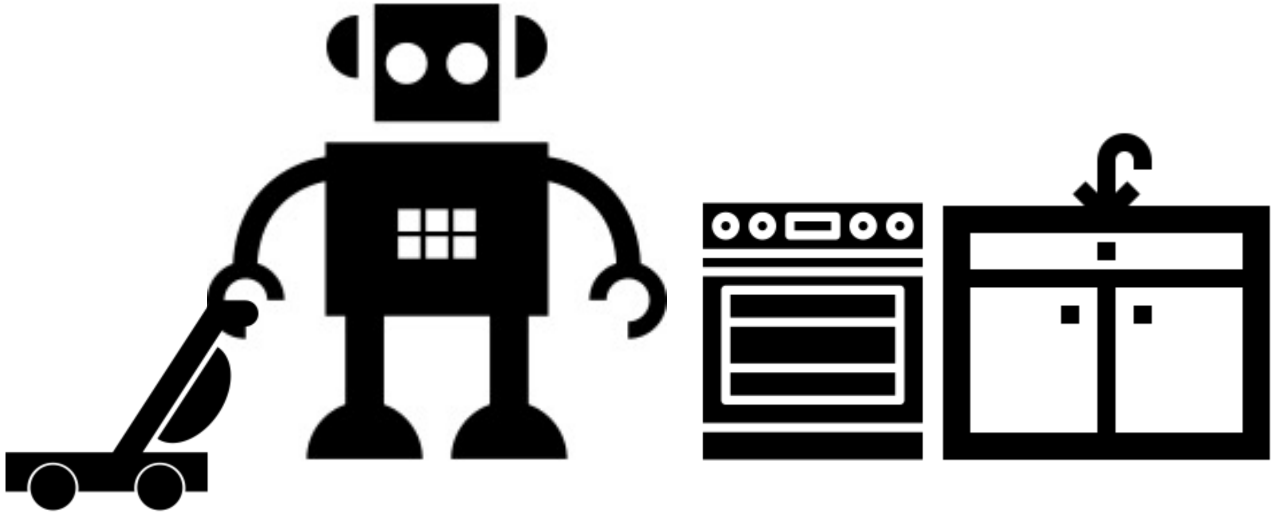
Leveraging model-learning for extreme generalization

Leslie Pack Kaelbling

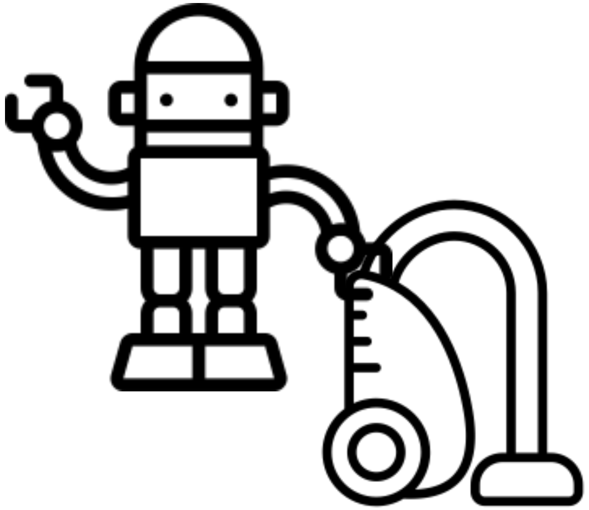
MIT CSAIL

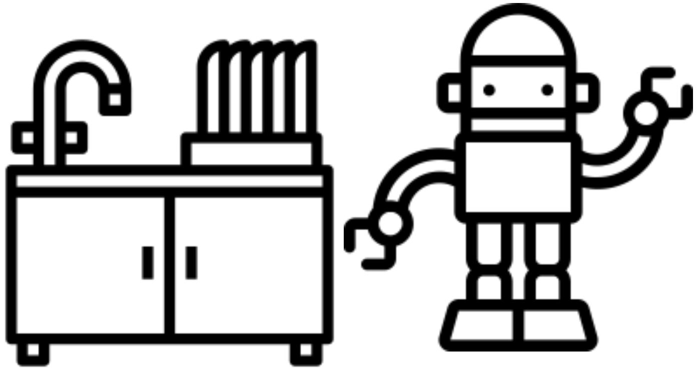
Research goal:
understand the computational mechanisms necessary to make
a general-purpose intelligent robot

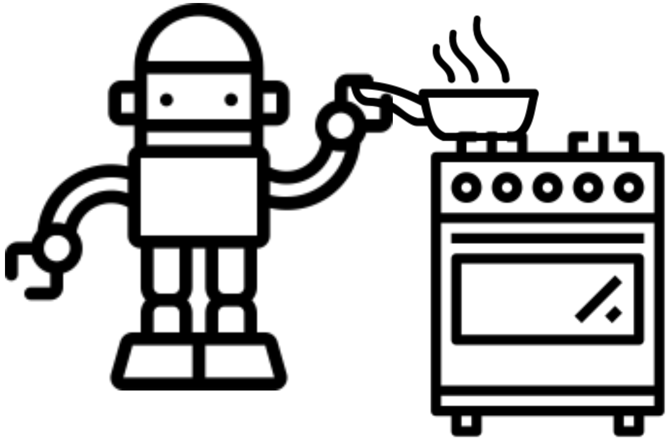


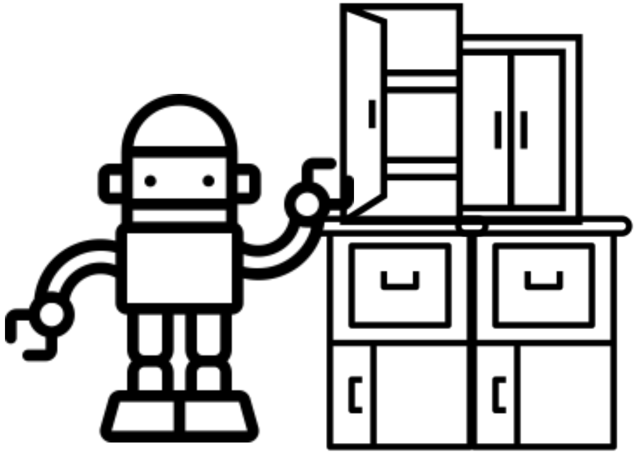
Domain variability: Make tea in any kitchen

Task variability: Do any household job





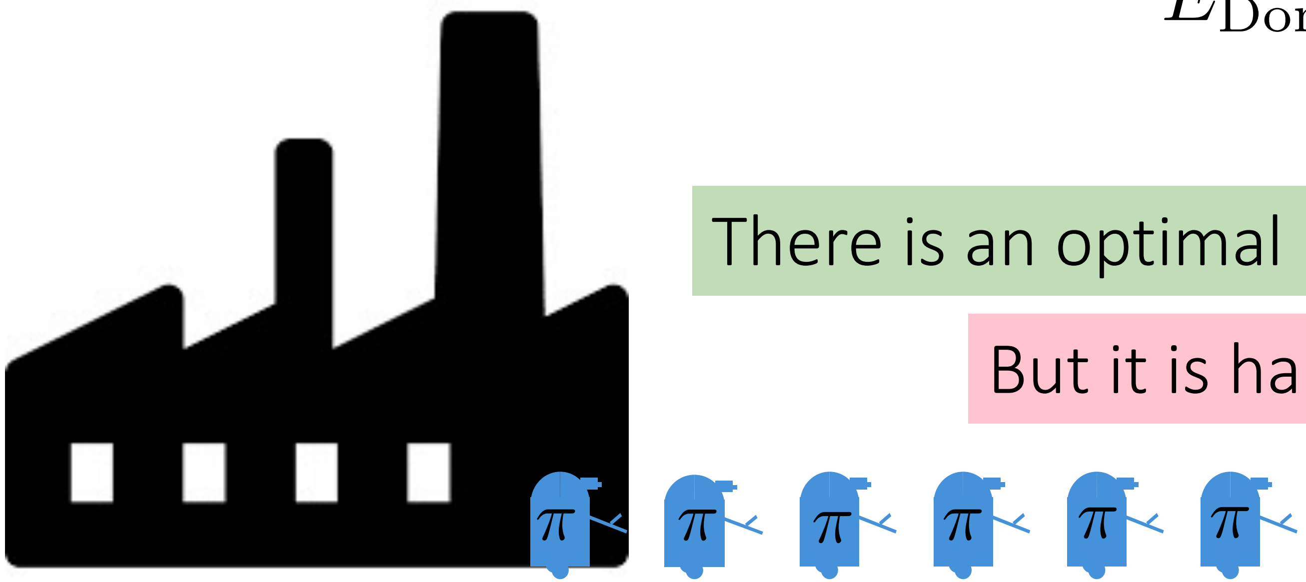




Robot **factory** generates robots that work in the **wild**

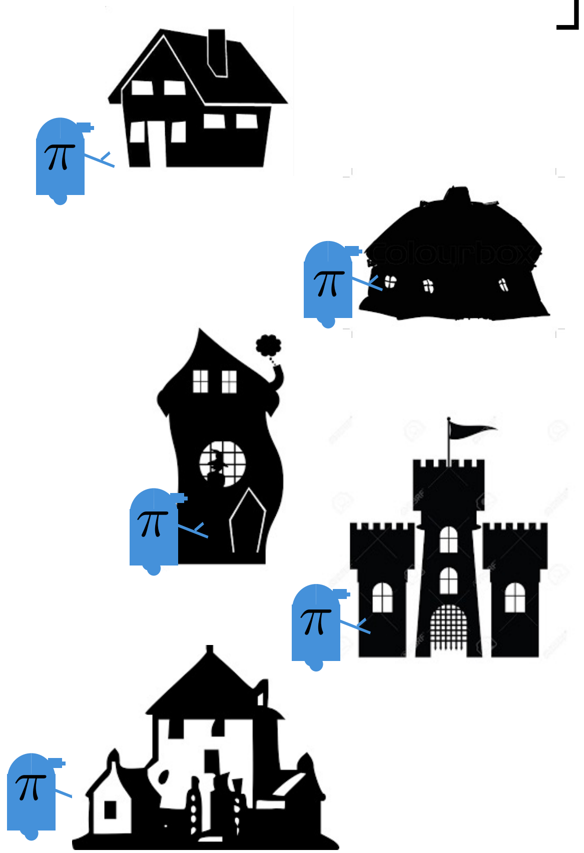
Find a mapping $\pi : (o, a)^* \rightarrow a$ that optimizes

$$E_{\text{Dom}}\left[\sum_{t=0}^{T_{\text{dom}}} R_{\text{dom}}(s_t, a_t) \mid \pi, W_{\text{dom}}\right]$$



There is an optimal policy :-)

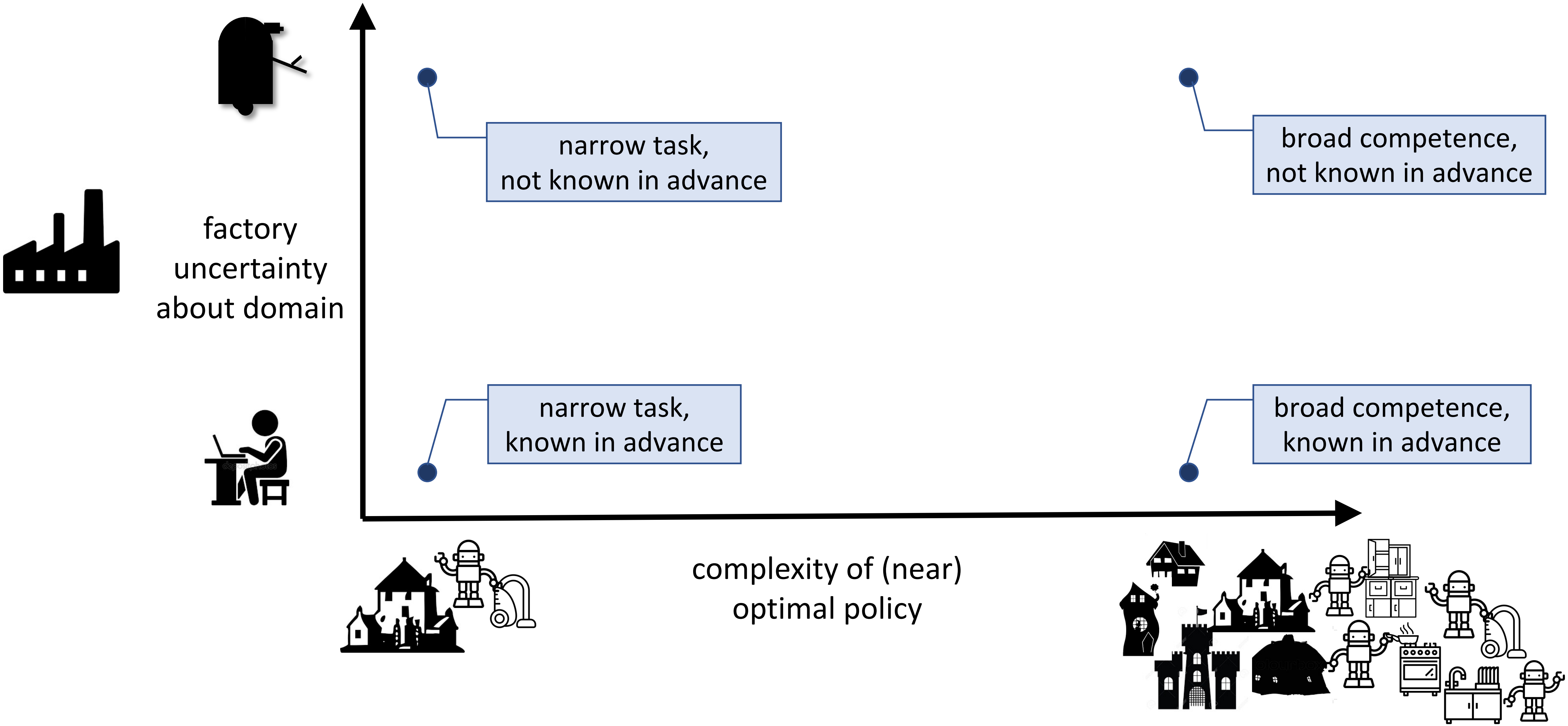
But it is hard to find :-(



Domain specifies

- World (start state dist, transition dynamics)
- Horizon T (could average or discount)
- Reward function (need not be additive, dense, shaped)

The space of distributions over domains $P(\mathrm{Dom})$



How to construct a good policy for a given $P(\mathrm{Dom})$?

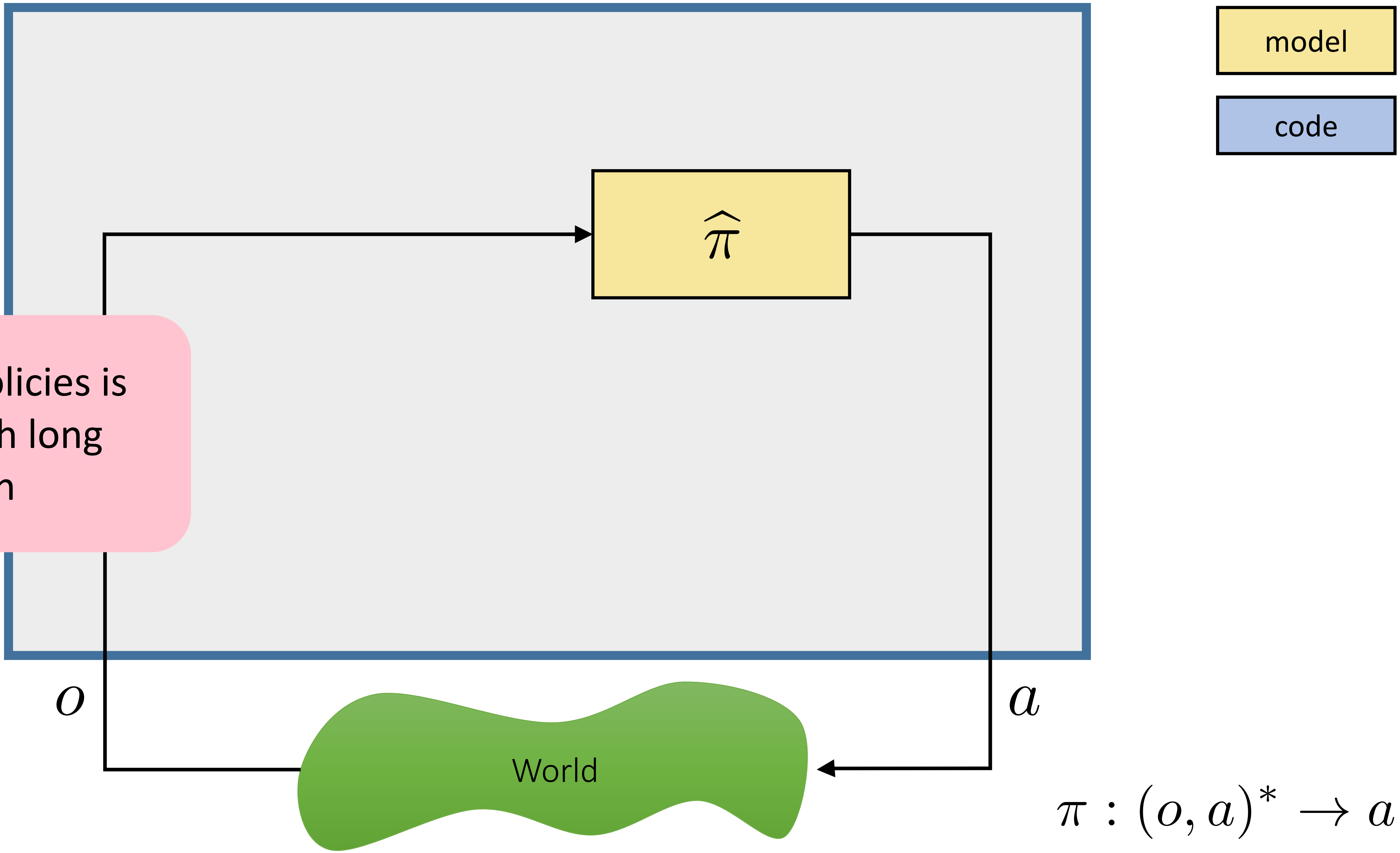
Pick a **model class** and **methodology for acquiring the models** that **perform well** while **minimizing total construction cost:**

- Human
  - model specification
  - simulator (physical or software) construction
  - reward shaping
  - data labeling, gathering, …
- Machine
  - optimization
  - learning
  - parameter tuning

Pick model class where

- humans have most insight
- optimization is easiest
- learning will generalize best

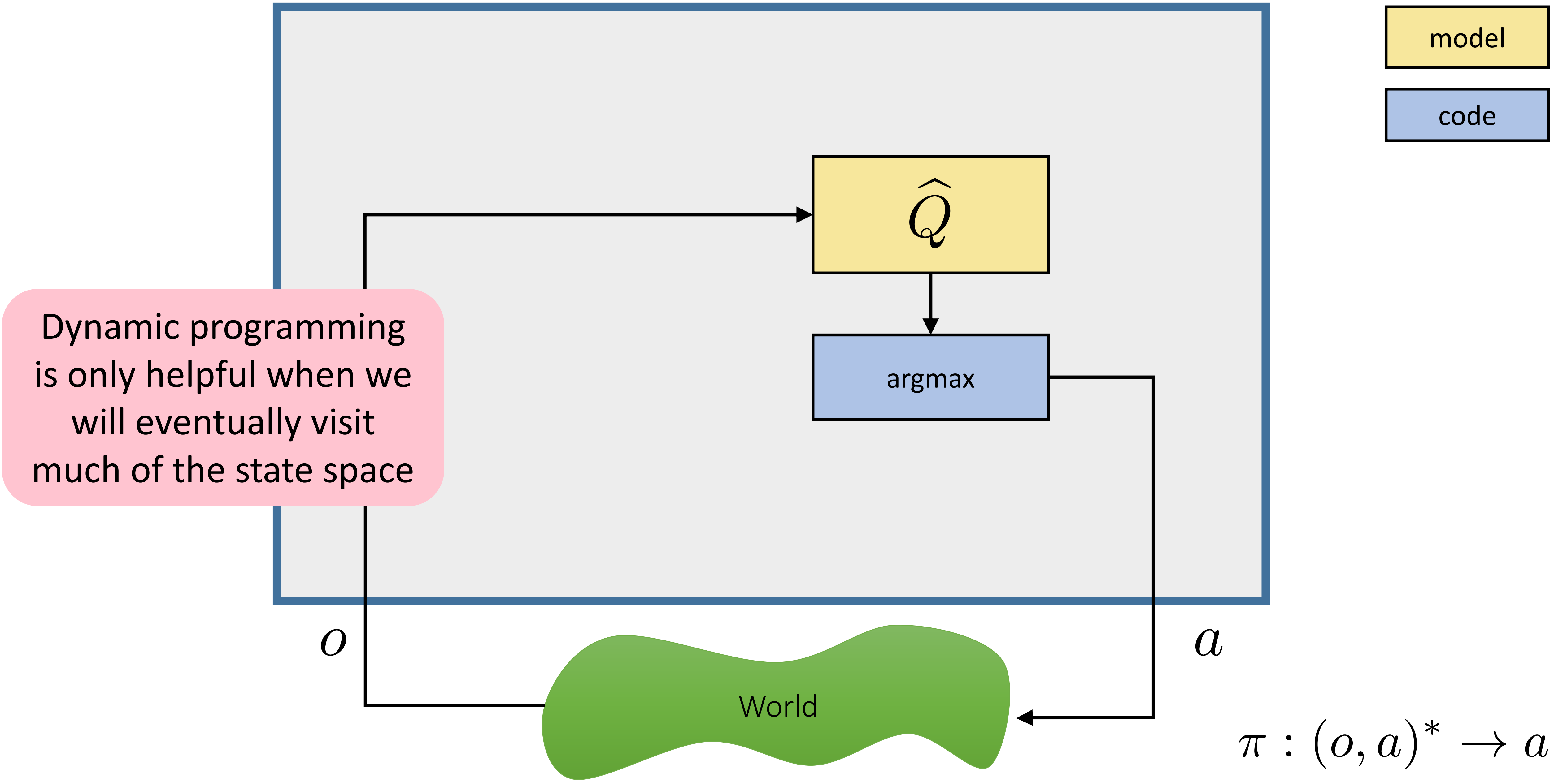
Ways to represent a policy: a policy!



