
# BACKWARD-FORWARD SEARCH FOR MANIPULATION PLANNING

**Caelan Garrett**, Tomás Lozano-Pérez, and Leslie Kaelbling
MIT CSAIL - IROS 2015

# Hybrid Planning

- **Mixed discrete/continuous state & actions**

# Hybrid Planning

- **Mixed discrete/continuous state & actions**
- e.x. robotic planning
  - Continuous - robot configuration, object poses, grasp transforms, …
  - Discrete - holding object label, object cleaned/ cooked, …

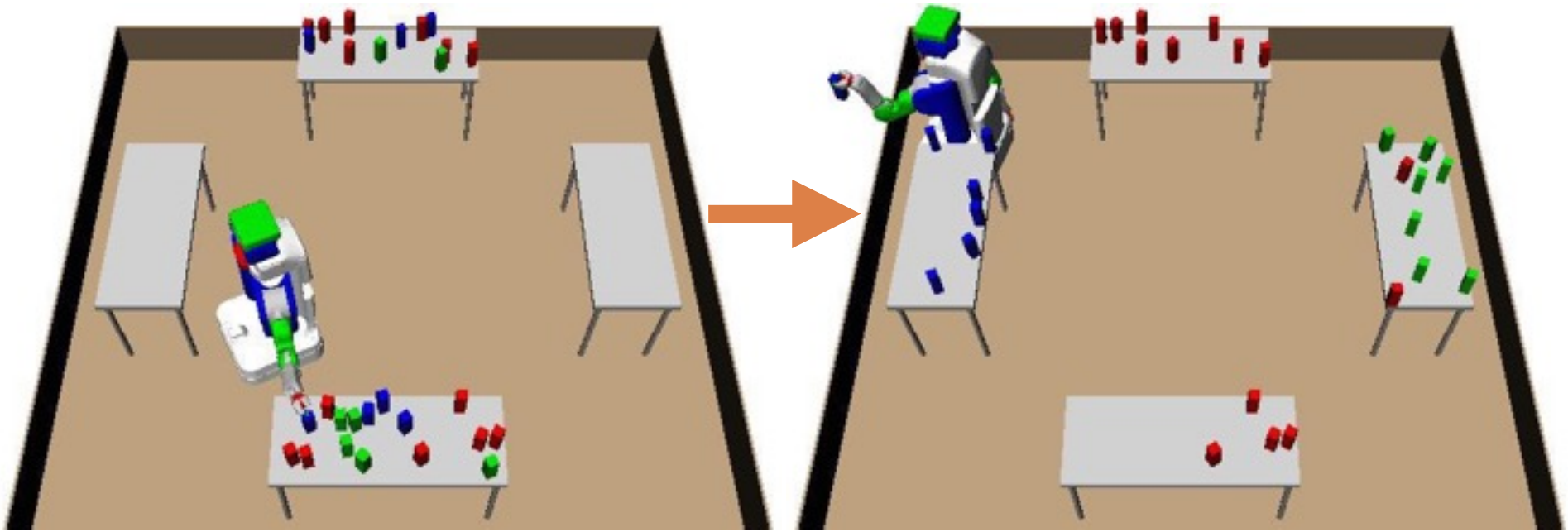
# Hybrid Planning

- **Mixed discrete/continuous state & actions**
- e.x. robotic planning
  - Continuous - robot configuration, object poses, grasp transforms, …
  - Discrete - holding object label, object cleaned/ cooked, …

- **Hybrid Backward-Forward (HBF) algorithm**
  - Probabilistically complete
  - Efficient empirical performance

# High-Dimensional Manipulation

Separate **blue blocks** and **green blocks**

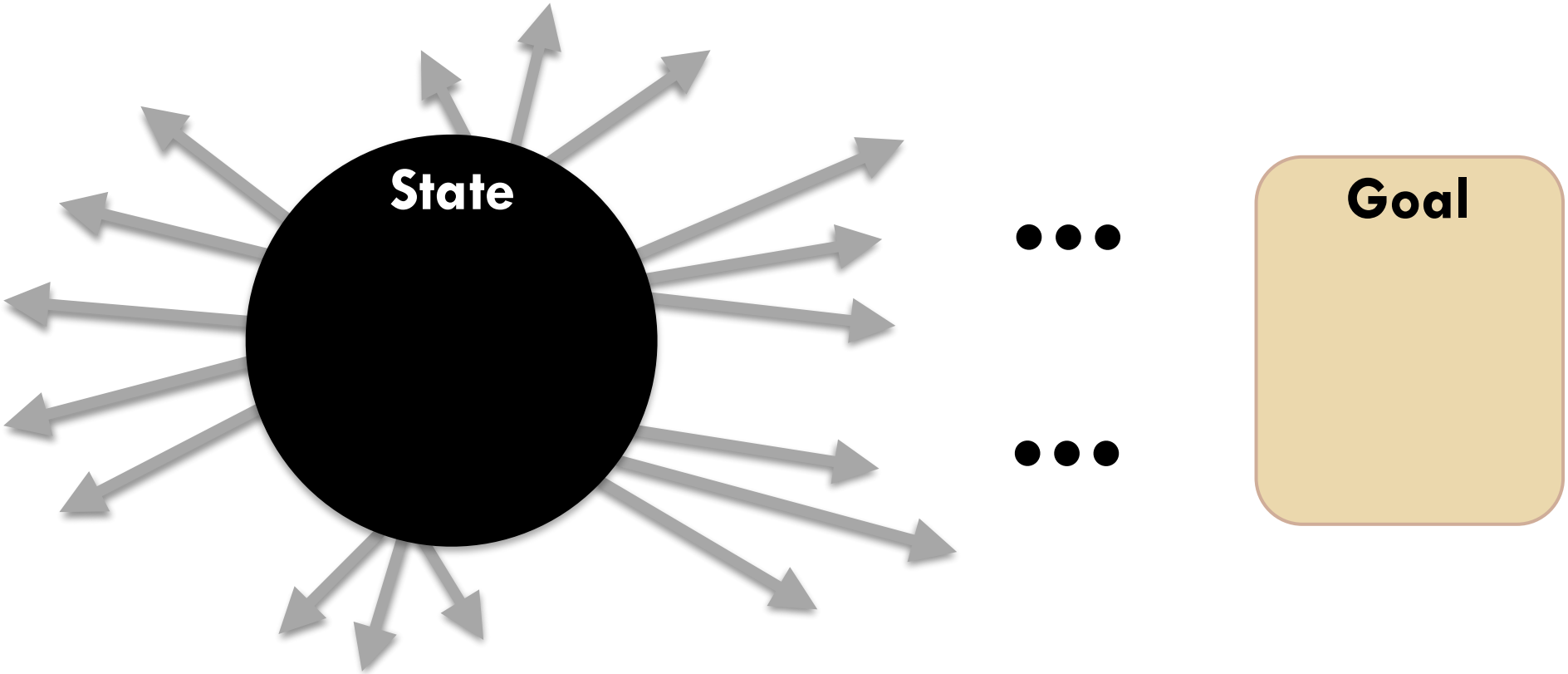


Early state

Late state

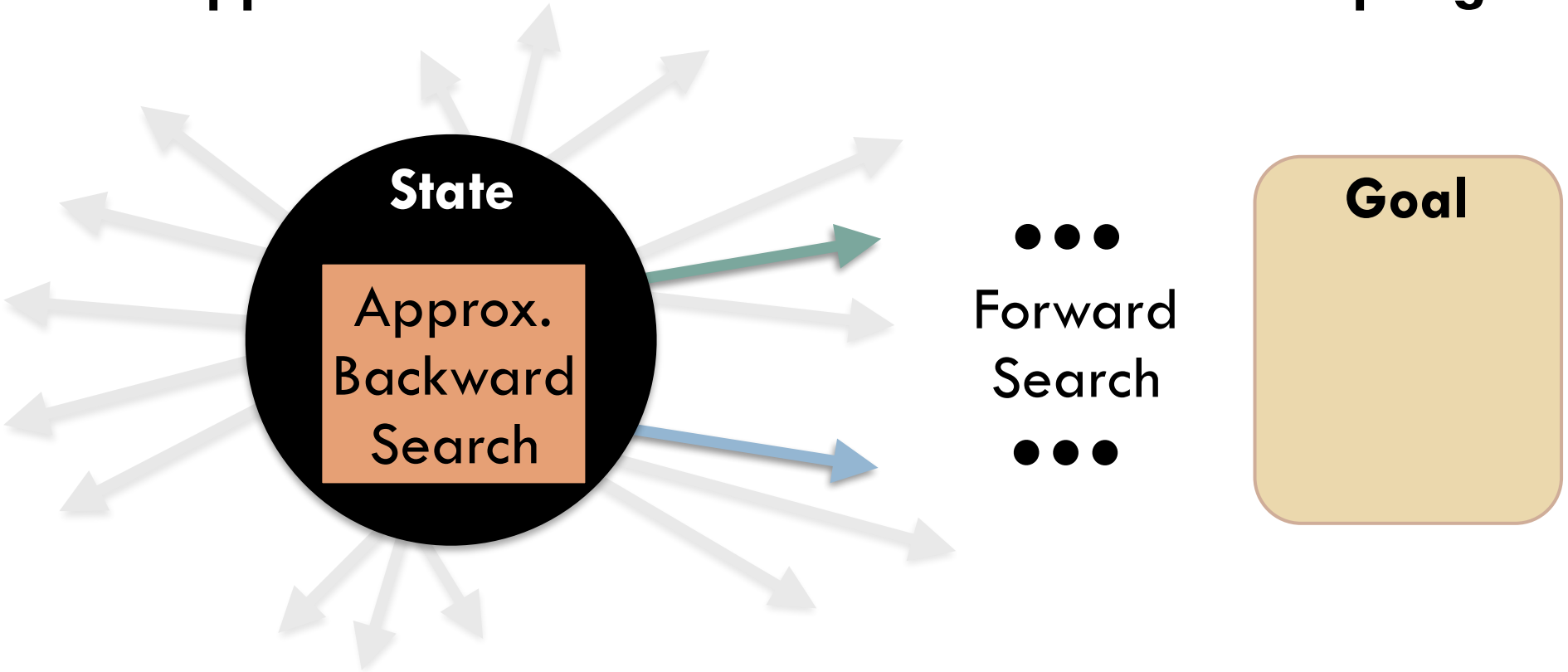
# Infinite Branching Factor and Long Horizon

- Pure forward or backward search overwhelmed
- Unguided action sampling ineffective



# Infinite Branching Factor and Long Horizon

- Pure forward or backward search overwhelmed
- Unguided action sampling ineffective
- **Approximate backwards search focuses sampling**



# Action Template Representation

**Place**(config, obj, transform)

# Action Template Representation

**Place**(config, obj, transform)

*constraints*

robot = config

holding = obj

grasp = transform

# Action Template Representation

**Place**(config, obj, transform)

*constraints*

robot = config

holding = obj

grasp = transform

*effects*

holding = **None**

obj = **Pose**(config, obj, transform)

# Action Template Representation

**MoveHolding**(config1, config2, objA, transform)

*constraints*

robot = config1

holding = objA

grasp = transform

objB ∈ **CollisionFreePoses**(config1, config2, objA, transform, objB) *for* objB ≠ objA

