DeepPlace: Learning to Place Applications in Multi-Tenant Clusters

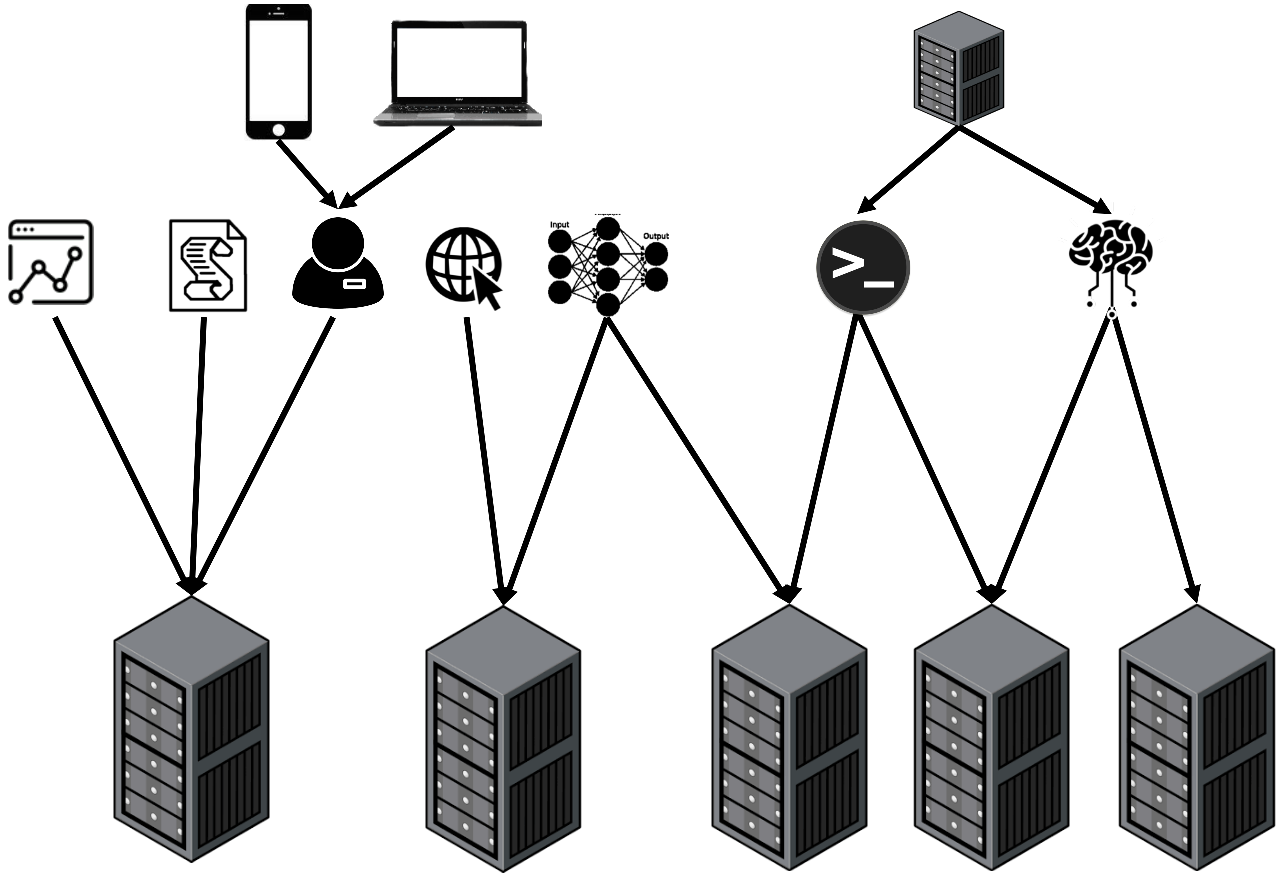
Nikhil Sheoran (Adobe Research - Bangalore, India)

Collaborators
Subrata Mitra (Adobe Research),
Shanka Subhra Mondal (IIT Kharagpur) *,
Neeraj Dhake (IIT Bombay) *,
Ravinder Nehra (IIT Roorkee) *,
Ramanuja Simha (Oakridge National Lab) *

* work done while at Adobe Research

The Multi-Tenant Computation Landscape

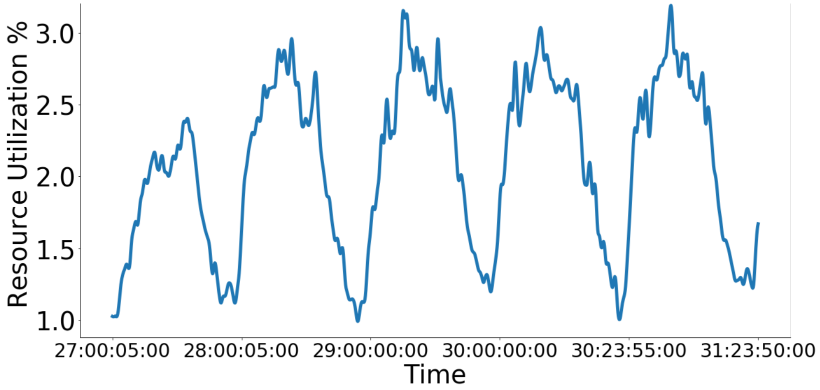
- Variety of **Applications**
  - Ex: User-facing, Batch Analytics, etc.
- Variety of **Resource Needs**
  - Ex: Resource intensive
- Variety of **User Expectations**
  - Ex: Latency Sensitive



Adobe

Improving Variance – Through Resource Limits

- Developers – Can specify **Resource Limits**.
- Overly Conservative estimates.
  - For adverse situation.
- Poor utilization.
  - Peak is (way) less than estimated.
  - Peak doesn't remain for majority of time.



Improving Variance – Through Constraints

- To give better control to developers, schedulers provide ways to **specify constraints.**
- Based on estimates (generally conservative) and heuristics.
- Issue – Limited Expressibility

```yaml
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: kubernetes.io/e2e-az-name
            operator: In
            values:
            - e2e-az1
            - e2e-az2
```

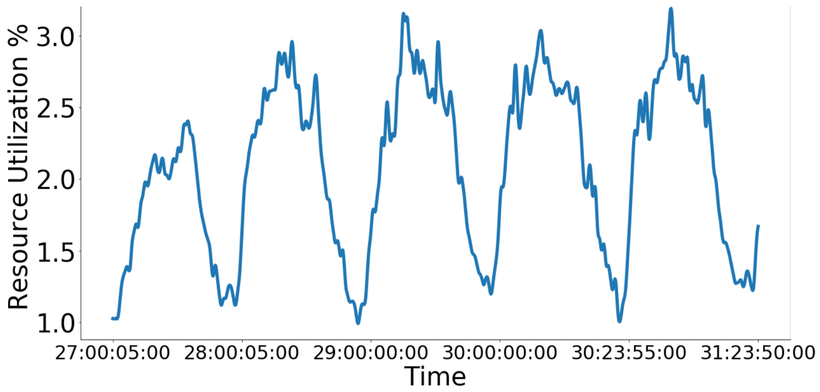
Affinity and Anti-Affinity Constraints in Kubernetes

```json
{
    "id": "simplehttpserver",
    "cmd": "sleep 30 && python -m SimpleHttpServer",
    "instances": 3,
    "constraints": [
        [
            "hostname",
            "UNIQUE"
        ]
    ]
}
```

Placement Constraints in Marathon (Apache Mesos)

