Learning Scheduling Algorithms for Data Processing Clusters

Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan,
Zili Meng, Mohammad Alizadeh

Motivation

Scheduling is a fundamental task in computer systems

- Cluster management (e.g., Kubernetes, Mesos, Borg)
- Data analytics frameworks (e.g., Spark, Hadoop)
- Machine learning (e.g., Tensorflow)

Efficient scheduler matters for large datacenters

- Small improvement can save millions of dollars at scale

Designing Optimal Schedulers is Intractable

Must consider many factors for optimal performance:

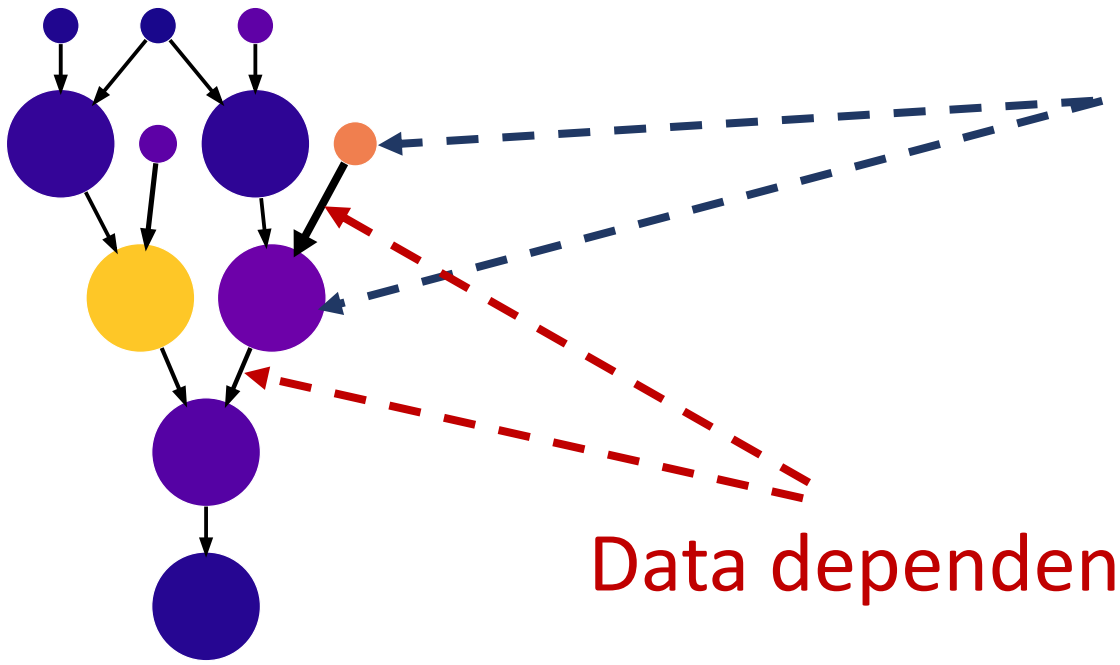
• Job dependency structure	*Graphene [OSDI '16], Carbyne [OSDI '16]*
• Modeling complexity	*Tetris [SIGCOMM '14], Jockey [EuroSys '12]*
• Placement constraints	*TetriSched [EuroSys '16], device placement [NIPS '17]*
• Data locality	*Delayed Scheduling [EuroSys '10]*
• ……	……

No "one-size-fits-all" solution:
Best algorithm depends on specific workload and system

Can machine learning help tame the complexity of efficient schedulers for data processing jobs?

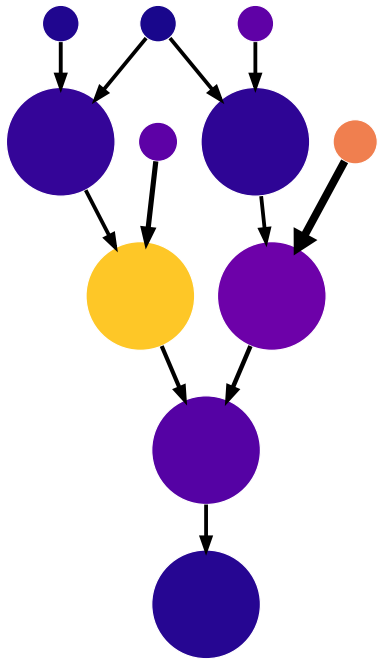
Decima: A Learned Cluster Scheduler

- Learns workload-specific scheduling algorithms for jobs with dependencies (represented as DAGs)

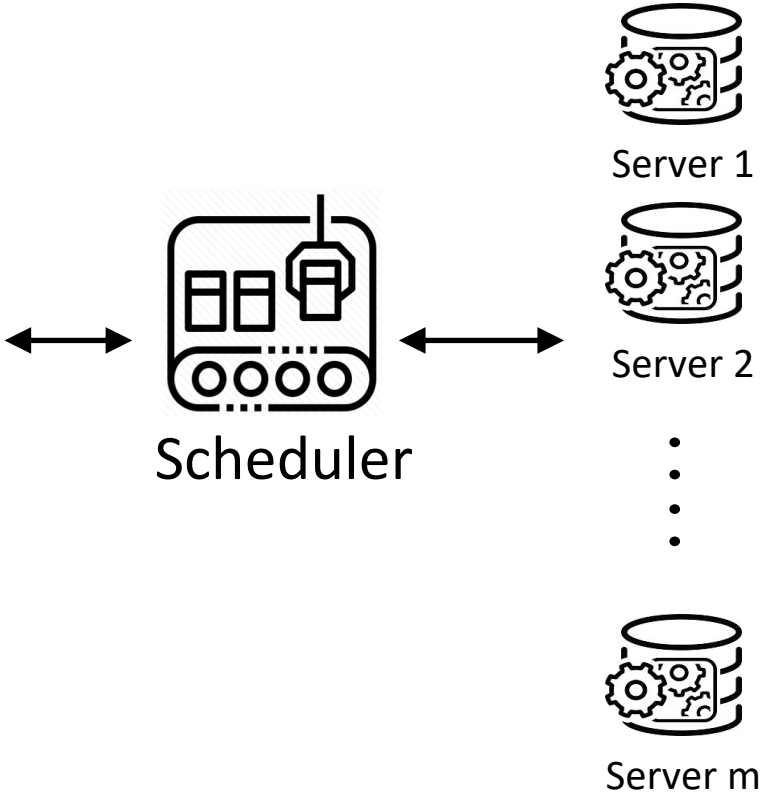


Decima: A Learned Cluster Scheduler

- Learns workload-specific scheduling algorithms for jobs with dependencies (represented as DAGs)



