

Fast, Flexible, and Intelligent AI-Native Networks

Muhammad Shahbaz

Assistant Professor



A network provides ...

Connectivity

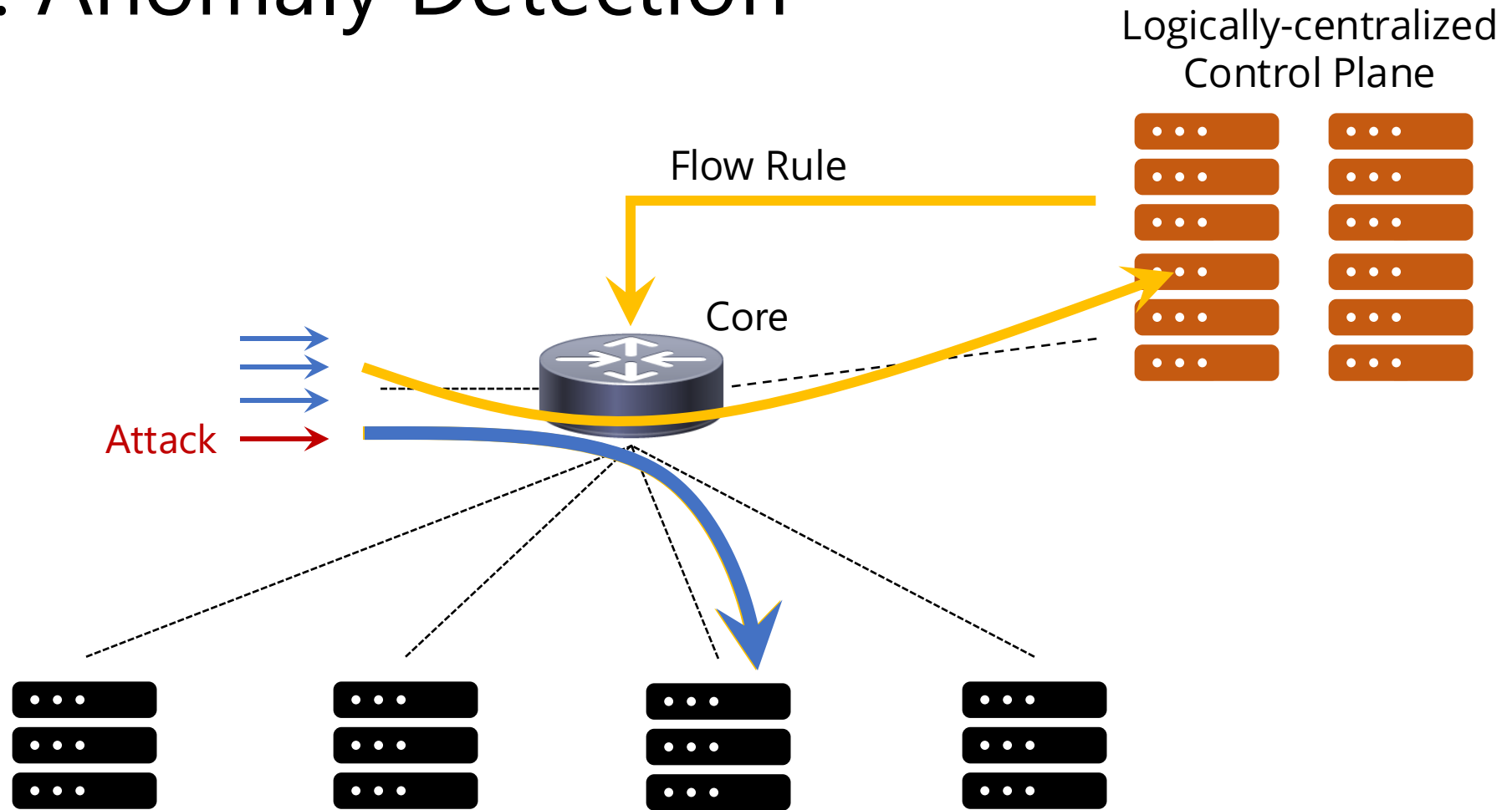
Deterministic, and impacts functionality