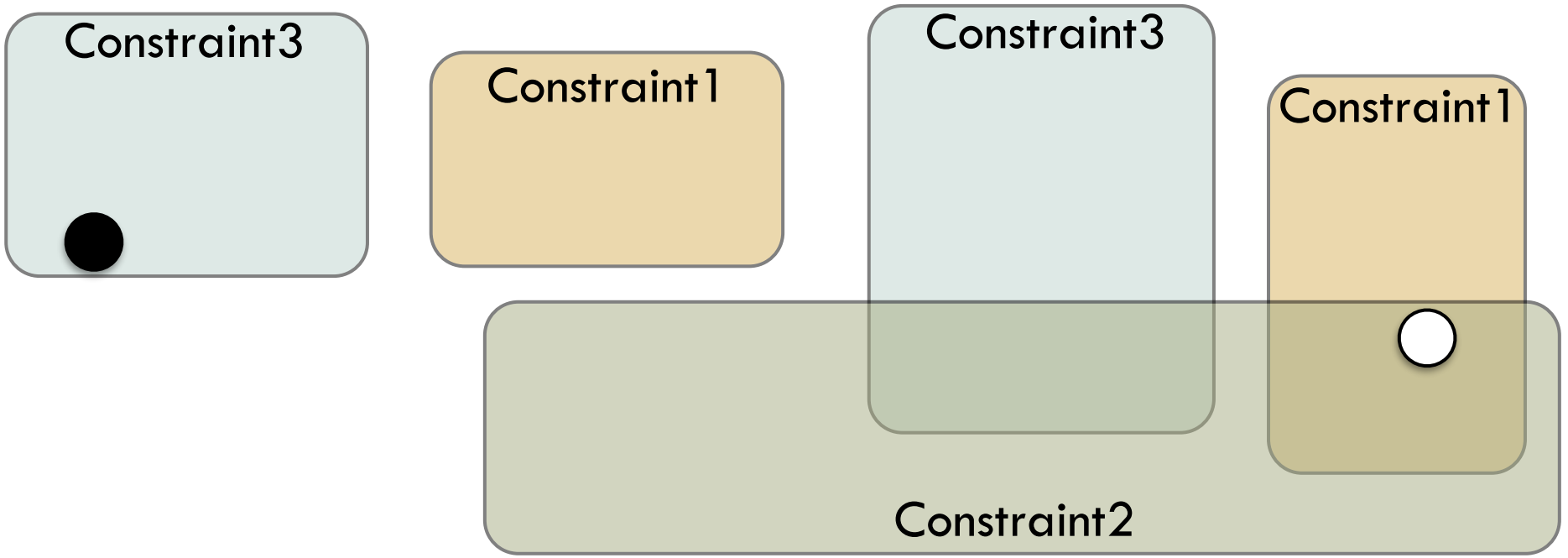
*effects*

robot = config2

objA = **Pose**(config2, objA, transform)

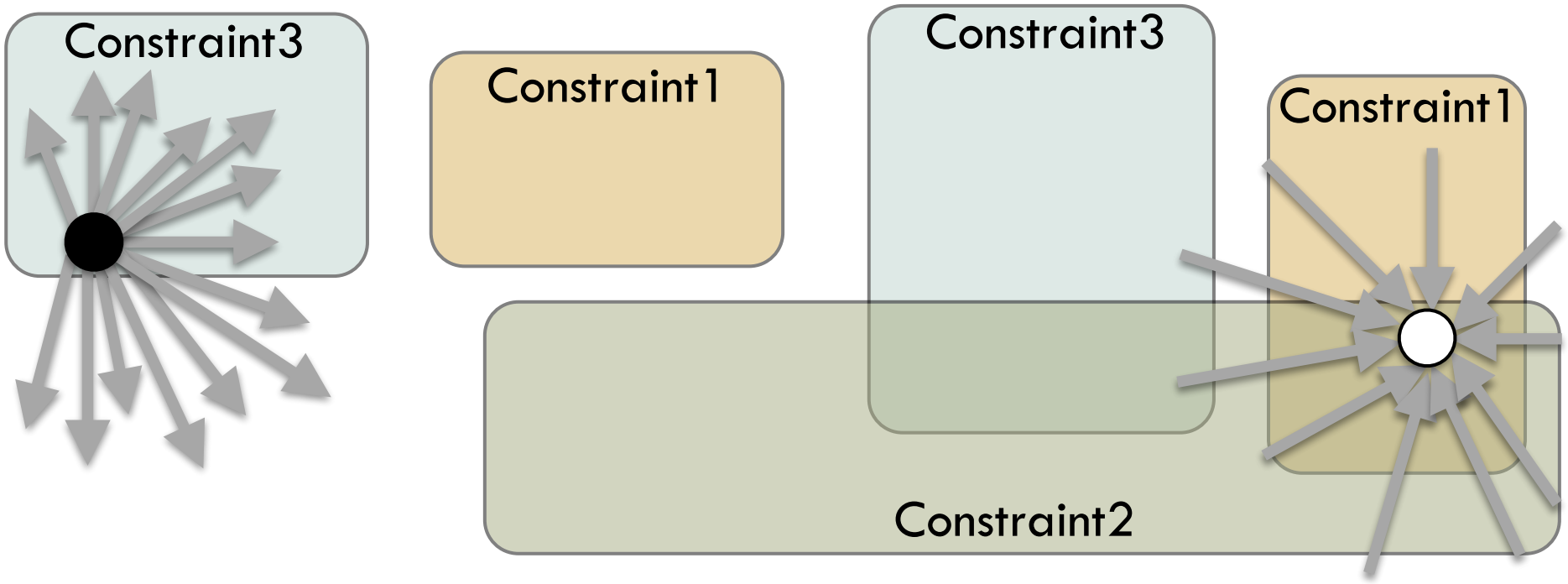
# Example Hybrid Planning Problem

- Constraints define regions of state-space
- Goal is to reach a state in the **intersection of Constraint1 and Constraint2**



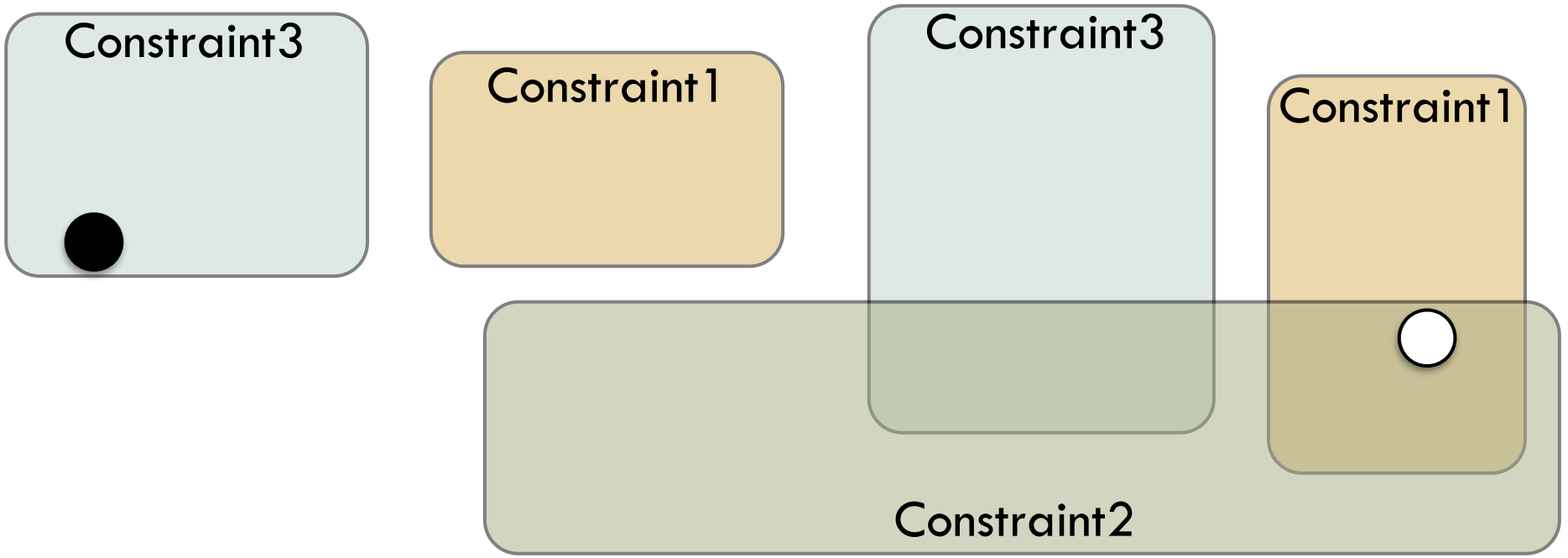
# Unguided Search is Ineffective

- Pure forward or backward search **overwhelmed** by infinite branching factor and long horizon



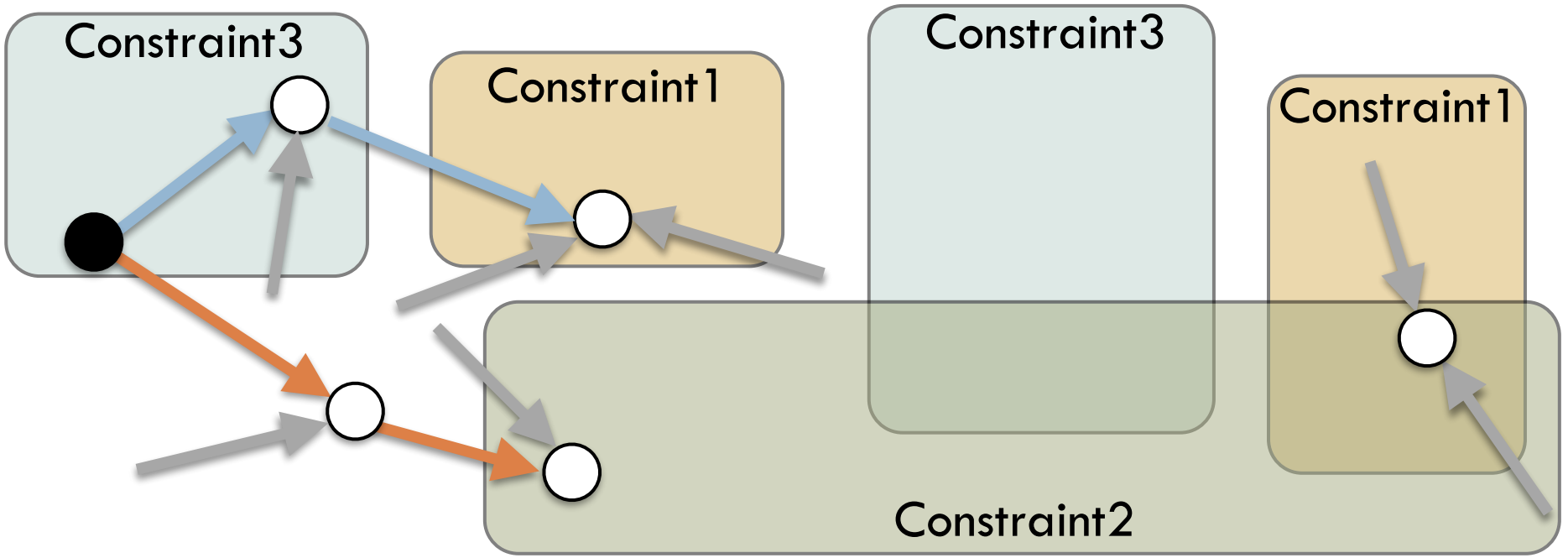
# Hybrid Backward-Forward (HBF)

