Navigating Complex Machine Learning Challenges in Streaming Data

ECML Tutorial 2024

Heitor Murilo Gomes[1*], Marco Heyden[2]
Maroua Bahri[3,4]
https://heymarco.github.io/ecml24-streamingchallenges/

* Corresponding author: heitor.gomes@vuw.ac.nz

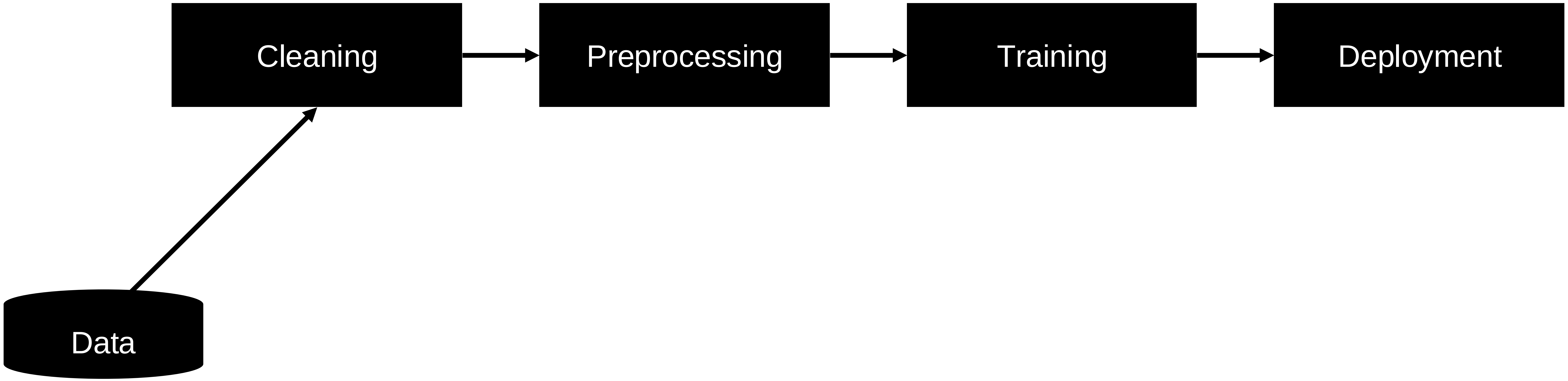
https://capymoa.org/

[1] Victoria University of Wellington, New Zealand, [2] KIT, Germany, [3] INRIA Paris, France,
[4] Sorbonne Université, France

Pipelines

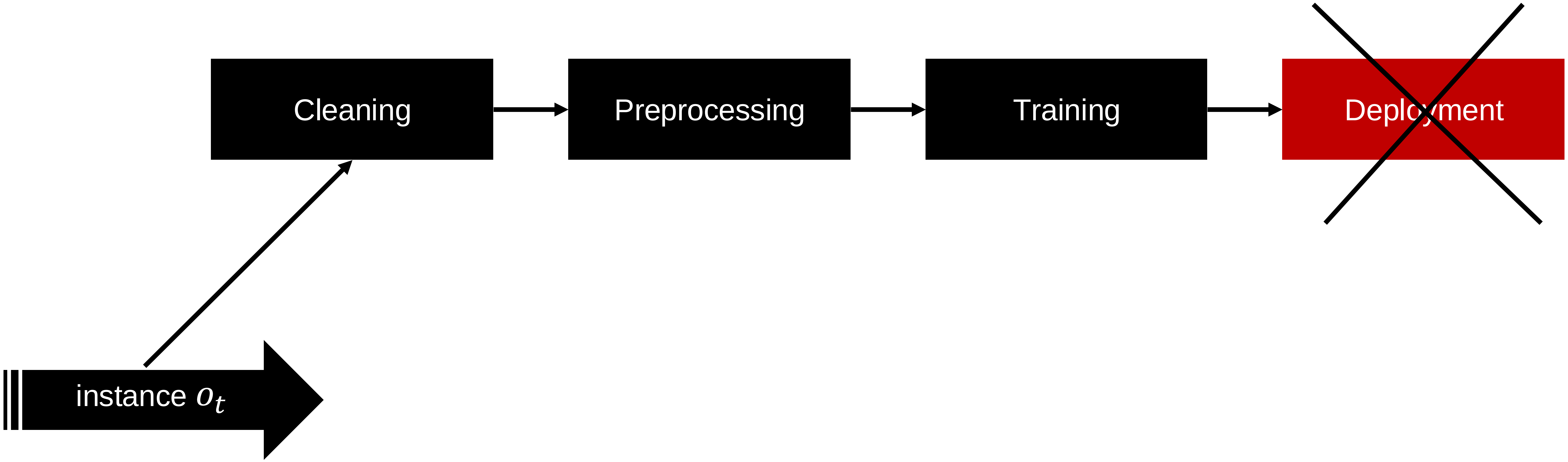
Traditional ML Pipelines

- Traditional ML Pipelines consist of well-defined, separate steps
- After training, model is deployed to make predictions



