Building Adaptive Networks with Machine Learning

Tushar Swamy

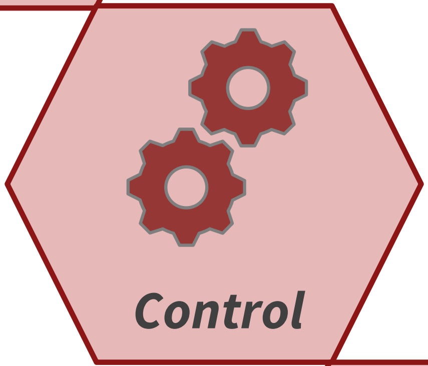
Stanford University

Networks need data-driven decisions:

→ ***Machine Learning***

Networks can benefit from machine learning

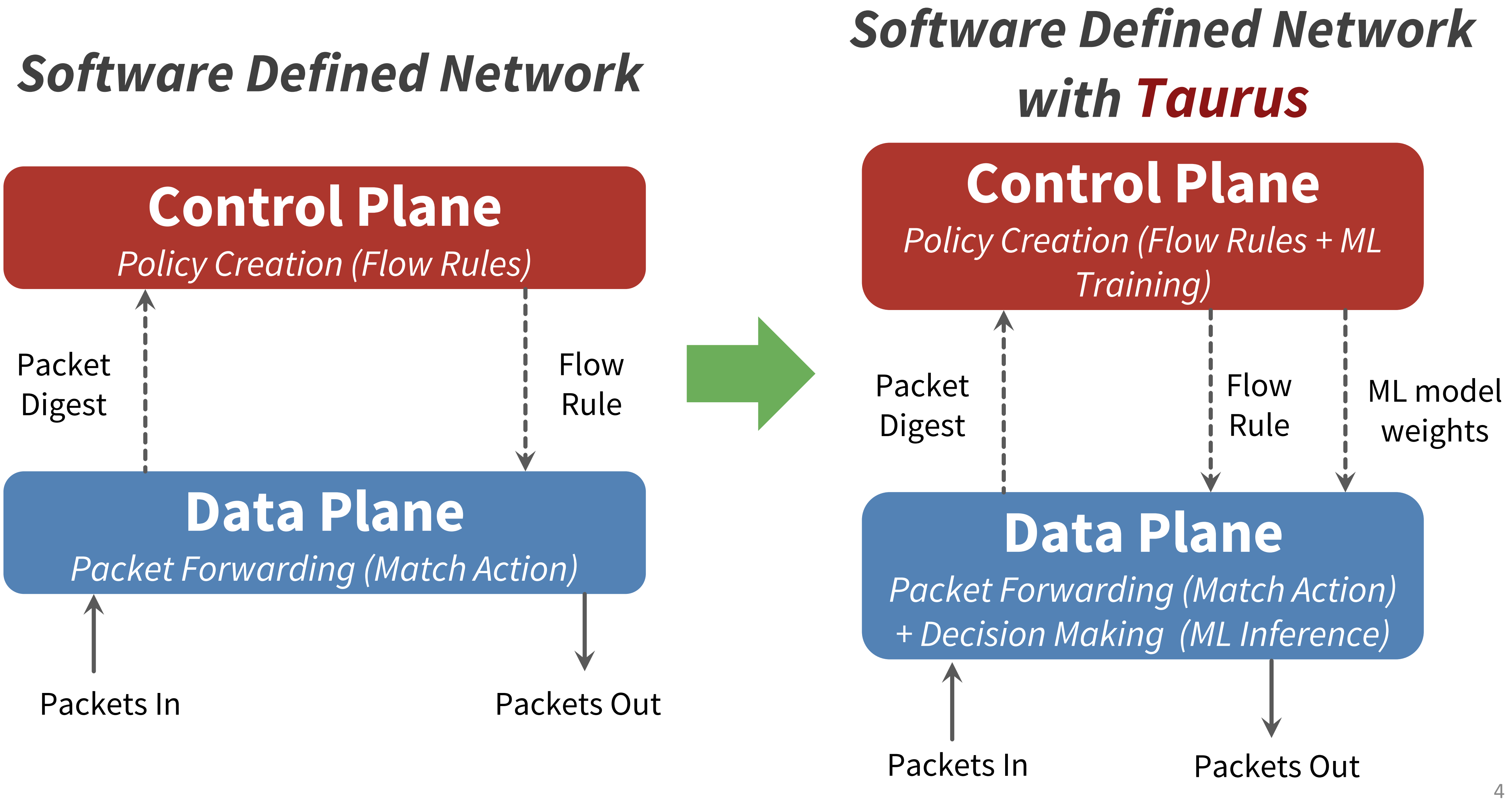
Anomaly Detection
Traffic Classification



