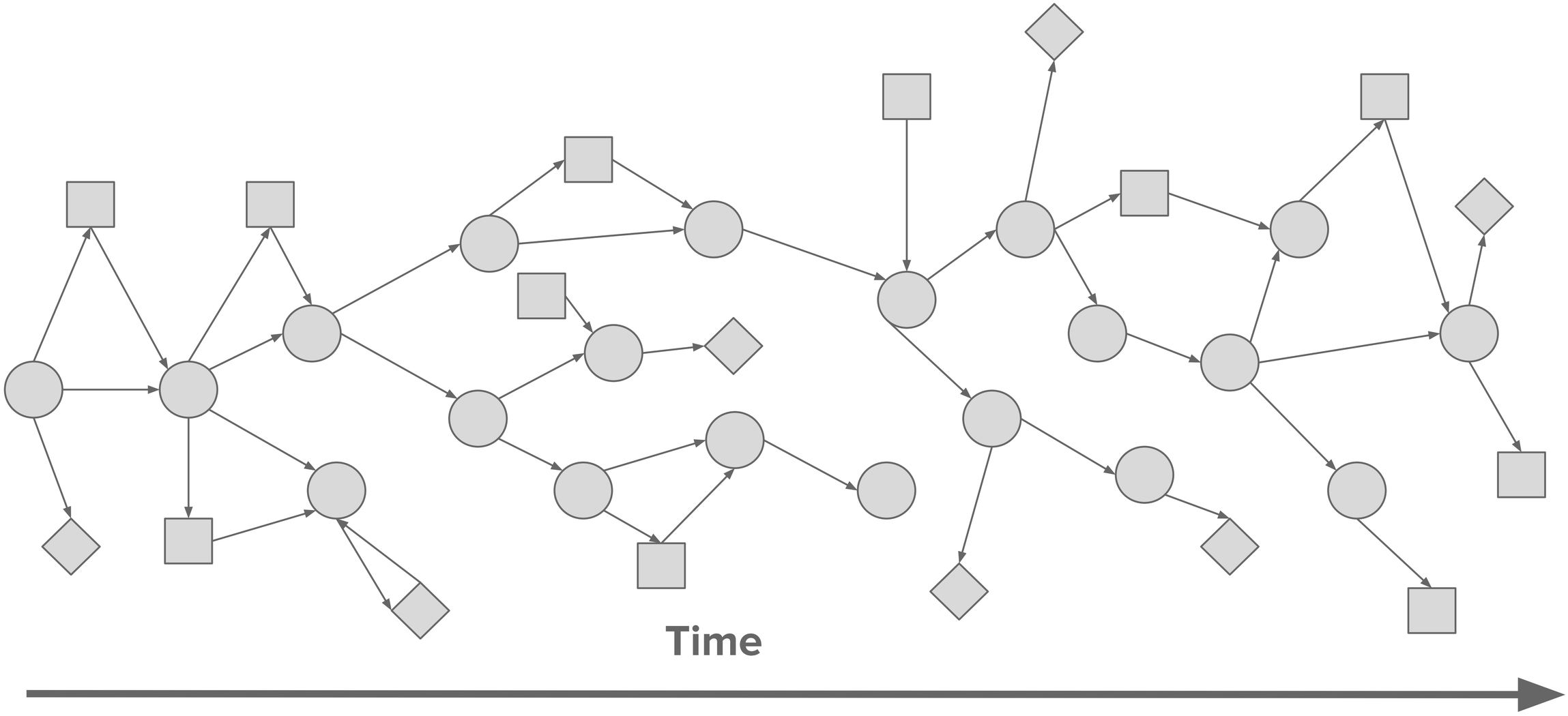
Coarse-grained Detection Granularity

➡ Most systems report near-perfect detection, but **using tricks to reduce false positives**

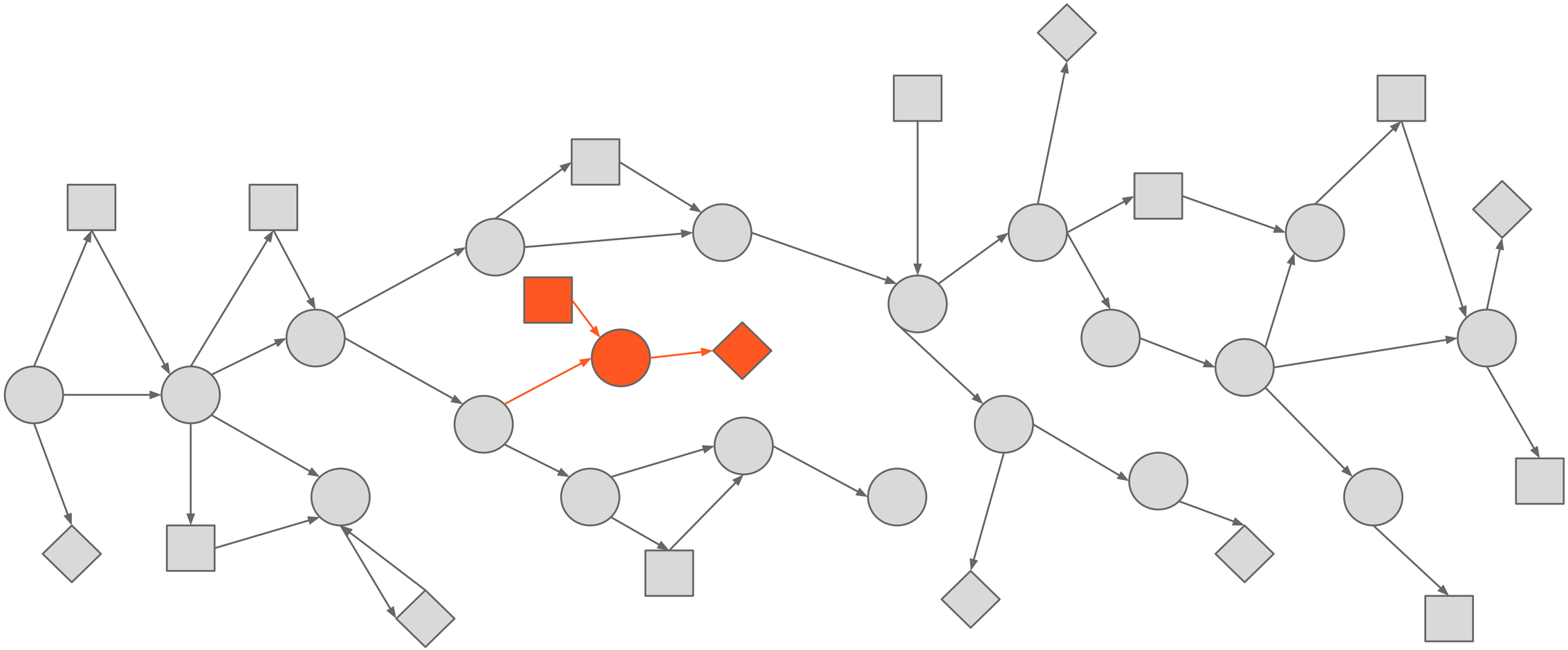
Dataset	Total Nodes	Neighborhood	Batch	Source
E5-CADETS	7,632,792	20,524	717,783	401,065
E5-THEIA	1,728,121	162,714	61,368	9,374
E5-CLEARSCOPE	326,338	48,488	8,636	1,020

Thousands of nodes reported to security analysts => Not practical

Assume a Provenance Graph

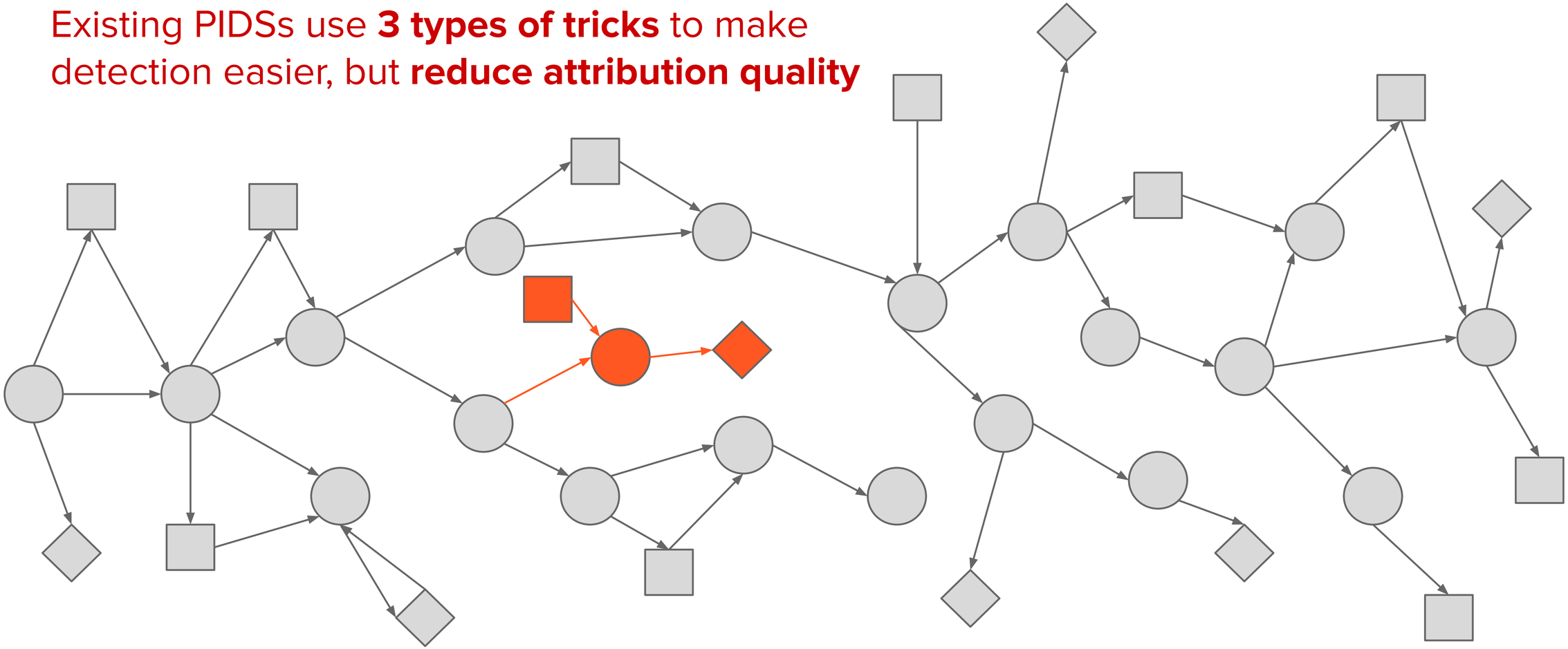


Assume an Attack



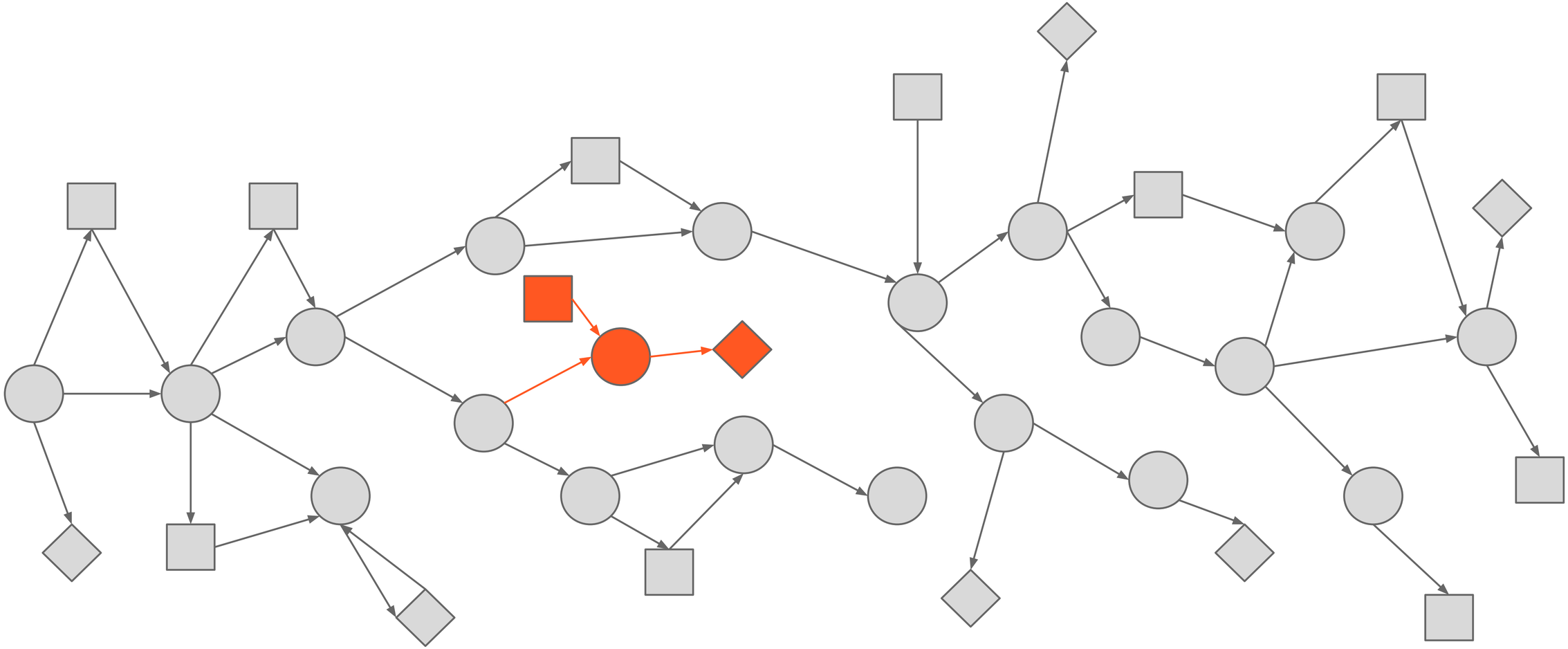
Assume an Attack

Existing PIDSs use **3 types of tricks** to make detection easier, but **reduce attribution quality**



Neighborhood Attribution

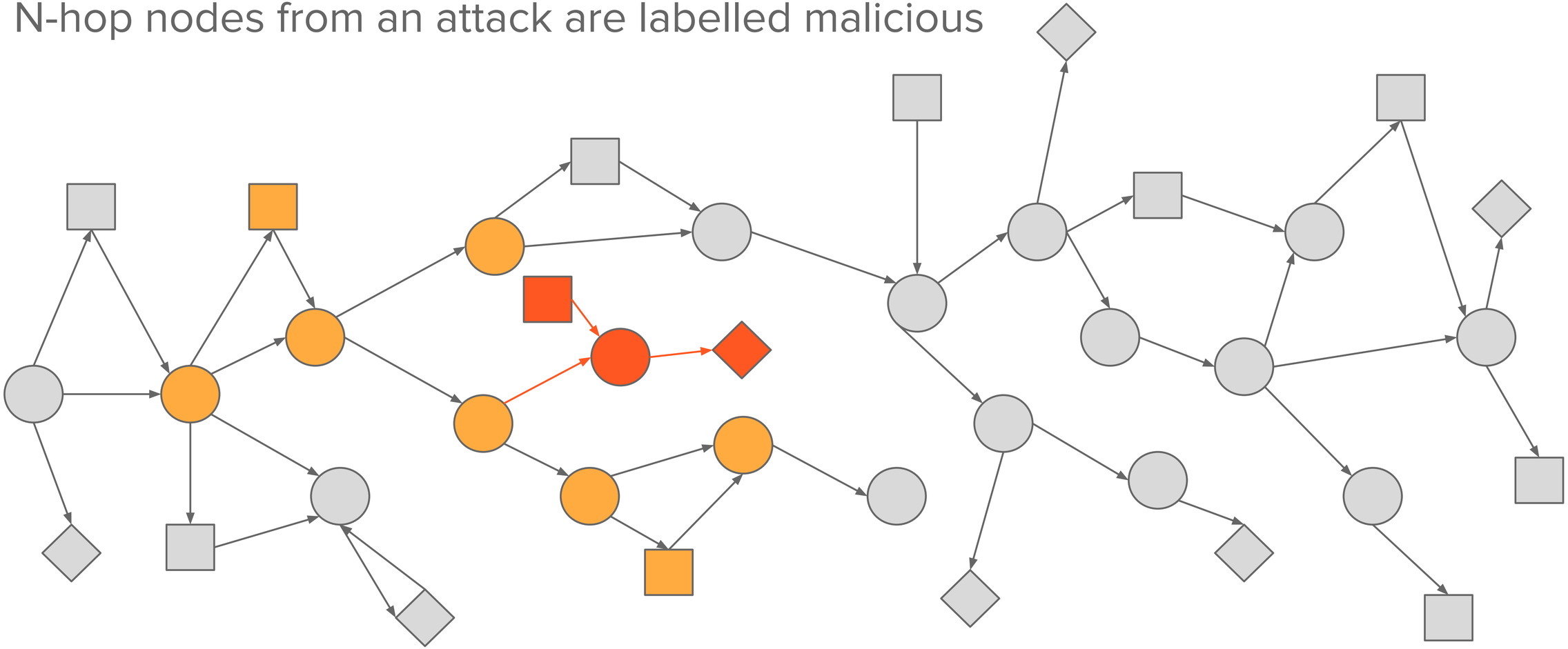
Concerned PIDSs:
Flash, S&P 2024
MAGIC, USENIX Sec 2024
ThreaTrace, TIFS 2022



