# Scalable Ride-hailing using Reinforcement Learning

Yueying Li, Yujia Zhang

May 17, 2021

ORIE 6590 Final Presentation

# Ride-hailing problems

- Large scale
- Complicated dynamics over space, time, and participants
- Combinatorial challenge to arrange many cars at the same time

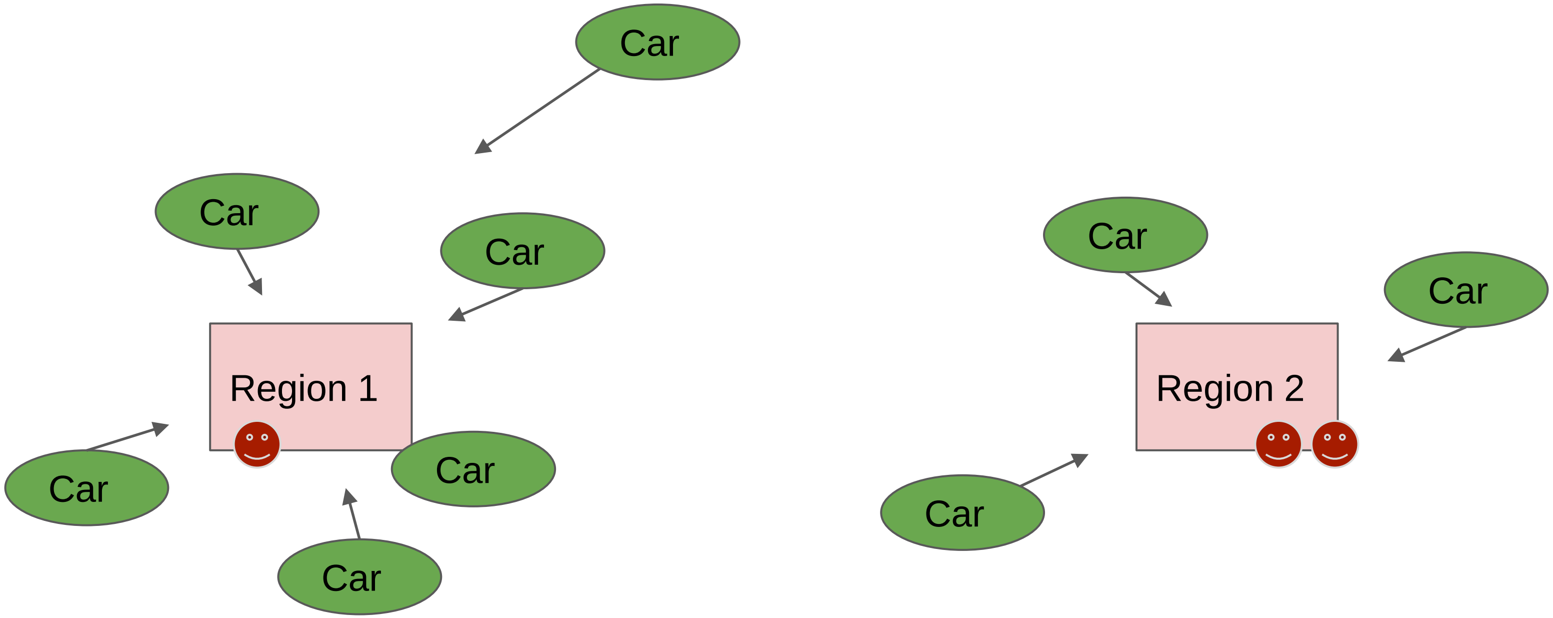
# Ride-hailing problems

- Large scale
- Complicated dynamics over space, time, and participants
- Combinatorial challenge to arrange many cars at the same time

- Can we do this sequentially, making the decision for one car at a time?

