

# Transfer and Multi-Task Learning

CS 294-112: Deep Reinforcement Learning

Sergey Levine

# Class Notes

1. Two weeks until the project milestone!

# How can we frame transfer learning problems?

**No single solution! Survey of various recent research papers**

1. “Forward” transfer: train on one task, transfer to a new task
  - a) Just try it and hope for the best
  - b) Architectures for transfer: progressive networks
  - c) Finetune on the new task
  - d) Randomize source task domain
2. Multi-task transfer: train on many tasks, transfer to a new task
  - a) Model-based reinforcement learning
  - b) Model distillation
  - c) Contextual policies
  - d) Modular policy networks
3. Multi-task meta-learning: learn to learn from many tasks
  - a) RNN-based meta-learning
  - b) Gradient-based meta-learning

# How can we frame transfer learning problems?

1. “Forward” transfer: train on one task, transfer to a new task
  - a) Just try it and hope for the best
  - b) Architectures for transfer: progressive networks
  - c) Finetune on the new task
  - d) Randomize source task domain
2. Multi-task transfer: train on many tasks, transfer to a new task
  - a) Model-based reinforcement learning
  - b) Model distillation
  - c) Contextual policies
  - d) Modular policy networks
3. Multi-task meta-learning: learn to learn from many tasks
  - a) RNN-based meta-learning
  - b) Gradient-based meta-learning

# Finetuning summary

- Try and hope for the best
  - Sometimes there is enough variability during training to generalize
- Finetuning
  - A few issues with finetuning in RL
  - Maximum entropy training can help
- Architectures for finetuning: progressive networks
  - Addresses some overfitting and expressivity problems by construction

# What if we can manipulate the source domain?

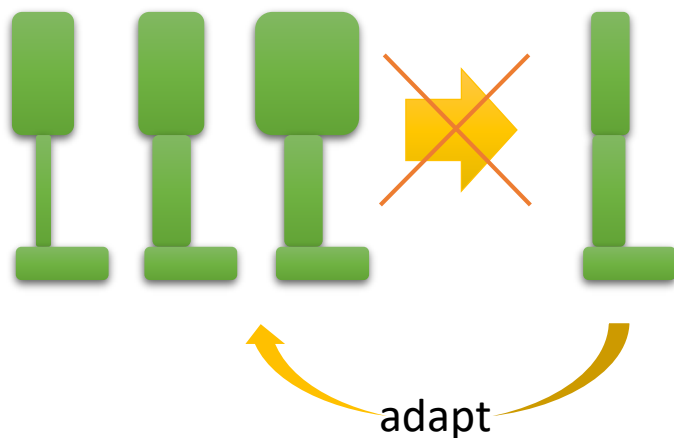
- So far: source domain (e.g., empty room) and target domain (e.g., corridor) are fixed
- What if we can **design** the source domain, and we have a **difficult** target domain?
  - Often the case for simulation to real world transfer
- Same idea: the more diversity we see at training time, the better we will transfer!

# EPOpt: randomizing physical parameters

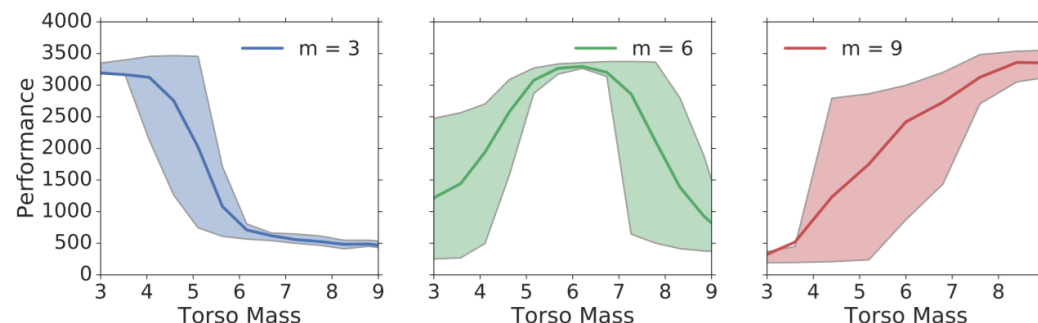


train

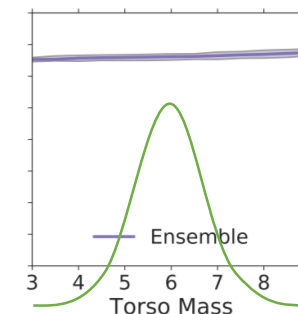
test



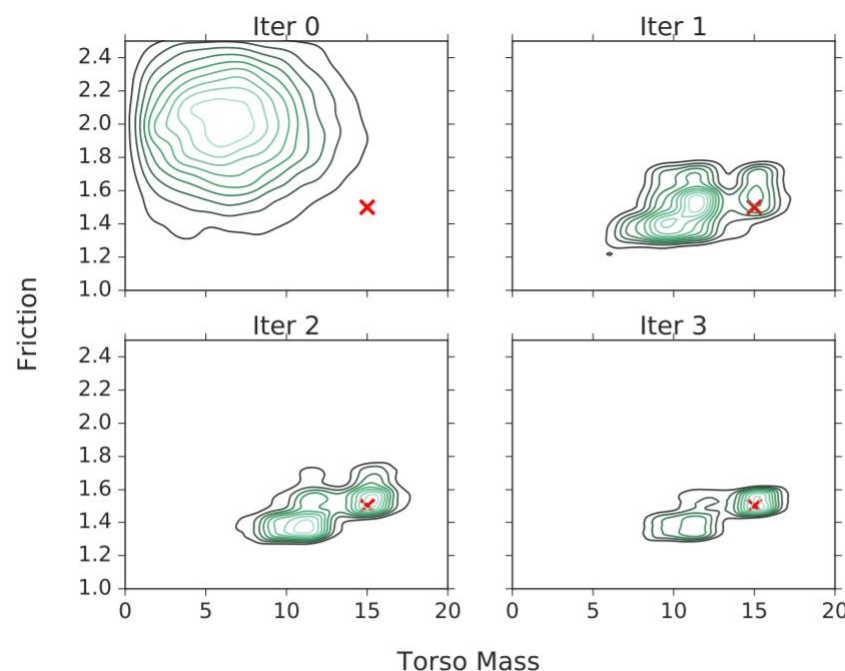
training on single torso mass



training on model ensemble

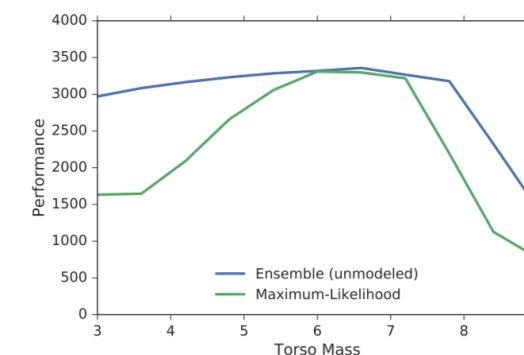


ensemble adaptation

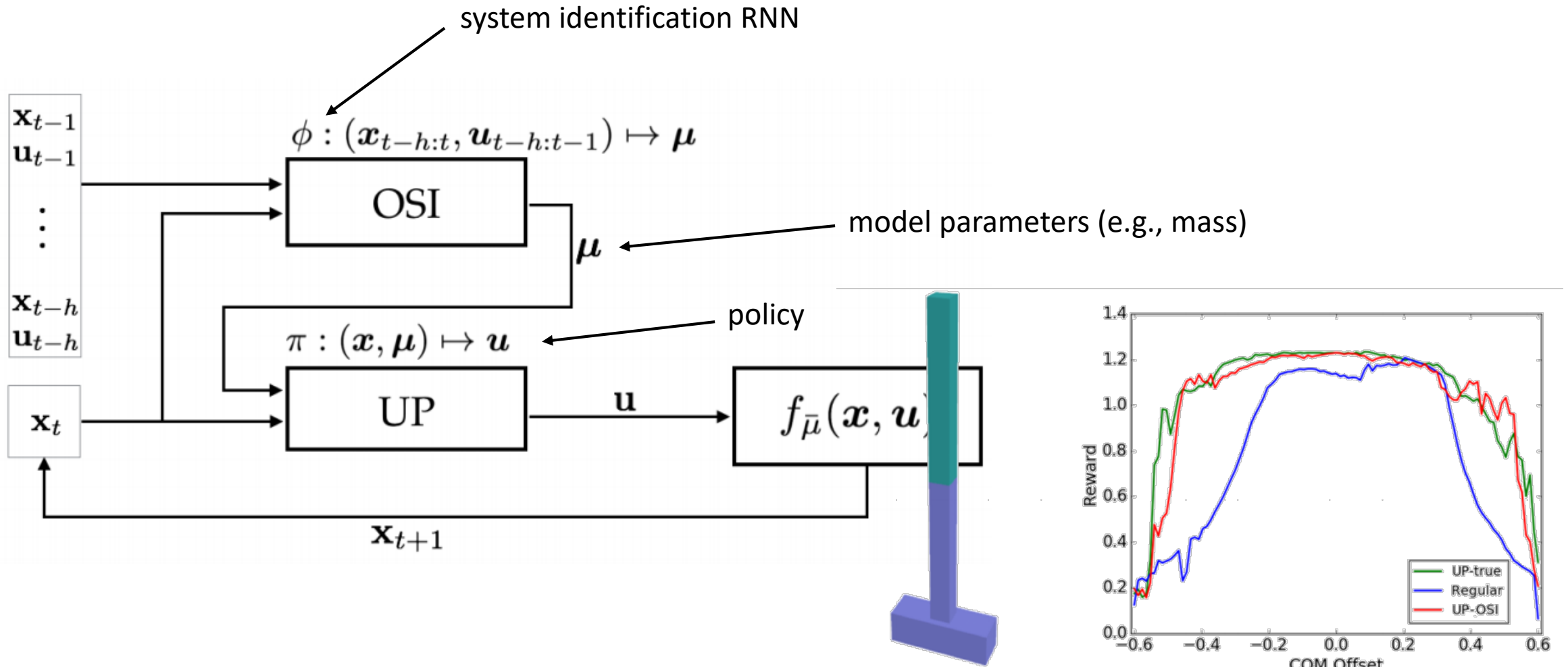


unmodeled effects

<b>Hopper</b>	$\mu$	$\sigma$	low	high
mass	6.0	1.5	3.0	9.0
ground friction	2.0	0.25	1.5	2.5
joint damping	2.5	1.0	1.0	4.0
armature	1.0	0.25	0.5	1.5
<b>Half-Cheetah</b>	$\mu$	$\sigma$	low	high
mass	6.0	1.5	3.0	9.0
ground friction	0.5	0.1	0.3	0.7
joint damping	1.5	0.5	0.5	2.5
armature	0.125	0.04	0.05	0.2