Job 1

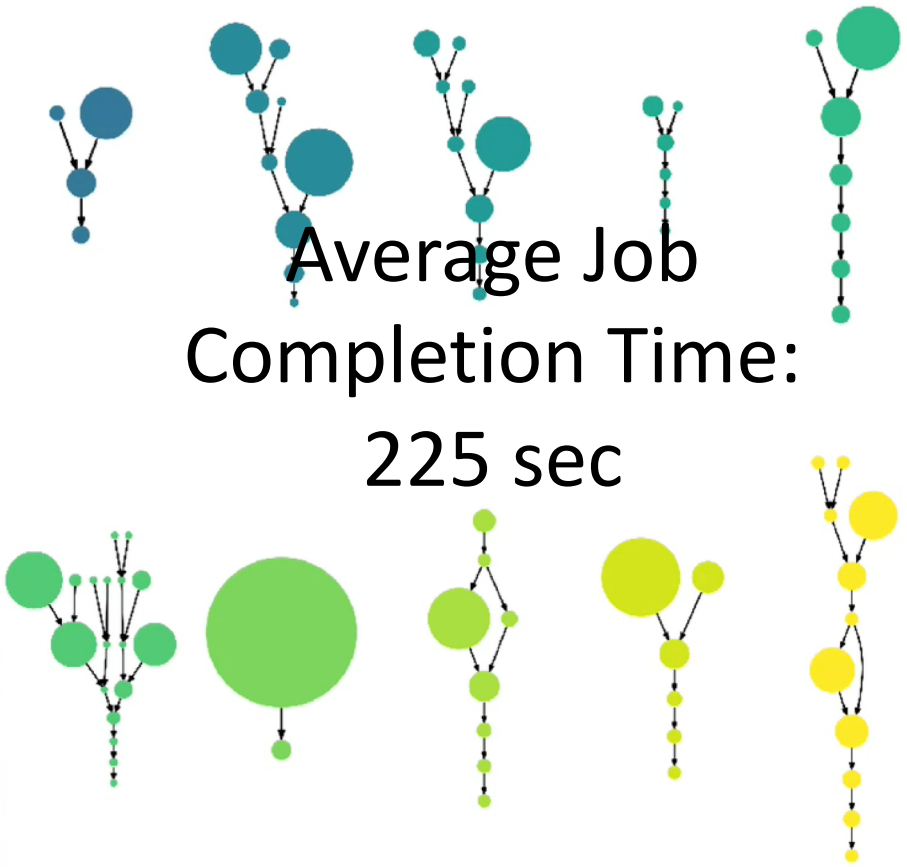


Design overview

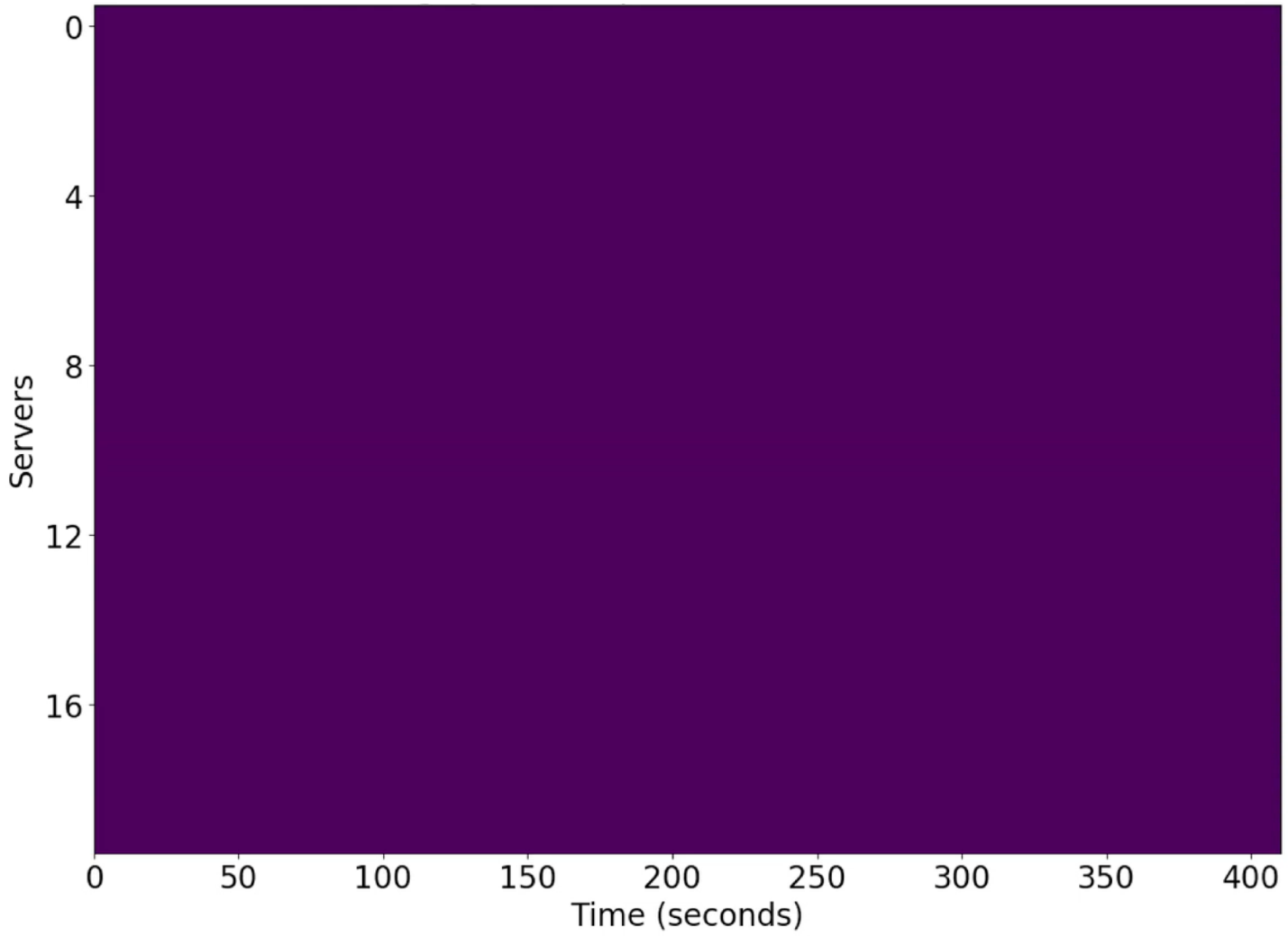
Scheduling policy: FIFO

Demo

Number of servers working on this job

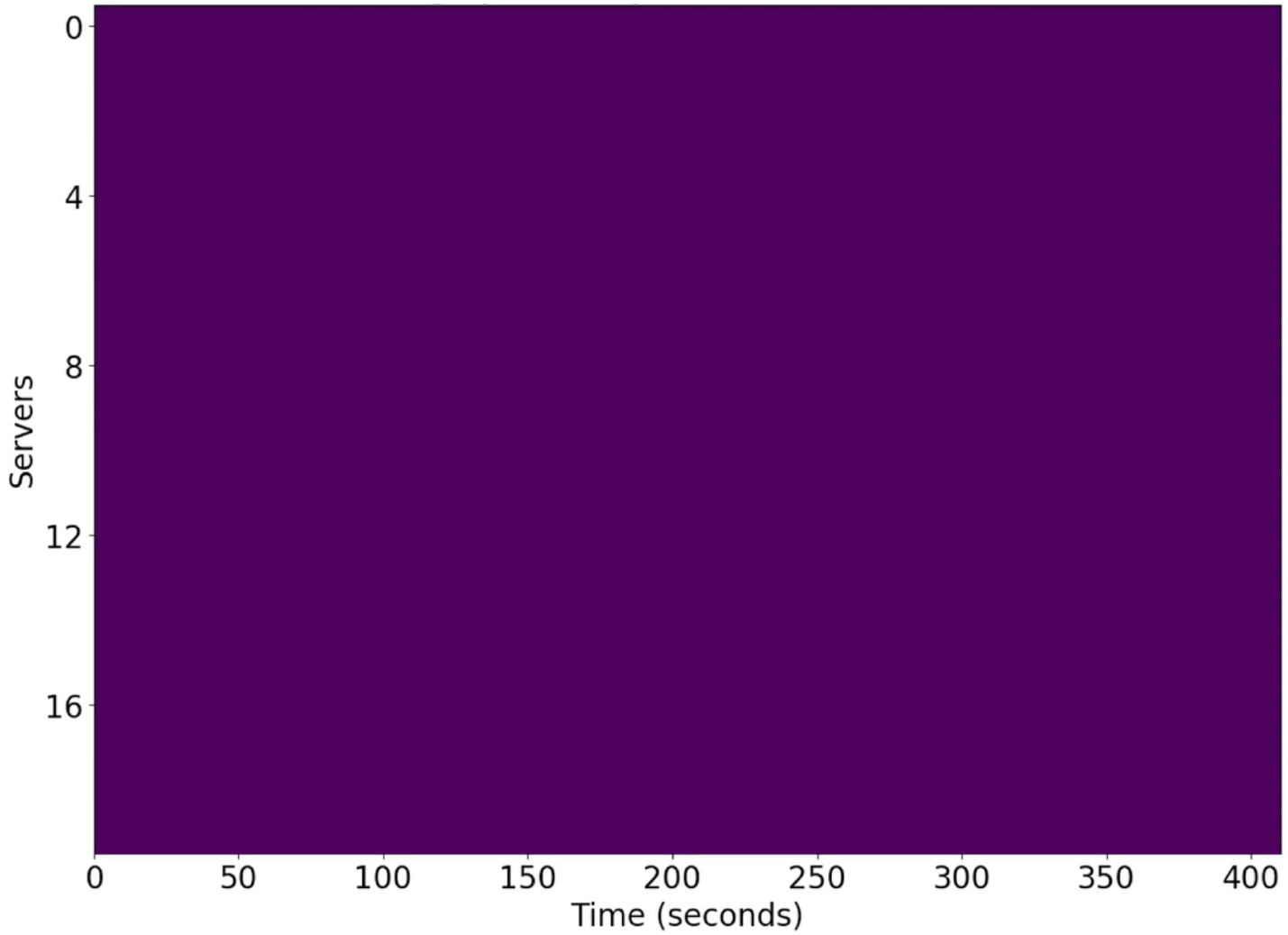


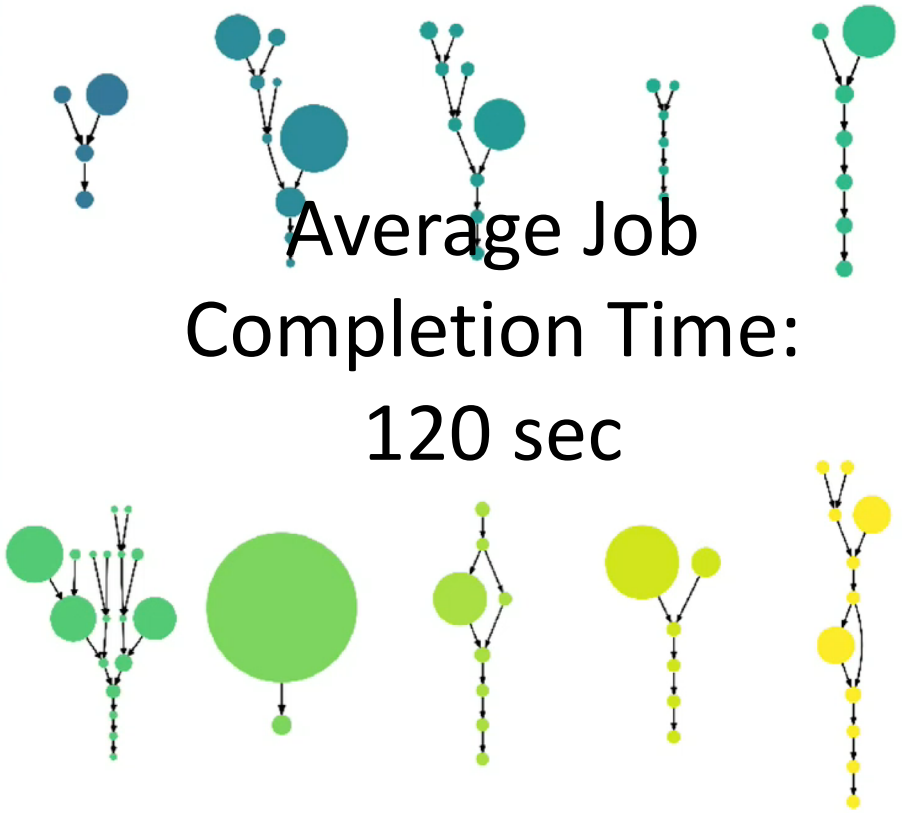
Scheduling policy: Shortest-Job-First



Average Job Completion Time: 135 sec

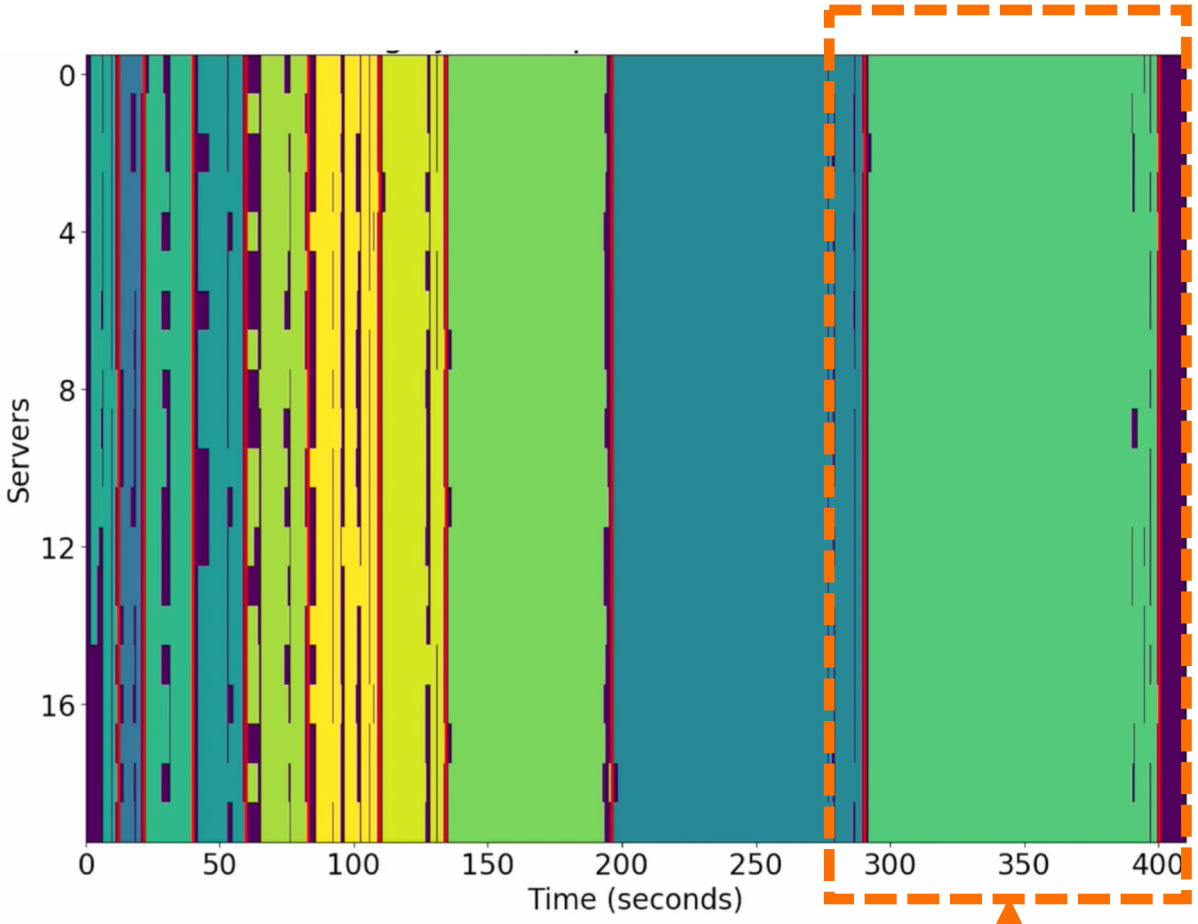
Scheduling policy: Fair



