Task and Motion Planning

Dylan Hadfield-Menell

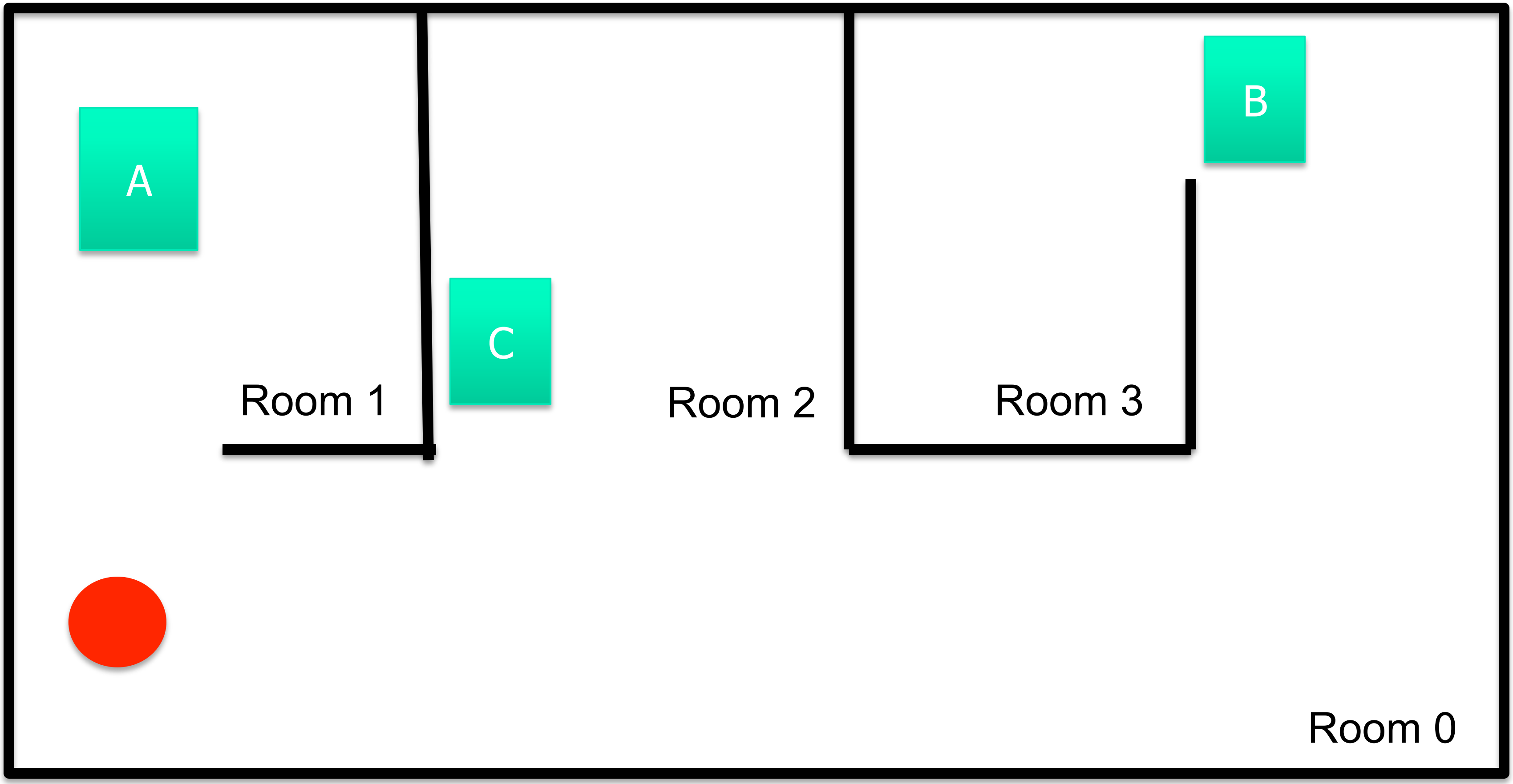
UC Berkeley CS 287 Guest Lecture

Planning for Complex Tasks

Outline

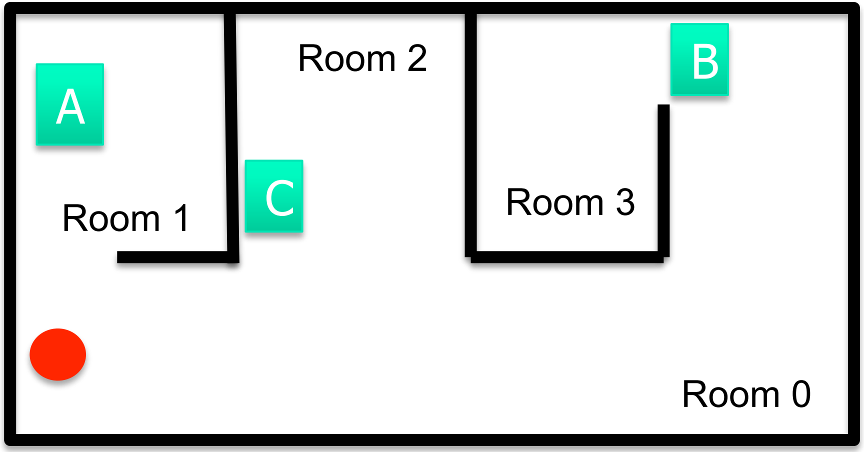
- Task Planning
    - Formulation
    - Fast-Forward
- Task and Motion Planning
    - Forward Search
    - Plan Skeletons
    - Extension: Partial observability

Example Domain

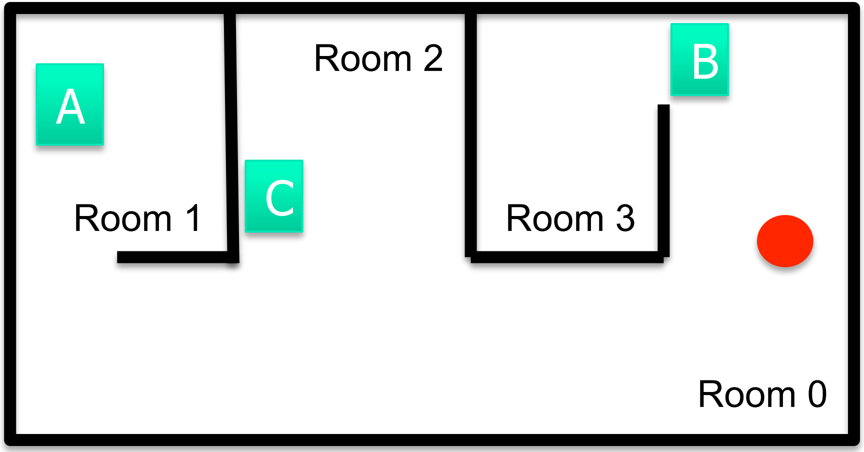


Motion Planning

- Initial State:

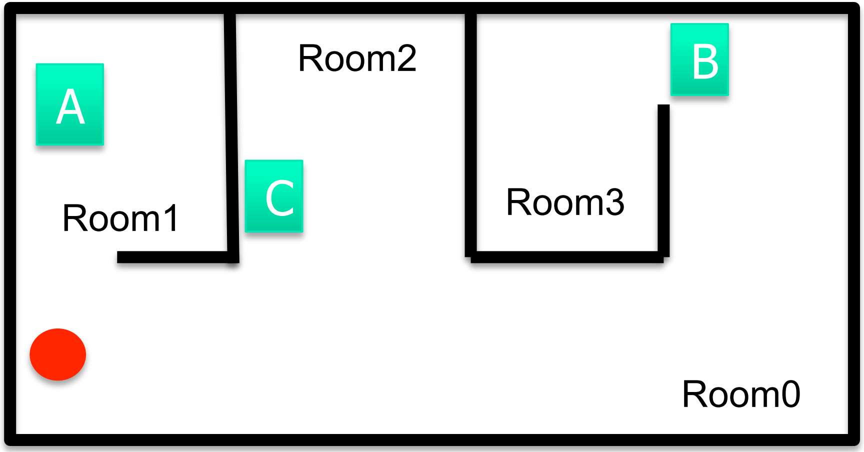


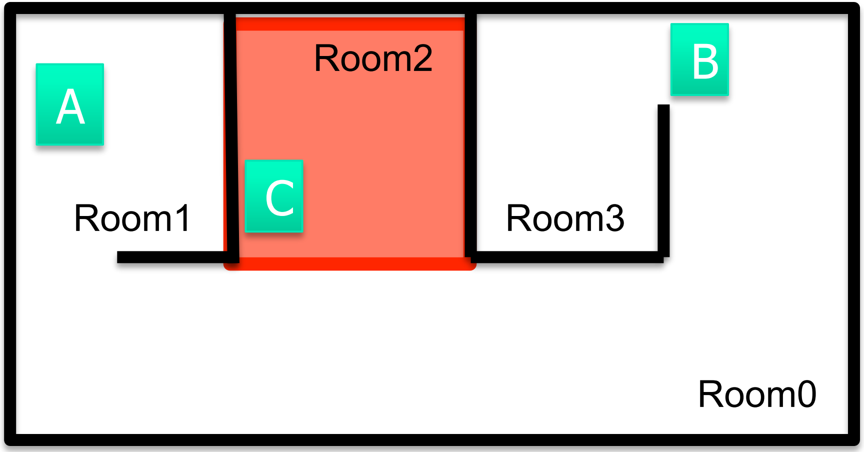
- Goal State:
  - Target robot pose



Motion Planning++

- Initial State:

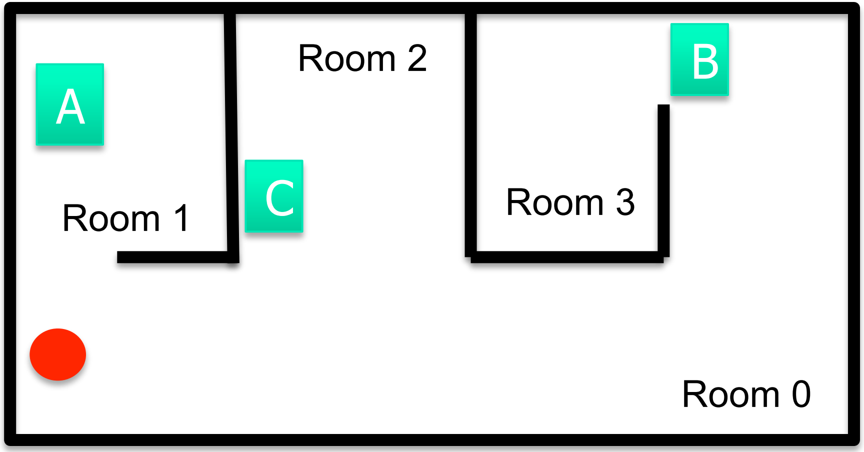




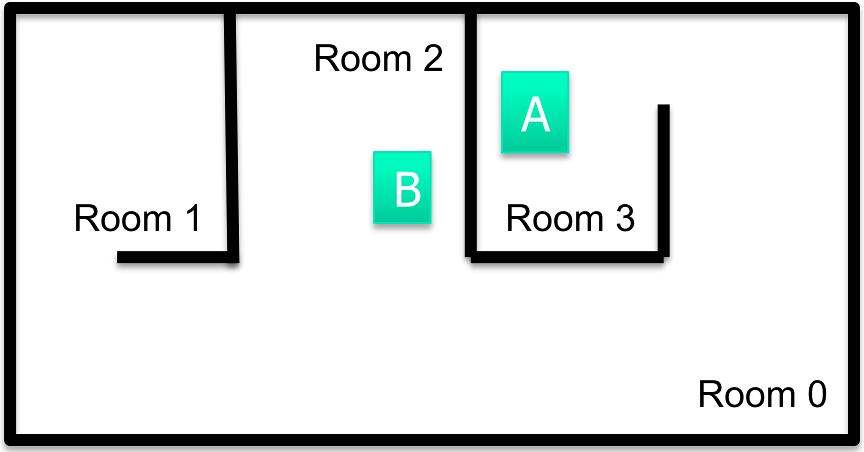
- Goal State:
  - Set of Robot Configurations
  - In(Robot, Room2)
  - In(Robot, Room3)?

Task and Motion Planning

- Initial State:



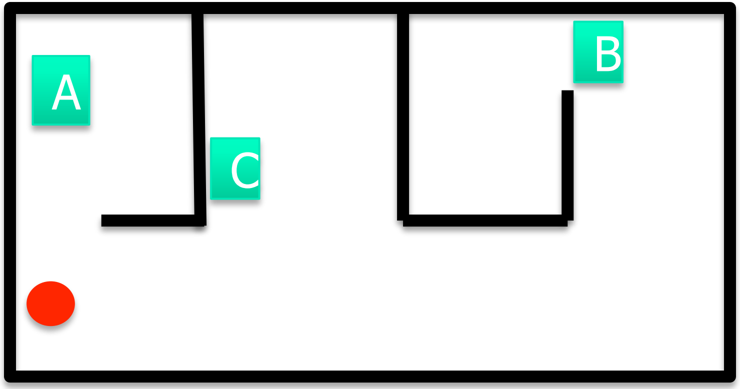
- Goal State:
  - In(A, Room3) ^ In(B, Room2)



Early Robotics: Shakey the Robot

Task Planning: State Representation

- Represent state of the world as list of true properties



In(Robot, R0)
In(A, R1)
In(C, R2)
In(B, R0)
Holding(Robot, None)
Blocks(B, R0, R3)
.
.
.