**Security
Performance**

Probabilistic, doesn't impact functionality

Machine Learning

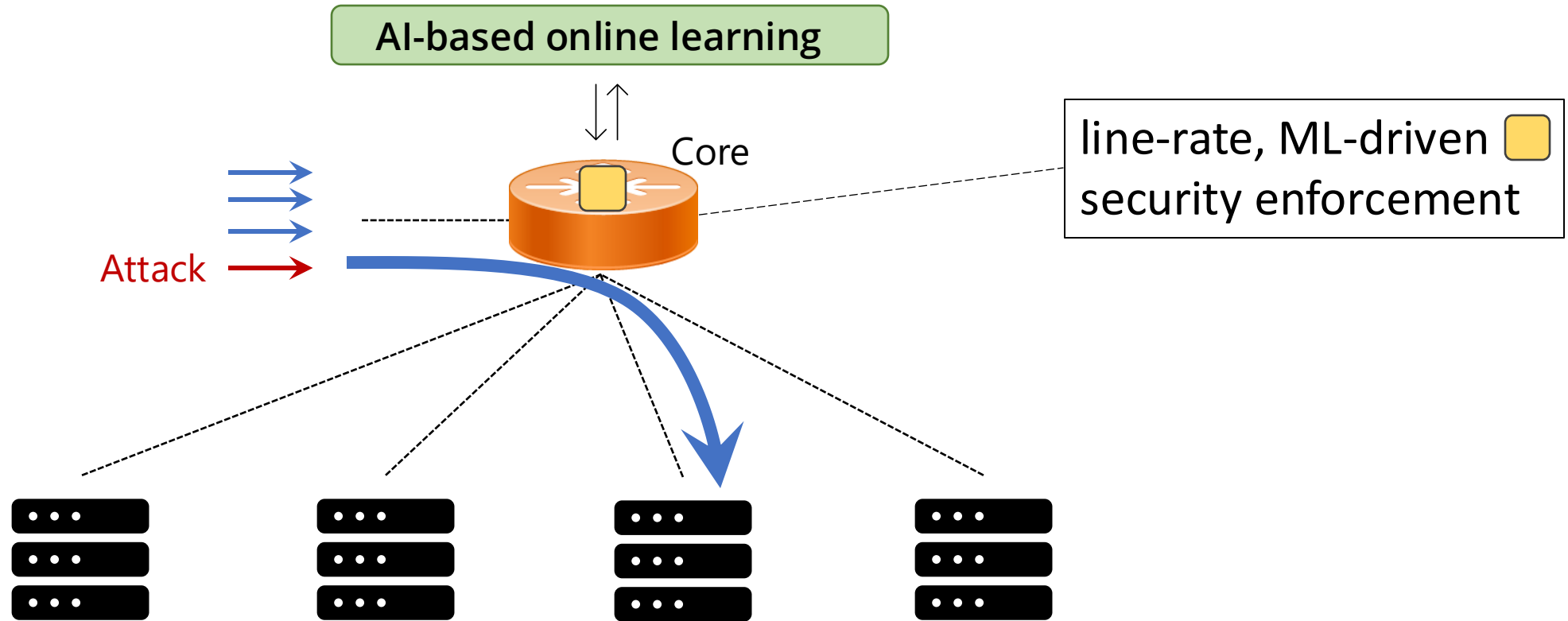
Security: Anomaly Detection



Learn and Enforce



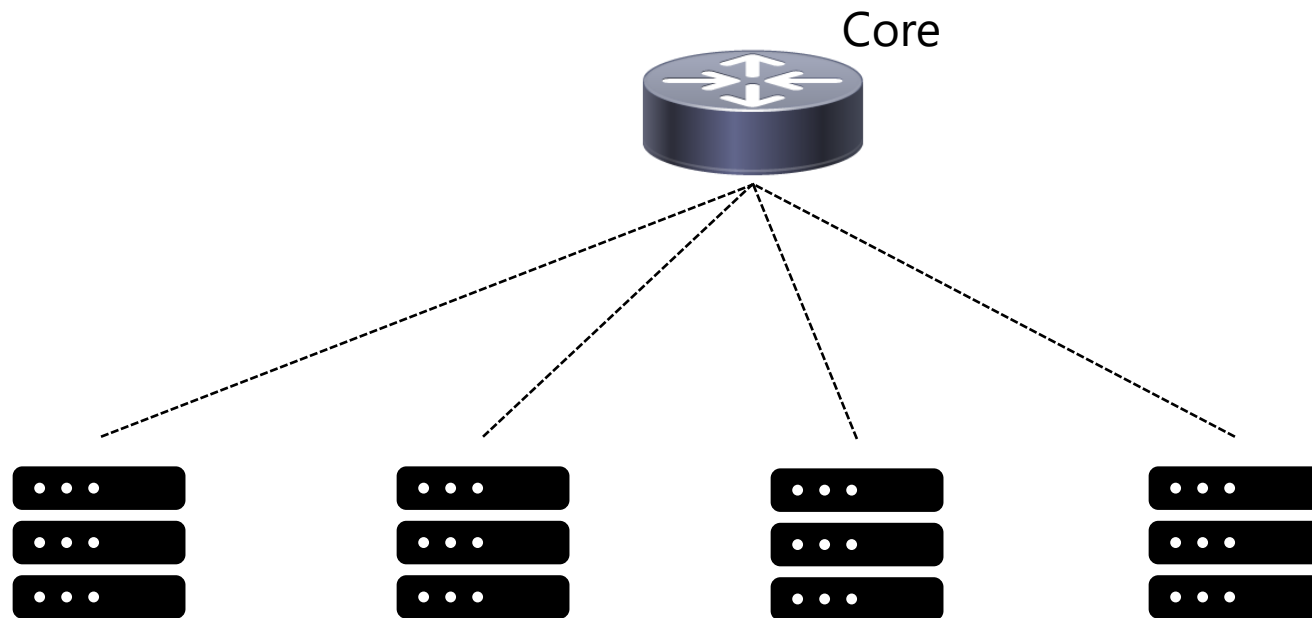
Security: Anomaly Detection



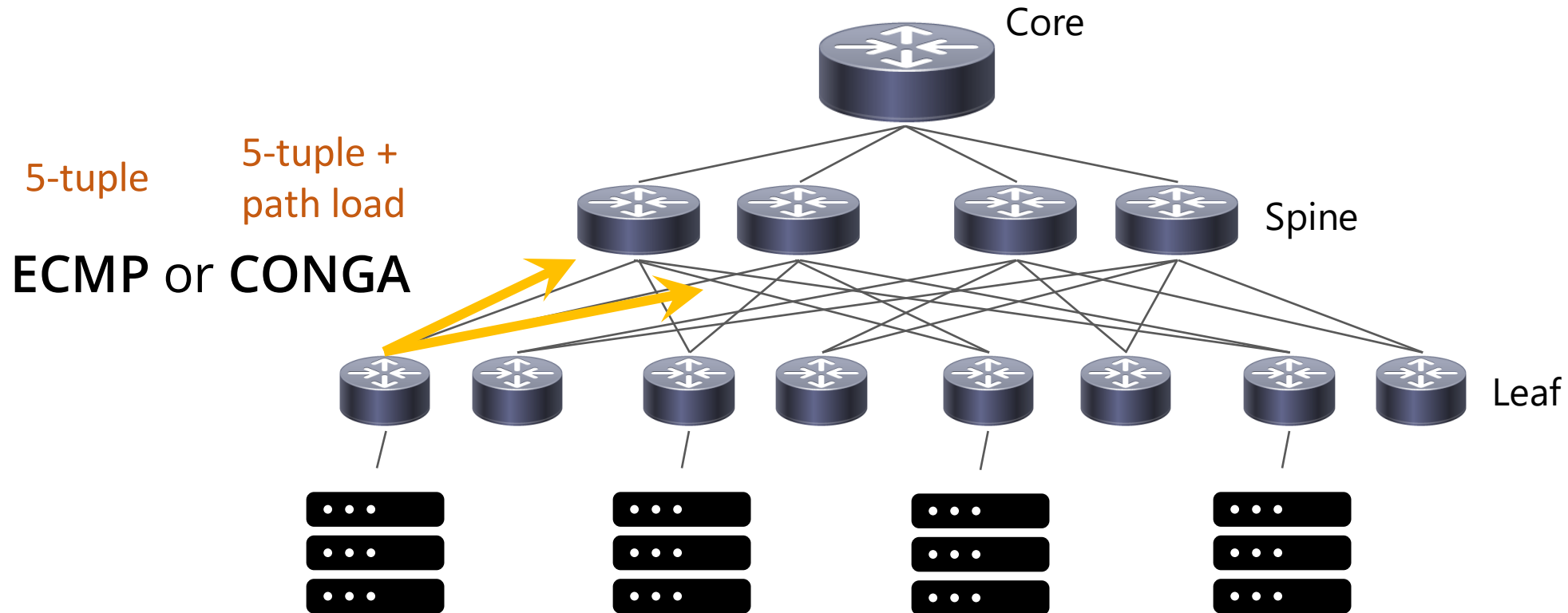
Enforce while Learning



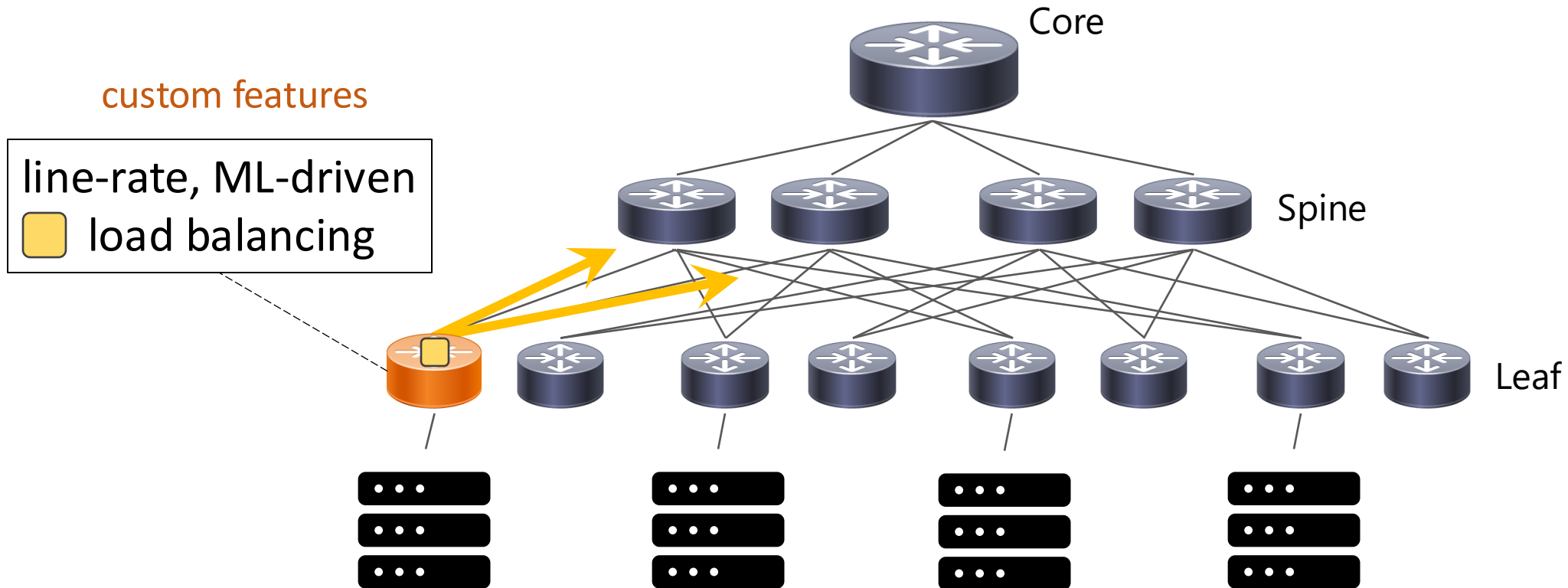
Performance: Load Balancing



Performance: Load Balancing



Performance: Load Balancing



History of Systems vs. Networking

Fixed-function
Processors



1930s

Programmable
Processors



1970s

Domain-Specific
Processors



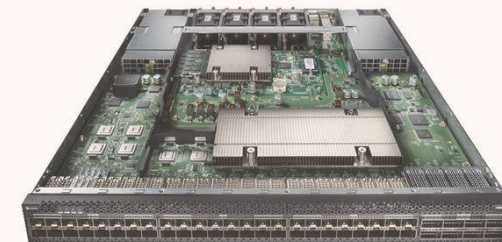
2010s

Networking is
trailing **40 years**
behind of Systems

Fixed-function
Switches



Programmable
Switches



Programmable Processors vs. Switches

ML

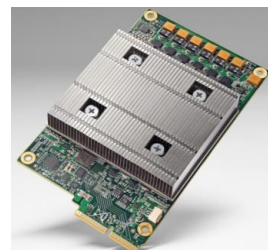
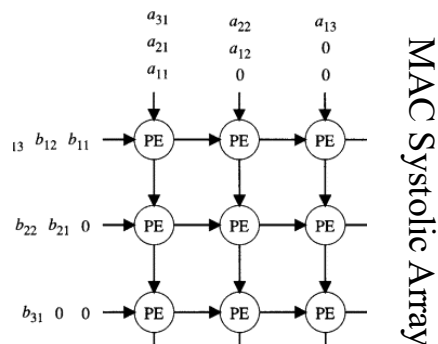
~~for $i = 1$ to m
for $j = 1$ to n
...~~

OS



CPU

ML



TPU

ML

~~A collection of
Match Function
(MLF) Rules~~

NOS



Prog. Switch
(Tofino)

ML

???

???

???

Fast and Intelligent Networking

- We want to bridge the gap between speed and intelligence!

"Inference inside switches (NICs)"

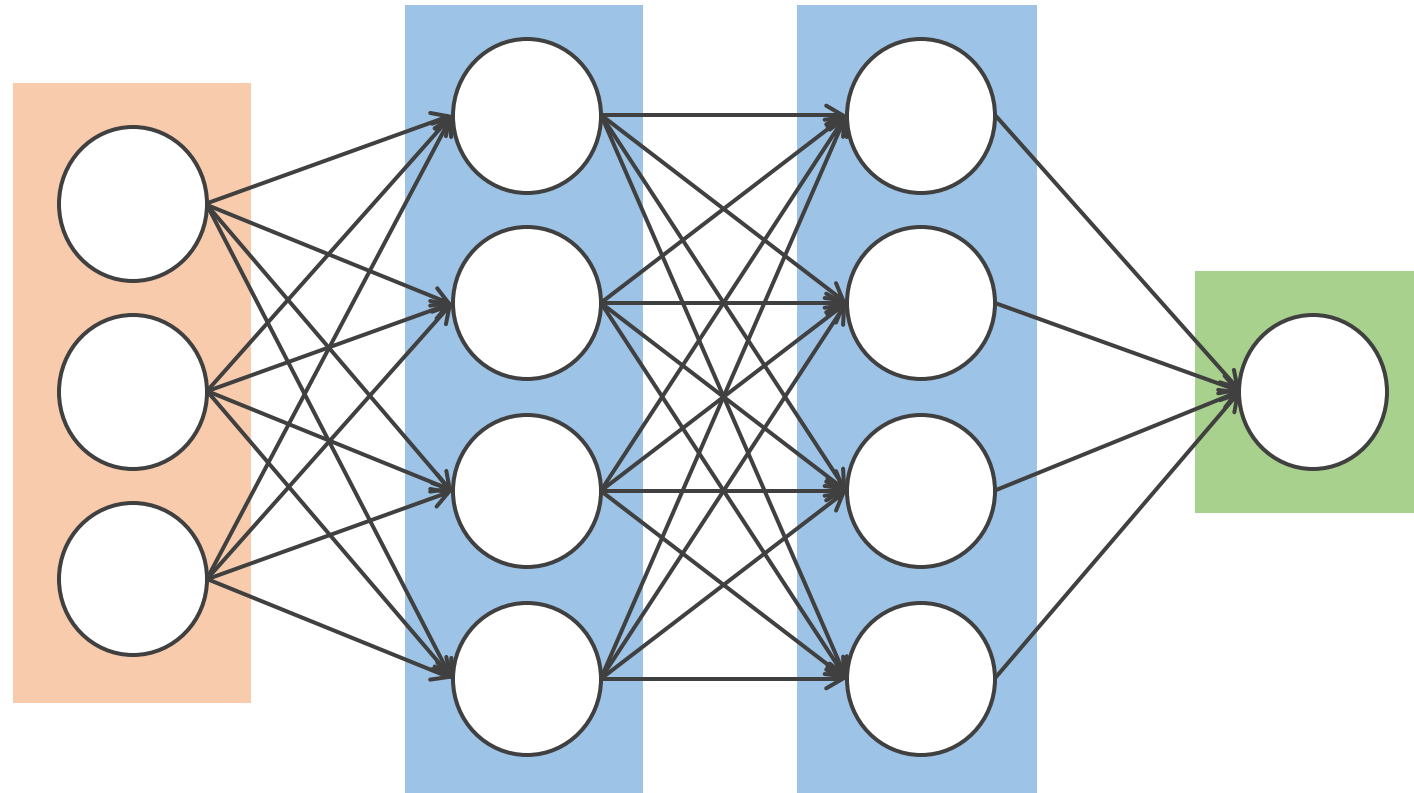
- Can quickly learn and adapt to network-wide behaviors and apply actions directly on every packet at line speed
- Hence, can provide
 - Strict security objectives with fast response time
 - Stringent service-level objectives with high performance

Taurus: A Data-Plane ML Switch

ASPLOS '22, NSDI'26 IETF/ANRP Prize, IEEE Micro Top Picks (Honorable Mention)



