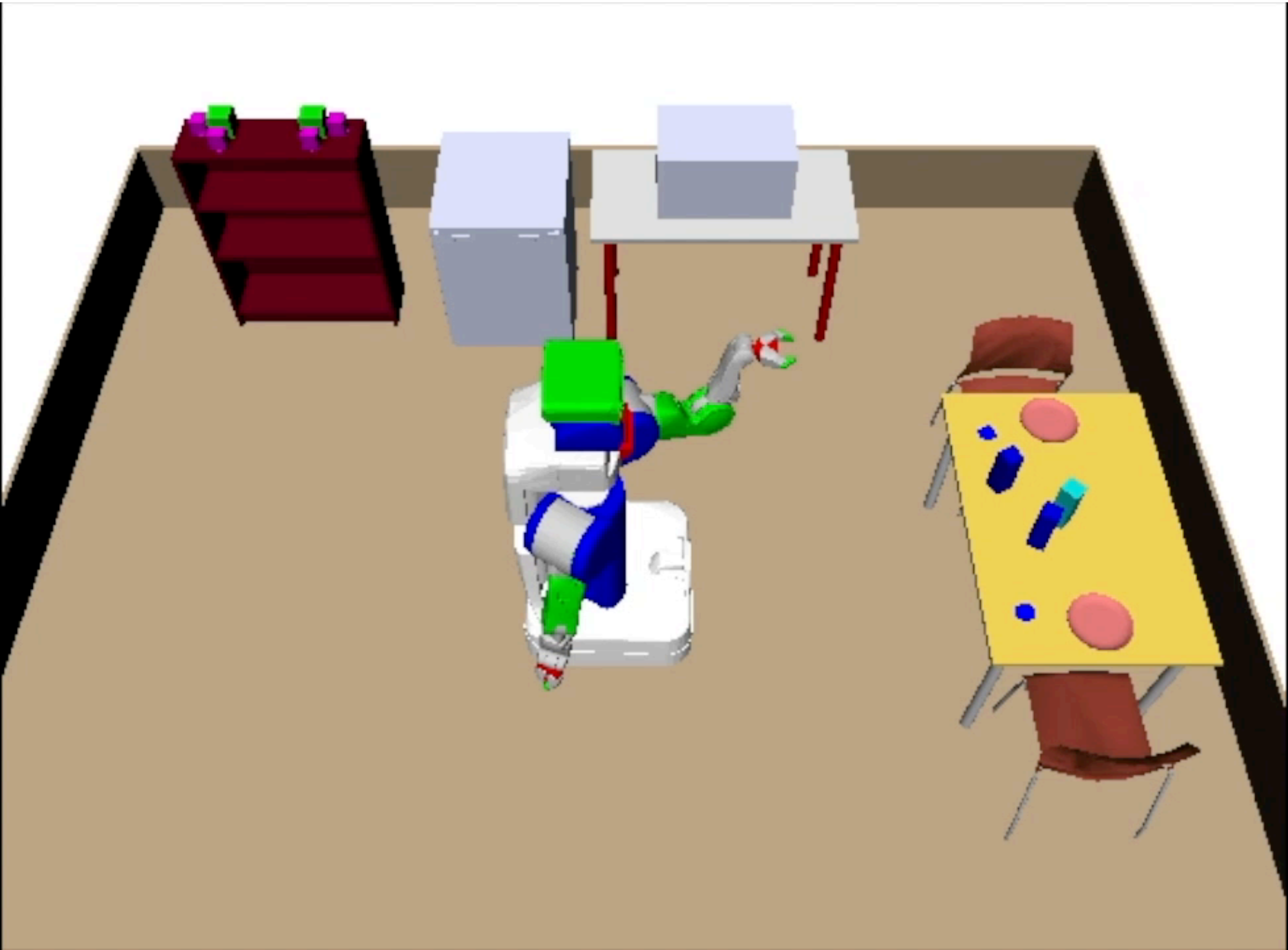
SAMPLING-BASED METHODS FOR FACTORED TASK AND MOTION PLANNING

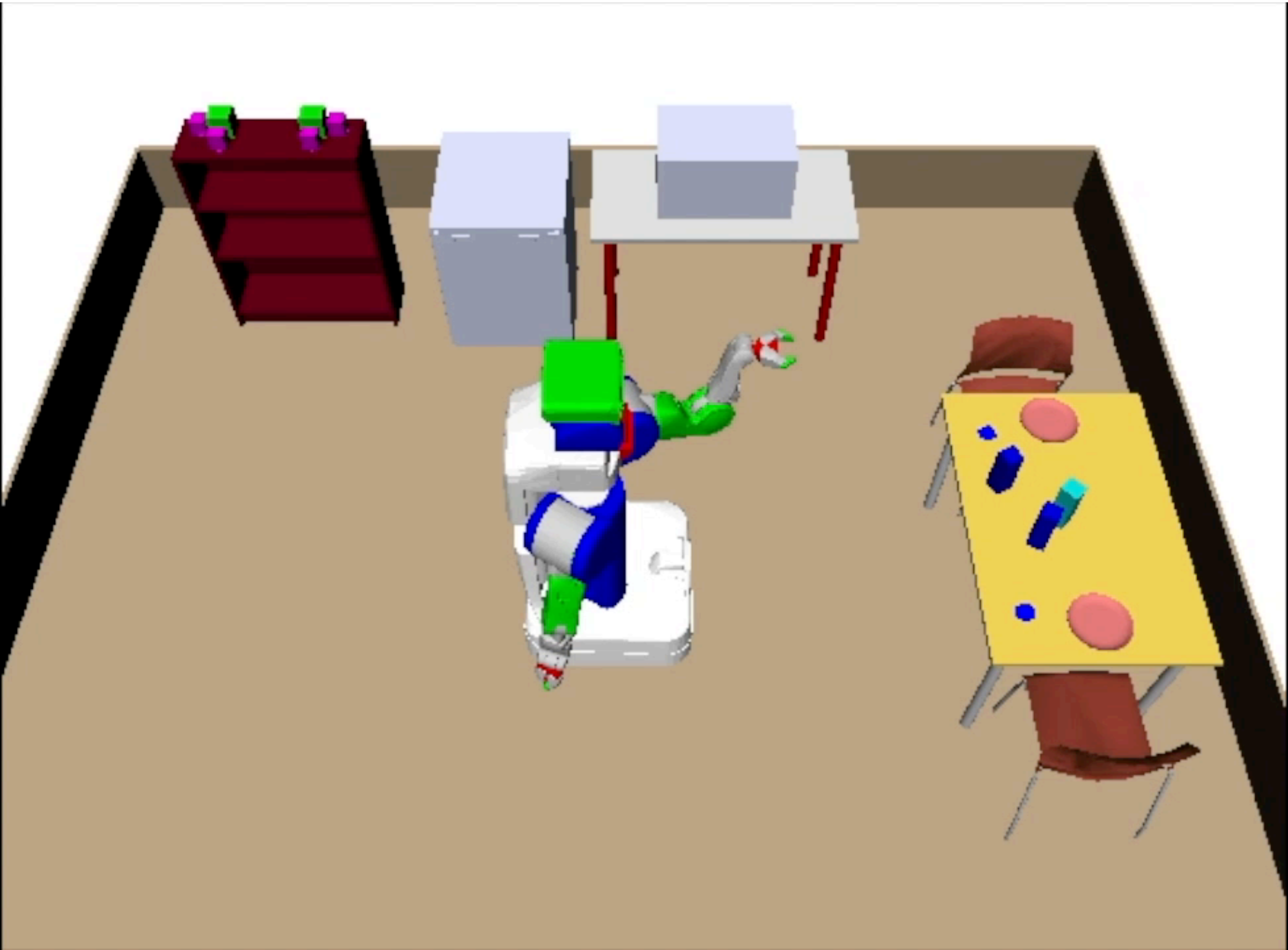
Caelan Garrett, Tomás Lozano-Pérez, and Leslie Kaelbling
MIT EECS Research Qualifying Exam

Task and Motion Planning

- **Application**
  - Fully autonomous robots in human environments
- **One (of many) challenges**
  - Planning in mixed discrete-continuous (hybrid) spaces

- **Task planning (AI planning)**
  - Discrete state/actions
- **Motion planning**
  - Continuous robot movements





Related Work

- **Multi-Modal Motion Planning**
  - *Alami et al., Siméon et al., Hauser and Latombe, (Jennifer) Barry, Vega-Brown and Roy*
  - Inefficient in high-dimensional state-spaces

- **Navigation Among Movable Obstacles (NAMO)**
  - *Stilman and Kuffner, Van Den Berg et al., Krontiris and Bekris*
  - Address a specialized subclass of manipulation planning

- **Task and Motion Planning**
  - Finite domains - *Dornhege et al., Erdem et al., Dantam et al.*
  - *Cambon et al., Kaelbling and Lozano-Pérez, Lagriffoul et al., Srivastava et al., Garrett et al., Toussaint*
  - Generally inflexible to new domains

Contributions

- **Sampling-based planning for a broad class of hybrid problems**
  - (Not just task and motion planning)

- **General-purpose algorithms**
  - Treat domain-specific samplers as blackboxes
  - Usable software that respects this abstraction

- **Efficient algorithms**
  - Leverage factoring while sampling and searching

Outline

1. Factored transition systems

2. Sampling-based planning

3. Algorithms

4. Future directions

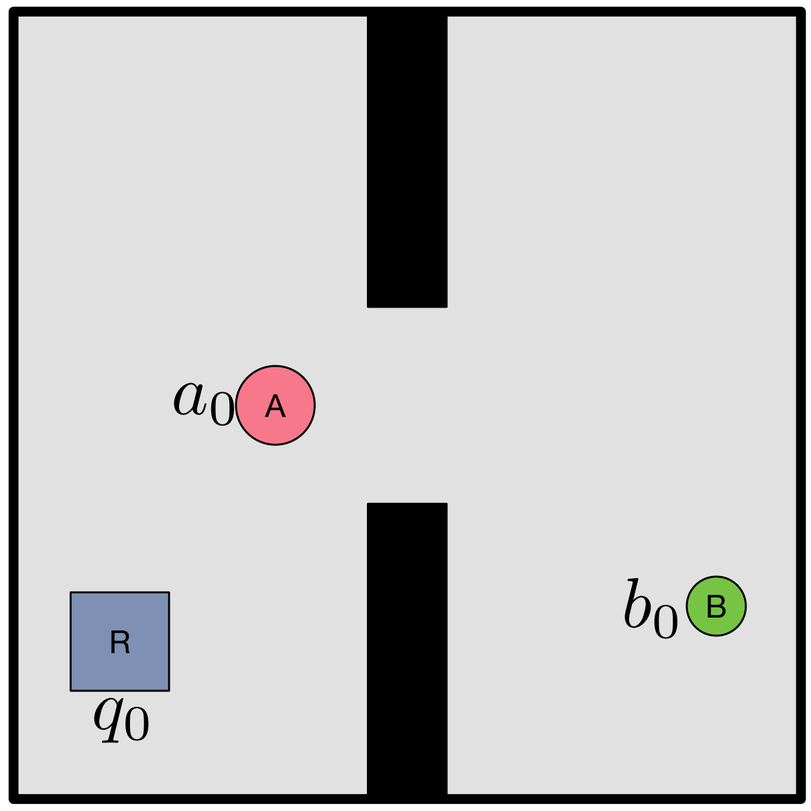
Factored Transition Systems

Problem Class

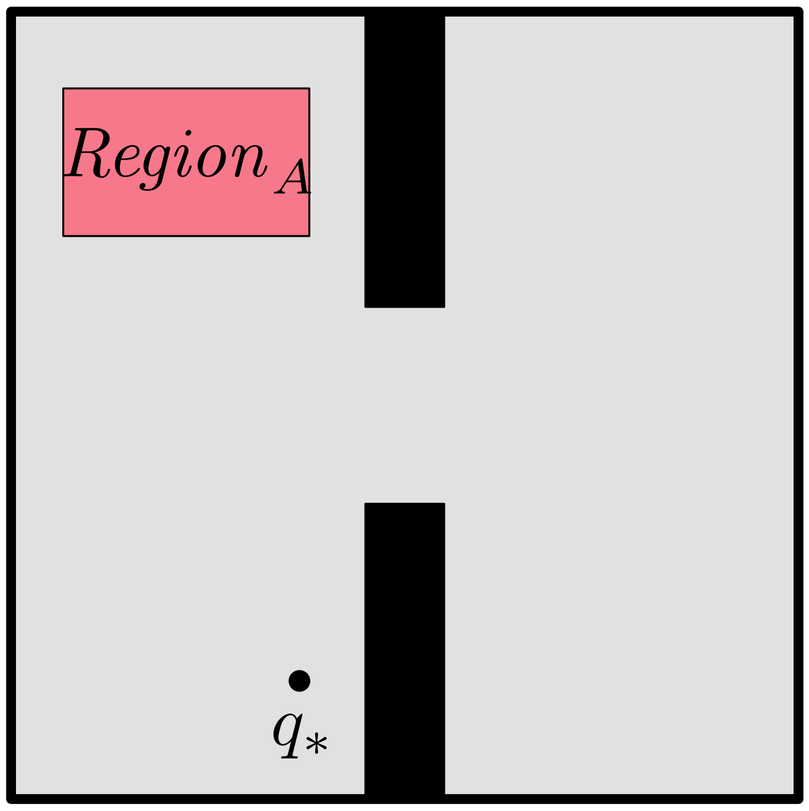
- **Discrete-time**
  - Plans are finite sequences of controls (but with continuous parameterizations)
- **Deterministic**
  - Actions always produce the intended effect
- **Observable**
  - Access to the full world state
- **Hybrid**
  - State & control composed of **mixed discrete-continuous variables**

Example Pick-and-Place Problem

- 1 robot: **R**
- 2 movable objects: **A, B**
- 1 region: **Region$_A$**
- Goal constraints: **robot at q_*, object A in Region$_A$**



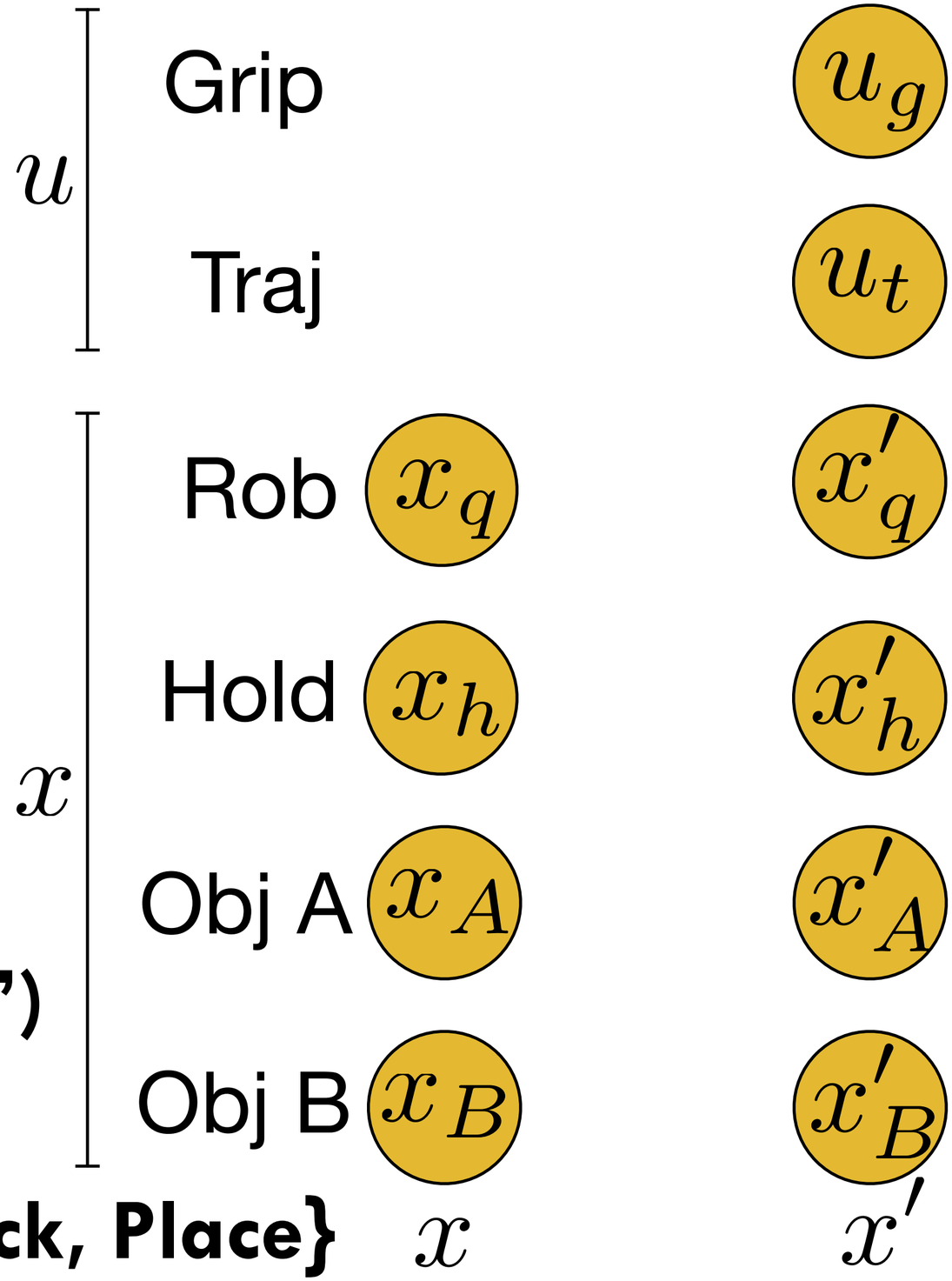
$\mathcal{C}_0 = \bar{x}^0 = (q_0, a_0, b_0, \textbf{None})$



$\mathcal{C}_* = \{Region_A, x_q = q_*\}$

Factored Transition System

- **State variables: x**
  - Robot configuration
  - Holding - A, B, None
  - Object poses
- **Control variables: u**
  - Trajectories, grasping
- **Transition Relation: T(x, u, x')**
  - Union of transition clauses
  - **Clauses: {Move, MoveH, Pick, Place}**



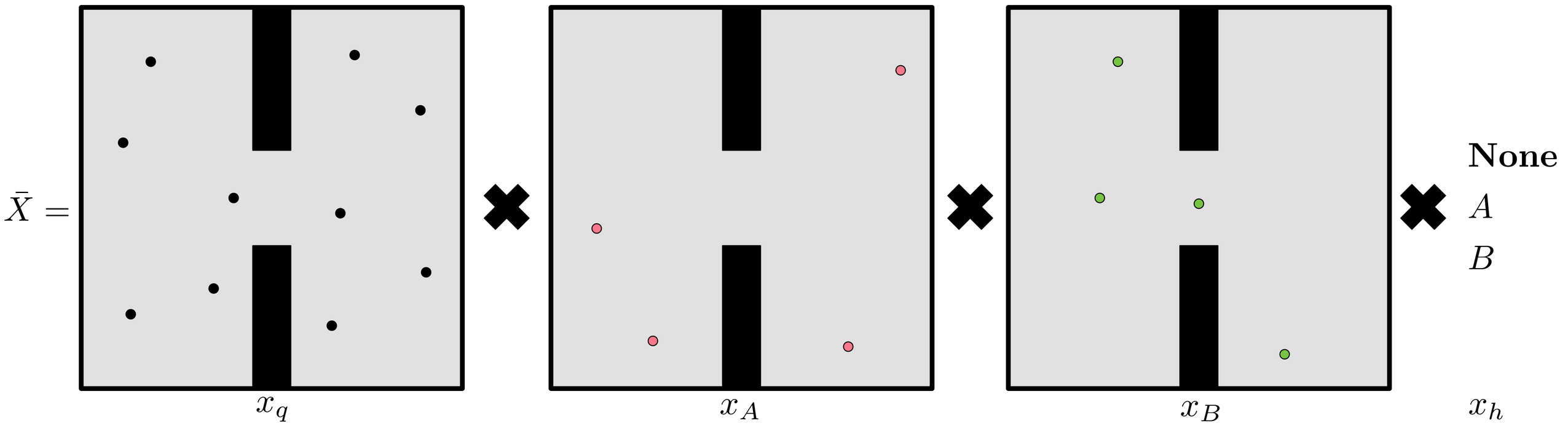
Benefits of Factoring

- Extremely **high-dimensional** - (e.g. 100 < DOFs)
  - Each movable object introduces new DOFs
- Discretized state-space **grows combinatorially**
  - Existing motion planning methods ineffective

- We develop algorithms that exploit **factoring**
  - **Sample subsets** of state/control variables at a time that satisfy particular constraints
  - Discrete search using state-of-the-art **AI planning algorithms**

Factored States

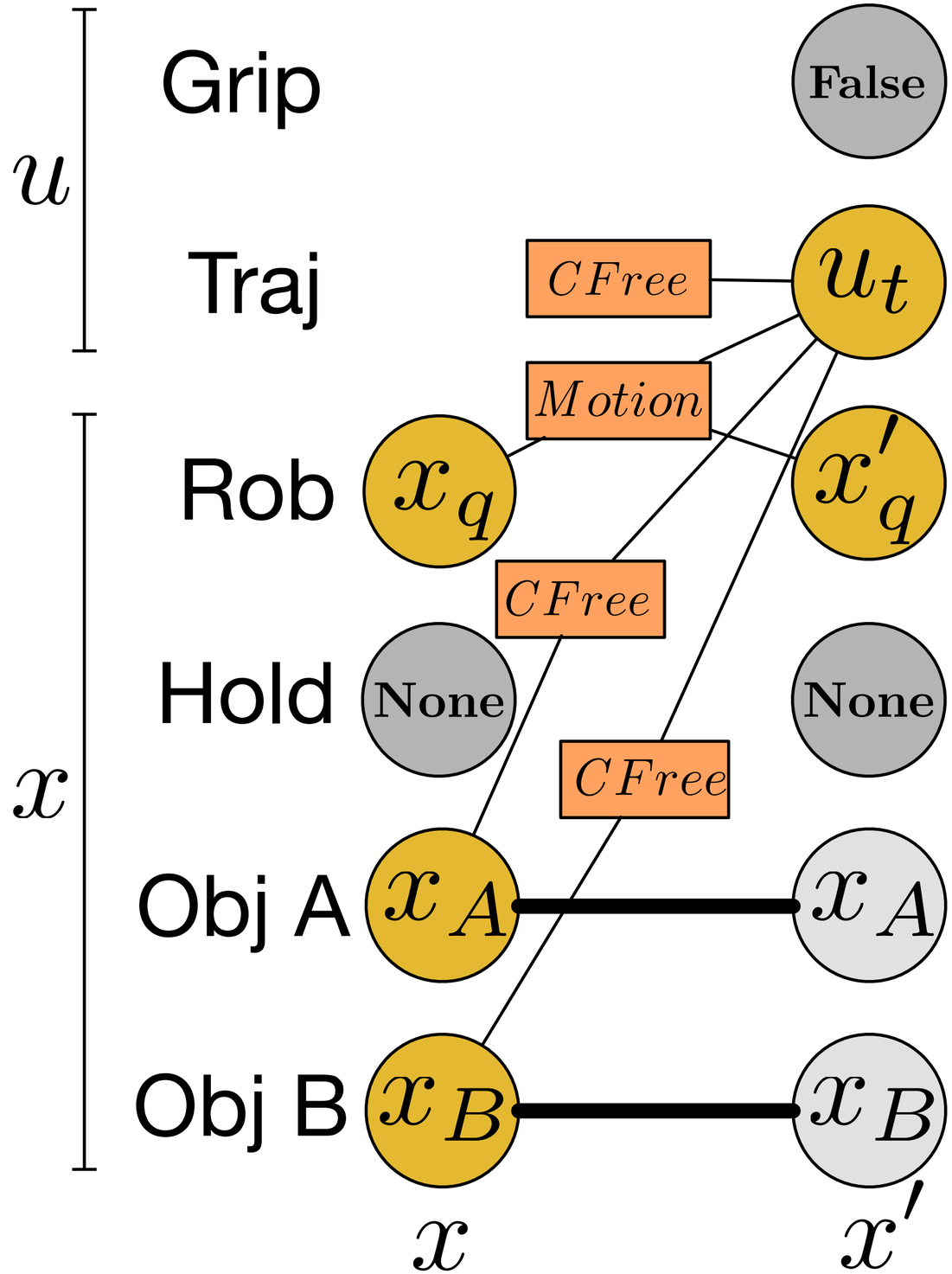
- **Sample efficiency**: a set of values for each variable can induce a large set of states
- 10 samples for R_q, 4 samples for A, and 4 samples for B induce a large state-space (**160 states**)



