Lecture 34

CS 6260 Automated Planning and Learning

Motion Planning

Department of Computer Science and Engineering

Indian Institute of Technology Madras

Material based on IJCAI 2023 Tutorial on Integrated Task and Motion Planning by Naman Shah, Sarah Keren, and Siddharth Srivastava.

Motion Planning

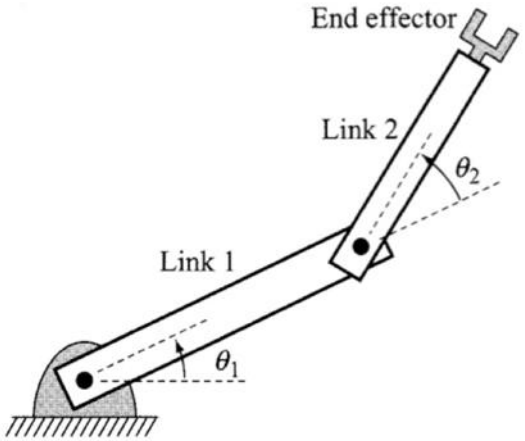
- Problem:
  - Given start state x_S, goal state x_G
  - Asked for: a sequence of control inputs that leads from start to goal

- Why tricky?
  - Need to avoid obstacles
  - For systems with underactuated dynamics: can't simply move along any coordinate at will
    - E.g., car, helicopter, airplane, but also robot manipulator hitting joint limits

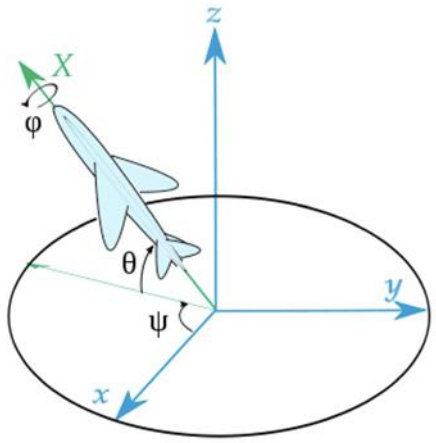
Configuration Space (C-Space): State Space for a Robot in an Environment

Configuration: A complete specification of the position of every point in the system

C-Space: Space of all possible system configurations



2 - Dim



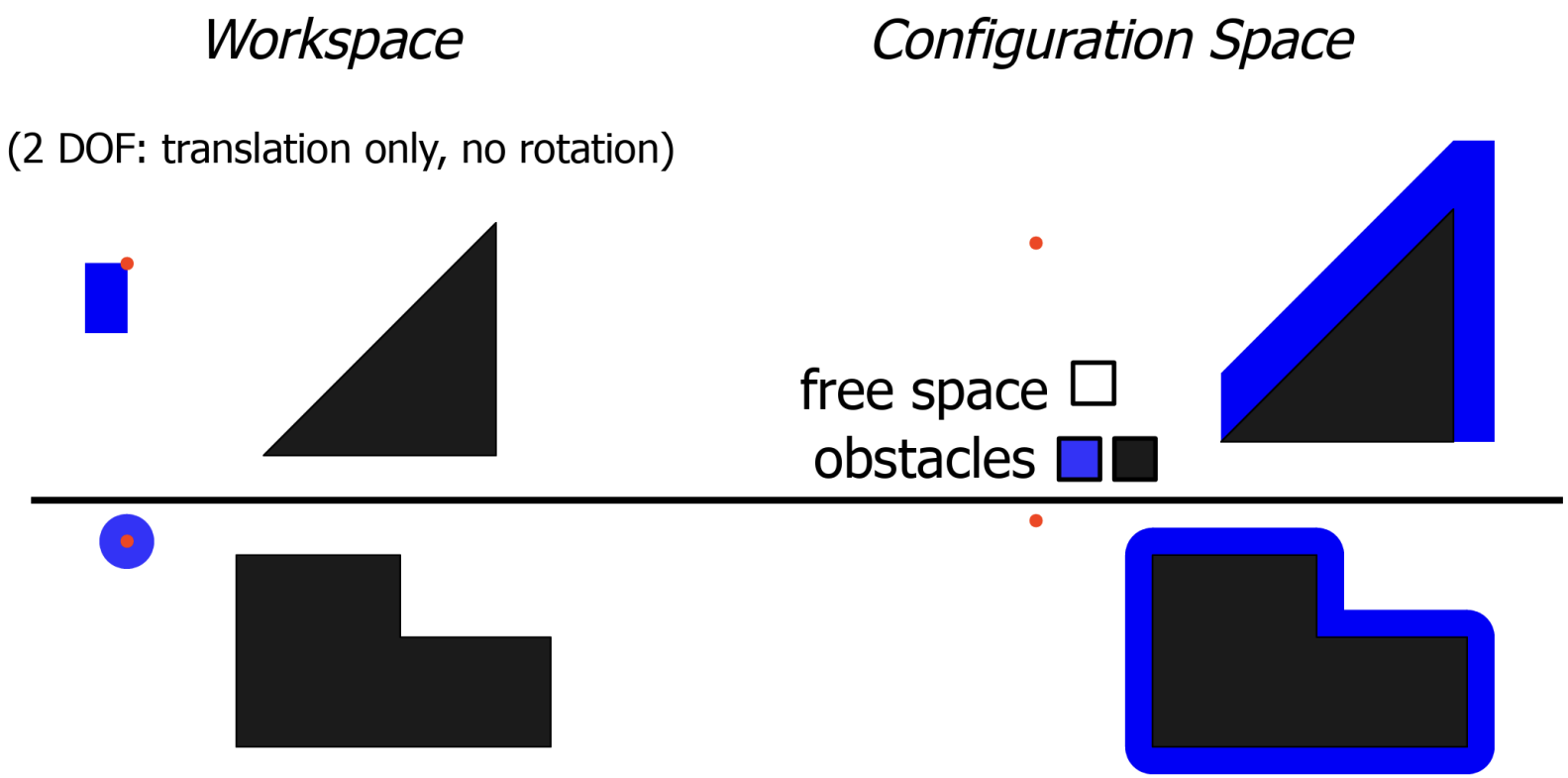
6 - Dim

20 - Dim

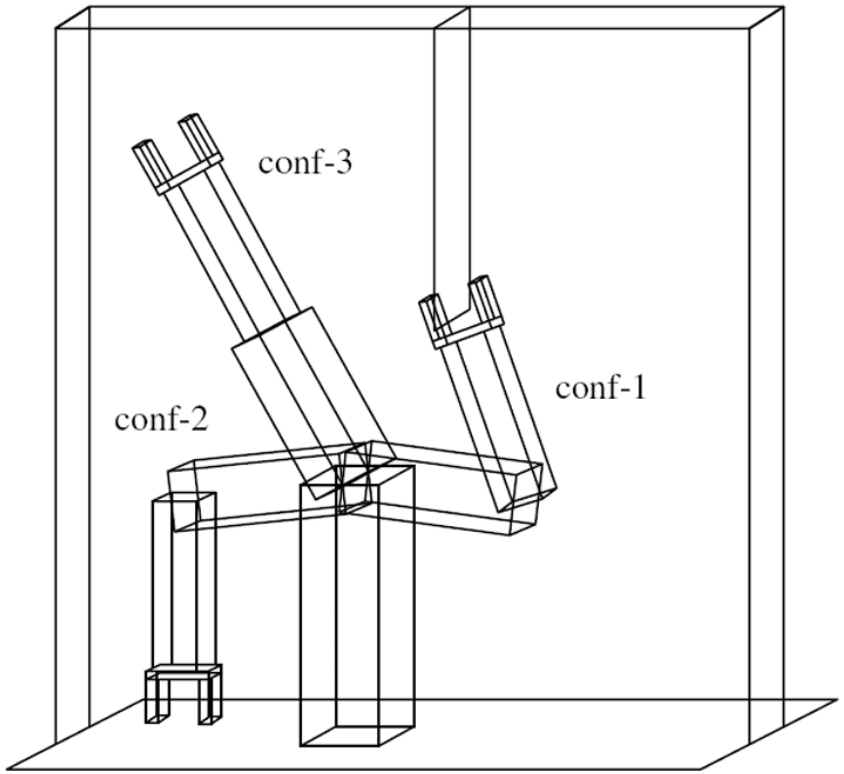
Configuration Space

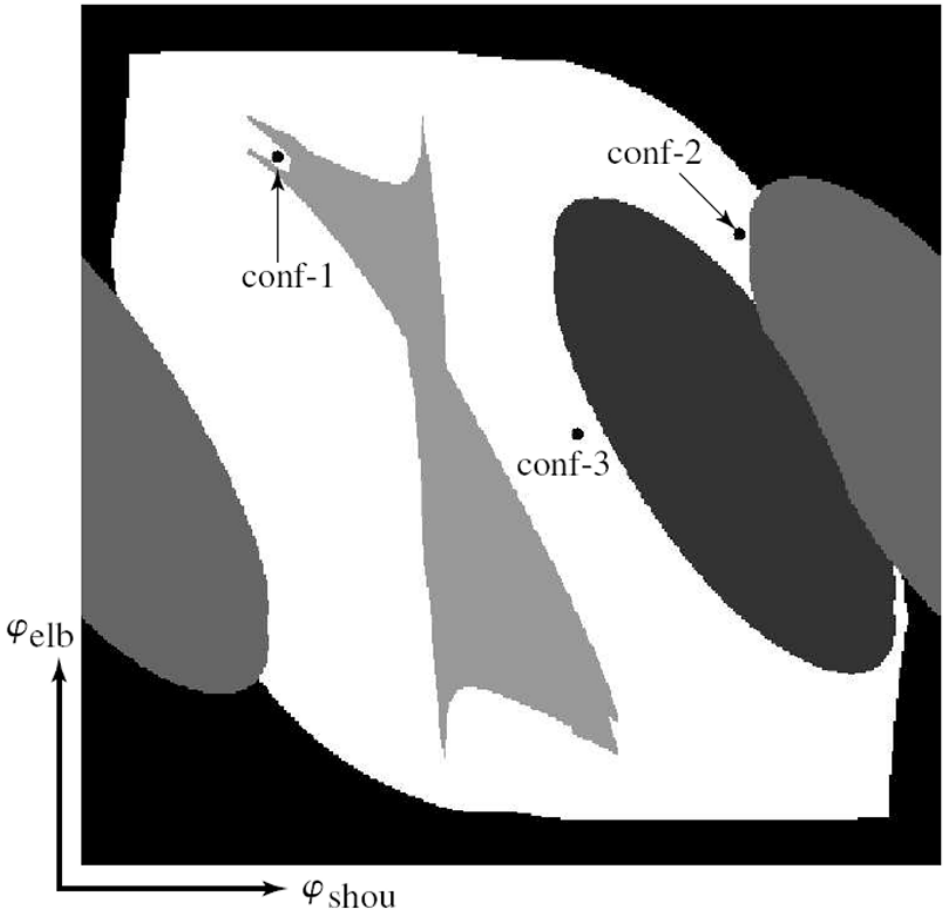
- Space of all possible configurations of a robot.