Neighborhood Attribution

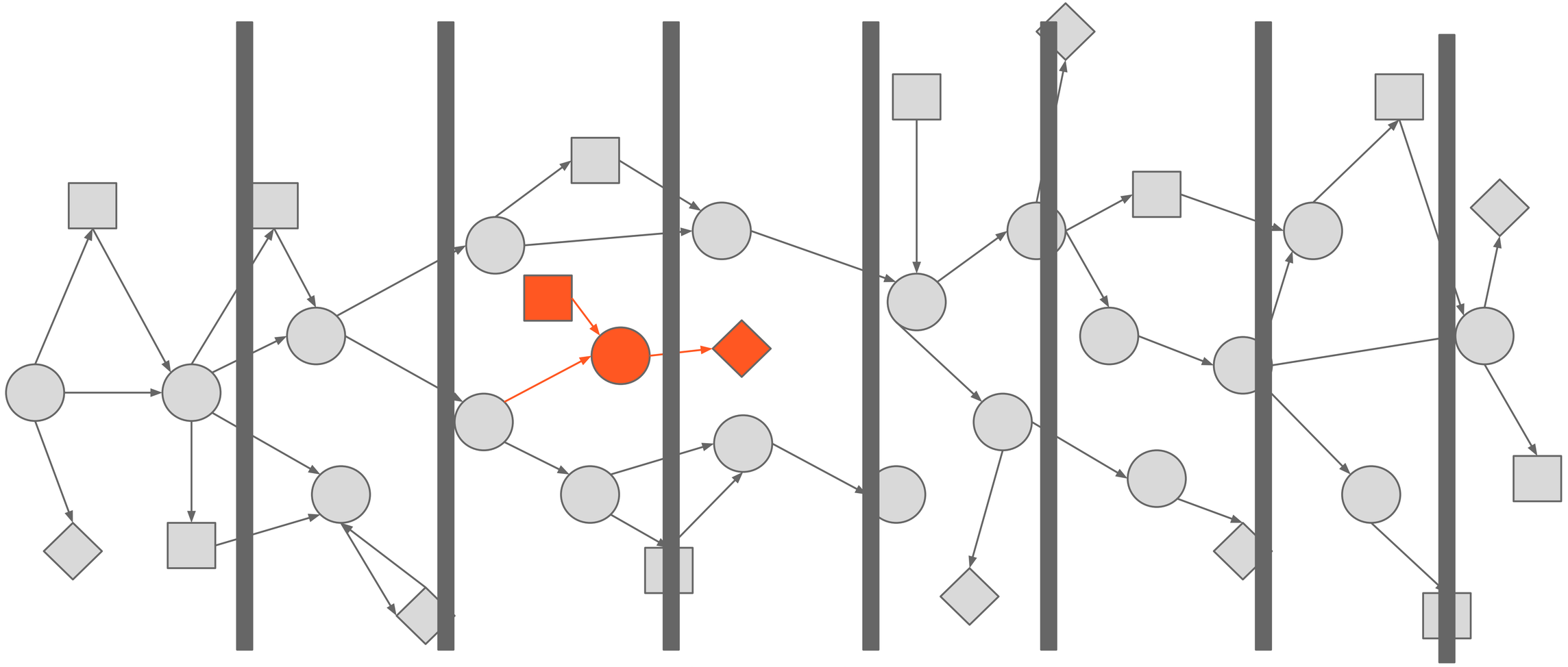
Concerned PIDSs:
Flash, S&P 2024
MAGIC, USENIX Sec 2024
ThreaTrace, TIFS 2022

N-hop nodes from an attack are labelled malicious



Batch Attribution

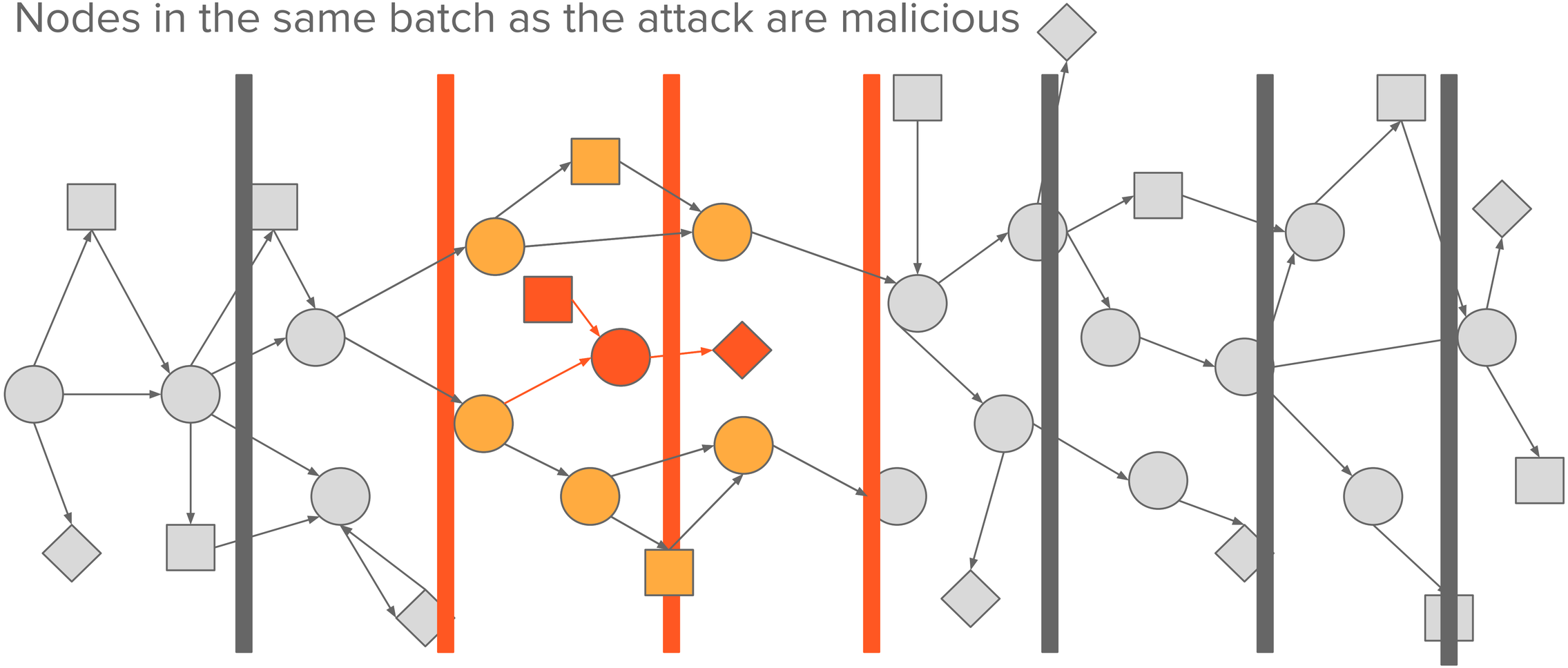
Concerned PIDSs:
Kairos, S&P 2024
EdgeTorrent, RAID's 2023



Batch Attribution

Concerned PIDSs:
Kairos, S&P 2024
EdgeTorrent, RAID's 2023

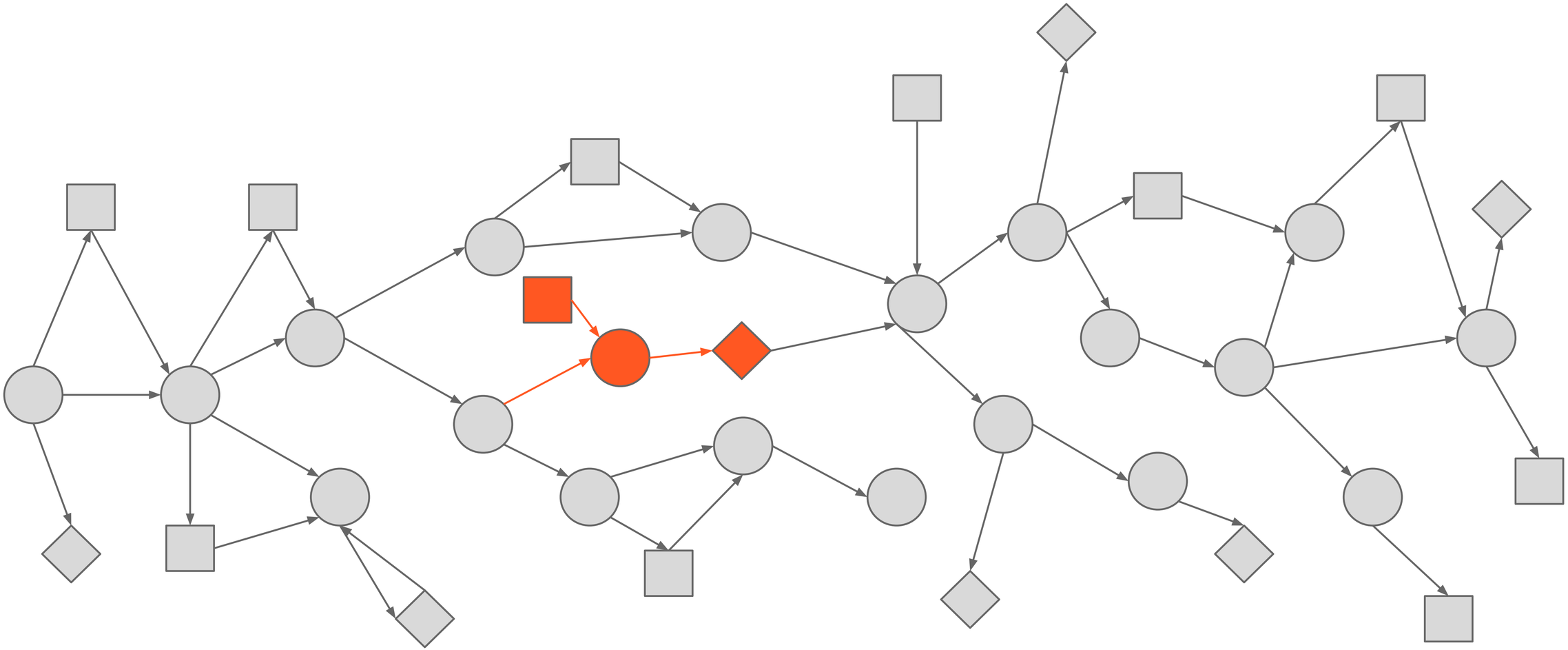
Nodes in the same batch as the attack are malicious



Batch of fixed duration (time window)

Source Attribution

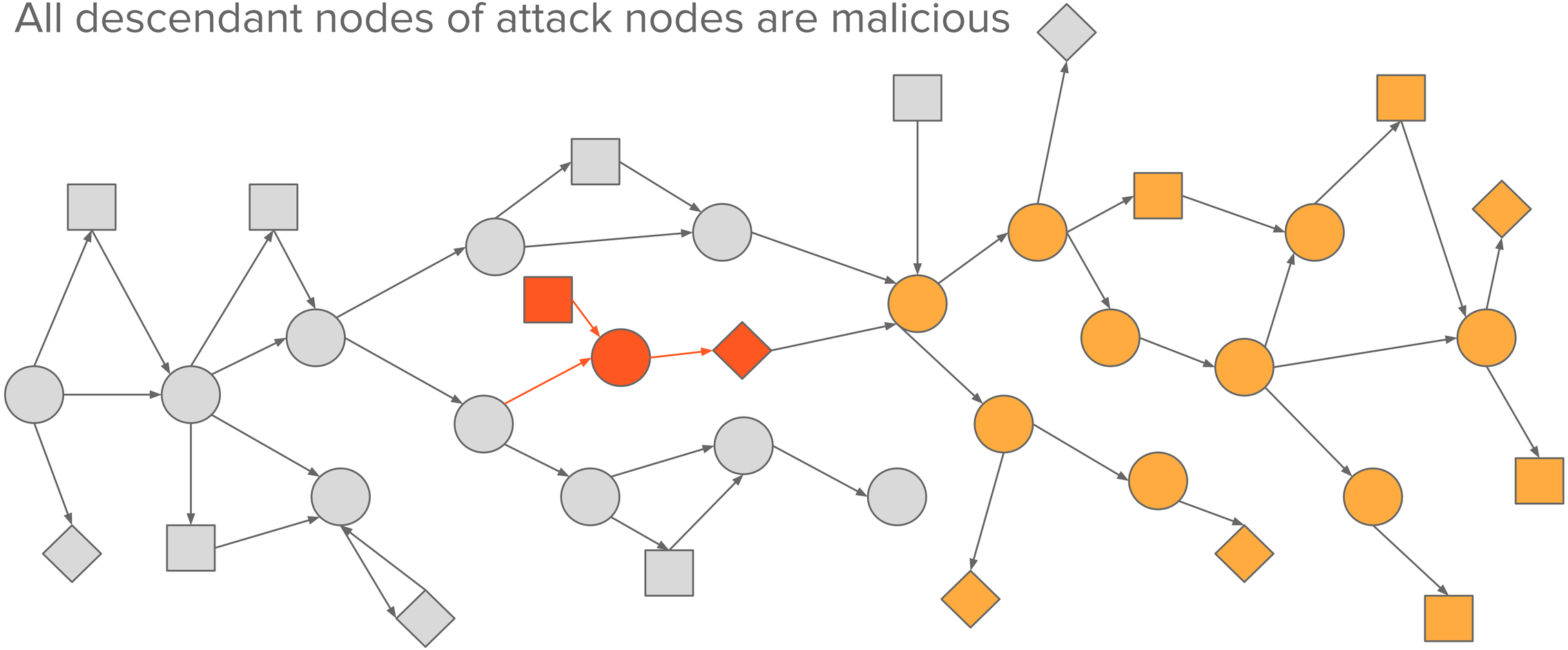
Concerned PIDS:
R-Caid, S&P 2024



Concerned PIDS:
R-Caid, S&P 2024

Source Attribution

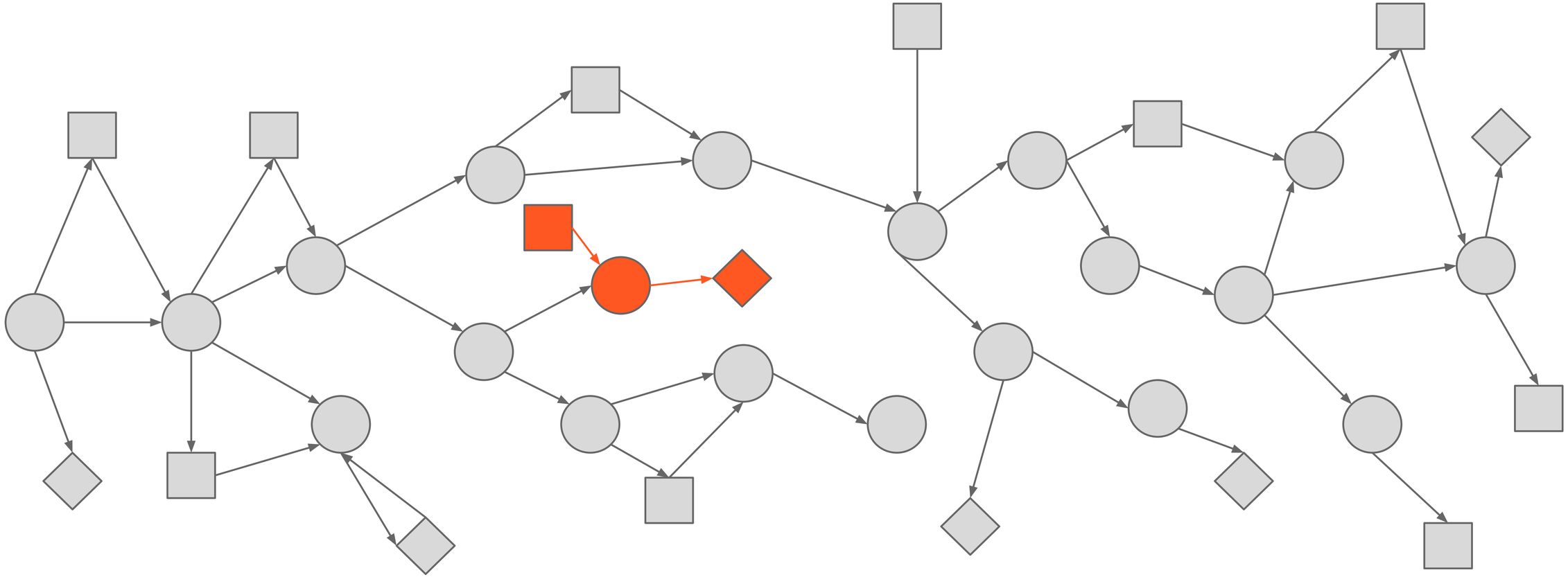
All descendant nodes of attack nodes are malicious



