TMLL

Automated Performance Diagnosis through ML-Driven Trace Analysis

Kaveh Shahedi, Matthew Khouzam, Heng Li

Ericsson Canada, Polytechnique Montreal

This should be 05, but I don't have access to change it anymore :((the day is coincidentally correct)

What is Trace Compass?

Trace Compass is a tool for solving **performance** and **reliability** issues by **analyzing** system **traces**, providing user-friendly **views**, **graphs**, and **metrics**.

What is Trace Server?

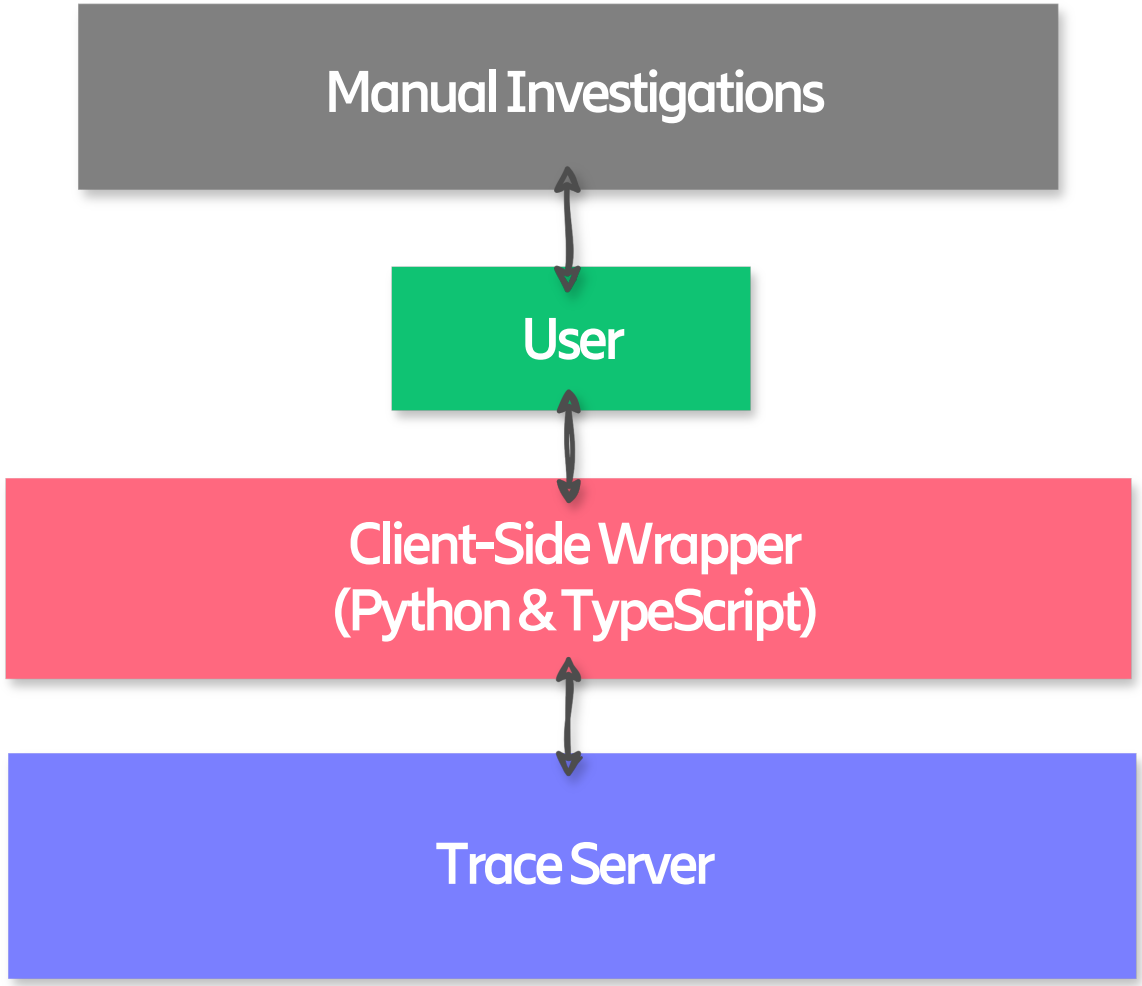
An **open-source tool** for **analyzing logs** and **traces**, enabling users to interact via **client-side wrappers** in **Python** and **TypeScript**, and providing **various analyses** and **outputs** without requiring direct use of Trace Compass.

Think of it as the core of Trace Compass, but without the UI

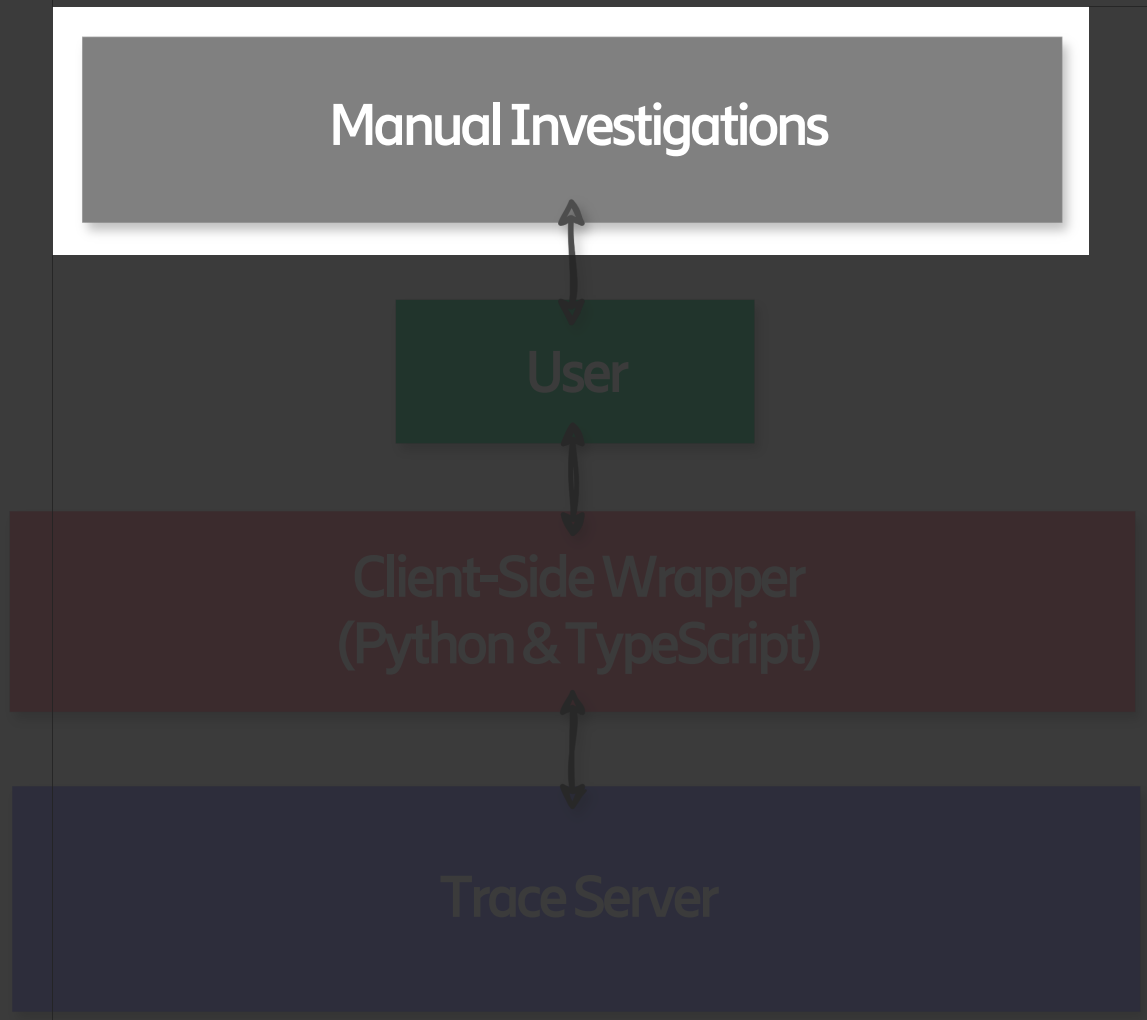
What areas can be investigated with TS?

