

Building Generalist Robots with Integrated Learning and Planning

Jiayuan Mao

Towards Generalist Robots

Goal:

Having a robot that can do many tasks, across many environments.

Towards Generalist Robots

Goal:

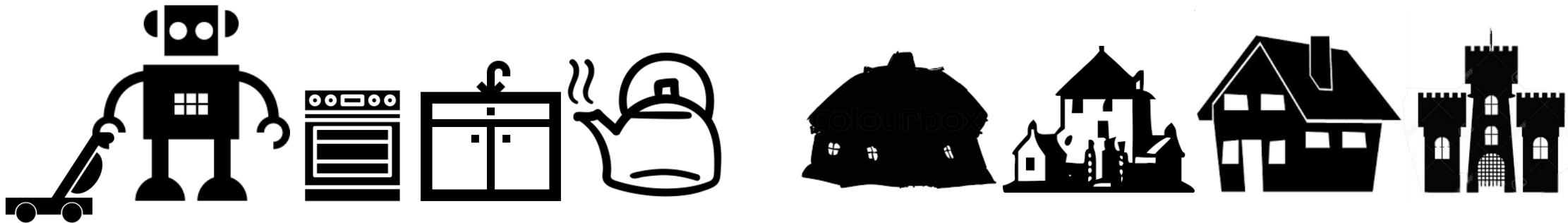
Having a robot that can do many tasks, across many environments.



Towards Generalist Robots

Goal:

Having a robot that can do many tasks, across many environments.



The robot should make long-horizon plans with rich contact with the environment, and generalize to unseen objects, states, and goals.

We want to achieve generalizations from a feasible amount of data.

Structures in Policies

$$\pi: \overbrace{(o, a)^*}^{\text{Historical Observations}} \rightarrow a$$

↓
Action

Structures in Policies

$$\pi: \overbrace{(o, a)^*}^{\text{Historical Observations}} \rightarrow a$$

Action

Historical Observations



Tabular Model



MLP



CNN



Q-Learning



Transformer



MCTS



What kinds of structures are
useful / needed for most physical
decision-making problems?



Classic AI Planning



Trajectory Optimization

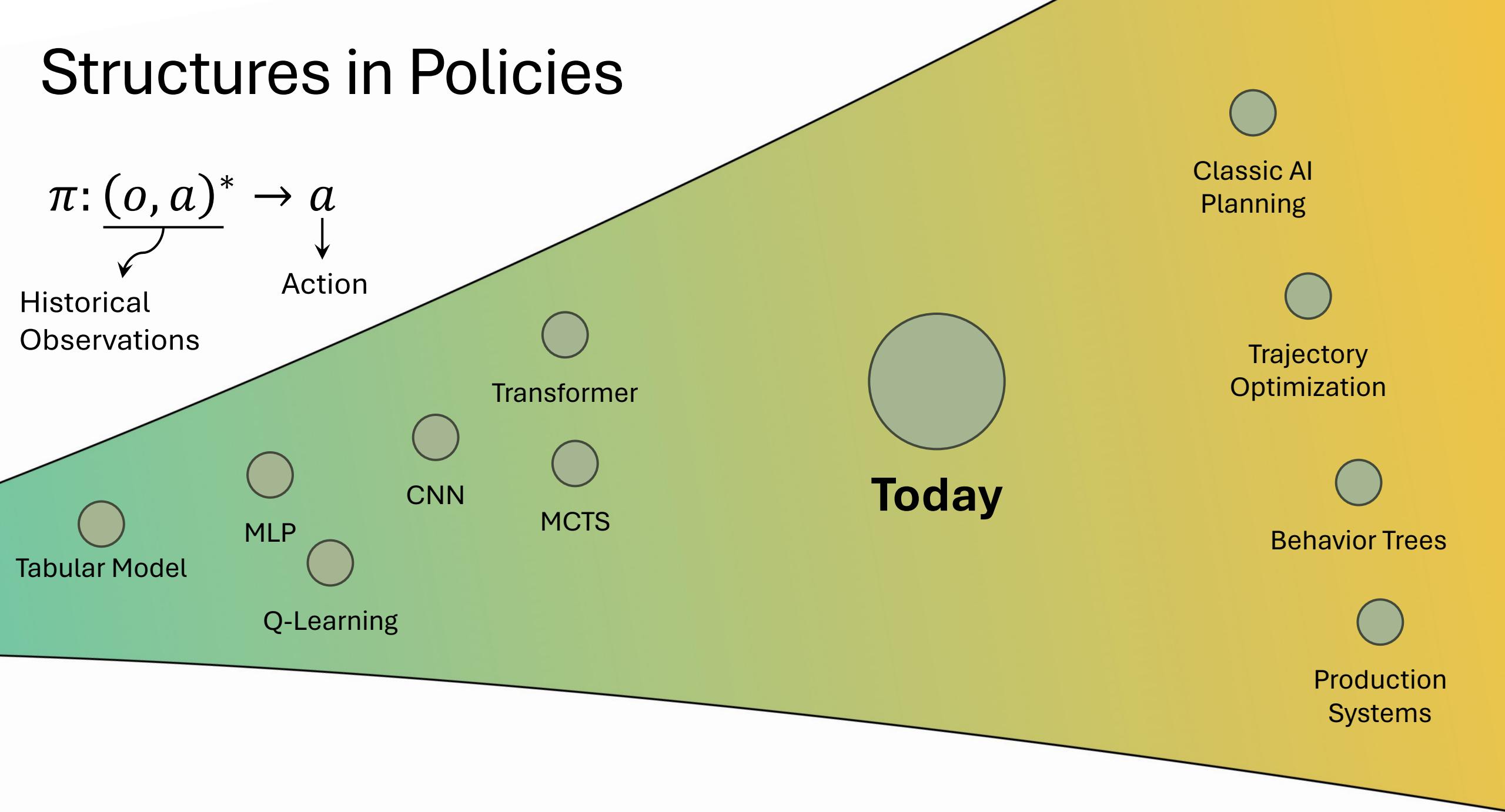


Behavior Trees

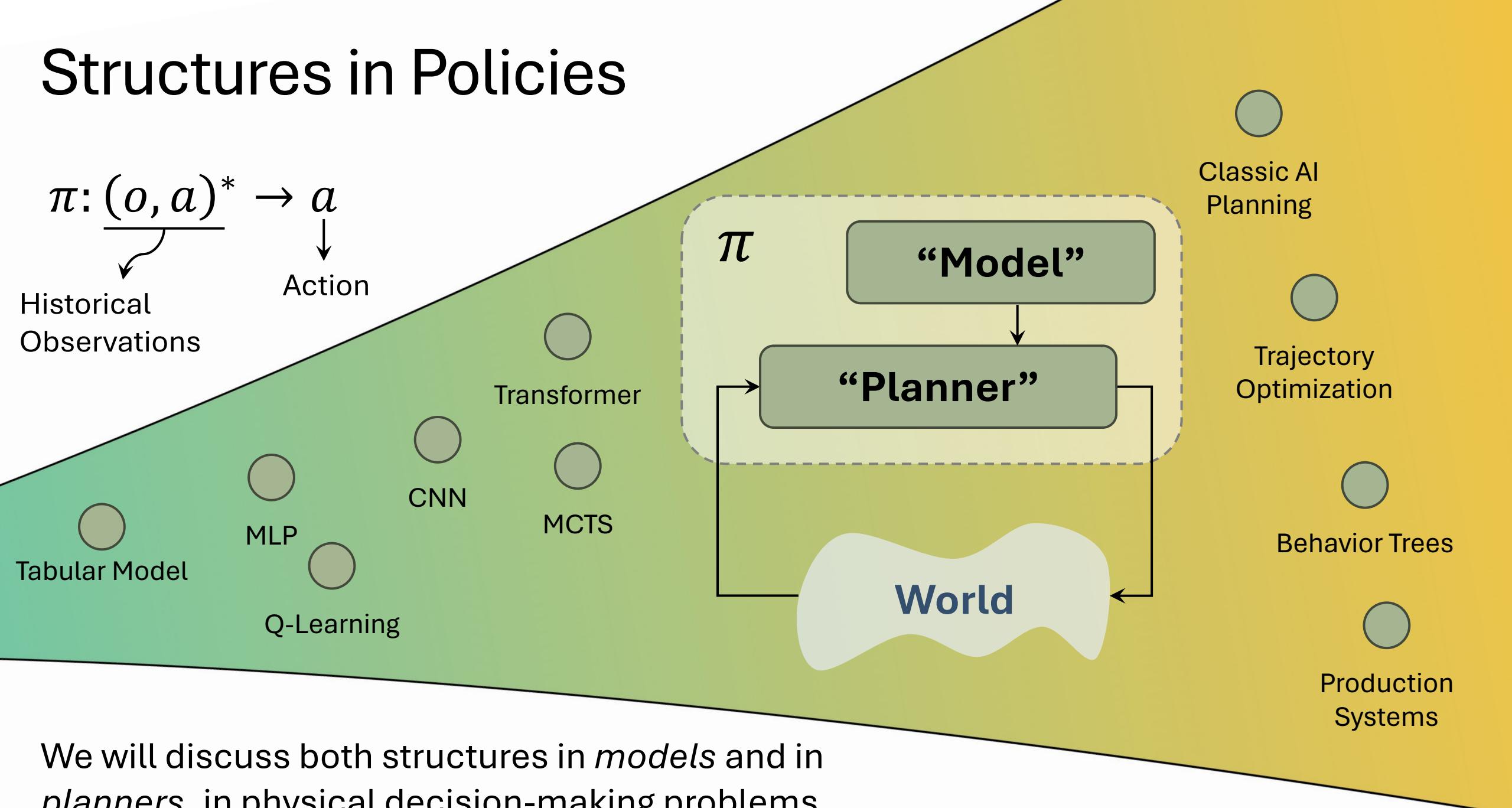


Production Systems

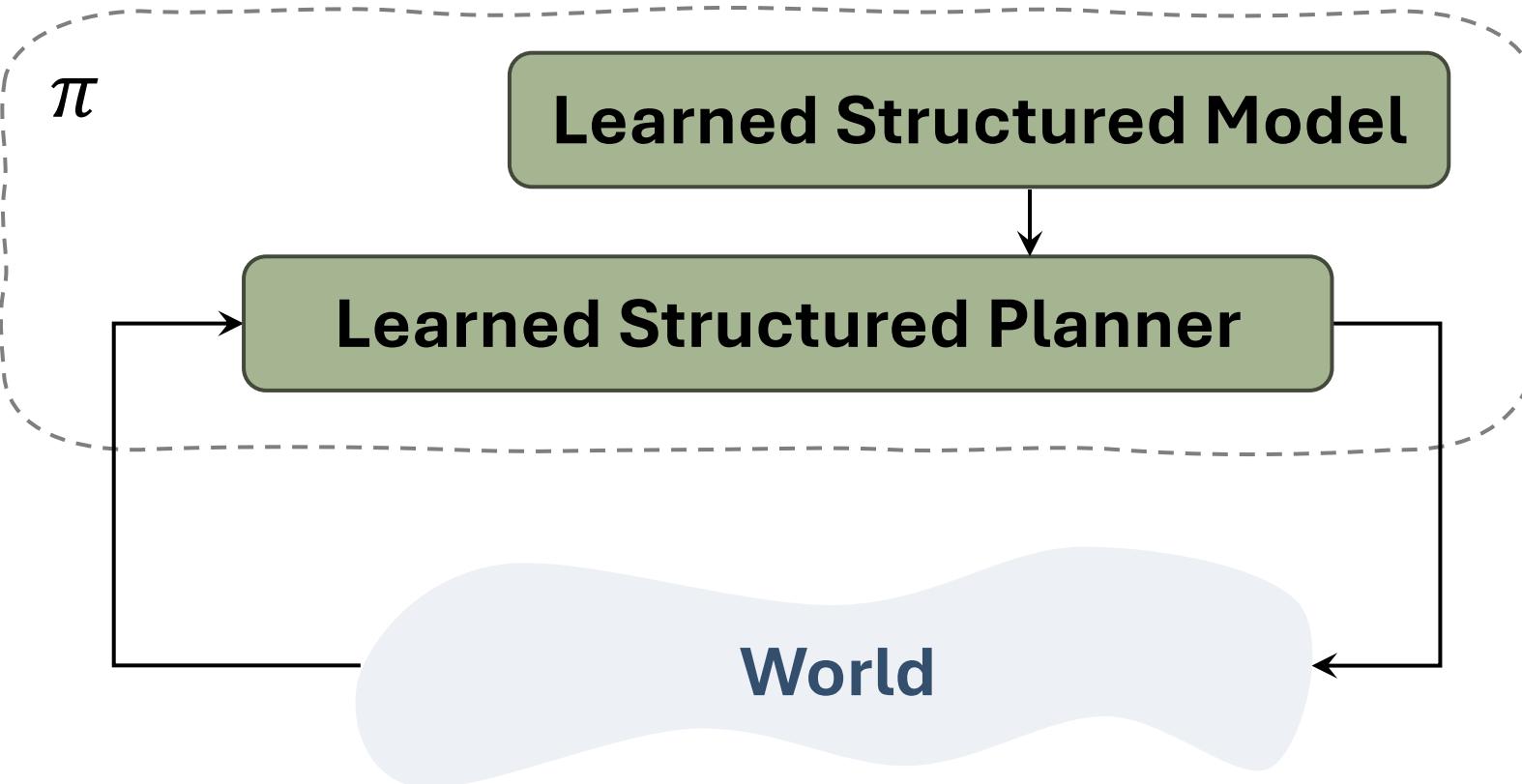
Structures in Policies



Structures in Policies



Learning Structured Representations



What structures in *models* and in *planners* do we need?
How do they improve our efficiency in learning and planning?
How will they help us achieve the goal of aggressive generalizations?

Task and Motion Planning — A Basic Framework



Instruction: Put all food items in the fridge.

Initial State: in(Cabbage, Pot),
on(Potato, Table), ...

Task Plan:

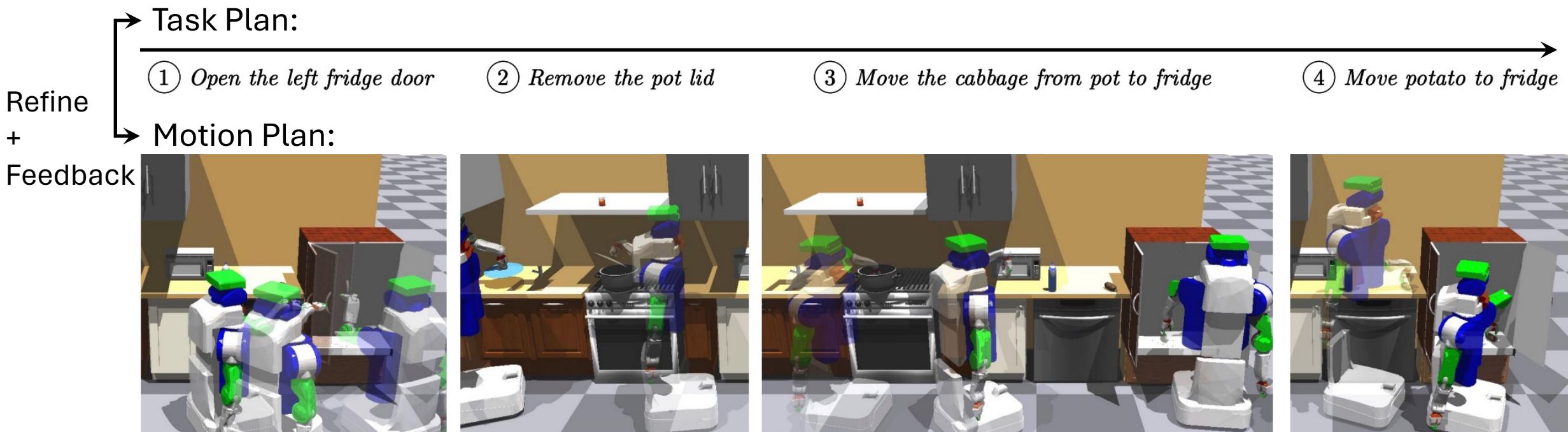
-
- ① Open the left fridge door
 - ② Remove the pot lid
 - ③ Move the cabbage from pot to fridge
 - ④ Move potato to fridge
- 

Task and Motion Planning — A Basic Framework



Instruction: Put all food items in the fridge.

Initial State: in(Cabbage, Pot),
on(Potato, Table), ...



Basic Elements in Task and Motion Planning

- Basic predicates.

```
predicate is-food(o: object)
```

```
    classifier: ...
```

```
predicate in(o: object, r: receptacle)
```

```
    classifier: ...
```

- Basic operators: preconditions, effects, and controllers.

```
action pick-up(o: object, p1: pose, g: grasp, t: trajectory)
```

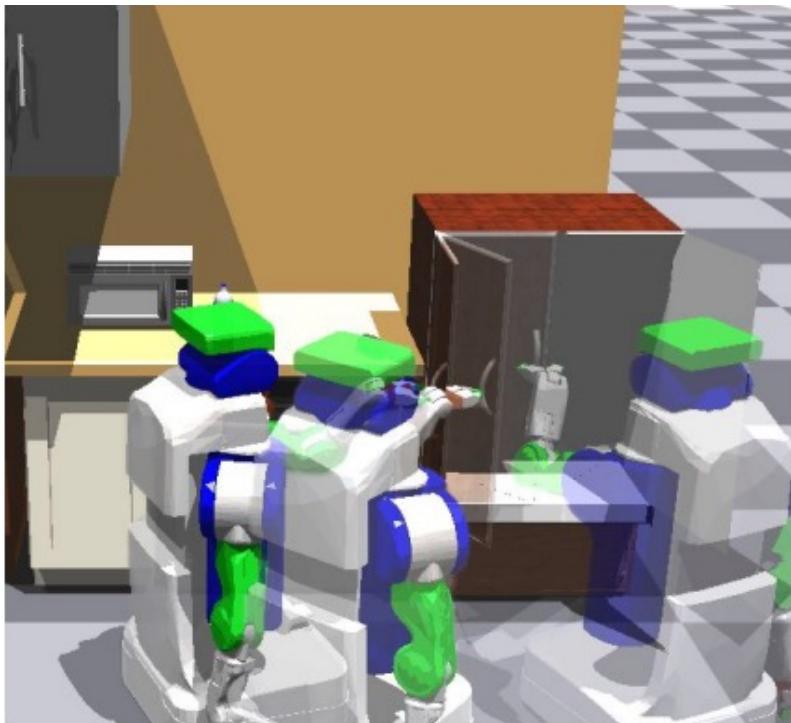
```
    pre: obj-at(p1), valid-trajectory(t, g, p1)
```

```
    eff: holding(o)
```

```
    controller: ...
```

Why Should We Factorize the Problem This Way?

Key Idea: *Build Compositional Abstractions.*



States are described using (state abstraction) :

- *on*(potato, table)
- *door-state*(fridge)

And they can be composed to form new concepts
“all food in fridge.”

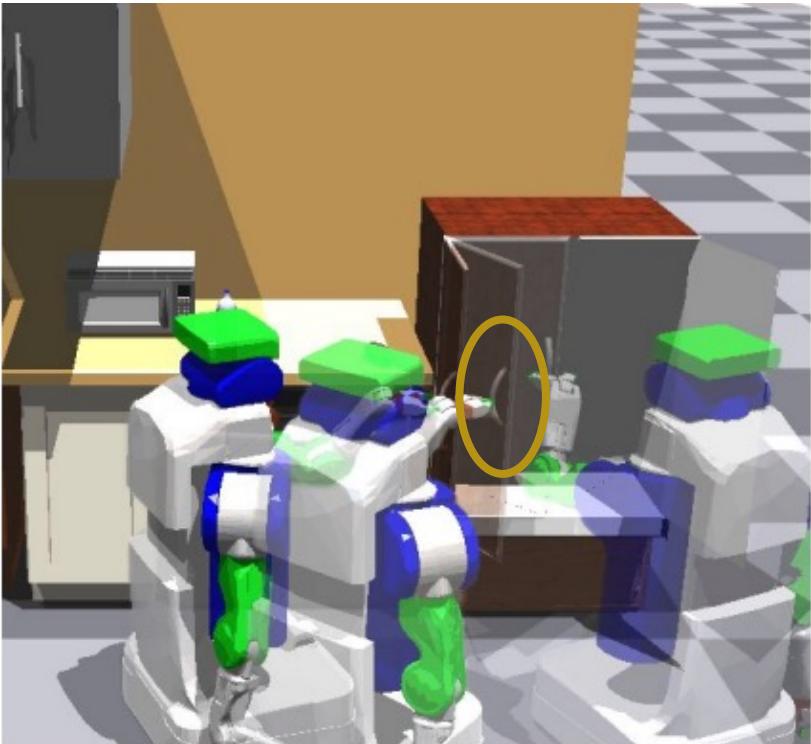
Actions are described using (temporal abstraction):

- *open*(door, degree, trajectory)
- *grasp*(object, pose, approaching-trajectory)

And they can be sequentially or hierarchically composed.

Why Are These Abstractions Helpful?

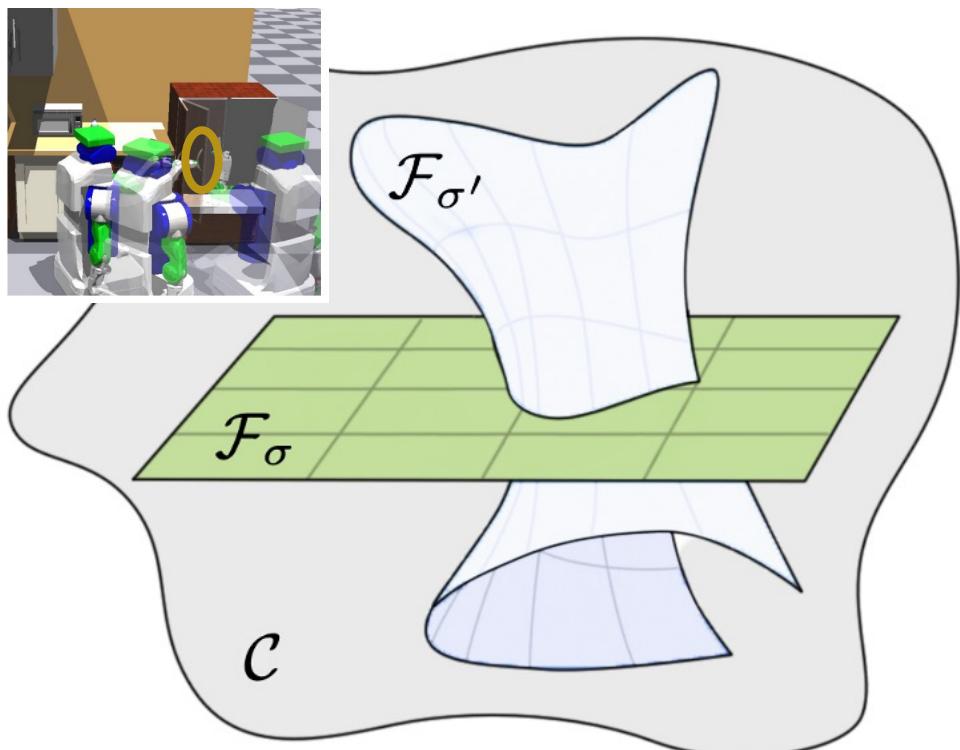
Compositional abstraction brings **sparsity** and **temporal decomposition**.



Why Are These Abstractions Helpful?

Compositional abstraction brings **sparsity** and **temporal decomposition**.

Models are sets of **low-dimensional manifolds** in the configuration space.



