

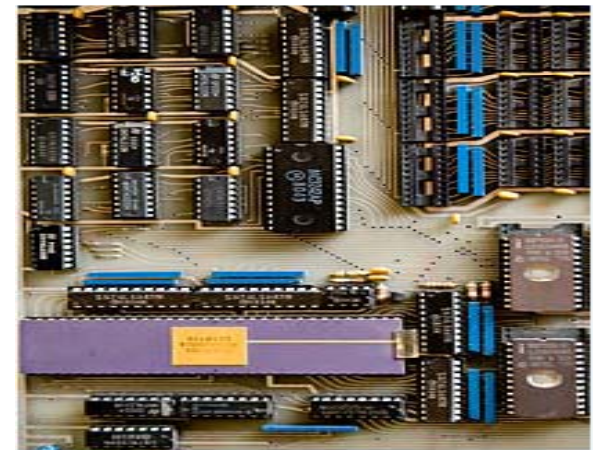
# Reinforcement Learning for Operating Systems Scheduling

Iftekhhar Naim

# Introduction & Motivation

---

- ▶ Resource scheduling: critical component of today's operating systems
  - ▶ CPU scheduling in heterogeneous multicore systems
    - ▶ Multiple cores inside each processor
    - ▶ Cores may have different specifications
  - ▶ Resource management in massive data centers, etc.
    - ▶ Resource (CPU, memory) sharing among multiple jobs/partitions
  - ▶ Effective memory and I/O scheduling
    - ▶ Off-chip bandwidth utilization



# Limitations of Traditional Schedulers

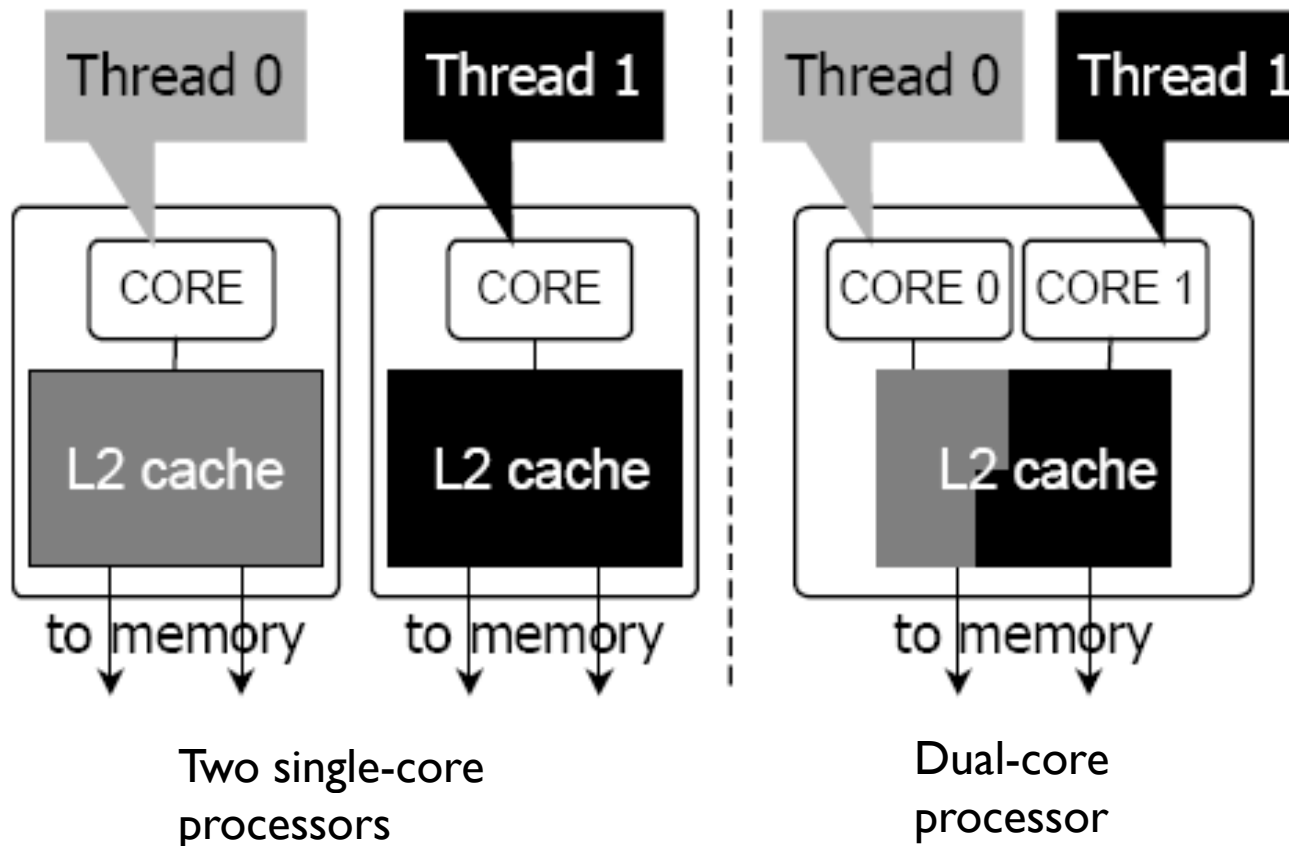
---

- ▶ Resource scheduling in modern multi-core systems have become complicated
  - ▶ Unequal cache sharing (Fedorova et al. , 2006 [1])
    - ▶ One thread's performance depends on the cache behavior of its co-runners
  - ▶ Poor priority enforcement
    - ▶ If high priority job is scheduled with 'bad' co-runners
  - ▶ Unbalanced core assignment (in heterogeneous case)
    - ▶ Ideally, a job's execution time should be spread across all cores evenly

