RLScheduler: An Automated HPC Batch Job Scheduler Using Reinforcement Learning

Di Zhang[1], Dong Dai[1], Youbiao He[2], Forrest Sheng Bao[2], Bing Xie[3]

[1]University of North Carolina at Charlotte

[2]Iowa State University

[3]Oak Ridge National Laboratory

Motivation & Background

RLScheduler Design

Evaluation & Analysis

Conclusion

Motivation & Background

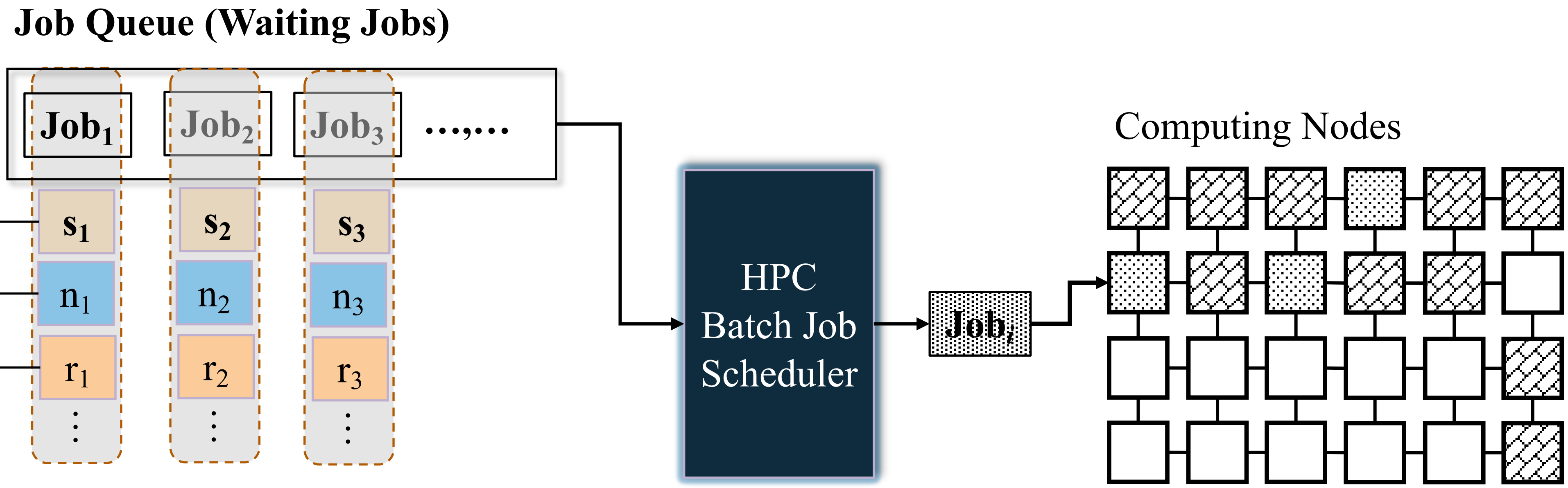
RLScheduler Design

Evaluation & Analysis

Conclusion

- Introduction of HPC batch job schedulers
- Challenges of existing schedulers
- Background of Reinforcement Learning

HPC Batch Job Scheduler

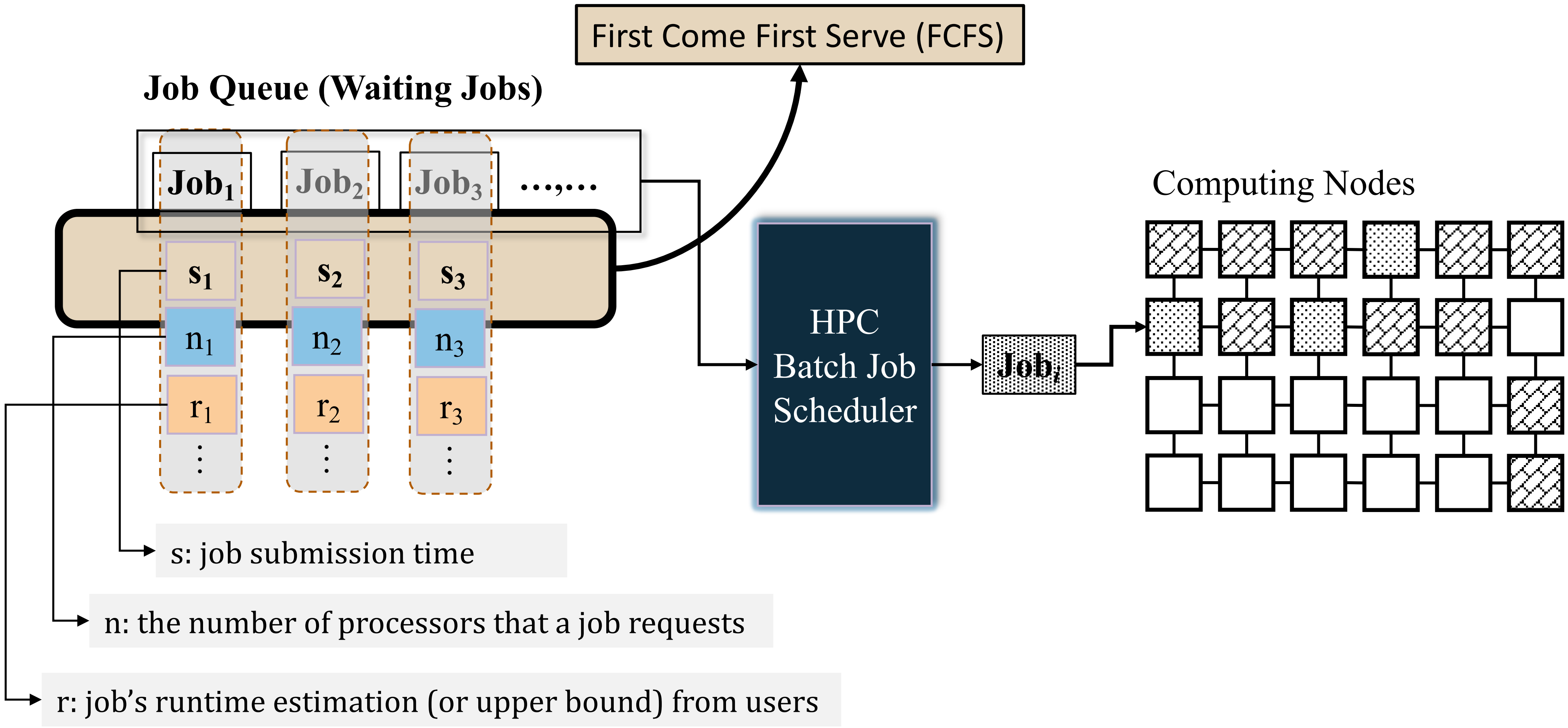


s: job submission time

n: the number of processors that a job requests

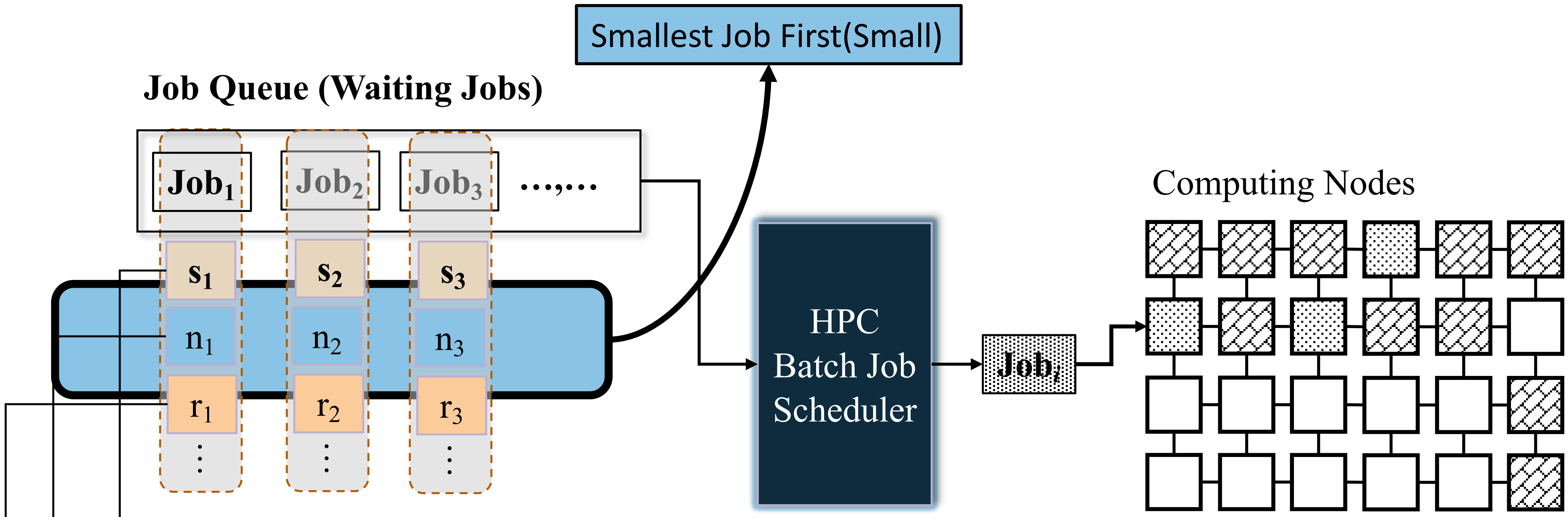
r: job's runtime estimation (or upper bound) from users

