# Parallelism in RL

# Overview

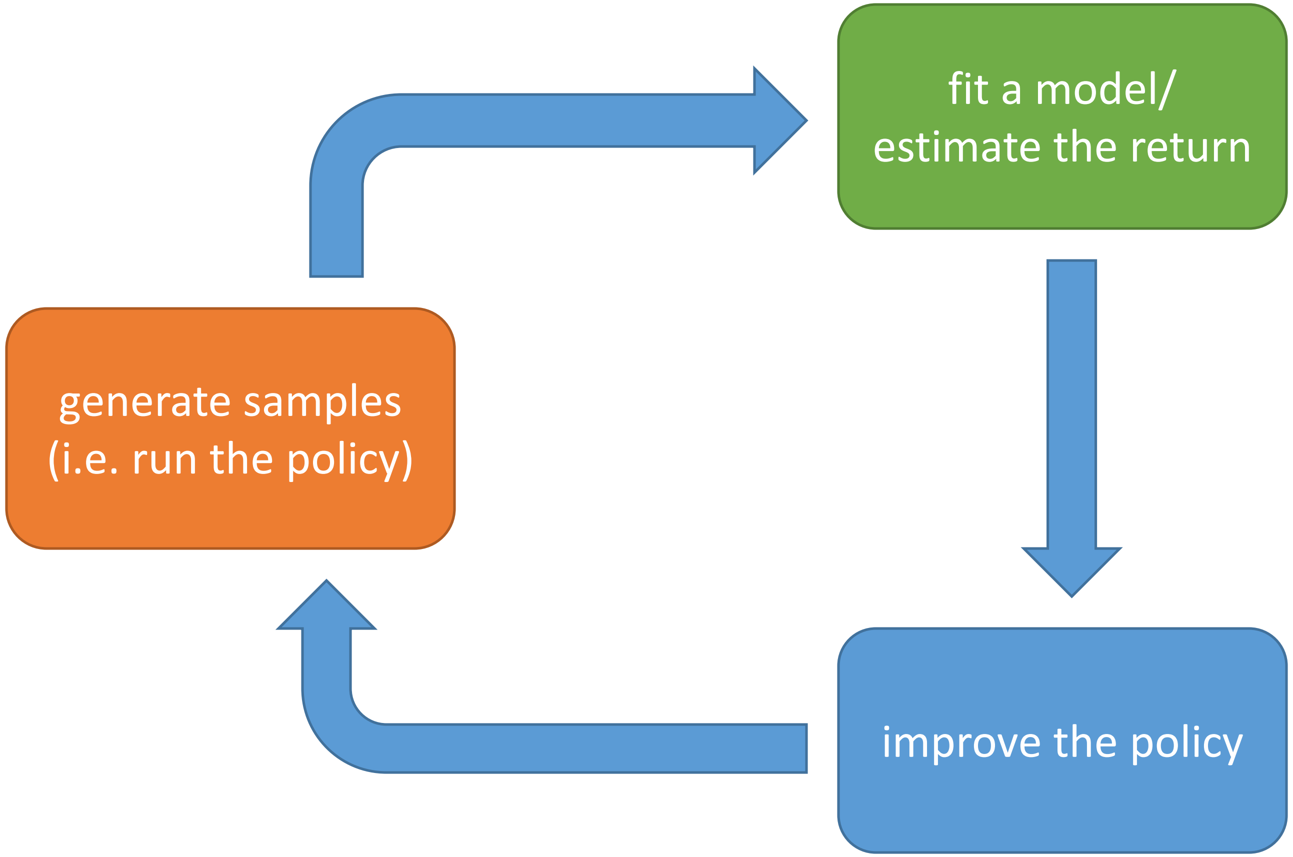
1. We learned about a number of policy search methods
2. These algorithms have all been *sequential*
3. Is there a natural way to parallelize RL algorithms?
   - Experience sampling vs learning
   - Multiple learning threads
   - Multiple experience collection threads

# Today's Lecture

1. What can we parallelize?
2. Case studies: specific parallel RL methods
3. Tradeoffs & considerations

- Goals
  - Understand the high-level anatomy of reinforcement learning algorithms
  - Understand standard strategies for parallelization
  - Tradeoffs of different parallel methods

# High-level RL schematic




estimate $p(\mathbf{s}'|\mathbf{s},\mathbf{a})$ (model-based)

compute $\hat{Q} = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ (MC policy gradient)

fit $Q_\phi(\mathbf{s},\mathbf{a})$ (actor-critic, Q-learning)

optimize $\pi_\theta(\mathbf{a}|\mathbf{s})$ (model-based)

$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$ (policy gradient)

$\pi(\mathbf{s}) = \arg\max Q_\phi(\mathbf{s},\mathbf{a})$ (Q-learning)