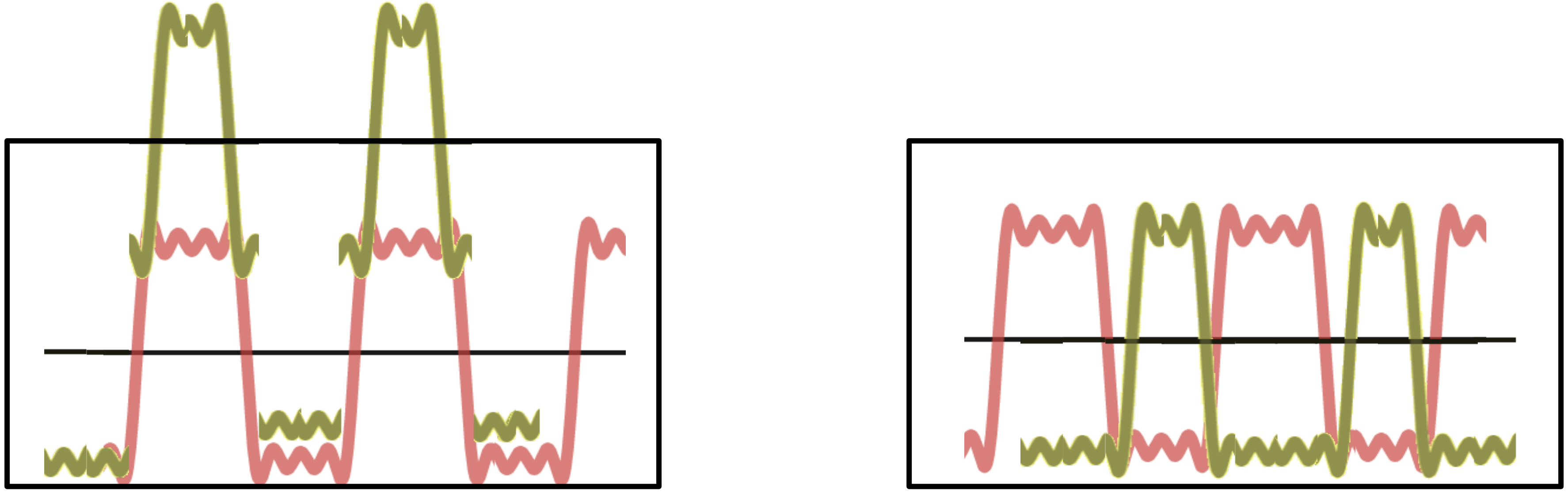
Where they fail – Temporal Patterns

- Temporal Patterns
  - Across Time (ex. Daily, Seasonal, etc.)
  - Across Algorithmic Phases (ex. Map-Reduce, etc.)
- Long-running Jobs
  - More peaks and valleys.
  - Relatively high predictability.
- Short-running Jobs
  - Can fit in valleys of long-running.
  - Though less predictability.



Where they fail – Temporal Alignment

- Minor temporal mis-alignment can lead to inefficient scheduling.
- Placement 1 overshoots the resource usage while Placement 2 efficiently completes.

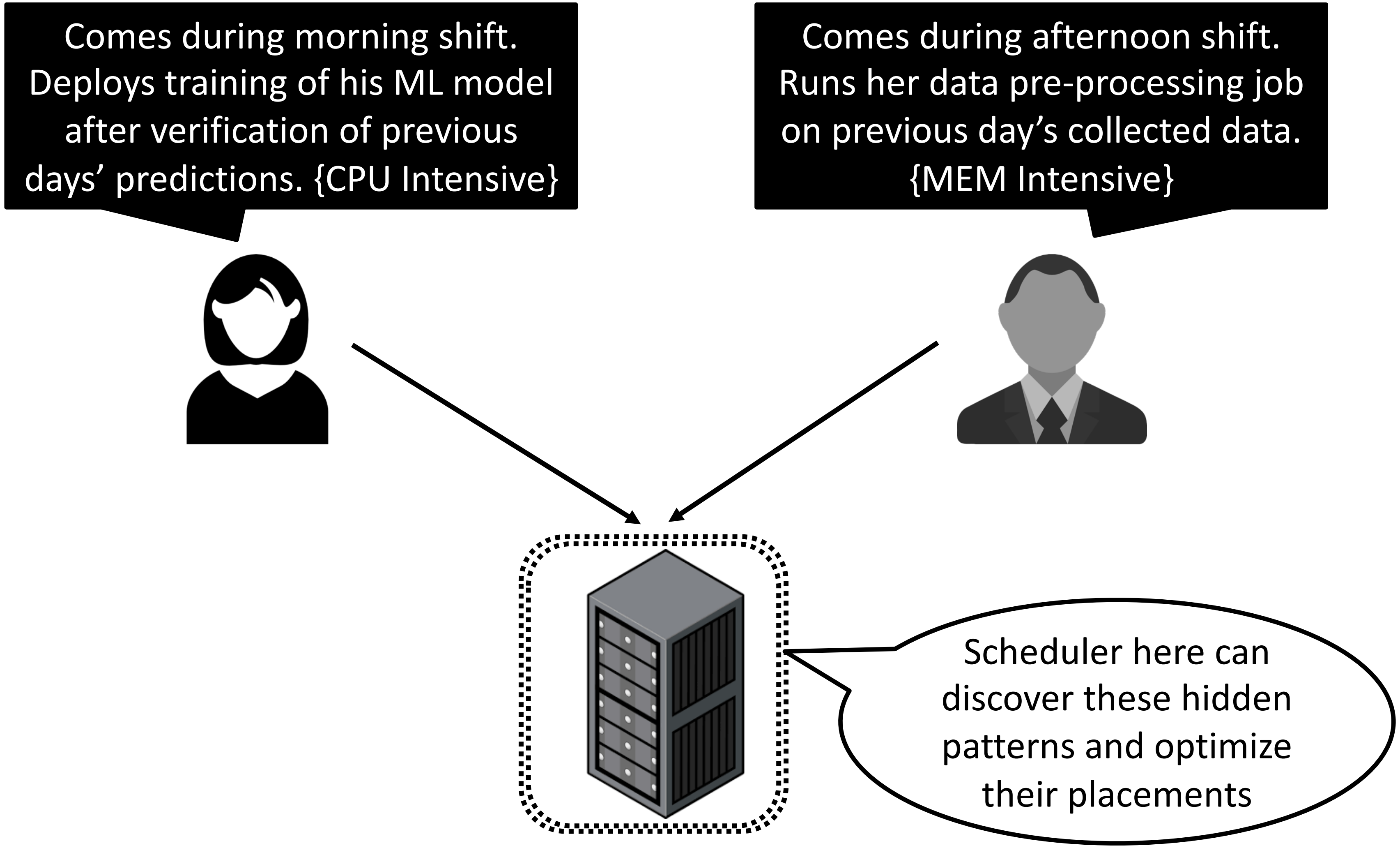


Placement 1 - Temporally Mis-aligned (Overshoots)

