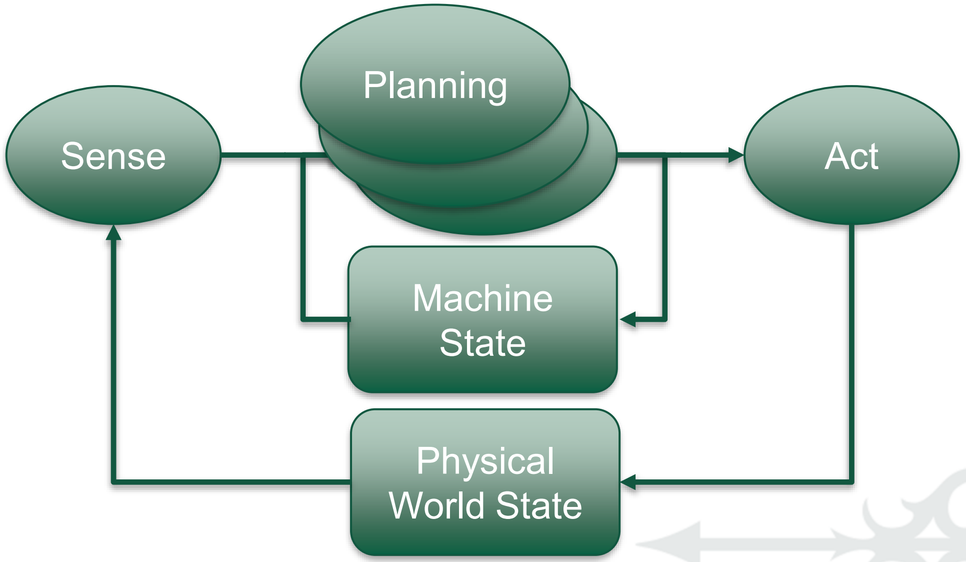
Planning

CSCI 420-04 Robotics

WILLIAM & MARY

CHARTERED 1693

How do we choose the action?

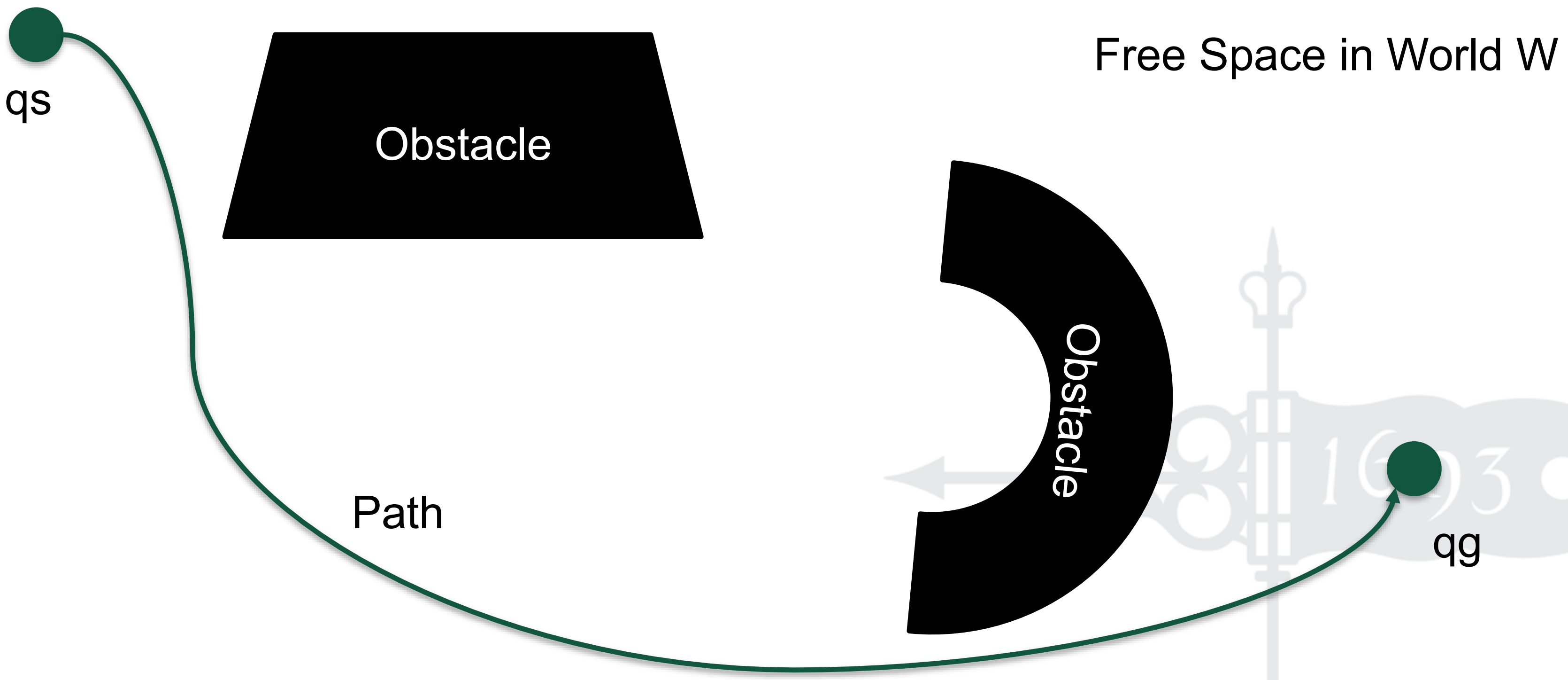


2D Motion Planning

- Given:
  - World Space W
  - Obstacle Areas O
  - Robot State R
  - Start qs
  - End qg

- Find:
  - Path from qs to qg
  - Inside of W
  - Avoiding O
  - (with efficiency)

Motion Planning



Model Planning Families

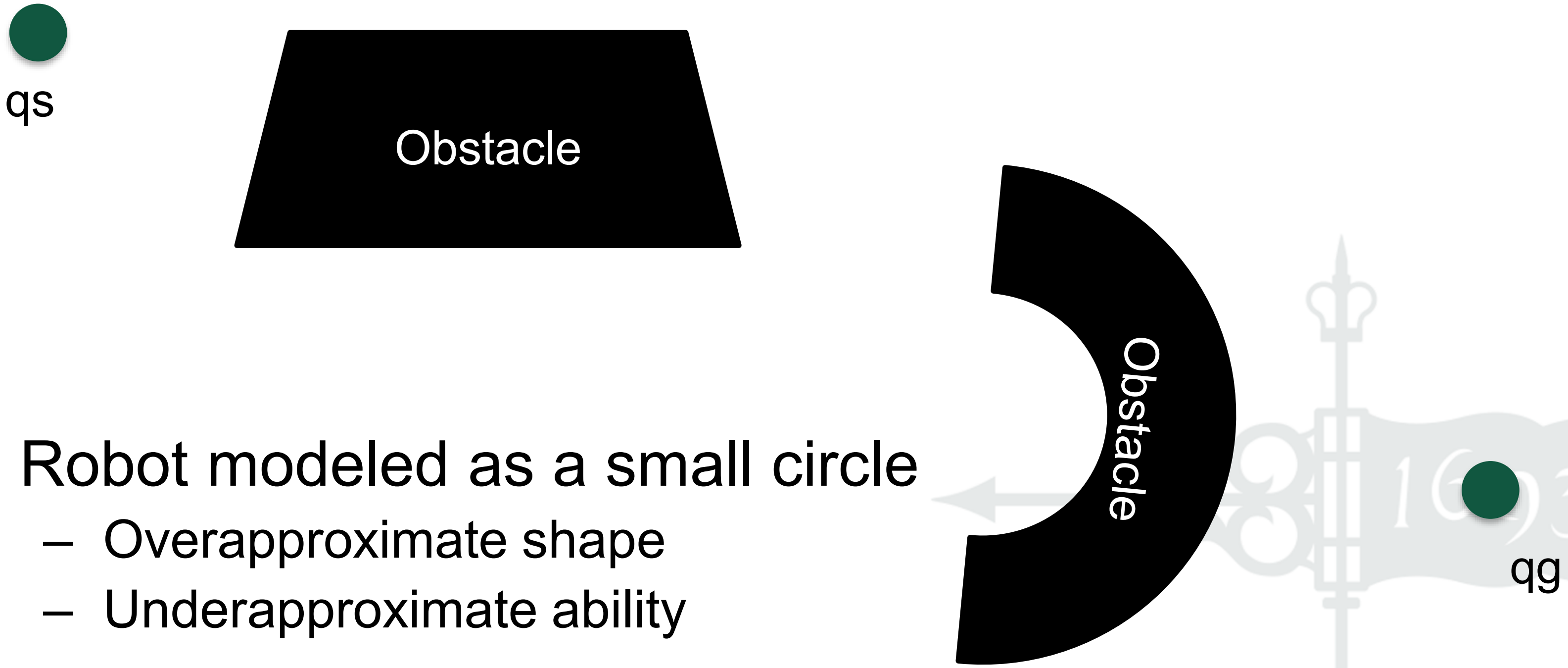
- Reactive
- Model-based

The right choice depends on the assumptions about sensor types and the available world models

Motion Planning Families

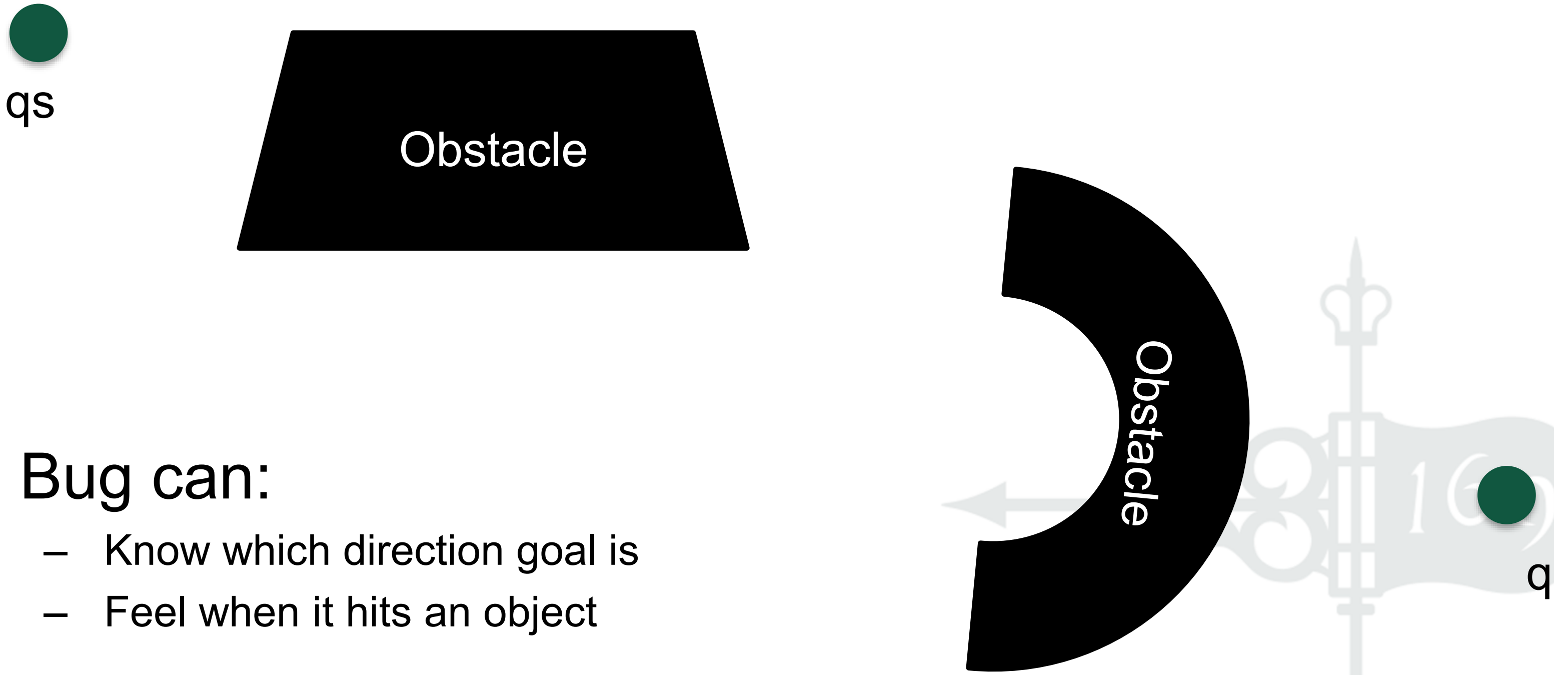
- Reactive
  - Online
  - Fast
  - Non-optimal

