HotCloud'17

Lube:
Mitigating Bottlenecks in Wide Area Data Analytics

Hao Wang*
Baochun Li

UNIVERSITY OF TORONTO
iQua

Wide Area Data Analytics

Flink hadoop Spark

DC

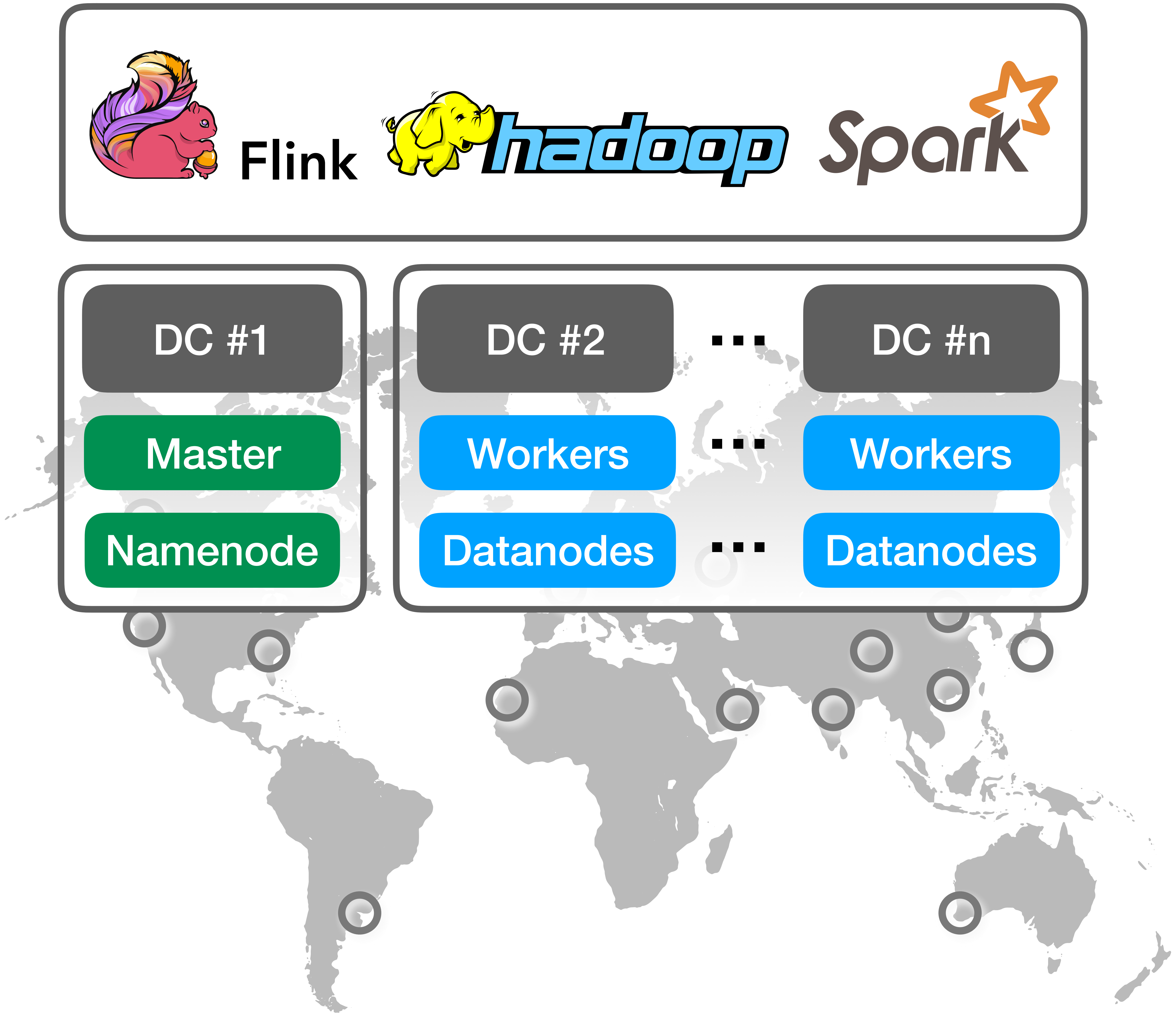
Master

Namenode

Workers

Datanodes

Wide Area Data Analytics



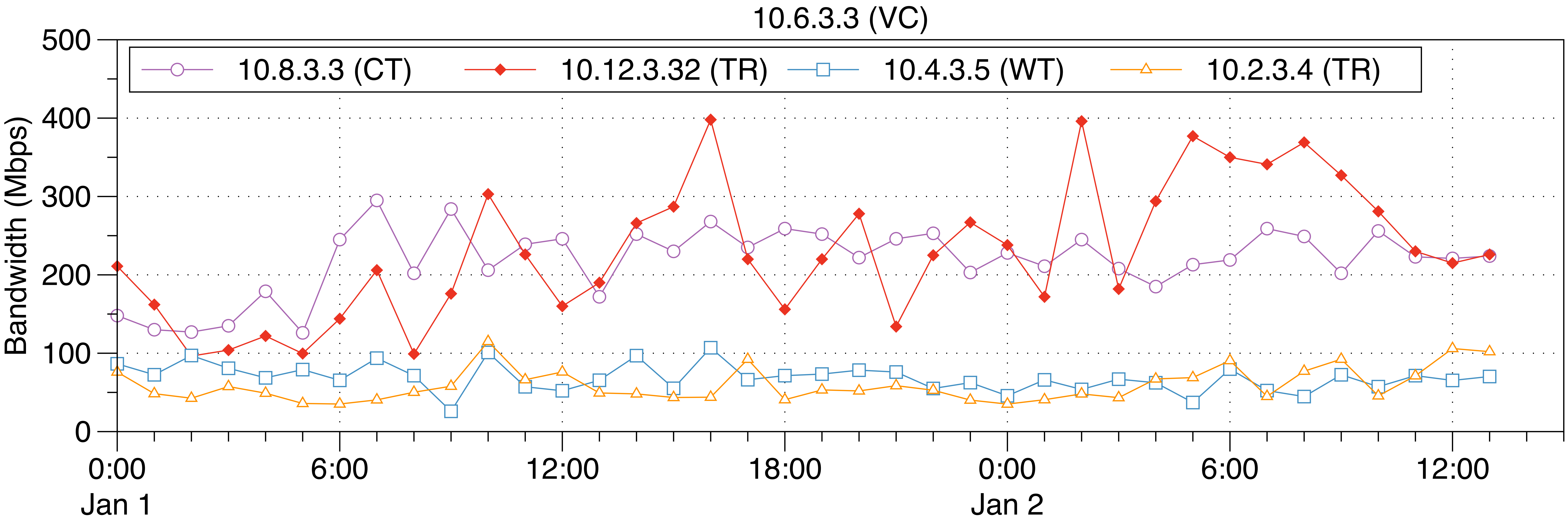
Why wide area data analytics?

