

Planning with Hierarchy and Abstraction

Tom Silver

Robot Planning Meets Machine Learning

Princeton University

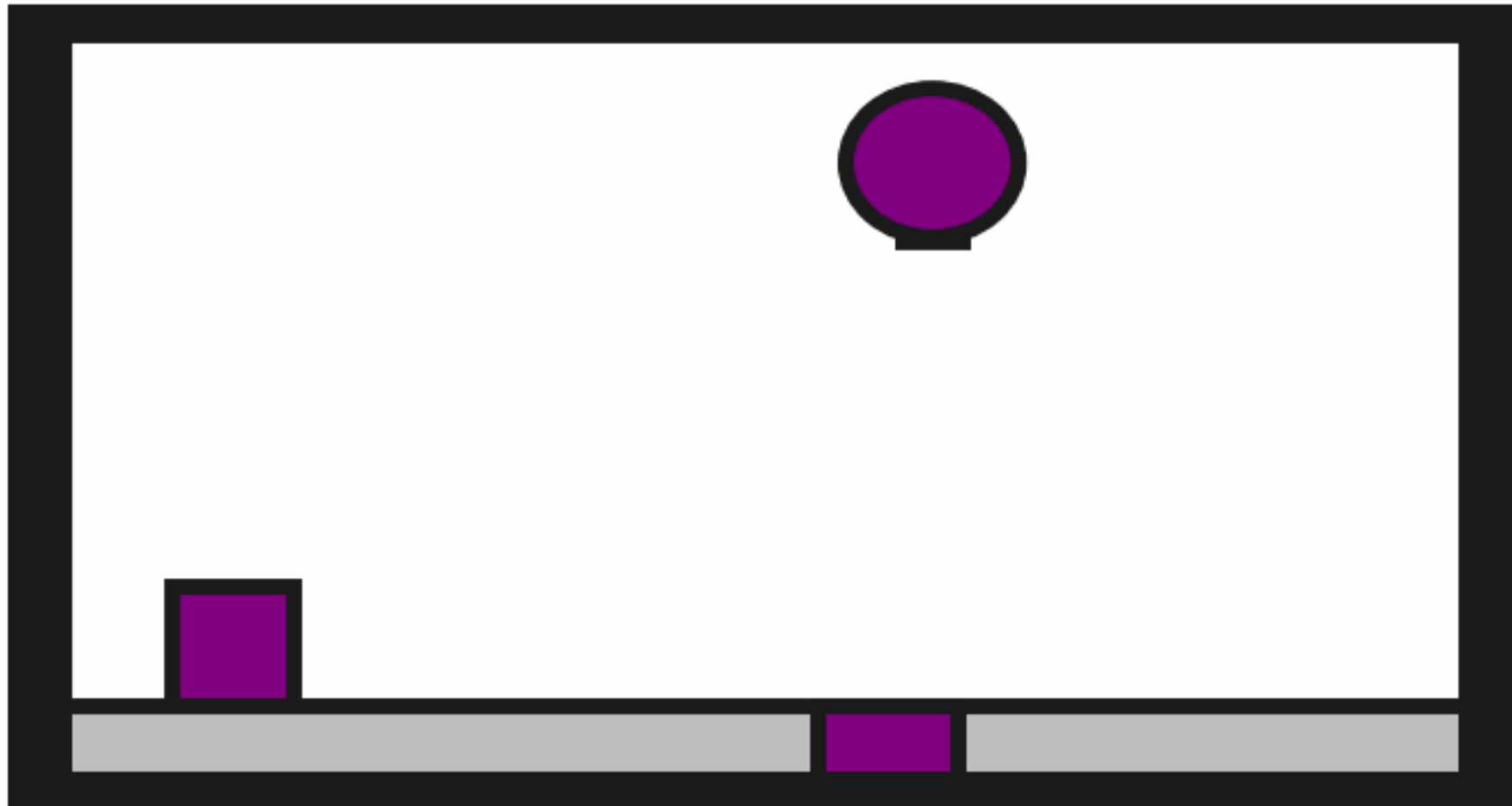
Fall 2025

Recap and Preview

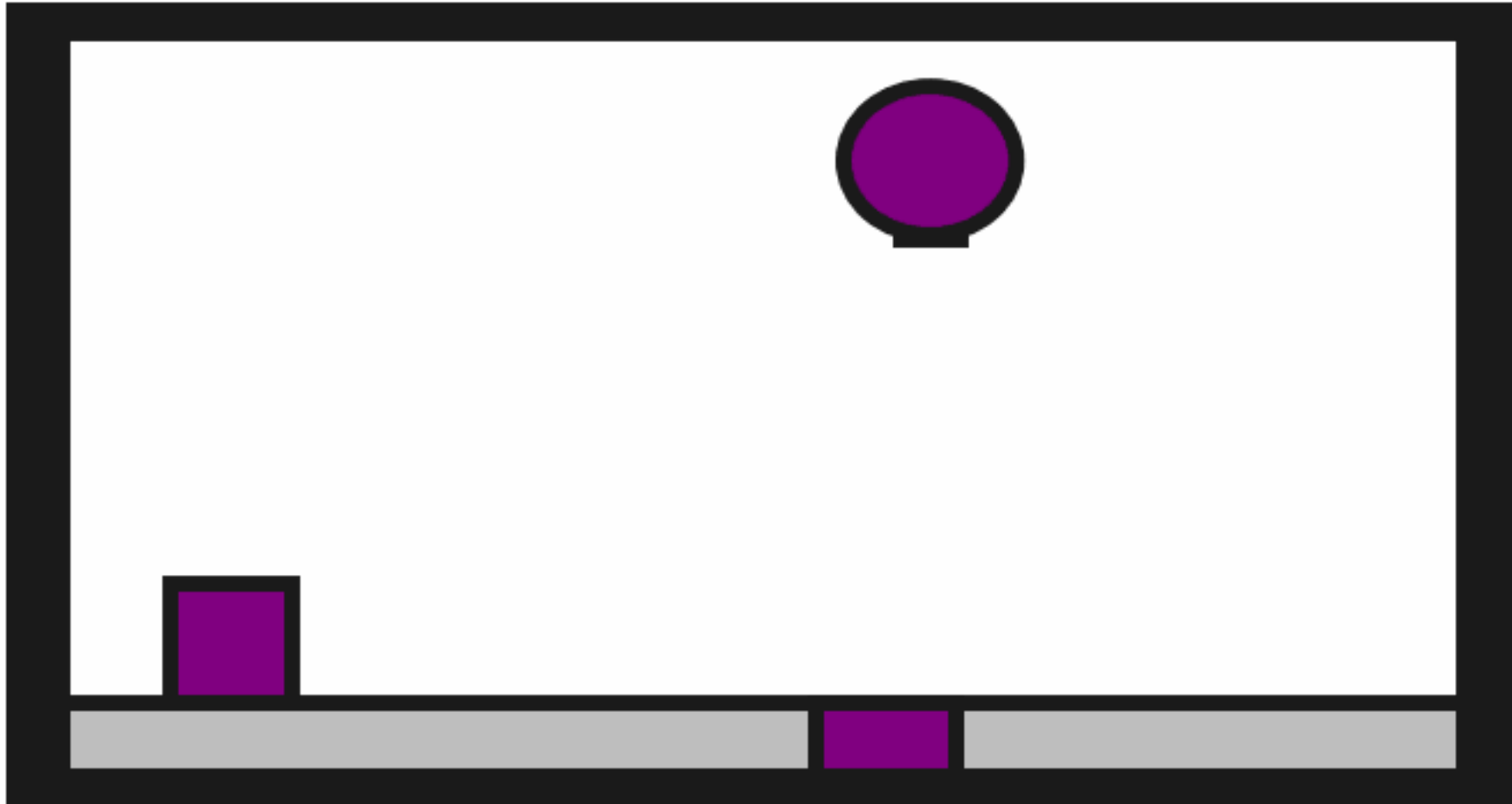
Last time: planning in continuous state and action spaces

This time: same problem setting, new tools: abstractions!

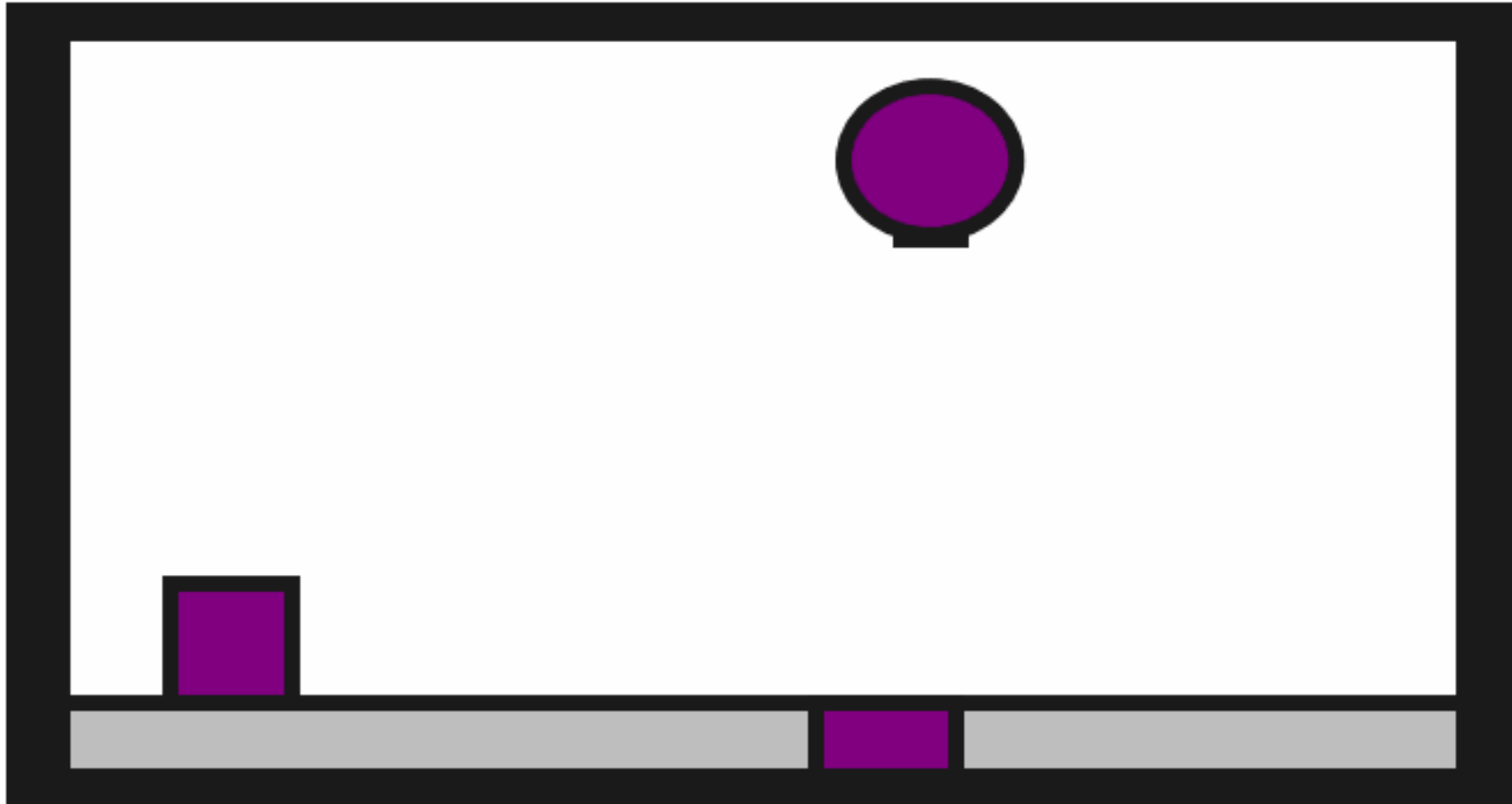
Human Demo



Random Actions



Task Distribution



Example

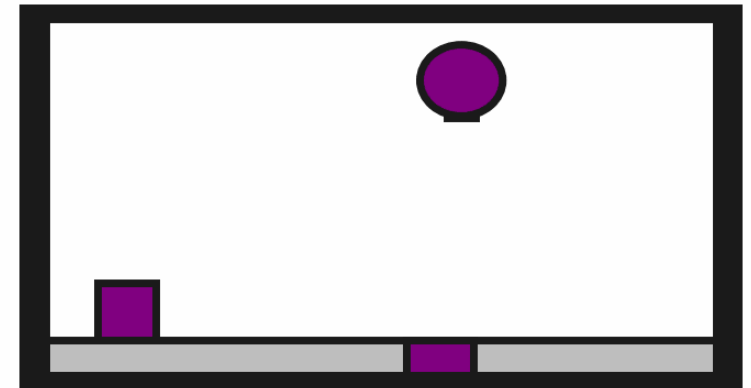
Which planners could we try? Would they work?