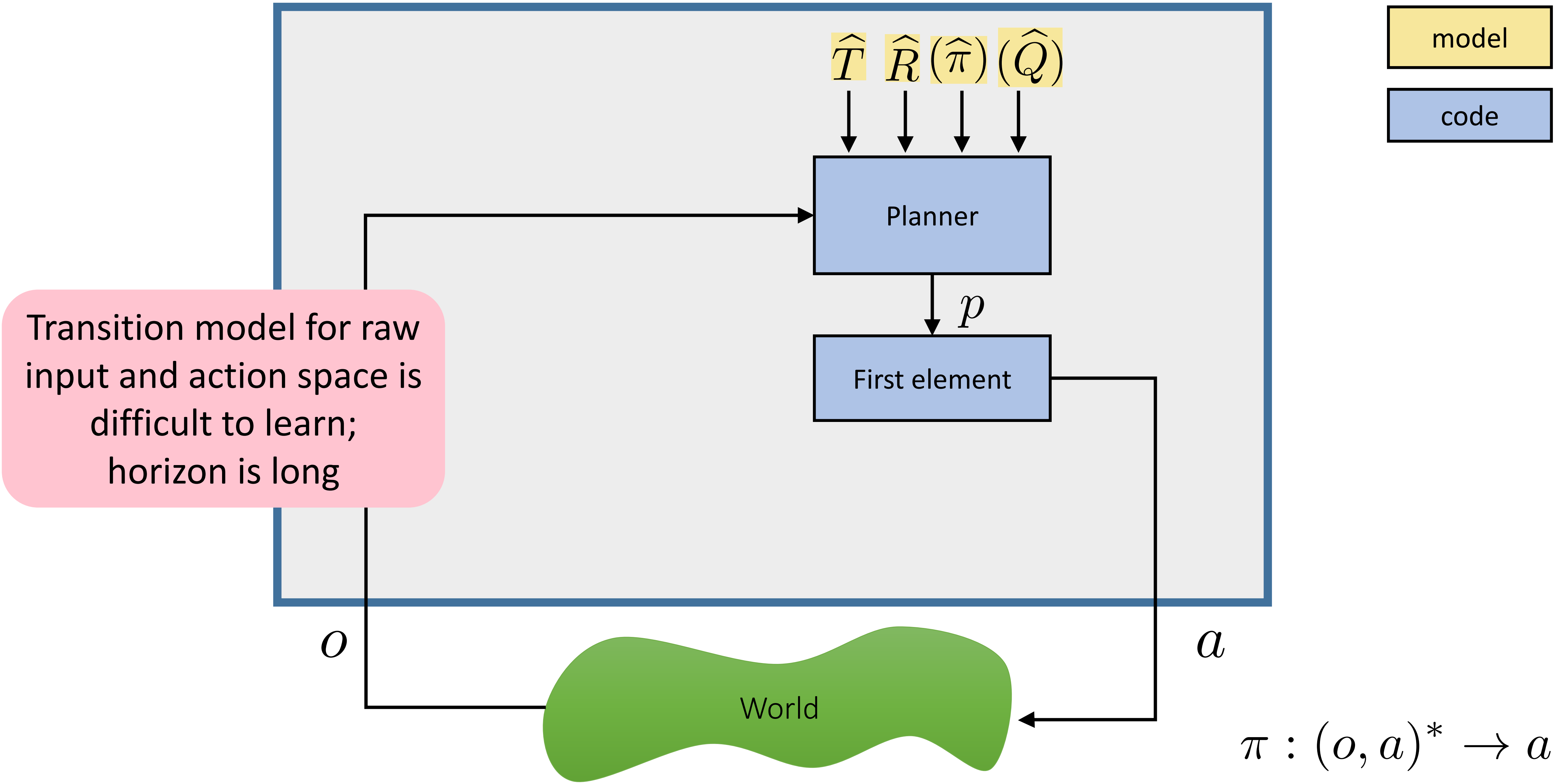
Search for policies is difficult with long horizon

$$\pi : (o, a)^* \to a$$

Ways to represent a policy: Q values

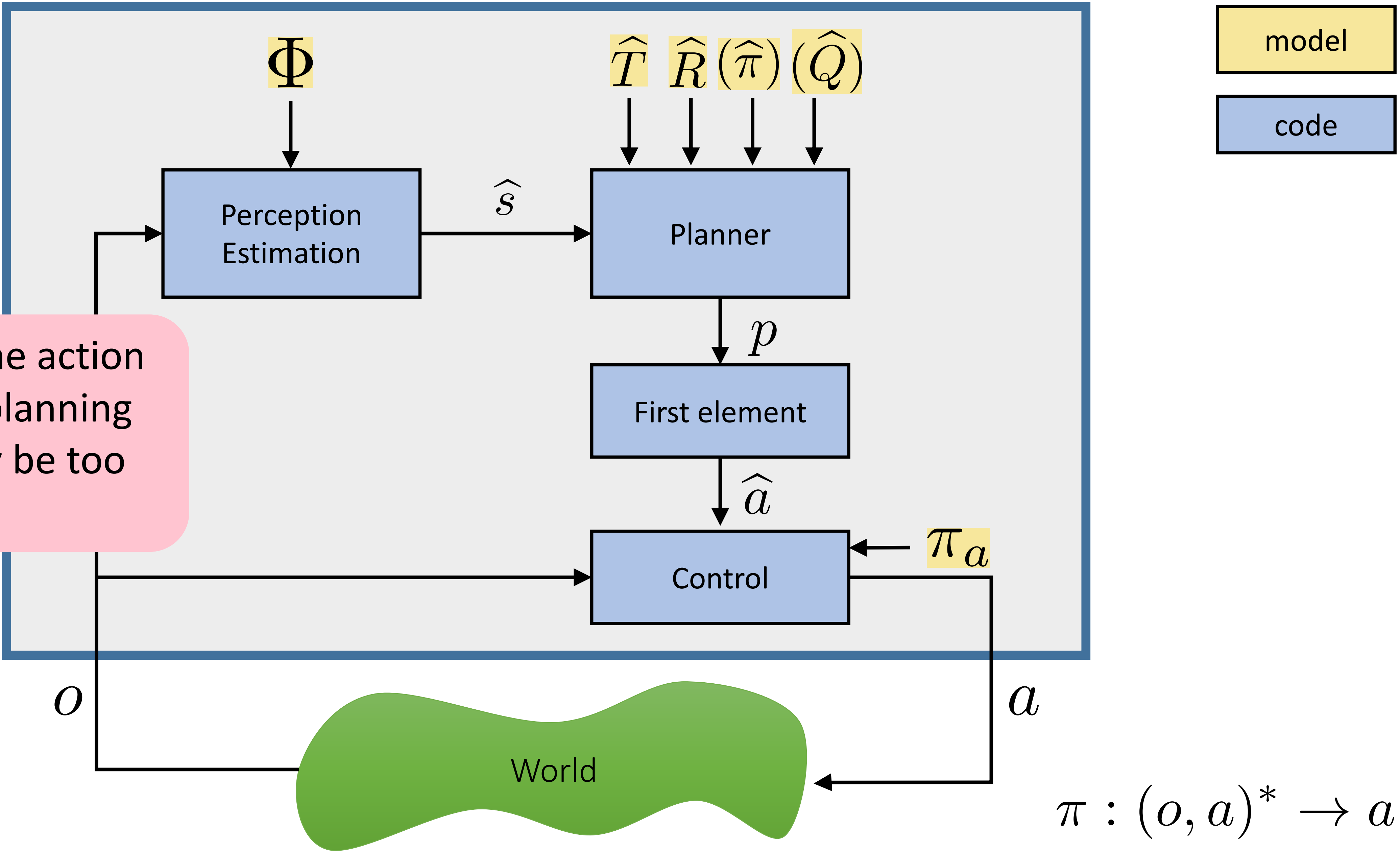


Ways to represent a policy: plan online, with search control



$$\pi : (o, a)^* \to a$$

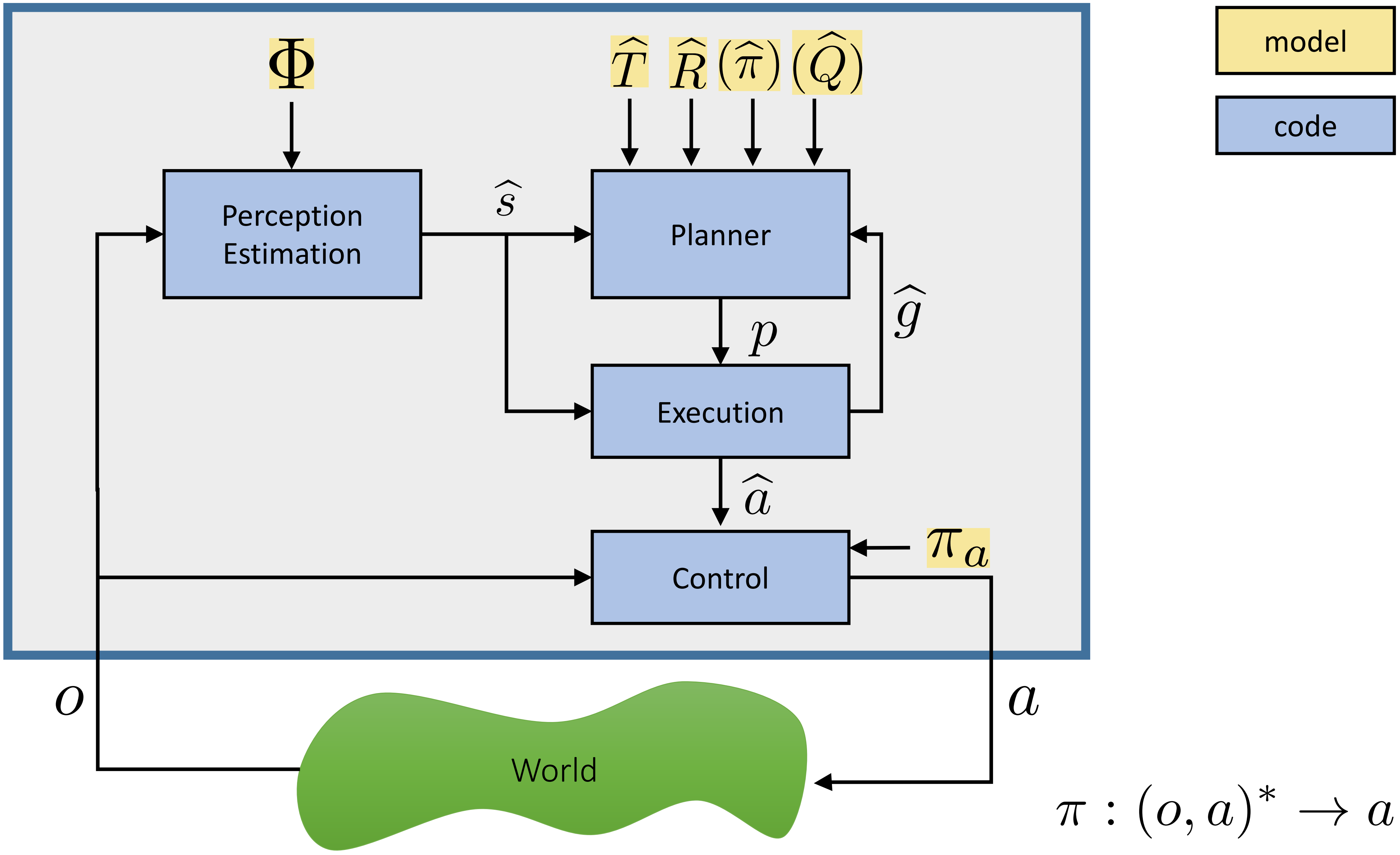
Ways to represent a policy: plan at higher level of abstraction



