Applying Geometric Thick Paths to Compute the Number of Additional Train Paths in a Railway Timetable

Anders Peterson Valentin Polishchuk **Christiane Schmidt**

LINKÖPING UNIVERSITY

- Marshalling yards: completed trains occupy highly demanded space until departure

- Marshalling yards: completed trains occupy highly demanded space until departure

➡ Depart ahead of schedule?

- Marshalling yards: completed trains occupy highly demanded space until departure

➡ Depart ahead of schedule?

➡ Should not contribute to congestion—ensure train path to the destination available

- Marshalling yards: completed trains occupy highly demanded space until departure

➡ Depart ahead of schedule?

➡ Should not contribute to congestion—ensure train path to the destination available

➡ Here: what if we want to add several additional trains?

- Marshalling yards: completed trains occupy highly demanded space until departure

➡ Depart ahead of schedule?

➡ Should not contribute to congestion—ensure train path to the destination available

➡ Here: what if we want to add several additional trains?

➡ Leave existing (passenger) traffic unaffected

- Marshalling yards: completed trains occupy highly demanded space until departure

➡ Depart ahead of schedule?

➡ Should not contribute to congestion—ensure train path to the destination available

➡ Here: what if we want to add several additional trains?

➡ Leave existing (passenger) traffic unaffected

➡ Q: How many can we add?

- Marshalling yards: completed trains occupy highly demanded space until departure

➡ Depart ahead of schedule?

➡ Should not contribute to congestion—ensure train path to the destination available

➡ Here: what if we want to add several additional trains?

➡ Leave existing (passenger) traffic unaffected

➡ Q: How many can we add?

➡ Residual capacity for additional train paths in a given time window

- Marshalling yards: completed trains occupy highly demanded space until departure

➡ Depart ahead of schedule?
➡ Should not contribute to congestion—ensure train path to the destination available
➡ Here: what if we want to add several additional trains?
➡ Leave existing (passenger) traffic unaffected
➡ Q: How many can we add?
➡ Residual capacity for additional train paths in a given time window

- UIC compression technique for computing capacity utilization:

- Marshalling yards: completed trains occupy highly demanded space until departure

➡ Depart ahead of schedule?
➡ Should not contribute to congestion—ensure train path to the destination available
➡ Here: what if we want to add several additional trains?
➡ Leave existing (passenger) traffic unaffected
➡ Q: How many can we add?
➡ Residual capacity for additional train paths in a given time window

- UIC compression technique for computing capacity utilization:
    - Compresses the timetable: existing train paths on the considered line section are shifted as close together as possible

- Marshalling yards: completed trains occupy highly demanded space until departure

➡ Depart ahead of schedule?

➡ Should not contribute to congestion—ensure train path to the destination available

➡ Here: what if we want to add several additional trains?

➡ Leave existing (passenger) traffic unaffected

➡ Q: How many can we add?

➡ Residual capacity for additional train paths in a given time window

- UIC compression technique for computing capacity utilization:
    - Compresses the timetable: existing train paths on the considered line section are shifted as close together as possible

    ➡ Trains no longer considered at the time at which they actually run

- Marshalling yards: completed trains occupy highly demanded space until departure

➡ Depart ahead of schedule?

➡ Should not contribute to congestion—ensure train path to the destination available

➡ Here: what if we want to add several additional trains?

➡ Leave existing (passenger) traffic unaffected

➡ Q: How many can we add?

➡ Residual capacity for additional train paths in a given time window

- UIC compression technique for computing capacity utilization:
  - Compresses the timetable: existing train paths on the considered line section are shifted as close together as possible

    ➡ Trains no longer considered at the time at which they actually run
- Saturation problem:

- Marshalling yards: completed trains occupy highly demanded space until departure

➡ Depart ahead of schedule?

➡ Should not contribute to congestion—ensure train path to the destination available

➡ Here: what if we want to add several additional trains?

➡ Leave existing (passenger) traffic unaffected

➡ Q: How many can we add?

➡ Residual capacity for additional train paths in a given time window

- UIC compression technique for computing capacity utilization:
    - Compresses the timetable: existing train paths on the considered line section are shifted as close together as possible

      ➡ Trains no longer considered at the time at which they actually run
- Saturation problem:

  Given: Existing (possibly empty) timetable, a set of saturation trains

- Marshalling yards: completed trains occupy highly demanded space until departure

➡ Depart ahead of schedule?

➡ Should not contribute to congestion—ensure train path to the destination available

➡ Here: what if we want to add several additional trains?

➡ Leave existing (passenger) traffic unaffected

➡ Q: How many can we add?

➡ Residual capacity for additional train paths in a given time window

- UIC compression technique for computing capacity utilization:
  - Compresses the timetable: existing train paths on the considered line section are shifted as close together as possible

    ➡ Trains no longer considered at the time at which they actually run
- Saturation problem:

  Given: Existing (possibly empty) timetable, a set of saturation trains

  Goal: Add as many trains as possible

- Marshalling yards: completed trains occupy highly demanded space until departure

➡ Depart ahead of schedule?

➡ Should not contribute to congestion—ensure train path to the destination available

➡ Here: what if we want to add several additional trains?

➡ Leave existing (passenger) traffic unaffected

➡ Q: How many can we add?

➡ Residual capacity for additional train paths in a given time window

- UIC compression technique for computing capacity utilization:
  - Compresses the timetable: existing train paths on the considered line section are shifted as close together as possible

    ➡ Trains no longer considered at the time at which they actually run
- Saturation problem:

  Given: Existing (possibly empty) timetable, a set of saturation trains

  Goal: Add as many trains as possible

  - Various train types considered

- Marshalling yards: completed trains occupy highly demanded space until departure

➡ Depart ahead of schedule?

➡ Should not contribute to congestion—ensure train path to the destination available

➡ Here: what if we want to add several additional trains?

➡ Leave existing (passenger) traffic unaffected

➡ Q: How many can we add?

➡ Residual capacity for additional train paths in a given time window

- UIC compression technique for computing capacity utilization:
    - Compresses the timetable: existing train paths on the considered line section are shifted as close together as possible

      ➡ Trains no longer considered at the time at which they actually run
- Saturation problem:

  Given: Existing (possibly empty) timetable, a set of saturation trains

  Goal: Add as many trains as possible

  - Various train types considered
  - Solved with heuristics (CAPRES) or MILP (Pellegrini et al., 2017)

- Marshalling yards: completed trains occupy highly demanded space until departure

➡ Depart ahead of schedule?

➡ Should not contribute to congestion—ensure train path to the destination available

➡ Here: what if we want to add several additional trains?

➡ Leave existing (passenger) traffic unaffected

➡ Q: How many can we add?

➡ Residual capacity for additional train paths in a given time window

- UIC compression technique for computing capacity utilization:
  - Compresses the timetable: existing train paths on the considered line section are shifted as close together as possible

    ➡ Trains no longer considered at the time at which they actually run
- Saturation problem:

  Given: Existing (possibly empty) timetable, a set of saturation trains

  Goal: Add as many trains as possible
  - Various train types considered
  - Solved with heuristics (CAPRES) or MILP (Pellegrini et al., 2017)
- We:

- Marshalling yards: completed trains occupy highly demanded space until departure

➡ Depart ahead of schedule?
➡ Should not contribute to congestion—ensure train path to the destination available
➡ Here: what if we want to add several additional trains?
➡ Leave existing (passenger) traffic unaffected
➡ Q: How many can we add?
➡ Residual capacity for additional train paths in a given time window

- UIC compression technique for computing capacity utilization:
    - Compresses the timetable: existing train paths on the considered line section are shifted as close together as possible

      ➡ Trains no longer considered at the time at which they actually run
- Saturation problem:

  Given: Existing (possibly empty) timetable, a set of saturation trains

  Goal: Add as many trains as possible
    - Various train types considered
    - Solved with heuristics (CAPRES) or MILP (Pellegrini et al., 2017)
- We:
    - Consider a single type (outlook on several types given)

- Marshalling yards: completed trains occupy highly demanded space until departure

➡ Depart ahead of schedule?

➡ Should not contribute to congestion—ensure train path to the destination available

➡ Here: what if we want to add several additional trains?

➡ Leave existing (passenger) traffic unaffected

➡ Q: How many can we add?

➡ Residual capacity for additional train paths in a given time window

- UIC compression technique for computing capacity utilization:
  - Compresses the timetable: existing train paths on the considered line section are shifted as close together as possible

    ➡ Trains no longer considered at the time at which they actually run
- Saturation problem:

  Given: Existing (possibly empty) timetable, a set of saturation trains

  Goal: Add as many trains as possible
  - Various train types considered
  - Solved with heuristics (CAPRES) or MILP (Pellegrini et al., 2017)
- We:
  - Consider a single type (outlook on several types given)
  - Aim to disturb passenger traffic as little as possible—trade-off with temporal distance to other trains

- Marshalling yards: completed trains occupy highly demanded space until departure

➡ Depart ahead of schedule?

➡ Should not contribute to congestion—ensure train path to the destination available

➡ Here: what if we want to add several additional trains?

➡ Leave existing (passenger) traffic unaffected

➡ Q: How many can we add?

➡ Residual capacity for additional train paths in a given time window

- UIC compression technique for computing capacity utilization:
  - Compresses the timetable: existing train paths on the considered line section are shifted as close together as possible

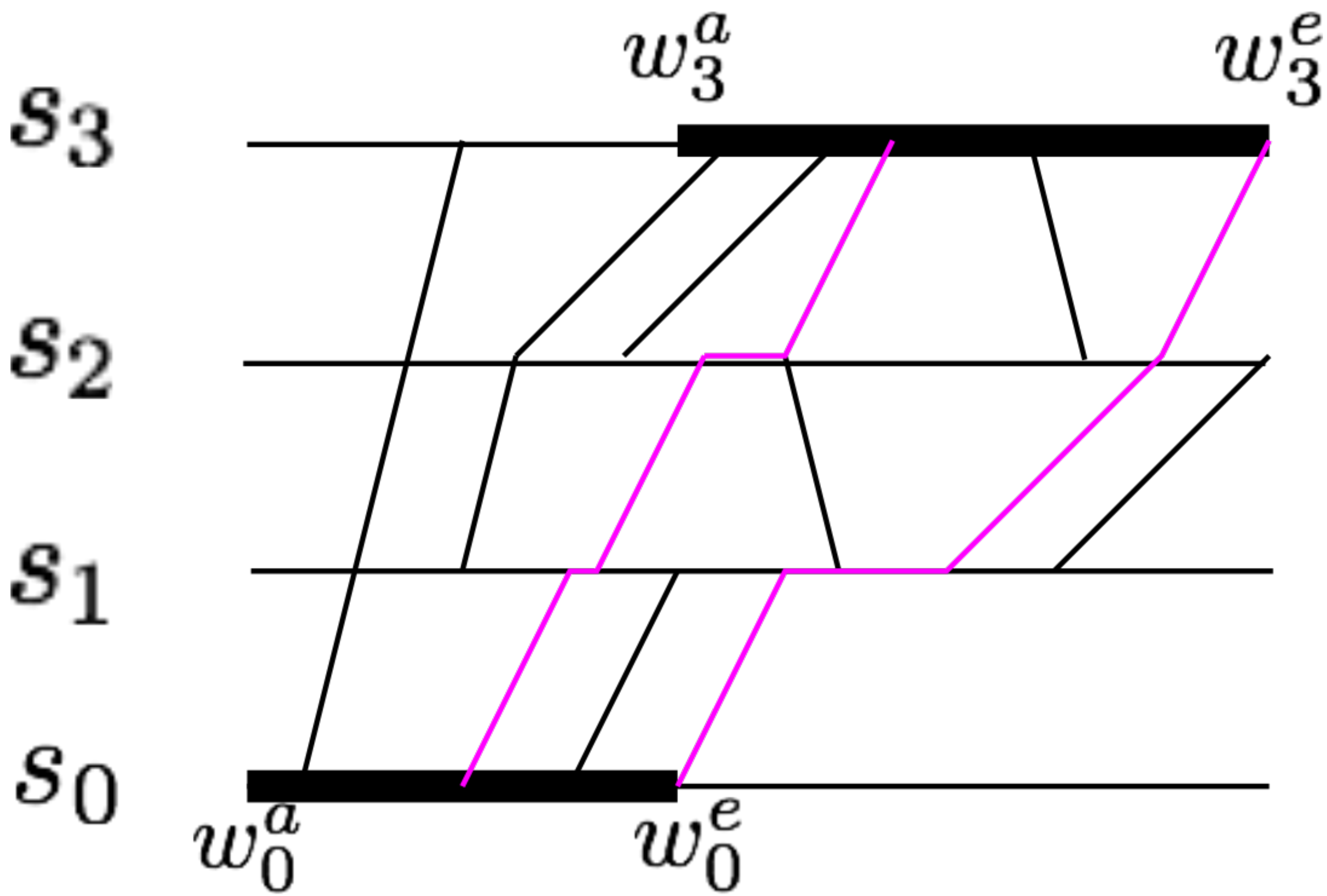
    ➡ Trains no longer considered at the time at which they actually run
- Saturation problem:

  Given: Existing (possibly empty) timetable, a set of saturation trains

  Goal: Add as many trains as possible
  - Various train types considered
  - Solved with heuristics (CAPRES) or MILP (Pellegrini et al., 2017)
- We:
  - Consider a single type (outlook on several types given)
  - Aim to disturb passenger traffic as little as possible—trade-off with temporal distance to other trains
  - We present optimal solution

Existing trains

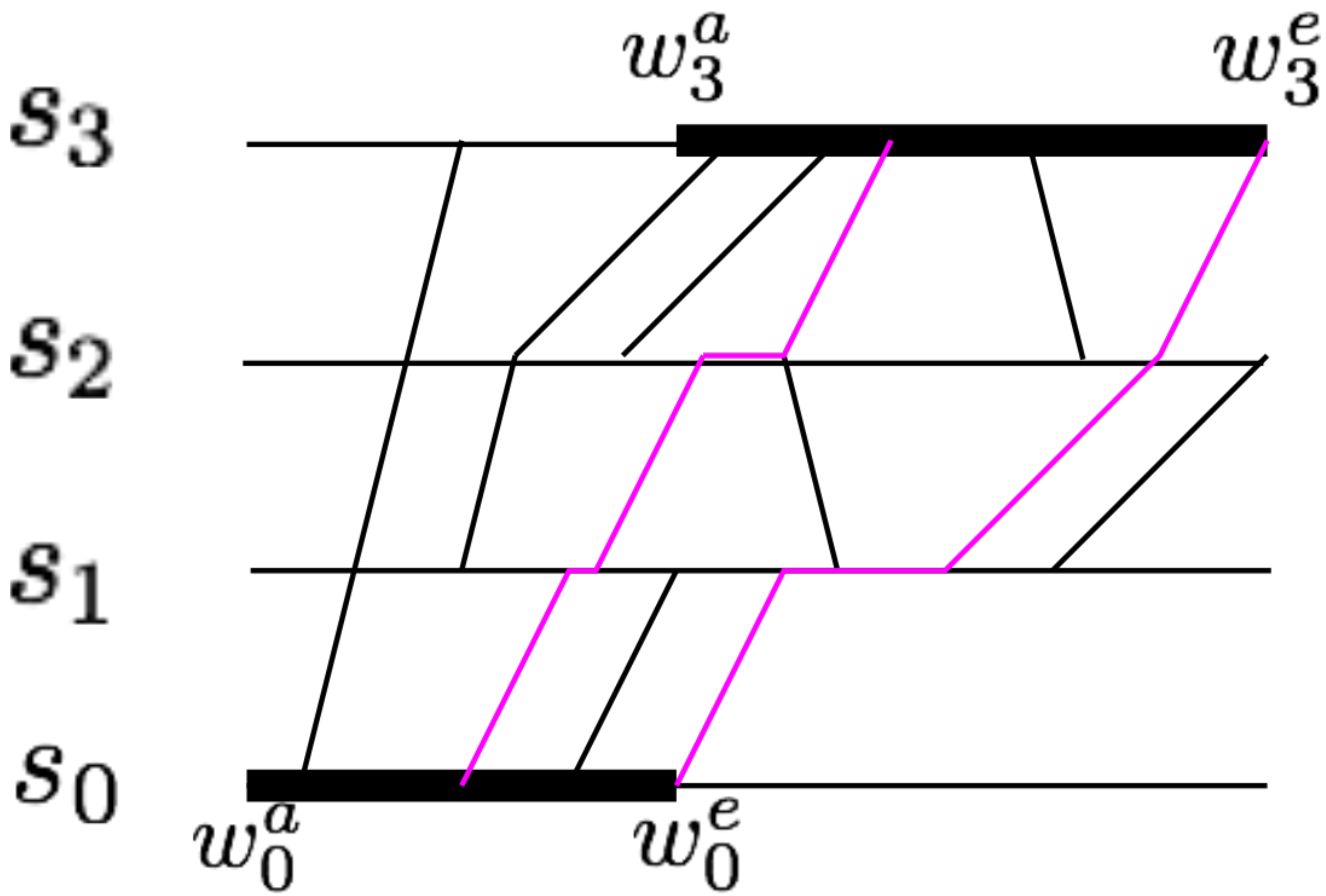
Additional trains: need to keep temporal distance



Existing trains

Additional trains: need to keep temporal distance

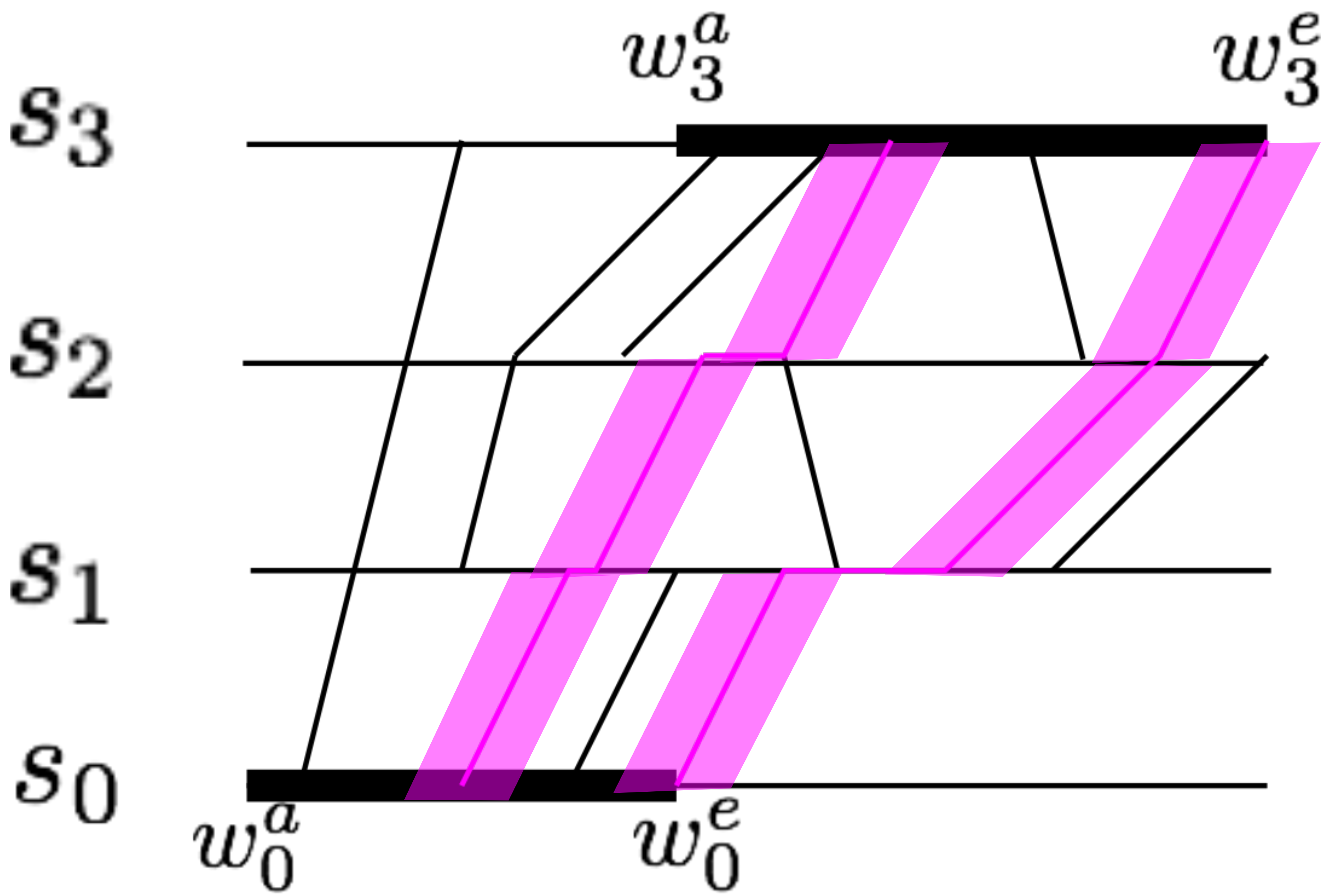
➡ Thick paths instead of lines



Existing trains

Additional trains: need to keep temporal distance

➡ Thick paths instead of lines



- We consider the time-space diagram—the geometric representation

- We consider the time-space diagram—the geometric representation
- Inserting a new train: Route path from start to end station

- We consider the time-space diagram—the geometric representation
- Inserting a new train: Route path from start to end station
- Paths not arbitrarily close ⇨ temporal distance (different to trains running in same or opposite direction)

- We consider the time-space diagram—the geometric representation
- Inserting a new train: Route path from start to end station
- Paths not arbitrarily close ⇨ temporal distance (different to trains running in same or opposite direction)
- We think of train paths as “blown-up” line segments = thick paths

- We consider the time-space diagram—the geometric representation
- Inserting a new train: Route path from start to end station
- Paths not arbitrarily close ⇨ temporal distance (different to trains running in same or opposite direction)
- We think of train paths as “blown-up” line segments = thick paths
- Blown up by temporal distance (can be minimum, or more)

- We consider the time-space diagram—the geometric representation
- Inserting a new train: Route path from start to end station
- Paths not arbitrarily close ⇨ temporal distance (different to trains running in same or opposite direction)
- We think of train paths as “blown-up” line segments = thick paths
- Blown up by temporal distance (can be minimum, or more)
- How to route those thick paths? Concepts from Computational Geometry

- We consider the time-space diagram—the geometric representation
- Inserting a new train: Route path from start to end station
- Paths not arbitrarily close ⇨ temporal distance (different to trains running in same or opposite direction)
- We think of train paths as "blown-up" line segments = thick paths
- Blown up by temporal distance (can be minimum, or more)
- How to route those thick paths? Concepts from Computational Geometry
- Need to make some adaptations, for example, if stations are lines, no path could cross these

- We consider the time-space diagram—the geometric representation
- Inserting a new train: Route path from start to end station
- Paths not arbitrarily close ⇨ temporal distance (different to trains running in same or opposite direction)
- We think of train paths as “blown-up” line segments = thick paths
- Blown up by temporal distance (can be minimum, or more)
- How to route those thick paths? Concepts from Computational Geometry
- Need to make some adaptations, for example, if stations are lines, no path could cross these

➡ 1. Show how to construct the appropriate polygonal domain

- We consider the time-space diagram—the geometric representation
- Inserting a new train: Route path from start to end station
- Paths not arbitrarily close ⇨ temporal distance (different to trains running in same or opposite direction)
- We think of train paths as “blown-up” line segments = thick paths
- Blown up by temporal distance (can be minimum, or more)
- How to route those thick paths? Concepts from Computational Geometry
- Need to make some adaptations, for example, if stations are lines, no path could cross these

➡ 1. Show how to construct the appropriate polygonal domain

➡ 2. Show how to route the maximum number of thick non-crossing paths in that domain:

- We consider the time-space diagram—the geometric representation
- Inserting a new train: Route path from start to end station
- Paths not arbitrarily close ⇨ temporal distance (different to trains running in same or opposite direction)
- We think of train paths as “blown-up” line segments = thick paths
- Blown up by temporal distance (can be minimum, or more)
- How to route those thick paths? Concepts from Computational Geometry
- Need to make some adaptations, for example, if stations are lines, no path could cross these

➡ 1. Show how to construct the appropriate polygonal domain

➡ 2. Show how to route the maximum number of thick non-crossing paths in that domain:
  - Paths should be x-monotone (we cannot go back in time)

- We consider the time-space diagram—the geometric representation
- Inserting a new train: Route path from start to end station
- Paths not arbitrarily close ⇨ temporal distance (different to trains running in same or opposite direction)
- We think of train paths as “blown-up” line segments = thick paths
- Blown up by temporal distance (can be minimum, or more)
- How to route those thick paths? Concepts from Computational Geometry
- Need to make some adaptations, for example, if stations are lines, no path could cross these

➡ 1. Show how to construct the appropriate polygonal domain

➡ 2. Show how to route the maximum number of thick non-crossing paths in that domain:

  - Paths should be x-monotone (we cannot go back in time)
  - Trains have a maximum speed ⇨ paths have a limited slope

- We consider the time-space diagram—the geometric representation
- Inserting a new train: Route path from start to end station
- Paths not arbitrarily close ⇨ temporal distance (different to trains running in same or opposite direction)
- We think of train paths as “blown-up” line segments = thick paths
- Blown up by temporal distance (can be minimum, or more)
- How to route those thick paths? Concepts from Computational Geometry
- Need to make some adaptations, for example, if stations are lines, no path could cross these

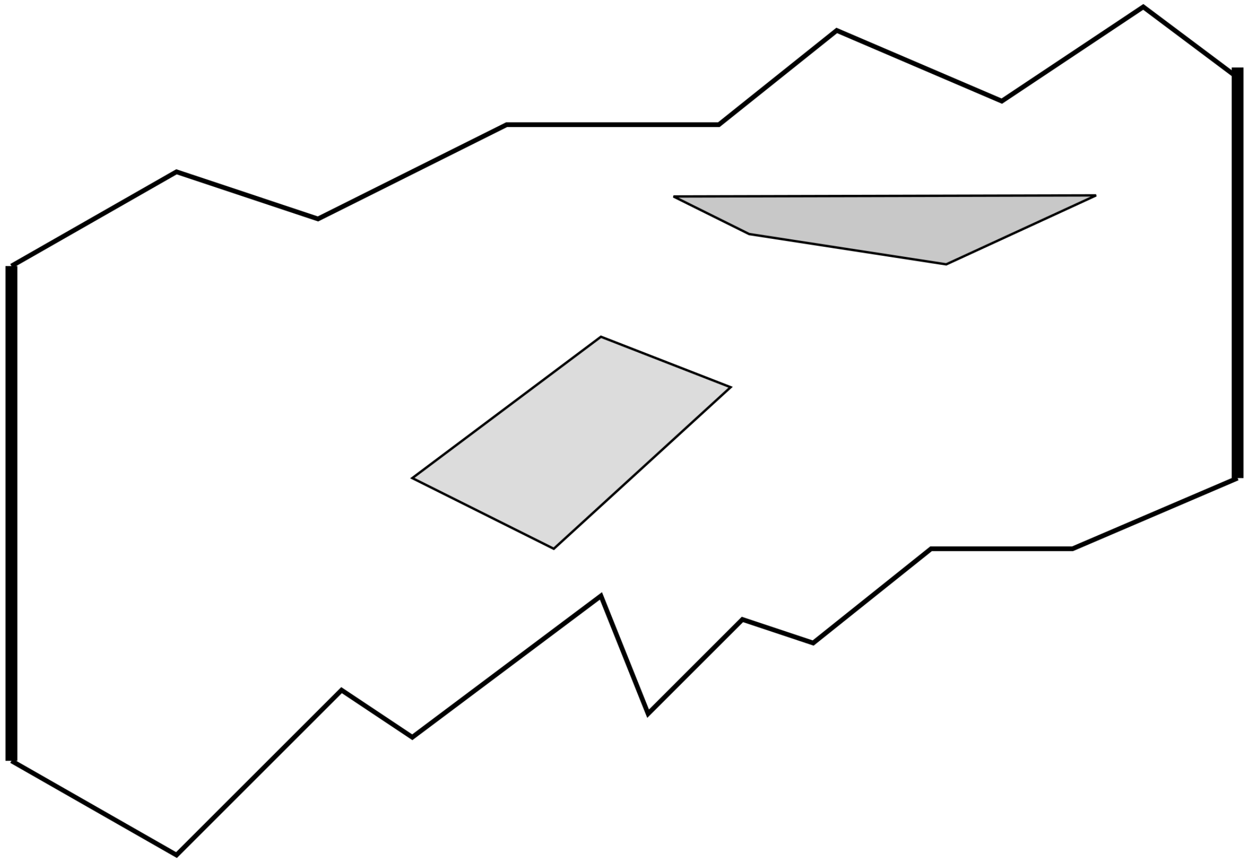
➡ 1. Show how to construct the appropriate polygonal domain

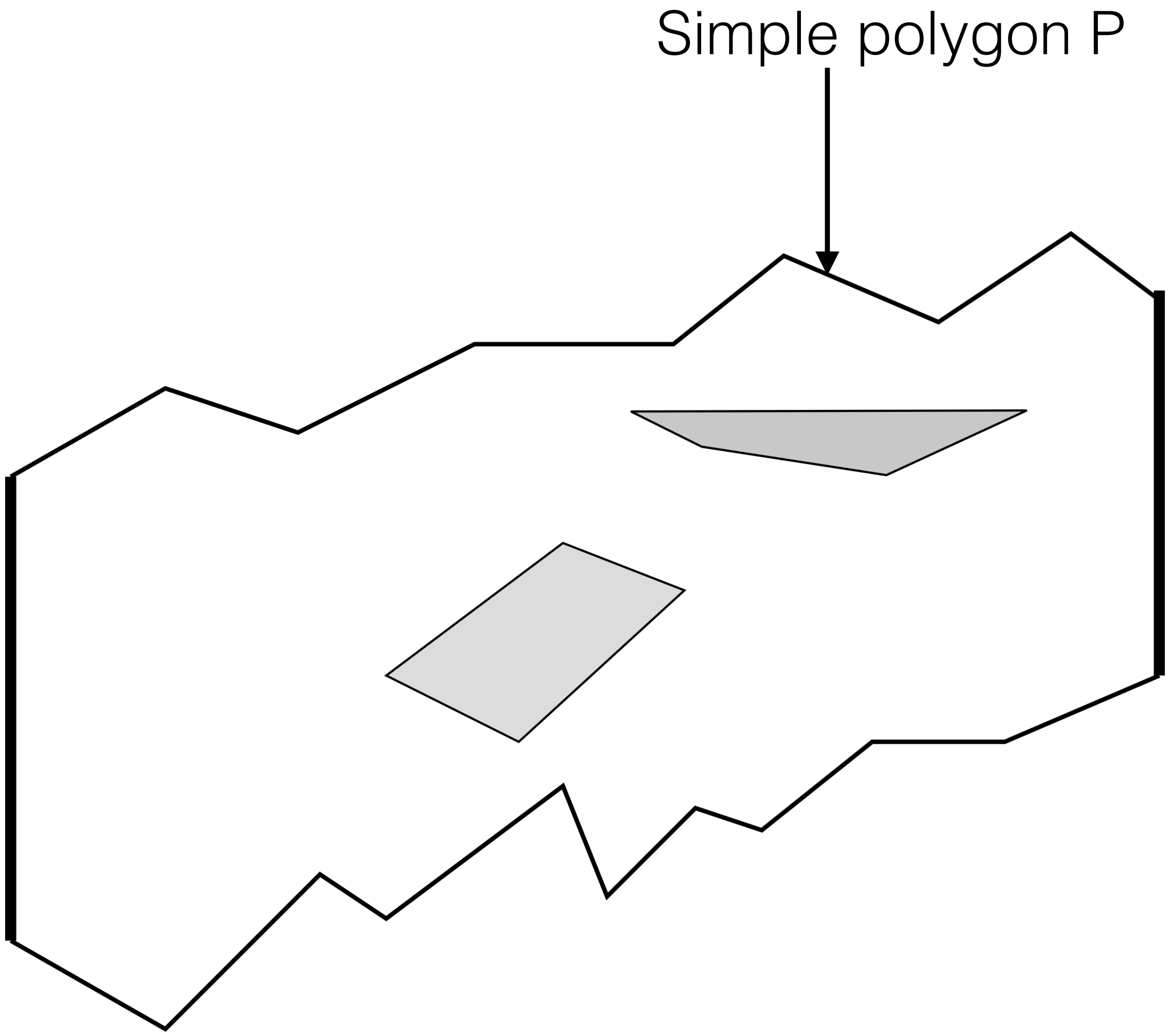
➡ 2. Show how to route the maximum number of thick non-crossing paths in that domain:

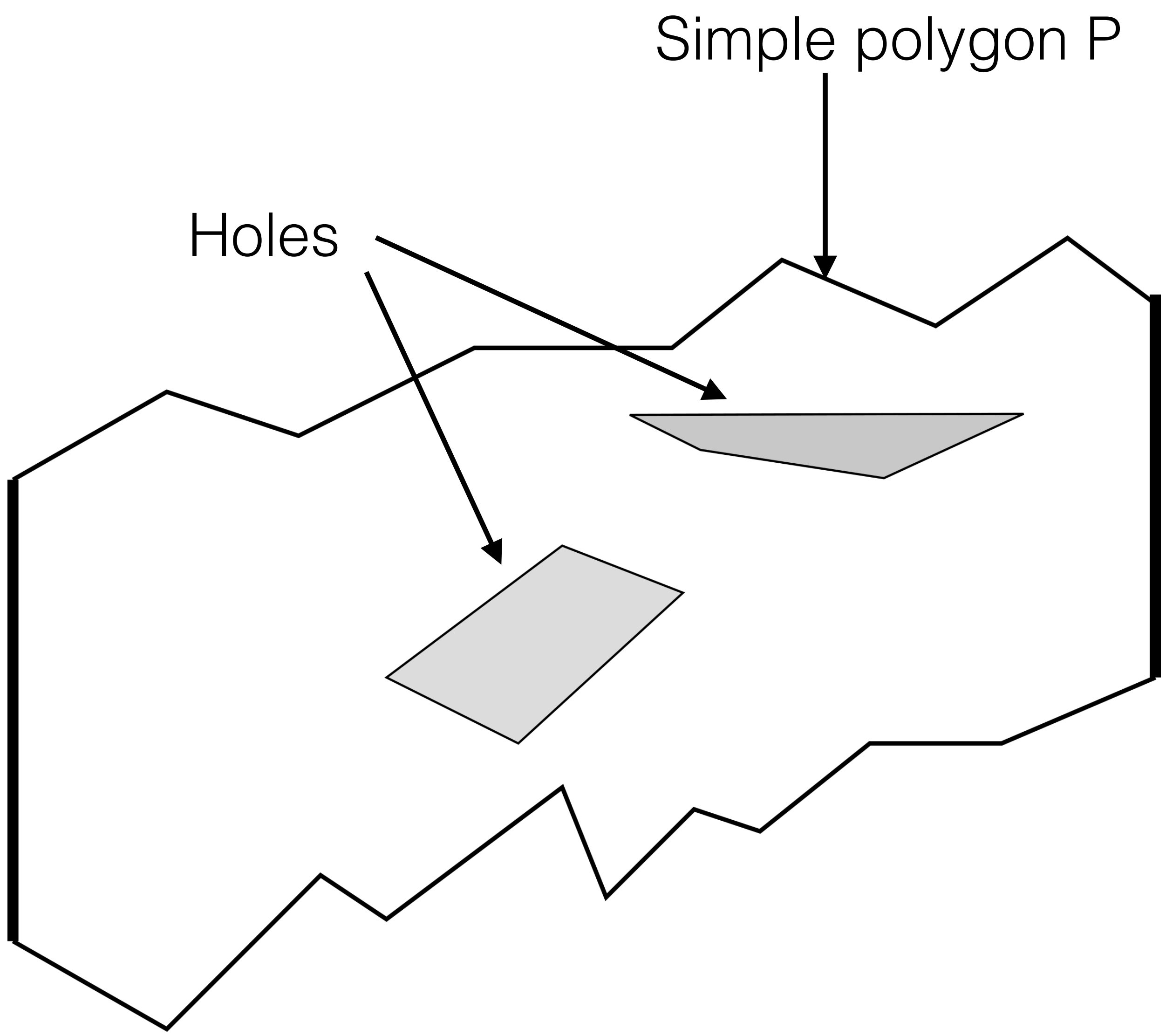
  - Paths should be x-monotone (we cannot go back in time)
  - Trains have a maximum speed ⇨ paths have a limited slope

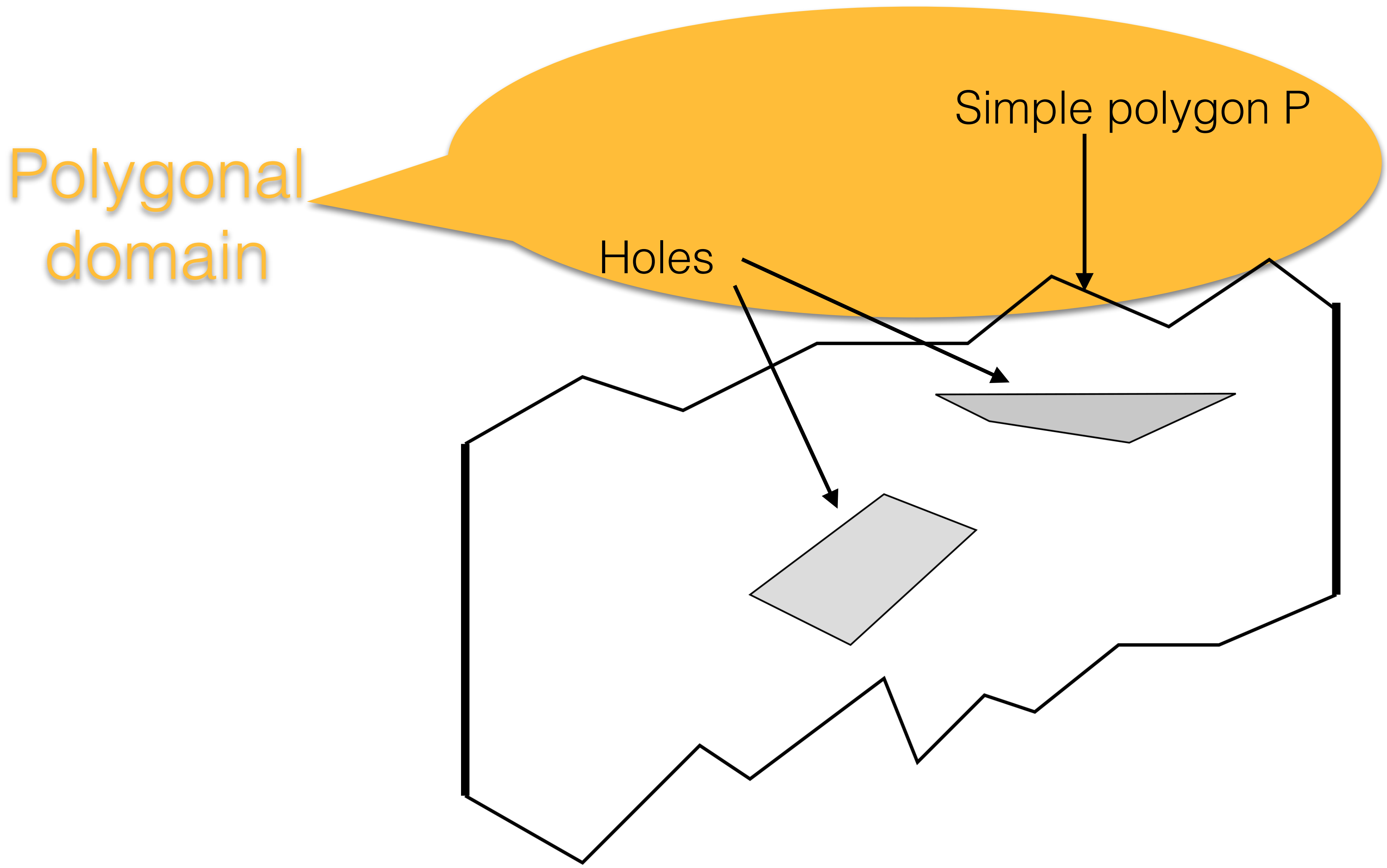
We start with 2

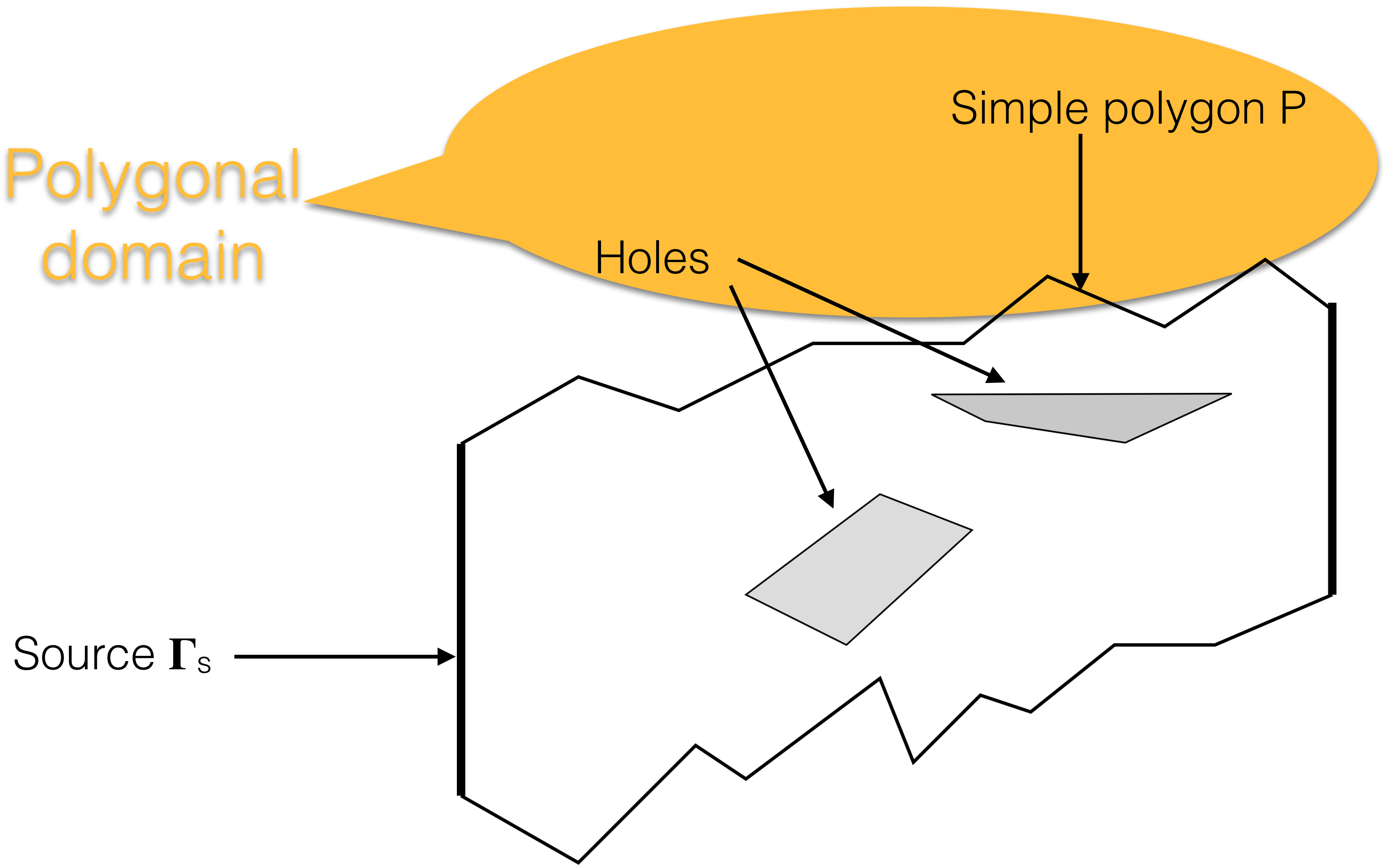
Routing a Maximum Number of Thick Paths through a Polygonal Domain