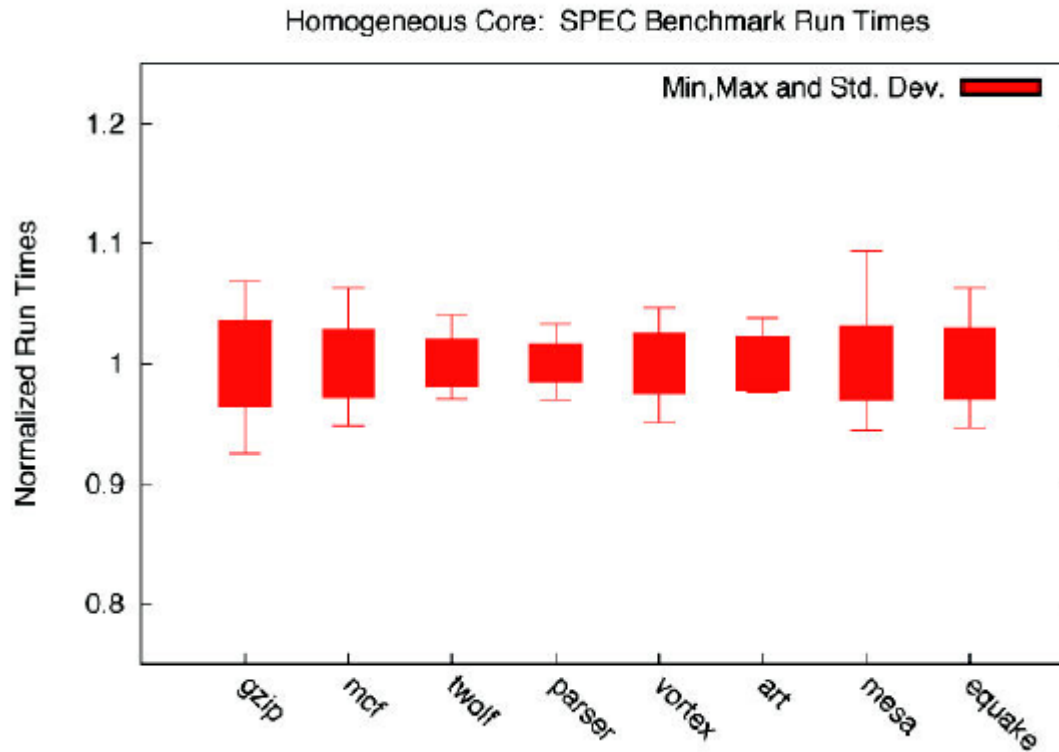
# Limitations of Traditional Schedulers

---

- ▶ Resource contention due to shared L2 cache



# Homogeneous vs Heterogeneous Multicore Scheduling-1 (Fedorova et al., 2007)

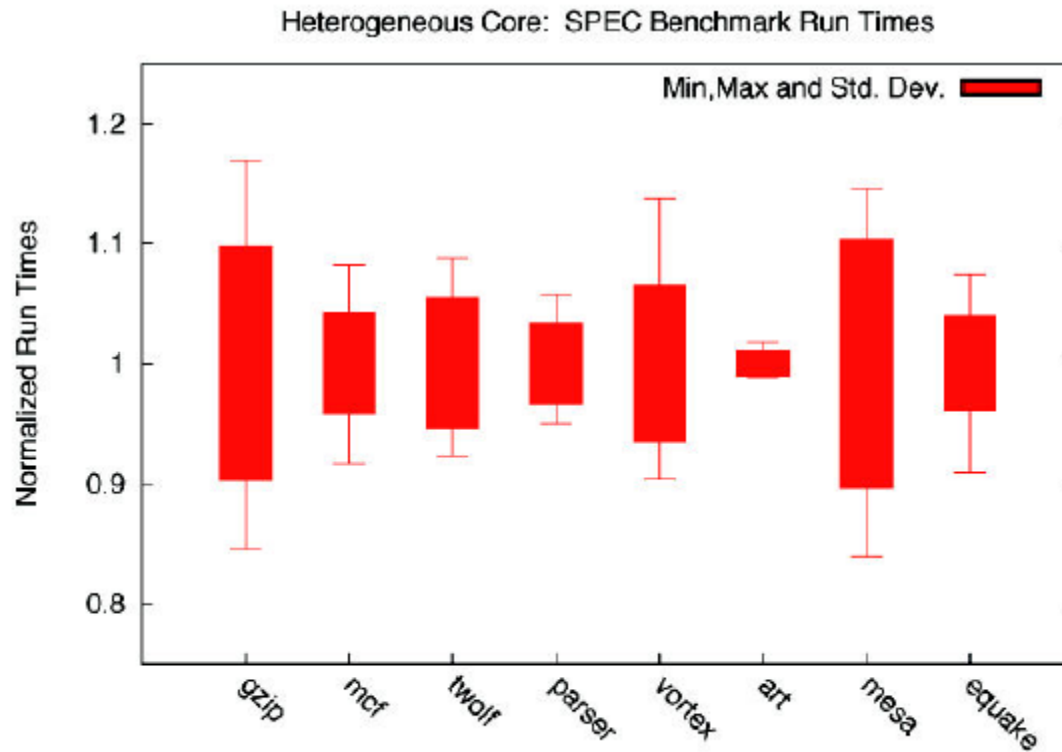


**Figure 1.** SPEC benchmark runtimes in the homogeneous setup.

- ▶ Two homogeneous cores (2.8 GHz each)

