OPTIMIS

Framework for optimal control of a message in social media using deep reinforcement learning

Presented by:

Deborah Zenobia Rachael Menezes

Supervisors:

Prof. Dr. Ajinkya Prabhune

Prof. Dr. Gerd Moeckel

HOCHSCHULE HEIDELBERG

MOTIVATION

Motivation

➔ EnFVe: An Ensemble Fact Verification Pipeline is a recently published paper
➔ The four stages of fact checking according to FullFact
➔ The ‘Create and publish’ block of the pipeline is the focus for this thesis.

Four stages of fact-checking [1]

INTRODUCTION

Business Use-case

"A product marketing associate has a need to maximize the visibility of the product in social media"

High Level Overview

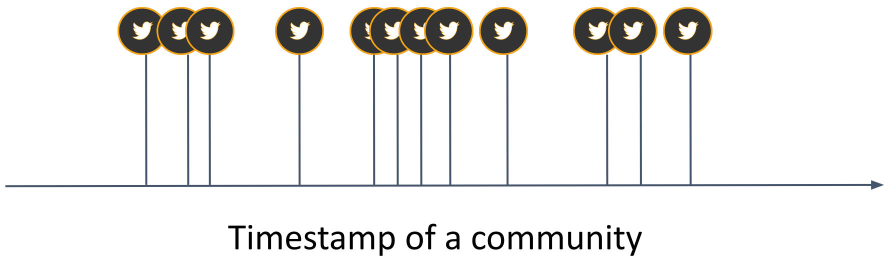
Problem

High Level Overview

Problem

High Level Overview

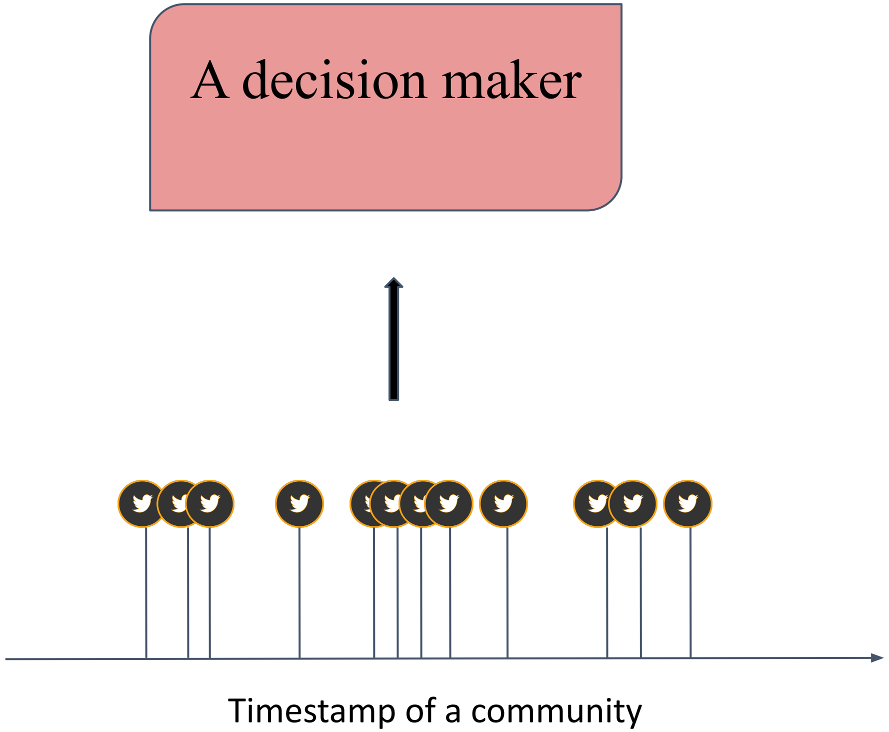
Problem



Solution

High Level Overview

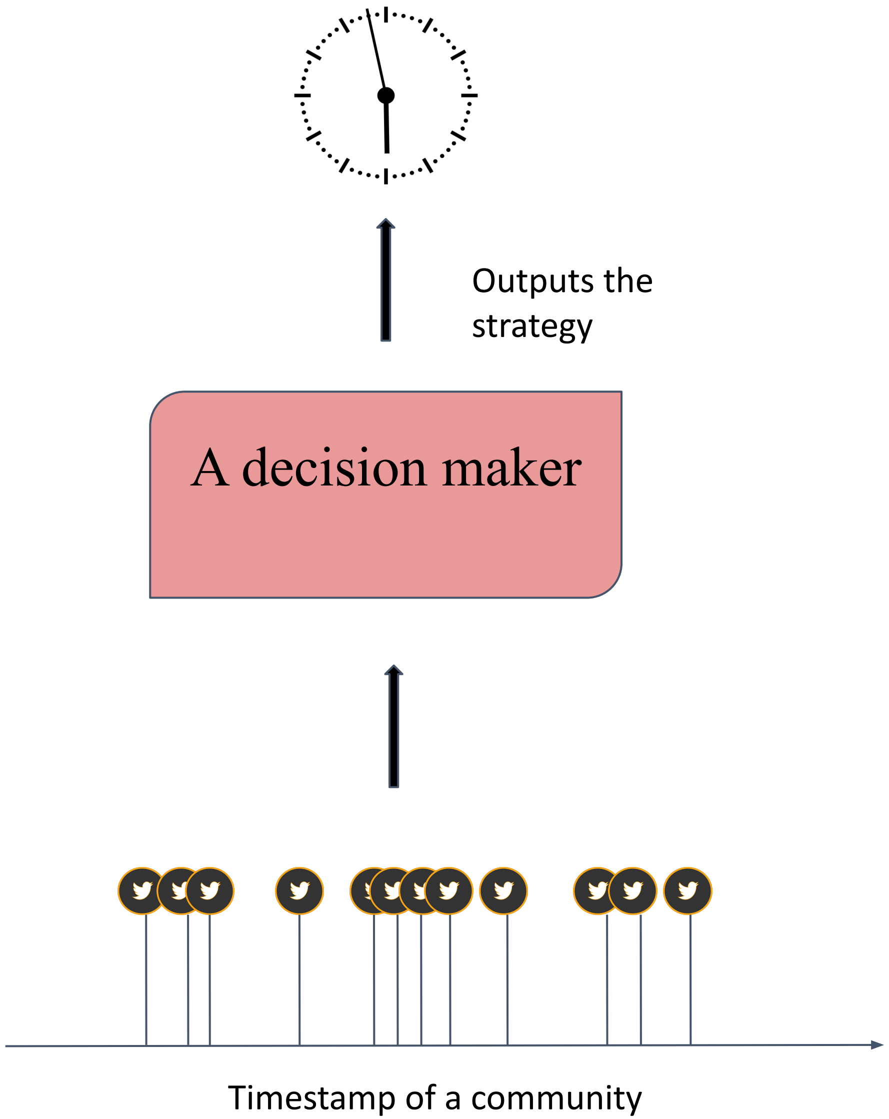
Problem



Solution

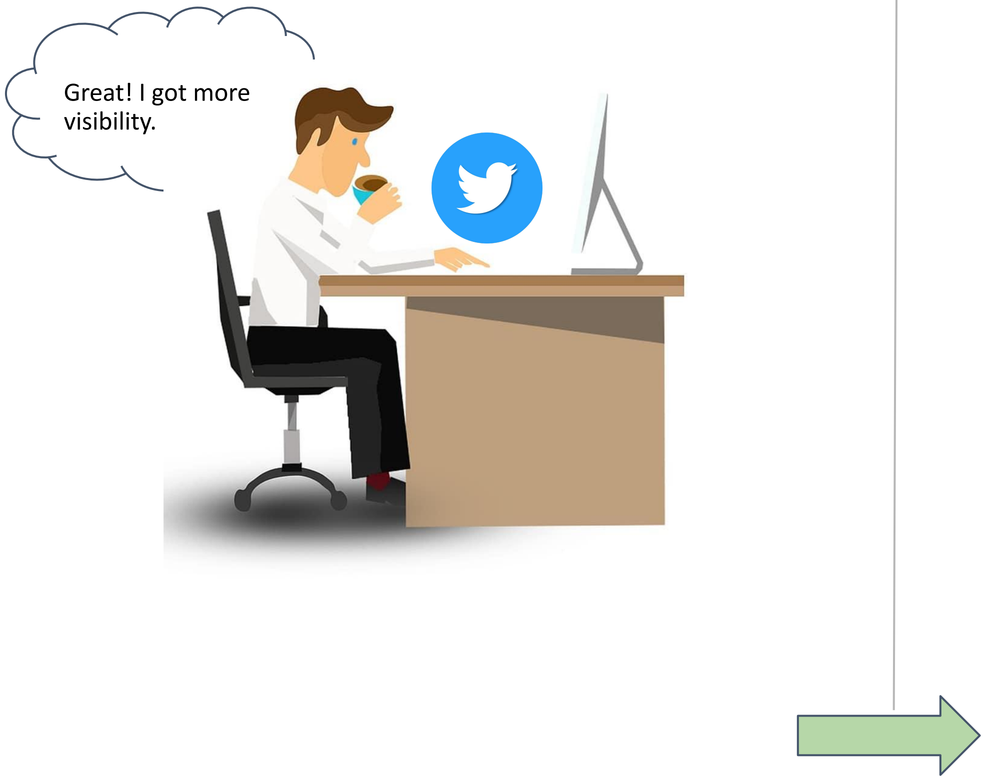
High Level Overview

Problem

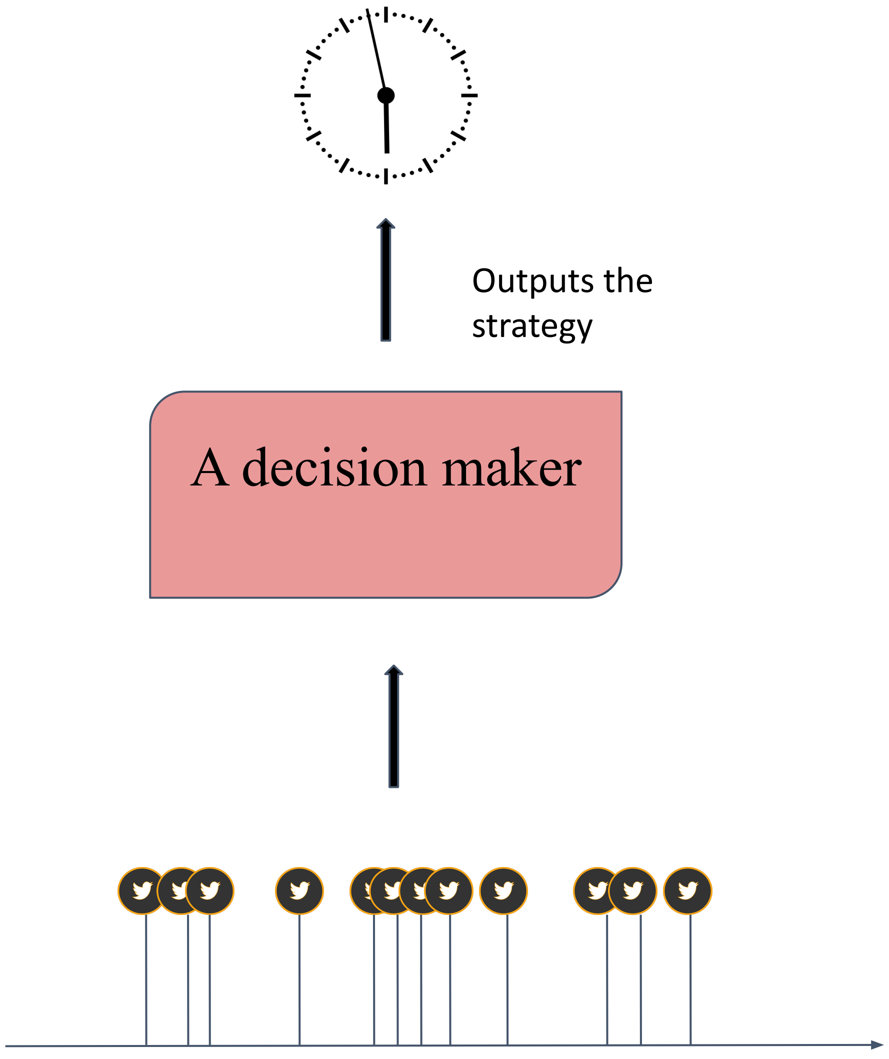


Solution

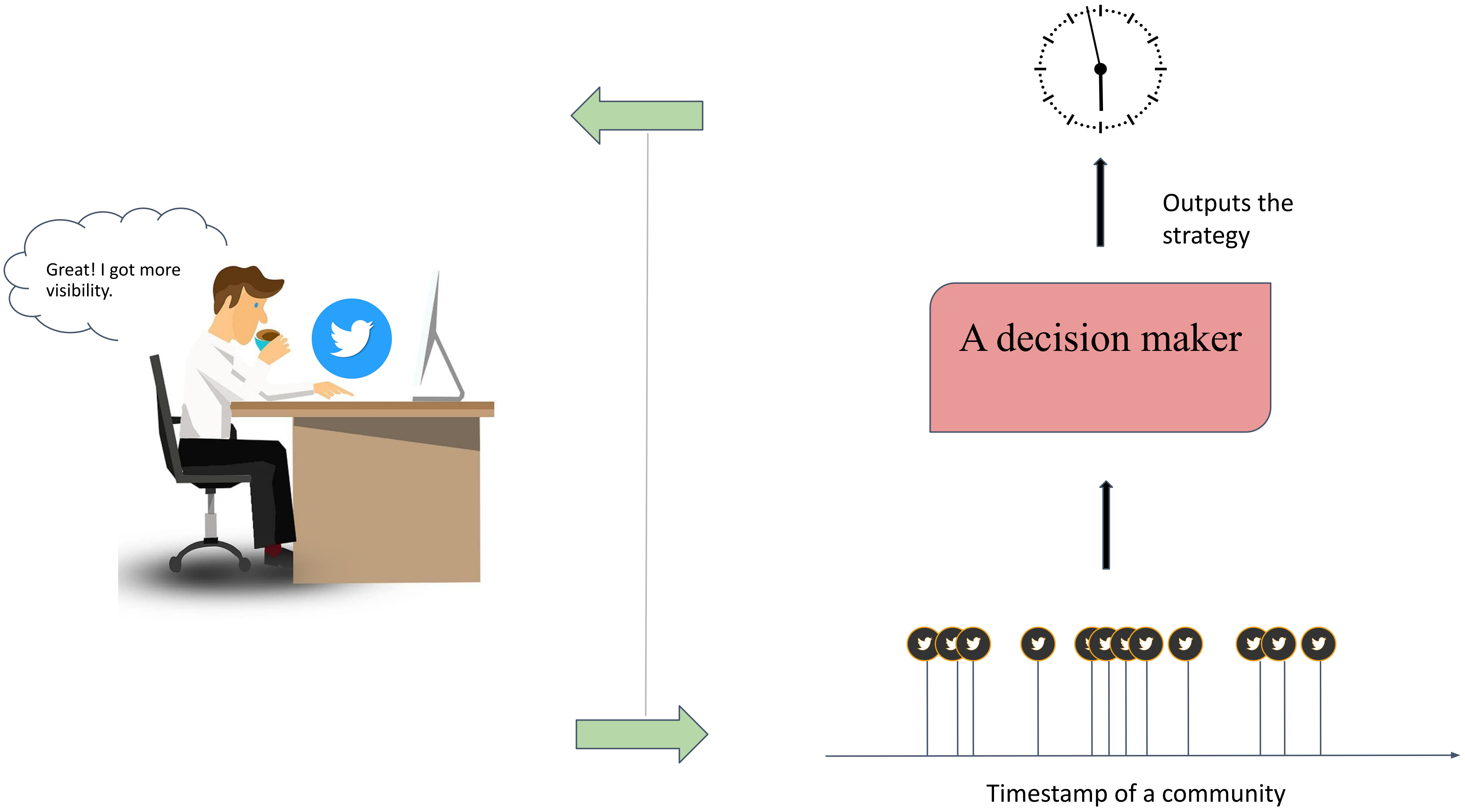
High Level Overview



Problem



Solution

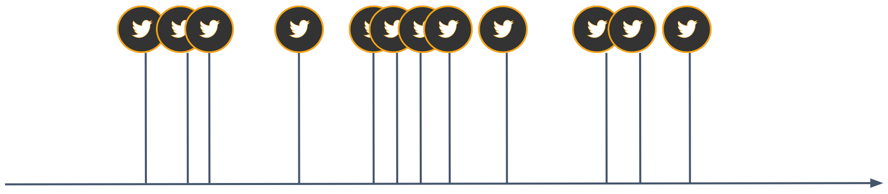


