PDDLStream: Integrating Symbolic Planners and Blackbox Samplers via Optimistic Adaptive Planning

Caelan R. Garrett, Tomás Lozano-Pérez,

and Leslie P. Kaelbling

ICAPS 2020

Contact: caelan@csail.mit.edu

Videos: https://tinyurl.com/pddlstream

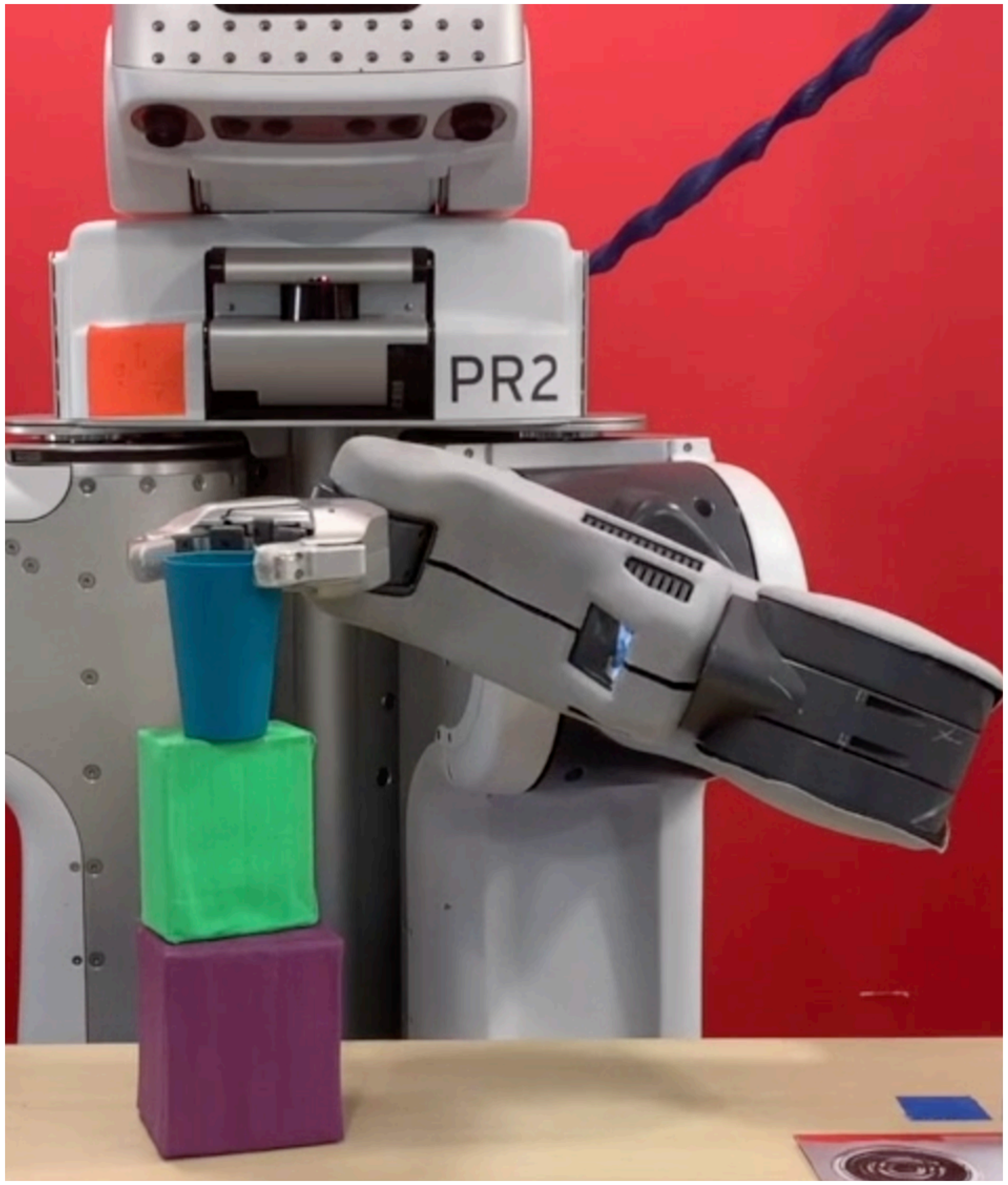
Code: https://github.com/caelan/pddlstream

CSAIL

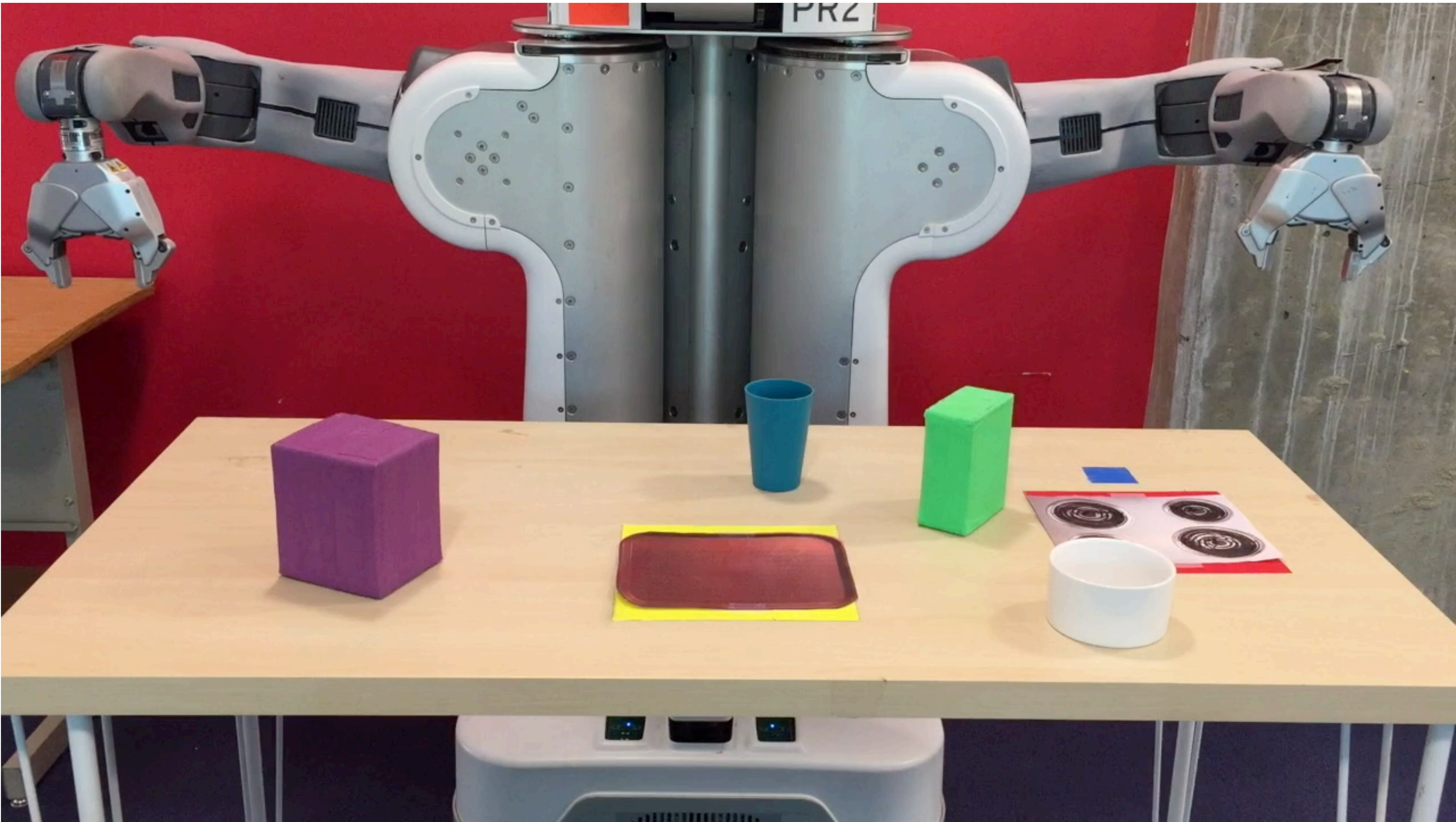
LEARNING & INTELLIGENT SYSTEMS

Task and Motion Planning (TAMP)