Placement 2 - Temporally Aligned

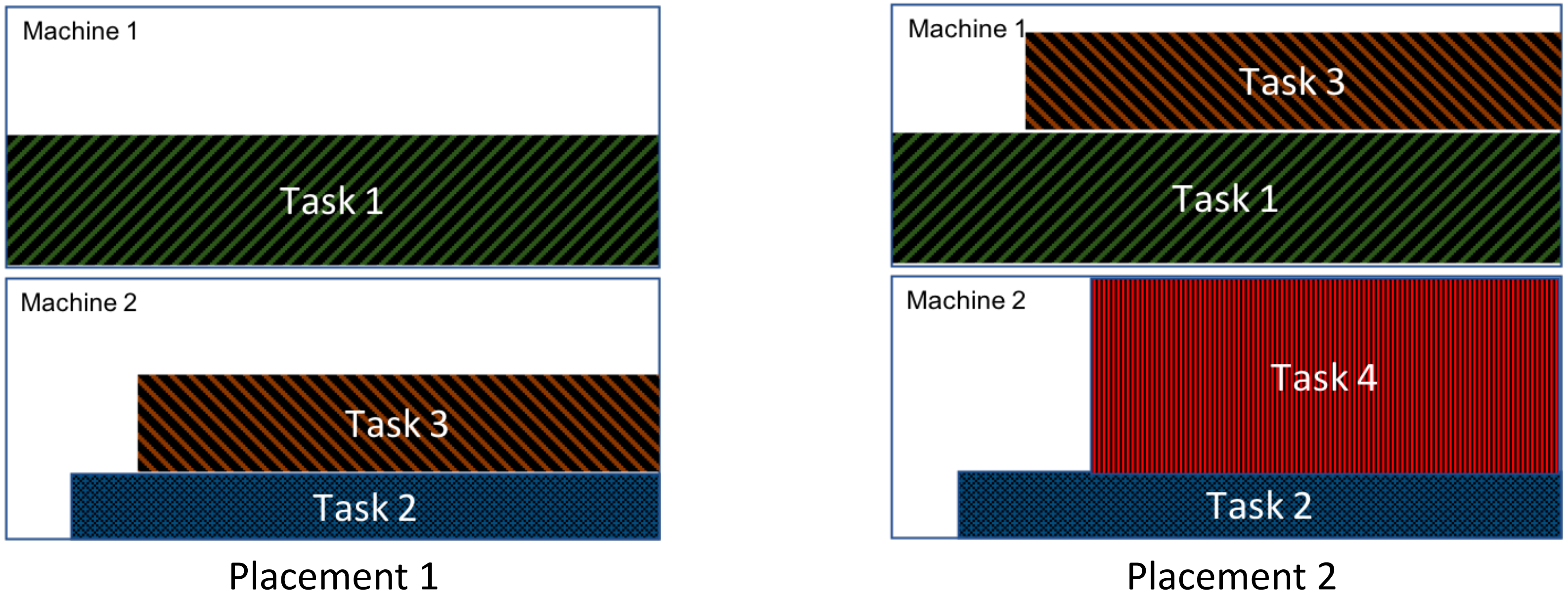
Job 1 Job 2

Where they fail – Job “Dependencies”



Where they fail – Fragmentation

- Given load be "Task 1 (0.5r), Task2 (0.25r), Task 3 (0.375r) and Task 4 (0.75r)"



- Placement 1 – Although same total resources available, but is fragmented.
- Placement 2 – Able to schedule all 4 tasks.

What should be done?

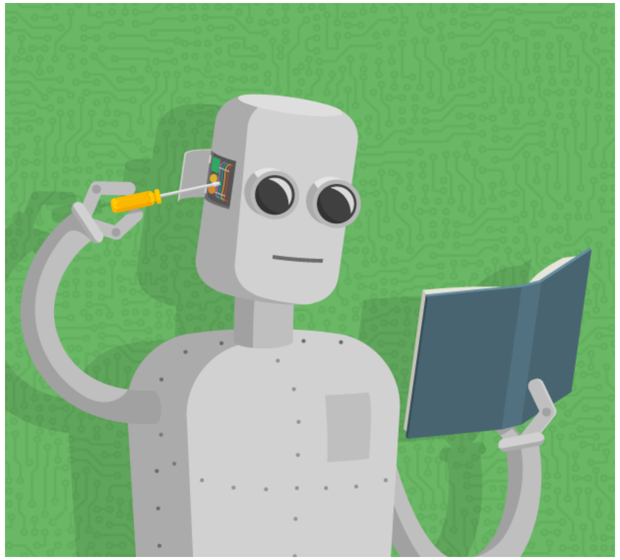
At a scheduling decision

- Analyze current state of all the machines.
- Decide on which machine to place the next application.
- Observe the benefits obtained from this decision.
- Improve our decisions based on these observations.

Formalizing It

Build a ***self-learning*** scheduler to ***opportunistically*** place containerized applications such that

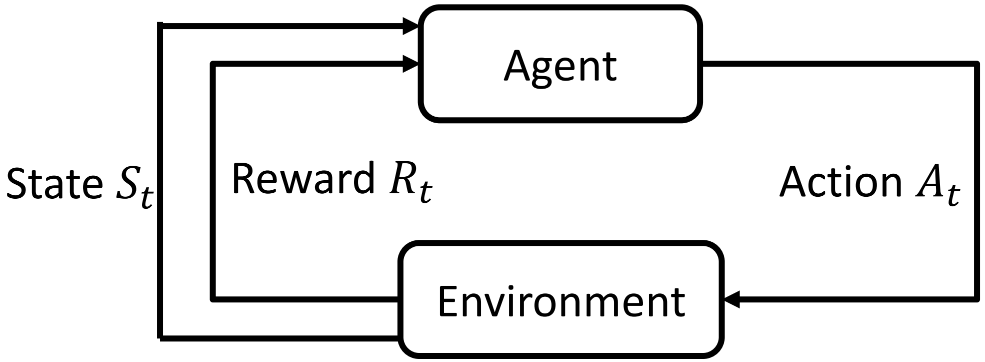
- **Temporal Usages** are aligned,
- **Resource Contentions** are minimized,
- **Quality of Service** is maintained and
- **Overall Resource Utilization** improved.



What should be done?

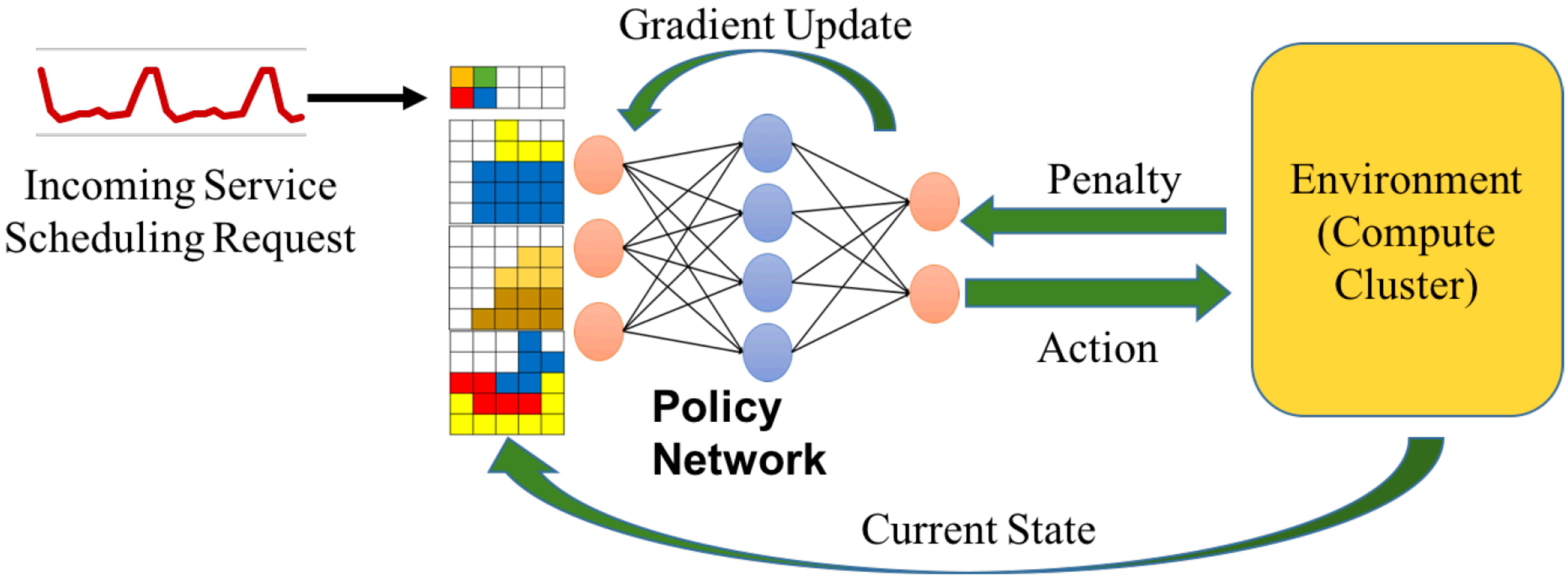
At a scheduling decision

- Analyze **current state** of all the machines – **State Representation (S_t)**
- Decide **on which machine** to place the next application – **Action Space (A_t)**
- Observe **the benefits obtained** from this decision – **Reward Function (R_t)**
- Improve **our decisions** based on these observations – **Policy Network (π)**