HPC Batch Job Scheduler



SC20

HPC Batch Job Scheduler

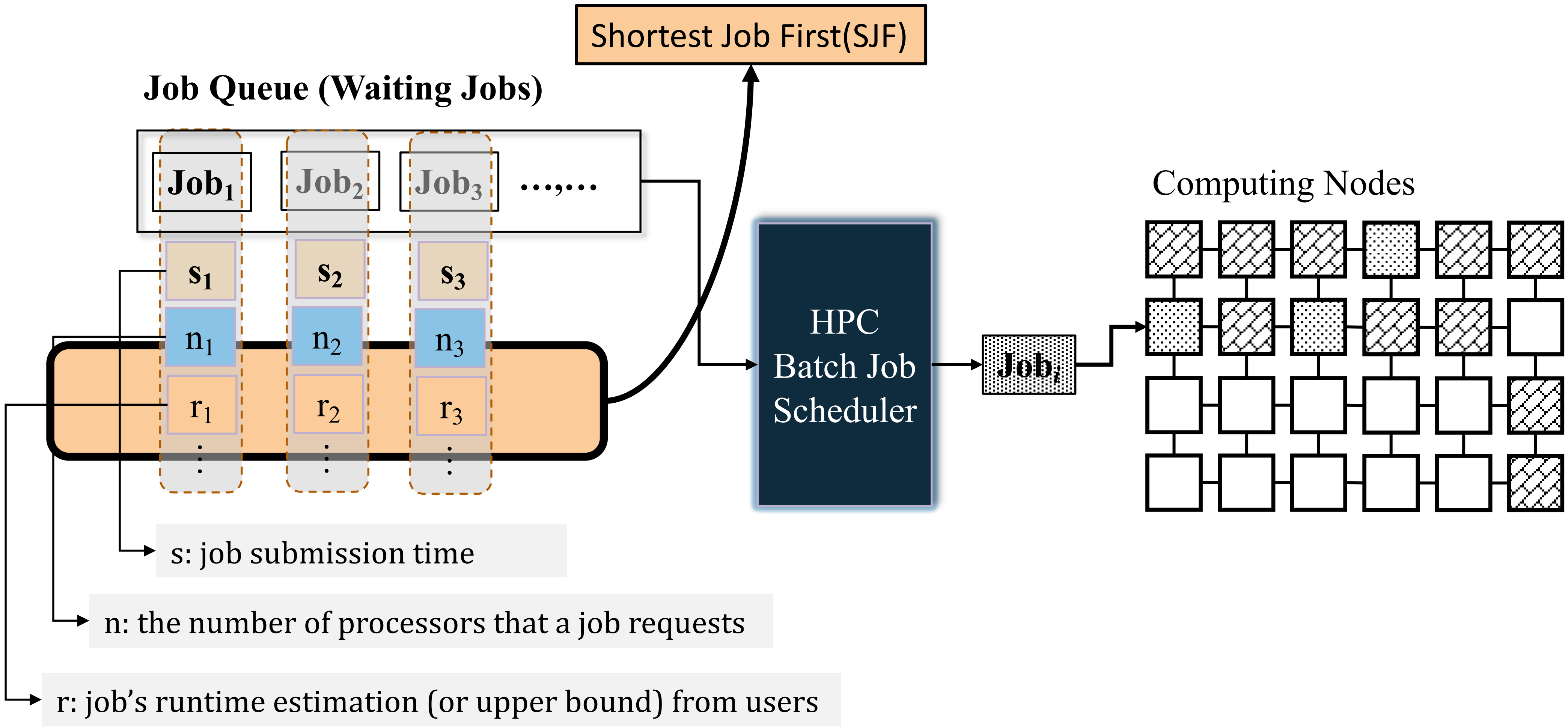


s: job submission time

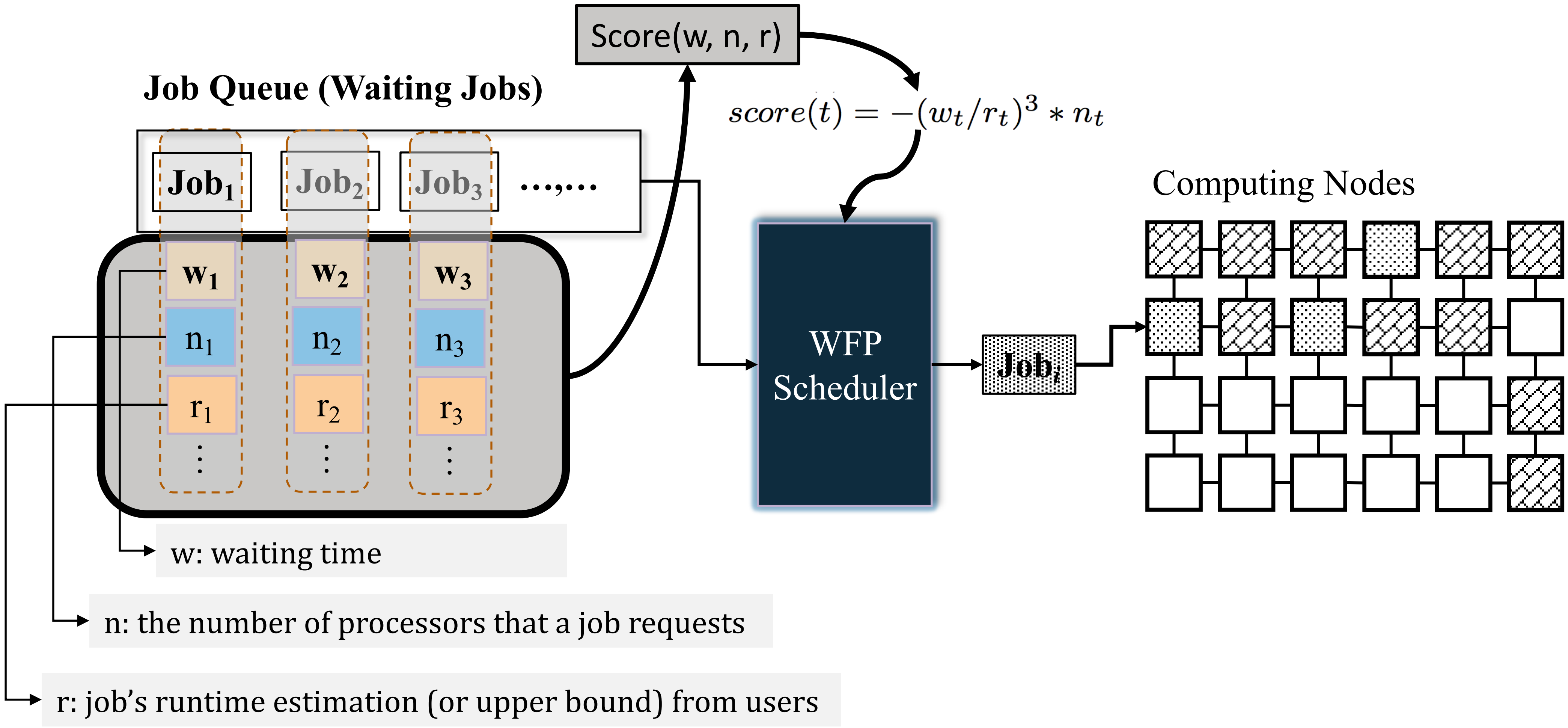
n: the number of processors that a job requests

r: job's runtime estimation (or upper bound) from users

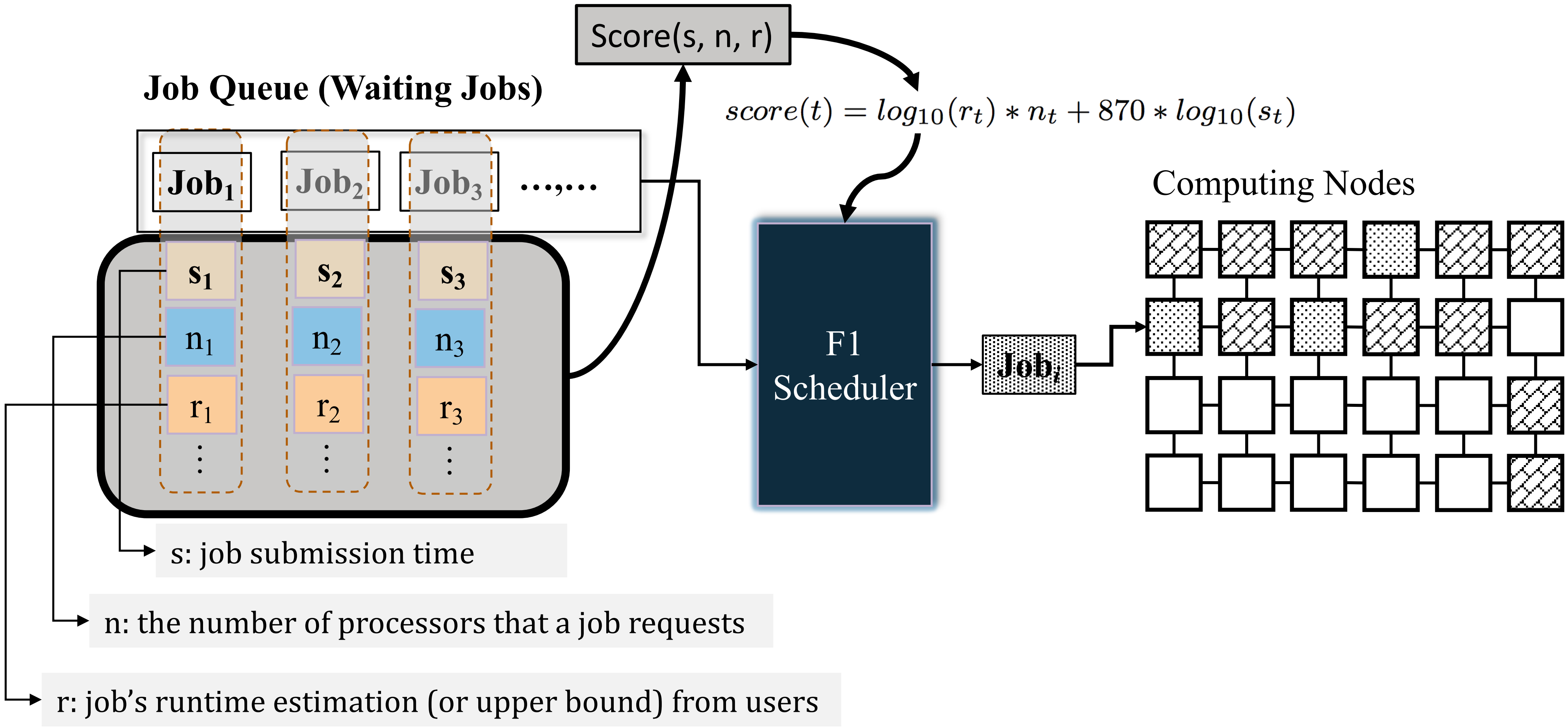
HPC Batch Job Scheduler



HPC Batch Job Scheduler



HPC Batch Job Scheduler

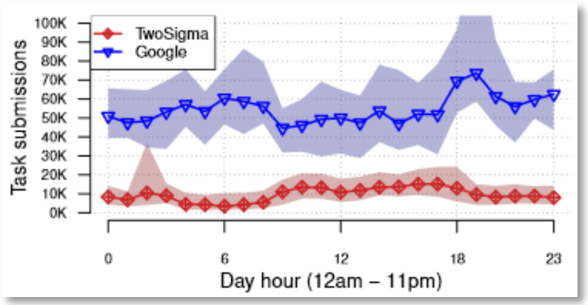


For a Given Scheduler

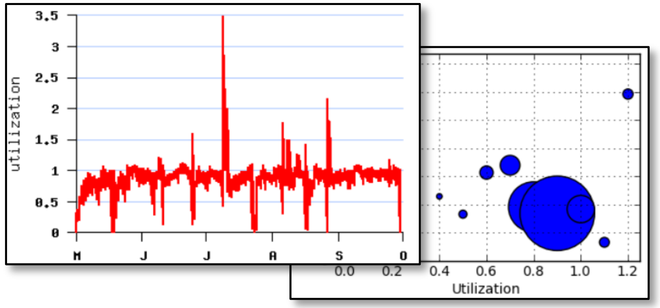
Different Job Traces

Different Scheduling Goals

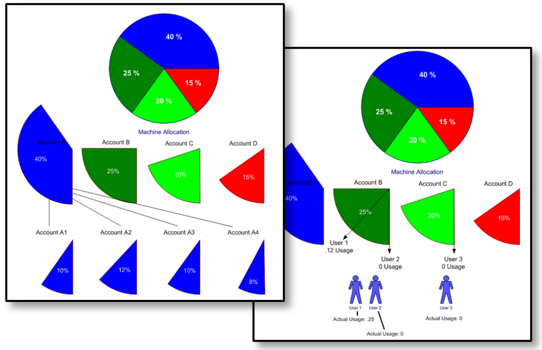
Complicated Scheduling Goals



Amvrosiadis, et. al. On the diversity of cluster workloads and its impact on research results, USNIX ATC'18