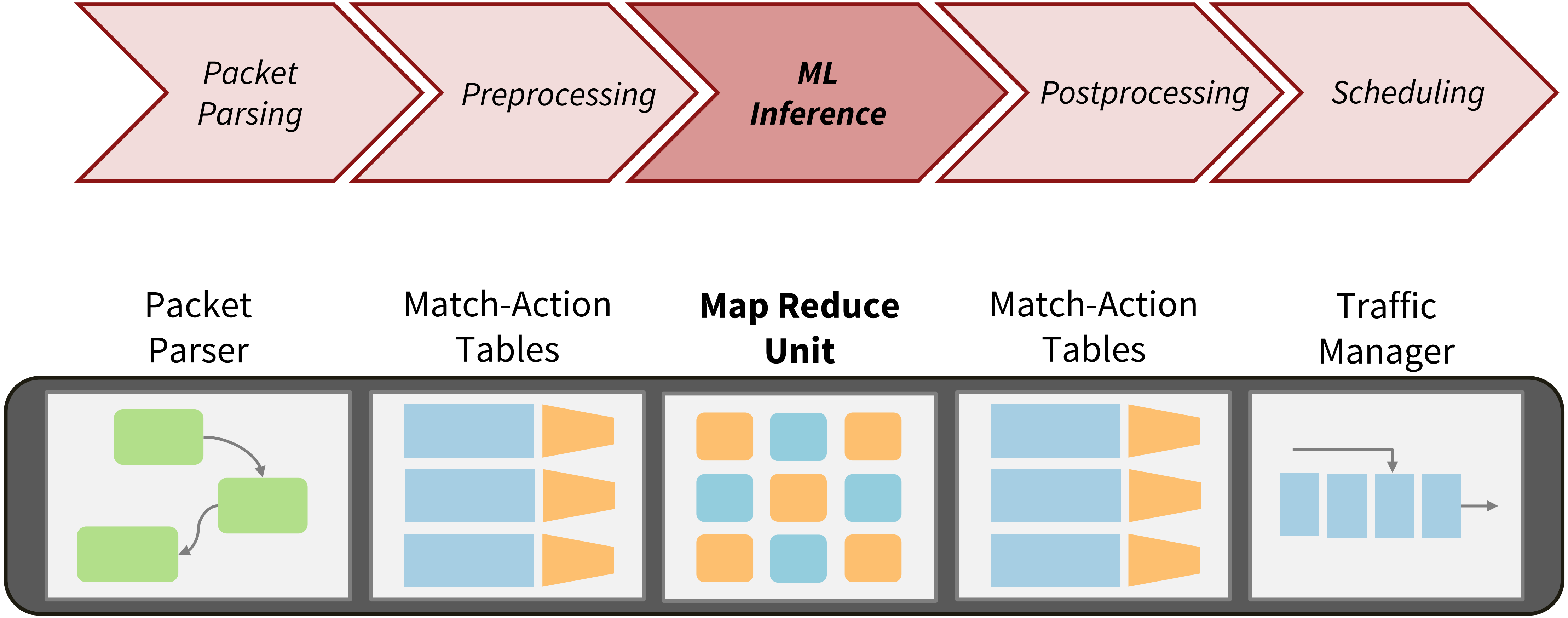
Congestion Control
Active Queue Management

Queue Reconstruction
QoS Correlation

A Taurus network introduces ML for management



The Taurus switch pipeline



Taurus achieves line-rate ML inference

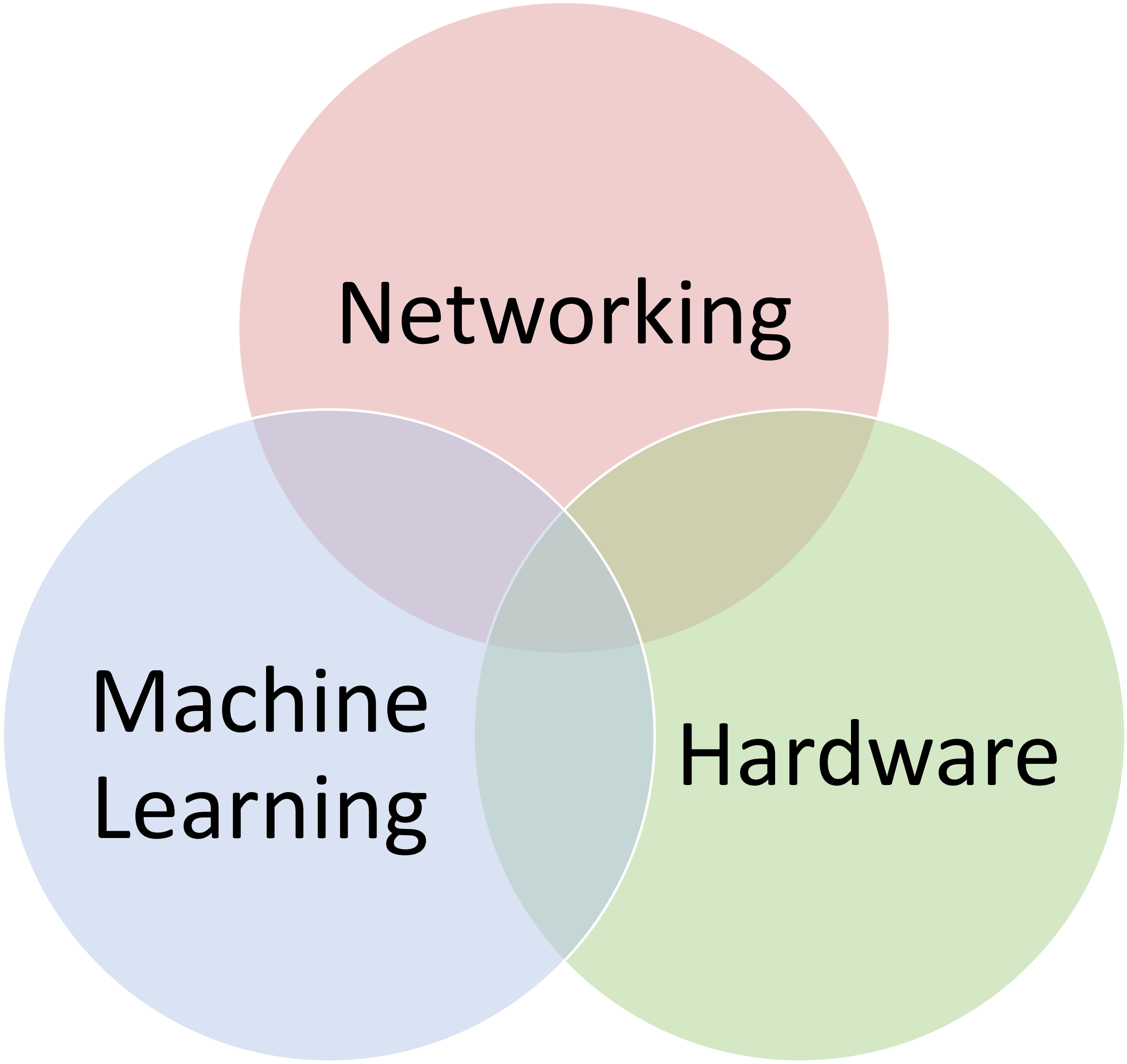
- Robustness and performance of the network is determined by ***quality and speed of reaction***

- ML inference should happen ***per-packet*** in the ***dataplane***

- ***Taurus*** enables line-rate, per-packet ML inference in the dataplane

Taurus: a data plane architecture for per-packet ML
(ASPLOS '22) https://dl.acm.org/doi/10.1145/3503222.3507726

