Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

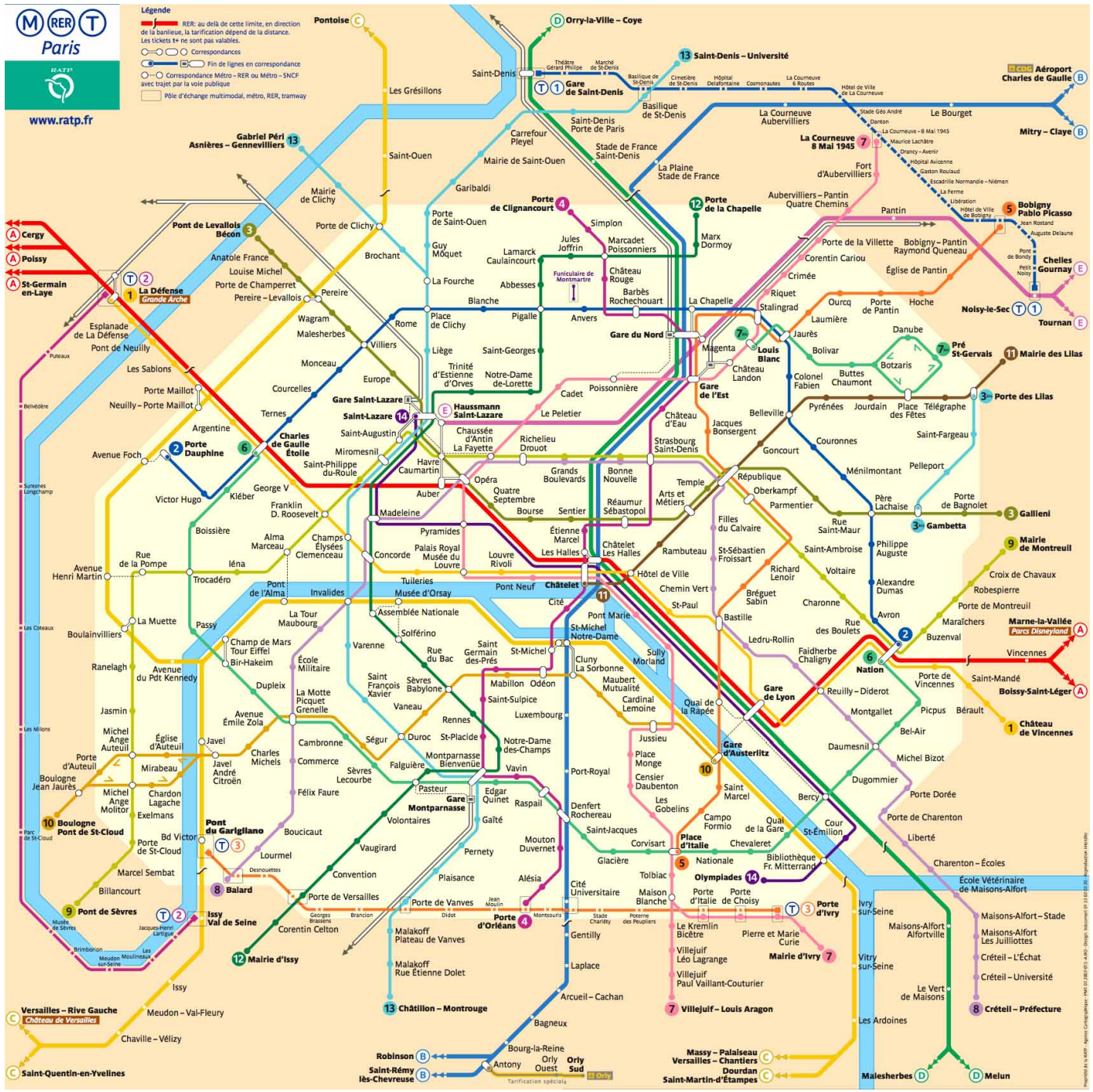
Ordering Metro Lines by Block Crossings

Martin Fink
Lehrstuhl für Informatik I
Universität Würzburg

Joint work with Sergey Pupyrev

Metro Maps – Vienna

Metro Maps – Paris



Metro Maps – Metro Lines

Metro Maps – Metro Lines

- Previous work, e.g. [Nöllenburg and Wolff, 2011]

Metro Maps – Metro Lines

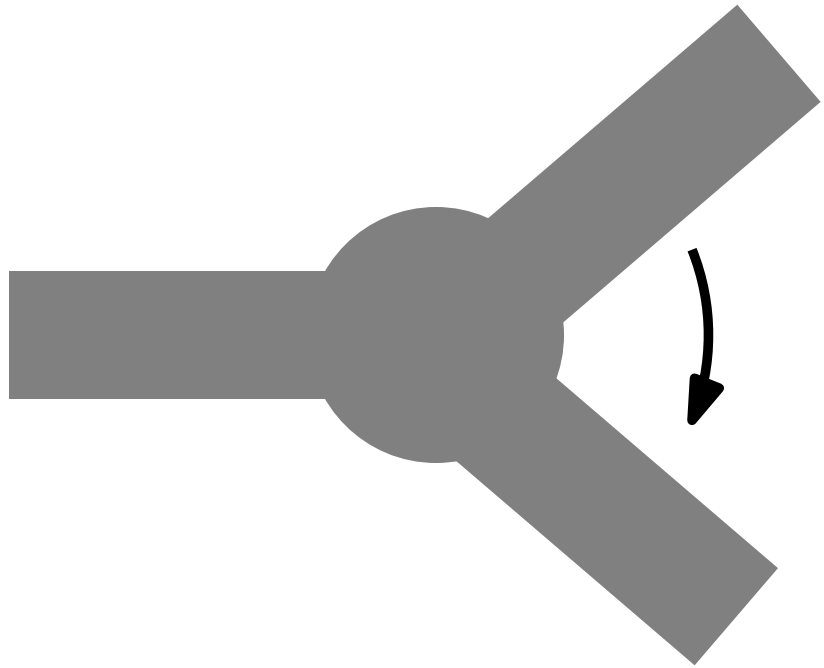
- Previous work, e.g. [Nöllenburg and Wolff, 2011]
- Focus on drawing underlying graph

Metro Line Crossing Minimization

Insert all lines L into *embedded graph* $G = (V, E)$ such that ...

Metro Line Crossing Minimization

Insert all lines L into *embedded graph* $G = (V, E)$ such that ...



Metro Line Crossing Minimization

Insert all lines L into *embedded graph* $G = (V, E)$ such that ...

- the number of crossings is minimized.

Metro Line Crossing Minimization

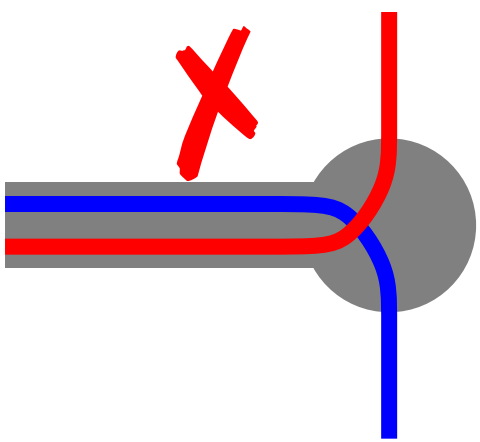
Insert all lines L into *embedded graph* $G = (V, E)$ such that ...

- the number of crossings is minimized.
- crossings are placed on edges if possible.

Metro Line Crossing Minimization

Insert all lines L into *embedded graph* $G = (V, E)$ such that ...

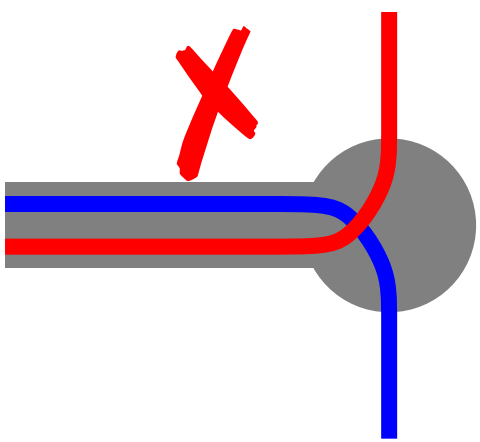
- the number of crossings is minimized.
- crossings are placed on edges if possible.

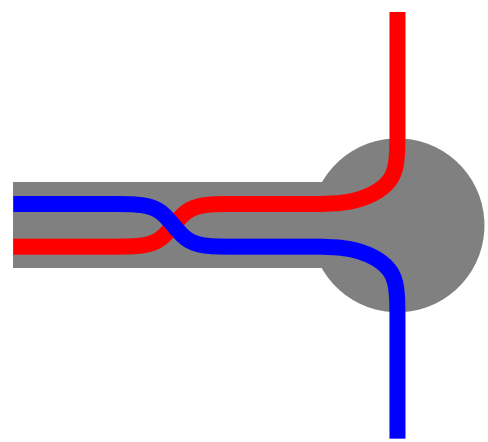


Metro Line Crossing Minimization

Insert all lines L into *embedded graph* $G = (V, E)$ such that ...

- the number of crossings is minimized.
- crossings are placed on edges if possible.

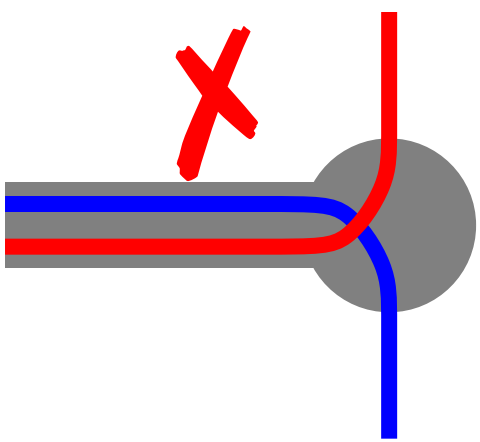


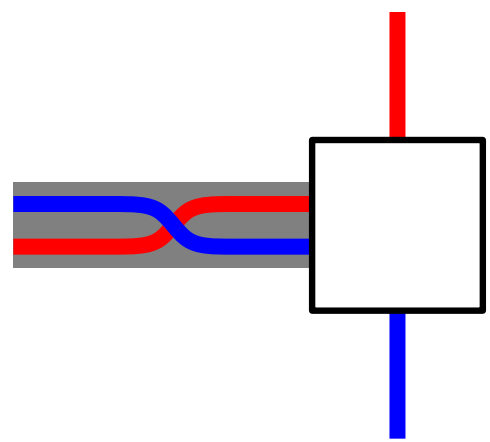


Metro Line Crossing Minimization

Insert all lines L into *embedded graph* $G = (V, E)$ such that ...

- the number of crossings is minimized.
- crossings are placed on edges if possible.

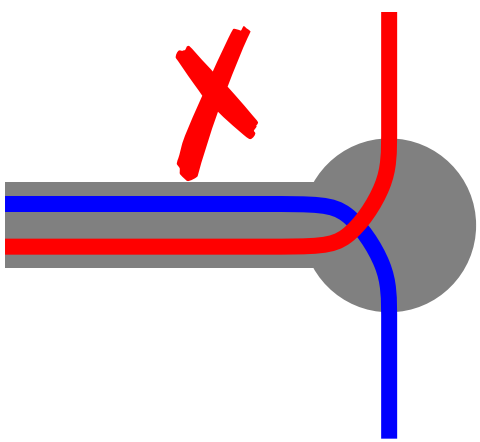


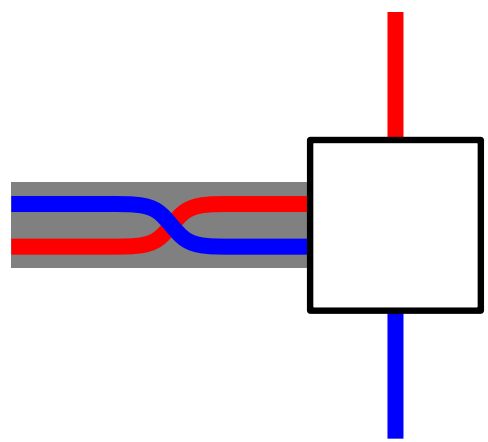


Metro Line Crossing Minimization

Insert all lines L into *embedded graph* $G = (V, E)$ such that ...

- the number of crossings is minimized.
- crossings are placed on edges if possible.





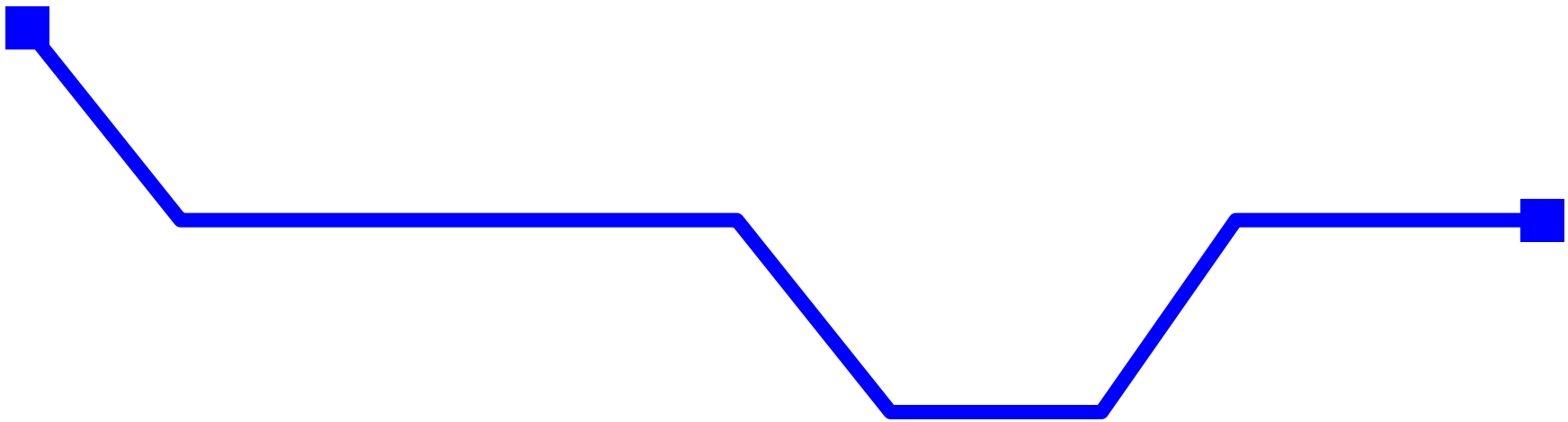
NP-hard [Bekos et al., 2007] & [Fink, Pupyrev, 2013]

Path Terminal Property

- Line terminals are leaves of the graph.

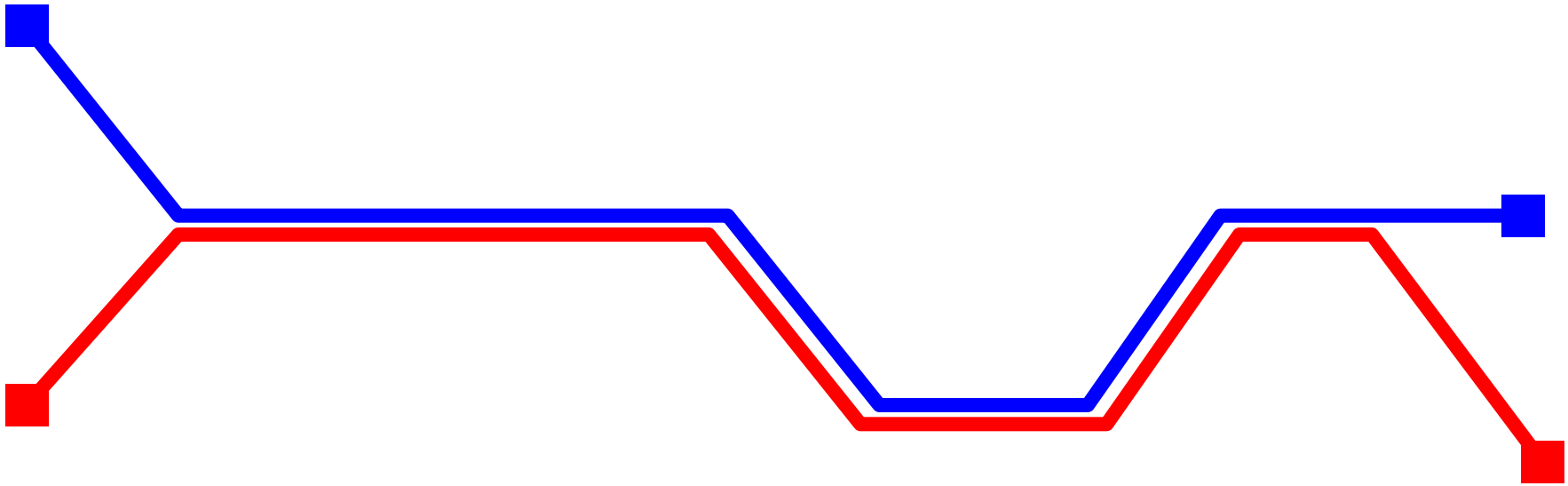
Path Terminal Property

- Line terminals are leaves of the graph.
- Follow two lines:



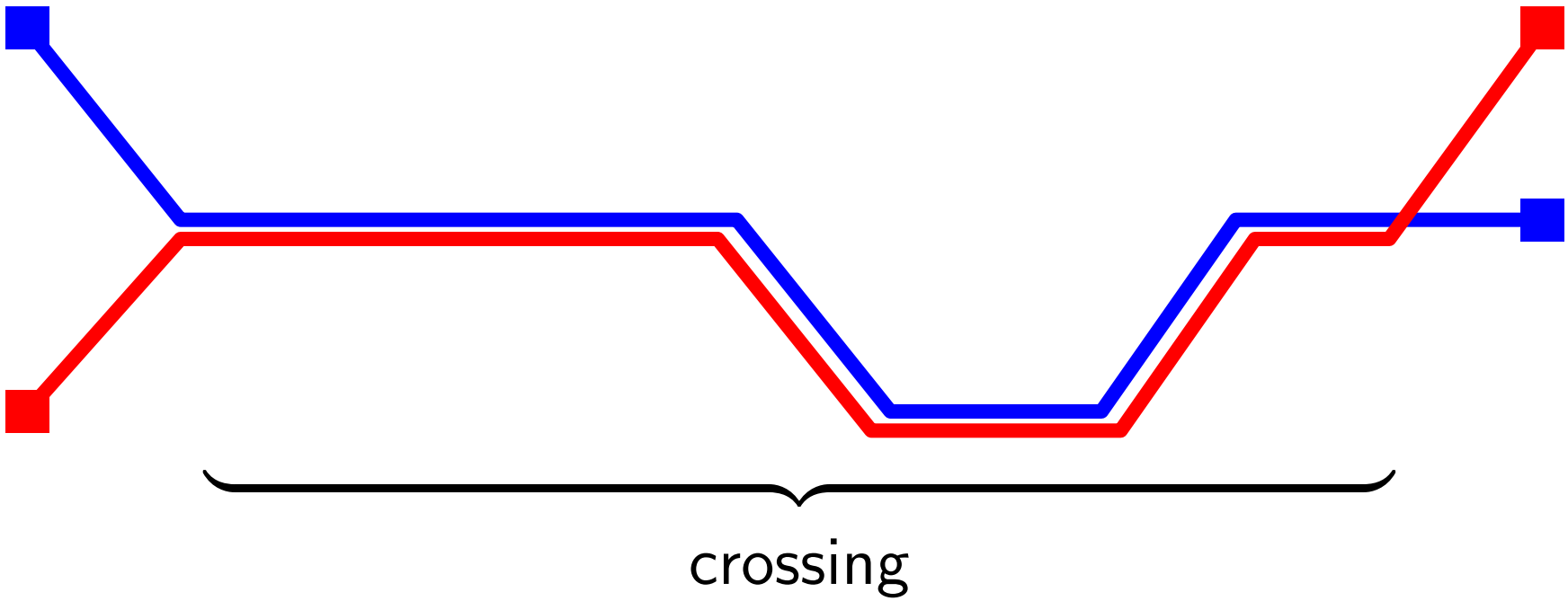
Path Terminal Property

- Line terminals are leaves of the graph.
- Follow two lines:



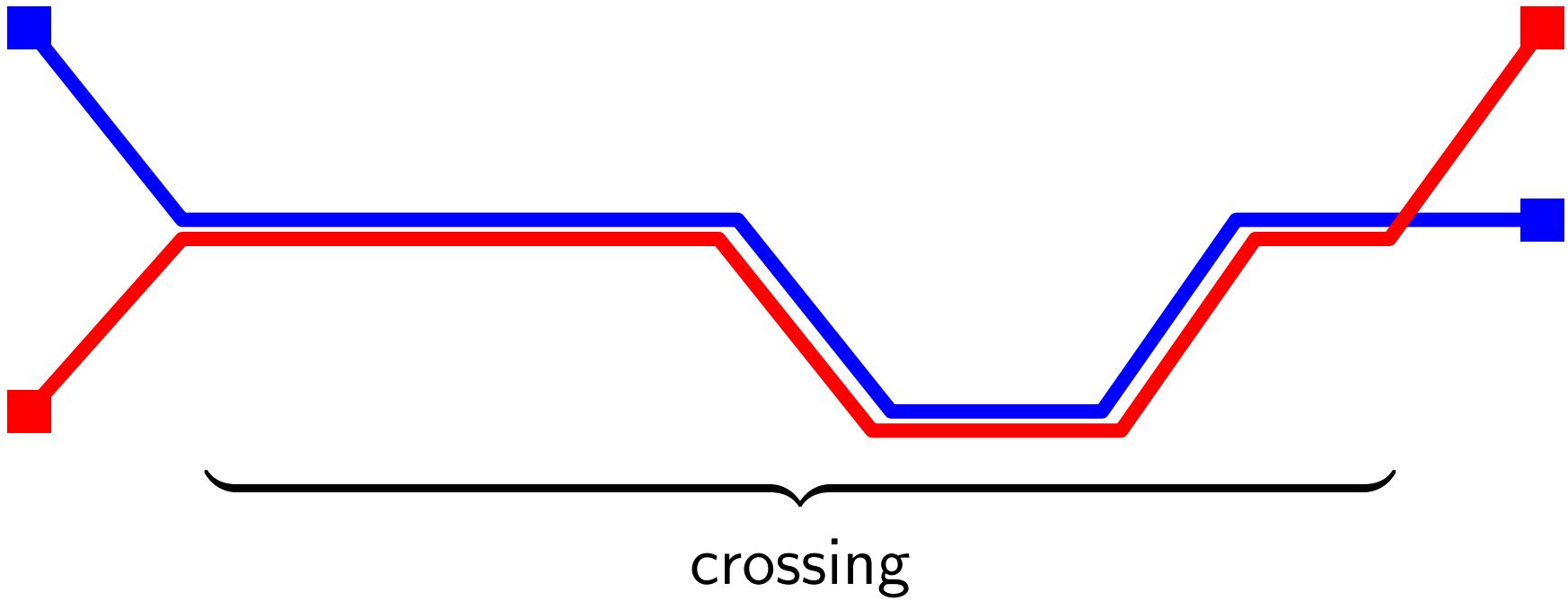
Path Terminal Property

- Line terminals are leaves of the graph.
- Follow two lines:



Path Terminal Property

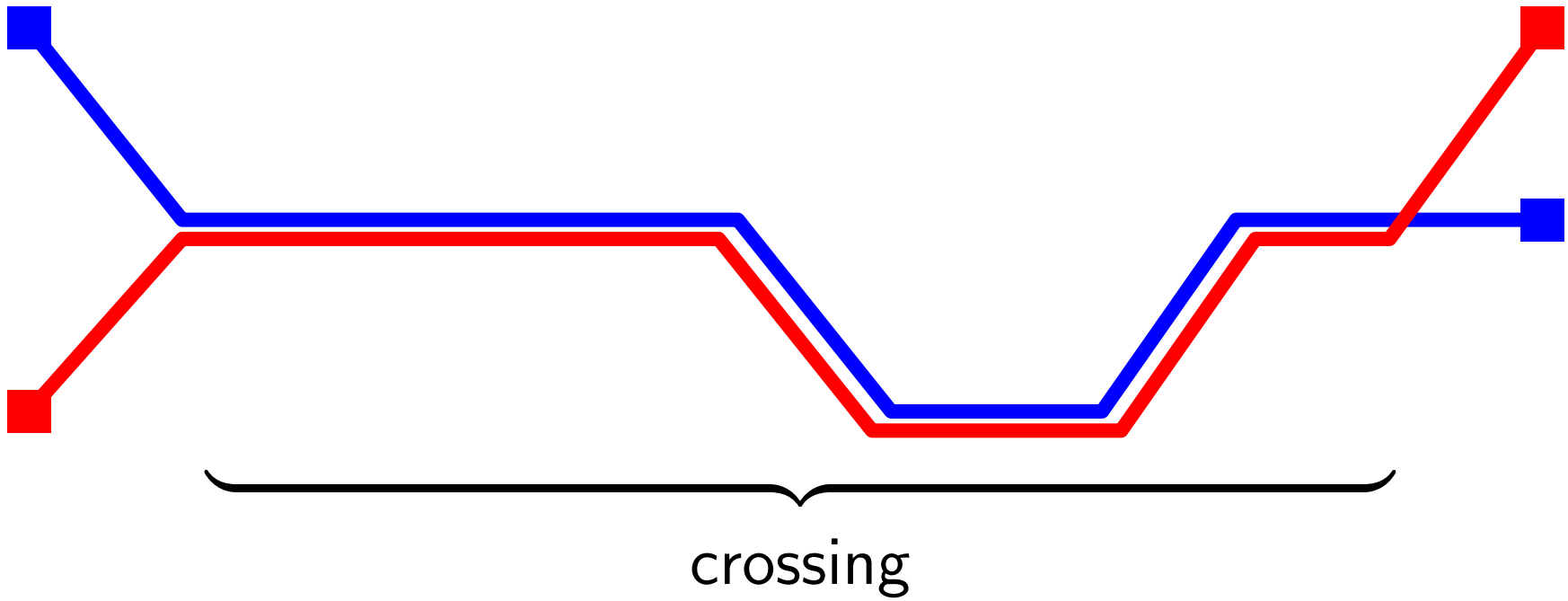
- Line terminals are leaves of the graph.
- Follow two lines:



- Crossing Minimization: Solvable in linear time [Pupyrev et al., 2012]

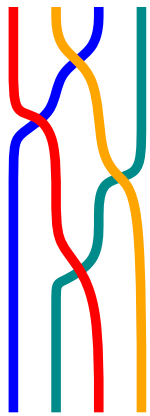
Path Terminal Property

- Line terminals are leaves of the graph.
- Follow two lines:



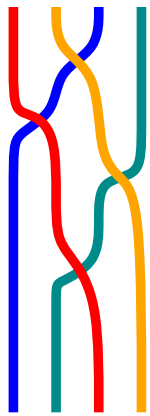
- Crossing Minimization: Solvable in linear time [Pupyrev et al., 2012]
- additional restriction: two lines intersect in a path

New Model: Block Crossings

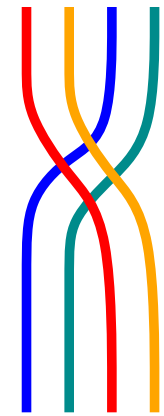


4 single crossings

New Model: Block Crossings

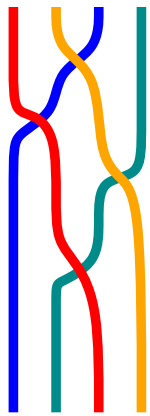


4 single crossings

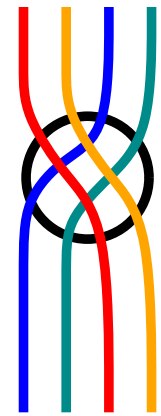


1 block crossing

New Model: Block Crossings

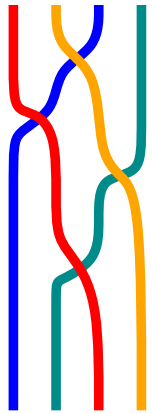


4 single crossings

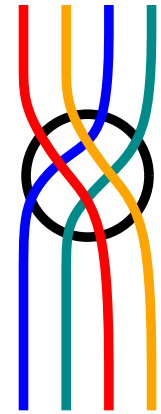


1 block crossing

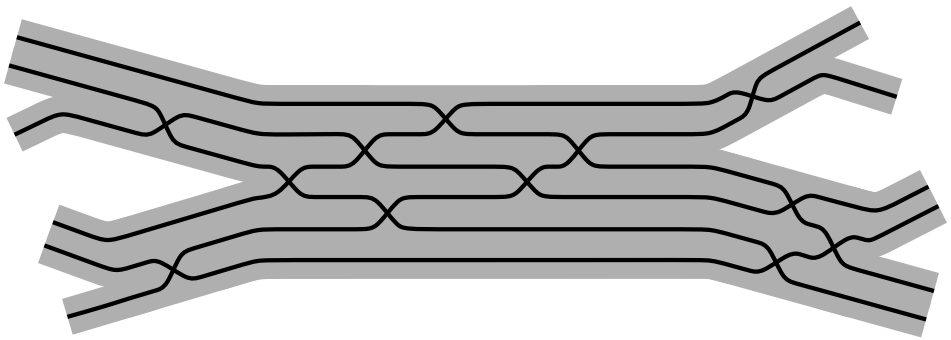
New Model: Block Crossings



4 single crossings

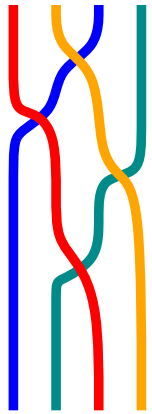


1 block crossing

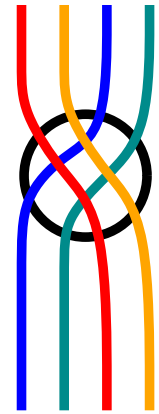


12 single crossings

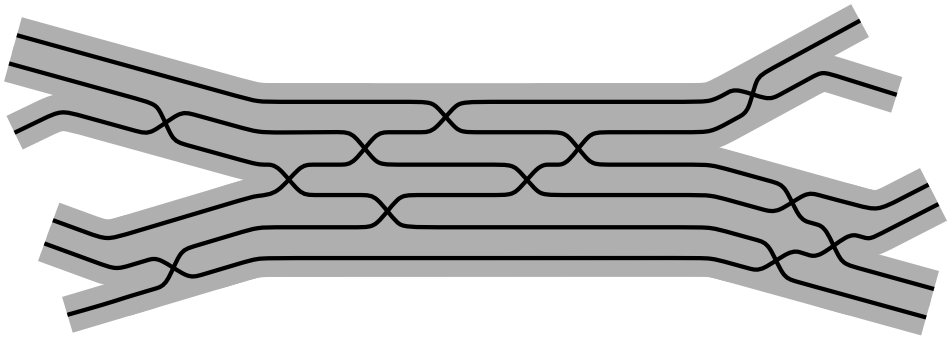
New Model: Block Crossings



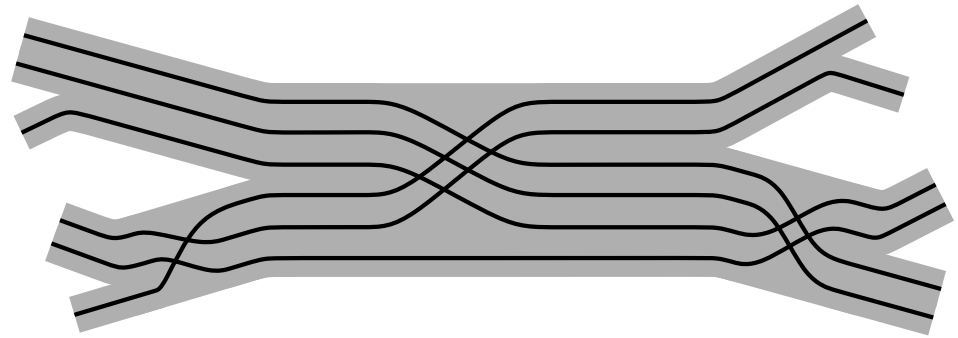
4 single crossings



1 block crossing



12 single crossings



3 block crossings

Block Crossing Minimization

New problem variants:

- **BCM**: Minimize the number of block crossings

Block Crossing Minimization

New problem variants:

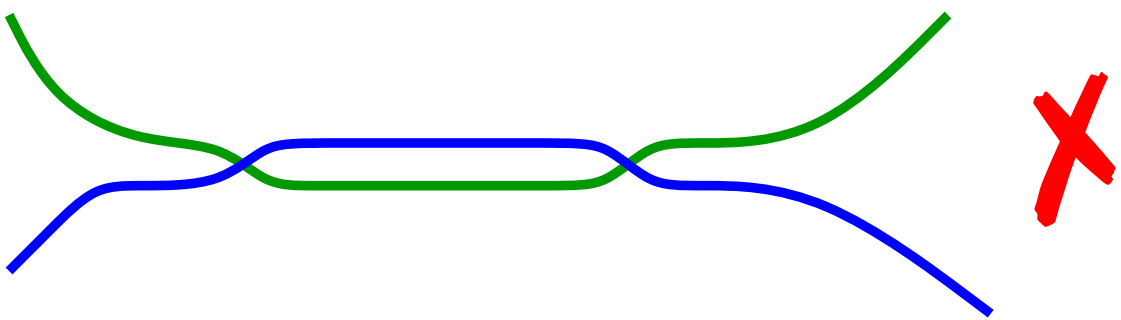
- **BCM**: Minimize the number of block crossings
- **MBCM**: Minimize the number of *monotone* block crossings

Block Crossing Minimization

New problem variants:

- **BCM**: Minimize the number of block crossings
- **MBCM**: Minimize the number of *monotone* block crossings

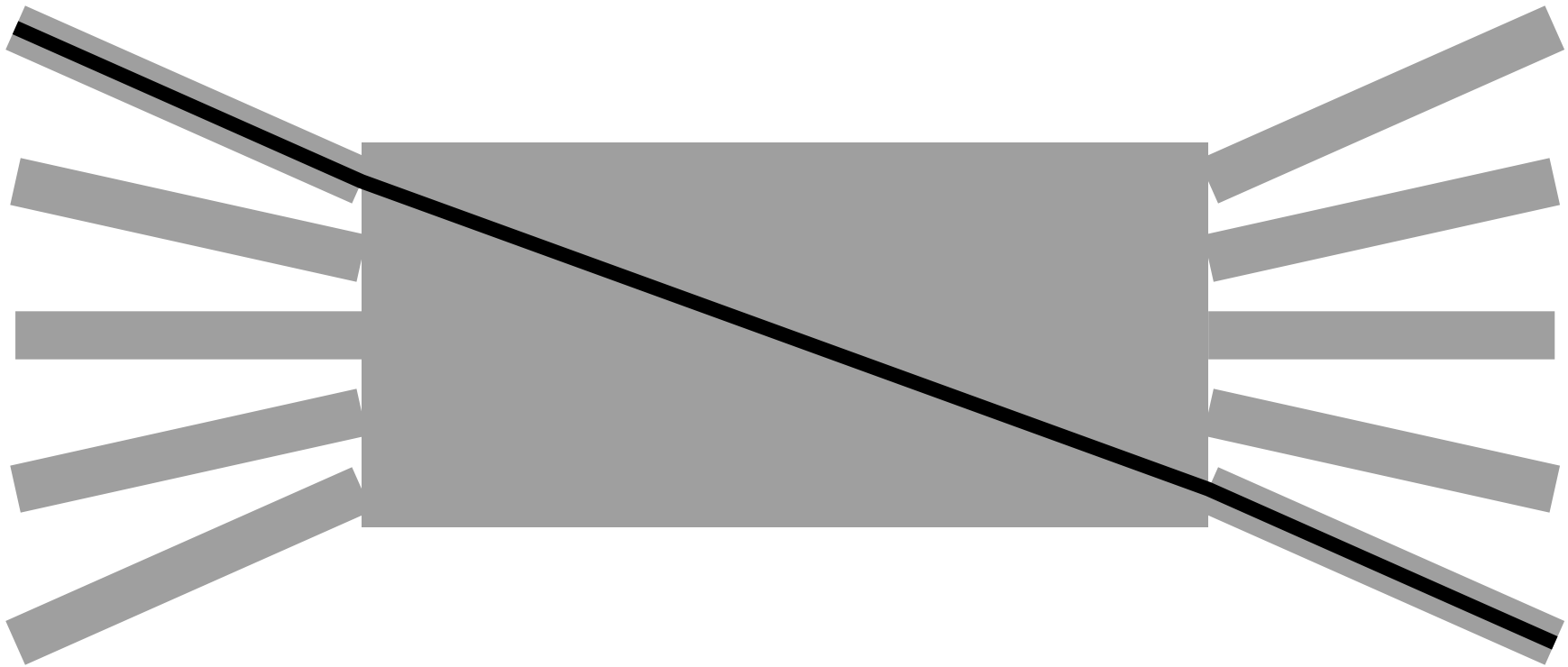
no double crossings!