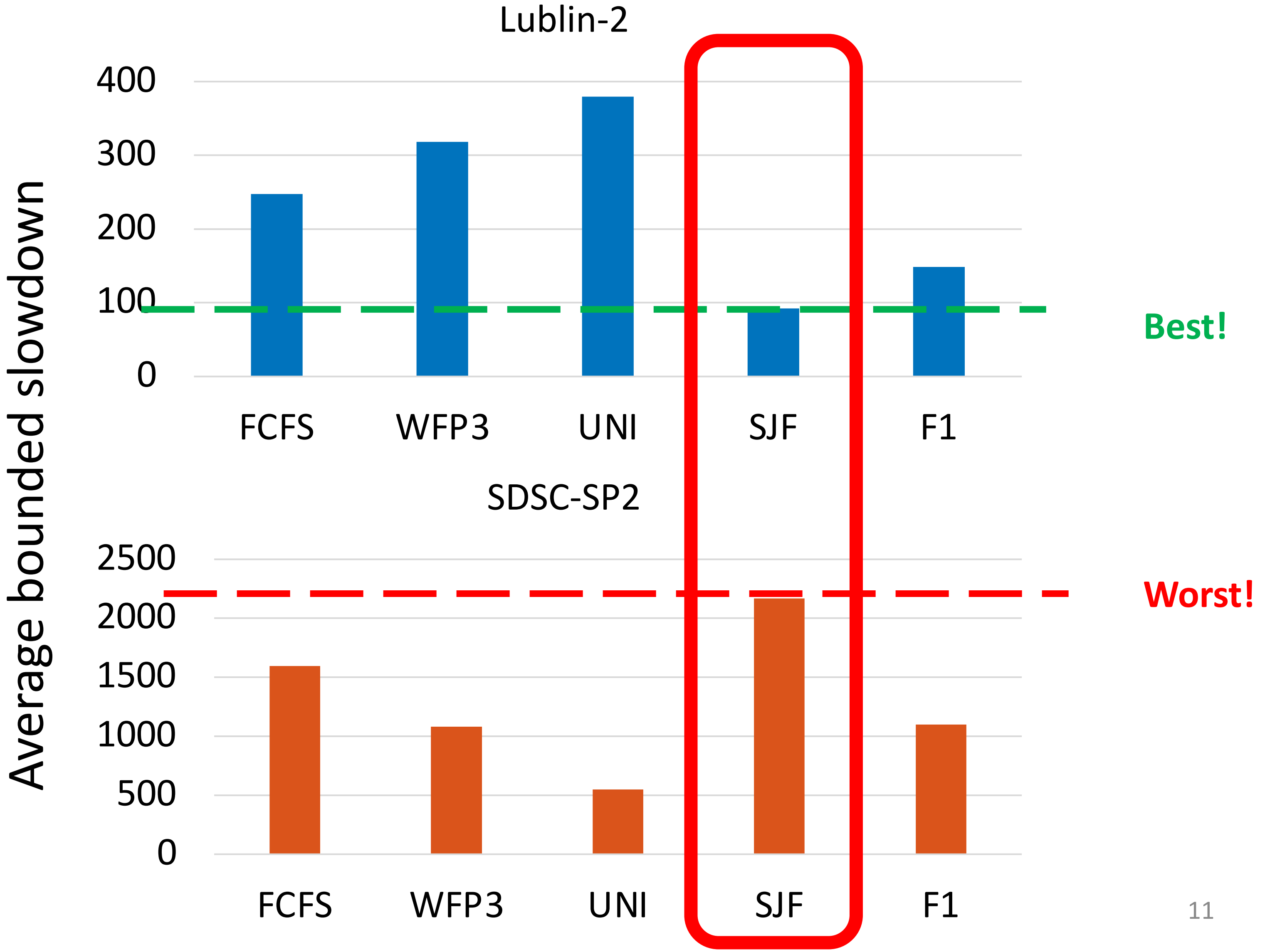
From https://www.cs.huji.ac.il/labs/parallel/workload/l_ricc/index.html



Slurm classic Fair Share https://slurm.schedmd.com/classic_fair_share.html

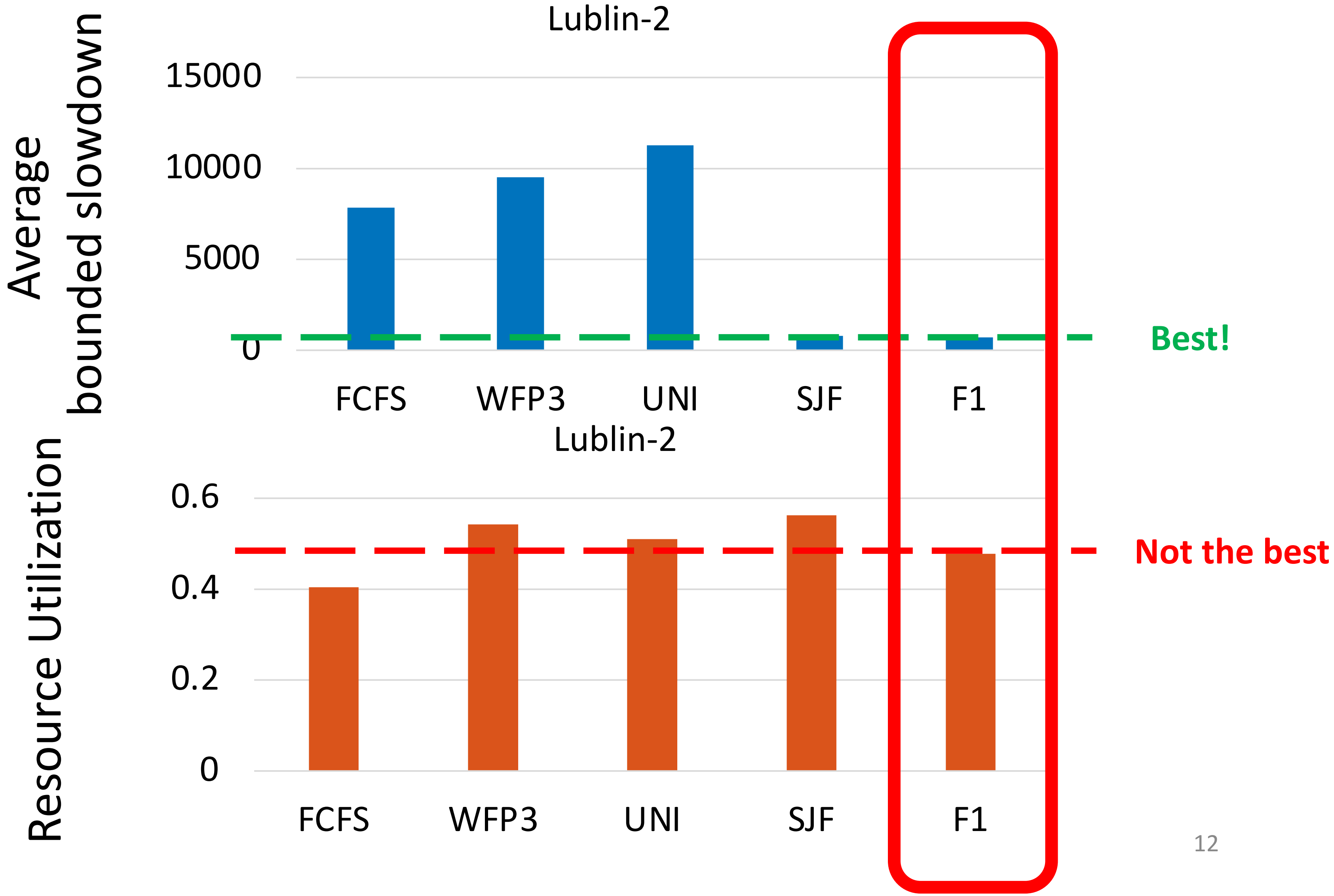
Impact of Different Job Traces

Job Schedulers behave differently on Different Job Traces



Impact of Scheduling Goals

Job schedulers behave differently toward different goals



Impact of Complicated Goals

Maximize Resource Utilization

Minimize Average Bounded Slowdown

Maximize User Fairness

Minimize Average Turnaround time

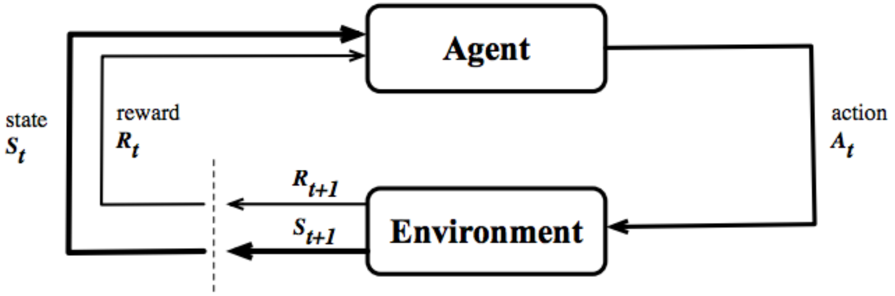
Minimize Average Waiting time

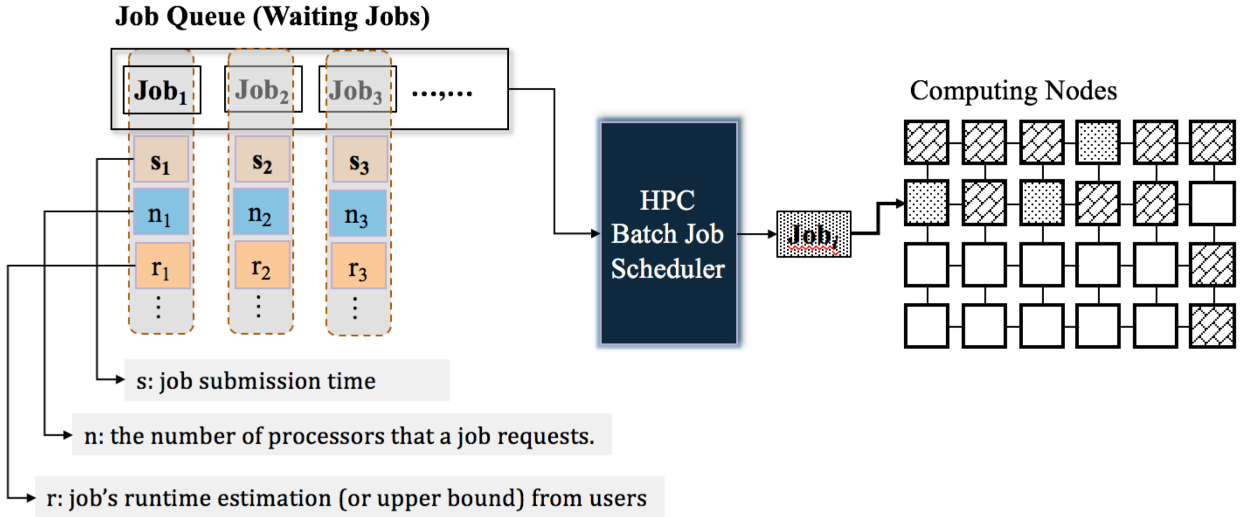
What Scheduling Policy?