- Robot plans **high-level** actions & **low-level** controls
- Plan in a **high-dimensional** and **hybrid** space
- Continuous/discrete **variables:**
  - Robot configuration, object poses, is-on, is-in-hand, …
- **Actions:** move, pick, place, push, pour, detect, cook, …

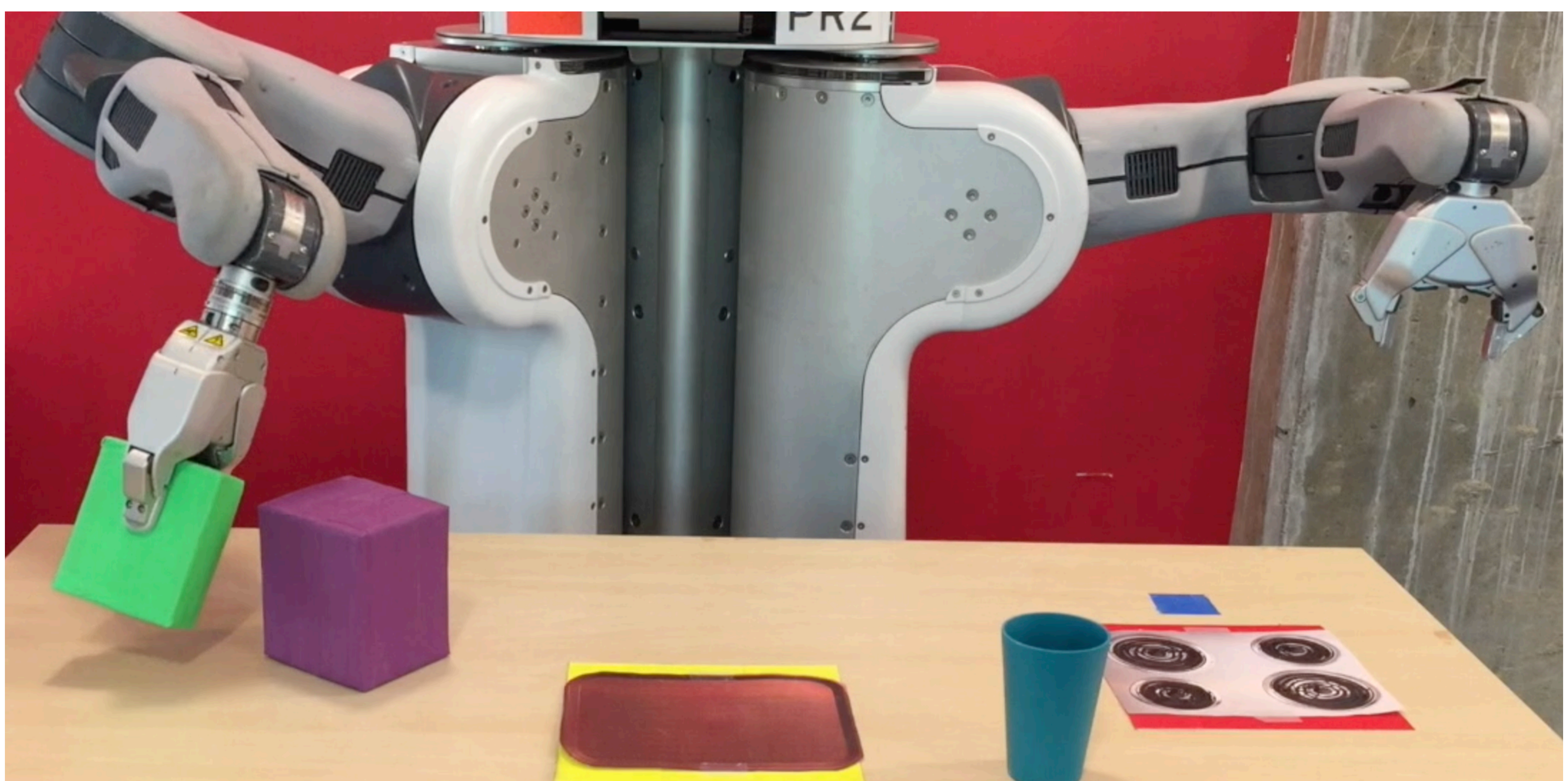


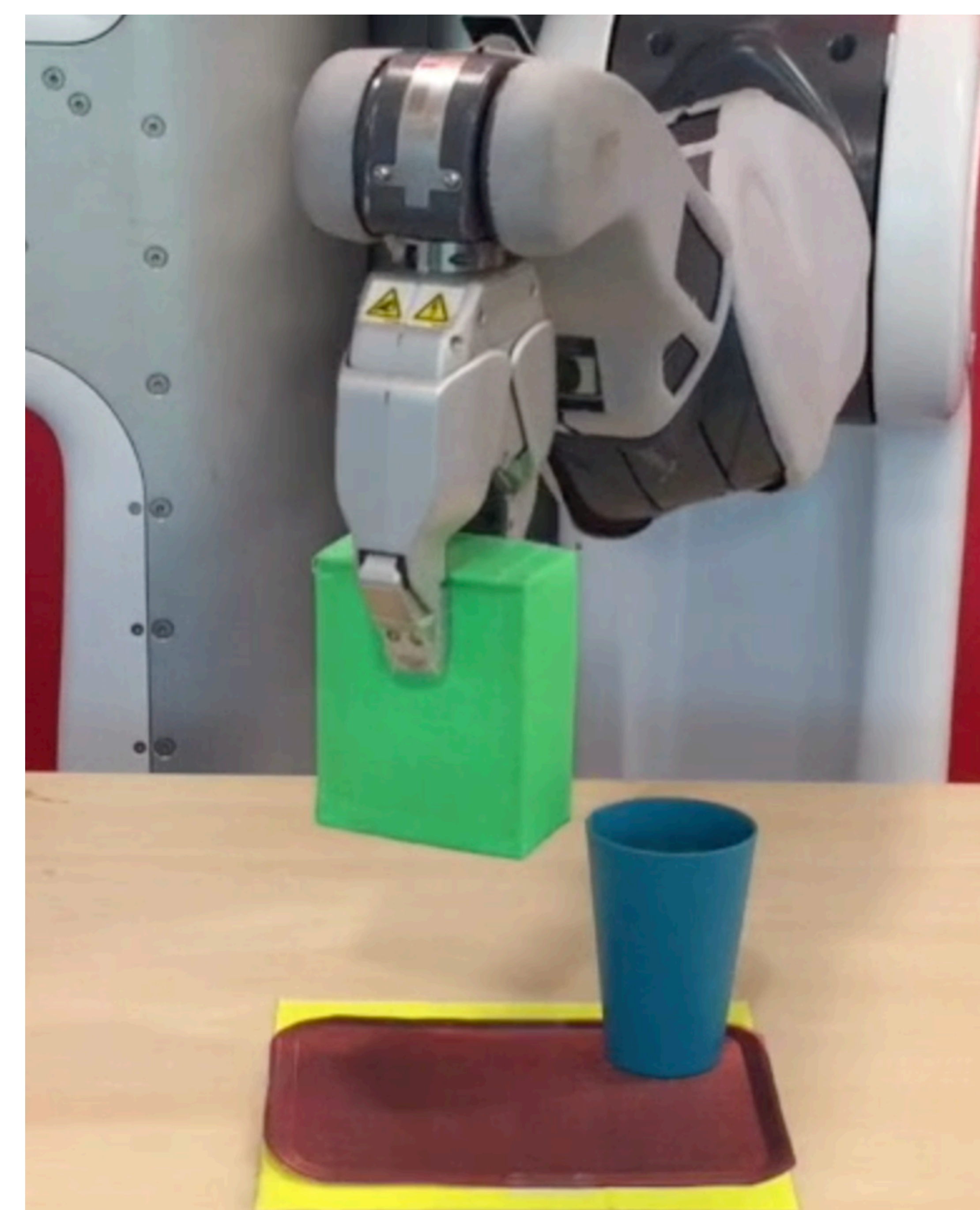
Manipulation: "Cooking"



Planner Produces Continuous Values

- Continuous action **parameter values** must satisfy dimensionality-reducing **constraints**
- Geometric constraints **limit high-level strategies**
  - Kinematics, reachability, joint limits, collisions, graspability, visibility, stability





