Robot Task and Motion Planning using Domain-Independent Algorithms

Caelan Reed Garrett

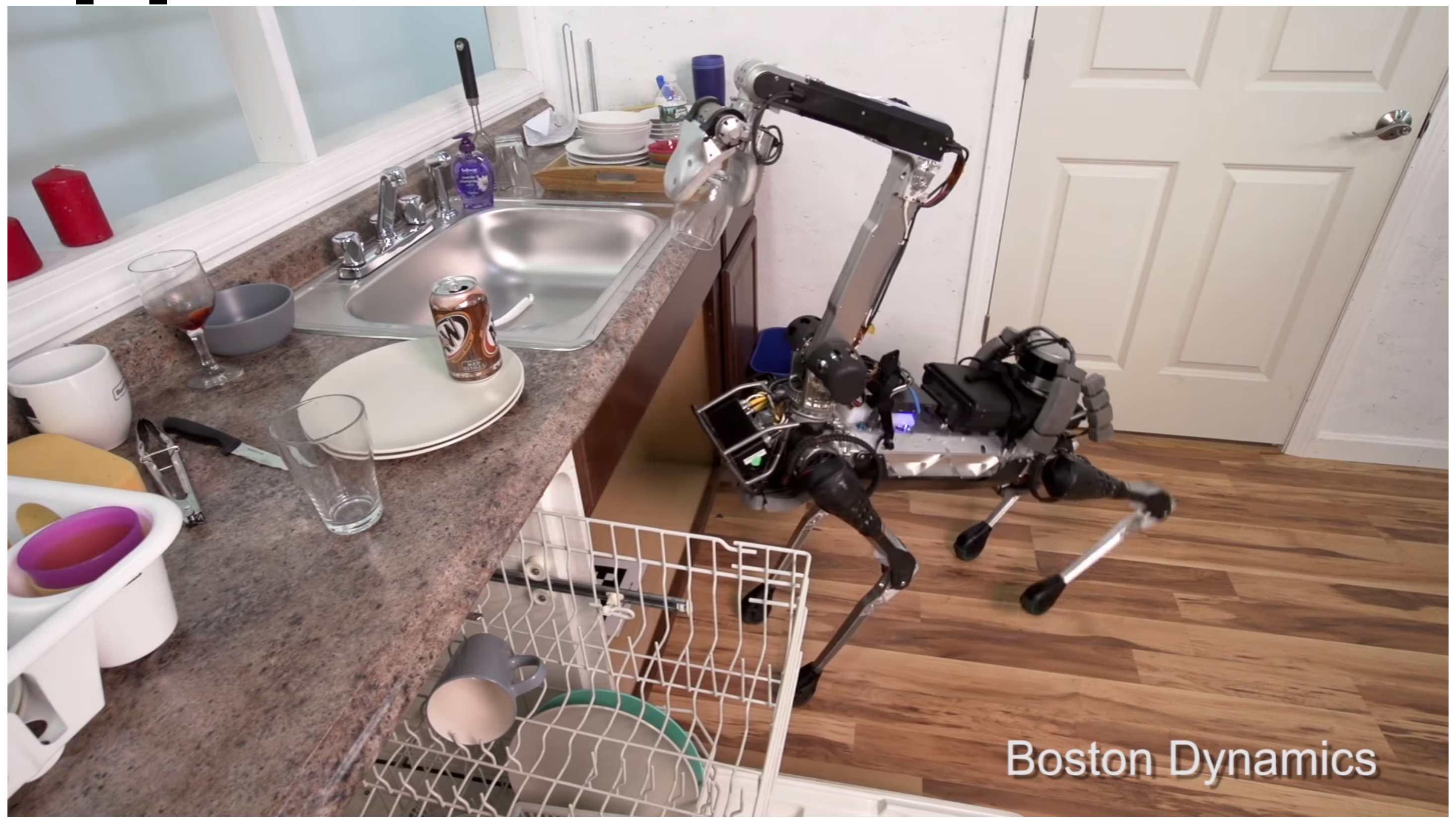
Advisors: Tomás Lozano-Pérez and Leslie Pack Kaelbling

04/19/2018 @ UNH

web.mit.edu/caelan/

Planning for Autonomous Robots

- Robot must select both **high-level** actions & **low-level** controls
- **Application areas:** semi-structured and human environments



Household

Warehouse fulfilment

Food service

Construction

Task and Motion Planning (TAMP)

- Plan in a **hybrid** space with many variables
  - **Discrete** and **continuous** variables & actions
- **Variables** - robot configuration, object poses, door joint positions, is-on, is-in-hand, is-holding-water, is-cooked, …
- **Actions** - move, pick, place, push, pull, pour, cook, …

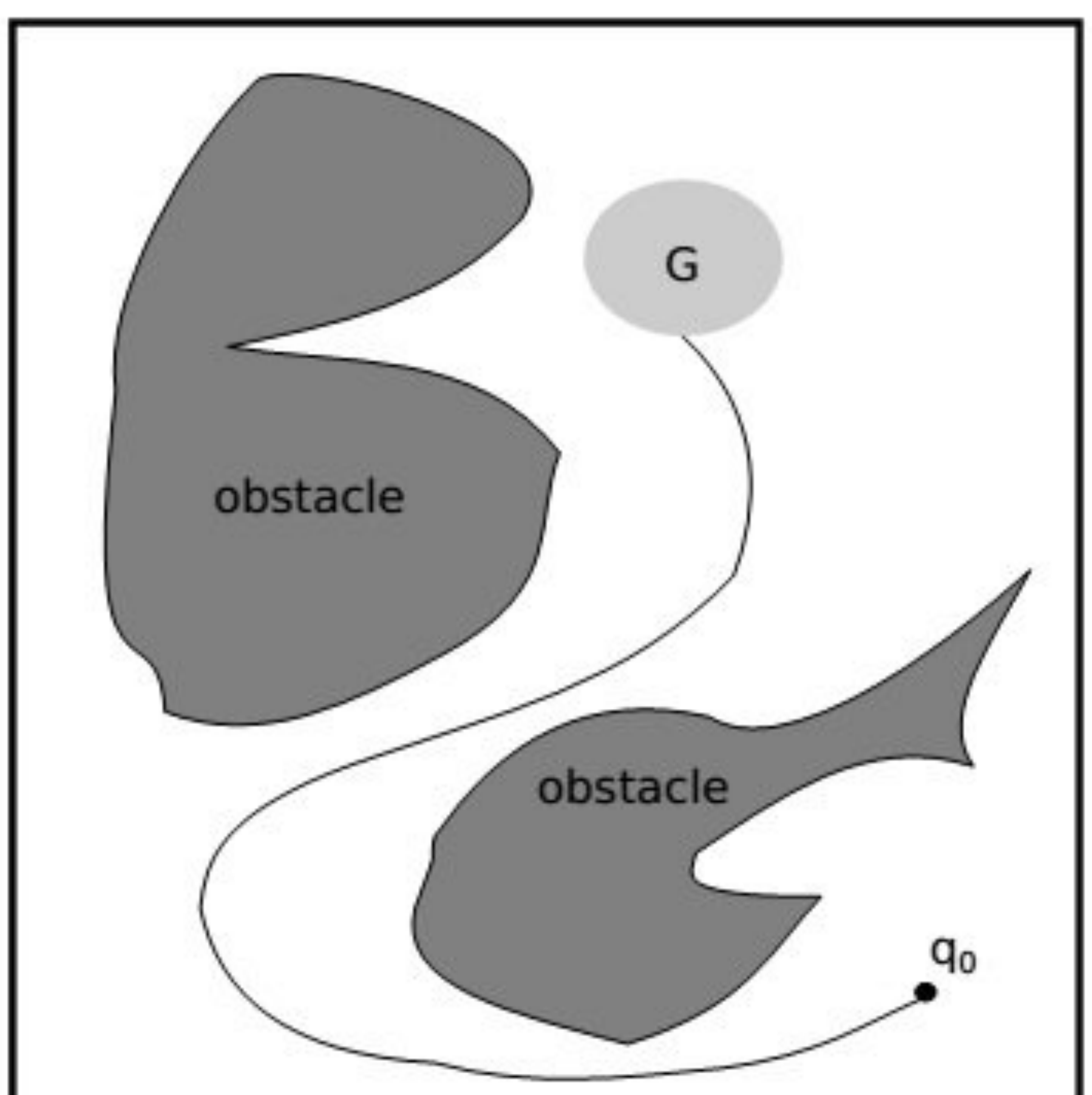
Cooking and Stacking

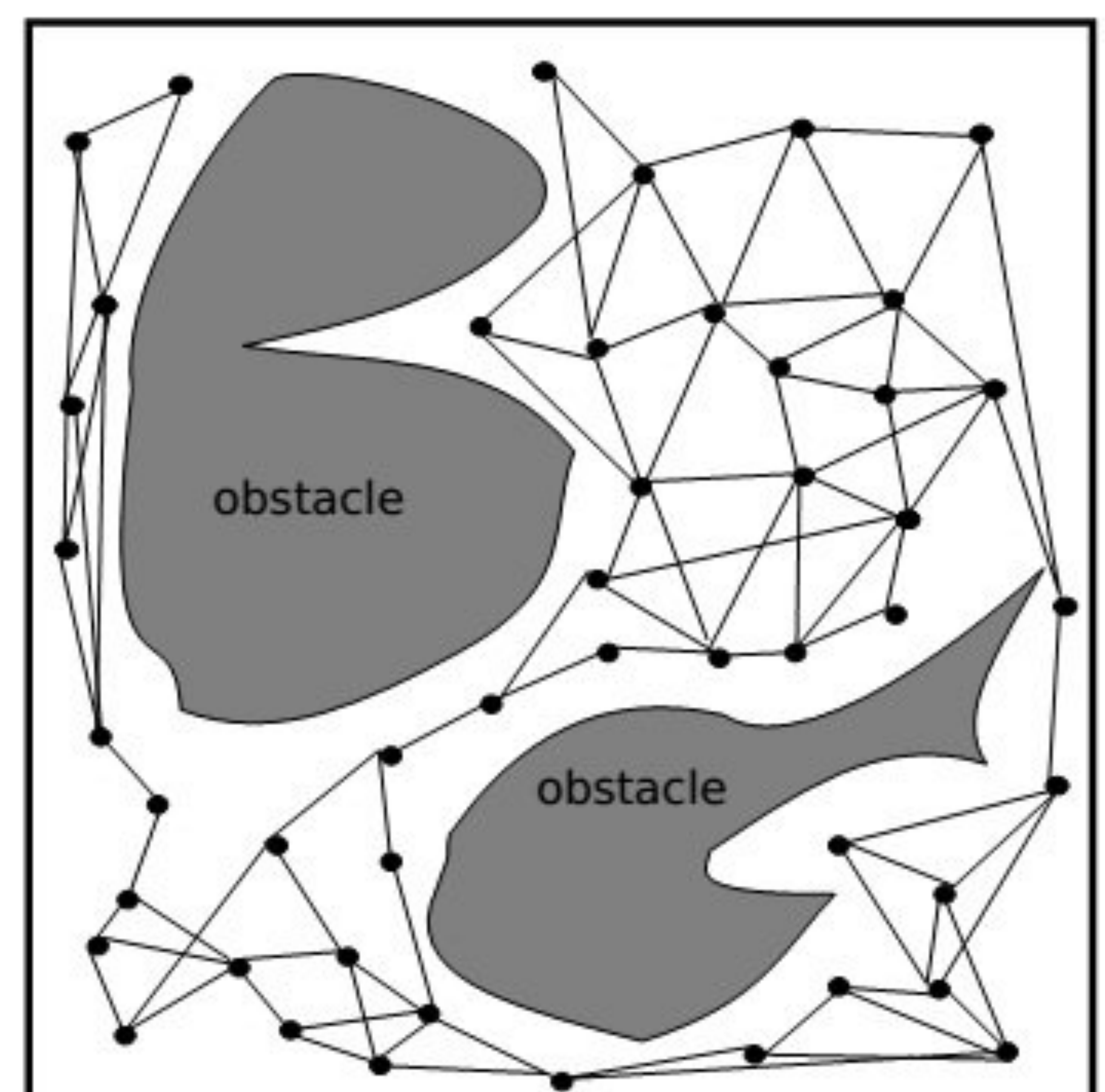
Cooking and Stacking

Motion Planning Background

- Plan in a **continuous** configuration space
- **Sampling-based** motion planning
  1. **Sample** robot configurations (randomly)
  2. **Connect** nearby configurations if collision-free path
  3. **Search** for a path within resulting graph

- PRM
- RRT
- RRT*

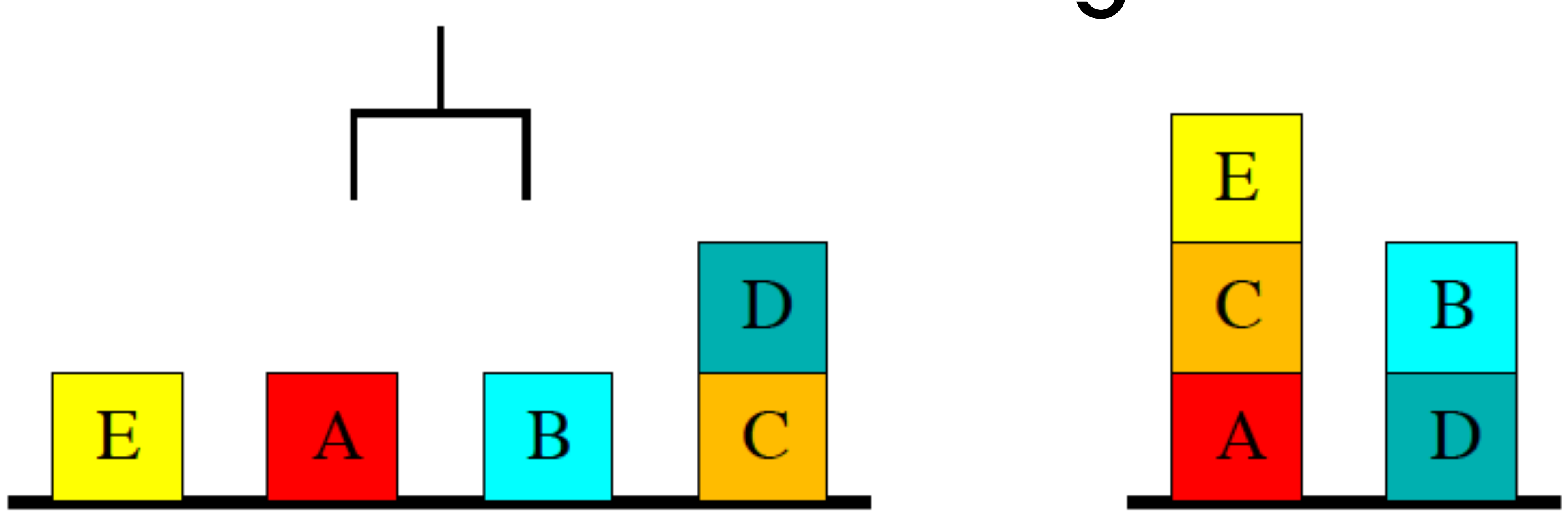




AI (Task) Planning Background

- Plan in a large **discrete** space with **many variables**
- **Planning languages:** STRIPS/PDDL
  - **Facts:** boolean state variables
  - **Parameterized** actions
    - **Preconditions** test validity
    - **Effects** change the state
- **Heuristic search** algorithms

```
(:action stack
  :parameters (?b1 ?b2)
  :precondition (and
    (Holding ?b1)
    (Clear ?b2))
  :effect (and
    (HandEmpty)
    (On ?b1 ?b2)
    (not (Holding ?b1))
    (not (Clear ?b2))))
```



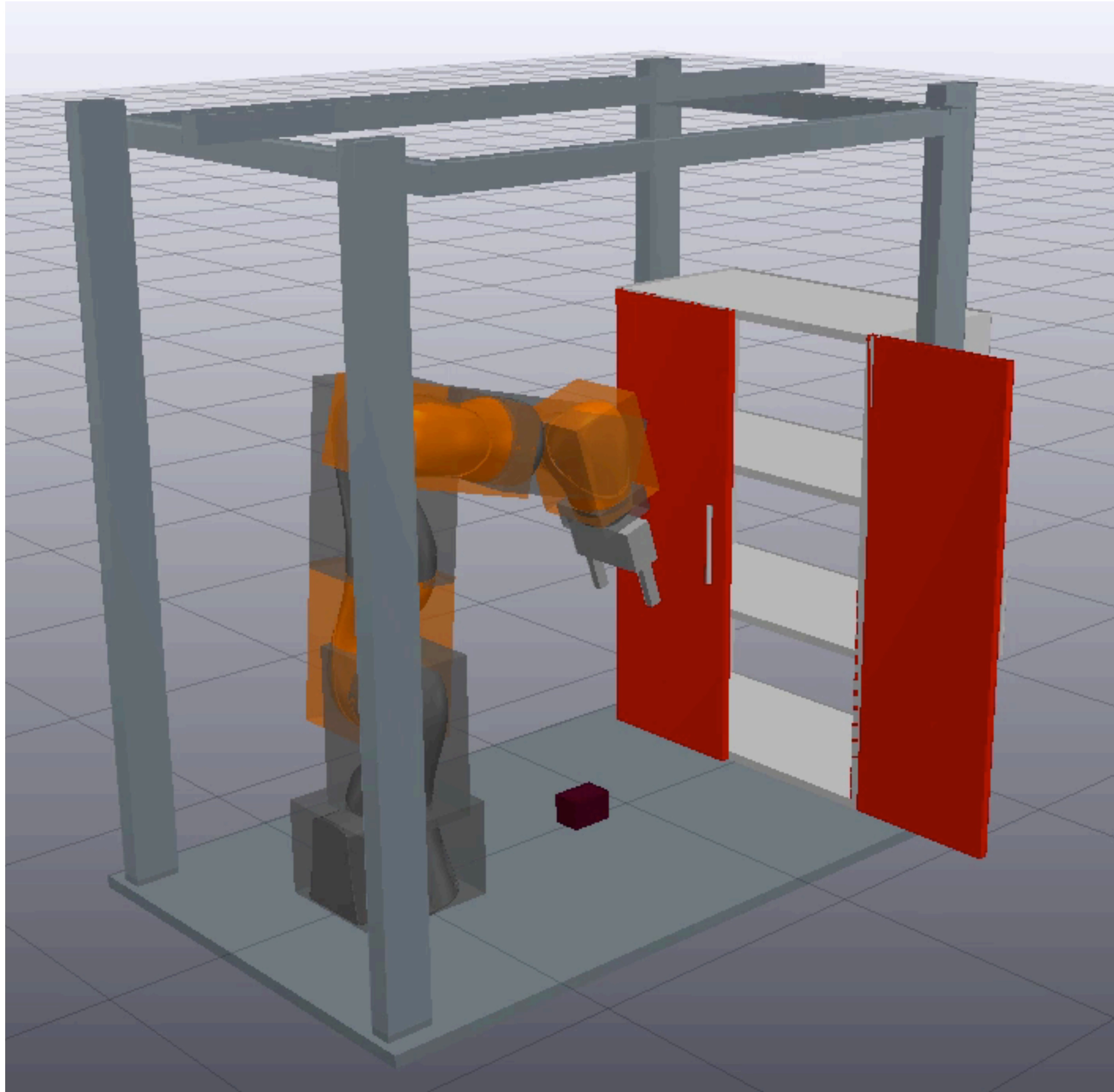
Initial State

Goal State

Geometric Constraints Affect Plan

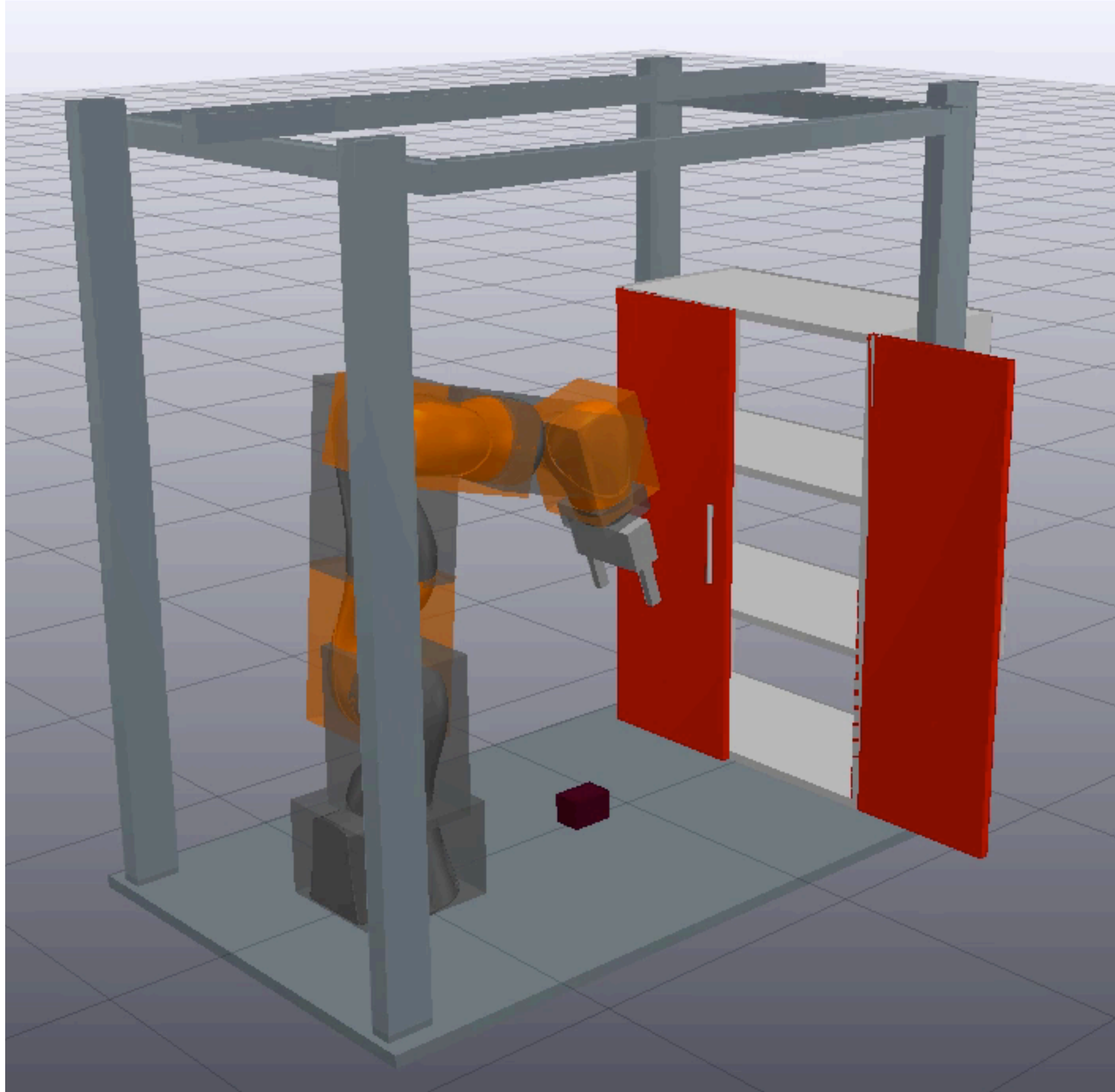
- **Inherits challenges** of both motion & AI planning
  - **High-dimensional, continuous** state-spaces
  - Discretized state-space **grows combinatorially**
  - **Long horizons**

- Continuous constraints **limit high-level strategies**
  - Kinematic reachability
  - Joint limits & collisions
  - Visibility
  - Stability & stiffness



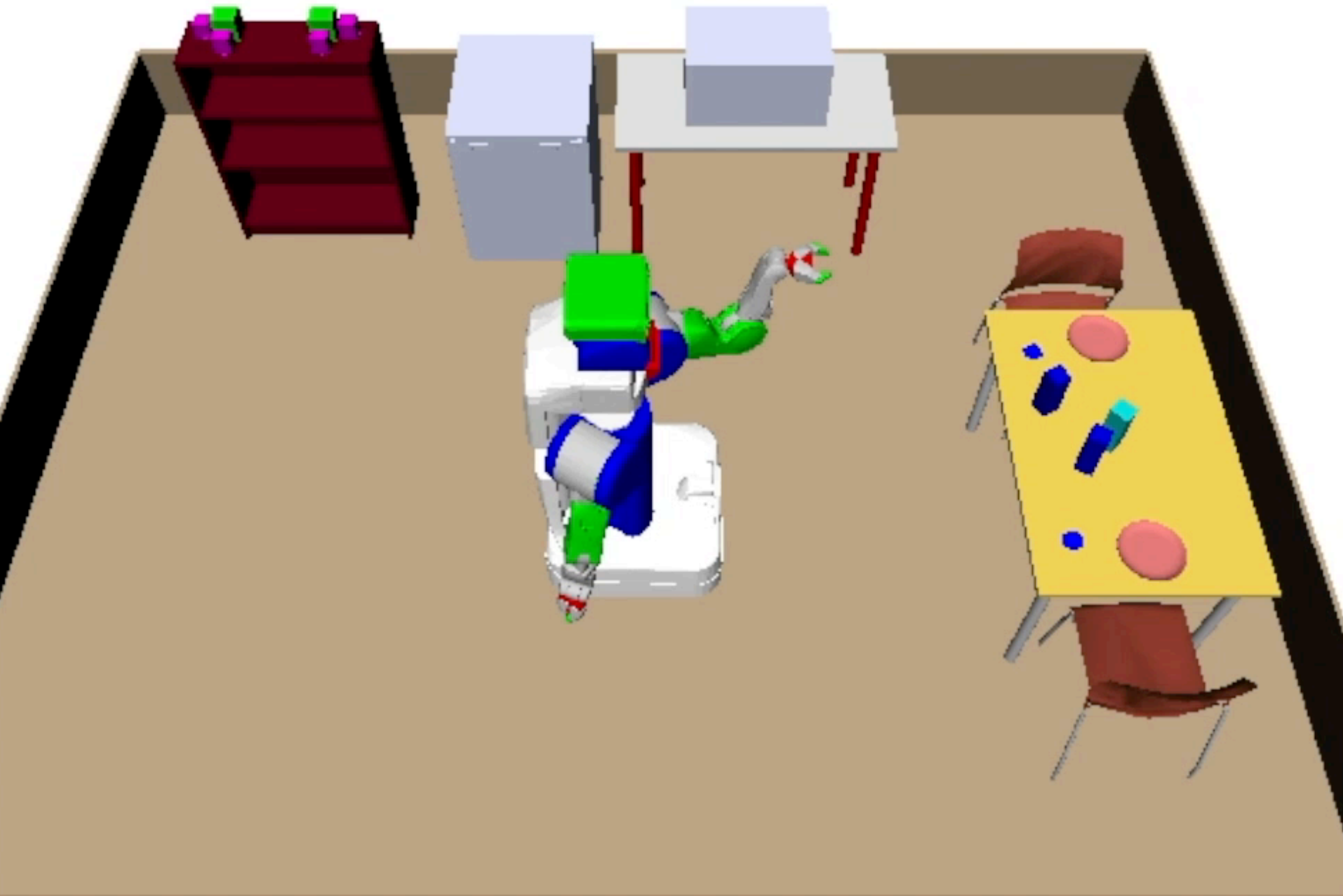
Geometric Constraints Affect Plan

- **Inherits challenges** of both motion & AI planning
  - **High-dimensional, continuous** state-spaces
  - Discretized state-space **grows combinatorially**
  - **Long horizons**

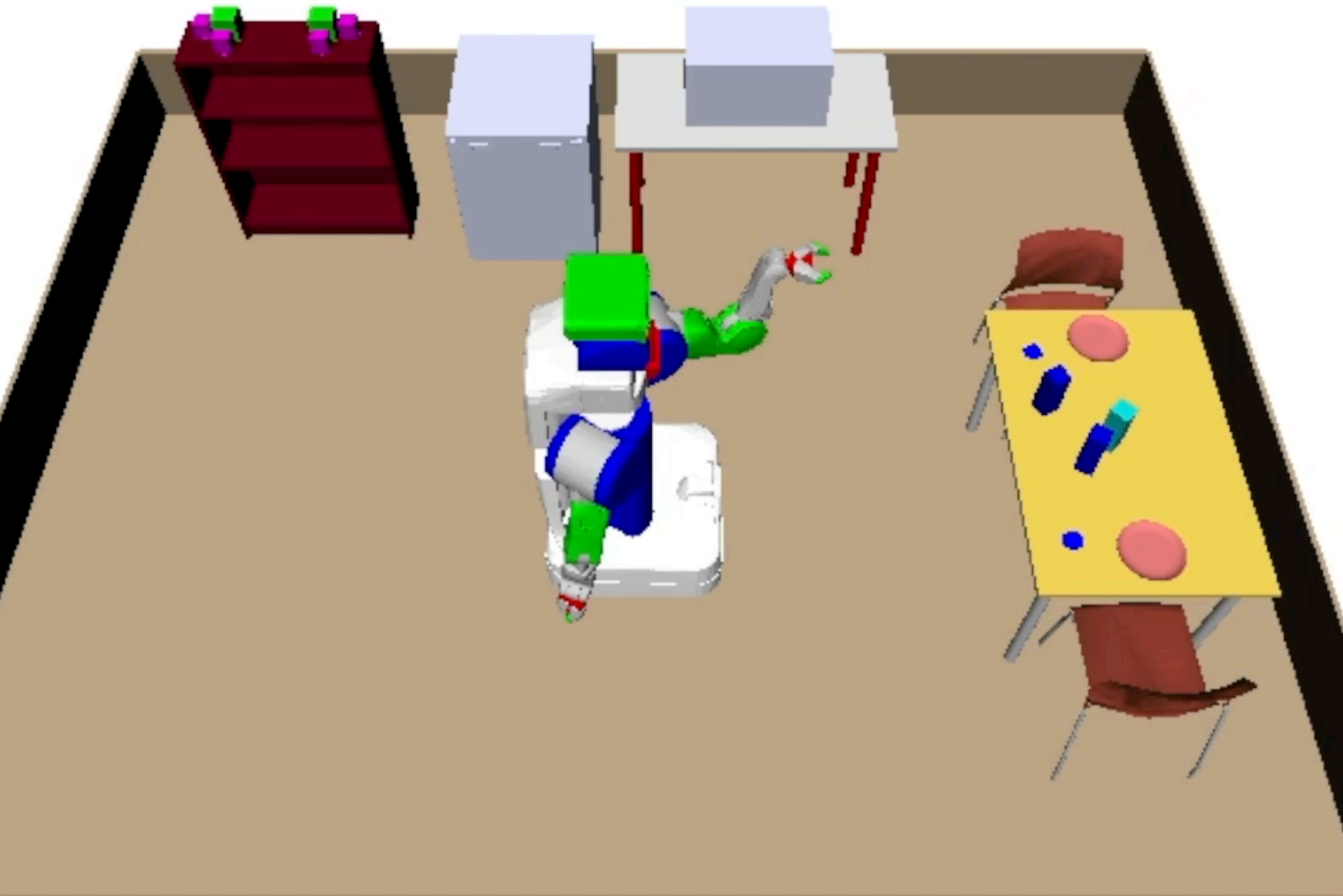


- Continuous constraints **limit high-level strategies**
  - Kinematic reachability
  - Joint limits & collisions
  - Visibility
  - Stability & stiffness

64 Continuous &10 Discrete Variables



64 Continuous &10 Discrete Variables



Prior Work

- **Multi-Modal Motion Planning -** *Alami et al., Siméon et al., Hauser and Latombe, Barry et al., Vega-Brown and Roy*
  - <u>Inefficient in high-dimensional state-spaces</u>

Prior Work

- **Multi-Modal Motion Planning -** *Alami et al., Siméon et al., Hauser and Latombe, Barry et al., Vega-Brown and Roy*
  - Inefficient in high-dimensional state-spaces
- **Semantic Attachments -** *Dornhege et al., Erdem et al., Dantam et al.*
  - Assume an a priori discretization

Prior Work

- **Multi-Modal Motion Planning -** *Alami et al., Siméon et al., Hauser and Latombe, Barry et al., Vega-Brown and Roy*
  - <u>Inefficient in high-dimensional state-spaces</u>
- **Semantic Attachments -** *Dornhege et al., Erdem et al., Dantam et al.*
  - <u>Assume an a priori discretization</u>
- **Task & Motion Interface -** *Cambon et al., Kaelbling and Lozano-Pérez, Lagriffoul et al., Srivastava et al., Toussaint*
  - <u>Inflexible to new domains</u>

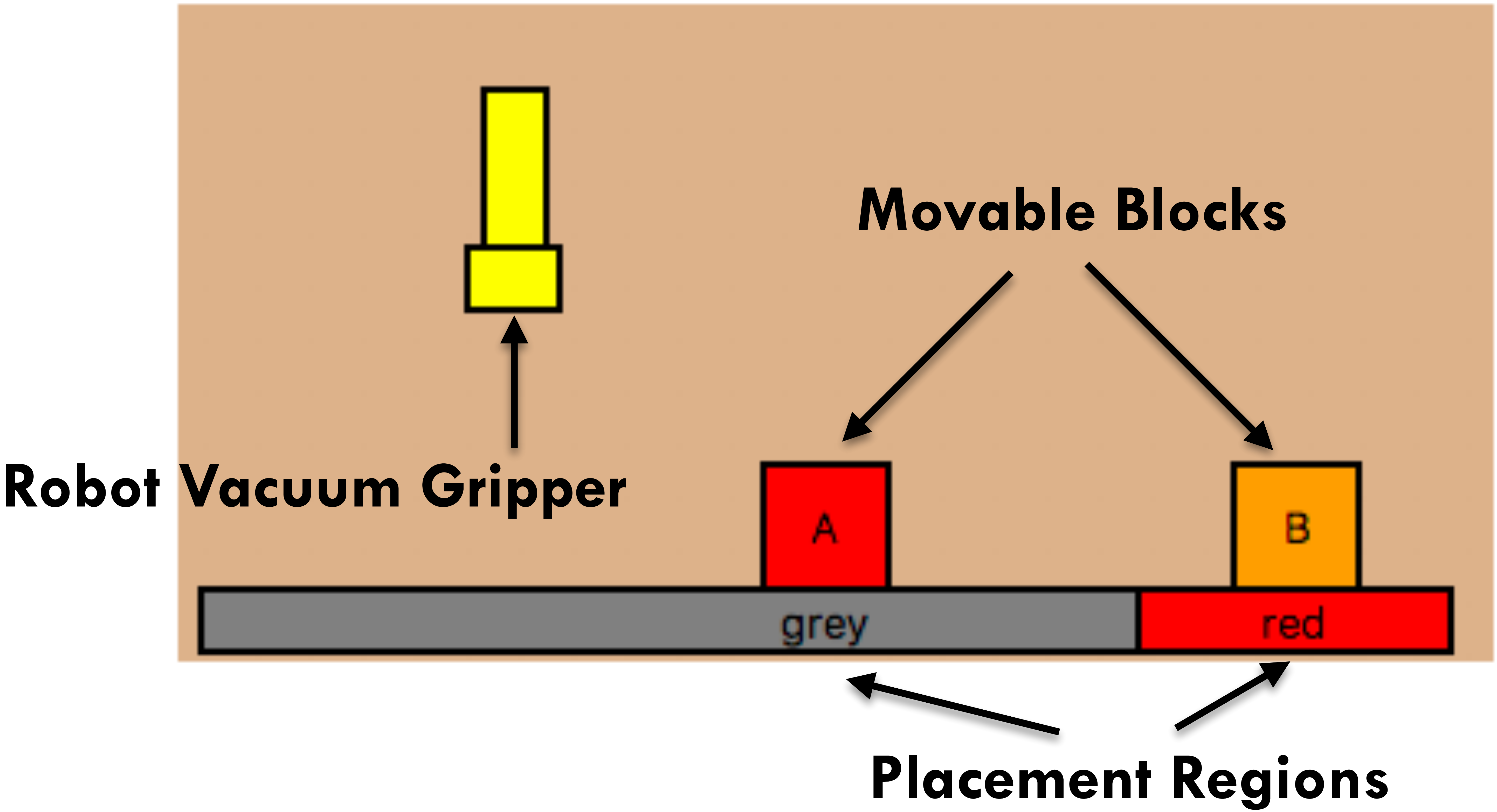
Our Approach

- Extends Planning Domain Description Language (**PDDL**)
  - Modular & **domain-independent**
- Enables the inclusion of **sampling procedures**
  - Can encode domains with **infinitely-many** actions

- Admits efficient, generic **algorithms**
  - Samplers are **blackbox inputs**
    - Software respects this abstraction
  - Algorithms solve a **sequence of finite** PDDL problems
    - Leverage fast **AI planners** as search subroutines

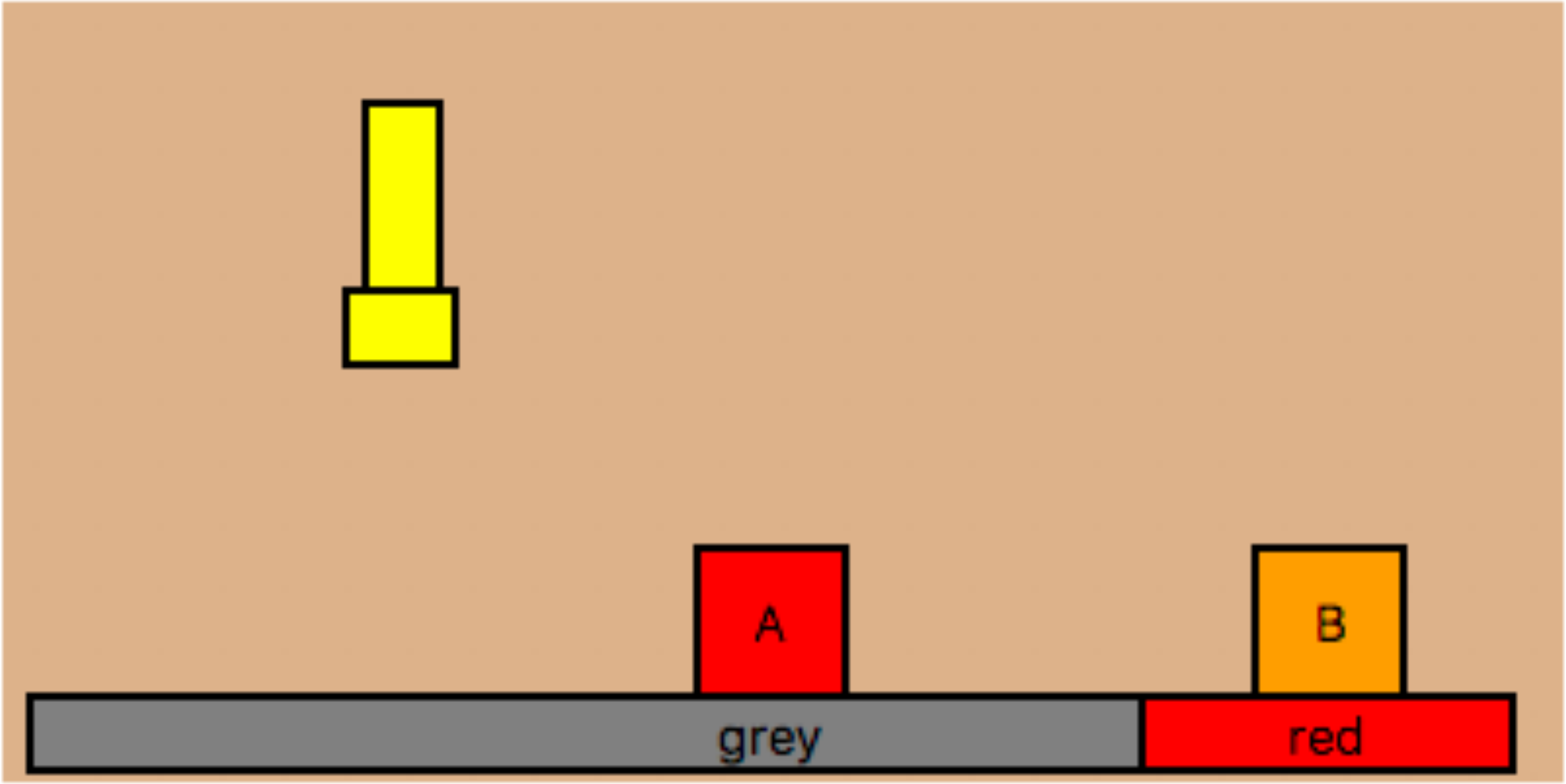
2D Pick-and-Place Example

- **Goal**: block **A** within the **red** region
- Robot and block poses are continuous (x, y) pairs
- Block **B** obstructs the placement of **A**