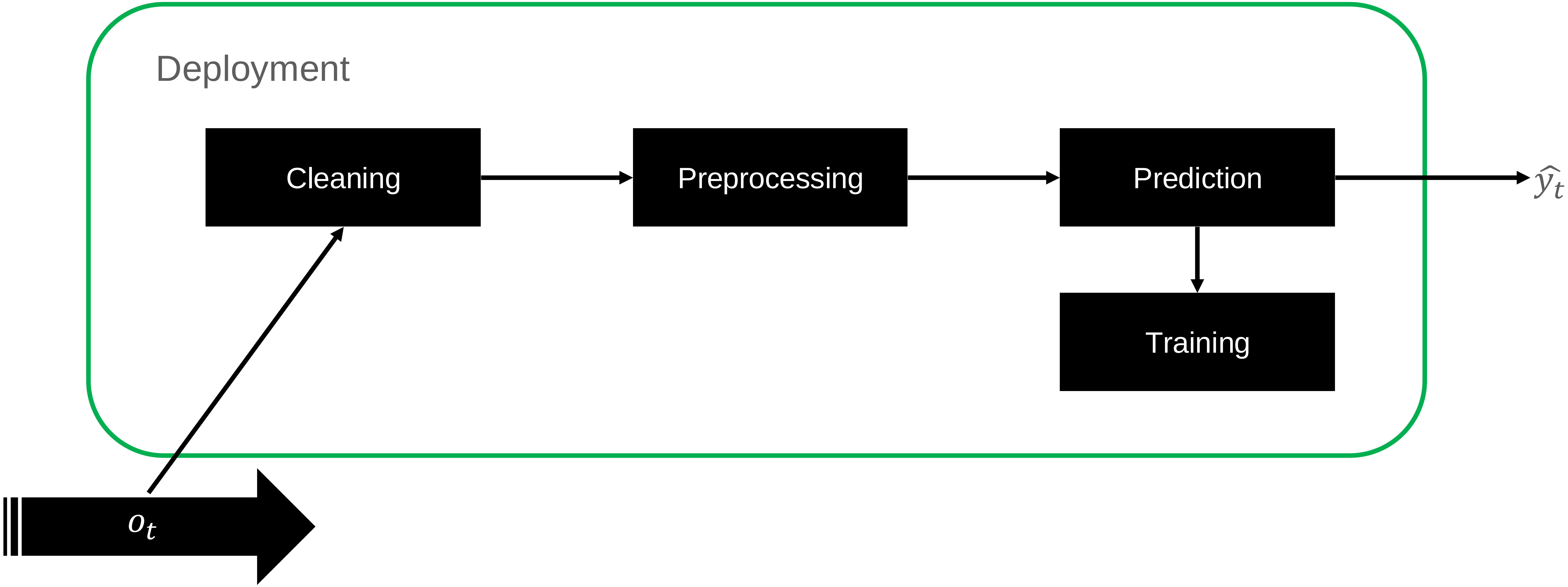
ML Pipelines for Data Streams

- Traditional ML Pipelines consist of well-defined, separate steps
- After training, model is deployed to make predictions
- In data streams, there is no separate deployment phase



ML Pipelines for Data Streams

- In data streams, there is no separate deployment phase
- Rather, one would deploy a pipeline as a trainable model that can predict at any time