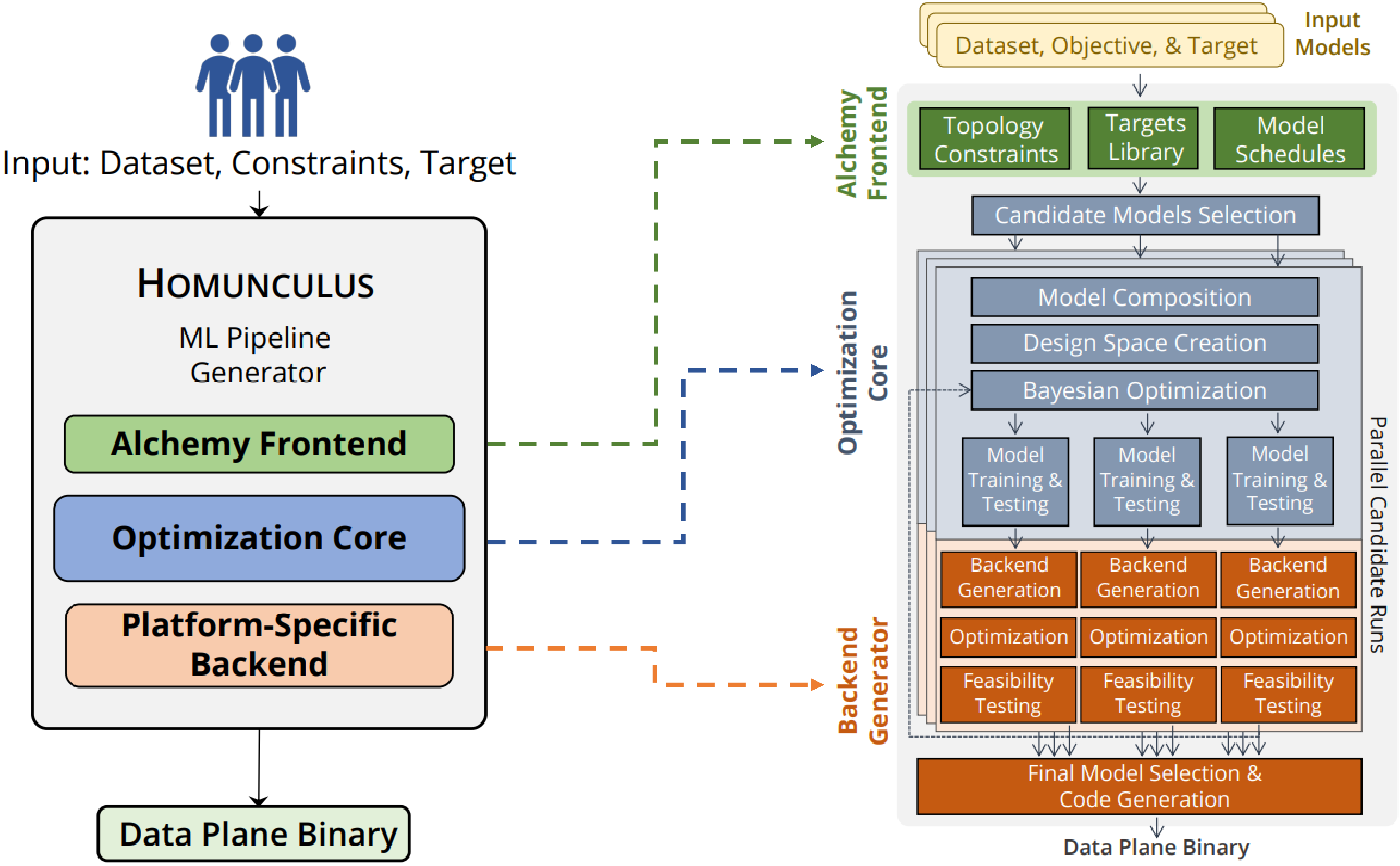
How do we program Taurus?



Homunculus: a framework for dataplane model generation

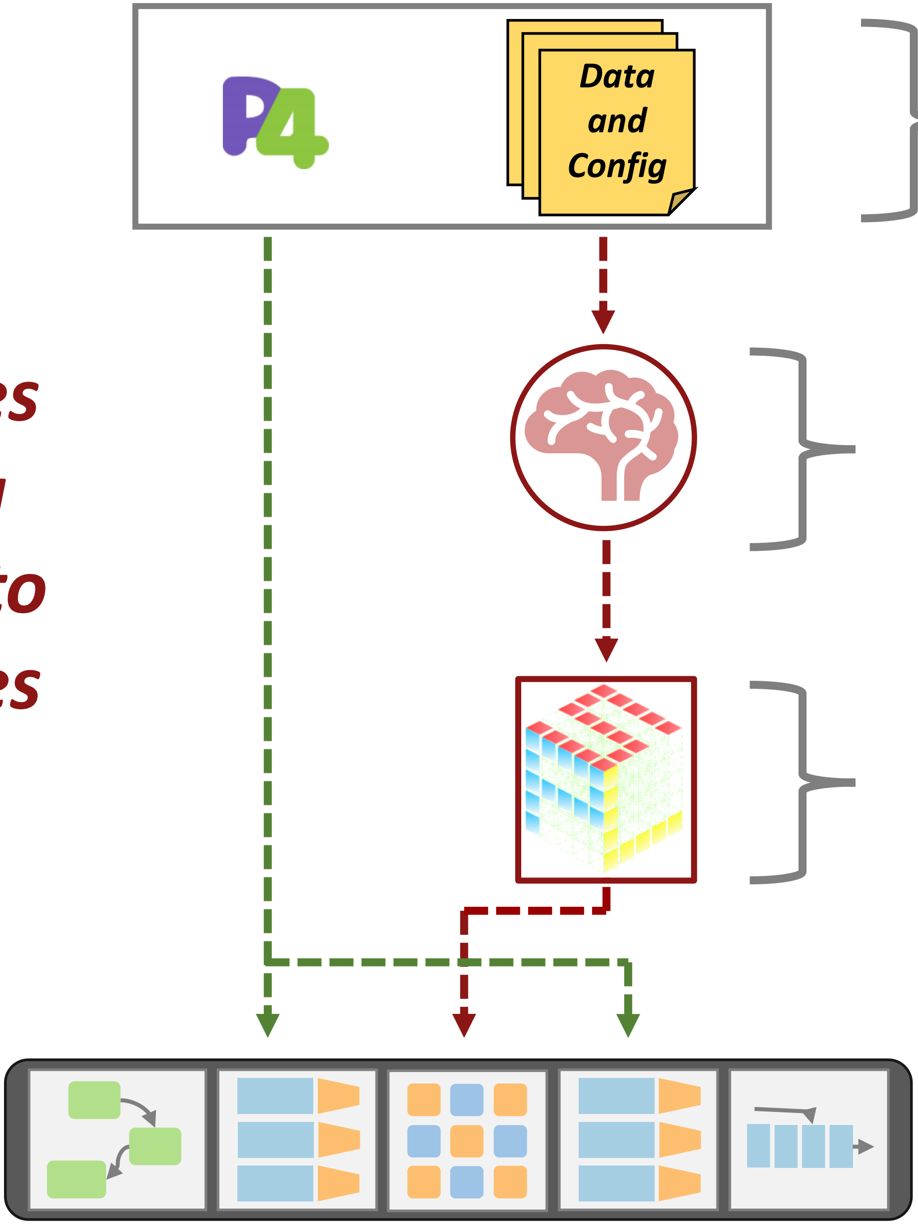
- Provide **high-level directives** to express user intent
- Take **network and resource constraints** into account when building ML models
- **Generate binaries** for different dataplane devices with optimized ML models