- Data Volume
- User Distribution
- Regulation Policy

Problems

- Widely shared resources
  - Fluctuating available provision
- Distributed runtime environment
  - Heterogenous utilizations

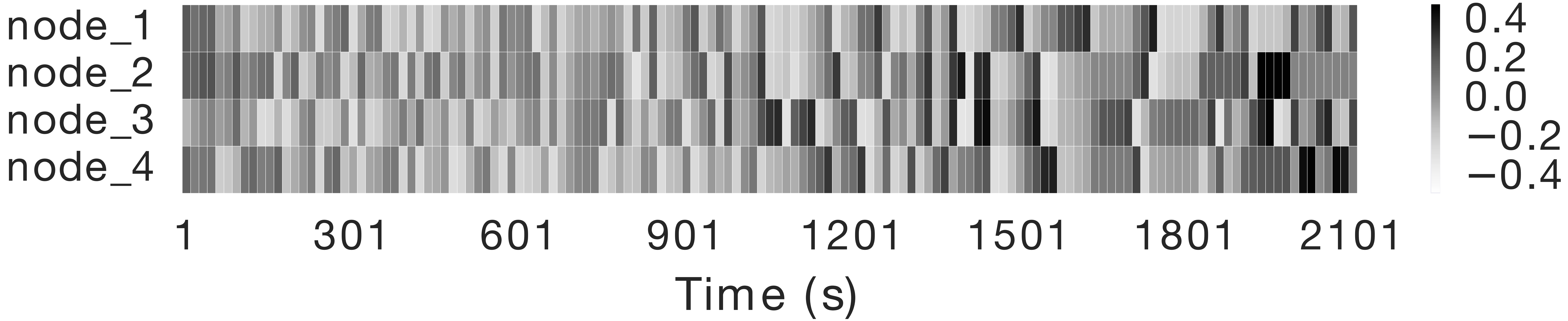
Fluctuating WAN Bandwidths



Measured by *iperf* on SAVI testbed
https://www.savinetwork.ca/

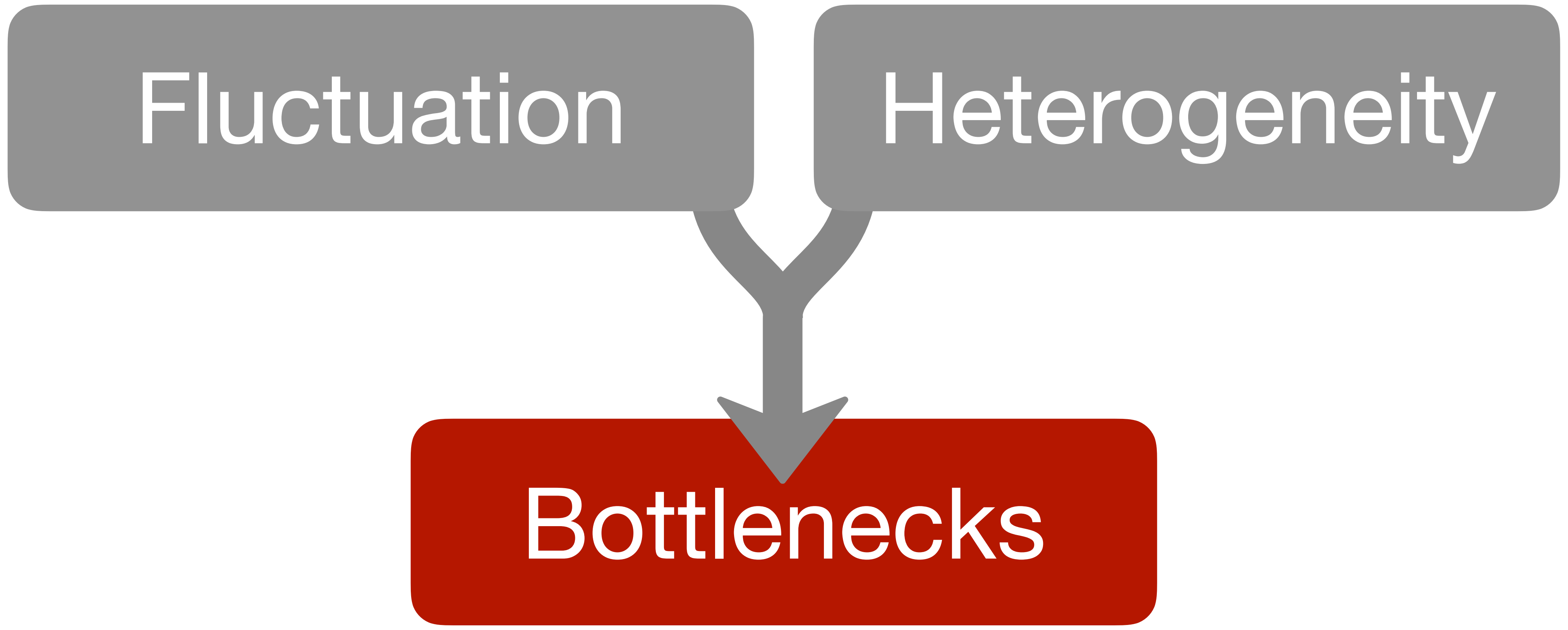
Heterogenous Memory Util

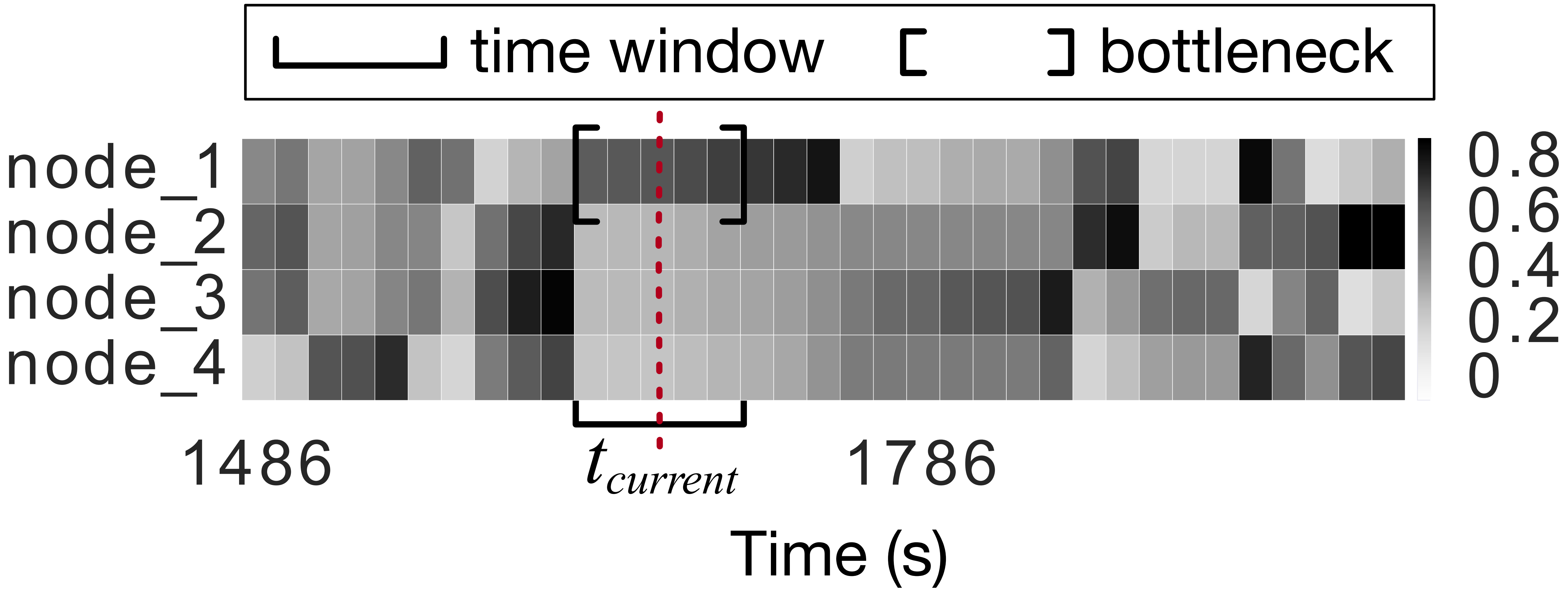
Nodes in different DCs may have different resource utilizations



Running Berkeley Big Data Benchmark
on AWS EC2 4 nodes across 4 regions.
Collected by *jvmtop*

Runtime Bottlenecks





Bottlenecks emerges at runtime

- Any time
- Any nodes
- Any resources