- obstacles -> configuration space obstacles



Configuration Space





Examples

PR2: Two 8 DoF arms + 1 DoF height + 3 DoF base

YuMi: Two 7-DoF arms

Formulation as Motion Planning Problems:

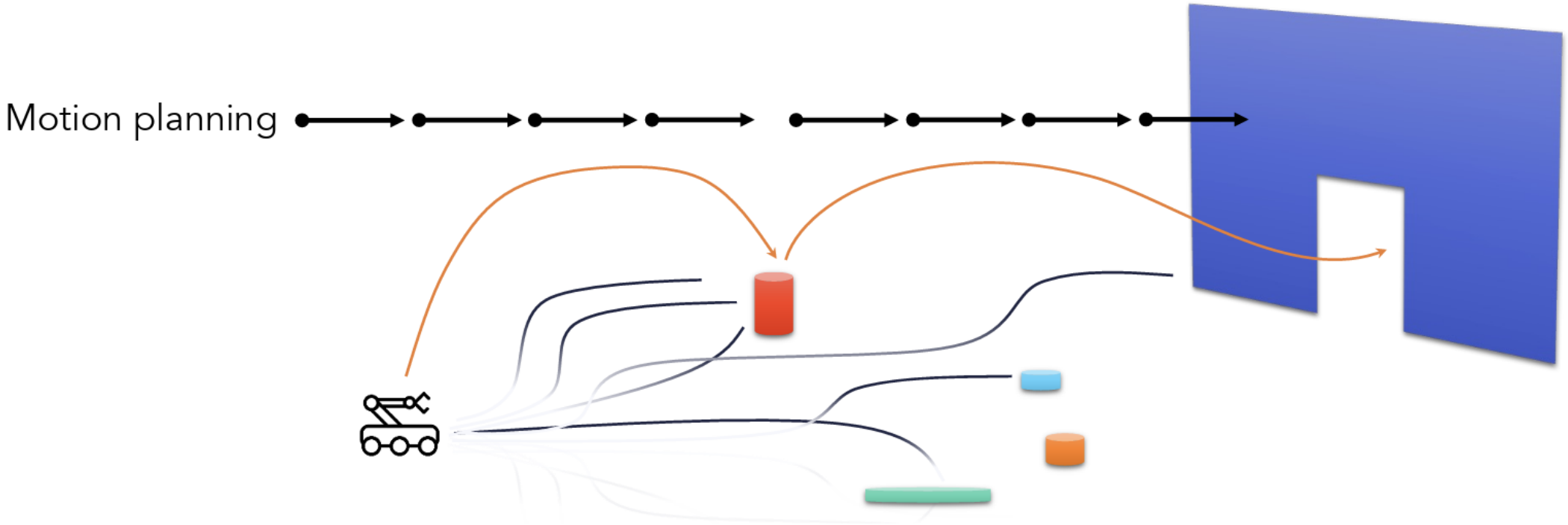
State (Config) Space X =?

x_i, x_g =?

What happens to the C-space when the robot picks up a plank?

Pure Motion Planning is Not Enough!

- Motion planning – which path (of waypoints) should the robot take?
  - But which motion planning problem should it solve? different pickups ⇒ different c-spaces
  - Where would motion planning goals in each C-space come from?
  - Clearly, motion planning is not enough



Would Pure Task-Planning Do the Trick?

Can a "higher-level" planner help us compute the strategy?

Then we could refine each action in the plan into a motion plan

Higher-level planning is typically done over states described using features, or properties

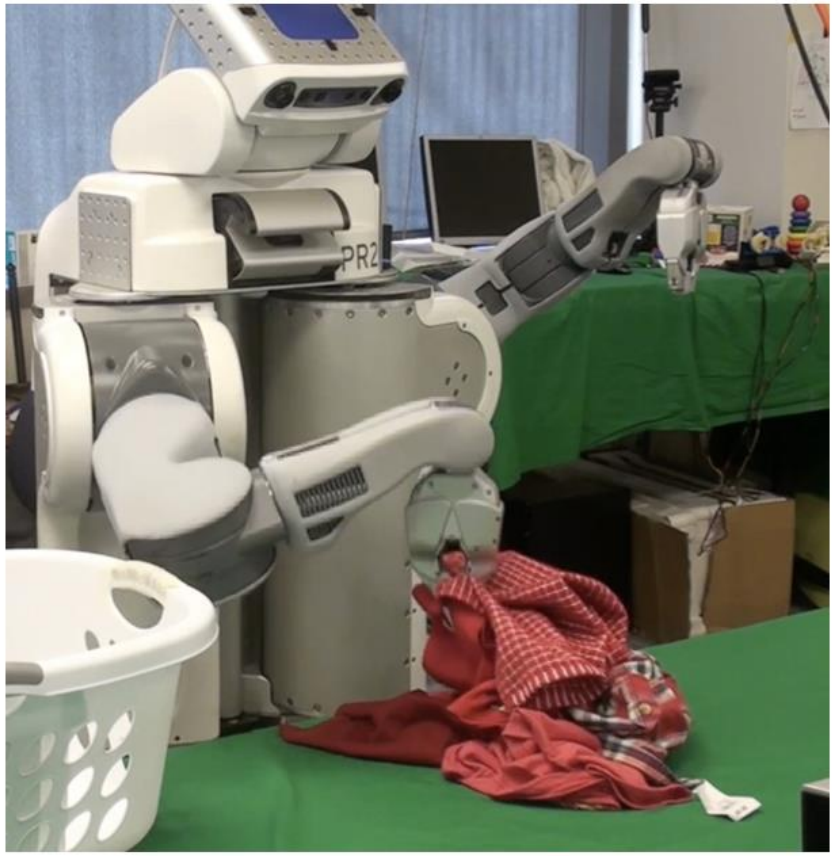
E.g., #clothes on table

IsHolding(robot, basket)

...

+

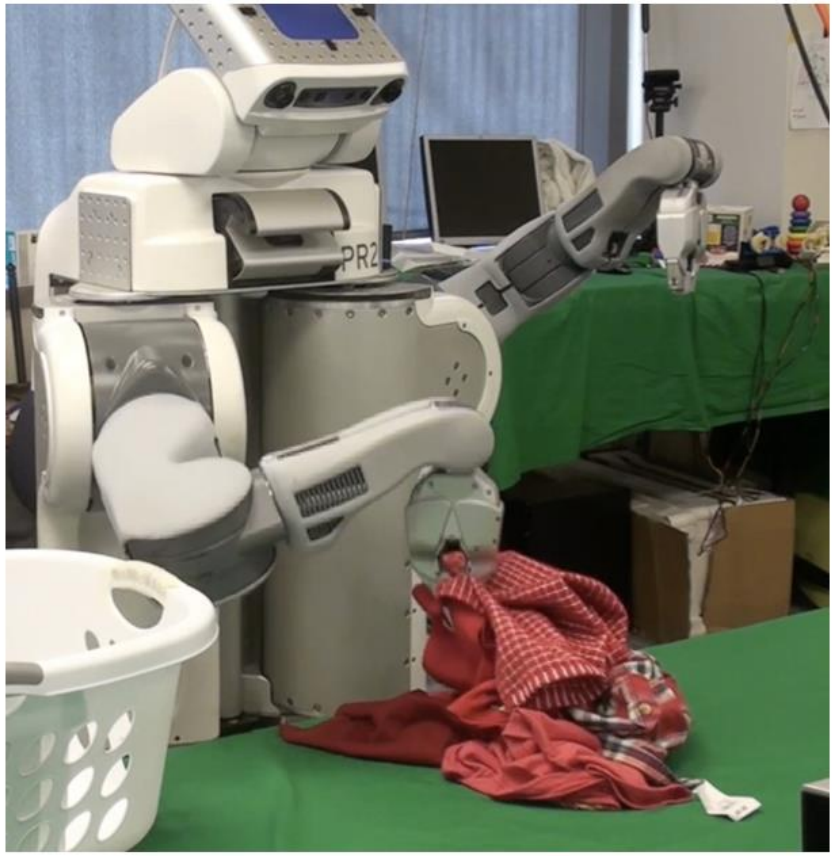
Actions describing how and when robot can change these properties



Would Pure Task-Planning Do the Trick?

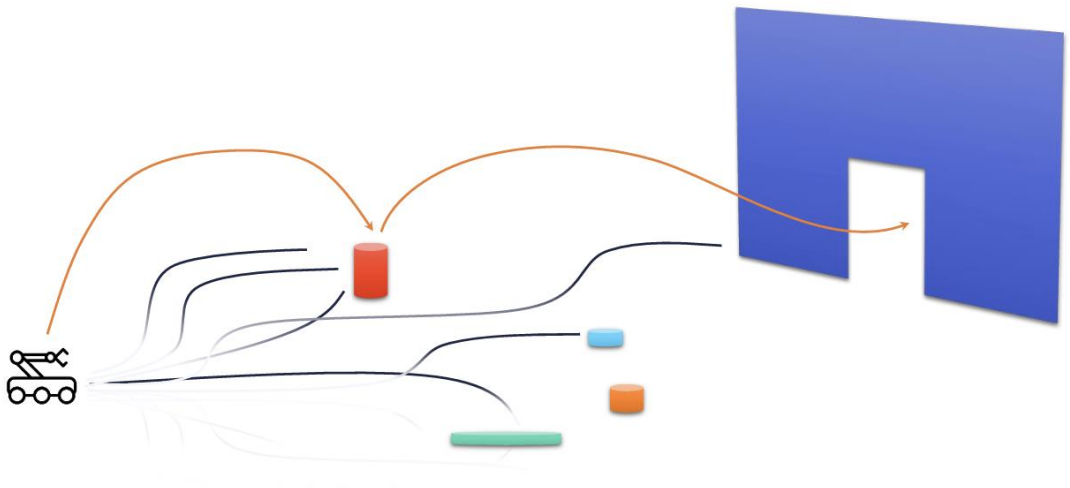
Example of a high-level action:

```
(:action pickup
  :parameters (?obj ?gripper)
  :precondition (and (empty ?gripper)
                     (ontable ?obj))
  :effect (and (not (empty ?gripper))
               (not (ontable ?obj)
               (in ?obj, ?gripper)))))
```



SDM problem: which sequence of actions will lead to the goal?