Factored Move Transition

- Robot moves while hand is empty

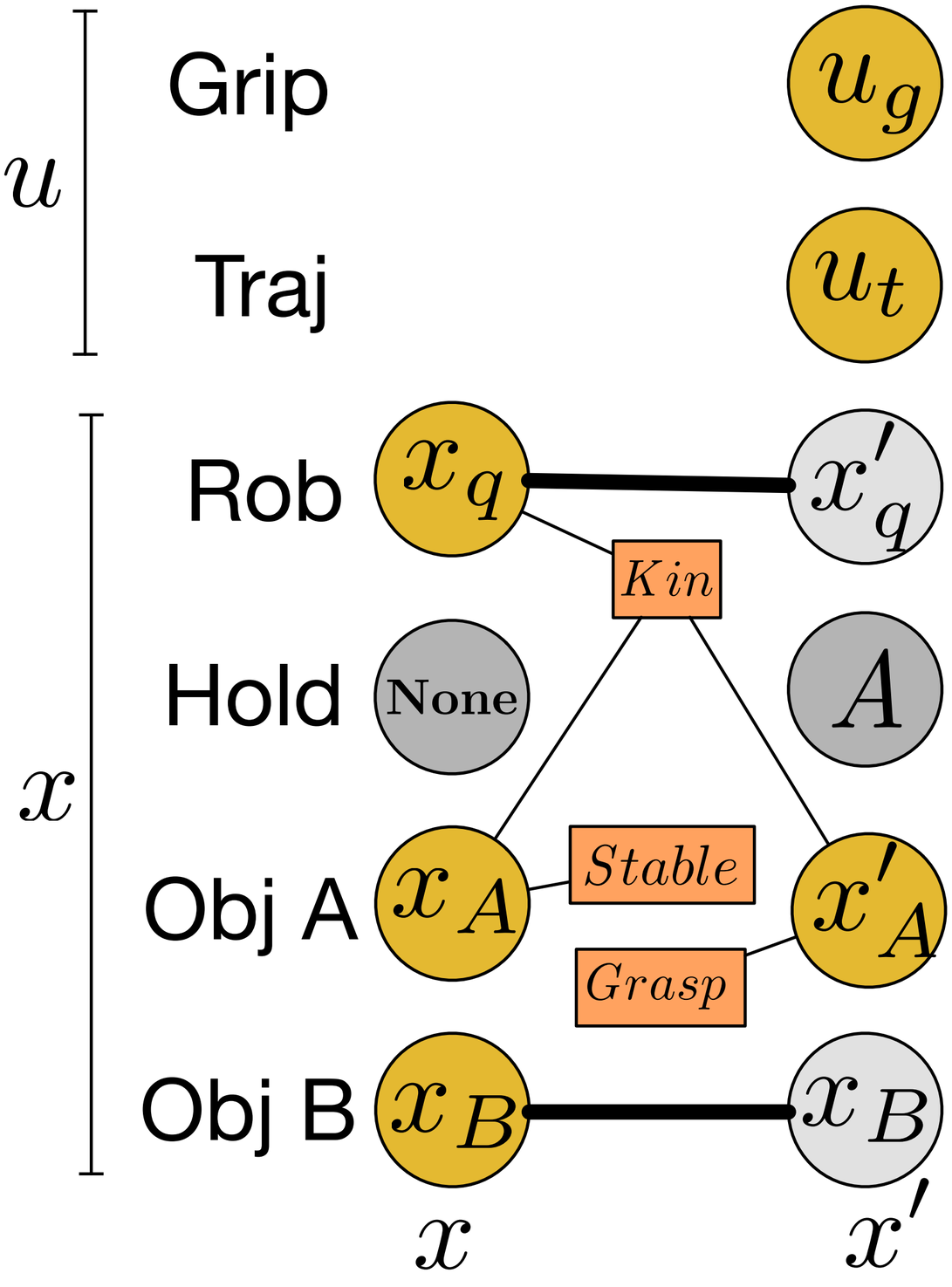
- **Factoring**
  - Poses of objects A & B **unchanged**
  - Collision-free (CFree) constraints mention objects **separately**



Factored Pick Transition

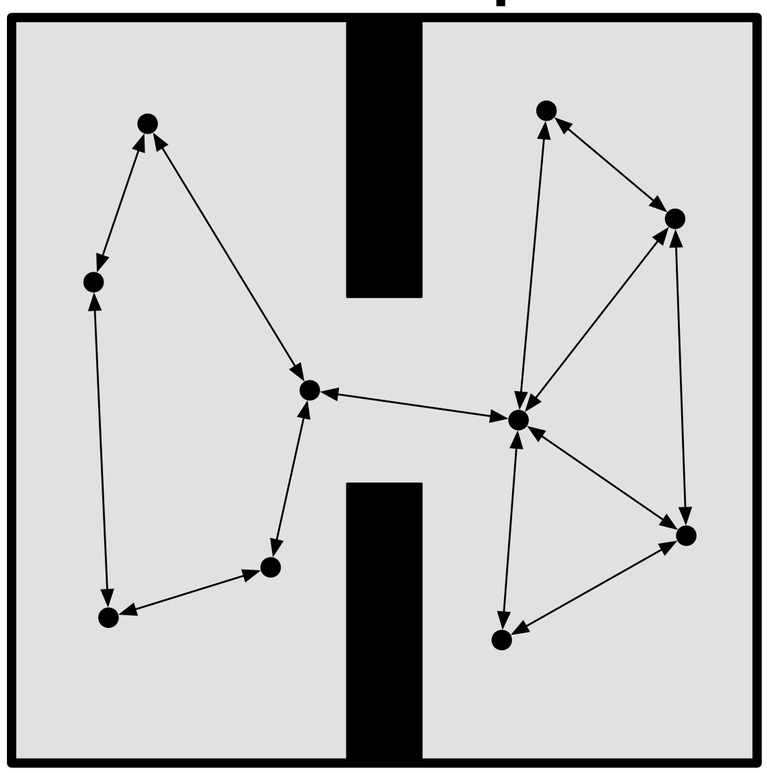
- Robot instantly grasps an object

- **Factoring**
  - Robot conf & pose of object B **unchanged**
  - Kinematics (Kin) constraint involves **just object A & robot**

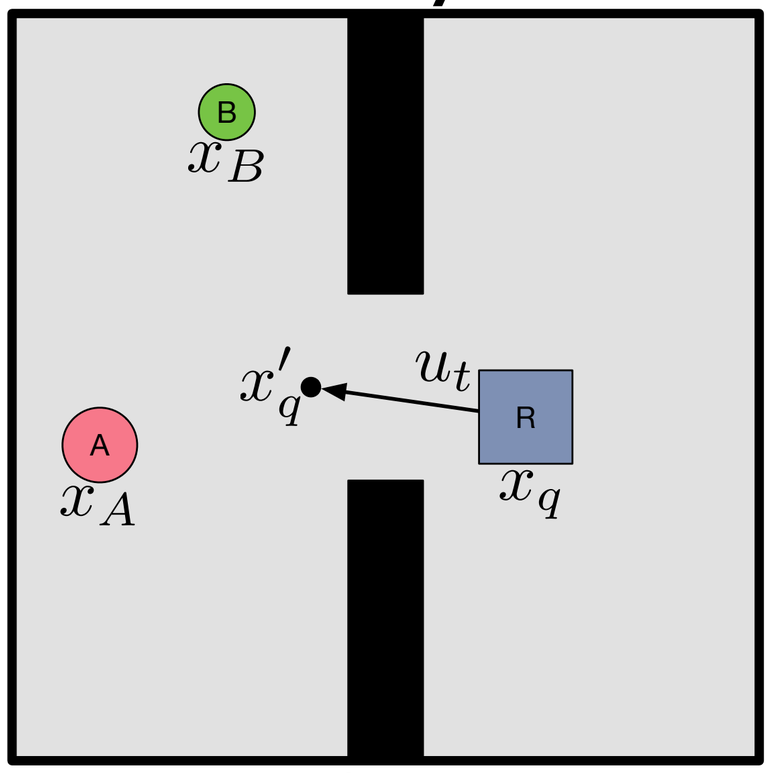


Factored Controls

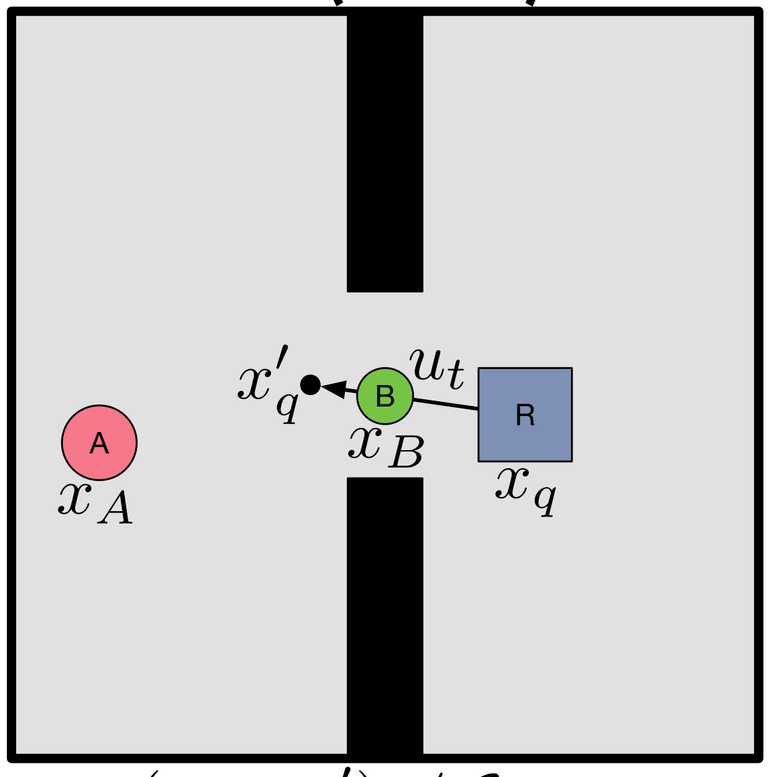
- The same control (u_t) can be applied in many different transitions
- Allows reasoning about states in which the control can be applied
- 26 trajectories admit many Move transitions (208)



u_t



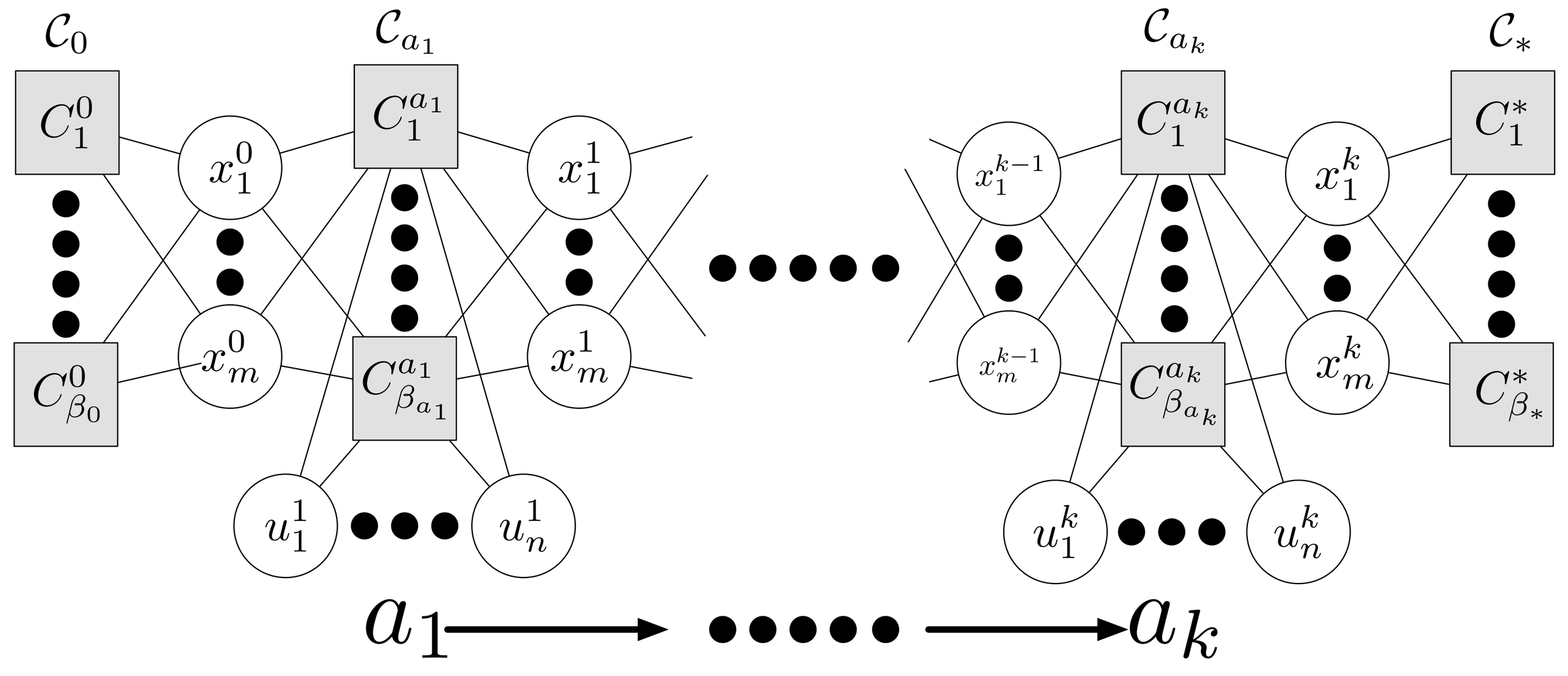
$(\bar{x}, \bar{u}, \bar{x}') \in \mathcal{C}_{Move}$



$(\bar{x}, \bar{u}, \bar{x}') \notin \mathcal{C}_{Move}$

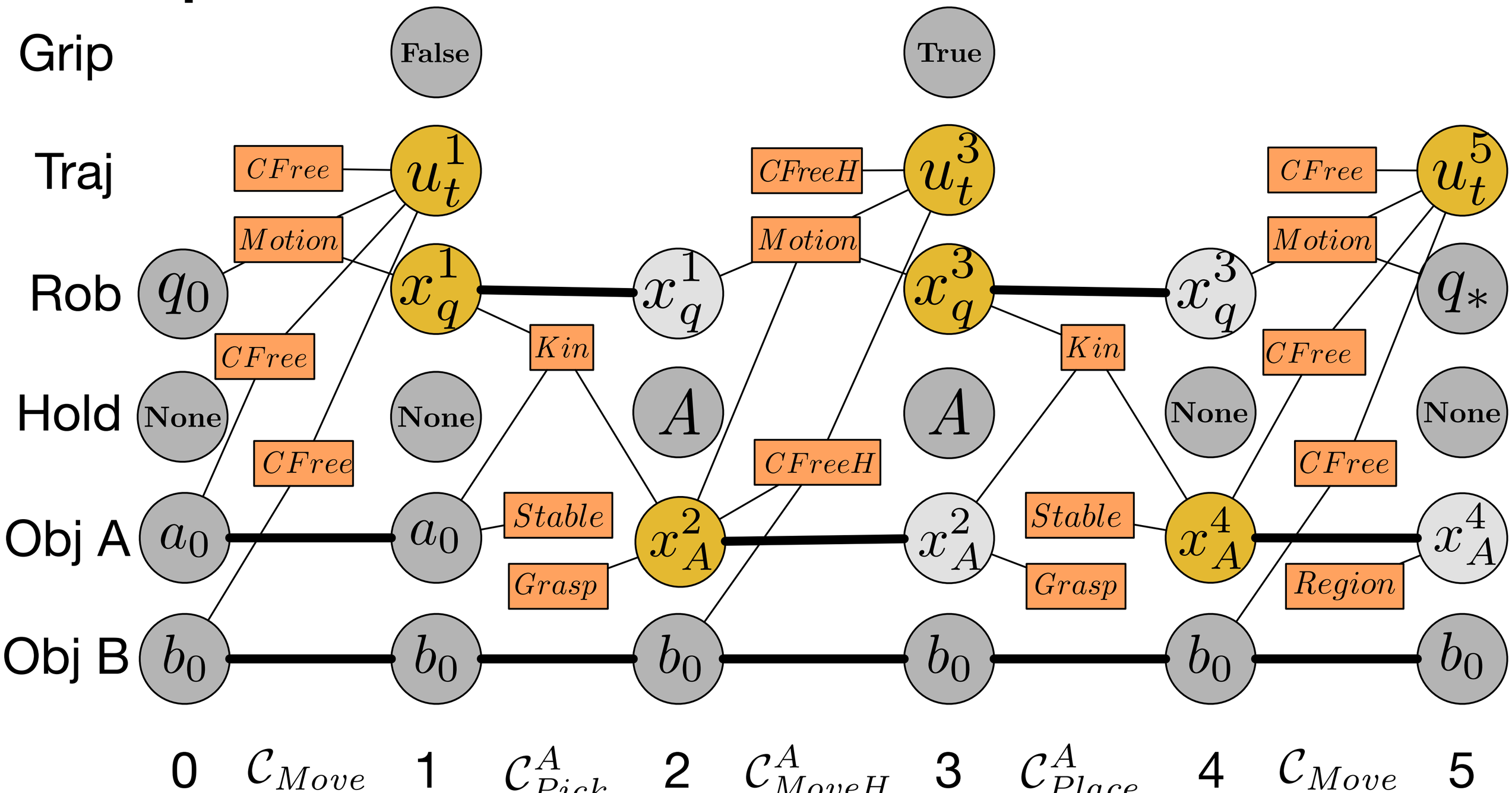
Constraint Network

- **Plan skeleton**: sequence of transition clauses
- **Constraint network**: bipartite graph from variables to constraints



Only 7 out of 29 variables free

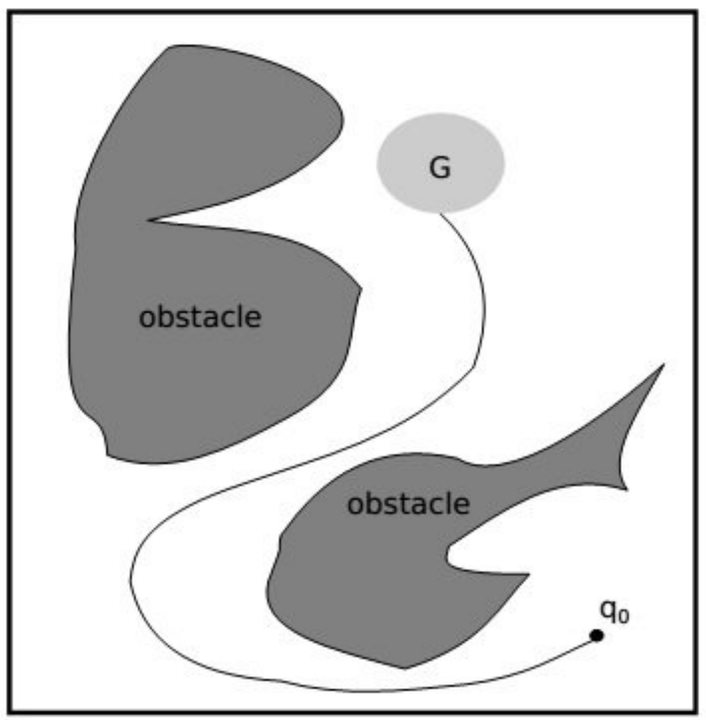
- **Sparse interactions** between variables

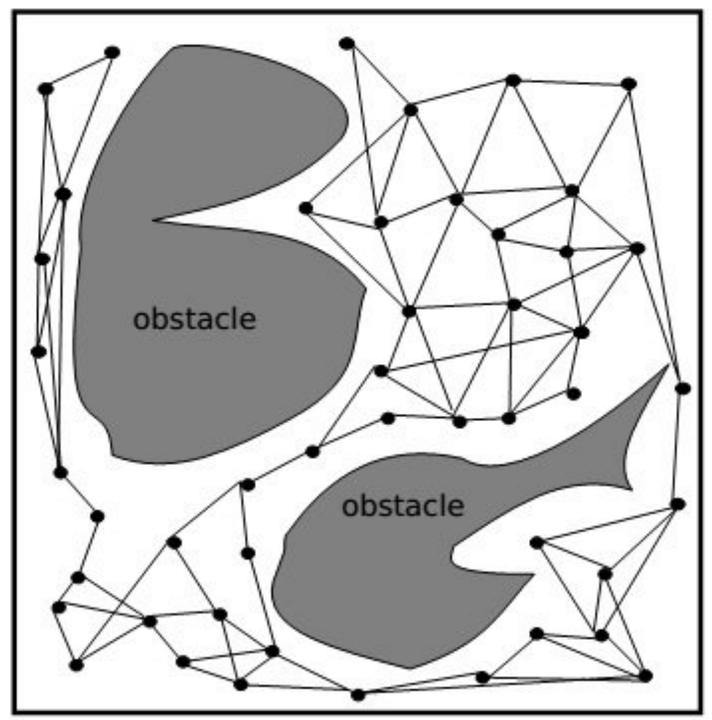


Sampling-Based Planning

Sampling-Based Motion Planning

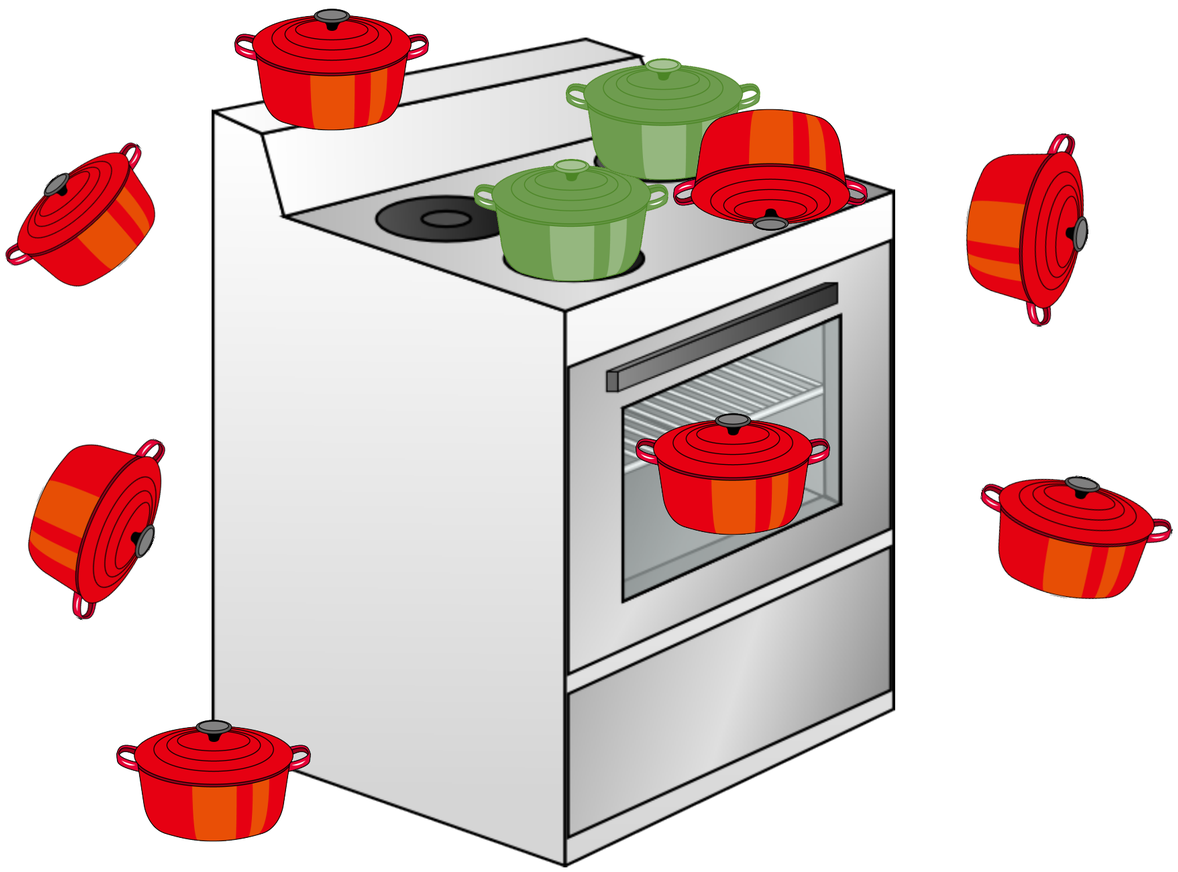
- **Primitive procedures**
  - Sample configurations (state variable)
  - Connect configurations (control variable)
  - Test collision
- **Probabilistic Roadmap (PRM)**
  - Sample values
  - Discrete search

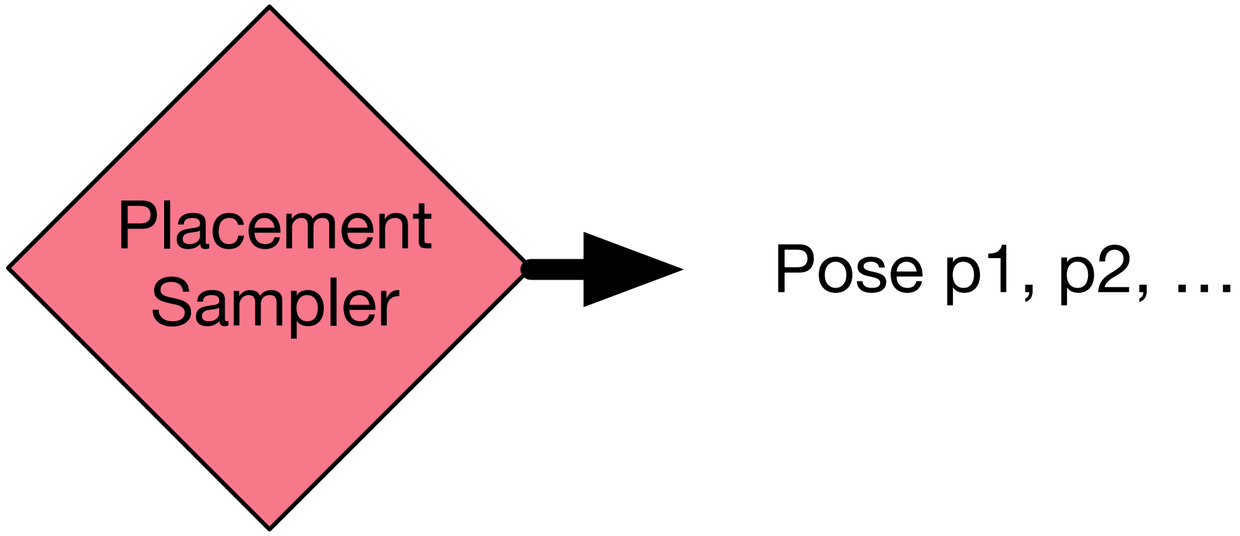




Dimensionality Reducing Constraints

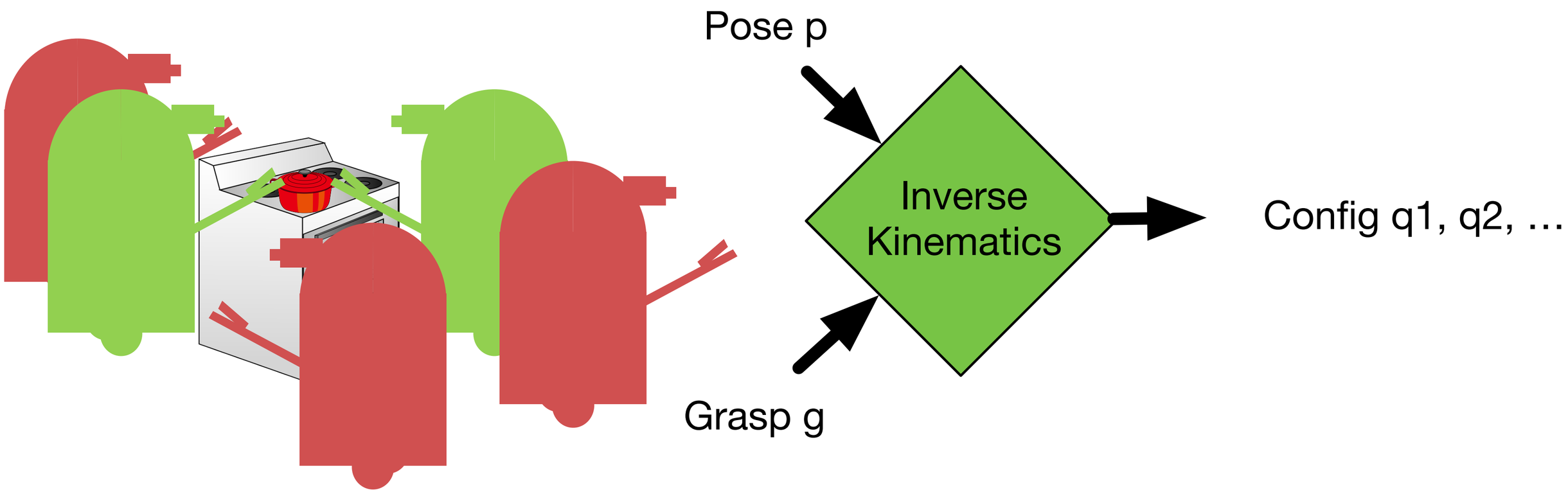
- **Low dimensional** pose stability constraint (*Stable*)
- Directly **sample the constraint**



