

Learning to plan in hybrid spaces

FA9550-17-1-0165

PI: Leslie Pack Kaelbling (MIT)

Co-PI: Tomas Lozano-Perez (MIT)

AFOSR Program Review:

Computational Cognition and Machine Intelligence Program

October 7, 2020, Arlington, VA



Learning to plan (Kaelbling)

Research Objectives:

Improve efficiency and quality of results from planning by learning through experience

Technical Approach:

- Learn samplers for continuous spaces
- Learn abstract operator models
- Learn heuristics and hierarchical models

Key Scientific Contributions:

- Active sampling to reduce data requirements
- Hybrid planning systems combine abstract and geometric reasoning

DoD Benefits:

Autonomous systems that can plan and execute over long time horizons

Extension to belief-space enables flexible interaction with humans and reasoning under uncertainty

Project Goals

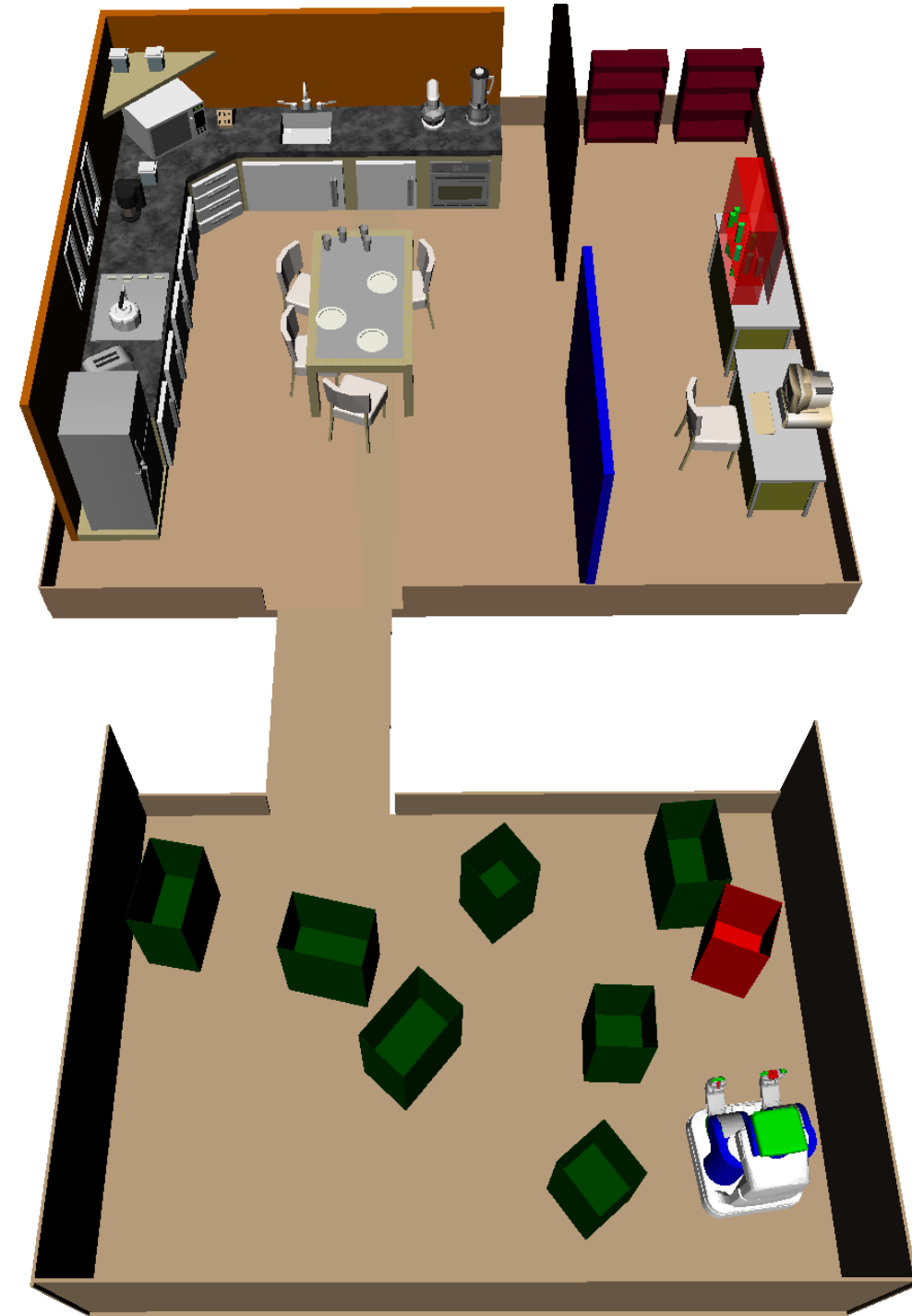
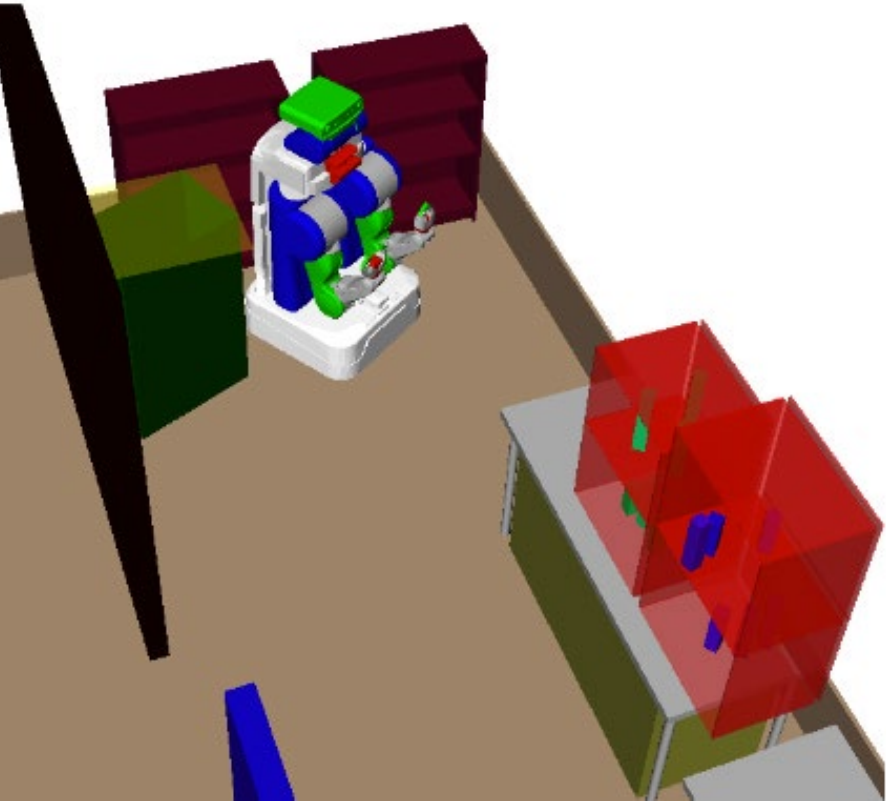
1. Learn heuristics: hybrid spaces, belief space
2. Predicting sub-goals and plan constraints
3. Learn to hedge
4. Learn to construct abstractions
5. Learn to control hierarchical planning
6. Demonstrate on real robot

This Year's Progress Towards Goals

1. Learn heuristics: hybrid spaces, belief space
 - Relational value functions as search heuristics
2. Predicting sub-goals and plan constraints
 - GLIB: Exploration via goal-literal babbling for lifted operator learning
3. Learn to hedge
4. Learn to construct abstractions
 - CAMPs: Learning Context-Specific Abstractions for Efficient Planning in Factored MDPs
5. Learn to control hierarchical planning
6. Demonstrate on real robot
7. STRIPStream planner: application to belief space
8. Learning within a single continuous-space planning problem
 - VOOT: Voronoi Optimistic Optimization in Trees

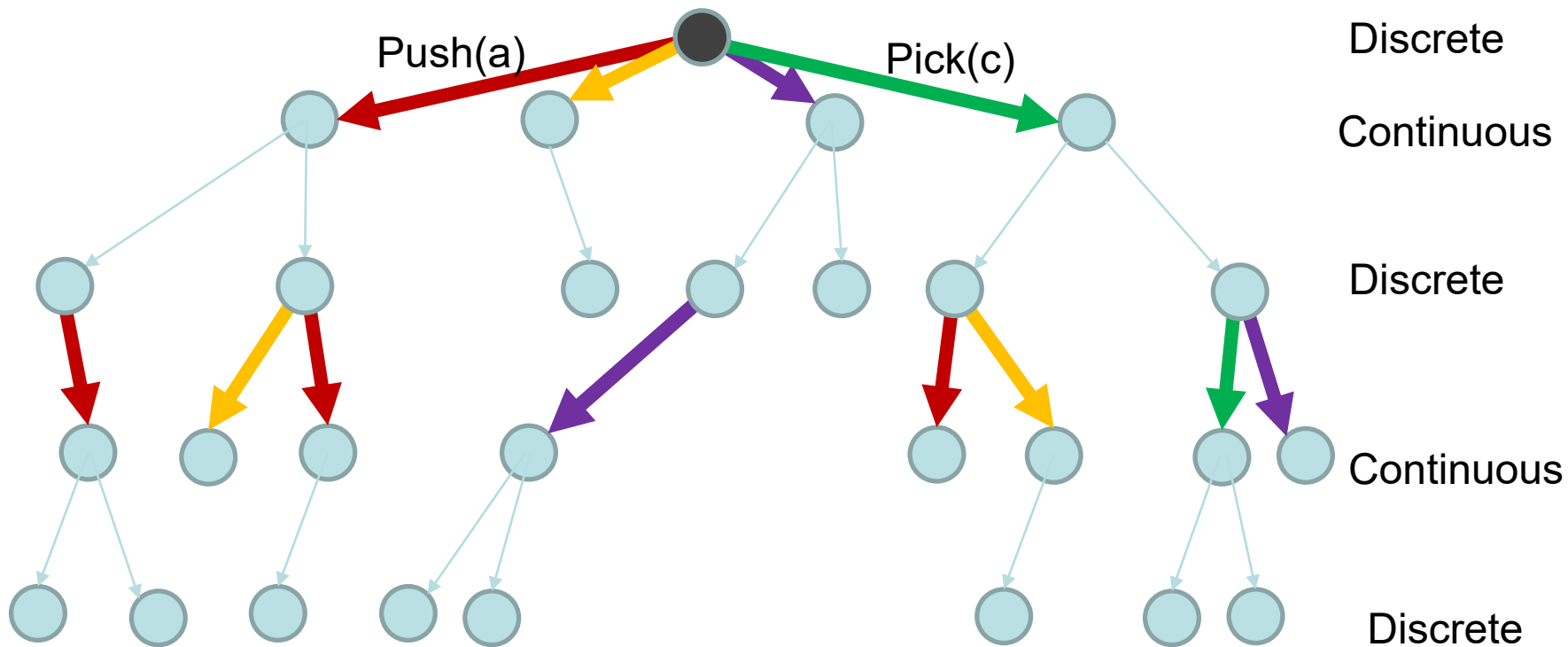
Learning search control

- 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

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

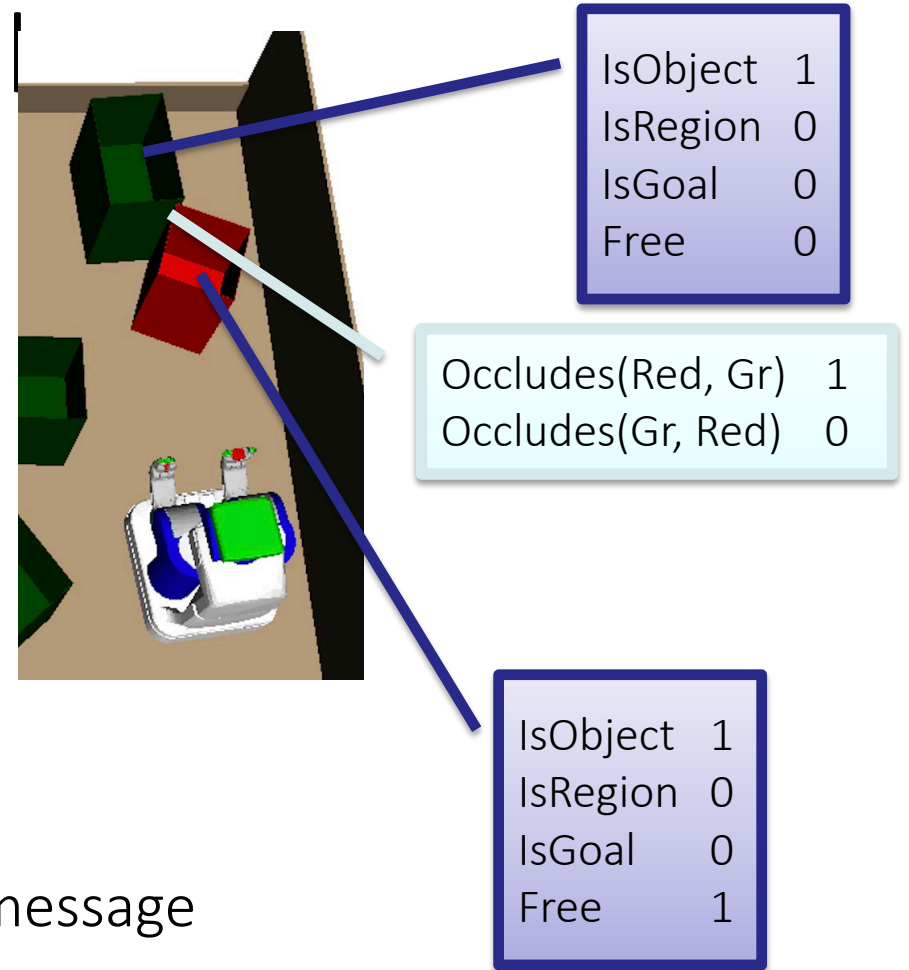
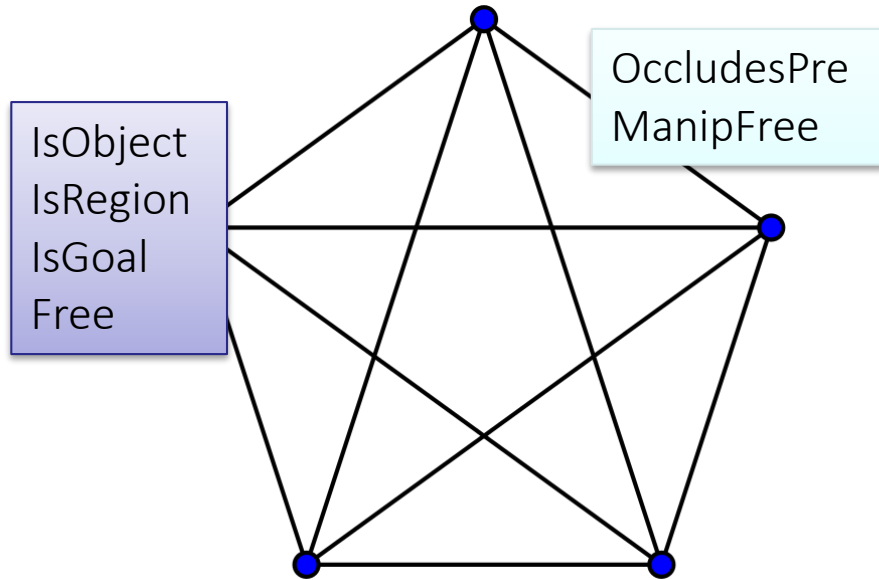