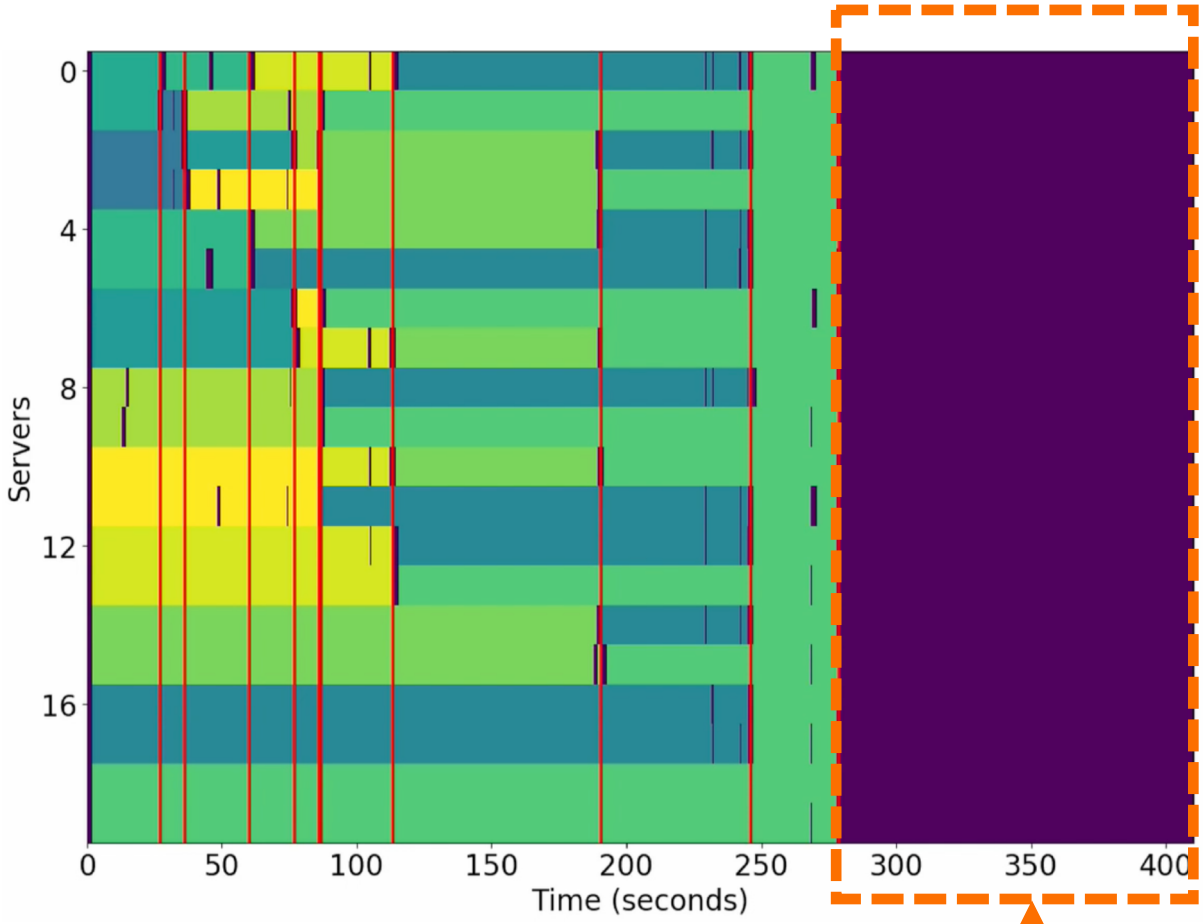


Average Job Completion Time: 120 sec

Shortest-Job-First



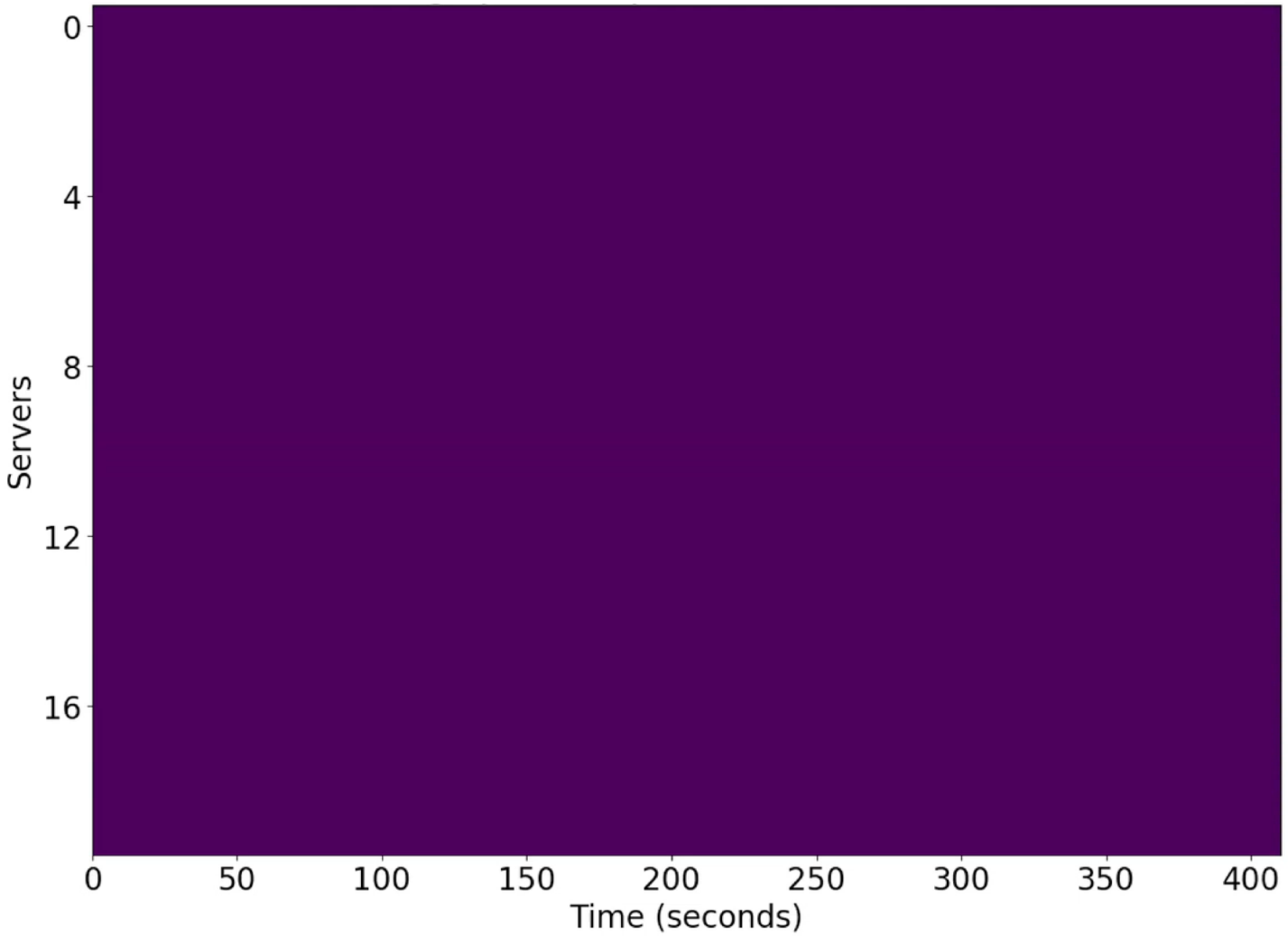
Fair

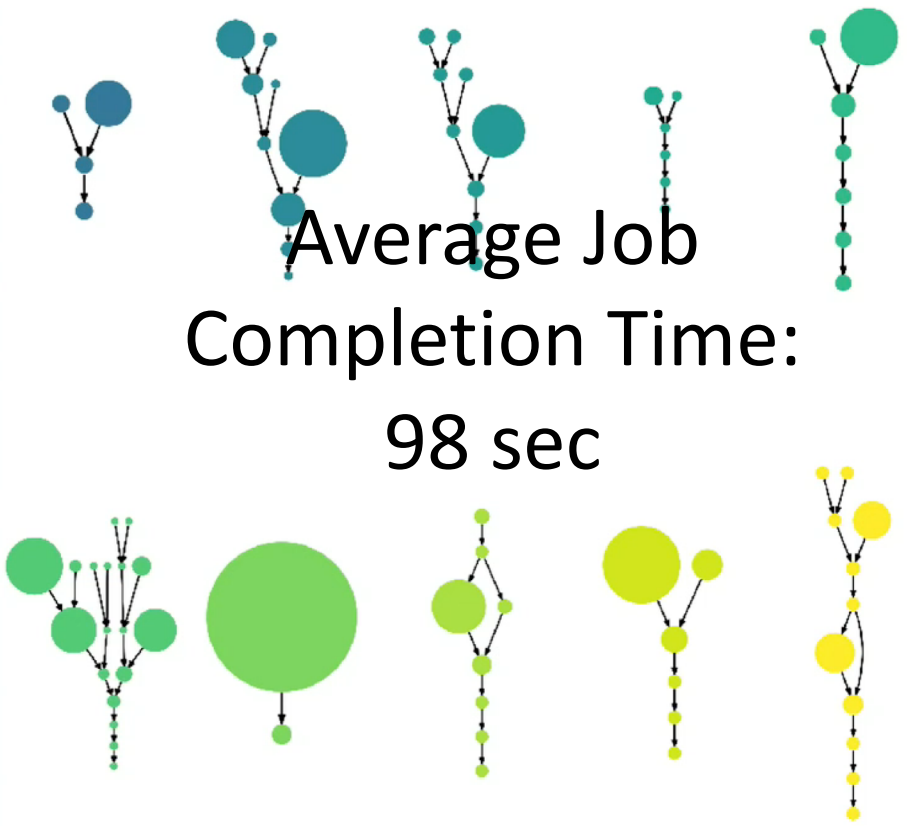


Average Job Completion Time:
135 sec

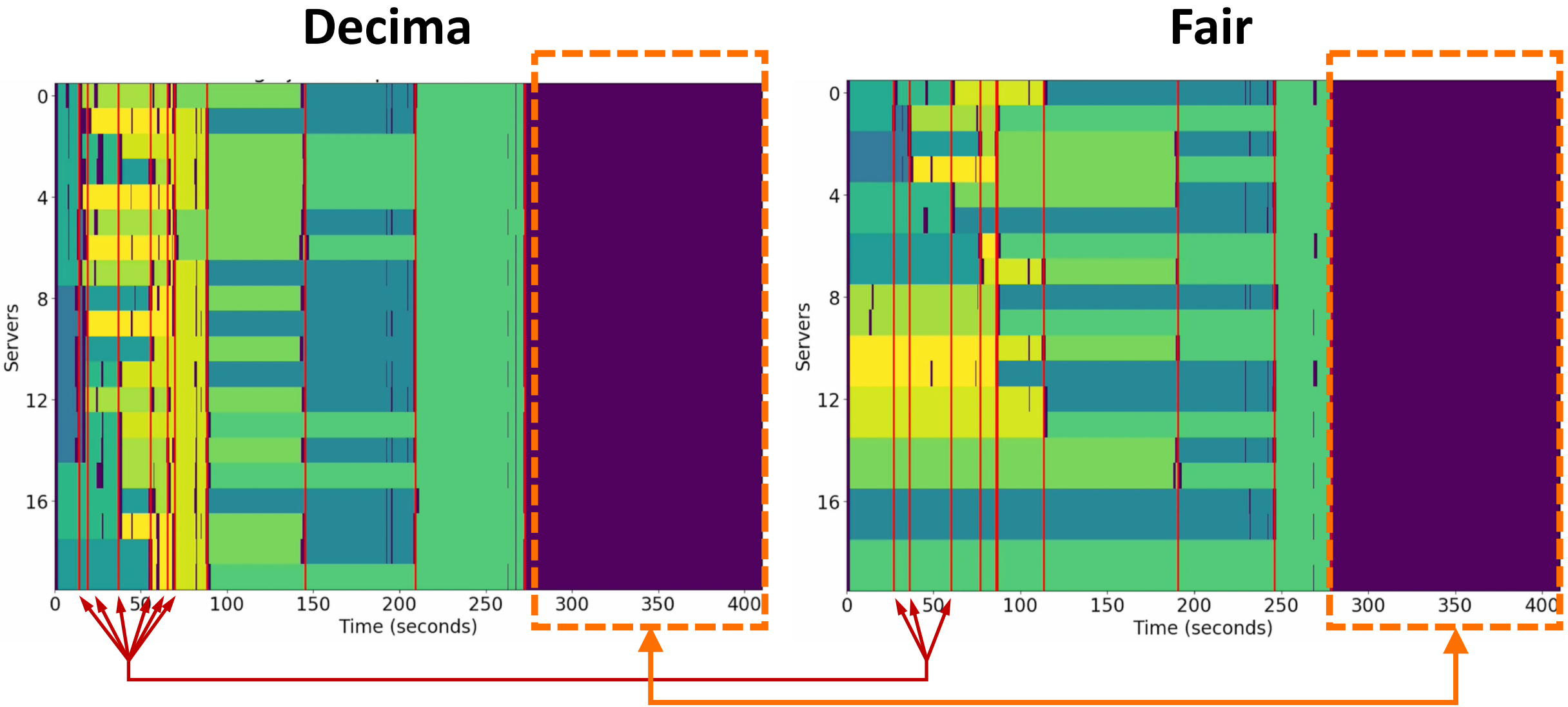
Average Job Completion Time:
120 sec

Scheduling policy: Decima





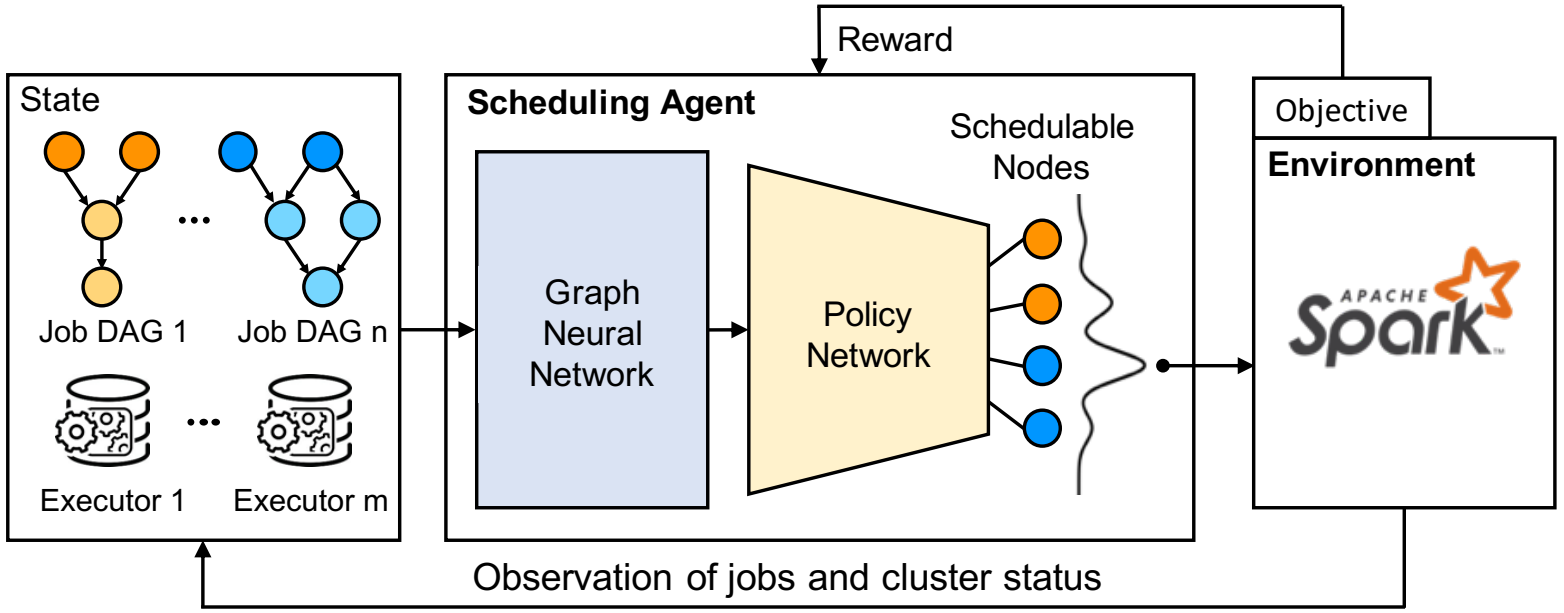