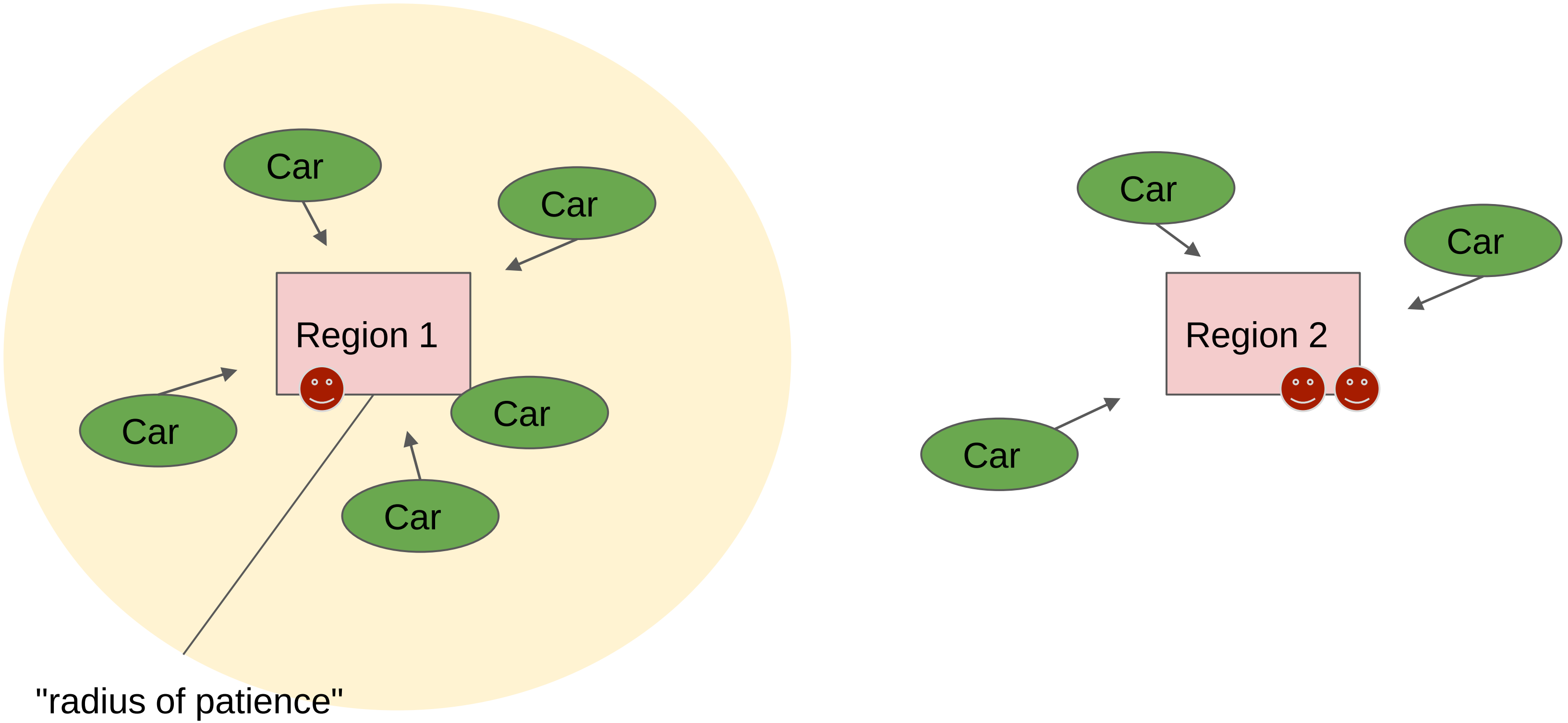
  *Feng, Jiekun, Mark Gluzman, and J. G. Dai. "Scalable Deep Reinforcement Learning for Ride-Hailing." IEEE Control Systems Letters (2020).*