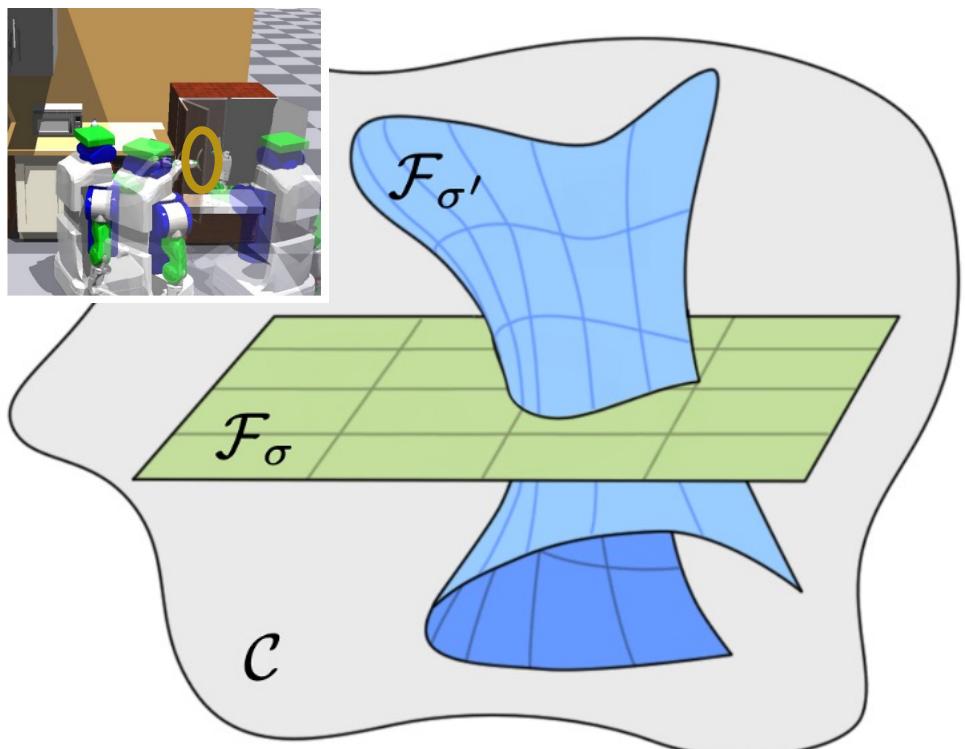
```
action move-to-grasp(o: obj, g: grasp, t: traj)
pre: robot-at(t[0]), valid-g(t[-1], pose(o), g)
eff: robot-at(t[-1]), holding(o, g)
controller: ...
```

Figure: Hauser and Latombe. Multi-Modal Motion Planning in Non-Expansive Spaces.

Why Are These Abstractions Helpful?

Compositional abstraction brings **sparsity** and **temporal decomposition**.

Models are sets of **low-dimensional manifolds** in the configuration space.



```
action move-to-grasp(o: obj, g: grasp, t: traj)
pre: robot-at(t[0]), valid-g(t[-1], pose(o), g)
eff: robot-at(t[-1]), holding(o, g)
controller: ...

action move-while-holding(o: obj, g: grasp, t: traj)
pre: robot-at(t[0]), holding(o, g), valid-obj-t(o, t)
eff: robot-at(t[-1]), obj-at(...)
```

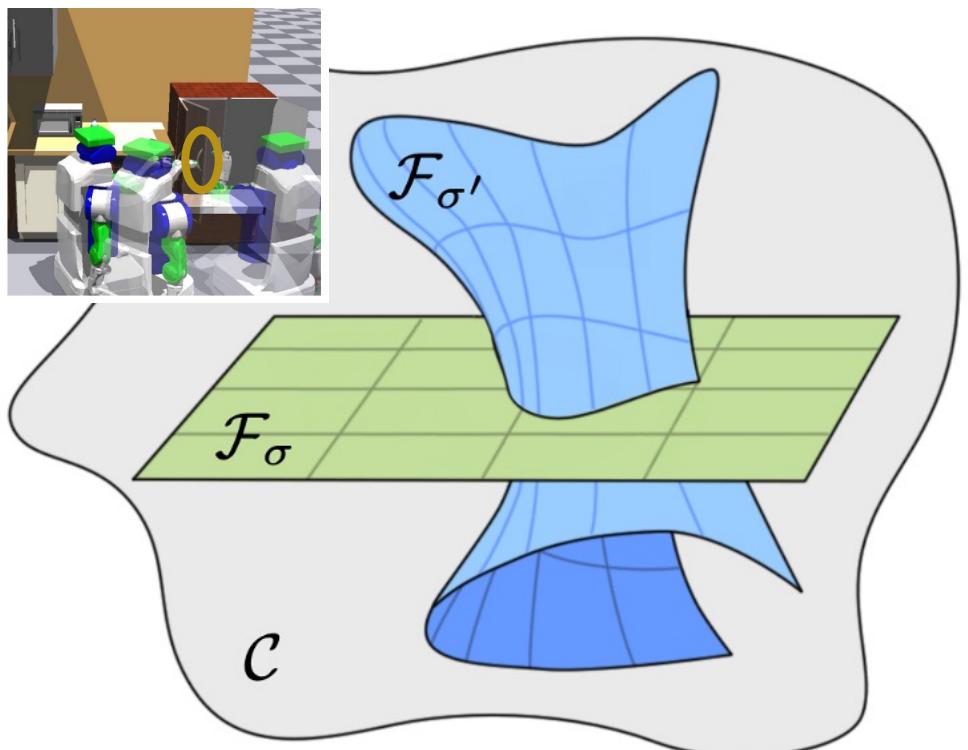
Figure: Hauser and Latombe. Multi-Modal Motion Planning in Non-Expansive Spaces.

Why Are These Abstractions Helpful?

Compositional abstraction brings **sparsity** and **temporal decomposition**.

Models are sets of **low-dimensional manifolds** in the configuration space.

They are connected at regions modeled by **preconditions** and **effects**.



```
action move-to-grasp(o: obj, g: grasp, t: traj)
  pre: robot-at(t[0]), valid-g(t[-1], pose(o), g)
  eff: robot-at(t[-1]), holding(o, g)
  controller: ...

action move-while-holding(o: obj, g: grasp, t: traj)
  pre: robot-at(t[0]), holding(o, g), valid-obj-t(o, t)
  eff: robot-at(t[-1]), obj-at(....)
  controller: ...
```

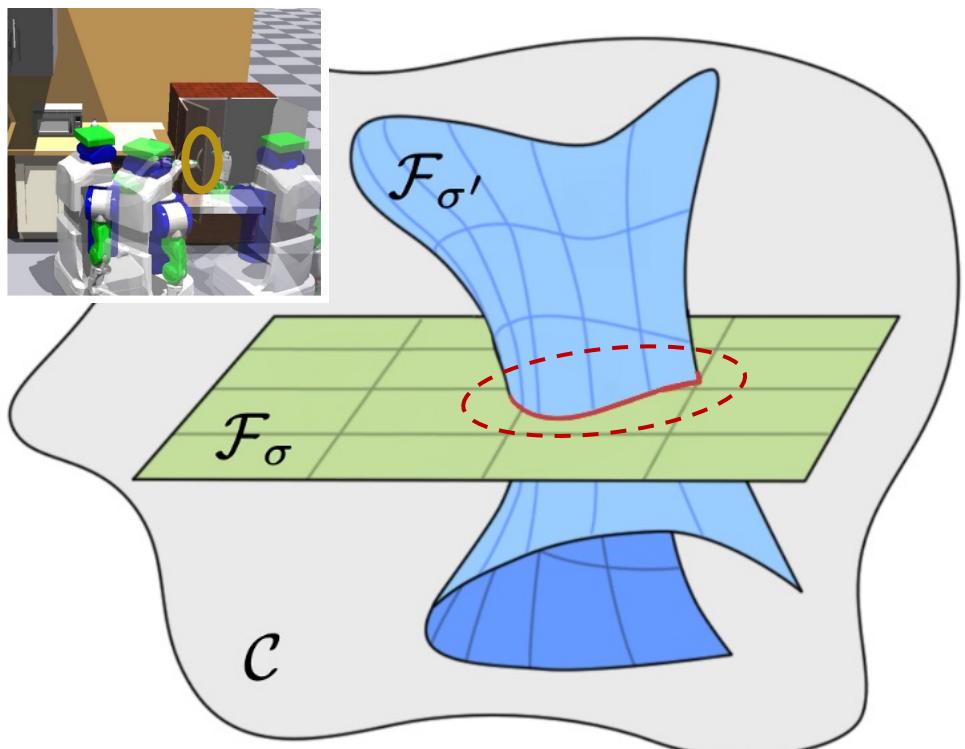
Figure: Hauser and Latombe. Multi-Modal Motion Planning in Non-Expansive Spaces.

Why Are These Abstractions Helpful?

Compositional abstraction brings **sparsity** and **temporal decomposition**.

Models are sets of **low-dimensional manifolds** in the configuration space.

They are connected at regions modeled by **preconditions** and **effects**.



```
action move-to-grasp(o: obj, g: grasp, t: traj)
  pre: robot-at(t[0]), valid-g(t[-1], pose(o), g)
  eff: robot-at(t[-1]), holding(o, g)
  controller: ...

action move-while-holding(o: obj, g: grasp, t: traj)
  pre: robot-at(t[0]), holing(o, g), valid-obj-t(o, t)
  eff: robot-at(t[-1]), obj-at(....)
  controller: ...
```

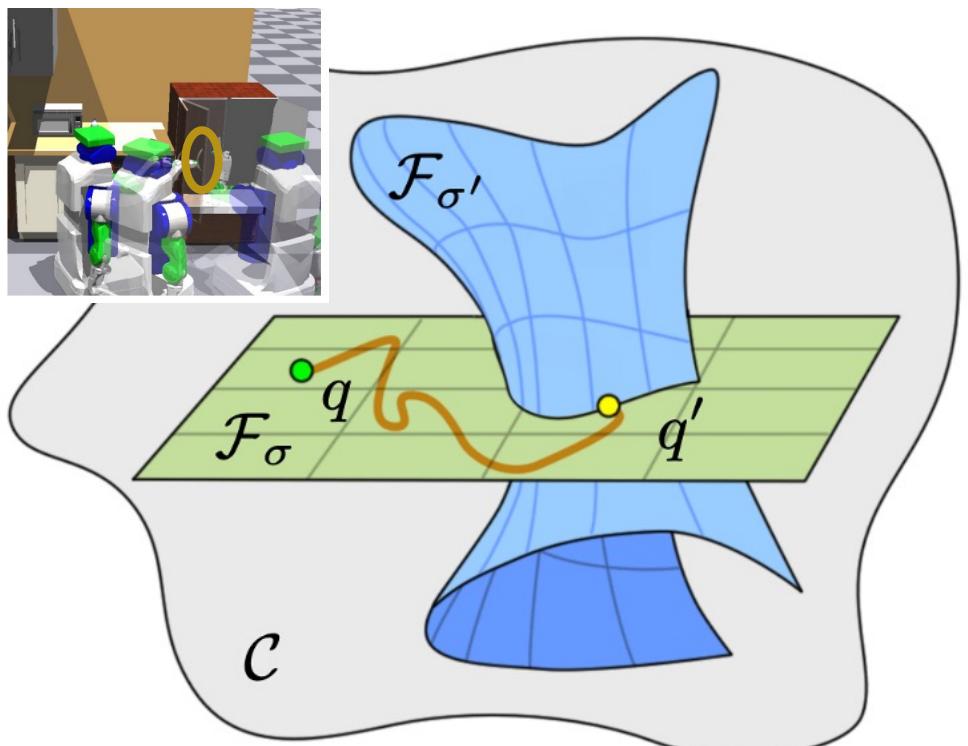
Figure: Hauser and Latombe. Multi-Modal Motion Planning in Non-Expansive Spaces.

Why Are These Abstractions Helpful?

Compositional abstraction brings **sparsity** and **temporal decomposition**.

Models are sets of **low-dimensional manifolds** in the configuration space.

They are connected at regions modeled by **preconditions** and **effects**.



```
action move-to-grasp(o: obj, g: grasp, t: traj)
  pre: robot-at(t[0]), valid-g(t[-1], pose(o), g)
  eff: robot-at(t[-1]), holding(o, g)
  controller: ...

action move-while-holding(o: obj, g: grasp, t: traj)
  pre: robot-at(t[0]), holding(o, g), valid-obj-t(o, t)
  eff: robot-at(t[-1]), obj-at(...)
```

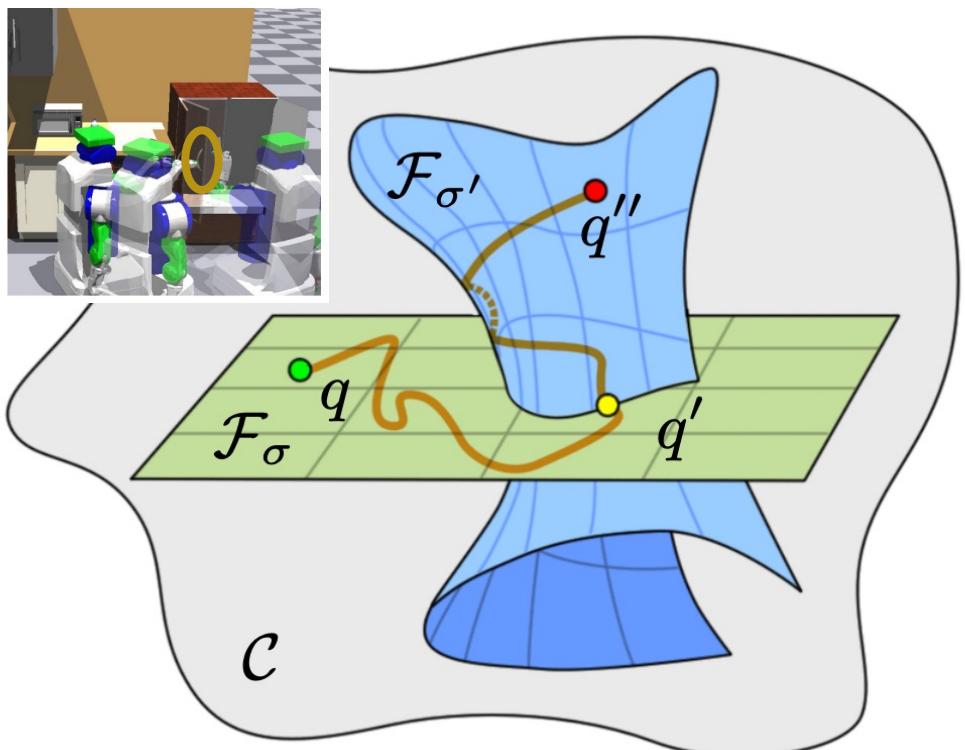
Figure: Hauser and Latombe. Multi-Modal Motion Planning in Non-Expansive Spaces.

Why Are These Abstractions Helpful?

Compositional abstraction brings **sparsity** and **temporal decomposition**.

Models are sets of **low-dimensional manifolds** in the configuration space.

They are connected at regions modeled by **preconditions** and **effects**.



```
action move-to-grasp(o: obj, g: grasp, t: traj)
  pre: robot-at(t[0]), valid-g(t[-1], pose(o), g)
  eff: robot-at(t[-1]), holding(o, g)
  controller: ...

action move-while-holding(o: obj, g: grasp, t: traj)
  pre: robot-at(t[0]), holding(o, g), valid-obj-t(o, t)
  eff: robot-at(t[-1]), obj-at(....)
  controller: ...
```

Figure: Hauser and Latombe. Multi-Modal Motion Planning in Non-Expansive Spaces.

Task and Motion Planning is General, But ...

There are a lot of details to be filled in:

① *Open the left fridge door*

- Where to grasp?
- How to move?
- How far?
- ...

② *Remove the pot lid*

- Where to grasp?
- Where to put?
- Any side-effects?
(e.g., hot item?)
- ...

③ *Move the cabbage from pot to fridge*

- Where to grasp?
- Where to place to be stable?
- Enough space for later items?
- Enough space for robot hand?
- Maybe need non-prehensile manipulation?
- What will happen to the cabbage?
- ...

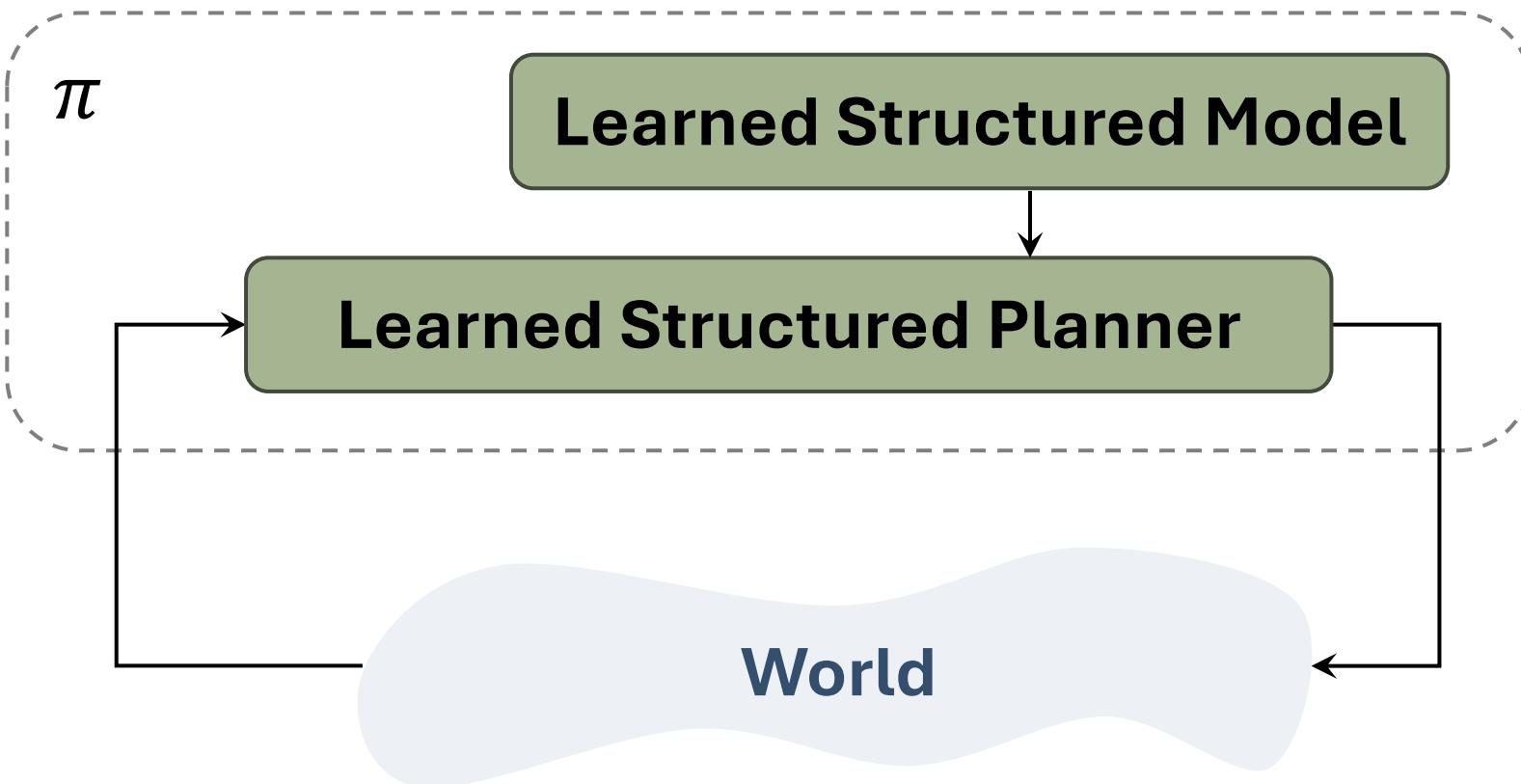
④ *Move potato to fridge*

- Where to grasp?
- Where to place to be ...
- How to organize the fridge?
- ...

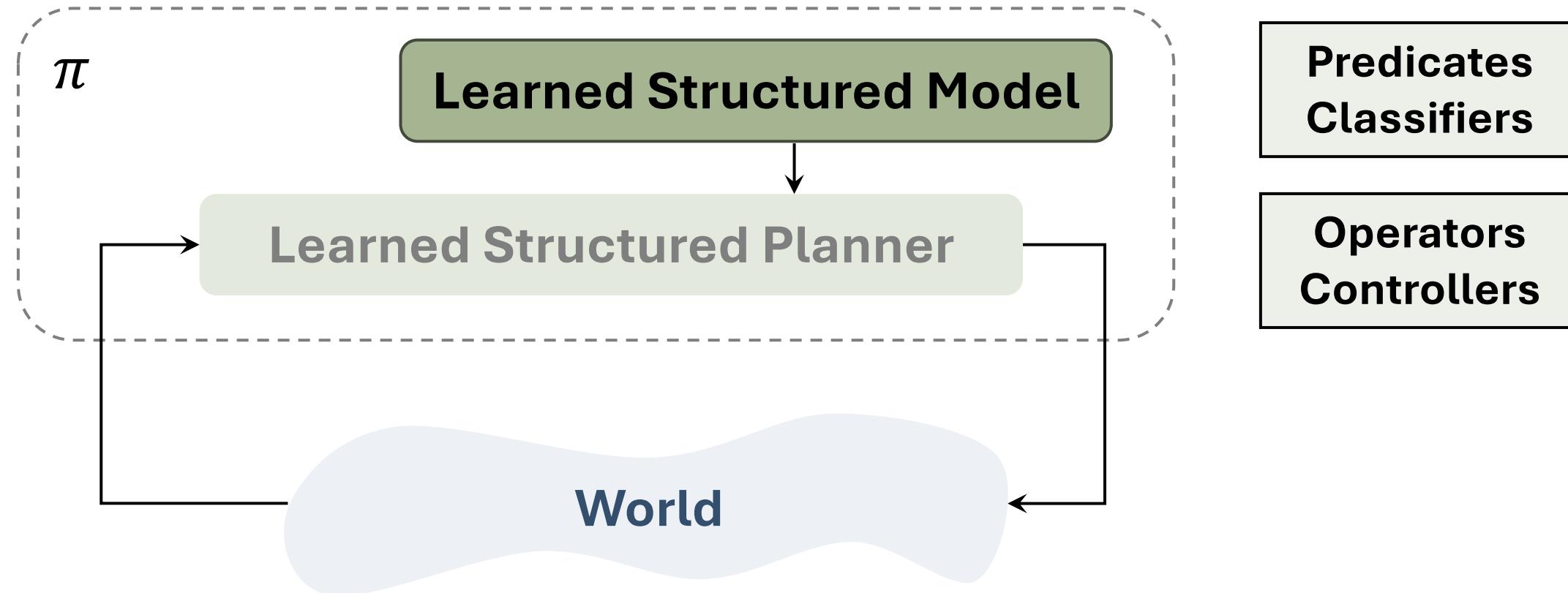
Let's Add Learning to Tackle These Challenges

- Task and motion planning is a general framework.
- Manually programming everything can be challenging, especially when dealing with perception and continuous parameters.
- We are interested in learning to tackle these challenges, in particular, learning structured representations for both the model and the planner.