Problem

Solution

LITERATURE REVIEW

Point processes & its applications



Tweets timestamps

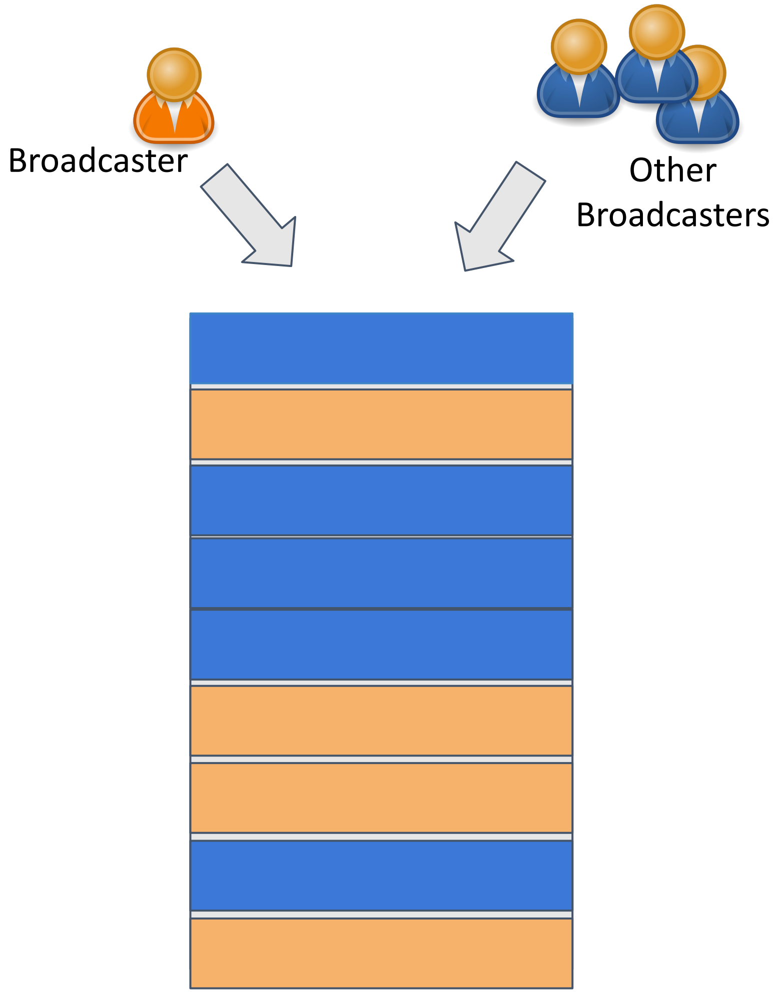
- ➔ Tweets are basically the temporal points.
- ➔ Typical domains where point processes exist: Trading, disease prevention, influence maximization, etc.
- ➔ The “when-to-post” problem is essentially the problem of influence maximization.

<u>The scope of the thesis is centered around influence maximisation</u>

Influence maximization

- ➔ Influence maximization encompasses a broad class of applications: effective broadcasting, ad placement, viral marketing, etc
- ➔ IM can be thought of as a decision making mechanism on point process where the broadcaster decides when to post to maximize her post visibility.
- ➔ Reinforcement learning is a good candidate for modeling on decision making processes.

Using Reinforcement learning on Influence Maximisation is a promising idea



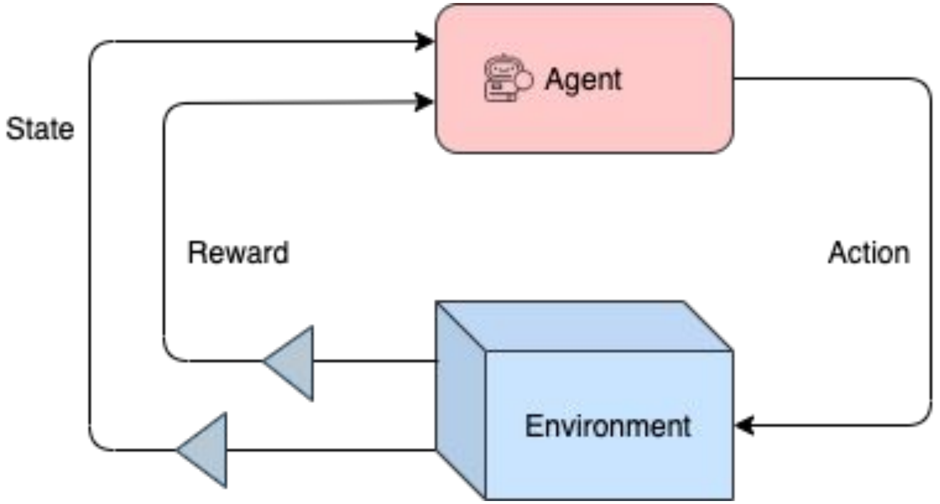
A user's feed

A short walkthrough of RL

Reinforcement learning is the idea of making the optimal sequence of actions in an environment in order to maximize reward.

