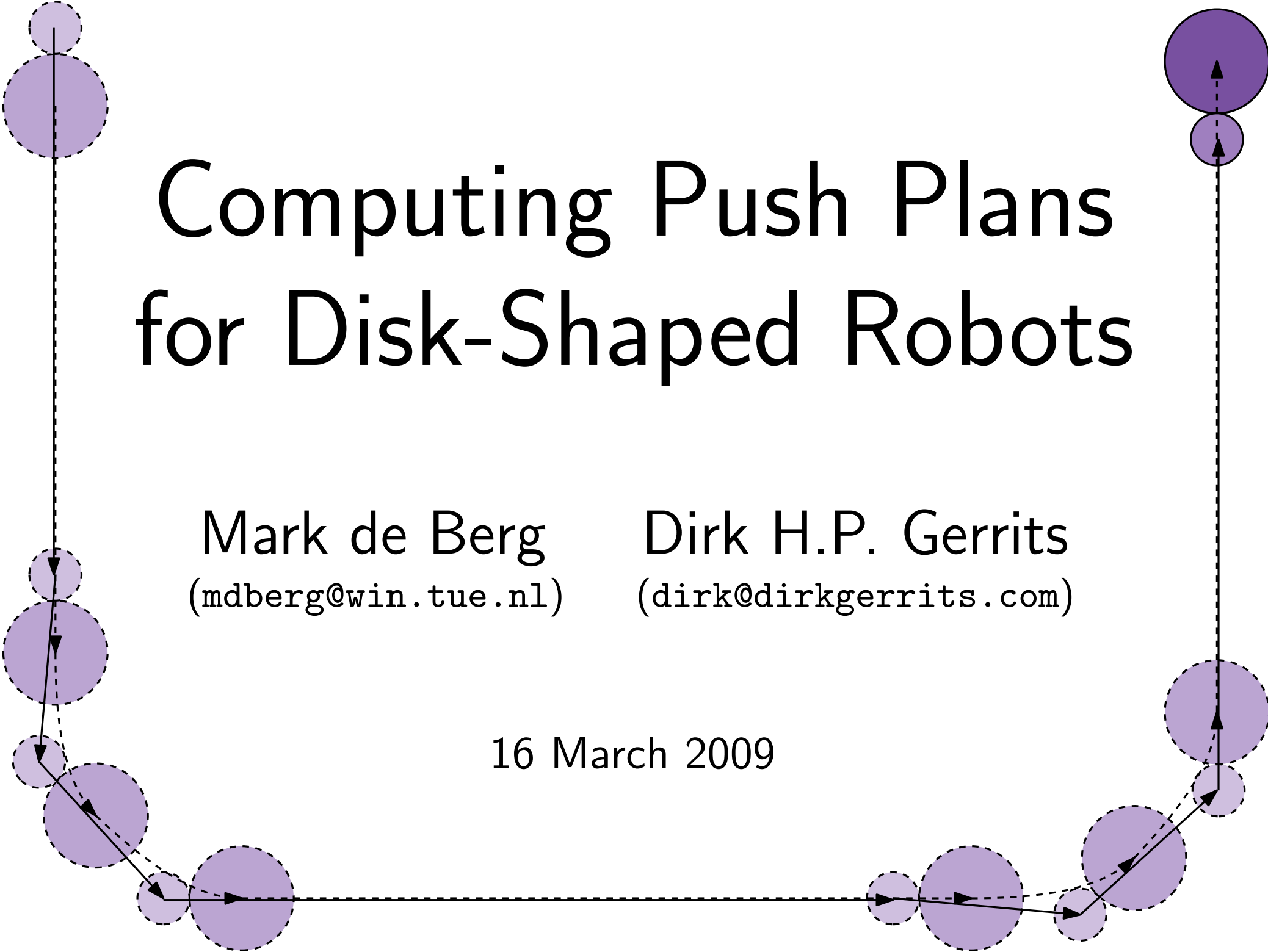


# Computing Push Plans for Disk-Shaped Robots

Mark de Berg  
([mdberg@win.tue.nl](mailto:mdberg@win.tue.nl))

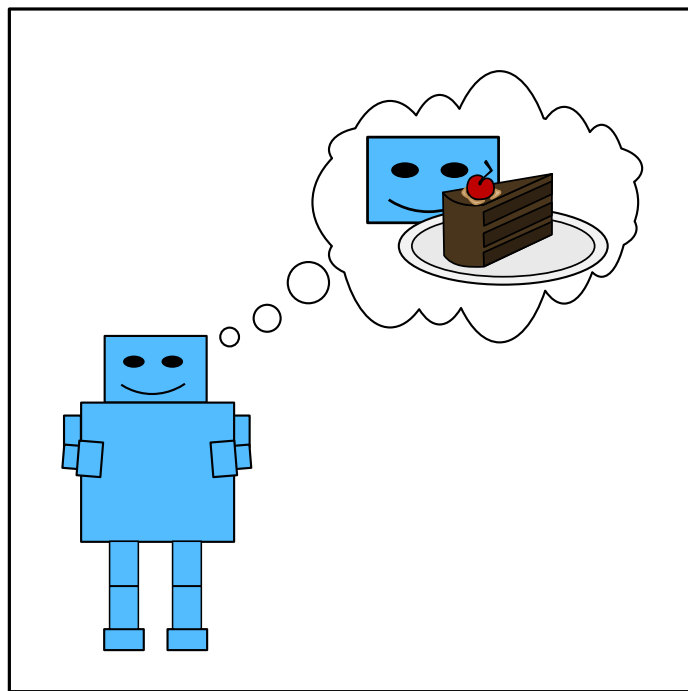
Dirk H.P. Gerrits  
([dirk@dirkgerrits.com](mailto:dirk@dirkgerrits.com))

16 March 2009

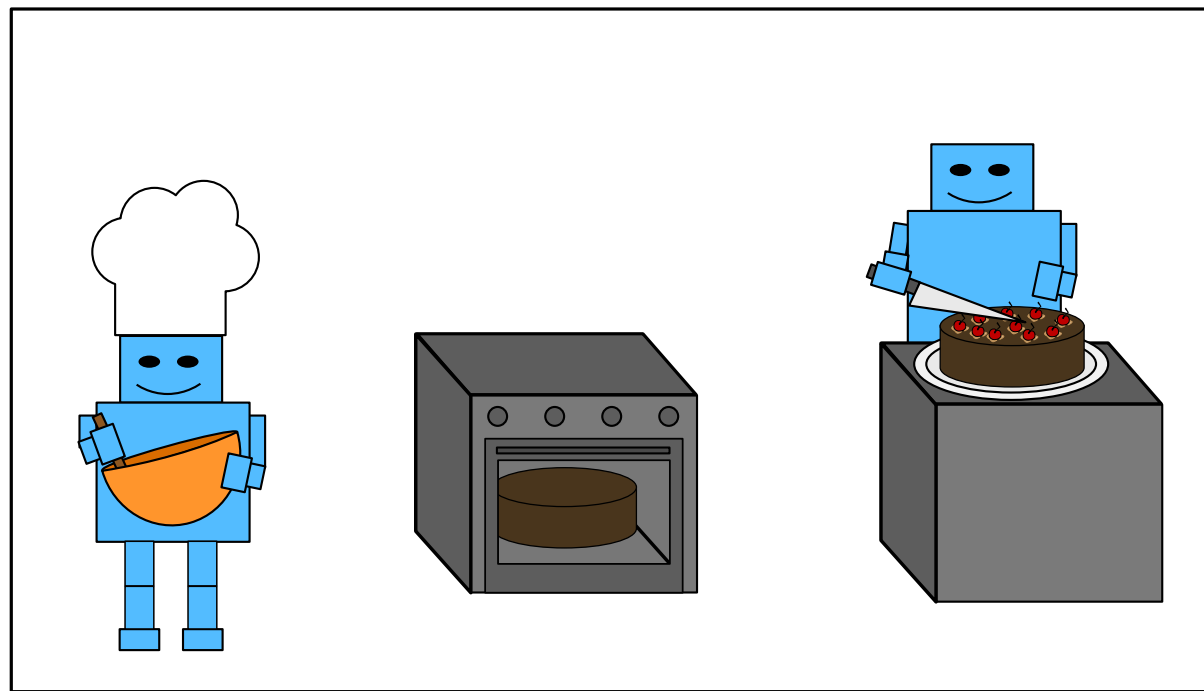


# Problem context

An important problem in robotics is **planning**.



current situation,  
desired situation

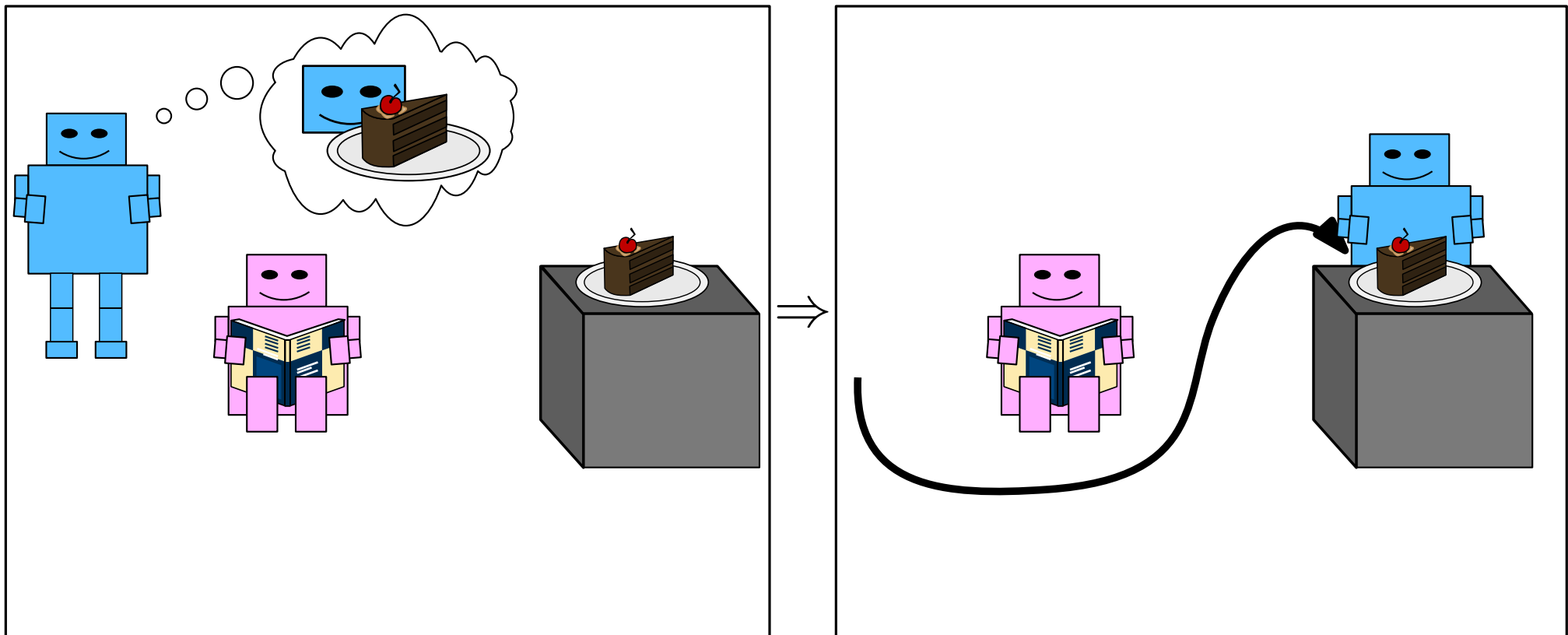


sequence of steps to reach the one  
from the other

# Problem context

An important problem in robotics is **planning**.