State space: Robot config, block pose (8D)

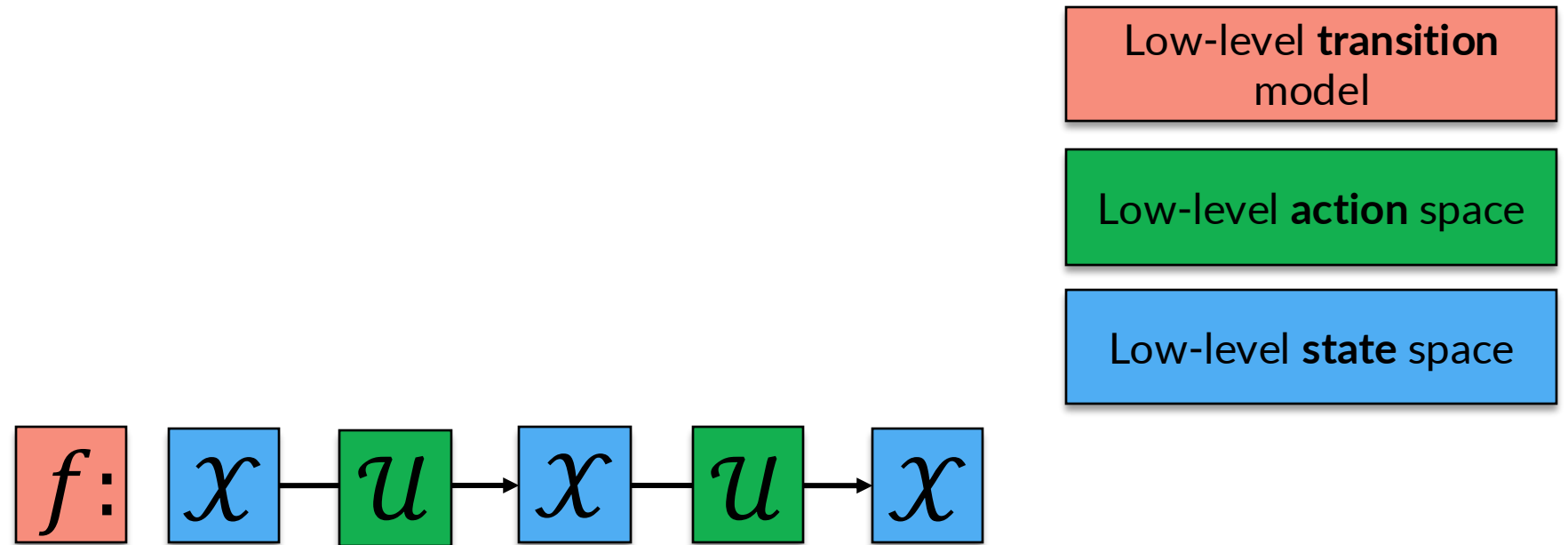
Action space: Pose change, vacuum (5D)

Transition function: Apply action but disallow collisions (no change)

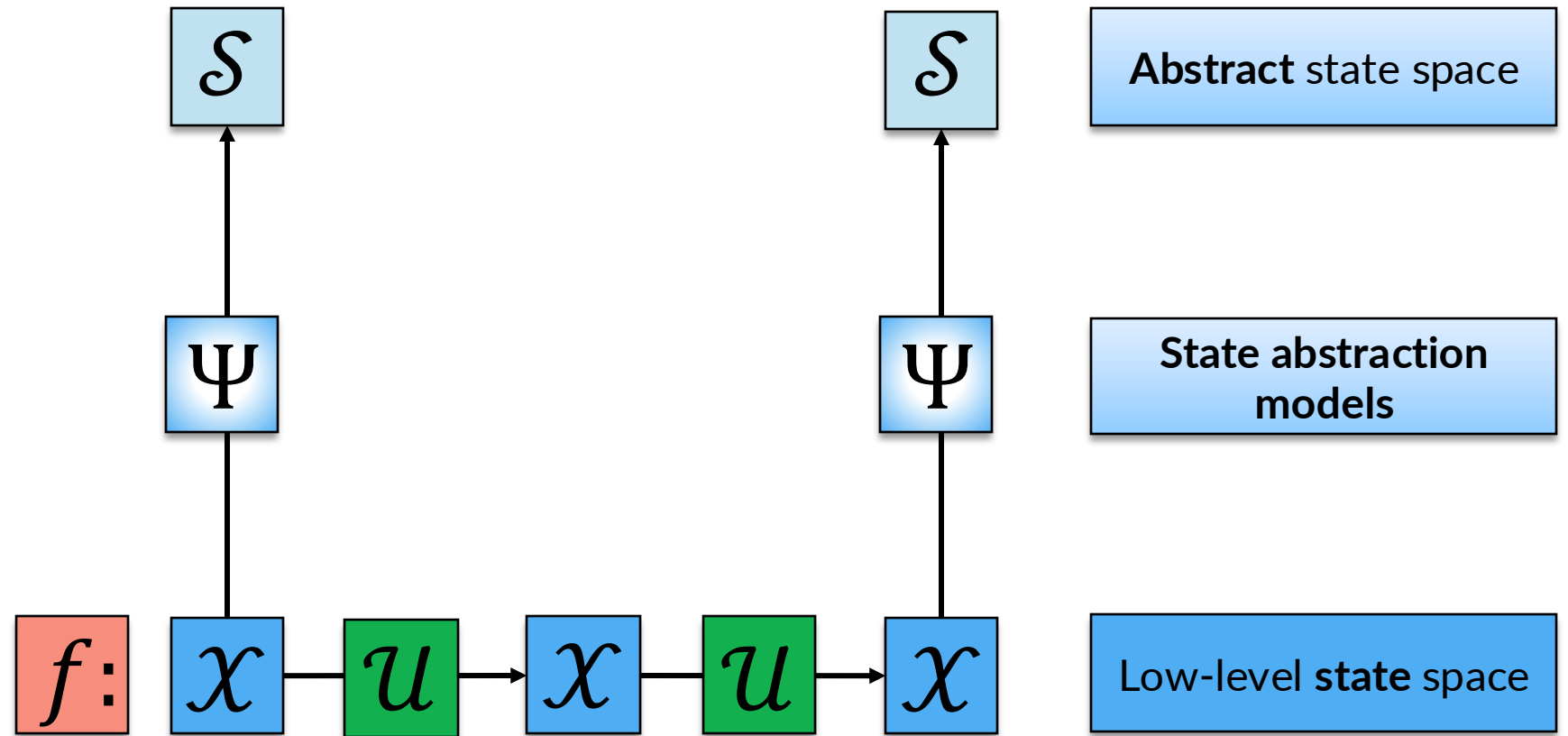
Cost function: -1 until block on target



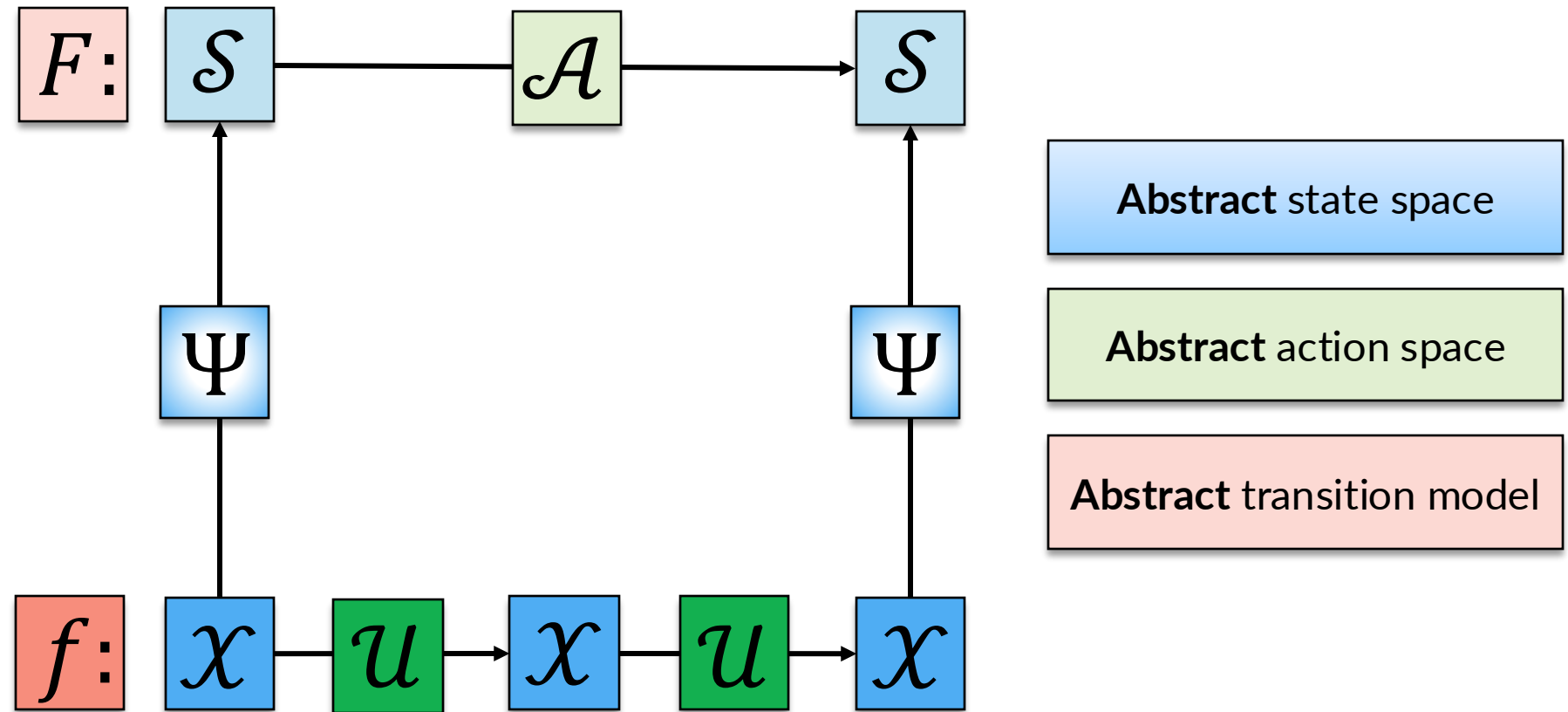
A Two-Level Hierarchy



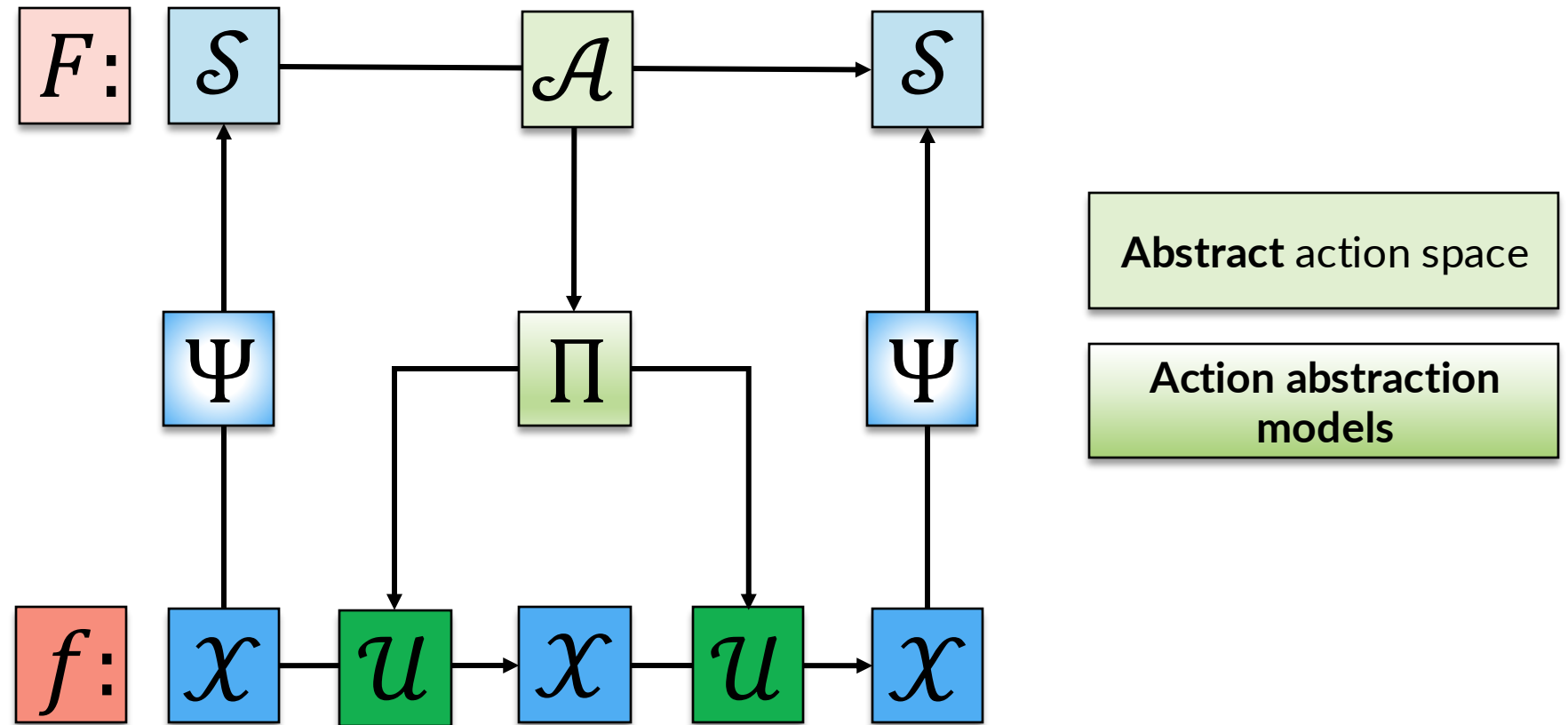
A Two-Level Hierarchy



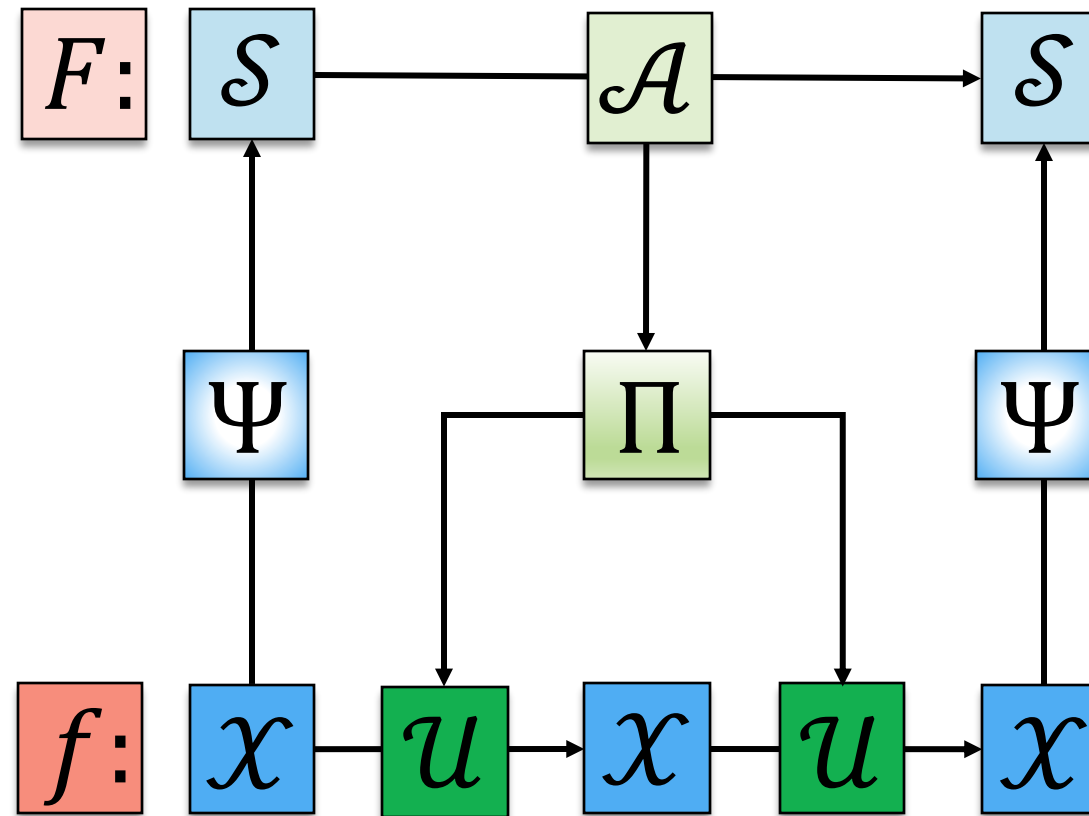
A Two-Level Hierarchy



A Two-Level Hierarchy

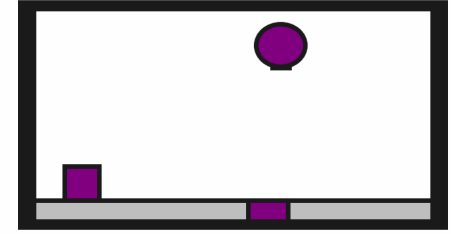


A Two-Level Hierarchy

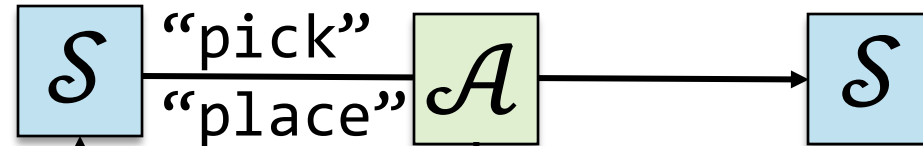


Maybe it's easier
to plan up here!