ML Inference: Neural Networks

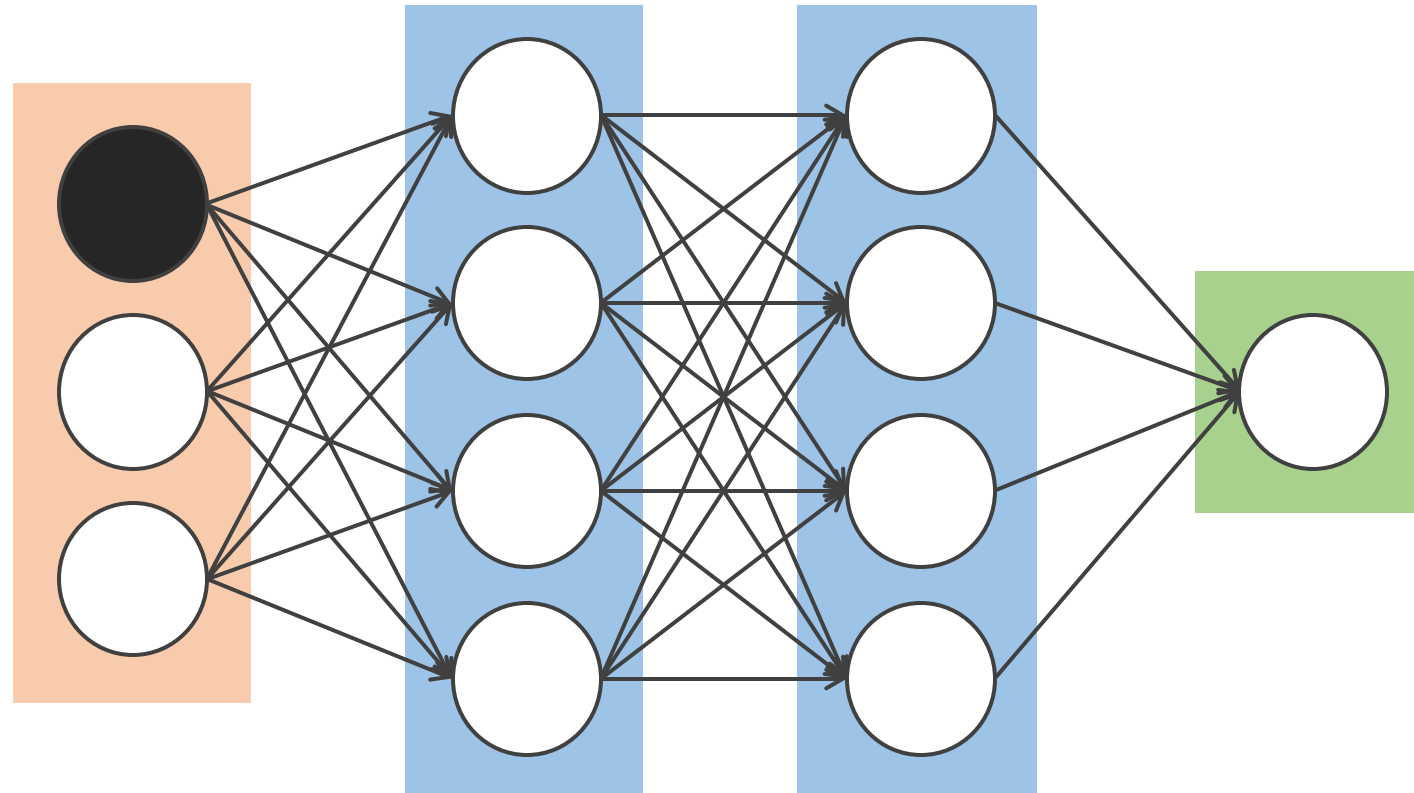


Input Layer

Hidden Layers

Output Layer

ML Inference: Neural Networks

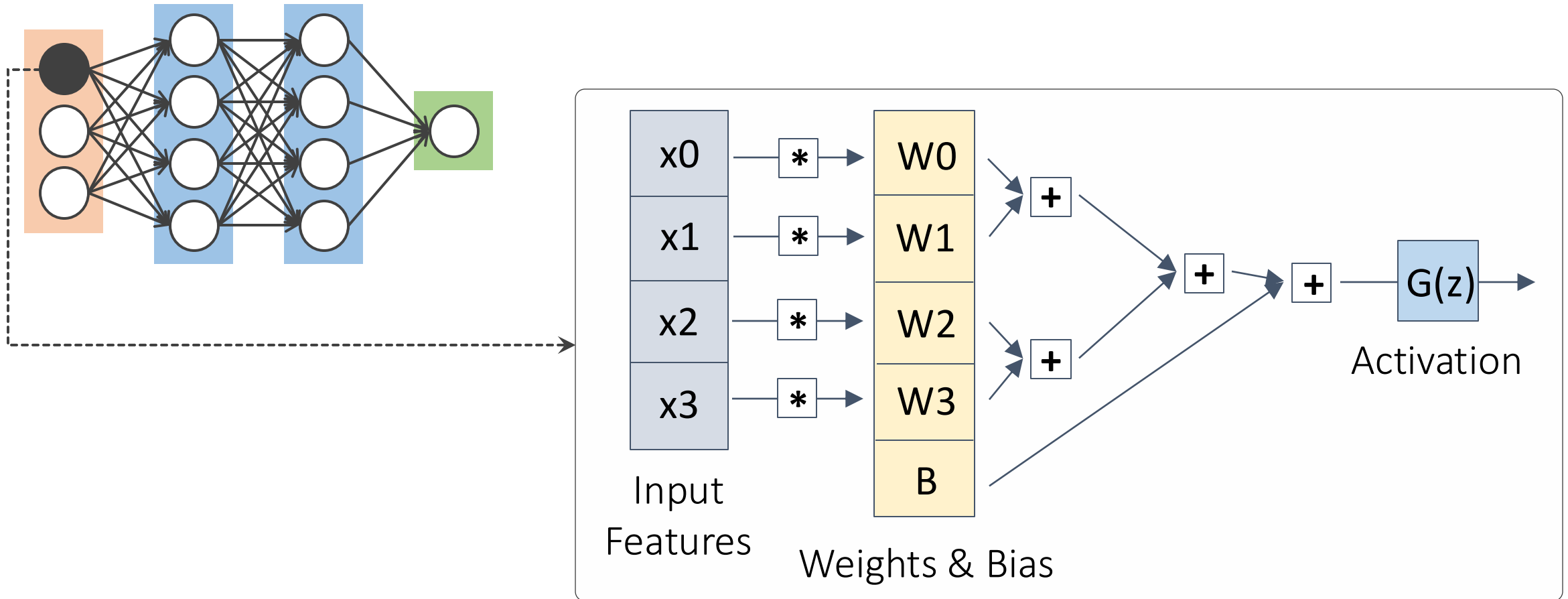


Input Layer

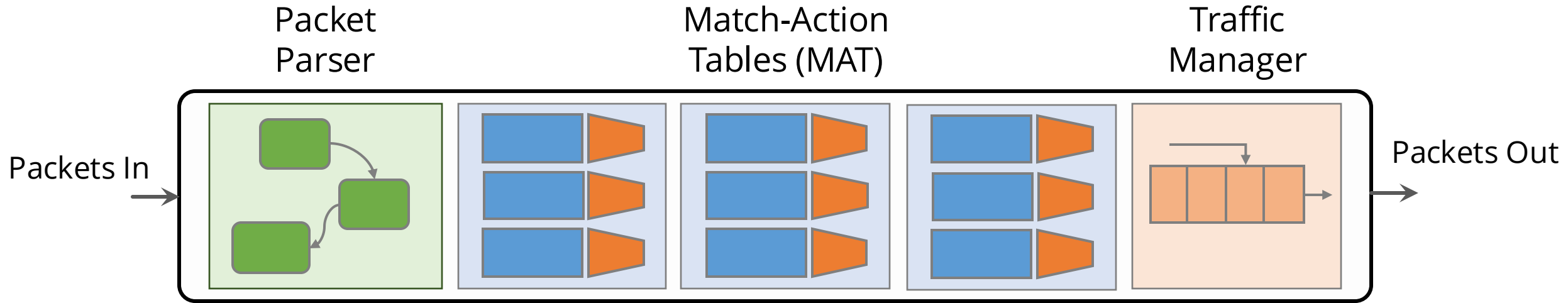
Hidden Layers

Output Layer

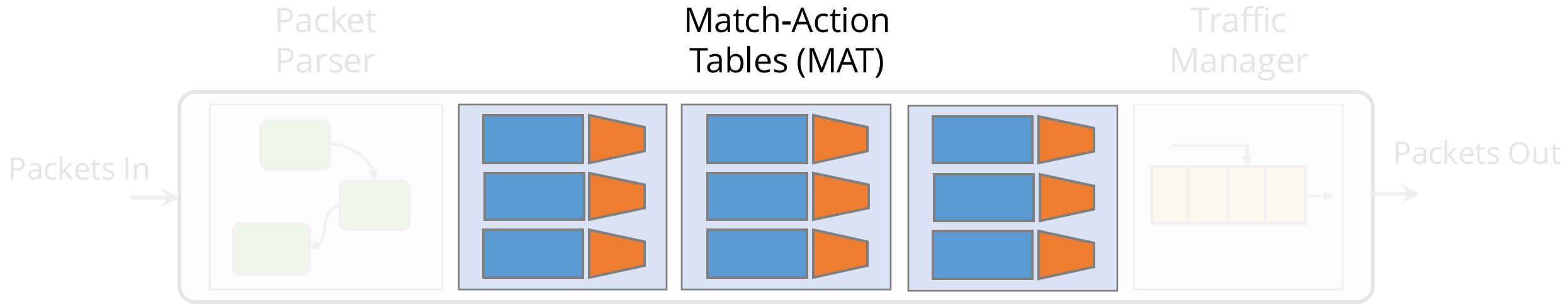
ML Inference: Neural Networks



Protocol Independent Switch Arch (PISA)



Protocol Independent Switch Arch (PISA)

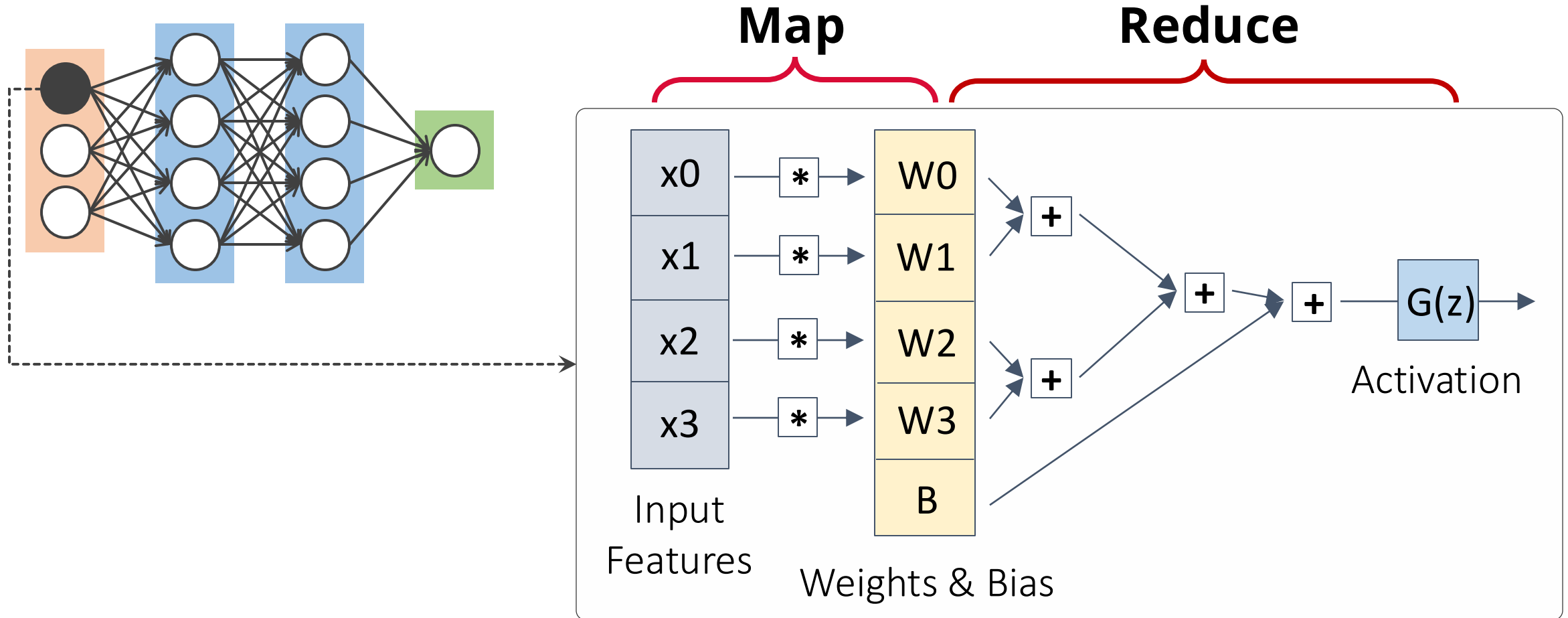


MAT == Lookup + VLIW

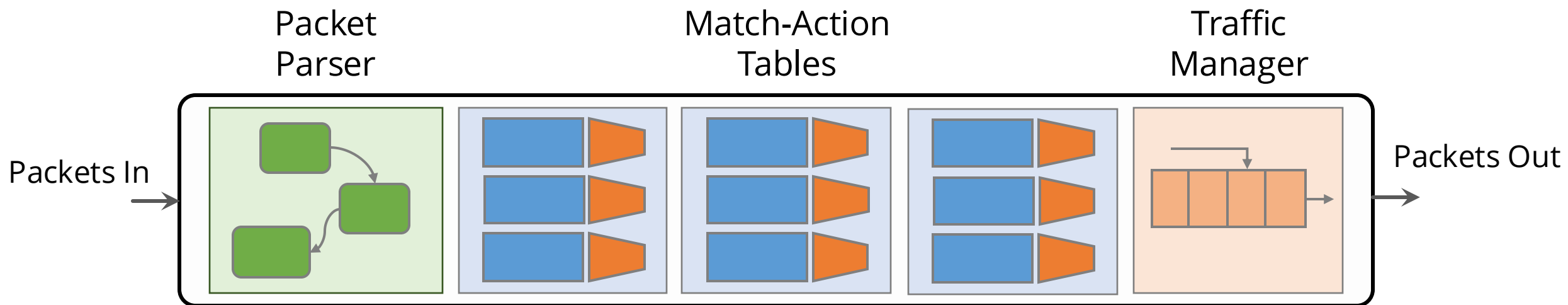
It is **not suitable** for **inference operations**:

- Arithmetic intensity is too low to perform ML operations
- Not enough intermediate storage to carry feature and state
- and more ...