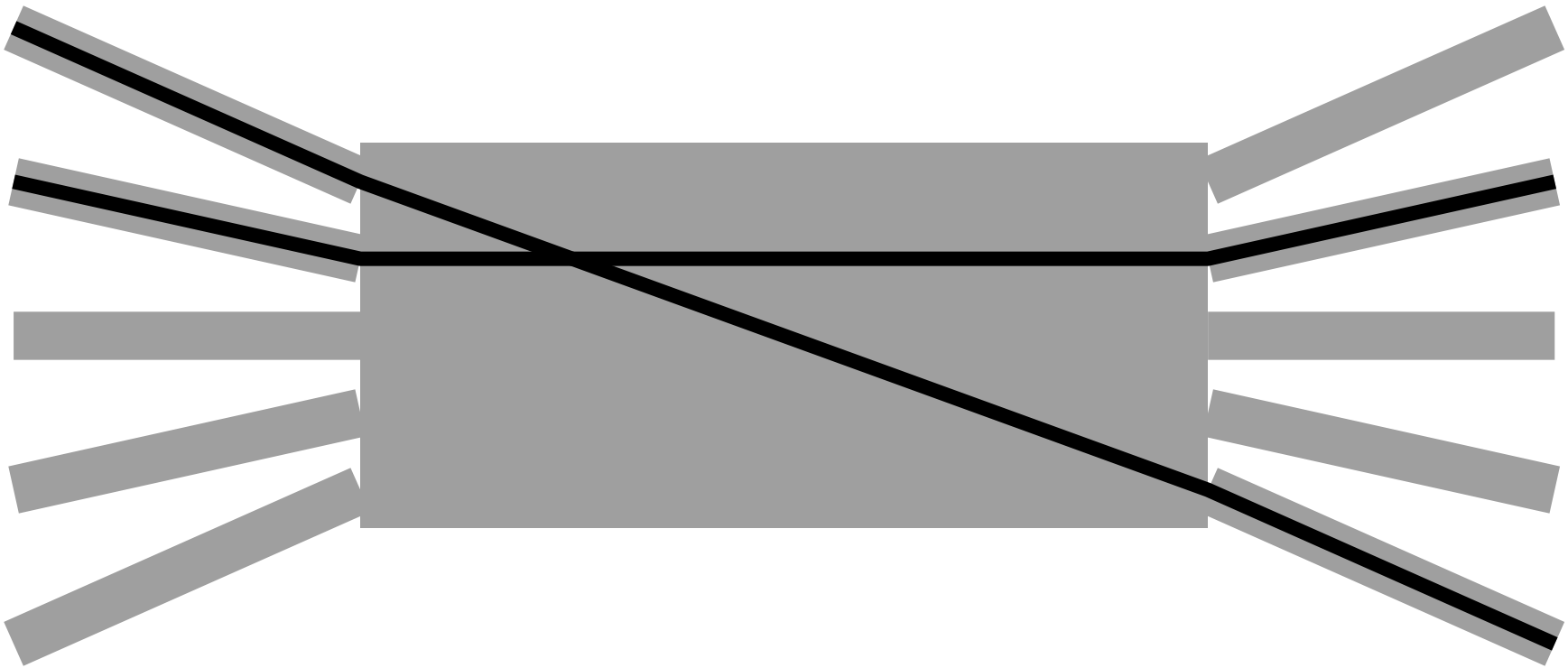
Single Edge

Single Edge

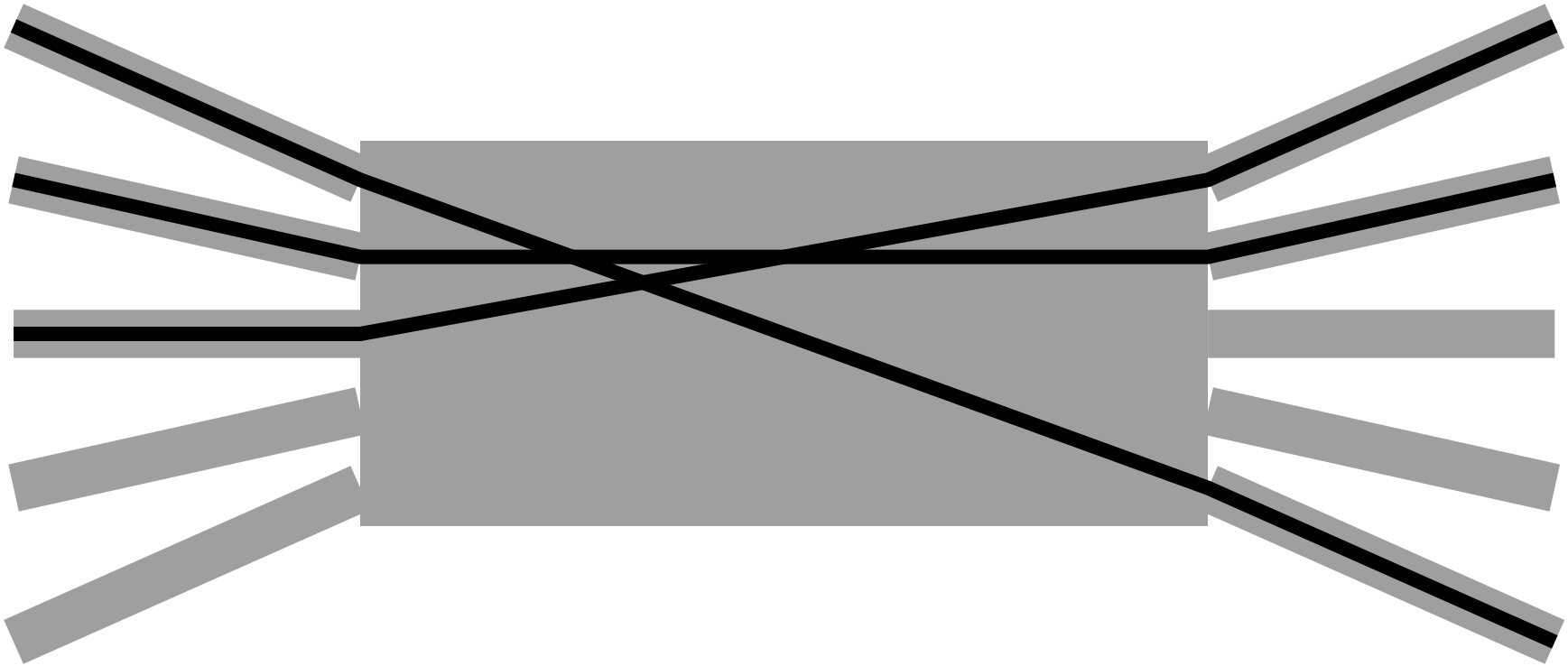
Single Edge



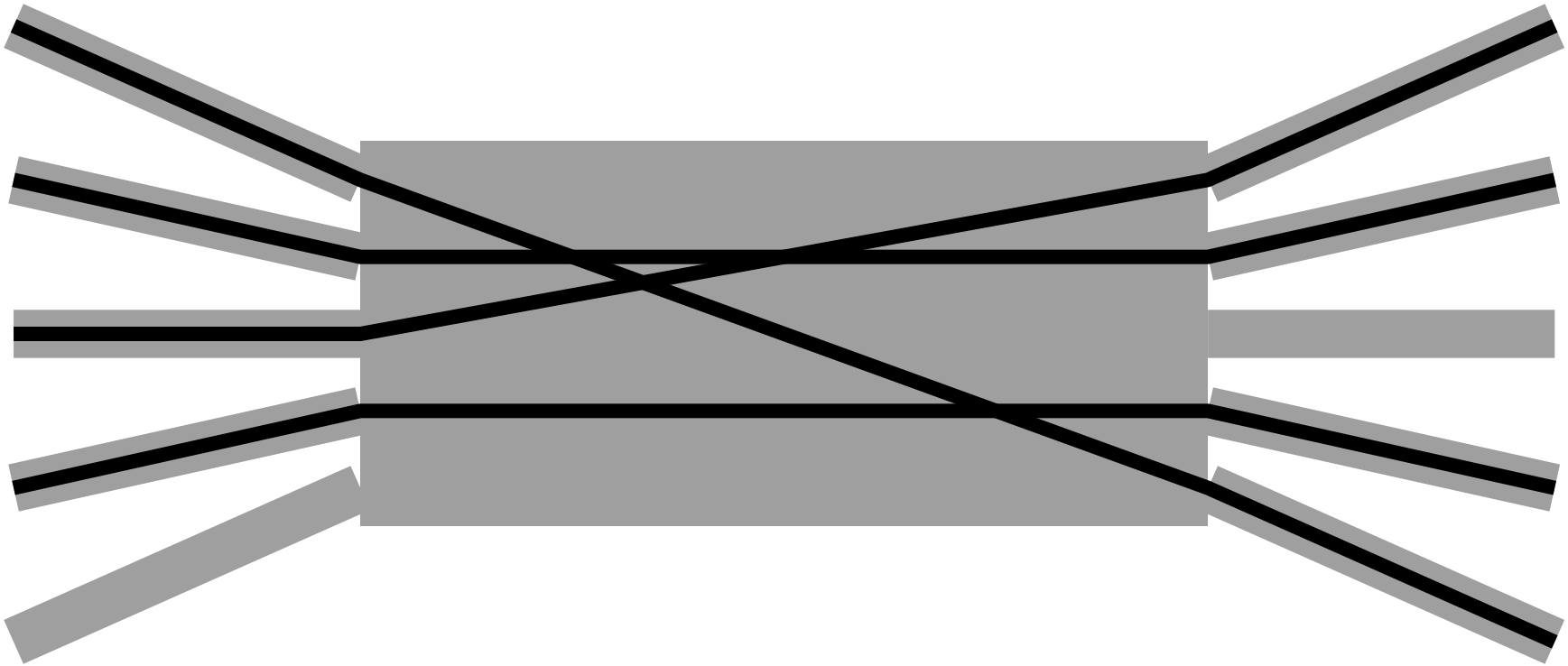
Single Edge



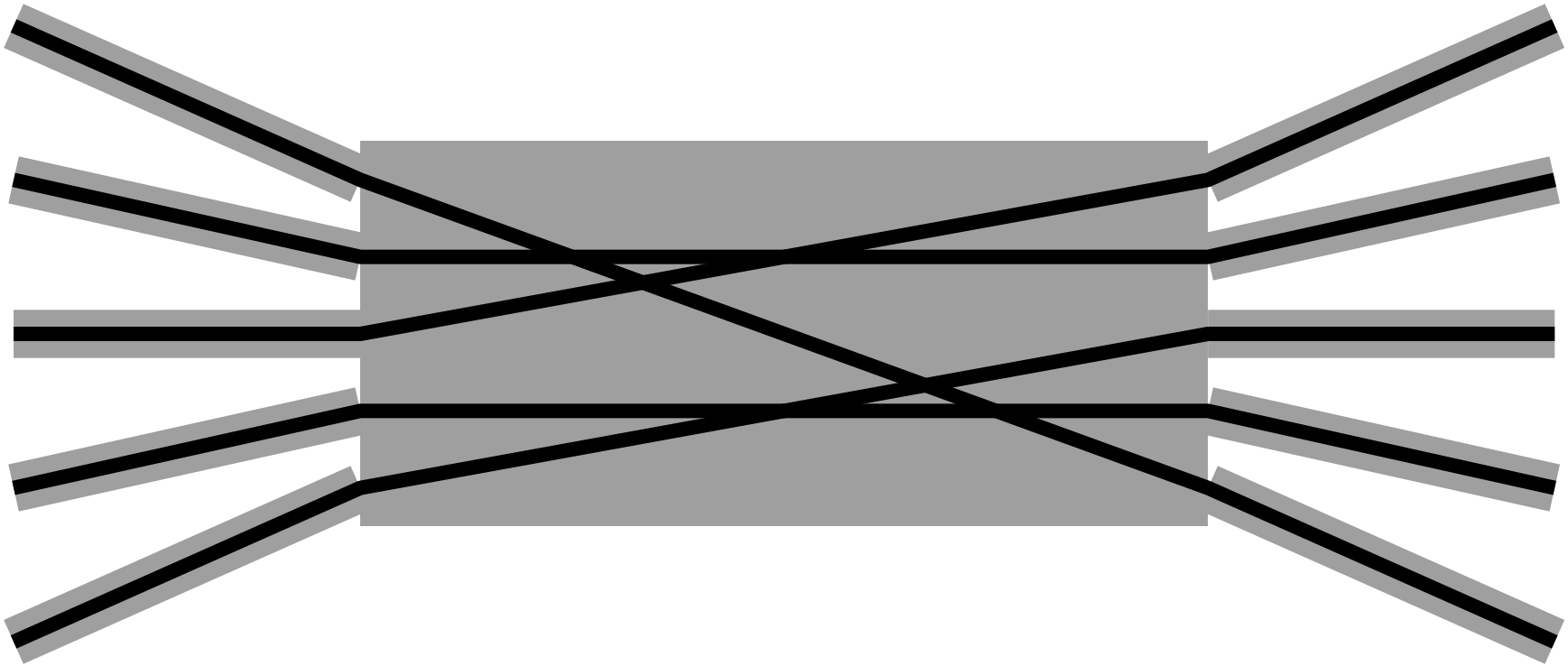
Single Edge



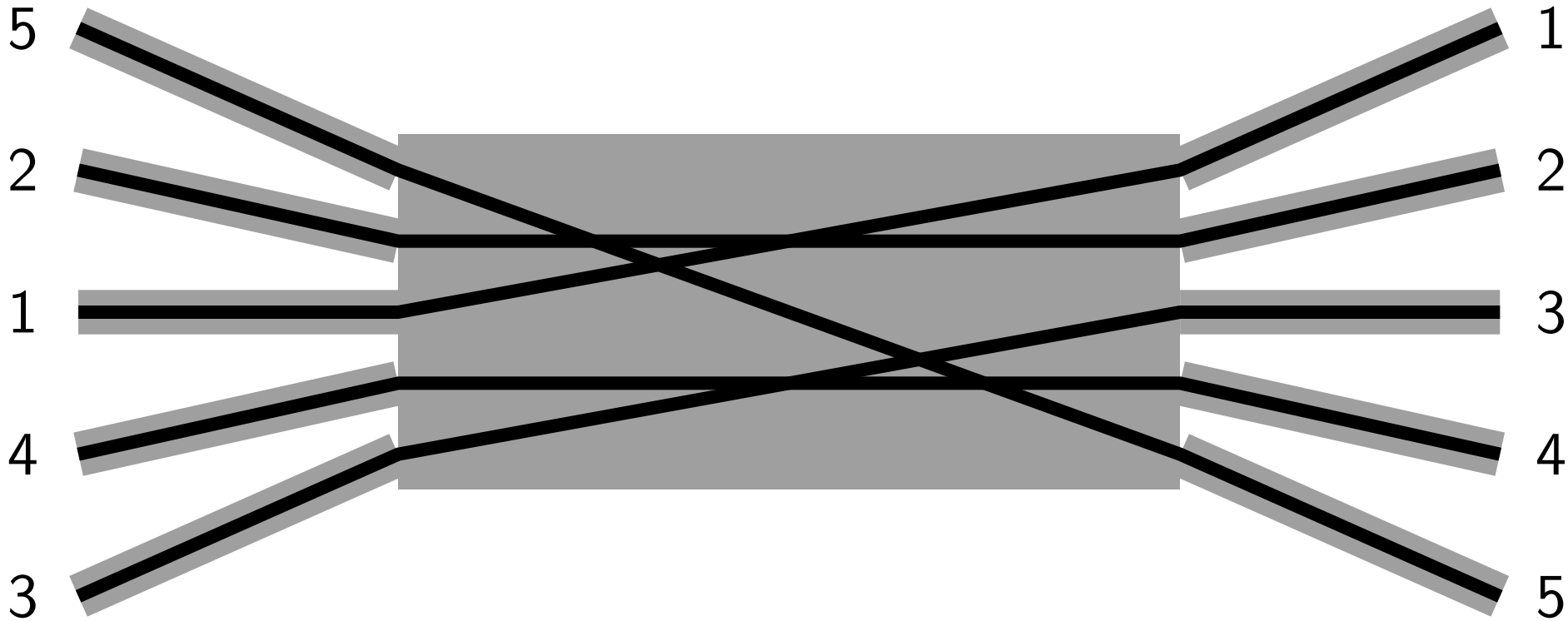
Single Edge



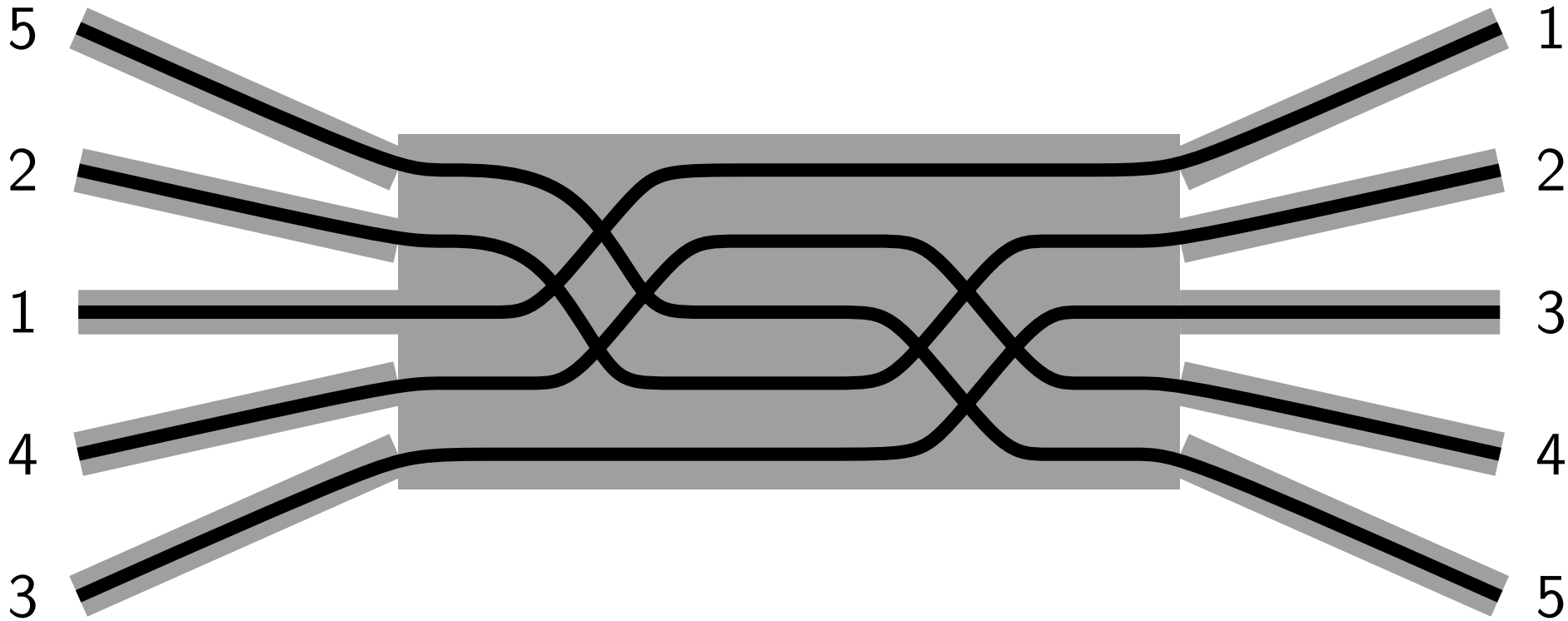
Single Edge



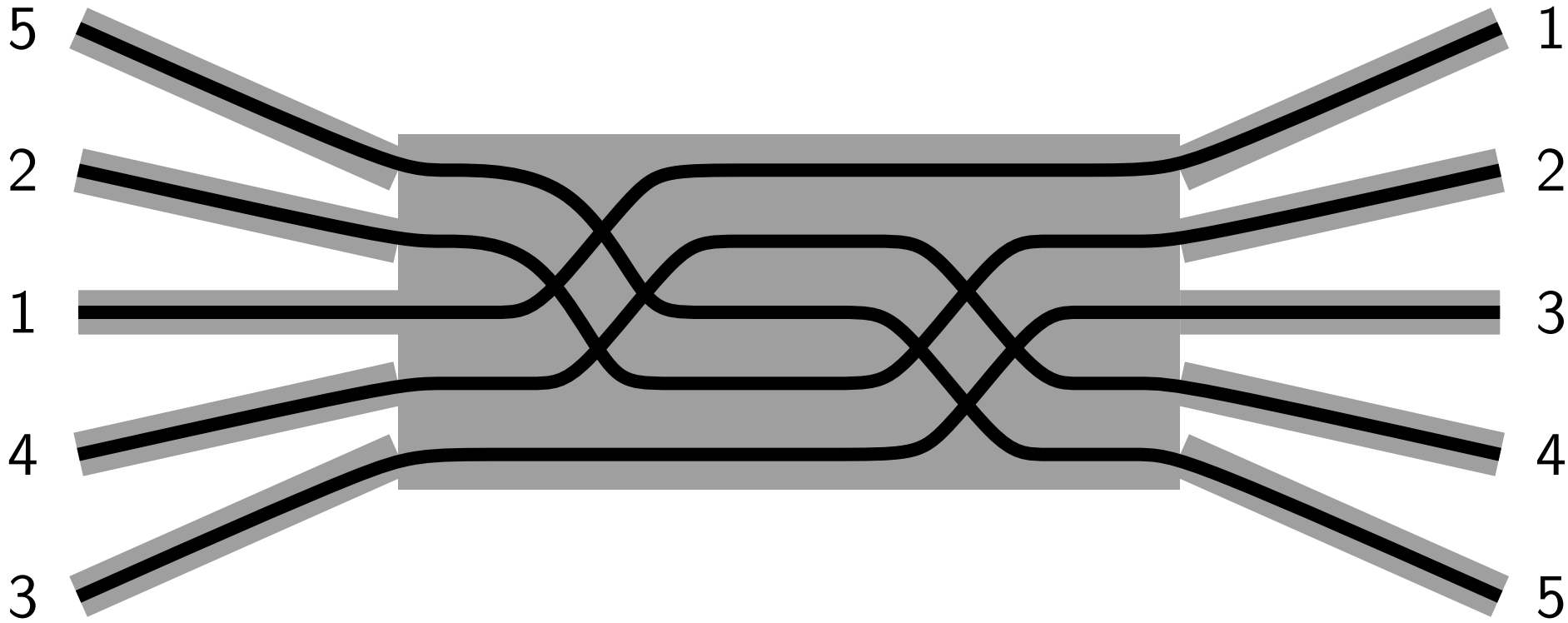
Single Edge



Single Edge

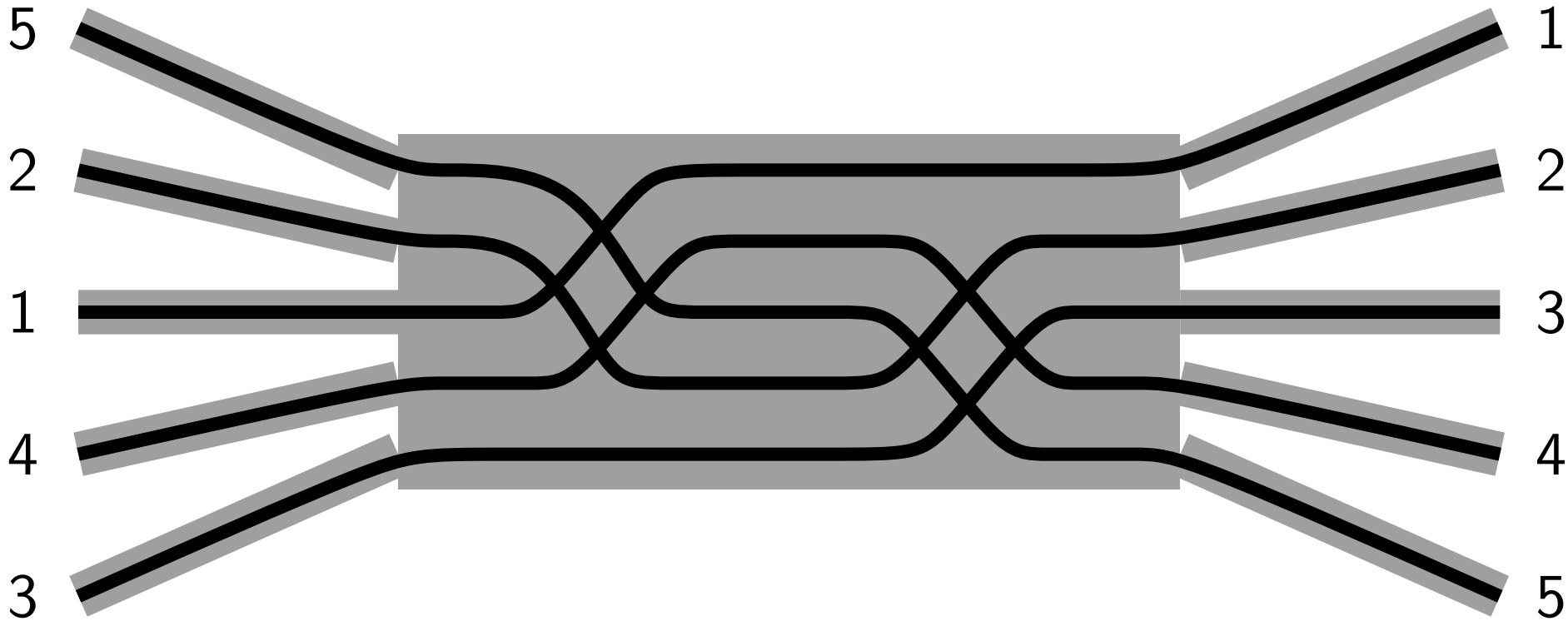


Single Edge



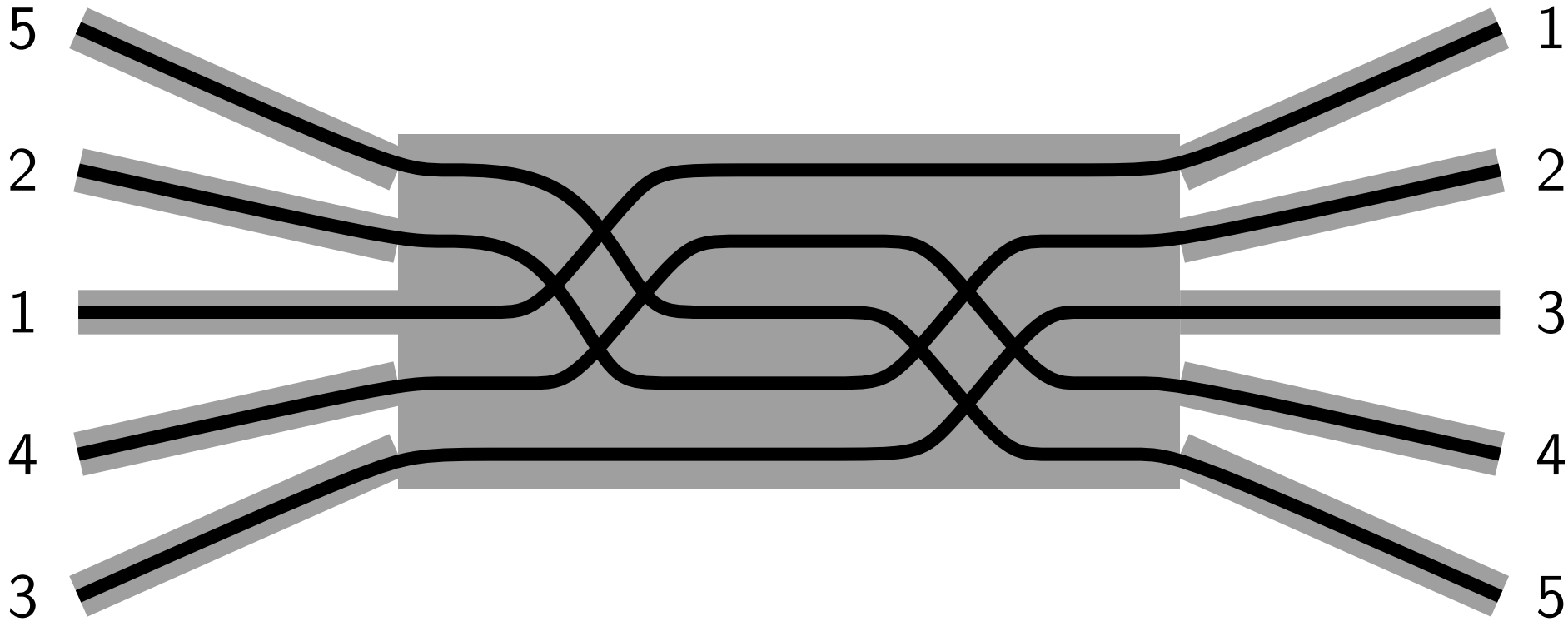
- Equivalent to *Sorting by Transpositions* for permutations