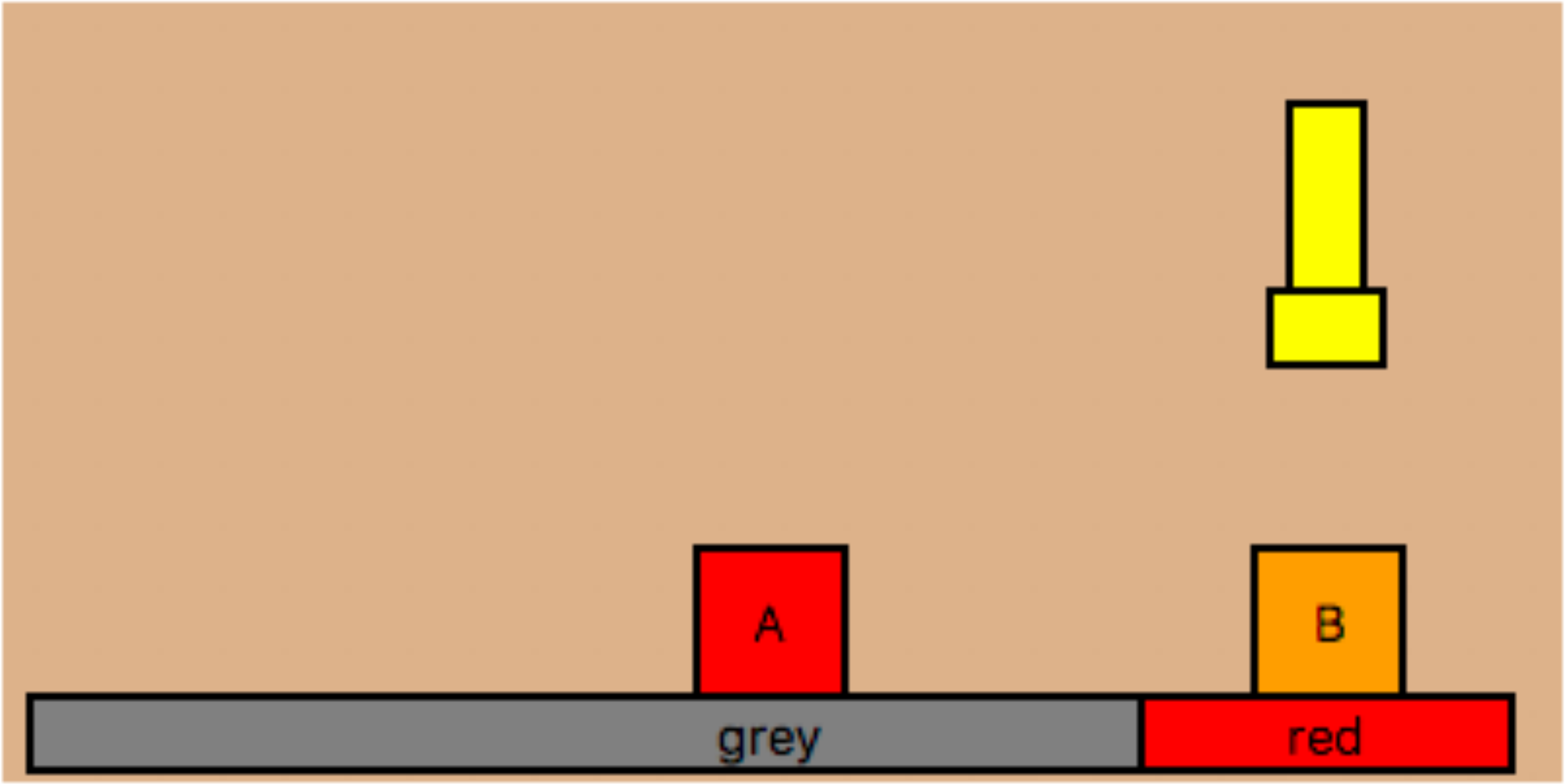
2D Pick-and-Place Solution

- One (of infinitely many) possible solutions
  - move, pick **B**, move, place **B**, move, pick **A**, move, place **A**



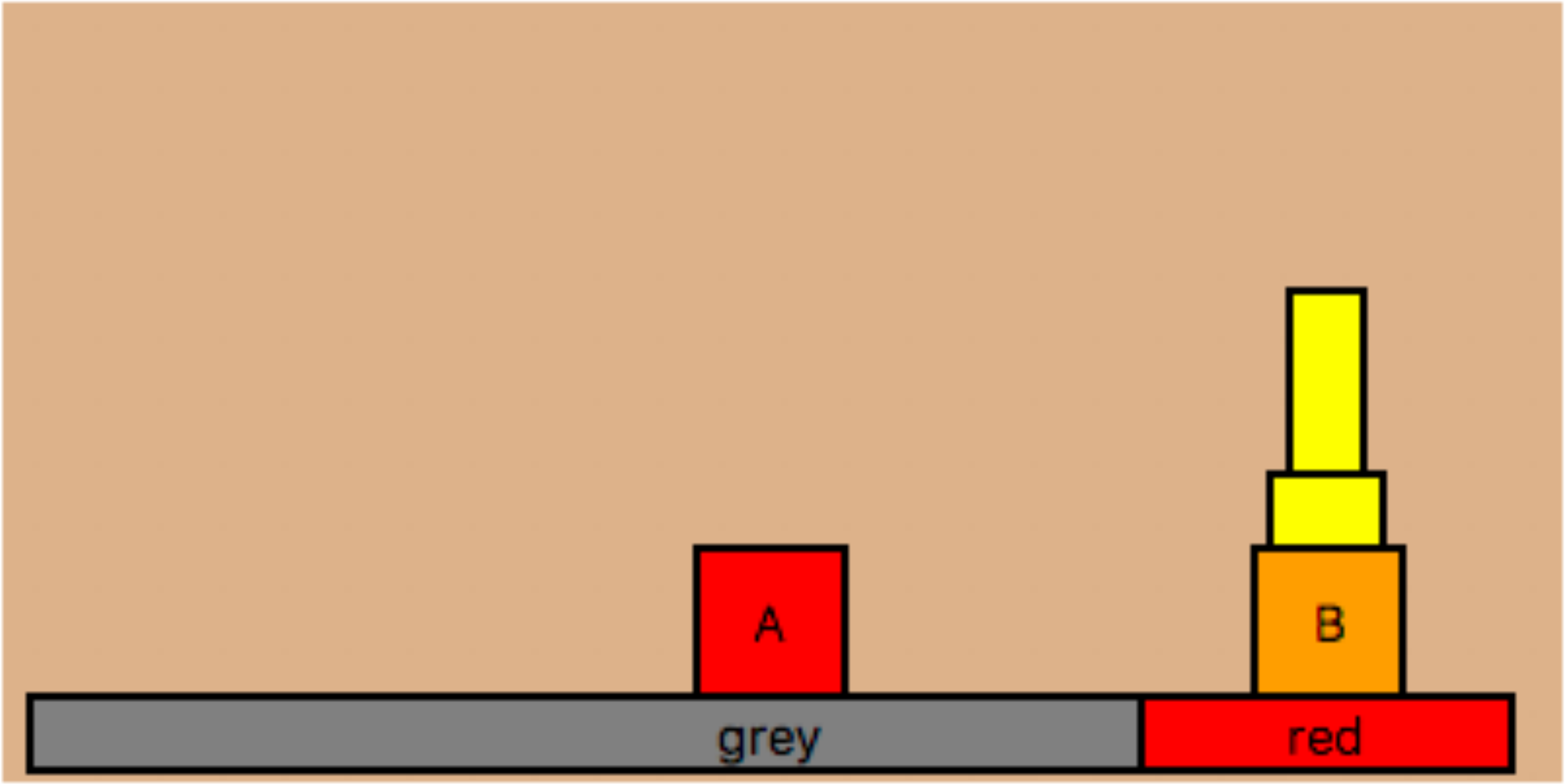
2D Pick-and-Place Solution

- One (of infinitely many) possible solutions
  - move, pick **B**, move, place **B**,
    move, pick **A**, move, place **A**



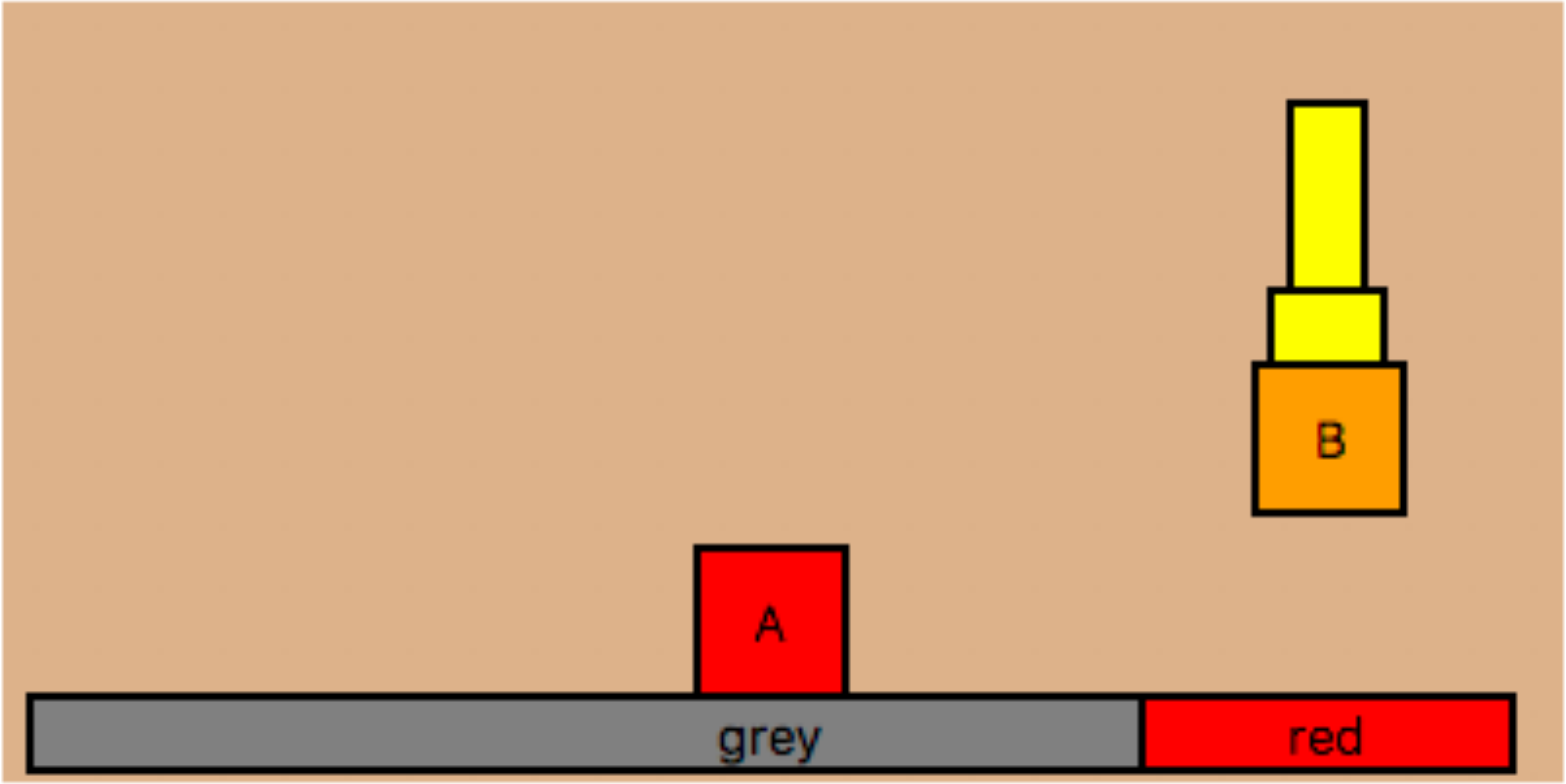
2D Pick-and-Place Solution

- One (of infinitely many) possible solutions
  - move, pick **B**, move, place **B**, move, pick **A**, move, place **A**



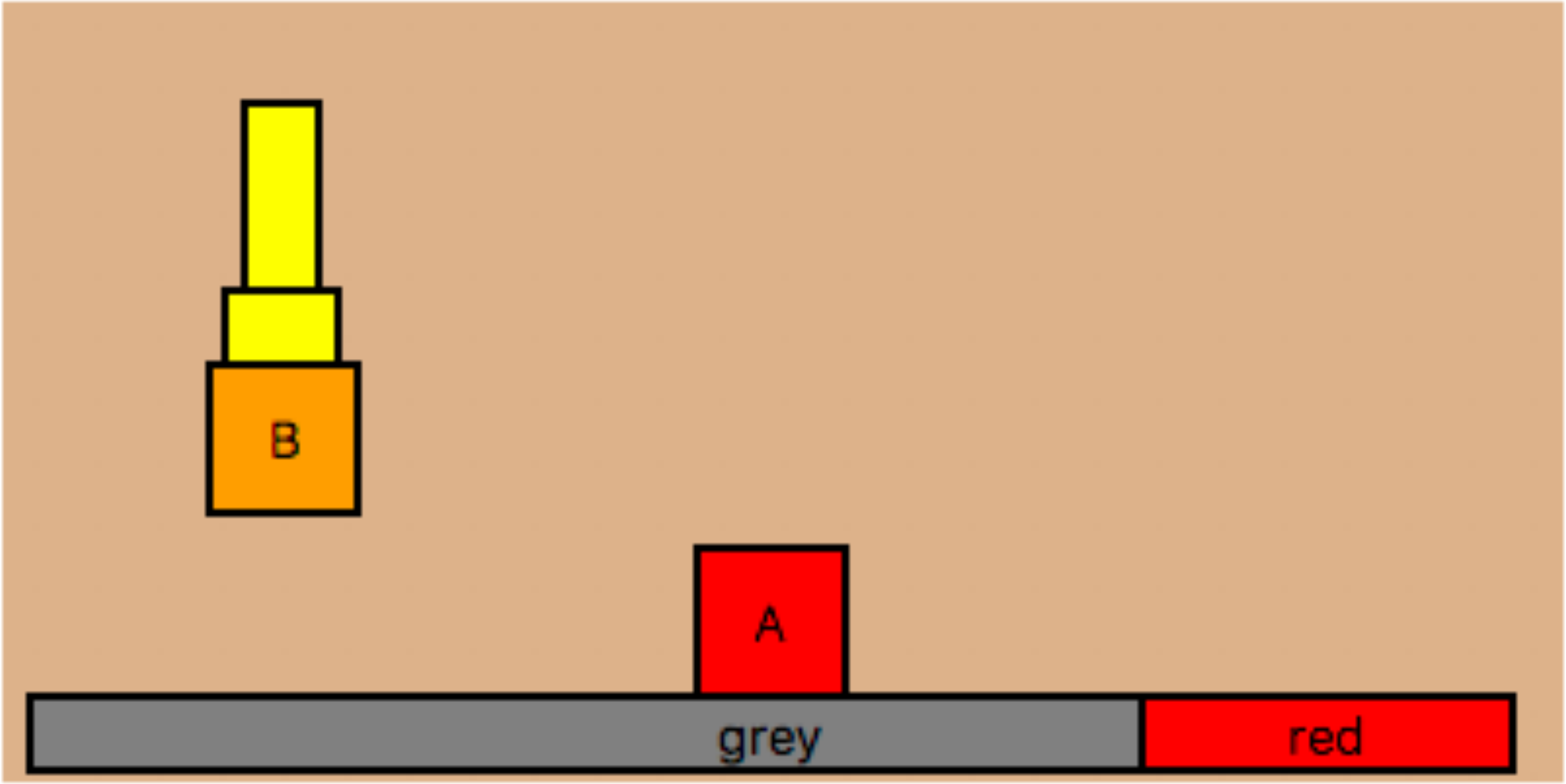
2D Pick-and-Place Solution

- One (of infinitely many) possible solutions
  - move, pick **B**, move, place **B**, move, pick **A**, move, place **A**



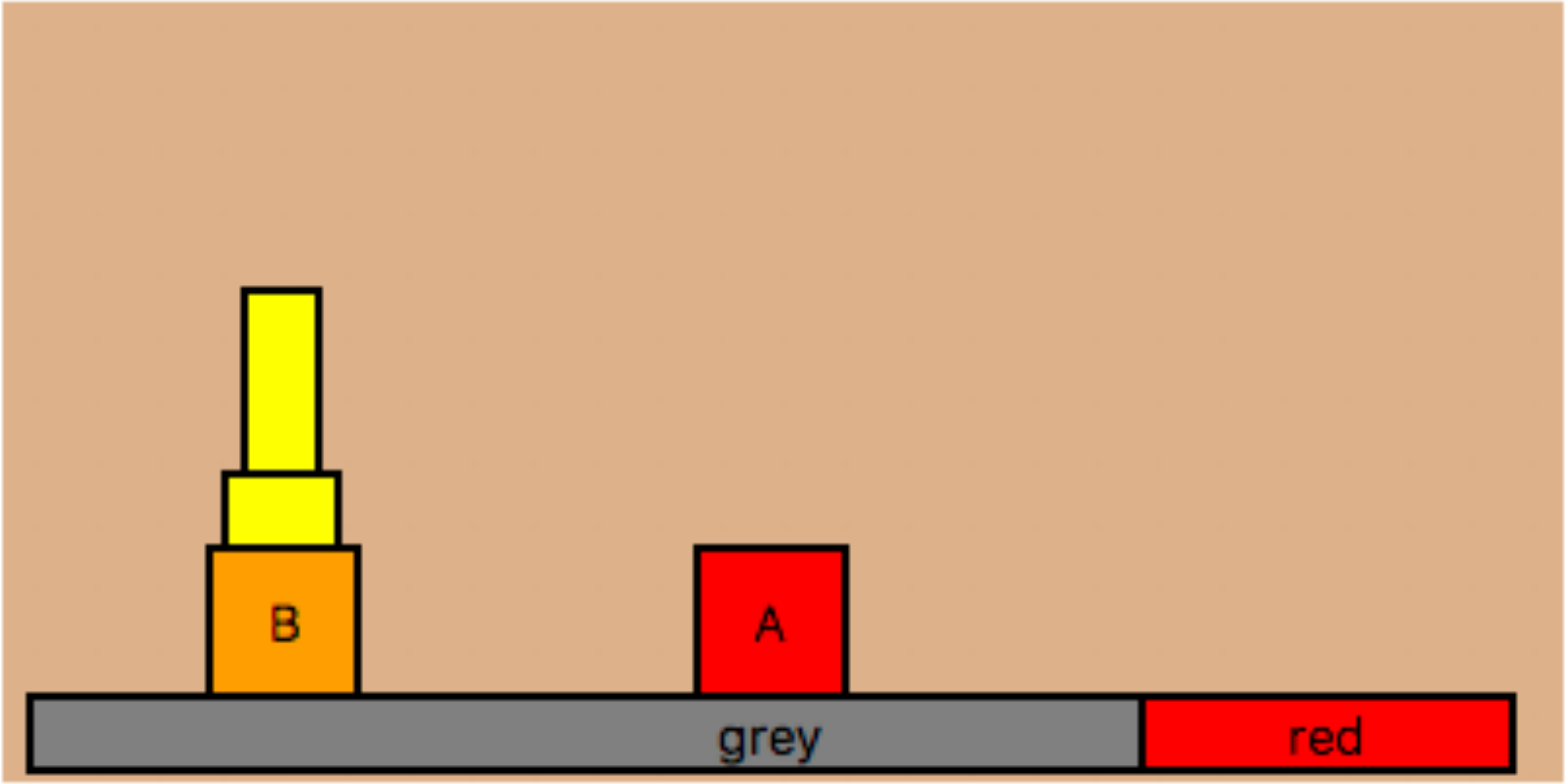
2D Pick-and-Place Solution

- One (of infinitely many) possible solutions
  - move, pick **B**, move, place **B**, move, pick **A**, move, place **A**



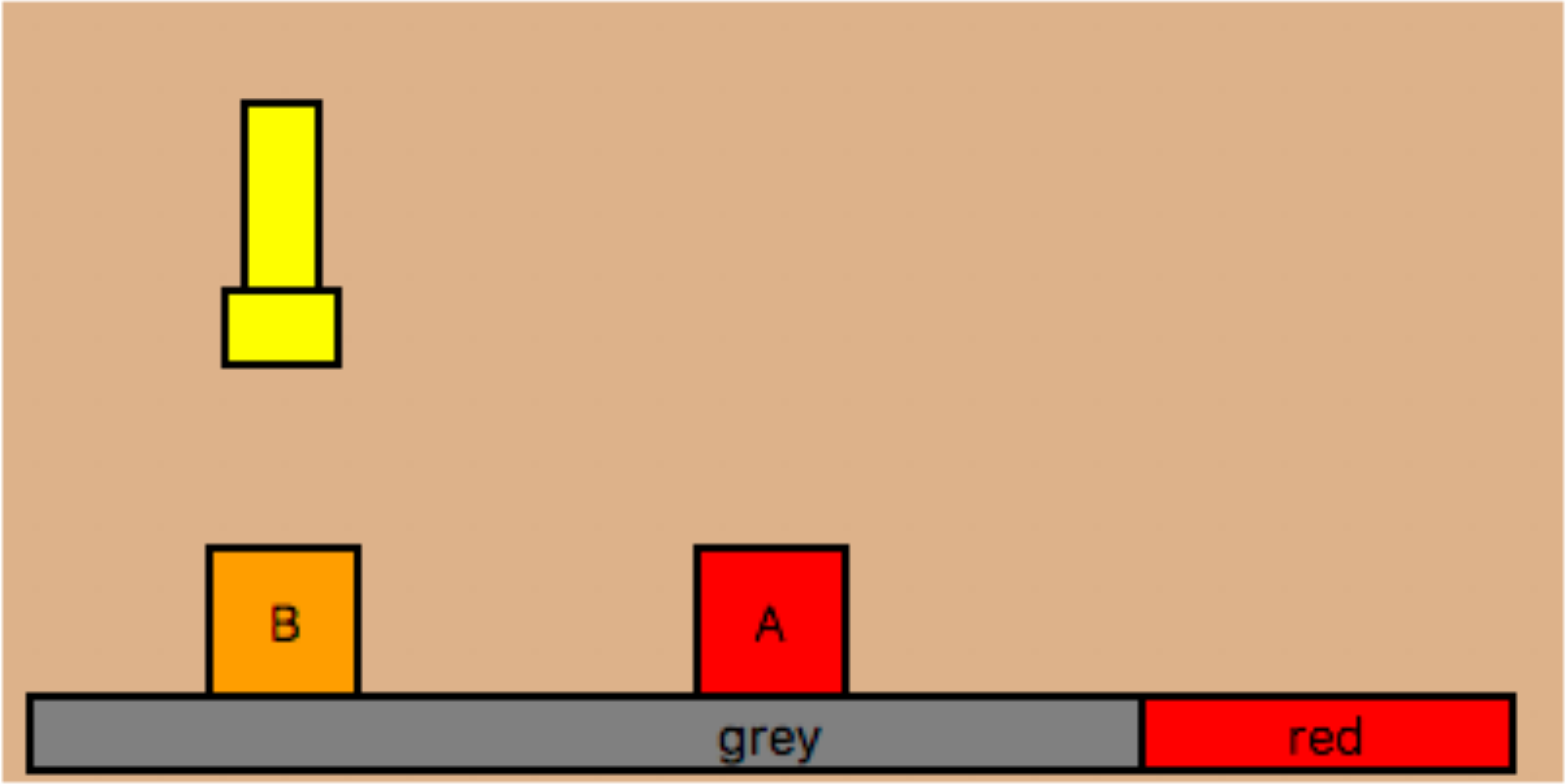
2D Pick-and-Place Solution

- One (of infinitely many) possible solutions
  - move, pick **B**, move, place **B**, move, pick **A**, move, place **A**



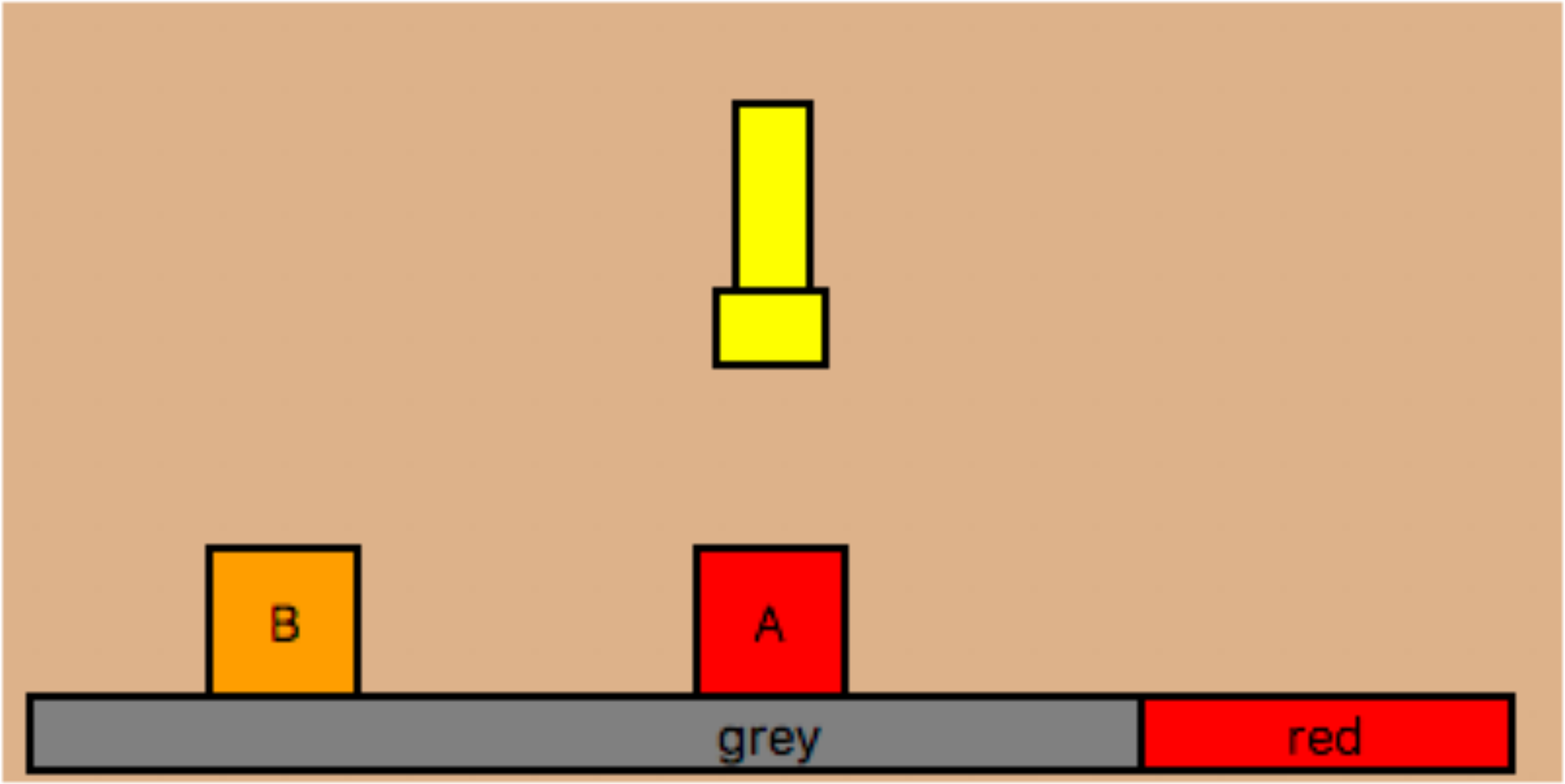
2D Pick-and-Place Solution

- One (of infinitely many) possible solutions
  - move, pick **B**, move, place **B**, move, pick **A**, move, place **A**



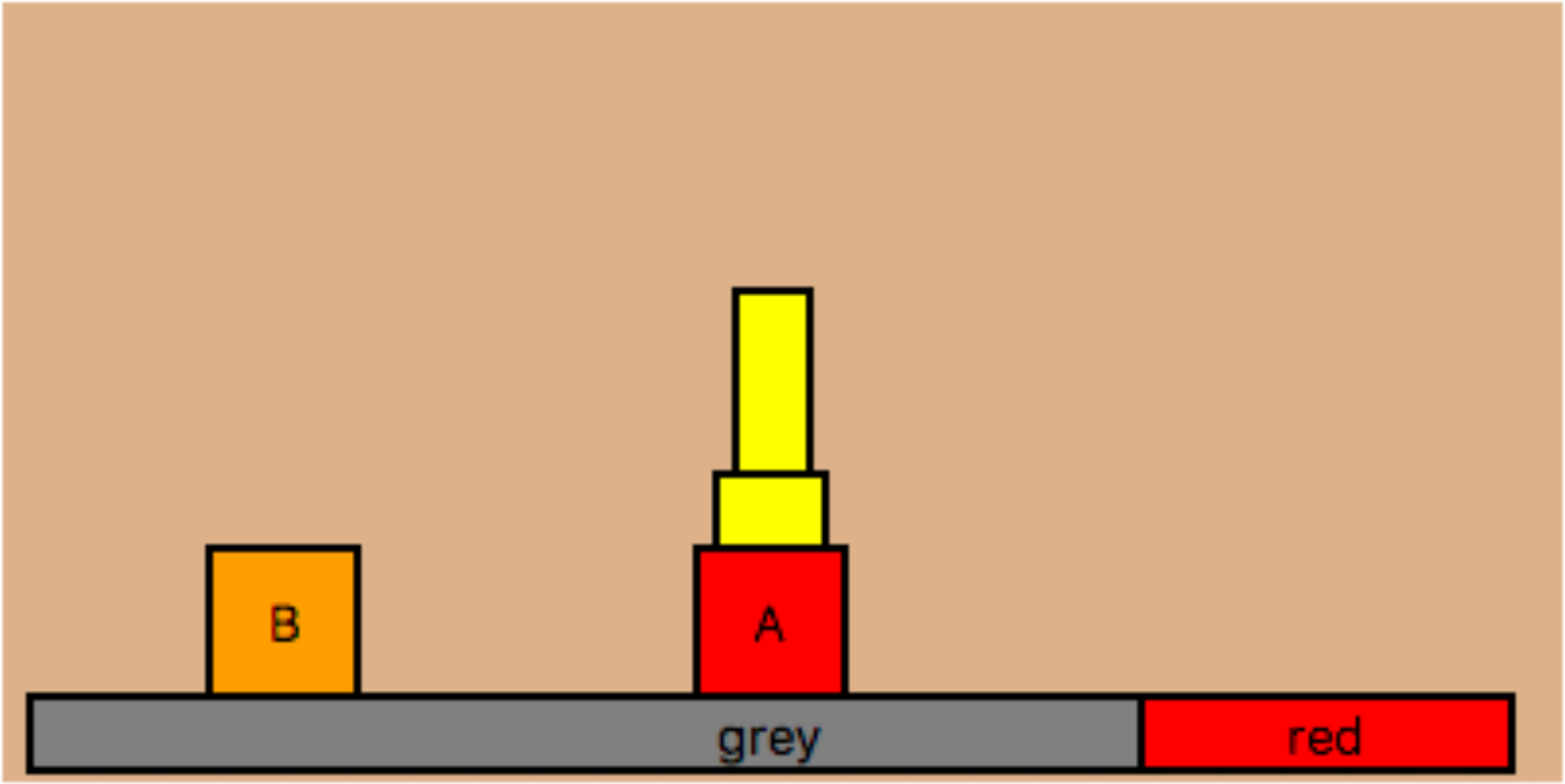
2D Pick-and-Place Solution

- One (of infinitely many) possible solutions
  - move, pick **B**, move, place **B**, move, pick **A**, move, place **A**



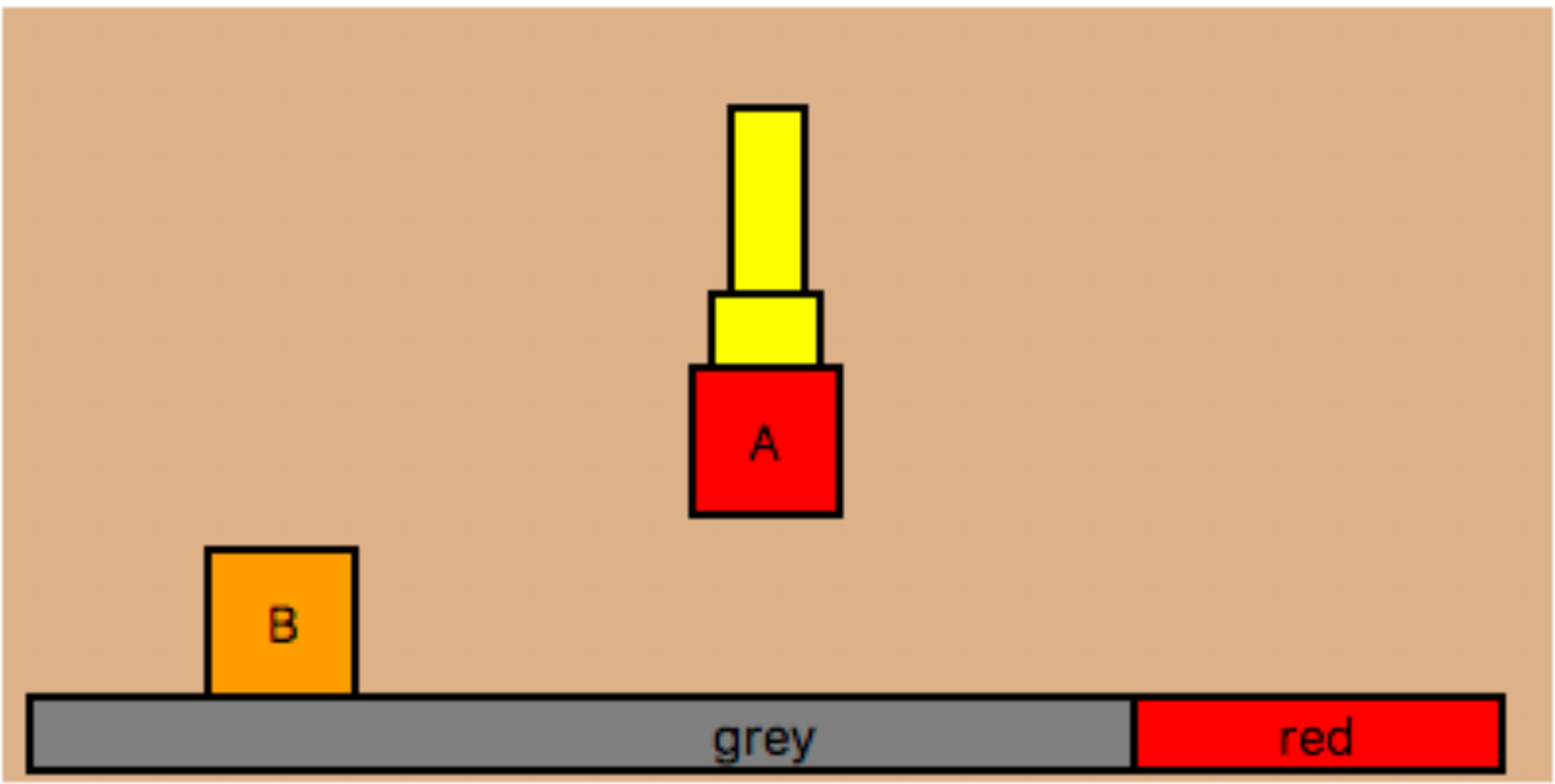
2D Pick-and-Place Solution

- One (of infinitely many) possible solutions
  - move, pick **B**, move, place **B**,
    move, pick **A**, move, place **A**



2D Pick-and-Place Solution

- One (of infinitely many) possible solutions
  - move, pick **B**, move, place **B**, move, pick **A**, move, place **A**



2D Pick-and-Place Solution

- One (of infinitely many) possible solutions
  - move, pick **B**, move, place **B**, move, pick **A**, move, place **A**

2D Pick-and-Place Solution

- One (of infinitely many) possible solutions
  - move, pick **B**, move, place **B**, move, pick **A**, move, place **A**

2D Pick-and-Place Initial & Goal

- Some constants are **numpy arrays**
- **Static initial facts** - value is constant over time
  - (Block, A), (Block, B), (Region, red), (Region, grey), (Conf, **[-7.5 5.]**), (Pose, A, **[0. 0.]**), (Pose, B, **[7.5 0.]**), (Grasp, A, **[0. -2.5]**), (Grasp, B, **[0. -2.5]**)
- **Fluent initial facts** - value changes over time
  - (AtConf, **[-7.5 5.]**), (HandEmpty), (AtPose, A, **[0. 0.]**), (AtPose, B, **[7.5 0.]**)
- **Goal formula:** `(`**`exists`**` (?p) (`**`and`**` (Contained `**A**` ?p `**red**`)`
  `(AtPose `**A**` ?p)))`

2D Pick-and-Place Actions

- Typical PDDL action description except that arguments are **high-dimensional** & **continuous**!
- To use the actions, must **prove** the following **static facts:**

```
(Motion ?q1 ?t ?q2), (Kin ?b ?p ?g ?q)
```

```
(:action move
  :parameters (?q1 ?t ?q2)
  :precondition (and (Motion ?q1 ?t ?q2) (AtConf ?q1))
  :effect (and (AtConf ?q2) (not (AtConf ?q1))))

(:action pick
  :parameters (?b ?p ?g ?q)
  :precondition (and (Kin ?b ?p ?g ?q)
                     (AtConf ?q) (AtPose ?b ?p) (HandEmpty))
  :effect (and (AtGrasp ?b ?g)
               (not (AtPose ?b ?p)) (not (HandEmpty))))
```

BFS in Discretized State-Space

- Suppose we were **given** the following additional static facts:
  - (Motion, [-7.5 5.], τ_1, [0. 2.5]), (Motion, [-7.5 5.], τ_2, [-5. 5.]), (Motion, [-5. 5.], τ_3, [0. 2.5]), (Kin, A, [0. 0.], [0. -2.5], [0. 2.5]), …

Initial State

<u>(AtConf, [-7.5 5.])</u>
(AtPose, A, [0. 0.])
(AtPose, B, [7.5 0.])
(HandEmpty)

BFS in Discretized State-Space

- Suppose we were **given** the following additional static facts:
  - (Motion, [-7.5 5.], τ_1, [0. 2.5]), (Motion, [-7.5 5.], τ_2, [-5. 5.]), (Motion, [-5. 5.], τ_3, [0. 2.5]), (Kin, A, [0. 0.], [0. -2.5], [0. 2.5]), …

Initial State

(AtConf, [-7.5 5.])
(AtPose, A, [0. 0.])
(AtPose, B, [7.5 0.])
(HandEmpty)

(**move**, [-7.5 5.], τ_1, [0. 2.5])

(AtConf, [0. 2.5])
(AtPose, A, [0. 0.])
(AtPose, B, [7.5 0.])
(HandEmpty)

BFS in Discretized State-Space

- Suppose we were **given** the following additional static facts:
  - (Motion, [-7.5 5.], τ_1, [0. 2.5]), (Motion, [-7.5 5.], τ_2, [-5. 5.]), (Motion, [-5. 5.], τ_3, [0. 2.5]), (Kin, A, [0. 0.], [0. -2.5], [0. 2.5]), …

Initial State

(AtConf, [-7.5 5.])
(AtPose, A, [0. 0.])
(AtPose, B, [7.5 0.])
(HandEmpty)