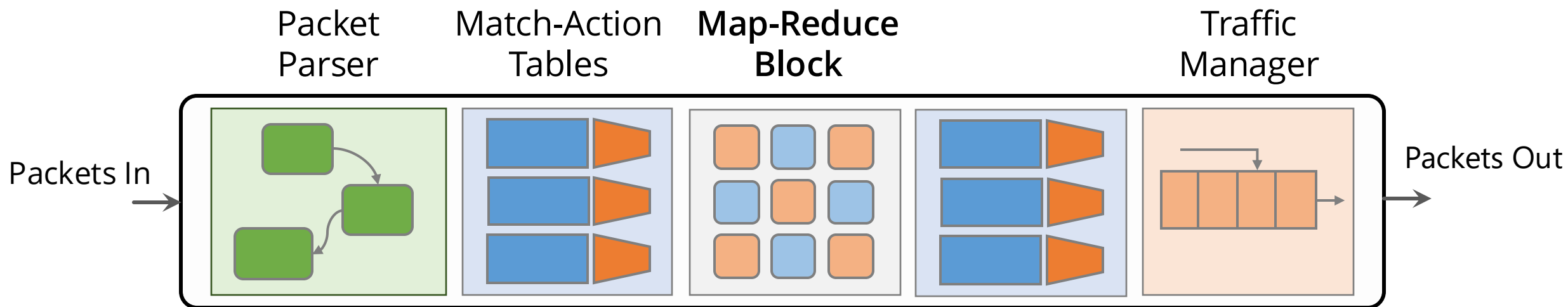
ML Inference: Neural Networks



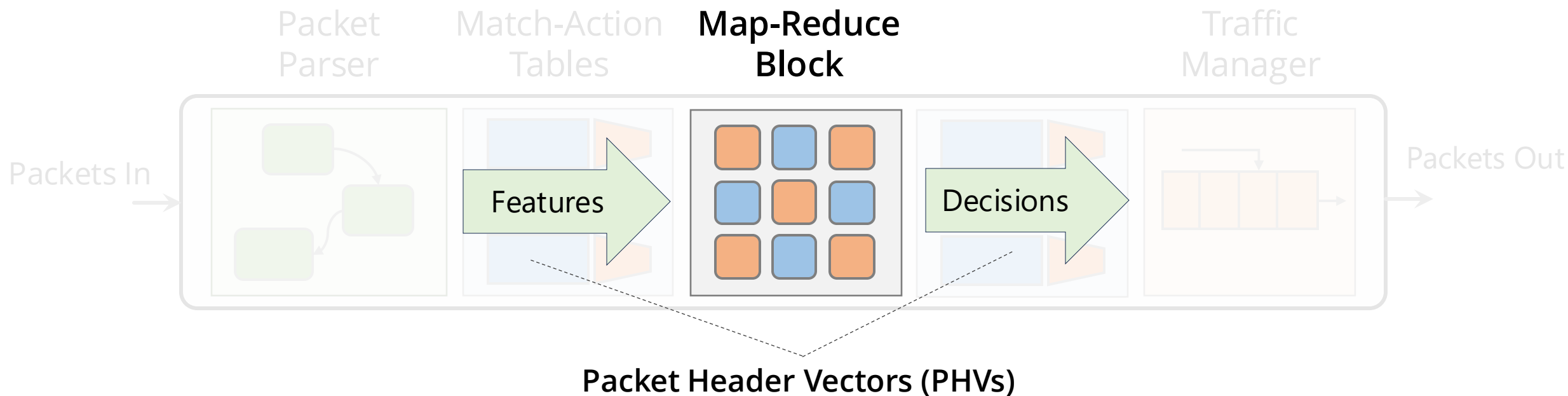
Taurus: A Data Plane ML Switch



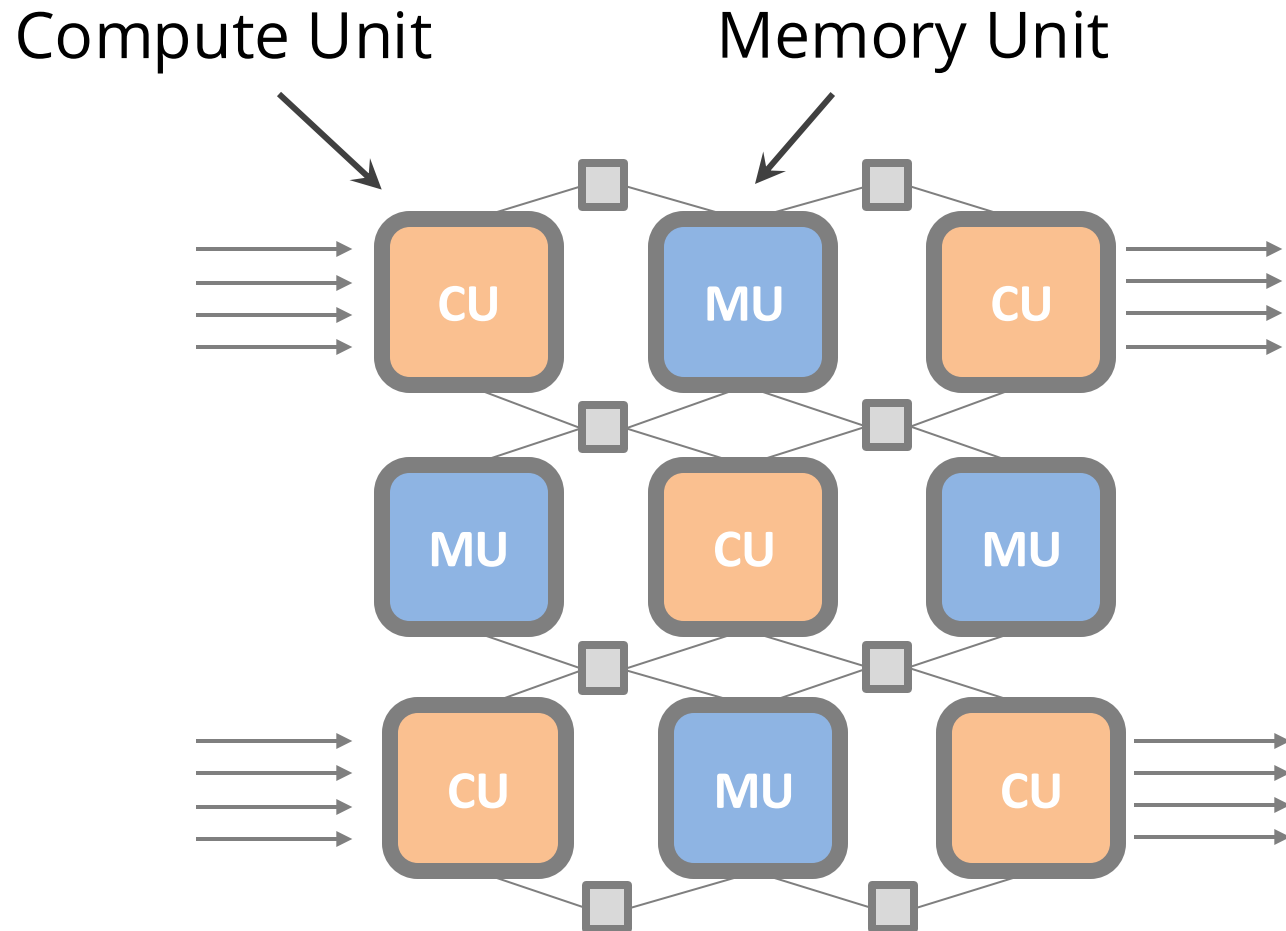
Taurus: A Data Plane ML Switch



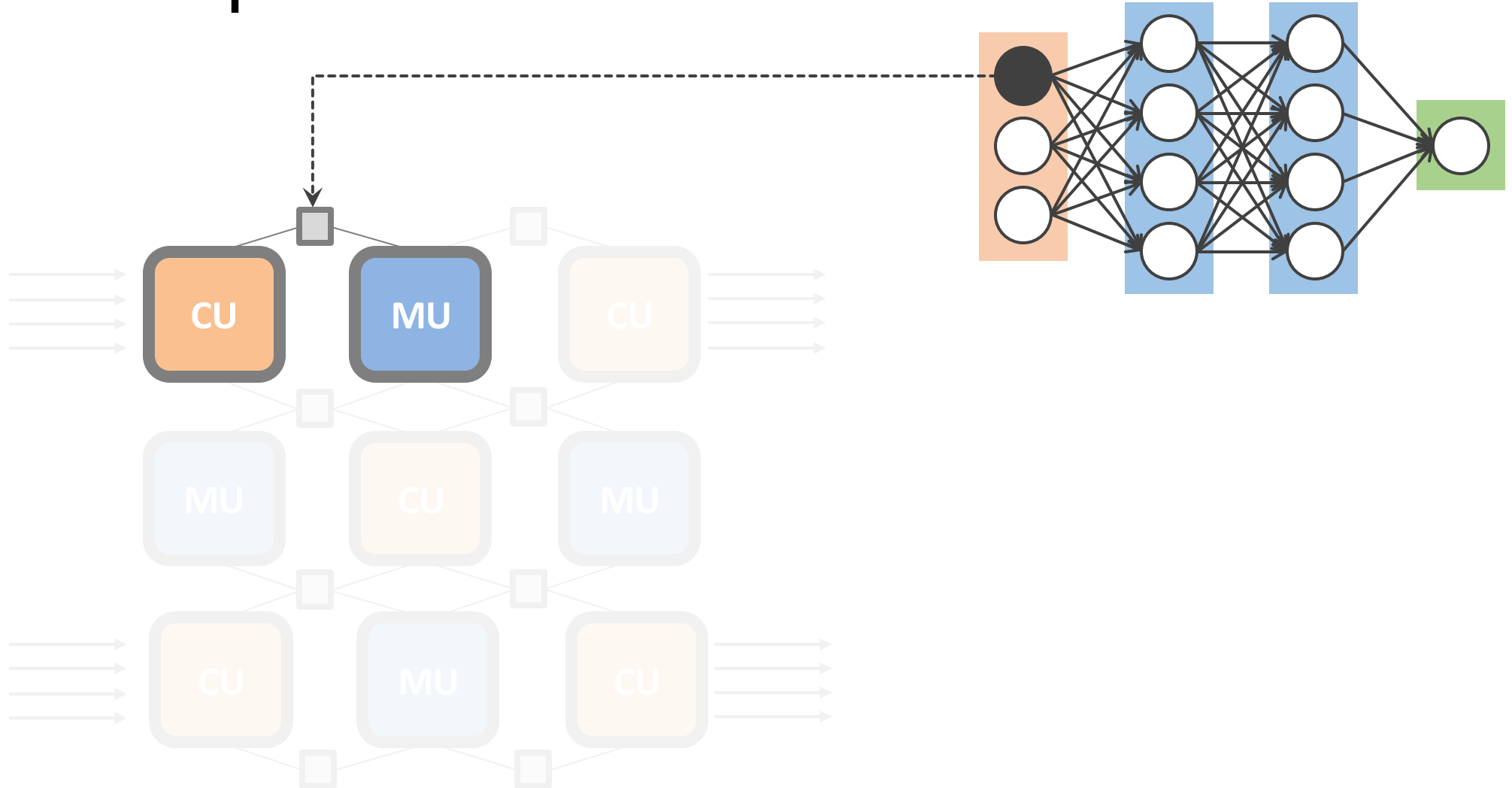
Taurus: A Data Plane ML Switch



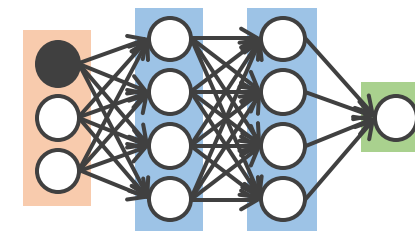
Taurus: Map-Reduce Block



Taurus: Map-Reduce Block

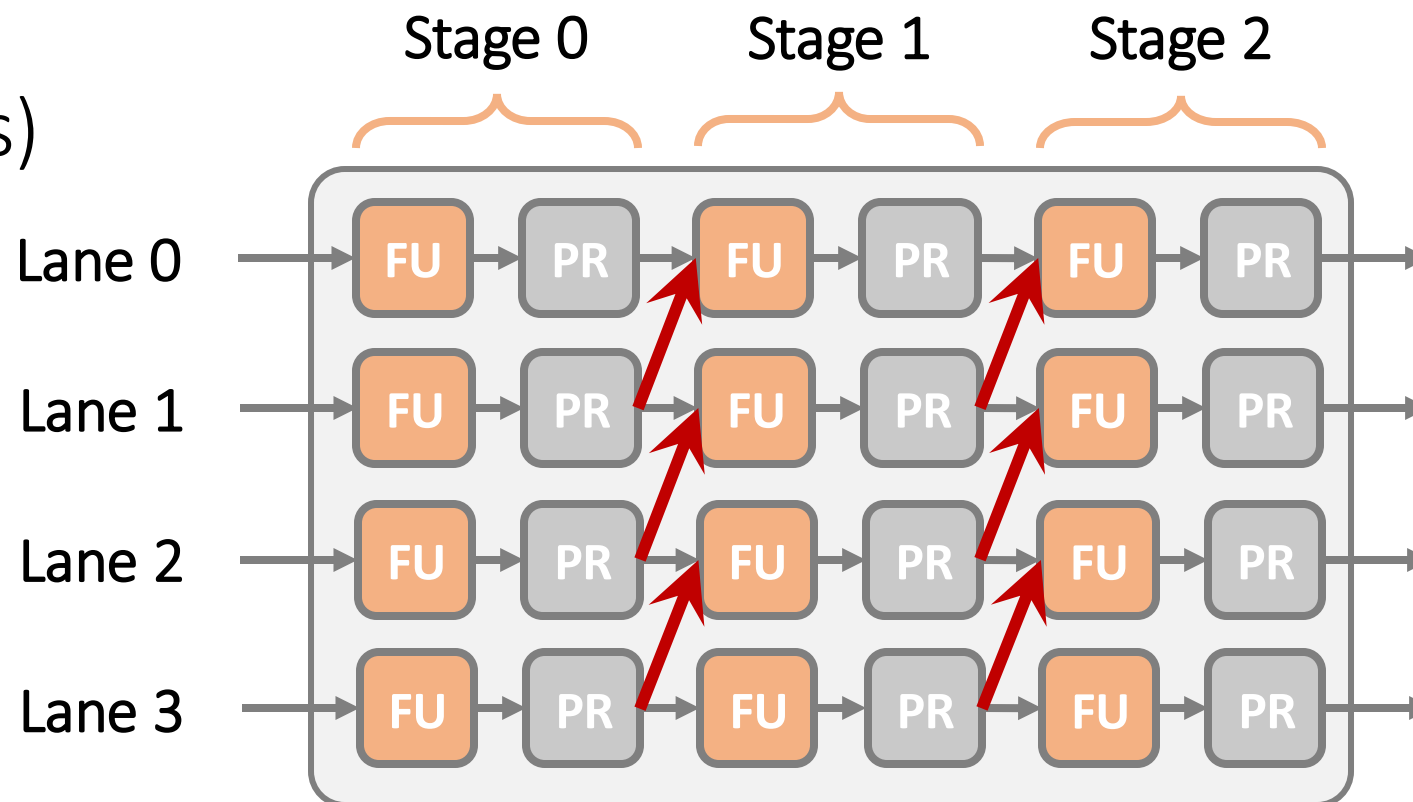


Map-Reduce Block: Control Unit



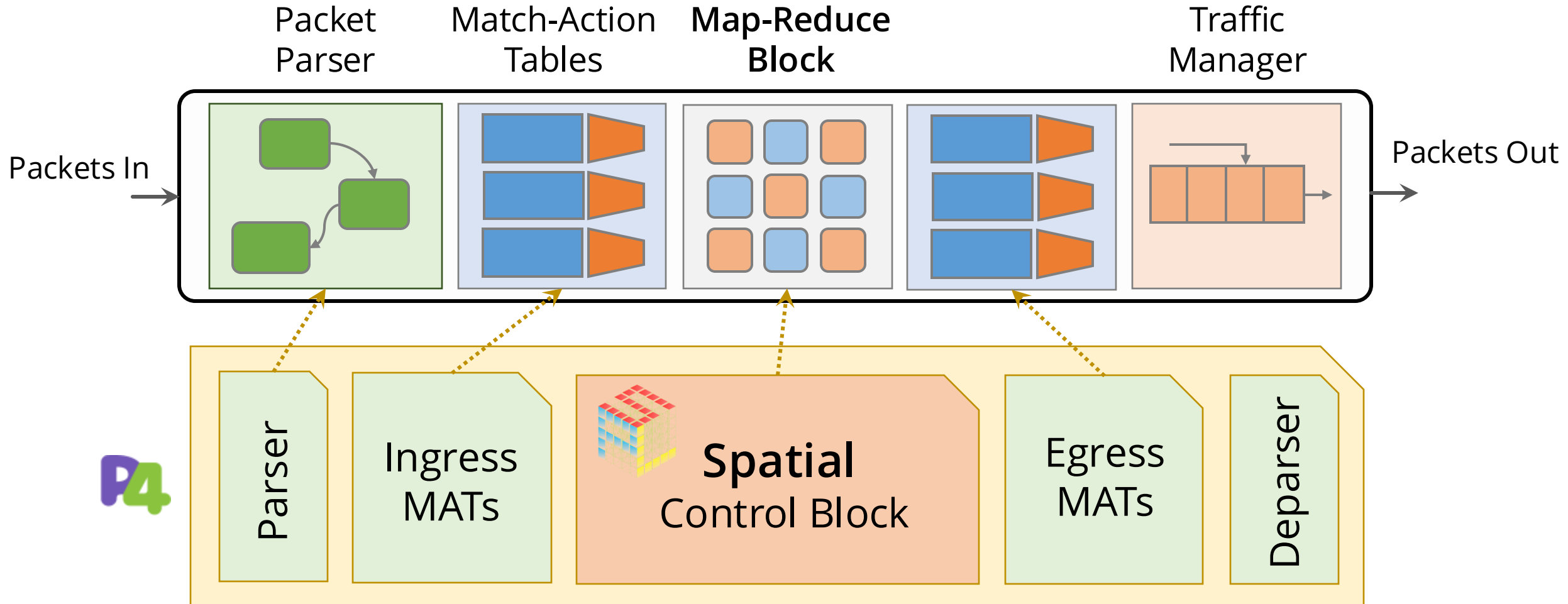
Taurus **CUs** are array-based:

- Functional Units (FUs)
- Pipeline Registers (PRs)



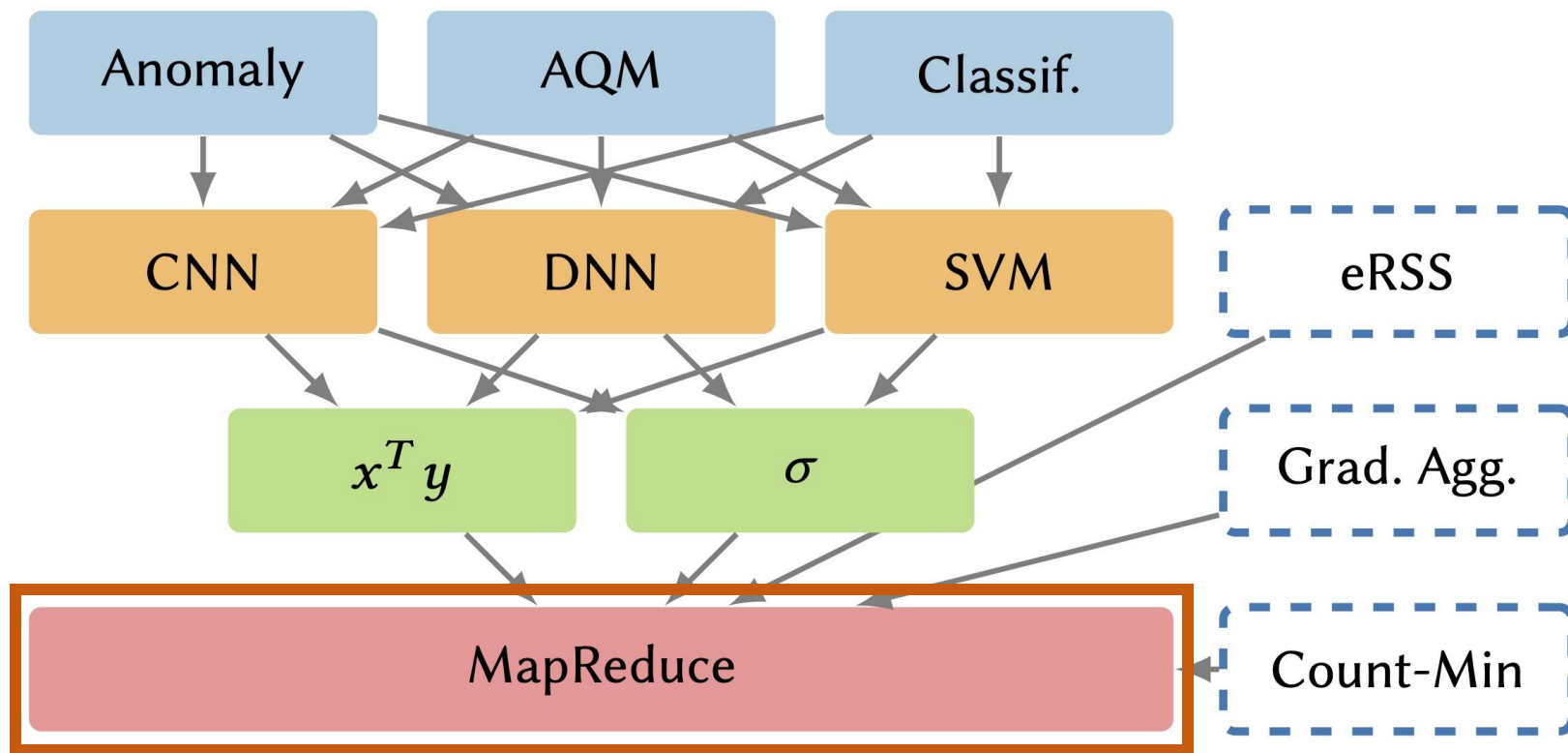
Reduction network condenses vectors to scalars