Single Edge



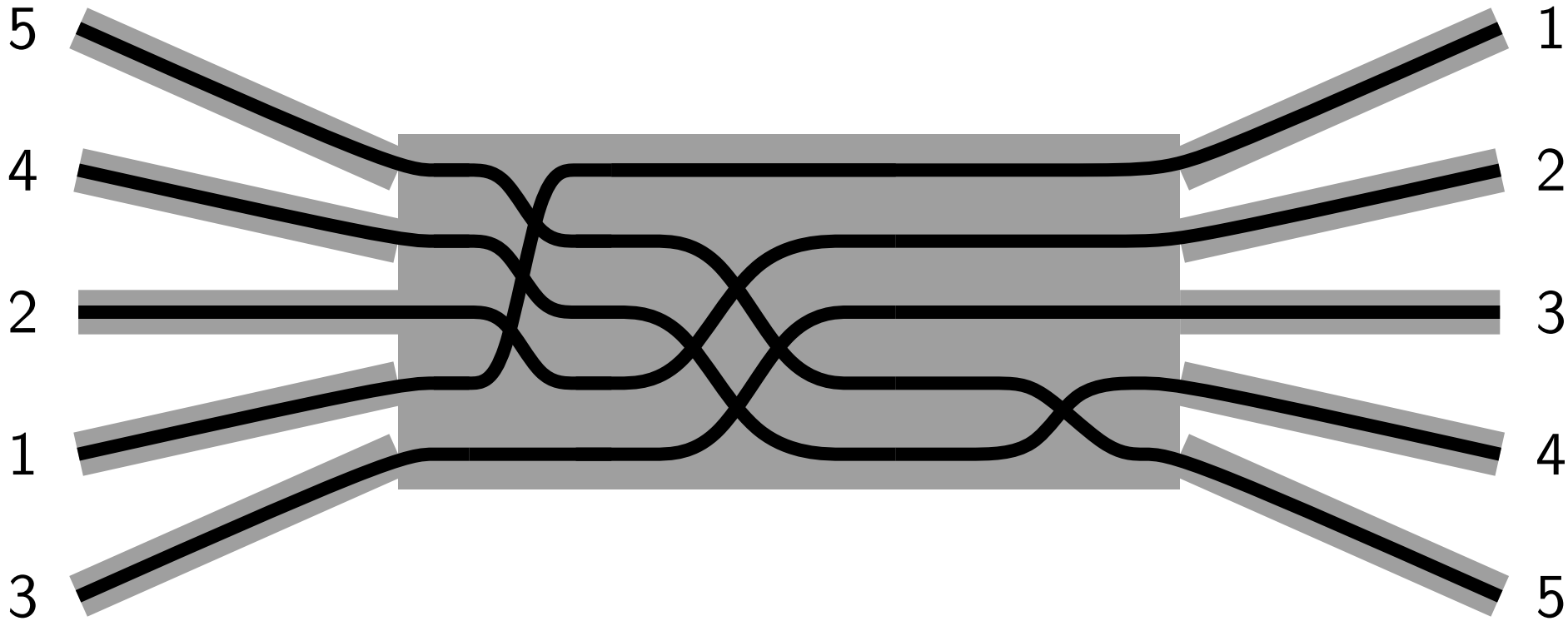
- Equivalent to *Sorting by Transpositions* for permutations
- NP-hard [Bulteau et al., 2012]

Single Edge

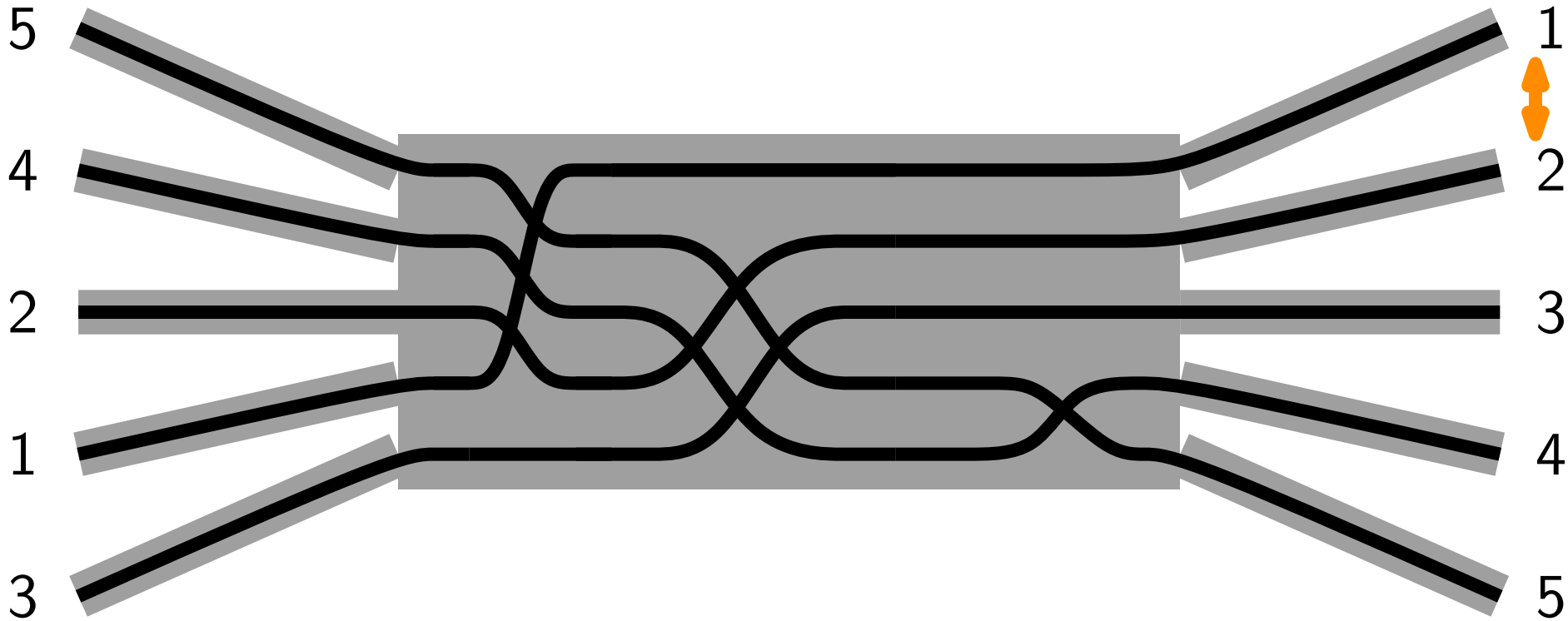


- Equivalent to *Sorting by Transpositions* for permutations
- NP-hard [Bulteau et al., 2012]
- simple 3-approximation [Bafna, Pevzner, 1998]