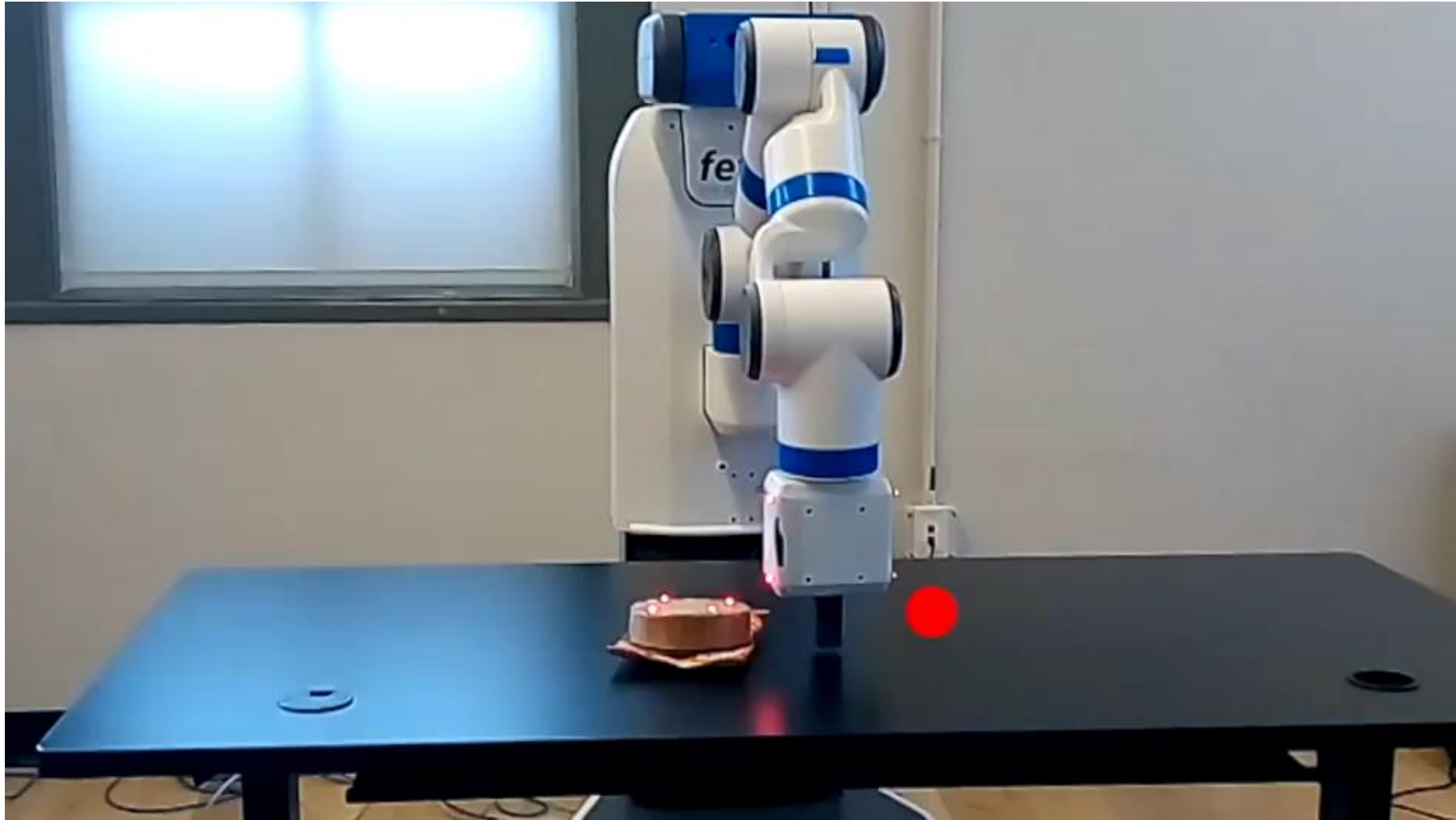


# Preparing for the unknown: explicit system ID

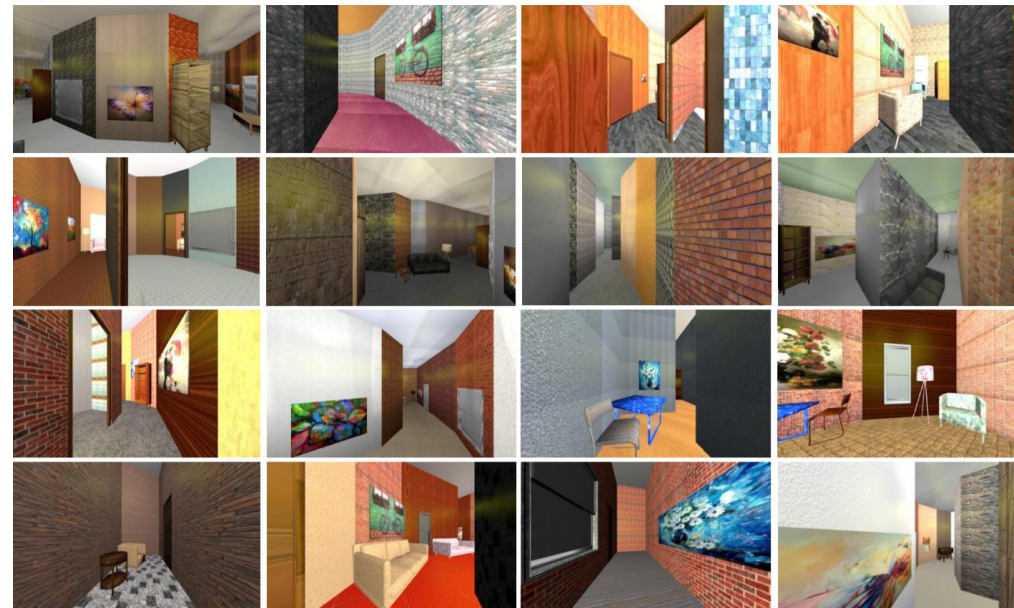




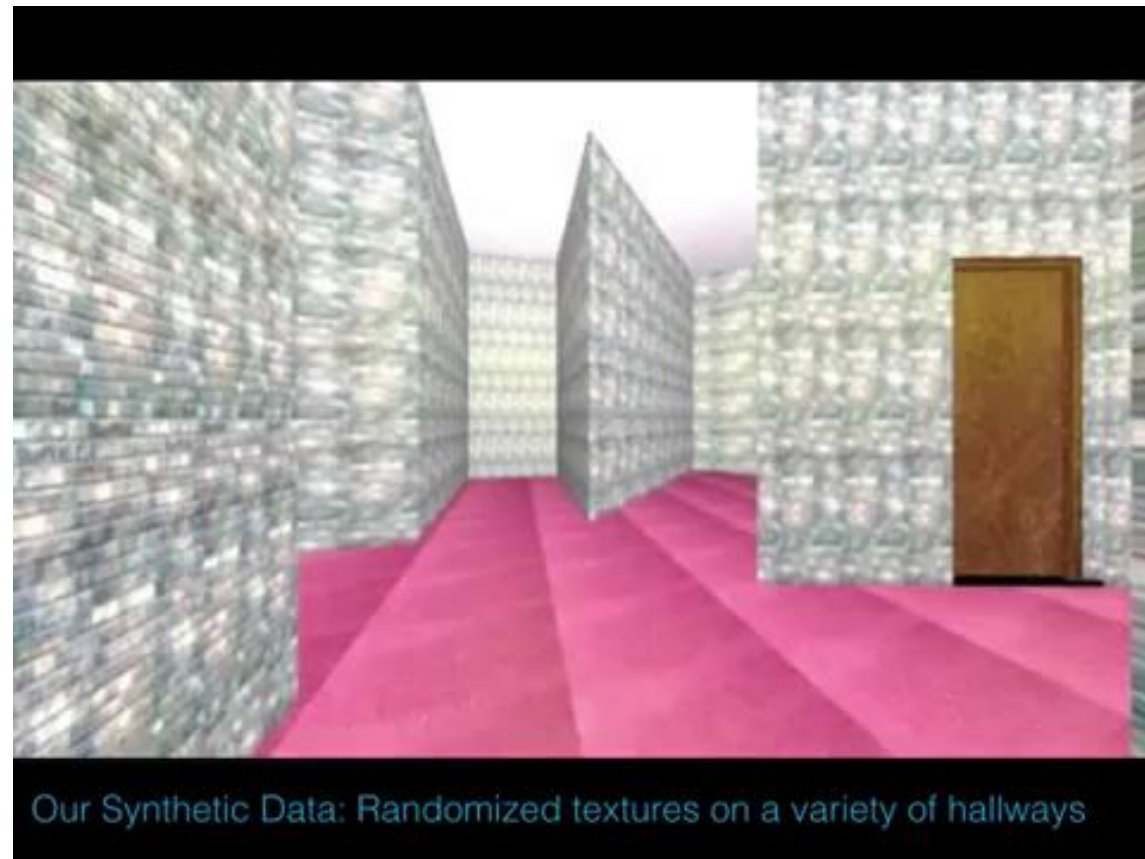
(Very) recent work

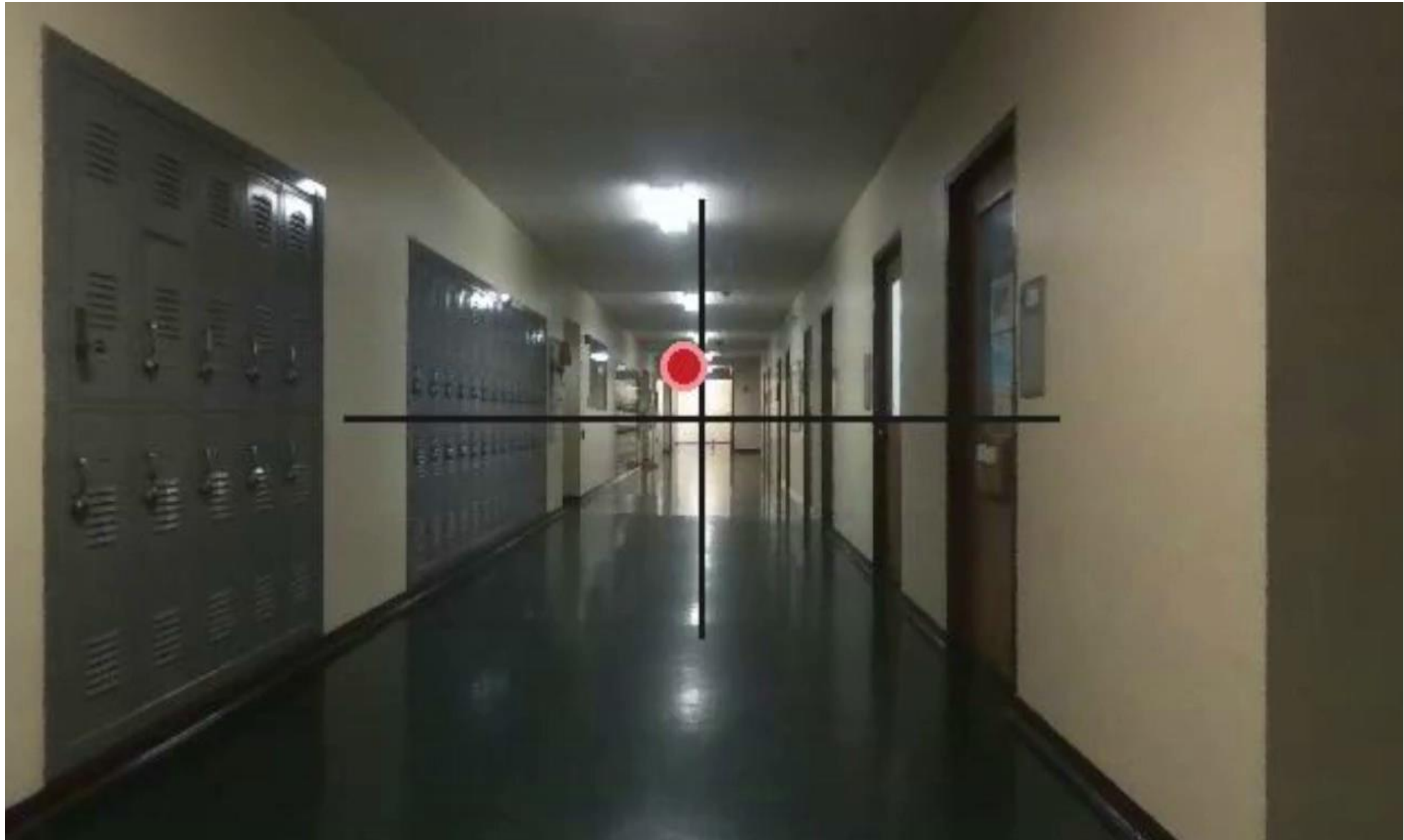


# CAD2RL: randomization for real-world control



# CAD2RL: randomization for real-world control

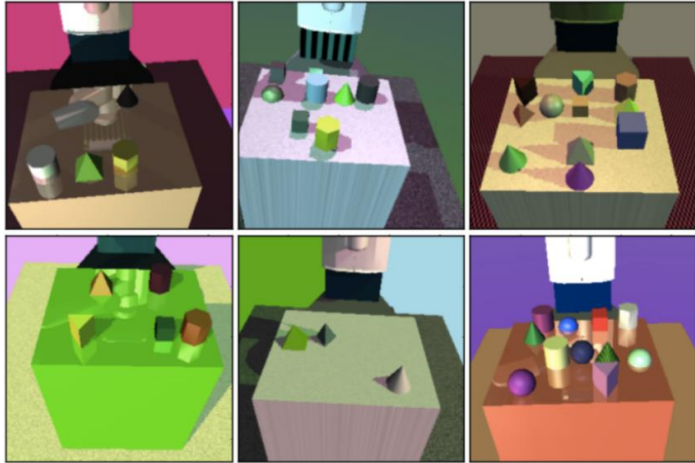




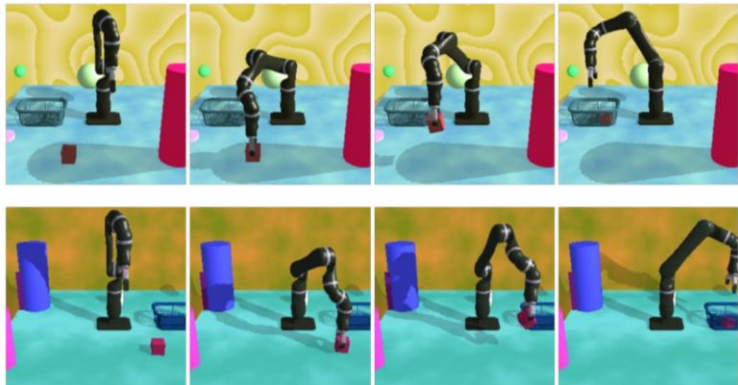
Sadeghi et al., "CAD2RL: Real Single-Image Flight without a Single Real Image"