Prior TAMP Work

- **Numeric Planning & Semantic Attachments** - [Fox, *Dornhege, Gregory, Cashmore]*
  - Assumes a finite action space
- **Task & Motion Interface -** *[Cambon, Kaelbling, Erdem, Srivastava, Garrett, Dantam]*
  - Application specific, no generic problem description
- **Multi-Modal Motion Planning -** [*Siméon, Hauser, Toussaint*]
  - Brute-force hybrid state-space search

- **No general-purpose, flexible framework** for modeling a variety of TAMP domains

Our Approach: PDDLStream

- Extends Planning Domain Definition Language (**PDDL**)
  - Modular & **domain-independent**

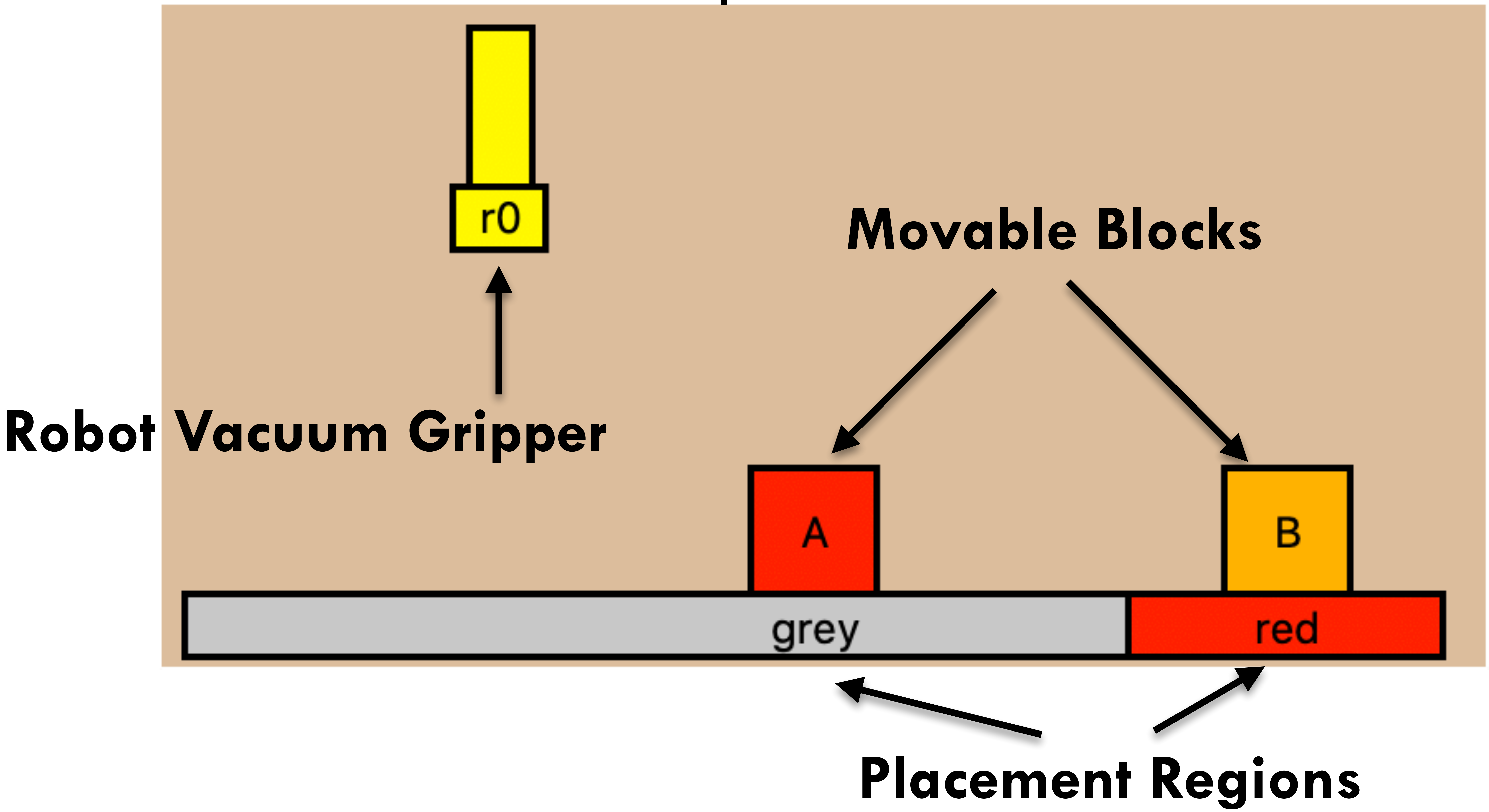
- Enables the specification of **sampling procedures**
  - Can encode domains with **infinitely-many** actions

- Admits **generic** algorithms that operate using the samplers as **blackbox inputs**
  - The **user** only needs to specify the samplers

PDDLStream Language

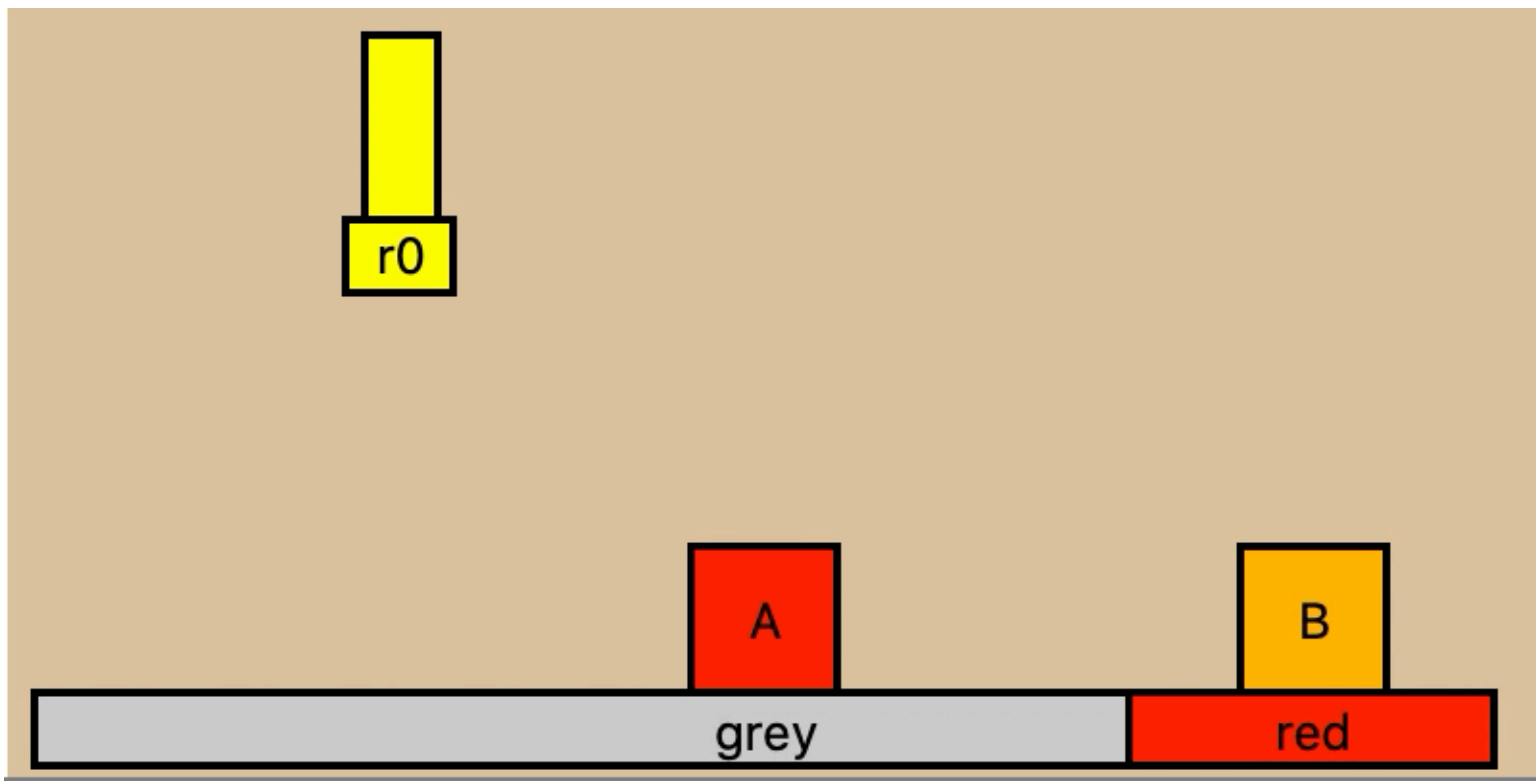
2D Pick-and-Place Example

- **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

- Discrete form of one (of **infinitely many**) 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**:

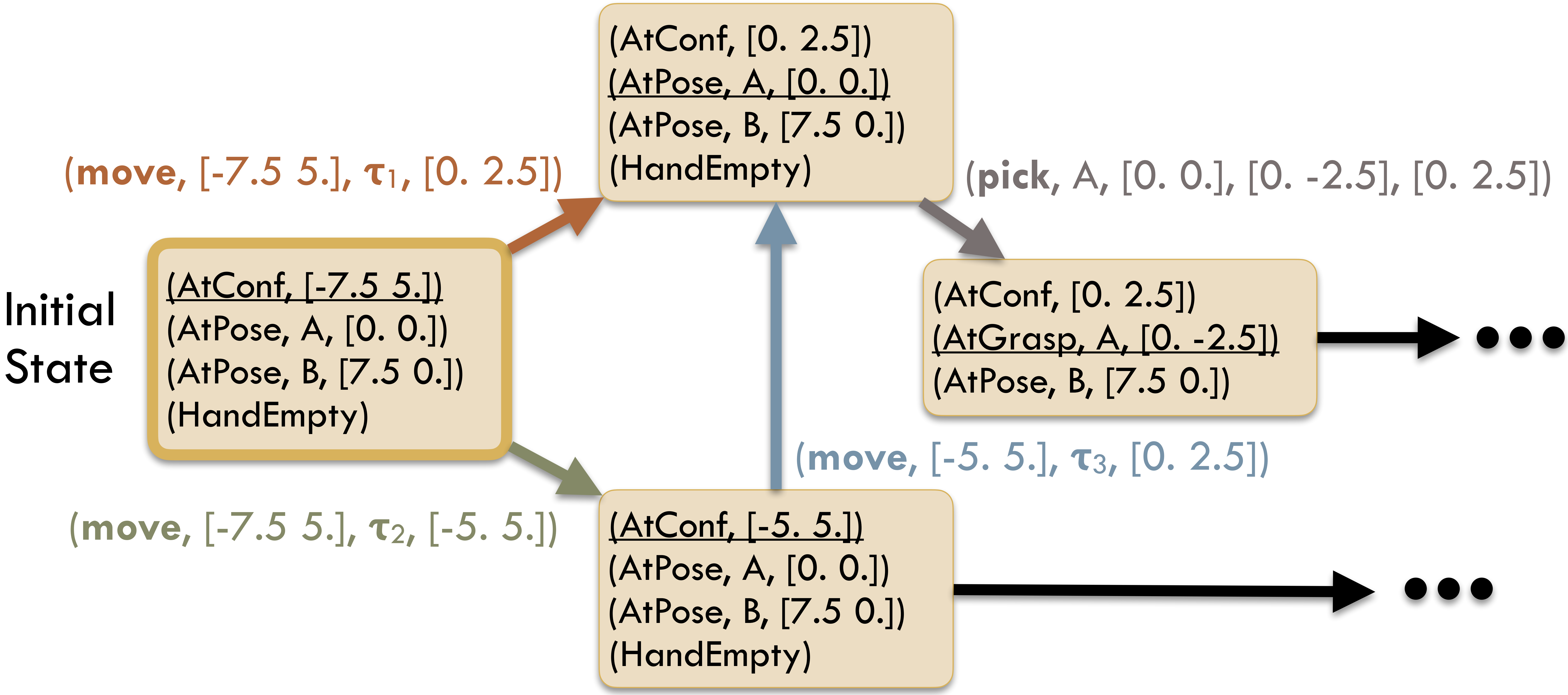
```
(Motion ?q1 ?t ?q2), (Kin ?b ?p ?g ?q)
```

```
(: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))))
```

Search 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]**)

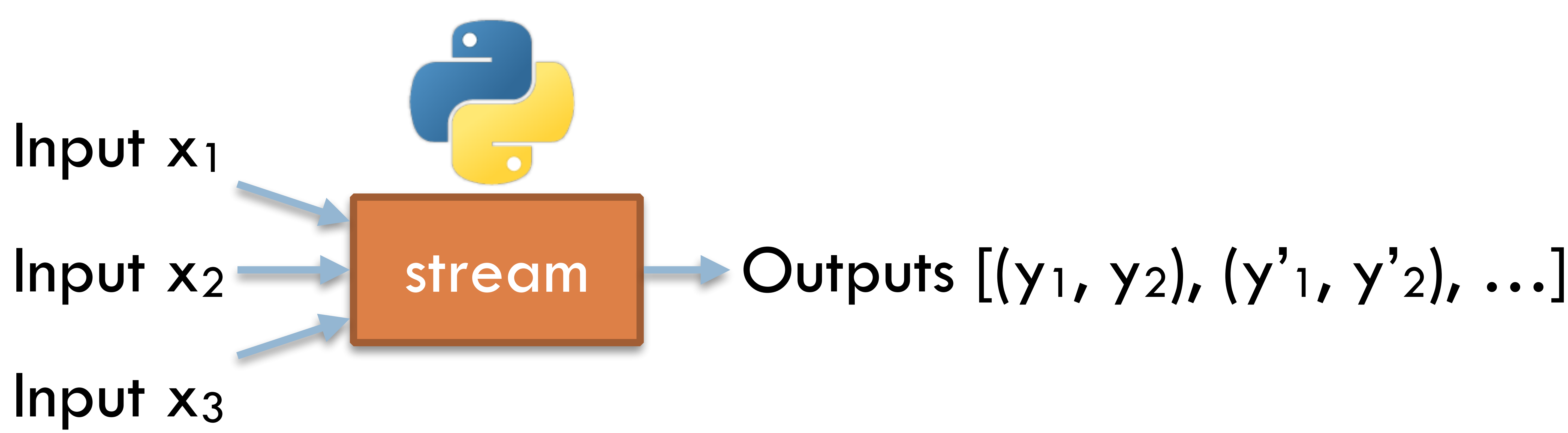
- **Planner needs to find:**
  - 1 pose within a region: `(Contain` **A** `?p` **red**`)`
  - 1 collision-free pose: `(CFree` **A** `?p ?` **B** `?p2)`
  - 4 grasping configurations: `(Kin ?b ?p ?g ?q)`
  - 4 robot trajectories: `(Motion ?q1 ?t ?q2)`