Minimize Average Bounded Slowdown
&
Maximize User Fairness

Complicated Goals require new schedulers

Reinforcement Learning





Different Job Traces

Different Scheduling Goals

Complicated Scheduling Goals

Motivation & Background

RLScheduler Design

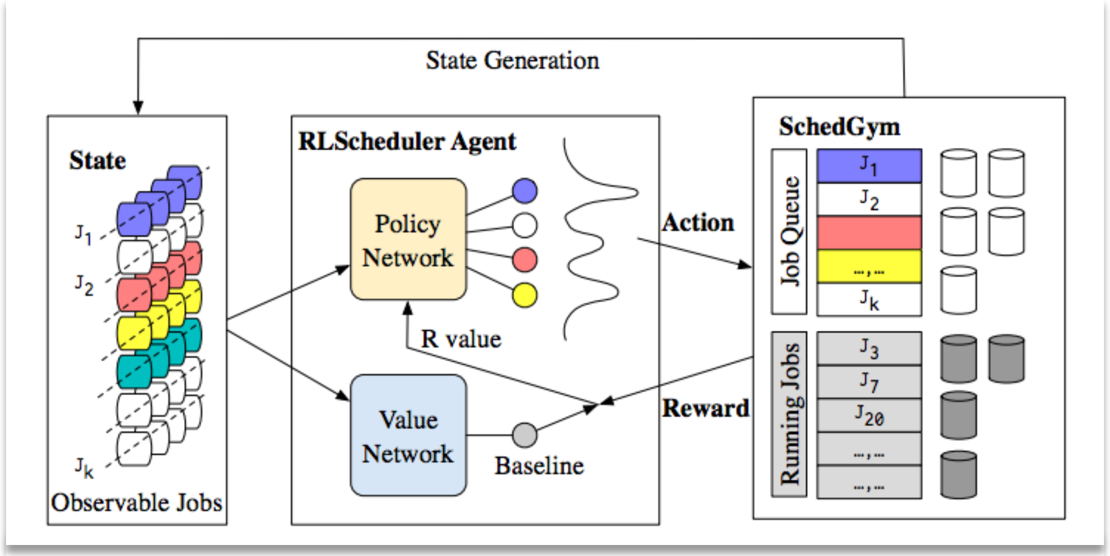
Evaluation & Analysis

Conclusion

- Overview of RLScheduler
- Challenges and Our Solutions

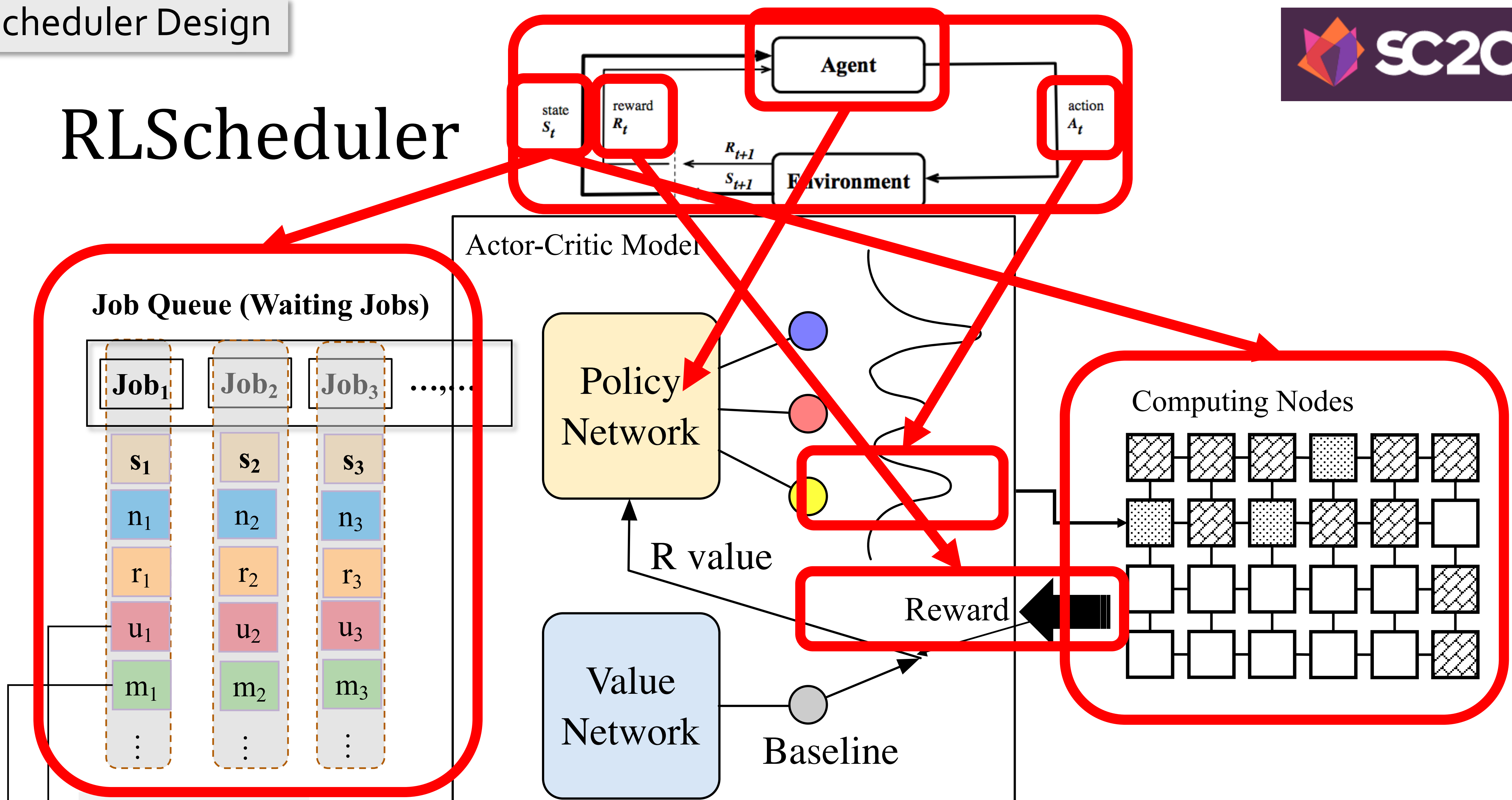
Our Contributions

- The first reinforcement learning based batch job scheduler for HPC systems
- New neural network and trajectory filtering mechanism to enable efficient RL training
- Extensively evaluations on efficiency, usability, and stability of RLScheduler.



RLScheduler Design

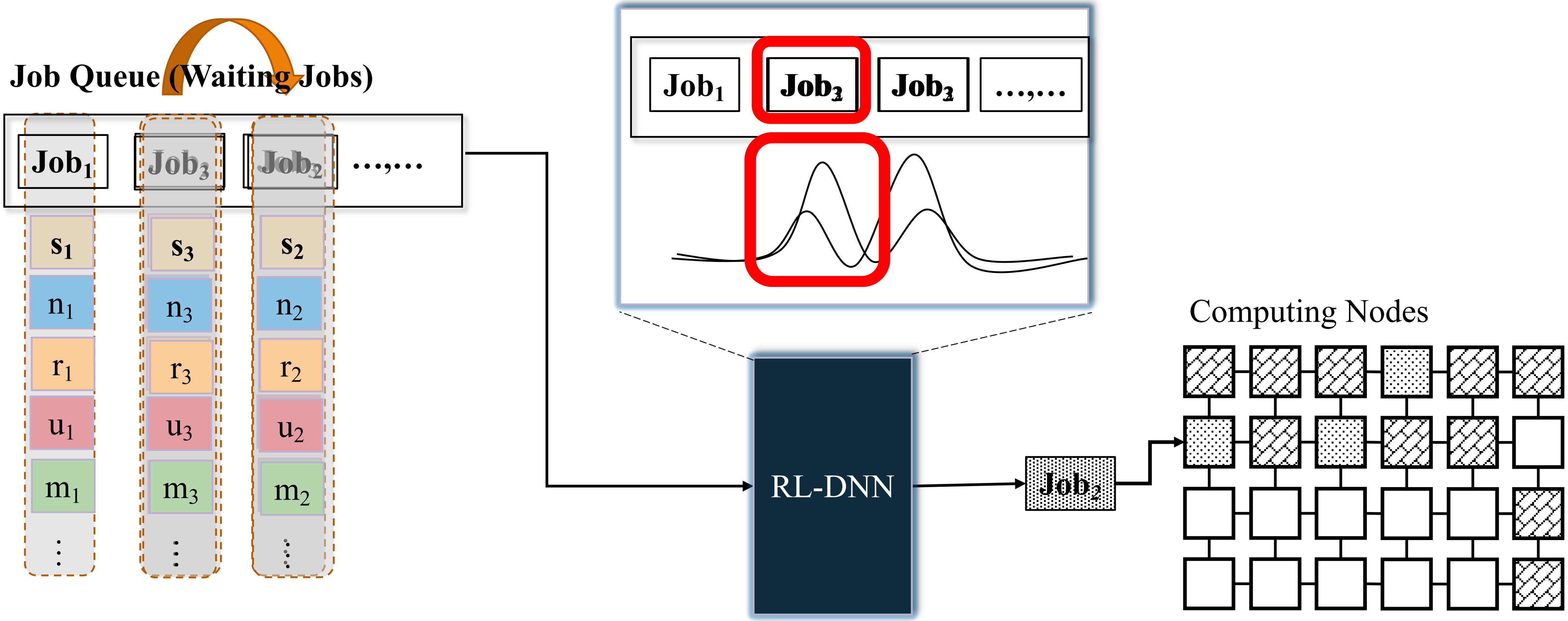
RLScheduler



u: the user's ID.