Data analytics performance

- Long completion times
- Low resource utilization
- Invalid optimization

Optimization of Data Analytics

Existing optimization method does not consider runtime bottlenecks

- **Clarient** [OSDI'16] considers the heterogeneity of available WAN bandwidth
- **Iridium** [SIGCOMM'15] trades off between time and WAN bandwidth usage
- **Geode** [NSDI'15] saves WAN usage via data placement and query plan selection
- **SWAG** [SoCC'15] reorders jobs across datacenters

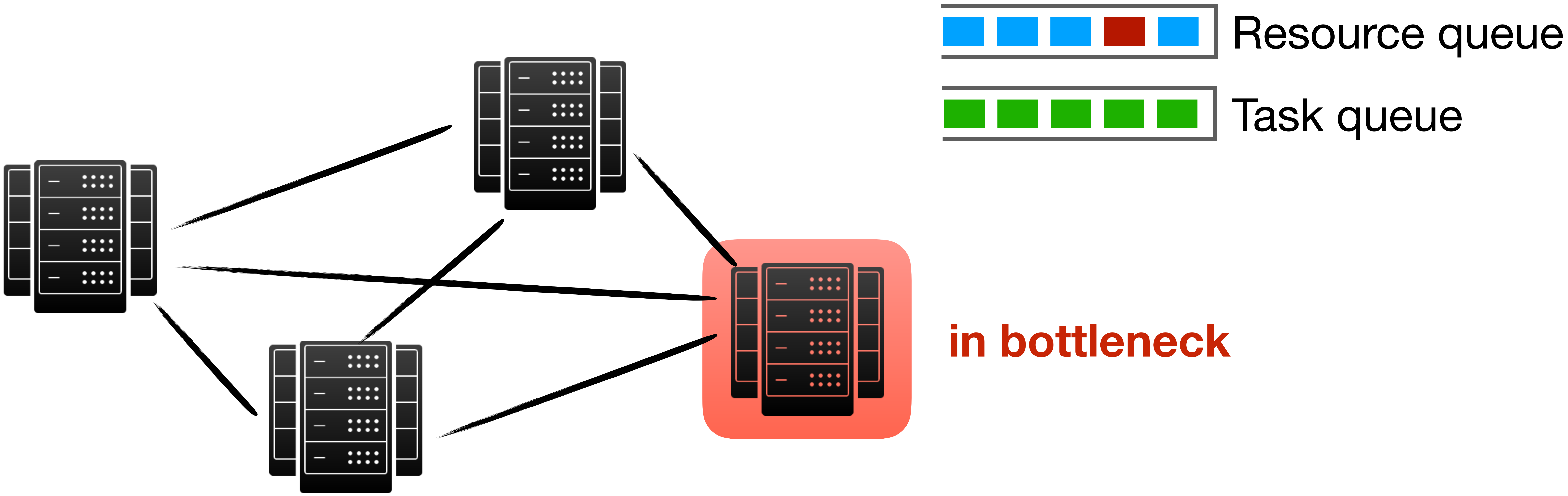
> "Much of this performance work has been motivated by three widely-accepted mantras about the performance of data analytics — **network**, **disk** and **straggler**."