Concerned PIDS:
Orthrus, USENIX Sec 2025

Node Attribution (most fine grained)

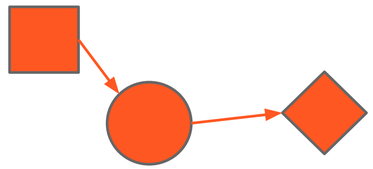
No tricks, we directly use the attack nodes from dataset ground truths



Concerned PIDS:
Orthrus, USENIX Sec 2025

Node Attribution (most fine grained)

No tricks, we directly use the attack nodes from dataset ground truths



- Much harder detection (few nodes)
- Concise detection reports (less work for analysts)

Concerned PIDS:
Orthrus, USENIX Sec 2025

Detection Granularity Comparison

- Assuming perfect detection based on their design
- Past systems overwhelm security analyst with large alerts

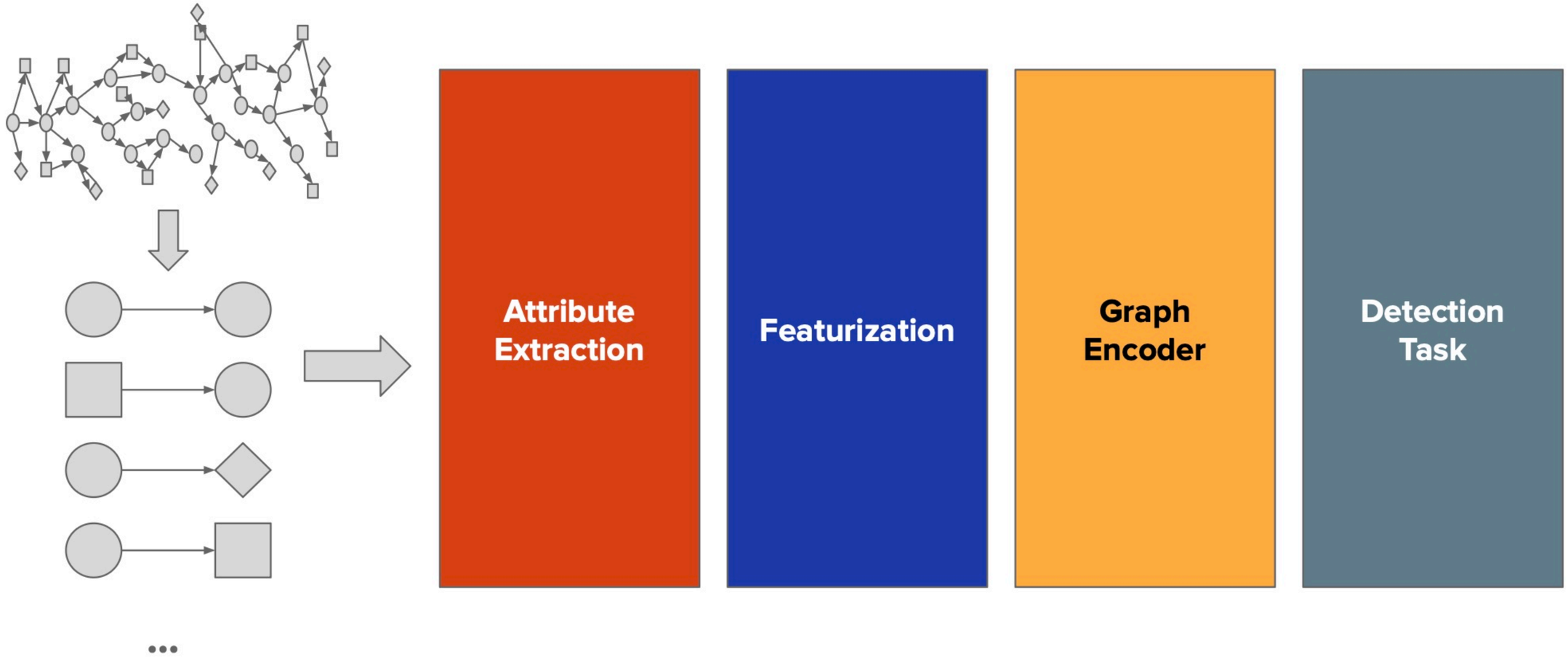
Number of attack nodes to detect per attribution strategy:

Dataset	Total Nodes	Neighborhood	Batch	Source	Node-level (Ours)
E5-CADETS	7,632,792	20,524	717,783	401,065	123
E5-THEIA	1,728,121	162,714	61,368	9,374	69
E5-CLEARSCOPE	326,338	48,488	8,636	1,020	51

Orthrus: Detecting APTs at the Node Level

Orthrus

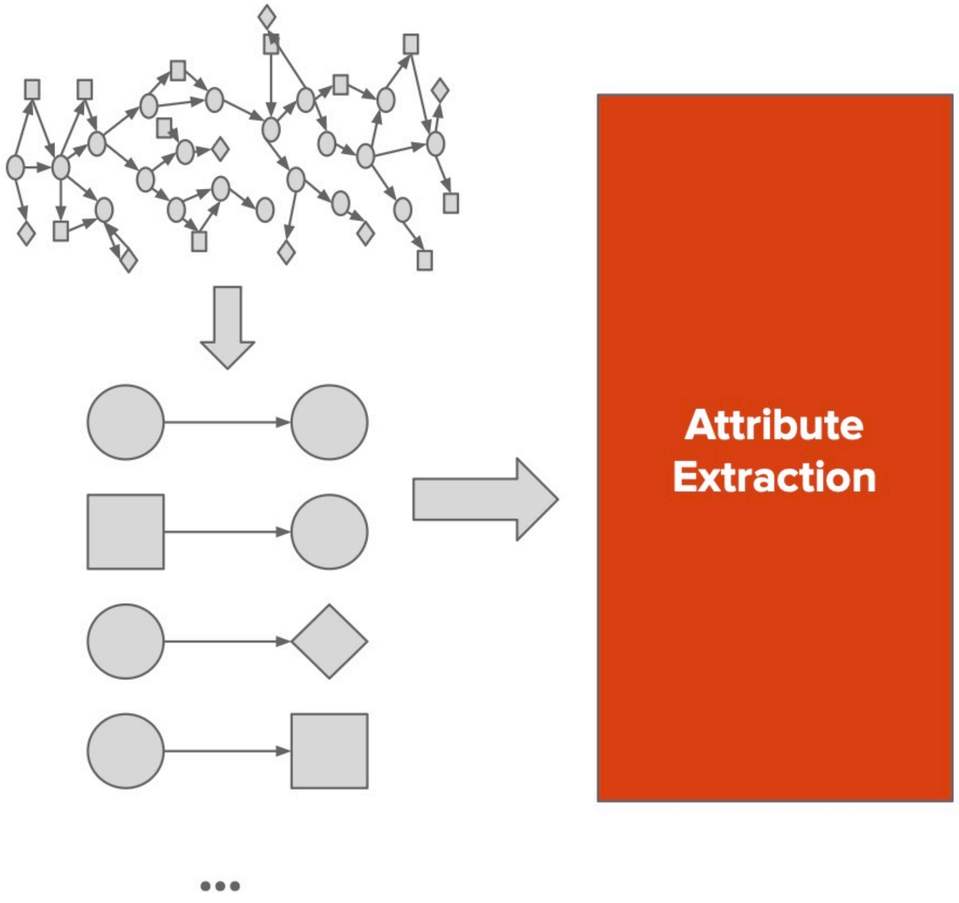
Design



Jiang, Baoxiang, et al. "ORTHRUS: Achieving High Quality of Attribution in Provenance-based Intrusion Detection Systems." *Security Symposium (USENIX Sec'25). USENIX*. 2025.

Orthrus

Design



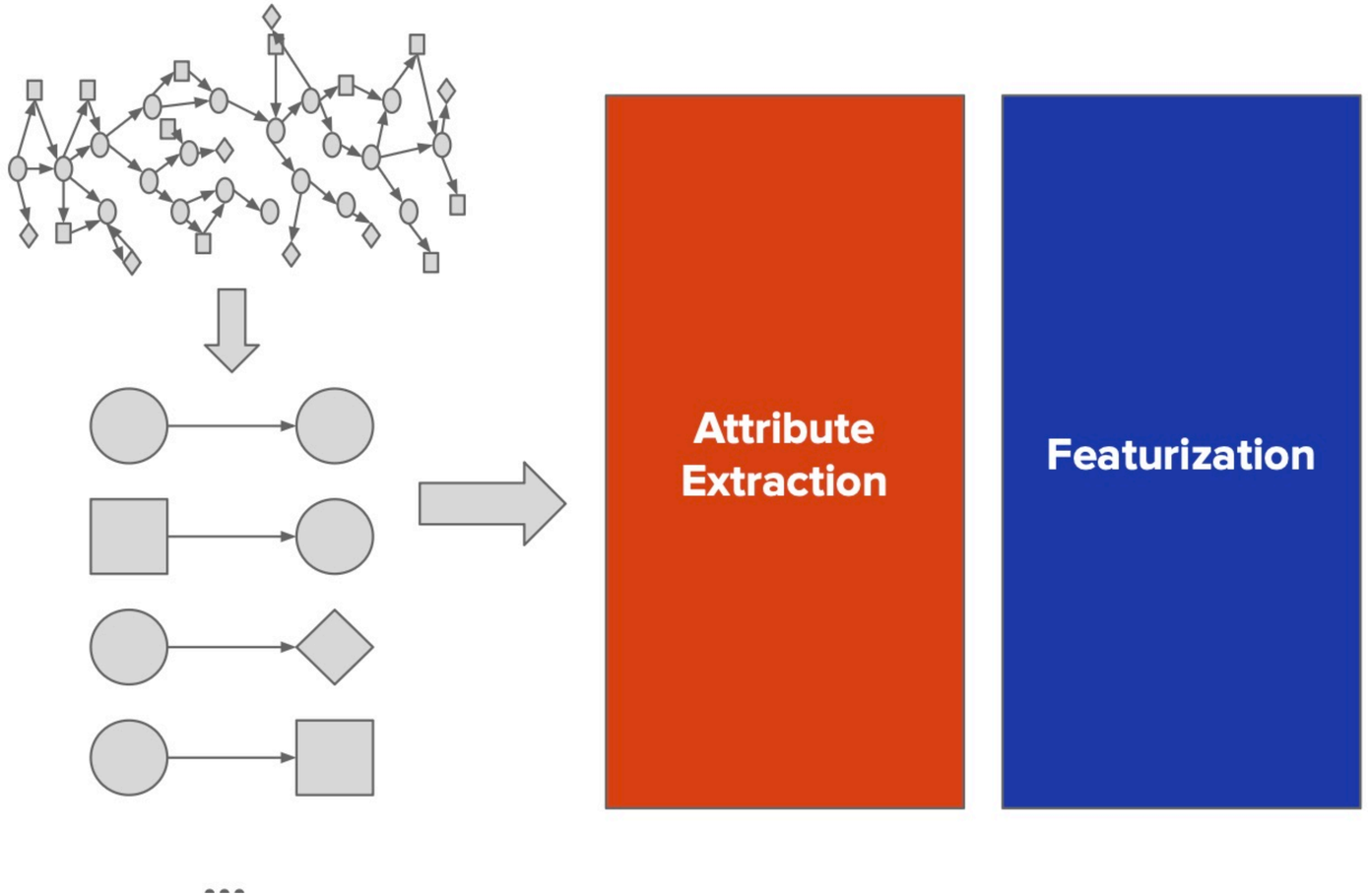
File: type + path (e.g. */etc/passwd)*

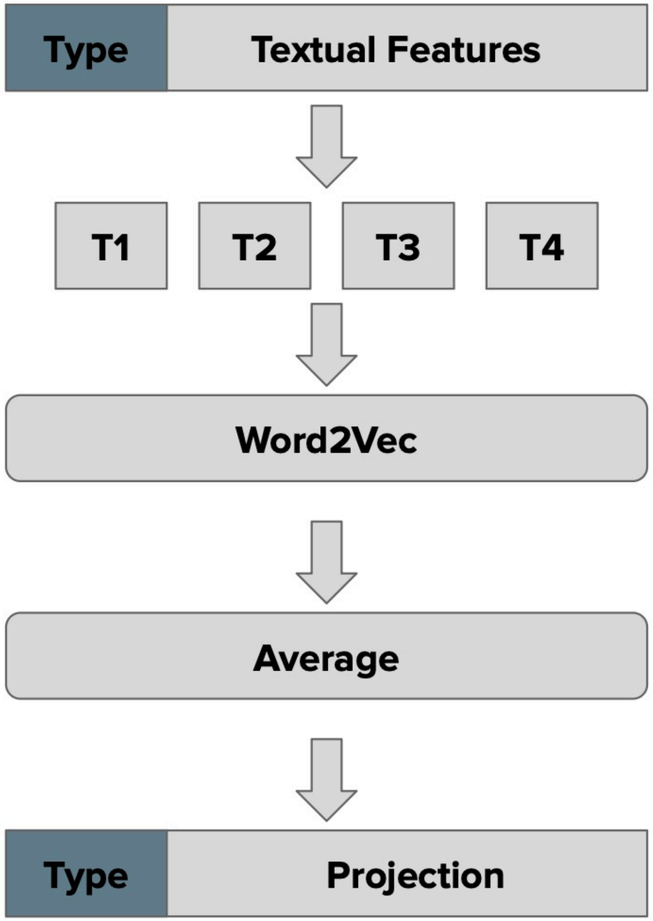
Process: type + cmd line (e.g. *ls -l -t -r)*