Reduce effective branching

Agenda contains (s,a) pairs

Abstract action values generalize well

Graph NN representation generalizes over scenes and goal



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

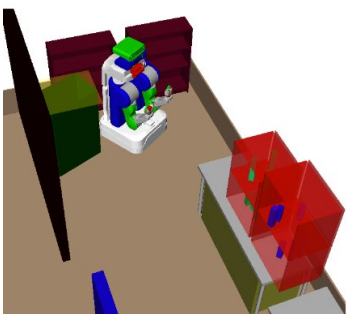
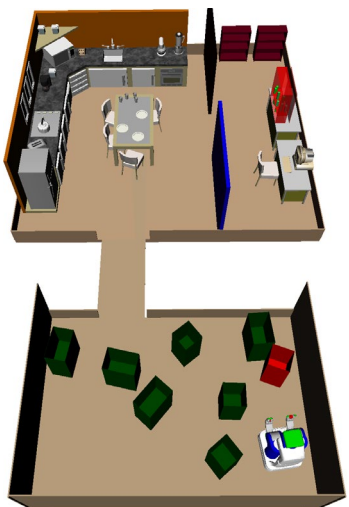
Square error on
predicted Q value

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

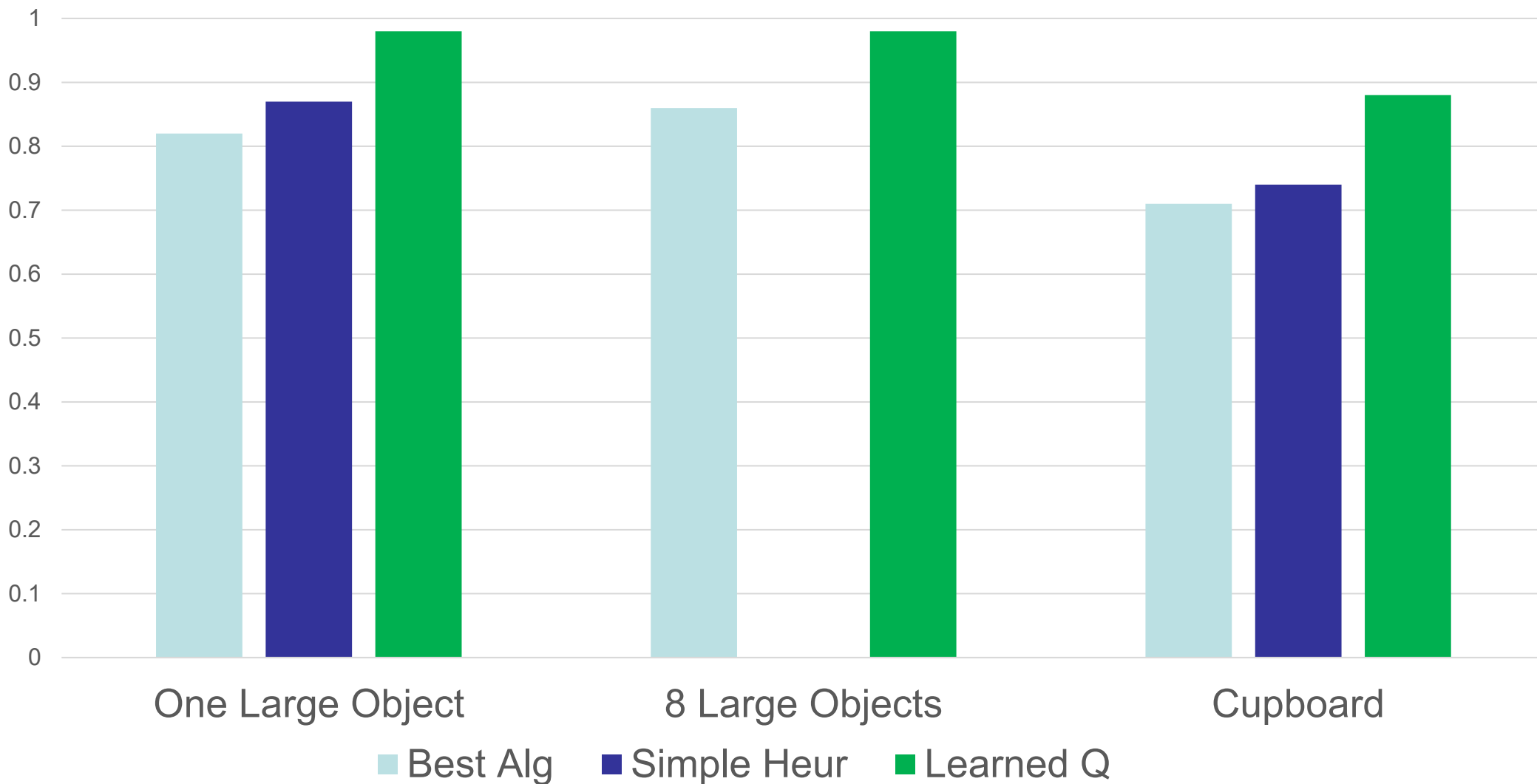
$$\mathcal{L}_{LM}(\theta) = \sum_{(s, \mathcal{G}, \delta, q) \in \mathcal{D}_o} (\hat{Q}_o(\alpha(s, \mathcal{G}), \delta; \theta) - q)^2 + \lambda \max(0, 1 - M_Q(\alpha(s, \mathcal{G}), \delta; \theta))$$

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

Results: trained only moving one large object

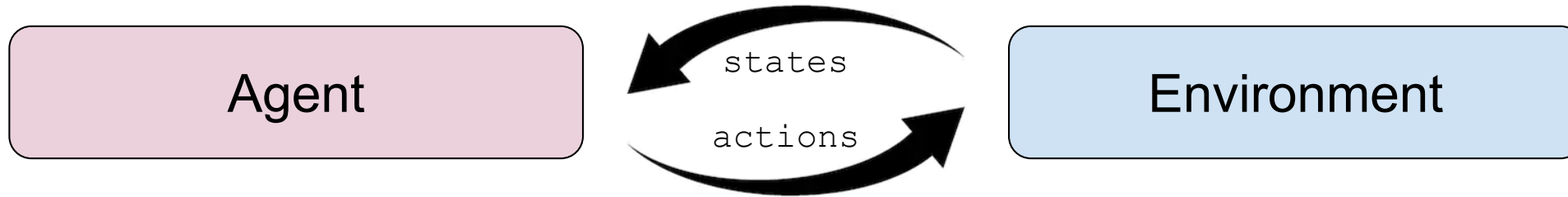


Percentage of problems solved in fixed time



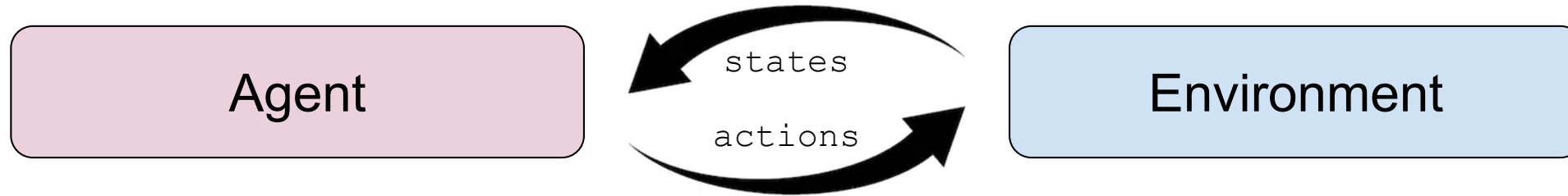
Consider an agent in a goal-free setting

- The agent takes actions, the environment gives states

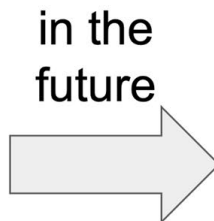


Consider an agent in a goal-free setting

- The agent takes actions, the environment gives states



- It will be given a set of unknown test goals later
 - But it has no extrinsic goals or rewards during “playtime”



Abstract away perception and control

- The agent sees predicates, not pixels
 - `{holding(A), onTable(C), ...}`
- Test-time goals will be predicate-based too
 - `holding(C) & not onTable(D)`



Running example: “Blocks”

Abstract away perception and control

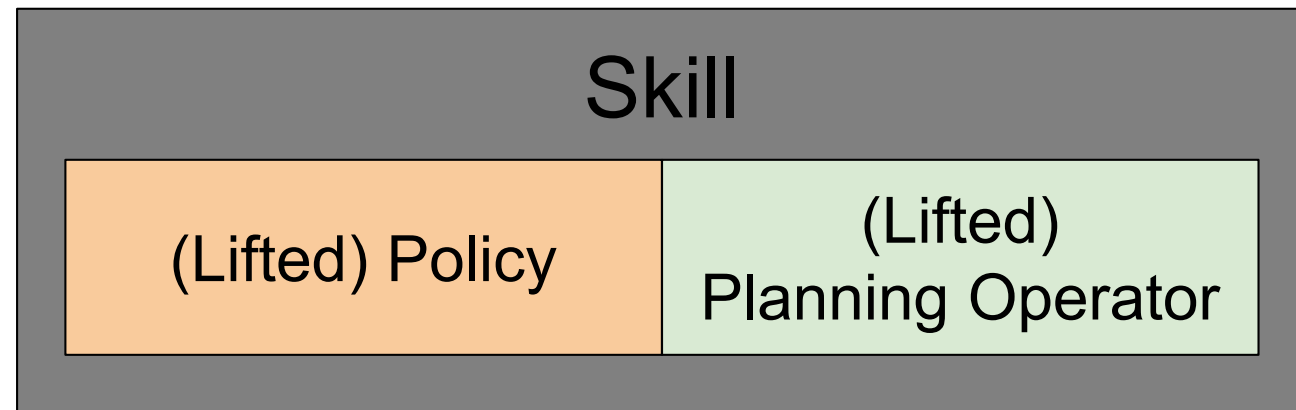
- The agent sees predicates, not pixels
 - `{holding(A), onTable(C), ...}`
- Test-time goals will be predicate-based too
 - `holding(C) & not onTable(D)`
- Actions are high-level primitives
 - `pick(A), stack(B, C), etc.`
- Everything is discrete!



Running example: “Blocks”

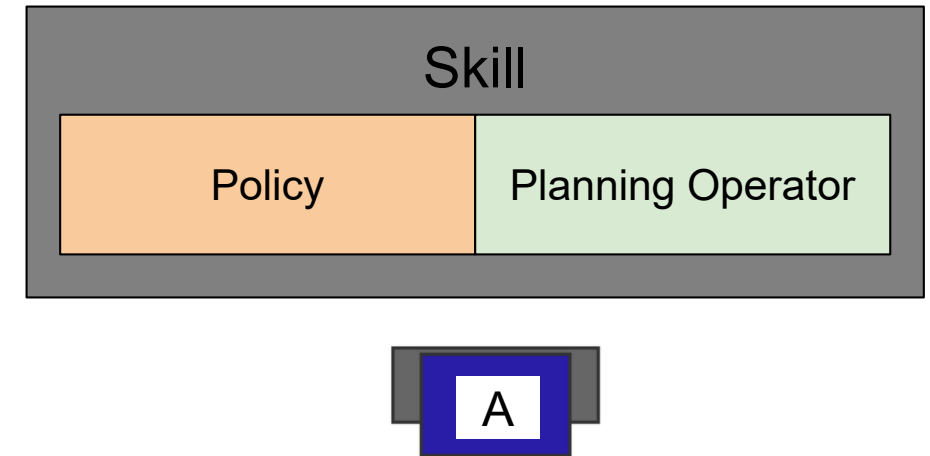
What should the agent learn during “playtime”?

- Skills! Skill == Policy + Planning Operator
- A planning operator describes the *preconditions* and *effects* of executing a particular *policy*
- Policies and operators are both *lifted* and share *parameters*
- Planning operators allow STRIPS-style symbolic planning
- Policies allow executing a plan



Skill Examples

- Primitive policies
 - `pick(X)`, `stack(X, Y)`, etc.
 - `X`, `Y` are just parameters!
- Learned policies
 - `learnedPolicy(X, Y, Z)`
 - Example: “make a stack of 3”
 - Learned policies can invoke primitive policies or other learned policies



Running example: “Blocks”

Skill Examples

Example planning operator for `pick(X)`:

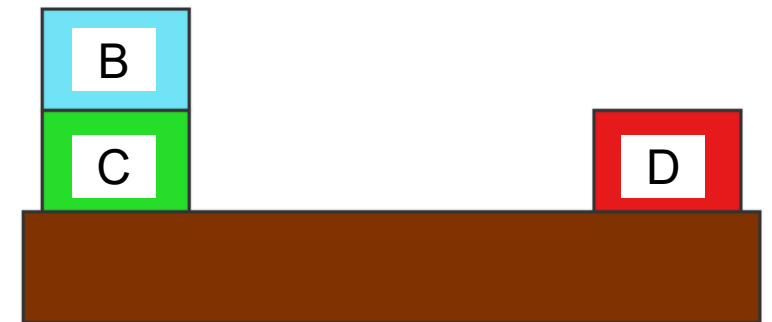
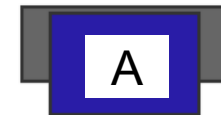
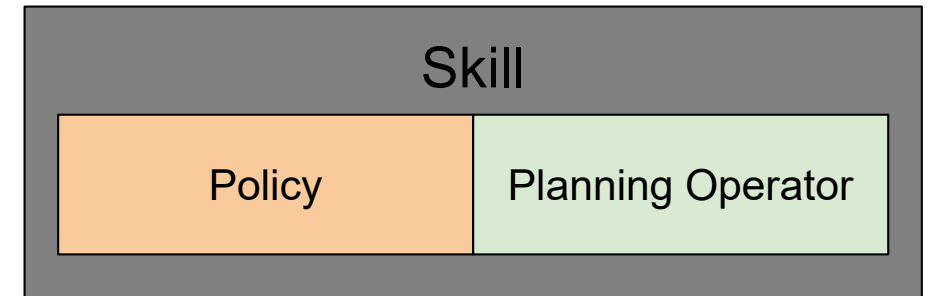
Parameters: `X`

Preconditions:

`nothingAbove(X) & handEmpty()`

Effects:

`holding(X) & (not onTable(X)) &
(not nothingAbove(X)) &
(not handEmpty())`

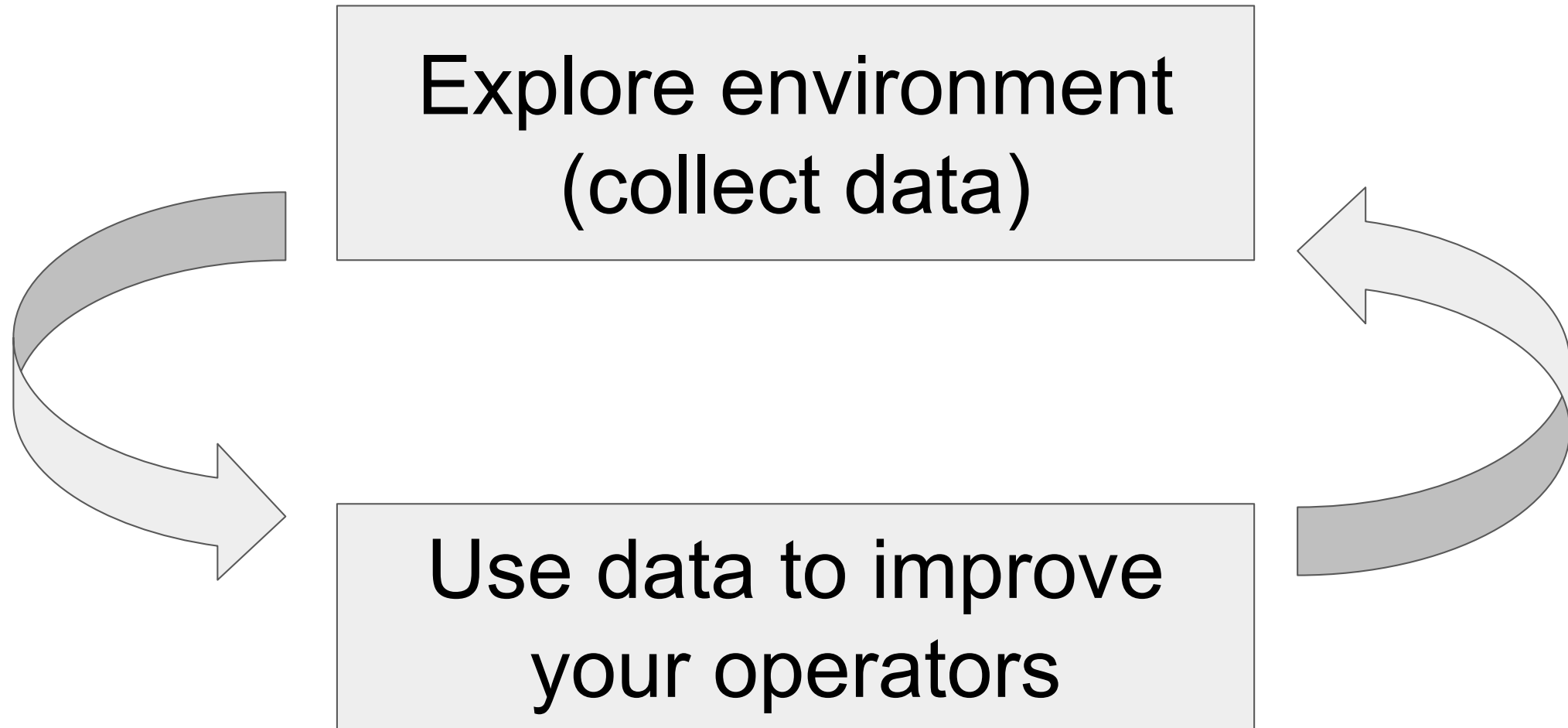


Running example: "Blocks"

The exploration problem

- **Given:** a predicate-based domain with unknown transition model, & a transition model learner
- **We need:** an exploration policy in the domain to gather the best possible data for learning the transition model quickly
 - Map from current state to an action

The self-contained exploration problem



“I wonder if I can...”

- CogSci research has found that humans often **set up problems** for themselves to try to solve during play



Wonder-if-I-can curiosity: unlike typical exploration in AI/ML

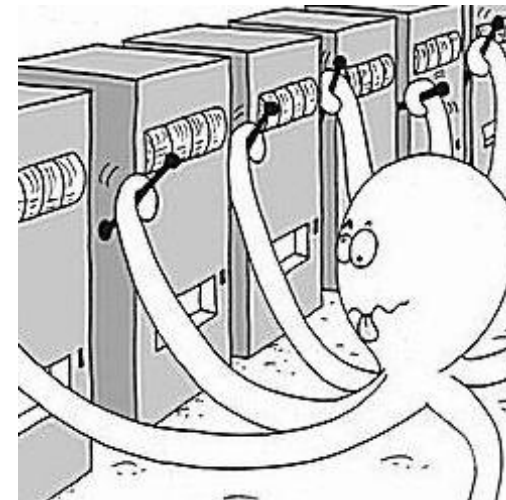
Wonder-if-I-can

- ✓ Repeatedly **setting** (arbitrary!) problems for yourself
- ✓ **Planning** to achieve a **goal**
- ✓ Pursuing **specific, targeted effects**



Traditional AI/ML Exploration

- No explicit problem-setting
- No planning, no lookahead
- (Sometimes) No factored state



Exploration via goal-literal babbling (GLIB)

Examples of goal literals:

- `holding(red-cup)`
- `on-peg(blue-ring)`
- `(not inside(kitchen)) & eaten(cookie)`

Babbling: fancy word for “randomly choosing”

Exploration via goal-literal babbling (GLIB)

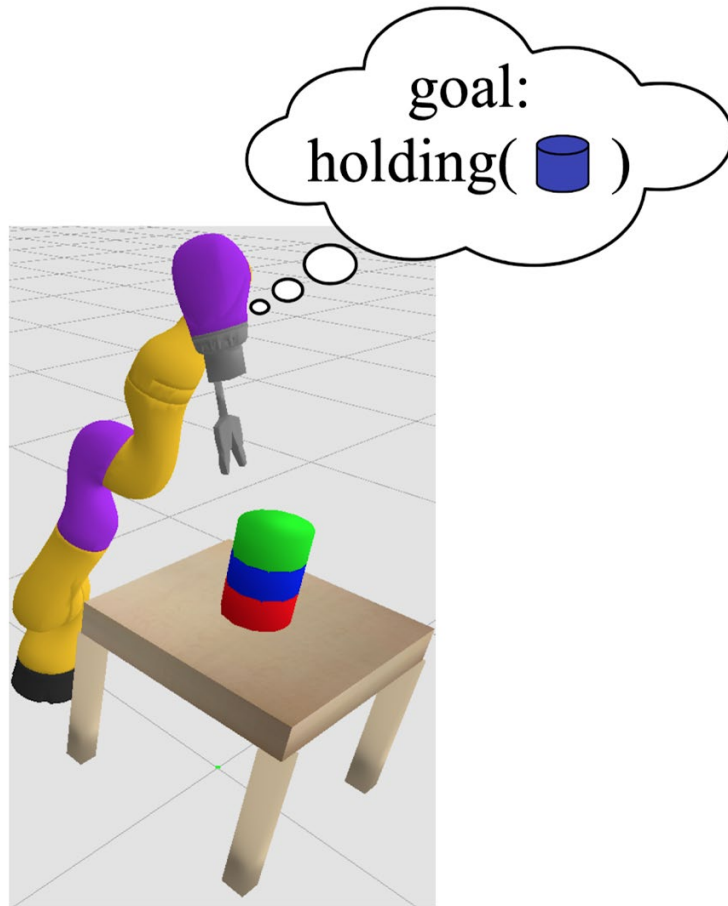
Examples of goal literals:

- `holding(red-cup)`
- `on-peg(blue-ring)`
- `(not inside(kitchen)) & eaten(cookie)`

Babbling: fancy word for “randomly choosing”

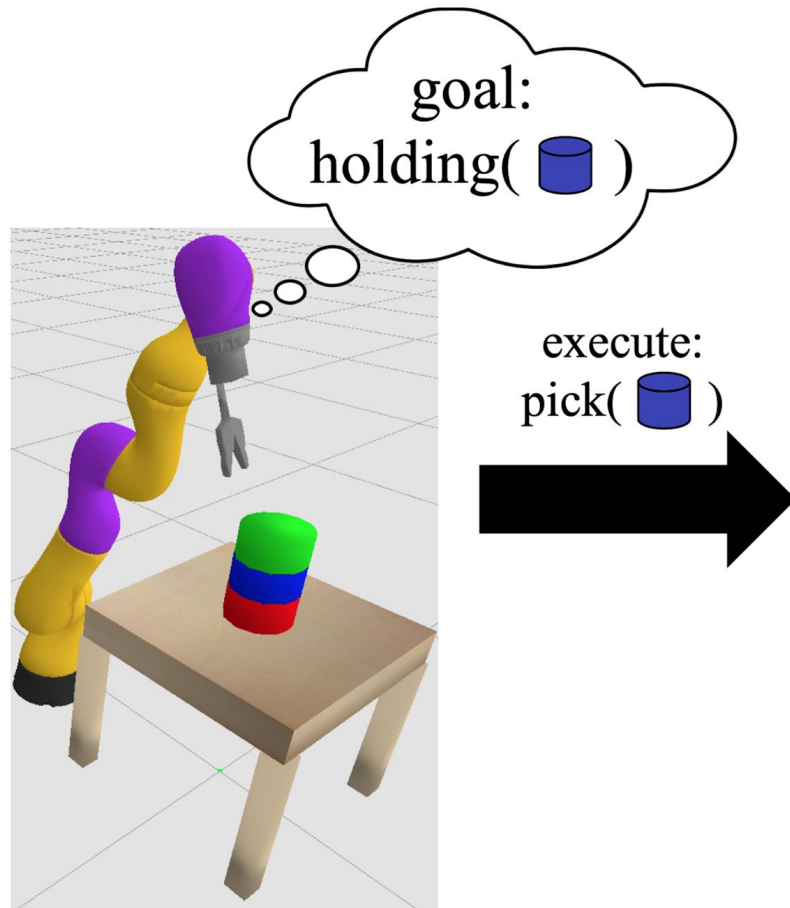
But how do we turn these *goals* into (exploratory) *actions*?

Idea: plan for the goals using the operators we're learning!



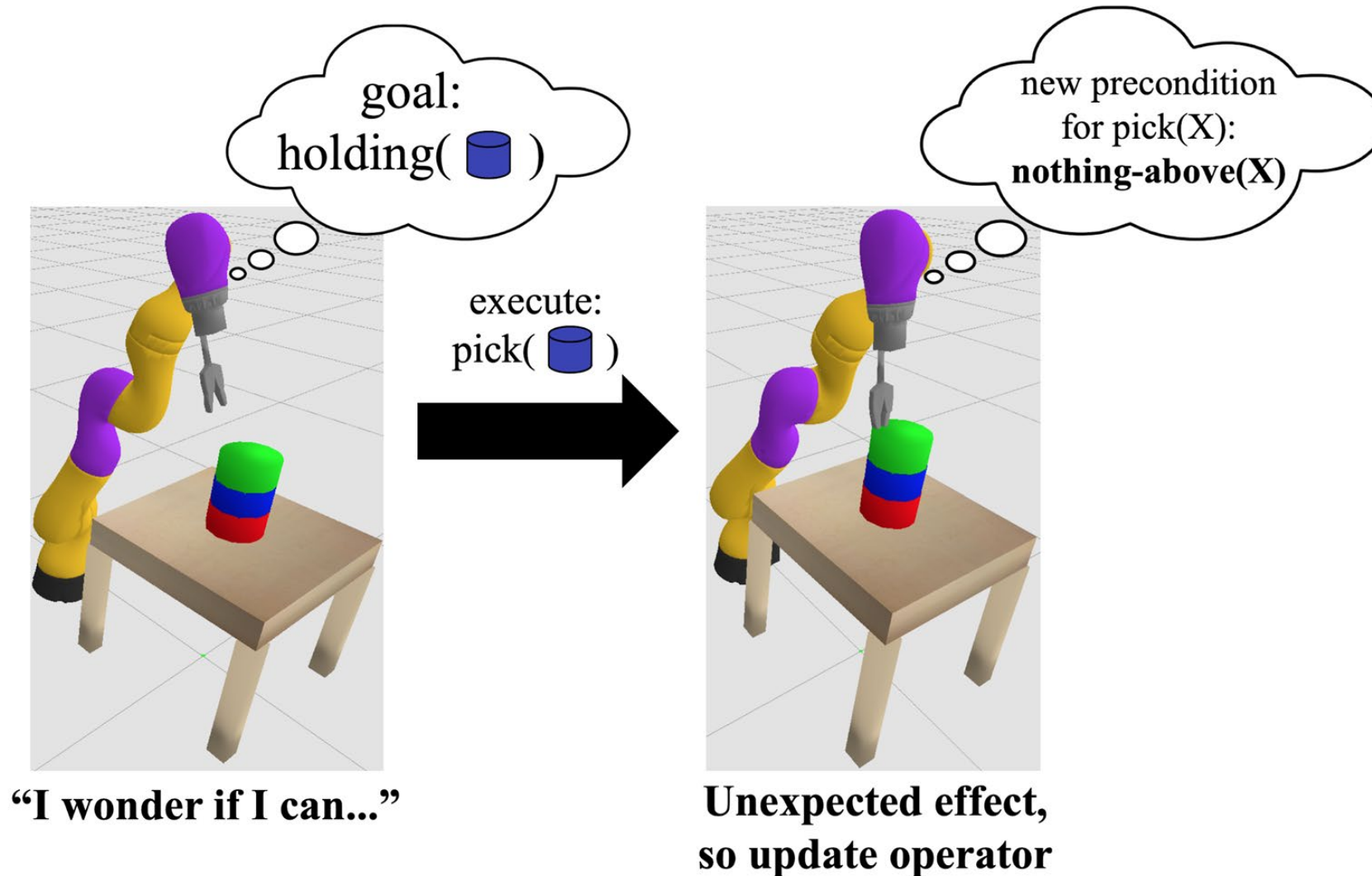
“I wonder if I can...”

Idea: plan for the goals using the operators we're learning!

