# Everyone Else is Using ML, Why Aren't We?

## Widespread adoption in other domains

AlphaGo AlphaFold ChatGPT CoPilot AlphaProof

## And learned systems

**Cluster Scheduling**
[Decima (SIGCOMM'20)]

**Query Optimization**
[Neo (VLDB'19), Bao (SIGMOD'22)]

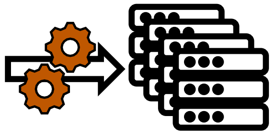
**Configuration Tuning**
[SelfTune (NSDI'23), MLOS (VLDB'24)]

# Everyone Else is Using ML, Why Aren't We?

## Widespread adoption in other domains

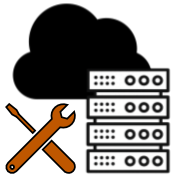
AlphaGo AlphaFold ChatGPT CoPilot AlphaProof

## And learned systems



**Cluster Scheduling**
[Decima (SIGCOMM'20)]

**Query Optimization**
[Neo (VLDB'19), Bao (SIGMOD'22)]



**Configuration Tuning**
[SelfTune (NSDI'23), MLOS (VLDB'24)]

## Does using ML in the OS make sense?

# Everyone Else is Using ML, Why Aren't We?

## Widespread adoption in other domains

AlphaGo AlphaFold ChatGPT CoPilot AlphaProof

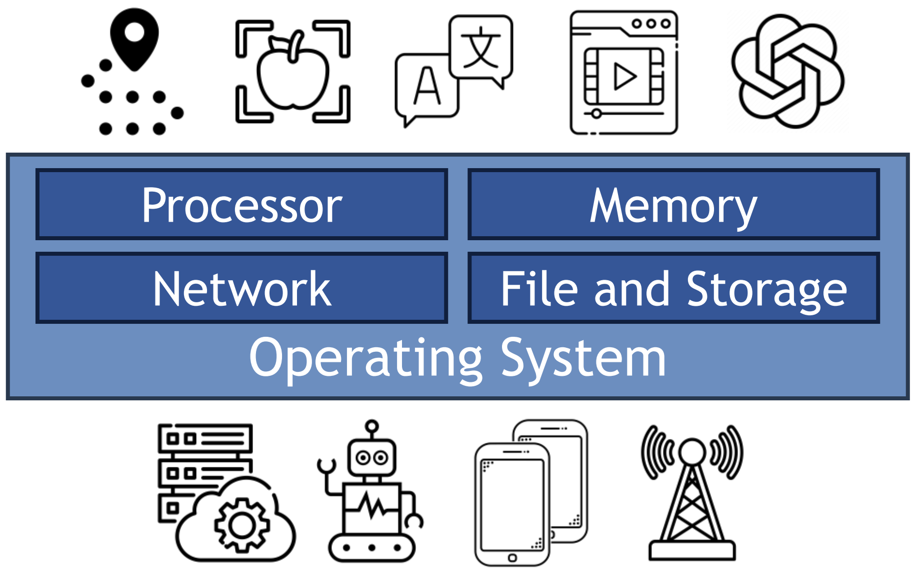
## And learned systems

**Cluster Scheduling**
[Decima (SIGCOMM'20)]

**Query Optimization**
[Neo (VLDB'19), Bao (SIGMOD'22)]

**Configuration Tuning**
[SelfTune (NSDI'23), MLOS (VLDB'24)]

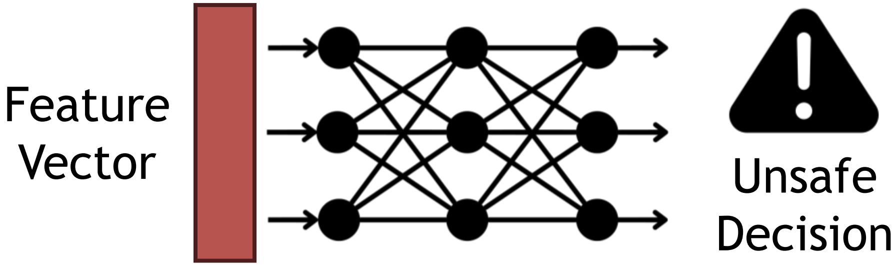
## Does using ML in the OS make sense?



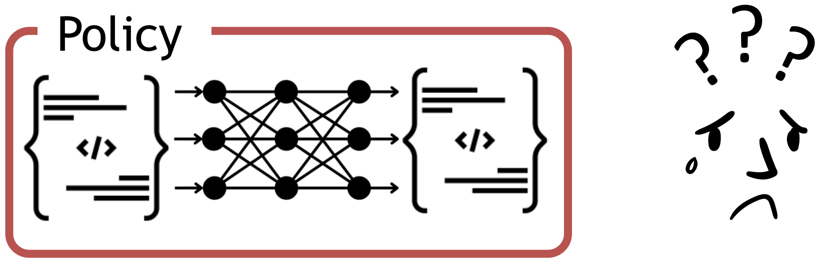

OS is subject to diverse applications and environments, necessitating **dynamic and adaptive policies**!

Exploit the capability of ML of using rich features and take predictive actions!

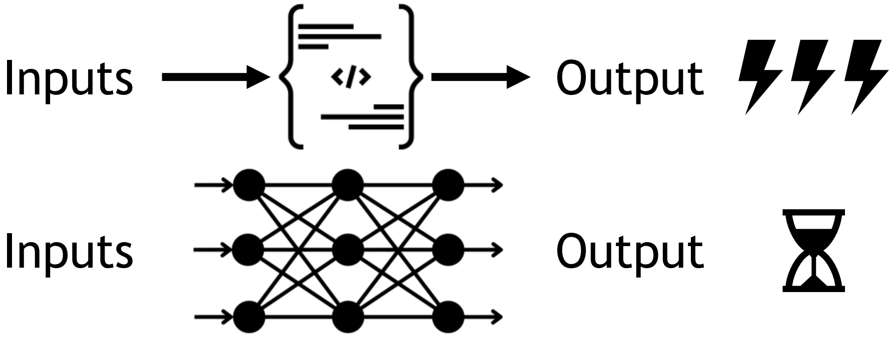
# “Keep Your Damn Models Out of My Kernel!”




Unsafe decisions because of incomplete training/unseen inputs

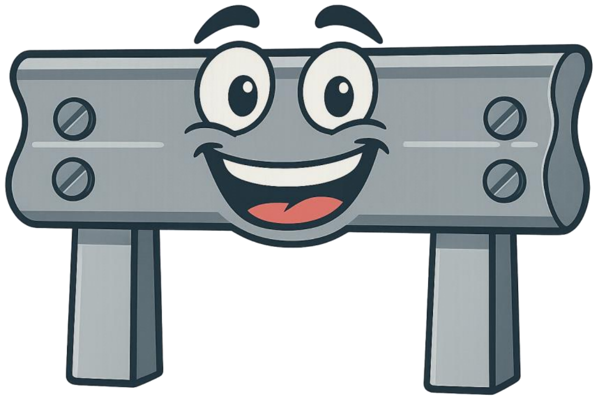



Opaque decisions undermine reproducibility and compromise security.