  **"Sequential decision making" process**

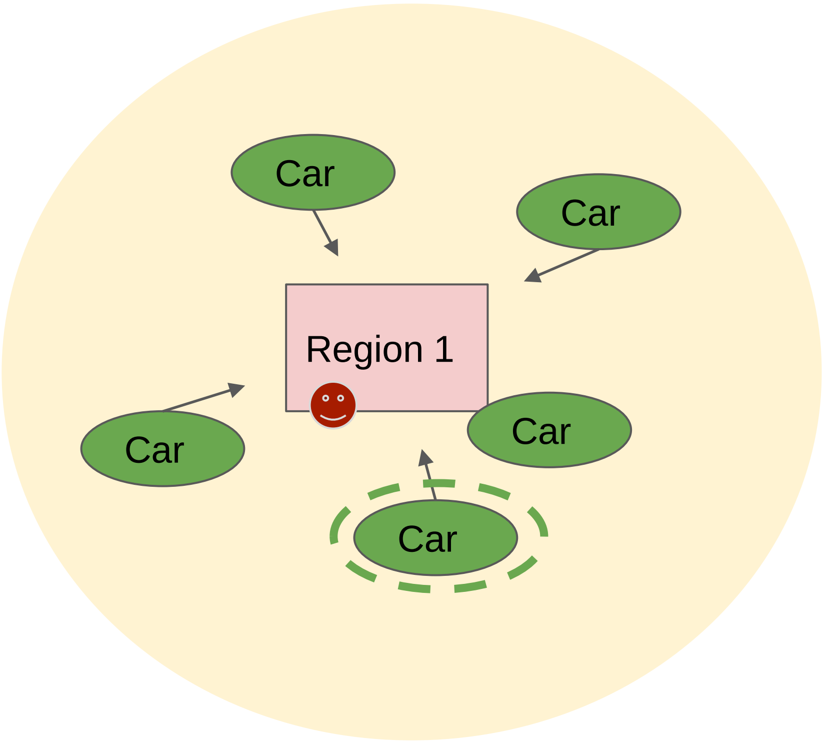


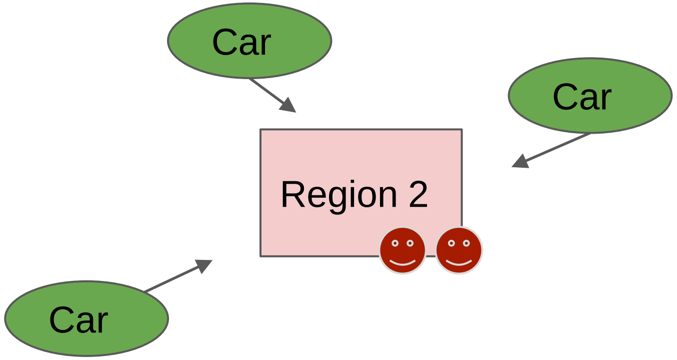

Cars are characterized by
(goal region, distance/time to goal region)

Passenger requests:
[R(1,1), R(1,2), R(2,1), R(2,2)]




Passenger requests:
[R(1,1), R(1,2), R(2,1), R(2,2)]