(**move**, [-7.5 5.], τ_1, [0. 2.5]) →

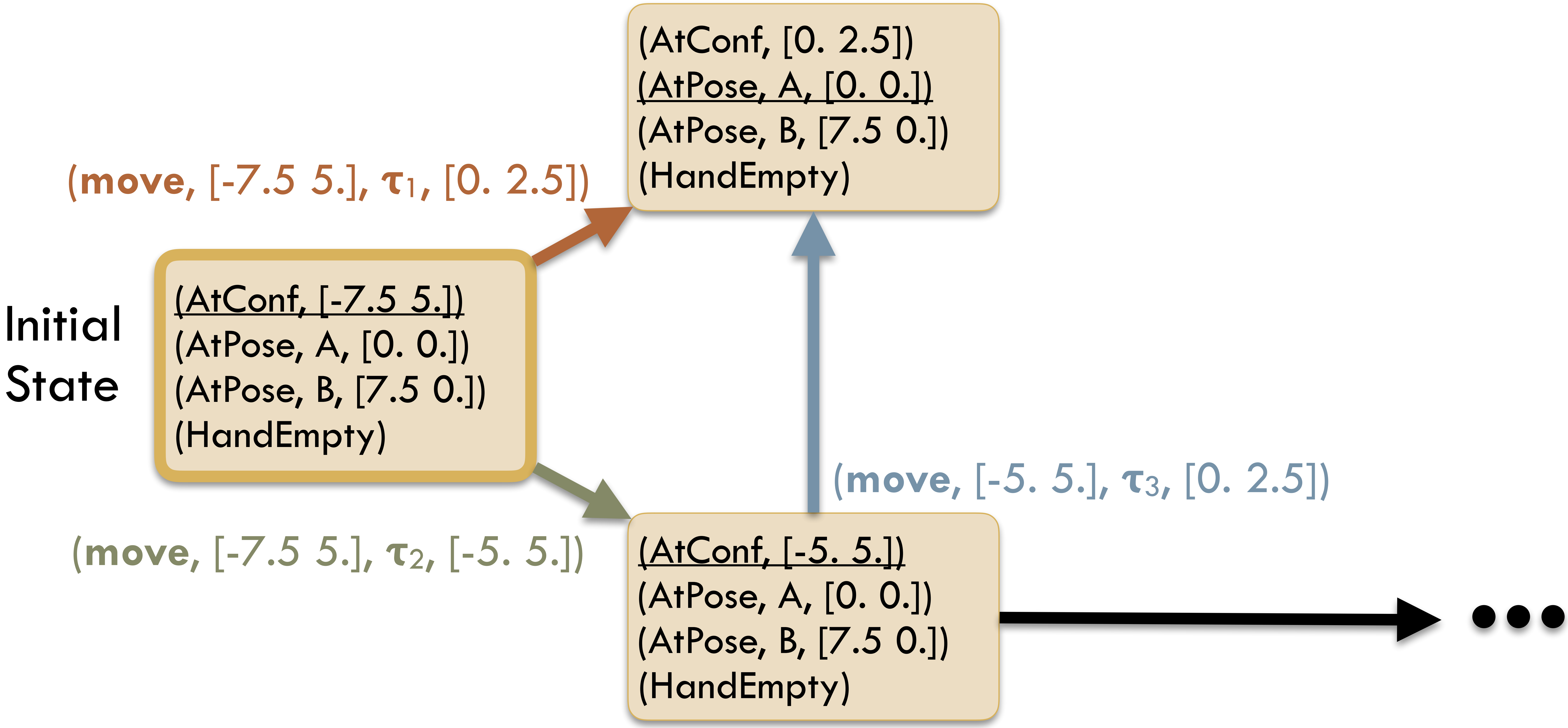
(AtConf, [0. 2.5])
(AtPose, A, [0. 0.])
(AtPose, B, [7.5 0.])
(HandEmpty)

(**move**, [-7.5 5.], τ_2, [-5. 5.]) →

(AtConf, [-5. 5.])
(AtPose, A, [0. 0.])
(AtPose, B, [7.5 0.])
(HandEmpty)

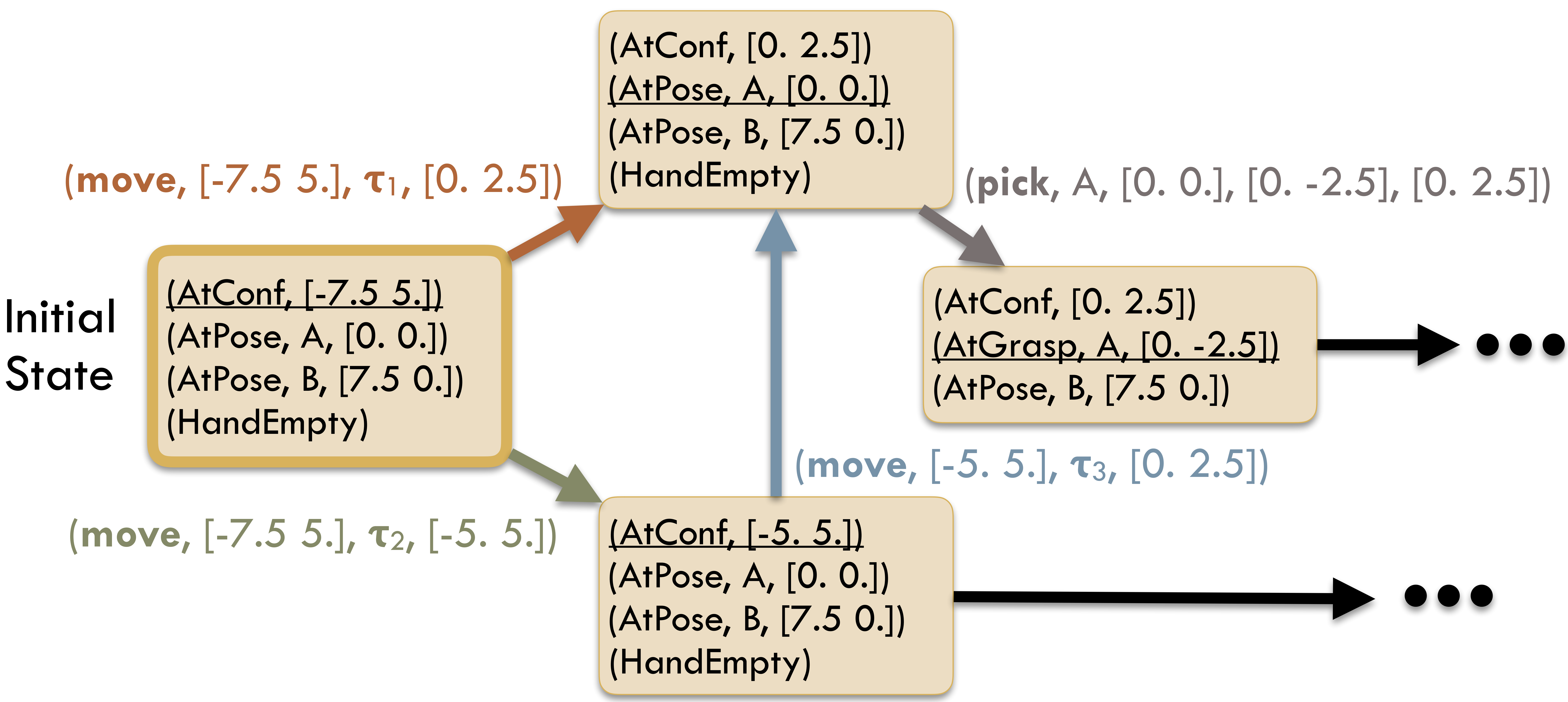
BFS in Discretized State-Space

- Suppose we were **given** the following additional static facts:
  - (Motion, [-7.5 5.], τ_1, [0. 2.5]), (Motion, [-7.5 5.], τ_2, [-5. 5.]), (Motion, [-5. 5.], τ_3, [0. 2.5]), (Kin, A, [0. 0.], [0. -2.5], [0. 2.5]), …



BFS in Discretized State-Space

- Suppose we were **given** the following additional static facts:
  - (Motion, [-7.5 5.], τ_1, [0. 2.5]), (Motion, [-7.5 5.], τ_2, [-5. 5.]), (Motion, [-5. 5.], τ_3, [0. 2.5]), (Kin, A, [0. 0.], [0. -2.5], [0. 2.5]), …



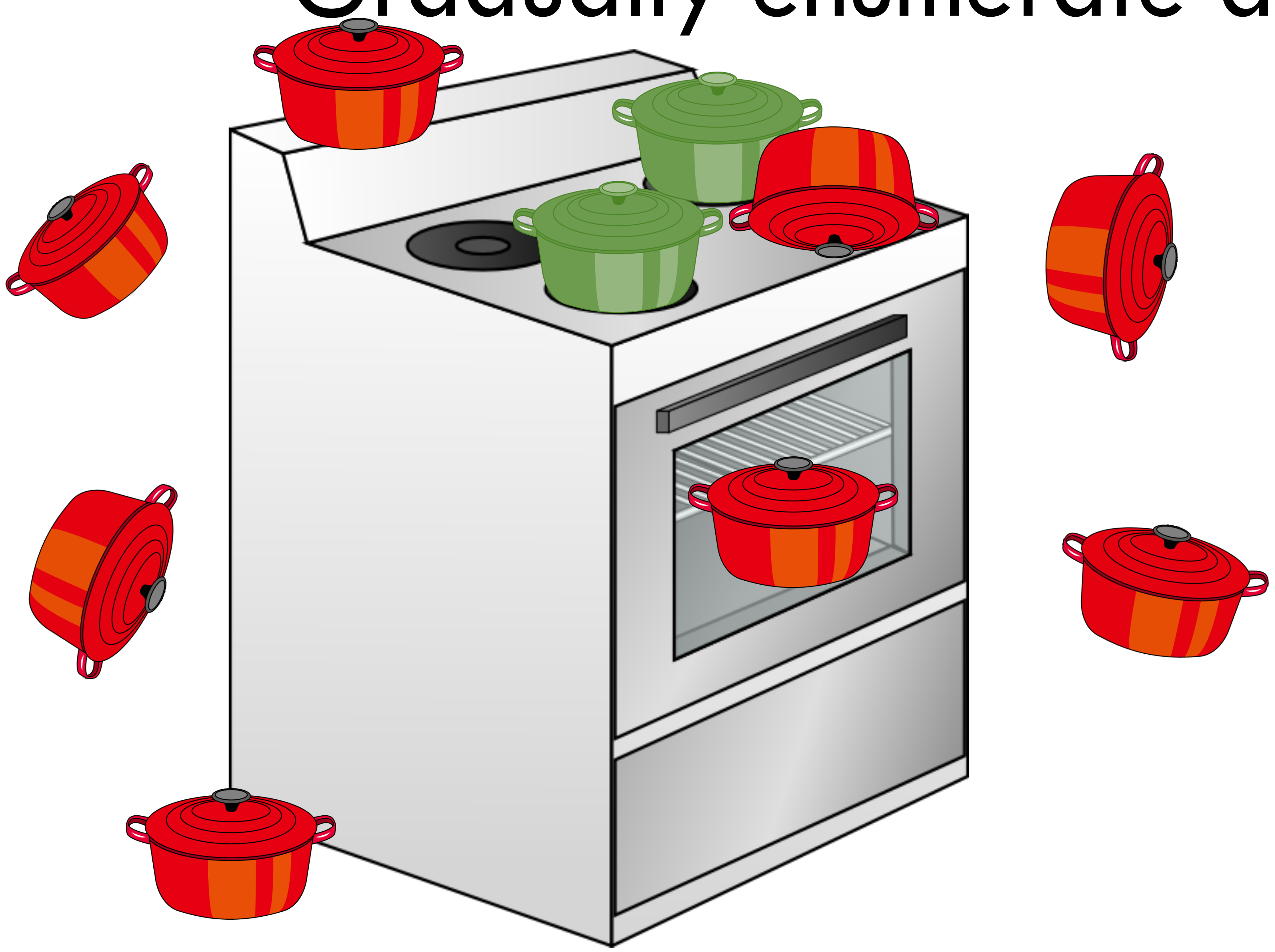
No a Priori Discretization

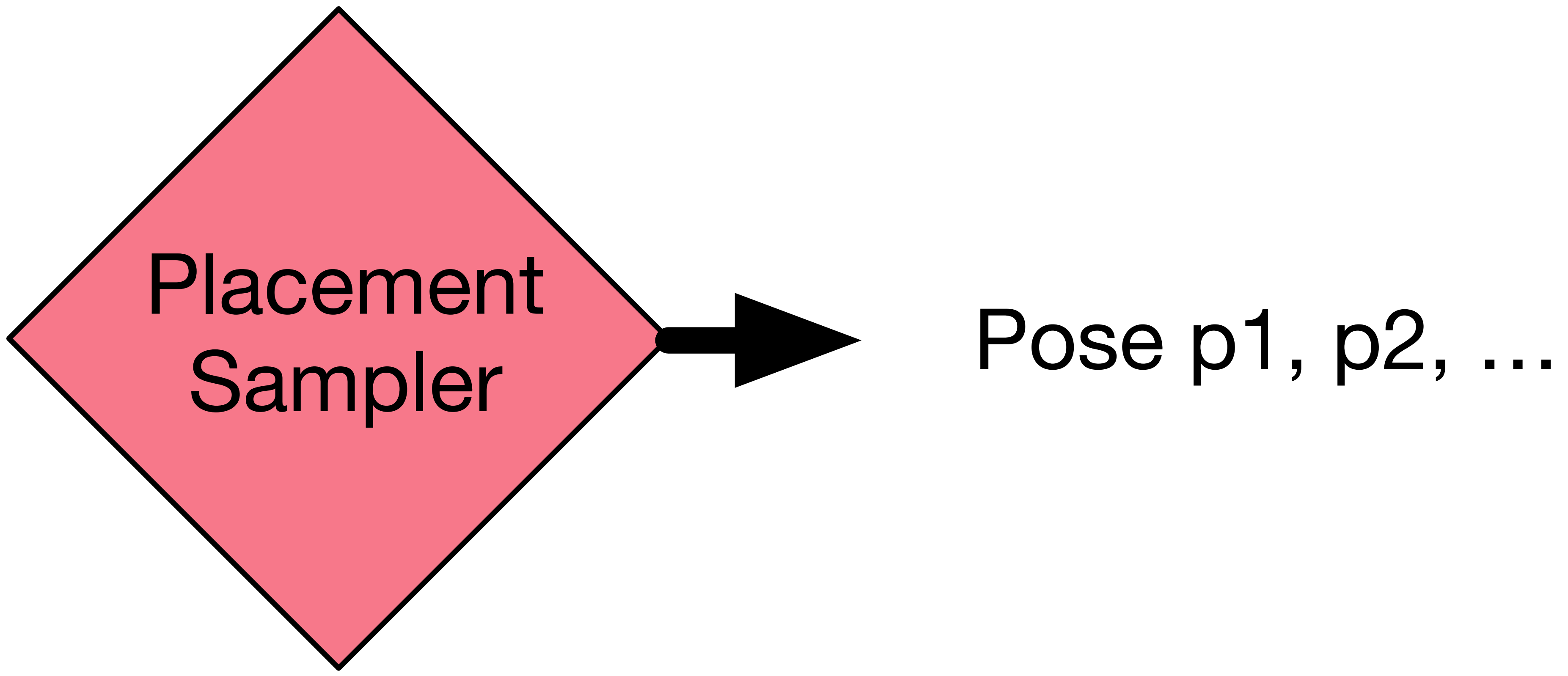
- **Values given at start:**
  - 1 initial configuration: (Conf, **[-7.5 5.]**)
  - 2 initial poses: (Pose, A, **[0. 0.]**), (Pose, B, **[7.5 0.]**)
  - 2 grasps: (Grasp, A, **[0. -2.5]**), (Grasp, B, **[0. -2.5]**)

- **Planner needs to find:**
  - 1 pose within a region: `(Contain A ?p` **red**`)`
  - 1 collision-free pose: `(CFree A ?p ? B ?p2)`
  - 4 grasping configurations: `(Kin ?b ?p ?g ?q)`
  - 4 robot trajectories: `(Motion ?q1 ?t ?q2)`

What Samplers Do We Need?

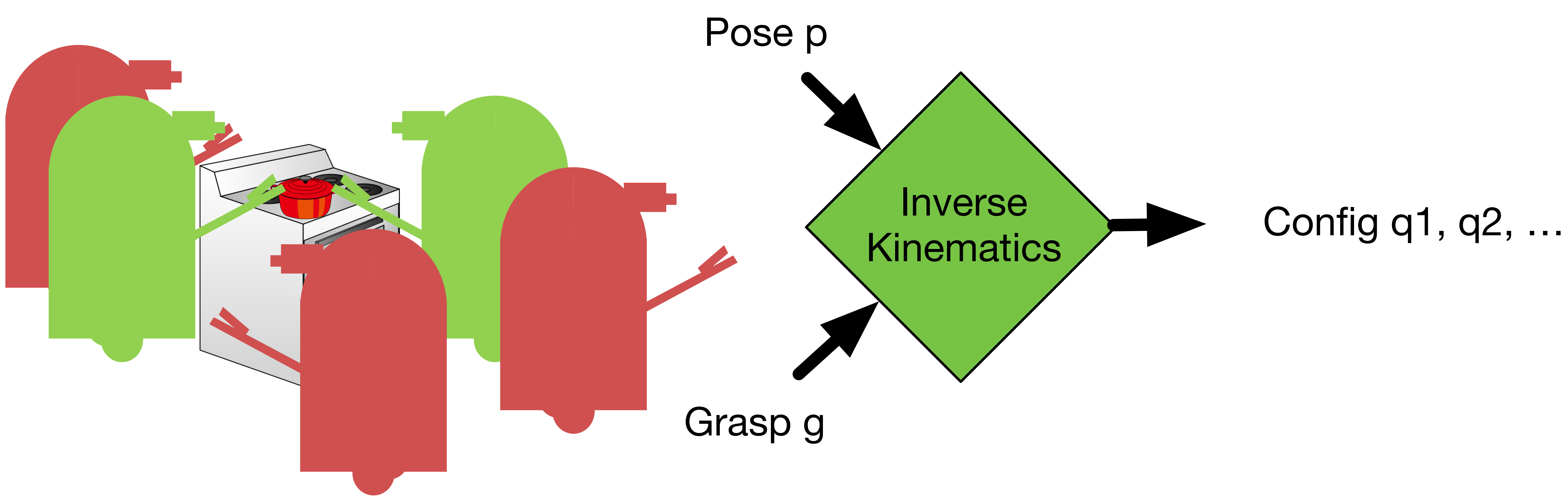
- **Low-dimensional** placement stability constraint (`Contain`)
  - *i*.e. 1D manifold embedded in 2D pose space
- Directly **sample values that satisfy the constraint**
- May need **arbitrarily many** samples
  - Gradually enumerate an **infinite sequence**



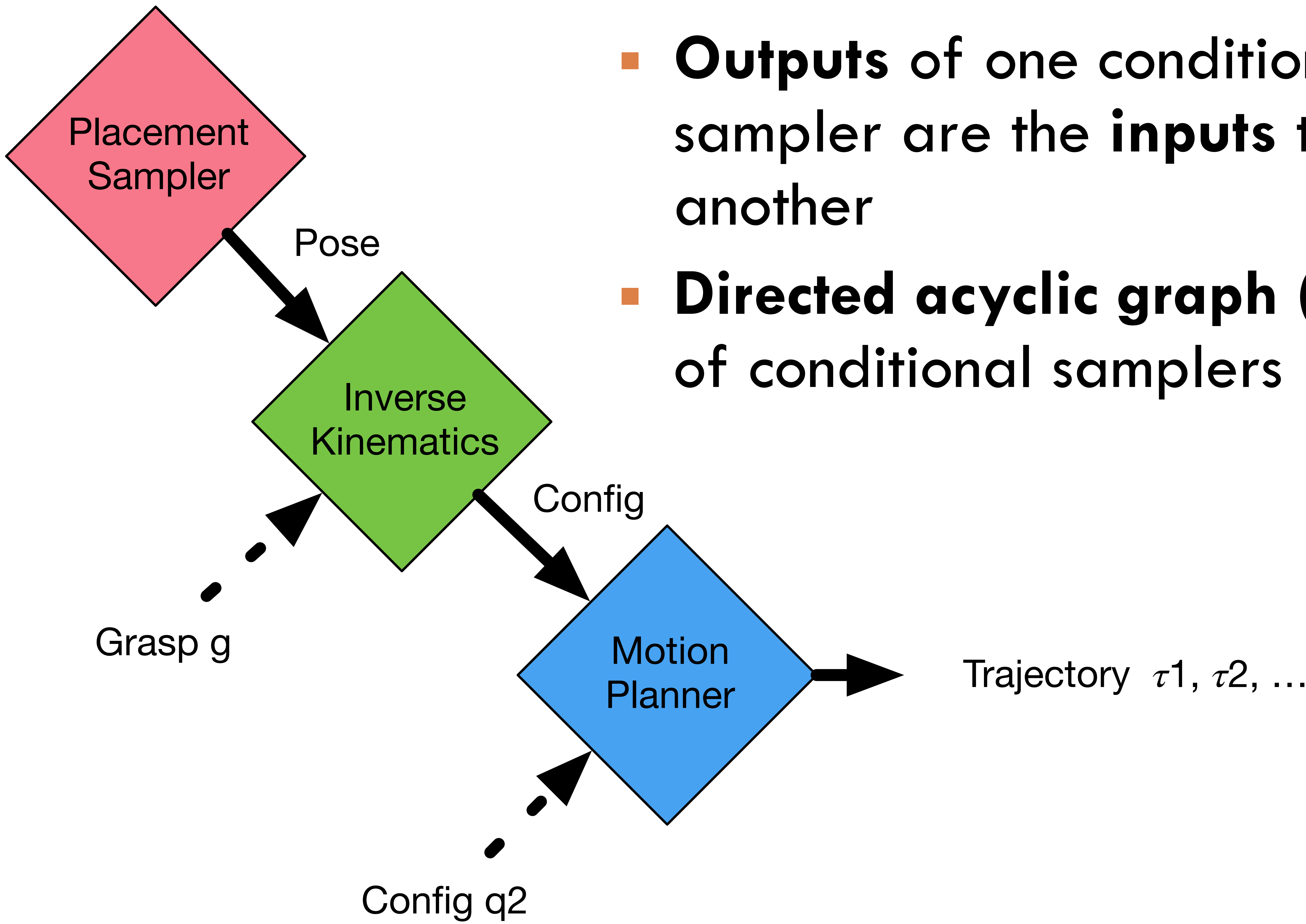


Intersection of Constraints

- **Kinematic constraint** (`Kin`) involves poses, grasps, and configurations
- **Conditional samplers** - samplers with inputs



Composing Conditional Samplers



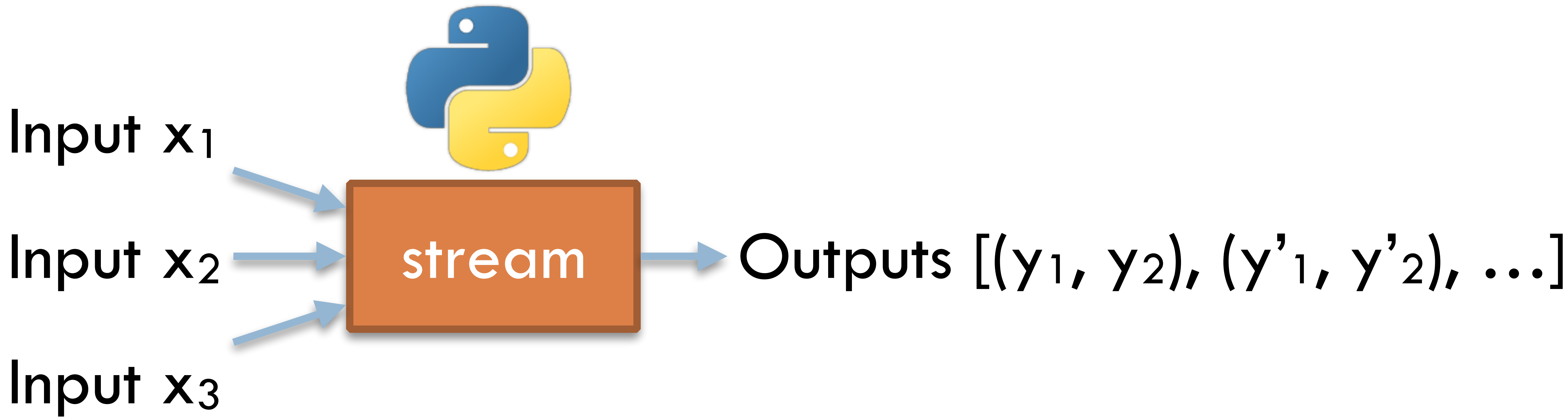
- **Outputs** of one conditional sampler are the **inputs** to another
- **Directed acyclic graph (DAG)** of conditional samplers

Stream: a function to a generator

- **Advantages**
  - Programmatic implementation
  - Compositional
  - Supports infinite sequences

```python
def stream(x1, x2, x3):
    i = 0
    while True:
        y1 = i*(x1 + x2)
        y2 = i*(x2 + x3)
        yield (y1, y2)
        i += 1
```

- **Stream** - function from an **input object tuple** (x_1, x_2, x_3) to a (potentially infinite) sequence of **output object tuples** $[(y_1, y_2), (y'_1, y'_2), \ldots]$



Stream Certified Facts

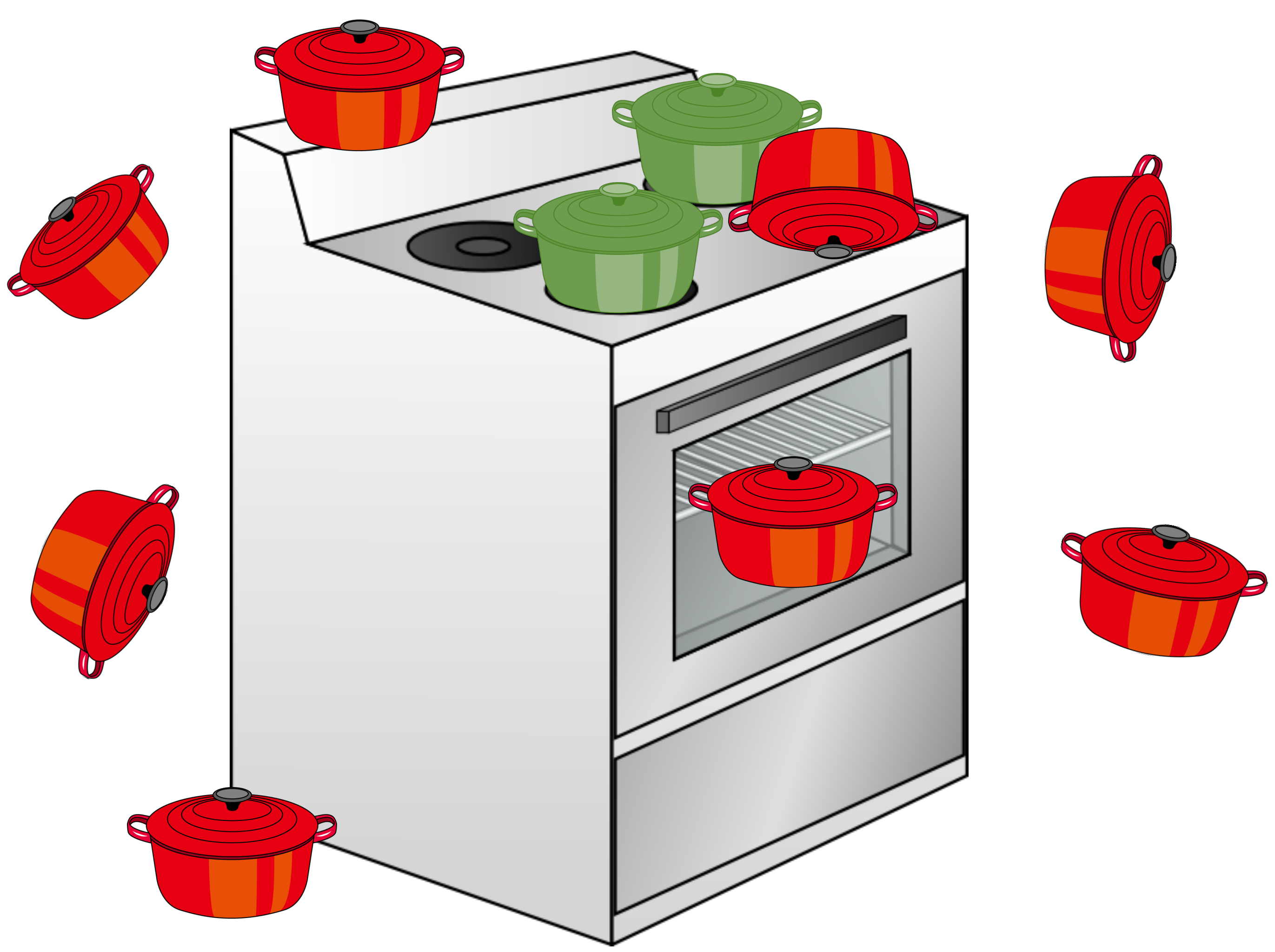
- Objects alone aren't helpful: **what do they represent?**
  - Communicate semantics using **predicates**!

- Augment stream specification with:
  - **Domain facts** - static facts declaring legal **inputs**
    - e.g. only configurations can be motion inputs

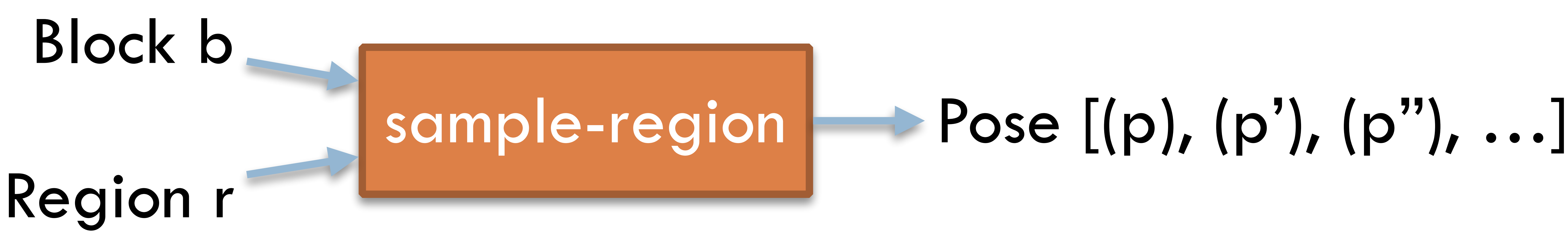
  - **Certified facts** - static facts that all **outputs** satisfy with their corresponding **inputs**
    - e.g. poses sampled from a region are within it

Sampling Contained Poses


```
(:stream sample-region
  :inputs (?b ?r)
  :domain (and (Block ?b) (Region ?r))
  :outputs (?p)
  :certified (and (Pose ?b ?p) (Contain ?b ?p ?r)))
```



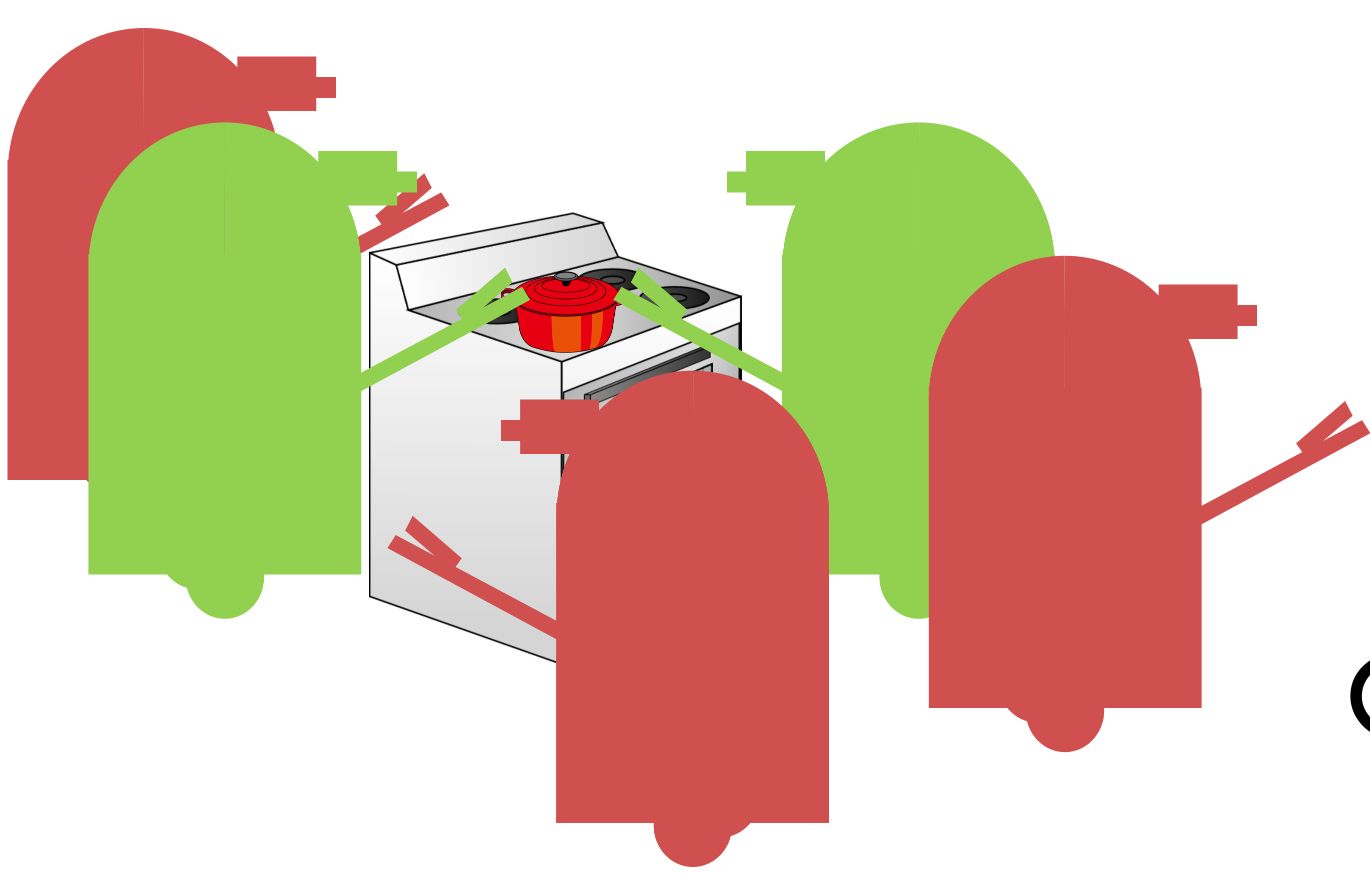
```python
def sample_region(b, r):
  x_min, x_max = REGIONS[r]
  w = BLOCKS[b].width
  while True:
      x = random.uniform(x_min + w/2,
                         x_max - w/2)
      p = np.array([x, 0.])
      yield (p,)
```

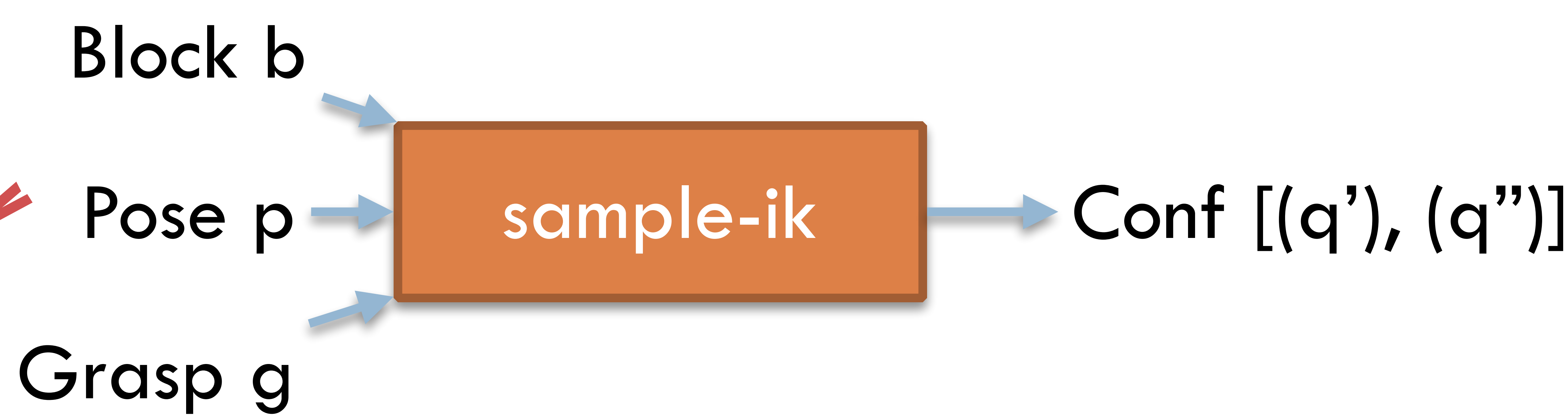


Sampling IK Solutions

- **Inverse kinematics** (IK) to produce robot grasping configuration
  - Trivial in 2D, non-trial in general (e.g. 7 DOF arm)

```
(:stream sample-ik
  :inputs (?b ?p ?g)
  :domain (and (Pose ?b ?p) (Grasp ?b ?g))
  :outputs (?q)
  :certified (and (Conf ?q) (Kin ?b ?p ?g ?q)))
```

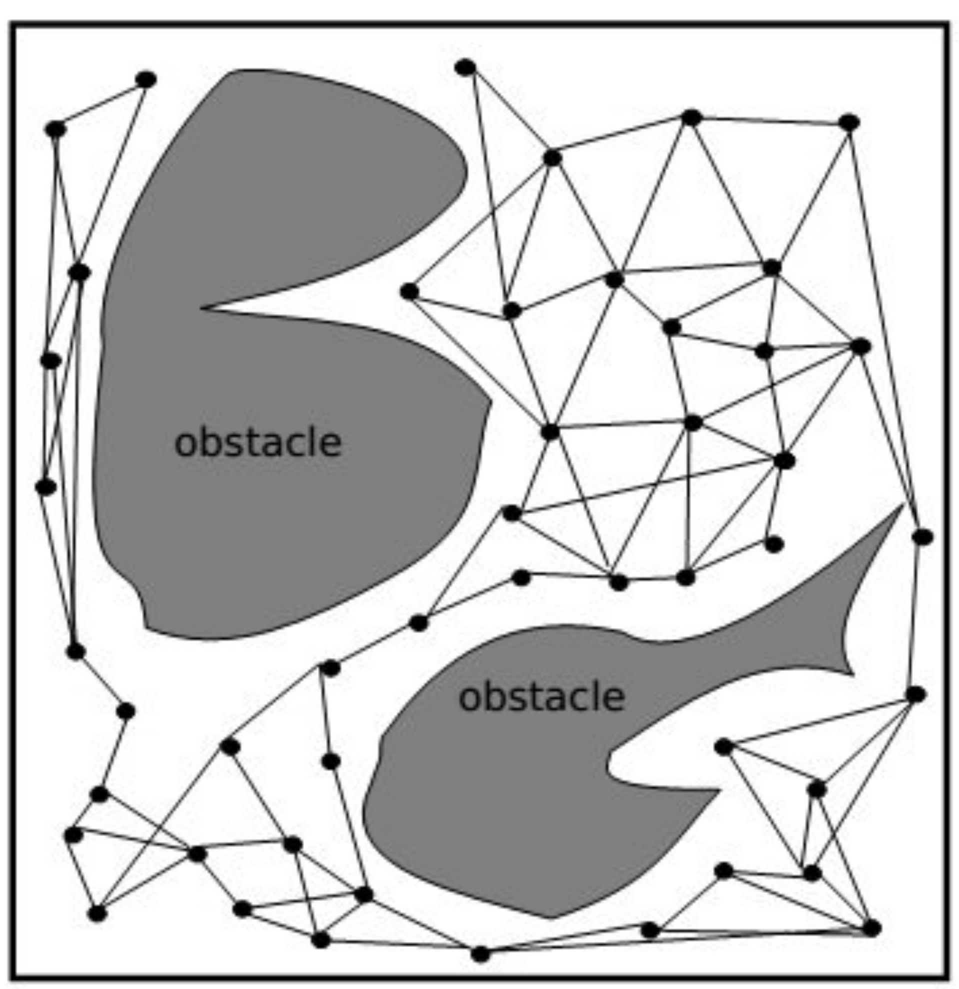




Calling Motion Planner

- "Sample" (e.g. via an RRT) **multi-waypoint trajectories**
- Include **joint limits & fixed obstacle collisions**, but not movable object collisions

```
(:stream sample-motion
    :inputs (?q1 ?q2)
    :domain (and (Conf ?q1) (Conf ?q2))
    :outputs (?t)
    :certified (and (Traj ?t) (Motion ?q1 ?t ?q2)))
```