Performance overheads of using learning-based policies

LDOS



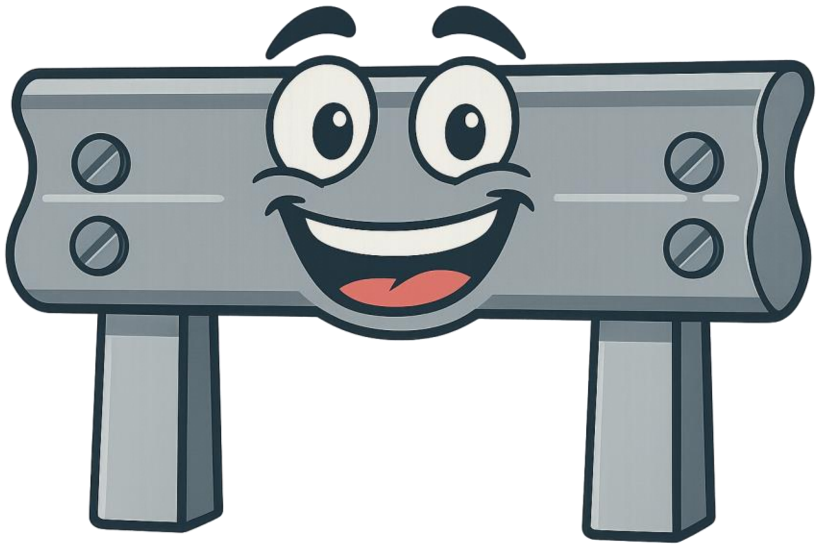
# How I learned to stop worrying and love learned OS policies

**Divyanshu Saxena***, Jiayi Chen*, Sujay Yadalam, Yeonju Ro, Rohit Dwivedula, Eric Campbell, Aditya Akella, Christopher Rossbach, Michael Swift

TEXAS

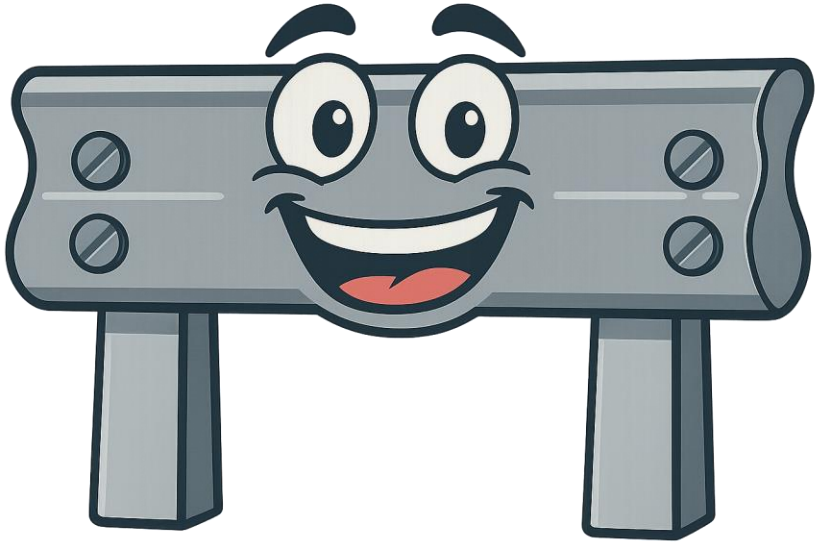
WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

# Guardrails for OS Policies



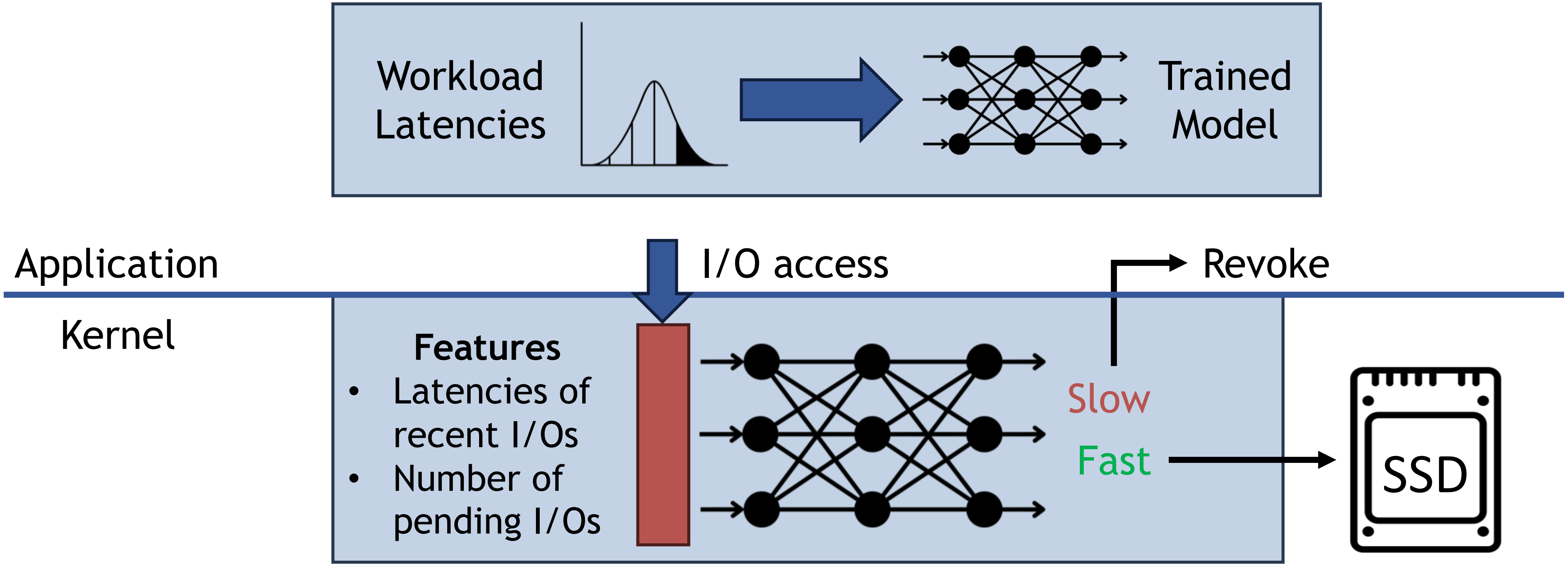
# Guardrails for OS Policies

Enable learned policies where beneficial and avoid catastrophic outcomes.