- Backwards approximation - **constraint independence**



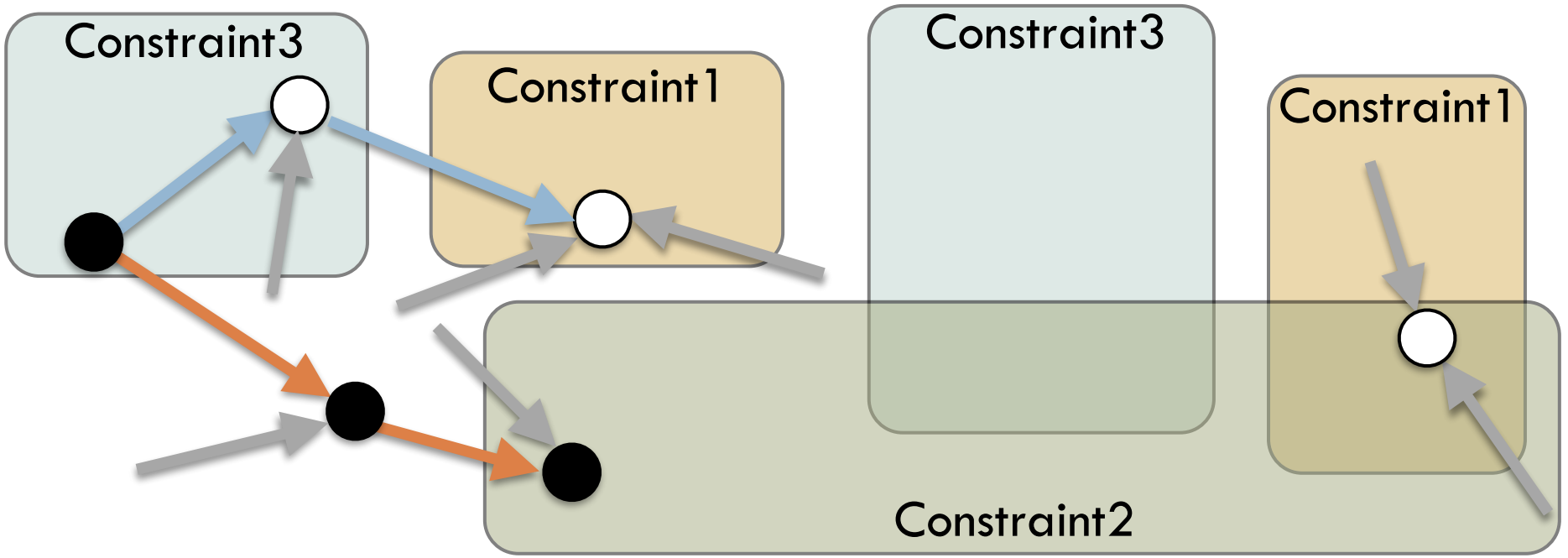
# Hybrid Backward-Forward (HBF)

- Backwards approximation - **constraint independence**
- Long problem becomes **many short problems**



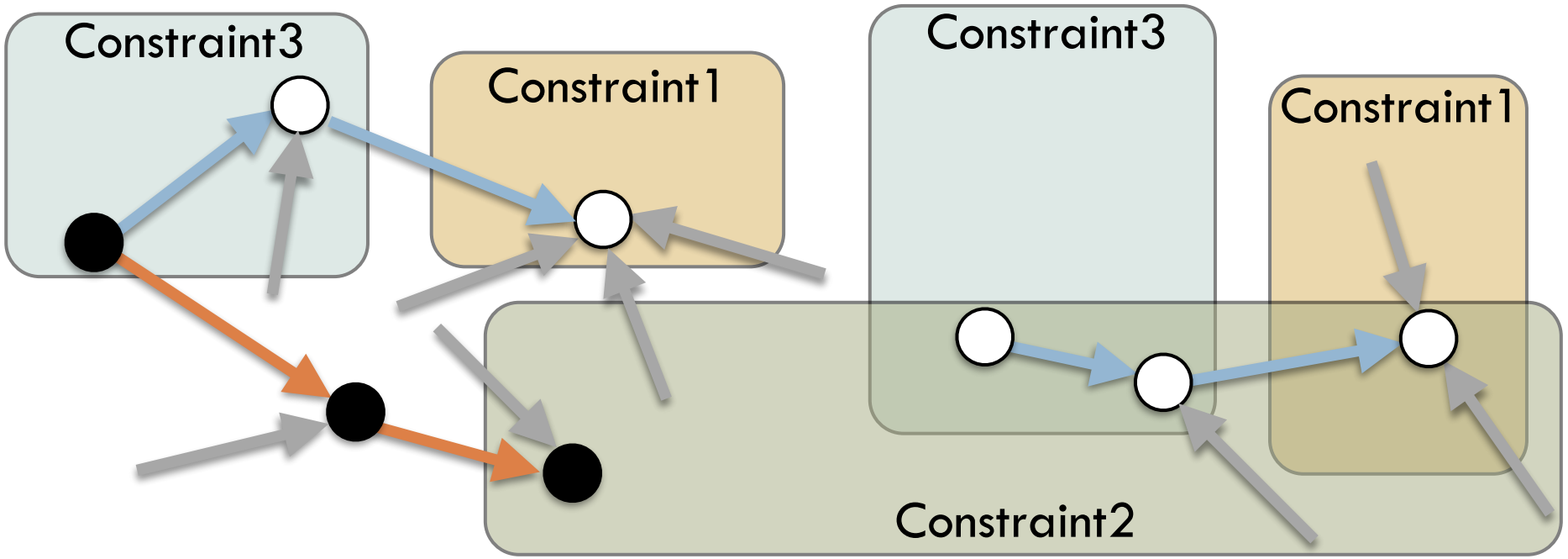
# Hybrid Backward-Forward (HBF)

- Backwards approximation - **constraint independence**
- Long problem becomes **many short problems**



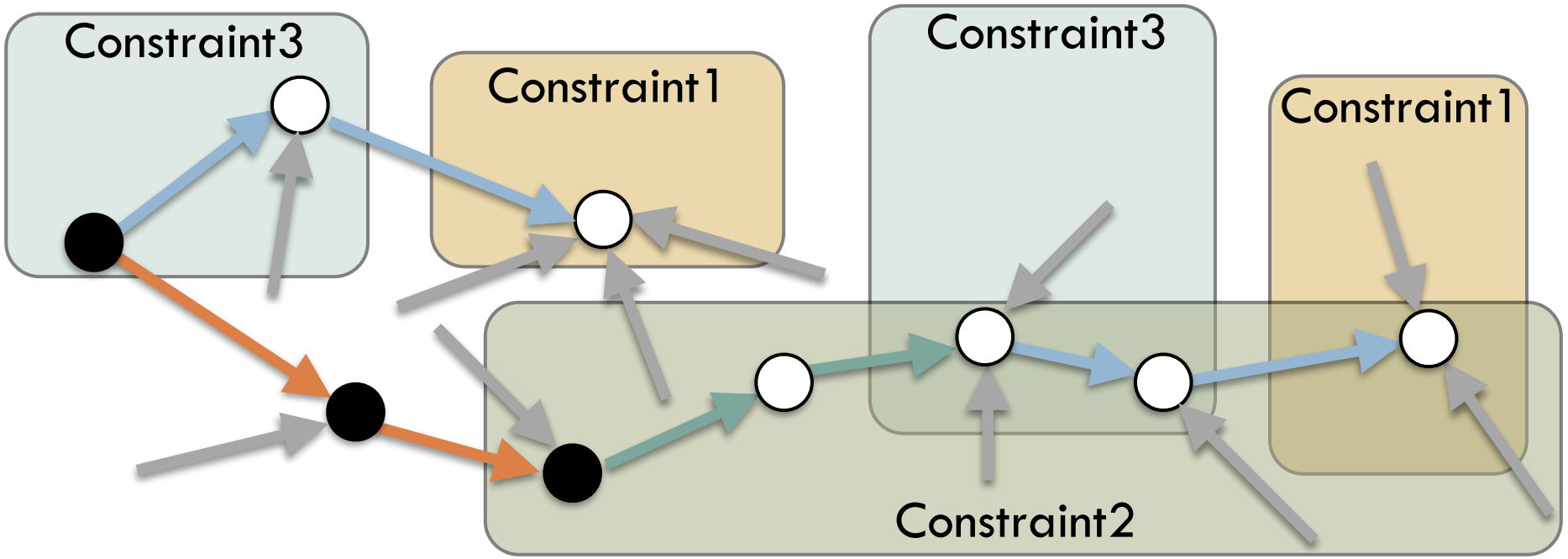
# Hybrid Backward-Forward (HBF)

- Backwards approximation - **constraint independence**
- Long problem becomes **many short problems**
- Forward search **resolves** approximation errors



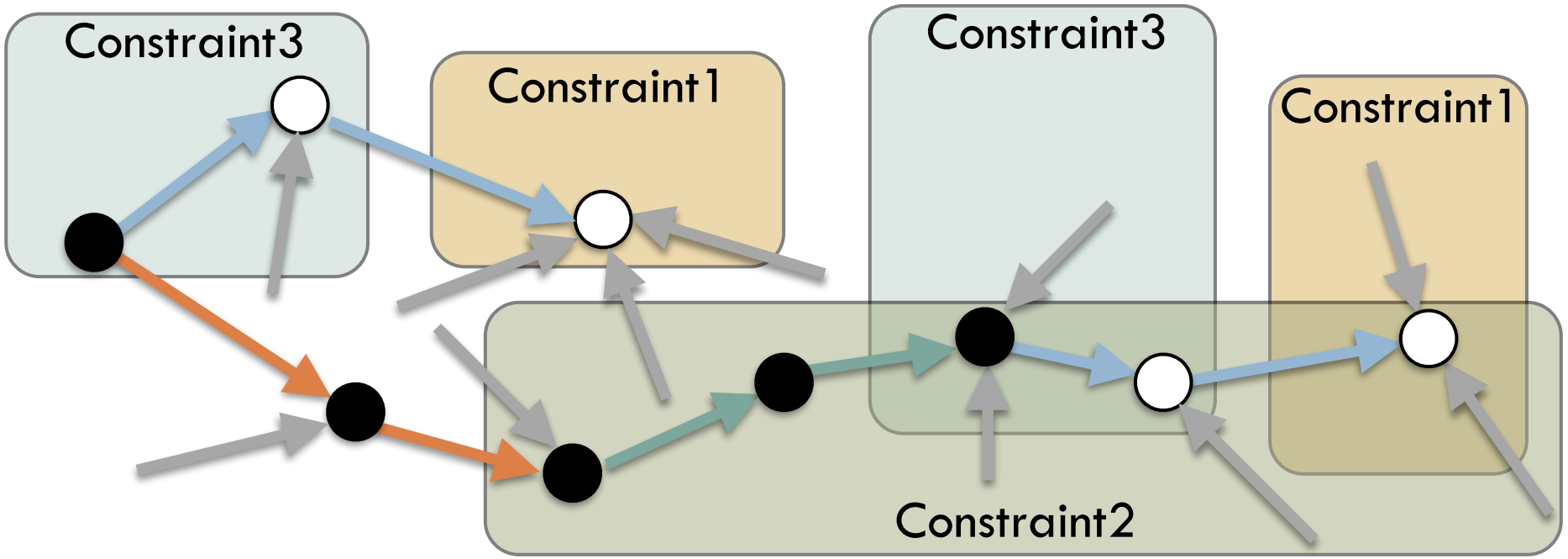
# Hybrid Backward-Forward (HBF)

- Backwards approximation - **constraint independence**
- Long problem becomes **many short problems**
- Forward search **resolves** approximation errors