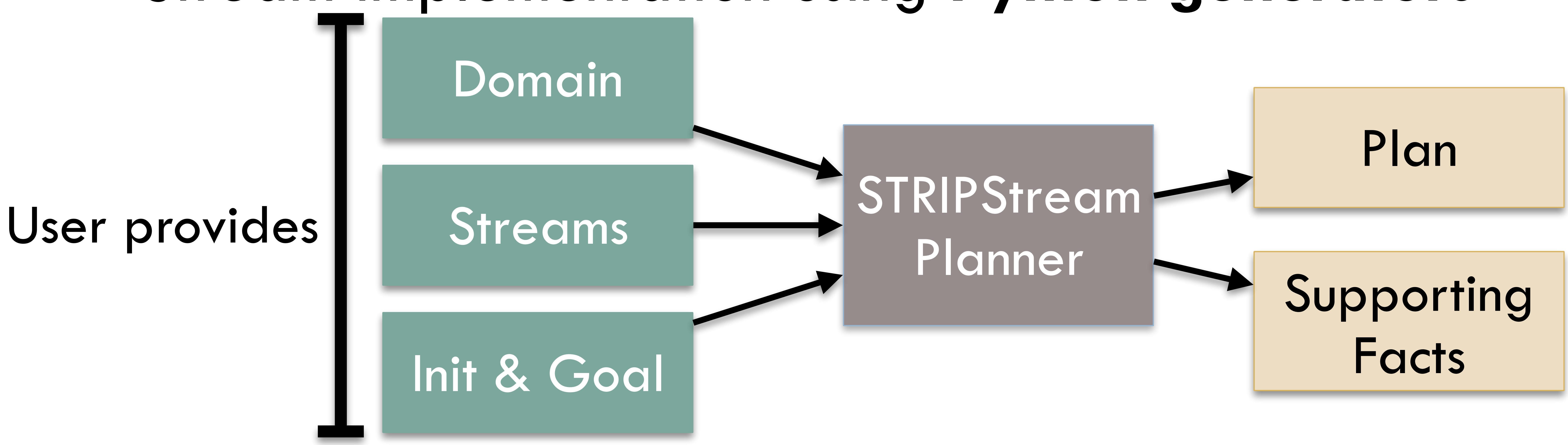
Check Block Collisions

- **Test stream:** stream without output objects
- Return True if **collision-free** placement (e.g. via FCL)

```
(:stream test-cfree
  :inputs (?b1 ?p1 ?b2 ?p2)
  :domain (and (Pose ?b1 ?p1) (Pose ?b2 ?p2))
  :outputs ()
  :certified (CFree ?b1 ?p1 ?b2 ?p2))
```

STRIPStream = STRIPS + Streams

- **Domain dynamics** (*domain.pddl*): *declares* actions
- **Stream properties** (*stream.pddl*)
  - Declares stream inputs, outputs, and certified facts
- **Problem** and **stream implementation** (*problem.py*)
  - Initial state, **Python constants**, & goal formula
  - Stream implementation using **Python generators**



Two STRIPStream Algorithms

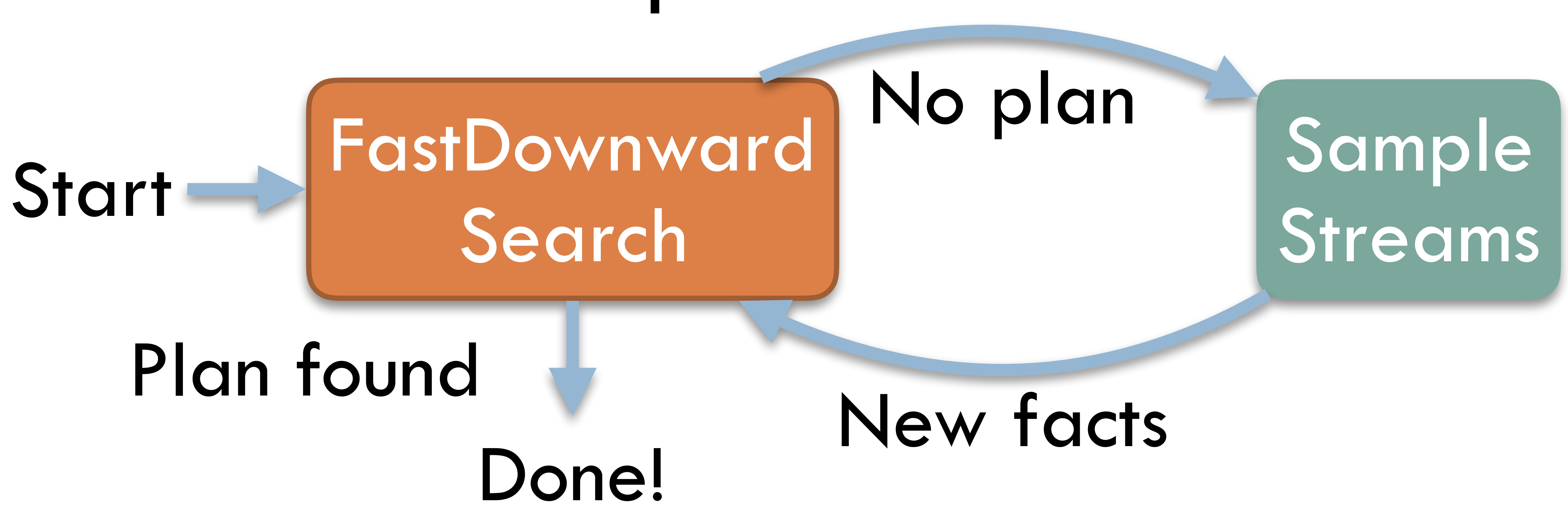
- **STRIPStream planners decide** which streams to use
- Algorithms alternate between **searching & sampling:**
  1. **Search** a finite PDDL problem for plan
  2. **Modify** the PDDL problem (depending on the plan)

- Search implemented using **off-the-shelf algorithms**
  - **Off-the-shelf AI planner** - FastDownward
    - Exploits factoring in its search heuristics (*e.g.* h_{FF})
    - http://www.fast-downward.org/
  - **Probabilistically complete** given *sufficient* samplers

Incremental Algorithm

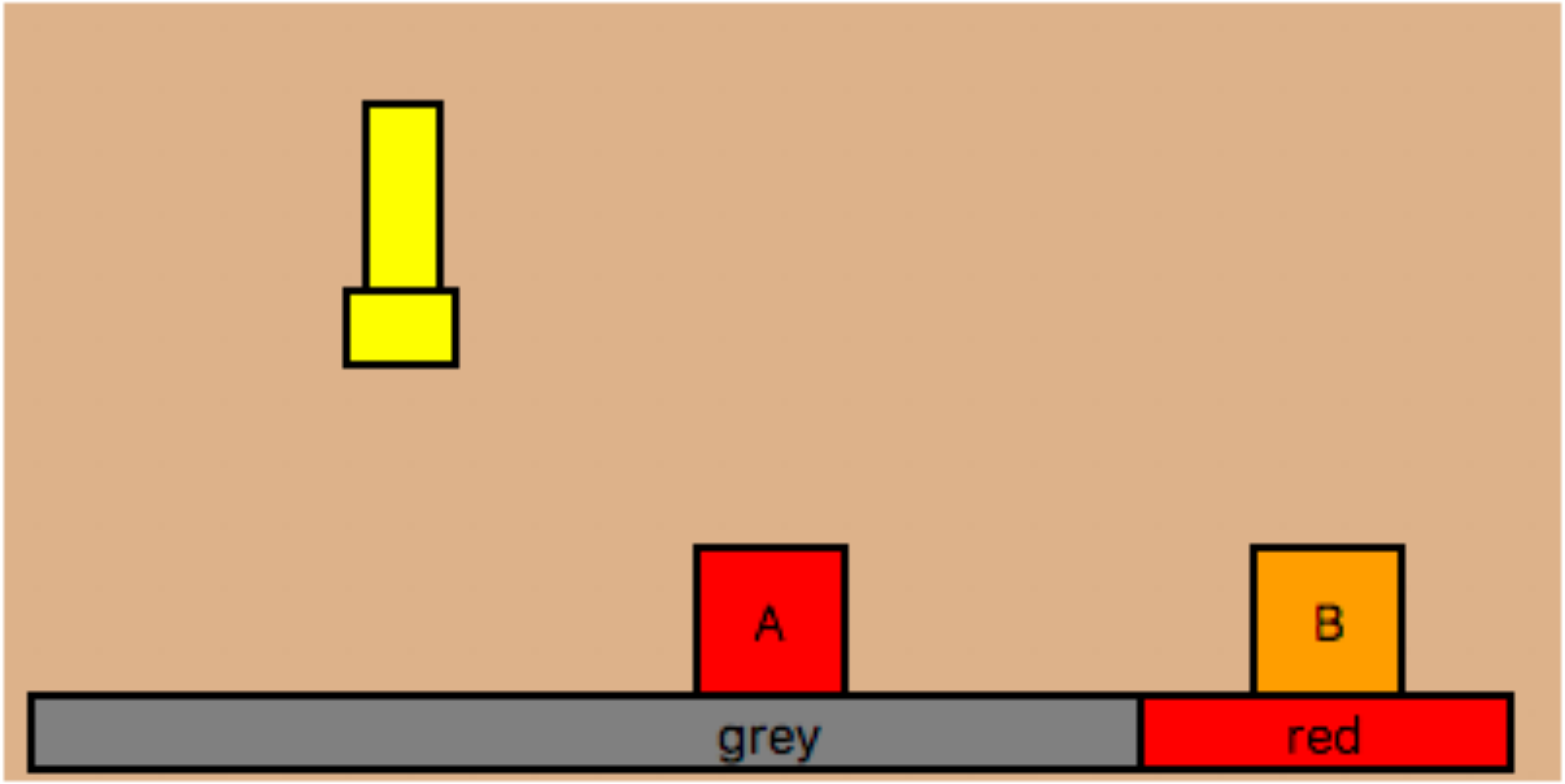
- Incrementally construct all possible initial facts
- Periodically check if a solution exists
- Repeat:

1. **Compose** and **evaluate** a finite number of streams to unveil more facts in the initial state
2. **Search** the current PDDL problem for plan
3. **Terminate** when a plan is found



Incremental: Sampling Iteration 1

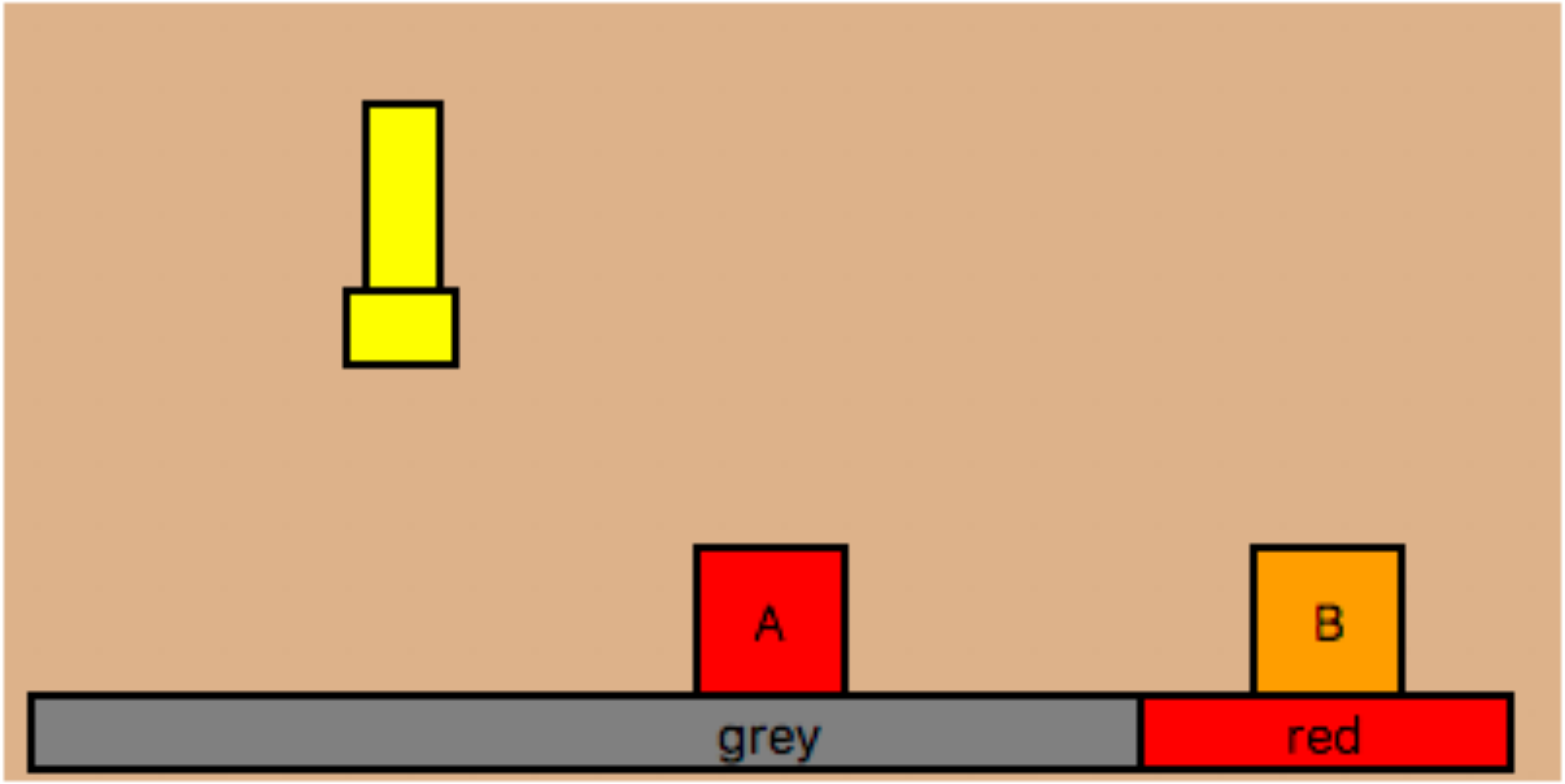
Iteration 1 - 14 stream evaluations



Incremental: Sampling Iteration 1

Iteration 1 - 14 stream evaluations

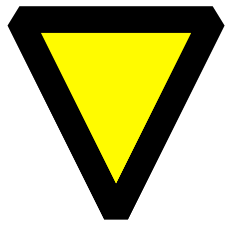
- **Sampled:**

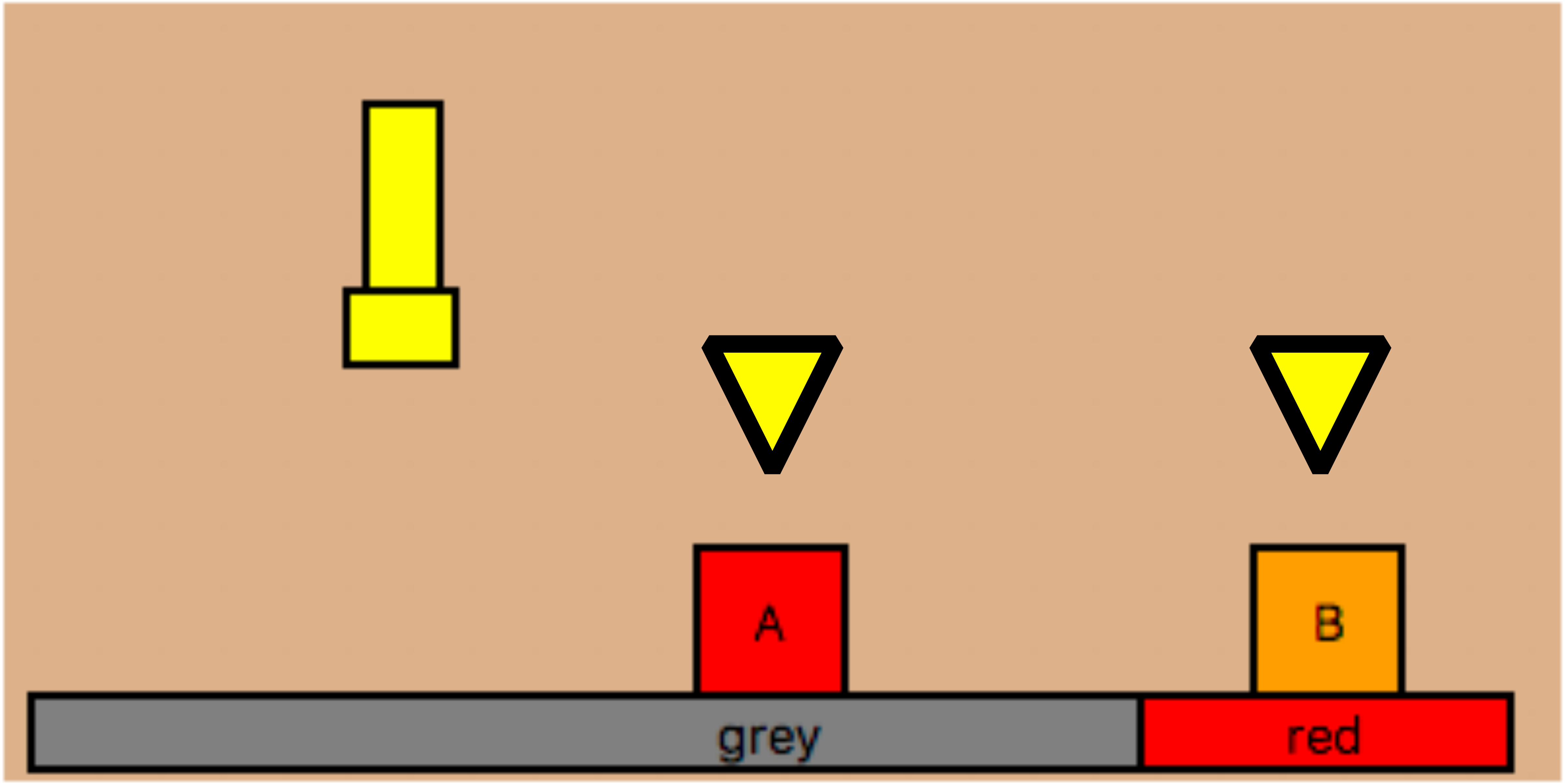


Incremental: Sampling Iteration 1

Iteration 1 - 14 stream evaluations

- **Sampled:**
  - 2 new robot configurations:

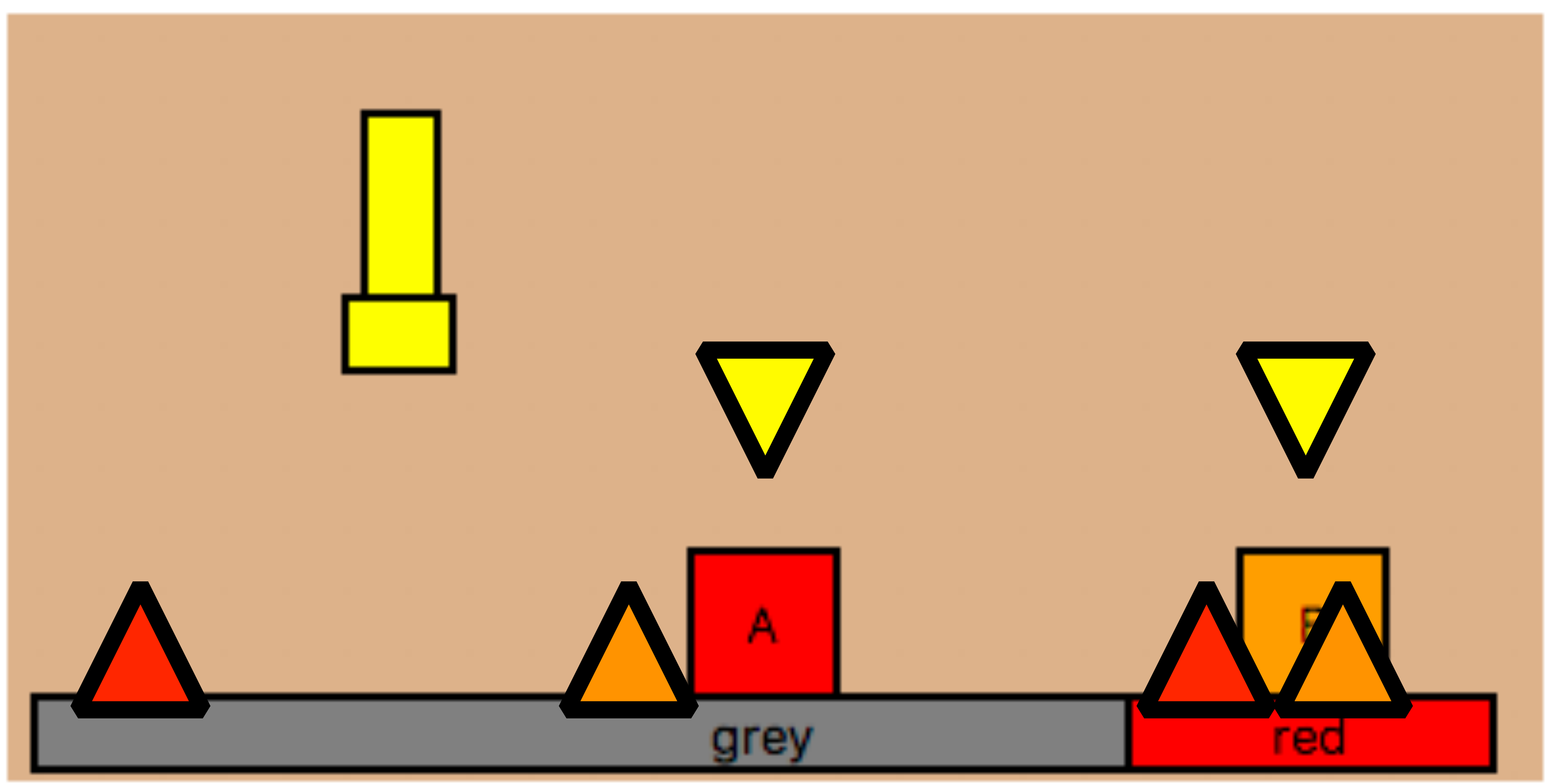




Incremental: Sampling Iteration 1

Iteration 1 - 14 stream evaluations

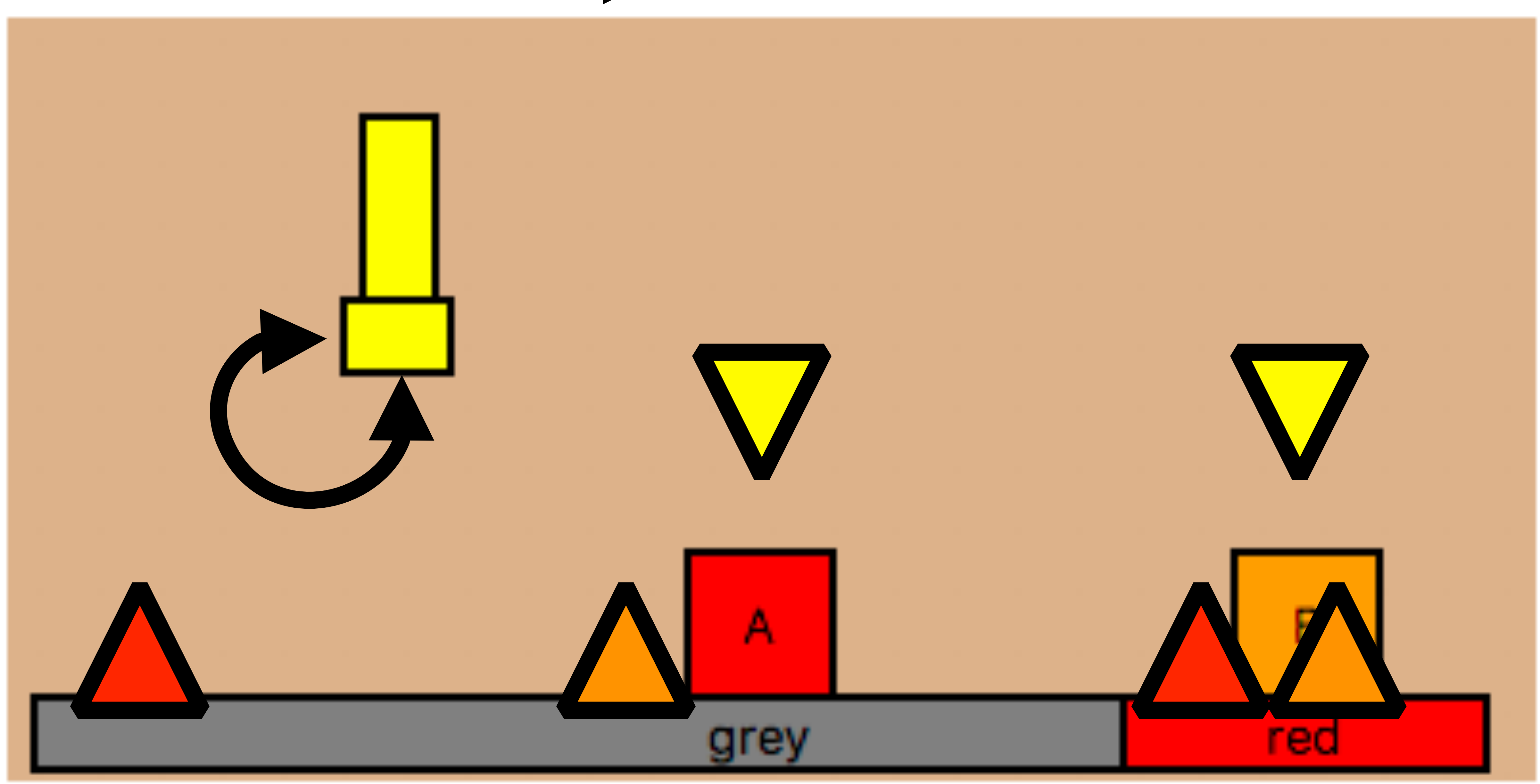
- **Sampled:**
  - 2 new robot configurations:
  - 4 new block poses:



Incremental: Sampling Iteration 1

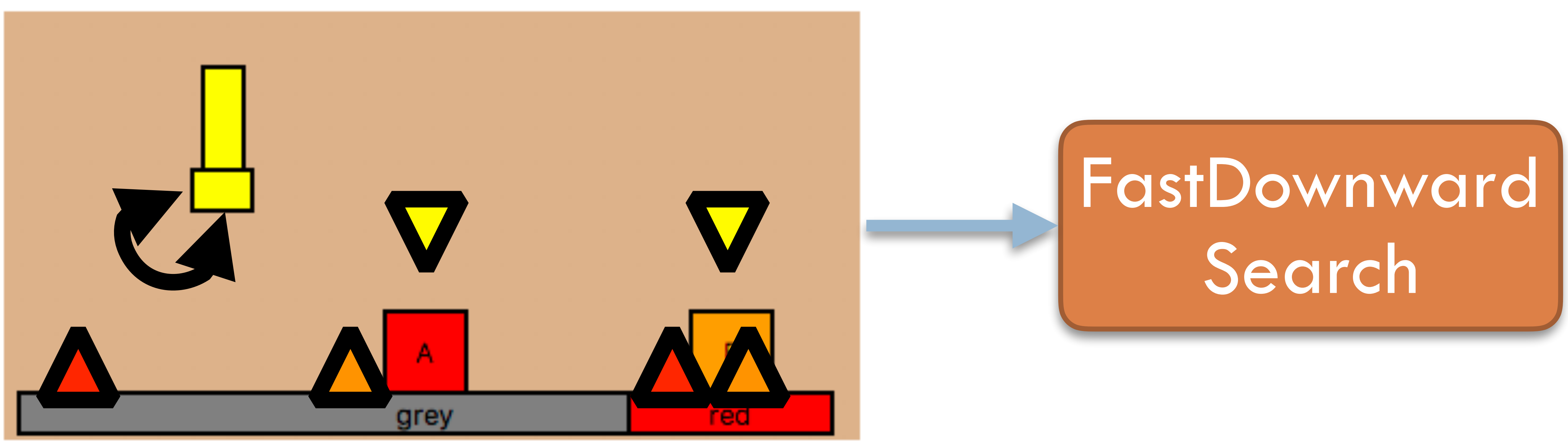
Iteration 1 - 14 stream evaluations

- **Sampled:**
  - 2 new robot configurations:
  - 4 new block poses:
  - 2 new trajectories:



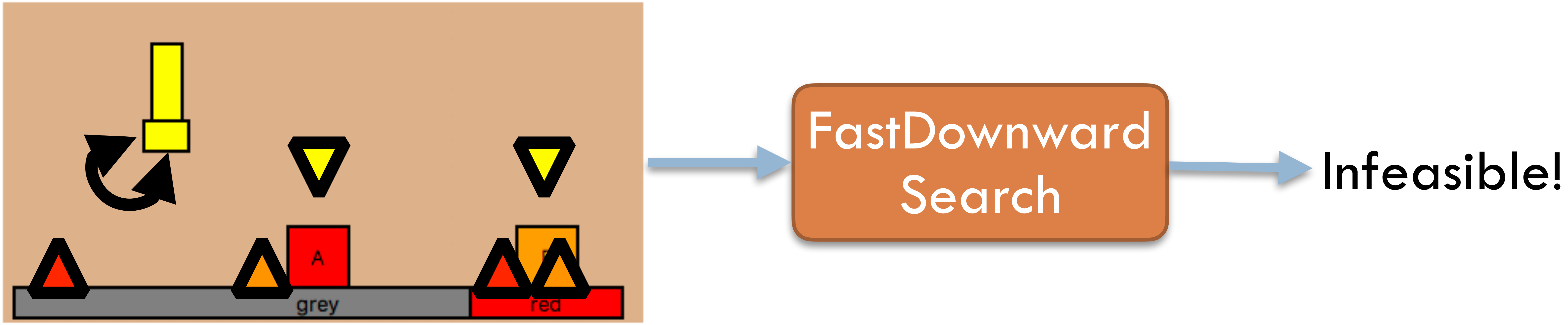
Incremental: Search Iteration 1

- Pass current discretization to FastDownward
- If **infeasible**, the current set of samples is insufficient



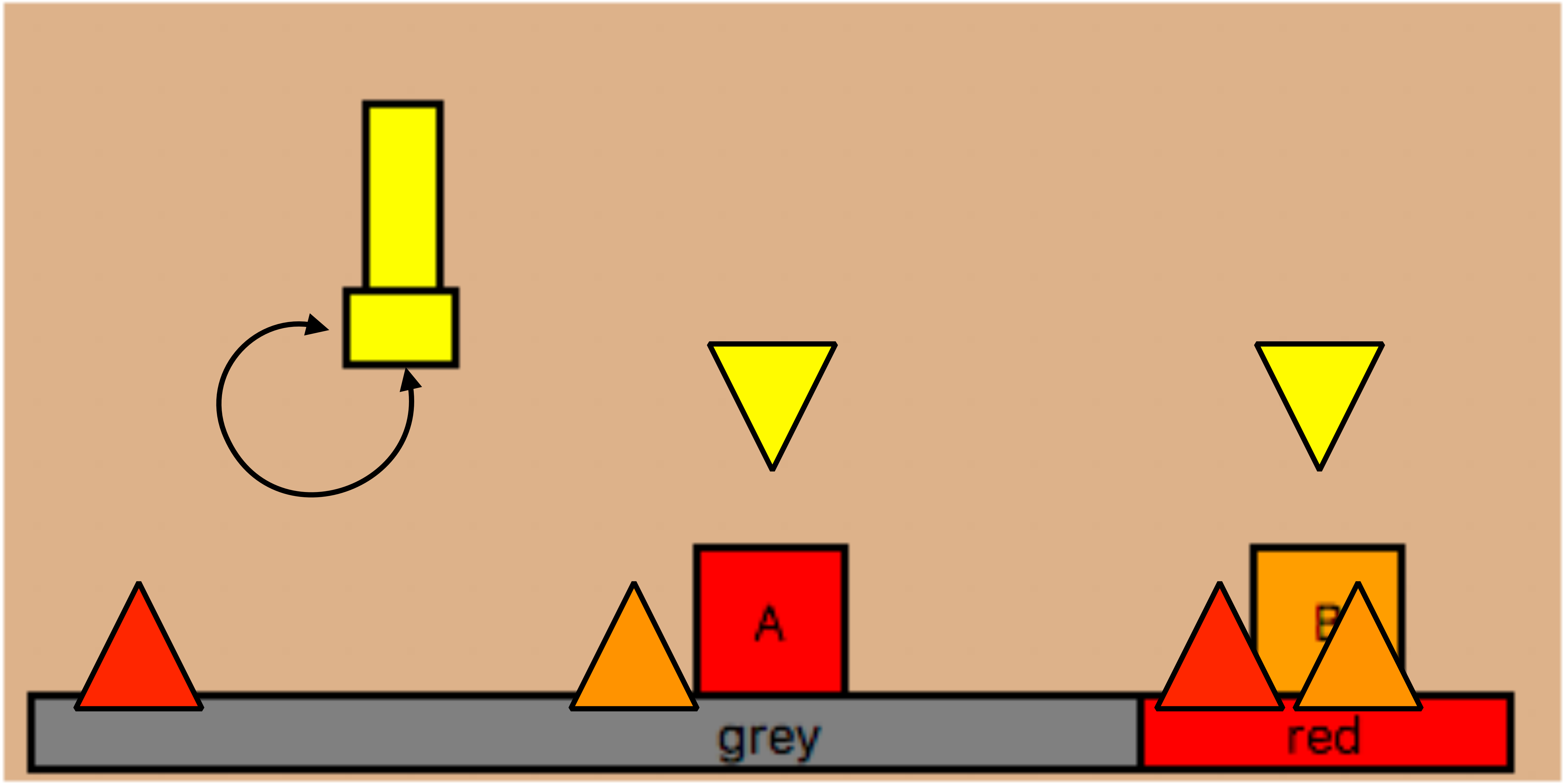
Incremental: Search Iteration 1

- Pass current discretization to FastDownward
- If **infeasible**, the current set of samples is insufficient



Incremental: Sampling Iteration 2

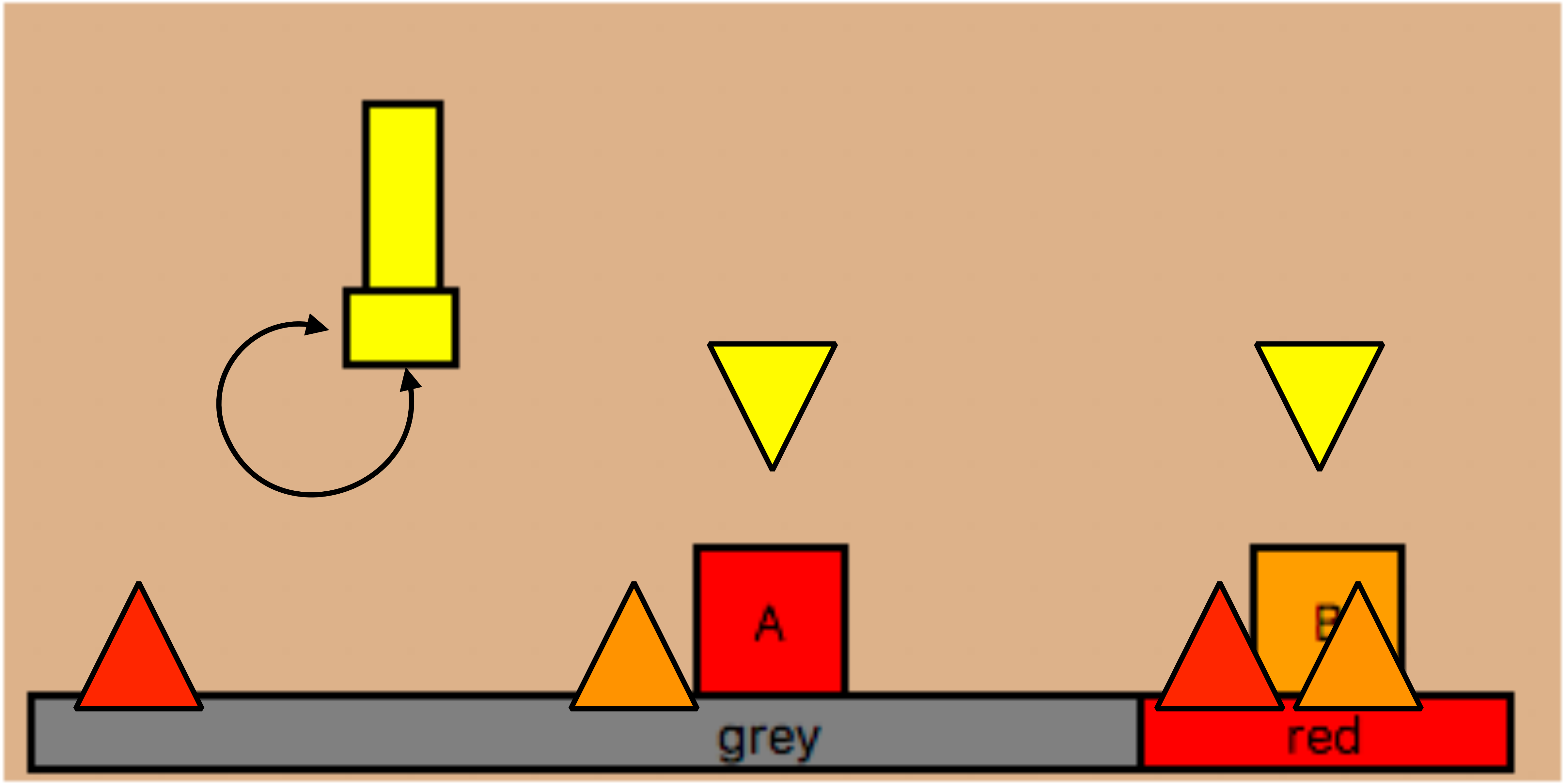
Iteration 2 - 54 stream evaluations



Incremental: Sampling Iteration 2

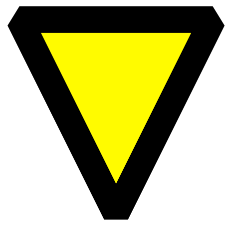
Iteration 2 - 54 stream evaluations

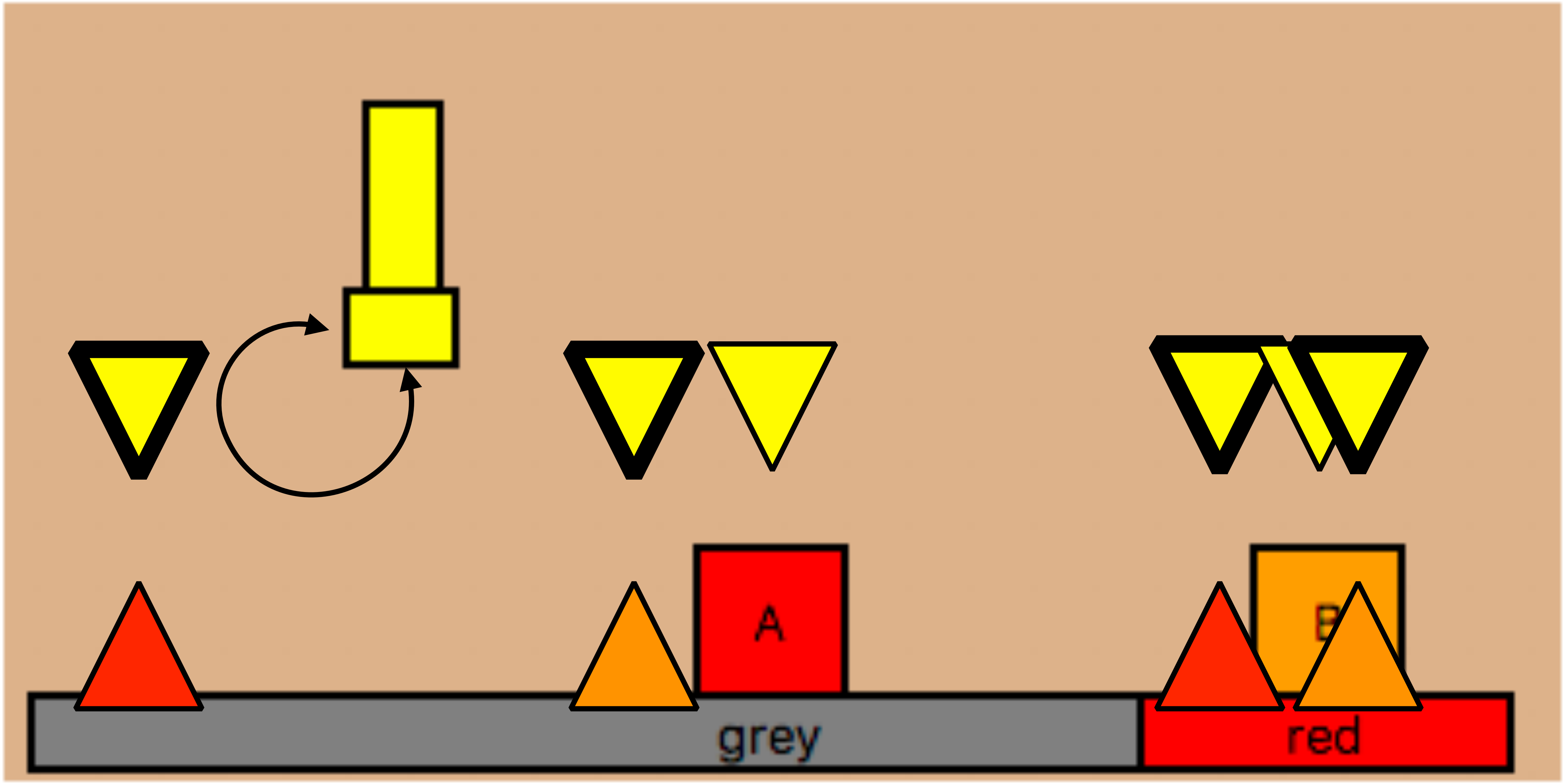
- **Sampled:**



Incremental: Sampling Iteration 2

Iteration 2 - 54 stream evaluations

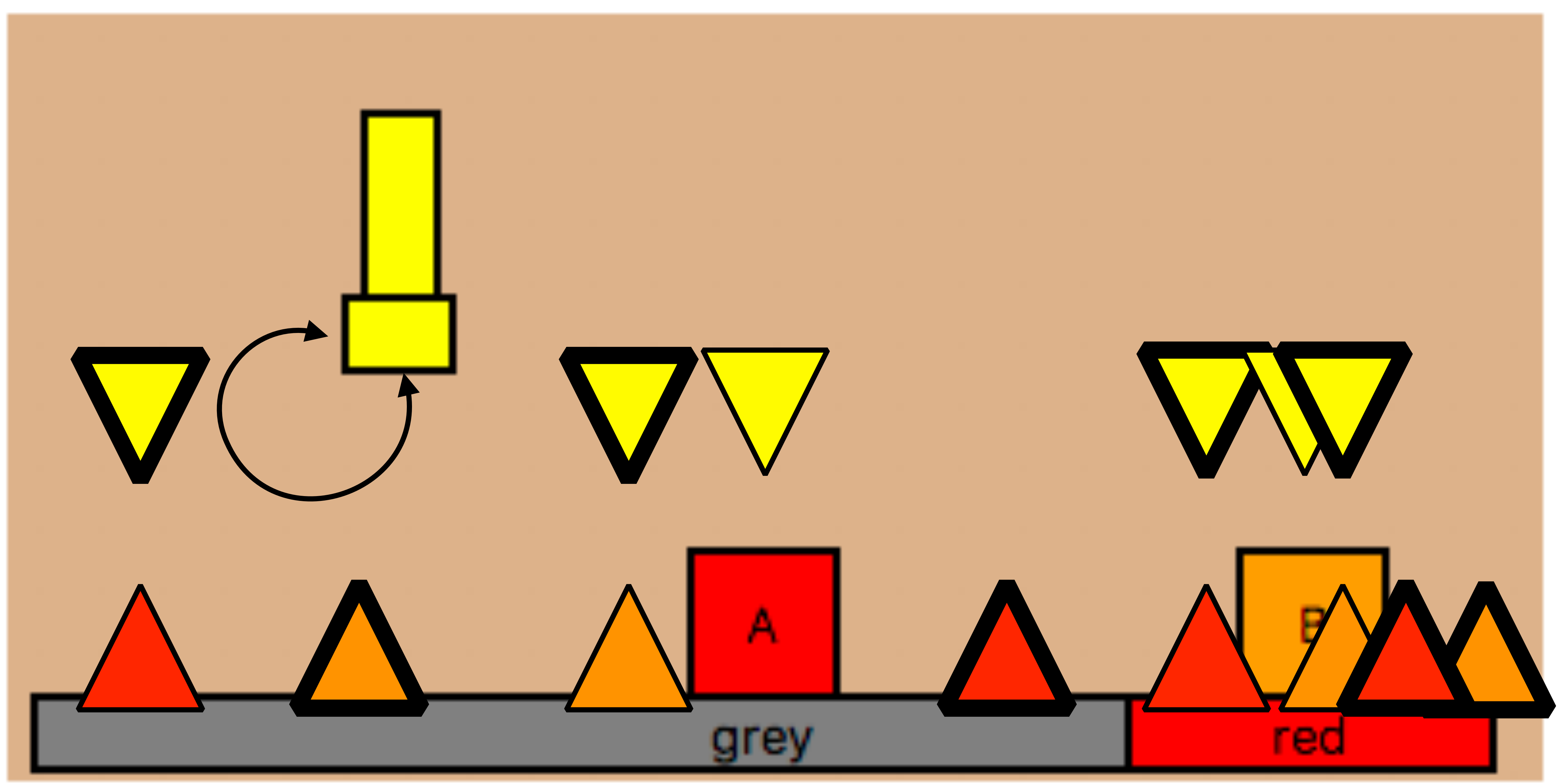
- **Sampled:**
  - 4 new robot configurations: 