# Randomization for manipulation



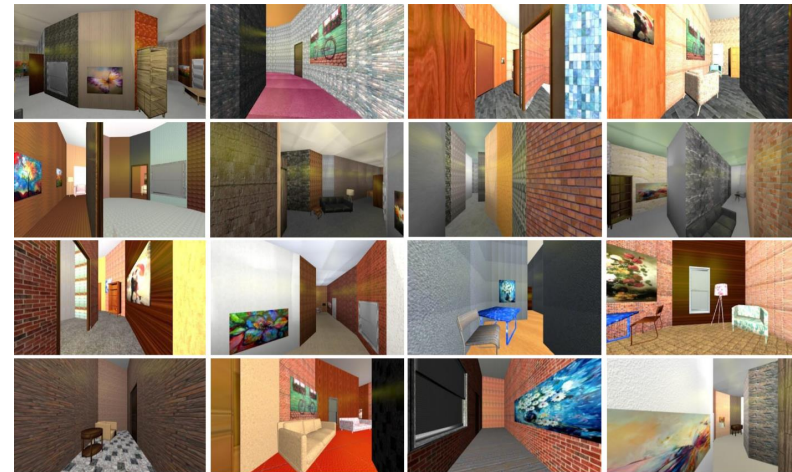
Tobin, Fong, Ray, Schneider, Zaremba, Abbeel



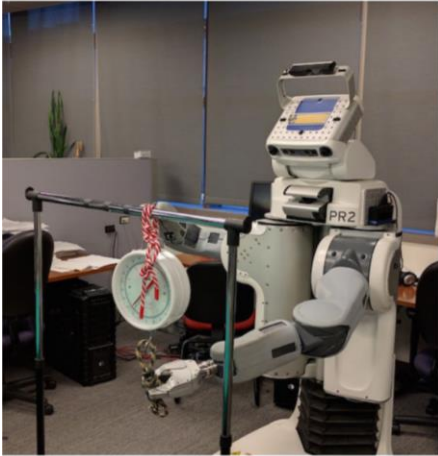
James, Davison, Johns

# What if we can peek at the target domain?

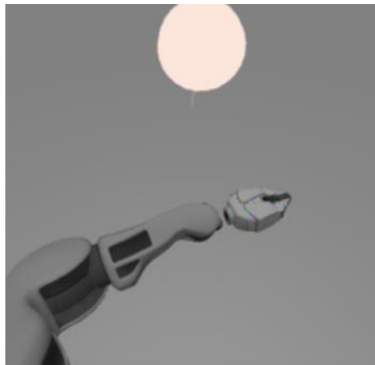
- So far: pure 0-shot transfer: learn in source domain so that we can succeed in **unknown** target domain
- Not possible in general: if we know nothing about the target domain, the best we can do is be as robust as possible
- What if we saw a few images of the target domain?