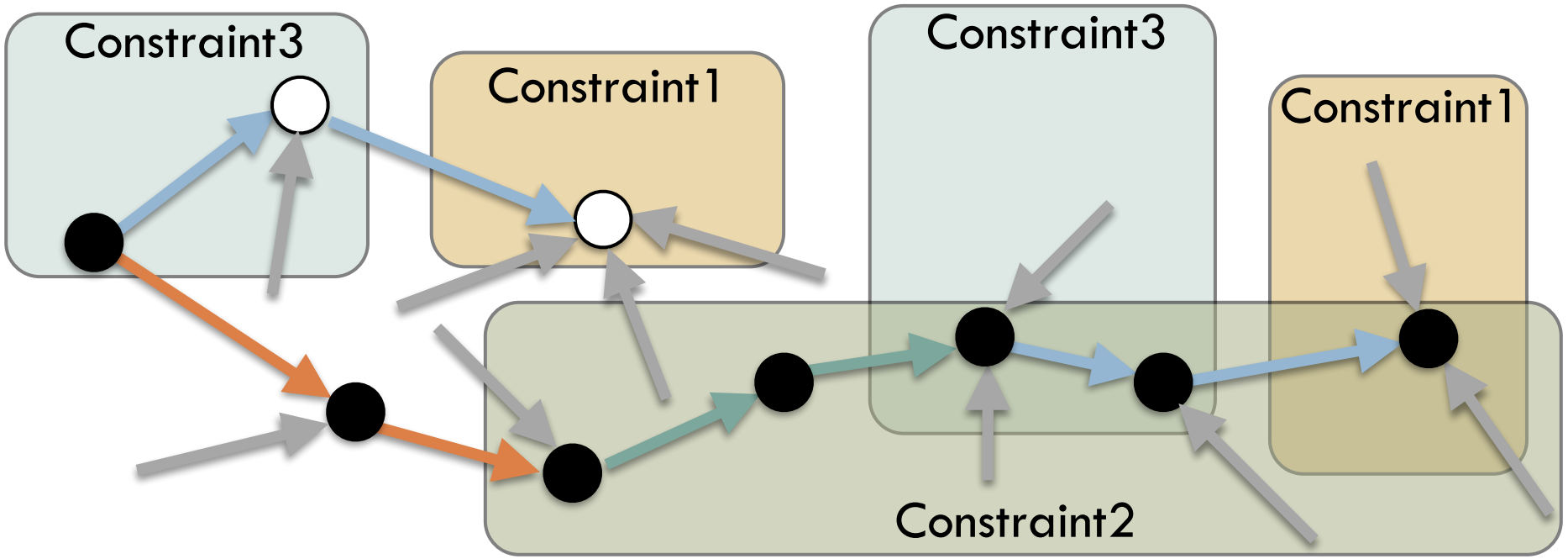
# Hybrid Backward-Forward (HBF)

- Backwards approximation - **constraint independence**
- Long problem becomes **many short problems**
- Forward search **resolves** approximation errors



# Hybrid Backward-Forward (HBF)

- Backwards approximation - **constraint independence**
- Long problem becomes **many short problems**
- Forward search **resolves** approximation errors



# Backwards Search Algorithm

**BackwardsSearch**(*state*, *goal-constraints*, *action-templates*):

1. *Queue* initialized to *goal-constraints*
2. **Repeat**
    3. Pop a *constraint* from the *queue*
    4. Sample *actions* from each *action-template* that achieve the *constraint*
    5. **Yield *actions* applicable from *state* to forward search**
    6. Add the new action *constraints* not satisfied by *state* to the *queue*

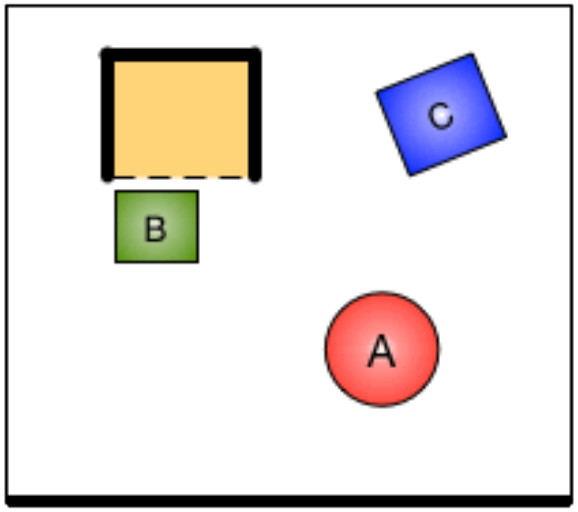
# Action Sampling

- Need to generate actions that satisfy a constraint

- For manipulation, we sample:
  - Configs/grasps/poses that **satisfy** the constraint
    - *Monte Carlo rejection sampling*
  - Other **relevant** configs/grasps/poses
    - *Current state, intersection of manifolds*
  - Actions that **connect** these values
    - *Blackbox motion planner (e.x. RRT)*

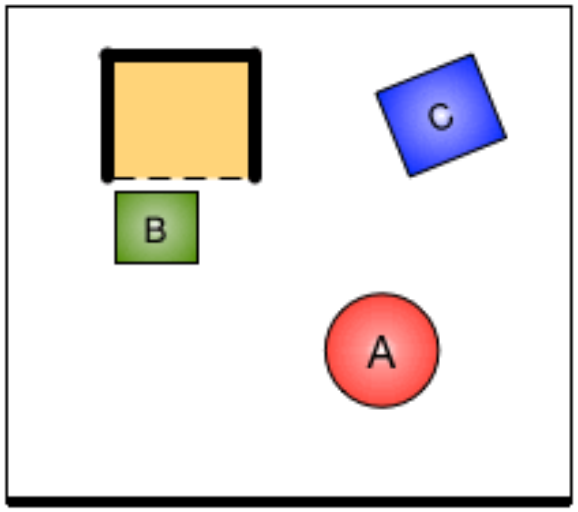
- Reuse previously sampled values/actions when possible

# Push A into Yellow Cabinet




- Robot is a point
- Goal - object A in yellow cabinet
  - Need to move object B!
- **Trace backwards search**

# Backwards Search - Iteration 1

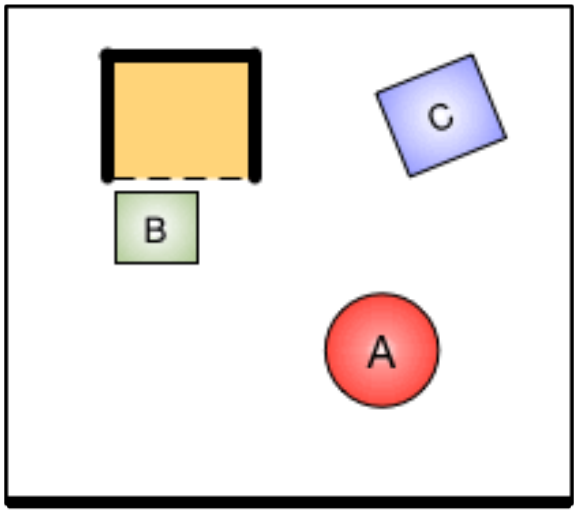



- Search starts at the goal constraint

A in cabinet

holding = **None** | A = poseA0 | B = poseB0 | C = poseC0 | robot = config0

# Backwards Search - Iteration 1

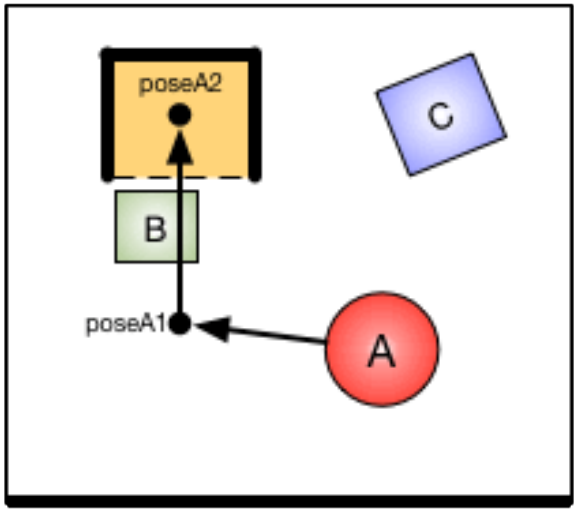



- Pop constraint “A in cabinet” from the queue

A in cabinet

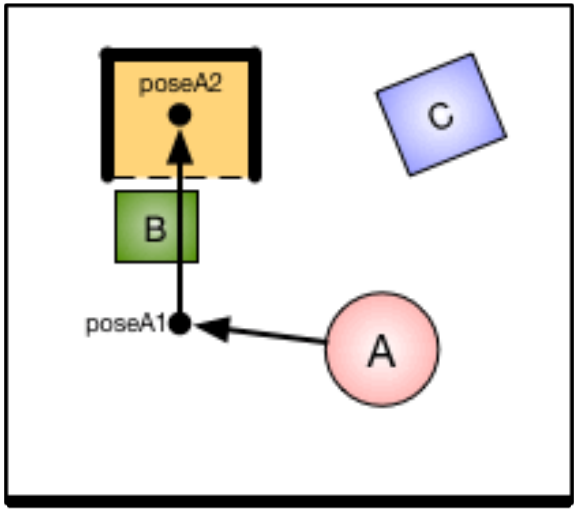
holding = **None** | A = poseA0 | B = poseB0 | C = poseC0 | robot = config0

