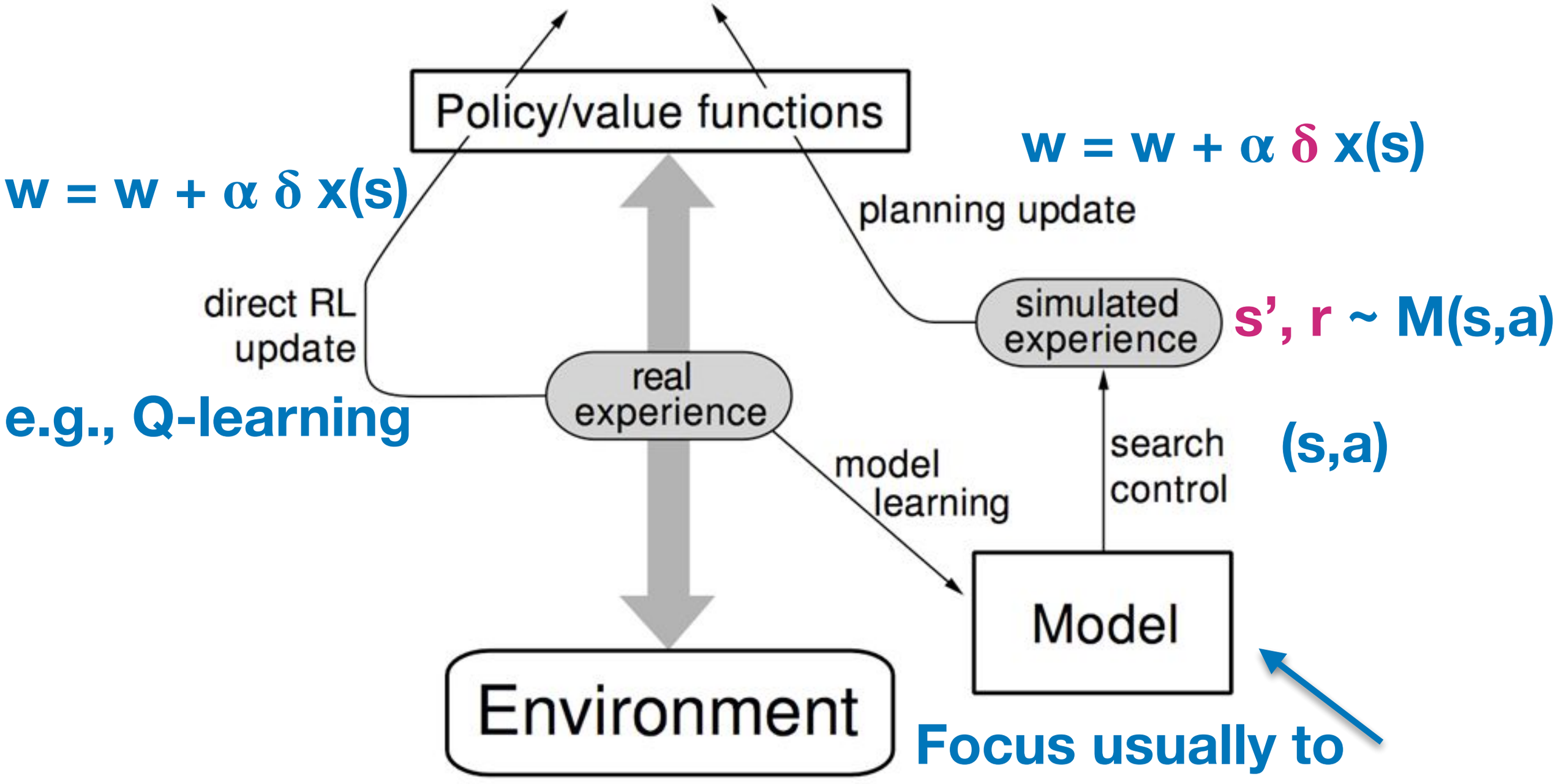
Planning with Learned Models in Dyna

Martha White
Assistant Professor
University of Alberta

What is Dyna?