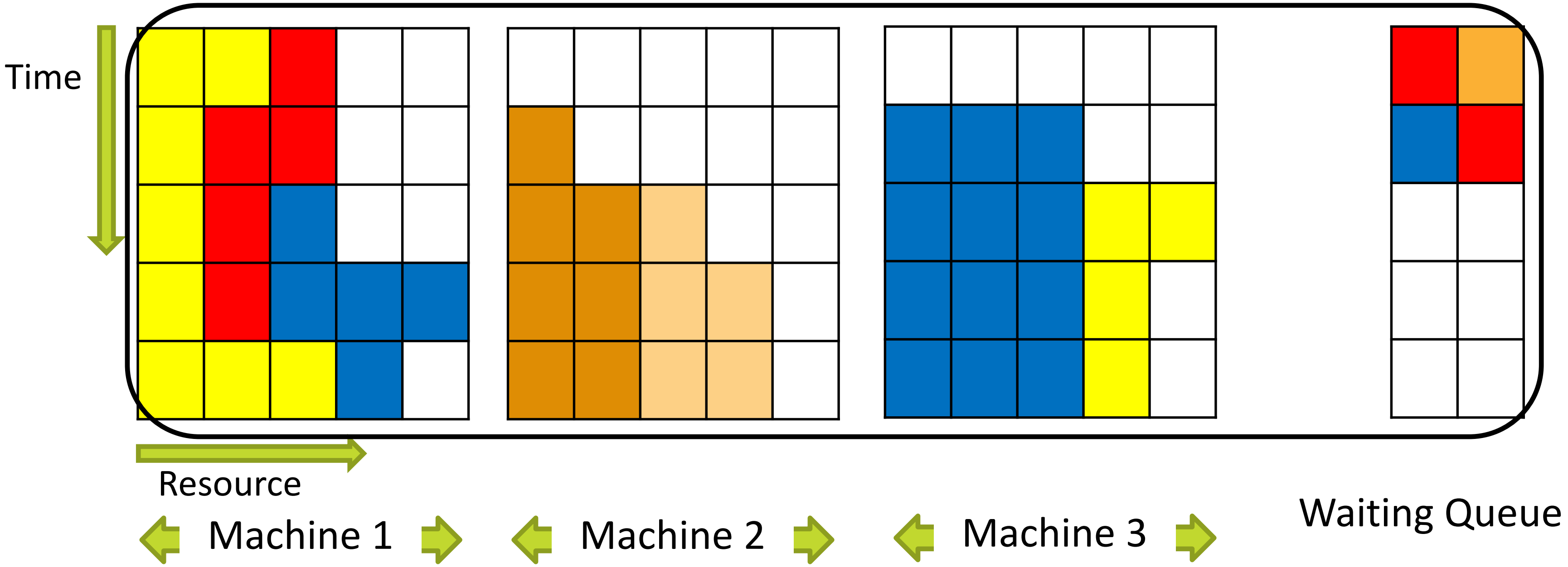


Leads to a natural map to a **Reinforcement Learning Problem**!