ML Pipelines for Data Streams

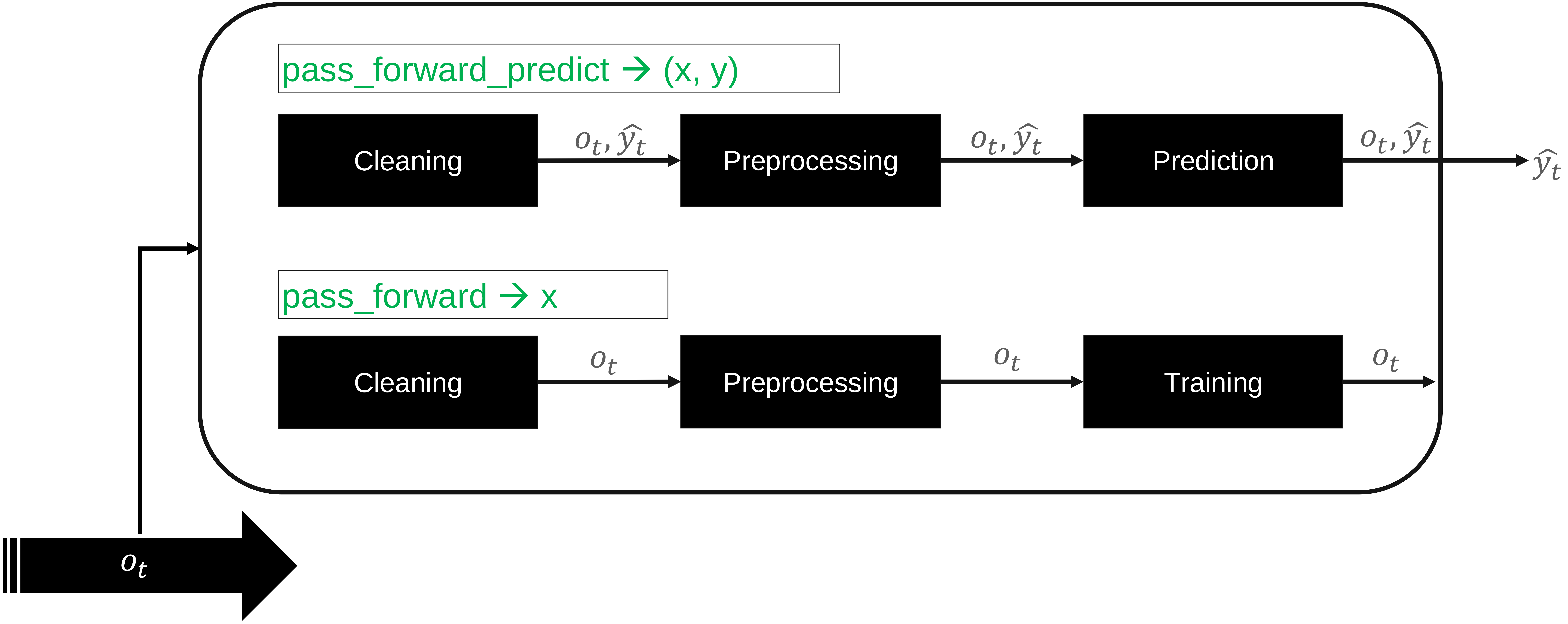
Test-then-train evaluation

For each new instance o_t:

- Clean and preprocess of o_t
- Predict label $\hat{y}_t$ // in the case of classification
- …
- Train model

Pipelines in CapyMOA

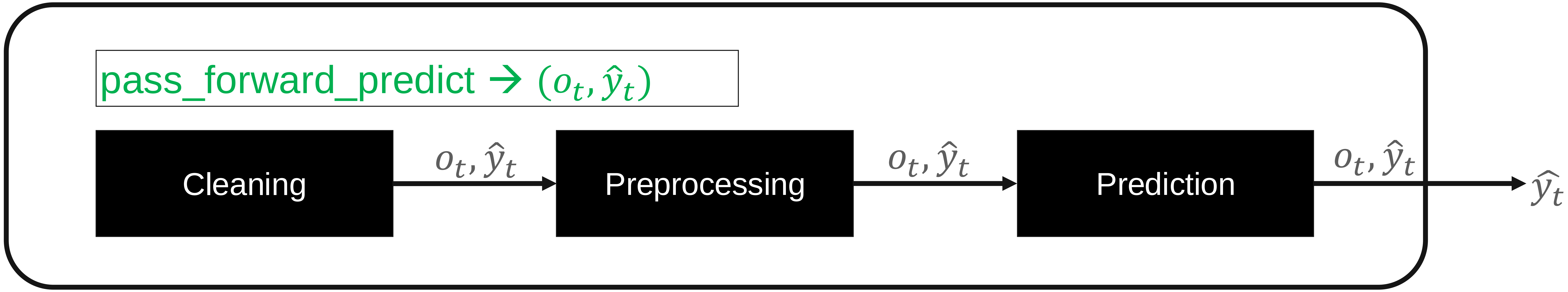
- We separate the pipeline into two basic procedures
- Ensures flexibility and interoperability within CapyMOA ecosystem