In **path planning** the robot wants to move to a desired location.

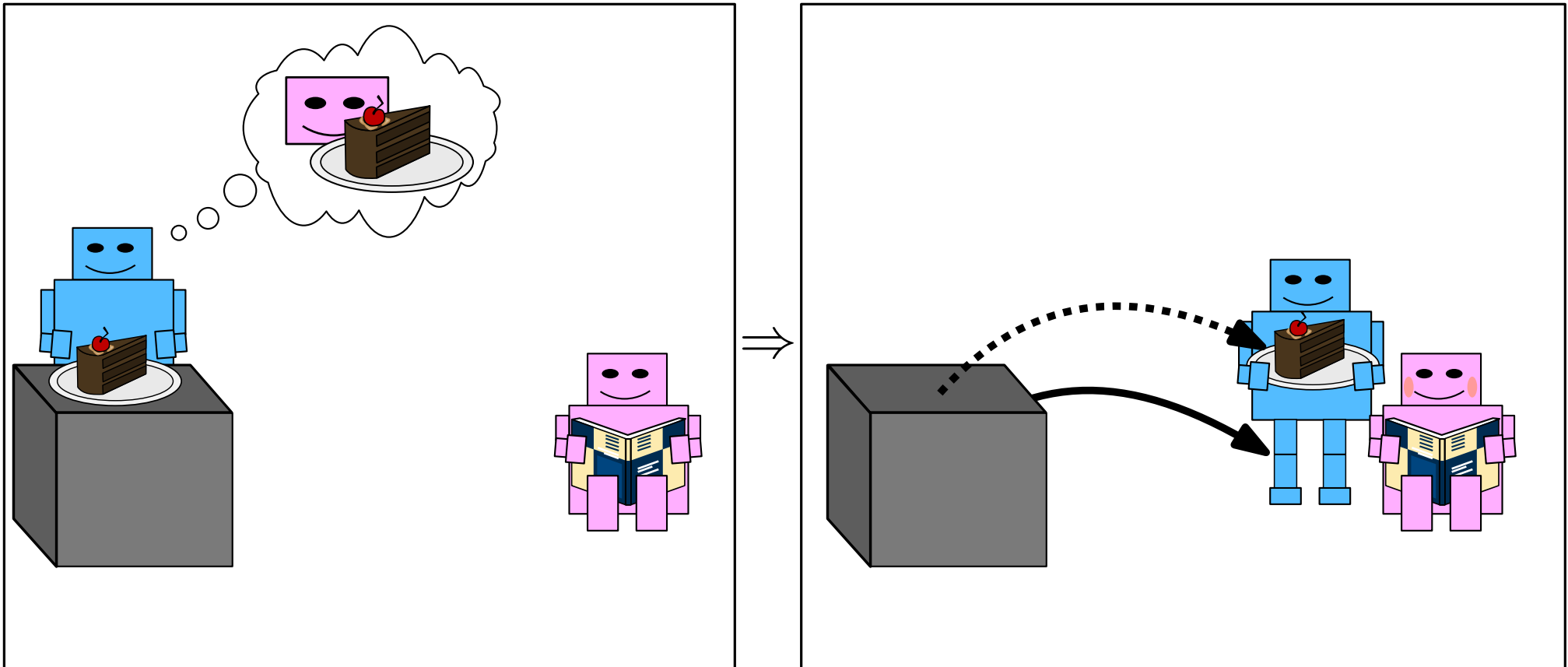


# Problem context

An important problem in robotics is **planning**.

In **path planning** the robot wants to move to a desired location.

In **manipulation path planning** the robot wants to move a passive object instead.



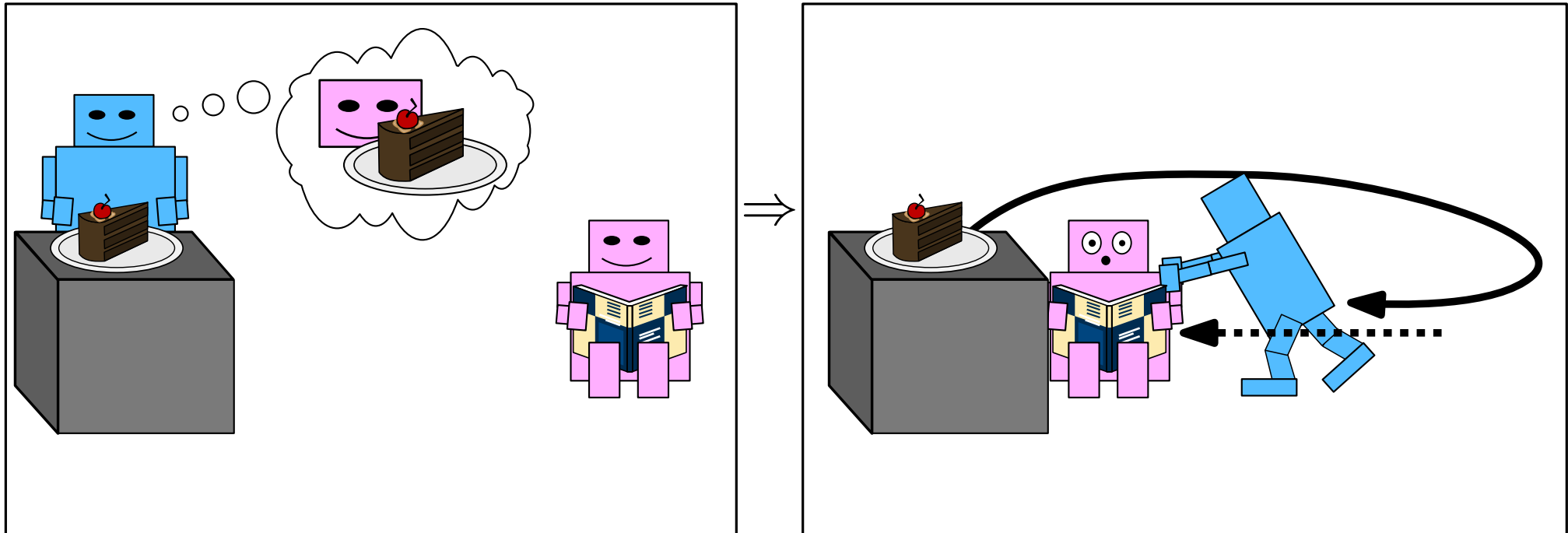
# Problem context

An important problem in robotics is **planning**.

In **path planning** the robot wants to move to a desired location.

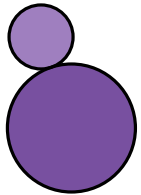
In **manipulation path planning** the robot wants to move a passive object instead.

We've studied a manipulation path planning problem involving **pushing** (especially useful for objects too heavy to lift).



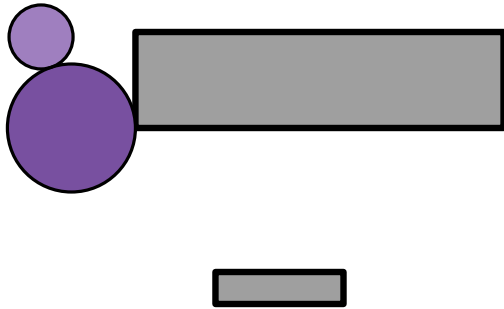
# Problem description

**Given:** disk-shaped **object** and (smaller) **pusher** robot,



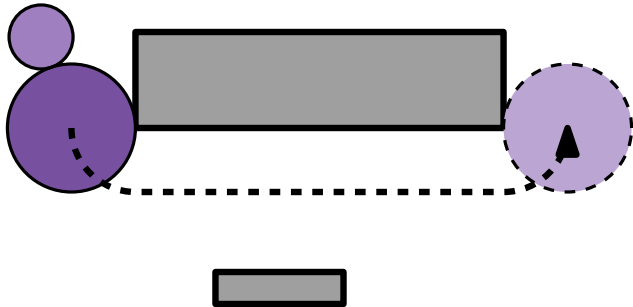
# Problem description

**Given:** disk-shaped **object** and (smaller) **pusher** robot,  
polygonal **obstacles**,



# Problem description

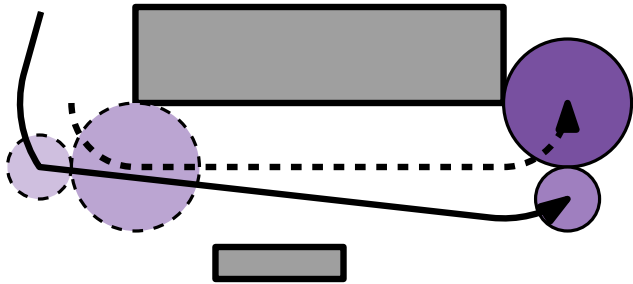
**Given:** disk-shaped **object** and (smaller) **pusher** robot,  
polygonal **obstacles**, collision-free **object path**



# Problem description

**Given:** disk-shaped **object** and (smaller) **pusher** robot,  
polygonal **obstacles**, collision-free **object path**

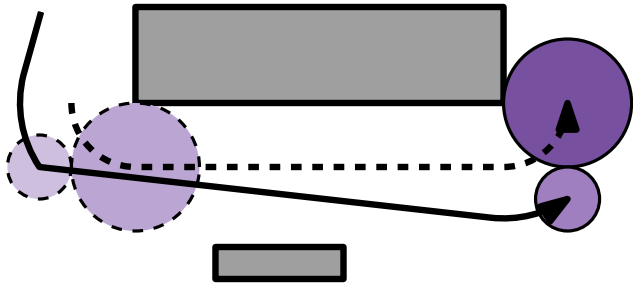
**Find:** a **push plan** (a path for the pusher that makes it push the object along its given path)



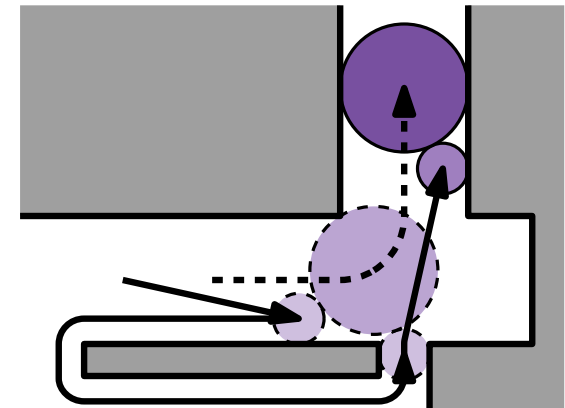
# Problem description

**Given:** disk-shaped **object** and (smaller) **pusher** robot,  
polygonal **obstacles**, collision-free **object path**