Solution – Workflow

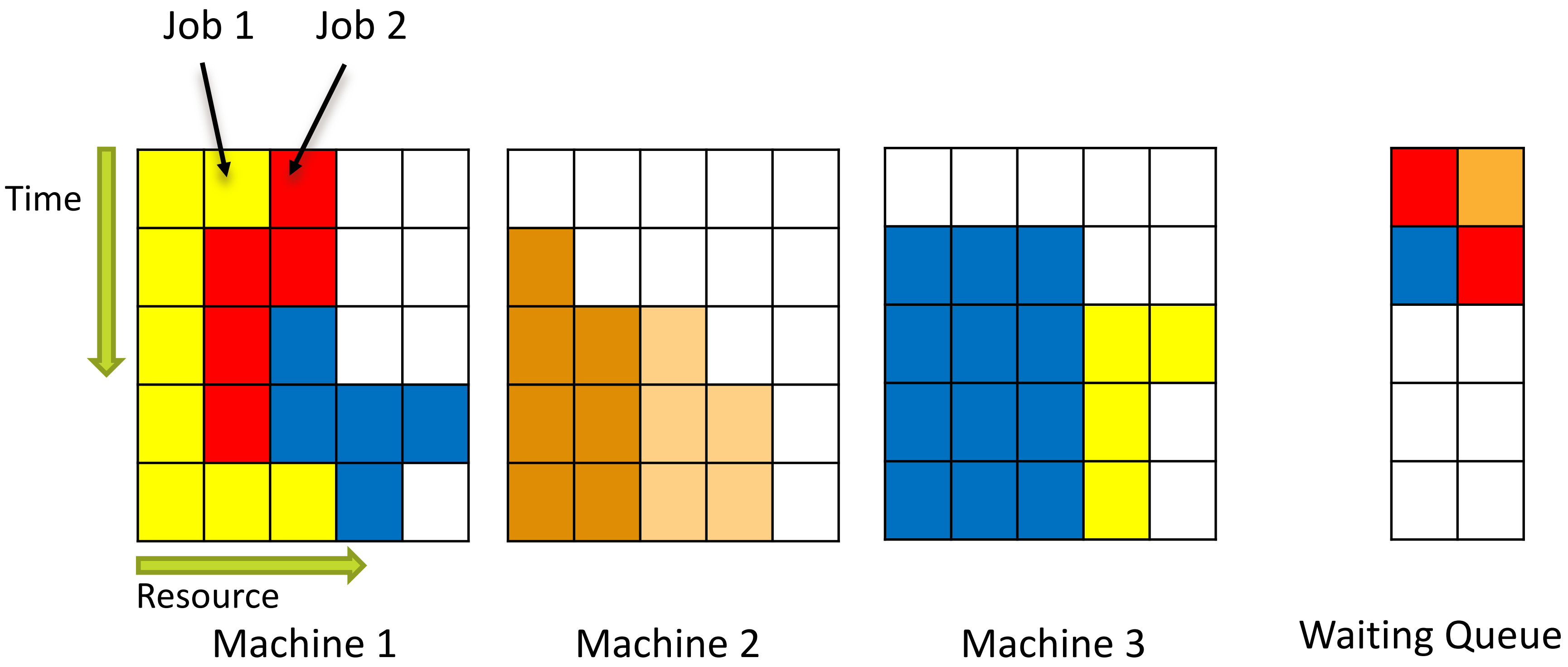


1. State Representation



* Inspired from DeepRM – HotNets'16

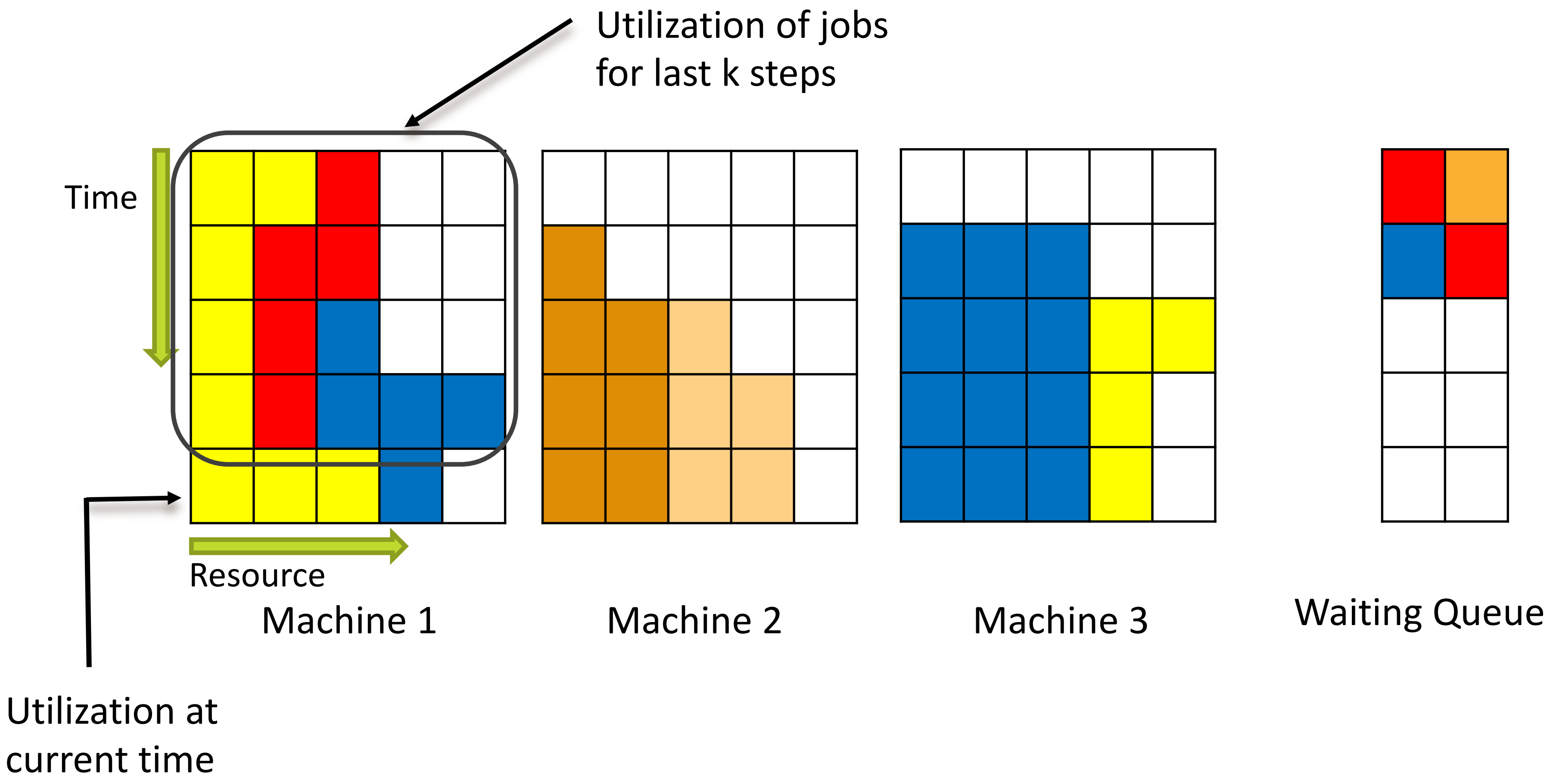
1. State Representation (2)



- **Different colors for different jobs**

* Inspired from DeepRM – HotNets'16

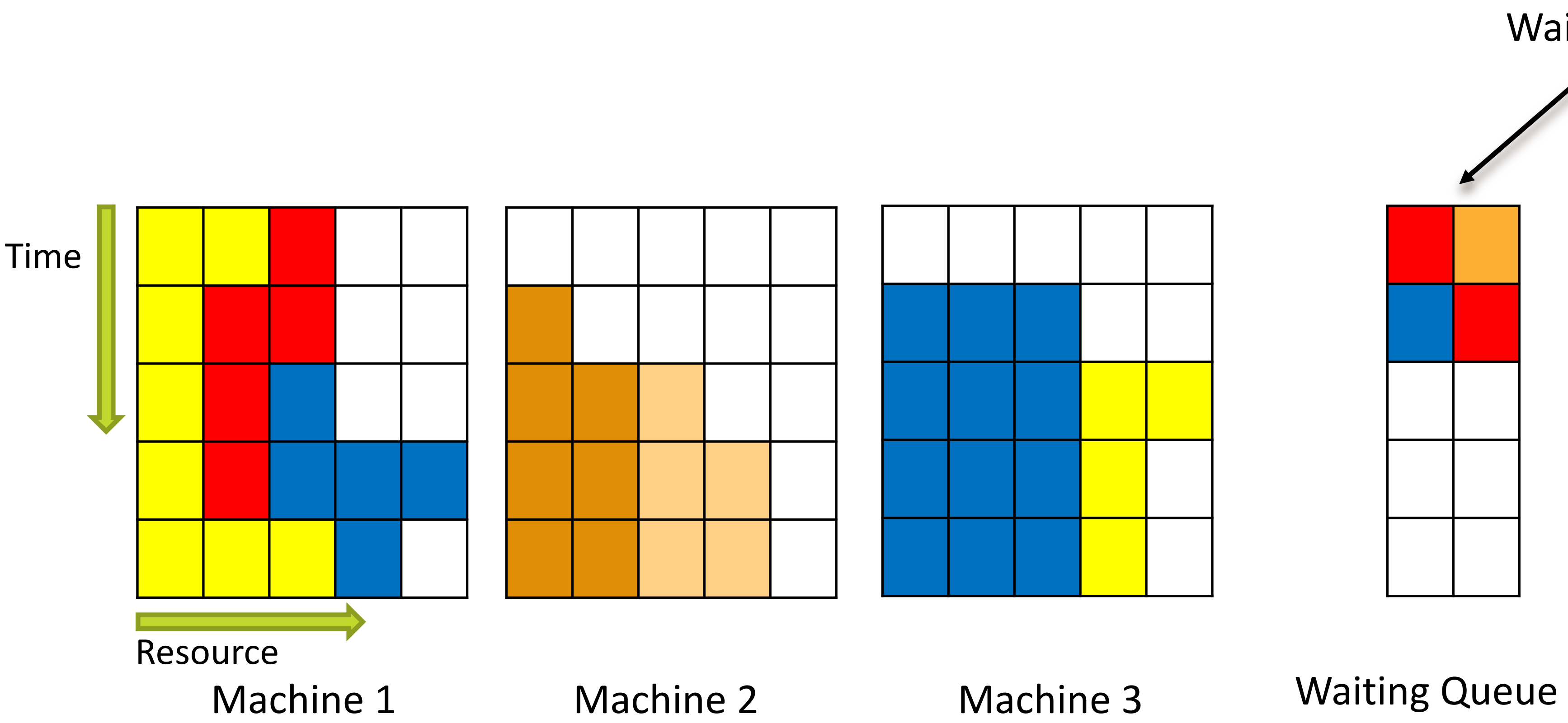
1. State Representation (3)