Even with some action abstraction, planning horizons may be too long

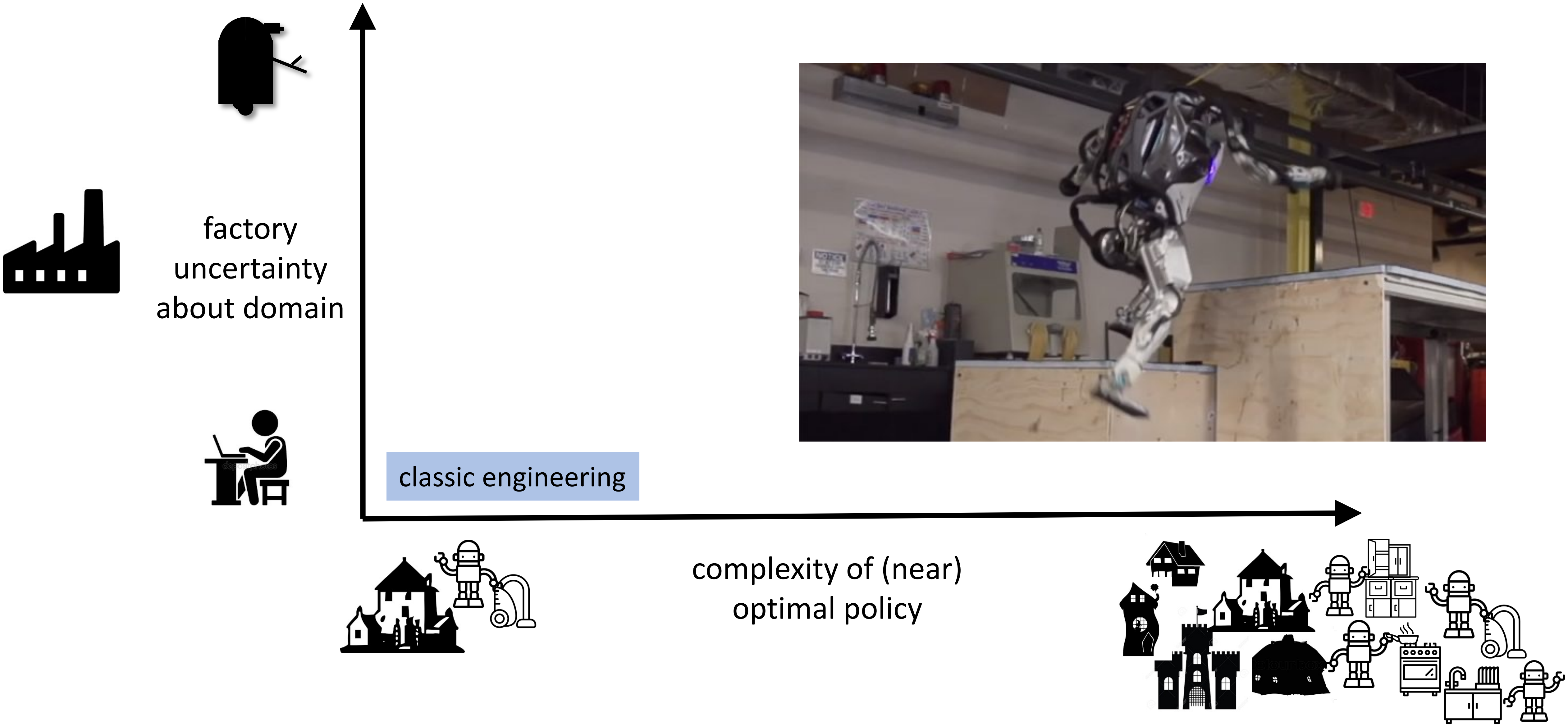
$$\pi : (o, a)^* \to a$$

Ways to represent a policy: add hierarchical plan / execution

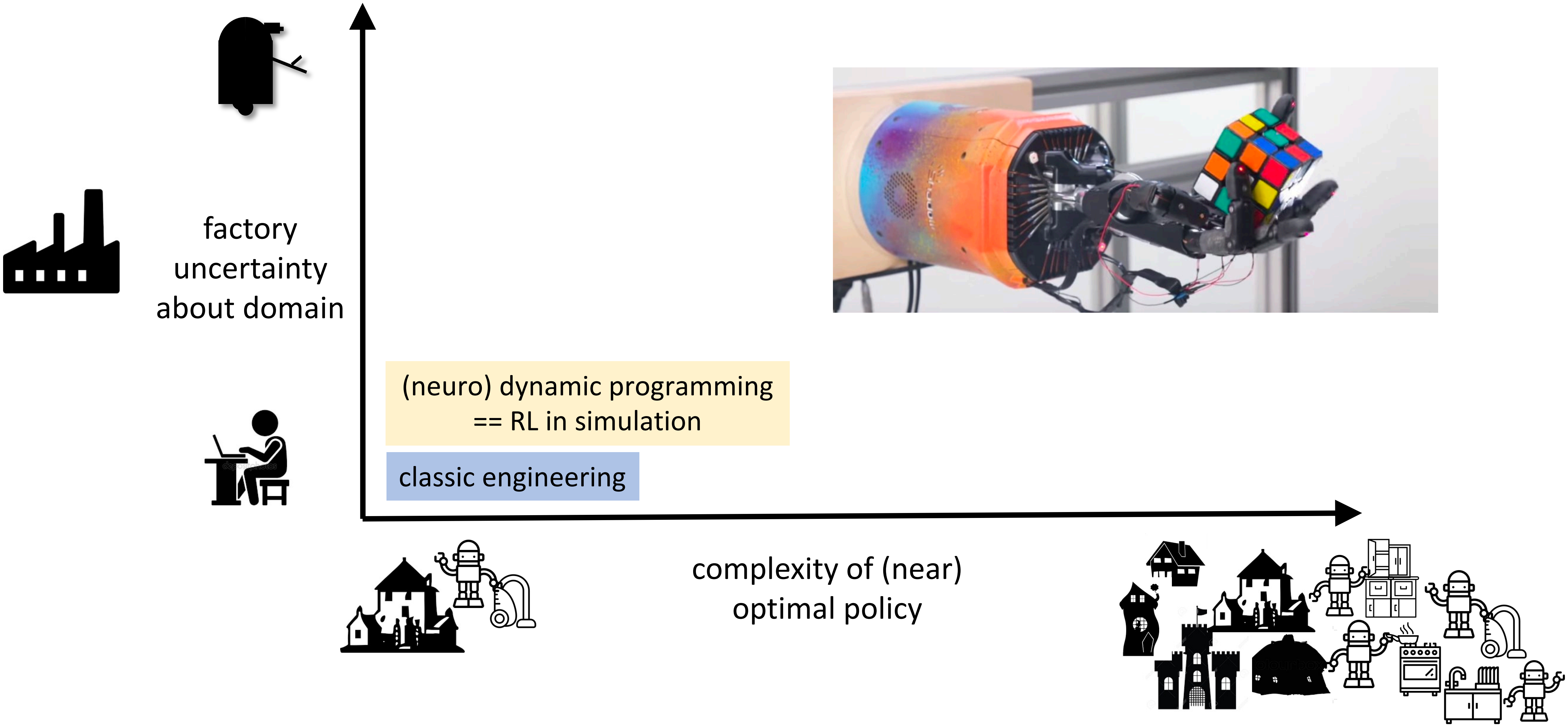


$$\pi : (o, a)^* \to a$$

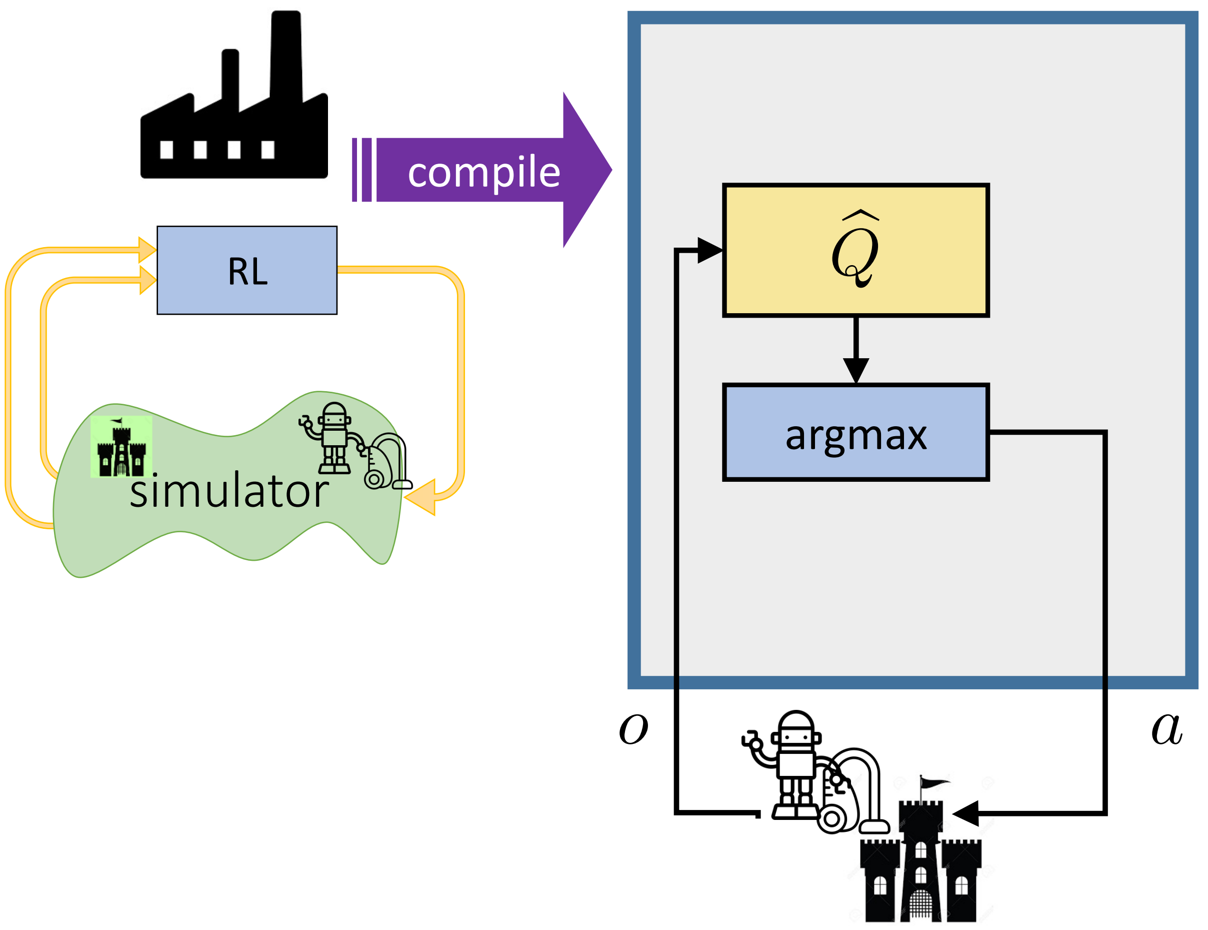
Different ways to obtain a policy



Different ways to obtain a policy

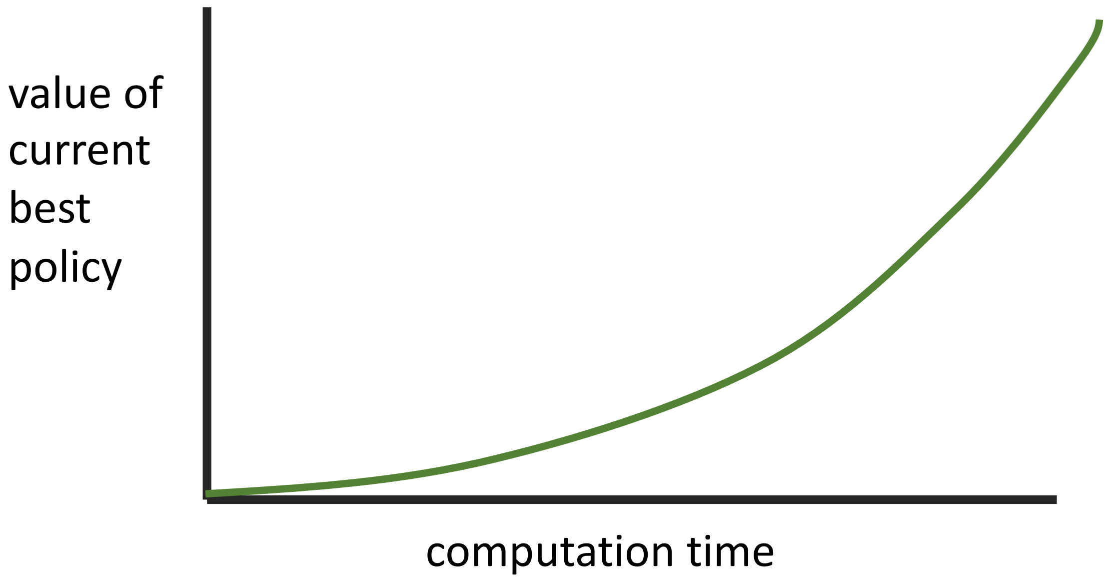


RL in the factory

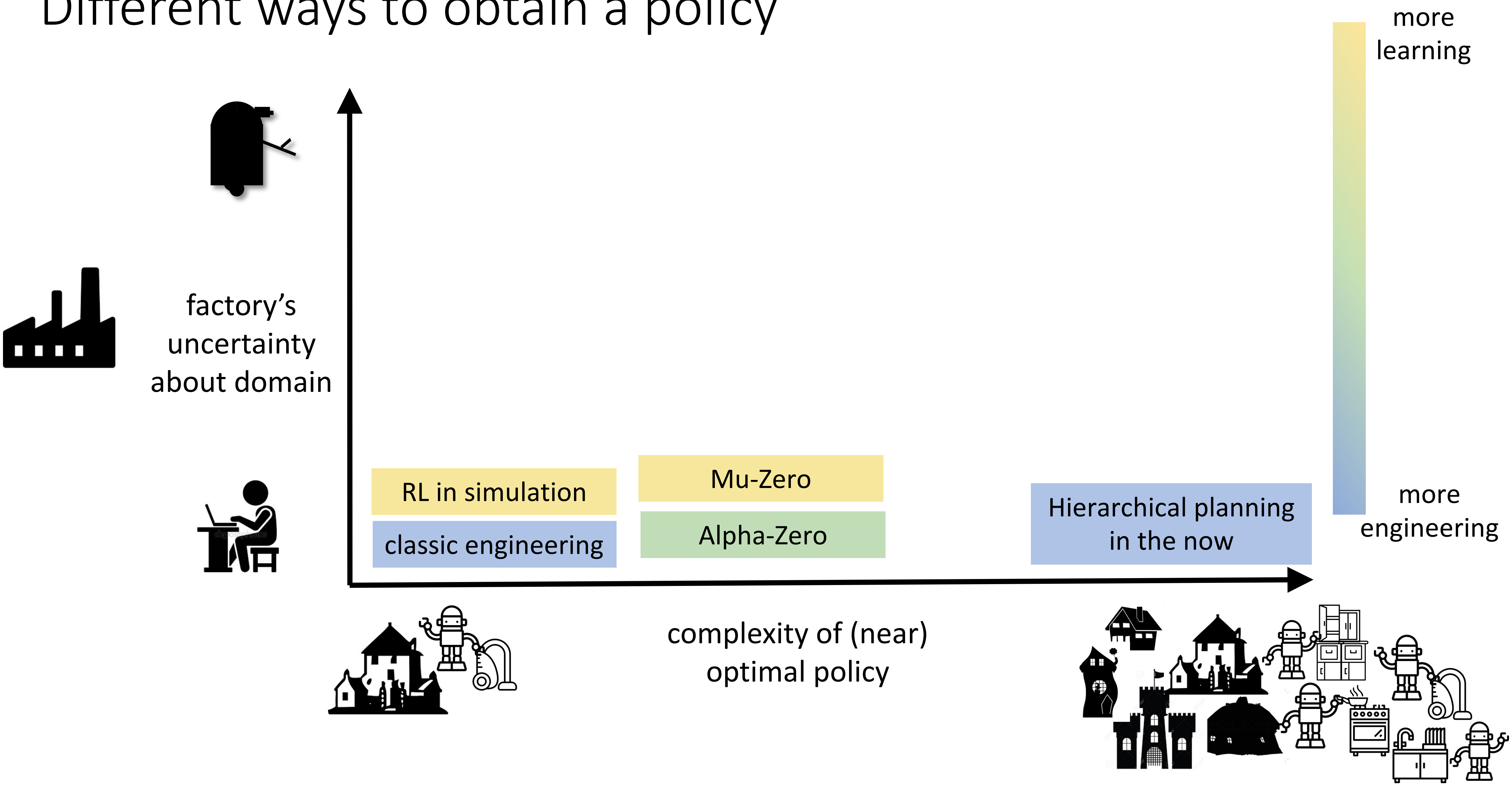


Evaluation

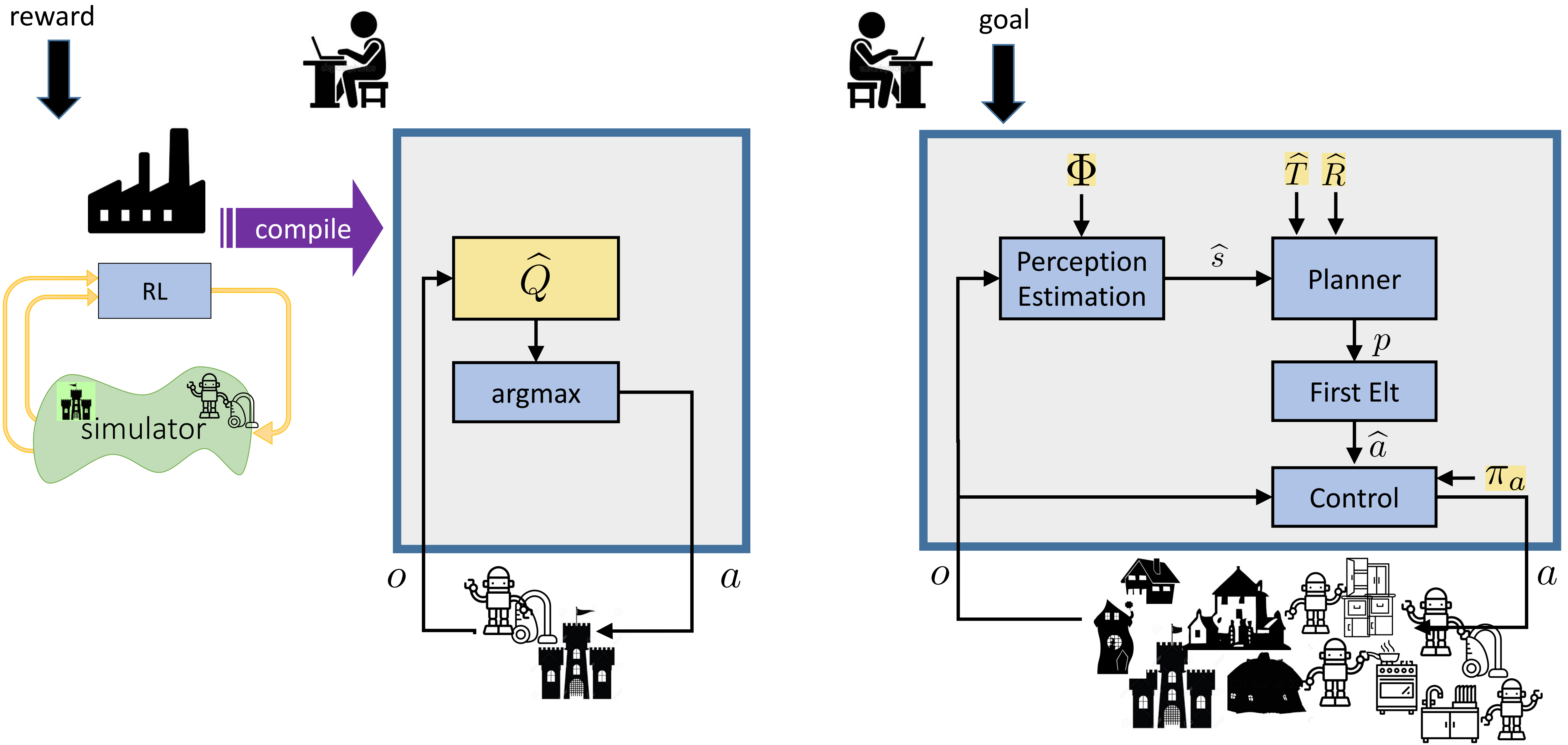
- Reward during learning doesn't matter
- Number of interactions with simulator doesn't matter
- Measure quality of best policy found versus computation time



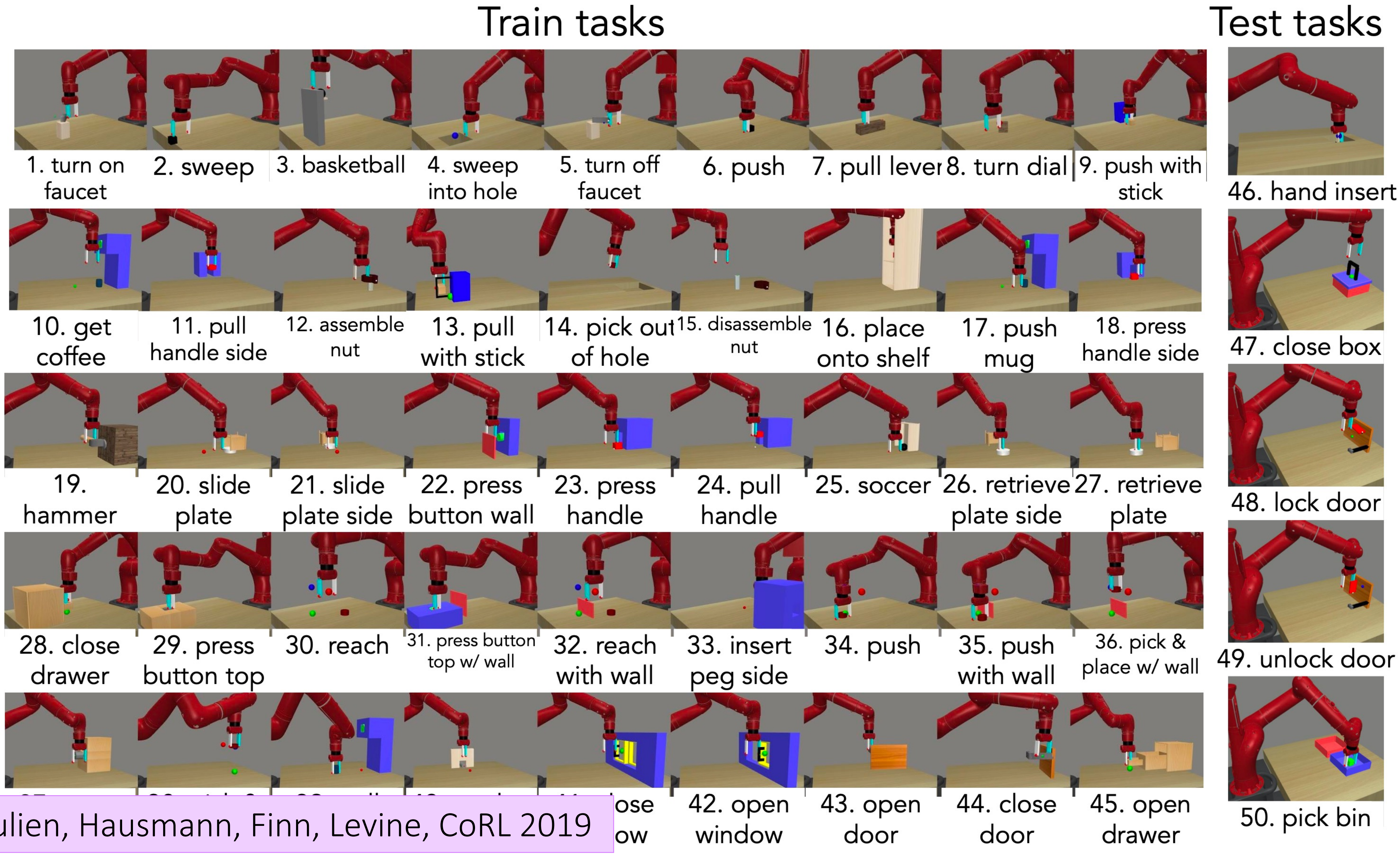
Different ways to obtain a policy



RL in the factory vs planning in the wild: engineering effort



MetaWorld: suite of RL problems designed for meta-learning



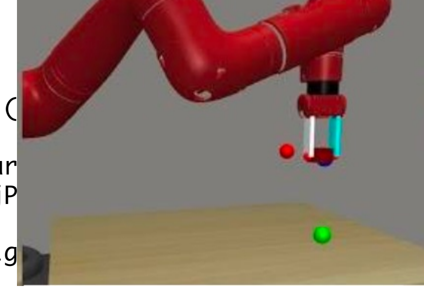
Yu, Quillen, He, Julien, Hausmann, Finn, Levine, CoRL 2019

Reward function for pick and place task

```
  def compute_reward(self, actions, obs, mode = 'general'):
       if isinstance(obs, dict):
           obs = obs['state_observation']
       objPos = obs[3:6]

       rightFinger, leftFinger =
self.get_site_pos('rightEndEffector'),

self.get_site_pos('leftEndEffector')
       fingerCOM  =  (rightFinger + leftFinger)/2

       heightTarget = self.heightTarget
       placingGoal = self._state_goal

       reachDist = np.linalg.norm(objPos - fingerCOM)
       placingDist = np.linalg.norm(objPos[:2] -
placingGoal[:-1])

       def reachReward():
           reachRew = -reachDist# + min(actions[-1], -1)/50
           reachDistxy = np.linalg.norm(objPos[:-1] -
fingerCOM[:-1])
           zRew = np.linalg.norm(fingerCOM[-1] -
self.init_fingerCOM[-1])
           if reachDistxy < 0.06: #0.02
               reachRew = -reachDist
           else:
               reachRew =  -reachDistxy - zRew
           #incentive to close fingers when reachDist is
small
           if reachDist < 0.05:
               reachRew = -reachDist + max(actions[-
1],0)/50
           return reachRew , reachDist

       def pickCompletionCriteria():
           tolerance = 0.01
           if objPos[2] >= (heightTarget- tolerance):
               return True
           else:
               return False

       if pickCompletionCriteria():
           self.pickCompleted = True
```

```
  def objDropped():
           return (objPos[2] <
(self.objHeight + 0.005)) and (placingDist
>0.02) and (reachDist > 0.02)
           # Object on the ground, far away
from the goal, and from the gripper
           #Can tweak the margin limits

       def objGrasped(thresh = 0):
           sensorData = self.data.sensordata
           return (sensorData[0]>thresh) and
(sensorData[1]> thresh)

       def placeCompletionCriteria():
           if abs(objPos[0] - placingGoal[0])
< 0.05 and \
               abs(objPos[1] -
placingGoal[1]) < 0.05 and \
               objPos[2] < self.objHeight +
0.05:
              return True
           else:
              return False
       if placeCompletionCriteria():
           self.placeCompleted = True

       def orig_pickReward():
           # hScale = 50
           hScale = 100
           if self.placeCompleted or
(self.pickCompleted and not(objDropped())):
               return hScale*heightTarget
           # elif (reachDist < 0.1) and
(objPos[2]> (self.objHeight + 0.005)) :
           elif (reachDist < 0.1) and
(objPos[2]> (self.objHeight + 0.005)) :
               return hScale*
min(heightTarget, objPos[2])
           else:
               return 0

       Dist]
```

```
def general_pickReward():
           hScale = 50
           if self.placeCompleted or (
objGrasped()):
               return hScale*heightTar
           elif objGrasped() and (objP
0.005)):
               return hScale* min(heig
           else:
               return 0

       def placeReward():
           # c1 = 1000 ; c2 = 0.01 ; c3 = 0.001
           c1 = 1000 ; c2 = 0.01 ; c3 = 0.001
           placeRew = 1000*(self.maxPlacingDist - placingDist)
+ c1*(np.exp(-(placingDist**2)/c2) + np.exp(-
(placingDist**2)/c3))
           placeRew = max(placeRew,0)
           if mode == 'general':
               cond = self.pickCompleted and objGrasped()
           else:
               cond = self.pickCompleted and (reachDist < 0.1)
and not(objDropped())
           if self.placeCompleted:
               return [-200*actions[-1] + placeRew,
placingDist]
           elif cond:
               if abs(objPos[0] - placingGoal[0]) < 0.05 and \
                   abs(objPos[1] - placingGoal[1]) < 0.05:
                   return [-200*actions[-1] + placeRew,
placingDist]
               else:
                   return [placeRew, placingDist]
           else:
               return [0 , placingDist

       reachRew, reachDist = reachReward()
       if mode == 'general':
           pickRew = general_pickReward()
       else:
           pickRew = orig_pickReward()
       placeRew , placingDist = placeReward()
       # assert ((placeRew >=0) and (pickRew>=0))
       if self.placeCompleted:
           reachRew = 0
           reachDist = 0
       reward = reachRew + pickRew + placeRew
       return [reward, reachRew, reachDist, pickRew, placeRew,
placing
```



Reward function for pick and place task

hand close to obj	obj grasped	pick ever completed	hand close to goal	place ever completed	reward
0	0	*	*	0	-distXY(hand, obj) – distZ(hand,obj)
1	0	0	*	0	-dist(hand, obj) + c1 * closeCmd
1	1	0	*	0	-dist(hand,obj) + c1 * closeCmd + c2 * min(obj.z, targetHt)
1	1	1	*	0	-dist(hand,obj) + c1 * closeCmd + c2 * targetHt + placeRew
1	1	1	0	0	-dist(hand,obj) + c1 * closeCmd + c2 * targetHt + placeRew + c3 * openCmd
*	*	*	*	1	c2 * targetHt + placeRew + c3 * openCmd

place ever completed: |objx – goalx| < c4 and |objy-goaly|< c5 and |objz – goalz| < c6
pick ever completed: objz > targetHt + c7
obj grasp: test on current finger sensors
hand close to obj, hand close to goal: constant x,y,z thresholds
placeDist = |objxy – goalxy|
placeRew = max(0, c8*(c9- placingDist) + c10*(exp(-(placingDist**2)/c11) +exp(-(placingDist**2)/c12)))

Reward function for pick and place task

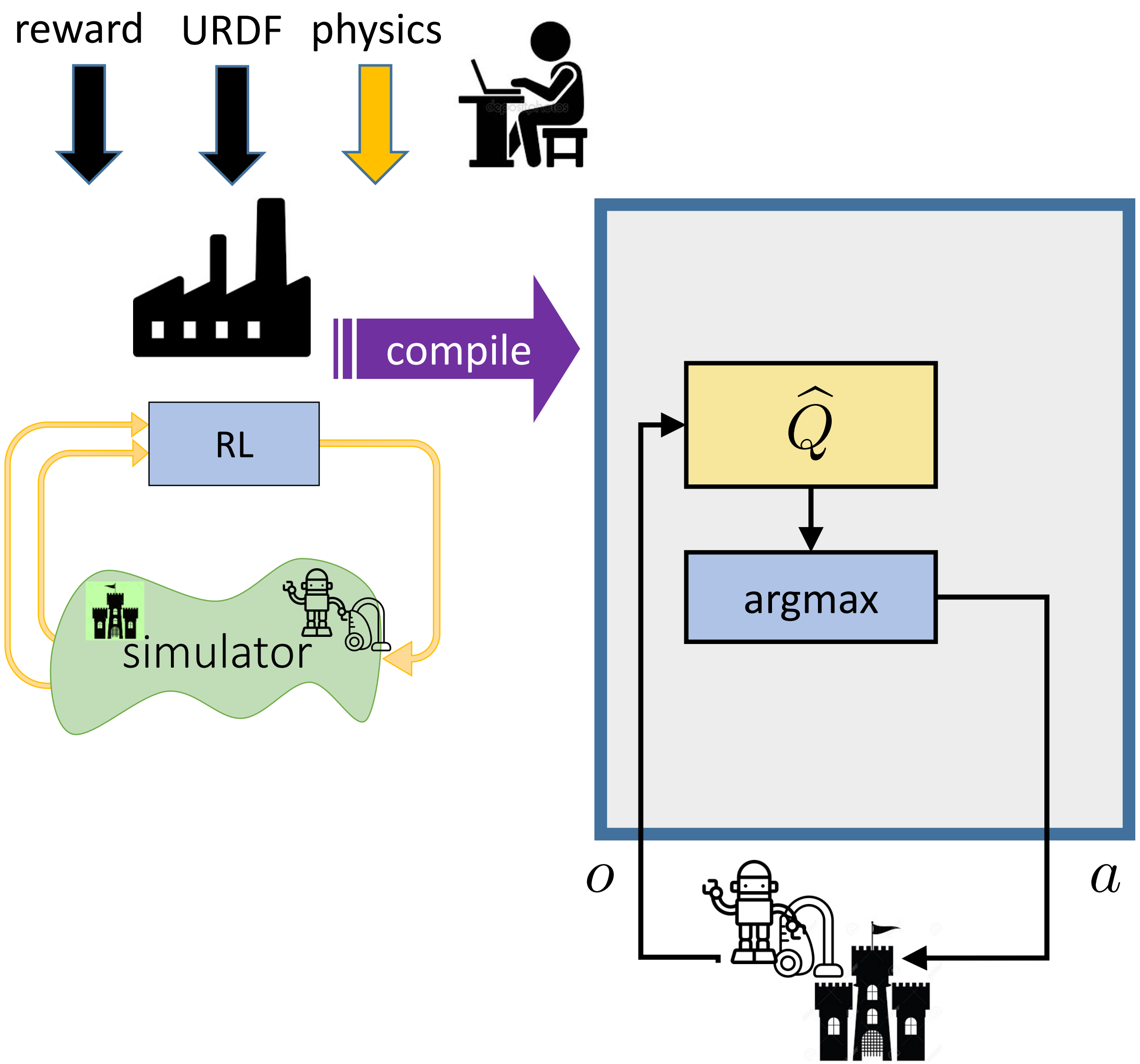
hand close to obj	obj grasped	pick ever completed	hand close to goal	place ever completed	reward
0	0	*	*	0	move toward object
1	0	0	*	0	move toward object + close fingers
1	1	0	*	0	move toward object + close fingers + raise
1	1	1	*	0	move toward object + close fingers + move toward goal
1	1	1	0	0	move toward object + close fingers + move toward goal + open fingers
*	*	*	*	1	move toward goal + open fingers