Homunculus architecture



Homunculus overview

Homunculus provides network operators a high-level compiler to program ML switches



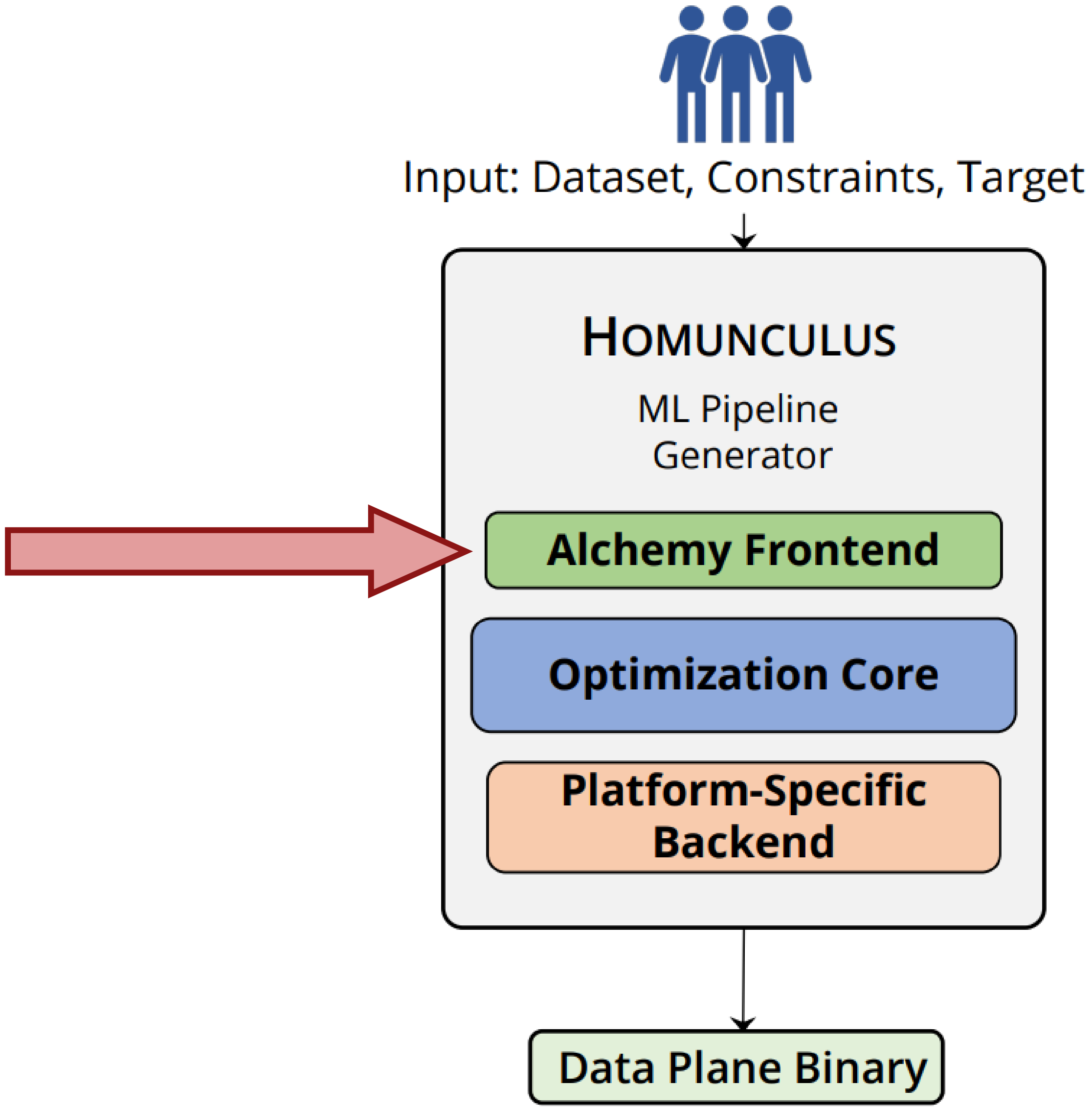
User specifies a **dataset, environment configuration** alongside a **network DSL** *(P4)*

ML models are generated through **Bayesian optimization** *(HyperMapper)* and **ML frameworks** *(Keras)* to meet optimization requirements

Platform-specific backends *(Spatial, Tungsten)* test resource usage and performance

P4 and Spatial are installed on a **switch** *(Taurus)*

Homunculus frontend: Alchemy



Homunculus frontend: Alchemy

```python
import homunculus
from homunculus.alchemy import
    DataLoader, Model, Platforms
import ad_loader

@DataLoader # training data loader definition
def wrapper_func():
    tnx, tny = ad_loader.load_from_file(
        "train_ad.csv")
    tsx, tsy = ad_loader.load_from_file(
        "test_ad.csv")
    return {
        "data": {"train": tnx, "test": tsx },
        "labels": {"train": tny, "test": tsy }}

# Specify the model of choice
model_spec = Model({
    "optimization_metric": ["f1"],
    "algorithm": ["dnn"],
    "name": "anomaly_detection",
    "data_loader": wrapper_func })

# Load platform
platform = Platforms.Taurus()
platform.constrain(
    "performance": {
        "throughput": 1, # GPkt/s
        "latency": 500 },    # ns
    "resources": { "rows": 16, "cols": 16 })

# Schedule model and generate code
platform.schedule(model_spec)
homunculus.generate(platform)
```