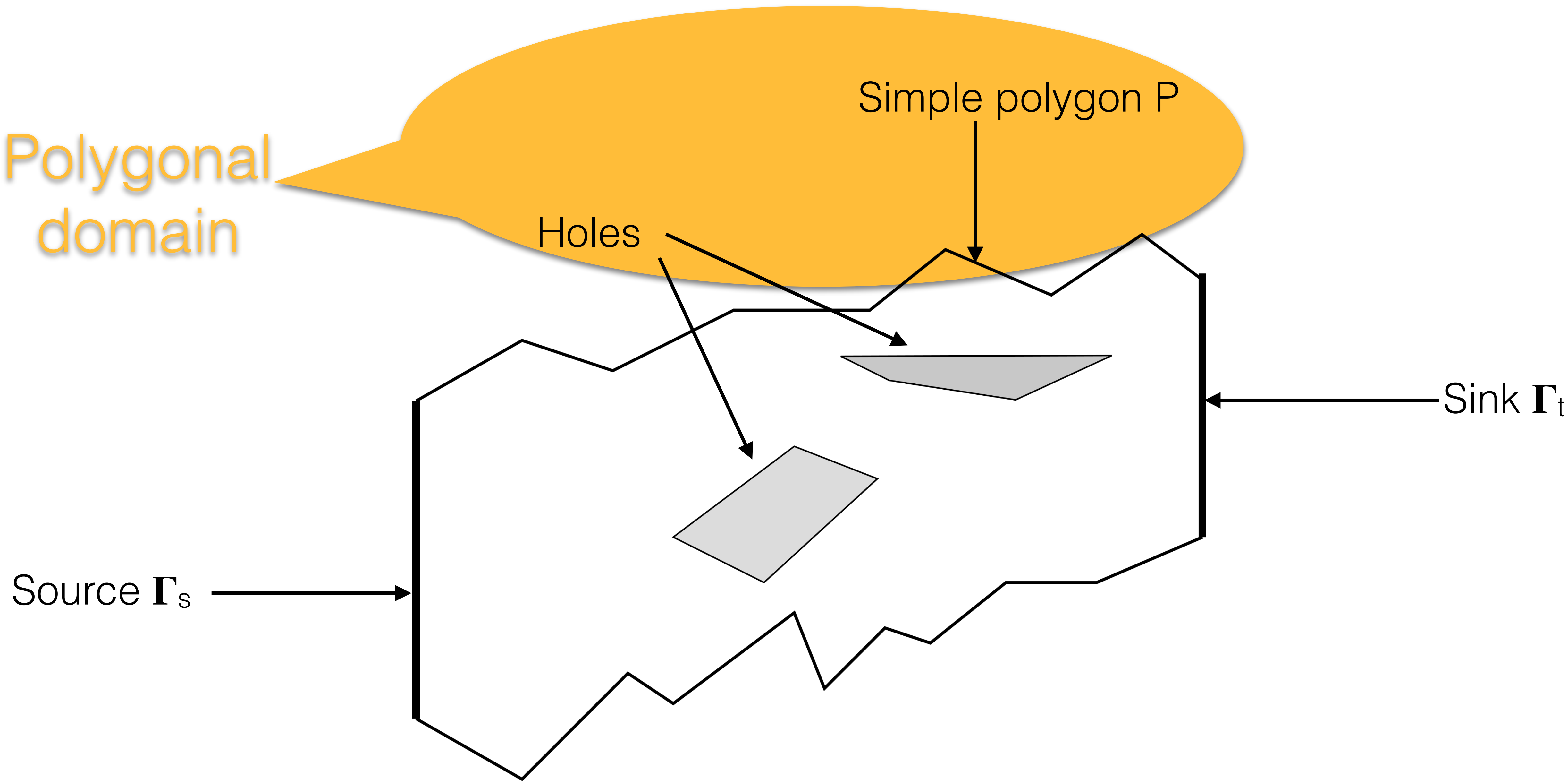

Simple polygon P

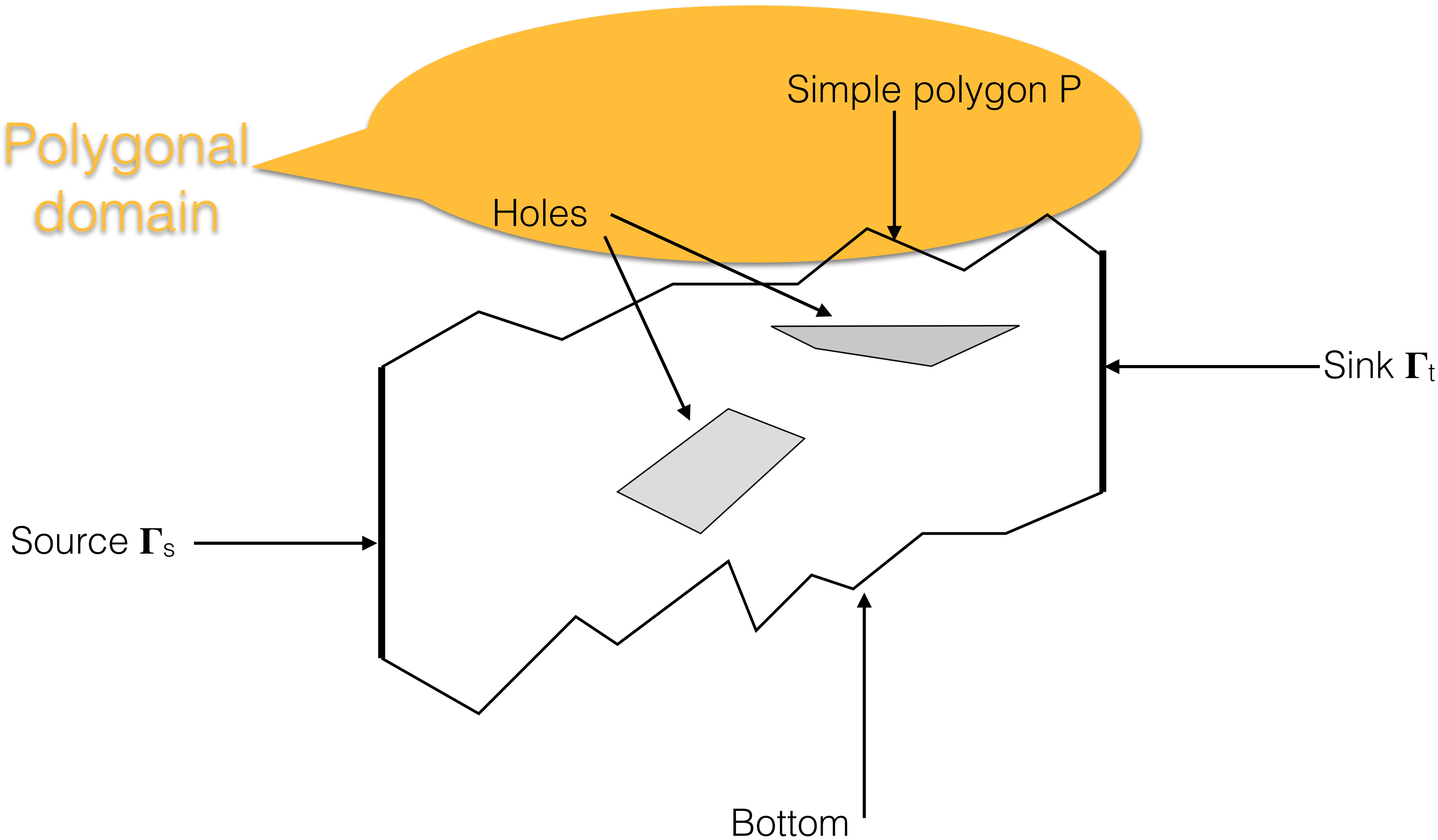

Simple polygon P
Holes

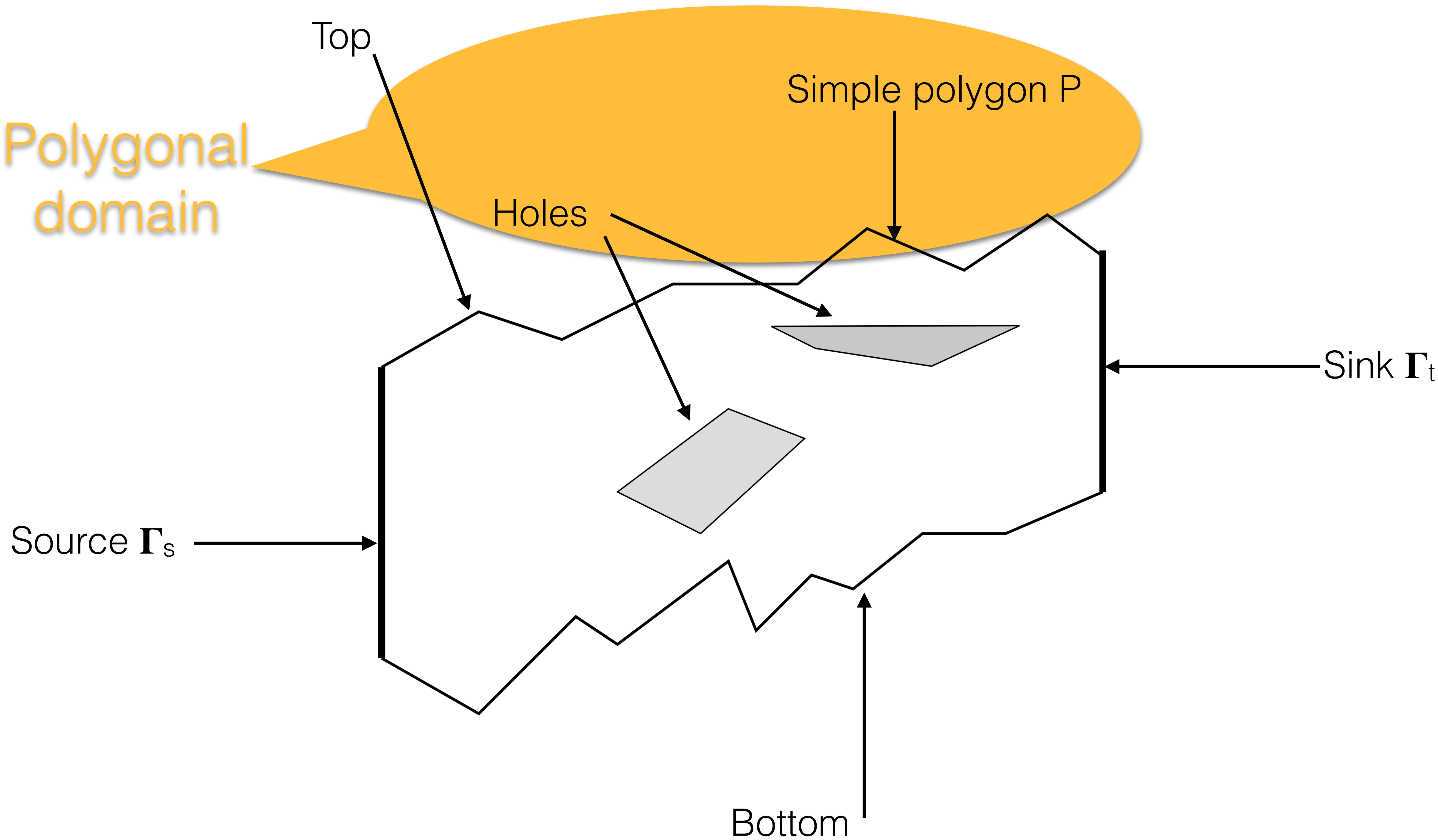

Polygonal
domain
Simple polygon P
Holes

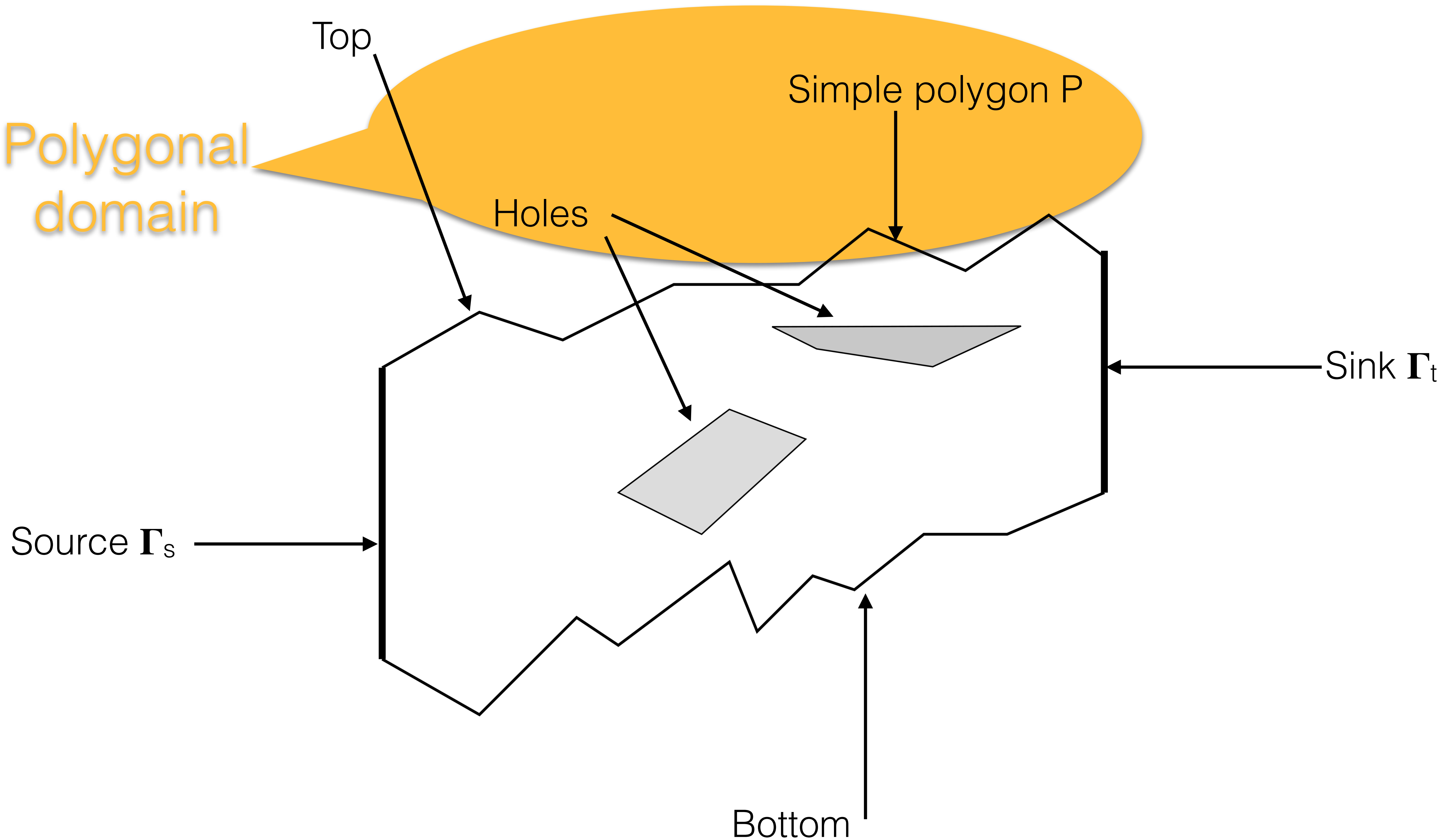


Polygonal domain
Simple polygon P
Holes
Source Γ_s


Polygonal domain
Simple polygon P
Holes
Sink Γ_t
Source Γ_s

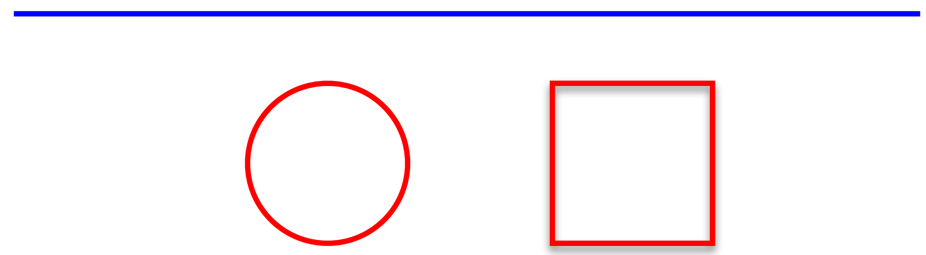

Polygonal domain
Simple polygon P
Holes
Sink $\mathbf{\Gamma}_t$
Source $\mathbf{\Gamma}_s$
Bottom


Top
Simple polygon P
Polygonal domain
Holes
Sink $\mathbf{\Gamma}_t$
Source $\mathbf{\Gamma}_s$
Bottom

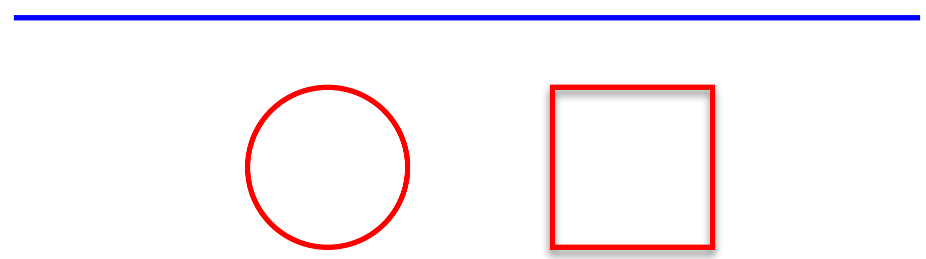


Route thick paths from the source to the sink, avoiding all holes (=obstacles)

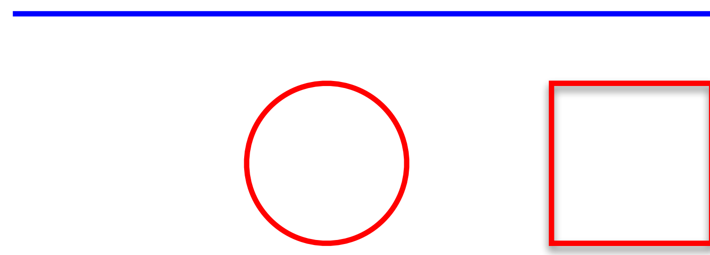
Thin path π: simple curve



Thin path π: simple curve
Let C_r denote the open desk of radius r centered at the origin



Thin path π: simple curve
Let C_r denote the open desk of radius r centered at the origin
For $S \subset R^2$: $(S)^r = S \oplus C_r = \{x+y \mid x \in S, y \in C_r\}$ - Minkowski sum

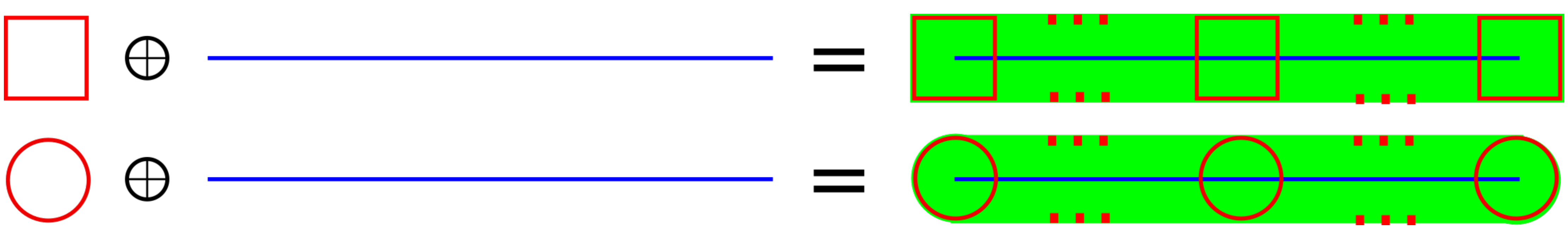


Thin path π: simple curve

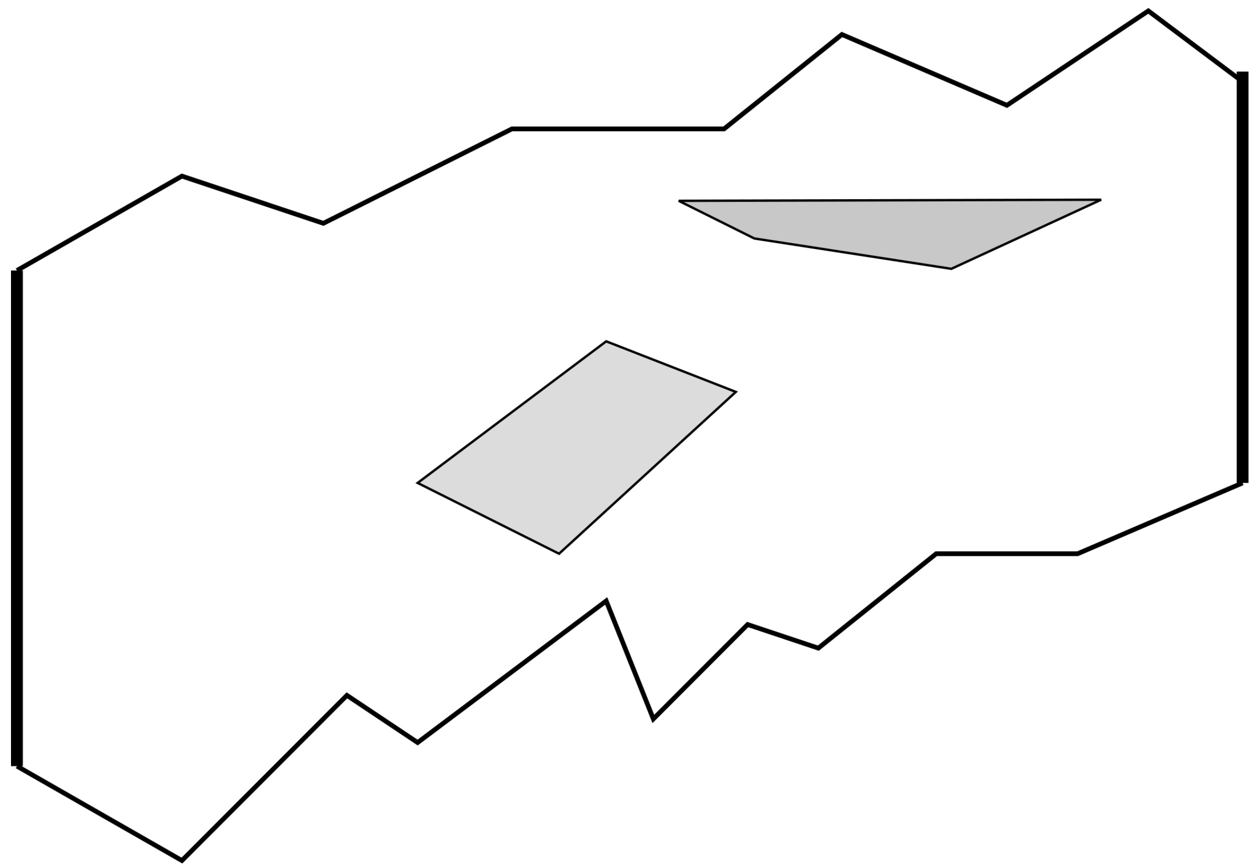
Let C_r denote the open desk of radius r centered at the origin

For $S \subset R^2$: $(S)^r = S \oplus C_r = \{x+y \mid x \in S, y \in C_r\}$ - Minkowski sum

Thick path $\mathbf{\Pi}$: Minkowski sum of a thin path and a unit disk $\mathbf{\Pi}=(\pi)^1$

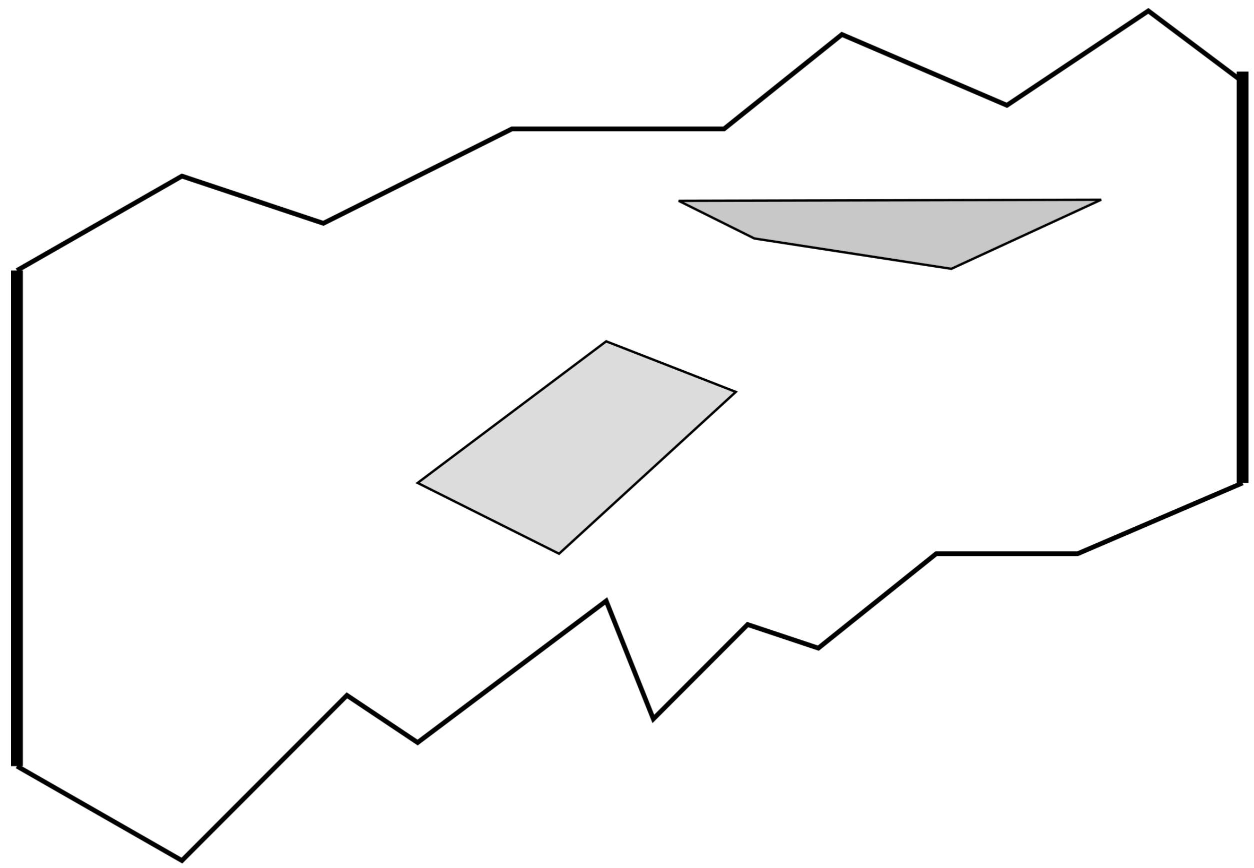


We want:



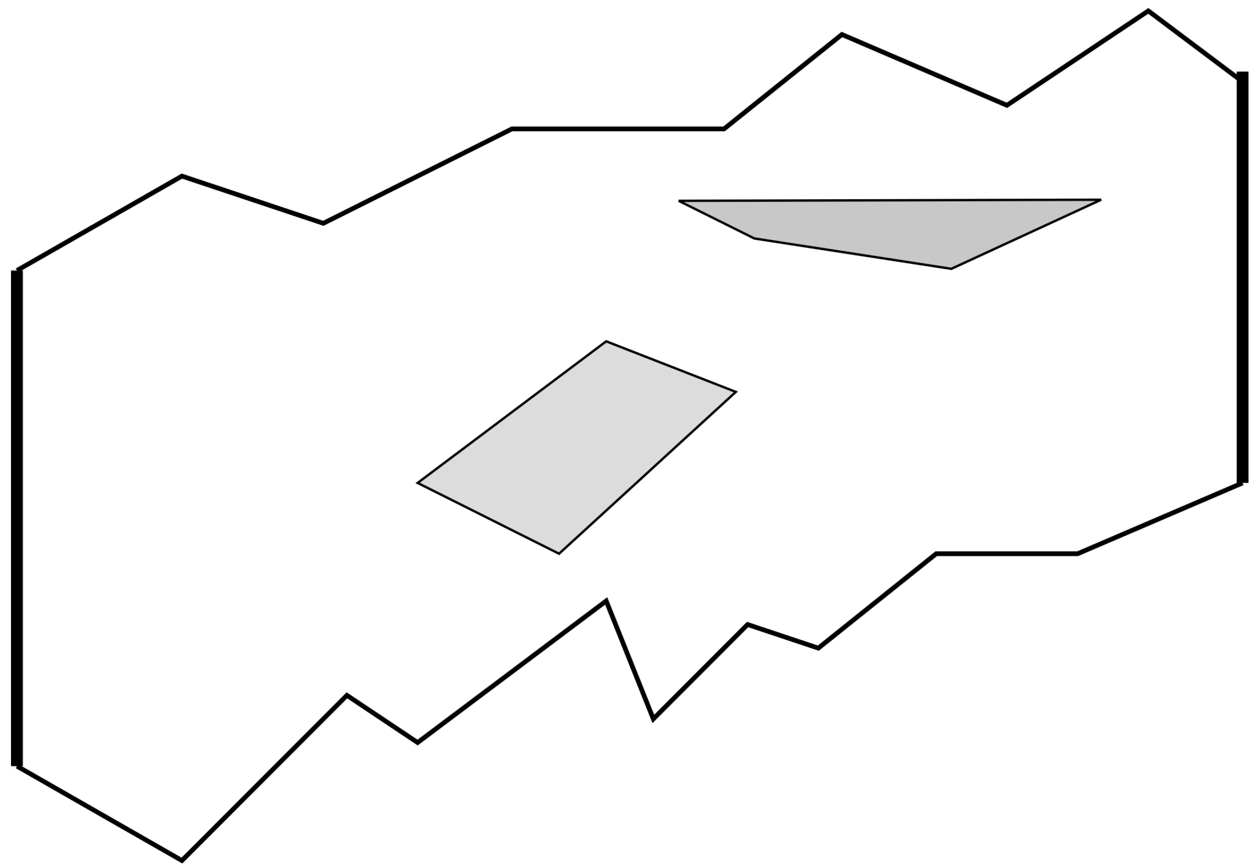
We want:

- Maximum number of non crossing thick paths from source to sink



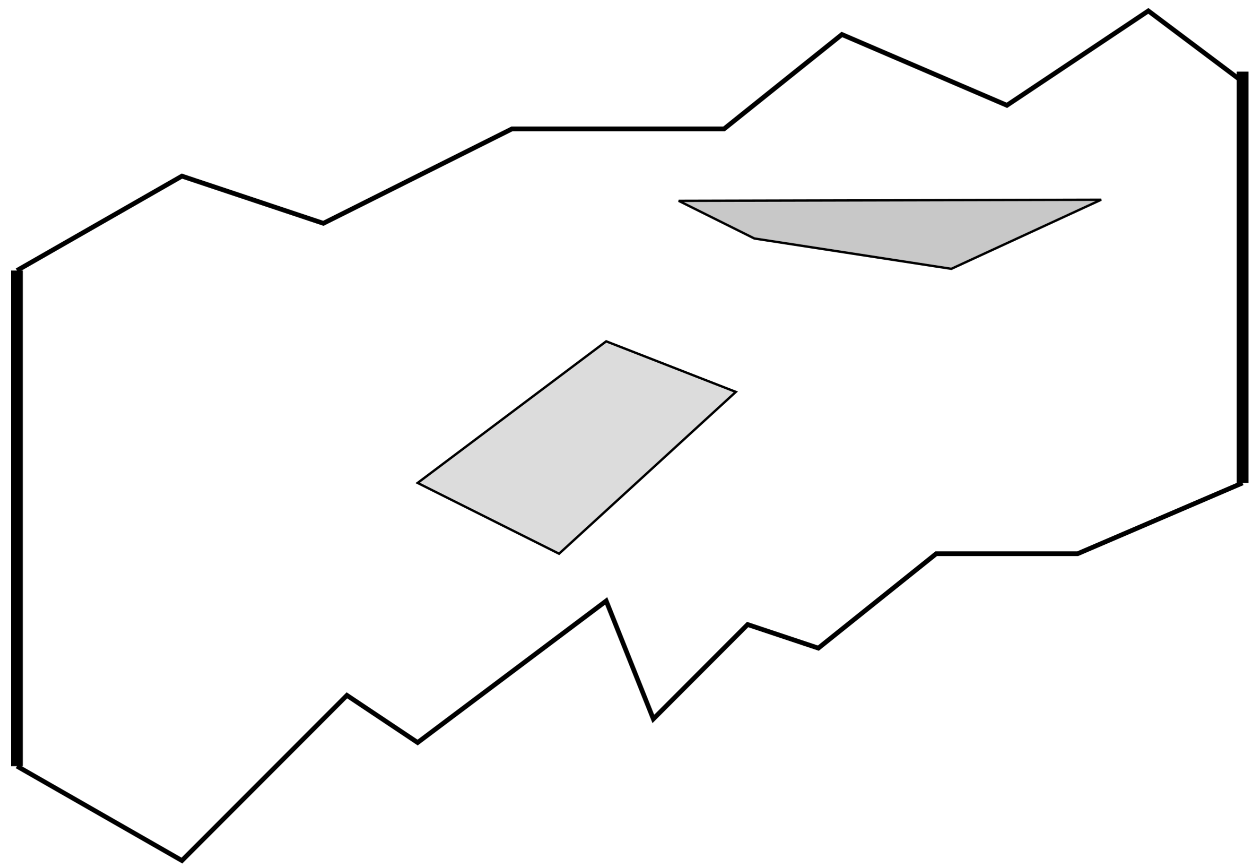
We want:

- Maximum number of non crossing thick paths from source to sink
- Paths should avoid all obstacles



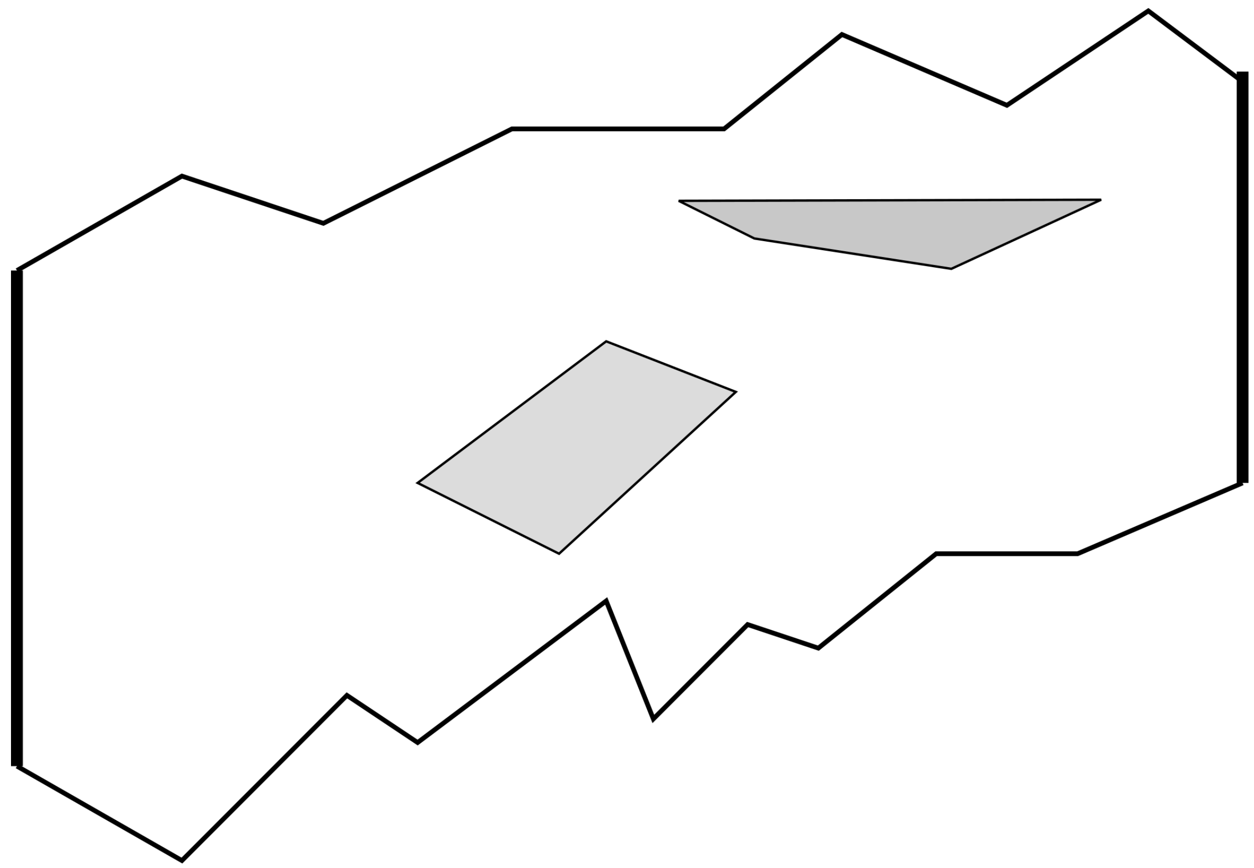
We want:

- Maximum number of non crossing thick paths from source to sink
- Paths should avoid all obstacles
- No path runs outside of polygonal domain



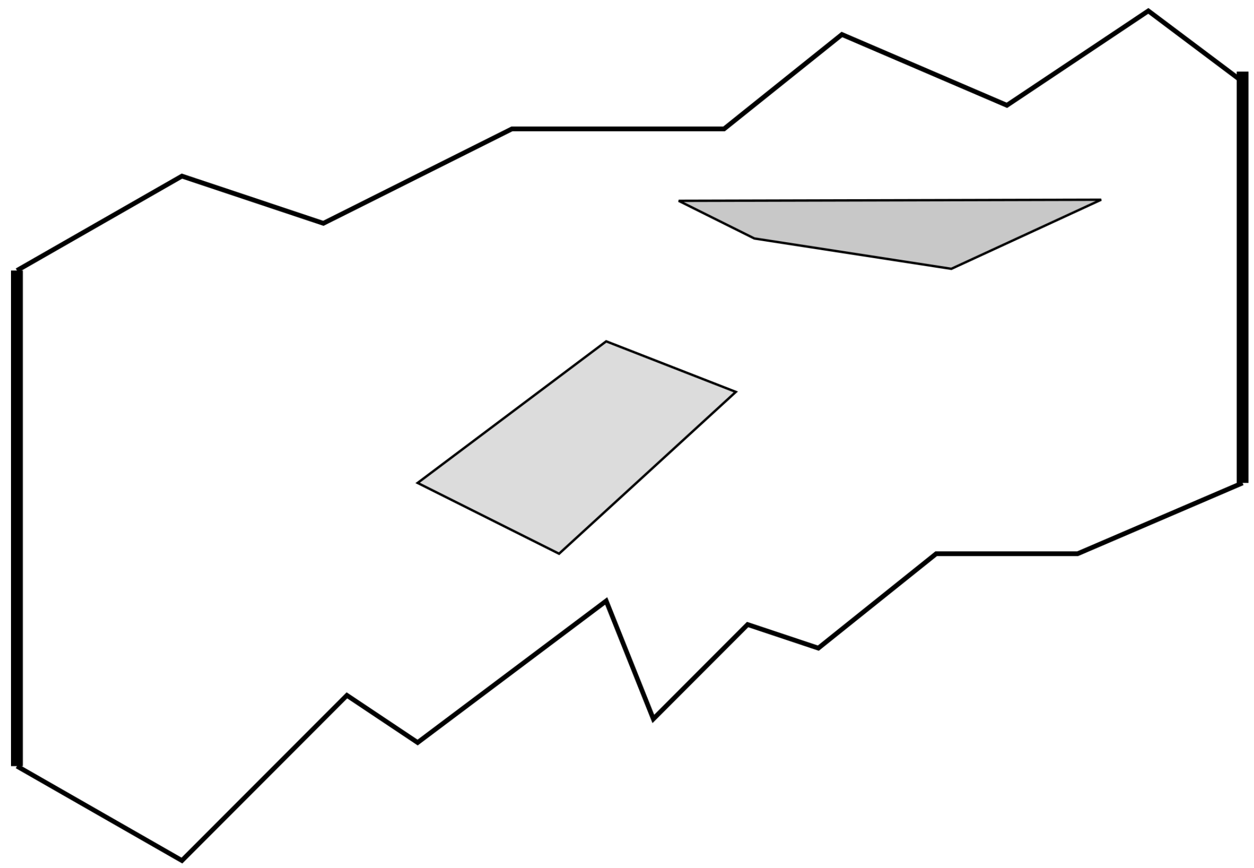
We want:

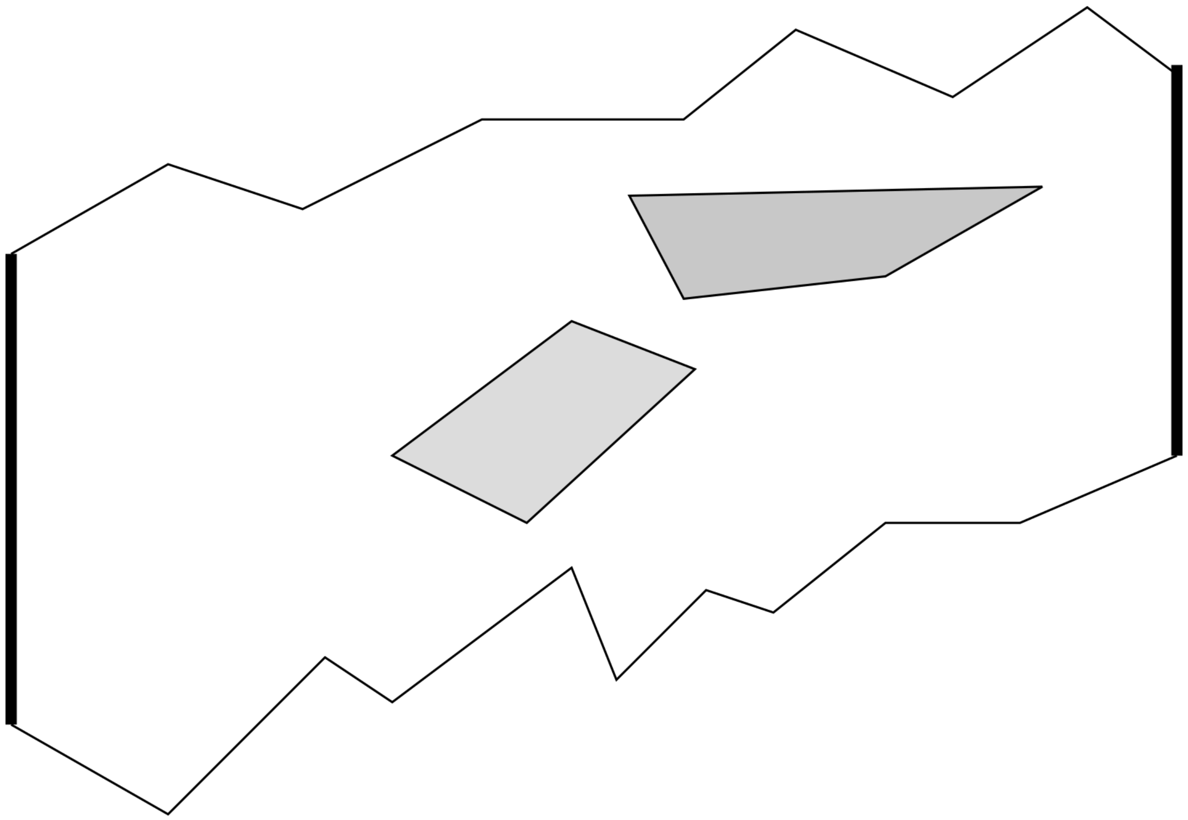
- Maximum number of non crossing thick paths from source to sink
- Paths should avoid all obstacles
- No path runs outside of polygonal domain
- non-crossing: $\mathbf{\Pi}_i \cap \mathbf{\Pi}_j = \varnothing$ (interiors disjoint, may share boundary)



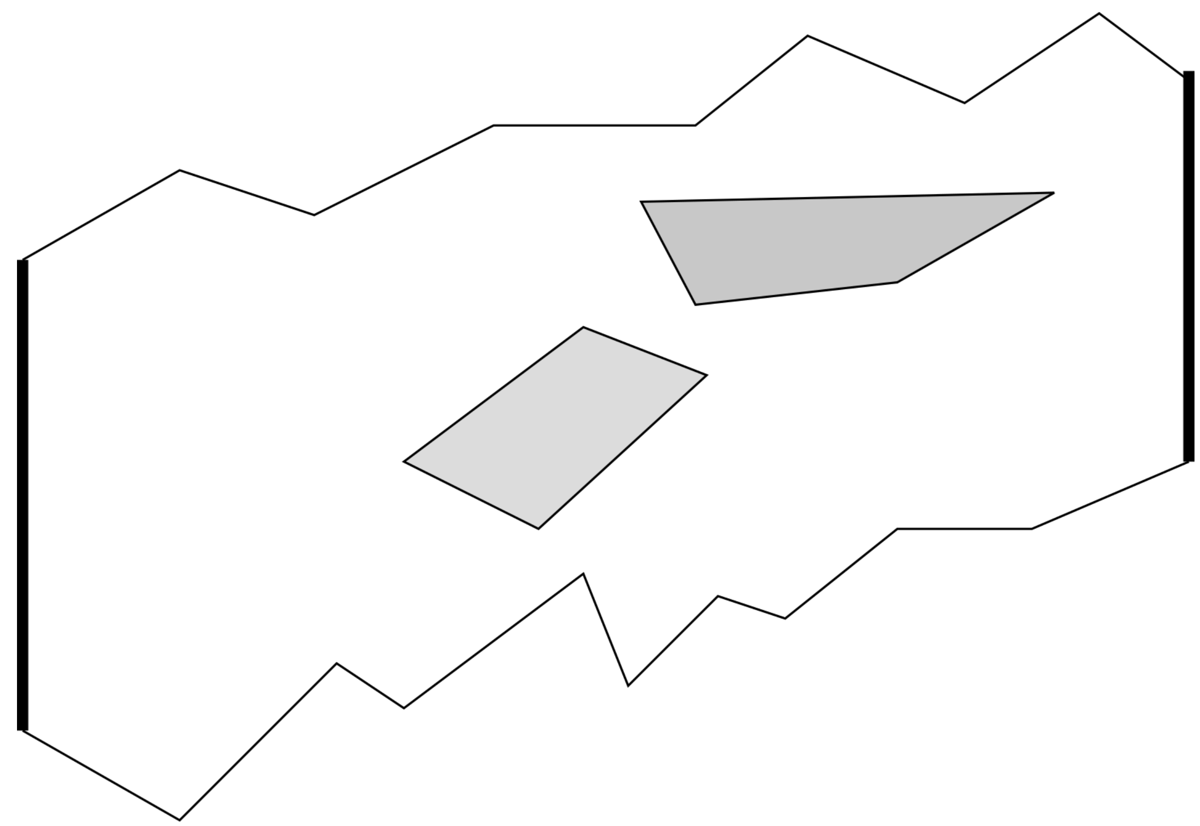
We want:

- Maximum number of non crossing thick paths from source to sink
- Paths should avoid all obstacles
- No path runs outside of polygonal domain
- non-crossing: $\mathbf{\Pi}_i \cap \mathbf{\Pi}_j = \varnothing$ (interiors disjoint, may share boundary)
- Need some more concepts (Ω perforated at the source and sinks and Riemann flaps glued to Ω, …)



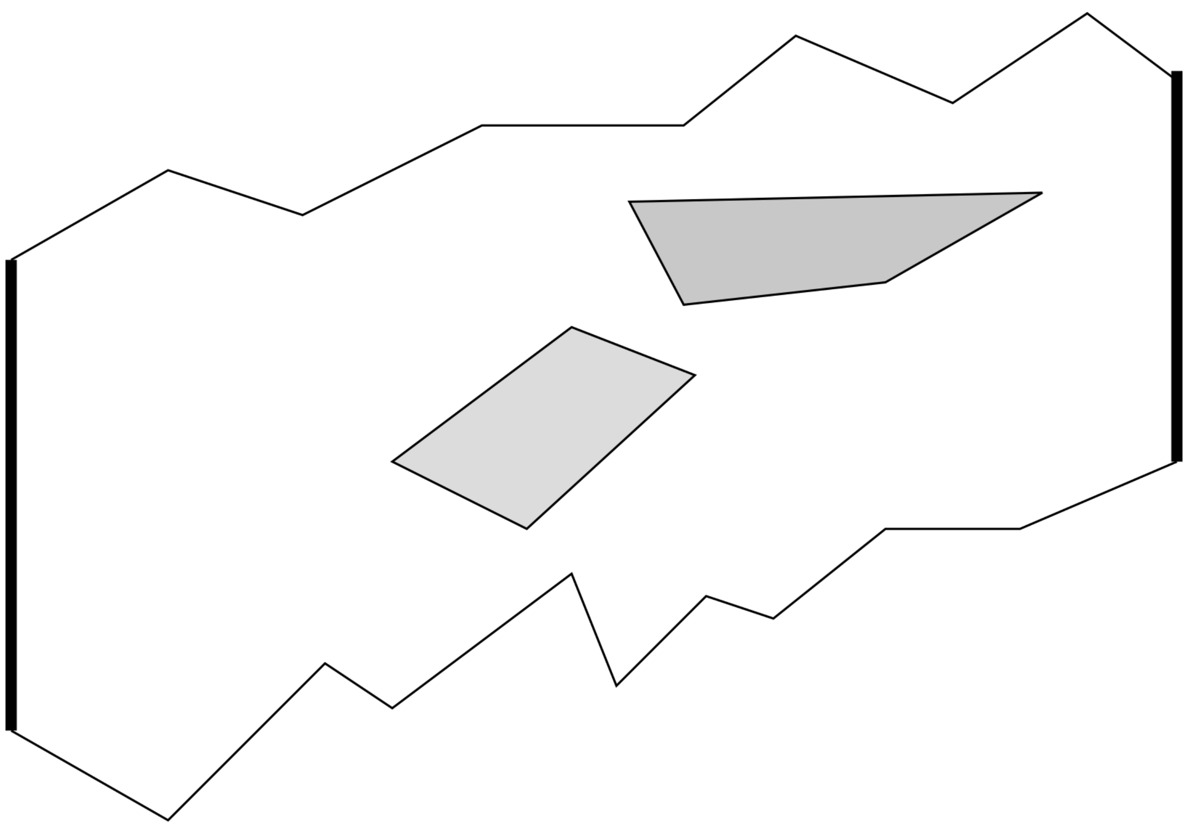


Algorithm by Arkin et al. (2010) to compute maximum number of thick paths:



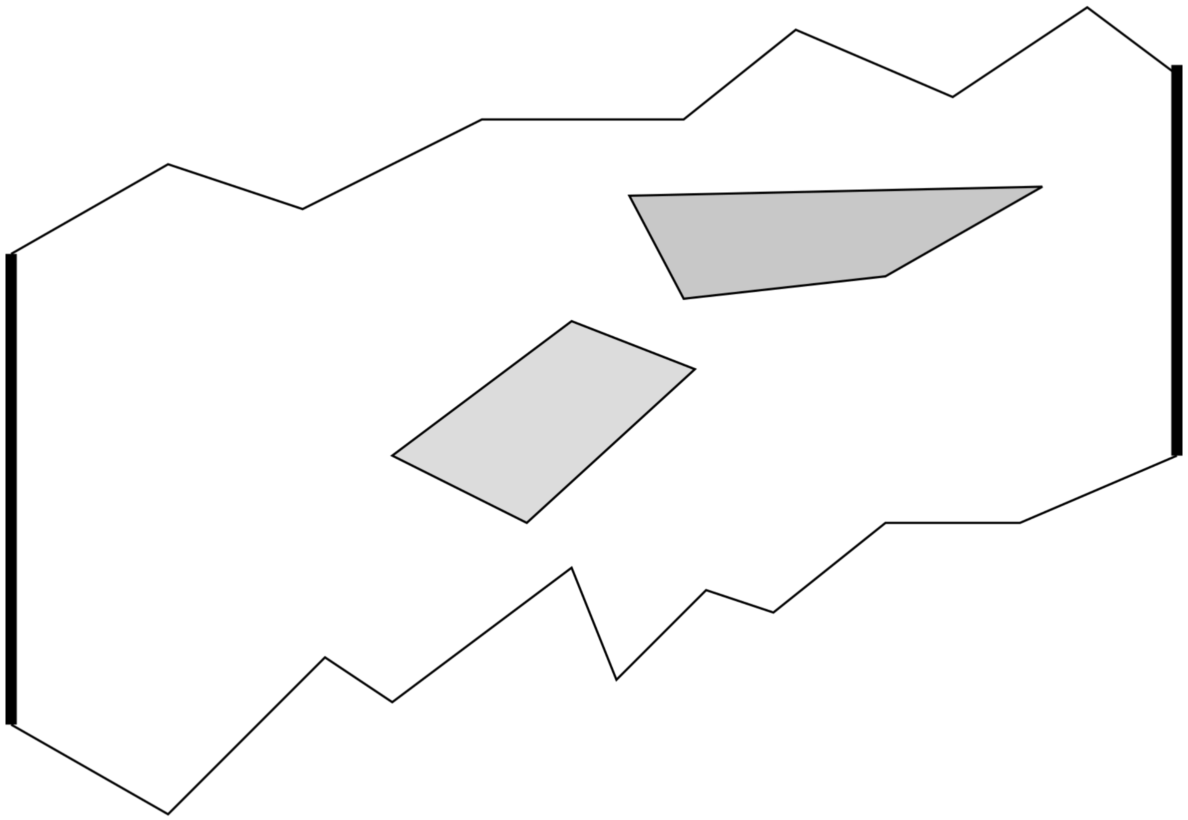
Algorithm by Arkin et al. (2010) to compute maximum number of thick paths:

- Grass-fire analogy



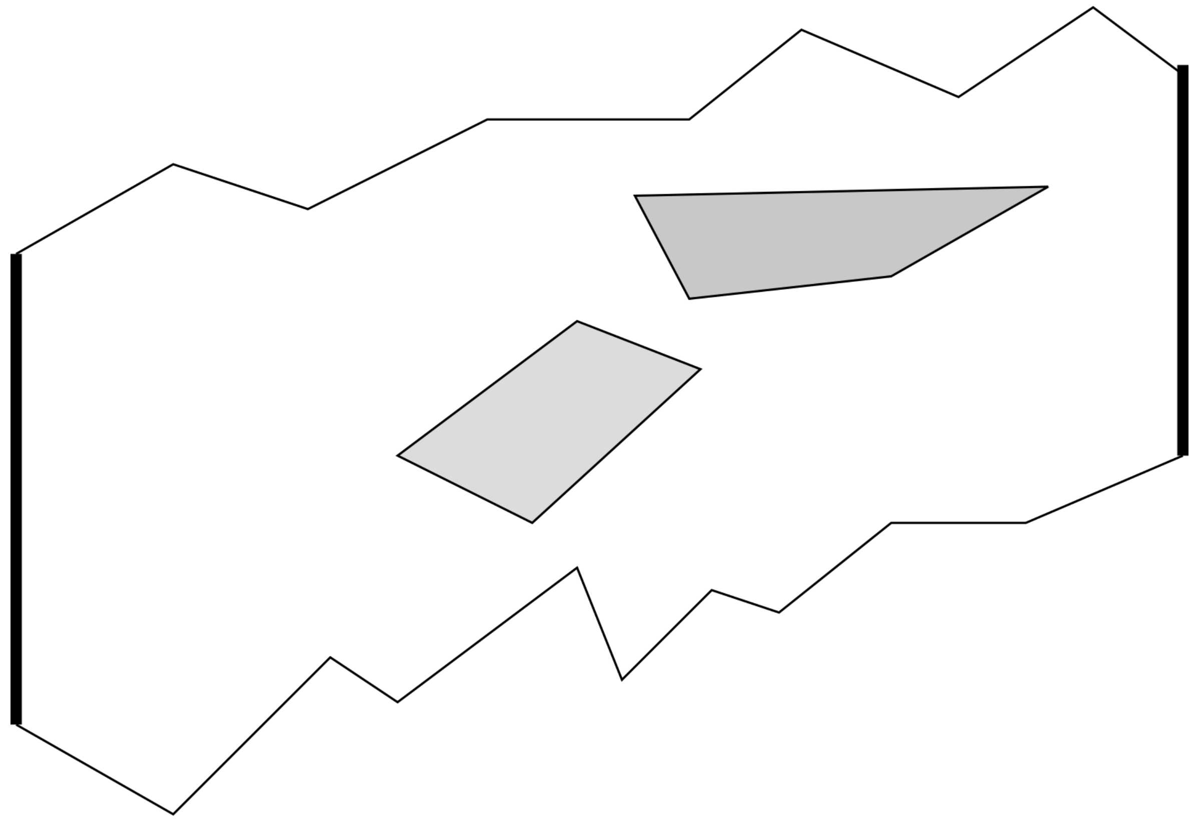
Algorithm by Arkin et al. (2010) to compute maximum number of thick paths:

- Grass-fire analogy
- Free space is grass over which fire travels with speed 1



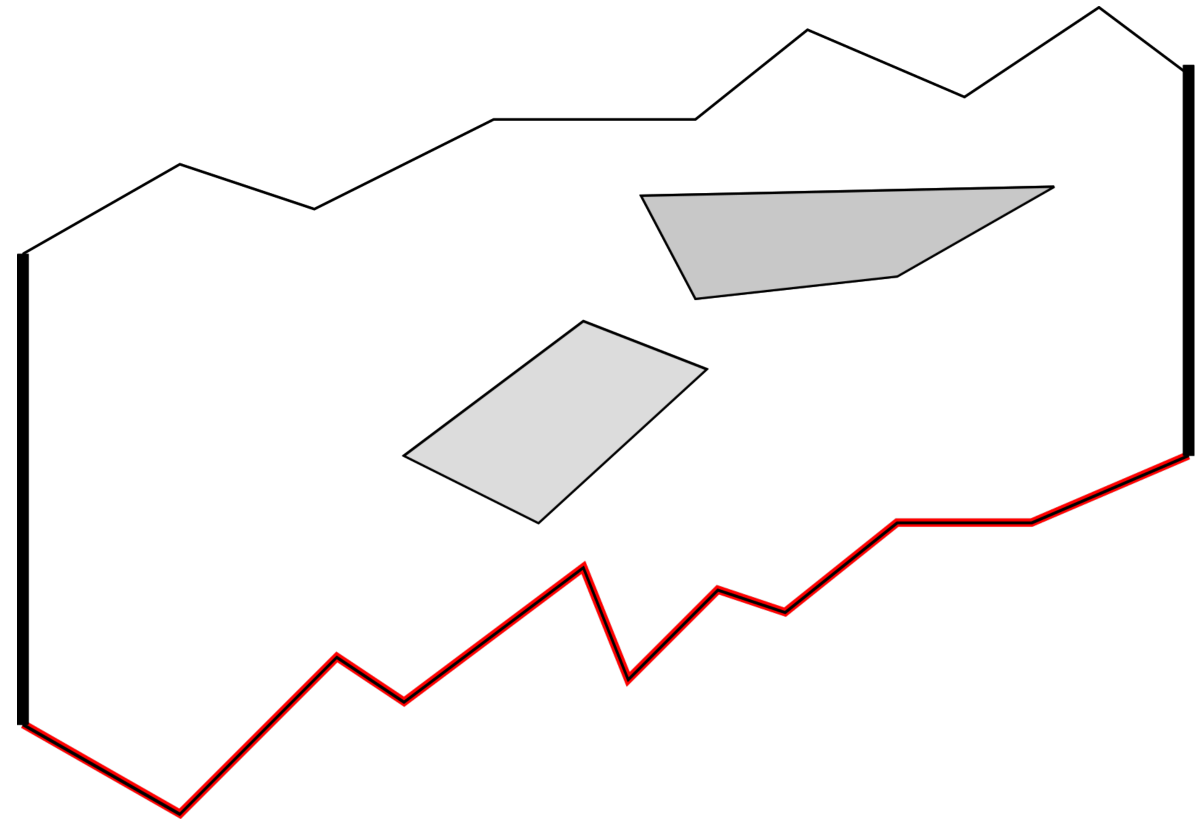
Algorithm by Arkin et al. (2010) to compute maximum number of thick paths:

- Grass-fire analogy
- Free space is grass over which fire travels with speed 1
- Holes are highly flammable: once ignited, fire moves through them with infinite speed



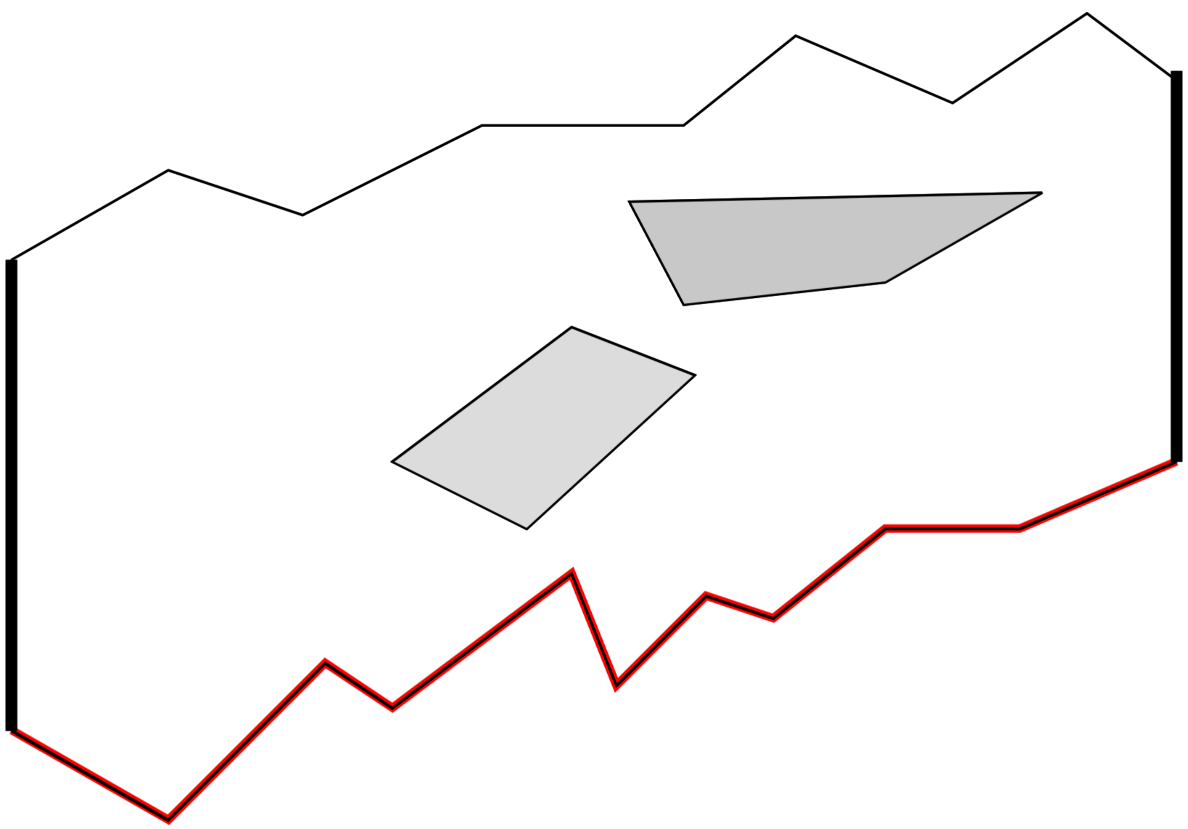
Algorithm by Arkin et al. (2010) to compute maximum number of thick paths:

- Grass-fire analogy
- Free space is grass over which fire travels with speed 1
- Holes are highly flammable: once ignited, fire moves through them with infinite speed
- We start setting the bottom on fire.



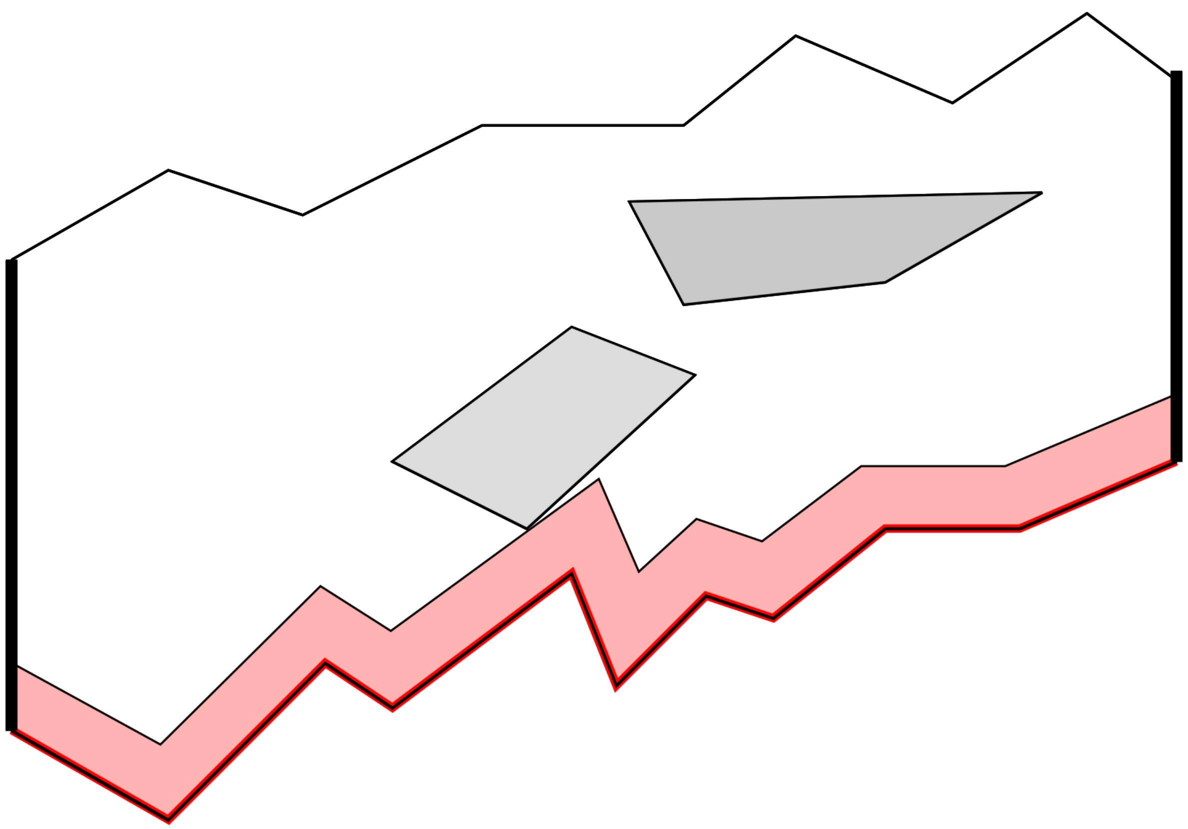
Algorithm by Arkin et al. (2010) to compute maximum number of thick paths:

- Grass-fire analogy
- Free space is grass over which fire travels with speed 1
- Holes are highly flammable: once ignited, fire moves through them with infinite speed
- We start setting the bottom on fire.
- Wavefront at time τ: boundary of burnt grass by time τ



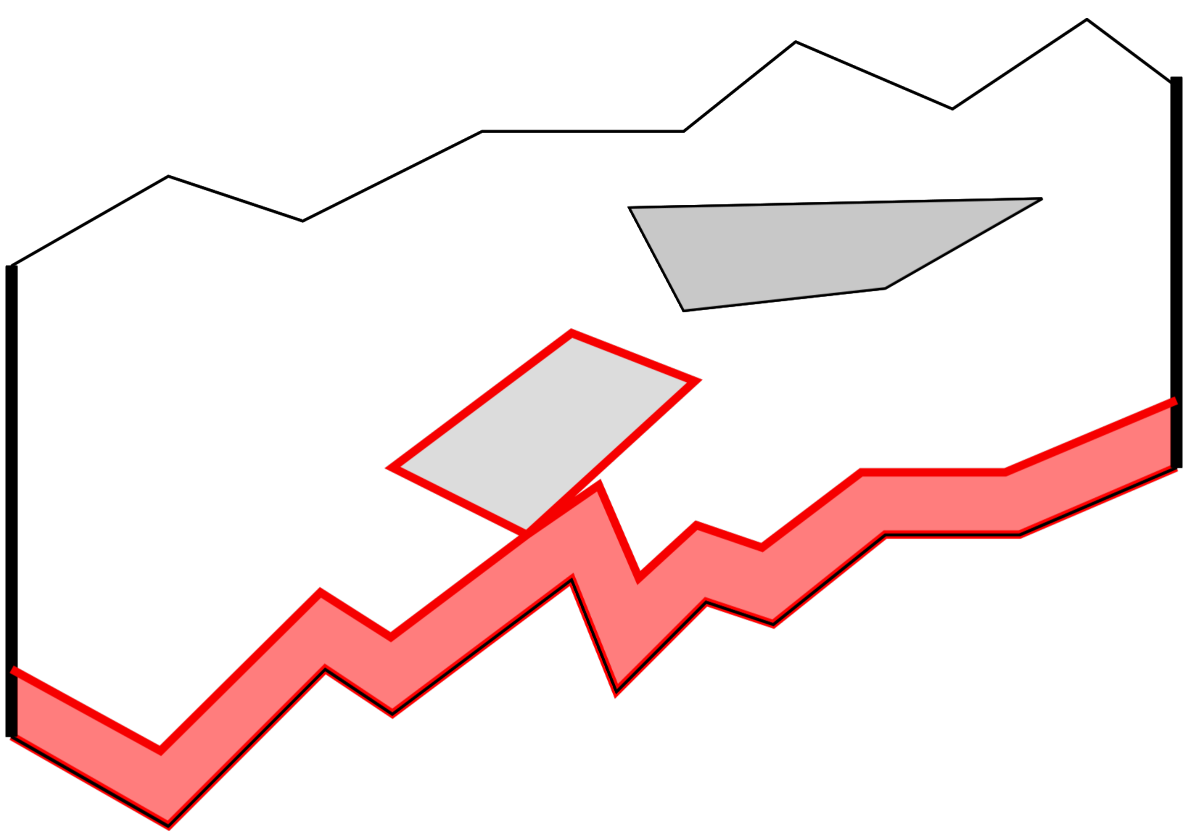
Algorithm by Arkin et al. (2010) to compute maximum number of thick paths:

- Grass-fire analogy
- Free space is grass over which fire travels with speed 1
- Holes are highly flammable: once ignited, fire moves through them with infinite speed
- We start setting the bottom on fire.
- Wavefront at time τ: boundary of burnt grass by time τ
- Whenever fire burns 2 time units w/o hitting hole —> we can route a thick path through the burnt grass



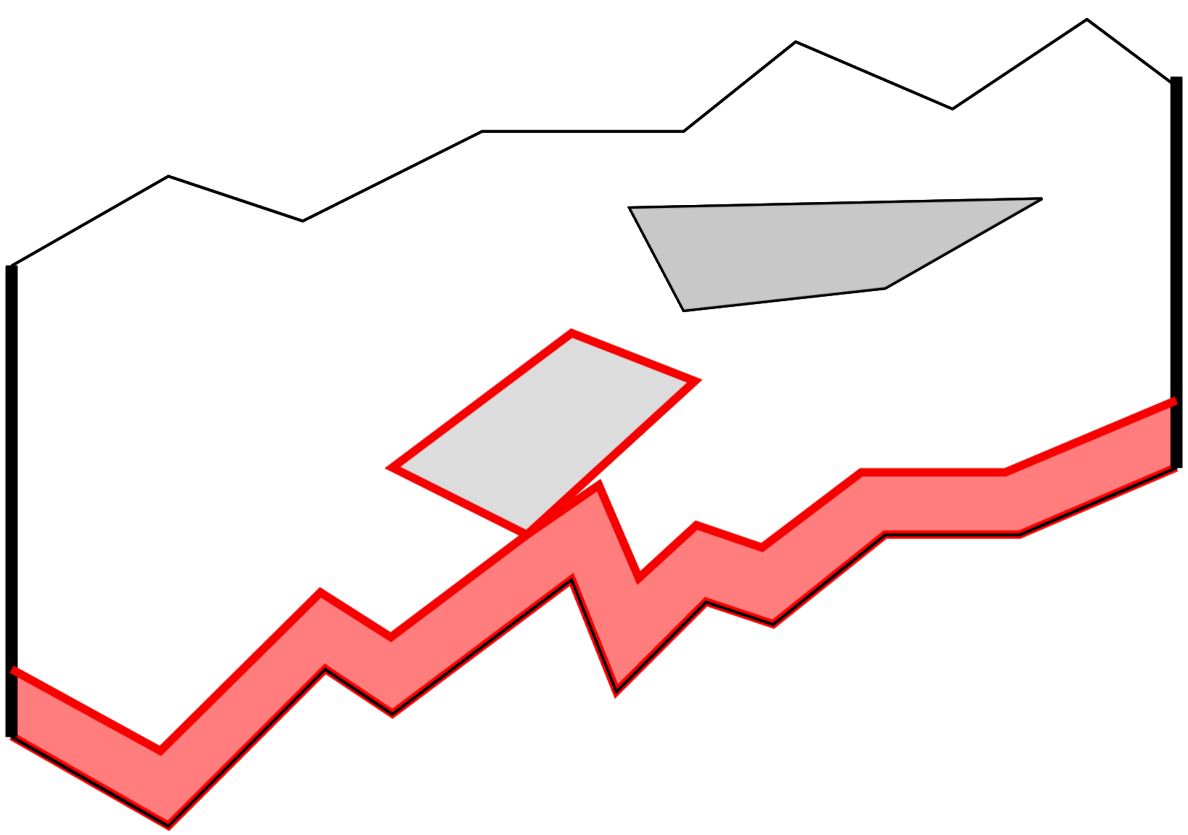
Algorithm by Arkin et al. (2010) to compute maximum number of thick paths:

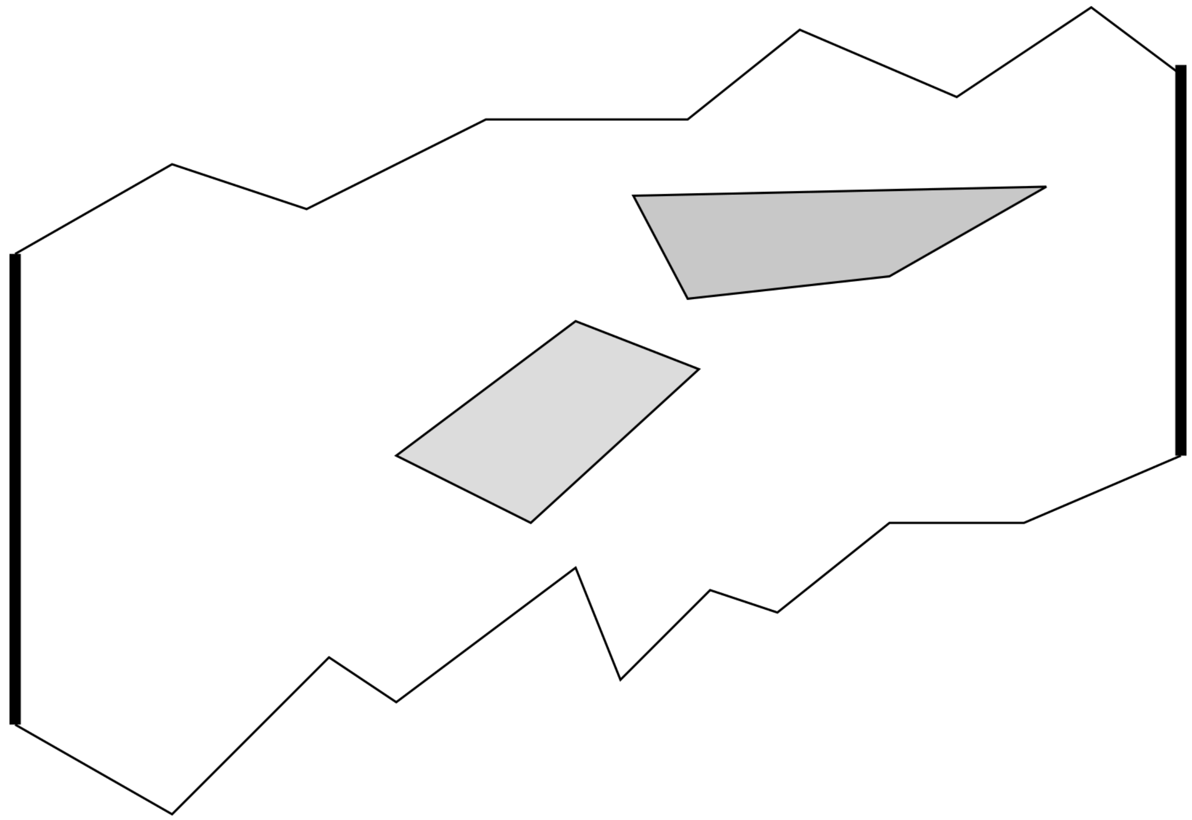
- Grass-fire analogy
- Free space is grass over which fire travels with speed 1
- Holes are highly flammable: once ignited, fire moves through them with infinite speed
- We start setting the bottom on fire.
- Wavefront at time τ: boundary of burnt grass by time τ
- Whenever fire burns 2 time units w/o hitting hole —> we can route a thick path through the burnt grass
- Once path has been routed: wavefront is new bottom, and we start over