Consists of two primary components:

- **The agent** is basically the brain of the framework, which learns a policy to act upon.
- **The environment** provides the agent with the state to act upon, and also given the reward for informing future actions.



Reinforcement Learning cycle [5]

The current research focus and gaps

Agents

- The main focus is on building the RL agents
- For Influence maximization, Deep RL approaches that employs complex policy gradient methods like TPPRL [2]

Environments

- Mostly game-like environments exist out there (atari)
- For time series, support is very limited as well. Especially in trading such as tensortrade library [3]
- Interesting environments simulating heat dynamics of a building [4].

Takeaway

Not much research in RL for temporal point processes

Problem

There exists no environments for enabling decision-making problems that involve influence maximisation

Solution

A custom RL environment that can support various influence maximisation strategies.

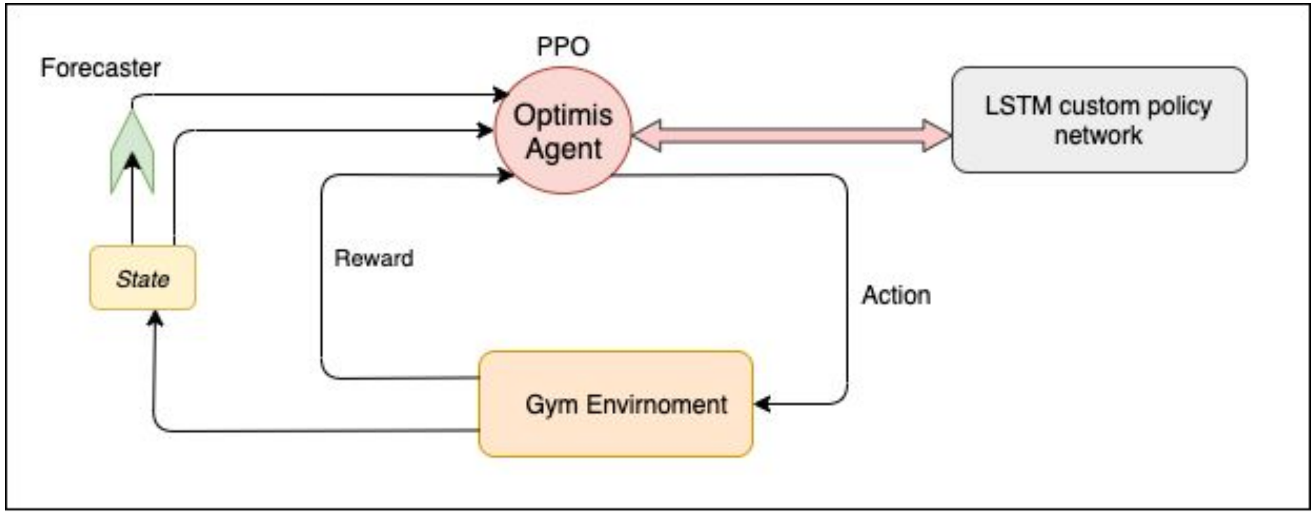
Contribution

Enables rapid development of influence-maximisation-based applications as well as accelerating research in the domain

IMPLEMENTATION OF THE OPTIMIS FRAMEWORK

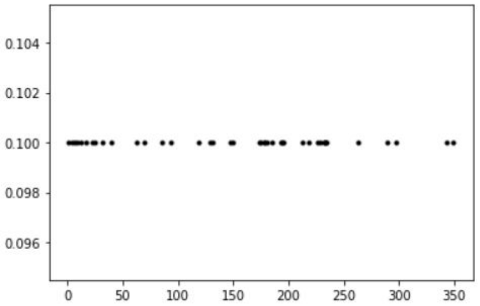
Architecture overview

- ➔ The framework consists of two primary components:
  - ◆ the environment
  - ◆ agent
- ➔ the environment is the highlight of the framework
- ➔ the agent is employed to illustrate its application of the environment
- ➔ The API layer enables users to customize various strategies