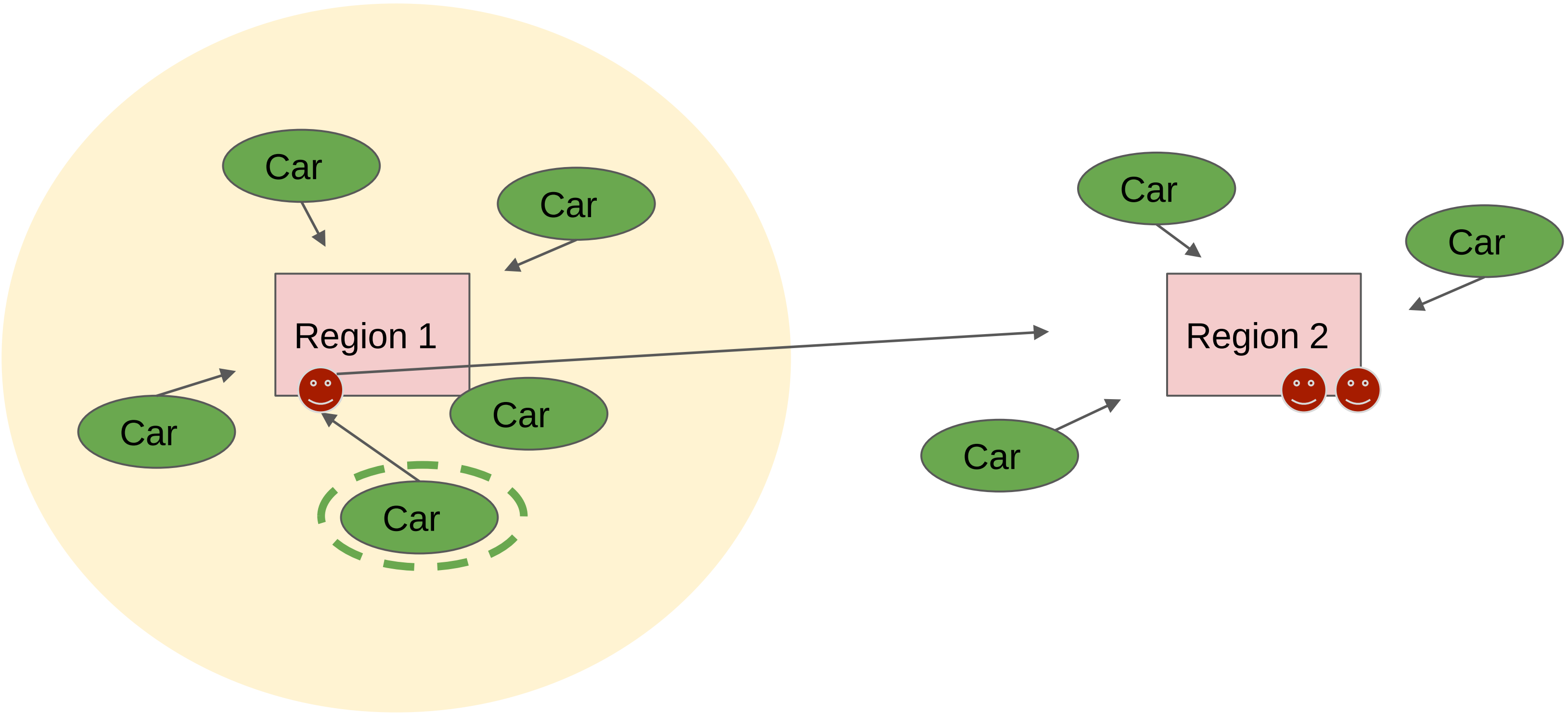



Choose a trip destination for each car:

- pick up a passenger requesting a ride from region 1

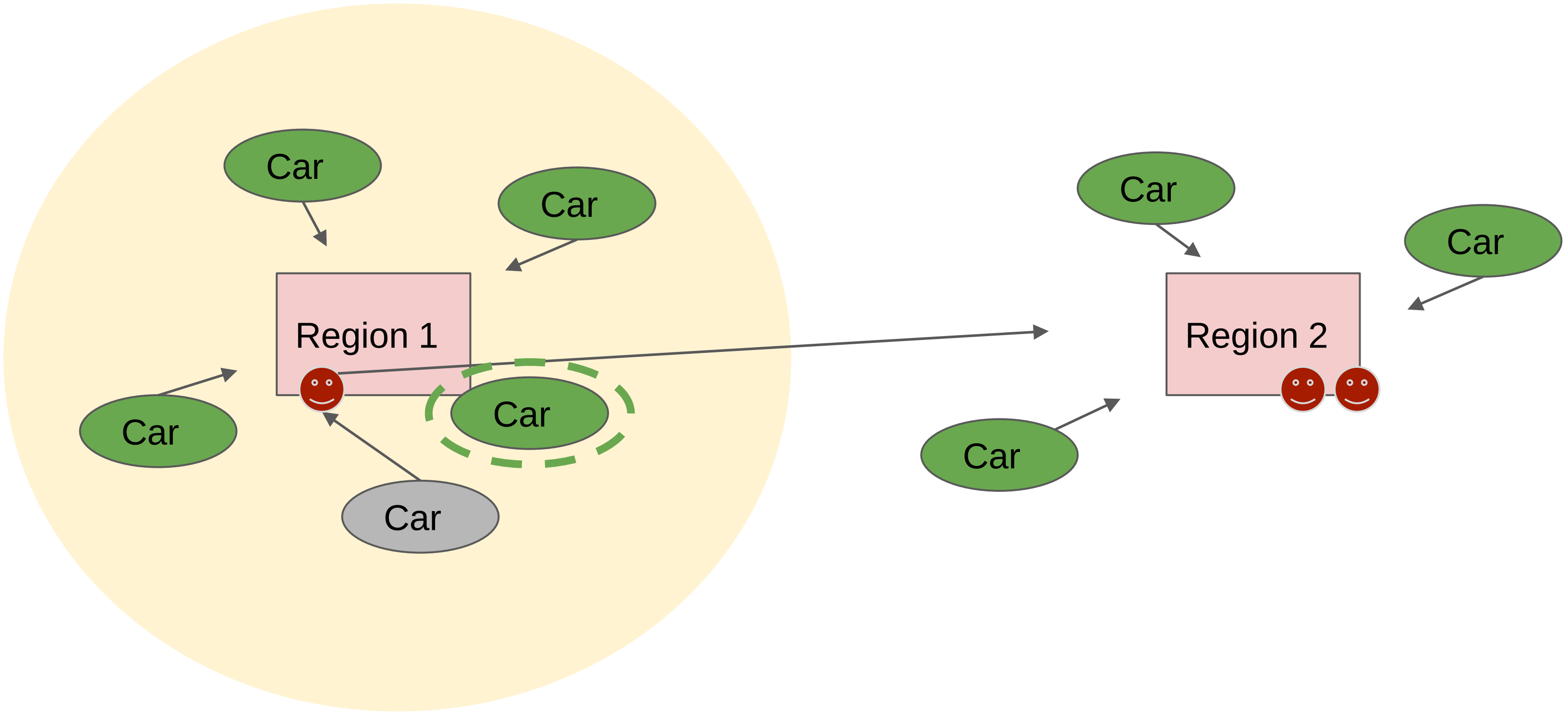
Passenger requests:
[R(1,1), R(1,2), R(2,1), R(2,2)]




Choose a trip destination for each car:

- pick up a passenger requesting a ride from region 1
    - change goal region of car to region 2
    - decrease unmet passenger requests by 1

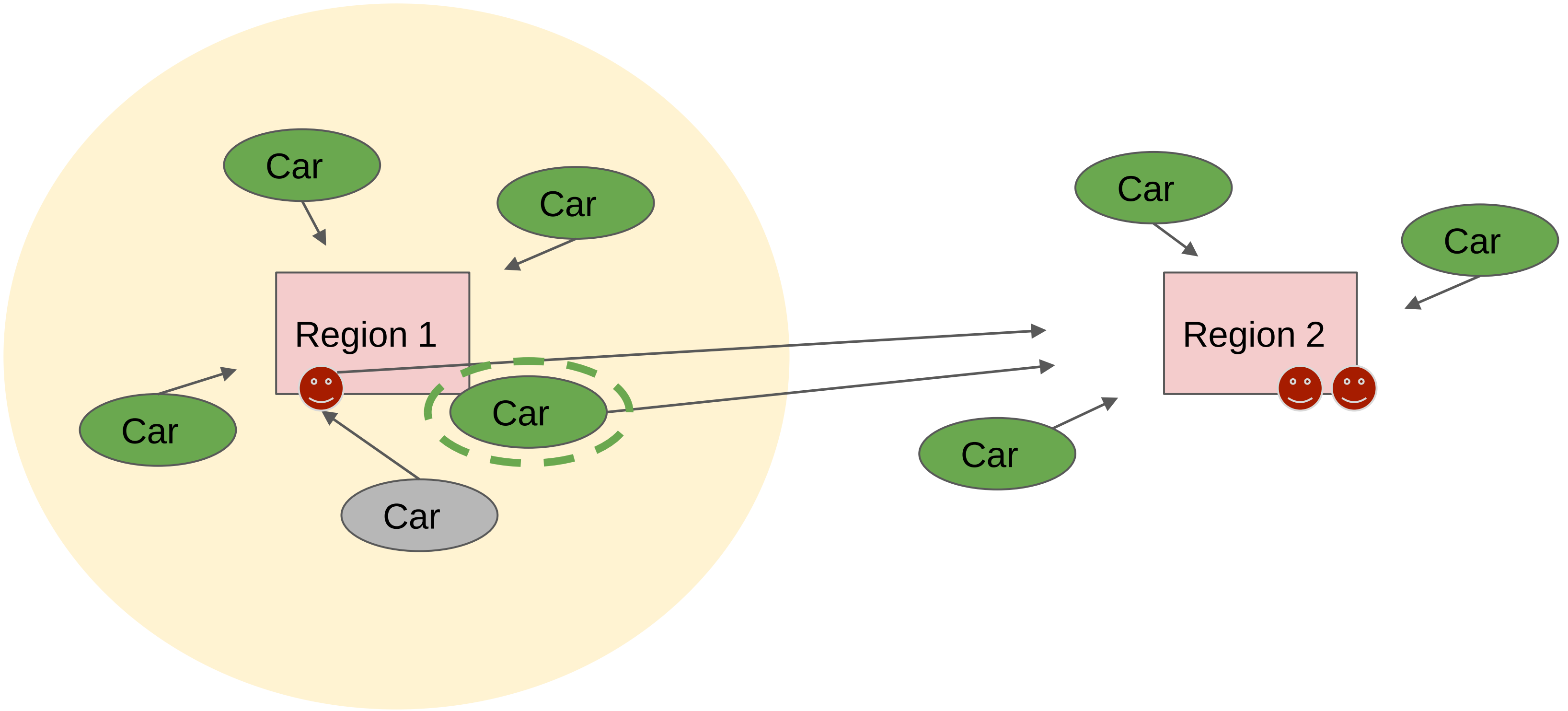
Passenger requests:
[R(1,1), R(1,2), R(2,1), R(2,2)]