# Which parts are slow?

real robot/car/power grid/whatever:
1x real time, until we invent time travel

MuJoCo simulator:
up to 10000x real time

generate samples
(i.e. run the policy)

fit a model/
estimate the return

$\hat{Q} = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$

trivial, fast

fit $Q_\phi(\mathbf{s}, \mathbf{a})$

expensive, but non-trivial to parallelize

improve the policy

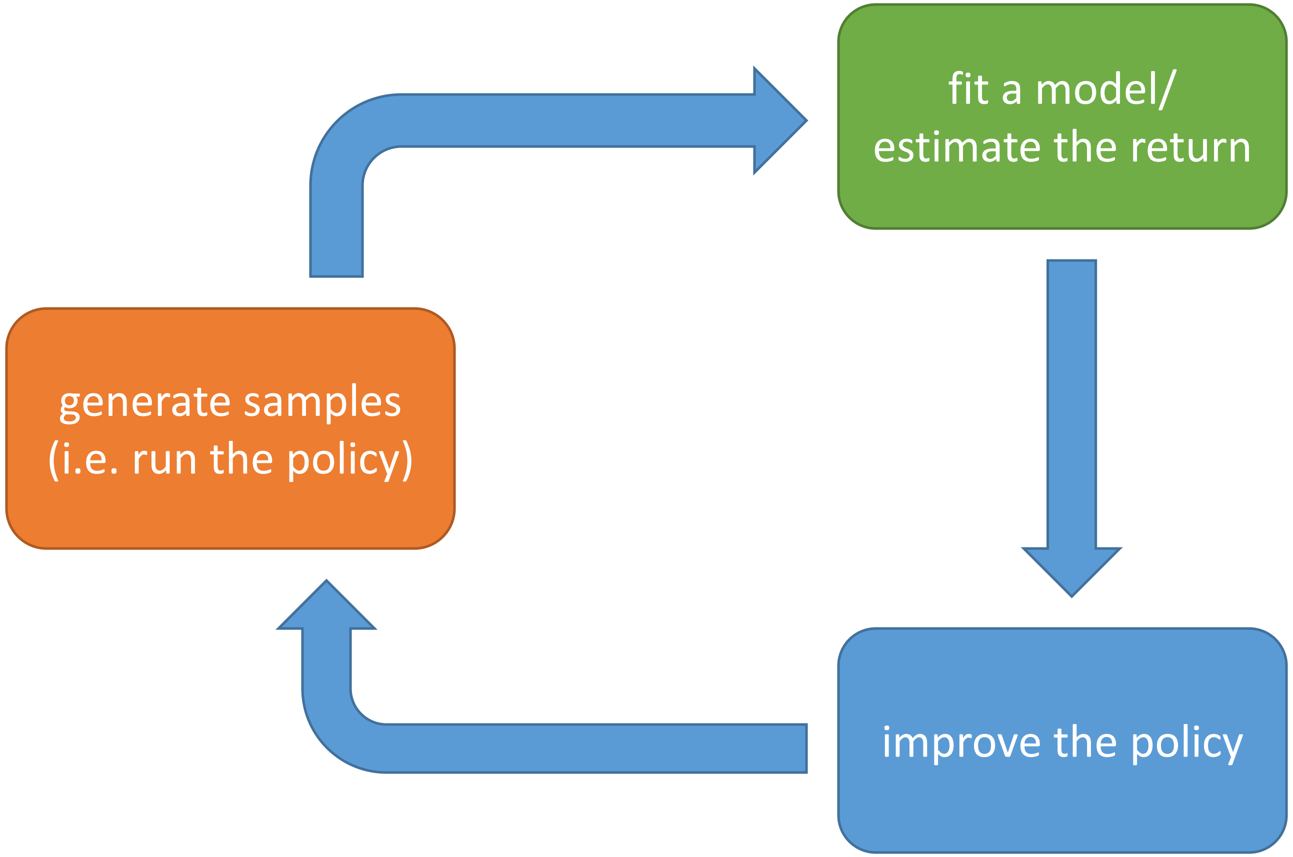
$\pi(\mathbf{s}) = \arg\max Q_\phi(\mathbf{s}, \mathbf{a})$

trivial, nothing to do

optimize $\pi_\theta(\mathbf{a}|\mathbf{s})$ (model-based)

expensive, but non-trivial to parallelize

# Which parts can we parallelize?




fit $Q_\phi(\mathbf{s},\mathbf{a})$

parallel SGD

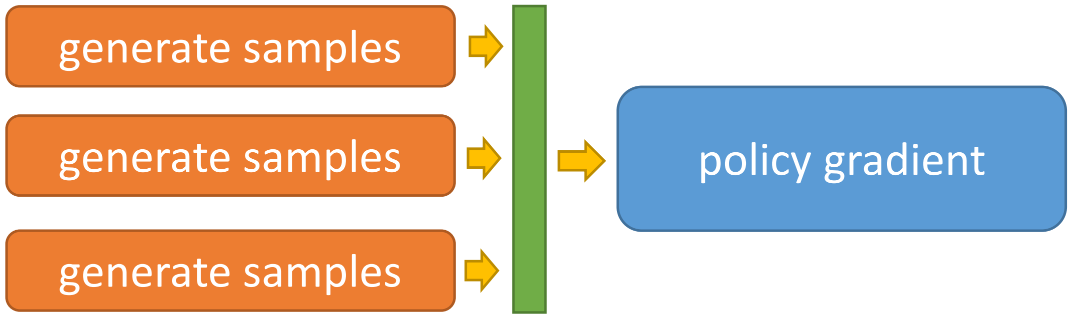
optimize $\pi_\theta(\mathbf{a}|\mathbf{s})$ (model-based)

