

# Leo: Online ML-based Traffic Classification at Multi-Terabit Line Rate

**Syed Usman Jafri**, Sanjay Rao, Vishal Shrivastav, Mohit Tawarmalani



# Why ML-based traffic classification?

- Detecting traffic anomalies, IoT device classification and application classification

## Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics

Arunan Sivanathan<sup>1</sup>, Hassan Habibi Gharakheili<sup>2</sup>,  
Arun Vishwanath<sup>3</sup>, Senior Member

## GGFAST: Automating Generation of Flexible Network Traffic Classifiers

Julien Piet  
Corelight  
San-Francisco, USA  
UC Berkeley  
Berkeley, USA

Dubem Nwoji  
Corelight  
San-Francisco, USA

Vern Paxson  
Corelight  
San-Francisco, USA  
UC Berkeley  
Berkeley, USA

### ABSTRACT

When employing supervised machine learning to analyze network traffic, the heart of the task often lies in developing effective features for the ML to leverage. We develop GGFAST, a unified, automated framework that can build powerful classifiers for specific network traffic analysis tasks, built on interpretable features. The framework uses only packet sizes, directionality, and sequencing, facilitating

### 1 INTRODUCTION

Many network traffic analysis problems can be viewed as classification tasks: given a set of characteristics of a network flow, decide what category to assign to the flow. These categories can refer to activity at different semantic levels, such as determining the application protocol employed by the flow, specific modes-of-use within the flow (e.g. determining the authentication mechanism within an

## CICADA: Cloud-based Intelligent Classification and Active Defense Approach for IoT Security

Jeupane, <sup>3</sup>Shaynoah Bedford, <sup>4</sup>Shreyas Prabhudev,  
anli Pan, <sup>1</sup>Prasad Calyam  
<sup>3</sup>Uni. of the Virgin Islands, <sup>4</sup>Uni. of California at San Diego,  
tx, calyamp}@missouri.edu, tmzobrist1s@semo.edu,  
prabhudev@ucsd.edu, jpan22@gmu.edu

are evolving and dynamic, IoT systems may be incapable of effective defense due to the use of prevalent static defense

## Traffic Classification using Deep Learning: Being Highly Accurate is Not Enough

Kang-Hee Lee  
Sangmyung University  
lk7054@gmail.com

Seung-Hun Lee  
Sangmyung University  
mr.leesh90@gmail.com

Hyun-Chul Kim\*  
Sangmyung University  
hyunchulk@gmail.com

## Replication: Contrastive Learning and Data Augmentation in Traffic Classification Using a Flowpic Input Representation

Alessandro Finamore  
Huawei Technologies SASU, France  
alessandro.finamore@huawei.com

Chao Wang  
Huawei Technologies SASU, France  
wang.chao3@huawei.com

Jonatan Krolikowski  
Huawei Technologies SASU, France  
jonatan.krolikowski@huawei.com

Jose M. Navarro  
Huawei Technologies SASU, France  
jose.manuel.navarro@huawei.com

Fuxing Chen  
Huawei Technologies SASU, France  
chenfuxing@huawei.com

Dario Rossi  
Huawei Technologies SASU, France  
dario.rossi@huawei.com

### ABSTRACT

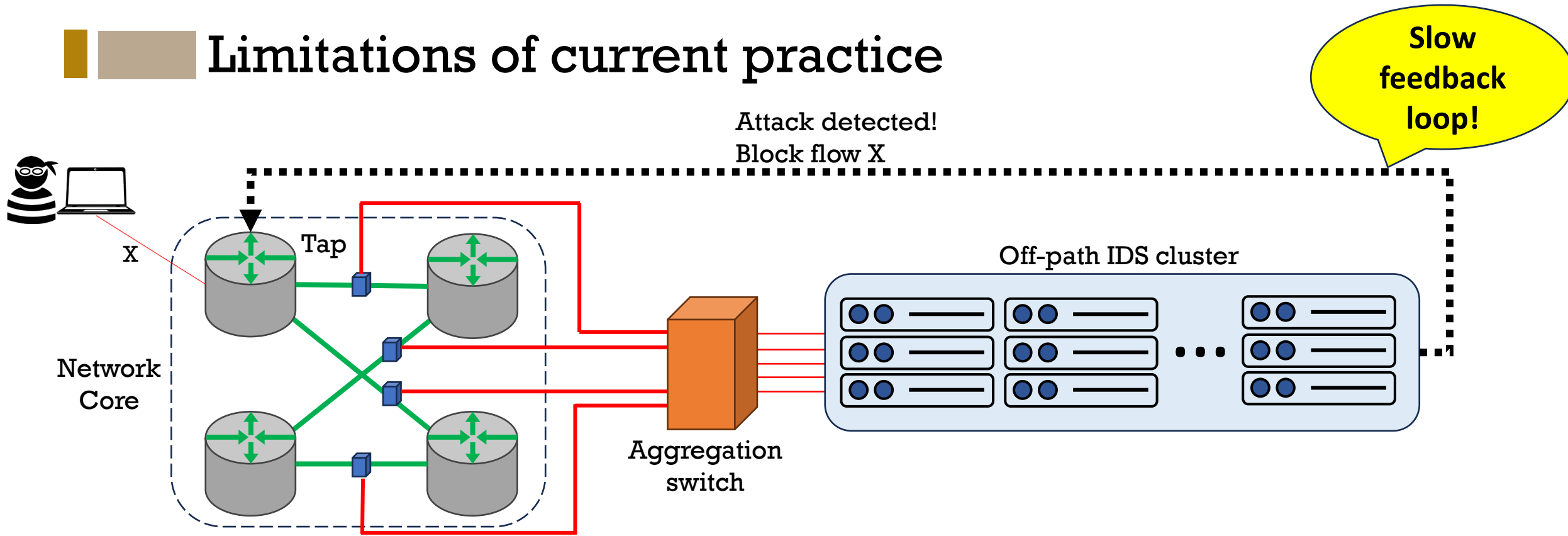
Over the last years we witnessed a renewed interest toward Traffic Classification (TC) captivated by the rise of Deep Learning (DL). Yet, the vast majority of TC literature lacks code artifacts, performance assessments across datasets and reference comparisons against Machine Learning (ML) methods. Among those works, a recent study

### ACM Reference Format:

Alessandro Finamore, Chao Wang, Jonatan Krolikowski, Jose M. Navarro, Fuxing Chen, and Dario Rossi. 2023. Replication: Contrastive Learning and Data Augmentation in Traffic Classification Using a Flowpic Input Representation. In *Proceedings of the 2023 ACM Internet Measurement Conference (IMC '23)*, October 24–26, 2023, Montreal, QC, Canada. ACM, New York, NY, USA, 16 pages. <https://doi.org/10.1145/3618257.3624820>