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

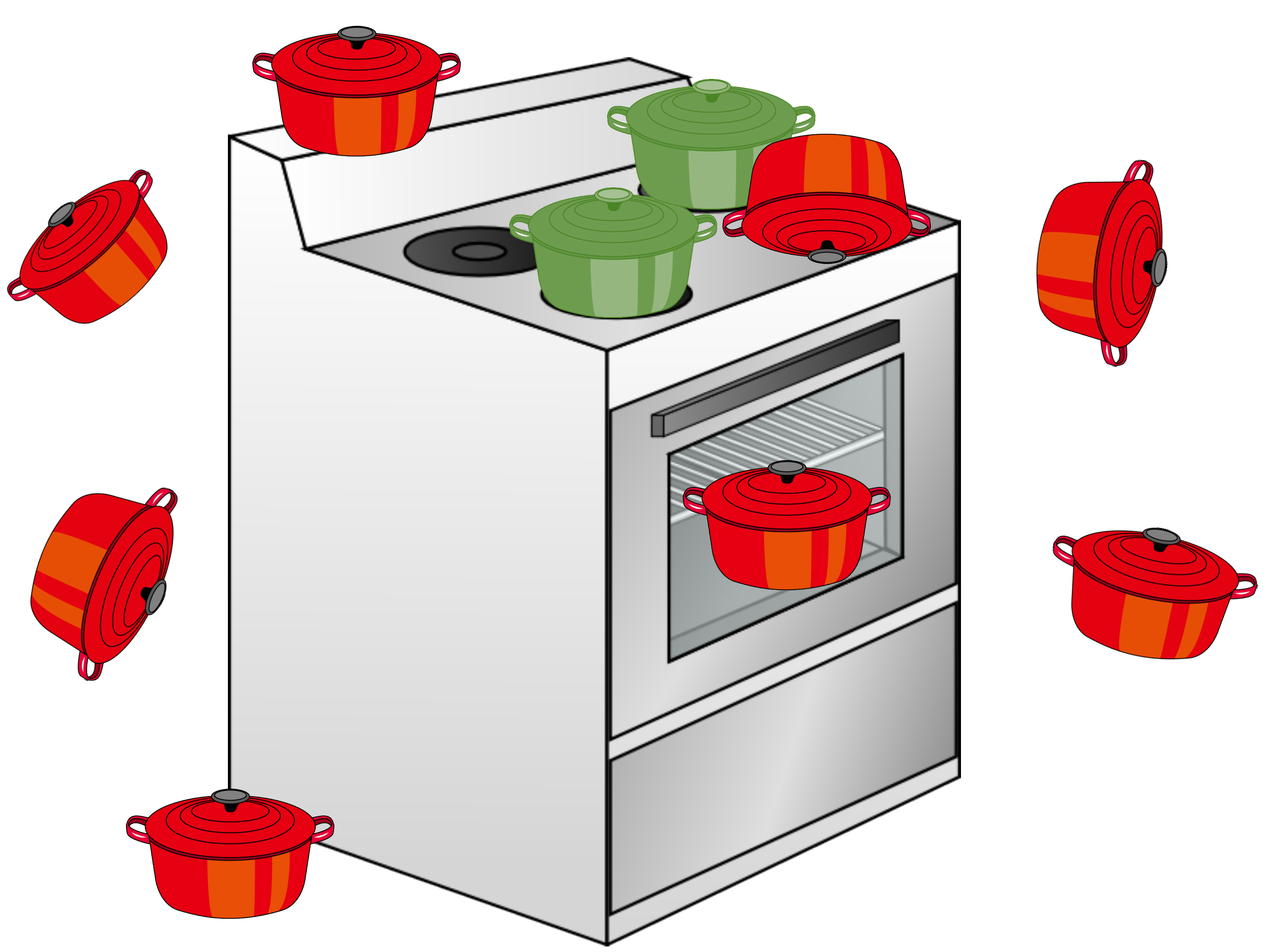
- Objects alone aren't helpful: **what do they represent?**
  - Communicate semantics using **predicates**!

- 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,)
```

Block b → sample-region

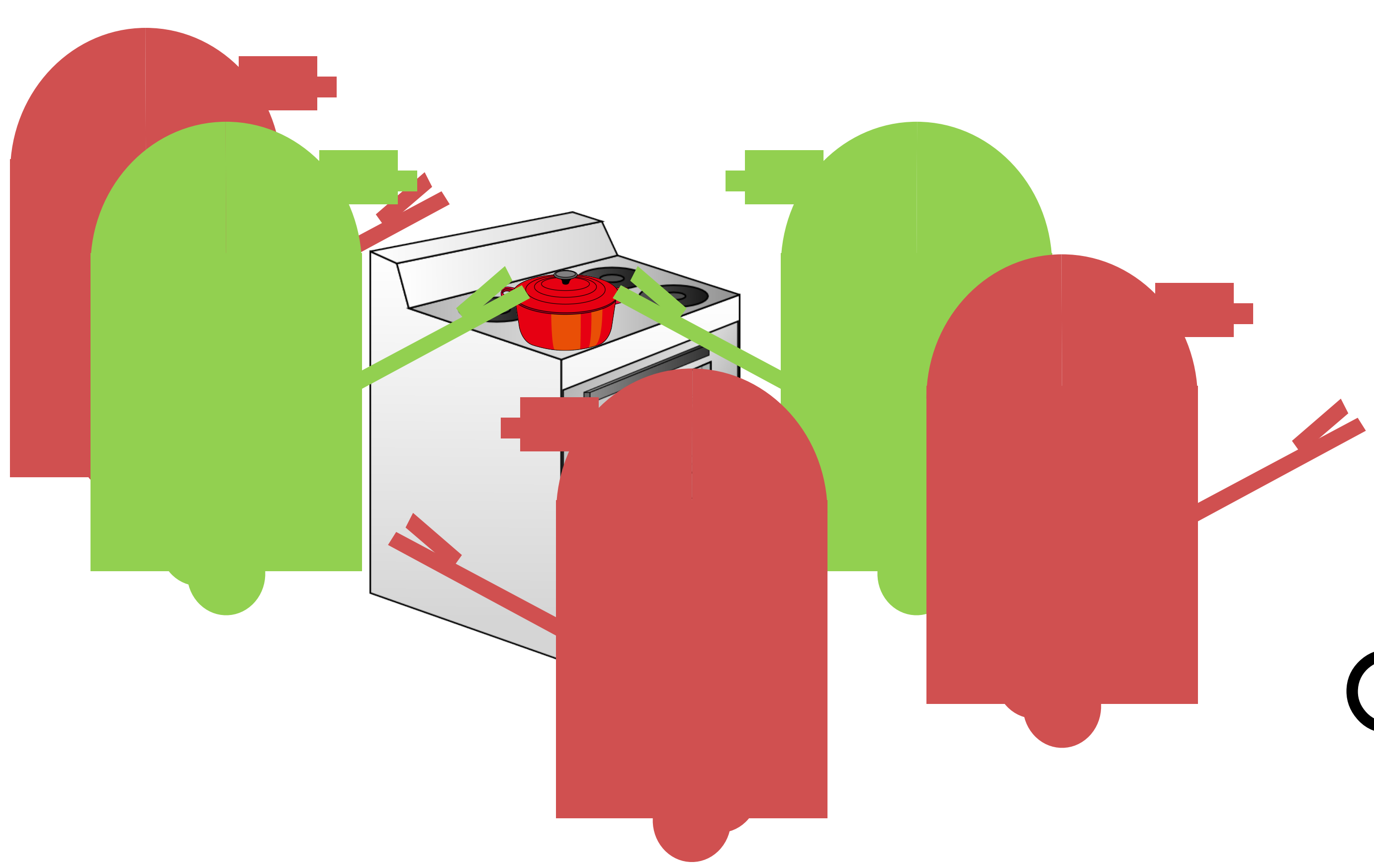
Region r → sample-region

sample-region → Pose [(p), (p’), (p”), …]

Sampling IK Solutions

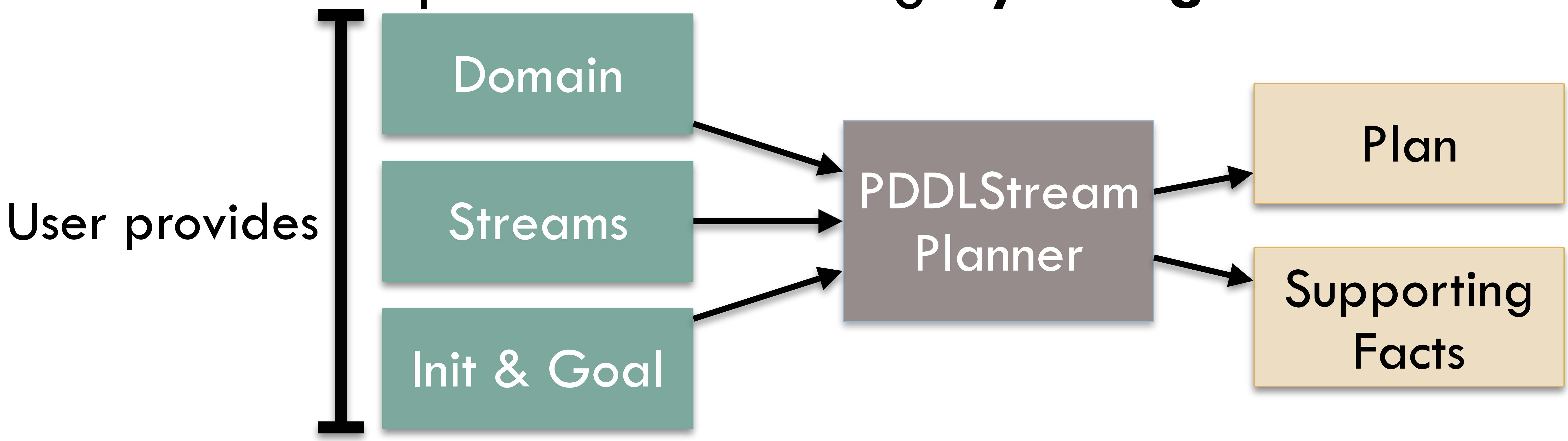
- **Inverse kinematics** (IK) to produce robot grasping configuration
- Trivial in 2D, non-trial in general (e.g. 7 DOF arm)