- Identifying performance bottlenecks
  - CPU usage, disk I/O view, memory usage, scheduler and system call latency
- Diagnosing synchronization issues
  - Futex contention and IRQ latency
- Analyzing system and application behavior
  - Events table, flame chart call stack, thread and resources statuses
- Optimizing memory usage
  - Memory usage, memory latency
- Detailed latency analysis
  - Futex contentions, IRQ, schedulers, system calls, call stack

What is the procedure right now?



What is the procedure right now?



Issues and Downsides

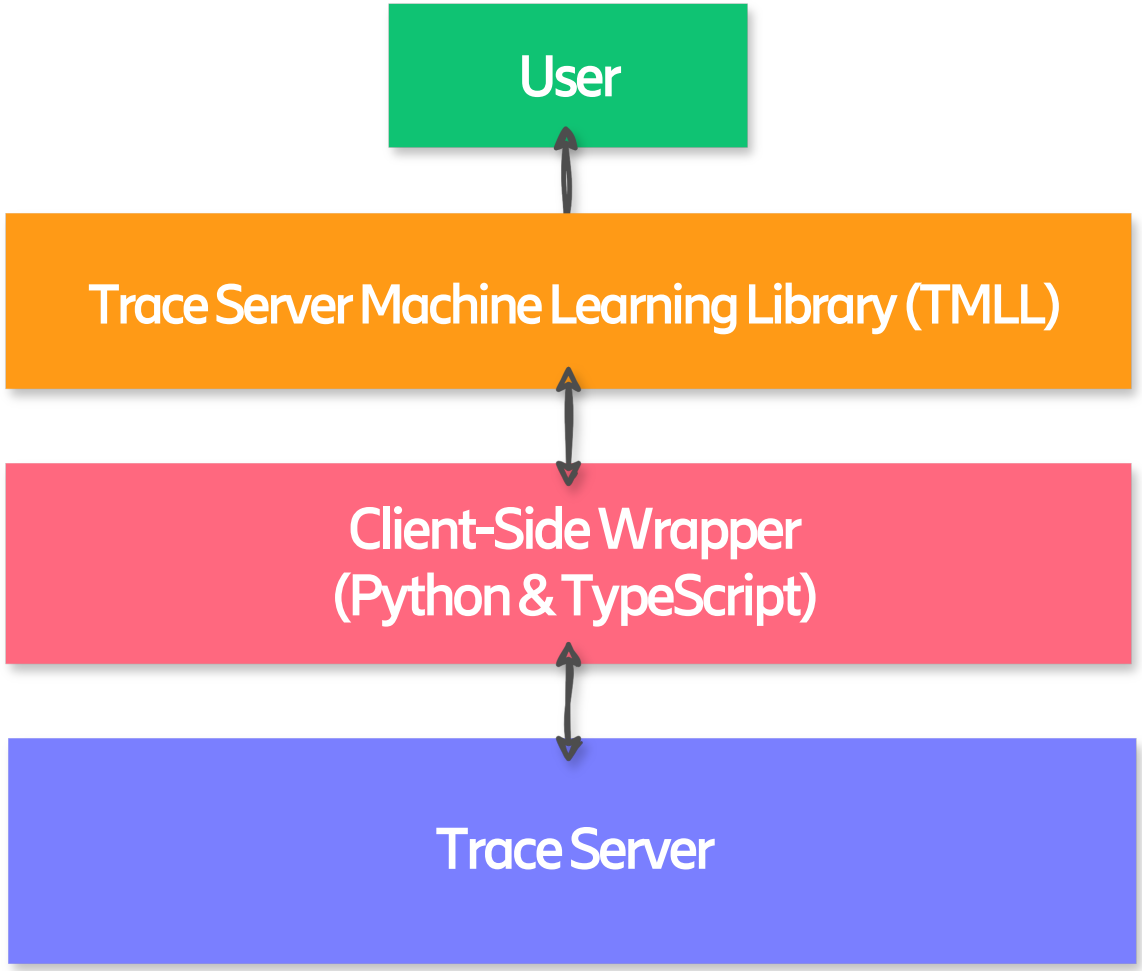
Trace Server

- User should know How to Get
- User should know What to Get

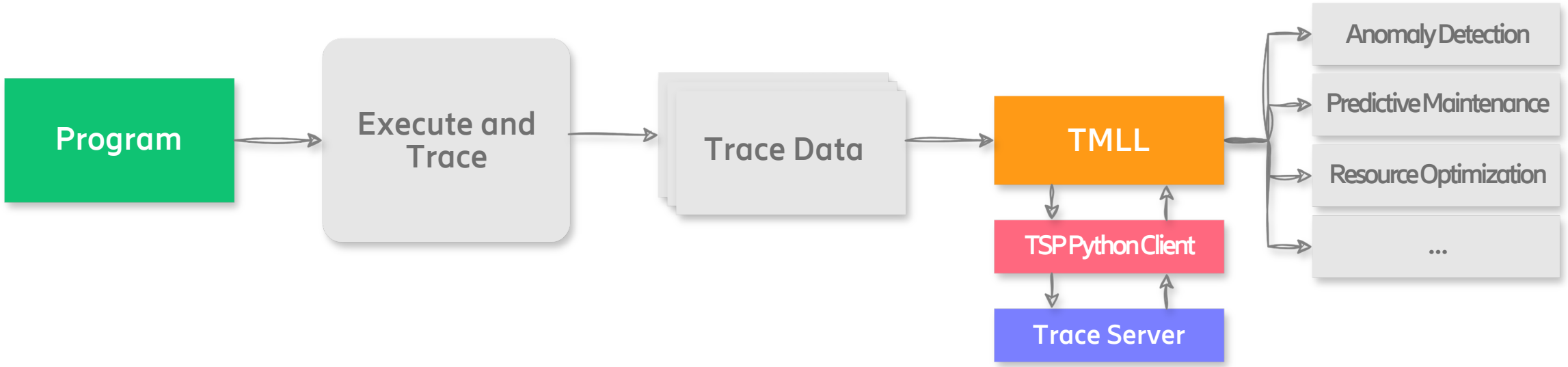
AI/ML

- User should know What to Use
- User should know How to Use
- User should know How to Analyze

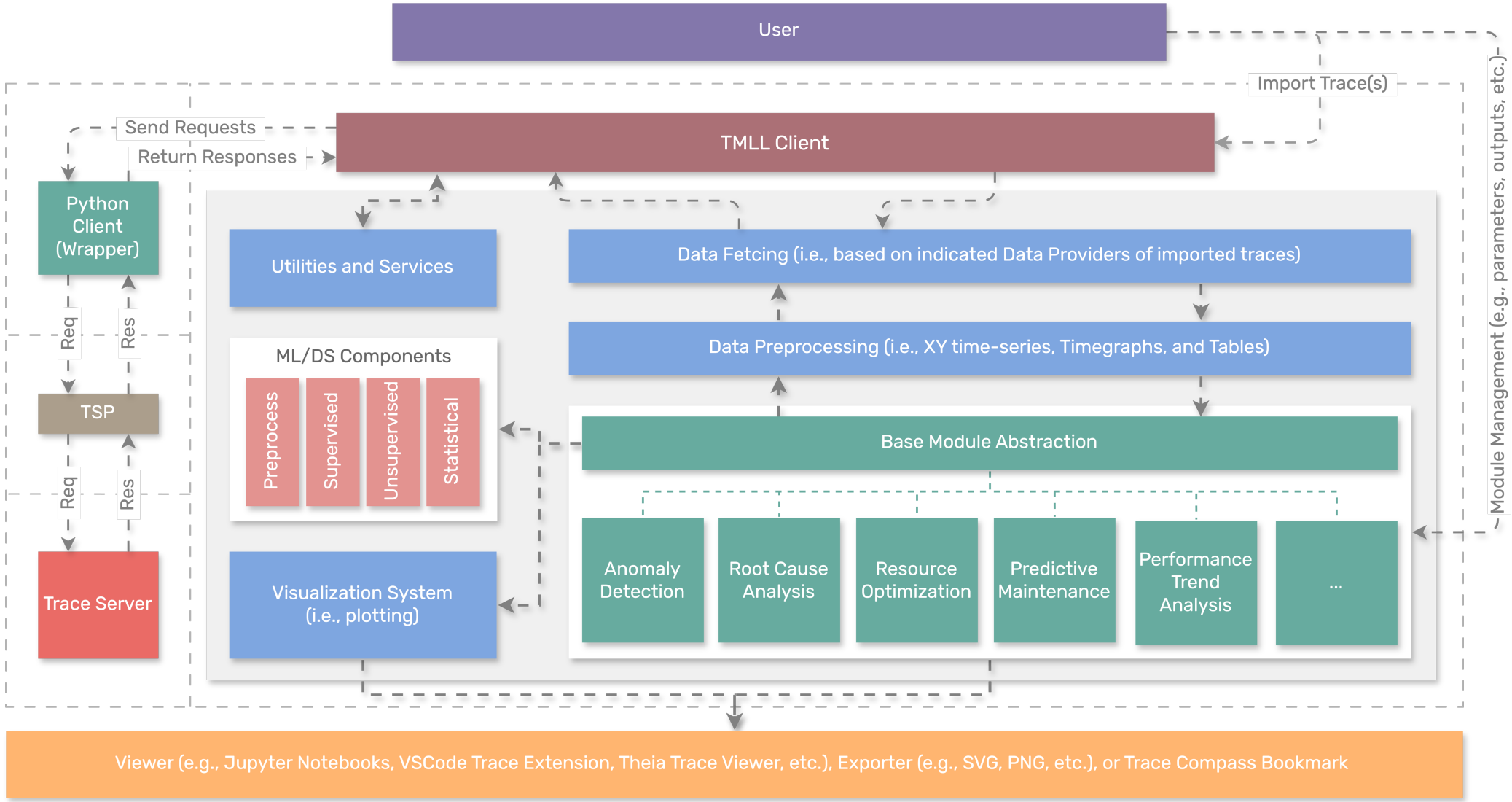
What are we proposing?