Average Job Completion Time: 98 sec



Average Job Completion Time:
98 sec

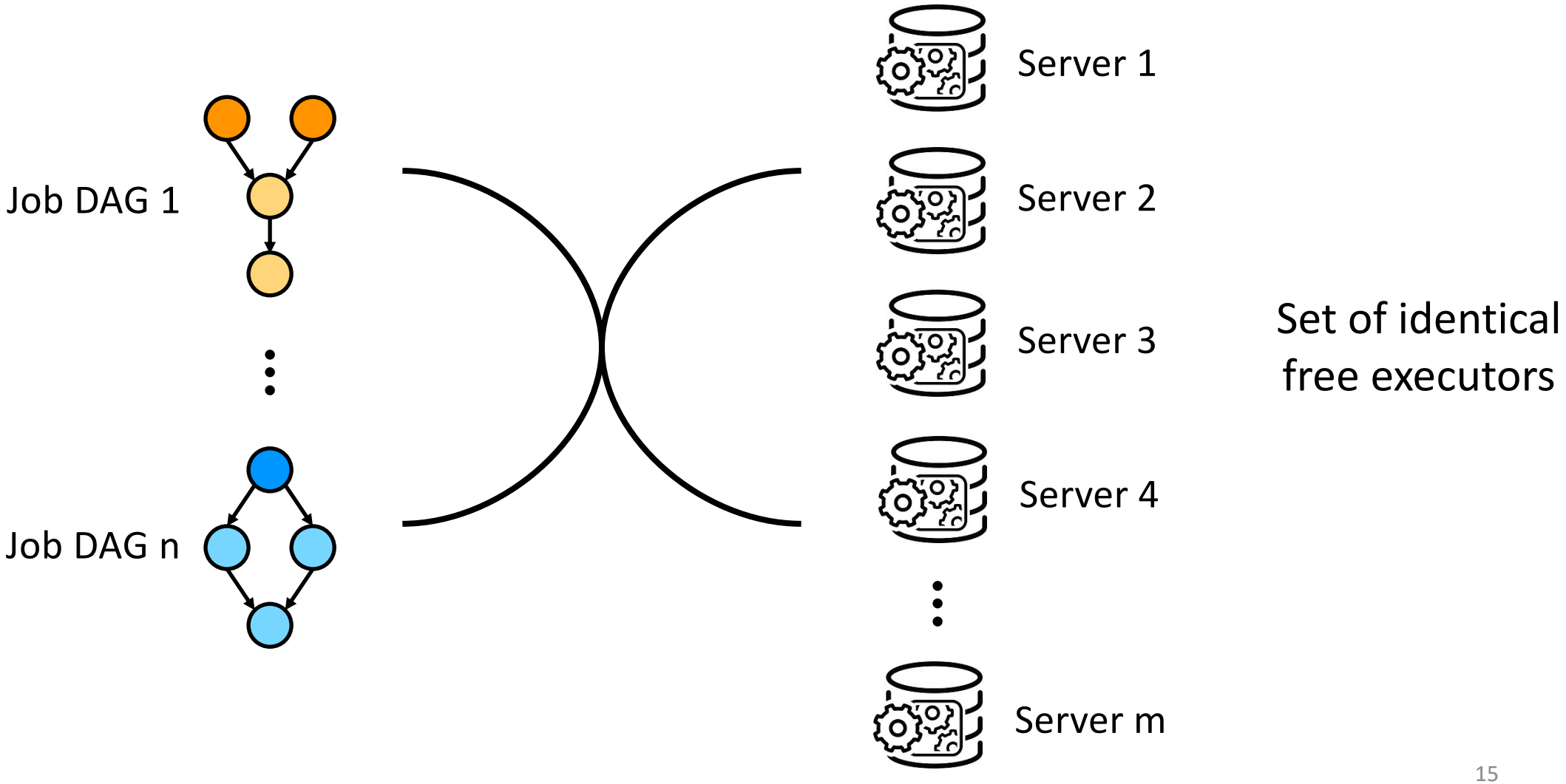
Average Job Completion Time:
120 sec

Contributions

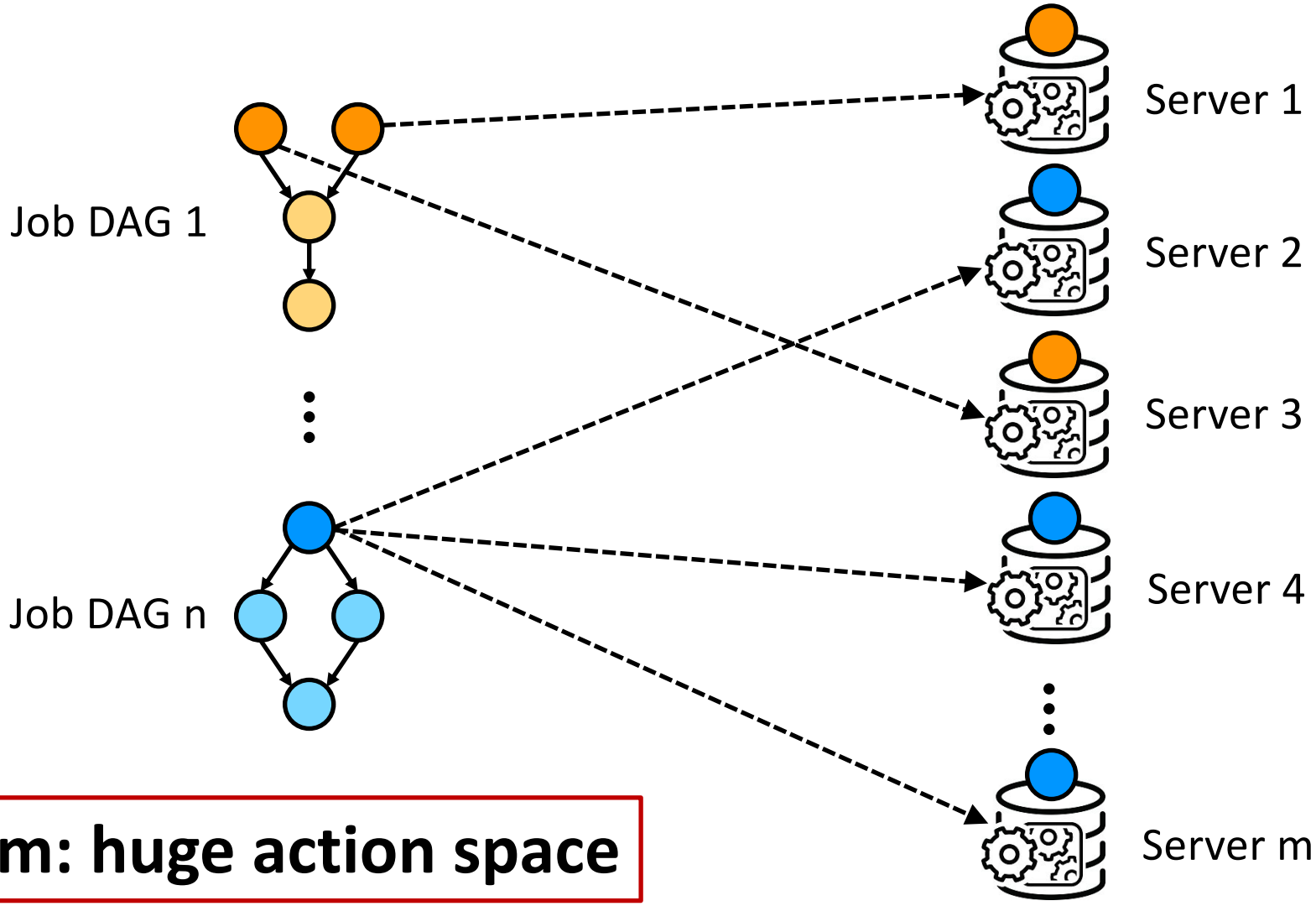


1. First RL-based scheduler for complex data processing jobs
2. Scalable graph neural network to express scheduling policies
3. New learning methods that enables training with online job arrivals