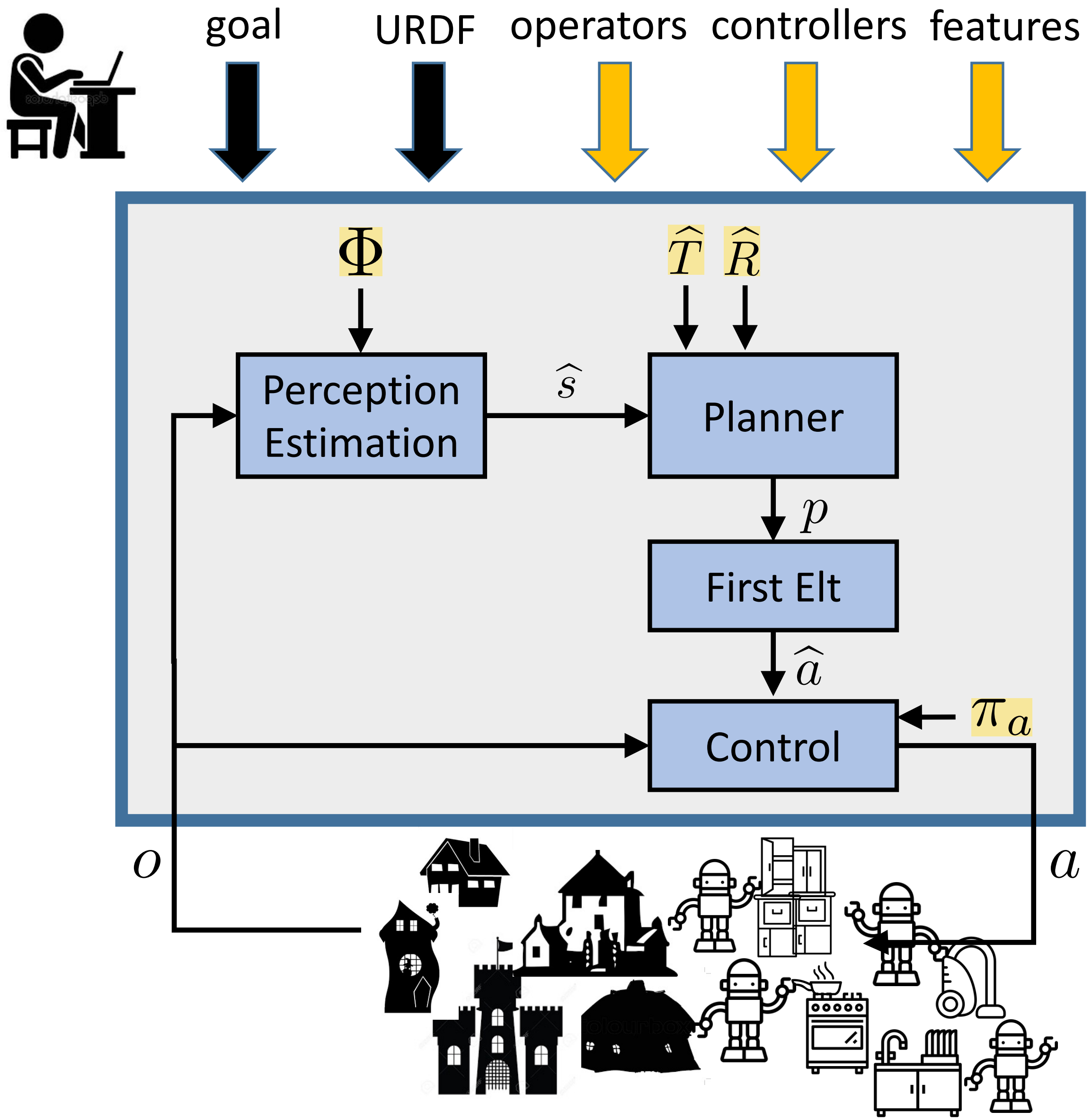
place ever completed: |objx – goalx| < c4 and |objy-goaly|< c5 and |objz – goalz| < c6
pick ever completed: objz > targetHt + c7
obj grasp: test on current finger sensors
hand close to obj, hand close to goal: constant x,y,z thresholds
placeDist = |objxy – goalxy|
placeRew = max(0, c8*(c9- placingDist) + c10*(exp(-(placingDist**2)/c11) +exp(-(placingDist**2)/c12)))

Goal for HPN planner

```
goal = Goal([Pose(target_obj, (0.2, -0.1, 0.5, 3.14)])
```

RL in the factory vs planning in the wild: engineering effort





Models for robot manipulation planning

Raw simulator

- Action space: incremental joint torques or displacements
- Horizon: several 1000
- Heuristic: learned goal-conditioned v
- Exploration needed to learn heuristic

What makes an abstraction useful?

- Makes computational problem much easier
- Doesn’t make solution quality too much worse
- Not too hard to implement / maintain
- Solves most problems you care about

Proposed abstract model

- Action space is
  - **lifted**: independent of particular objects
  - **compositional**: can address huge space of states and goals
- Horizon: hierarchical ~8 mode-level, ~20 motion-level, ~2 controller-level
- Heuristic: domain independent (can be improved by learning)
- Exploration: not needed

Multi-modal motion planning: a useful abstraction for many problems!

Assume:

- robot and objects (for now) are kinematic
- state represented in terms of poses (and c

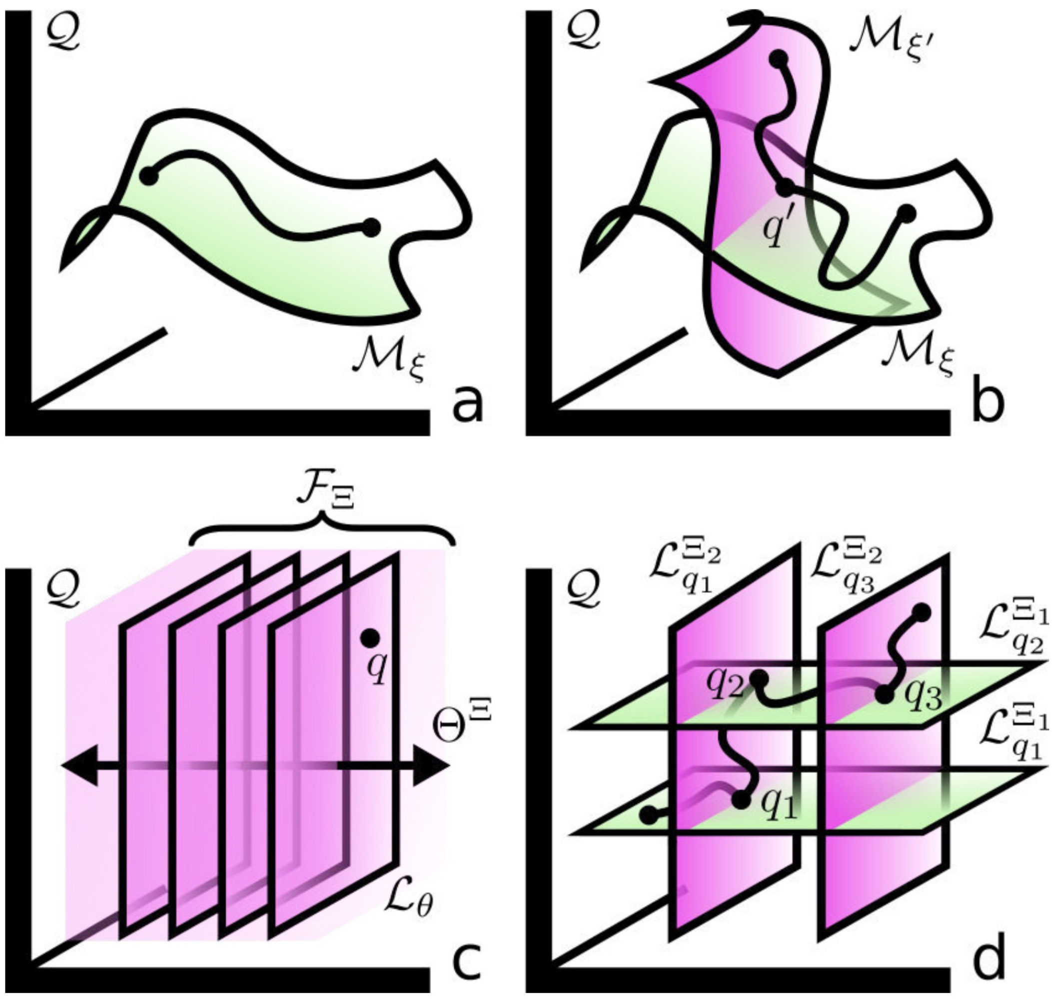
Mode: smooth **dynamics of a small set of state**

- robot moving in free space
- robot moving while holding A in grasp G a
- …

Mode family: set of modes with same changing

Mode family specified by:

- changeable params
- controller for entering, moving through, a
- testable description of set of world configu and into which it can be exited



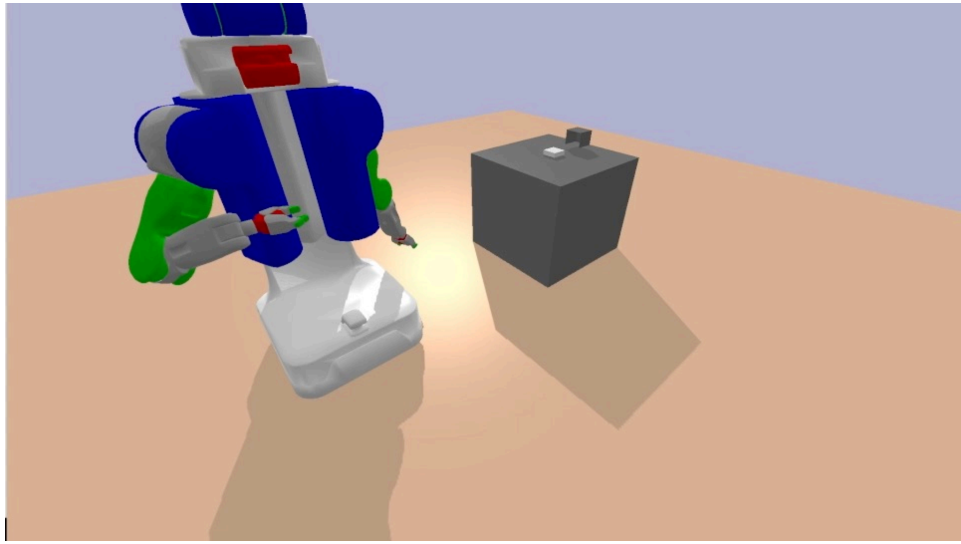
Kingston, Wells, Moll, Kavraki ICRA 2020

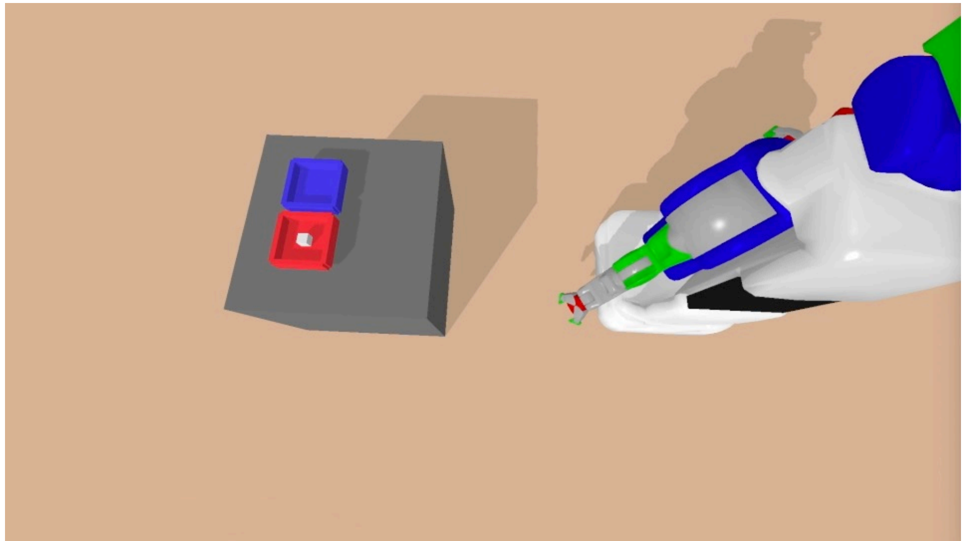
Multi-modal motion planning:
a useful abstraction for many problems!

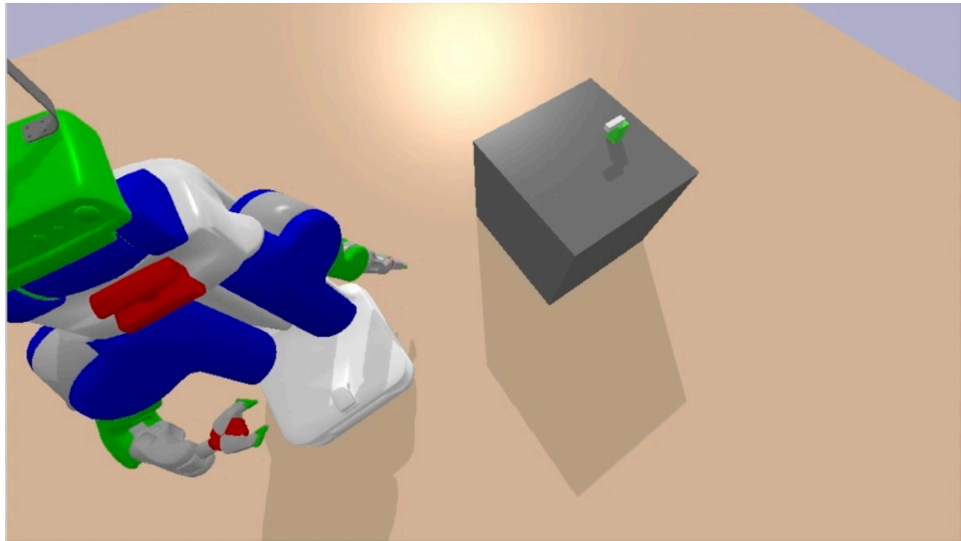
We model meta-world as a **lifted** MMMP with mode families:

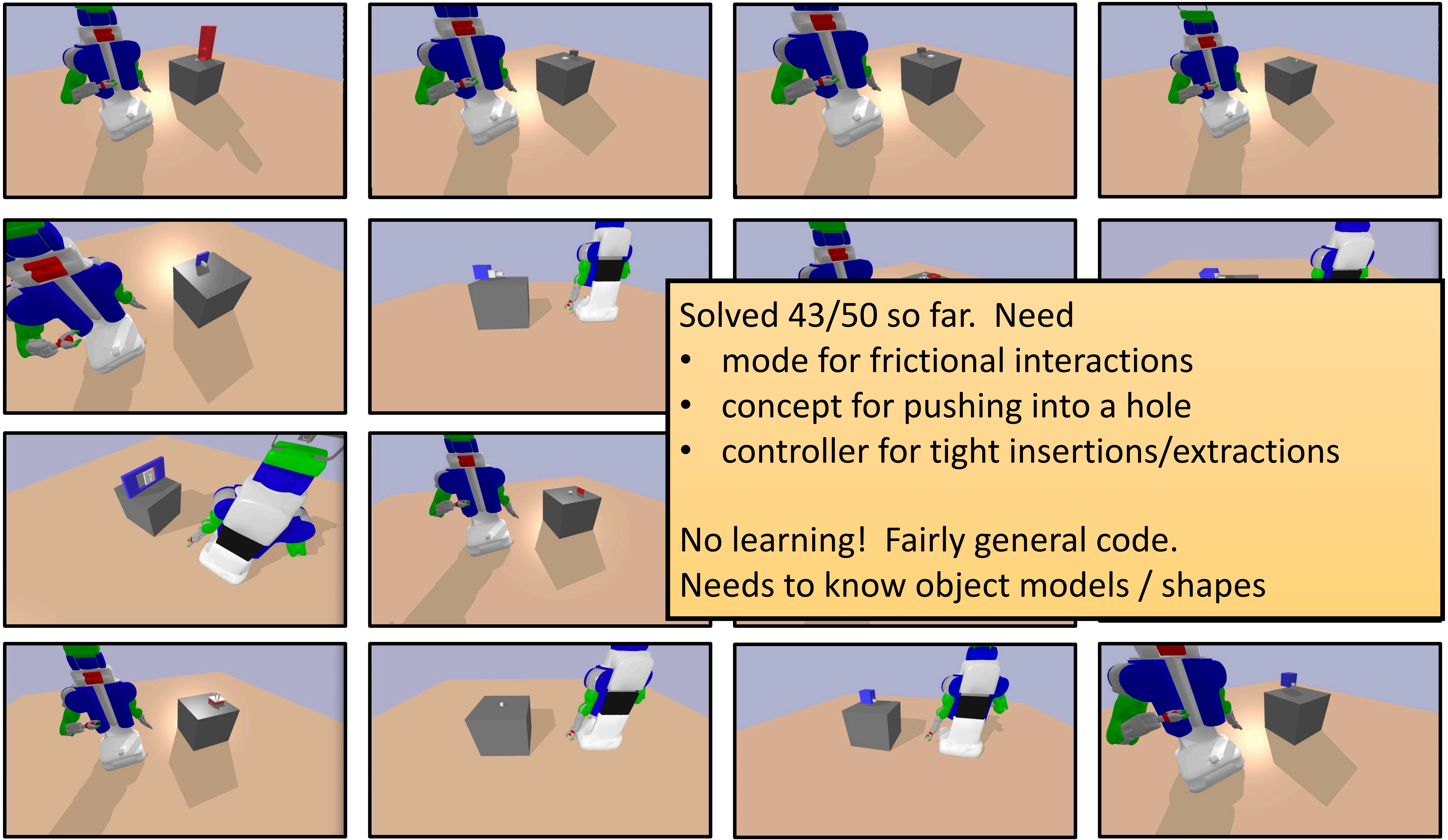
- free-space motion
- pick; move-holding; place
- touch; push; disengage
- grasp; operate; ungrasp

Easy to add new ones!









Solved 43/50 so far. Need

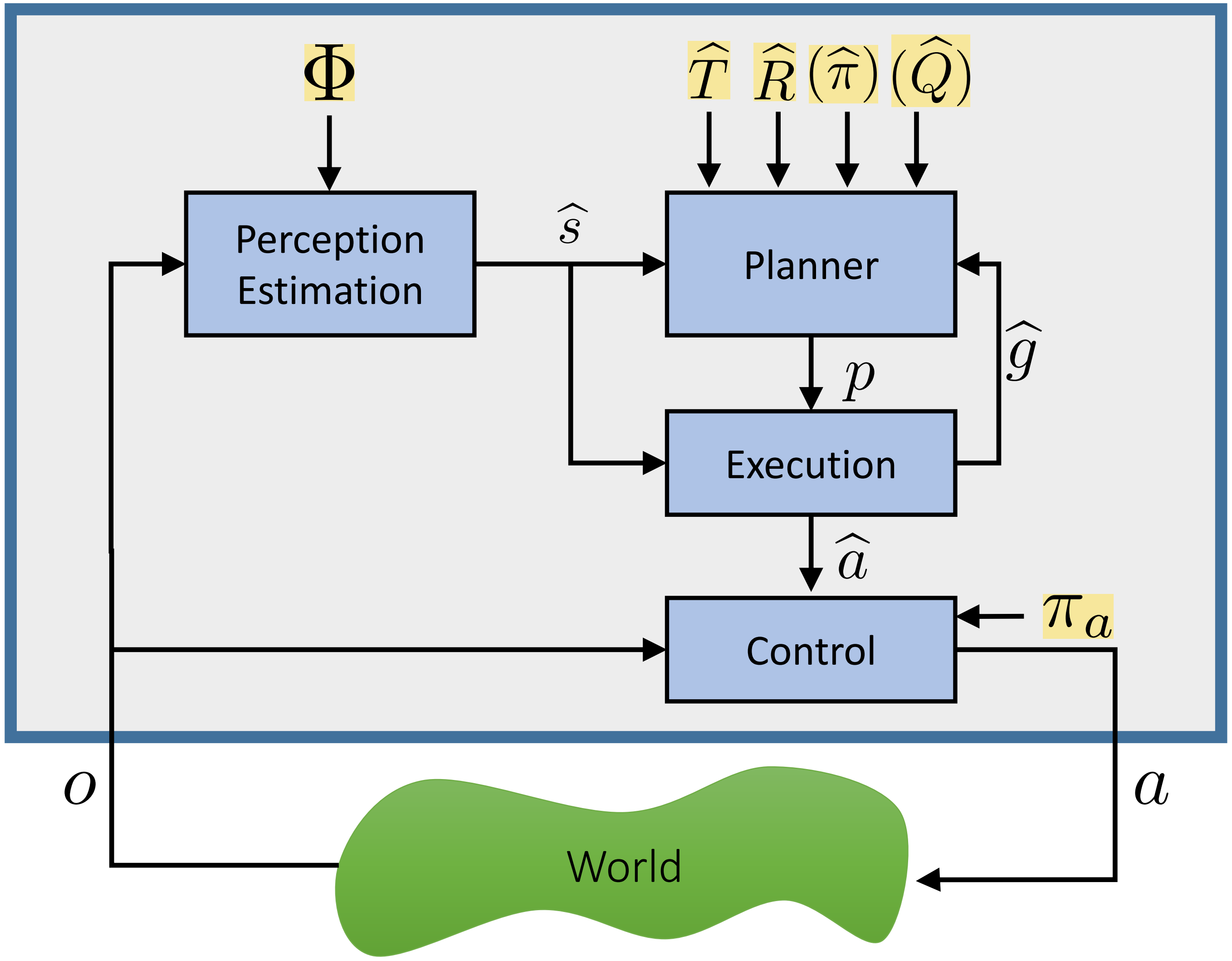
- mode for frictional interactions
- concept for pushing into a hole
- controller for tight insertions/extractions

No learning! Fairly general code.
Needs to know object models / shapes

Built-in mechanisms: planner, hierarchical execution

Modules and models:

- control
  - basic pick and place controllers
- perception
  - object segmenters and
  - simple shape matching
- planning
  - operator models for basic controllers