Learning Structured Representations



Learning Structured Representations for Models



Learning Structured Models

- Model each “skill” as a sequence of *intra-mode movements and inter-mode transitions, with parameters.*

Learning Structured Models

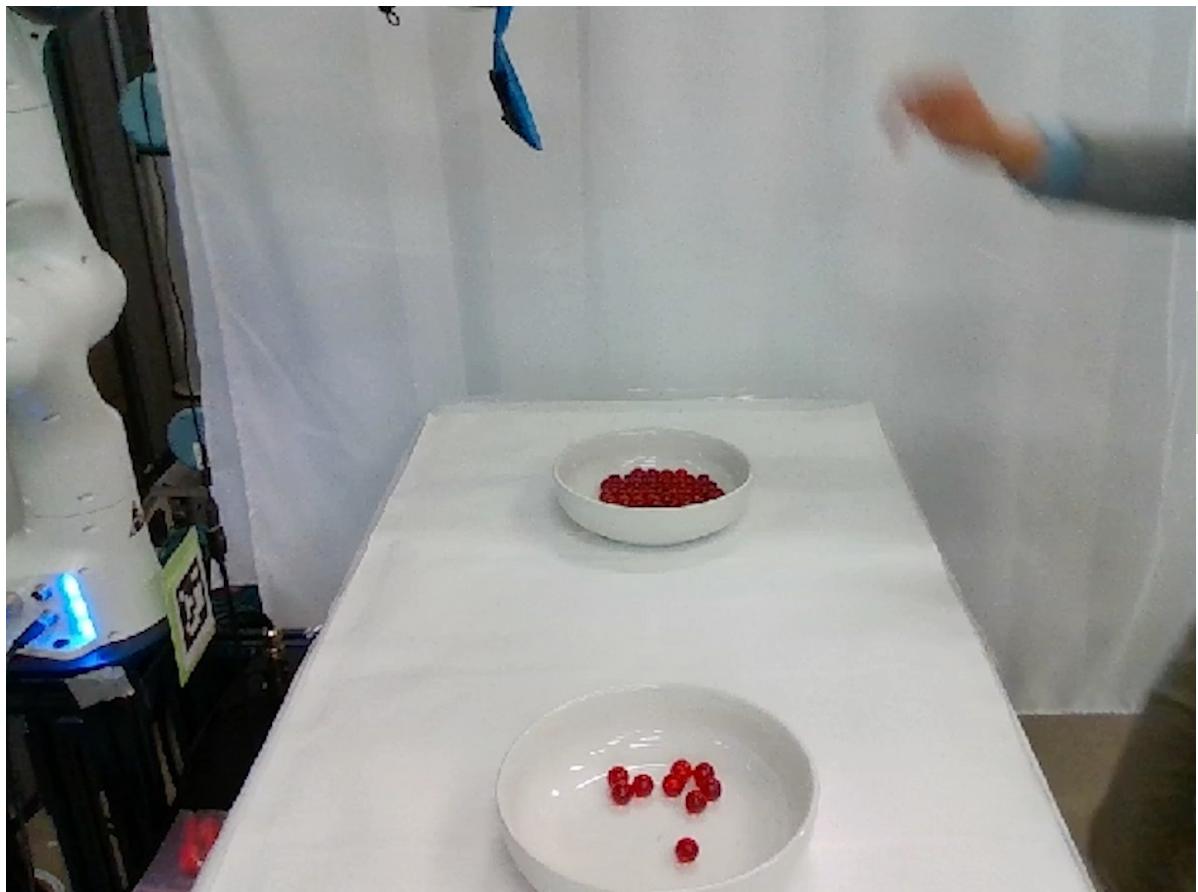
- Model each “skill” as a sequence of *intra-mode movements and inter-mode transitions, with parameters.*



```
action scoop(from, to, tool):  
    precondition: holding(tool), empty(tool)  
                  contains-marble(from)  
    body:  
        # move to the bowl to scoop from  
        move(tool, from)  
        # scoop the piles  
        move-with-contact(tool, from)  
        # move to the bowl to drop the piles  
        move(tool, to)  
        # drop the piles  
        move(tool)  
    effects: marble-update(from)  
             marble-update(to)
```

Learning Structured Models

- Model each “skill” as a sequence of *intra-mode movements and inter-mode transitions, with parameters.*



```
action scoop(from, to, tool):  
    precondition: holding(tool), empty(tool)  
                  contains-marble(from)  
    body:  
        # move to the bowl to scoop from  
        move(tool, from)  
        # scoop the piles  
        move-with-contact(tool, from)  
        # move to the bowl to drop the piles  
        move(tool, to)  
        # drop the piles  
        move(tool)  
    effects: marble-update(from)  
             marble-update(to)
```

Learning Structured Models

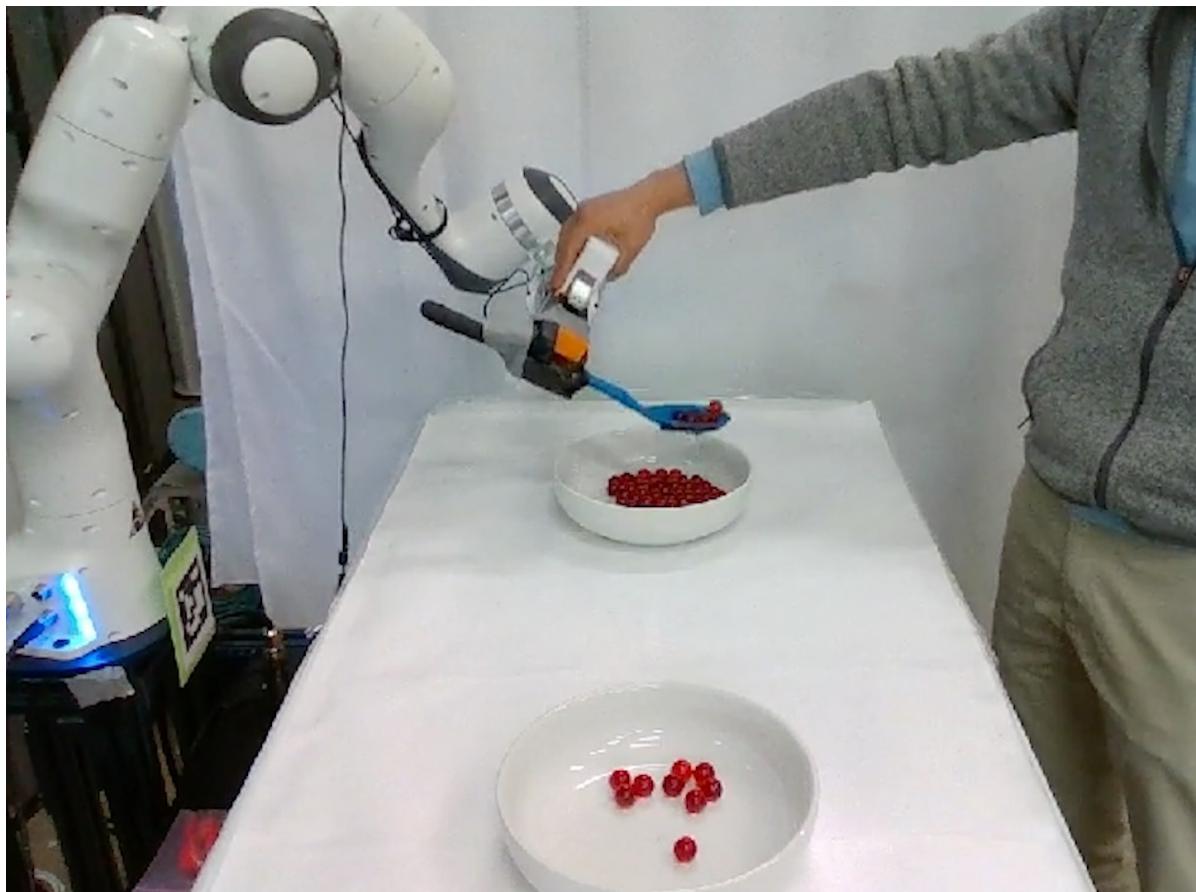
- Model each “skill” as a sequence of *intra-mode movements and inter-mode transitions, with parameters.*



```
action scoop(from, to, tool):  
    precondition: holding(tool), empty(tool)  
                  contains-marble(from)  
    body:  
        # move to the bowl to scoop from  
        move(tool, from)  
        # scoop the piles  
        move-with-contact(tool, from)  
        # move to the bowl to drop the piles  
        move(tool, to)  
        # drop the piles  
        move(tool)  
    effects: marble-update(from)  
             marble-update(to)
```

Learning Structured Models

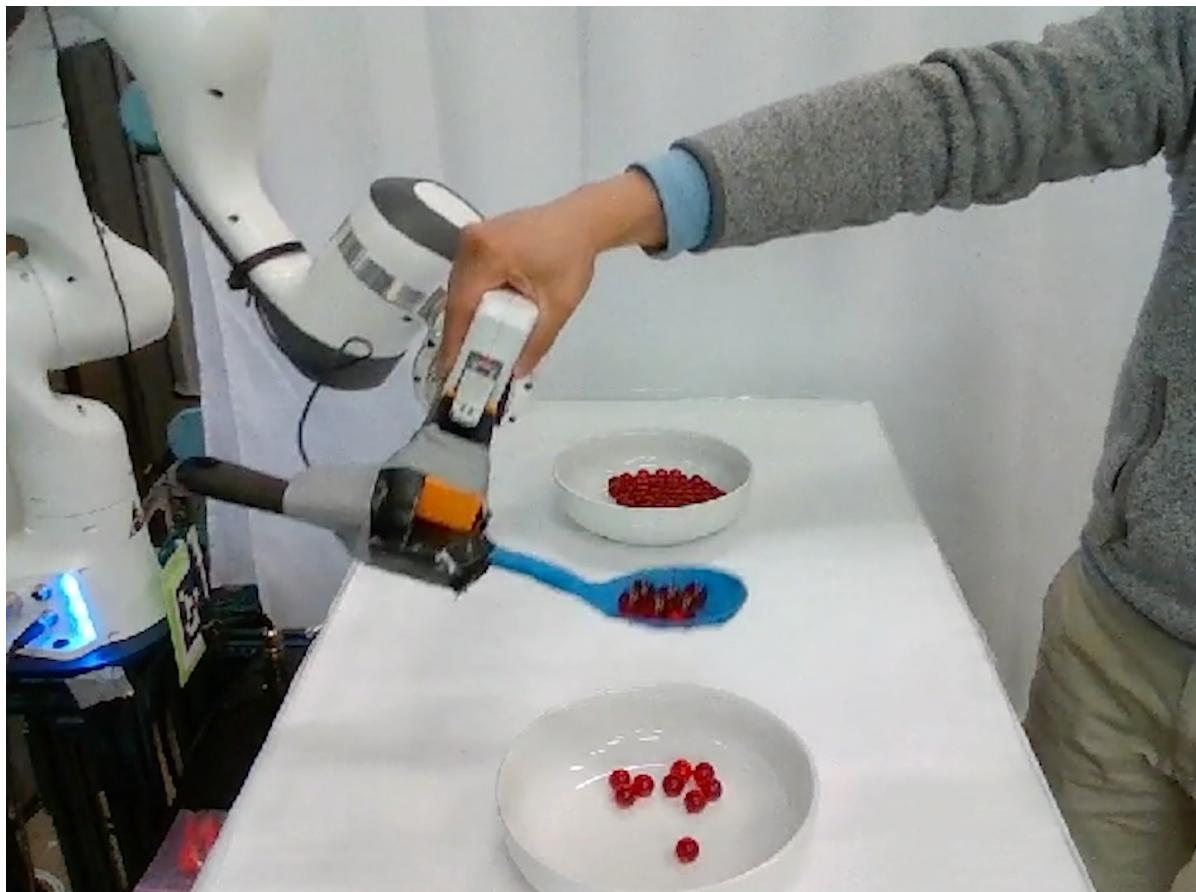
- Model each “skill” as a sequence of *intra-mode movements and inter-mode transitions, with parameters.*



```
action scoop(from, to, tool):  
    precondition: holding(tool), empty(tool)  
                contains-marble(from)  
    body:  
        # move to the bowl to scoop from  
        move(tool, from)  
        # scoop the piles  
        move-with-contact(tool, from)  
        # move to the bowl to drop the piles  
        move(tool, to)  
        # drop the piles  
        move(tool)  
    effects: marble-update(from)  
            marble-update(to)
```

Learning Structured Models

- Model each “skill” as a sequence of *intra-mode movements and inter-mode transitions, with parameters.*



```
action scoop(from, to, tool):  
    precondition: holding(tool), empty(tool)  
                  contains-marble(from)  
    body:  
        # move to the bowl to scoop from  
        move(tool, from)  
        # scoop the piles  
        move-with-contact(tool, from)  
        # move to the bowl to drop the piles  
        move(tool, to)  
        # drop the piles  
        move(tool)  
    effects: marble-update(from)  
             marble-update(to)
```

PDSketch

Integrated Domain Programming, Learning, and Planning



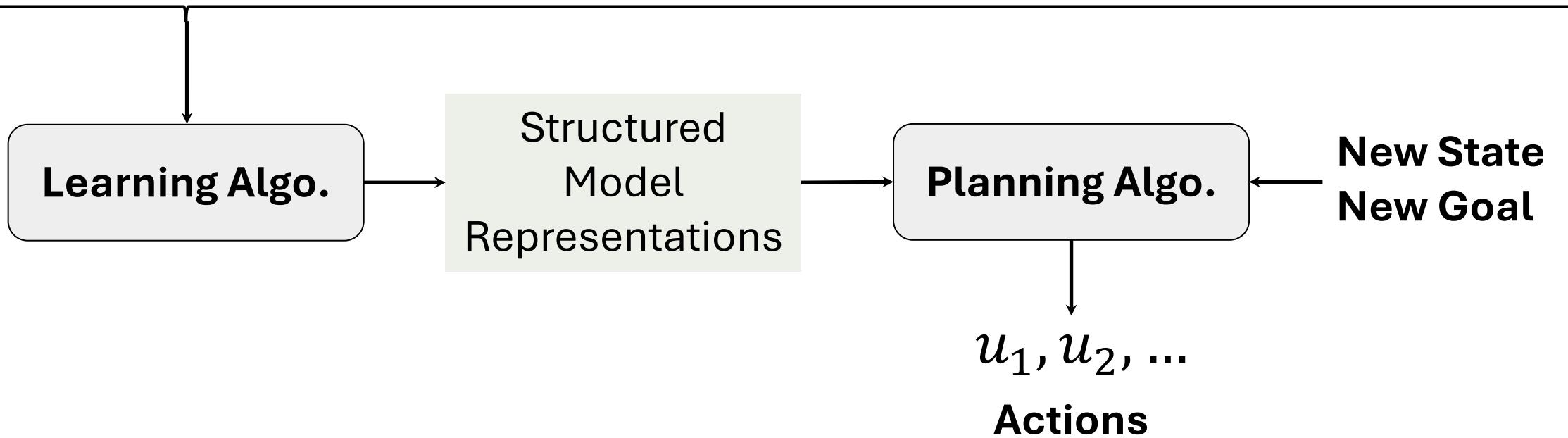
→ ... →



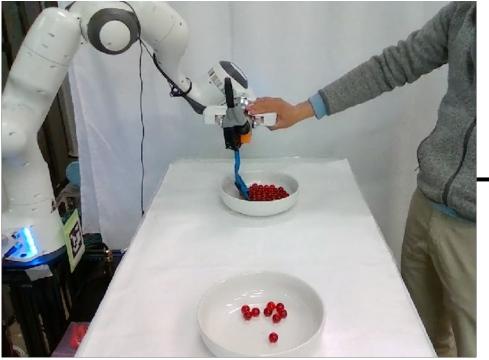
Training Data: Trajectories (e.g., demonstrations)

```
action scoop(from, to, tool):  
    precondition: ...  
    body: ...  
    effect: ...
```

Programmatic Definition (from Humans or LLMs)



The Objective of Learning



→ ... →



Training Data: Trajectories (e.g., demonstrations)

action **scoop**(*from*, *to*, *tool*):

precondition: **holding**(*tool*), **empty**(*tool*)
contains-marble(*from*)

body:

move(*tool*, *from*)

move-with-contact(*tool*, *from*)

move(*tool*, *to*)

move(*tool*)

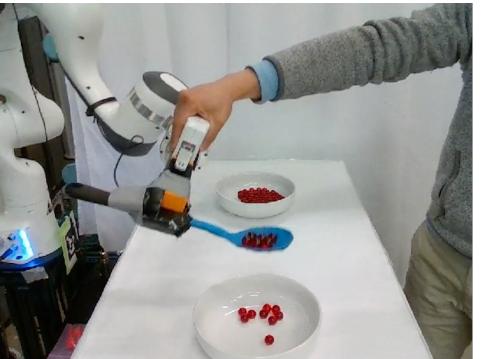
effects: **marble-update**(*from*)

marble-update(*to*)

The Objective of Learning



→ ... →



Training Data: Trajectories (e.g., demonstrations)

action **scoop**(*from*, *to*, *tool*):

 precondition: **holding**(*tool*), **empty**(*tool*)
 contains-marble(*from*)

 body:

move(*tool*, *from*)
 move-with-contact(*tool*, *from*)
 move(*tool*, *to*)
 move(*tool*)

 effects: **marble-update**(*from*)
 marble-update(*to*)

Target 1: Classifiers for predicates
Learning to classify objects and relations.

The Objective of Learning



→ ... →



Training Data: Trajectories (e.g., demonstrations)

action **scoop**(*from*, *to*, *tool*):

precondition: **holding**(*tool*), **empty**(*tool*)
contains-marble(*from*)

body:

move(*tool*, *from*)
move-with-contact(*tool*, *from*)
move(*tool*, *to*)
move(*tool*)

effects: **marble-update**(*from*)
marble-update(*to*)

Target 1: Classifiers for predicates.
Learning to classify objects and relations.

Target 2: Controllers for sub-actions.

The Objective of Learning



→ ... →



Training Data: Trajectories (e.g., demonstrations)

action **scoop**(*from*, *to*, *tool*):

precondition: **holding**(*tool*), **empty**(*tool*)
contains-marble(*from*)

body:

move(*tool*, *from*)

move-with-contact(*tool*, *from*)

move(*tool*, *to*)

move(*tool*)

effects: **marble-update**(*from*)
marble-update(*to*)

Target 1: Classifiers for predicates.
Learning to classify objects and relations.

Target 2: Controllers for sub-actions.

Target 3: Transition models.

The Objective of Learning



→ ... →



Training Data: Trajectories (e.g., demonstrations)

action **scoop**(*from*, *to*, *tool*):

 precondition: **holding**(*tool*), **empty**(*tool*)
 contains-marble(*from*)

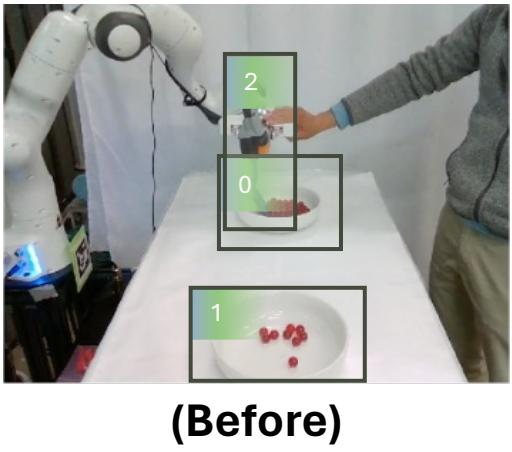
 body:

move(*tool*, *from*)
 move-with-contact(*tool*, *from*)
 move(*tool*, *to*)
 move(*tool*)

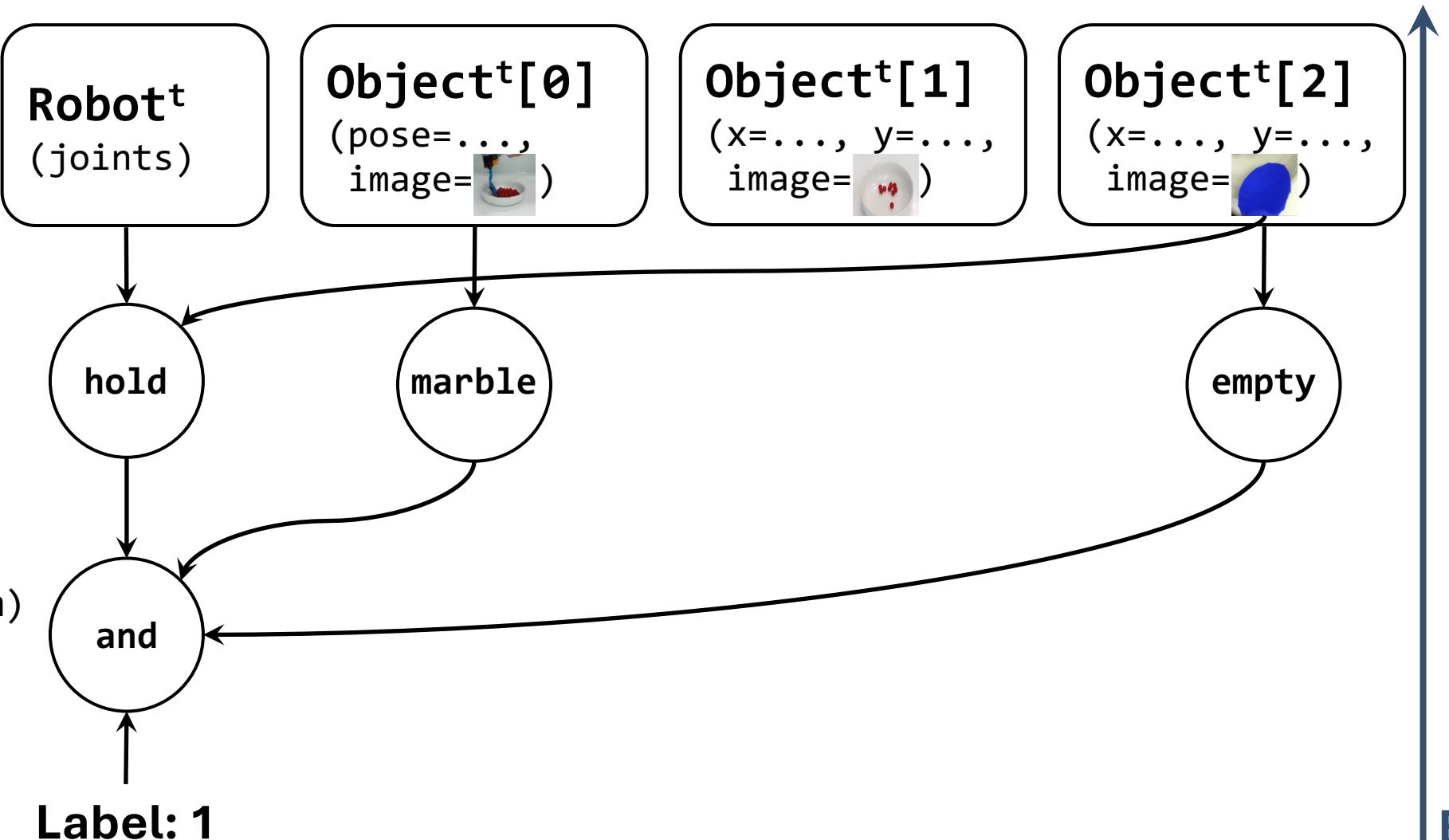
 effects: **marble-update**(*from*)
 marble-update(*to*)

Target 1: Classifiers for predicates
Learning to classify objects and relations.

Learning Classifiers by Evaluating Preconditions



precondition:
holding(tool),
empty(tool)
contains-marble(from)



Back
Prop

The Objective of Learning



→ ... →



Training Data: Trajectories (e.g., demonstrations)

action **scoop**(*from*, *to*, *tool*):

 precondition: **holding**(*tool*), **empty**(*tool*)
 contains-marble(*from*)

 body:

move(*tool*, *from*)

move-with-contact(*tool*, *from*)

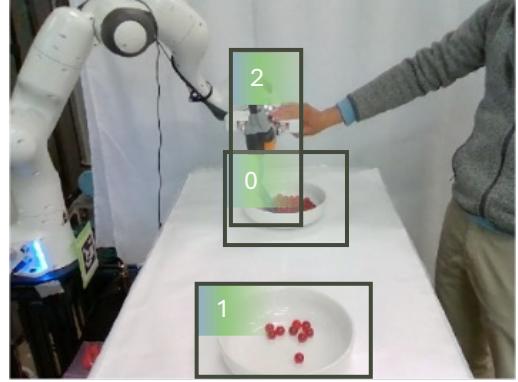
move(*tool*, *to*)

move(*tool*)

 effects: **marble-update**(*from*)
 marble-update(*to*)

Target 3: Transition models.