How do These High-Level Actions Connect to the C-Space?



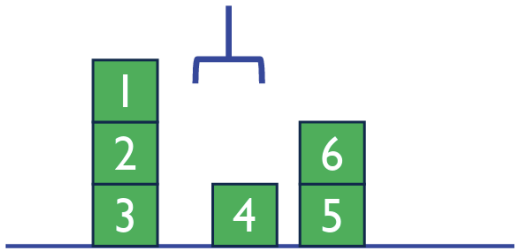
GoTo(l)
Pickup(x)
PutDown(x)

Temporal Abstraction
≡ Abstract Actions,
Macros,
Options…

At(x, l)
InGripper(x)
AtDestination(x)

State
Abstraction

Sometimes Intuitive Abstractions are … Perfect



Pickup(x)

PutDown(x)

OnTable(3)

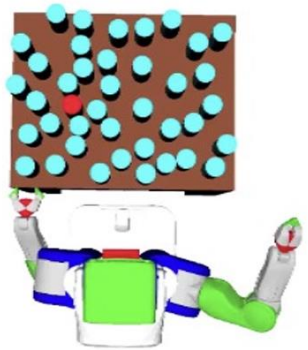
On(1,2)

…

```
(:action pickup
  :parameters (?obj ?gripper)
  :precondition (and (empty ?gripper)
                     (ontable ?obj) (clear ?obj)
  :effect (and (not (empty ?gripper))
               (not (ontable ?obj)
               (in ?obj, ?gripper) (not (clear ?obj)) ))))
```

But Human Intuition has its Limits

Pickup the red can!



Can pickup only from the side

Obstructions depend on choice of movement trajectory

?!

Prevailing Abstraction

Pickup(x)	Abstract
PutDown(x)	Actions

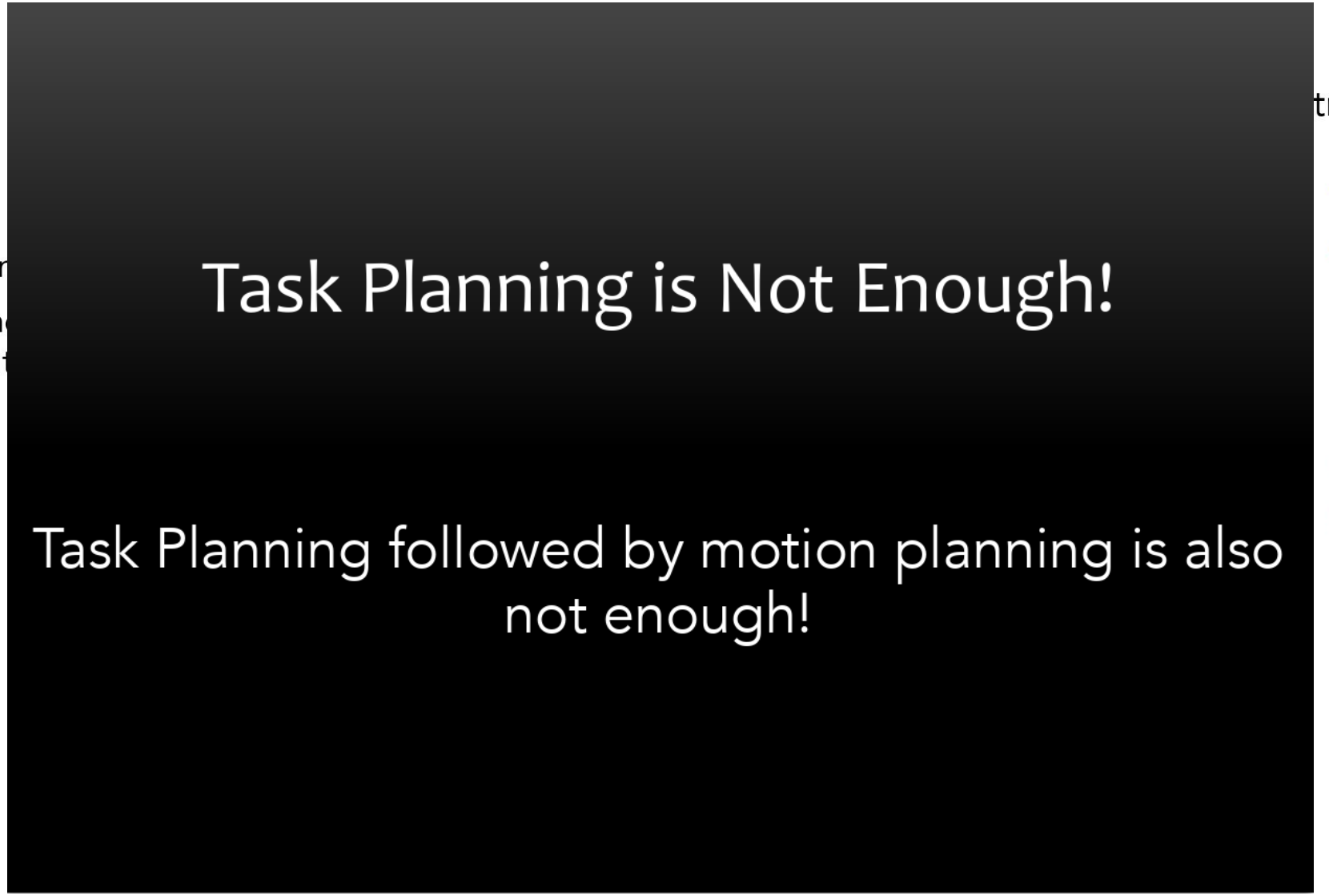
OnTable(3)	
~~On(1,2)~~	Abstract
OnTable(2)	State
OnTable(1)	

…

Abstract model *thinks* this is a trivial problem

Solutions from abstract model: Mostly infeasible

But Human Intuition has its Limits



traction

Abstract Actions

Abstract State

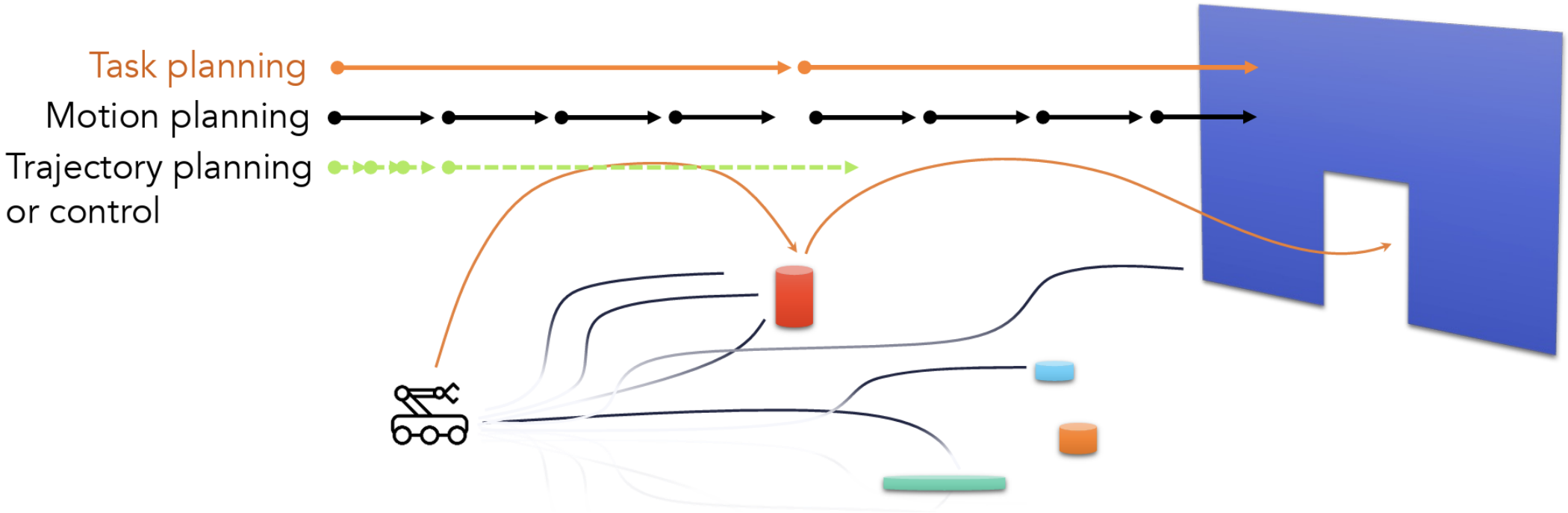
Can pickup only fro

Obstructions depe
choice of movemen

Summary: We Need to Integrate Task and Motion Planning!

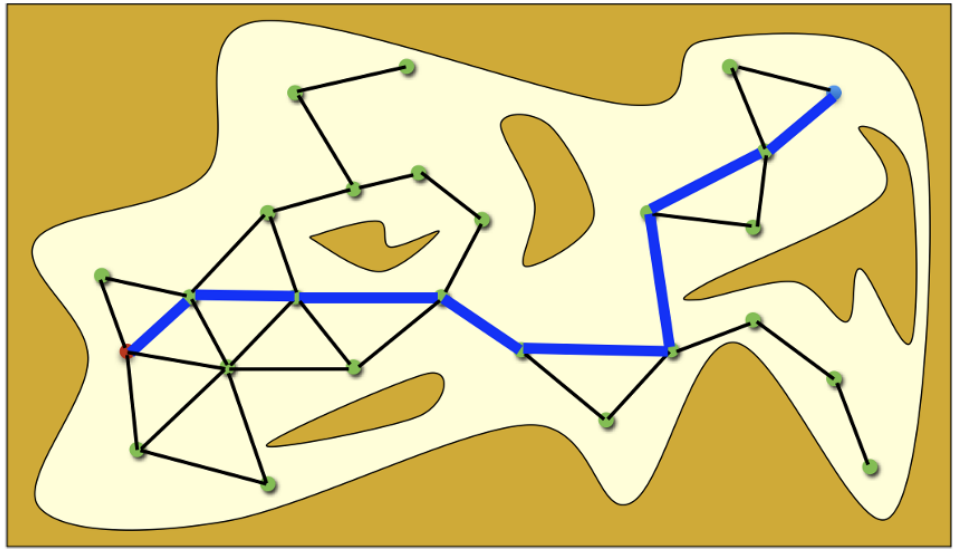
- Task planning – given a task planning problem, computes the high-level action the robot should perform at each step
  - But that action may have no feasible motion plan (recall: cluttered table)
- Motion planning – given a motion planning problem, computes the path that the robot should take
  - But which motion planning problem should it solve?