R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction

High-level take-away

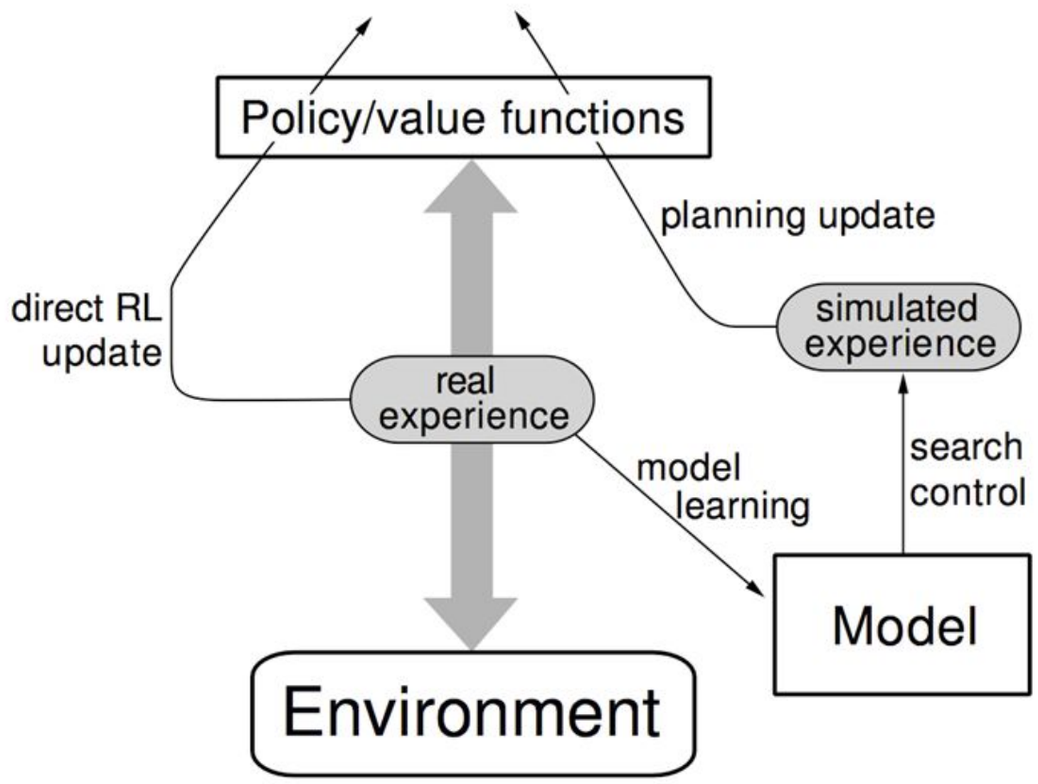
- Planning with **learned models** in **Dyna** can improve **data efficiency**
- **Key insight**: the choice of what states to sample from (search-control) is critical to get improvements

Planning with learned models

- **Trajectory-based** approaches: generate rollouts for
  - Monte carlo planning, e.g., Dagger (Ross and Bagnell), Hallucinated Dagger (Talvitie)
  - On-policy planning, e.g. NAF (Gu et al.)
  - Trajectory optimization, e.g., control with LDS (Levine & Abbeel)
- **Approximate value iteration** with factored models (stochastic factorization, kernel mean embeddings,…)

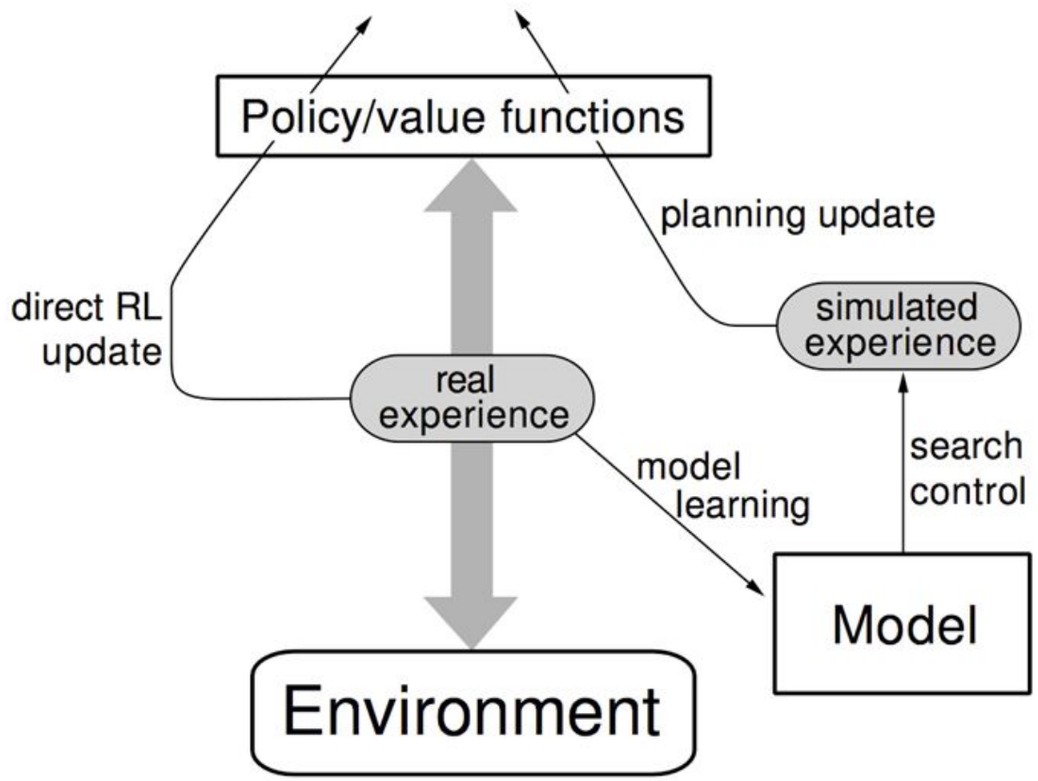
Advantages of Dyna

- **Interleaves** learning and planning
  - agent interacts with the world at a given timescale
  - a model can be simulated in the background to improve action-value estimates