**Find:** a **push plan** (a path for the pusher that makes it push the object along its given path)



contact-preserving push plan

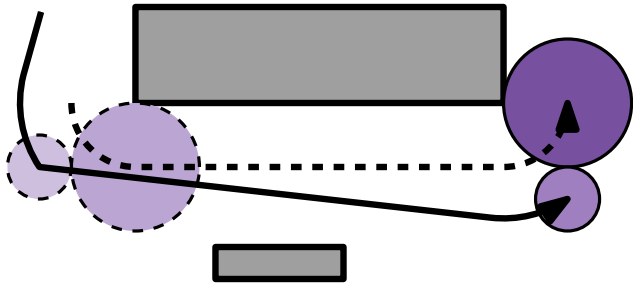


unrestricted push plan

# Problem description

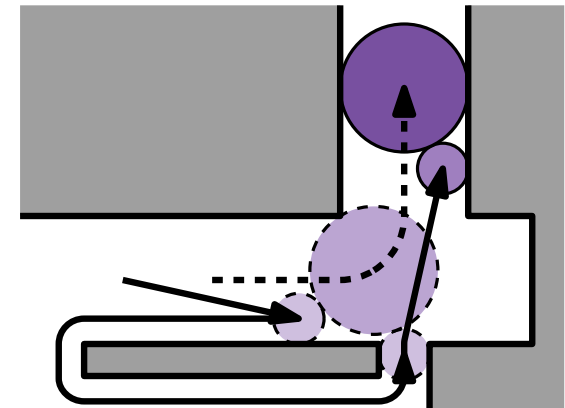
**Given:** disk-shaped **object** and (smaller) **pusher** robot, polygonal **obstacles**, collision-free **object path**

**Find:** a **push plan** (a path for the pusher that makes it push the object along its given path)



contact-preserving push plan

- faster, more general algorithm than Nieuwenhuisen et al. (2005)
- first shortest-path algorithm

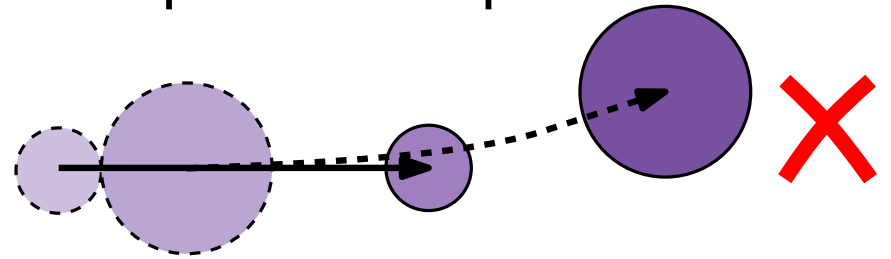
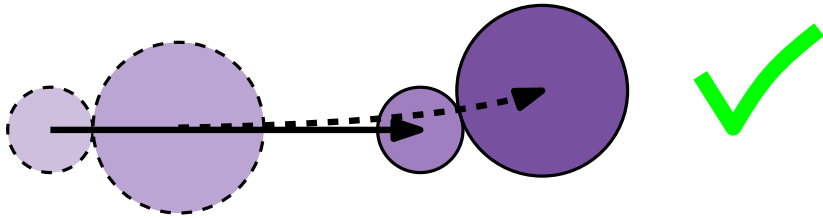


unrestricted push plan

- shown to be necessary
- first algorithm

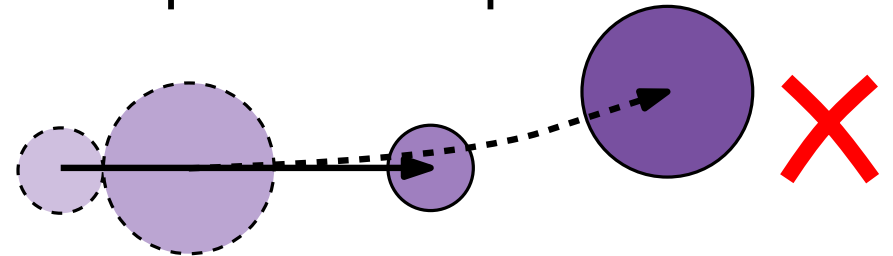
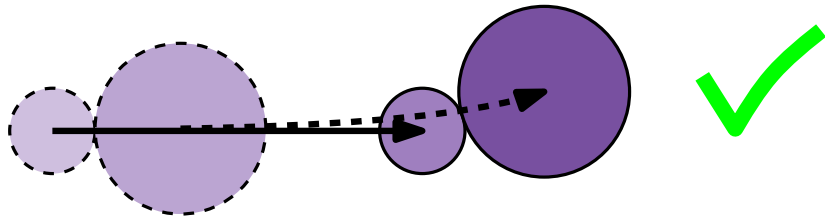
# Assumptions

- The object stops instantly when the pusher stops.

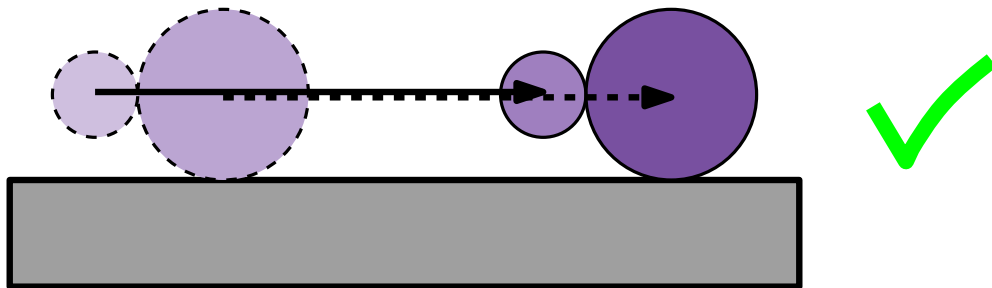


# Assumptions

- The object stops instantly when the pusher stops.

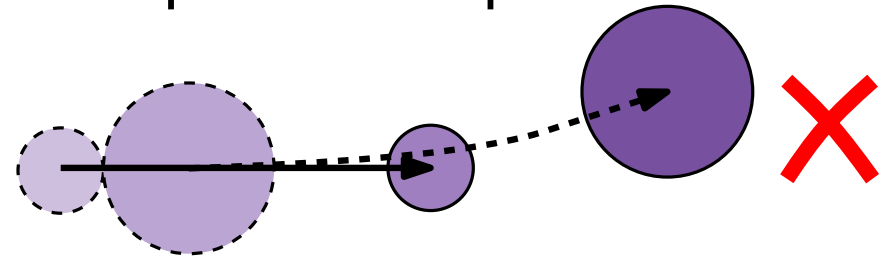
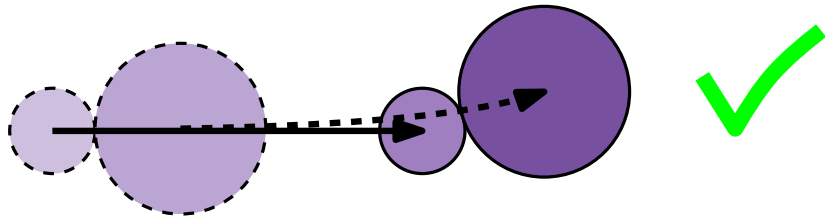


- The disks are allowed to slide along obstacle boundaries

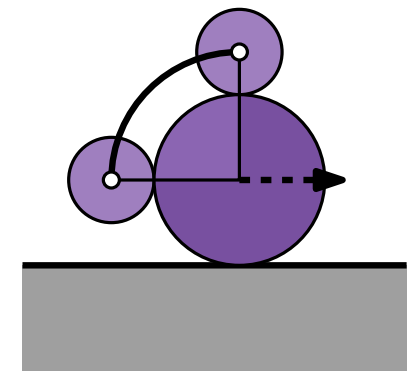
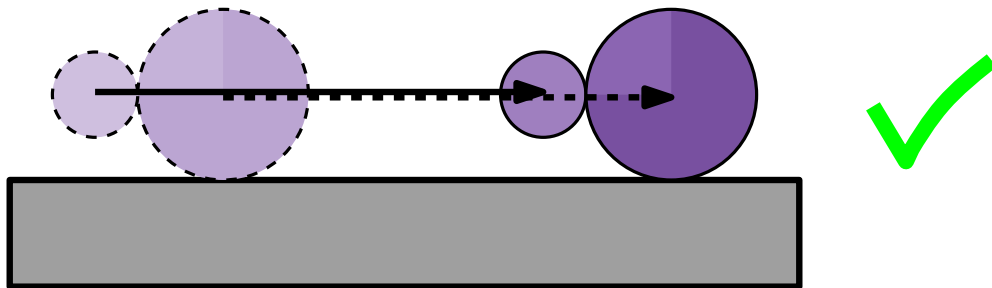


# Assumptions

- The object stops instantly when the pusher stops.

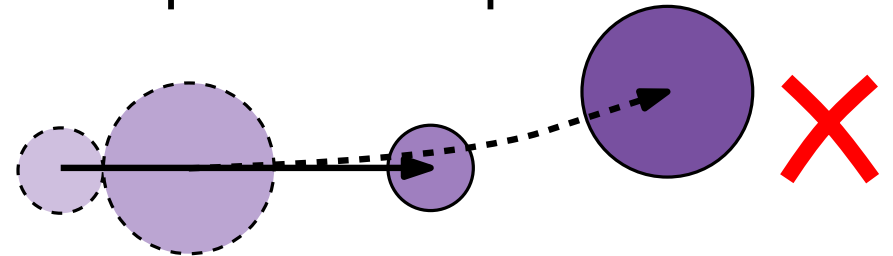
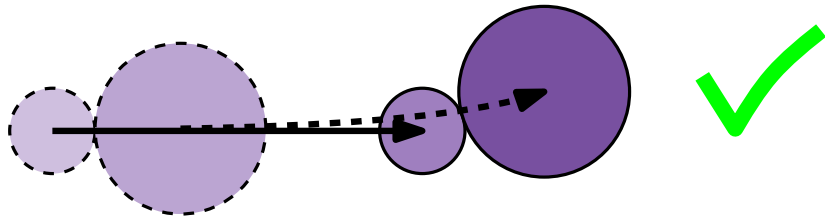


- The disks are allowed to slide along obstacle boundaries so that the pusher can push from a wider range of positions (the **push range**).