(Trajectory planning – selects the control inputs that should go to the robot's motors)

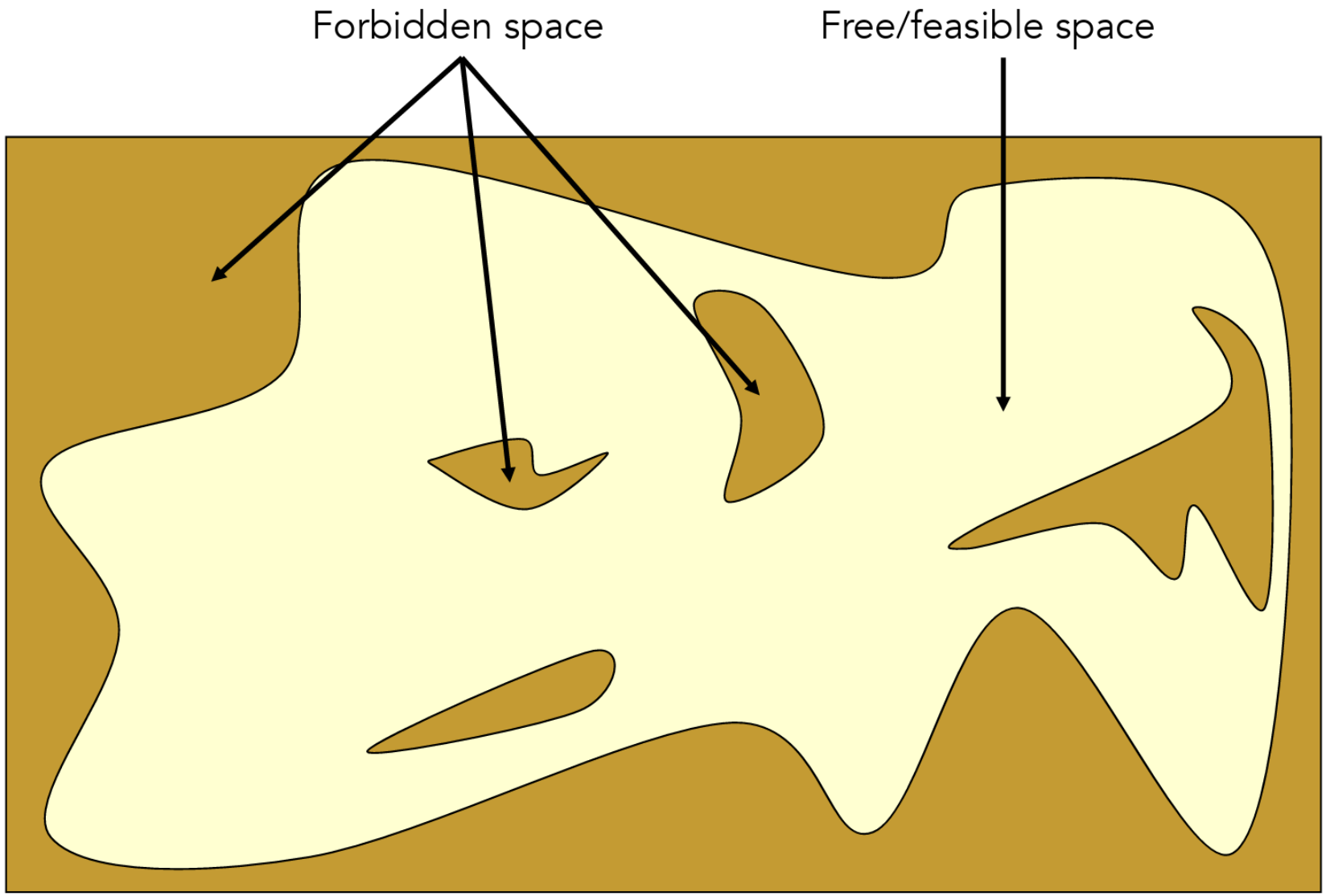


Sampling-based Motion Planning

- Sampling-based solutions sample the C-Space instead of explicitly computing it
  - Probabilistic Roadmap (PRM)
  - Rapidly-exploring random tree (RRT)
- Simply need to know if the robot is in collision (workspace query)



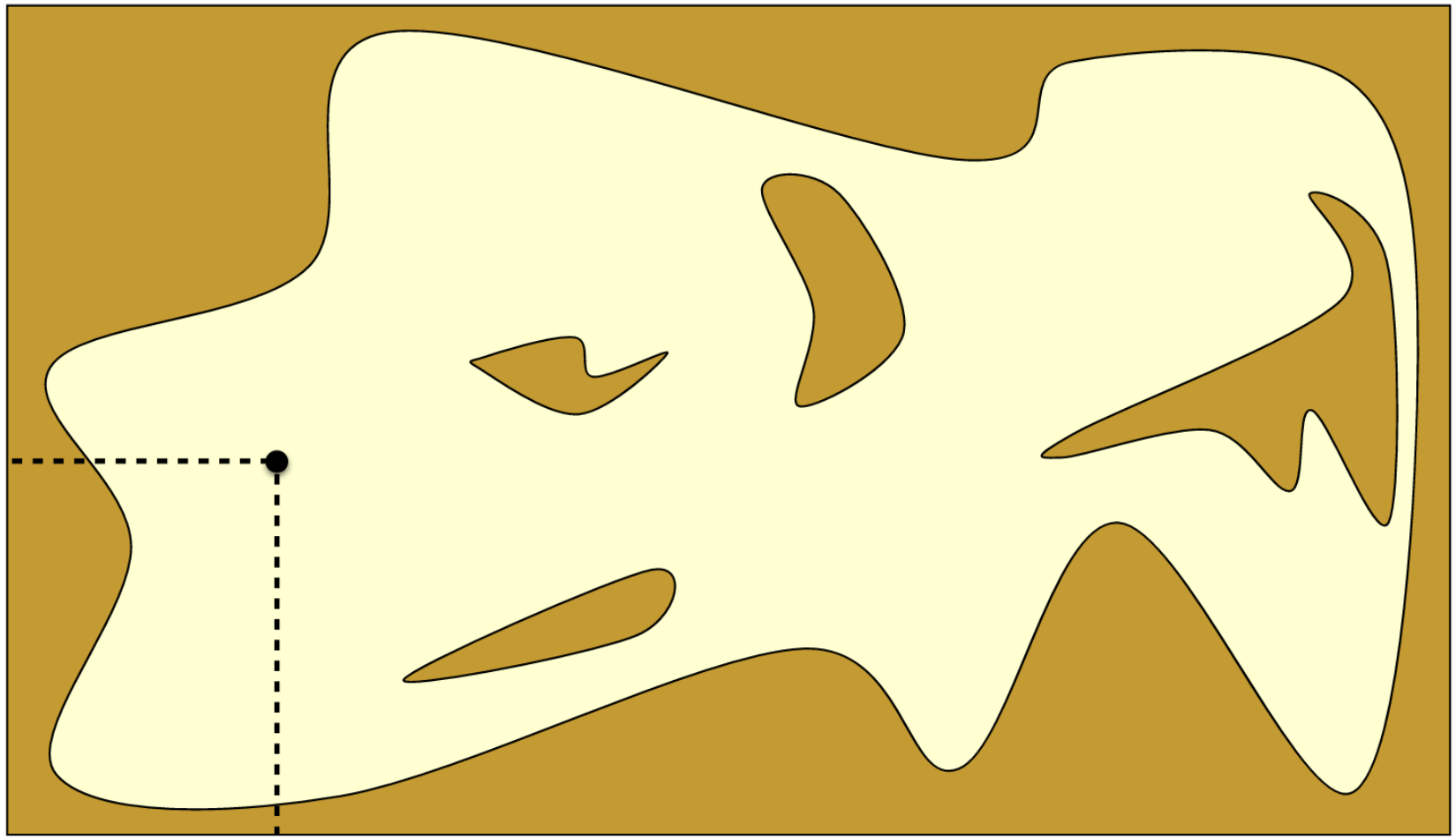
Probabilistic Road Map (PRM)



[Source: CS287, Pieter Abbeel, UC Berkeley]

Probabilistic Road Map (PRM)

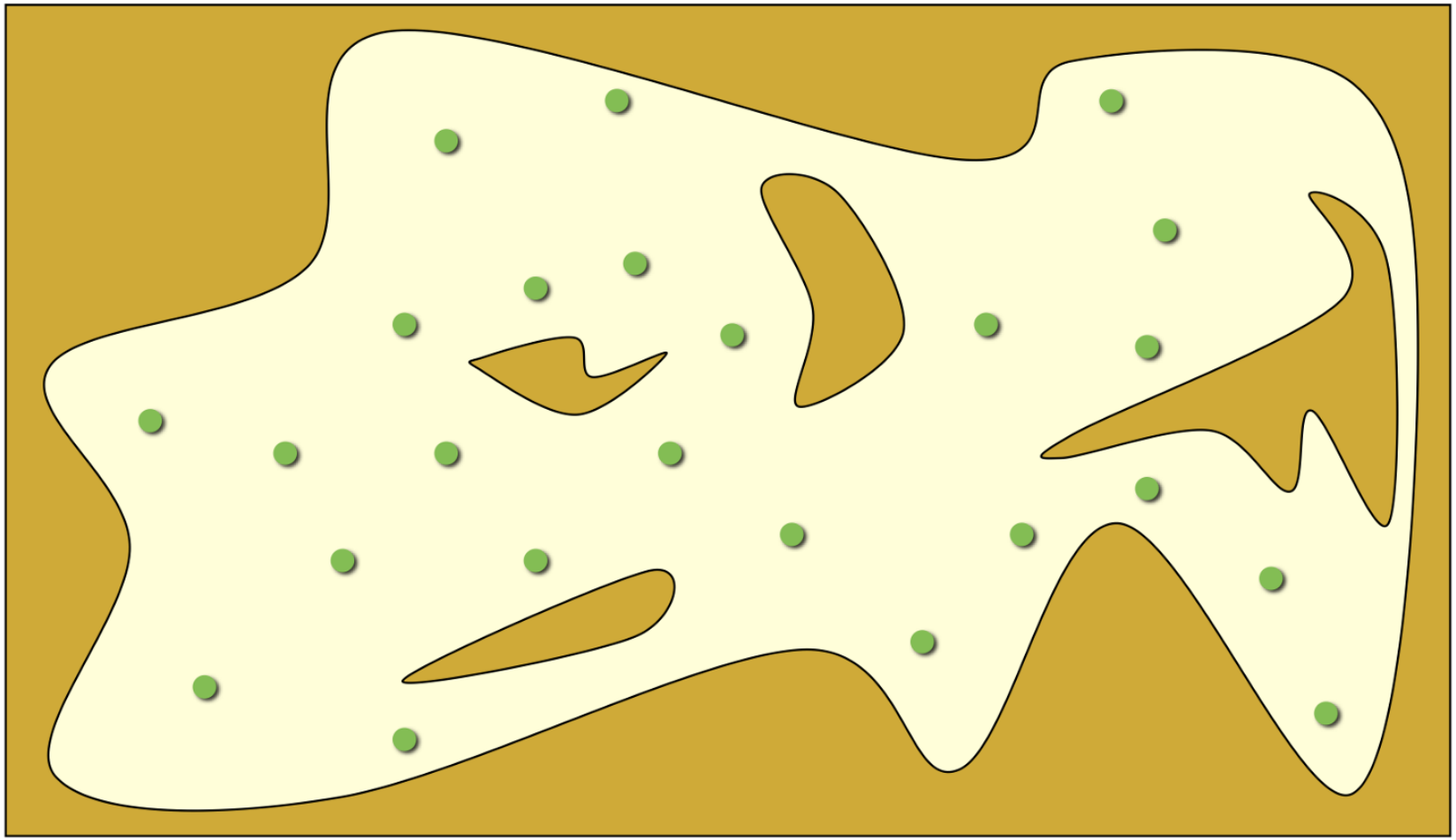
Configurations are sampled by picking coordinates at random