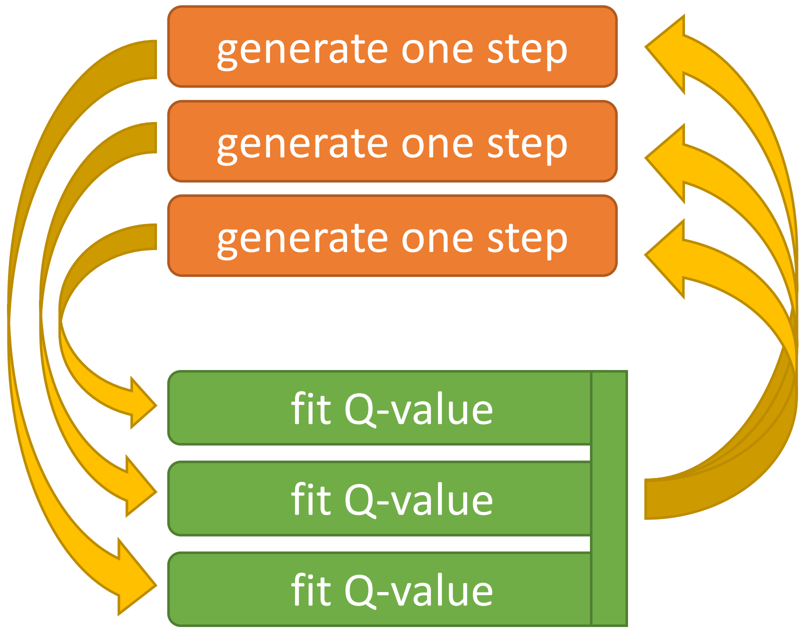
parallel SGD

Helps to group data generation and training
(worker generates data, computes gradients, and gradients are pooled)

# High-level decisions

1. Online or batch-mode?
2. Synchronous or asynchronous?

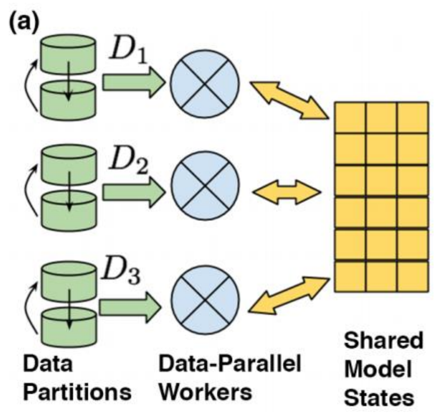


# Relationship to parallelized SGD

fit a model/
estimate the return

improve the policy

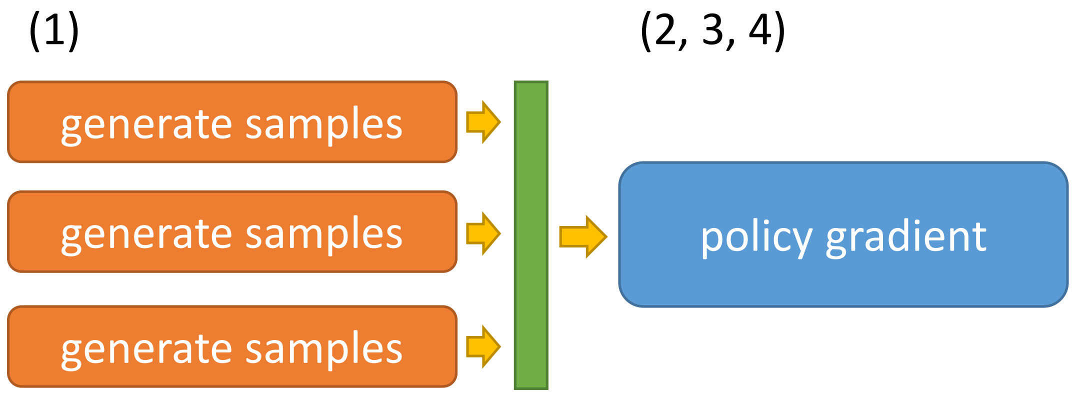



Dai et al. '15

1. Parallelizing model/critic/actor training typically involves parallelizing SGD
2. Simple parallel SGD:
   1. Each worker has a different slice of data
   2. Each worker computes gradients, sums them, sends to parameter server
   3. Parameter server sums gradients from all workers and sends back new parameters
3. Mathematically equivalent to SGD, but not asynchronous (communication delays)
4. Async SGD typically does not achieve perfect parallelism, but lack of locks can make it much faster
5. Somewhat problem dependent

# Simple example: sample parallelism with PG

1. collect samples $\tau_i = \{\mathbf{s}_1^i, \mathbf{a}_1^i, \ldots, \mathbf{s}_T^i, \mathbf{a}_T^i\}$ by running $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ N times
2. compute $r_i = r(\tau_i)$
3. compute $\nabla_i = \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)(r_i - b)$
4. update: $\theta \leftarrow \theta + \alpha \sum_i \nabla_i$