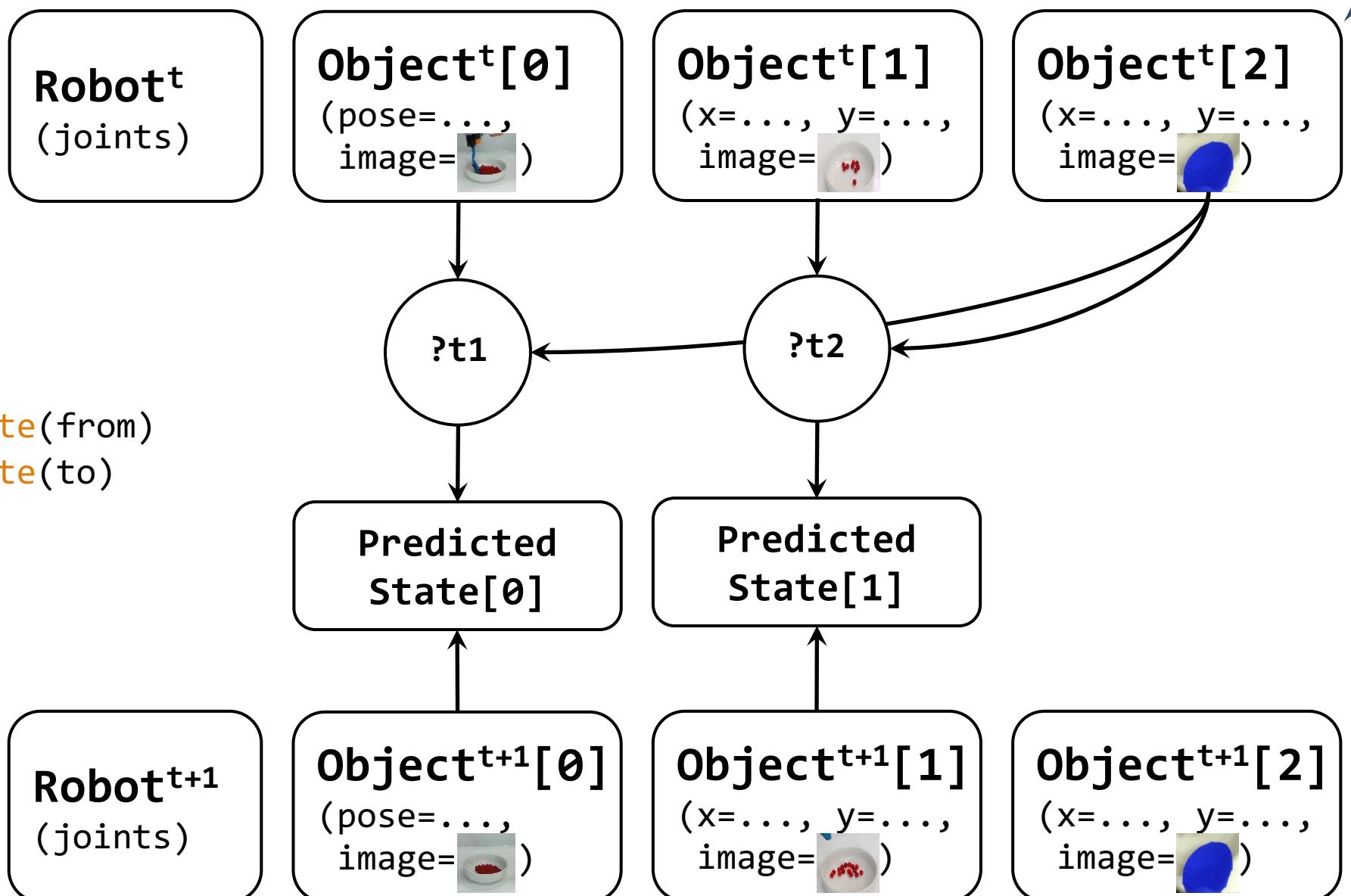
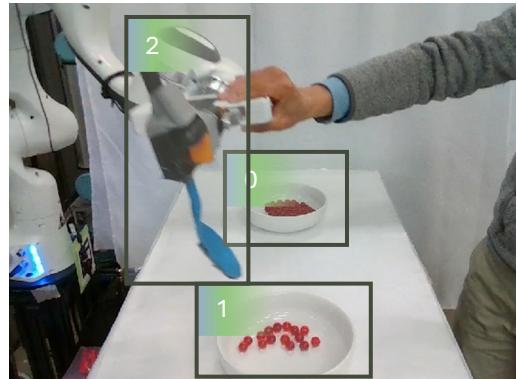
Learning Transitions with Self-Supervision



(Before)

effects: marble-update(from)
marble-update(to)

(After)



The Objective of Learning



→ ... →



Training Data: Trajectories (e.g., demonstrations)

```
action scoop( from, to, tool):  
    precondition: holding(tool), empty(tool)  
                  contains-marble(from)
```

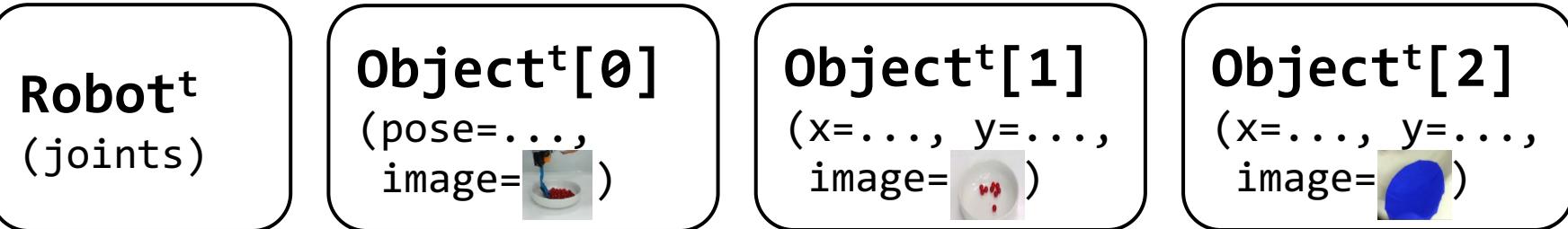
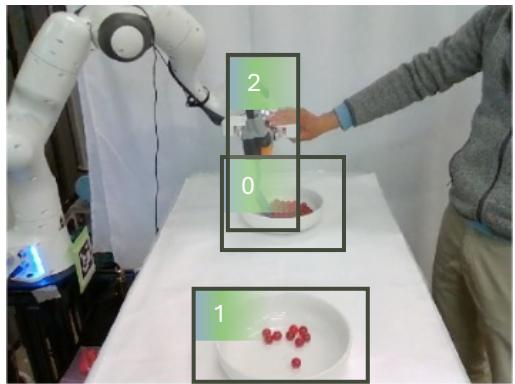
body:

```
    move(tool, from)  
    move-with-contact(tool, from)  
    move(tool, to)  
    move(tool, to)
```

```
effects: marble-update(from)  
        marble-update(to)
```

Target 2: Controllers for sub-actions.

Learning Continuous Parameters or Controllers



action `scoop(from, to, tool)`:

body:

```
# move to the bowl to scoop from  
move(tool, from)
```

...

A simple implementation can be done with segmented trajectories, but we can also **jointly learn to segment them**.

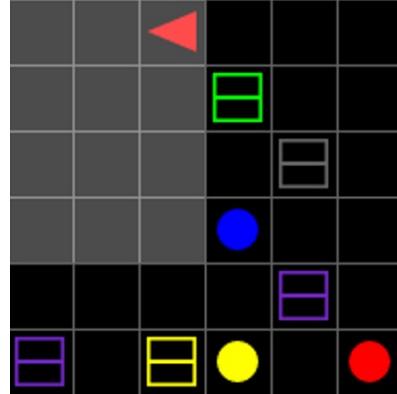
Option 1: Directly output a joint command.

- +: Most general. Does not rely on any prior knowledge.
- : Poor generalization for unseen configurations and obstacles.

Option 2: Output a target relative pose, and then call a motion planner.

- : Need additional knowledge.
- +: Better generalization for unseen configurations and obstacles.

Learning and Planning Efficiency



PDS-Rob

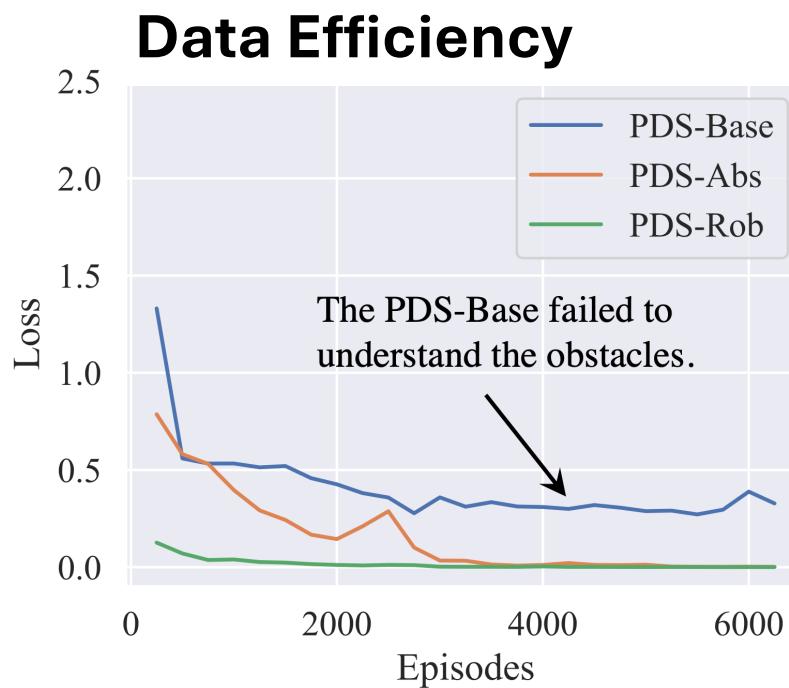
Full robot movement models.
Need to learn object classifiers.

PDS-Abs

Abstract robot models.
(With ??)

PDS-Base

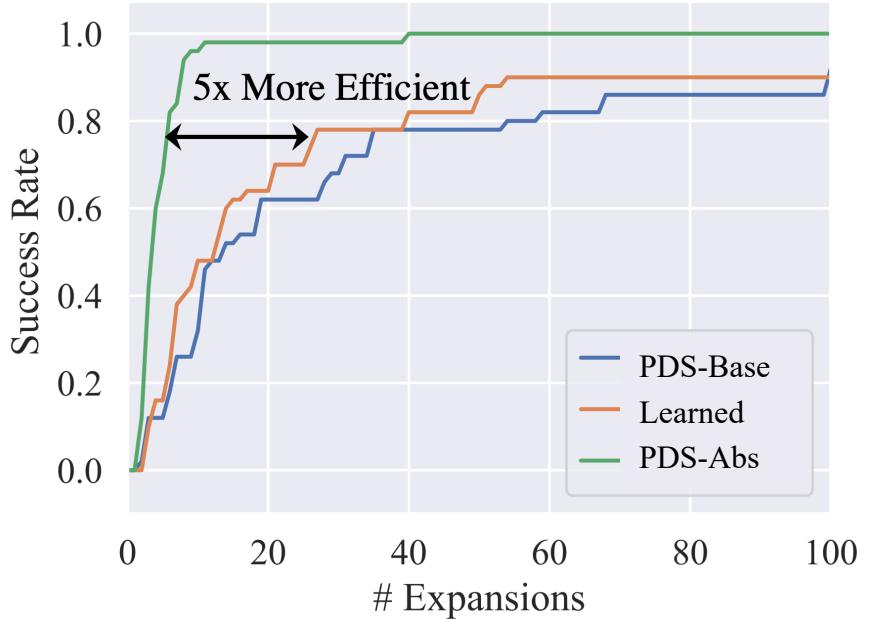
GNNs.
(Weakest prior)



Success Rate

Behavior Cloning	0.79
Decision Xformer	0.82
DreamerV2	0.79
PDS-Base	0.62
PDS-Abs	0.98
PDS-Rob	1.00

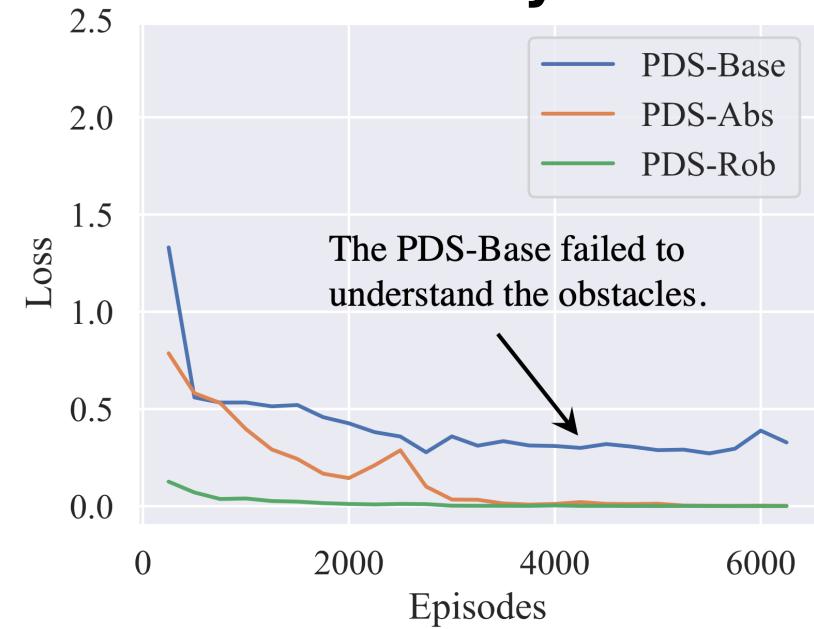
Planning Efficiency



Learning and Planning Efficiency

PDS-Abs
Abstract robot models.
(With Structures)

Data Efficiency



Success Rate

Very small amount of prior knowledge significantly improves the *data efficiency*.

DreamerV2

PDS-Base

PDS-Abs

PDS-Rob

Planning Efficiency



Learning and Planning Efficiency

PDS-Abs
Abstract robot models.
(With Structures)

Data Efficiency



Success Rate

Behavior Cloning	0.79
Decision Xformer	0.82
DreamerV2	0.79
PDS-Base	0.62
PDS-Abs	0.98
PDS-Rob	1.00

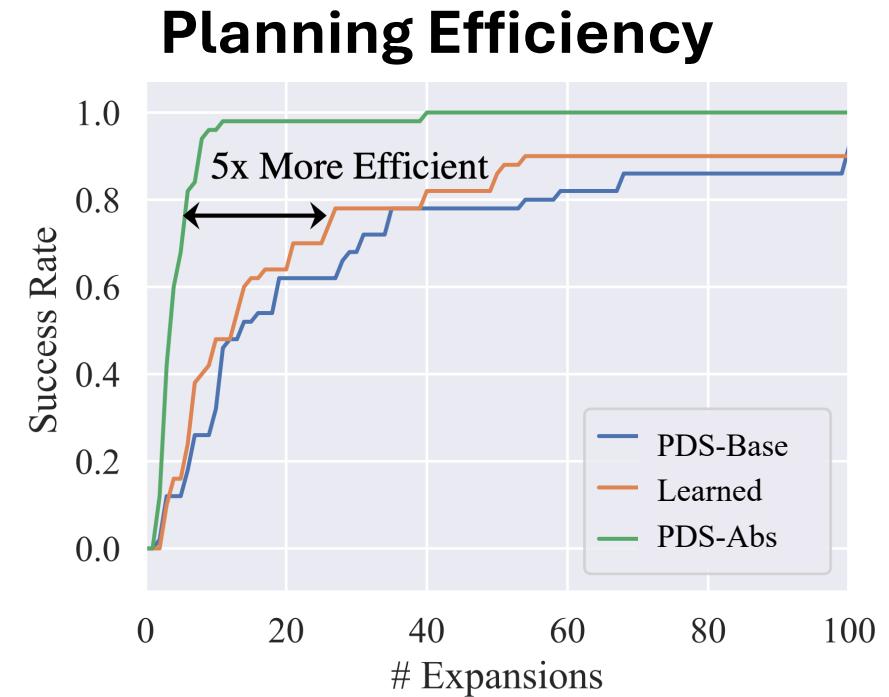
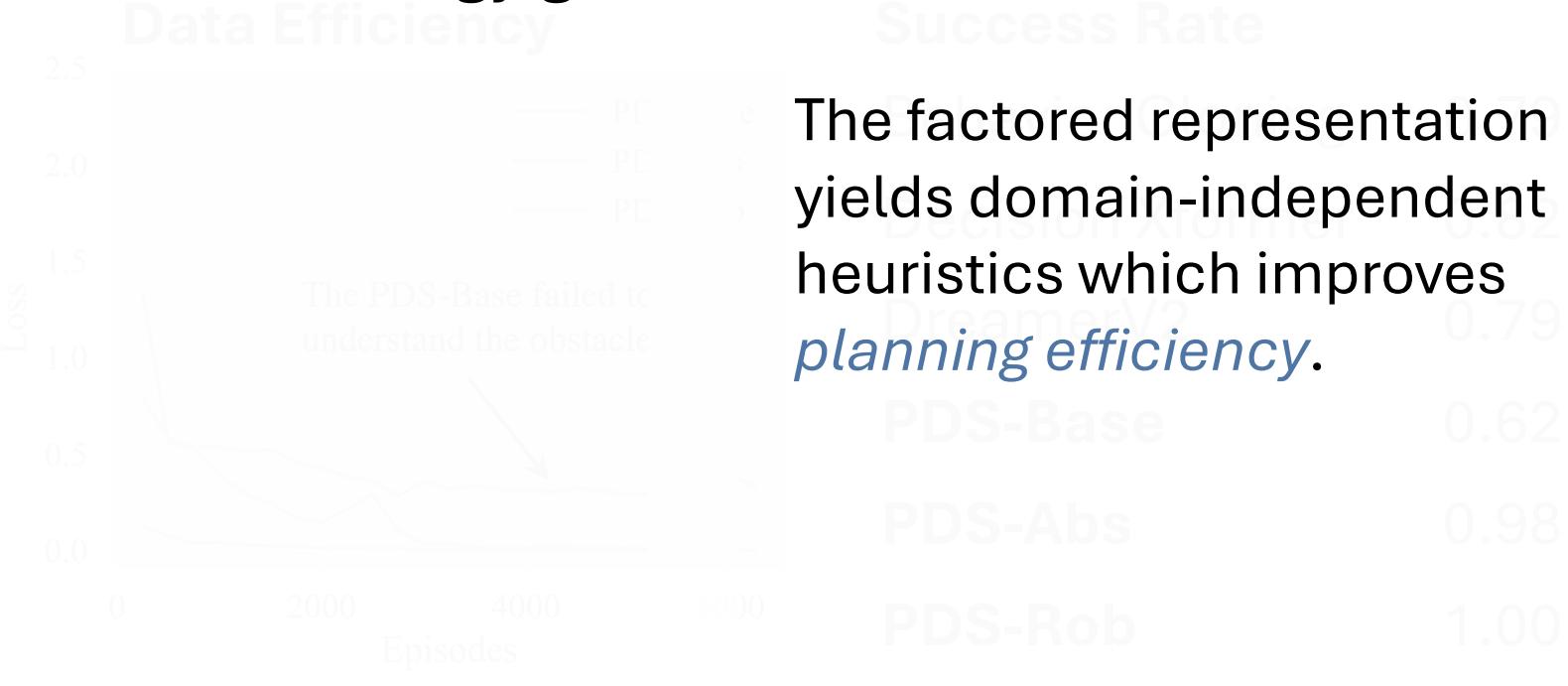
Planning Efficiency

The performance in model learning also translates to *better performance*.



Learning and Planning Efficiency

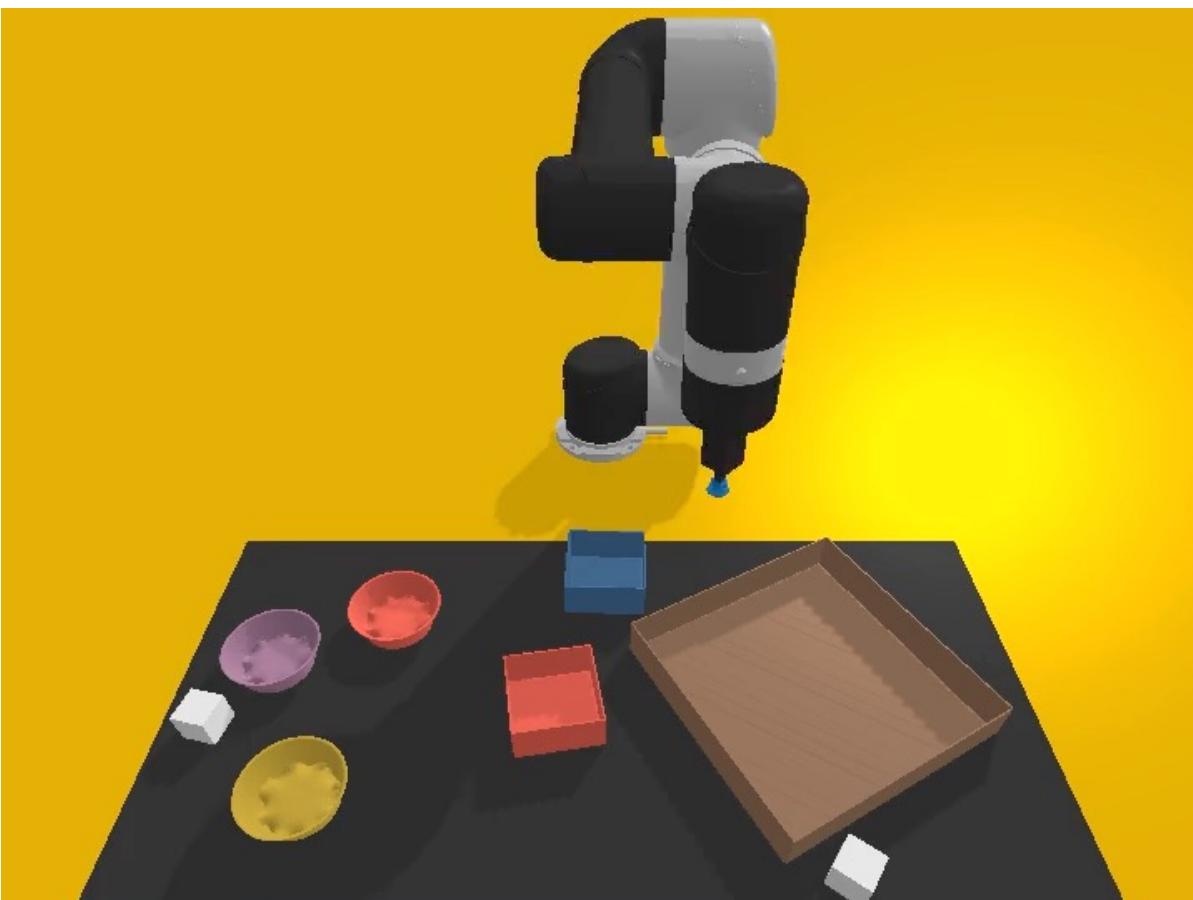
- Suppose an action has two preconditions.
- Solve two planning problems separately, and “add” the costs together.
- This usually gives a good estimate of the cost-to-go.
- Such strategy generalizes to structured neural models.