[Source: CS287, Pieter Abbeel, UC Berkeley]

Probabilistic Road Map (PRM)

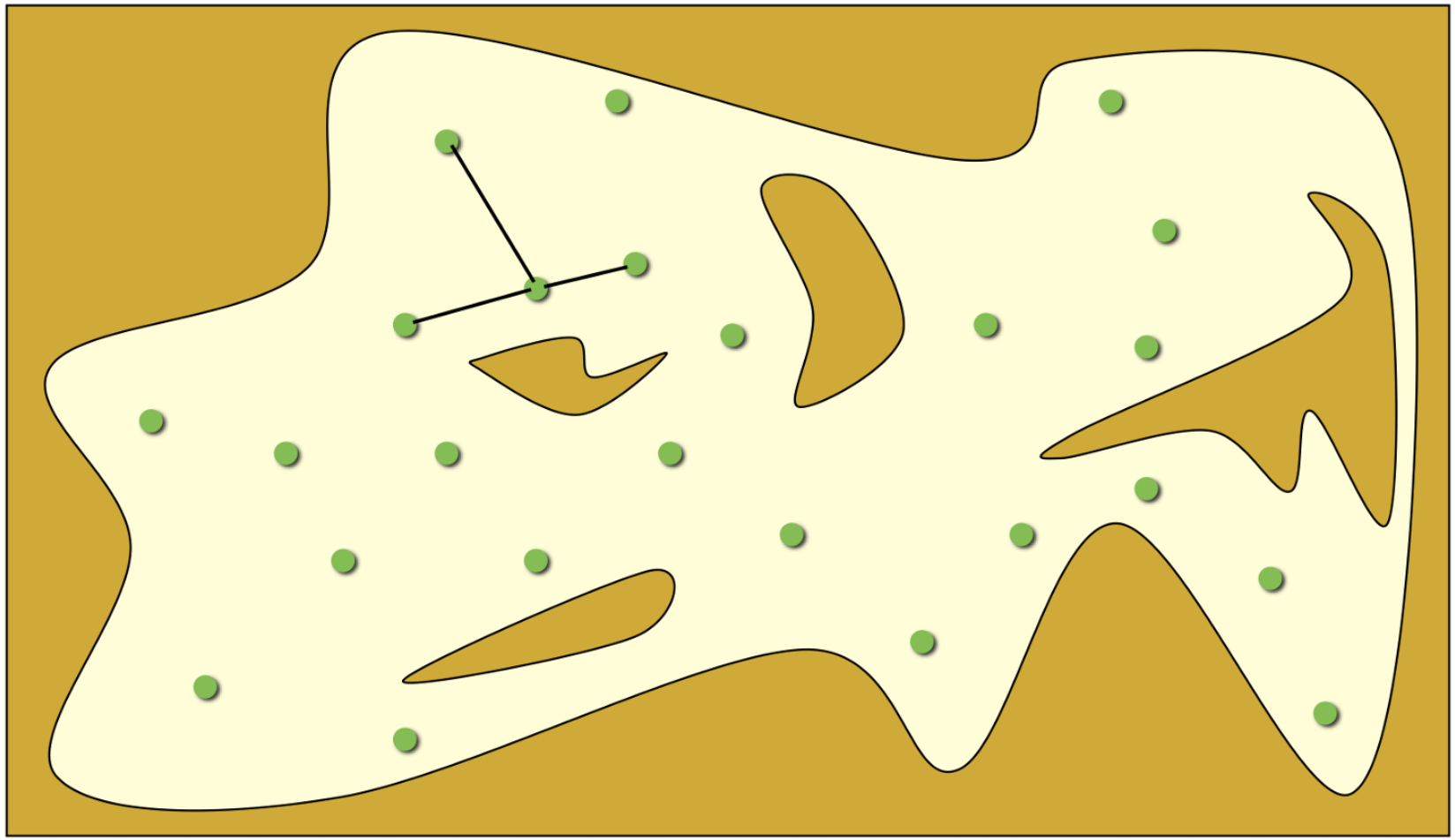
Sampled configurations are tested for collisions



[Source: CS287, Pieter Abbeel, UC Berkeley]

Probabilistic Road Map (PRM)

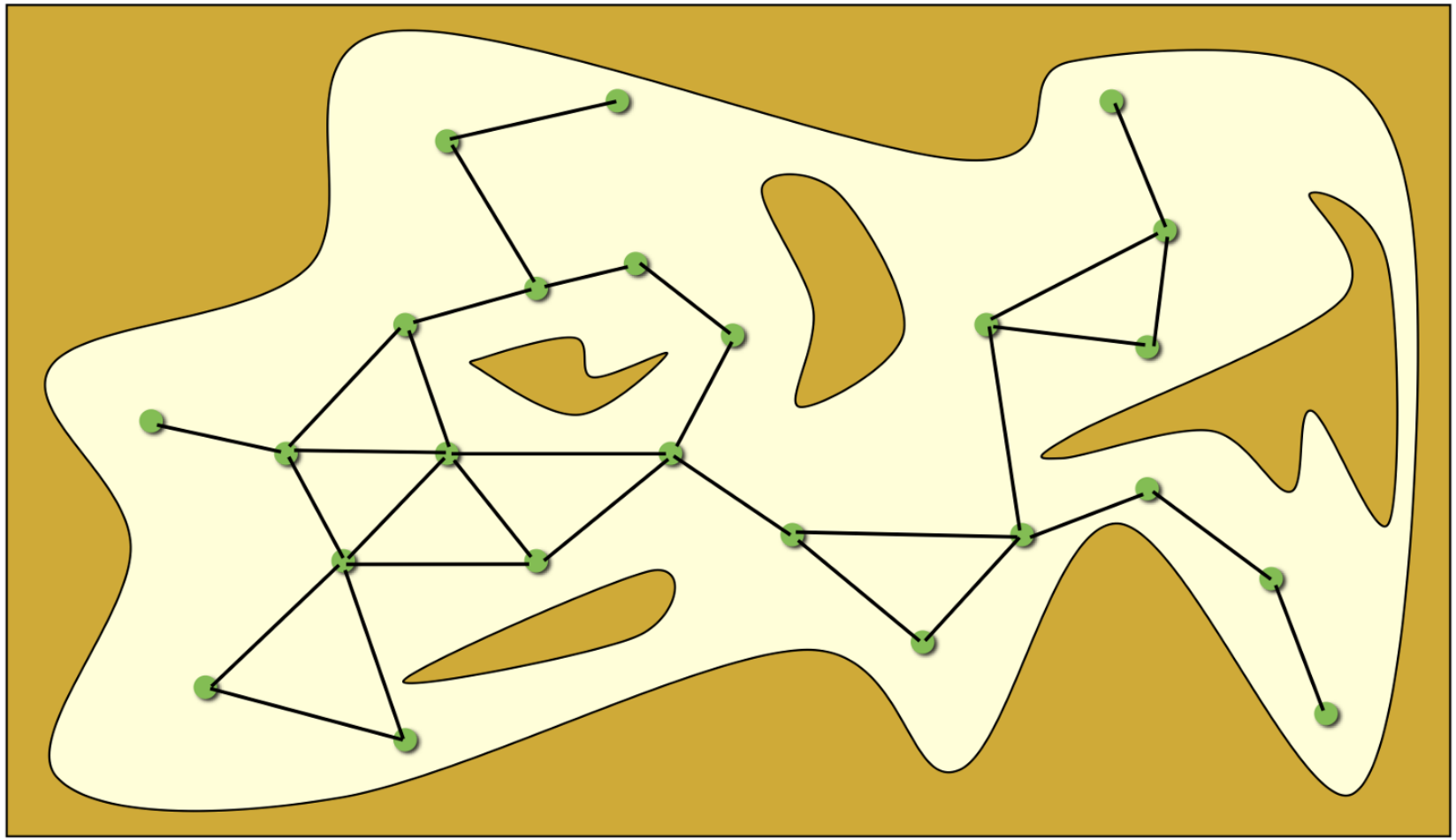
PRM is searched for a path from start (s) to goal (g)



[Source: CS287, Pieter Abbeel, UC Berkeley]

Probabilistic Road Map (PRM)