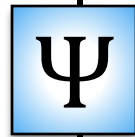
Abstractions in Example



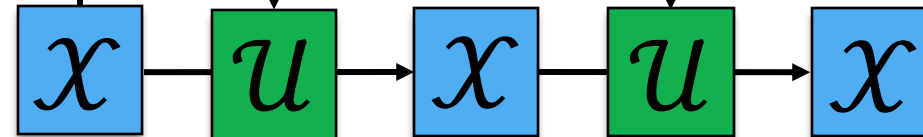
HoldingBlock: bool
BlockOnTarget: bool



State abstraction

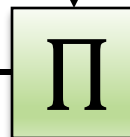


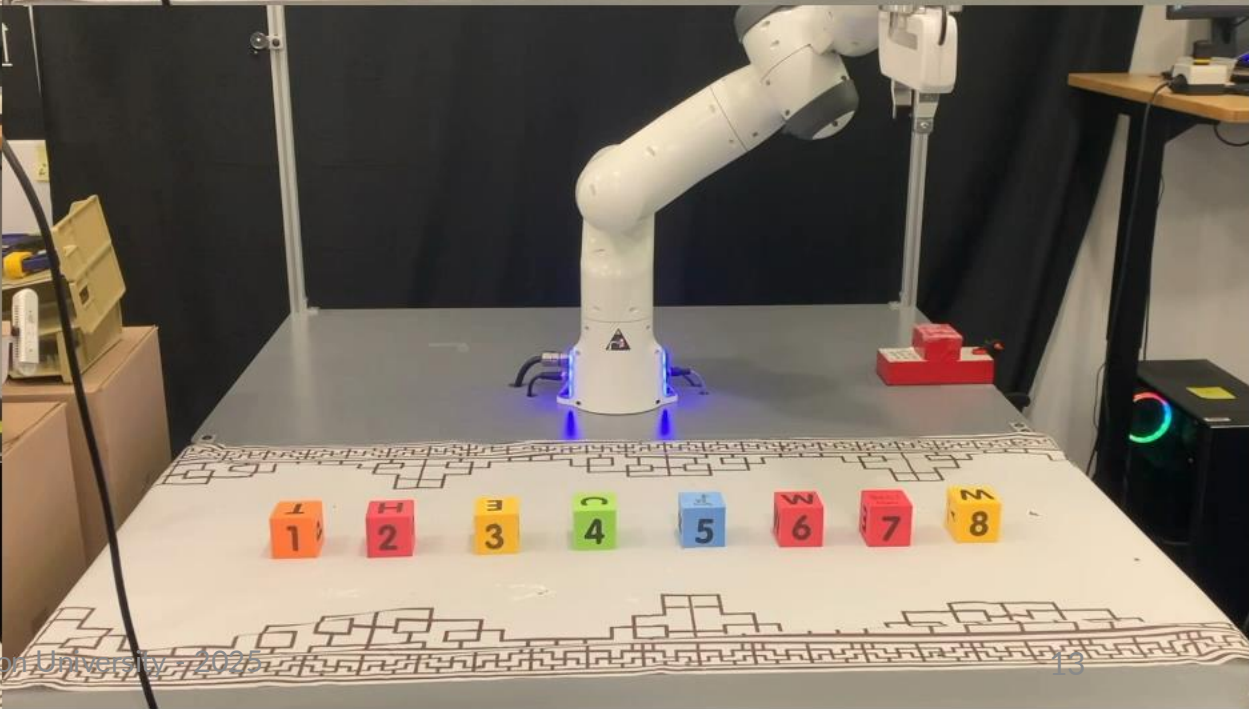
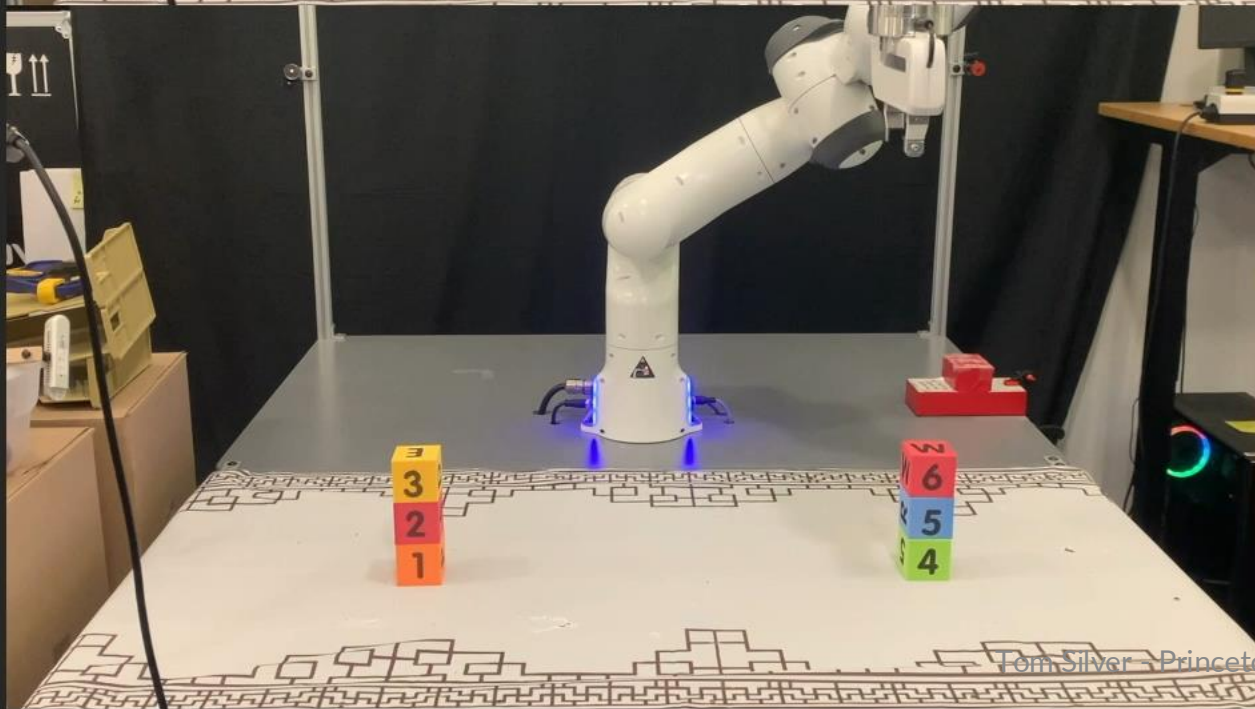
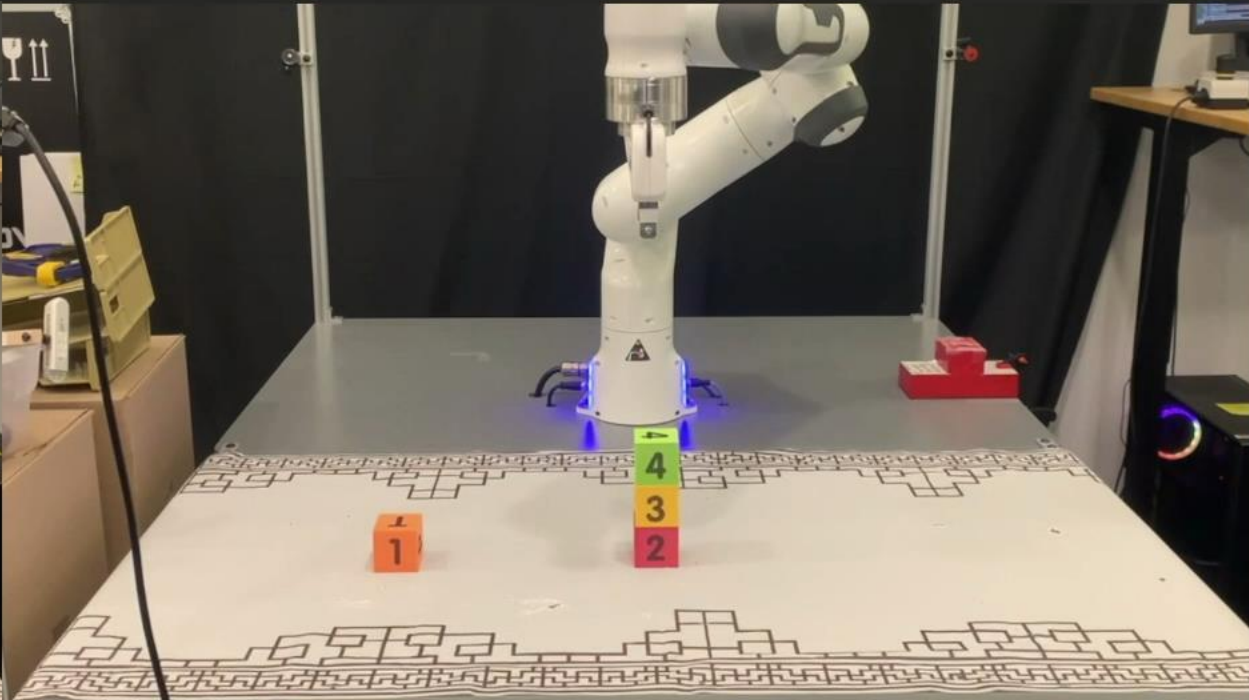
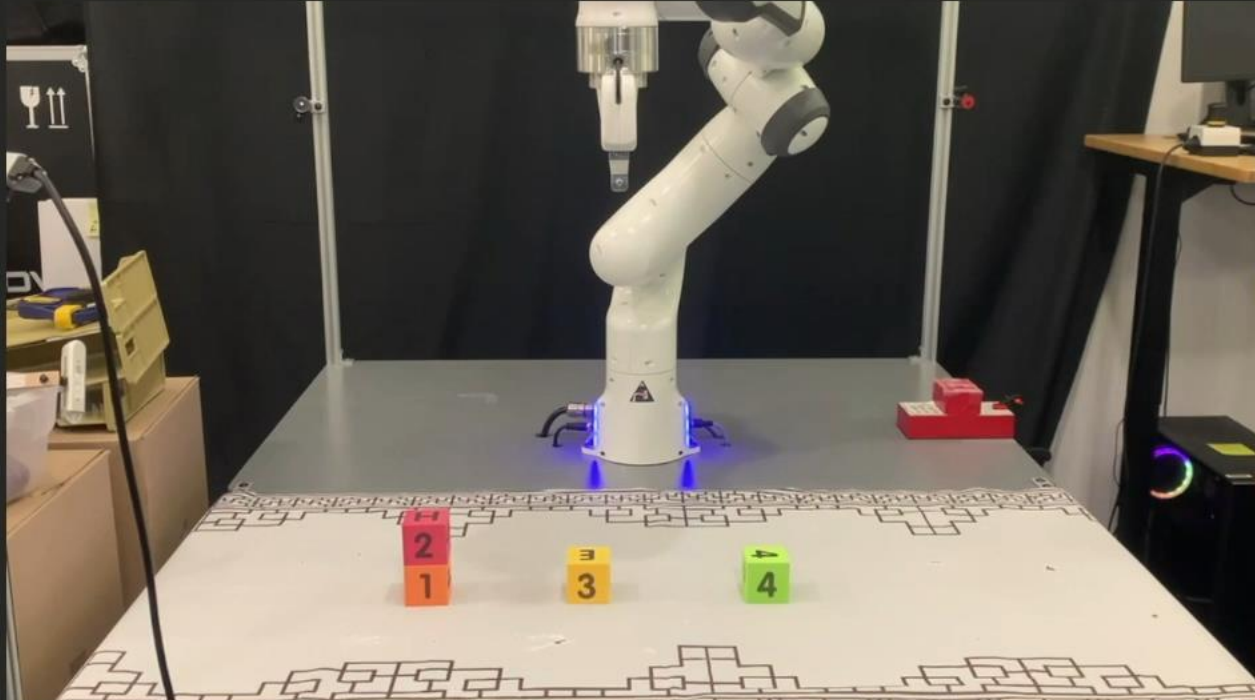
Robot config, block pose