Choose a trip destination for each car:

- pick up a passenger requesting a ride from region 1
- empty-car rerouting

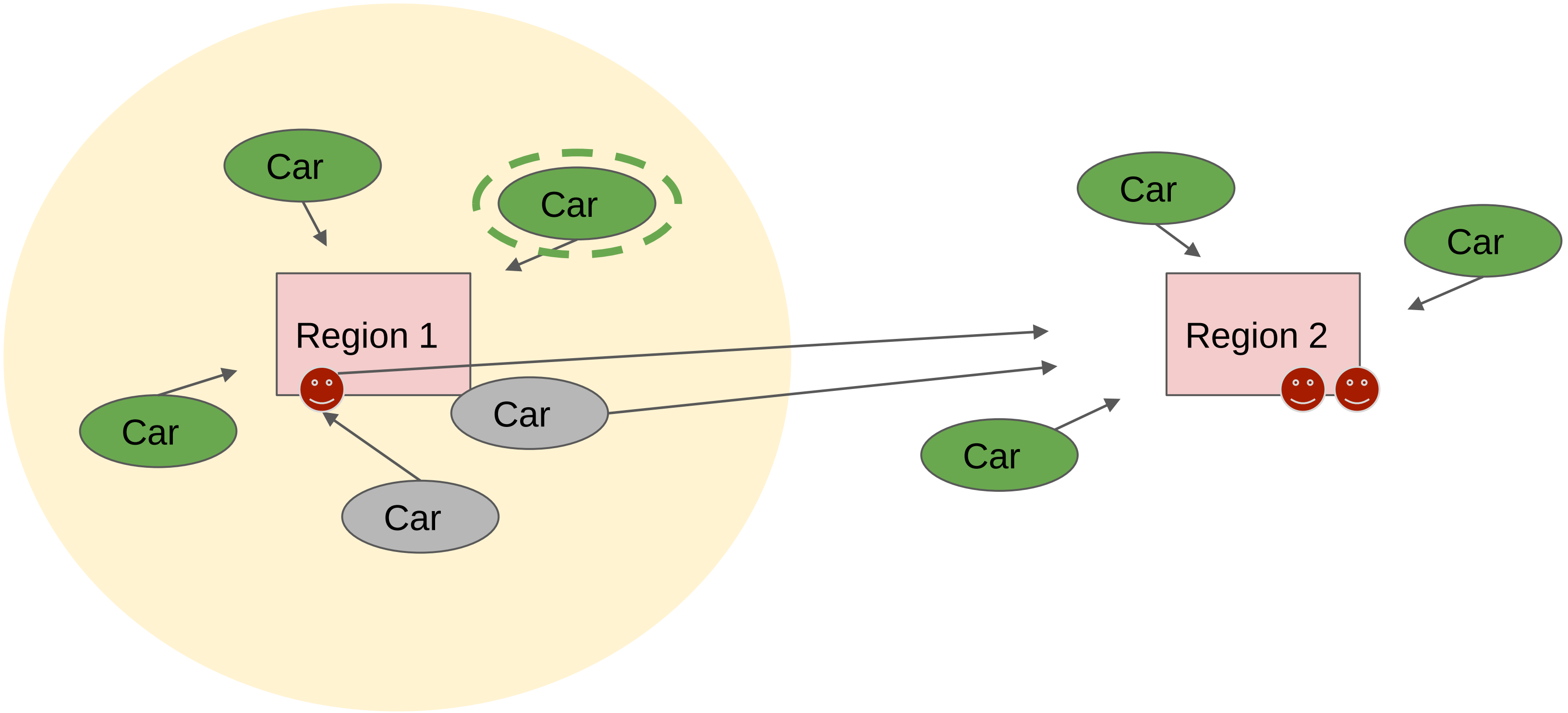
Passenger requests:
[R(1,1), R(1,2)-1, R(2,1), R(2,2)]