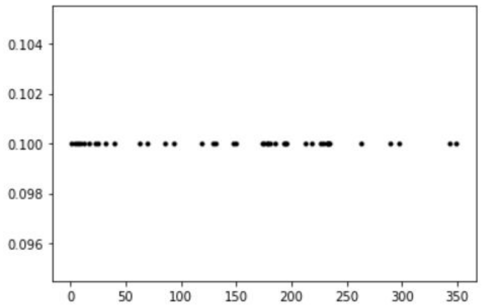


OPTIMIS FRAMEWORK

Preprocessing

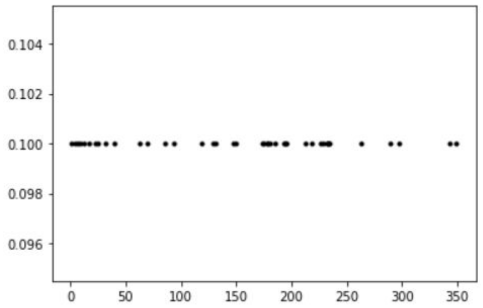


Preprocessing: first order time difference

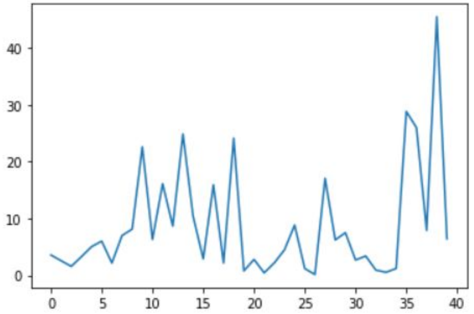


First-order time difference

Preprocessing: first order time difference

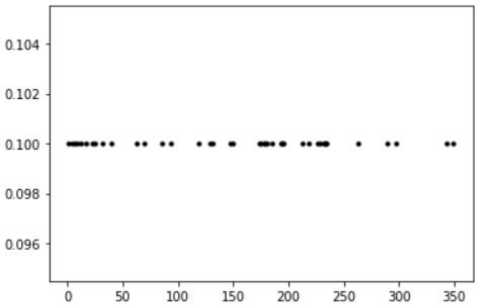


First-order time difference

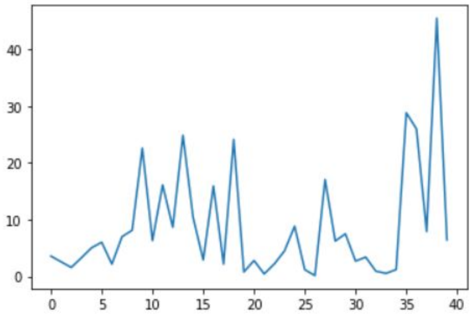


transformed values

Forecaster module



First-order time difference

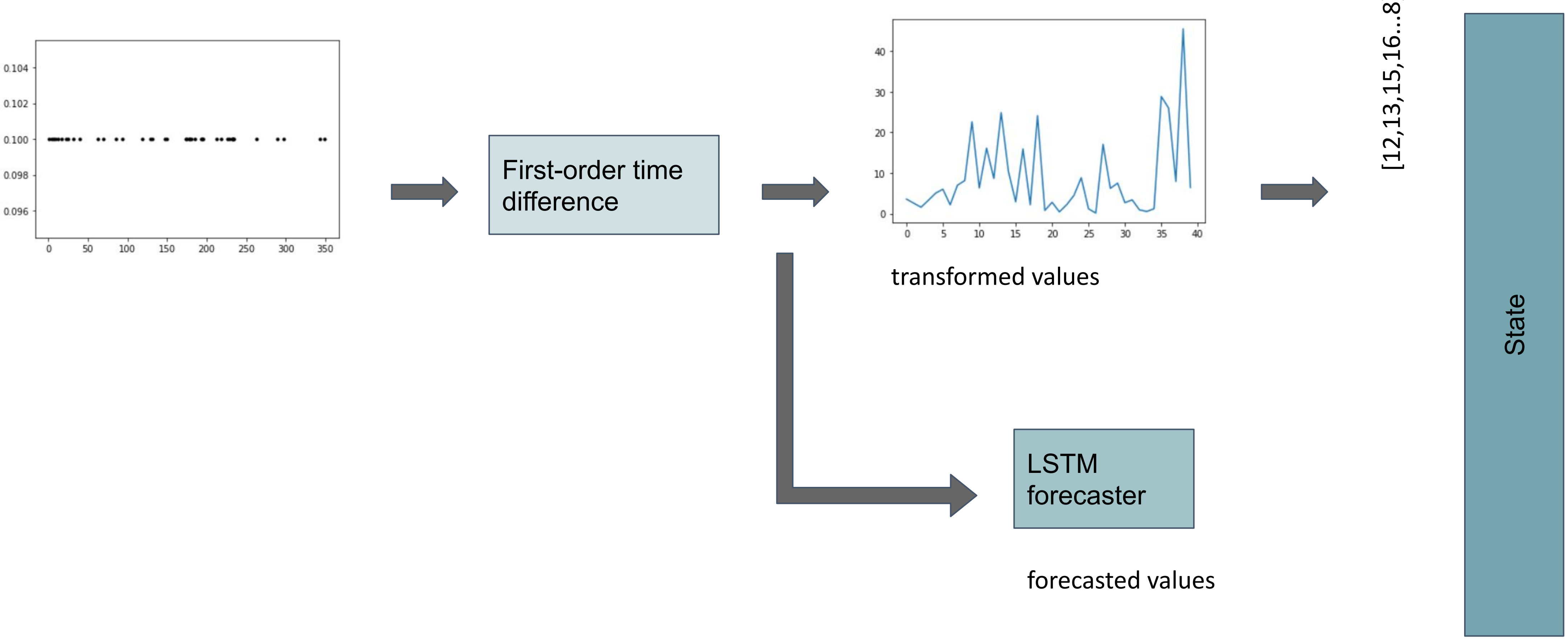


transformed values

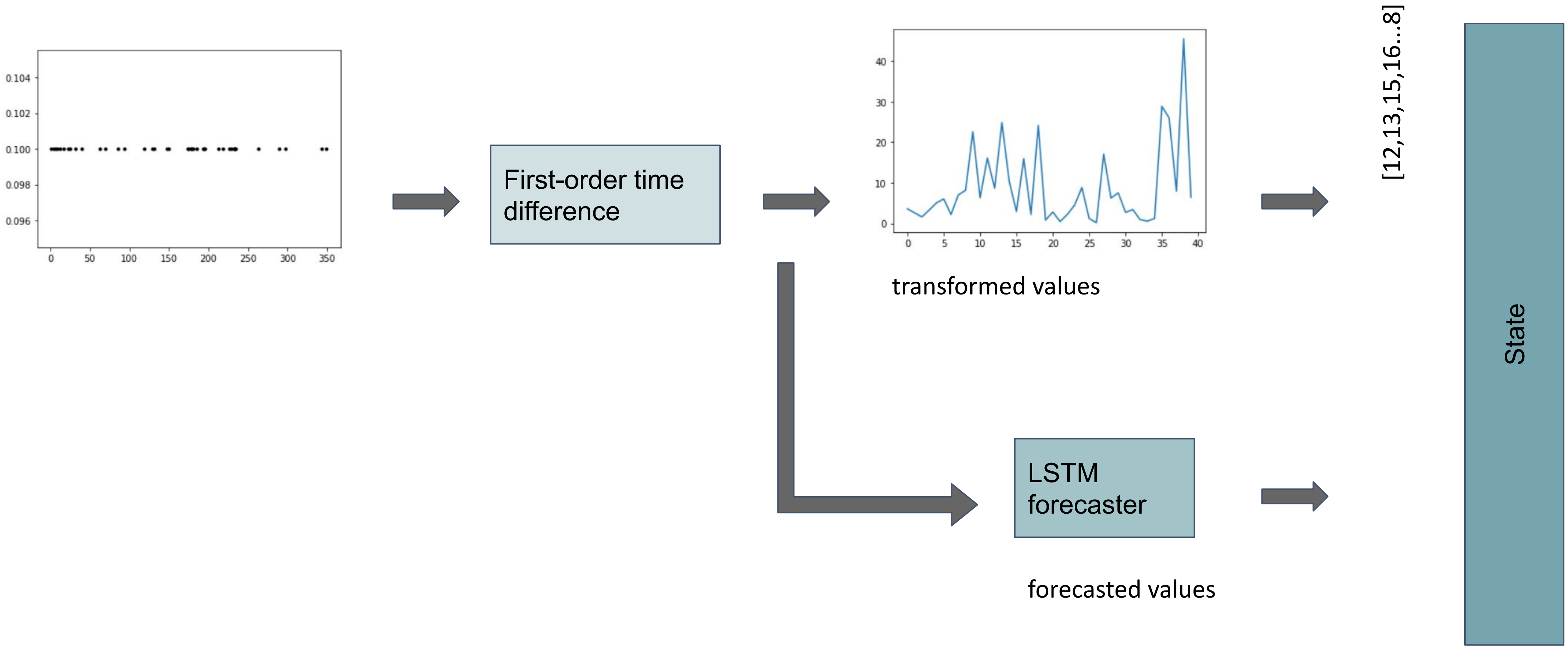
[12,13,15,16...8]

State

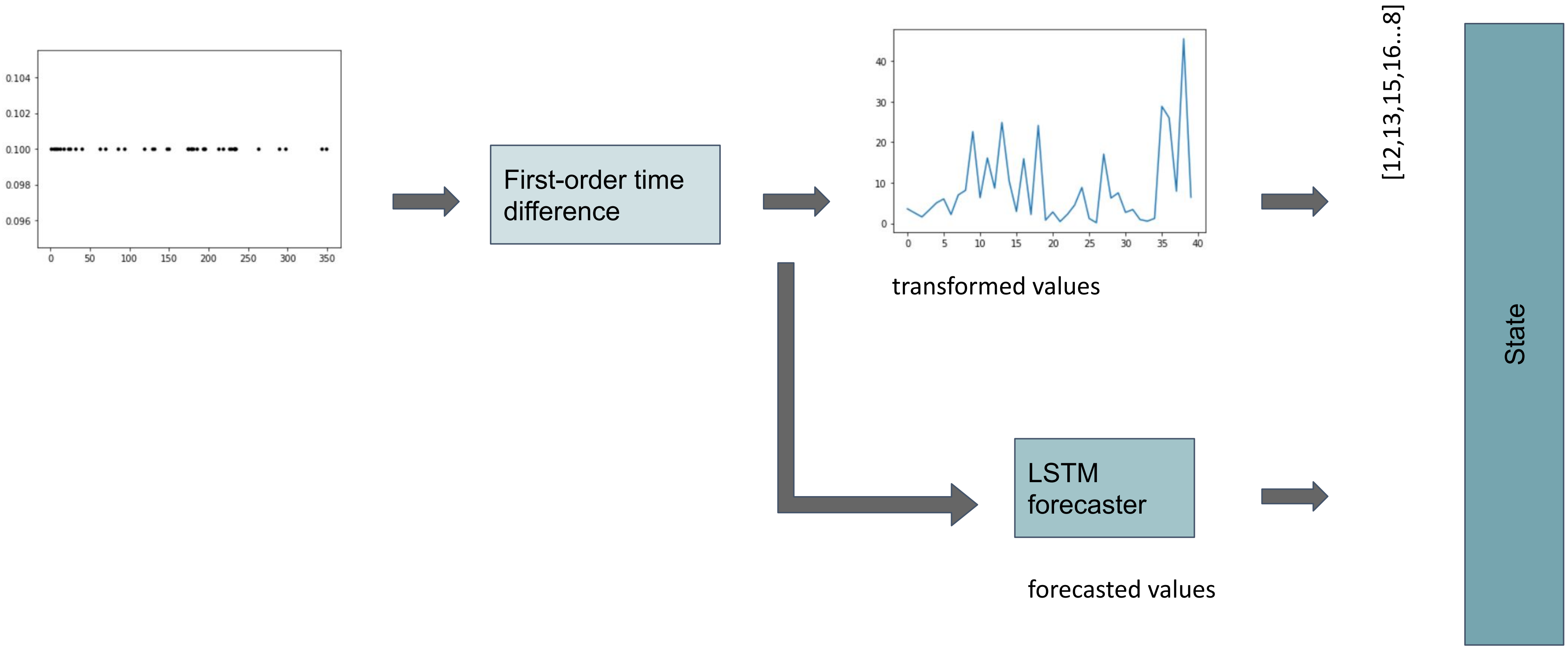
Forecaster module



Forecaster module



Forecaster module



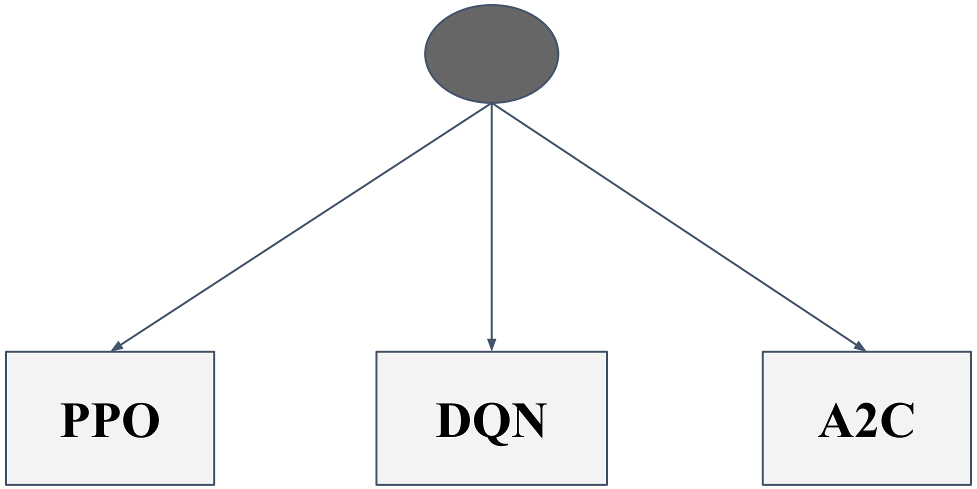
User has the control over which preprocessing step to be used and which ones to be omitted

Agents

- Basic support for agents are included, with stable baselines and rllib environments which has the algorithms

- The algorithms are chosen as they support discrete actions as outputs

- The code shows an implementation of how a PPO agent can be used through RLLib.




```
from ray import tune
from ray.rllib.agents.ppo import PPOTrainer
#Running the PPO with recurrent policy
Optimis = tune.run(PPOTrainer, config={"env":'my_env',
                            'num_workers': 1, 'num_gpus': 1,
                            "model": {"use_lstm": True}})
```

Description of environment

Based on Open AI gym environment

Reward function:
Reward = future_timestamp - current timestamp - action_penalty

Episode termination conditions:
- If the current step exceeds the total episode length
- If the actions made by the agent is more than the allowed actions
- If the current time step exceeds the allowed time limit

State:
- A sliding window of past n timesteps
- Forecaster is added to the state

API interface

```
# loading twitter data
#returns a list of timestamps
data = load_twitter_data()

#initialise the optimis environment
env = OptimisEnv()

#load the temporal points into the environment
env.set_timestamps(data)

#helper function to display information about the data
env.display_data_stats()

#setting strategies
env.set_constraints(action_penalty=30,
                    max_allowed_actions=None)

#train forecaster
env.train_forecaster(num_steps=18, num_epochs=350)

#enable or disable preprocessing techniques
env.settings(use_forecaster=True,
             use_previous_timestamps=False,
             first_difference=True)
```

API interface

```
# loading twitter data
#returns a list of timestamps
data = load_twitter_data()

#initialise the optimis environment
env = OptimisEnv()

#load the temporal points into the environment
env.set_timestamps(data)

#helper function to display information about the data
env.display_data_stats()

#setting strategies
env.set_constraints(action_penalty=30,
                    max_allowed_actions=None)

#train forecaster
env.train_forecaster(num_steps=18, num_epochs=350)

#enable or disable preprocessing techniques
env.settings(use_forecaster=True,
             use_previous_timestamps=False,
             first_difference=True)
```

API interface

```
# loading twitter data
#returns a list of timestamps
data = load_twitter_data()

#initialise the optimis environment
env = OptimisEnv()

#load the temporal points into the environment
env.set_timestamps(data)

#helper function to display information about the data
env.display_data_stats()

#setting strategies
env.set_constraints(action_penalty=30,
                    max_allowed_actions=None)

#train forecaster
env.train_forecaster(num_steps=18, num_epochs=350)

#enable or disable preprocessing techniques
env.settings(use_forecaster=True,
             use_previous_timestamps=False,
             first_difference=True)
```

API interface

```
# loading twitter data
#returns a list of timestamps
data = load_twitter_data()

#initialise the optimis environment
env = OptimisEnv()

#load the temporal points into the environment
env.set_timestamps(data)

#helper function to display information about the data
env.display_data_stats()

#setting strategies
env.set_constraints(action_penalty=30,
                    max_allowed_actions=None)

#train forecaster
env.train_forecaster(num_steps=18, num_epochs=350)

#enable or disable preprocessing techniques
env.settings(use_forecaster=True,
             use_previous_timestamps=False,
             first_difference=True)
```