Alchemy is a Python library

Homunculus frontend: Alchemy

```python
import homunculus
from homunculus.alchemy import
    DataLoader, Model, Platforms
import ad_loader

@DataLoader # training data loader definition
def wrapper_func():
    tnx, tny = ad_loader.load_from_file(
        "train_ad.csv")
    tsx, tsy = ad_loader.load_from_file(
        "test_ad.csv")
    return {
        "data": {"train": tnx, "test": tsx },
        "labels": {"train": tny, "test": tsy }}

# Specify the model of choice
model_spec = Model({
    "optimization_metric": ["f1"],
    "algorithm": ["dnn"],
    "name": "anomaly_detection",
    "data_loader": wrapper_func })

# Load platform
platform = Platforms.Taurus()
platform.constrain(
    "performance": {
        "throughput": 1, # GPkt/s
        "latency": 500 },    # ns
    "resources": { "rows": 16, "cols": 16 })

# Schedule model and generate code
platform.schedule(model_spec)
homunculus.generate(platform)
```

Load data

Homunculus frontend: Alchemy

```python
import homunculus
from homunculus.alchemy import
    DataLoader, Model, Platforms
import ad_loader

@DataLoader # training data loader definition
def wrapper_func():
    tnx, tny = ad_loader.load_from_file(
        "train_ad.csv")
    tsx, tsy = ad_loader.load_from_file(
        "test_ad.csv")
    return {
        "data": {"train": tnx, "test": tsx },
        "labels": {"train": tny, "test": tsy }}

# Specify the model of choice
model_spec = Model({
    "optimization_metric": ["f1"],
    "algorithm": ["dnn"],
    "name": "anomaly_detection",
    "data_loader": wrapper_func })

# Load platform
platform = Platforms.Taurus()
platform.constrain(
    "performance": {
        "throughput": 1, # GPkt/s
        "latency": 500 },    # ns
    "resources": { "rows": 16, "cols": 16 })

# Schedule model and generate code
platform.schedule(model_spec)
homunculus.generate(platform)
```

Specify a model

Homunculus frontend: Alchemy

```
import homunculus
from homunculus.alchemy import
    DataLoader, Model, Platforms
import ad_loader

@DataLoader # training data loader definition
def wrapper_func():
    tnx, tny = ad_loader.load_from_file(
        "train_ad.csv")
    tsx, tsy = ad_loader.load_from_file(
        "test_ad.csv")
    return {
        "data": {"train": tnx, "test": tsx },
        "labels": {"train": tny, "test": tsy }}

# Specify the model of choice
model_spec = Model({
    "optimization_metric": ["f1"],
    "algorithm": ["dnn"],
    "name": "anomaly_detection",
    "data_loader": wrapper_func })

# Load platform
platform = Platforms.Taurus()
platform.constrain(
    "performance": {
        "throughput": 1, # GPkt/s
        "latency": 500 },    # ns
    "resources": { "rows": 16, "cols": 16 })

# Schedule model and generate code
platform.schedule(model_spec)
homunculus.generate(platform)
```