Reactive: Bug Algorithms



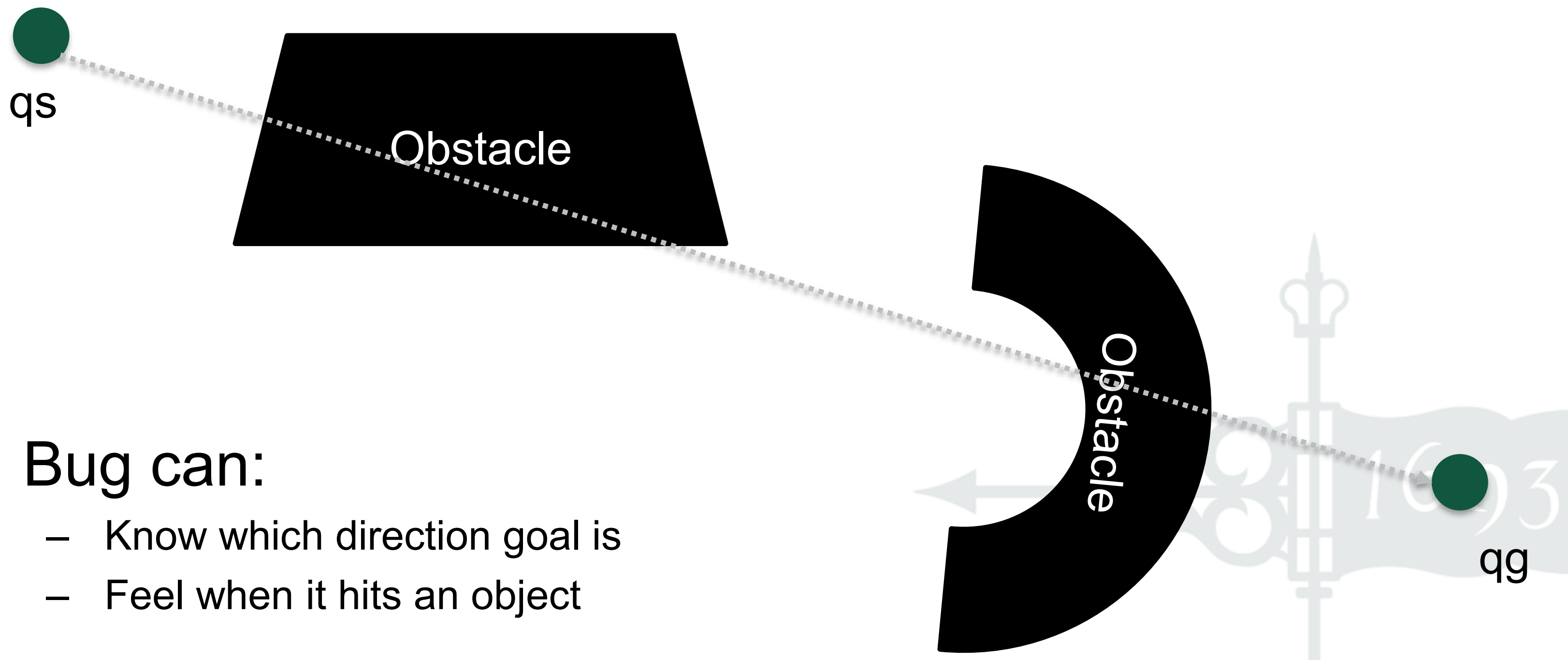
- Robot modeled as a small circle
  - Overapproximate shape
  - Underapproximate ability

Reactive: Bug 1



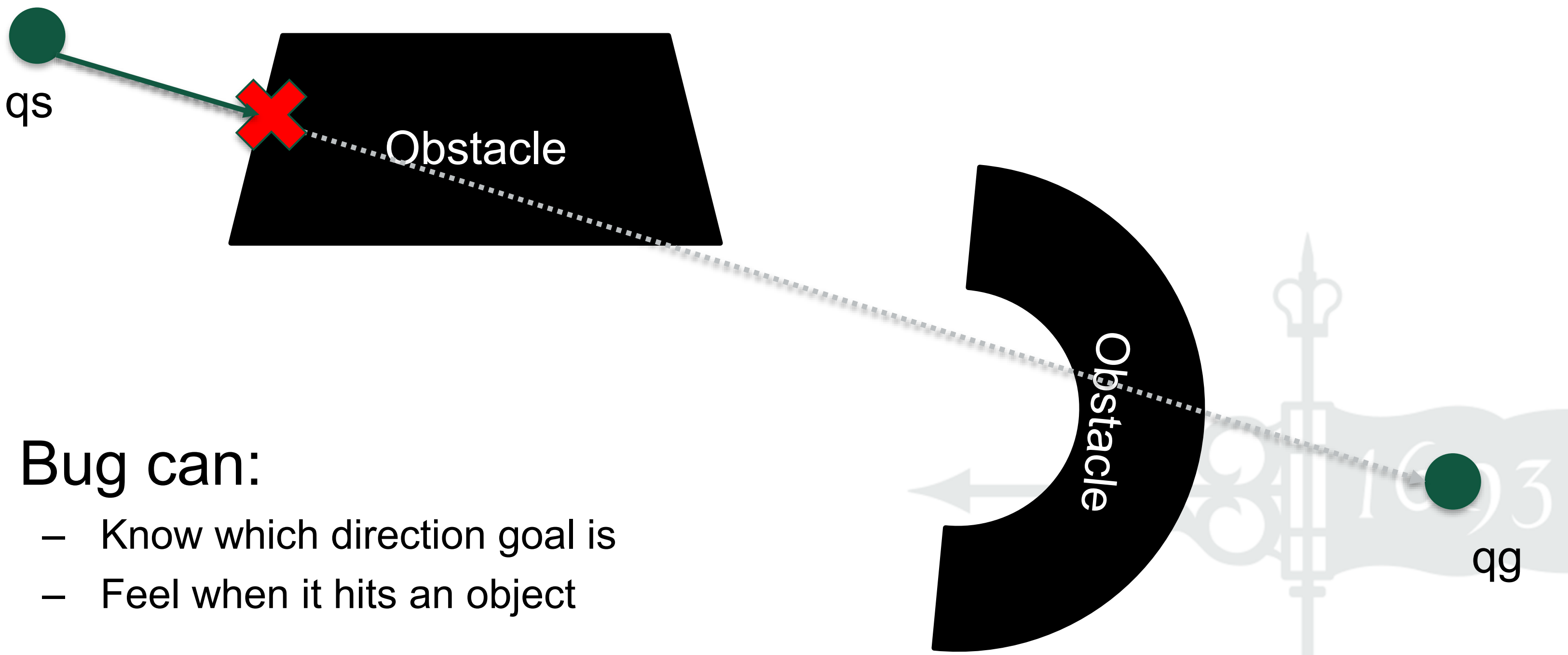
- Bug can:
  - Know which direction goal is
  - Feel when it hits an object

Reactive: Bug 1



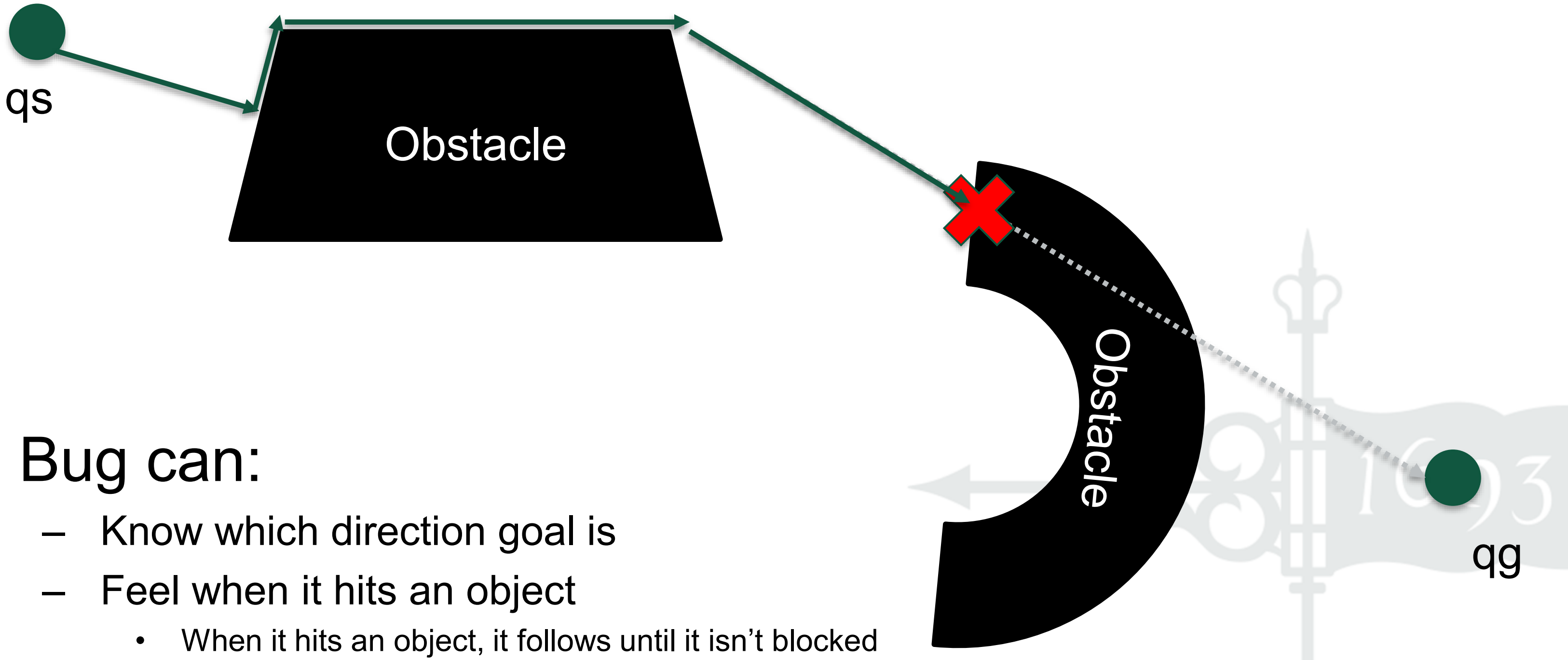
- Bug can:
  - Know which direction goal is
  - Feel when it hits an object

Reactive: Bug 1



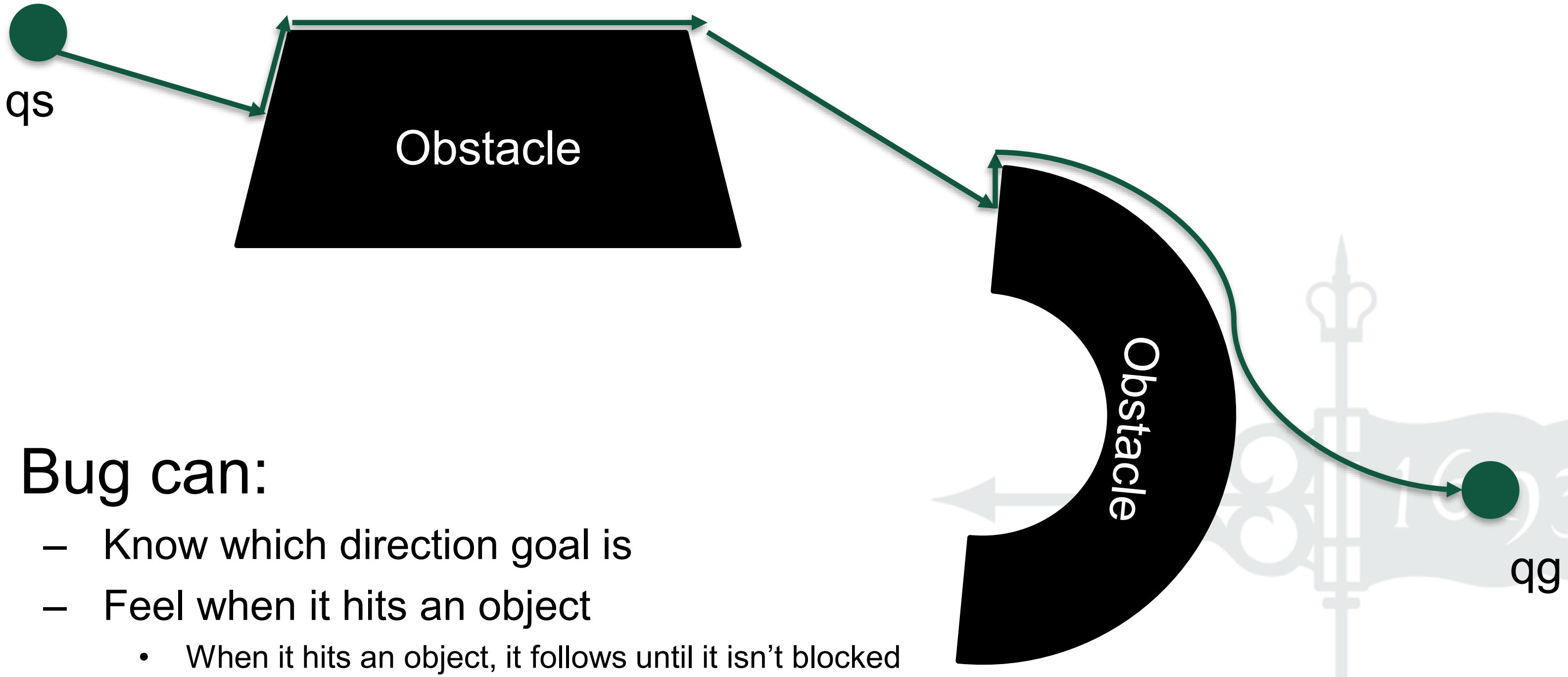
- Bug can:
  - Know which direction goal is
  - Feel when it hits an object