TMLL Overview



TMLL Architecture



TMLL Features and Modules

Anomaly Detection

- Identify unusual patterns in system
- Provide potential regions of interest

Resource Optimization

- Optimize system resource allocation
- Analyze which areas can be optimized

Predictive Maintenance

- Predict and prevent system failures
- Done in real-time analysis scenarios

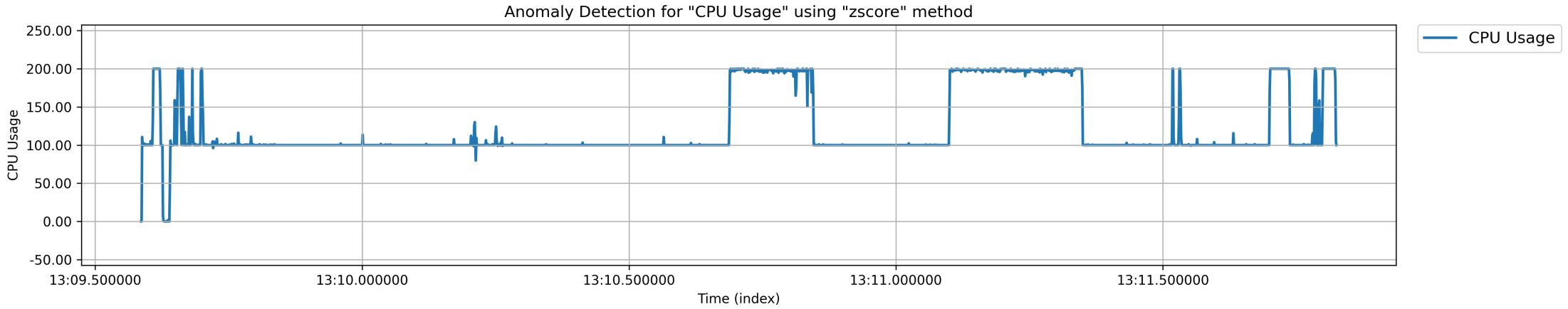
Performance Trend Analysis

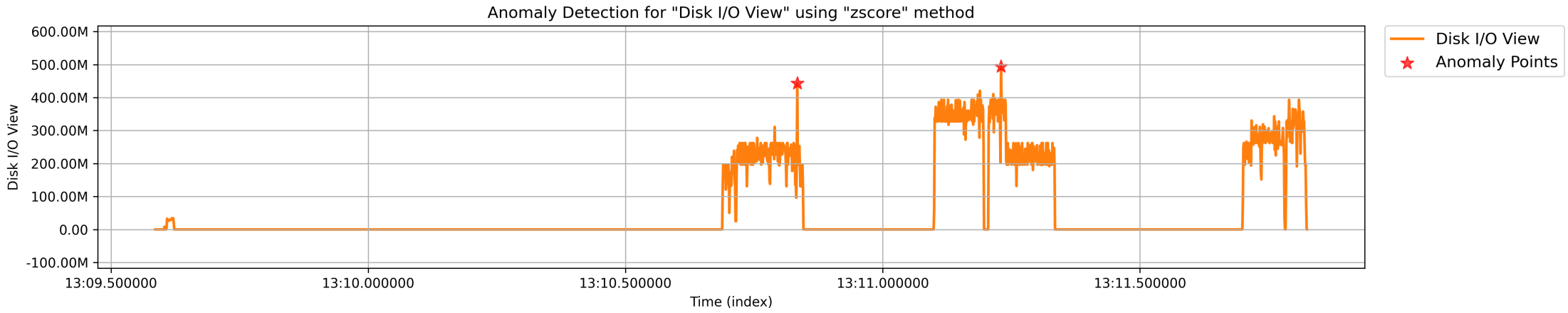
- Understand long-term performance trends
- Provide statistical insights on how system evolves

Scenario 1

Something weird is happening, which is out of our expectations...

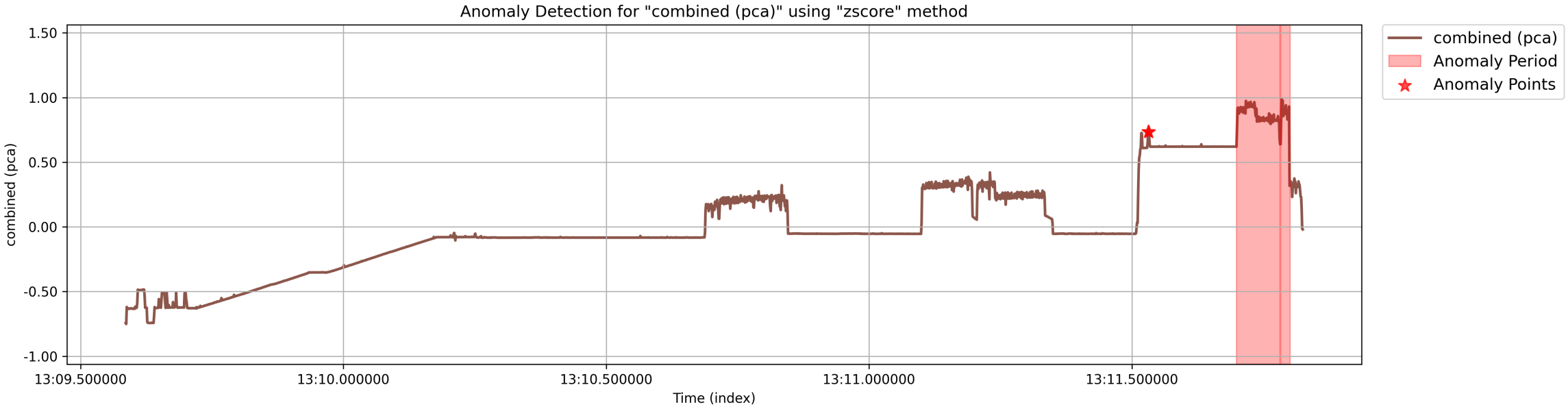
Anomaly Detection





Anomaly Detection (cont'd)

Anomaly Detection (cont'd)