Incremental: Sampling Iteration 2

Iteration 2 - 54 stream evaluations

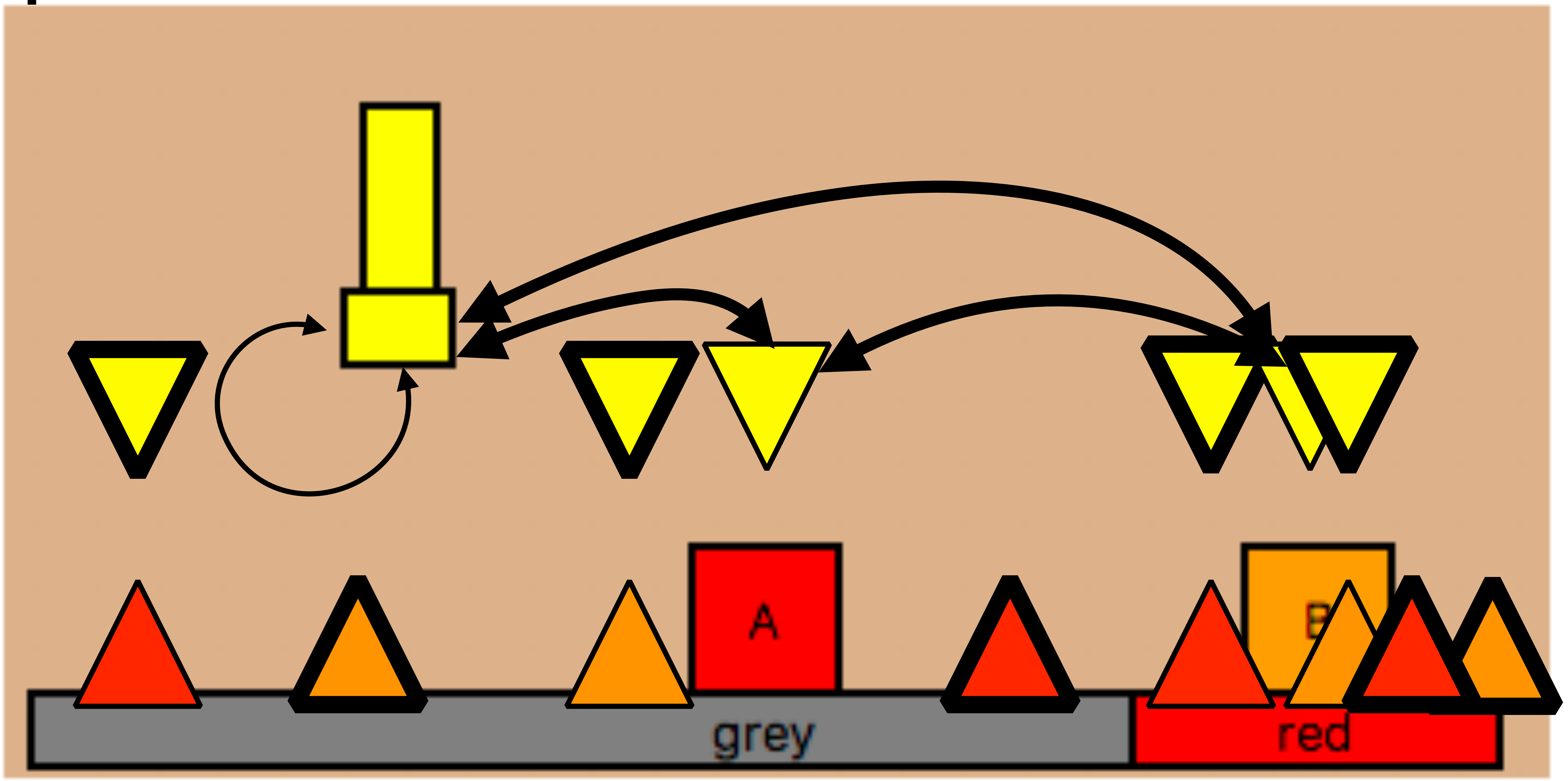
- **Sampled:**
  - 4 new robot configurations:
  - 4 new block poses:



Incremental: Sampling Iteration 2

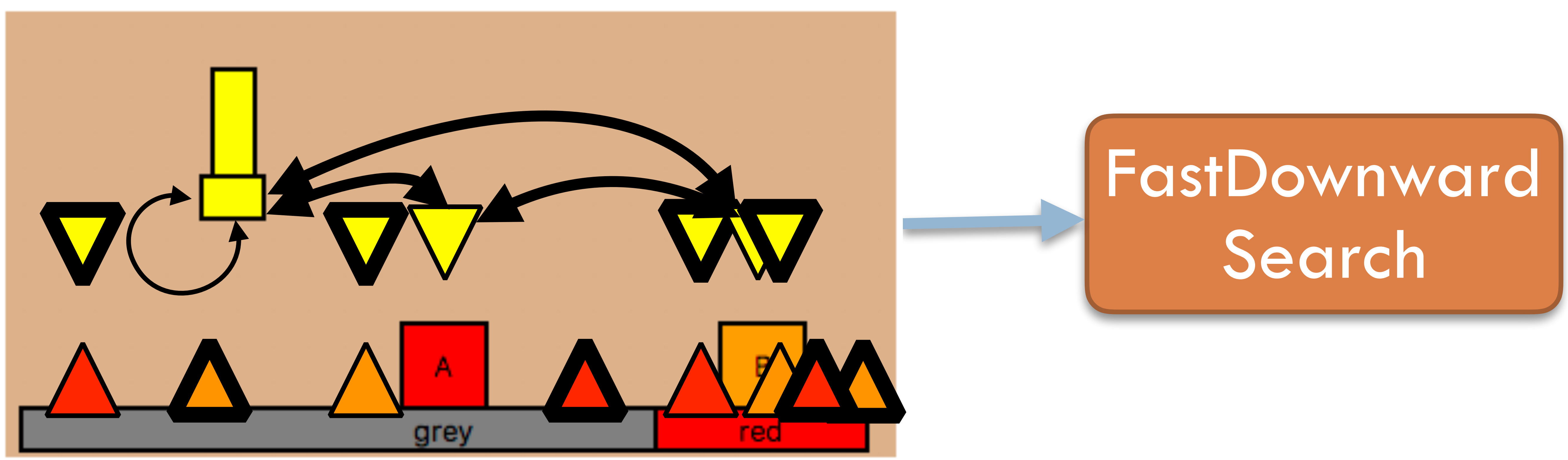
Iteration 2 - 54 stream evaluations

- **Sampled:**
  - 4 new robot configurations:
  - 4 new block poses:
  - 10 new trajectories:



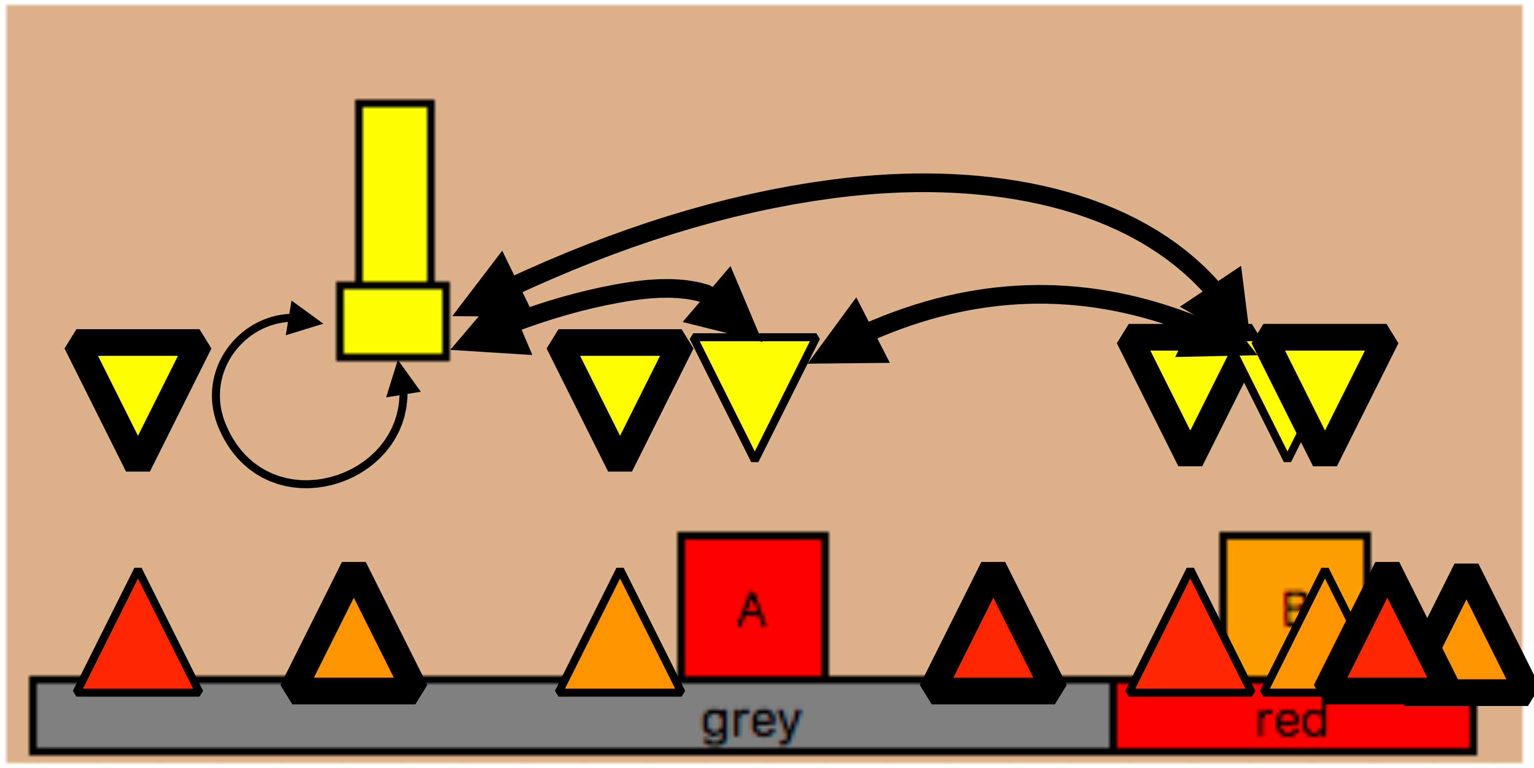
Incremental: Search Iteration 2

- Pass current discretization to FastDownward
- If **infeasible**, the current set of samples is insufficient



Incremental: Search Iteration 2

- Pass current discretization to FastDownward
- If **infeasible**, the current set of samples is insufficient



FastDownward Search → Still infeasible!

Incremental Example: Iterations 3-4

Iteration 3 - 118 stream evaluations
Iteration 4 - 182 stream evaluations

Solution:
1) **move** [-7.5 5.] [[-7.5 5.], [-7.5 5.], [7.5 5.], [7.5 2.5]] [7.5 2.5]
2) **pick B** [7.5 0.] [0. -2.5] [7.5 2.5]
3) **move** [7.5 2.5] [[7.5 2.5], [7.5 5.], [10.97 5.], [10.97 2.5]] [10.97 2.5]
4) **place B** [10.97 0.] [0. -2.5] [10.97 2.5]
5) **move** [10.97 2.5] [[10.97 2.5], [10.97 5.], [0. 5.], [0. 2.5]] [0. 2.5]
6) **pick A** [0. 0.] [0. -2.5] [0. 2.5]
7) **move** [0. 2.5] [[0. 2.5], [0. 5.], [7.65 5.], [7.65 2.5]] [7.65 2.5]
8) **place A** [7.65 0.] [0. -2.5] [7.65 2.5]

- **Drawback** - many unnecessary samples produced
  - **Computationally expensive** to generate
  - Induces **large discrete-planning problems**

Optimistic Stream Outputs

- Many TAMP streams are exceptionally **expensive**
  - Inverse kinematics, motion planning, collision checking
- **Only** query streams that are **identified** as useful
  - Plan with **optimistic hypothetical** outputs
- Inductively create **unique placeholder** output objects for each stream instance (has **#** as its prefix)

Optimistic evaluations:

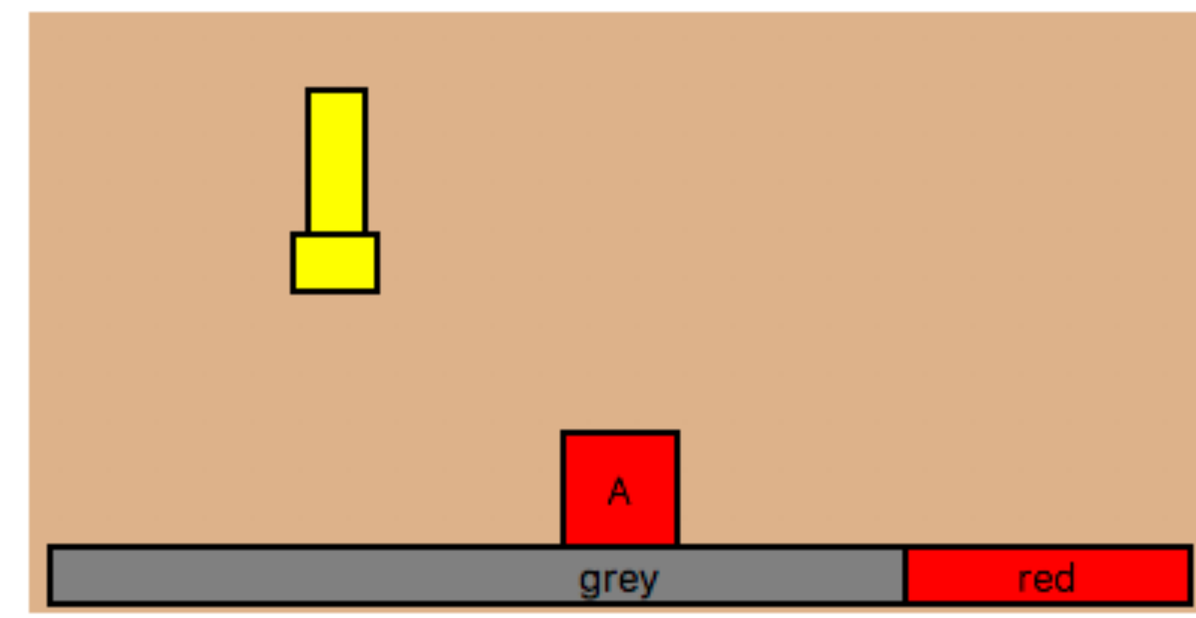
1. **s-region:**(b0, red)->(<u>**#p0**</u>)
2. **s-ik:**(b0, [0. 0.], [0. -2.5])->(<u>**#q0**</u>),
3. **s-ik:**(b0, <u>**#p0**</u>, [0. -2.5]) ->(<u>**#q2**</u>)

Focused Algorithm

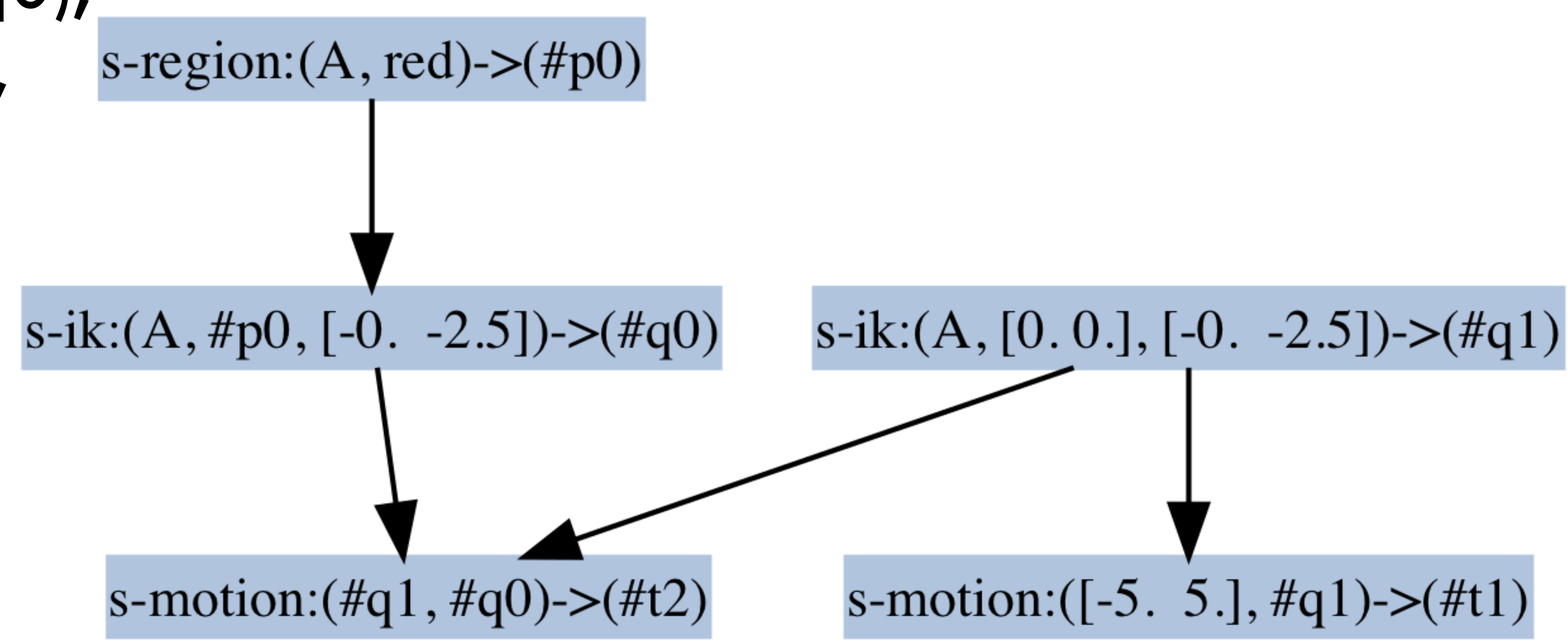
- **Lazily** plan using optimistic outputs **before** real outputs
- **Recover** set of streams used by the optimistic plan
- Repeat:
  1. Construct active **optimistic** objects
  2. **Search** with **real & optimistic** objects
  3. If **only real objects** used, **return plan**
  4. **Sample** used streams
  5. **Disable** used streams

Focused Example 1

Optimistic Plan:
move([-5. 5.], #t0, #q0), **pick**(A, [0. 0.], [-0. -2.5], #q0), **move**(#q0, #t2, #q1), **place**(A, #p0, [-0. -2.5], #q1)



Constraints:
(kin, A, #q0, #p0, [-0. -2.5]),
(kin, A, #q1, [0. 0.], [-0. -2.5]),
(motion, [-5. 5.], #t1, #q1),
(motion, #q1, #t2, #q0),
(contain, A, #p0, red),



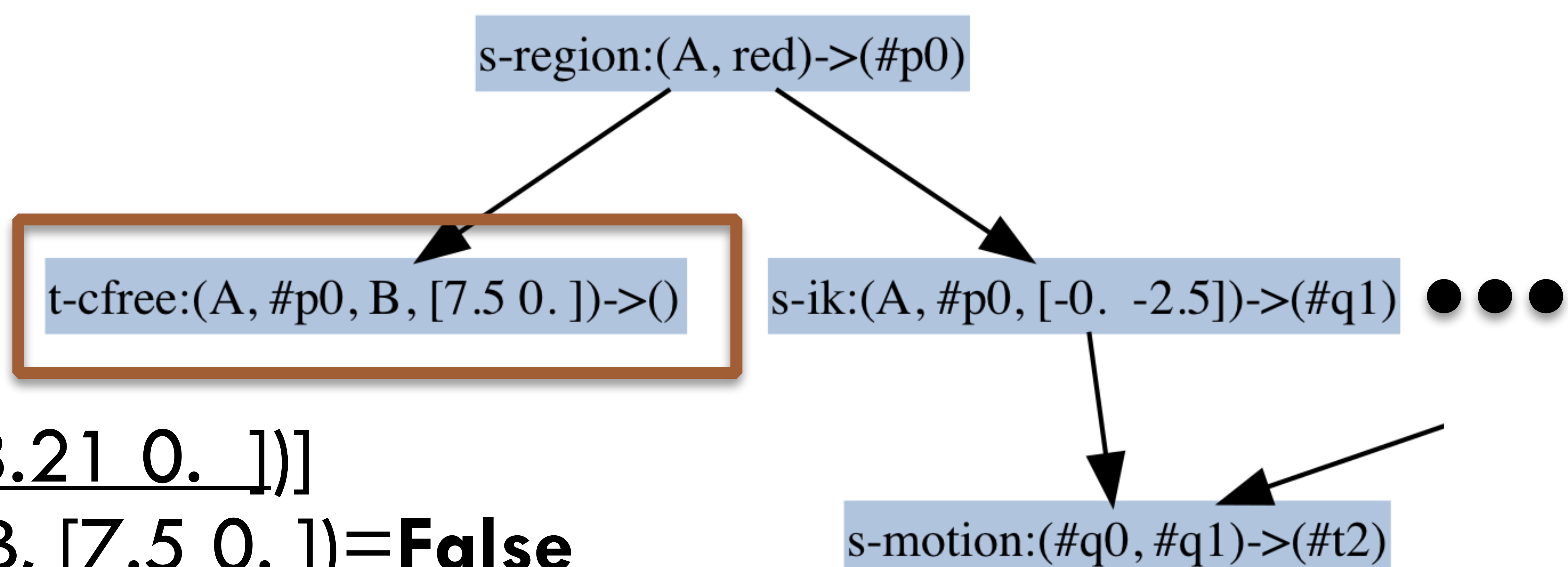
Focused Example 2: Iteration 1

Optimistic Plan:

move([-5. 5.], #t0, #q0), **pick**(A, [0. 0.], [-0. -2.5], #q0), **move**(#q0, #t2, #q1), **place**(A, #p0, [-0. -2.5], #q1)

Constraints:

(cfree, A, #p0, B, [7.5 0.]), (contain, A, #p0, red), (kin, A, #q0, [0. 0.], [-0. -2.5]), (kin, A, #q1, #p0, [-0. -2.5]), (motion, #q0, #t2, #q1), (motion, [-5. 5.], #t0, #q0)



Stream evaluations:

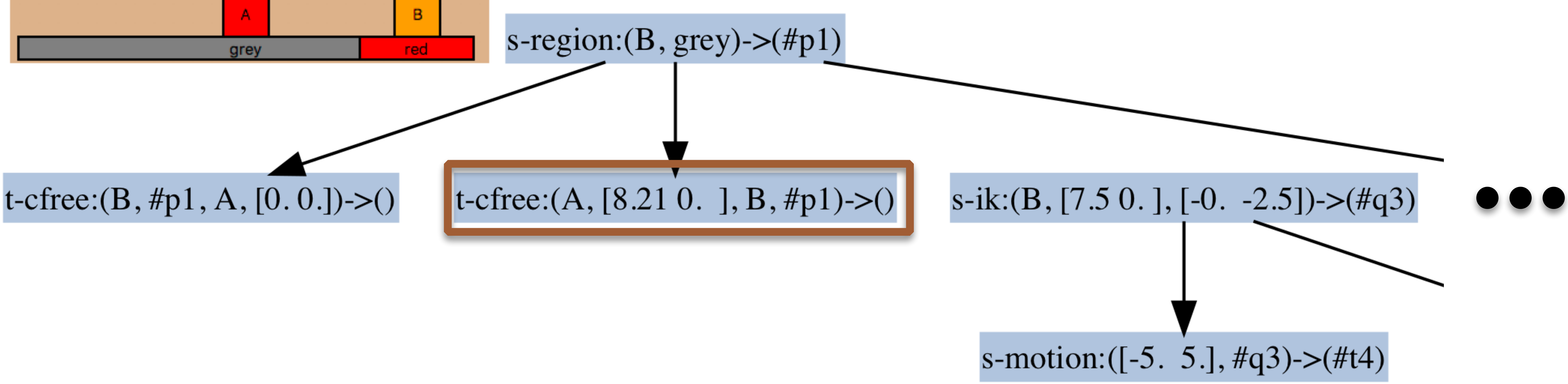
1. **s-region:**(A, red)->[([8.21 0.])]
2. **t-cfree:**(A, [8.21 0.], B, [7.5 0.])=**False**

These stream instances are **removed** from subsequent searches

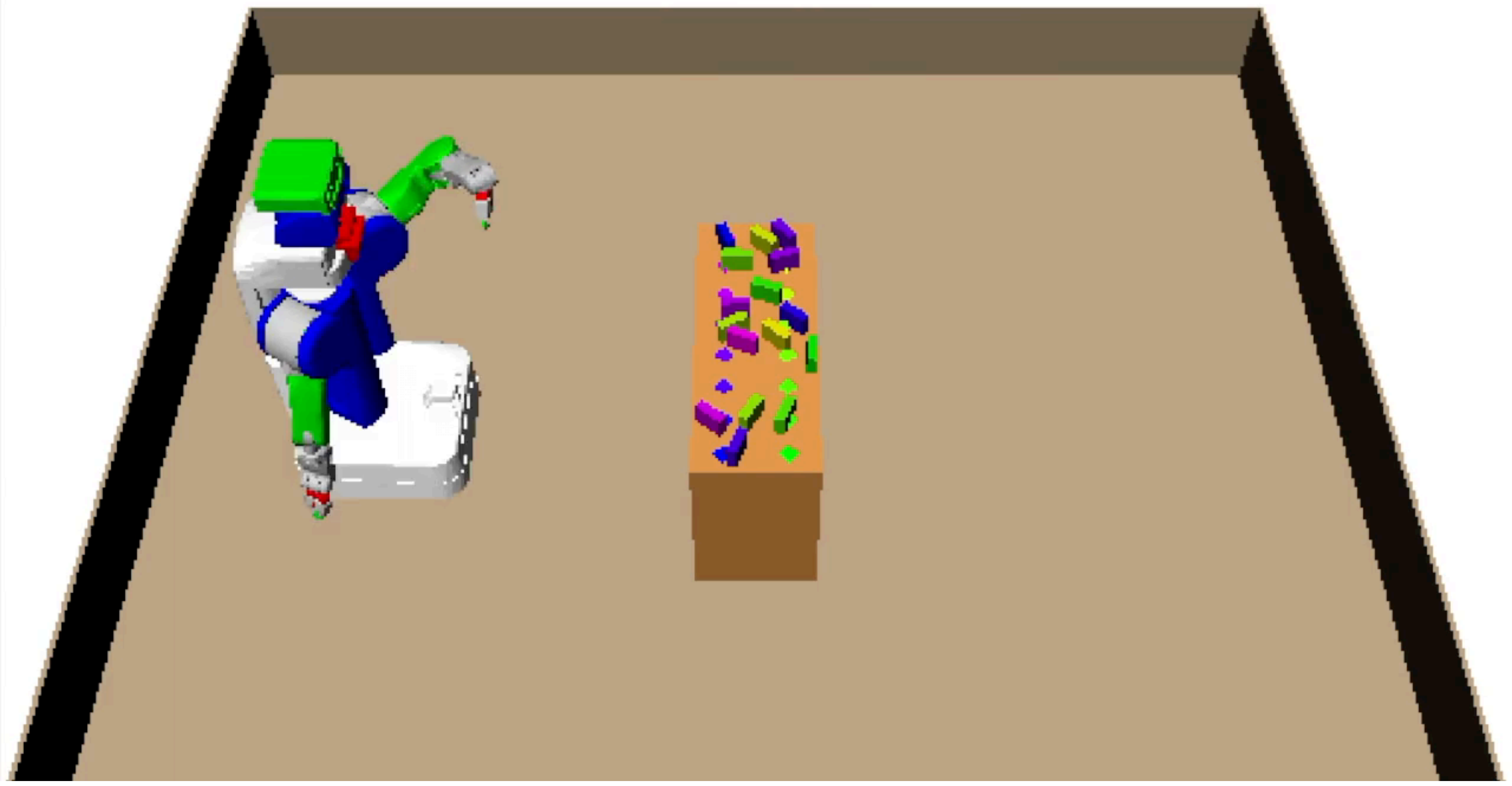
Focused Example: Iteration 2

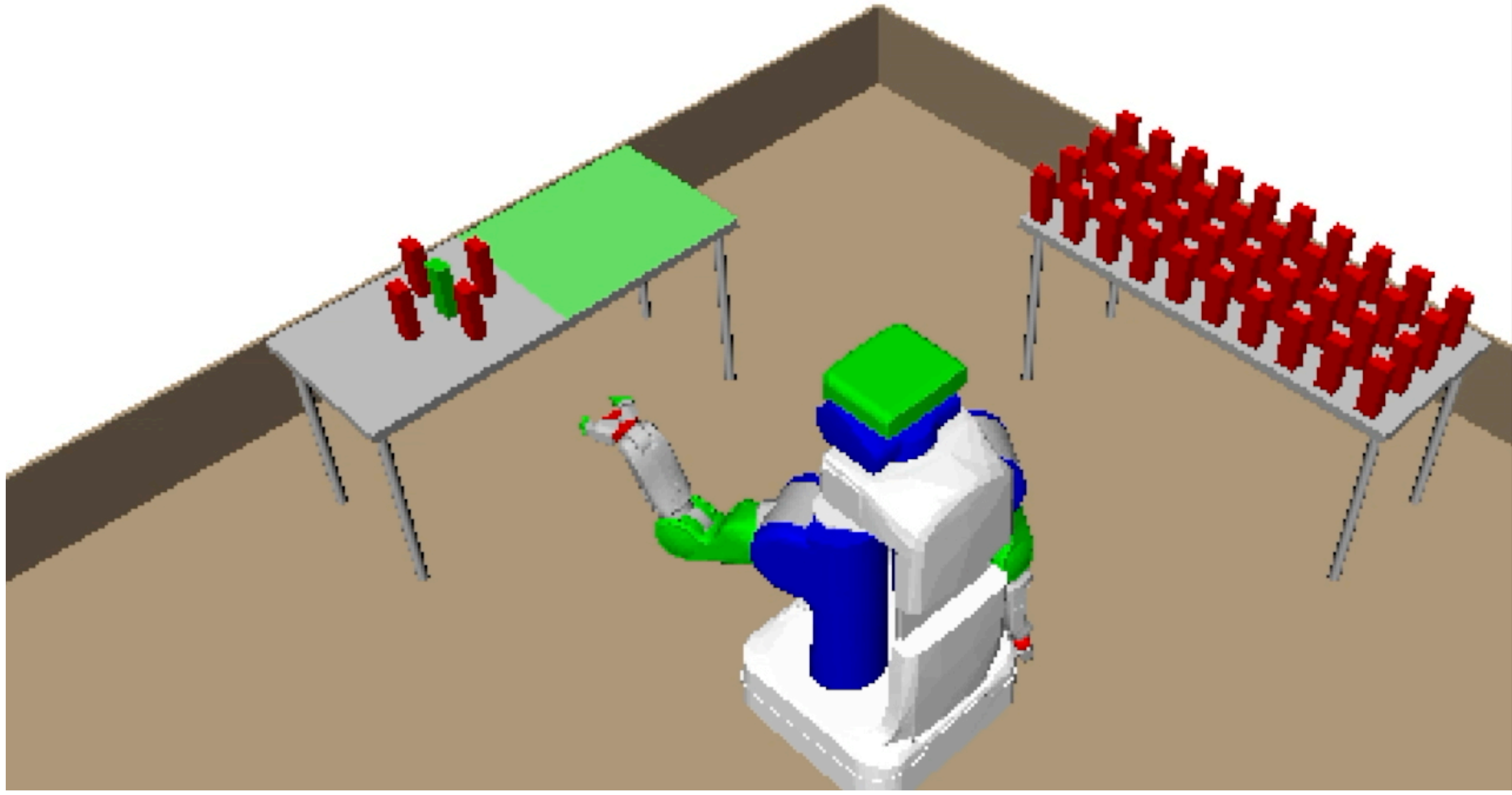
Optimistic Plan:
move([-5. 5.], #t4, #q2), **pick**(B, [7.5 0.], [-0. -2.5], #q2),
move(#q2, #t9, #q3), **place**(B, #p1, [-0. -2.5], #q3),
move(#q3, #t6, #q0), **pick**(A, [0. 0.], [-0. -2.5], #q0),
move(#q0, #t8, #q4), **place**(A, [8.21 0.], [-0. -2.5], #q4)

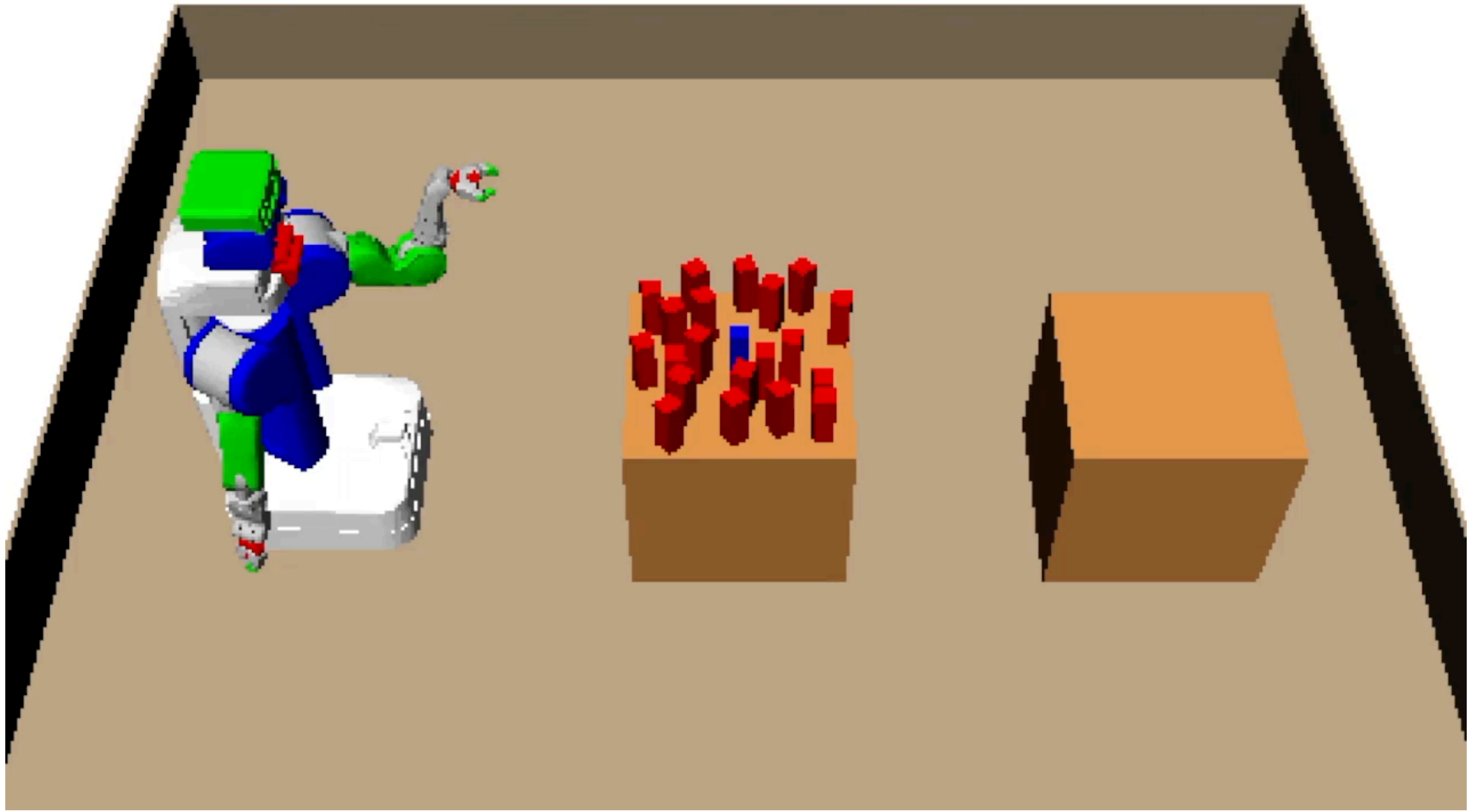
t-cfree:(A, [8.21 0.], B, [7.5 0.]) previously **failed**
t-cfree:(A, [8.21 0.], B, #p1) might **succeed**