# Homogeneous vs Heterogeneous Multicore Scheduling-2 (Fedorova et al., 2007)

---



**Figure 2.** SPEC benchmark runtimes in the heterogeneous setup.

- ▶ Two heterogeneous cores (2.8 GHz, and 1.05 GHz)

# Limitations of Traditional Schedulers

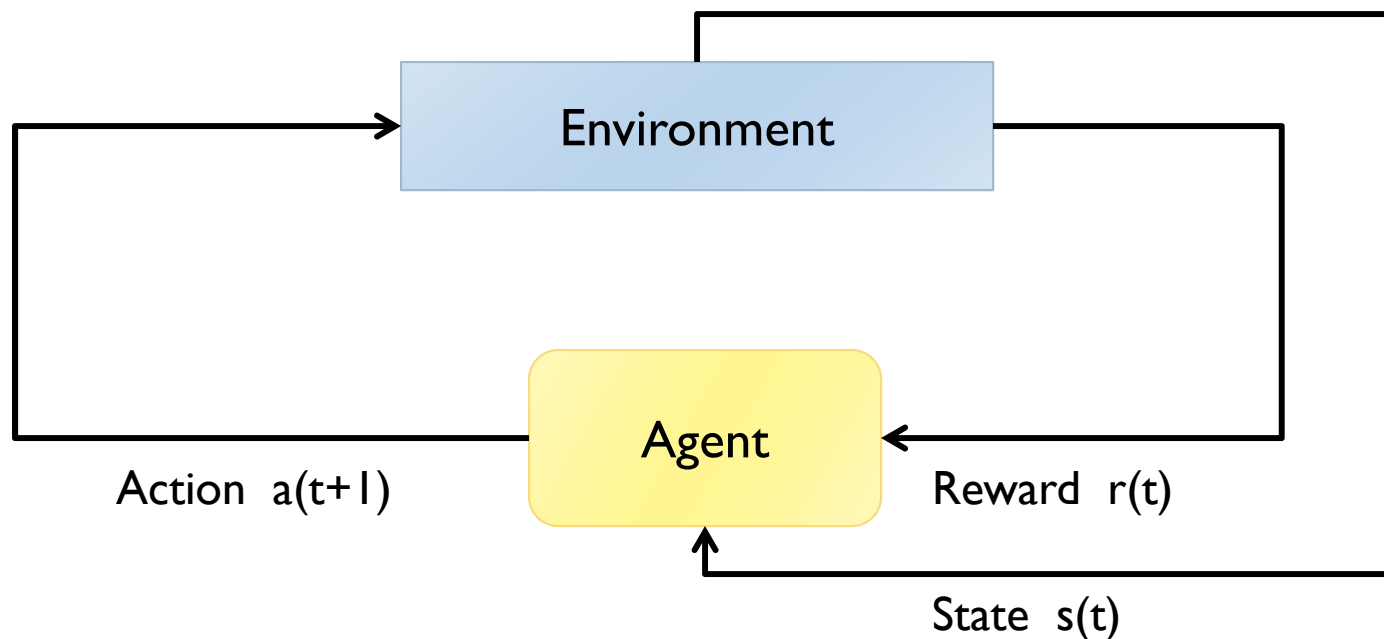
---

- ▶ **Traditional schedulers employ a fixed policy**
  - ▶ Often chosen in ad hoc manner by human experts
- ▶ **Major limitations:**
  - ▶ Fixed policy schedulers can not adapt to varying environment
  - ▶ Can not anticipate long term consequences of decisions
  - ▶ Incapable of learning from past experiences
- ▶ **Solution:**
  - ▶ Self-tuning/self-optimizing scheduler
    - ▶ One possibility: Reinforcement Learning (RL) based scheduler