Task Planning: Action Representation

- An operator is a defined by 3 attributes
  - Name
    - Identifier for action
  - Preconditions
    - List of fluents that must be true in order to take action
  - Effects
    - Add list: fluents that become true after the action
    - Delete list: fluents that become false after the action

Move(R0, R1)

Preconditions
In(robot, R0)
Connected(R0, R1)
~Blocks(A, R0, R1)
~Blocks(B, R0, R1)
~Blocks(C, R0, R1)

Effects
In(robot, R1)
~In(robot, R0)

Task Planning: More Actions

Pick(A, R0)

Preconditions
- Holding(None)
- In(A, R0)
- In(robot, R0)

Effects
- ~Holding(None)
- Holding(A)

Clear(B, R0, R1)

Preconditions
- Blocks(B, R0, R1)
- In(robot, R0)
- Holding(None)

Effects
- ~Blocks(B, R0, R1)

MoveHolding(A, R0, R1)

Preconditions
- In(robot, R0)
- Holding(A)
- Connected(R0, R1)
- ~Blocks(A, R0, R1)
- ~Blocks(B, R0, R1)
- ~Blocks(C, R0, R1)

Effects
- In(robot, R1)
- ~In(robot, R0)
- In(A, R1)
- ~In(A, R0)

Planning Domain Description Language

- Standardized format to represent planning problems
- Used for International Planning Competitions
  - Lots of published code that can read this representation
- Domain file defines
  - Fluents, object types, operator schemas
- Problem file defines
  - Objects, Initial state, Goal condition

Example PDDL Domain

```
(define (domain gripper-strips)
  (:predicates (room ?r) (ball ?b)
               (gripper ?g)
               (at-robby ?r)
               (at ?b ?r)  (free ?g)
               (carry ?o ?g))
(:action move
   :parameters (?from ?to)
   :precondition (and (room ?from)
                 (room ?to)
                 (at-robby ?from))
   :effect (and (at-robby ?to)
             (not (at-robby ?from))))
```

Example from Manuela Veloso

Example PDDL Domain (cont'd)

```
(:action pick
   :parameters (?obj ?room ?gripper)
   :precondition (and (ball ?obj)
                      (room ?room)
                      (gripper ?gripper)
                      (at ?obj ?room)
                      (at-robby ?room)
                     (free ?gripper))
  :effect (and (carry ?obj ?gripper)
            (not (at ?obj ?room)) (not
(free ?gripper))))
```

```
(:action drop
  :parameters (?obj ?room ?gripper)
  :precondition (and (ball ?obj)
                     (room ?room)
                      (gripper ?gripper)
                      (carry ?obj ?gripper)
                      (at-robby ?room))
  :effect (and (at ?obj ?room)
            (free ?gripper)
            (not (carry ?obj ?gripper)))))
```

Example from Manuela Veloso

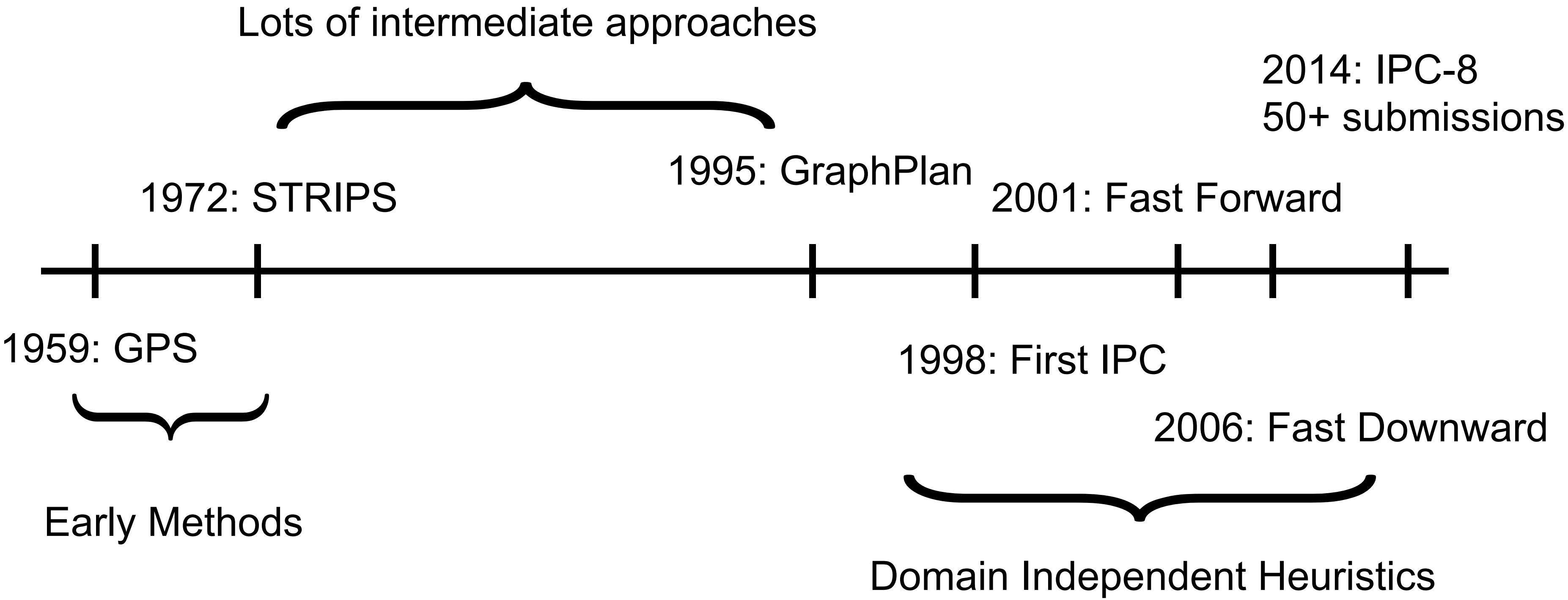
Example PDDL Problem

```
(define (problem strips-gripper2)
   (:domain gripper-strips)
   (:objects rooma roomb ball1 ball2 left right)
   (:init (room rooma)  (room roomb)
           (ball ball1)  (ball ball2)
           (gripper left) (gripper right)
           (at-robby rooma)
           (free left)   (free right)
           (at ball1 rooma)  (at ball2 rooma))
  (:goal (at ball1 roomb)))
```

```
Solution:
   pick(ball1 rooma left)
   move(rooma roomb)
   drop(ball1 roomb left)
```

Example from Manuela Veloso

Algorithms for Task Planning



Lots of intermediate approaches

2014: IPC-8
50+ submissions

1995: GraphPlan

1972: STRIPS

2001: Fast Forward

1959: GPS

1998: First IPC

2006: Fast Downward

Early Methods

Domain Independent Heuristics

Not to scale

Planning Graph [Blum & Furst ‘95]

- Preprocessing Step before planning
- Can reveal natural structure in problem
- Compute over-approximation of reachable set of literals

Planning Graph [Blum & Furst ‘95]

```
$L_0$ ← all facts true in initial state
$t \leftarrow 0$
While $goal \notin L_t$
  $L_t$ ← facts from $L_{t-1}$
  For each action with $pre(a) \in L_{t-1}$
      $L_t = L_t \cup eff(a)$
  $t \leftarrow t + 1$
```

Theorem: L_t is a superset of reachable set of fluents for plans of length t

Fast-Forward [Hoffmann 2001]

- Early use of plan graphs analyzed the plan graph to extract a sequence of actions
- Fast-Forward: use the length of the planning graph as a heuristic inside of a forward search
  - Actually use relaxed planning graph, which ignores delete effects
  - Some modifications to handle very slow heuristic computation

Fast-Forward [Hoffmann 2001]

```
$Q \leftarrow PriorityQueue()$
$Q.push(init, 0)$
While goal not found
    $s \leftarrow Q.pop()$
    $pg \leftarrow RelaxPlanGraph(s, goal)$
    $for\ c\ in\ s.children$
        $Q.push(c, len(pg))$
```

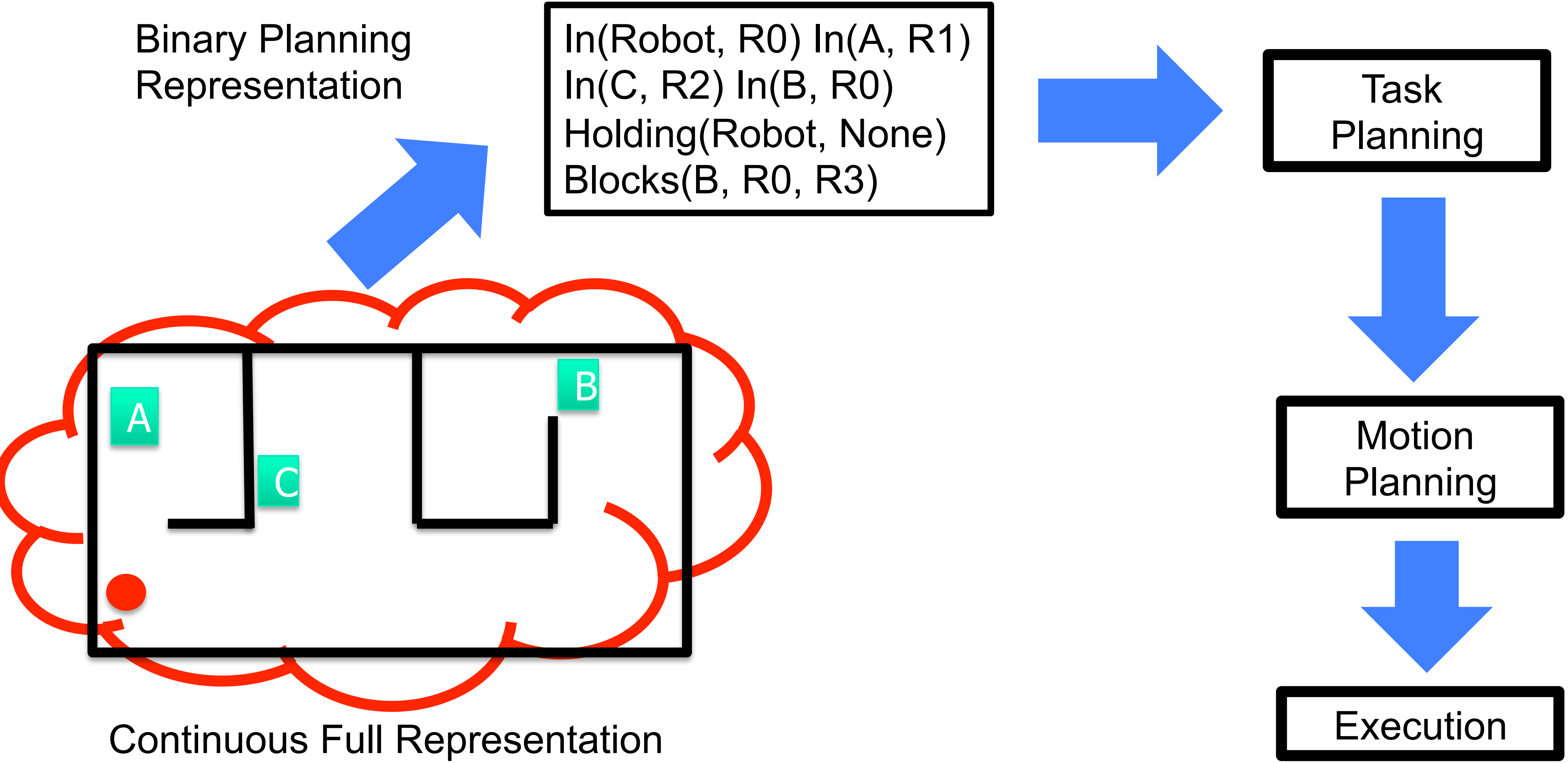
Fast-Forward Details

- Enforced hill climbing
  - Greedy search + breadth-first search to account for plateaus
- Push children with heuristic evaluated on parent
  - 1 heuristic evaluation/expansion
  - Alternative is 1 heuristic evaluation/child
- Helpful actions
  - When planning graph terminates, we can extract a plan with simultaneous actions
  - Search those actions first

Task Planning Summary

- Binary State Representation
  - Properties of the world that change over time
- Actions defined by preconditions and effects
- State-of-the-art relies on heuristic forward search with domain independent heuristics

Task Planning for Robots (the hope)



Binary Planning Representation

In(Robot, R0) In(A, R1)
In(C, R2) In(B, R0)
Holding(Robot, None)
Blocks(B, R0, R3)

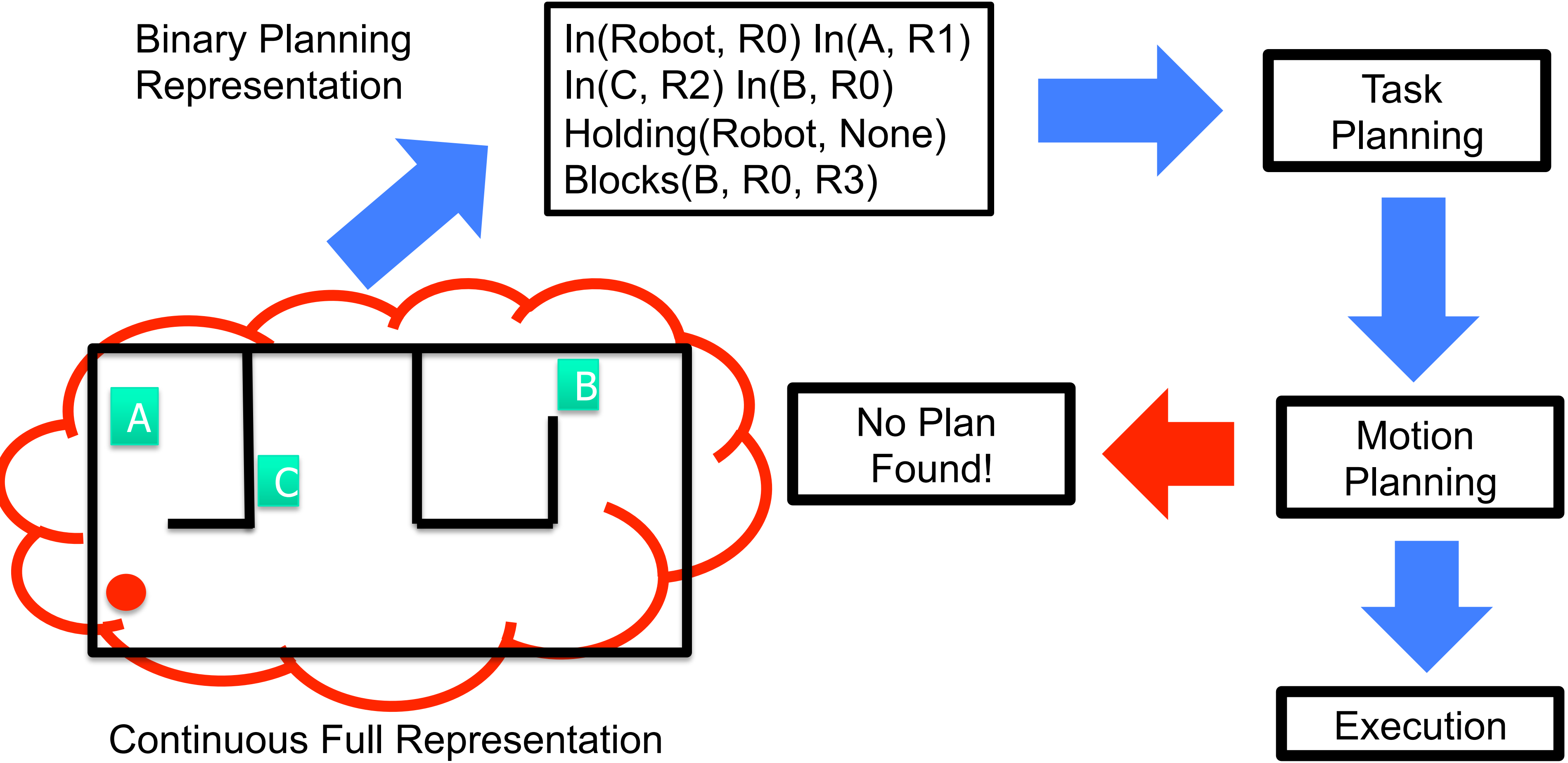
Task Planning

Motion Planning

Execution

Continuous Full Representation

Task Planning for Robots (the reality)