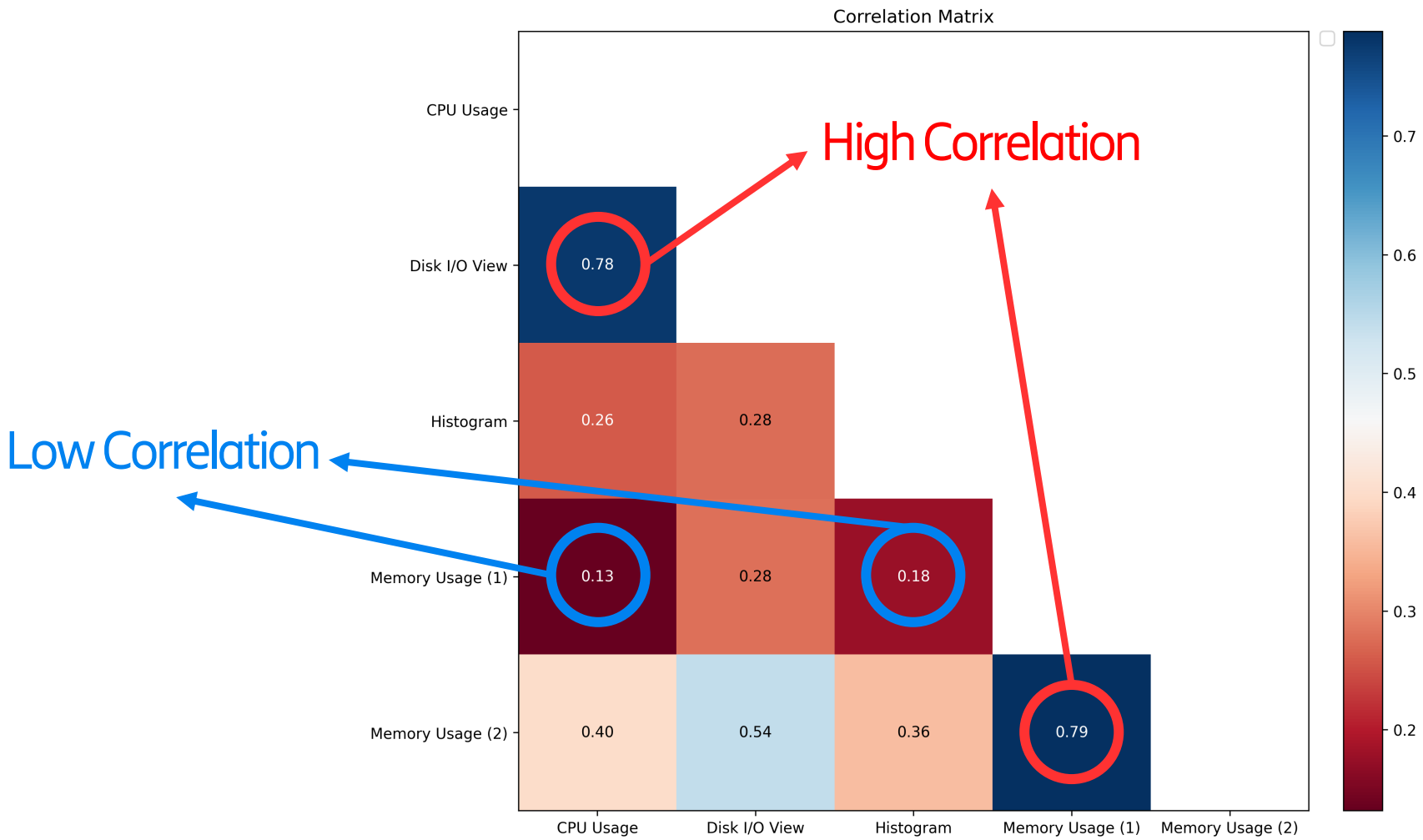


Potential Anomaly Period: 2024-04-24 02:13:**11.796**169984 to 2024-04-24 02:13:**11.890**216704

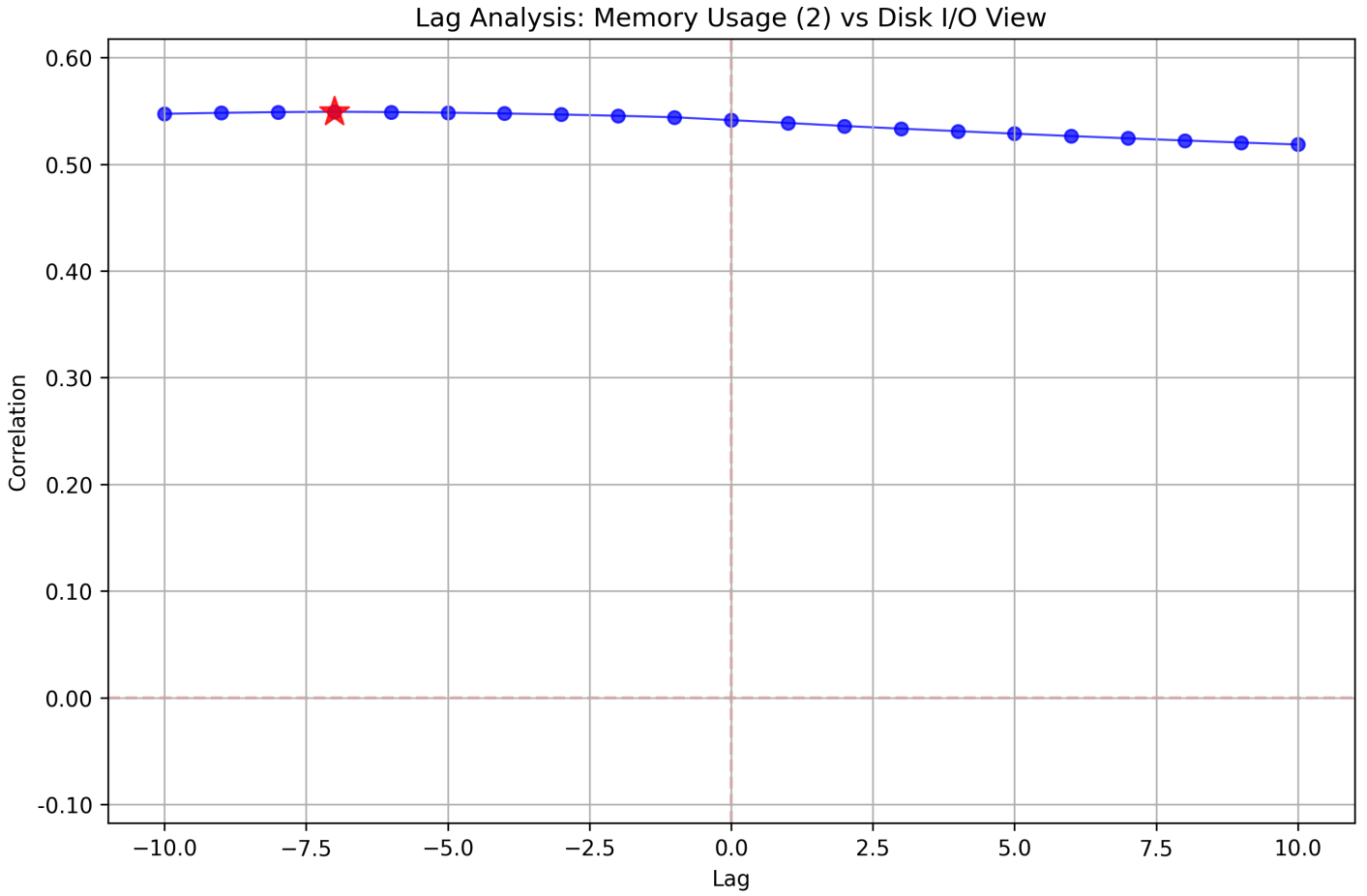
Scenario 2

How are my system components interacting with each other?