# Reinforcement Learning Overview

---

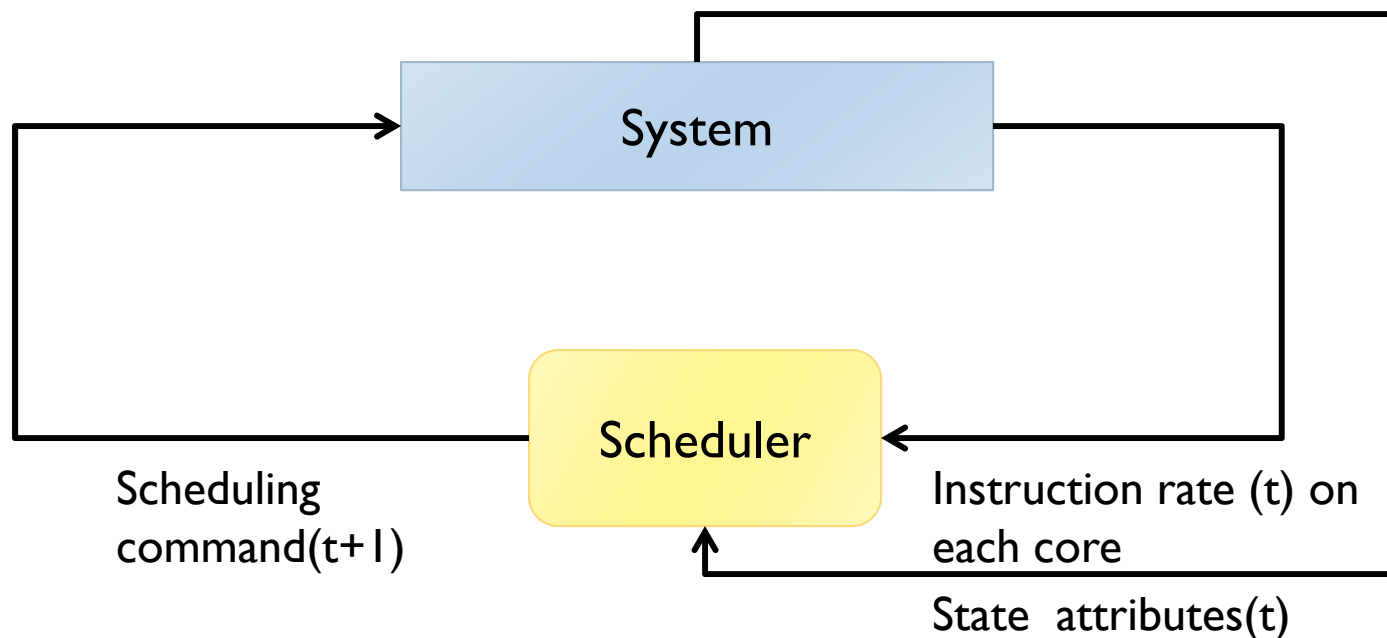
- ▶ **Reinforcement Learning:** learning how to map actions to situations
  - ▶ Learning while interacting with the environment
  - ▶ Maximizing the long term cumulative reward signal



# Reinforcement Learning Overview

---

- ▶ **Reinforcement Learning:** learning how to map actions to situations
  - ▶ Learning while interacting with the environment
  - ▶ Maximizing the long term cumulative reward signal



# Markov Decision Process (MDP)

---

- ▶ Reinforcement learning problems are typically modeled as a Markov Decision Process (MDP)
- ▶ An MDP consists of five key elements:  $(S, A, P_{sa}, \gamma, r)$ 
  - ▶  $S$ : set of states in the system
  - ▶  $A$ : set of actions
  - ▶  $\{P_{sa}\}$ : state transition probabilities
  - ▶  $\gamma$ : discount factor, a real value such that  $0 < \gamma < 1$ 
    - ▶ Future rewards are less valuable
  - ▶  $r(s)$ : the reward function for state  $s$ .
    - ▶ Short term benefit

# MDP: Value function and Policy

---

▶ **Policy**  $\pi : S \rightarrow A$

▶ a mapping from each state  $s \in S$  to an action  $a \in A$ .

▶ **State Value function:**  $B^\pi(s) : S \rightarrow \mathfrak{R}$

▶ Expected cumulative reward achieved given the starting state  $s$ , and policy  $\pi$

$$B^\pi(s) = E[r(s_t) + \gamma r(s_{t+1}) + \gamma^2 r(s_{t+2}) + \dots \mid s_t = s]$$