```
(: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)))
```



PDDLStream = PDDL + 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**



PDDLStream Algorithms

PDDLStream Algorithms

- **PDDLStream planners decide** which streams to use
- Our 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 any **off-the-shelf classical planner** (e.g. FastDownward)

Optimistic Stream Outputs

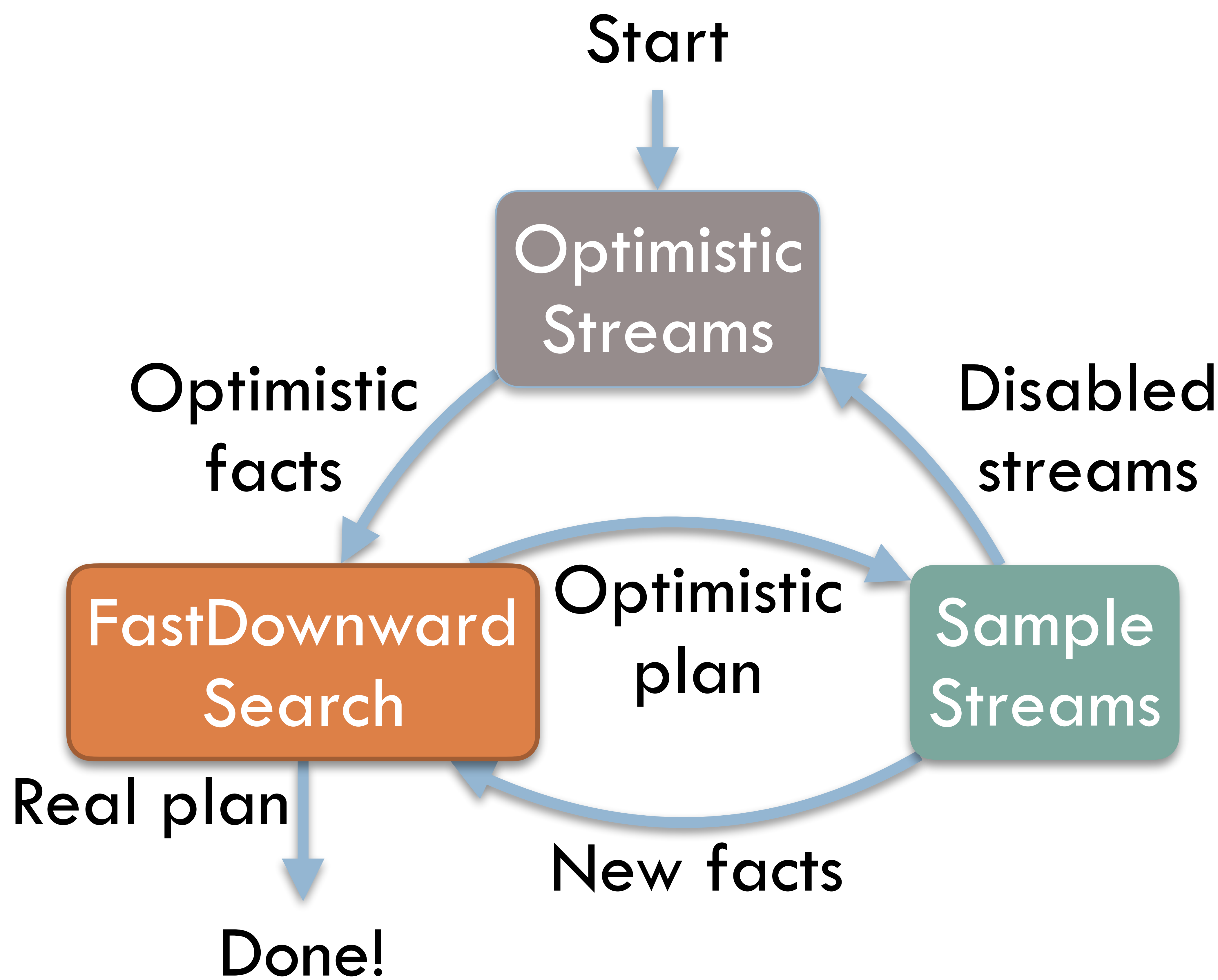
- Many TAMP streams are exceptionally **expensive**
  - Inverse kinematics, motion planning, collision checking
- **Only** query streams that are **identified** as useful
  - Plan with **optimistic hypothetical** outputs
- Inductively create **unique first-class** placeholder object for each stream instance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