Netflow: type + IP + port (e.g. *192.168.1.2 80)*

Orthrus

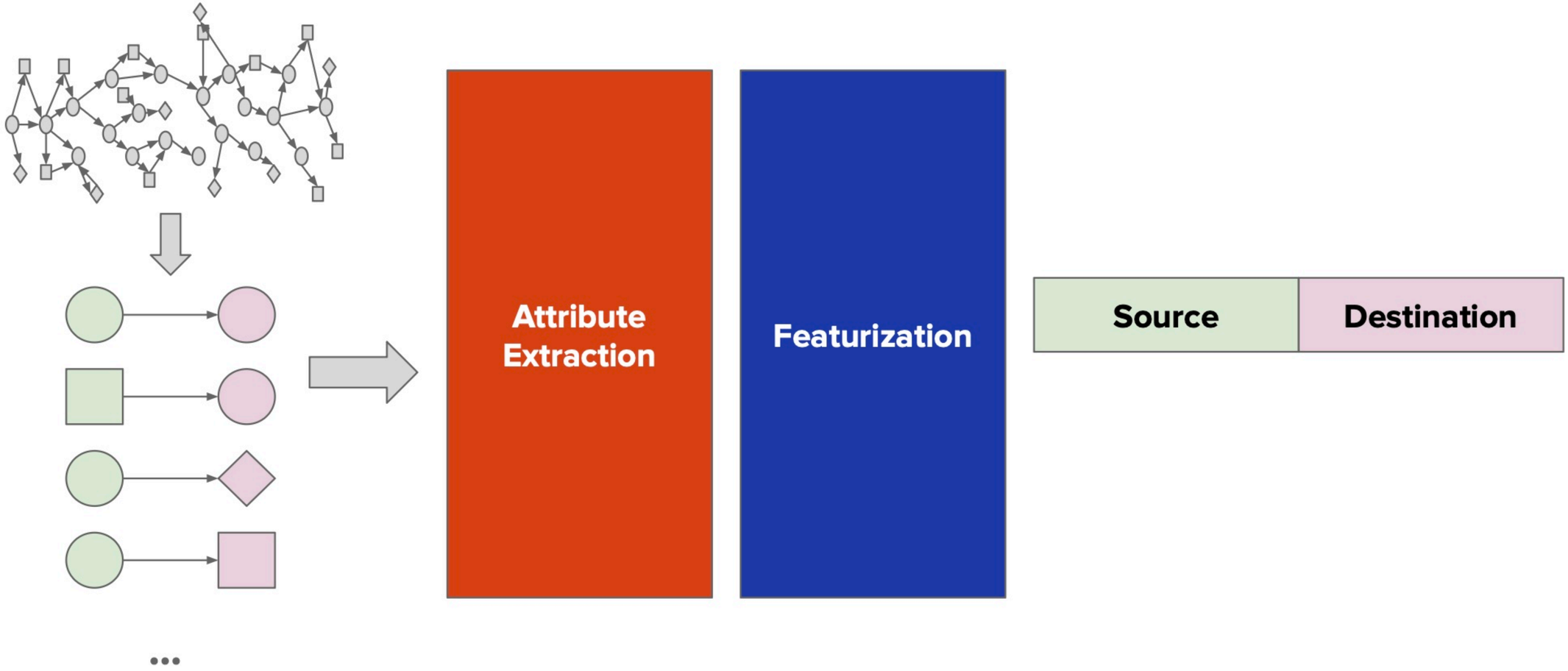
Design





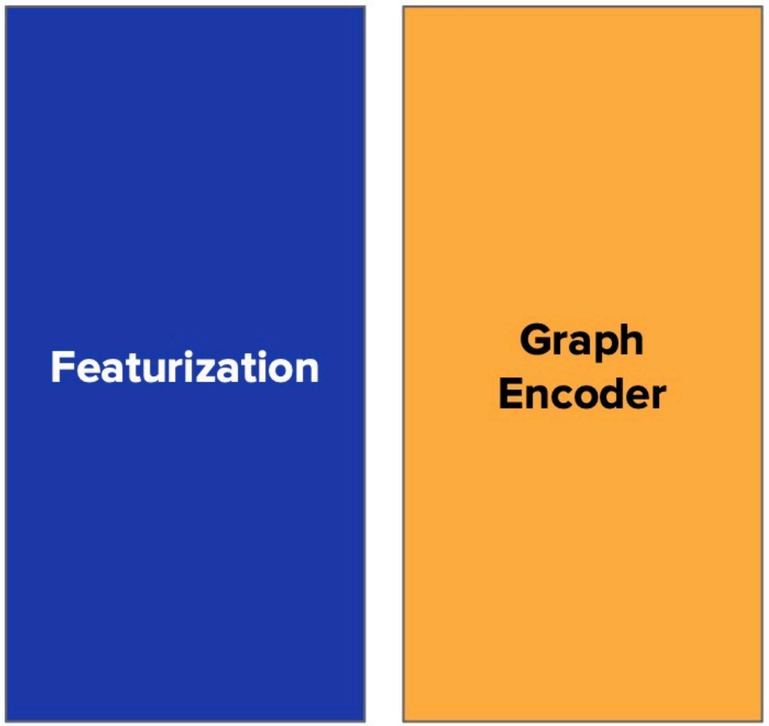
Orthrus

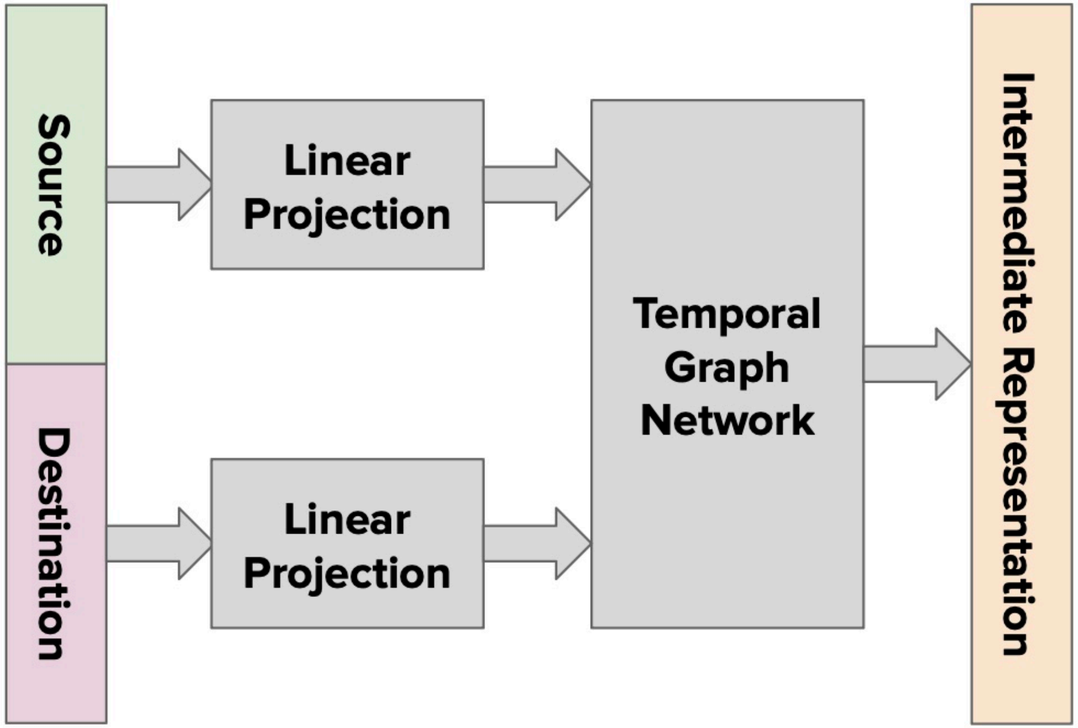
Design



Orthrus

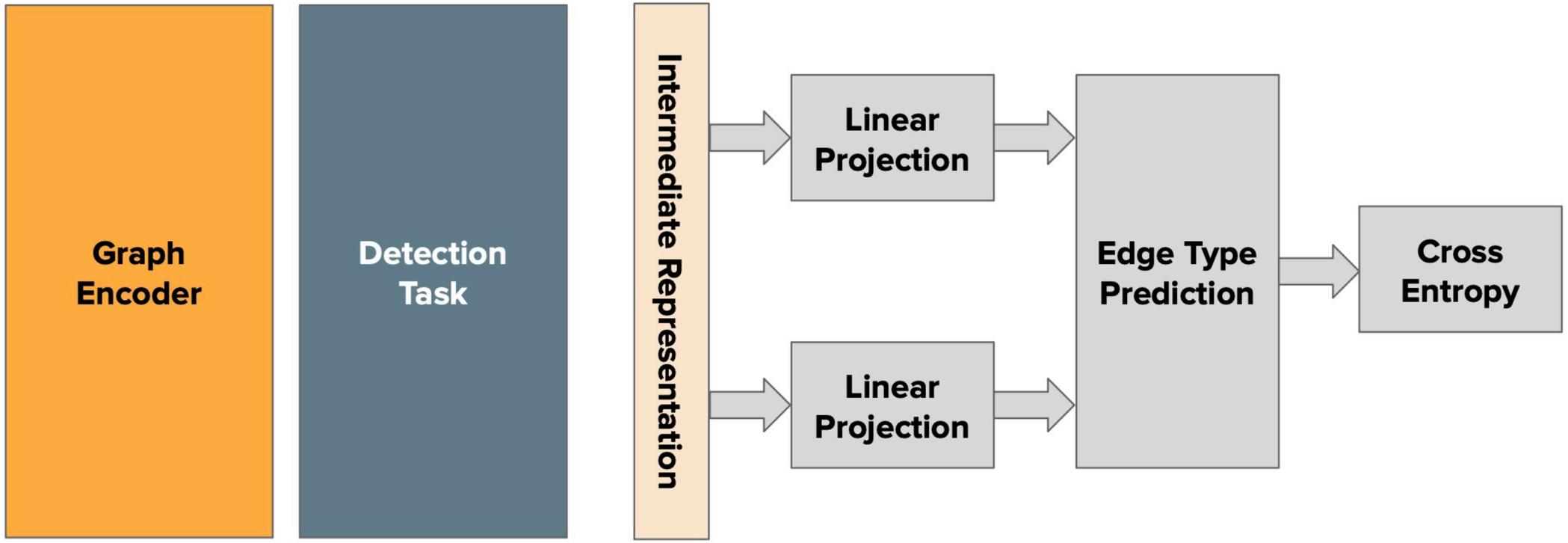
Design





Orthrus

Design



Orthrus

Evaluation: Node-level Results

DARPA TC E3 [25] **and E5** [26] datasets (1-150M edges, 300k-5M nodes)

Dataset	System	TP	FP	TN	FN	Precision	MCC
E3-CADETS	Kairos	0	9	268k	68	0.00	0.00
	ThreaTrace	61	252k	16k	7	0.00	0.00
	Sigl	0	80	268k	68	0.00	0.00
	Magic	63	80k	188k	5	0.00	0.02
	Flash	13	2.4k	266k	55	0.01	0.03
	Orthrus	10	0	268k	58	**1.00**	0.38
E3-THEIA	Kairos	4	0	699k	114	**1.00**	0.18
	ThreaTrace	88	672k	27k	30	0.00	-0.01
	Sigl	1	29	699k	117	0.03	0.02
	Magic	115	395k	304k	3	0.00	0.01
	Flash	22	32k	667k	96	0.00	0.01
	Orthrus	8	0	699k	110	**1.00**	0.26
E3-CLEARSCOPE	Kairos	0	7	111k	41	0.00	0.00
	ThreaTrace	41	88k	24k	0	0.00	0.01
	Sigl	1	11k	100k	40	0.00	0.00
	Magic	40	102k	9.6k	1	0.00	0.00
	Flash	0	15k	96k	41	0.00	-0.01
	Orthrus	1	1	111k	40	**0.50**	**0.11**

Dataset	System	TP	FP	TN	FN	Precision	MCC
E5-CADETS	Kairos	0	6	3M	123	0.00	0.00
	ThreaTrace	91	3M	7k	32	0.00	-0.03
	Sigl	0	66	3M	123	0.00	0.00
	Magic	123	3M	541	0	0.00	0.00
	Flash	45	34k	3M	78	0.00	0.02
	Orthrus	1	5	3M	122	**0.17**	0.04
E5-THEIA	Kairos	0	2	747k	69	0.00	0.00
	ThreaTrace	66	739k	8k	3	0.00	0.00
	Sigl	0	23	747k	69	0.00	0.00
	Magic	1	297k	451k	68	0.00	-0.01
	Flash	43	296k	452k	26	0.00	0.00
	Orthrus	2	0	747k	67	**1.00**	**0.17**
E5-CLEARSCOPE	Kairos	1	3	151k	50	0.25	0.07
	ThreaTrace	41	142k	8k	10	0.00	-0.01
	Sigl	10	63	151k	41	0.14	0.16
	Magic	51	139k	11k	0	0.00	0.01
	Flash	15	4.6k	146k	36	0.00	0.03
	Orthrus	2	7	151k	49	0.22	0.09

[25] DARPA, Transparent Computing Engagement 3 Data Release (2018). https://github.com/darpa-i2o/Transparent-Computing/blob/master/README-E3.md
[26] DARPA, Transparent Computing Engagement 5 Data Release (2019). https://github.com/darpa-i2o/Transparent-Computing

Building More Practical Systems

Building More Practical Systems

Current detection systems suffer from **9 key shortcomings (SC)** that **hinder real-world deployment:**

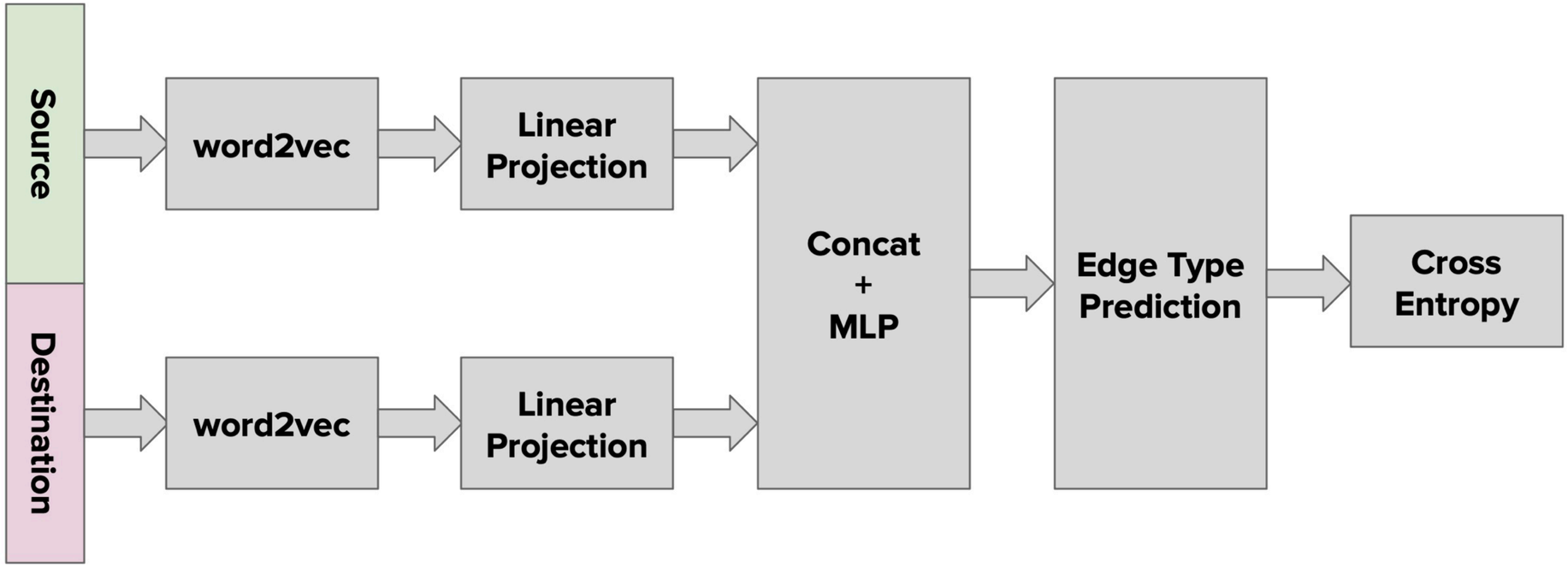
SC1: Insufficient Detection Granularity
SC2: Missing Metric to Measure Attack Detection
SC3: Impractical Thresholding Methods
SC4: Unfair Comparison with Baselines
SC5: Not Measuring Instability
SC6: Featurization Methods Trained on Test Data
SC7: Overly Complex Architectures
SC8: Insufficient Scalability
SC9: Lacking Real-Time Detection

Goal: design a practical PIDS that addresses these shortcomings

Bilot, Tristan, et al. "Sometimes Simpler is Better: A Comprehensive Analysis of {State-of-the-Art}{Provenance-Based} Intrusion Detection Systems." *34th USENIX Security Symposium (USENIX Security 25)*. 2025.