# Backwards Search - Iteration 1



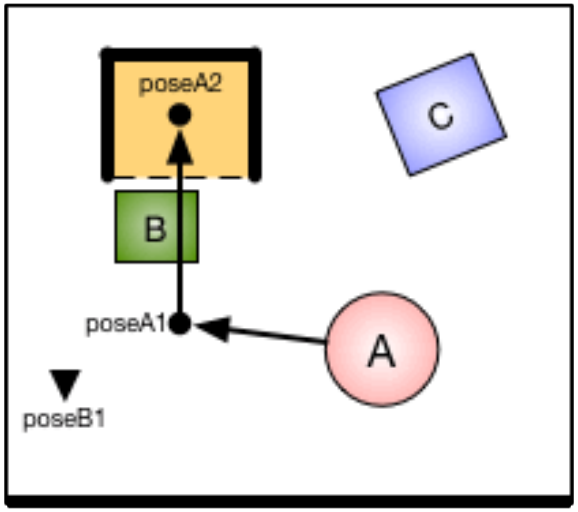

- Sample **Push** actions
- Add new constraints to queue

# Backwards Search - Iteration 2

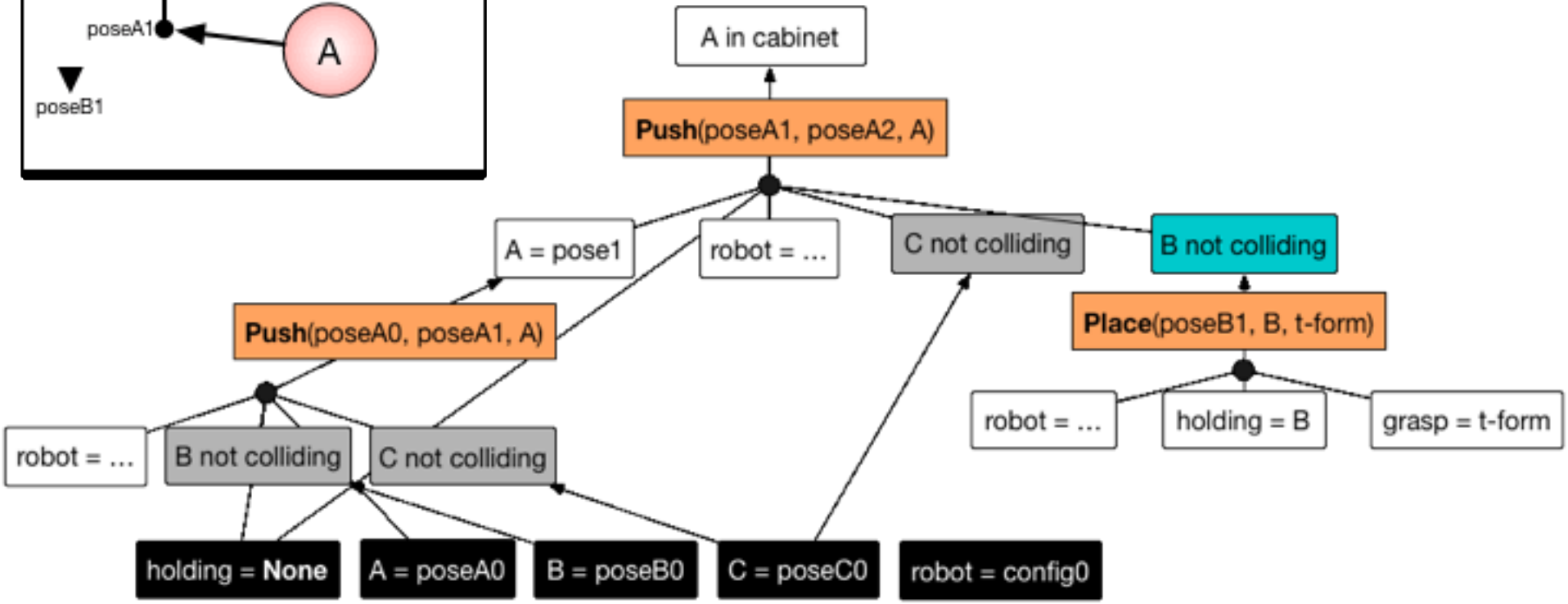



- Pop constraint “B not colliding” with the push from the queue

# Backwards Search - Iteration 2

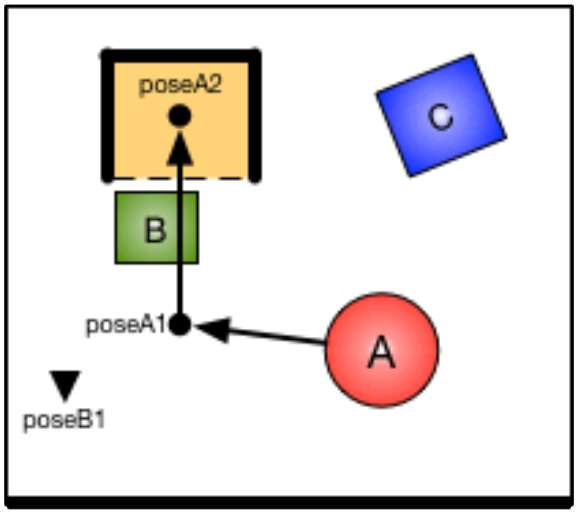



- Sample collision-free **Place** for B
- Add new constraints to the queue

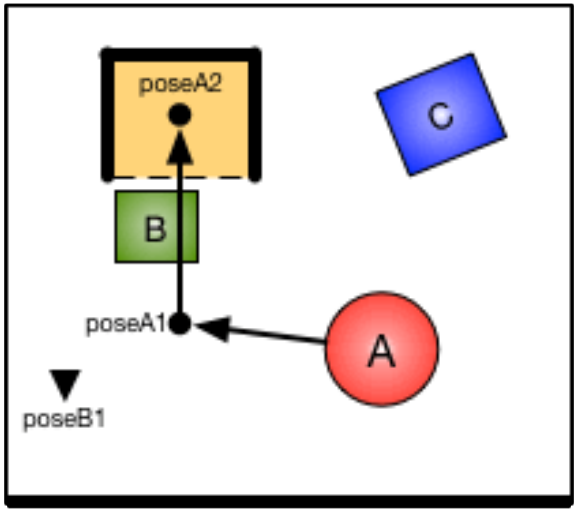


# Backwards Search - Continued…

- Continue process…



# Backwards Search - Continued…




- Move actions are forward search successors

# Forward Search Algorithm

- Forward state-space heuristic search
  - Hill climbing, best-first search, …

- Backward search focuses forward search by:
  - Identifying **successor actions**
  - Incidentally creating **approximate plans**
    - **Heuristic** is length of an approximate plan (idea from AI planning)

# Probabilistic Completeness

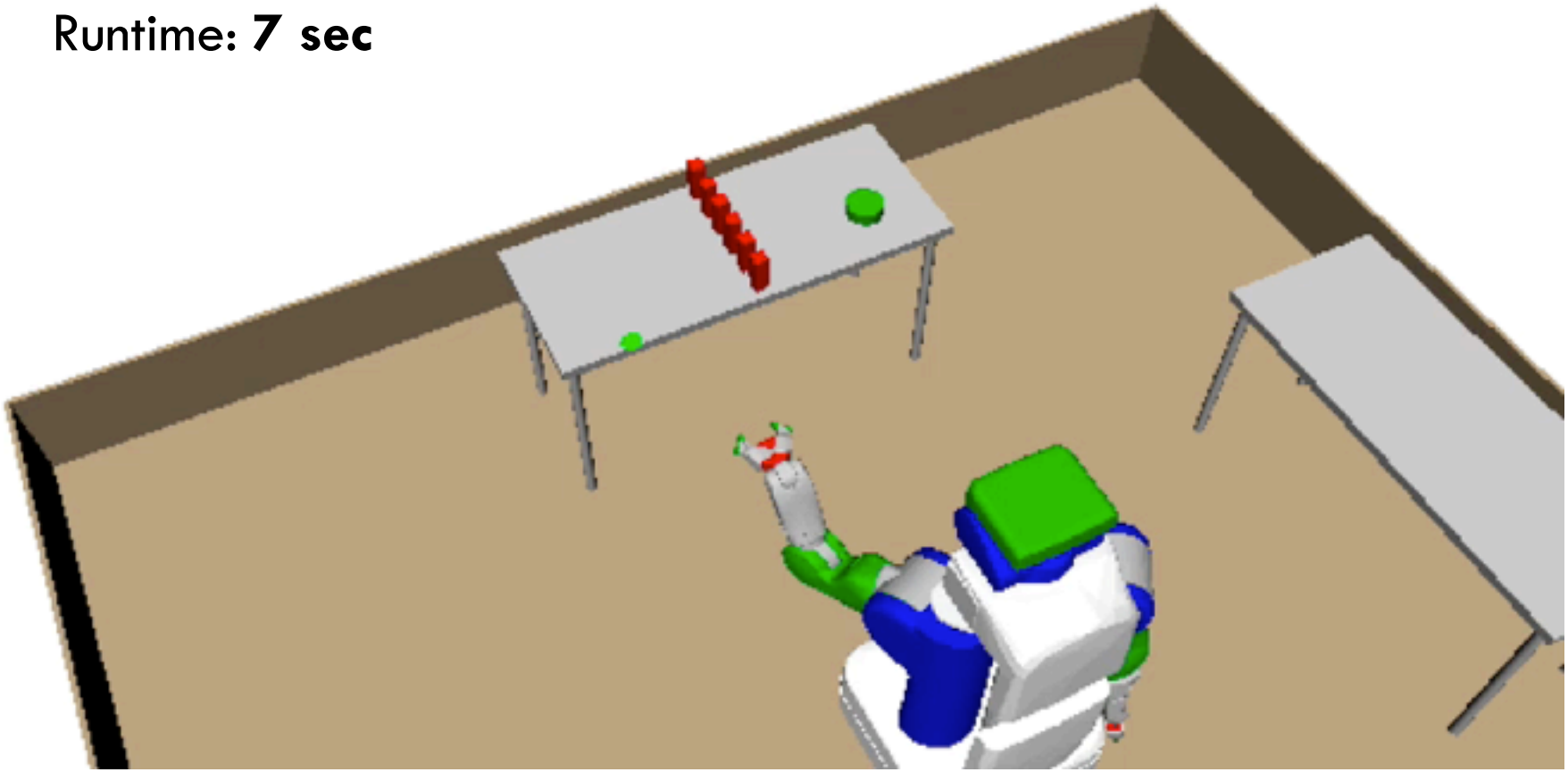
- **Definition** - A hybrid planning problem is **robustly feasible** if there exists sequence of action templates with a **family of solutions** such that for all state and pre-image pairs on a solution, the solution family actions that partially satisfy the state and achieve the pre-image have **nonzero measure**

- **Lemma -** the backwards search performed from any state in the solution family will generate a successor action in the solution family with a **probability of one** as $n \to \infty$

- **Theorem -** HBF will solve any robustly feasible hybrid planning problem with a **probability of one** as $n \to \infty$
  - *Proof* - inductively apply the lemma

# Non-prehensile Actions

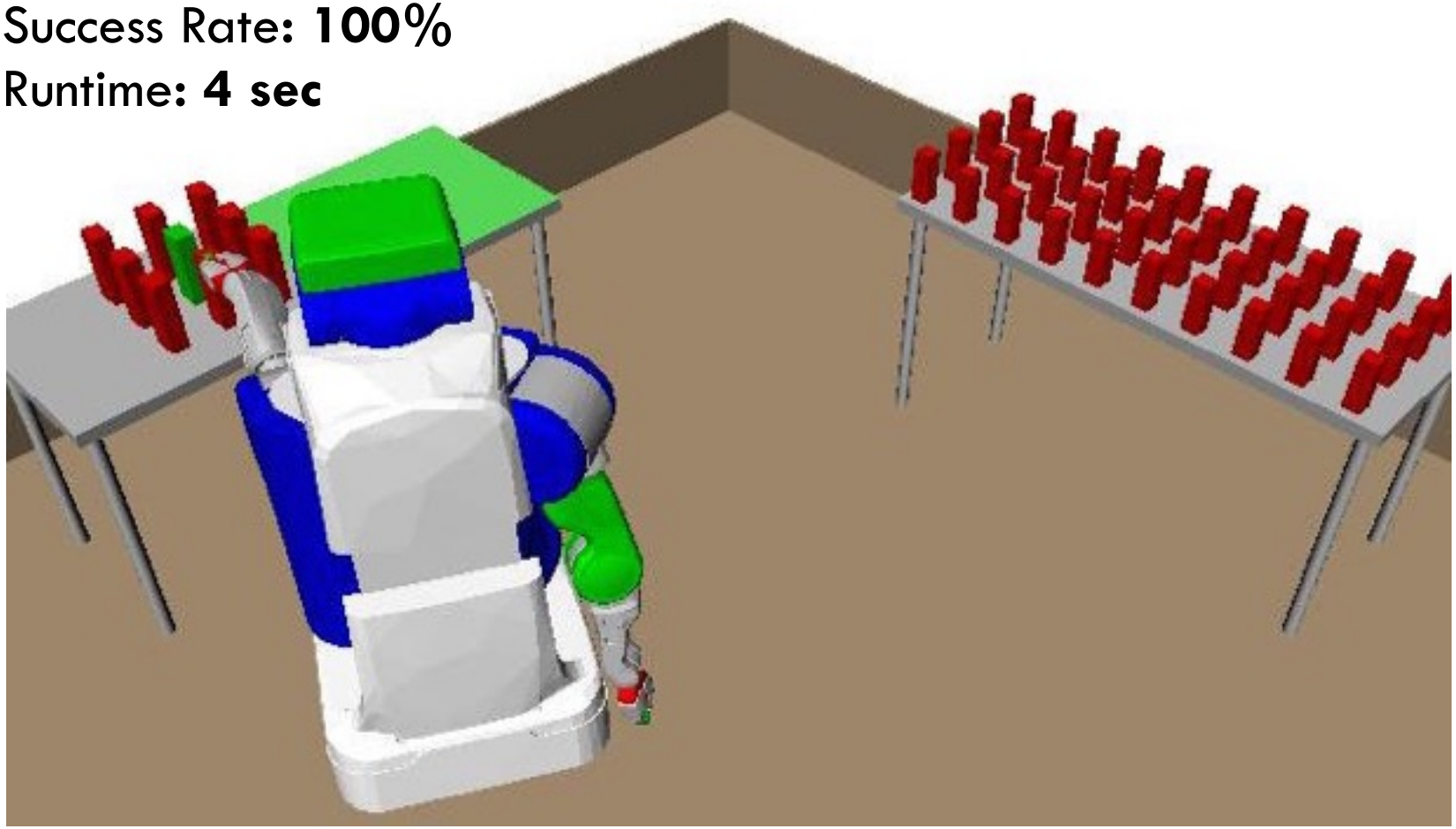
Push **green cylinder** to **green dot**
Success rate: **100%**
Runtime: ***7* sec**