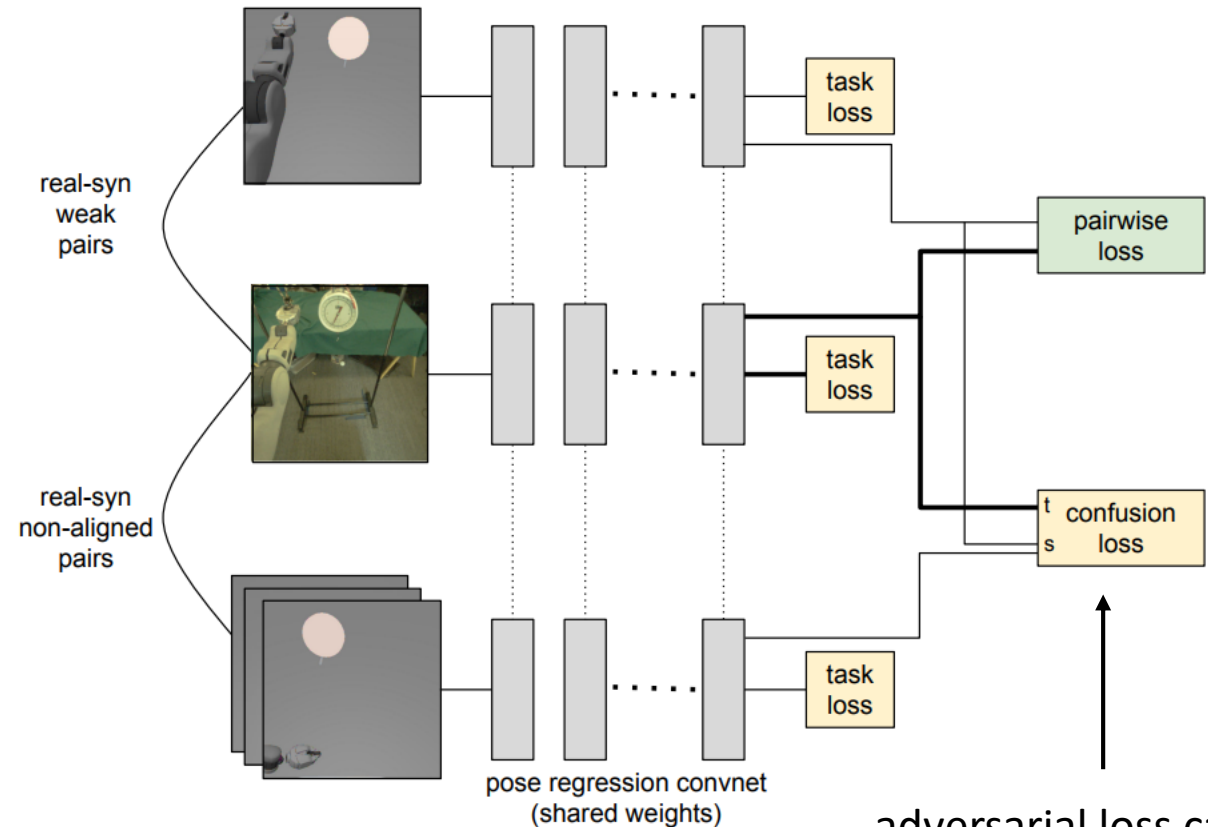
# Better transfer through domain adaptation



simulated images



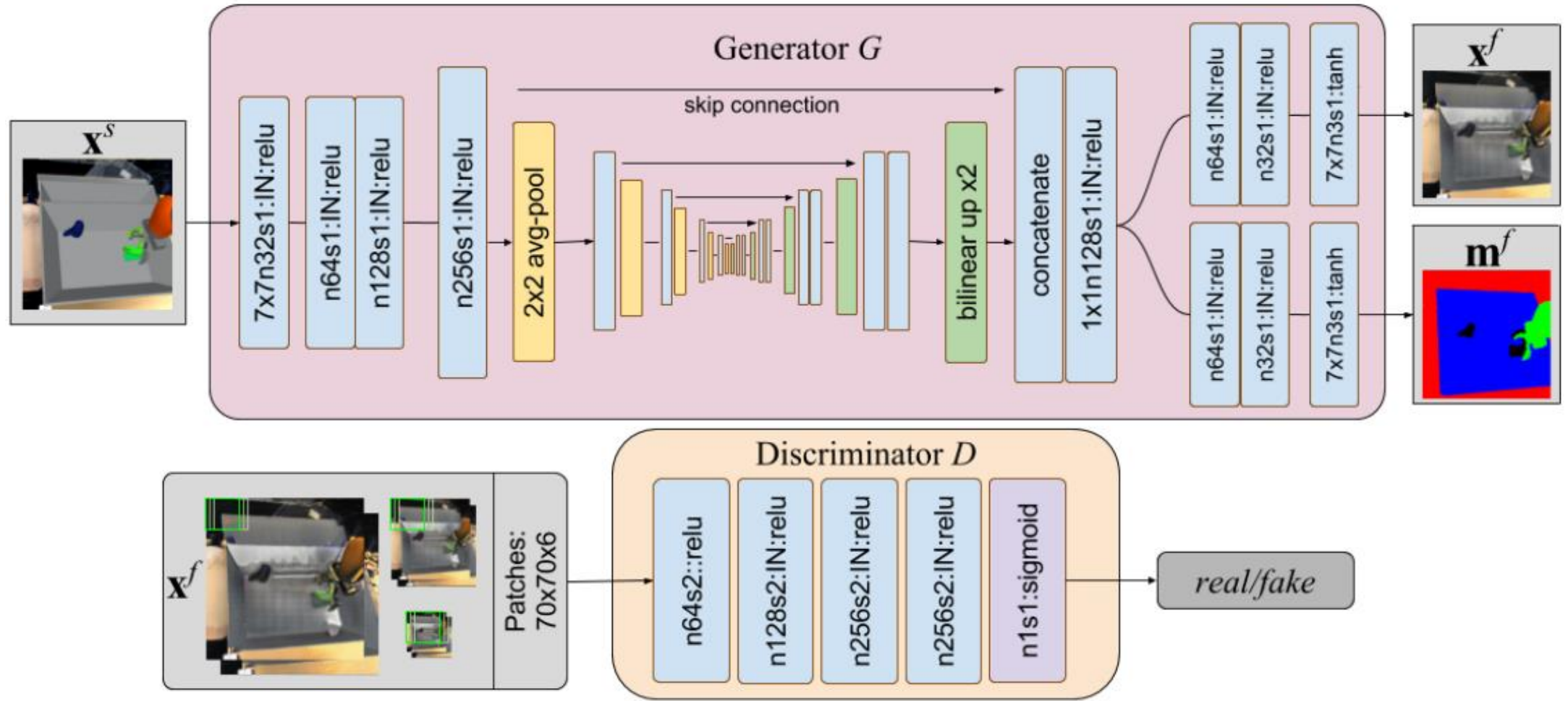
real images



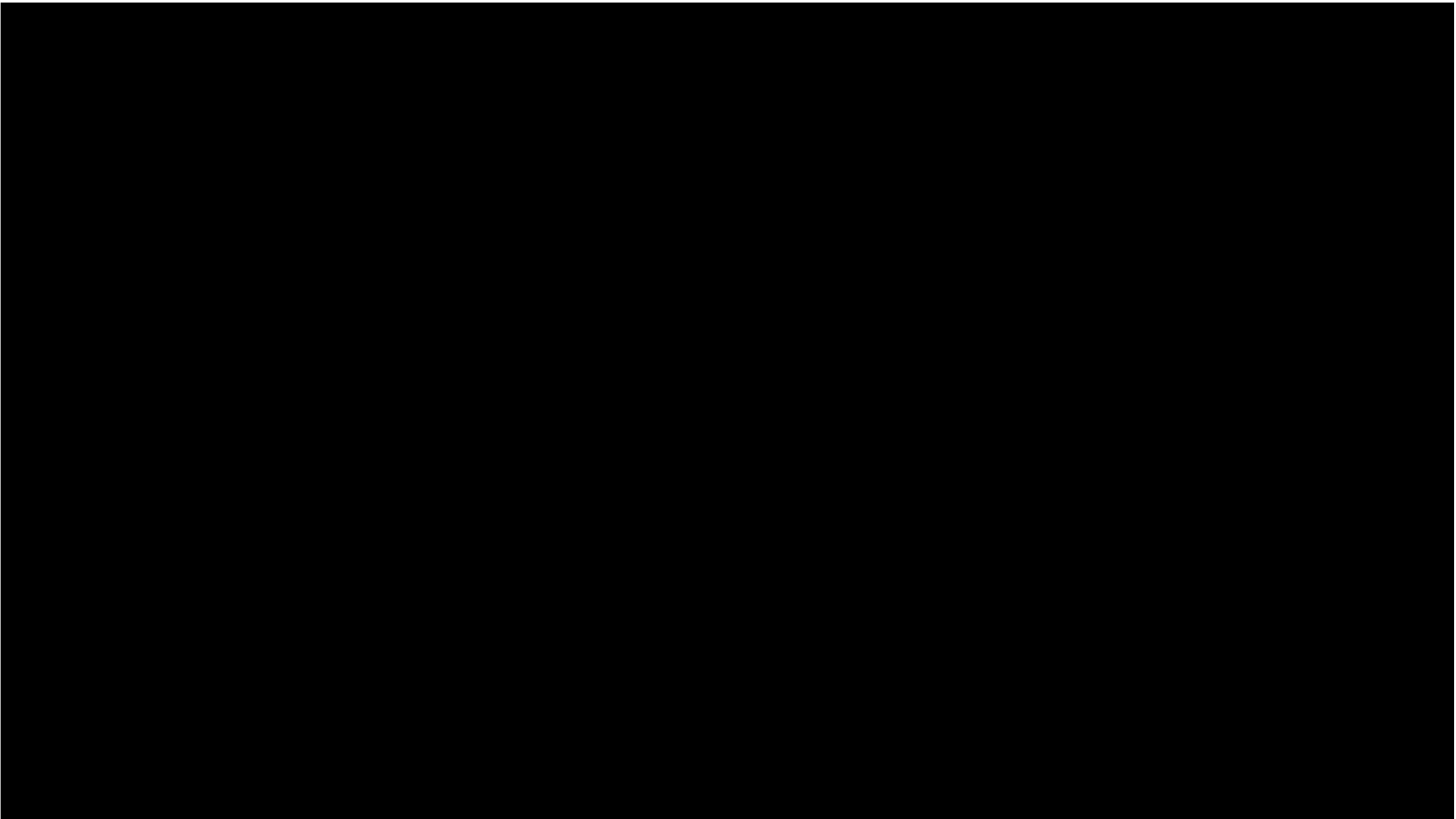
adversarial loss causes  
internal CNN features to be  
indistinguishable for sim and real

# Domain adaptation at the pixel level

can we *learn* to turn synthetic images into *realistic* ones?







# Forward transfer summary

- Pretraining and finetuning
  - Standard finetuning with RL is hard
  - Maximum entropy formulation can help
- How can we modify the source domain for transfer?
  - Randomization can help a lot: the more diverse the better!
- How can we use modest amounts of target domain data?
  - Domain adaptation: make the network unable to distinguish observations from the two domains
  - ...or modify the source domain observations to look like target domain
  - Only provides **invariance** – assumes all differences are functionally irrelevant; this is not always enough!