Velox

A **simple architecture** that addresses all shortcomings



- This design has been selected after **~453 days of GPU compute**
- Unlike all other PIDSs, no complex graph encoder is used

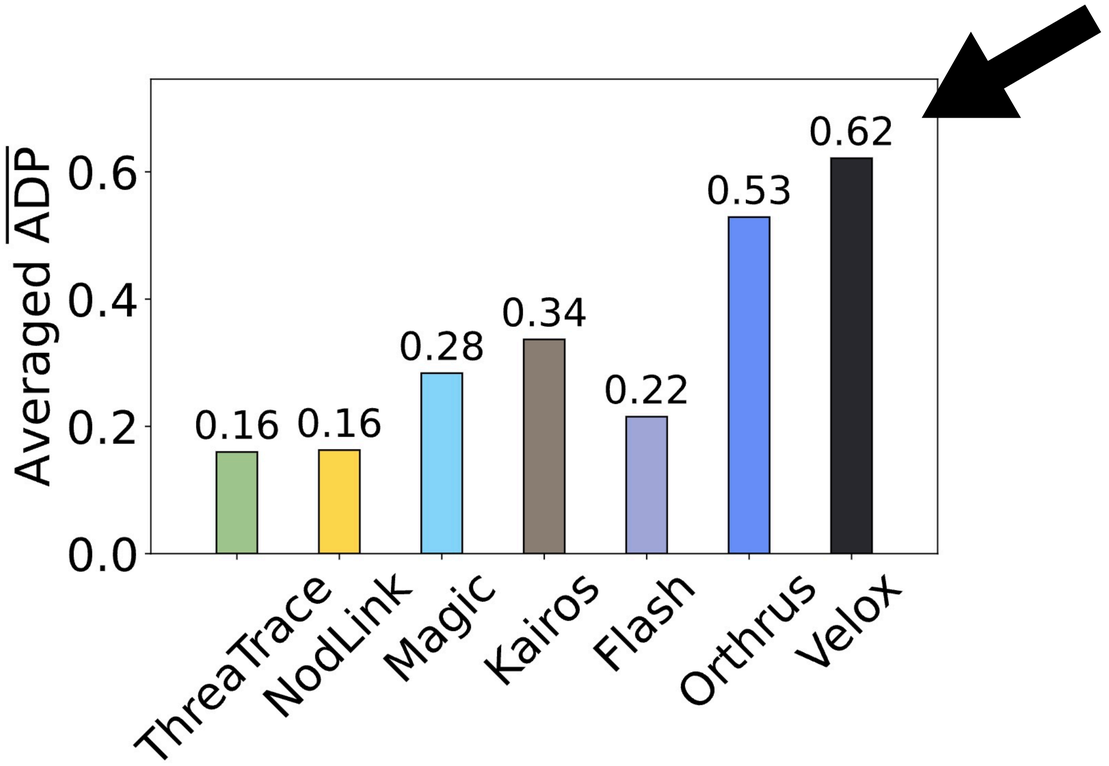
Bilot, Tristan, et al. "Sometimes Simpler is Better: A Comprehensive Analysis of {State-of-the-Art}{Provenance-Based} Intrusion Detection Systems." *34th USENIX Security Symposium (USENIX Security 25)*. 2025.

Velox

An Unexpected Discovery

Velox reaches **state-of-the-art performance**

Attack Detection Precision (ADP):
how well a model detects attacks with high precision



ADP averaged across 9 DARPA datasets

Importance of Architecture Search

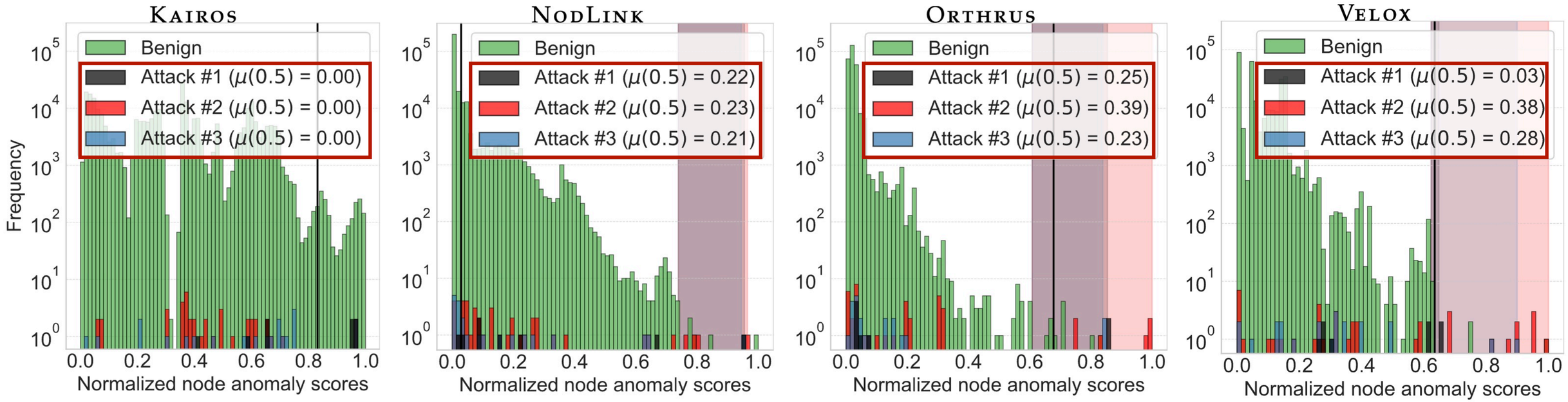
Key Findings

- Searching for simpler architectures is important
- Complexity may not be always needed
- Textual attributes suffice to detect most attacks

A Closer Look at the Distribution

Is the Detection Reliable?

Threshold margin $\mu(p)$: fraction of the anomaly score space where a threshold can be set to achieve at least precision p for a given attack



Node anomaly score distributions predicted on attacks #1, #2 and #3 in the E3-CADETS

Conclusion

- Self-supervised learning is **promising** for unknown attack detection
- Graph approaches should be **considered carefully**
- Such methods **should not replace** traditional ones, but **supplement them**

Limitations

And how to address them

Limitation		Future Work
Benign anomalies could be false positives	→	Design datasets with benign anomalies
Training instability due to SSL	→	Add some supervision during training
Concept drift could degrade detection	→	Establish dedicated benchmarks

Implementing an IDS From Scratch

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.nn import TransformerConv

class AttentionGNN(nn.Module):
    """
    Applies attention to an input provenance graph.
    """
    def __init__(self, in_dim, hid_dim, edge_dim=None, dropout=0.1):
        super().__init__()
        self.edge_dim = edge_dim
        self.drop = nn.Dropout(dropout)

        self.conv1 = TransformerConv(
            in_dim, hid_dim, heads=1, concat=False,
            dropout=dropout, edge_dim=edge_dim
        )
        self.conv2 = TransformerConv(
            hid_dim, hid_dim, heads=1, concat=False,
            dropout=dropout, edge_dim=edge_dim
        )

    def forward(self, x, edge_index, edge_attr=None):
        # x: [N, in_dim]
        if self.edge_dim is None:
            x = self.conv1(x, edge_index)          # -> [N, hid_dim]
            x = F.relu(x)
            x = self.drop(x)
            x = self.conv2(x, edge_index)          # -> [N, hid_dim]
        else:
            x = self.conv1(x, edge_index, edge_attr)
            x = F.relu(x)
            x = self.drop(x)
            x = self.conv2(x, edge_index, edge_attr)

        return x
```

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.nn import TransformerConv

class AttentionGNN(nn.Module):
    """
    Applies attention to an input provenance graph.
    """
    def __init__(self, in_dim, hid_dim, edge_dim=None, dropout=0.1):
        super().__init__()
        self.edge_dim = edge_dim
        self.drop = nn.Dropout(dropout)

        self.conv1 = TransformerConv(
            in_dim, hid_dim, heads=1, concat=False,
            dropout=dropout, edge_dim=edge_dim
        )
        self.conv2 = TransformerConv(
            hid_dim, hid_dim, heads=1, concat=False,
            dropout=dropout, edge_dim=edge_dim
        )

    def forward(self, x, edge_index, edge_attr=None):
        # x: [N, in_dim]
        if self.edge_dim is None:
            x = self.conv1(x, edge_index)          # -> [N, hid_dim]
            x = F.relu(x)
            x = self.drop(x)
            x = self.conv2(x, edge_index)          # -> [N, hid_dim]
        else:
            x = self.conv1(x, edge_index, edge_attr)
            x = F.relu(x)
            x = self.drop(x)
            x = self.conv2(x, edge_index, edge_attr)

        return x
```

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.nn import TransformerConv

class AttentionGNN(nn.Module):
    """
    Applies attention to an input provenance graph.
    """
    def __init__(self, in_dim, hid_dim, edge_dim=None, dropout=0.1):
        super().__init__()
        self.edge_dim = edge_dim
        self.drop = nn.Dropout(dropout)

        self.conv1 = TransformerConv(
            in_dim, hid_dim, heads=1, concat=False,
            dropout=dropout, edge_dim=edge_dim
        )
        self.conv2 = TransformerConv(
            hid_dim, hid_dim, heads=1, concat=False,
            dropout=dropout, edge_dim=edge_dim
        )

    def forward(self, x, edge_index, edge_attr=None):
        # x: [N, in_dim]
        if self.edge_dim is None:
            x = self.conv1(x, edge_index)          # -> [N, hid_dim]
            x = F.relu(x)
            x = self.drop(x)
            x = self.conv2(x, edge_index)          # -> [N, hid_dim]
        else:
            x = self.conv1(x, edge_index, edge_attr)
            x = F.relu(x)
            x = self.drop(x)
            x = self.conv2(x, edge_index, edge_attr)

        return x
```

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.nn import TransformerConv

class AttentionGNN(nn.Module):
    """
    Applies attention to an input provenance graph.
    """
    def __init__(self, in_dim, hid_dim, edge_dim=None, dropout=0.1):
        super().__init__()
        self.edge_dim = edge_dim
        self.drop = nn.Dropout(dropout)

        self.conv1 = TransformerConv(
            in_dim, hid_dim, heads=1, concat=False,
            dropout=dropout, edge_dim=edge_dim
        )
        self.conv2 = TransformerConv(
            hid_dim, hid_dim, heads=1, concat=False,
            dropout=dropout, edge_dim=edge_dim
        )

    def forward(self, x, edge_index, edge_attr=None):
        # x: [N, in_dim]
        if self.edge_dim is None:
            x = self.conv1(x, edge_index)          # -> [N, hid_dim]
            x = F.relu(x)
            x = self.drop(x)
            x = self.conv2(x, edge_index)          # -> [N, hid_dim]
        else:
            x = self.conv1(x, edge_index, edge_attr)
            x = F.relu(x)
            x = self.drop(x)
            x = self.conv2(x, edge_index, edge_attr)

        return x
```

```python
class NodePIDS(nn.Module):
    """
    For each node, outputs logits over node types.
    """
    def __init__(self, in_dim, num_types, hid_dim=64, edge_dim=None, dropout=0.1):
        super().__init__()
        self.gnn = TransformerGNN(in_dim, hid_dim, edge_dim=edge_dim, dropout=dropout)
        self.mlp = nn.Linear(hid_dim, num_types)        # [N, hid_dim] -> [N, num_types]

    def forward(self, data):
        h = self.gnn(data.x, data.edge_index, getattr(data, "edge_attr", None))  # [N, hid
        logits = self.mlp(h)                                                      # [N, num
        return logits

def train_epoch(model, train_set, opt, device):
    """
    Self-supervised training: predict node types (process/file/socket/...)
    """
    model.train()
    total, n = 0.0, 0

    for g in train_set:                     # one 15-min window
        g = g.to(device)

        logits = model(g)                   # [N, num_types]
        loss = F.cross_entropy(
            logits,                         # [N, num_types]
            g.node_type.long()              # [N]
        )

        opt.zero_grad()
        loss.backward()
        opt.step()

        total += float(loss)
        n += 1

    return total / max(n, 1)
```

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.nn import TransformerConv

class AttentionGNN(nn.Module):
    """
    Applies attention to an input provenance graph.
    """
    def __init__(self, in_dim, hid_dim, edge_dim=None, dropout=0.1):
        super().__init__()
        self.edge_dim = edge_dim
        self.drop = nn.Dropout(dropout)

        self.conv1 = TransformerConv(
            in_dim, hid_dim, heads=1, concat=False,
            dropout=dropout, edge_dim=edge_dim
        )
        self.conv2 = TransformerConv(
            hid_dim, hid_dim, heads=1, concat=False,
            dropout=dropout, edge_dim=edge_dim
        )

    def forward(self, x, edge_index, edge_attr=None):
        # x: [N, in_dim]
        if self.edge_dim is None:
            x = self.conv1(x, edge_index)          # -> [N, hid_dim]
            x = F.relu(x)
            x = self.drop(x)
            x = self.conv2(x, edge_index)          # -> [N, hid_dim]
        else:
            x = self.conv1(x, edge_index, edge_attr)
            x = F.relu(x)
            x = self.drop(x)
            x = self.conv2(x, edge_index, edge_attr)

        return x
```

```python
class NodePIDS(nn.Module):
    """
    For each node, outputs logits over node types.
    """
    def __init__(self, in_dim, num_types, hid_dim=64, edge_dim=None, dropout=0.1):
        super().__init__()
        self.gnn = TransformerGNN(in_dim, hid_dim, edge_dim=edge_dim, dropout=dropout)
        self.mlp = nn.Linear(hid_dim, num_types)        # [N, hid_dim] -> [N, num_types]

    def forward(self, data):
        h = self.gnn(data.x, data.edge_index, getattr(data, "edge_attr", None))  # [N, hid
        logits = self.mlp(h)                                                     # [N, num
        return logits

def train_epoch(model, train_set, opt, device):
    """
    Self-supervised training: predict node types (process/file/socket/...)
    """
    model.train()
    total, n = 0.0, 0

    for g in train_set:                     # one 15-min window
        g = g.to(device)

        logits = model(g)                   # [N, num_types]
        loss = F.cross_entropy(
            logits,                         # [N, num_types]
            g.node_type.long()              # [N]
        )

        opt.zero_grad()
        loss.backward()
        opt.step()

        total += float(loss)
        n += 1

    return total / max(n, 1)
```

```python
def main(train_set, val_set, test_set, in_dim, edge_dim=None, epochs=5):
    device = "cuda" if torch.cuda.is_available() else "cpu"
    model = NodePIDS(in_dim, hid_dim=64, edge_dim=edge_dim).to(device)
    opt = torch.optim.Adam(model.parameters(), lr=1e-3)

    for e in range(1, epochs + 1):
        loss = train_epoch(model, train_set, opt, device)
        print(f"epoch={e:02d} train_loss={loss:.4f}")

    threshold = compute_threshold(model, val_set, device)
    print(f"threshold (max val node loss) = {threshold:.6f}")

    prec, rec, f1 = test(model, test_set, threshold, device)
    print(f"test: precision={prec:.3f} recall={rec:.3f} f1={f1:.3f}")
```

```python
def main(train_set, val_set, test_set, in_dim, edge_dim=None, epochs=5):
    device = "cuda" if torch.cuda.is_available() else "cpu"
    model = NodePIDS(in_dim, hid_dim=64, edge_dim=edge_dim).to(device)
    opt = torch.optim.Adam(model.parameters(), lr=1e-3)

    for e in range(1, epochs + 1):
        loss = train_epoch(model, train_set, opt, device)
        print(f"epoch={e:02d} train_loss={loss:.4f}")

    threshold = compute_threshold(model, val_set, device)
    print(f"threshold (max val node loss) = {threshold:.6f}")

    prec, rec, f1 = test(model, test_set, threshold, device)
    print(f"test: precision={prec:.3f} recall={rec:.3f} f1={f1:.3f}")
```

```python
def main(train_set, val_set, test_set, in_dim, edge_dim=None, epochs=5):
    device = "cuda" if torch.cuda.is_available() else "cpu"
    model = NodePIDS(in_dim, hid_dim=64, edge_dim=edge_dim).to(device)
    opt = torch.optim.Adam(model.parameters(), lr=1e-3)

    for e in range(1, epochs + 1):
        loss = train_epoch(model, train_set, opt, device)
        print(f"epoch={e:02d} train_loss={loss:.4f}")

    threshold = compute_threshold(model, val_set, device)
    print(f"threshold (max val node loss) = {threshold:.6f}")

    prec, rec, f1 = test(model, test_set, threshold, device)
    print(f"test: precision={prec:.3f} recall={rec:.3f} f1={f1:.3f}")
```

```python
@torch.no_grad()
def compute_threshold(model, val_set, device):
    model.eval()
    threshold = 0.0

    for g in val_set:
        g = g.to(device)
        logits = model(g)                                  # [N]
        node_loss = F.binary_cross_entropy_with_logits(
            logits, g.y.float(), reduction="none"  # [N]
        )
        threshold = max(threshold,
float(node_loss.max()))
    return threshold
```