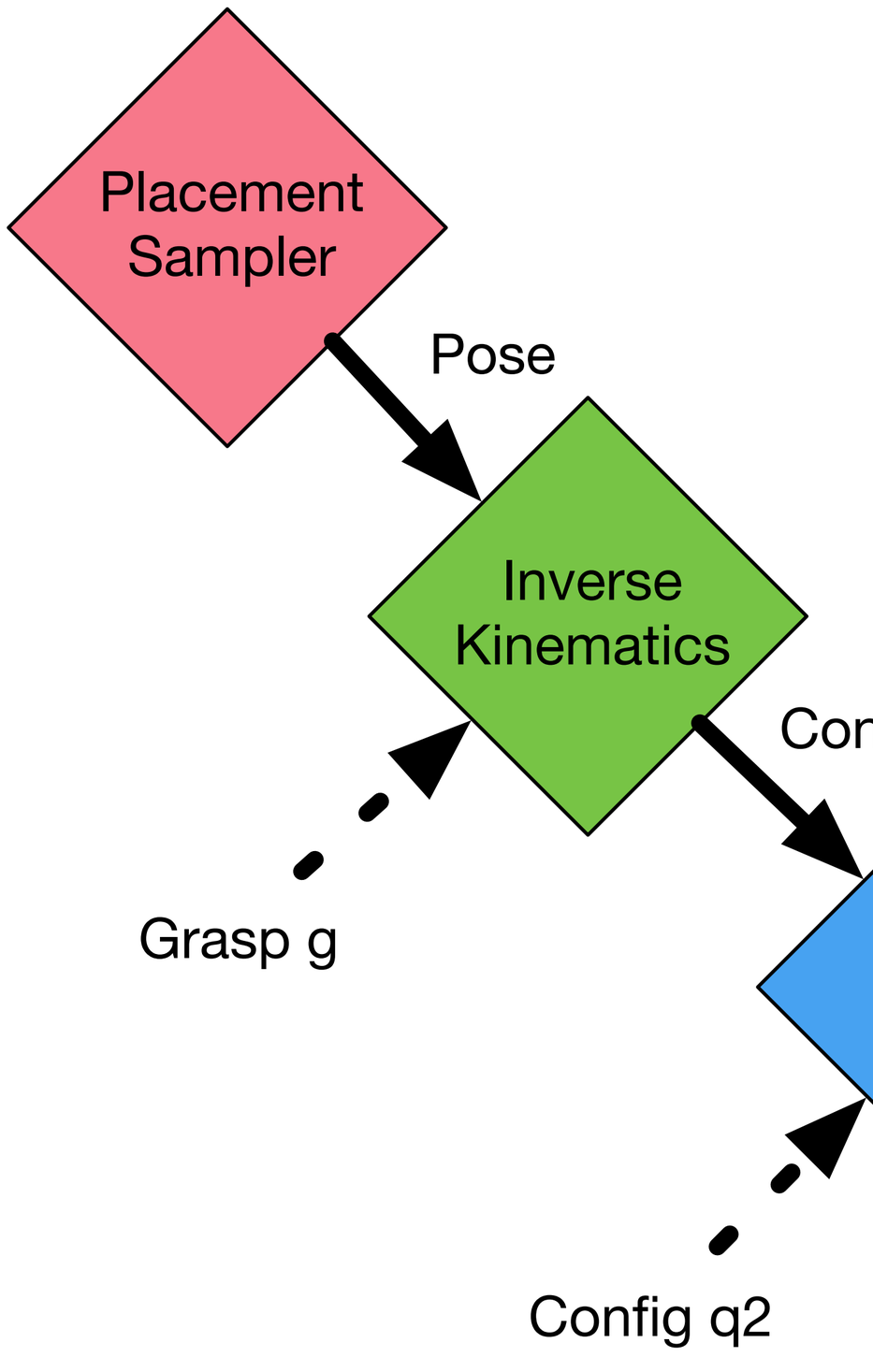


Intersection of Constraints

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



Composing Conditional Samplers



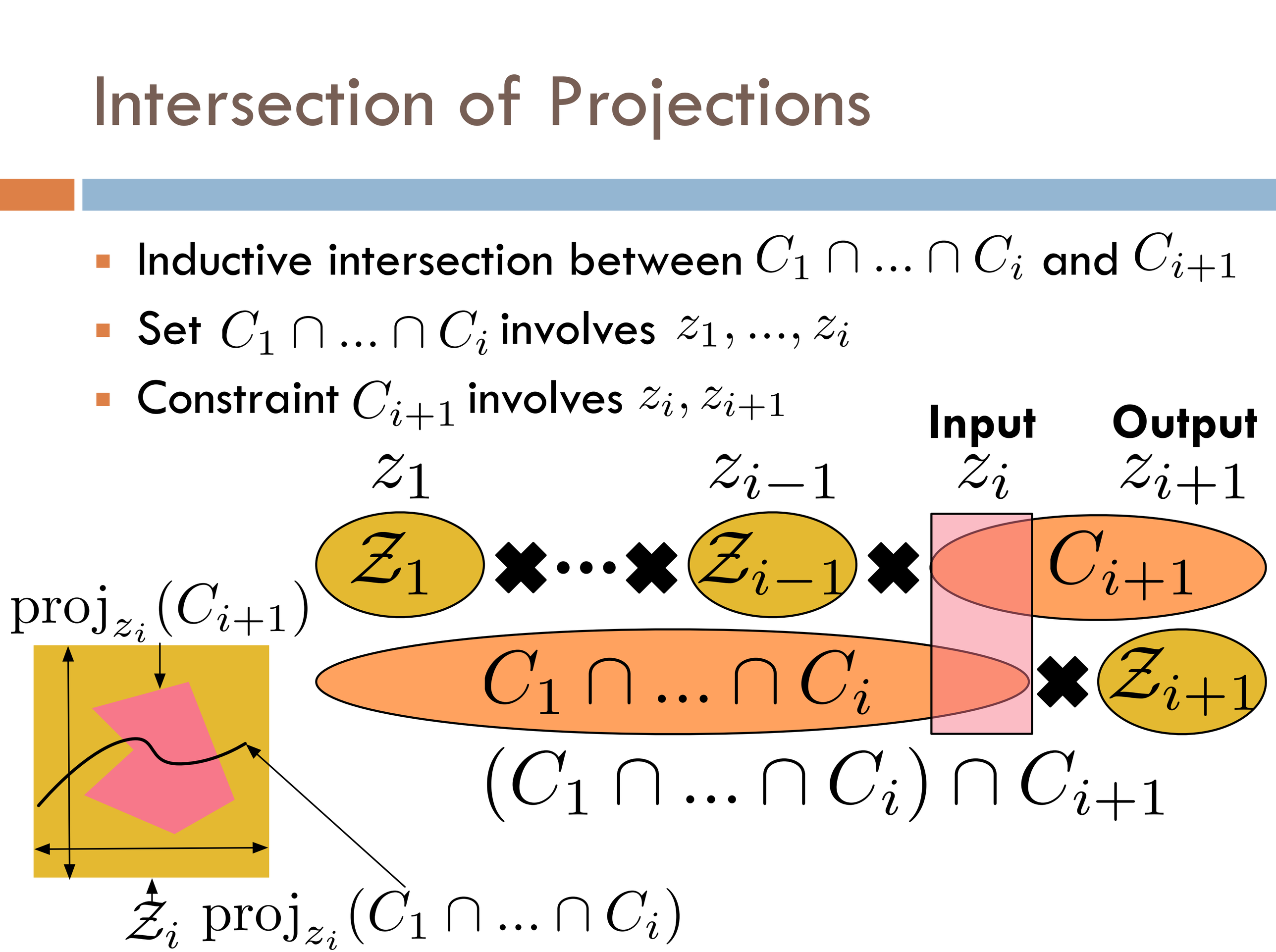
- **Outputs** of one conditional sampler are the **inputs** to another
- **Directed acyclic graph** of conditional samplers

Lower-dimensional Solution Space

- **Sample at the intersection** of several dimensionality-reducing constraints
- When can the intersection be derived using each constraint individually?
- **Conditional constraint** - partition of a constraint into input and output variables
- **Theorem:** *intersection is a submanifold if there exists a conditioning and ordering of conditional constraints:*
  1. Each variable is an **output once**
  2. A variable is an **output before it is an input**
  3. For each constraint, **the projection onto its input variables** has full dimensionality*

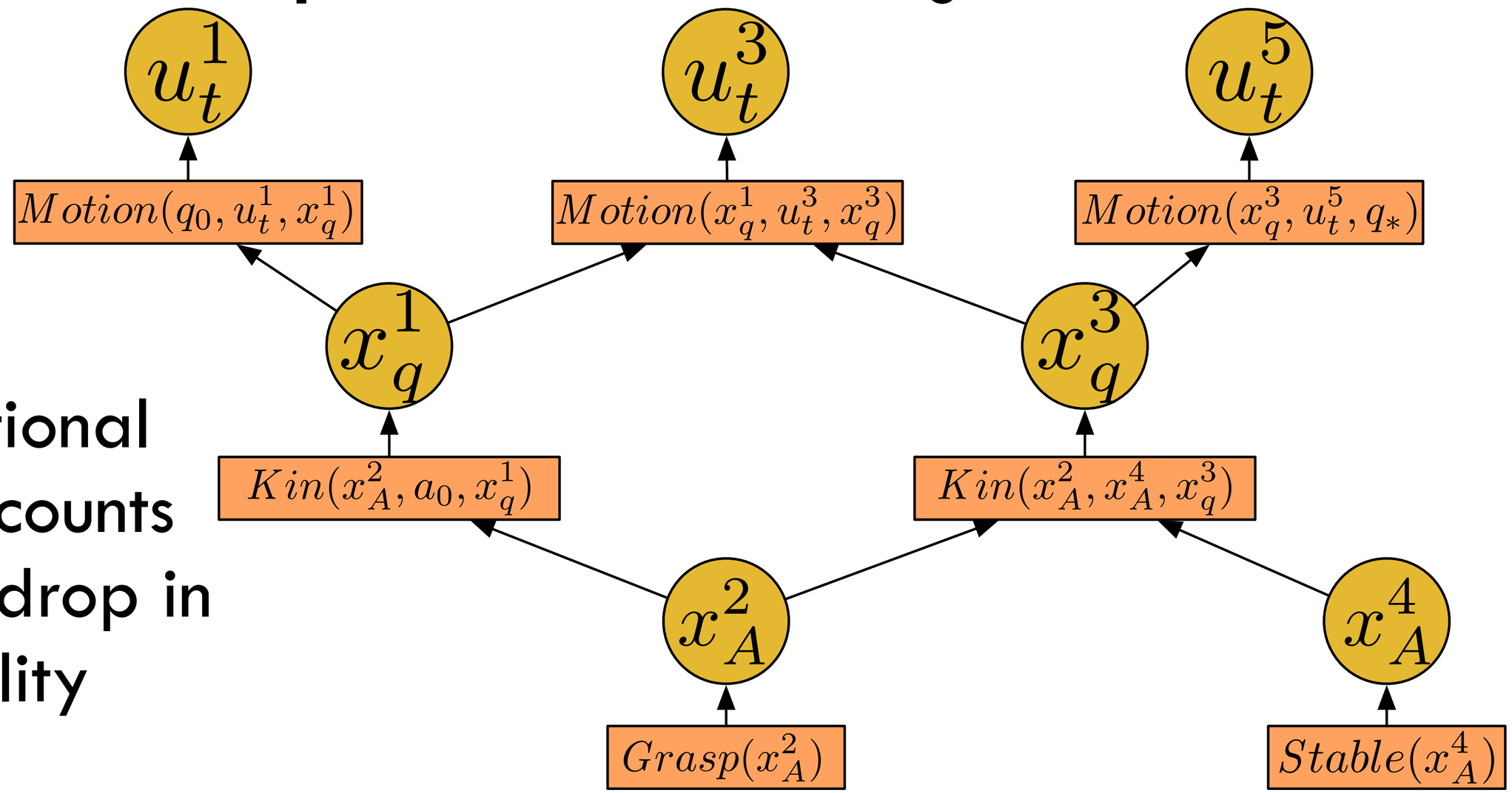
Intersection of Projections

- Inductive intersection between $C_1 \cap \ldots \cap C_i$ and C_{i+1}
- Set $C_1 \cap \ldots \cap C_i$ involves $z_1, \ldots, z_i$
- Constraint C_{i+1} involves z_i, z_{i+1}



Graphical Interpretation

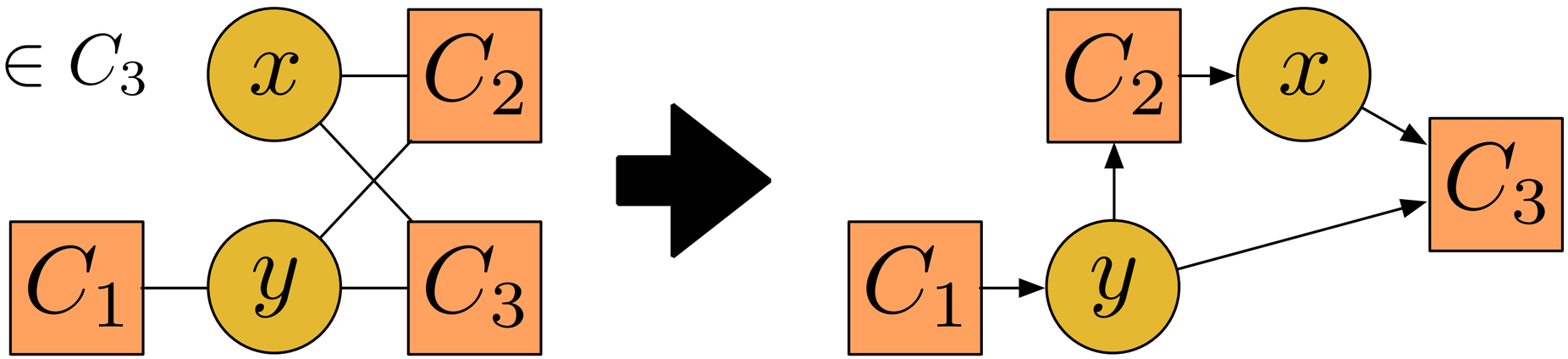
1. Each variable "sampled" exactly **once**
2. Sampler inputs must be chosen **before** outputs
3. Set of **valid input values** is non-degenerate

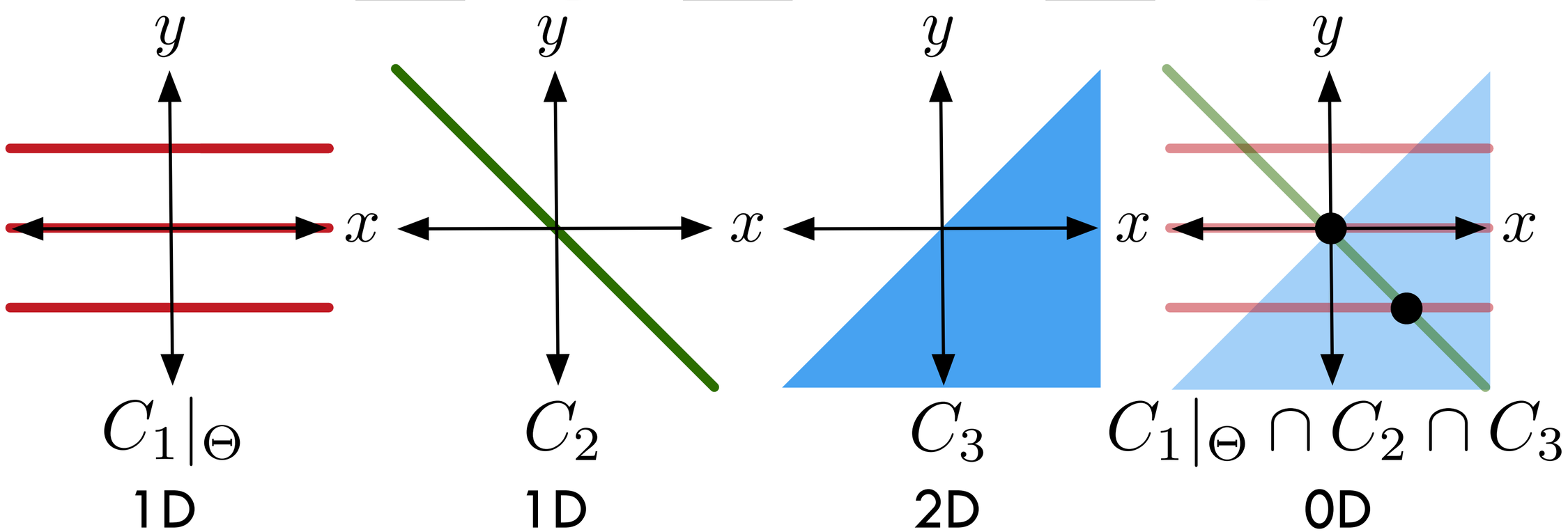


Each conditional sampler accounts for its own drop in dimensionality

2 Variables, 3 Constraints

1. Sample $y \sim C_1$
2. Conditionally sample $x \sim C_2(y)$
3. Test $(x, y) \in C_3$





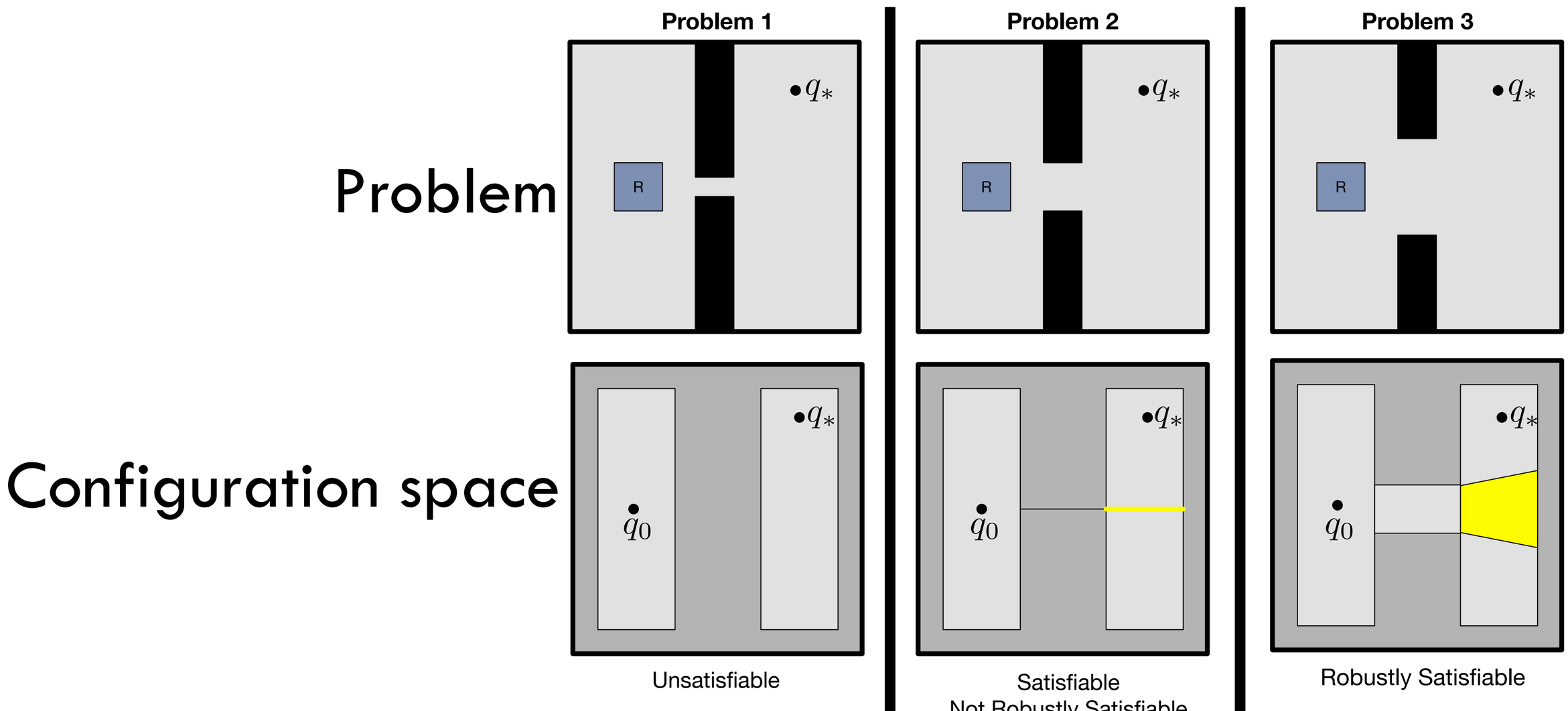
Robust Feasibility

- Sampling-based methods typically **cannot identify infeasibility**
- Also ineffective for feasible problems with a **degenerate set of solutions**

- Investigate completeness for **robustly feasible** problems
  - Set of solutions is an **open set in solution-space**

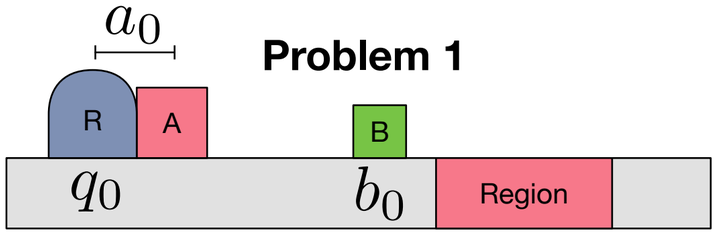
Motion Planning

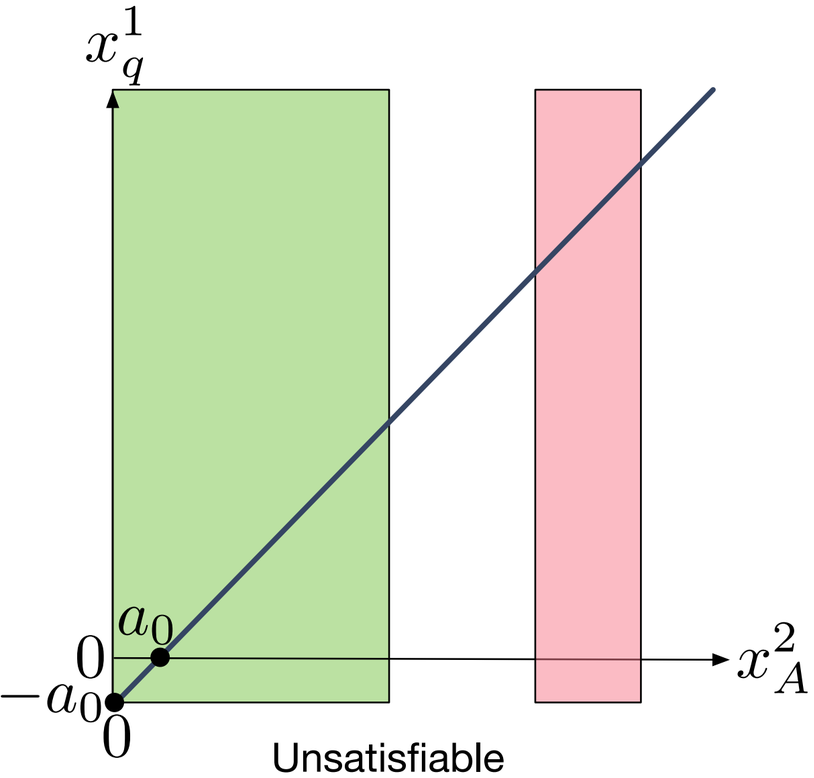
- Robust feasibility ≈ **positive clearance**, positive ε-good, expansive
- Motion skeleton with **1 free waypoint**



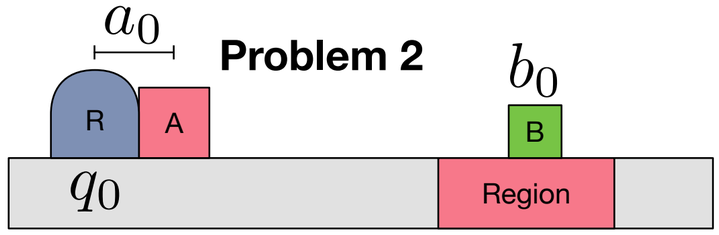
Task and Motion Planning

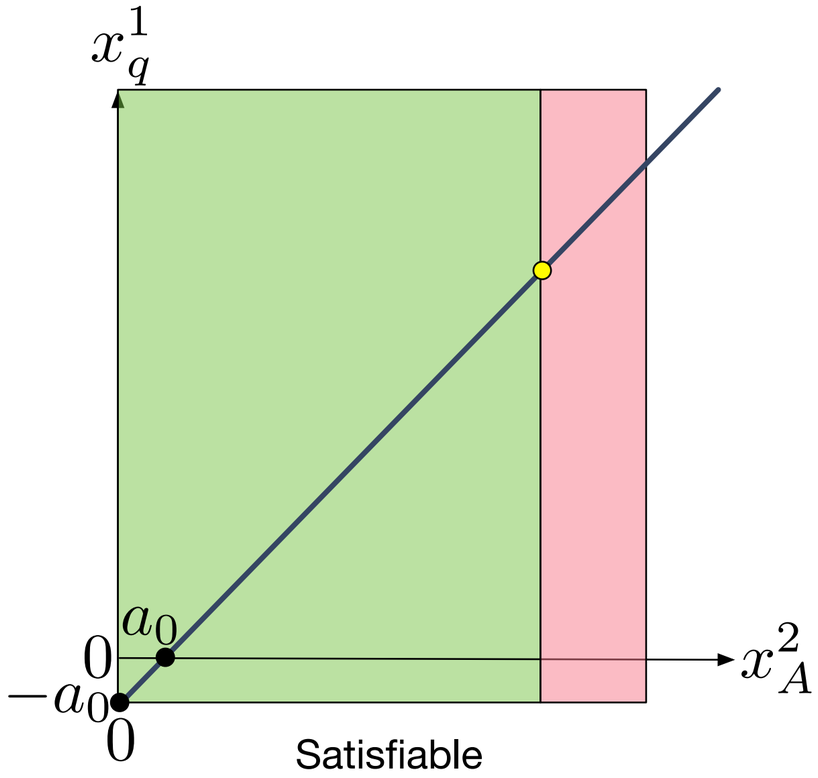
- **Plan skeleton:** $(MoveH^A, Place^A)$**, Free parameters:** x_A^2, x_q^1
- **Constraints:** $\{Motion(q_0, u_t^1, x_q^1), CFreeH(u_t^1, a_0), CFreeH(u_t^1, a_0, b_0),$
$Grasp(a_0), Stable(x_A^2), Kin(a_0, x_A^2, x_q^1), Region(x_A^2)\}$