Binary Planning Representation

In(Robot, R0) In(A, R1)
In(C, R2) In(B, R0)
Holding(Robot, None)
Blocks(B, R0, R3)

Task Planning

Motion Planning

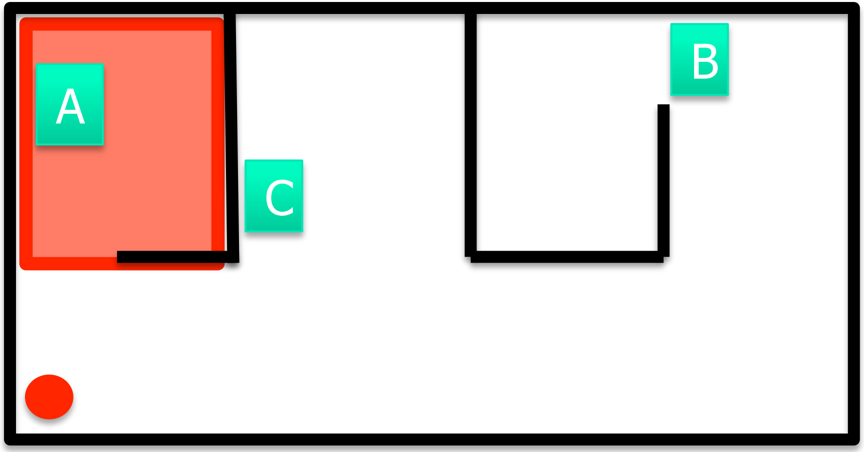
Execution

No Plan Found!

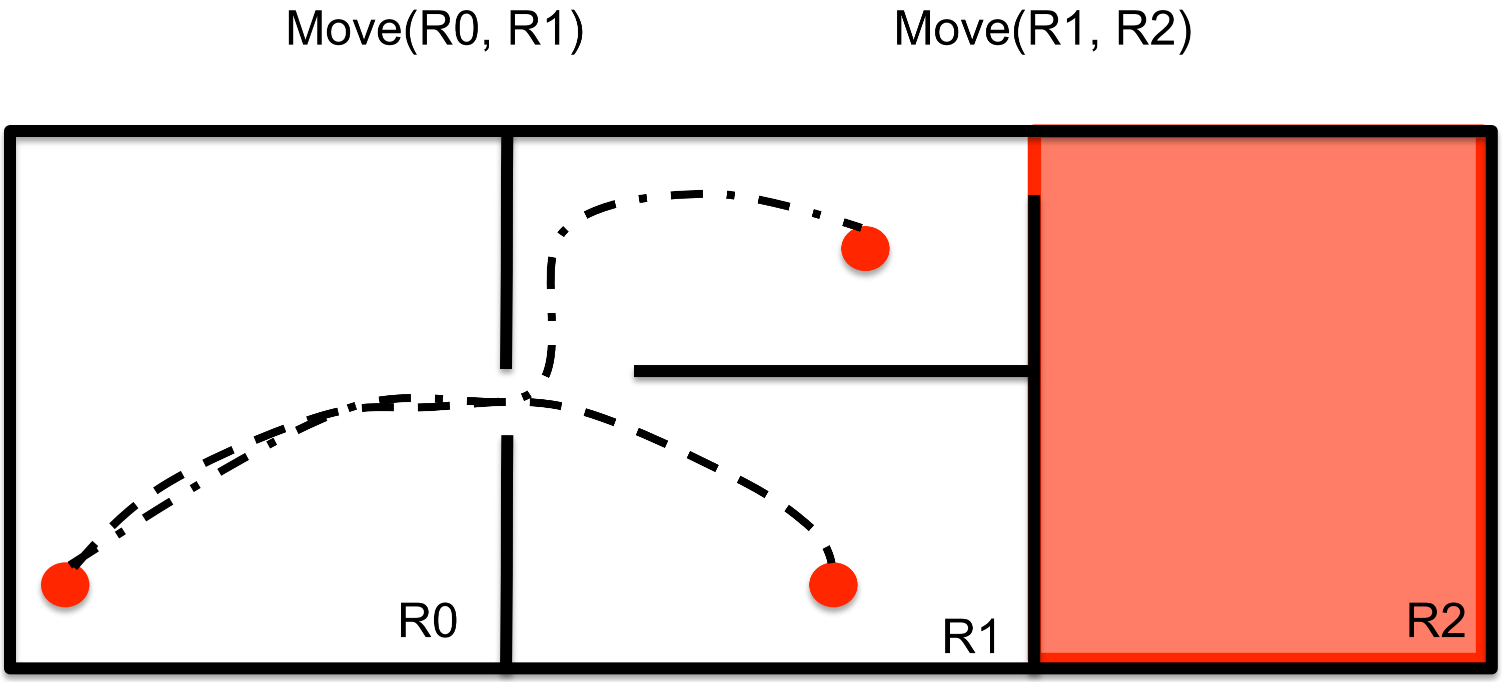
Continuous Full Representation

Executing a Task Plan

- Each high level action encodes a motion planning problem
- Ex. Move(R0, R1)
  - Initial State: Current robot pose
  - Goal State: anything in R1
- Motion plan each step in sequence
  - Issue: dependency between intermediate steps of plan



Dependency for intermediate states

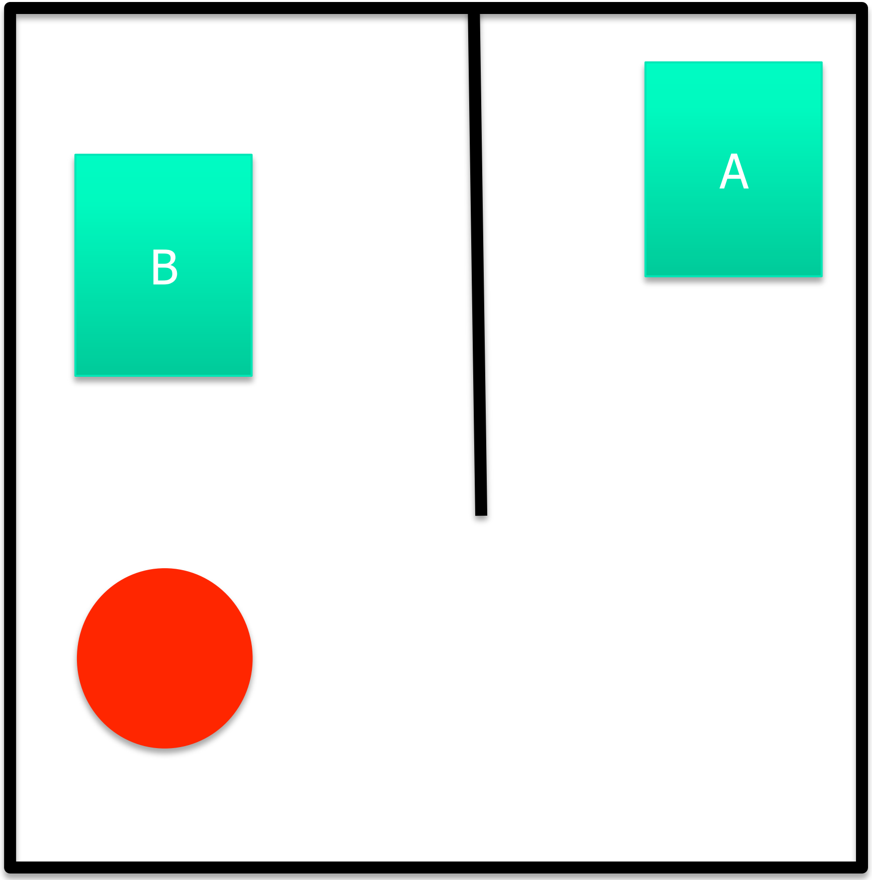


Solution: Try several intermediate poses for each action

What if the task plan itself is wrong?

A Continuous Representation

- Goal: Holding(robot, A)
- High-Level Actions
  - Grasp(robot, r_pose, obj, o_pose, grasp)
  - Move(robot, pose1, pose2)
  - Place(robot, r_pose, obj, grasp, obj_pose)
- Grasps, poses, and locations are all continuous



A Continuous Operator

Continuous parameters

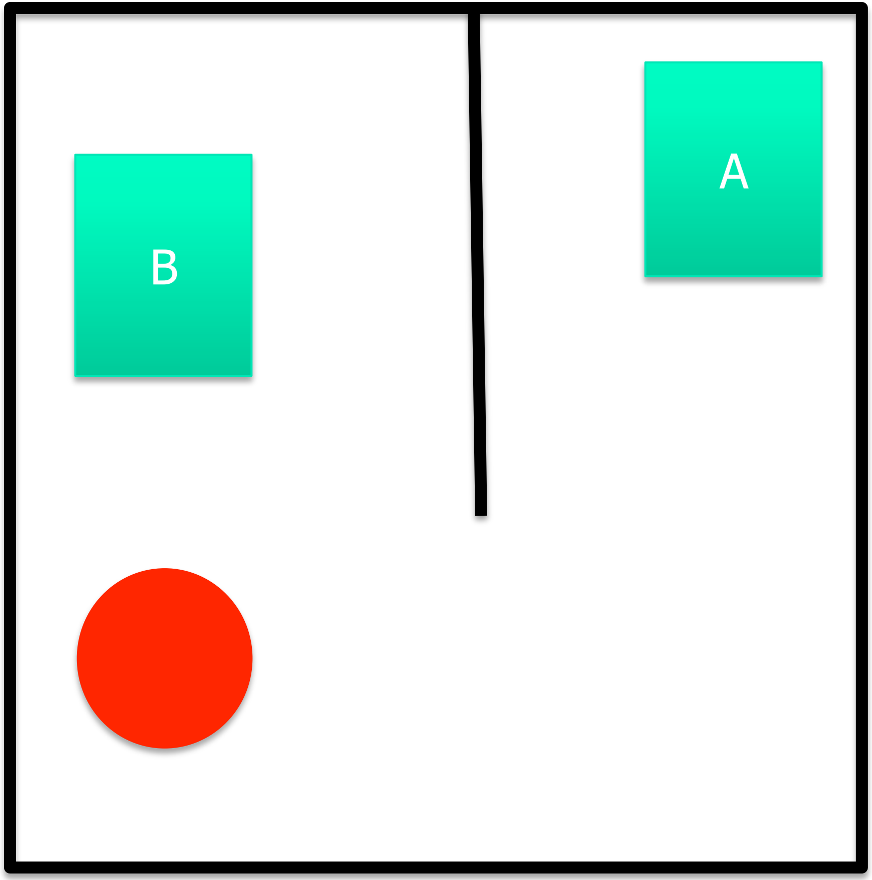
Grasp(robot, r_pose, obj, o_pose, grasp)

Preconditions:

- GraspPose(r_pose, o_pose, grasp)
- At(robot, r_pose)
- At(obj, o_pose)
- Holding(robot, None)

Effects:

- ~At(obj, o_pose)
- Holding(robot, obj)
- ~Holding(robot, None)
- $\forall$ p1, p2 ~Obstructs(obj, p1, p2)