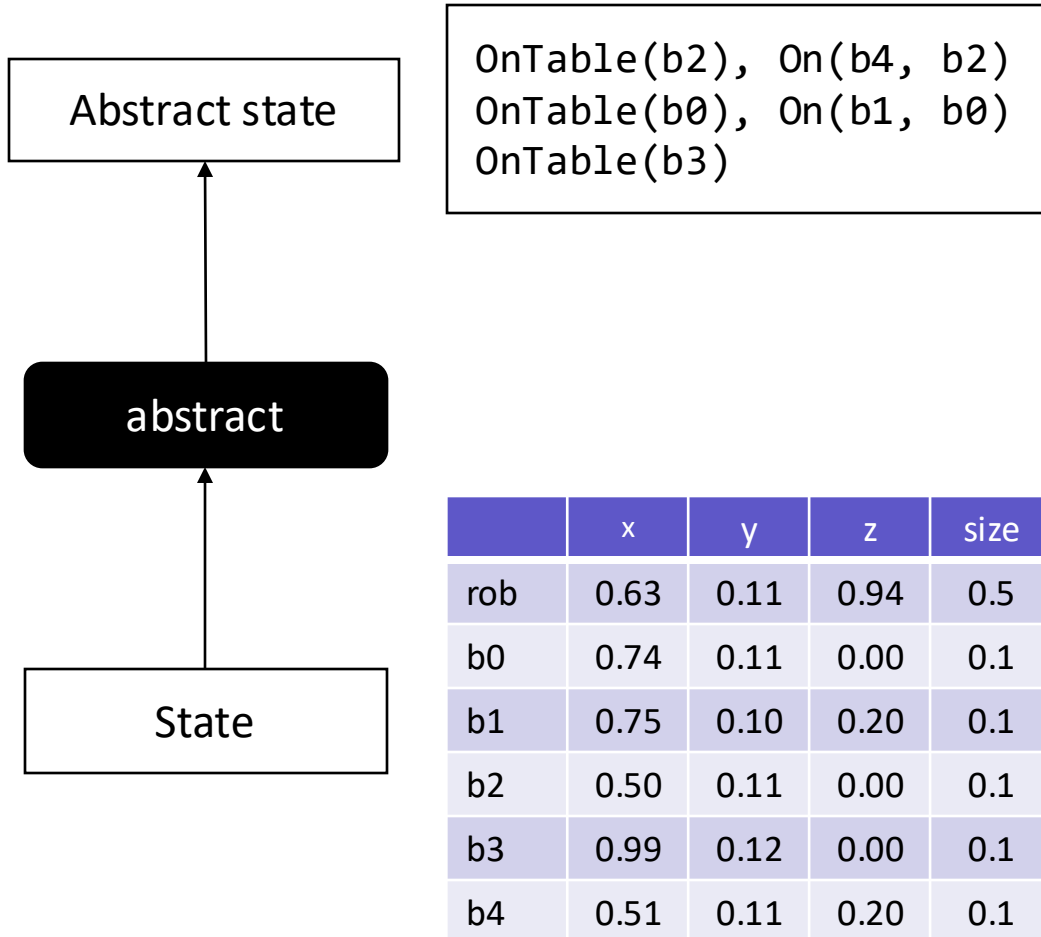
Pose change, vacuum (5D)

Action refinement

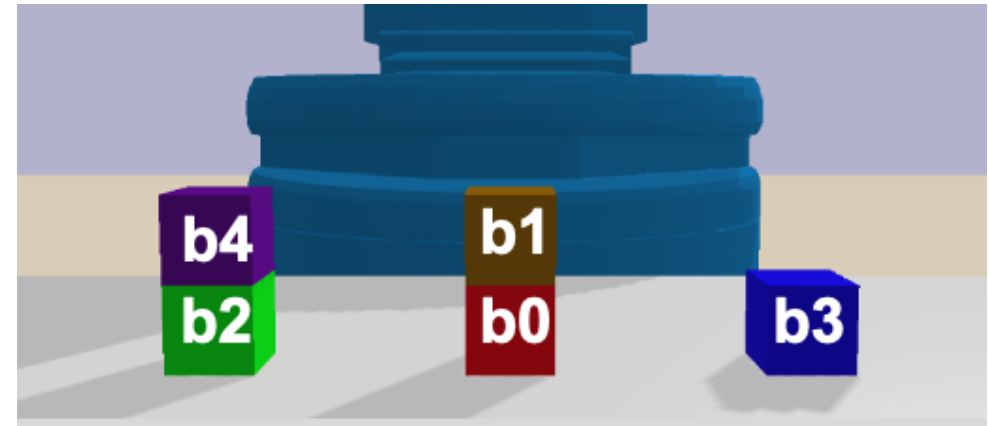




State Abstraction with Predicates



```
def classifyOnTable(state, ?block):  
    state[?block].z < 1e-5  
  
def classifyOn(state, ?top, ?bot):  
    (state[?top].z - state[?bot].z -  
     state[?bot].size) < 1e-5 &  
    (state[?top].x - state[?bot].x) < 1e-5 &  
    (state[?top].y - state[?bot].y) < 1e-5 &
```



Operators as Abstract Actions

Arguments

List of typed variables

Preconditions

What must be true in order to use this operator?

Add/Delete Effects

How is the abstract state changed by this operator?

Operator-PickFromTable:

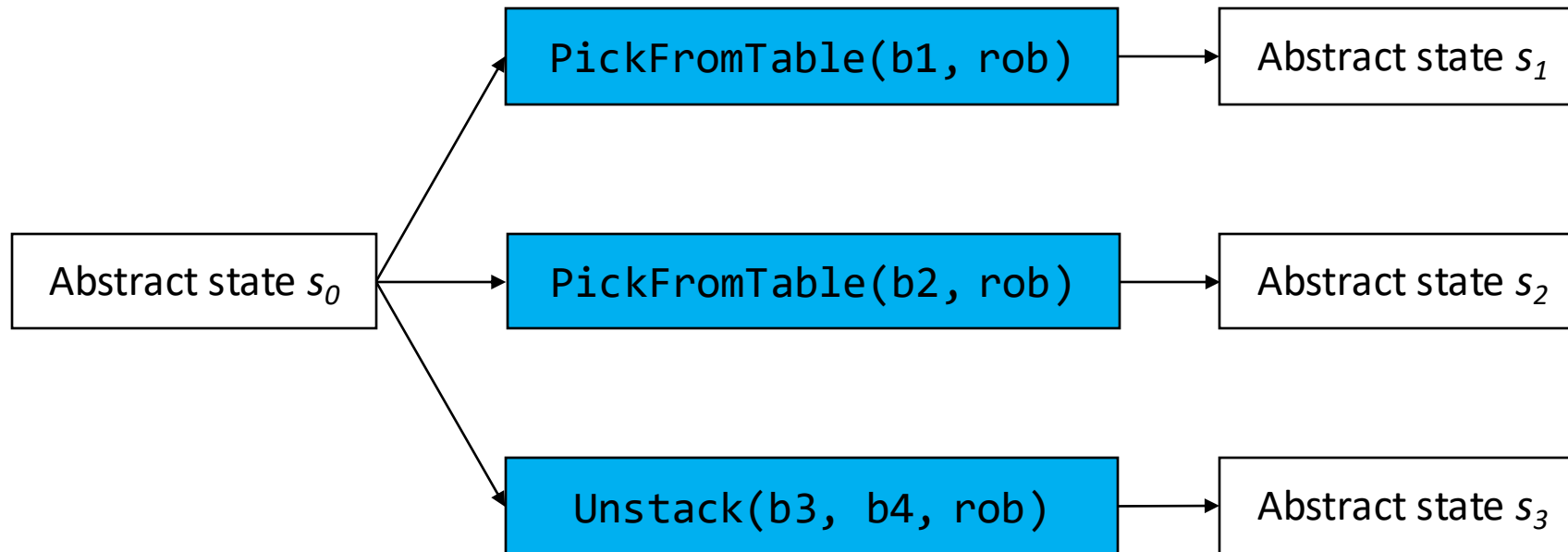
Arguments: [**?b** - block, **?r** - robot]

Preconditions: {GripperOpen(**?r**),
OnTable(**?b**)}

Add effects: {Holding(**?b**)}

Delete effects: {GripperOpen(**?r**),
OnTable(**?b**)}

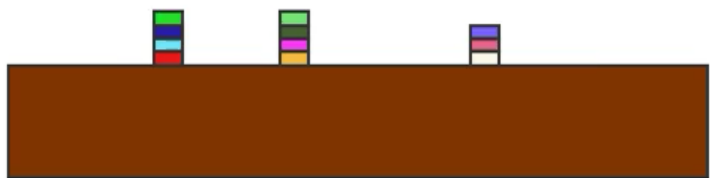
An Abstract Transition Model



An abstract (partial) transition model

Why Predicates and Operators?

If we have predicates and operators, then we can use very powerful off-the-shelf symbolic planners!



Blocks World

Plan length: 28

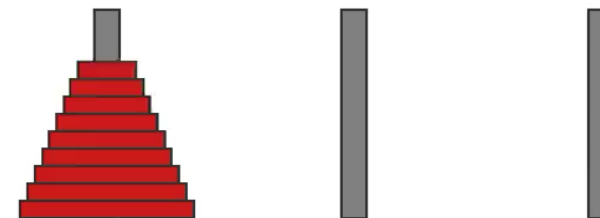
Planning time: 0.12 s



Sokoban

Plan length: 167

Planning time: 0.25 s

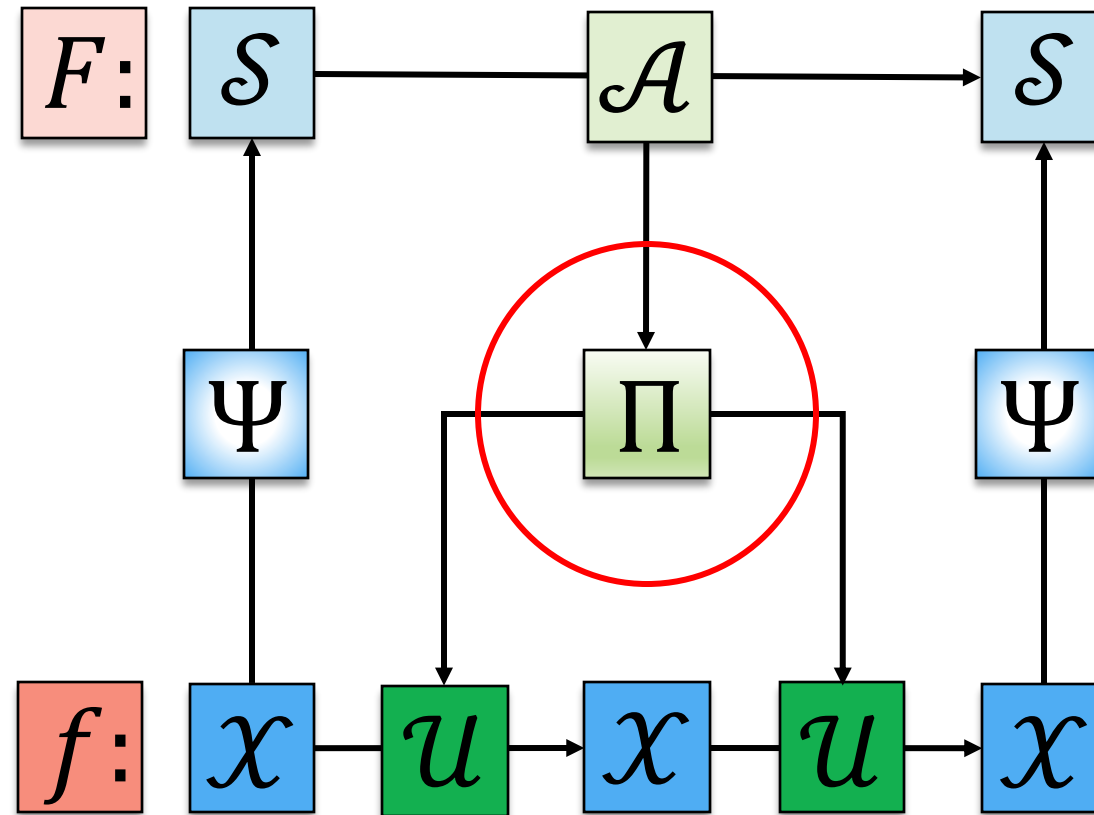


Hanoi

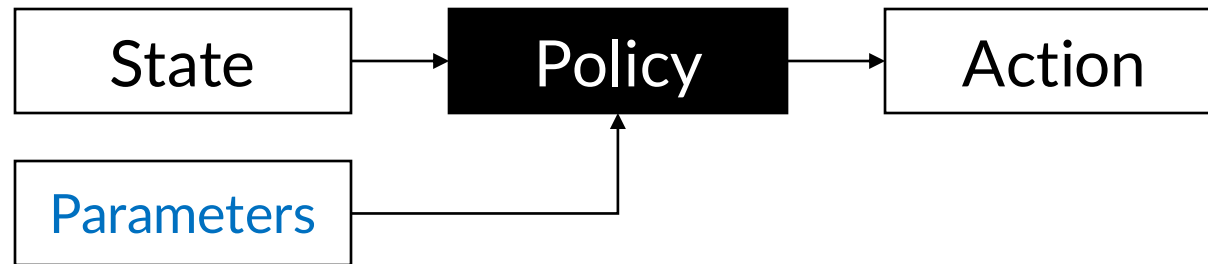
Plan length: 579

Planning time: 0.22 s

Planning with Fast Downward (<https://www.fast-downward.org>)
Rendering and simulation with PDDLgym (<https://github.com/tomsilver/pddlgy>)



Policies as Abstract Action Models



Same as
operator

```
def policyPickFromTable(state, ?b, ?r):  
    dx = (state[?b].x - state[?r].x)  
    dy = (state[?b].y - state[?r].y)  
    dz = (state[?b].z - state[?r].z)  
    return [dx, dy, dz]
```

Simplified example

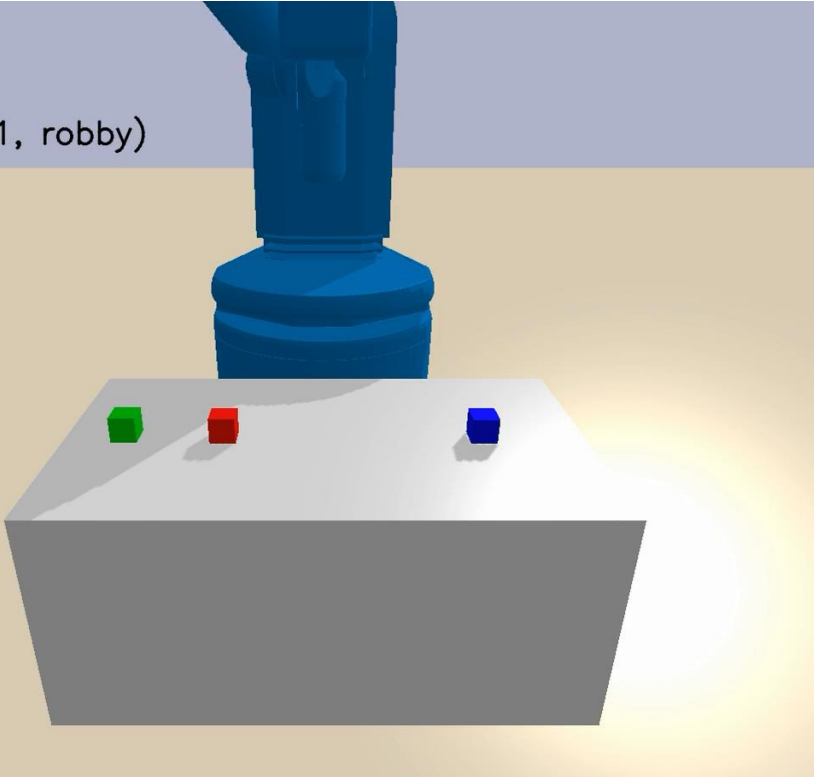
The policy should *achieve* the operator effects
when the operator preconditions hold