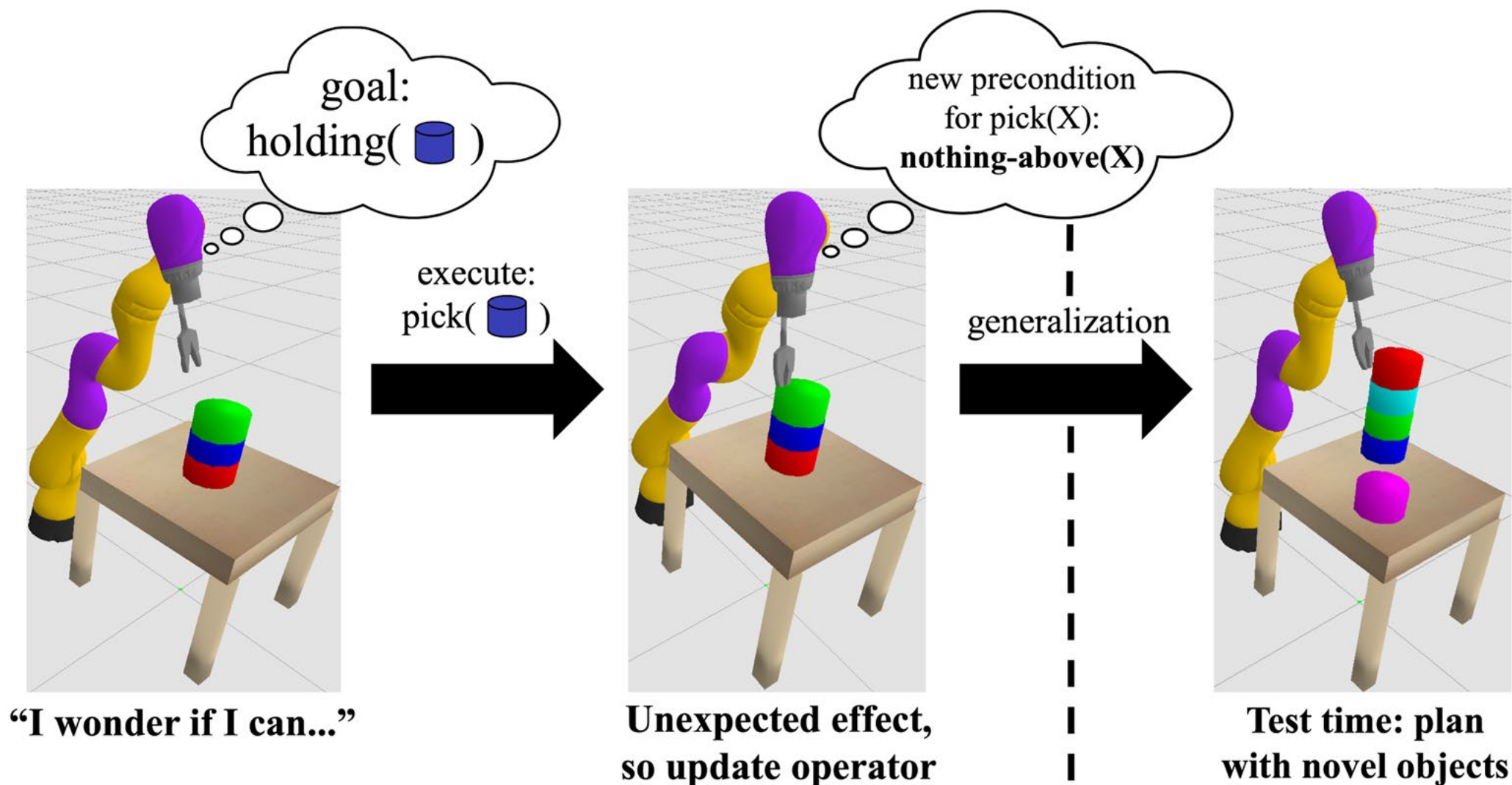


“I wonder if I can...”

Idea: plan for the goals using the operators we're learning!



Idea: plan for the goals using the operators we're learning!



Idea: plan for the goals using the operators we're learning!

So...is this actually a good idea?

- If models are overly pessimistic: "I don't know how to pick up the bottle"
- If models are overly optimistic: "I can go climb Everest right now!"

Idea: plan for the goals using the operators we're learning!

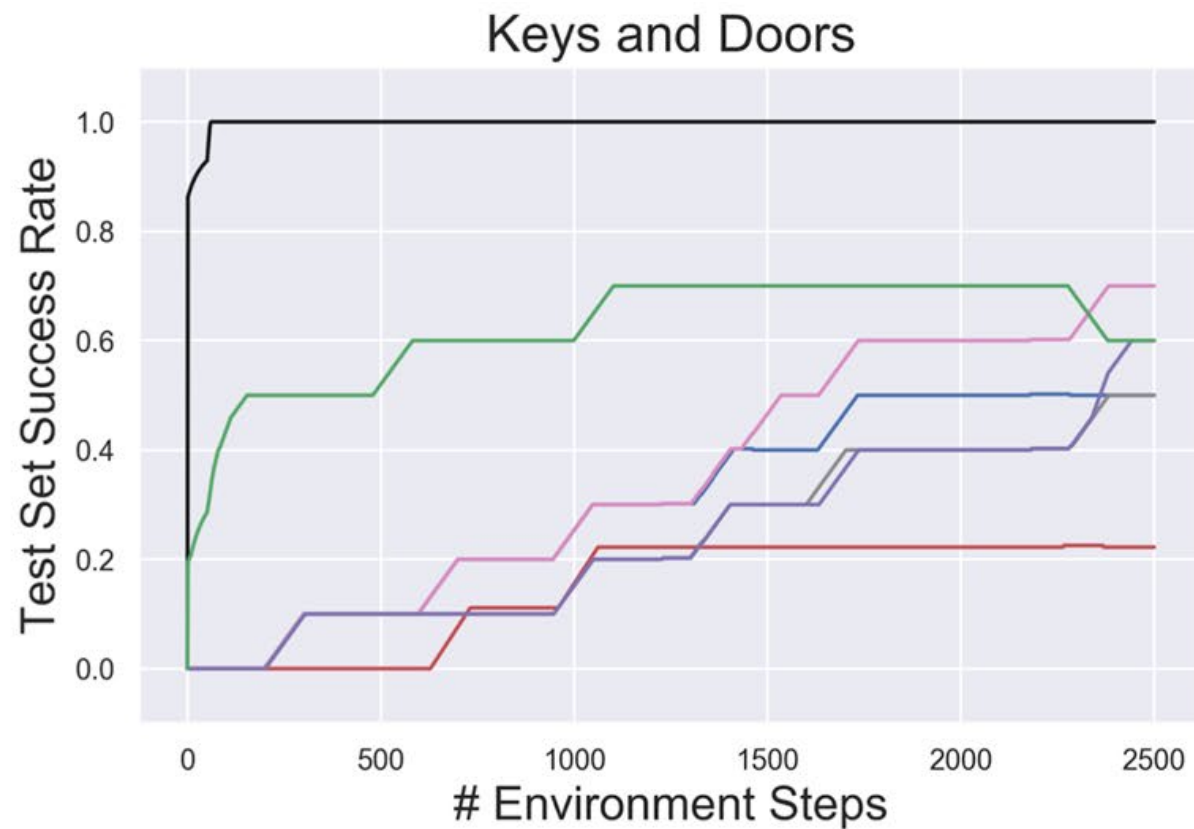
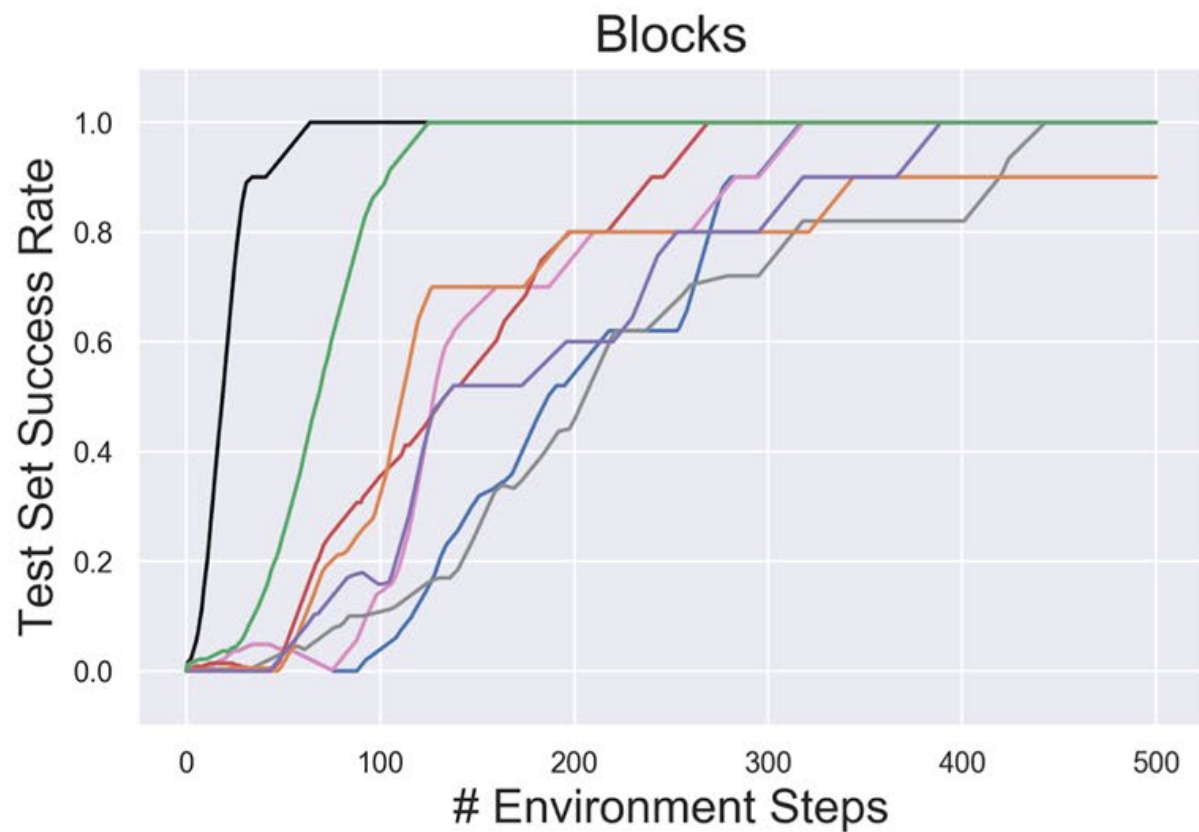
So...is this actually a good idea?

- If models are overly pessimistic: "I don't know how to pick up the bottle"
- If models are overly optimistic: "I can go climb Everest right now!"

Optimism is not a problem: agent will try to achieve some goal, and perhaps fail & learn something new

Pessimism is a big problem: agent will sit there, thinking it can't do anything!

GLIB Results



CAMPs: Learning Context-Specific Abstractions for Efficient Planning in Factored MDPs

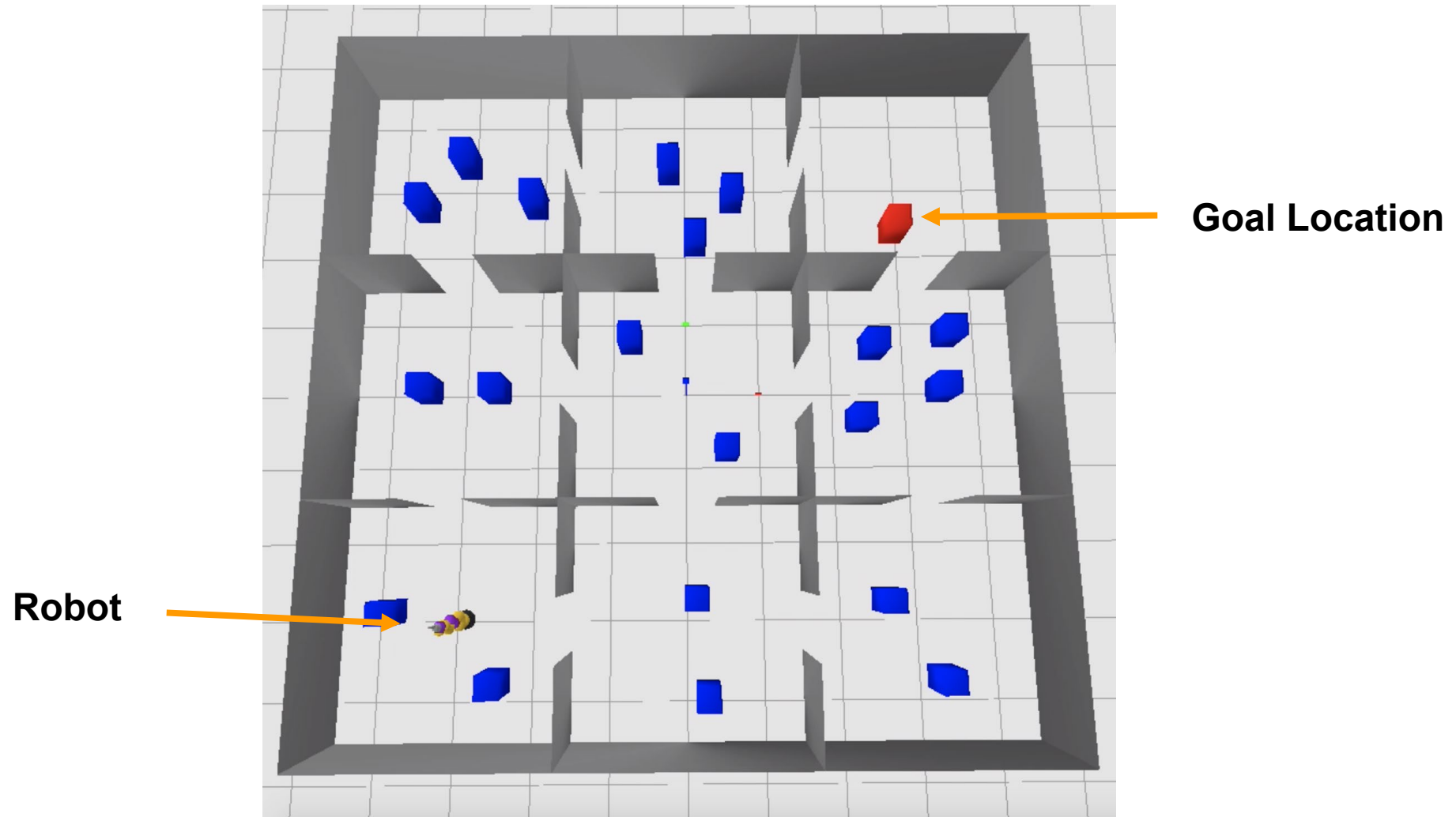
Rohan Chitnis & Tom Silver

With Beomjoon Kim, Tomas Lozano-Perez and Leslie Kaelbling



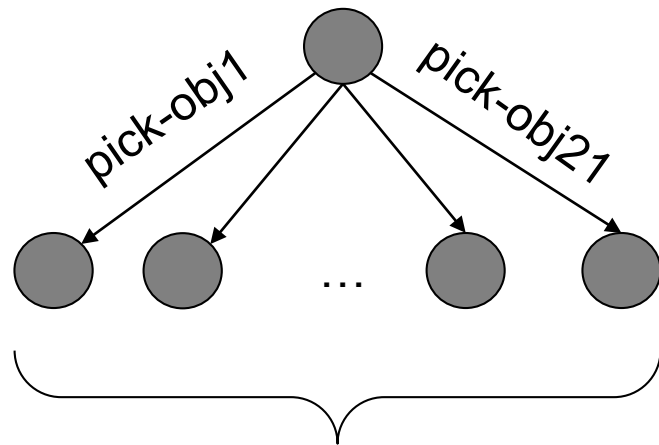
COMPLICATED CHOICES RESULT IN POSTPONED DECISIONS