Advantages of Dyna

- Naturally enables **incomplete models**
  - action-values are still learned and direct behaviour
  - model can be used in any way to try to improve action-value estimates



Advantages of Dyna

- **Avoids** multi-step **rollouts**
  - iterated predictions compound error; can produce implausible states
  - can do long-term planning use temporal abstraction (option models)

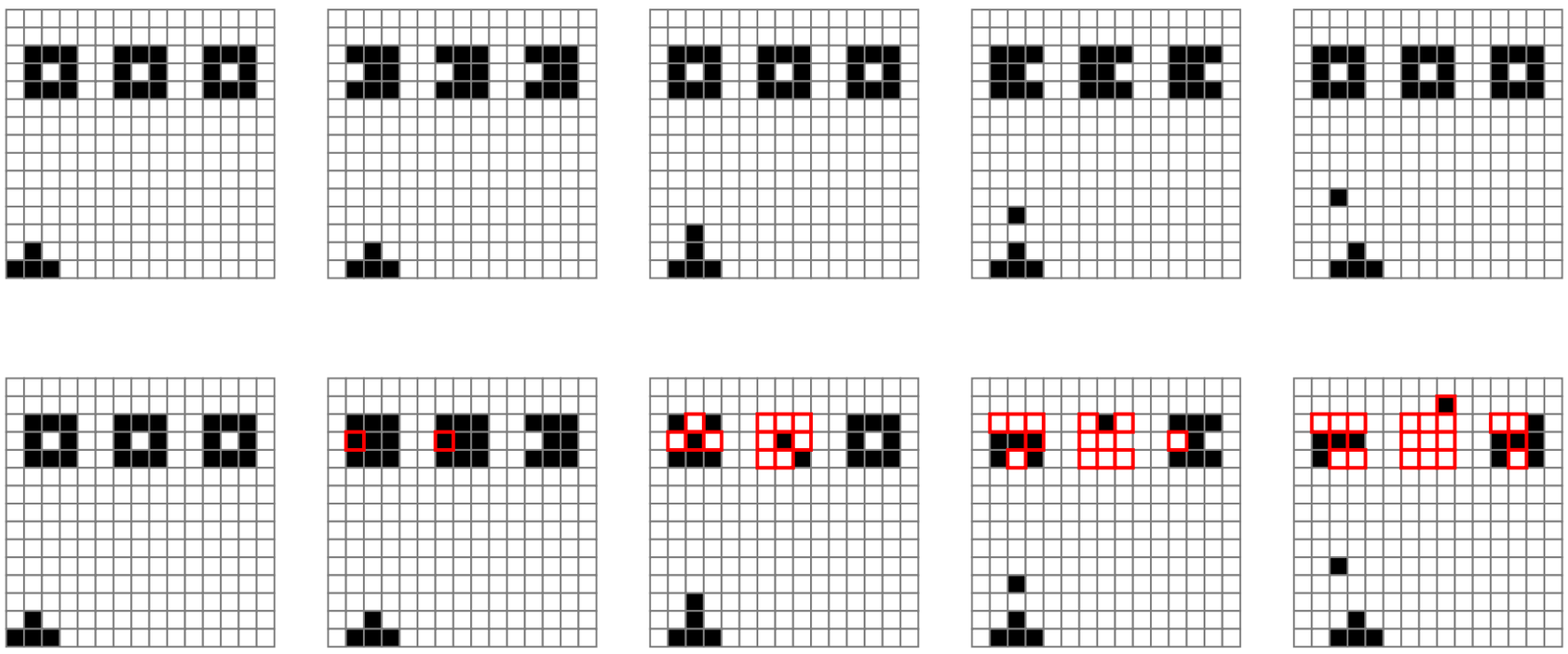
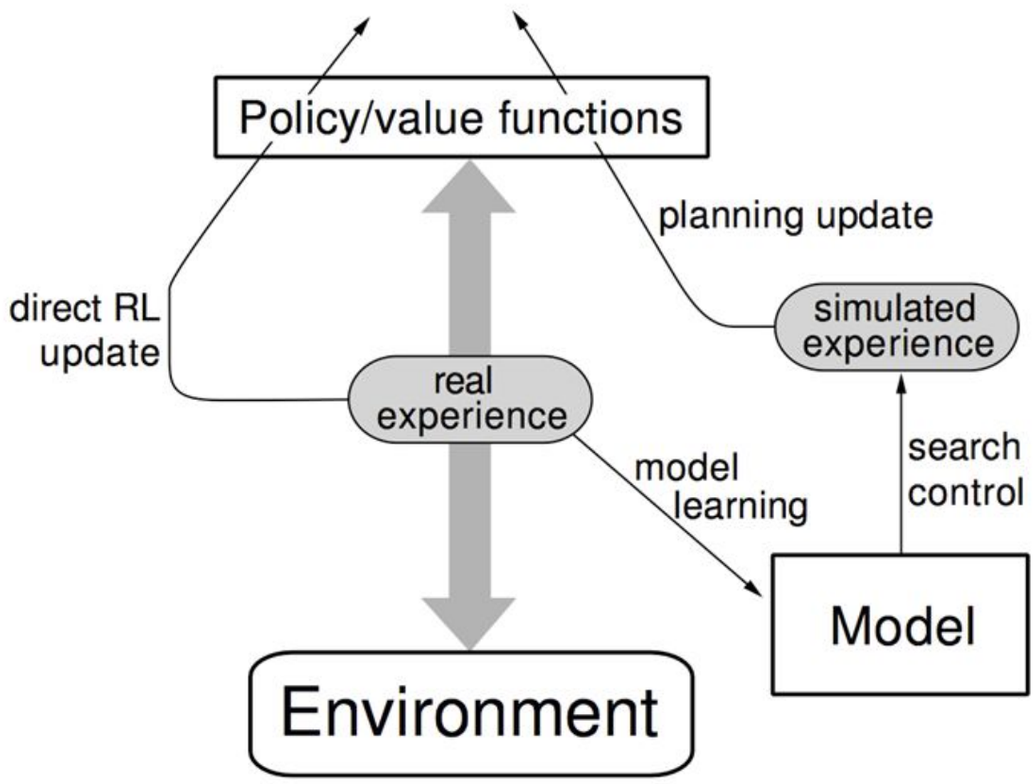


***image used with permission from Erik Talvitie**

Advantages of Dyna

- Only requires **conditional sample-based** model
  - do not need to explicitly estimate densities
  - can take advantage of advances in generative models
  - only need P(s, r, gamma | s, a), simplifying estimation



Advantages of Dyna

- **Interleaves** learning and planning
- Naturally enables **partial models**
- **Avoids** multi-step **rollouts**
- Can still do **long-term planning** use temporal abstraction
- Only requires **conditional sample-based** model

Why aren’t we using Dyna?

- We sort of are with **experience replay**
  - “simulate” transitions from (recent) experience to update action-values
- **Buffer of transitions**, instead of search-control queue
  - replay entire transition, rather than sampling from M(s,a)
- …but does that matter? Can we gain something from learning an explicit conditional model? Is it worth the potential bias in the model?

Advantages from a learned model vs transition buffer

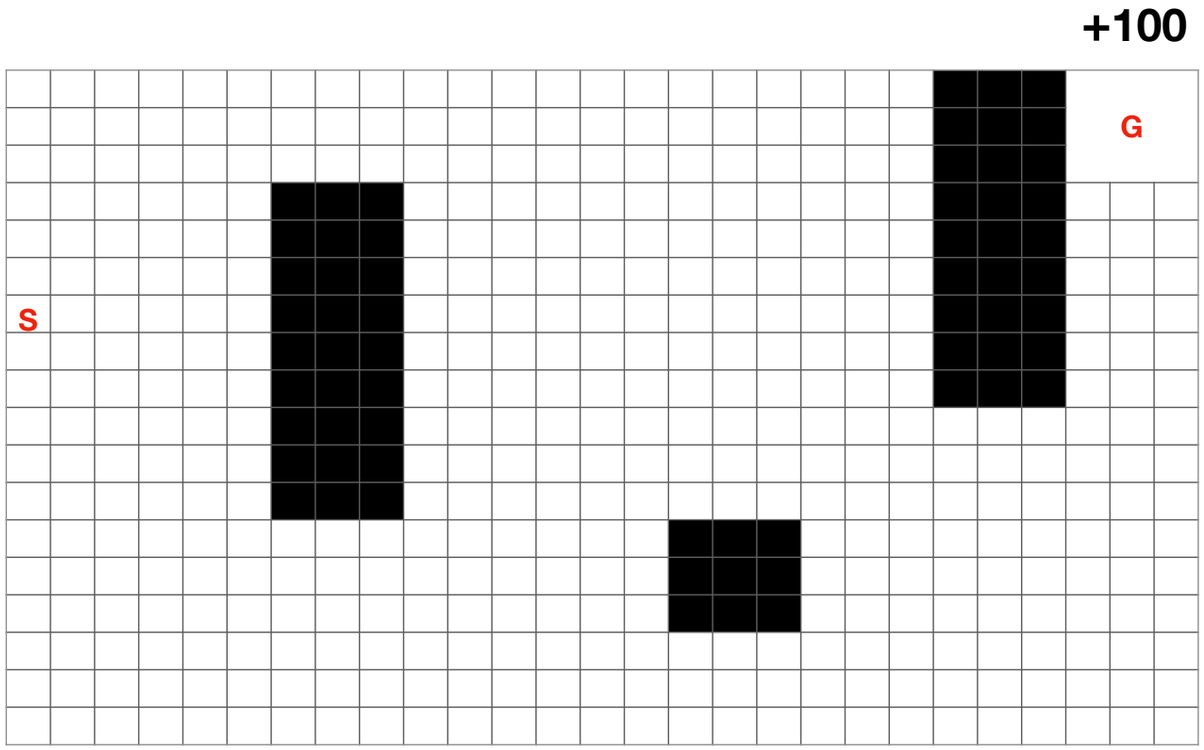
- **Compactness**: summarizes experience
- **Coverage**: cannot store all experience, so in ER common to use most recent experience (does not cover space)
- **Querying**: can query a model from particular (s,a)

Benefits of explicitly querying a model

- **Predecessors**: given a high-priority state, sample predecessor states that led to it
  - need model P(s | s', a)
- **On-policy transitions**: given a high-priority state, sample the outcome for the action a taken by the current policy
  - ER can only replay what old policy did

Experiments in Gridworlds

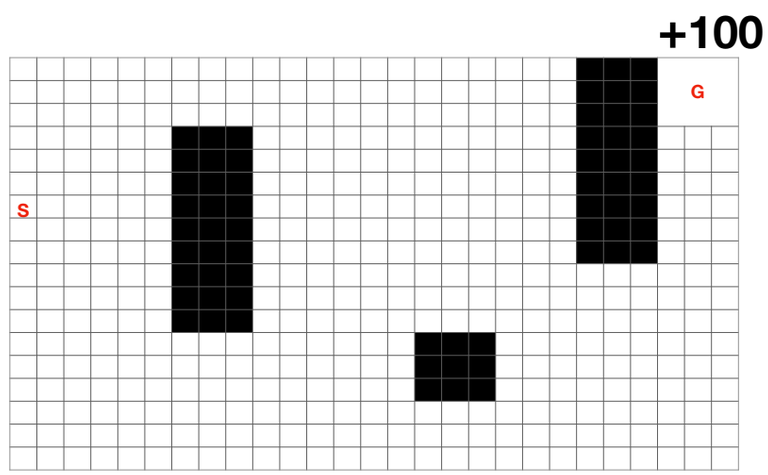
- Tabular and Continuous
- Deterministic and stochastic
- Q-learning, epsilon = 0.1
- Number planning steps = 5
- Buffer/queue size = 1024, oldest samples deleted



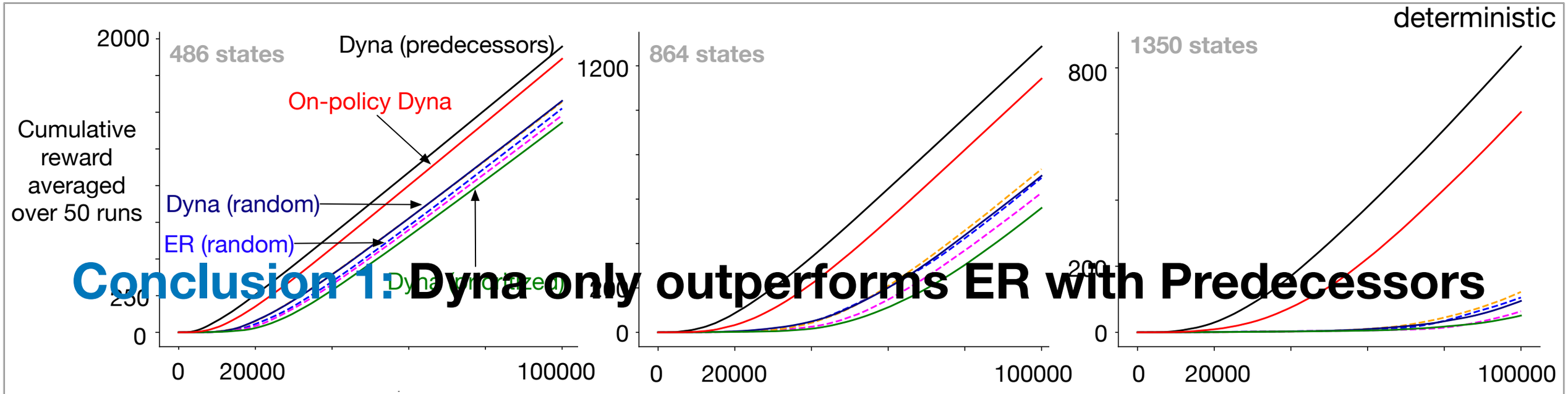
Sampling strategy

- Random
- Prioritized
- Predecessors

Experiments in a Tabular Gridworld



deterministic

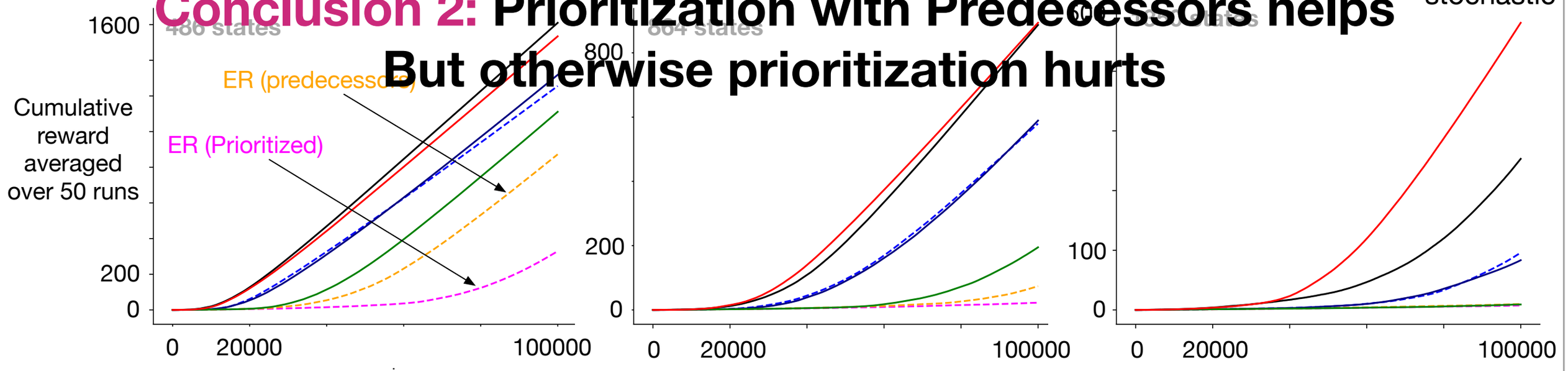


Conclusion 1: Dyna only outperforms ER with Predecessors

Conclusion 2: Prioritization with Predecessors helps

But otherwise prioritization hurts

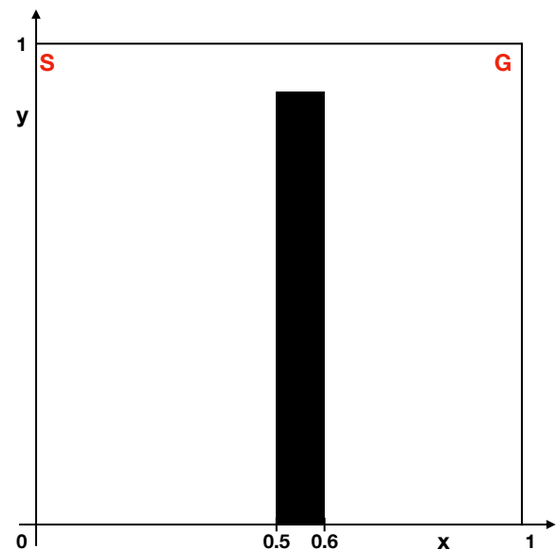
stochastic

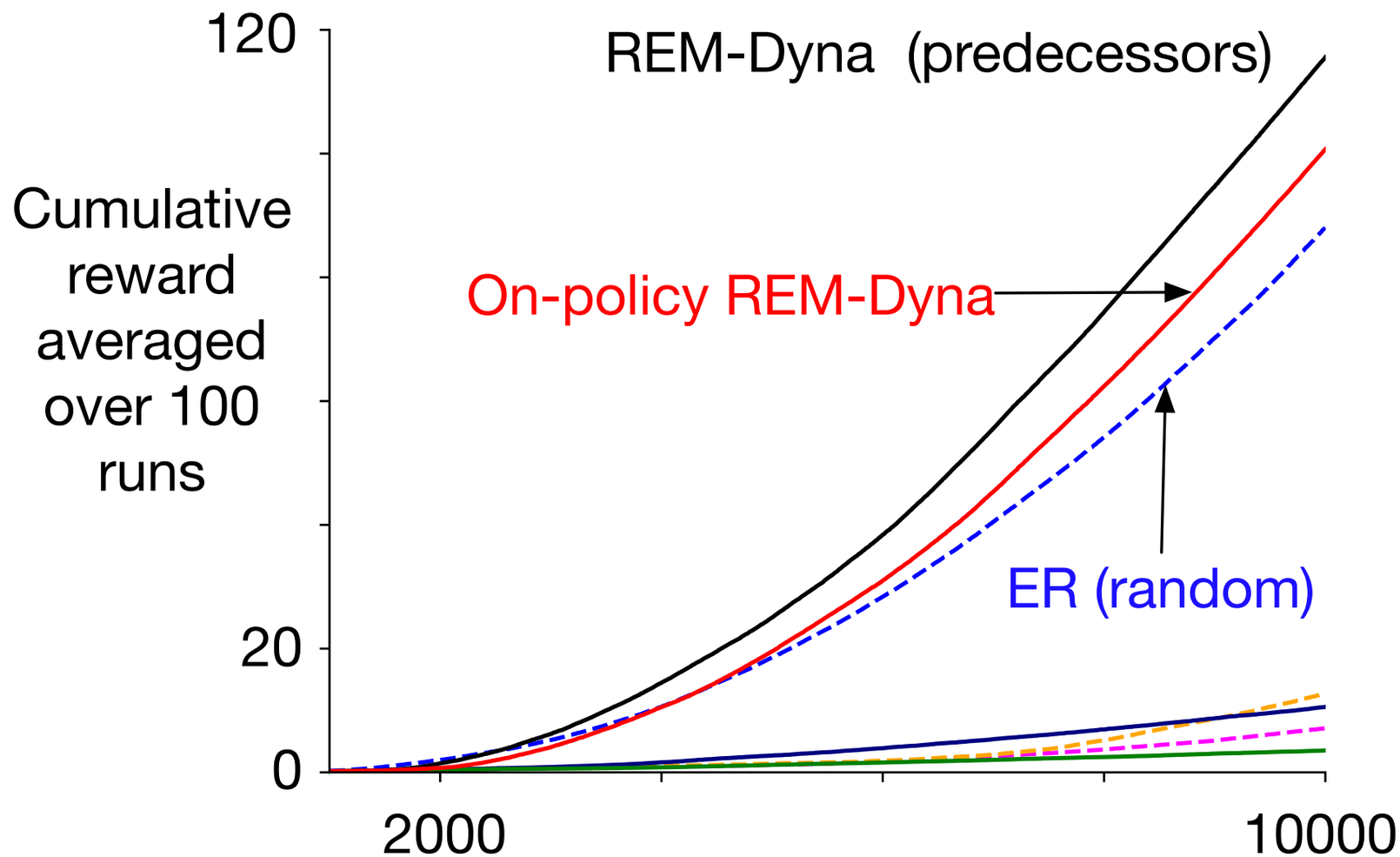


Number planning steps = 5

Random action with 10%

Experiments in a Continuous Gridworld





What are the desired properties for learned models in Dyna?

- **Online** — incrementally learned and adaptive
- **Data-efficient** — to feasibly improve data-efficiency of learning value functions
- **Robust to forgetting** — non-stationarity policies and localized sampling can cause forgetting if not careful
- **Computationally efficient sampling** — a slow sampler will reduce the feasible number of planning steps

Memory-based models

- Consider Kernel Density Estimation, given T samples

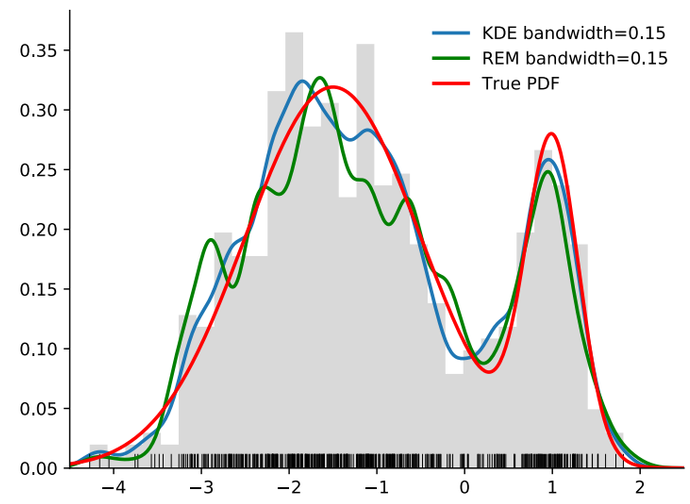
$$p(\mathbf{s}, a, \mathbf{s}', r, \gamma) = \frac{1}{T} \sum_{i=1}^{T} k((\mathbf{s}, a, \mathbf{s}', r, \gamma), (\mathbf{s}_i, a_i, \mathbf{s}'_i, r_i, \gamma_i))$$

- Conditional distribution, using product kernel

$$p(\mathbf{s}', r, \gamma | \mathbf{s}, a) = \frac{1}{N(\mathbf{s}, a)} \sum_{i=1}^{T} k_{\mathbf{s}}(\mathbf{s}, \mathbf{s}_i) k_a(a, a_i) k_{\mathbf{s}', r, \gamma}((\mathbf{s}', r, \gamma), (\mathbf{s}'_i, r_i, \gamma_i))$$

$$N(\mathbf{s}, a) = \frac{1}{T} \sum_{i=1}^{T} k_{\mathbf{s}}(\mathbf{s}, \mathbf{s}_i) k_a(a, a_i)$$

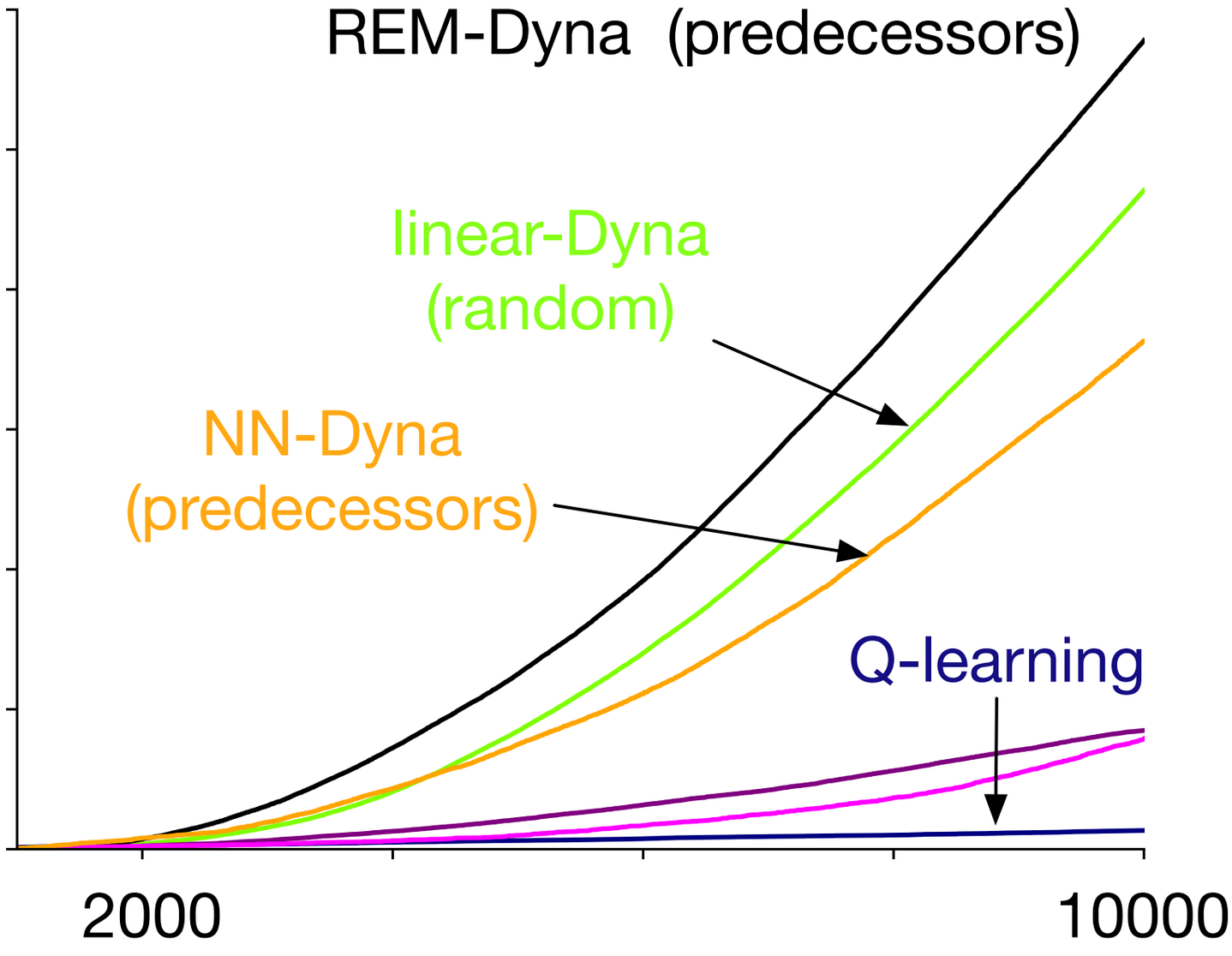
- Issue: Cannot store all data



REM: Reweighted Experience Models

- Selects representative **prototypes** from experience
- Take advantage of the fact that only need to sample from conditional distributions P(s, r, gamma | s, a) or P(s | s', a)
  - conditional **reweighting** can be obtained incrementally
- **Semi-parameteric**: only needs to learn reweightings and subselect prototypes, so learns quickly

Results compared to other models



Conclusion: NN model learns too slowly

Open questions

- Do we observe the same empirical phenomena (e.g., importance of predecessors) in other domains?
- Can we improve training of DQN, using Dyna with a learned model instead of ER?
- What generative models are the most effective within Dyna?
- What is the convergence rate and fixed point, using learned models in Dyna?

Thank you! Questions?