Example Policy Executions

PickFromTable(block1, robby)

Abstract State:

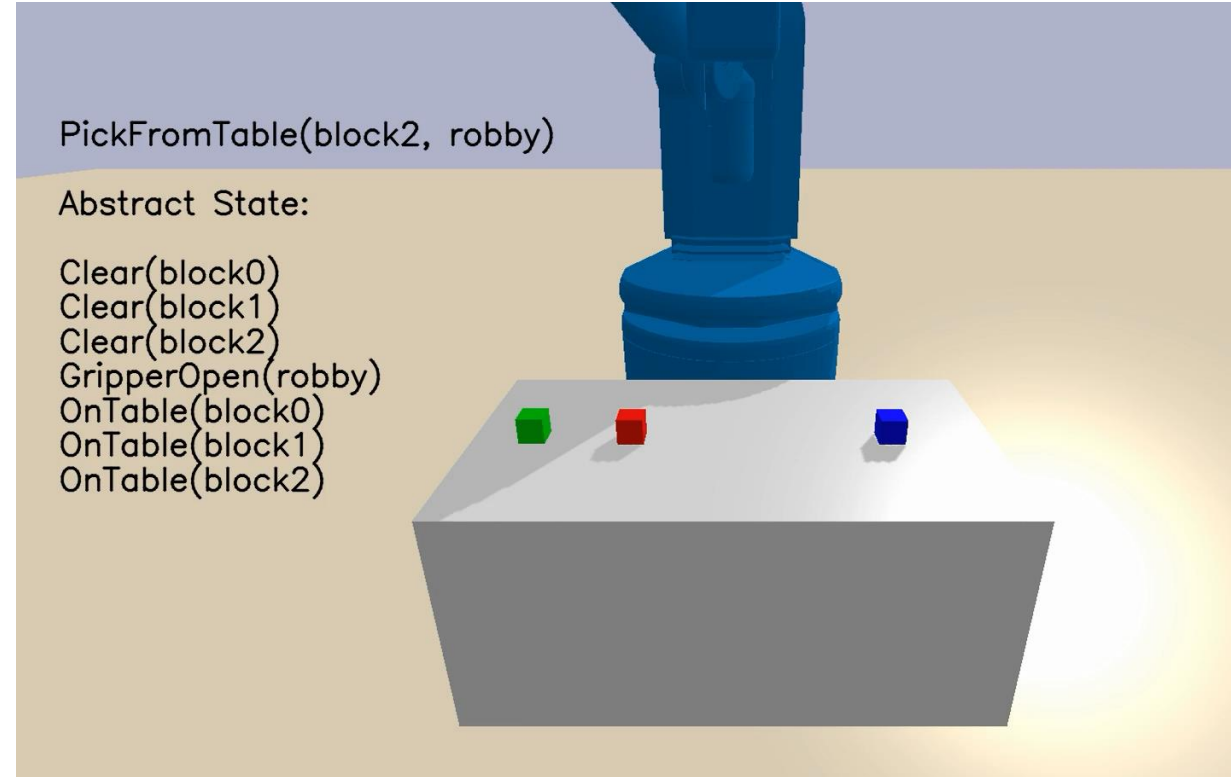
```
Clear(block0)
Clear(block1)
Clear(block2)
GripperOpen(robby)
OnTable(block0)
OnTable(block1)
OnTable(block2)
```



PickFromTable(block2, robby)

Abstract State:

```
Clear(block0)
Clear(block1)
Clear(block2)
GripperOpen(robby)
OnTable(block0)
OnTable(block1)
OnTable(block2)
```

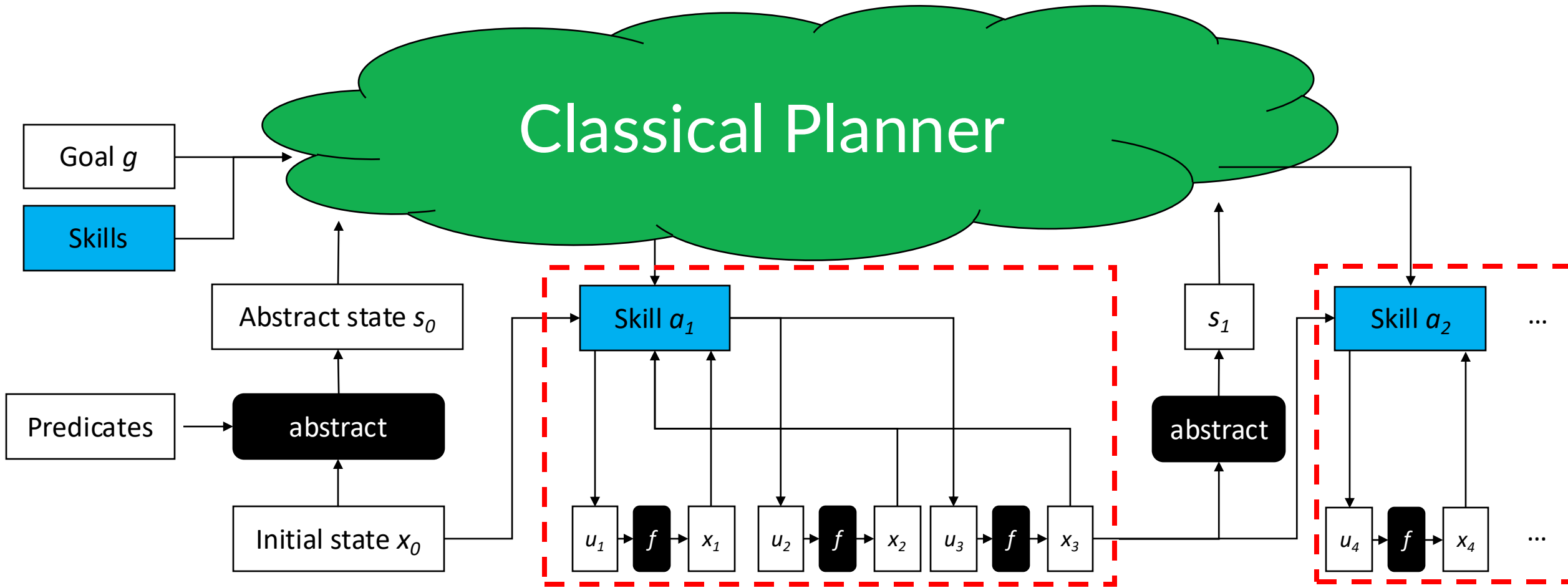


Skills: abstract actions that bring the robot from one abstract state to another

What abstract state transition?

How should I get there?

A skill has an **operator** and a **policy**

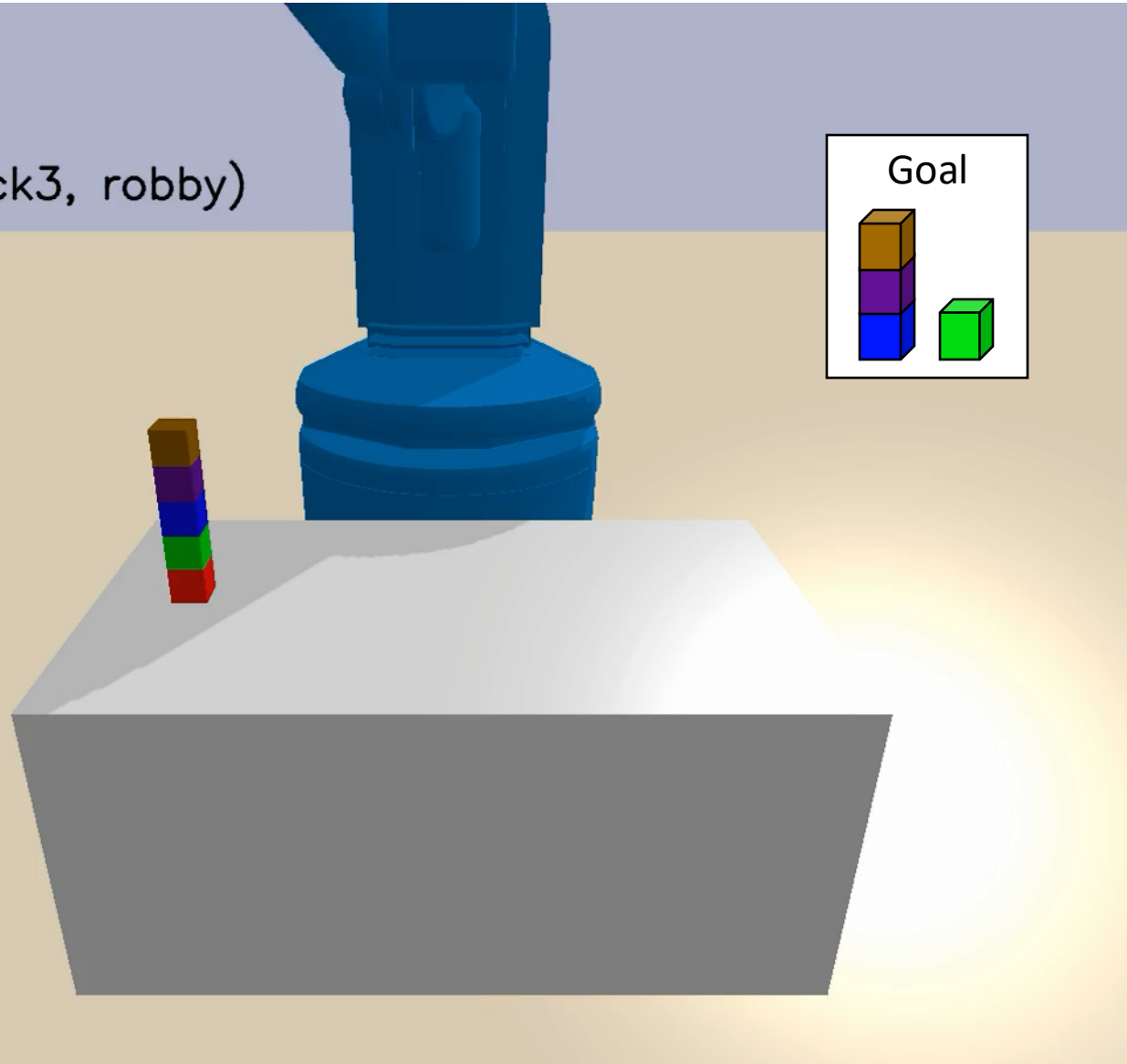
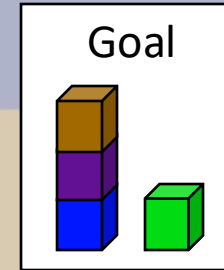


Action abstraction via skills

Unstack(block4, block3, robby)

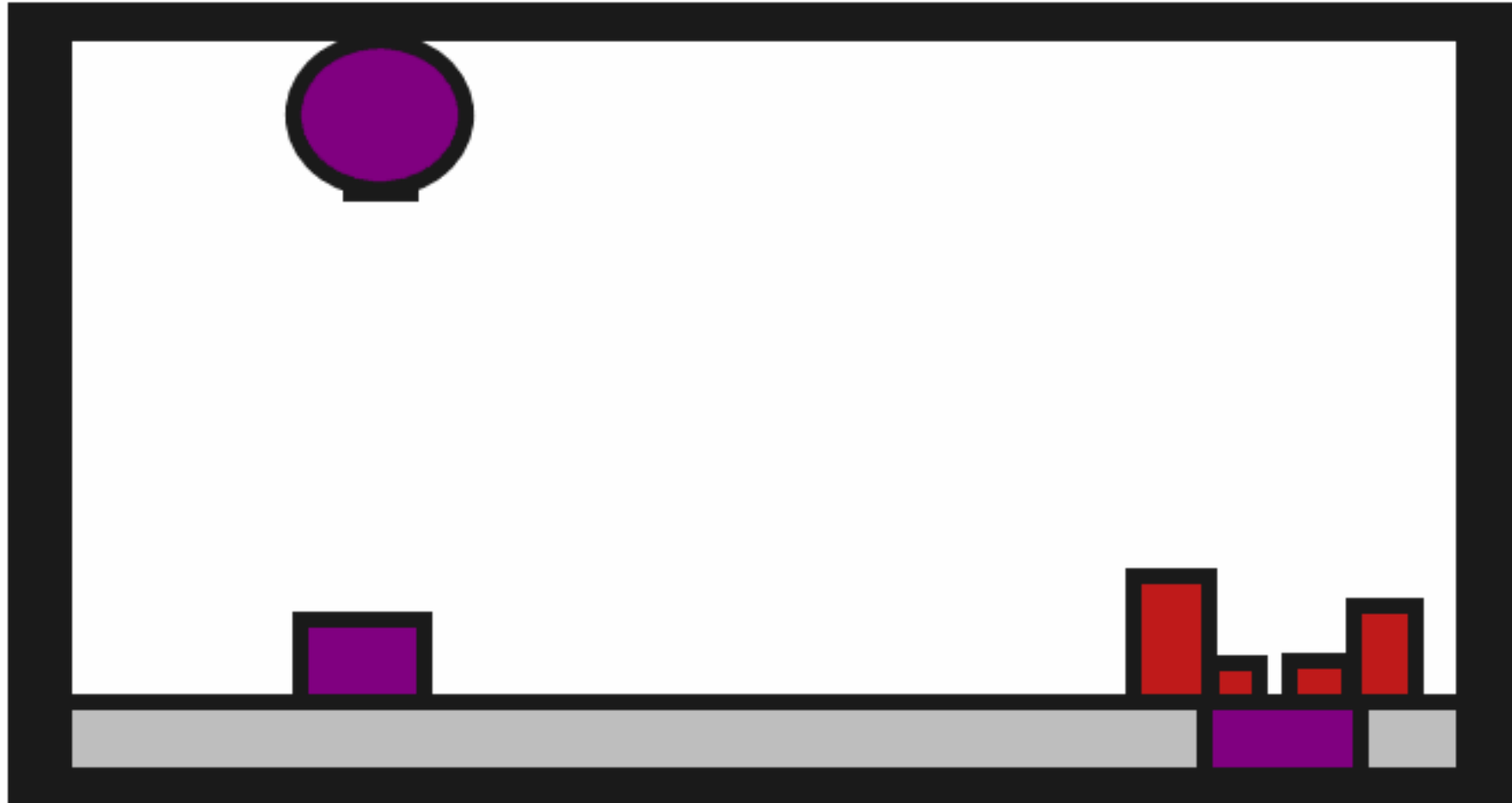
Abstract State:

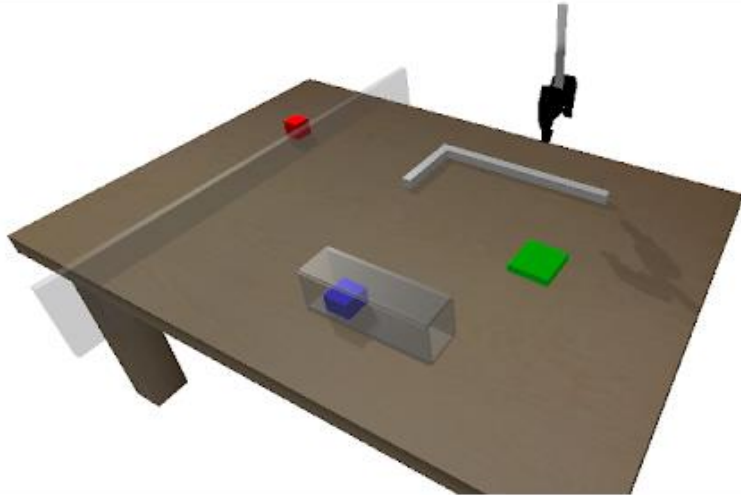
Clear(block4)
GripperOpen(robby)
On(block1, block0)
On(block2, block1)
On(block3, block2)
On(block4, block3)
OnTable(block0)



The abstractions might be liars...