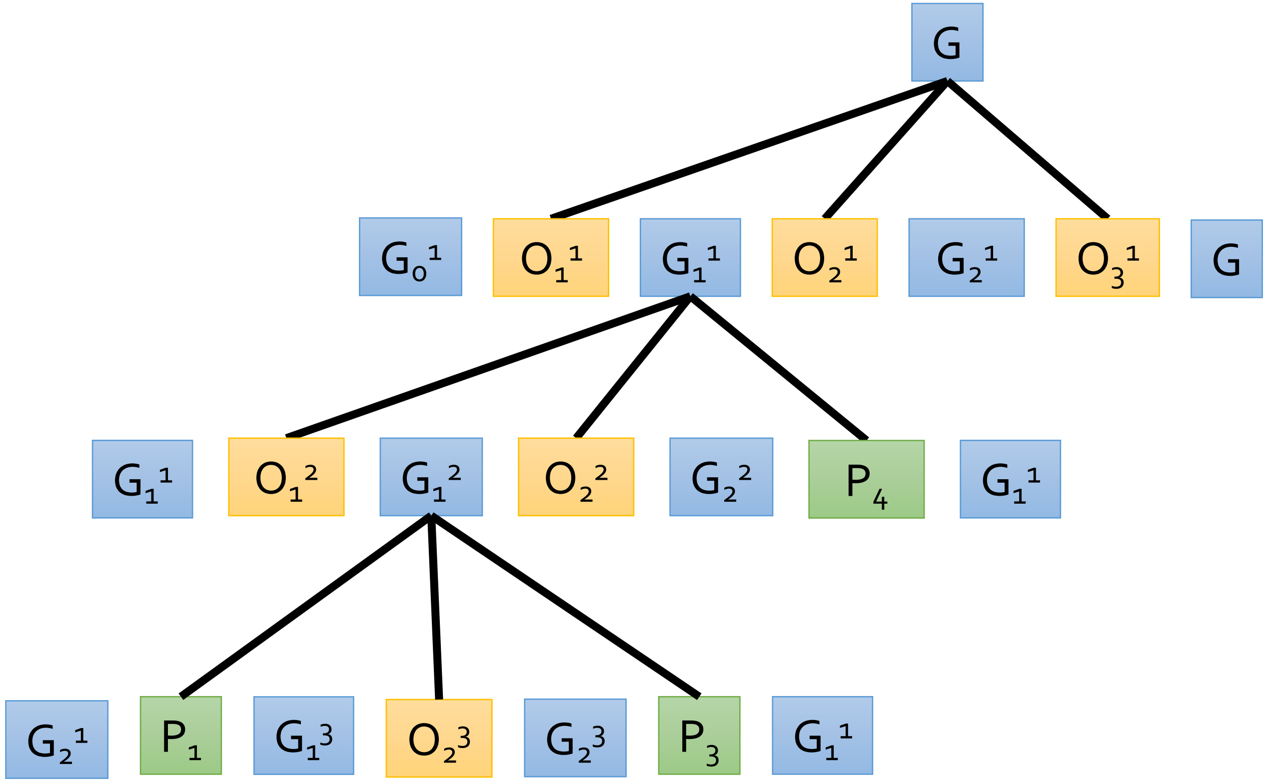


Hierarchical planning in the now

many short plans

optimistic execution

flexible reconsideration and replanning

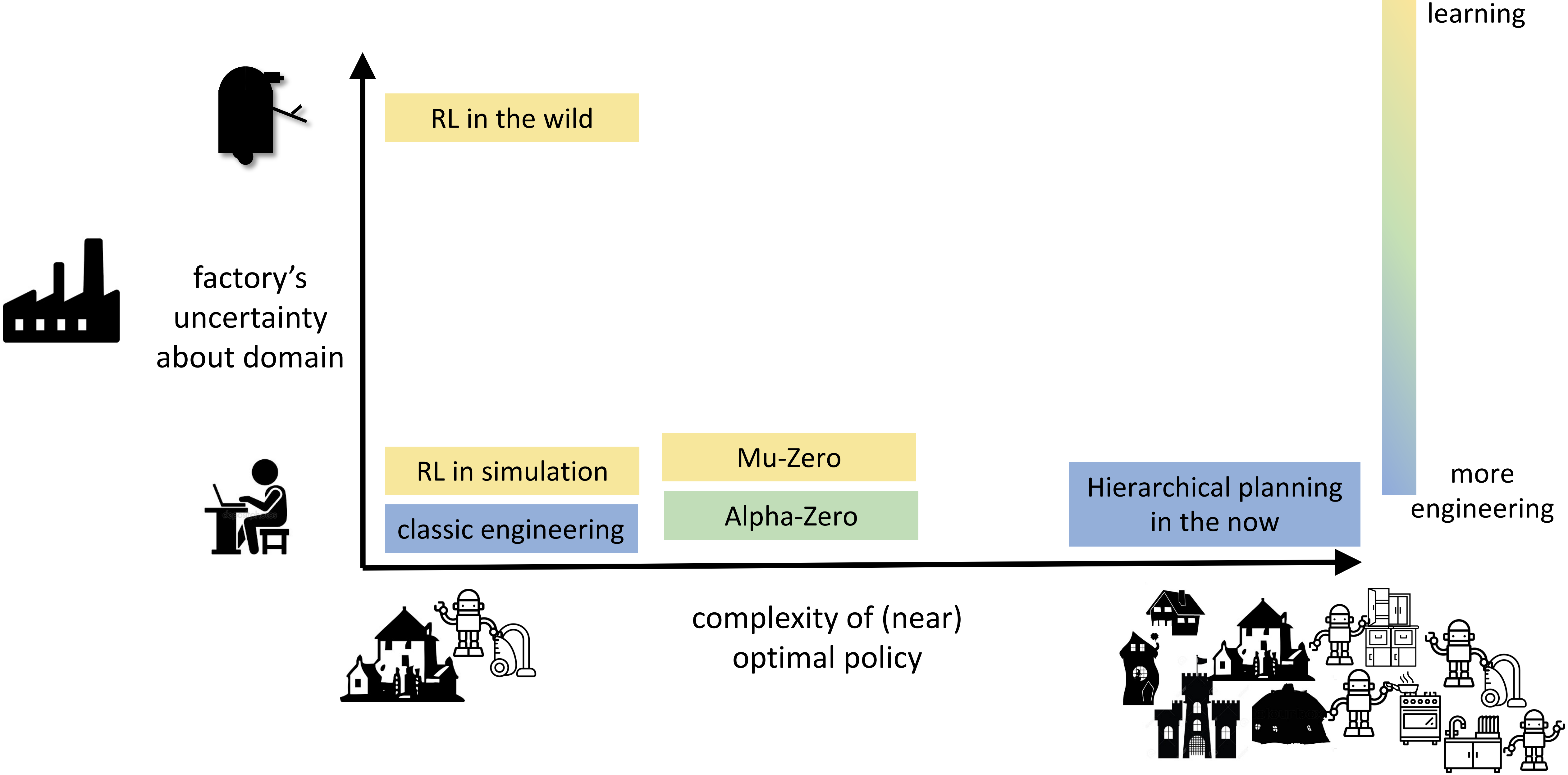


LPK and Lozano-Perez, IJRR 2013

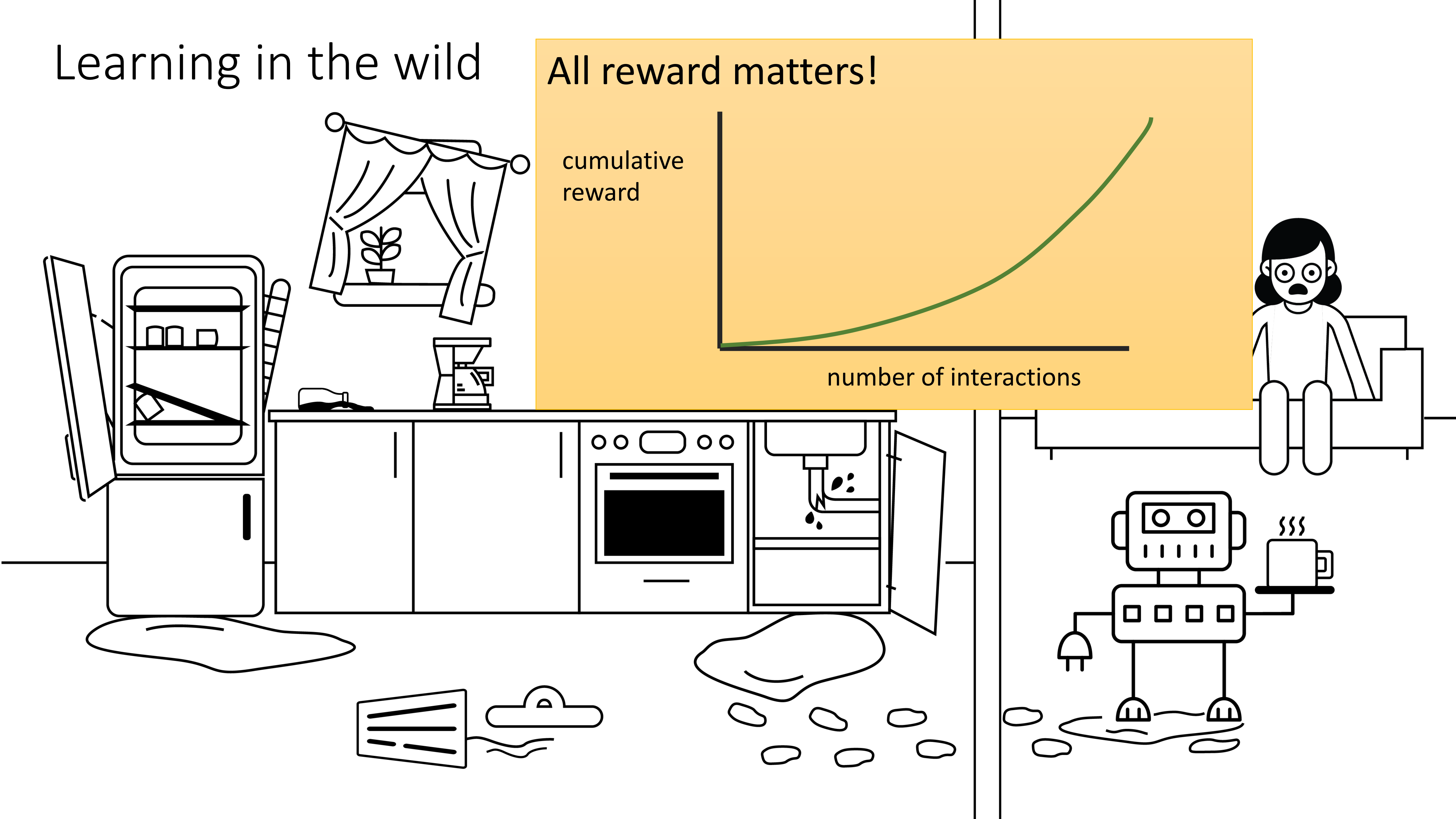
Hierarchical planning in the now

Only learning was done by lpk and tlp, not the robot!

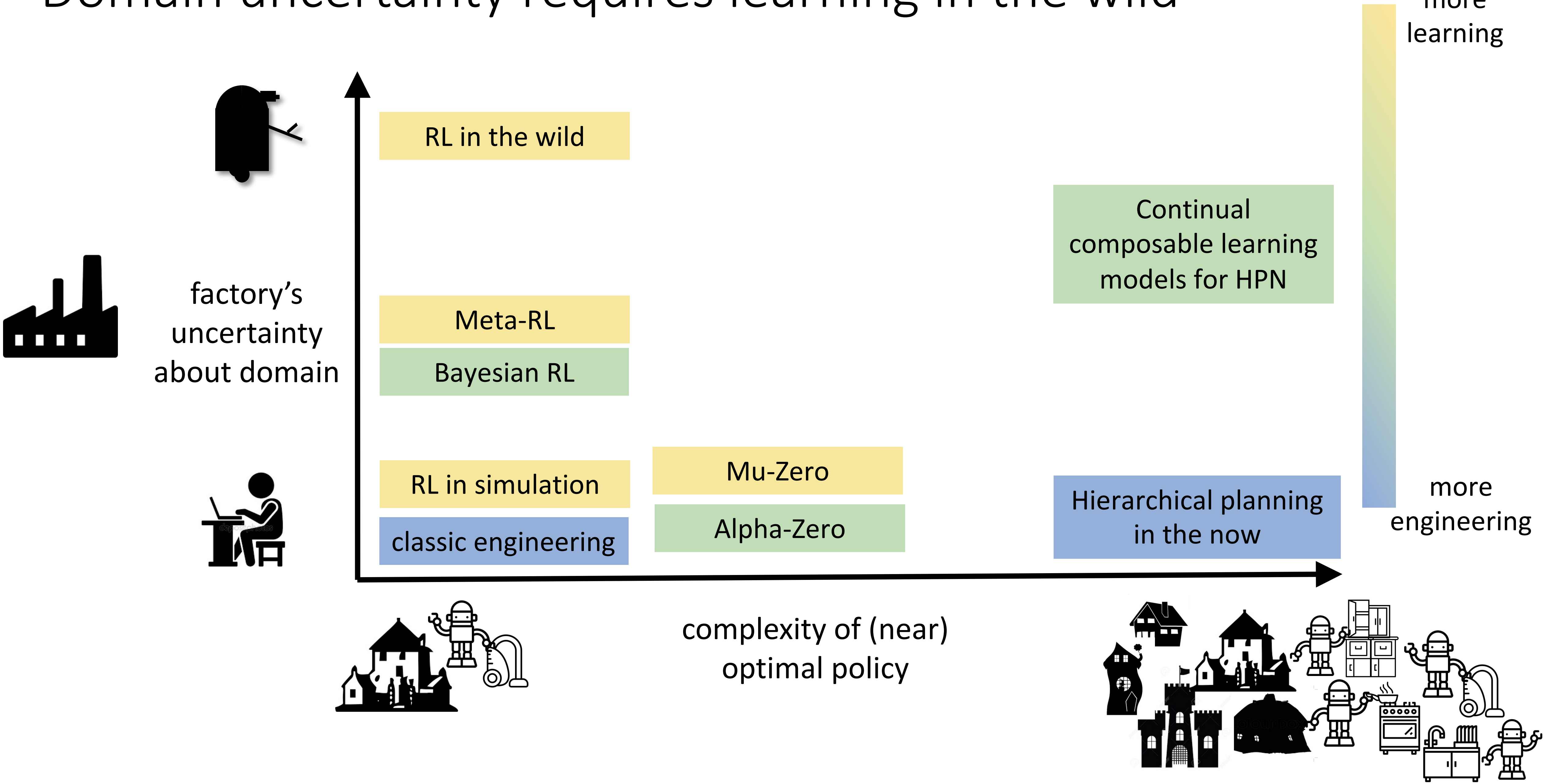
Domain uncertainty requires learning in the wild



Learning in the wild



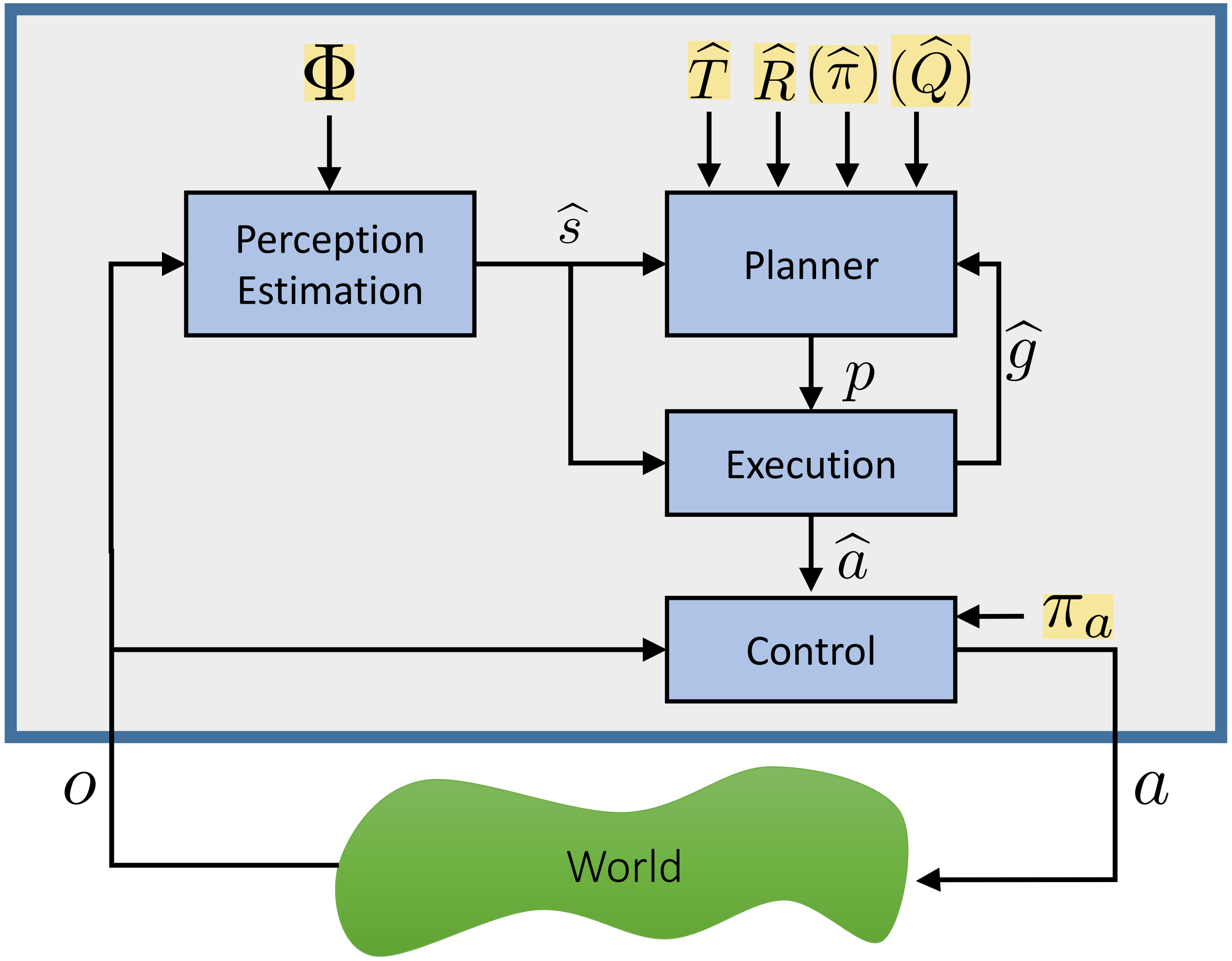
Domain uncertainty requires learning in the wild



Now, what can we learn?

Modules and models:

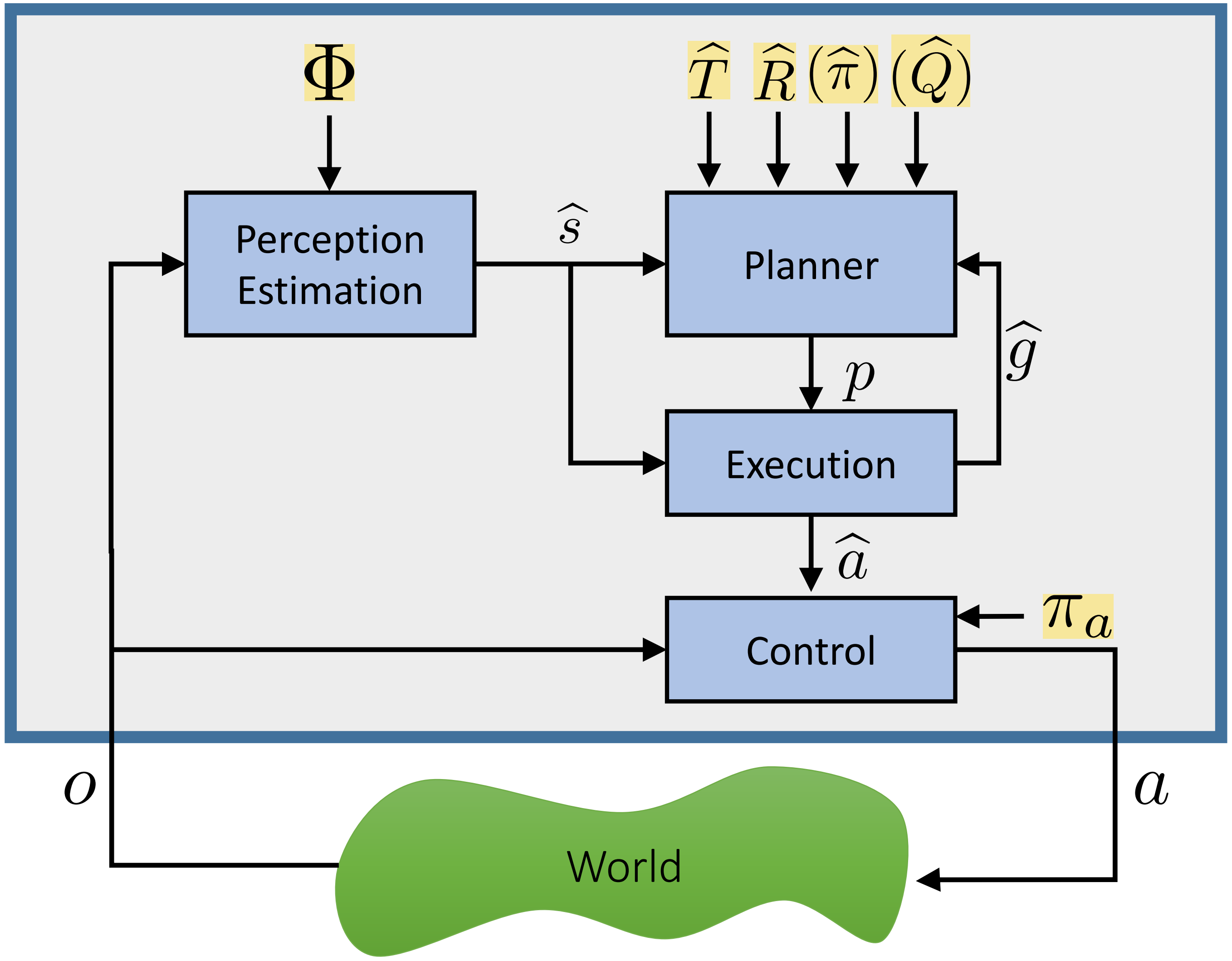
- control
  - sensorimotor controllers
- perception
  - object segmenters and
  - feature detectors
- execution
  - abstract **partial** polices
- planner
  - operator models for controllers and policies



Now, what can we learn?