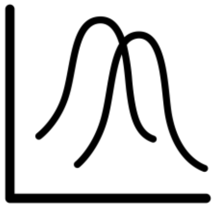
# A Guardrail Case Study – I/O Latency Predictor

- **Task:** Predict whether an I/O access will be slow or fast [LinnOS (OSDI'20)]
- Trained using the latency distribution of current workload.



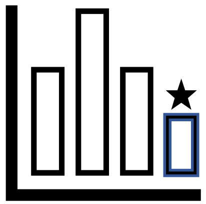
# Detecting When Things Go Wrong

Detect potential issues by monitoring inputs, outputs and system behavior.



$f \longrightarrow$ Slow

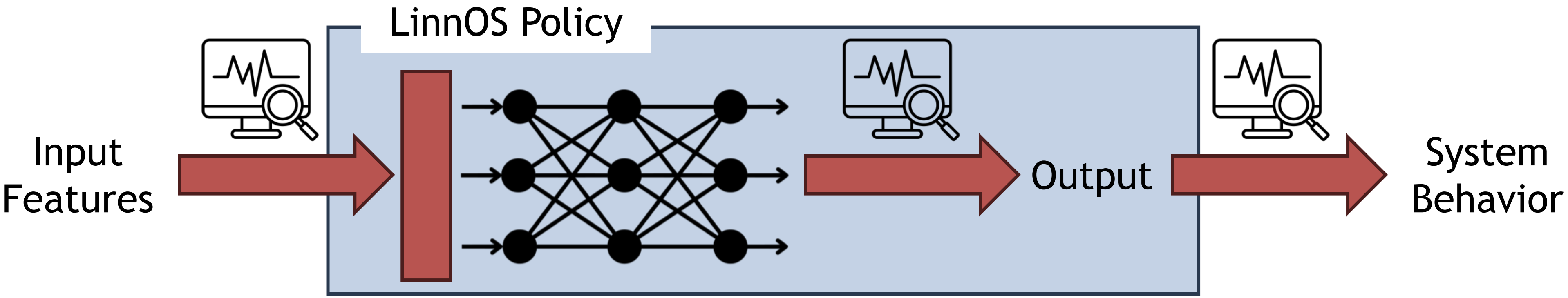
$f + \epsilon \longrightarrow$ Fast?



**Out-of-distribution inputs:**
Workload changes can change the fast/slow threshold.

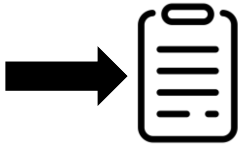
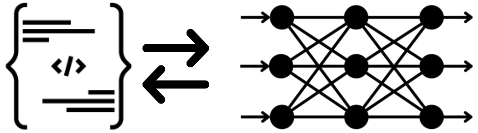
**Poor decisions because of noise:**
Model may yield different outputs for similar inputs.

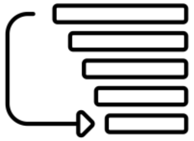
**Poor end-to-end performance:**
Gains of good decisions may be negated by model overhead.



# Recovering from Undesirable Outcomes

Simple detection is not enough ⇒ Automatic recovery when problems arise.

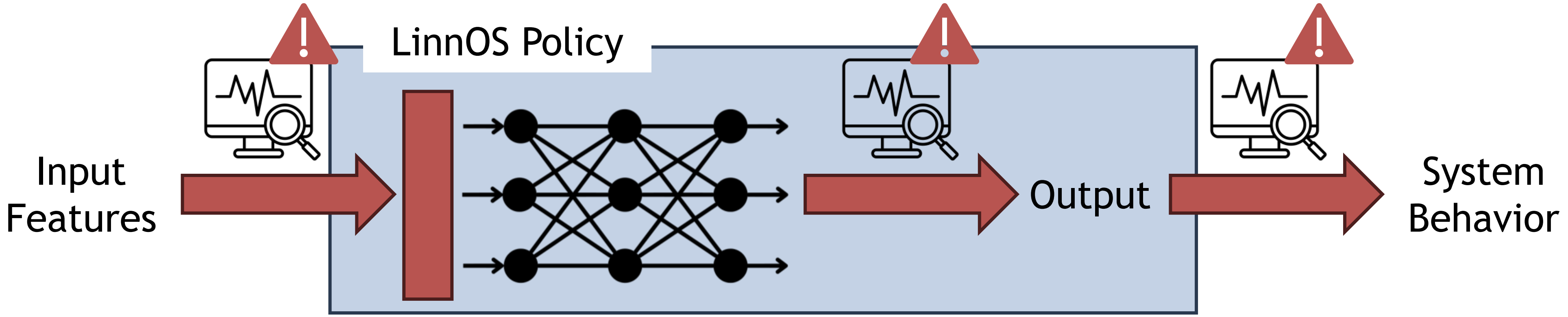


**Report** to log on an incorrect prediction.

**Replace** model with a hedging-based heuristic.

**Retrain** the latency prediction model.

**Deprioritize** kswapd to reduce I/O requests.



# The Guardrail Abstraction

**Example Properties:**
- In-distribution inputs
- Robustness to noise
- Better performance than default

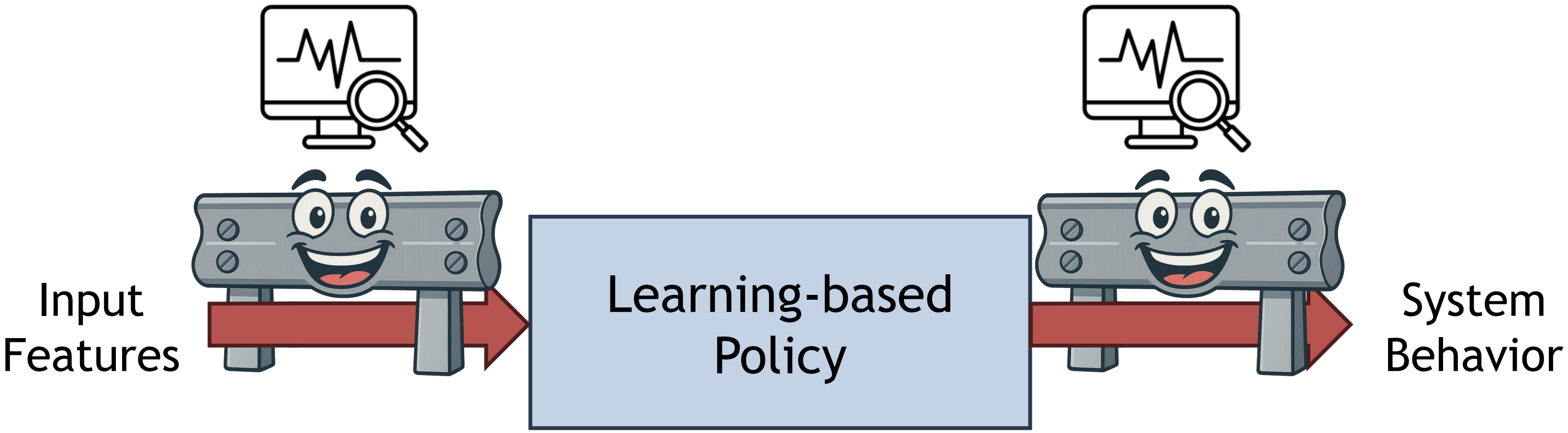
**Property**

Description of desired behaviors, constraints and invariants.