```python
def main(train_set, val_set, test_set, in_dim, edge_dim=None, epochs=5):
    device = "cuda" if torch.cuda.is_available() else "cpu"
    model = NodePIDS(in_dim, hid_dim=64, edge_dim=edge_dim).to(device)
    opt = torch.optim.Adam(model.parameters(), lr=1e-3)

    for e in range(1, epochs + 1):
        loss = train_epoch(model, train_set, opt, device)
        print(f"epoch={e:02d} train_loss={loss:.4f}")

    threshold = compute_threshold(model, val_set, device)
    print(f"threshold (max val node loss) = {threshold:.6f}")

    prec, rec, f1 = test(model, test_set, threshold, device)
    print(f"test: precision={prec:.3f} recall={rec:.3f} f1={f1:.3f}")
```

```python
@torch.no_grad()
def compute_threshold(model, val_set, device):
    model.eval()
    threshold = 0.0

    for g in val_set:
        g = g.to(device)
        logits = model(g)                                   # [N]
        node_loss = F.binary_cross_entropy_with_logits(
            logits, g.y.float(), reduction="none"   # [N]
        )
        threshold = max(threshold,
float(node_loss.max()))
    return threshold
```

```python
@torch.no_grad()
def test(model, test_set, threshold, device):
    model.eval()
    tp = fp = fn = 0

    for g in test_set:
        g = g.to(device)
        logits = model(g)                                   # [N]
        node_loss = F.binary_cross_entropy_with_logits(
            logits, g.y.float(), reduction="none"
        )
        pred = (node_loss > threshold).long()       # [N]
        y = g.y.long()

        tp += int(((pred == 1) & (y == 1)).sum())
        fp += int(((pred == 1) & (y == 0)).sum())
        fn += int(((pred == 0) & (y == 1)).sum())