# Forward transfer suggested readings

Haarnoja\*, Tang\*, Abbeel, Levine. (2017). **Reinforcement Learning with Deep Energy-Based Policies.**

Rusu et al. (2016). **Progress Neural Networks.**

Rajeswaran, Ghotra, Levine, Ravindran. (2017). **EPOpt: Learning Robust Neural Network Policies Using Model Ensembles.**

Sadeghi, Levine. (2017). **CAD2RL: Real Single Image Flight without a Single Real Image.**

Tobin, Fong, Ray, Schneider, Zaremba, Abbeel. (2017). **Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World.**

Tzeng\*, Devin\*, et al. (2016). **Adapting Deep Visuomotor Representations with Weak Pairwise Constraints.**

Bousmalis et al. (2017). **Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping.**

Break

# How can we frame transfer learning problems?

1. “Forward” transfer: train on one task, transfer to a new task
  - a) Just try it and hope for the best
  - b) Finetune on the new task
  - c) Architectures for transfer: progressive networks
  - d) Randomize source task domain
2. Multi-task transfer: train on many tasks, transfer to a new task
  - a) Model-based reinforcement learning
  - b) Model distillation
  - c) Contextual policies
  - d) Modular policy networks
3. Multi-task meta-learning: learn to learn from many tasks
  - a) RNN-based meta-learning
  - b) Gradient-based meta-learning

# Multiple source domains

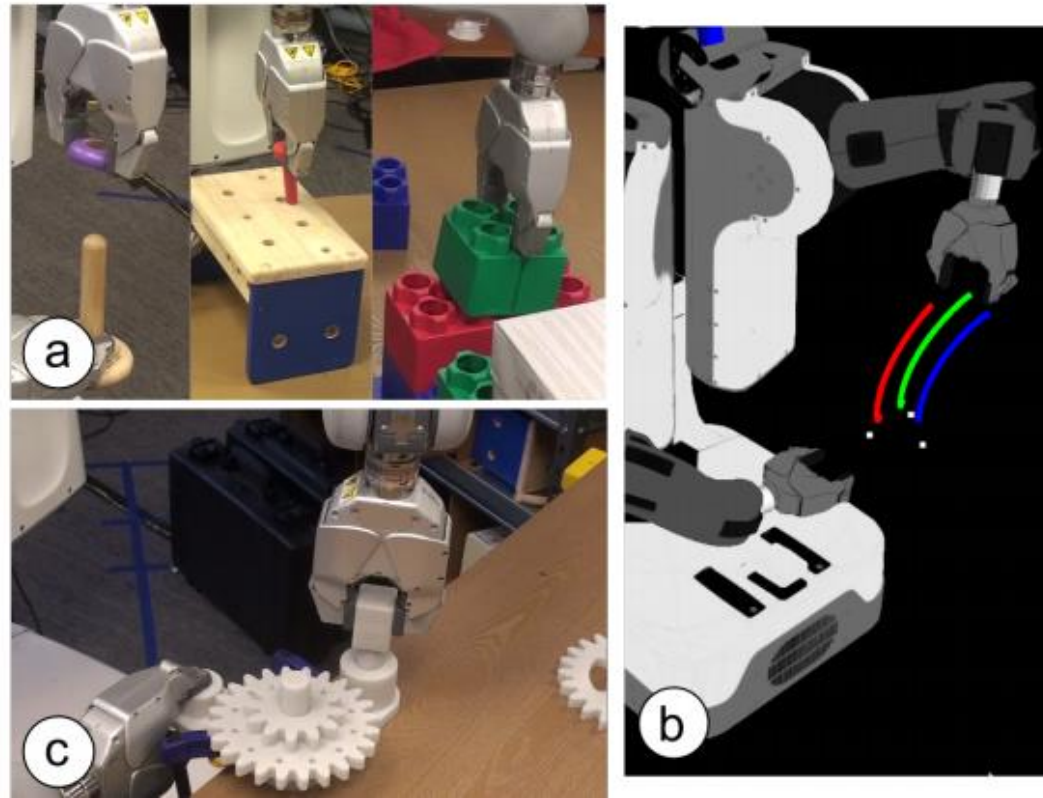
- So far: more diversity = better transfer
- Need to design this diversity
  - E.g., simulation to real world transfer: randomize the simulation
- What if we transfer from multiple *different* tasks?
  - In a sense, closer to what people do: build on a lifetime of experience
  - Substantially harder: past tasks don't directly tell us how to solve the task in the target domain!

# Model-based reinforcement learning

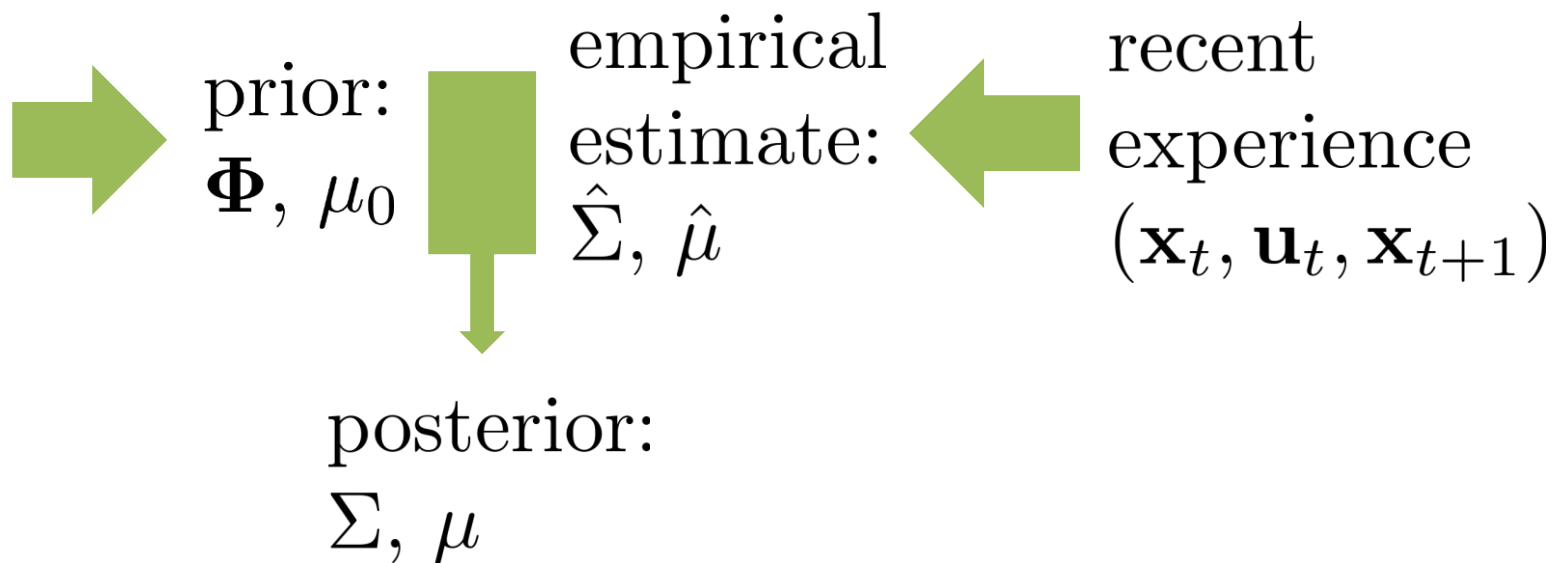
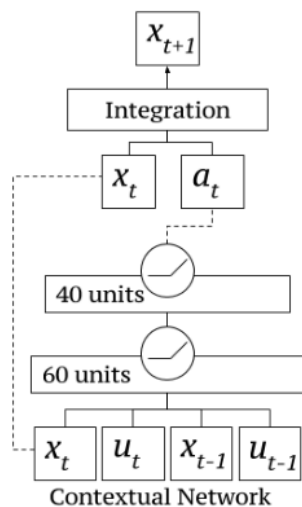
- If the past tasks are all different, what do they have in common?
- Idea 1: the laws of physics
  - Same robot doing different chores
  - Same car driving to different destinations
  - Trying to accomplish different things in the same open-ended video game
- Simple version: train model on past tasks, and then use it to solve new tasks
- More complex version: adapt or finetune the model to new task
  - Easier than finetuning the policy is task is very different but physics are mostly the same