Task and Motion Planning Approaches

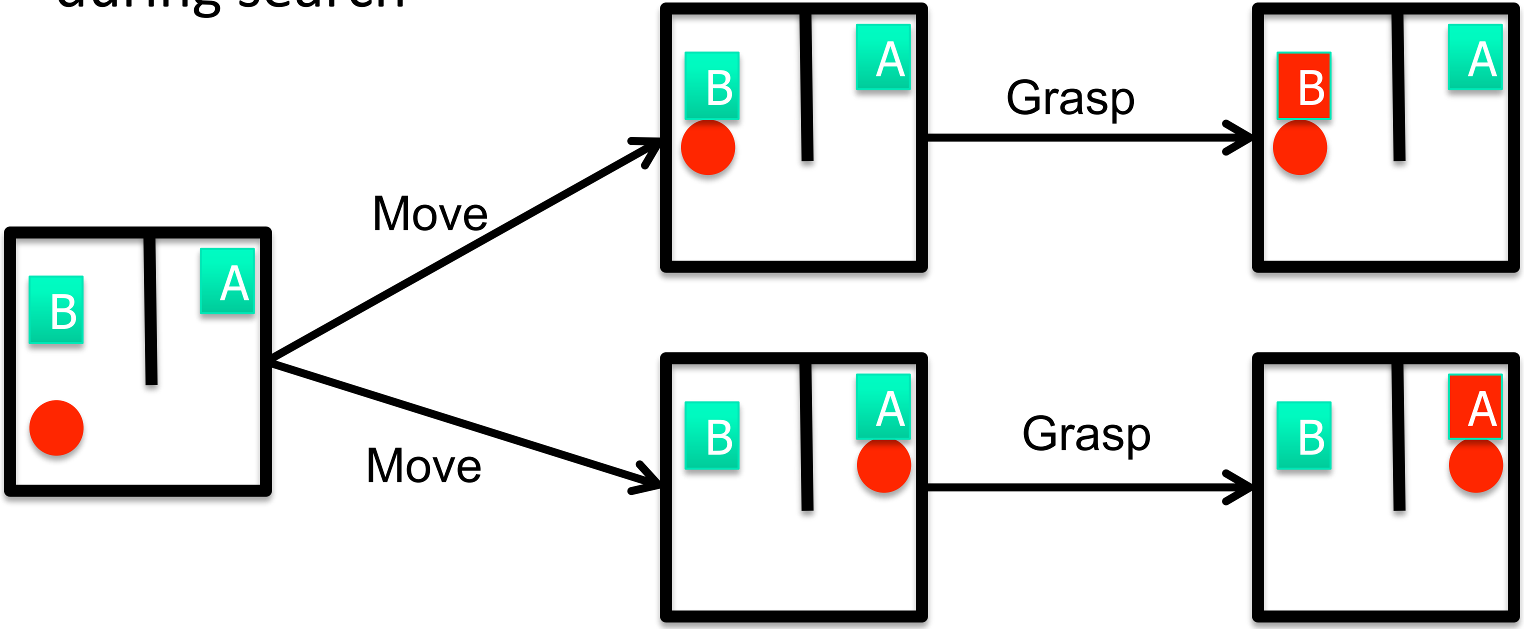
- Forward Search
  - Gravot, Fabien, Stephane Cambon, and Rachid Alami. "aSyMov: a planner that deals with intricate symbolic and geometric problems." *Robotics Research. The Eleventh International Symposium*. Springer Berlin Heidelberg, 2005.
  - Dornhege, Christian, et al. "Semantic attachments for domain-independent planning systems." *Towards Service Robots for Everyday Environments*. Springer Berlin Heidelberg, 2012.
  - Garrett, Caelan Reed, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. ”FFROB: An efficient heuristic for task and motion planning." *Algorithmic Foundations of Robotics XI*. Springer International Publishing, 2015. 179-195.
- Hierarchical TAMP
  - Kaelbling, Leslie Pack, and Tomás Lozano-Pérez. "Hierarchical task and motion planning in the now." *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011.
- Plan Skeleton
  - Srivastava, Siddarth, et al. "Combined task and motion planning through an extensible planner-independent interface layer." *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014.
  - Lozano-Pérez, Tomás, and Leslie Pack Kaelbling. "A constraint-based method for solving sequential manipulation planning problems." *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, 2014.
  - Toussaint, Marc. "Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning." 2015.

Strawman TAMP Algorithm: Discretize

- Replace each continuous value with a set of discrete options
- Compute all relevant properties
- Run your favorite task planner
  - Now it sets intermediate poses as well
- Issues?
  - Curse of dimensionality
  - Lots of irrelevant motion planning

TAMP via Forward Search

- Main idea: lazily discretize values and compute properties during search



Forward Search

```
$Q \leftarrow PriorityQueue()$
$Q.push(init, 0)$
While goal not found
    $s \leftarrow Q.pop()$
    $pg \leftarrow RelaxPlanGraph(s, goal)$
    for each applicable action, $a$ s.t. $pre(a) \in s$
        $children \leftarrow Discretize(s, a)$
        for $c \in children$
            $Q.push(c, h(c))$
```

Challenge: What goes here?

Forward Search Challenges

- Node expansions are very slow
    - >95% of running time is spent answering motion planning queries
    - Efficient caching strategies can help a lot
    - [aSyMov '05] interleave PRM iterations with search iterations
- Useful heuristic information
    - Obtaining useful heuristic information has been a primary bottleneck
    - Recent work investigates efficient computation of plan graph heuristic [Garrett '15]

Task and Motion Planning Approaches

- Forward Search
    - Gravot, Fabien, Stephane Cambon, and Rachid Alami. "aSyMov: a planner that deals with intricate symbolic and geometric problems." *Robotics Research. The Eleventh International Symposium*. Springer Berlin Heidelberg, 2005.
    - Dornhege, Christian, et al. "Semantic attachments for domain-independent planning systems." *Towards Service Robots for Everyday Environments*. Springer Berlin Heidelberg, 2012.
    - Garrett, Caelan Reed, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. ”FFROB: An efficient heuristic for task and motion planning." *Algorithmic Foundations of Robotics XI*. Springer International Publishing, 2015. 179-195.
- Hierarchical TAMP
    - Kaelbling, Leslie Pack, and Tomás Lozano-Pérez. "Hierarchical task and motion planning in the now." *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011.
- Plan Skeleton
    - Srivastava, Siddarth, et al. "Combined task and motion planning through an extensible planner-independent interface layer." *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014.
    - Lozano-Pérez, Tomás, and Leslie Pack Kaelbling. "A constraint-based method for solving sequential manipulation planning problems." *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, 2014.
    - Toussaint, Marc. "Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning." 2015.

Plan Skeleton Methods

- Initially plan with abstract representation that ignores continuous dynamics
- Output can be thought of as a continuous constraint satisfaction problem
  - Preconditions → constraints
- Algorithm sketch
  - Generate task plan
  - Attempt to solve CSP
  - If failure, generate new plan

A Continuous Operator

Continuous parameters

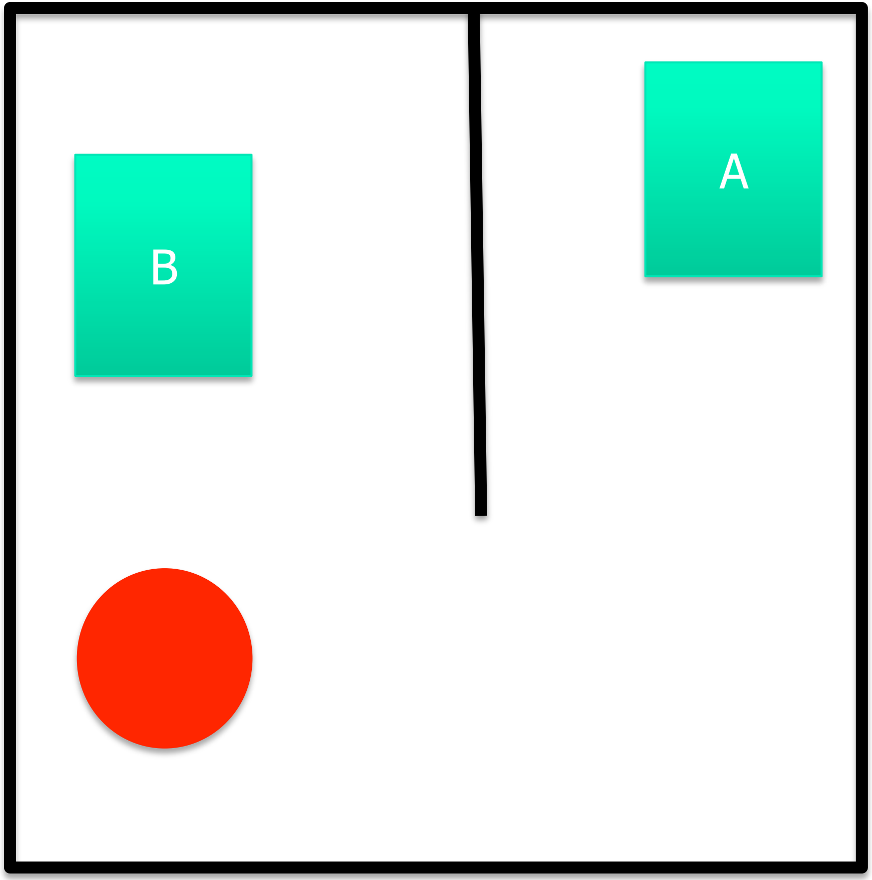
Grasp(robot, r_pose, obj, o_pose, grasp)

Preconditions:

GraspPose(r_pose, o_pose, grasp)
At(robot, r_pose)
At(obj, o_pose)
Holding(robot, None)

Effects:

~At(obj, o_pose)
Holding(robot, obj)
~Holding(robot, None)
$\forall$ p1, p2 ~Obstructs(obj, p1, p2)



Poses → Pose References

- Replace continuous values with symbolic references
- Leave these values *uninstantiated* during task planning
- *Refine* task plan to pick values for continuous parameters

Symbols

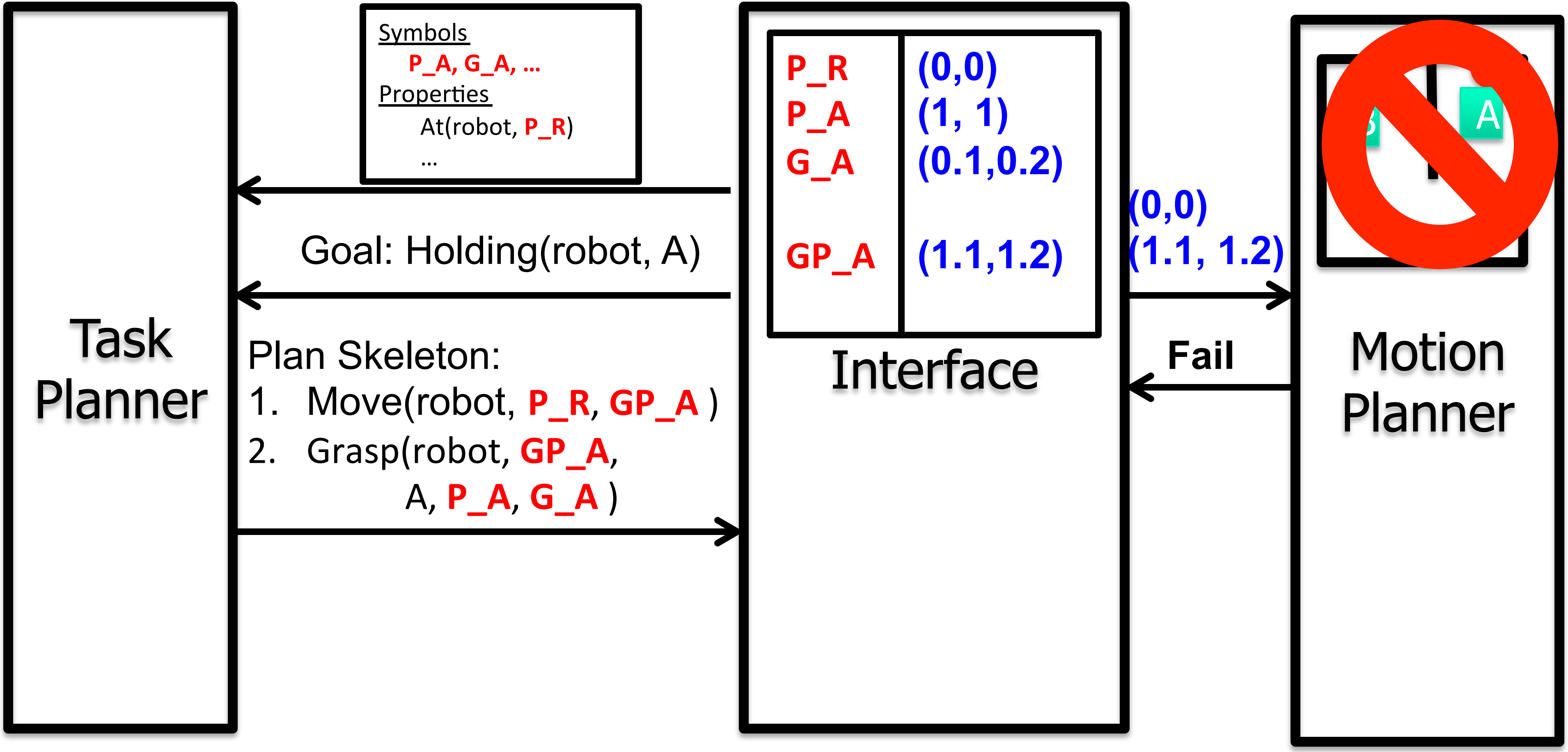
P_A : "object pose where A is"
type: object pose
G_A : "grasp we can use for A"
type: grasp
GP_A : "pose with a valid grasp for A"
type: robot pose
P_R : "initial robot pose"
type: robot pose

Properties

At(robot, **P_R**)
At (A, **P_A**)
GraspPose(**GP_A**, **P_A**, **G_A**)

[Srivastava, Siddharth, et al. "Combined task and motion planning through an extensible planner-independent interface layer." ICRA, 2014.]