Reactive: Bug 1



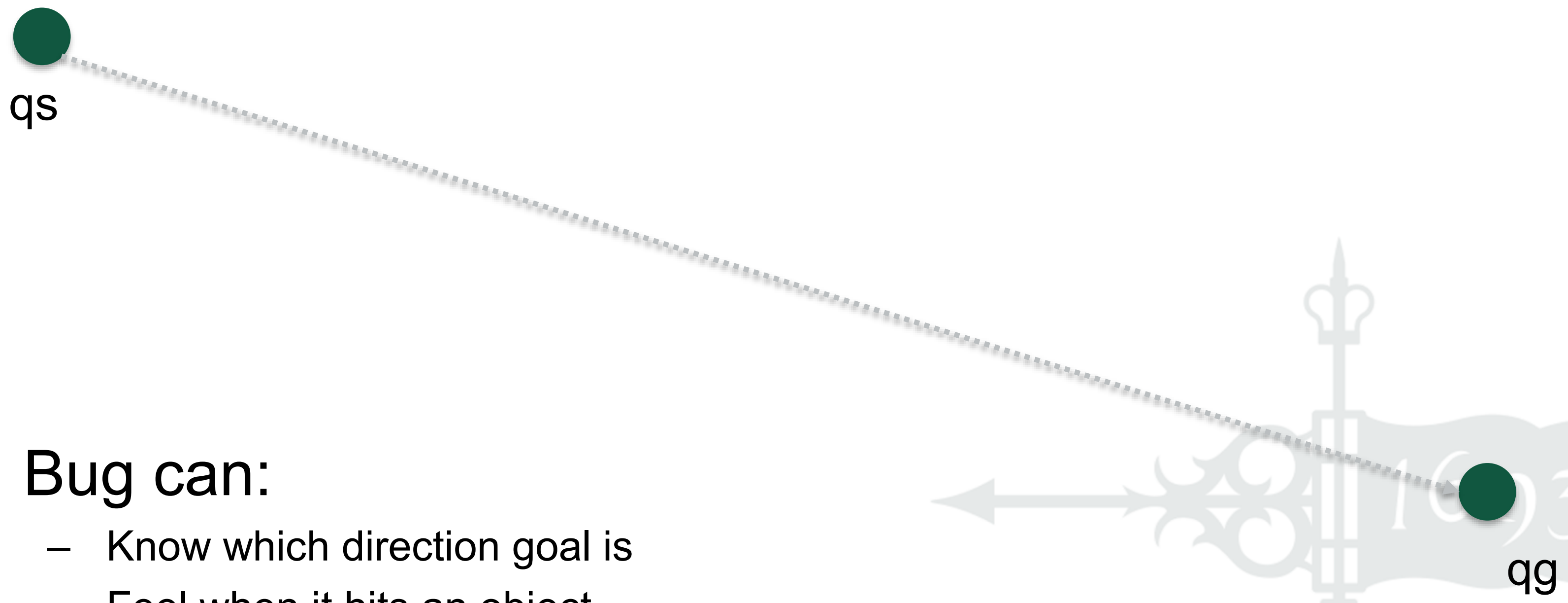
- Bug can:
  - Know which direction goal is
  - Feel when it hits an object
    - When it hits an object, it follows until it isn't blocked

Reactive: Bug 1



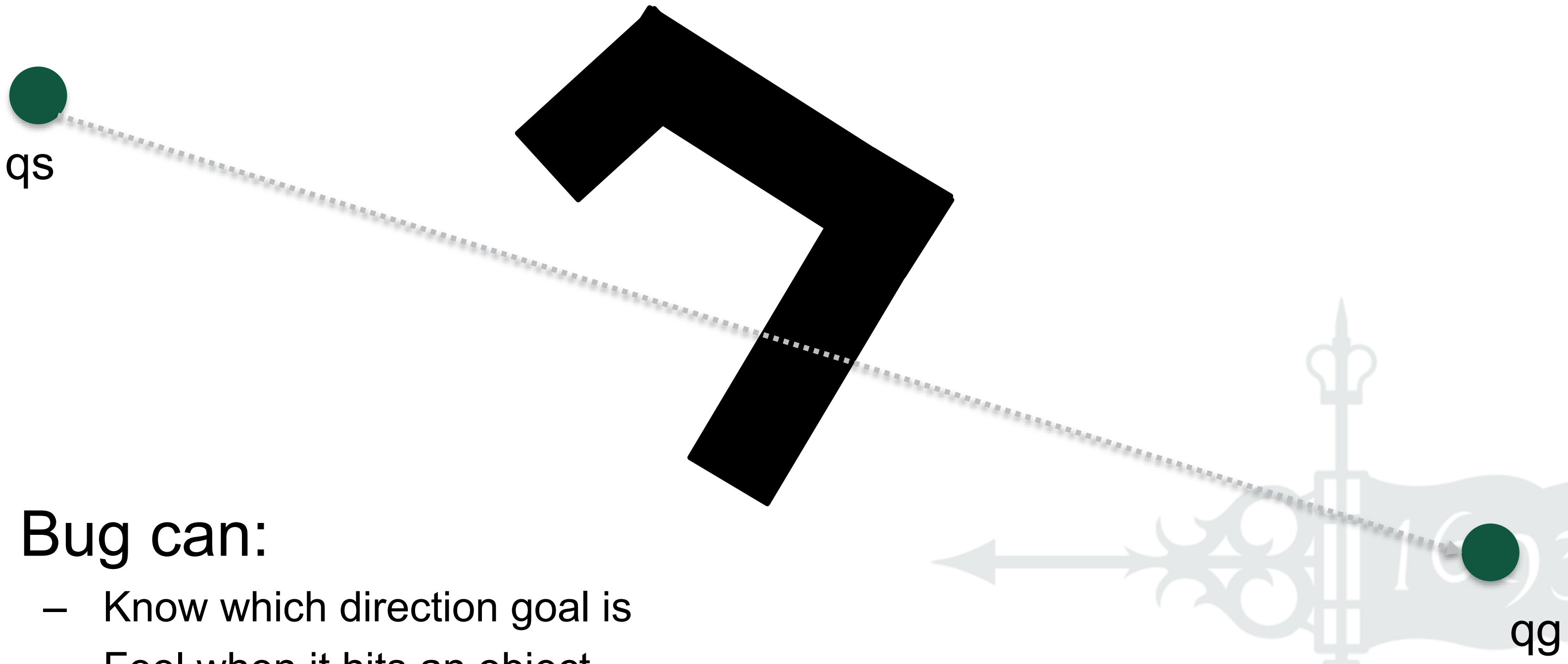
- Bug can:
  - Know which direction goal is
  - Feel when it hits an object
    - When it hits an object, it follows until it isn't blocked

When does Bug 1 Fail?



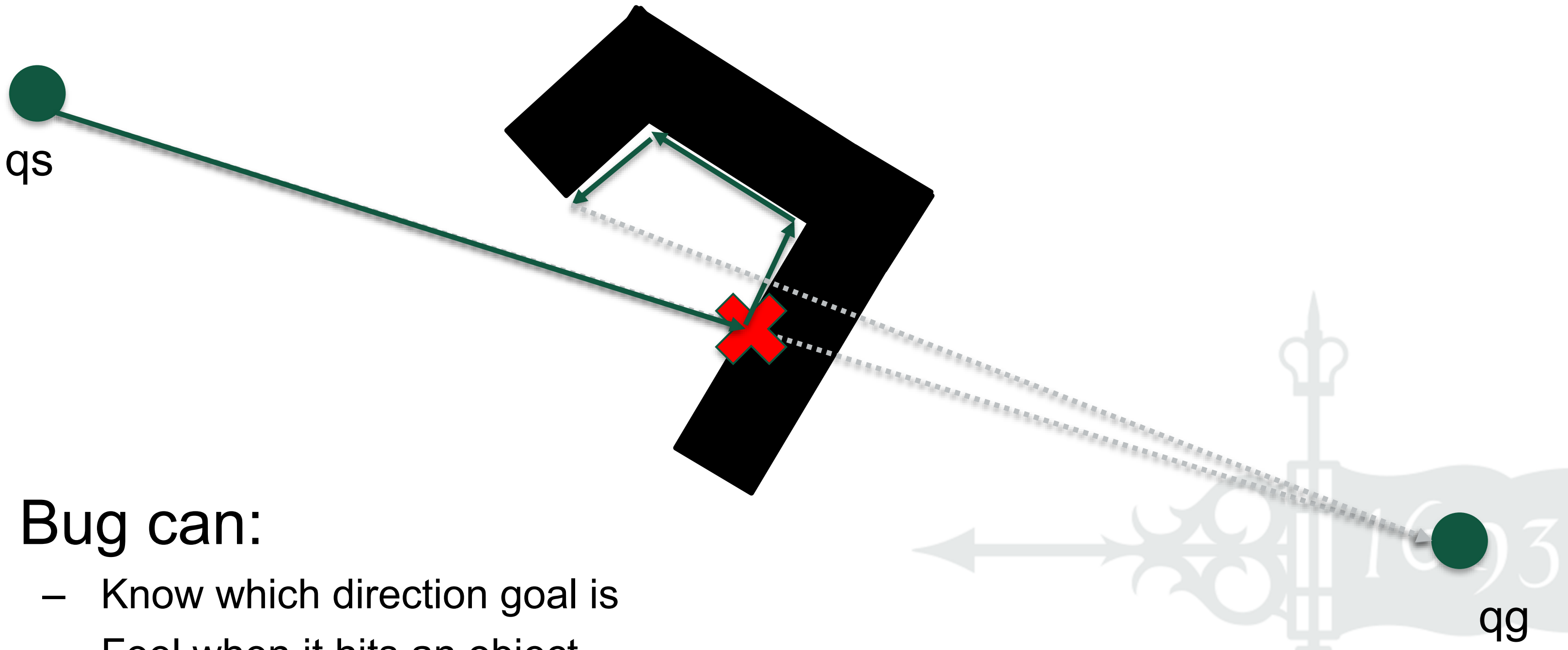
- Bug can:
  - Know which direction goal is
  - Feel when it hits an object
    - When it hits an object, it follows until it isn't blocked

When does Bug 1 Fail?



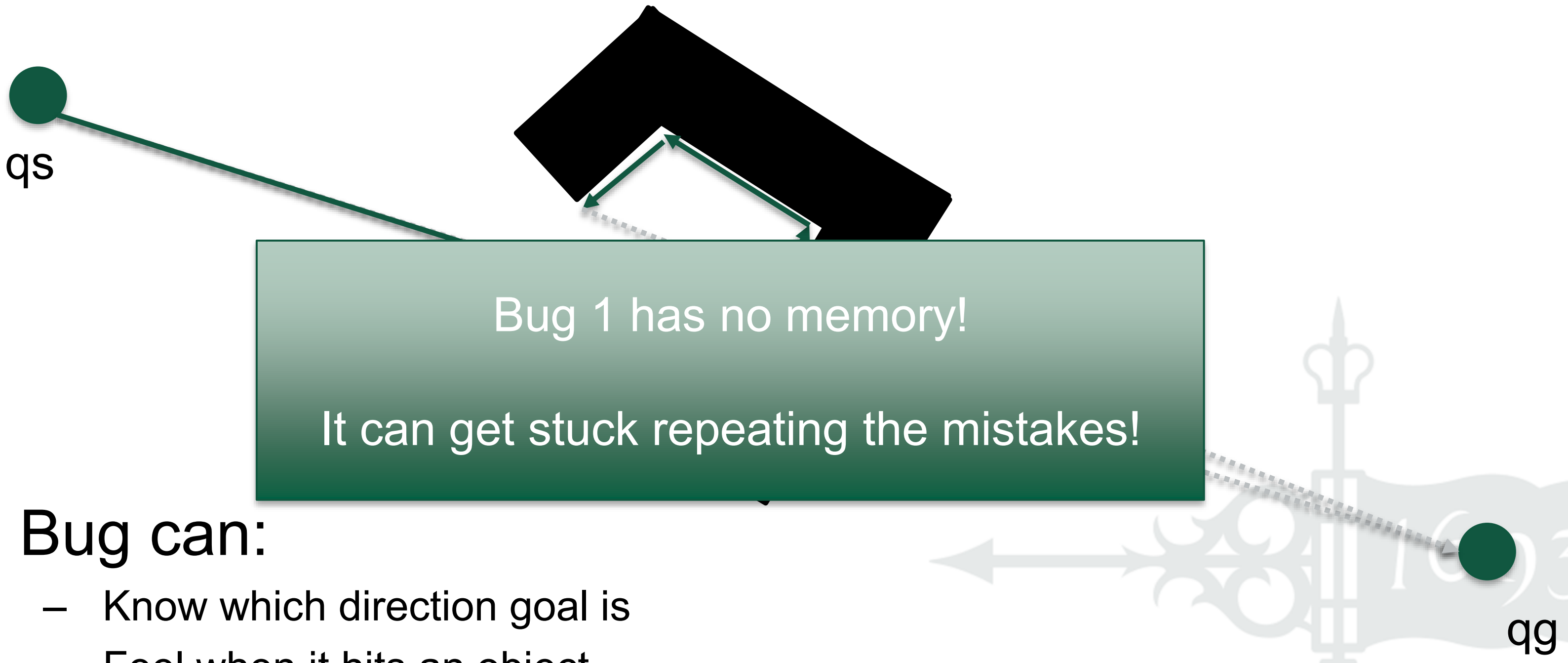
- Bug can:
  - Know which direction goal is
  - Feel when it hits an object
    - When it hits an object, it follows until it isn't blocked

When does Bug 1 Fail?



- Bug can:
  - Know which direction goal is
  - Feel when it hits an object
    - When it hits an object, it follows until it isn't blocked

When does Bug 1 Fail?