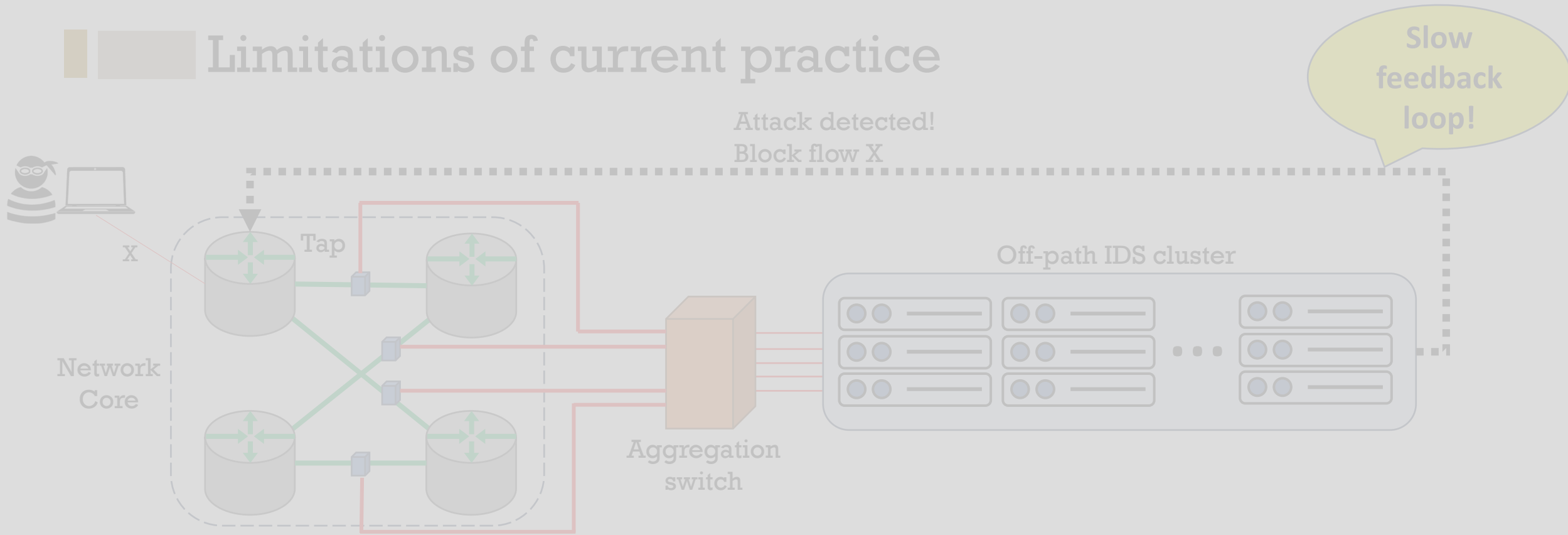
- Capture complex patterns w/o peeking into payload
  - Learn behavioral patterns from network flow statistics
- ➔ Applicable to encrypted traffic

# Limitations of current practice



- Traffic sent off-path for analysis
- Asynchronous → slow reaction time
- Challenging to analyze traffic at line rate

# Limitations of current practice



- Traffic sent off-path for analysis
- Asynchronous → slow reaction time
- Challenging to analyze traffic at line rate

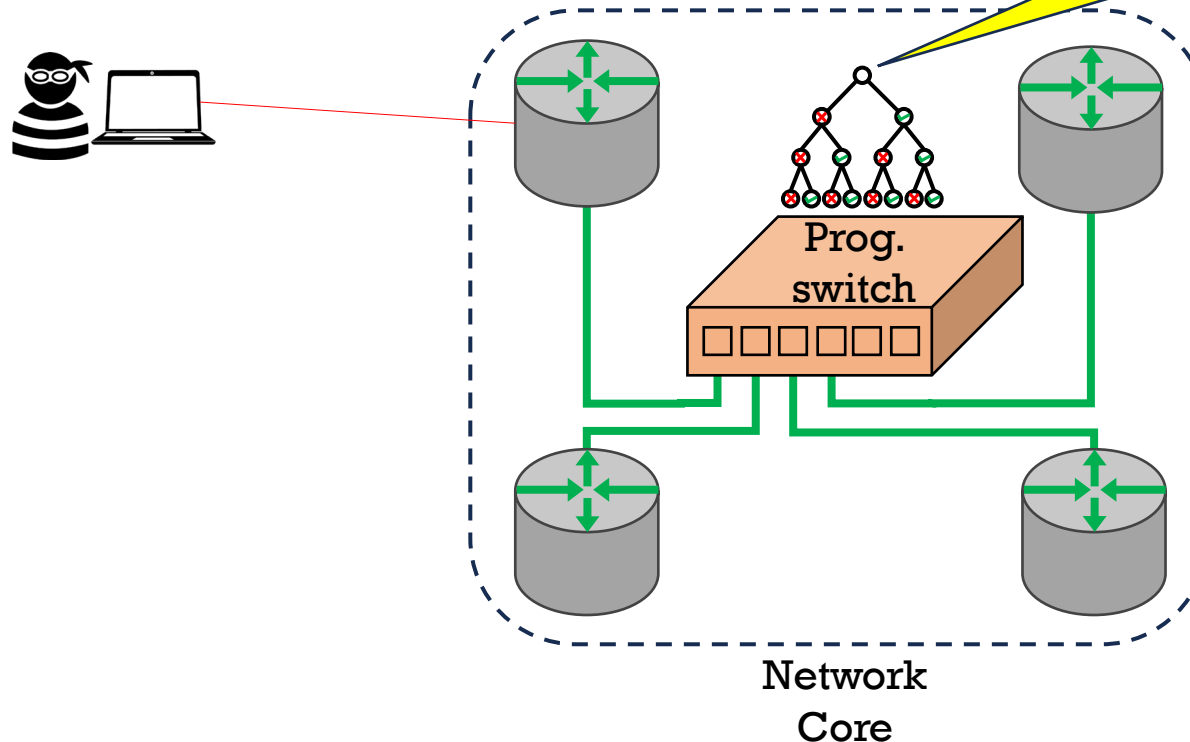
**Can we do classification synchronously at line rate?**

# Opportunity: In-network compute

## Programmable switches offer new opportunities

- Ability to define custom packet processing logic
- **Multi-terabit** execution of user programs

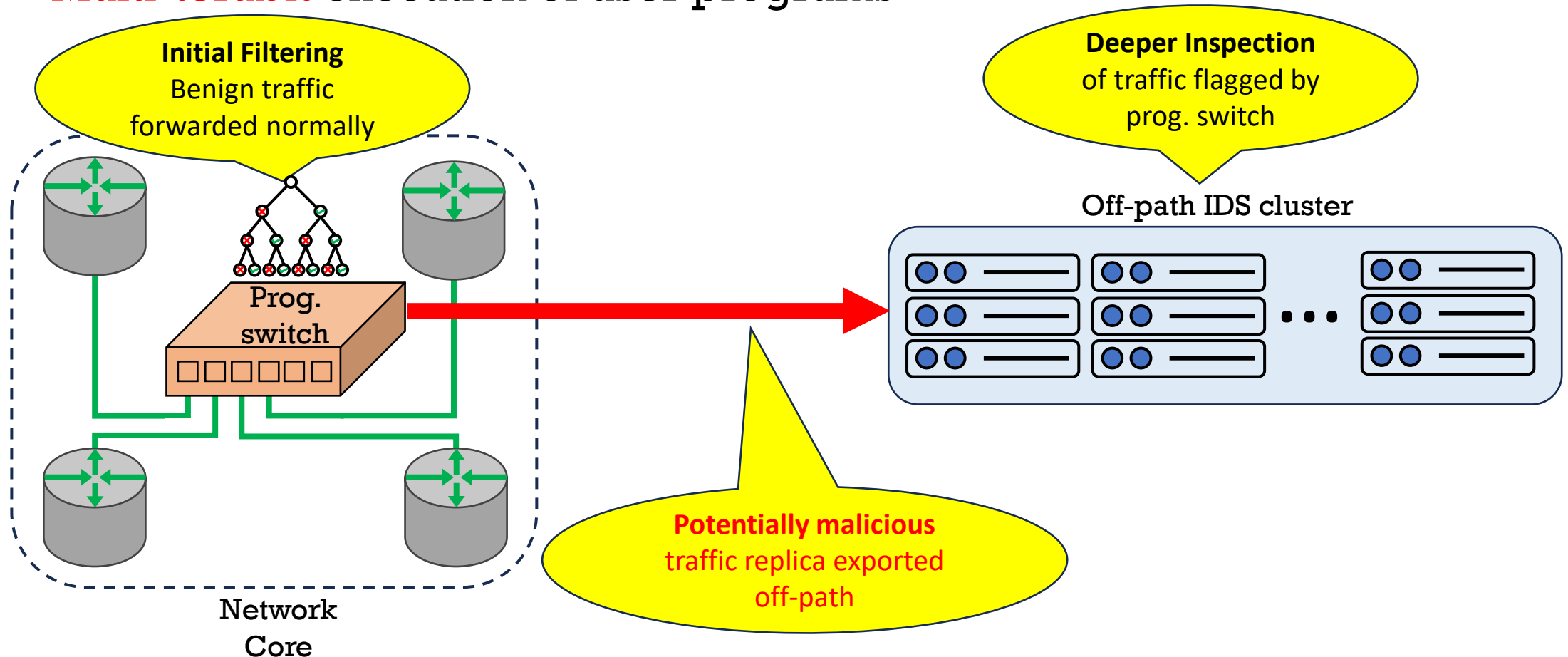
**Goal:** Classify packets in real-time!