Pipelines in CapyMOA

pass_forward_predict → $(o_t, \hat{y}_t)$

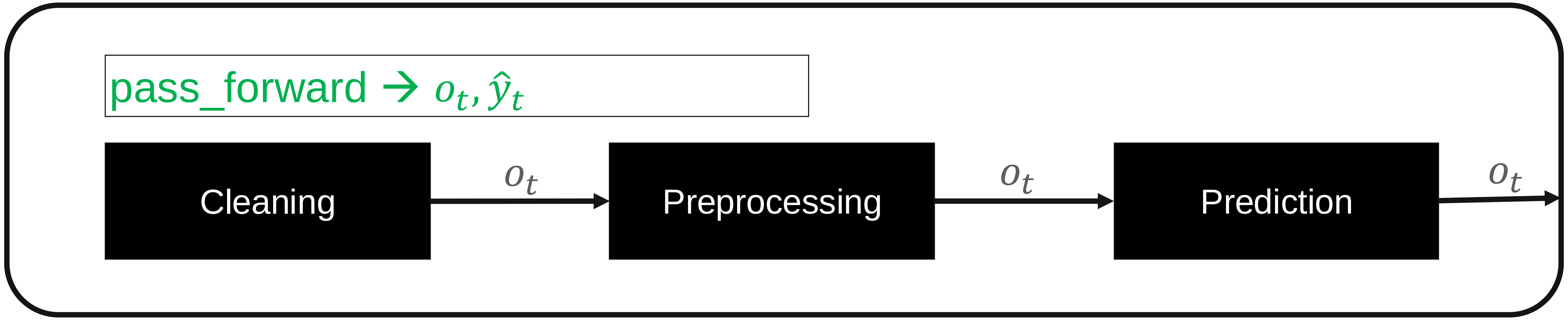
- Each element in the pipeline takes as input $(o_t, \hat{y}_t)$ and ouputs $(o_t, \hat{y}_t)$
- What happens within each pipeline element depends on its type
- For example: preprocessing → transforms o_t, prediction → classifier predicts $\hat{y}_t$



Pipelines in CapyMOA

pass_forward → o_t

- Each element in the pipeline takes as input o_t and outputs o_t
- What happens within each pipeline element depends on its type
- For example: preprocessing → transforms o_t, classifier → trains on o_t