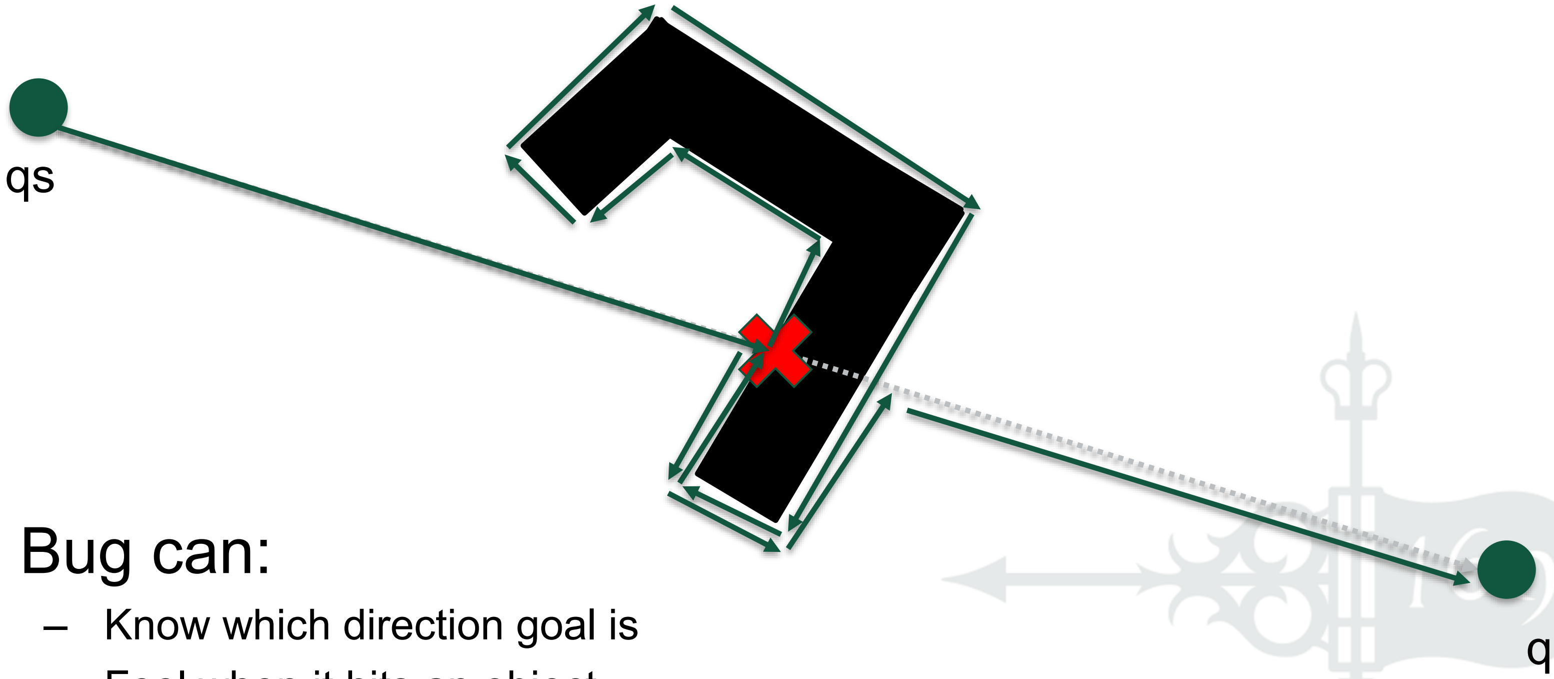


Bug 1 has no memory!

It can get stuck repeating the mistakes!

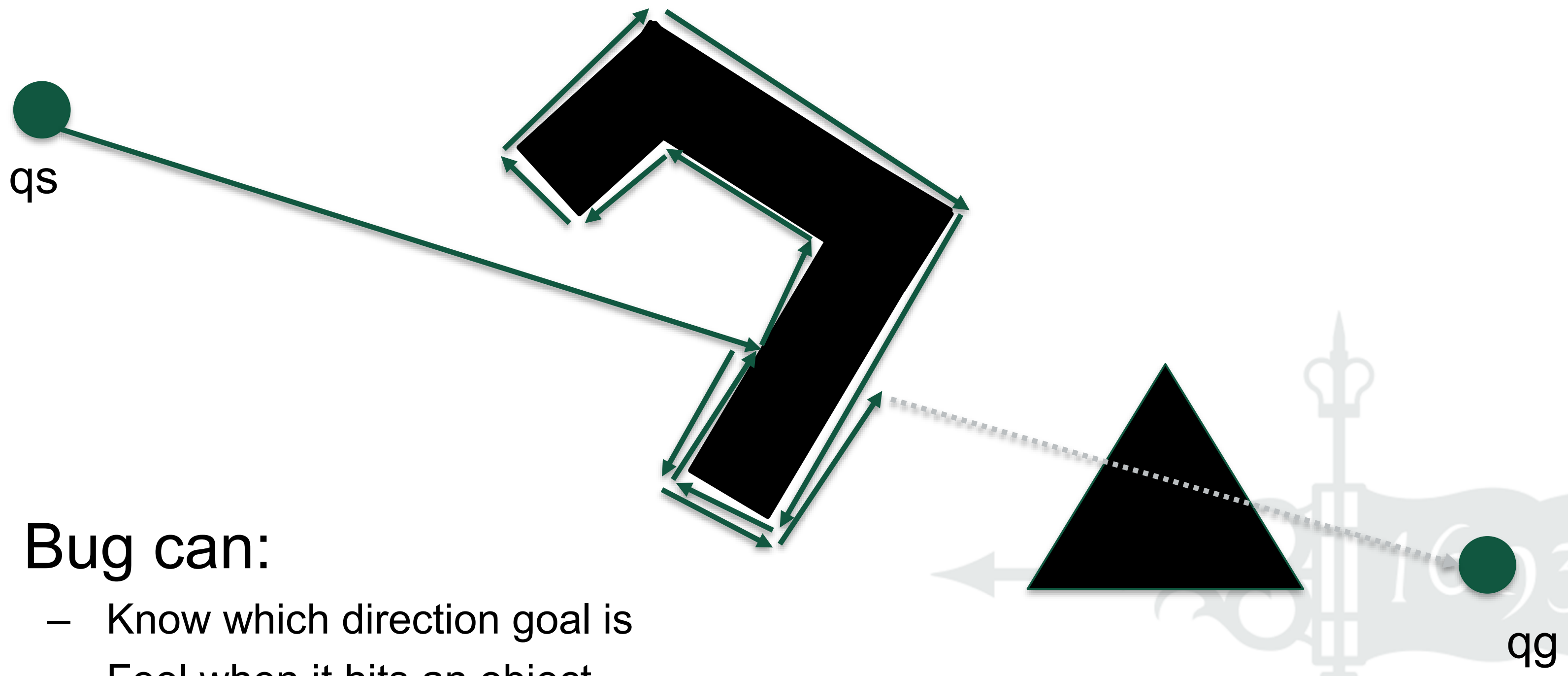
- Bug can:
  - Know which direction goal is
  - Feel when it hits an object
    - When it hits an object, it follows until it isn't blocked

Reaction with memory: Bug 2



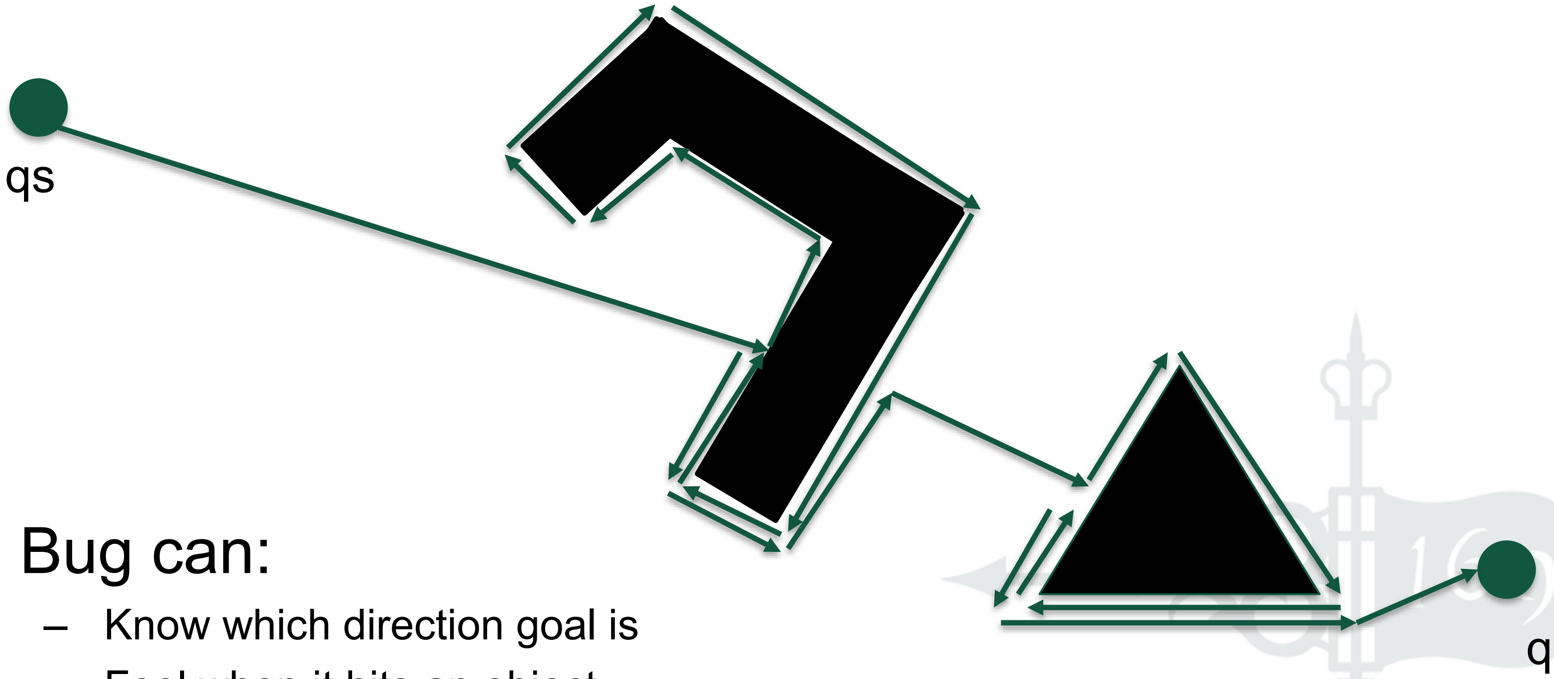
- Bug can:
  - Know which direction goal is
  - Feel when it hits an object
    - Goes all the way around the object to map it
    - Backtracks to the nearest location before going to goal

Reaction with memory: Bug 2



- Bug can:
  - Know which direction goal is
  - Feel when it hits an object
    - Goes all the way around the object to map it
    - Backtracks to the nearest location before going to goal

Reaction with memory: Bug 2



- Bug can:
  - Know which direction goal is
  - Feel when it hits an object
    - Goes all the way around the object to map it
    - Backtracks to the nearest location before going to goal

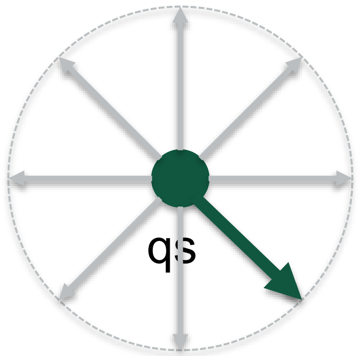
Model Planning Families

- Reactive
  - Bug family
  - Dynamic Window
- Model-based

Dynamic Window

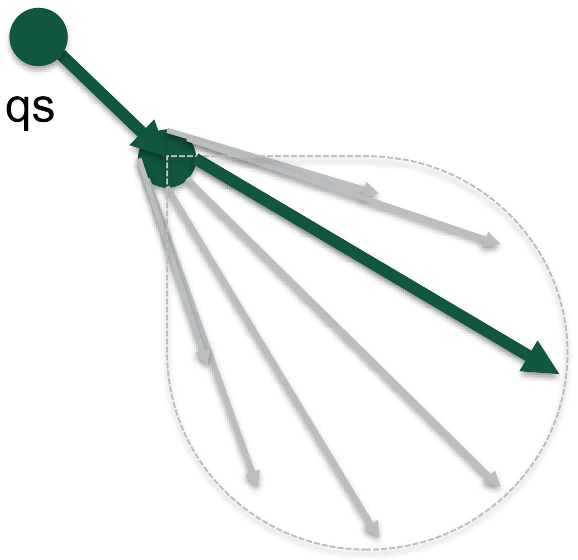
- Robot can make short-horizon plans
- Plans depend on the robot’s dynamics
  - At a stop, it will take time to accelerate
  - At high speeds, the robot can’t turn sharply

Dynamic Windows



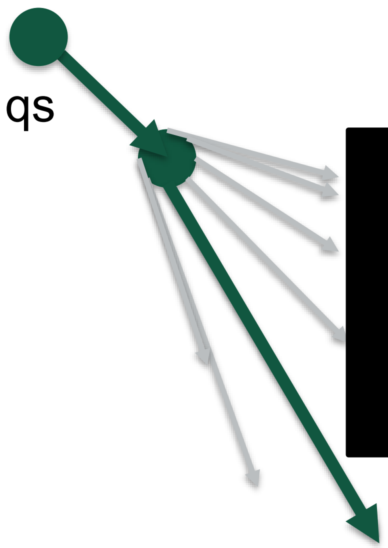
- Identify feasible space
- Generate list of paths
- Pick best path

Dynamic Windows



- Identify feasible space
- Generate list of paths
- Pick best path

Dynamic Windows



- Identify feasible space
- Generate list of paths
- Pick best path
  - Considering Obstacles