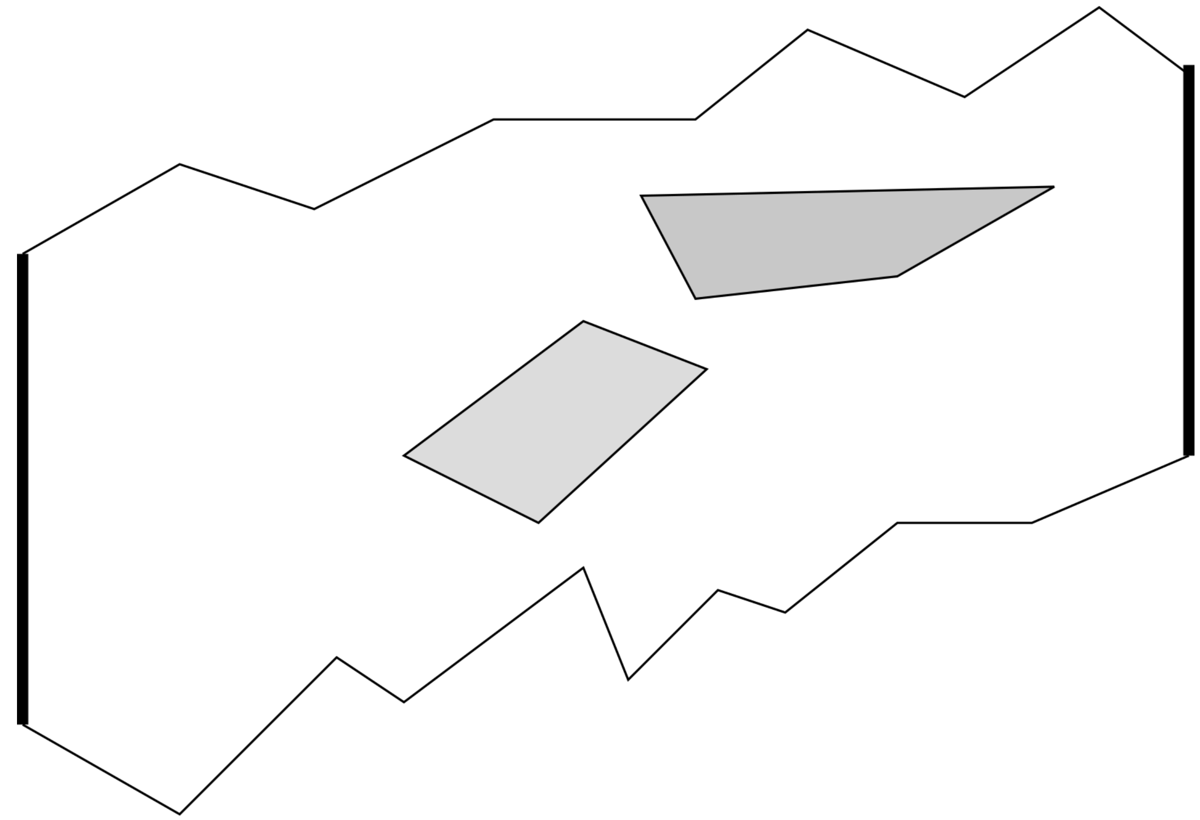
Algorithm by Arkin et al. (2010) to compute maximum number of thick paths:

- Grass-fire analogy
- Free space is grass over which fire travels with speed 1
- Holes are highly flammable: once ignited, fire moves through them with infinite speed
- We start setting the bottom on fire.
- Wavefront at time τ: boundary of burnt grass by time τ
- Whenever fire burns 2 time units w/o hitting hole —> we can route a thick path through the burnt grass
- Once path has been routed: wavefront is new bottom, and we start over
- Some additional tweaks when we hit a hole after $\tau<2$



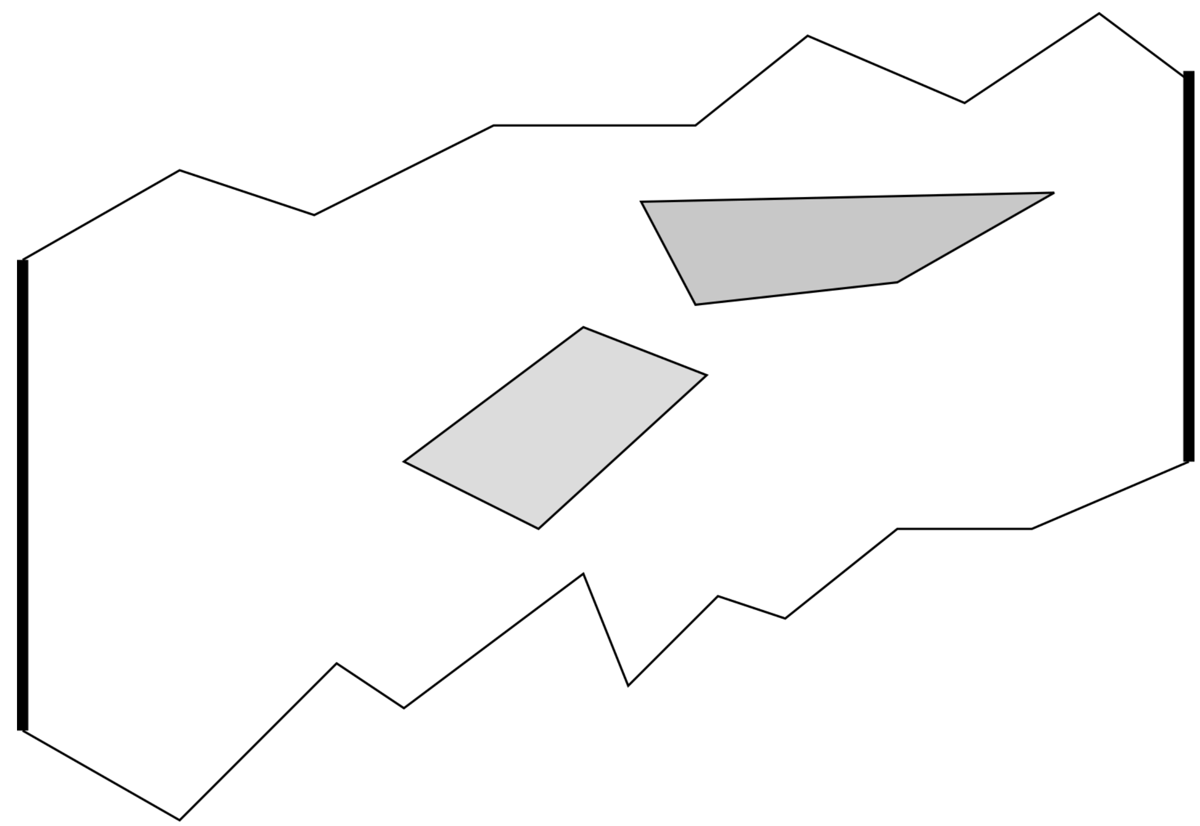


Polishchuk (2007) extended this to x-monotone paths:



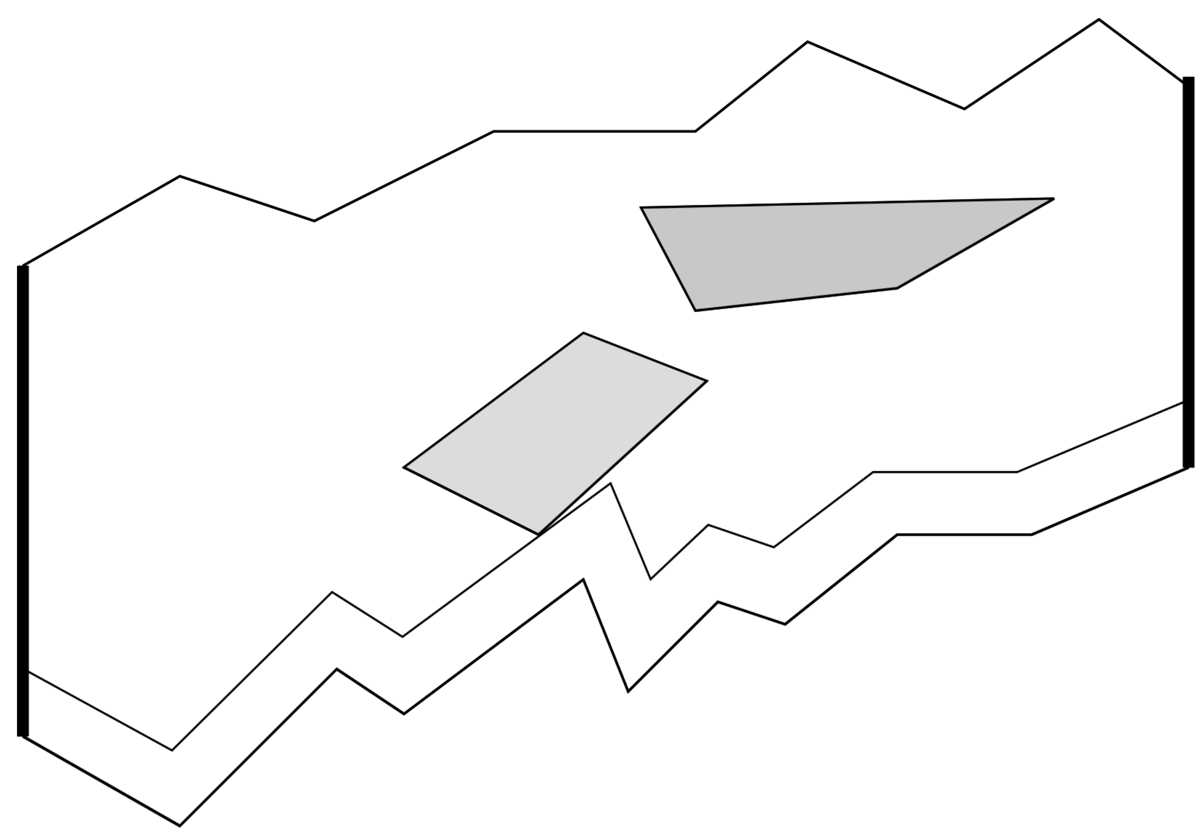
Polishchuk (2007) extended this to x-monotone paths:

- Need a monotone boundary, if not, add “waterfalls”



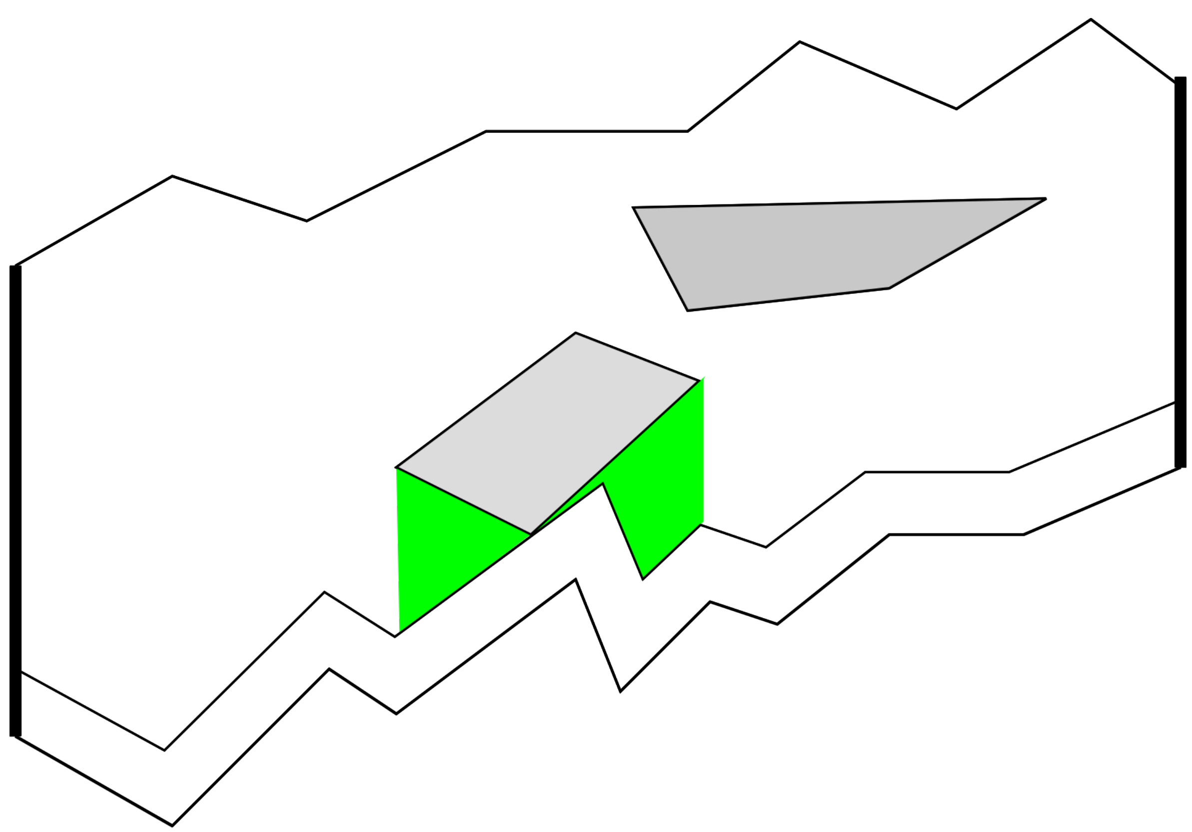
Polishchuk (2007) extended this to x-monotone paths:

- Need a monotone boundary, if not, add “waterfalls”
- Again, let fire burn



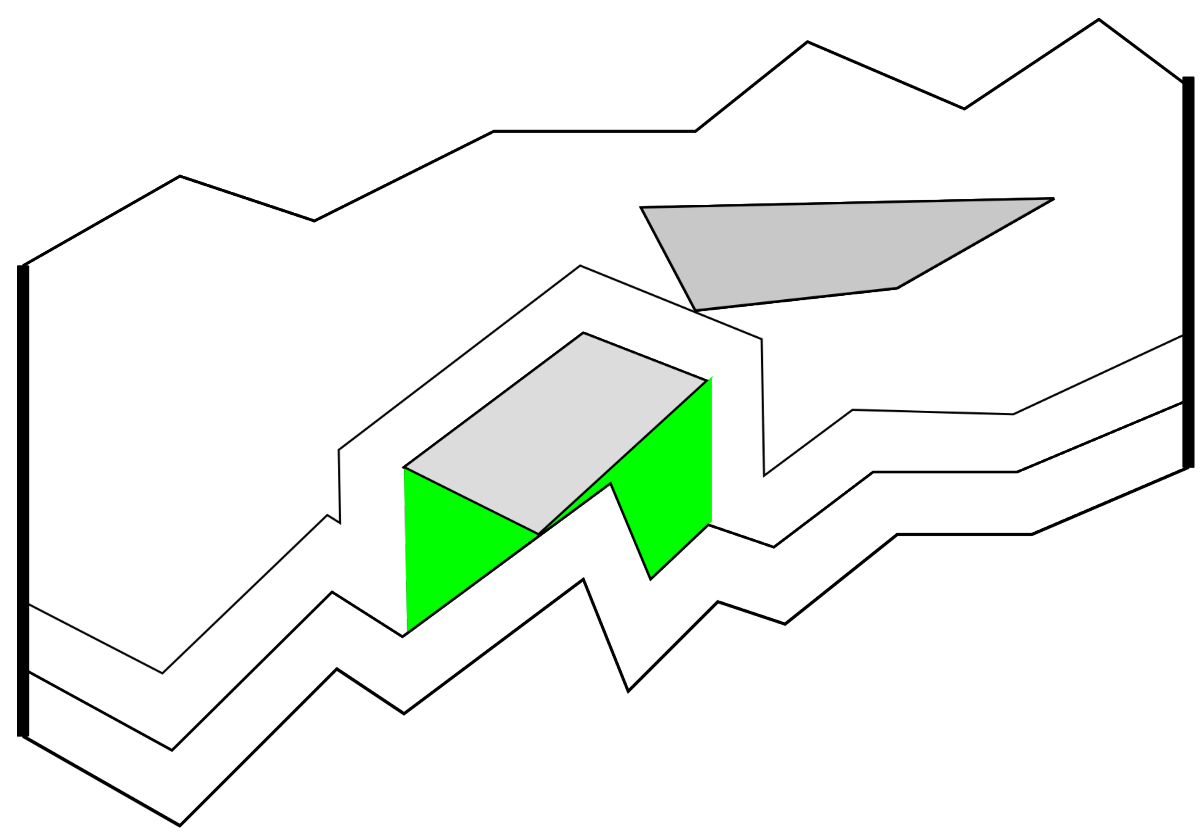
Polishchuk (2007) extended this to x-monotone paths:

- Need a monotone boundary, if not, add “waterfalls”
- Again, let fire burn
- If we hit a hole in the process, outer-monotonize holes using waterfalls



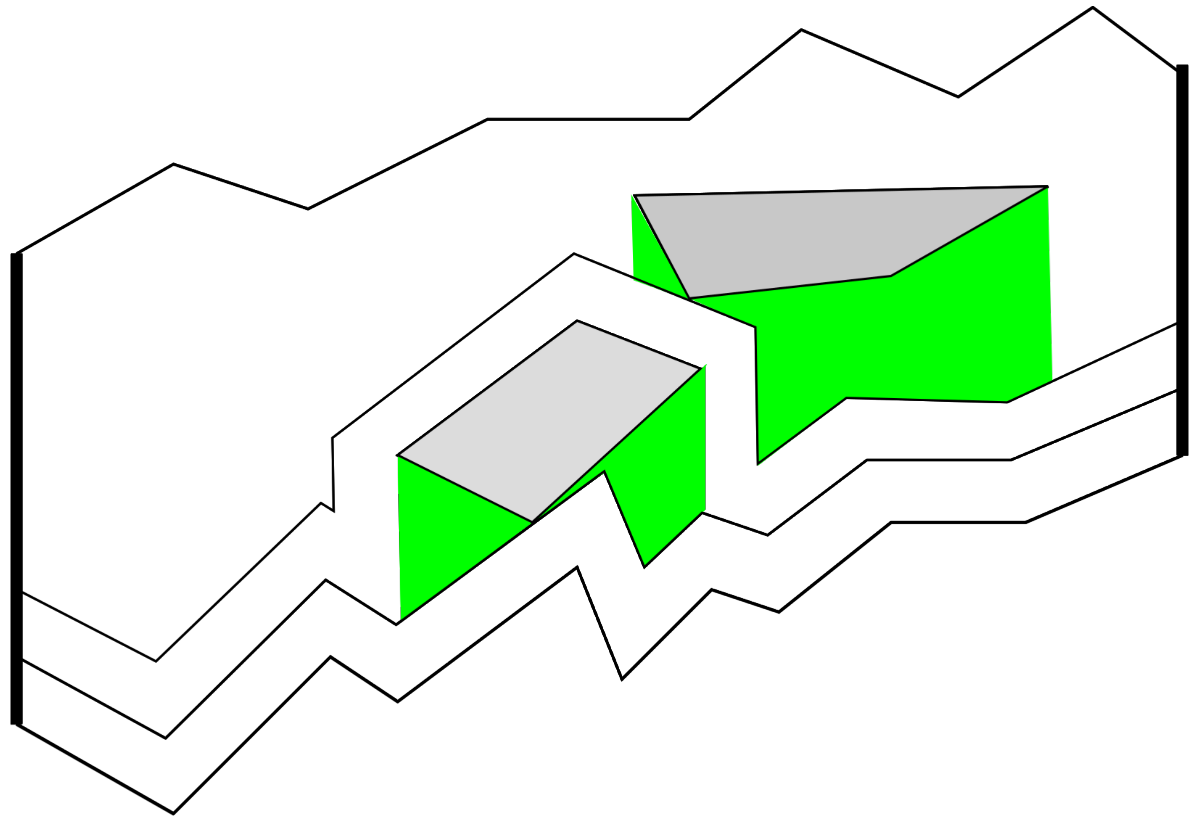
Polishchuk (2007) extended this to x-monotone paths:

- Need a monotone boundary, if not, add "waterfalls"
- Again, let fire burn
- If we hit a hole in the process, outer-monotonize holes using waterfalls



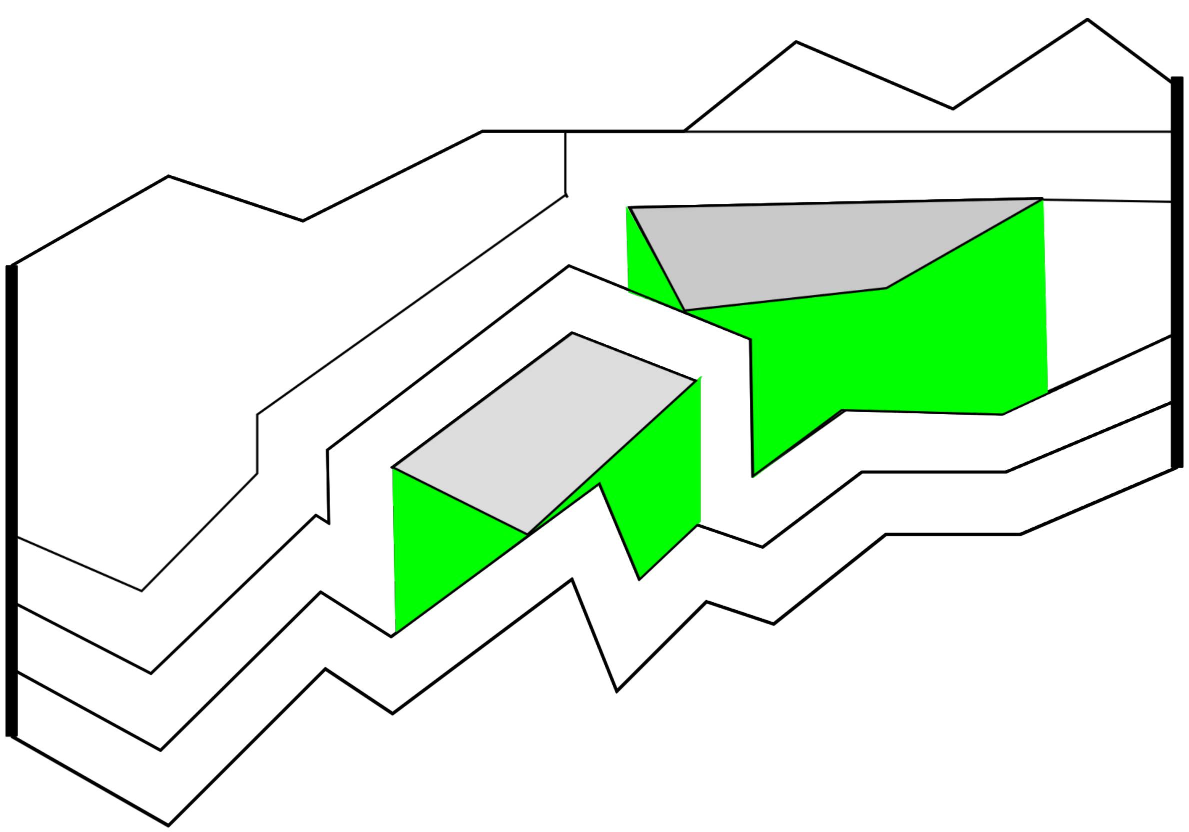
Polishchuk (2007) extended this to x-monotone paths:

- Need a monotone boundary, if not, add “waterfalls”
- Again, let fire burn
- If we hit a hole in the process, outer-monotonize holes using waterfalls



Polishchuk (2007) extended this to x-monotone paths:

- Need a monotone boundary, if not, add “waterfalls”
- Again, let fire burn
- If we hit a hole in the process, outer-monotonize holes using waterfalls



Thick Paths with Limited Slope

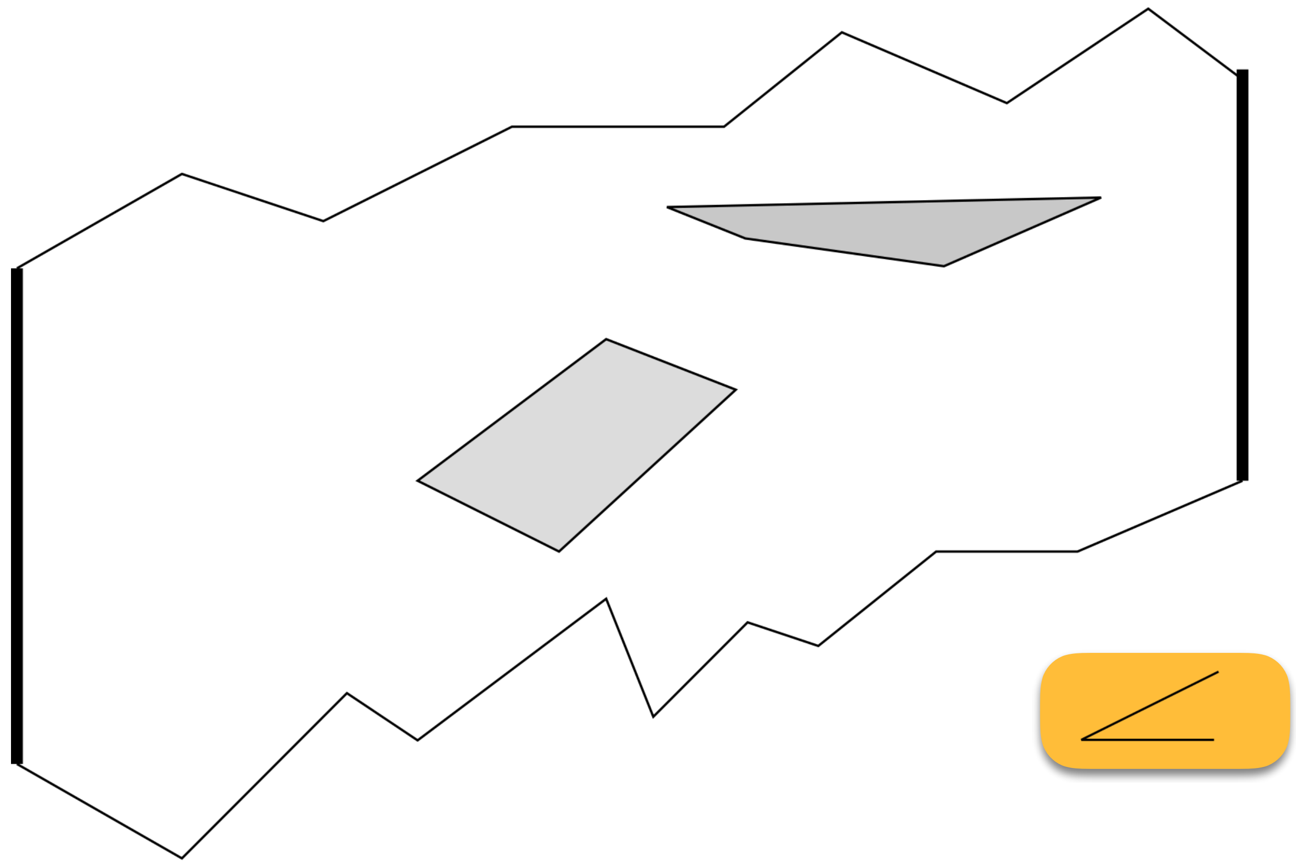
We want:

We want:

- Maximum number of non crossing thick paths from source to sink

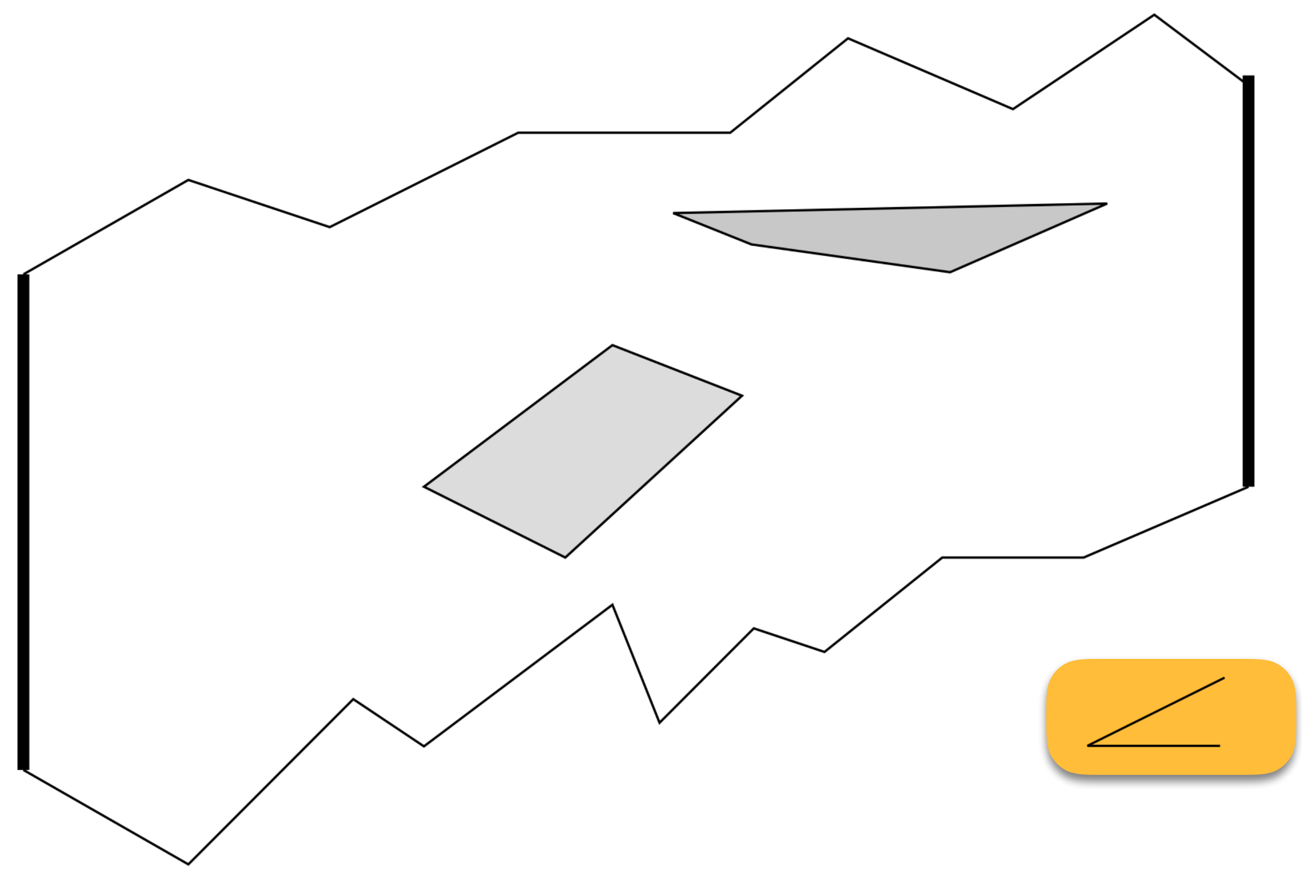
We want:

- Maximum number of non crossing thick paths from source to sink
- Slope should be within a given cone C



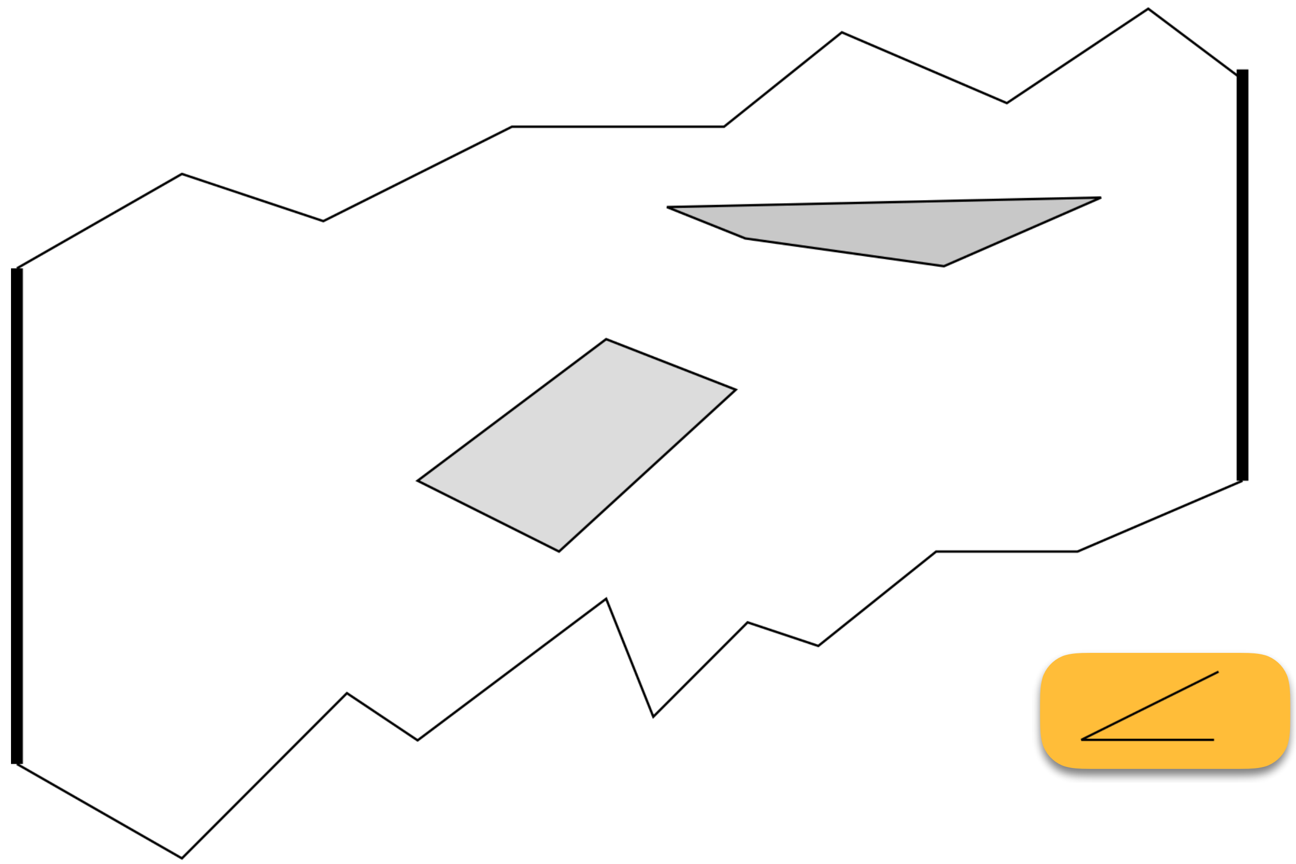
We want:

- Maximum number of non crossing thick paths from source to sink
- Slope should be within a given cone C
    - X-monotone



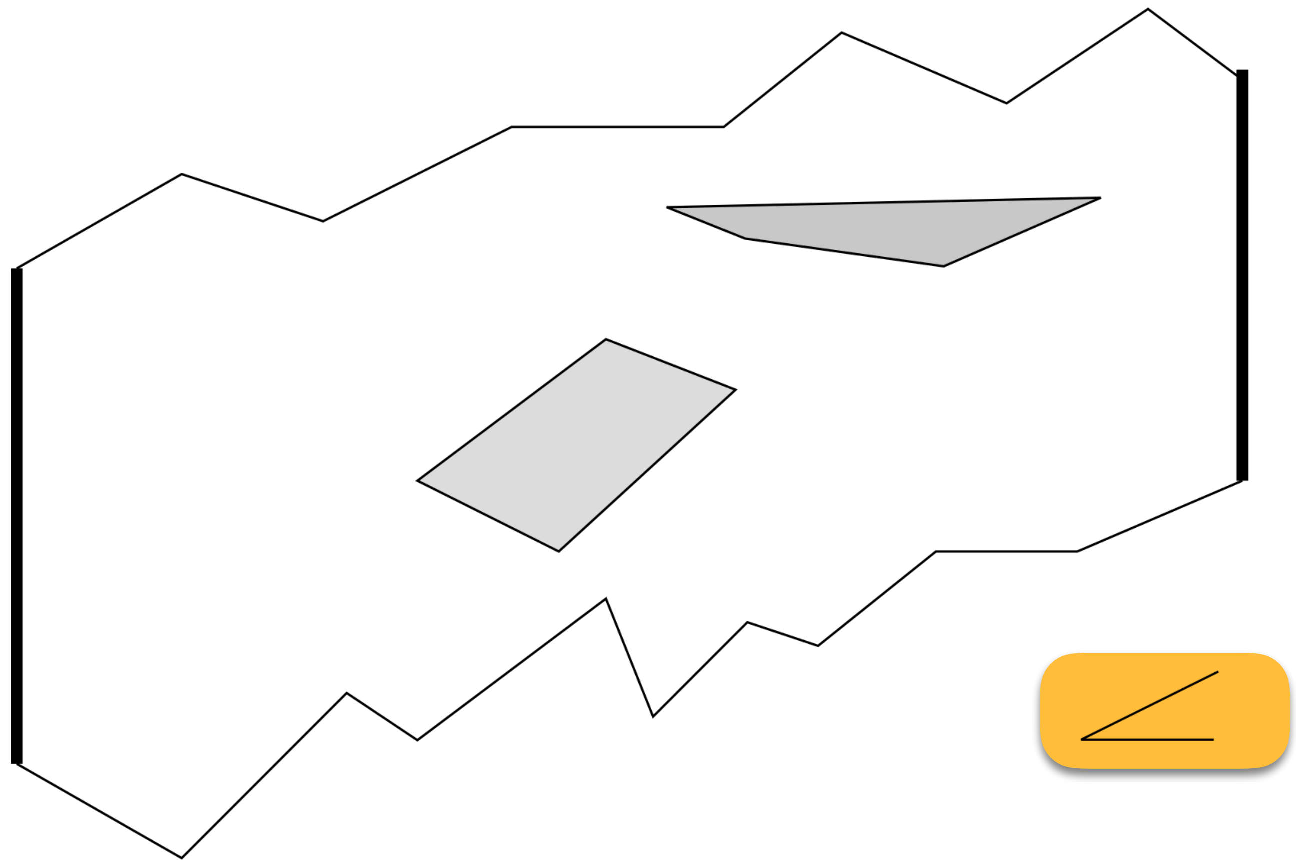
We want:

- Maximum number of non crossing thick paths from source to sink
- Slope should be within a given cone C
  - X-monotone
  - Limited speed ⇨ Limited slope



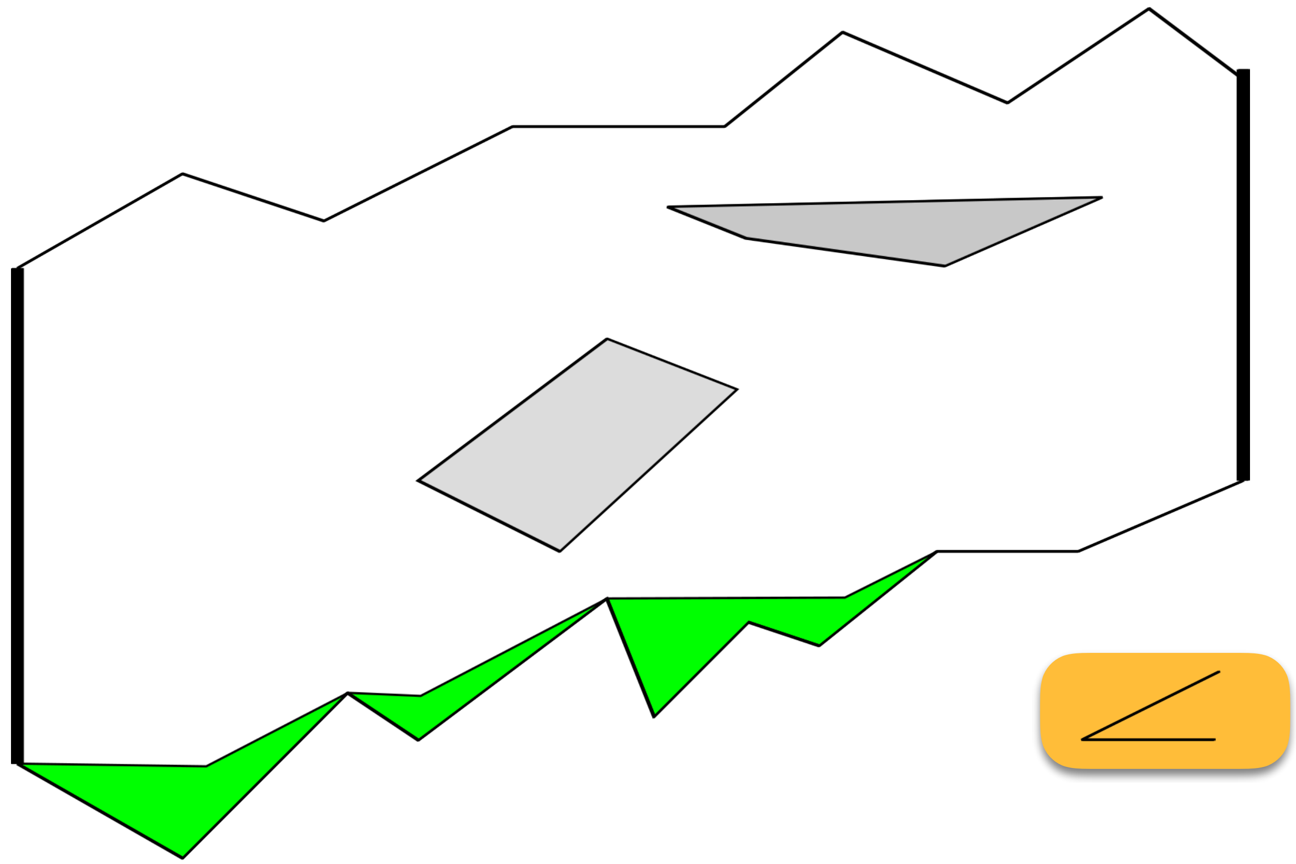
We want:

- Maximum number of non crossing thick paths from source to sink
- Slope should be within a given cone C
  - X-monotone
  - Limited speed ⇨ Limited slope
- We showed how to adapt the waterfall construction to compute the maximum number of thick non-crossing paths with a given slope range (≙C-respecting)



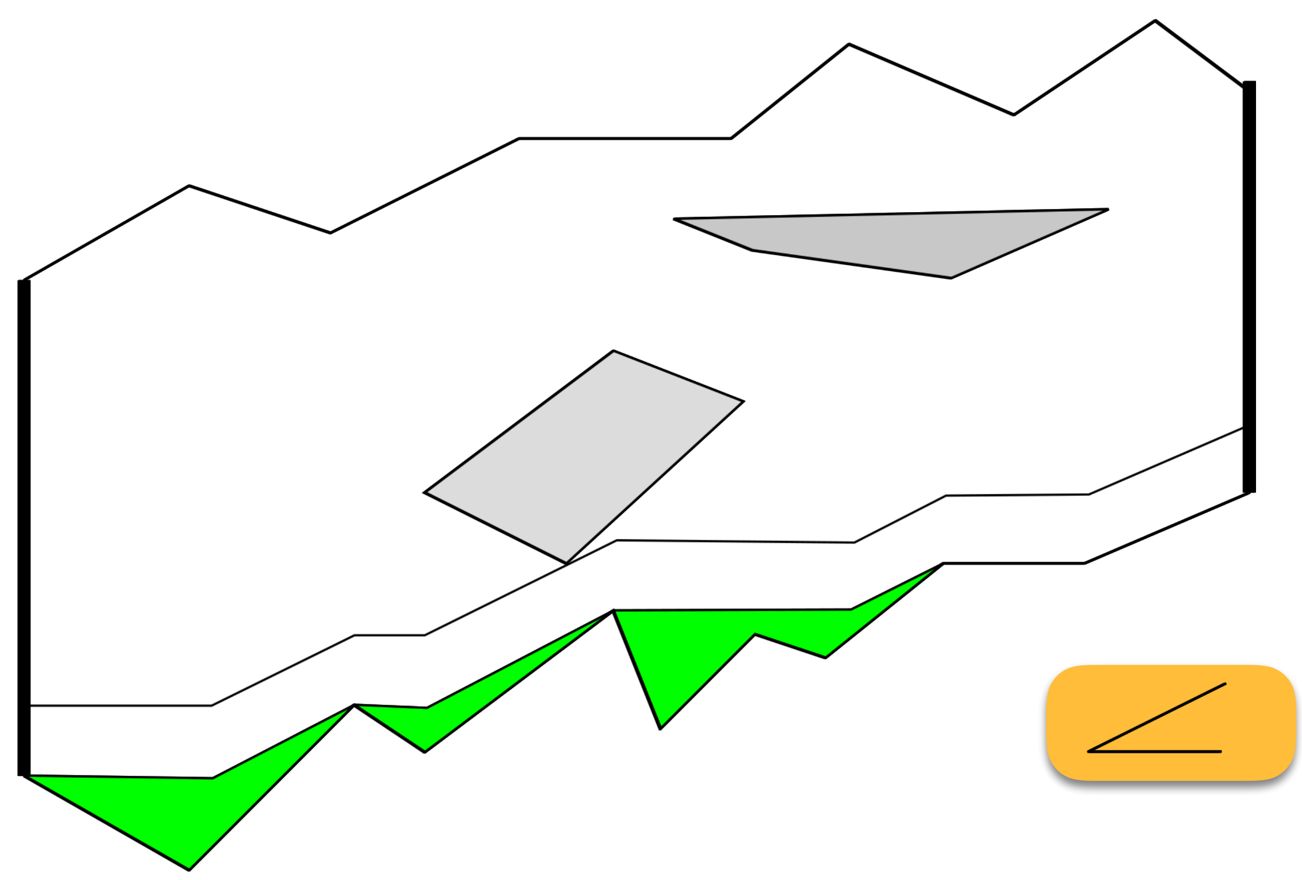
We want:

- Maximum number of non crossing thick paths from source to sink
- Slope should be within a given cone C
    - X-monotone
    - Limited speed ⇨ Limited slope
- We showed how to adapt the waterfall construction to compute the maximum number of thick non-crossing paths with a given slope range (≙C-respecting)



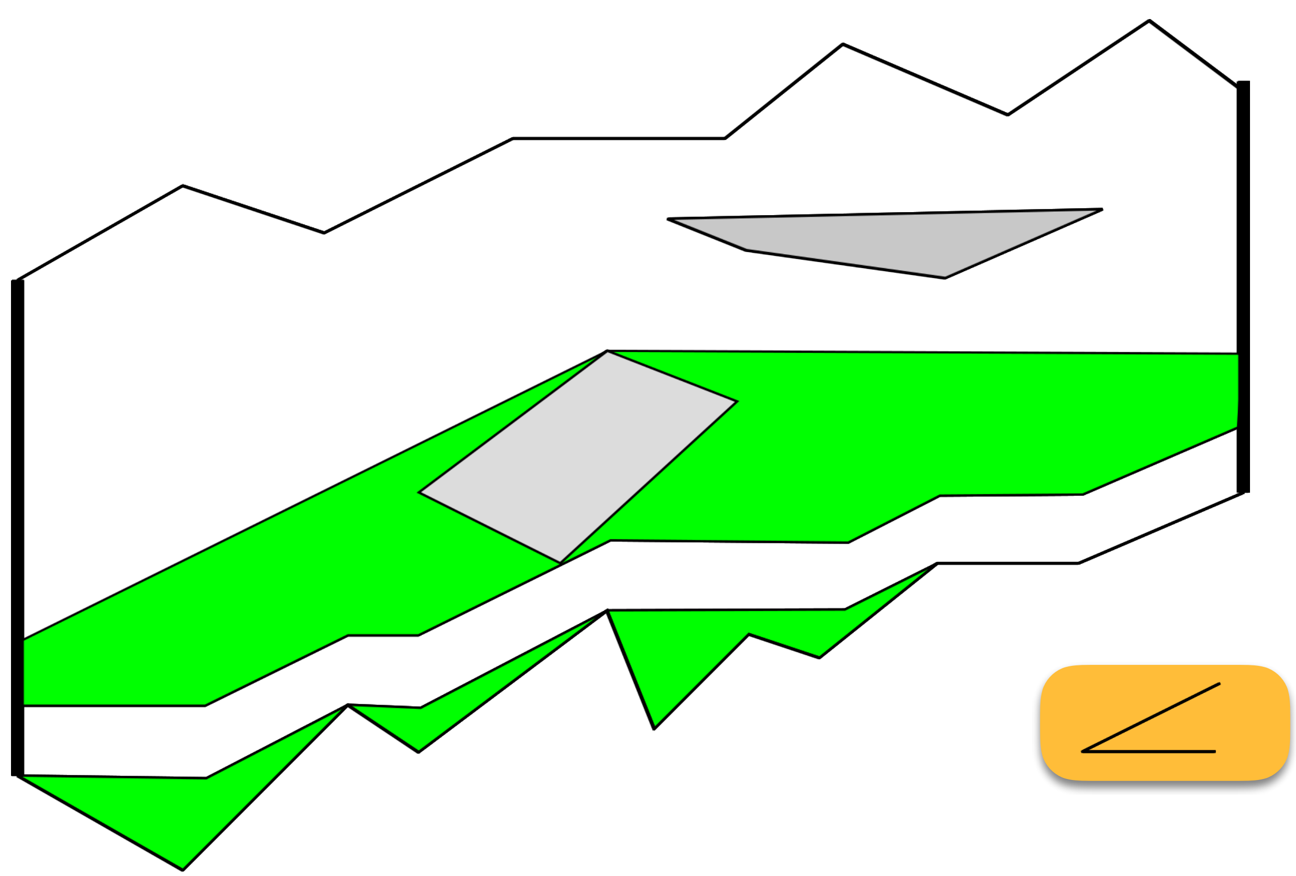
We want:

- Maximum number of non crossing thick paths from source to sink
- Slope should be within a given cone C
  - X-monotone
  - Limited speed ⇨ Limited slope
- We showed how to adapt the waterfall construction to compute the maximum number of thick non-crossing paths with a given slope range (≙C-respecting)



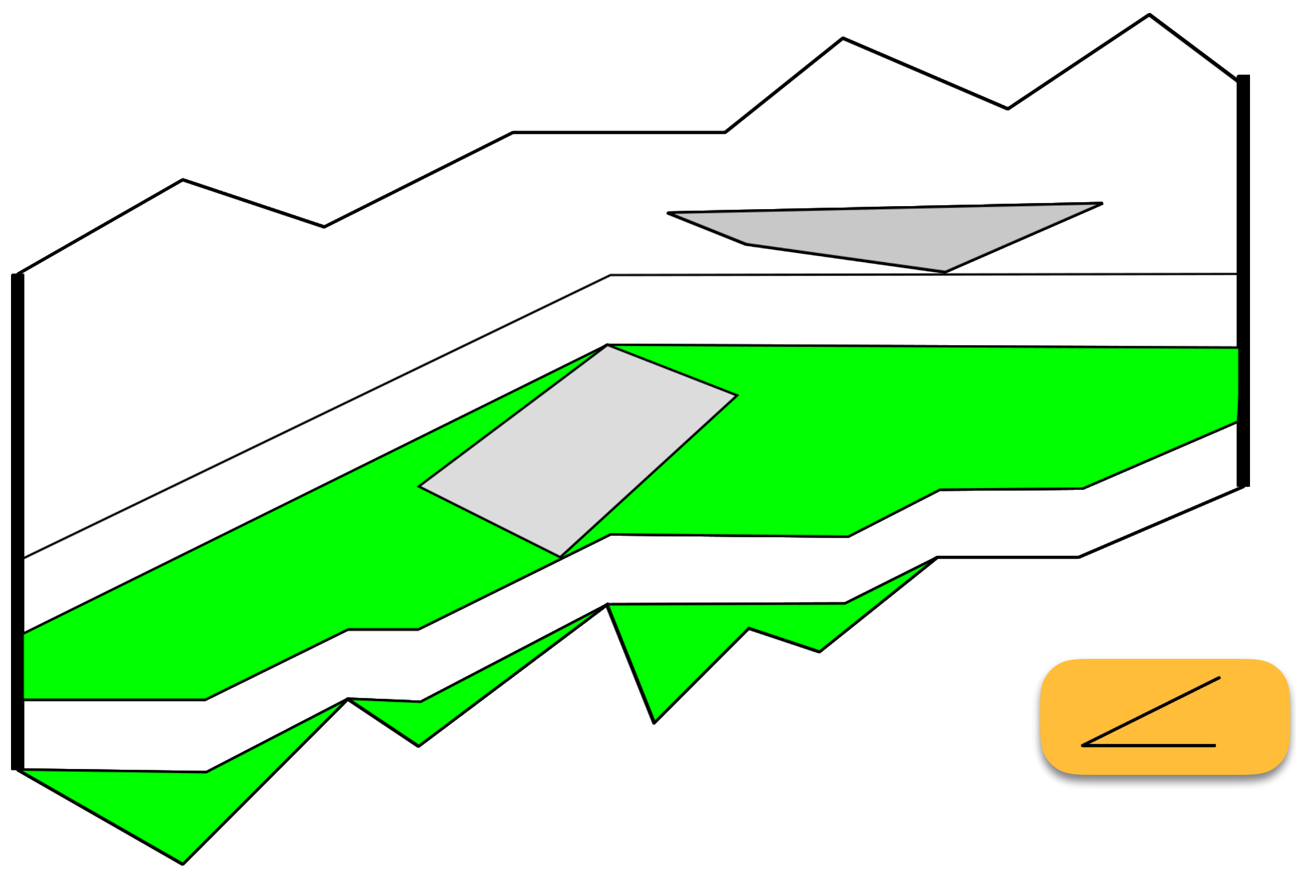
We want:

- Maximum number of non crossing thick paths from source to sink
- Slope should be within a given cone C
    - X-monotone
    - Limited speed ⇨ Limited slope
- We showed how to adapt the waterfall construction to compute the maximum number of thick non-crossing paths with a given slope range (≙C-respecting)



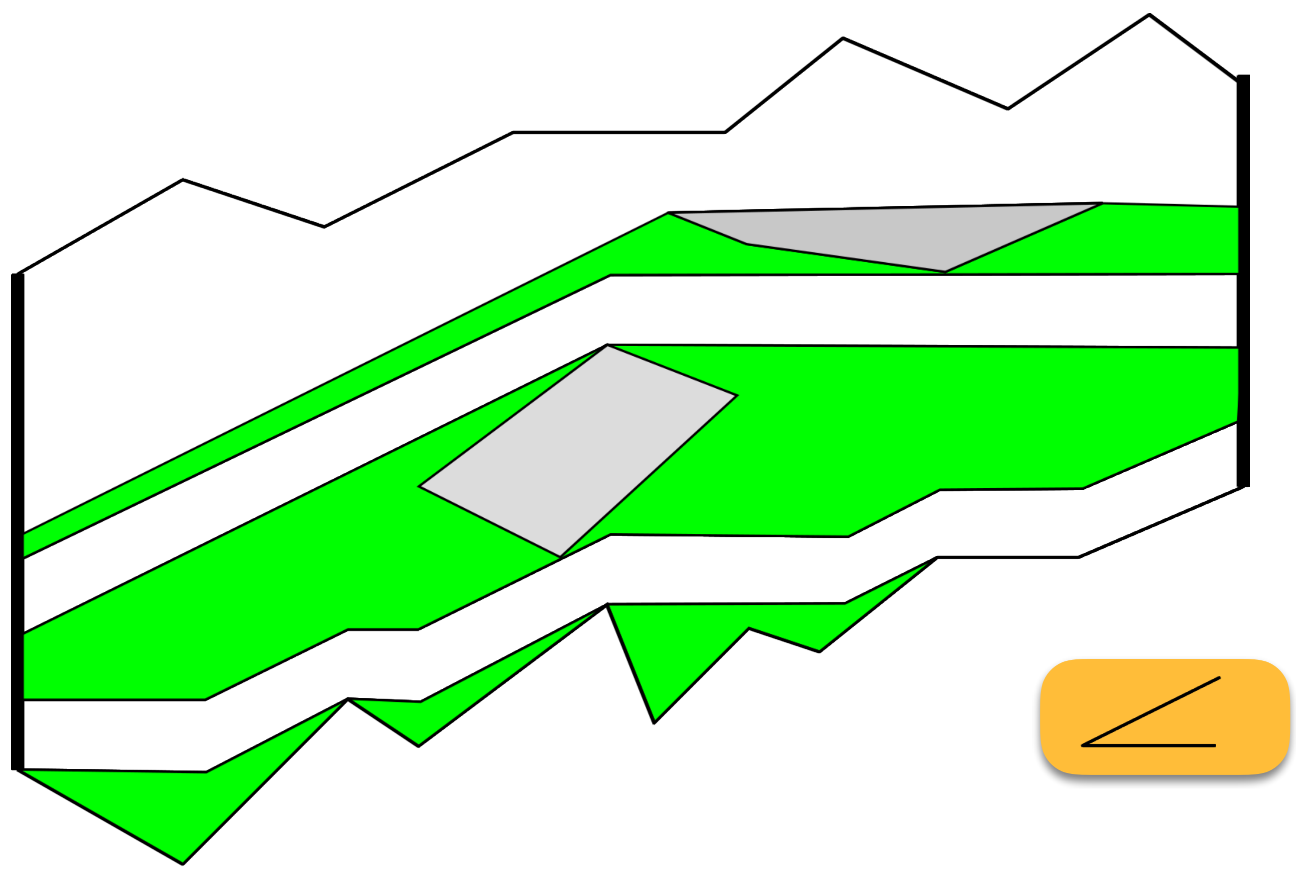
We want:

- Maximum number of non crossing thick paths from source to sink
- Slope should be within a given cone C
    - X-monotone
    - Limited speed ⇨ Limited slope
- We showed how to adapt the waterfall construction to compute the maximum number of thick non-crossing paths with a given slope range (≙C-respecting)



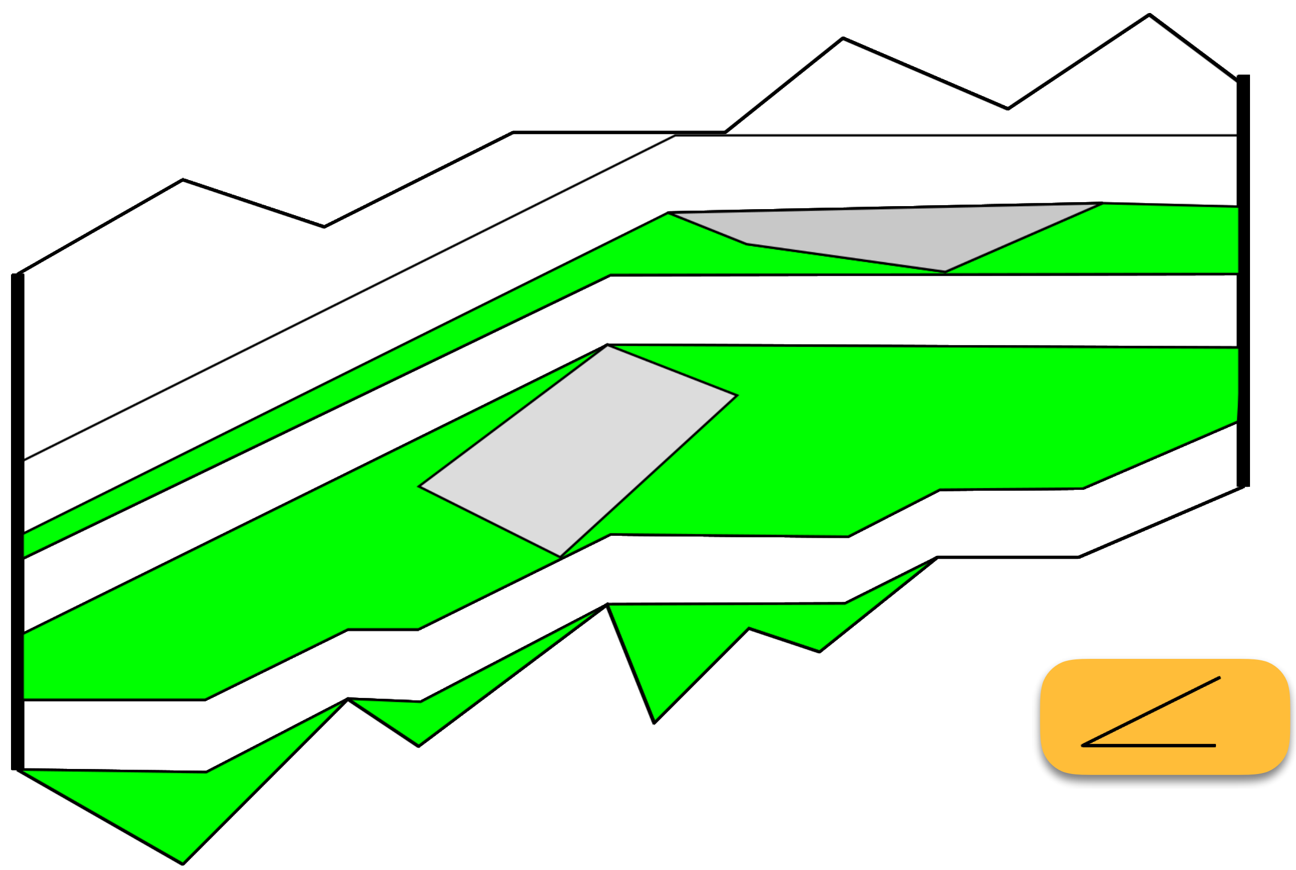
We want:

- Maximum number of non crossing thick paths from source to sink
- Slope should be within a given cone C
    - X-monotone
    - Limited speed ⇨ Limited slope
- We showed how to adapt the waterfall construction to compute the maximum number of thick non-crossing paths with a given slope range (≙C-respecting)



We want:

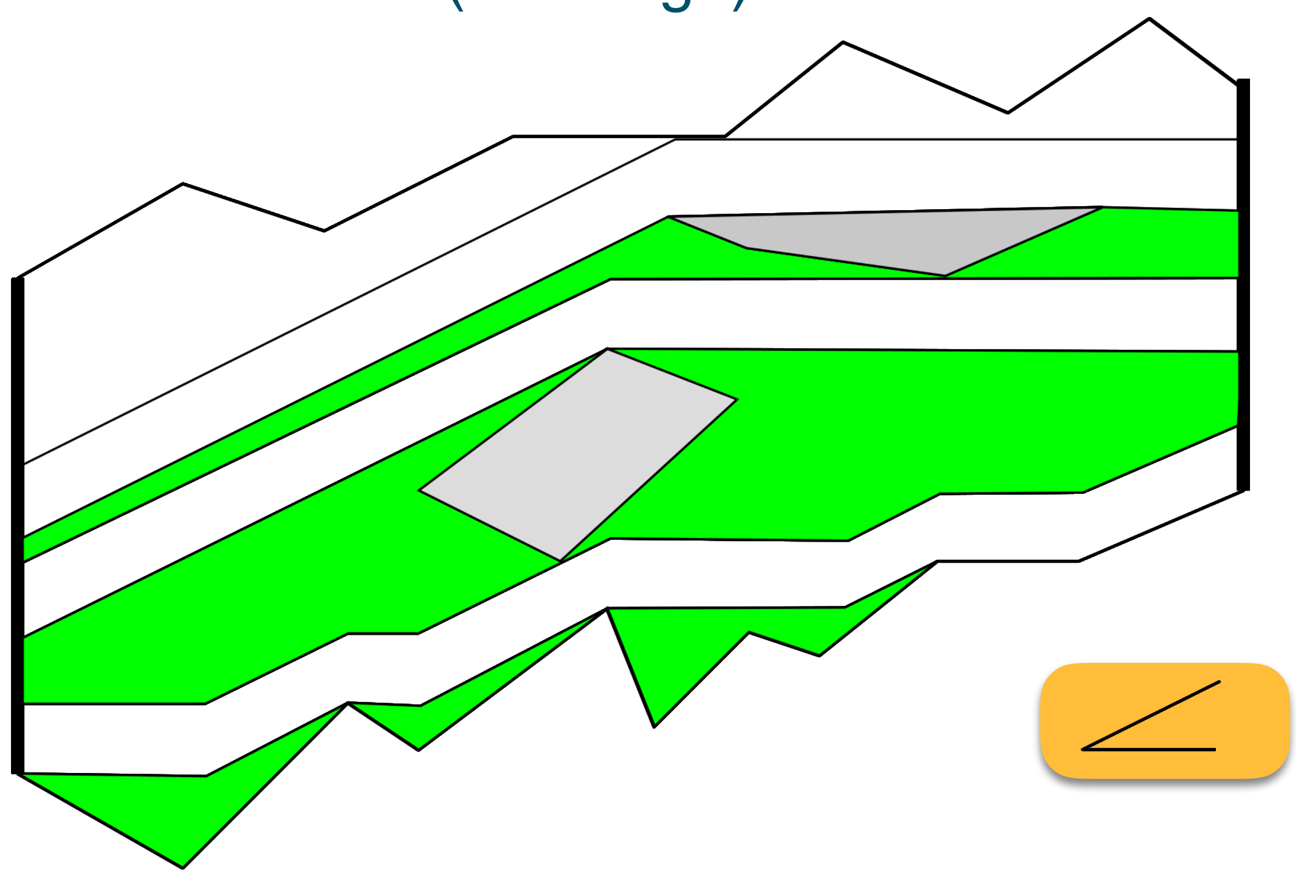
- Maximum number of non crossing thick paths from source to sink
- Slope should be within a given cone C
  - X-monotone
  - Limited speed ⇨ Limited slope
- We showed how to adapt the waterfall construction to compute the maximum number of thick non-crossing paths with a given slope range (≙C-respecting)



We want:

- Maximum number of non crossing thick paths from source to sink
- Slope should be within a given cone C
  - X-monotone
  - Limited speed ⇨ Limited slope
- We showed how to adapt the waterfall construction to compute the maximum number of thick non-crossing paths with a given slope range (≙C-respecting)

  Theorem: A representation of the maximum number of C-respecting thick-non-crossing paths can be found in O(nh+nlogn) time.



- We consider the time-space diagram—the geometric representation
- Inserting a new train: Route path from start to end station
- Paths not arbitrarily close ⇨ temporal distance (different to trains running in same or opposite direction)
- We think of train paths as “blown-up” line segments = thick paths
- Blown up by temporal distance (can be minimum, or more)
- How to route those thick paths? Concepts from Computational Geometry
- Need to make some adaptations, for example, if stations are lines, no path could cross these

➡ 1. Show how to construct the appropriate polygonal domain

➡ 2. Show how to route the maximum number of thick non-crossing paths in that domain:

  - Paths should be x-monotone (we cannot go back in time)
  - Trains have a maximum speed ⇨ paths have a limited slope

- We consider the time-space diagram—the geometric representation
- Inserting a new train: Route path from start to end station
- Paths not arbitrarily close ⇨ temporal distance (different to trains running in same or opposite direction)
- We think of train paths as “blown-up” line segments = thick paths
- Blown up by temporal distance (can be minimum, or more)
- How to route those thick paths? Concepts from Computational Geometry
- Need to make some adaptations, for example, if stations are lines, no path could cross these

➡ 1. Show how to construct the appropriate polygonal domain

➡ 2. Show how to route the maximum number of thick non-crossing paths in that domain: ✔
  - Paths should be x-monotone (we cannot go back in time)
  - Trains have a maximum speed ⇨ paths have a limited slope

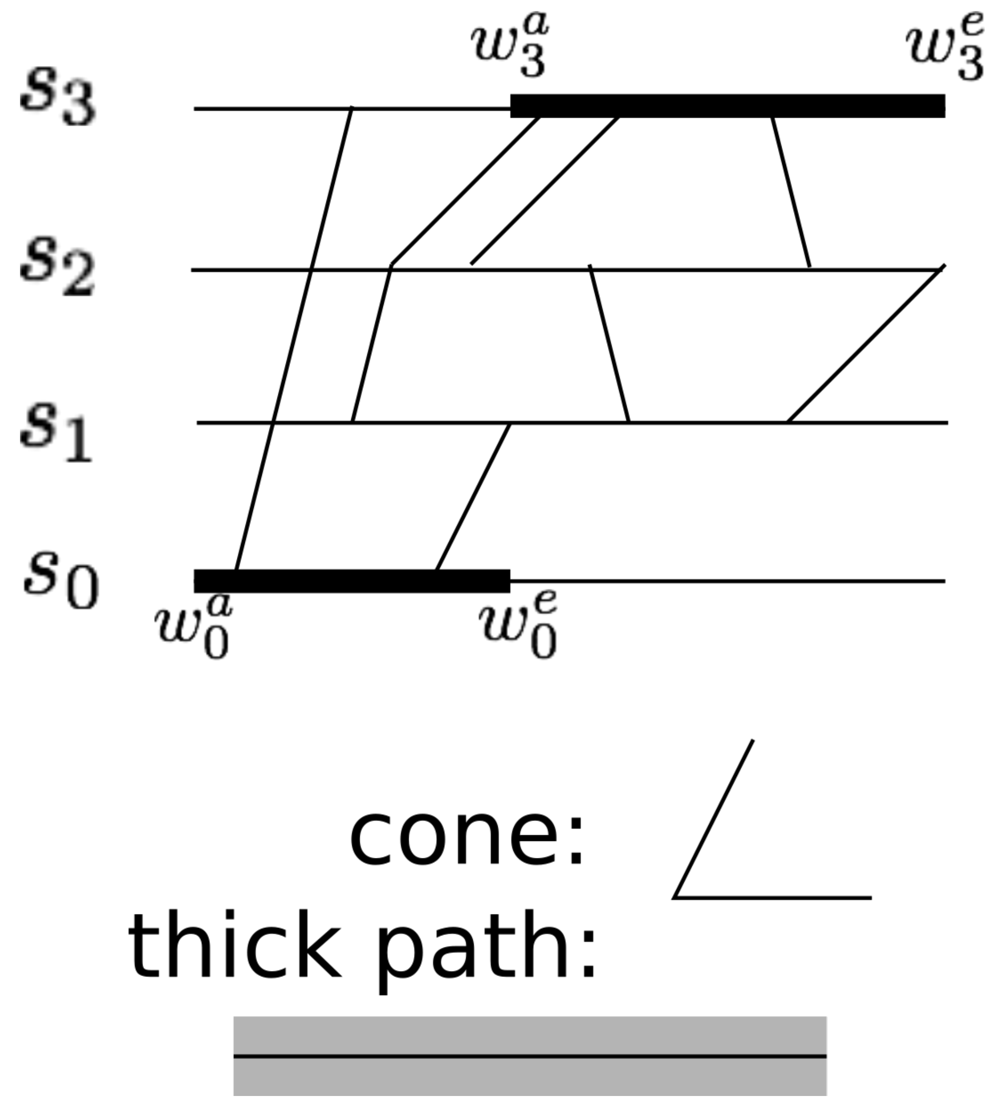
- We consider the time-space diagram—the geometric representation
- Inserting a new train: Route path from start to end station
- Paths not arbitrarily close ⇨ temporal distance (different to trains running in same or opposite direction)
- We think of train paths as “blown-up” line segments = thick paths
- Blown up by temporal distance (can be minimum, or more)
- How to route those thick paths? Concepts from Computational Geometry
- Need to make some adaptations, for example, if stations are lines, no path could cross these

➡ 1. Show how to construct the appropriate polygonal domain

➡ 2. Show how to route the maximum number of thick non-crossing paths in that domain: ✔
  - Paths should be x-monotone (we cannot go back in time)
  - Trains have a maximum speed ⇨ paths have a limited slope

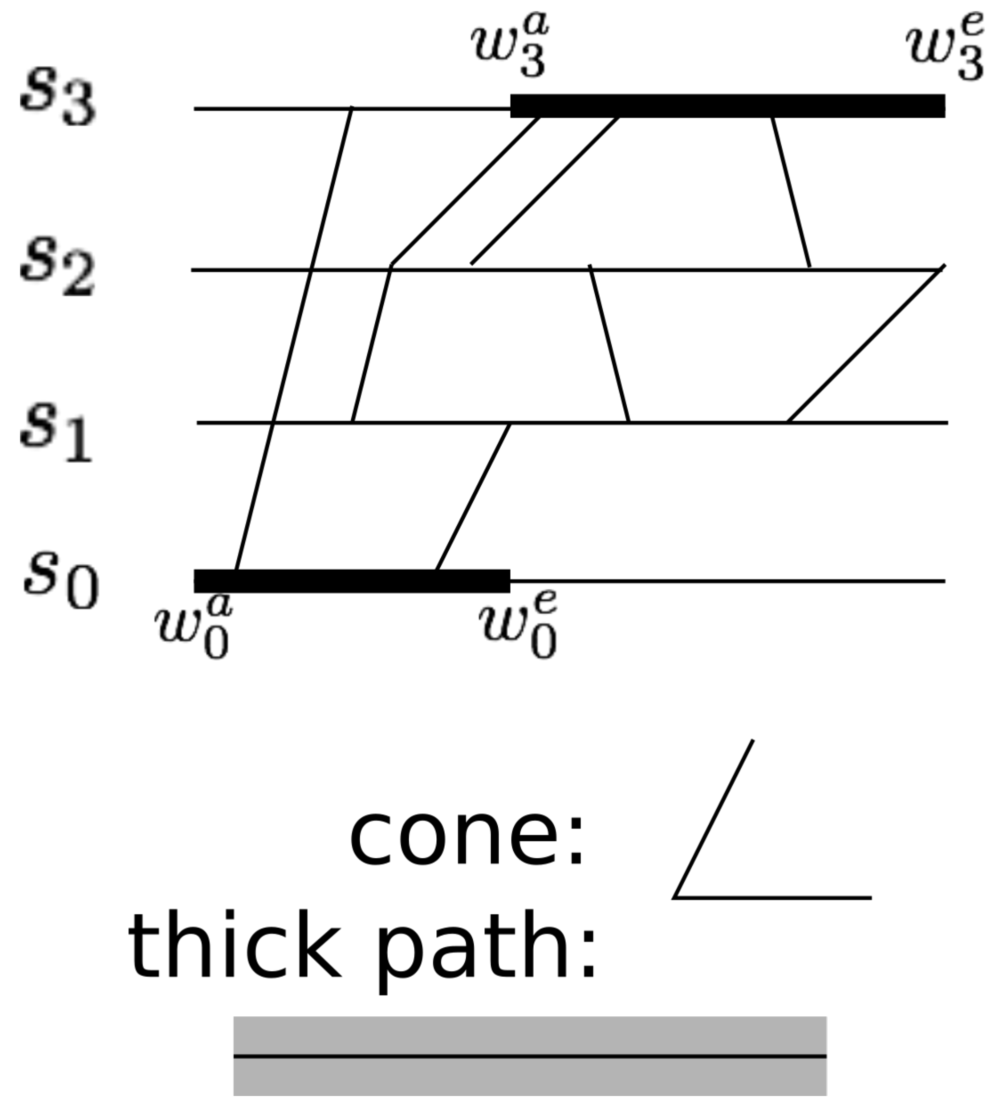
Still left to do

Construction of Polygonal Domain from the Timetable

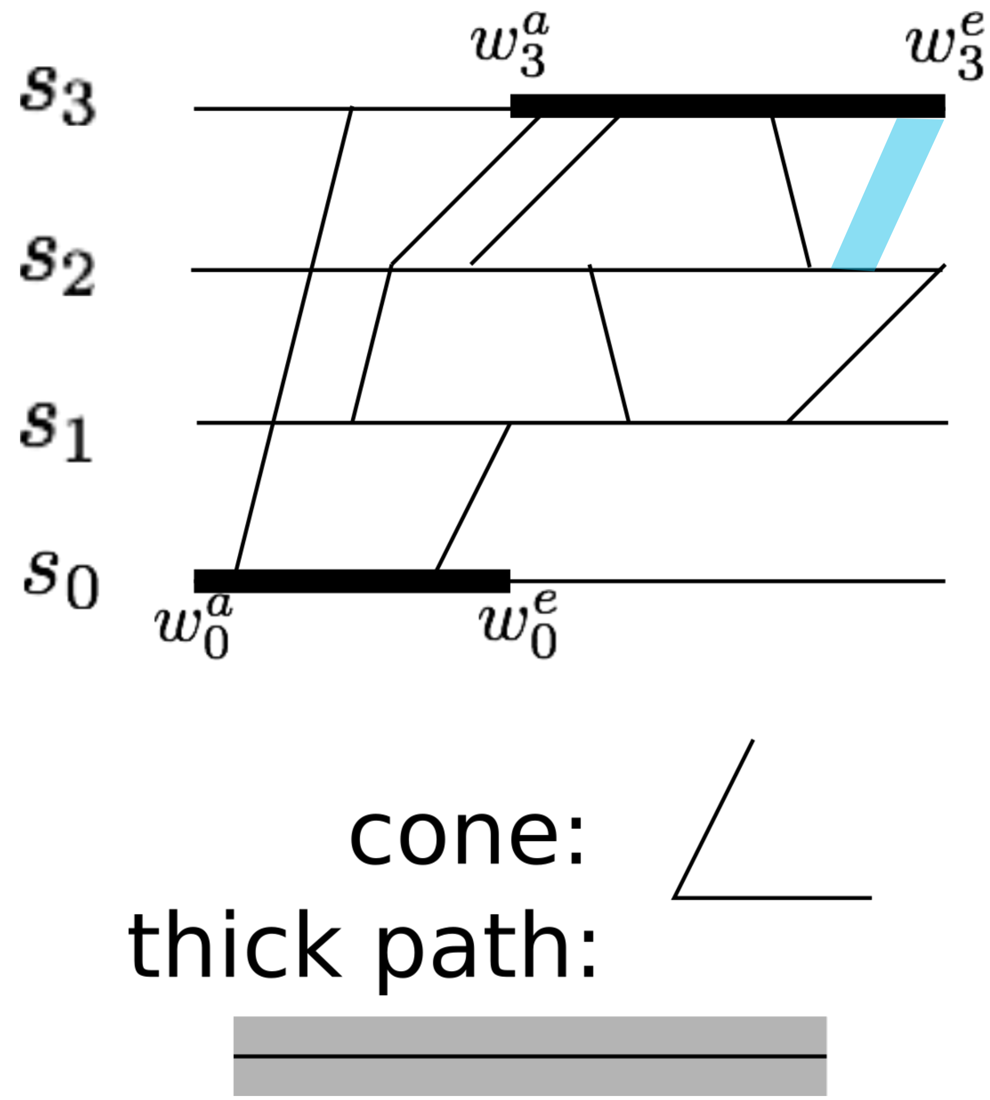


s_3
s_2
s_1
s_0
w_3^a
w_3^e
w_0^a
w_0^e
cone:
thick path:

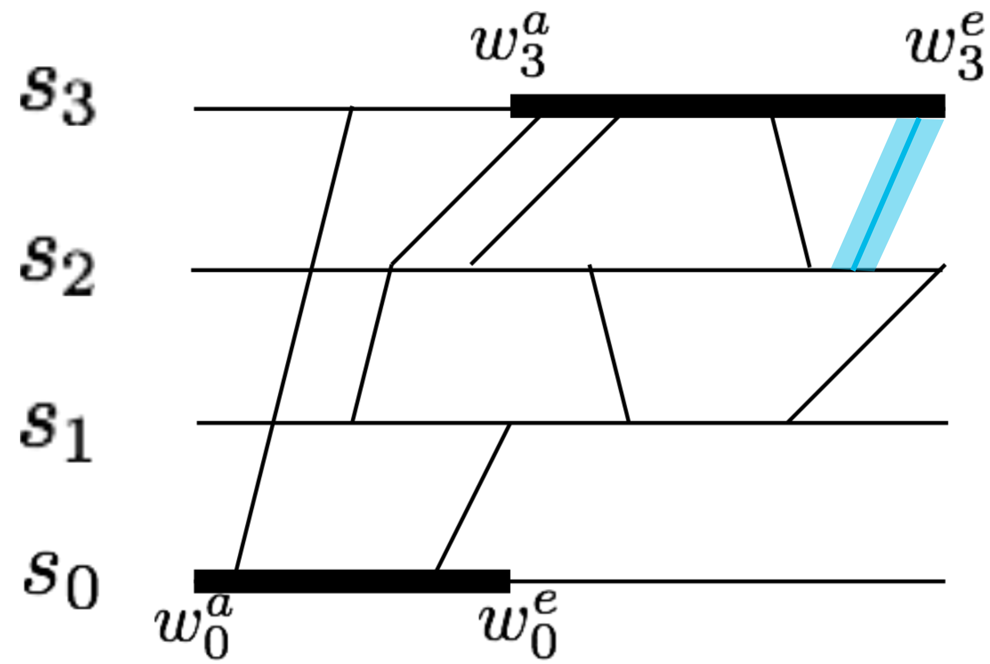
If we would define the time windows as source and sink