Making Sense of Performance in Data Analytics Frameworks
NSDI'15, Kay Ousterhout

Mitigating Bottlenecks at Runtime

Mitigating bottlenecks

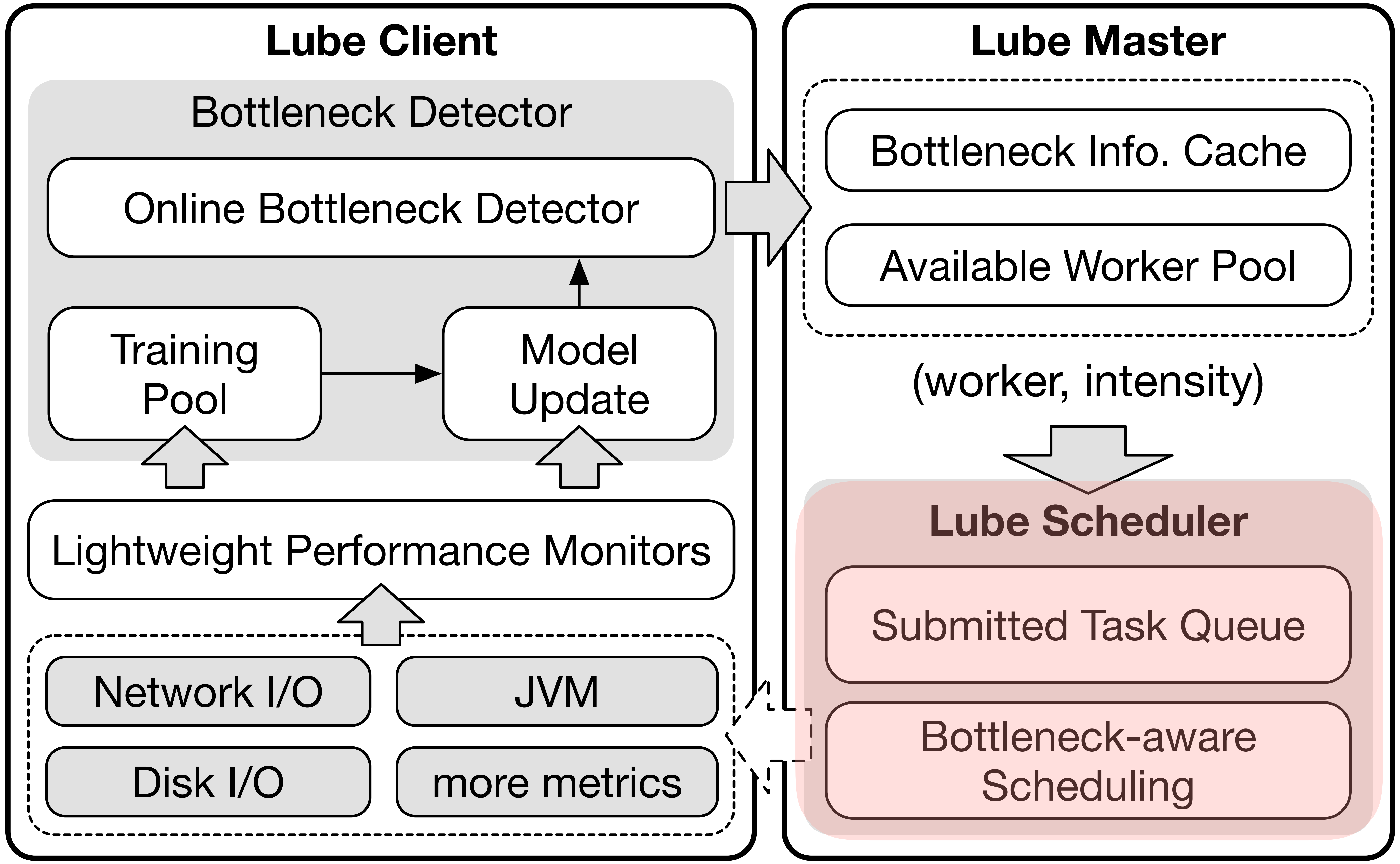
- How to detect bottlenecks?
- How to overcome the scheduling delay?
- How to enforce the bottleneck mitigation?



Architecture of Lube

Three major components

- Performance monitors
- Bottleneck detecting module
- Bottleneck-aware scheduler



Detecting Bottlenecks — ARIMA

$$y_t = \theta_0 + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \epsilon_t - \theta_1 \epsilon_{t-1} - \theta_2 \epsilon_{t-2} - \ldots - \theta_q \epsilon_{t-q}$$