Planning with an Interface



Task Planner

Symbols
P_A, G_A, …
Properties
At(robot, P_R)
…

Goal: Holding(robot, A)

Plan Skeleton:

1. Move(robot, P_R, GP_A)
2. Grasp(robot, GP_A, A, P_A, G_A)

P_R	(0,0)
P_A	(1, 1)
G_A	(0.1,0.2)
GP_A	(1.1,1.2)

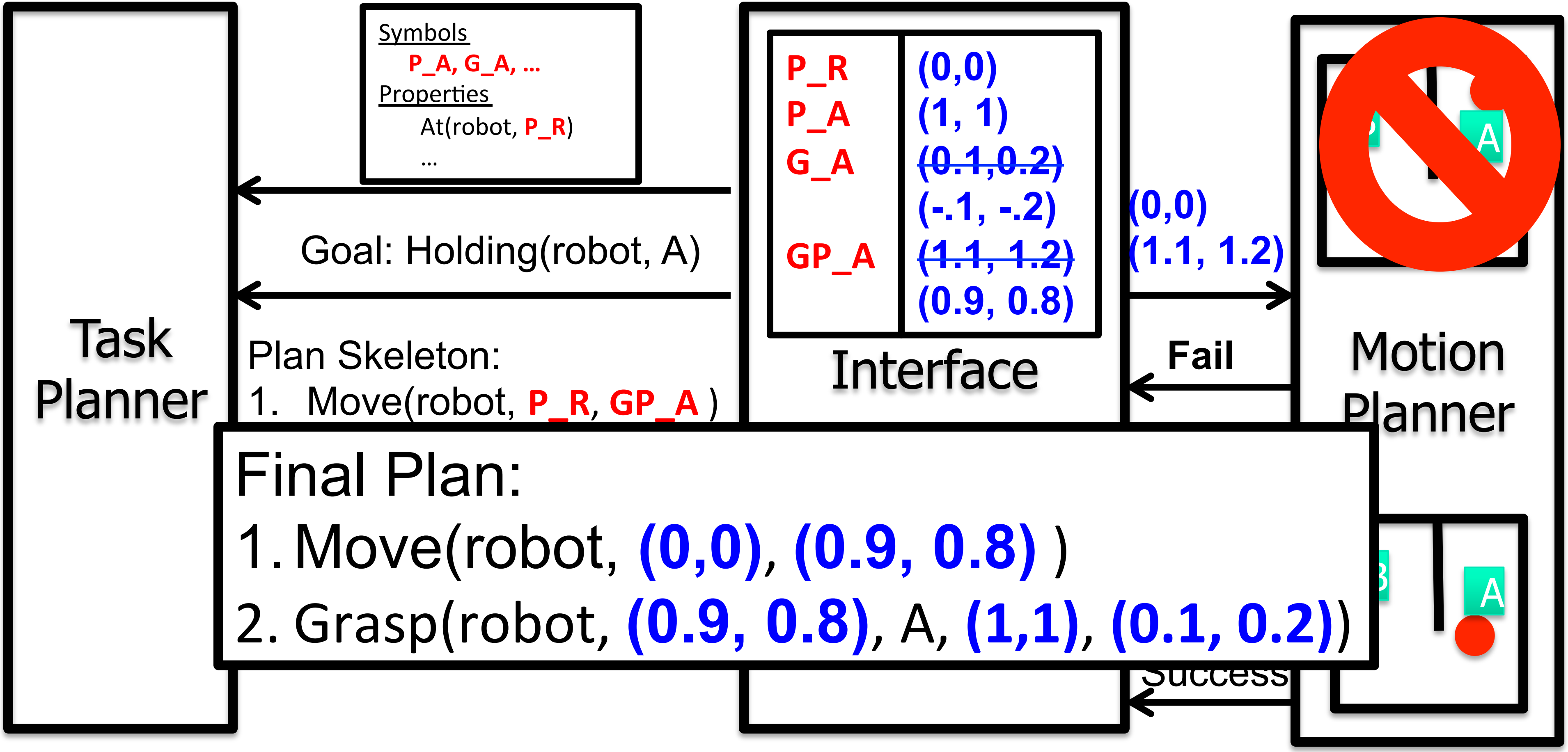
Interface

(0,0)
(1.1, 1.2)

Fail

Motion Planner

Planning with an Interface



Task Planner

Symbols
P_A, G_A, …
Properties
At(robot, **P_R**)
…

Goal: Holding(robot, A)

Plan Skeleton:
1. Move(robot, **P_R**, **GP_A**)

P_R	**(0,0)**
P_A	**(1, 1)**
G_A	~~**(0.1,0.2)**~~ **(-.1, -.2)**
GP_A	~~**(1.1, 1.2)**~~ **(0.9, 0.8)**

Interface

(0,0)
(1.1, 1.2)

Fail

Success

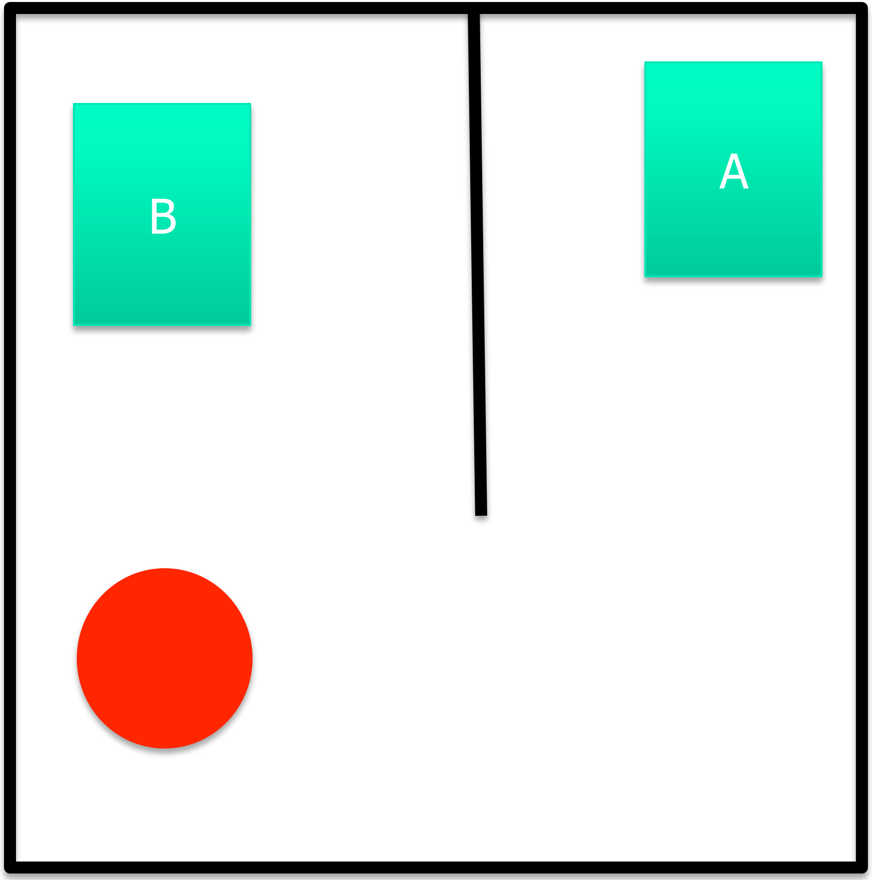
Motion Planner

Final Plan:

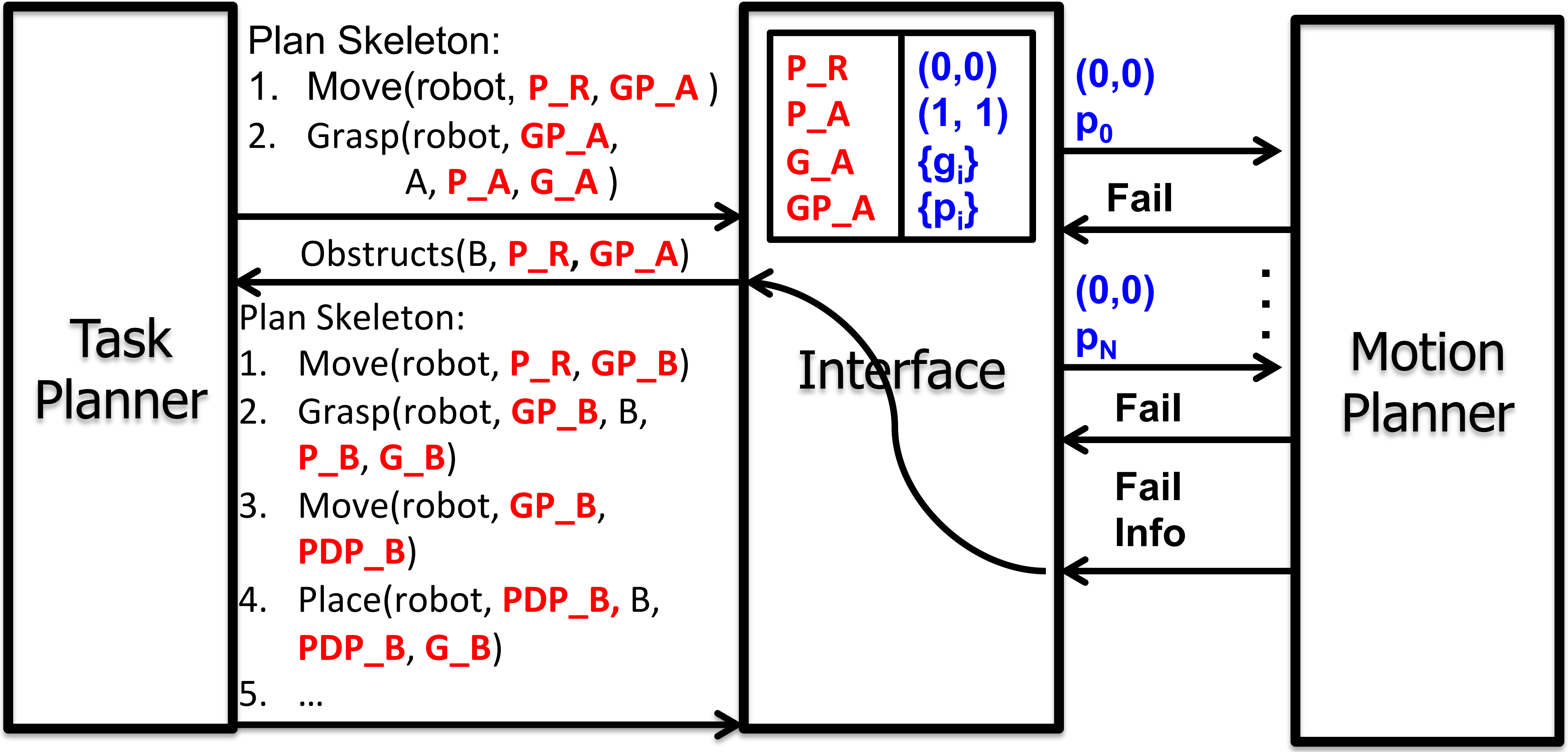
1. Move(robot, **(0,0)**, **(0.9, 0.8)**)
2. Grasp(robot, **(0.9, 0.8)**, A, **(1,1)**, **(0.1, 0.2)**)

Error Propagation

- What do we lose with symbol references?
  - High level can't know anything that depends on specific values of parameters
  - E.g. what if B blocks A
- Solution:
  - Interface queries motion planner to determine failure
  - Updates high level



Error Propagation



Task Planner

Plan Skeleton:

1. Move(robot, **P_R**, **GP_A**)
2. Grasp(robot, **GP_A**, A, **P_A**, **G_A**)

Obstructs(B, **P_R**, **GP_A**)

Plan Skeleton:

1. Move(robot, **P_R**, **GP_B**)
2. Grasp(robot, **GP_B**, B, **P_B**, **G_B**)
3. Move(robot, **GP_B**, **PDP_B**)
4. Place(robot, **PDP_B**, B, **PDP_B**, **G_B**)
5. …

Interface

P_R	**(0,0)**
P_A	**(1, 1)**
G_A	**$\{g_i\}$**
GP_A	**$\{p_i\}$**

(0,0)
p_0

Fail

(0,0)
p_N

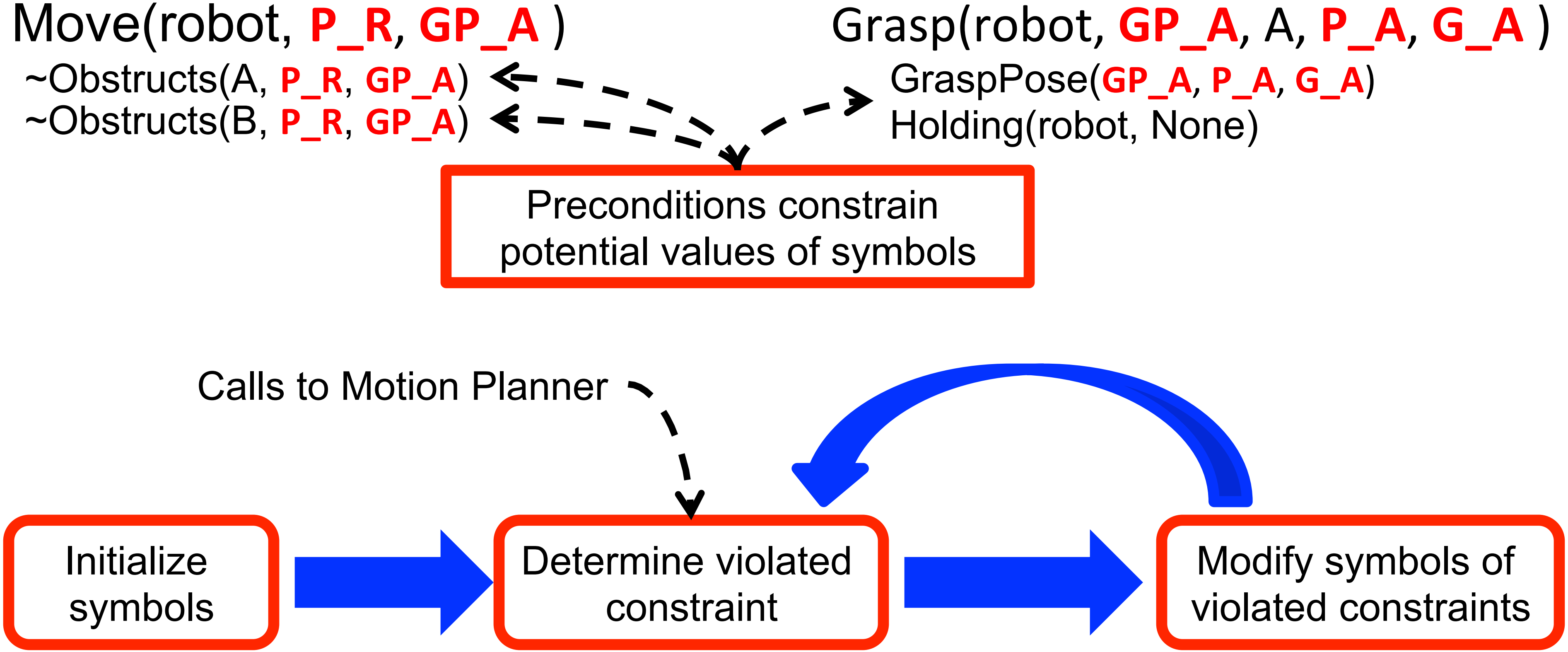
Fail

Fail Info

Motion Planner

Plan Refinement via Local Search

Plan Skeleton



Move(robot, **P_R**, **GP_A**)

~Obstructs(A, **P_R**, **GP_A**)
~Obstructs(B, **P_R**, **GP_A**)

Grasp(robot, **GP_A**, A, **P_A**, **G_A**)

GraspPose(**GP_A**, **P_A**, **G_A**)
Holding(robot, None)