# Opportunity: In-network compute

## Programmable switches offer new opportunities

- Ability to define custom packet processing logic
- **Multi-terabit** execution of user programs



# ■ ■ ■ Challenges

## **Run-time programmable**

- Allow model updates with no switch downtime!

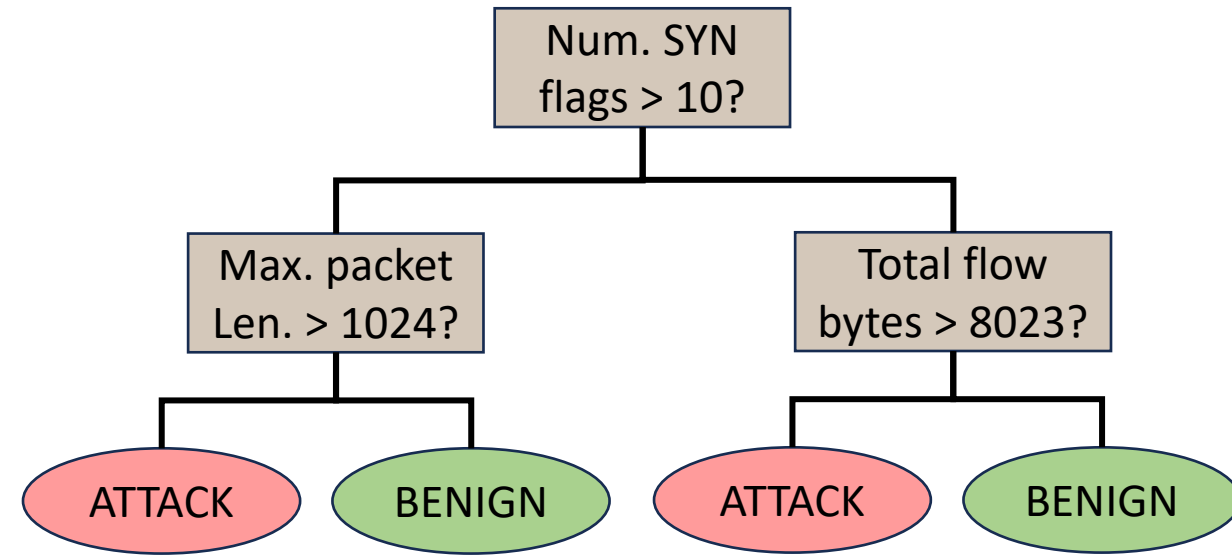
## **Resource-efficient**

- Switch HW resources are reserved at compile time
- **Worst-case** bounds on resources are important!

## **Programmable switches have limited expressivity**

# Our focus: Decision Trees

- Programmable switches sufficiently expressive for decision tree operations
- Easily interpretable compared to “black box” models
- Competitive accuracy <sup>[1, 2]</sup>



[1] Nigel Williams, Sebastian Zander, and Grenville Armitage, “A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification,” 2006 SIGCOMM Computer Communication Review 36, 5. (SIGCOMM CCR)

[2] M. Shafiq, X. Yu, A. A. Laghari, L. Yao, N. K. Karn and F. Abdessamia, "Network Traffic Classification techniques and comparative analysis using Machine Learning algorithms," 2016 2nd IEEE International Conference on Computer and Communications (ICCC)



# Leo contributions

## Support a **class** of decision trees in a runtime programmable fashion

- Can support **any** tree within a (depth, leaves, features) class
  - Emphasis on **supporting a class** of trees, **not a specific tree**
- Update the model with **no switch downtime**

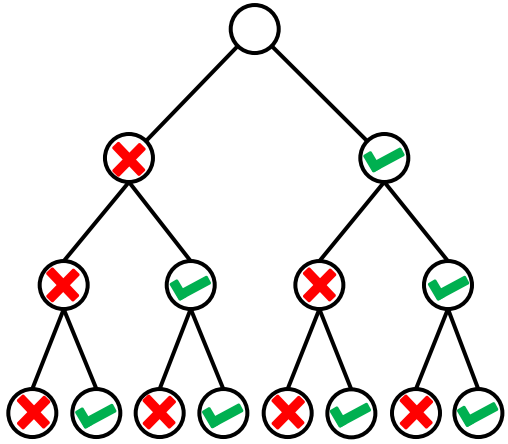
## Scalable

- Resource-efficient (memory, ALUs and pipeline stages)
- Scales to large depths (**2x** of prior work)
- 1 million flows (using 56 stateful feature bits per flow)

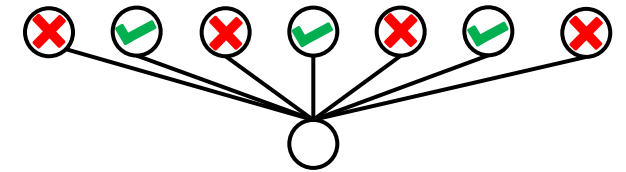
## High accuracy

- Comparable F1 score to control plane (SRAM : 93%, TCAM : 98%)

# ■ ■ ■ Prior attempts to support decision trees in data plane



**Follow natural tree dependency**



**Break tree dependency**

Memory ↓  
Stages ↑

pForest [1]  
SwitchTree [2]  
Infocom [3]

Bottleneck: switch stages

Memory ↑  
Stages ↓

ILsy [4]

**Question:**  
How well does it perform?

## Our analysis of IIsy

- To support all trees with depth  $\leq D$ , a subset of  $N$  features taking values  $[0..K]$  in a run-time programmable manner:
  - **Proposition 1:** The total SRAM to provision with IIsy is **exponential** in number of features
  - **Proposition 2:** There exist a family of trees with **polynomial # of leaves** w.r.t  $K$  but requires at least an **exponential # of TCAM rules**
- See paper for full analysis!