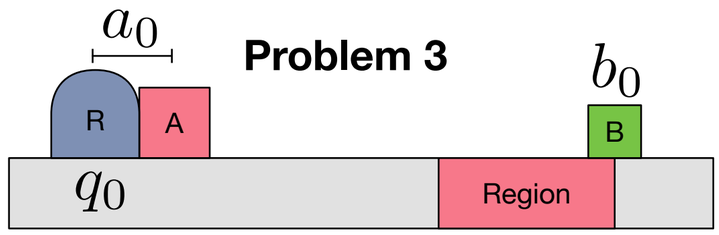


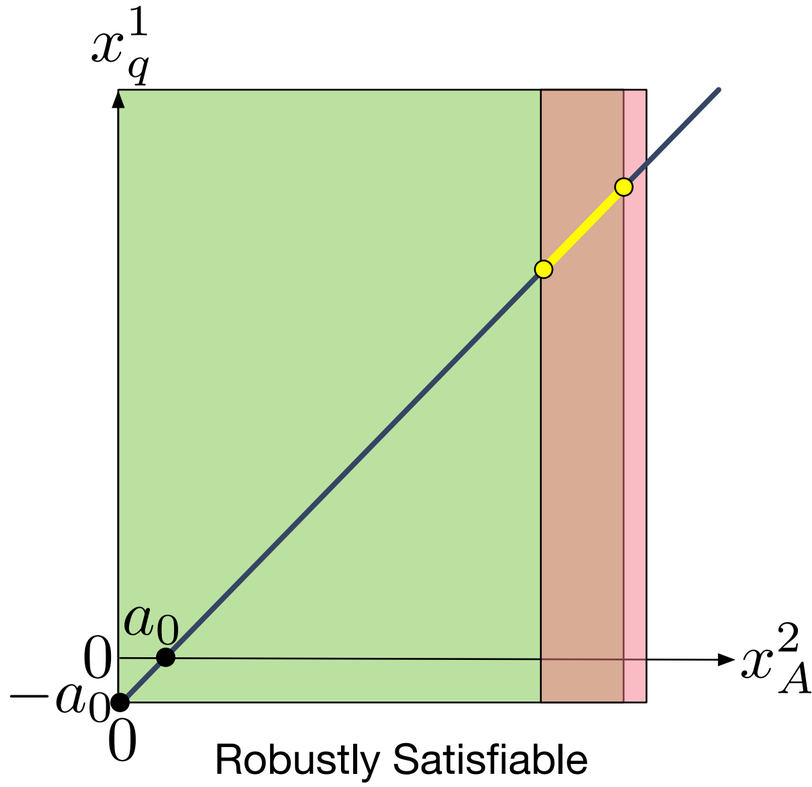
Unsatisfiable





Satisfiable
Not Robustly Satisfiable





Robustly Satisfiable

Planning Algorithms

Domain-Independent Algorithms

- **Meta-parameter** - set of conditional samplers
  - Samplers treated as blackboxes
  - Complete with respect to conditional samplers
  - **Probabilistically complete** given *sufficient* samplers

- Algorithms still must:
  - Search through possible **plan skeletons**
  - **Compose and order** condition samplers
  - Perform **rejection sampling** over the solution-space

Introduce 2 New Algorithms

- Each algorithm repeats
  1. **Sample** values for discretization
  2. **Search** discretized problem for plan

- Search implemented using blackbox algorithms
  - Breadth-First Search (BFS)
  - **Off-the-shelf AI planner** (FastDownard)
    - Compile to AI planning language (SAS+)
    - Exploits factoring in its search heuristics

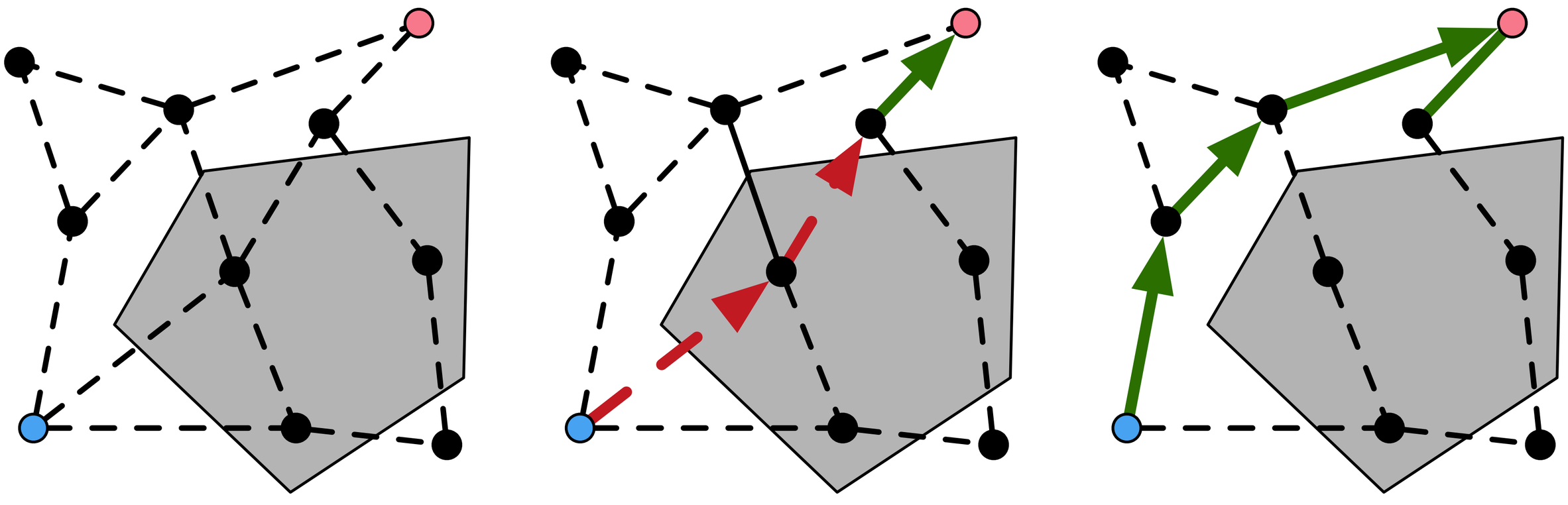
Incremental Algorithm

- **Incremental** ≈ probabilistic roadmap (PRM)
  - Repeat:
    1. **Compose and sample** conditional samplers
    2. **Search** discretized problem

- **Drawback** - produces many unnecessary samples
- Example: [Placement(A)→pA1, Grasp(A)→gA1, Placement(B)→pB1, Grasp(B)→gB1, IK(A, pA0, gA1)→q1, IK(B, pB0, gB1)→q2, IK(A, pA1, gA1)→q3, IK(B, pB1, gB1)→q4, Motion(q0, q1)→t1, Motion(q0, q2)→t2, Motion(q0, q3)→t3, Motion(q0, q4)→t4, Motion(q1, q0)→t5, Motion(q1, q2)→t6, Motion(q1, q3)→t7, …]

Focused Algorithm

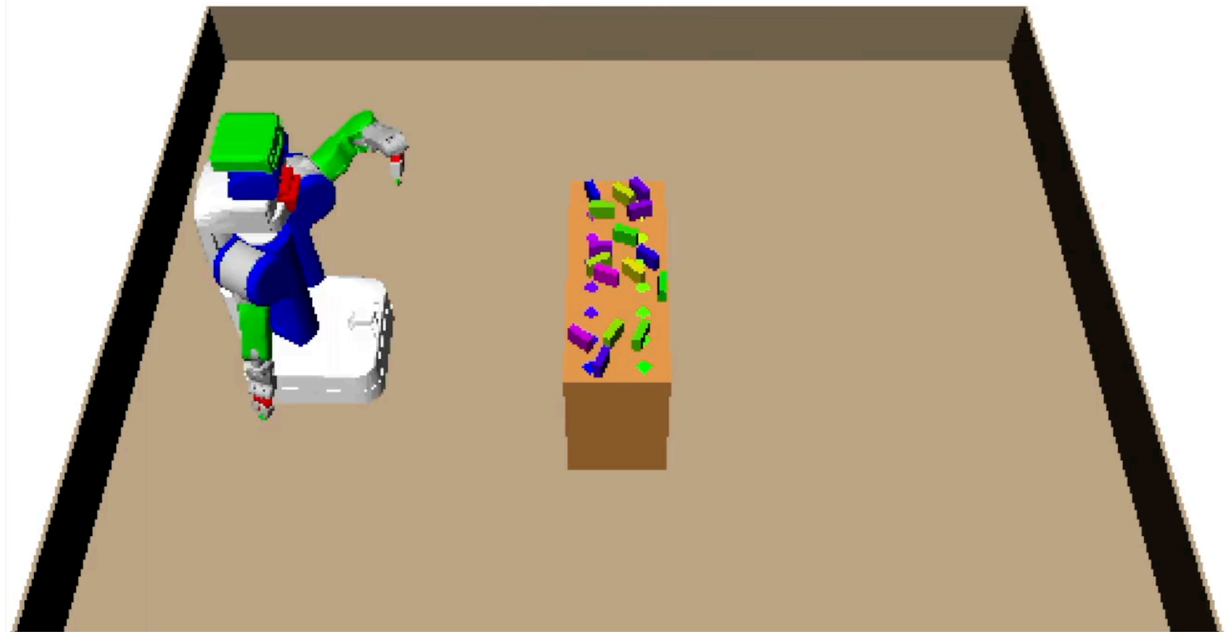
- **Focused** ≈ lazy PRM
  - Repeat:
    1. **Search** with **real & lazy** samples
    2. **Sample** values for lazy samples on **found plan**
- **Lazy samples** ≈ lazy collision checking

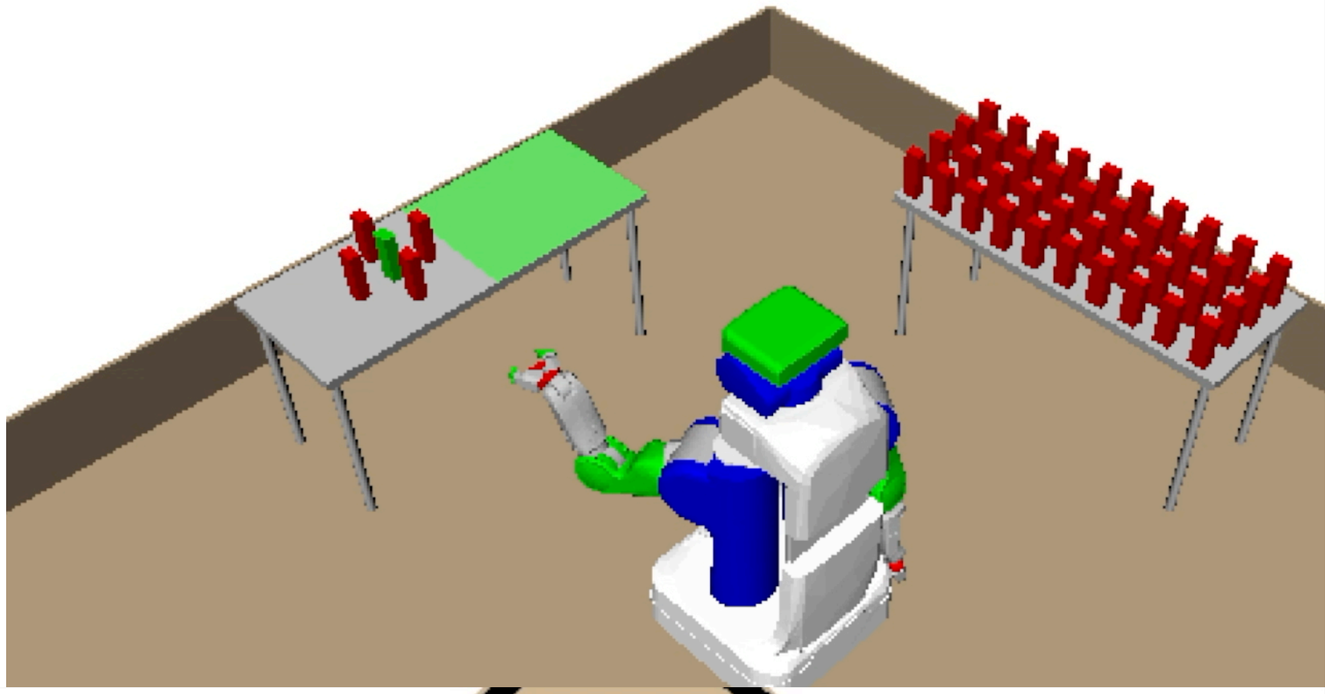


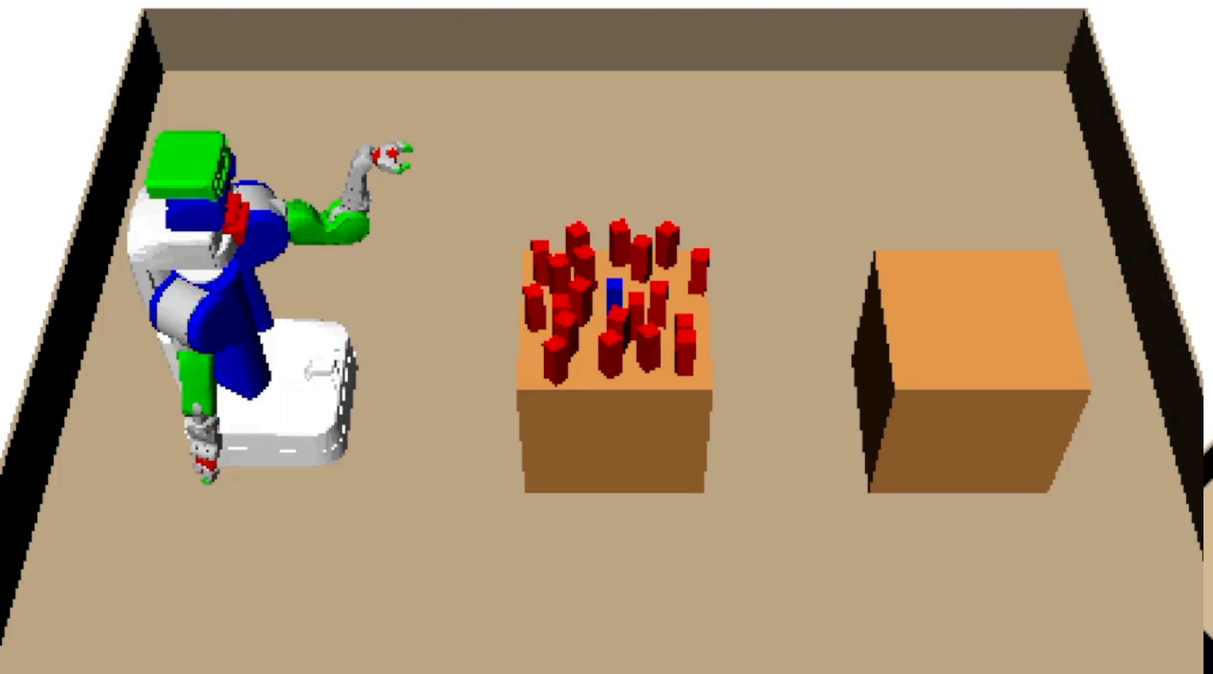
Lazy Samples

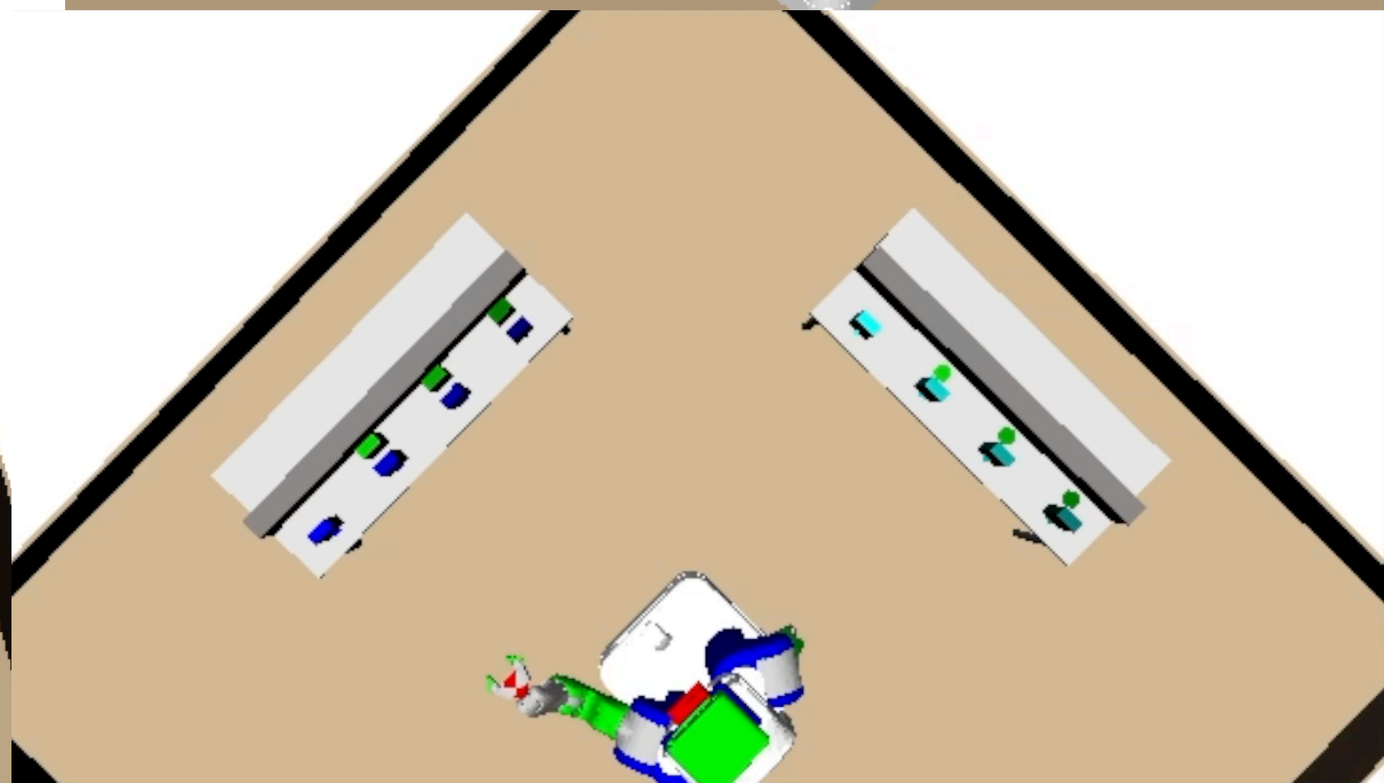
- Lazily sampling values **reduces overhead** from unnecessarily using conditional samplers
- Example: [Placement(*A*)→**pA1**, Grasp(*A*)→**gA1**, IK(*A*, *pA0*, **gA1**)→**q1**, IK(*A*, **pA1**, **gA1**)→**q2**, Motion(*q0*, **q1**)→**t1**, MotionH(**q1**, **q2**, *A*, **gA1**)→**t2**, Motion(**q2**, *q0*)→**t3**]
  - *Italics* are real samples, **bold** are lazy samples
- Lazy discretization can still be large
  - **Share lazy samples** across a sampler
  - Overly **optimistic**, but resolved through extra search