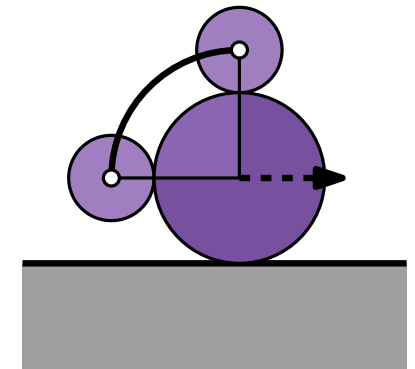
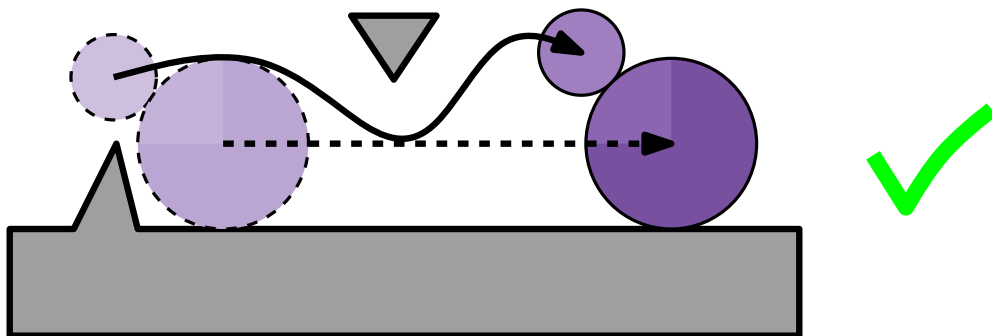
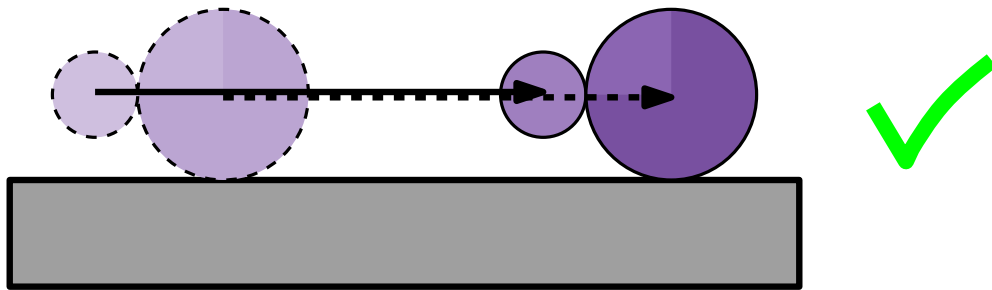


# Assumptions

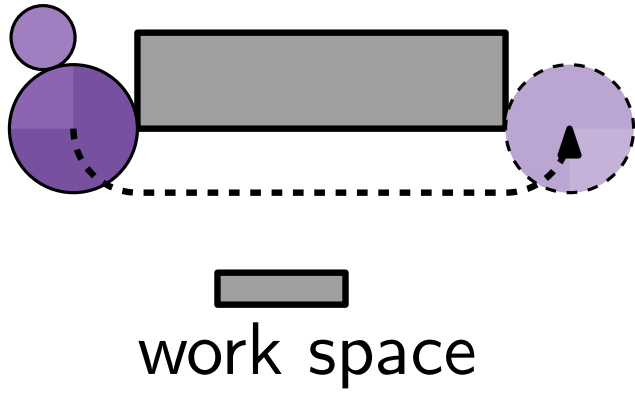
- The object stops instantly when the pusher stops.



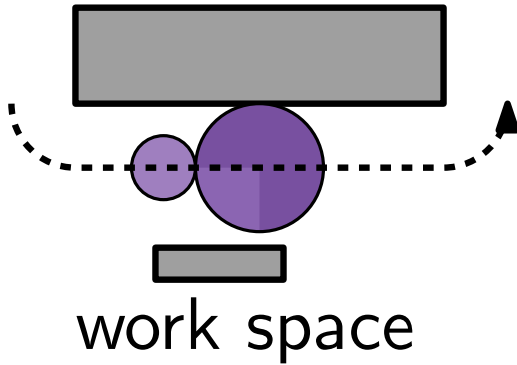
- The disks are allowed to slide along obstacle boundaries so that the pusher can push from a wider range of positions (the **push range**).



# The configuration space

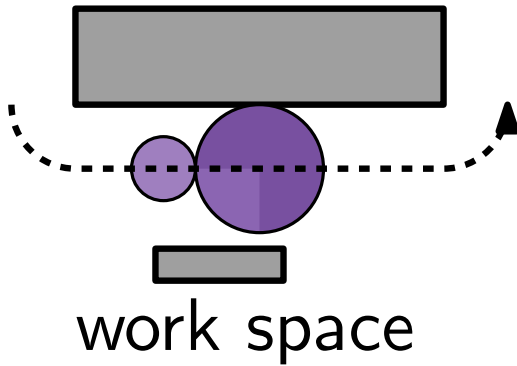


# The configuration space

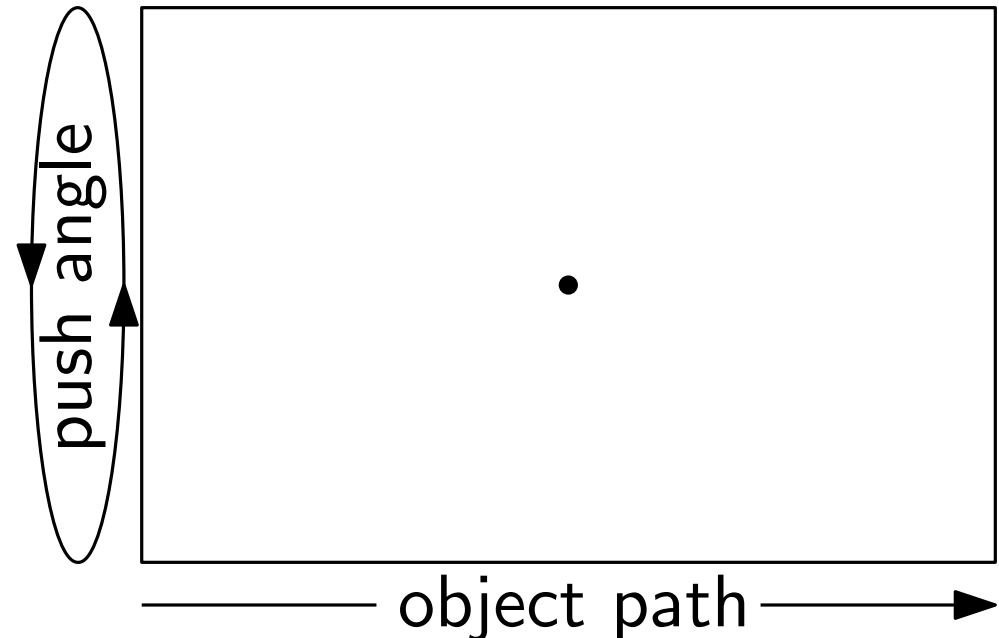


In our given work space, a **configuration** is a placement of the object and pusher.

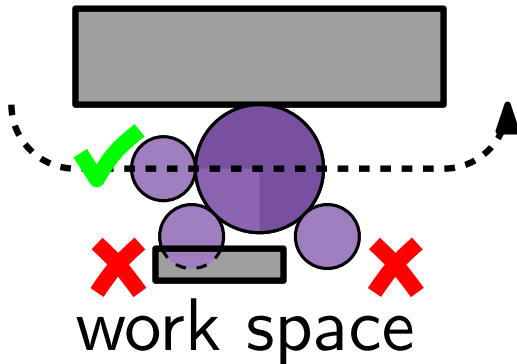
# The configuration space



In our given work space, a **configuration** is a placement of the object and pusher. If the pusher always touches the object, which in turn always stays on its path, then a configuration is a point in a 2-dimensional **configuration space**.

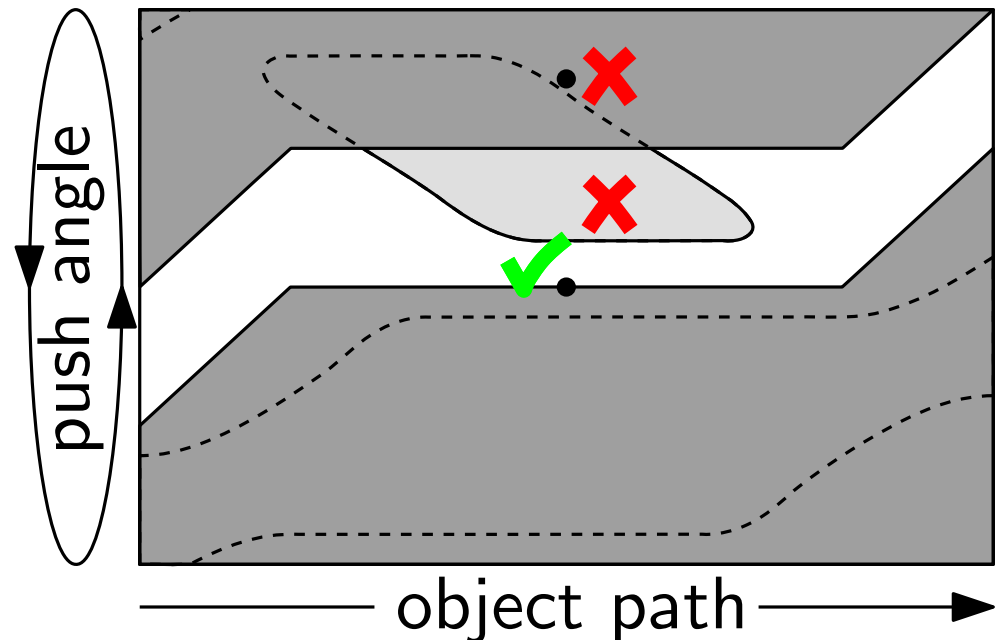


# The configuration space

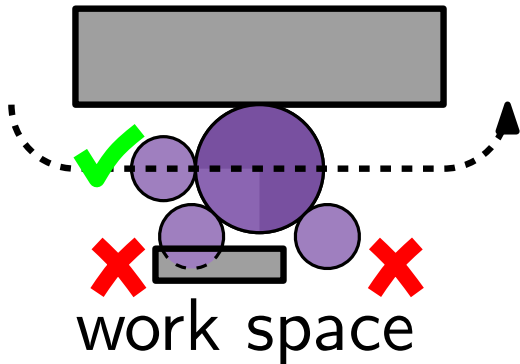


In our given work space, a **configuration** is a placement of the object and pusher. If the pusher always touches the object, which in turn always stays on its path, then a configuration is a point in a 2-dimensional **configuration space**.

Configurations where obstacles are intersected, or the push range is not respected, form the **forbidden space**.