Real Example #1: NAMO

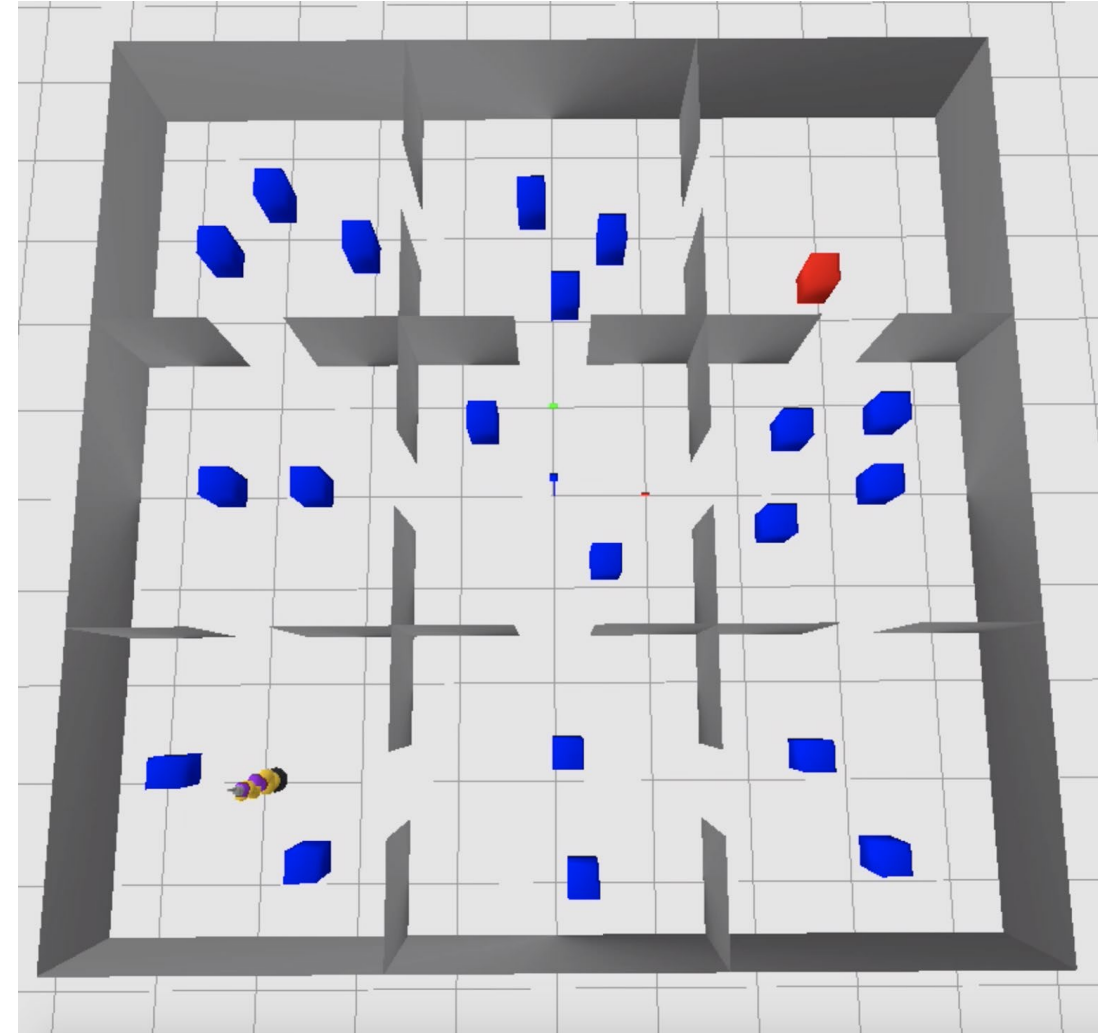


Real Example #1: NAMO

Planning with a General-Purpose
TAMP Planner

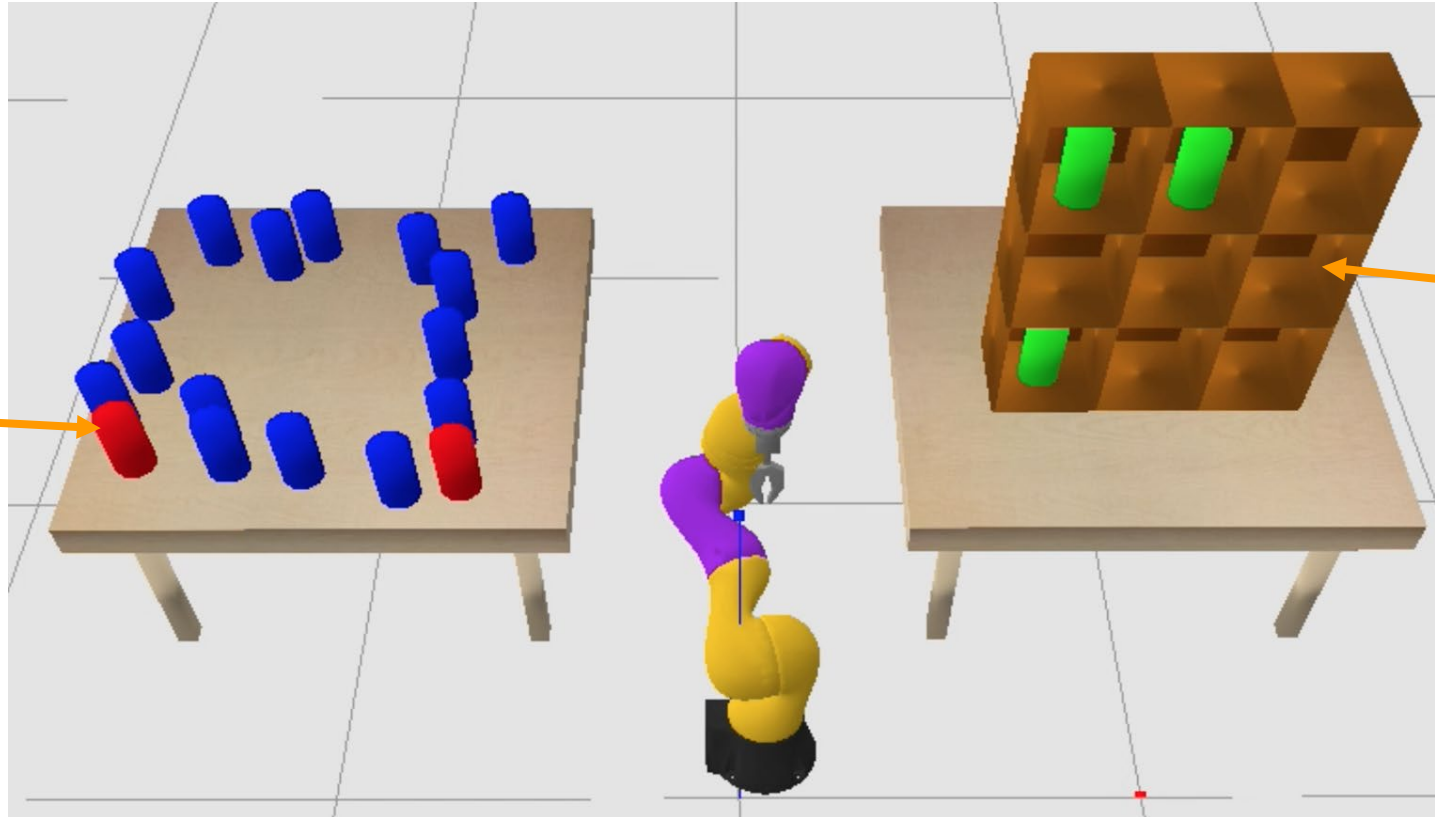


Painful branching factor



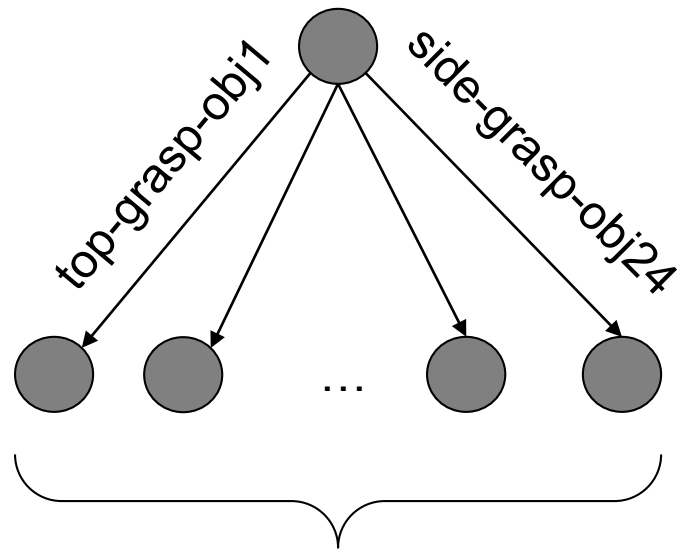
Real Example #2: Sequential Manipulation

Goal: put red
objects on
shelves

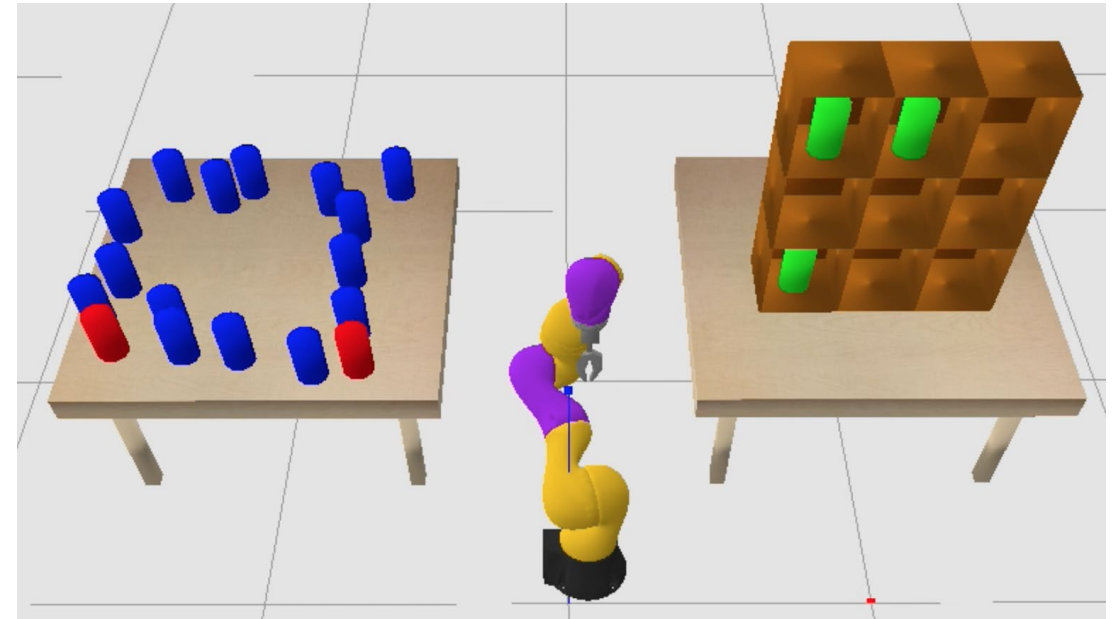


Real Example #2: Sequential Manipulation

Planning with a General-Purpose
TAMP Planner



Painful branching factor



Strategy for Faster Planning: **Constraints**

Constraints can be imposed on states and actions.

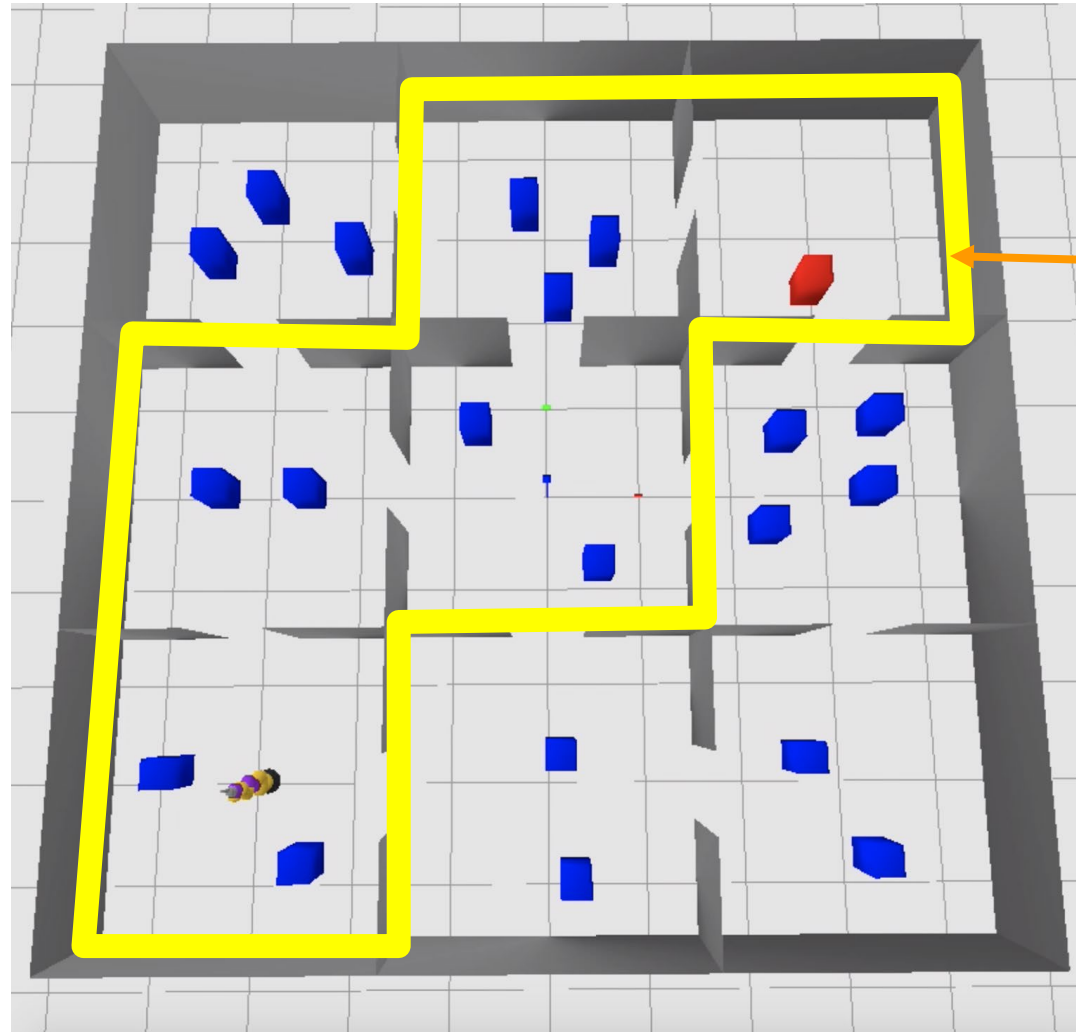
The agent can impose such constraints “**angelically**” (on itself, for the purpose of producing a more efficient problem).