Choose a trip destination for each car:

- pick up a passenger requesting a ride from region 1
- empty-car rerouting

Passenger requests:
[R(1,1), R(1,2)-1, R(2,1), R(2,2)]




Choose a trip destination for each car:

- pick up a passenger requesting a ride from region 1
- empty-car rerouting
- don't do anything

Passenger requests:
[R(1,1), R(1,2)-1, R(2,1), R(2,2)]

# MDP formulation

- Finite horizon H=360, discrete state and action spaces, no discount
- **State space:** (time-of-day, cars, passengers)
- **Action space:** trip assignments {1, ..., R=5} x {1, ..., R} for each car
- **Transition:**
  - Within the same time-of-day:
    - change the (destination, distance to destination) of one car
    - record whether passenger request is met
  - At transition of time-of-day:
    - sample passenger requests
    - move cars forward one step
- **Reward:** 1 if a passenger ride request is met; 0 otherwise
- **Overall performance measure:** fraction of total requests filled

# PPO

$$\theta_{k+1} = \theta_k + \underbrace{\sqrt{\frac{2\delta}{g^T F^{-1} g}} F^{-1} g}_{\text{natural policy gradient}}$$

- Policy Gradient is challenging
  - Convergence Problem
  - Sensitive to the choice of step size
  - Poor sample efficiency
  - Second-order derivative can not be scalable
- PPO strikes a balance between ease of implementation, sample complexity, and ease of tuning.
  - Instead of a hard constraint, formalize it as a penalty in objective (PPO-Penalty)
    - Limit how far we can change the policy through KL-divergence
  - Use clipping to remove incentives for new policy to go far from old one

$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_k}\left[\sum_{t=0}^{T}\left[\min(r_t(\theta)\hat{A}_t^{\pi_k}, \mathrm{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k})\right]\right]$$

# One-step and n-step deep Q network

$$\min_{\theta} \sum_{s_t, a_t, s_{t+1} \in \mathcal{B}} L\left(Q_\theta(s_t, a_t),\ r(s_t, a_t) + \gamma \cdot \max_{a'} Q_{\bar{\theta}}(s_{t+1}, a')\right)$$