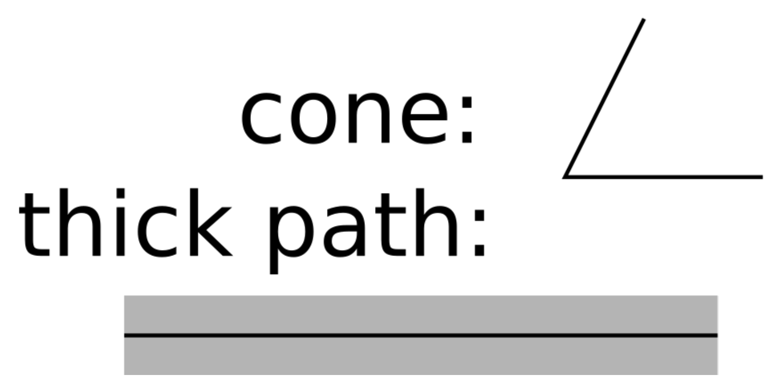


If we would define the time windows as source and sink

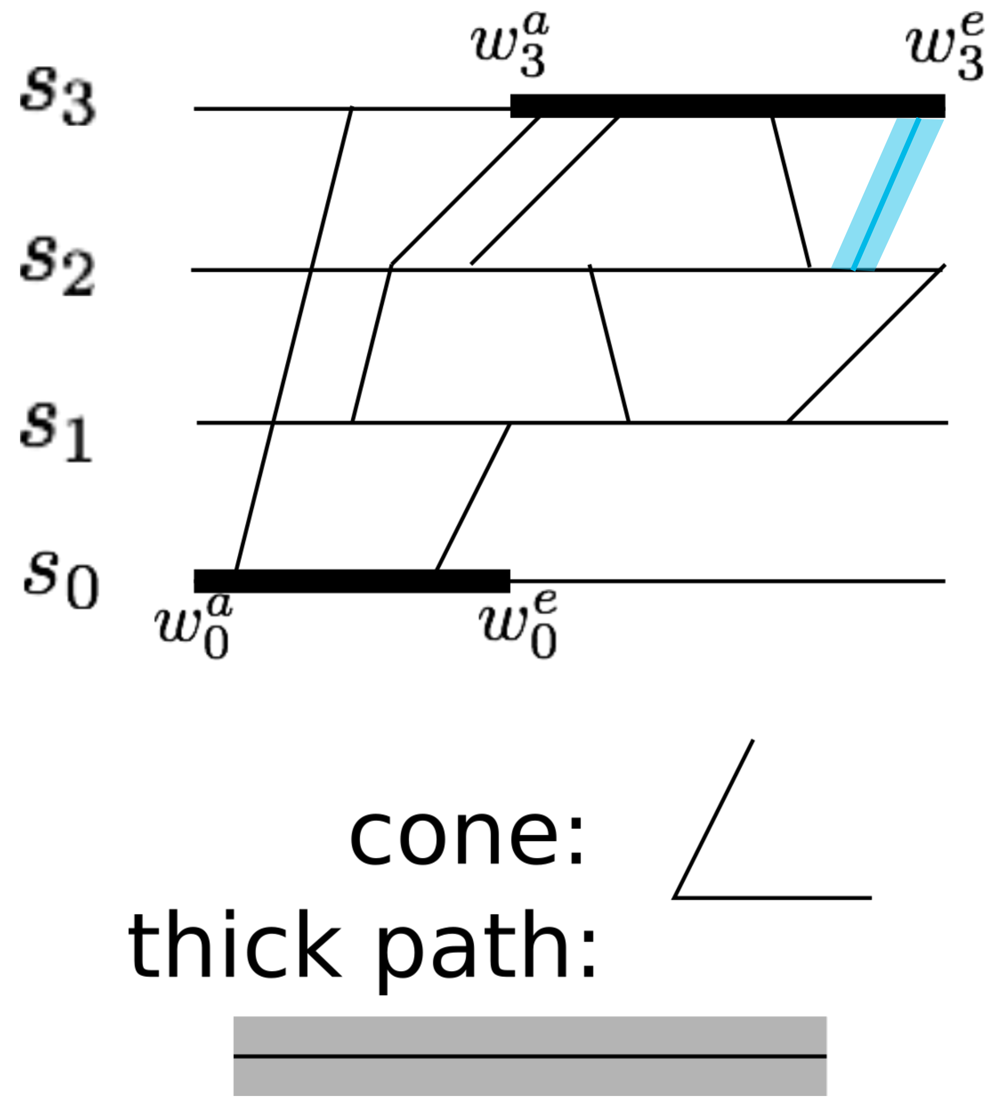


If we would define the time windows as source and sink





If we would define the time windows as source and sink
⇨ Possible thick paths would correspond to train paths in a smaller time interval

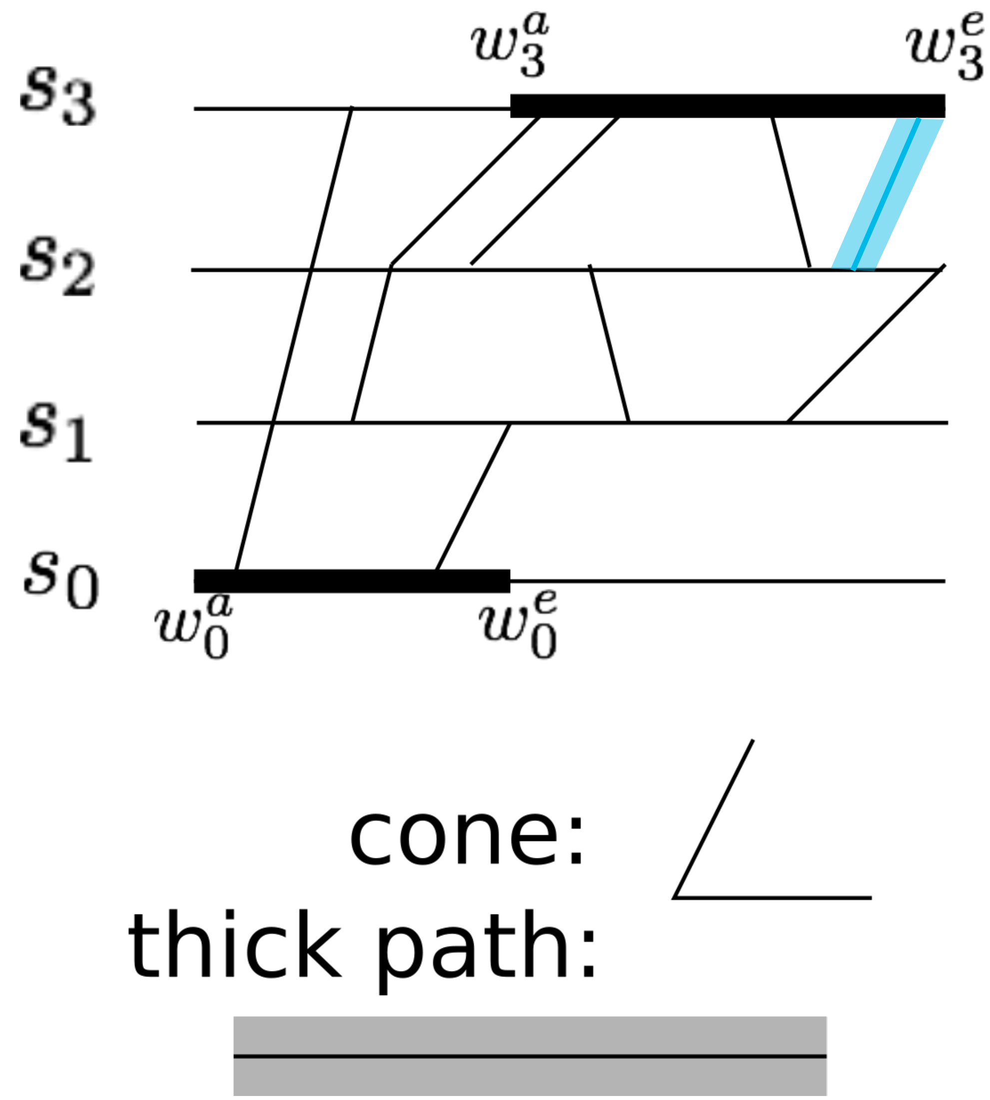


If we would define the time windows as source and sink

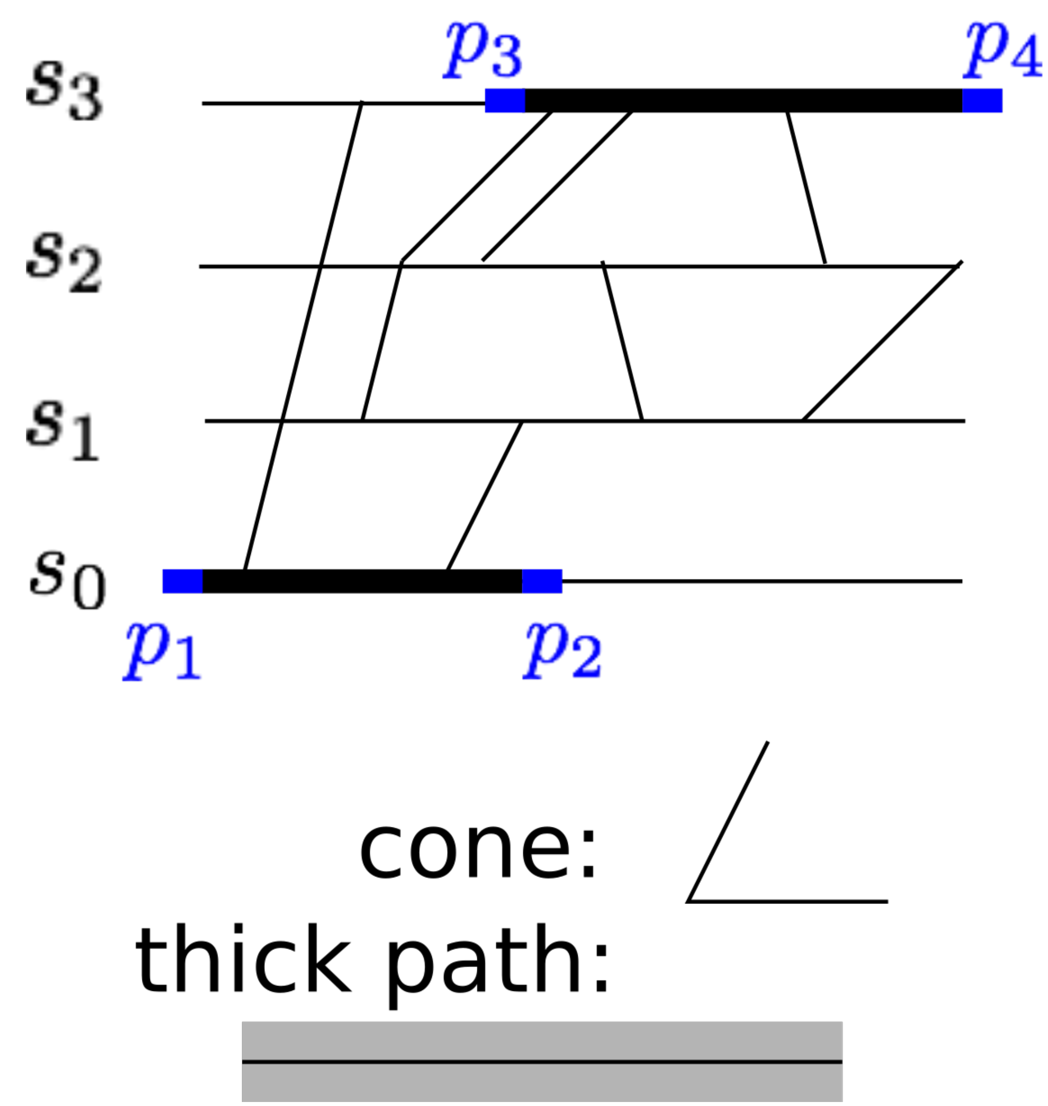
⇨ Possible thick paths would correspond to train paths in a smaller time interval

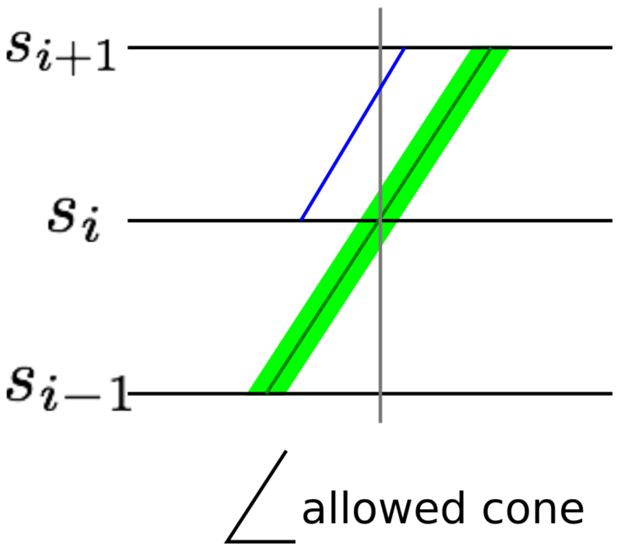
⇨ Extend the time windows by d/2to both sides to create $\mathbf{\Gamma}_s$ and $\mathbf{\Gamma}_t$

($\mathbf{\Gamma}_s=[p_1,p_2]$, $\mathbf{\Gamma}_t=[p_3,p_4]$)

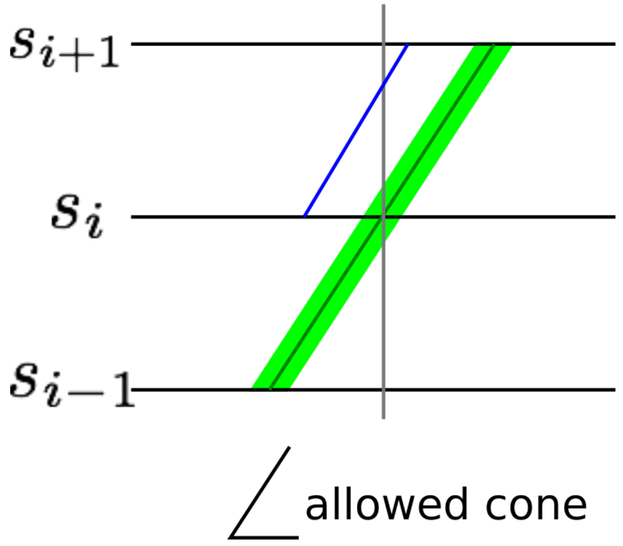


If we would define the time windows as source and sink
⇨ Possible thick paths would correspond to train paths in a smaller time interval
⇨ Extend the time windows by d/2to both sides to create $\boldsymbol{\Gamma}_s$ and $\boldsymbol{\Gamma}_t$
($\boldsymbol{\Gamma}_s=[p_1,p_2]$, $\boldsymbol{\Gamma}_t=[p_3,p_4]$)

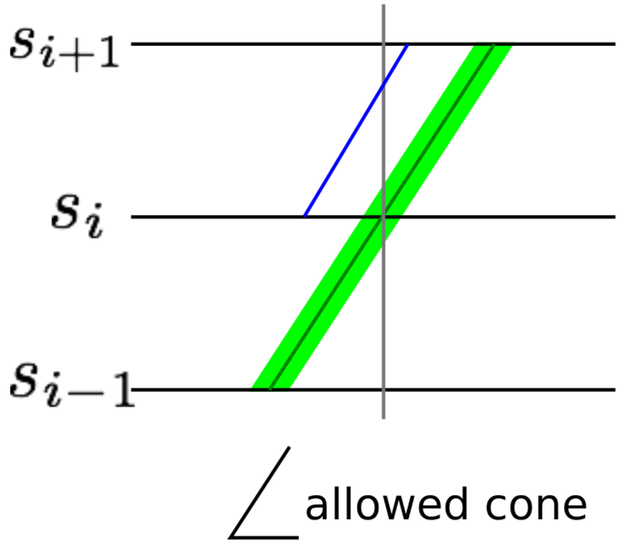



s_{i+1}
s_i
s_{i-1}
allowed cone

Vertical lines at stations are obstacles



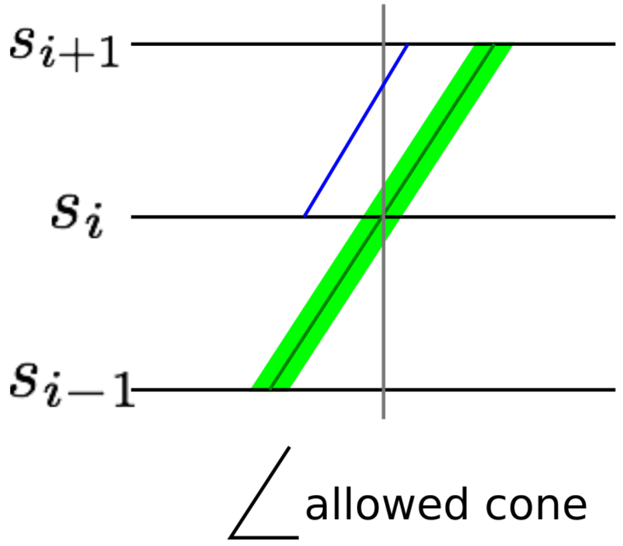
Vertical lines at stations are obstacles
⇨ We need to delete them



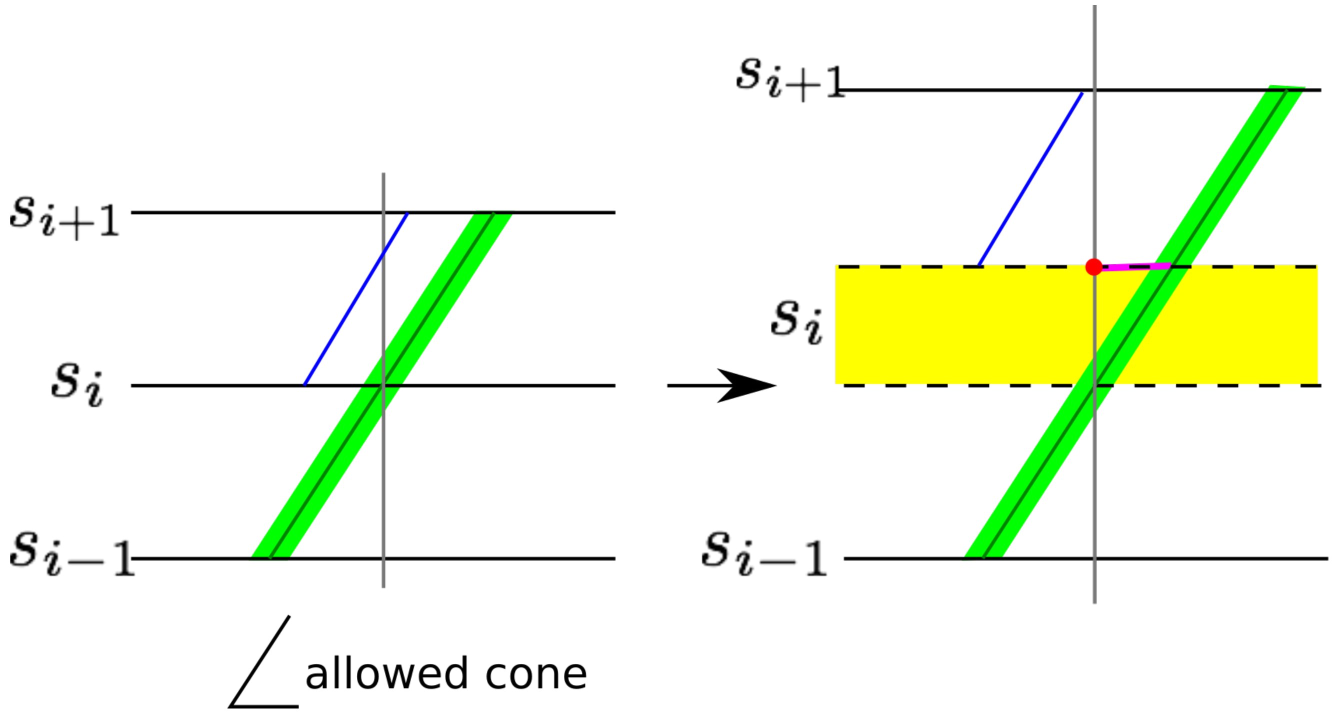
Vertical lines at stations are obstacles

⇨ We need to delete them

We need to be able to spend some time at a station



Vertical lines at stations are obstacles
⇨ We need to delete them
We need to be able to spend some time at a station
⇨ “Cut” each station open and blow up by vertical distance:



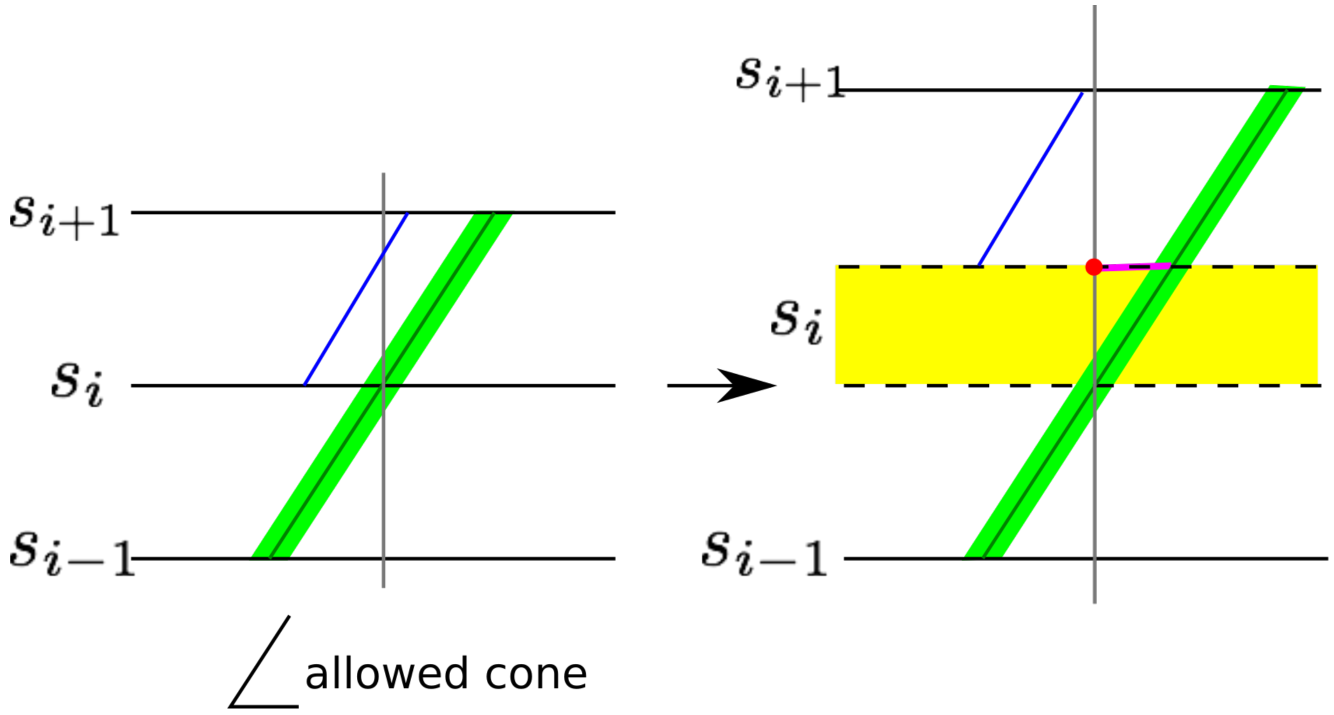
Vertical lines at stations are obstacles

⇨ We need to delete them

We need to be able to spend some time at a station

⇨ “Cut” each station open and blow up by vertical distance:

- If the station s has exactly k sidetracks, we insert a vertical distance of k*d



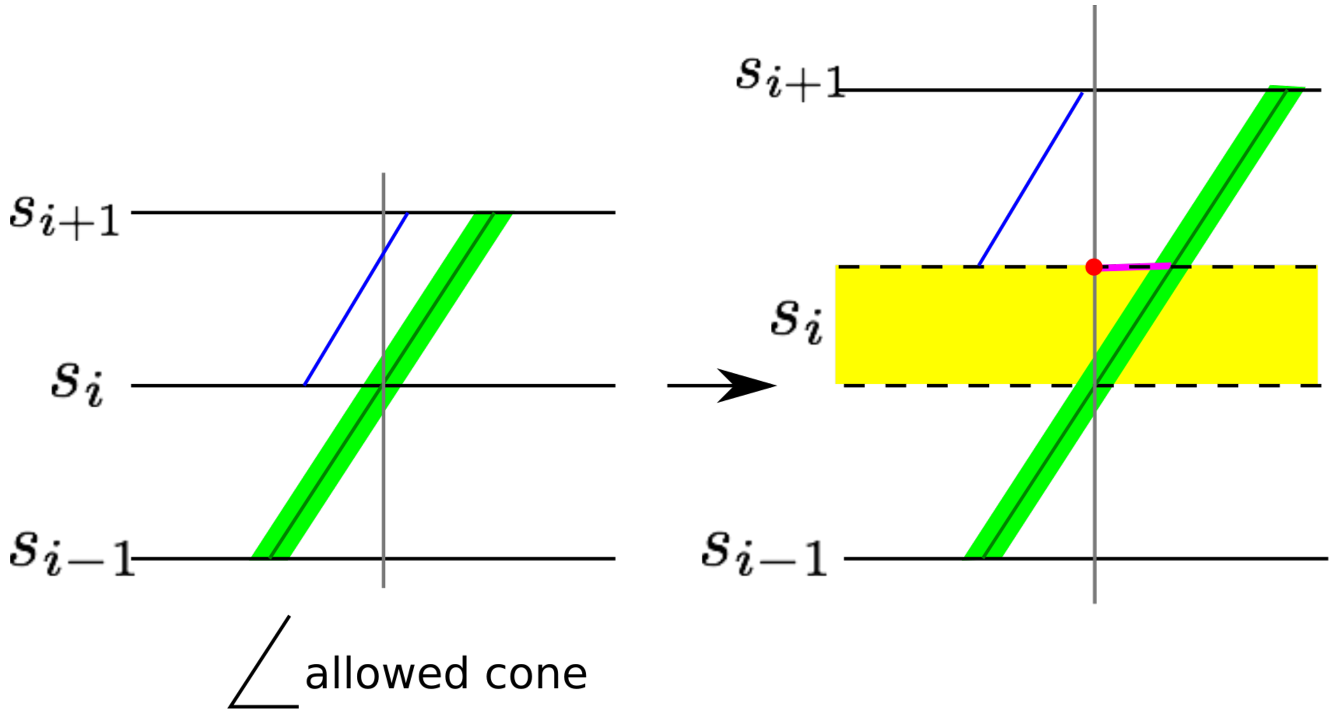
Vertical lines at stations are obstacles

⇨ We need to delete them

We need to be able to spend some time at a station

⇨ "Cut" each station open and blow up by vertical distance:

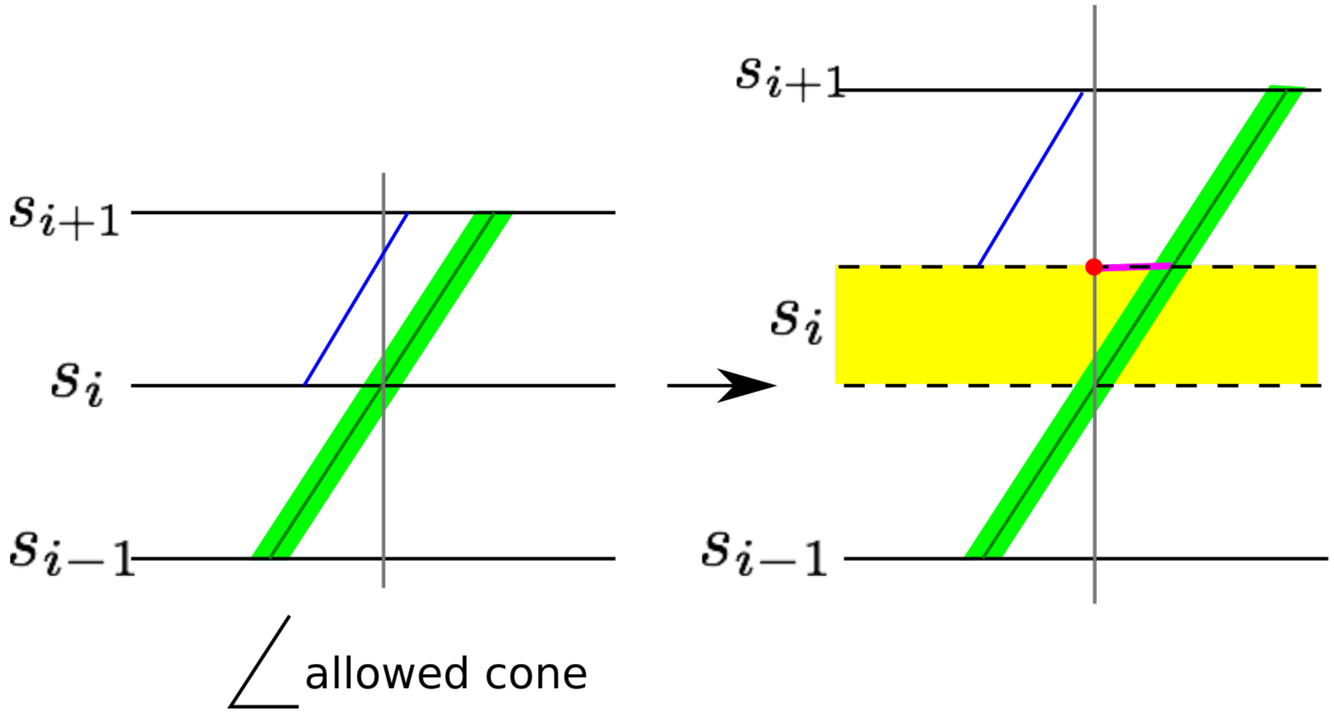
- If the station s has exactly k sidetracks, we insert a vertical distance of k*d
- If no such limit exists, we can insert a vertical distance of min{$|\boldsymbol{\Gamma}_s|$+d, $|\boldsymbol{\Gamma}_t|$+d}



Vertical lines at stations are obstacles
⇨ We need to delete them

We need to be able to spend some time at a station
⇨ "Cut" each station open and blow up by vertical distance:
- If the station s has exactly k sidetracks, we insert a vertical distance of k*d
- If no such limit exists, we can insert a vertical distance of min{$|\mathbf{\Gamma}_s|$+d, $|\mathbf{\Gamma}_t|$+d}

But now the time of departure cannot be reached by our paths with limited slope



Vertical lines at stations are obstacles

⇨ We need to delete them

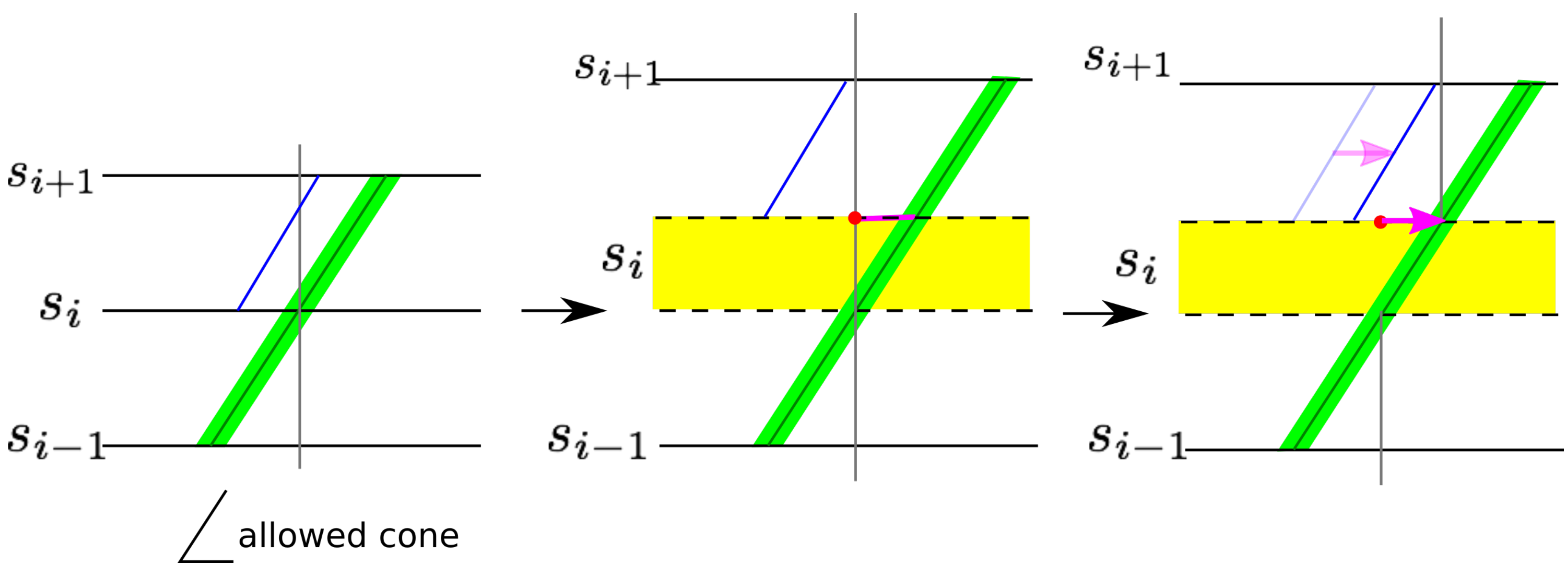
We need to be able to spend some time at a station

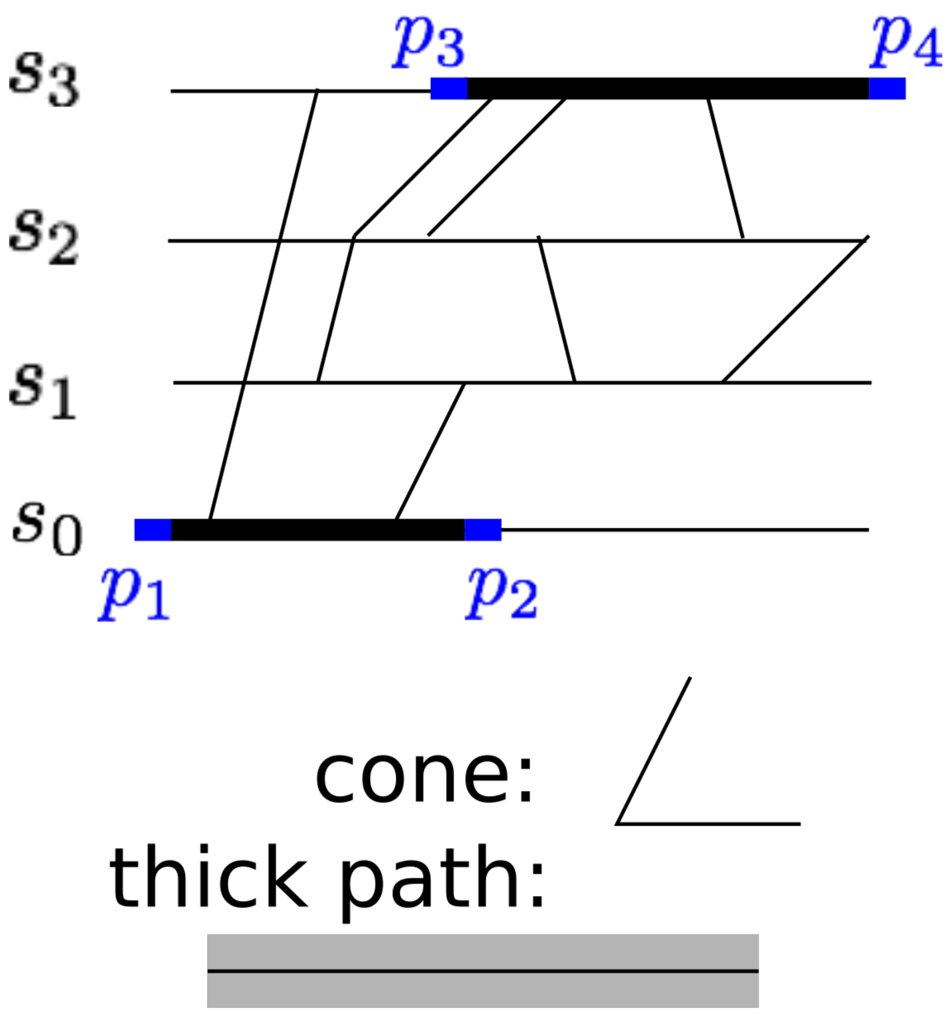
⇨ "Cut" each station open and blow up by vertical distance:

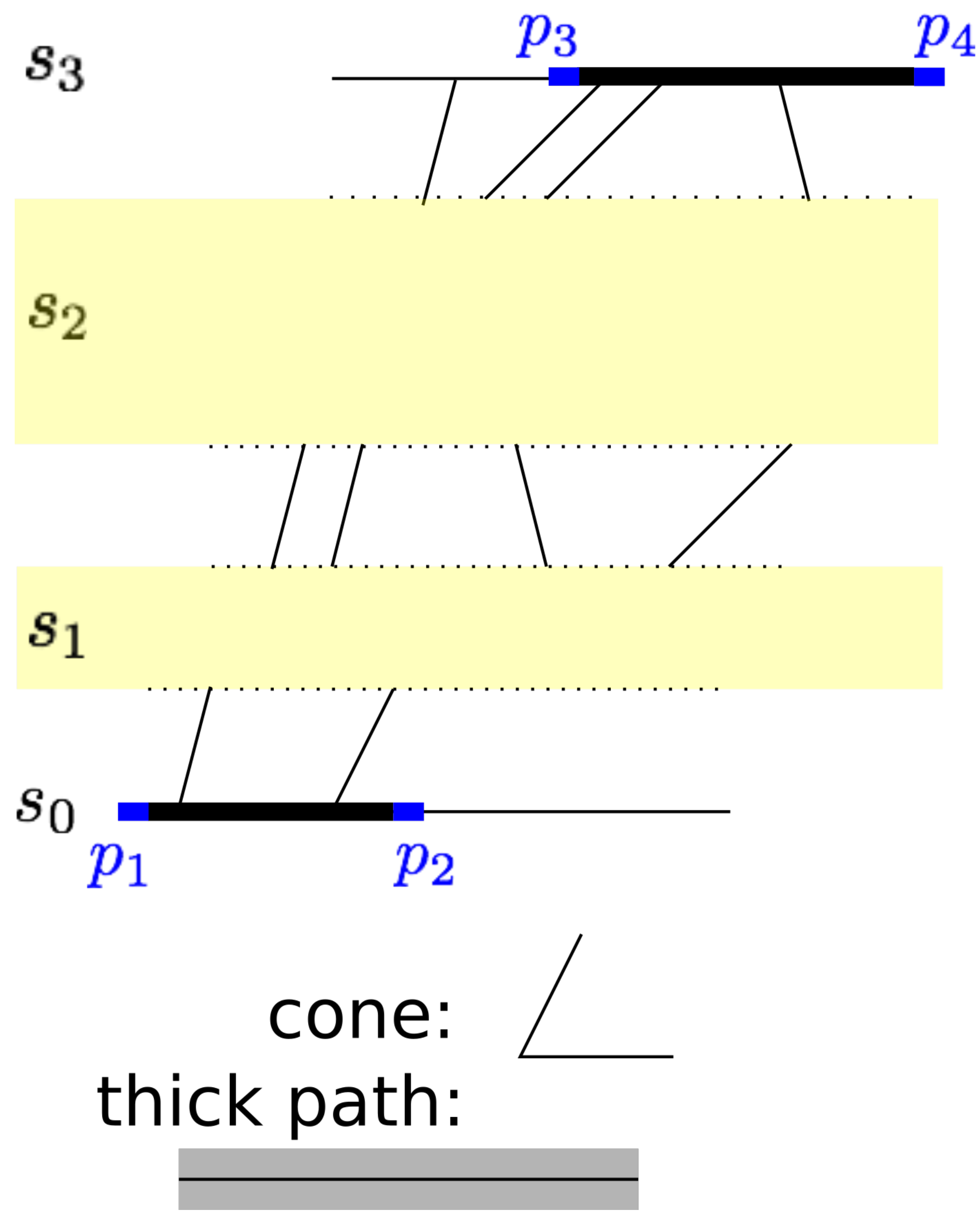
- If the station s has exactly k sidetracks, we insert a vertical distance of k*d
- If no such limit exists, we can insert a vertical distance of min{$|\Gamma_s|$+d, $|\Gamma_t|$+d}