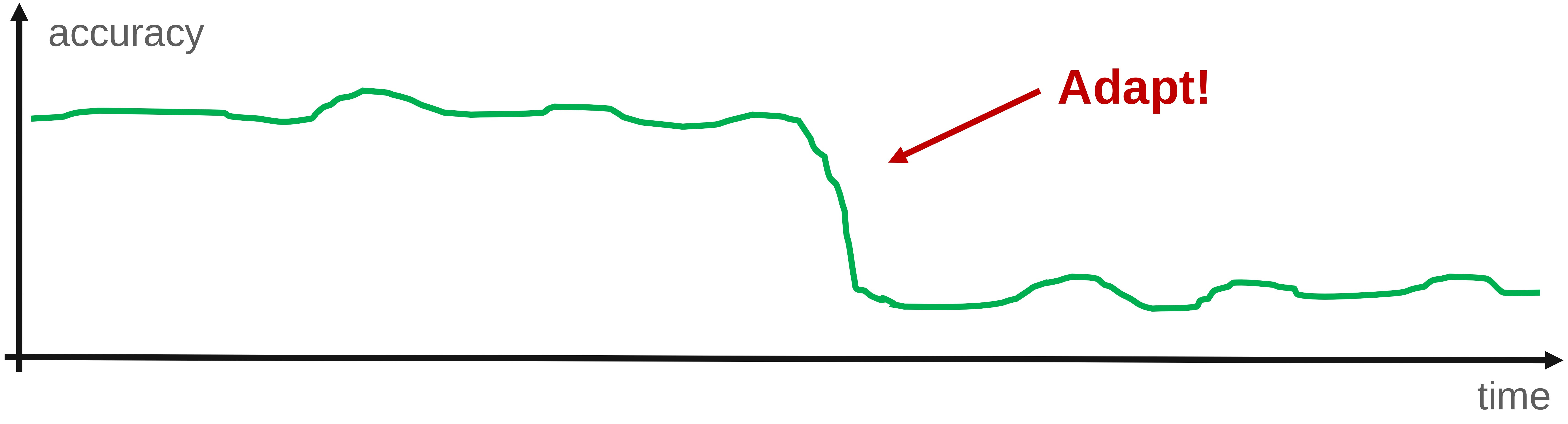
Pipelines

Adaptation to Concept Drift

Adaptation to Concept Drift

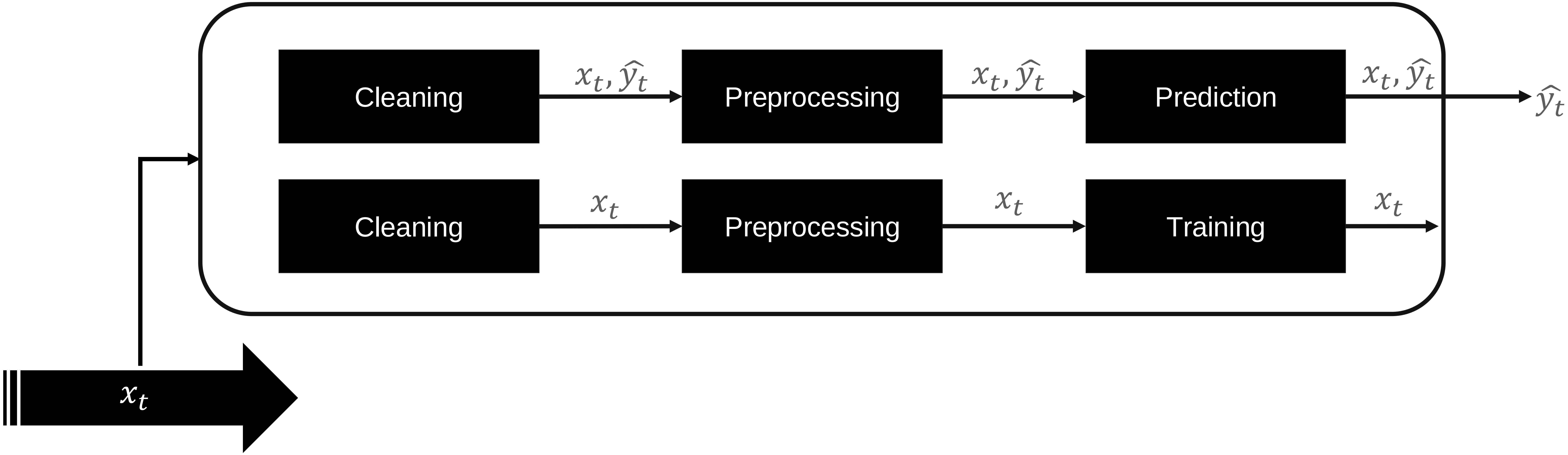
Data stream pipelines should offer ways to **adapt to concept drift**
- For example, one might want to reset a classifier after a change

This opens up several **design choices**, e.g.:
- Where in the pipeline to place drift detector? Multiple ones?
- How to facilitate drift adaptation?
- How to prepare input to drift detector? (can we maintain flexibility?)



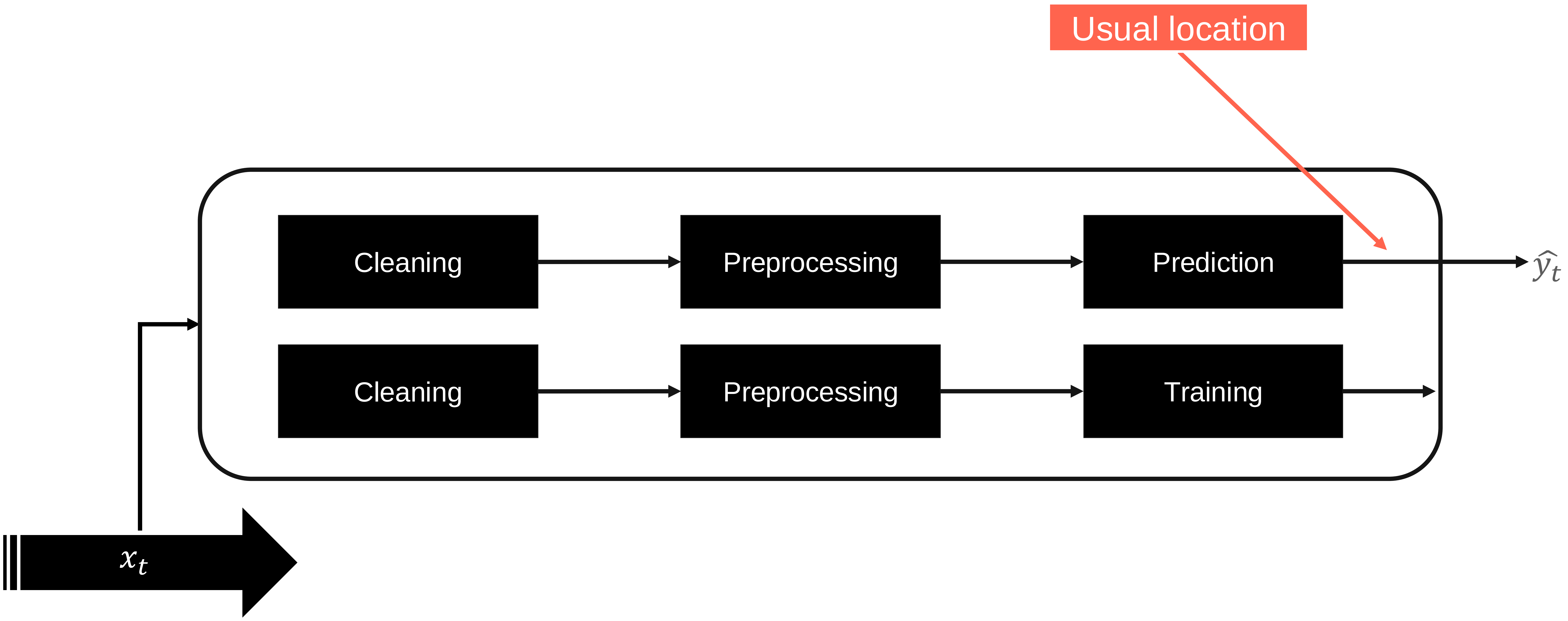
Adaptation to Concept Drift

Where in the pipeline to place drift detector?