Encode scheduling decisions as actions

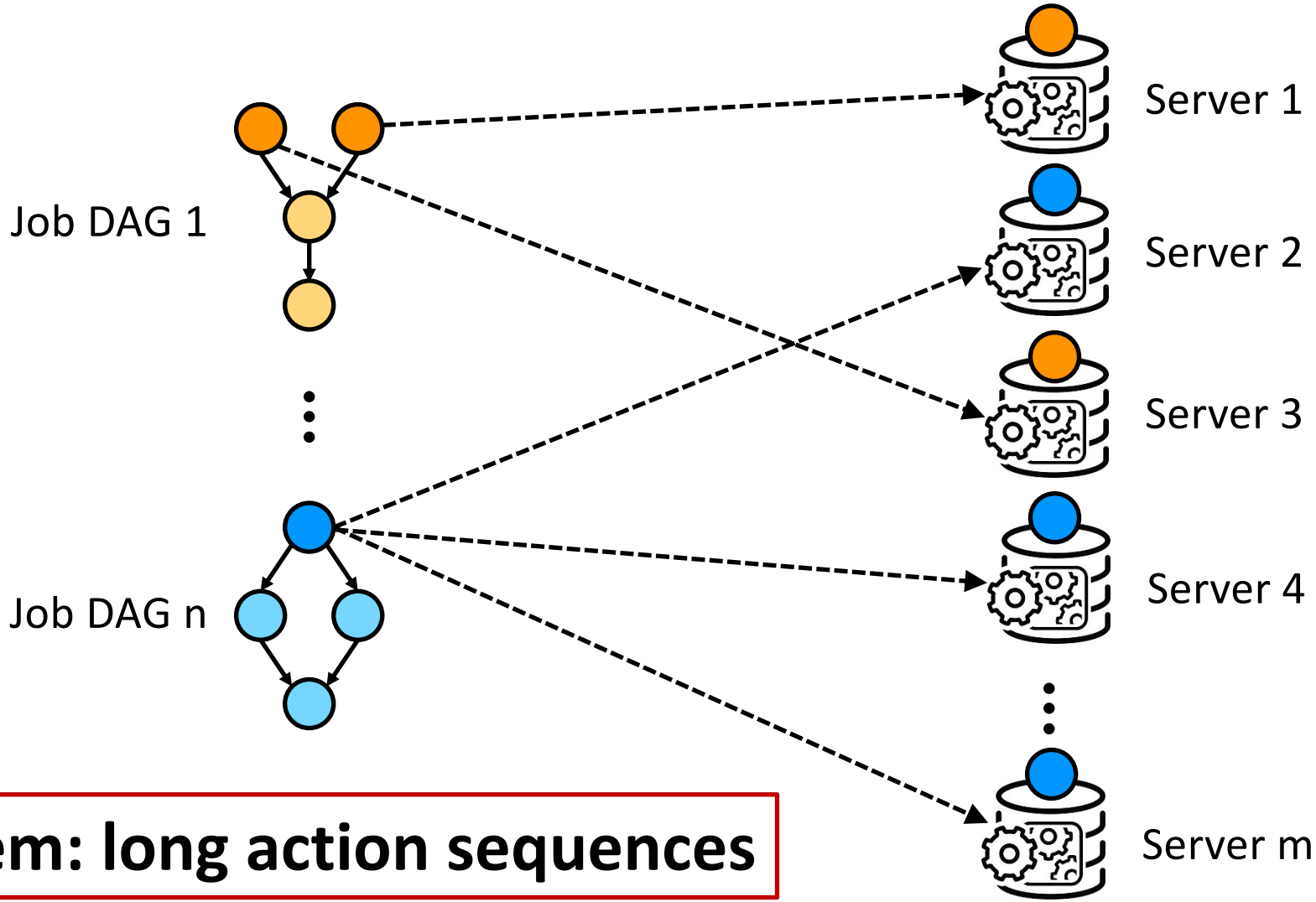


Option 1: Assign all Executors in 1 Action



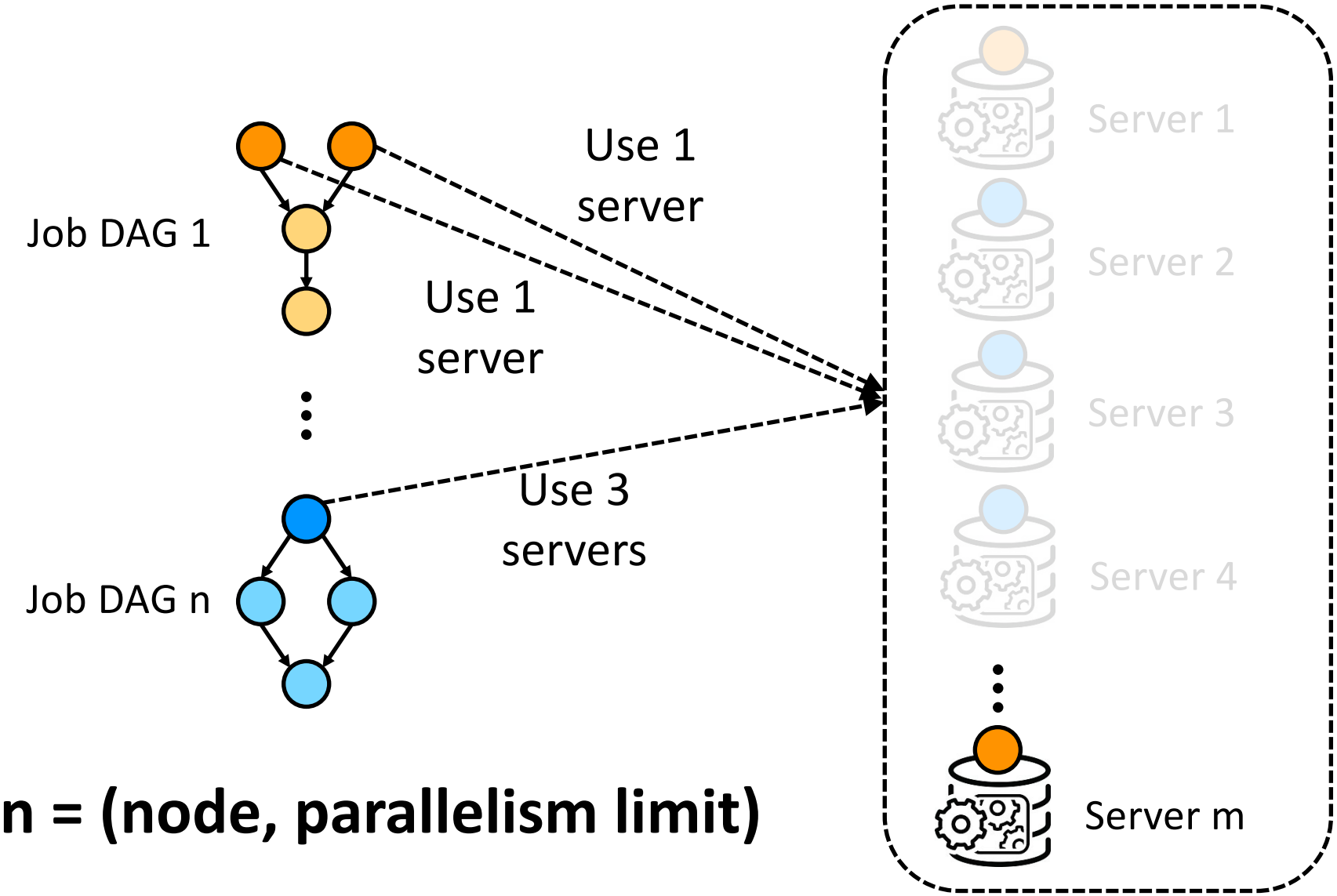
Problem: huge action space

Option 2: Assign One Executor Per Action



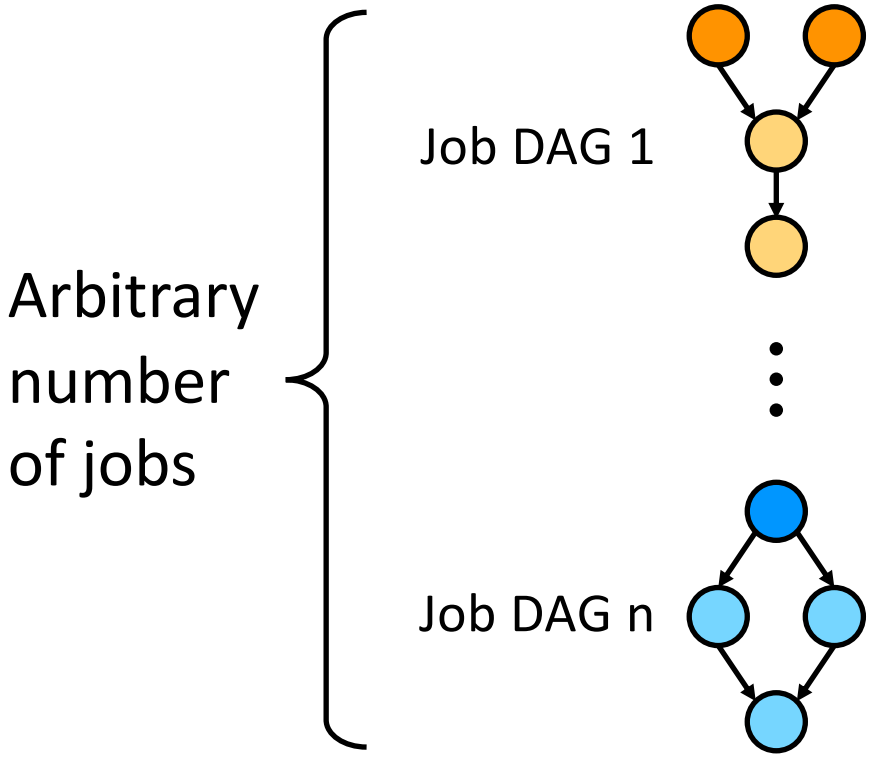
Problem: long action sequences

Decima: Assign Groups of Executors per Action



Action = (node, parallelism limit)

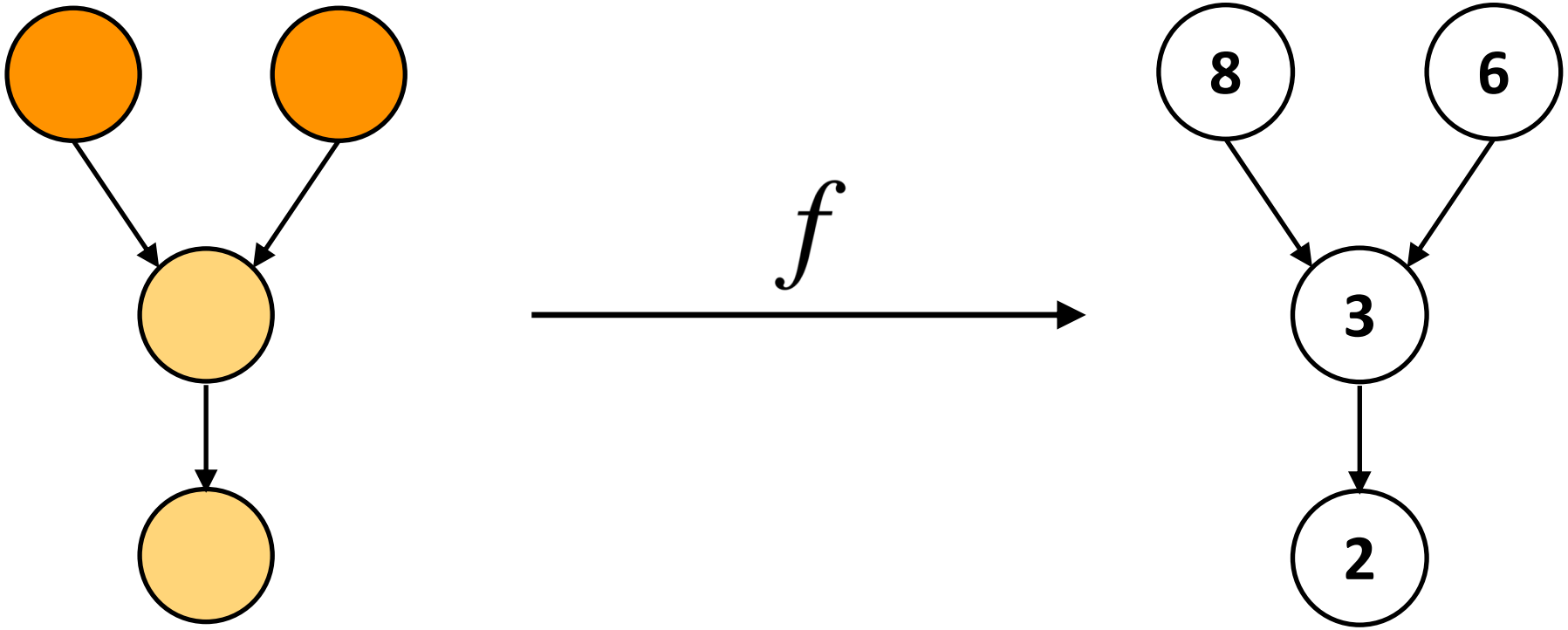
Process Job Information



Node features:

- # of tasks
- avg. task duration
- # of servers currently assigned to the node
- are free servers local to this job?