API interface

```
# loading twitter data
#returns a list of timestamps
data = load_twitter_data()

#initialise the optimis environment
env = OptimisEnv()

#load the temporal points into the environment
env.set_timestamps(data)

#helper function to display information about the data
env.display_data_stats()

#setting strategies
env.set_constraints(action_penalty=30,
                    max_allowed_actions=None)

#train forecaster
env.train_forecaster(num_steps=18, num_epochs=350)

#enable or disable preprocessing techniques
env.settings(use_forecaster=True,
             use_previous_timestamps=False,
             first_difference=True)
```

API interface

```
# loading twitter data
#returns a list of timestamps
data = load_twitter_data()

#initialise the optimis environment
env = OptimisEnv()

#load the temporal points into the environment
env.set_timestamps(data)

#helper function to display information about the data
env.display_data_stats()

#setting strategies
env.set_constraints(action_penalty=30,
                    max_allowed_actions=None)

#train forecaster
env.train_forecaster(num_steps=18, num_epochs=350)

#enable or disable preprocessing techniques
env.settings(use_forecaster=True,
             use_previous_timestamps=False,
             first_difference=True)
```

API interface

```
# loading twitter data
#returns a list of timestamps
data = load_twitter_data()

#initialise the optimis environment
env = OptimisEnv()

#load the temporal points into the environment
env.set_timestamps(data)

#helper function to display information about the data
env.display_data_stats()

#setting strategies
env.set_constraints(action_penalty=30,
                    max_allowed_actions=None)

#train forecaster
env.train_forecaster(num_steps=18, num_epochs=350)

#enable or disable preprocessing techniques
env.settings(use_forecaster=True,
             use_previous_timestamps=False,
             first_difference=True)
```

API interface

```
# loading twitter data
#returns a list of timestamps
data = load_twitter_data()

#initialise the optimis environment
env = OptimisEnv()

#load the temporal points into the environment
env.set_timestamps(data)

#helper function to display information about the data
env.display_data_stats()

#setting strategies
env.set_constraints(action_penalty=30,
                    max_allowed_actions=None)

#train forecaster
env.train_forecaster(num_steps=18, num_epochs=350)

#enable or disable preprocessing techniques
env.settings(use_forecaster=True,
             use_previous_timestamps=False,
             first_difference=True)
```

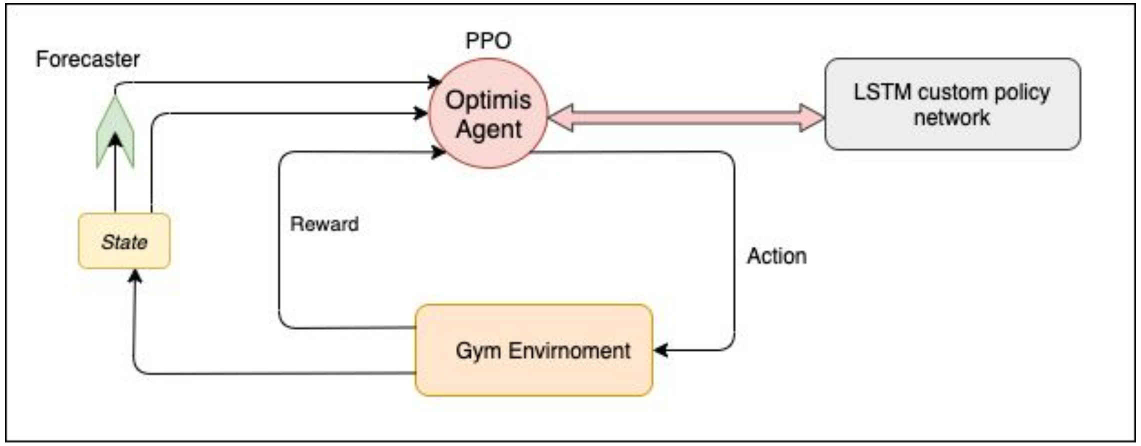
API interface

```
# loading twitter data
#returns a list of timestamps
data = load_twitter_data()

#initialise the optimis environment
env = OptimisEnv()

#load the temporal points into the environment
env.set_timestamps(data)

#helper function to display information about the data
env.display_data_stats()

#setting strategies
env.set_constraints(action_penalty=30,
                    max_allowed_actions=None)

#train forecaster
env.train_forecaster(num_steps=18, num_epochs=350)

#enable or disable preprocessing techniques
env.settings(use_forecaster=True,
             use_previous_timestamps=False,
             first_difference=True)

#training the model
model = env.train_agent(policy='PPO', iterations=50_000)
```

Summary

The Optimis framework consists of:

- Data preprocessing for handling point processes
- A high-level API that lets users set strategies
- A list of RL policies



OPTIMIS FRAMEWORK

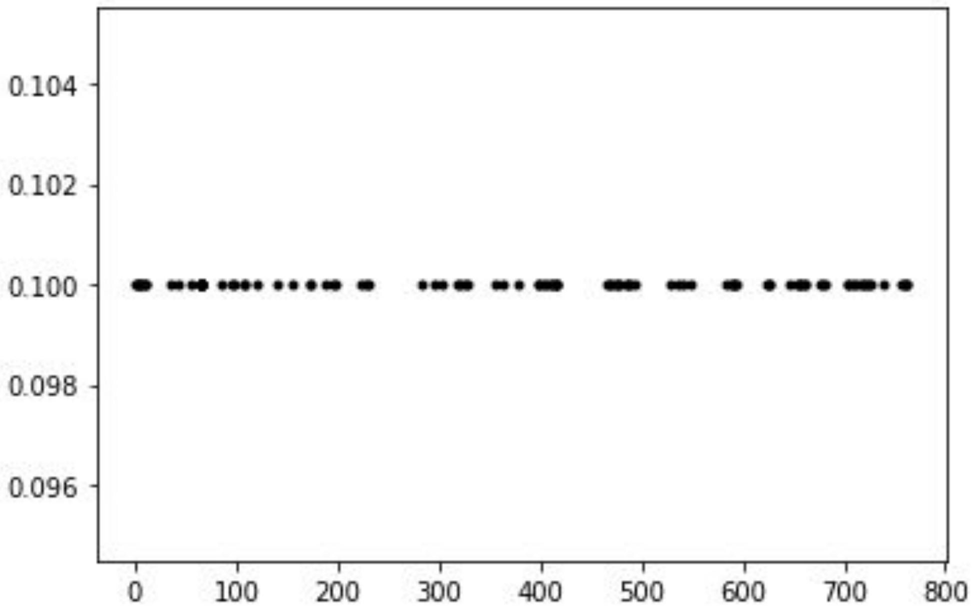
EVALUATION

Evaluation: on twitter dataset

Collected 1913 tweets over 5 days

average interval between two points: 3.76 minutes

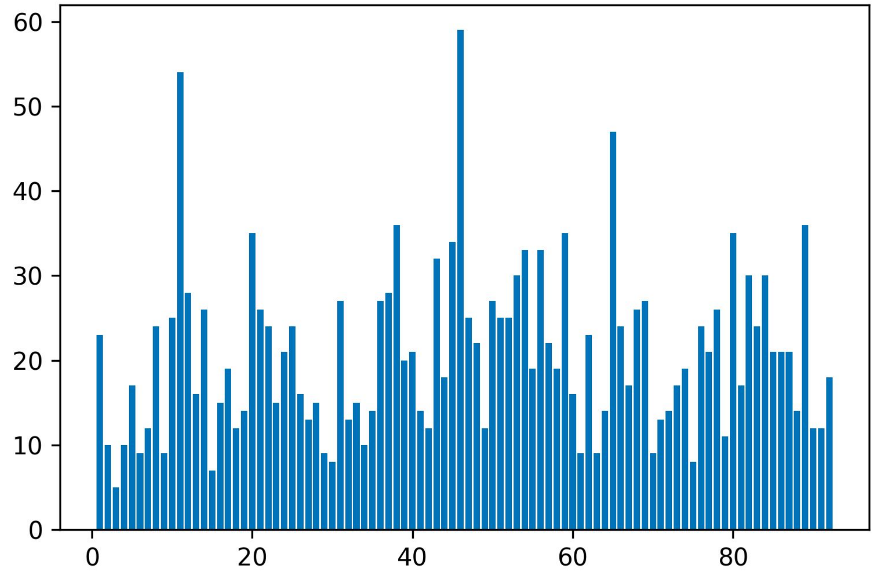
Top 10 intervals:[42.4, 41.6, 38.47, 35.33, 35.33, 33.76, 32.98, 30.62, 30.62, 29.83] minutes



interesting pattern of increasing and decreasing intensities can be observed

Evaluation

- Fetched tweets related to #machinelearning
- Total number of tweets: 1913
- Duration: 2.5 hours
- Average interval between two points: 4.79 secs
- Top 5 intervals found in the dataset: 54.0, 53.0, 49.0, 45.0, 45.0 secs



Algorithm	No preprocessing	First-order time difference	Forecaster	Forecaster+FO
DQN	(-32.3) N/A	(7.1) N/A	(8.5) 32.9	(11.2) 34.1
A2C	(-12.8) N/A	(-13.4) N/A	(-4.8) N/A	(-4.2) N/A
PPO	(8.6) 35.7	(42.6) 45.1	(56.9) 44.9	(63.2) 45.8

RUNNING EXAMPLE

Consider these scenarios

"I want to make a post that lasts at least half an hour on social media"

"I want to maximize the visibility of the three posts that I have"

"I want to make 3 posts in the next hour and make sure that they gets an average airtime of at least 30 seconds"

Posting strategy

"I want to make 3 posts in the next hour and make sure that it gets an airtime of at least 30 seconds"

User wants to post:

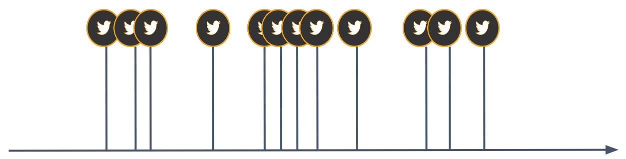
No of posts → 3 times

Time limit → in the next hour

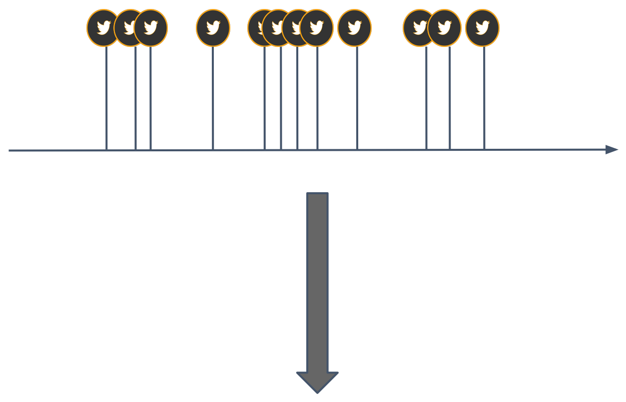
Post time should be minimum → 30 seconds

```
#setting strategies
#unit of time is seconds
env.set_constraints(action_penalty=30,
                    max_allowed_actions=3,
                    time_limit=3600)
```

How each timestep works

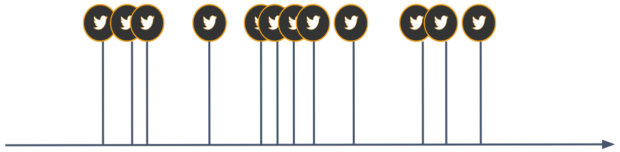


How each timestep works



[1.0, 3.0, 4.0, 5.0, 6.0, 7.0, 10.0, 11.0, 36.0, 42.0]

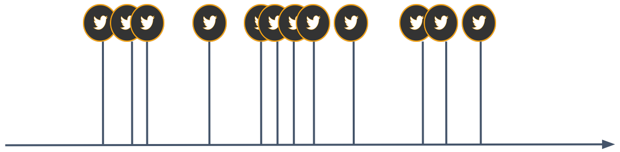
How each timestep works



[1.0, 3.0, 4.0, 5.0, 6.0, 7.0, 10.0, 11.0, 36.0, 42.0]

[3., 0., 1., 1., 2., 3., 1., 25., 6.]