- Different colors for different jobs
- **Captures history of job profiles**

* Inspired from DeepRM – HotNets'16

1. State Representation (4)

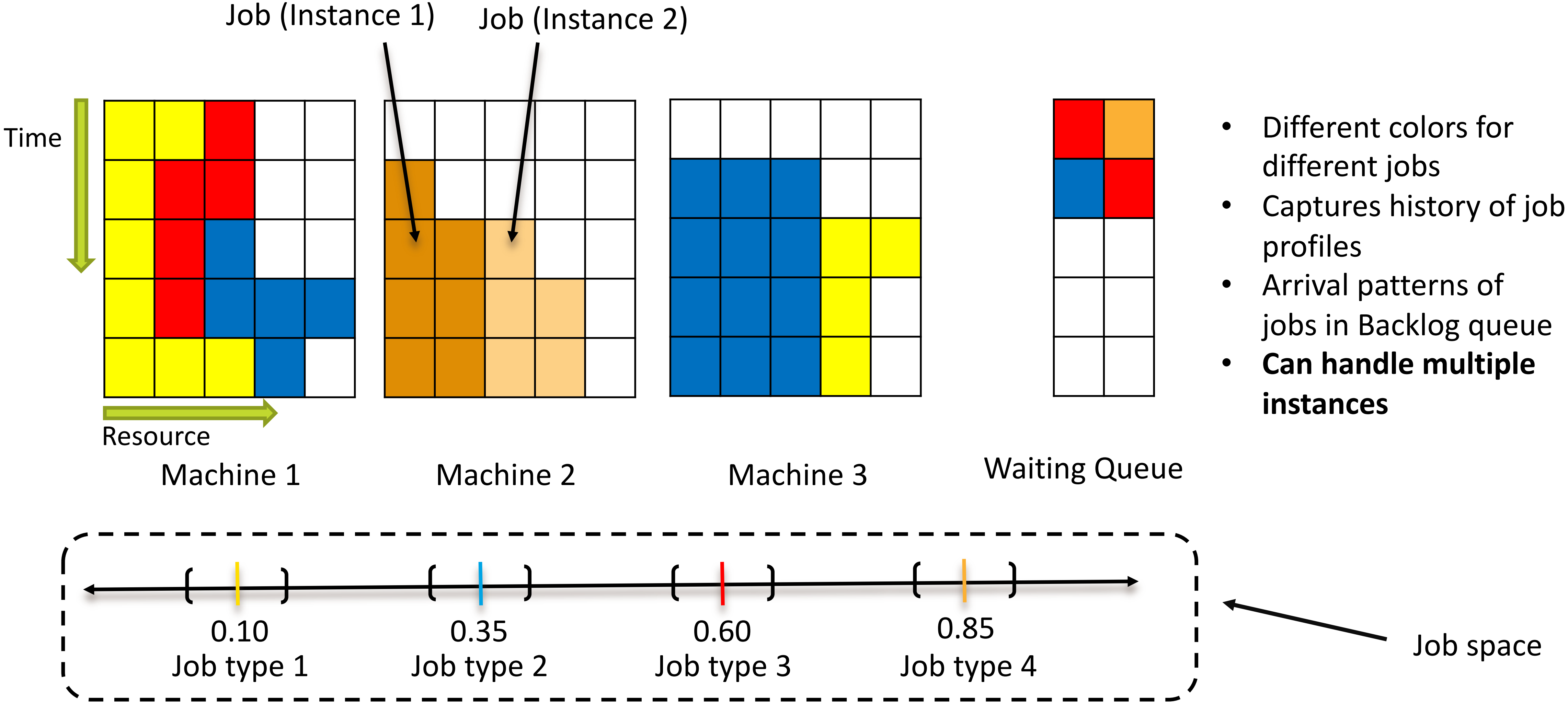


- Different colors for different jobs
- Captures history of job profiles
- **Arrival patterns of jobs in Backlog queue**

* Inspired from DeepRM – HotNets’16

Adobe

1. State Representation (5)



- Different colors for different jobs
- Captures history of job profiles
- Arrival patterns of jobs in Backlog queue
- **Can handle multiple instances**

2. Action Space

- $A = \{0,1,2,\ldots,M\}$ i.e. {0 ∪ Set of machines}
- A_t = 0 means choosing to **not schedule the job**.
  - Knowingly delay.
  - Probably better alignment later.
- At a given timestep, multiple actions can be taken.
  - On the set of jobs in the queue.

3. Reward Function - Art of Penalizing

- **Resource Contention Penalty**
  - **Prevent resource contention among tasks scheduled in the same machine.**
- Resource Over-Utilization Penalty
  - Prevent scheduling of more tasks than can be handled.
- Wait-Time Penalty
  - Prevent the scheduler from holding jobs for a long time.
- Under-Utilization Penalty
  - Improve overall utilization by achieving tighter packing.

$$\mathrm{Cr}(i, j, d) = \sum_{\mathrm{t=0}}^{\mathrm{min(Ti,Tj)}} \mathrm{res\_usage}(i, t, d) \times \mathrm{res\_usage}(j, t, d)$$

3. Reward Function - Art of Penalizing (2)

- Resource Contention Penalty
  - Prevent resource contention among tasks scheduled in the same machine.
- **Resource Over-Utilization Penalty**
  - **Prevent scheduling of more tasks than can be handled.**
- Wait-Time Penalty
  - Prevent the scheduler from holding jobs for a long time.
- Under-Utilization Penalty
  - Improve overall utilization by achieving tighter packing.

$$\mathrm{Cr}(i, j, d) = \sum_{\mathrm{t=0}}^{\mathrm{min(Ti,Tj)}} \mathrm{res\_usage}(i, t, d) \times \mathrm{res\_usage}(j, t, d)$$

$$P_O = -\sum_{m=1}^{M} K\,[Resources\ overshooted\ for\ m]$$

3. Reward Function - Art of Penalizing (3)

- Resource Contention Penalty
  - Prevent resource contention among tasks scheduled in the same machine.
- Resource Over-Utilization Penalty
  - Prevent scheduling of more tasks than can be handled.
- **Wait-Time Penalty**
  - **Prevent the scheduler from holding jobs for a long time.**
- Under-Utilization Penalty
  - Improve overall utilization by achieving tighter packing.

$$\mathrm{Cr}(i, j, d) = \sum_{\mathrm{t=0}}^{\mathrm{min(Ti,Tj)}} \mathrm{res\_usage}(i, t, d) \times \mathrm{res\_usage}(j, t, d)$$

$$P_O = -\sum_{m=1}^{M} K\,[Resources\ overshooted\ for\ m]$$

$$P_w = -W * |Job\ Queue|$$

3. Reward Function - Art of Penalizing (4)

- Resource Contention Penalty
  - Prevent resource contention among tasks scheduled in the same machine.

$$\mathrm{Cr}(i, j, d) = \sum_{\mathrm{t=0}}^{\mathrm{min(Ti,Tj)}} \mathrm{res\_usage}(i, t, d) \times \mathrm{res\_usage}(j, t, d)$$

- Resource Over-Utilization Penalty
  - Prevent scheduling of more tasks than can be handled.

$$P_O = -\sum_{m=1}^{M} K\,[Resources\ overshooted\ for\ m]$$

- Wait-Time Penalty
  - Prevent the scheduler from holding jobs for a long time.

$$P_w = -W * |Job\ Queue|$$

- **Under-Utilization Penalty**
  - **Improve overall utilization by achieving tighter packing.**

$$P_U = \sum_{m \in \mathrm{used\ VMs}} \#(unused\ resources_j)$$

4. Policy Network

- A Deep Neural Network
- Output – Probability distribution over Action Space.
- Learning – REINFORCE Algorithm.
- Multiple workers on different examples to accumulate gradients.
  - One worker – Combines the gradients.