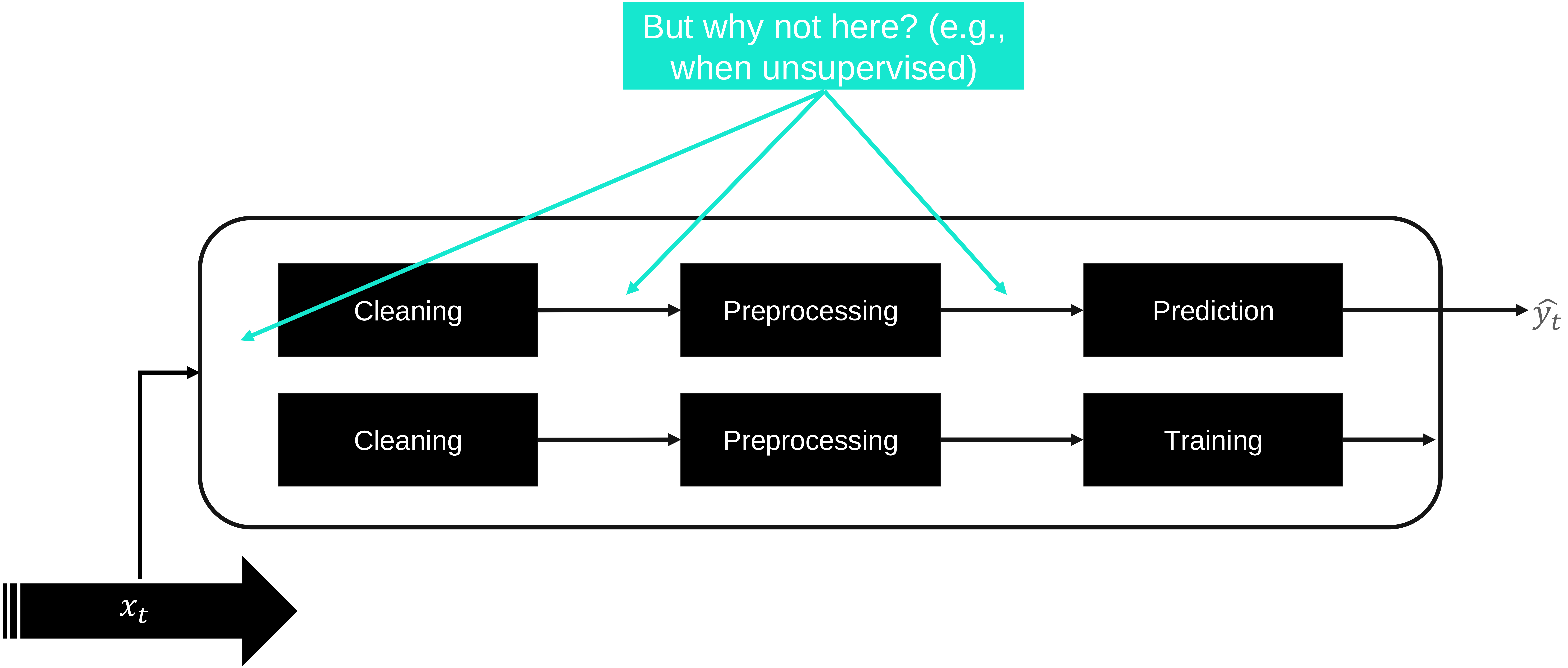
Adaptation to Concept Drift

Where in the pipeline to place drift detector?