How each timestep works

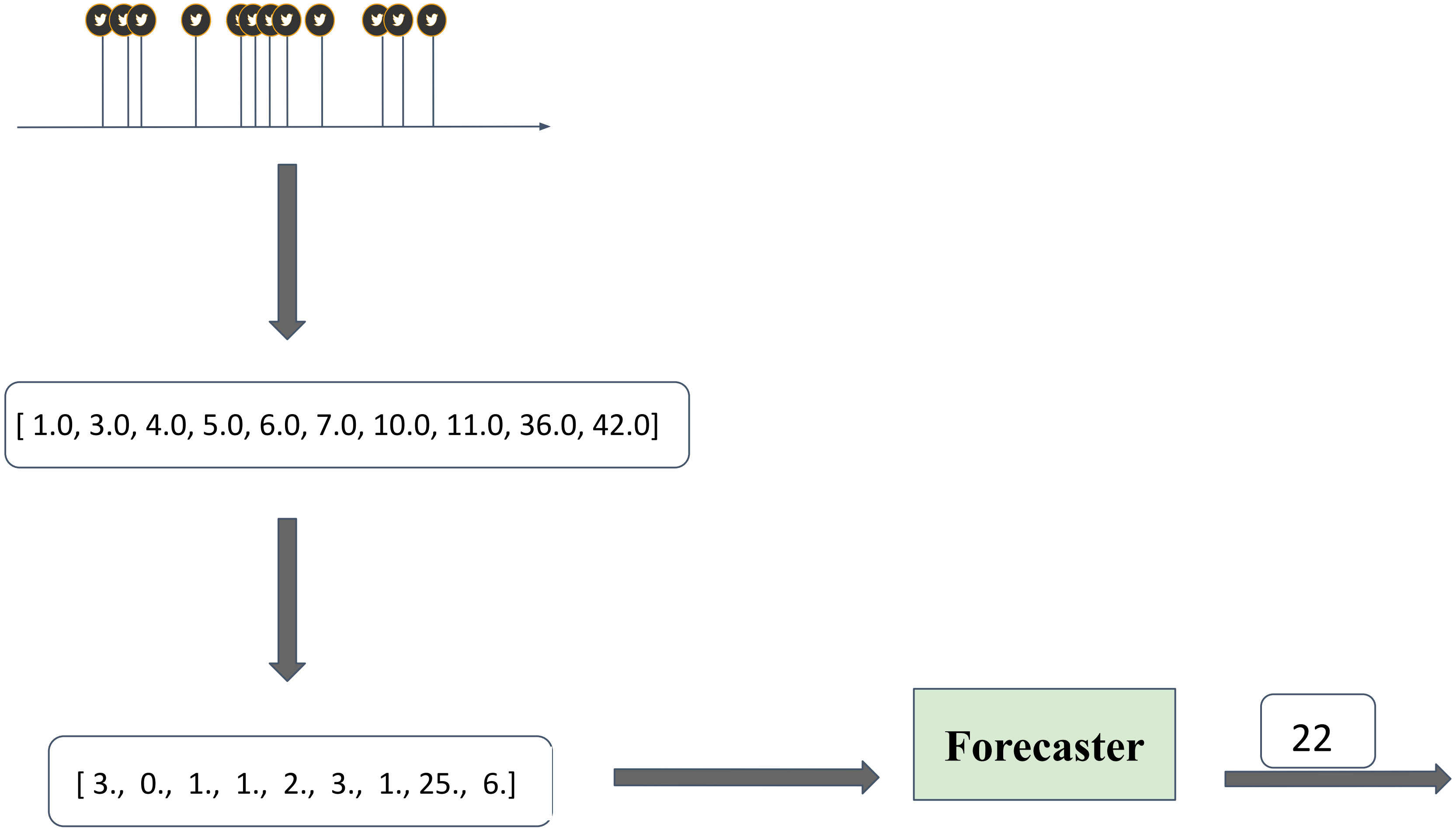


[1.0, 3.0, 4.0, 5.0, 6.0, 7.0, 10.0, 11.0, 36.0, 42.0]

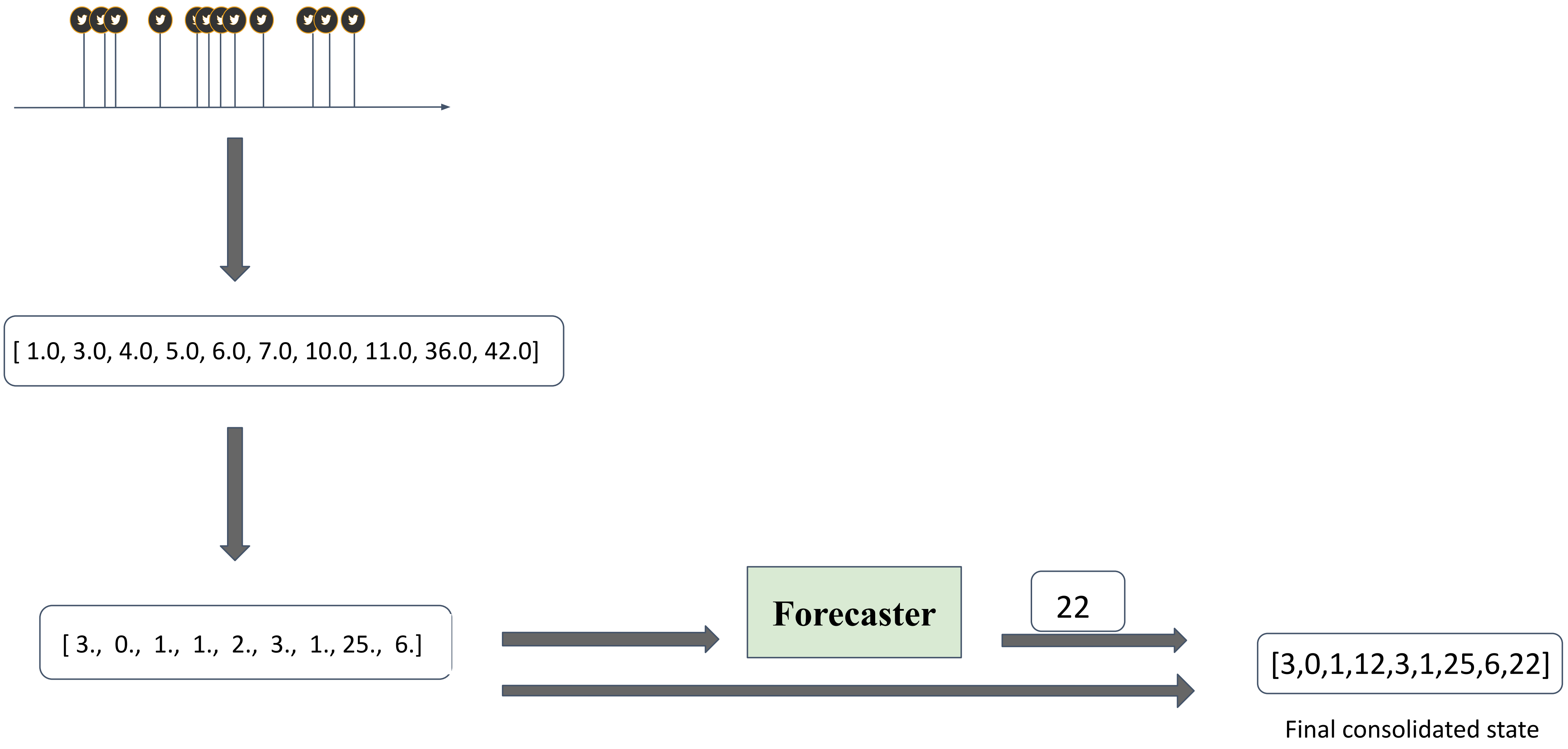
[3., 0., 1., 1., 2., 3., 1., 25., 6.]