“Deep Affordance Foresight: Planning Through What Can Be Done in the Future.” Danfei Xu, Ajay Mandlekar, Roberto Martin-Martin , Yuke Zhu, Silvio Savarese and Li Fei-Fei. ICRA 2021.

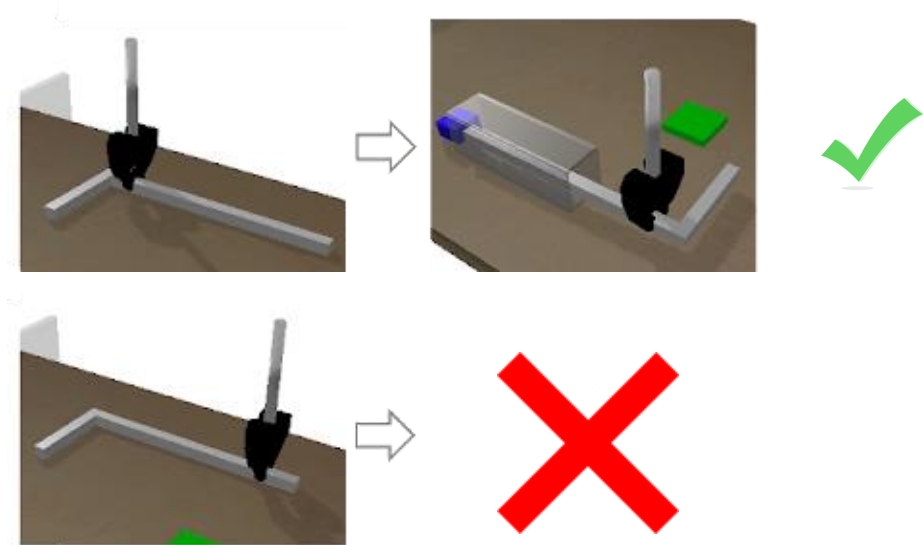
Operator-PushOutOfTube:

Arguments: [, , ]

Preconditions: {Holding(, ,
InTube()}

Add effects: {OutOfTube()}

Delete effects: {InTube()}



Possible Conclusions from this Example

1. Insufficient predicates → learn new predicates

HoldingBottom, HoldingTop, etc.

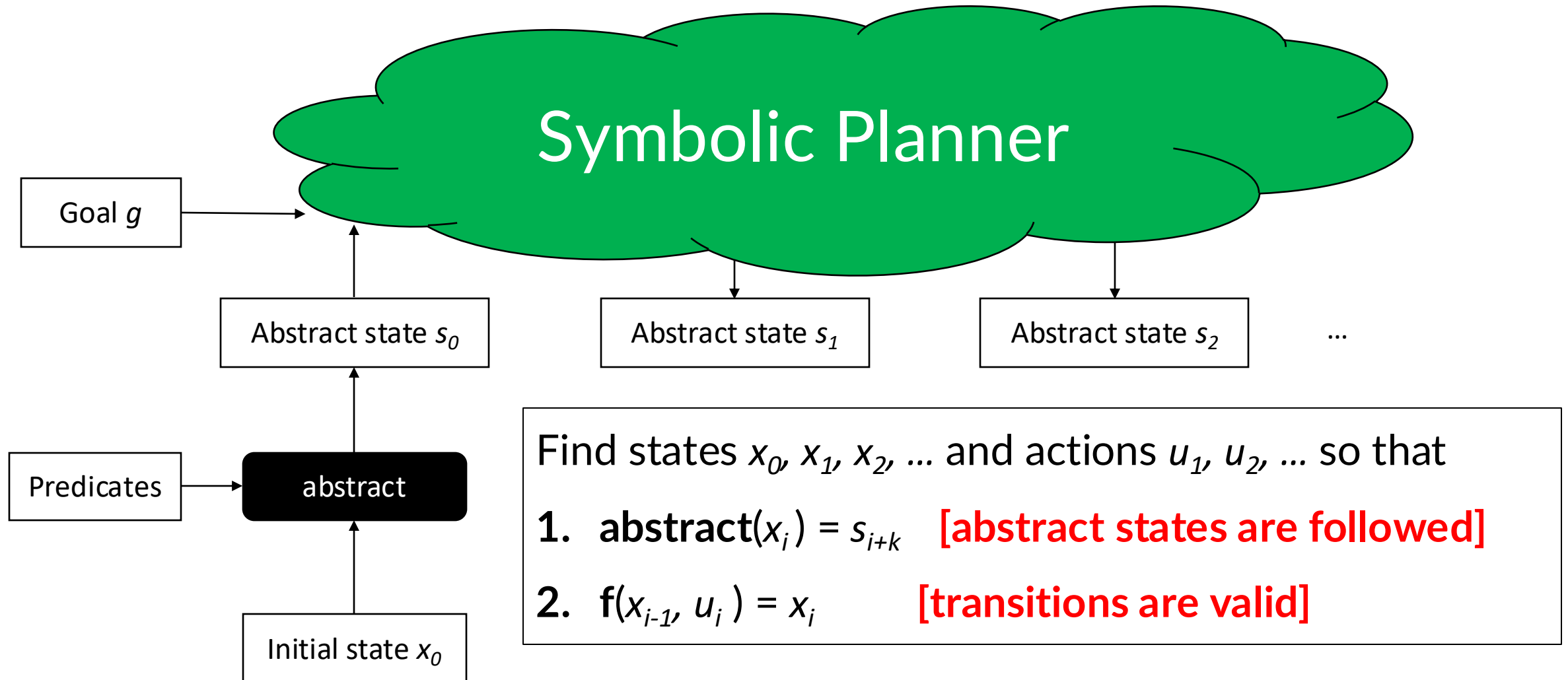
2. Insufficient policies → learn better policies

Put down the tool and regrasp if needed

3. Insufficient planner → be less trusting of the abstractions

View abstractions as *guidance* for low-level planning

Bilevel Planning: View Abstractions as *Constraints*



Logic-Geometric Programming

Toussaint (2015)

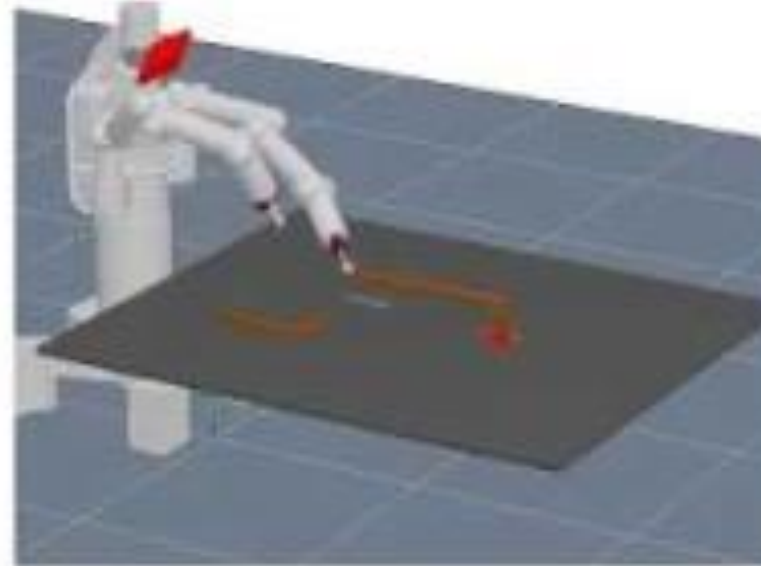
Found Solutions

The only goal specification is to touch the red ball with either hand, or to let the blue ball touch the green patch.

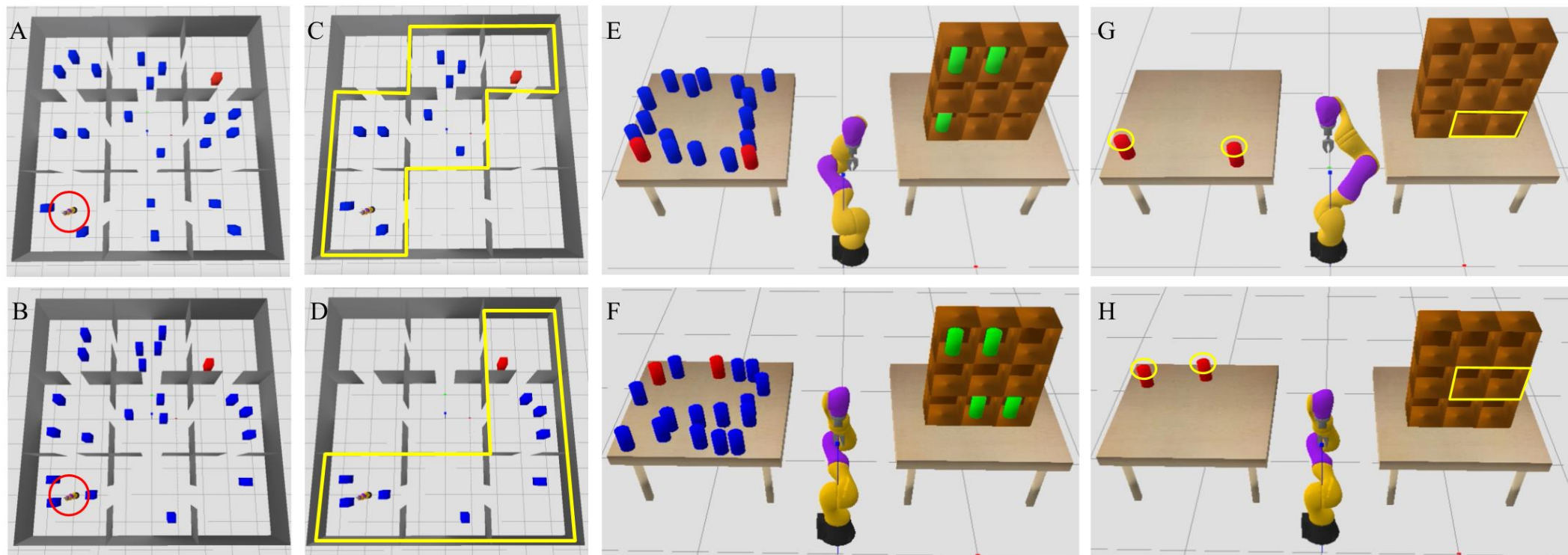
The system has full knowledge of the scene, including the geometric shapes of all objects, BUT knows of no further semantics specific to objects.