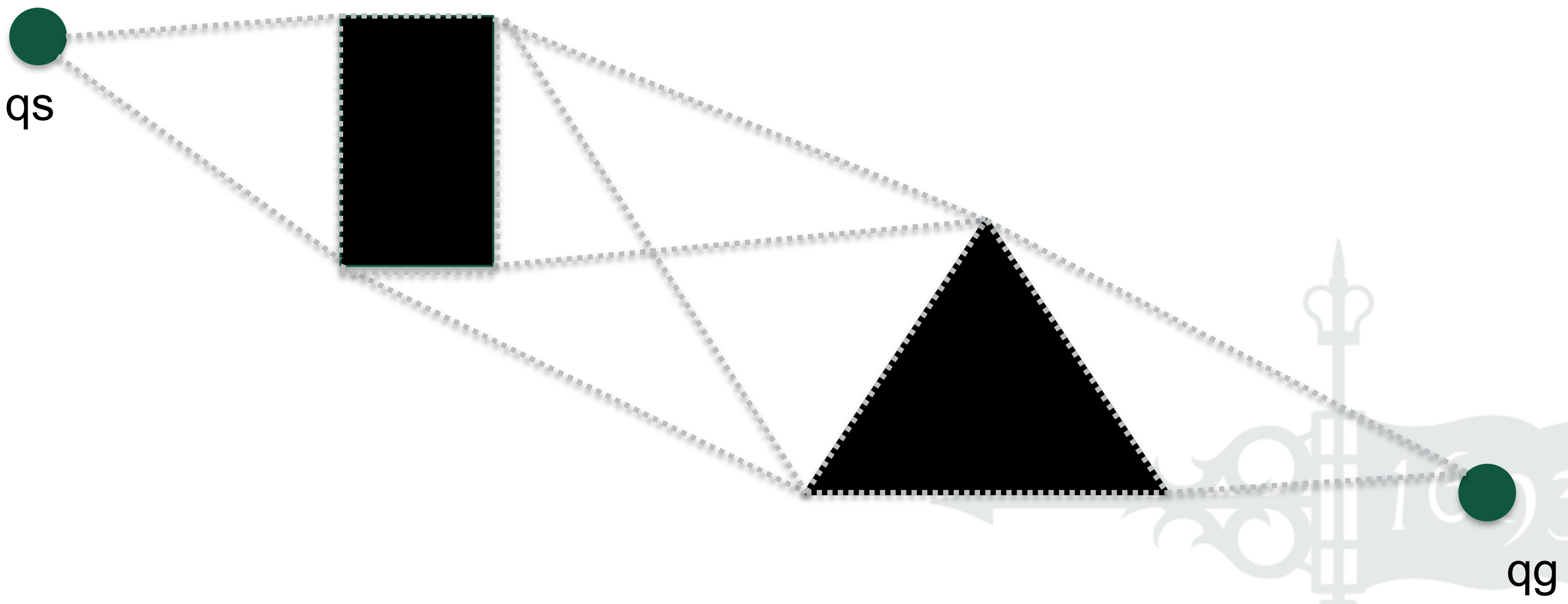
Model Planning Families

- Reactive
  - Bug family
  - Dynamic Window
- Model-based
  - Predictive model of actions in **known world**
  - Simplify world model, search for solution

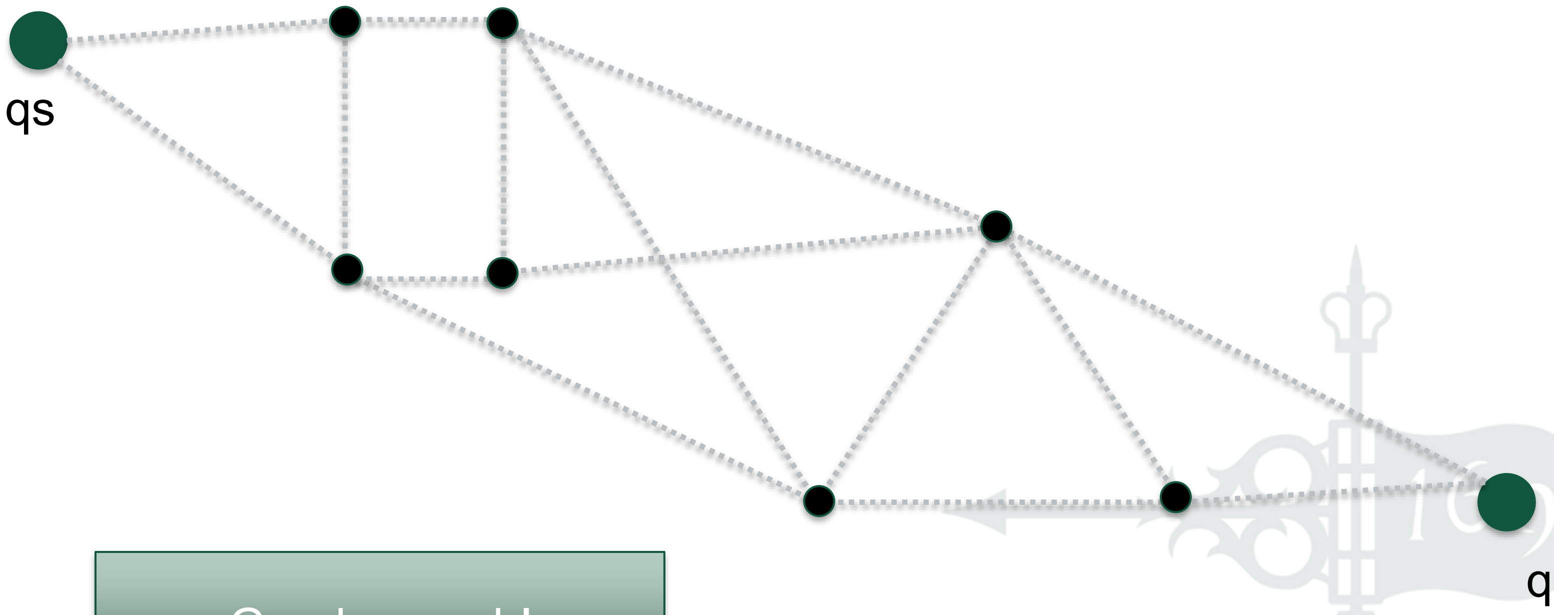
Path Planning: Visibility

- Given:
  - Known location of polygonal obstacles
- Compute:
  - All edges through free space between all vertices
- Find:
  - Shortest path

Path Planning: Visibility

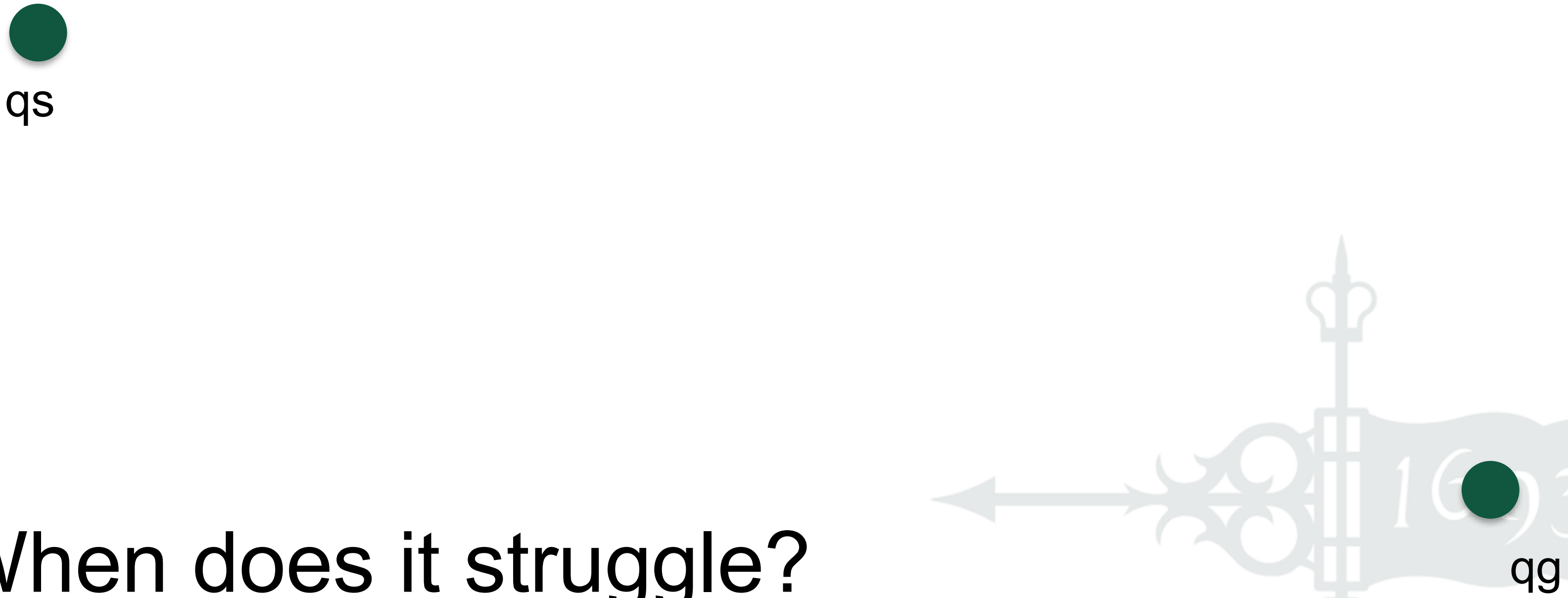


Path Planning: Visibility



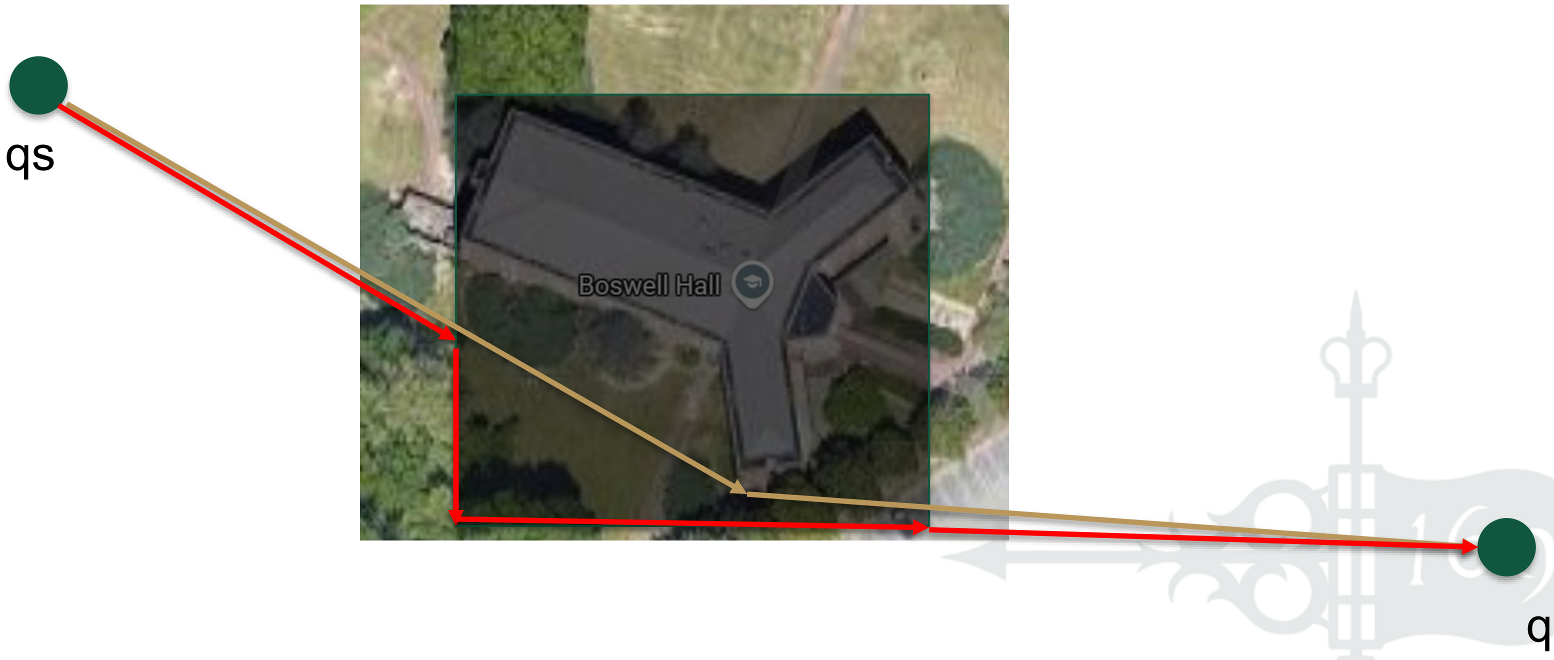
Graph search!