Evaluations – Baselines

DeepRM – RL Agent

- Identifies job to be scheduled next.
- RL agent - learns policy to optimize the defined reward.
- Treats cluster as monolithic.
- Doesn't specify where to schedule.
- **Fair comparison not possible.**
- [DeepRM - HotNets'16]

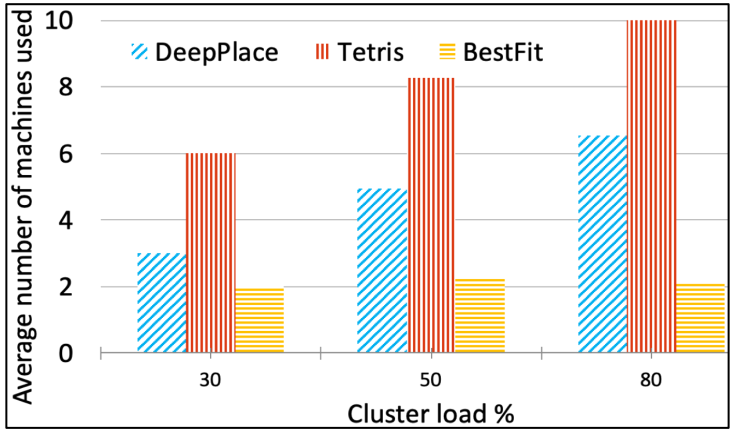
Tetris – Heuristic Based

- Schedules jobs on machines.
- How well resource requirement aligns with the machine's available resources.
- Adapts heuristics from multi-dimensional bin packing.
- [Tetris – SIGCOMM'14]

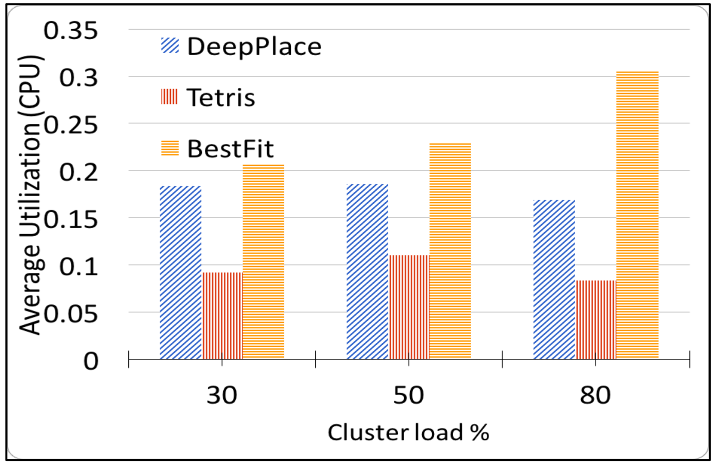
BestFit – Heuristic Based

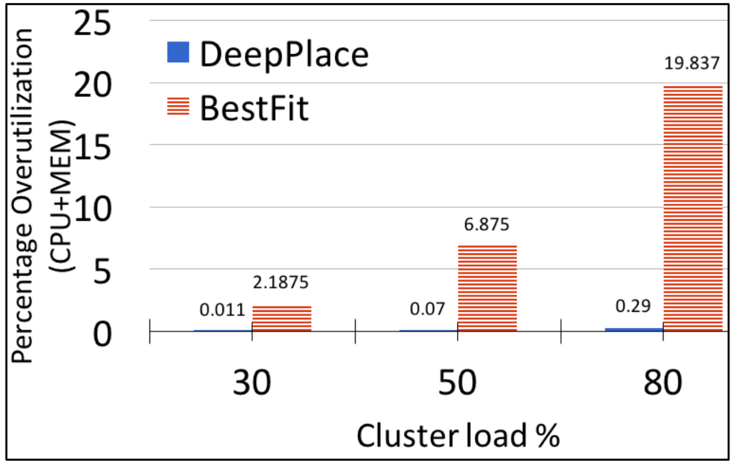
- Schedules jobs on machines.
- Chooses the machine which has the least units of the task's dominant resource available.
- Heuristic closest to packing.