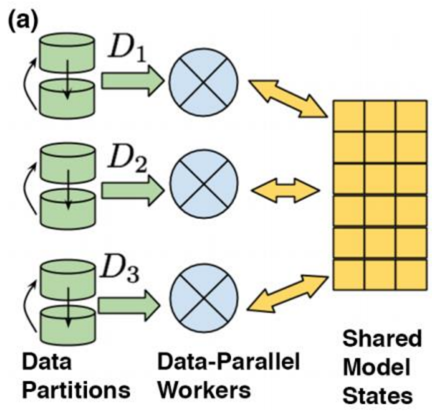


# Simple example: sample parallelism with PG

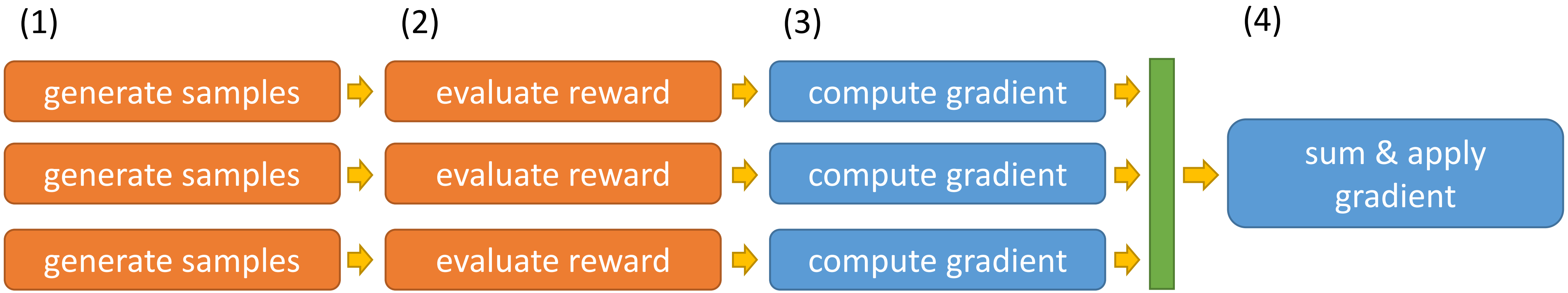
1. collect samples $\tau_i = \{\mathbf{s}_1^i, \mathbf{a}_1^i, \ldots, \mathbf{s}_T^i, \mathbf{a}_T^i\}$ by running $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ N times
2. compute $r_i = r(\tau_i)$
3. compute $\nabla_i = \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)(r_i - b)$
4. update: $\theta \leftarrow \theta + \alpha \sum_i \nabla_i$

# Simple example: sample parallelism with PG

1. collect samples $\tau_i = \{\mathbf{s}_1^i, \mathbf{a}_1^i, \ldots, \mathbf{s}_T^i, \mathbf{a}_T^i\}$ by running $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ N times
2. compute $r_i = r(\tau_i)$
3. compute $\nabla_i = \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)(r_i - b)$
4. update: $\theta \leftarrow \theta + \alpha \sum_i \nabla_i$



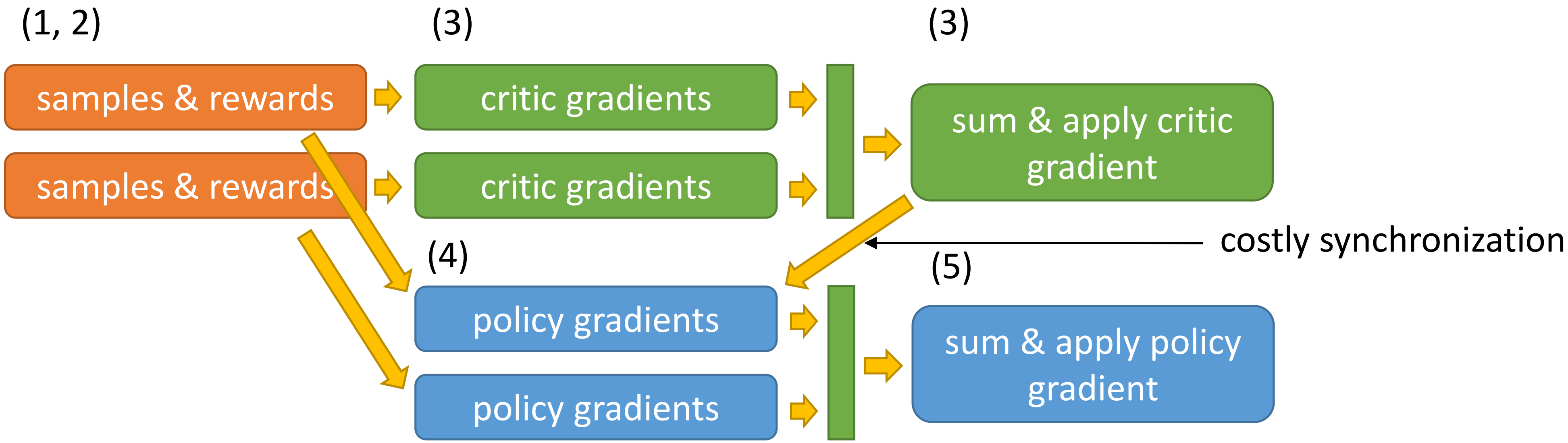

Dai et al. '15



# What if we add a critic?

1. collect samples $\tau_i = \{\mathbf{s}_1^i, \mathbf{a}_1^i, \dots, \mathbf{s}_T^i, \mathbf{a}_T^i\}$ by running $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ N times
2. compute $r_i = r(\tau_i)$
3. update $\hat{A}_\phi(\mathbf{s}_t^i, \mathbf{a}_t^i)$ with regression to target values
4. compute $\nabla_i = \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right) \hat{A}_\phi(\mathbf{s}_t^i, \mathbf{a}_t^i)$
5. update: $\theta \leftarrow \theta + \alpha \sum_i \nabla_i$