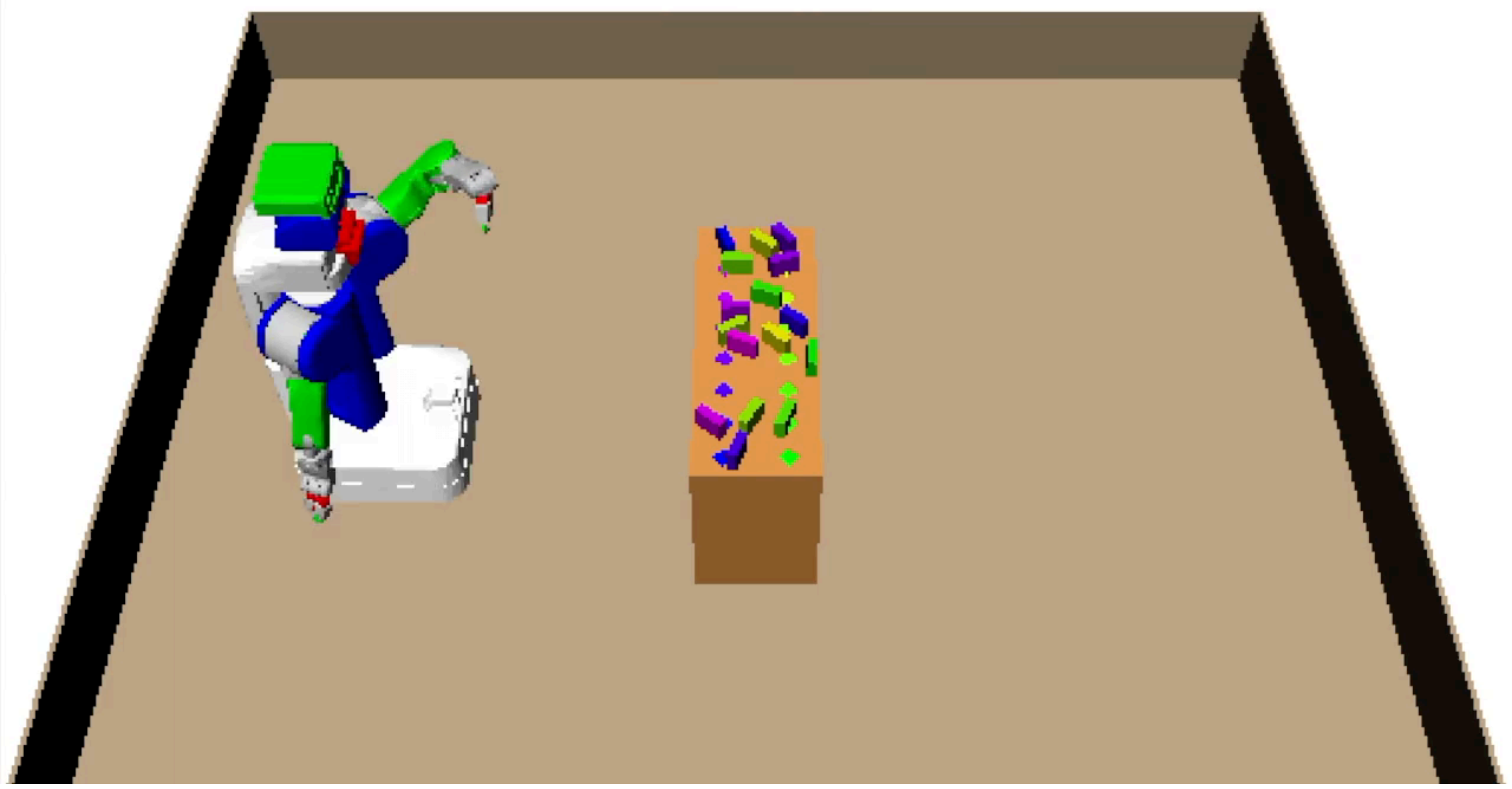
Scaling Experiments

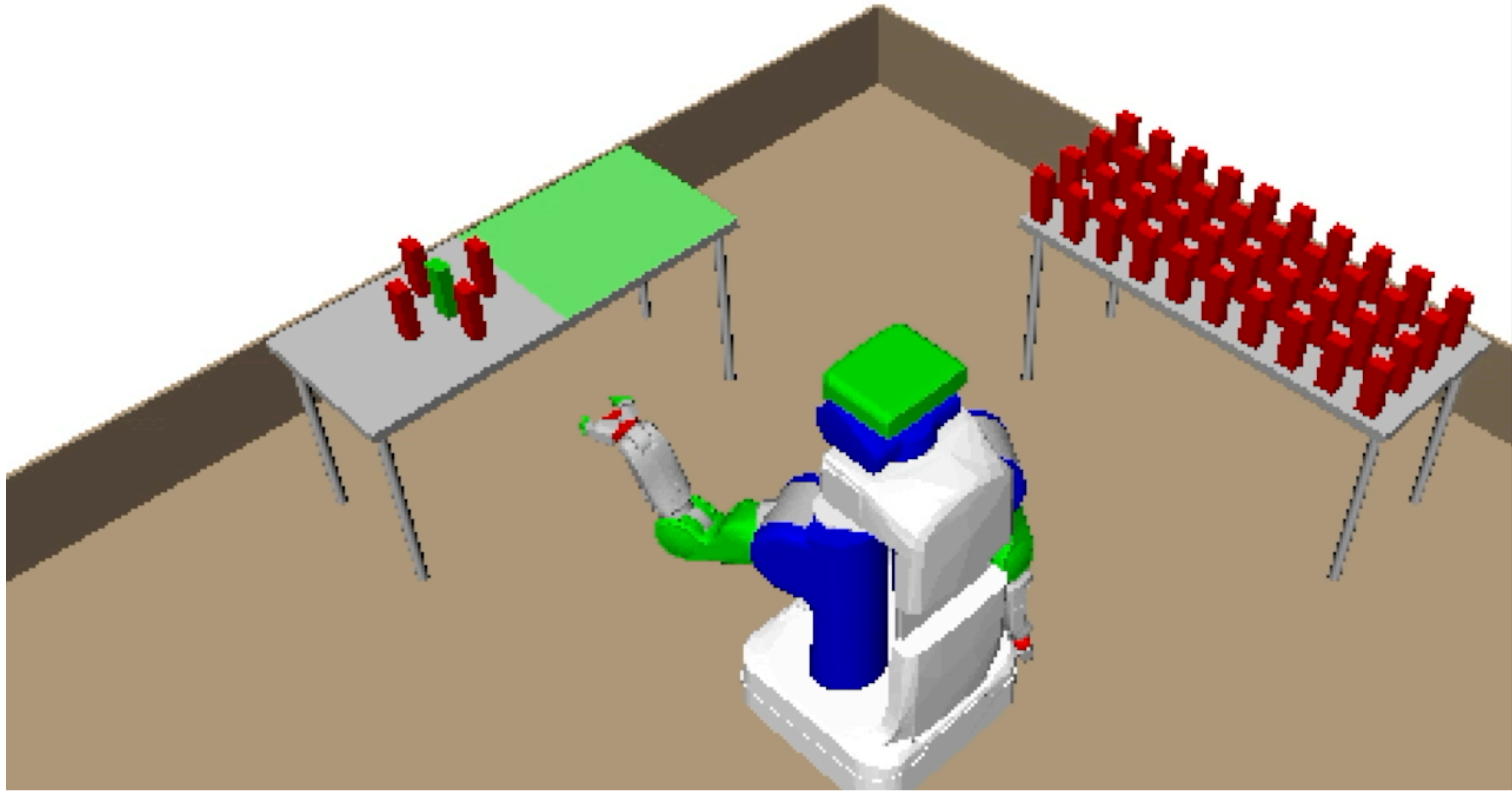


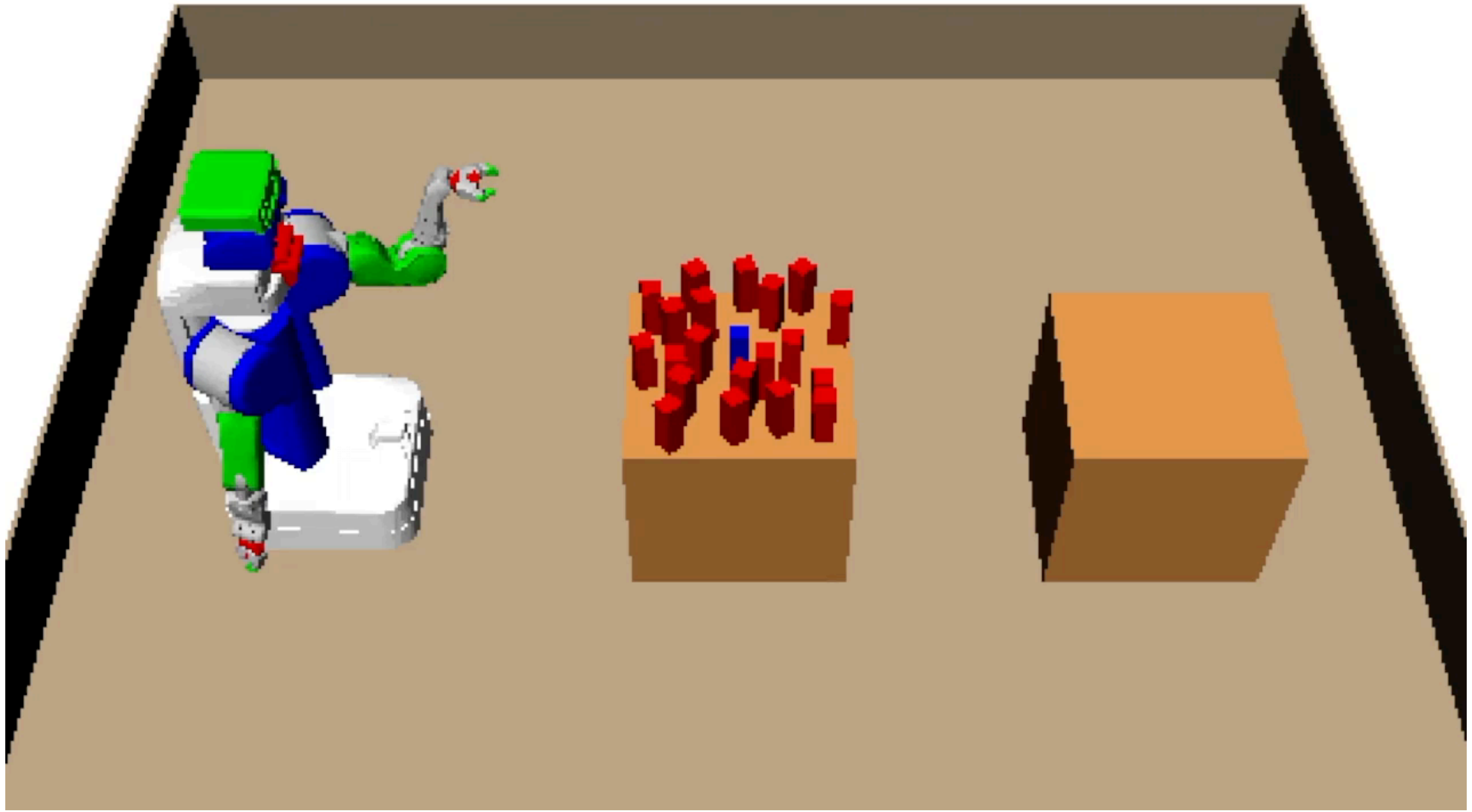




Scaling Experiments

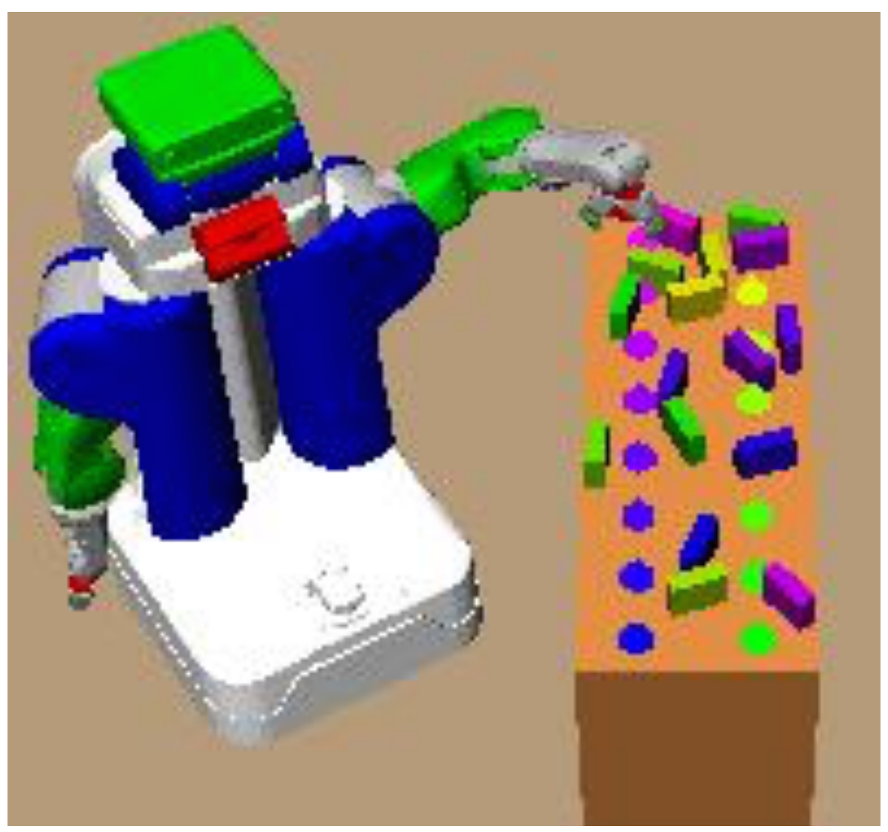


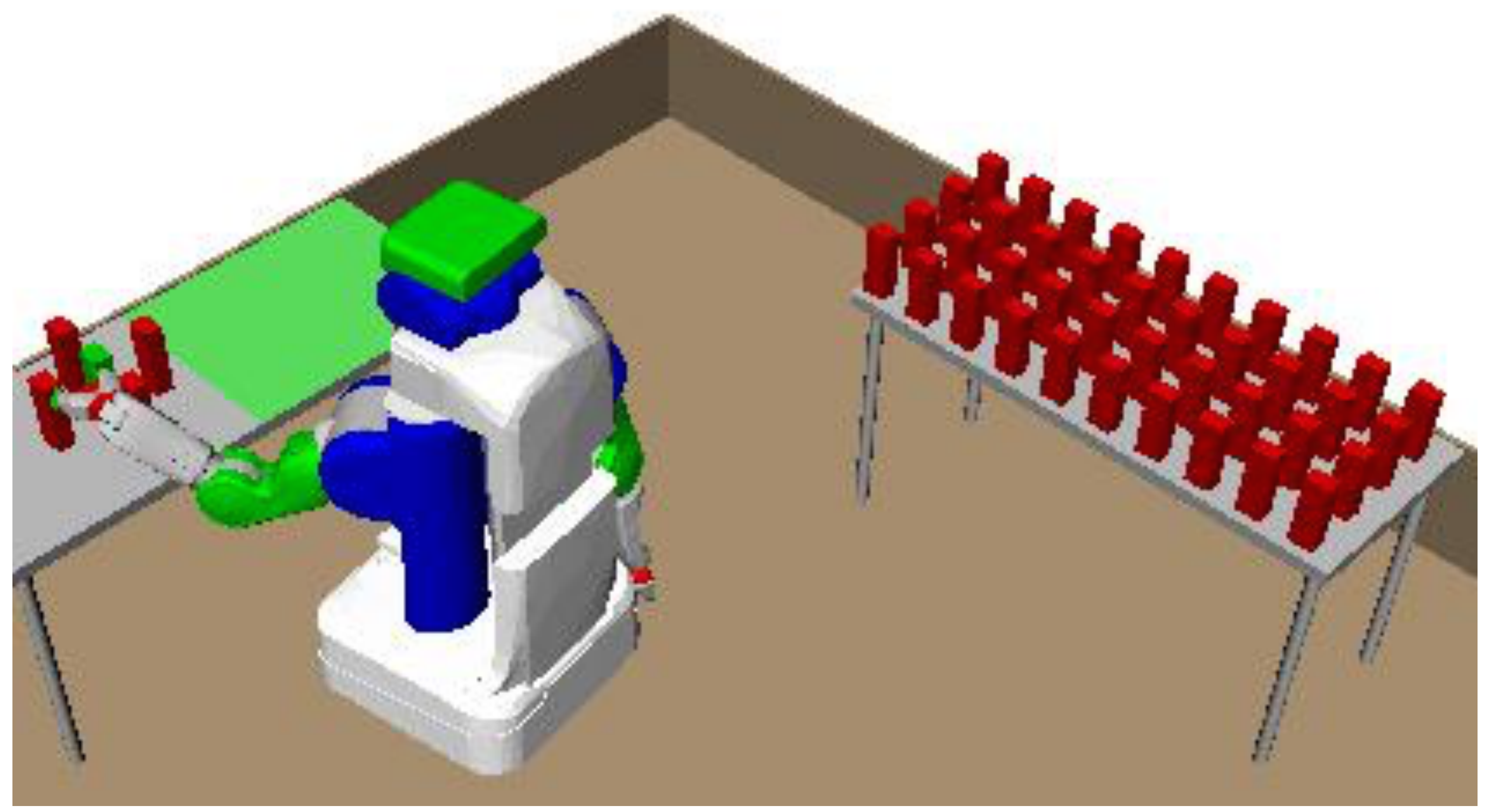


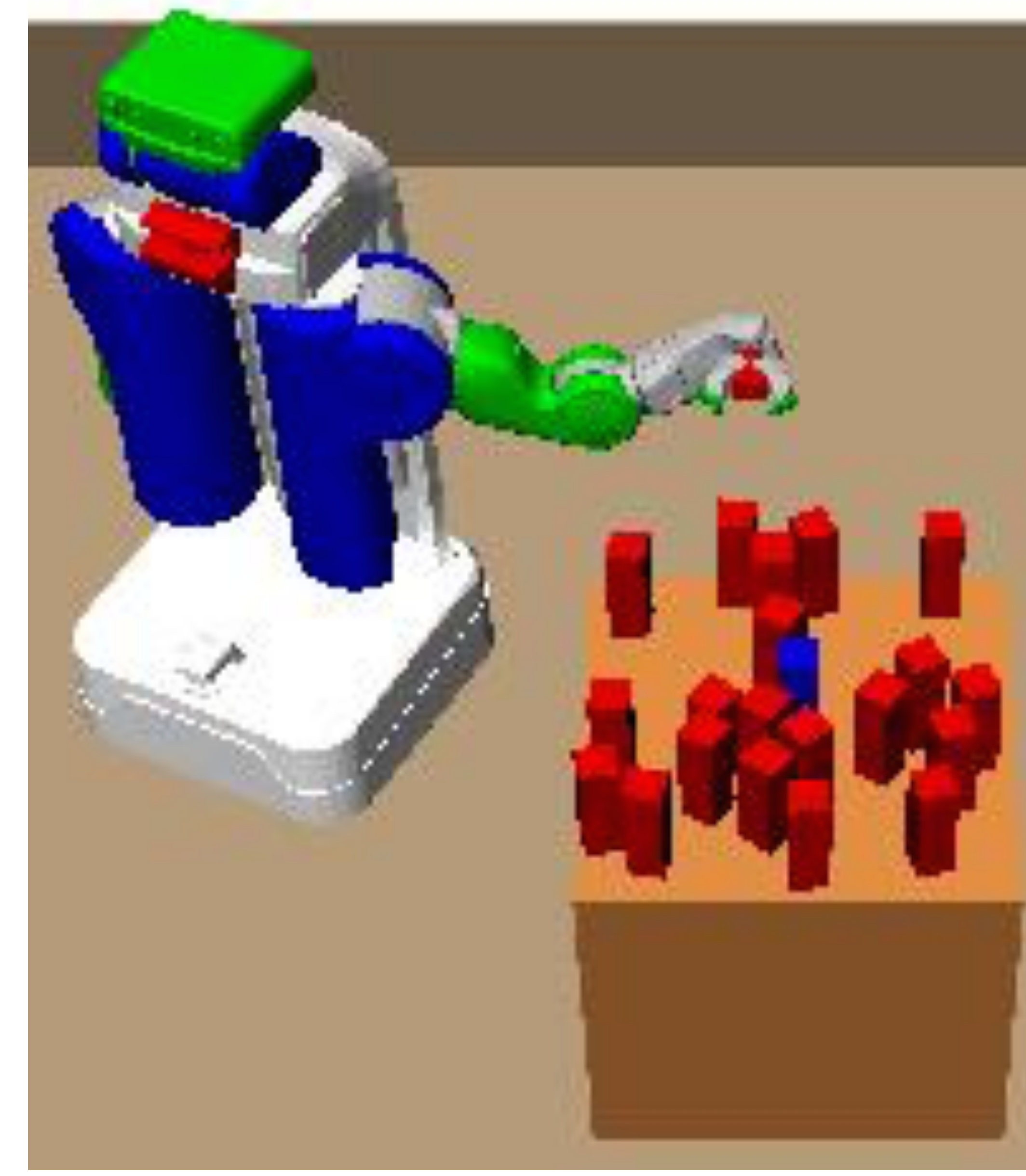


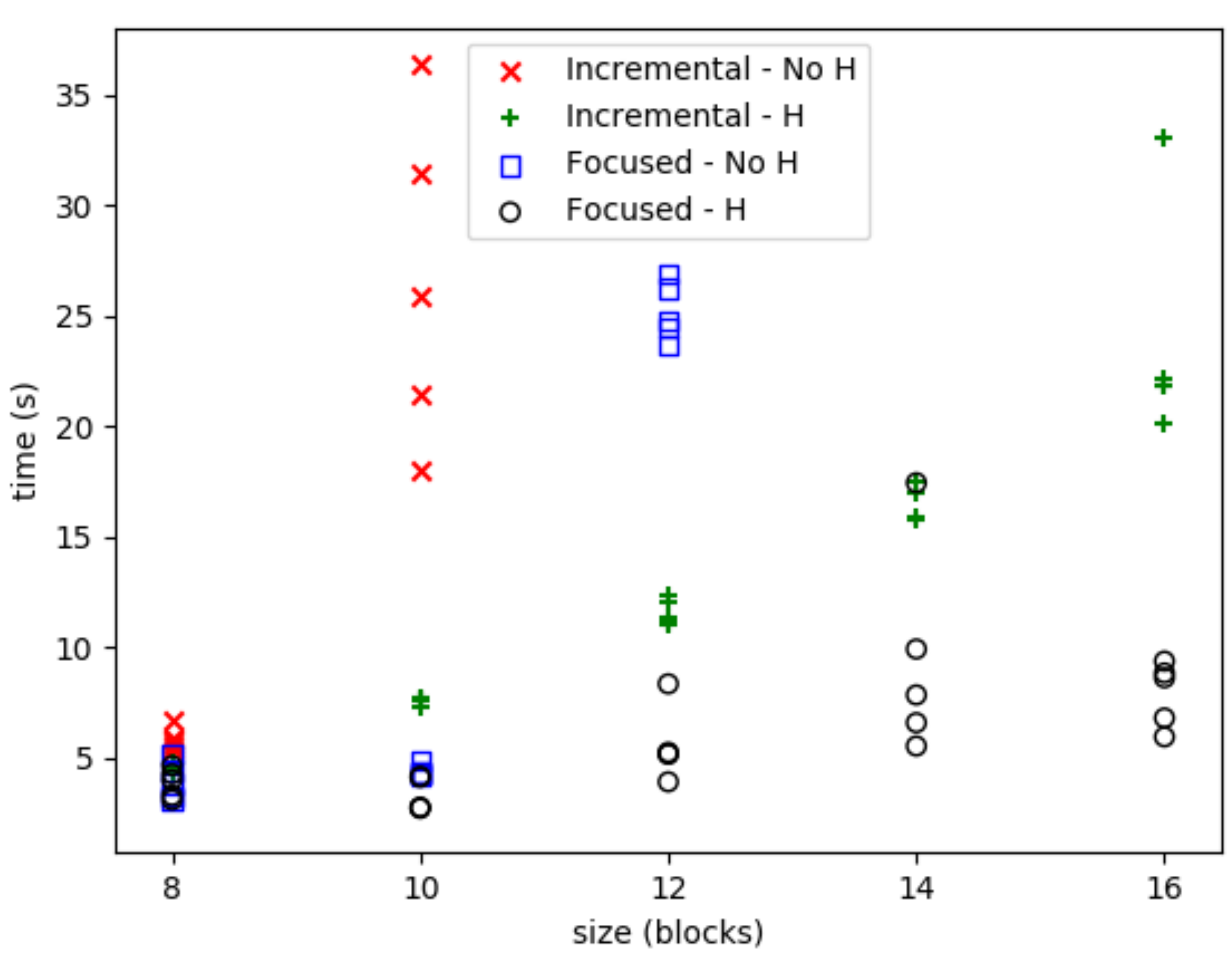
Problem Size vs Runtime

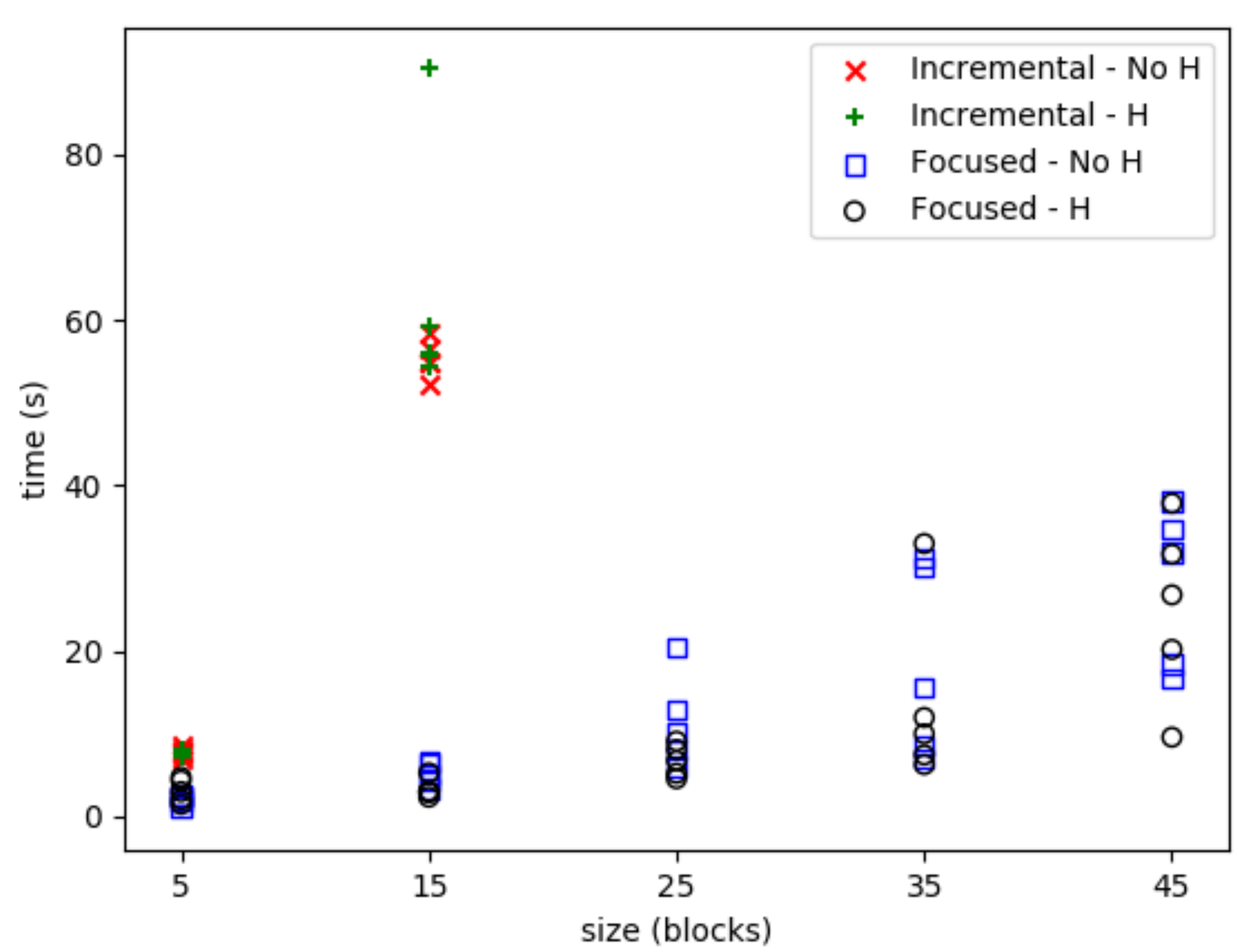
- **Focused** outperforms incremental
- **FastDownward** outperforms BFS

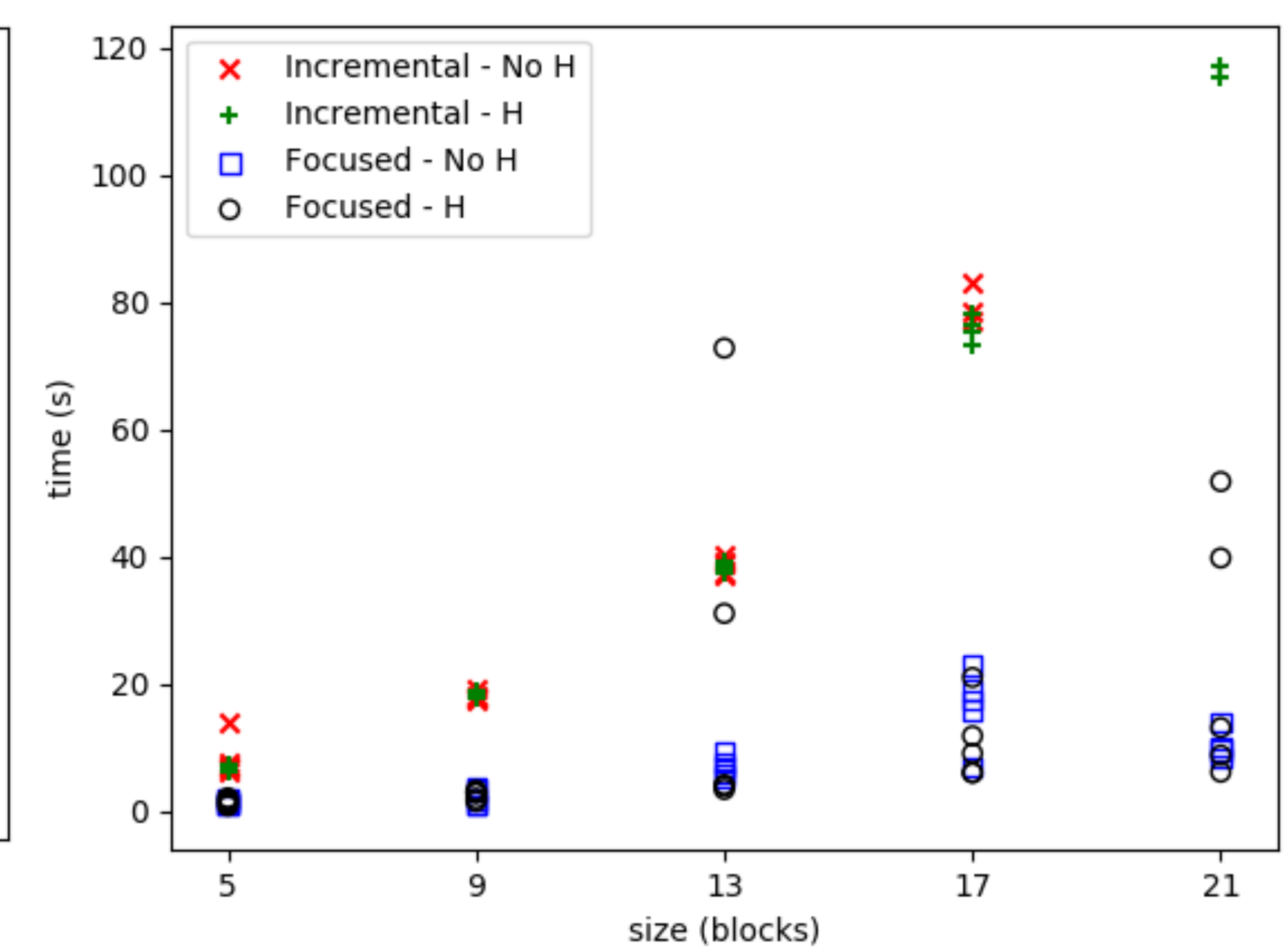










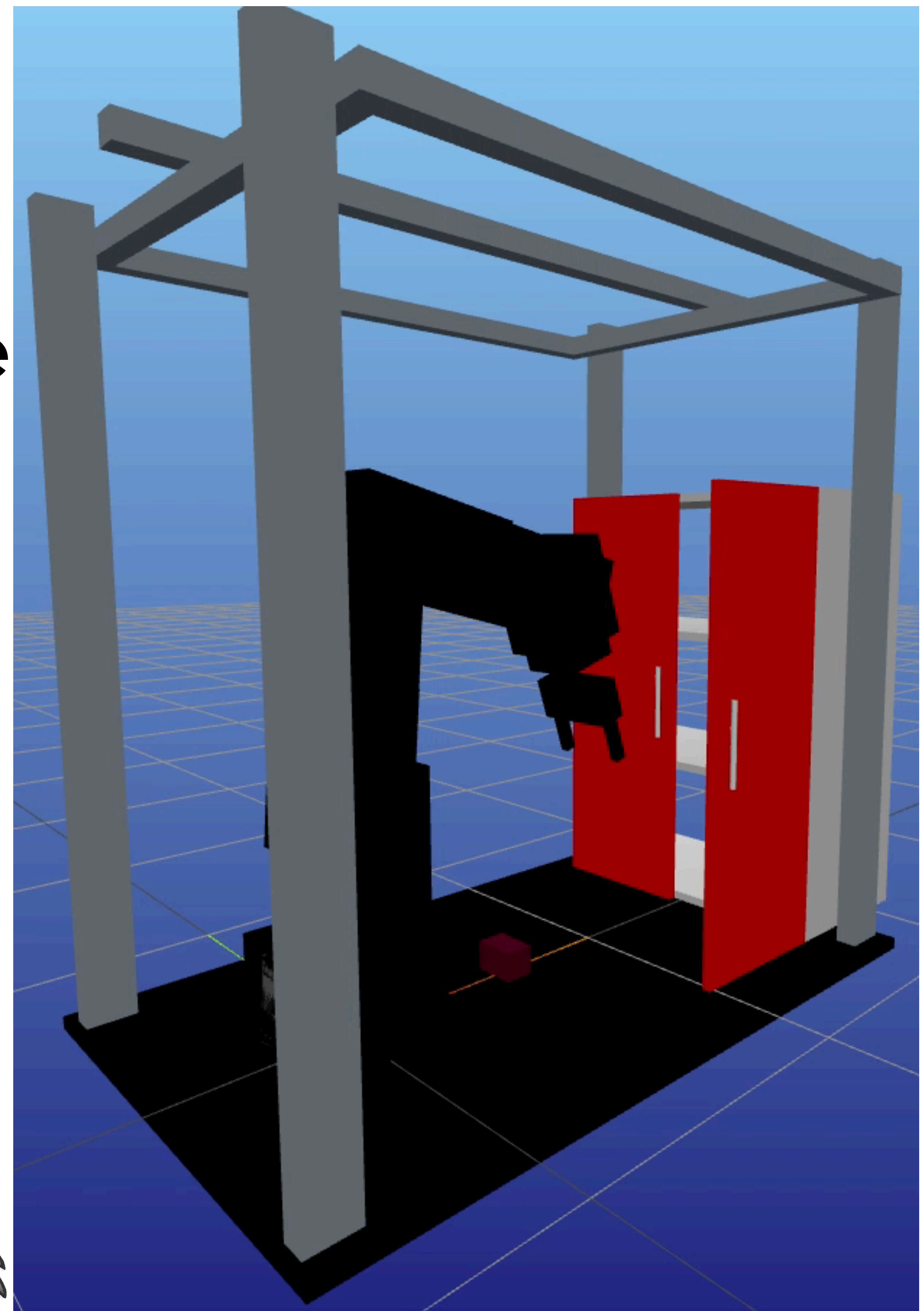


Applications: Pantry Manipulation

- Framework is **independent** of robot and robotics software
- To stow the object, the **robot decides** to open the door

DRAKE

Model-Based Design and Verification for Robotics

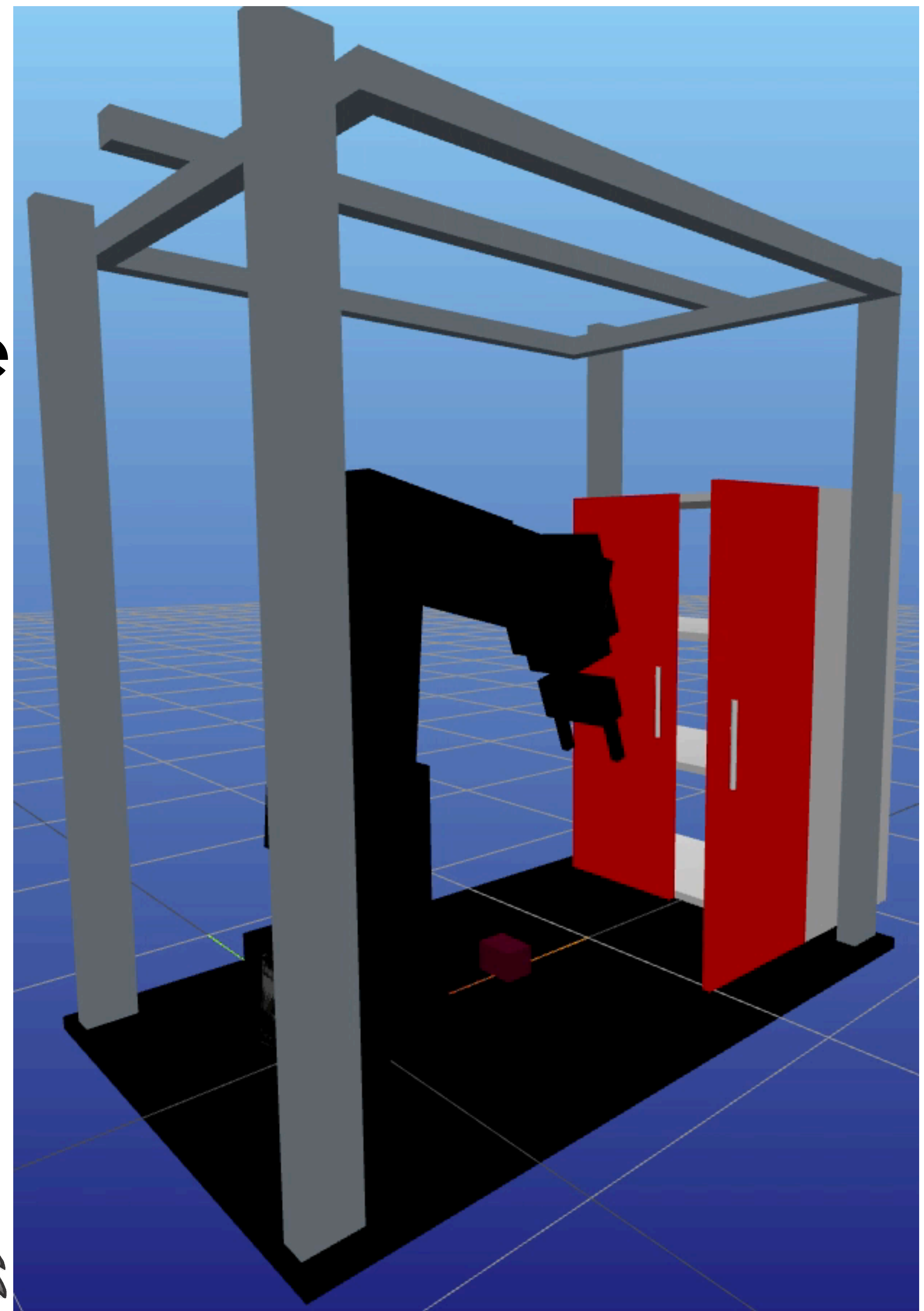


Applications: Pantry Manipulation

- Framework is **independent** of robot and robotics software
- To stow the object, the **robot decides** to open the door