**Action**

Prescriptions for system responses when a property is violated

**Example Actions:**
- Report
- Replace
- Retrain
- Deprioritize



# The Guardrail Abstraction

**Example Properties:**
- In-distribution inputs
- Robustness to noise
- Better performance than default

**Property**

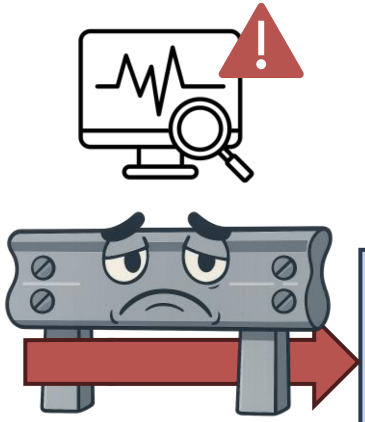
Description of desired behaviors, constraints and invariants.

**Action**

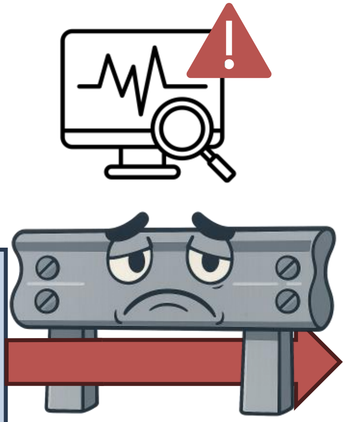
Prescriptions for system responses when a property is violated

**Example Actions:**
- Report
- Replace
- Retrain
- Deprioritize

Input Features



Learning-based Policy



System Behavior

# The Guardrail Abstraction

**Example Properties:**
- In-distribution inputs
- Robustness to noise
- Better performance than default

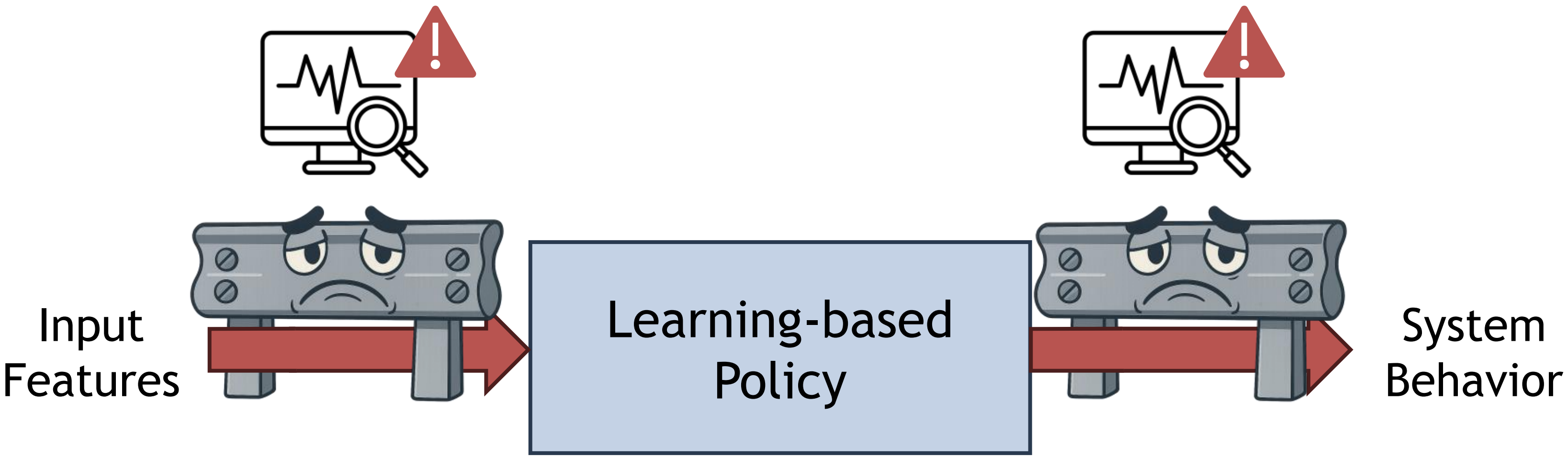
**Property**

Description of desired behaviors, constraints and invariants.

**Action**

Prescriptions for system responses when a property is violated

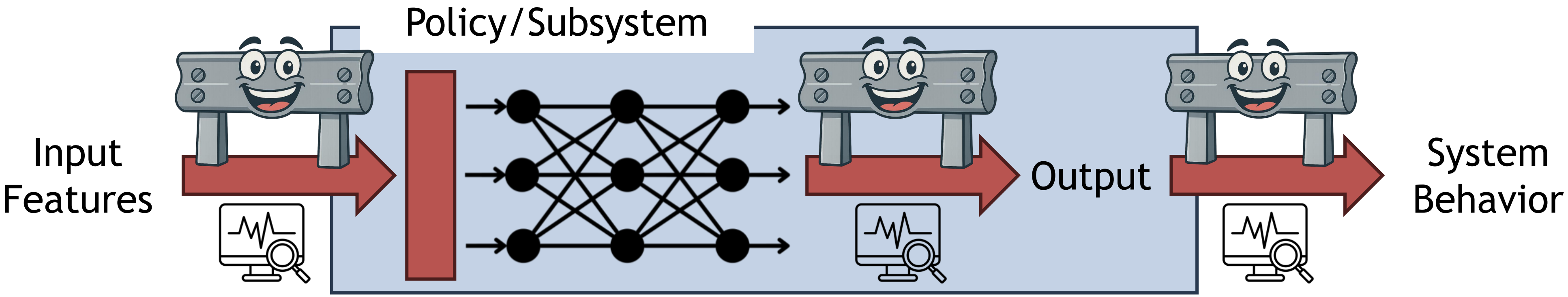
**Example Actions:**
- Report
- Replace
- Retrain
- Deprioritize