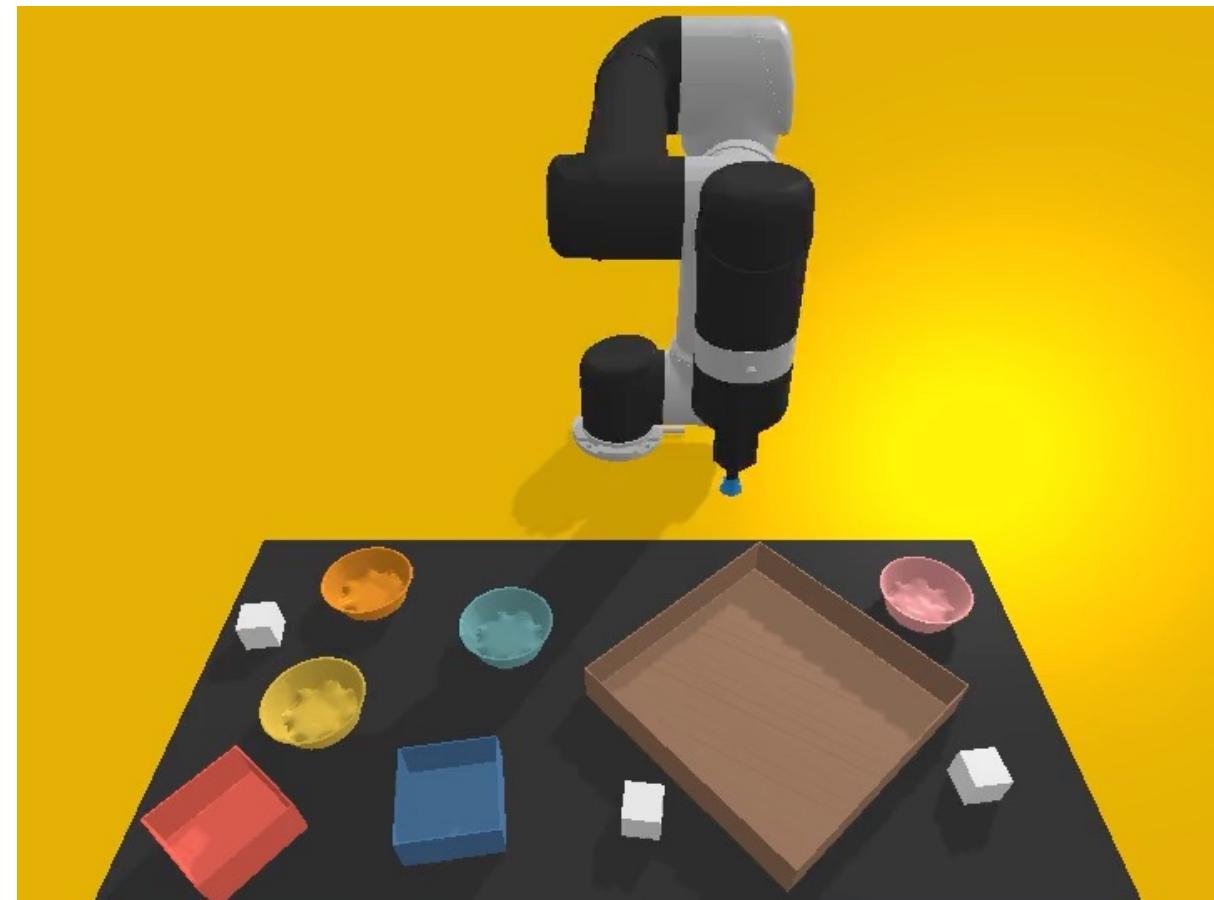
Generalization to Unseen States and Goals

Trained on goals: $\exists x.y. \text{color}(x) \& \text{color}(y) \& \text{rel}(x, y)$ Positions, number of objects, colors vary.

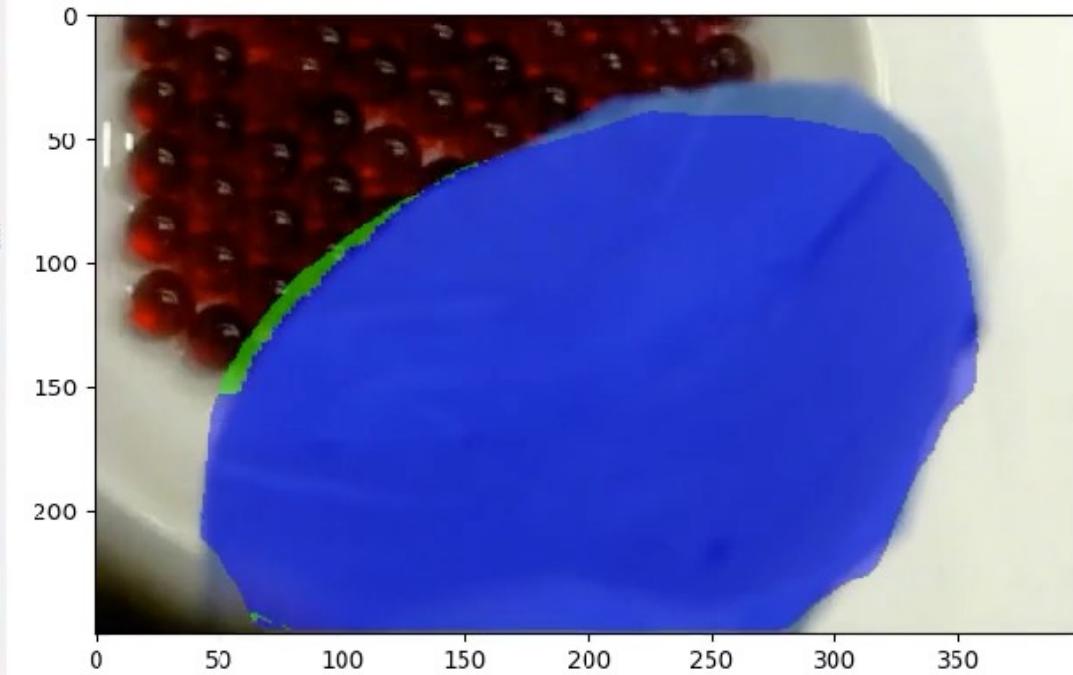
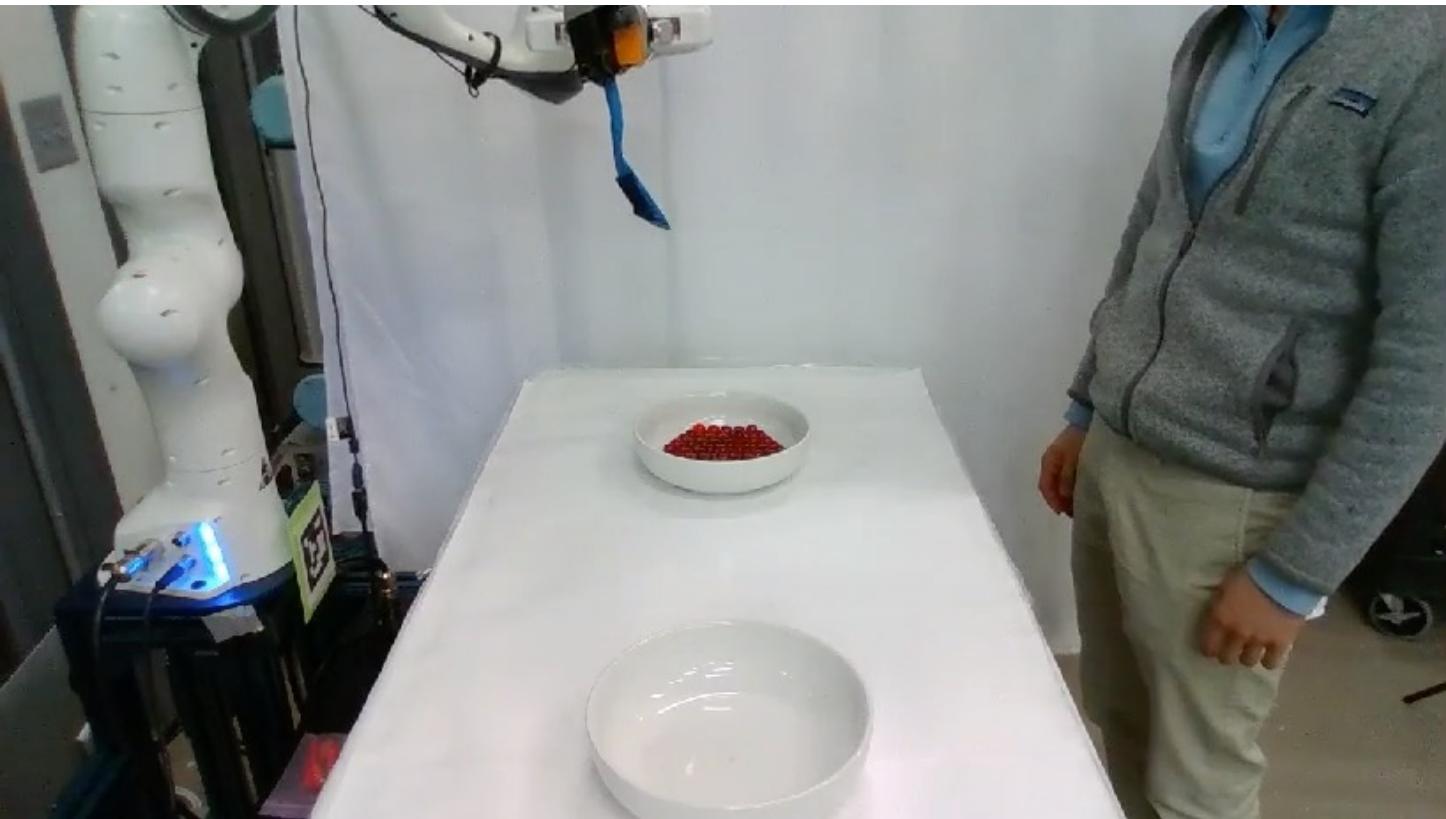
$\exists x.y. \text{purple}(x) \& \text{yellow}(y) \&$
 $\text{inbox}(x) \& \text{inbox}(y) \& \text{left-of}(x, y)$



$\forall x. \text{yellow}(x) \& \text{inbox}(x)$



Robust under Local and Global Perturbation

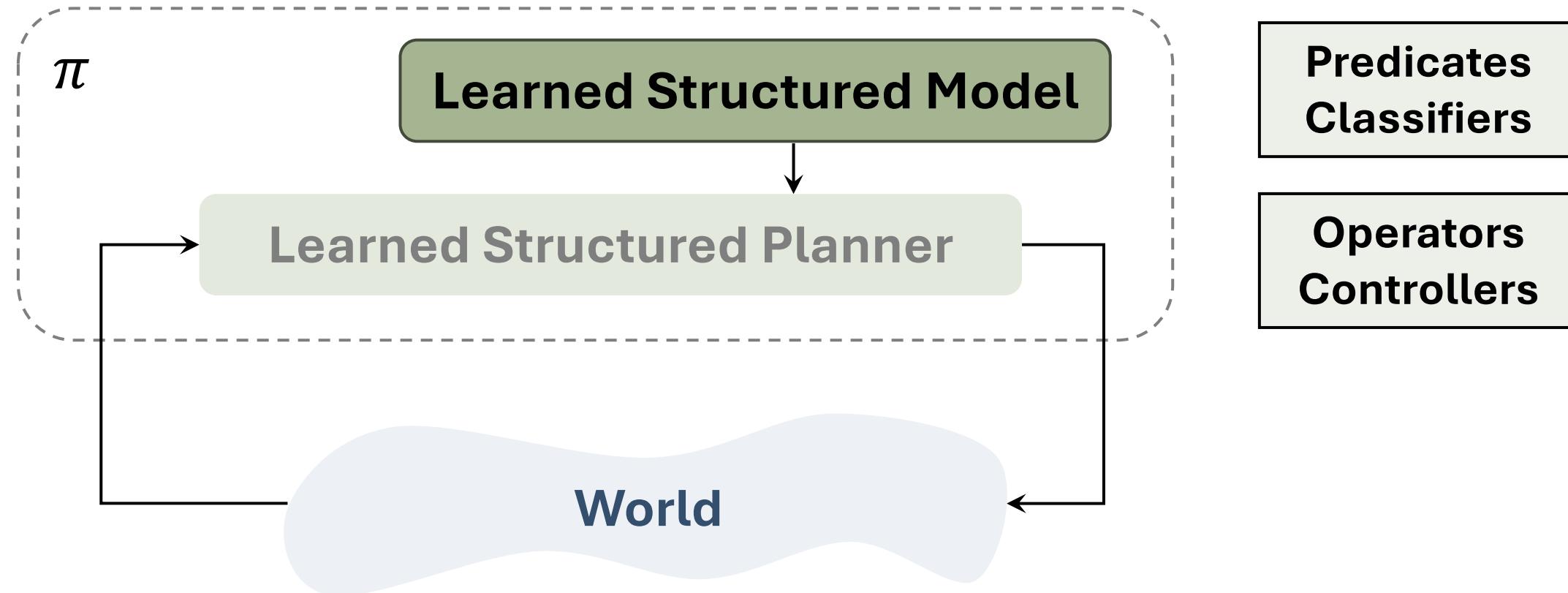


- Explicitly learned mode classifiers and transition rules enables online re-planning.
- Using motion planners enables generalization in “getting back to pre-scoop poses.”

* Trained with 17 human-collected demonstrations.

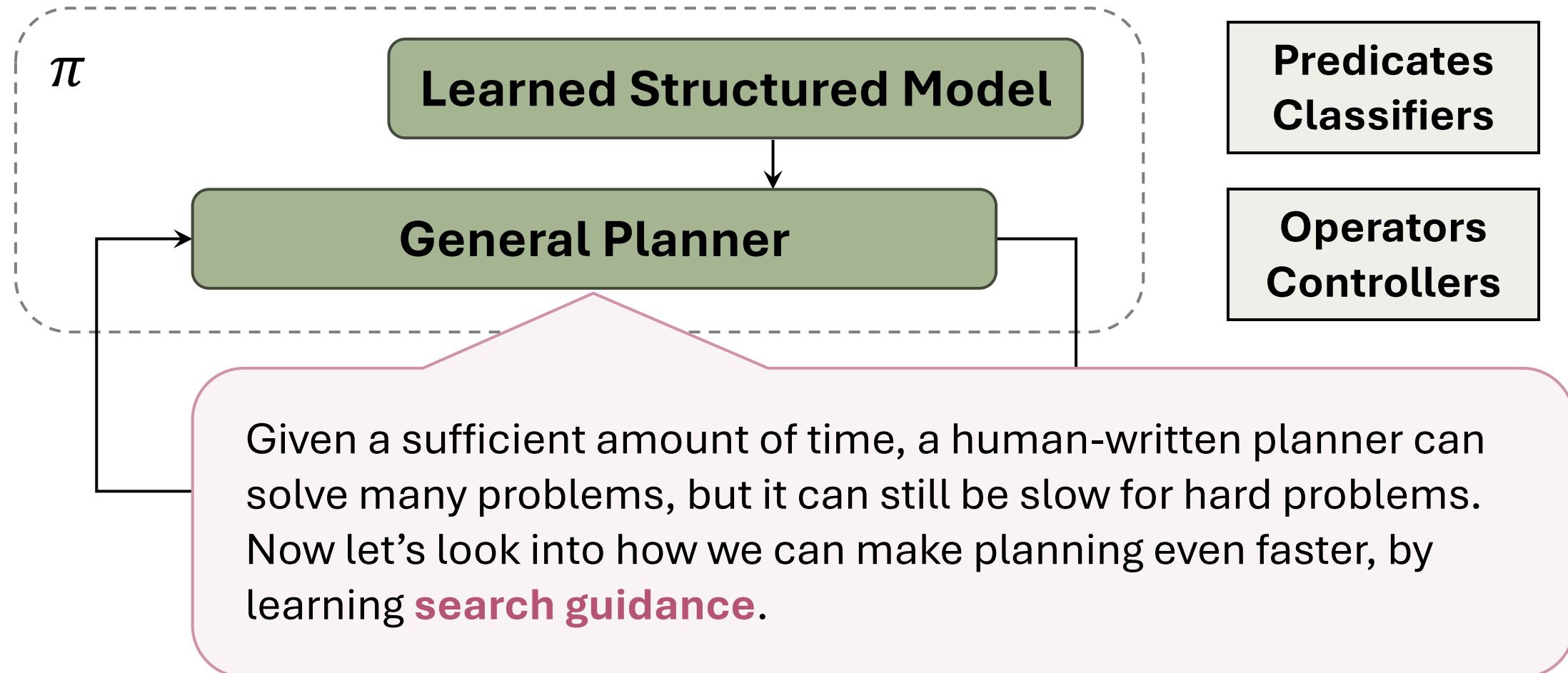
Grounding Language Plans in Demonstrations through Counter-factual Perturbations. Wang, Wang, Mao, Hagenow, Shah. 2024.

Learning Structured Representations for Models

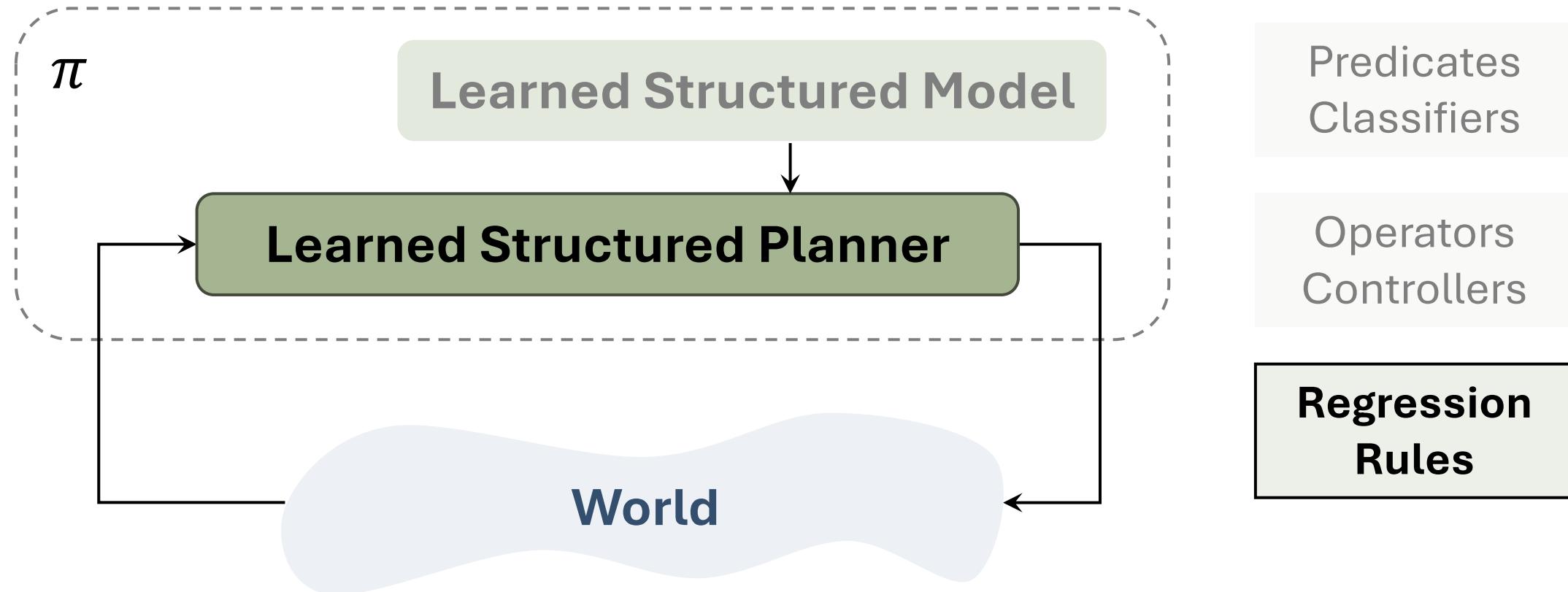


Factorization and sparsity structures improves learning and planning efficiency.
Temporal structures supports generalization to unseen goals and states.

Learning Structured Representations for Models



Learning Structured Representations for Planners



What Can We Learn from One Demonstration?



What Can We Learn from One Demonstration?

A “**strategy**” for picking up the cylinder.

- Push to rotate.
- Exert force on one end so that it tilts.
- Move the bucket.

You might not be able to execute it robustly now, but you have some “**ideas**.”

We aim to learn such “strategies” from a single demonstration and apply them compositionally.

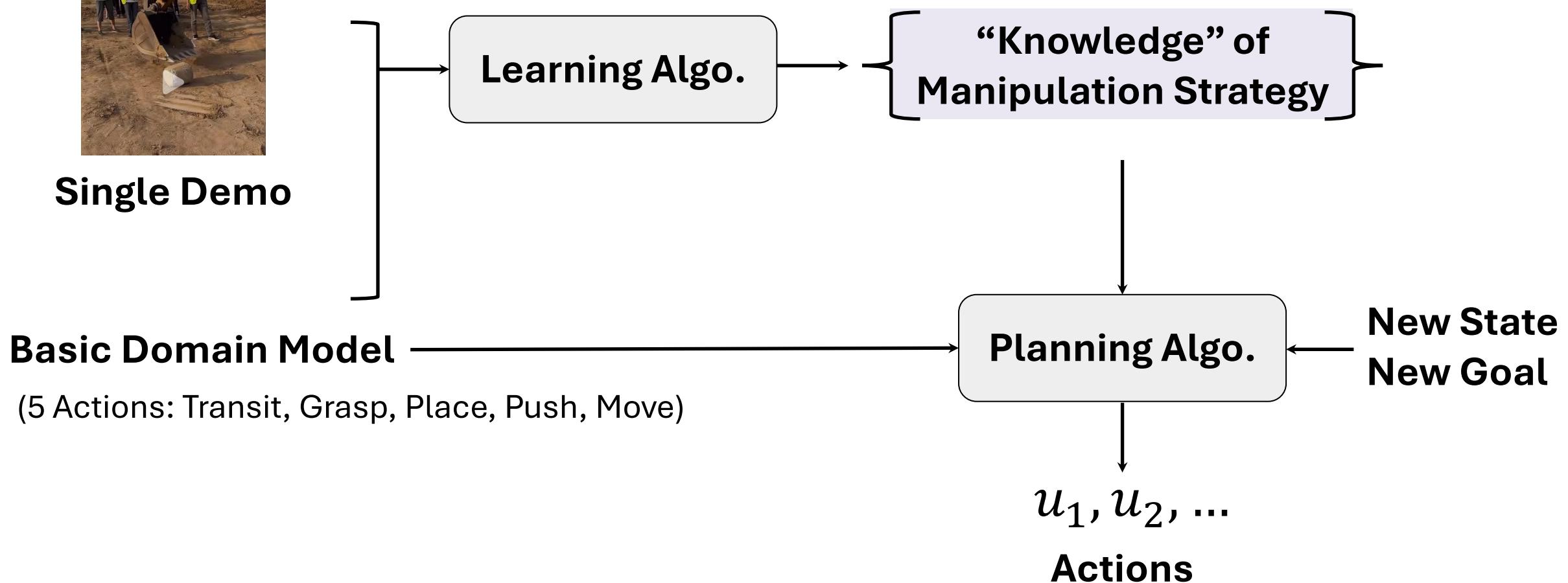


Problem Formulation

We have a basic model for object manipulation & ***one demonstration***.