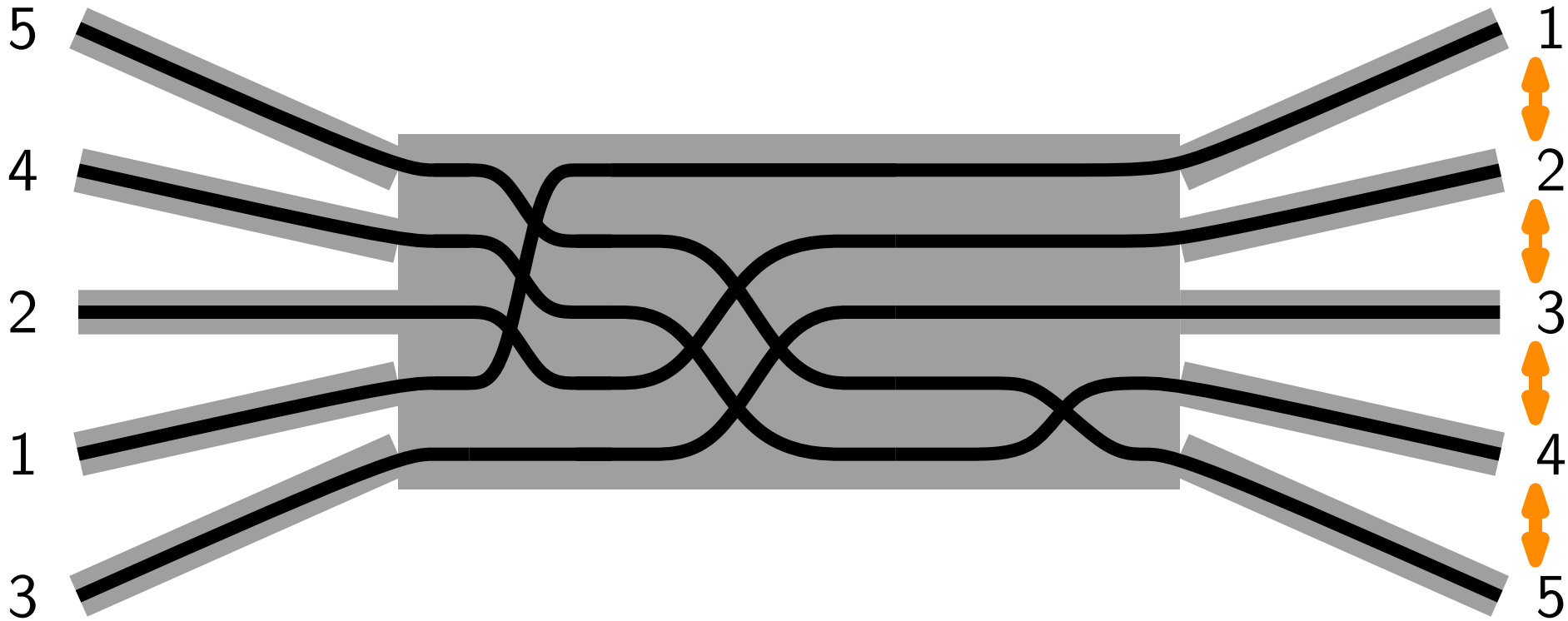
Single Edge – 3-approximation



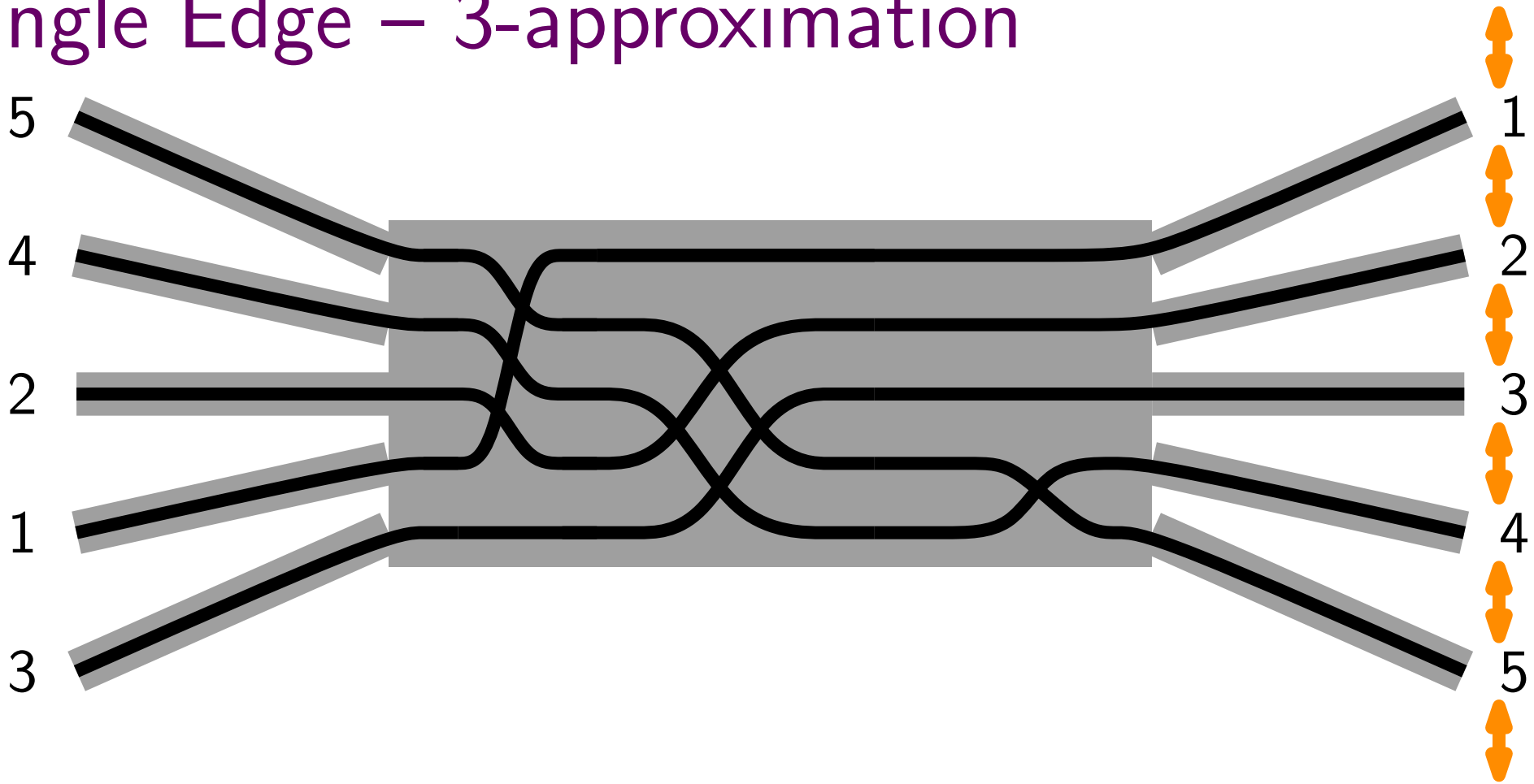
Single Edge – 3-approximation



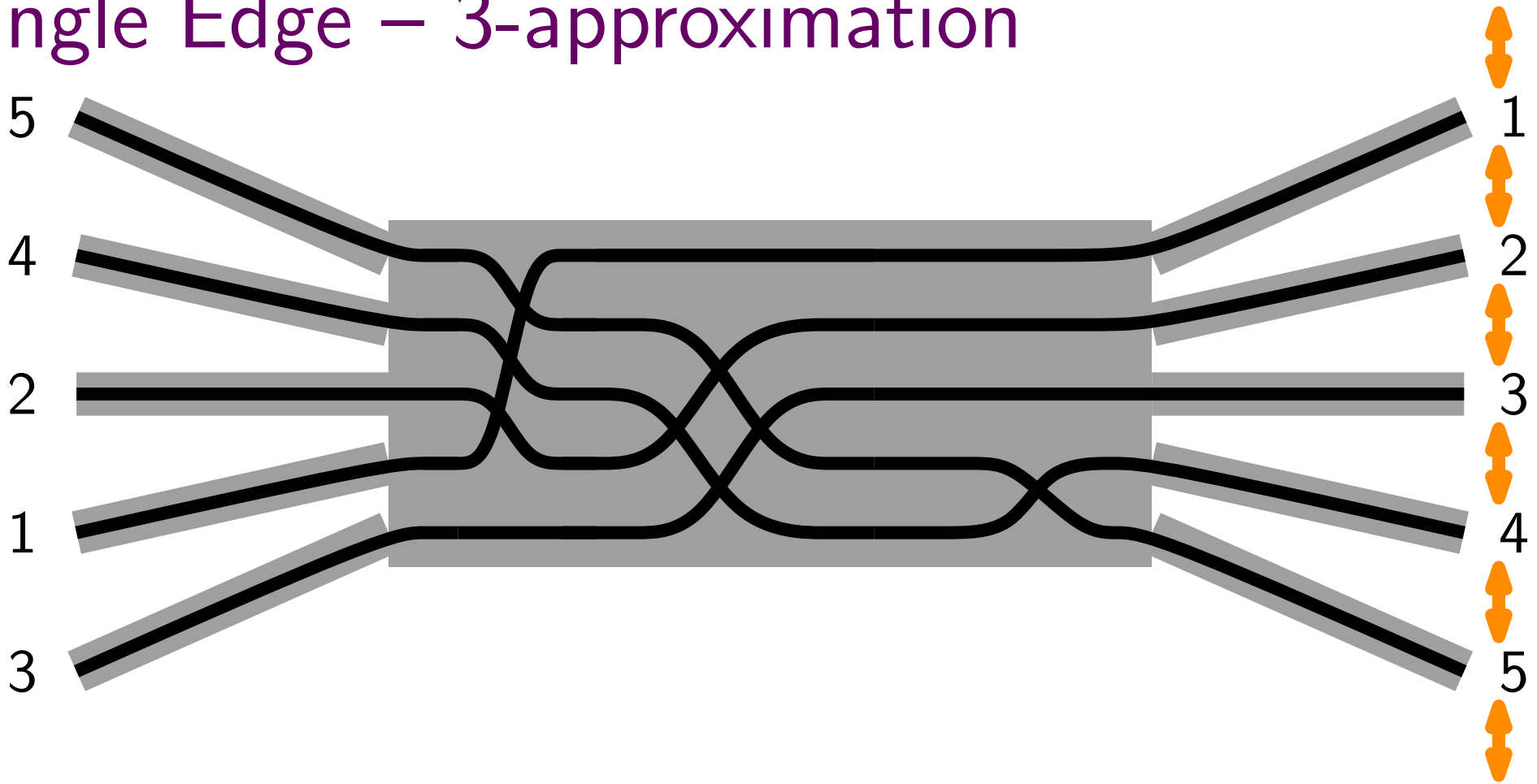
Single Edge – 3-approximation



Single Edge – 3-approximation

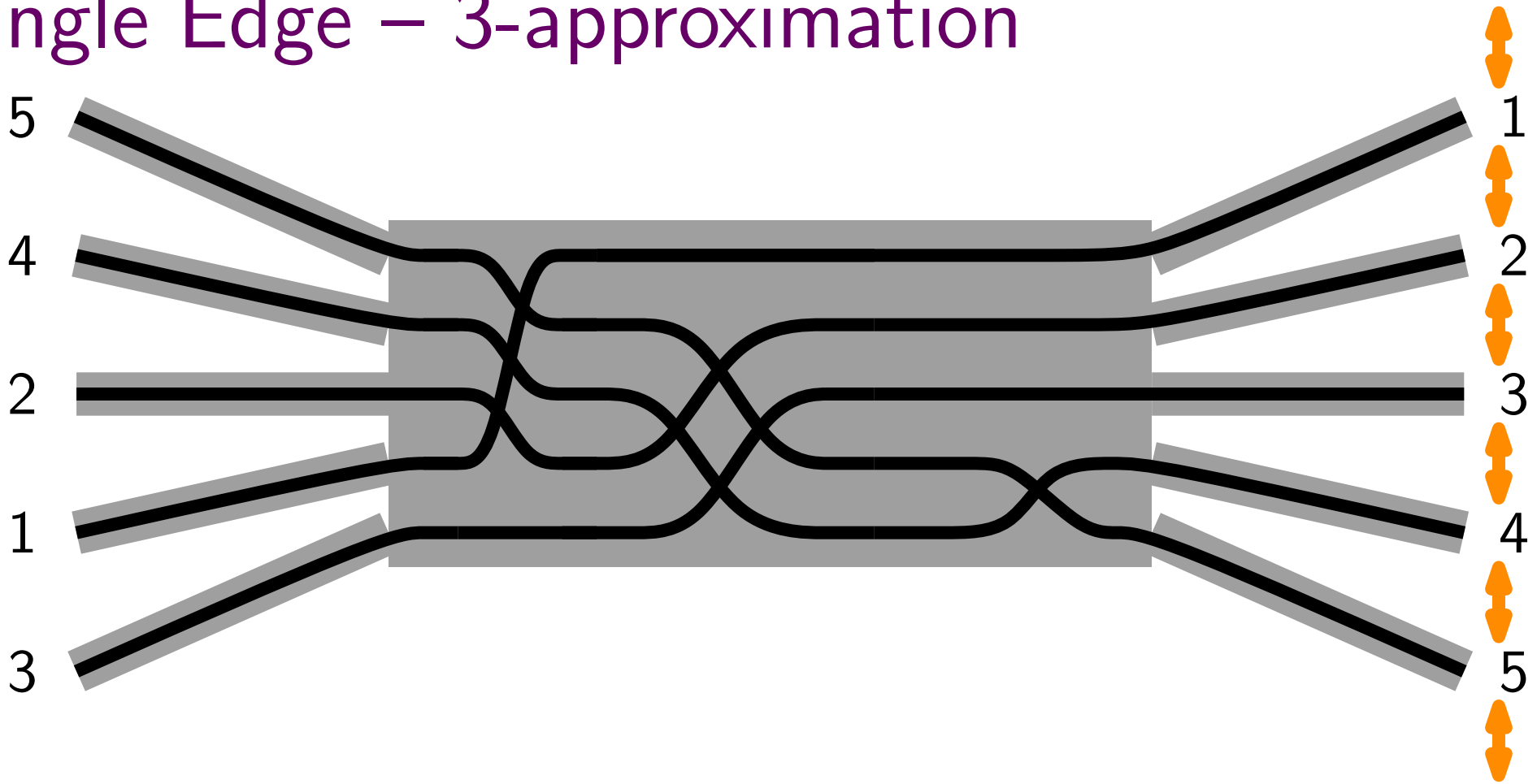


Single Edge – 3-approximation



$|L| + 1$ *good pairs*

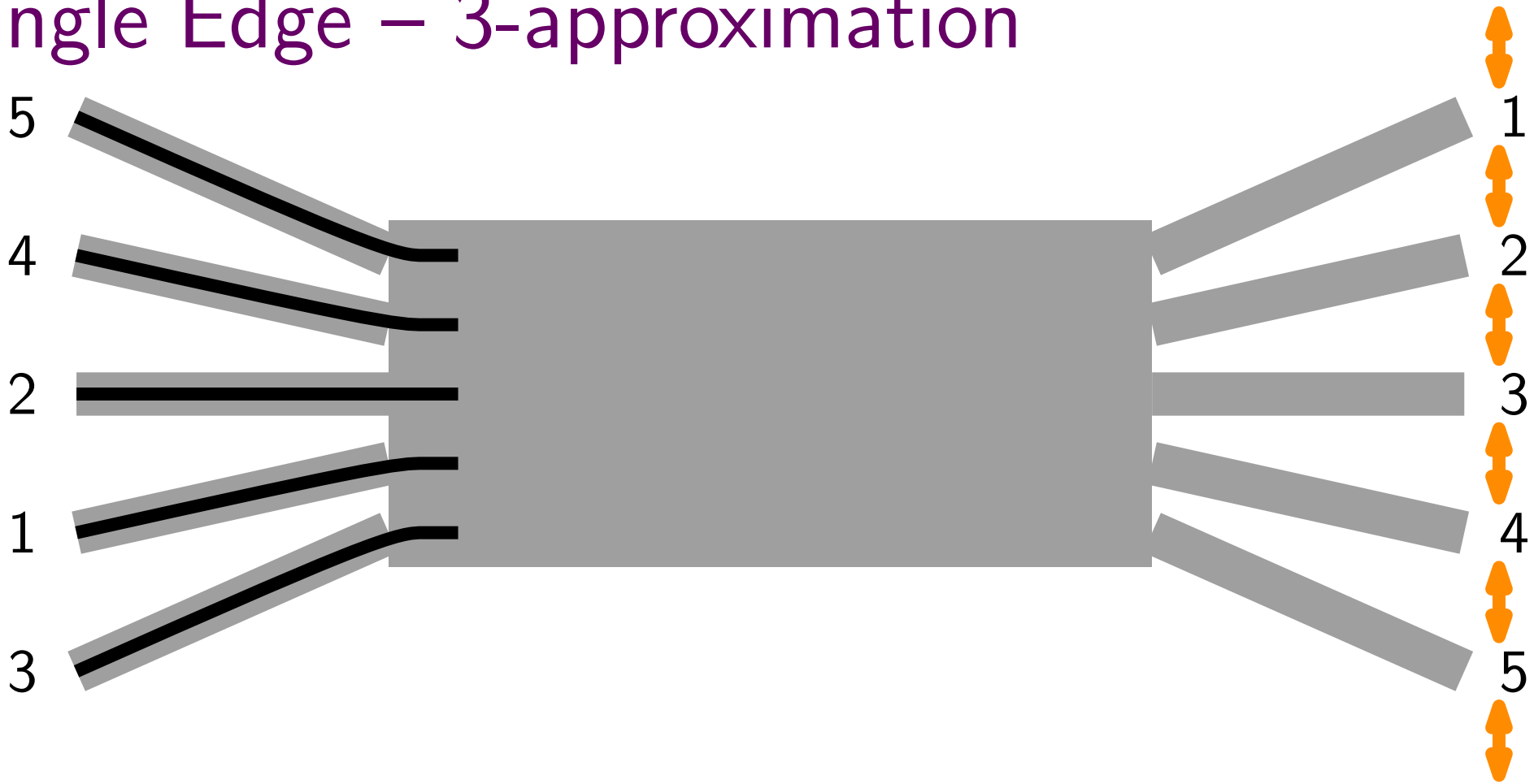
Single Edge – 3-approximation



no good pair
for *simple* permutation

$|L| + 1$ *good pairs*

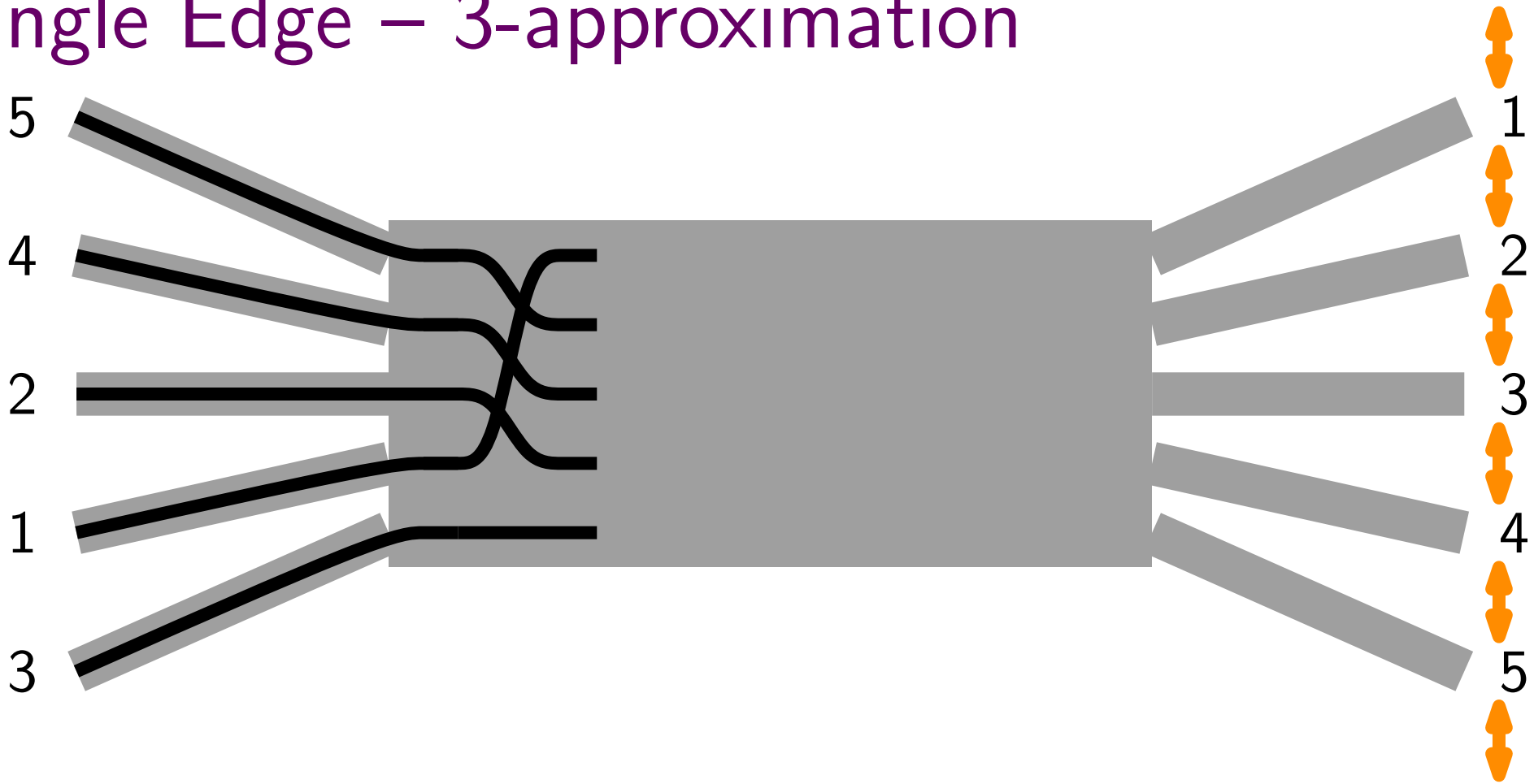
Single Edge – 3-approximation



no good pair
for *simple* permutation

$|L| + 1$ *good pairs*

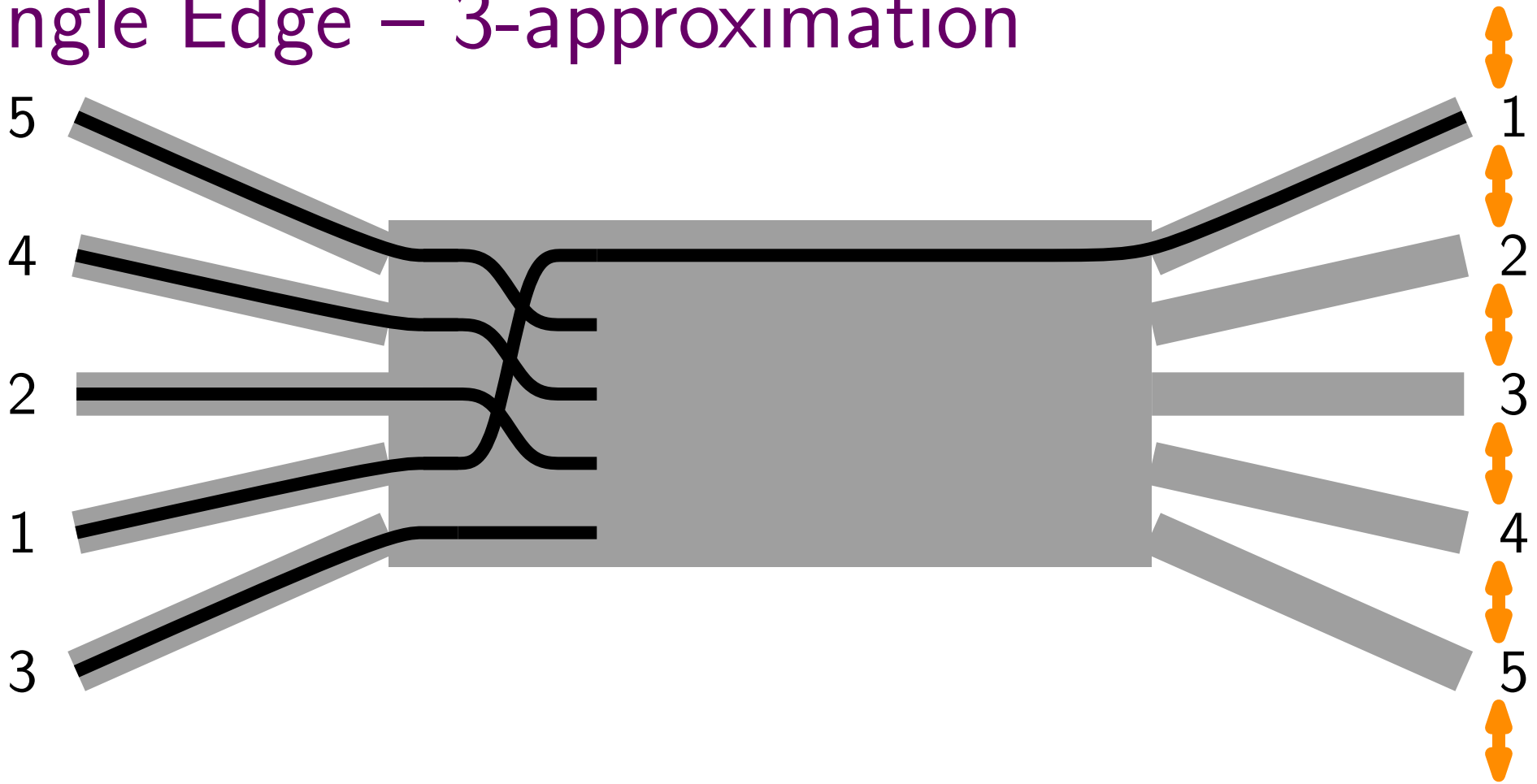
Single Edge – 3-approximation



no good pair
for *simple* permutation

$|L| + 1$ *good pairs*

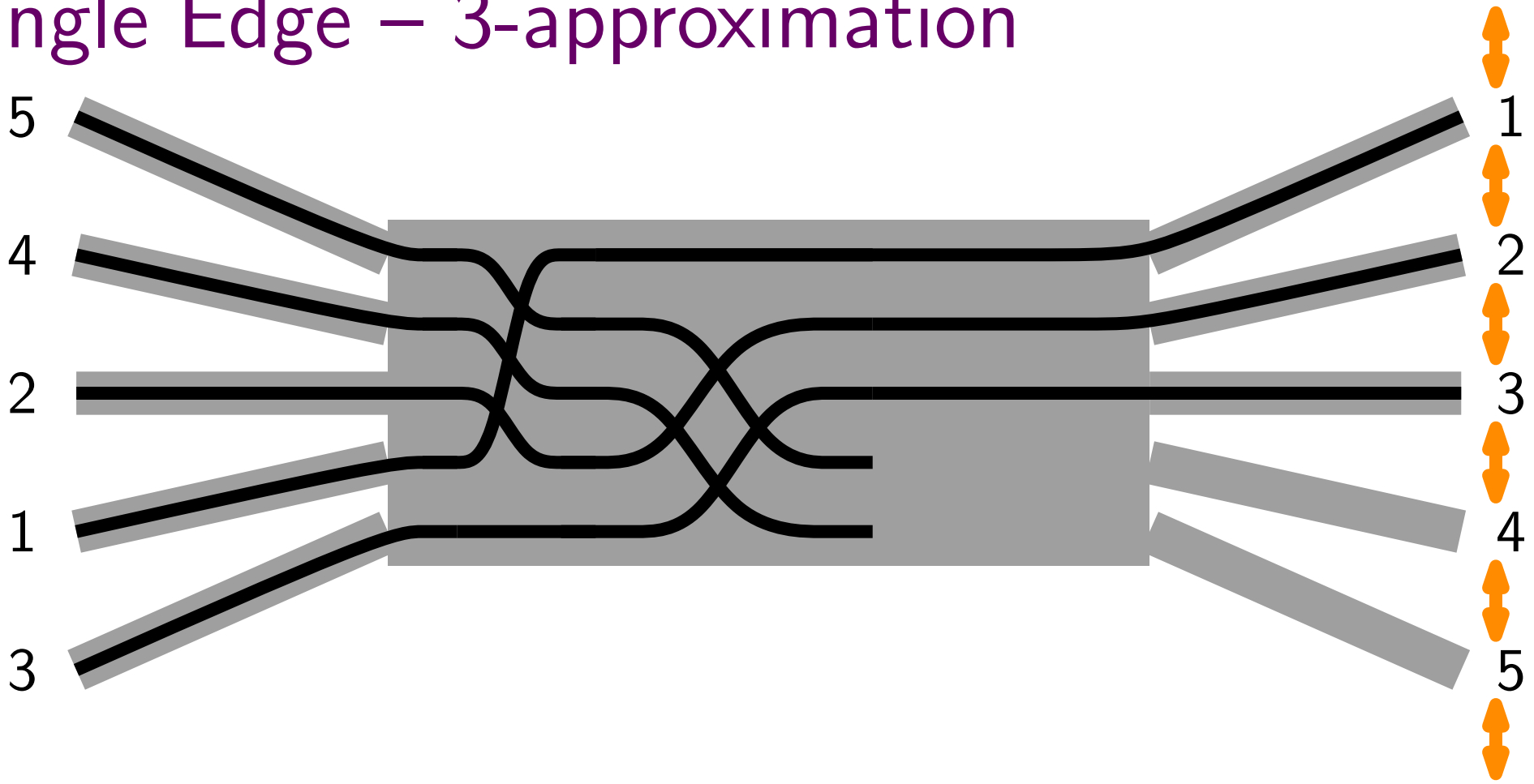
Single Edge – 3-approximation



no good pair
for *simple* permutation

$|L| + 1$ *good pairs*

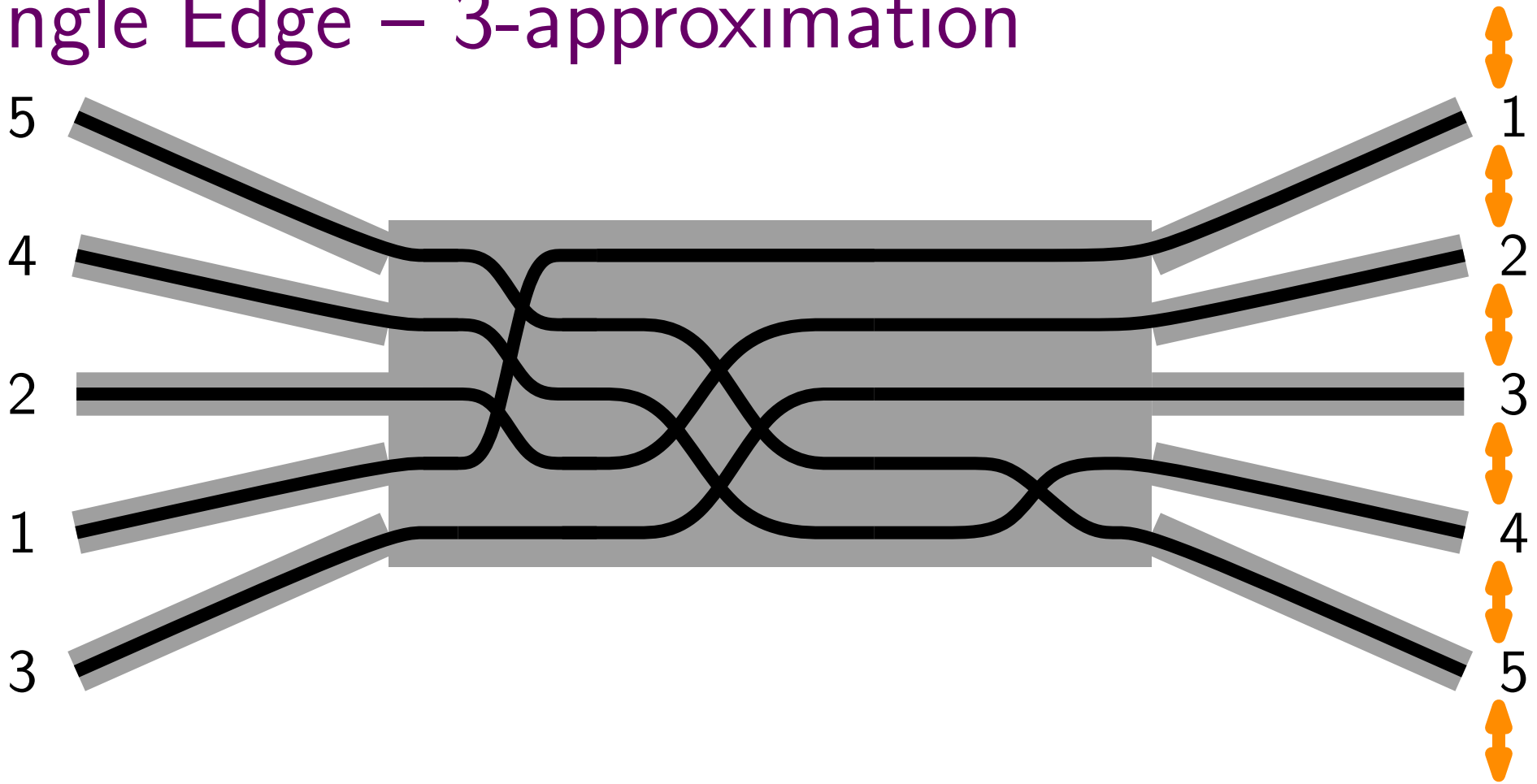
Single Edge – 3-approximation



no good pair
for *simple* permutation

$|L| + 1$ *good pairs*

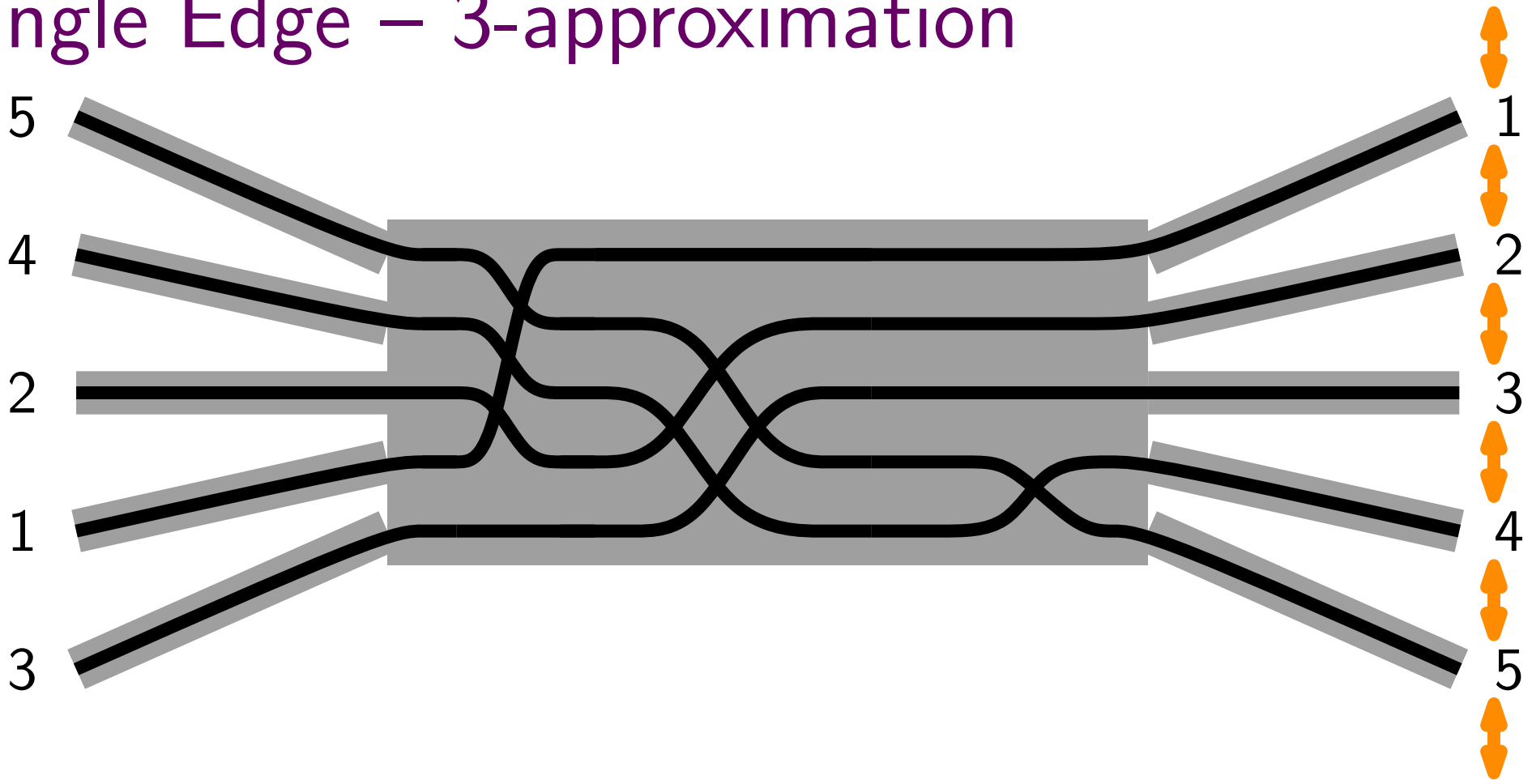
Single Edge – 3-approximation



no good pair
for *simple* permutation

$|L| + 1$ *good pairs*

Single Edge – 3-approximation

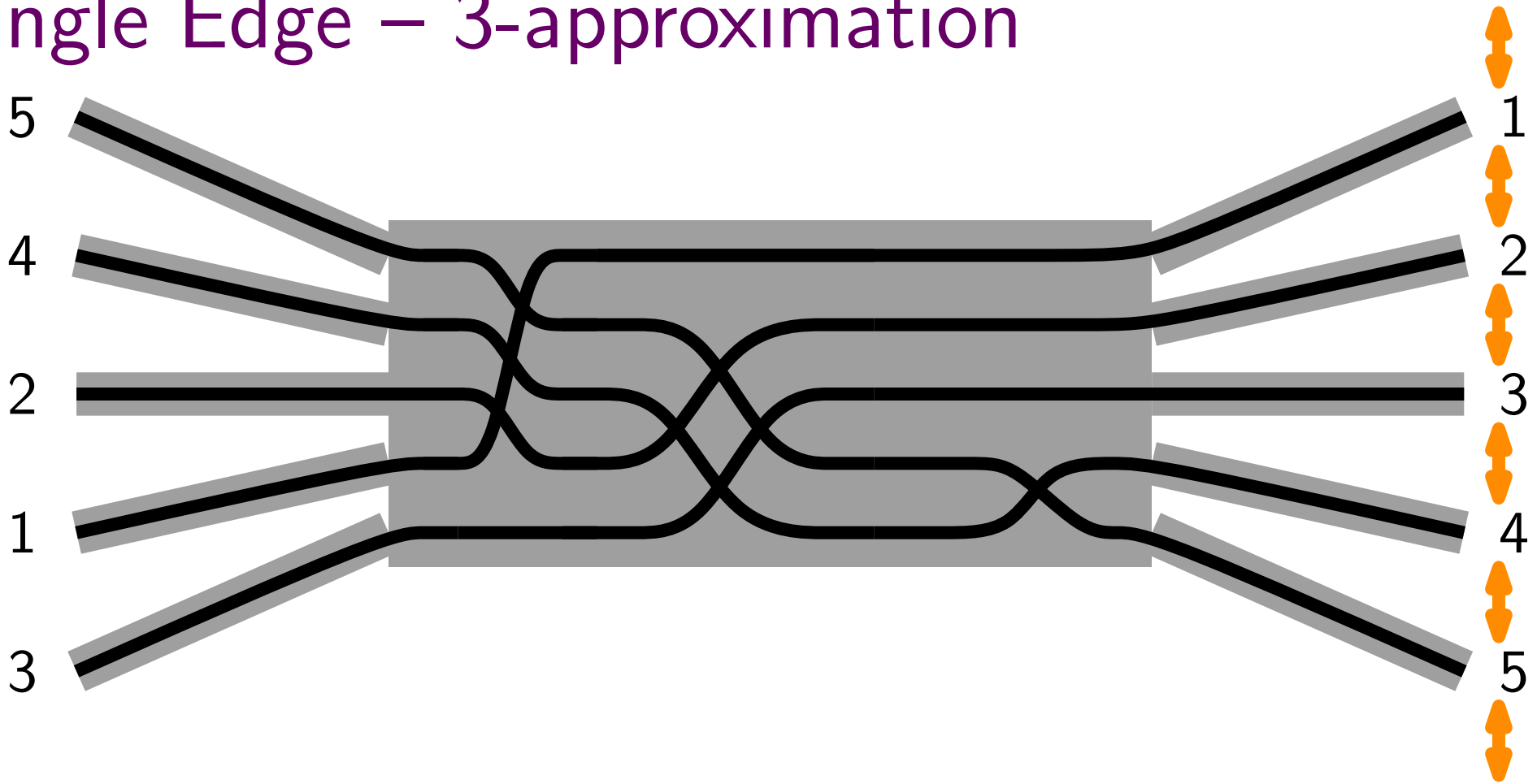


no good pair
for *simple* permutation

$\leq |L| - 1$ crossings

$|L| + 1$ *good pairs*

Single Edge – 3-approximation

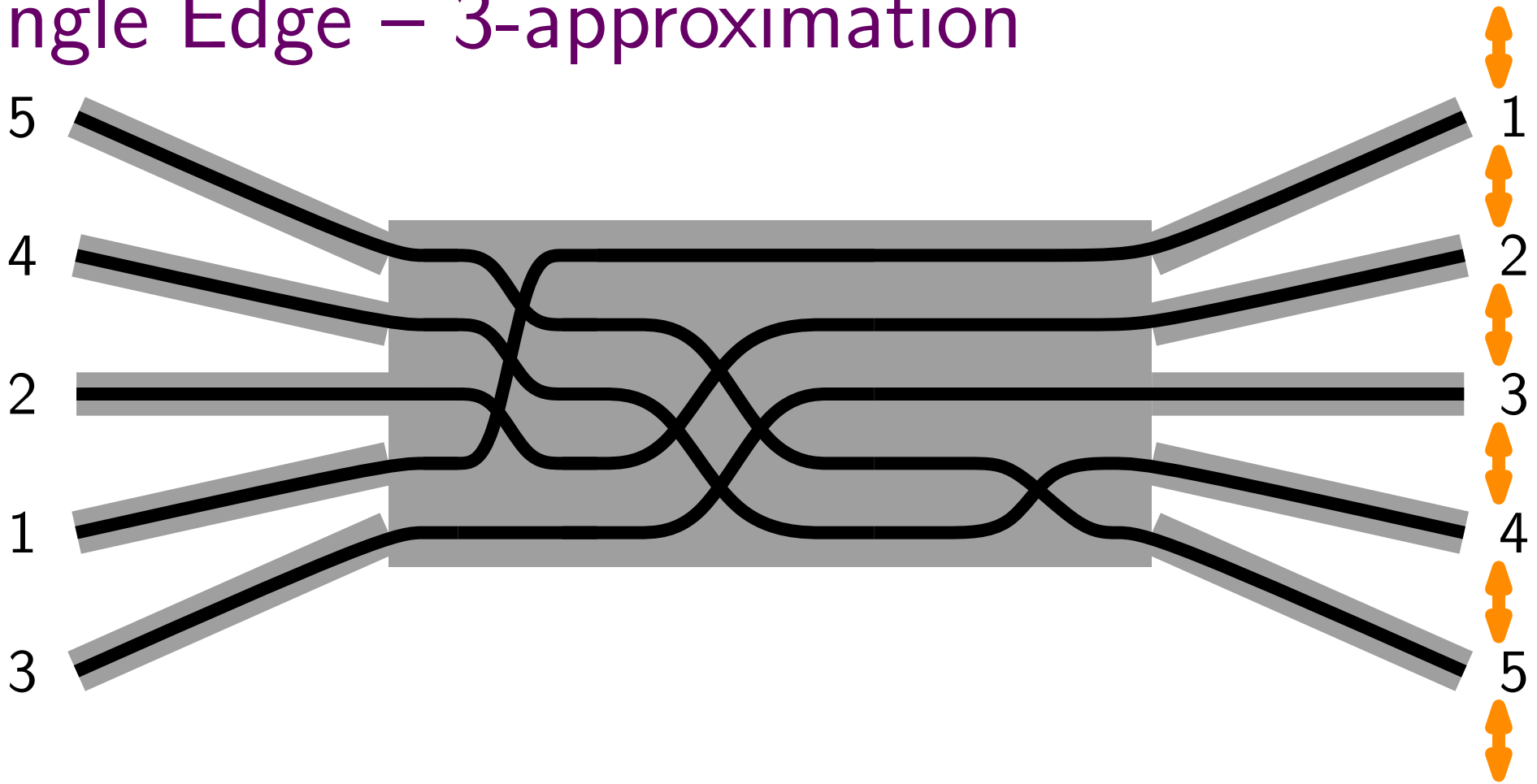


no good pair
for *simple* permutation

$\leq |L| - 1$ crossings

$|L| + 1$ *good pairs*

Single Edge – 3-approximation

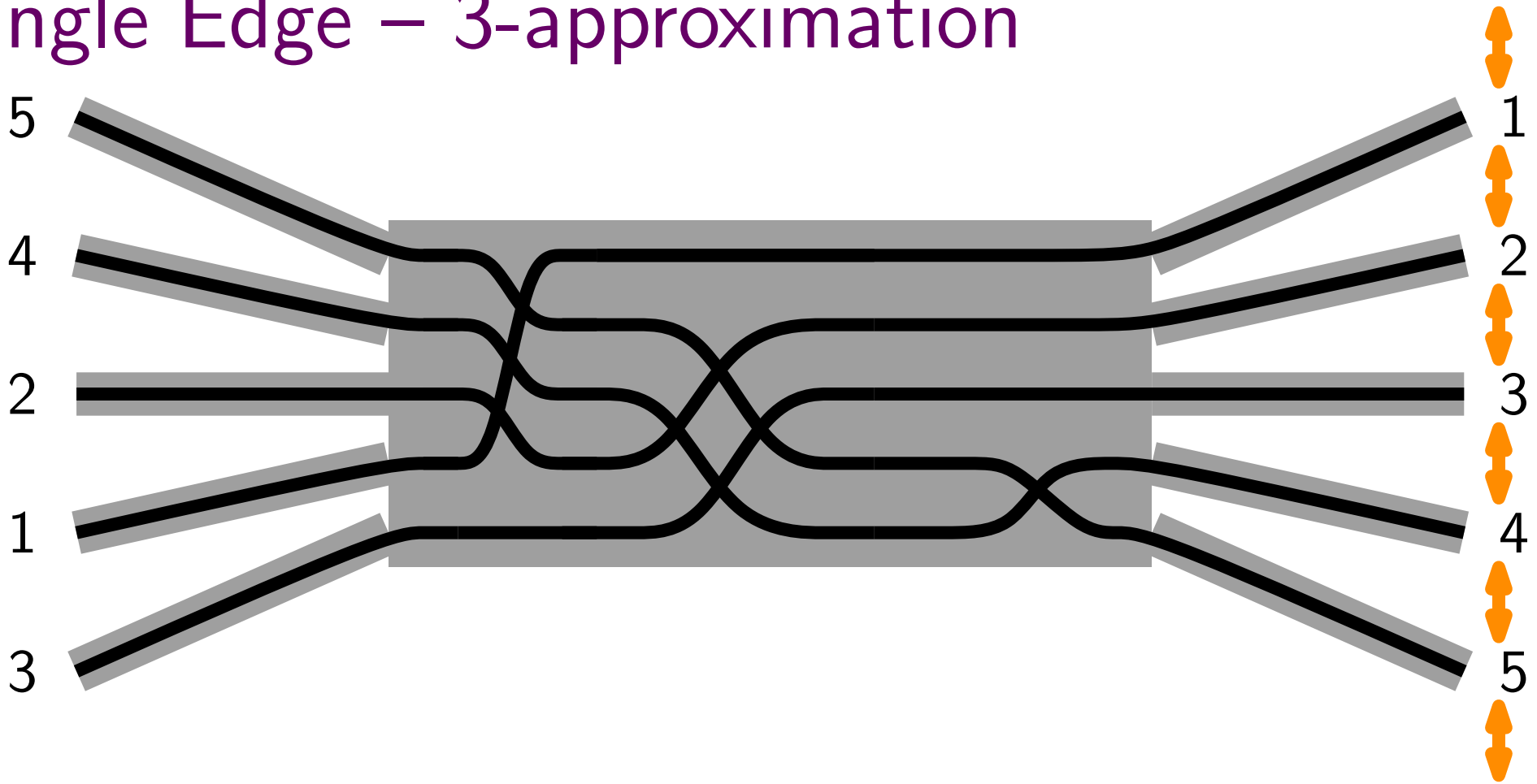


no good pair
for *simple* permutation

$\leq |L| - 1$ crossings

$|L| + 1$ *good pairs*

Single Edge – 3-approximation



no good pair
for *simple* permutation

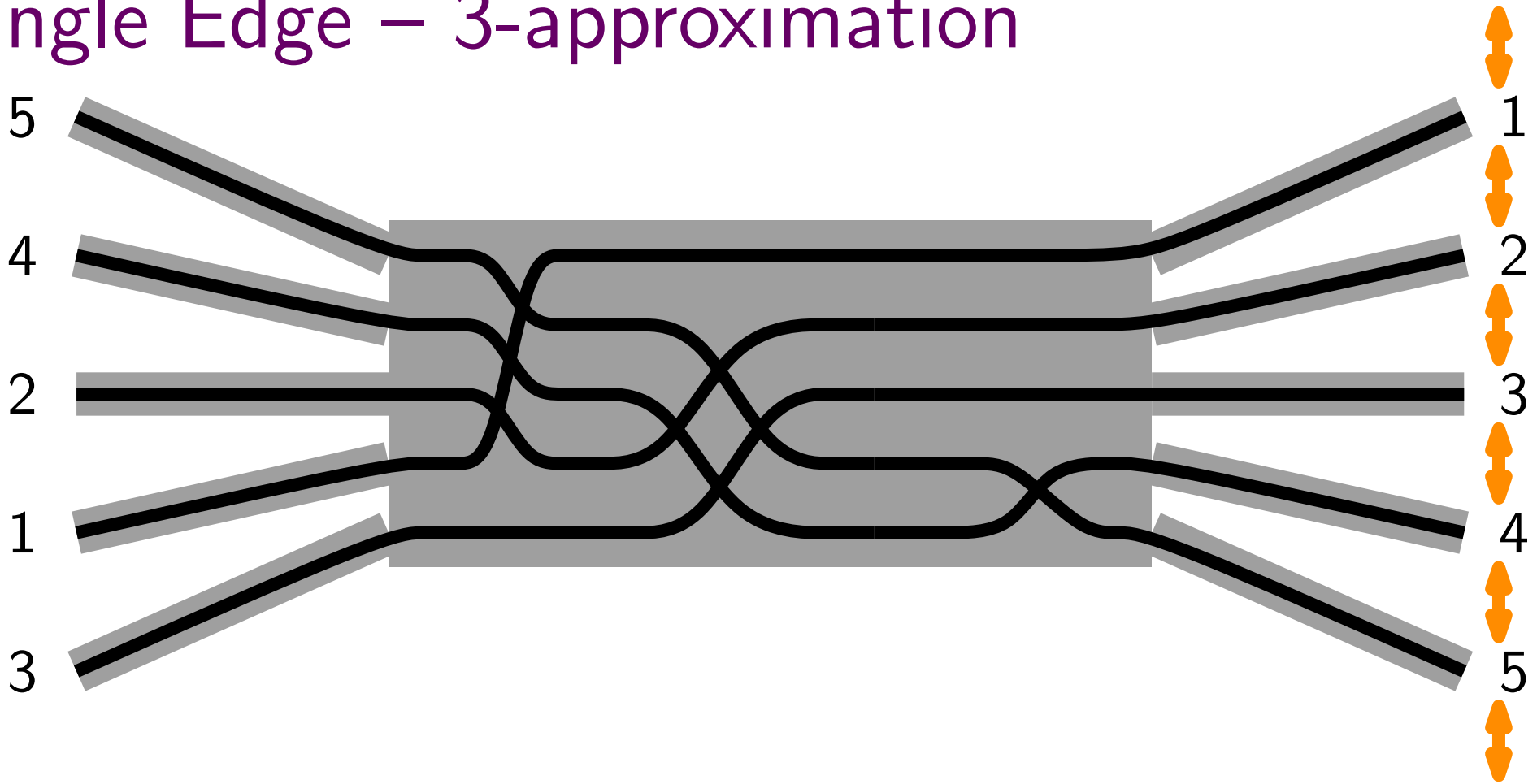
$\leq |L| - 1$ crossings

$|L| + 1$ *good pairs*

≤ 3 good pairs can be created per crossing

3-approx.

Single Edge – 3-approximation



no good pair
for *simple* permutation

$\leq |L| - 1$ crossings

$|L| + 1$ *good pairs*

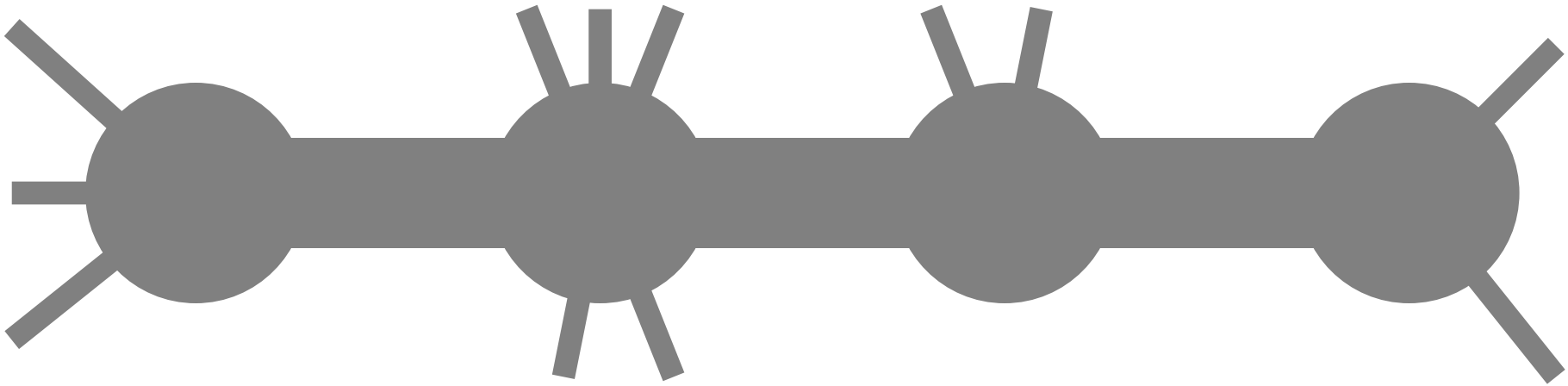
solution uses monotone block crossings!

≤ 3 good pairs can be created per crossing

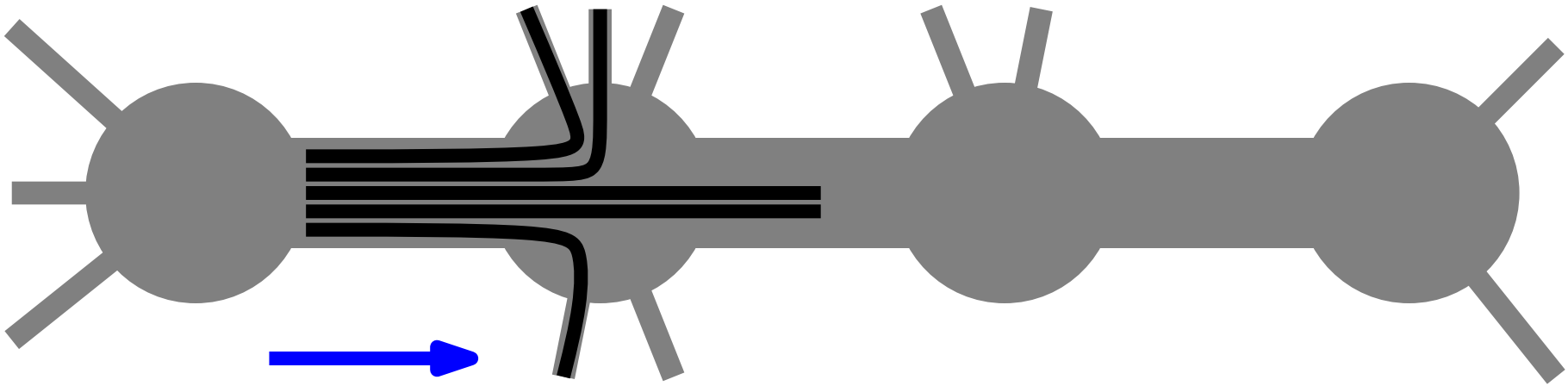
3-approx.