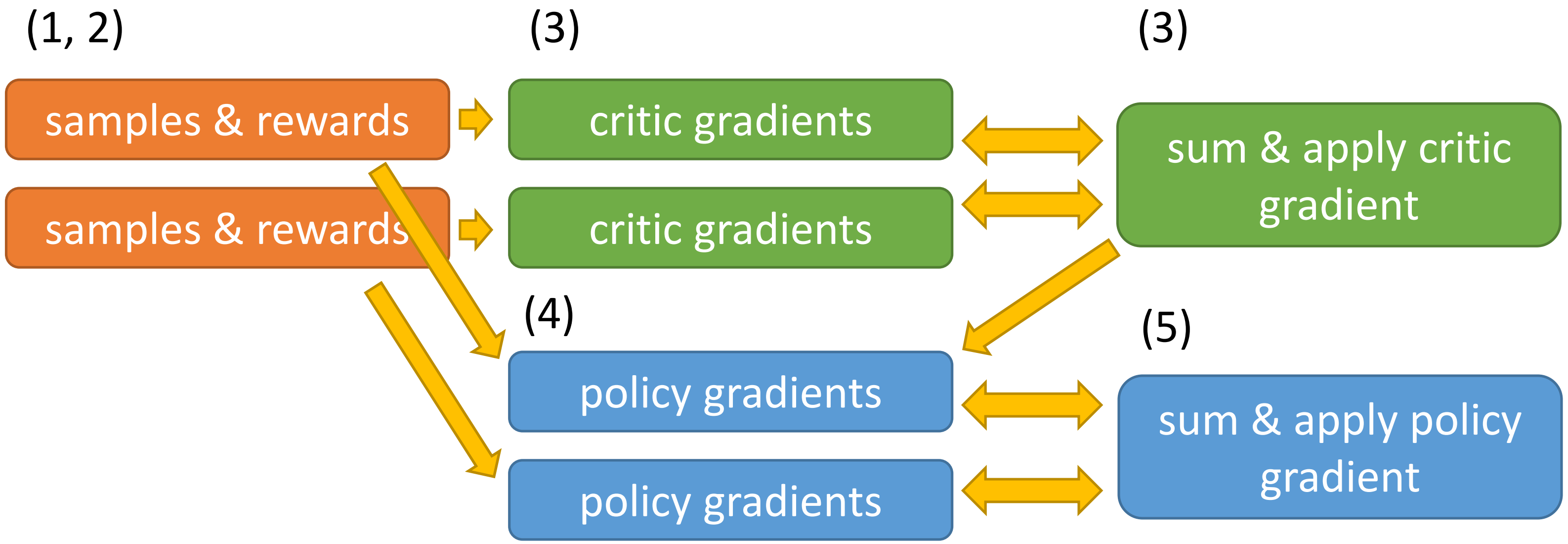
see John's actor-critic lecture for what the options here are



# What if we add a critic?

1. collect samples $\tau_i = \{\mathbf{s}_1^i, \mathbf{a}_1^i, \ldots, \mathbf{s}_T^i, \mathbf{a}_T^i\}$ by running $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ N times
2. compute $r_i = r(\tau_i)$
3. update $\hat{A}_\phi(\mathbf{s}_t^i, \mathbf{a}_t^i)$ with regression to target values
4. compute $\nabla_i = \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right) \hat{A}_\phi(\mathbf{s}_t^i, \mathbf{a}_t^i)$
5. update: $\theta \leftarrow \theta + \alpha \sum_i \nabla_i$

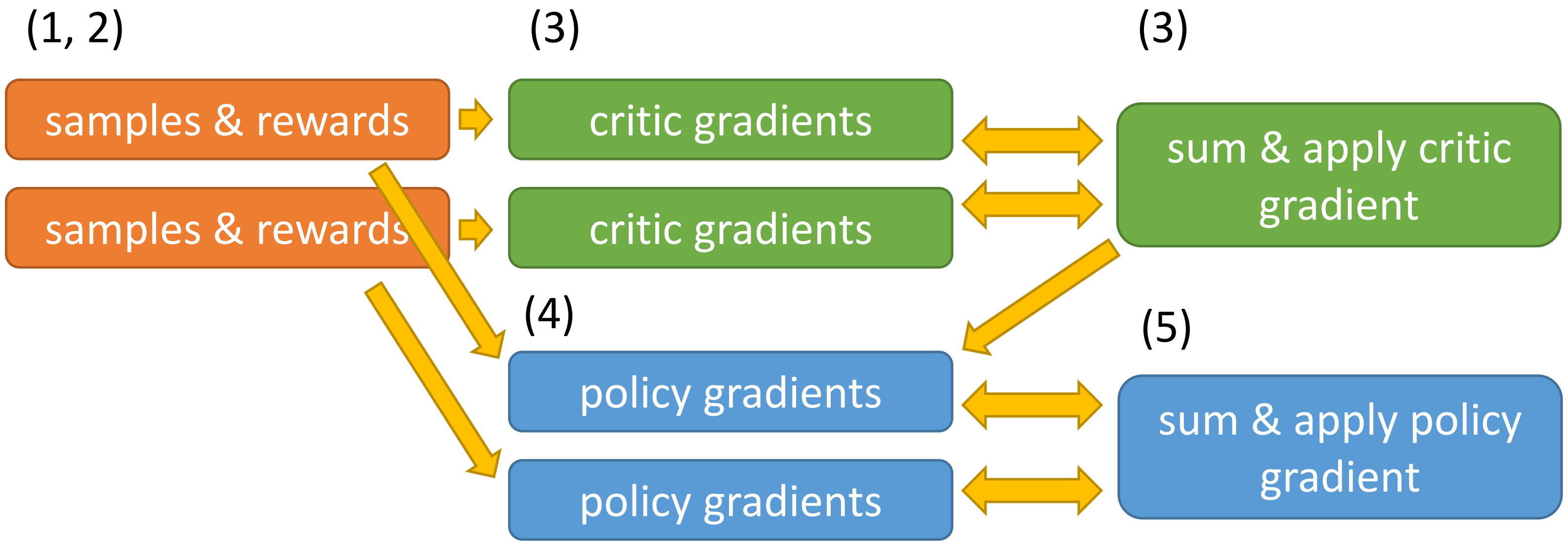
see John's actor-critic lecture for what the options here are