What can we learn from the demonstration?



What Can We Learn from One Demonstration?

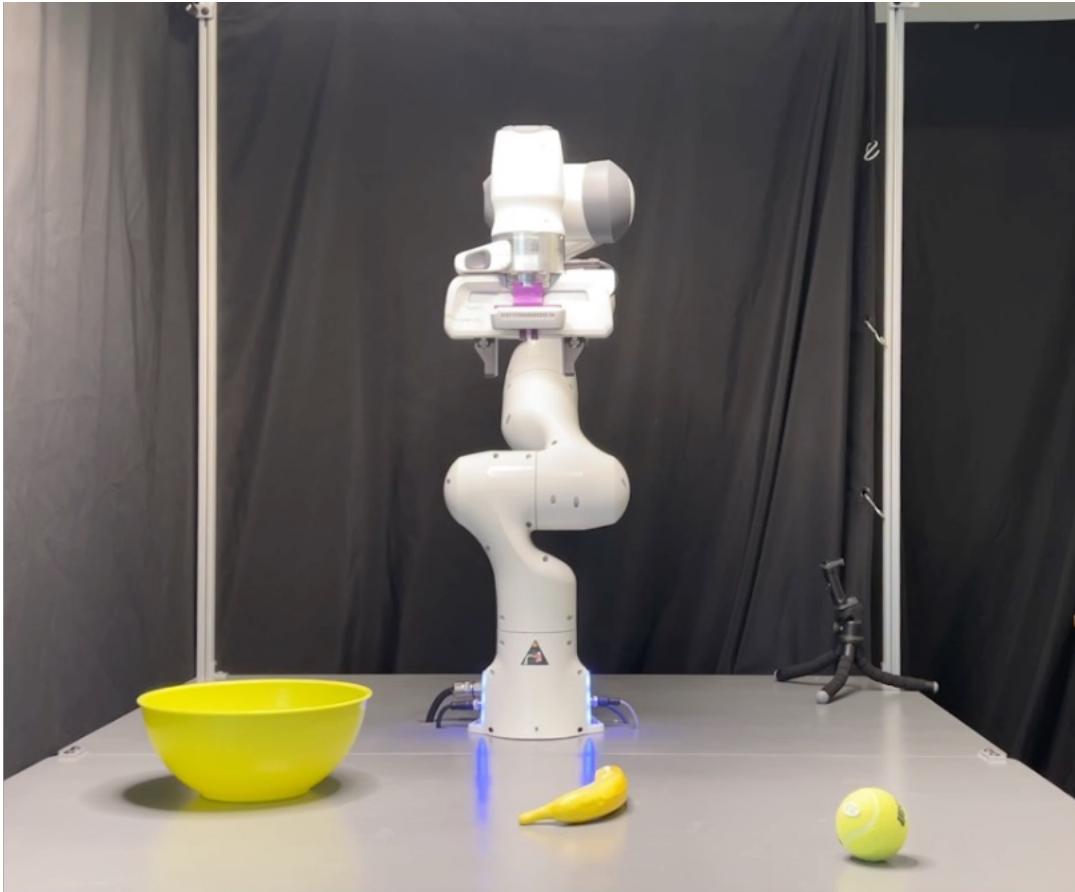
Key idea: some manipulation “strategies” can be modeled by a sequence of subgoals about contacts among objects.

Let’s talk about a familiar example: hook-using.



The Contact Mode Subgoals in Hook-Using

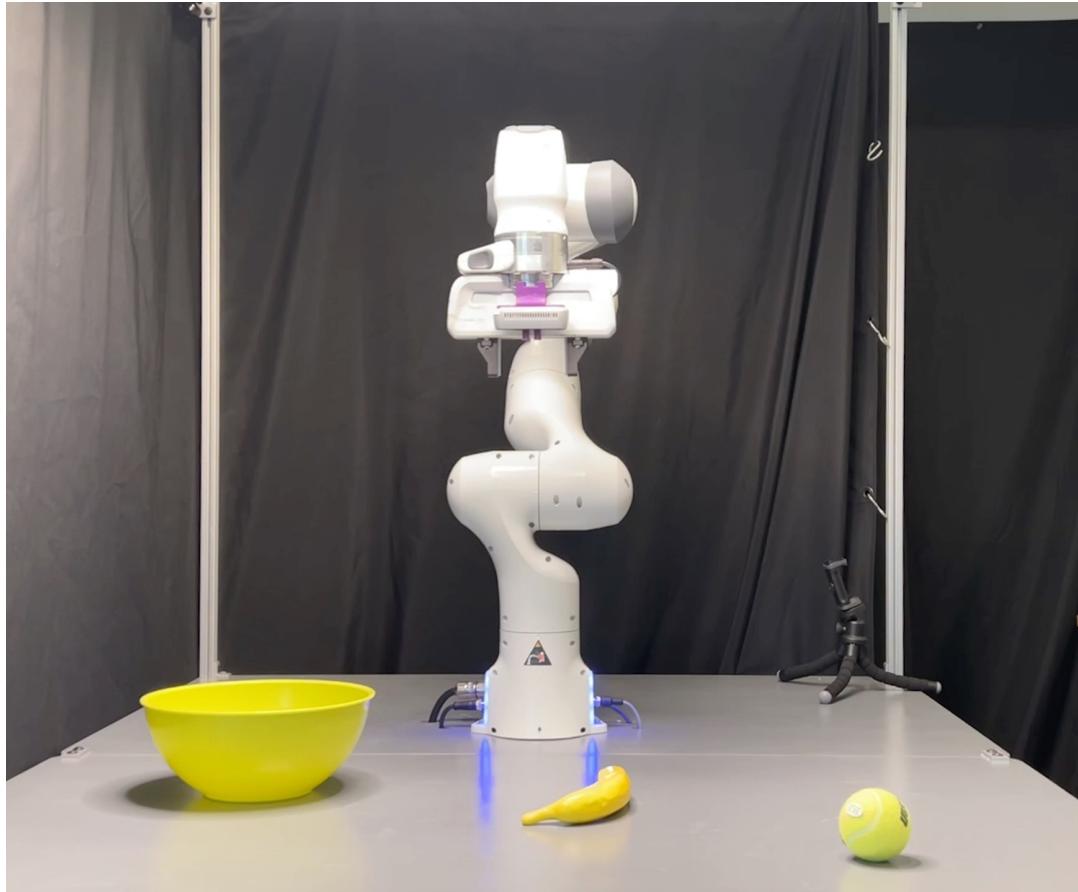
Key idea: some manipulation “strategies” can be modeled by a sequence of subgoals about contacts among objects.



```
rule hook(target, tool, support):  
    goal: holding(target)  
    precondition: on(target, support)  
                on(tool, support)  
    body:  
        grasp(tool, ?pose, ?traj)  
        move-with-contact(tool, target, ?traj)  
        place(tool, support, ?pose, ?traj)  
        grasp(target, ?pose, ?traj)
```

The Contact Mode Subgoals in Hook-Using

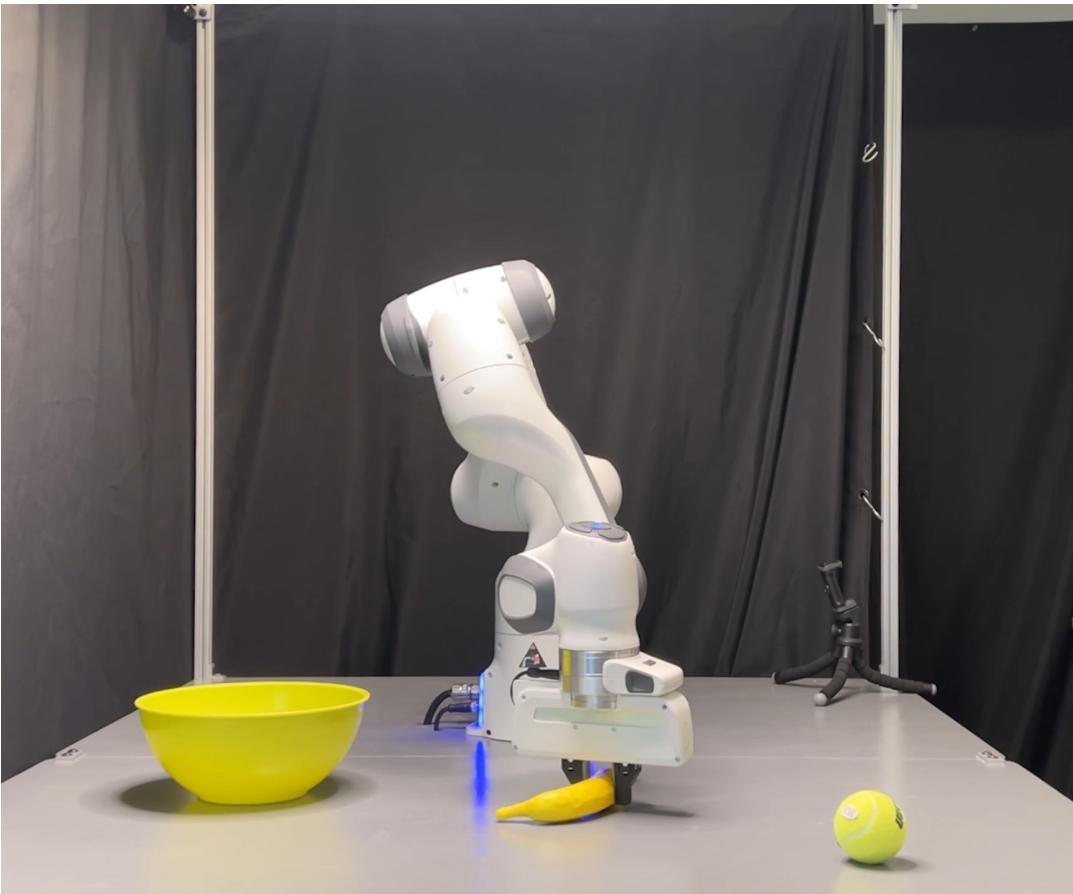
Key idea: some manipulation “strategies” can be modeled by a sequence of subgoals about contacts among objects.



```
rule hook(target, tool, support):  
    goal: holding(target)  
    precondition: on(target, support)  
                on(tool, support)  
    body:  
        grasp(tool, ?pose, ?traj)  
        move-with-contact(tool, target, ?traj)  
        place(tool, support, ?pose, ?traj)  
        grasp(target, ?pose, ?traj)
```

The Contact Mode Subgoals in Hook-Using

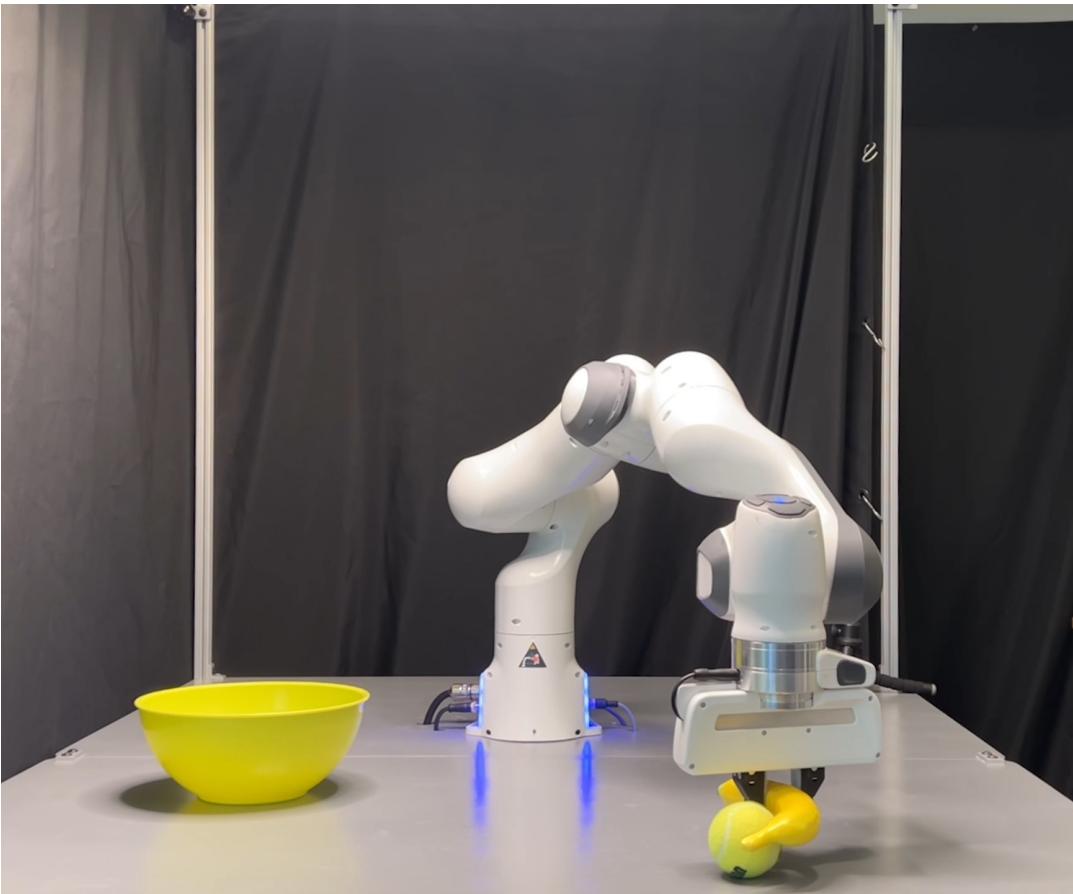
Key idea: some manipulation “strategies” can be modeled by a sequence of subgoals about contacts among objects.



```
rule hook(target, tool, support):  
    goal: holding(target)  
    precondition: on(target, support)  
                on(tool, support)  
    body:  
        grasp(tool, ?pose, ?traj)  
        move-with-contact(tool, target, ?traj)  
        place(tool, support, ?pose, ?traj)  
        grasp(target, ?pose, ?traj)
```

The Contact Mode Subgoals in Hook-Using

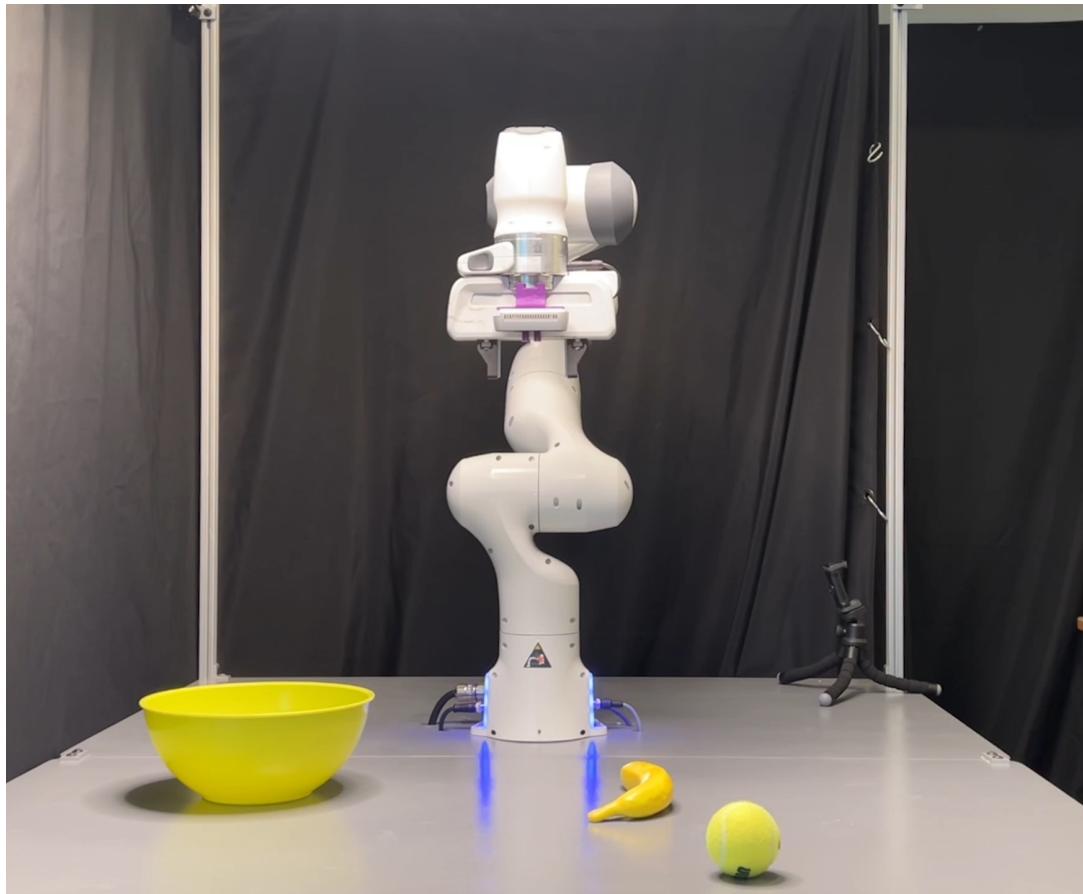
Key idea: some manipulation “strategies” can be modeled by a sequence of subgoals about contacts among objects.



```
rule hook(target, tool, support):  
    goal: holding(target)  
    precondition: on(target, support)  
                on(tool, support)  
    body:  
        grasp(tool, ?pose, ?traj)  
        move-with-contact(tool, target, ?traj)  
        place(tool, support, ?pose, ?traj)  
        grasp(target, ?pose, ?traj)
```

The Contact Mode Subgoals in Hook-Using

Key idea: some manipulation “strategies” can be modeled by a sequence of subgoals about contacts among objects.

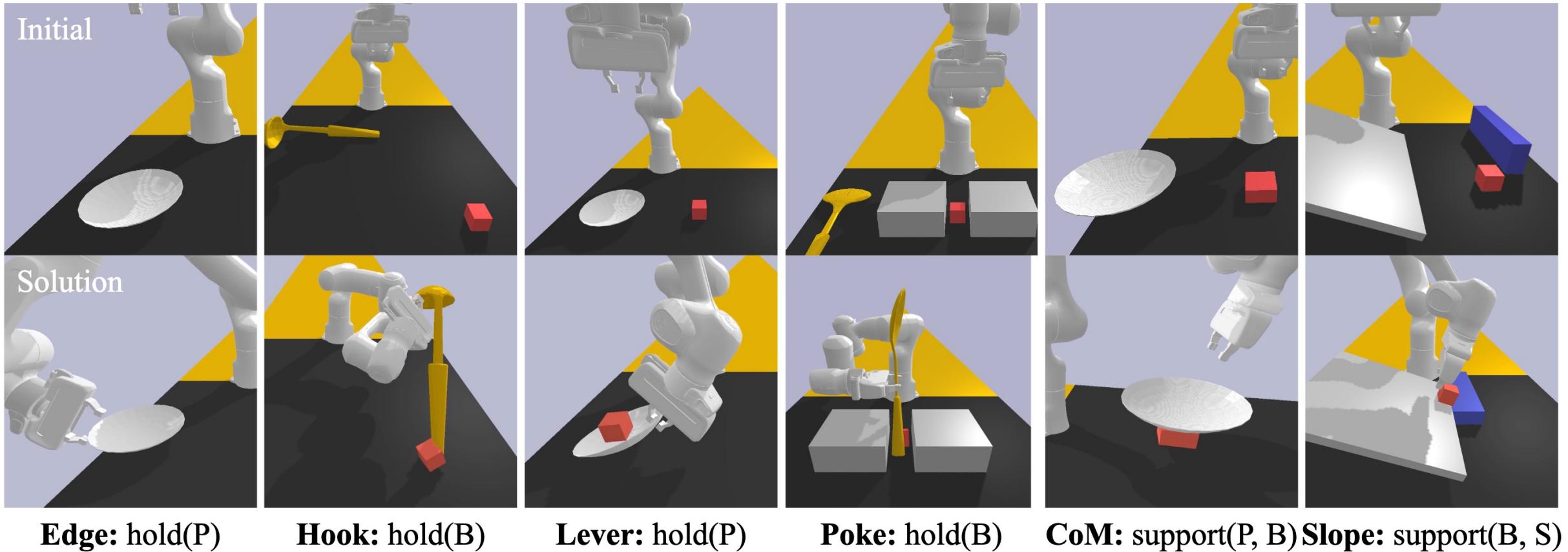


```
rule hook(target, tool, support):  
    goal: holding(target)  
    precondition: on(target, support)  
                on(tool, support)  
  
    body:  
        grasp(tool, ?pose, ?traj)  
        move-with-contact(tool, target, ?traj)  
        place(tool, support, ?pose, ?traj)  
        grasp(target, ?pose, ?traj)
```

Previously we were learning causal models of actions and plans with them. Now we can memorize “partial solutions” as shortcuts.

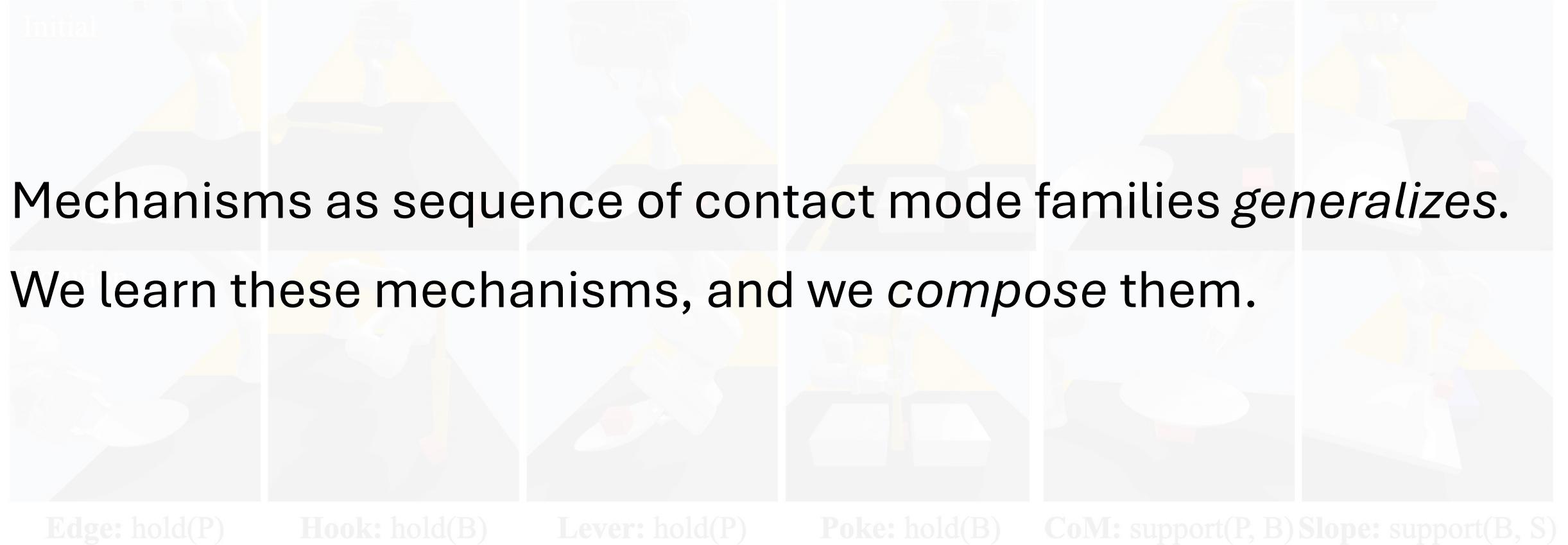
Many Strategies Can Be Represented This Way

We call these manipulation strategies “*mechanisms.*”



Many Strategies Can Be Represented This Way

We call these manipulation strategies “***mechanisms.***”



Overview of the Framework

There are two **learning problems**:

1. Learning of the contact mode sequence.
2. Learning samplers for parameters of the contact modes: where to grasp, how to move, etc.