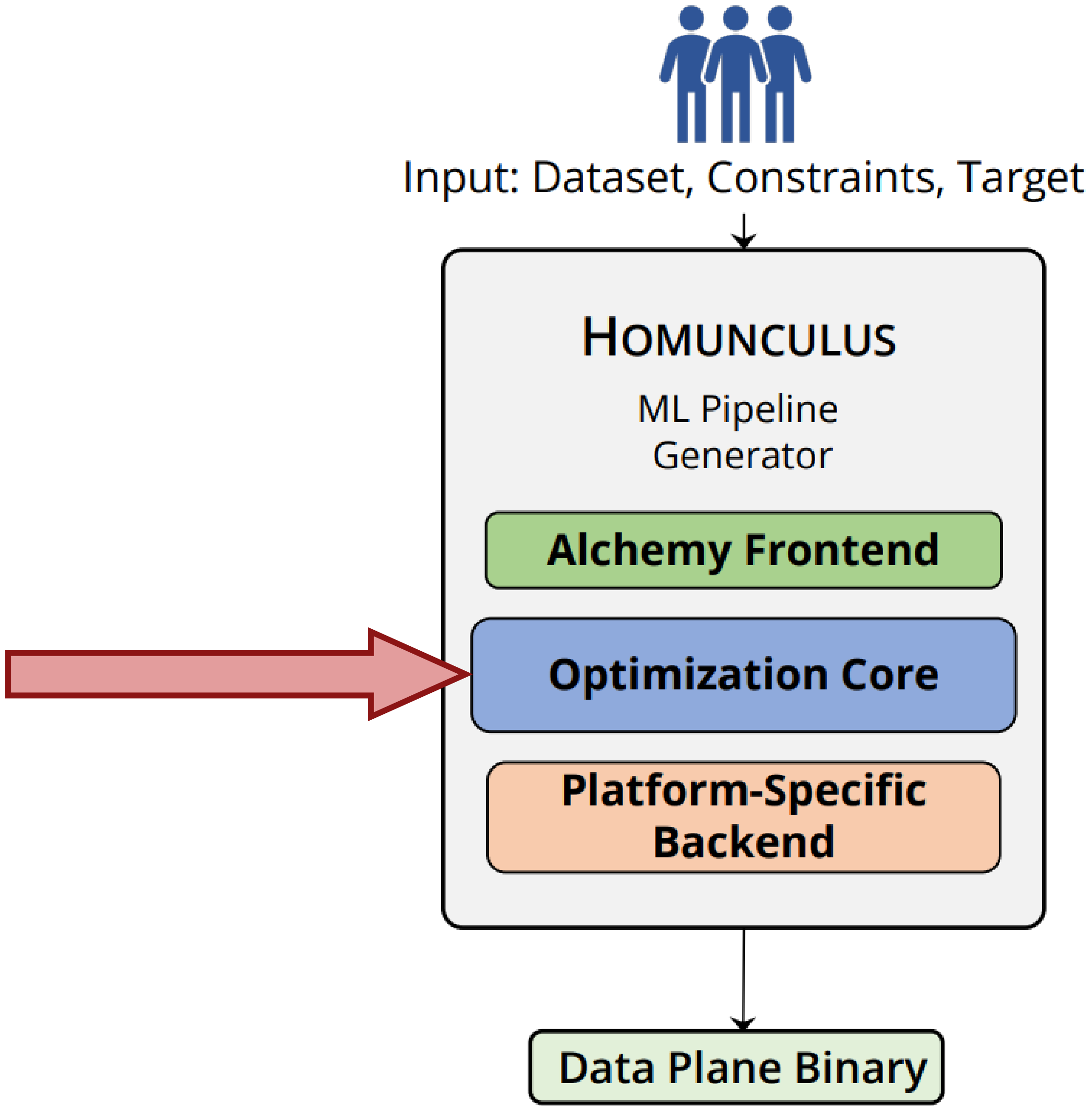
Constrain the platform

Homunculus frontend: Alchemy

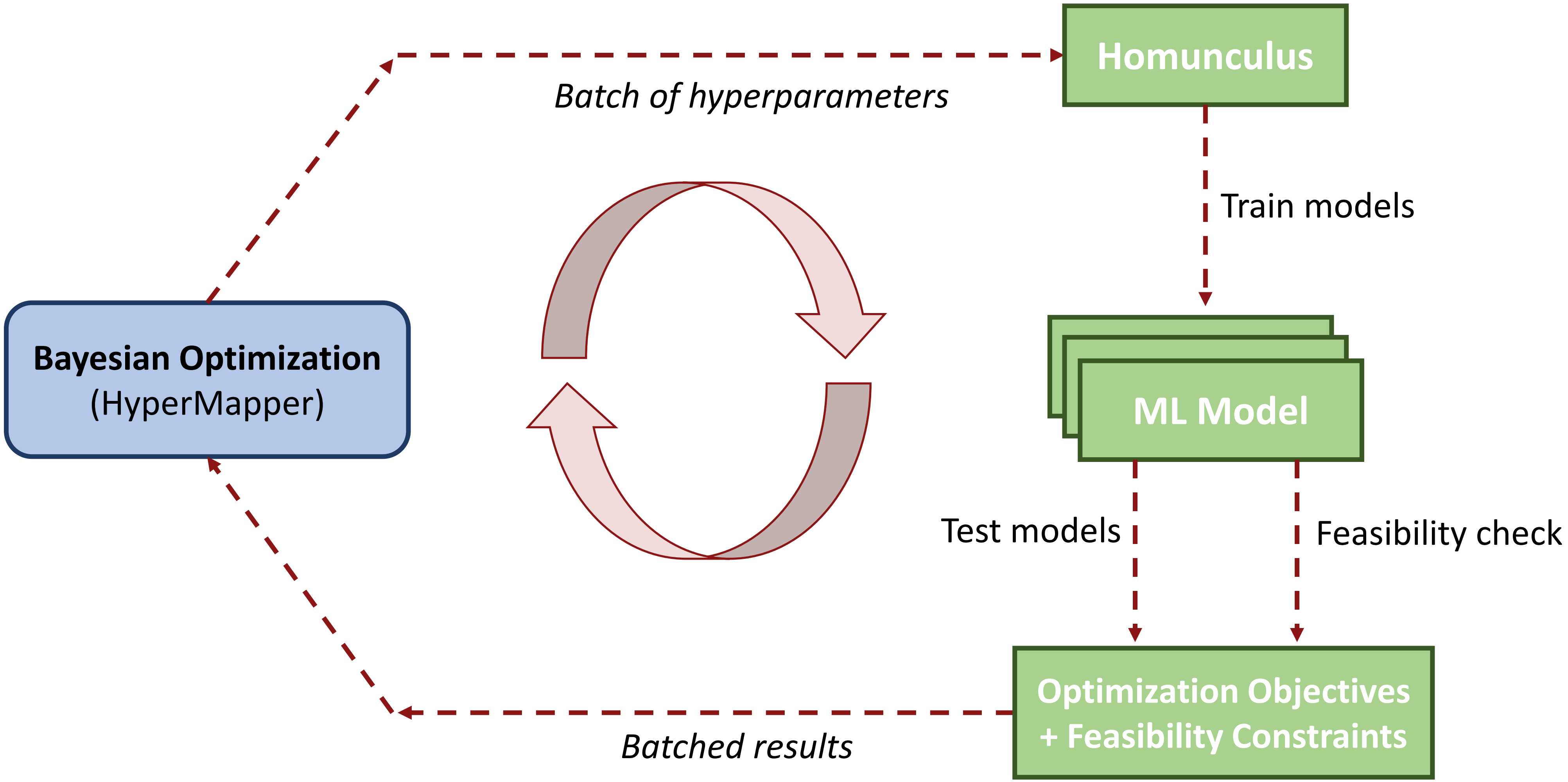
```python
import homunculus
from homunculus.alchemy import
    DataLoader, Model, Platforms
import ad_loader

@DataLoader # training data loader definition
def wrapper_func():
    tnx, tny = ad_loader.load_from_file(
        "train_ad.csv")
    tsx, tsy = ad_loader.load_from_file(
        "test_ad.csv")
    return {
        "data": {"train": tnx, "test": tsx },
        "labels": {"train": tny, "test": tsy }}

# Specify the model of choice
model_spec = Model({
    "optimization_metric": ["f1"],
    "algorithm": ["dnn"],
    "name": "anomaly_detection",
    "data_loader": wrapper_func })

# Load platform
platform = Platforms.Taurus()
platform.constrain(
    "performance": {
        "throughput": 1, # GPkt/s
        "latency": 500 },    # ns
    "resources": { "rows": 16, "cols": 16 })

# Schedule model and generate code
platform.schedule(model_spec)
homunculus.generate(platform)
```