Forecaster

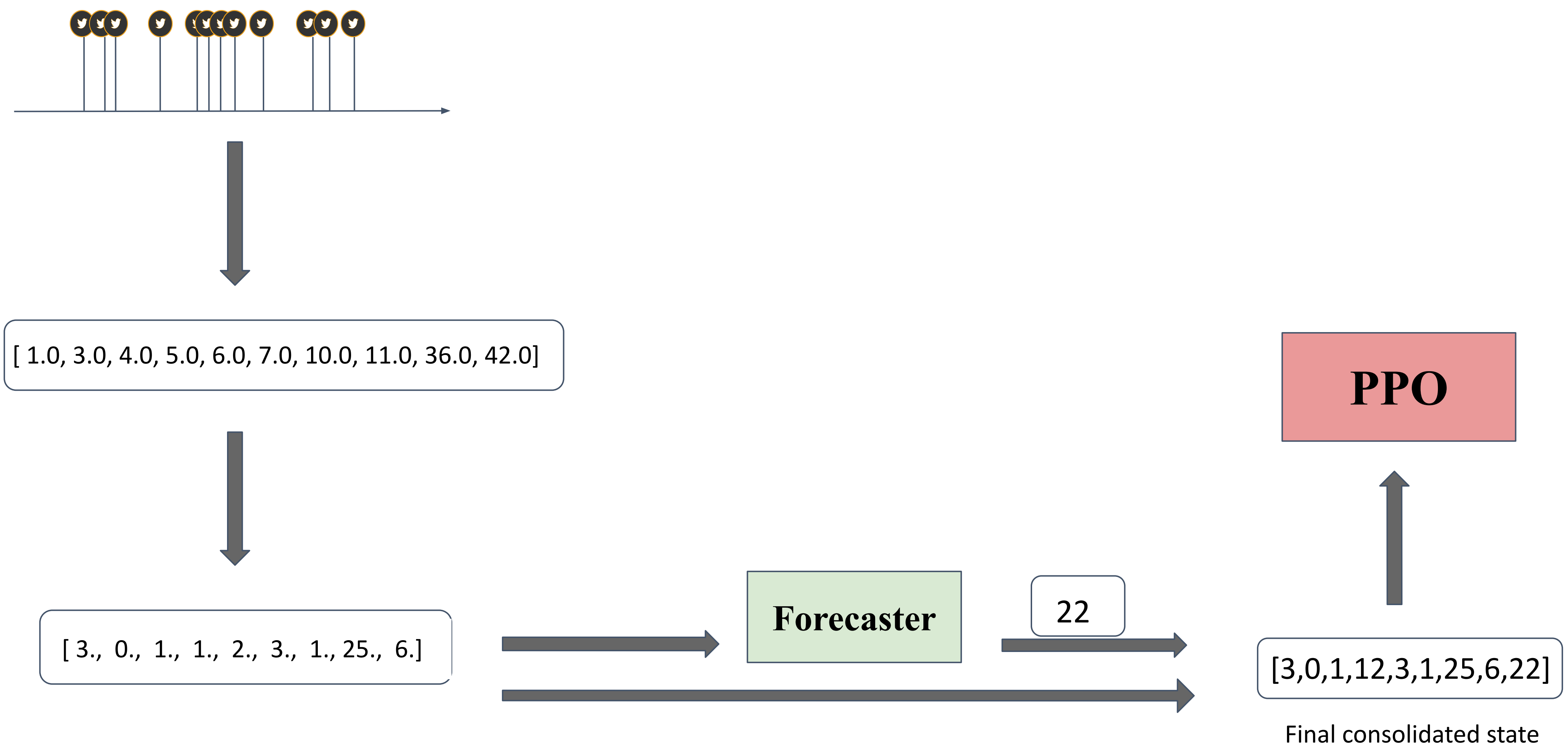
How each timestep works



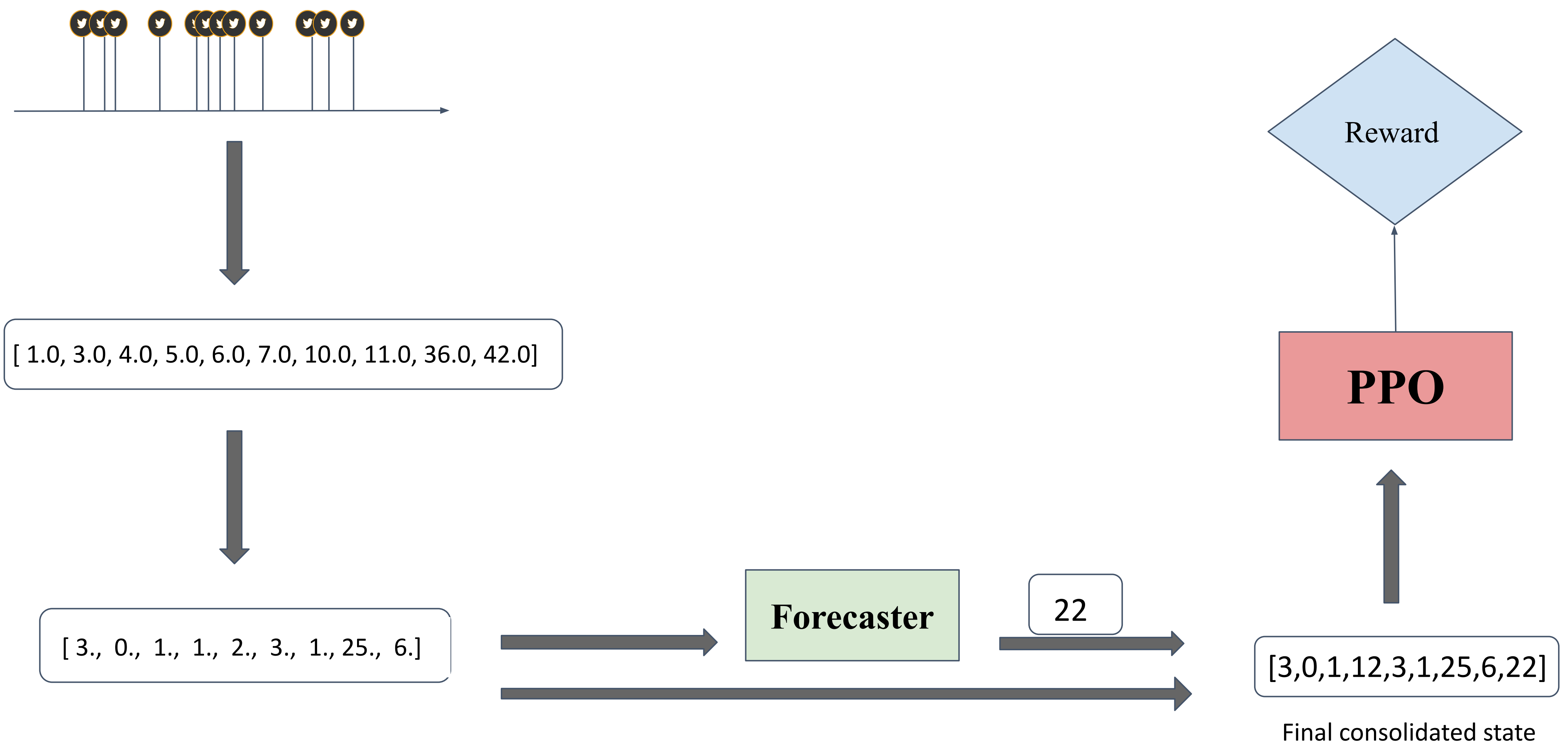
How each timestep works



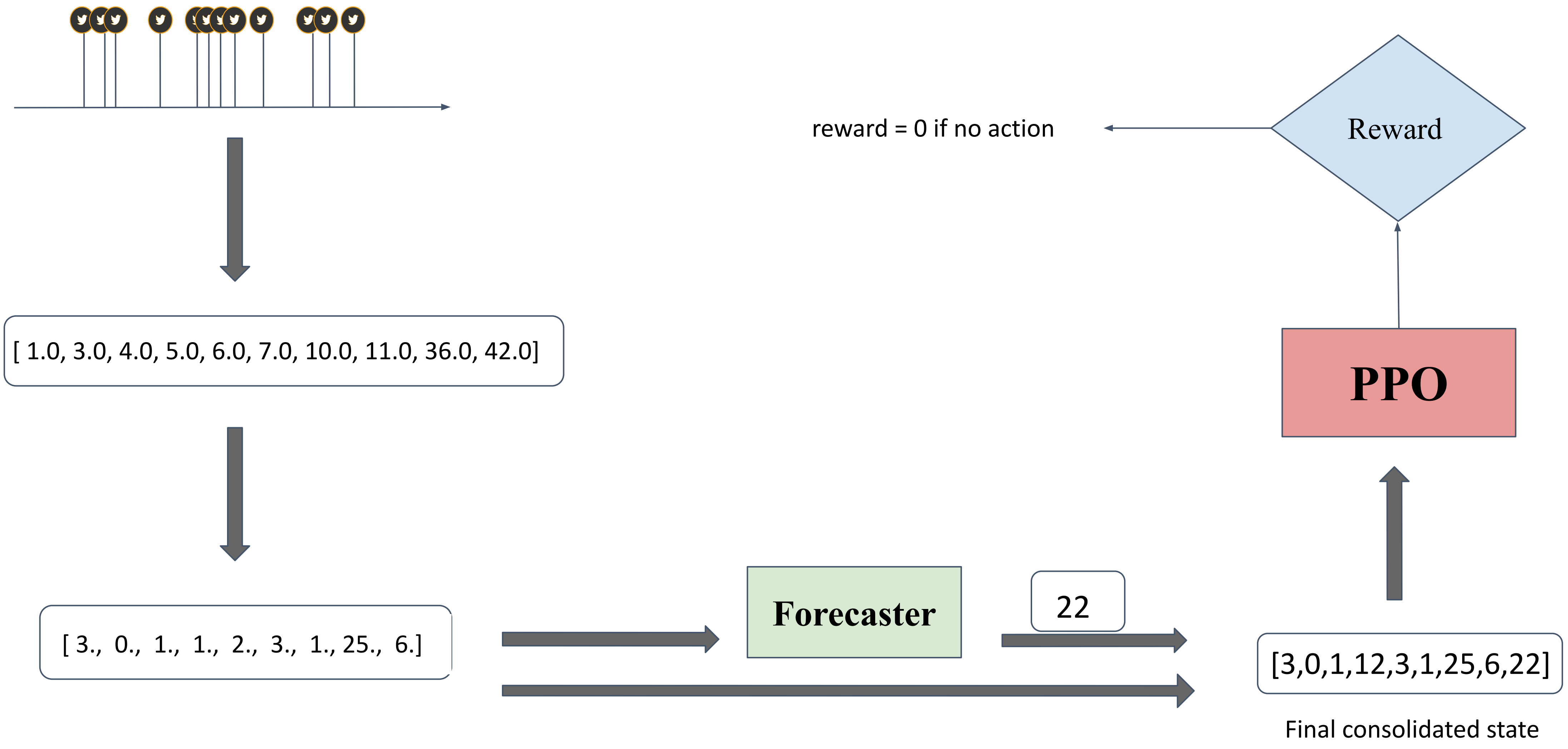
How each timestep works



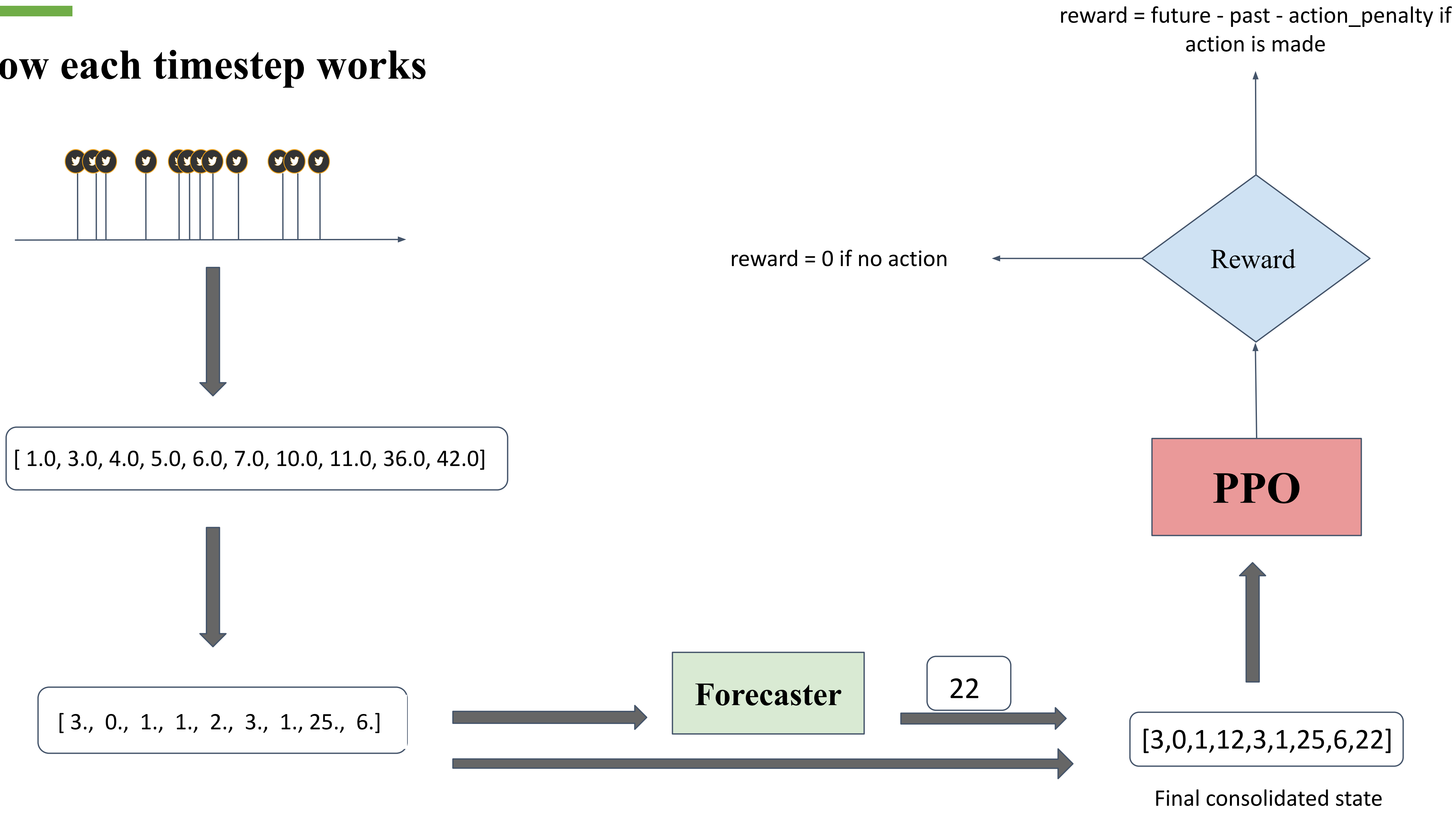
How each timestep works

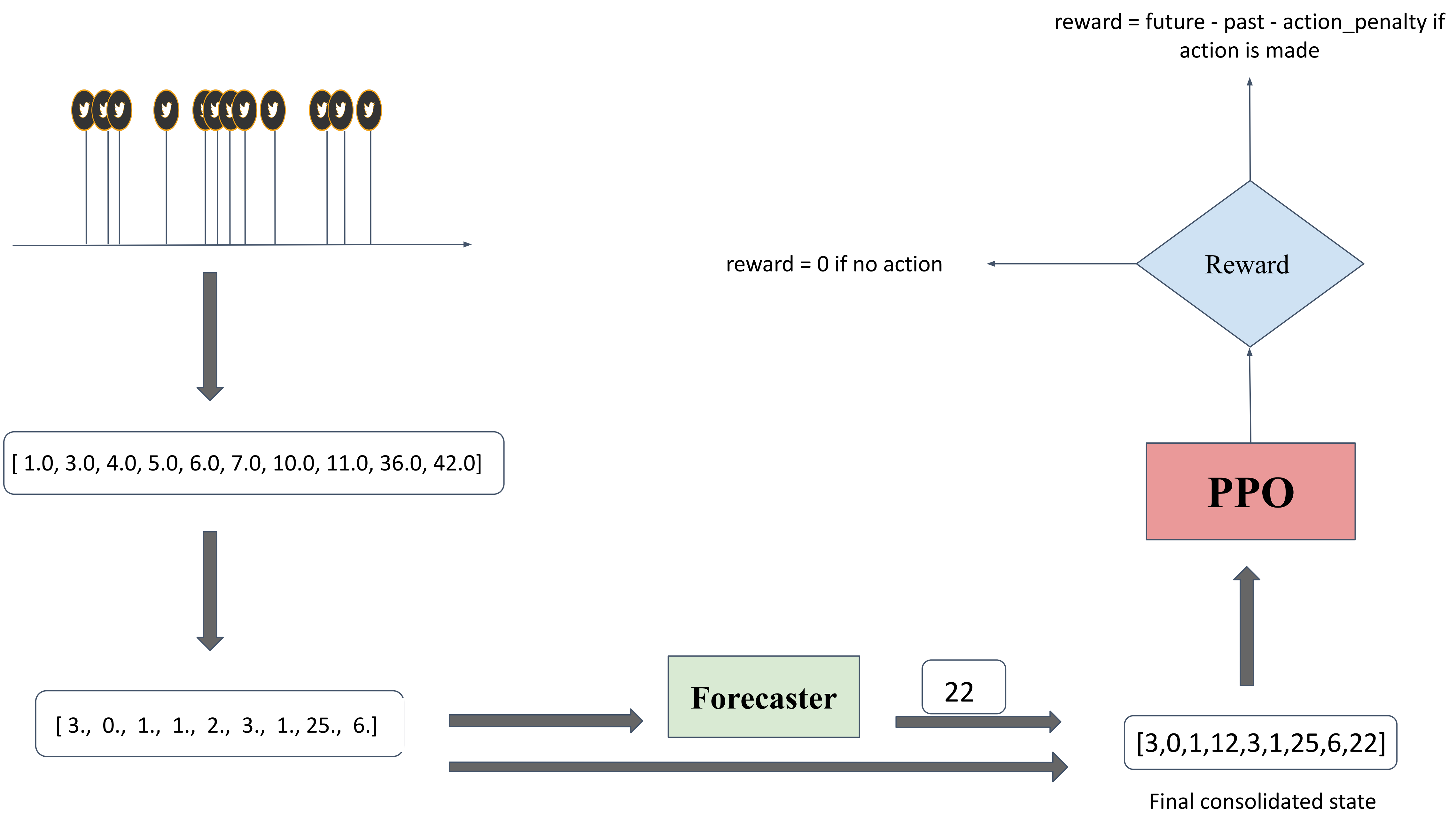


How each timestep works



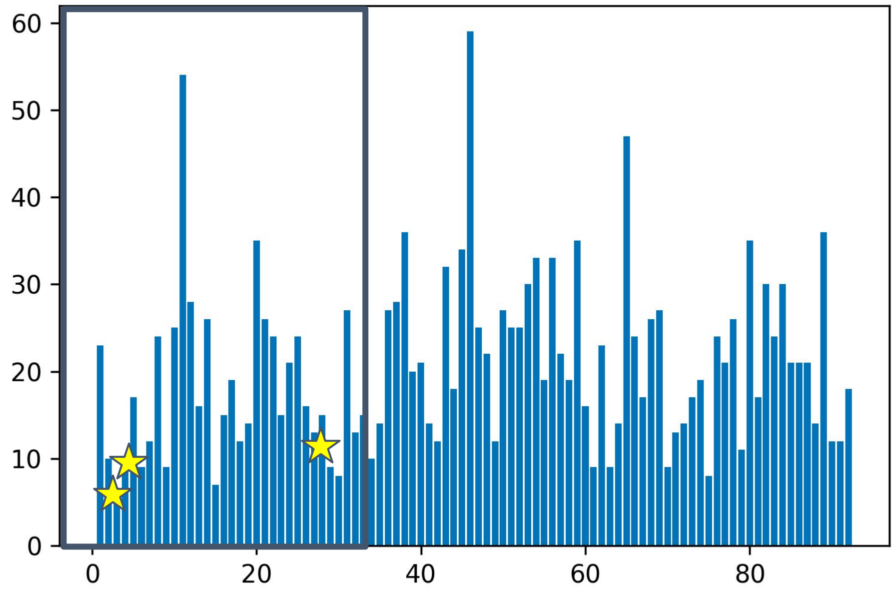
How each timestep works




reward = future - past - action_penalty if action is made
reward = 0 if no action
Reward
[1.0, 3.0, 4.0, 5.0, 6.0, 7.0, 10.0, 11.0, 36.0, 42.0]
PPO
Forecaster
22
[3., 0., 1., 1., 2., 3., 1., 25., 6.]
[3,0,1,12,3,1,25,6,22]
Final consolidated state

Results

```
{'current_step': 35, 'current_post_time': 231.0, 'next_post_time': 284.0}
reward: 2.5704999999999956
{'current_step': 54, 'current_post_time': 416.0, 'next_post_time': 465.0}
reward: -1.4295000000000044
{'current_step': 511, 'current_post_time': 2707.0, 'next_post_time': 2752.0}
reward: -5.429500000000004
episode end
```



CONCLUSION

Conclusion

Contributions	How the Framework achieves them
1. A custom RL environment that can work with temporal point processes, and can **accommodate the needs of various influence maximization strategies.**	➔ By adding a layer of customization over the API like action penalty, max allowed action and time limit, many high-level posting strategies are possible