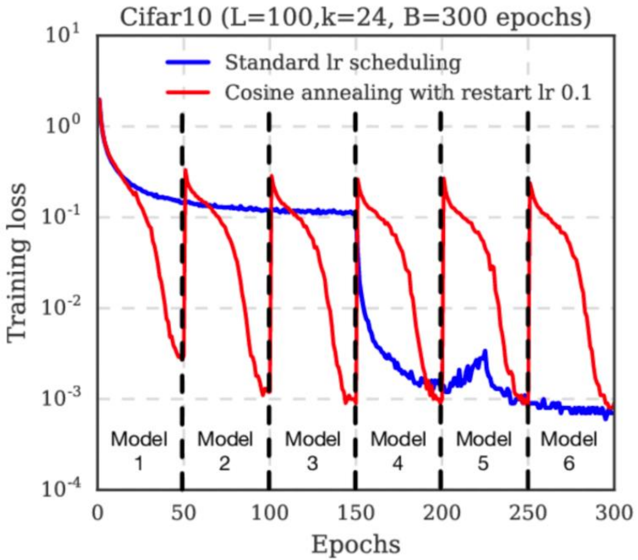
$$\min_{\theta} \sum_{s_t, a_t, s_{t+n} \in \mathcal{B}} L\left(Q_\theta(s_t, a_t),\ \sum_{i=0}^{n-1} \gamma^i r(s_{t+i}, a_{t+i}) + \gamma^n \cdot \max_{a'} Q_{\bar{\theta}}(s_{t+n}, a')\right).$$

- Tried n=5
- Sample from a replay buffer (size 512)
- Target network is updated periodically (every 200 steps)
- Use Huber loss to measure the difference between main Q network and target

- **Problem: some (s,a) are not visited enough (or at all) to guarantee an accurate estimate**

# Lessons learned in implementation

- Learning Rate Scheduling
  - Using Cosine Scheduler is helpful for convergence
  - Simulated restart of the learning process, reuse good weights
- Activation Function
  - ReLU works better than TanH
  - Sparser model
  - Avoid vanishing gradient problem
  - Fewer computation
- Regularization
  - L2 on Embedding layer
- Hyperparameter optimization
  - Clipping parameter




$$\eta_t = \eta^i_{min} + \frac{1}{2}(\eta^i_{max} - \eta^i_{min})\left(1 + \cos\left(\frac{T_{cur}}{T_i}\pi\right)\right)$$

# Future work

- Variance Reduction Techniques:
  - TD(lambda) or TD(n) incorporated in the advantage function estimation
- Domain adaptation
  - Zero-shot learning for unseen state

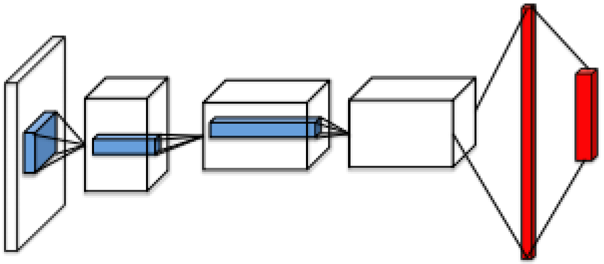
| | SVRG | SAG | SGD | SDCA |
|---|---|---|---|---|
| **Memory** | O(d) | O(nd) | O(d) | O(nd) |
| **Convergence*** | Linear | Linear | Sub-linear | Linear |
| **Non-smooth**** | ❎ | ❎ | ✅ | ✅ |

* In terms of SC, smooth function

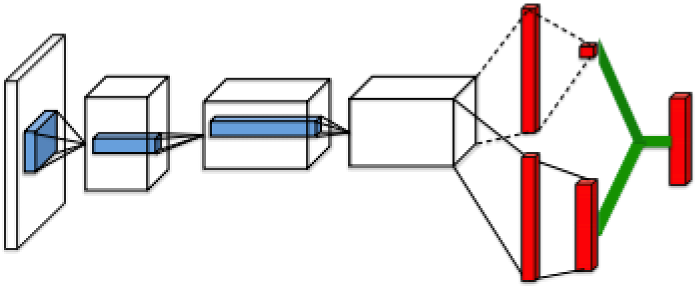
** At the time when the paper is published, this paper did not show convergence evidence on non-smooth objective functions

# Future work

- Use Dueling Network:



estimate Q(s,a) directly



estimate Q(s,a) = V(s) + A(s,a)