Correlation Analysis



Correlation Analysis (Lag Analysis)



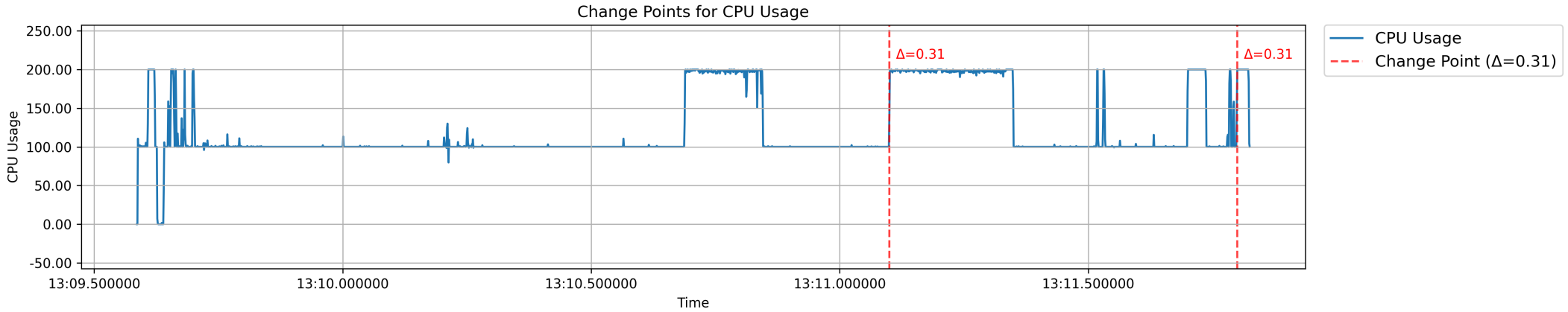
The highest correlation shows up with 7 steps difference (i.e., 7ms in this case as the resampling frequency is 1ms)

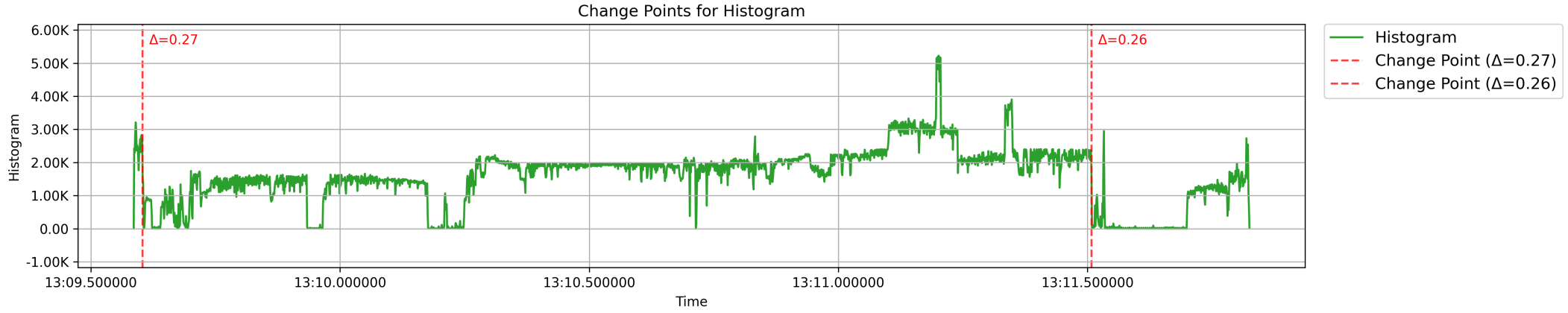
e.g.: Memory usage increases, after 7ms, the disk usage also begins to rise

Scenario 3

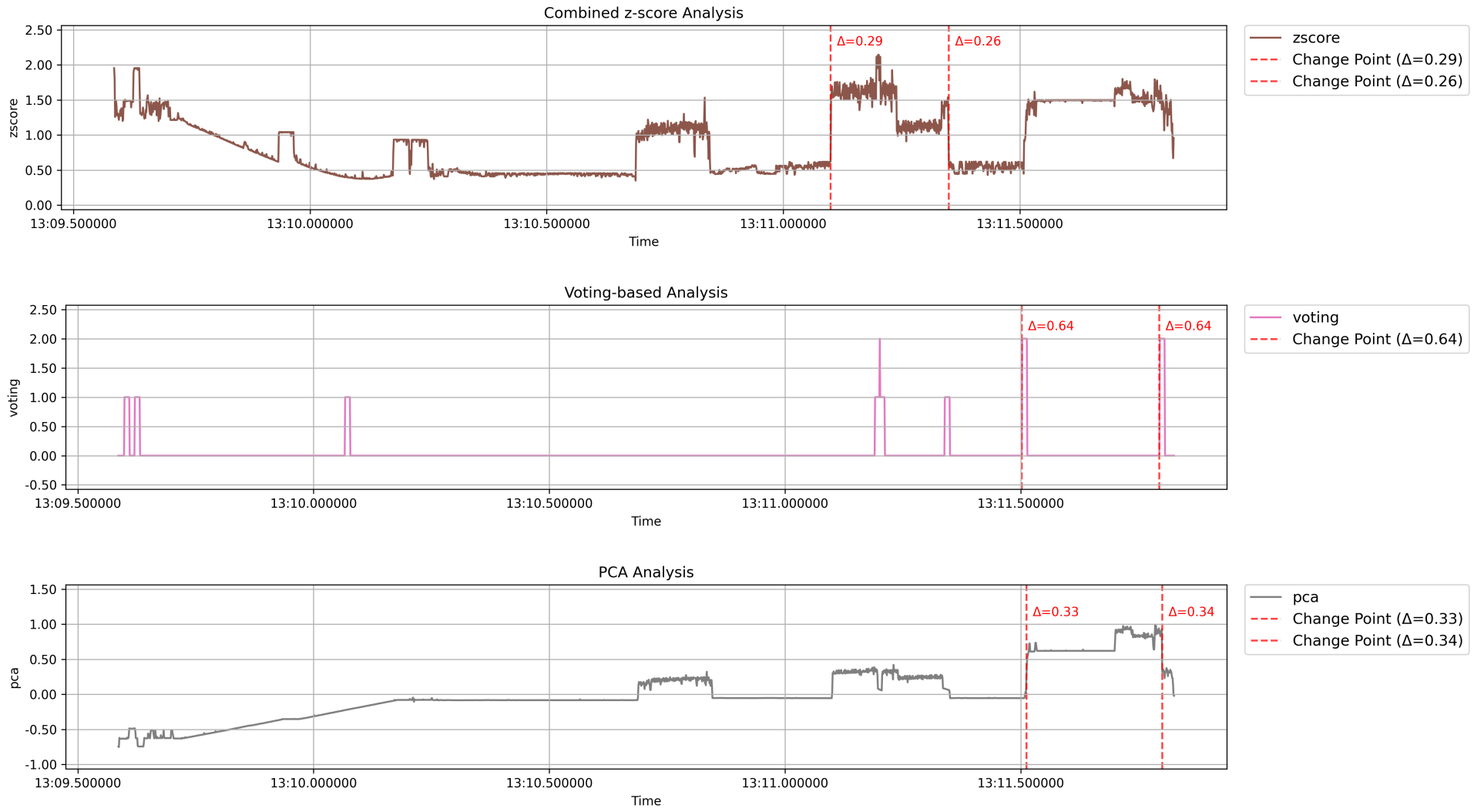
No significant internal/external factors have changed, yet the system's behavior has altered at an unknown point in time...