Python, 60 trials, 300 sec timeout, median runtime

# High-Dimensional State-Space

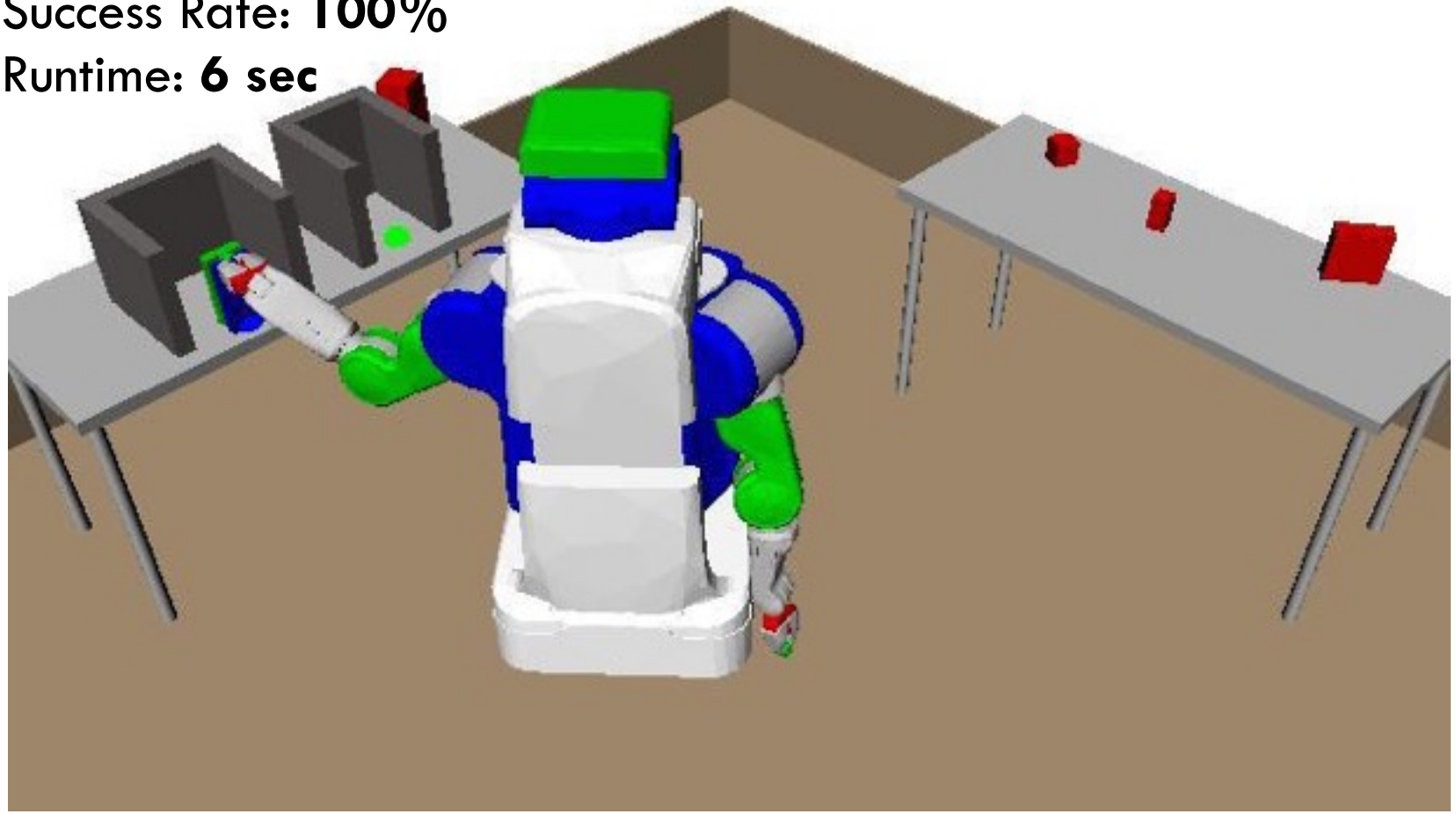
Pick and place **green block**
Success Rate: **100%**
Runtime: **4 sec**



Python, 60 trials, 300 sec timeout, median runtime

# Regrasping and Non-monotonicity

Move **blue block**, regrasp **green block**, and replace **blue block**
Success Rate: **100%**
Runtime: **6 sec**



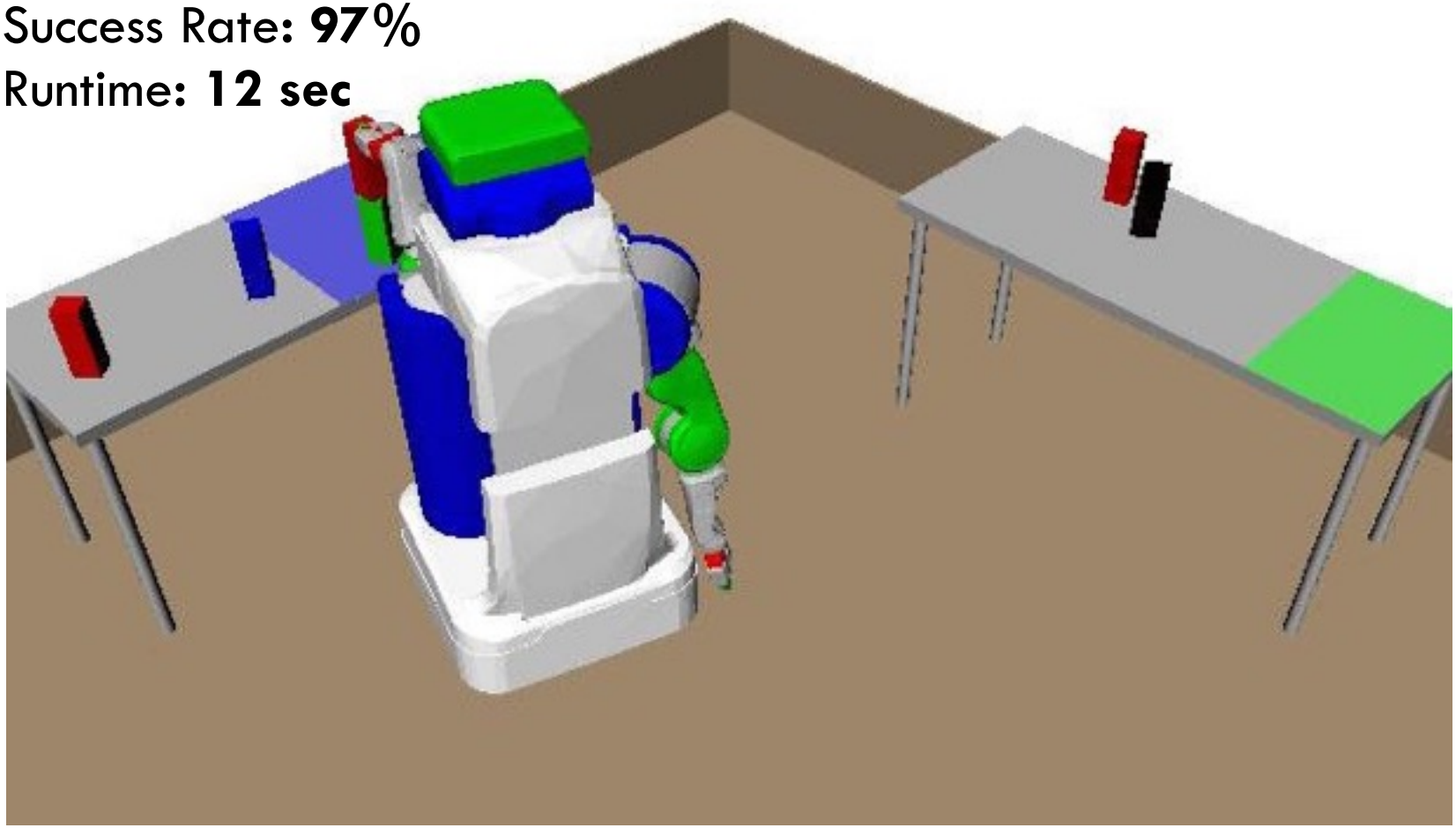
Python, 60 trials, 300 sec timeout, median runtime

# Dynamic Unstacking and Stacking

Unstack **red block** and stack **black block** on **blue block**

Success Rate: **97%**

Runtime: **12 sec**



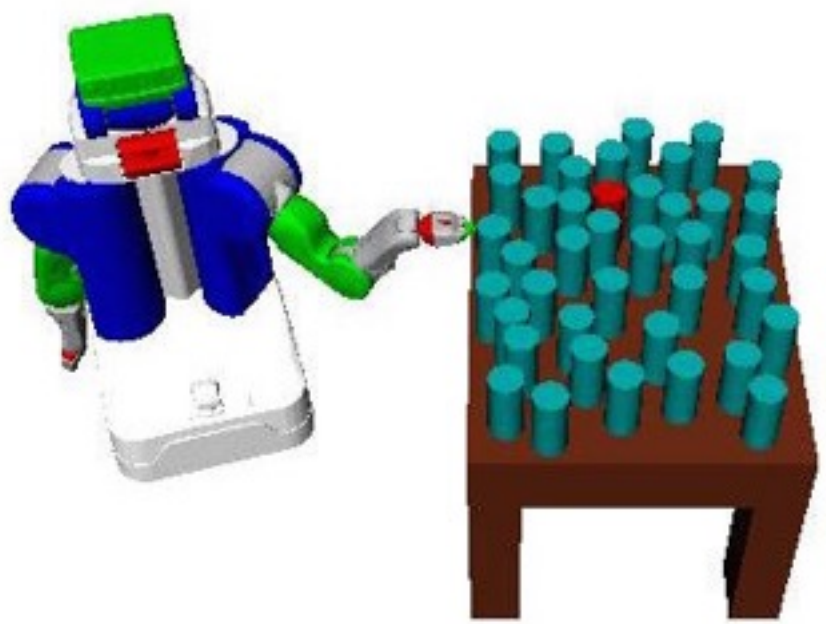
Python, 60 trials, 300 sec timeout, median runtime

# Comparison with Srivastava et al.

Grasp **red cylinder** on crowded table

*Single goal but must move many objects*

| | Success Rate | Runtime |
|---|---|---|
| Srivastava et al. | 63% | 68 sec |
| HBF | 98% | 23 sec |