Path Planning: Visibility

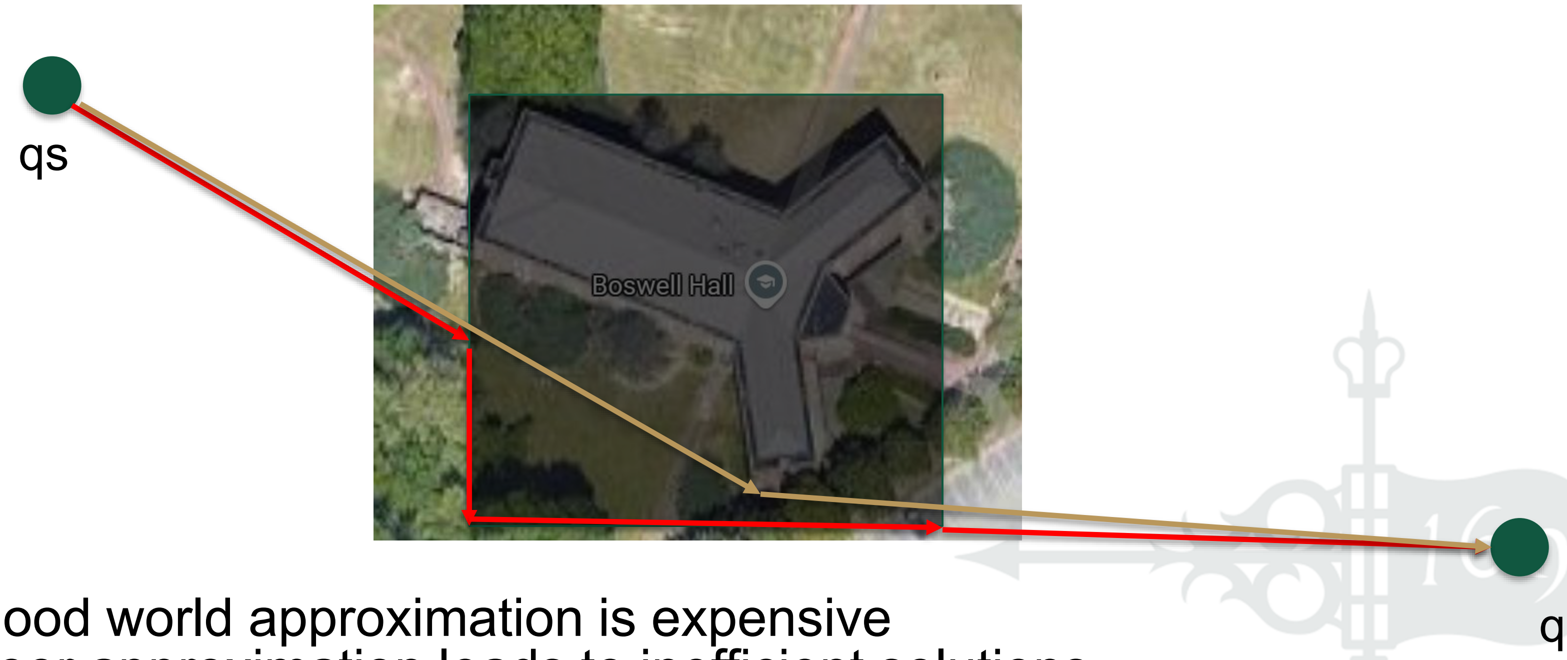


When does it struggle?

Path Planning: Visibility



Path Planning: Visibility

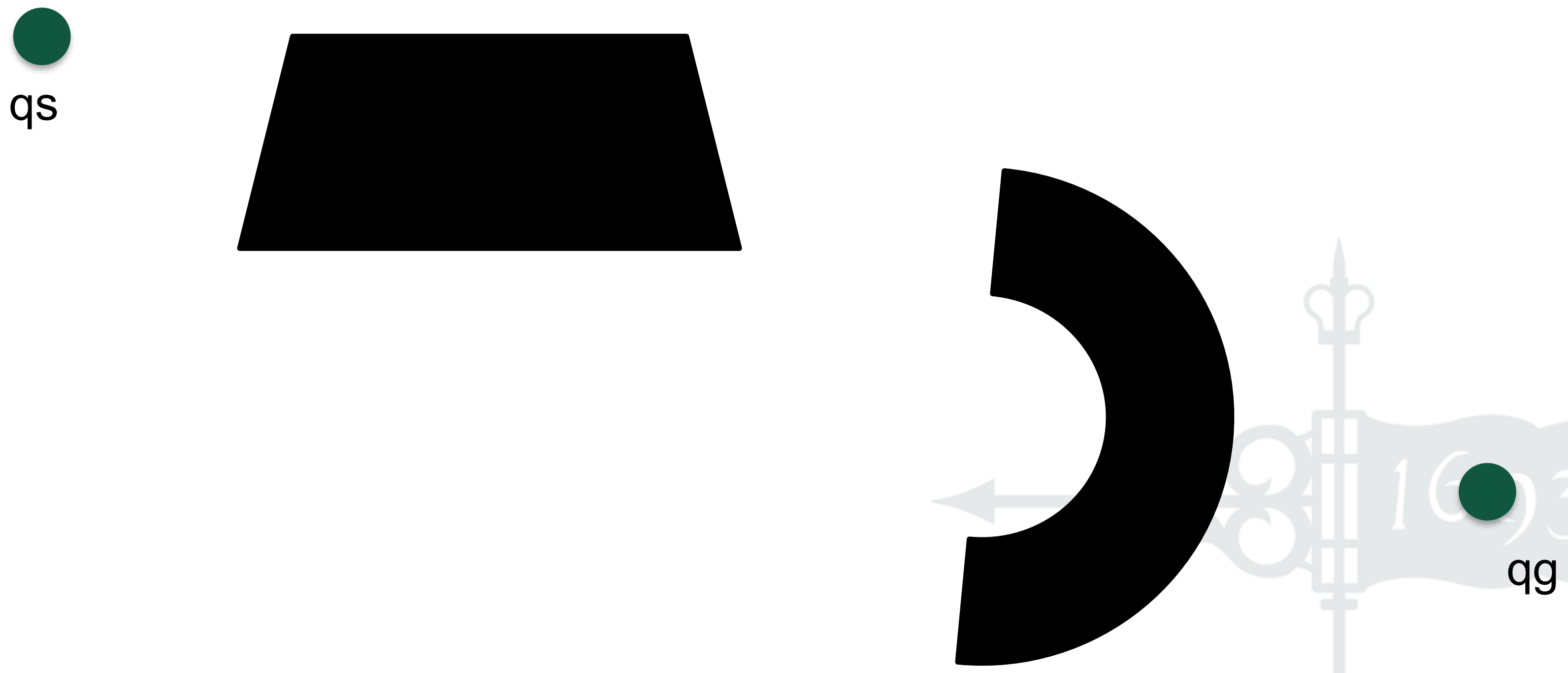


Good world approximation is expensive
Poor approximation leads to inefficient solutions

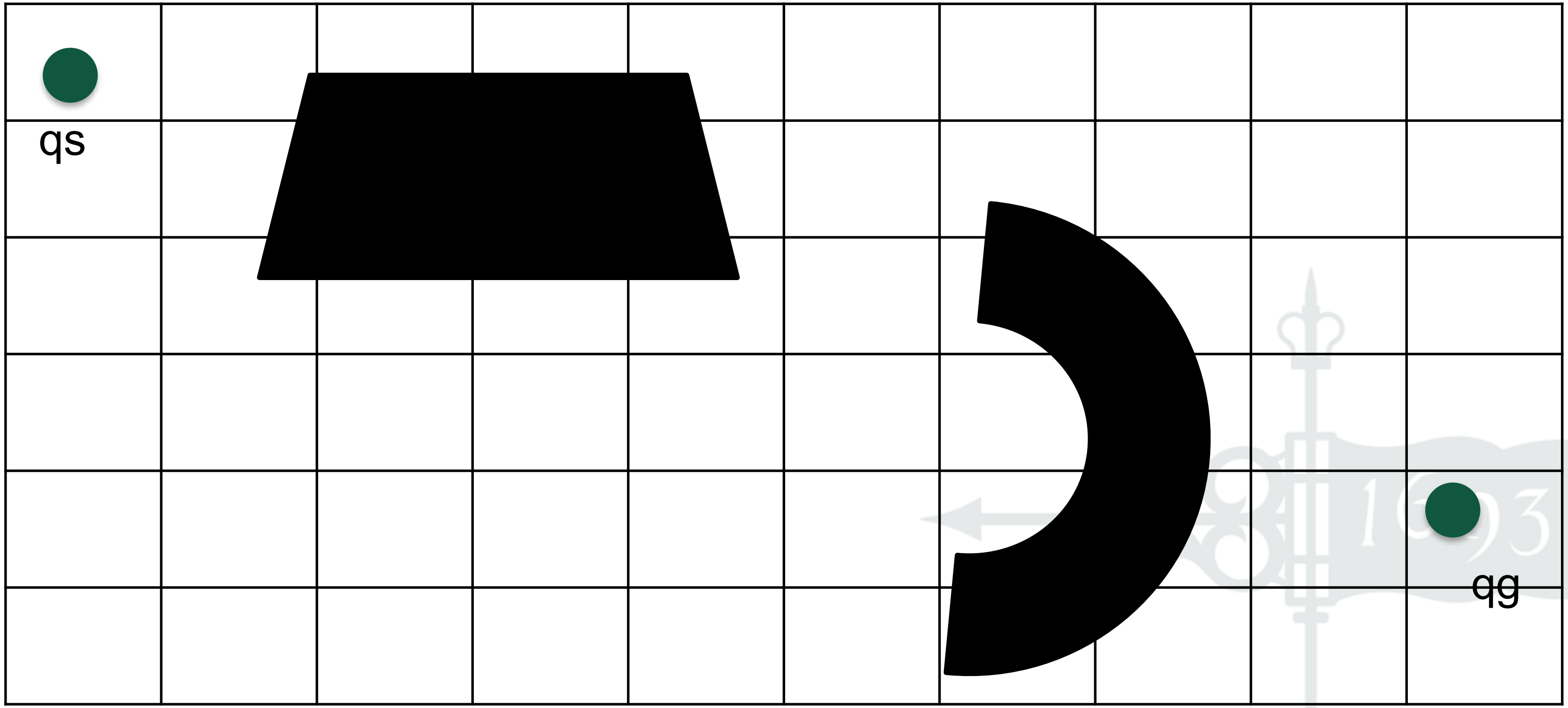
Model Planning Families

- Reactive
  - Bug family
  - Dynamic Window
- Model-based
  - Visibility
  - Grid