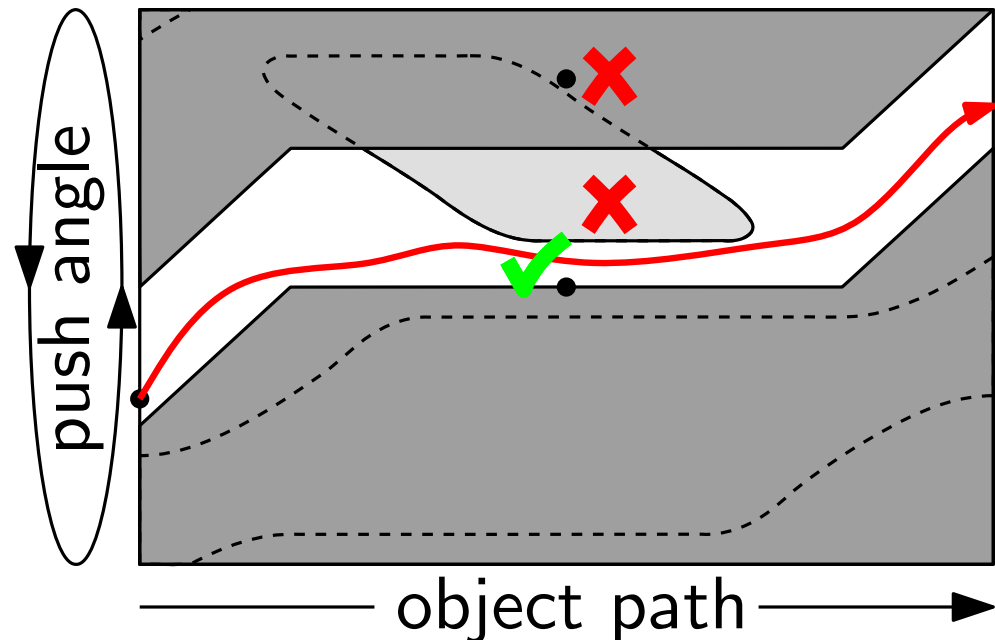
# The configuration space



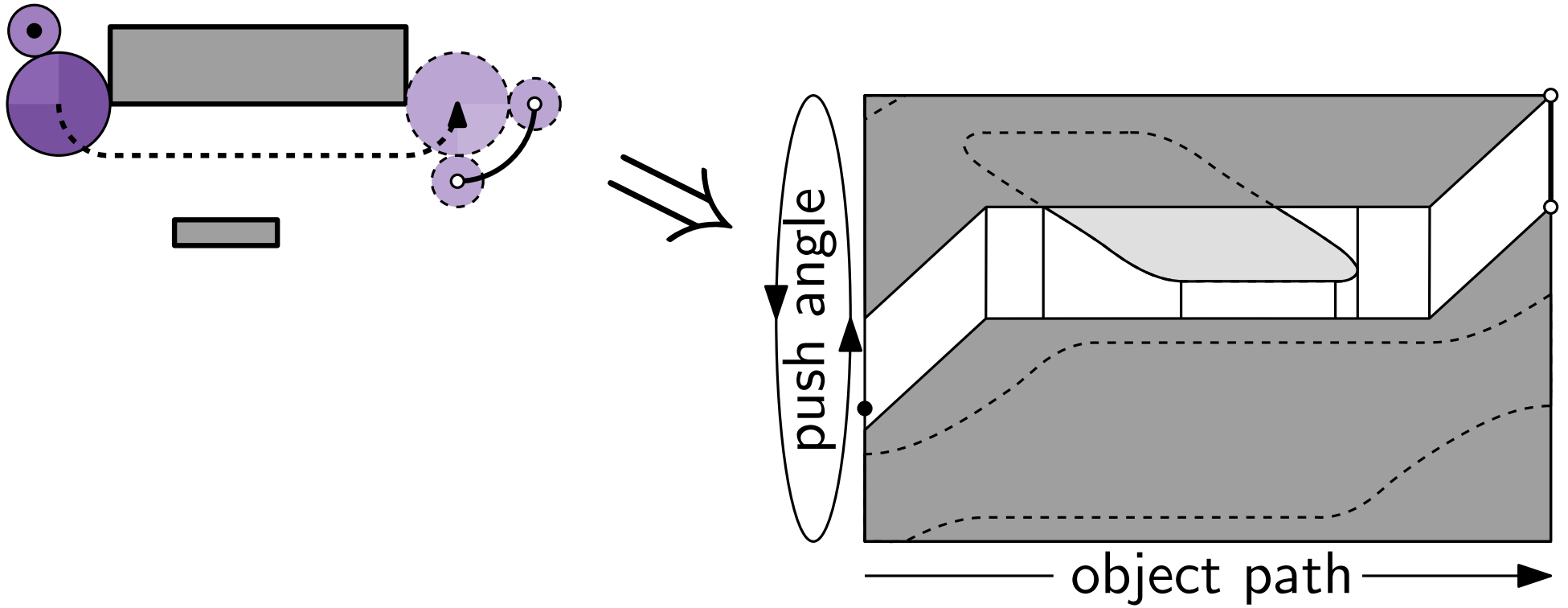
In our given work space, a **configuration** is a placement of the object and pusher. If the pusher always touches the object, which in turn always stays on its path, then a configuration is a point in a 2-dimensional **configuration space**.

Configurations where obstacles are intersected, or the push range is not respected, form the **forbidden space**.

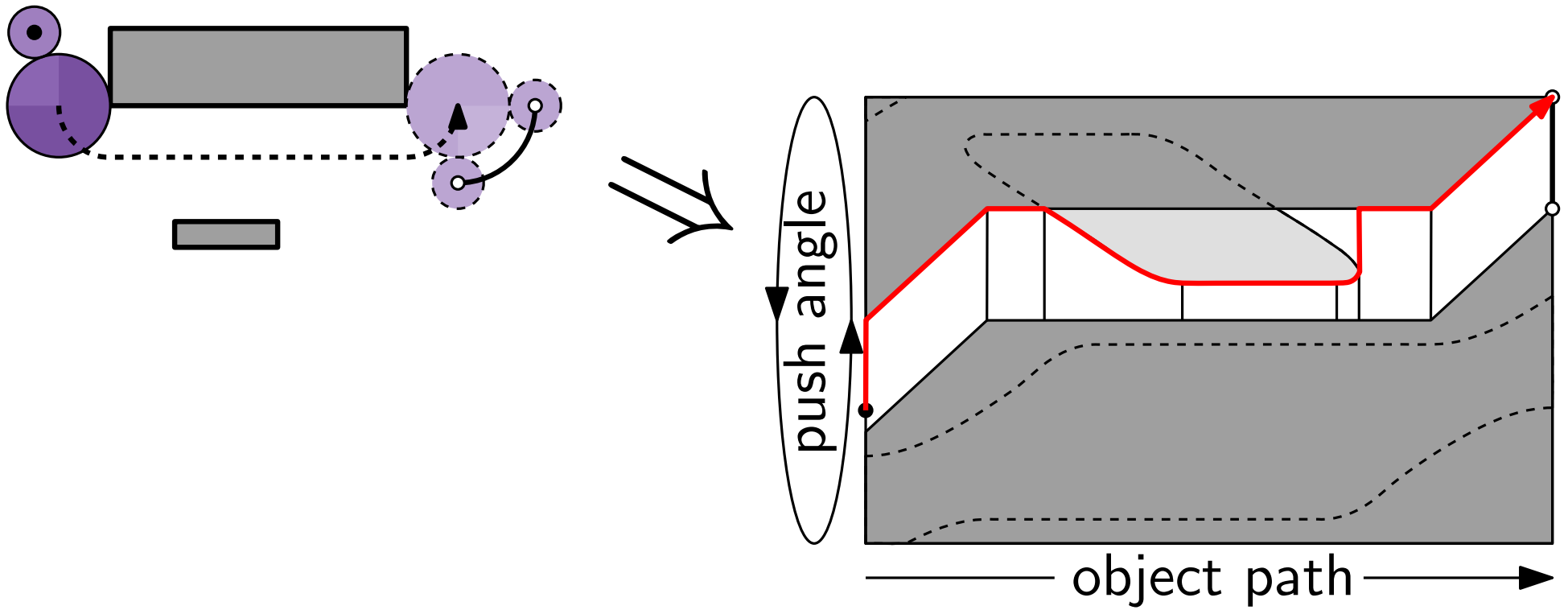
A path through the remaining **free space** yields a push plan.