    prec = tp / max(tp + fp, 1)
    rec  = tp / max(tp + fn, 1)
    f1   = 2 * prec * rec / max(prec + rec, 1e-12)
    return prec, rec, f1
```

A Framework for Designing Neural Network-based IDSs

What Researchers Usually Do

When working on APT detection, most researchers:

- implement their system from scratch
- re-implement systems from the literature for benchmarking
- preprocess very large datasets
- sometimes, label the attacks by hand

Drawbacks

- a lot of work => weeks of work before beginning actual research
- inconsistent evaluation compared to existing systems

PIDSMaker

A framework for PIDS experimentation

Repo:
https://github.com/ubc-provenance/PIDSMaker

README | Contributing | Apache-2.0 license

PIDSMaker

Paper | Documentation | Installation

The first framework designed to build and experiment with provenance-based intrusion detection systems (PIDSs) using deep learning architectures. It provides a single codebase to run most recent state-of-the-arts systems and easily customize them to develop new variants.

Currently supported PIDSs:

- **Velox** (USENIX Sec'25): Sometimes Simpler is Better: A Comprehensive Analysis of State-of-the-Art Provenance-Based Intrusion Detection Systems
- **Orthrus** (USENIX Sec'25): ORTHRUS: Achieving High Quality of Attribution in Provenance-based Intrusion Detection Systems
- **R-Caid** (IEEE S&P'24): R-CAID: Embedding Root Cause Analysis within Provenance-based Intrusion Detection
- **Flash** (IEEE S&P'24): Flash: A Comprehensive Approach to Intrusion Detection via Provenance Graph Representation Learning
- **Kairos** (IEEE S&P'24): Kairos: Practical Intrusion Detection and Investigation using Whole-system Provenance
- **Magic** (USENIX Sec'24): MAGIC: Detecting Advanced Persistent Threats via Masked Graph Representation Learning
- **NodLink** (NDSS'24): NODLINK: An Online System for Fine-Grained APT Attack Detection and Investigation
- **ThreaTrace** (IEEE TIFS'22): THREATRACE: Detecting and Tracing Host-Based Threats in Node Level Through

PIDSMaker

A framework for PIDS experimentation

Integrates 8 state-of-the-art PIDSs in a **unified** (open source) **framework**

1. **Kairos**
2. **ThreaTrace**
3. **NodLink**
4. **Magic**
5. **Flash**
6. **R-Caid**
7. **Orthrus**
8. **Velox**

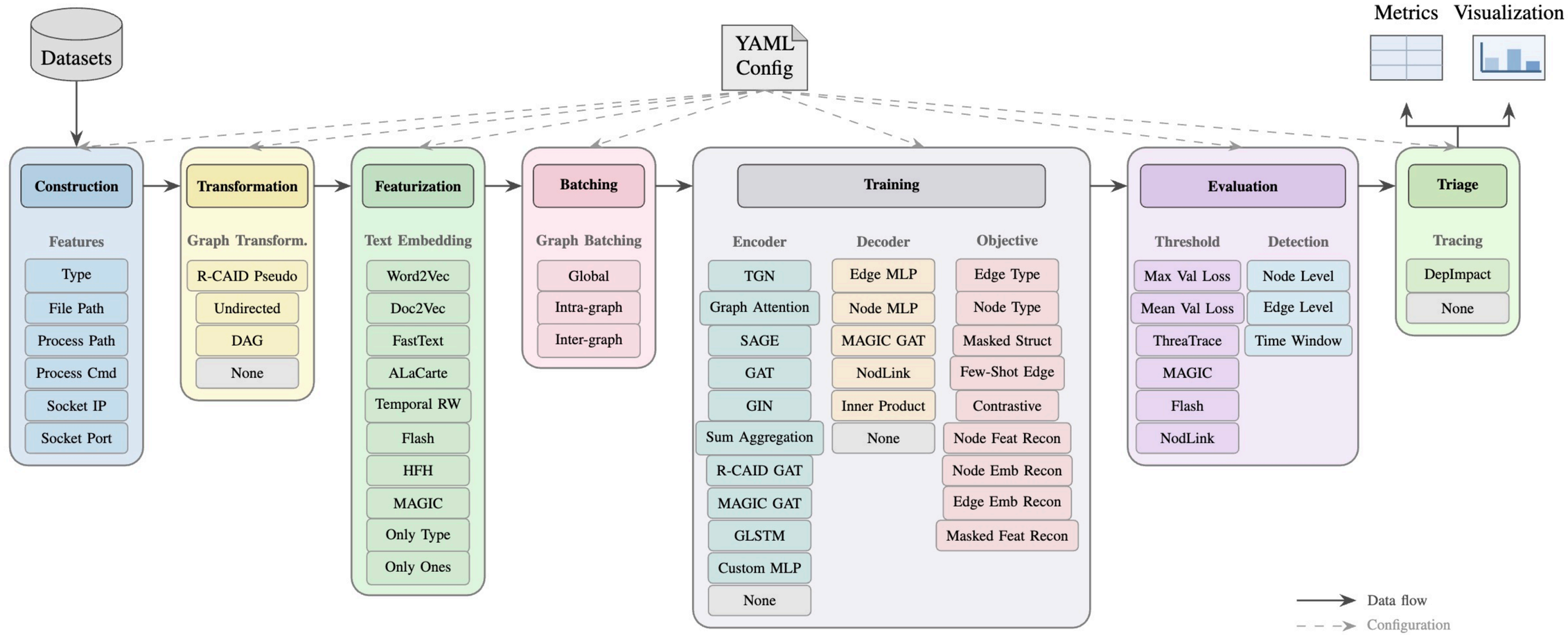
Goal:

- Combinatorial architecture search
- Easy evaluation

orthrus.yml

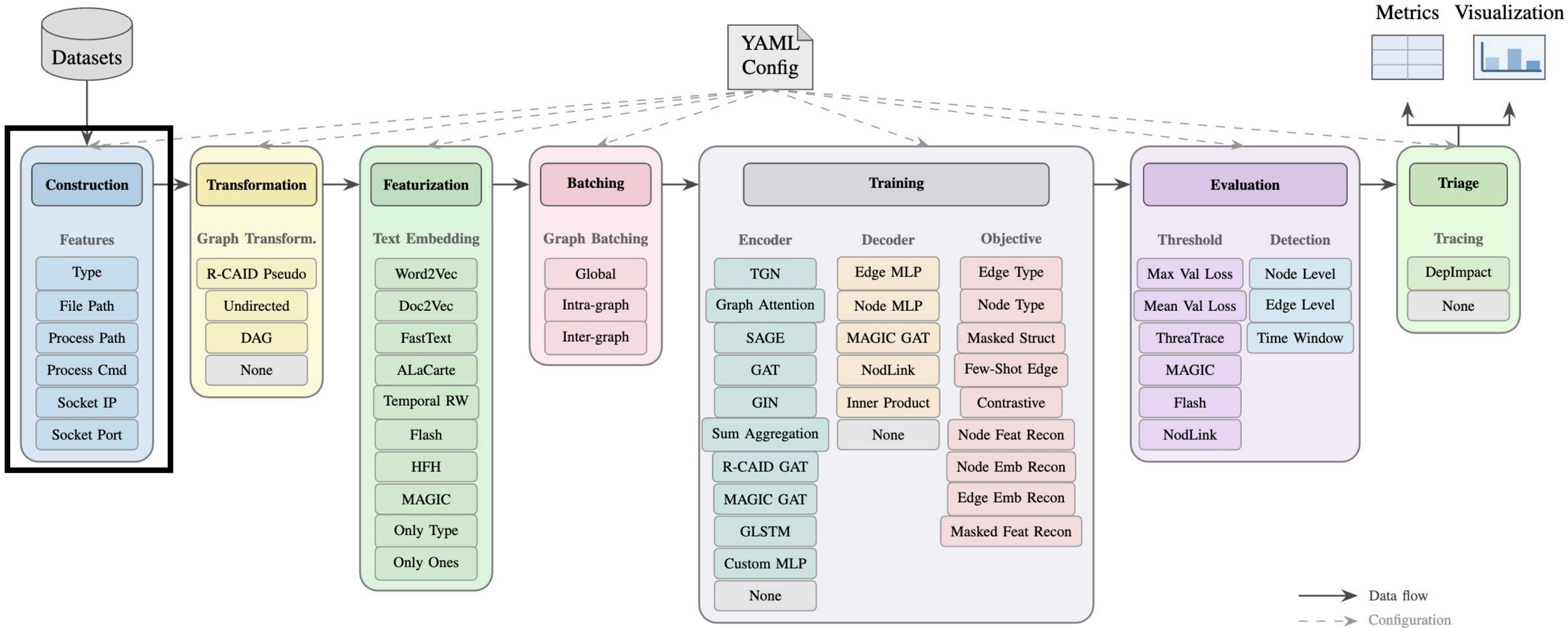
```yaml
construction:
  node_features:
    subject: type, path, cmd_line
    file: type, path
    netflow: type, remote_ip, remote_port
    [...]
transformation:
  used_methods: none
featurization:
  used_method: word2vec
  emb_dim: 128
  epochs: 50
  word2vec:
    alpha: 0.025
    [...]
batching:
  intra_graph_batching:
    used_methods: edges, tgn_last_neighbor
    [...]
training:
  lr: 0.00001
  node_hid_dim: 128
  encoder:
    used_methods: tgn, graph_attention
    graph_attention:
      num_heads: 8
      [...]
  decoder:
    [...]
evaluation:
  used_method: node_evaluation
  node_evaluation:
    threshold_method: max_val_loss
    [...]
triage:
  used_method: depimpact
  use_kmeans: True
  [...]
```

PIDSMaker's Pipeline



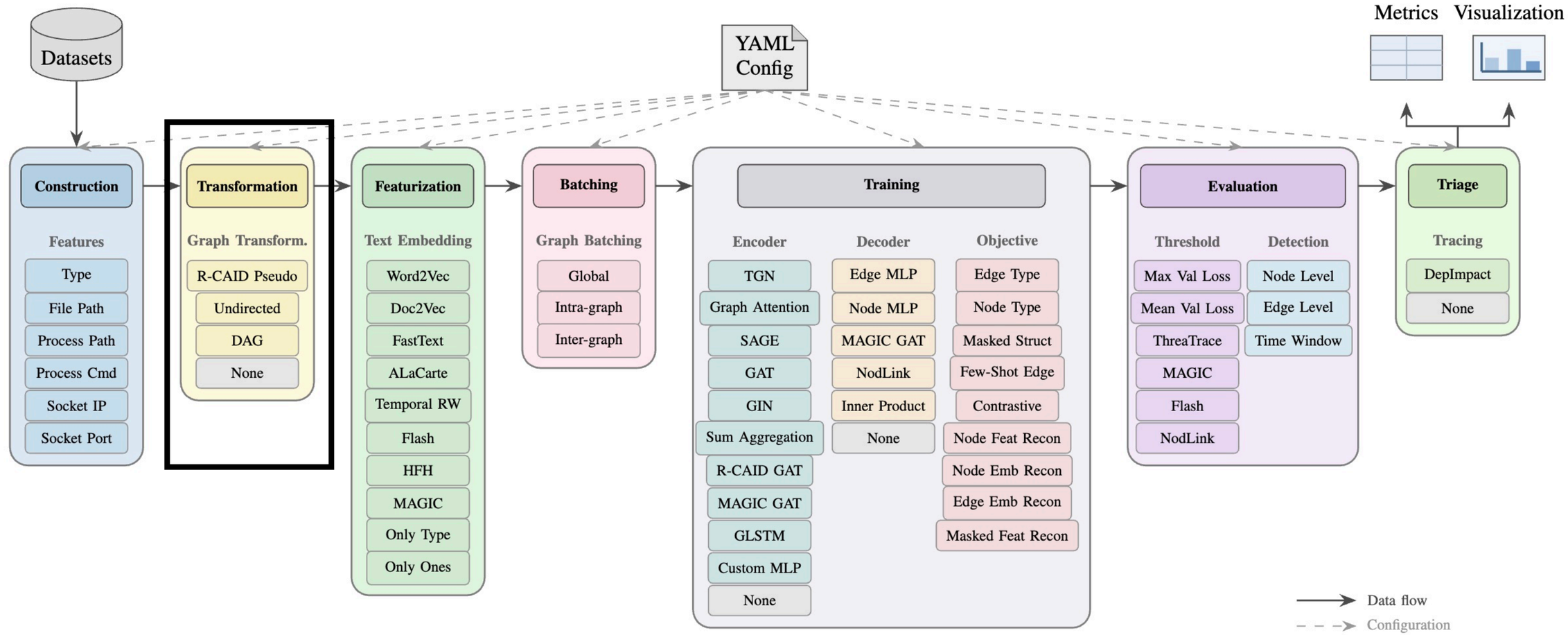
Modular pipeline with on-disk caching and automatic restarting

PIDSMaker's Pipeline



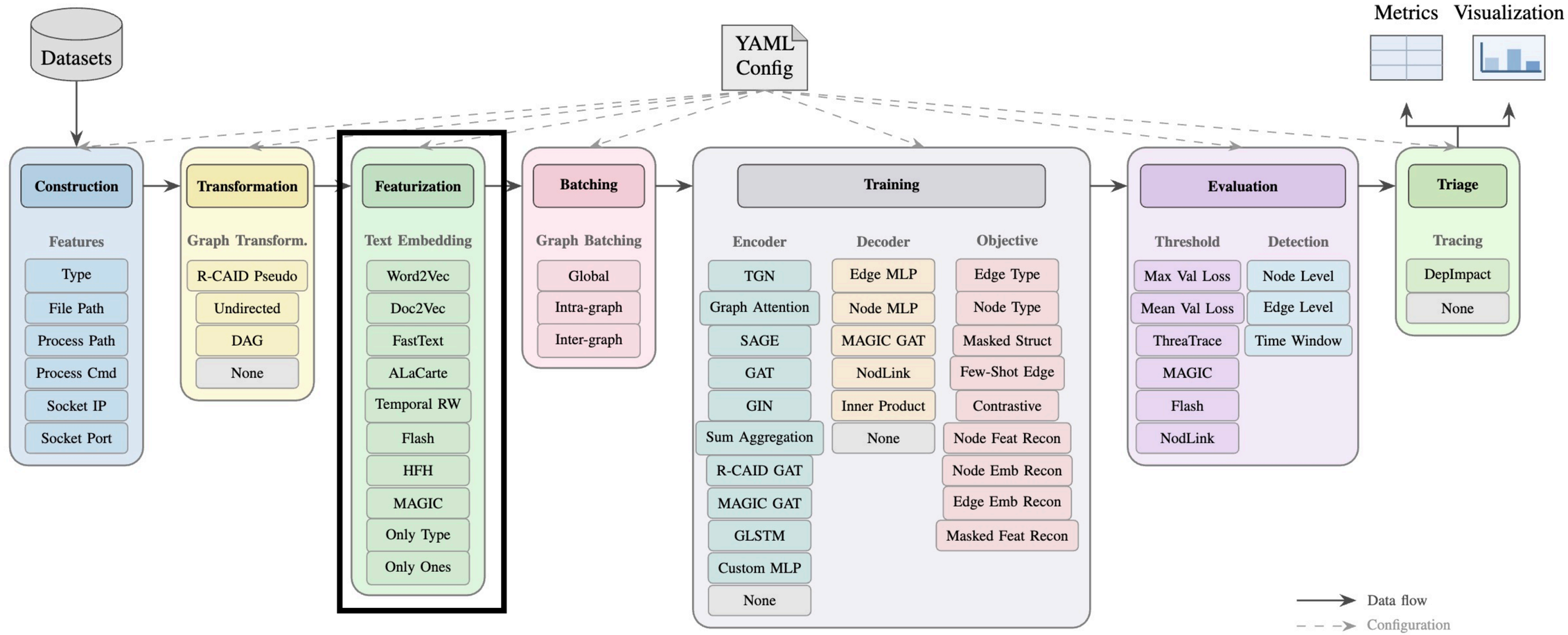
Modular pipeline with on-disk caching and automatic restarting

PIDSMaker's Pipeline



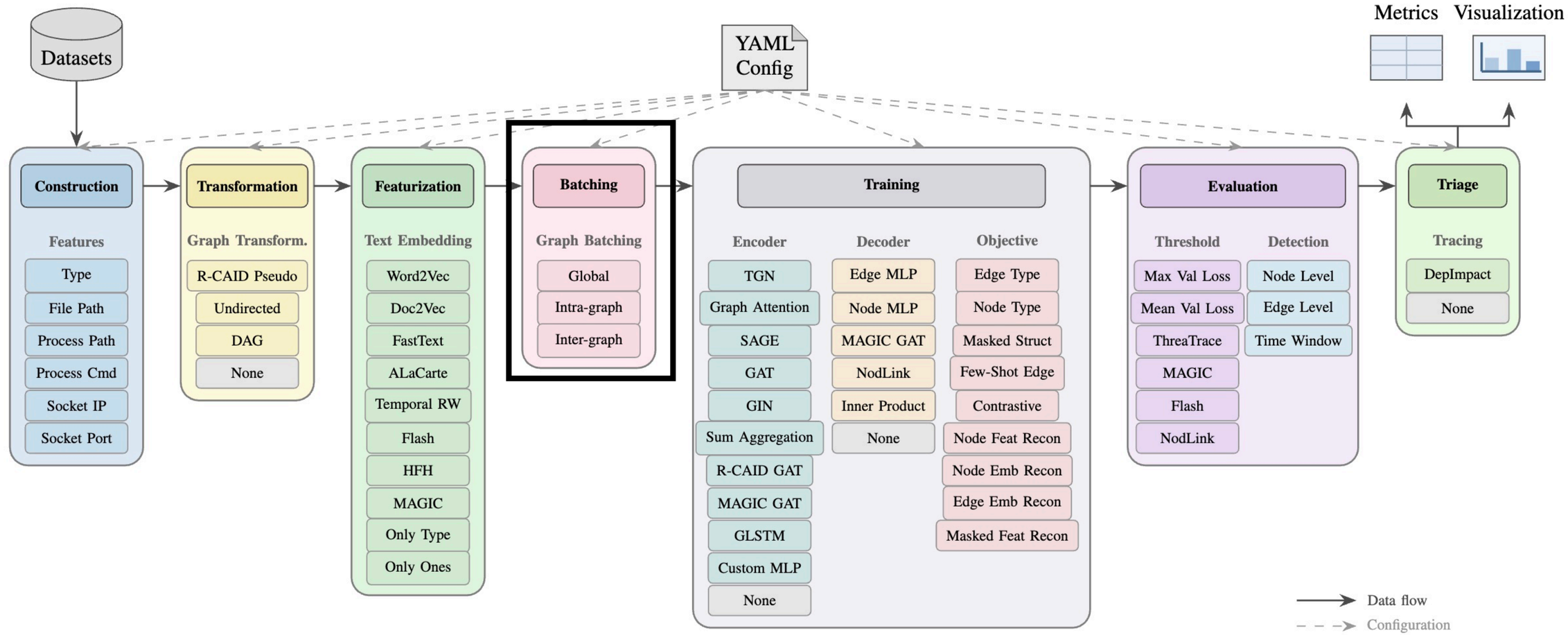
Modular pipeline with on-disk caching and automatic restarting

PIDSMaker's Pipeline



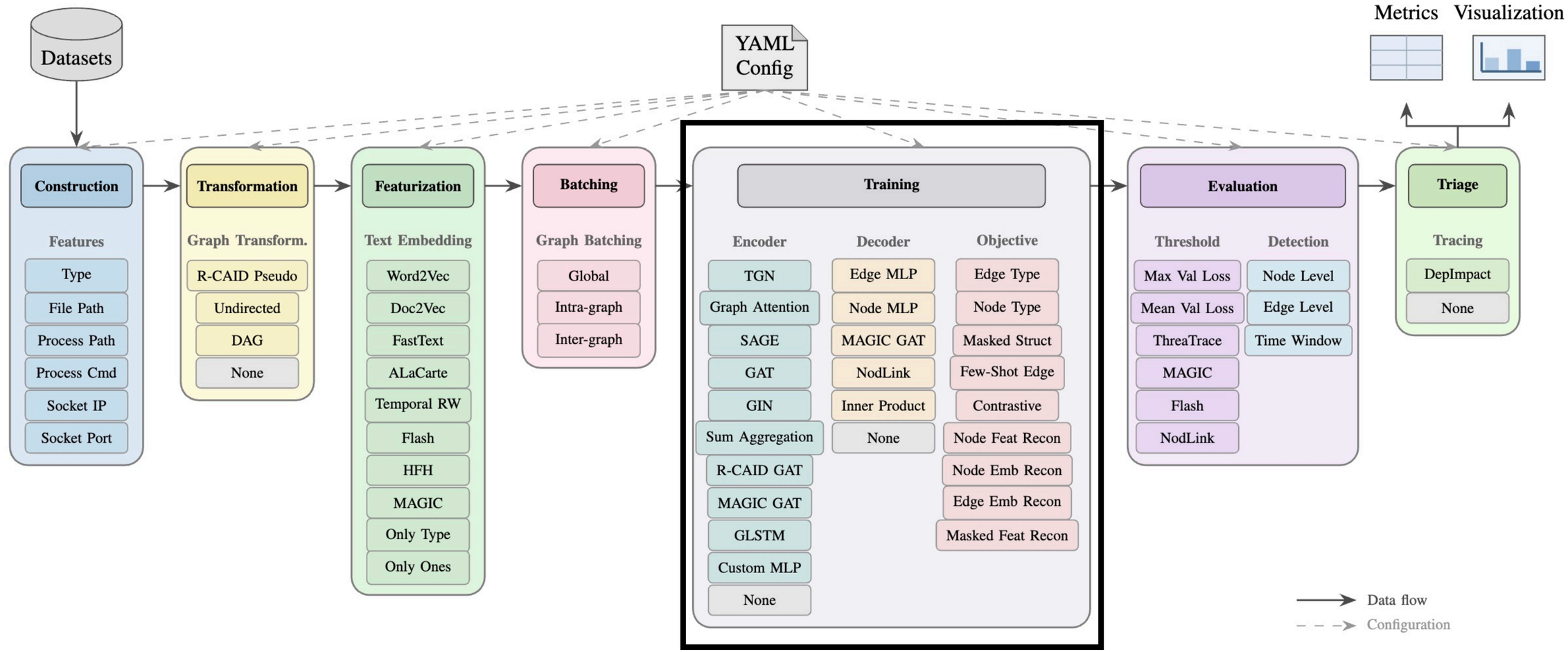
Modular pipeline with on-disk caching and automatic restarting

PIDSMaker's Pipeline



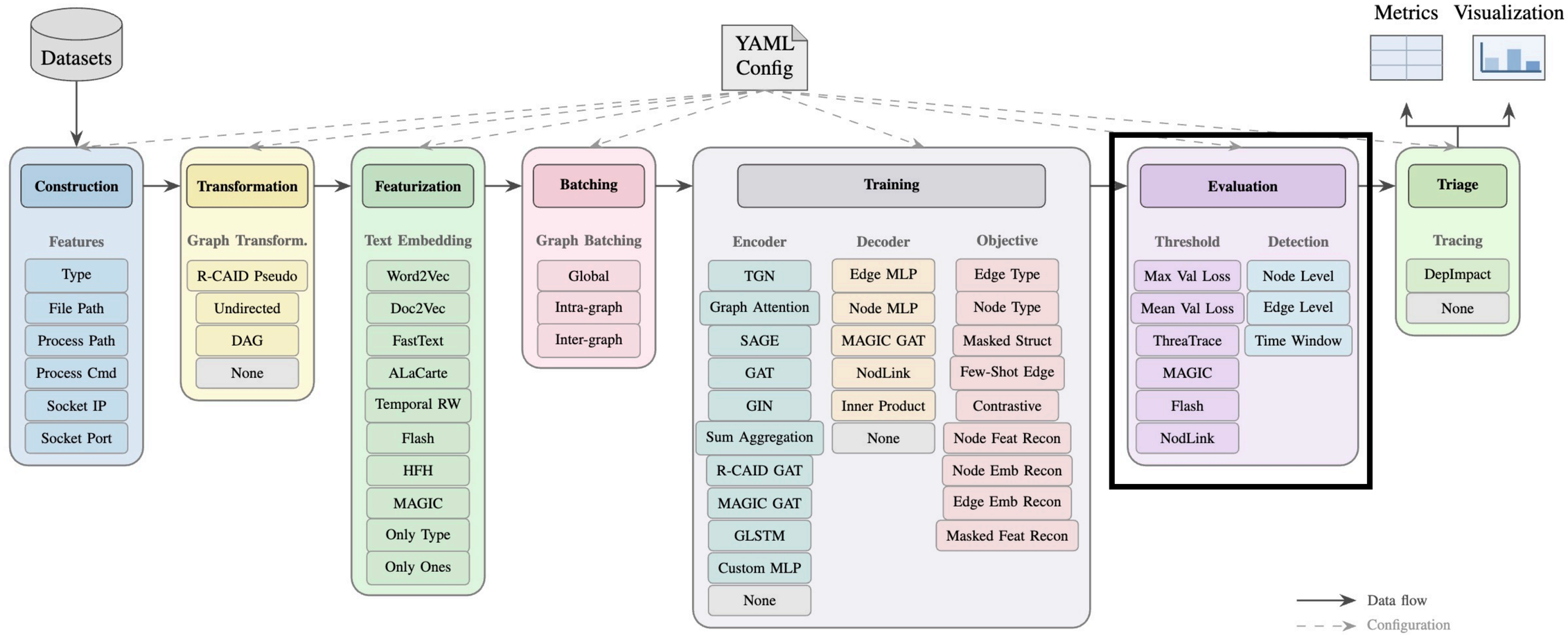
Modular pipeline with on-disk caching and automatic restarting

PIDSMaker's Pipeline



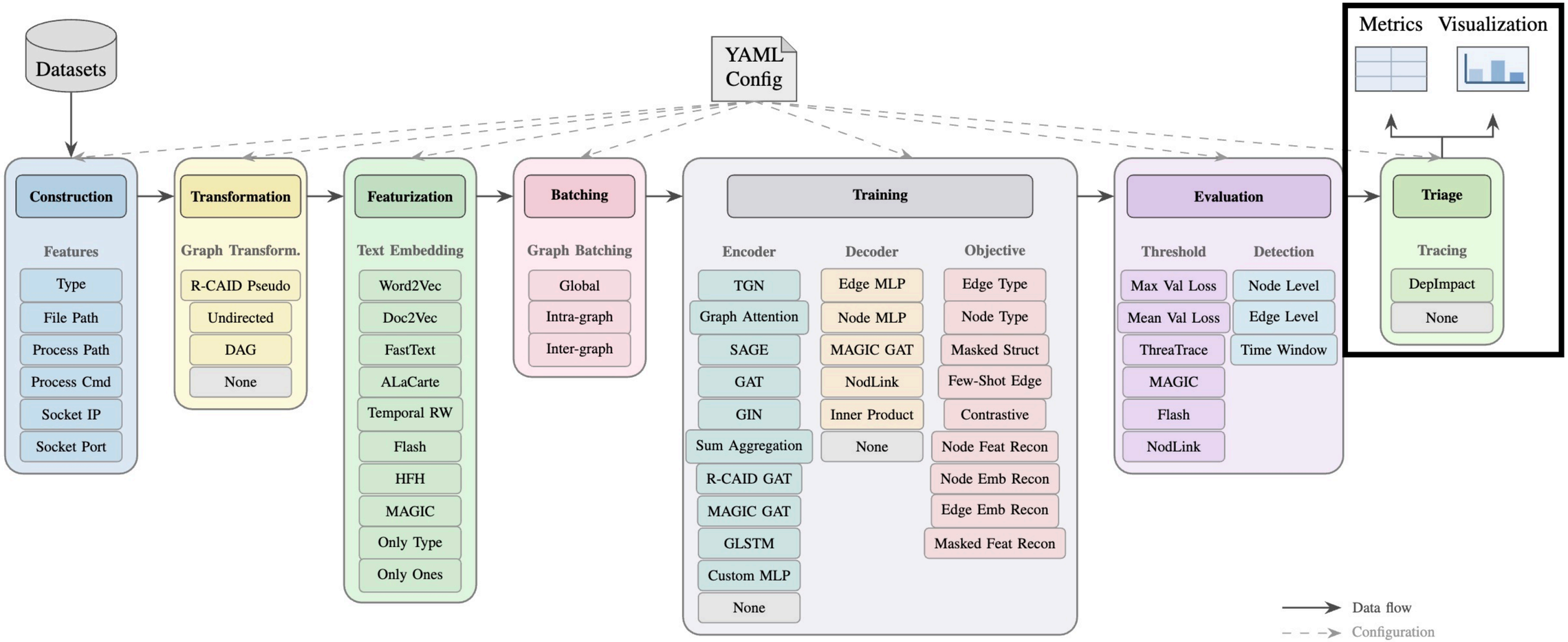
Modular pipeline with on-disk caching and automatic restarting

PIDSMaker's Pipeline



Modular pipeline with on-disk caching and automatic restarting

PIDSMaker's Pipeline



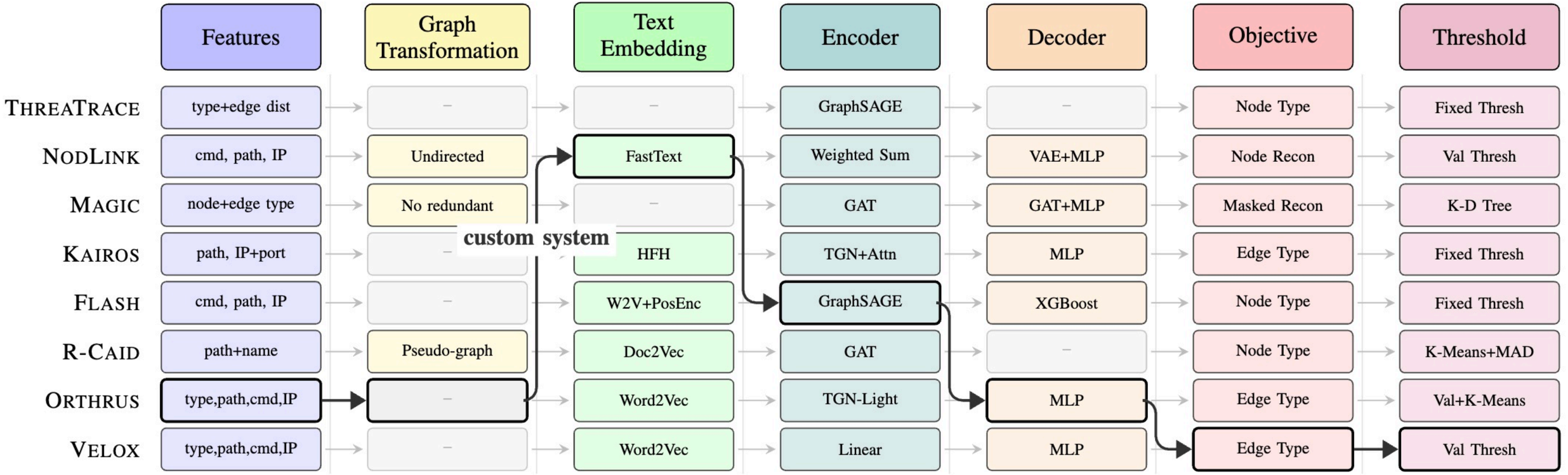
Modular pipeline with on-disk caching and automatic restarting

Running an Existing PIDS

Terminal