Optimistic evaluations:

1. **s-region**:(block-A, red-region)->(**#p0**)
2. **s-ik**:(block-A, [0. 0.], [0. -2.5])->(**#q0**),
3. **s-ik**:(block-A, **#p0**, [0. -2.5]) ->(**#q2**)

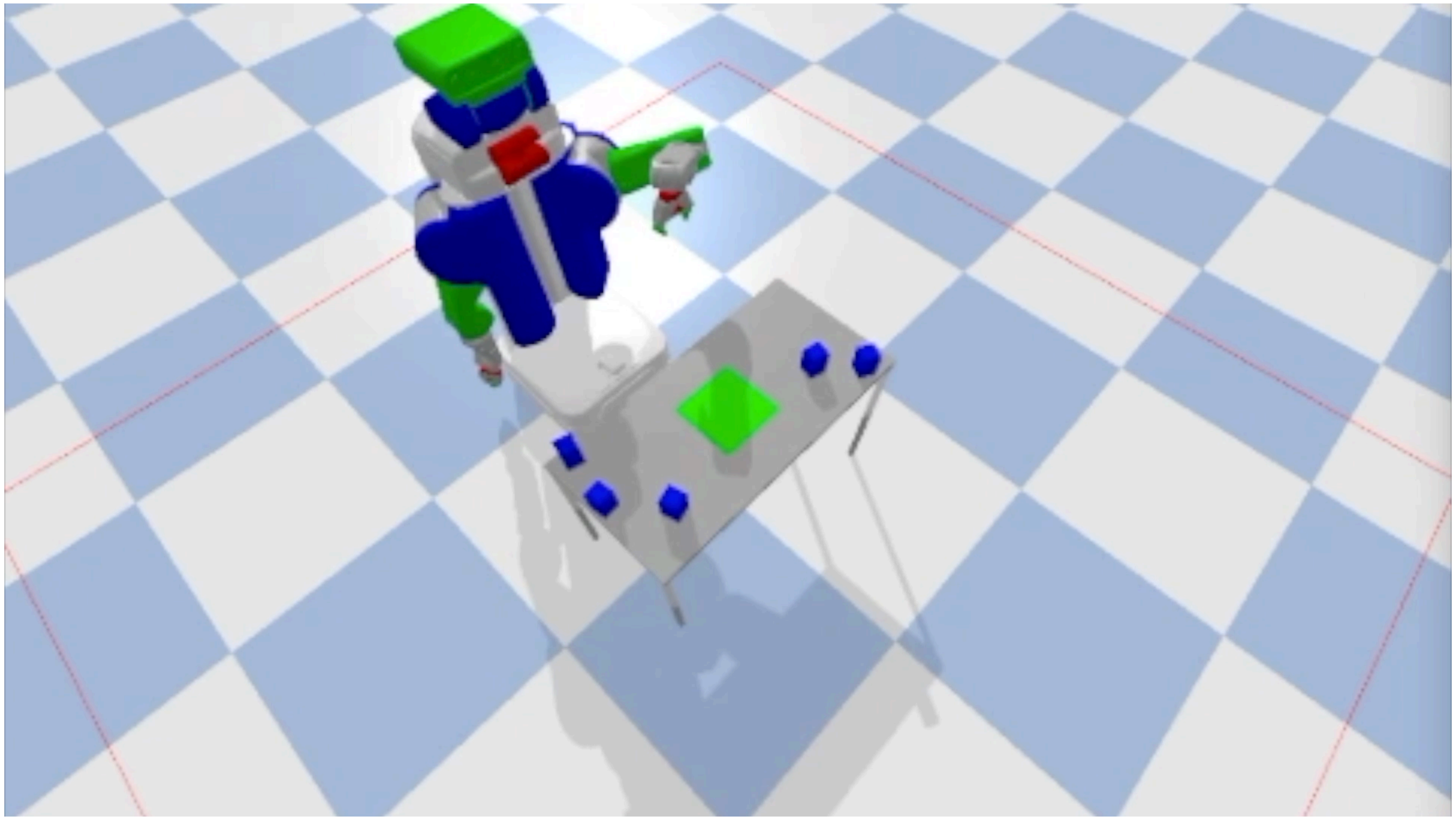
Binding (& ≈Focused) Algorithm

- **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



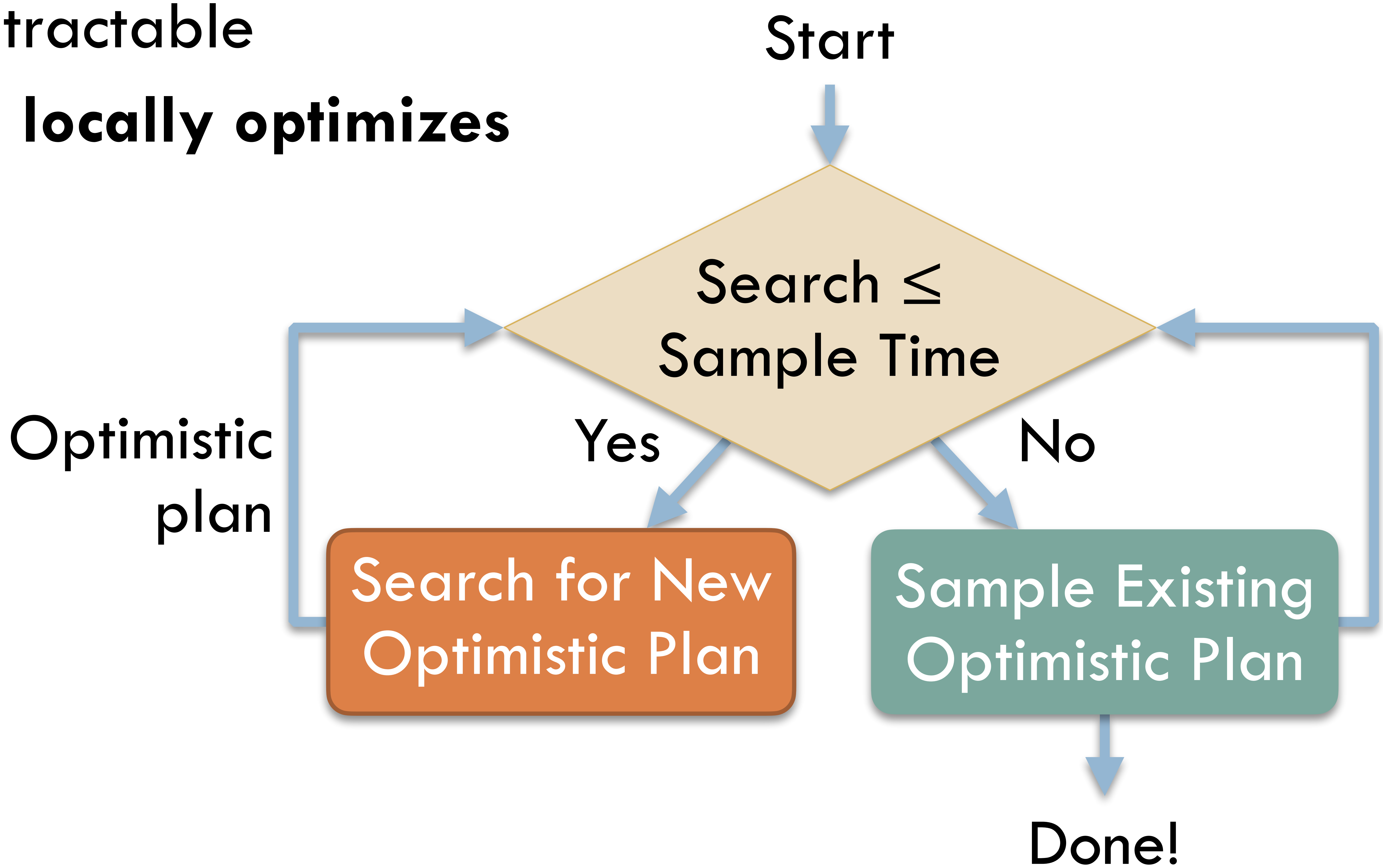
Problems with Tight Constraints

- Example: **pack** 5 blue blocks into a small green region
- Optimistic plan may be **feasible** but require a substantial amount of **rejection sampling**
- Binding algorithm would require **many iterations**



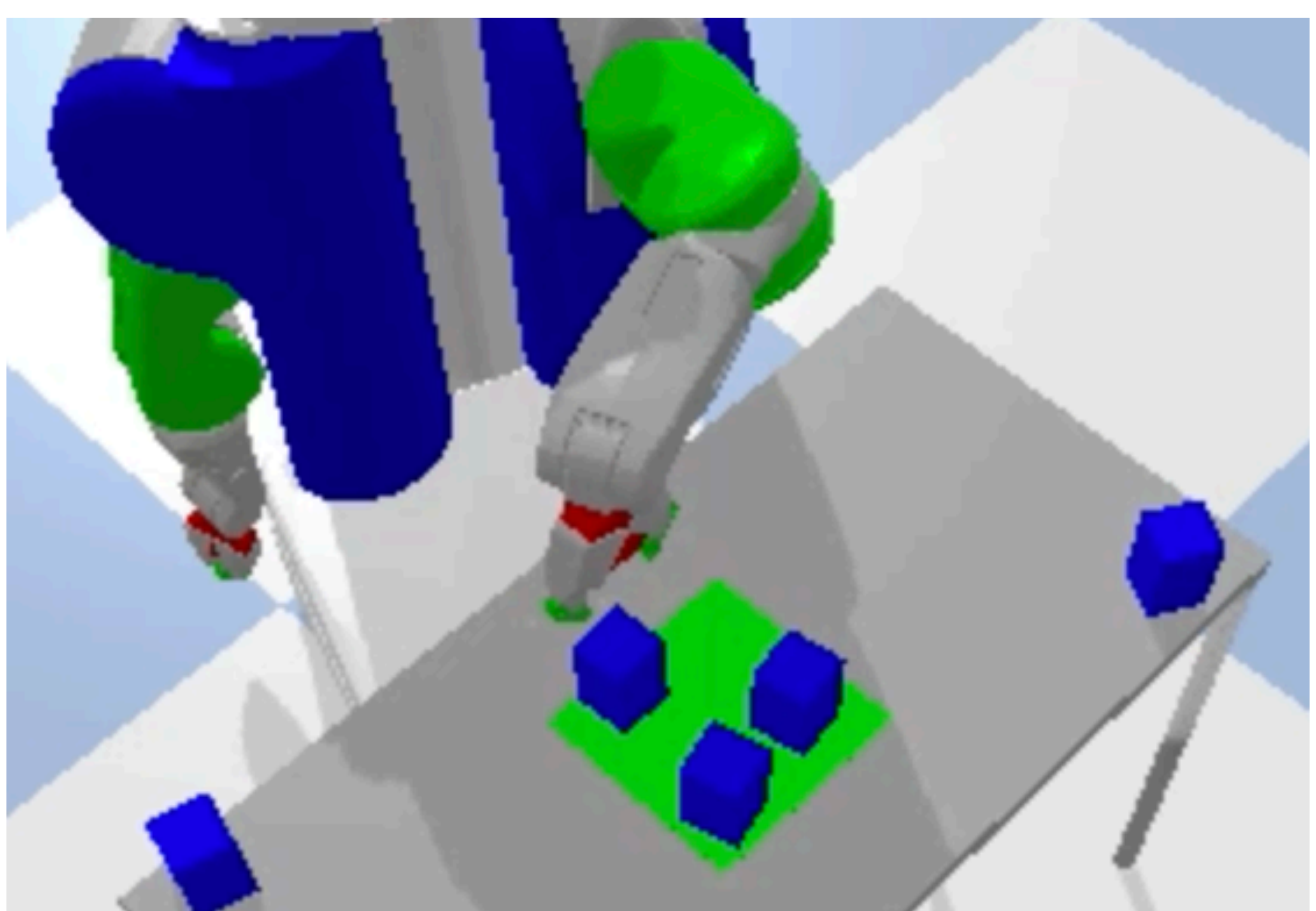
Adaptive Algorithm

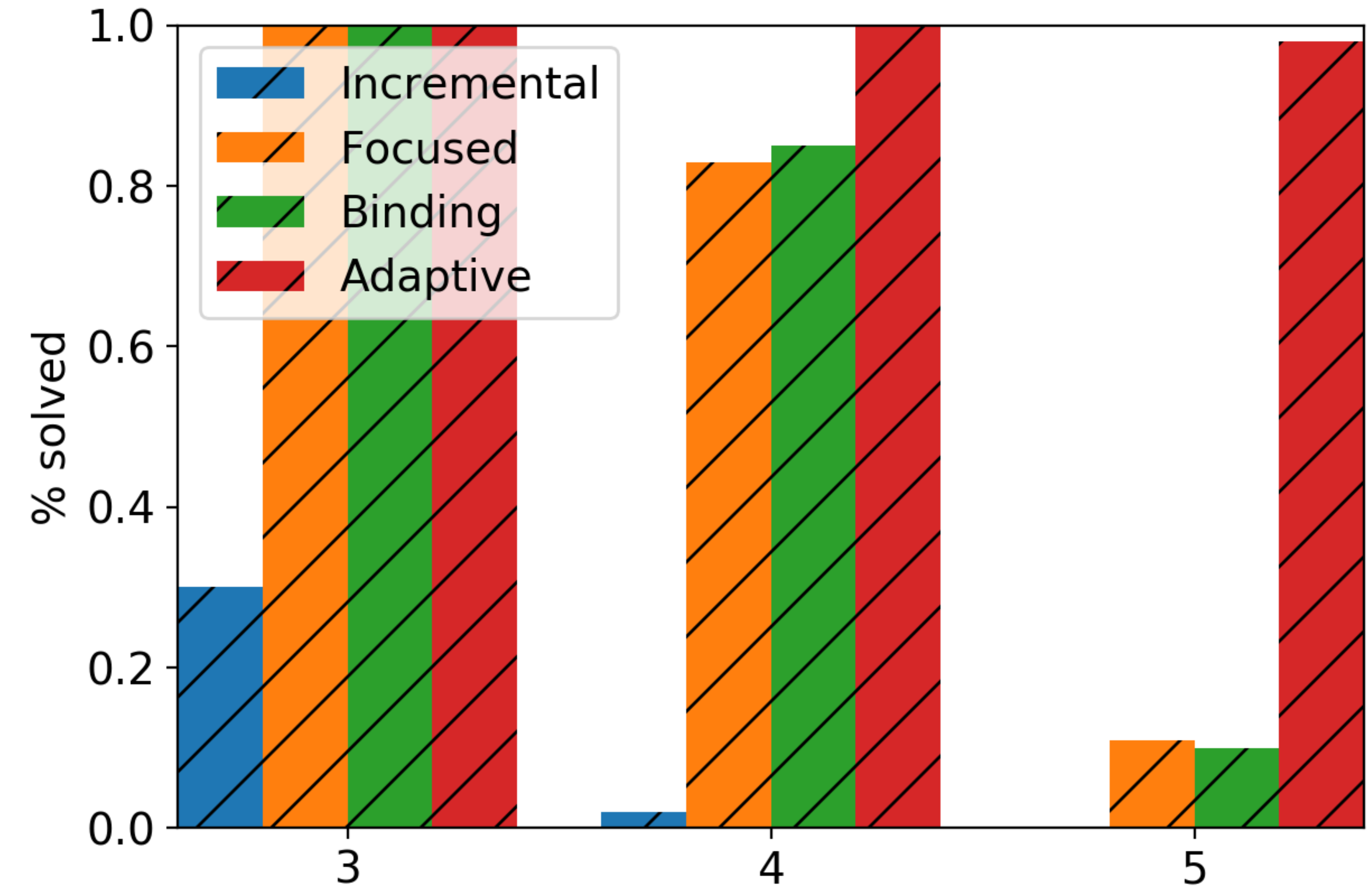