# What if we run online?

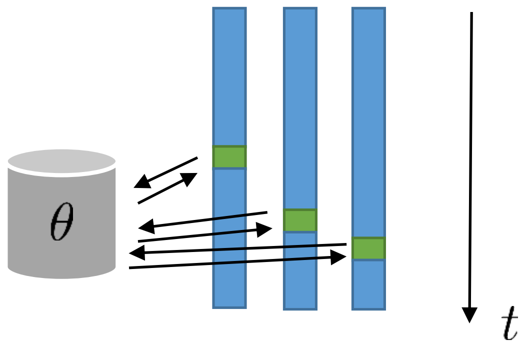
1. collect sample $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i)$ by running $\pi_\theta(\mathbf{a}|\mathbf{s})$ for 1 step
2. compute $r_i = r(\mathbf{s}_i, \mathbf{a}_i)$
3. update $\hat{A}_\phi(\mathbf{s}^i_t, \mathbf{a}^i_t)$ with regression to target values
4. compute $\nabla_i = \nabla_\theta \log \pi_\theta(\mathbf{a}^i|\mathbf{s}^i)\hat{A}_\phi(\mathbf{s}^i, \mathbf{a}^i)$
5. update: $\theta \leftarrow \theta + \alpha \sum_i \nabla_i$

only the parameter update
requires synchronization (actor + critic params)



# Actor-critic algorithm: A3C

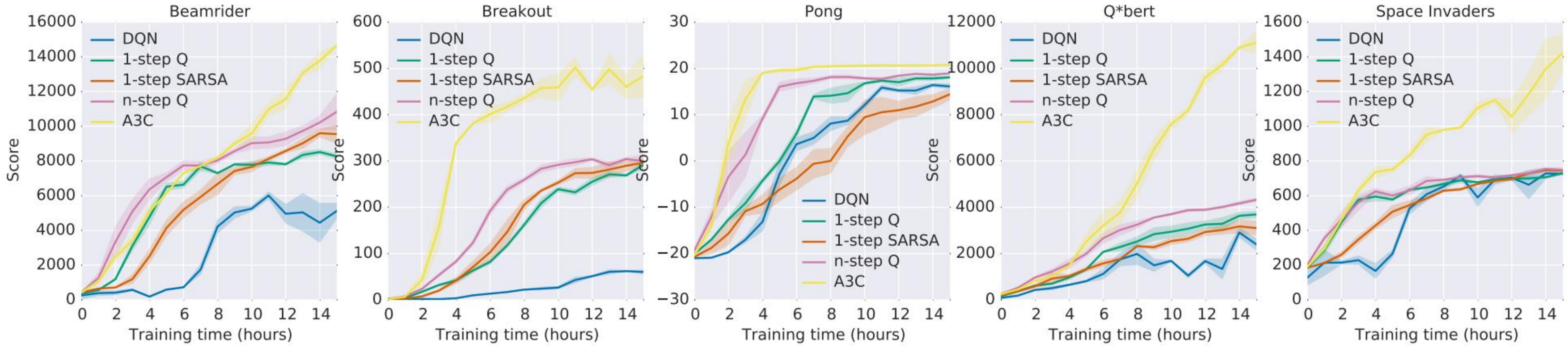
1. collect sample $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i)$ by running $\pi_\theta(\mathbf{a}|\mathbf{s})$ for 1 step
2. compute $r_i = r(\mathbf{s}_i, \mathbf{a}_i)$
3. update $\hat{A}_\phi(\mathbf{s}_t^i, \mathbf{a}_t^i)$ with regression to target values
4. compute $\nabla_i = \nabla_\theta \log \pi_\theta(\mathbf{a}^i|\mathbf{s}^i)\hat{A}_\phi(\mathbf{s}^i, \mathbf{a}^i)$
5. update: $\theta \leftarrow \theta + \alpha \sum_i \nabla_i$ (only do this every $n$ steps)