Generate binary

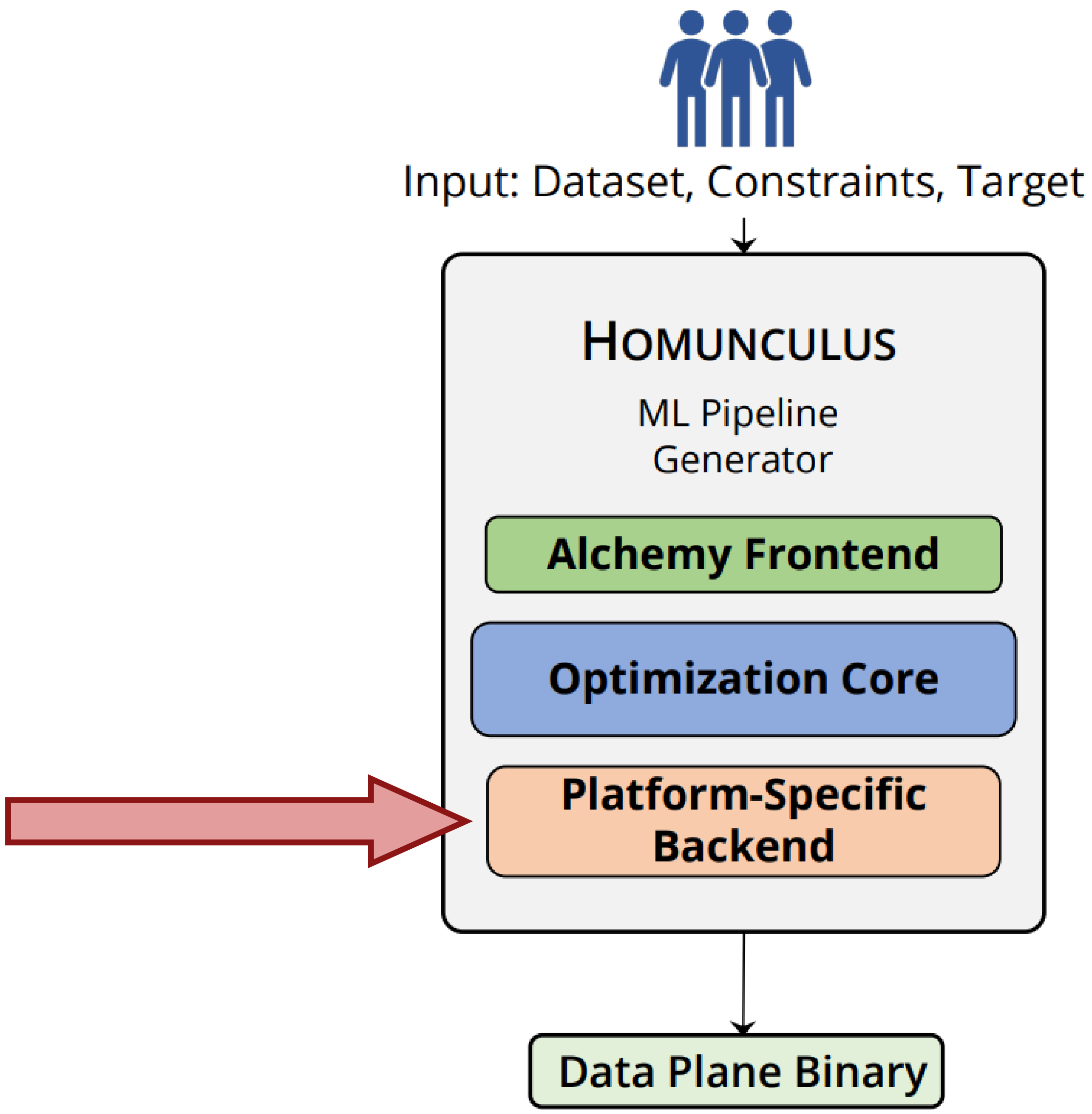
Homunculus Optimization Core



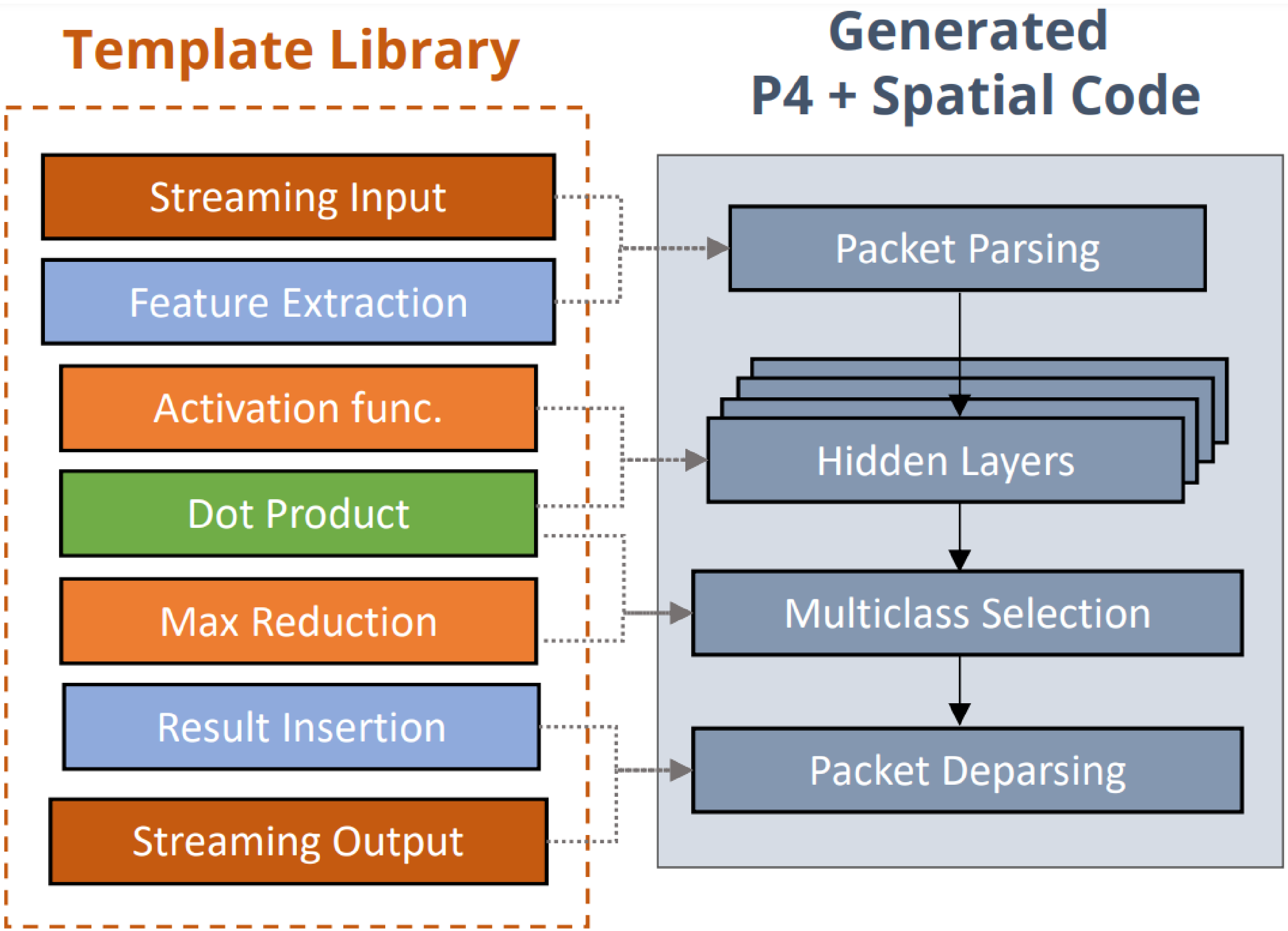
Homunculus Optimization Core



Homunculus Backend



Homunculus Backend Template Library



Homunculus results

- F1 Score improvements in anomaly detection (**AD**), traffic classification (**TC**), botnet detection (**BD**)

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

Homunculus makes it easy to generate high-quality models

- Provides a **high-level interface** for users to express intent

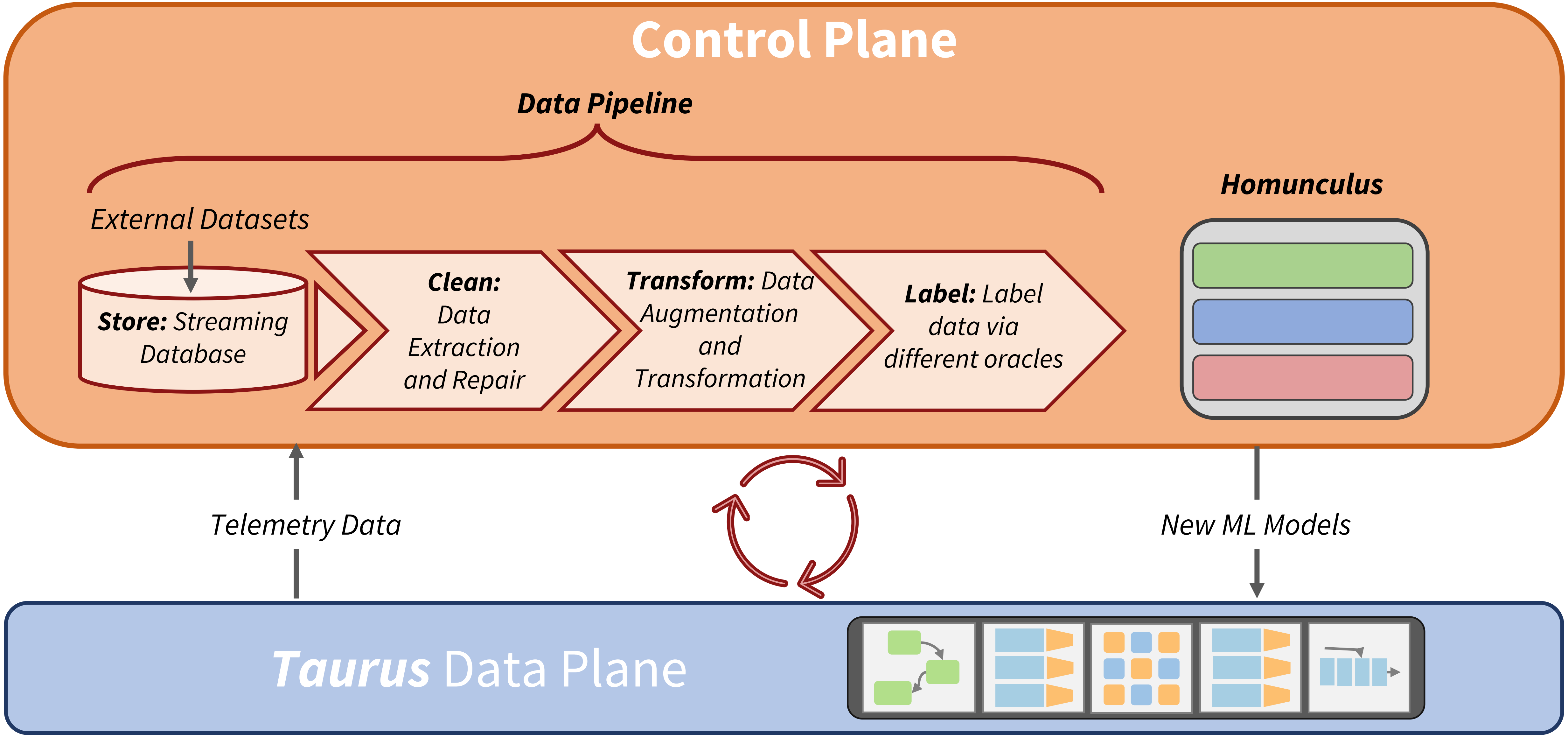
- Uses **network and resource constraints** to traverse AutoML search space

- **Generate binaries** for different dataplane devices with optimized ML models

Homunculus: Auto-Generating Efficient Data-Plane ML Pipelines for Datacenter Networks https://arxiv.org/pdf/2206.05592

How do we supply data to Homunculus?

Data Pipeline supplies high quality data to Homunculus



Questions?

Tushar Swamy

tswamy@stanford.edu

Taurus: https://dl.acm.org/doi/10.1145/3503222.3507726

Enable ML inference in the dataplane

Run line-rate, per-packet operation

Homunculus: https://arxiv.org/pdf/2206.05592

Allows automated construction of ML models

More accurate than hand-tuned models

Try it out! https://gitlab.com/dataplane-ai/taurus