Preconditions constrain potential values of symbols

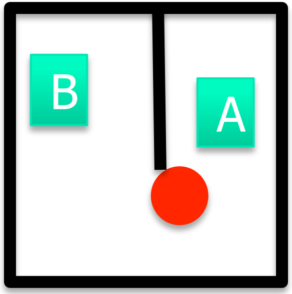
Calls to Motion Planner

Initialize symbols

Determine violated constraint

Modify symbols of violated constraints

Plan Refinement via Local Search



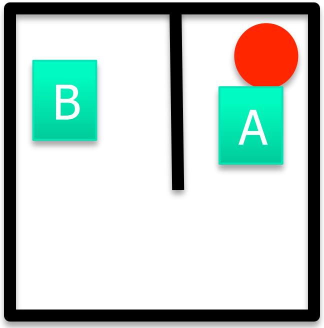
P_R	**(0,0)**
P_A	**(1, 1)**
G_A	**(0.1,0.2)**
GP_A	**(0.5,0.5)**

Initialize symbols

Determine violated constraint

Modify symbols of violated constraints

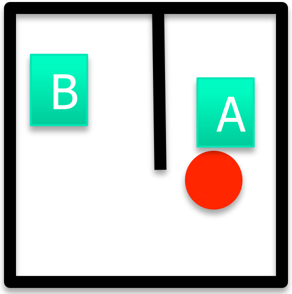
P_R	**(0,0)**
P_A	**(1, 1)**
G_A	**(0.1,0.2)**
GP_A	**(1.1,1.2)**



Obstructs(A, **P_R**, **GP_A**)

~GraspPose(**GP_A**, **P_A**, **G_A**)

Plan Refinement via Local Search



P_R	**(0,0)**
P_A	**(1, 1)**
G_A	**(-0.1,-0.2)**
GP_A	**(0.9,0.8)**

Initialize symbols

Determine violated constraint

Modify symbols of violated constraints

P_R	**(0,0)**
P_A	**(1, 1)**
G_A	**(0.1,0.2)**
GP_A	**(1.1,1.2)**

Obstructs(A, **P_R**, **GP_A**)

~GraspPose(**GP_A**, **P_A**, **G_A**)

Constraint ordering

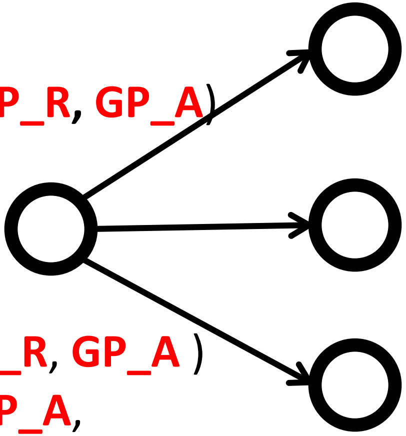
Conditional distribution over symbol values

Searching over Plan Skeletons

- Using the failure information to generate the next state defines a graph
  - Nodes are plan skeletons
  - Edges are failure explanations
- Interleave node expansion (failure propagation) and node refinement (motion planning)

[Guided Search for Task and Motion Plans Using Learned Heuristics Rohan Chitnis, Dylan Hadfield-Menell, Abhishek Gupta, Siddharth Srivastava, Pieter Abbeel. ICRA, 2016 (under review)].

Searching over Plan Skeletons



Obstructs(B, **P_R**, **GP_A**)

Plan Skeleton:
1. Move(robot, **P_R**, **GP_A**)
2. Grasp(robot, **GP_A**, A, **P_A**, **G_A**)

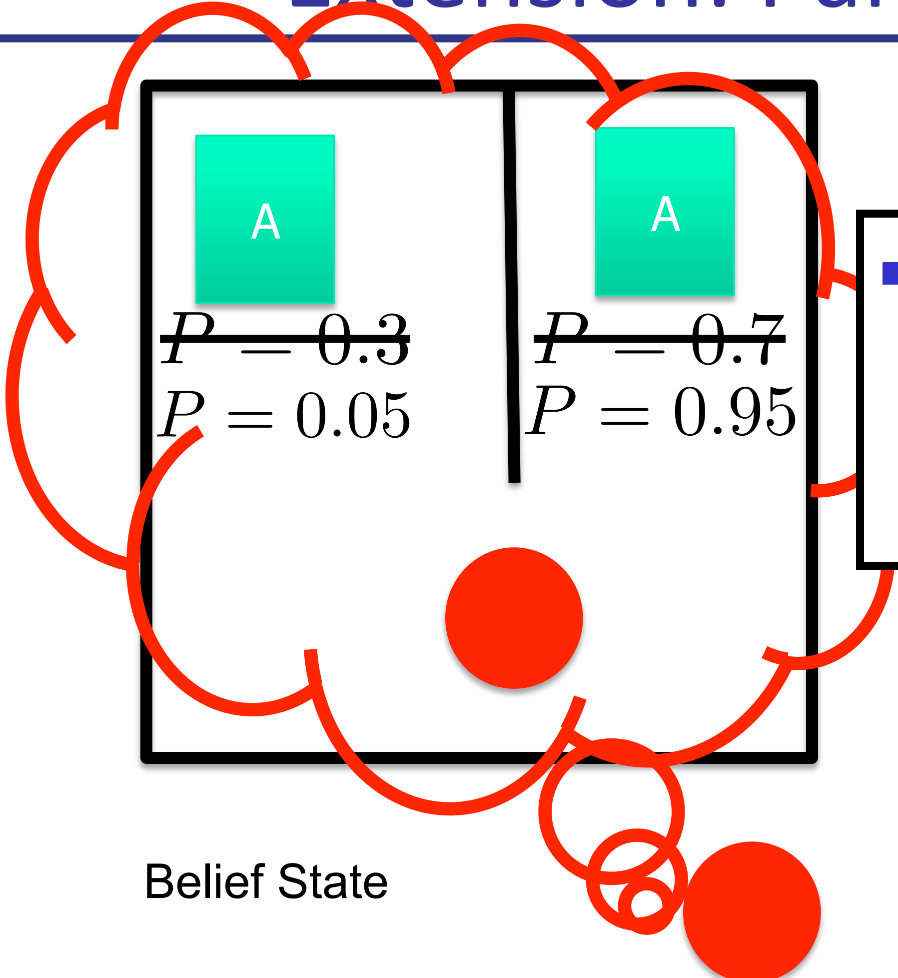
Plan Skeleton:
1. Move(robot, **P_R**, **GP_B**)
2. Grasp(robot, **GP_B**, B, **P_B**, **G_B**)
3. Move(robot, **GP_B**, **PDP_B**)
4. Place(robot, **PDP_B**, B, **PDP_B**, **G_B**)
5. …

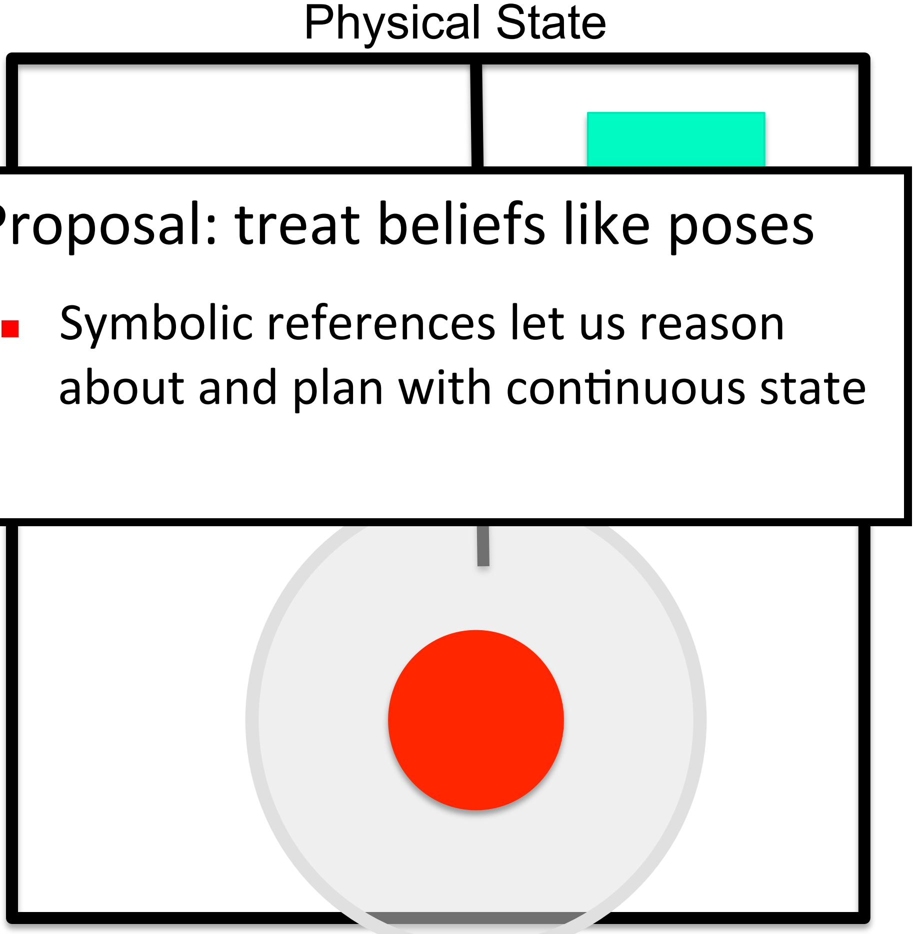
Challenge: need useful heuristics to effectively search this graph.
Solution: learn a heuristic
(details at final project presentations)

Task and Motion Planning Summary

- Pure Task Planning doesn't work directly because of
  - Abstracted continuous dynamics
  - Long horizons
- Solution methods
  - Discretize and represent everything logically
  - Discretize lazily and run motion planning during search
  - Plan abstractly and fill in continuous values later
    - Get a new plan if that doesn't work

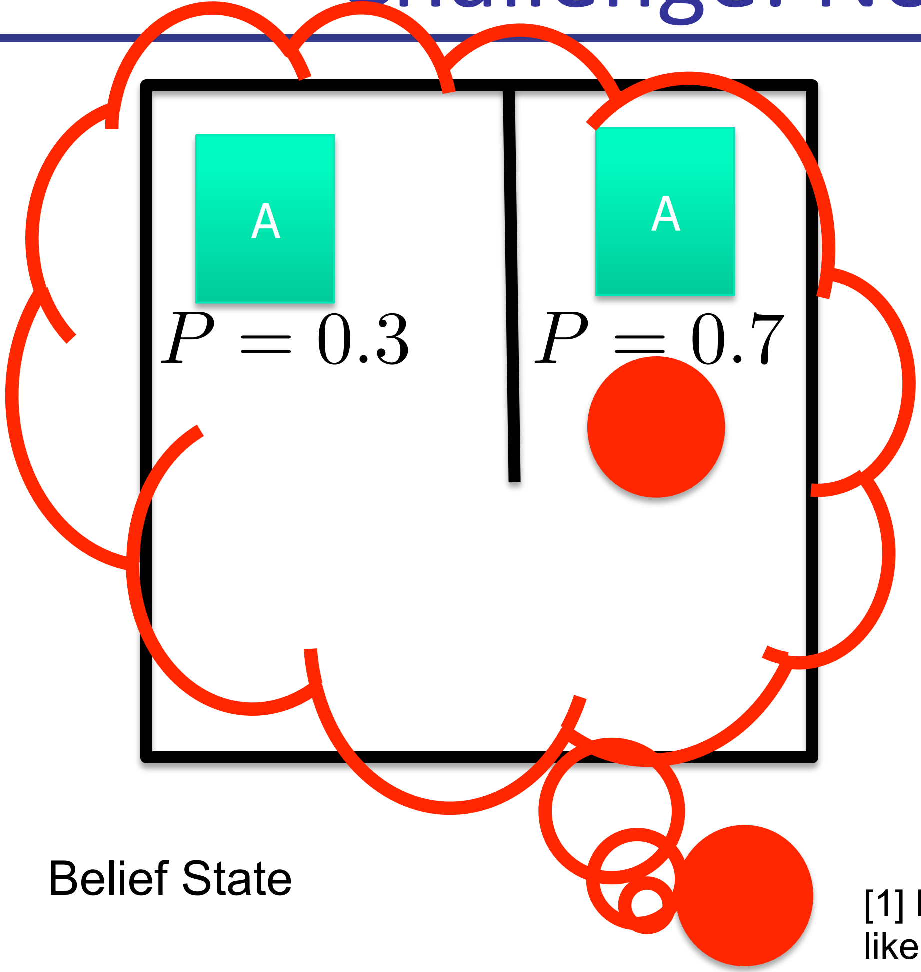
Extension: Partial Observability





- Proposal: treat beliefs like poses
  - Symbolic references let us reason about and plan with continuous state