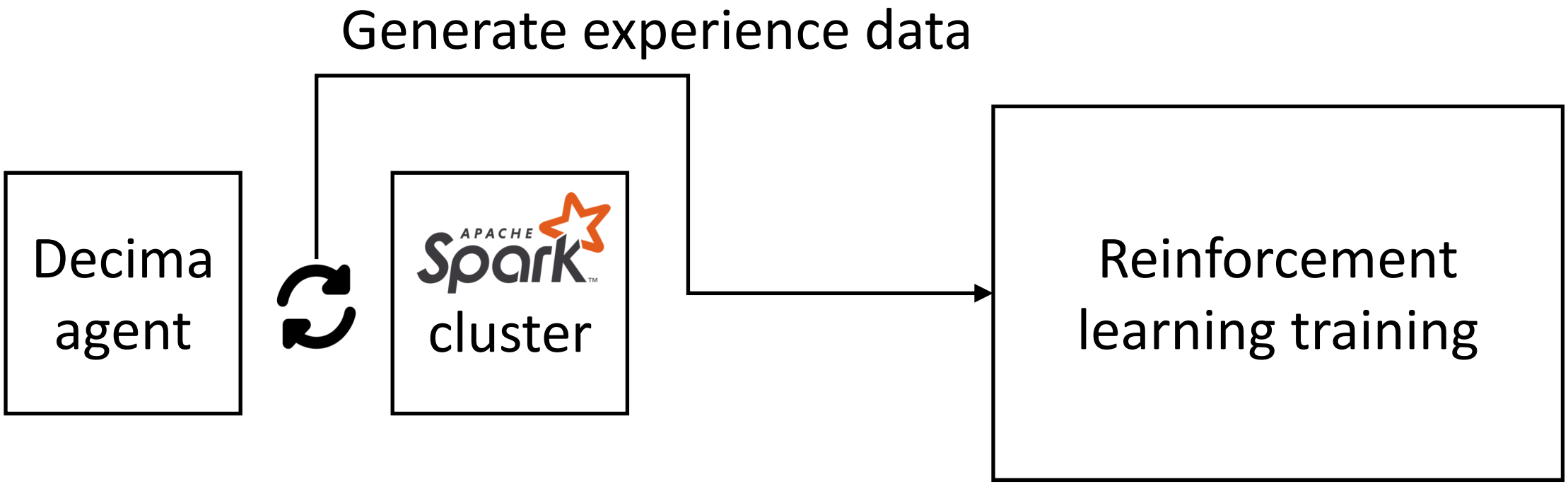
Graph Neural Network



Job DAG

Score on each node

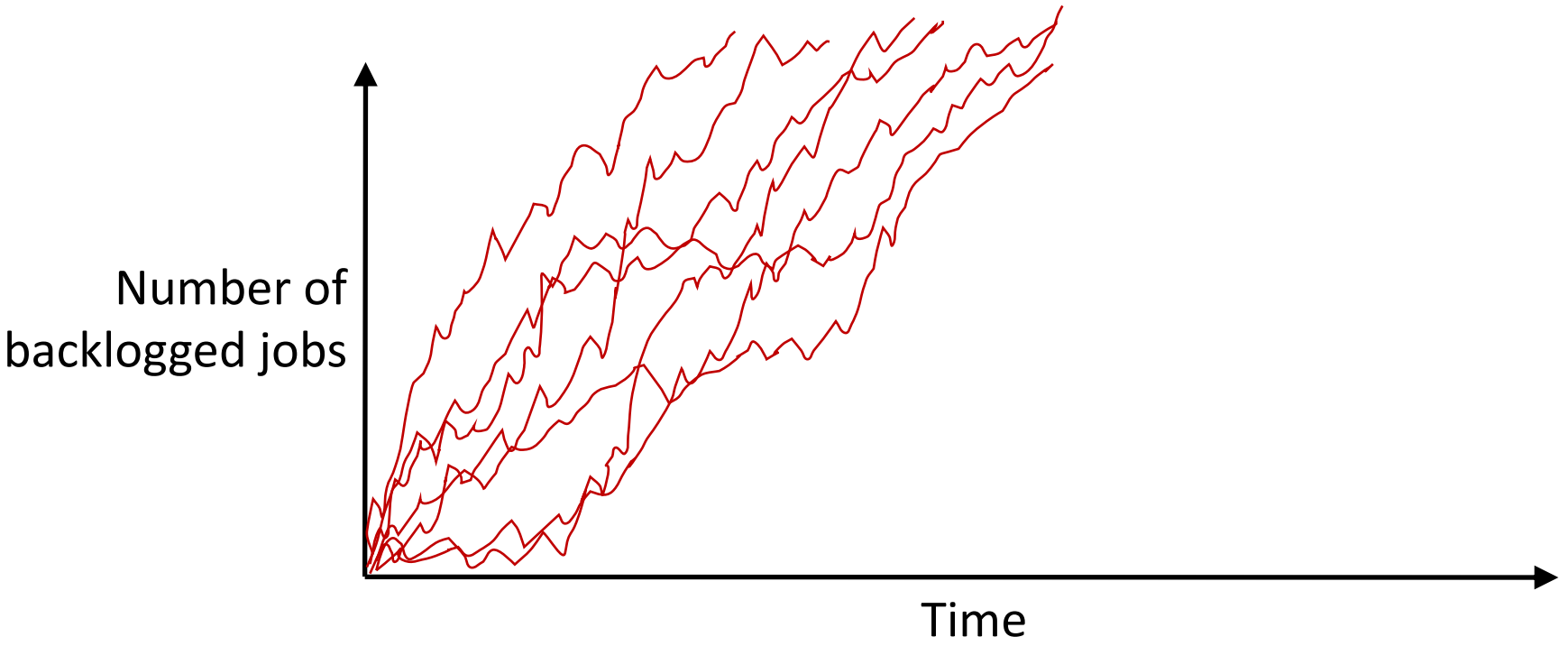
Training


Generate experience data
Decima agent
APACHE Spark
cluster
Reinforcement learning training

Handle Online Job Arrival

The RL agent has to experience **continuous job arrival** during **training**.

→ inefficient if simply feeding long sequences of jobs

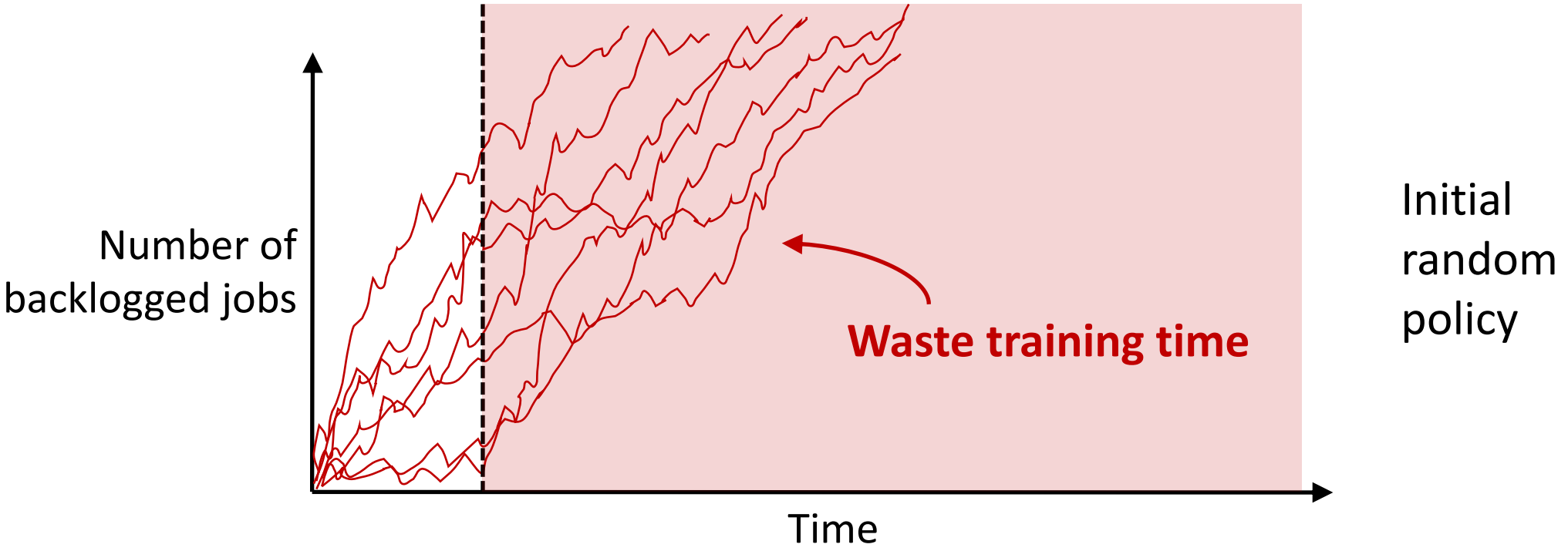


Initial random policy

Handle Online Job Arrival

The RL agent has to experience **continuous job arrival** during **training**.

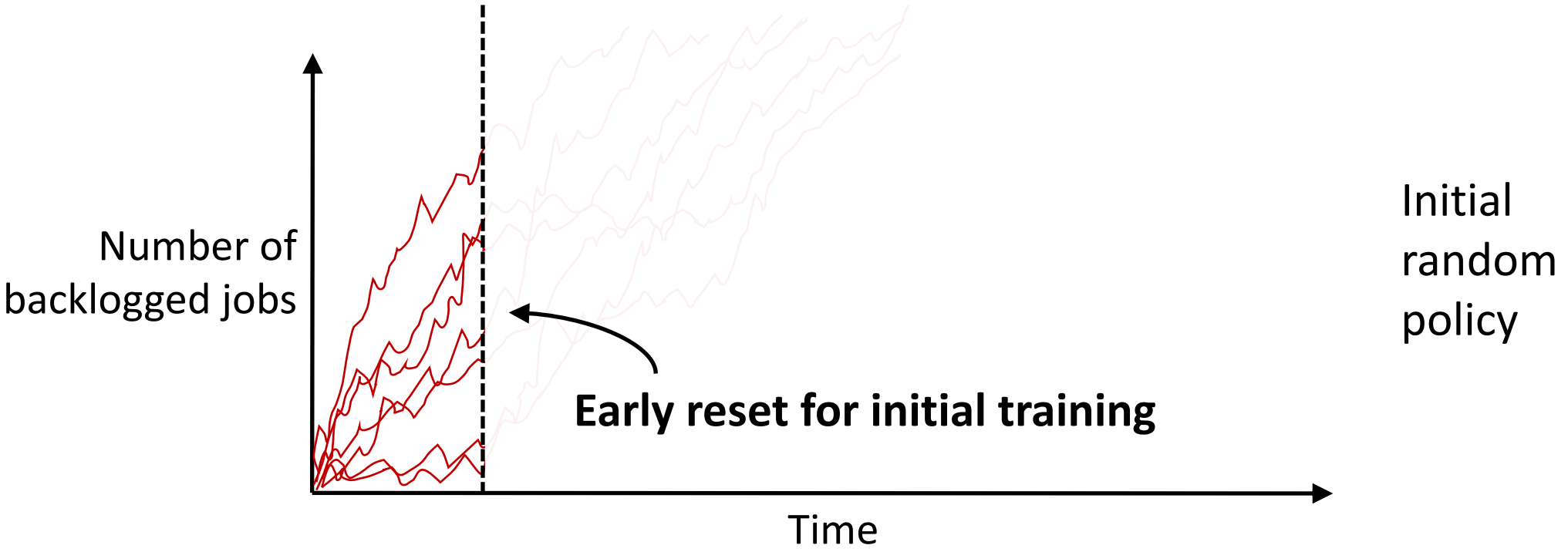
→ inefficient if simply feeding long sequences of jobs



Handle Online Job Arrival

The RL agent has to experience **continuous job arrival** during **training**.