Evaluations – Scheduling Efficiency

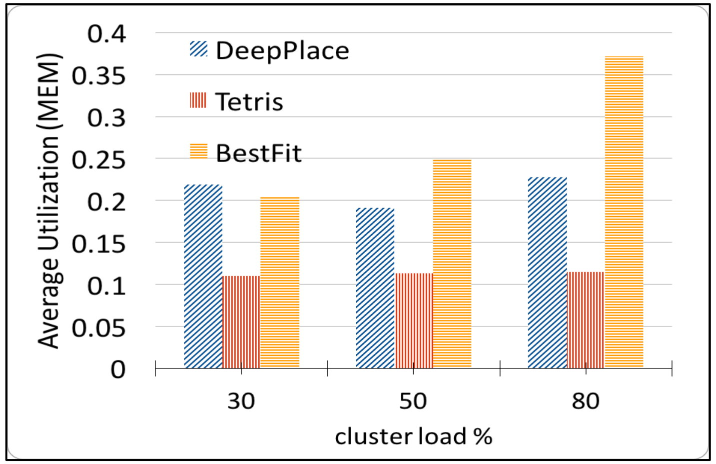


Comparison of Number of machines used in the cluster





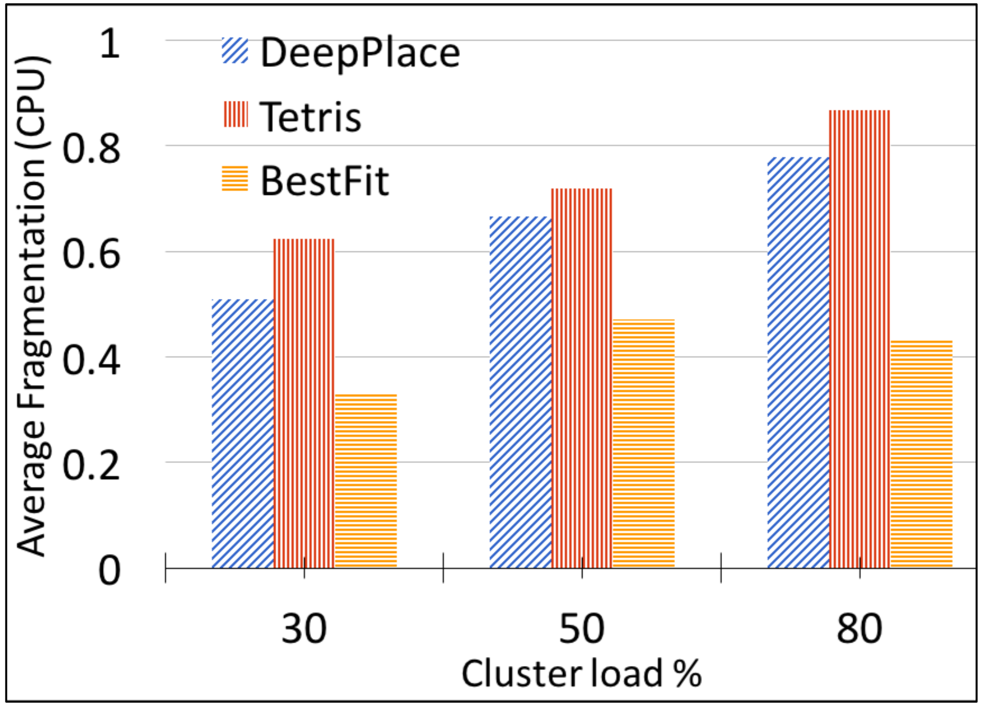
Comparison of Resource Over-Utilization

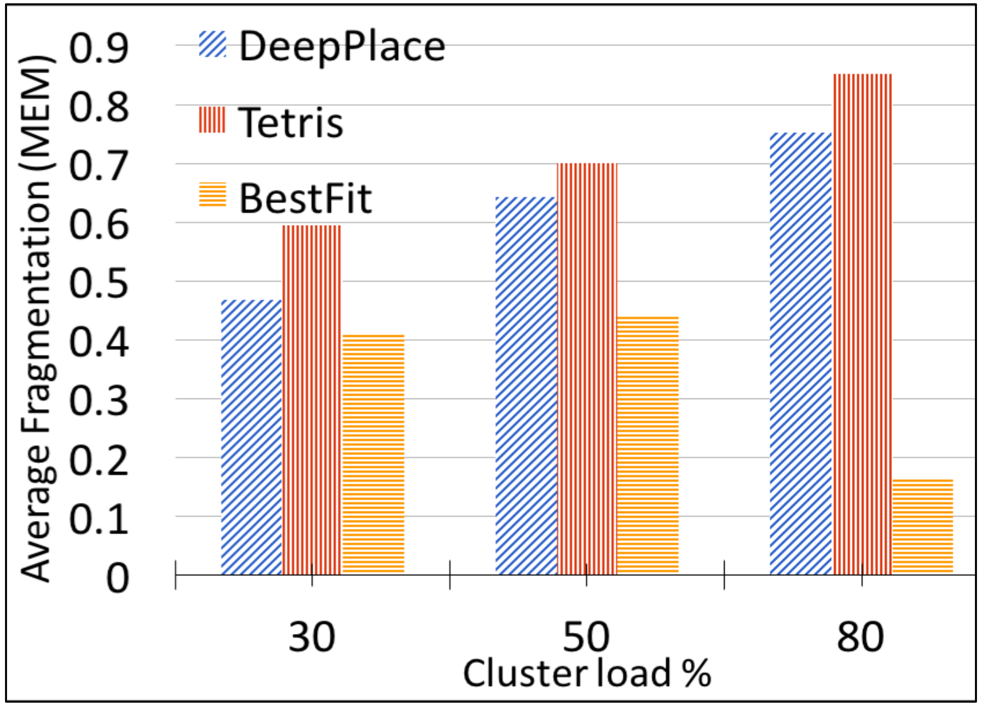


Comparison of Average Resource Utilization in the cluster (CPU and MEM)

Evaluations – Fragmentation

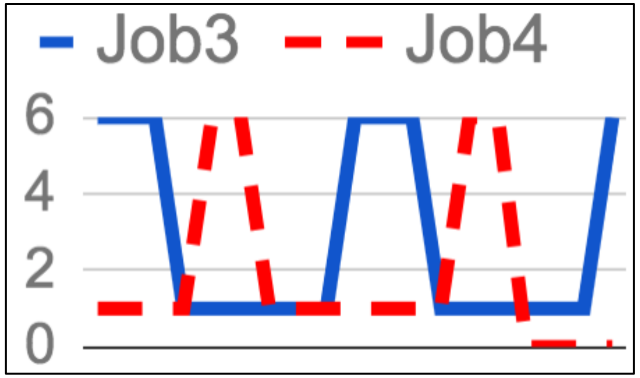
- We define Average Fragmentation: $$\text{Avg Frag} = 1 - \sum_{t=0}^{T} \frac{\max(\text{available space across all machines at t})}{\text{Sum of available space over all machines at t}}$$





Discussions

- What It Learned?
  - Learned patterns among job’s Resource Usages.
  - Ex: Job finishing before peak of other job.
  - Ex: Jobs’ with alternating peaks.



Discussions - Deployments

- Scheduling Granularity for effectiveness.
  - Decision Process – Frequent or not?
  - Job Length – Allows for pattern discovery?

- Boot-strapping Learning.
  - Avoid learning from scratch.
  - Use replays of historical time-series.

SPIRAL

Google DeepMind

Future Work

- Cluster Size Dependency.
  - Input space representation is function of cluster size.
  - Policy learning takes more time to train and converge.

- Evaluation on Real-Life Workloads.
  - Current experimentation on synthetic workloads.
  - Real-life workloads have noisier time-series.

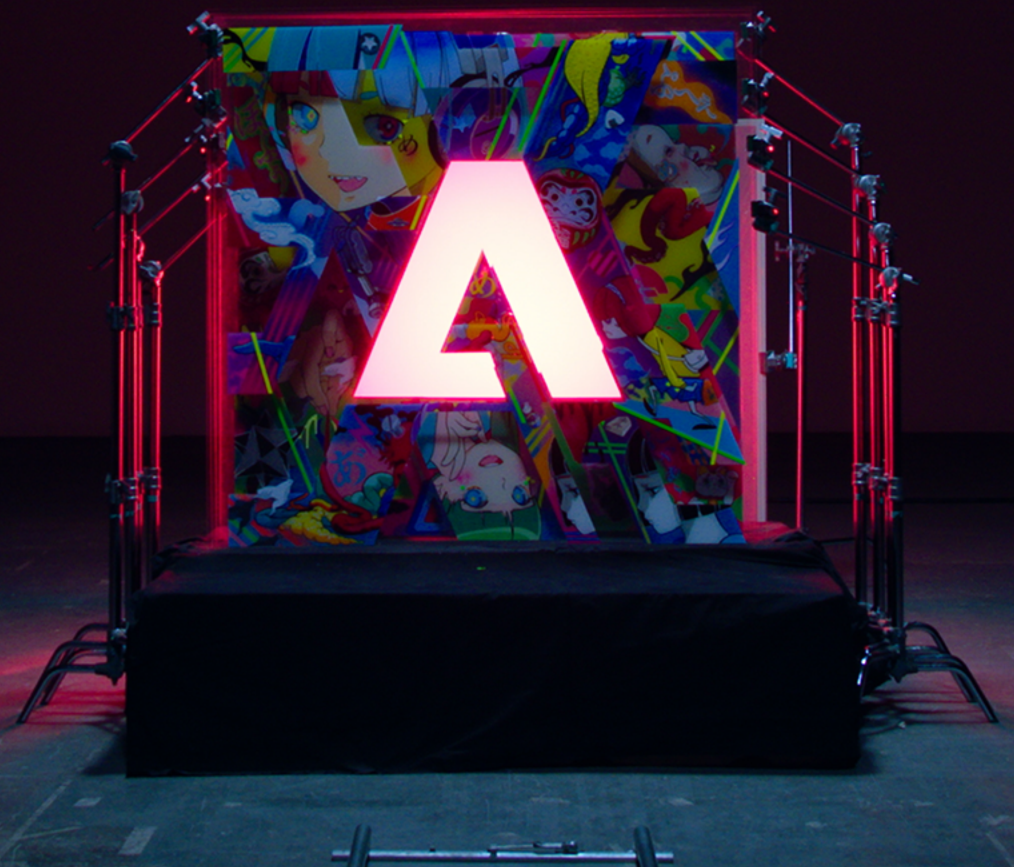
Conclusion

- Current Multi-Tenant Clusters need to handle **variety of services** with different type of user expectations and characteristics at production.

- **Not possible** to design **hand-crafted heuristics** to orchestrate these services due to numerous latent factors.

- Our self-learning scheduler, **DeepPlace**, based on Reinforcement Learning shows promise and improvements than heuristics based approaches.

Thank You! Any Questions?



#AdobeRemix
Hiroyuki-Mitsume Takahashi