Taurus: Programming Map-Reduce



<https://spatial-lang.org> (PLDI '18)

Data-Plane Apps using MapReduce



Evaluation: Taurus ASIC Overhead

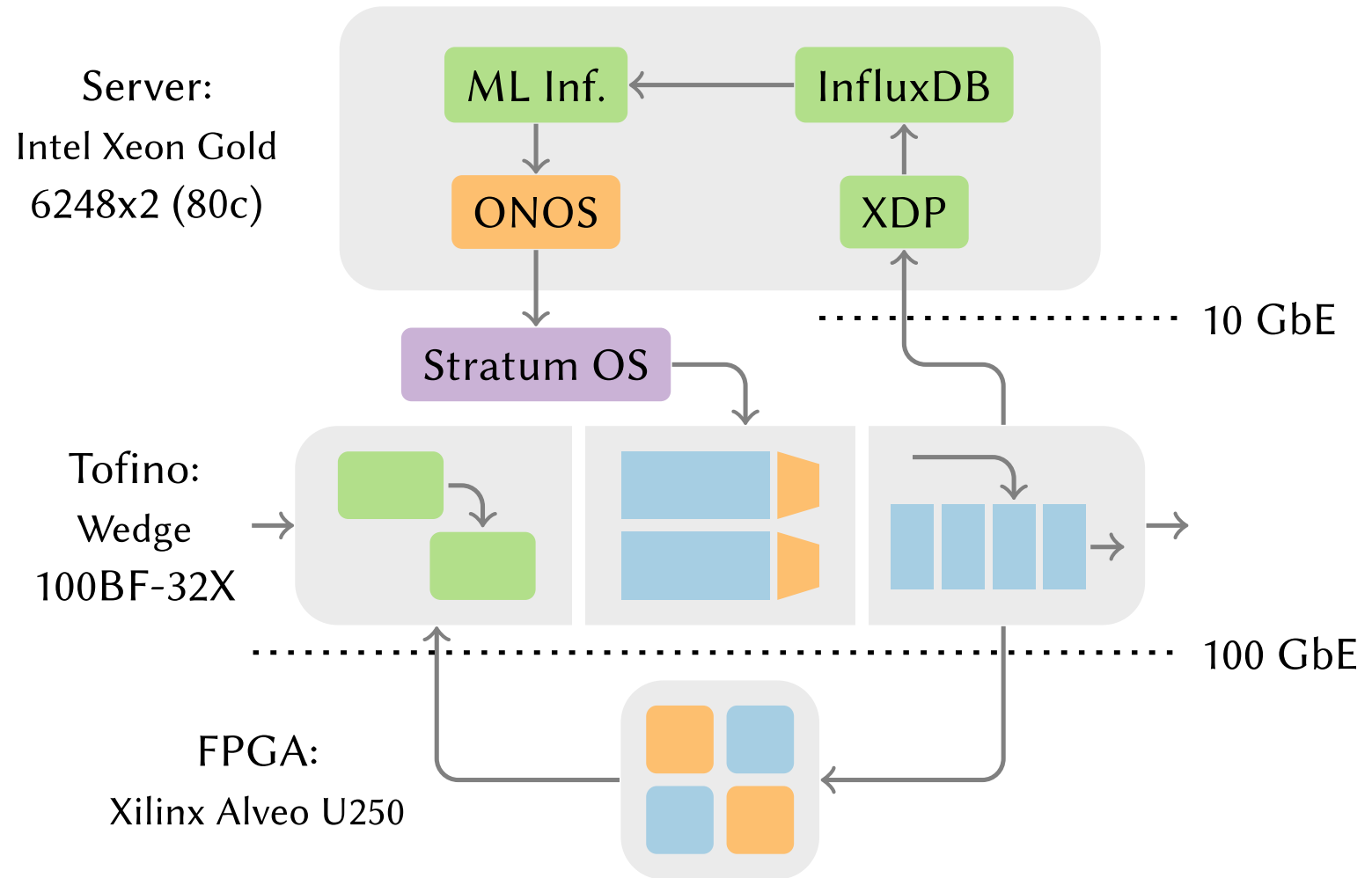
- Our evaluation platform is based on Plasticine.*
- We program our map-reduce applications using Spatial HDL.
- Anomaly detection models: SVM (8 vector), DNN (12x6x3x2 neurons)

<i>Overhead of Map-Reduce Unit</i>			Area	Power
Model	Throughput	Latency	+%	+%
SVM	1 GPkt/s	83 ns	0.5	0.6
DNN	1 GPkt/s	221 ns	0.8	1.0

Overheads are calculated relative to a 300 mm² chip with 4 reconfigurable pipelines, each drawing an estimated 25 W

* <https://github.com/stanford-ppl/PlasticineTutorial> (ISCA '17)

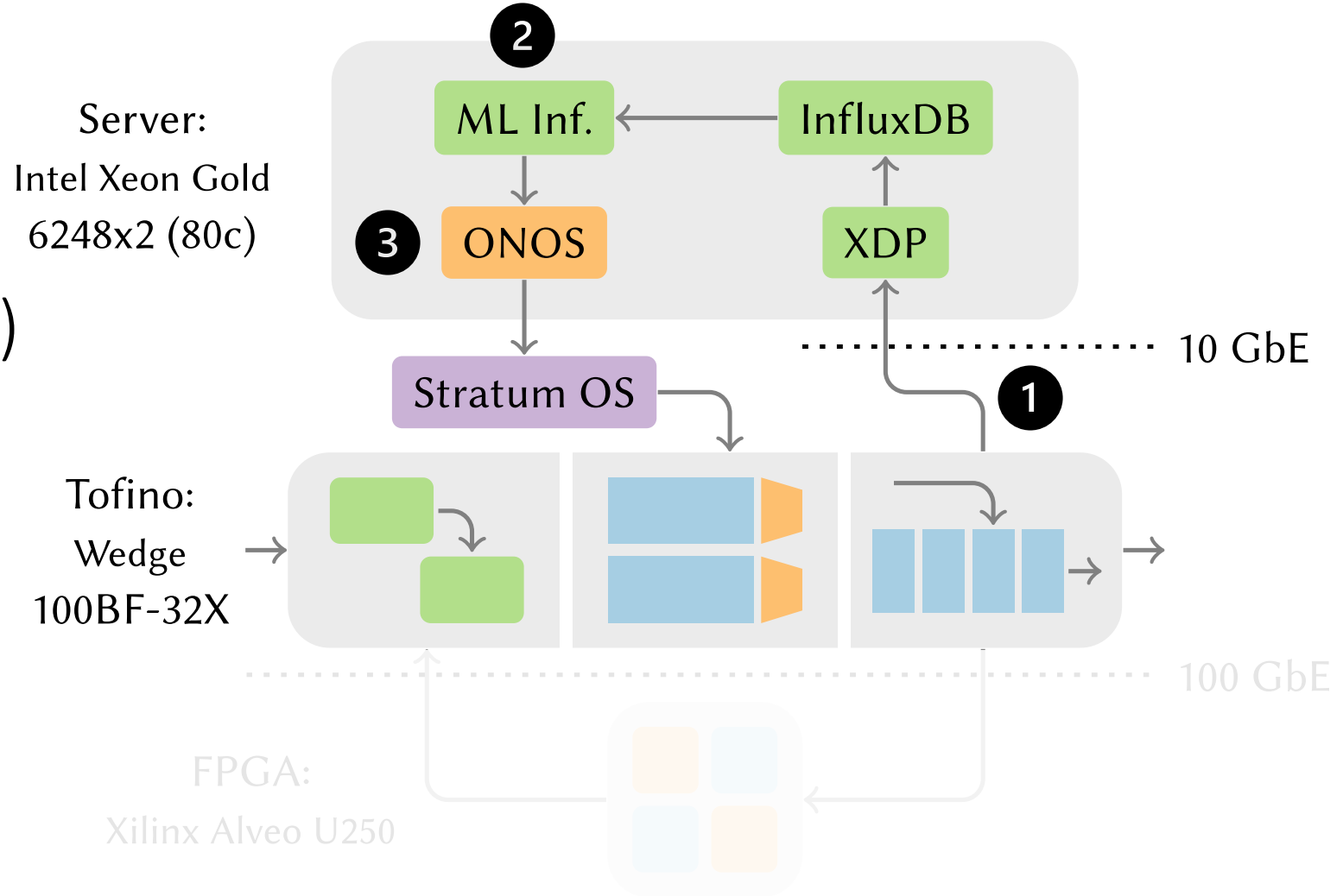
Evaluation: Taurus Converged Testbed



Evaluation: Taurus Converged Testbed

Control-Plane Baseline:

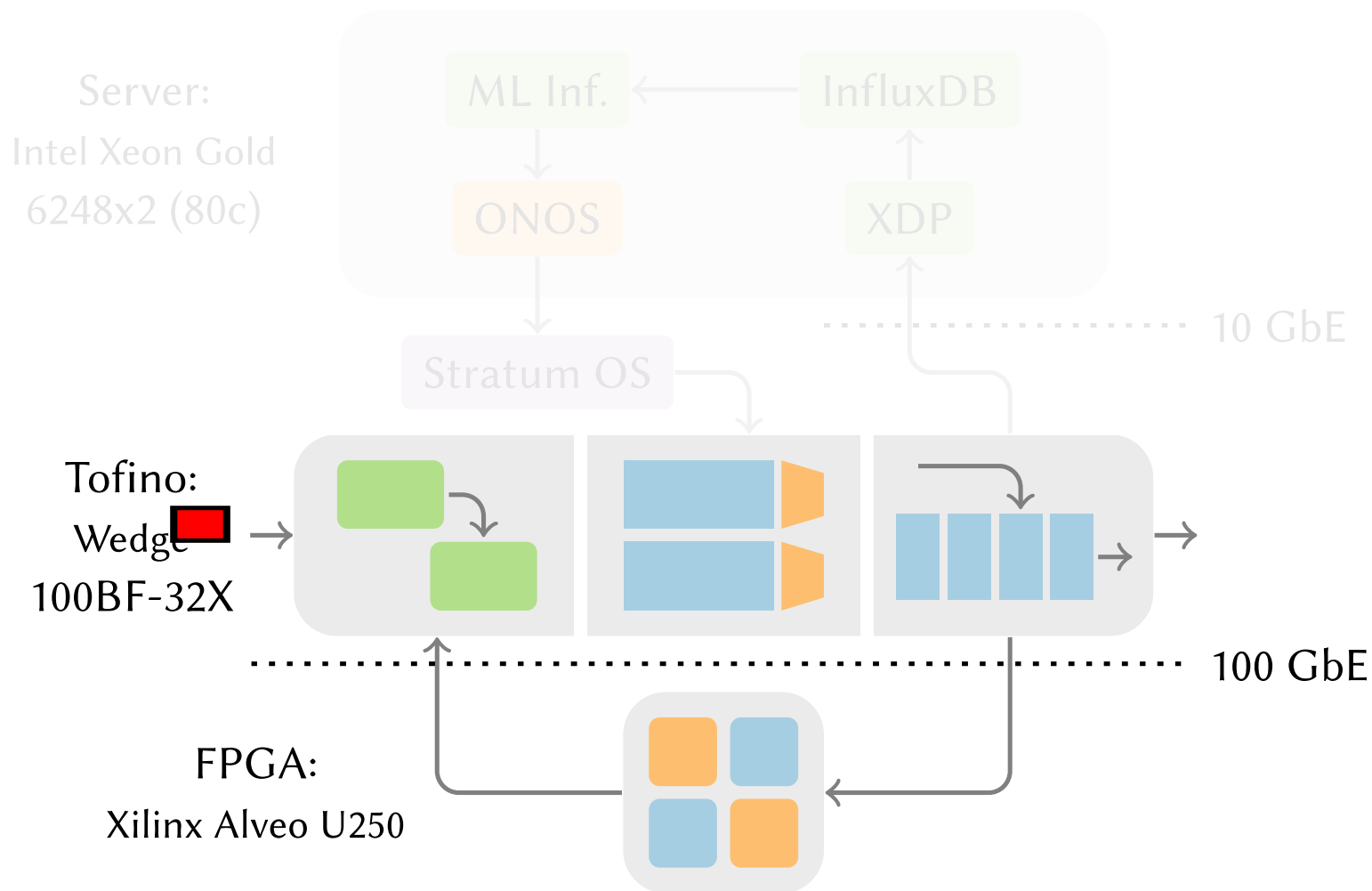
1. Sample Traffic (XDP)
2. Apply Inference (ML Inf.)
3. Install Rules (ONOS)