Scaling Experiments

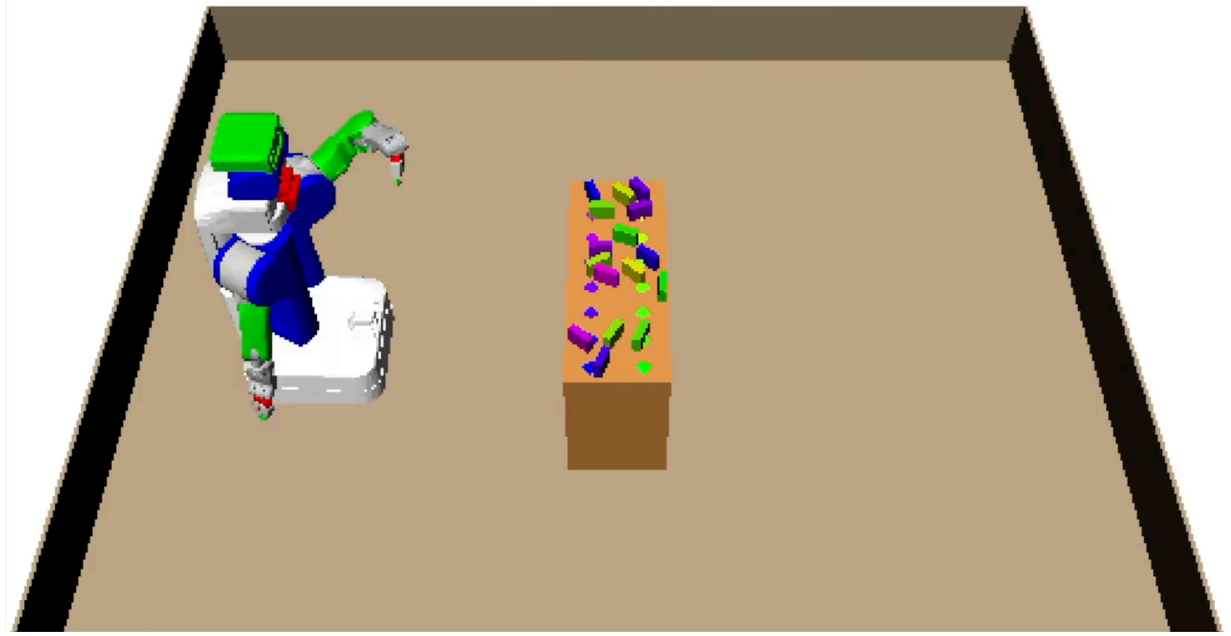


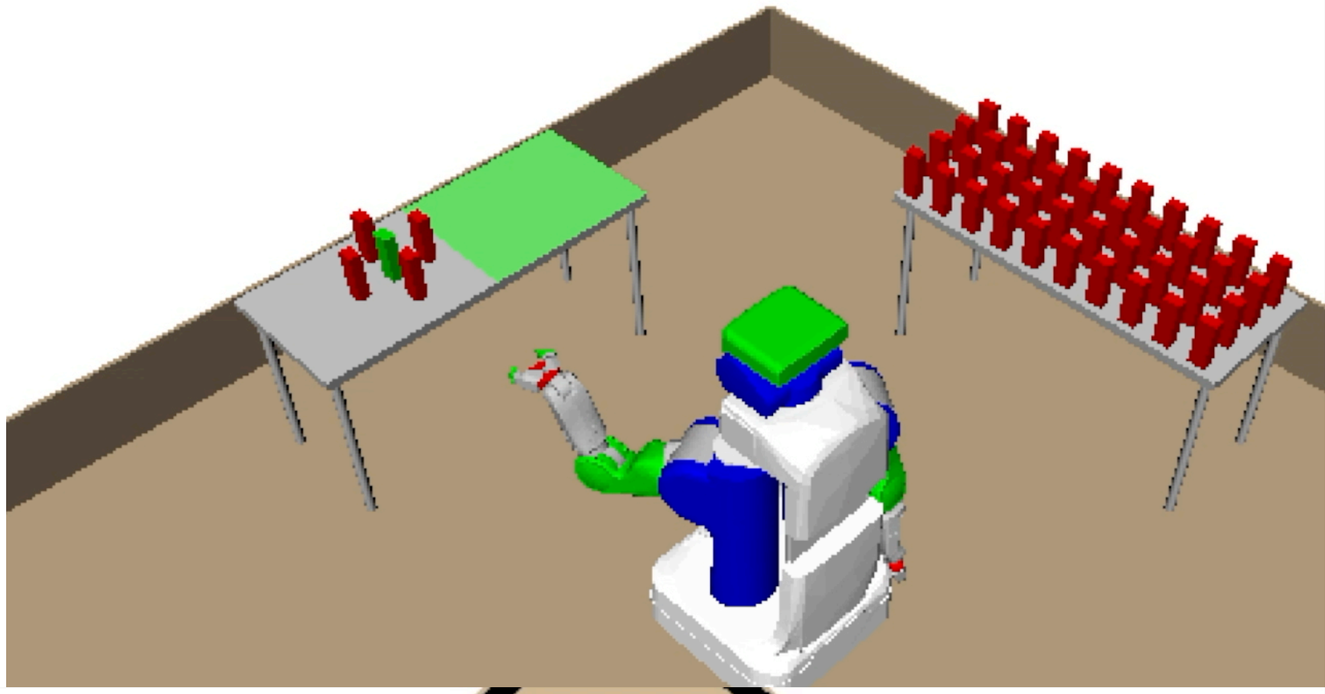


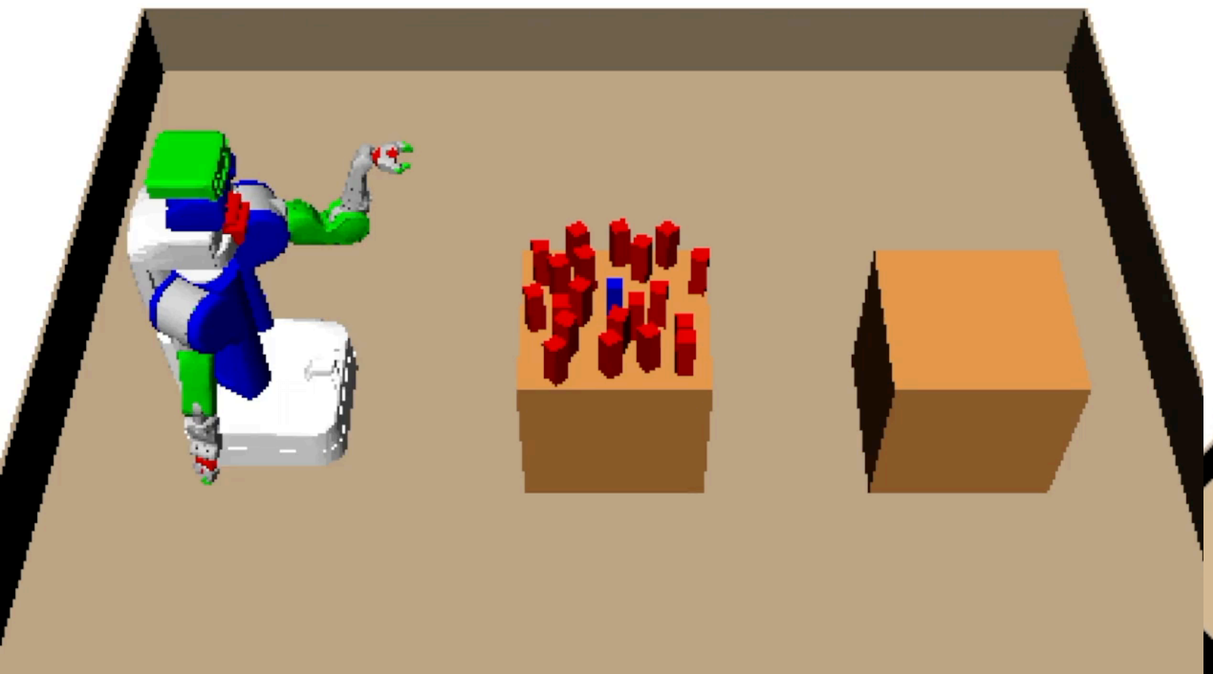


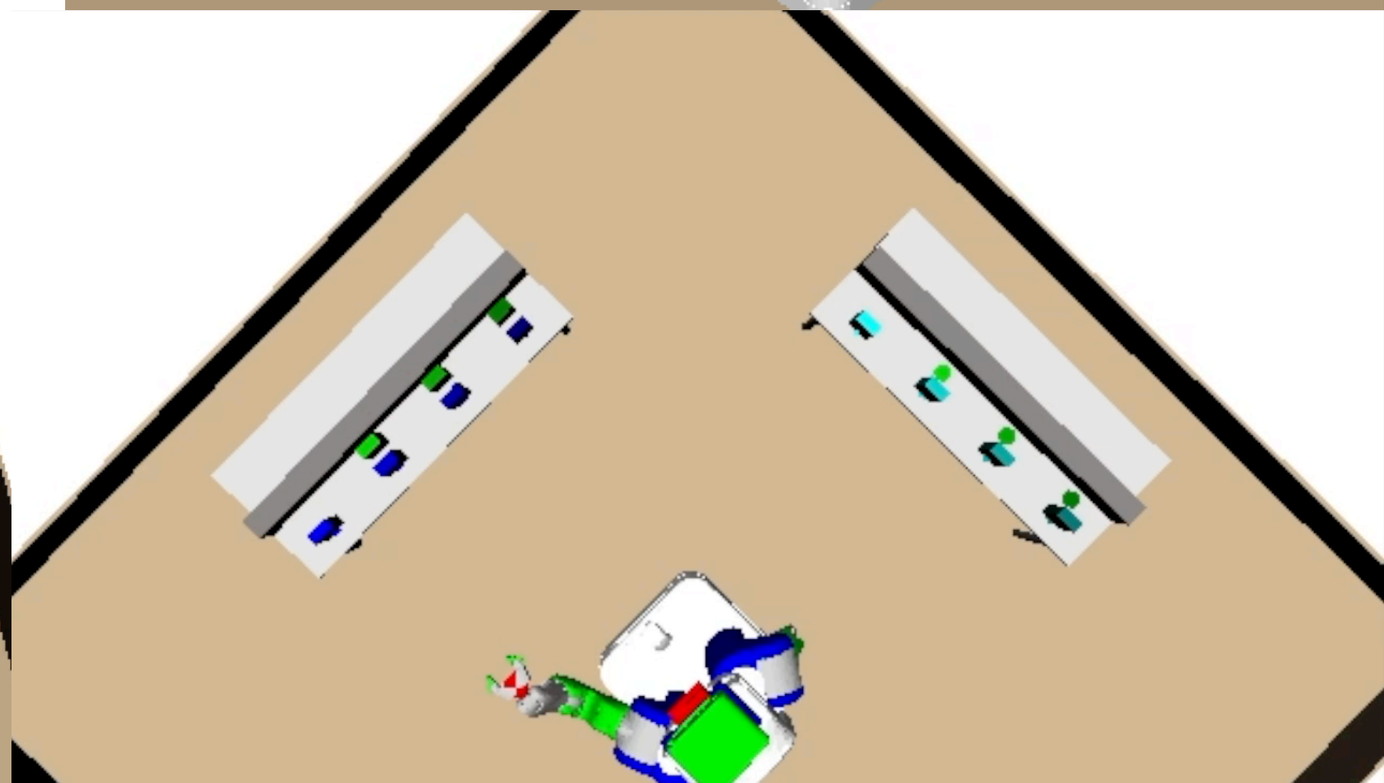


Scaling Experiments



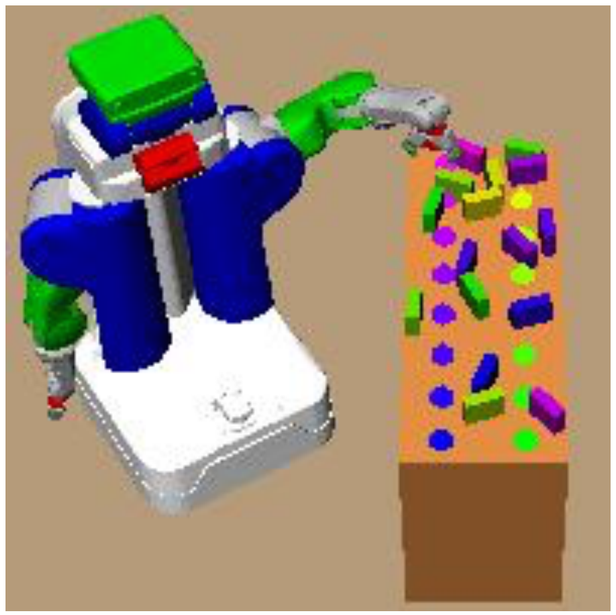


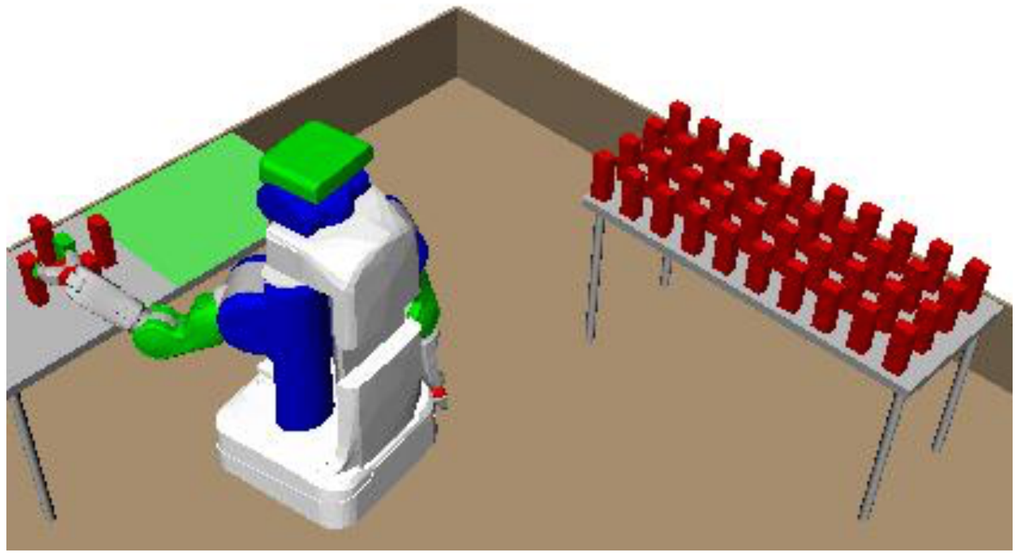


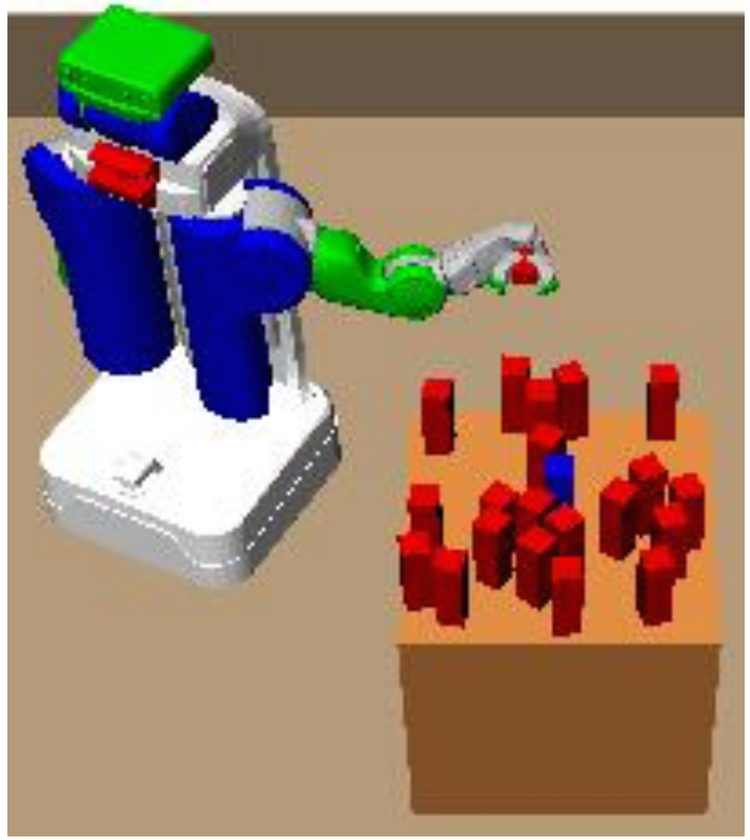


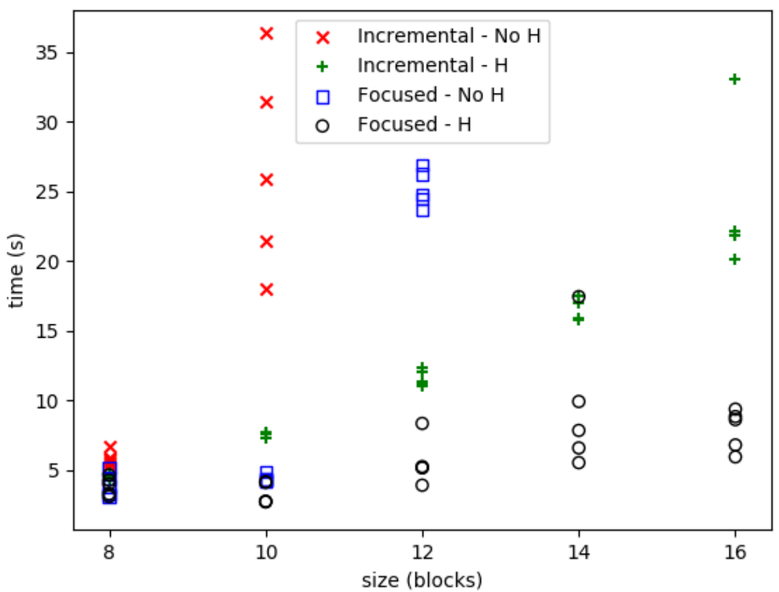
Scaling Experiments

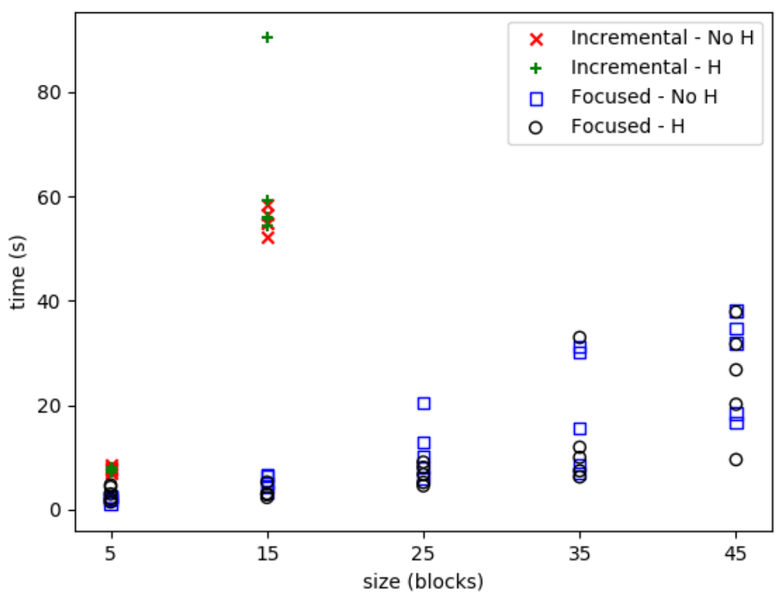
- Focused outperforms incremental
- Heuristic search outperforms BFS