# Leo – Overview

## Workflow



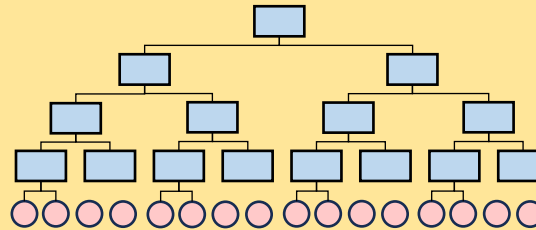
**User**

I want to support all  
trees in the class:  
(D=4, L=16, F=5)

1. User specifies a tree class: (Depth, Leaves, Feature set)

**Leo Compiler**

2. Chooses a representative tree structure



3. Provisions resources for the representative tree in the switch data plane
4. At runtime, switch control plane can configure any tree in the (D, L, F) class into the data plane

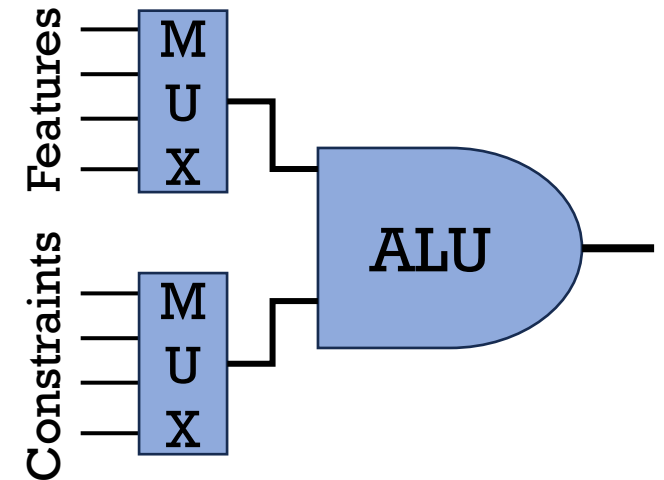
# Leo – Programmable Tree Node

**Challenge:** To implement different trees in the (D, L, F) class:  
Node **features** and **constraints** need to be runtime programmable

Leo provides a Multiplexed ALU abstraction:

Now to build a runtime programmable tree:

- Reserve a Mux ALU for every node in tree
- **Prohibitively expensive ALU requirement!**



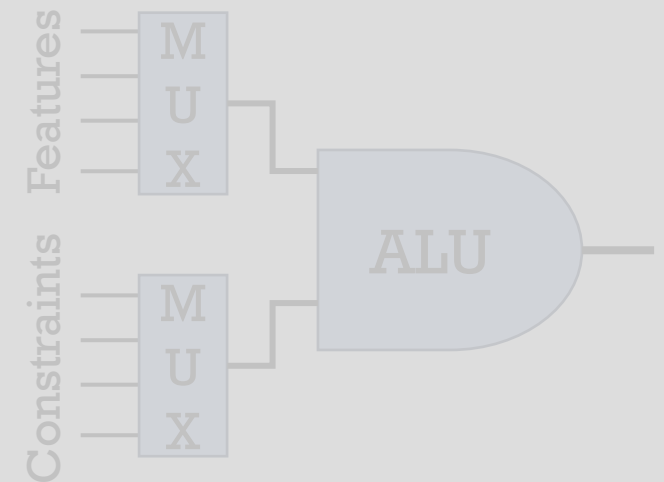
## Leo – Programmable Tree Node

**Challenge:** To implement different trees in the (D, L, F) class:  
Node **features** and **constraints** need to be runtime programmable

Leo provides a Multiplexed ALU abstraction:

Now to build a runtime programmable tree:

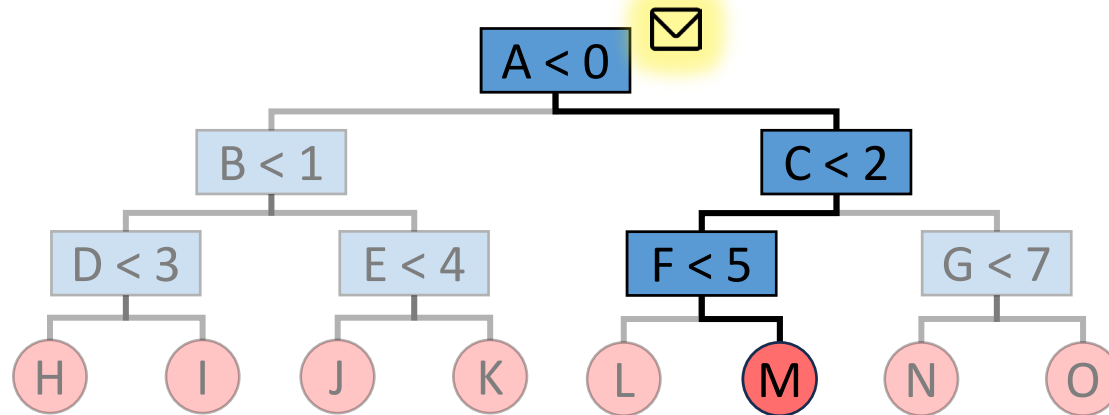
- Reserve a Mux ALU for every node in tree
- Prohibitively expensive ALU requirement!



**Question: How to make resource requirements tractable?**

# Leo – Node Multiplexing

**Key Observation:** At runtime, only 1 node per level is accessed



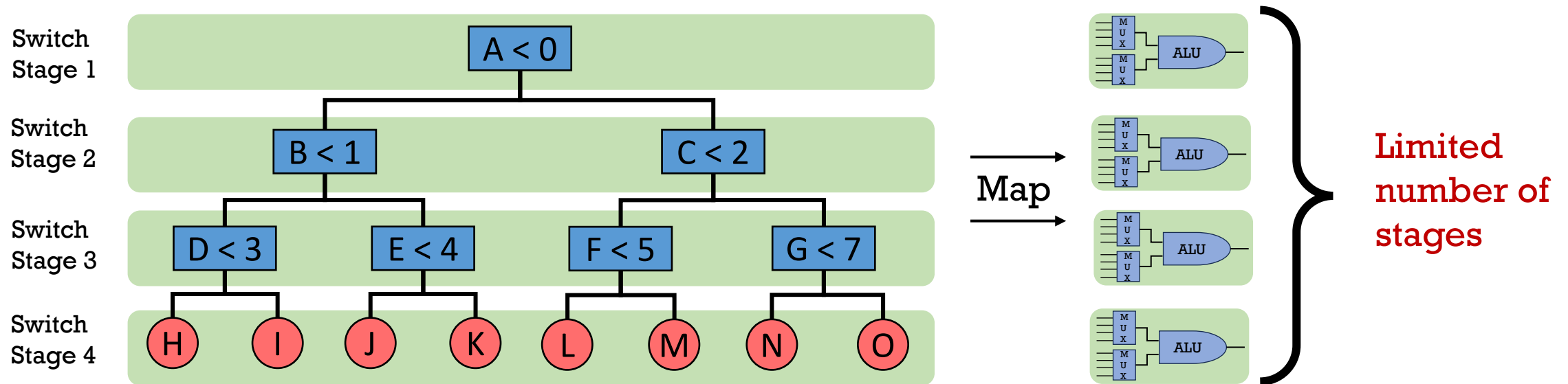
- Allocate resources for only **one node per level**
- At runtime, **multiplex** the feature comparisons at each node

Results in resource (ALU) efficiency!

# Node Multiplexing – Limitation

→ Depth of decision tree limited by # of switch stages

- A switch with D match-action stages can support a decision tree of depth at most D!