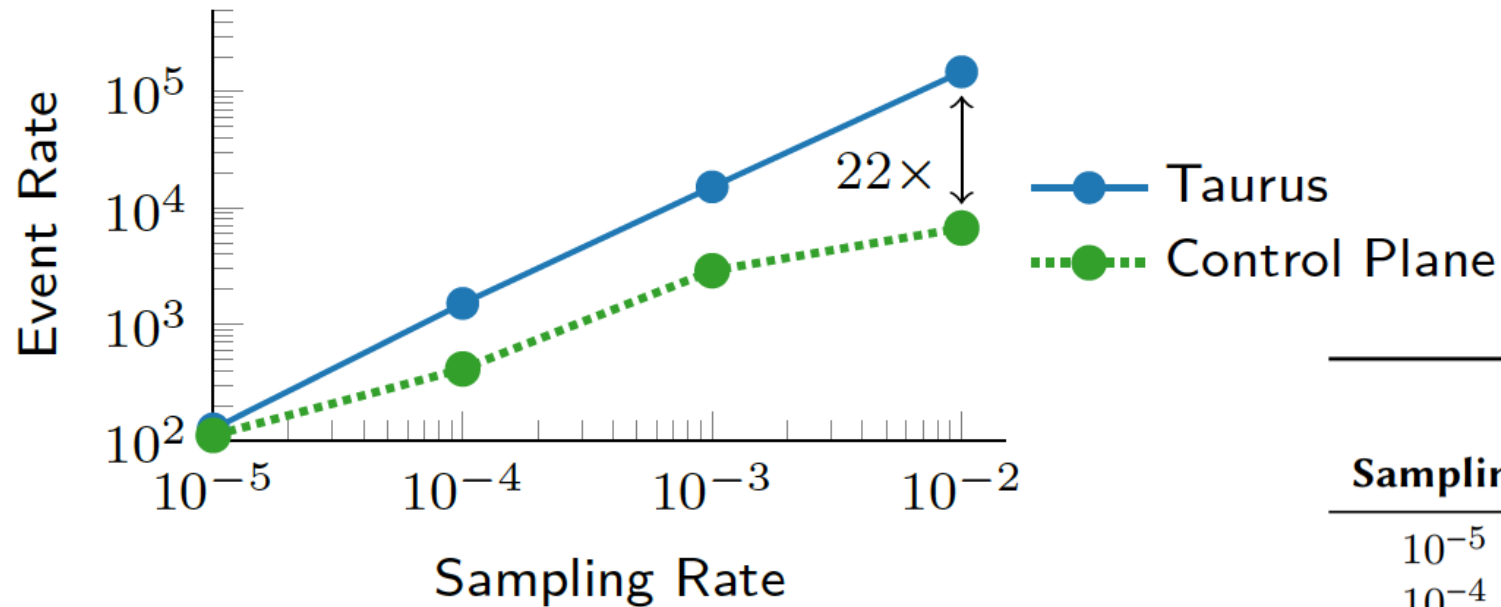
Evaluation: Taurus Converged Testbed

Taurus FPGA:

- Inline, bump-in-the-wire processing



Evaluation: Taurus Perf. Benchmarks

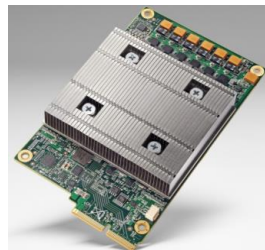
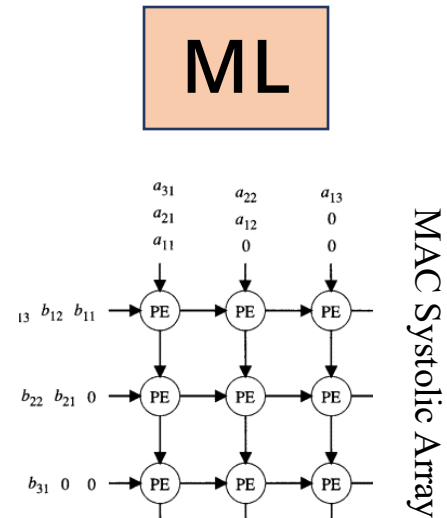


45x more accurate

Sampling	Detected (%)		F1 Score	
	SW	Taurus	SW	Taurus
10^{-5}	0.781	58.2	1.549	71.1
10^{-4}	2.553	58.2	4.944	71.1
10^{-3}	0.015	58.2	0.031	71.1
10^{-2}	0.000	58.2	0.001	71.1

- Handles 22x more anomalous events compared to a SW control-plane design
- Operates at line-rate while sustaining full model accuracy

Programmable Processors vs Switches



TPU

ML

???

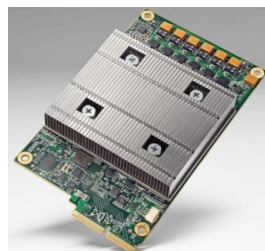
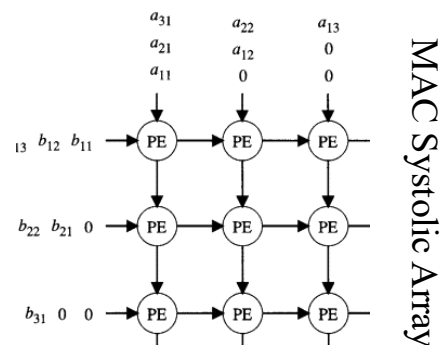
???

???

Programmable Processors vs Switches

PyTorch or TensorFlow

ML



TPU

???

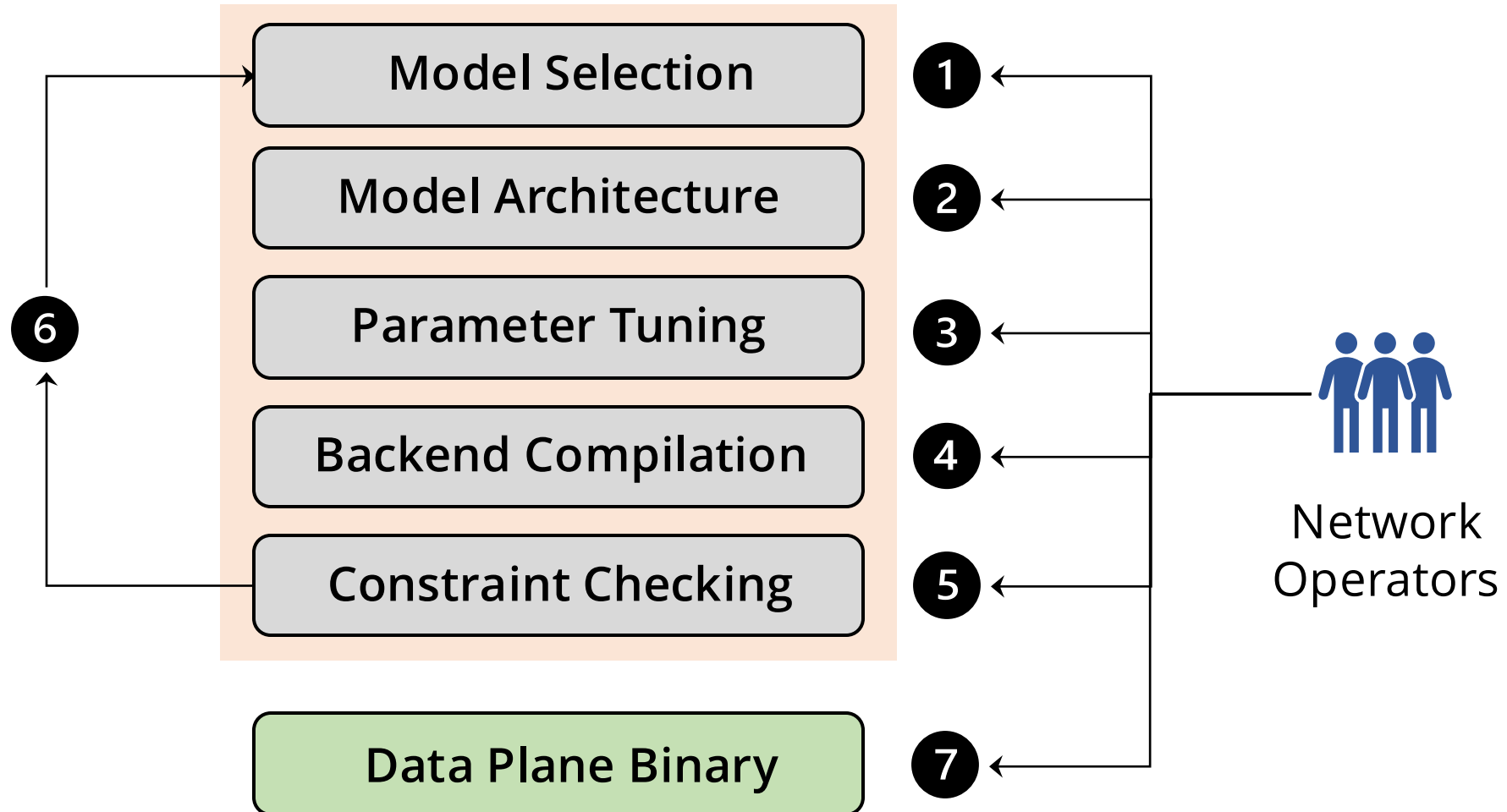
ML

MapReduce
Abstraction

Taurus

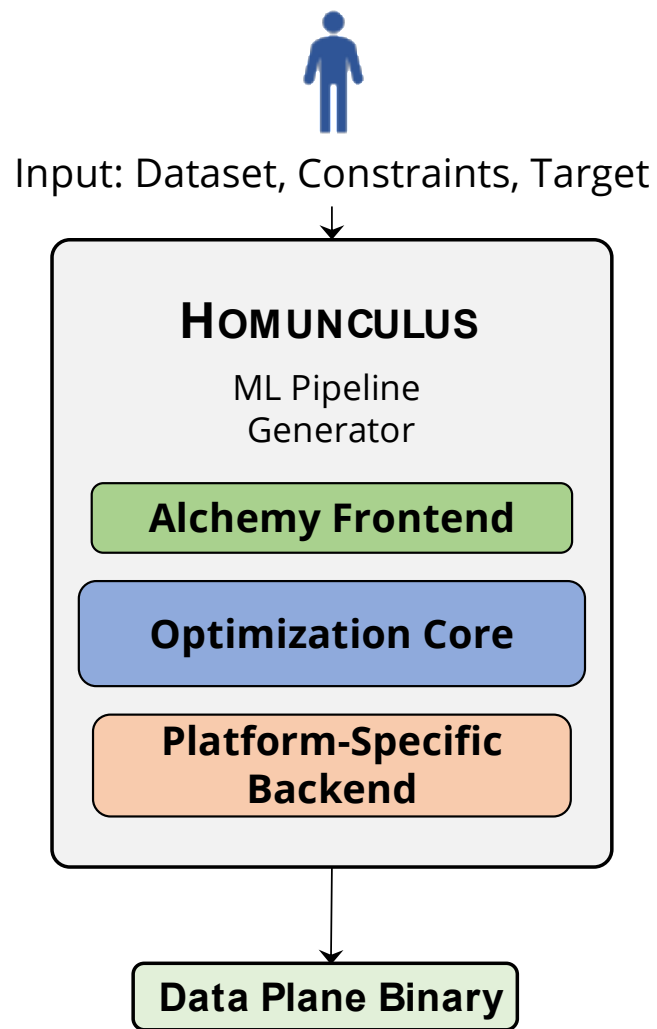
ML Switch

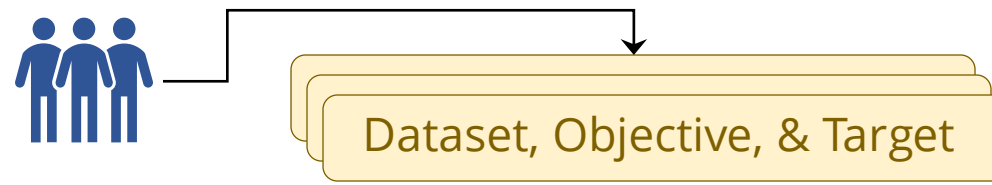
Manually Crafting ML Pipelines is Complex



Homunculus: Declarative Data-Plane ML

ASPLOS '23, Best Artifact Award







Dataset, Objective, & Target

Topology

Target Libs

Model Scheds

Alchemy
Frontend

```
1 import homunculus
2 from homunculus.alchemy import
3     DataLoader, Model, Platforms
4 import ad_loader
5
6 @DataLoader # training data loader definition
7 def wrapper_func():
8     tnx, tny = ad_loader.load_from_file(
9         "train_ad.csv")
10    tsx, tsy = ad_loader.load_from_file(
11        "test_ad.csv")
12    return {
13        "data": {"train": tnx, "test": tsx },
14        "labels": {"train": tny, "test": tsy }}
15
16 # Specify the model of choice
17 model_spec = Model({
18     "optimization_metric": ["f1"],
19     "algorithm": ["dnn"],
20     "name": "anomaly_detection",
21     "data_loader": wrapper_func })
22
23 # Load platform
24 platform = Platforms.Taurus()
25 platform.constrain(
26     "performance": {
27         "throughput": 1, # GPkt/s
28         "latency": 500 }, # ns
29     "resources": { "rows": 16, "cols": 16 })
30
31 # Schedule model and generate code
32 platform.schedule(model_spec)
33 homunculus.generate(platform)
```

(1) Import Libraries



Dataset, Objective, & Target

Topology

Target Libs

Model Scheds

Alchemy
Frontend

```
1 import homunculus
2 from homunculus.alchemy import
3     DataLoader, Model, Platforms
4 import ad_loader
5
6 @DataLoader # training data loader definition
7 def wrapper_func():
8     tnx, tny = ad_loader.load_from_file(
9         "train_ad.csv")
10    tsx, tsy = ad_loader.load_from_file(
11        "test_ad.csv")
12    return {
13        "data": {"train": tnx, "test": tsx },
14        "labels": {"train": tny, "test": tsy }}
15
16 # Specify the model of choice
17 model_spec = Model({
18     "optimization_metric": ["f1"],
19     "algorithm": ["dnn"],
20     "name": "anomaly_detection",
21     "data_loader": wrapper_func })
22
23 # Load platform
24 platform = Platforms.Taurus()
25 platform.constrain(
26     "performance": {
27         "throughput": 1, # GPkt/s
28         "latency": 500 }, # ns
29     "resources": { "rows": 16, "cols": 16 })
30
31 # Schedule model and generate code
32 platform.schedule(model_spec)
33 homunculus.generate(platform)
```