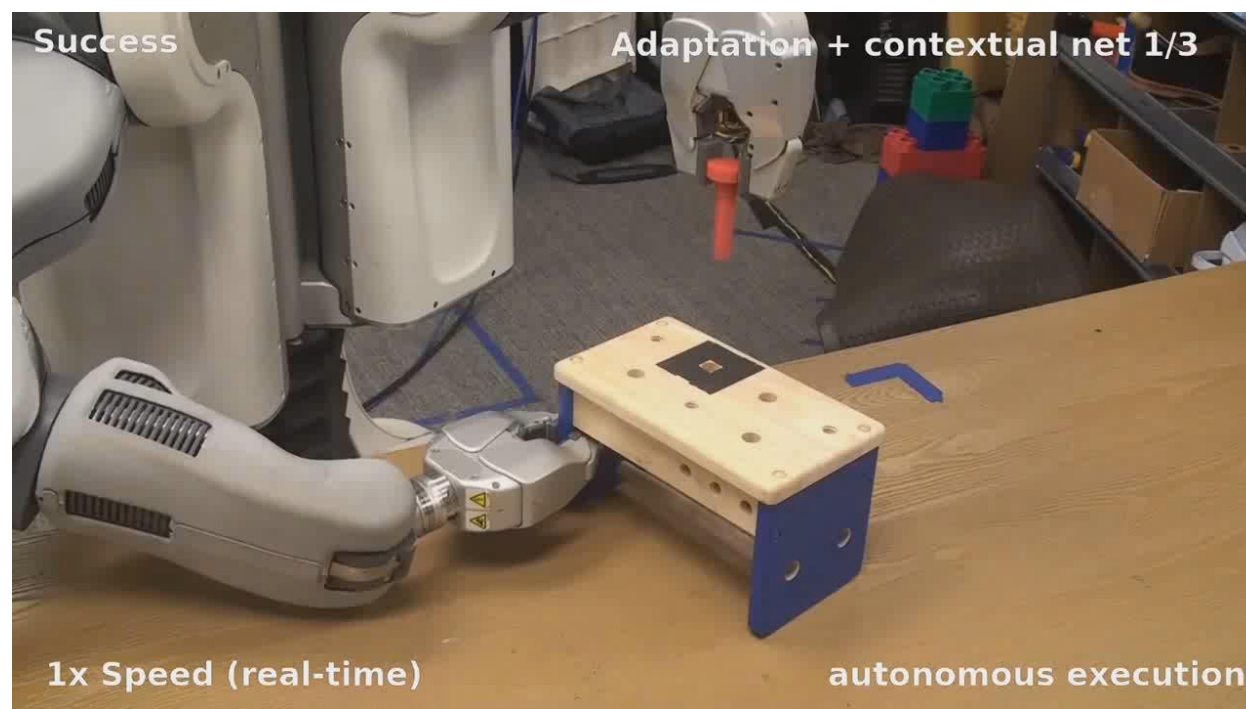
# Model-based reinforcement learning

Example: 1-shot learning with model priors



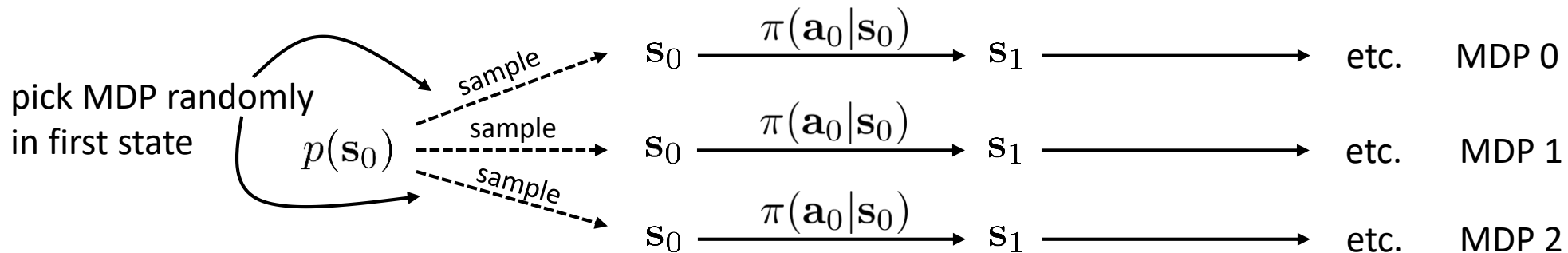






# Can we solve multiple tasks at once?

- Sometimes learning a model is very hard
- Can we learn a multi-task policy that can *simultaneously* perform many tasks?
- Should be *easier* to adapt to new tasks
- Idea 1: construct a joint MDP



- Idea 2: train in each MDP separately, and then combine the policies

# Actor-mimic and policy distillation

Goal: learn a single policy that can play all Atari games

## POLICY DISTILLATION

**Andrei A. Rusu, Sergio Gómez Colmenarejo, Çağlar Gülçehre\*, Guillaume Desjardins,  
James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu & Raia Hadsel**  
Google DeepMind

## ACTOR-MIMIC DEEP MULTITASK AND TRANSFER REINFORCEMENT LEARNING

**Emilio Parisotto, Jimmy Ba, Ruslan Salakhutdinov**  
Department of Computer Science  
University of Toronto

# Background: Ensembles & Distillation

**Ensemble models:** single models are often not the most robust – instead train many models and average their predictions

this is how most ML competitions (e.g., Kaggle) are won

this is very expensive at test time

**Can we make a single model that is as good as an ensemble?**

**Distillation:** train on the ensemble's predictions as “soft” targets

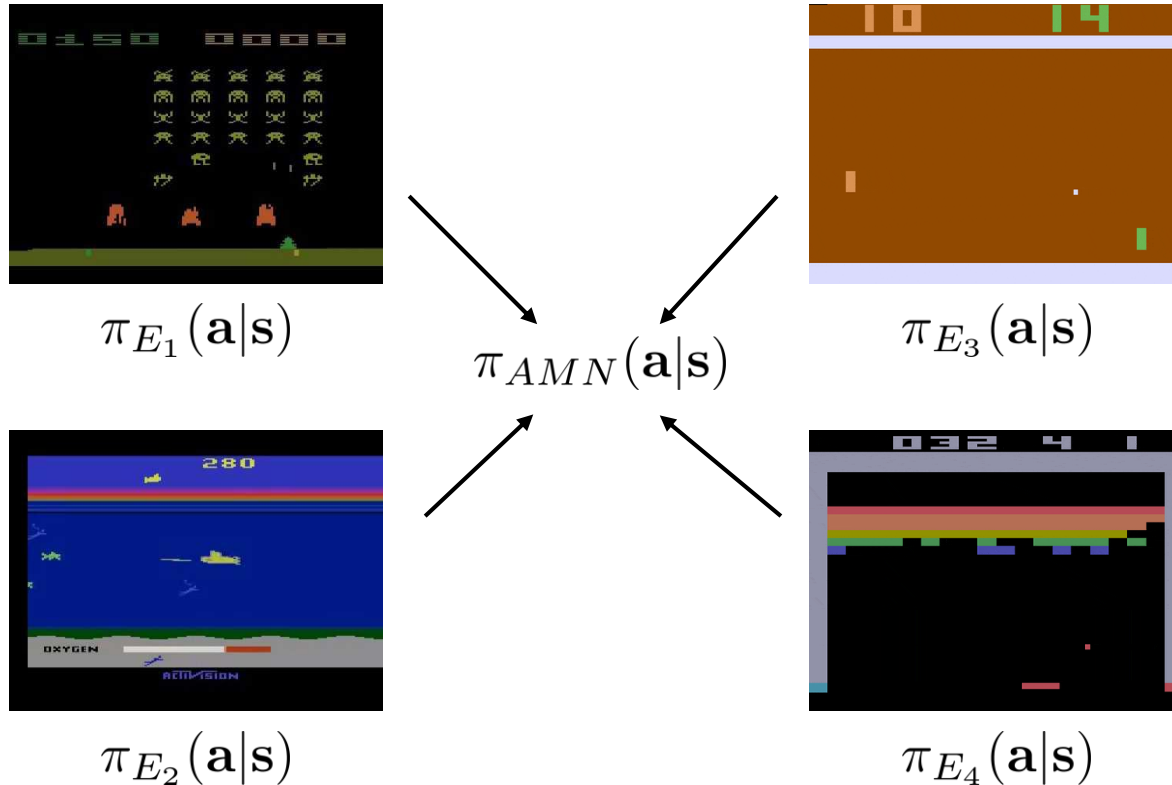
$$p_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

logit  $\rightarrow$   $\exp(z_i/T)$   $\leftarrow$  temperature

**Intuition:** more knowledge in soft targets than hard labels!



# Distillation for Multi-Task Transfer



$$\mathcal{L} = \sum_{\mathbf{a}} \pi_{E_i}(\mathbf{a}|\mathbf{s}) \log \pi_{AMN}(\mathbf{a}|\mathbf{s})$$

(just supervised learning/distillation)

analogous to guided policy search, but  
for transfer learning

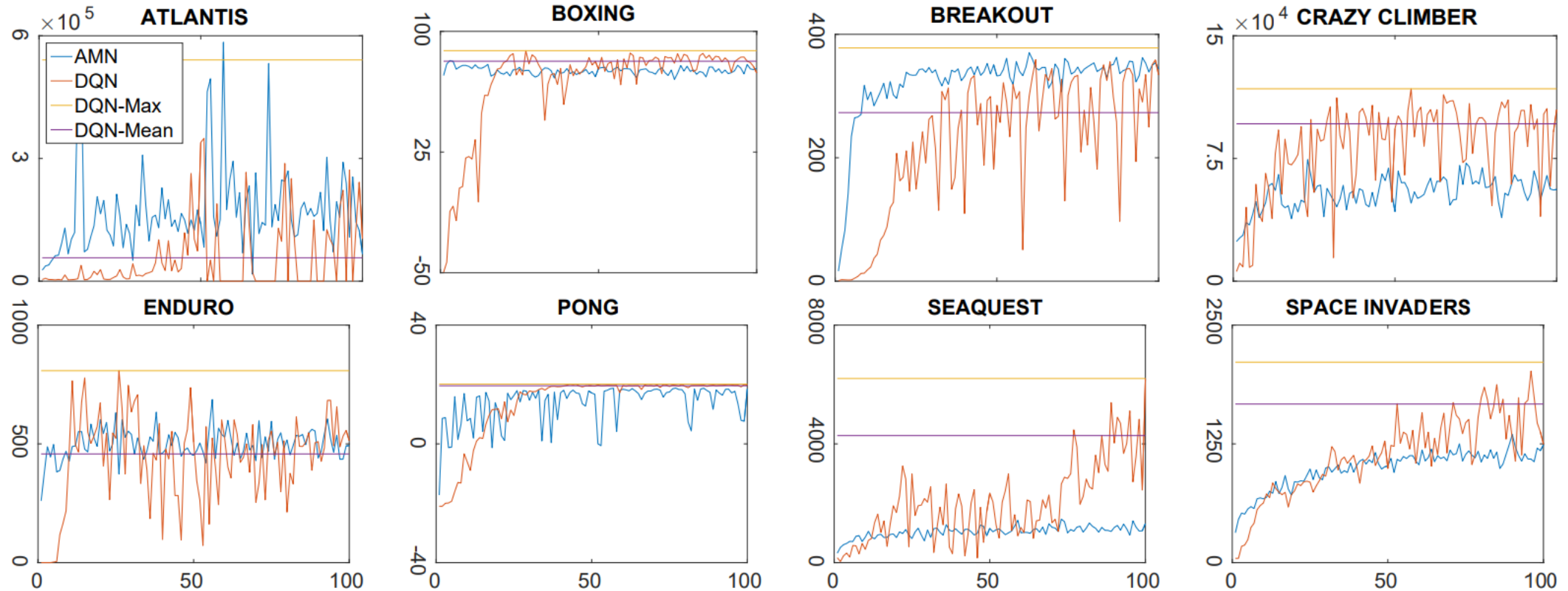
-> see model-based RL slides

some other details

(e.g., feature regression objective)