Path

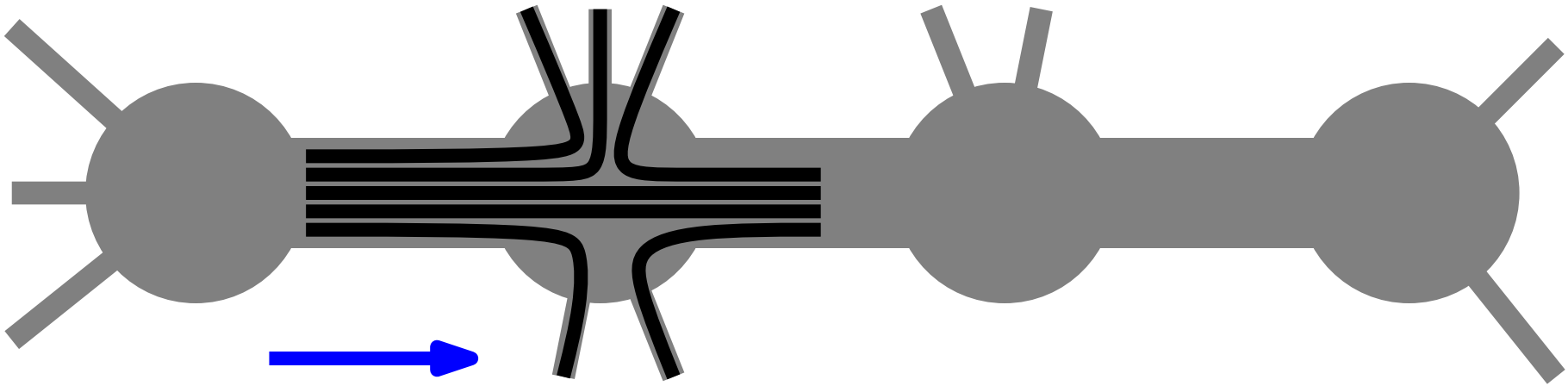
Path



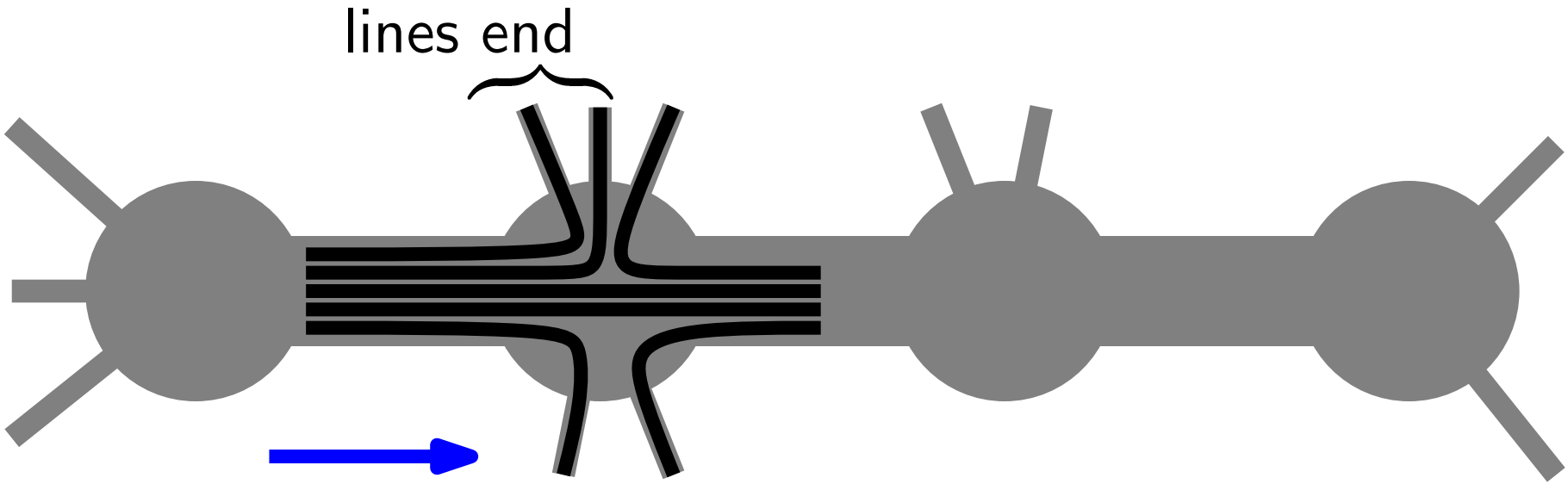
Path



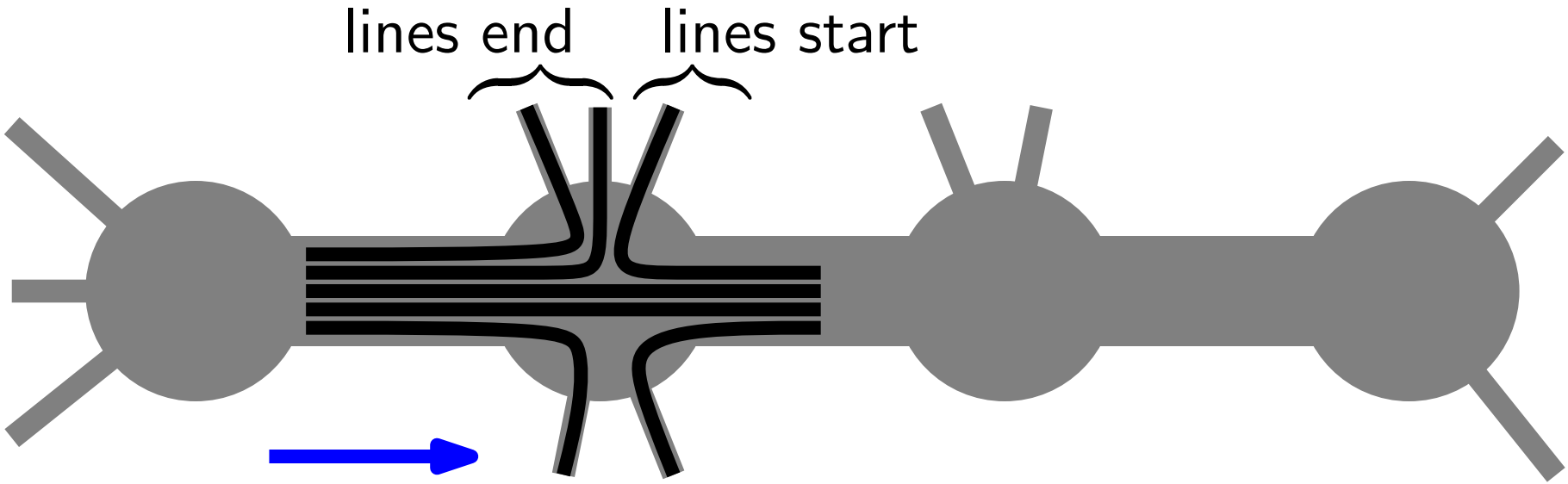
Path



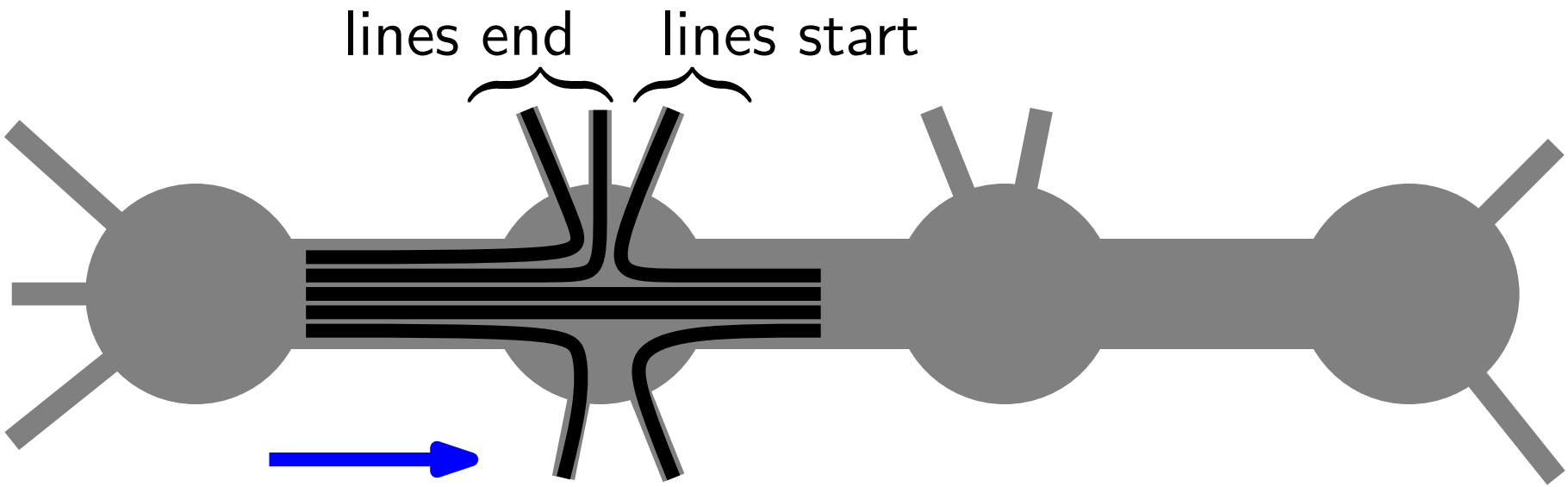
Path



Path

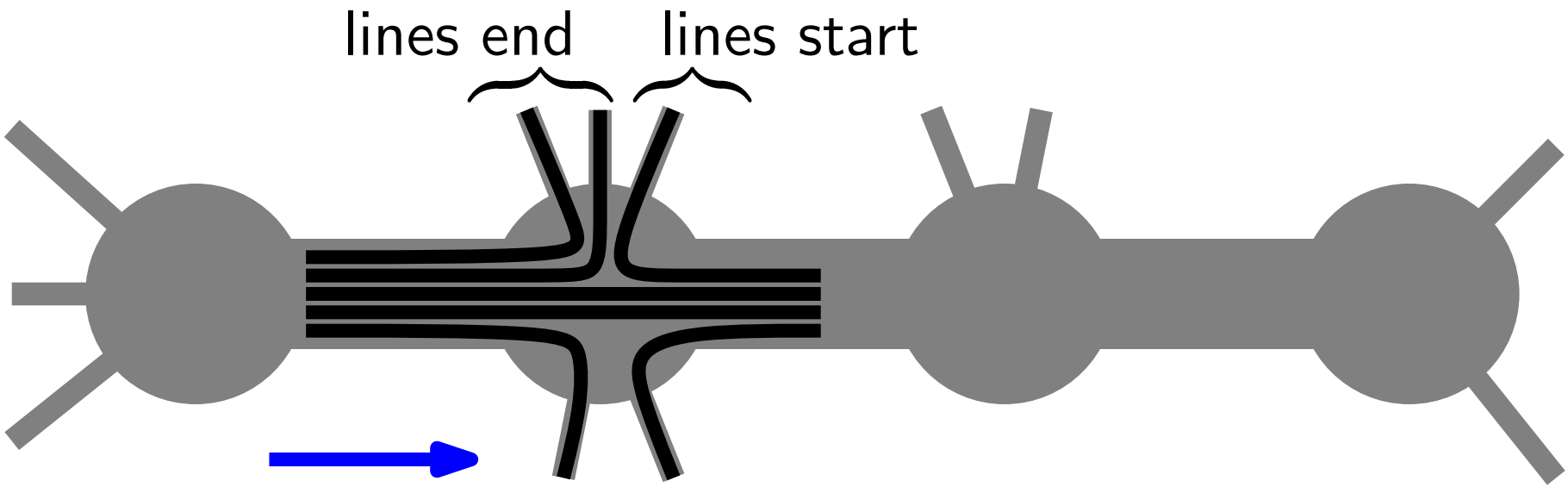


Path

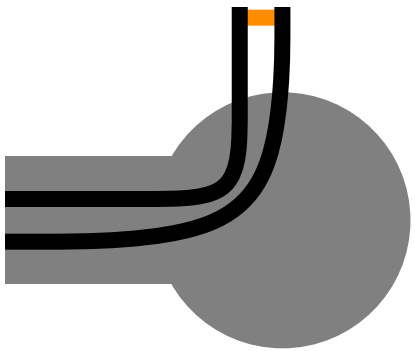


- Redefine good pairs:

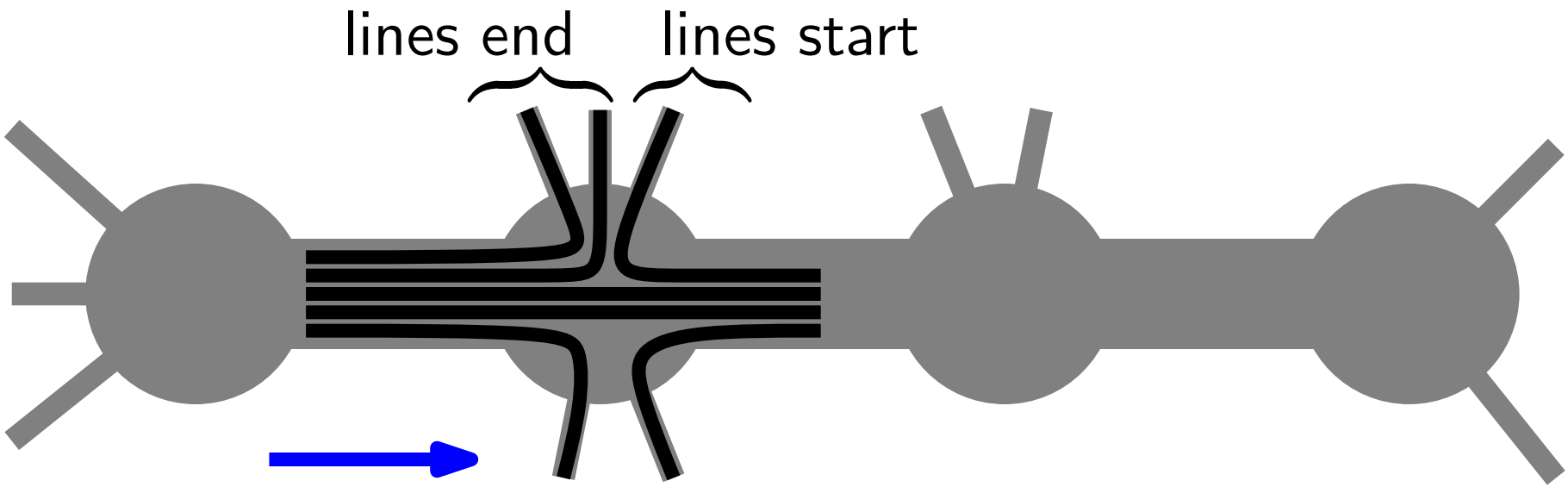
Path



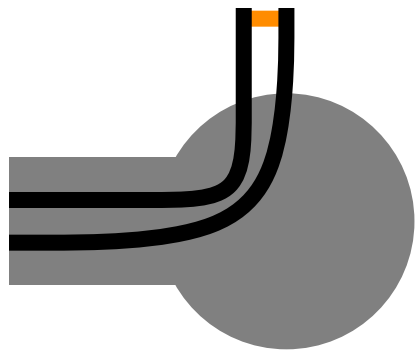
- Redefine good pairs:
  - lines end together

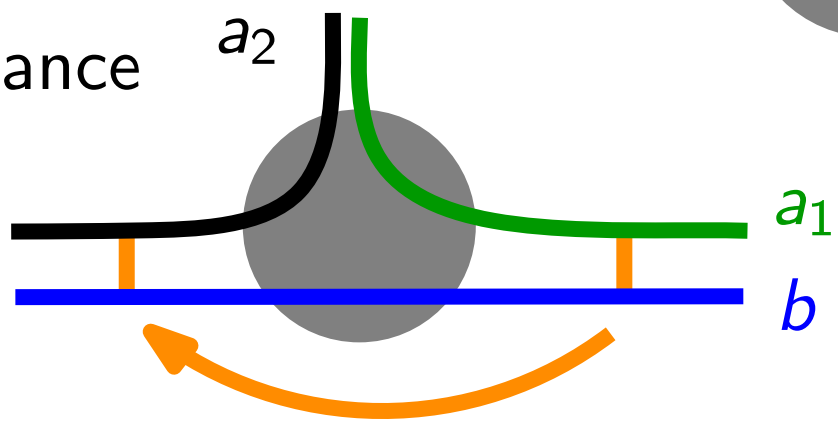


Path

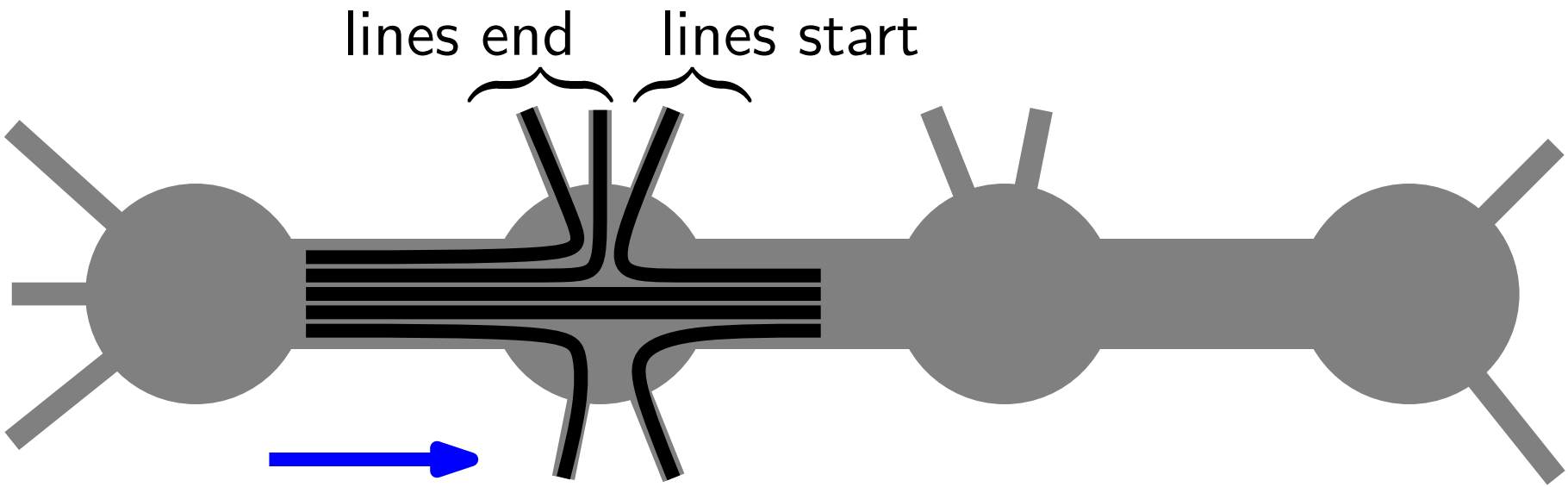


- Redefine good pairs:
  - lines end together
  - inheritance

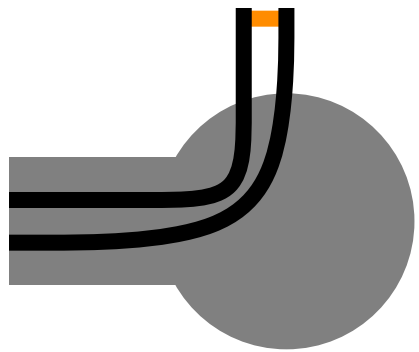


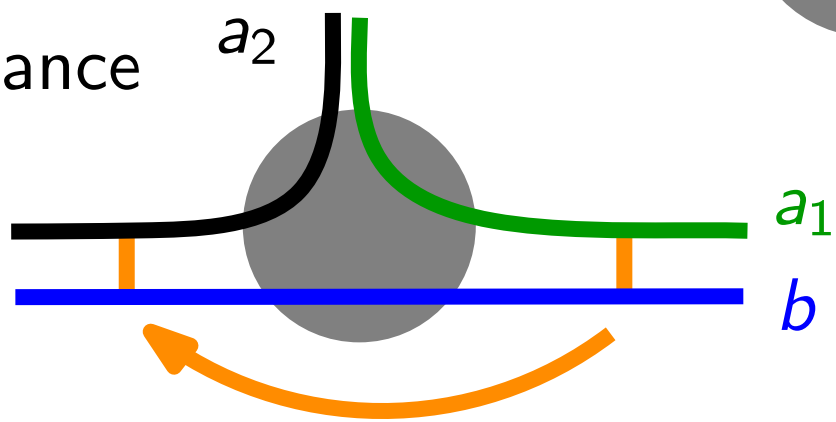


Path



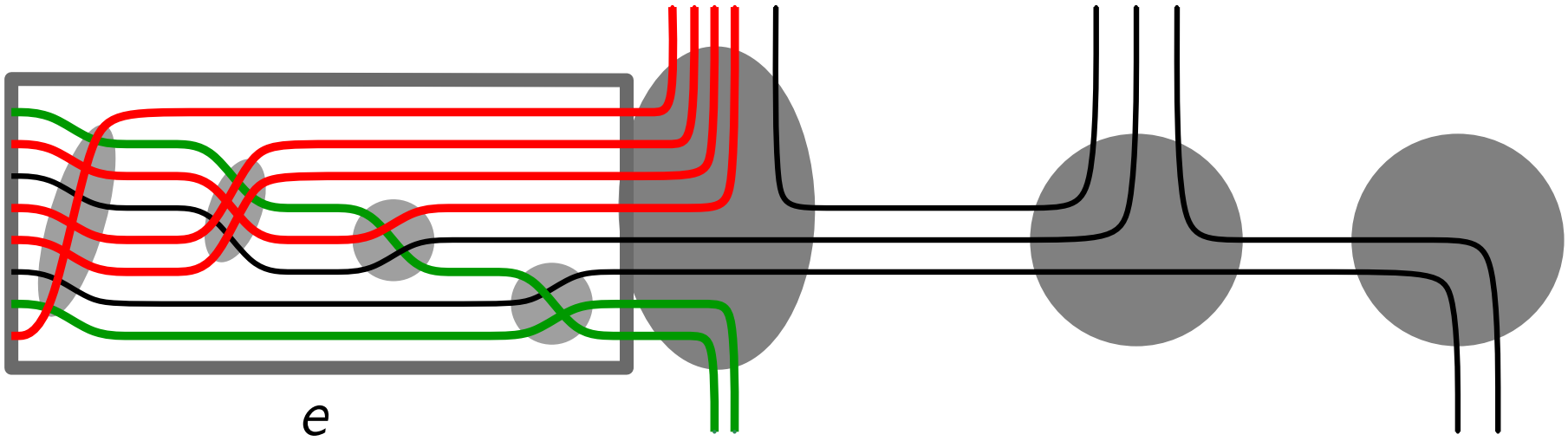
- Redefine good pairs:
  - lines end together
  - inheritance





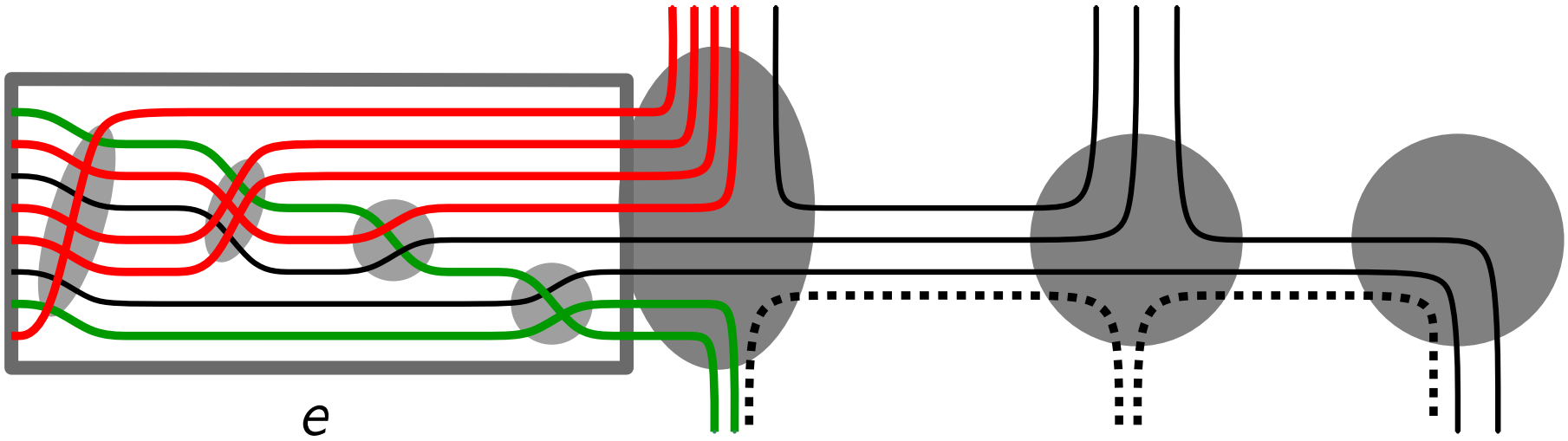
≤ 3 good pairs created per crossing

Path – 3-approximation



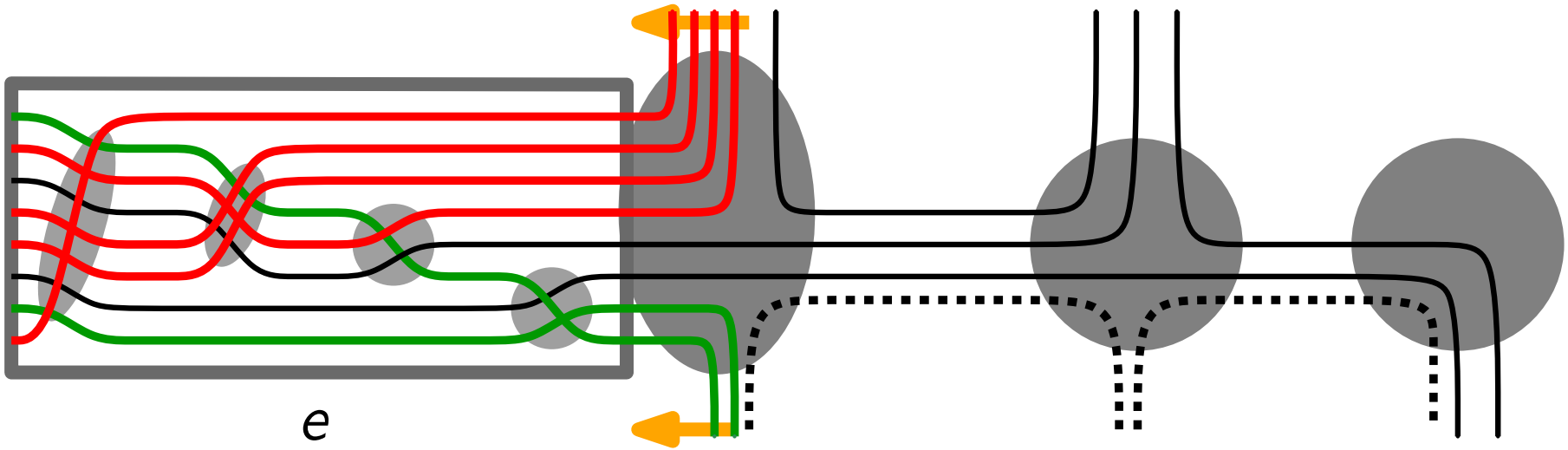
- treat edges from left to right

Path – 3-approximation



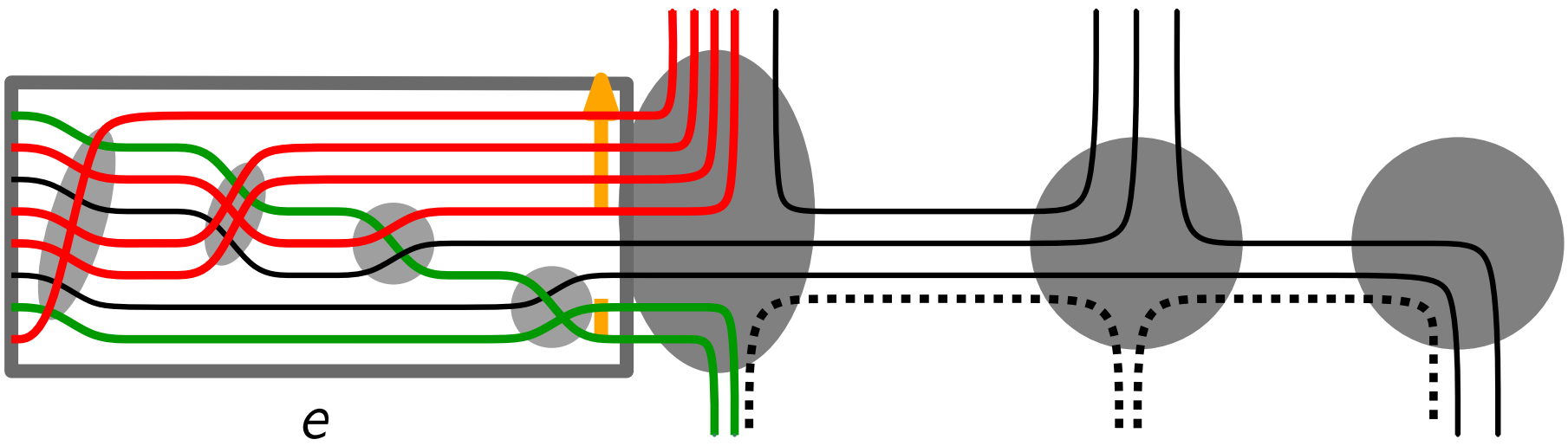
- treat edges from left to right

Path – 3-approximation



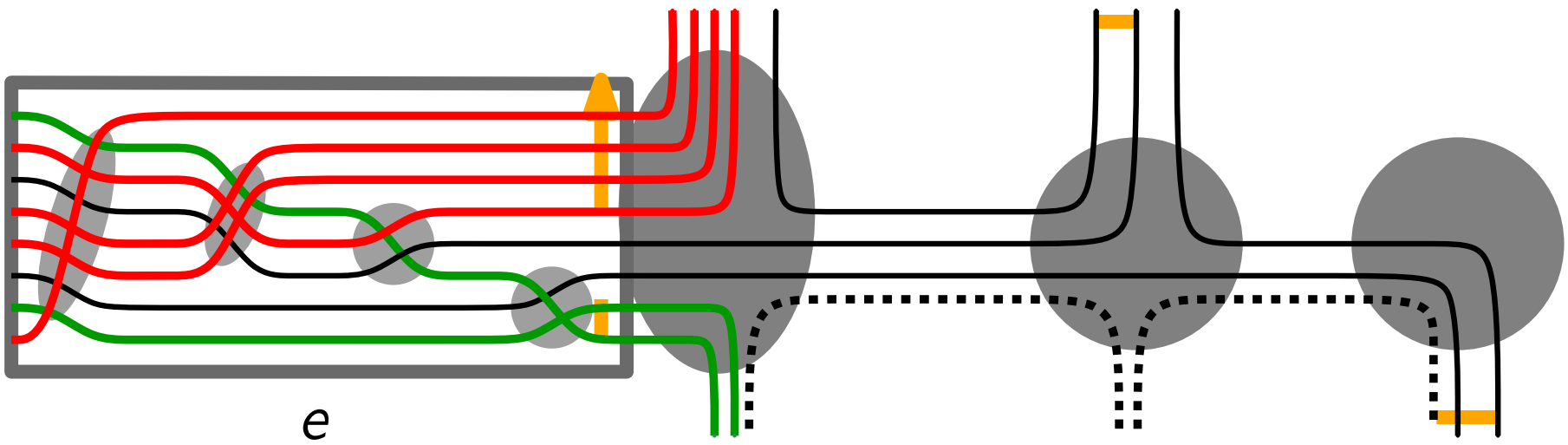
- treat edges from left to right
- identify good pairs