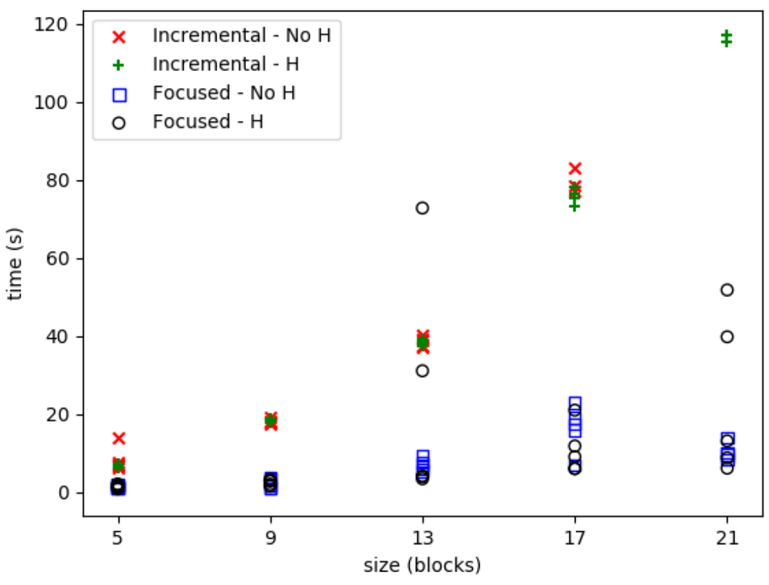




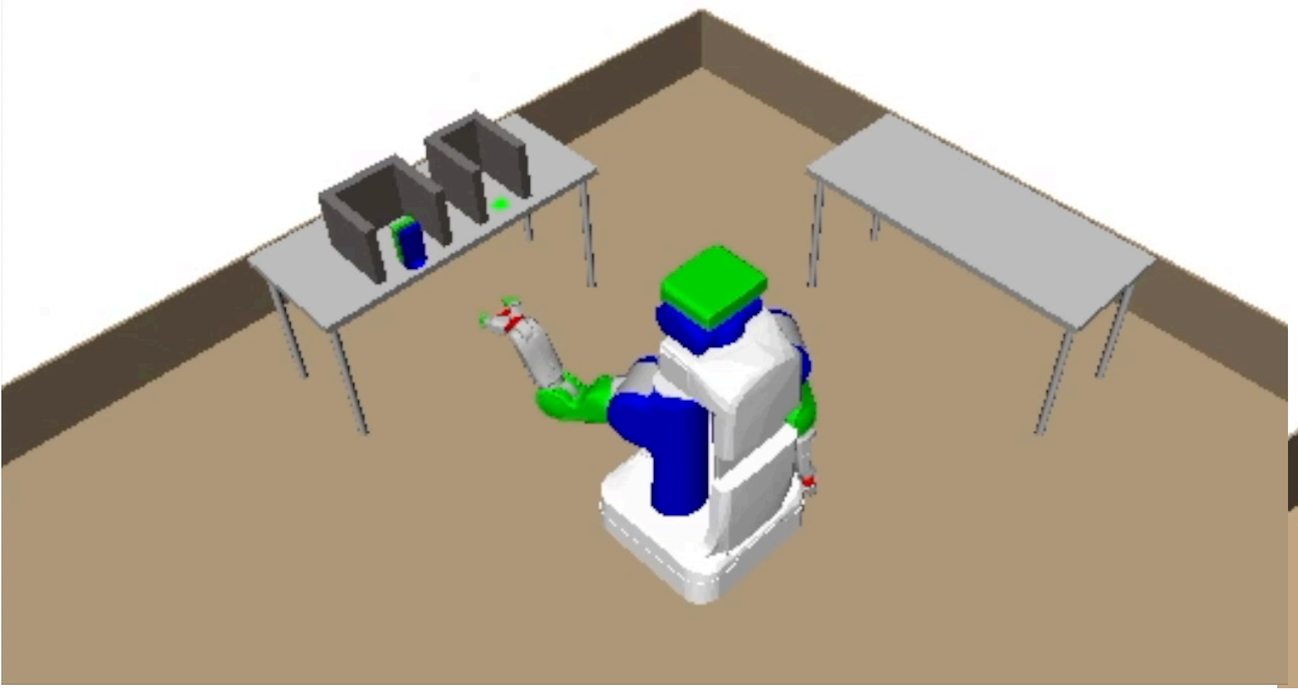


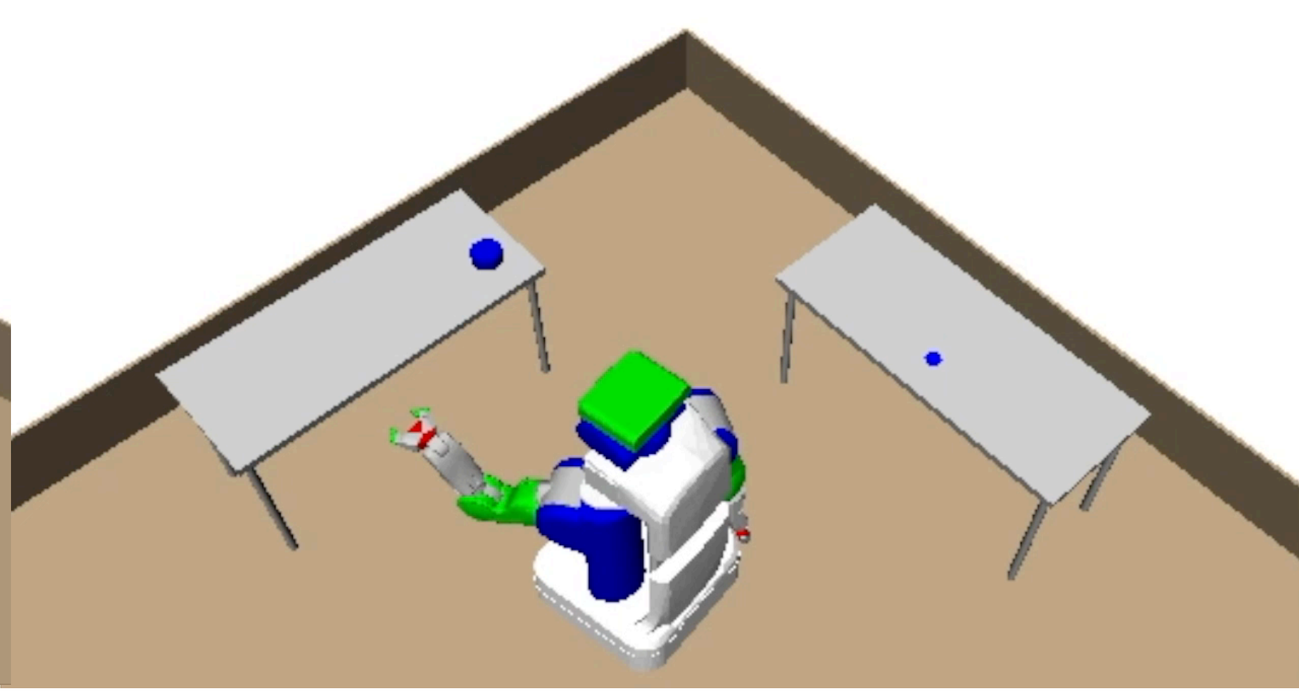


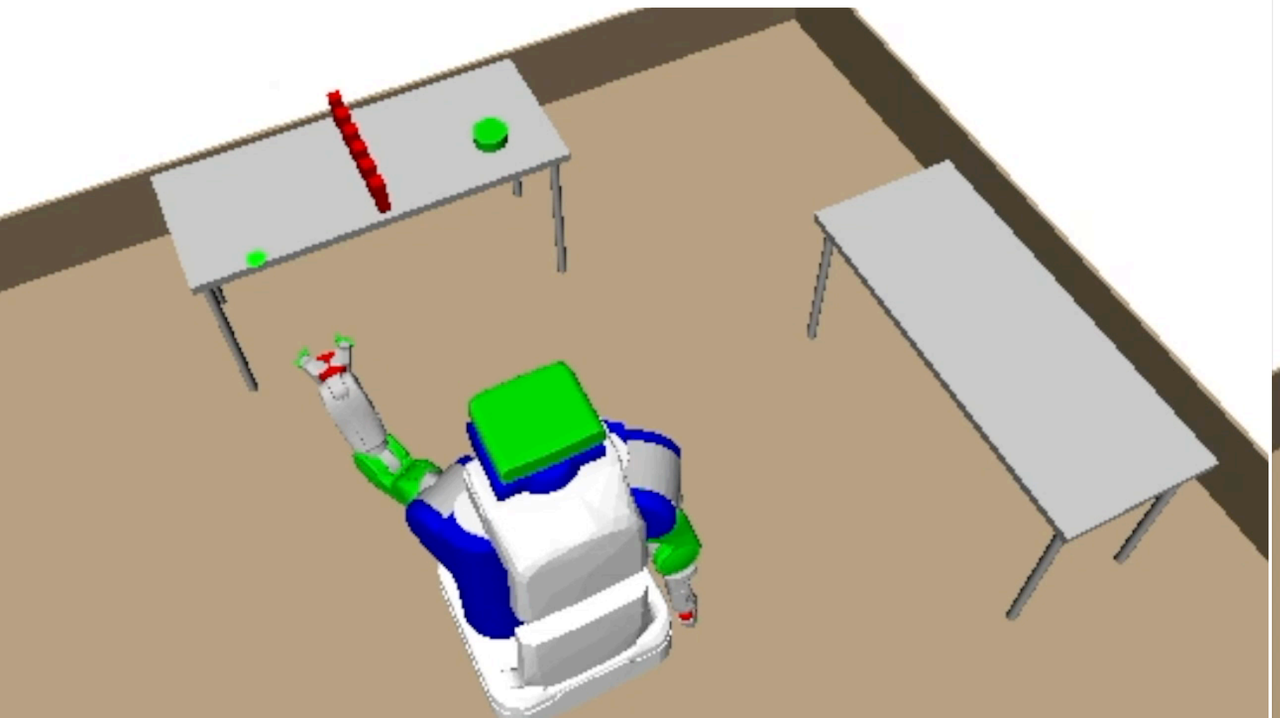


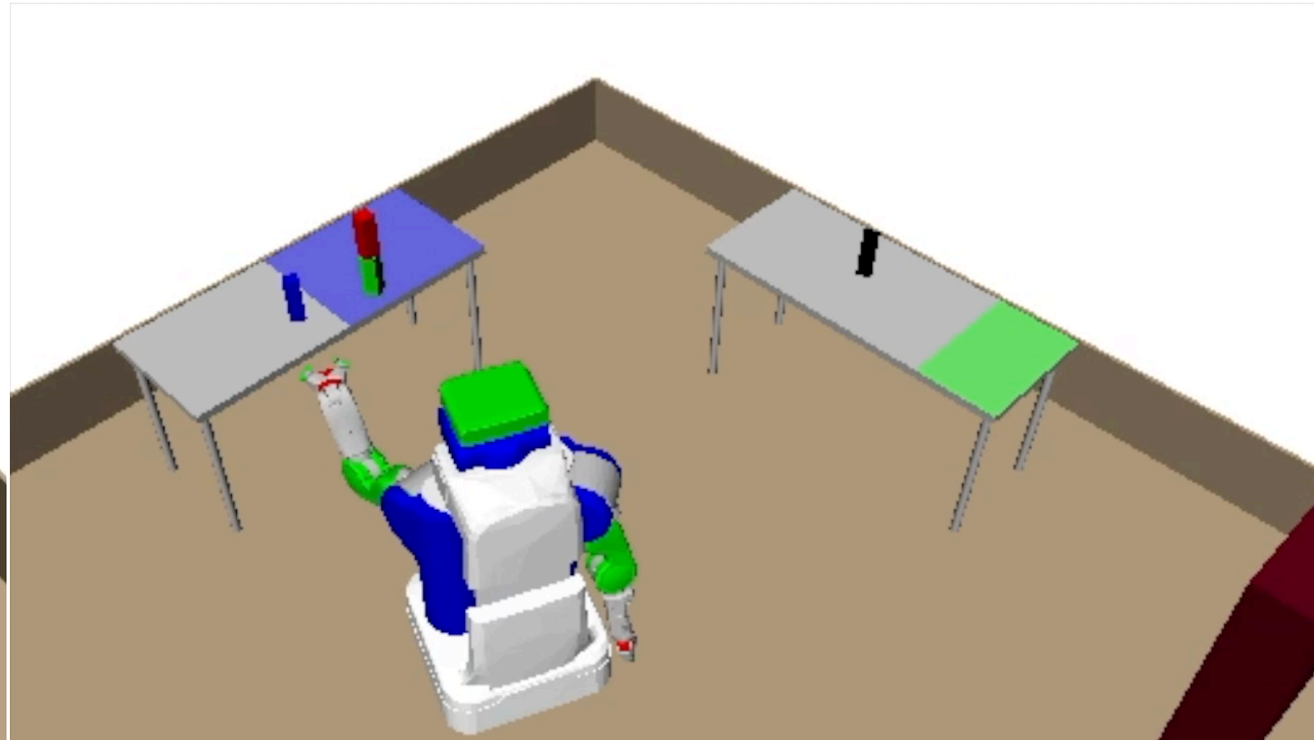


Diverse Experiments

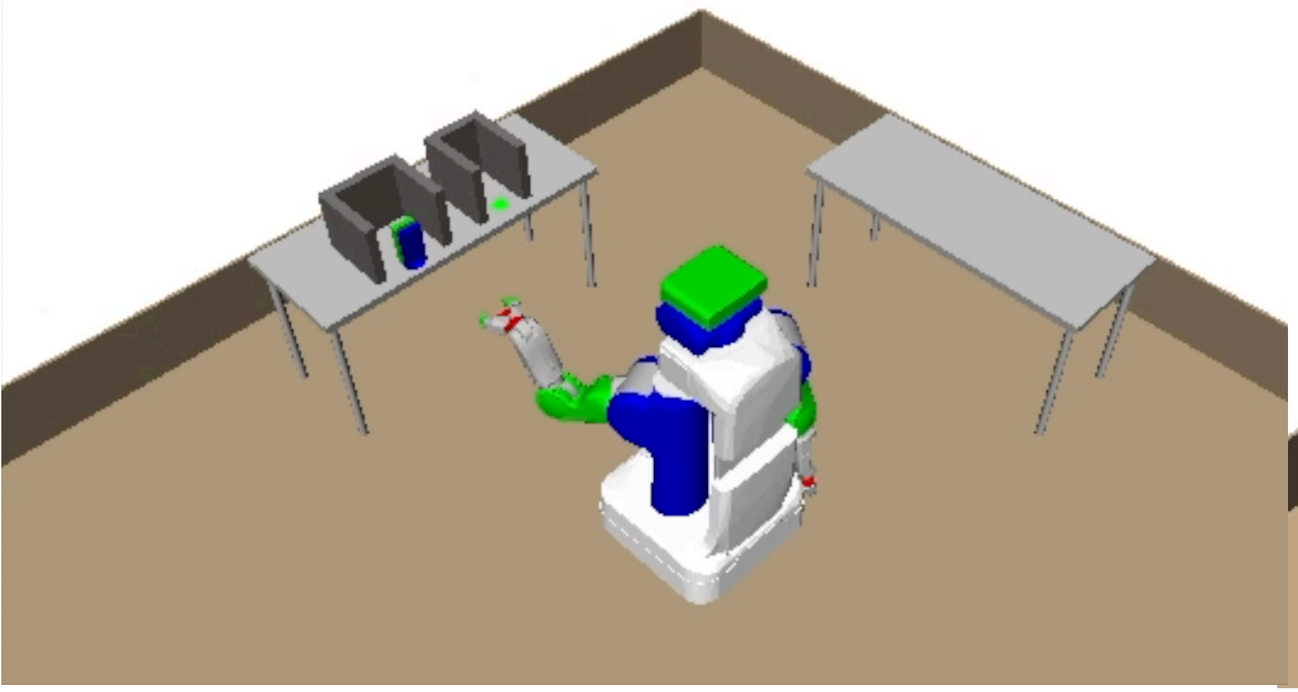


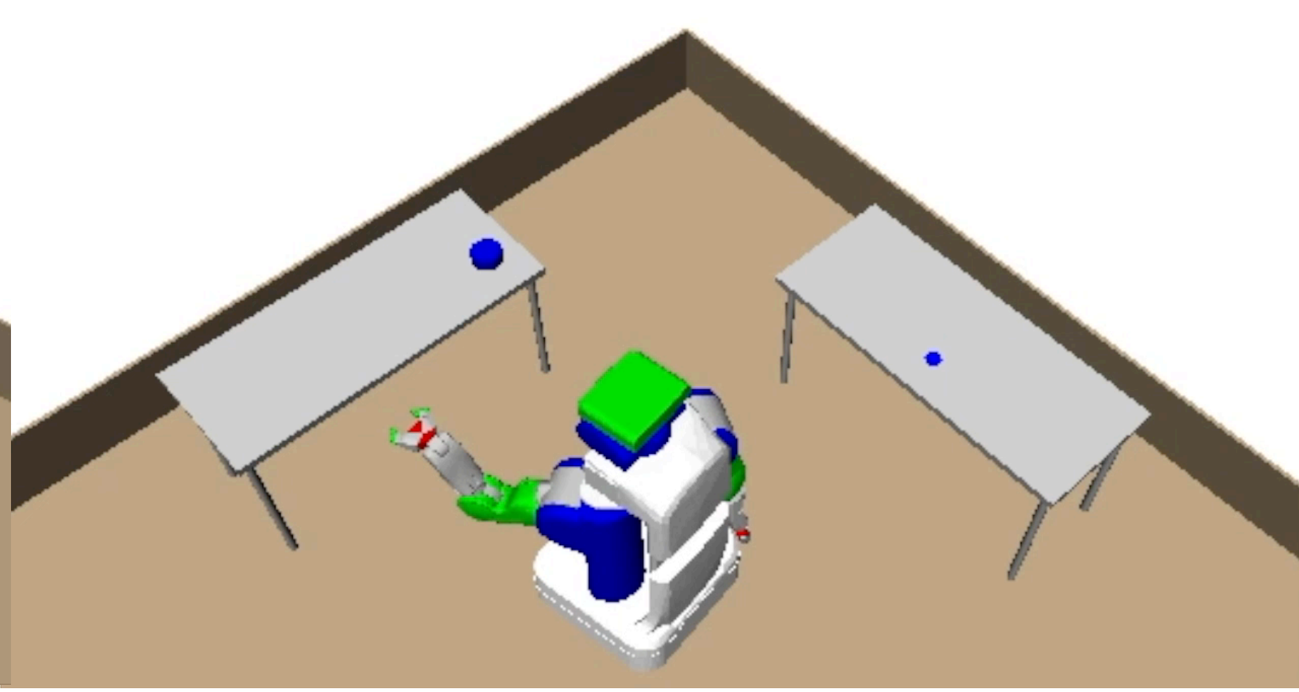


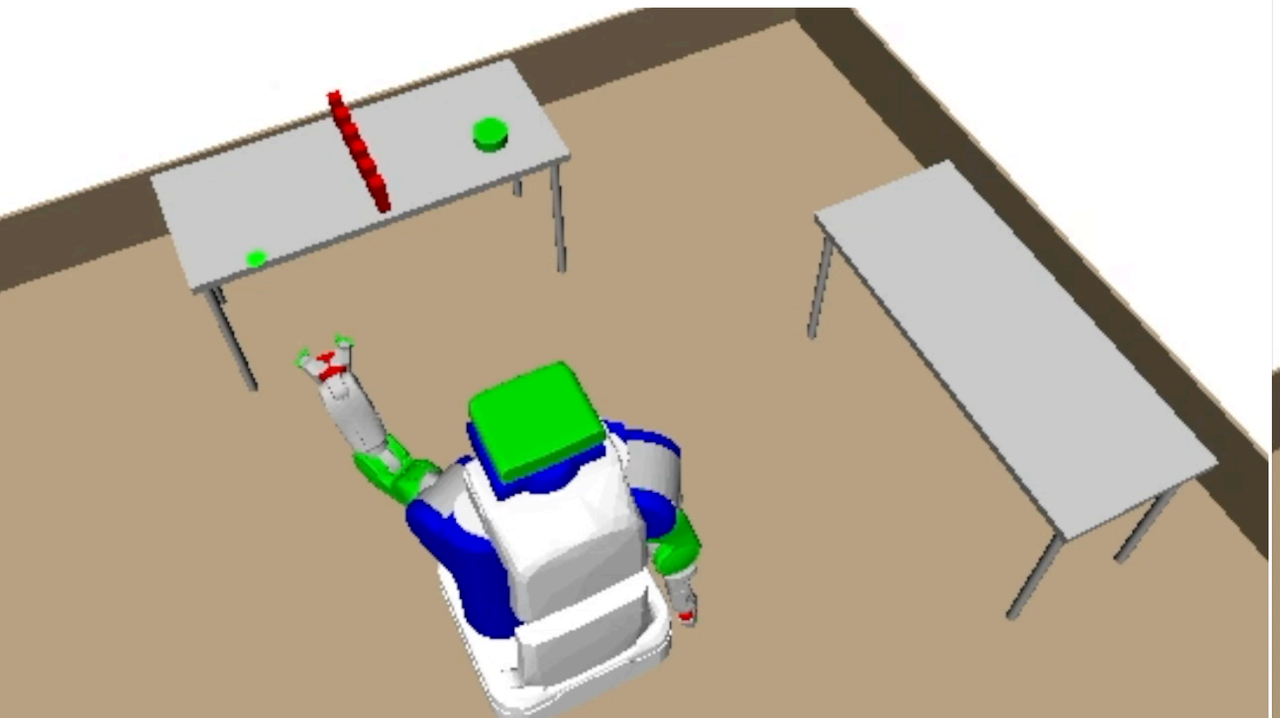


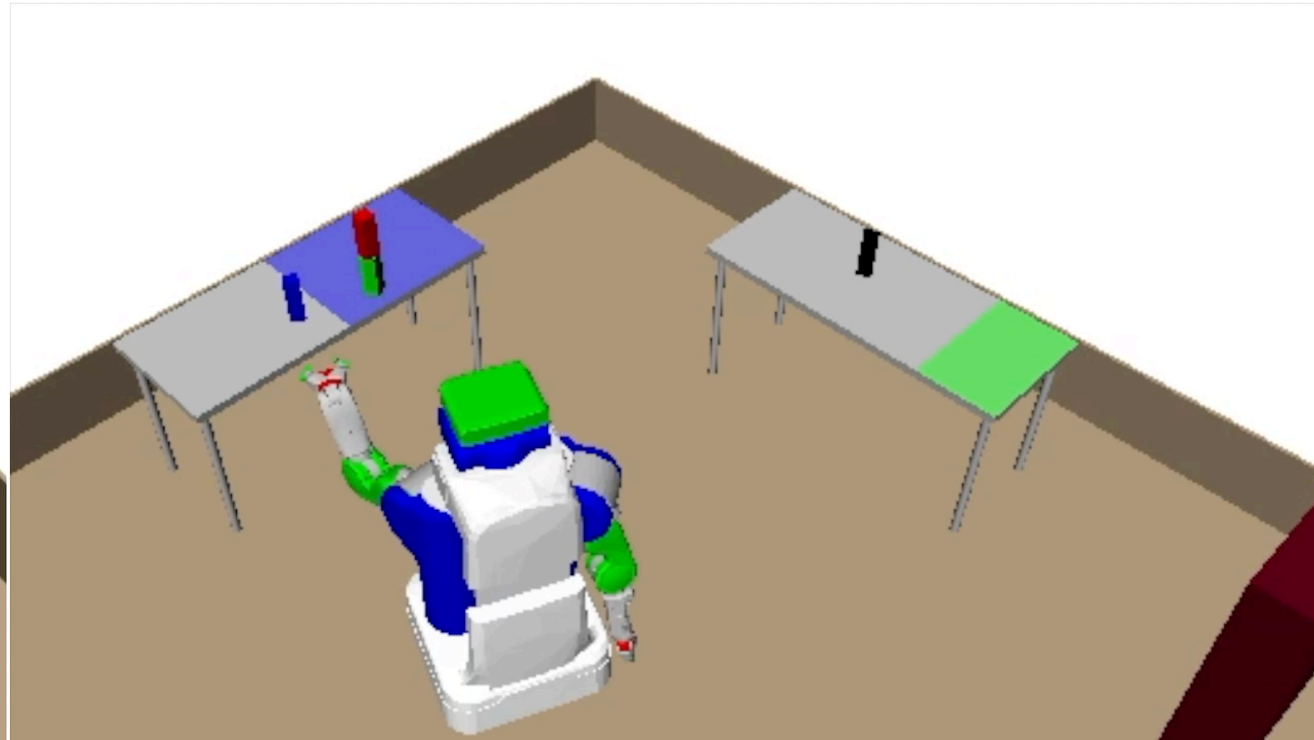


Diverse Experiments

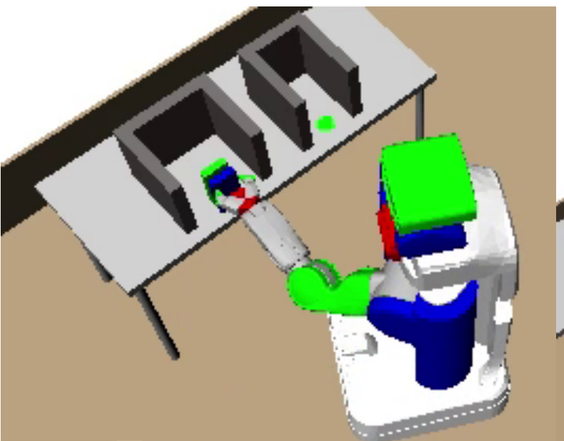
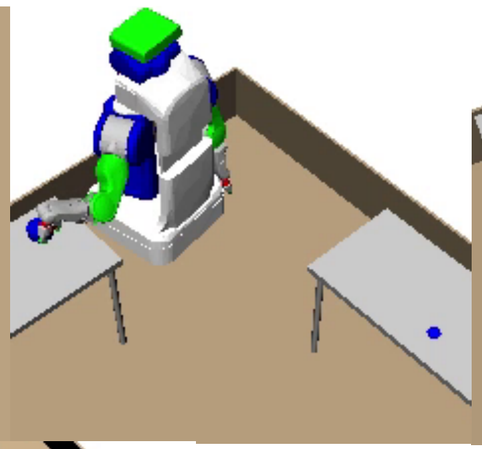
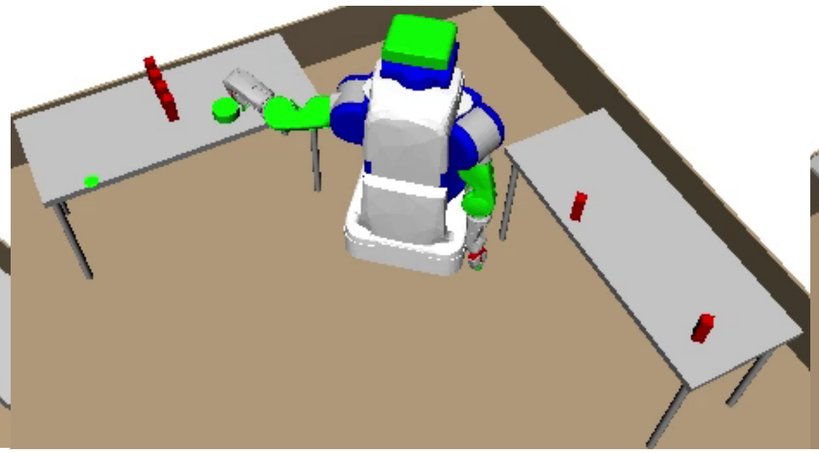
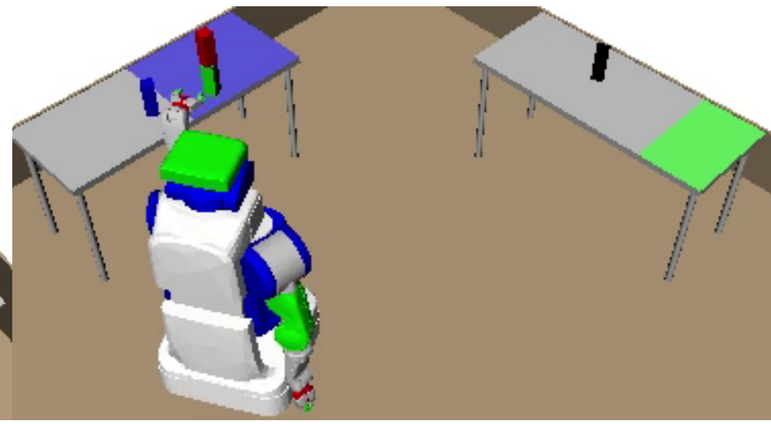
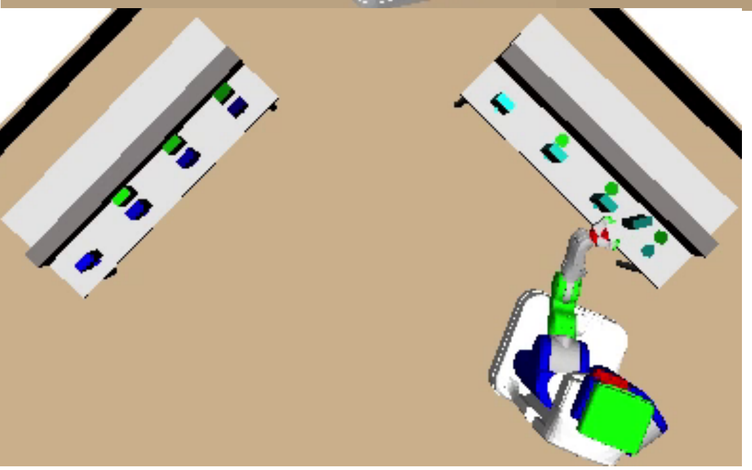
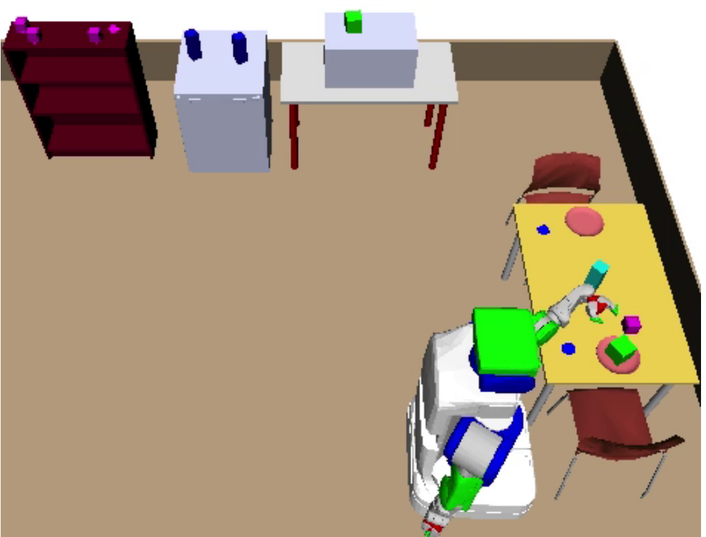








Diverse Experiments

	Incr.		*Incr. - H*		*Focus*		*Focus - H*	
Problem	**%**	**t**	**%**	**t**	**%**	**t**	**%**	**t**
Regrasp	98	1	100	2	98	1	95	1
Push	100	11	100	13	100	13	100	9
Wall	95	10	98	13	100	6	100	8
Stacking	100	9	100	9	100	2	100	3
Nonmon.	25	21	98	15	0	-	88	43
Dinner	0	-	100	27	0	-	98	22

Success percentage (%), Average runtime in sec. (t)

https://github.com/caelan/factored-transition-systems

```
from samplers import plan_motion, sample_q, collision, pose_gen, grasp_gen, inverse_kin_gen
Motion = ConstraintType(); Kin = ConstraintType()
Pose = ConstraintType(); Grasp = ConstraintType()
CFree = ConstraintType(); CFreeH = ConstraintType()
initial_state = dict([('q', q0), ('h', None)] + init_poses)
goal_constraints = [Equal(X[o], p) for o, p in goal_poses.items()]
samplers = [
    Sampler(outputs=('q',), gen_fn=sample_q),
    Sampler(inputs=('q', 'q'), outputs=('t',), constraints=[Motion(I[0], O[0], I[1])], gen_fn=plan_motion)]
transition = [
    [Motion(X['q'], U['t'], nX['q']), Equal(X['h'], None), const('h')] +
        [CFree(U['t'], X[o]) for o in init_poses] + [const(o) for o in init_poses]]
for o in init_poses:
    transition += [
        [Kin(X[o], nX[o], X['q']), Equal(X['h'], None), Equal(nX['h'], o), const('q')] +
            [const(o2) for o2 in init_poses if o2 != o],
        [Kin(nX[o], X[o], X['q']), Equal(X['h'], o), Equal(nX['h'], None), const('q')] +
            [const(o2) for o2 in init_poses if o2 != o],
        [Motion(X['q'], U['t'], nX['q']), Equal(X['h'], o), const('h')] +
            [CFreeH(U['t'], X[o], X[o2]) for o2 in init_poses if o2 != o] +
            [const(o2) for o2 in init_poses]]
    samplers += [
        Test(inputs=('t', o), domain=[Pose(I[1])], constraints=[CFree(I[0], I[1])],
             test_fn=lambda t, p: not collision(t, o, p, None, None)),
        Sampler(outputs=(o,), constraints=[
                Pose(O[0])], gen_fn=lambda: pose_gen(o)),
        Sampler(outputs=(o,), constraints=[
                Grasp(O[0])], gen_fn=lambda: grasp_gen(o)),
        Sampler(inputs=(o, o), domain=[Pose(I[0]), Grasp(I[1])],
                outputs=('q',), constraints=[Kin(I[0], I[1], O[0])],
                gen_fn=lambda p, g: inverse_kin_gen(o, p, g))] + \
        [Test(inputs=('t', o, o2), domain=[Grasp(I[1]), Pose(I[2])],
              constraints=[CFreeH(I[0], I[1], I[2])],
              test_fn=lambda t, p, g: not collision(t, o, p, o2, g))
         for o2 in init_poses if o != o2]
```

STRIPStream

- Predicate-based AI Planning Language
  - STRIPS/PDDL
  - pre/eff actions
- More expressive
- Same algorithms

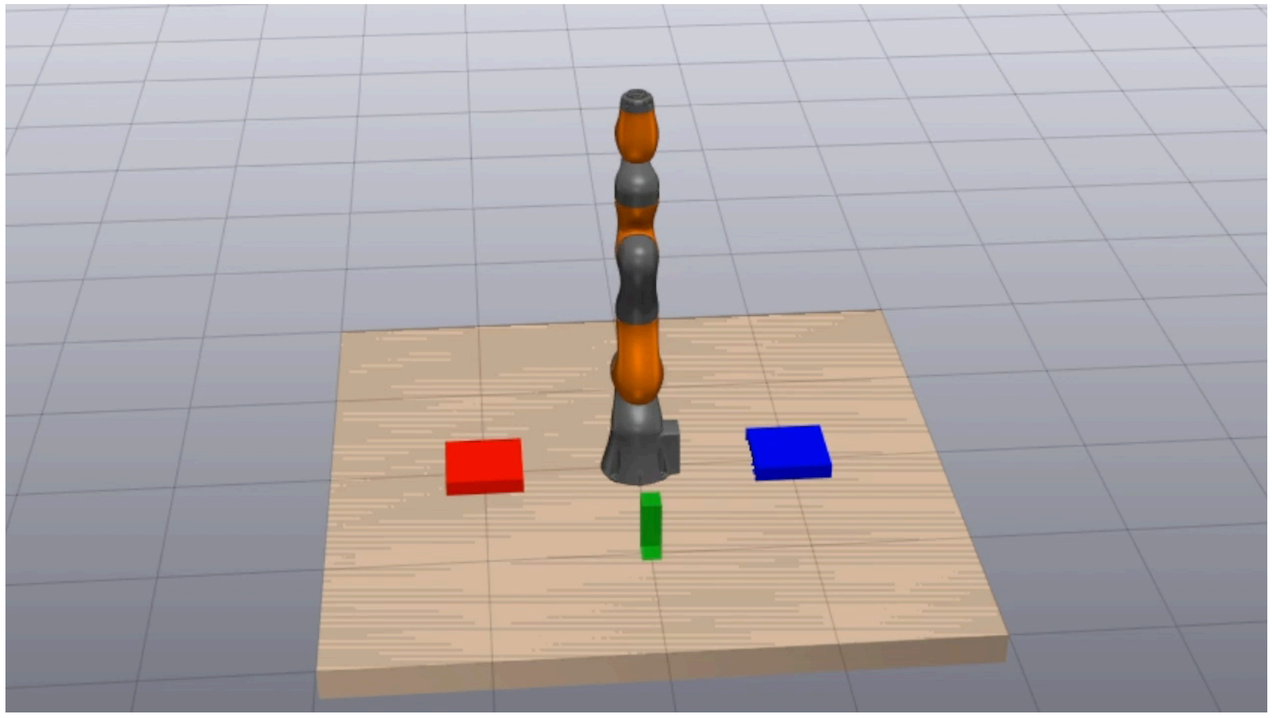
https://github.com/caelan/ss

https://github.com/caelan/pddlstream

```
Block = Predicate('?b') # Static
Pose = Predicate('?b ?p'); Grasp = Predicate('?b ?p'); Conf = Predicate('?q')
Kin = Predicate('?b ?p ?g ?q'); Traj = Predicate('?t'); Motion = Predicate('?q ?t ?q2')
AtConf = Predicate('?q') # Fluent
AtPose = Predicate('?b ?p'); Empty = Predicate(''); Holding = Predicate('?b ?g')
Unsafe = Predicate('?t') # Derived
Distance = Function('?t', dom=[Traj('?t')], fn=path_length) # External
Collision = Predicate('?t ?b ?p', dom=[Traj('?t'), Pose('?b ?p')], fn=check_collision)
CollisionH = Predicate('?t ?b ?p ?b2 ?g', dom=[Traj('?t'), Pose('?b ?p'), Grasp('?b2 ?g')],
                       fn=check_holding_collision)
actions = [
    Action(name='move', param='?q ?t ?q2',
           pre=[Motion('?q ?t ?q2'), AtConf('?q'), not Unsafe('?t')],
           eff=[AtConf('?q2'), not AtConf('?q'), Increase(TotalCost(), Distance('?t'))]),
    Action(name='pick', param='?b ?p ?g ?q',
           pre=[Kin('?b ?p ?g ?q'), AtPose('?b ?p'), Empty(), AtConf('?q')],
           eff=[Holding('?b ?g'), not AtPose('?b ?p'), not Empty(), Increase(TotalCost(),1)]),
    Action(name='place', param='?b ?p ?g ?q',
           pre=[Kin('?b ?p ?g ?q'), Holding('?b ?g'), AtConf('?q')],
           eff=[AtPose('?b ?p'), Empty(), not Holding('?b ?g'), Increase(TotalCost(), 1)])]
axioms = [
    Axiom(param='?t ?b ?p',
          pre=[Collision('?t ?b ?p'), AtPose('?b ?p'), Empty()],
          eff=Unsafe('?t')),
    Axiom(param='?t ?b ?p ?b2 ?g',
          pre=[CollisionH('?t ?b ?p ?b2 ?g'), AtPose('?b ?p'), Holding('?b2 ?g')],
          eff=Unsafe('?t'))]
streams = [
    Stream(name='placement', inp='?b', dom=[Block('?b')], fn=sample_placements,
           out='?p', cert=[Pose('?b ?p')]),
    Stream(name='grasps', inp='?b', dom=[Block('?b')], fn=compute_grasps,
           out='?g', cert=[Grasp('?b ?g')]),
    Stream(name='ik', inp='?b ?p ?g', dom=[Pose('?b ?p'),Grasp('?b ?g')], fn=inverse_kinematics,
           out='?q', cert=[Kin('?b ?p ?g ?q'), Conf('?q')]),
    Stream(name='motion', inp='?q ?q2', dom=[Conf('?q'), Conf('?q2')], fn=plan_motion,
           out='?t', cert=[Motion('?q ?t ?q2'), Traj('?t')])]
```