# A contact-preserving push plan

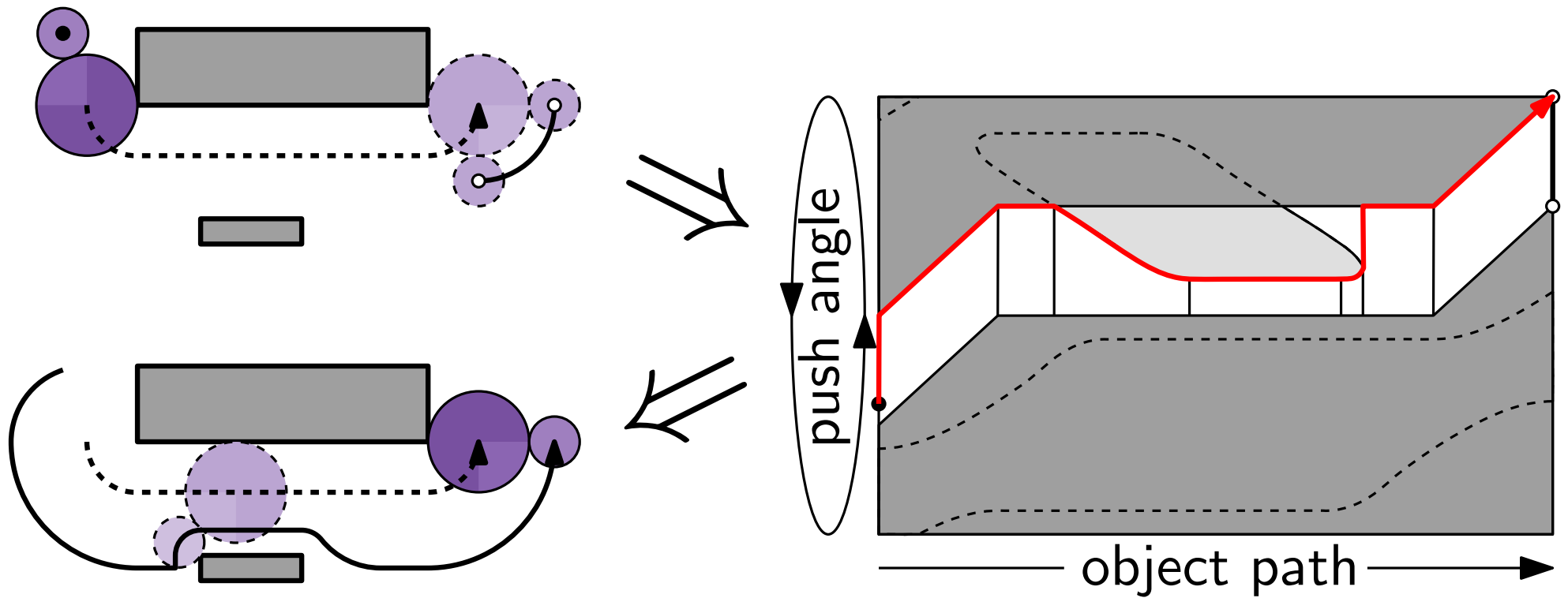


# A contact-preserving push plan



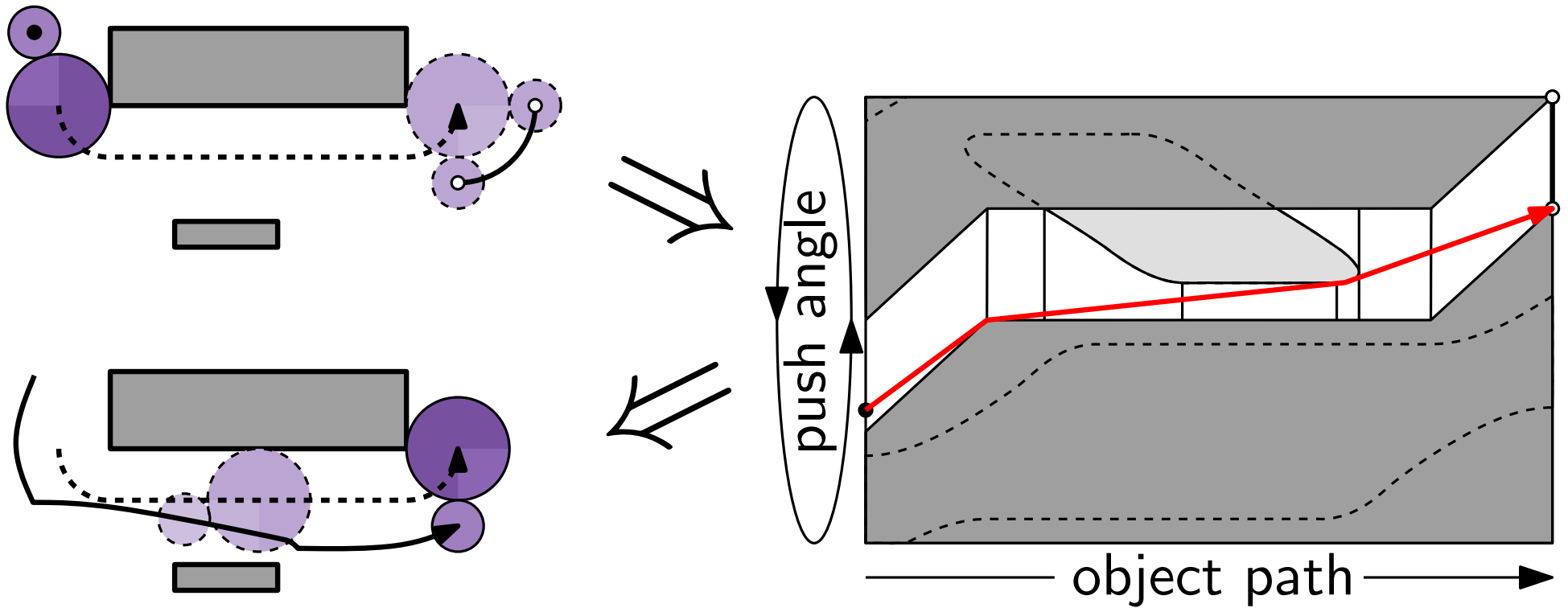
It is easy to find a push plan in the configuration space by using a vertical decomposition and following cell boundaries

# A contact-preserving push plan



It is easy to find a push plan in the configuration space by using a vertical decomposition and following cell boundaries, but the result may not be very good.

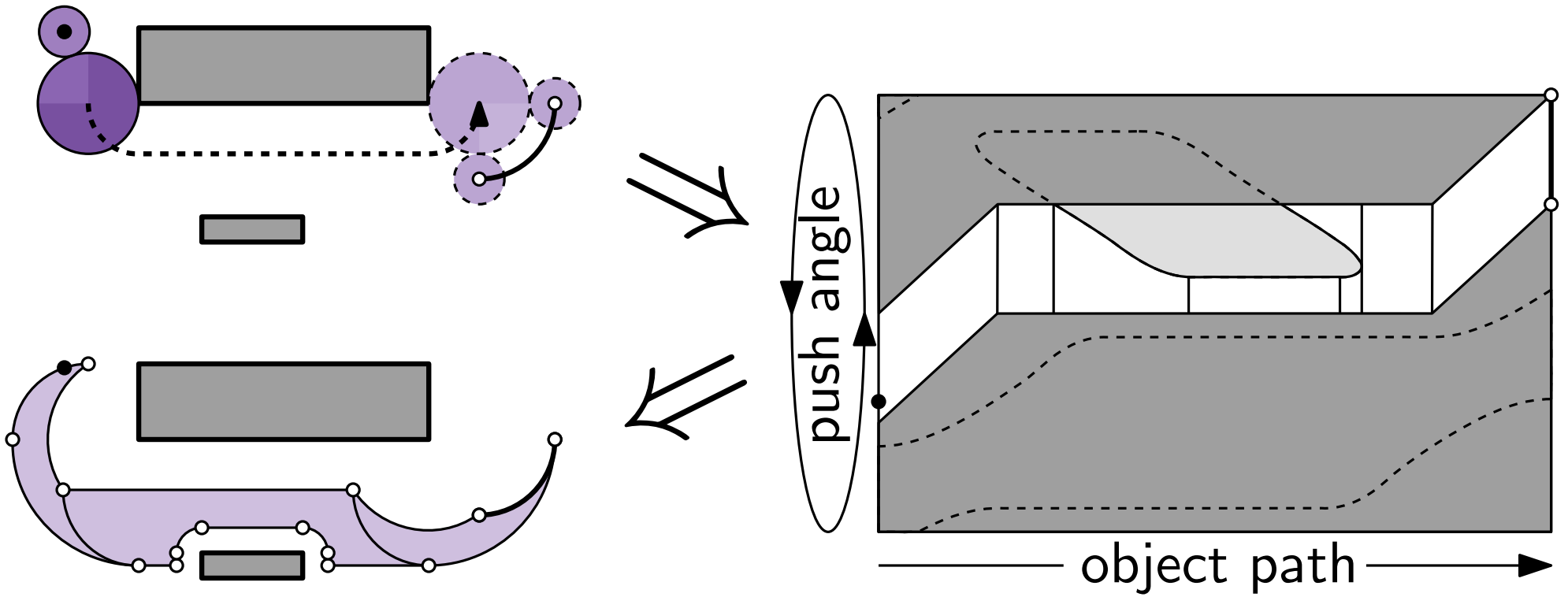
# A contact-preserving push plan



It is easy to find a push plan in the configuration space by using a vertical decomposition and following cell boundaries, but the result may not be very good.

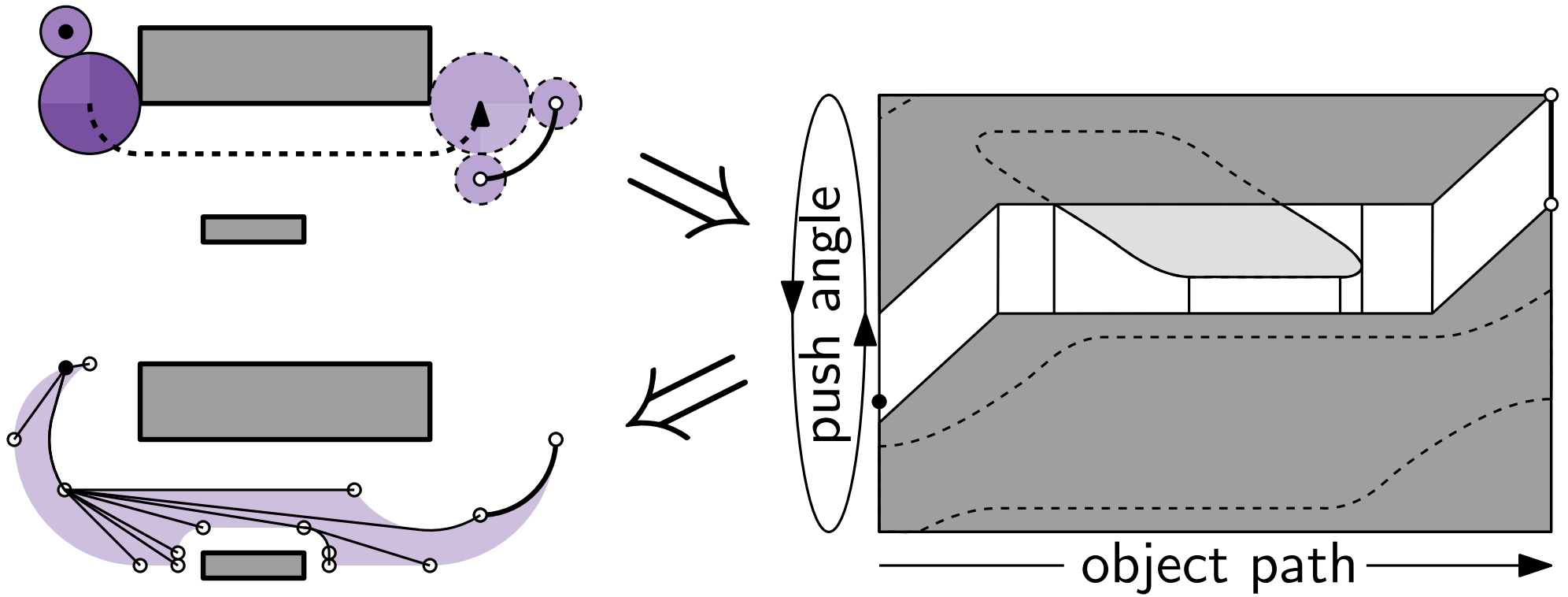
A Euclidean shortest path is better, but still not optimal.

# A shortest contact-preserving push plan



Translate the free space into the region through which the pusher may move to push the object.

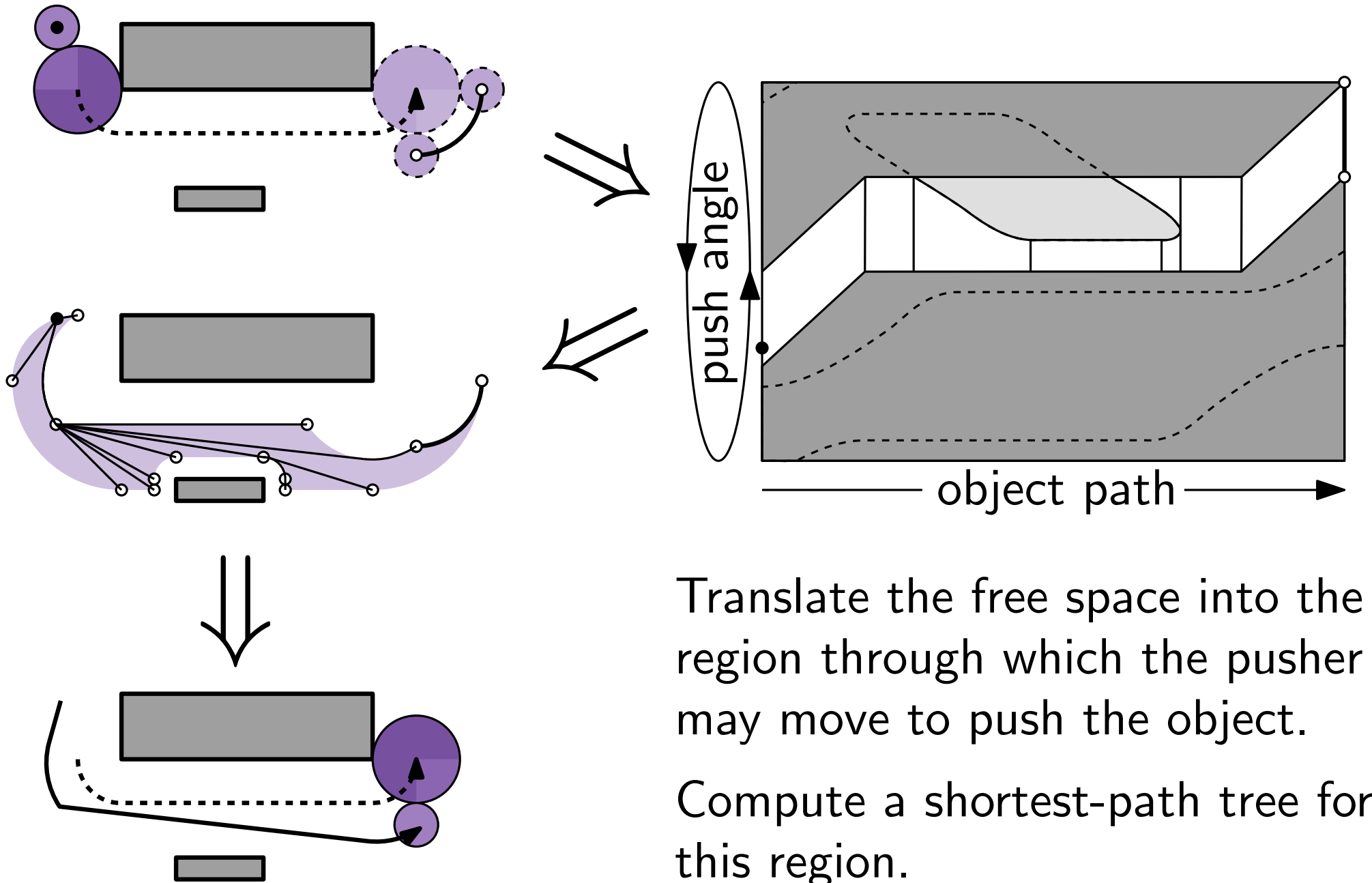
# A shortest contact-preserving push plan



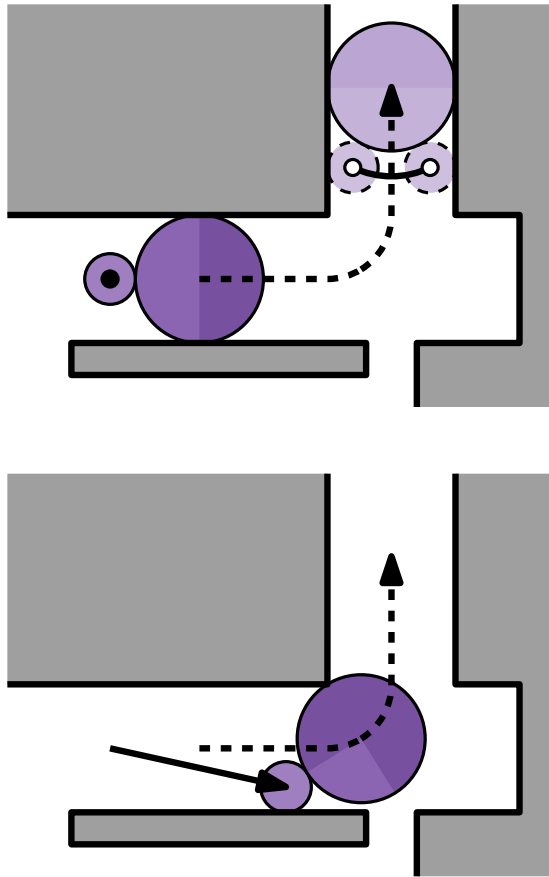
Translate the free space into the region through which the pusher may move to push the object.

Compute a shortest-path tree for this region.

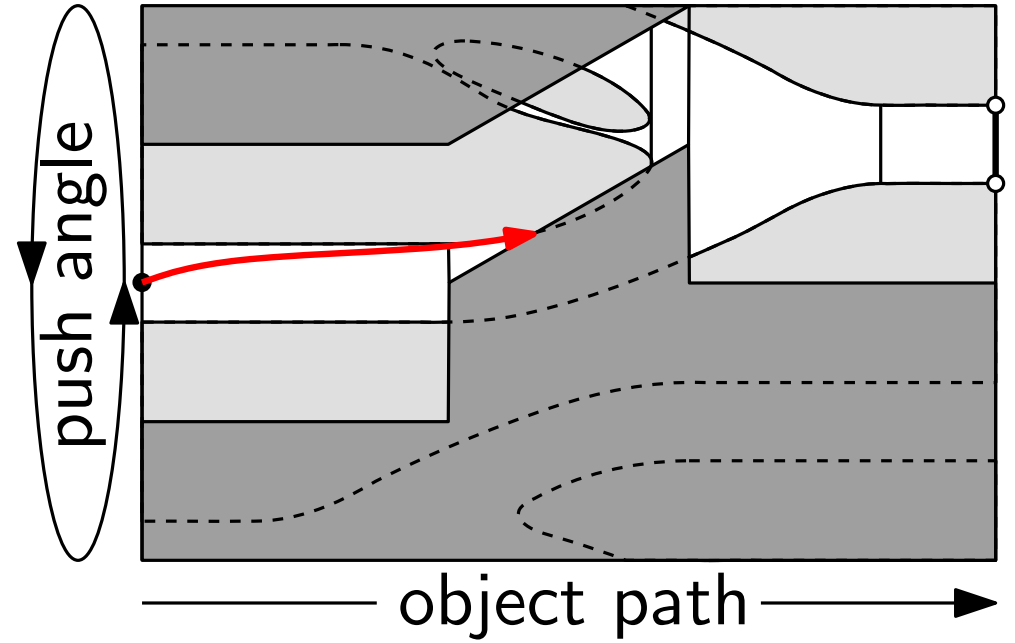
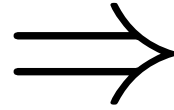
# A shortest contact-preserving push plan



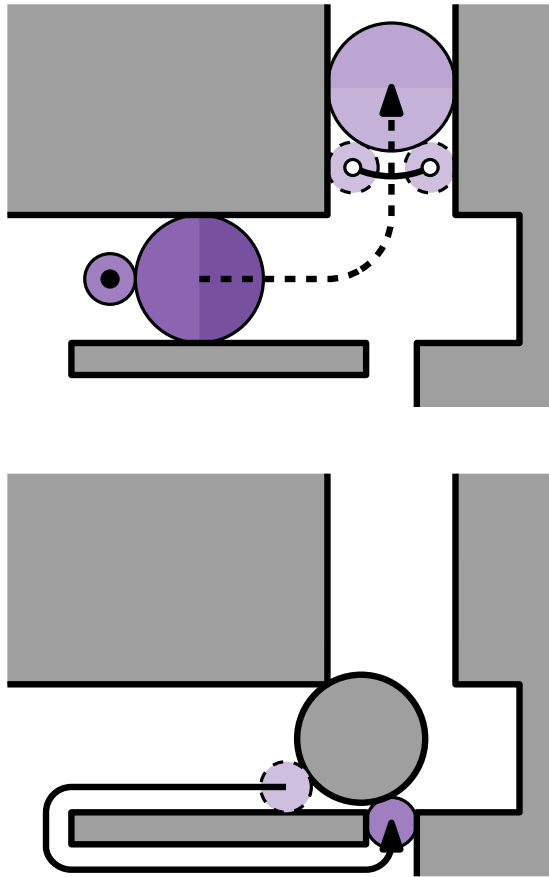
# An unrestricted push plan



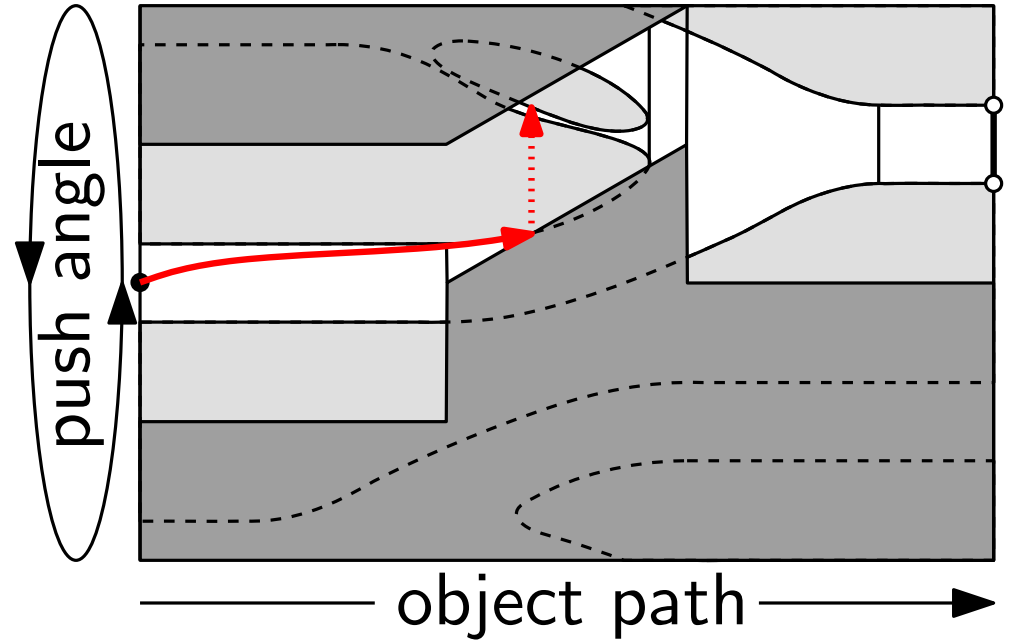
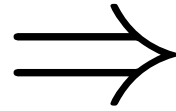
A contact-preserving push plan may not exist.



# An unrestricted push plan

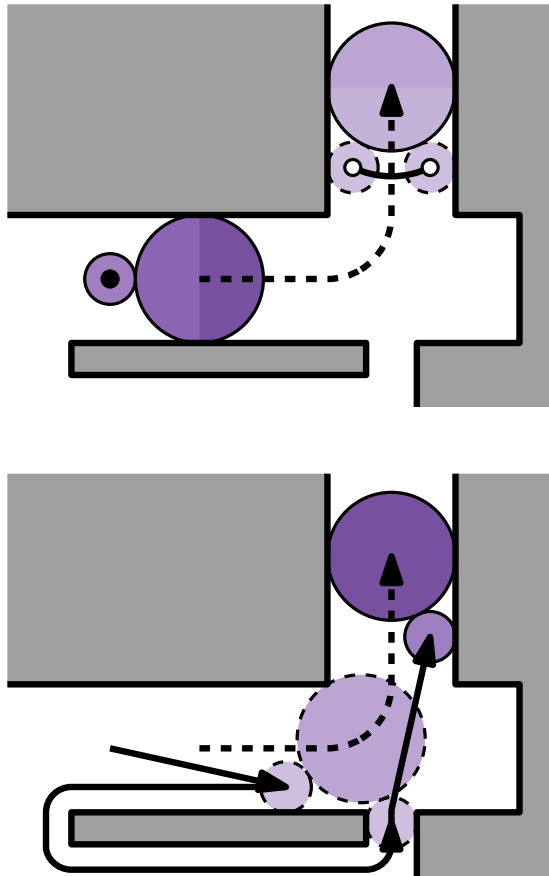


A contact-preserving push plan may not exist.