Toussaint, Allen, Smith, Tenenbaum:
Differentiable Physics and Stochastic Modes for
Task-Learning and Manipulation Planning (RSS 2018)

The double hook, in analogy to Betty the Crow



Side Note: Constraints Can Help Planning in Multiple Ways



From Chitnis*, Silver*, et al. (2020)

Logic-Geometric Programming

Toussaint (2015)

Possible issues:

1. Optimizing in low-level state and action space remains hard for long-horizon problems

Recall from last lecture

General Trick 2: Optimize Splines Instead

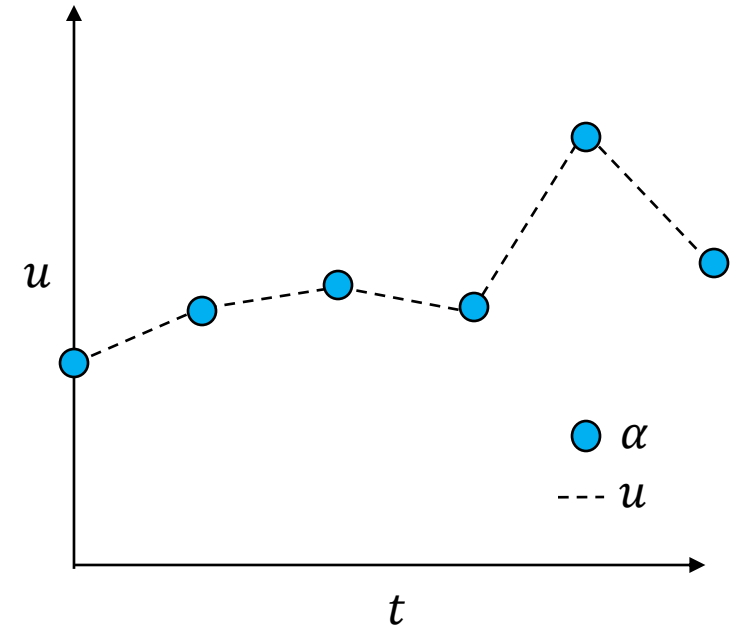
Optimizing $(u_0, u_1, \dots, u_{H-1})$ is slow for large H

Instead, optimize over lower-dimensional α :

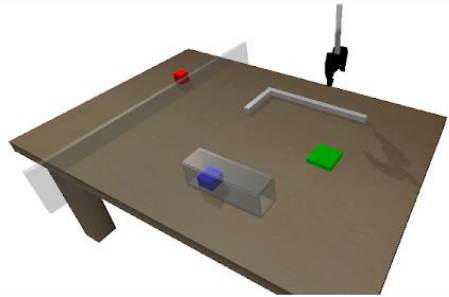
$$u_t = f(t, \alpha) \text{ where } \alpha \in \mathbb{R}^d \text{ and } d \ll mH$$

Common: think of α as “action waypoints” and interpolate between them

For example, linear splines (see right)




Parameterized Skill Policies



Operator-PickTool:

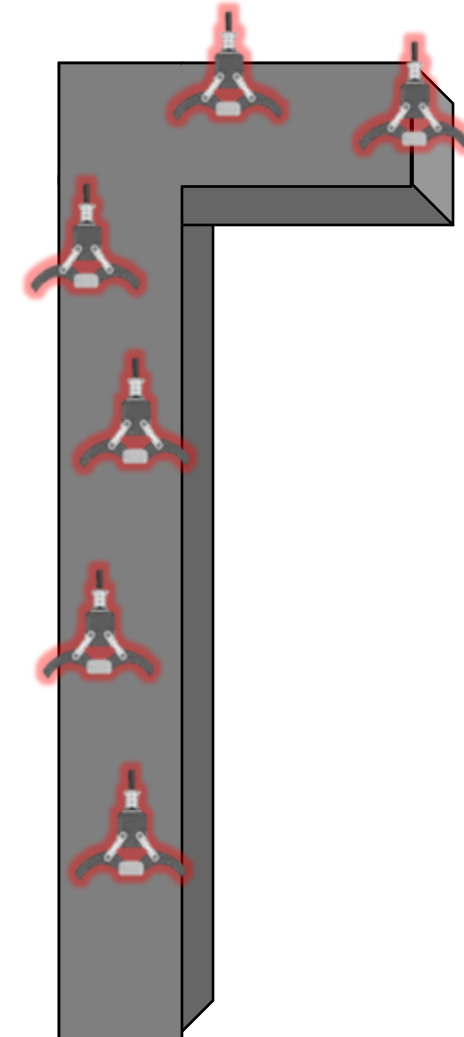
Arguments: [\mathbb{T} , ]

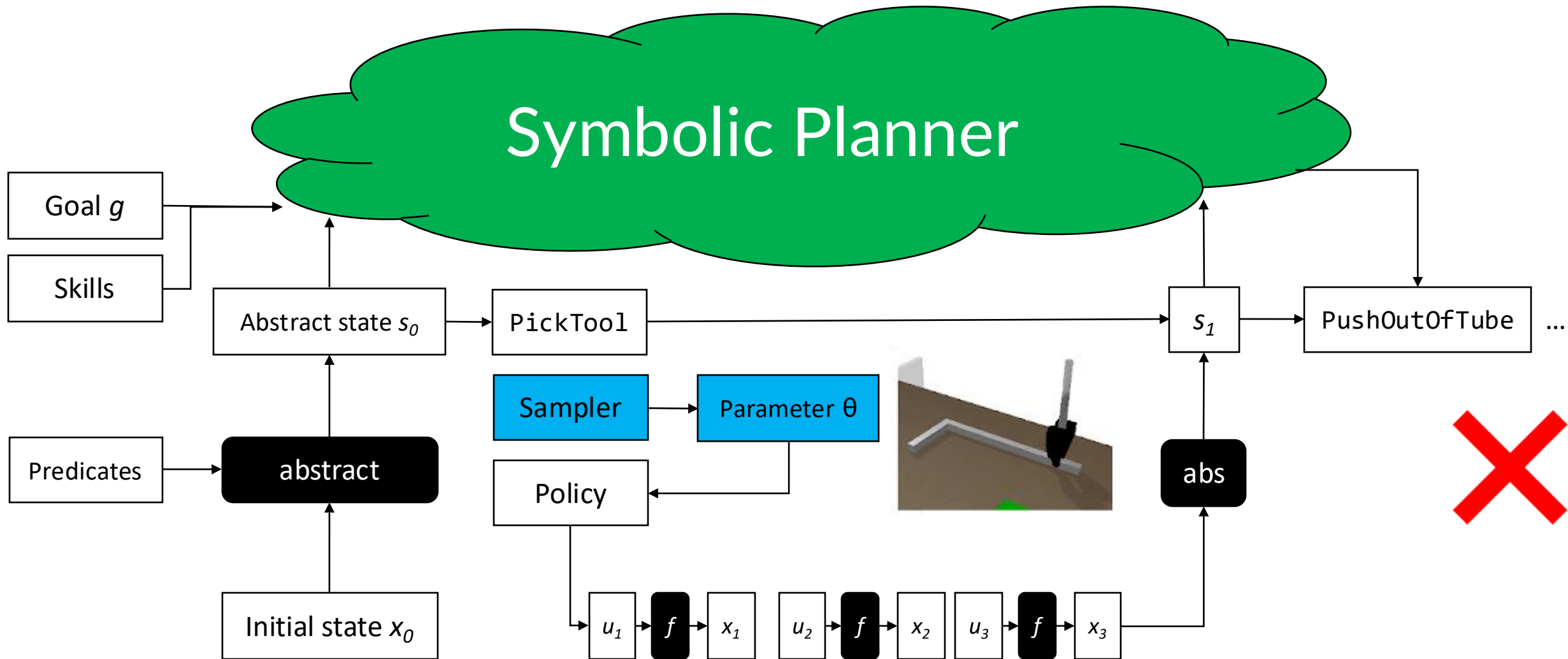
Preconditions: {GripperOpen()}

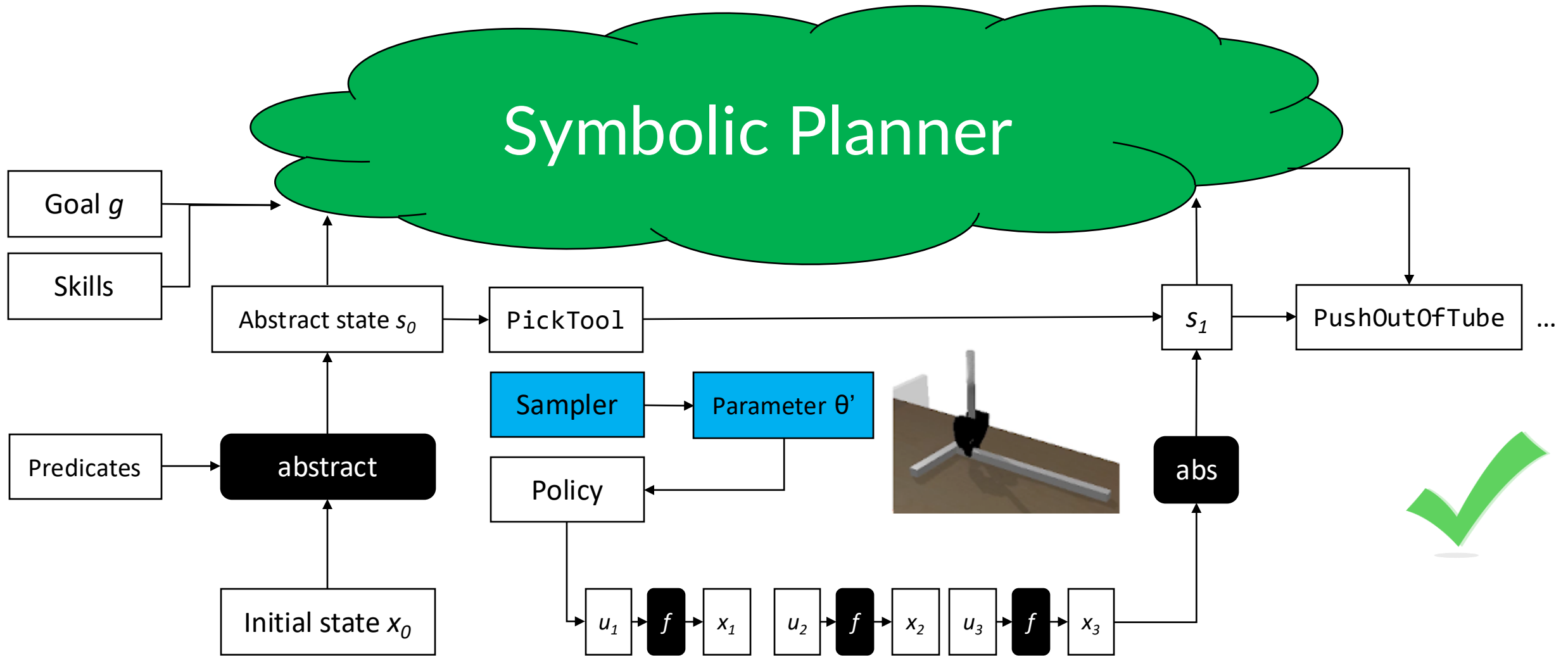
Add effects: {Holding(\mathbb{T} , )}

Delete effects: {GripperOpen()}

Different Parameters







Logic-Geometric Programming

Toussaint (2015)

Possible issues:

1. Optimizing in low-level state and action space remains hard for long-horizon problems
2. **We still may have “contract disputes”...**

The abstractions may be
pathological liars...

An abstract plan may not be
refinable at all!

Coffee Domain

Abstract Plan

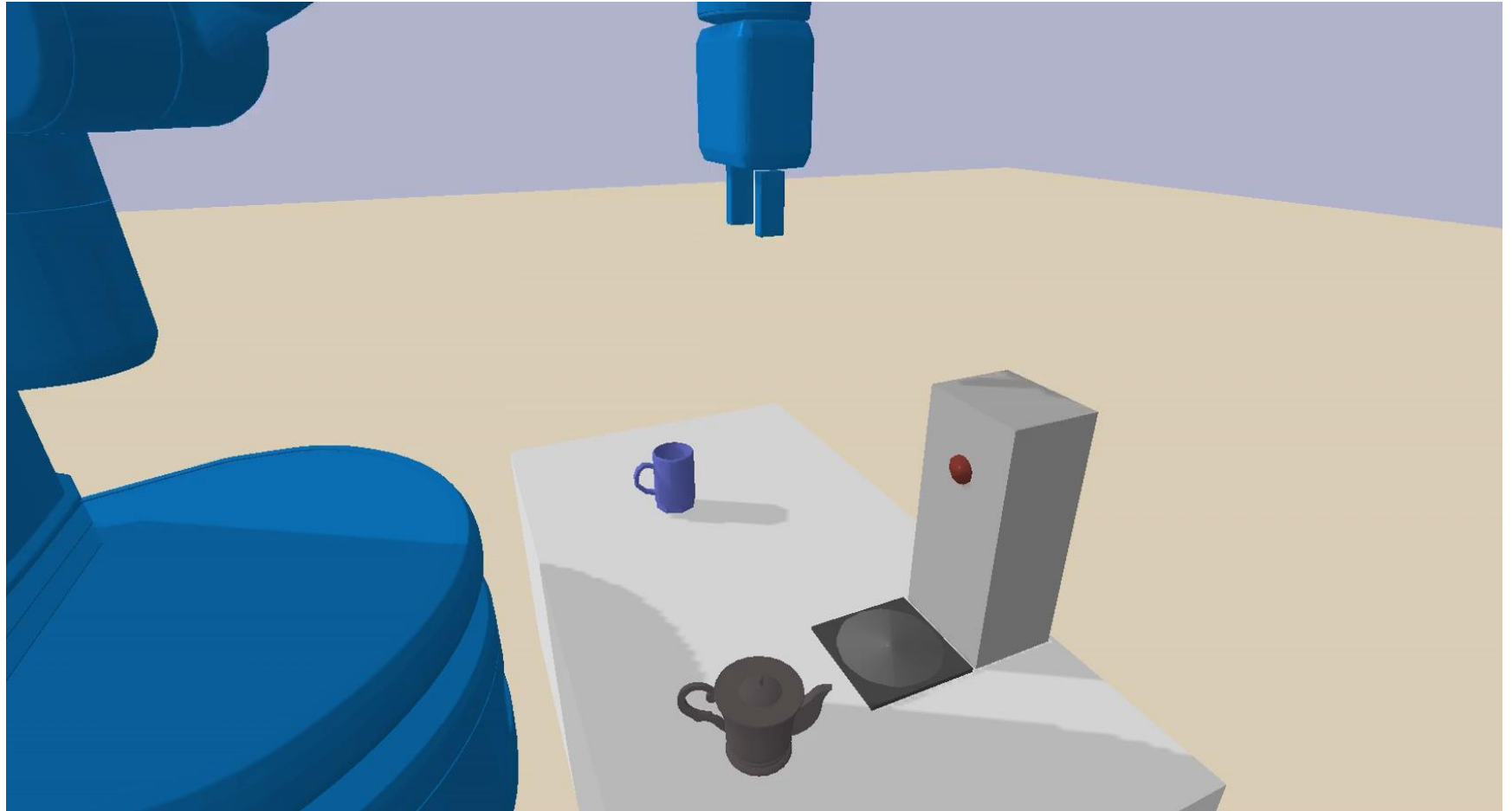
Grasp handle

Place on plate

Turn plate on

Pick up pot

Pour into cup



We need a different
abstract plan!