Ongoing / Future Work

Benchmarking in many frameworks

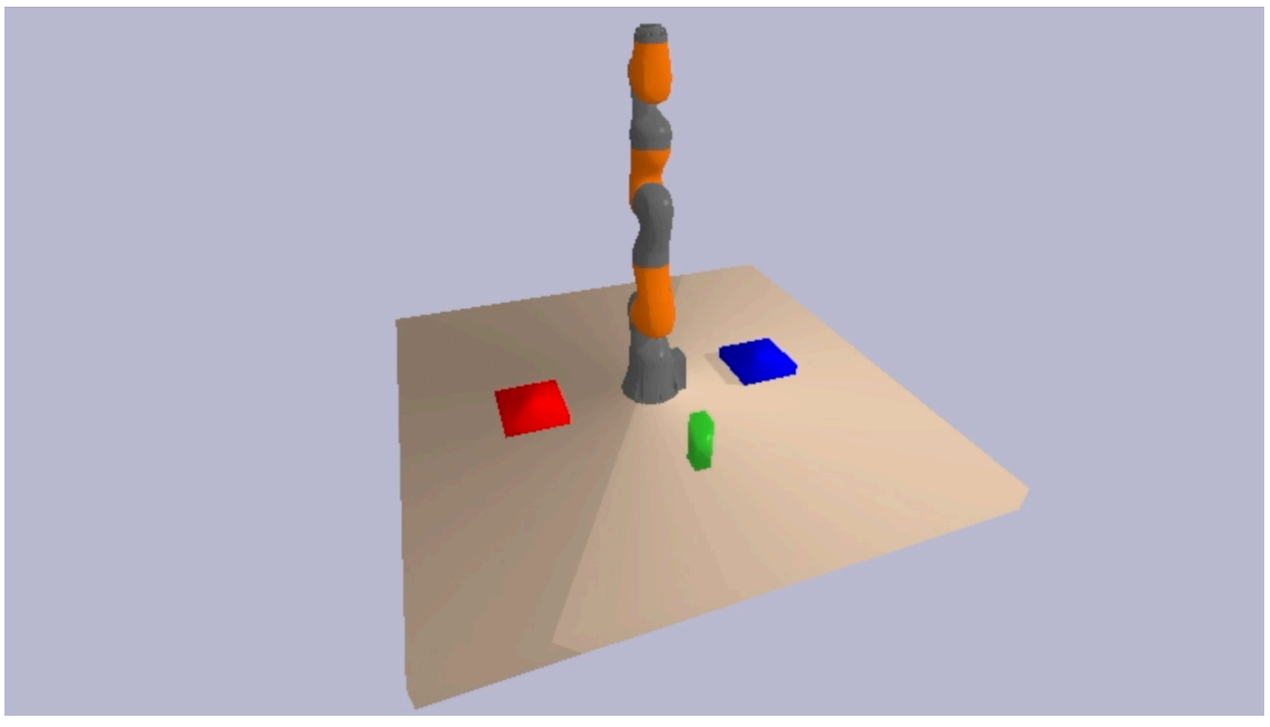


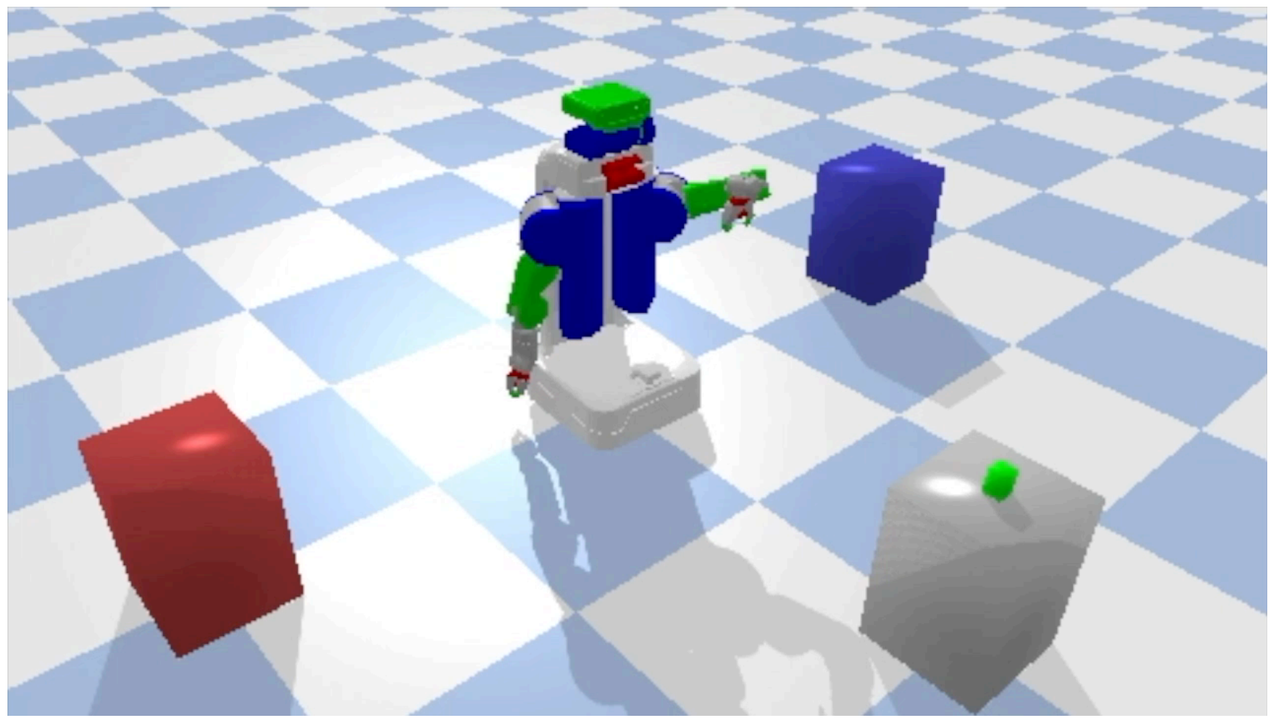
Drake

https://github.com/caelan/ss-drake

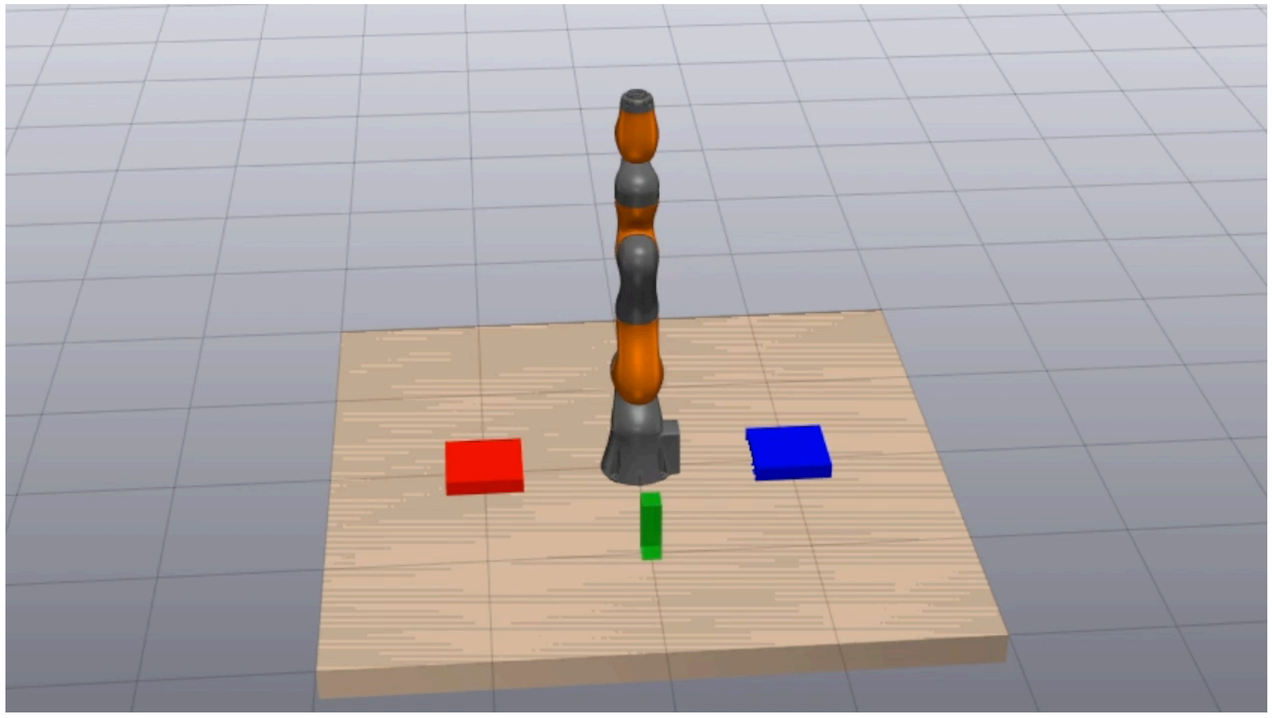
PyBullet

https://github.com/caelan/ss-pybullet





Benchmarking in many frameworks

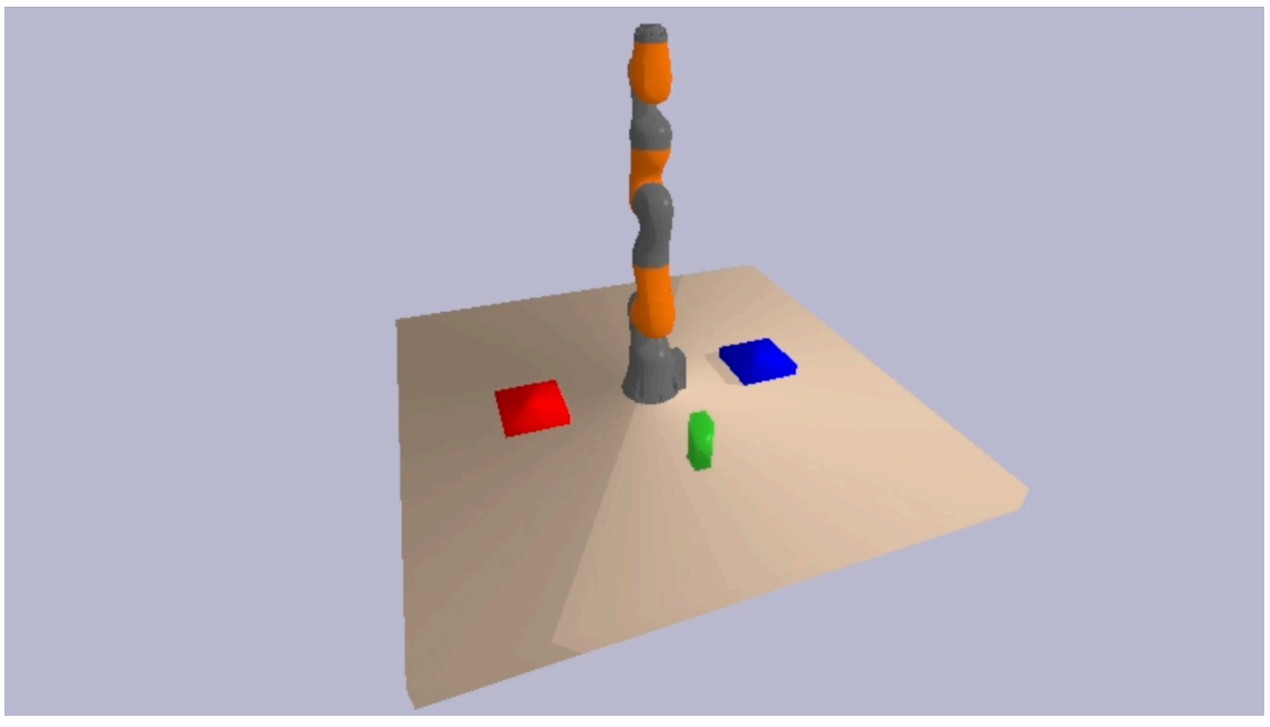


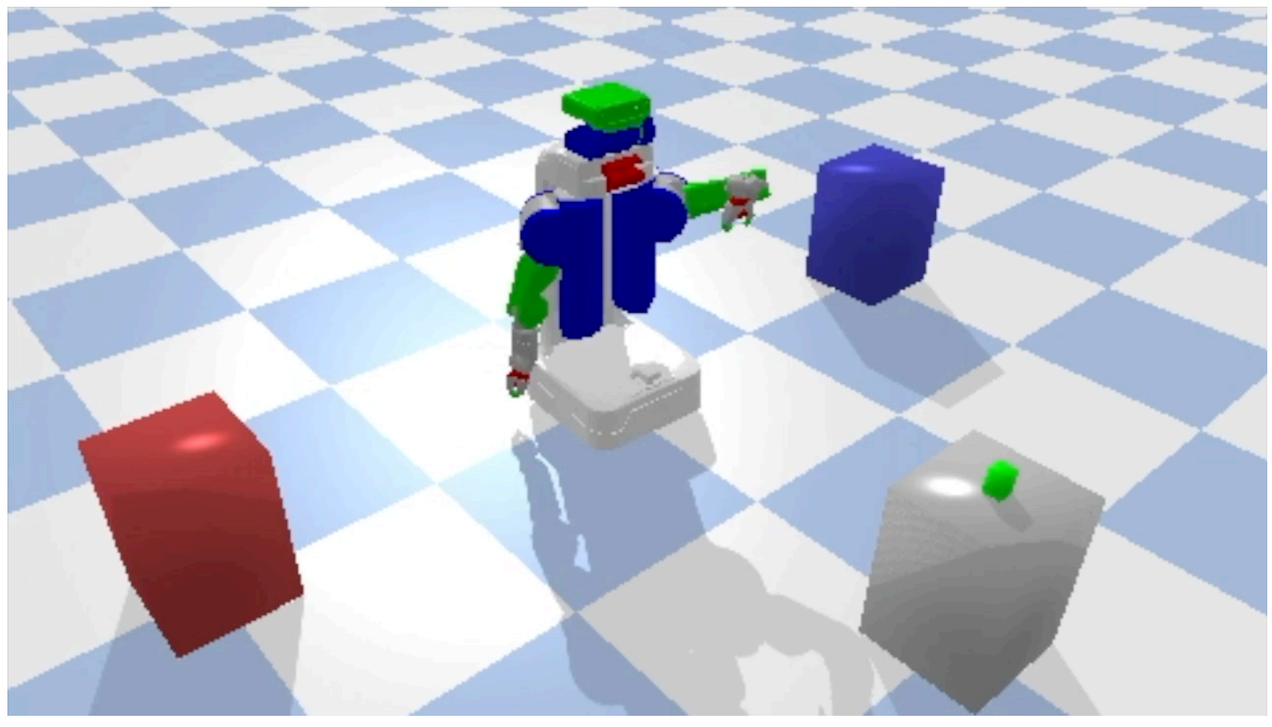
Drake

https://github.com/caelan/ss-drake

PyBullet

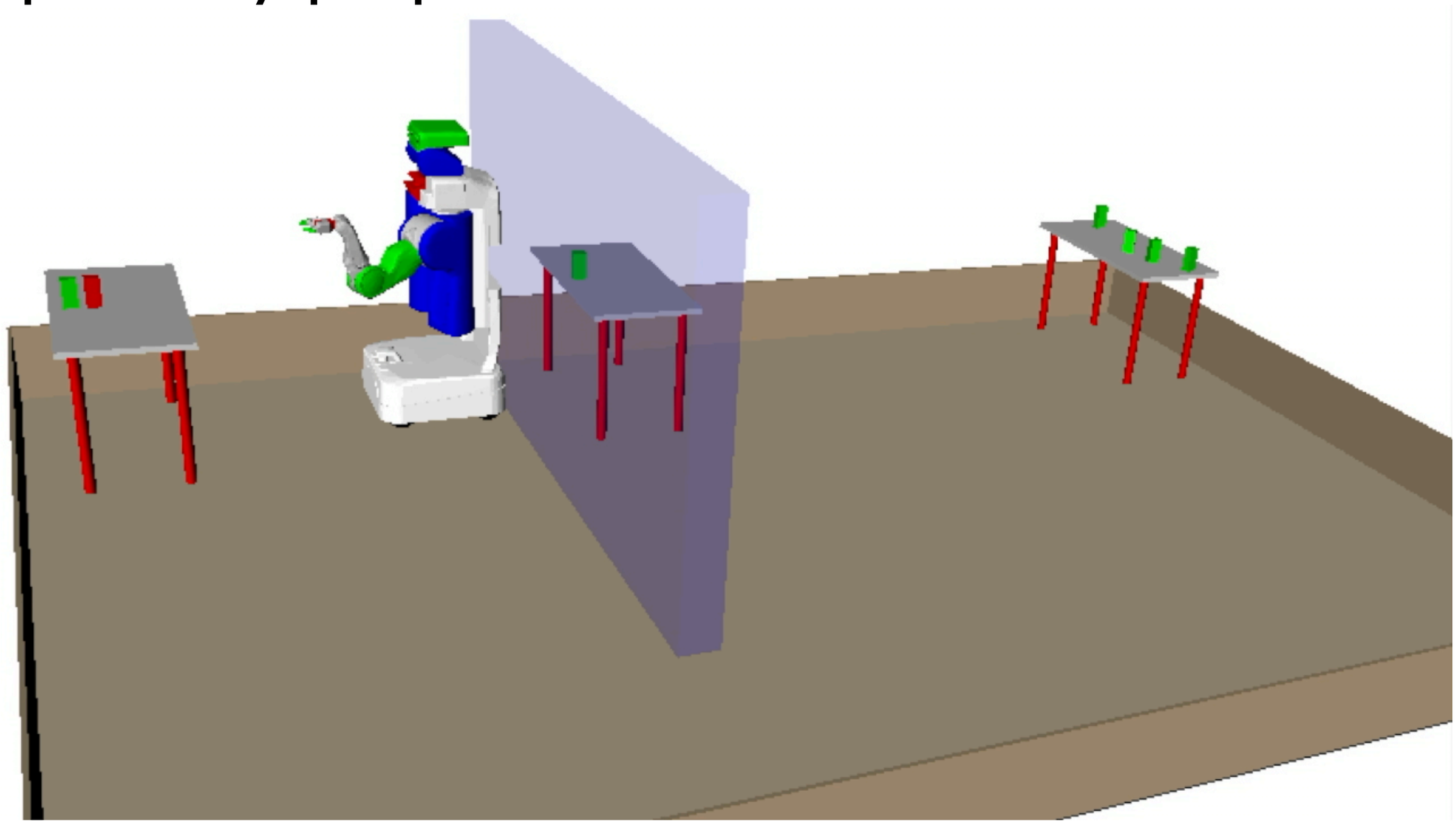
https://github.com/caelan/ss-pybullet





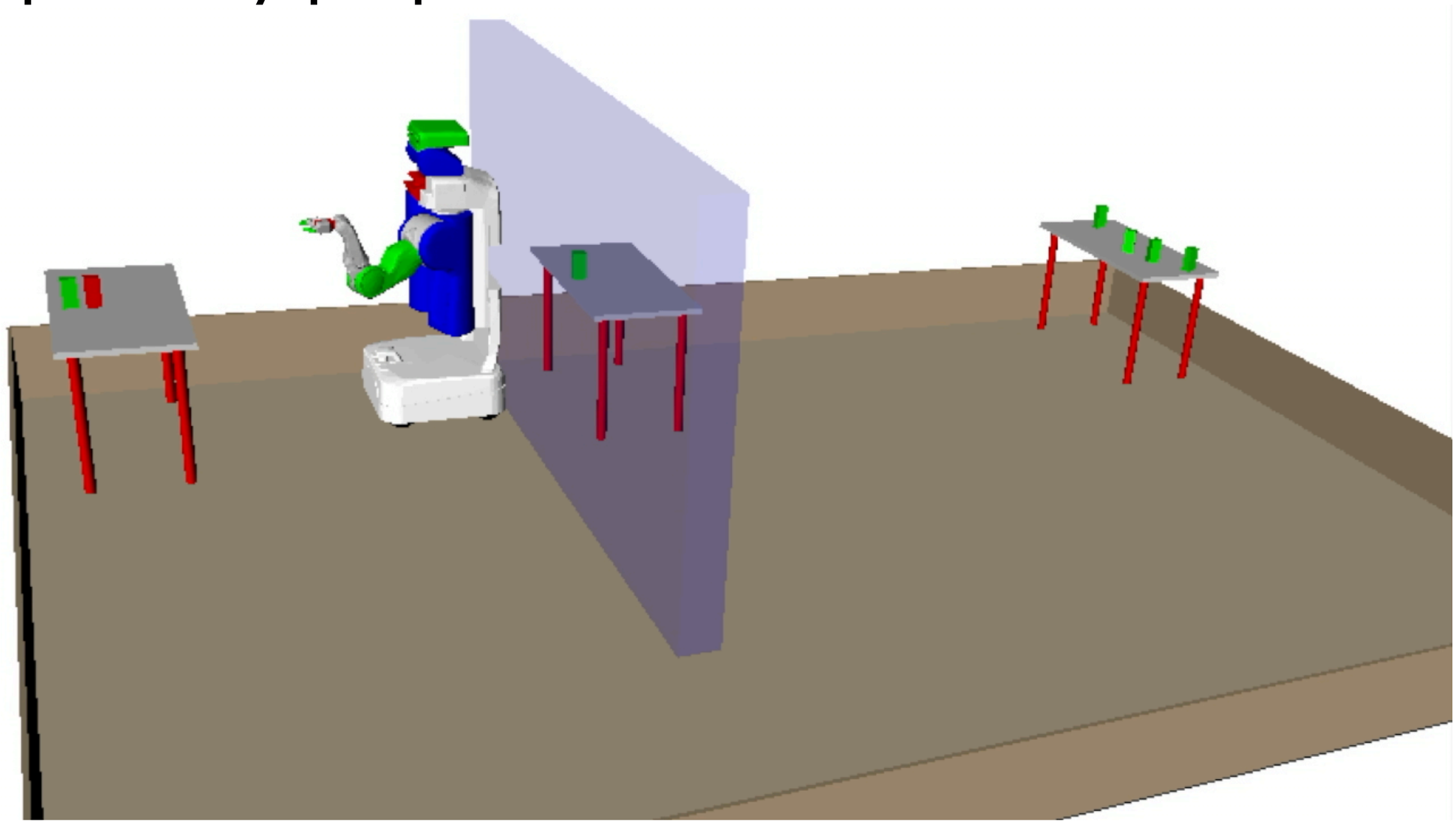
Cost-Sensitive Planning

- **Lower bounds on costs** improve performance
- **Future work:** theoretical analysis of asymptotic optimality properties



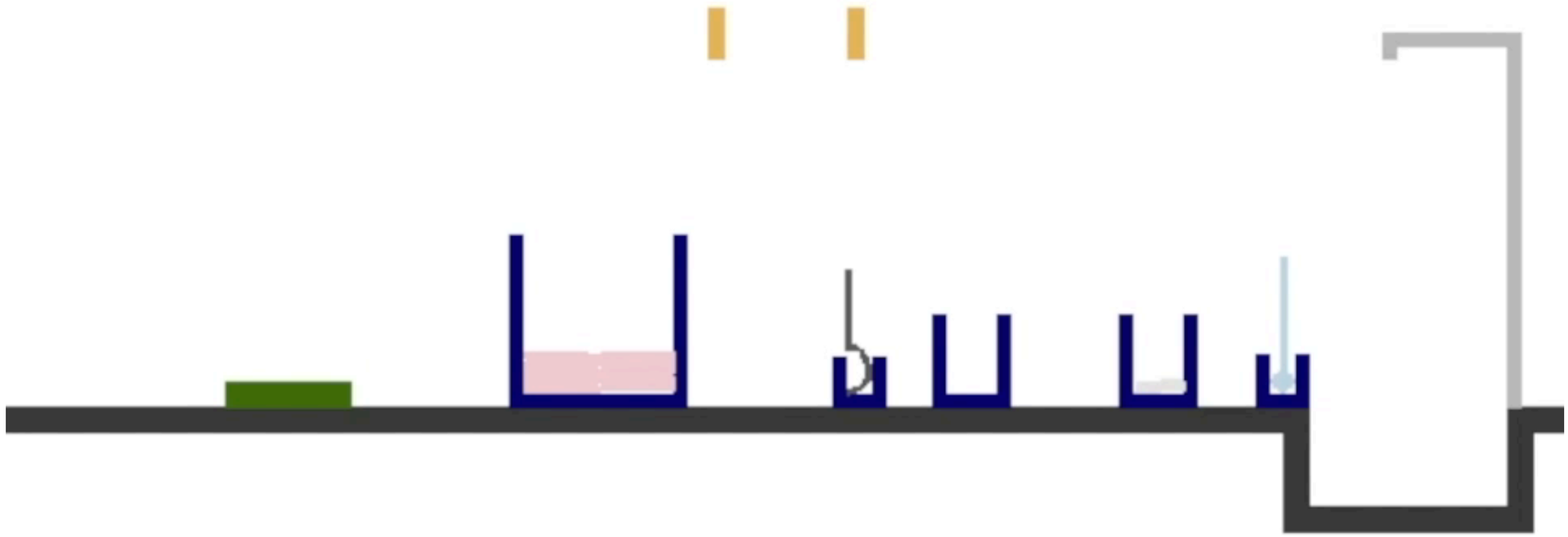
Cost-Sensitive Planning

- **Lower bounds on costs** improve performance
- **Future work:** theoretical analysis of asymptotic optimality properties