y_t Current state	ϵ Ramdon error	θ ϕ Coefficients

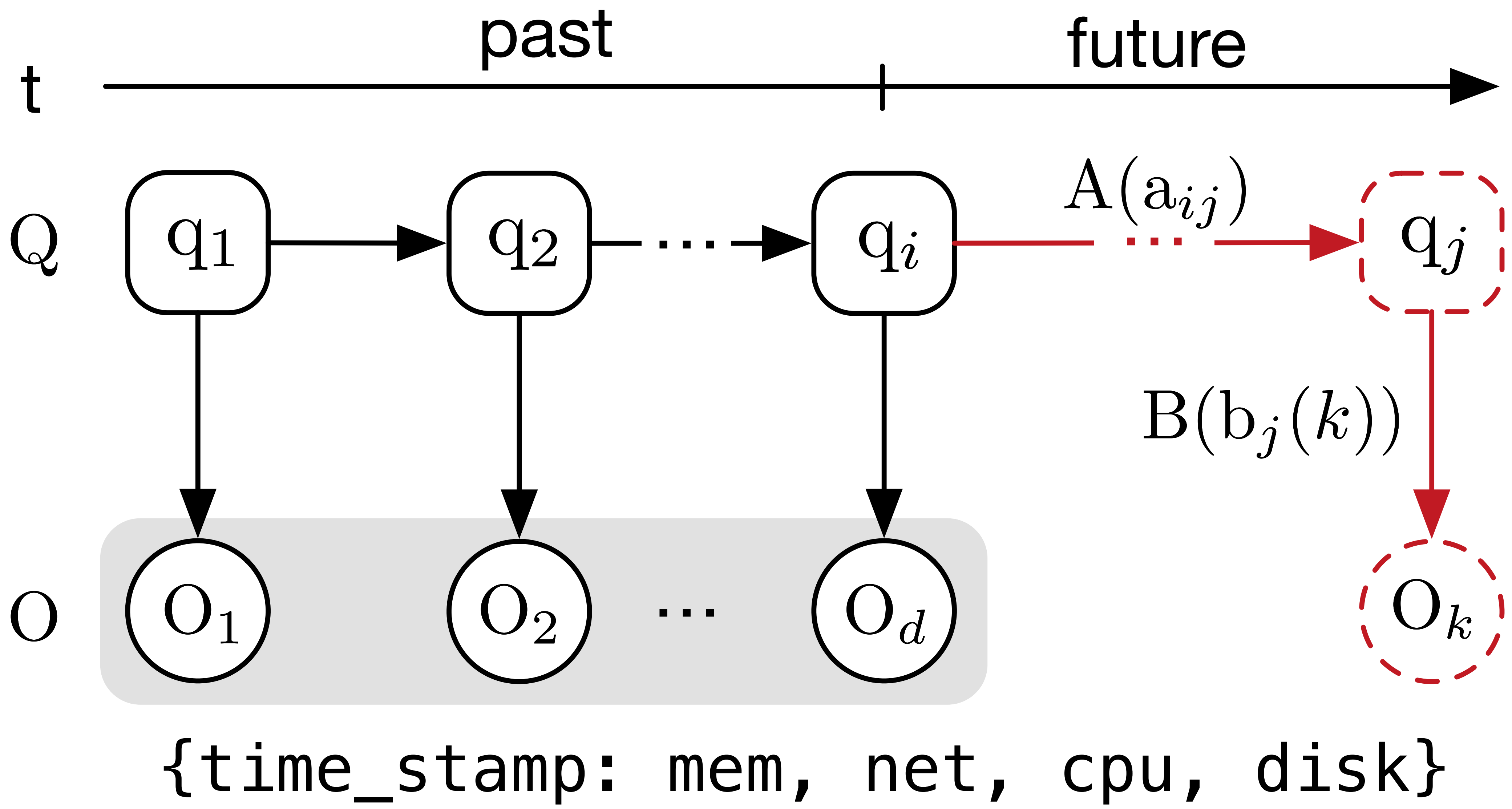
Historical states $\xrightarrow{\text{input}}$ Autoregressive (AR) + Moving Average(MA) $\xrightarrow{\text{output}}$ Current state

```
(time_1, mem_util)
(time_2, mem_util)
…
(time_t-1, mem_util)
```

```
ARIMA(p, d, q)
```

```
(time_t, mem_util)
```

Detecting Bottlenecks — HMM



Hidden Markov Model

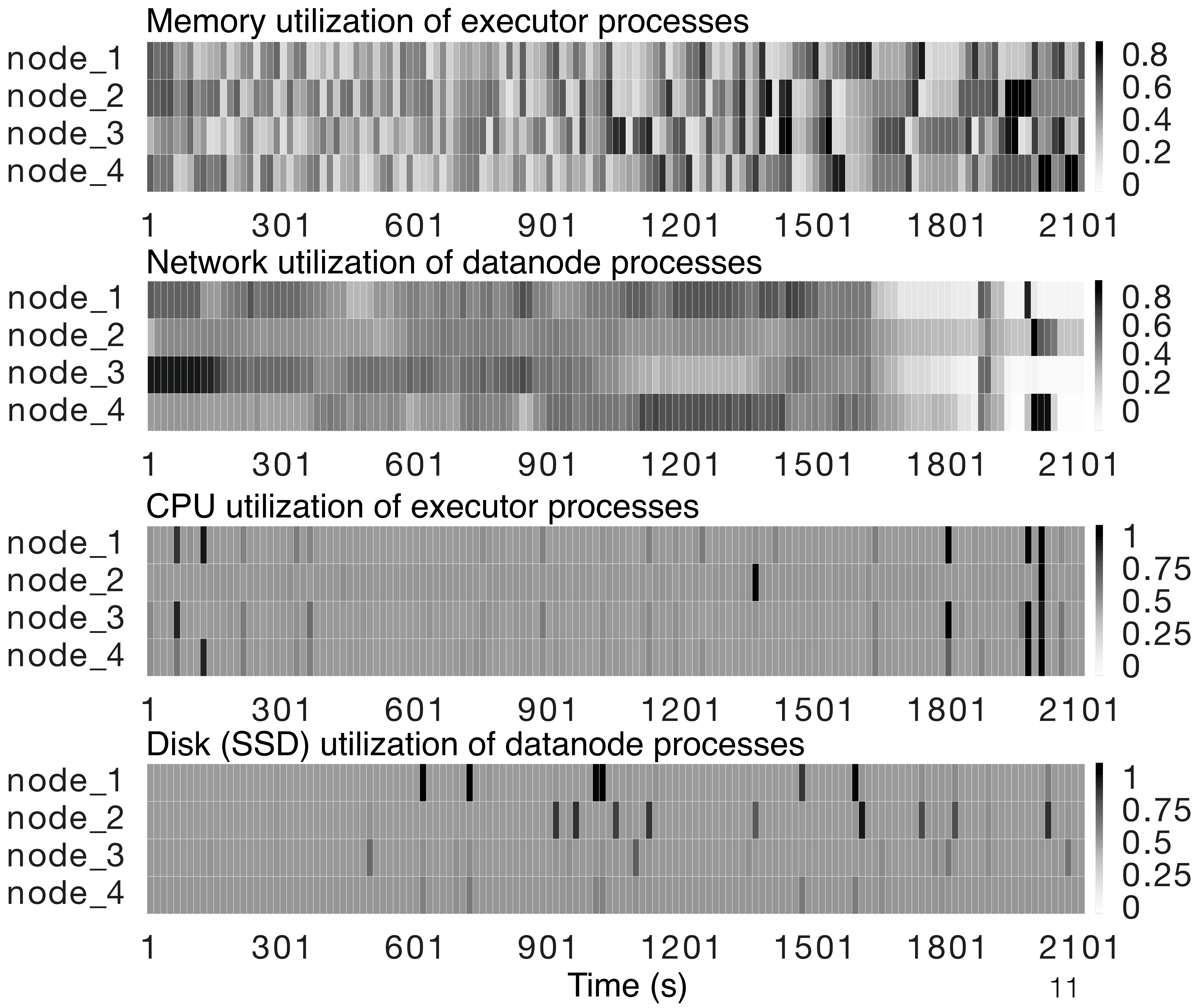
- Hidden states: O
- Observation states: Q
- Emission probability: A
- Transition probability: B

To make HMM online

Sliding Hidden Markov Model

- A sliding window for new observations
- A moving average approximation for outdated observations

Bottleneck-Aware Scheduling



Built-in task schedulers:

- Data-locality

Bottleneck-aware scheduler:

- Data-locality
- Bottlenecks at runtime

A single worker node is bottlenecked continuously while all nodes are rarely bottlenecked at the same time

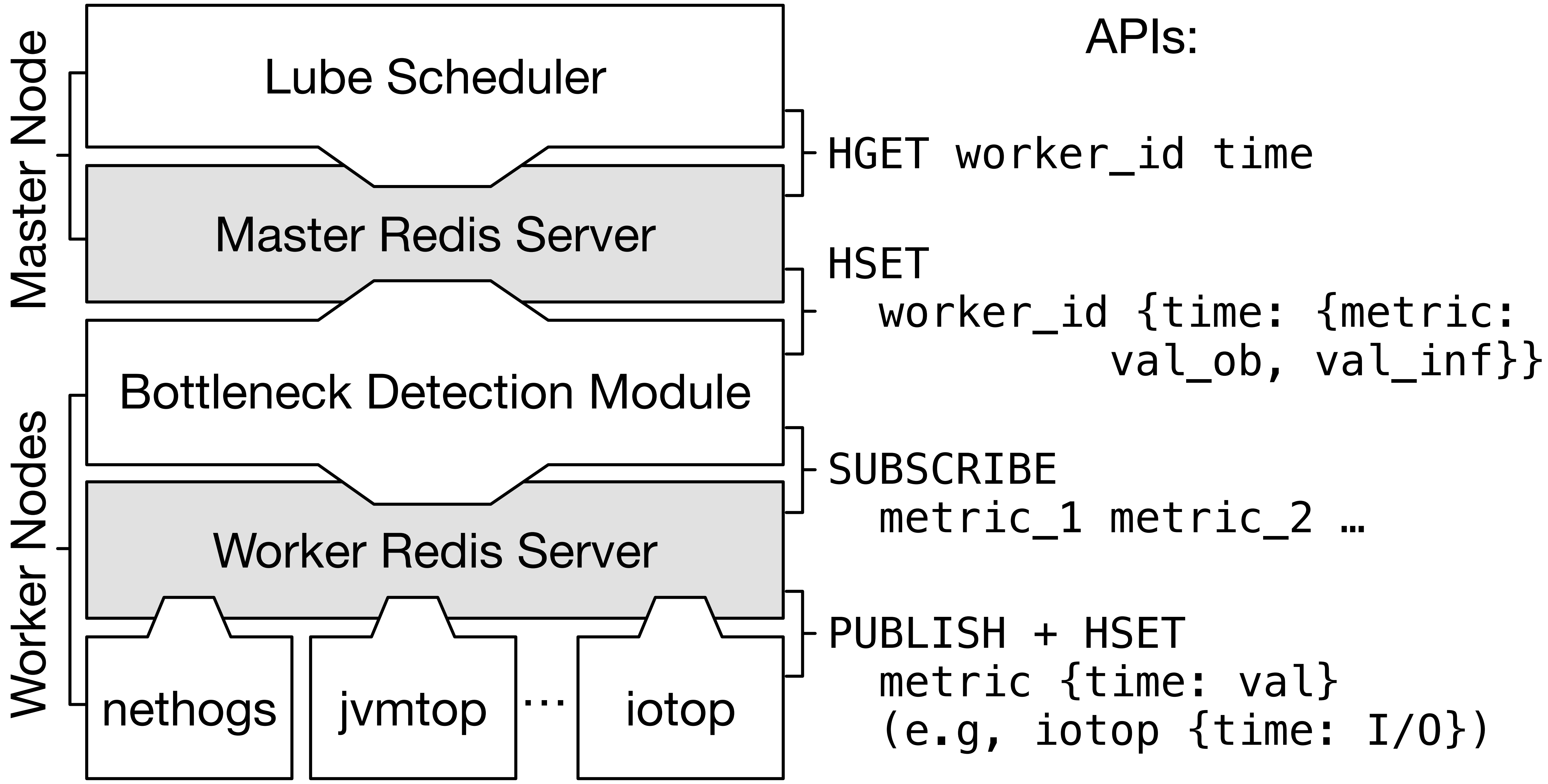
Implementation & Deployment

Implementation

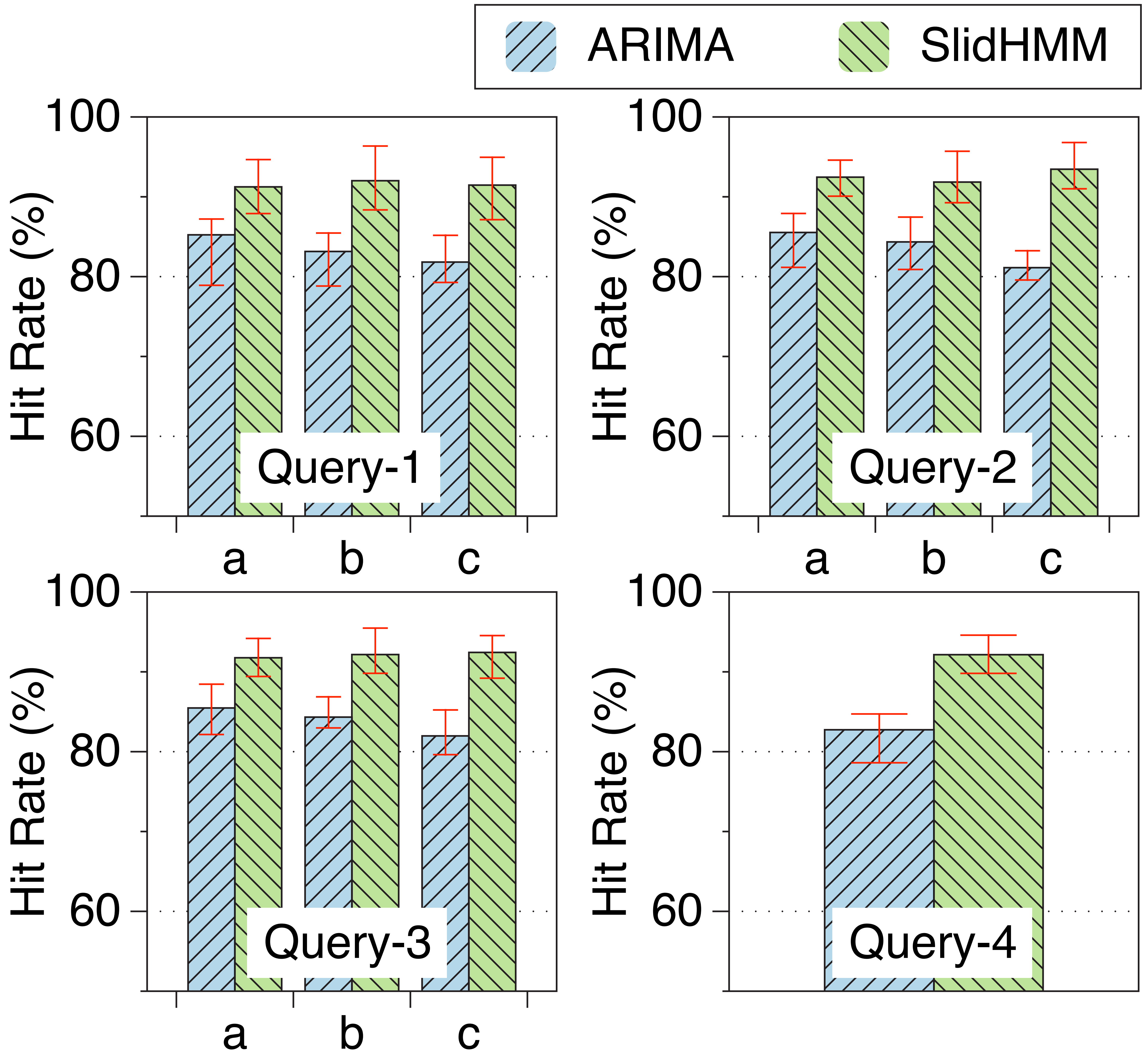
- Spark-1.6.1 (scheduler)
- redis database (cache)
- Python *scikit-learn,* Keras (ML)

Deployment

- 37 EC2 m4.2xlarge instances
- 9 regions
- Berkeley Big Data Benchmark
- An 1.1 TB dataset



Evaluation — Accuracy

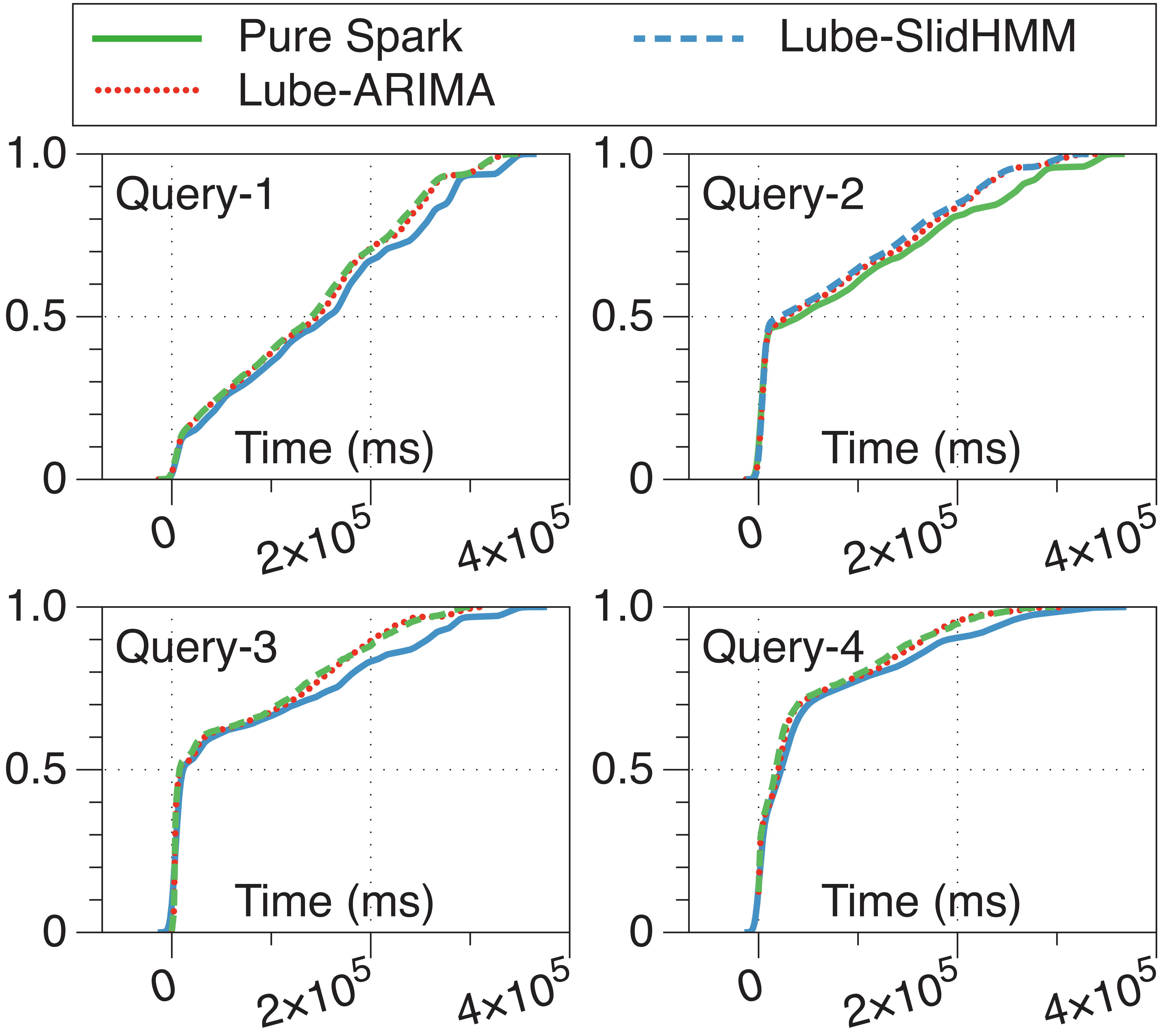


Calculation

$$hitrate = \frac{\#((\text{time}, \text{detection}) \cap (\text{time}, \text{observation}))}{\#(\text{time}, \text{detection})}$$

ARIMA ignores nonlinear patterns

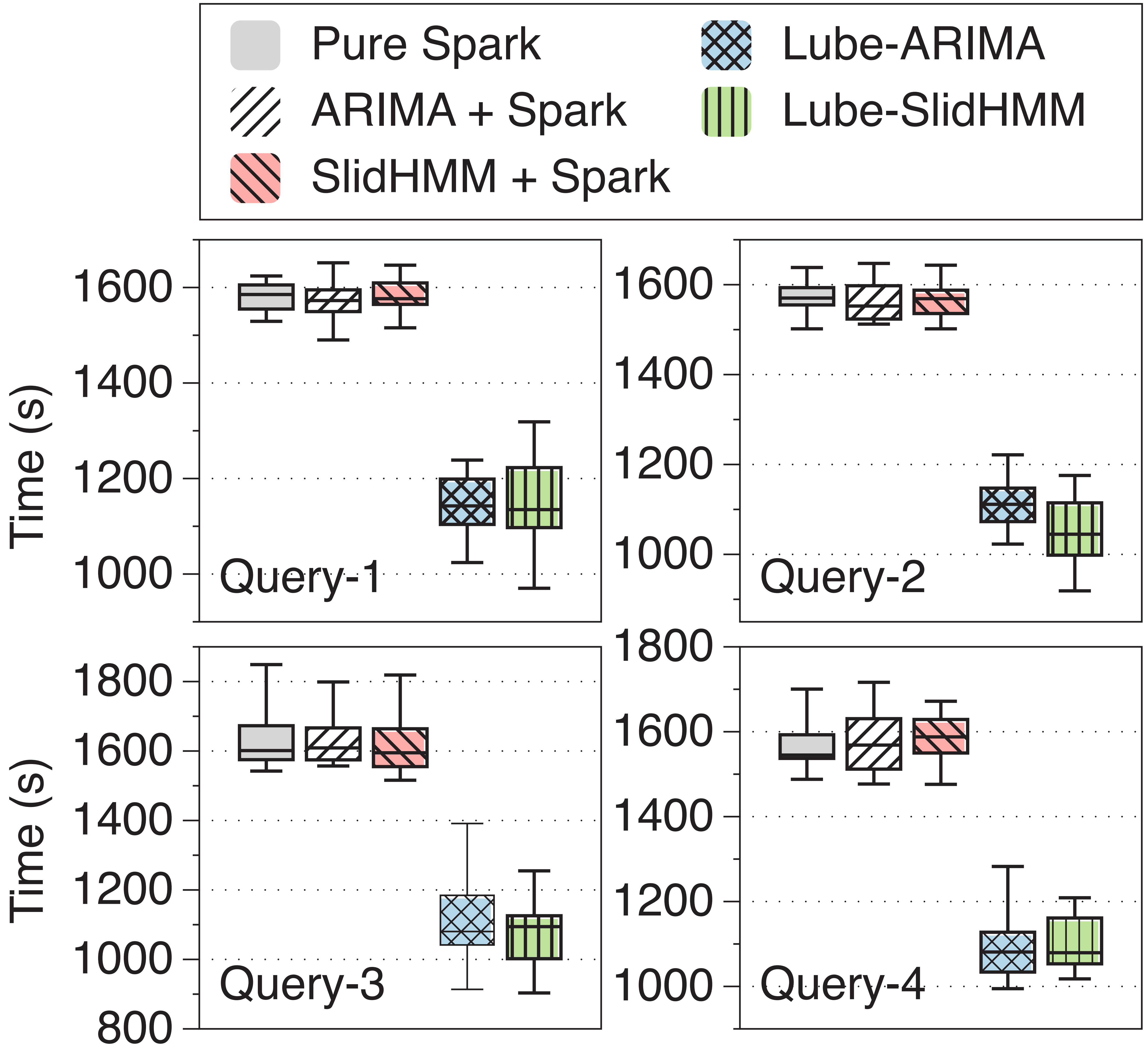
Evaluation — Completion Times



Task completion times

	Average	75th
Lube-ARIMA	12.454s	22.075s
Lube-SlidHMM	14.783s	27.469s

Evaluation — Completion Times



Query completion times

- Lube-ARIMA
- Lube-SlidHMM
- Reduce median query response time by up to 33%

Control Groups for overhead

- ARIMA + Spark
- SlidHMM + Spark
- **Negligible overhead**

Conclusion

- Runtime performance bottleneck detection
  - ARIMA, HMM
- A simple greedy bottleneck-aware task scheduler
  - Jointly consider data-locality and bottlenecks
- **Lube**, a closed-loop framework mitigating bottlenecks at runtime.

The End | Thank You

Discussion

Bottleneck detection models

- More performance metrics could be explored
- More efficient models for time series prediction, *e.g.*, Reinforcement Learning, LSTM

Bottleneck-aware scheduling

- Fine-grained scheduling with specific resource awareness

WAN conditions

- We measure pair-wise WAN bandwidths by a *cron* job running *iperf* locally
- Try to exploit support from SDN interfaces