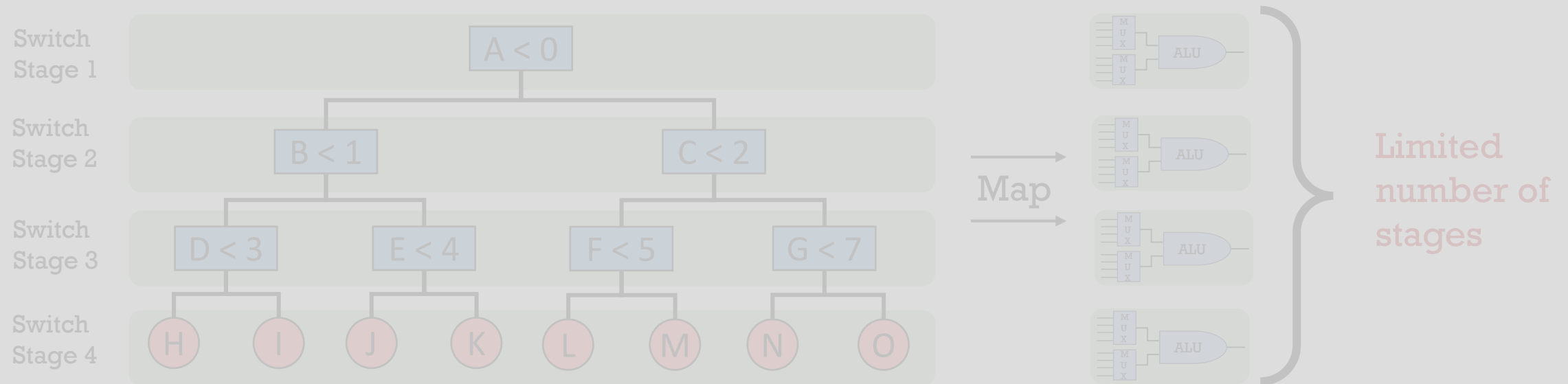




## Node Multiplexing – Limitation

→ Depth of decision tree limited by # of switch stages

- A switch with D match-action stages can support a decision tree of depth at most D!



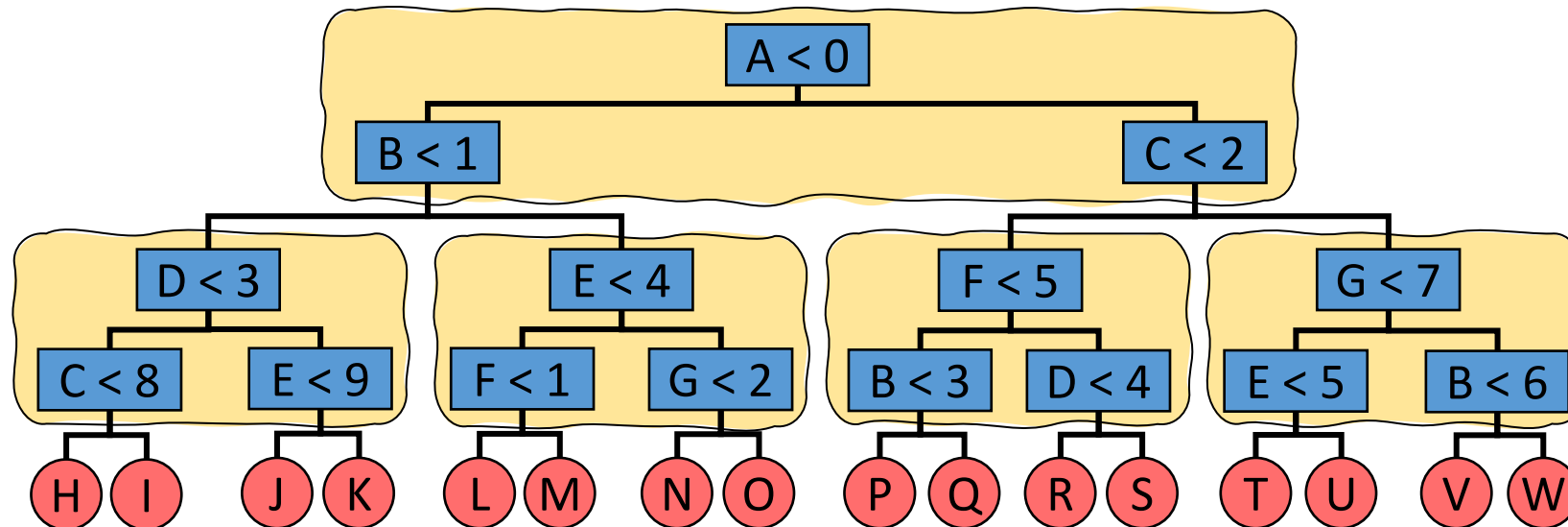
**Question: How to scale tree depth?**

# Leo – Subtree Flattening & Multiplexing

→ **Key insight:** trees have a common substructure → Subtrees

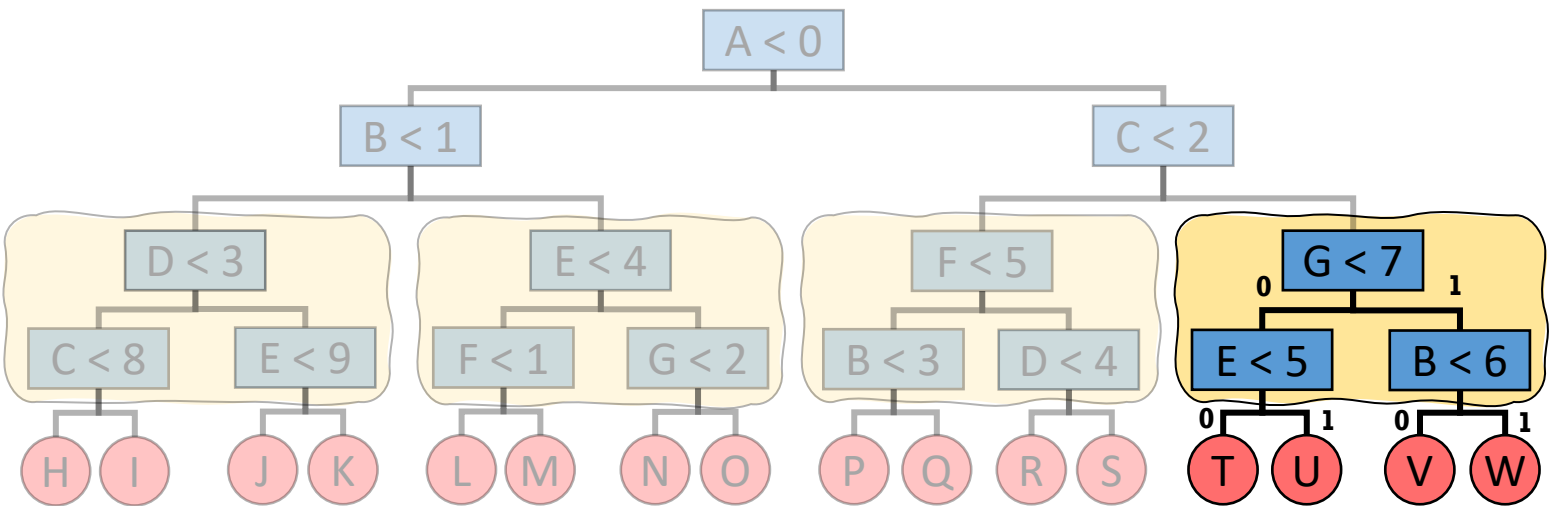
- **Flatten** the subtree
- **Multiplex** between subtrees