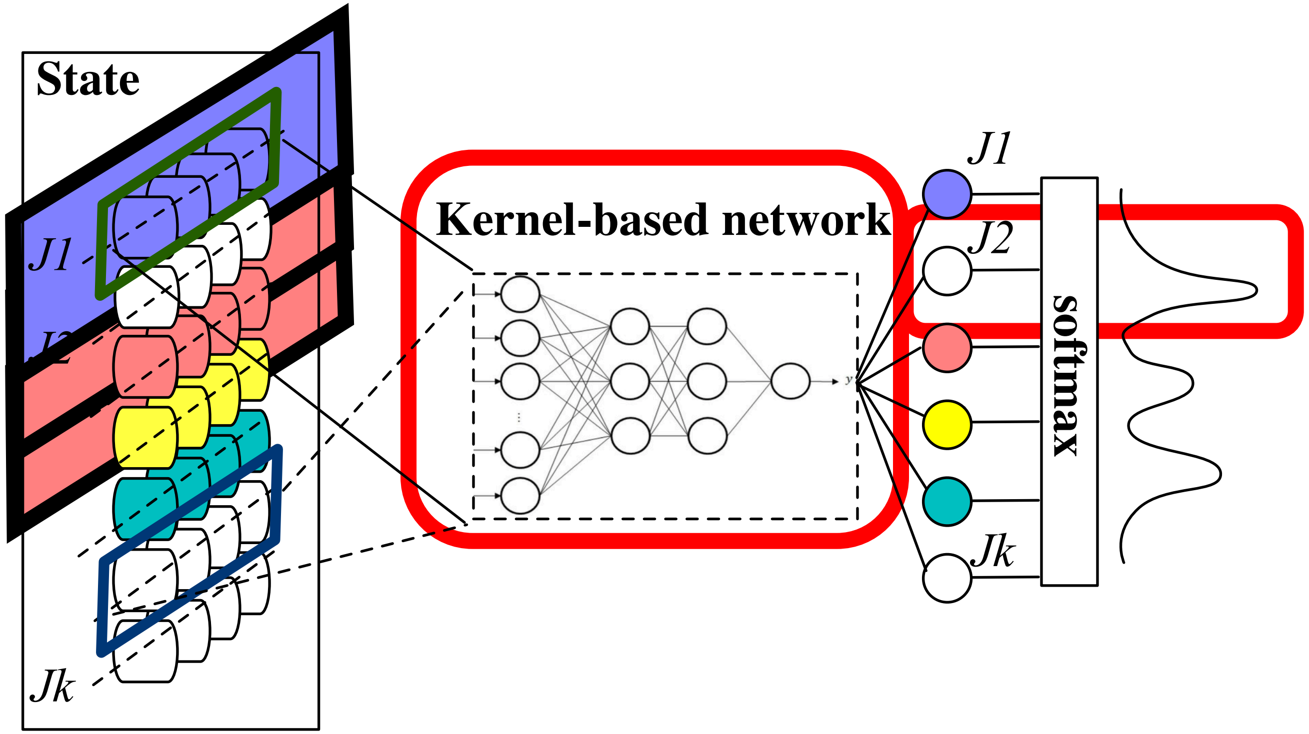
m: the request memory per processor

Challenge 1: Impact of Input Order



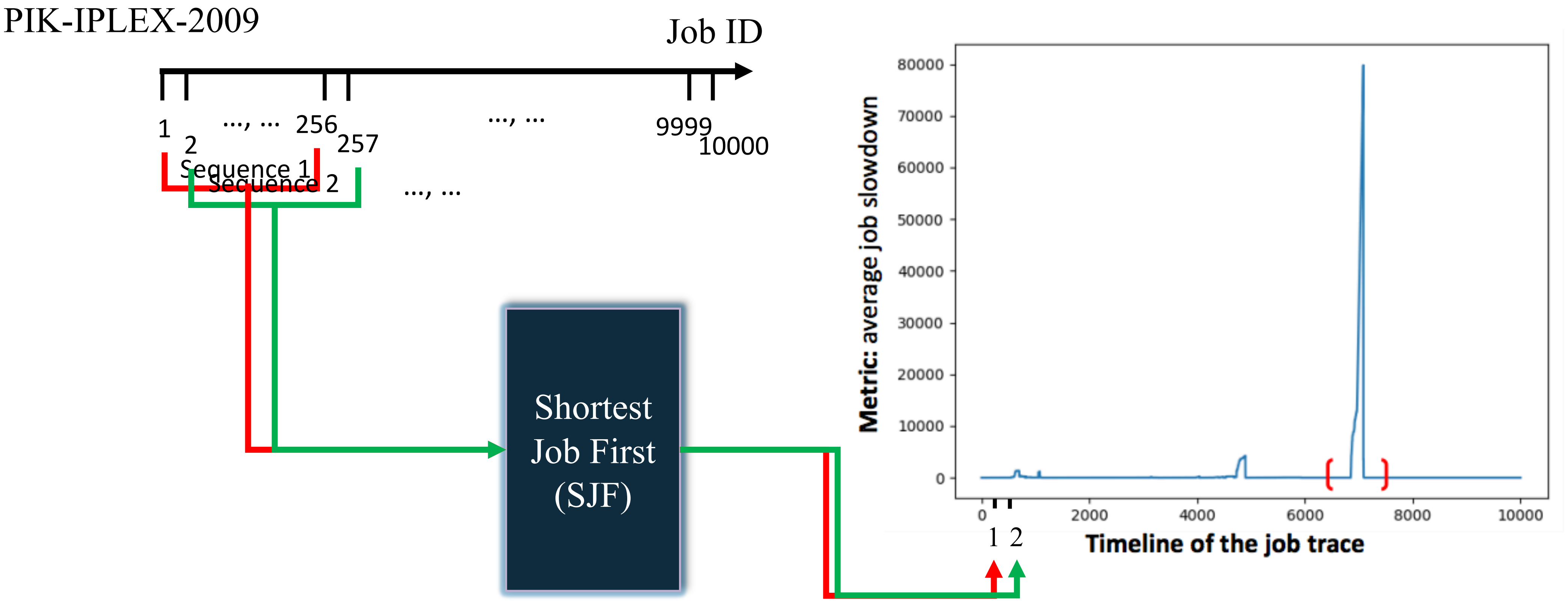
Solution: Kernel-based Policy Network

Kernel-based Policy Network is insensitive to the order of jobs



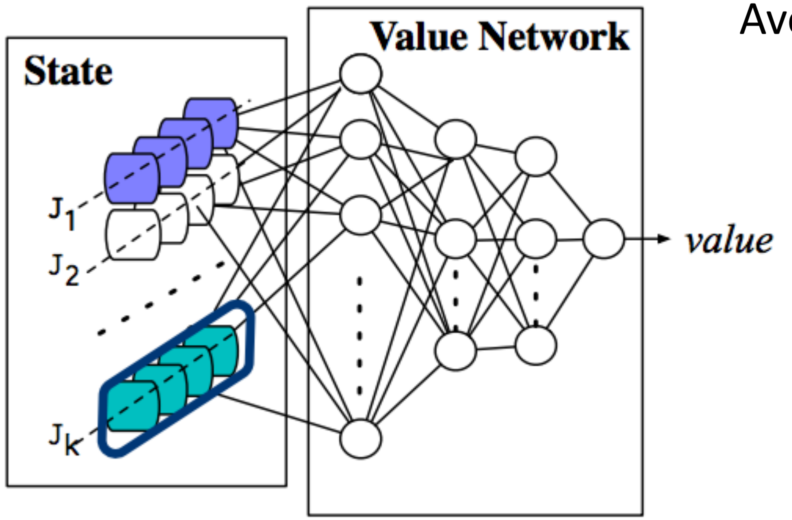
Policy network: Kernel-based network

Challenge 2: High Variance in Samples

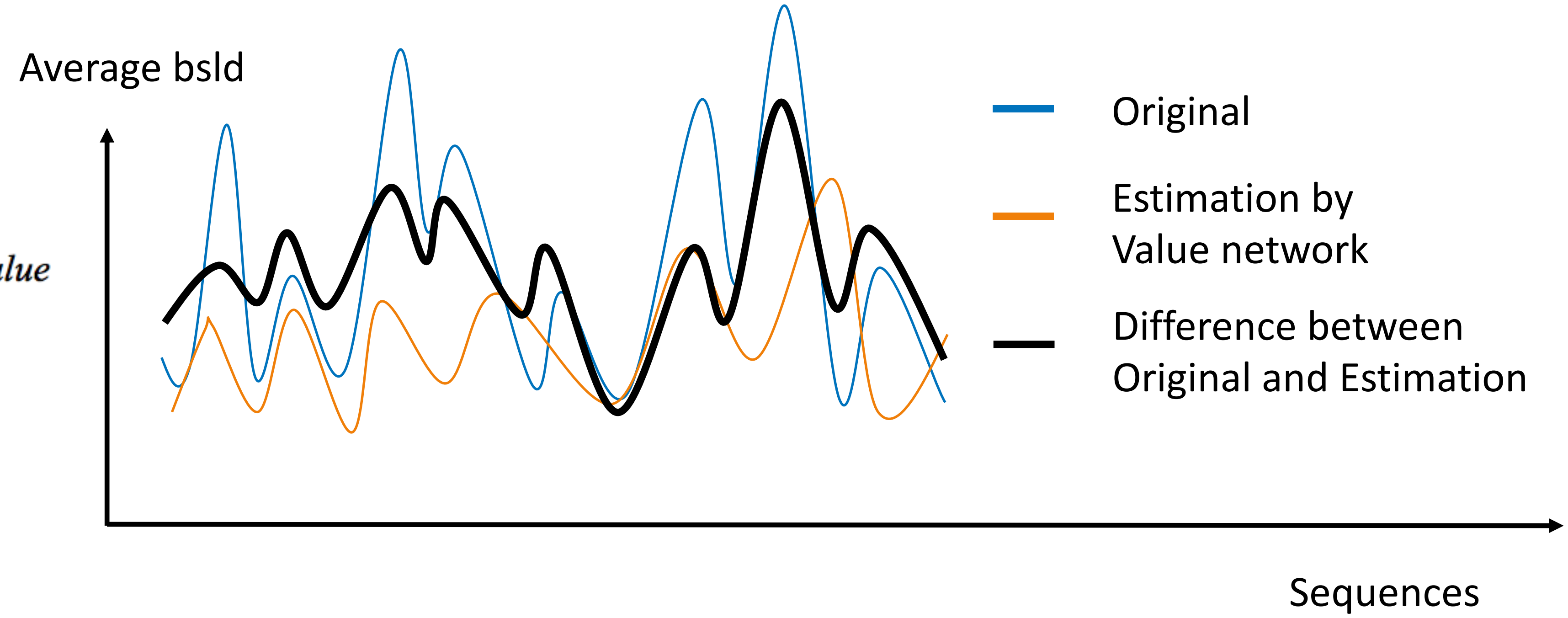


The average bounded slowdown of scheduling sequence of 256 jobs in PIK-IPLEX-2009 job trace.