– see paper

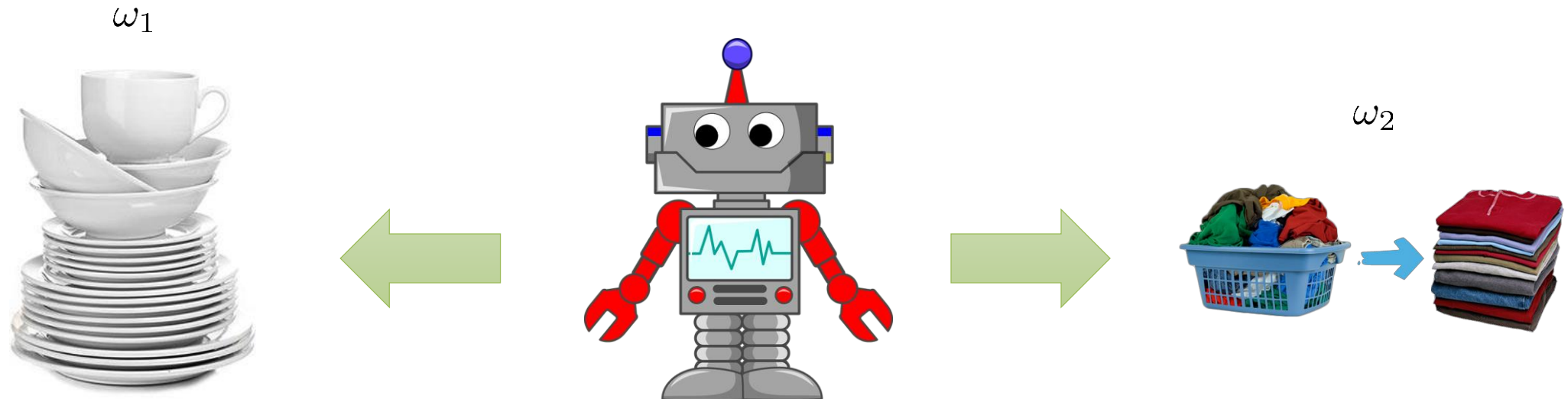
# Distillation Transfer Results





# How does the model know what to do?

- So far: what to do is apparent from the input (e.g., which game is being played)
- What if the policy can do *multiple* things in the *same* environment?



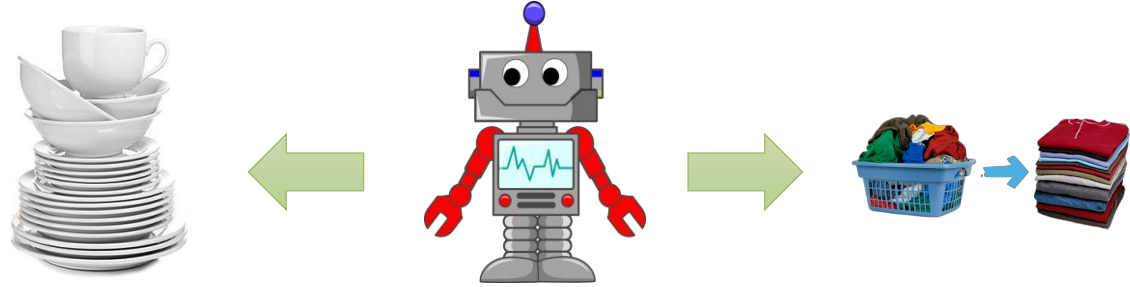


# Contextual policies

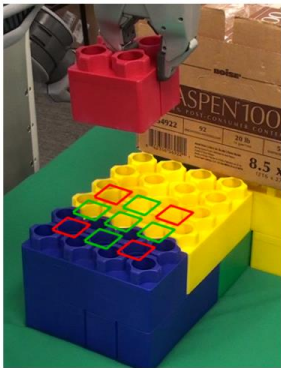
standard policy:  $\pi_{\theta}(\mathbf{a}|\mathbf{s})$

contextual policy:  $\pi_{\theta}(\mathbf{a}|\mathbf{s}, \omega)$

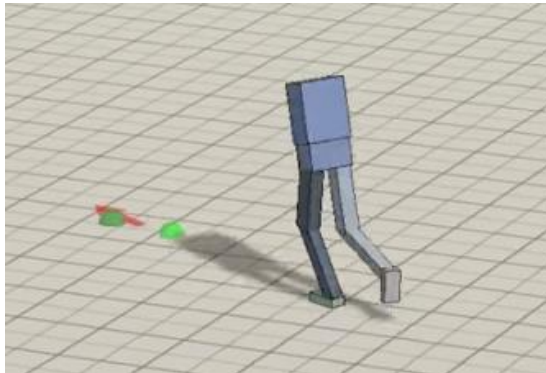
e.g., do dishes or laundry



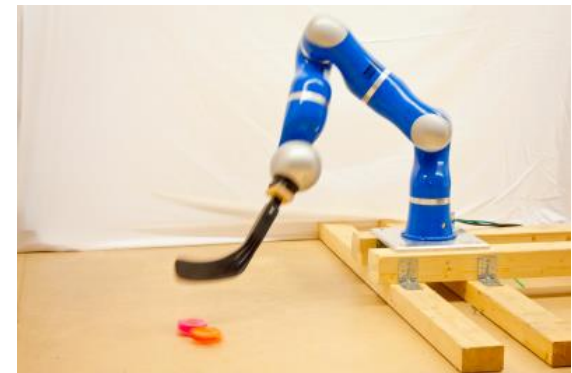
formally, simply defines augmented state space:  $\tilde{\mathbf{s}} = \begin{bmatrix} \mathbf{s} \\ \omega \end{bmatrix}$   $\tilde{\mathcal{S}} = \mathcal{S} \times \Omega$



$\omega$ : stack location



$\omega$ : walking direction

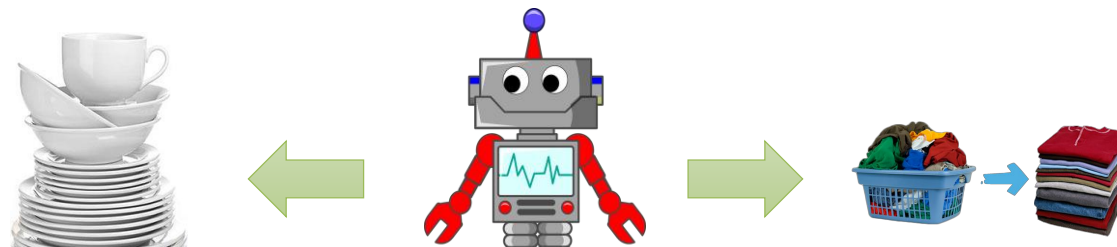


$\omega$ : where to hit puck

# Contextual policies

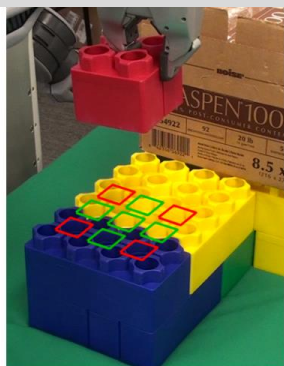
standard policy:  $\pi_{\theta}(\mathbf{a}|\mathbf{s})$

contextual policy:  $\pi_{\theta}(\mathbf{a}|\mathbf{s}, \omega)$

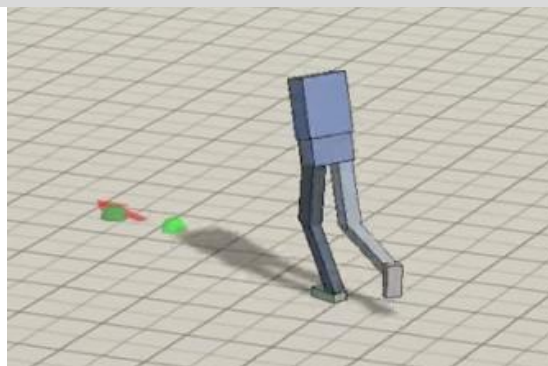


will discuss more in the context  
of meta-learning!