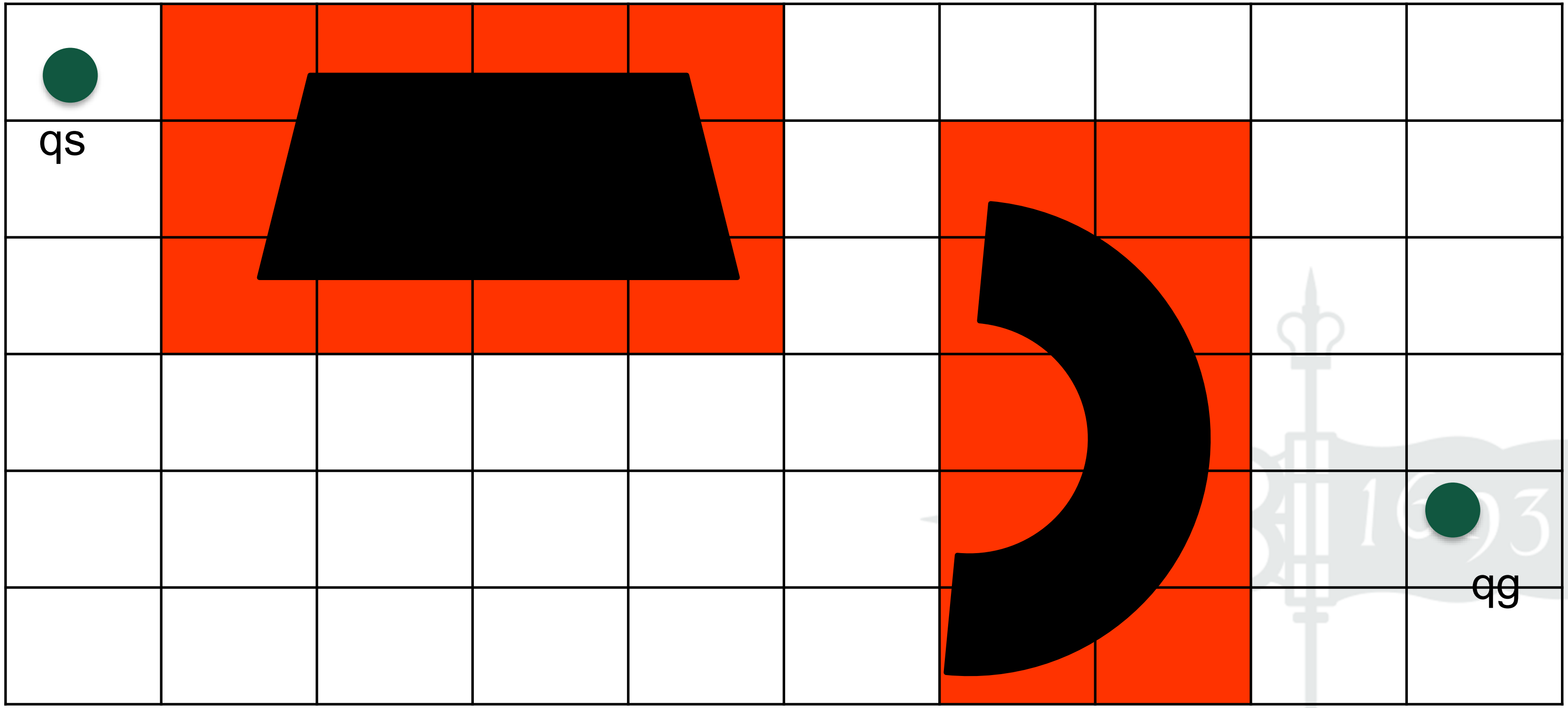
Path Planning: Grid



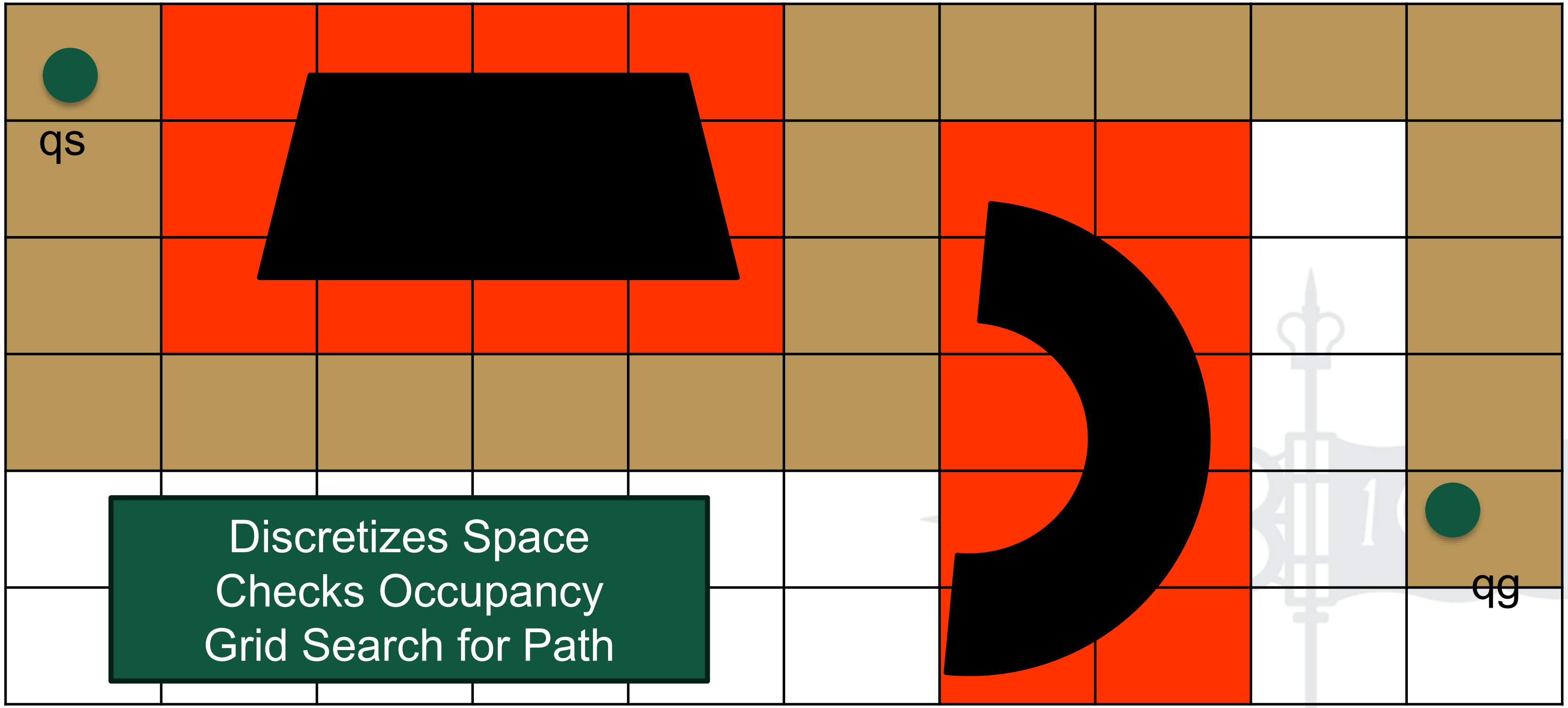
Path Planning: Grid



Path Planning: Grid



Path Planning: Grid

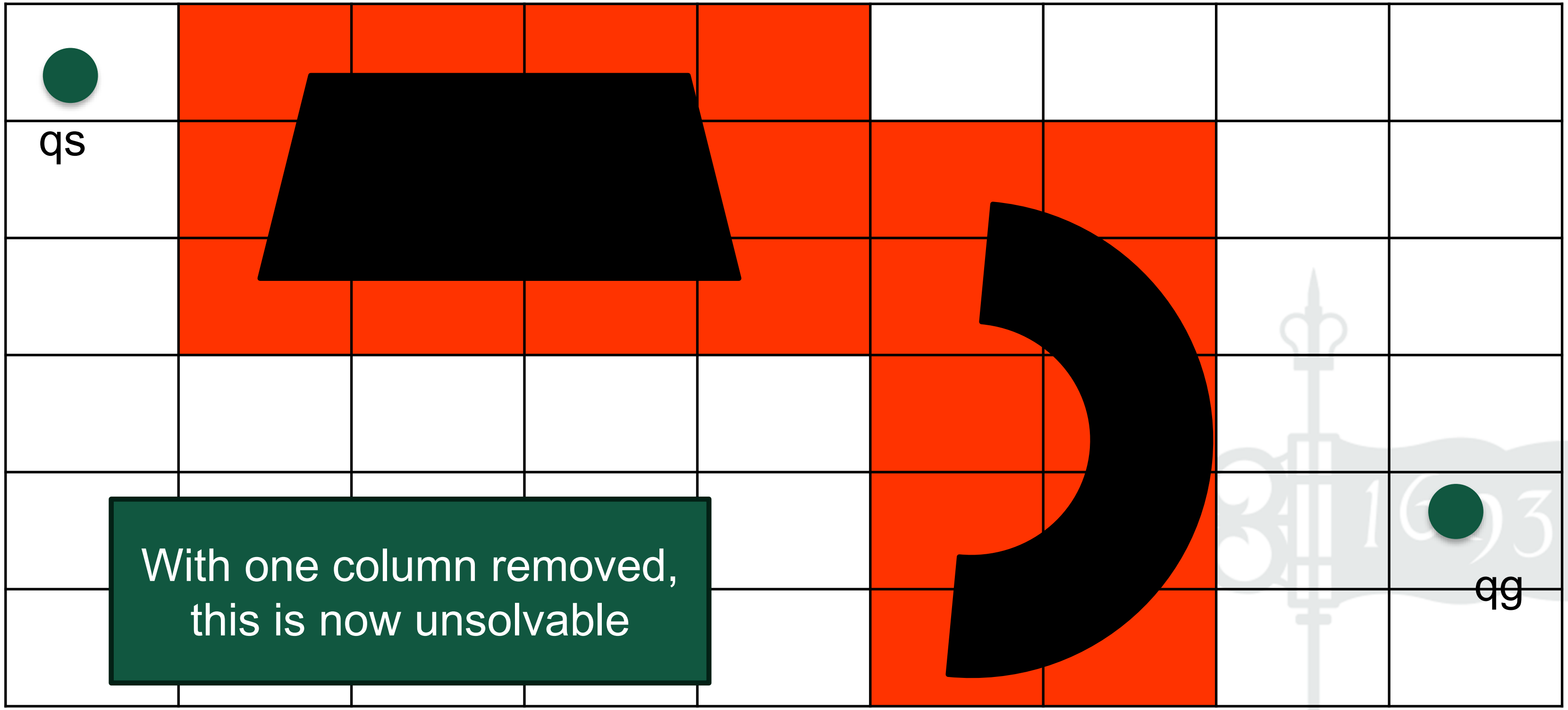


Discretizes Space
Checks Occupancy
Grid Search for Path

Path Planning: Grid

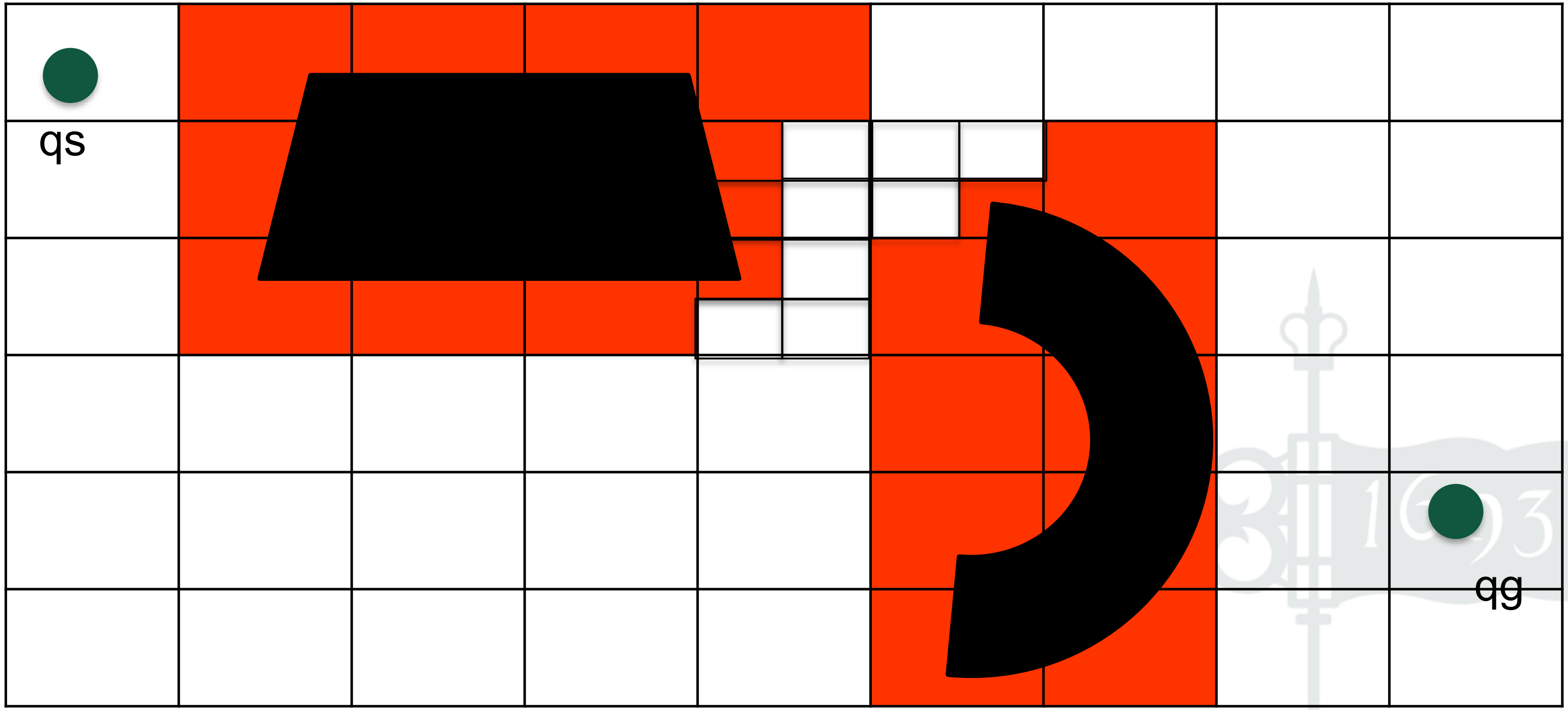
- What can go wrong?
  - Depends on:
    - Shapes encountered
    - Resolution of the grid