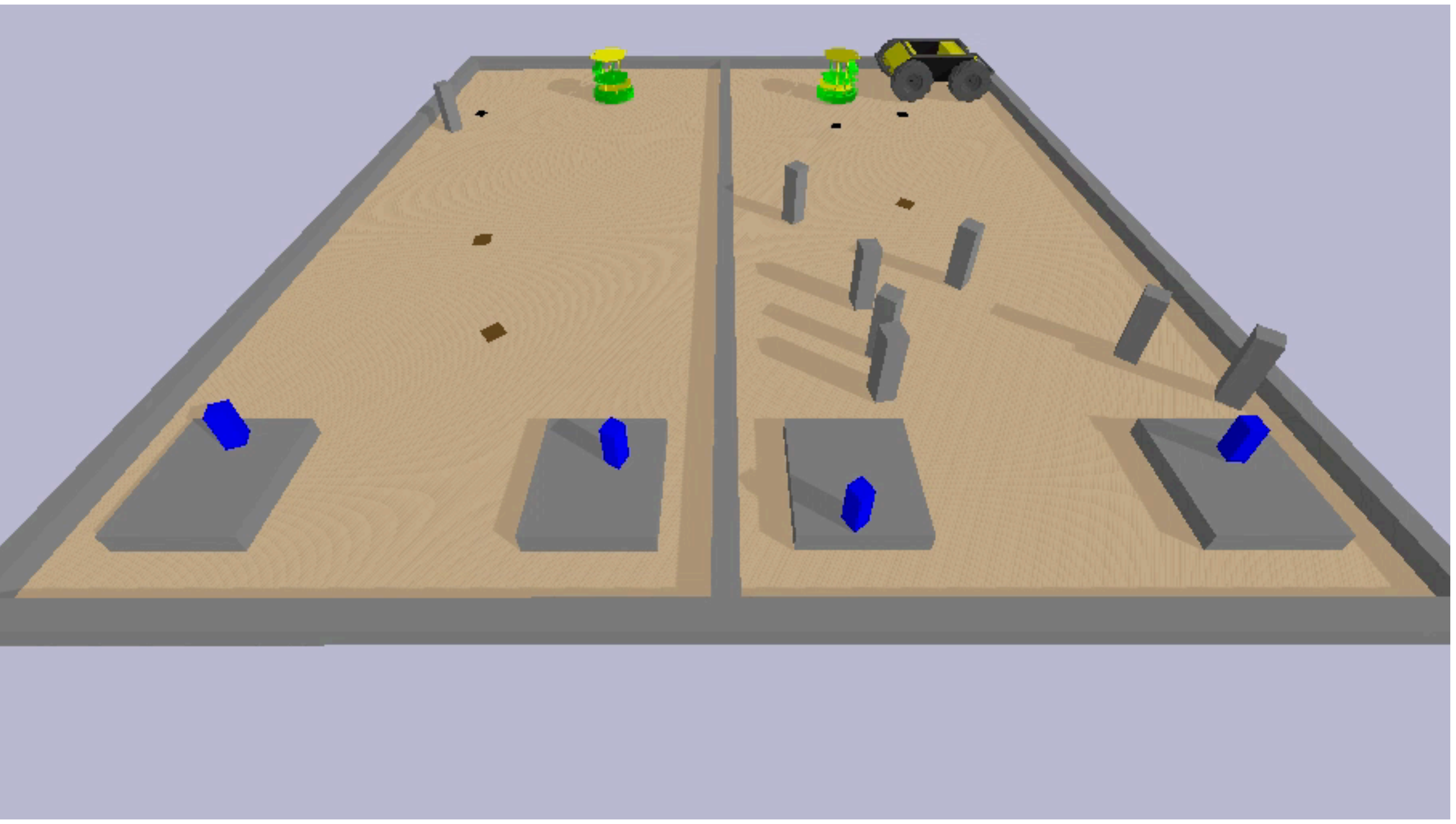
DRAKE

MODEL-BASED DESIGN AND VERIFICATION FOR ROBOTICS



Applications: Multi-Robot Planning

- **Centralized scheduling** of a team of robots
- PDDL rovers domain with **visibility** and **reachability**
- Use **temporal planners** as search subroutine (e.g. Temporal FastDownward)

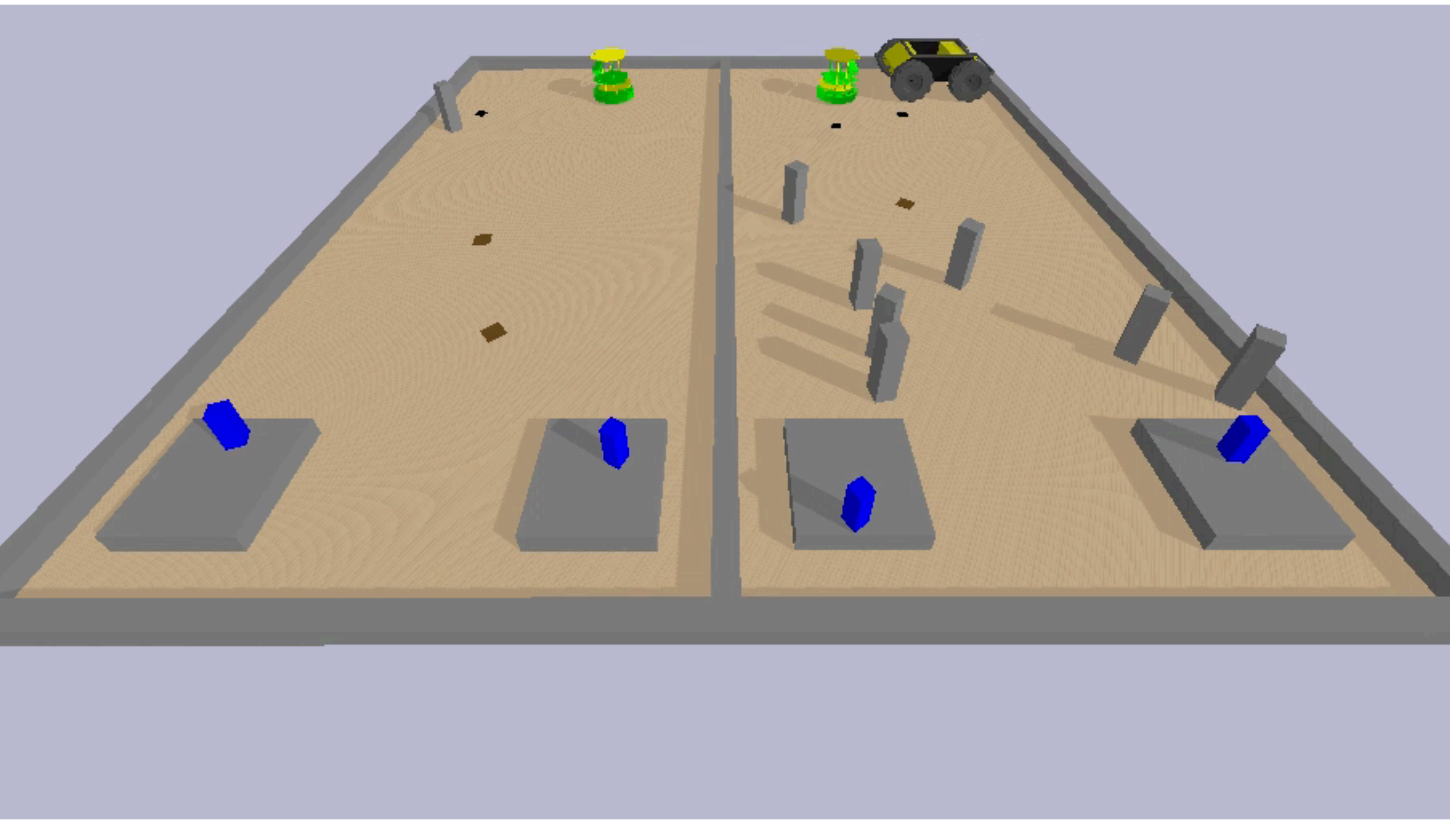


BULLET PHYSICS LIBRARY

OpenRAVE

Applications: Multi-Robot Planning

- **Centralized scheduling** of a team of robots
- PDDL rovers domain with **visibility** and **reachability**
- Use **temporal planners** as search subroutine (e.g. Temporal FastDownward)

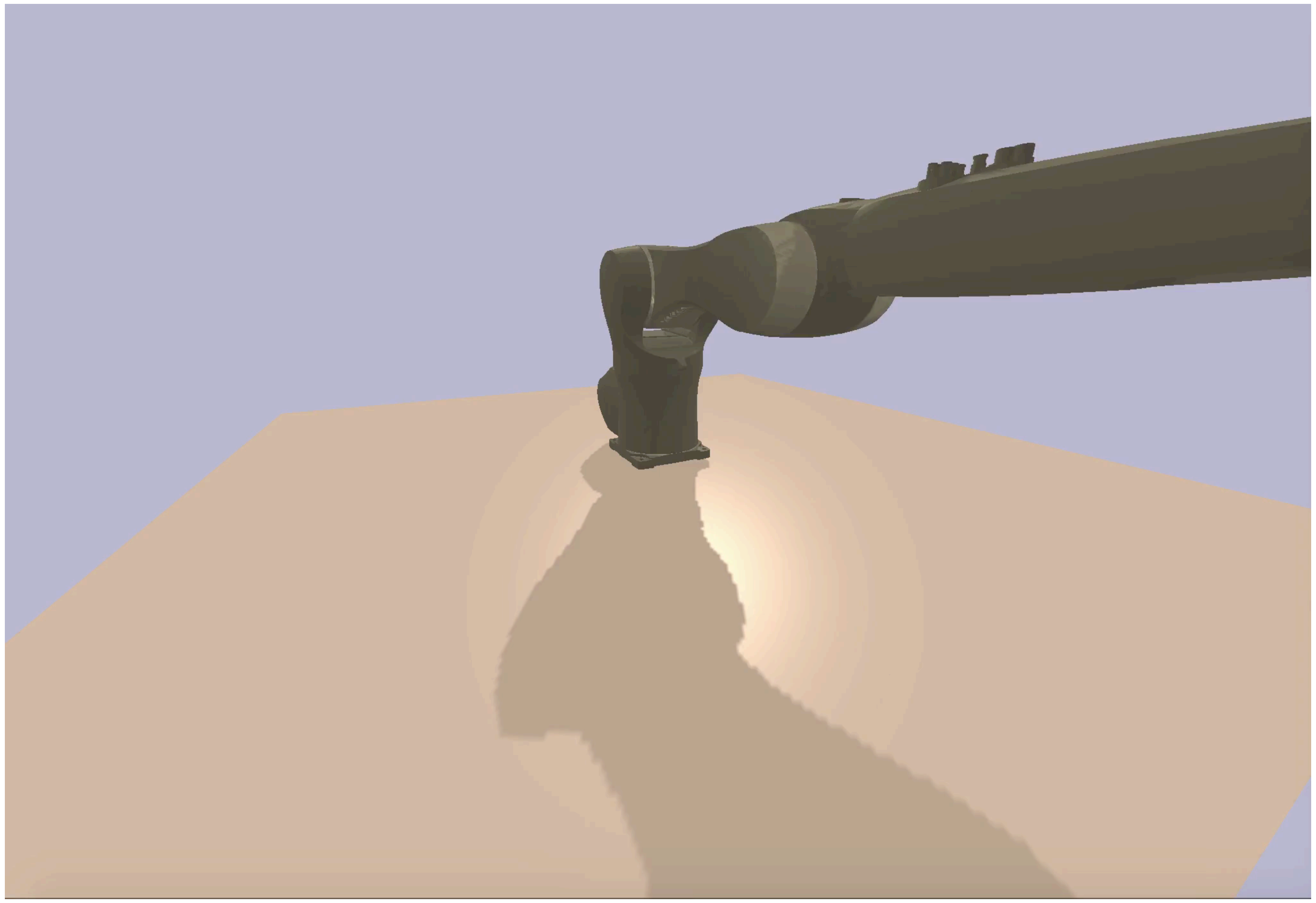


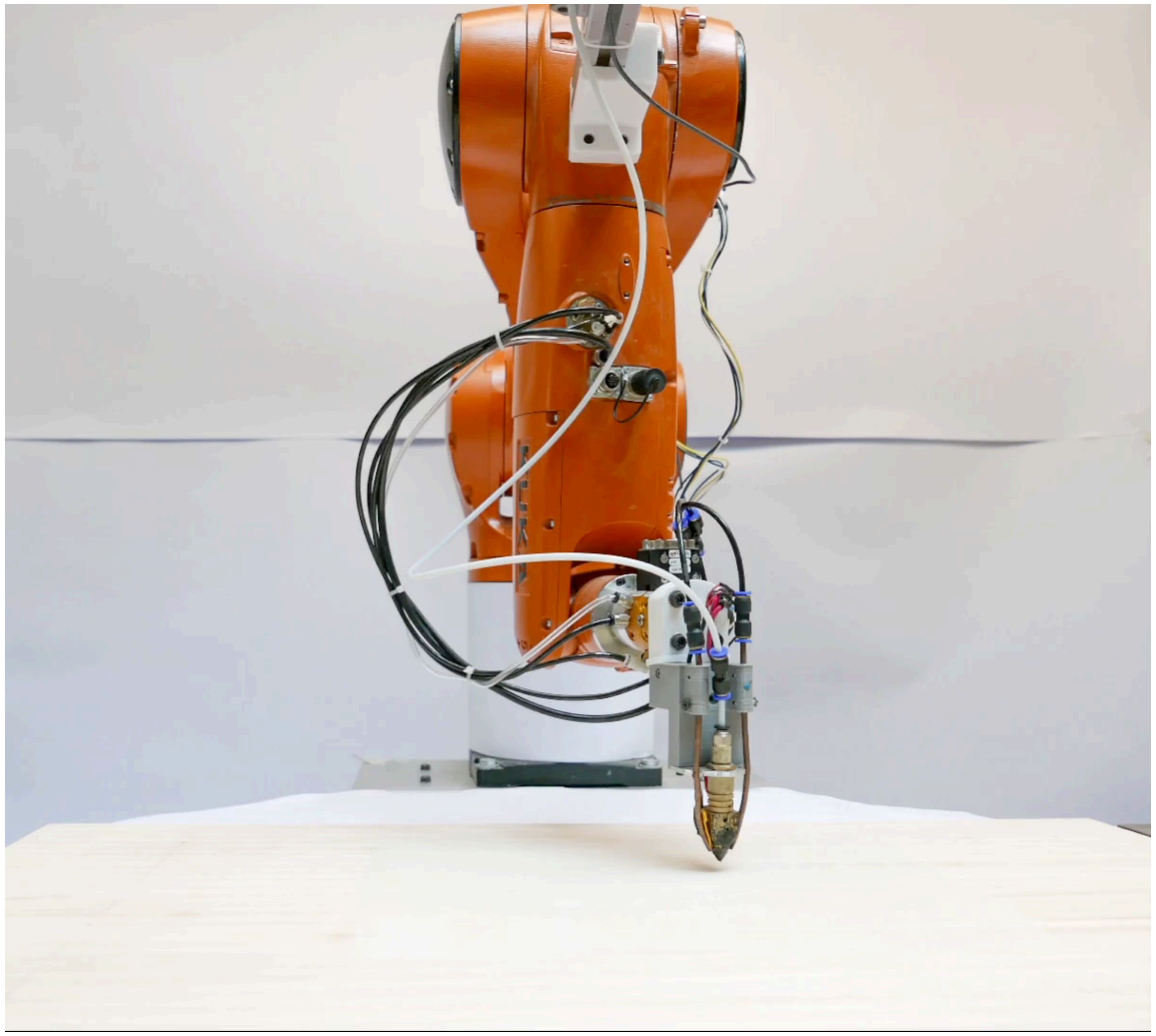
BULLET PHYSICS LIBRARY

OpenRAVE

Applications: Automated Fabrication

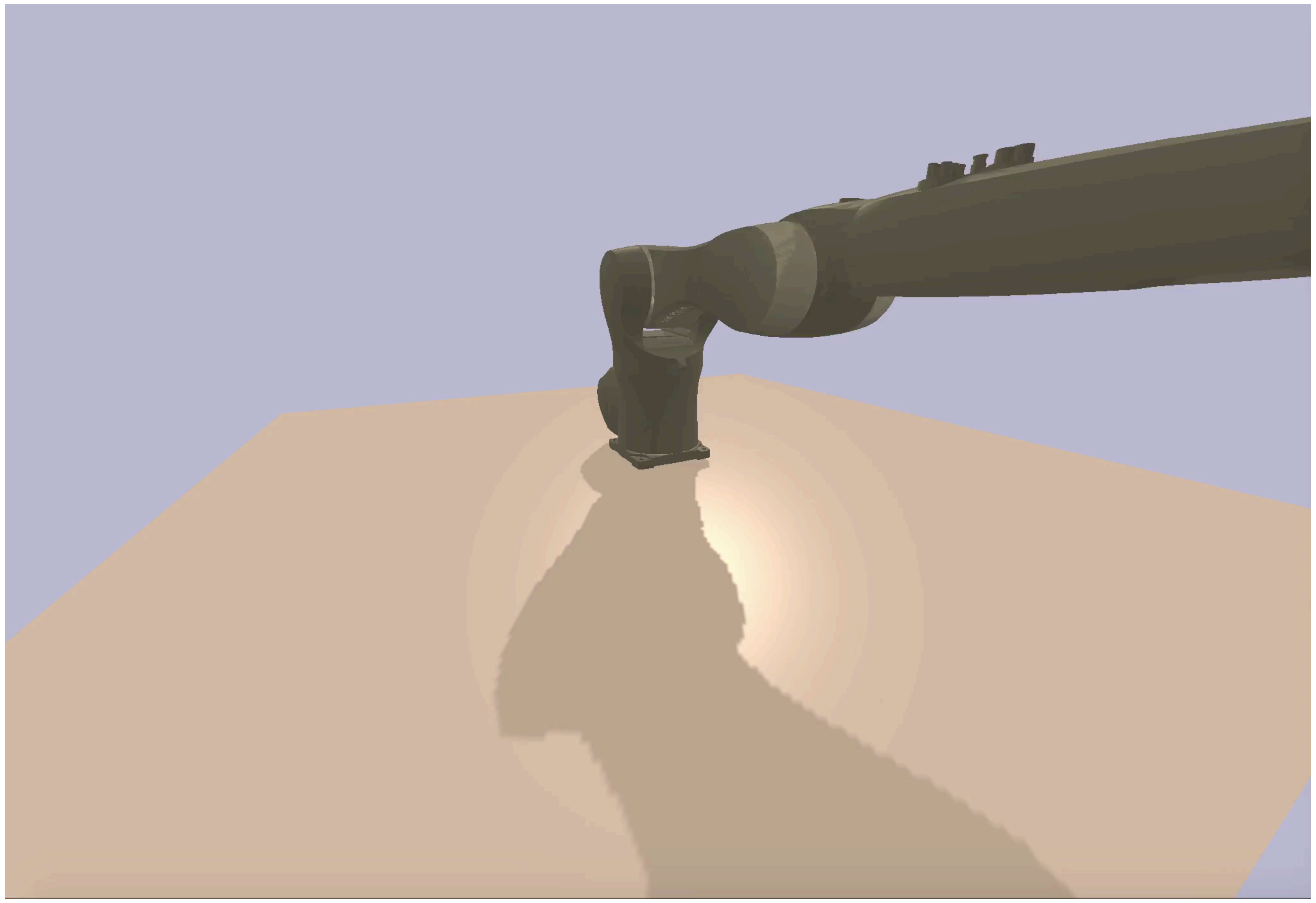
- Plan sequence of 306 3D printing extrusions
- Collision, kinematic, **stability** and **stiffness** constraints
- *Collaborators: Yijiang Huang and Caitlin Mueller*

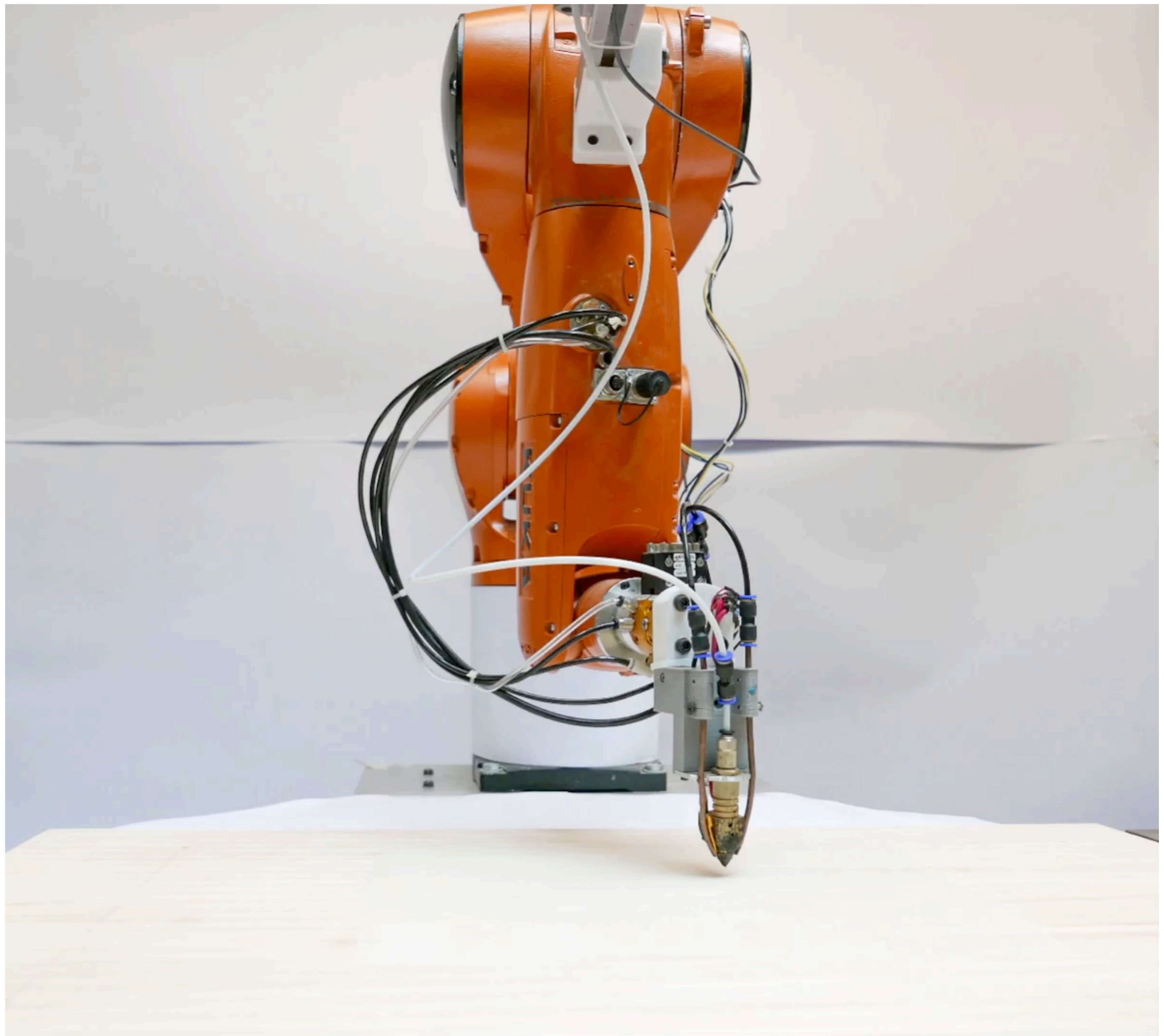




Applications: Automated Fabrication

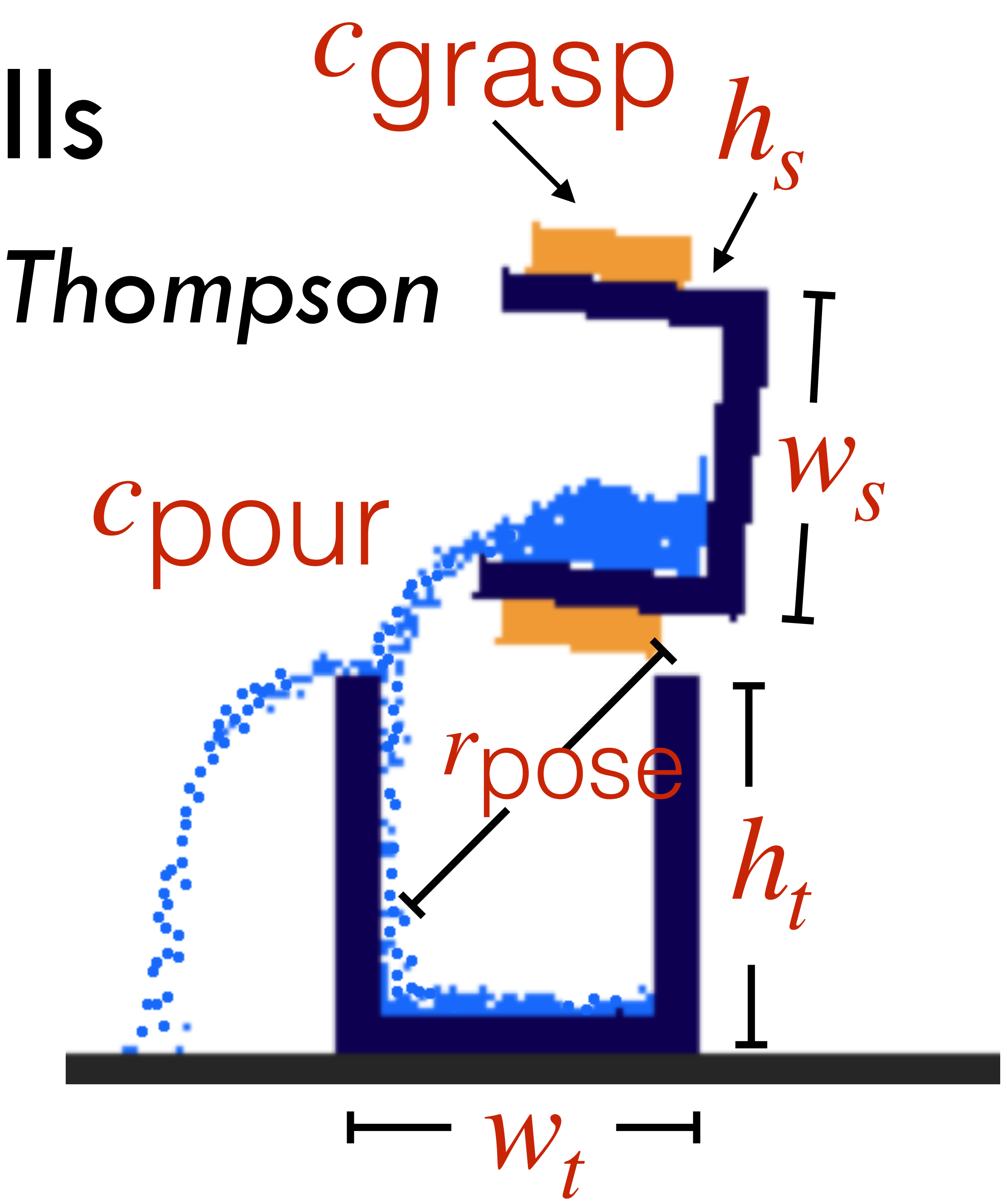
- Plan sequence of 306 3D printing extrusions
- Collision, kinematic, **stability** and **stiffness** constraints
- *Collaborators: Yijiang Huang and Caitlin Mueller*





Extension: Learning to Pour

- **Learn** good **samplers** for dynamic skills
- *Collaborators: Zi Wang, Alex LaGrassa, Skye Thompson*



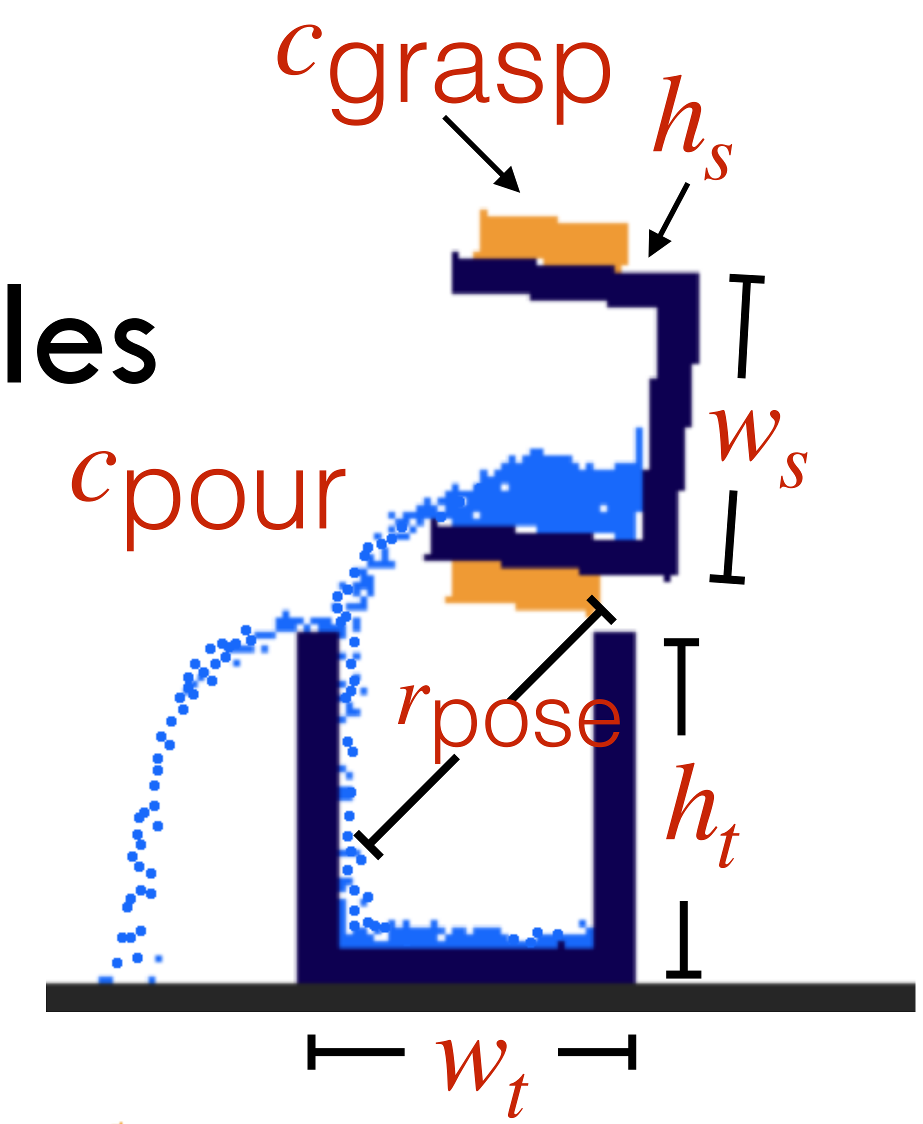
Precondition: (GoodPour ?arm ?bowl ?pose ?cup ?grasp ?conf ?traj)

Learner: $\text{score}(\underbrace{\overbrace{w_s, h_s, w_t, h_t}^{\text{given}}, \overbrace{c_{grasp}, c_{pour}, r_{pose}}^{\text{sample}}}_{\theta}) \geq 0$

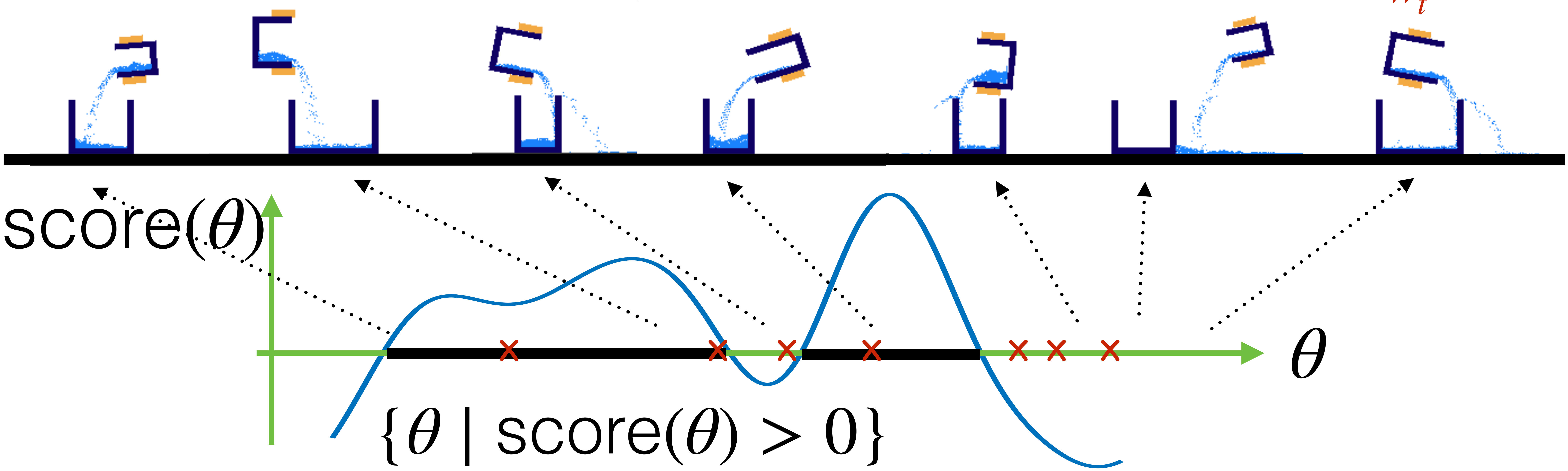
```
(:action pour
 :parameters (?arm ?bowl ?pose ?cup ?grasp ?conf ?traj)
 :precondition
    (and (GoodPour ?arm ?bowl ?pose ?cup ?grasp ?conf ?traj)
    (AtPose ?bowl ?pose) (AtGrasp ?arm ?cup ?grasp)
    (AtConf ?arm ?conf) (HasWater ?cup)
    (not (= ?bowl ?cup)) (not (UnsafeControl ?arm ?traj)))
 :effect (and (HasWater ?bowl) (not (HasWater ?cup))))
```

Sampling Good Pours

- **Learn classifier** for successful pours
- **Rejection sampling** for good pour samples

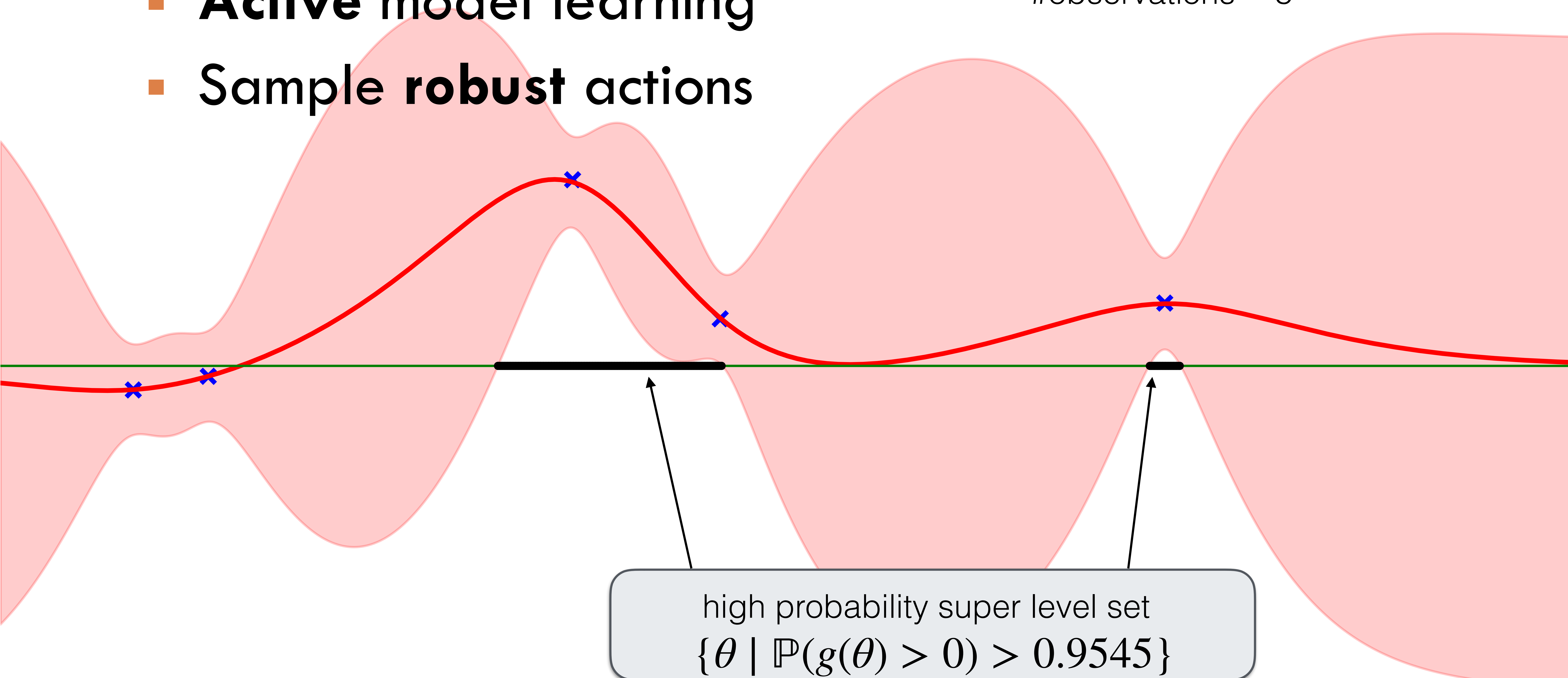


Learner: $\text{score}(\underbrace{\overbrace{w_s, h_s, w_t, h_t}^{\text{given}}, \overbrace{c_{grasp}, c_{pour}, r_{pose}}^{\text{sample}}}_{\theta}) > 0$