“I’m only going to take allowed actions that keep me in the allowed states.”

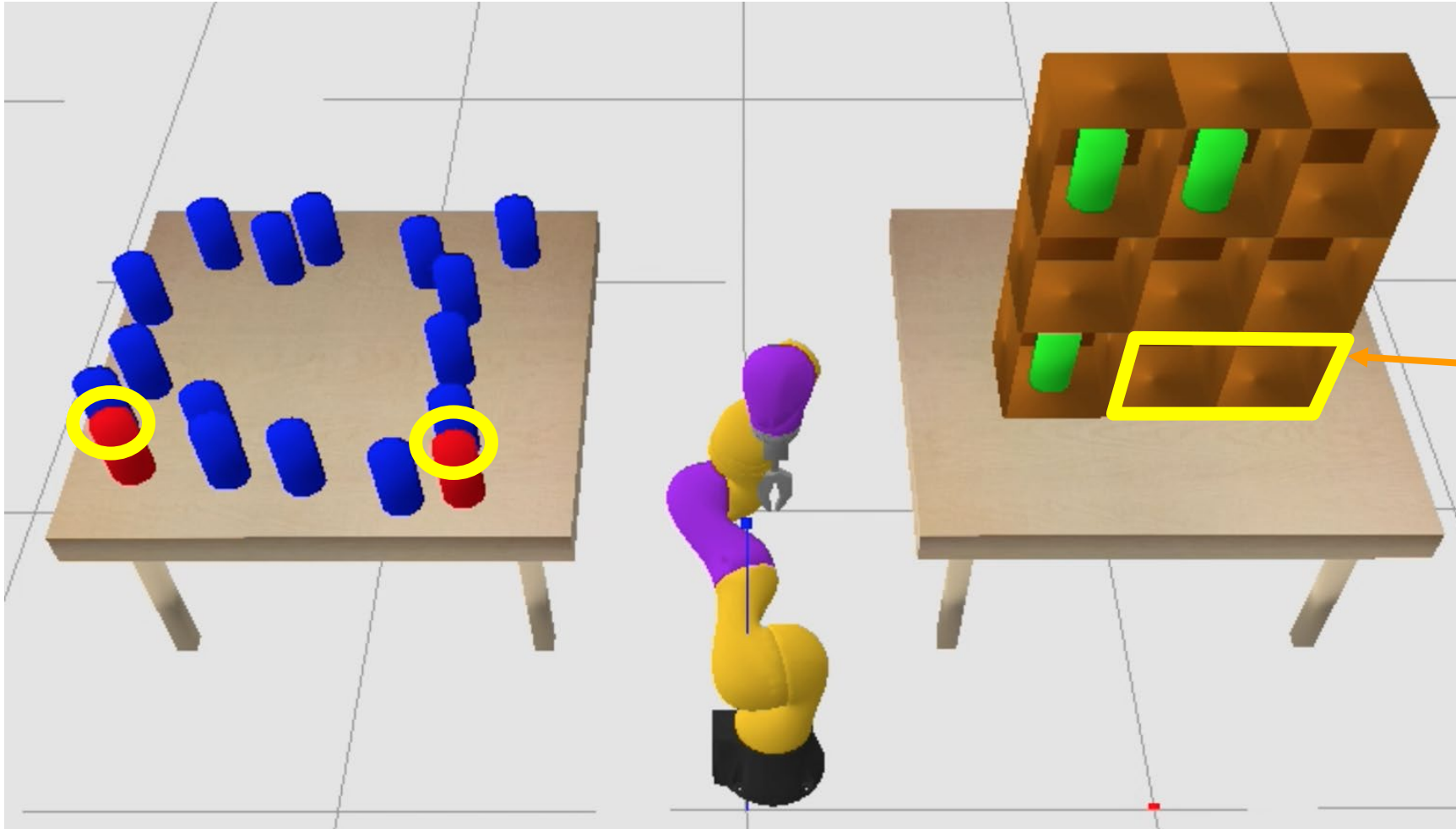
This can result in an easier planning problem.

Strategy for Faster Planning: Constraints



A constraint

Strategy for Faster Planning: Constraints



A constraint

Main Insights of this Work

1. Imposing constraints can make a planning problem easier to solve.
2. But also, we can get *more* out of these constraints.

Main Insights of this Work

1. Imposing constraints can make a planning problem easier to solve.
2. **But also, we can get *more* out of these constraints.**
 - Constraints induce *context-specific independences* between variables in a factored planning problem.

Main Insights of this Work

1. Imposing constraints can make a planning problem easier to solve.
2. **But also, we can get *more* out of these constraints.**
 - Constraints induce *context-specific independences* between variables in a factored planning problem.

Definition (Boutlier 96): Given random variables X, Y, C , and a set of values \mathbf{C} for C , if

$$P(X | Y, C = c) = P(X | C = c) \quad \forall c \in \mathbf{C}$$

then X and Y are *independent in the context* (C, \mathbf{C}) .

Main Insights of this Work

1. Imposing constraints can make a planning problem easier to solve.
2. **But also, we can get**
 - Constraints induce conditional independencies between variables in a factored planning problem.

Example: My outfit (X) is independent of the weather (Y) only when I stay quarantined at home (C=True).

Definition (Boutlier 96): Given random variables X, Y, C, and a set of values \mathbf{C} for C, if

$$P(X \mid Y, C = c) = P(X \mid C = c) \quad \forall c \in \mathbf{C}$$

then X and Y are *independent in the context* (C, \mathbf{C}).

Main Insights of this Work

1. Imposing constraints can make a planning problem easier to solve.
2. **But also, we can get *more* out of these constraints.**
 - Constraints induce *context-specific independences* between variables in a factored planning problem.
 - These independences render some variables *irrelevant* to the problem.

Main Insights of this Work

1. Imposing constraints can make a planning problem easier to solve.
2. **But also, we can get *more* out of these constraints.**
 - Constraints induce *context-specific independences* between variables in a factored planning problem.
 - These independences render some variables *irrelevant* to the problem.
 - Dropping irrelevant variables constitutes an *abstraction* of the original problem.

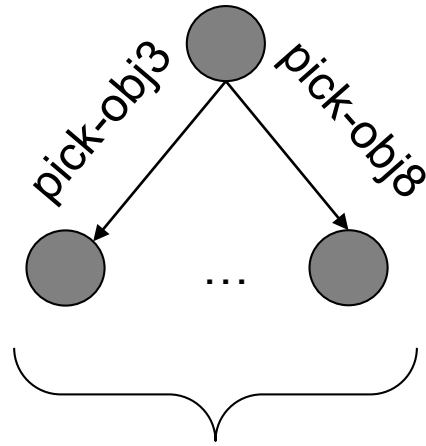
Main Insights of this Work

1. Imposing constraints can make a planning problem easier to solve.
2. **But also, we can get *more* out of these constraints.**
 - Constraints induce *context-specific independences* between variables in a factored planning problem.
 - These independences render some variables *irrelevant* to the problem.
 - Dropping irrelevant variables constitutes an *abstraction* of the original problem.

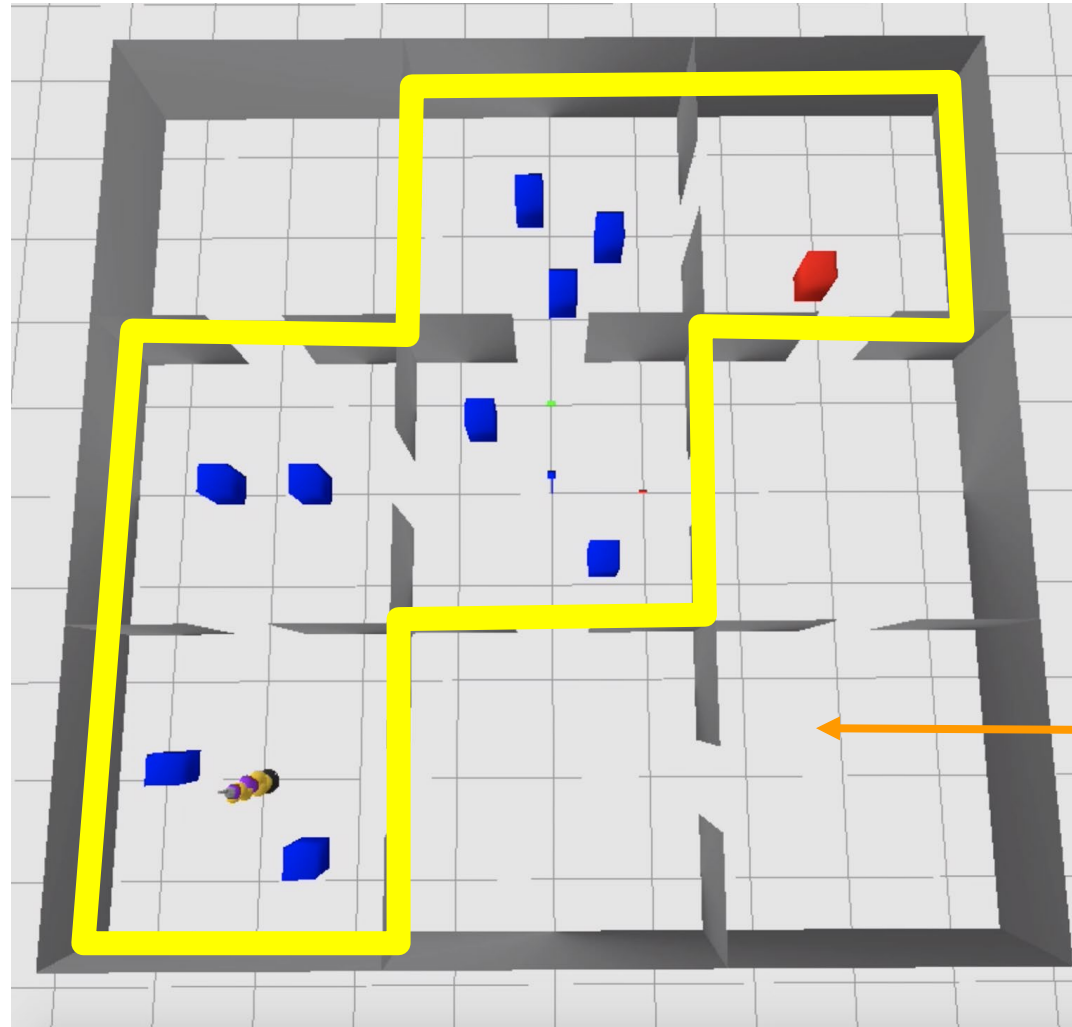
Angelic constraints → automatically derived abstractions!