Overview of the Framework

There are two **learning problems**:

1. Learning of the contact mode sequence.

We will recover it from the single demonstration.

2. Learning samplers for parameters of the contact modes: where to grasp, how to move, etc.



```
(:macro hook  
:parameters (  
  ?tool - item  
  ?target - item  
  ?support - item  
)  
:certified (  
  (holding ?target)  
)  
...)
```

Single Demo

Contact Modes and Goals

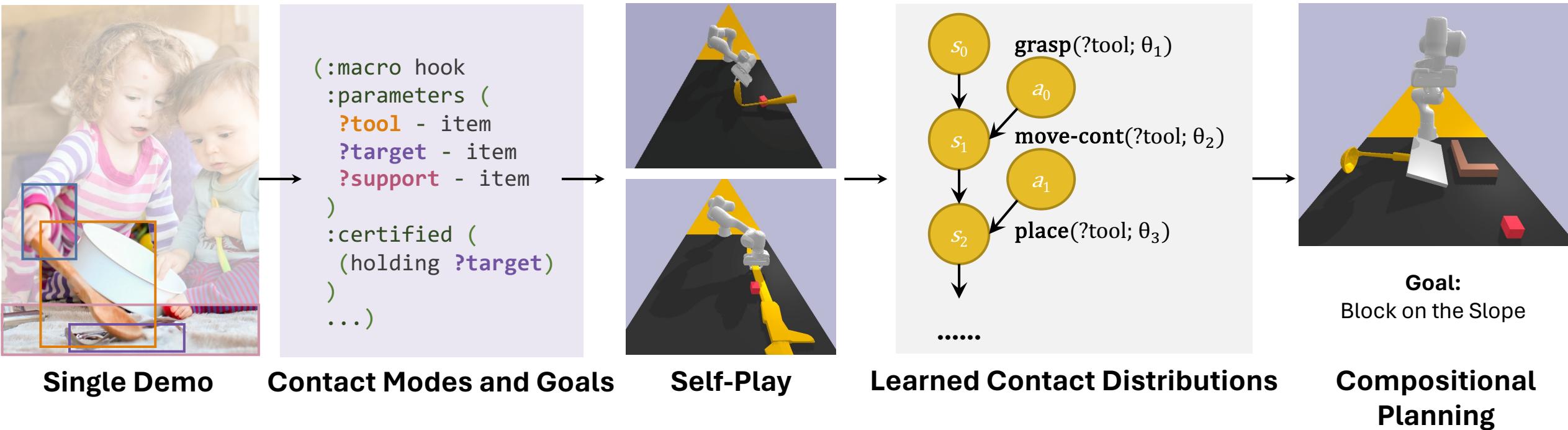
Overview of the Framework

There are two **learning problems**:

1. Learning of the contact mode sequence.

We will recover it from the single demonstration.

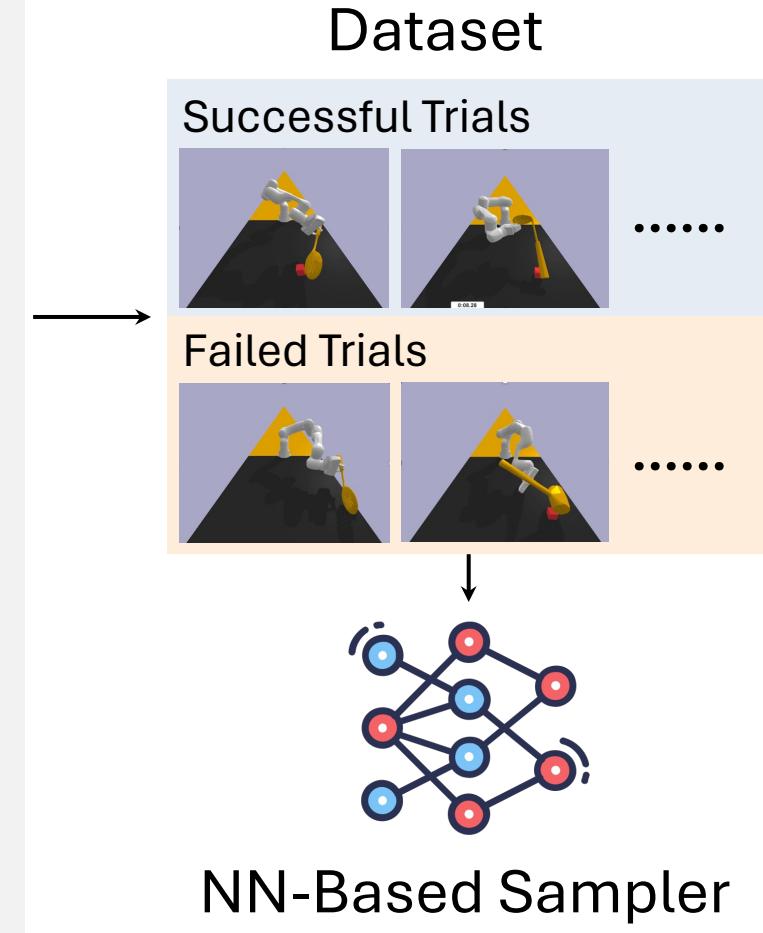
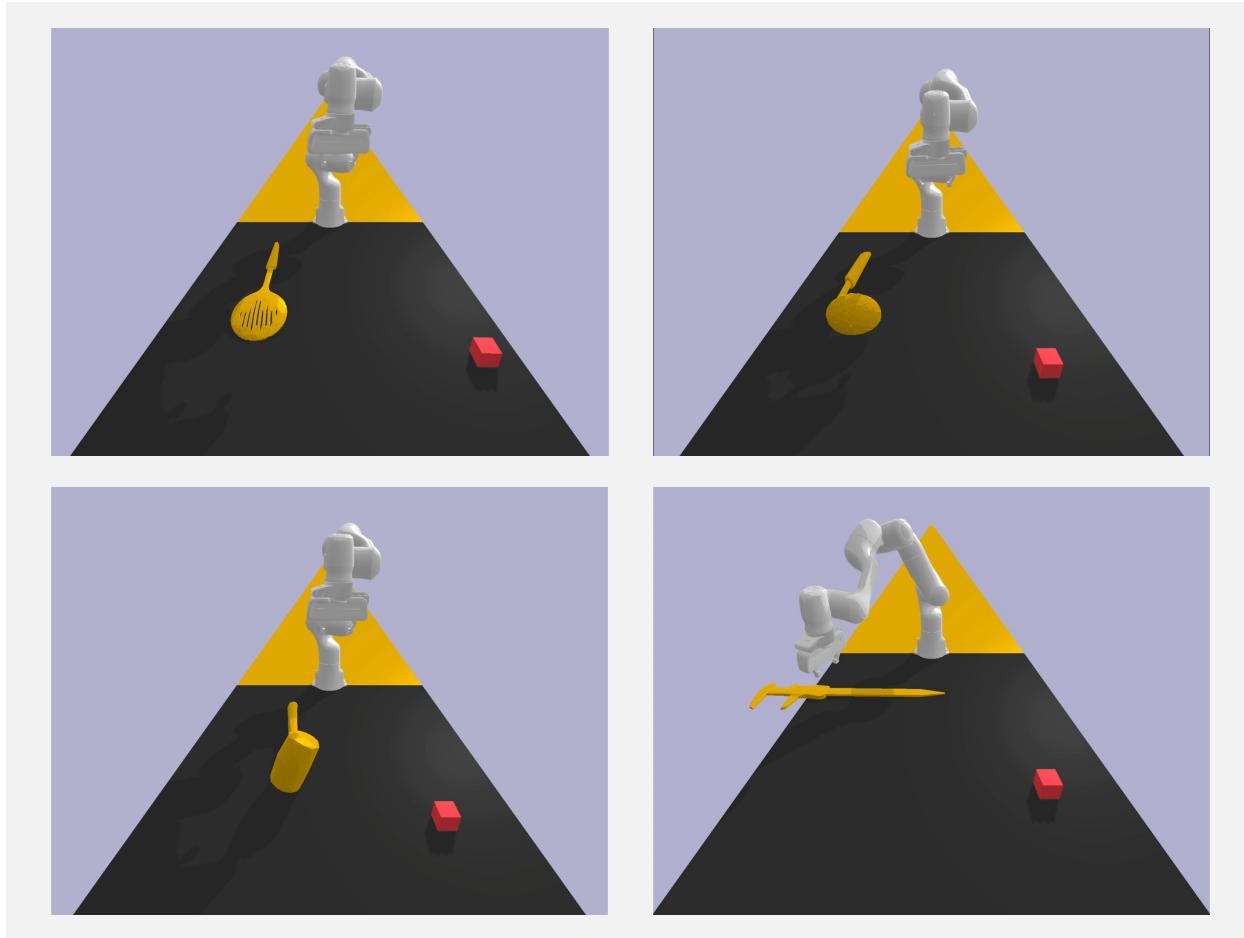
2. Learning samplers for parameters of the contact modes: where to grasp, how to move, etc.



Step 2: Learn Mechanism-Specific Samplers

We will learn those samplers (parameter generators) from self-plays.

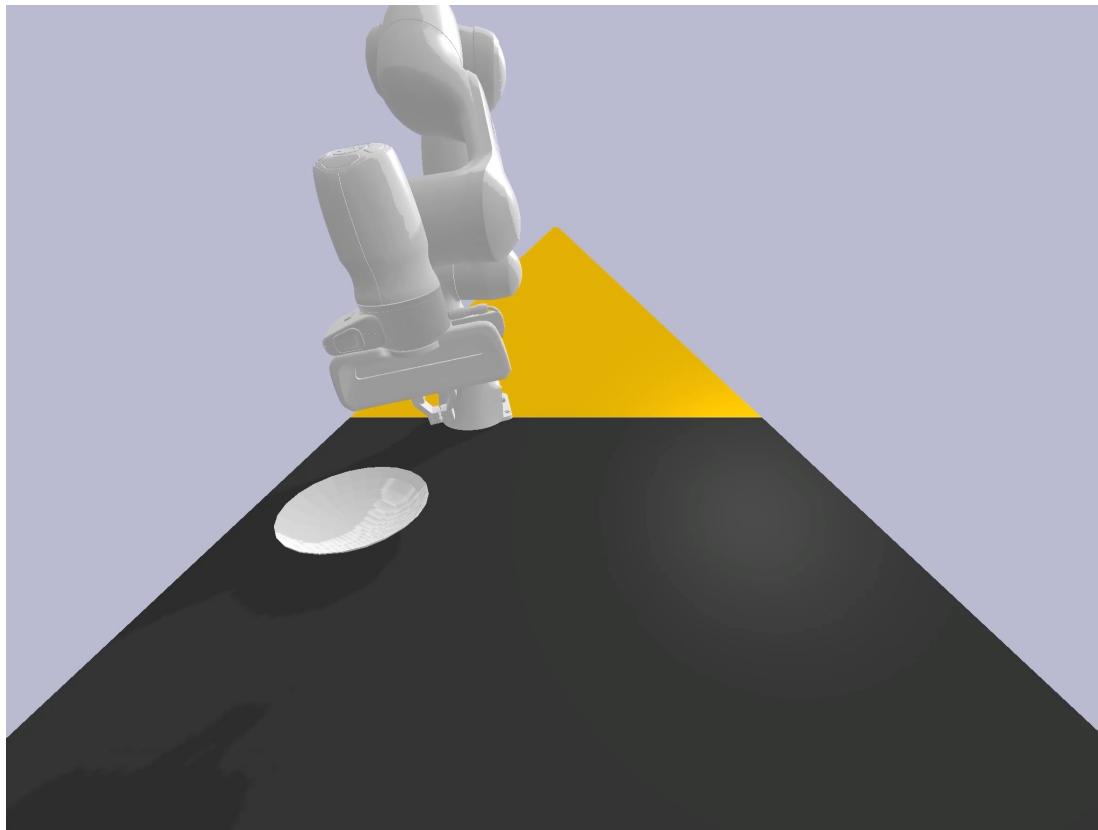
Contact
Modes and →
Goals



Learning Mechanisms Improves Efficiency

Method	Edge	Hook	Lever	Poking	CoM	Slope&Blocker
Basis Ops Only	89.45 ± 5.53	>600	523.18 ± 9.22	>600	19.30 ± 2.82	>600
Ours (Macro+Sampler)	0.57 ± 0.05	3.84 ± 1.56	1.55 ± 0.29	97.76 ± 10.67	0.97 ± 0.09	4.11 ± 0.94

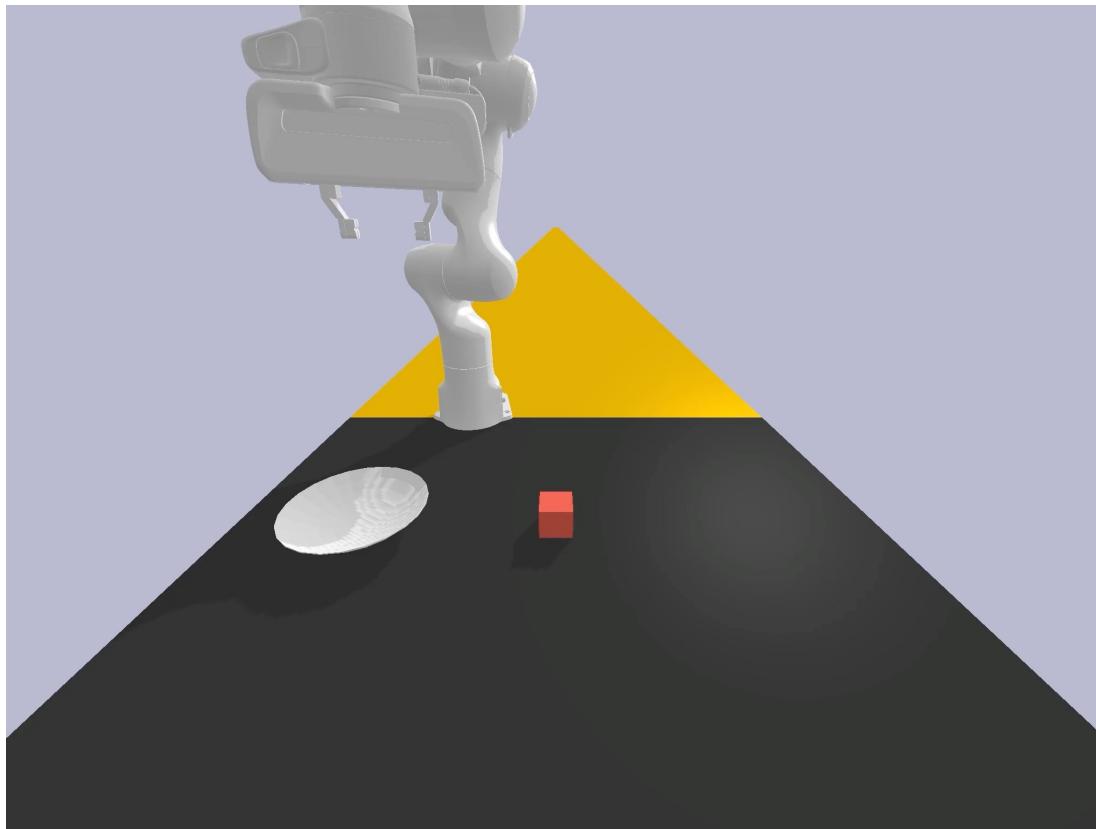
Learning Mechanisms Improves Planning Efficiency



Goal:
holding(plate)

Method	Edge	Hook	Lever	Poking	CoM	Slope&Blocker
Basis Ops Only	89.45 ± 5.53	>600	523.18 ± 9.22	>600	19.30 ± 2.82	>600
Ours (Macro+Sampler)	0.57 ± 0.05	3.84 ± 1.56	1.55 ± 0.29	97.76 ± 10.67	0.97 ± 0.09	4.11 ± 0.94

Learning Mechanisms Improves Planning Efficiency



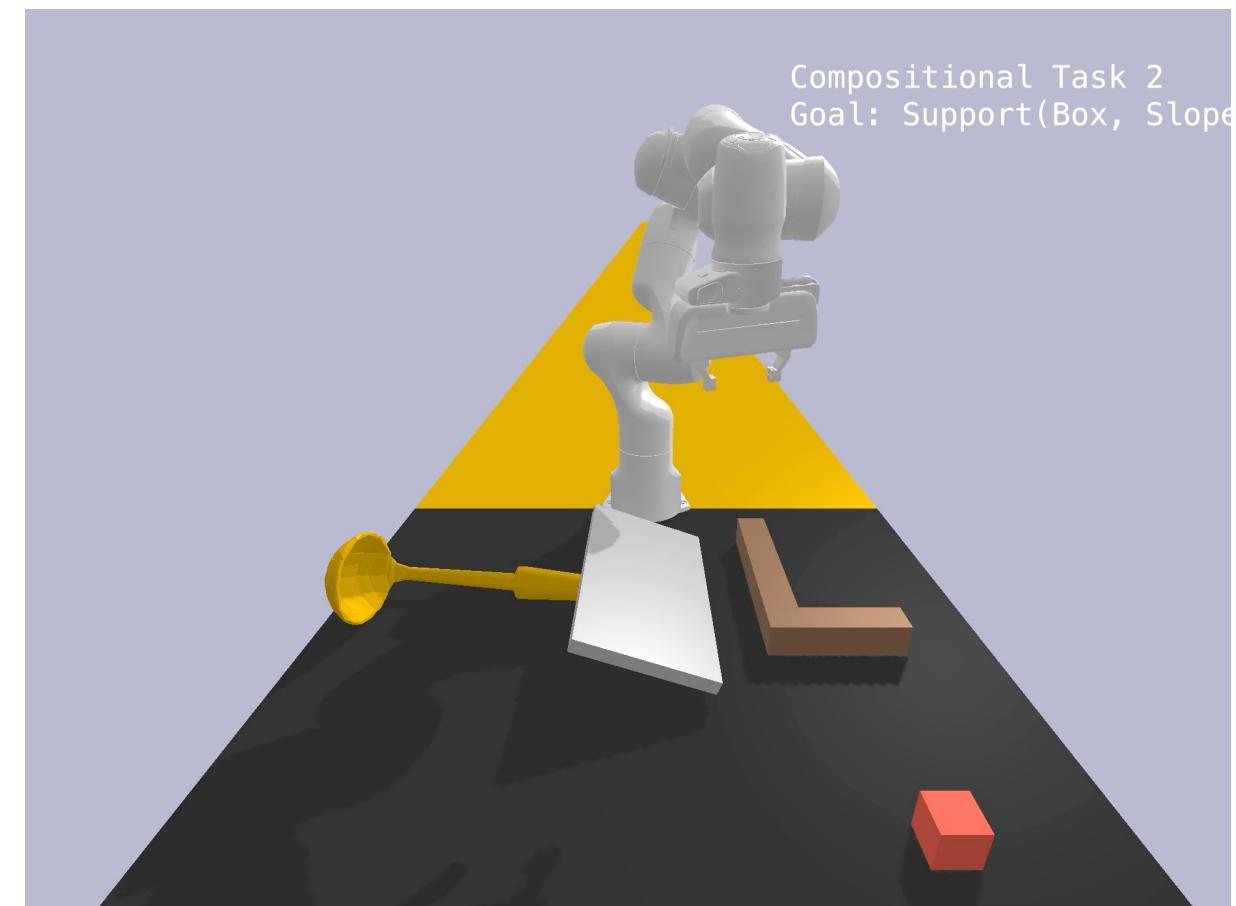
Goal:
holding(plate)

Method	Edge	Hook	Lever	Poking	CoM	Slope&Blocker
Basis Ops Only	89.45 ± 5.53	>600	523.18 ± 9.22	>600	19.30 ± 2.82	>600
Ours (Macro+Sampler)	0.57 ± 0.05	3.84 ± 1.56	1.55 ± 0.29	97.76 ± 10.67	0.97 ± 0.09	4.11 ± 0.94

Composing Mechanisms Automatically by Planning



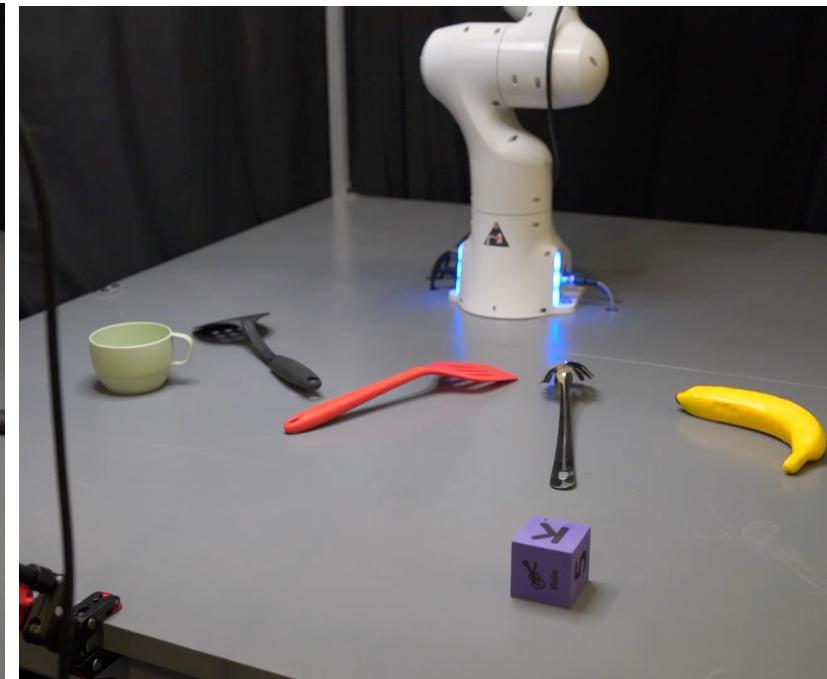
Goal: holding(box)
The caliper is too flat to be grasped.



Goal: on(box, ramp)
Box may slide down the ramp.

Real Robot Execution of the Learned Strategies

Goal: in(cube, cup)

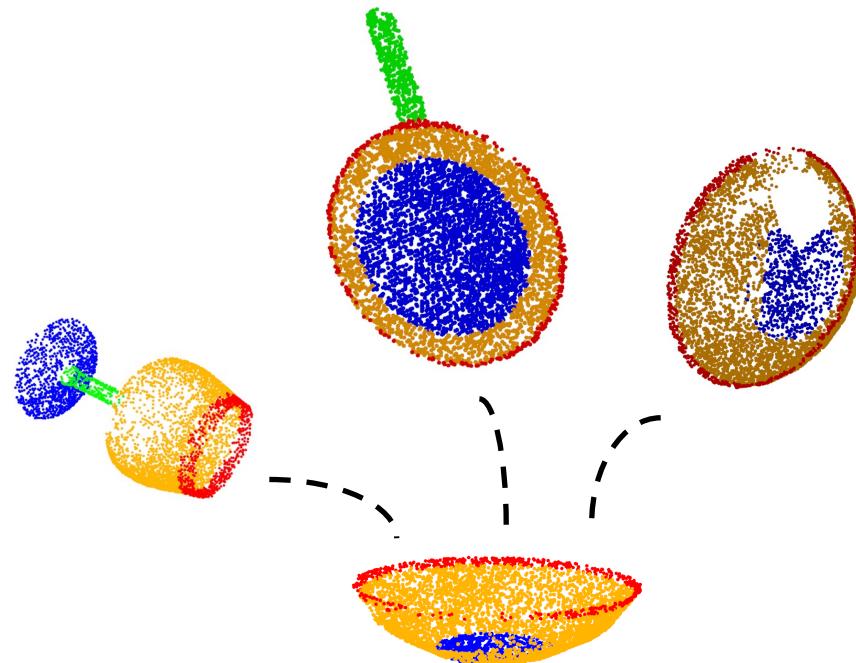


Generalization to new tools, with no 3D model required.
We apply our structured model and planner based on point cloud inputs.