Challenge: Non-determinism

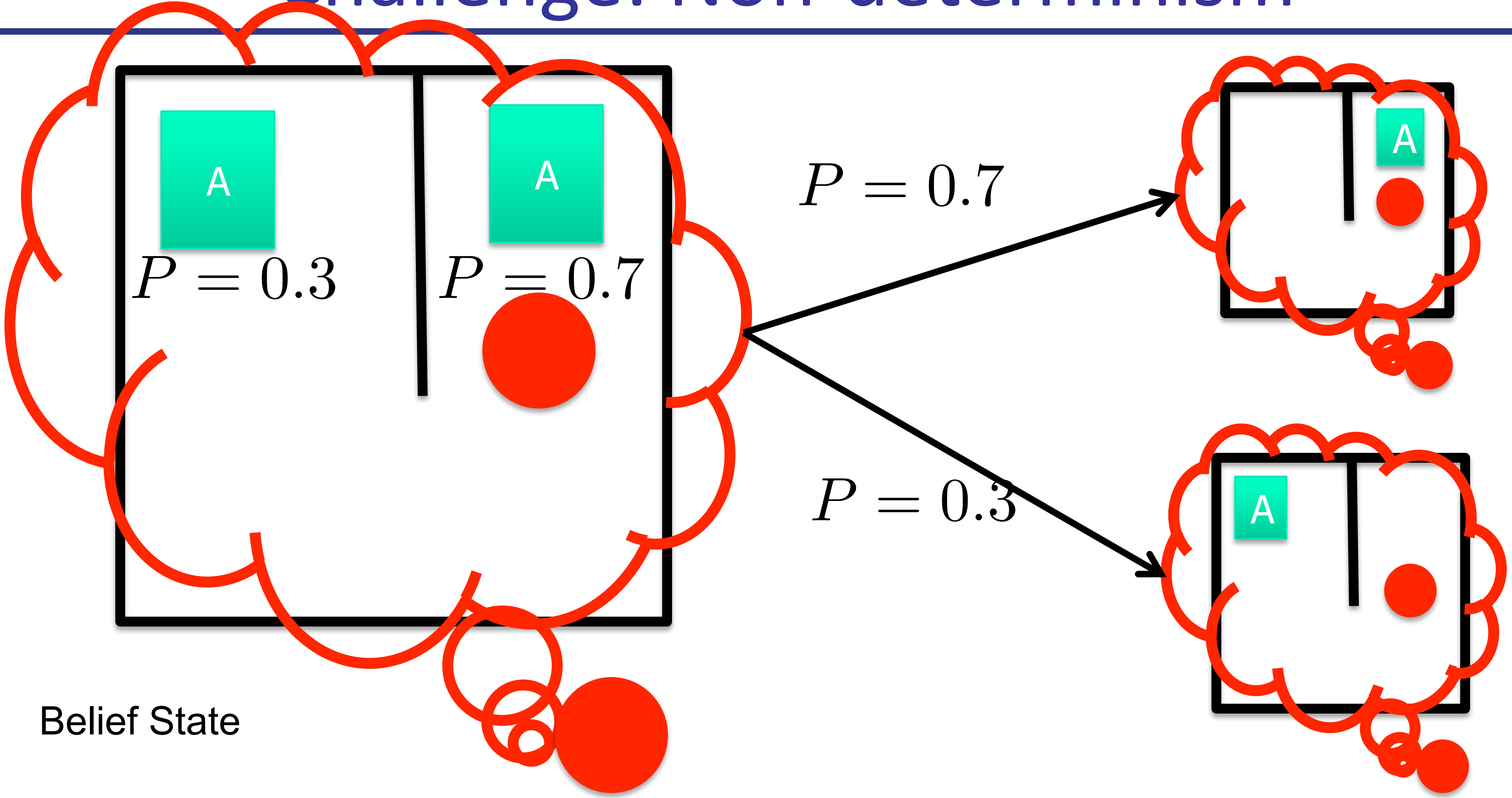


Belief State

- Observations depend on physical state
  - Which we don't know!
- Approximate solution:
  - Assume that each belief state deterministically generates its maximum likelihood observation[1]
  - Re-plan if necessary

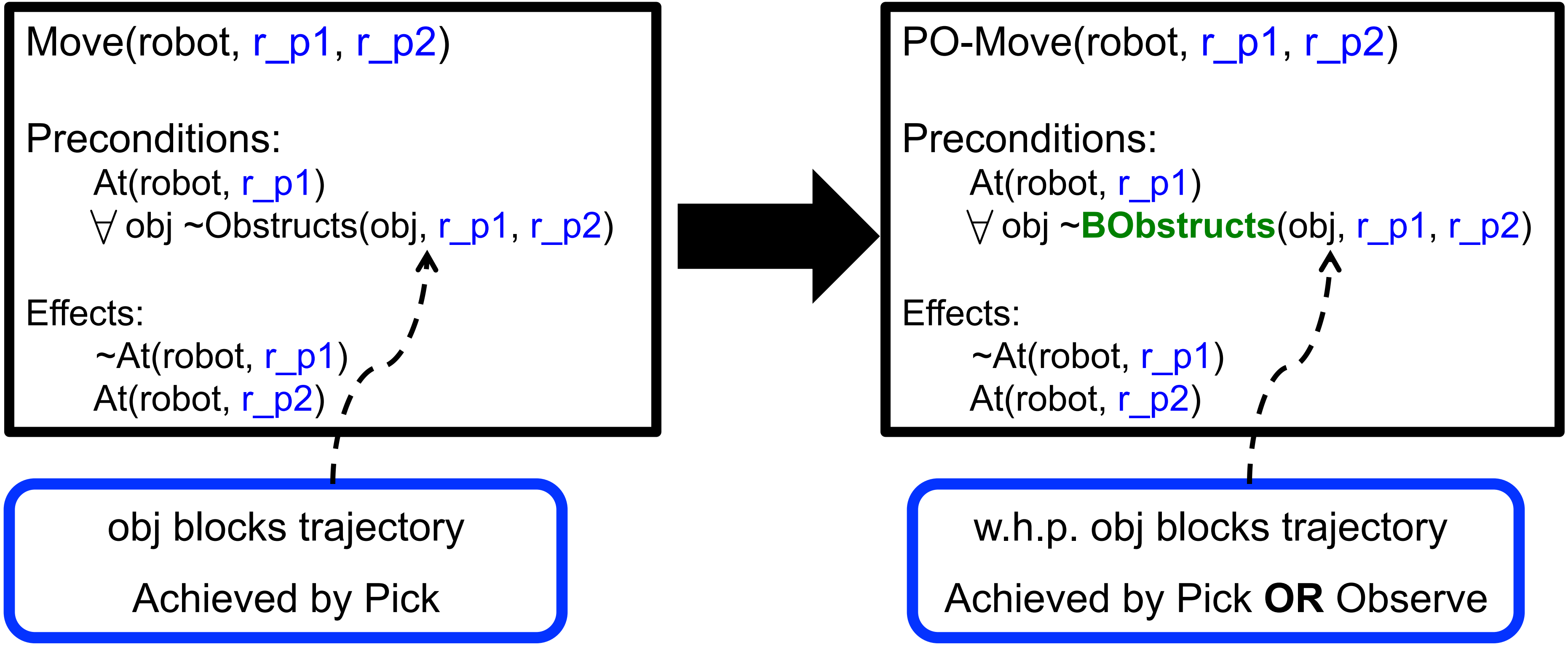
[1] Platt et al. "Belief space planning assuming maximum likelihood observations." RSS (2010).

Challenge: Non-determinism



Belief State

A Partially Observed Move



Move(robot, r_p1, r_p2)

Preconditions:
At(robot, r_p1)
∀ obj ~Obstructs(obj, r_p1, r_p2)

Effects:
~At(robot, r_p1)
At(robot, r_p2)

obj blocks trajectory

Achieved by Pick

PO-Move(robot, r_p1, r_p2)

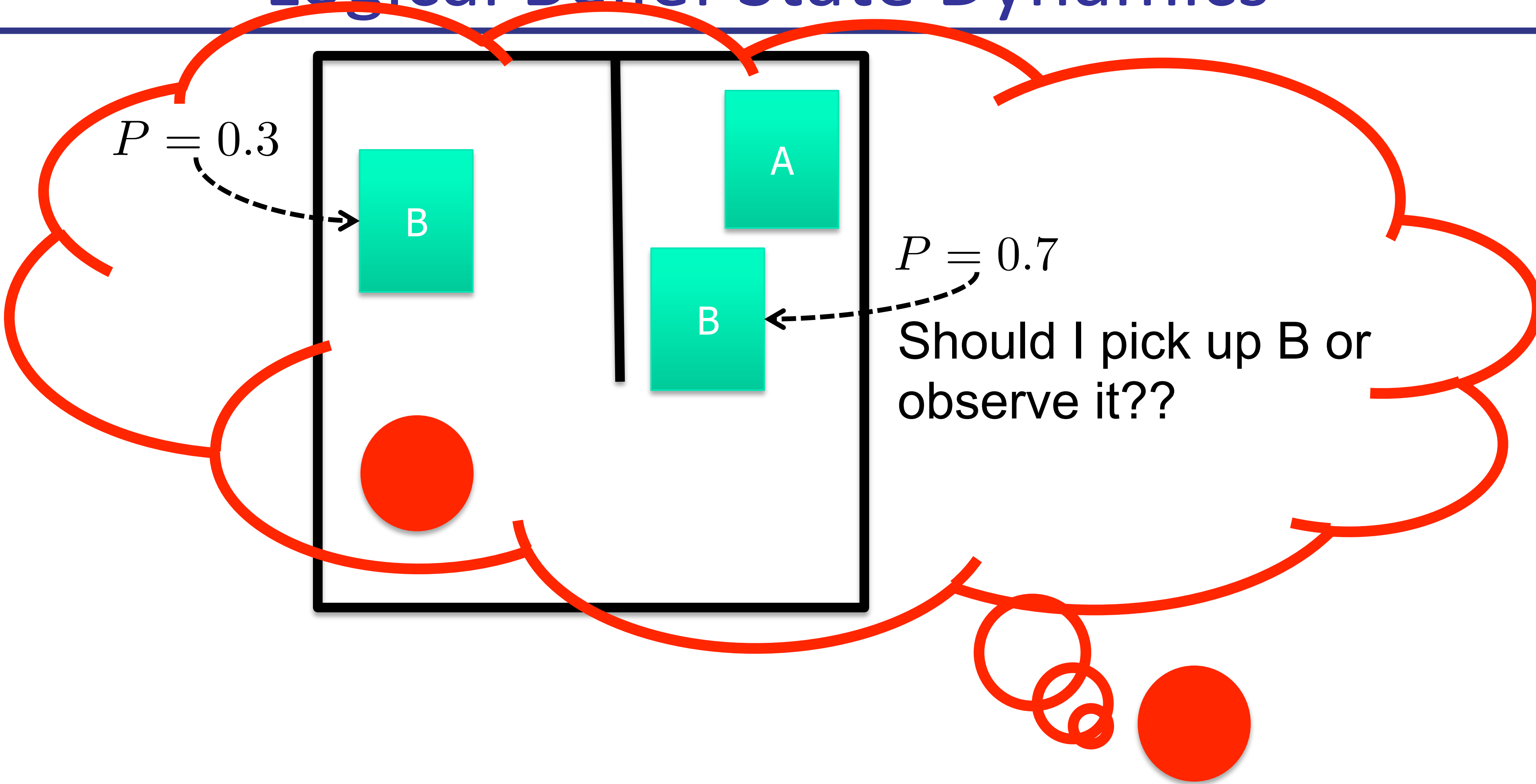
Preconditions:
At(robot, r_p1)
∀ obj ~**BObstructs**(obj, r_p1, r_p2)

Effects:
~At(robot, r_p1)
At(robot, r_p2)

w.h.p. obj blocks trajectory

Achieved by Pick **OR** Observe

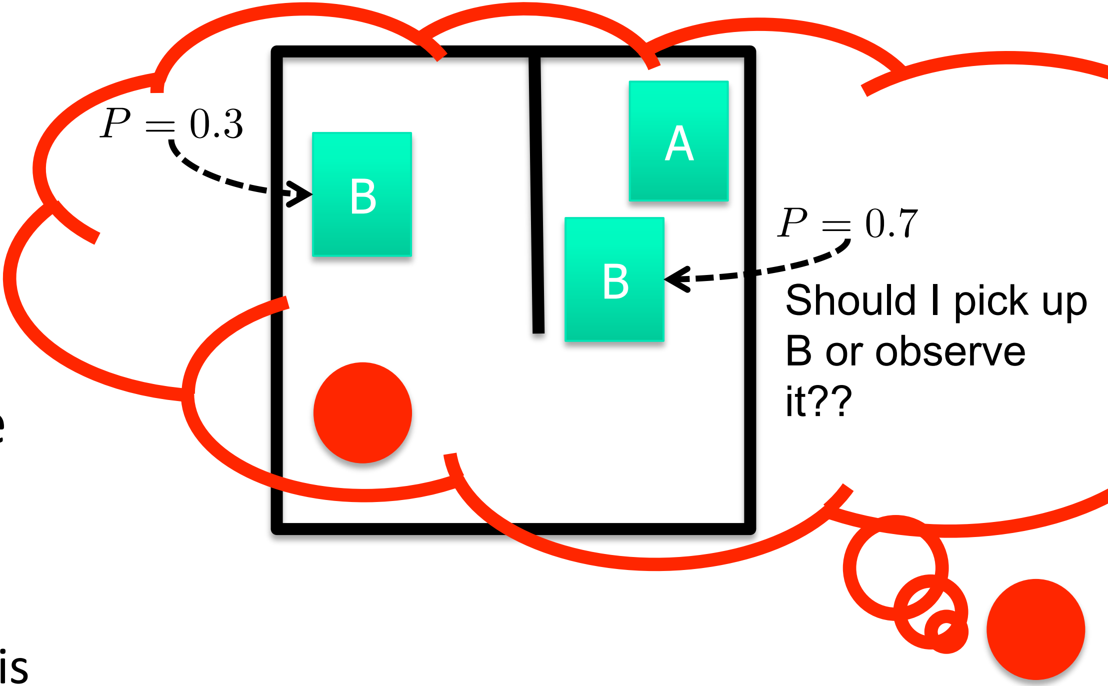
Logical Belief State Dynamics



Should I pick up B or observe it??