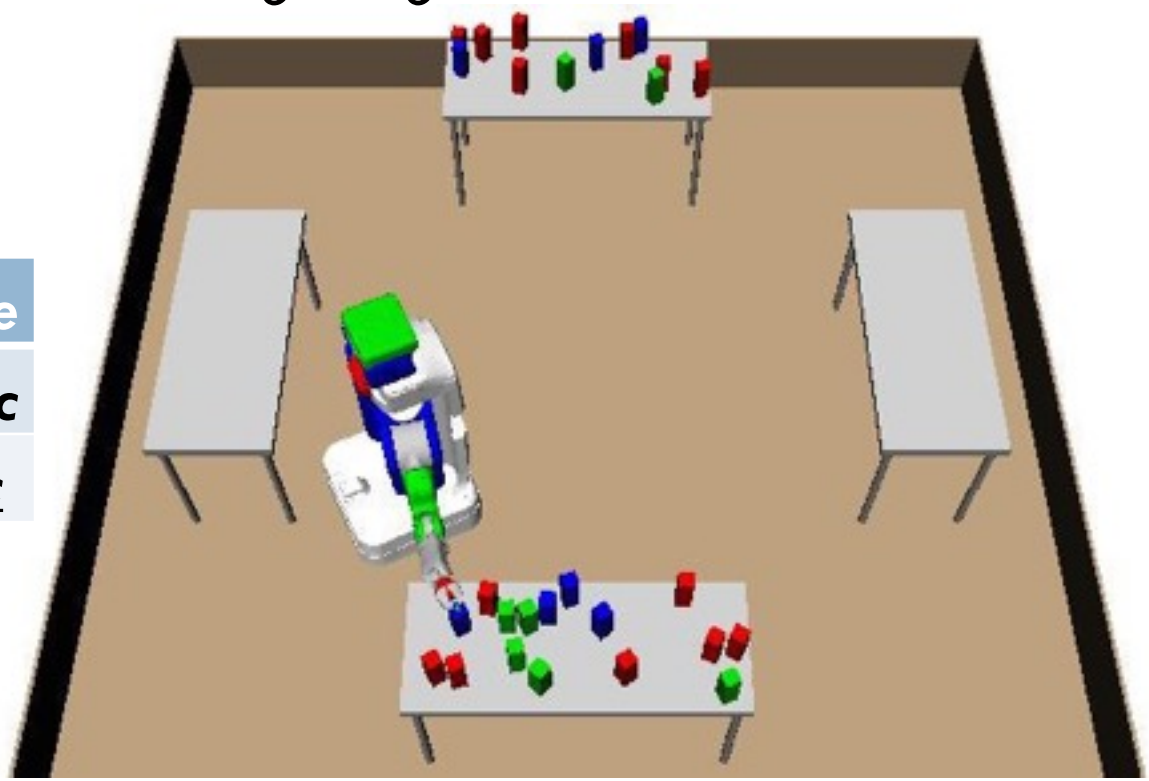
Python, 60 trials, 300 sec timeout, median runtime

# Comparison with FFRob

Separate **blue blocks** and **green blocks**

*Many goals and must order achieving the goals*

| | Success Rate | Runtime |
|---|---|---|
| FFRob | 84% | 157 sec |
| HBF | 100% | 82 sec |



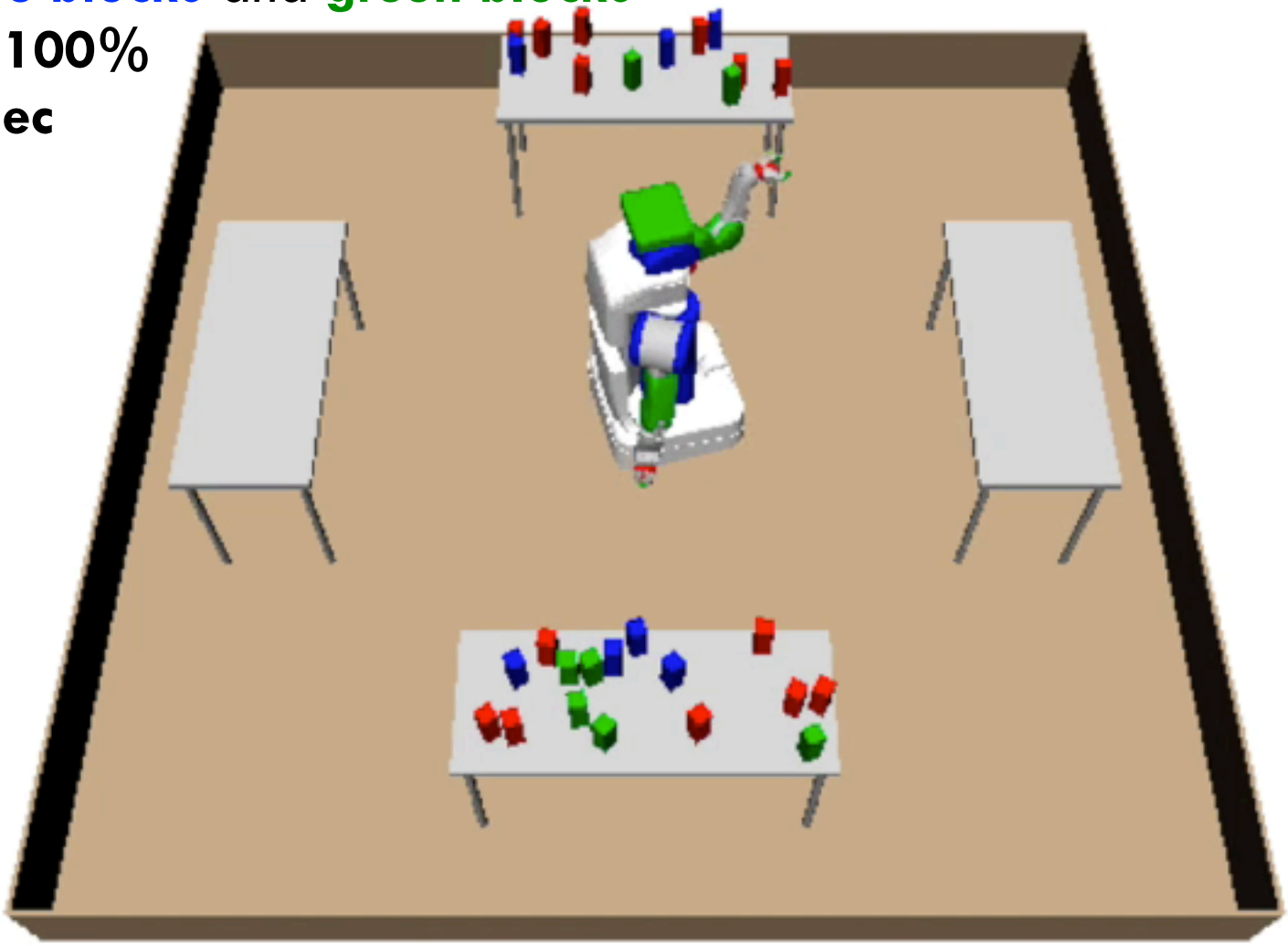
Python, 60 trials, 300 sec timeout, median runtime

# Takeaways

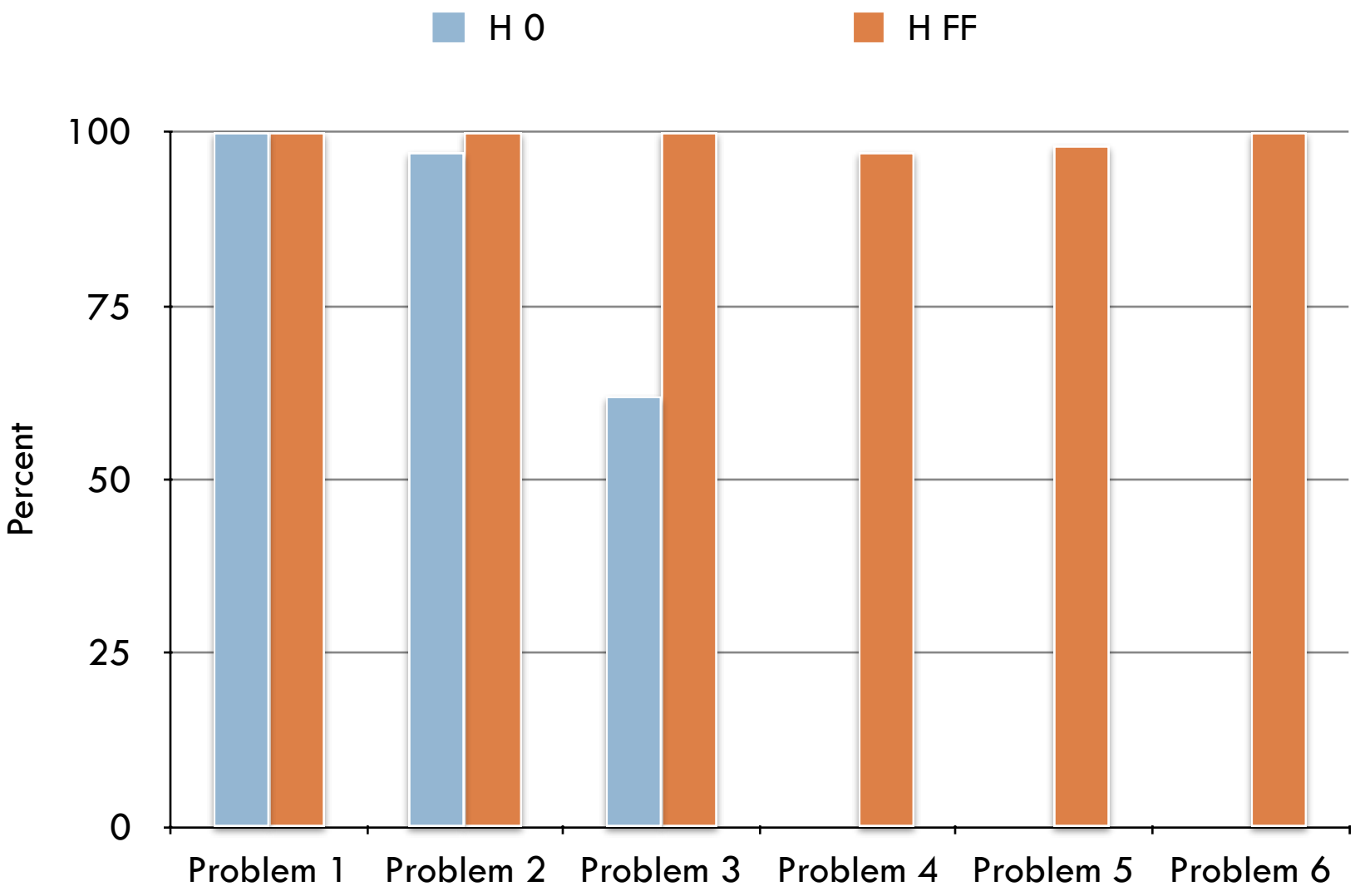
- General purpose hybrid planing algorithm (**HBF**)
  - **Approximate backward search**
    - Focuses successor actions to reduce branching factor
    - Gives heuristic cost
  - **Forwards search**
    - Resolves backwards approximations
- **Probabilistically complete**
- **Application** - efficiently solves manipulation problems