Solution 1: Value Network

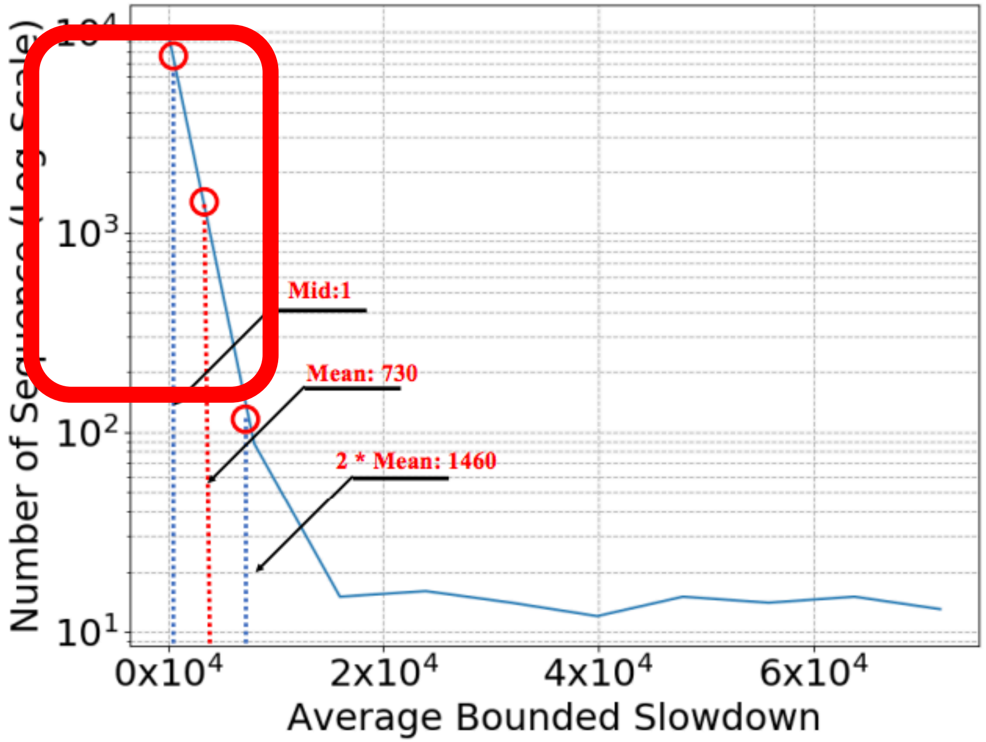


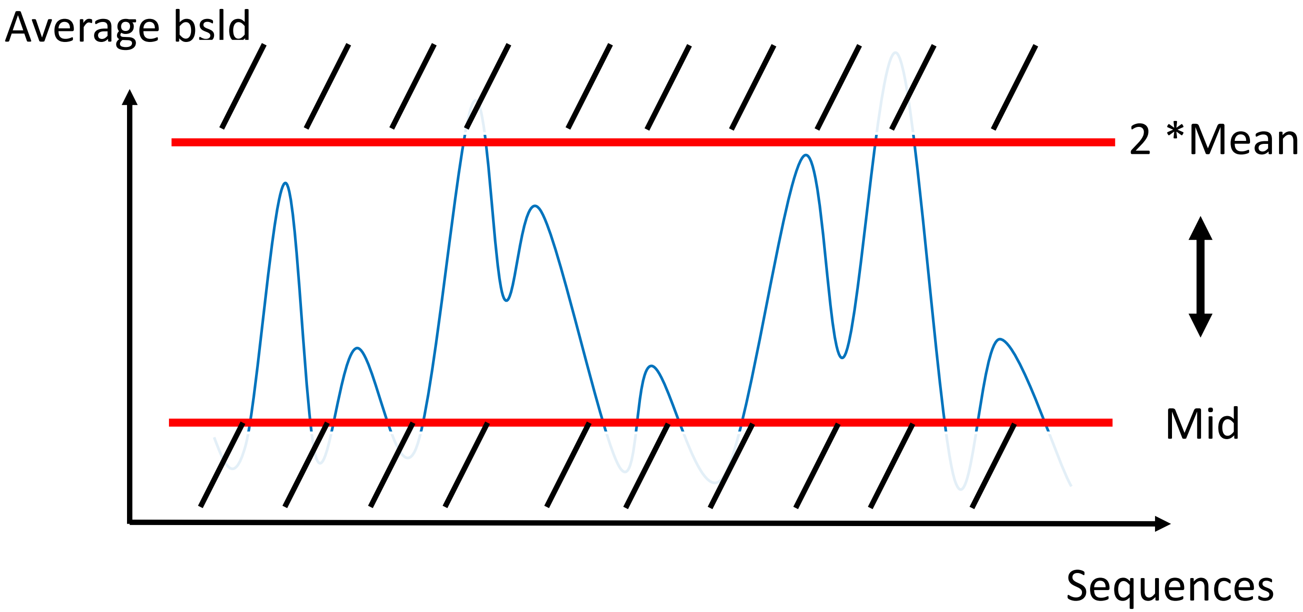
Value network



Value Network helps reduce the variance by estimating the value of different states

Solution 2: Trajectory Filter





Filter out jobs and retrain jobs with average bounded slowdown in between Mid and 2*Mean

Motivation & Background

RLScheduler Design

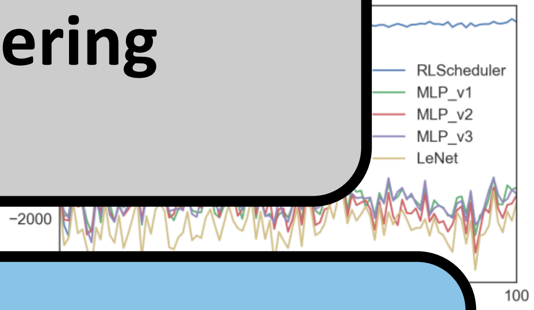
Evaluation & Analysis

- Efficiency Evaluations
- Usability Evaluations
- Stability Evaluations

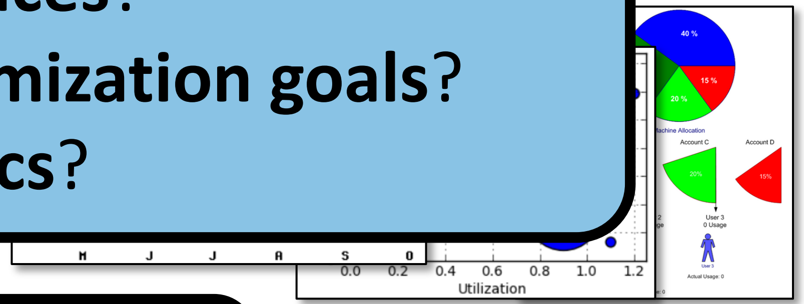
Conclusion

Evaluation & Analysis

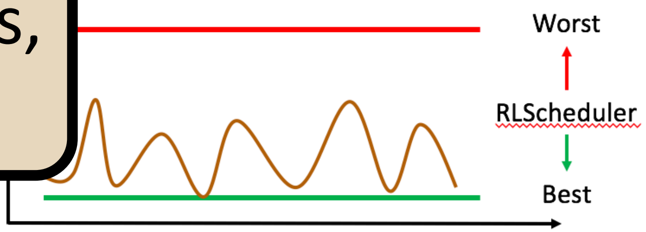
- How is the performance of **kernel-based neural network**?
- How well can **value network** and **trajectory filtering** reduce the variance?



- How is the performance on **various job traces**?
- How is the performance for **different optimization goals**?
- How is the performance of **complex metrics**?



If RLScheduler works well in above scenarios,
is it stable?



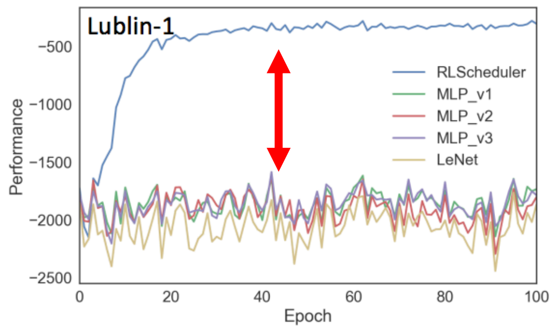
Evaluation & Analysis

Evaluation Outline

Efficiency

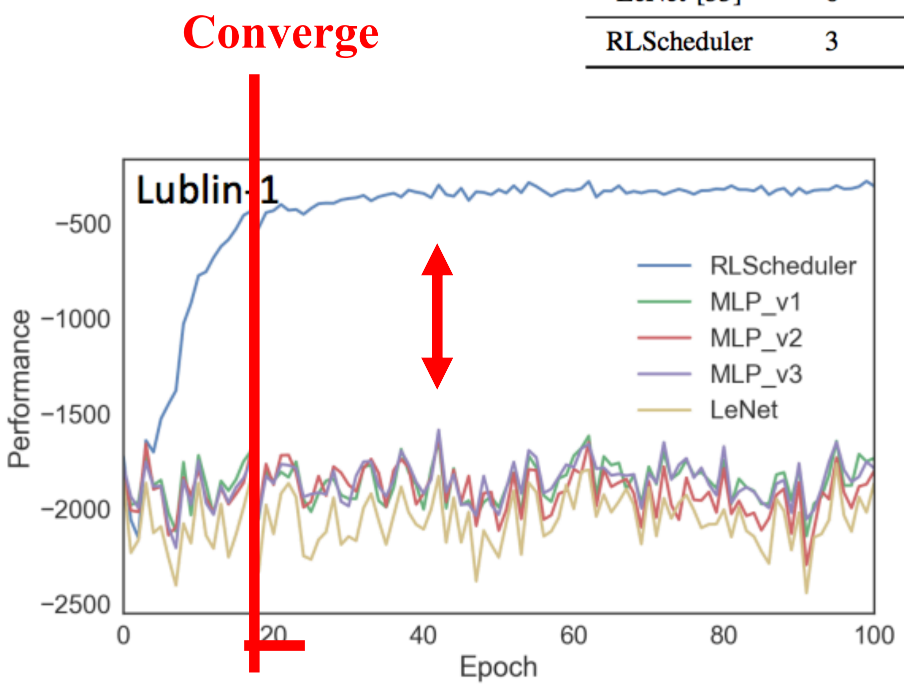
Usability

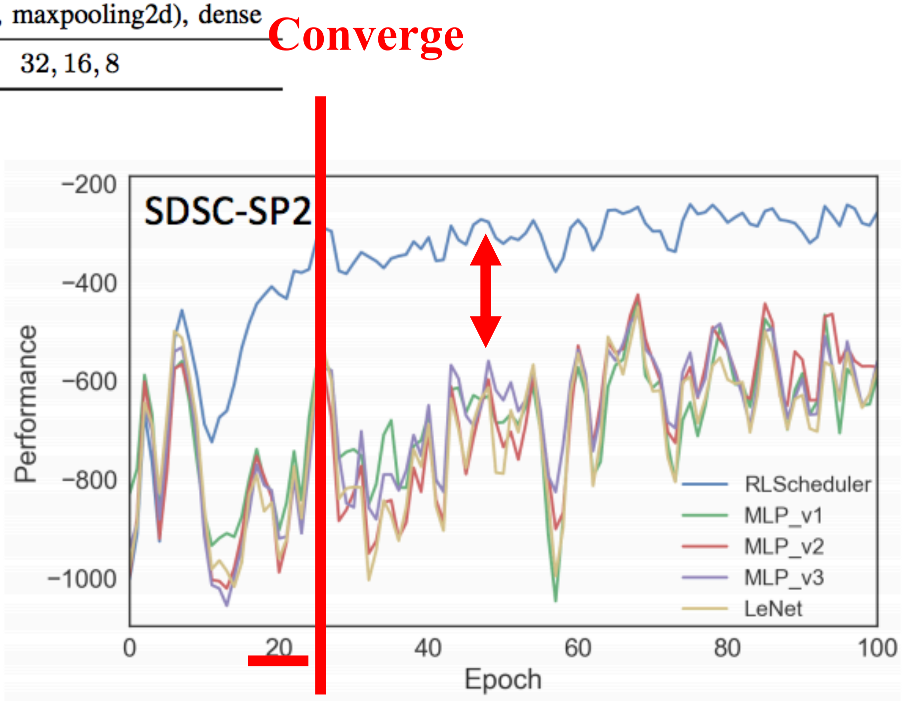
Stability



Compare Different Neural Networks

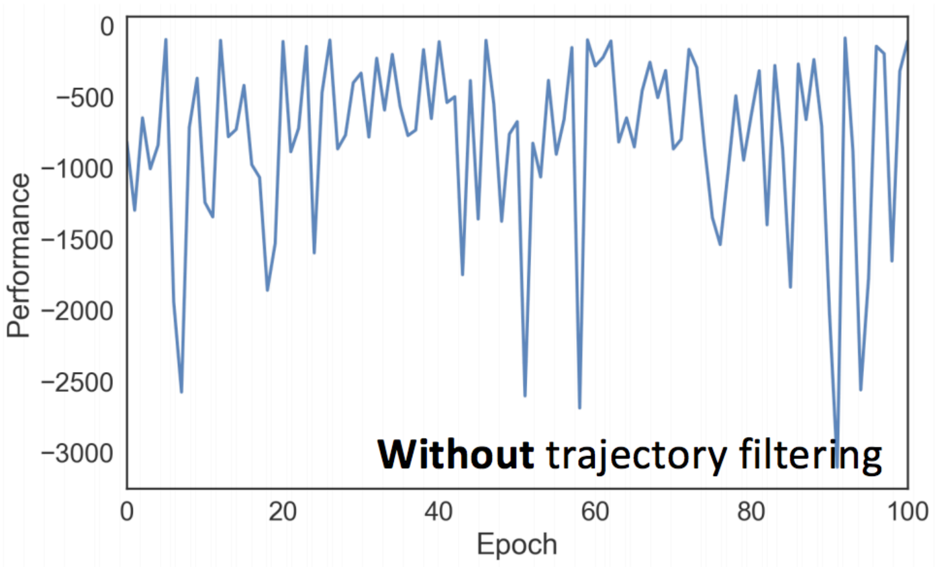
Name	Layers	Size of each layer
MLP_v1	3	128, 128, 128
MLP_v2	3	32, 16, 8
MLP_v3	5	32, 32, 32, 32, 32
LeNet [33]	6	2x(conv2d, maxpooling2d), dense
RLScheduler	3	32, 16, 8