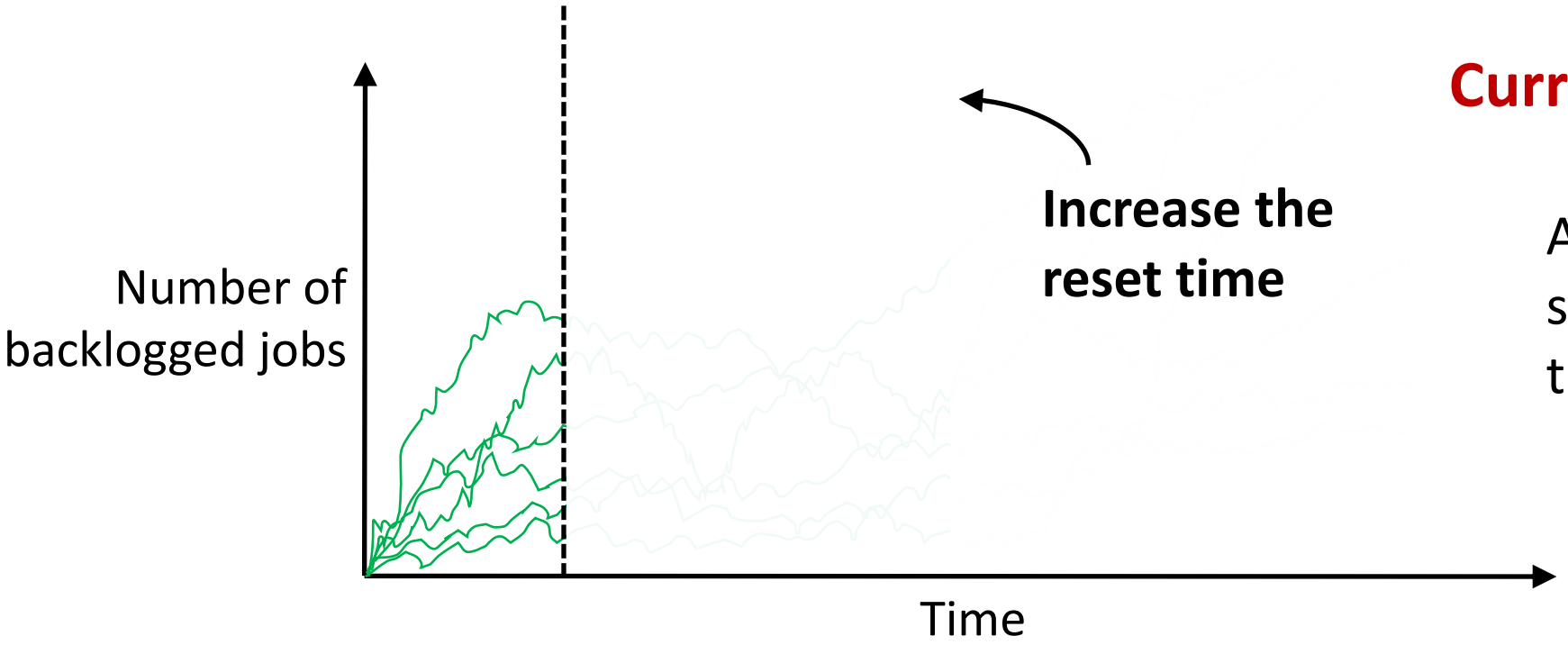
→ inefficient if simply feeding long sequences of jobs



Handle Online Job Arrival

The RL agent has to experience **continuous job arrival** during **training**.

→ inefficient if simply feeding long sequences of jobs



Curriculum learning

As training proceeds, stronger policy keeps the queue stable

Variance from Job Sequences

RL agent needs to be **robust** to the **variation** in job arrival patterns.

→ huge variance can throw off the training process

Variance from Job Sequences

Must consider the entire job sequence to score actions

Score for action a_t = (*return after* a_t) − (*average return*)

$$= \sum_{t'=t}^{T} r_{t'} - b(s_t)$$

Input-Dependent Baseline

Score for action a_t = $\sum_{t'=t}^{T} r_{t'} - b(s_t, z_t, z_{t+1}, \dots)$

Average return for trajectories from state s_t
with job sequence z_t, z_{t+1}, …

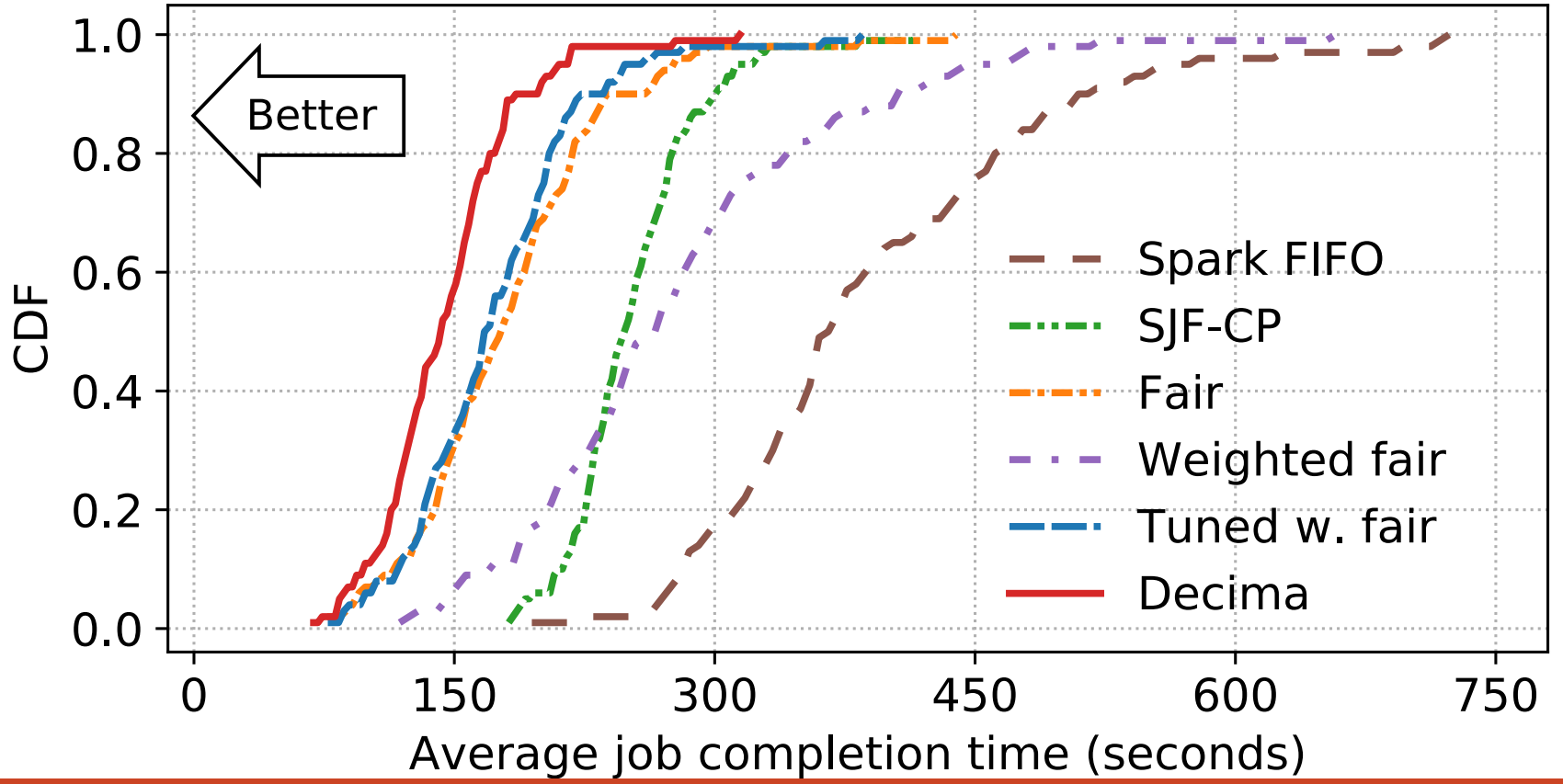
Broadly applicable to other systems with external input process:

Adaptive video streaming, load balancing, caching, robotics with disturbance…

- **Variance reduction for reinforcement learning in input-driven environments**. Hongzi Mao, Shaileshh Bojja Venkatakrishnan, Malte Schwarzkopf, Mohammad Alizadeh. *International Conference on Learning Representations (ICLR)*, 2019.

Decima vs. Baselines: Batched Arrivals

- 20 TPC-H queries sampled at random; input sizes: 2, 5, 10, 20, 50, 100 GB
- Decima trained on simulator; tested on real Spark cluster