## 1. Identify subtrees

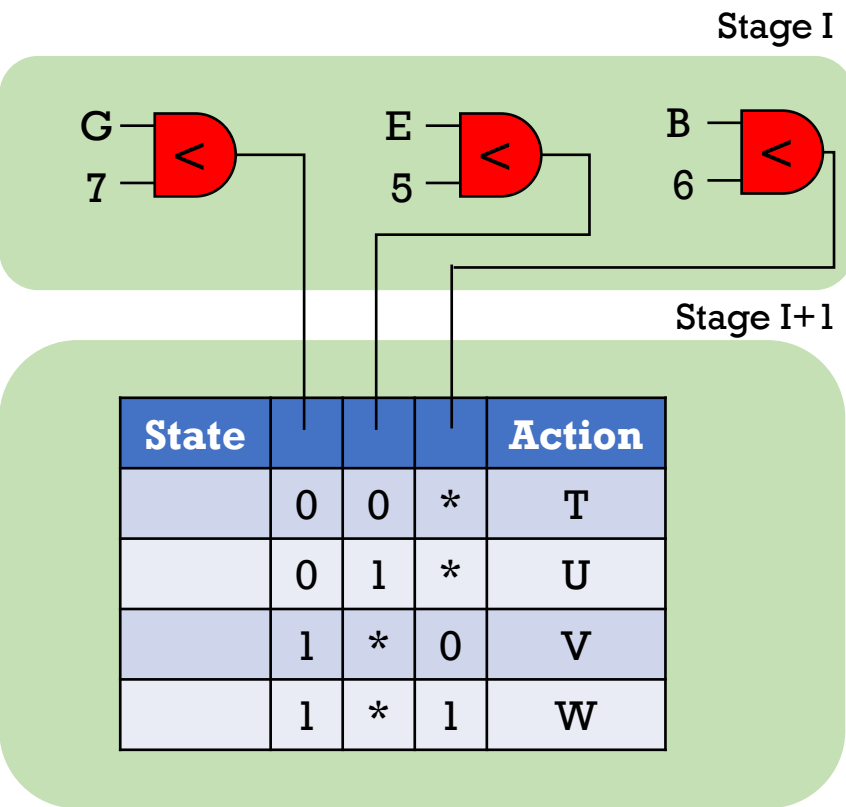


# Subtree Flattening & Multiplexing

## 2. Flatten subtrees

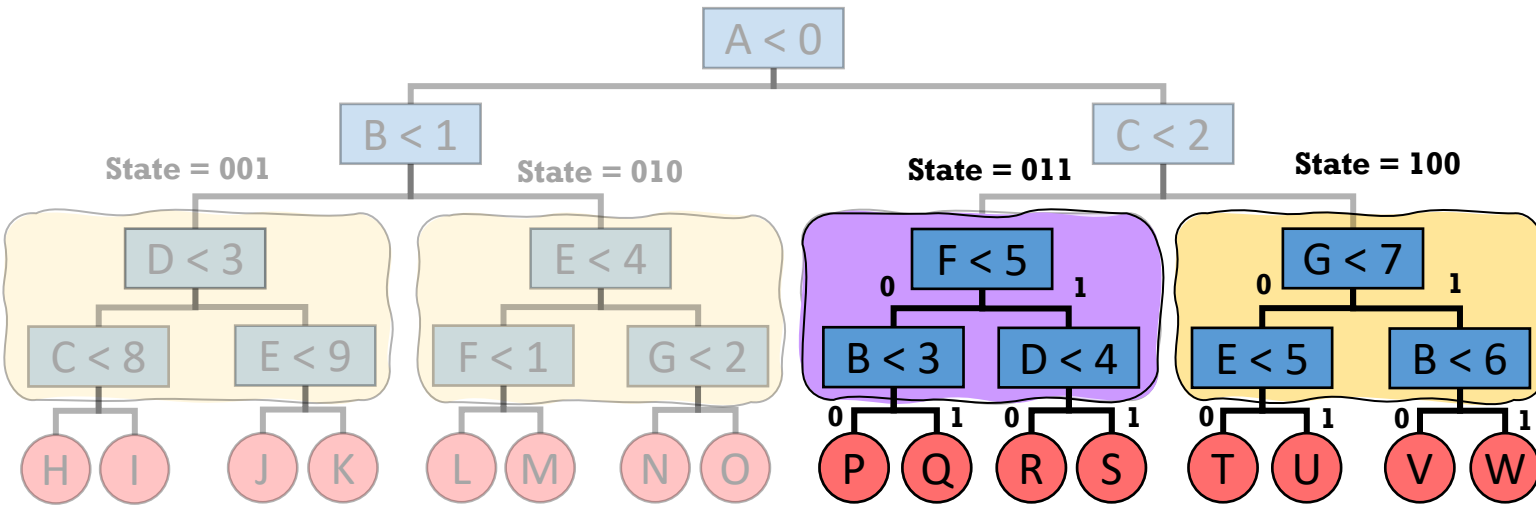


A decision tree may be represented as binary table



# Subtree Flattening & Multiplexing

## 3. Multiplex subtrees

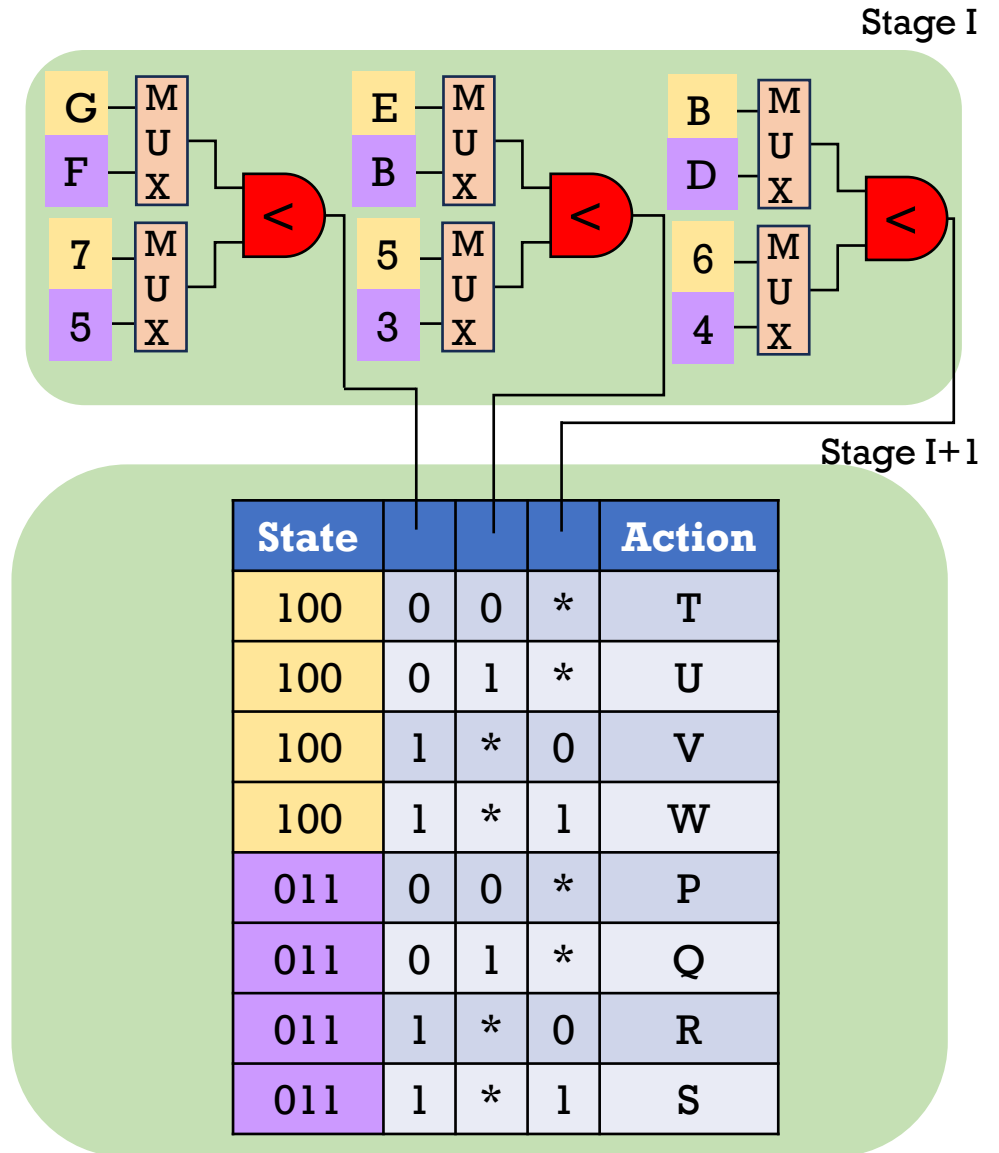


**Need half as many stages!** (For subtree size = 3)

**In general:**

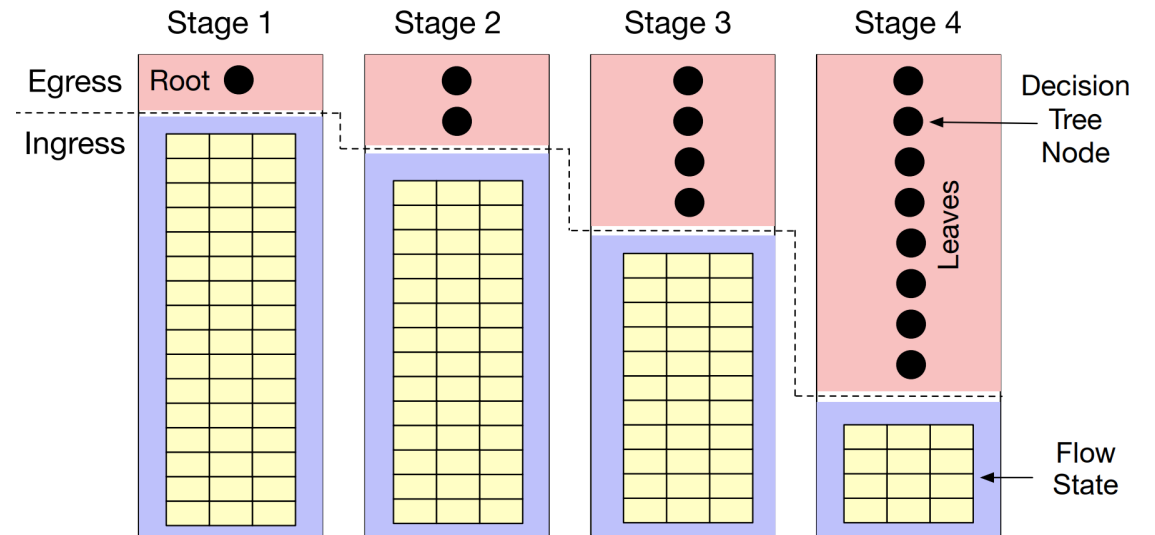
Reduce # of stages by factor of  $\lceil \log K \rceil$

$K$  = subtree size



# Leo – Implementation Optimizations

- **Handling stateful features while making efficient use of pipeline stages**



- **Handling transient states correctly during tree updates**
- **Optimization: TCAM stage reduction**
- [See paper for details](#)

## Leo – Worst-case bounds

Leo has **acceptable** upper bound on the **resource requirement**:

- For subtrees with  $k = 3$  and  $I$  internal nodes  
**SRAM entries**  $= 8I + 3$
- Itsy number of entries is **exponential** in number of features  $N$ :  
**SRAM entries**  $= \left(\frac{I}{N} + 1\right)^N$

See paper for more analysis (TCAM entries, general subtrees, ...)

# Evaluation: setup

## **Compare Leo with:**

- IIsy, pForest and SwitchTree

## **Methodology:**

- Deploy Leo and related work on Intel Tofino switch to find supported tree classes
- Train decision trees to find highest accuracy tree in supported class

## **Two intrusion detection datasets:**

- UNSW-NB15 – as a binary classification problem
- CICIDS-2017 – as a multi-class classification

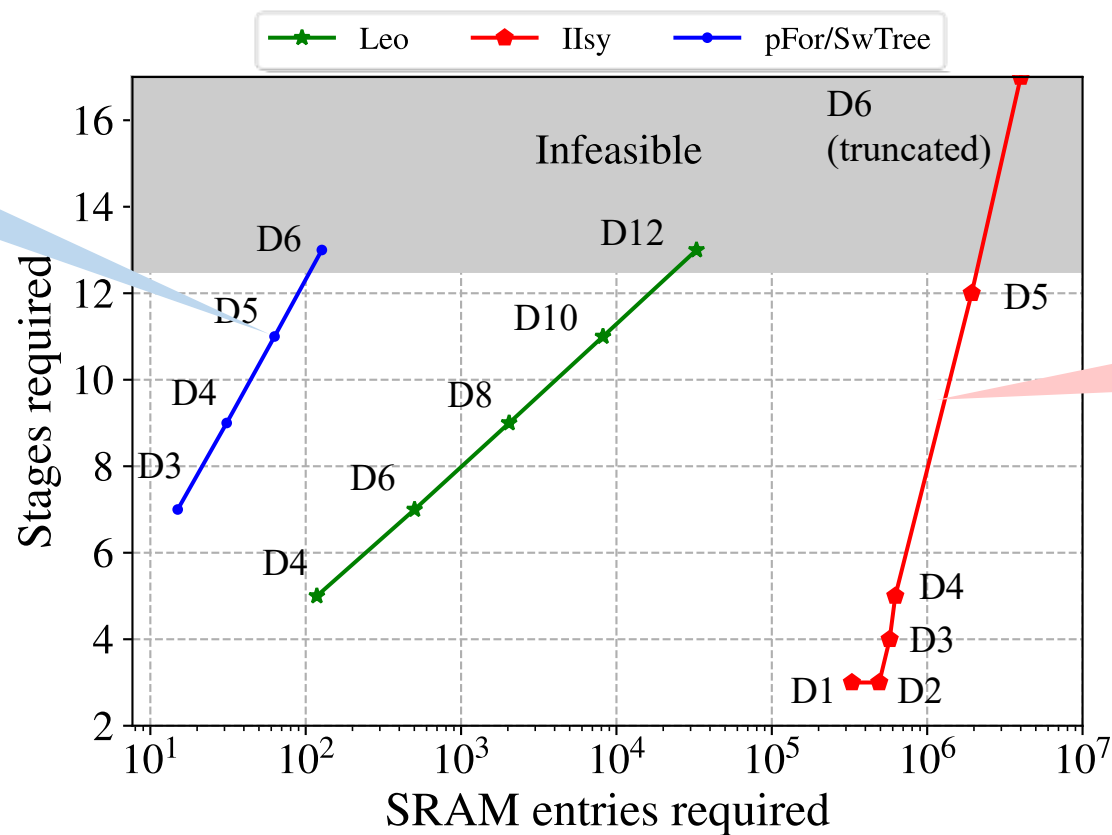
## **Metrics:**

- Evaluate on both SRAM and TCAM memory
- **(i)** Number of table entries, **(ii)** Number of switch stages, **(iii)** Mean F1 score
- **(iv)** Num. flows supported and **(iv)** Scaling depth by introducing leaf limits