- **Balance computation time** spent searching and sampling
  - Adapts **online** to overhead of each phase per problem
- **Gradually instantiate** with new objects to keep finite PDDL problems small & tractable
- **Anytime** mode to **locally optimizes** for low-cost plans

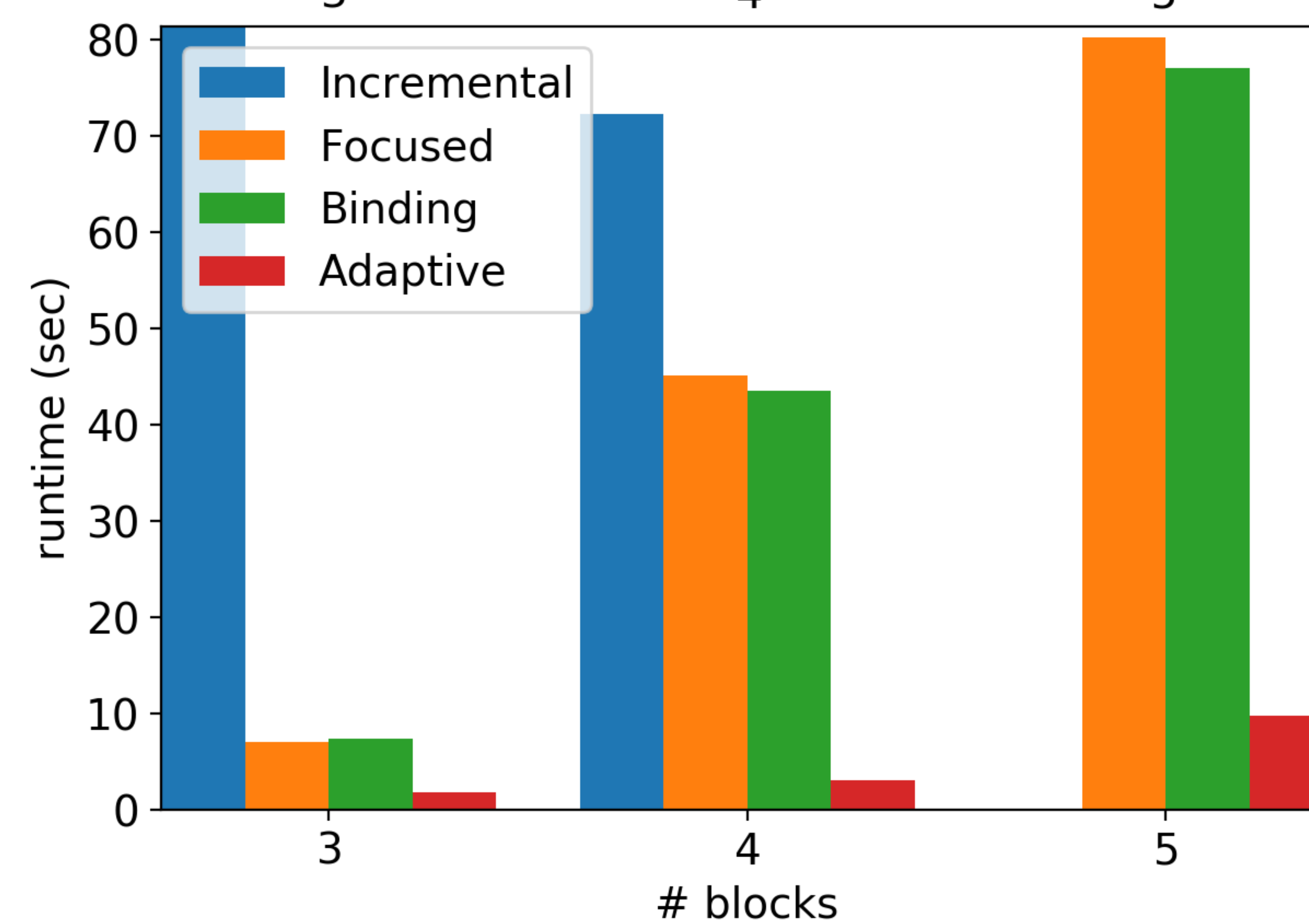


Experiments: Coverage & Runtime

- Scale the **number** of blue blocks while the green region maintains its size
- **Adaptive** solves the most problems (and most quickly) for most difficult (5 blocks)







Rovers Domain & Takeaways

- **PDDLStream:** generic extension of PDDL that supports **sampling procedures** as blackbox streams
- **Optimistic** planning intelligently queries only a small number of samplers
- **Adaptively** balancing searching & sampling performs best