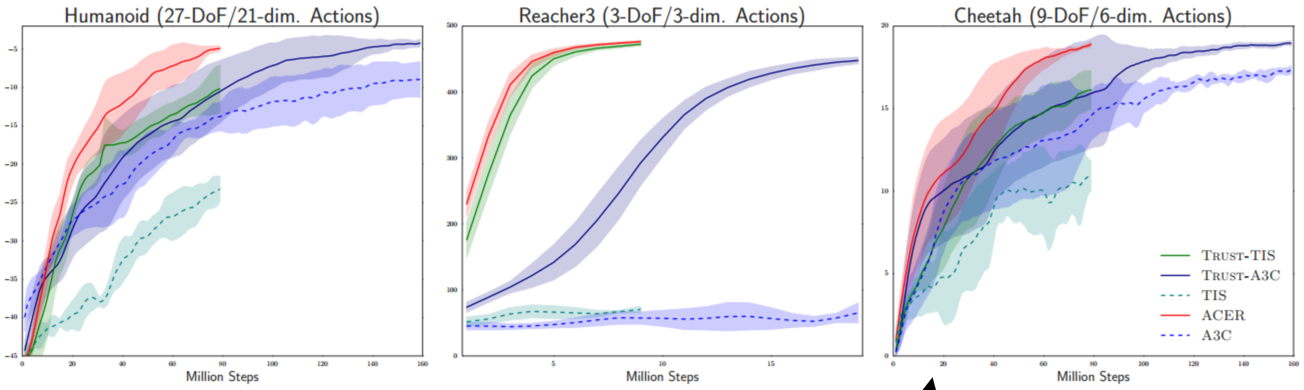



- Some differences vs DQN, DDPG, etc:
  - No replay buffer, instead rely on diversity of samples from different workers to decorrelate
  - Some variability in exploration between workers
- Pro: generally much faster in terms of wall clock
- Con: generally must slower in terms of # of samples (more on this later…)

Mnih et al. ‘16

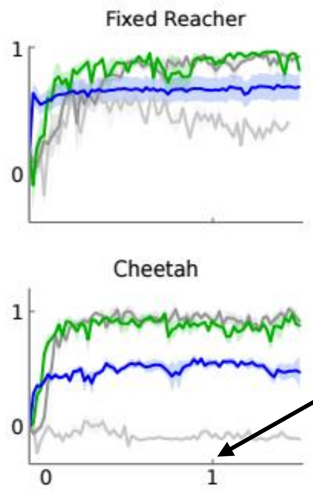
# Actor-critic algorithm: A3C




20,000,000 steps

DDPG:

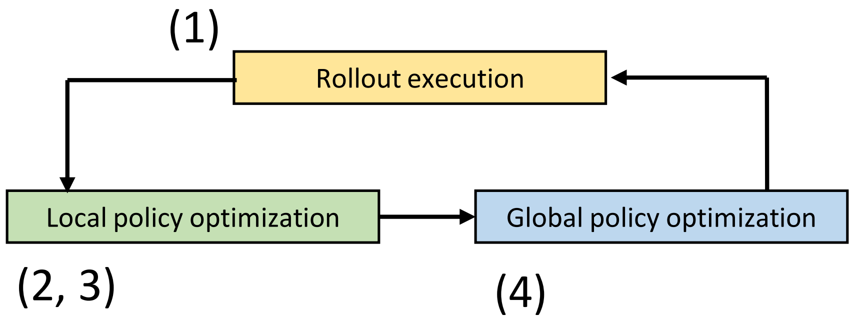


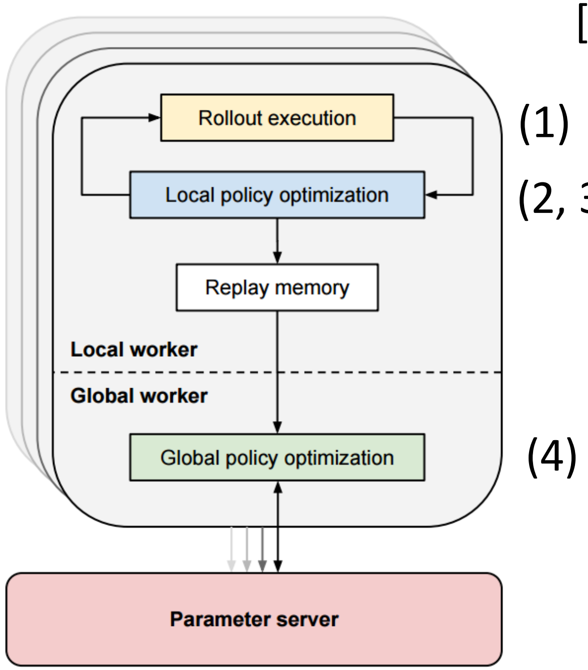

more on this later…

1,000,000 steps

# Model-based algorithms: parallel GPS

1. get $N$ samples $\tau_i$ by running $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ N times for each initial state $\mathbf{s}_0^j$ [parallelize sampling]
2. fit local models for each initial state [parallelize dynamics]
3. use LQR to get updated local policies $p_j(\mathbf{a}_t|\mathbf{s}_t)$ for each initial state $\mathbf{s}_0^j$ [parallelize LQR]
4. update policy $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ by imitating all $p_j(\mathbf{a}_t|\mathbf{s}_t)$ [parallelize SGD]

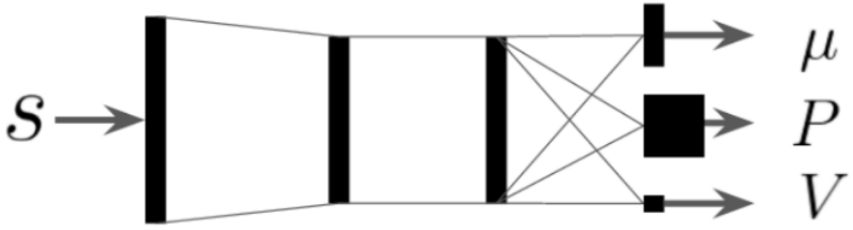



Yahya, Li, Kalakrishnan, Chebotar, L., ‘16

# Model-based algorithms: parallel GPS

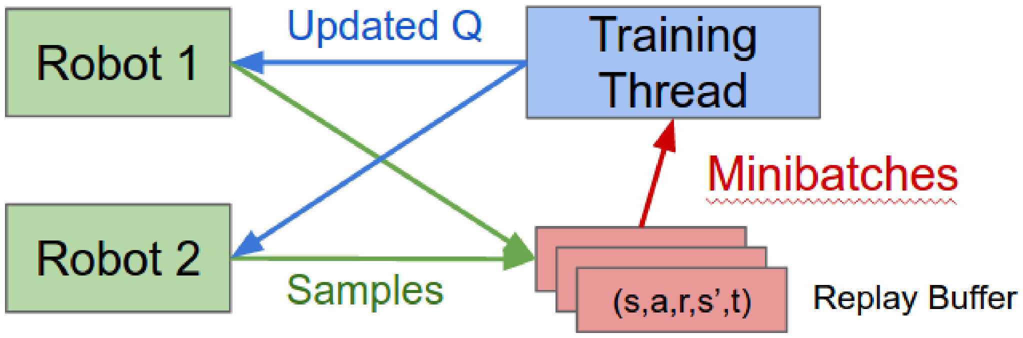
# Real-world model-free deep RL: parallel NAF

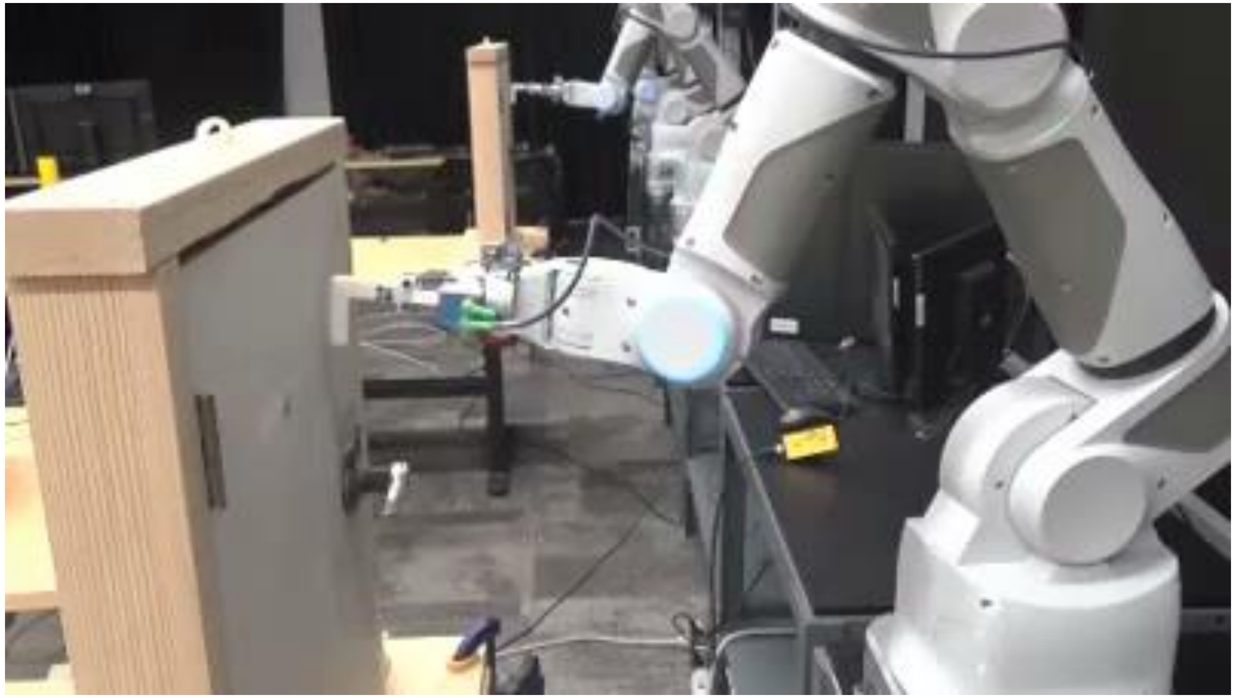



NAF Architecture.

$$Q(\boldsymbol{x},\boldsymbol{u}|\theta^Q) = A(\boldsymbol{x},\boldsymbol{u}|\theta^A) + V(\boldsymbol{x}|\theta^V)$$

$$A(\boldsymbol{x},\boldsymbol{u}|\theta^A) = -\frac{1}{2}(\boldsymbol{u}-\boldsymbol{\mu}(\boldsymbol{x}|\theta^\mu))^T \boldsymbol{P}(\boldsymbol{x}|\theta^P)(\boldsymbol{u}-\boldsymbol{\mu}(\boldsymbol{x}|\theta^\mu))$$



Gu*, Holly*, Lillicrap, L., '16

# Simplest example: sample parallelism with off-policy algorithms