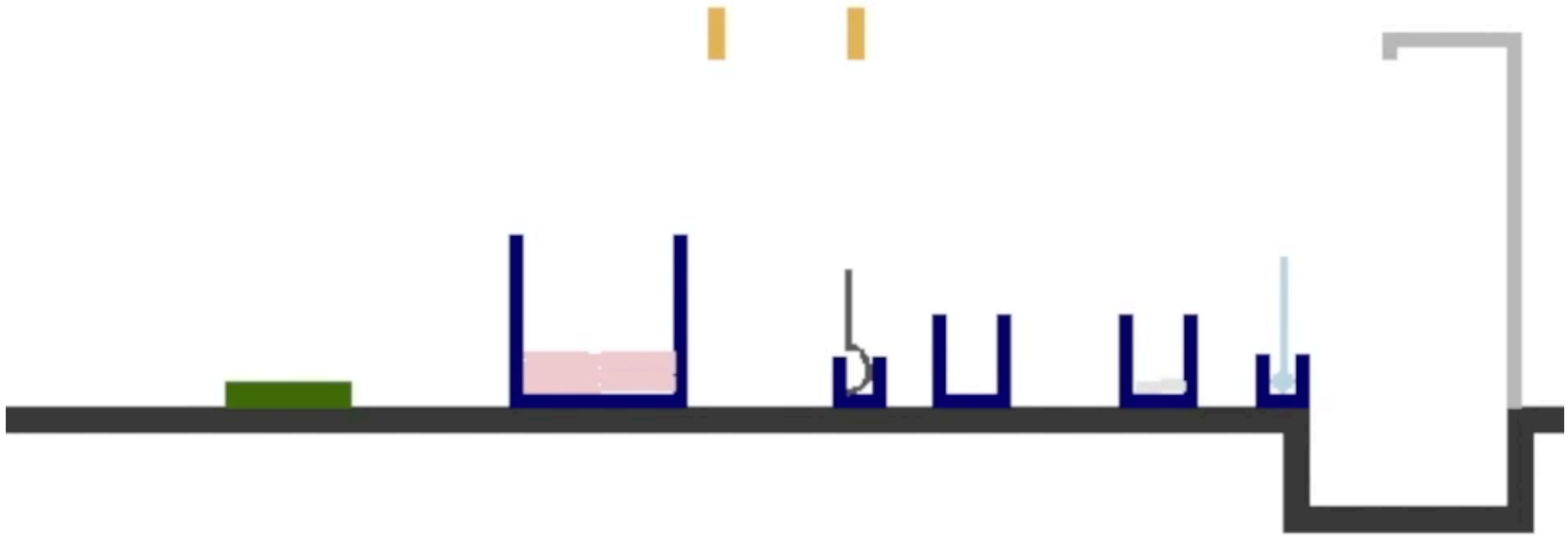
Planning using Learned Samplers

- **Learn diverse samplers** using Bayesian Optimization, Generative Adversarial Networks, …
  - **Active model learning and diverse action sampling for task and motion planning.** Zi Wang, Caelan Reed Garrett, Leslie Pack Kaelbling, Tomás Lozano-Pérez



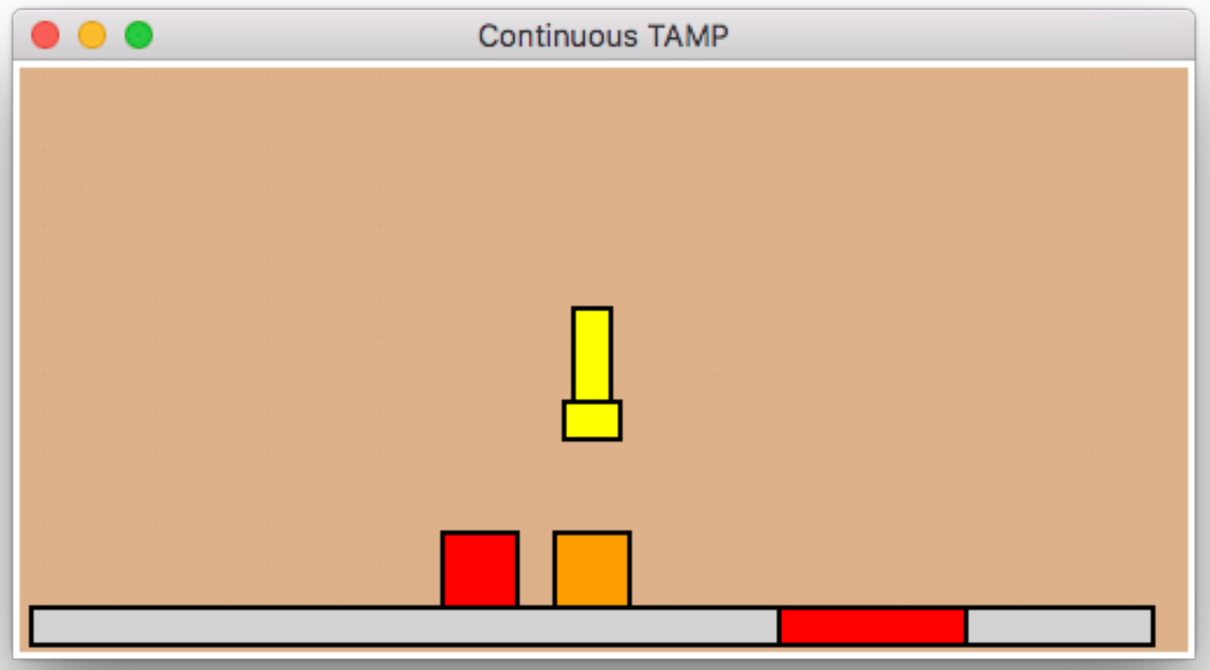
Planning using Learned Samplers

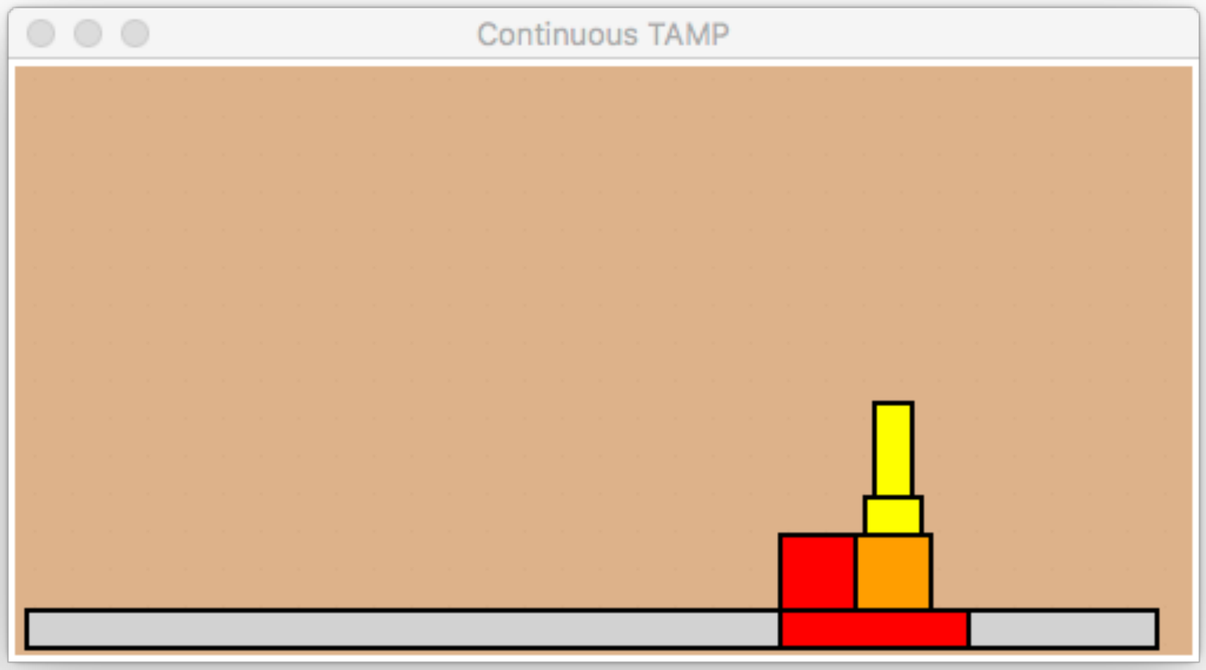
- **Learn diverse samplers** using Bayesian Optimization, Generative Adversarial Networks, …
  - **Active model learning and diverse action sampling for task and motion planning.** Zi Wang, Caelan Reed Garrett, Leslie Pack Kaelbling, Tomás Lozano-Pérez



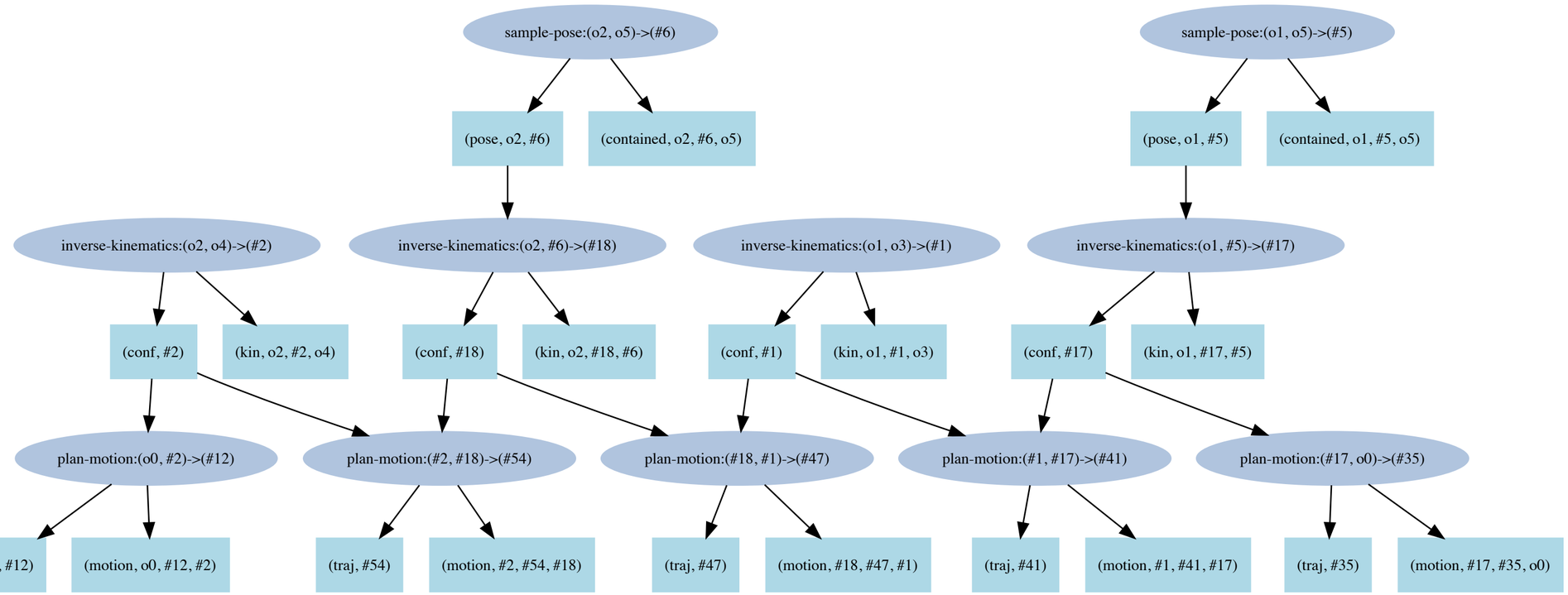
Constraint Network Optimization

- When tractable, **jointly solve** for parameter values satisfying several constraints at once
  - Useful when constraints are **strongly connected**
  - Revert to sampling-based methods upon failure
    - **Local optimization** of solution
- Preliminary work with Marc Toussaint

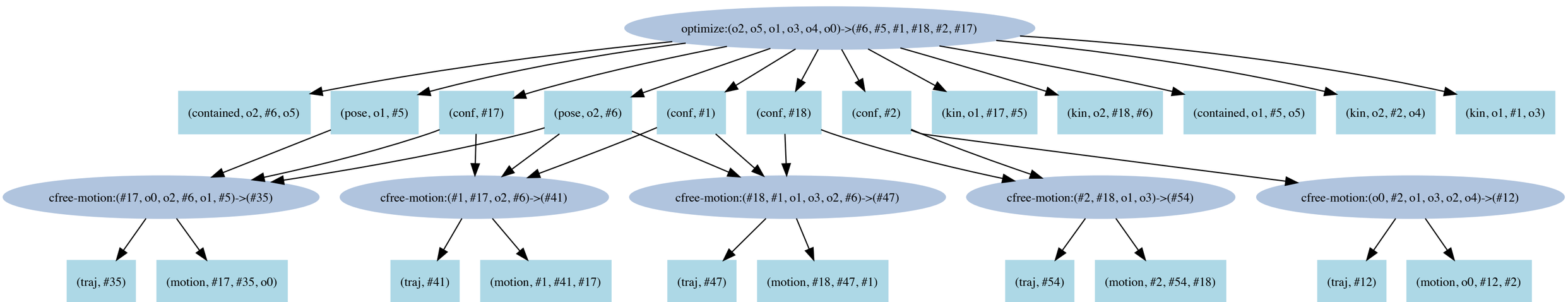




Fusing Sampler Instances



After fusing into a **placement optimizer** (Gurobi for now) and **collision-free motion planners**

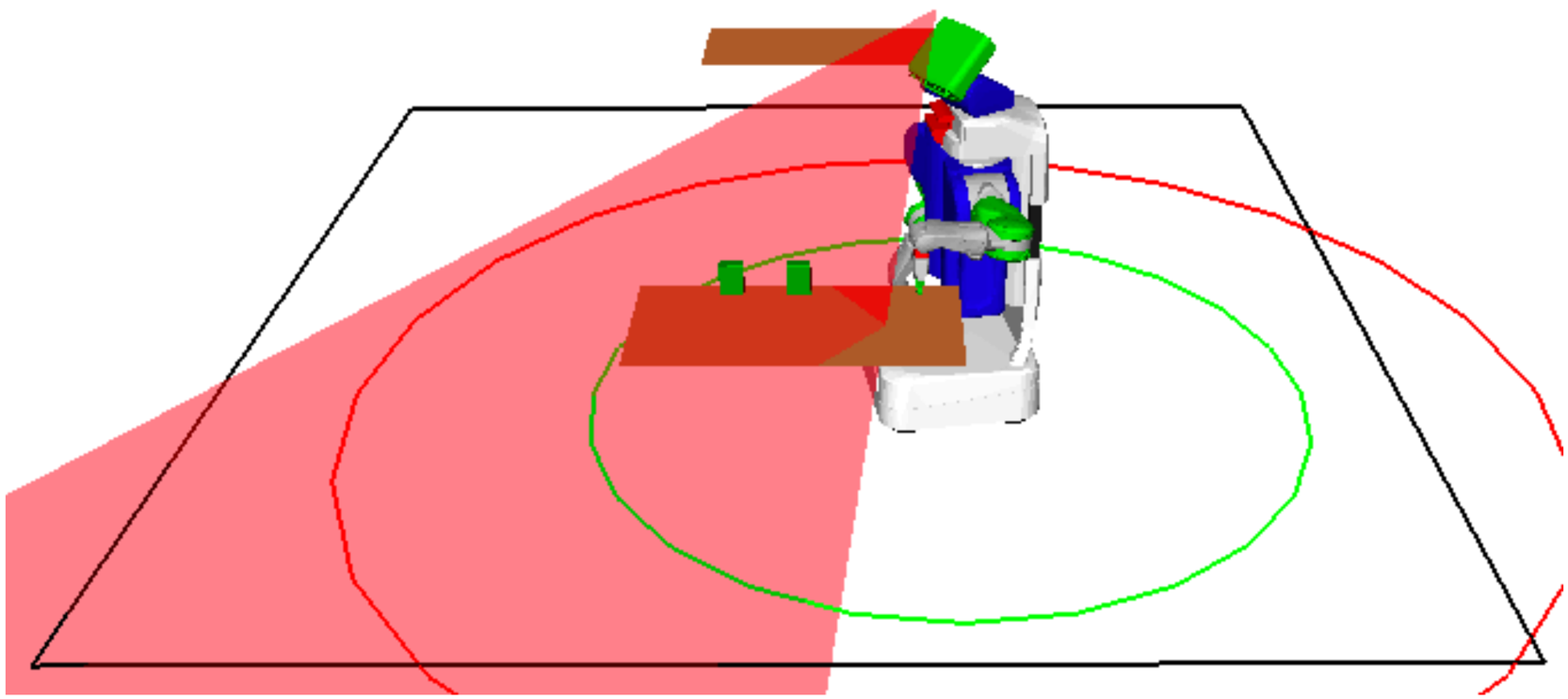


Stochastic Planning

- Approximate stochastic effects by **determinization**
- **Replan** when unanticipated effects
- **Cost-sensitive planning** to ensure induced policy makes progress towards goal
- Action cost equal to the **expected cost** under a simple model
- Induces an optimal policy for some probabilistically simple MDPs
- These MDPs are **reasonable approximations** for some problems

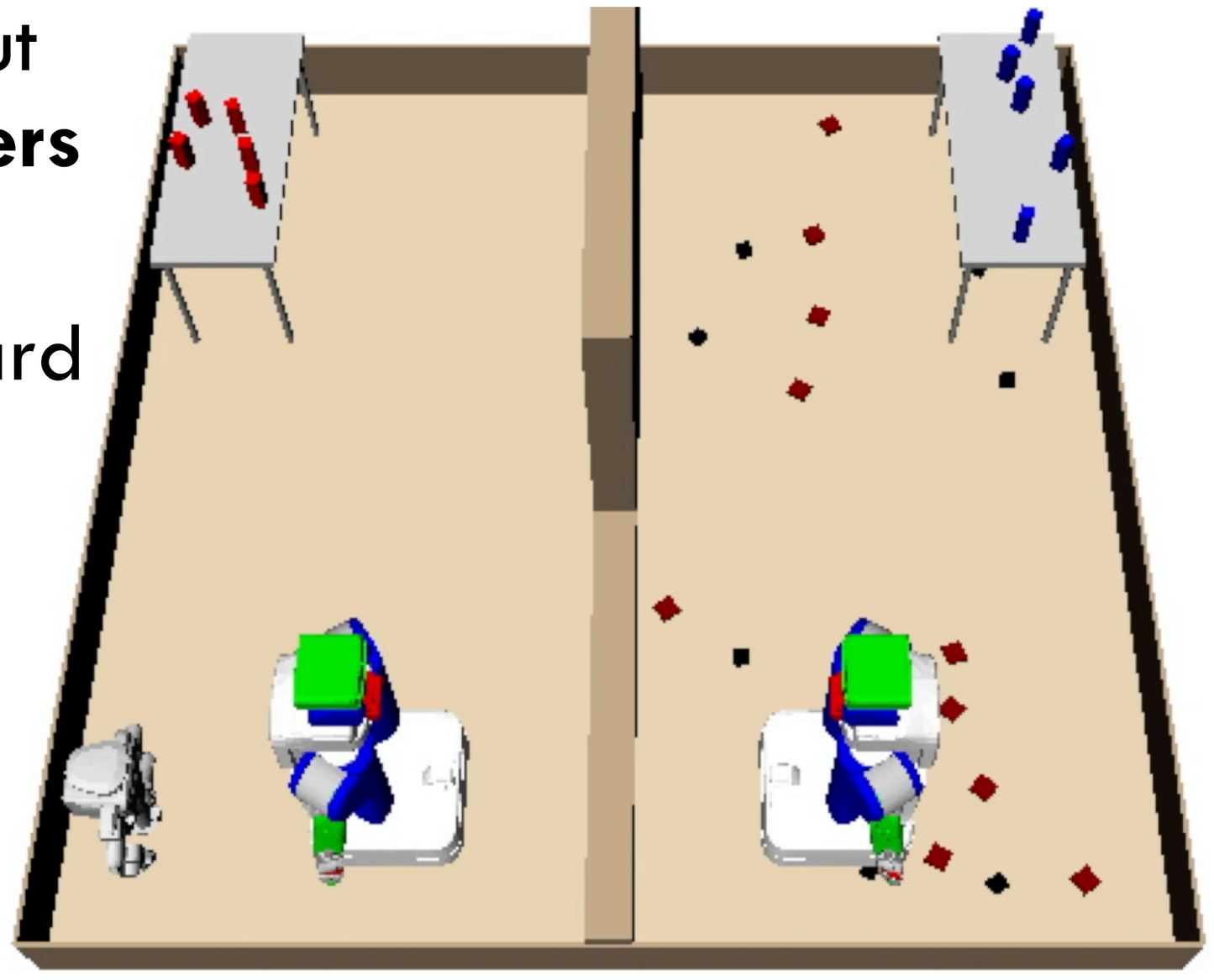
Partially Observable Planning

- Plan over distributions of states (**belief-space**)
  - Samplers **operate on probability distributions** (e.g. Multinoulli, Multivariate Gaussian, …)
  - **Exogenous observations** produce new values
- Optimistically assume helpful observations



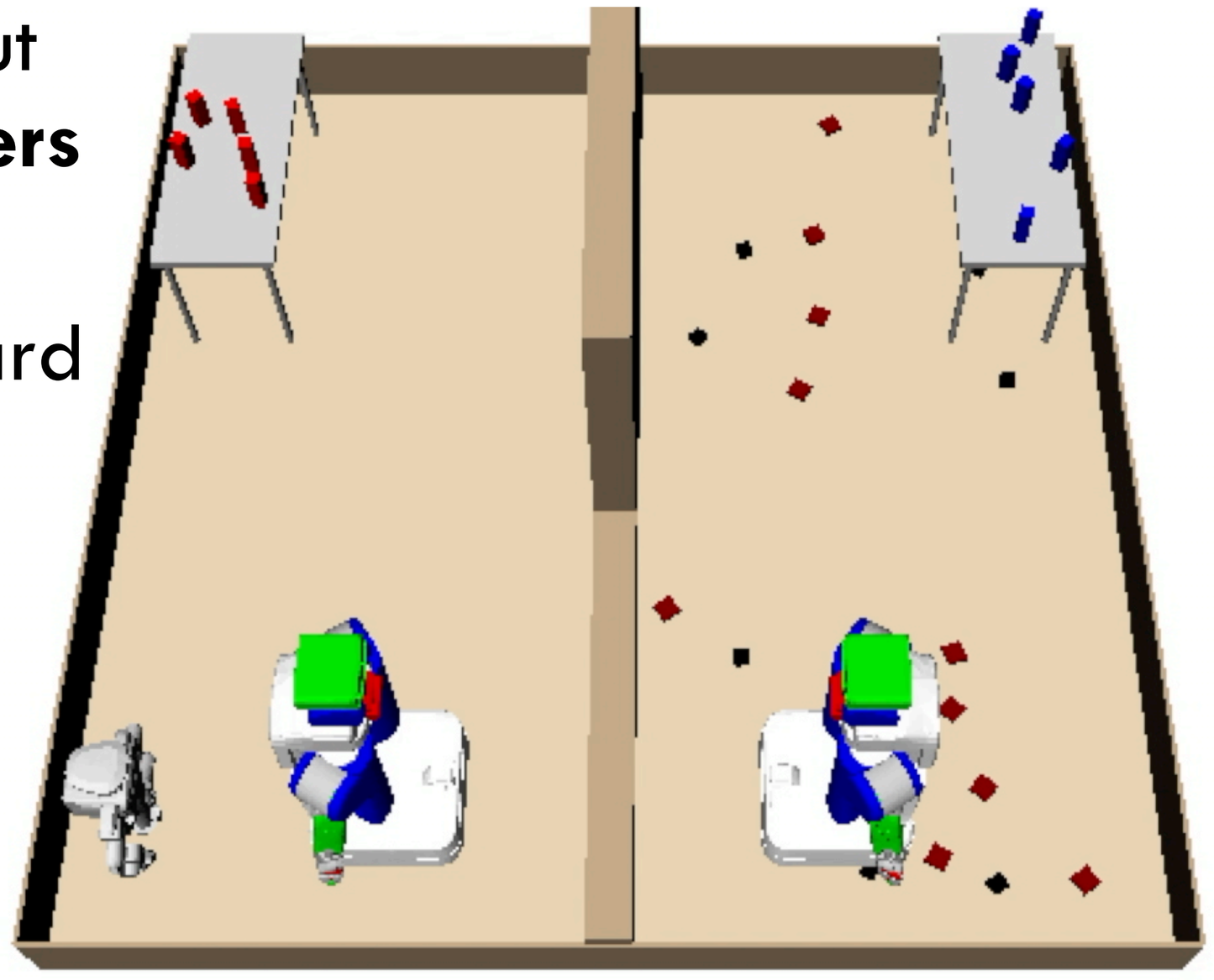
Multi-Robot Planning

- **Centralized scheduling** of a team of robots
- Similar algorithms but use **temporal planners** as search subroutine
- Temporal FastDownard
- PDDL rovers domain



Multi-Robot Planning

- **Centralized scheduling** of a team of robots
- Similar algorithms but use **temporal planners** as search subroutine
- Temporal FastDownard
- PDDL rovers domain



Planning and Execution

- **Scheduling sampler evaluations**
  - Prioritize samplers with **low overhead and low probability of success** (by solving a "meta" MDP)
  - **Estimate** overhead and probability of success per sampler

- When replanning, many sampler evaluations may not be used
  - **Defer sampler evaluations** by scheduling samplers and actions together
  - Plan to sample a real value in the future

Conclusion

- **General-purpose framework** for exposing **factoring** in discrete-time **hybrid systems**

- Techniques for solving a subclass of these systems using **sampling**

- **Domain-independent algorithms** that operate on **conditional samplers** as blackboxes

- Future directions include **learning samplers, cost-sensitive planning,** and **planning & execution**

Questions?

Hierarchy

- **Hierarchical action specifications**
  - Assumption that refinement likely possible
  - Provide search guidance to a planner
  - Can postpone planning in some cases when planning and executing

- Focused algorithm effective when few things to achieve

PR2 Demonstration

- Visual object detection for coarse pose estimates
  - Tensorflow RCNN
- Point cloud registration for fine pose estimates
  - PCL
- Occupancy grid for collision checking
  - Octomap

Robotic Fabrication

- Preliminary work with Yijiang Huang and Caitlin Mueller in the architecture department