Coffee Domain

Abstract Plan



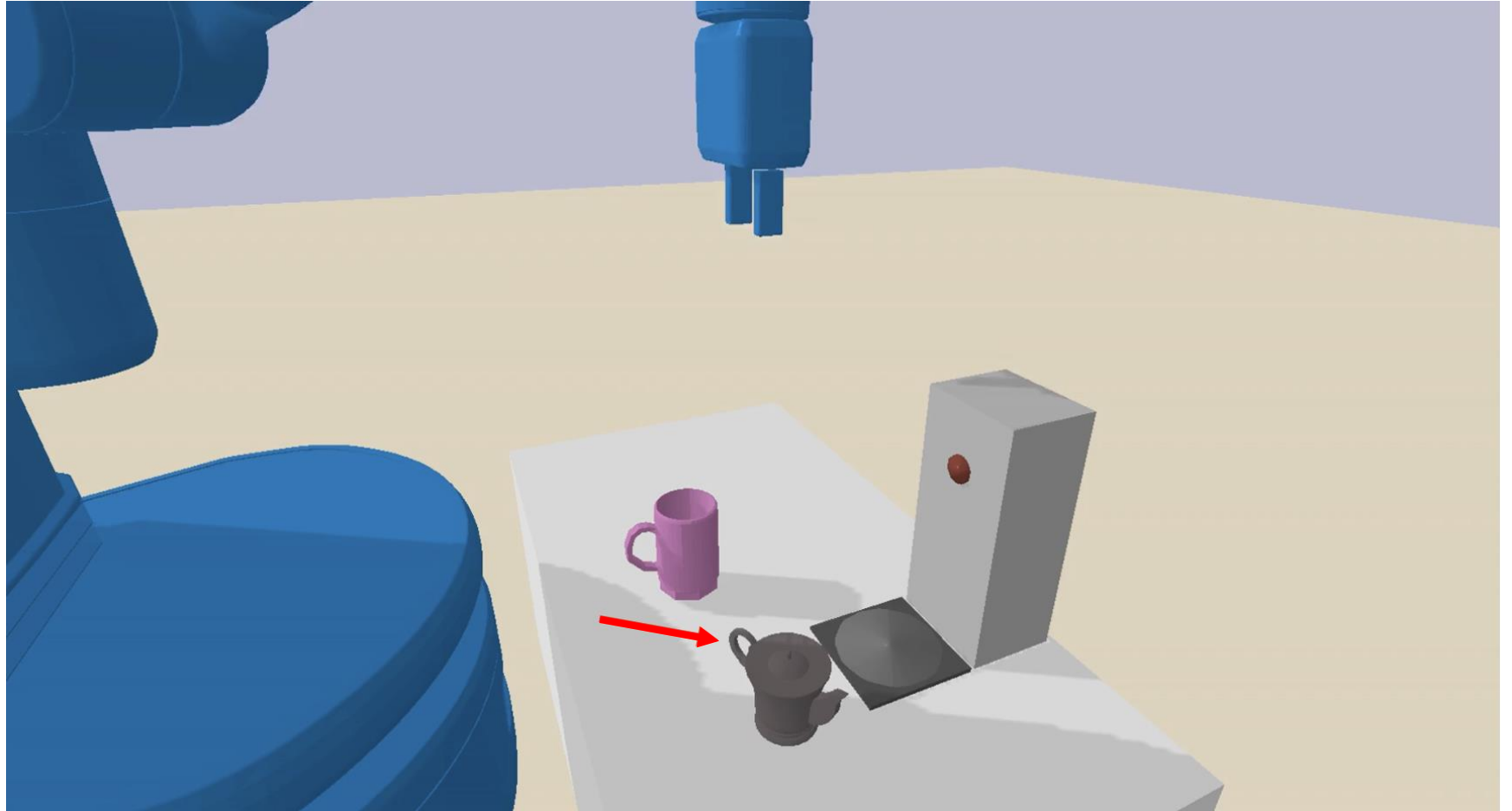
Grasp handle

Place on plate

Turn plate on

Pick up pot

Pour into cup



Coffee Domain

Abstract Plan

Rotate pot

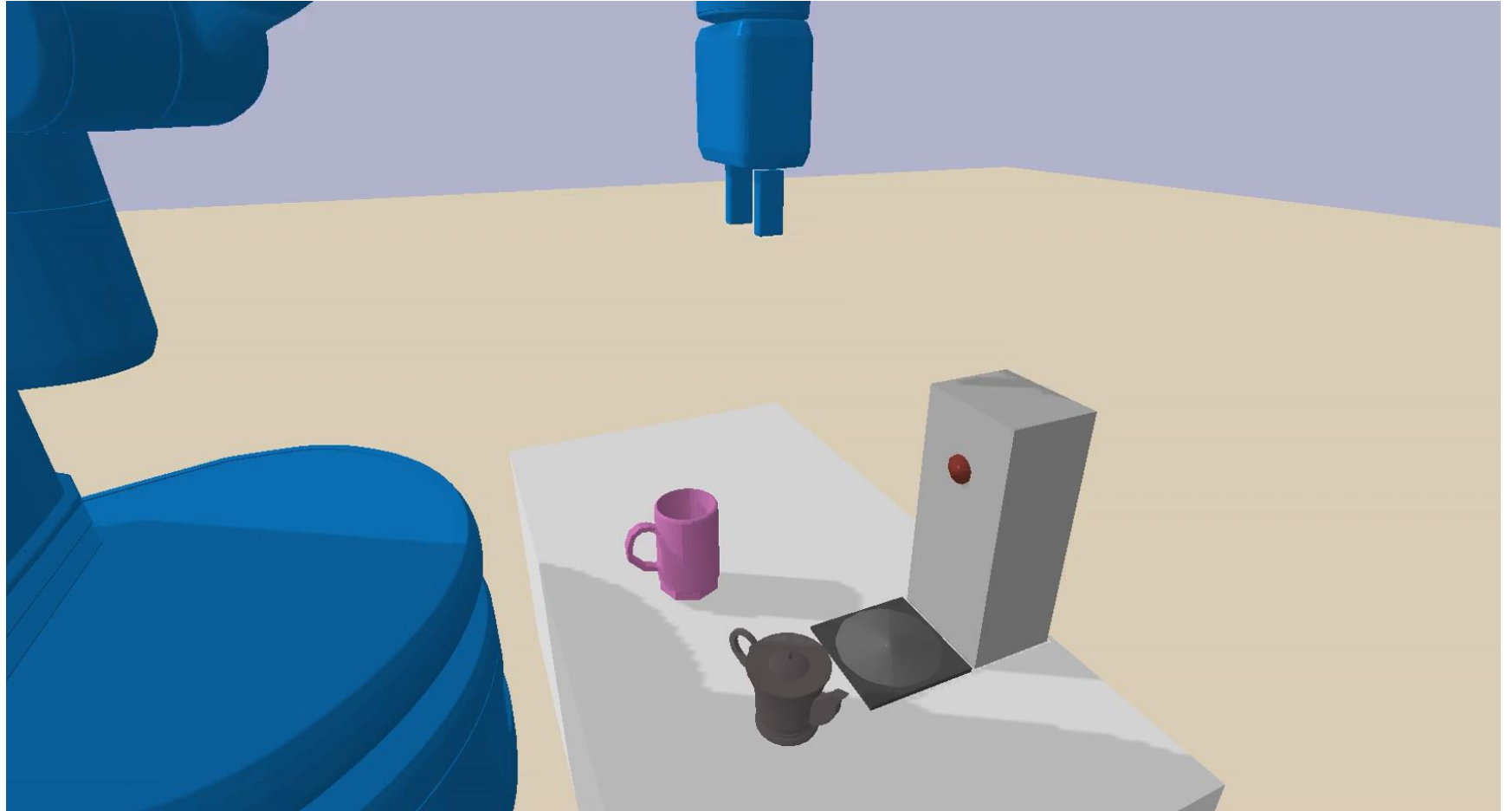
Grasp handle

Place on plate

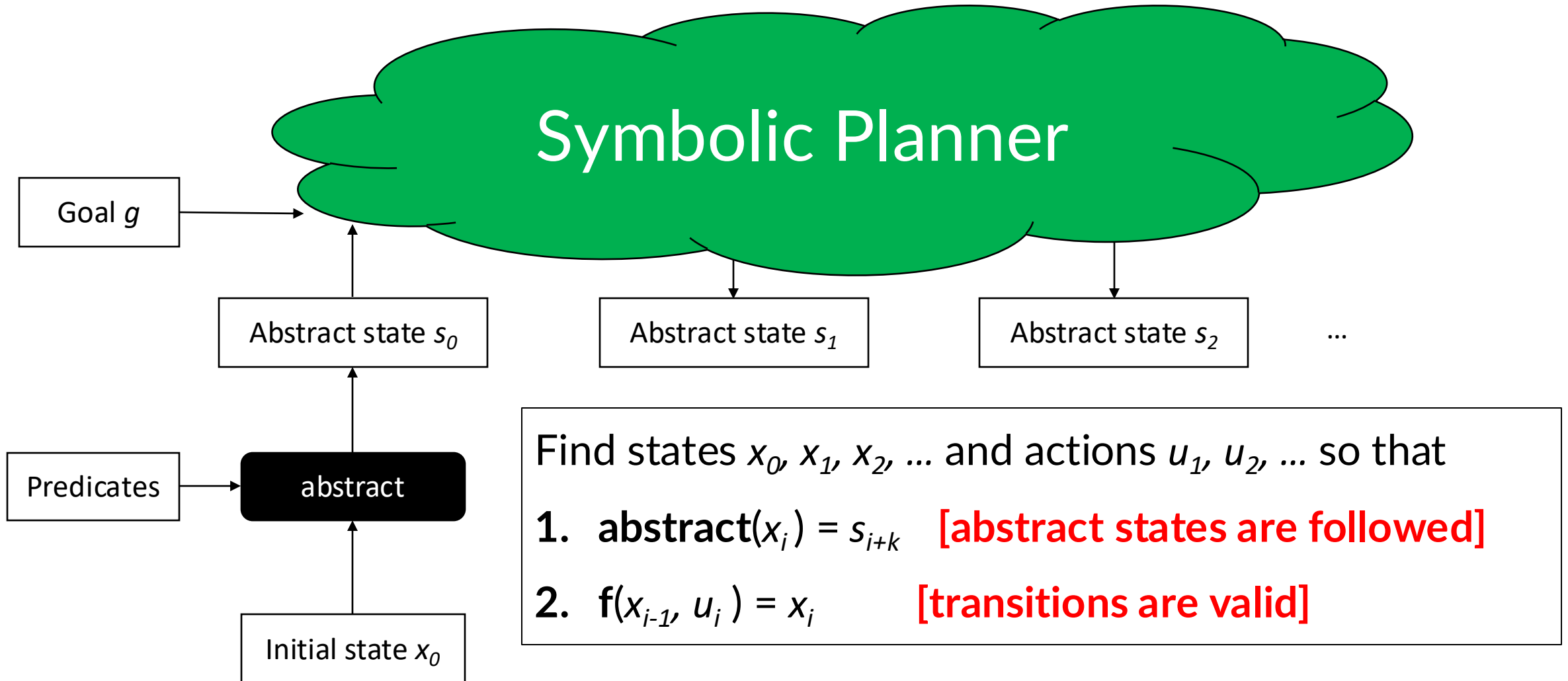
Turn plate on

Pick up pot

Pour into cup



One Remedy: Try Multiple Abstract Plans



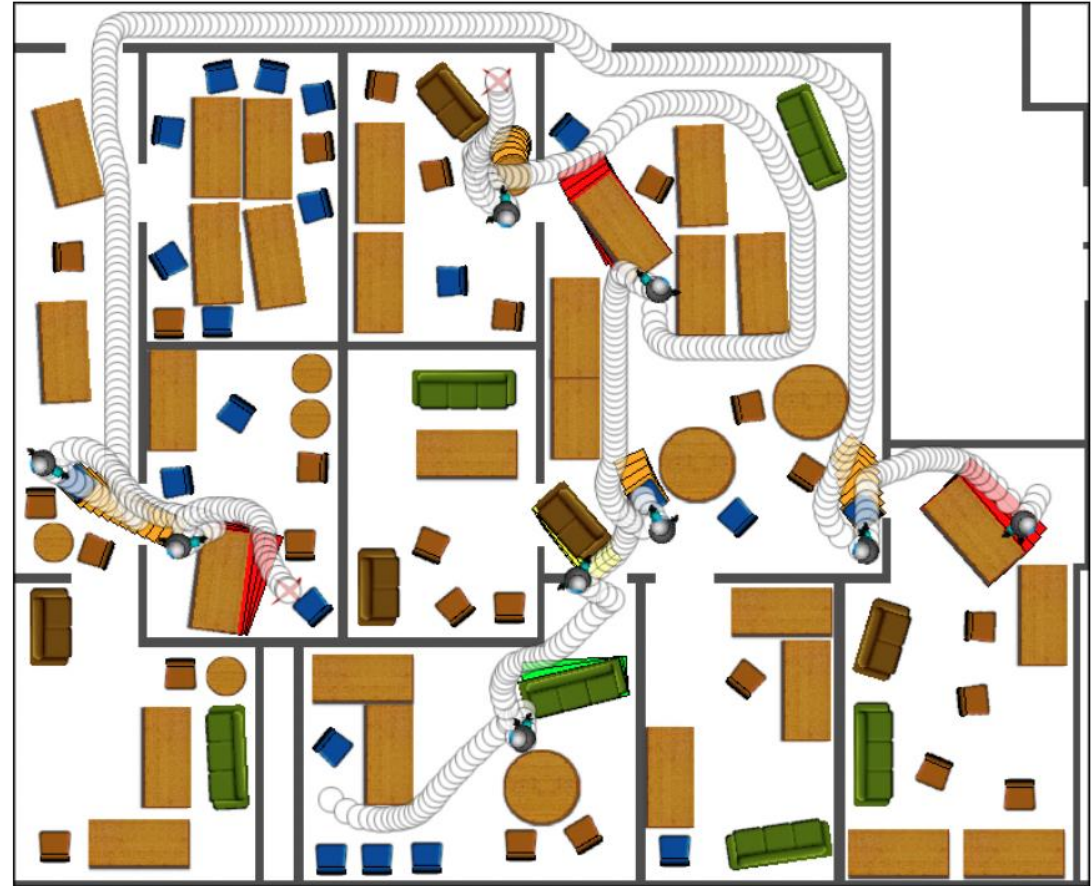
Better: Use Feedback from Refinement Failure to Influence Task Planning

Example: “Navigation Among Moveable Obstacles (NAMO)”

(Stilman & Kuffner 2004)

(Simplified explanation)

When a collision is encountered during refinement, make a plan to move the collided object out of the way first



Another Approach: Sample *then* Search

- Extends ideas from sample-based motion planning (RRT, PRM)
- Instead of sampling just robot configurations, sample...
 - Candidate grasps
 - Candidate positions of objects
 - ...
- Sample in a factored and conditional way
 - Example: conditioned on a future object position, sample a grasp
 - Conditioned on a grasp, sample a robot base position
 - Can sample “forward”, “backward”, or any-which-way in time
- See: PDDLStream (Garrett et al. 2018)

Summary: Task and Motion Planning (TAMP)

- Plan with state and action abstractions
- Use relational abstractions (e.g., PDDL) when possible
- Beware that the abstractions might be “liars”
 - TAMP is most interesting in this case!
- Use the abstractions as “guidance” for planning
- Closely related to hierarchical RL