Extension Beyond Rigid-Body Contacts

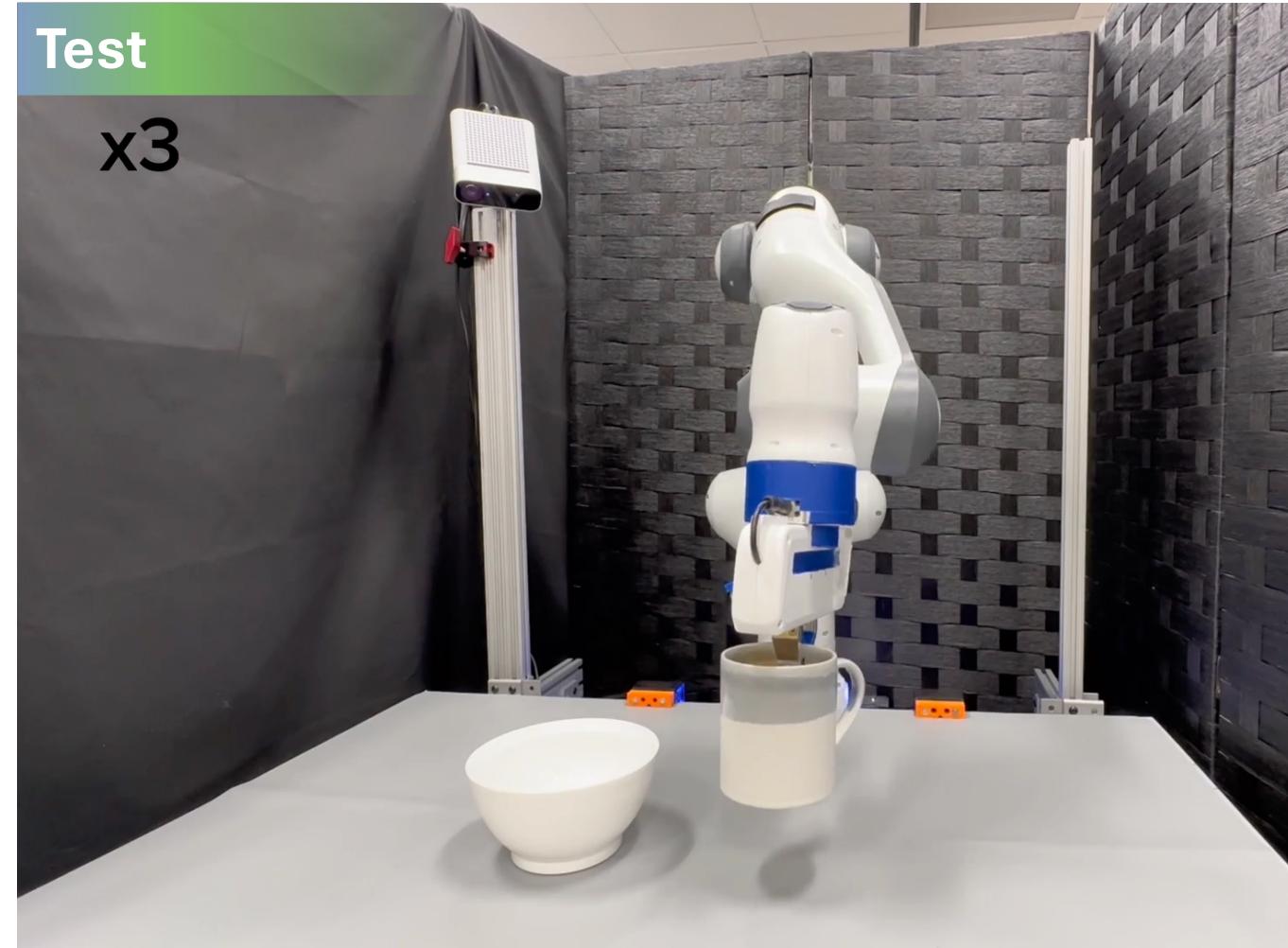
Trained on glasses, bowls, and frypans. Generalize to mugs.

Train



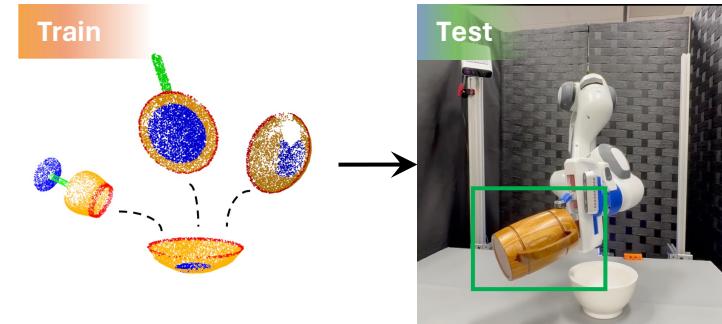
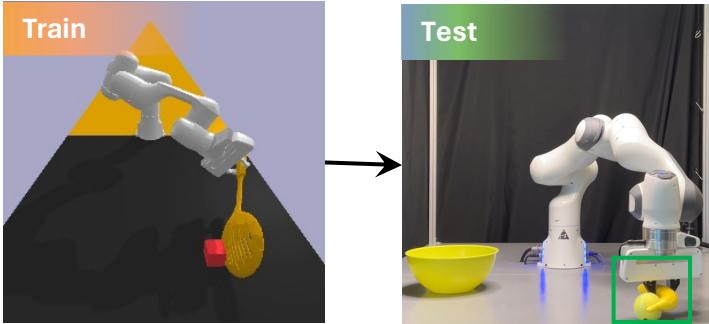
Test

x3



Compositional Abstractions Enable Generalization

Generalization to Novel Objects

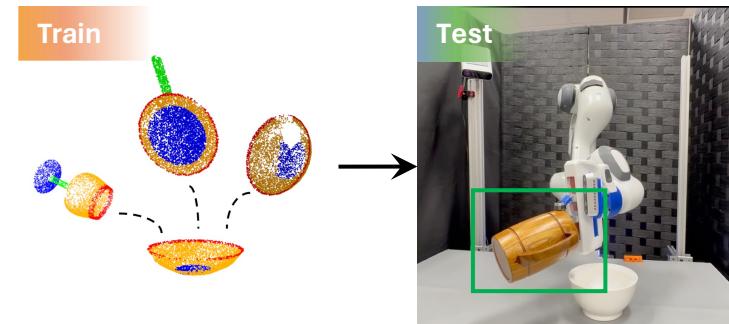
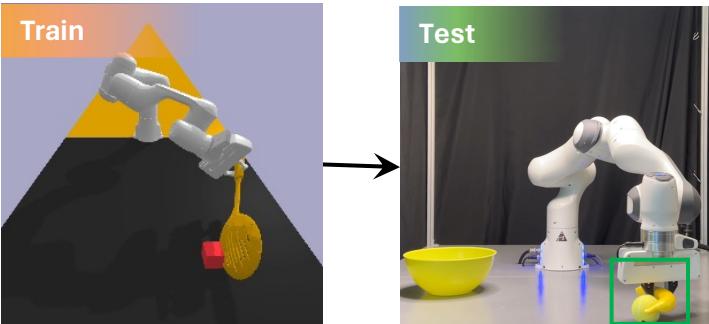


Generalization to Novel States



Compositional Abstractions Enable Generalization

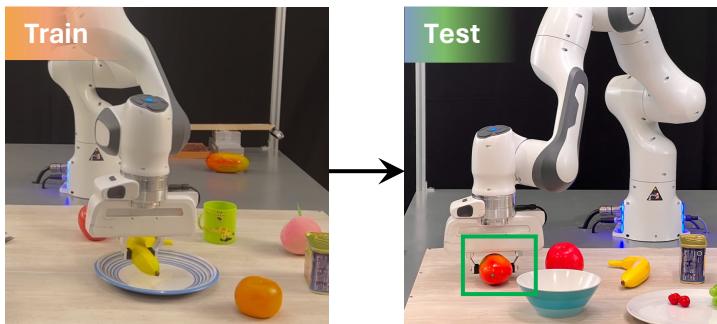
Generalization to Novel Objects



Generalization to Novel States



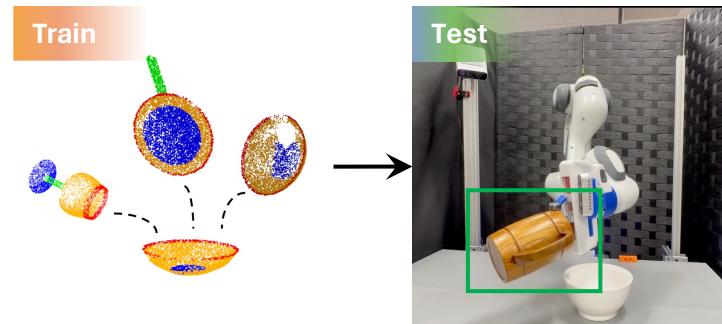
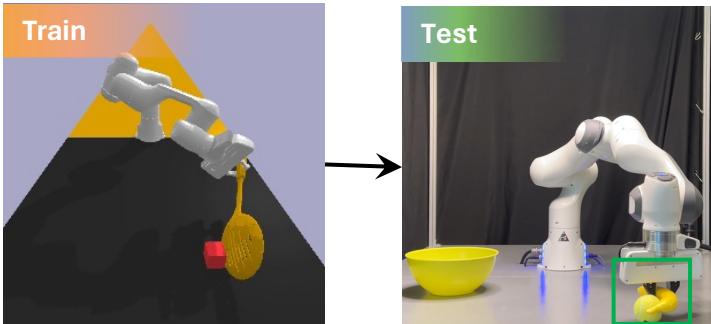
Generalization to Novel Words



By factorizing action controller learning and visual recognition of objects (using CLIP), we can **zero-shot generalize to instructions with unseen words**.

Compositional Abstractions Enable Generalization

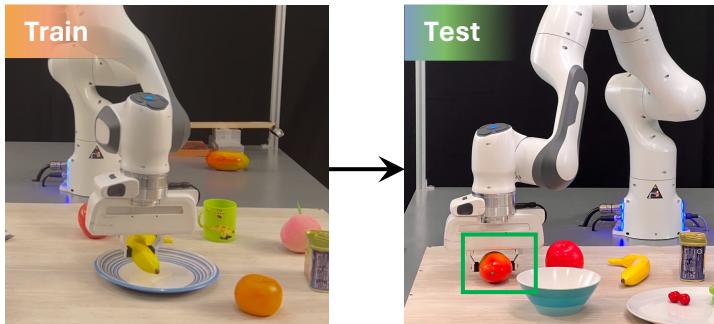
Generalization to Novel Objects



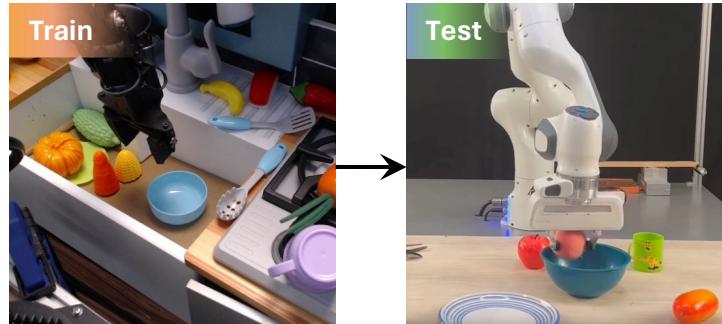
Generalization to Novel States



Generalization to Novel Words



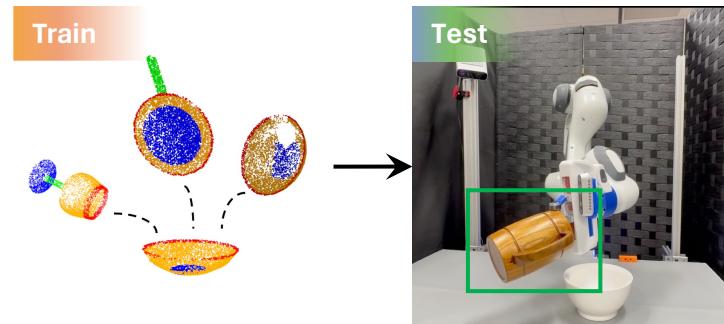
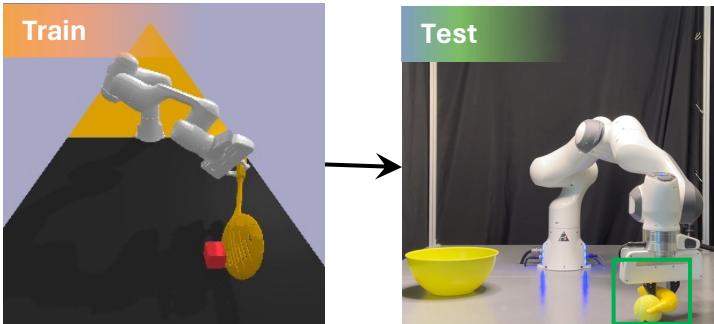
Generalization to Novel Embodiments



By factorizing the robot controller and the generation of object trajectories, we can train policies on videos of other robots and even humans, and deploy on a different robot.

Compositional Abstractions Enable Generalization

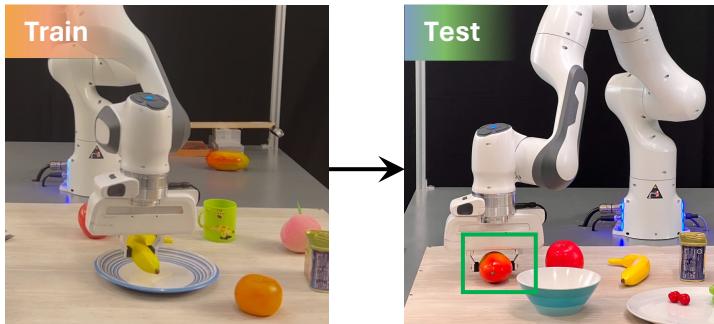
Generalization to Novel Objects



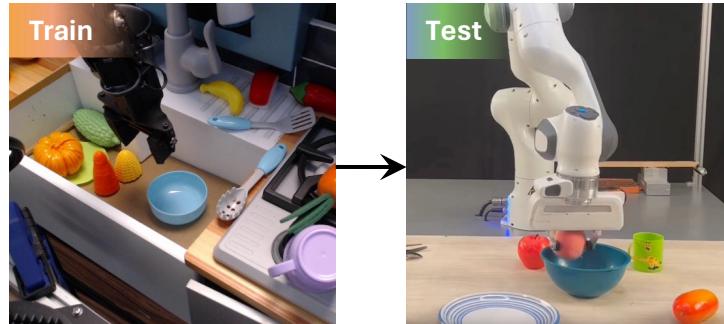
Generalization to Novel States



Generalization to Novel Words



Generalization to Novel Embodiments



Interpretation of Under-Specified Goals

Set up a table for my breakfast.



By factorizing goals into finer-grained object relationships using LLMs, we build systems that can **interpret under-specified human goals**.

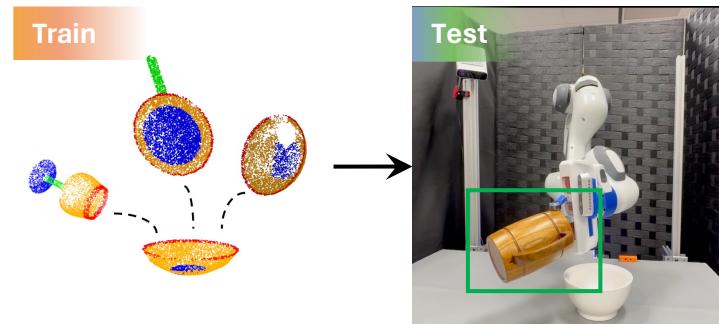
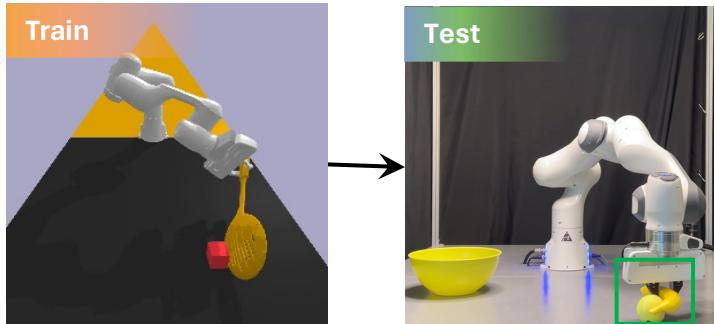
Compositional Abstractions Enable Generalization

Principles: Compositional abstractions for

- states (objects, relations, and sparse transition models), and
- actions and plans (hierarchical compositions and decompositions)

enable data-efficient learning, faster planning, and better generalization.

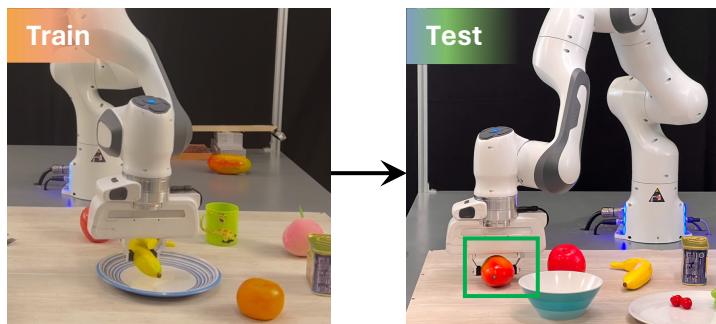
Generalization to Novel Objects



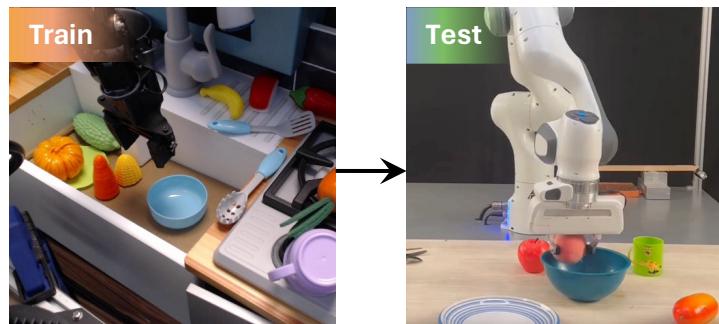
Generalization to Novel States



Generalization to Novel Words



Generalization to Novel Embodiments



Interpretation of Under-Specified Goals

Set up a table for my breakfast.





Word	Syntax	Semantics	Concept Representations
<i>orange</i>	<i>set/set</i>	$\lambda x. \text{filter}(x, \text{orange})$	ORANGE
		$\text{orange}(\text{object_1}) = \text{TRUE}$	
<i>left</i>	<i>set\set/set</i>	$\lambda x \lambda y. \text{relate}(x, y, \text{left})$	LEFT
		$\text{left}(\text{object_1}, \text{object_2}) = \text{FALSE}$	
<i>move</i>	<i>action\set/set</i>	$\lambda x \lambda y. \text{action}(x, y, \text{move})$	MOVE
Precondition: $\text{relate}(\text{cylin}, \text{hand}, \text{holding})$ Postcondition: $\text{not}(\text{relate}(\text{cylin}, \text{hand}, \text{holding})) \text{ relate}(\text{cylin}, \text{bottle}, \text{left})$			

Visual representation

object_1	[Color Swatches]
object_2	[Color Swatches]

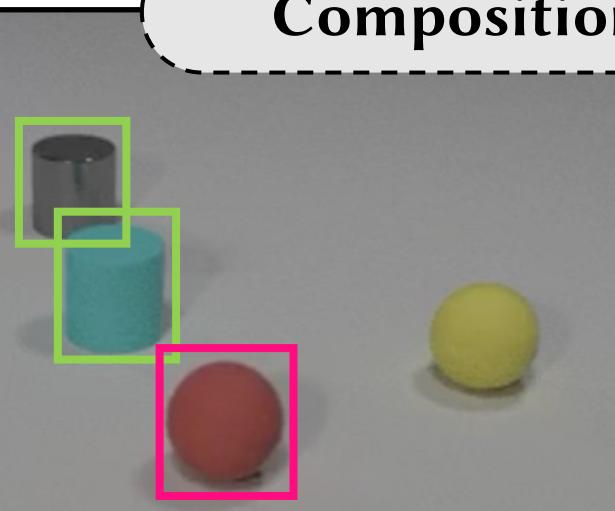
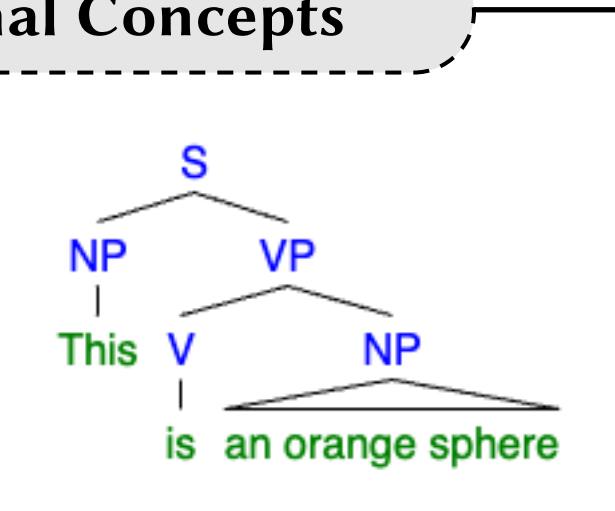
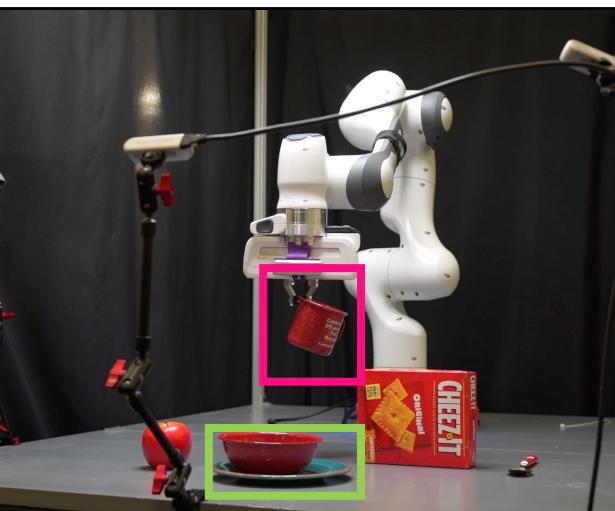


Word	Syntax	Semantics	Concept Representations
<i>orange</i>	<i>set/set</i>	$\lambda x. \text{filter}(x, \text{orange})$	ORANGE 
	<i>orange(object_1)</i> = TRUE		
<i>left</i>	<i>set\set/set</i>	$\lambda x \lambda y. \text{relate}(x, y, \text{left})$	LEFT 
	<i>left(object_1, object_2)</i> = FALSE		
<i>move</i>	<i>action\set/set</i>	$\lambda x \lambda y. \text{action}(x, y, \text{move})$	MOVE 

Visual representation
object₁ 
object₂ 

Precondition: relate(cylin, hand, holding)
Postcondition: not(relate(cylin, hand, holding)) relate(cylin, bottle, left)

Compositional Concepts

			
<p>Query: Is there a dresser on the left side of the cabinet?</p> <p>Visual Reasoning</p>	<p>Query: Which ball is responsible to the cylinder collision?</p> <p>Dynamics and Causality</p>	<p>Query: This is an orange sphere.</p> <p>Grounded Syntax Learning</p>	<p>Query: Put the mug to the right of the Plate.</p> <p>Robotic Manipulation</p>