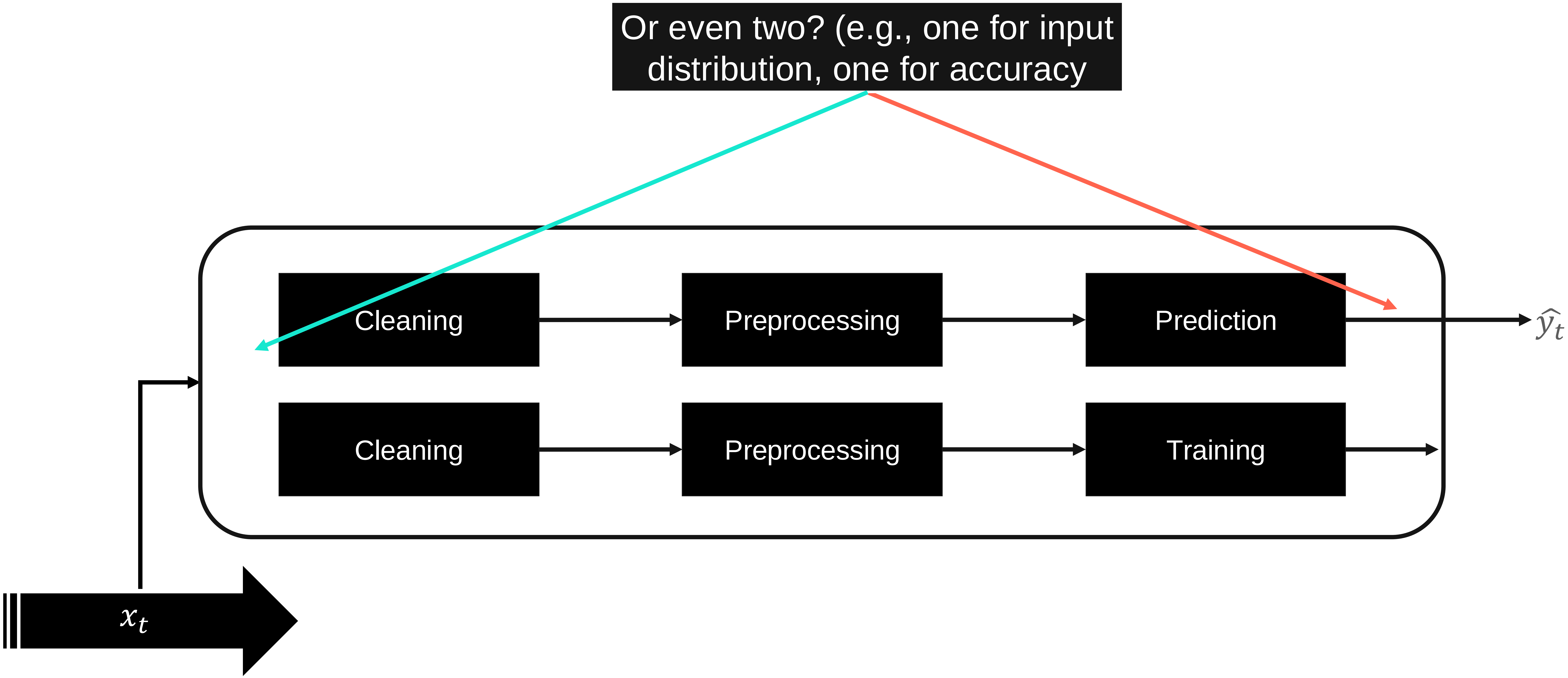
Adaptation to Concept Drift

Where in the pipeline to place drift detector?



Adaptation to Concept Drift

Where in the pipeline to place drift detector?