Path Planning: Grid

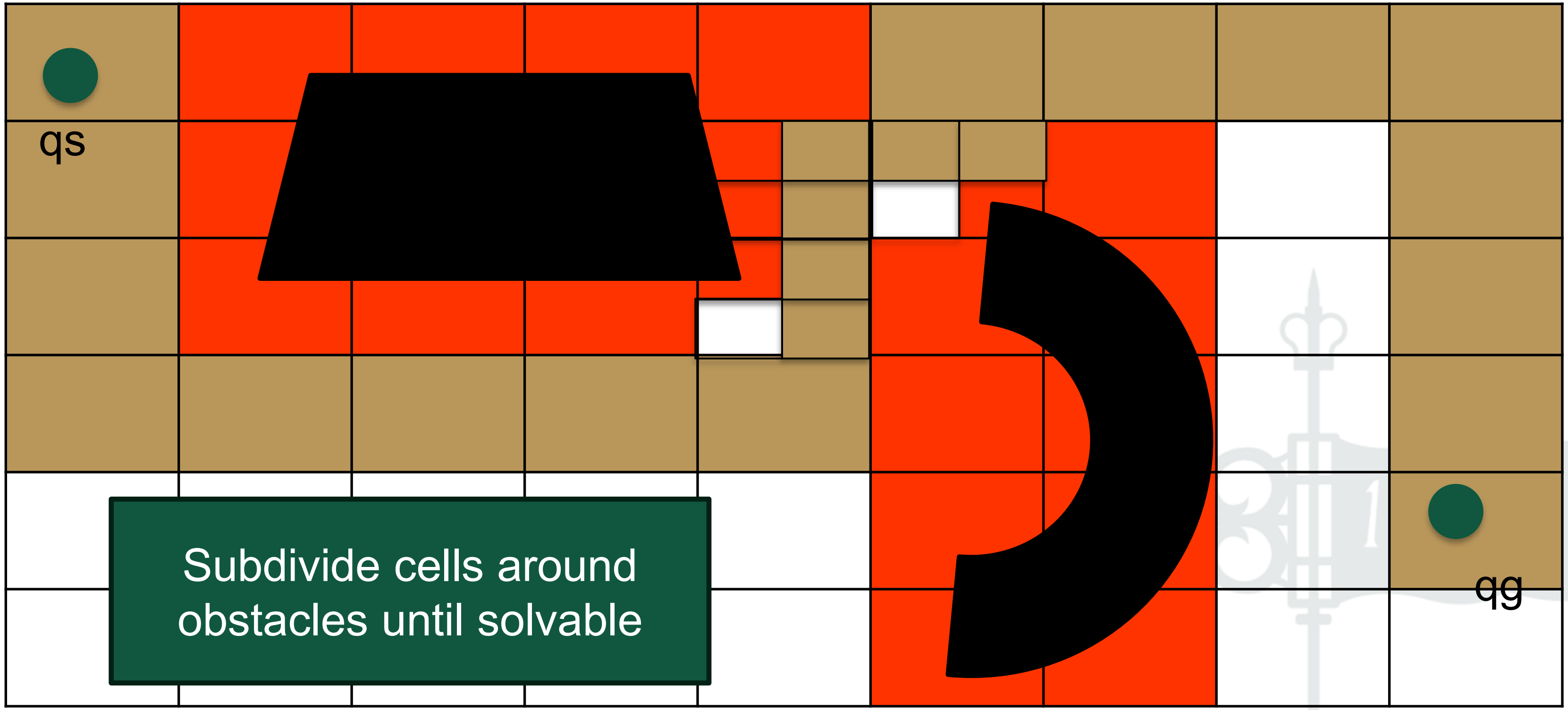


With one column removed, this is now unsolvable

Grid With Refinement



Grid With Refinement

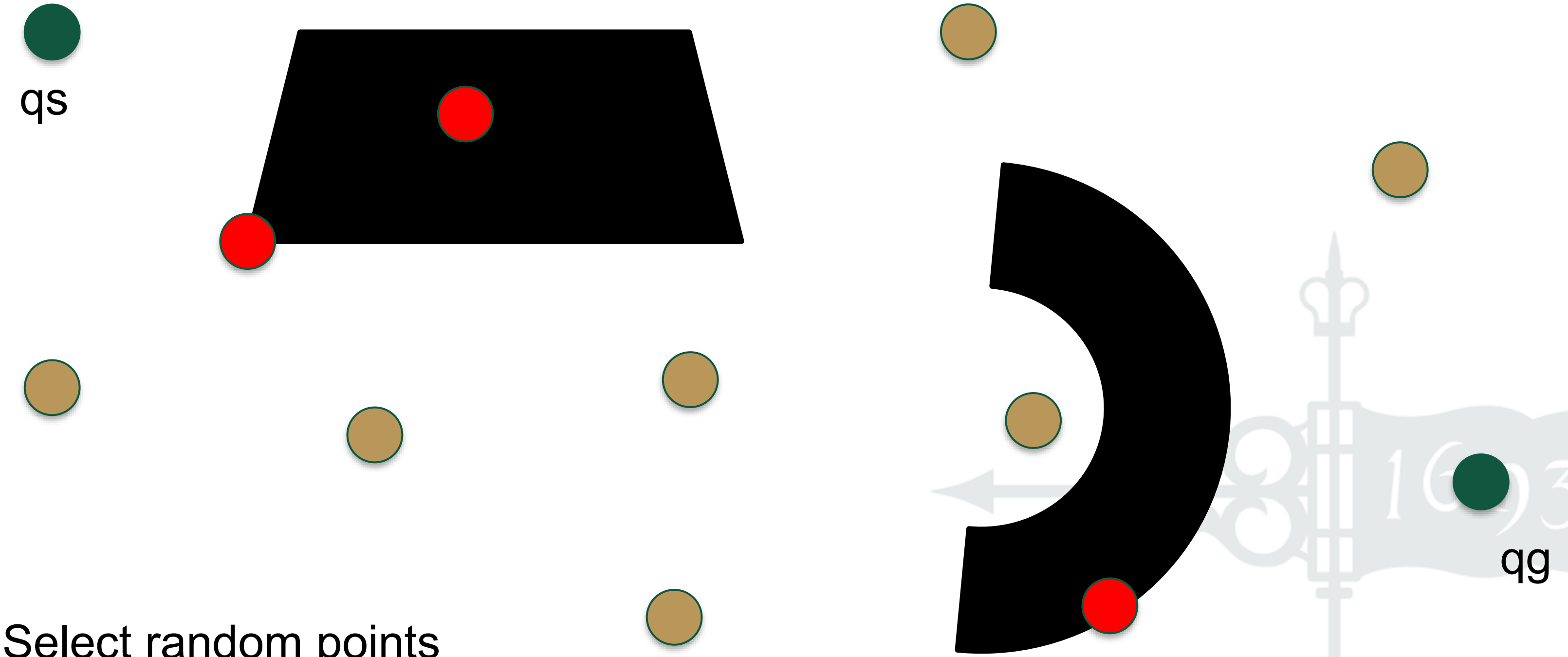


Subdivide cells around obstacles until solvable

Model Planning Families

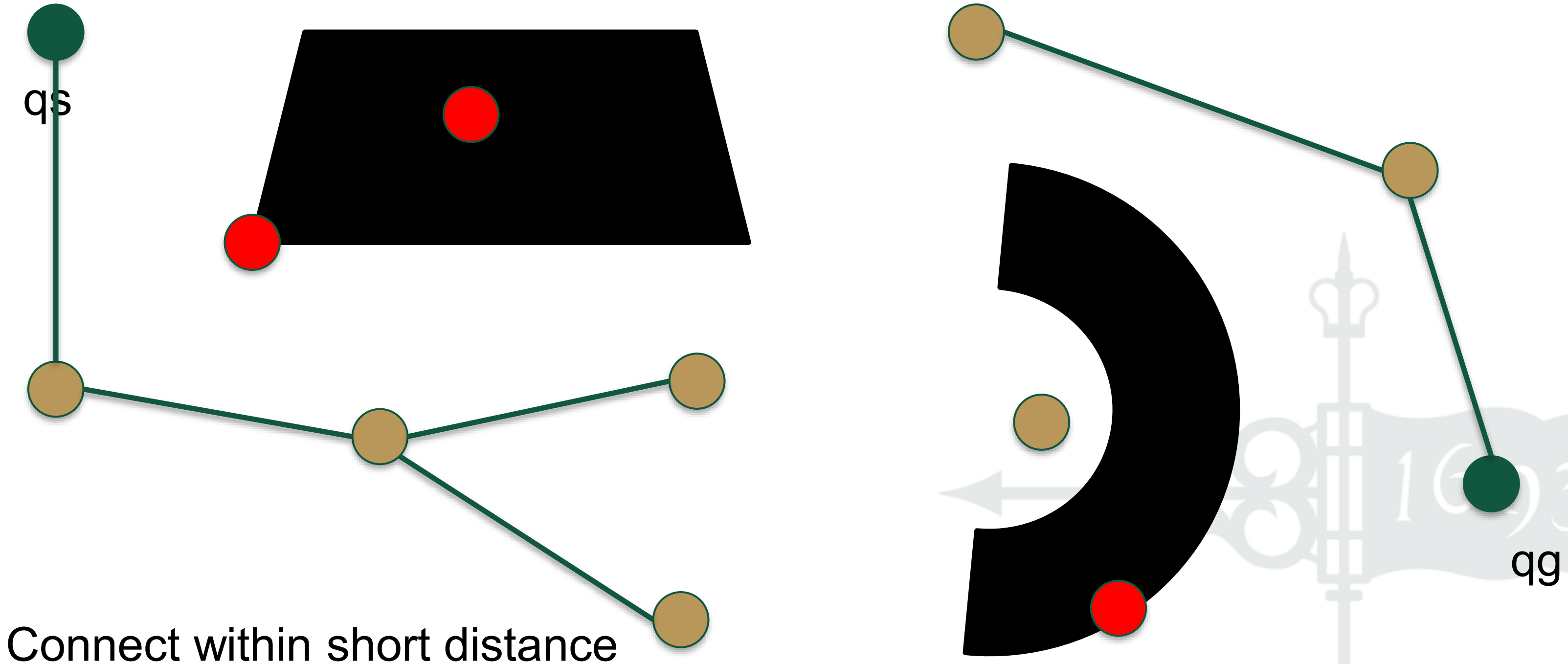
- Reactive
  - Bug family
  - Dynamic Window
- Model-based
  - Visibility
  - Grid
  - Probabilistic

Path Planning: Probabilistic



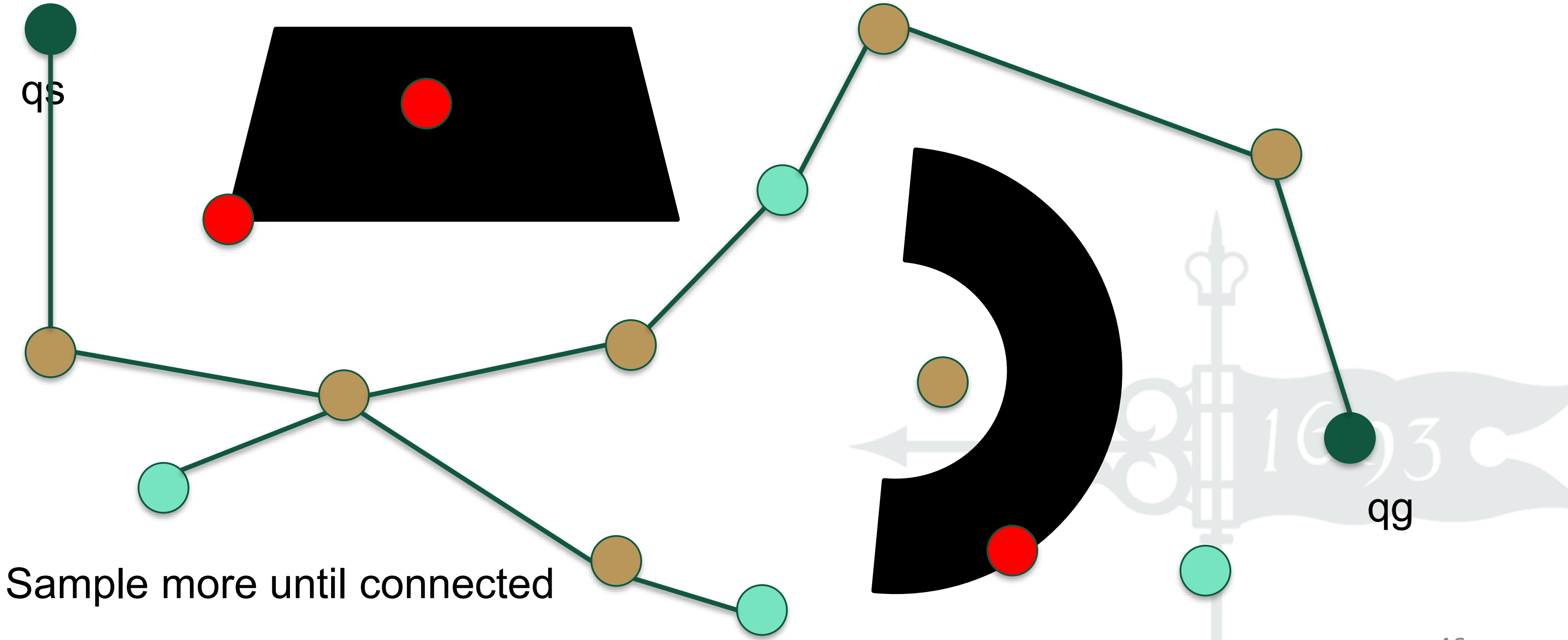
Select random points

Path Planning: Probabilistic



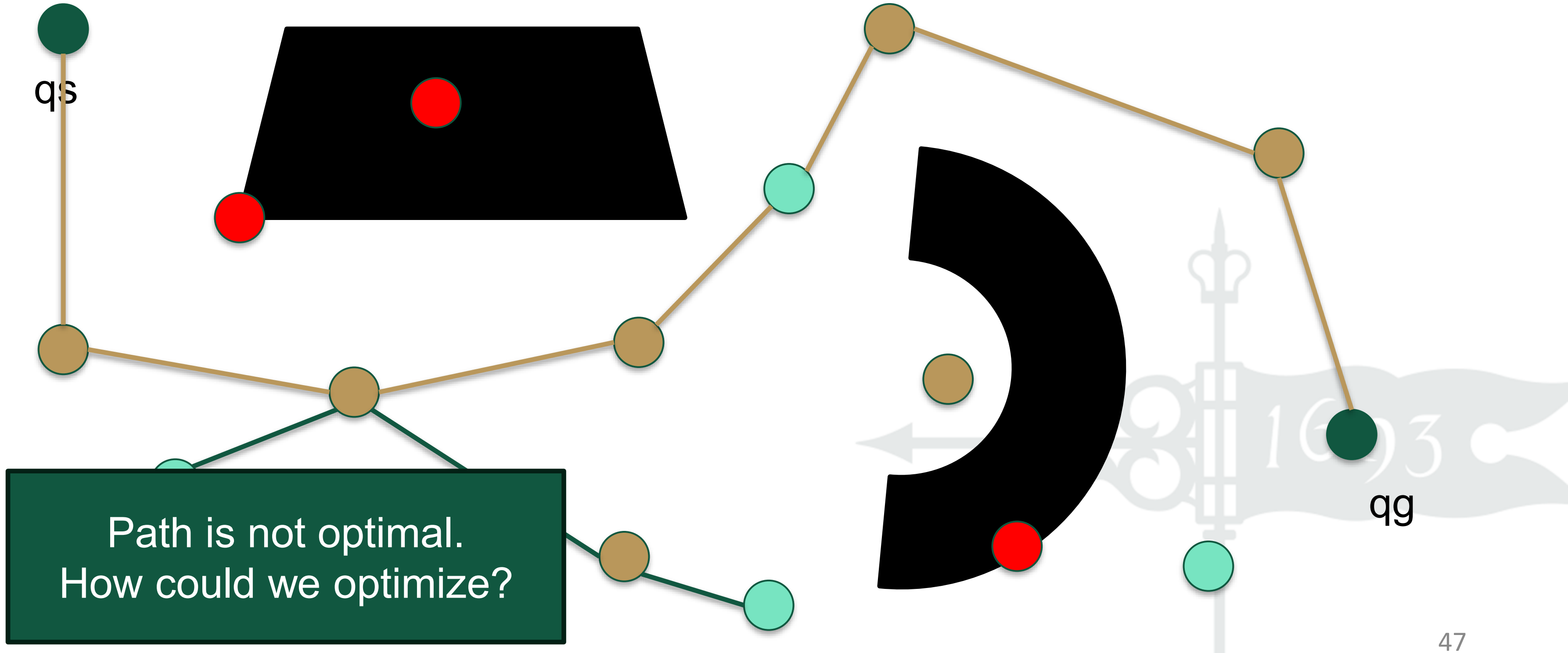
Connect within short distance

Path Planning: Probabilistic



Sample more until connected

Path Planning: Probabilistic



Path is not optimal.
How could we optimize?

Path Planning

- Reactive
  - – Local knowledge/feedback
  - – Fast, but can get stuck if not careful
- Model
  - – Big picture
  - – Can be efficient, but takes time