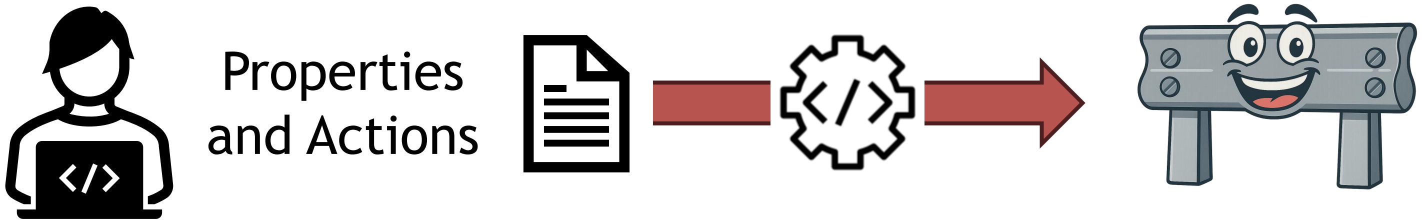

Need introspective support to monitor properties at run time and take corrective actions.

# Providing Support for OS Guardrails

High-level interface to
specify guardrails



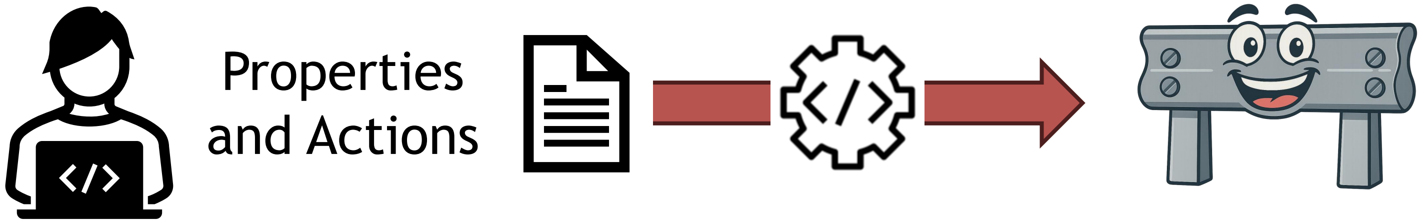
# Guardrail Interface – Specifying Properties




**Interface Grammar**

```
⟨Guardrail⟩ ::= ⟨Property⟩ ⟨Action⟩+
⟨Property⟩  ::= ⟨Trigger⟩+ ⟨Rule⟩+
⟨Trigger⟩   ::= TIMER | FUNCTION
⟨Rule⟩      ::= ⟨Expression⟩
```

# Guardrail Interface – Specifying Properties




**Interface Grammar**

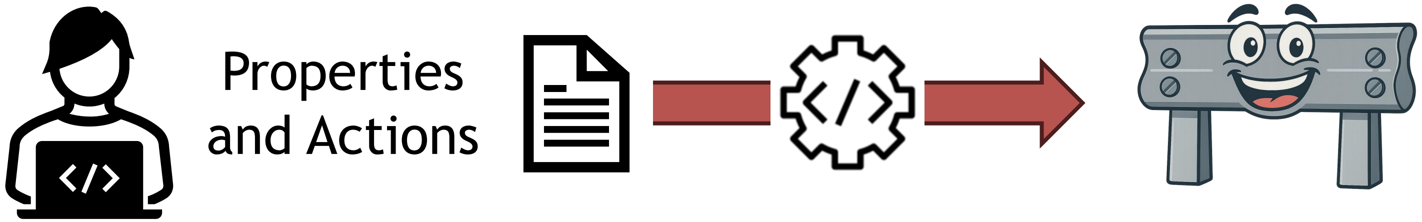
```
⟨Guardrail⟩ ::= ⟨Property⟩ ⟨Action⟩+
⟨Property⟩  ::= ⟨Trigger⟩+ ⟨Rule⟩+
⟨Trigger⟩   ::= TIMER | FUNCTION
⟨Rule⟩      ::= ⟨Expression⟩
```

**In-distribution inputs:**
At every model invocation, check if $input \sim D_{train}$

**Robustness to noise:**
At every model invocation, check if $M(input) \approx M(input + \delta)$

**Better performance than default:**
At 10 second intervals, check if $Perf(M) > Perf(baseline)$

# Guardrail Interface – Specifying Properties




**Interface Grammar**

```
⟨Guardrail⟩ ::= ⟨Property⟩ ⟨Action⟩+
⟨Property⟩  ::= ⟨Trigger⟩+ ⟨Rule⟩+
⟨Trigger⟩   ::= TIMER | FUNCTION
⟨Rule⟩      ::= ⟨Expression⟩
```

**In-distribution inputs:**
At every model invocation, check if $input \sim D_{train}$

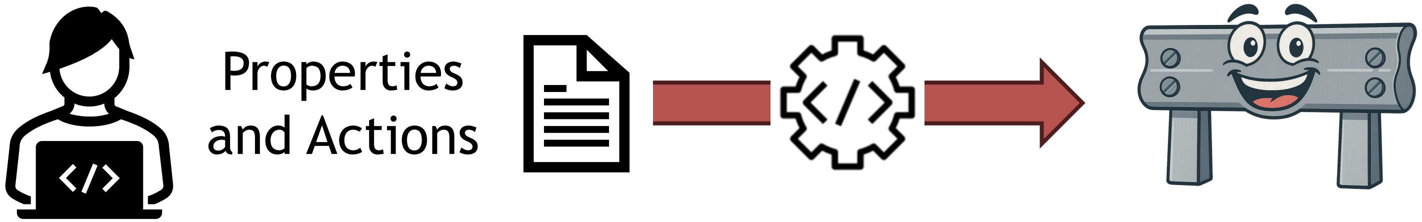
**Robustness to noise:**
At every model invocation, check if $M(input) \approx M(input + \delta)$

**Better performance than default:**
At 10 second intervals, check if $Perf(M) > Perf(baseline)$

RULE

TRIGGER

# Guardrail Interface – Specifying Actions




**Interface Grammar**

```
⟨Guardrail⟩ ::= ⟨Property⟩ ⟨Action⟩+
⟨Property⟩  ::= ⟨Trigger⟩+ ⟨Rule⟩+
⟨Trigger⟩   ::= TIMER | FUNCTION
⟨Rule⟩      ::= ⟨Expression⟩
⟨Action⟩    ::= REPORT | REPLACE | RETRAIN |
                DEPRIORITIZE | <Expression>
```

**Report to a log**
$Report(state, log)$

**Replace with a heuristic**
$Replace(model, baseline)$

**Retrain the model**
$Retrain(model, inputs)$

**Deprioritize tasks**
$Deprioritize(functions)$

# Guardrail State Store

- Rich properties and actions may require states, such as:
  - States available in the learned policy,
  - States tracked by the rule, e.g., counters, aggregates, etc.
  - System metrics, e.g., CPU utilization.