Constraints \Rightarrow Abstraction

Planning with a General-Purpose
TAMP Planner



Much smaller branching factor

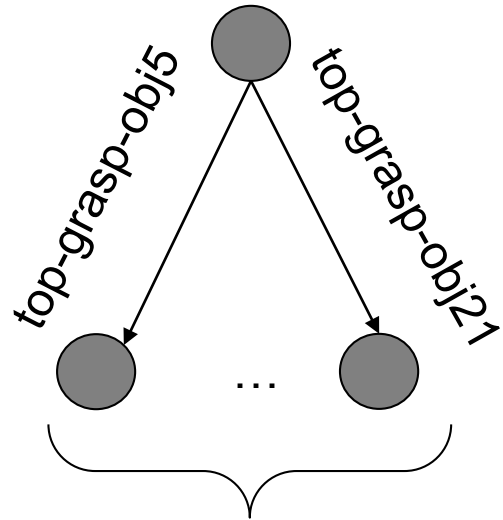


**Abstraction
ignores
objects in the
other rooms**

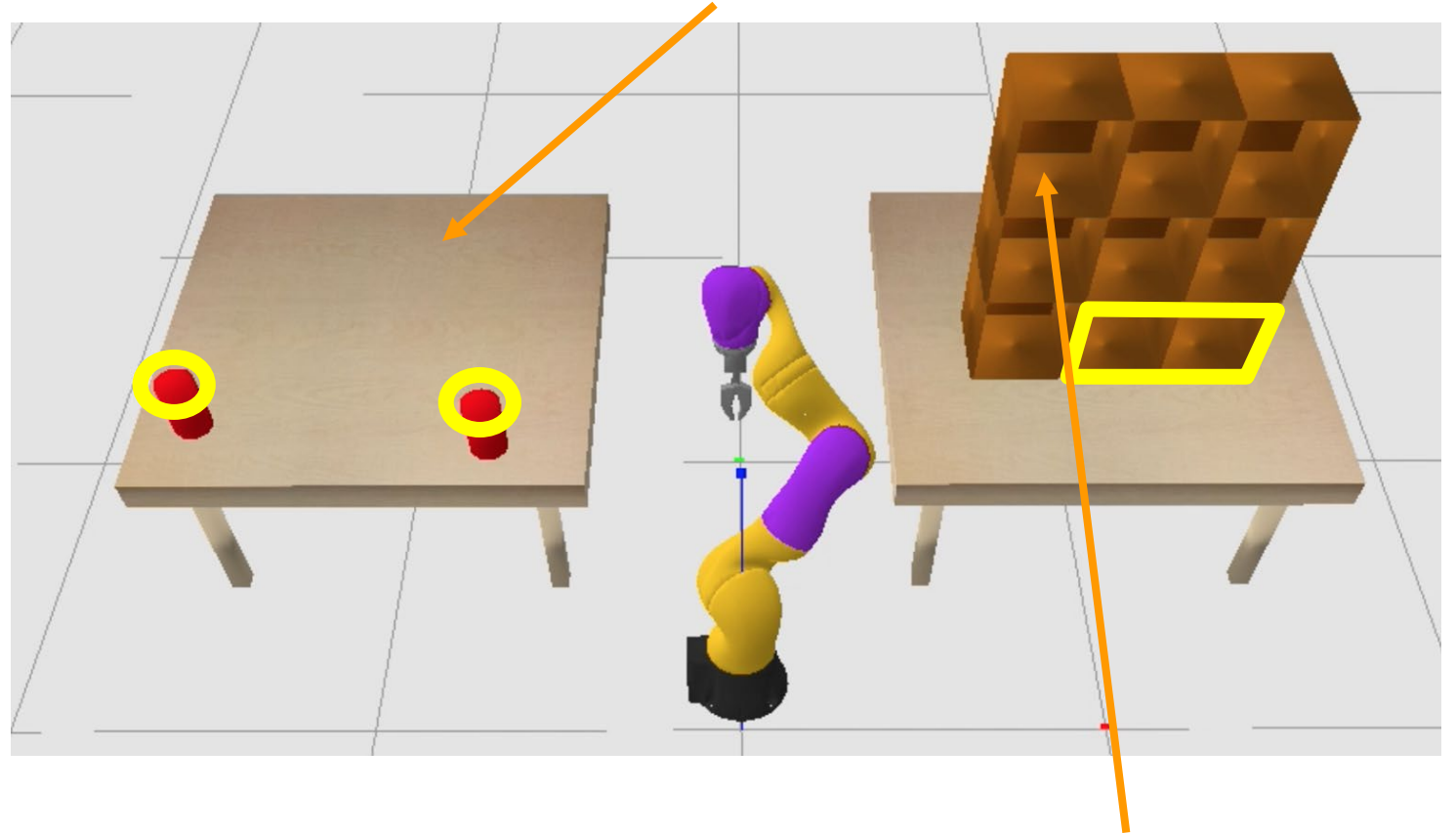
Constraints \Rightarrow Abstraction

Abstraction ignores other objects on table due to top-grasp constraint

Planning with a General-Purpose TAMP Planner



Much smaller branching factor

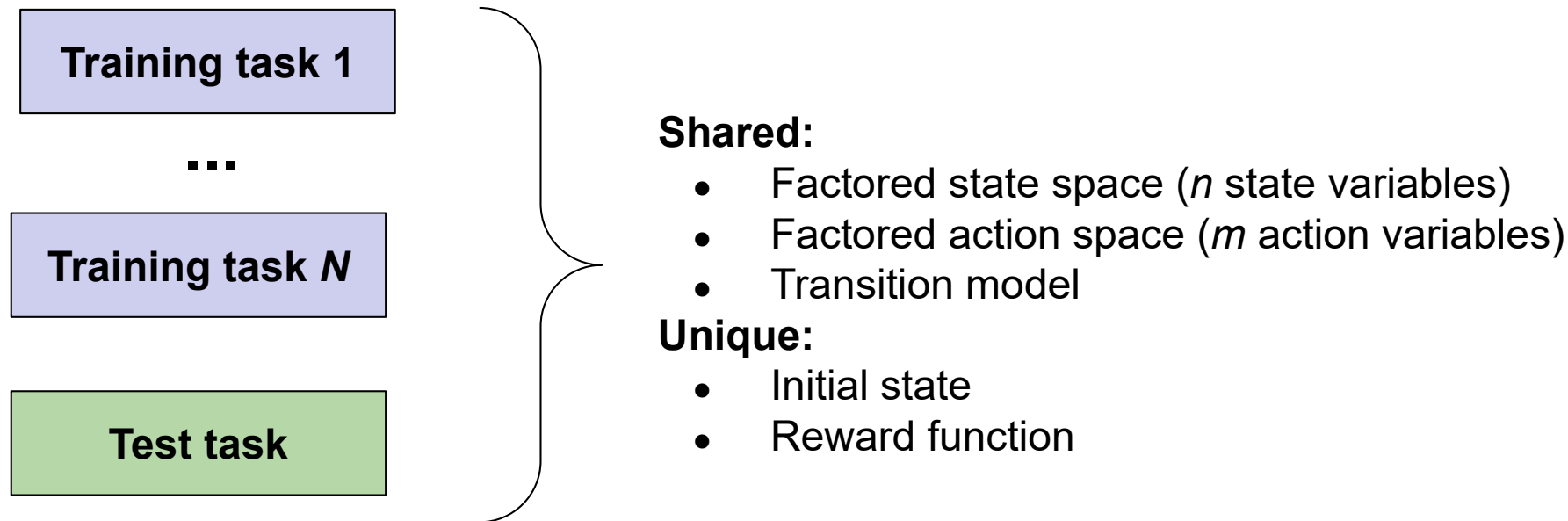


Abstraction ignores objects in other shelves

Foreshadowing Two Learning Problems

1. Given a constraint and a task, which variables are contextually independent in this task, under this constraint?
2. Given a task, which constraint should the agent impose on itself?

Setting: multiple tasks, each is a factored MDP



- Training time, then test time. Test task unknown at training
- Objective: in the test task, find a *good* policy *quickly*

Objective: find a good policy quickly!

$$J(\pi, \omega) = \mathbb{E} \left[\sum_{t=0}^H R(s_t) - \lambda \cdot \text{COMPUTECOST}(\pi, s_t) \right]$$

Policy

Task

Make a tasty dinner

But don't spend
forever thinking!

Objective: find a good policy quickly!

$$J(\pi, \omega) = \mathbb{E} \left[\sum_{t=0}^H R(s_t) - \lambda \cdot \text{COMPUTECOST}(\pi, s_t) \right]$$

Policy Task Make a tasty dinner But don't spend forever thinking!

- Extreme 1: **Plan for the full test task.** High reward but high compute cost.
- Extreme 2: **Learn a policy at training time.** Low compute cost but low reward (e.g., due to inability to generalize).

To maximize objective, make *abstraction* of test task

Our approach: *learn* to generate an *abstraction* of the test task in which
we can plan more efficiently

To maximize objective, make *abstraction* of test task

Our approach: *learn* to generate an *abstraction* of the test task in which we can plan more efficiently

Useful definition: A **relevant variable** is one that has *any* possibility of (directly or indirectly) affecting the reward function.

Context-Specific Abstract MDPs (CAMPs)

Given an MDP and a constraint on the states and actions, a CAMP is a new, smaller MDP that makes two modifications:

- Transition model updated so any transition violating the constraint sends you to *limbo* --- a sink state with infinite negative reward!



Context-Specific Abstract MDPs (CAMPs)

Given an MDP and a constraint on the states and actions, a CAMP is a new, smaller MDP that makes two modifications:

- Transition model updated so any transition violating the constraint sends you to *limbo* --- a sink state with infinite negative reward!
- Irrelevant state/action variables under this constraint are removed.



Context-Specific Abstract MDPs (CAMPs)

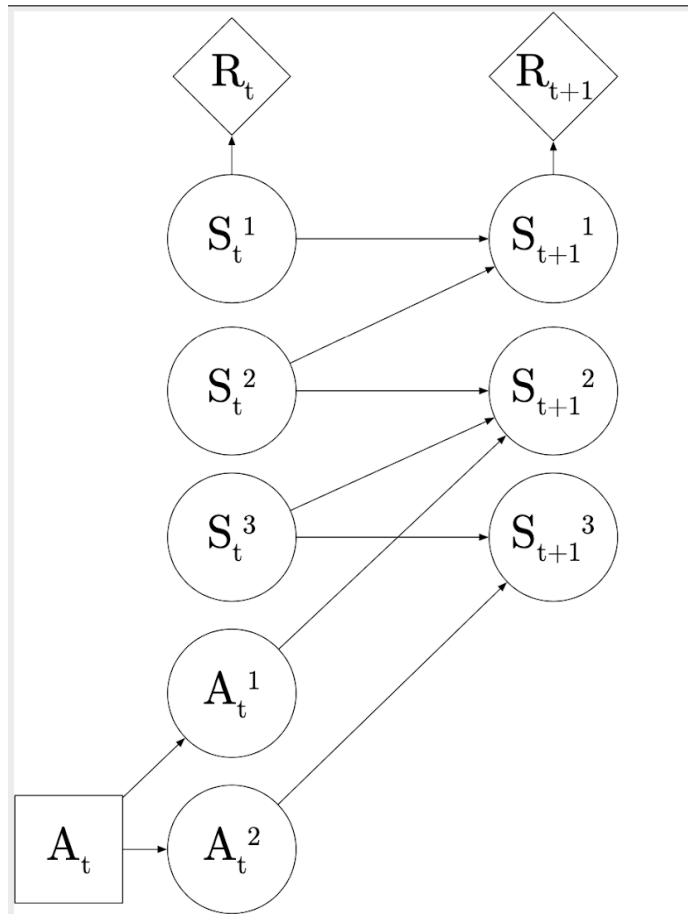
Given an MDP and a constraint on the states and actions, a CAMP is a new, smaller MDP that makes two modifications

- Transition model updated so any transition violating the constraint sends you to *limbo* --- a sink state with infinite negative reward!
- Irrelevant state/action variables under this constraint are removed.



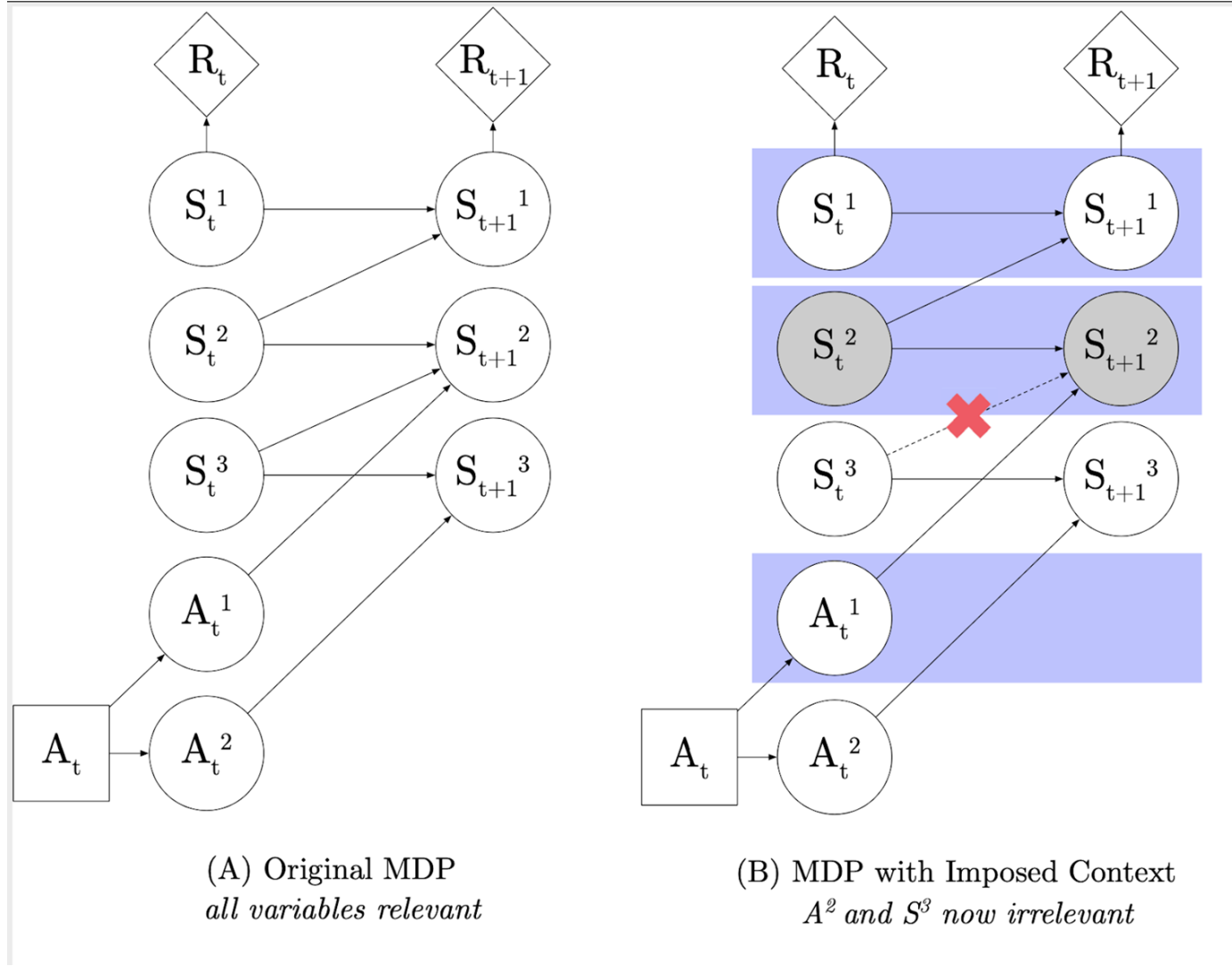
NOT optimality-preserving: the constraint may be so restrictive that the agent just can't do anything!

CAMP: Graphical example

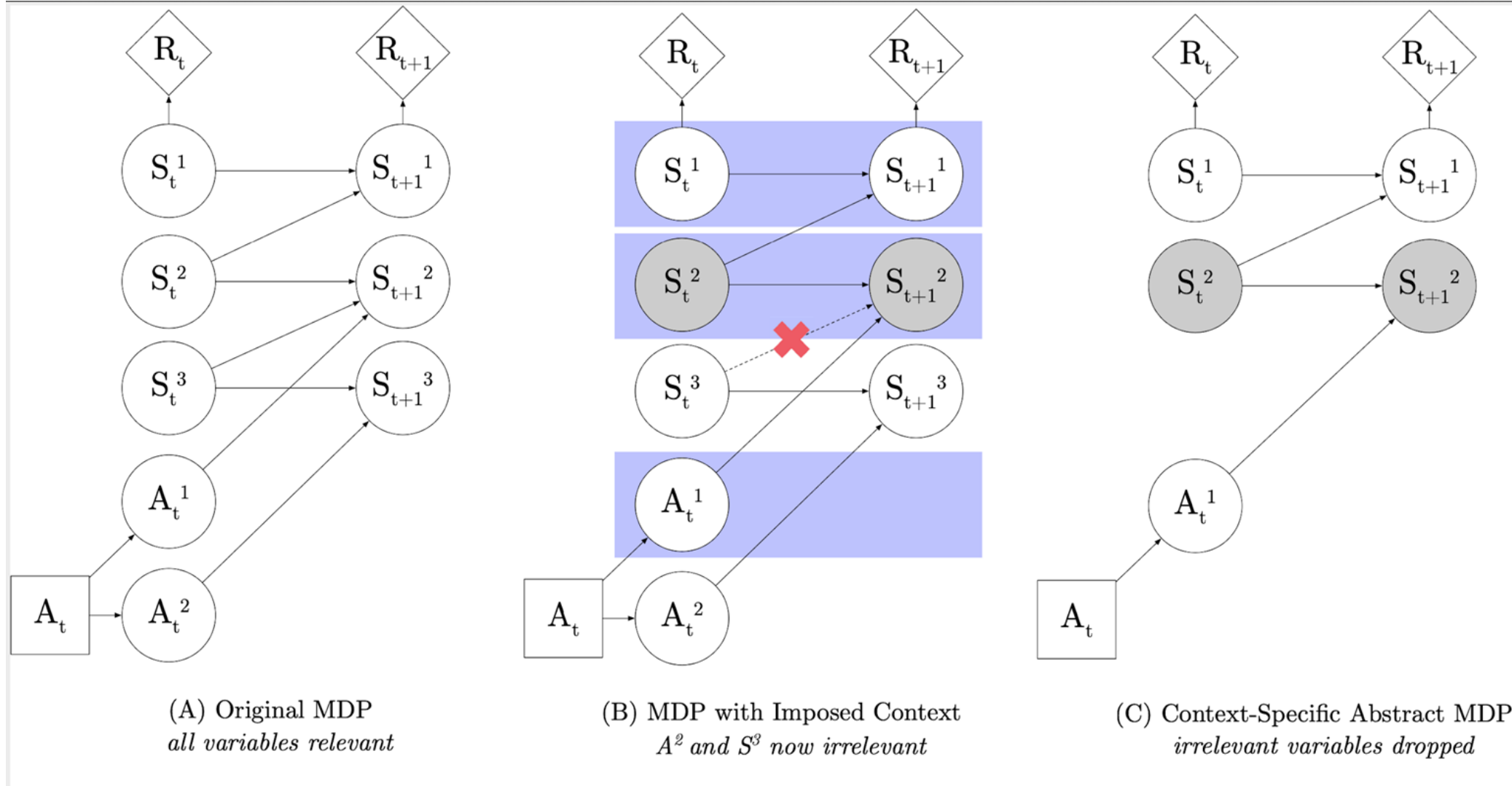


(A) Original MDP
all variables relevant

CAMP: Graphical example



CAMP: Graphical example



Two Itty Bitty Learning Problems

1. Given a constraint and a task, which variables are contextually independent in this task, under this constraint?
2. Given a task, which constraint should the agent impose on itself?