# Evaluation: resource utilization

SRAM utilization of **complete** tree classes ( $D, D^2, |F|=10$ )

pForest/SwitchTree:  
Limited tree depth

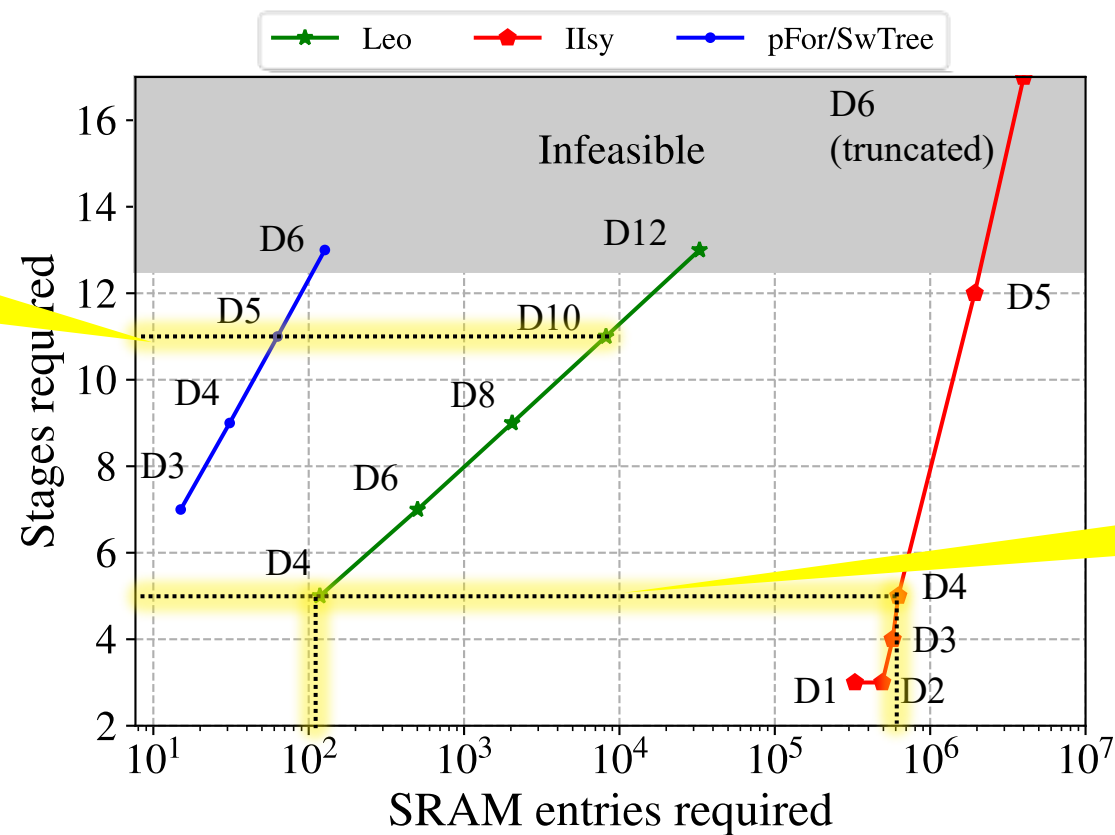


Exponential  
increase in  
resource



# Evaluation: resource utilization

SRAM utilization of **complete** tree classes ( $D, D^2, |F|=10$ )

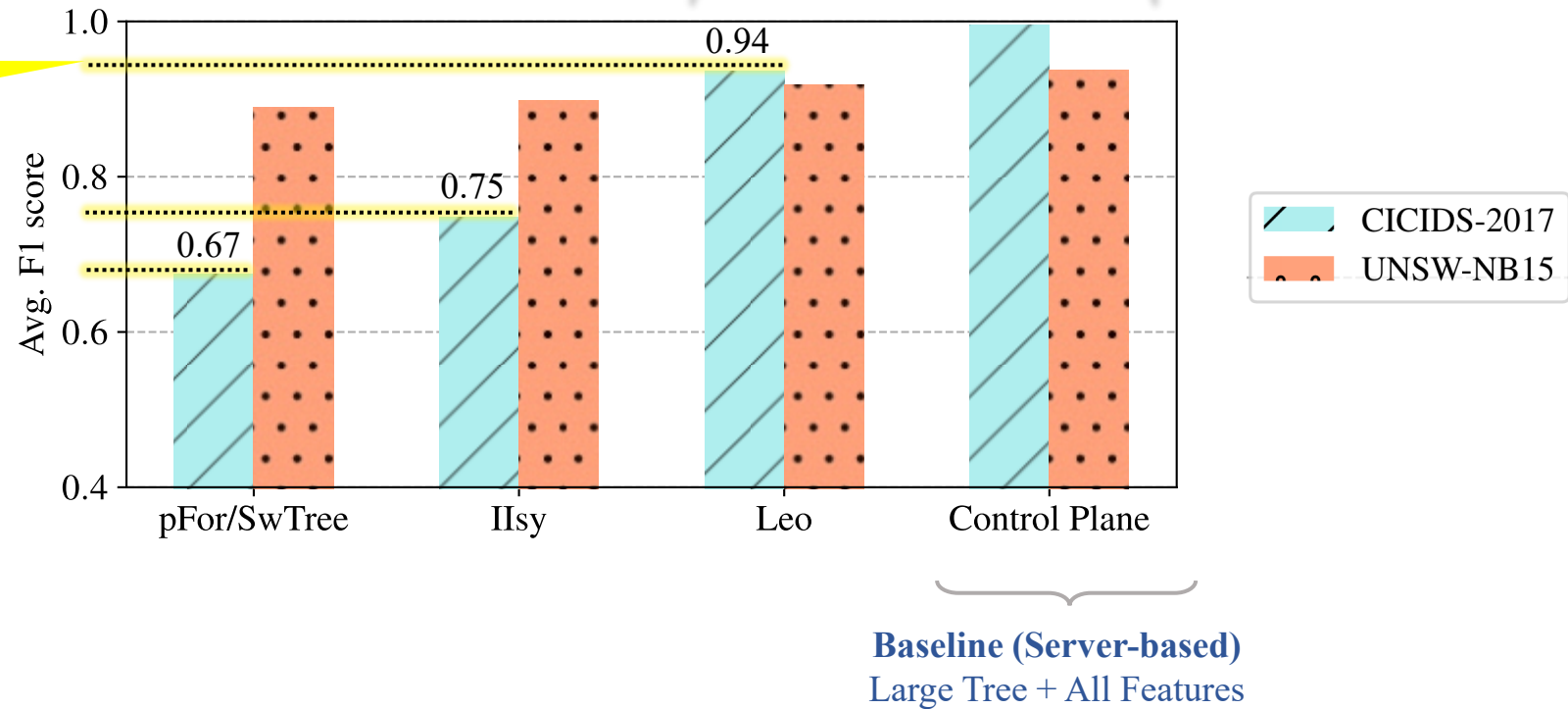


Leo supports trees with 2x larger depth (using same # stages)

Leo: order of magnitude fewer entries

# Evaluation: accuracy

Leo: accuracy much higher than prior work



## More evaluation in the paper:

- With TCAM, Leo can support:
  - Complete trees of depth 13
  - **Depth 22** with 1024 leaves
- Impact of per-flow state on number of flows
- TCAM classification accuracy results

# Conclusion

**Support a **class** of decision trees in a runtime programmable fashion**

- Can support **any** tree within a (depth, leaves, features) class

**Scalable**

- To large depths (2x of prior work)
- 1 million flows (using 56 stateful feature bits per flow)

**High accuracy**

- Comparable F1 score to control plane (SRAM : 93%, TCAM : 98%)

We released Leo source code  
<https://github.com/Purdue-ISL/Leo>