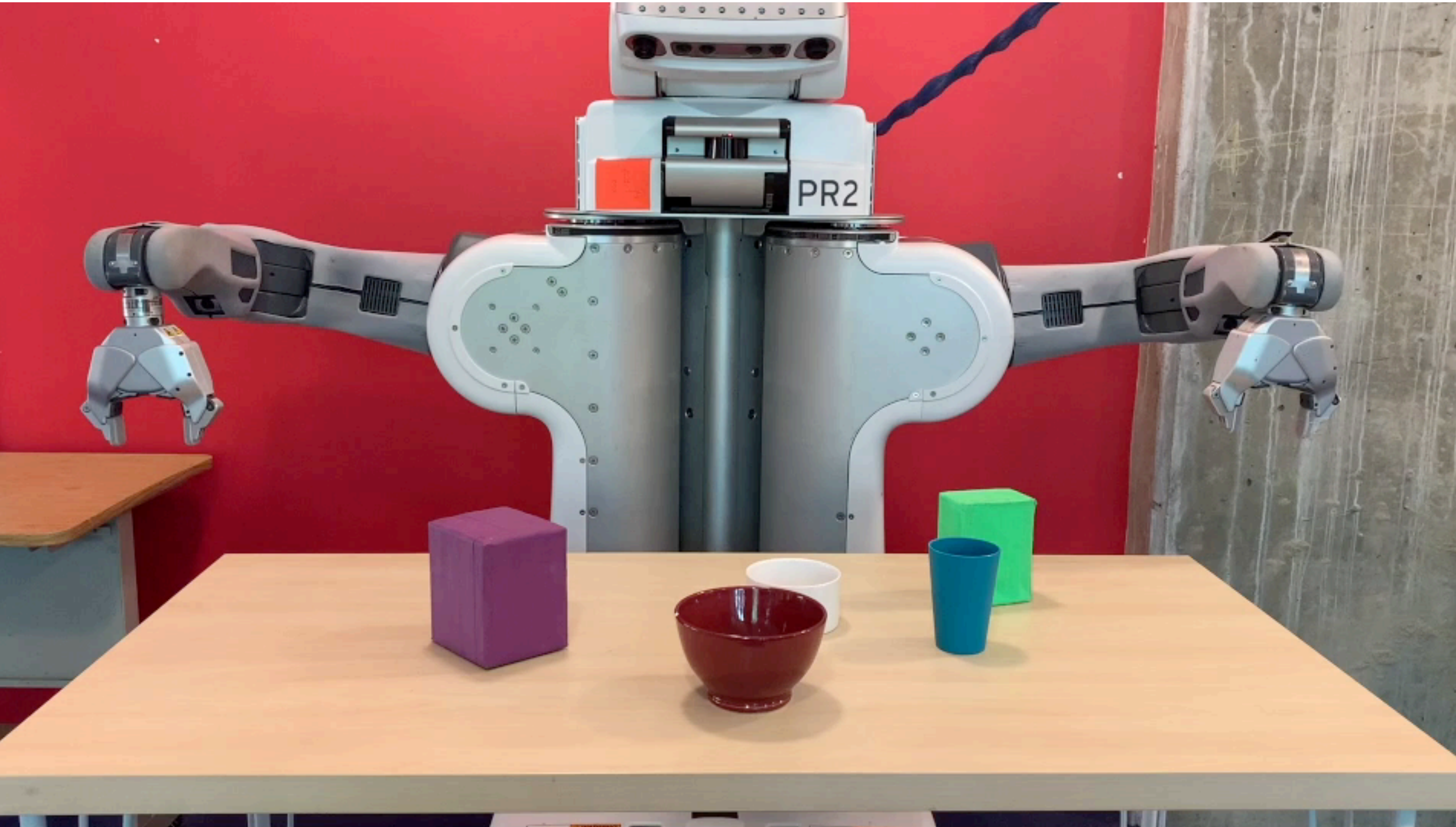


Gaussian Process (GP) Regression

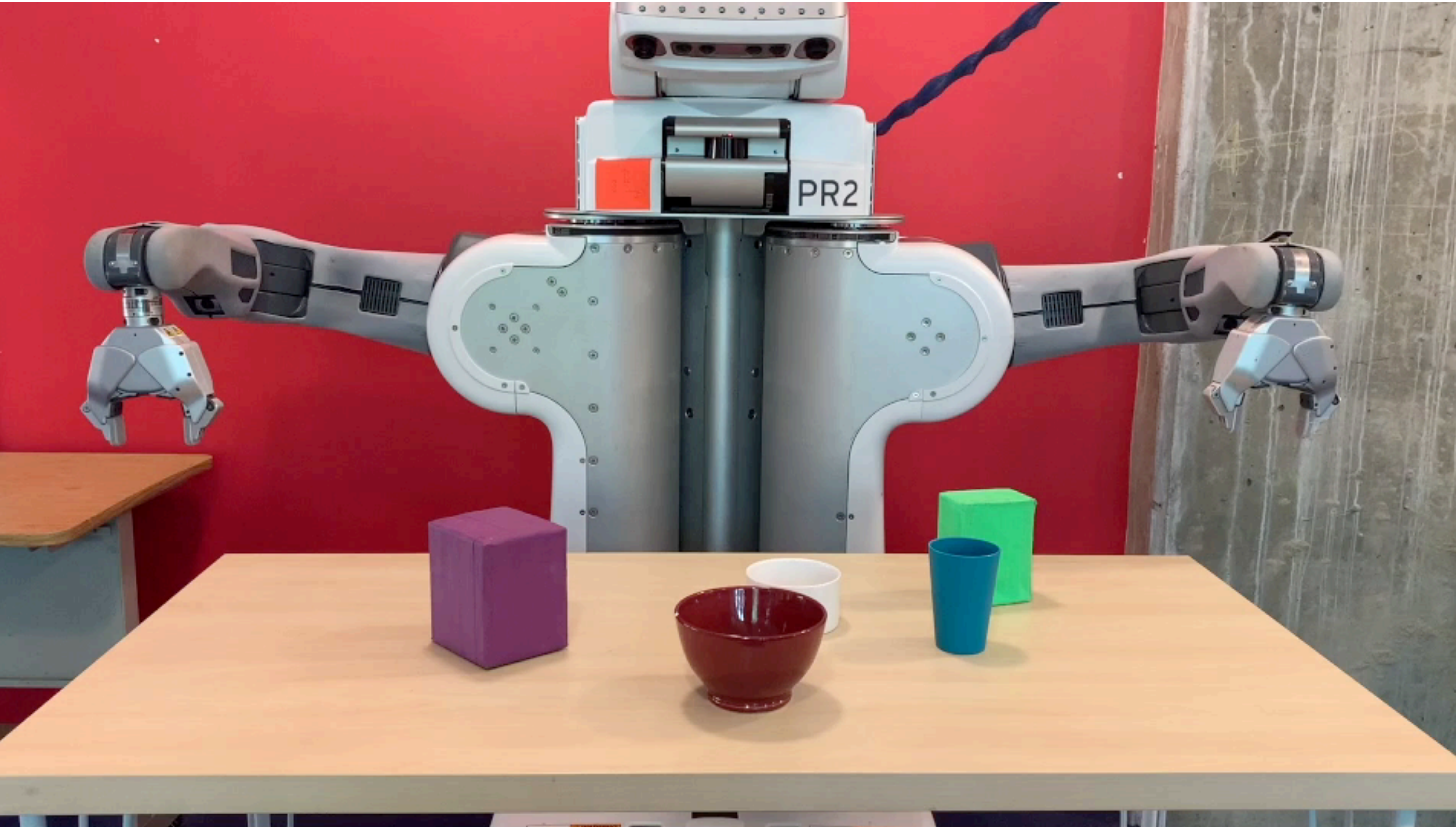
- Real robot data is expensive
- GPs encode **uncertainty**
  - **Active** model learning
  - Sample **robust** actions



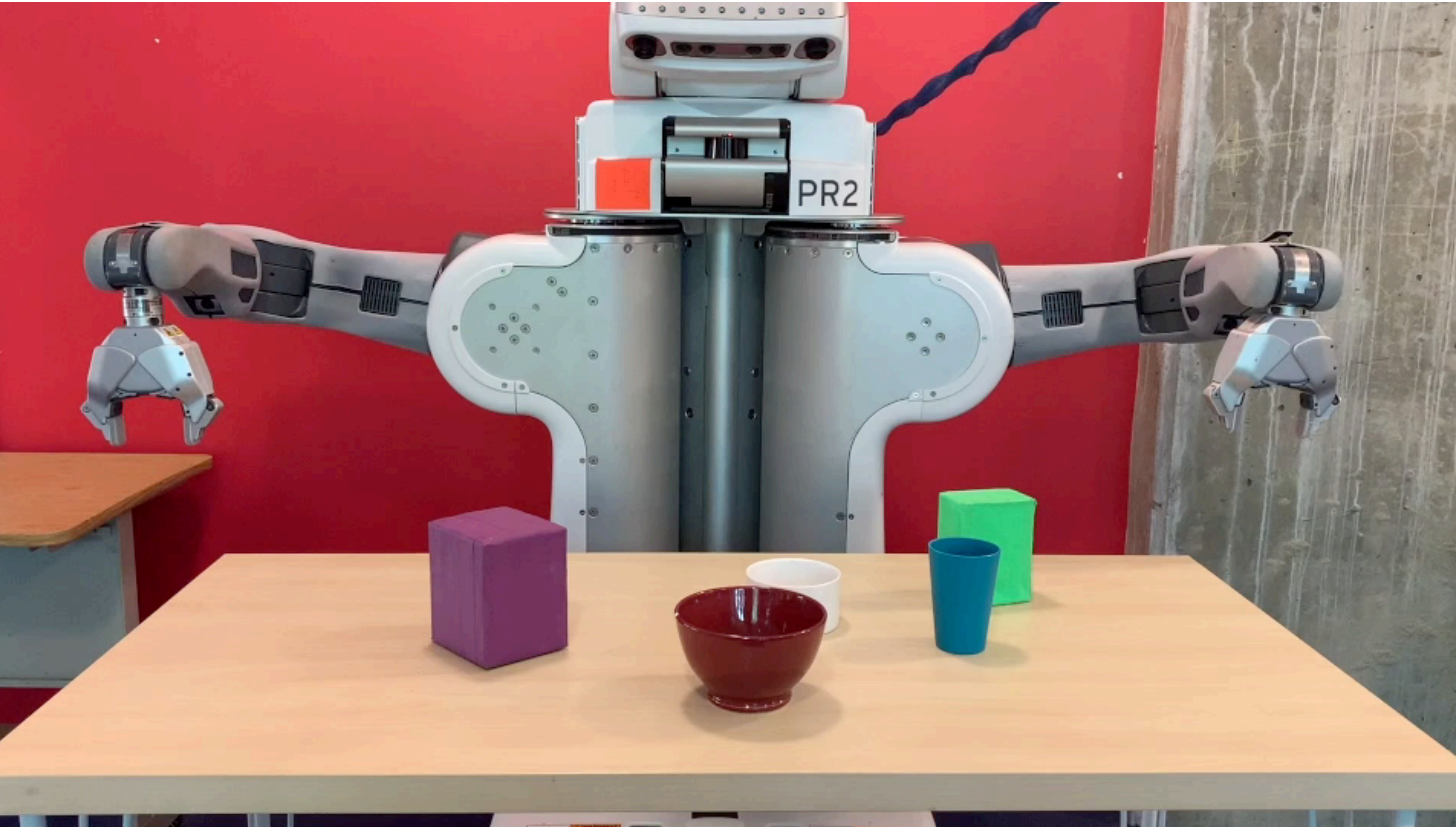
Planning with Learned Pours



Planning with Learned Pours

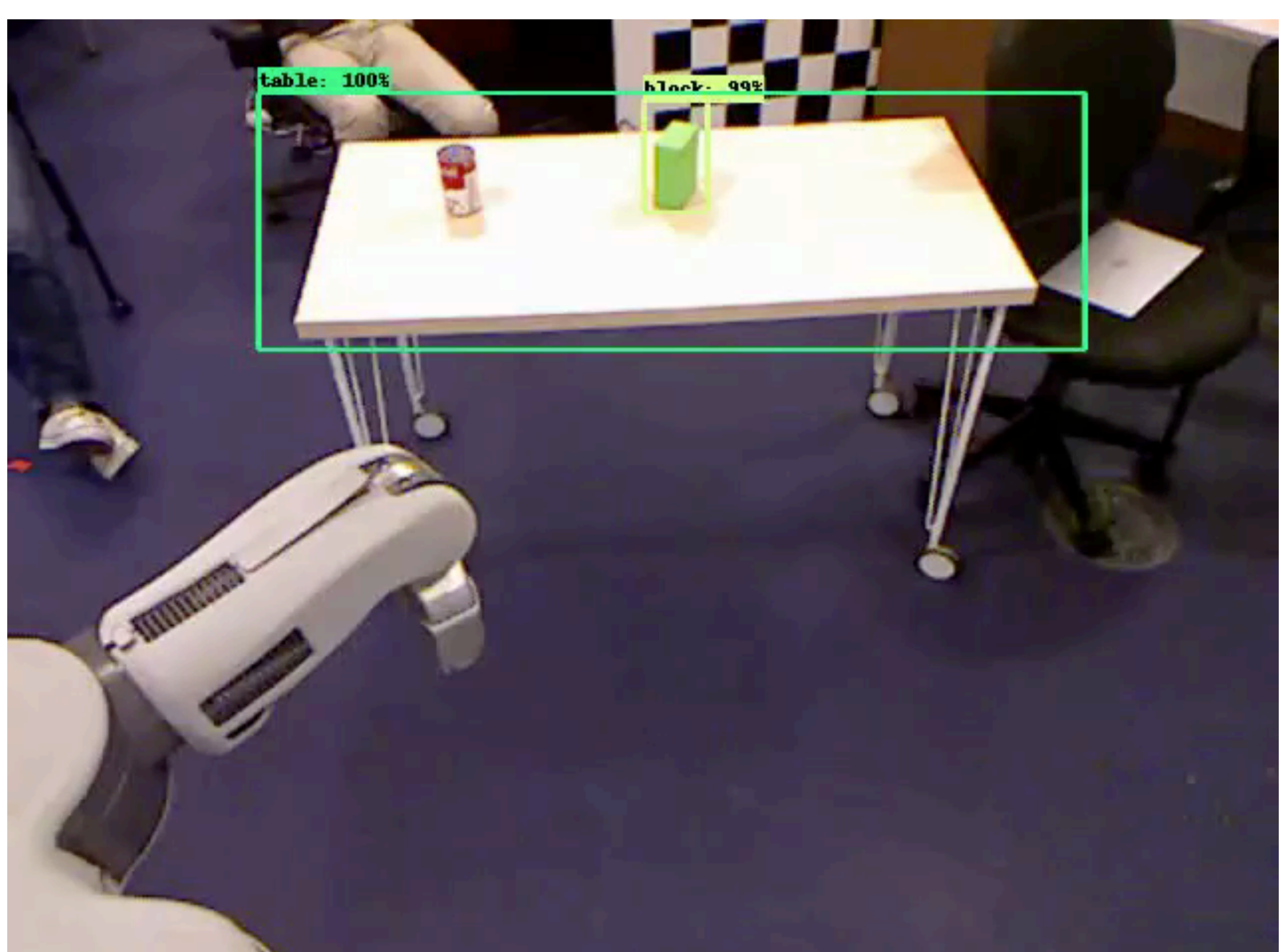


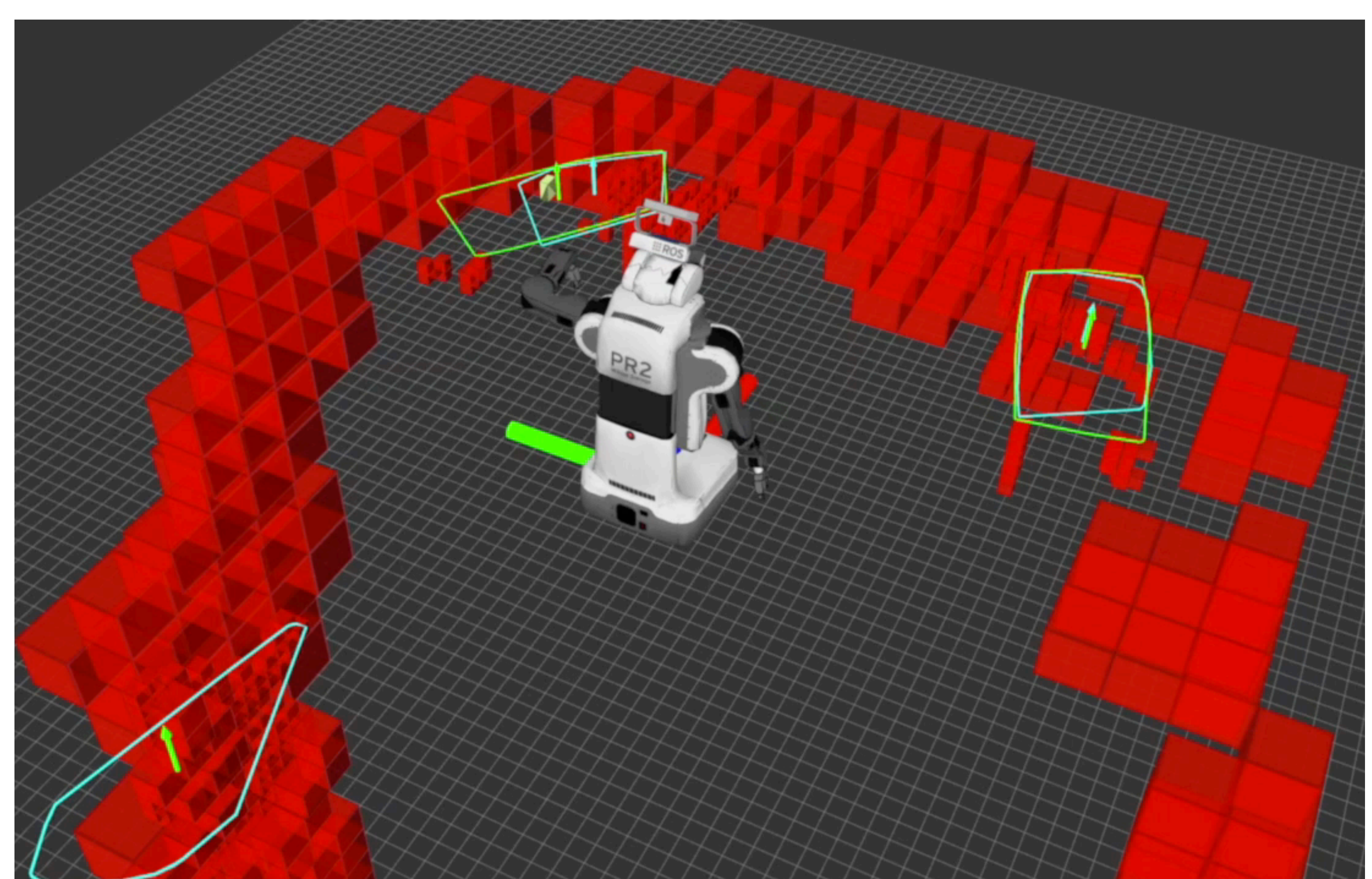
Planning with Learned Pours

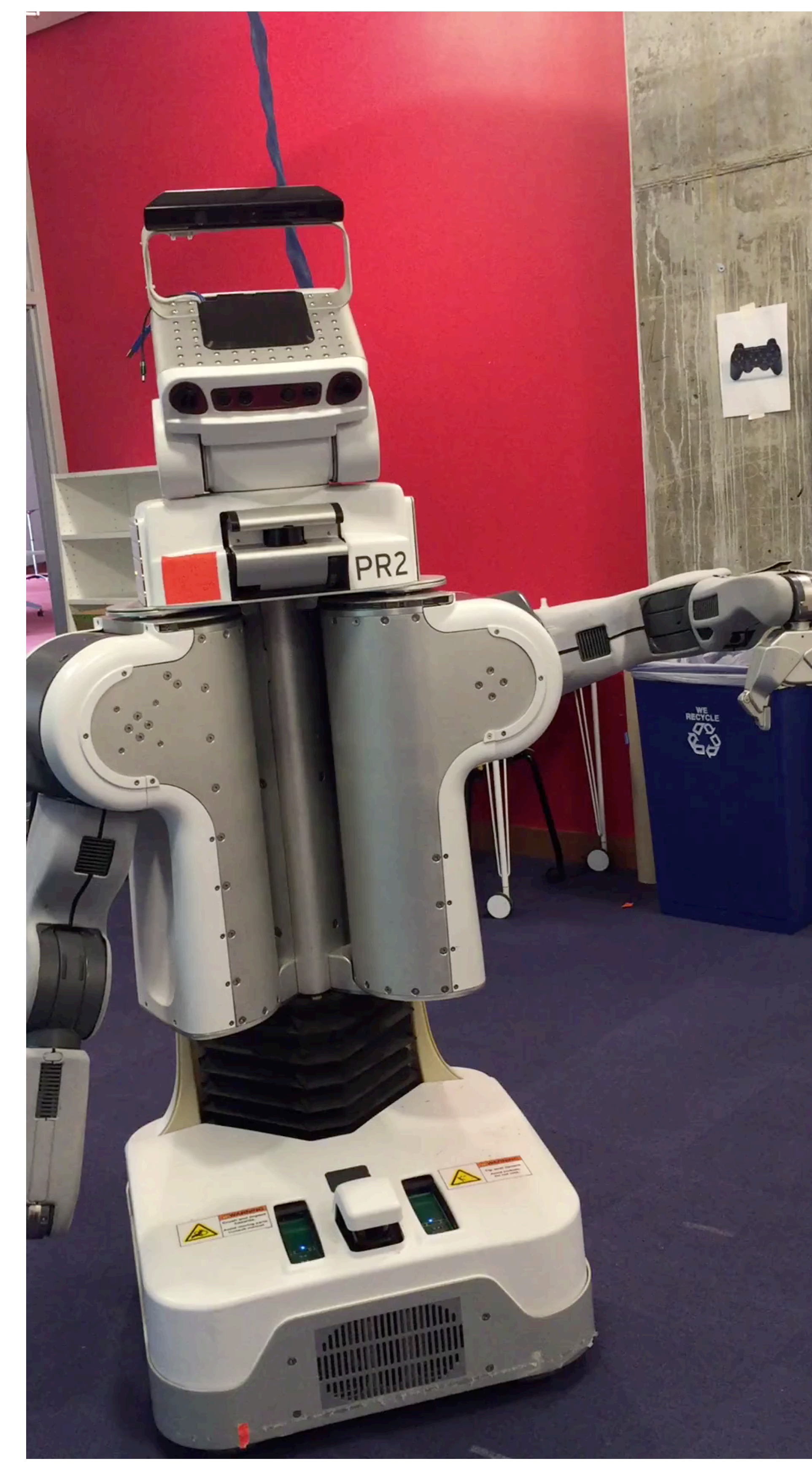


Planning & Execution with Uncertainty

- World is **stochastic** (MDP):
  - **Determinize** (select the effect of) action outcomes
  - Penalize **unlikely** and **costly** outcomes
  - **Replan** after execution
- World is **partially observable** (POMDP):
  - Plan on **distributions** over states

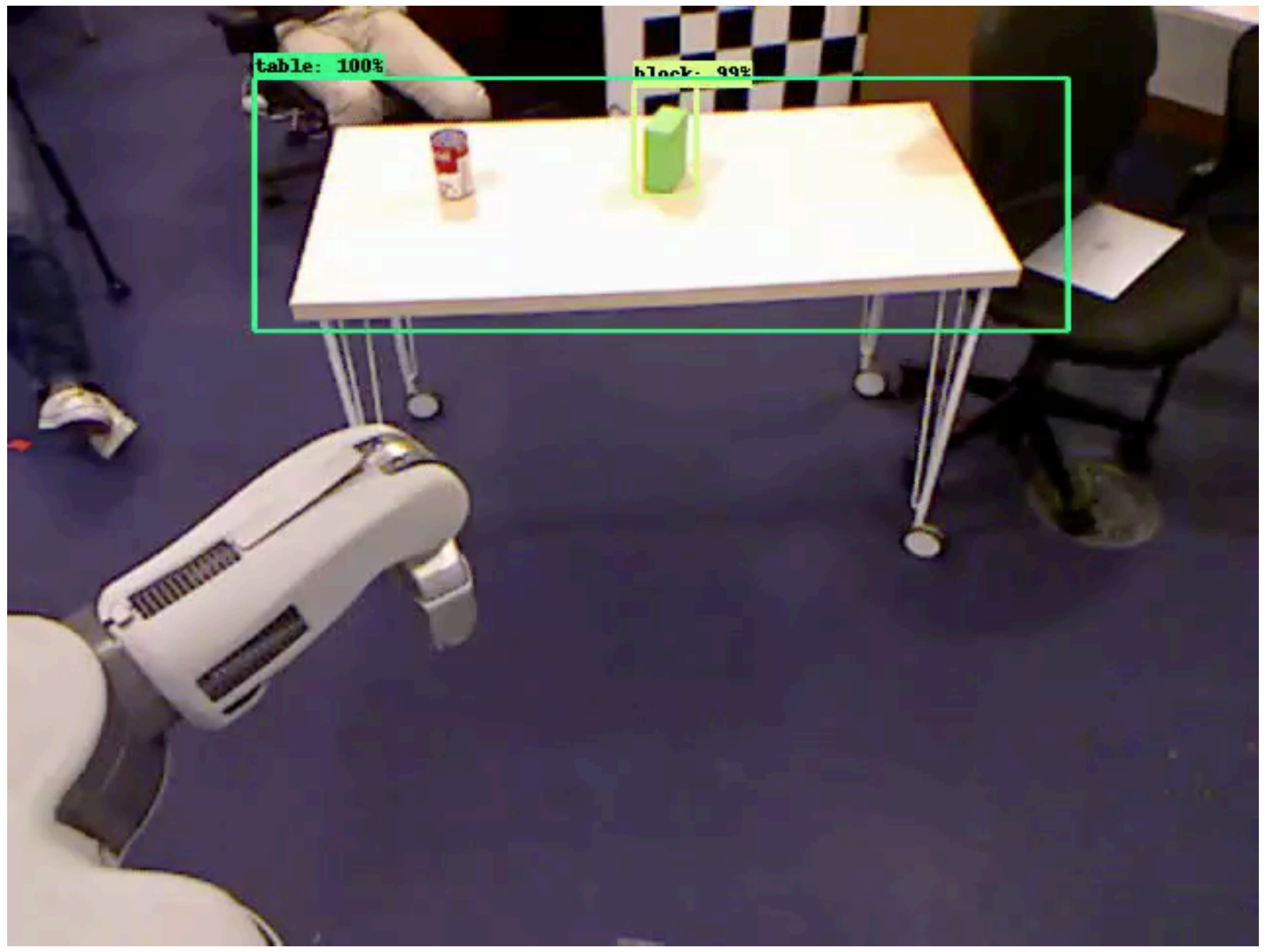


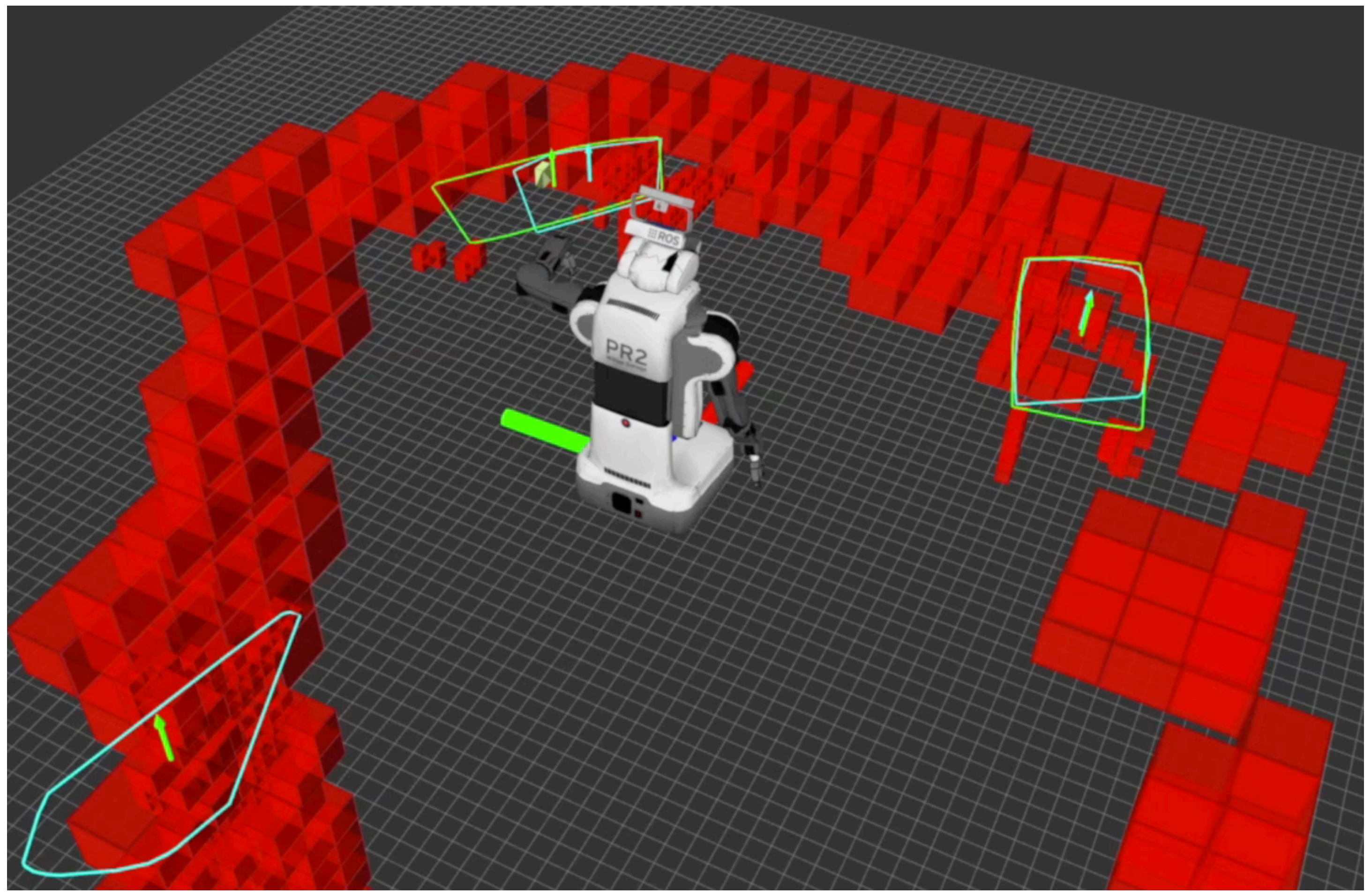


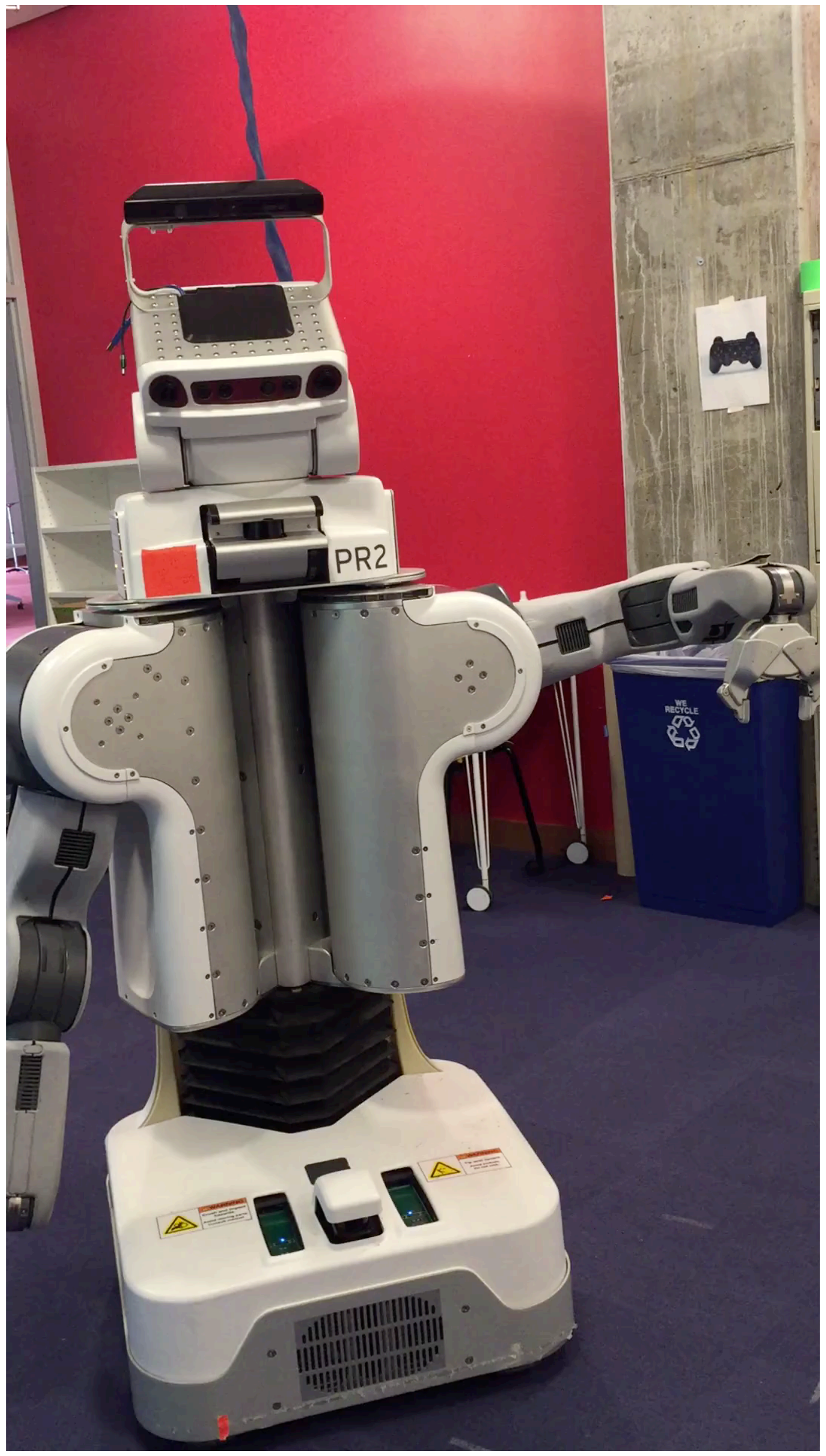


Planning & Execution with Uncertainty

- World is **stochastic** (MDP):
  - **Determinize** (select the effect of) action outcomes
  - Penalize **unlikely** and **costly** outcomes
  - **Replan** after execution
- World is **partially observable** (POMDP):
  - Plan on **distributions** over states



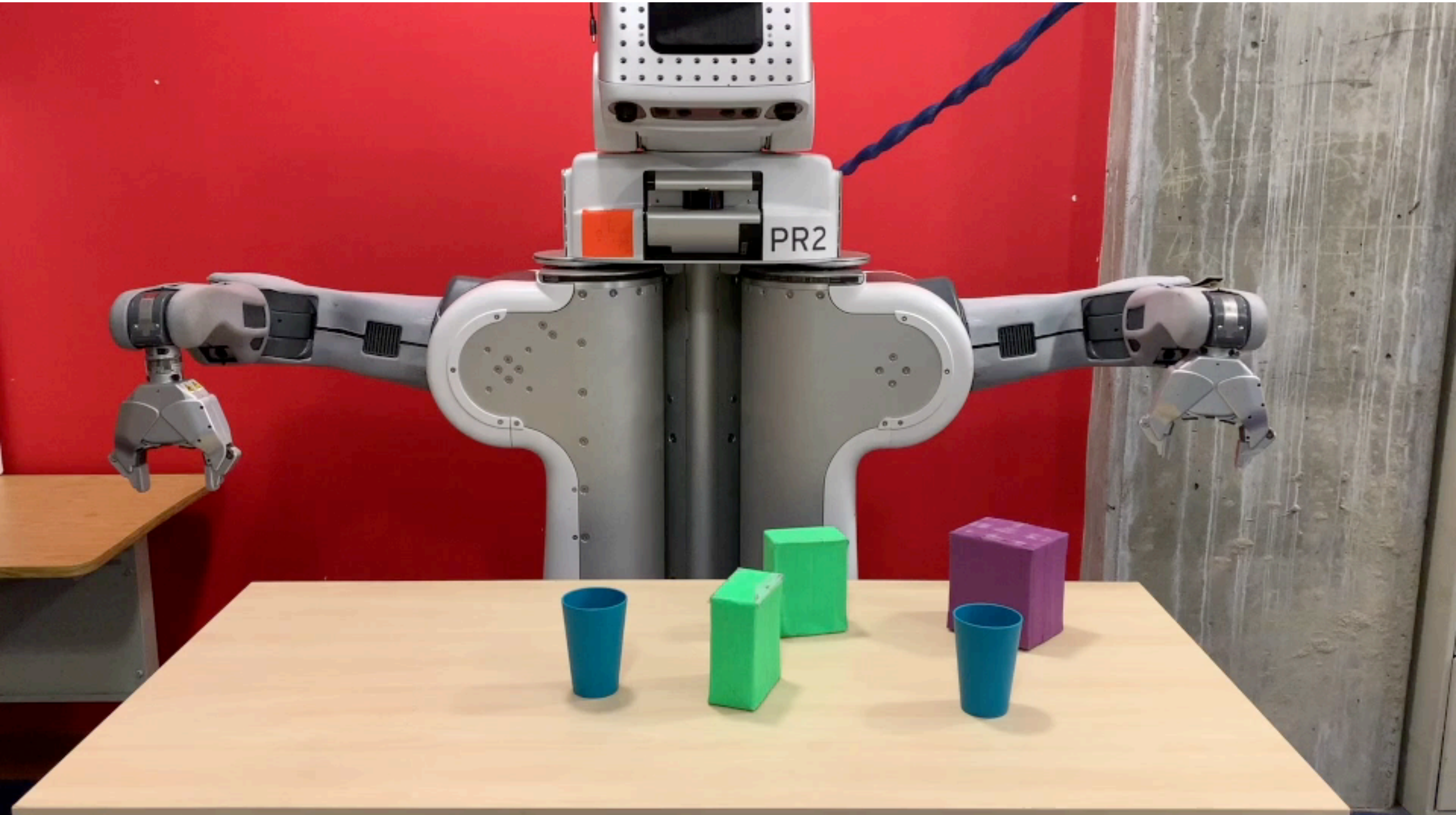




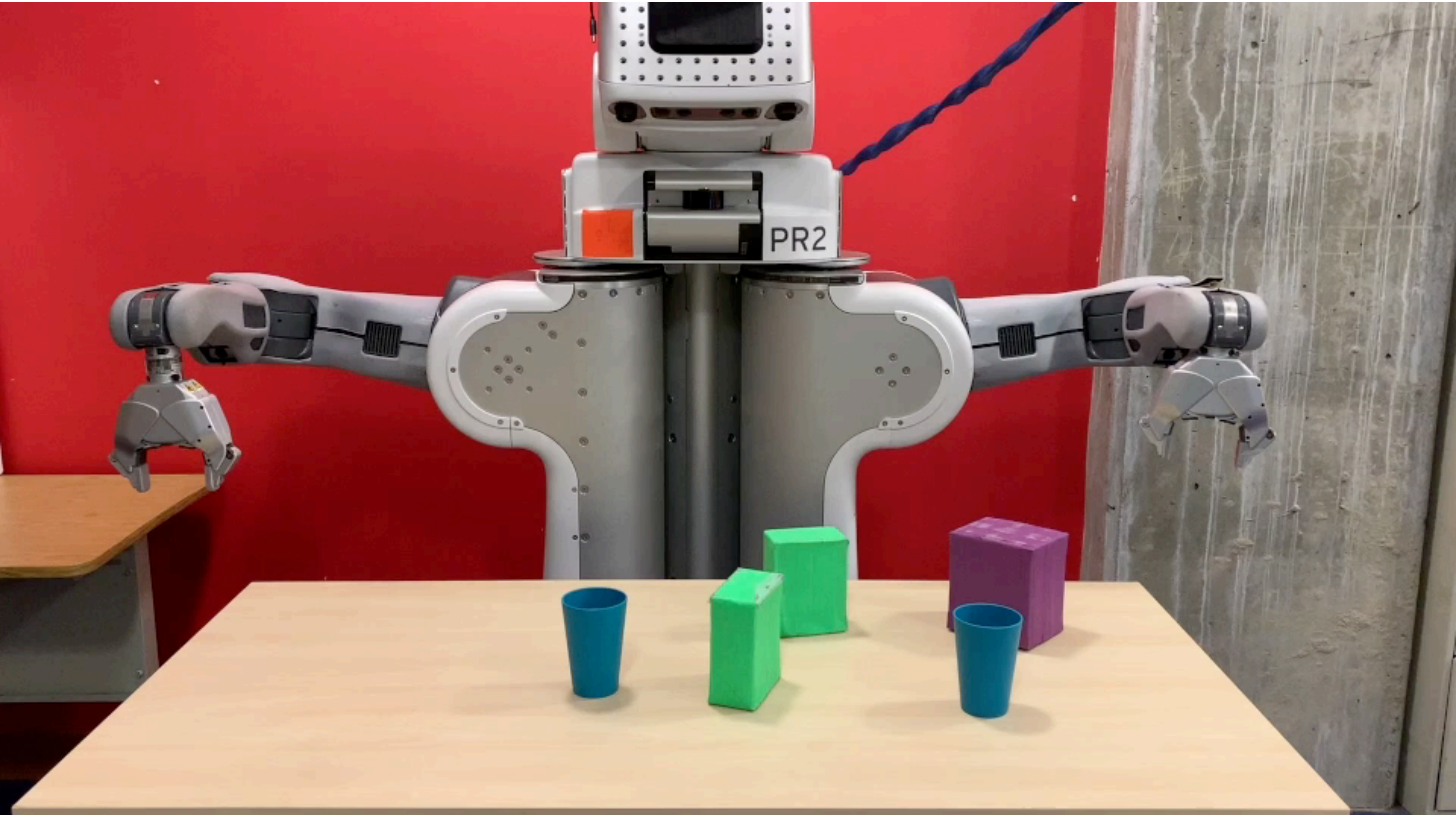
Takeaways

- **STRIPStream: general-purpose** planning language that supports **sampling procedures (streams)**
- **Domain-independent algorithms** that operate on **streams** as blackboxes
- **Focused algorithm** able to intelligently query only a small number of samplers
- Ongoing work involving **multi-agent** planning, **fabrication**, **learning** samplers, **cost-sensitive** planning, and **planning & execution**

Questions? (and Outtakes!)



Questions? (and Outtakes!)



Questions? (and Outtakes!)