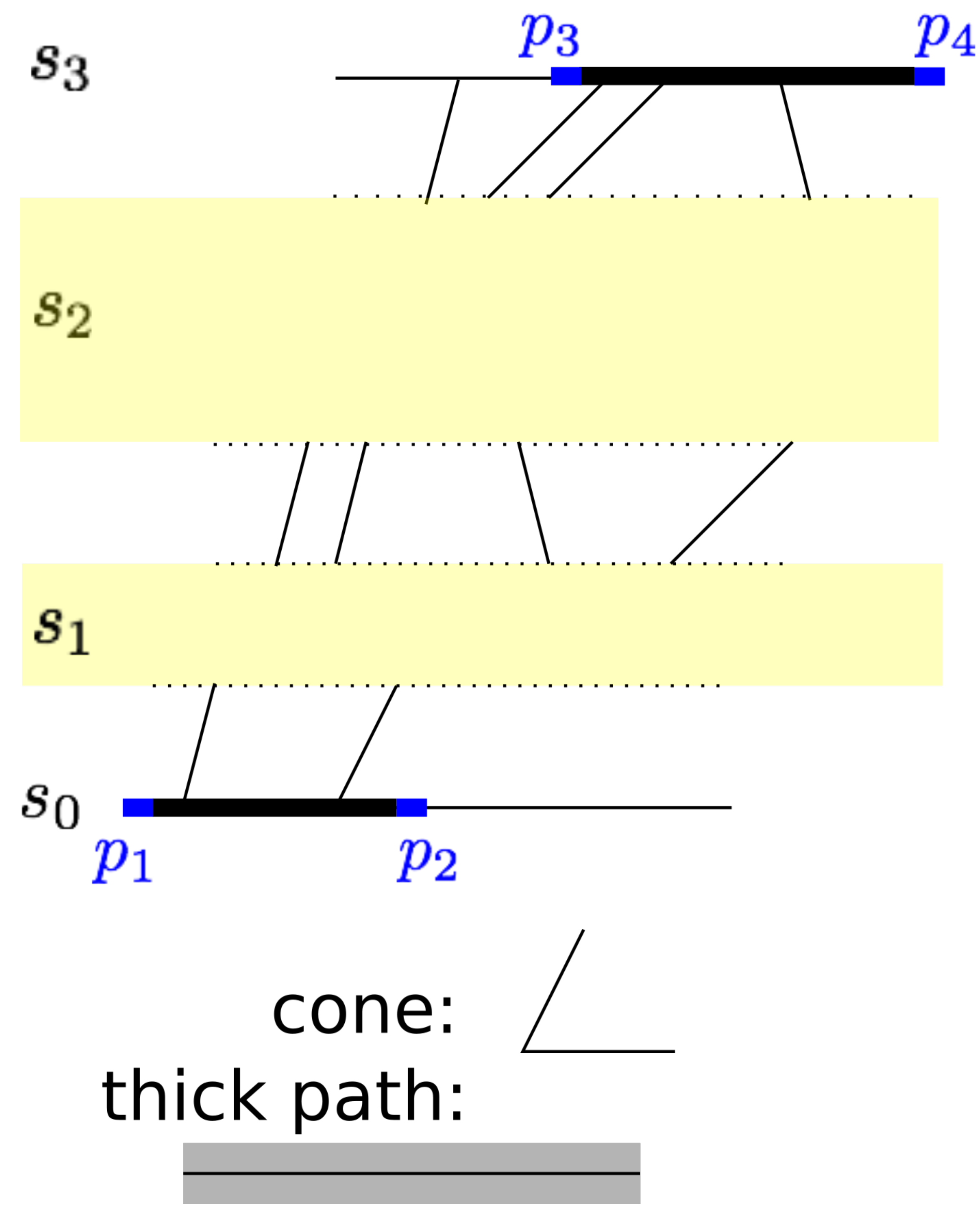
But now the time of departure cannot be reached by our paths with limited slope

⇨We need to shift the consecutive stations to the right, such that this path can be reached with limited slope



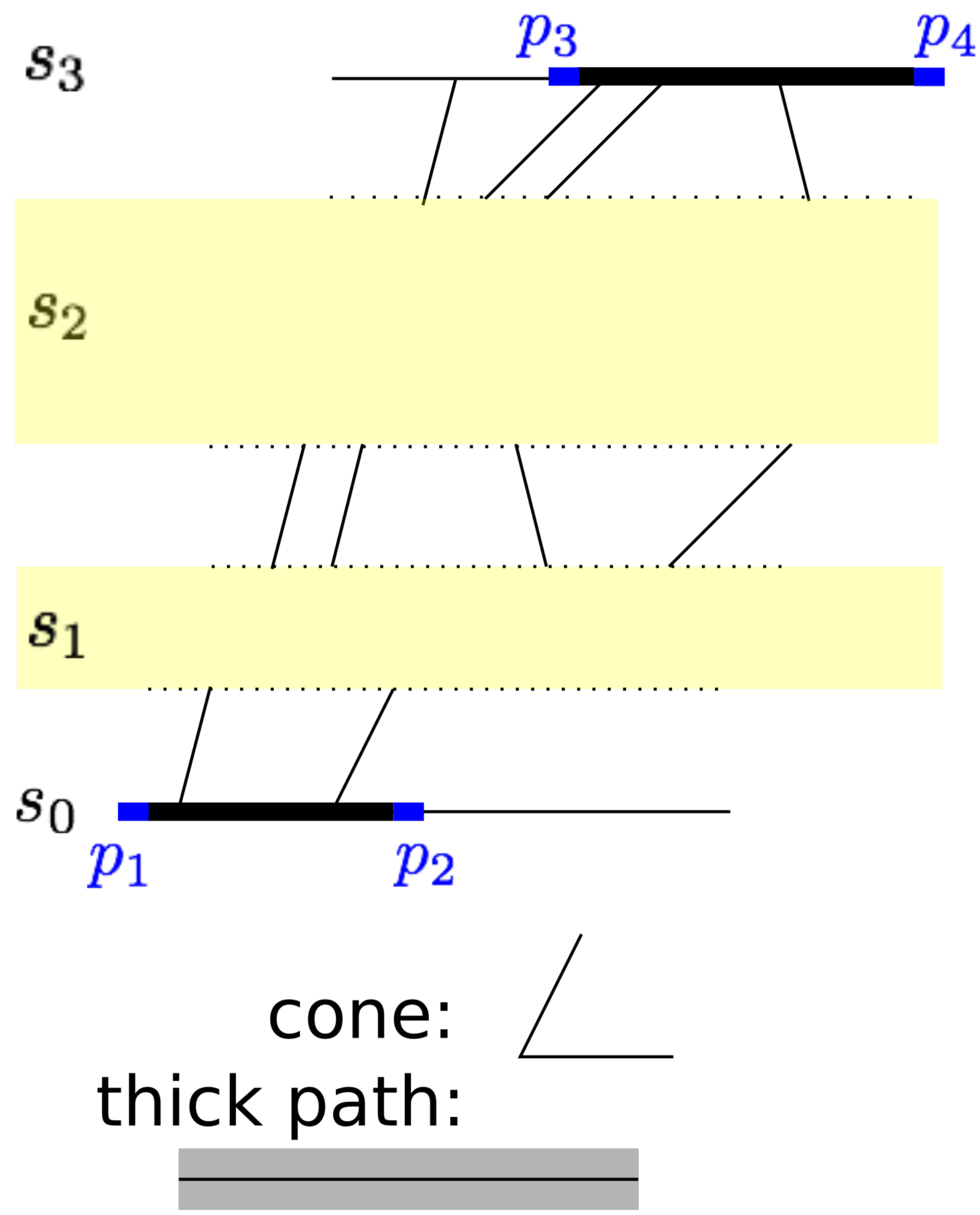

s_3
p_3
p_4
s_2
s_1
s_0
p_1
p_2
cone:
thick path:


s_3
p_3
p_4
s_2
s_1
s_0
p_1
p_2
cone:
thick path:

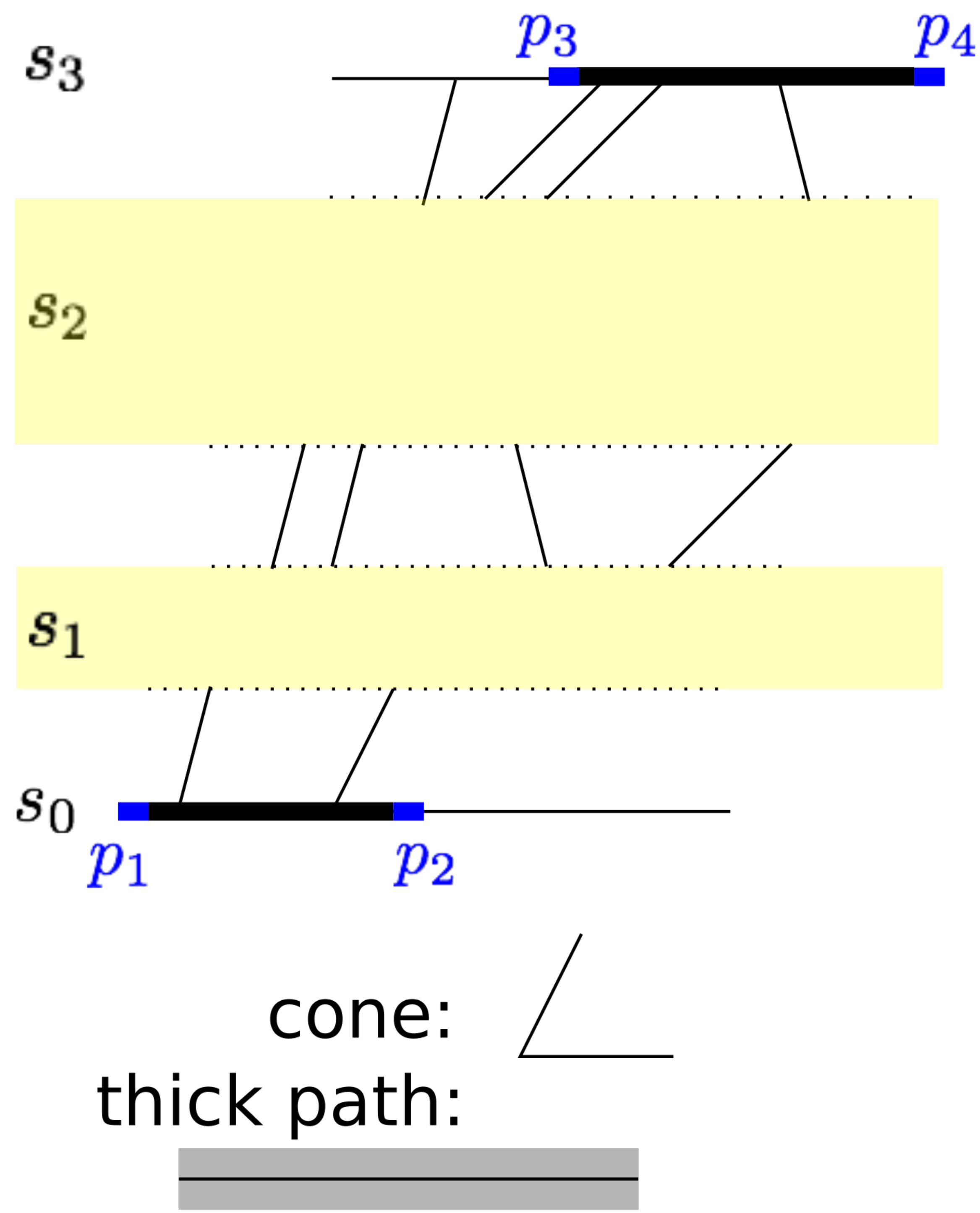


s_3
p_3
p_4
s_2
s_1
s_0
p_1
p_2
cone:
thick path:

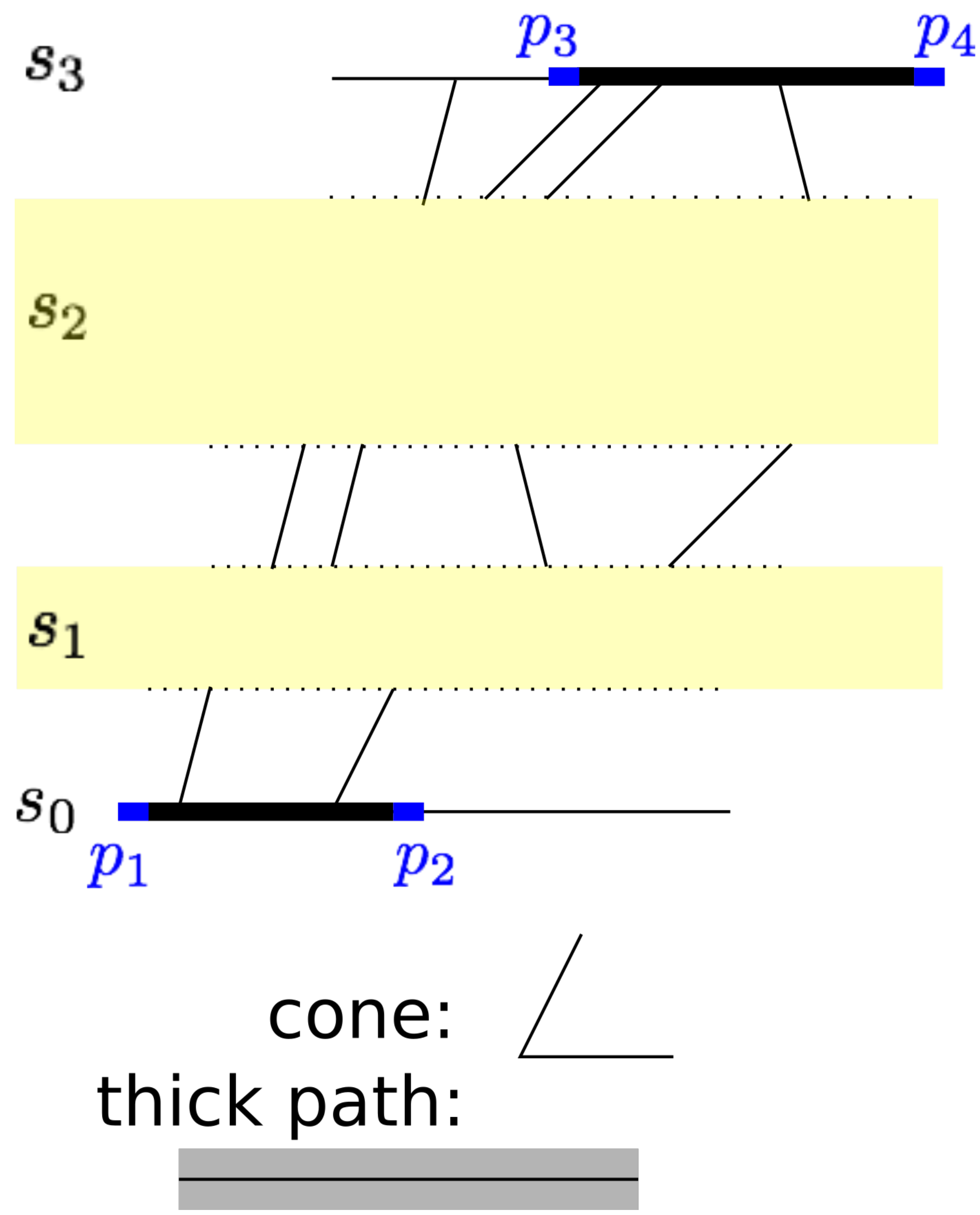
We need to keep a temporal distance to the existing trains in the timetable



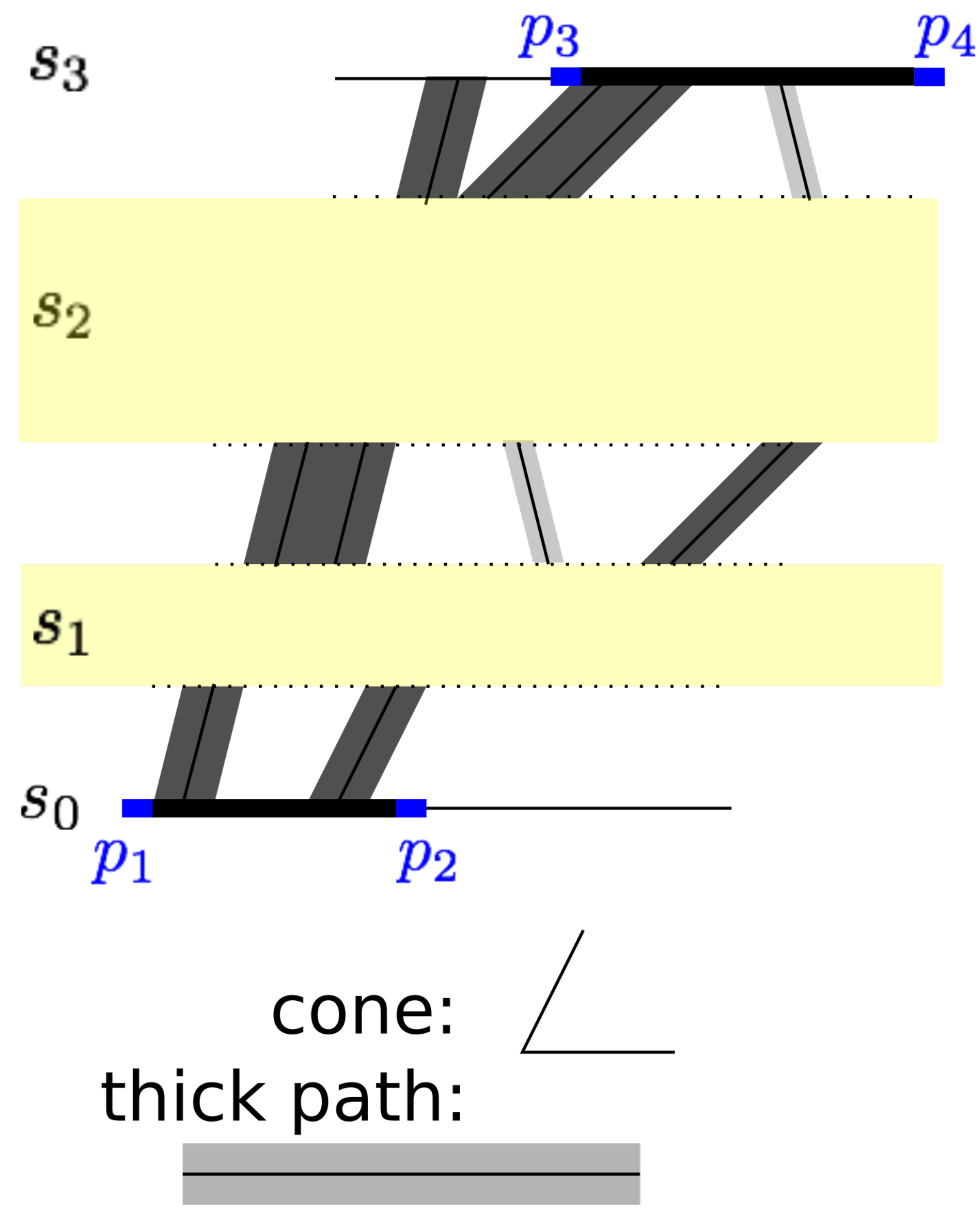
We need to keep a temporal distance to the existing trains in the timetable
⇨ “Blow them up” as polygonal obstacles:



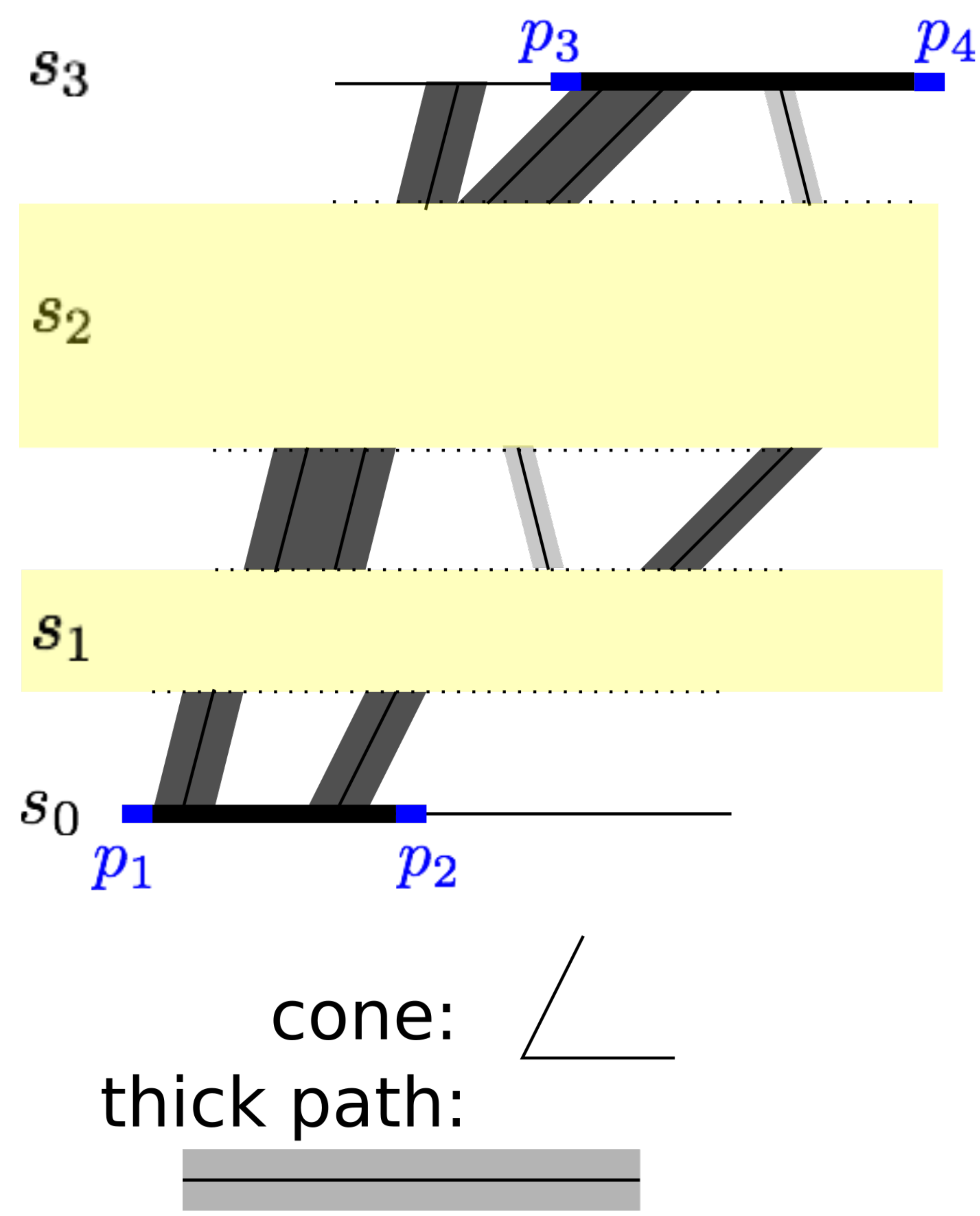
We need to keep a temporal distance to the existing trains in the timetable
⇨ "Blow them up" as polygonal obstacles: Insert the security distance (d_s, d_o)



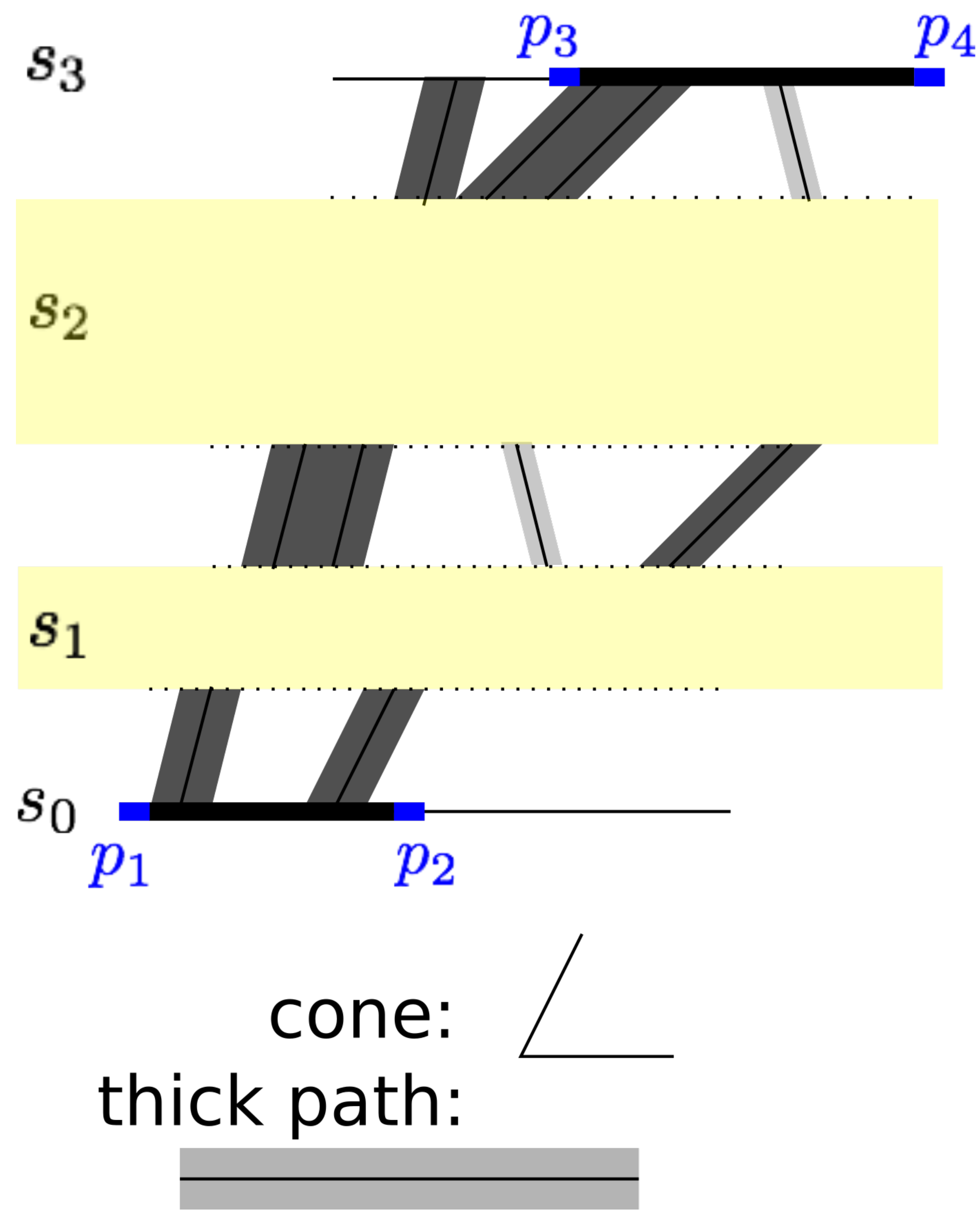
We need to keep a temporal distance to the existing trains in the timetable
⇨ “Blow them up” as polygonal obstacles: Insert the security distance (d_s, d_o)



In the example we used d_s=d, d_0=d/2

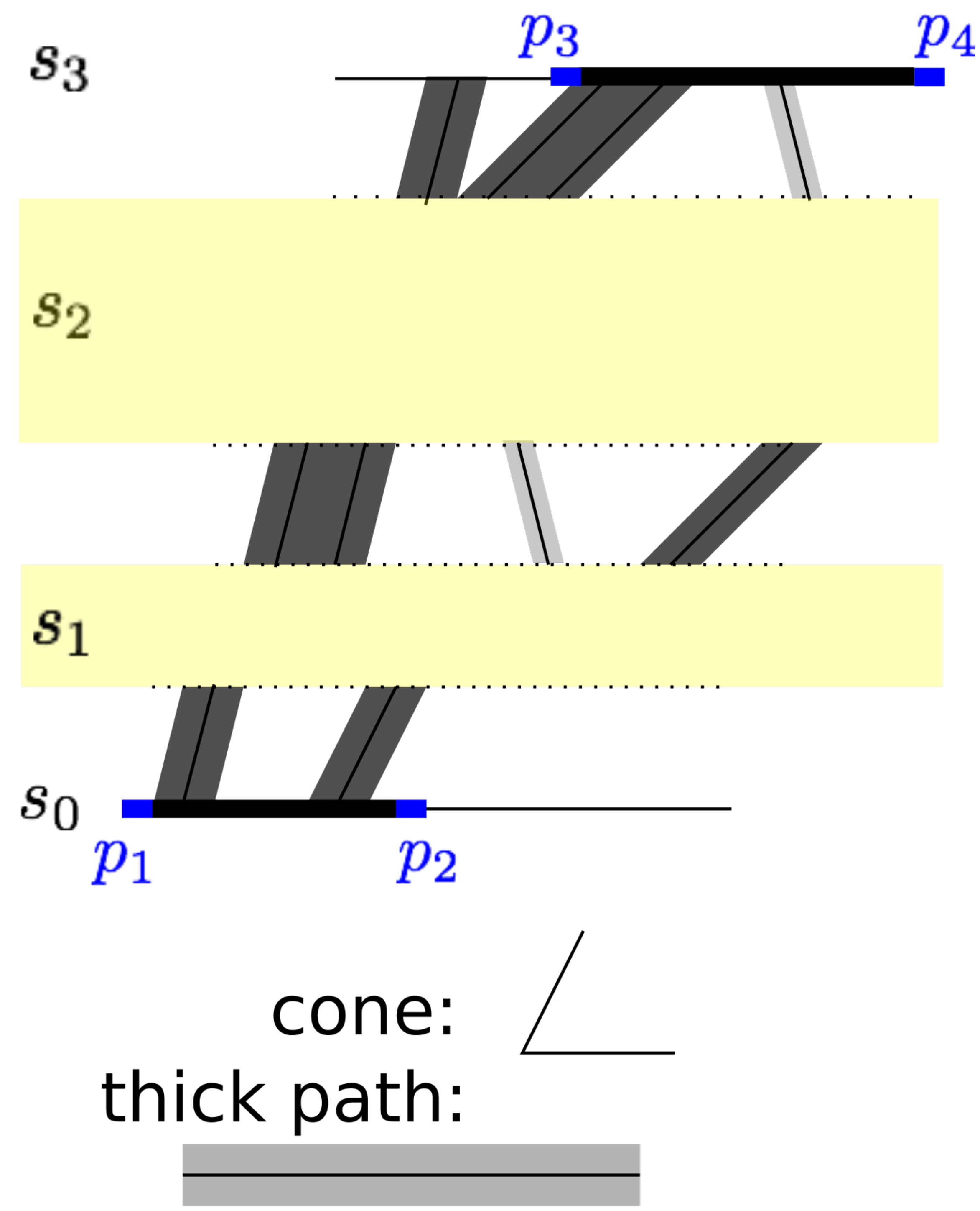

s_3
p_3
p_4
s_2
s_1
s_0
p_1
p_2
cone:
thick path:

We need to limit our outer polygon:



We need to limit our outer polygon:

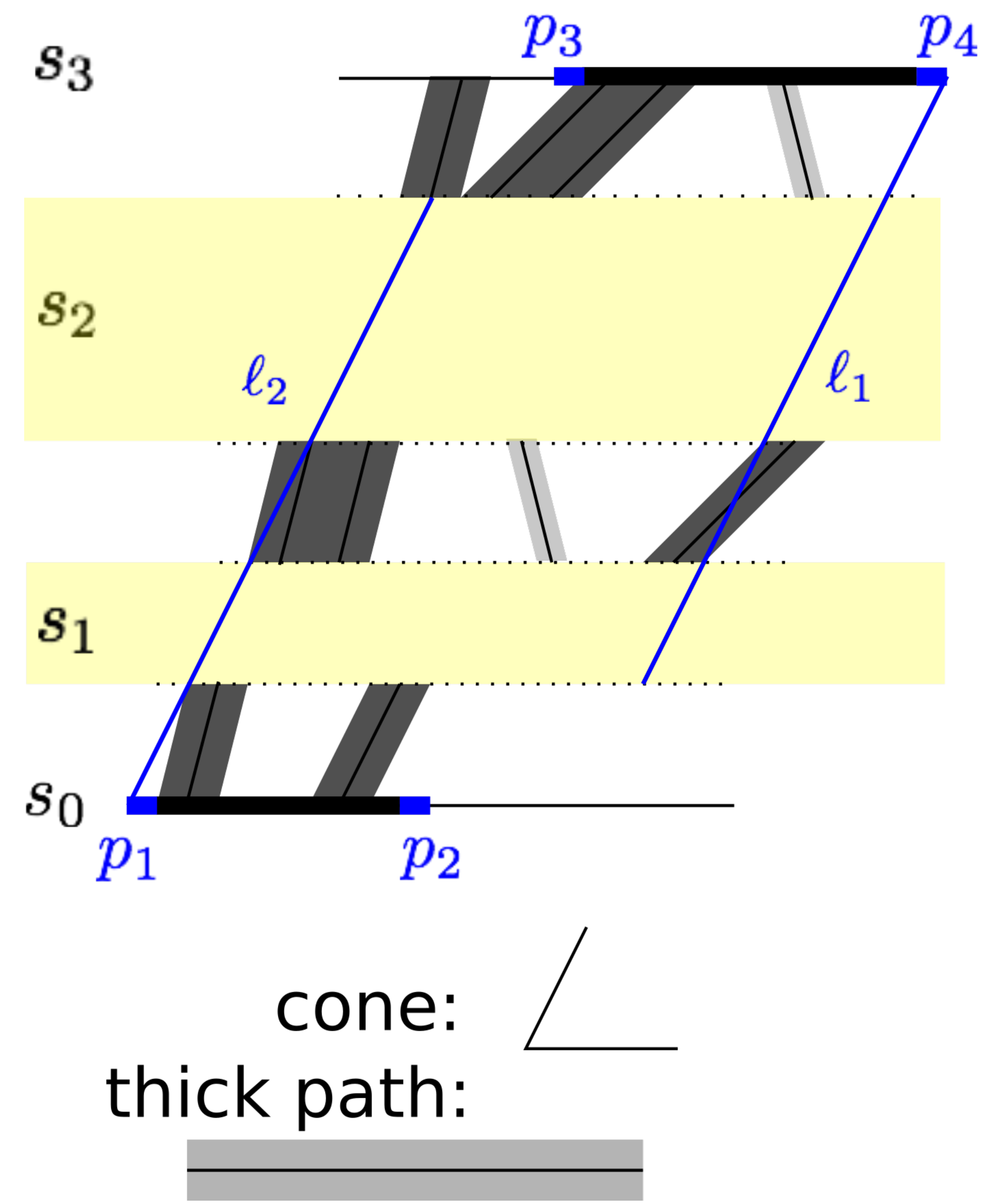
- No train can run earlier than departing earliest with highest speed



We need to limit our outer polygon:

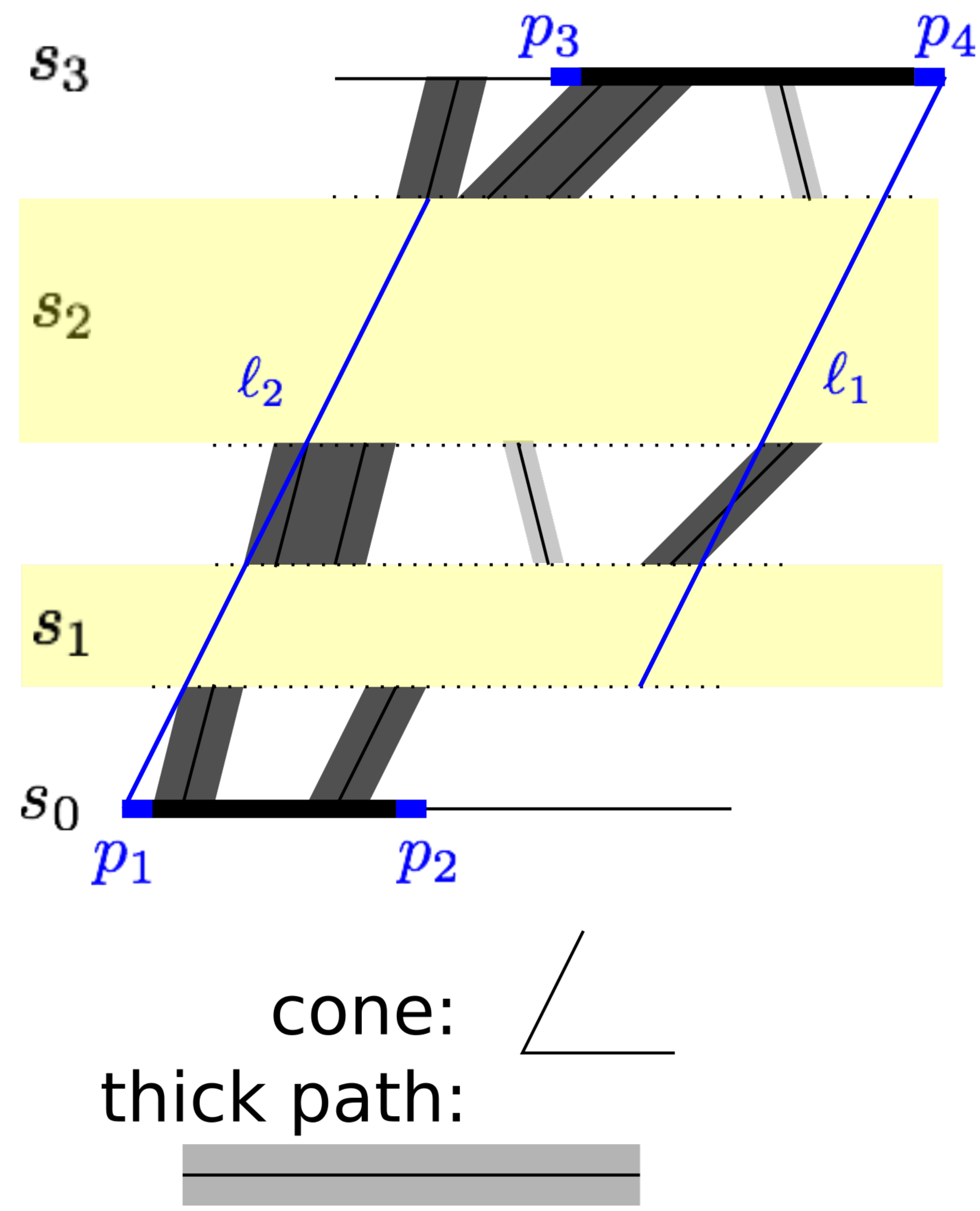
- No train can run earlier than departing earliest with highest speed

  ⇨ ℓ_2



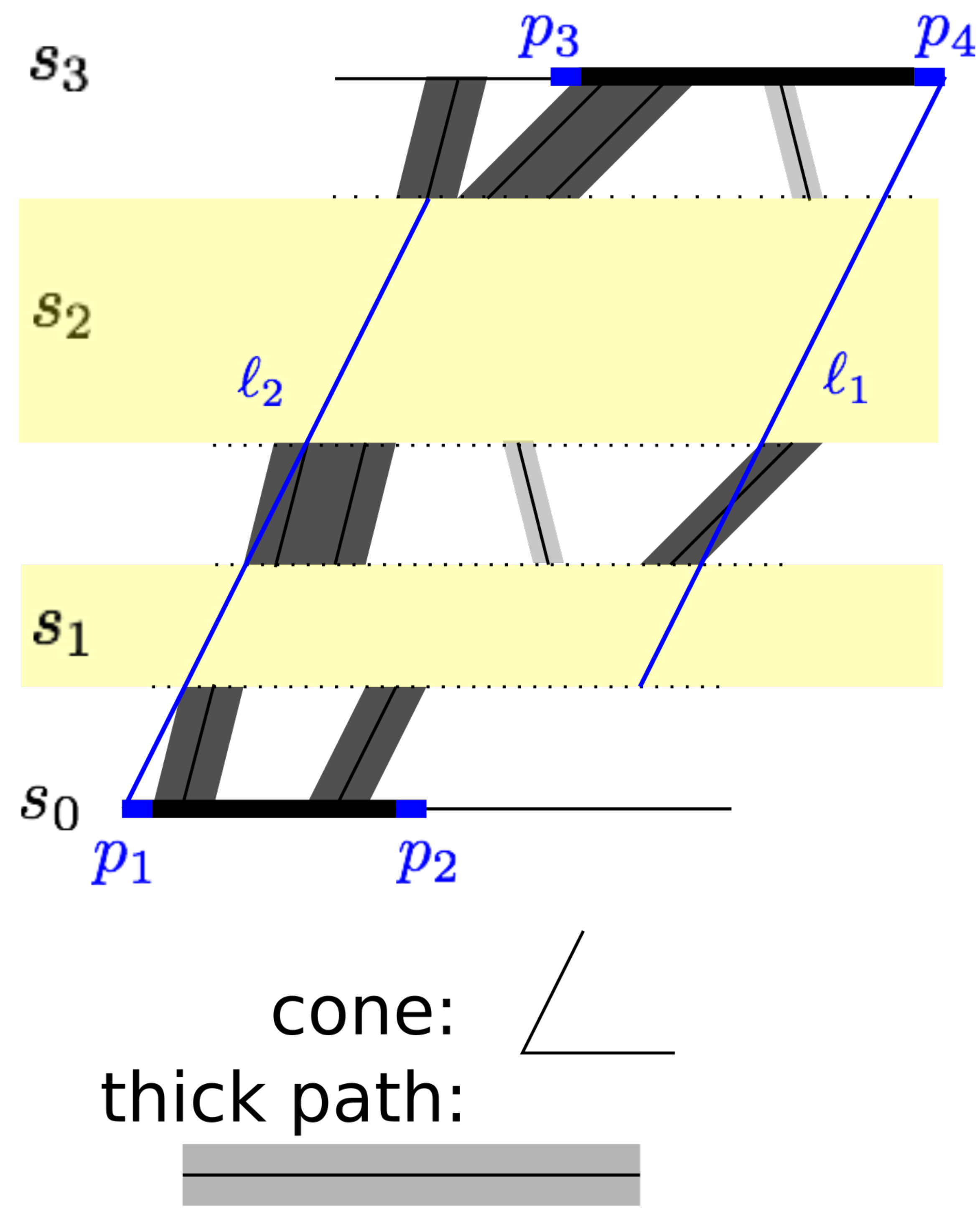
We need to limit our outer polygon:

- No train can run earlier than departing earliest with highest speed
  ⇨ ℓ_2
- No train can run later than arriving latest with highest speed



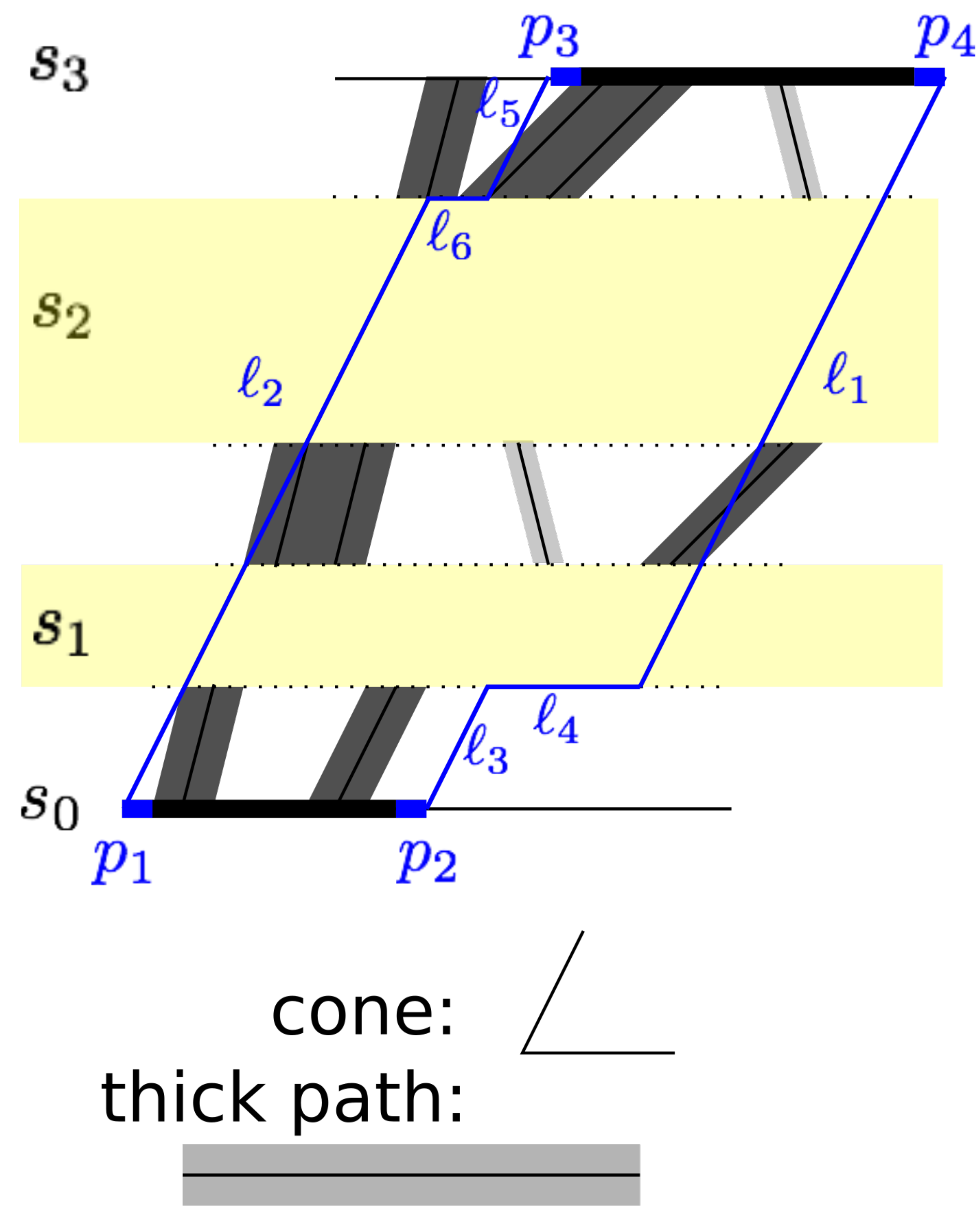
We need to limit our outer polygon:

- No train can run earlier than departing earliest with highest speed
  ⇨ ℓ_2
- No train can run later than arriving latest with highest speed
  ⇨ ℓ_1



We need to limit our outer polygon:

- No train can run earlier than departing earliest with highest speed
  ⇨ ℓ_2
- No train can run later than arriving latest with highest speed
  ⇨ ℓ_1
- Some further boundary parts

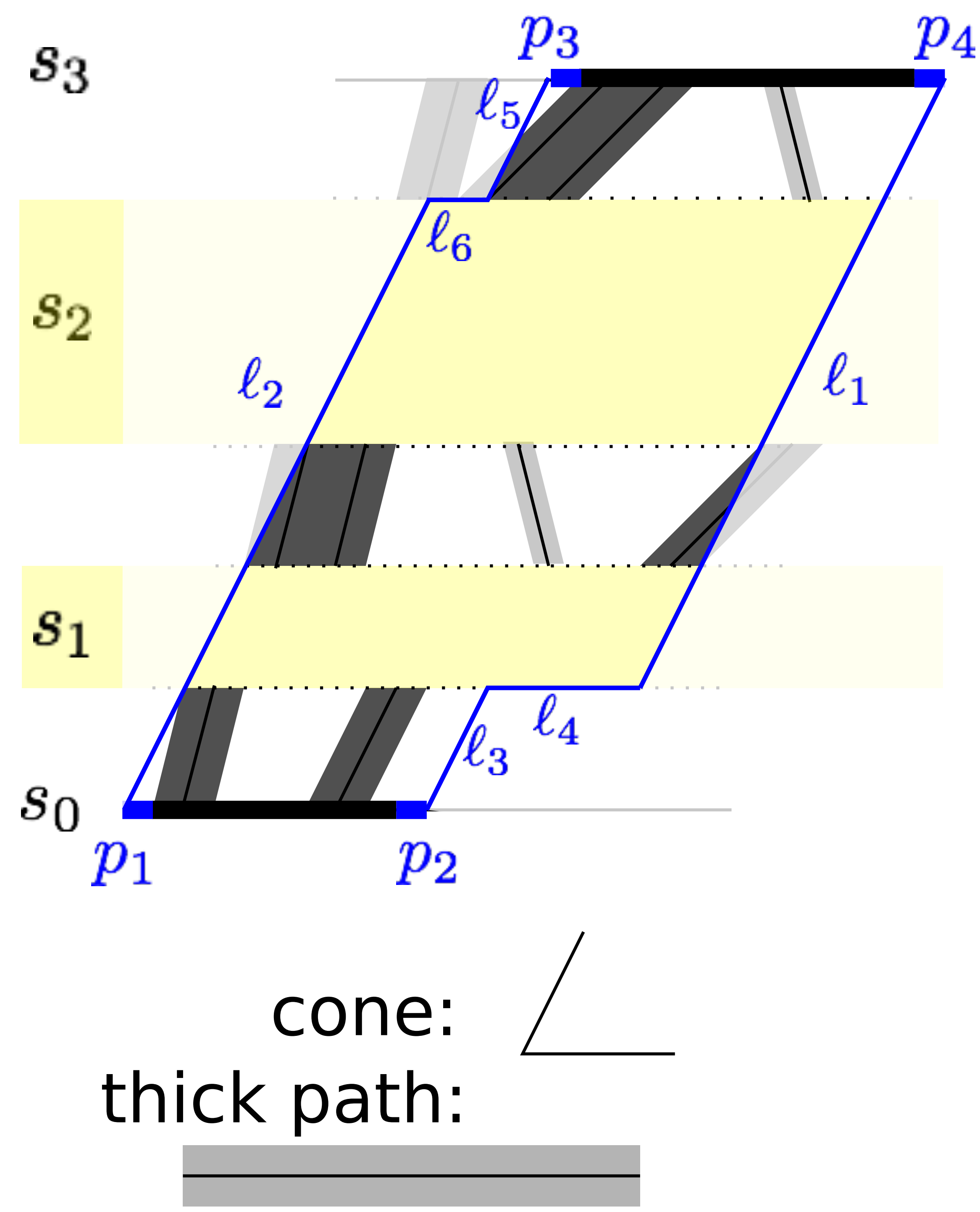


We need to limit our outer polygon:

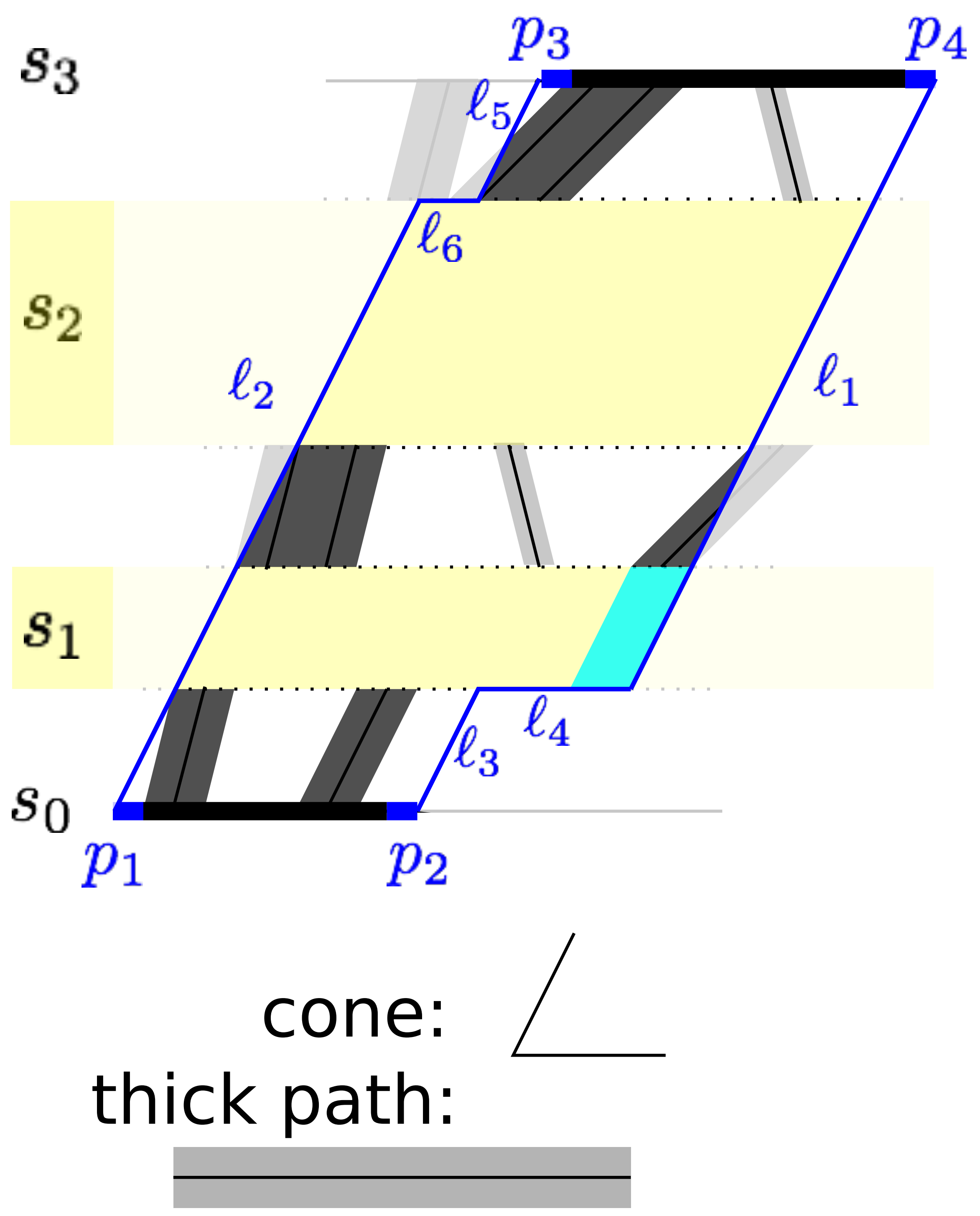
- No train can run earlier than departing earliest with highest speed

  ⇨ ℓ_2
- No train can run later than arriving latest with highest speed

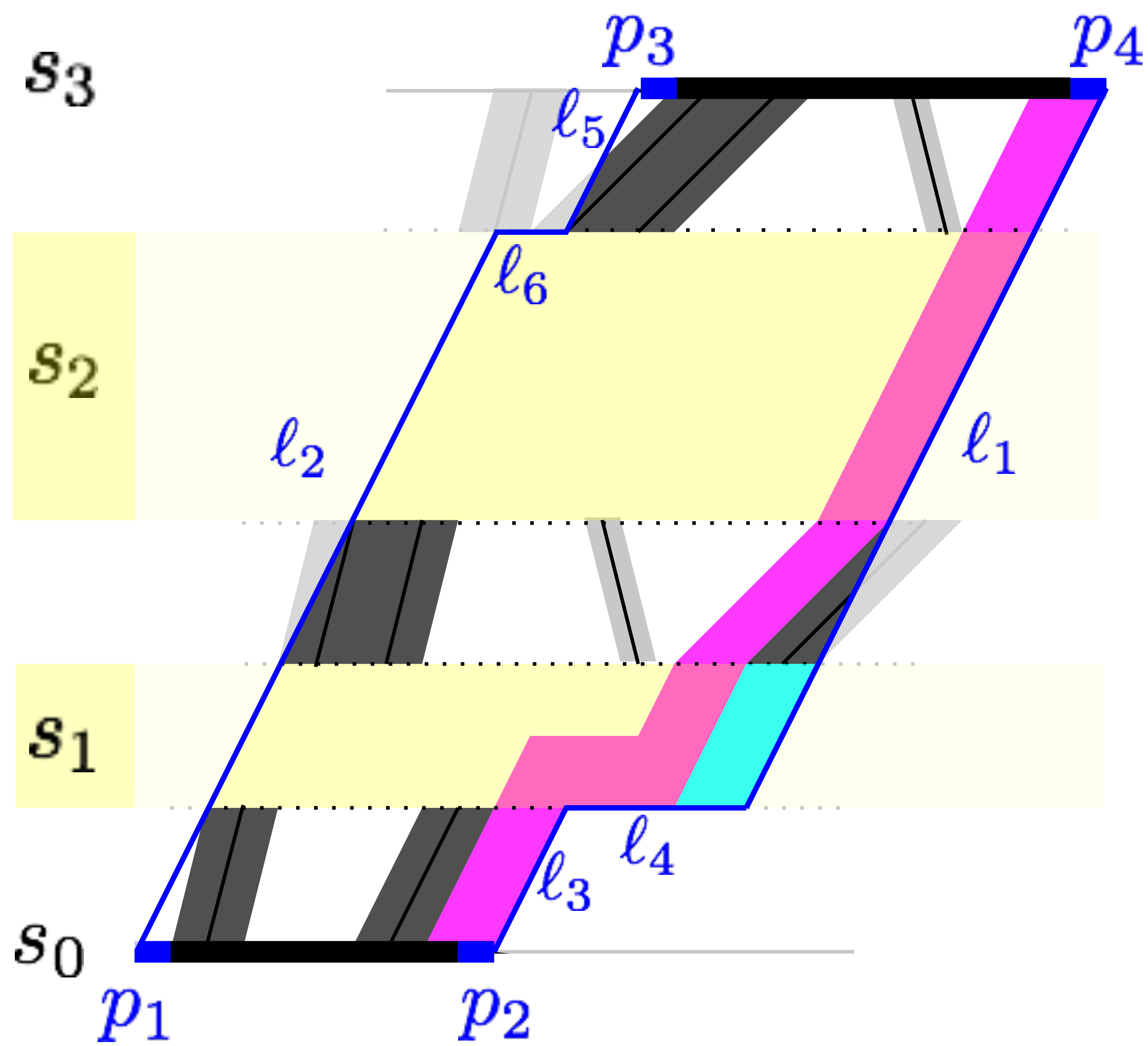
  ⇨ ℓ_1
- Some further boundary parts
- Intersect holes with boundary



Example

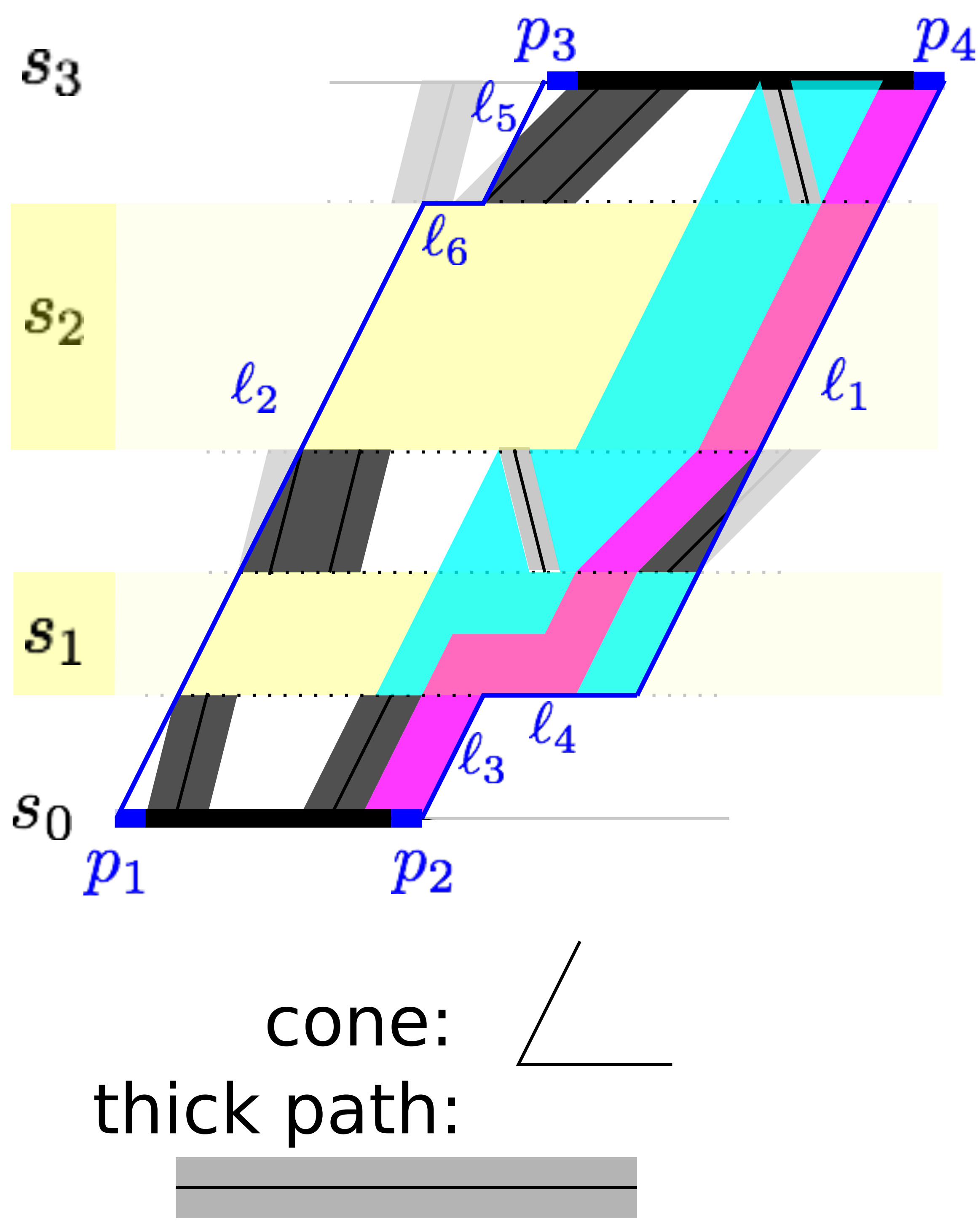


s_3
p_3
p_4
ℓ_5
ℓ_6
s_2
ℓ_2
ℓ_1
s_1
ℓ_4
ℓ_3
s_0
p_1
p_2
cone:
thick path:

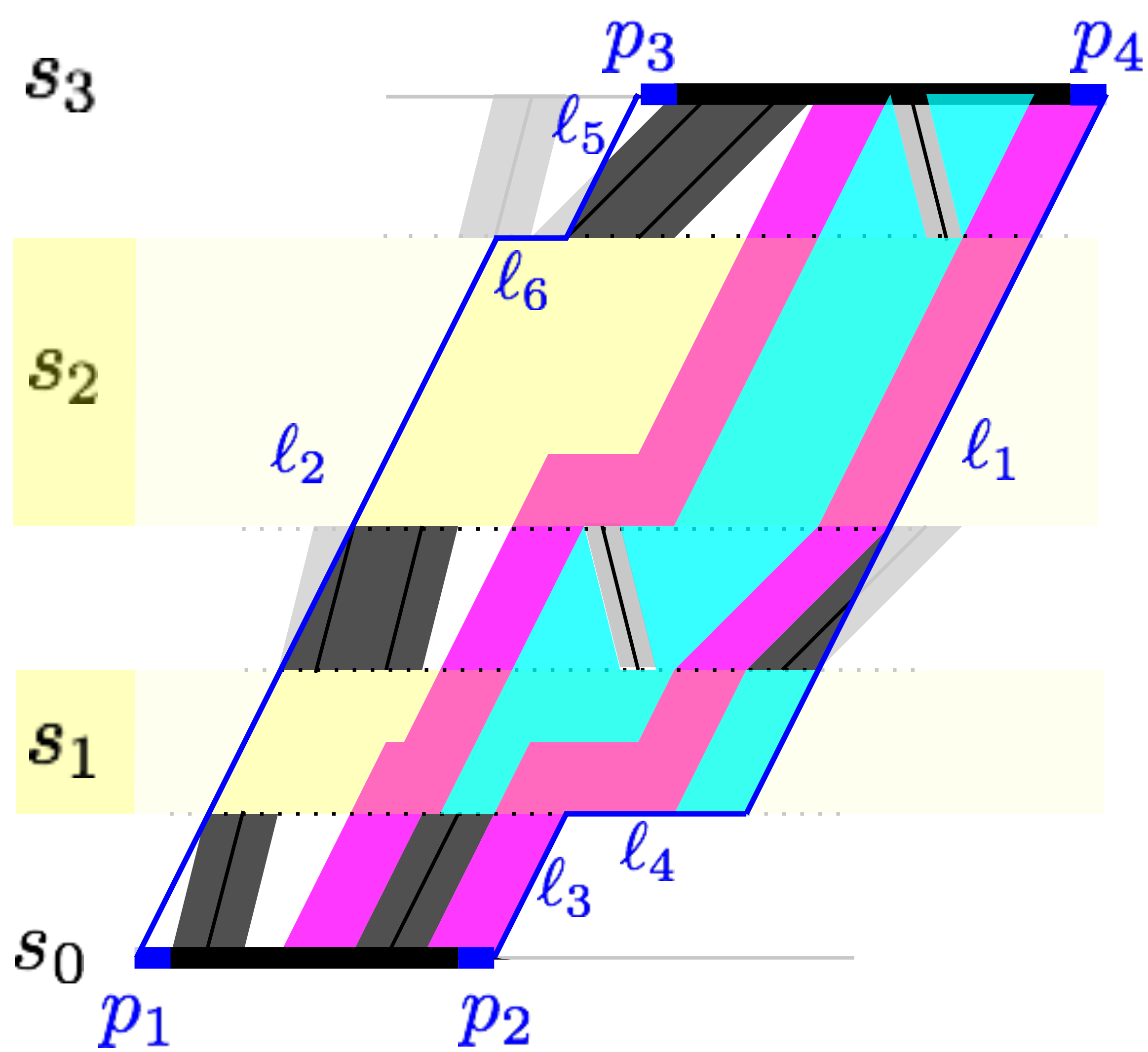


cone:

thick path:

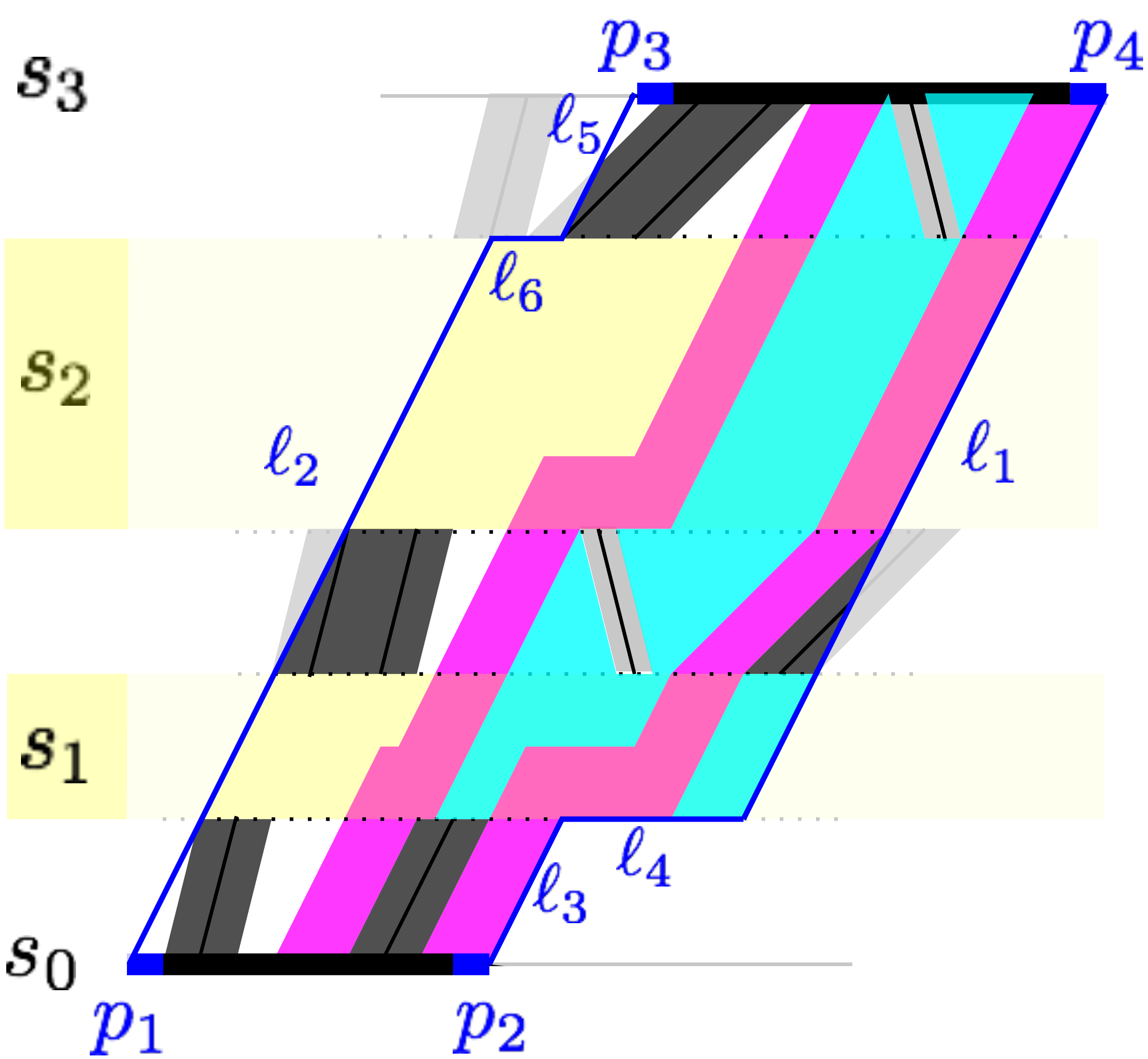


s_3
p_3
p_4
ℓ_5
ℓ_6
s_2
ℓ_2
ℓ_1
s_1
ℓ_4
ℓ_3
s_0
p_1
p_2
cone:
thick path:



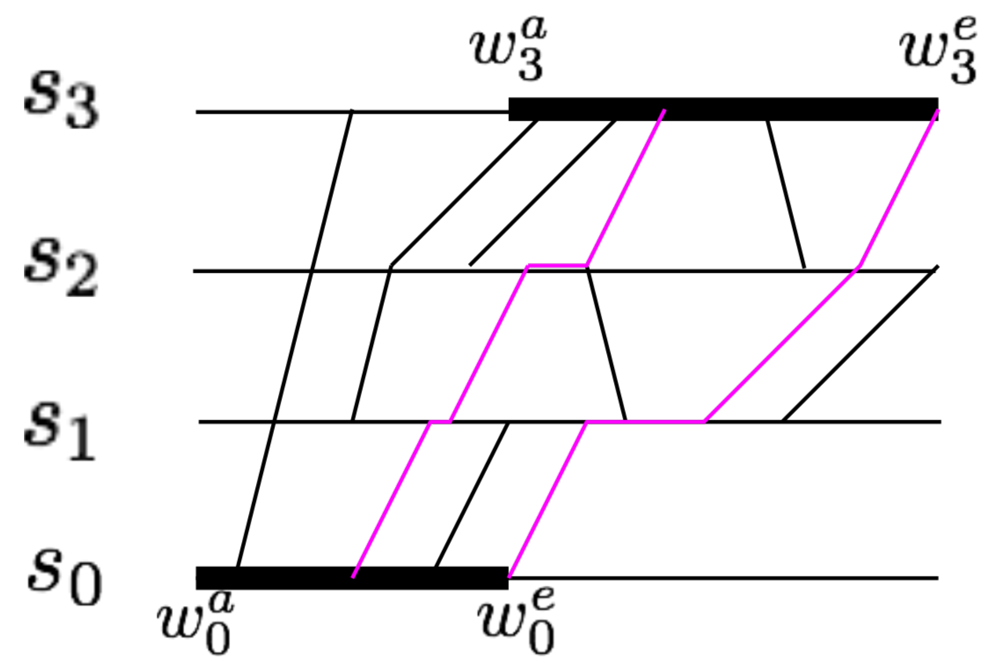
cone:

thick path:



cone:

thick path:



cone:

thick path:

Conclusion and Outlook

- Paths of Different Thickness (different temporal buffers required):

- Paths of Different Thickness (different temporal buffers required):
  - NP-hard in general

- Paths of Different Thickness (different temporal buffers required):
  - NP-hard in general
  - Same algorithm if the order of paths, that is, the order of trains is given

- Paths of Different Thickness (different temporal buffers required):
  - NP-hard in general
  - Same algorithm if the order of paths, that is, the order of trains is given
- Paths with Different Cones (different train types)

- Paths of Different Thickness (different temporal buffers required):
  - NP-hard in general
  - Same algorithm if the order of paths, that is, the order of trains is given
- Paths with Different Cones (different train types)
  - Again possible with the algorithm if the order of paths/order of trains is given: We simply make the new bottom respecting each consecutive cone

- Paths of Different Thickness (different temporal buffers required):
  - NP-hard in general
  - Same algorithm if the order of paths, that is, the order of trains is given
- Paths with Different Cones (different train types)
  - Again possible with the algorithm if the order of paths/order of trains is given: We simply make the new bottom respecting each consecutive cone

Outlook

- Paths of Different Thickness (different temporal buffers required):
  - NP-hard in general
  - Same algorithm if the order of paths, that is, the order of trains is given
- Paths with Different Cones (different train types)
  - Again possible with the algorithm if the order of paths/order of trains is given: We simply make the new bottom respecting each consecutive cone

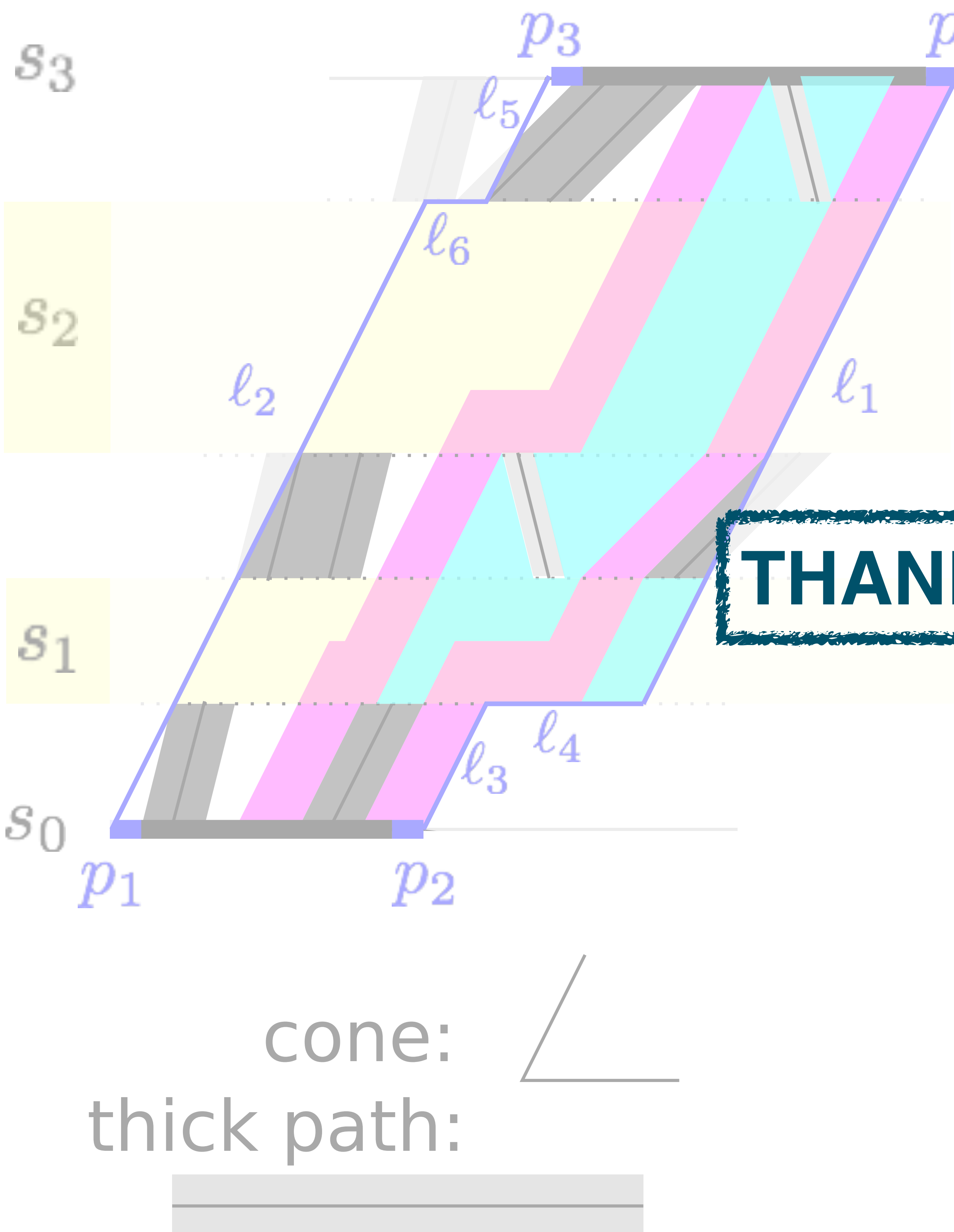
Outlook

- Application to real-world example

- Paths of Different Thickness (different temporal buffers required):
  - NP-hard in general
  - Same algorithm if the order of paths, that is, the order of trains is given
- Paths with Different Cones (different train types)
  - Again possible with the algorithm if the order of paths/order of trains is given: We simply make the new bottom respecting each consecutive cone

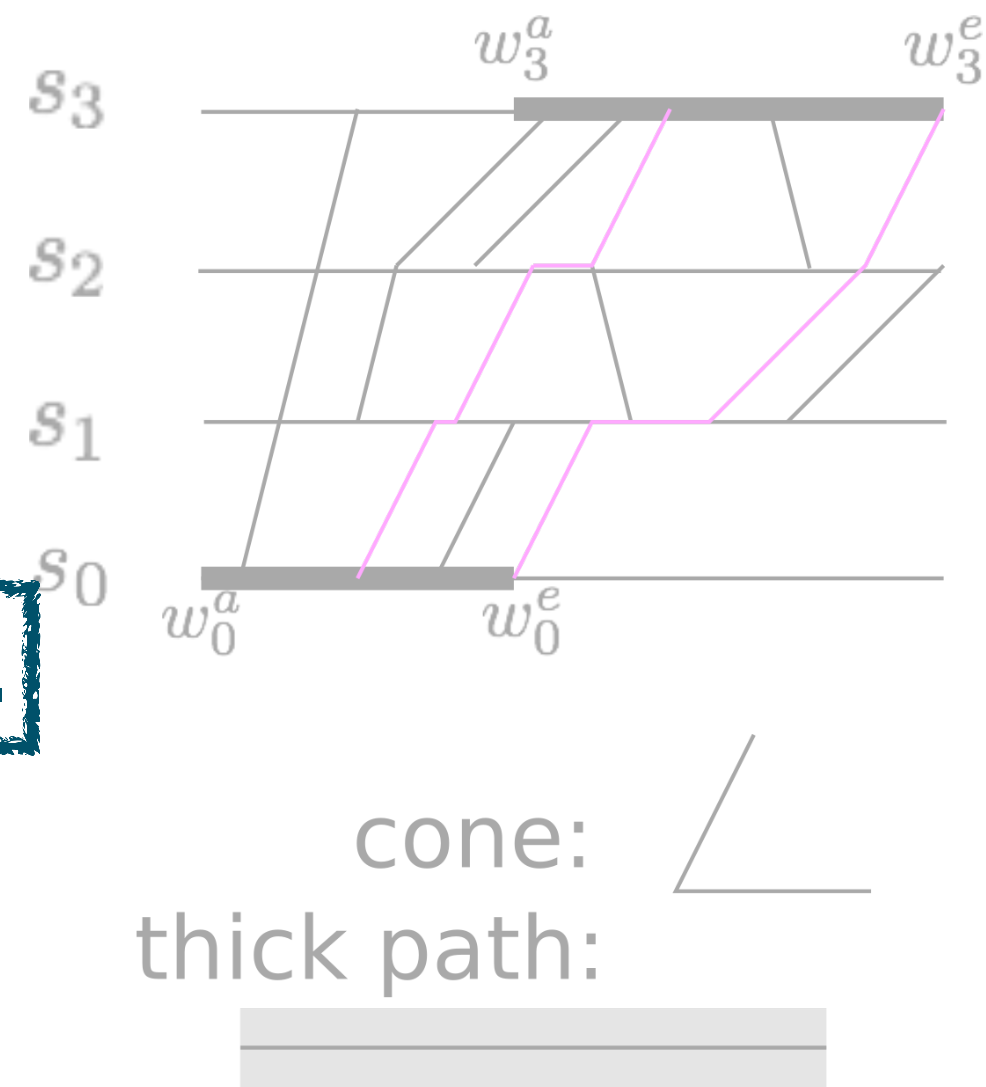
Outlook

- Application to real-world example
- What other geometric concepts can be used?



s_3
p_3
p_4
ℓ_5
ℓ_6
s_2
ℓ_2
ℓ_1
s_1
ℓ_3
ℓ_4
s_0
p_1
p_2
cone:
thick path:

THANKS.



w_3^a
w_3^e
s_3
s_2
s_1
s_0
w_0^a
w_0^e
cone:
thick path: