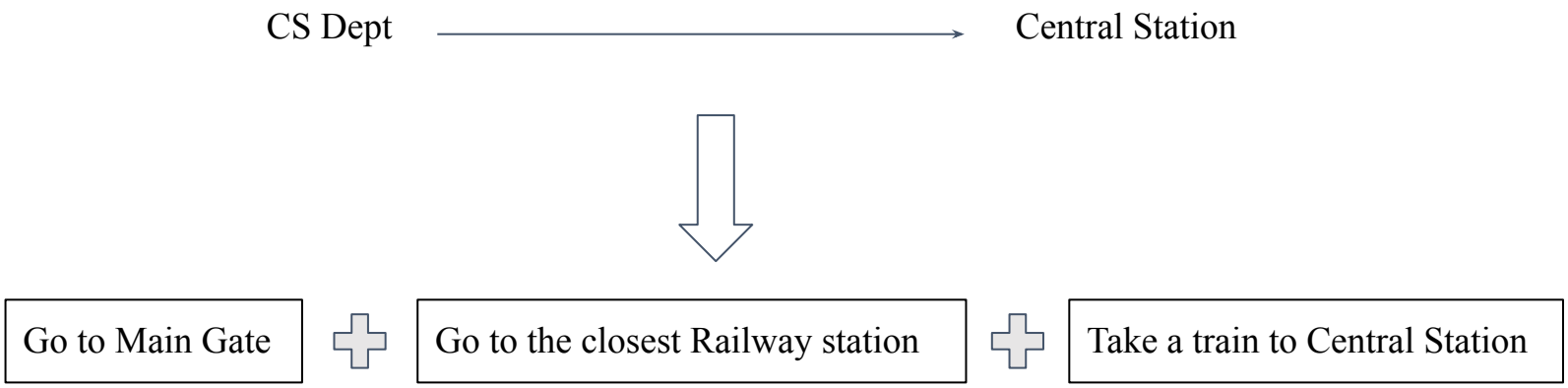
Introduction to Hierarchical Reinforcement Learning

B. Ravindran

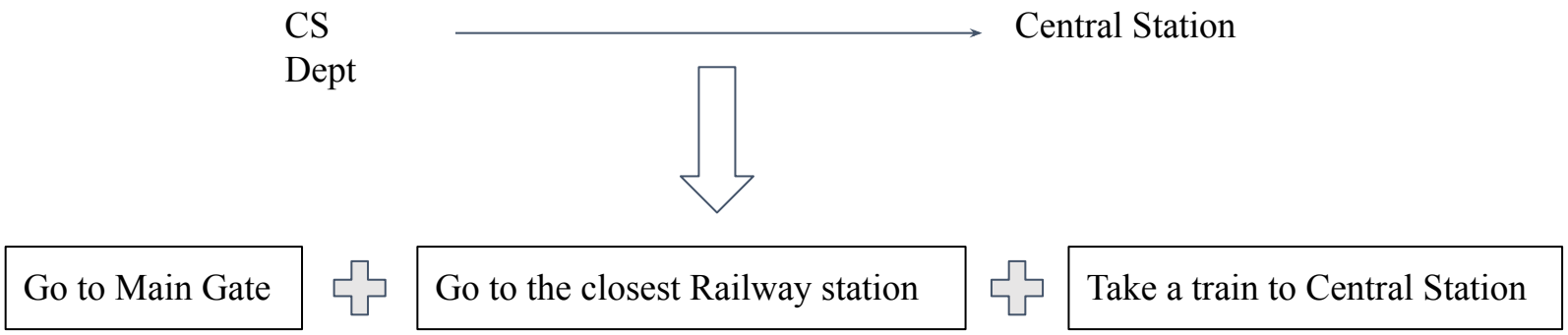
- Why function approximation?
  - ‘Scale up’ the solution methods for solving large problems
  - ‘Trying’ to find structure in the problem (Generalization)

- Another approach:
  - We can look for a ‘higher order structure’.
    [Sub-problems that repeatedly occurs as part of the larger problem
    Example: Navigation]



TEMPORAL ABSTRACTION:

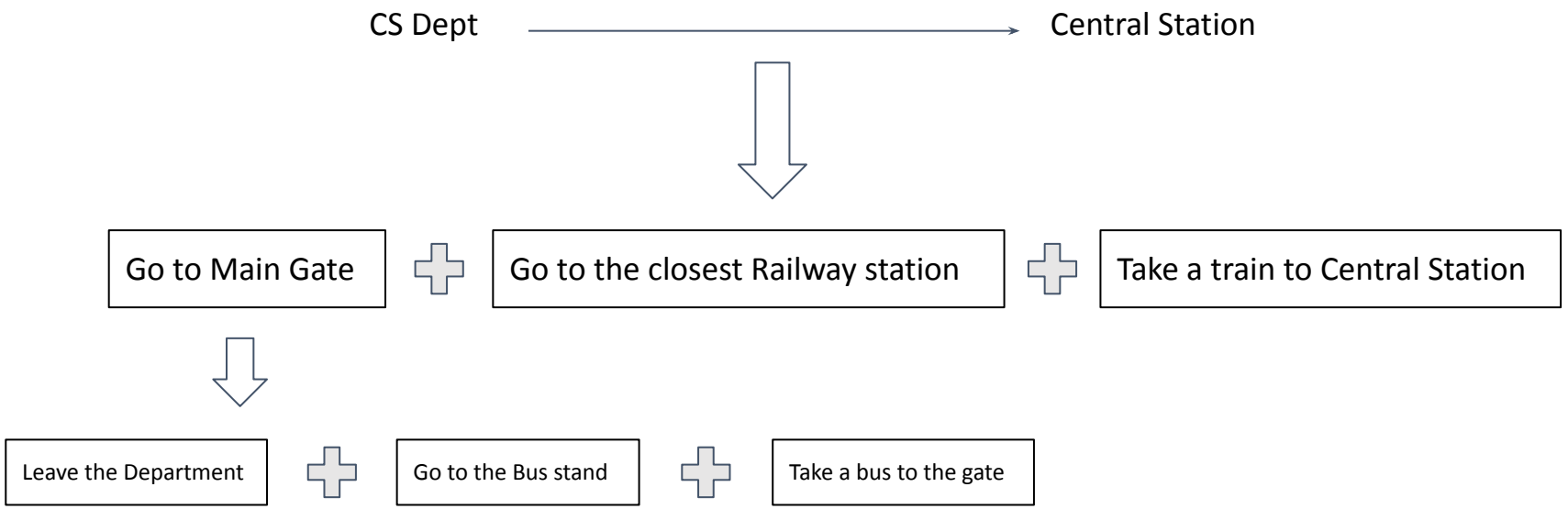
- These new sub-problems are obtained by abstracting away the notion of time.



- Solving '<u>Sub-problems</u>' (+) that could take '<u>varying amounts of time</u>'.

- **TRANSFER/REUSABILITY:**

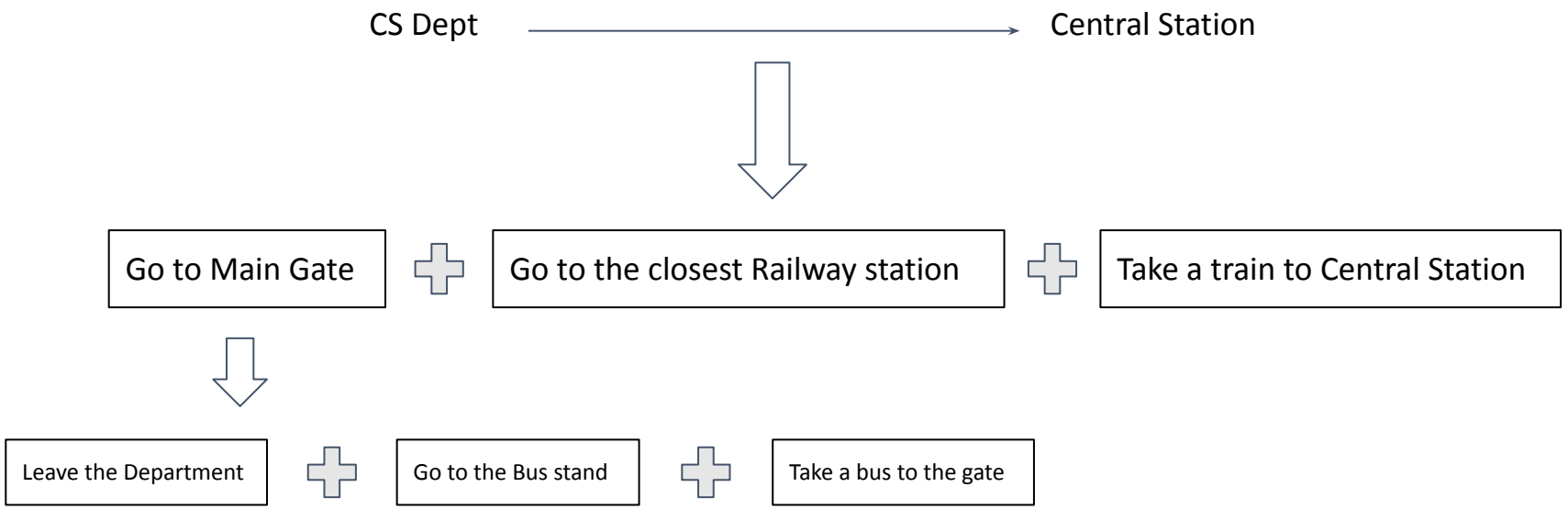
  - These ‘sub-problems’ can be broken down into even smaller problems.



  - These ‘Sub-problems’ tend to re-occur very often.
    (as sub-parts of other problems as well)
  - Hierarchies could give us ‘Meaningful Modules’ to optimize.

- **MEANINGFUL ABSTRACTIONS:**

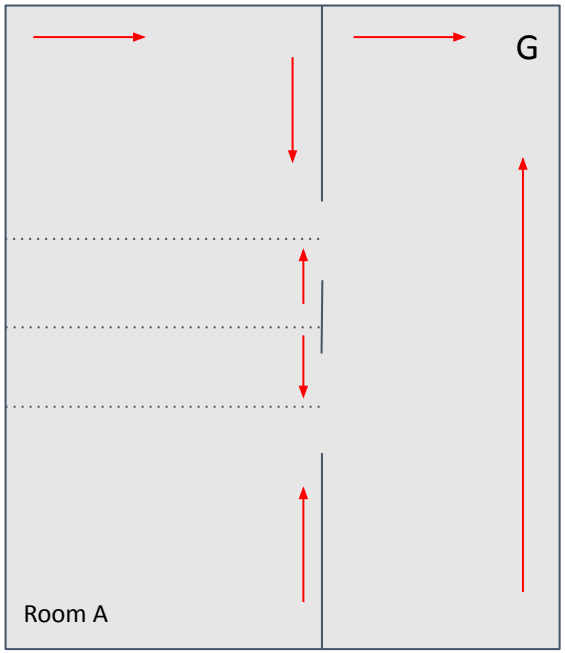
  - Optimal solution for a ‘sub-problem’ does not depend on other parallel sub-problems.

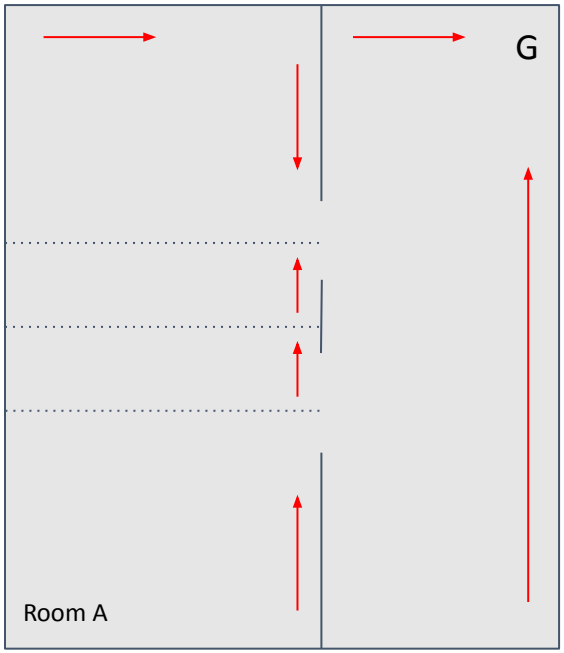


  - This way, we have a simpler problem to solve.
    (**State-space** of a sub problem is way smaller than a bigger problem)

  - Efficient learning of sub problems (+) Reusability for other problems.

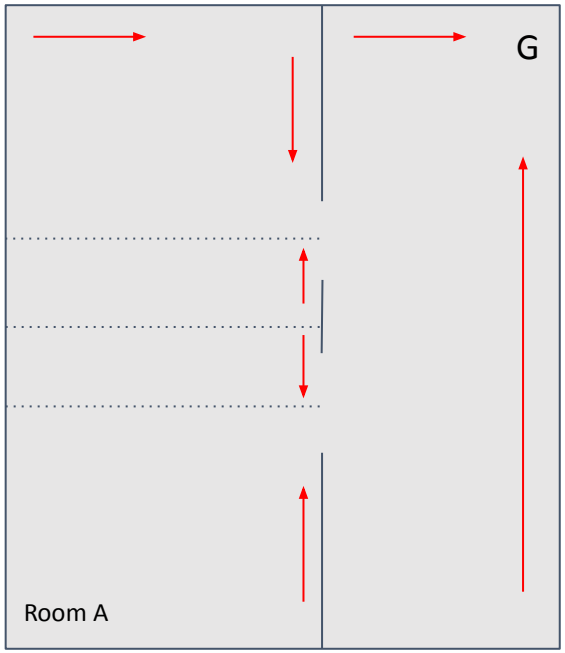
Optimality

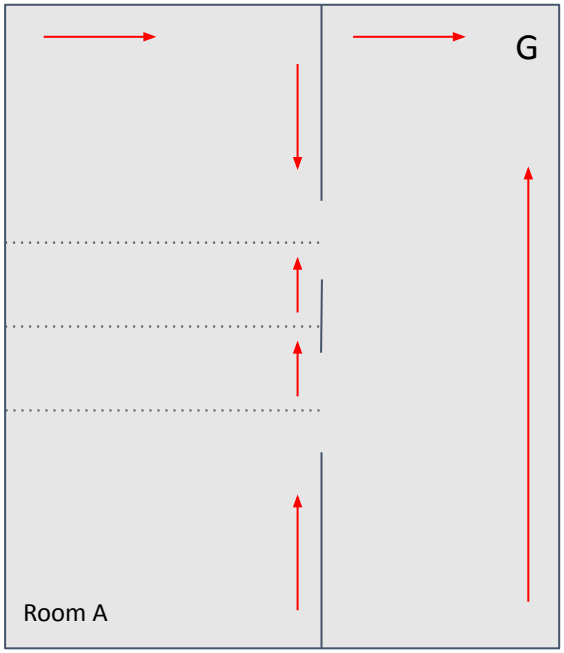




Which of the following is the ‘optimal’ policy for an agent starting from Room A that wishes to reach ‘G’?

Optimality



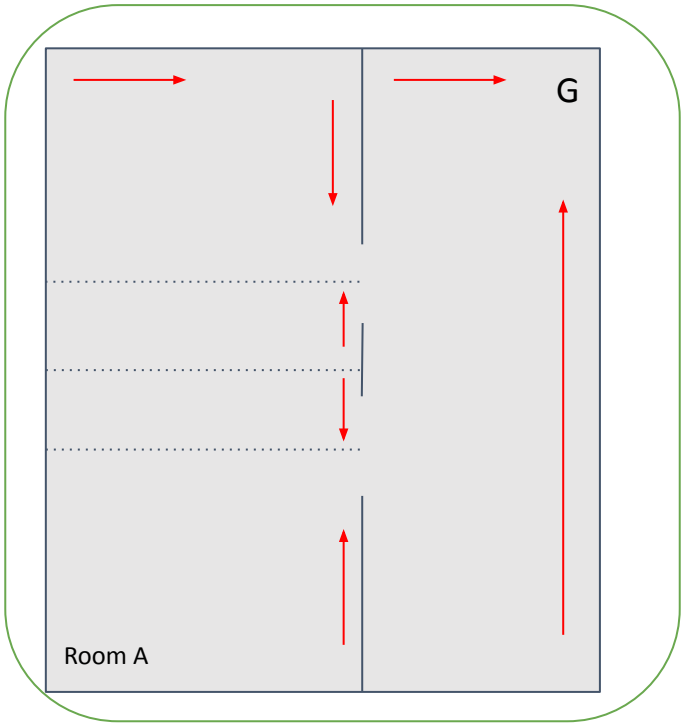


Which of the following is the ‘optimal’ policy for an agent starting from Room A that wishes to reach ‘G’?

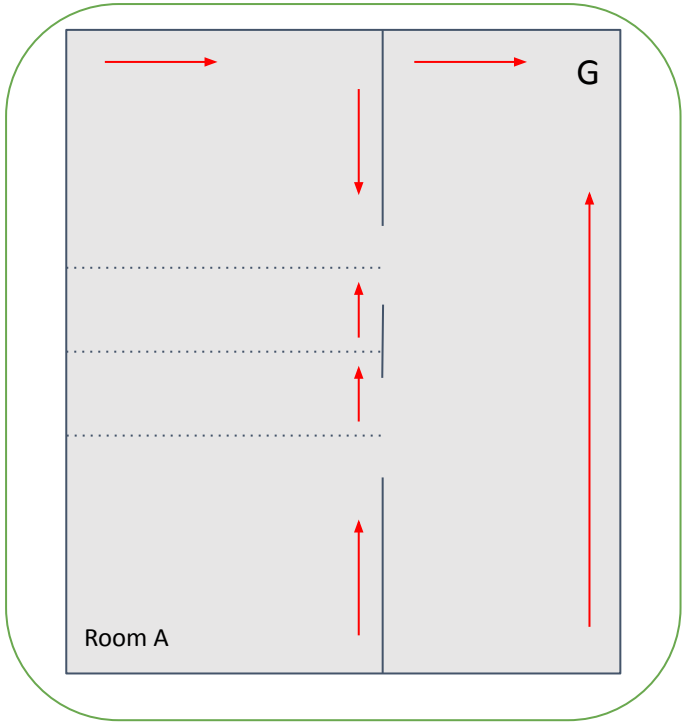
Both ‘LEFT’ & ‘RIGHT’ could be the answer depending on the which optimality is considered.

- Hierarchically Optimal & Recursively Optimal

Recursively Optimal (module-wise best way)



Hierarchically Optimal (Overall best way)

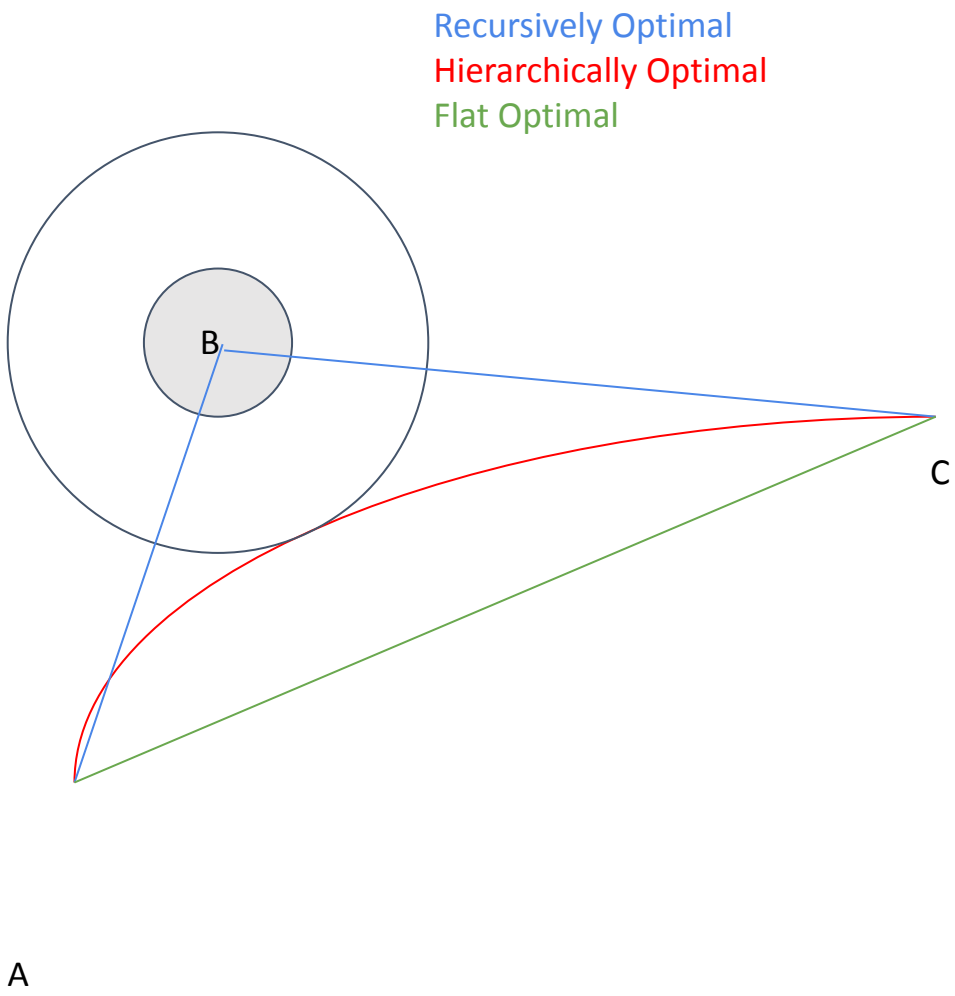


- Search through policies that respect the hierarchy. (Leave Room A to go to next room)
- Hierarchically Optimal policies are policies that gives the best solution yet satisfies the proposed hierarchy
- Recursively Optimal policies are the best policies formed by putting together component-wise policies obtained by optimizing for each component.

- Flat Optimality -> Optimal policy (which does not consider any hierarchy)

Goal: A -> C

Hierarchy presented:
A -> C : (A -> B) + (B -> C)



Comparison

- Cost-wise:
  Flat (better than) Hierarchical (better than) Recursive

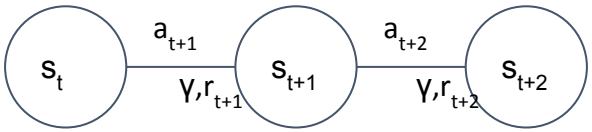
- Flat optimal is difficult to obtain.
- Recursive is better than hierarchical?
  - Modular solutions (reusable)
  - Learning horizon could be simpler (if problem can be broken down into meaningful subproblems)
  - The optimal sub-solutions put together could be bad.

- Recursive Optimality works well if there is a mechanism that can treat the 'sub-problem' solutions as actions that can be used to reduce the learning horizon of the bigger problem. SMDPs help us here.

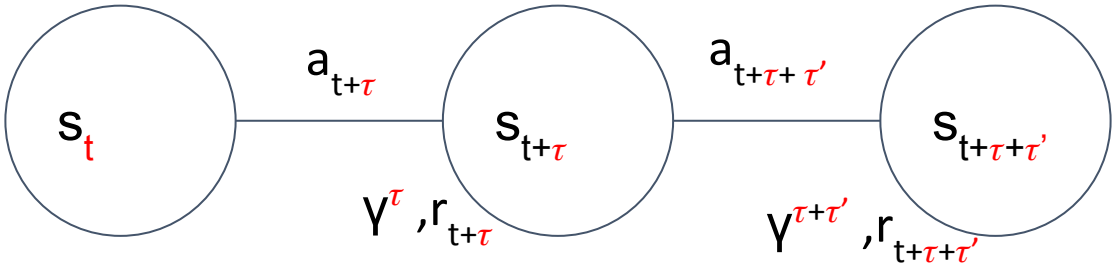
SMDP - (Semi Markov Decision Process)

- Actions have durations (τ - holding time)

MDP



SMDP



- Each action could have different time duration. (τ & τ')

SMDPs

Transitions: <S, A, P, R>

One-step transition probabilities:

$$p(s', \tau \mid s, a) \vdash \begin{matrix} p(s' \mid s, a) \\ p(\tau \mid s, a) \end{matrix}$$

One-step expected-rewards:

$$E[r \mid s, a, s', \tau]$$

SMDP Q-Learning

- Q-Update Equation (SMDPs):

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left[\bar{r}_{t+\tau} + \gamma^{\tau} \max_{a'} Q\left(s_{t+\tau}, a'\right) - Q(s_t, a_t) \right]$$

- The ‘reward’ used in the above update (reward obtained from the update sample with ‘tou’ timestep) can be framed as

$$\bar{r}_{t+\tau} = r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{\tau-1} r_{t+\tau}$$