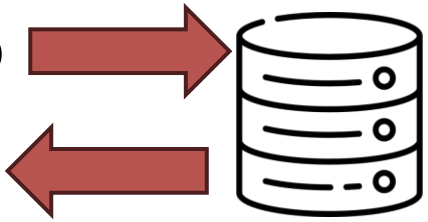
**Interface Grammar**

```
⟨Guardrail⟩ ::= ⟨Property⟩ ⟨Action⟩+
⟨Property⟩  ::= ⟨Trigger⟩+ ⟨Rule⟩+
⟨Trigger⟩   ::= TIMER | FUNCTION
⟨Rule⟩      ::= ⟨Expression⟩
⟨Action⟩    ::= REPORT | REPLACE | RETRAIN |
                DEPRIORITIZE | <Expression>
```

**A lightweight, global state store**

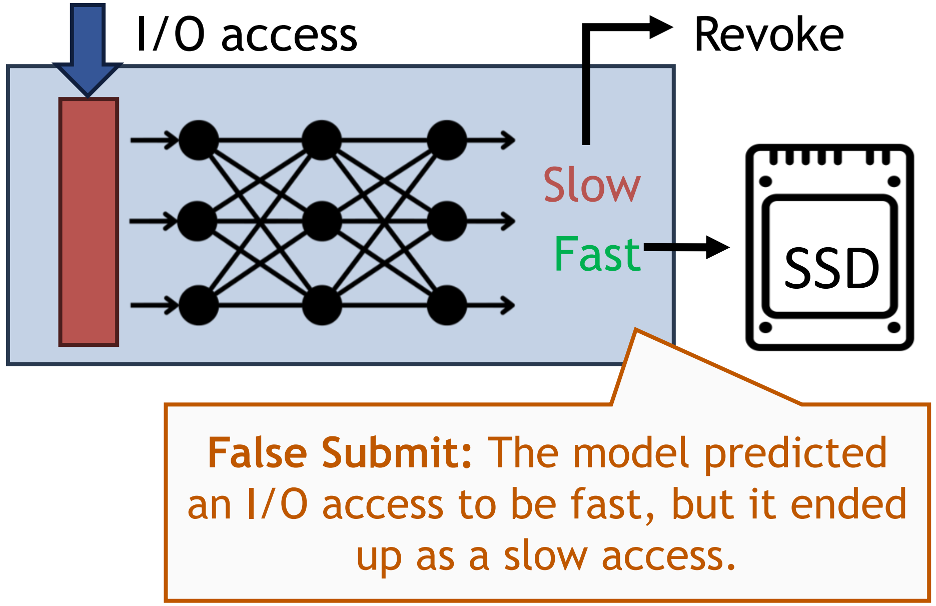
SAVE(key, value)

LOAD(key)



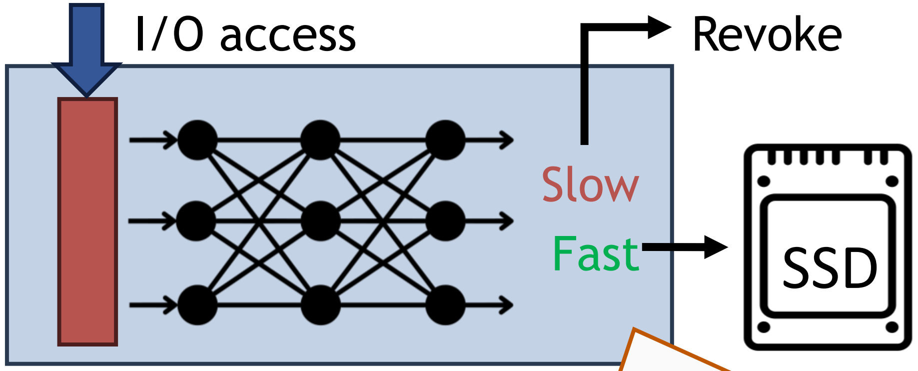
# How Guardrails May Help in Practice

- **Target policy**: I/O device latency predictor in LinnOS (OSDI'20)
- **Property**:
  - Rule: False submits should not be greater than 5%
  - Trigger: Periodically, every 1 second
- **Action**: Fallback to the default kernel policy.



# How Guardrails May Help in Practice

- **Target policy:** I/O device latency predictor in LinnOS (OSDI'20)
- **Property:**
  - Rule: False submits should not be greater than 5%
  - Trigger: Periodically, every 1 second
- **Action:** Fallback to the default kernel policy.

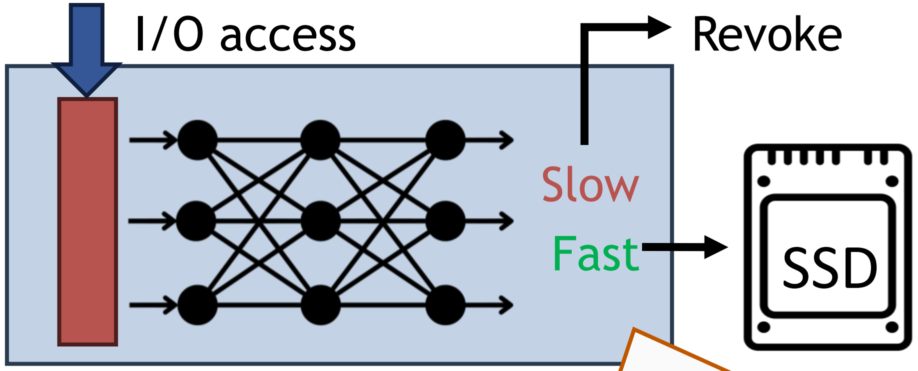



**False Submit:** The model predicted an I/O access to be fast, but it ended up as a slow access.

```
guardrail low-false-submit {
  trigger: { TIMER(1) },
  rule: { false_submit_rate <= 0.05 },
  action: { ml_enabled = false }
}
```

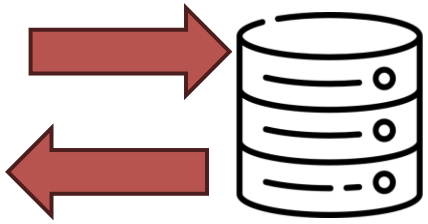
# How Guardrails May Help in Practice

- **Target policy:** I/O device latency predictor in LinnOS (OSDI'20)
- **Property:**
  - Rule: False submits should not be greater than 5%
  - Trigger: Periodically, every 1 second
- **Action:** Fallback to the default kernel policy.