Path – 3-approximation



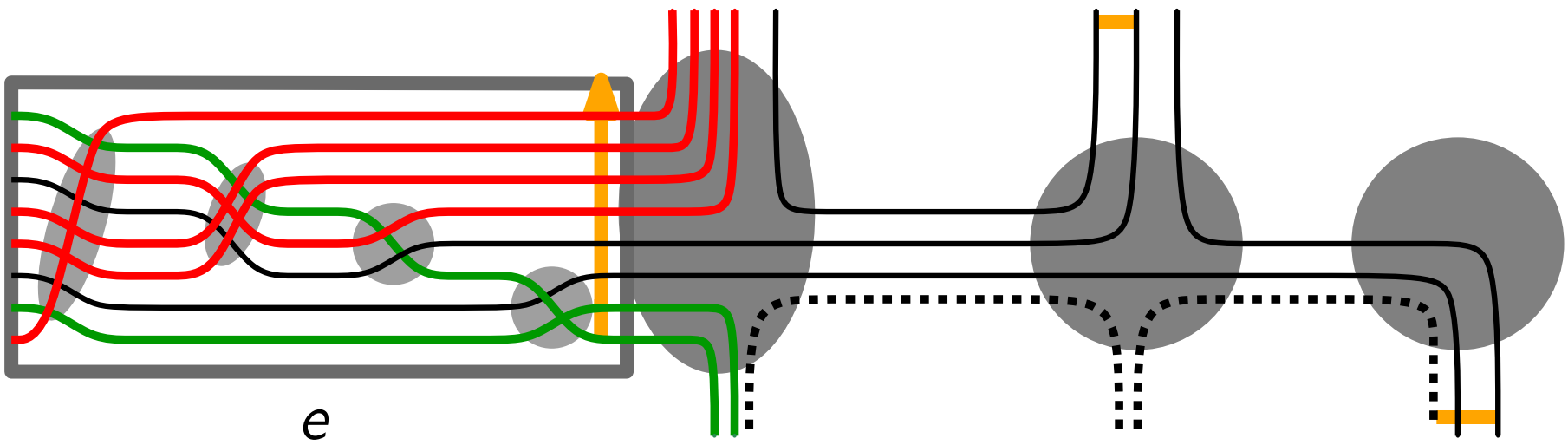
- treat edges from left to right

Path – 3-approximation



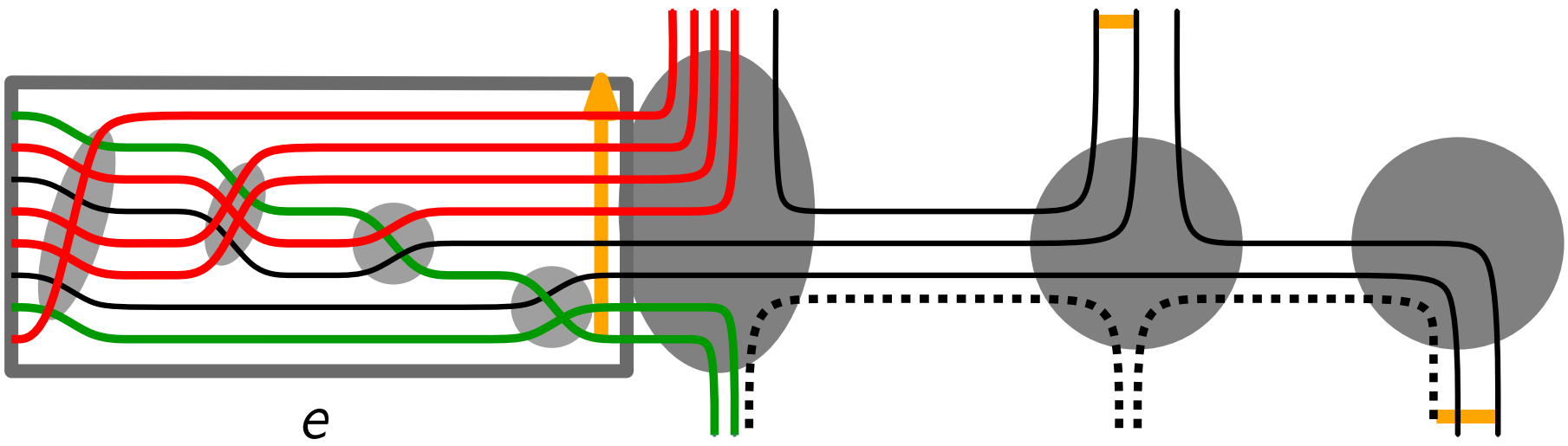
- treat edges from left to right

Path – 3-approximation



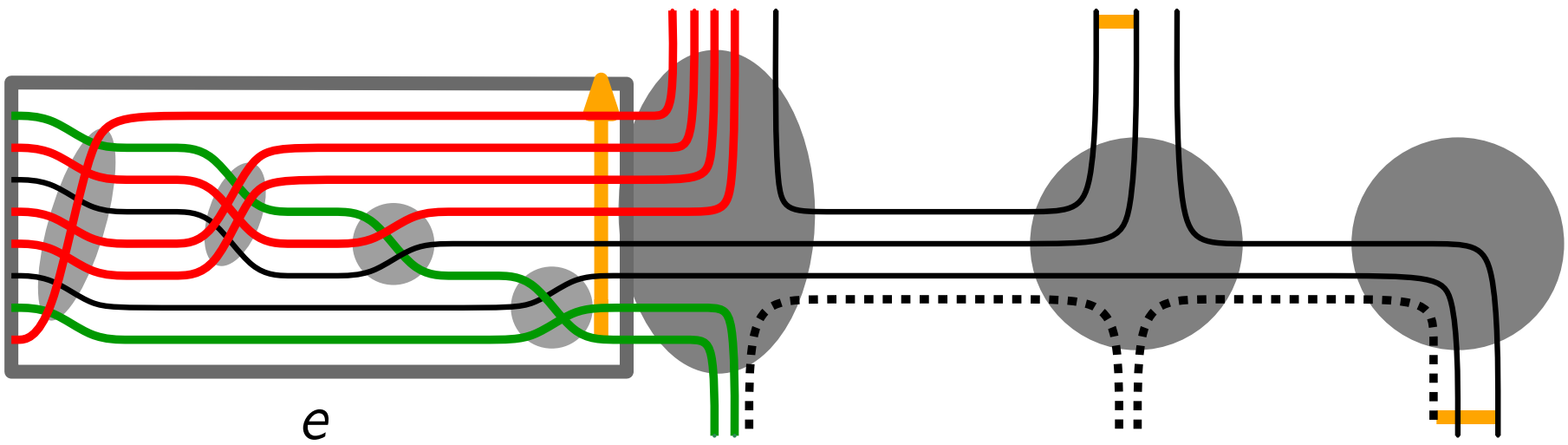
- treat edges from left to right

Path – 3-approximation



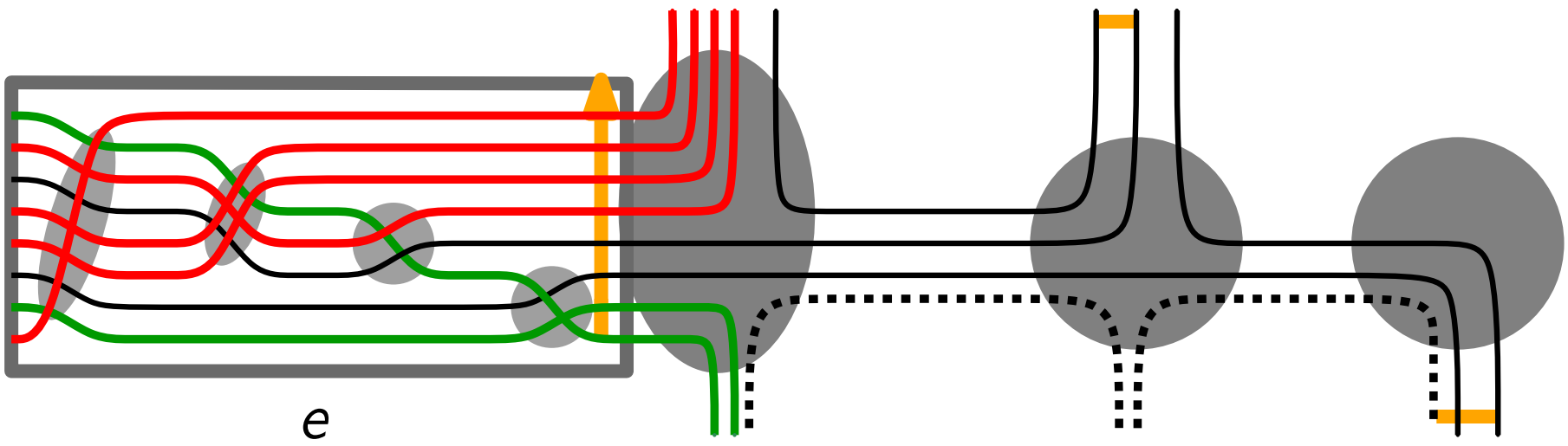
- treat edges from left to right
- bring ending lines to top/bottom keeping good pairs together

Path – 3-approximation



- treat edges from left to right
- bring ending lines to top/bottom keeping good pairs together
- create 1 good pair per block crossing crossing

Path – 3-approximation

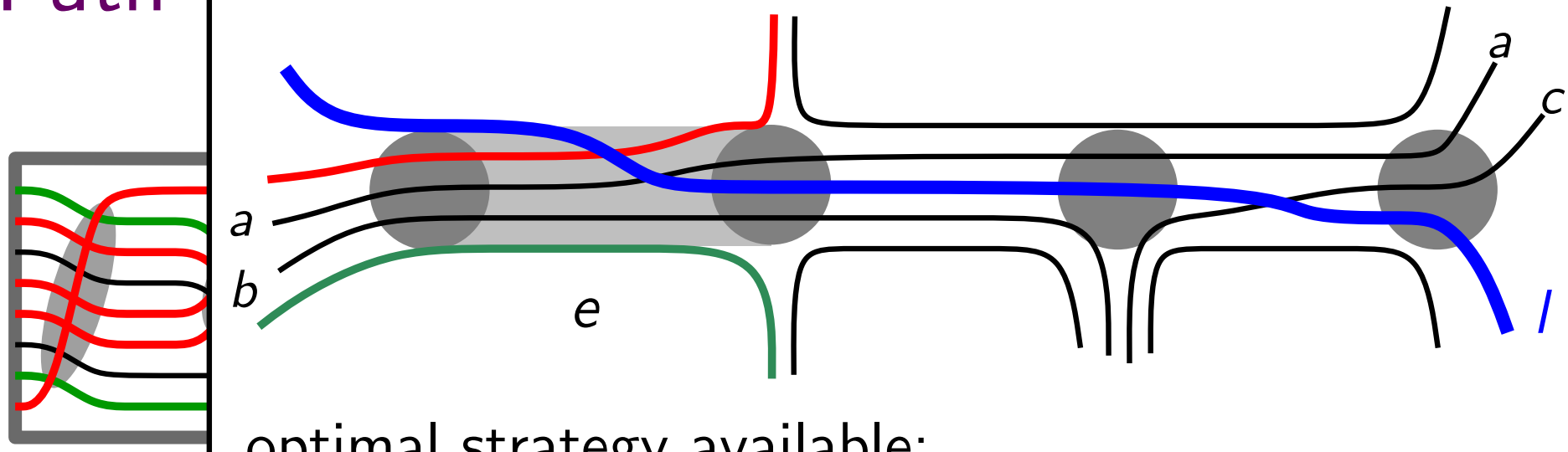


- treat edges from left to right
- bring ending lines to top/bottom keeping good pairs together
- create 1 good pair per block crossing crossing
- optimum creates up to 3

→ 3-approximation

Path -

Special Case: Destroying good pairs



optimal strategy available:

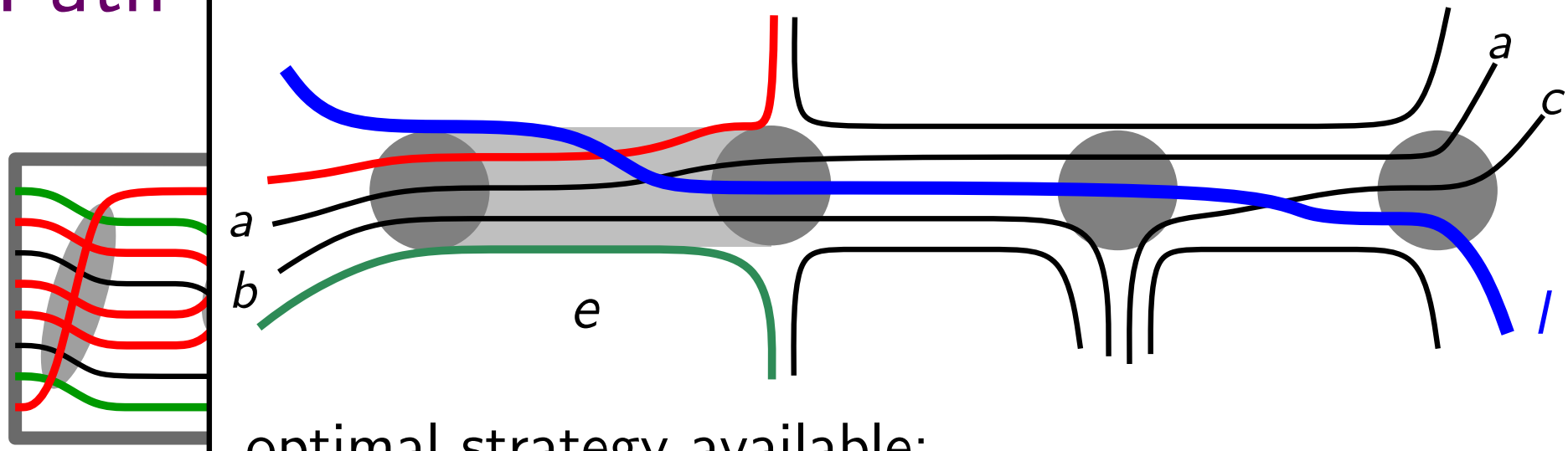
- treat
  right
- bring ending lines to top/bottom keeping good pairs together

- optimum creates up to 3

→ 3-approximation

Path -

Special Case: Destroying good pairs



optimal strategy available:

– preferably destroy non-initial good pair

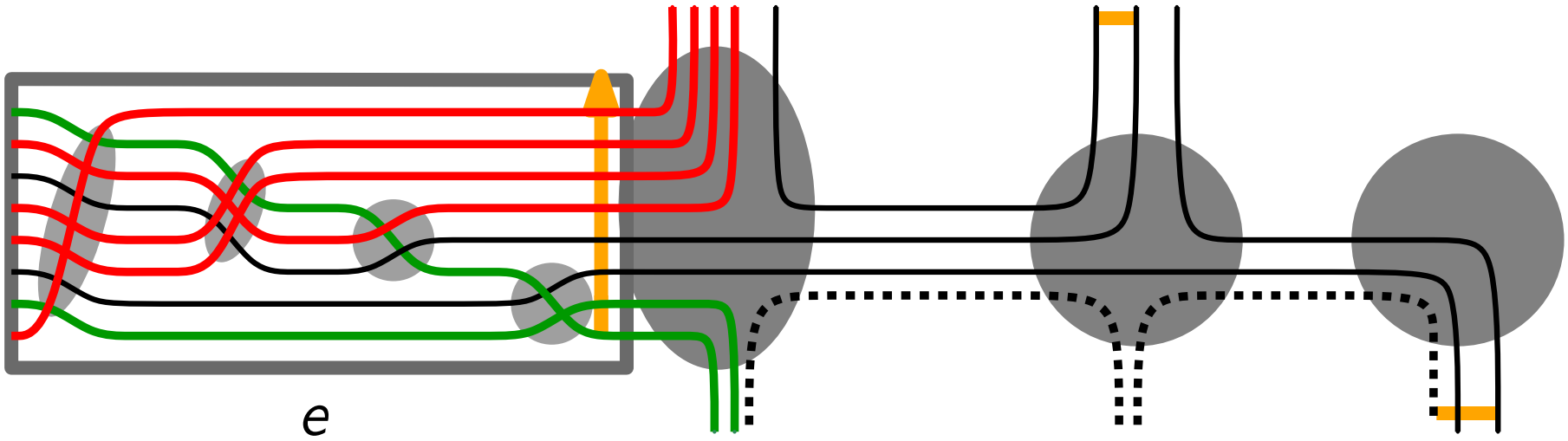
– if none: destroy longest-living good pair

- treat
  right
- bring ending lines to top/bottom keeping good pairs together

- optimum creates up to 3

→ 3-approximation

Path – 3-approximation



- treat edges from left to right
- bring ending lines to top/bottom keeping good pairs together

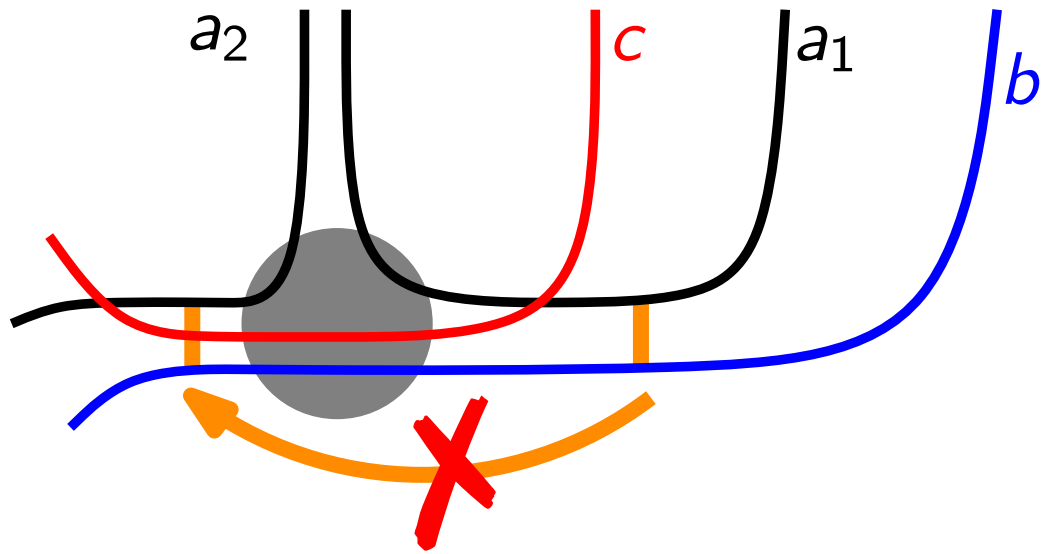
- create 1 good pair per block crossing crossing
- optimum creates up to 3

→ 3-approximation

- algorithm can be adjusted for monotone block crossings

Path – 3-approximation

- adjust inheritance of good pairs:



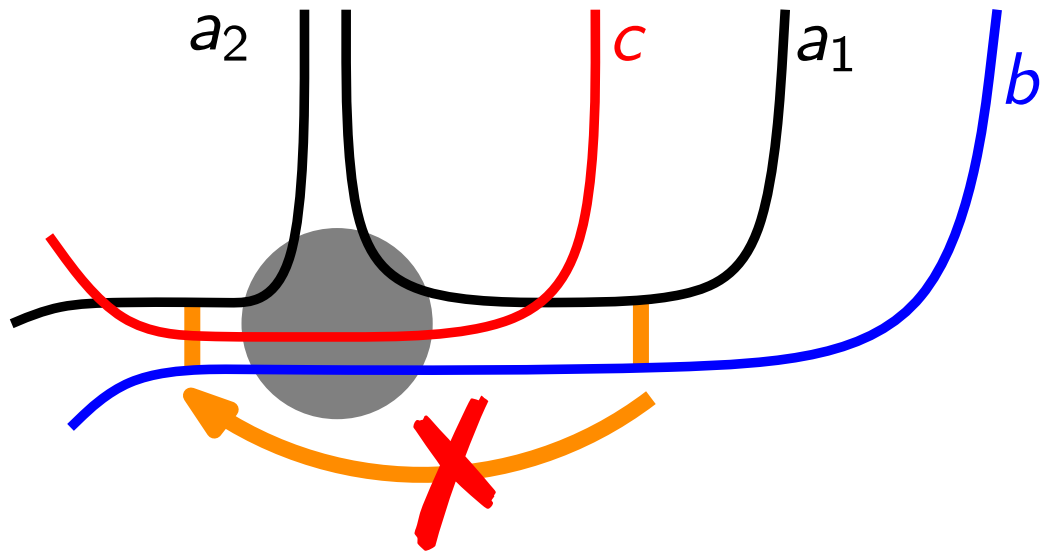
- t… air per
  r… rossing
- b… s up to 3
  top/bottom keeping good pairs together

→ 3-approximation

- algorithm can be adjusted for monotone block crossings

Path – 3-approximation

- adjust inheritance of good pairs:



- new strategy when destroying good pairs: ensure monotonicity

→ 3-approximation

- algorithm can be adjusted for monotone block crossings

- t… air per ri… rossing
- b… s up to 3 top/bottom keeping good pairs together

Trees - an upper bound

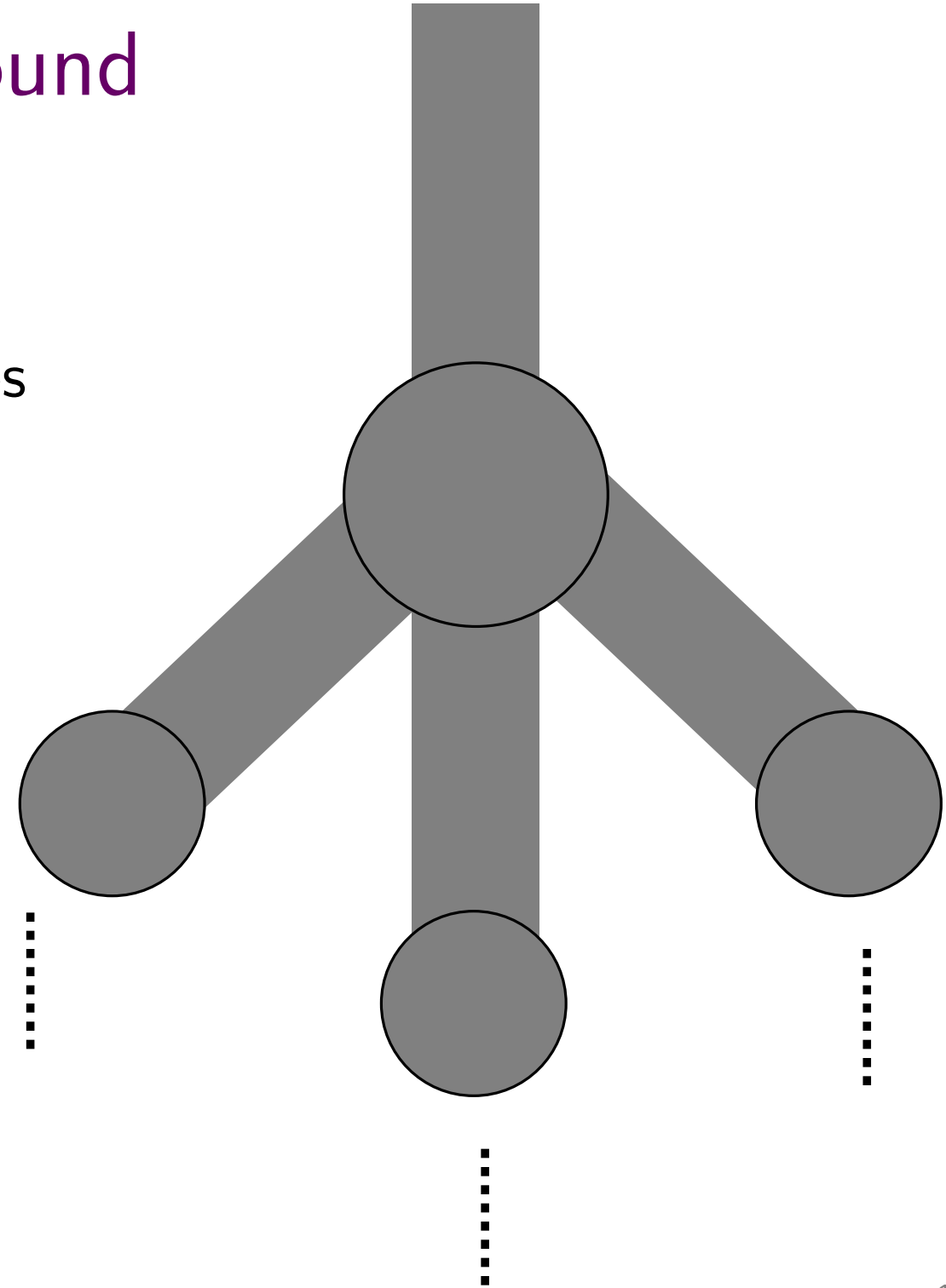
- root at some leave

Trees - an upper bound

- root at some leave
- after treating edge
  recursively order subtrees

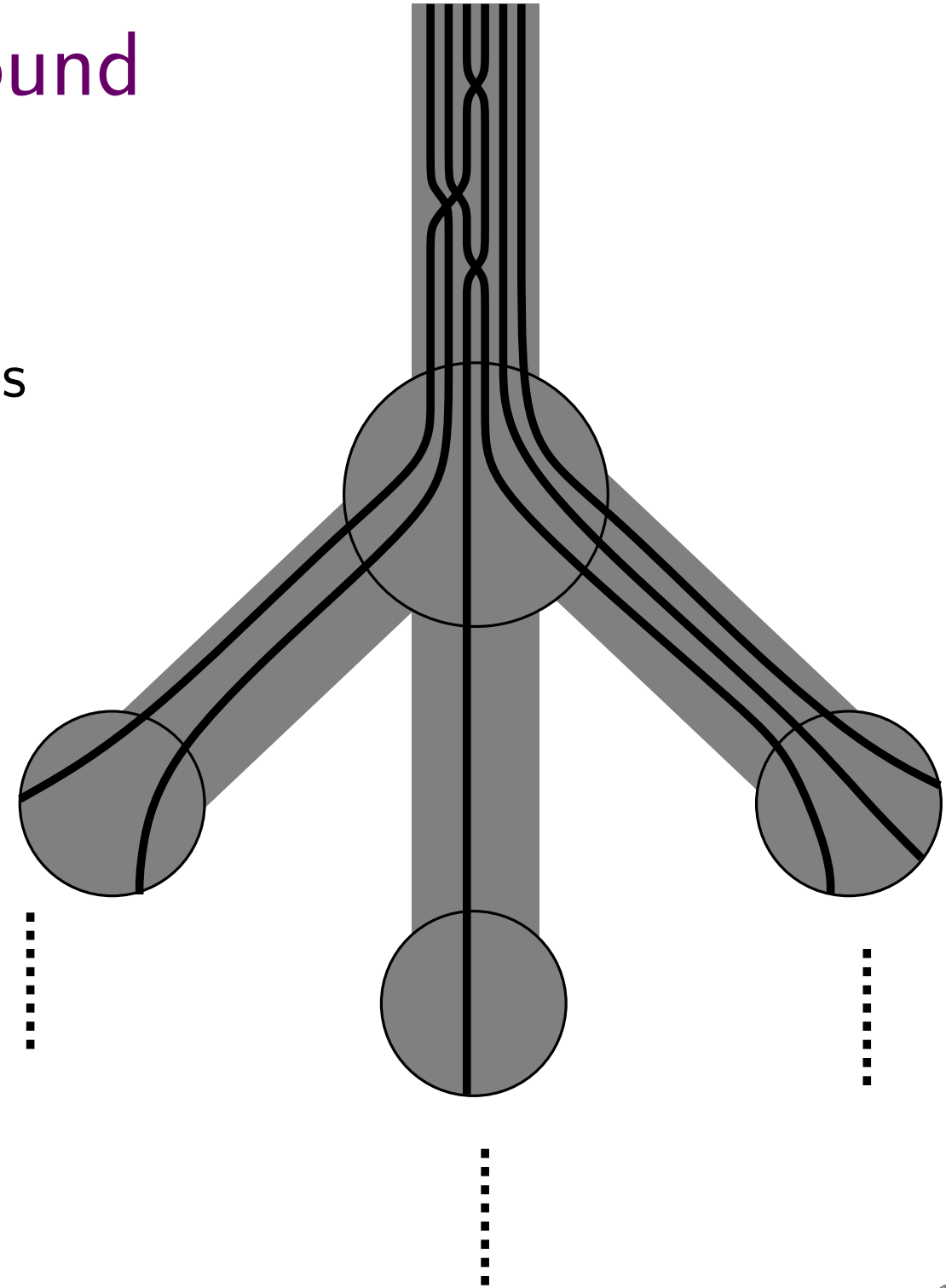
Trees - an upper bound

- root at some leave
- after treating edge
  recursively order subtrees



Trees - an upper bound

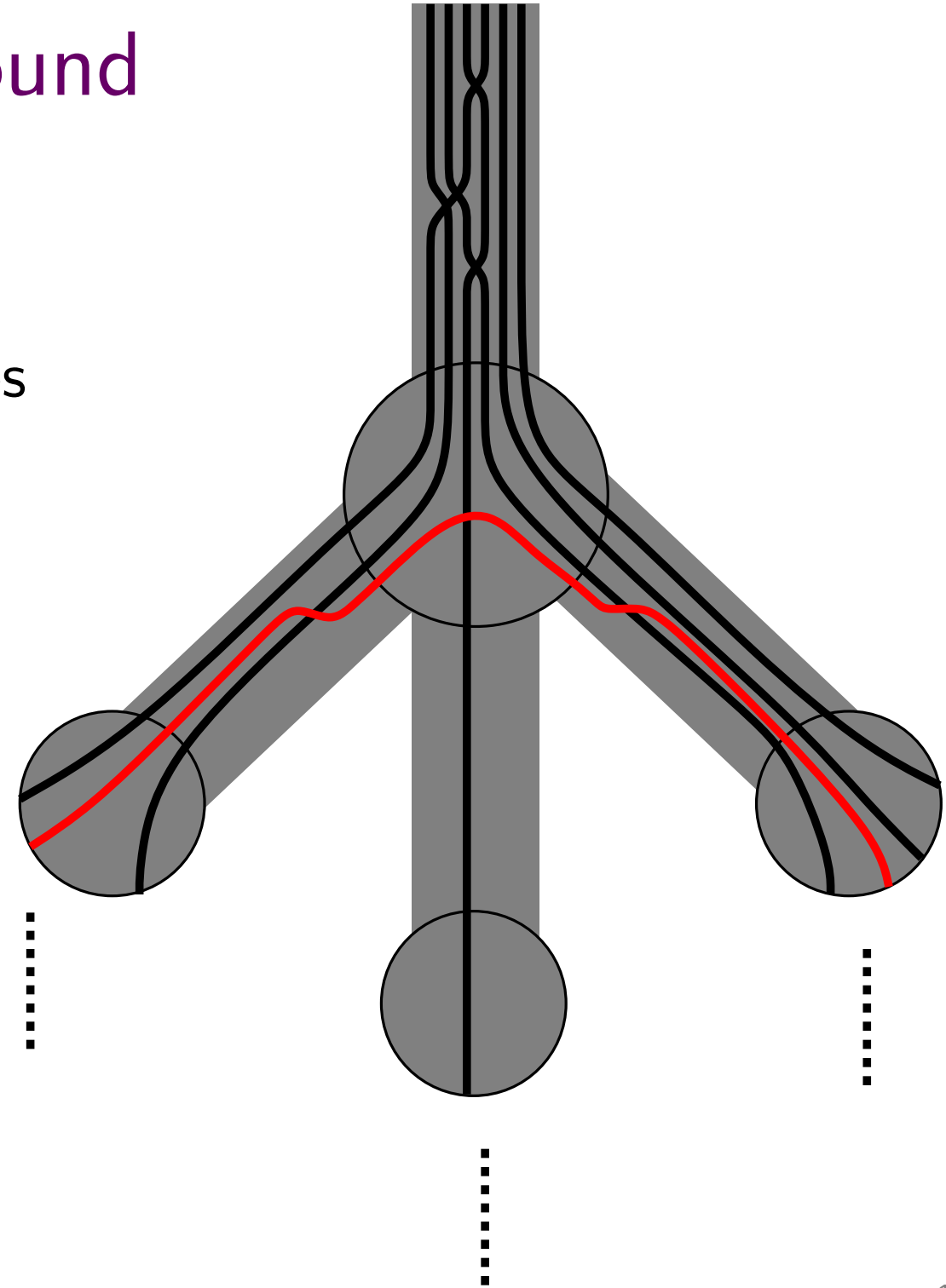
- root at some leave
- after treating edge
  recursively order subtrees