2. The RL environment enables the use of simple policy models in discrete domain so that they can be **implemented with ease by application developers & researchers.**	➔ By implementing an environment with preprocessing techniques that make point processes suitable for simple in-built agents like PPO, DQN, and A2C.

Limitations

- ➔ Marks could be added for additional complexity of strategies
- ➔ The API could be extended to enable custom reward function (this could greatly extend the application)
- ➔ Various other preprocessing techniques could be investigated

Future scope

There can be several applications that can be built on top of Optimis:

- ➔ Fake news mitigation as a direct consequence of influence maximisation
- ➔ Viral marketing applications

References

1. *Automated Fact Checking. en.URL:https://fullfact .org/about/automated/(visited on 03/15/2021)*

2. *Okawa, Maya, et al. "Marked Temporal Point Processes for Trip Demand Prediction in Bike Sharing Systems." IEICE Transactions on Information and Systems, vol. E102.D, no. 9, Sept. 2019, pp. 1635–43. DOI.org (Crossref), doi:10.1587/transinf.2018OFP0004.*

3. *EconomistGrant.EconomistGrant/HTFE-Tensortrade. 2019. 2021. GitHub, https://github.com/EconomistGrant/HTFE-tensortrade.*

4. *Blad, C., et al. "Control of HVAC-Systems with Slow Thermodynamic Using Reinforcement Learning." Procedia Manufacturing, vol. 38, 2019, pp. 1308–15. DOI.org (Crossref), doi:10.1016/j.promfg.2020.01.159.*

5. *Richard S. Sutton and Andrew G. Barto. Reinforcement learning: an introduction. Second edition. Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, 2018. ISBN: 9780262039246.*

THANK YOU