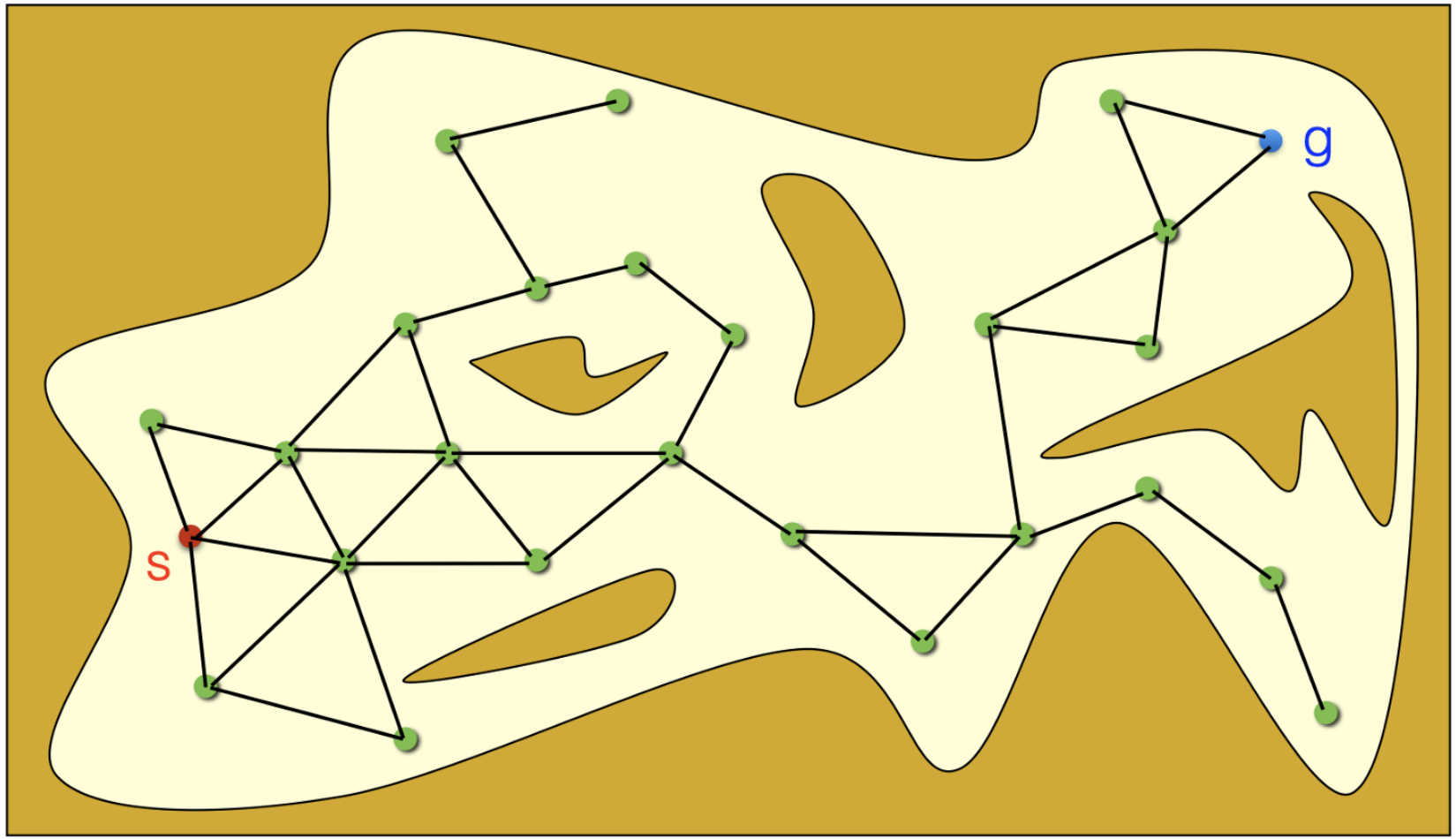
Collision-free edges are retained as local paths to form PRM



[Source: CS287, Pieter Abbeel, UC Berkeley]

Probabilistic Road Map (PRM)

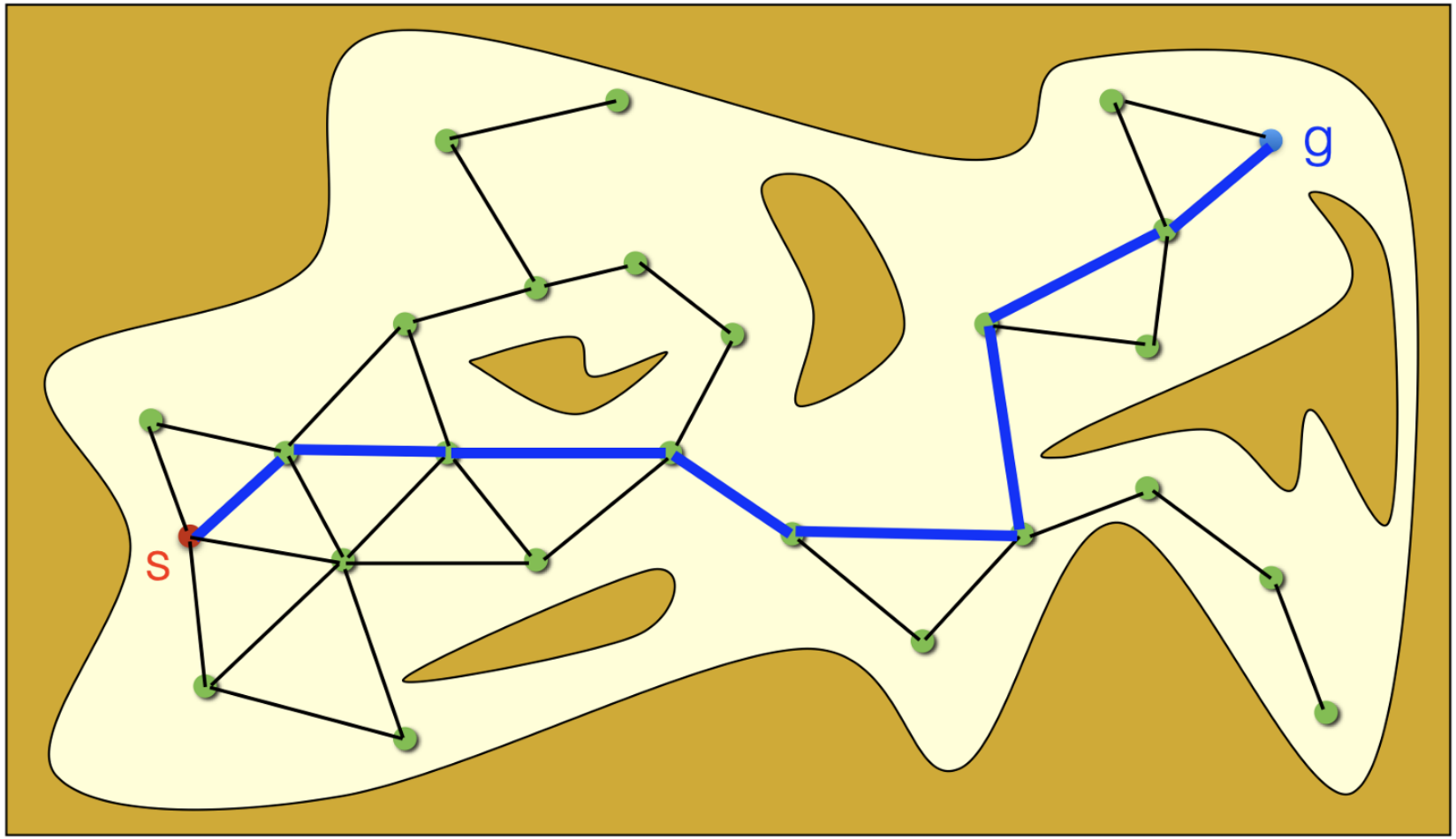
Start and goal configurations are included as milestones



[Source: CS287, Pieter Abbeel, UC Berkeley]

Probabilistic Road Map (PRM)

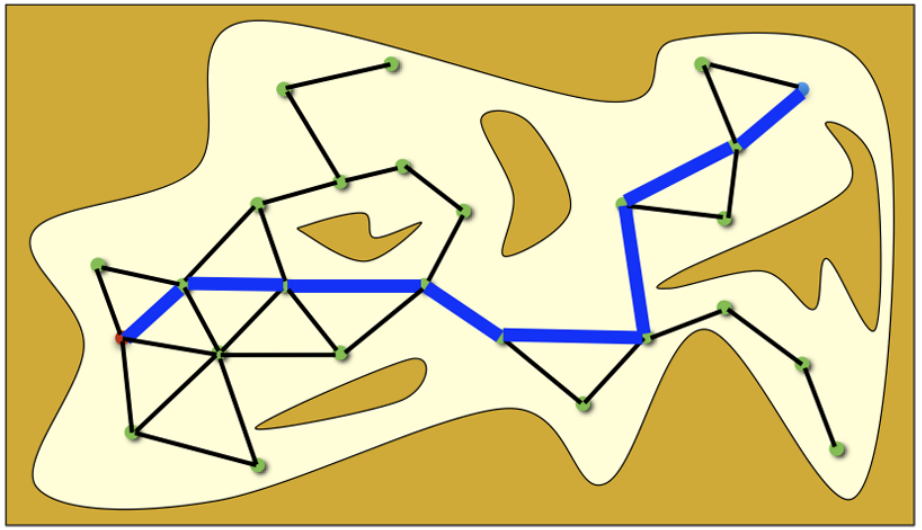
Collision-free edges are retained as local paths to form PRM



[Source: CS287, Pieter Abbeel, UC Berkeley]

Probabilistic Road Map (PRM)

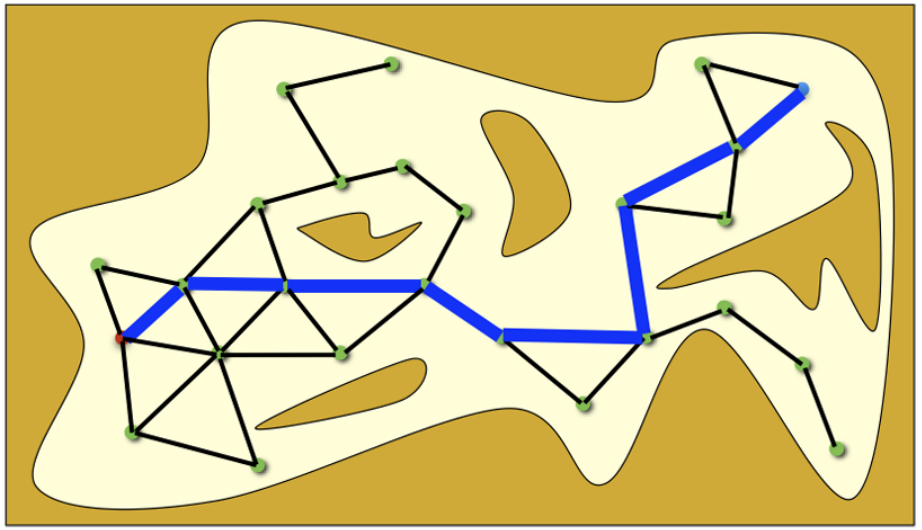
- Recap
  - Randomly sample configurations from C-Space
  - Connect them to nearest neighbors (if no collisions with obstacles)
  - Two primitive procedures (workspace collision query):
    - Check if configuration is in free space
    - Check if an edge is in free space
- PRM can be used for multiple queries



How is this different from vanilla search problems?

Probabilistic Road Map (PRM)

- Recap
  - Randomly sample configurations from C-Space
  - Connect them to nearest neighbors (if no collisions with obstacles)
  - Two primitive procedures (workspace collision query):
    - Check if configuration is in free space
    - Check if an edge is in free space
- PRM can be used for multiple queries



How is this different from vanilla search problems?