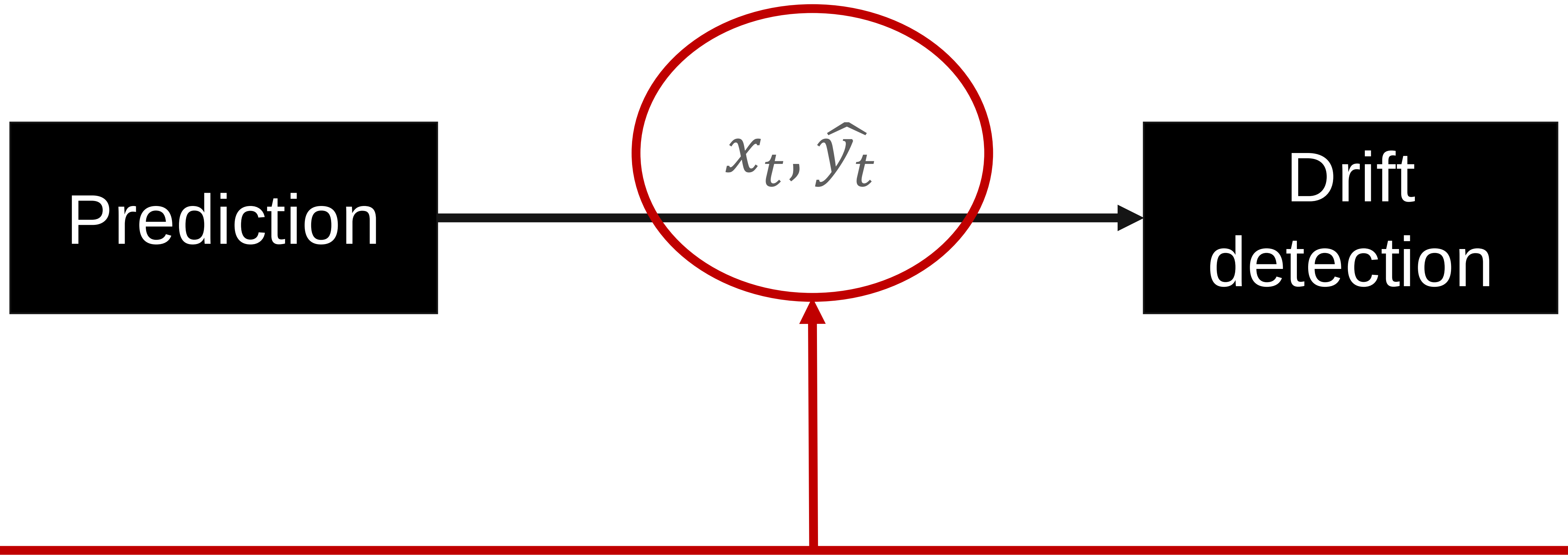


Adaptation to Concept Drift

- **Our pipelines can facilitate all the discussed combinations**
- **We are still working on the drift adaptation feature, so stay tuned!**

Adaptation to Concept Drift

How to prepare input to drift detector? (can we maintain flexibility?)



- As part of a pipeline, input to drift detector pipeline element always $(x_t, \hat{y}_t)$
- What if drift detector monitors accuracy (i.e., if $y_t = \hat{y}_t$)? Or the correlation between certain input features?

→ **We don't know. But we want to provide this flexibility!**

Adaptation to Concept Drift

→ Specify a function that prepares the input for the drift detector

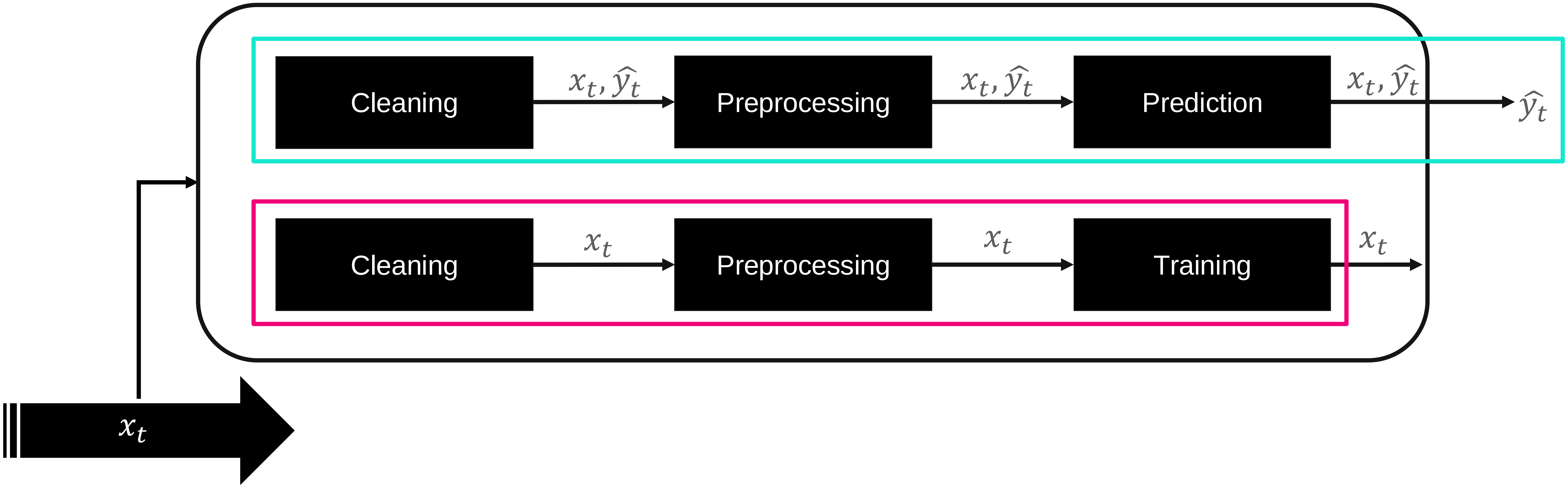
```
def accuracy(instance, y_hat):
    return int(y == y_hat)

detector = Adwin()
drift_pe = DriftDetectorPipelineElement(
                                  detector=detector,
                                  input_func=accuracy)

// Internally, the pipeline element runs:
input = self.input_func(instance, y_hat)
self.detector.update(input)
```

Classifier and Regressor Pipelines

- Besides "abstract" pipelines, we also support classifier and regressor pipelines
- These pipelines implement **predict** and **train** and can thus be used like any other classifier



Practical examples

ECML_2024_pipelines.ipynb