Modules and models:

- control
    - sensorimotor controllers
- perception
    - object segmenters and
    - feature detectors
- execution
    - abstract **partial** polices
- planner
    - **operator models for new controllers and policies**

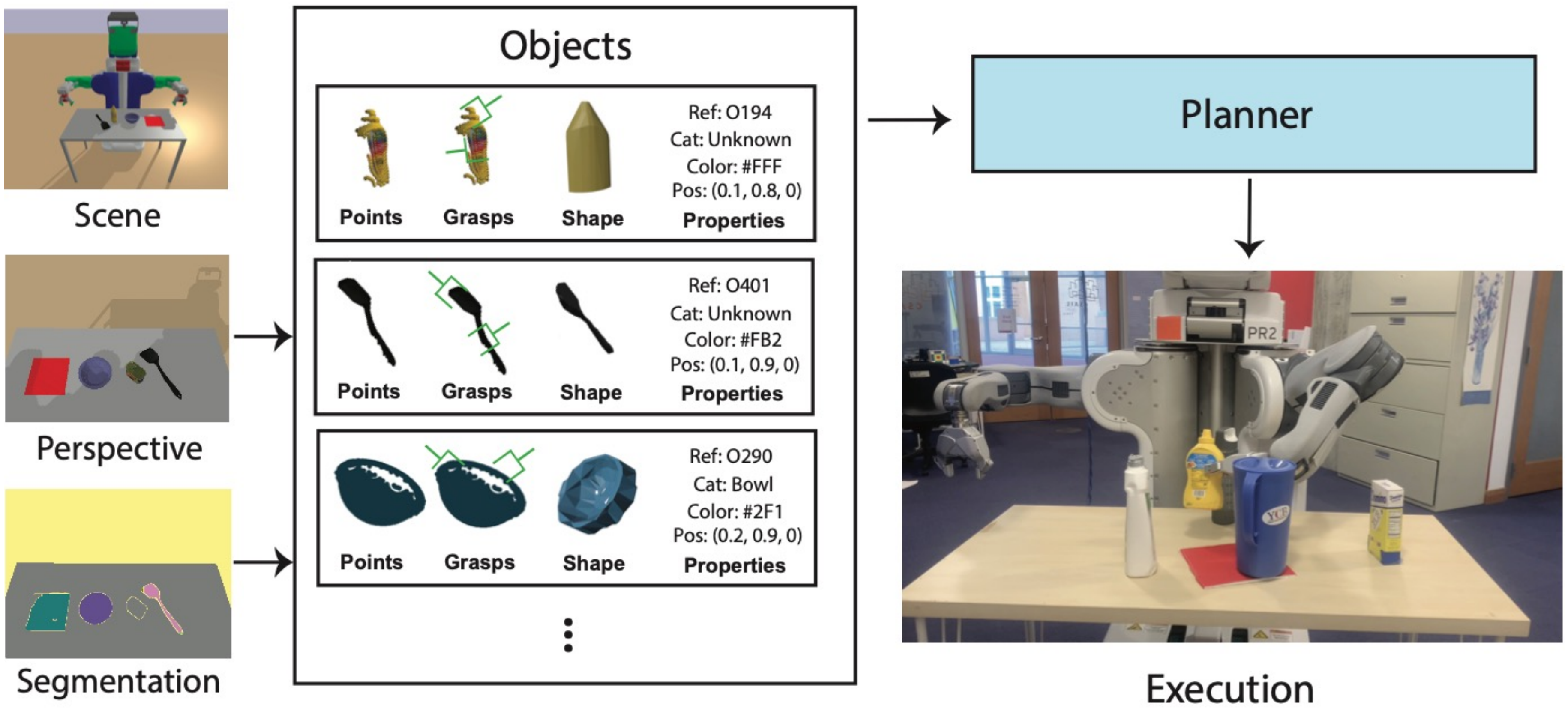


Curtis, Fang, Garrett, LPK, Lozano-Perez

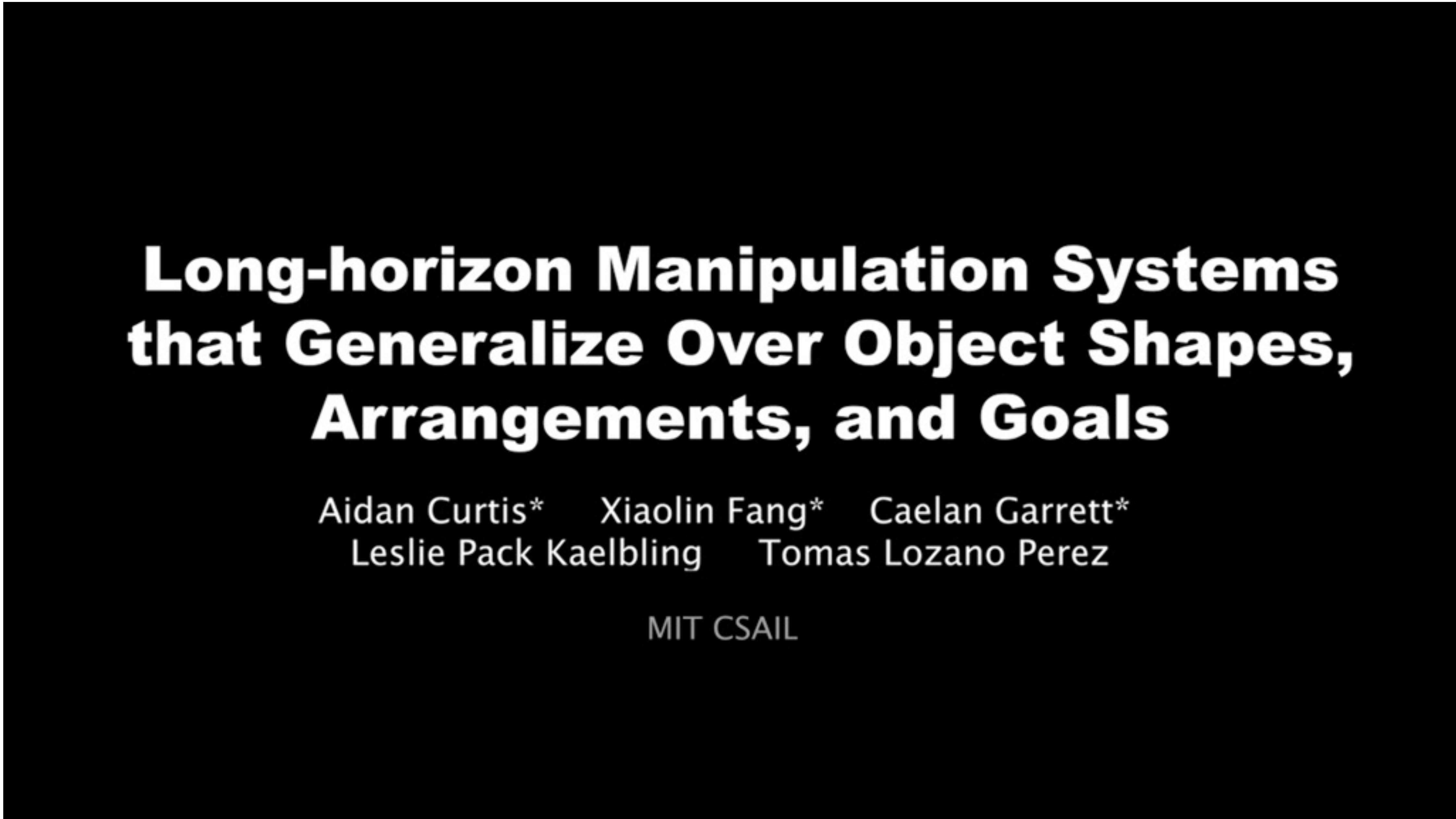
M0M: Manipulation with 0 Models

We
- believe in objects (for now restricted to be rigid)
- know the robot kinematics, and an abstract multi-modal model for pick/place/push

Use **pre-trained** perception modules
- **UOIS-Net** for class agnostic (!) segmentation [Xie et al, 2020]
- **GPD** to predict grasps [Gualtieri et al, 2016]
- **MSN** for point-cloud shape completion [Liu et al, 2020]
- **MaskRCNN** for some category detections [He et al, 2017]



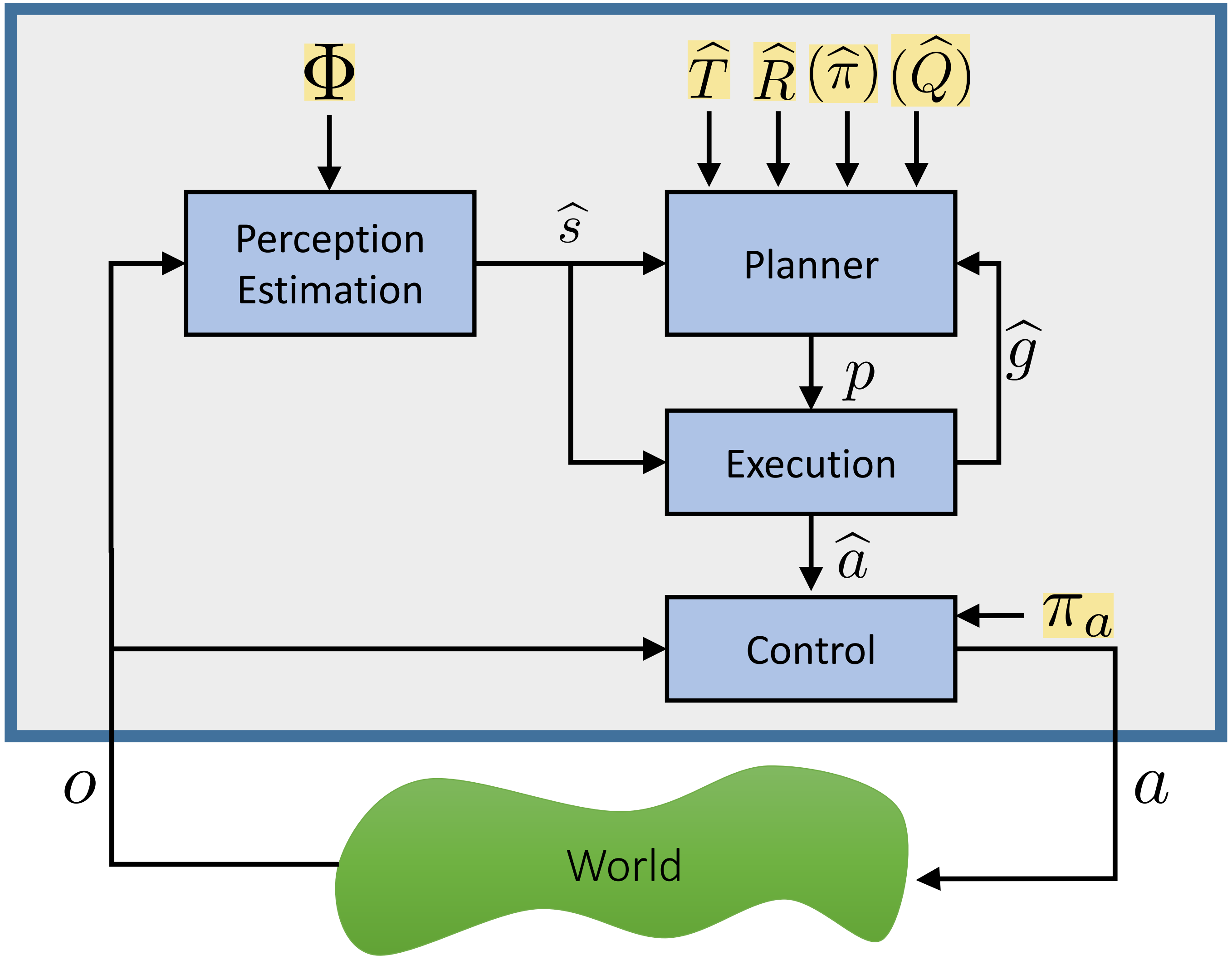
Look M0M, no additional learning!



Now, what can we learn?

Modules and models:

- control
  - sensorimotor controllers
- perception
  - object segmenters and
  - feature detectors
- execution
  - abstract **partial** polices
- planner
  - operator models for new controllers and policies



How can a **competent** robot acquire a new ability?

- Learn new primitive skill
  - Examples: Cutting, pushing, stirring, pouring, throwing
  - Closed-loop low-level policy intended to achieve some objective, possibly parameterized

Most robot learning: assume given

- Add that skill to existing skill set to accomplish new goals!
  - For flexibility, use a general-purpose planner
  - Learn description of skill's preconditions and effects
  - Representation should generalize over objects, locations, etc.

Our focus

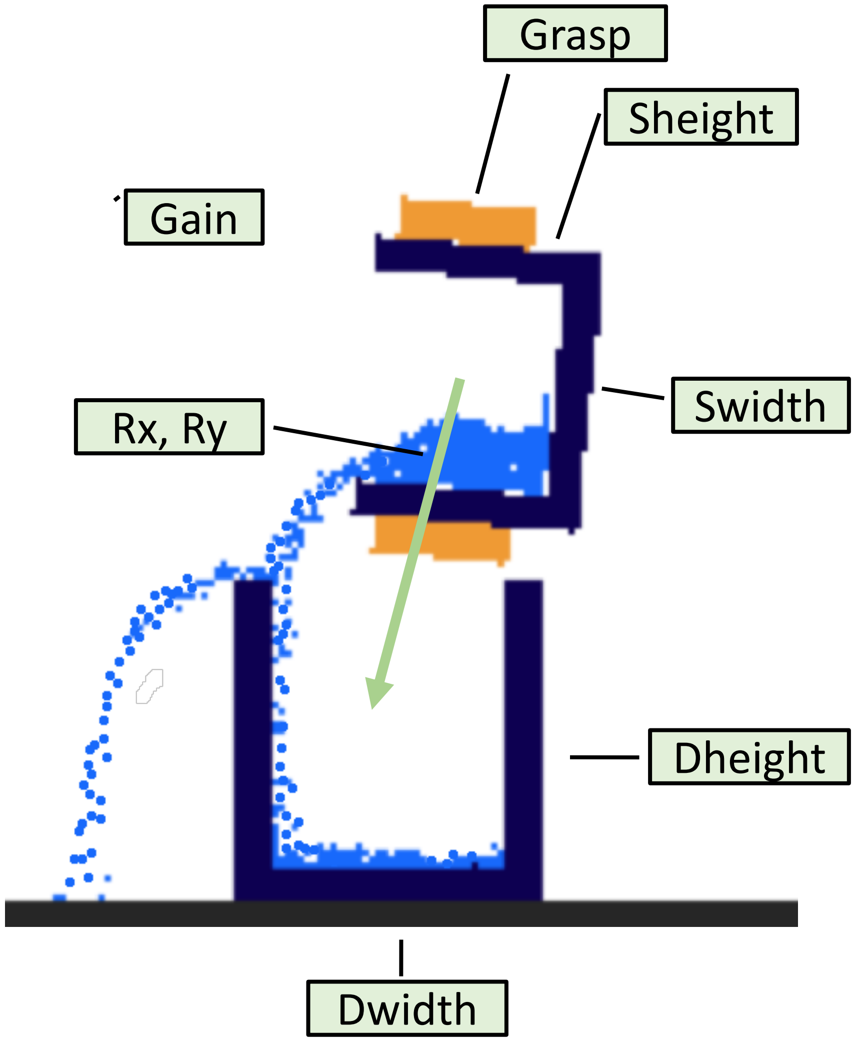
Operator description: when will **skill** achieve **result**?
Generalized idea of mode switch

Result: Contains(Dest, Liquid)

Skill: Pour(**Gain**)

Preconditions:

- Contains(Source, Liquid)
- Holding(Source, **Grasp**)
- Shape(Source) = (**Swidth, Sheight**)
- Shape(Dest) = (**Dwidth, Dheight**)
- RelPose(Source, Dest) = (**Rx, Ry**)
- **Constraint(Sw, Sh, Dw, Dh, Rx, Ry, Grasp, Gain)**

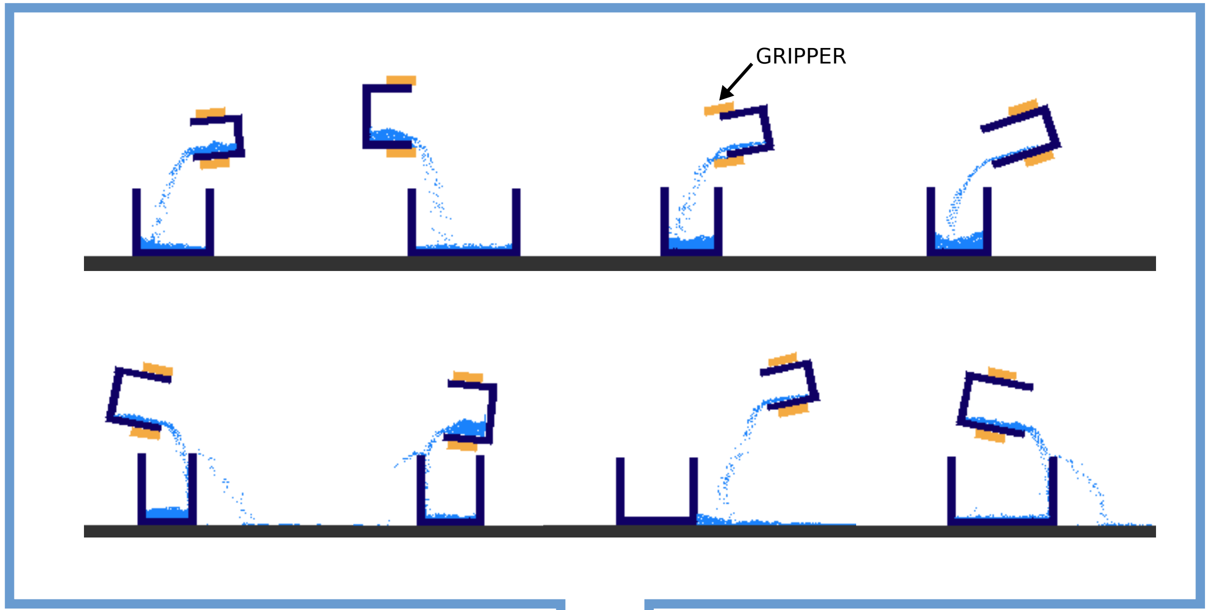


Wang, Garrett, LPK, Lozano-Perez, IROS 2018

Learning the operator constraint: supervised training

$$\text{constraint}(\theta) \equiv g(\theta) > 0$$

labeled training data



Sw, Sh, Dw, Dh, Rx, Ry, Grasp, Gain

θ

Gaussian Process Regression

score

$g(\theta)$

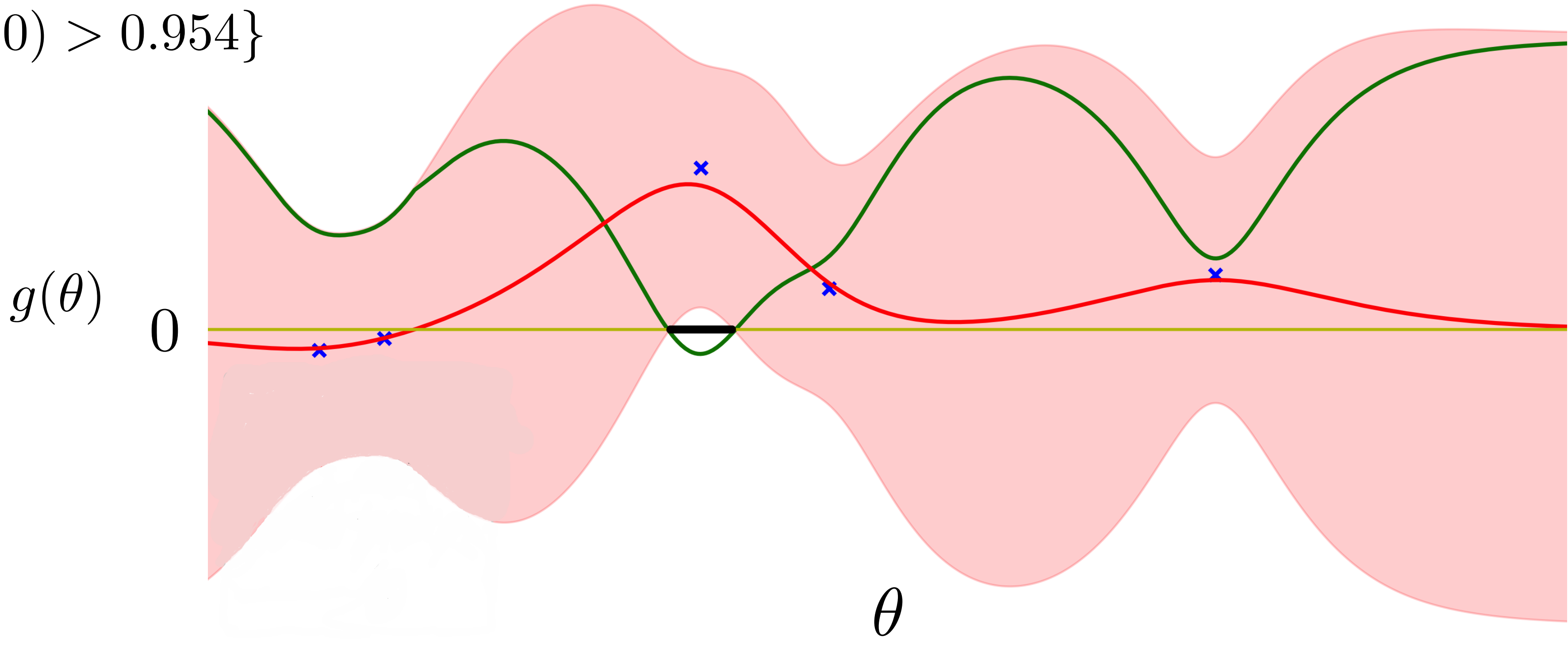
Gaussian process regression

Represent distribution over functions!

$$\text{constraint}(\theta) \equiv g(\theta) > 0$$

- ✕ : observations $(\theta_i, g(\theta_i))$
- — : mean $\mu(\theta)$
- ■ : stdev $\mu(\theta) \pm 2\sigma(\theta)$
- — : high probability super level set

$$\{\theta \mid P(g(\theta) > 0) > 0.954\}$$



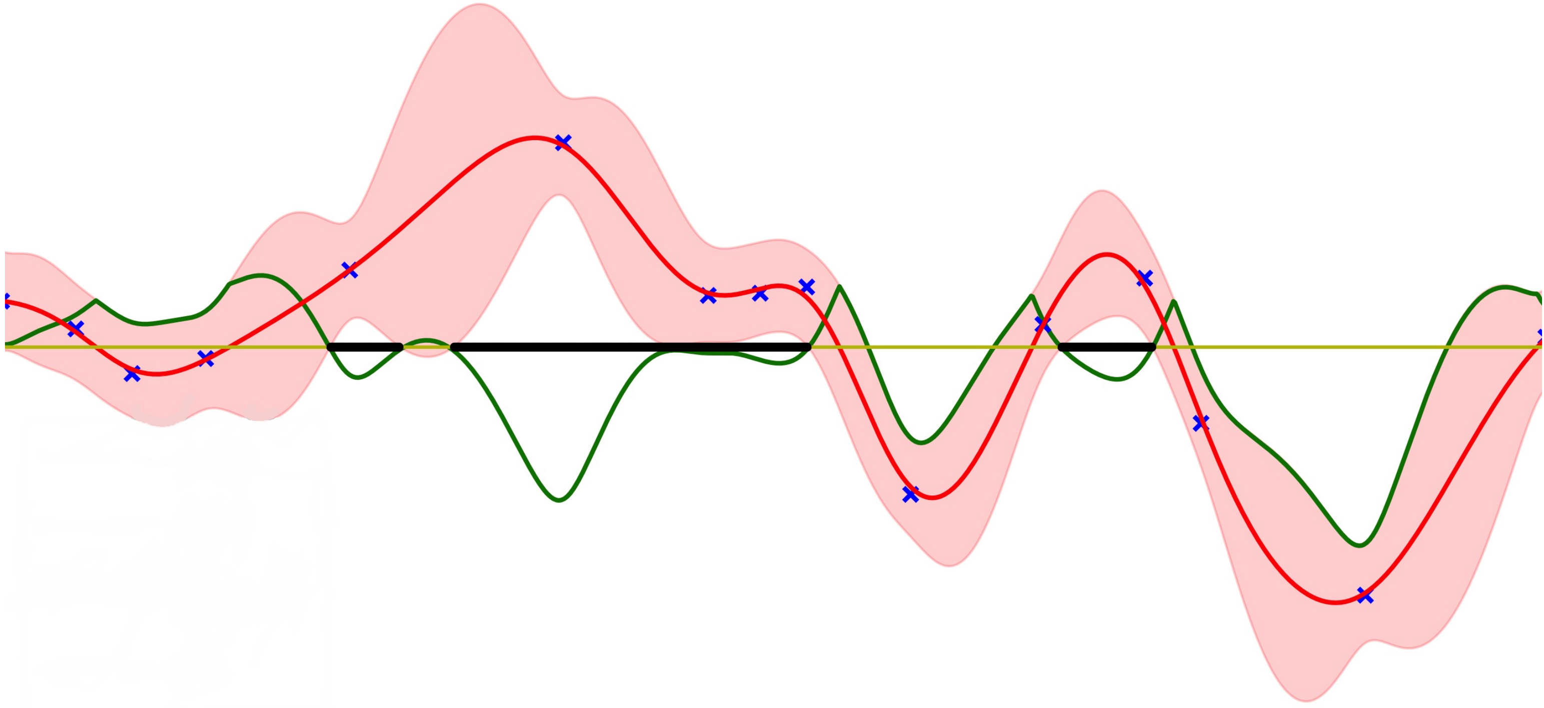
Active learning: robot experience is expensive!