Trees - an upper bound

- root at some leave
- after treating edge
  recursively order subtrees

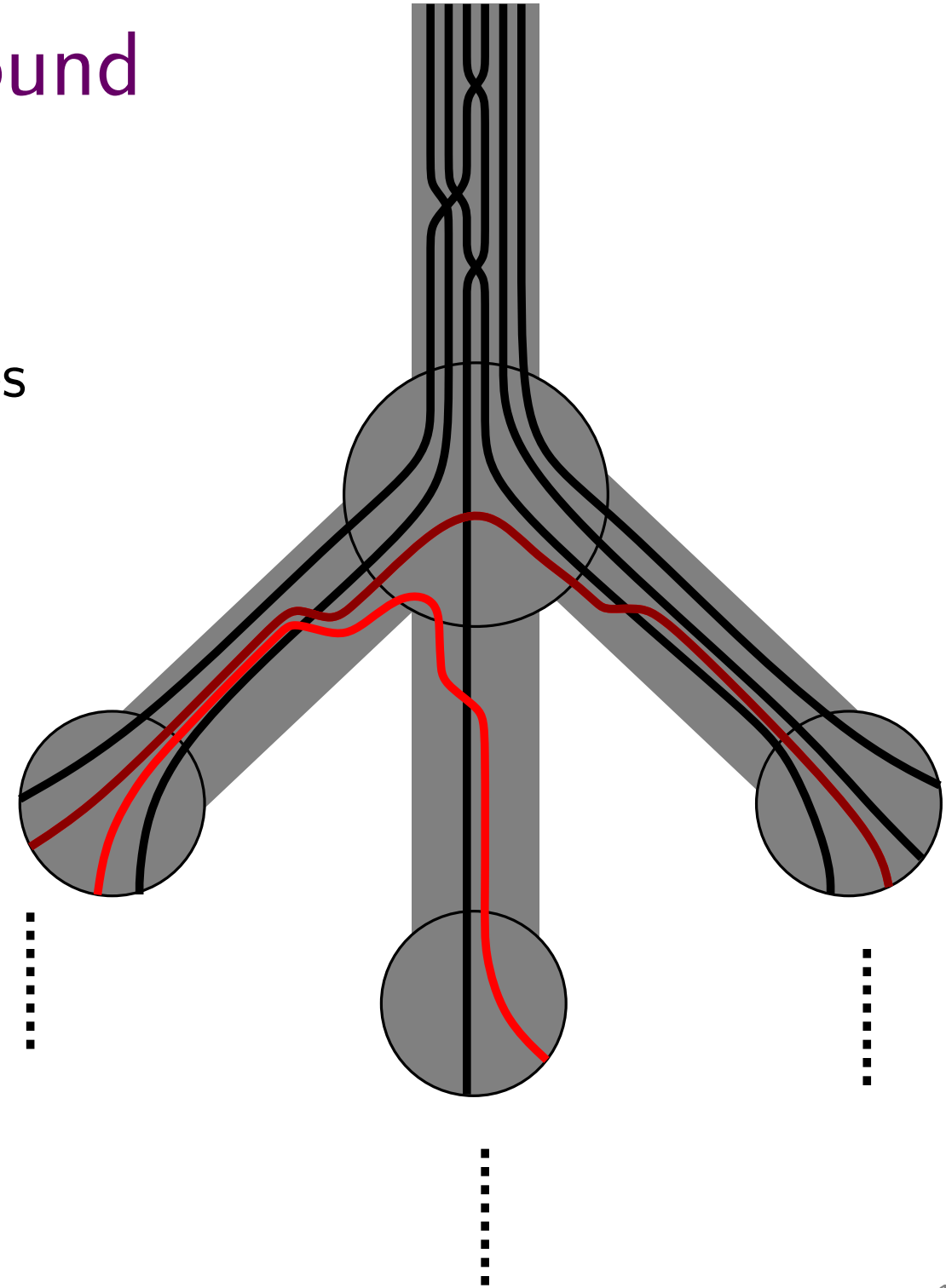
- insert lines between
  subtrees



Trees - an upper bound

- root at some leave
- after treating edge recursively order subtrees

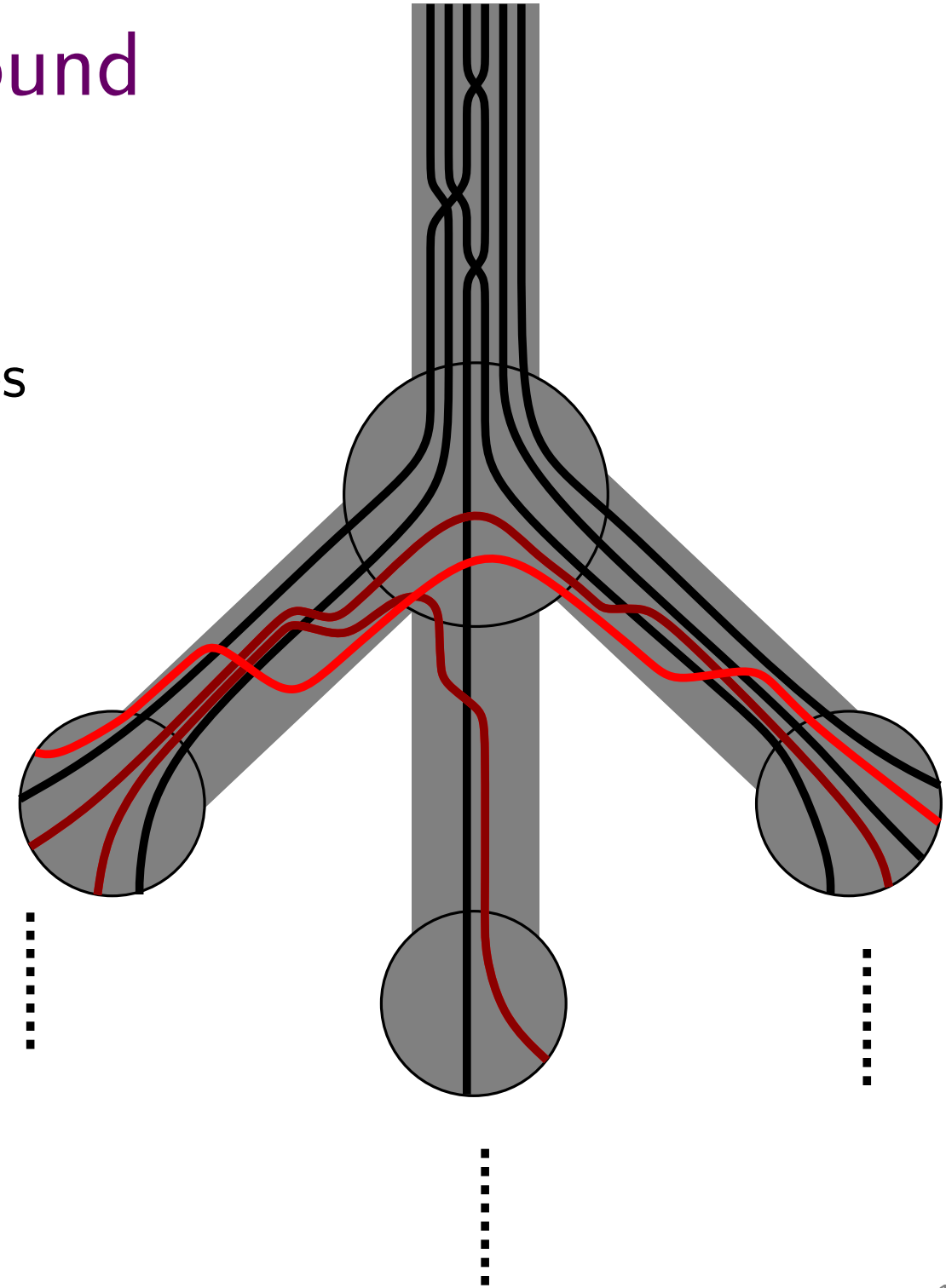
- insert lines between subtrees
- ≤ 2 crossings per line



Trees - an upper bound

- root at some leave
- after treating edge recursively order subtrees

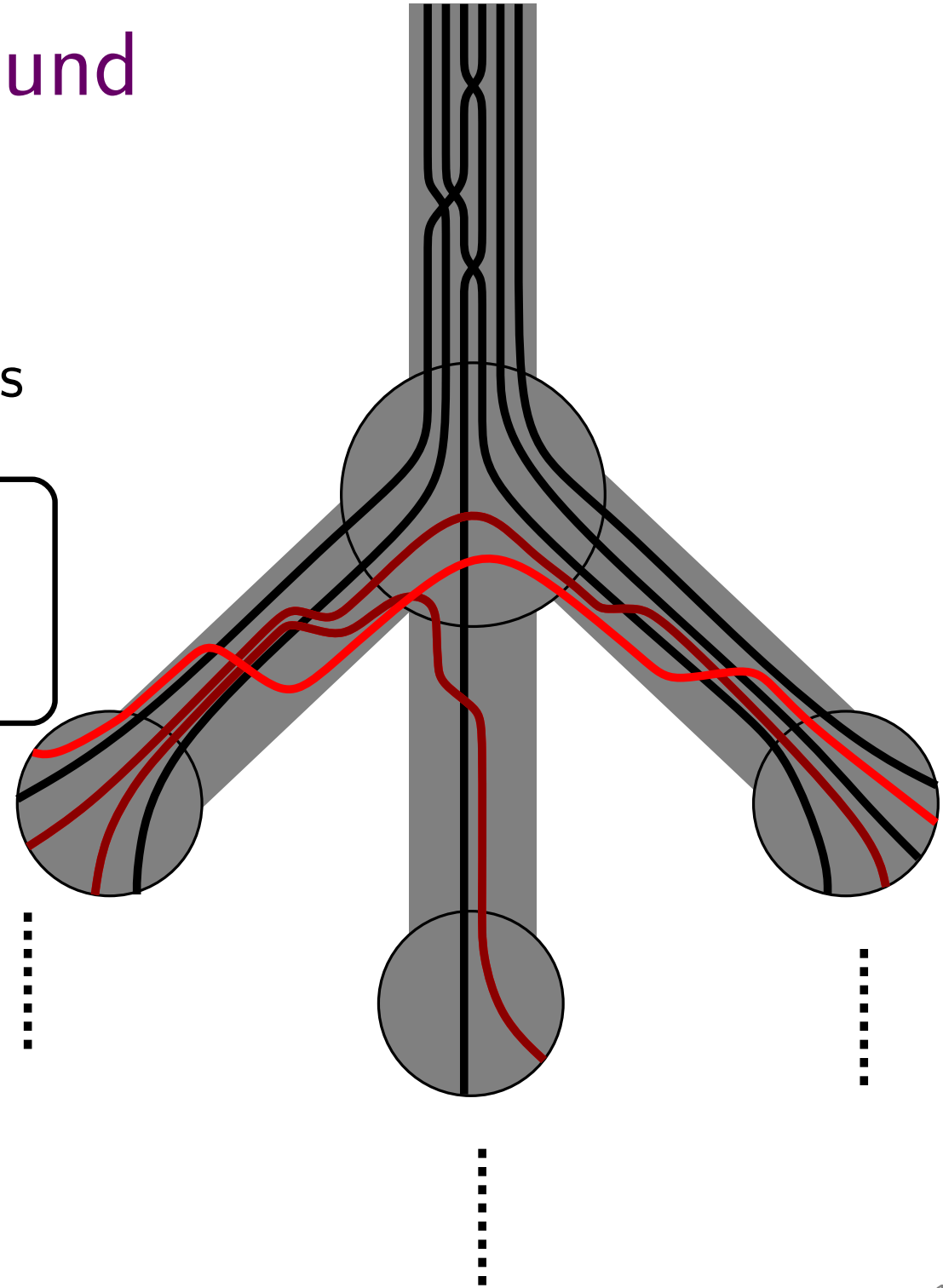
- insert lines between subtrees
- ≤ 2 crossings per line
- right insertion order needed for:
  – avoiding vertex crossings – avoiding double crossings



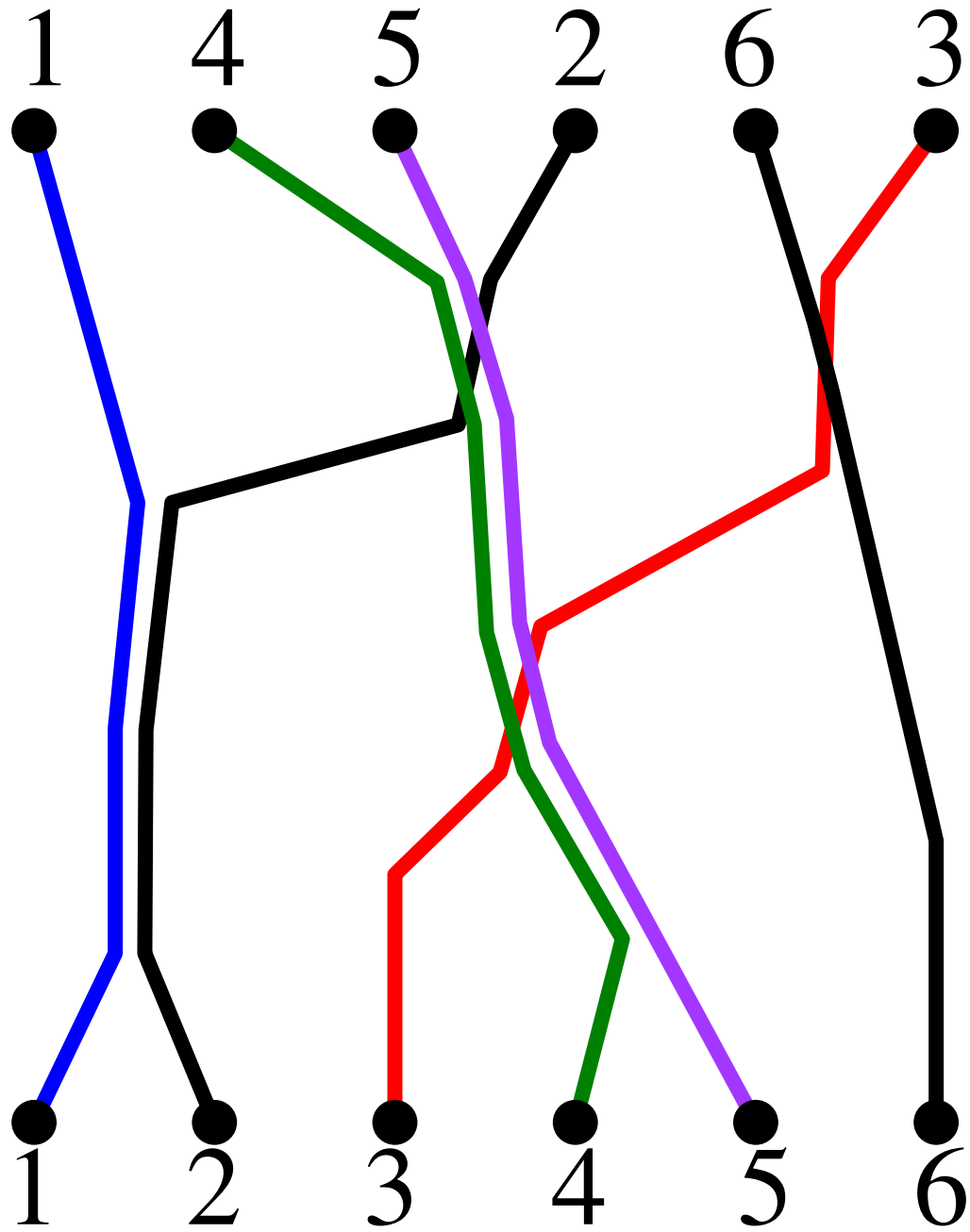
Trees - an upper bound

- root at some leave
- after treating edge recursively order subtrees
- in su
- ≤ 2 crossings per line
- right insertion order needed for:
  – avoiding vertex crossings – avoiding double crossings

worst-case instances:
$2|L| - 3$ crossings necessary

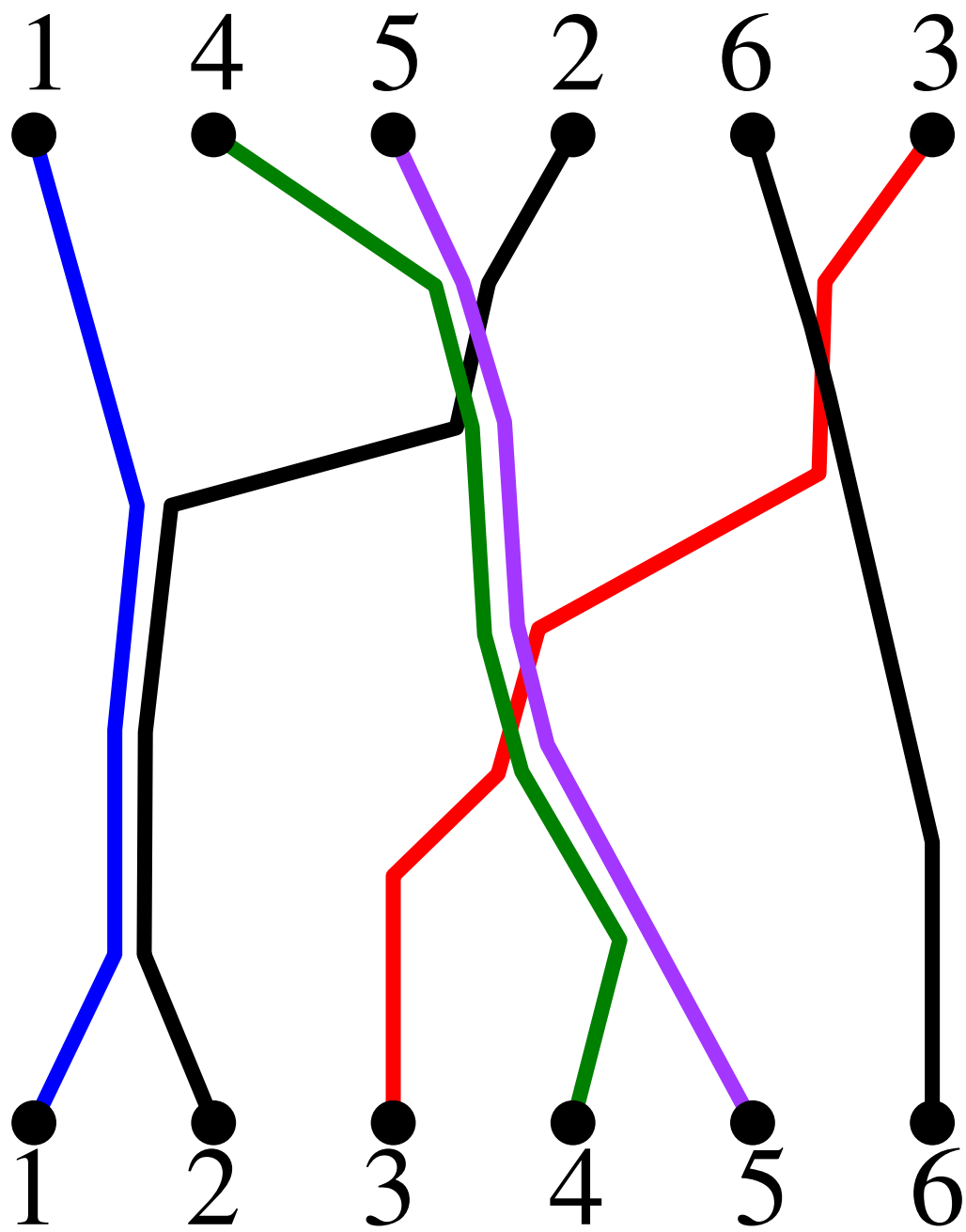


Upward Trees



Upward Trees

- simplification
- use tree algorithm
- 6-approximation for monotone block crossings



General Graphs

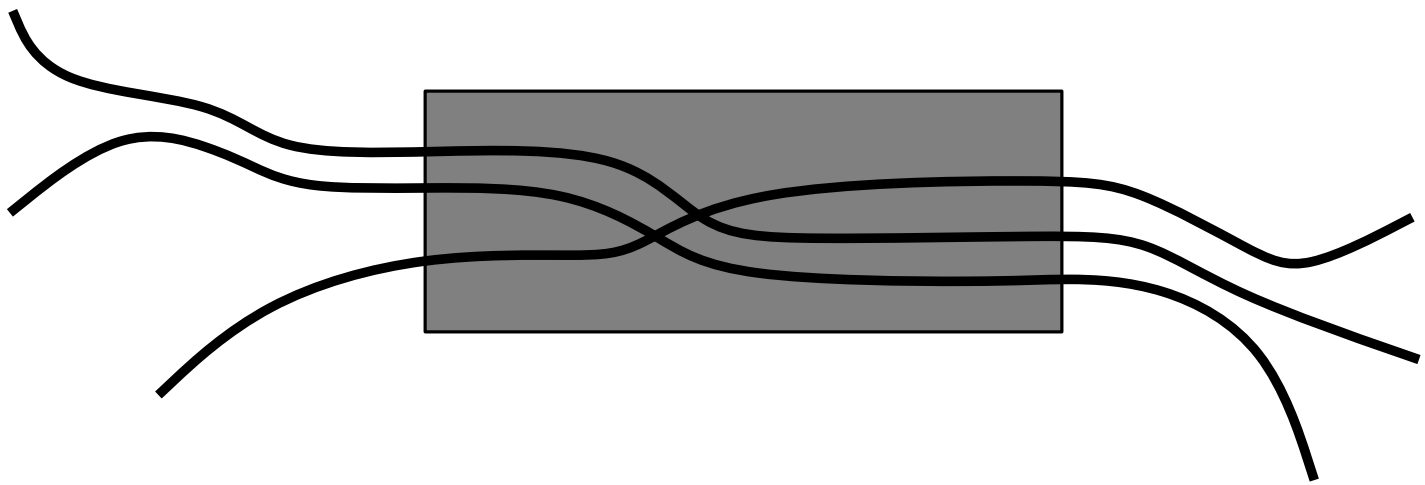
- Process edges in arbitrary order

General Graphs

- Process edges in arbitrary order
- Completely sort lines on an edge

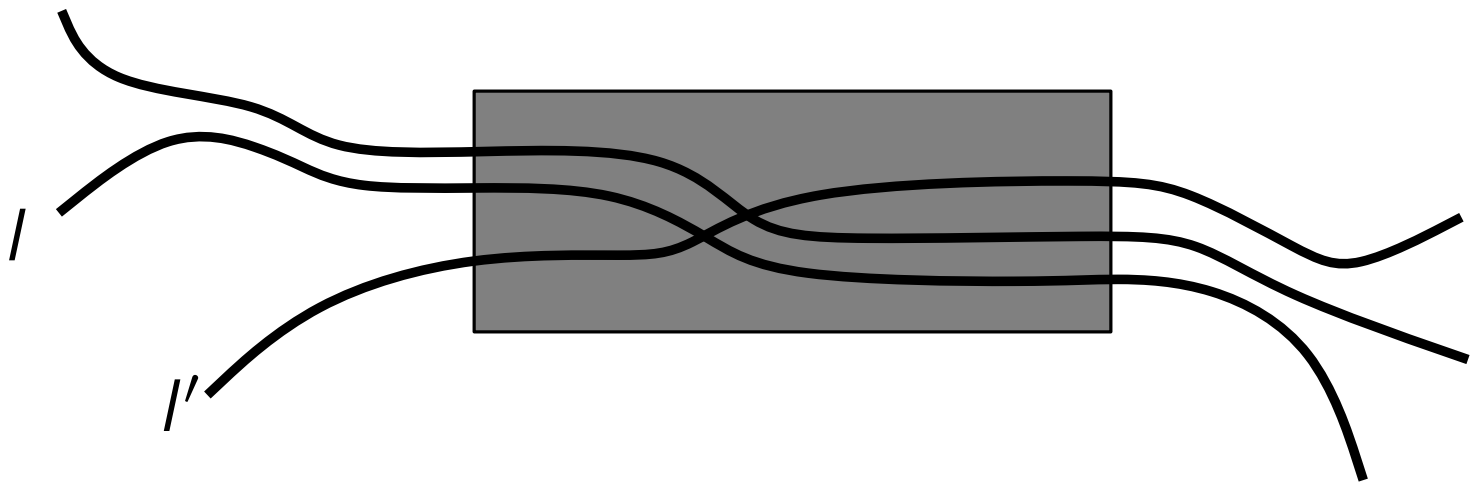
General Graphs

- Process edges in arbitrary order
- Completely sort lines on an edge



General Graphs

- Process edges in arbitrary order
- Completely sort lines on an edge

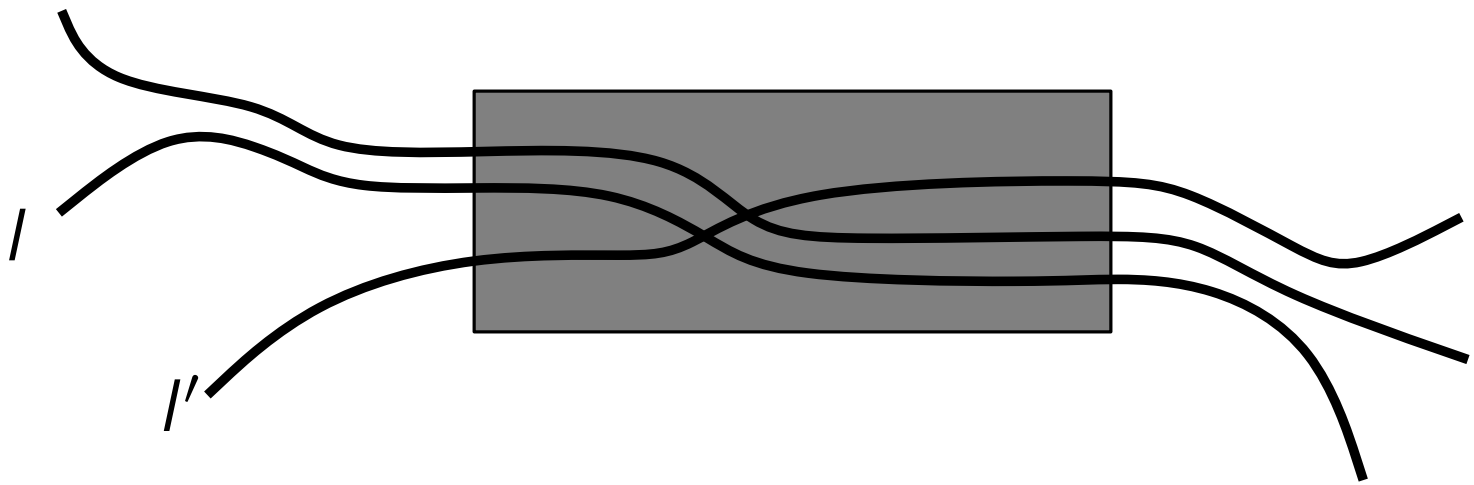


- lines l, l' seen together on edge

  → will never cross (again)

General Graphs

- Process edges in arbitrary order
- Completely sort lines on an edge



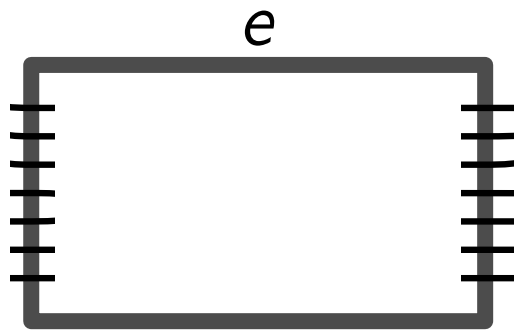
- lines l, l' seen together on edge

  → will never cross (again)

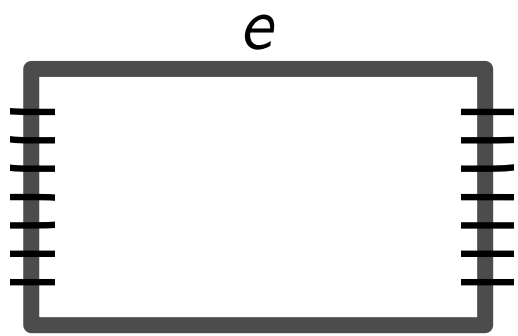
- lines l, l' seen together for the first time