**False Submit:** The model predicted an I/O access to be fast, but it ended up as a slow access.

```
guardrail low-false-submit {
  trigger: { TIMER(1) },
  rule: { false_submit_rate <= 0.05 },
  action: { ml_enabled = false }
}
```



Accessed via global state store

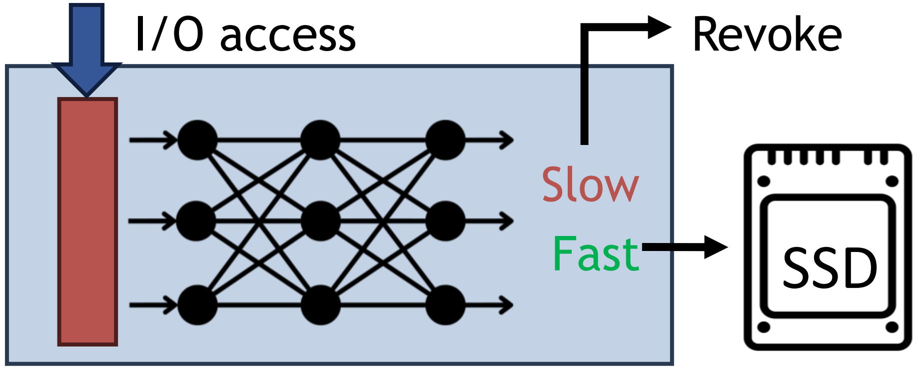
# How Guardrails May Help in Practice

```
guardrail low-false-submit {
  trigger: { TIMER(1) },
  rule: { false_submit_rate <= 0.05 },
  action: { ml_enabled = false }
}
```

**+**

Changes inside LinnOS code

```
if LOAD("ml_enabled") {
  // Use LinnOS predictions
}
...
// Update false submit rate
SAVE("false_submit_rate", false_submit_rate)
```



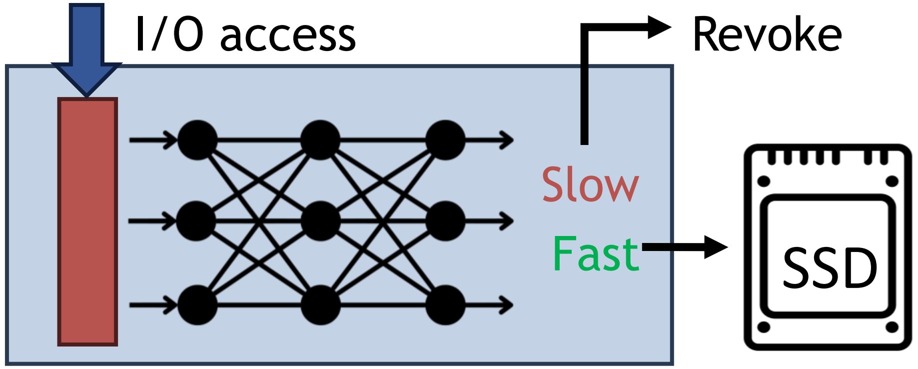
# How Guardrails May Help in Practice

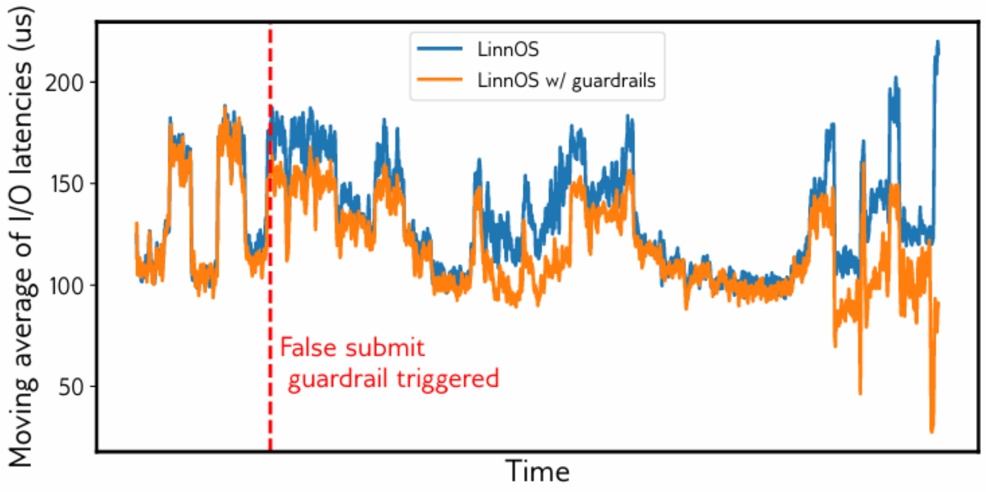
```
guardrail low-false-submit {
  trigger: { TIMER(1) },
  rule: { false_submit_rate <= 0.05 },
  action: { ml_enabled = false }
}
```

+

Changes inside LinnOS code

```
if LOAD("ml_enabled") {
  // Use LinnOS predictions
}
...
// Update false submit rate
SAVE("false_submit_rate", false_submit_rate)
```



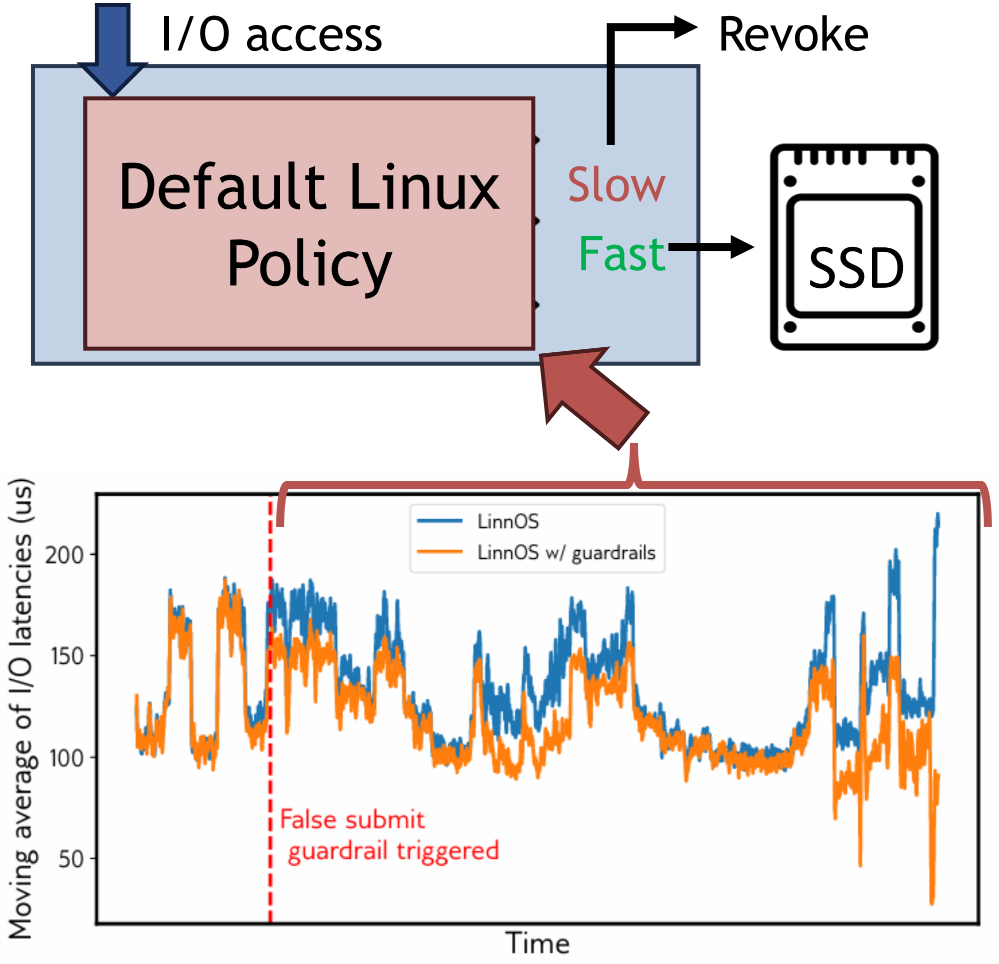
# How Guardrails May Help in Practice

```
guardrail low-false-submit {
  trigger: { TIMER(1) },
  rule: { false_submit_rate <= 0.05 },
  action: { ml_enabled = false }
}
```