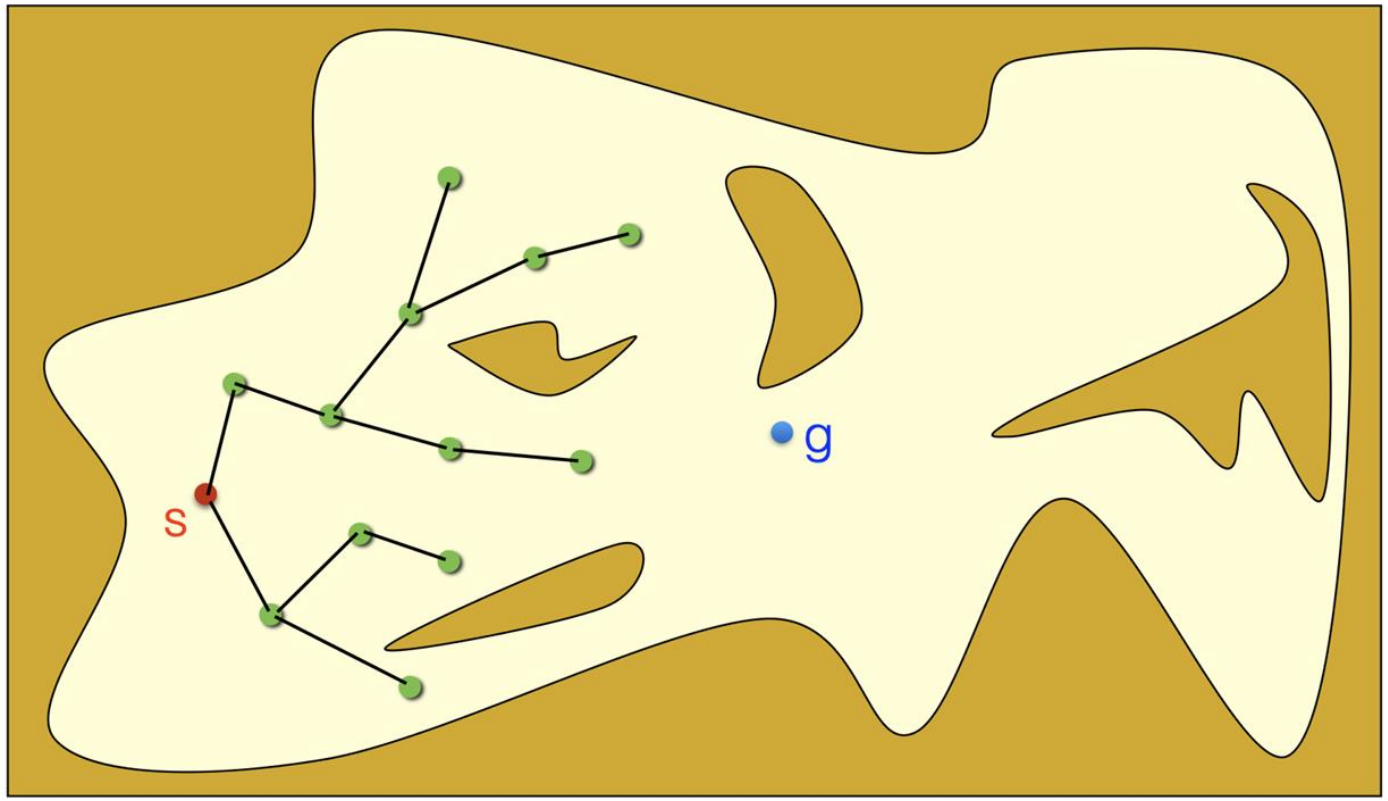
- Uncountably infinite state space (hence the need for sampling)

- Connectivity needs to be computed on the fly

+ Known metric as a starting point

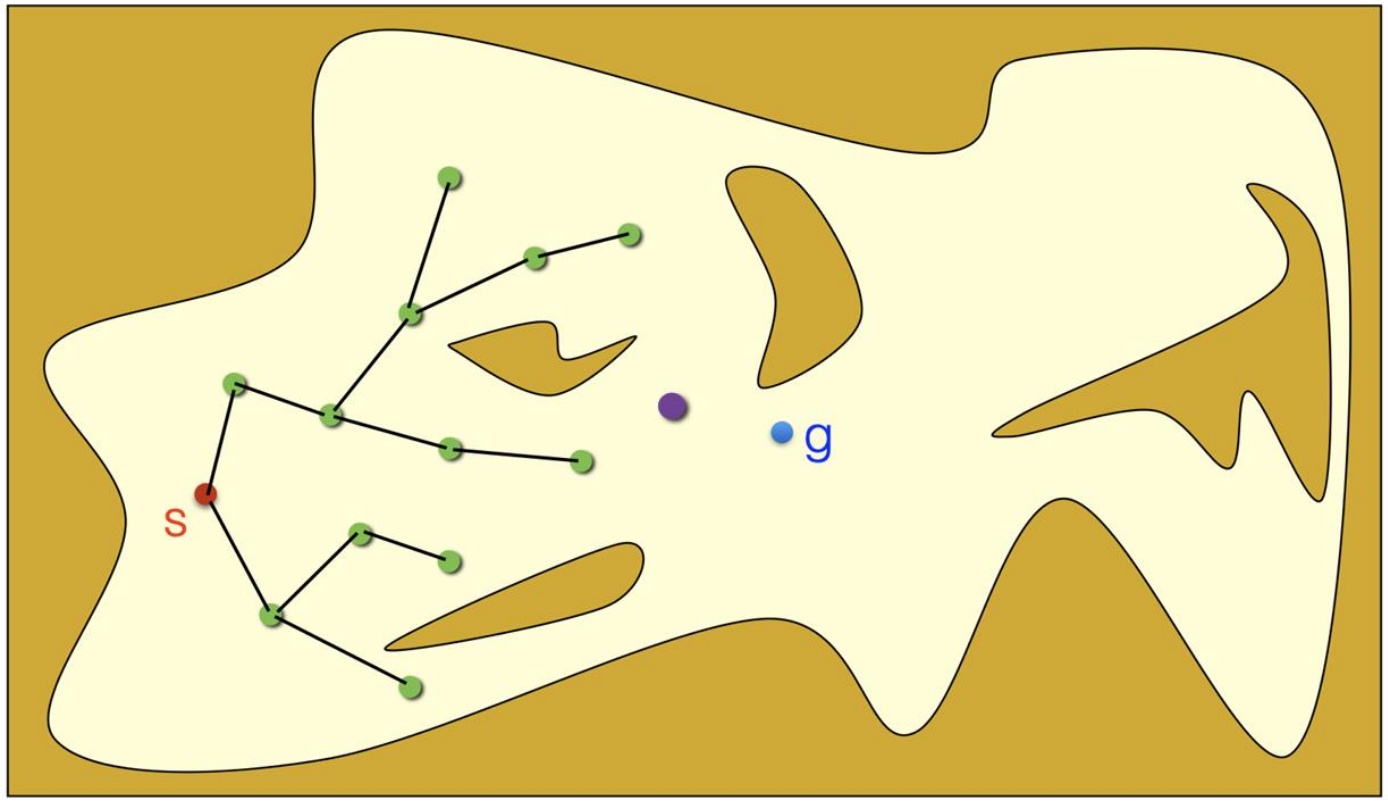
Rapidly-Exploring Random Trees (RRT)

1. $x \leftarrow$ `Sample`$()$ // Sample configuration from C-space
2. $v \leftarrow$ `Nearest`(T, x) // Find nearest node in the tree to sample
3. $v' \leftarrow$ `Extend`(v, x) // Try extension nearest node towards sample
4. If (`ObstacleFree`(v, v')) then // If extension does not collide with obstacles
5. $\quad V \leftarrow V \cup \{v'\}; E \leftarrow E \cup \{(v, v')\}$ // Add new node and edge to the tree



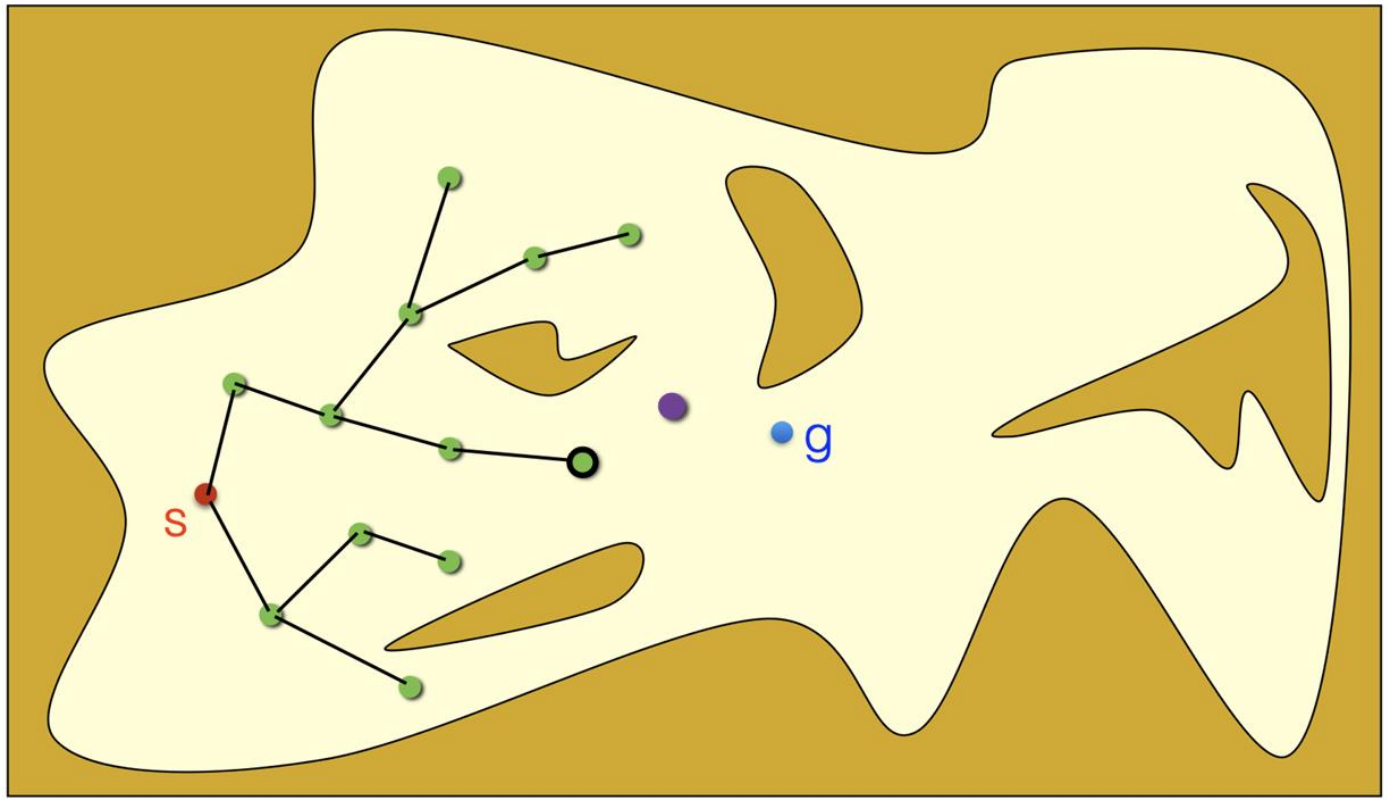
Rapidly-Exploring Random Trees (RRT)

1. $x \leftarrow$ `Sample()` // Sample configuration from C-space
2. $v \leftarrow$ `Nearest`(T, x) // Find nearest node in the tree to sample
3. $v' \leftarrow$ `Extend`(v, x) // Try extension nearest node towards sample
4. If (`ObstacleFree`(v, v')) then // If extension does not collide with obstacles
5. $V \leftarrow V \cup \{v'\}; E \leftarrow E \cup \{(v, v')\}$ // Add new node and edge to the tree



Rapidly-Exploring Random Trees (RRT)

1. $x \leftarrow$ `Sample()` // Sample configuration from C-space
2. $v \leftarrow$ `Nearest`(T, x) // Find nearest node in the tree to sample
3. $v' \leftarrow$ `Extend`(v, x) // Try extension nearest node towards sample
4. If (`ObstacleFree`(v, v')) then // If extension does not collide with obstacles
5. $V \leftarrow V \cup \{v'\}; E \leftarrow E \cup \{(v, v')\}$ // Add new node and edge to the tree



Rapidly-Exploring Random Trees (RRT)

1. $x \leftarrow$ `Sample`$()$
2. $v \leftarrow$ `Nearest`(T, x)
3. $v' \leftarrow$ `Extend`(v, x)
4. If (`ObstacleFree`(v, v')) then
5. $\quad V \leftarrow V \cup \{v'\}; E \leftarrow E \cup \{(v, v')\}$

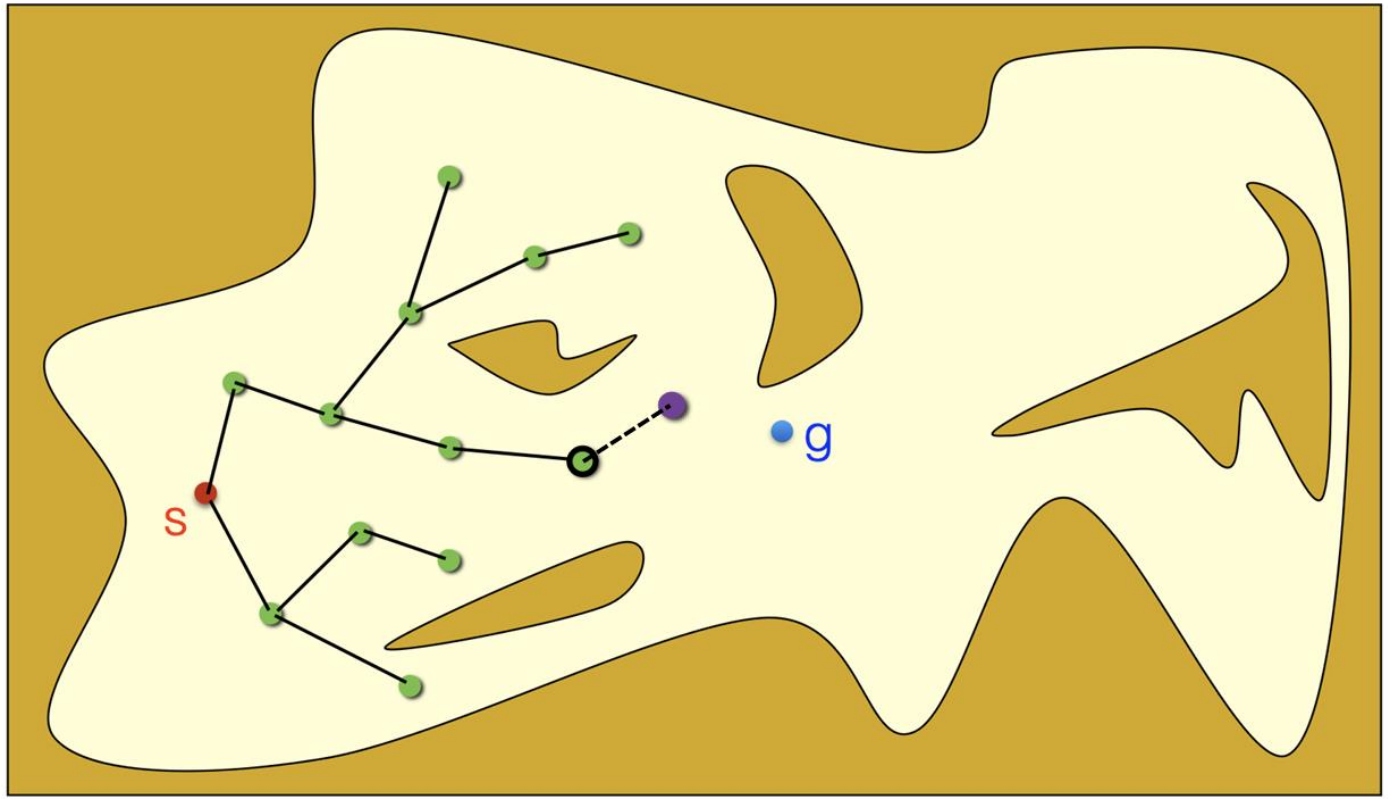
// Sample configuration from C-space

// Find nearest node in the tree to sample

// Try extension nearest node towards sample

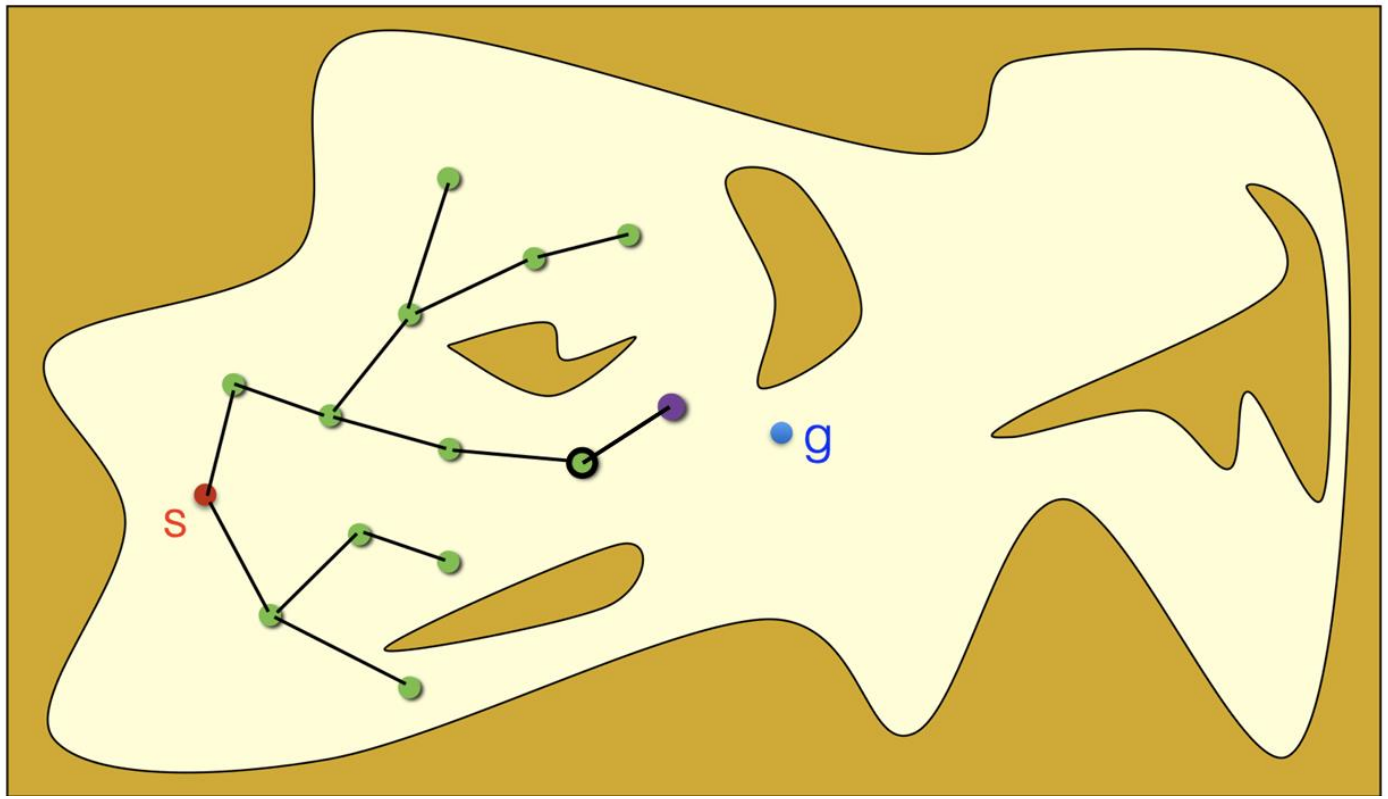
// If extension does not collide with obstacles

// Add new node and edge to the tree



Rapidly-Exploring Random Trees (RRT)

1. $x \leftarrow$ `Sample()` // Sample configuration from C-space
2. $v \leftarrow$ `Nearest`(T, x) // Find nearest node in the tree to sample
3. $v' \leftarrow$ `Extend`(v, x) // Try extension nearest node towards sample
4. If (`ObstacleFree`(v, v')) then // If extension does not collide with obstacles
5. $V \leftarrow V \cup \{v'\}; E \leftarrow E \cup \{(v, v')\}$ // Add new node and edge to the tree



Rapidly-Exploring Random Trees (RRT)

1. $x \leftarrow$ `Sample()` // Sample configuration from C-space
2. $v \leftarrow$ `Nearest`(T, x) // Find nearest node in the tree to sample
3. $v' \leftarrow$ `Extend`(v, x) // Try extension nearest node towards sample
4. If (`ObstacleFree`(v, v')) then // If extension does not collide with obstacles
5. $V \leftarrow V \cup \{v'\}; E \leftarrow E \cup \{(v, v')\}$ // Add new node and edge to the tree