▶ **Goal:**

▶ Determine the optimal policy for each state

$$\pi^*(s) = \arg \max_{\pi} B^\pi(s)$$

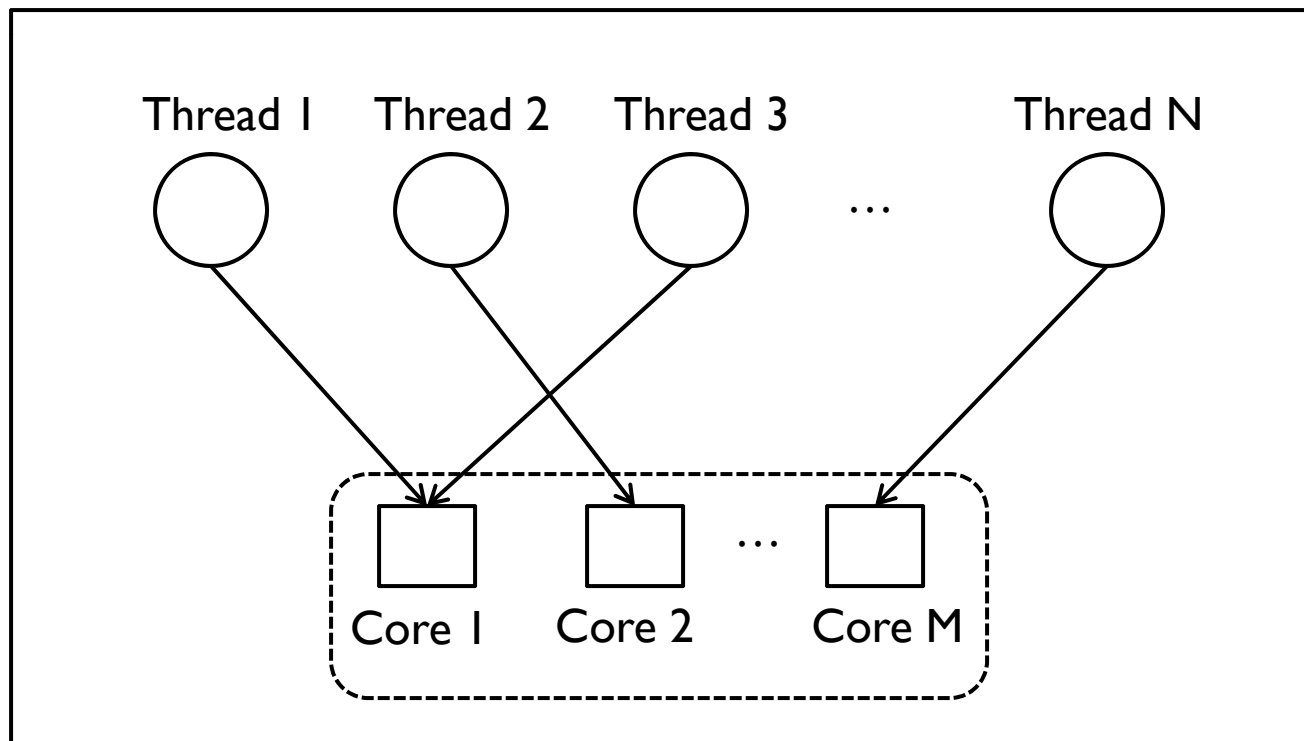
# RL-based Scheduler

---

- ▶ We summarize two papers for resource scheduling
- ▶ **Thread scheduler on multi-core systems:**
  - ▶ Assign threads to different CPU cores (Fedorova et al., 2007 [2])
- ▶ **Data Center Resource Schedulers:**
  - ▶ Assign memory and CPU units to different partitions in a data center (Vengerov, 2007 [3])

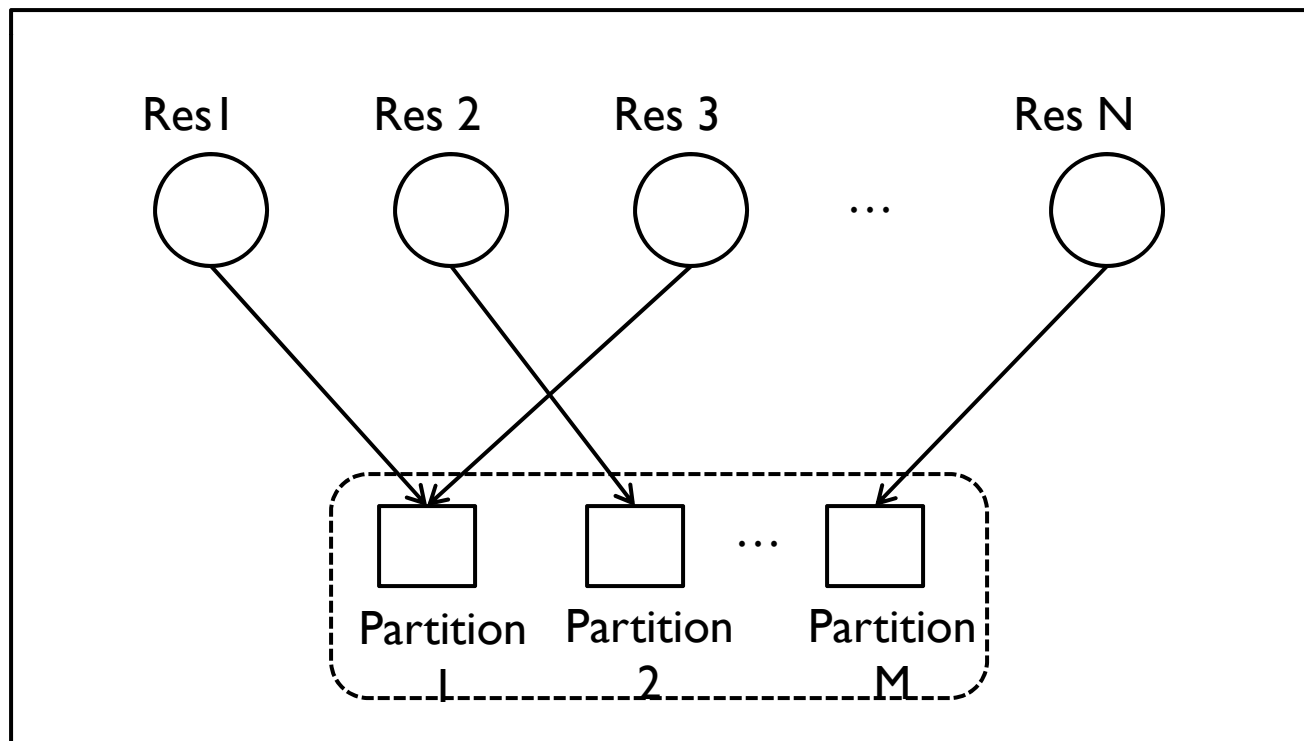
# RL-based Scheduler (Example-1)

- ▶ Thread scheduler in a heterogeneous multi-core system (Fedorova et al., 2007 [2])
  - ▶ Assigns threads to CPU cores