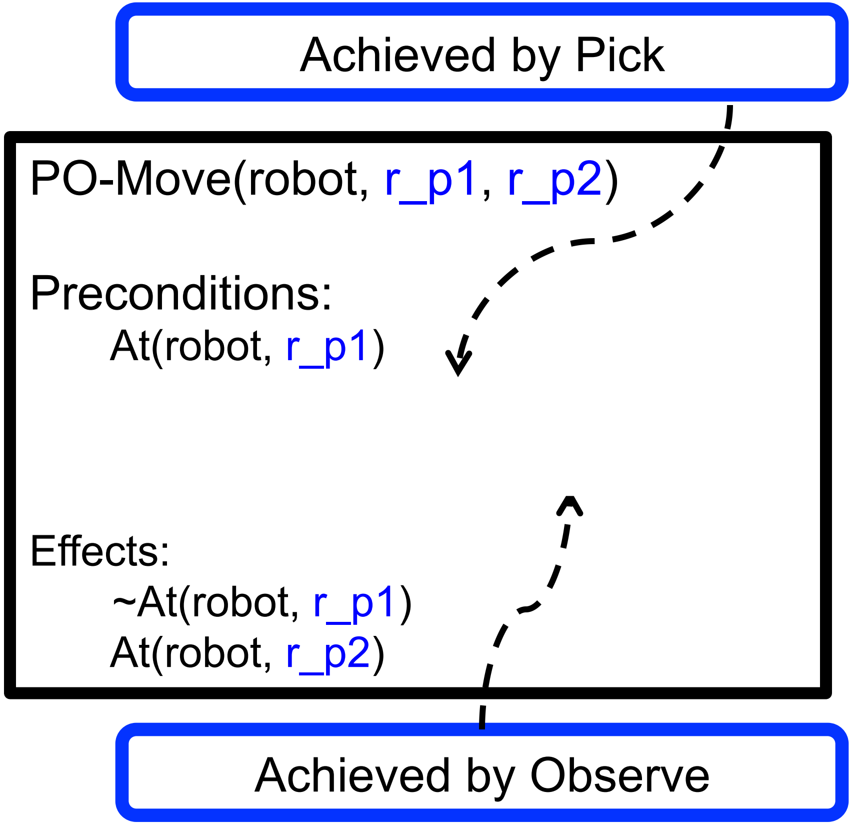
Logical Belief State Dynamics

- In the POMDP formulation, answering this question is complicated…
- Key Idea: observation will only be useful if it lets us conclude that B is not in the way
  - We've assumed maximum likelihood observations, so this is tractable



Logical Belief Space Dynamics

- Split properties of belief states into 2 cases
    - Properties of maximum likelihood states
    - Properties of associated uncertainty
- Interface determines which caused failure and updates high level



Achieved by Pick

PO-Move(robot, r_p1, r_p2)

Preconditions:
At(robot, r_p1)

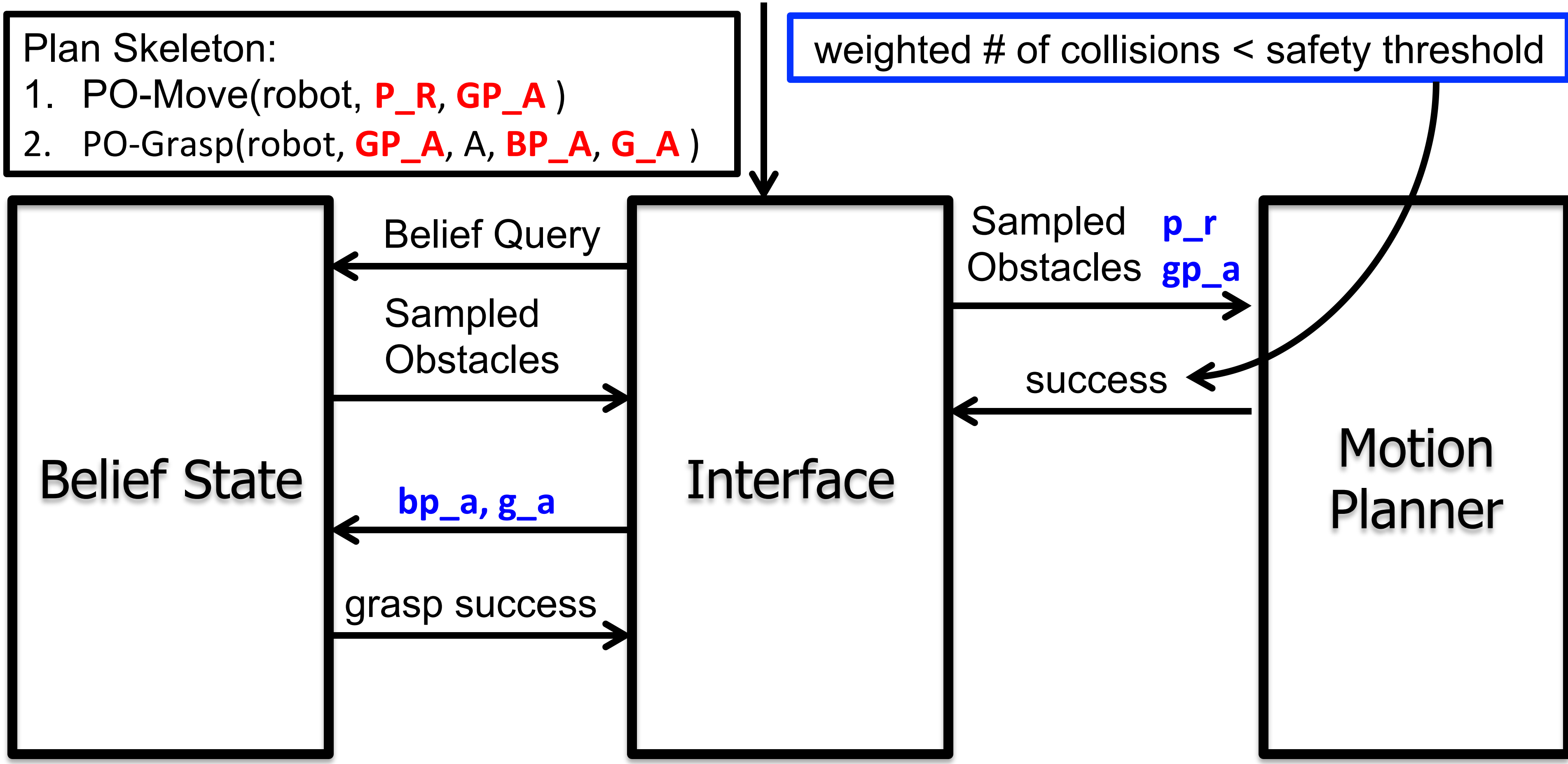
Effects:
~At(robot, r_p1)
At(robot, r_p2)

Achieved by Observe

Refining a Plan Skeleton in Belief Space

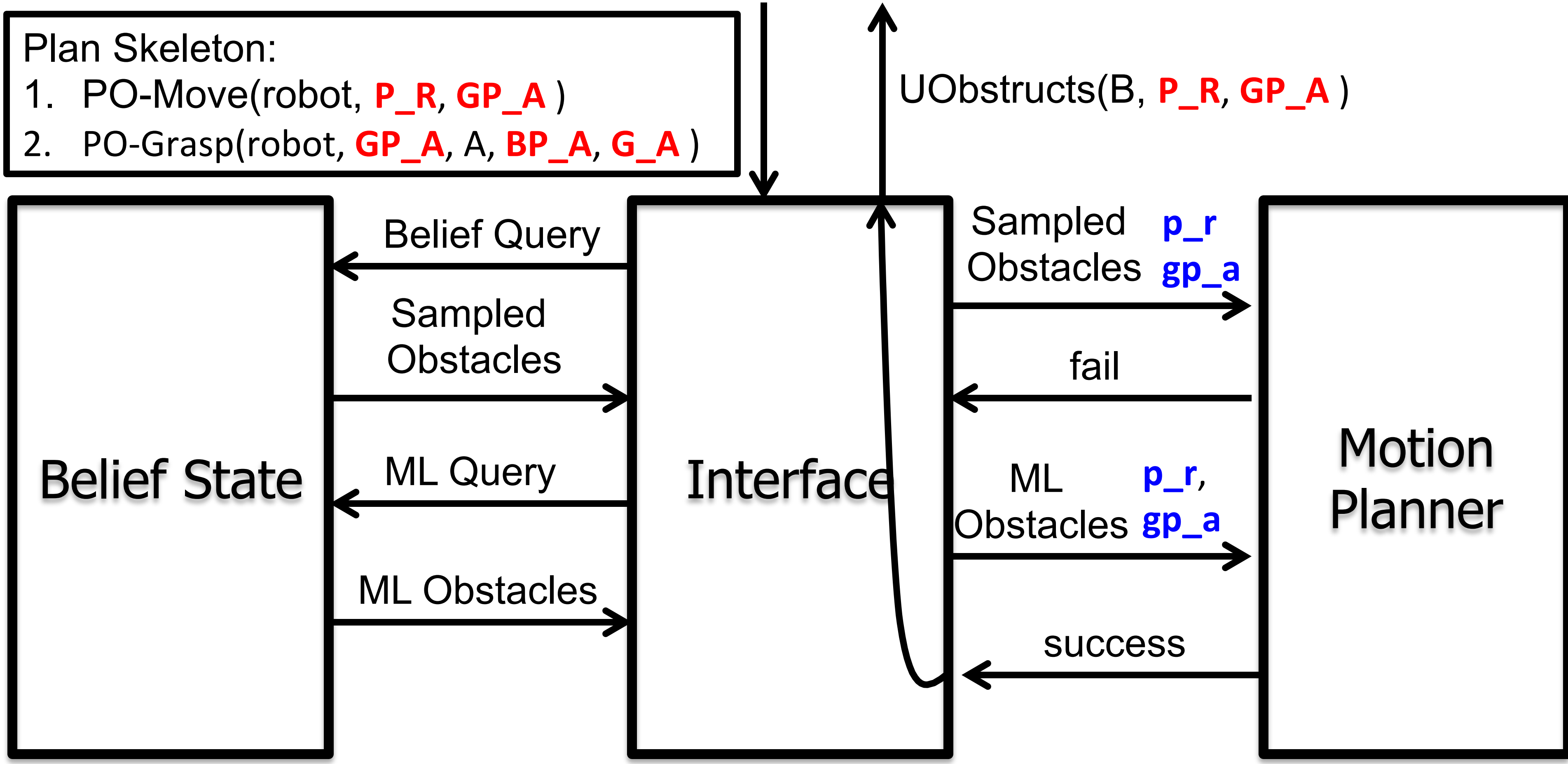
Plan Skeleton:

1. PO-Move(robot, **P_R**, **GP_A**)
2. PO-Grasp(robot, **GP_A**, A, **BP_A**, **G_A**)



weighted # of collisions < safety threshold

Error Propagation in Belief Space



Plan Skeleton:

1. PO-Move(robot, **P_R**, **GP_A**)
2. PO-Grasp(robot, **GP_A**, A, **BP_A**, **G_A**)

UObstructs(B, **P_R**, **GP_A**)

Belief State

Interface

Motion Planner

Belief Query

Sampled Obstacles

ML Query

ML Obstacles

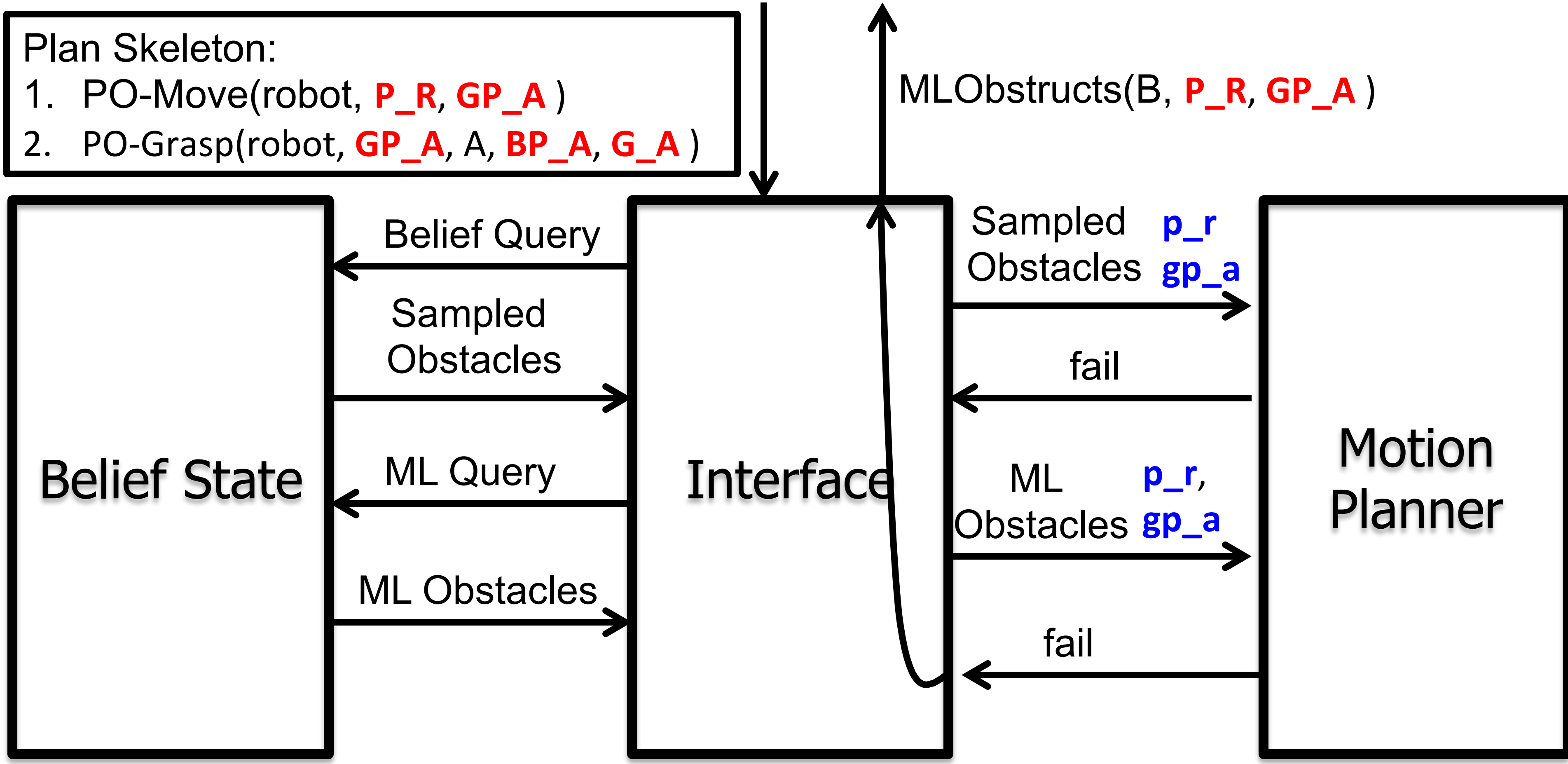
Sampled Obstacles **p_r** **gp_a**

fail

ML Obstacles **p_r**, **gp_a**

success

Error Propagation in Belief Space



Plan Skeleton:

1. PO-Move(robot, **P_R**, **GP_A**)
2. PO-Grasp(robot, **GP_A**, A, **BP_A**, **G_A**)

MLObstructs(B, **P_R**, **GP_A**)

Belief State

Belief Query

Sampled Obstacles

ML Query

ML Obstacles

Interface

Sampled Obstacles **p_r** **gp_a**

fail

ML Obstacles **p_r**, **gp_a**

fail

Motion Planner