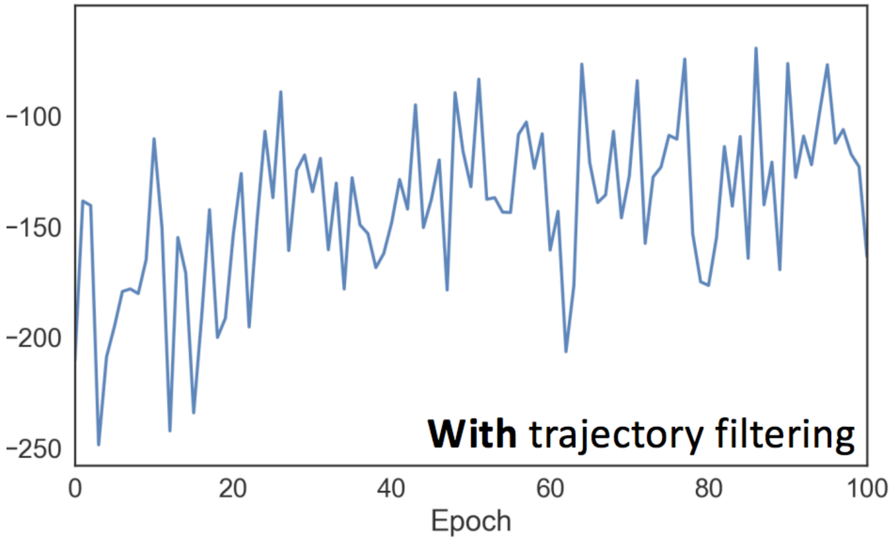




The horizontal axis shows the total number of training epoch; the vertical axis shows the performance of the agent. The larger vertical axis value indicates a smaller average bounded job slowdown and is better

With/Without Trajectory Filtering





The training curves of RLScheduler on PIK-IPLEX2009 job trace with and without trajectory filtering.

With trajectory enabled, RLScheduler converges within 100 epochs.

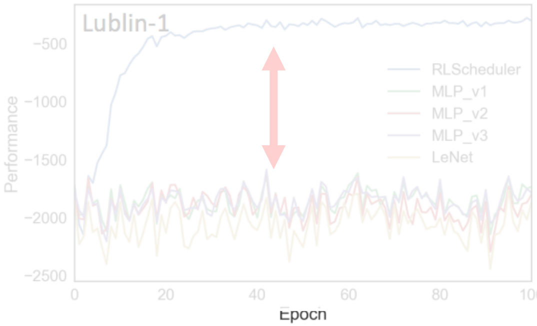
Evaluation & Analysis

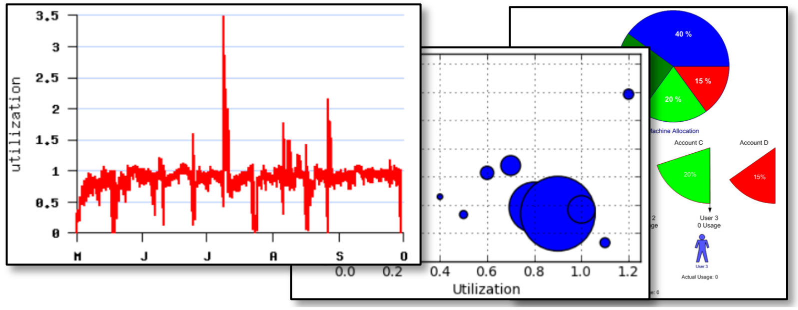
Evaluation Outline

Efficiency

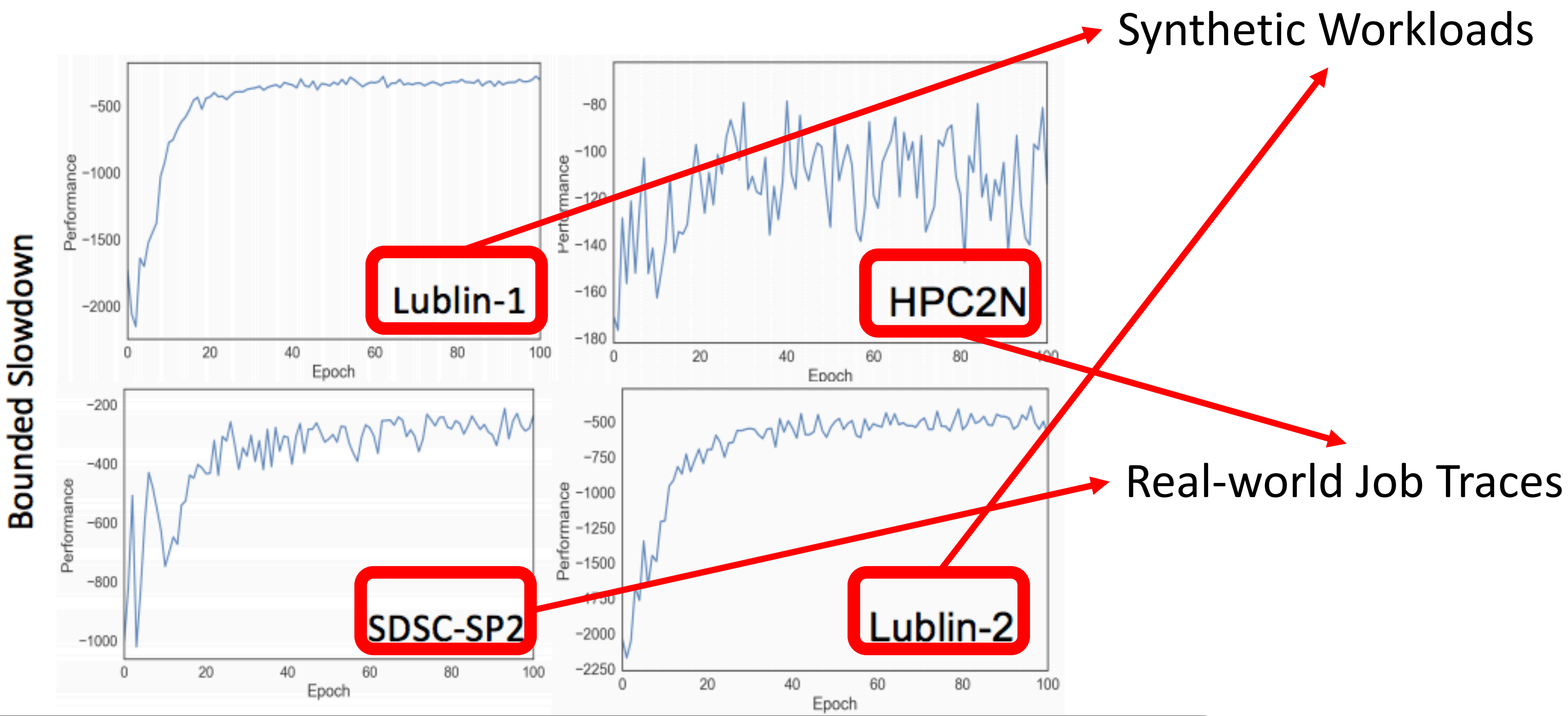
Usability

Stability





Training on **Different Job Traces**:



RLScheduler converges in all of the workloads within 100 training epochs and different job traces have different converge pattern.

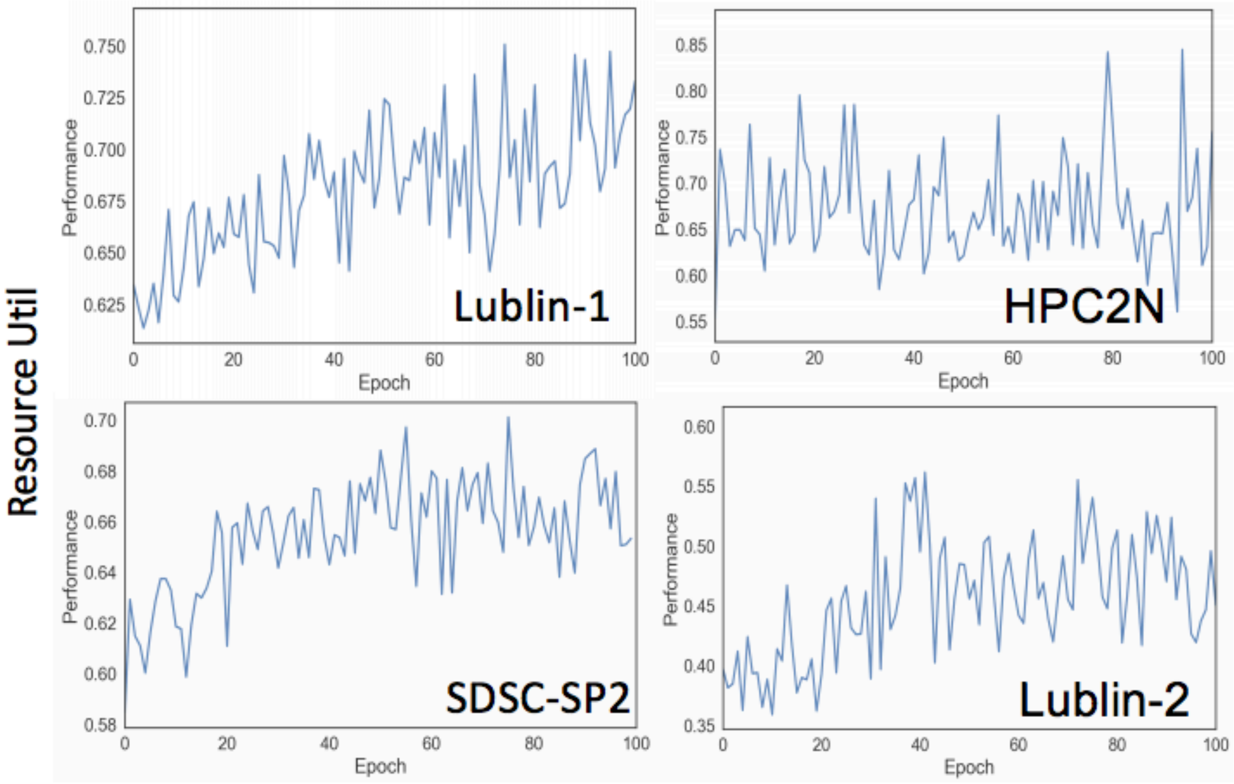
SC20

Testing on **Different Job Traces**:

Trace	FCFS	WFP3	UNI	SJF	F1	RL
*Scheduling **without** Backfilling*						
Lublin-1	7273.8	19754	22275	277.35	258.37	**254.67**
SDSC-SP2	1727.5	3000.9	1848.5	2680.6	1232.1	**466.44**
HPC2N	297.18	426.99	609.77	157.71	118.01	**117.01**
Lublin-2	7842.5	9523.2	11265	787.89	**698.34**	724.51
*Scheduling **with** Backfilling*						
Lublin-1	235.82	133.87	307.23	73.31	75.07	**58.64**
SDSC-SP2	1595.1	1083.1	548.01	2167.8	1098.2	**397.82**
HPC2N	127.38	97.39	175.12	122.04	**71.95**	86.14
Lublin-2	247.61	318.35	379.59	**91.99**	148.25	118.79

RLScheduler performs either comparably well to the best or is the best among the presented schedulers.