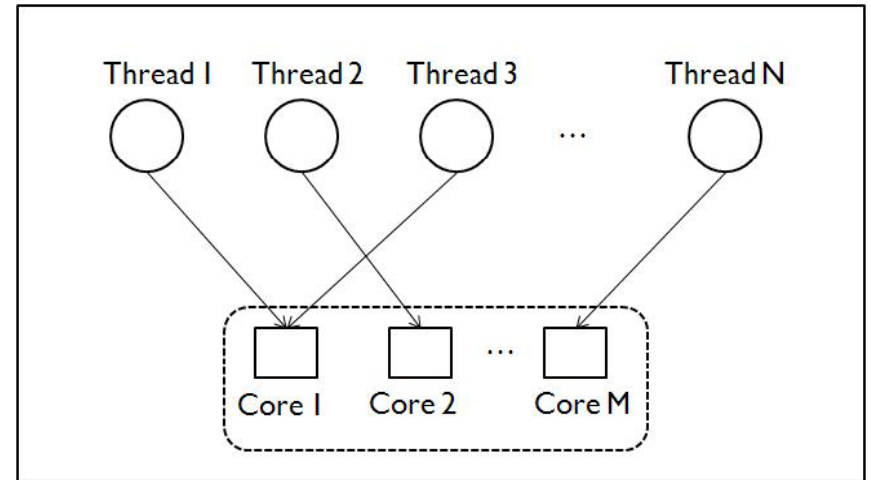
## RL-based Scheduler (Example-2)

- ▶ Dynamic resource scheduler in massive data centers (Vengerov, 2007 [3])
  - ▶ Assigns resources (CPU , memory units) to partitions



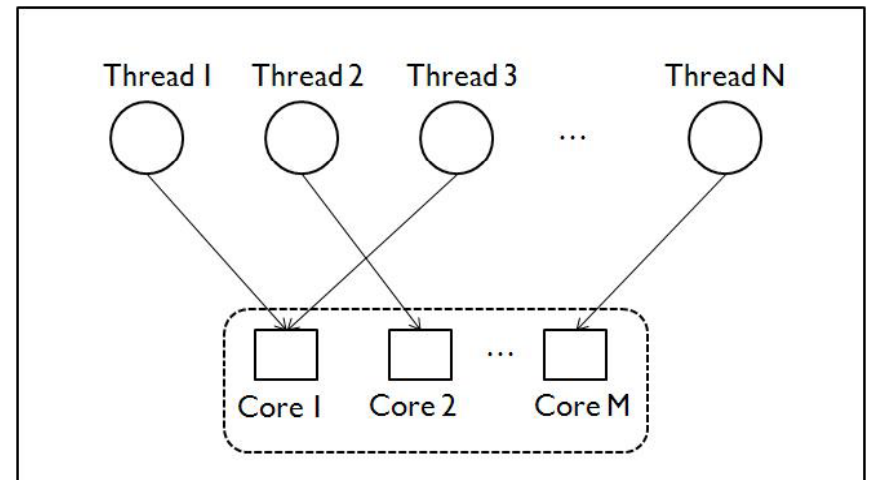
# MDP Formulation for Thread Scheduler (States)

- ▶ Determined by state variables/attributes
- ▶ Maintain states per core
- ▶ **State variables (on core i):**
  - ▶ Average normalized instructions per cycle
  - ▶ Average cache affinity
  - ▶ Average cache miss rate



# MDP Formulation for Thread Scheduler (Actions)

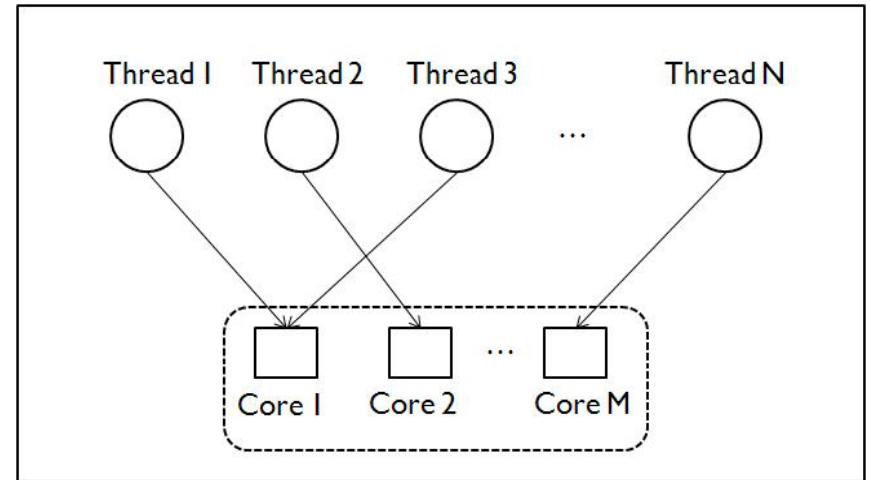
- ▶ **Actions:**
  - ▶ Migrate  $k^{\text{th}}$  thread from **core  $i$**  to **core  $j$**
- ▶  **$O(MN)$  possible actions**
  - ▶  $M$  = number of cores
  - ▶  $N$  = number of threads



# MDP Formulation for Thread Scheduler

## (Reward Function)

- ▶ Reward function is defined per state, per core
- ▶  $R_i(s_t)$  is the reward at state  $s_t$  in core  $i$ .
- ▶  $R_i(s_t) =$  Instruction rate in the  $i^{\text{th}}$  core, in state  $s_t$  during the interval  $(t, t+1)$