Change Point Detection





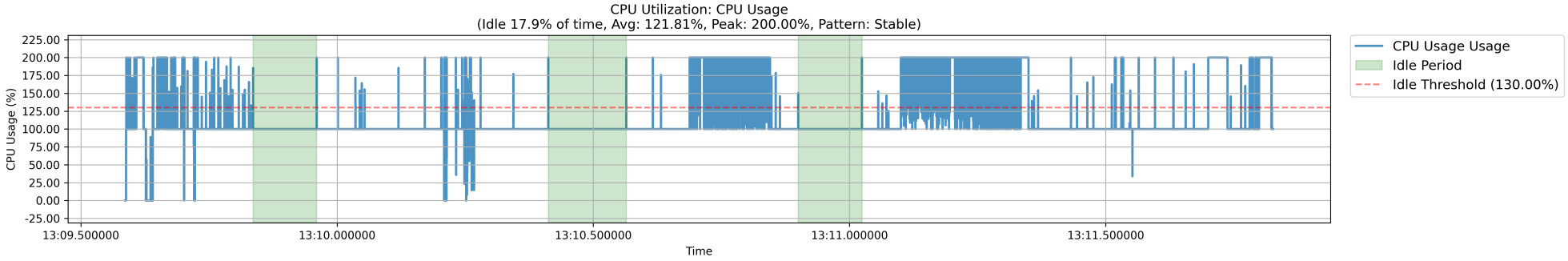
Change Point Detection (cont'd)

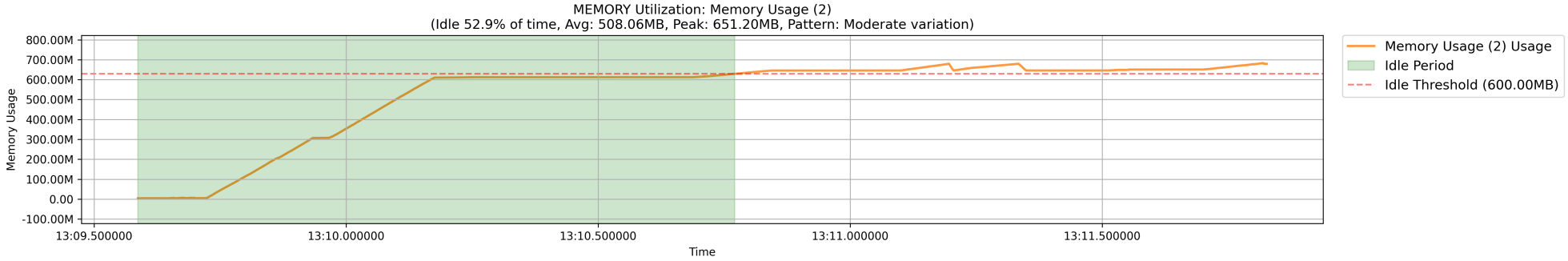


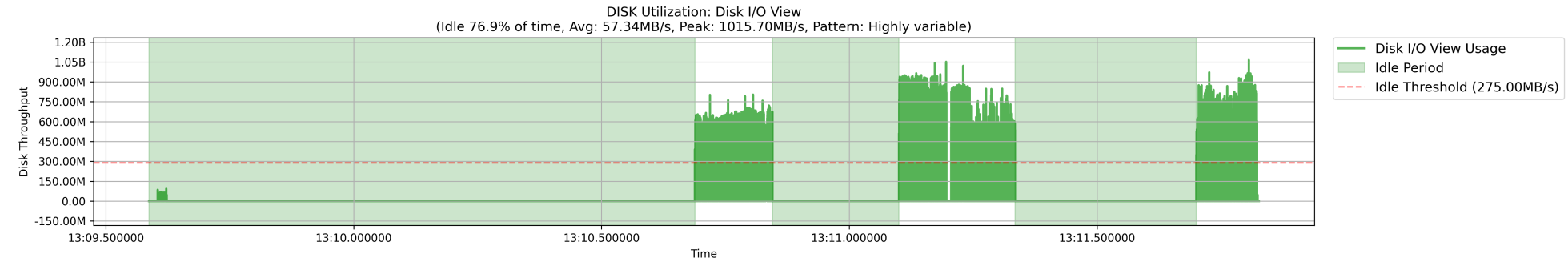
Scenario 4

Are there significant idle times in my system components? I want to save up some money...