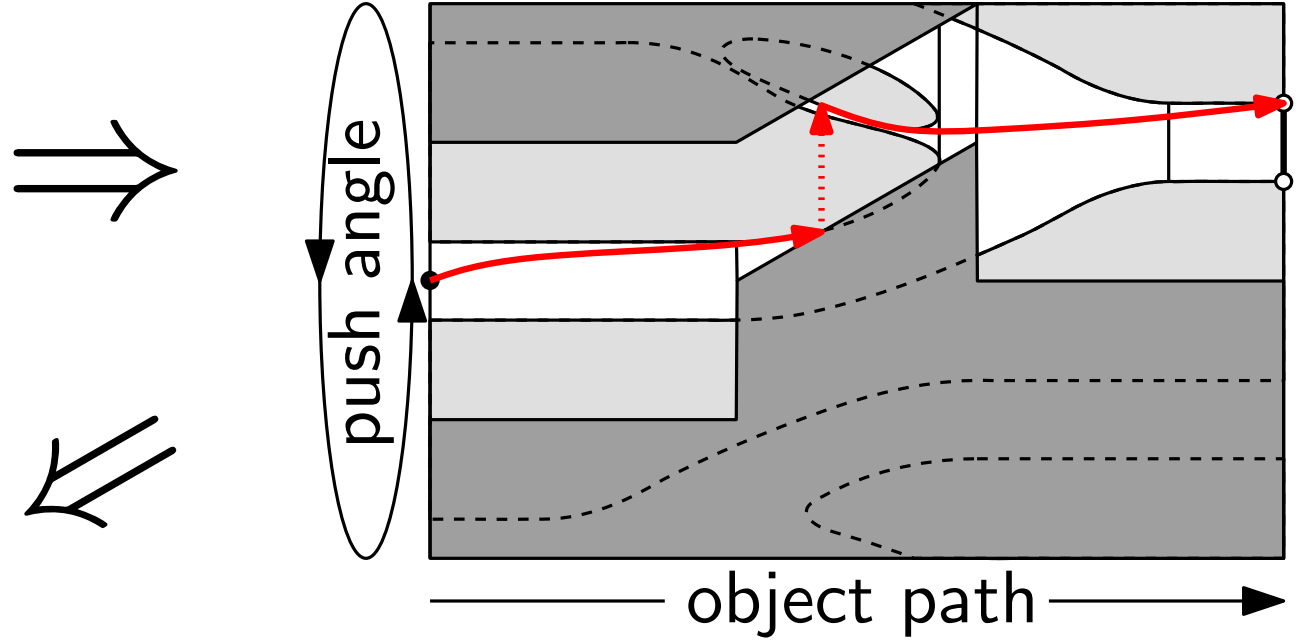


If we can find suitable **canonical releasing positions**, we can try to find an unrestricted push plan by using traditional path planning at these locations.

# An unrestricted push plan



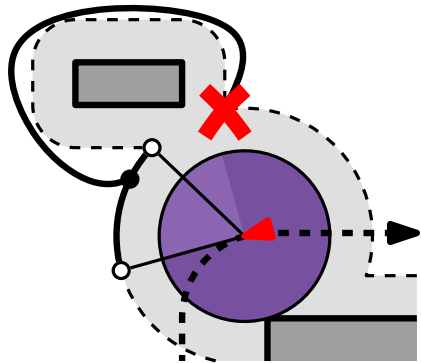
A contact-preserving push plan may not exist.



If we can find suitable **canonical releasing positions**, we can try to find an unrestricted push plan by using traditional path planning at these locations.

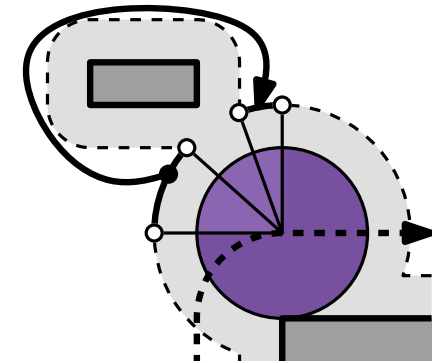
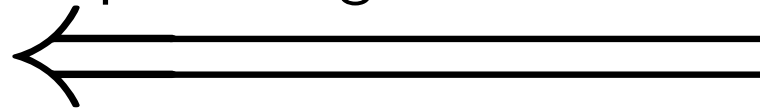
# Canonical releasing positions

Suppose a push plan has a release at a non-canonical position. “Rounding it down” to a canonical position can fail in 2 ways:

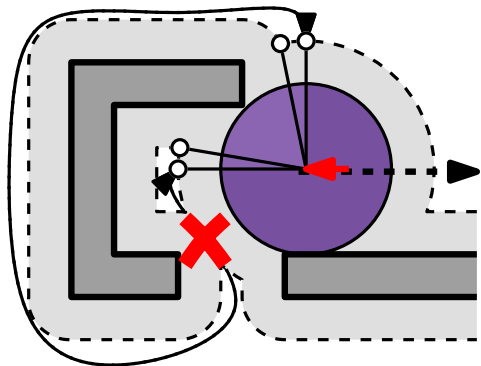


non-feasible

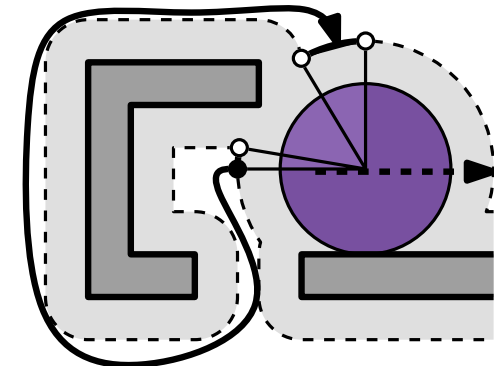
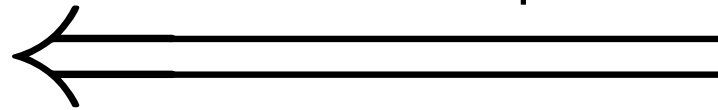
a component of the  
push range vanishes



non-canonical

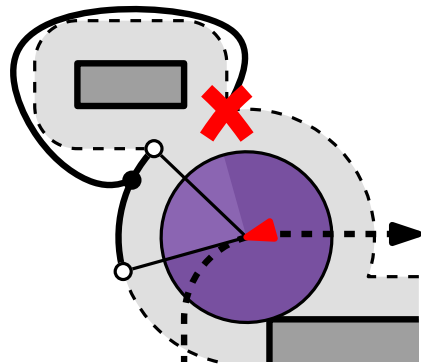


a corridor gets too  
narrow for the pusher

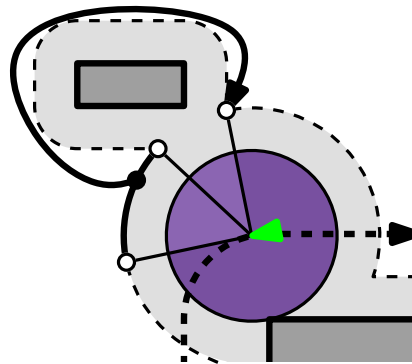
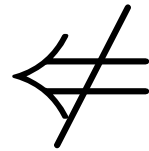


# Canonical releasing positions

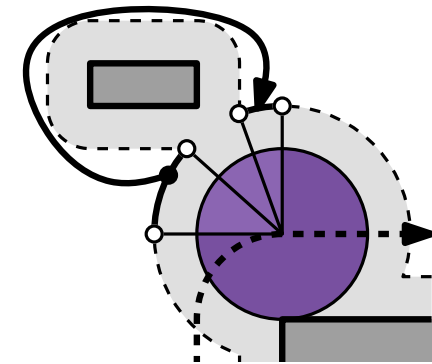
Suppose a push plan has a release at a non-canonical position. “Rounding it down” to a canonical position can fail in 2 ways:



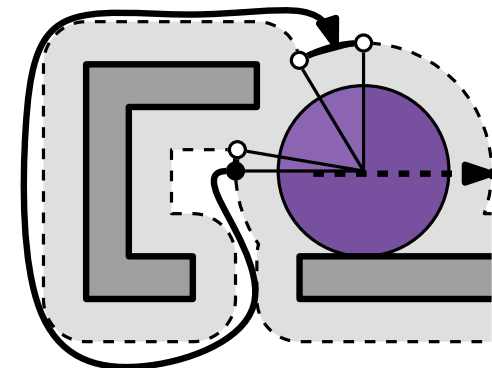
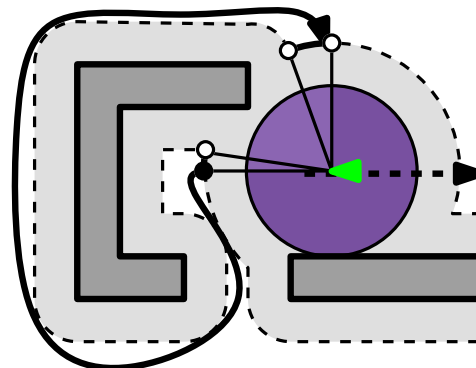
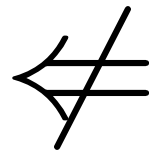
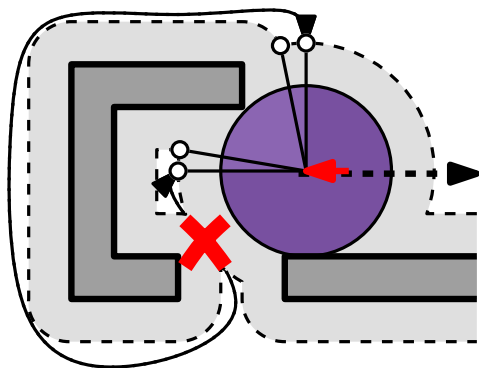
non-feasible



canonical



non-canonical



Make the positions just before failure canonical releasing points!

# Conclusion

**Running times:** ( $k$  is object-path complexity,  $n$  is # obstacle edges)

	High obstacle density		Low obstacle density	
	Nieuwenh.	Our method	Nieuwenhuisen	Our method
Preprocess	$n^2 \log n$	$n \log n$ (*)	$n^2 \log n$	$n \log n$ (*)
Any CPPP	$kn \log n$	$kn \log n$ (*)	$(k+n) \log(k+n)$	$(k+n) \log(k+n)$
A shortest CPPP	—	$kn \log(kn)$	—	$kn \log(kn)$
Any UPP	—	$kn \log(kn)$ $+ kn^2 \log n$	—	$(k+n) \log(k+n)$ $+ kn$

(\*) These entries are expected times. For worst-case times, replace  $\log n$  by  $\log^2 n$ .

# Conclusion

**Running times:** ( $k$  is object-path complexity,  $n$  is # obstacle edges)

	High obstacle density		Low obstacle density	
	Nieuwenh.	Our method	Nieuwenhuisen	Our method
Preprocess	$n^2 \log n$	$n \log n$ (*)	$n^2 \log n$	$n \log n$ (*)
Any CPPP	$kn \log n$	$kn \log n$ (*)	$(k+n) \log(k+n)$	$(k+n) \log(k+n)$
A shortest CPPP	—	$kn \log(kn)$	—	$kn \log(kn)$
Any UPP	—	$kn \log(kn)$ $+ kn^2 \log n$	—	$(k+n) \log(k+n)$ $+ kn$

(\*) These entries are expected times. For worst-case times, replace  $\log n$  by  $\log^2 n$ .

## Open questions:

- 💡 Can our running times be improved?
- 💡 Can we find shortest unrestricted push plans?
- 💡 Can we generalize to a non-disk object and/or pusher?
- 💡 Can we find push plans given only the object's destination?  
(Probabilistically complete algorithm by Nieuwenhuisen et al.)