# Decision Making in RL-based Scheduler

---

- ▶ **Decision making in RL Agents:**

- ▶ Estimate the long term rewards for each (state, action)
- ▶ Take the decision to optimize the long term reward

- ▶ **Long Term Reward (State Value Function):**

- ▶ Expected cumulative reward achieved given the starting state  $s$ , and policy  $\pi$

$$B^\pi(s) = E[r(s_t) + \gamma r(s_{t+1}) + \gamma^2 r(s_{t+2}) + \dots \mid s_t = s]$$

- ▶ **Optimum Decision:**

- ▶ Take the decision that optimizes long term rewards

$$\pi^*(s) = \arg \max_{\pi} B^\pi(s)$$

# Decision Making in RL-based Scheduler

---

## ▶ Thread Scheduler:

- ▶ Predict the expected long term reward value for each thread migration for each core
- ▶ For each pair of cores (i,j) and each thread k:
  - ▶  $B_{i,k-}(s_t)$  = expected cumulative reward of migrating thread k away from core i
  - ▶  $B_{i,0}(s_t)$  = expected cumulative reward if no migrations from core i
  - ▶  $B_{j,k+}(s'_t)$  = expected cumulative reward of migrating thread k to core j
  - ▶  $B_{j,0}(s'_t)$  = expected cumulative reward if no migrations to core j

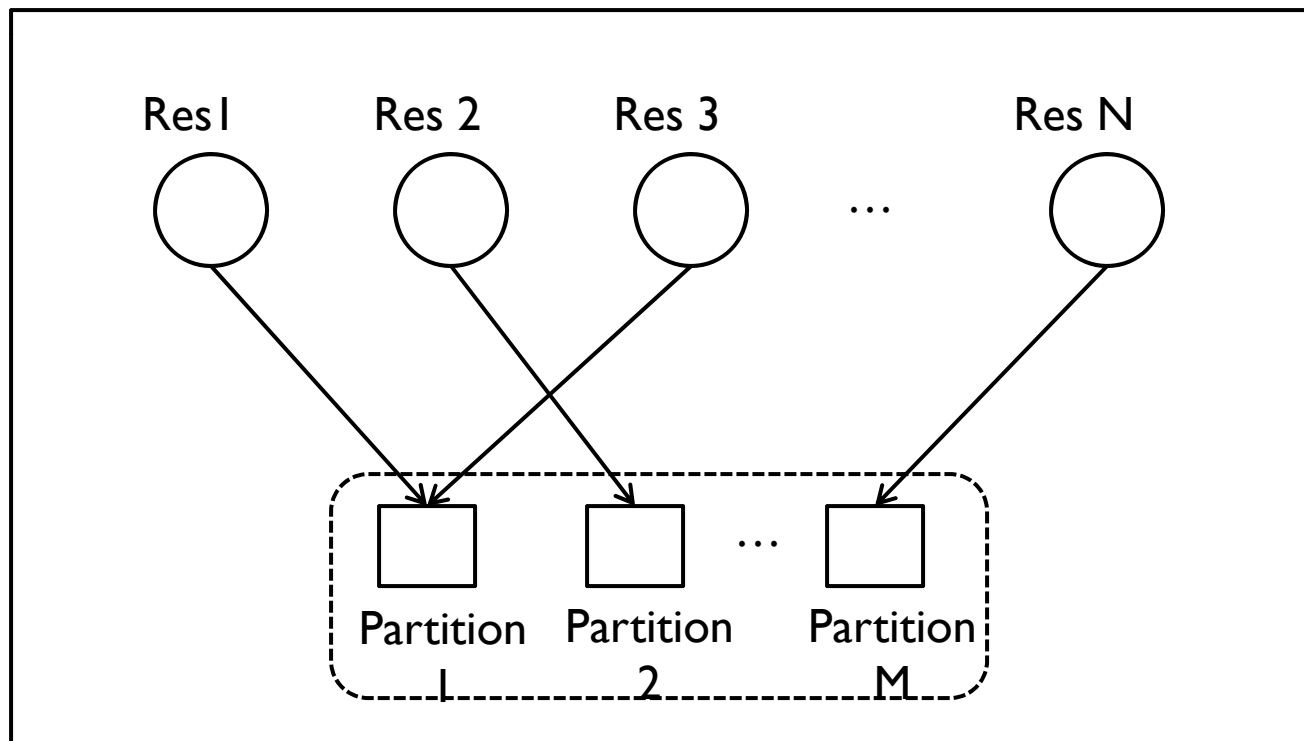
# Decision Making in RL-based Scheduler

---

- ▶ **Migration Decision:** migrate thread  $k$  from core  $i$  to core  $j$  if:
  - ▶  $B_{i,k-} + B_{j,k+} > B_{i,0} + B_{j,0} + a \cdot D_{CAB} + b \cdot D_{RTF}$
- ▶ **CAB = Core assignment balance**
  - ▶ The percentage of jobs satisfying balanced core assignment constraint
  - ▶  $D_{CAB}$  = expected change in CAB
- ▶ **RTF = Response time fairness**
  - ▶ The percentage of jobs satisfying response time fairness constraints
  - ▶  $D_{RTF}$  = expected change in RTF

# Experimental Results

- ▶ Dynamic resource scheduler in massive data centers (Vengerov, 2007)
  - ▶ Assigns resources (CPU , memory units) to partitions