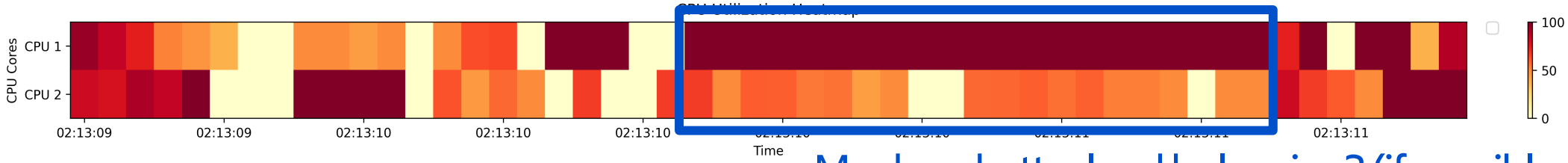
Idle Resource Detection



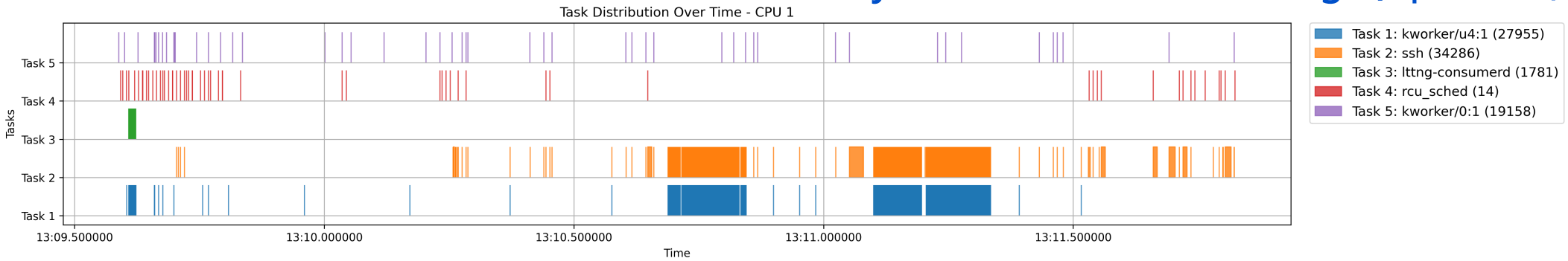


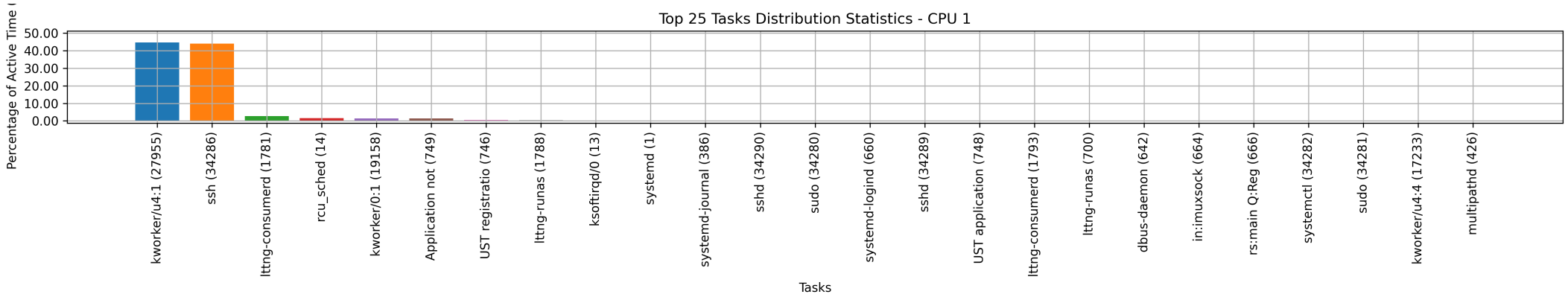


Idle Resource Detection (CPU Scheduling)



Maybe a better load balancing? (if possible, ofc)





Scenario 5

I'm going on vacation. Should I be concerned about any upcoming workloads while I'm away?

Capacity Planning (i.e., Forecasting)

Scenario X

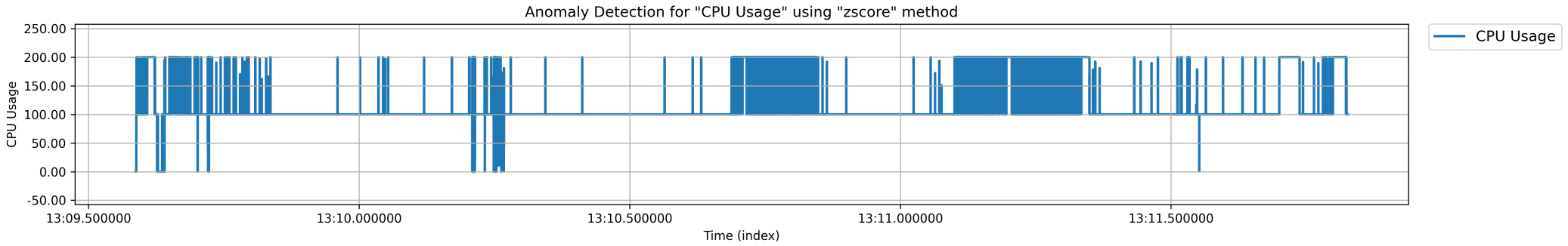
I want to analyze features c_1, c_2, …, c_n to perform tasks t_1, t_2, …, t_n. What can I do?

TMLL is highly extensible. Since the data is now in Python Pandas Data Frames, you can essentially do anything you want!

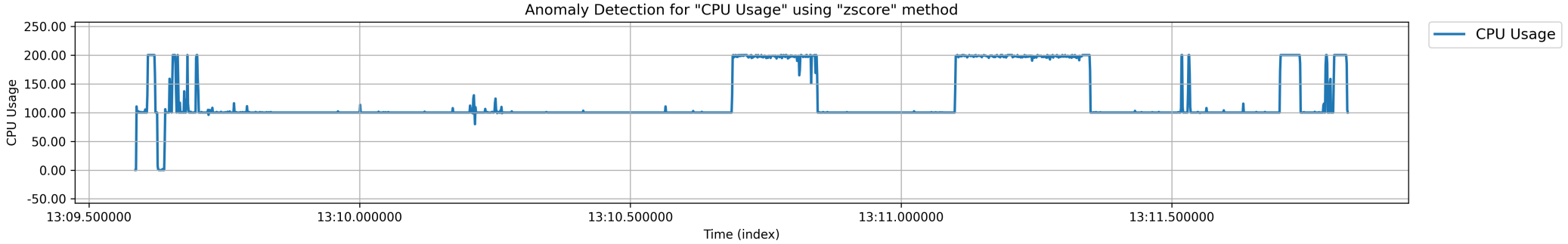
Just checkout the documentations!

Data Density Adjustments

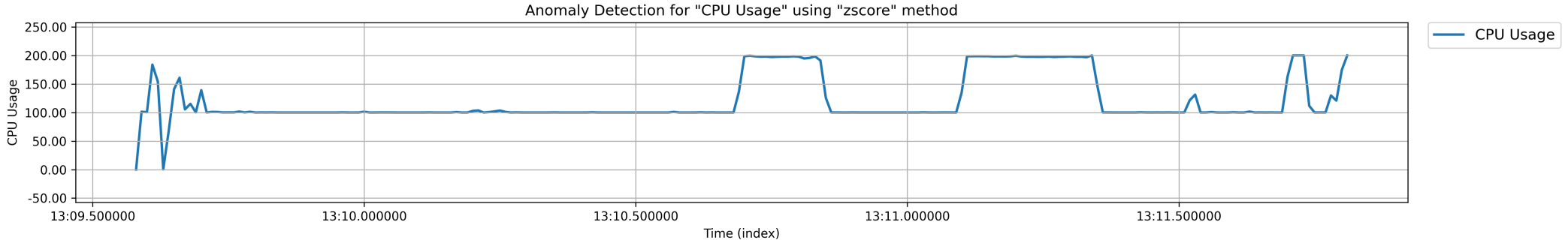
Freq: 5us



Freq: 1ms



Freq: 10ms



How to Use?

main.py

```python
from tmll.tmll_client import TMLLClient
from tmll.ml.modules.anomaly_detection.anomaly_detection_module import AnomalyDetection

# Initialize the TMLL client
client = TMLLClient()

# Create an experiment from trace files
experiment = client.create_experiment(traces=[
    {
        "path": "/path/to/trace/file",
    }
], experiment_name="My Experiment")

# Run anomaly detection
outputs = experiment.find_outputs(keyword=["cpu usage", "memory usage"], type=["xy"])
ad = AnomalyDetection(client, experiment, outputs)
anomalies = ad.find_anomalies(method="iforest", zscore_threshold=3.5)
ad.plot_anomalies(anomalies)
```

Which trace(s)?

Which output(s)?

Which module?