```
$
./run.sh SYSTEM DATASET
```

Creating a New PIDS



Creating a New PIDS

config/custom_system.yml

```
_include_yml: orthrus

featurization:
  used_method: fasttext
  fasttext:
    min_count: 2
    alpha: 0.01
    window_size: 3
    negative: 3
    num_workers: 1
training:
  encoder:
    used_methods: sage
    sage:
      activation: relu
      num_layers: 2
evaluation:
  node_evaluation:
    threshold_method: max_val_loss
    use_kmeans: False
```

Run Custom System Config

```
$
./run.sh custom_system CADETS_E3
```

Hyperparameter Tuning

tuning_custom_system.yml

```
method: grid

parameters:
  training.lr:
    values: [0.001, 0.0001]
  training.node_hid_dim:
    values: [32, 64, 128, 256]
```

Launch Hyperparameter Sweep

```
$
./run.sh custom_system CADETS_E3 \
  --tuning_mode=hyperparameters \
  --exp=my_sweep_name \
  --project=my_project
```

Run Tuned System

```
$
./run.sh custom_system CADETS_E3 --tuned
```

Ablation Study

ablation_custom_system.yml

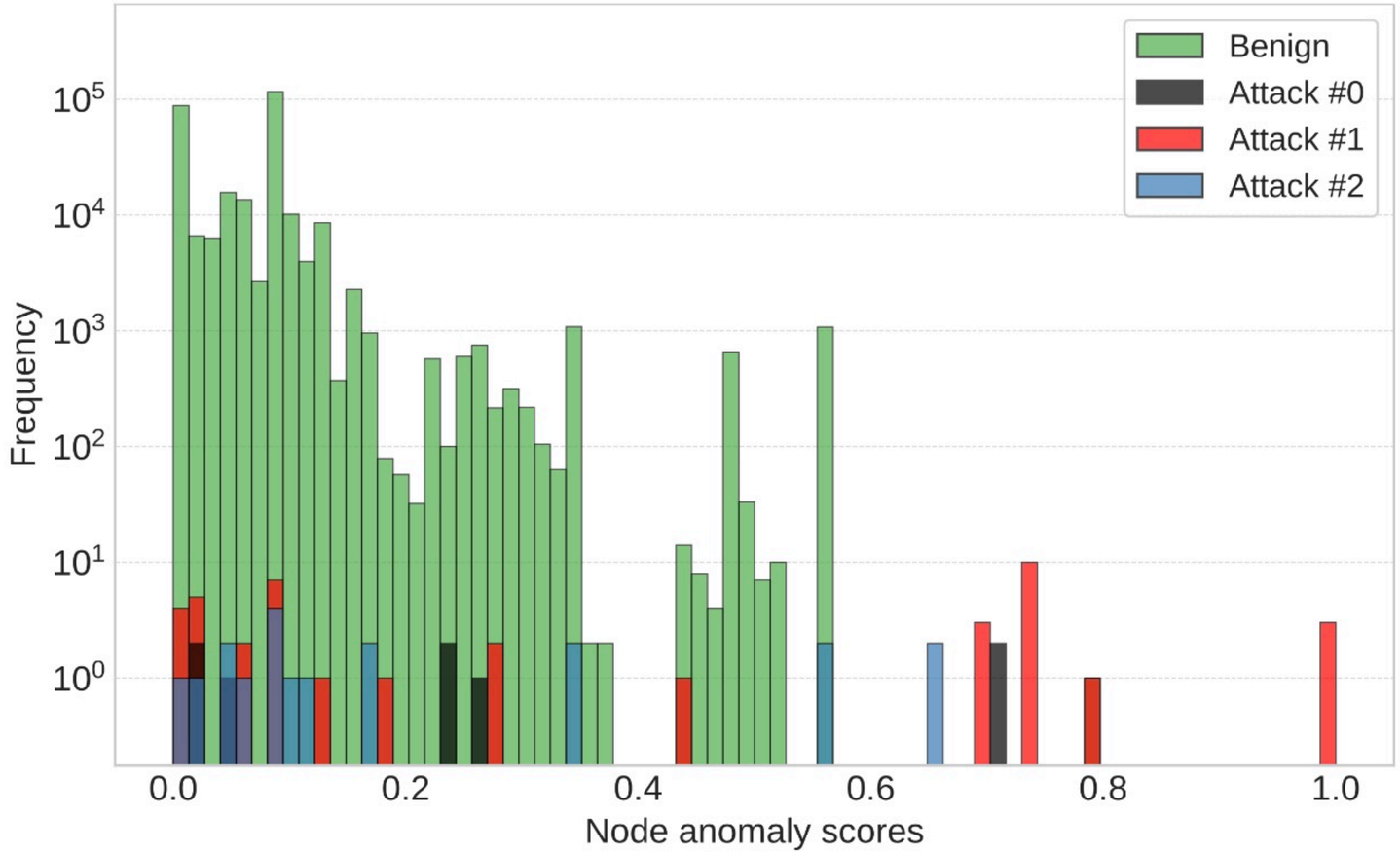
```
method: grid

parameters:
  featurization.used_method:
    values: [fasttext, word2vec]
  training.encoder.used_methods:
    values: [sage, graph_attention, none]
```

Launch Ablation Sweep

```
$
./run.sh custom_system CADETS_E3 \
  --tuning_mode=hyperparameters \
  --tuning_file_path=systems/default/
      ablation_custom_system.yml
```

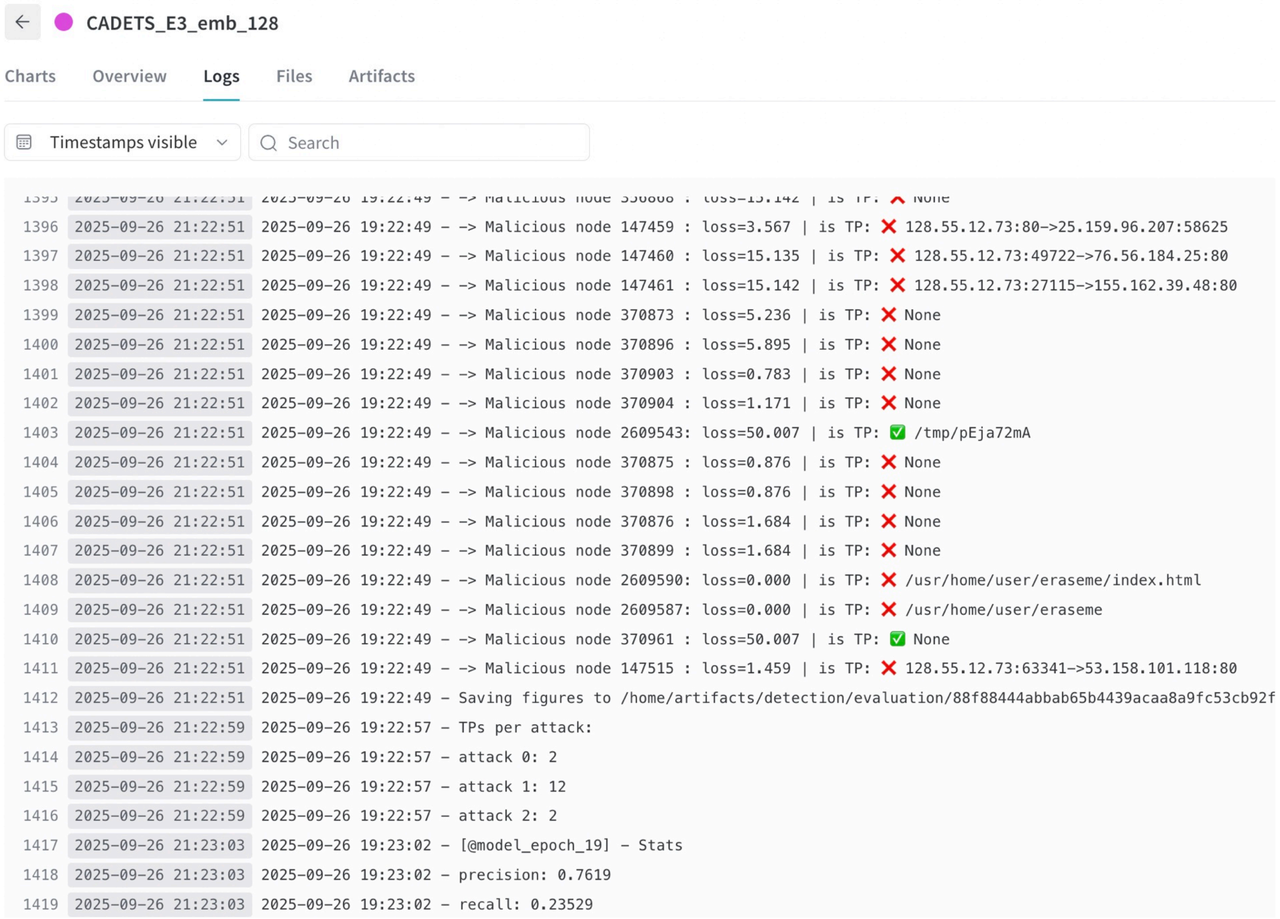
Visualising Results

Distribution of predicted node anomaly scores of a custom system on the E3-CADETS dataset



Weights & Biases

Log & model weights historization

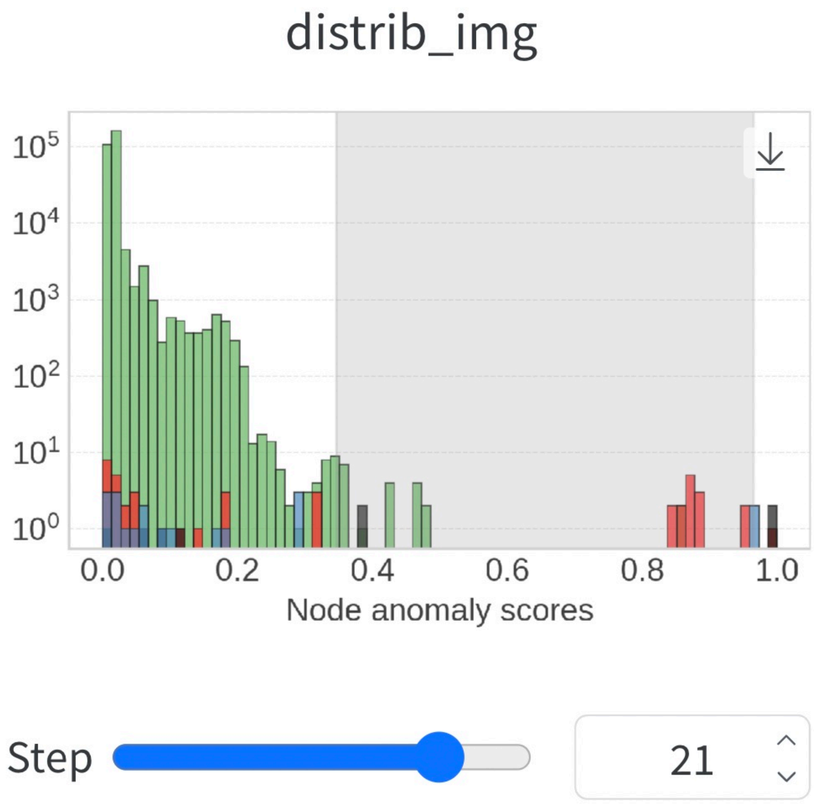


Weights & Biases

Epoch-level metric tracking

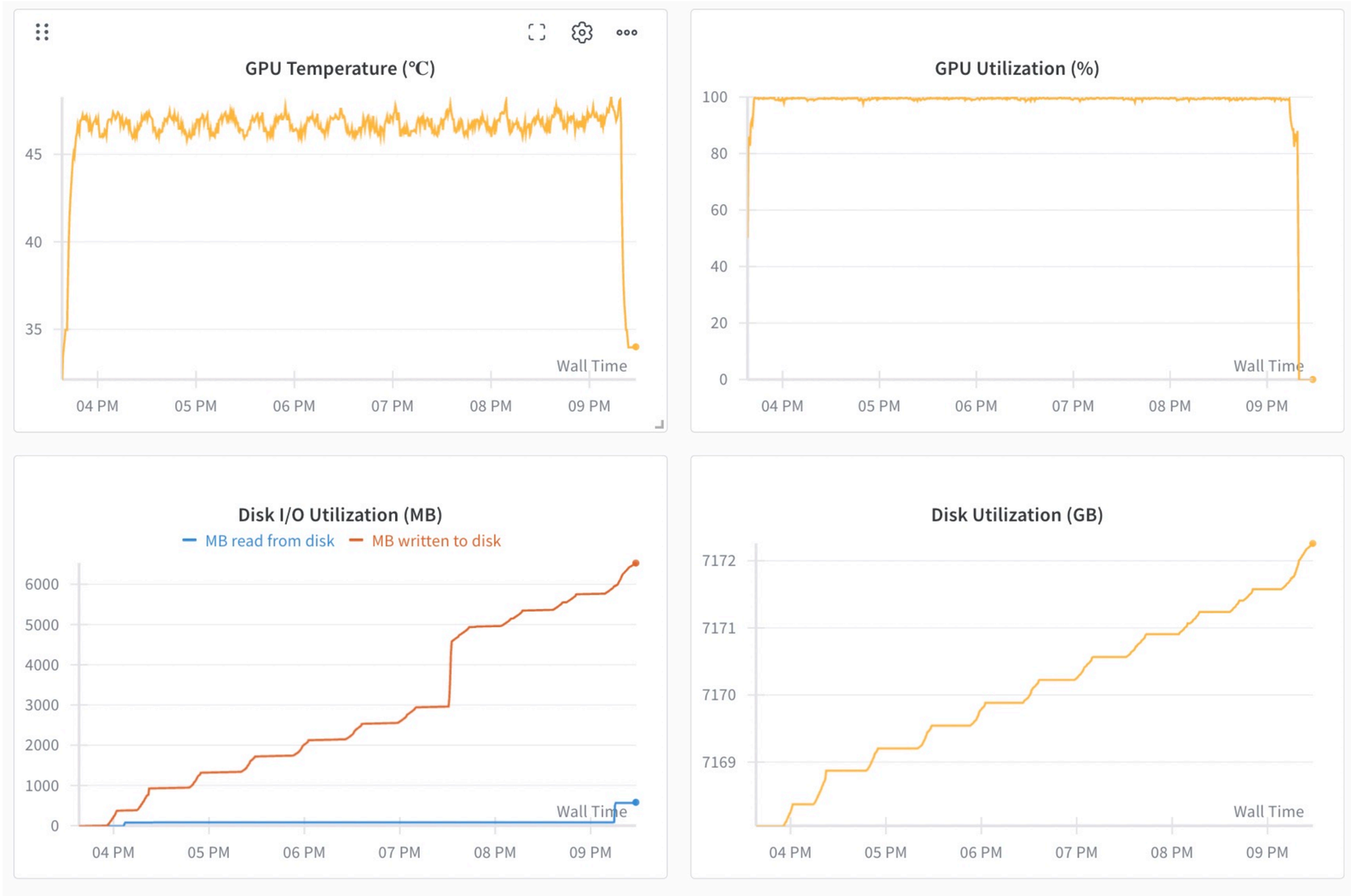
Weights & Biases

Epoch-level figures



Weights & Biases

System metrics tracking (CPU, GPU, memory, network, etc)



Want to Work with Us?

THE UNIVERSITY OF BRITISH COLUMBIA

We always look for motivated students (collaboration, funded internship or PhD in Vancouver)

If interested, drop me an e-mail: tristan.bilot@universite-paris-saclay.fr

PIDSMaker Repo:
https://github.com/ubc-provenance/PIDSMaker