# Experimental Results

---

- ▶ **Allocating a pool of 19 CPUs and 22GB of RAM**
  - ▶ Among 3 partitions
  - ▶ Sun Fire <sup>TM</sup> 2900 machine
- ▶ **Action:**
  - ▶ Consider migrating one CPU or one GB RAM among partitions at a time
- ▶ **State Attributes (per partition):**
  - ▶  $X_{i1}$  = CPU utilization in partition i
  - ▶  $X_{i2}$  = Memory utilization in partition i
- ▶ **Reward Function:**
  - ▶  $R_i(s_t) = - [ K * B_i(t+1) + N_{CPU}(t) C_{CPU}(t) + N_{MEM}(t) C_{MEM}(t) ]$
  - ▶  $B_i(t+1)$  = backlog experienced in partition i at time (t+1)
  - ▶  $K$  = a constant, empirically set to 5

# Comparison of DRA-FRL against two other policies

---

- ▶ **Static optimal policy**
  - ▶ Obtained by solving a queuing model for expected average queue length in each partition
- ▶ **Reactive policy:**
  - ▶ Identify the partition  $i$  with the longest backlog
  - ▶ For every resource having more than 75% resource usage in partition  $i$ , transfer the resource to partition  $i$  from partition  $j$  where
    - ▶ Partiton  $j$  has smallest utilization of that resource and has no backlog
- ▶ **Utility based policy:**
  - ▶ Reinforcement learning based agent
  - ▶ Initialized using reactive policy. Trained for 40k time steps.

# Comparison of DRA-FRL against two other policies

---

- ▶ Tested over 40k time steps.
- ▶ Average scheduling cost:
  - ▶  $Cost_i(t) = [ K * B_i(t+1) + N_{CPU}(t) C_{CPU}(t) + N_{MEM}(t) C_{MEM}(t) ]$
  - ▶ Averaged over all partitions  $i$ , and all time steps  $t$ .

Table 1

Average cost per time step of the considered resource allocation policies

	Optimal fixed allocation	Reactive policy	Utility-based policy
Transfer cost = 0	7.9	2.12	1.56
Transfer cost = 0.31	7.9	2.36	1.99

---

# Advantages of RL based Schedulers

---

- ▶ **Predicts long term rewards**
  - ▶ More flexible than the fixed policy schedulers
- ▶ **Adaptive to randomly varying environment**
  - ▶ Learns from past experiences
- ▶ **Avoids the burden of devising sophisticated scheduling policies**
  - ▶ Just choose appropriate reward function, and state variables
- ▶ **Simple algorithm. Can be implemented in Hardware.**
  - ▶ Self optimizing memory controller (Ipek et al., ISCA 2008 [5])
- ▶ **Ample training data availability**
  - ▶ New data arrives in every scheduling clock cycle.

# Challenges

---

- ▶ **Proper care must be taken to limit number of states**
  - ▶ Number of states grow exponentially with number of state variables
- ▶ **State value function (long term reward) are usually difficult to estimate exactly**
  - ▶ Transition probabilities are often not known
  - ▶ Usually approximated using Q-learning, or fuzzy rule base
- ▶ **State value functions need to be stabilized quickly**
  - ▶ Environment states may change drastically causing the learned state values obsolete
  - ▶ Typically useful if threads have long life
- ▶ **Difficulty to interpret and predict scheduler behavior**
  - ▶ Works as a black box

# Conclusion

---

- ▶ **RL-based scheduling has recently gained significant interest**
  - ▶ On going research leading research groups
- ▶ **Wide applicability in many different areas:**
  - ▶ Thread schedulers, Resource Schedulers
  - ▶ Memory controller scheduling (Ipek et al., ISCA 2008 [5])
  - ▶ Power minimization in large scale data centers (Berral et al., 2010[6])

# References

---

- ▶ [1] Cache-fair thread scheduling for multicore processors, A. Fedorova, M. Seltzer, and M.D. Smith, *Technical Report TR-17-06, 2006*, Division of Engineering and Applied Sciences, Harvard University, 2006.
- ▶ [2] Operating System Scheduling on Heterogeneous Core Systems, A. Fedorova, David Vengerov, and Daniel Doucette, in *Proceedings of the First Workshop on Operating System Support for Heterogeneous Multicore Architectures*, at PACT 2007, Brasov, Romania.
- ▶ [3] A reinforcement learning approach to dynamic resource scheduling allocation, D. Vengerov, *Engineering Applications of Artificial Intelligence*, vol. 20, no 3, p. 383-390, Elsevier, 2007.
- ▶ [4] A reinforcement learning framework for utility-based scheduling in resource-constrained systems, D. Vengerov, *Future Generation Computer Systems*, vol. 25, p. 728-736, Elsevier, 2009.
- ▶ [5] Self-optimizing memory controllers: A reinforcement learning approach, E. Ipek, O. Mutlu, J. Martinez, R. Caruana. *In proceedings of ISCA*, 2008.

# References

---

- ▶ [6] Towards energy-aware scheduling in data centers using machine learning, J. I. Berral et al., in *Proceedings of the 1st International Conference on energy-Efficient Computing and Networking*, ACM, 2010,.

# Methods to Solving the Optimization Problem: Policy Iteration

- ▶ **Policy iteration:** efficient iterative algorithm
  - ▶ Assumes knowledge of state transition probabilities
  - ▶ Based on dynamic programming