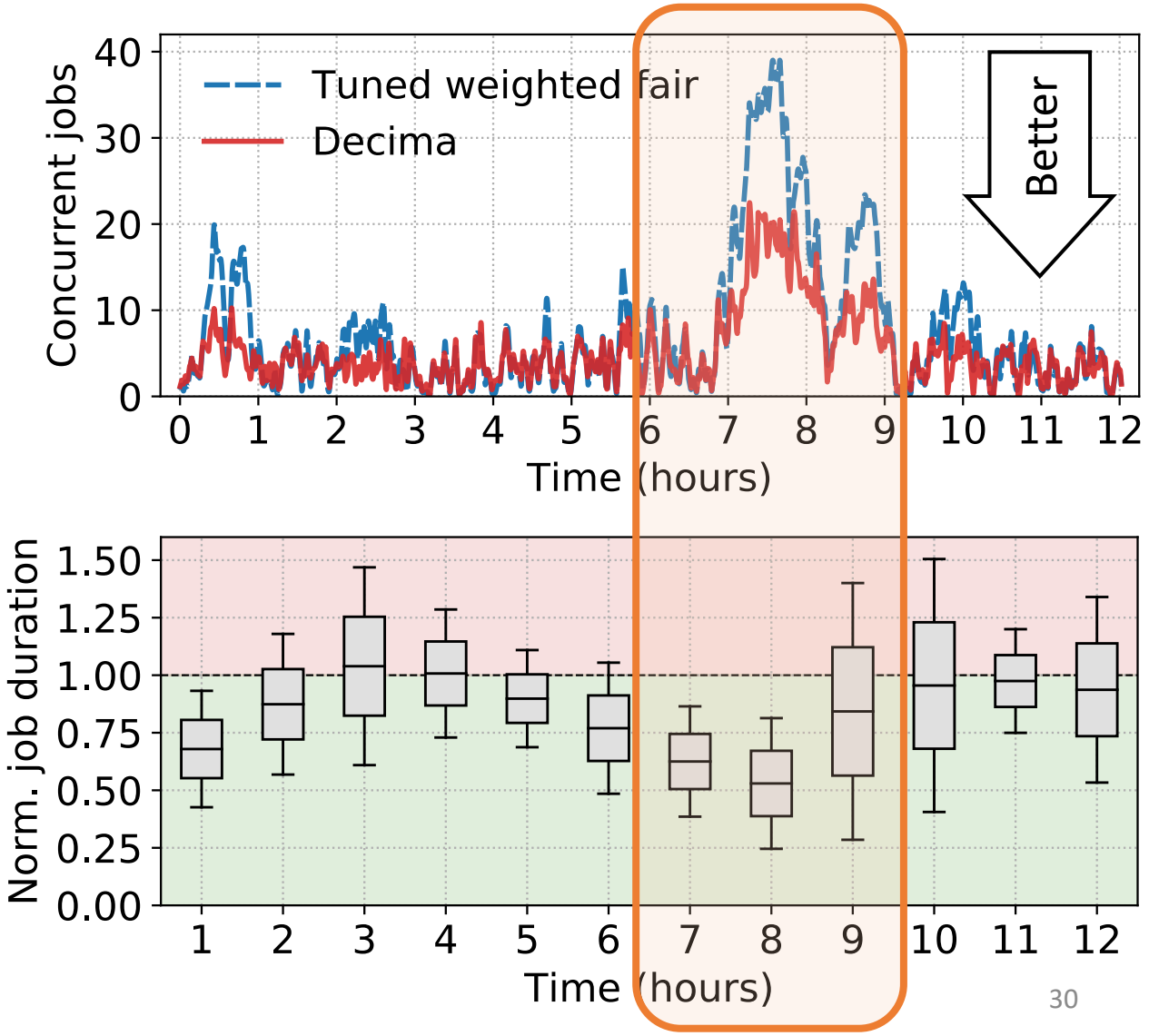


Decima improves average job completion time by 21%-3.1x over baseline schemes

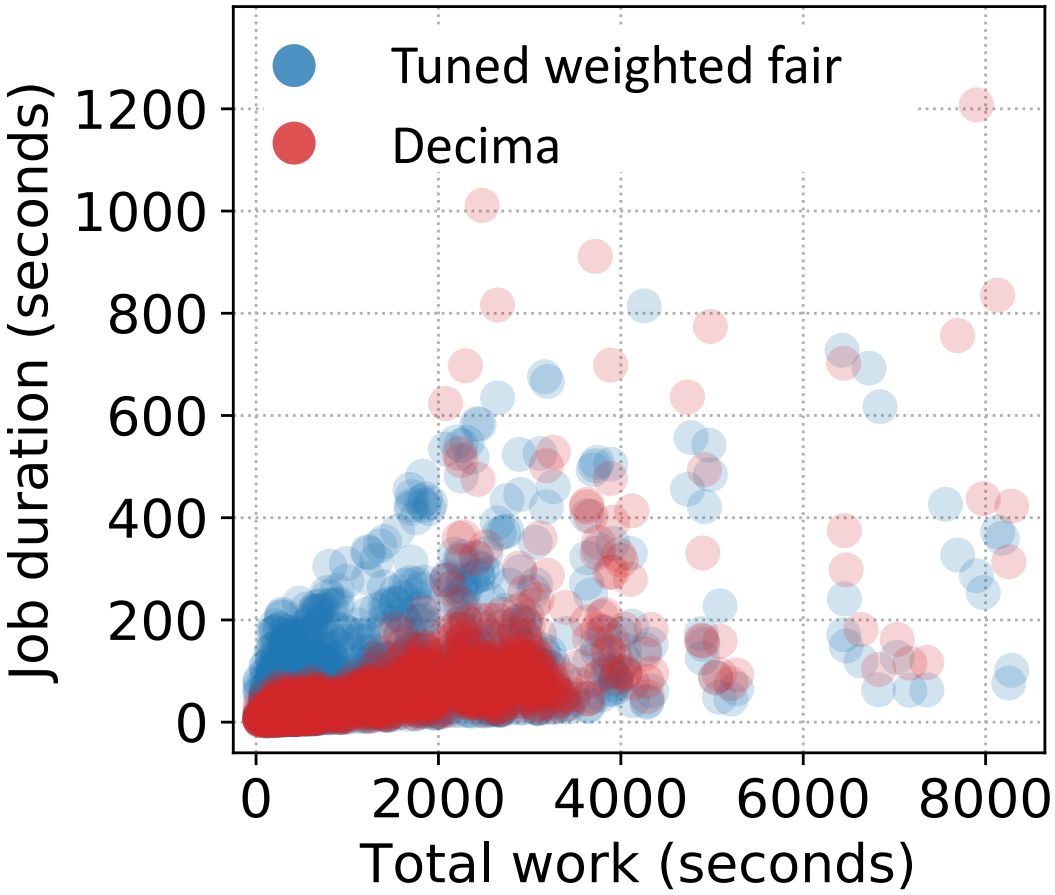
Decima with Continuous Job Arrivals

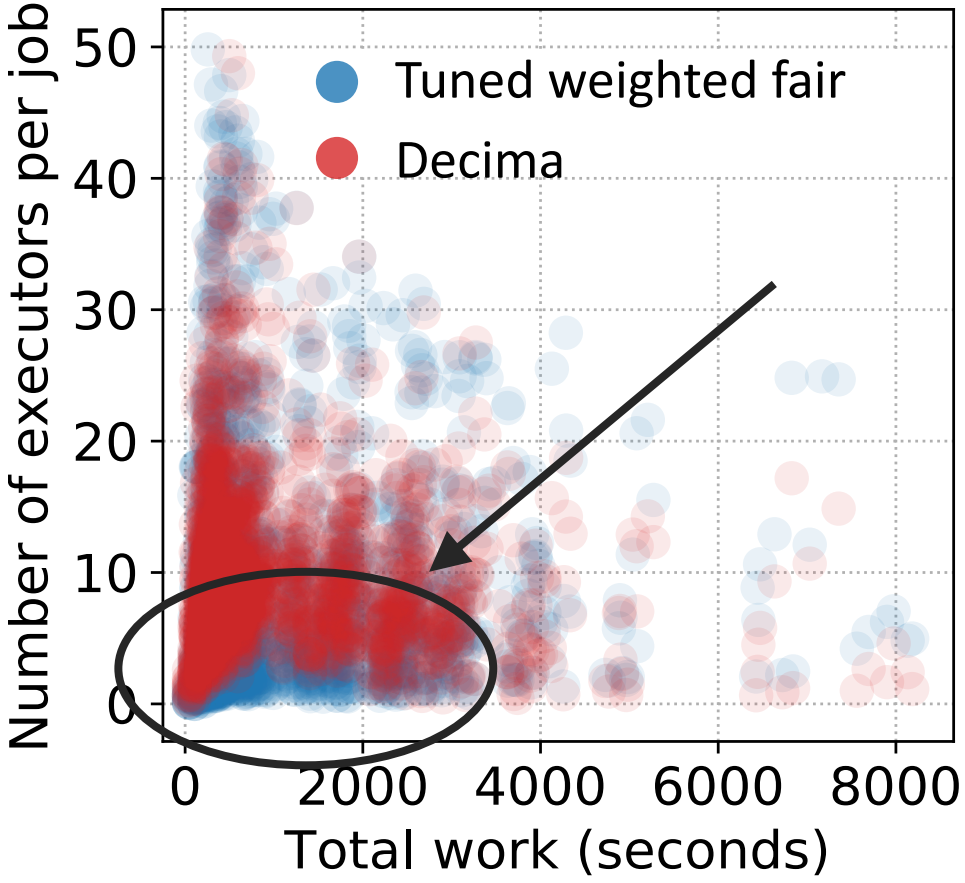
1000 jobs arrives as a Poisson process with avg. inter-arrival time = 25 sec

Decima achieves 28% lower average JCT than best heuristic, and 2X better JCT in overload



Understanding Decima





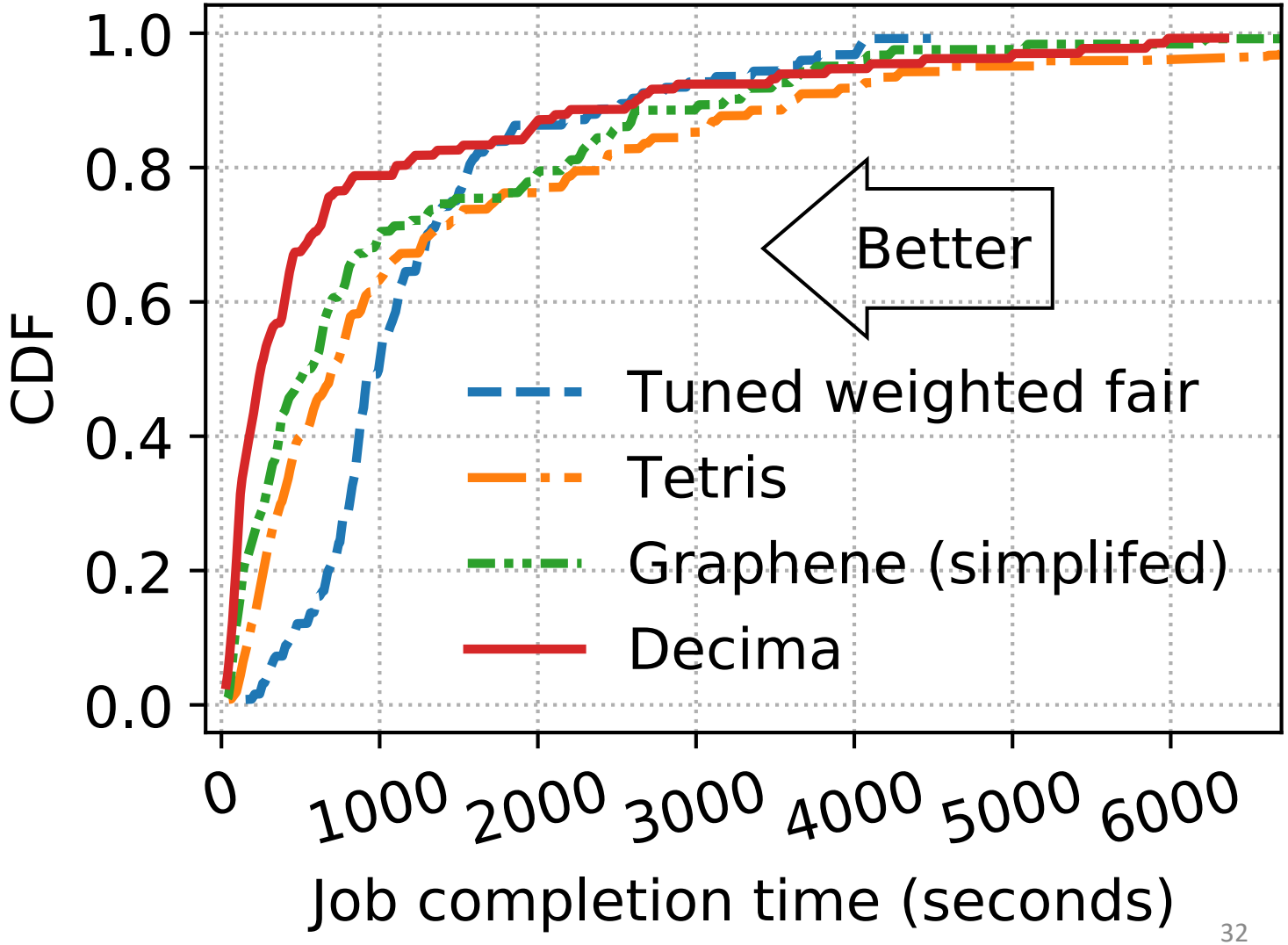
Flexibility: Multi-Resource Scheduling

Industrial trace (Alibaba):

20,000 jobs from production cluster

Multi-resource requirement:

CPU cores + memory units



Other Evaluations

- Impact of each component in the learning algorithm
- Generalization to different workloads
- Training and inference speed
- Handling missing features
- Optimality gap

Summary

- Decima uses reinforcement learning to generate workload-specific scheduling algorithms
- Decima employs curriculum learning and variance reduction to enable training with stochastic job arrivals
- Decima leverages a scalable graph neural network to process arbitrary number of job DAGs
- Decima outperforms existing heuristics and is flexible to apply to other applications

http://web.mit.edu/decima/