Recall that we have a set of training tasks and we can do whatever with them.

Two Itty Bitty Learning Problems

1. Given a constraint and a task, which variables are contextually independent in this task, under this constraint?
2. Given a task, which constraint should the agent impose on itself?

Recall that we have a set of training tasks and we can do whatever with them.

Approximating Context-Specific Independences

Suppose we're given a constraint. Want to find CSIs under that constraint.

Approximating Context-Specific Independences

Suppose we're given a constraint. Want to find CSIs under that constraint.

Sample a bunch of full (state, action) assignments under the constraint.

Approximating Context-Specific Independences

Suppose we're given a constraint. Want to find CSIs under that constraint.

Sample a bunch of full (state, action) assignments under the constraint.

For each pair of variables (V_1, V_2) not involved in the constraint:

For each sampled (state, action):

Check whether the transition distribution for V_1 changes by “wiggling around” V_2 within (state, action)

If not, (V_1, V_2) are independent in the context of the constraint.

Where does the space of constraints come from?

All possible settings of *discrete variables* & conjunctions/disjunctions of them (search bounded by hyperparameters like maximum domain size)

This is just a choice we made to have a domain-independent set of constraints; could also use domain-specific constraint families involving continuous variables.

Two Itty Bitty Learning Problems

1. Given a constraint and a task, which variables are contextually independent in this task, under this constraint?
2. **Given a task, which constraint should the agent impose on itself?**

Recall that we have a set of training tasks and we can do whatever with them.

Learning a Constraint Selector

Given a problem, what constraint should be imposed?

Learning a Constraint Selector

Given a problem, what constraint should be imposed?

Set this up as a **supervised learning** problem. Learn on training tasks.

Learning a Constraint Selector

Given a problem, what constraint should be imposed?

Set this up as a **supervised learning** problem. Learn on training tasks.

Input: (Some representation of) a task.

Output: Best constraint to impose.

Learning a Constraint Selector

Given a problem, what constraint should be imposed?

Set this up as a **supervised learning** problem. Learn on training tasks.

Input: (Some representation of) a task.

Output: Best constraint to impose.

$$\text{Best } J(\pi, \omega) = \mathbb{E} \left[\sum_{t=0}^H R(s_t) - \lambda \cdot \text{COMPUTECOST}(\pi, s_t) \right]$$

Training Task \nearrow

Learning a Constraint Selector

Given a problem, what constraint should be imposed?

Set this up as a **supervised learning** problem. Learn on training tasks.

Input: (Some representation of) a task.

Output: Best constraint to impose.

Training data: For each training task, score each candidate constraint.

Learning a Constraint Selector

Given a problem, what constraint should be imposed?

Set this up as a **supervised learning** problem. Learn on training tasks.

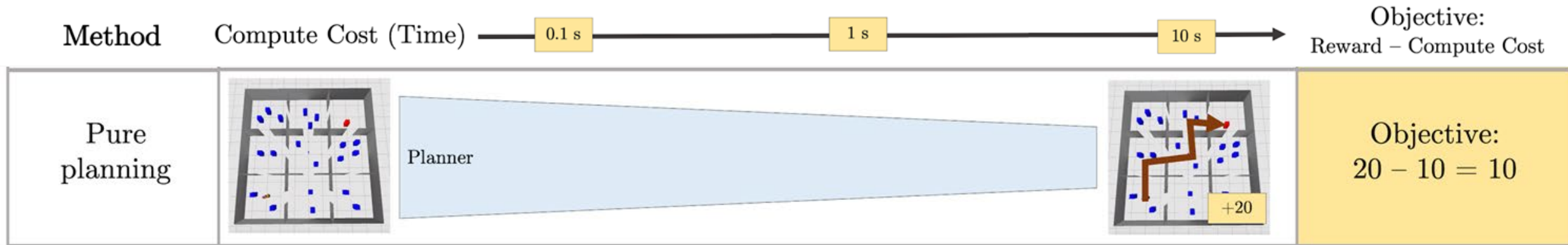
Input: (Some representation of) a task.

Output: Best constraint to impose.

Training data: For each training task, score each candidate constraint.

Model: neural network, so the paper can get accepted

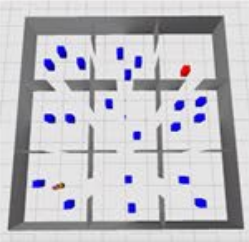

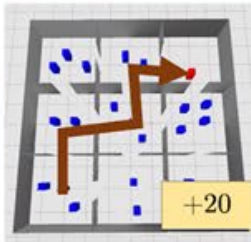
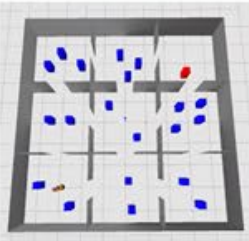

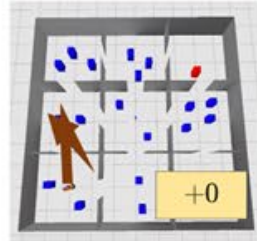
If you forget everything else, remember this slide



Learned

Given


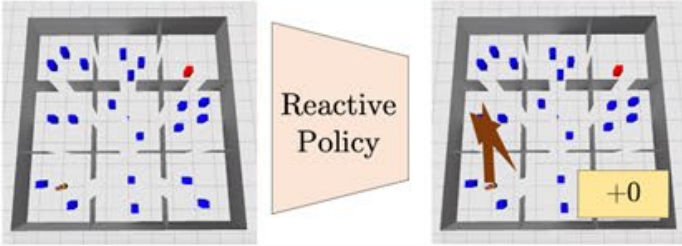

If you forget everything else, remember this slide

Method	Compute Cost (Time) → 0.1 s 1 s 10 s			Objective: Reward – Compute Cost	
Pure planning		<div>Planner</div> 			Objective: $20 - 10 = 10$
Pure policy learning		<div>Reactive Policy</div> 			Objective: $0 - 0.1 = -0.1$

Learned

Given

If you forget everything else, remember this slide

Method	<div> <div>0.1 s</div> <div>1 s</div> <div>10 s</div> </div> Compute Cost (Time)			Objective: Reward - Compute Cost
Pure planning				Objective: $20 - 10 = 10$
Pure policy learning				Objective: $0 - 0.1 = -0.1$
Ours: Learn to generate a CAMP & plan in it				Objective: $20 - 1 = 19$

Learned


Given

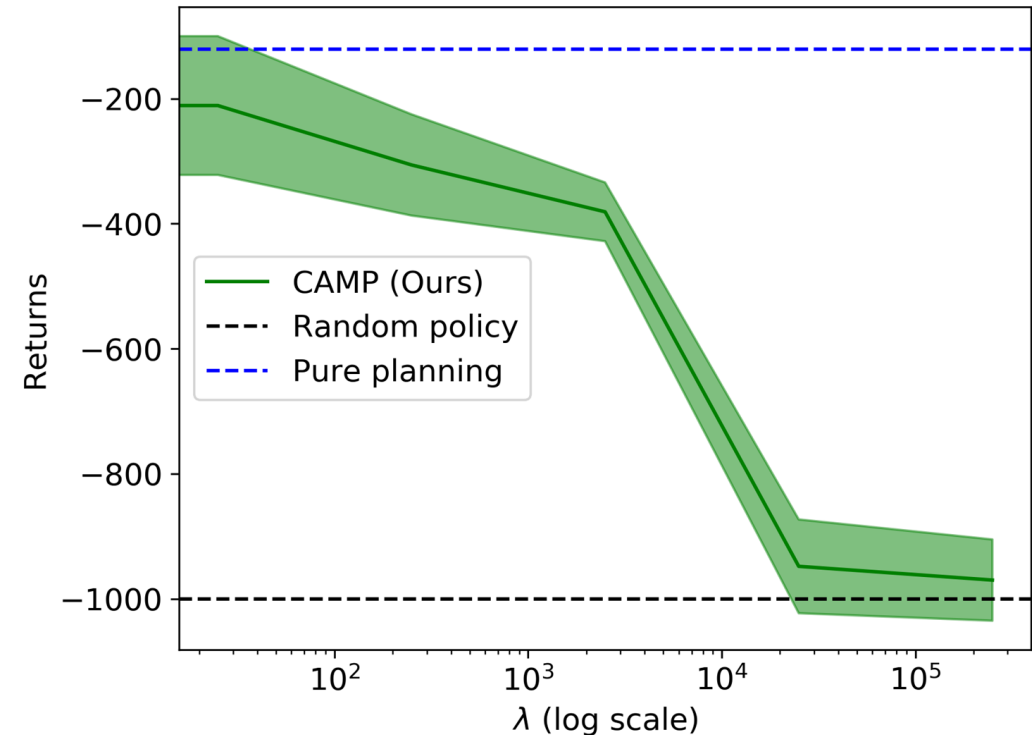
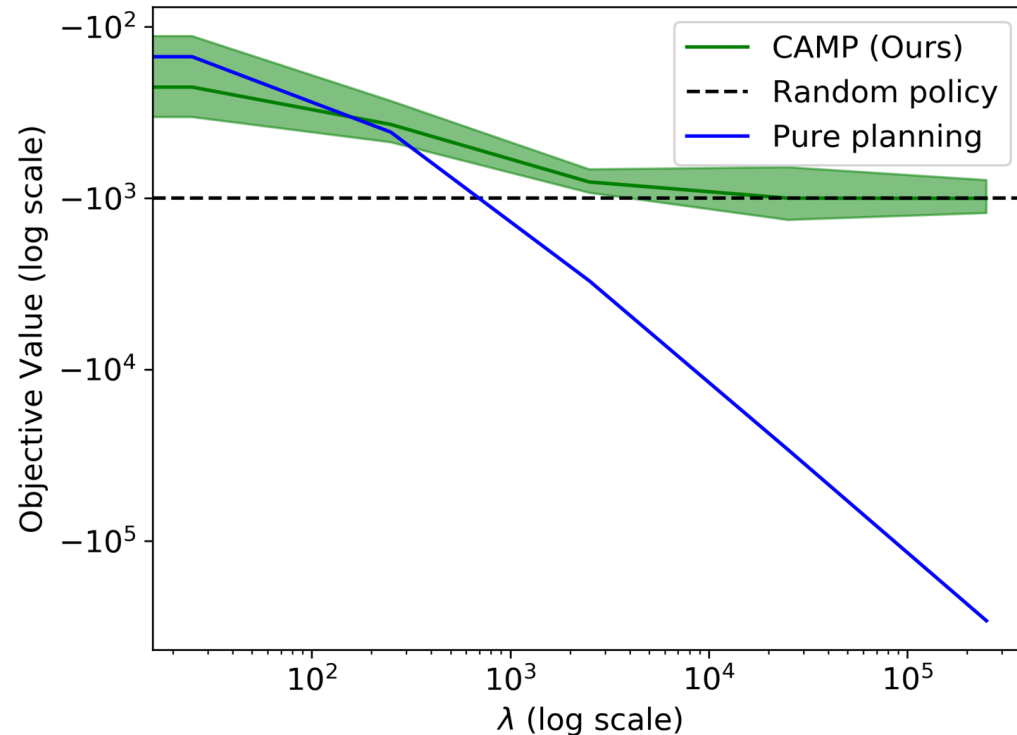
Experiments!

We compared CAMPs to pure planning and pure policy learning on a bunch of domains: gridworlds, classical planning, robotic NAMO, robotic manipulation.

Planners we tried: value iteration, MCTS, graph search, Fast-Downward, TAMP

Q: How does performance change with lambda?

$$J(\pi, \omega) = \mathbb{E} \left[\sum_{t=0}^H R(s_t) - \lambda \cdot \text{COMPUTECOST}(\pi, s_t) \right]$$




The Big Table

Method	Test Task Objective Value (St. Dev.)				
	Grid, MCTS	Grid, BFSReplan	Classical	NAMO	Manipulation
CAMP (ours)	70 (16)	21 (10)	-286 (9.6)	896 (63)	744 (94)
CAMP ablation	25 (11)	0.9 (24)	-308 (52)	707 (154)	453 (237)
Pure planning	6 (5)	-17 (11)	-414 (20)	242 (385)	335 (86)
Plan transfer	-7 (0.4)	-6 (15)	-467 (0.02)	141 (227)	21 (34)
Policy learning	-3 (4)	-11 (13)	-469 (0.2)	-0.2 (0.01)	-0.2 (0.01)
Task-conditioned	5 (5)	-2 (11)	-145 (0.4)	-0.3 (0.01)	-0.2 (0.02)
Stilman's [12]	-	-	-	826 (36)	-

- CAMP ablation: no variable dropping; only limbo.



List of Publications, Awards, Honors, etc.

Attributed to the Grant

Papers:

- C. Garrett, T Lozano-Perez, L P Kaelbling, PDDLStream: Integrating Symbolic Planners and Blackbox Samplers, ICAPS 2020.
- C. Garrett, C. Paxton, T Lozano-Perez, L P Kaelbling, D. Fox, Online Replanning in Belief Space for Partially Observable Task and Motion Problems, ICRA 2020.
- B. Kim, K. Lee, S.Lim, L P Kaelbling, T. Lozano-Perez, Monte Carlo Tree Search in continuous spaces using Voronoi optimistic optimization with regret bounds, AAAI 2020.
- T. Silver, K. Allen, A. Lew, L P Kaelbling, J. Tennenbaum, Few-shot Bayesian Imitation Learning with Logical Program Policies, AAAI 2020.
- B. Kim, Z. Wang, L P Kaelbling, T. Lozano-Perez, Learning to guide task and motion planning using score-space representation, Int. Journal of Robotics Research, 2019.
- F. Alet, A. Jeewajee, M. Bauza Villalonga, A. Rodriguez, T. Lozano-Perez, L. P. Kaelbling, Graph Element Networks: adaptive, structured computation and memory, ICML 2019.
- B. Kim, L. Shimanuki, Learning value functions with relational state representations for guiding task-and-motion planning, CoRL 2019
- R. Chitnis, T. Lozano-Perez, Learning Compact Models for Planning with Exogenous Processes, CoRL 2019
- T. Silver, R. Chitnis, A. Ajay, J. Tennenbaum, L.P. Kaelbling, Learning Skill Hierarchies from Predicate Descriptions and Self-Supervision, AAAI Workshop on Generalization in Planning, 2020.