Training on **Different Goals**:



RLScheduler converges towards this new goal but with different patterns

Testing on **Different Goals**:

Trace	FCFS	WFP3	UNI	SJF	F1	**RL**
*Scheduling **without** Backfilling*						
Lublin-2	7842.5	9523.2	11265	787.89	**698.34**	724.51

Average bounded slowdown

Best!

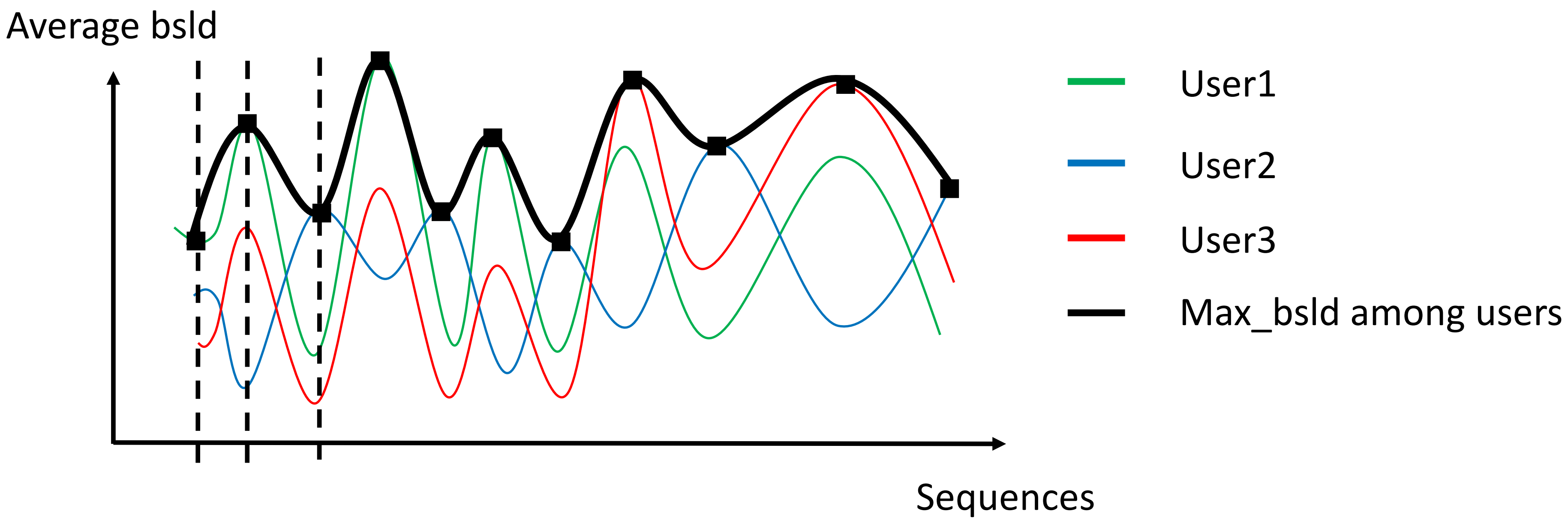
Not the best

Trace	FCFS	WFP3	UNICEP	SJF	F1	**RL**
*Scheduling **without** Backfilling*						
Lublin-1	0.657	0.747	0.691	0.762	**0.816**	0.714
SDSC-SP2	0.670	0.658	**0.688**	0.645	0.674	0.671
HPC2N	0.638	0.636	0.636	0.640	0.637	**0.640**
Lublin-2	0.404	0.543	0.510	0.562	0.478	**0.562**
*Scheduling **with** Backfilling*						
Lublin-1	0.868	0.864	**0.883**	0.778	0.840	0.850
SDSC-SP2	0.682	0.681	0.706	0.661	0.677	**0.707**
HPC2N	0.639	0.637	0.638	0.641	0.638	**0.642**
Lublin-2	0.587	0.583	0.587	0.593	0.552	**0.593**

Resource Utilization

RLScheduler has good performance among all the presented schedulers.

RLScheduler with Fairness



Minimizing maximal average bounded slowdown among users is a **complicated metrics** considering **Performance** and **Fairness** at the same time.

RLScheduler with Fairness

Trace	FCFS	WFP3	UNICEP	SJF	F1	**RL**
*Scheduling **without** Backfilling*						
SDSC-SP2	7257	14858	12234	12185	8260	**4116**
HPC2N	2058	5107	5145	1255	1310	**1147**
*Scheduling **with** Backfilling*						
SDSC-SP2	7356	8464	3840	10121	7799	**2712**
HPC2N	1502	2125	2081	1491	583	**519**

Results of scheduling different job traces towards average bounded slowdown with Maximal Fairness.

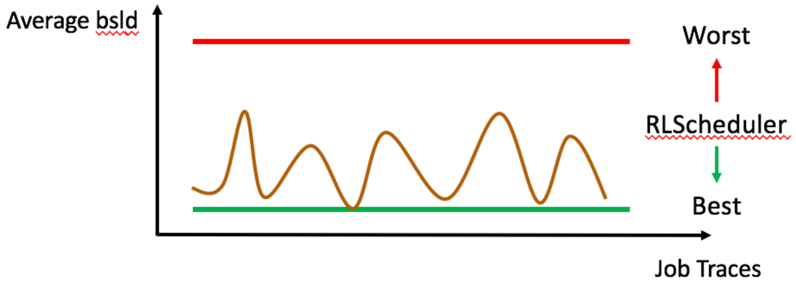
RLScheduler can consider multiple metrics at the same time: minimizing average bounded slowdown and keeping fairness among users together.

Evaluation Outline

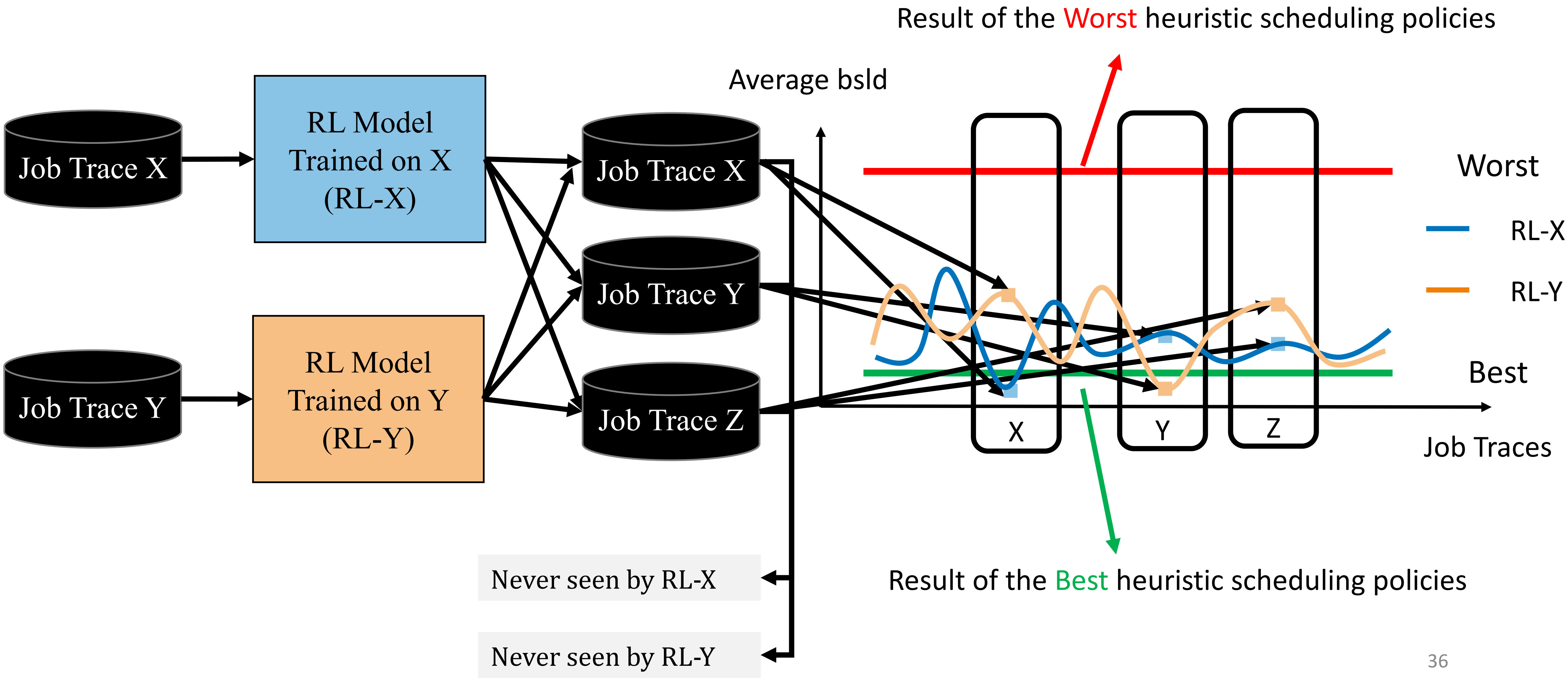
Efficiency

Usability

Stability



Stability Evaluation



Stability Evaluation

Trace	Best Heuristic Sched	Worst Heuristic Sched	RL-*Lublin-1*	RL-*SDSC-SP2*	RL-*HPC2N*	RL-*Lublin-2*
			*Scheduling **without** Backfilling*			
Lublin-1	258.37 (F1)	22274.74 (UNICEP)	**254.67**	482.62	283.00	334.73
SDSC-SP2	1232.06 (F1)	3000.88 (WFP3)	1543.40	**466.44**	1016.83	1329.41
HPC2N	**118.01 (F1)**	660.77 (UNICEP)	169.91	300.43	186.42	236.00
Lublin-2	698.34 (F1)	11265.3 (UNICEP)	665.49	805.16	648.52	**724.51**
ANL Intrepid	**8.39 (F1)**	35.11 (FCFS)	9.91	9.61	8.93	9.75
			*Scheduling **with** Backfilling*			
Lublin-1	73.31 (SJF)	307.23 (UNICEP)	58.64	93.16	**54.65**	64.45
SDSC-SP2	548.01 (UNICEP)	2167.84 (SJF)	1364.43	**397.82**	746.65	1192.97
HPC2N	**71.95 (F1)**	175.12 (UNICEP)	115.93	128.73	115.79	144.54
Lublin-2	**91.99 (SJF)**	379.59 (UNICEP)	172.15	183.98	139.80	118.79
ANL Intrepid	**2.73 (F1)**	4.12 (UNICEP)	3.63	4.56	3.99	3.58

A learned RLScheduler model, regardless of which job trace it was trained on, can be safely applied to other job traces

Motivation & Background

RLScheduler Design

Evaluation & Analysis

Conclusion

SC20

Summary

- We designed and implemented the first RL-based HPC batch job scheduler.
  - https://github.com/DIR-LAB/deep-batch-scheduler

DIR-LAB / **deep-batch-scheduler**

Watch 2 | Unstar 11 | Fork 2

deep-batch-scheduler

This repo includes the deep batch scheduler source code and necessary datasets to run the experiments/tests.

The code has been tested on Ubuntu 18.04/16.04 with Tensorflow 1.14 and SpinningUp 0.2. Newer version of Tensorflow (such as 2.x) does not work because of the new APIs. Windows 10 should be OK to run the code, only the installation of the dependencies (such as Gym environment and SpinningUp could be bumpy).

DOI 10.5281/zenodo.4009286

- We introduced new network design and trajectory filtering mechanism in RLScheduler to stabilize and speedup the training.
- We conducted extensive evaluations to show the efficiency, usability, and stability of RLScheduler across various HPC job traces and scheduling goals.

Thank you! & Questions?