Dataset, Objective, & Target

Topology

Target Libs

Model Scheds

Alchemy
Frontend

```
1 import homunculus
2 from homunculus.alchemy import
3     DataLoader, Model, Platforms
4 import ad_loader
5
6 @DataLoader # training data loader definition
7 def wrapper_func():
8     tnx, tny = ad_loader.load_from_file(
9         "train_ad.csv")
10    tsx, tsy = ad_loader.load_from_file(
11        "test_ad.csv")
12    return {
13        "data": {"train": tnx, "test": tsx },
14        "labels": {"train": tny, "test": tsy }}
15
16 # Specify the model of choice
17 model_spec = Model({
18     "optimization_metric": ["f1"],
19     "algorithm": ["dnn"],
20     "name": "anomaly_detection",
21     "data_loader": wrapper_func })
22
23 # Load platform
24 platform = Platforms.Taurus()
25 platform.constrain(
26     "performance": {
27         "throughput": 1, # GPkt/s
28         "latency": 500 }, # ns
29     "resources": { "rows": 16, "cols": 16 })
30
31 # Schedule model and generate code
32 platform.schedule(model_spec)
33 homunculus.generate(platform)
```

(2) Load Training Data



Dataset, Objective, & Target

Topology

Target Libs

Model Scheds

Alchemy
Frontend

```
1 import homunculus
2 from homunculus.alchemy import
3     DataLoader, Model, Platforms
4 import ad_loader
5
6 @DataLoader # training data loader definition
7 def wrapper_func():
8     tnx, tny = ad_loader.load_from_file(
9         "train_ad.csv")
10    tsx, tsy = ad_loader.load_from_file(
11        "test_ad.csv")
12    return {
13        "data": {"train": tnx, "test": tsx },
14        "labels": {"train": tny, "test": tsy }}
15
16 # Specify the model of choice
17 model_spec = Model({
18     "optimization_metric": ["f1"],
19     "algorithm": ["dnn"],
20     "name": "anomaly_detection",
21     "data_loader": wrapper_func })
22
23 # Load platform
24 platform = Platforms.Taurus()
25 platform.constrain(
26     "performance": {
27         "throughput": 1, # GPkt/s
28         "latency": 500 }, # ns
29     "resources": { "rows": 16, "cols": 16 })
30
31 # Schedule model and generate code
32 platform.schedule(model_spec)
33 homunculus.generate(platform)
```

(3) Specify Model



Dataset, Objective, & Target

Topology

Target Libs

Model Scheds

Alchemy
Frontend

```
1 import homunculus
2 from homunculus.alchemy import
3     DataLoader, Model, Platforms
4 import ad_loader
5
6 @DataLoader # training data loader definition
7 def wrapper_func():
8     tnx, tny = ad_loader.load_from_file(
9         "train_ad.csv")
10    tsx, tsy = ad_loader.load_from_file(
11        "test_ad.csv")
12    return {
13        "data": {"train": tnx, "test": tsx },
14        "labels": {"train": tny, "test": tsy }}
15
16 # Specify the model of choice
17 model_spec = Model({
18     "optimization_metric": ["f1"],
19     "algorithm": ["dnn"],
20     "name": "anomaly_detection",
21     "data_loader": wrapper_func })
22
23 # Load platform
24 platform = Platforms.Taurus()
25 platform.constrain(
26     "performance": {
27         "throughput": 1, # GPkt/s
28         "latency": 500 }, # ns
29     "resources": { "rows": 16, "cols": 16 })
30
31 # Schedule model and generate code
32 platform.schedule(model_spec)
33 homunculus.generate(platform)
```

(4) Specify Platform
and Constraints



Dataset, Objective, & Target

Topology

Target Libs

Model Scheds

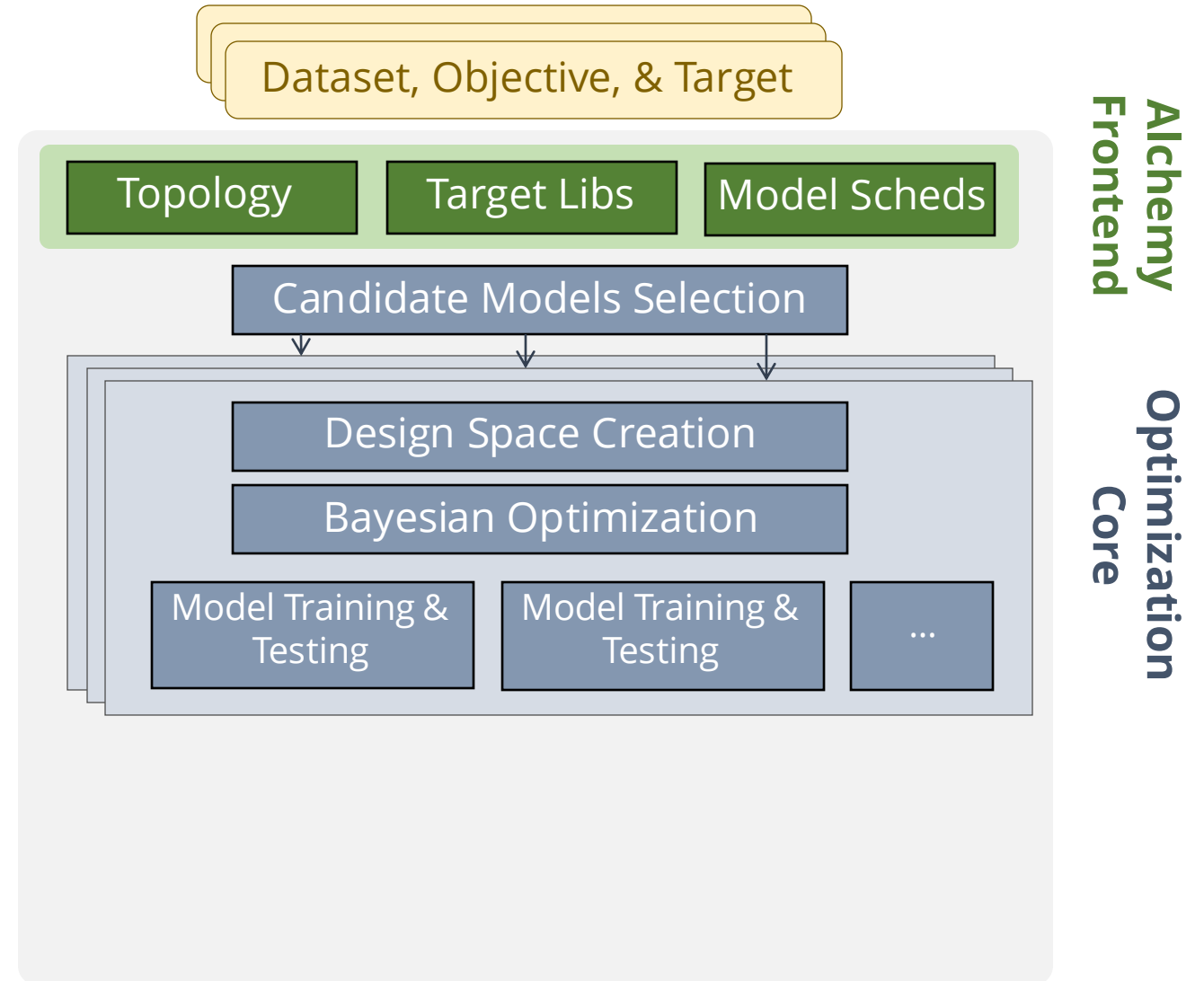
Alchemy
Frontend

```
1 import homunculus
2 from homunculus.alchemy import
3     DataLoader, Model, Platforms
4 import ad_loader
5
6 @DataLoader # training data loader definition
7 def wrapper_func():
8     tnx, tny = ad_loader.load_from_file(
9         "train_ad.csv")
10    tsx, tsy = ad_loader.load_from_file(
11        "test_ad.csv")
12    return {
13        "data": {"train": tnx, "test": tsx },
14        "labels": {"train": tny, "test": tsy }}
15
16 # Specify the model of choice
17 model_spec = Model({
18     "optimization_metric": ["f1"],
19     "algorithm": ["dnn"],
20     "name": "anomaly_detection",
21     "data_loader": wrapper_func })
22
23 # Load platform
24 platform = Platforms.Taurus()
25 platform.constrain(
26     "performance": {
27         "throughput": 1, # GPkt/s
28         "latency": 500 }, # ns
29     "resources": { "rows": 16, "cols": 16 })
30
31 # Schedule model and generate code
32 platform.schedule(model_spec)
33 homunculus.generate(platform)
```

(5) Run Homunculus

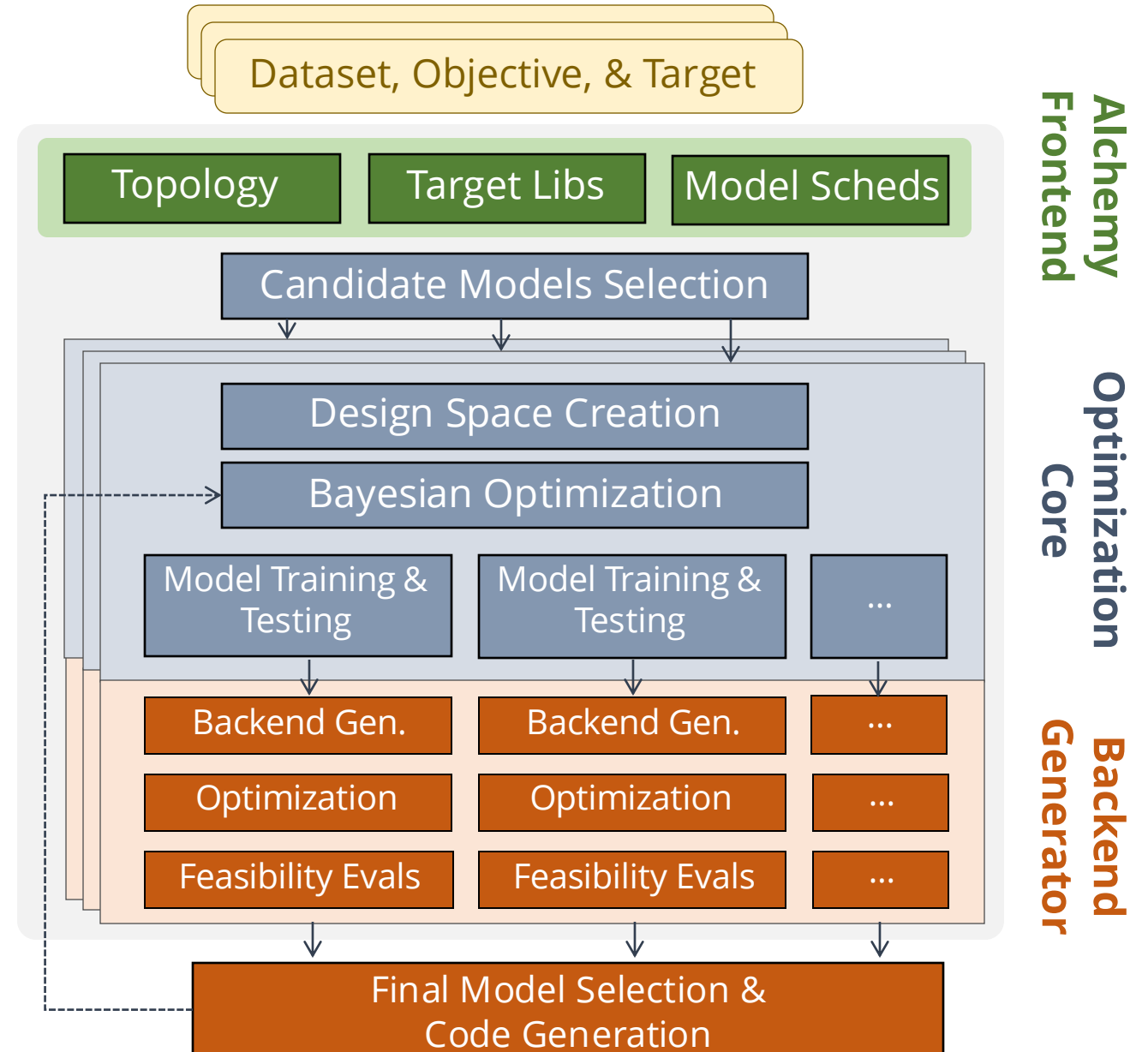
The optimization core builds ML model candidates by:

- Creating a design space
- Running Bayesian optimization
- Model training and testing



The backend generator is responsible for:

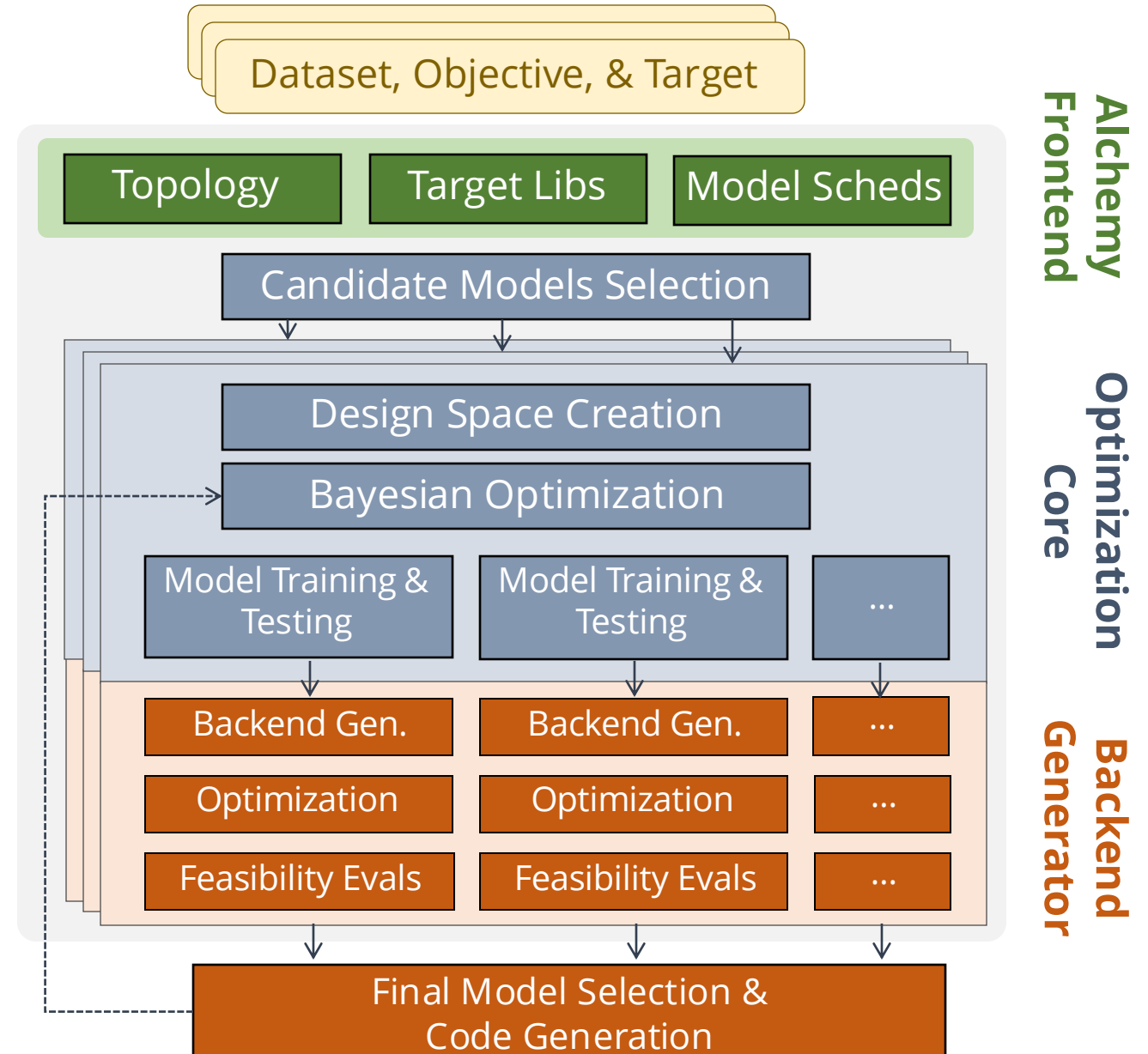
- Performing feasibility tests and to iteratively send results back to the optimization core
- Generating the final binary for the target backend (e.g., as P4 or Spatial code)



Why Bayesian Optimization?

- Optimization core must consider outcomes from prior iterations to give progressively better results
- Due to the complexity of the systems, the optimization core must operate in a black-box setting (with inputs and outputs).

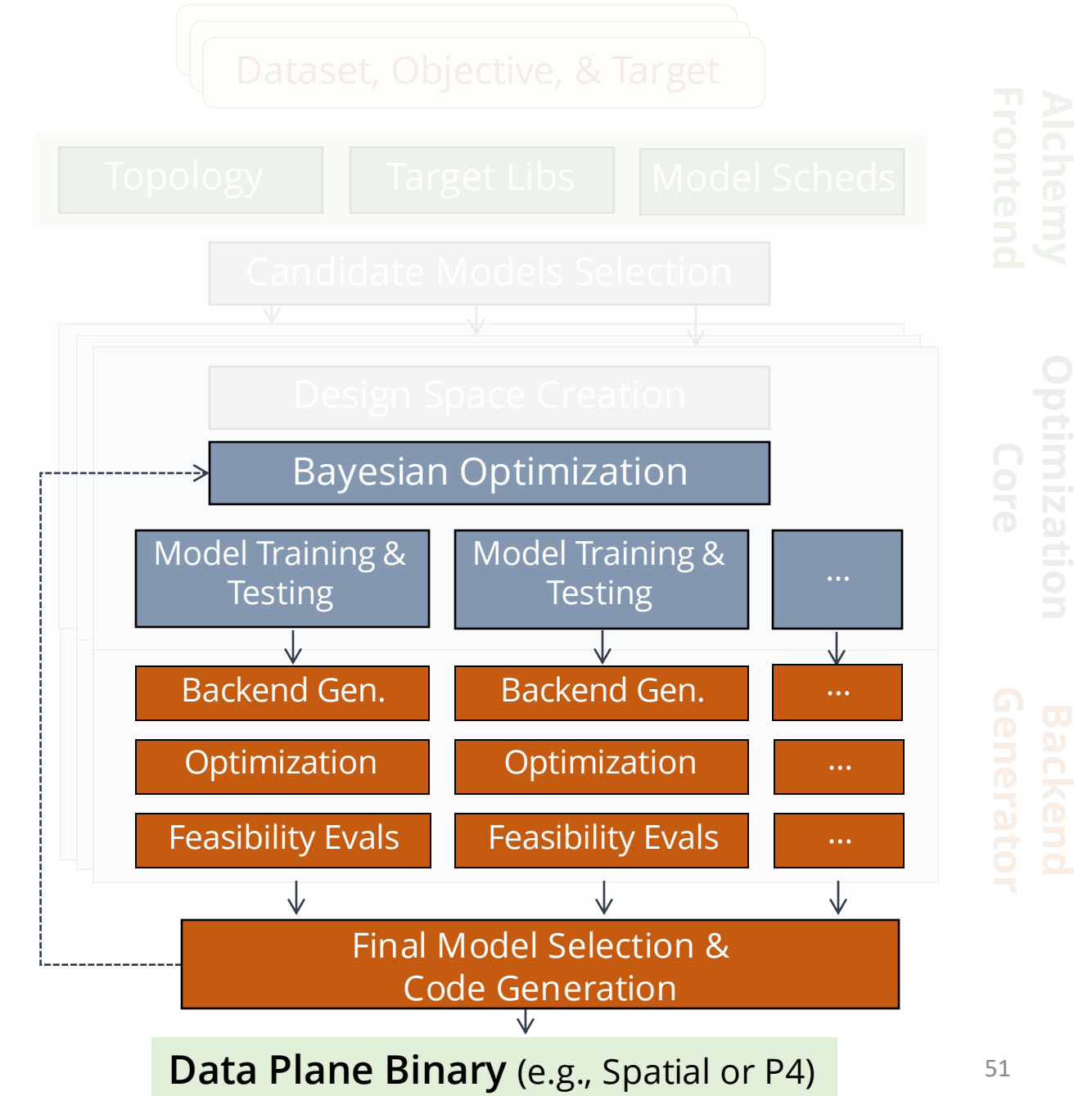
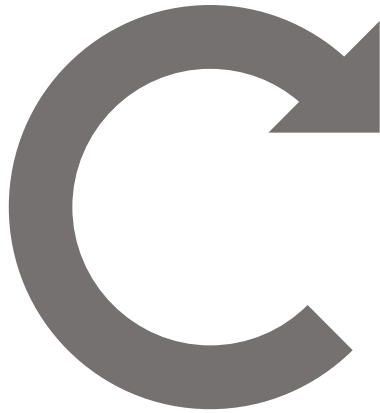
Use the open-source HyperMapper tool.*



* <https://github.com/luinardi/hypermapper> (MASCOT '19)

Homunculus Optimization Loop

- Repeats **N** times before generating the target binary.



Evaluation: Homunculus Benchmarks

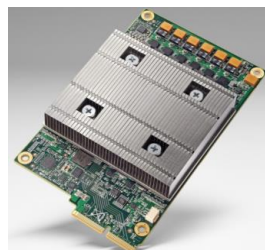
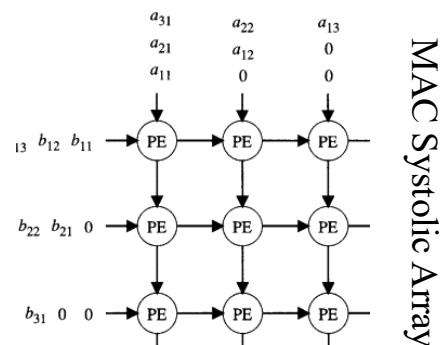
- F1 Score improvements in anomaly detection (**AD**), traffic classification (**TC**), botnet detection (**BD**) with $N = 20$

App	Features	# NN Param	F1 Score	
Base-AD	7	203	71.10	
Hom-AD	7	254	83.10	+16.9%
Base-TC	7	275	61.04	
Hom-TC	7	370	68.75	+12.6%
Base-BD	30	662	77.0	
Hom-BD	30	501	79.8	+3.6%

Programmable Processors vs Switches

PyTorch or TensorFlow

ML



TPU

???

ML

MapReduce
Abstraction

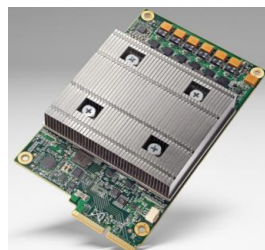
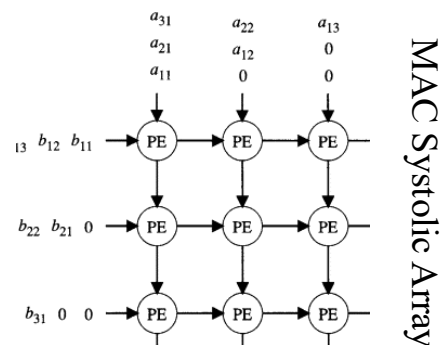
Taurus

ML Switch

Programmable Processors vs Switches

PyTorch or TensorFlow

ML



TPU

Homunculus

ML

OSDI '24
SRC JUMP 2.0 Best Paper

Caravan
LLM-Assisted
Online Learning

MapReduce
Abstraction



Taurus

ML Switch

Open Research Challenges

Co-Design of

ML

+

Network

+

Architecture

Feature engineering

Model design

Data collection

Operator Interfaces

Auto labeling

And more ...

Homunculus

ML

MapReduce
Abstraction

Scaling to larger models
- via **Chiplets**

Newer switch
abstractions

Handling sparsity

Floating-points ops

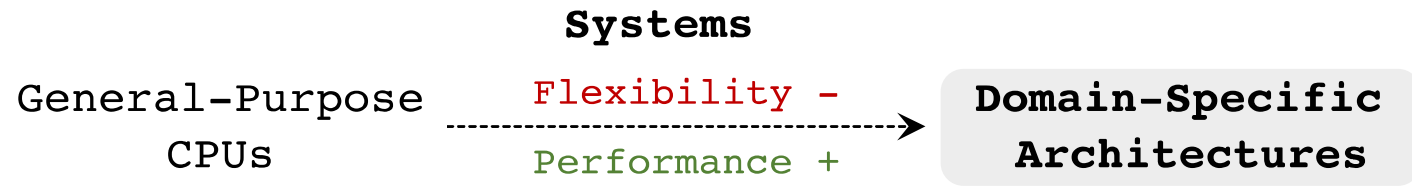
Model training

And more ...

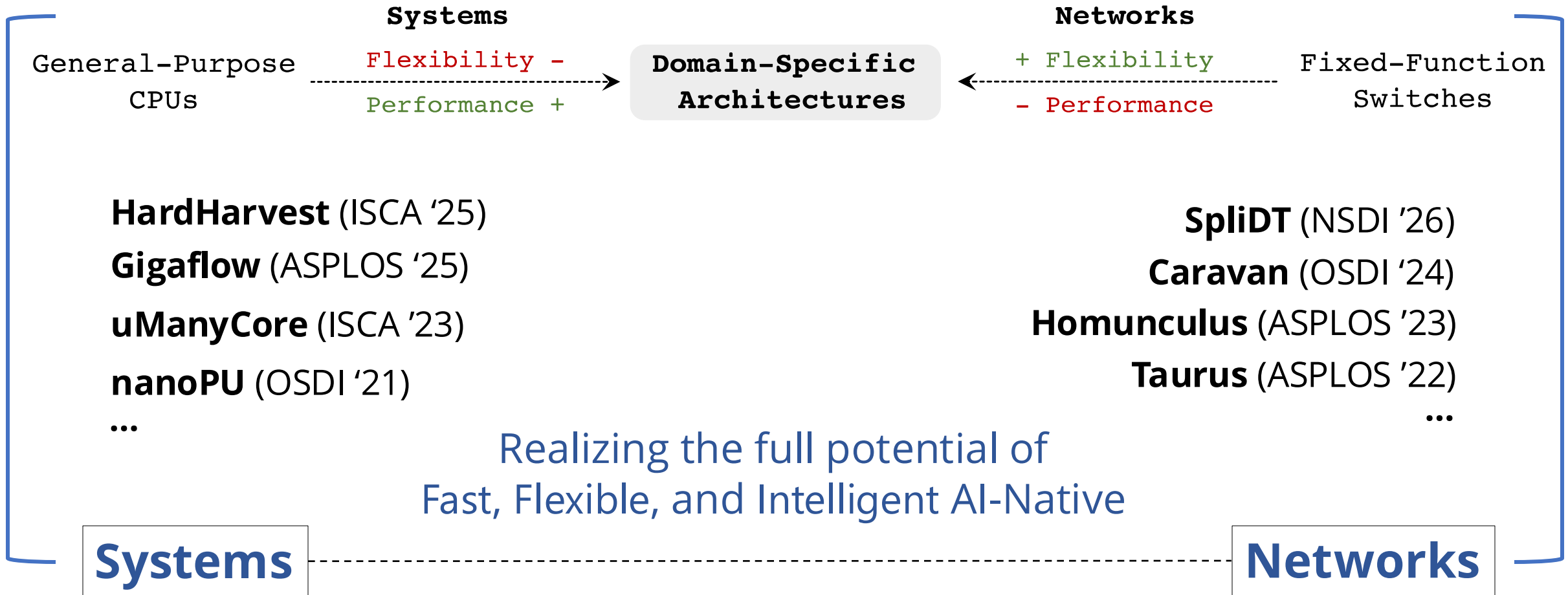
Taurus

ML Switch

Broader Research Vision and Goals



Broader Research Vision and Goals



Questions?



[Mission](#) [News](#) [Team](#) [Publications](#) [Funding](#)

[Contact Us](#)

NextGArch Lab

Pioneering the Future of NextGen Systems and Networks!

At the University of Michigan, NextGArch Lab is dedicated to revolutionizing computer systems, networks, and architectures through cutting-edge research and innovation in CSE and EECS.

[Learn More](#)