for



$\omega$ : stack location



$\omega$ : walking direction

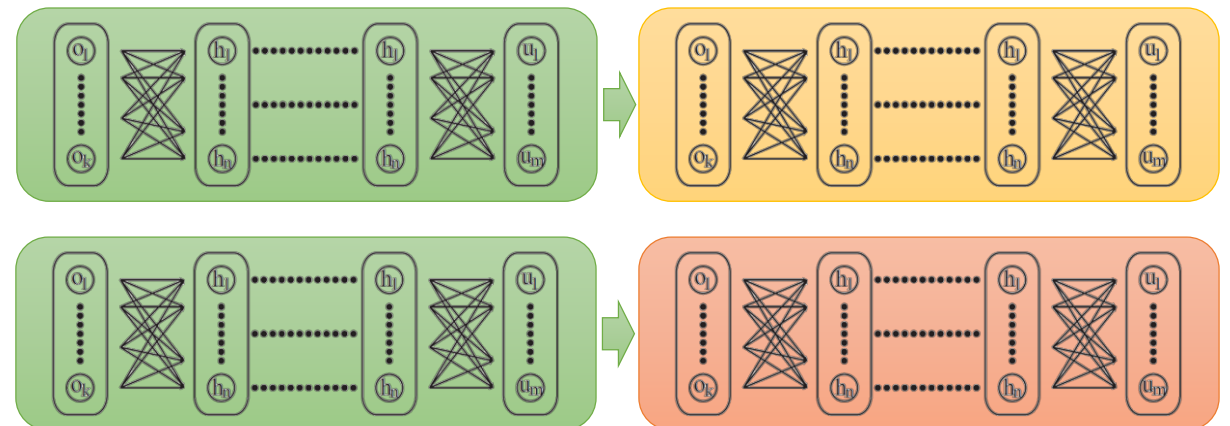


$\omega$ : where to hit puck

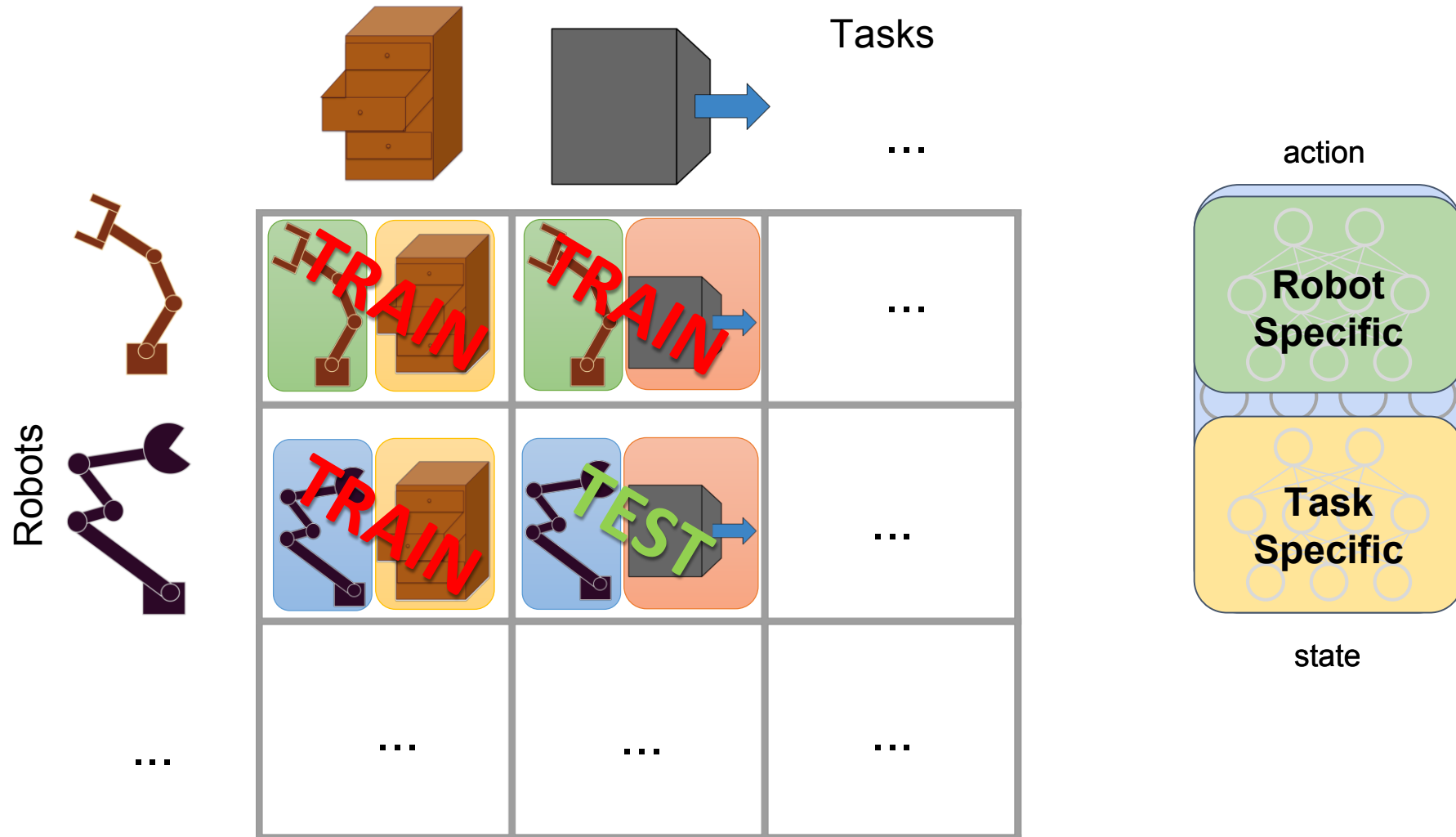
# Architectures for multi-task transfer

- So far: single neural network for all tasks (in the end)
- What if tasks have some shared parts and some distinct parts?
  - Example: two cars, one with camera and one with LIDAR, driving in two different cities
  - Example: ten different robots trying to do ten different tasks
- Can we design architectures with *reusable components*?

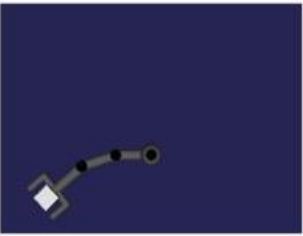
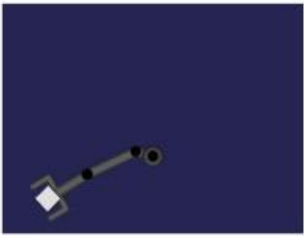


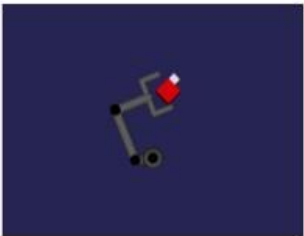

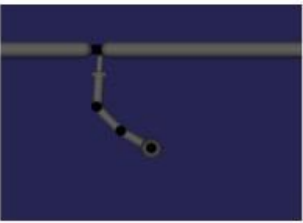
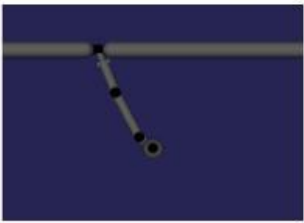
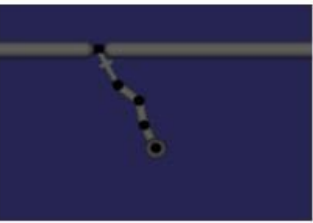
## Modular Policies

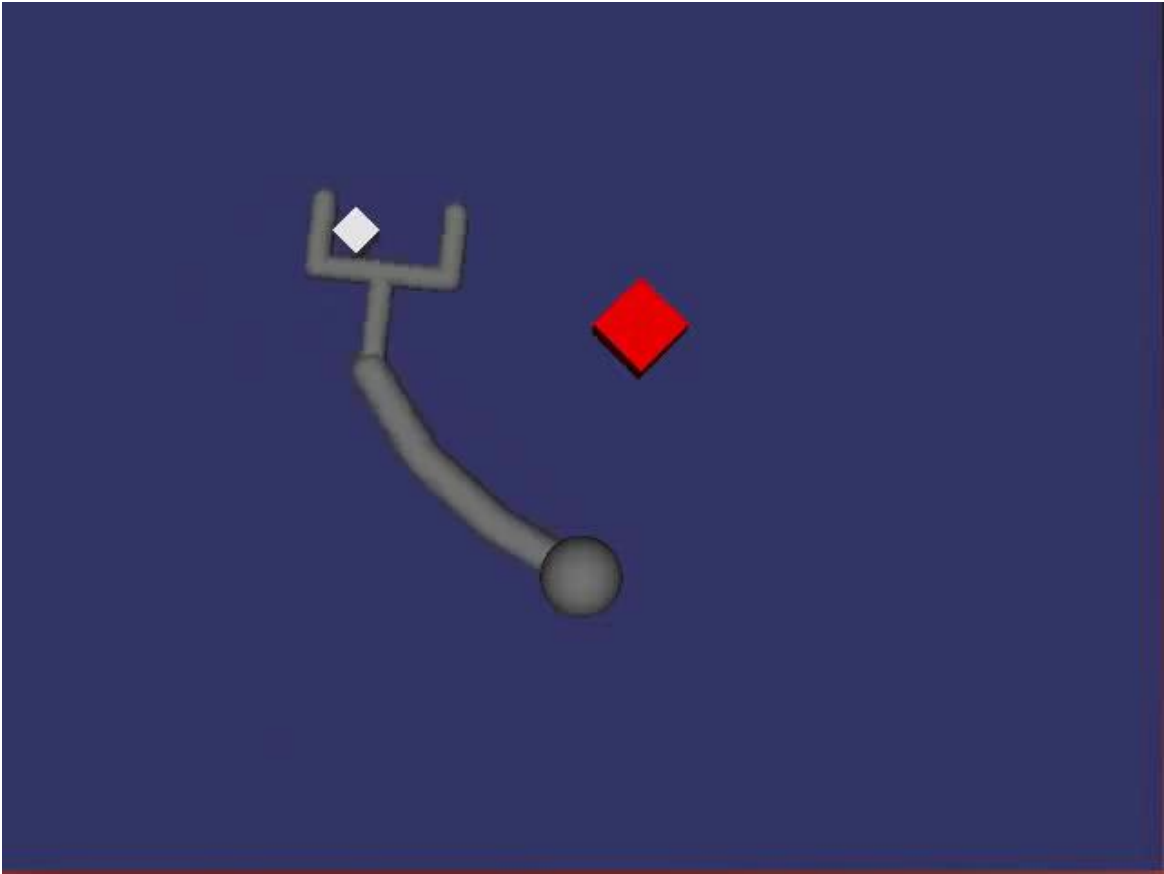


# Modular networks



# Modular networks

Robots Tasks	3link	3link different config	4link
Reach			
Push			
Peg insert			



# Multi-task learning summary

- More tasks = more diversity = better transfer
- Often easier to obtain multiple different but relevant prior tasks
- Model-based RL: transfer the physics, not the behavior
- Distillation: combine multiple policies into one, for concurrent multi-task learning (accelerate all tasks through sharing)
- Contextual policies: policies that are told *what* to do
- Architectures for multi-task learning: modular networks

# Suggested readings

Fu, Levine, Abbeel. (2016). **One-Shot Learning of Manipulation Skills with Online Dynamics Adaptation and Neural Network Priors.**

Rusu et al. (2016). **Policy Distillation.**

Parisotto, Ba, Salakhutdinov. (2016). **Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning.**

Devin\*, Gupta\*, Darrell, Abbeel, Levine. (2017). **Learning Modular Neural Network Policies for Multi-Task and Multi-Robot Transfer.**

# How can we frame transfer learning problems?

1. “Forward” transfer: train on one task, transfer to a new task
  - a) Just try it and hope for the best
  - b) Finetune
  - c) Architecture search
  - d) Random search
2. Multi-task transfer: train on many tasks, transfer to a new task
  - a) Model-based reinforcement learning
  - b) Model distillation
  - c) Contextual policies
  - d) Modular policy networks
3. Multi-task meta-learning: learn to learn from many tasks
  - a) RNN-based meta-learning
  - b) Gradient-based meta-learning

**more on this next time!**