Try pouring in situations that will give us the **most useful** information

- sample to maximize **acquisition function**

$$\phi(\theta) = 2\sigma(\theta) - |\mu(\theta)|$$

- high values when
  - mean is close to 0
    (near a boundary)
  - stdev is high
    (uncertain about the value)
- —: $\phi(\theta)$

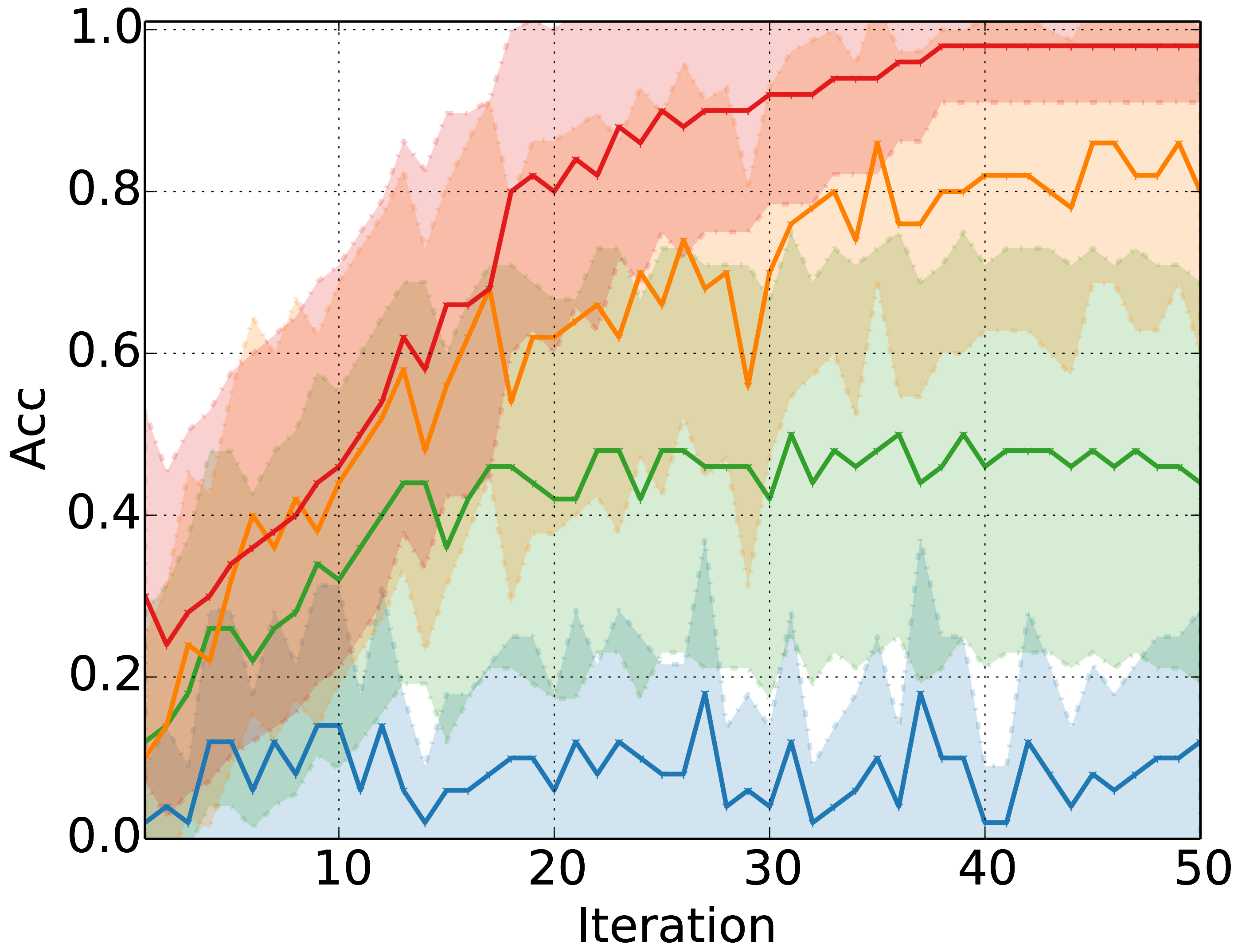


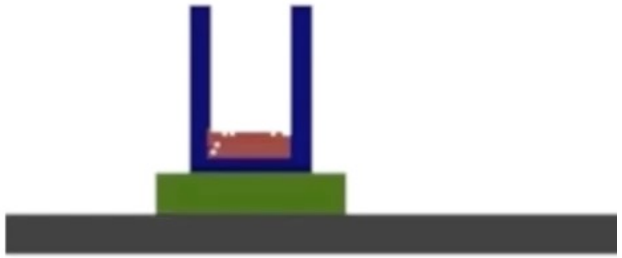
Straddle algorithm, Bryan et al, NIPS 2016

GP learning is data efficient and better for sampling!

Percent successful **pouring** actions as a function of the **number of training examples**

- : randomly chosen
- : neural network classifier
- : neural network regression
- : GP active learning

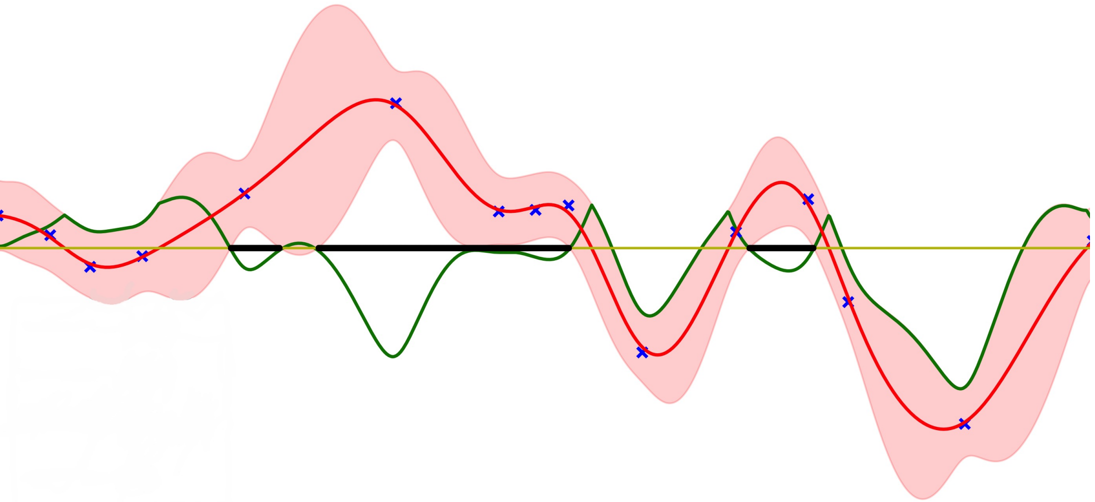


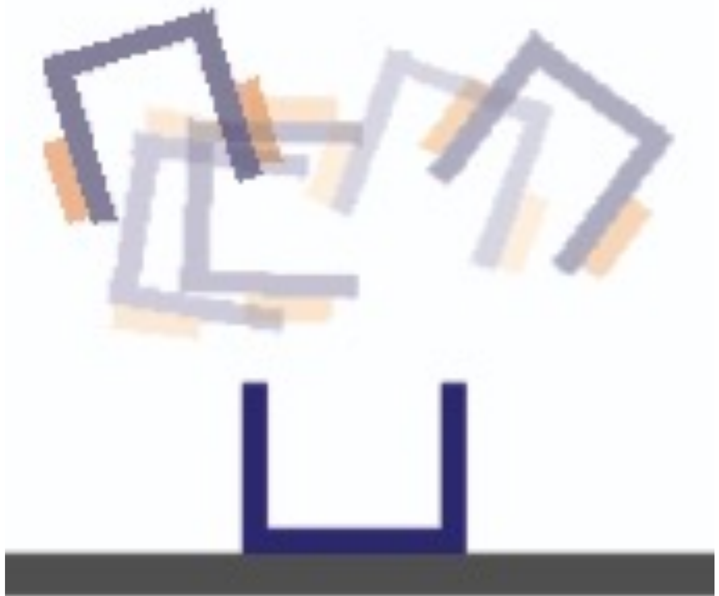


Sampling for planning: quality and diversity

Given values for some parameters, sample values of the others so that:

- action is likely to be successful
  - start with most likely to succeed: $\arg\max_{\theta} \frac{\mu(\theta)}{\sigma(\theta)}$
  - continue with guided rejection sampling in super level set
- action differs from previous attempts $D(S) = \log\det(\Xi^S + I)$





Training on real robot is expensive!

Note that we do grasp and motion planning during data acquisition!

Integrated results on real robot

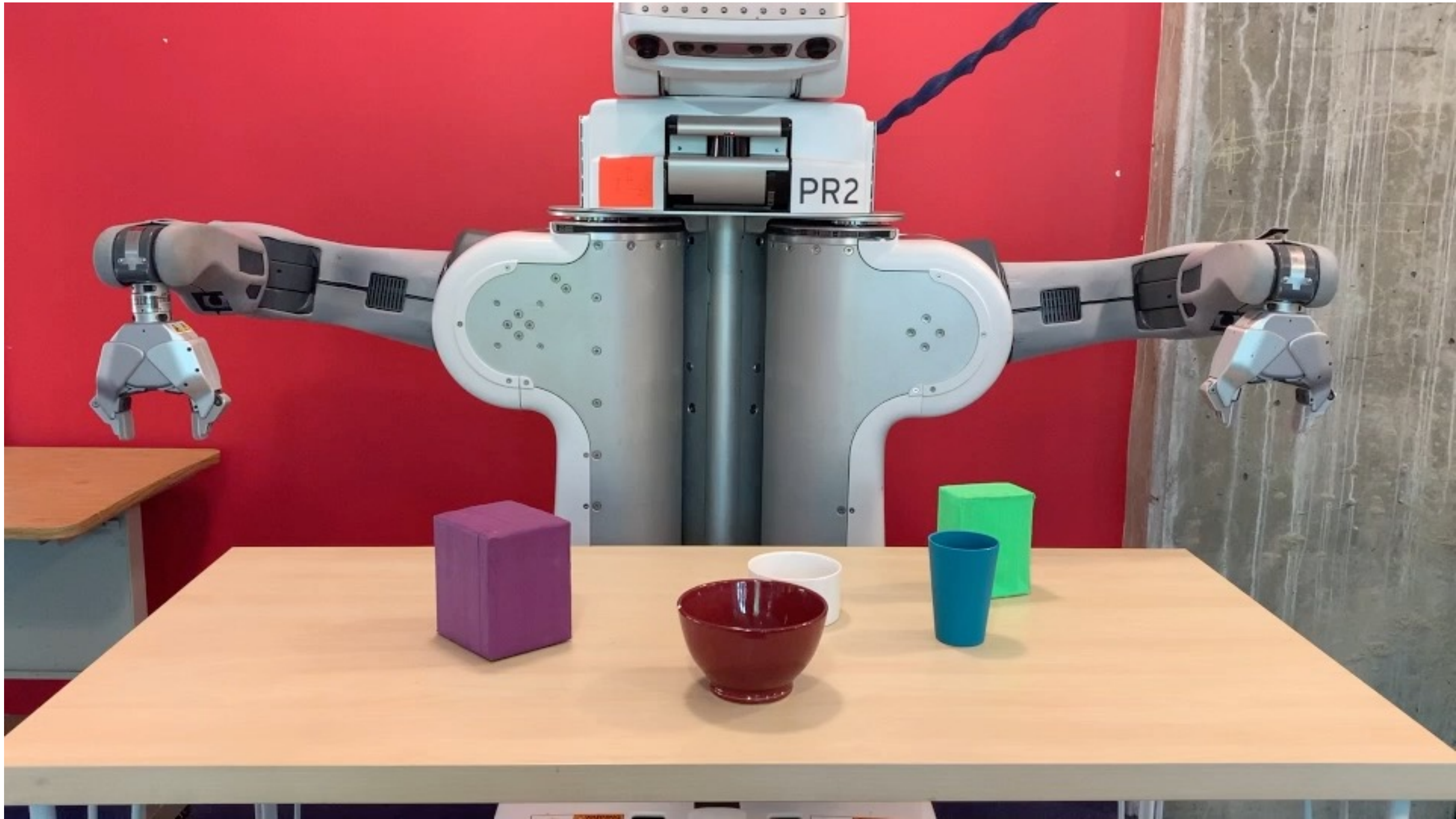
Substantial variability in

- starting arrangement
- goal

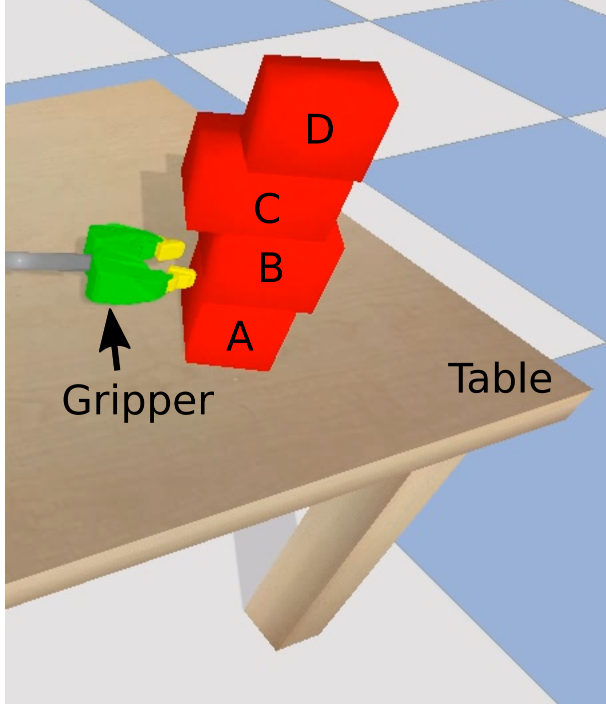
Given pick/place operators

Learned pour and push

PDDLStream planner

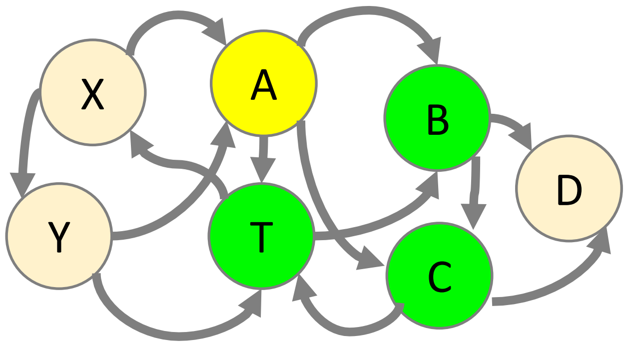


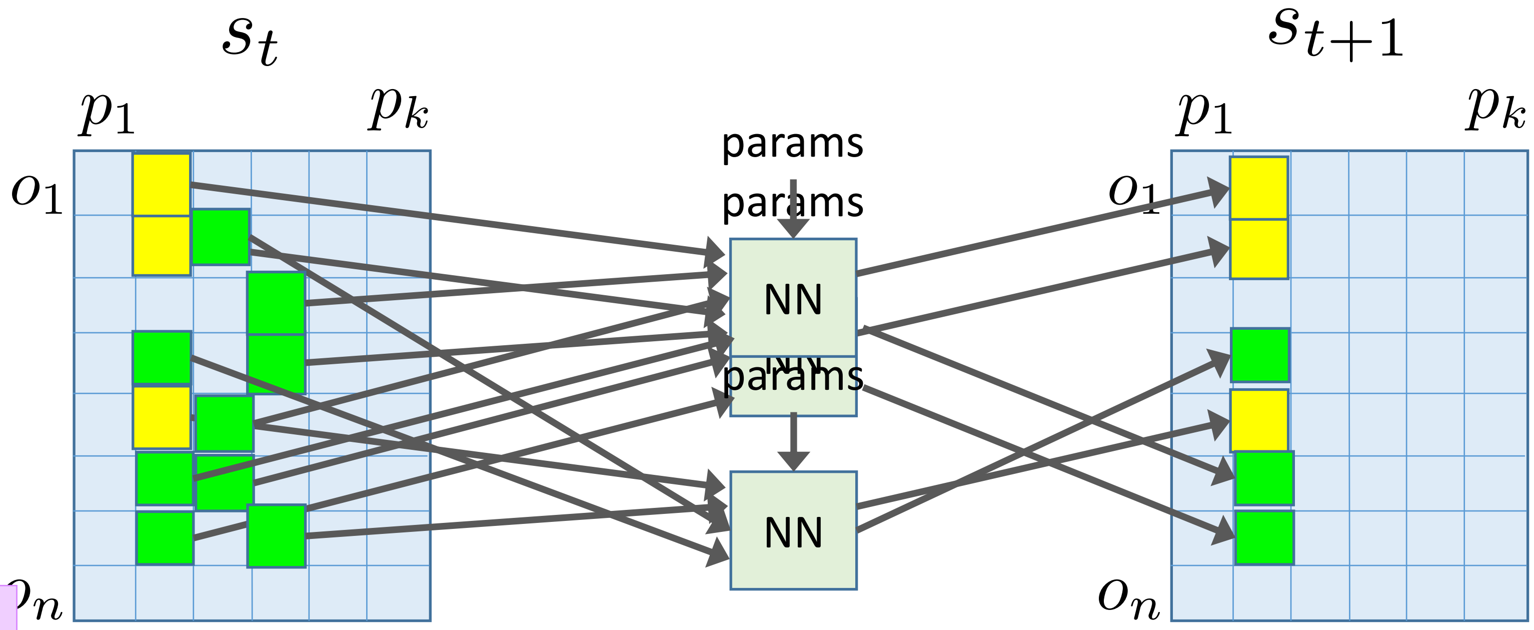
Deictic rule: "lifted" neural network



Like a graph neural-network, in that

- finite size neural network "kernel"
- can be applied to inputs and outputs of different dimension
- generalizes over object identity



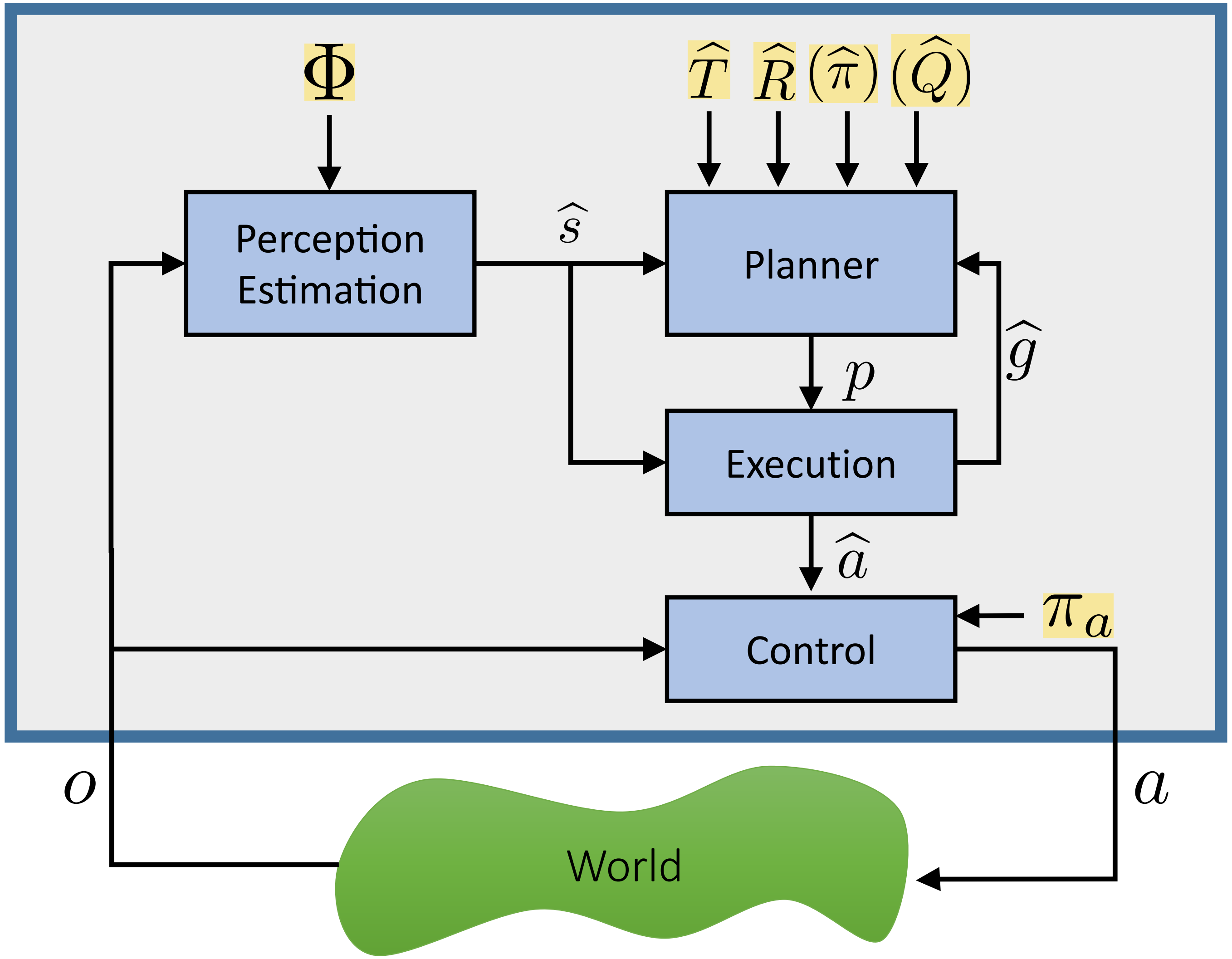


Xia, Wang, Allen, Silver, K, ICLR19

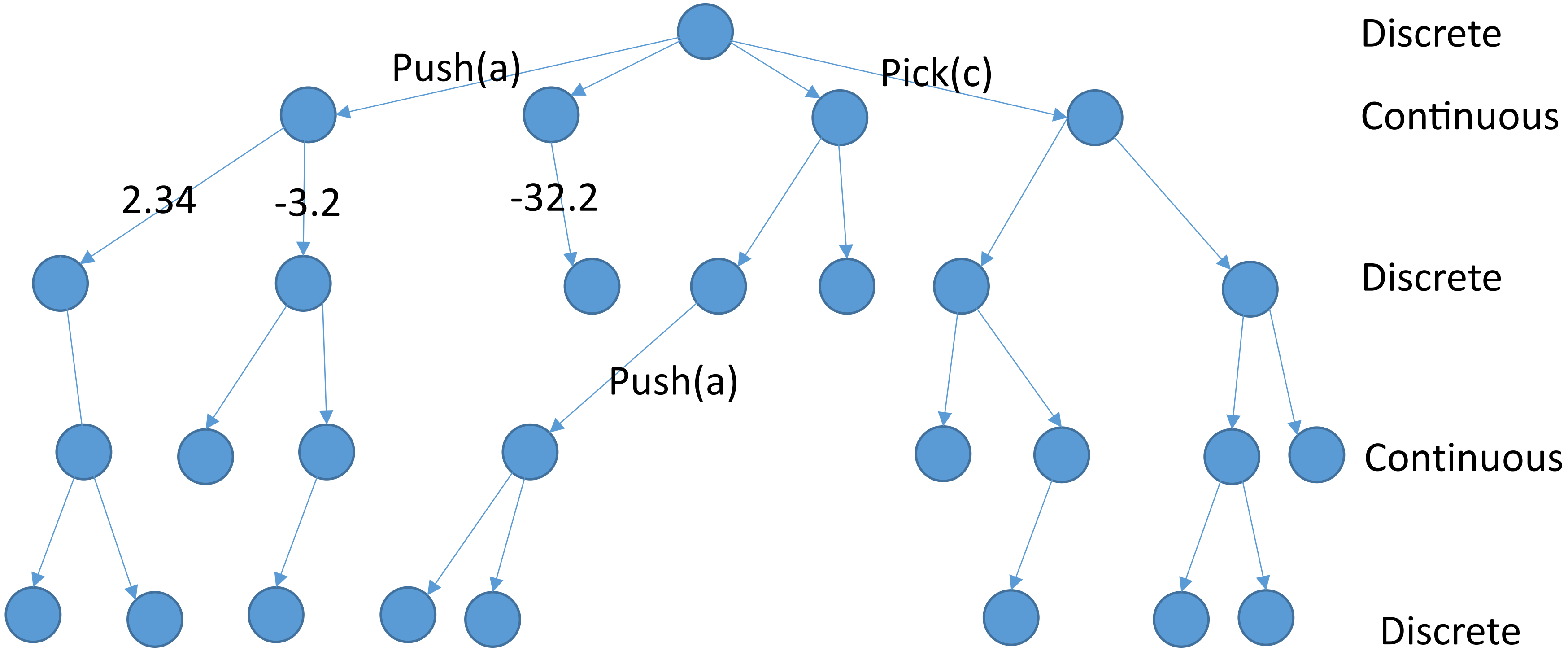
Now, what can we learn?