# Any Questions?

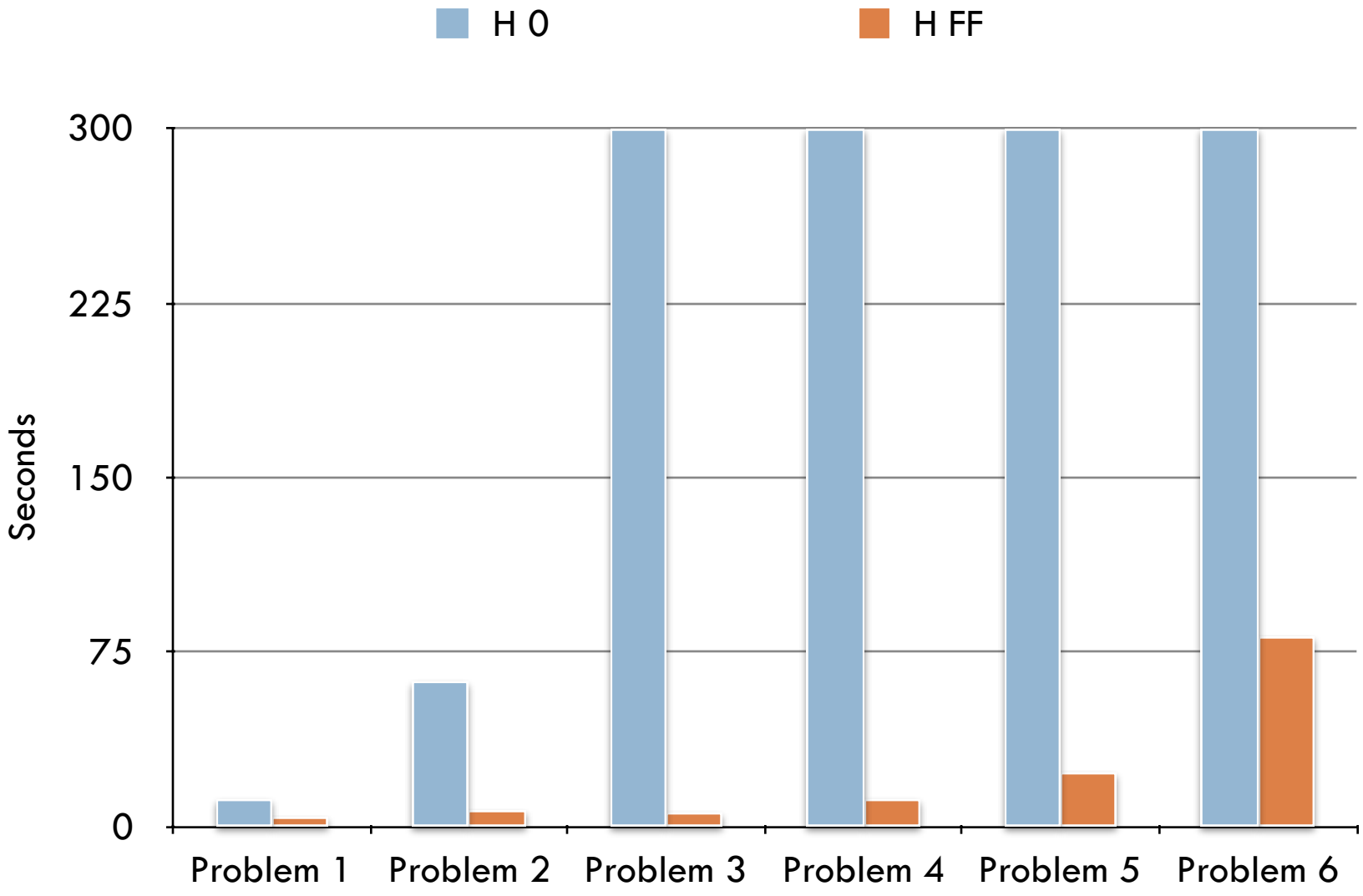
Separate **blue blocks** and **green blocks**
Success rate: **100%**
Runtime: **82 sec**



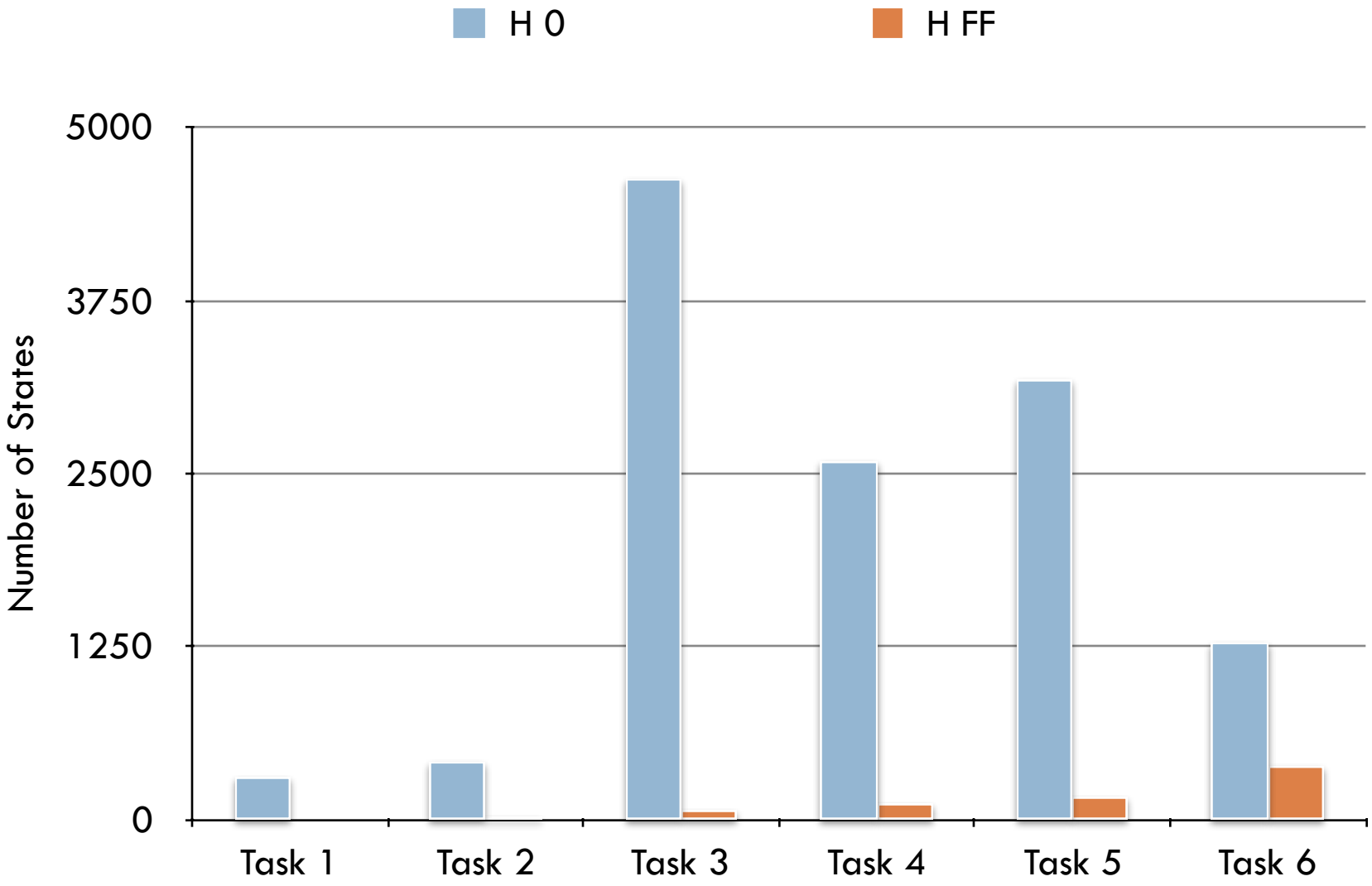
# Success Rate (60 Trials)



# Median Runtime (300s Timeout)



# Median States Visited

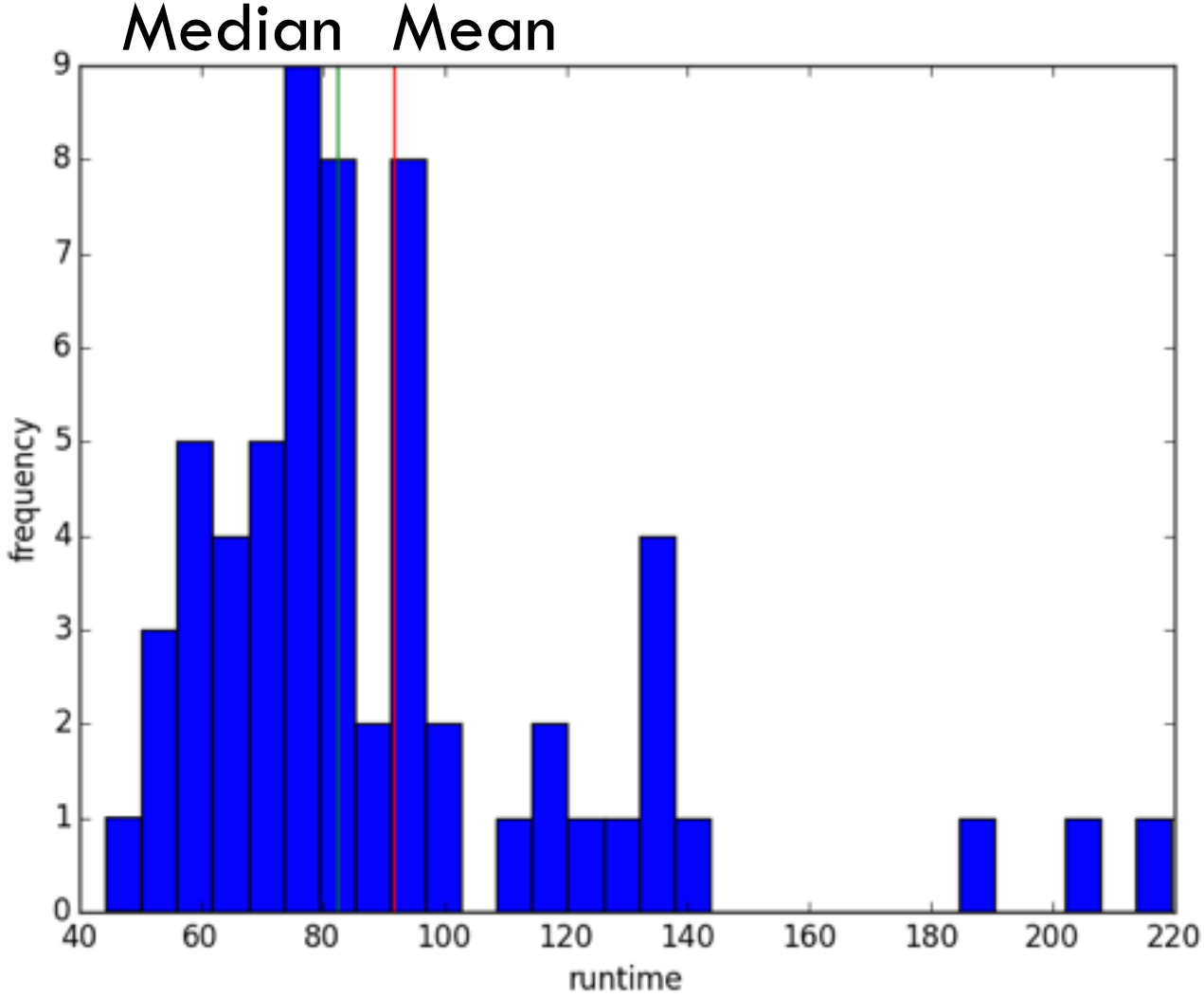


# Full Experimental Results

| P | $H_0$ | | | | $H_{FF}$ | | | |
|---|---|---|---|---|---|---|---|---|
| | % | runtime | length | visited | % | runtime | length | visited |
| 1 | 100 | 12 (7) | 8 (0) | 156 (140) | 100 | 4 (1) | 12 (2) | 6 (2) |
| 2 | 97 | 62 (26) | 16 (0) | 208 (37) | 100 | 7 (1) | 16 (0) | 10 (1) |
| 3 | 62 | 238 (62) | 16 (0) | 2315 (712) | 100 | 6 (1) | 16 (0) | 37 (2) |
| 4 | 0 | 300 (0) | - (-) | 1293 (93) | 97 | 12 (4) | 24 (4) | 56 (24) |
| 5 | 0 | 300 (0) | - (-) | 1591 (381) | 98 | 23 (9) | 24 (4) | 85 (37) |
| 6 | 0 | 300 (0) | - (-) | 637 (40) | 100 | 82 (13) | 72 (4) | 191 (37) |

- 6 problems
- 60 trials per algorithm and problem
- Timeout of 300 seconds
- Median statistics (MAD in parentheses)
- Python implementation uses OpenRAVE

# Problem 6 $H_{FF}$ Runtime Histogram



# Prior Work

- **Manipulation Planning**
  - Lozano-Pérez - explicit configuration space
  - Siméon et al. - manipulation graph
  - Hauser - multi-modal motion planning
  - Barry et al. - multi-modal biRRT
- **Task and Motion Planning**
  - Lagriffoul, Saffiotti, Dornhege & Nebel, Cambon et al., …
  - Srivastava et al. - planner-independent interface
  - Garrett et al. - FFRob
- **Discrete Planning**
  - Bonet & Geffner - HSP
  - Hoffmann & Nebel - FF