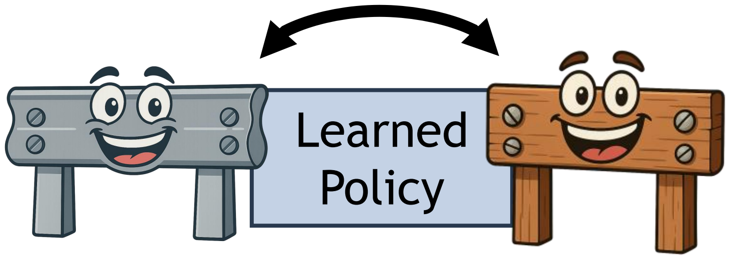
+

Changes inside LinnOS code

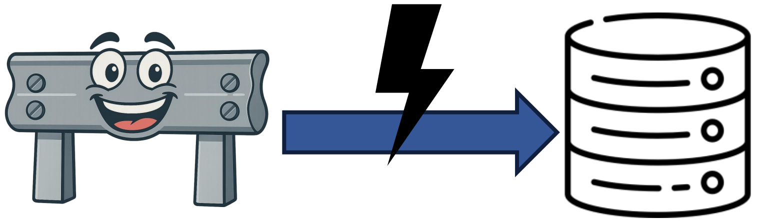
```
if LOAD("ml_enabled") {
  // Use LinnOS predictions
}
...
// Update false submit rate
SAVE("false_submit_rate", false_submit_rate)
```



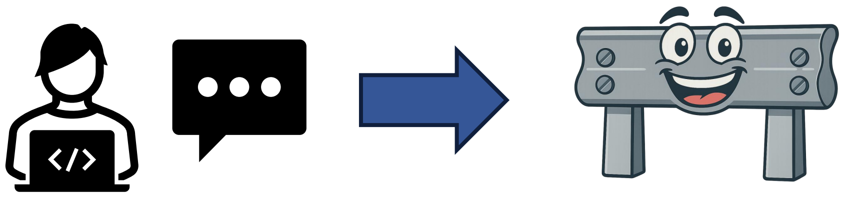
# Open Research Directions



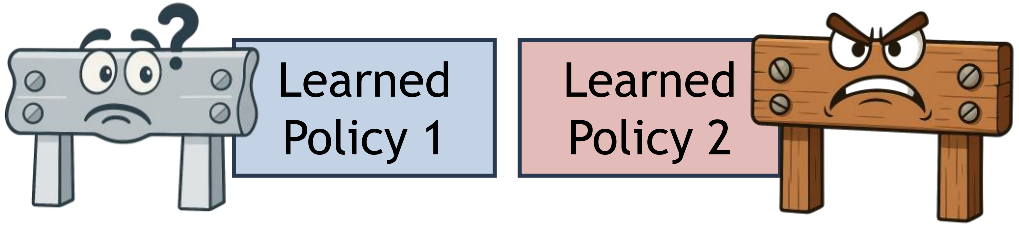

Evolve guardrails as properties or actions change.



Low-overhead property tracking, when using system-wide features



Seamless guardrail compilation for in-kernel enforcement.

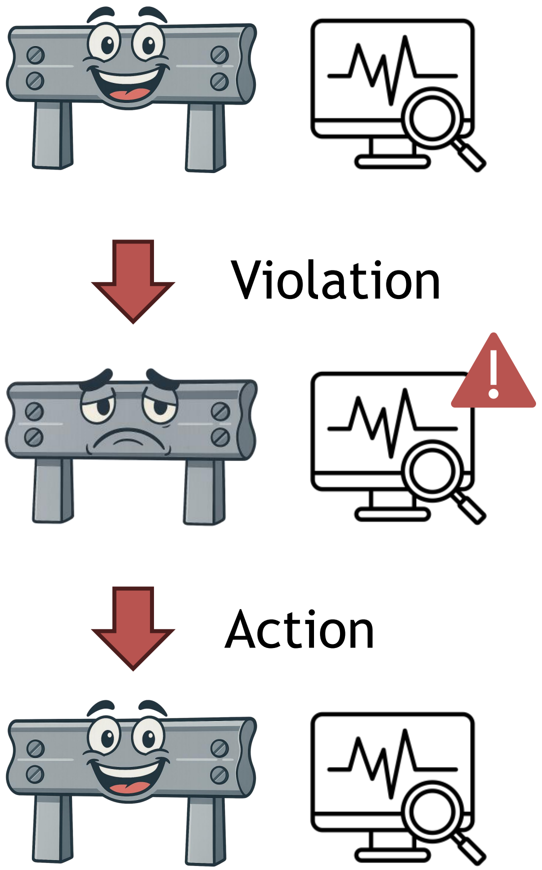



Managing interference among guardrails monitoring different properties

...and many more

# Summary

- We propose **OS Guardrails**—a framework that enables safe, effective, and high-impact use of learned policies.
- Guardrails track **adherence to a property** and allow **taking corrective actions** when violated.
- Preliminary experiments show **promising results** for the proposed interface and design.
- Opens **several avenues for research** on enabling low-overhead and flexible guardrails in the OS.




# Thank You!