Modules and models:

- control
  - sensorimotor controllers
- perception
  - object segmenters and
  - feature detectors
- execution
  - abstract **partial** polices
- planner
  - operator models for controllers and policies
  - search control Q



One class of task and motion planning methods: sample-based forward search



Branching factor: huge!

Discrete layer

- operation, objects

Continuous layer

- values in R^d

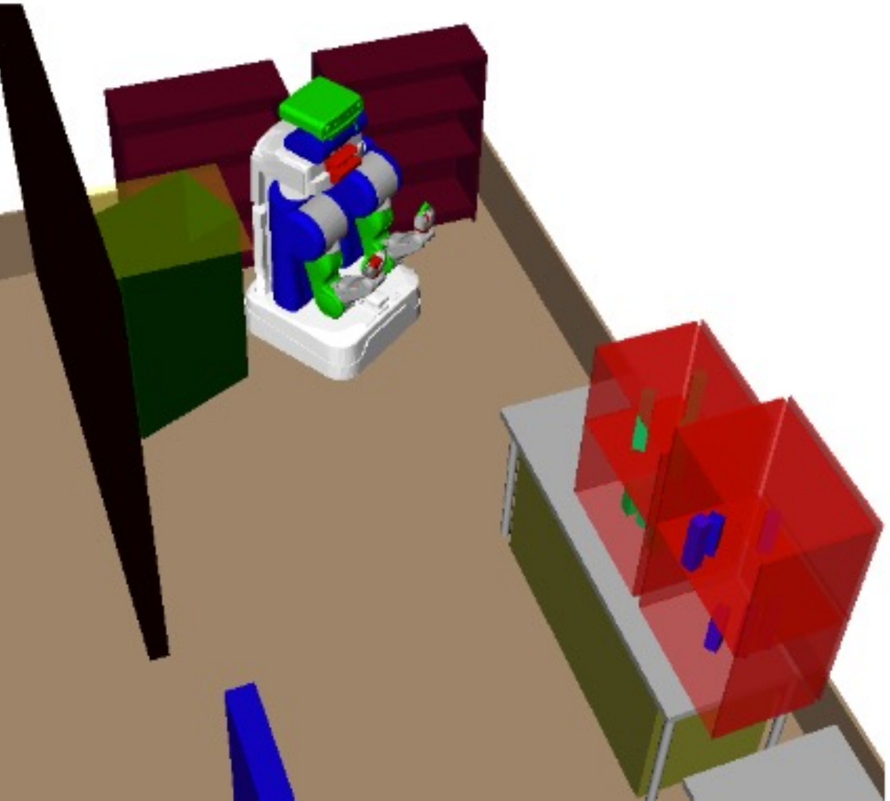
Node expansion: expensive!

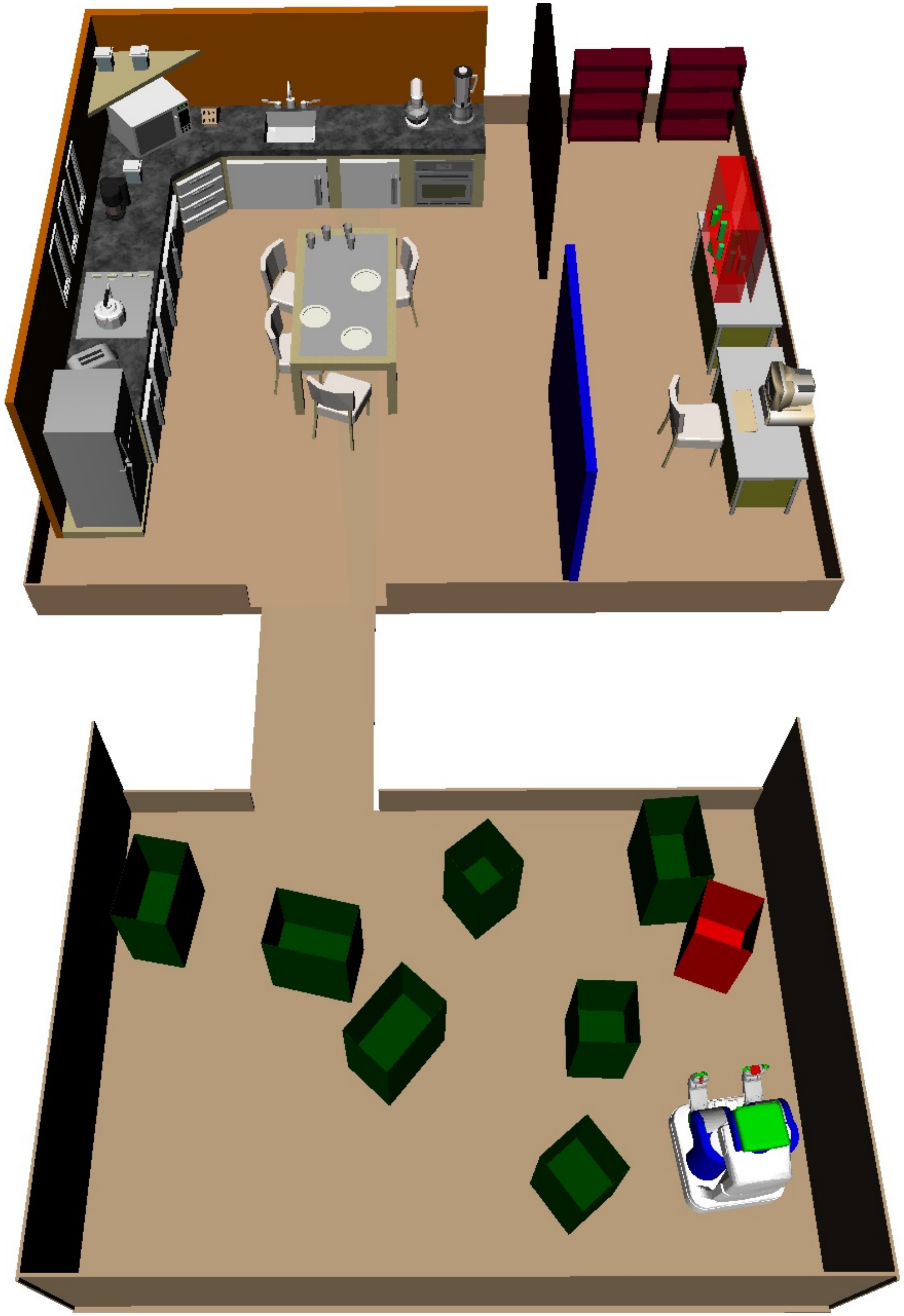
- inverse kinematics
- robot motion planning

Kim and Shimanuki, CORL 2019

Learning search control

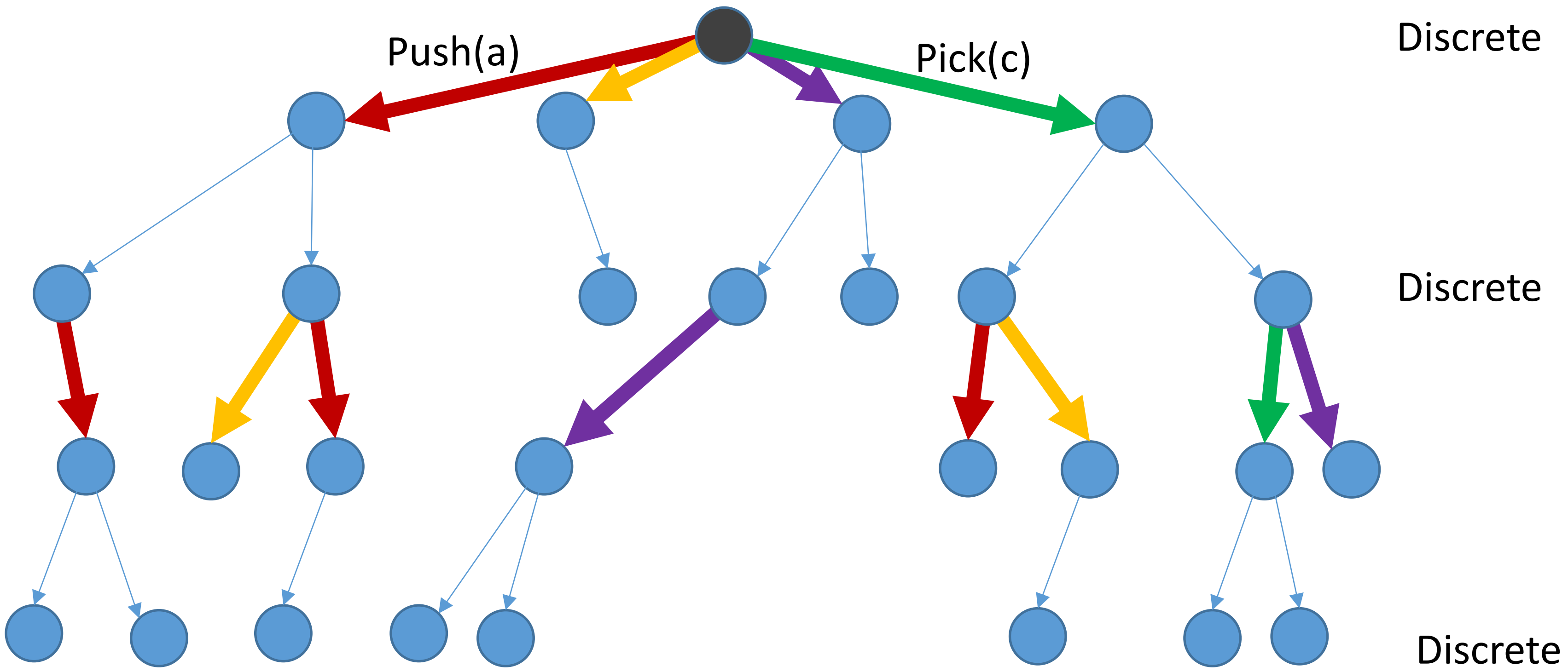
- Learning value function and policy worked great for AlphaZero!
- In our problem:
  - how to represent a state is much less clear
  - we need very **aggressive generalization**





Learn to predict value of abstract action choices

$$Q(s, a_{\text{discrete}}) = \sup_{a_{\text{continuous}}} Q(s, (a_{\text{discrete}}, a_{\text{continuous}}))$$

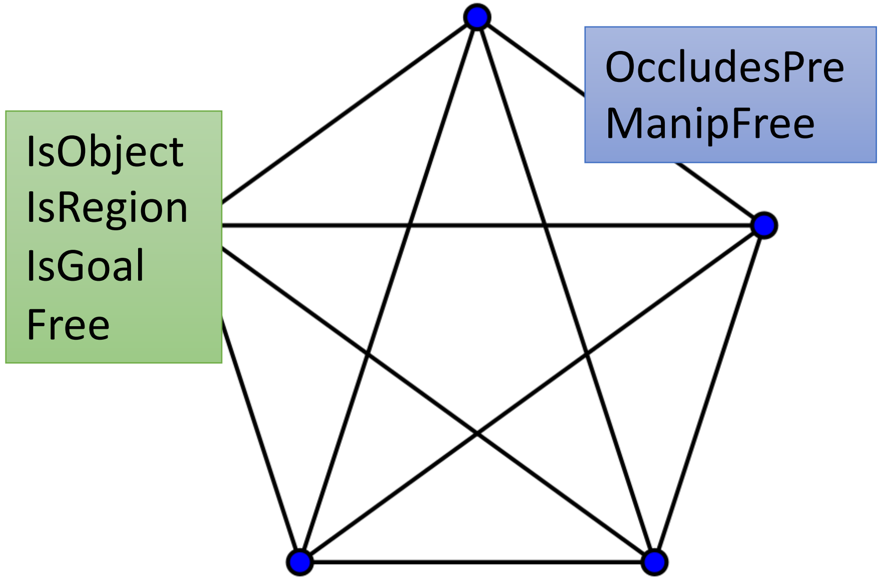


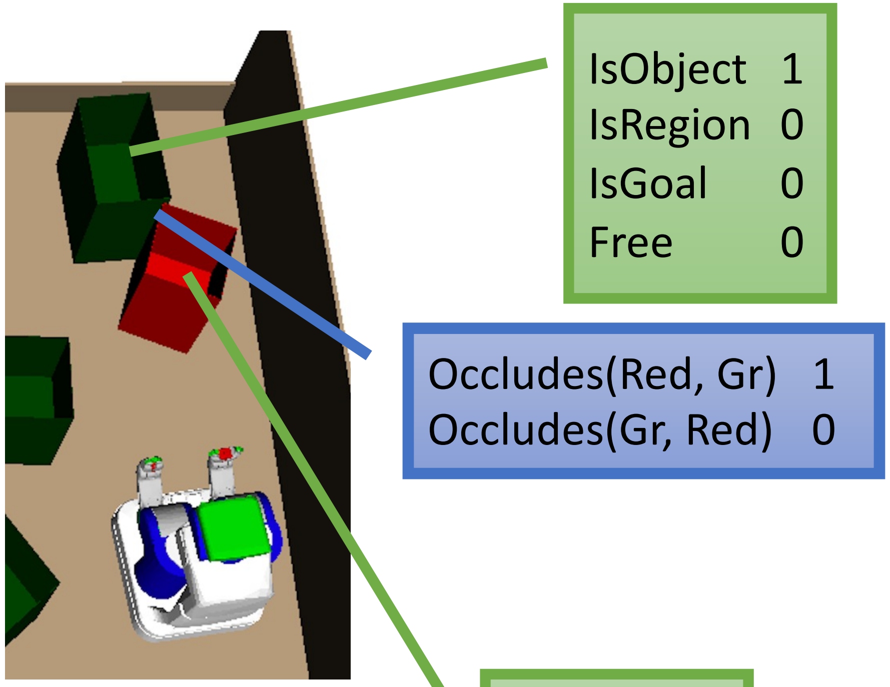
Reduce effective branching

Agenda contains (s,a) pairs

Abstract action values generalize well

Graph NN representation generalizes over scenes and goals





Graph neural network with fixed-size parameterization

- W1: map node and arc input to initial node state
- W2: map neighboring node states and arc inputs to message
- W3: map aggregate node states to predicted output

Use several rounds of message-passing to infer relational Q value

Training based on solving simple planning problems

Training data tuples: (s, g, op, q)

- squared loss for (s, g, op) in training data
- enforce margin Q(s, g, op') < Q(s, g, op) – 1 for op' not in training data

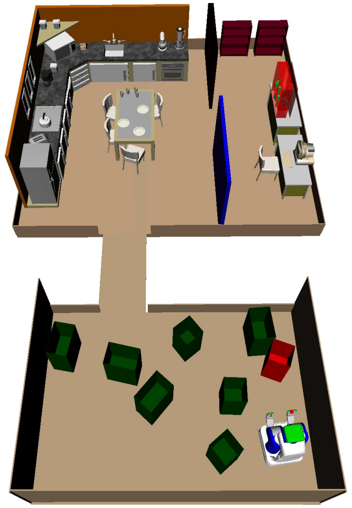
Square error on predicted Q value

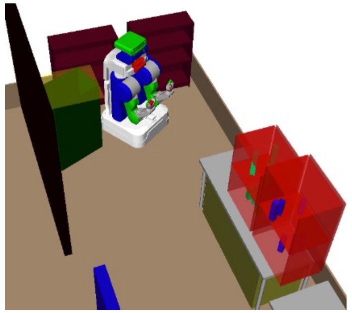
predicted Q value should be < Q value of action in training set

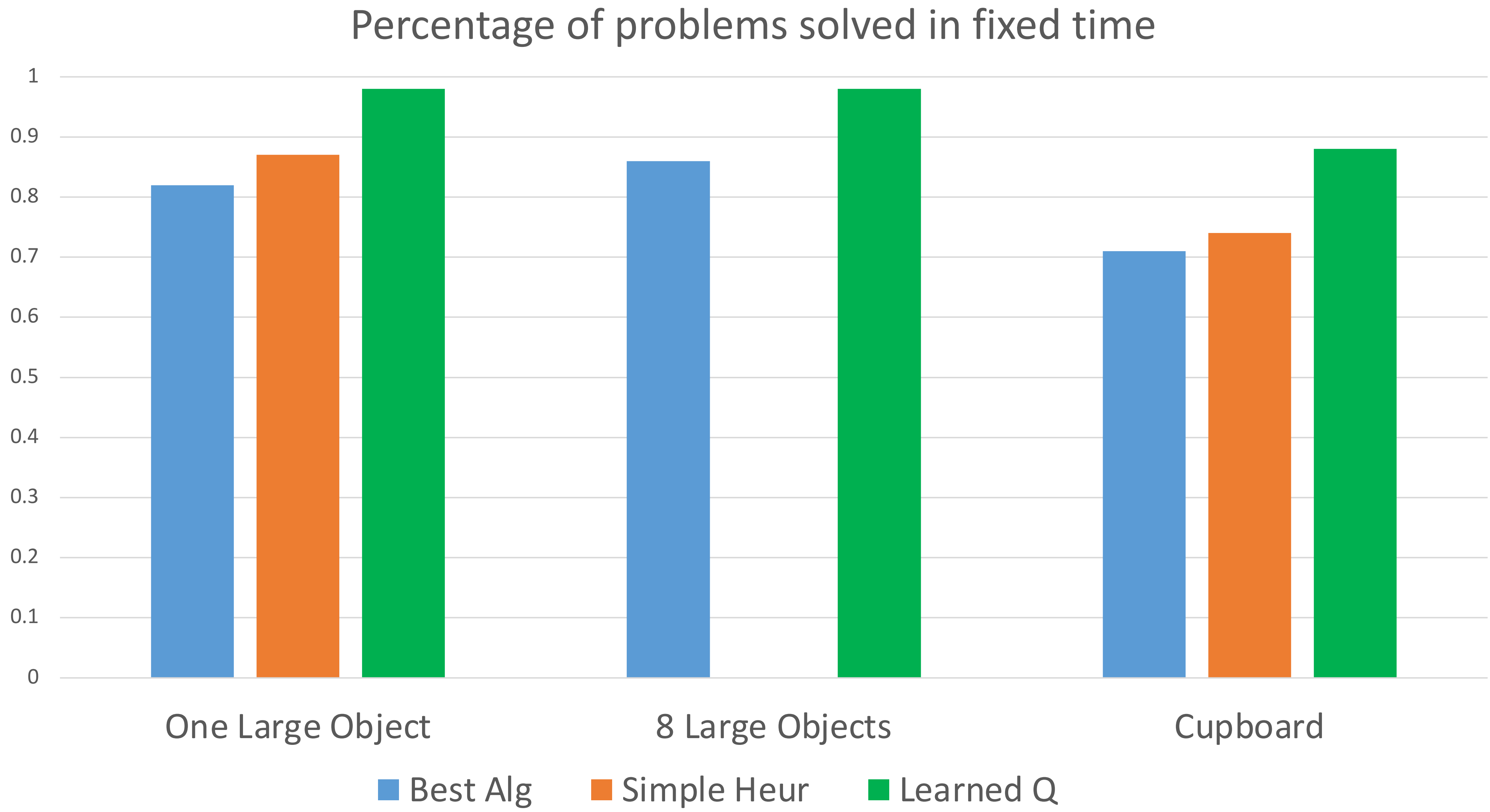
$$\mathcal{L}_{LM}(\theta) = \sum_{(s,\mathcal{G}\delta,q)\in\mathcal{D}_{\mathsf{o}}} (\widehat{Q}_{\mathsf{o}}(\alpha(s,\mathcal{G}),\delta;\theta) - q)^2 + \lambda \max(0, 1 - M_Q(\alpha(s,\mathcal{G}),\delta;\theta))$$

$$M_Q(\alpha(s,\mathcal{G}),\delta;\theta) = \widehat{Q}_{\mathsf{o}}(\alpha(s,\mathcal{G}),\delta;\theta) - \max_{\delta'\in\Delta\setminus\{\delta\}} \widehat{Q}_{\mathsf{o}}(\alpha(s,\mathcal{G}),\delta';\theta) \ .$$

Early results: trained only moving one large object







IJCAI 1997

IJCAI 2019

Conclusions

really enormously

- There has been major progress in algorithms for supervised and reinforcement learning
- This does not directly yield solutions for building generally intelligent autonomous agents
- Human insight is needed to complement the strengths of these algorithms

in the form of algorithmic and structural biases

Thanks to lots of people!

- Tomas Lozano-Perez
- Caelan Garrett
- Aidan Curtis
- Xiaolin Fang
- Beomjoon Kim
- Luke Shimanuki
- Zi Wang

Out-takes to watch during questions