  → information gain

General Graphs – Sorting an Edge

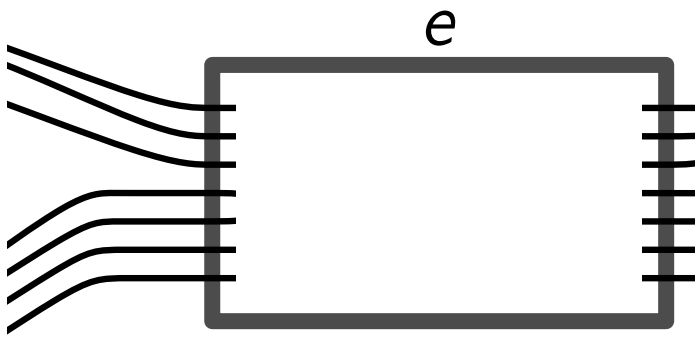


General Graphs – Sorting an Edge



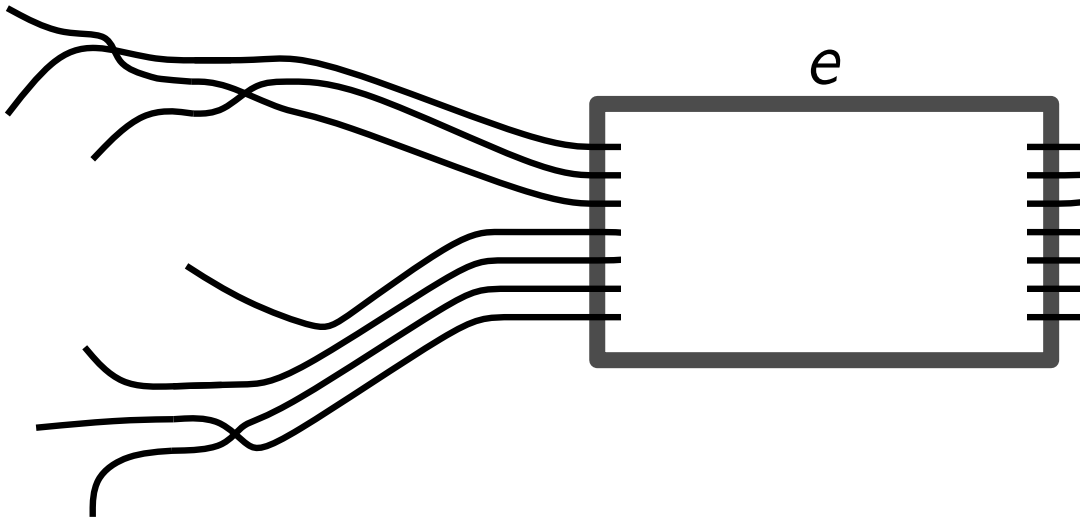
- follow lines

General Graphs – Sorting an Edge



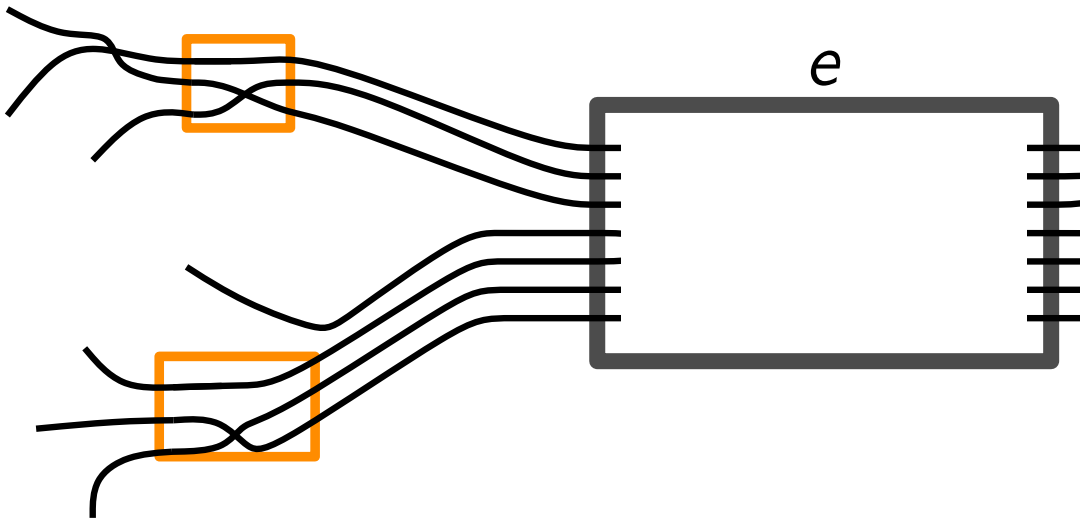
- follow lines

General Graphs – Sorting an Edge



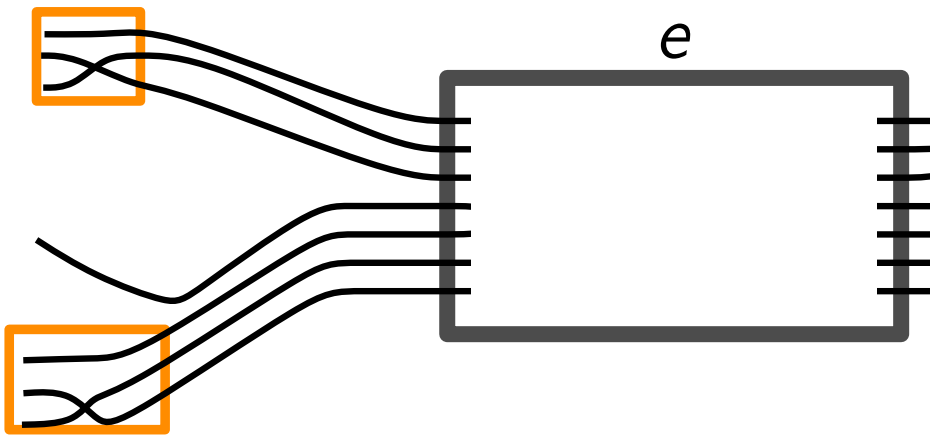
- follow lines

General Graphs – Sorting an Edge



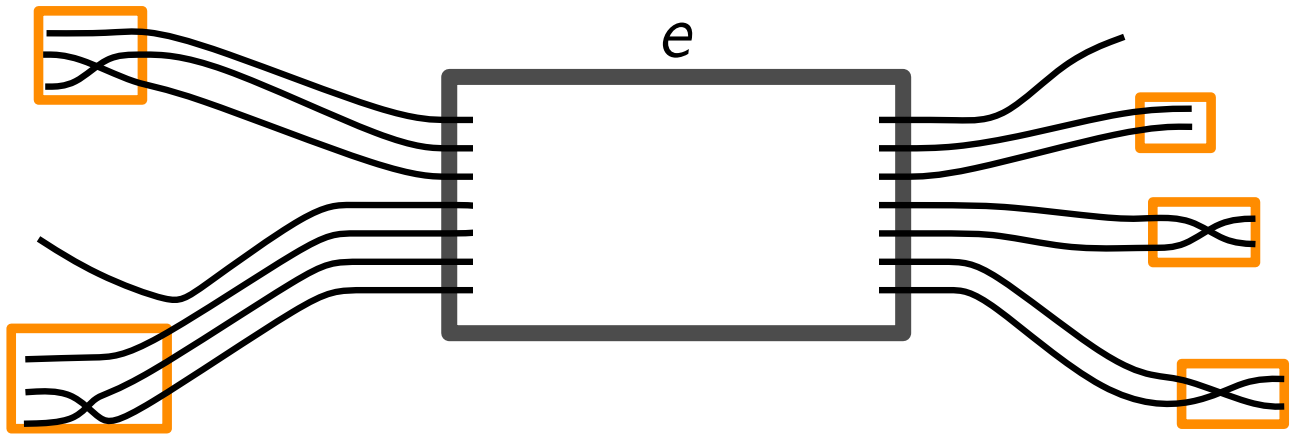
- follow lines
- find cut edges

General Graphs – Sorting an Edge



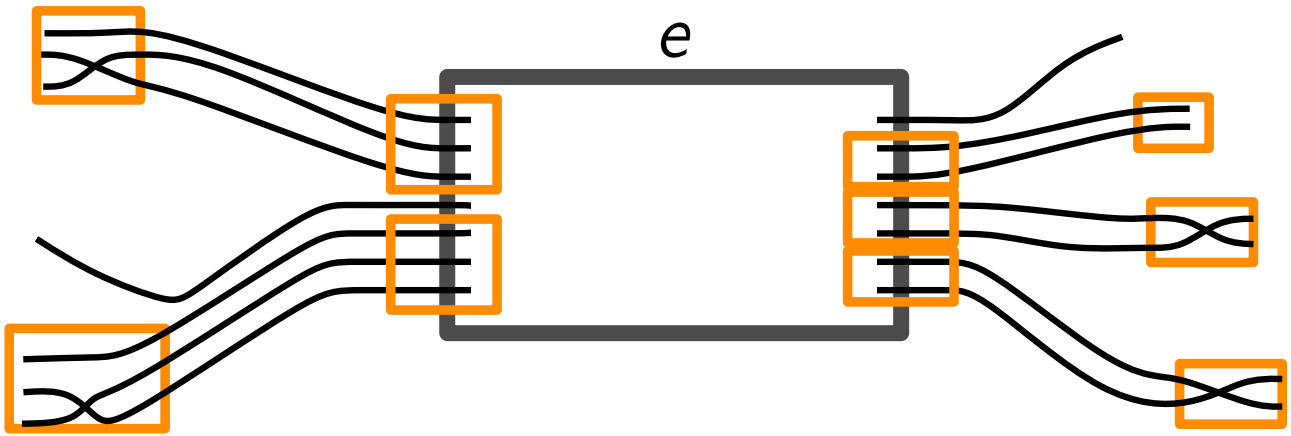
- follow lines
- find cut edges

General Graphs – Sorting an Edge



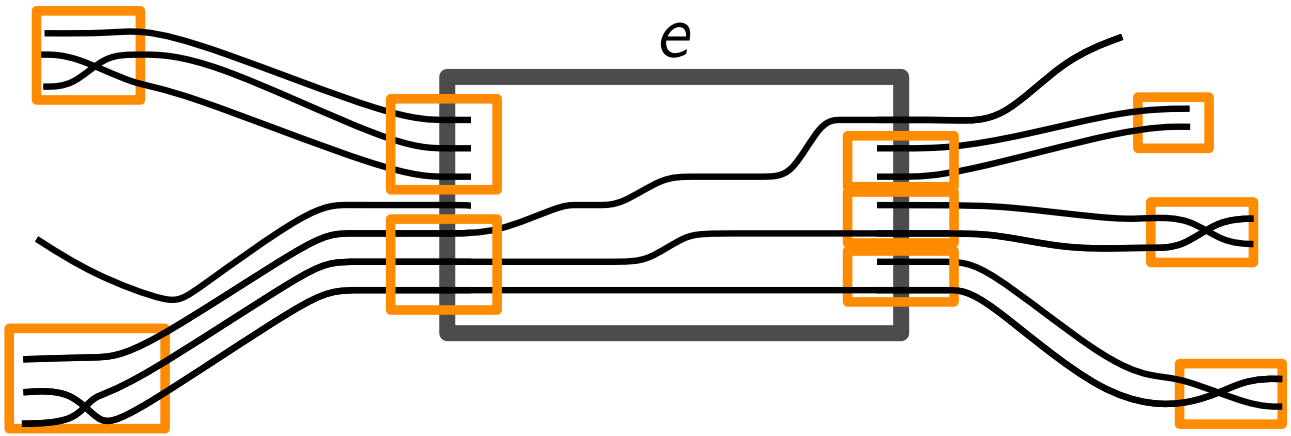
- follow lines
- find cut edges

General Graphs – Sorting an Edge



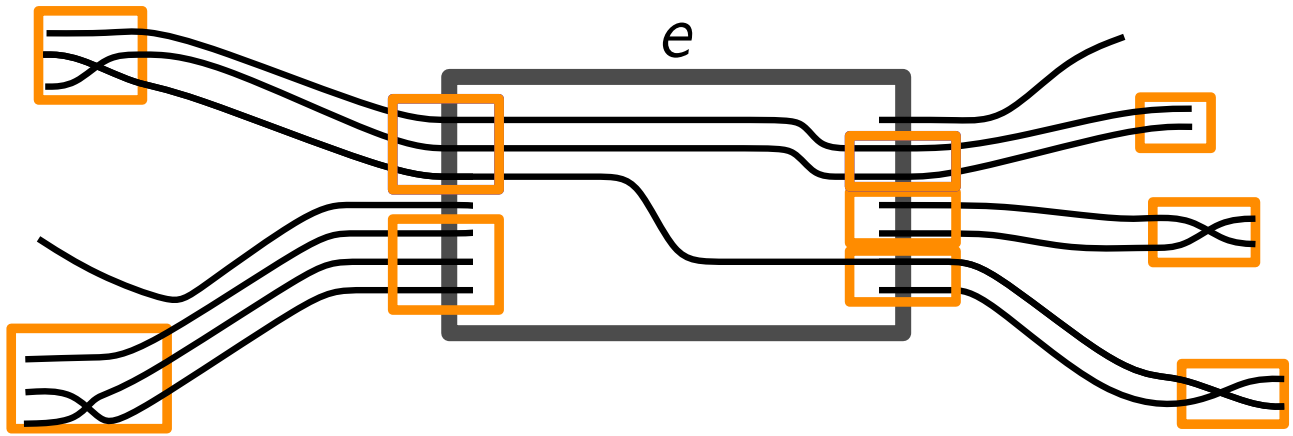
- follow lines
- find cut edges
- identify *groups* of lines

General Graphs – Sorting an Edge



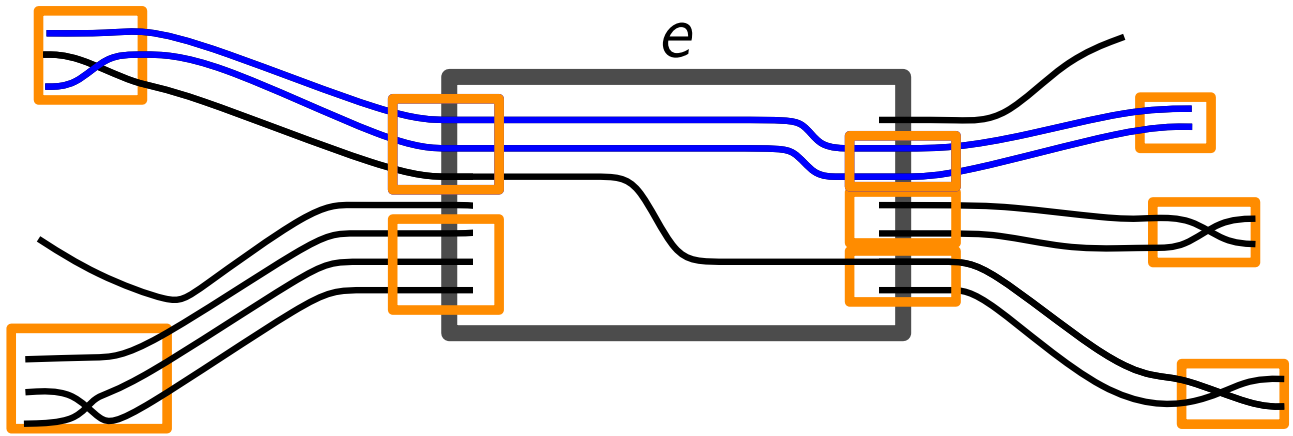
- follow lines
- find cut edges
- identify *groups* of lines
- group stays parallel

General Graphs – Sorting an Edge



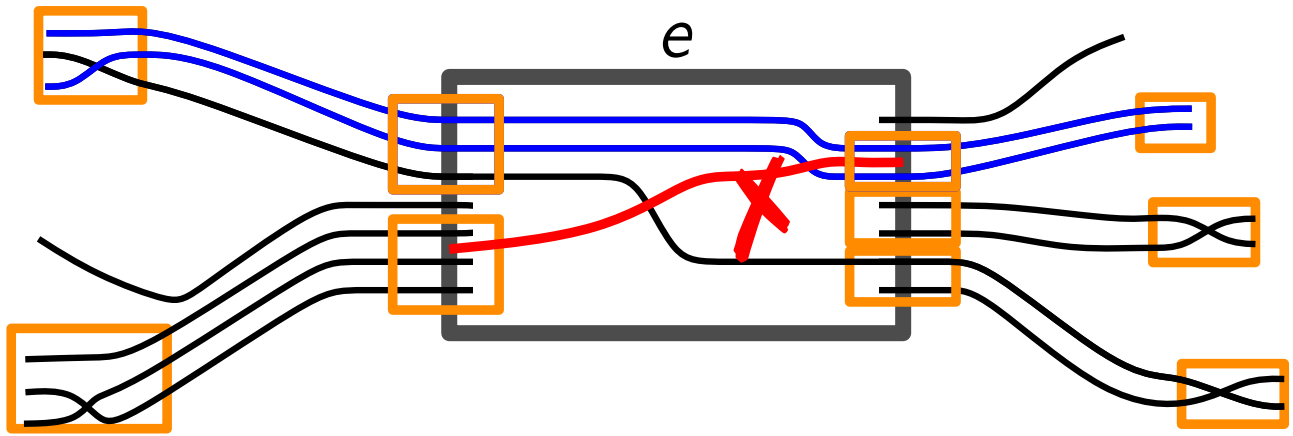
- follow lines
- find cut edges
- identify *groups* of lines
- group stays parallel
- consider pairs of groups

General Graphs – Sorting an Edge



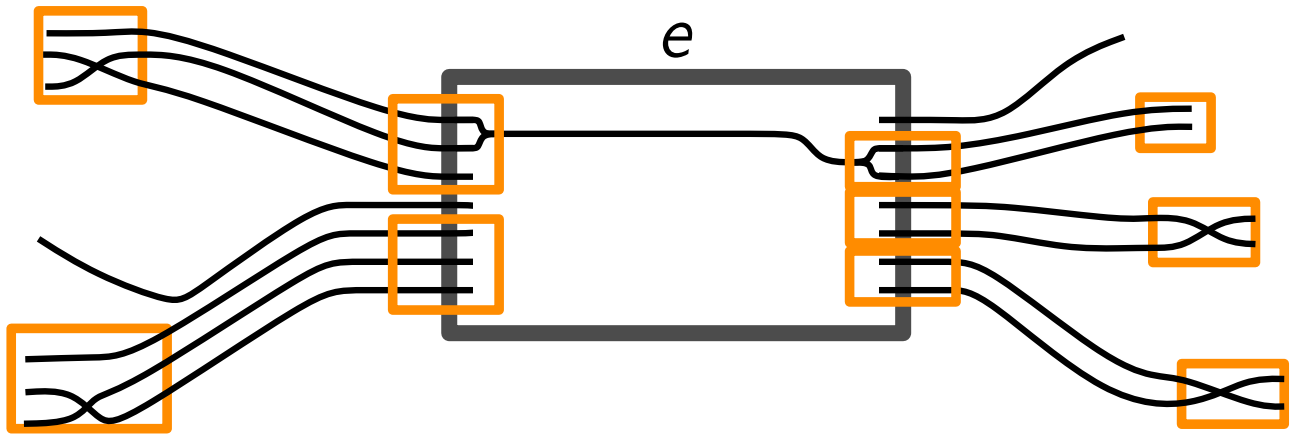
- follow lines
- find cut edges
- identify *groups* of lines
- group stays parallel
- consider pairs of groups

General Graphs – Sorting an Edge



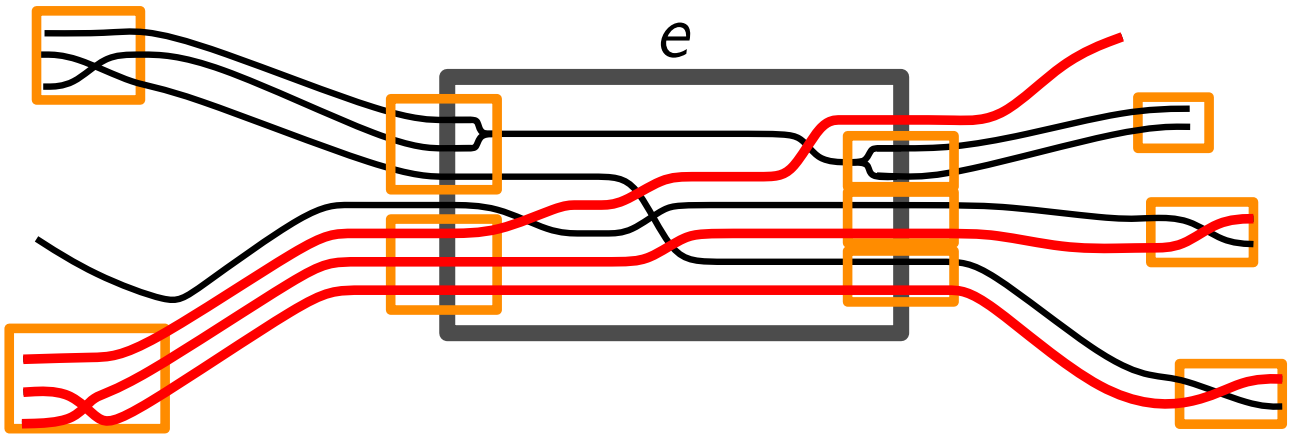
- follow lines
- find cut edges
- identify *groups* of lines
- group stays parallel
- consider pairs of groups

General Graphs – Sorting an Edge



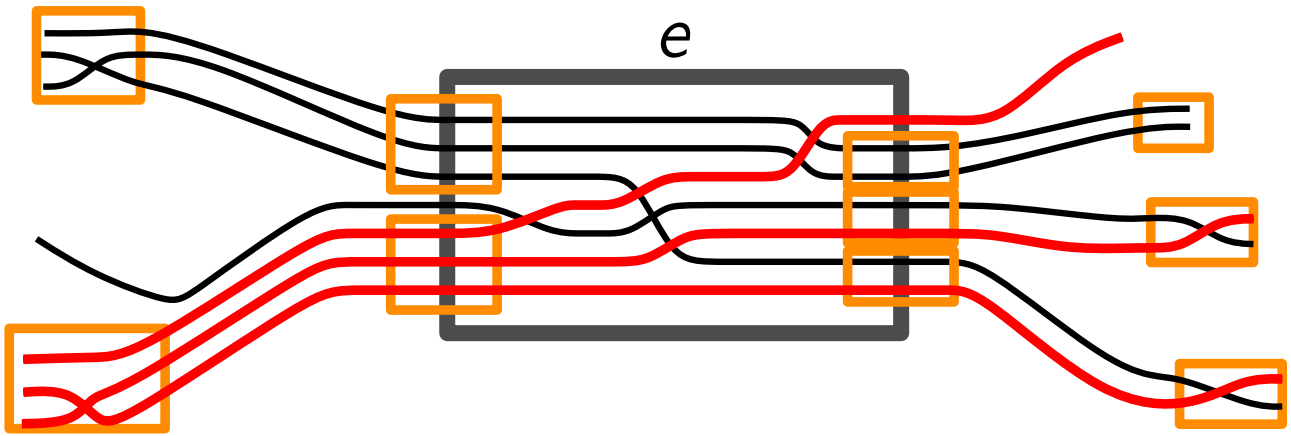
- follow lines
- find cut edges
- identify *groups* of lines
- group stays parallel
- consider pairs of groups
- merge lines if possible

General Graphs – Sorting an Edge



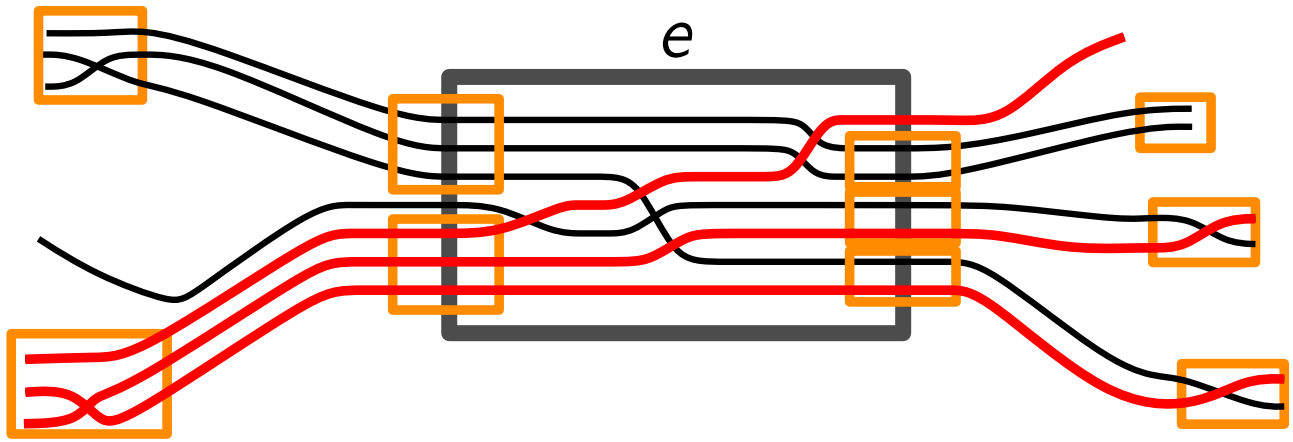
- follow lines
- find cut edges
- identify *groups* of lines
- group stays parallel
- consider pairs of groups
- merge lines if possible
- sort by insertion into largest group

General Graphs – Sorting an Edge



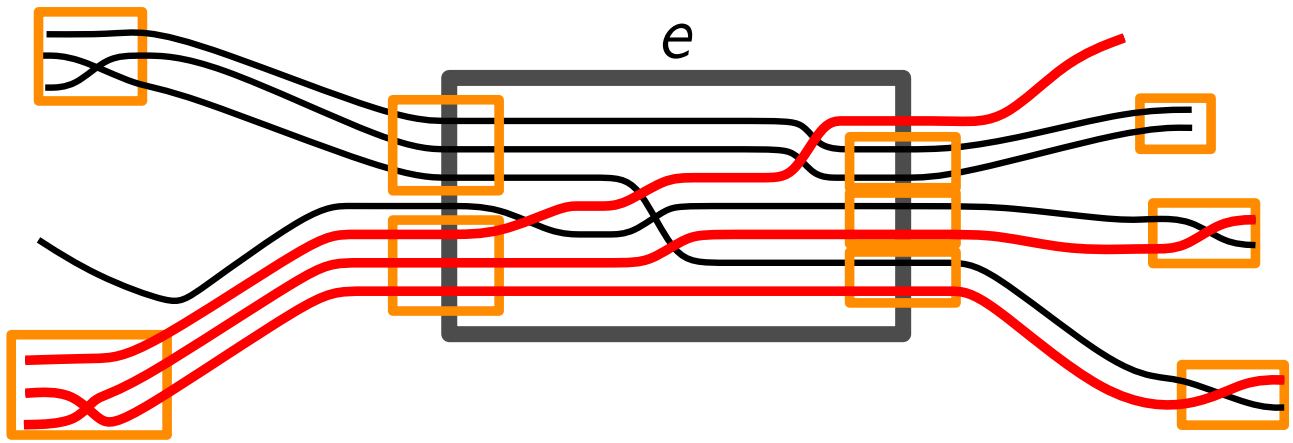
- follow lines
- find cut edges
- identify *groups* of lines
- group stays parallel
- consider pairs of groups
- merge lines if possible
- sort by insertion into largest group
- undo merging

General Graphs – Analysis



- finally: all edges ordered

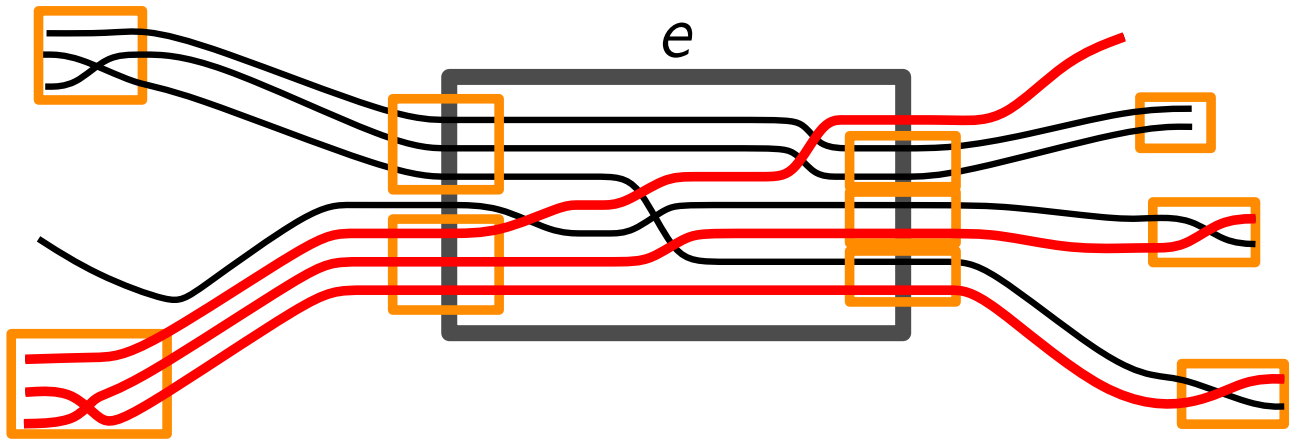
General Graphs – Analysis



- finally: all edges ordered
- pairs of lines cross at most once

monotone block crossings

General Graphs – Analysis



- finally: all edges ordered
- pairs of lines cross at most once

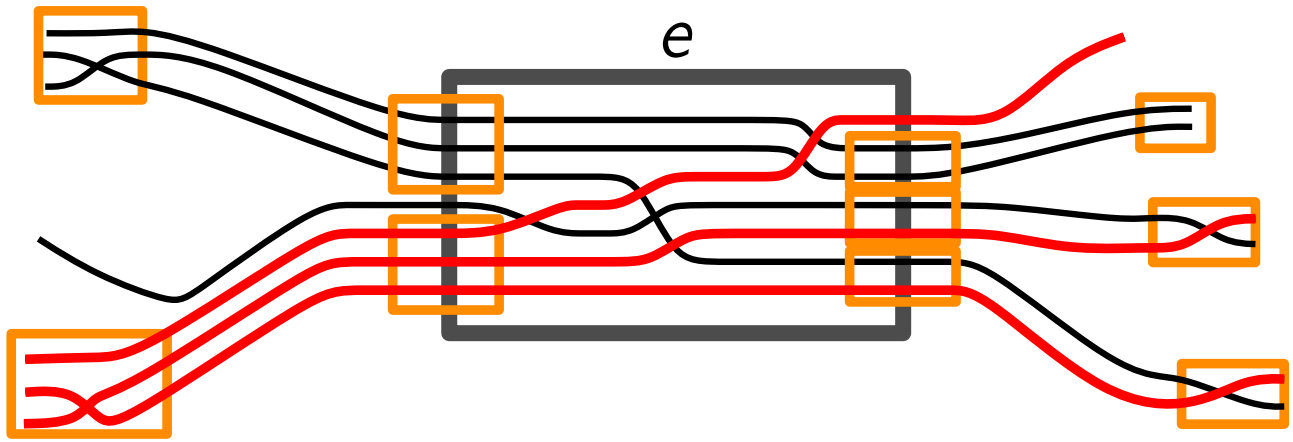
monotone block crossings

- $(\mathrm{bc}(e))^2 \leq I(e)$

information gain

block crossings on edge

General Graphs – Analysis



- finally: all edges ordered
- pairs of lines cross at most once

monotone block crossings

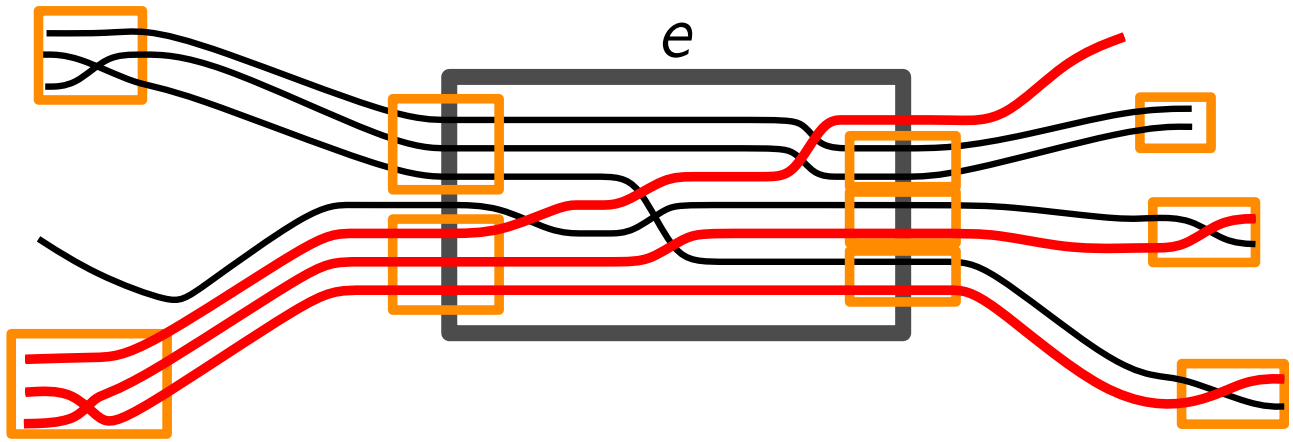
- $(\mathrm{bc}(e))^2 \leq I(e)$

information gain

block crossings on edge

- $\sum_{e \in E} (\mathrm{bc}(e))^2 \leq |L|^2$

General Graphs – Analysis



- finally: all edges ordered
- pairs of lines cross at most once

monotone block crossings

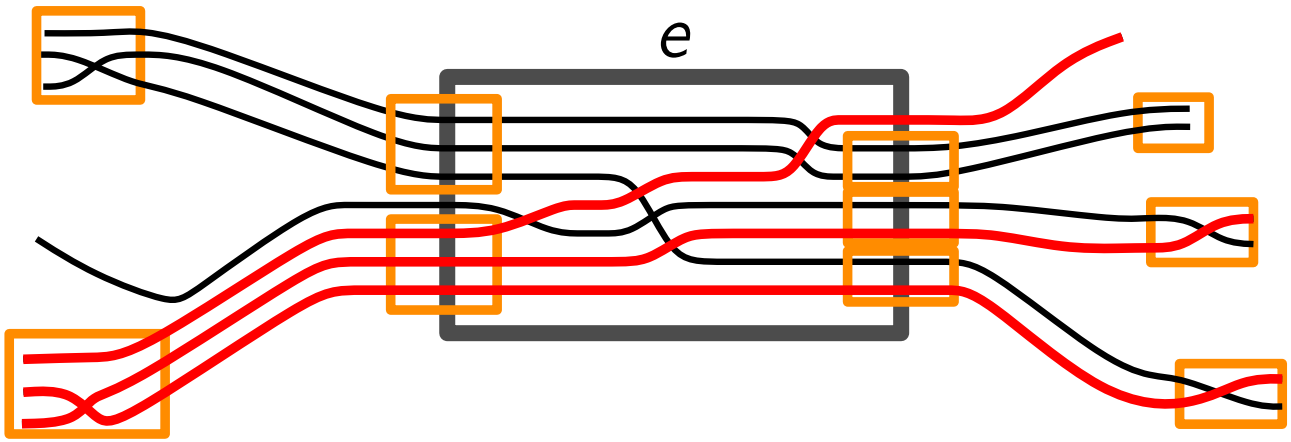
- $(\mathrm{bc}(e))^2 \leq I(e)$

information gain

block crossings on edge

- $\sum_{e \in E} (\mathrm{bc}(e))^2 \leq |L|^2$
- Using Cauchy-Schwarz: $\sum_{e \in E} \mathrm{bc}(e) \leq |L| \sqrt{|E'|}$

General Graphs – Analysis



- finally: all edges ordered
- pairs of lines cross at most once

worst-case instances: $\Omega(|L|\sqrt{|E'|})$ block crossings necessary

- $(\mathrm{bc}(e))^2 \leq I(e)$
  - information gain
  - block crossings on edge
- $\sum_{e \in E} (\mathrm{bc}(e))^2 \leq |L|^2$
- Using Cauchy-Schwarz: $\sum_{e \in E} \mathrm{bc}(e) \leq |L|\sqrt{|E'|}$

Conclusion

- new model for counting crossings
- approximations for paths and upward trees
- tight upper bound for trees
- algorithm for general graphs
- upper bound asymptotically tight

Conclusion

- new model for counting crossings
- approximations for paths and upward trees
- tight upper bound for trees
- algorithm for general graphs
- upper bound asymptotically tight

Open Questions:

- Complexity of monotone block crossings on a single edge?
- Approximations for trees / general graphs?

Conclusion

- new model for counting crossings
- approximations for paths and upward trees
- tight upper bound for trees
- algorithm for general graphs
- upper bound asymptotically tight

Open Questions:

- Complexity of monotone block crossings on a single edge?
- Approximations for trees / general graphs?

Thank you!