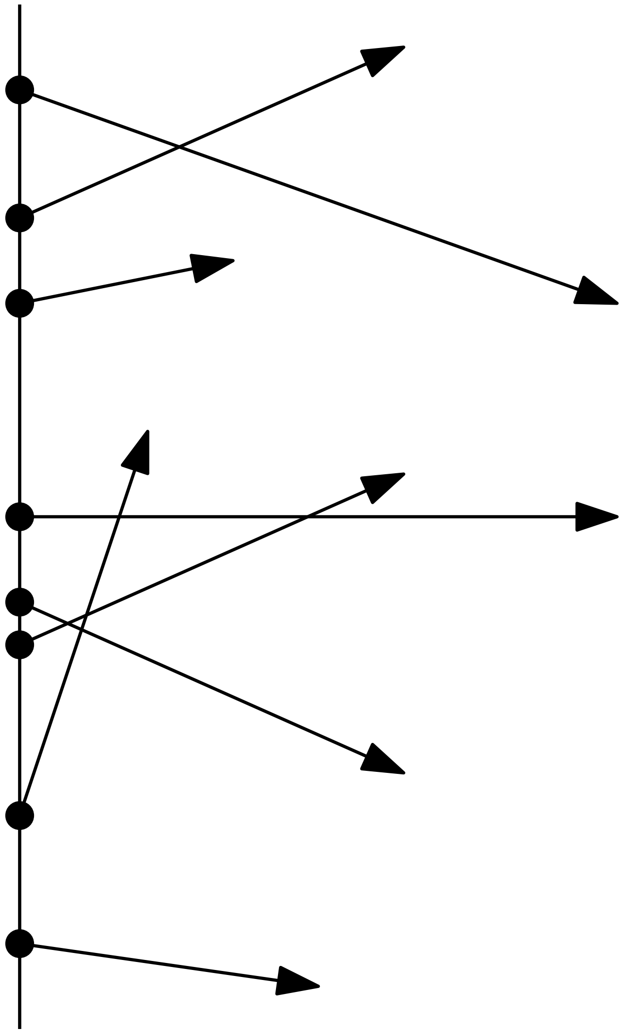
Motorcycles

A group of bikers start at the same time at unit speed. If a biker runs into the tracks of another biker it stops. Tiebreaks (right-hand rule).

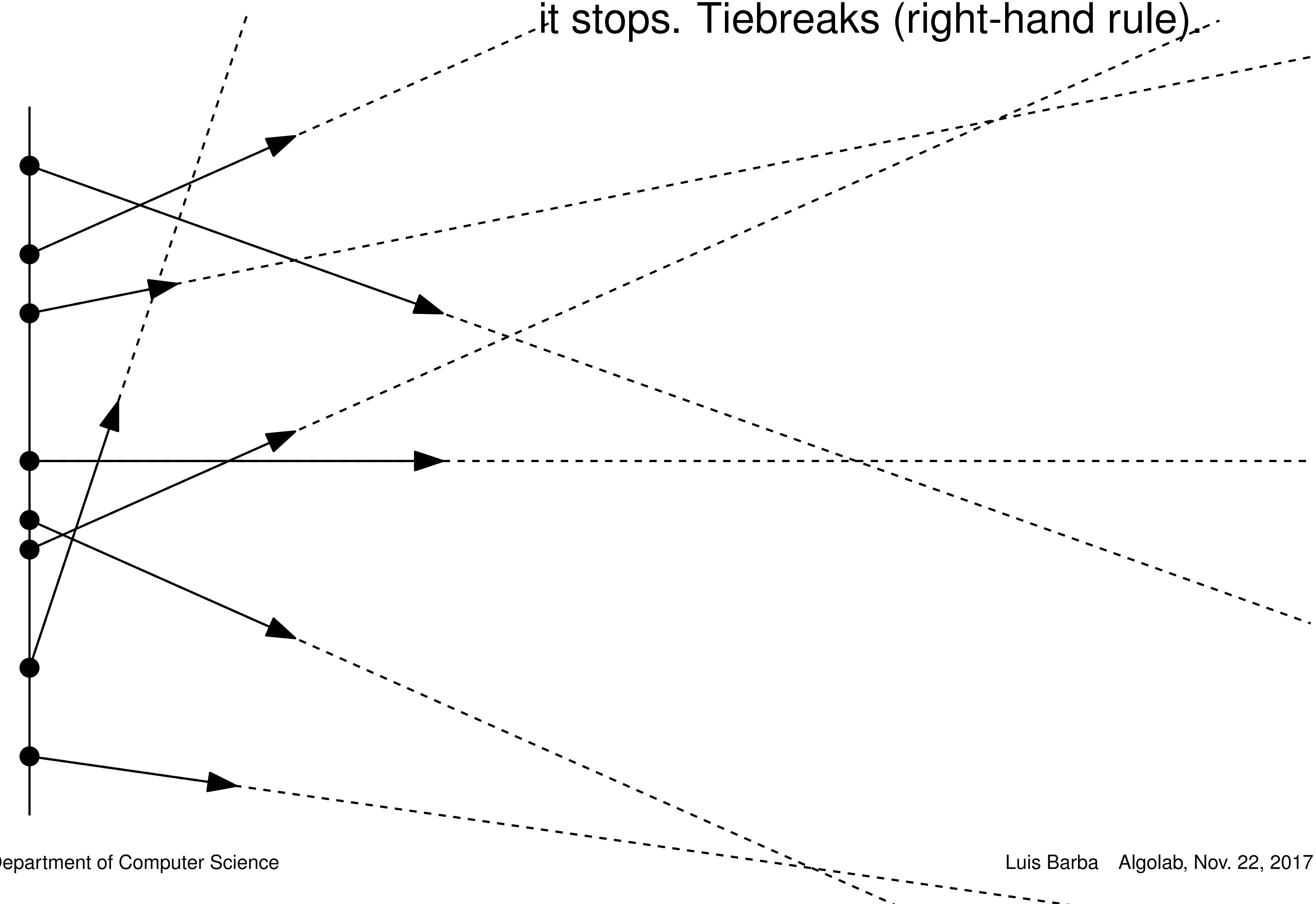
Motorcycles

A group of bikers start at the same time at unit speed. If a biker runs into the tracks of another biker it stops. Tiebreaks (right-hand rule).



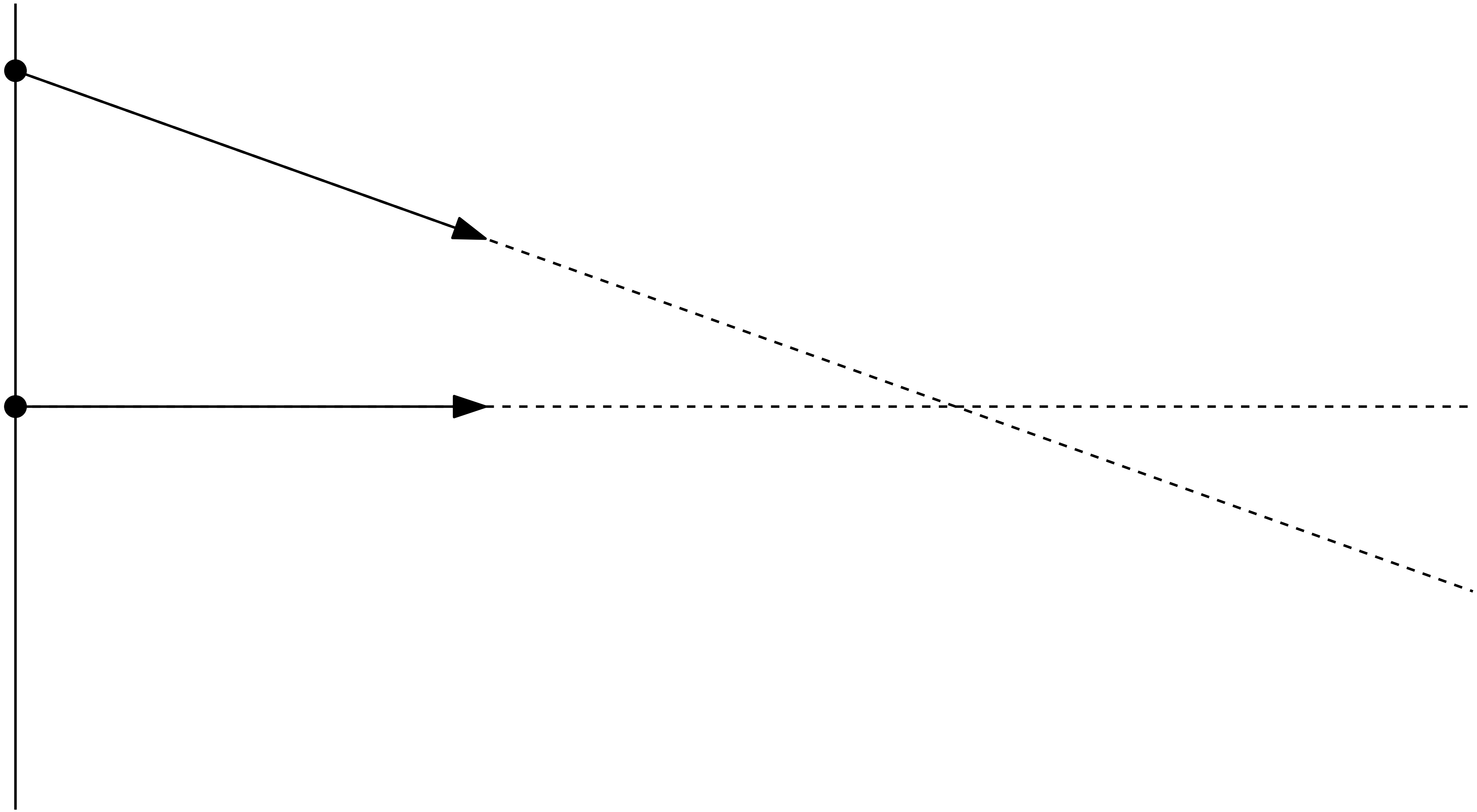
Motorcycles

A group of bikers start at the same time at unit speed. If a biker runs into the tracks of another biker it stops. Tiebreaks (right-hand rule).



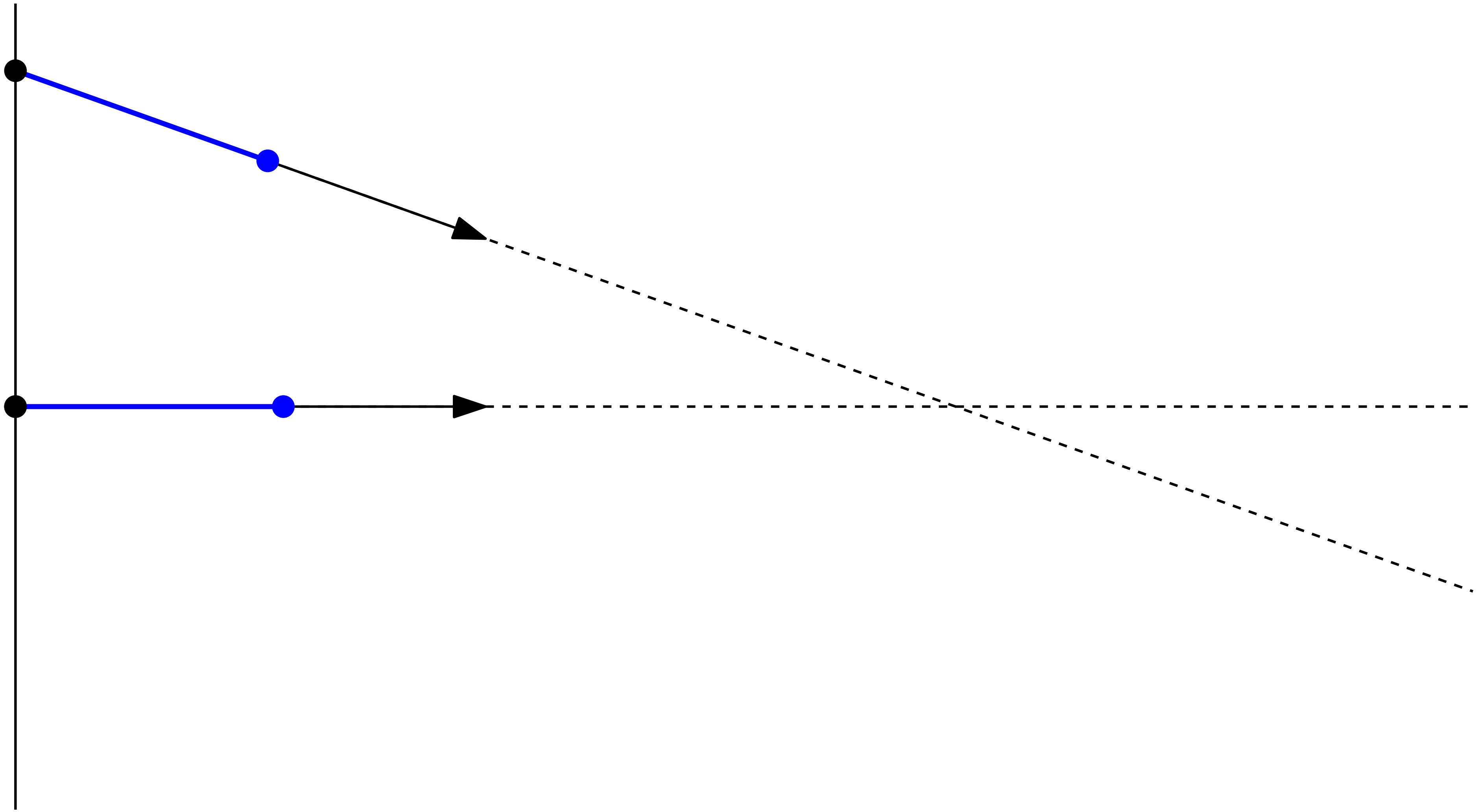
Motorcycles

A group of bikers start at the same time at unit speed. If a biker runs into the tracks of another biker it stops. Tiebreaks (right-hand rule).



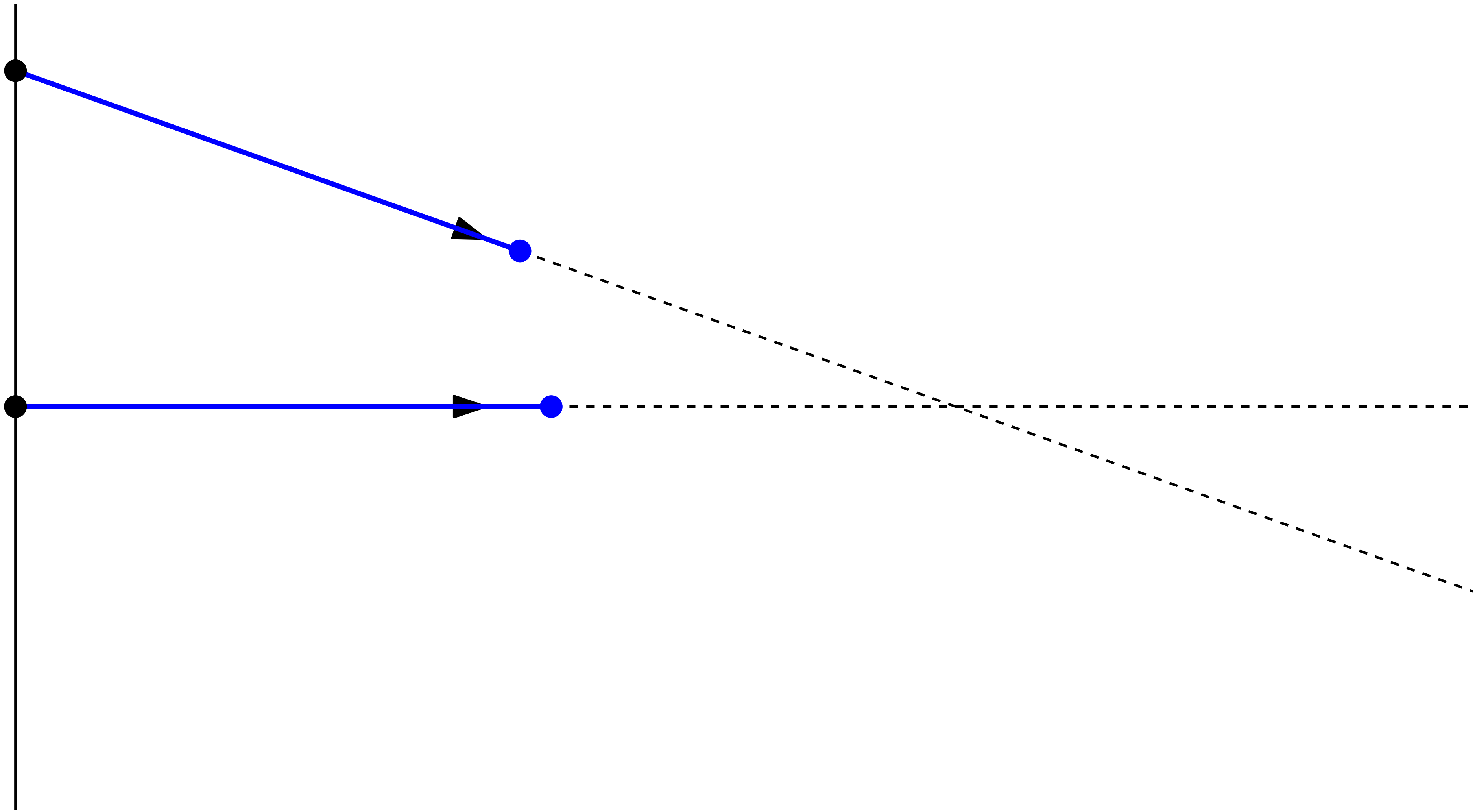
Motorcycles

A group of bikers start at the same time at unit speed. If a biker runs into the tracks of another biker it stops. Tiebreaks (right-hand rule).



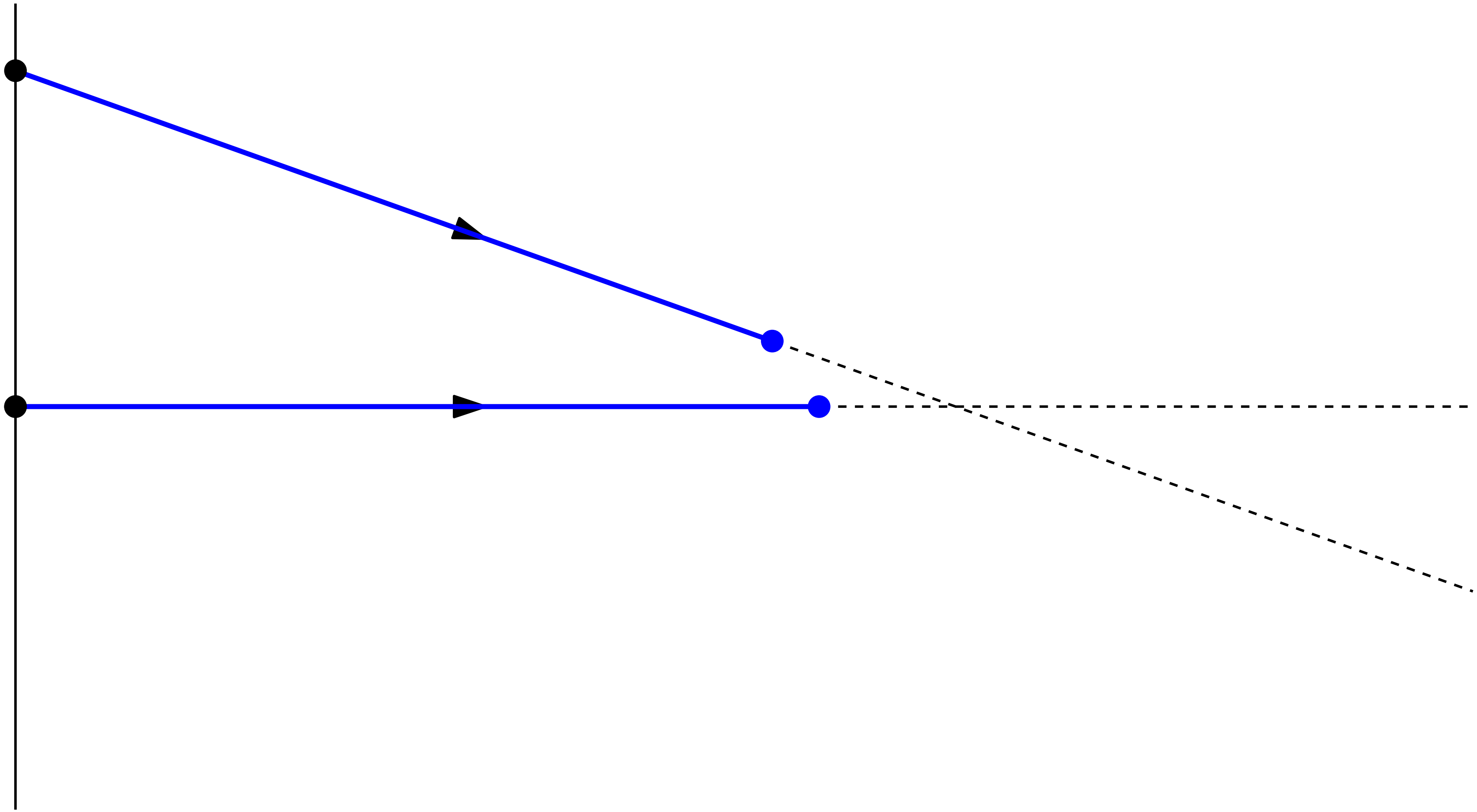
Motorcycles

A group of bikers start at the same time at unit speed. If a biker runs into the tracks of another biker it stops. Tiebreaks (right-hand rule).



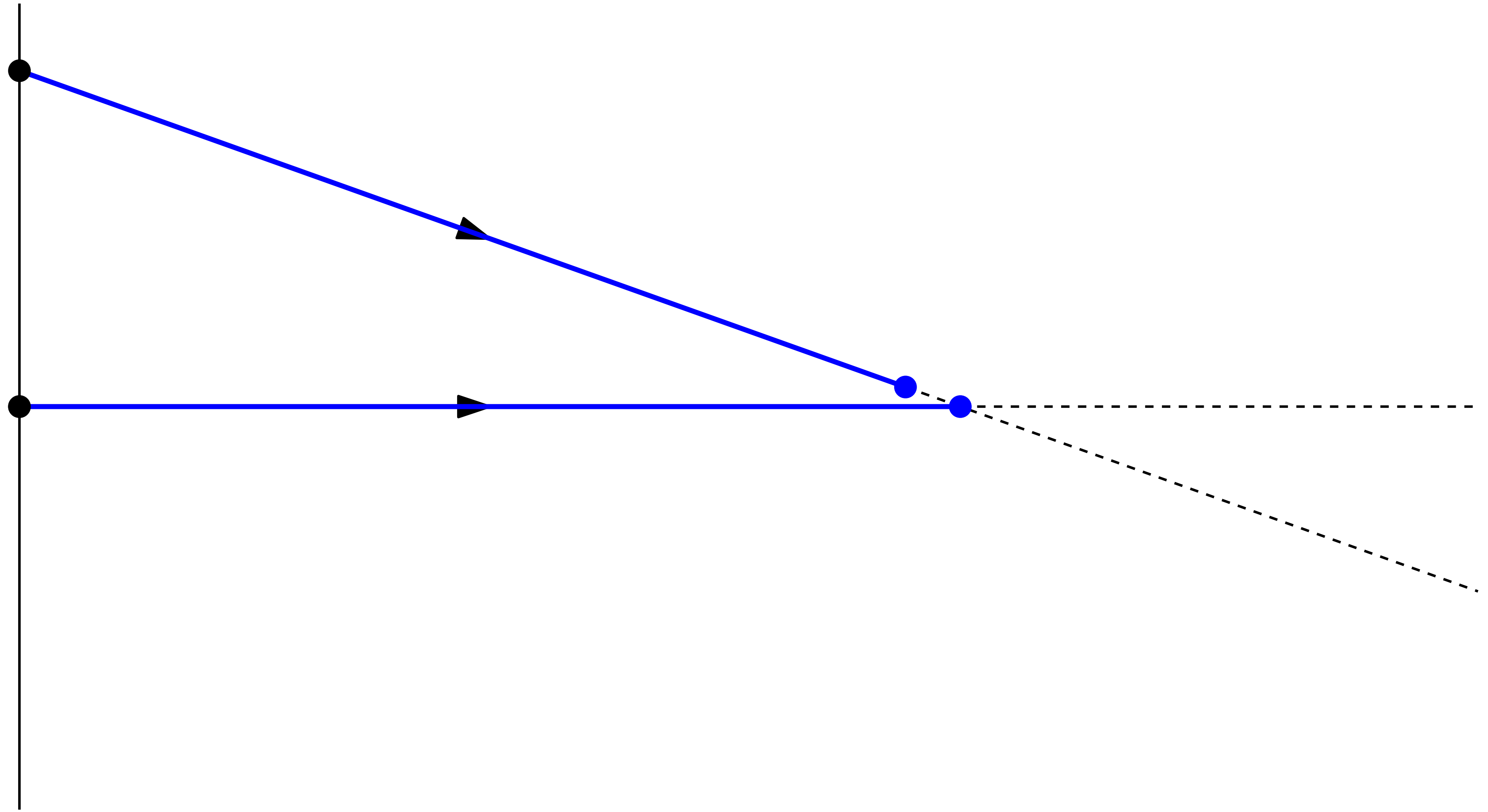
Motorcycles

A group of bikers start at the same time at unit speed. If a biker runs into the tracks of another biker it stops. Tiebreaks (right-hand rule).



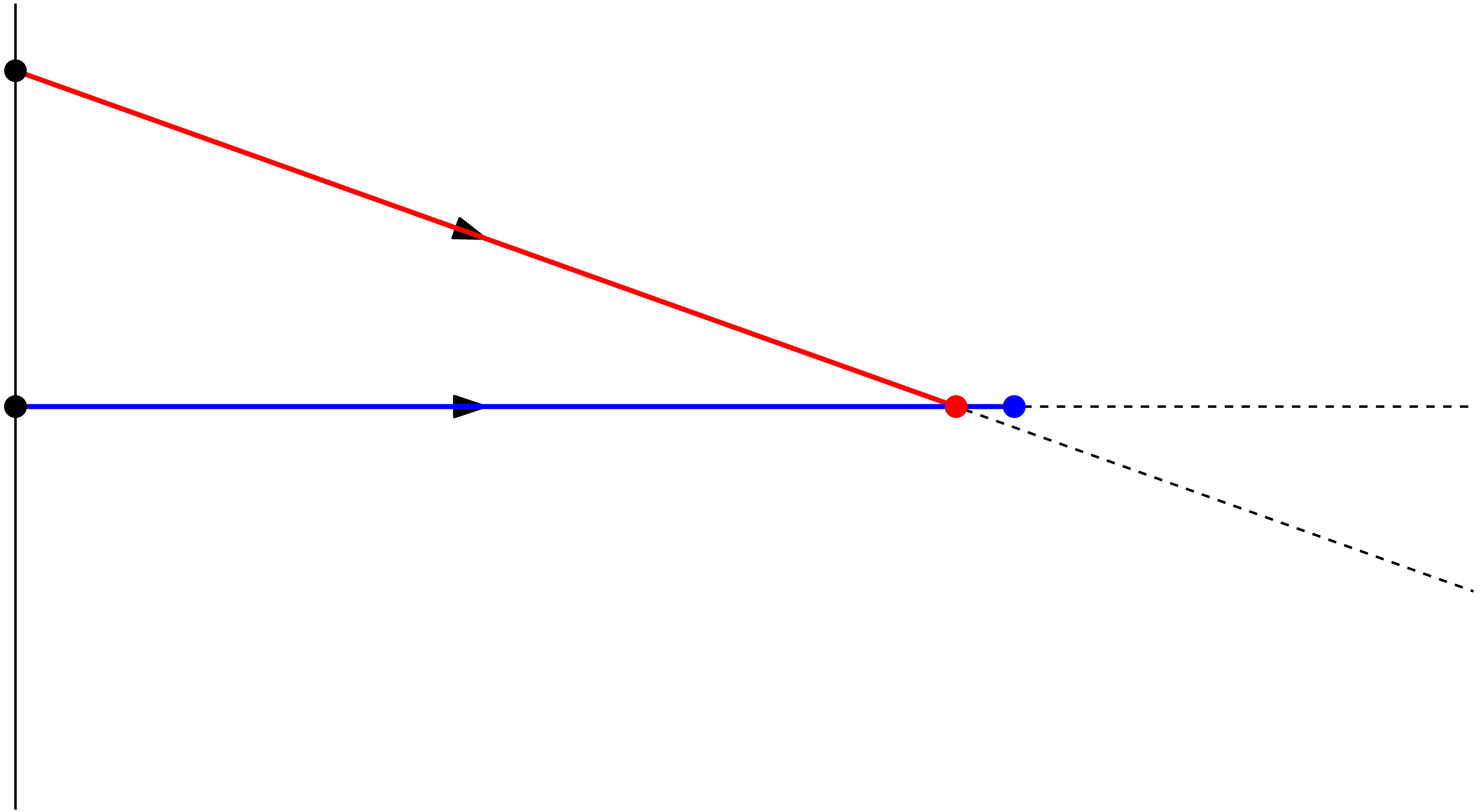
Motorcycles

A group of bikers start at the same time at unit speed. If a biker runs into the tracks of another biker it stops. Tiebreaks (right-hand rule).



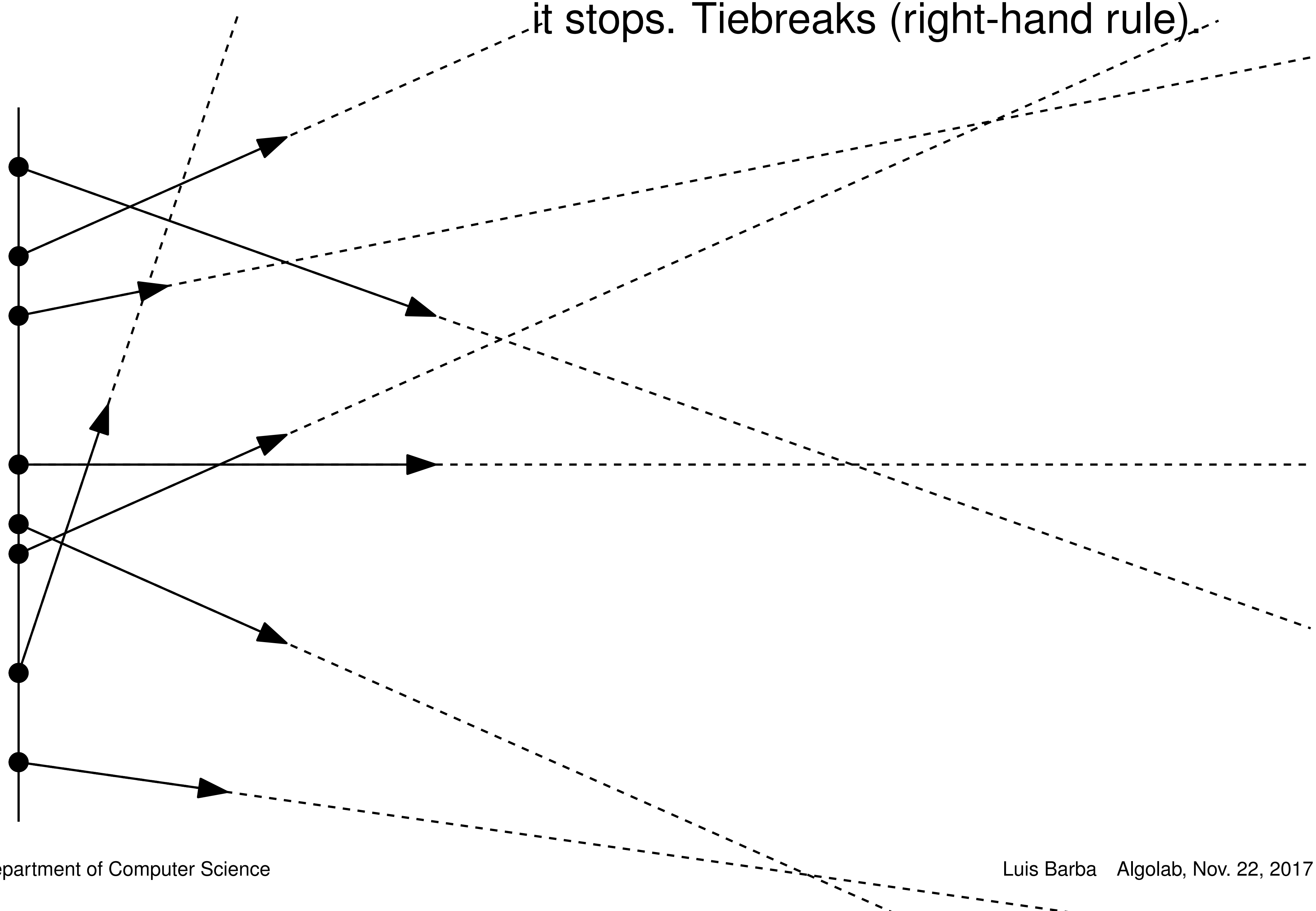
Motorcycles

A group of bikers start at the same time at unit speed. If a biker runs into the tracks of another biker it stops. Tiebreaks (right-hand rule).



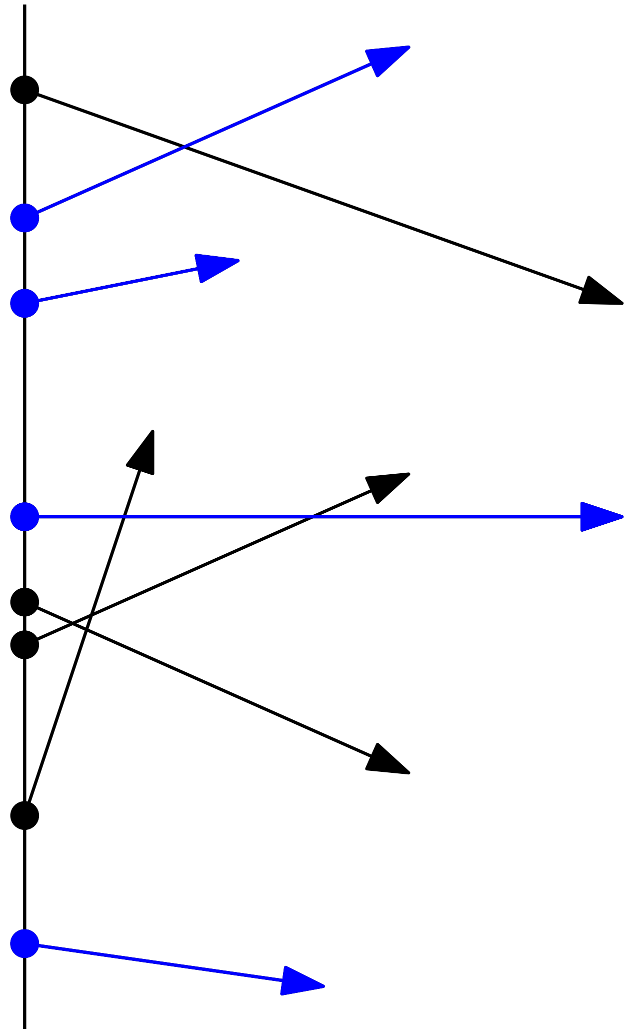
Motorcycles

A group of bikers start at the same time at unit speed. If a biker runs into the tracks of another biker it stops. Tiebreaks (right-hand rule).



Motorcycles

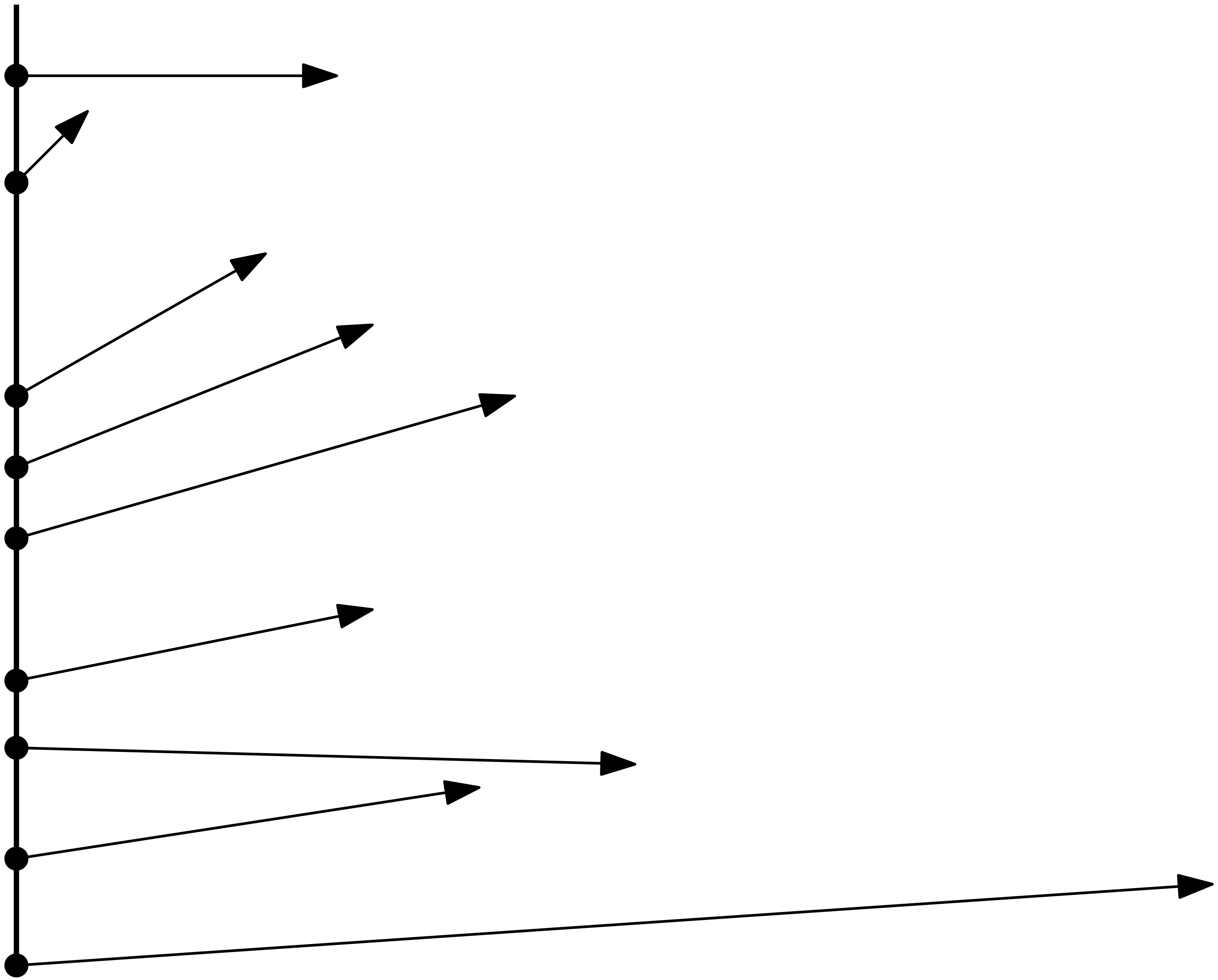
A group of bikers start at the same time at unit speed. If a biker runs into the tracks of another biker it stops. Tiebreaks (right-hand rule).



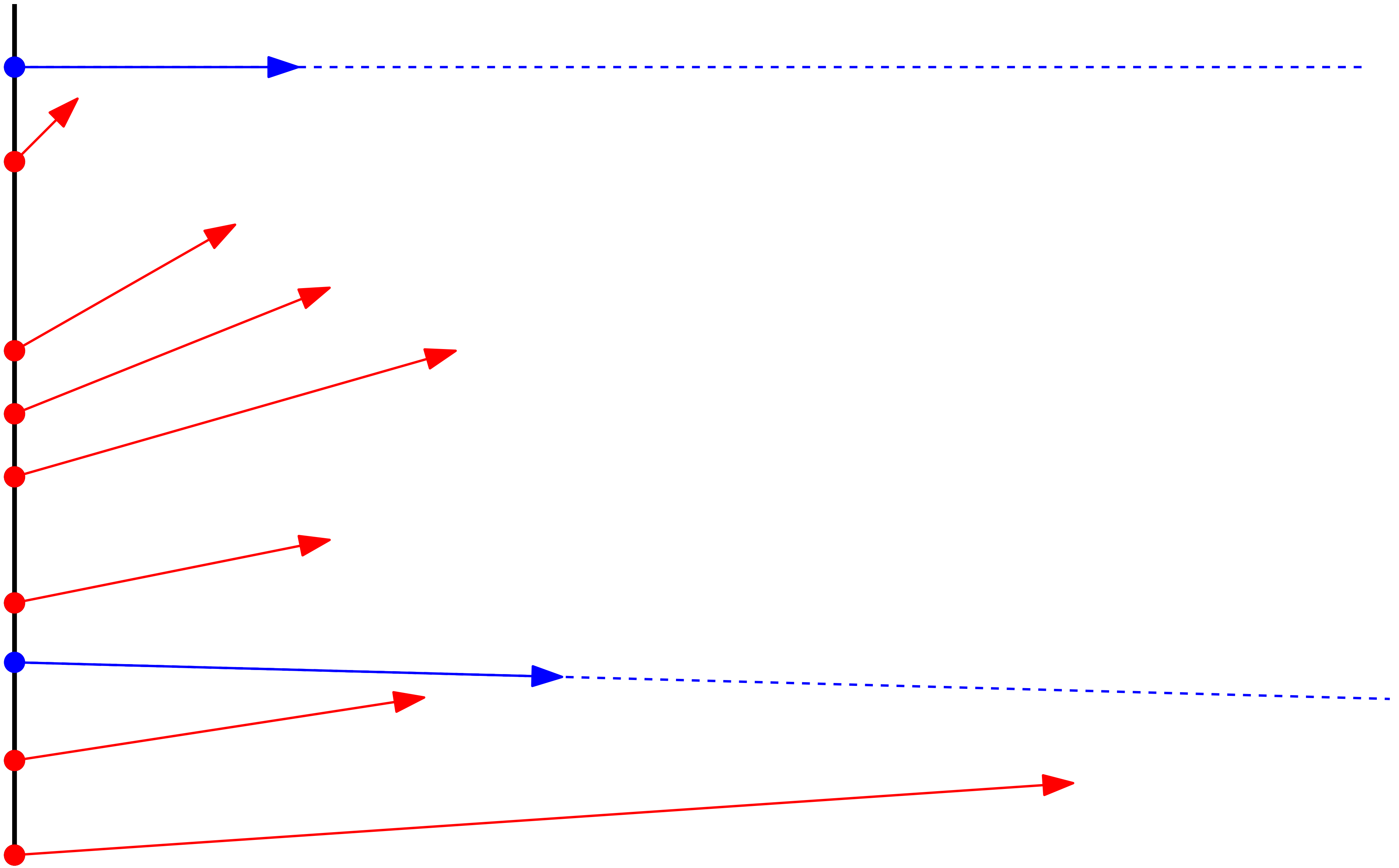
Find which riders get to ride forever.

Admission Control

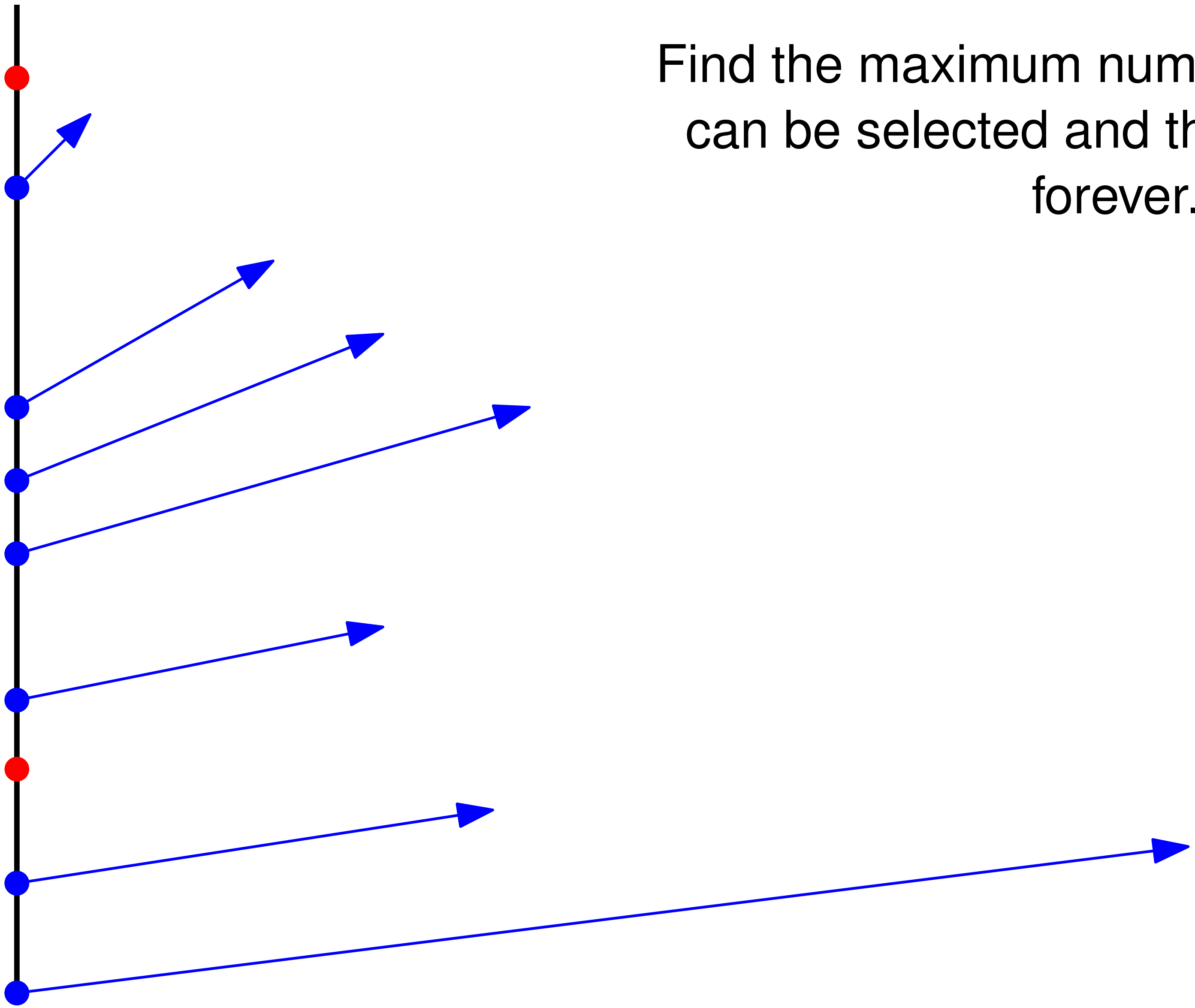
Admission Control



Admission Control

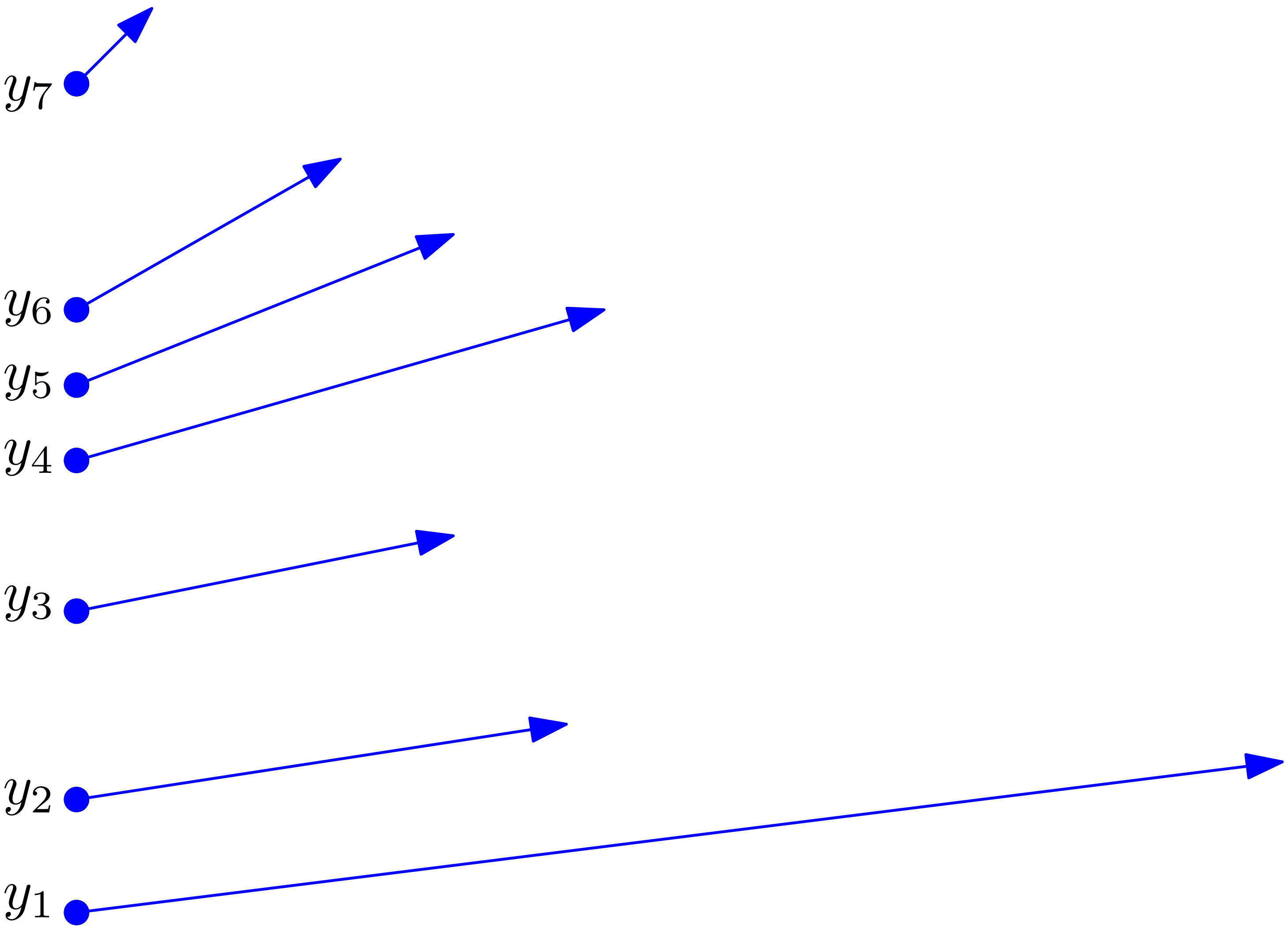


Admission Control

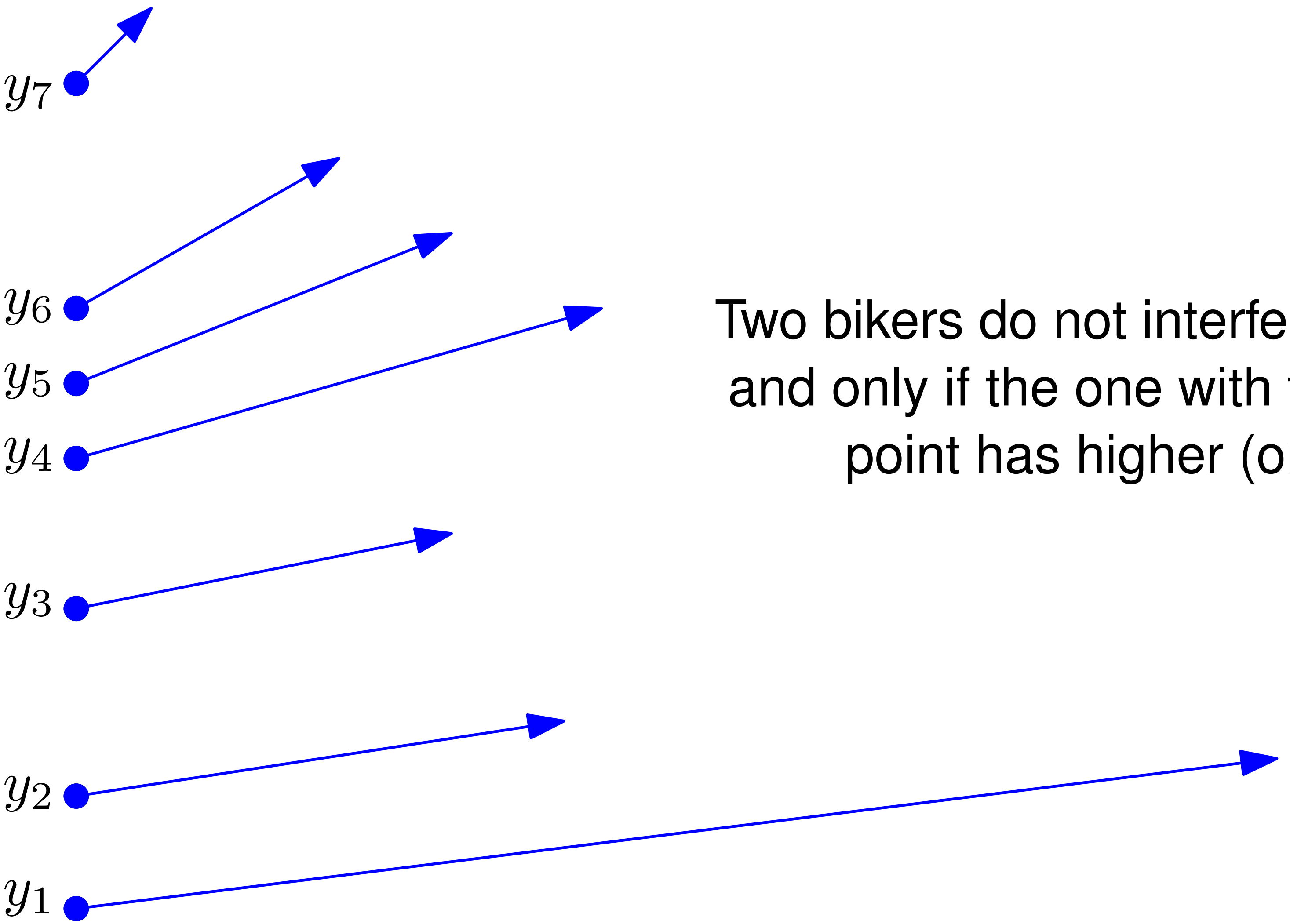


Find the maximum number of riders that can be selected and that will ALL ride forever.

Admission Control

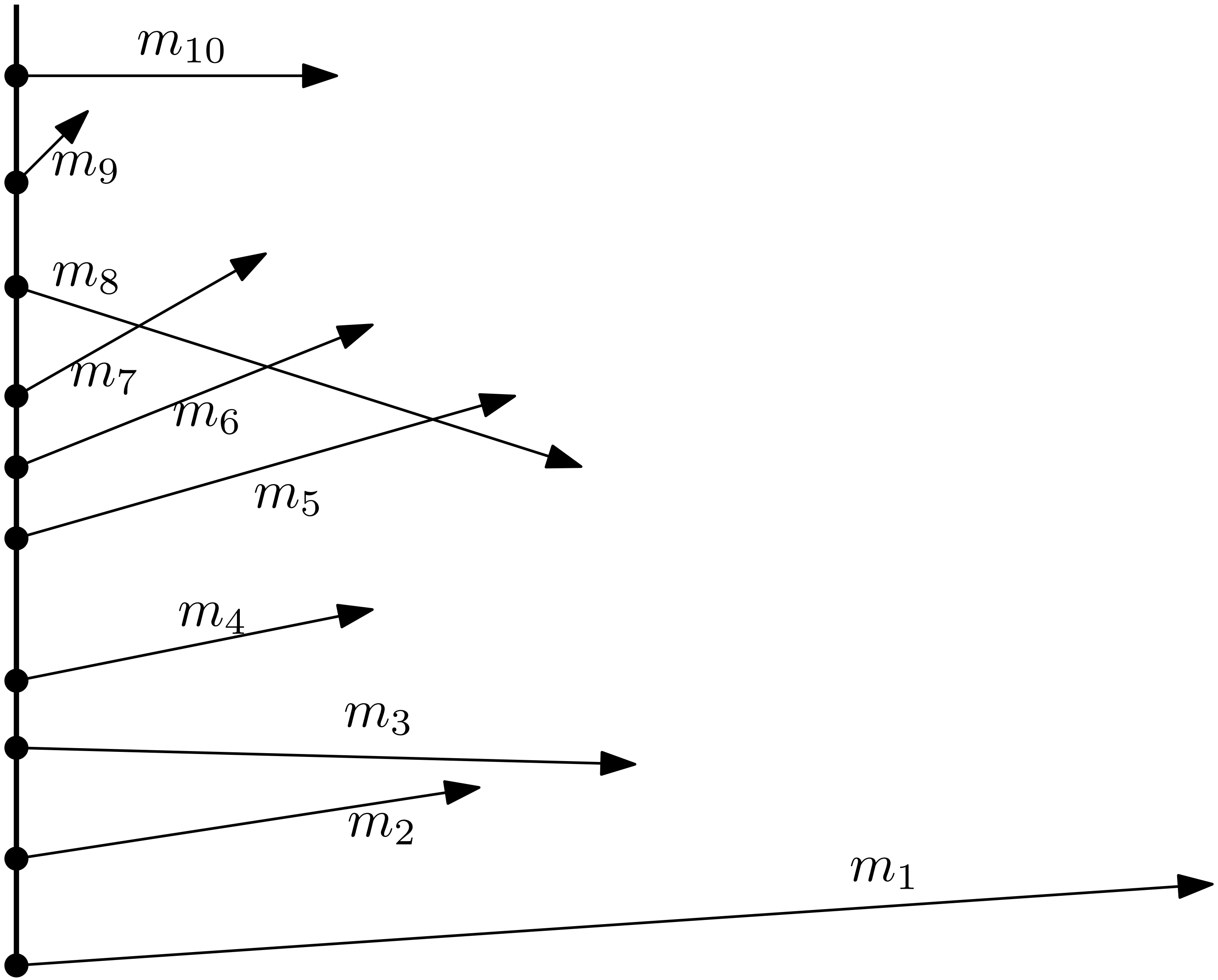


Admission Control



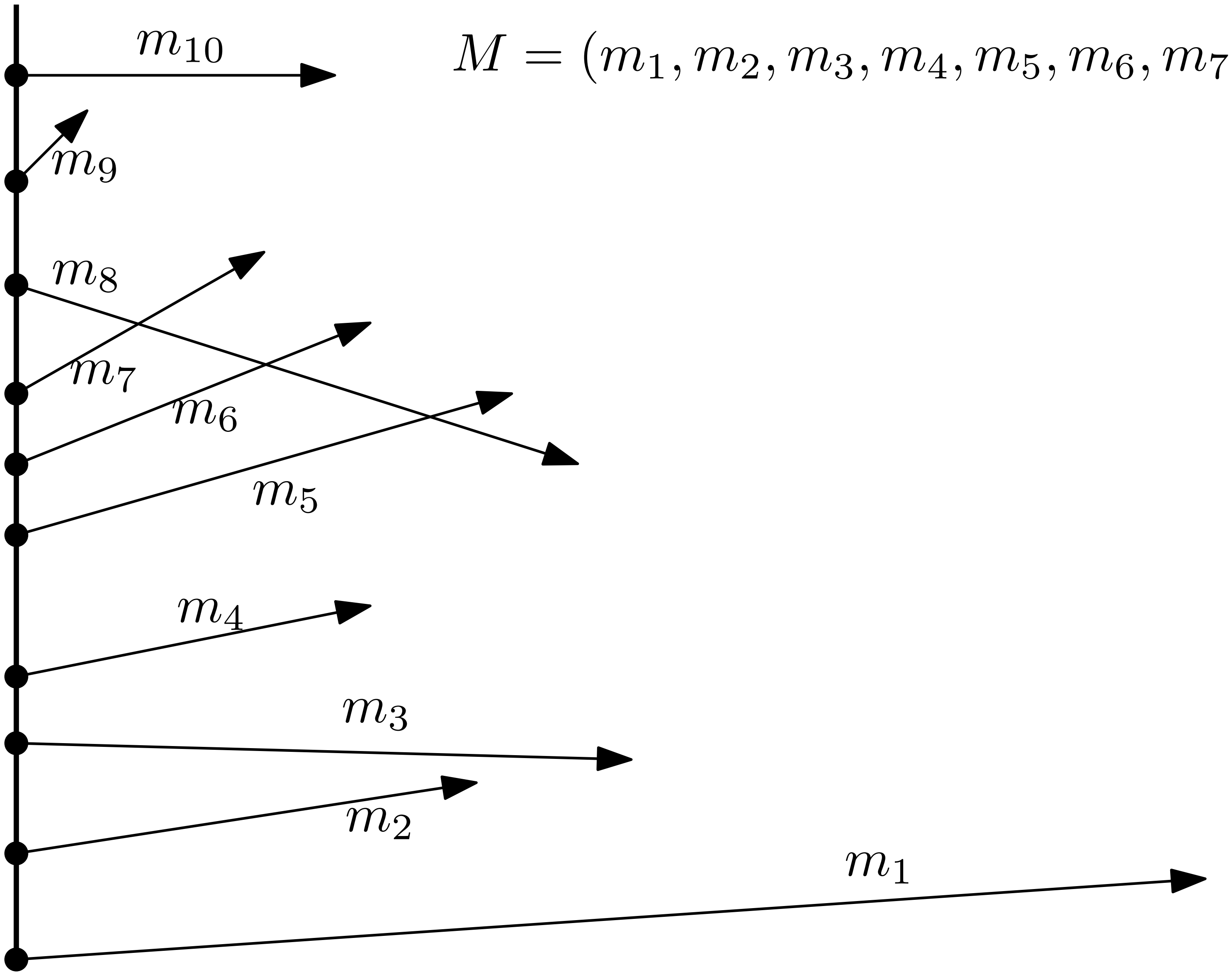
Two bikers do not interfere with each other if and only if the one with the highest starting point has higher (or equal) slope.

Admission Control

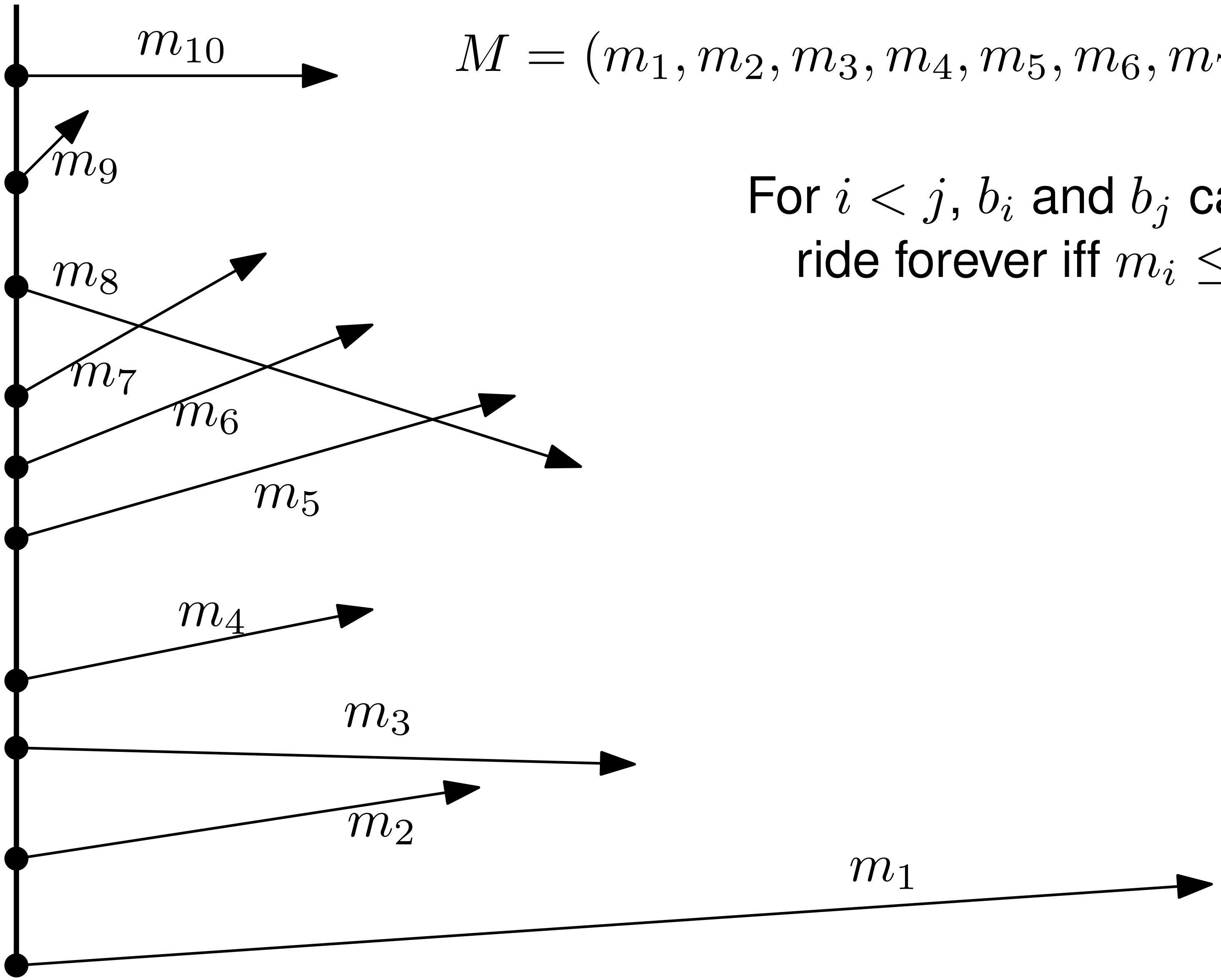


Admission Control

$$M = (m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8, m_9, m_{10})$$



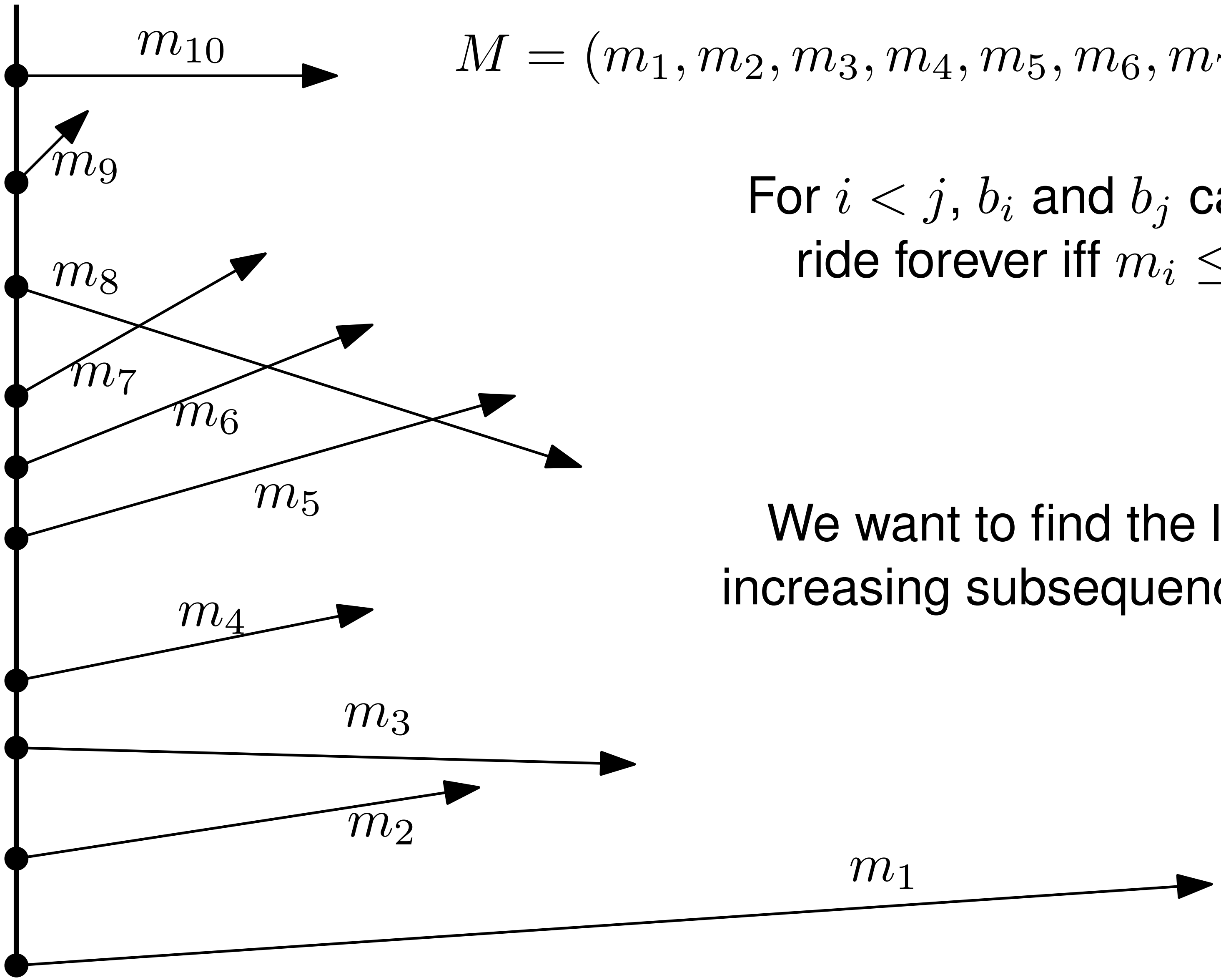
Admission Control



$$M = (m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8, m_9, m_{10})$$

For $i < j$, b_i and b_j can both ride forever iff $m_i \leq m_j$.

Admission Control



$$M = (m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8, m_9, m_{10})$$

For $i < j$, b_i and b_j can both ride forever iff $m_i \leq m_j$.

We want to find the largest increasing subsequence of M.

Largest increasing subsequence

$$M = (m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8, m_9, m_{10})$$

Largest increasing subsequence

$$M = (m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8, m_9, m_{10})$$

- Classical Dynamic programming approach leads to $O(n^2)$ time algorithm.
- For longest increasing subsequence however, there is an $O(n \log n)$ time algorithm to solve the problem.

Largest increasing subsequence

$$M = (0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15)$$

$L1$

$L2$

$L3$

$L4$

$L5$

$L6$

Largest increasing subsequence

$$M = (\boxed{0}, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15)$$

$L1$ 0

$L2$

$L3$

$L4$

$L5$

$L6$

Largest increasing subsequence

$$M = (0, \boxed{8}, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15)$$

$L1$ 0

$L2$ 8

$L3$

$L4$

$L5$

$L6$

Largest increasing subsequence

$$M = (0, 8, \boxed{4}, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15)$$

$L1$ 0

$L2$ 8

$L3$

$L4$

$L5$

$L6$

We search the first subsequence than ends with a number larger than the current number.

Largest increasing subsequence

$$M = (0, 8, \boxed{4}, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15)$$

$L1$ 0

$L2$ $0\ \cancel{8}\ \ 4$

$L3$

$L4$

$L5$

$L6$

We search the first subsequence than ends with a number larger than the current number.

Largest increasing subsequence

$$M = (0, 8, \boxed{4}, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15)$$

$L1$ 0

$L2$ $0\ 4$

$L3$

$L4$

$L5$

$L6$

Largest increasing subsequence

$$M = (0, 8, 4, \boxed{12}, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15)$$

$L1$ 0

$L2$ $0\ 4$

$L3$ $0\ 4\ 12$

$L4$

$L5$

$L6$

We search the first subsequence than ends with a number larger than the current number.

12 Increases the length of all previous subsequences, so we have a new longer increasing subsequence.

Largest increasing subsequence

$$M = (0, 8, 4, 12, \boxed{2,} 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15)$$

$L1$ 0

$L2$ $0\ \cancel{4}\quad 2$

$L3$ $0\ 4\ 12$

$L4$

$L5$

$L6$

We search the first subsequence than ends with a number larger than the current number.

Largest increasing subsequence

$$M = (0, 8, 4, 12, 2, \boxed{10}, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15)$$

$L1$ 0

$L2$ $0\ 2$

$L3$ $0\ 4\ \cancel{12}\quad 10$

$L4$

$L5$

$L6$

We search the first subsequence than ends with a number larger than the current number.

Largest increasing subsequence

$$M = (0, 8, 4, 12, 2, 10, \boxed{6,} 14, 1, 9, 5, 13, 3, 11, 7, 15)$$

$L1$ 0

$L2$ $0\ 2$

$L3$ $0\ 4\ \cancel{10}\quad 6$

$L4$

$L5$

$L6$

We search the first subsequence than ends with a number larger than the current number.

Largest increasing subsequence

$$M = (0, 8, 4, 12, 2, 10, 6, \boxed{14}, 1, 9, 5, 13, 3, 11, 7, 15)$$

$L1$ 0

$L2$ $0\ 2$

$L3$ $0\ 4\ 6$

$L4$ $0\ 4\ 6\ 14$

$L5$

$L6$

We search the first subsequence than ends with a number larger than the current number.

14 increases the length of all previous subsequences, so we create a new one.

Largest increasing subsequence

$$M = (0, 8, 4, 12, 2, 10, 6, 14, \boxed{1,} 9, 5, 13, 3, 11, 7, 15)$$

$L1$ 0

$L2$ $0\ \cancel{2}\quad 1$

$L3$ $0\ 4\ 6$

$L4$ $0\ 4\ 6\ 14$

$L5$

$L6$

We search the first subsequence than ends with a number larger than the current number.

Largest increasing subsequence

$$M = (0, 8, 4, 12, 2, 10, 6, 14, 1, \boxed{9,} 5, 13, 3, 11, 7, 15)$$

$L1$ 0

$L2$ $0\ 1$

$L3$ $0\ 4\ 6$

$L4$ $0\ 4\ 6\ 1\cancel{4}$ 9

$L5$

$L6$

We search the first subsequence than ends with a number larger than the current number.

Largest increasing subsequence

$$M = (0, 8, 4, 12, 2, 10, 6, 14, 1, 9, \boxed{5,} 13, 3, 11, 7, 15)$$

$L1$ 0

$L2$ 0 1

$L3$ 0 4 $\cancel{6}$ 5

$L4$ 0 4 6 9

$L5$

$L6$

We search the first subsequence than ends with a number larger than the current number.

Largest increasing subsequence

$$M = (0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, \boxed{13}, 3, 11, 7, 15)$$

$L1$ 0

$L2$ $0\ 1$

$L3$ $0\ 4\ 5$

$L4$ $0\ 4\ 6\ 9$

$L5$ $0\ 4\ 6\ 9\ 13$

$L6$

We search the first subsequence than ends with a number larger than the current number.

Largest increasing subsequence

$$M = (0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13\boxed{, 3,} 11, 7, 15)$$

$L1$ 0

$L2$ $0\ 1$

$L3$ $0\ 4\ \cancel{5}\quad 3$

$L4$ $0\ 4\ 6\ 9$

$L5$ $0\ 4\ 6\ 9\ 13$

$L6$

We search the first subsequence than ends with a number larger than the current number.

Largest increasing subsequence

$$M = (0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, \boxed{11}, 7, 15)$$

$L1$ 0

$L2$ $0\ 1$

$L3$ $0\ 4\ 3$

$L4$ $0\ 4\ 6\ 9$

$L5$ $0\ 4\ 6\ 9\ \cancel{13}\quad 11$

$L6$

We search the first subsequence than ends with a number larger than the current number.

Largest increasing subsequence

$$M = (0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, \boxed{7,} 15)$$

$L1$ 0

$L2$ $0\ 1$

$L3$ $0\ 4\ 3$

$L4$ $0\ 4\ 6\ \cancel{9}\quad 7$

$L5$ $0\ 4\ 6\ 9\ 13$

$L6$

We search the first subsequence than ends with a number larger than the current number.

Largest increasing subsequence

$$M = (0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, \boxed{15})$$

$L1$ 0

$L2$ 0 1

$L3$ 0 4 3

$L4$ 0 4 6 7

$L5$ 0 4 6 9 13

$L6$ 0 4 6 9 13 15

We search the first subsequence than ends with a number larger than the current number.

Largest increasing subsequence

$$M = (0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, \boxed{15})$$

$L1$ 0

$L2$ $0\ 1$

$L3$ $0\ 4\ 3$

$L4$ $0\ 4\ 6\ 7$

$L5$ $0\ 4\ 6\ 9\ 13$

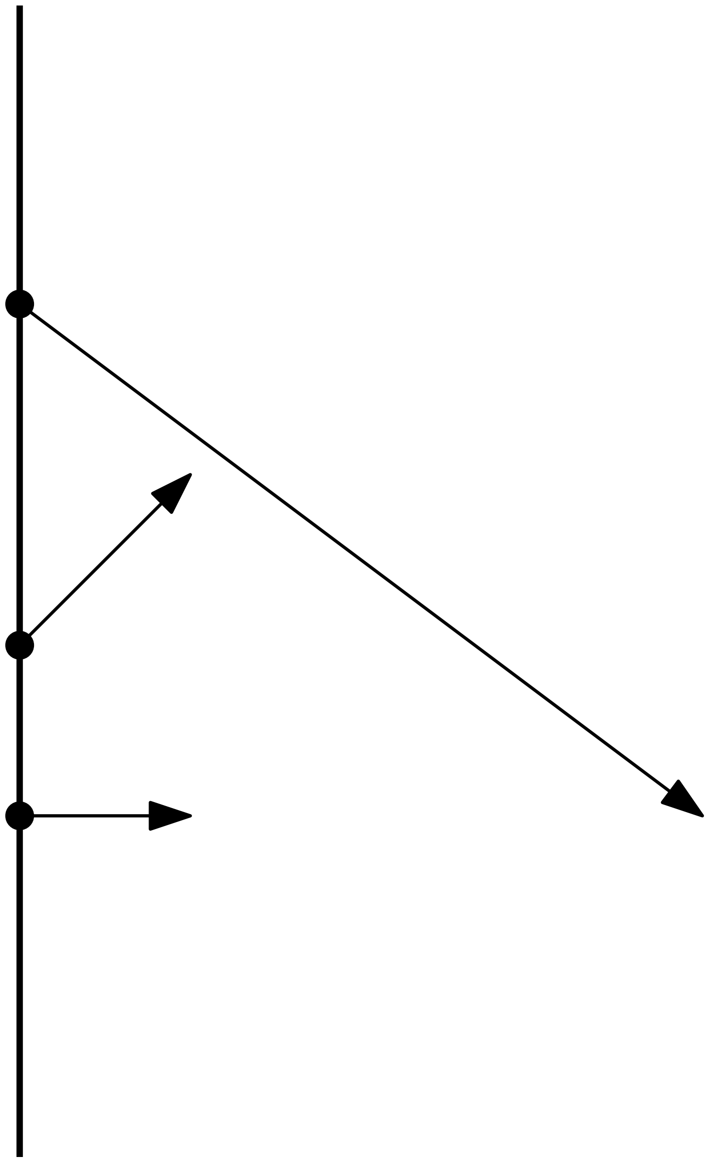
$L6$ $0\ 4\ 6\ 9\ 13\ 15$

We search the first subsequence than ends with a number larger than the current number.

The algorithm performs a binary search for each element of the original vector. Thus, the running time is $O(n \log n)$.

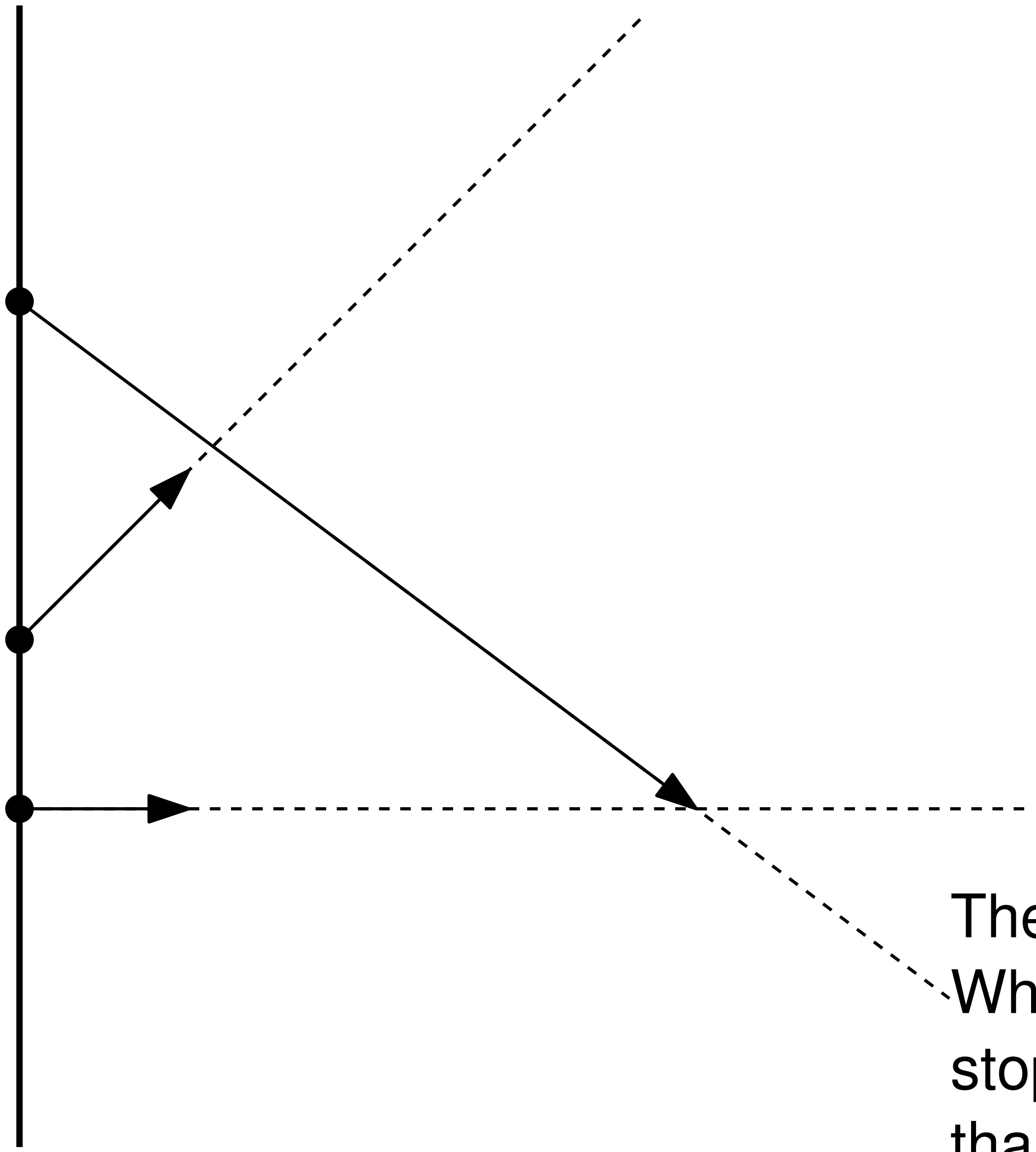
Starting Schedules

We are allowed to modify the starting time s_i of each bike b_i.



There is a frustration tolerance f. Whenever b_i meets the tracks of b_j, it stops unless b_j was there no more than f time units ago.

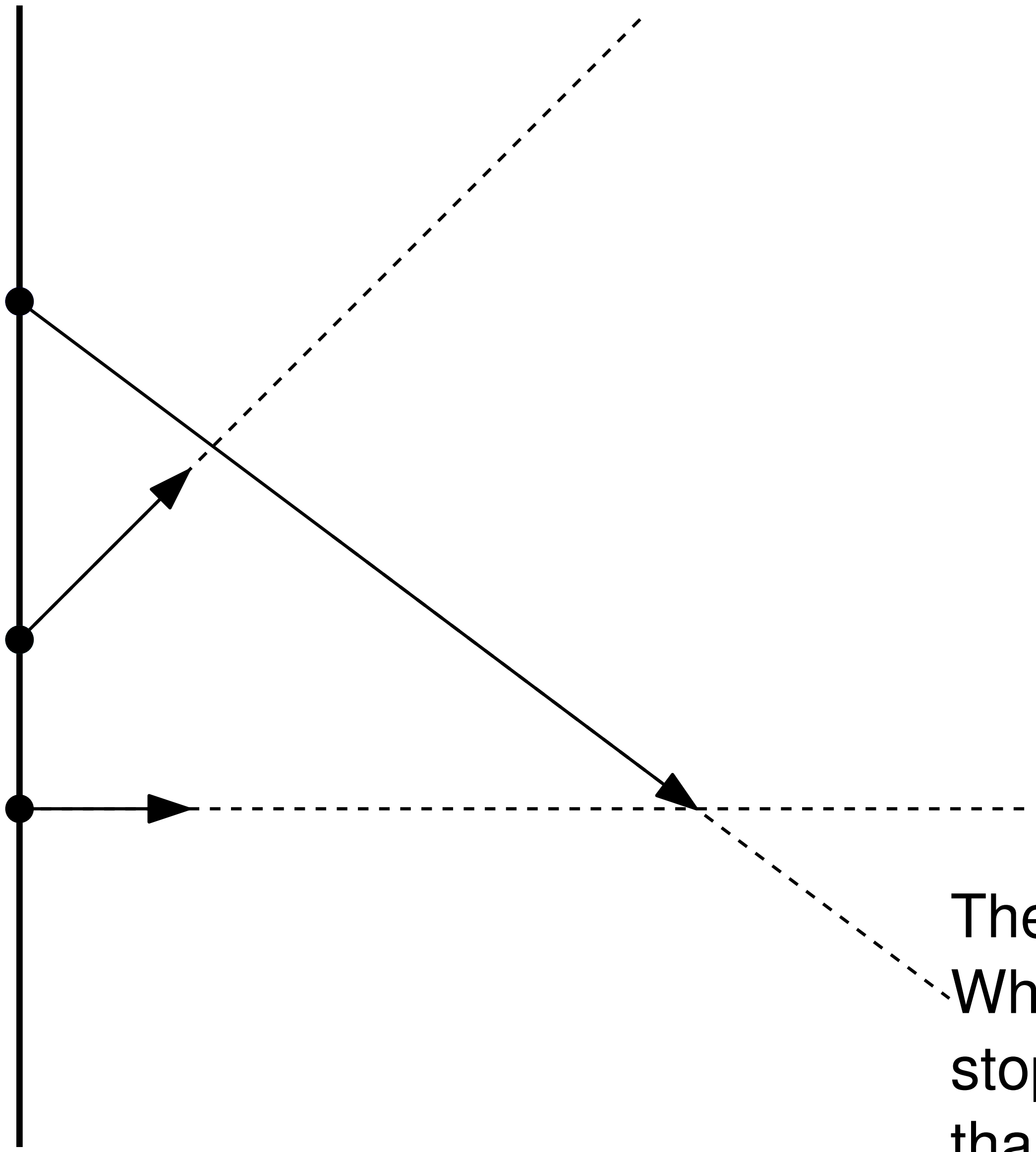
Starting Schedules



We are allowed to modify the starting time s_i of each bike b_i.

There is a frustration tolerance f. Whenever b_i meets the tracks of b_j, it stops unless b_j was there no more than f time units ago.

Starting Schedules

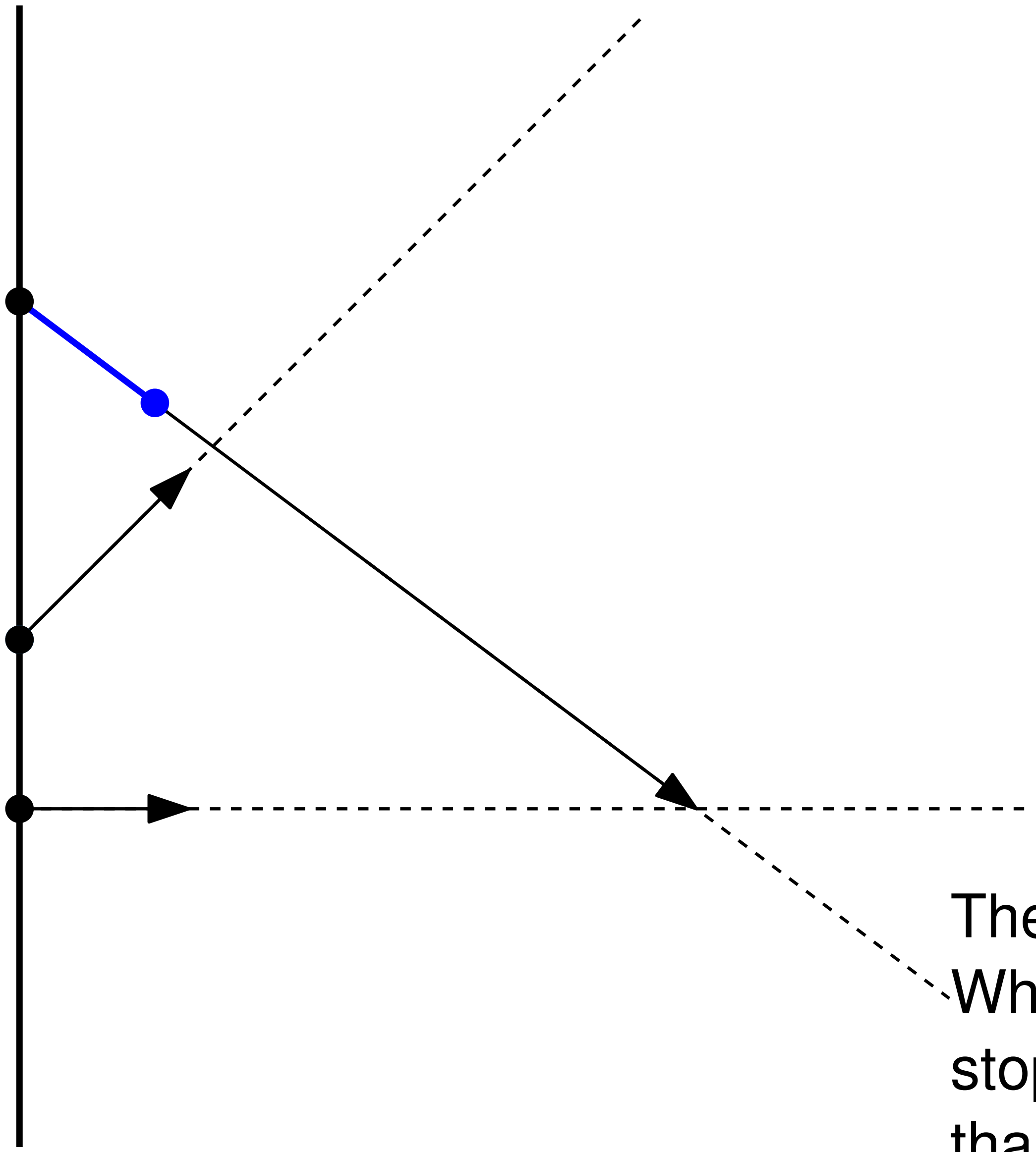


We are allowed to modify the starting time s_i of each bike b_i.

There is a frustration tolerance f. Whenever b_i meets the tracks of b_j, it stops unless b_j was there no more than f time units ago.

Starting Schedules

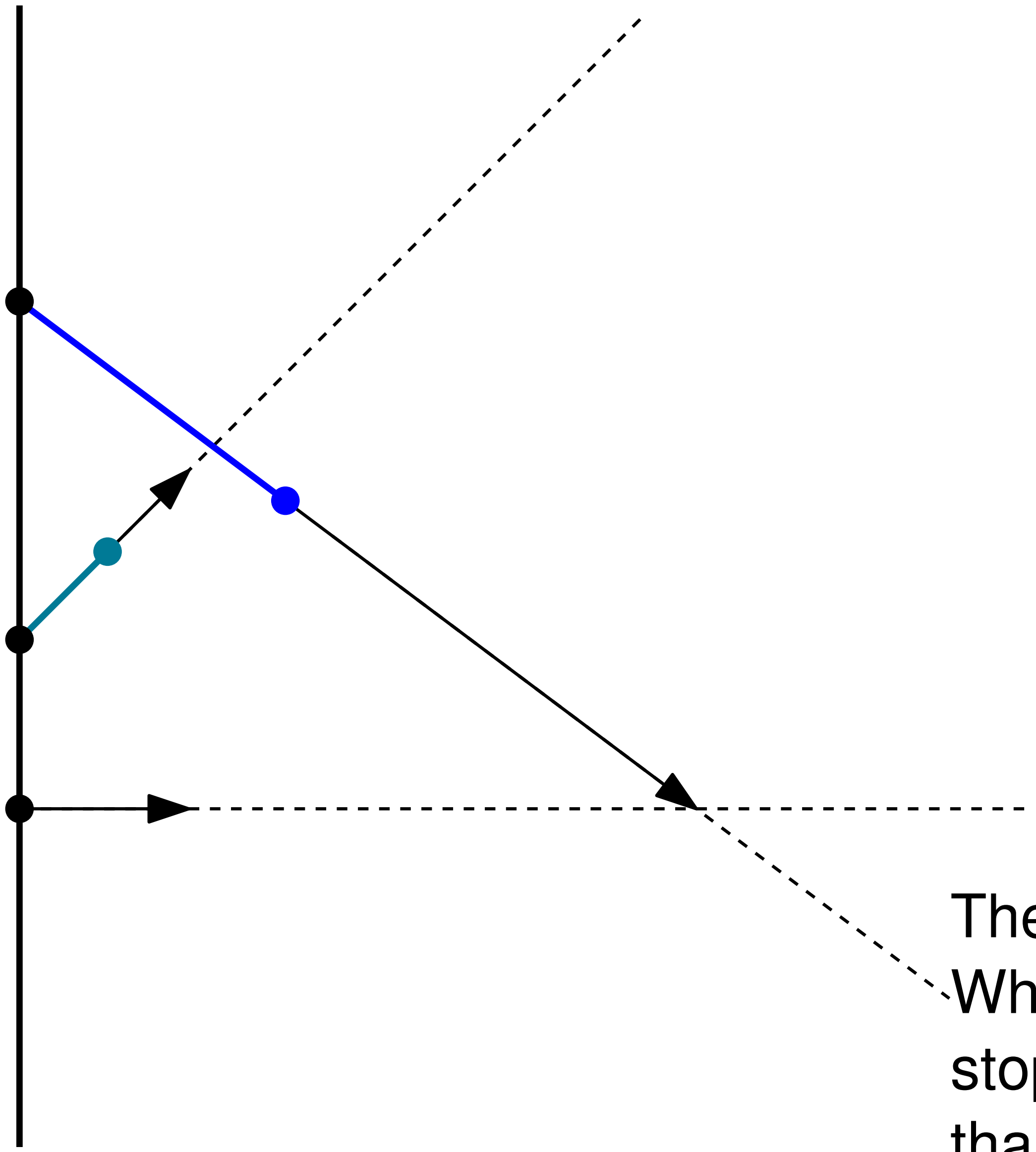
We are allowed to modify the starting time s_i of each bike b_i.



There is a frustration tolerance f. Whenever b_i meets the tracks of b_j, it stops unless b_j was there no more than f time units ago.

Starting Schedules

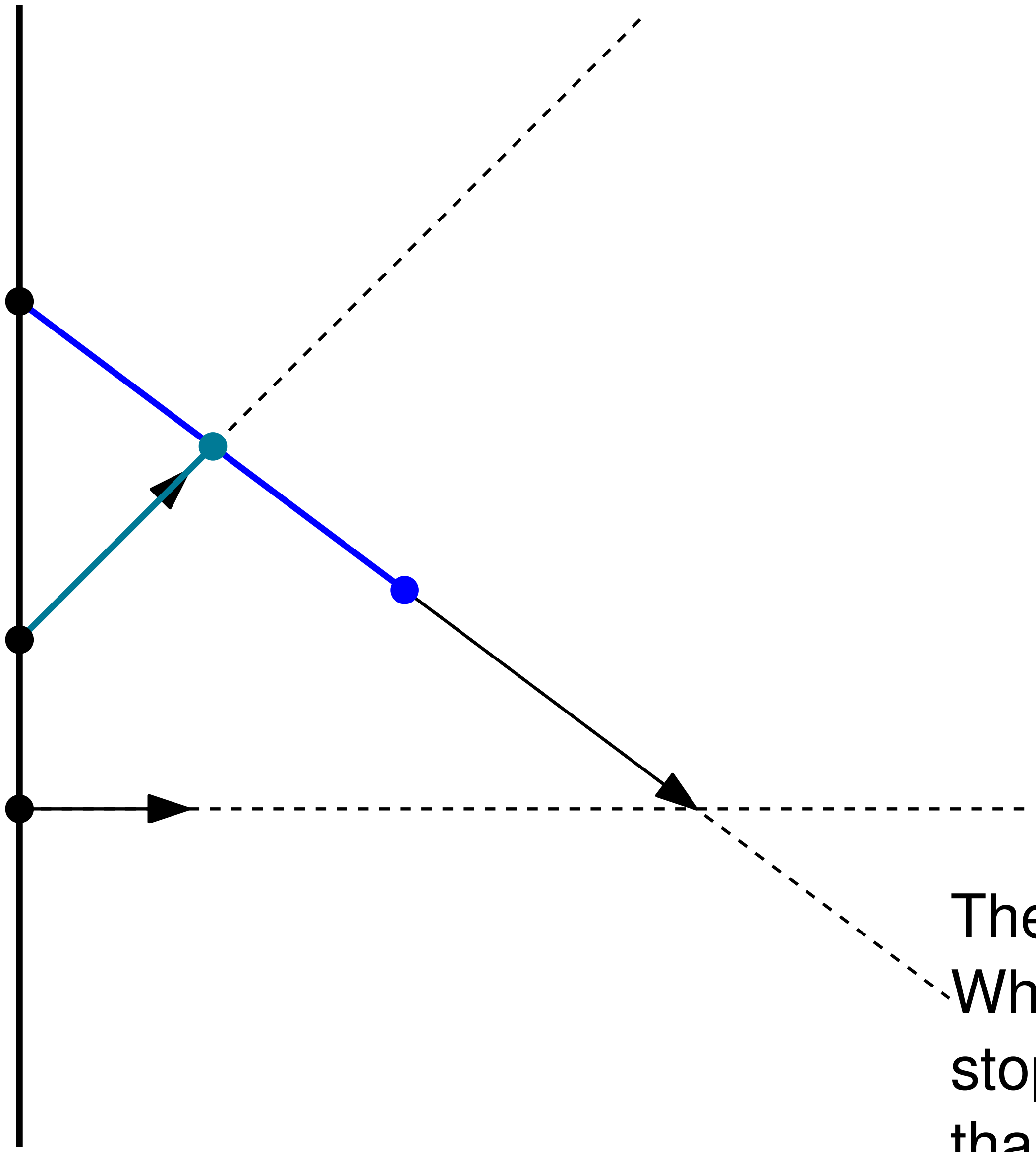
We are allowed to modify the starting time s_i of each bike b_i.



There is a frustration tolerance f. Whenever b_i meets the tracks of b_j, it stops unless b_j was there no more than f time units ago.

Starting Schedules

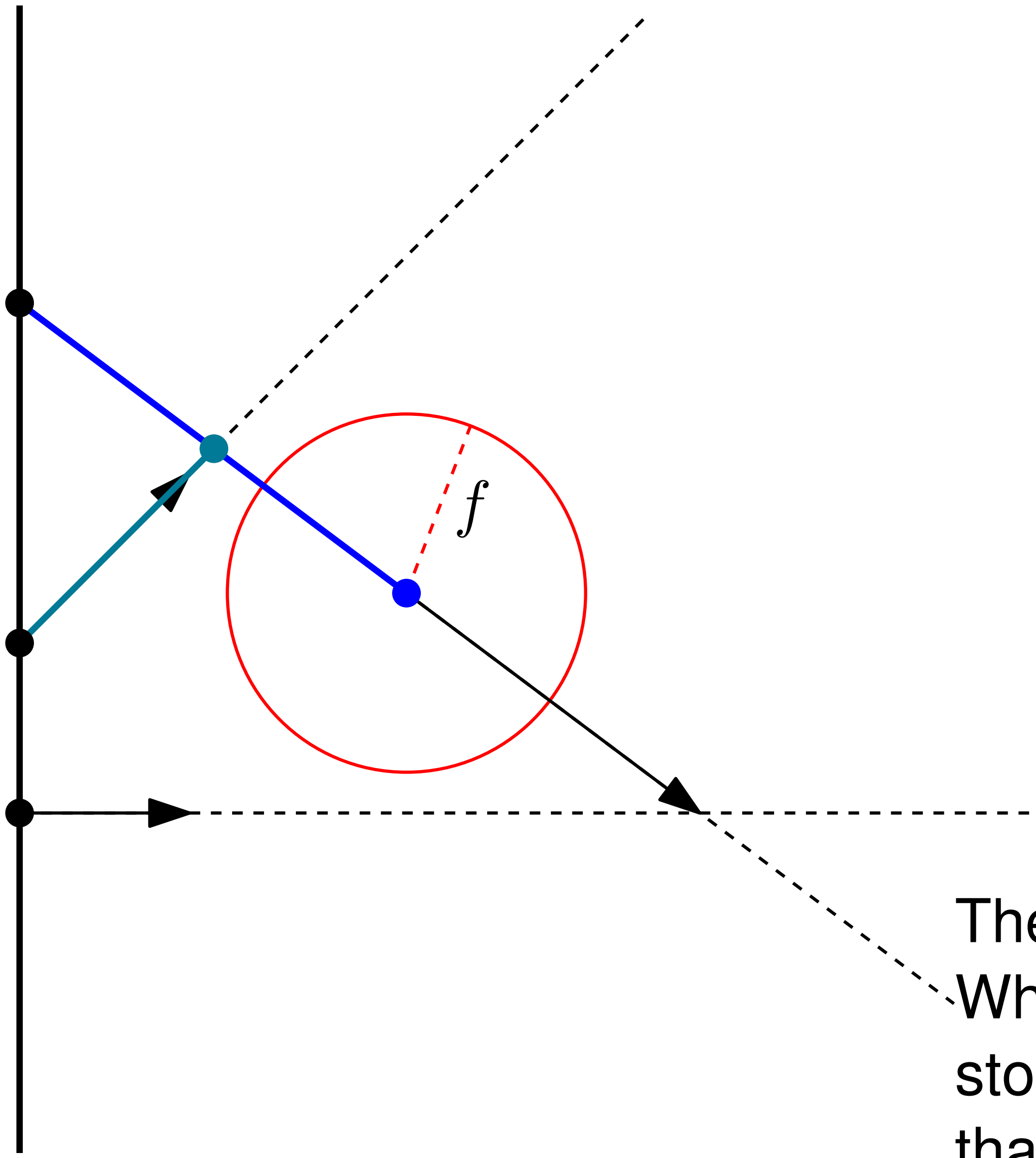
We are allowed to modify the starting time s_i of each bike b_i.



There is a frustration tolerance f. Whenever b_i meets the tracks of b_j, it stops unless b_j was there no more than f time units ago.

Starting Schedules

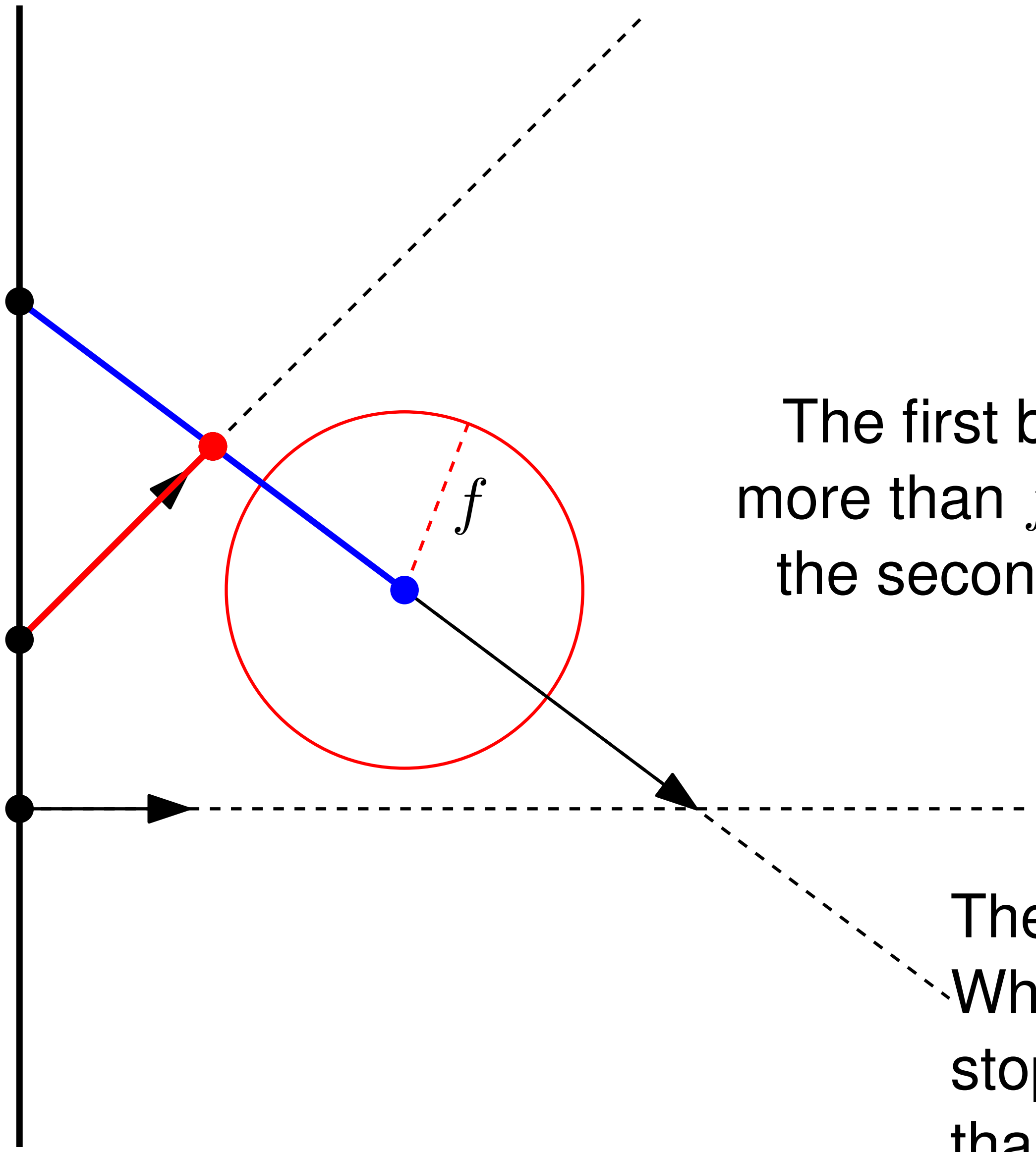
We are allowed to modify the starting time s_i of each bike b_i.



There is a frustration tolerance f. Whenever b_i meets the tracks of b_j, it stops unless b_j was there no more than f time units ago.

Starting Schedules

We are allowed to modify the starting time s_i of each bike b_i.

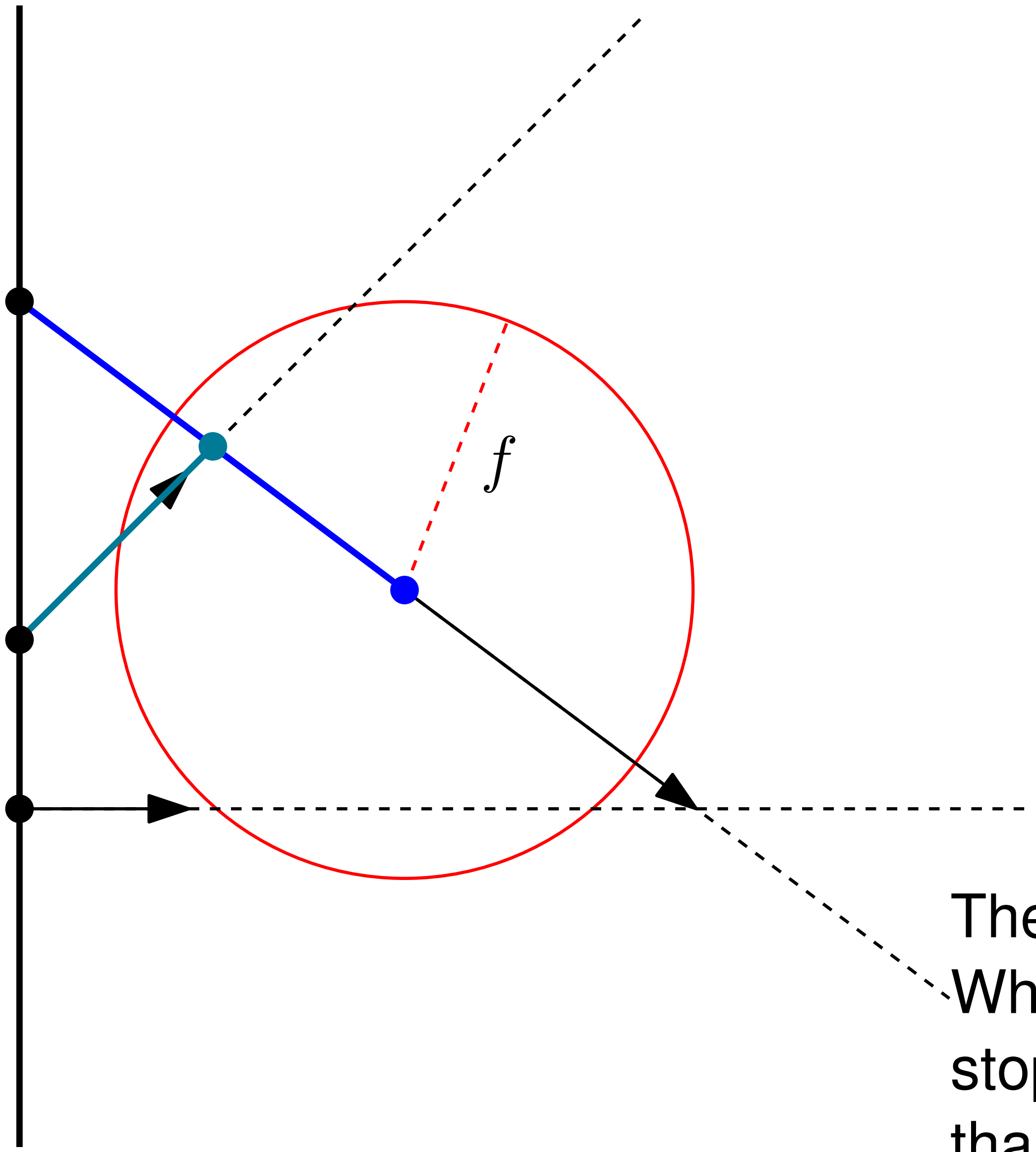


The first biker passed more than f time ago, so the second must stop.

There is a frustration tolerance f. Whenever b_i meets the tracks of b_j, it stops unless b_j was there no more than f time units ago.

Starting Schedules

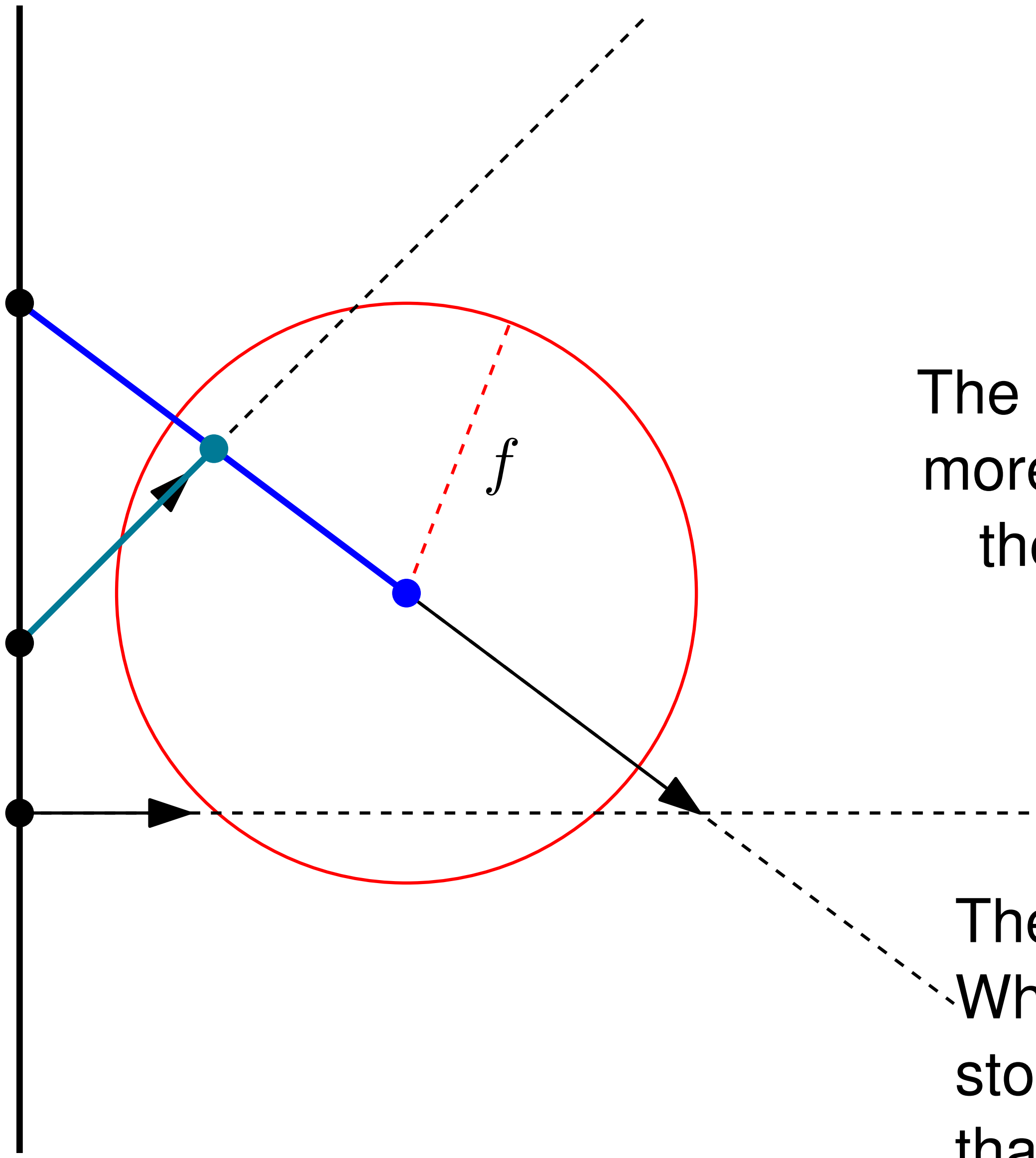
We are allowed to modify the starting time s_i of each bike b_i.



There is a frustration tolerance f. Whenever b_i meets the tracks of b_j, it stops unless b_j was there no more than f time units ago.

Starting Schedules

We are allowed to modify the starting time s_i of each bike b_i.

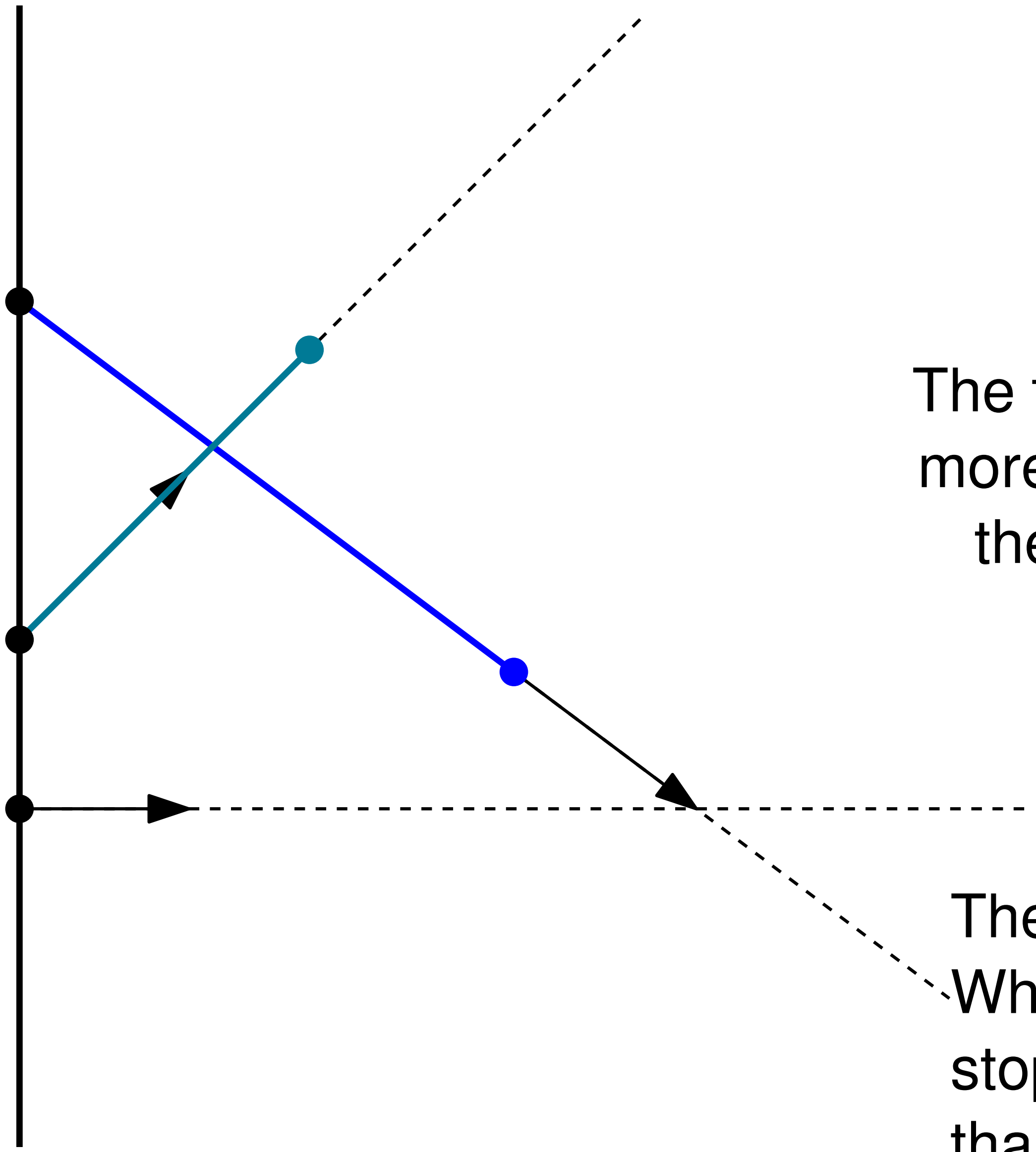


The first biker passed NO more than f time ago, so the second can keep riding.

There is a frustration tolerance f. Whenever b_i meets the tracks of b_j, it stops unless b_j was there no more than f time units ago.

Starting Schedules

We are allowed to modify the starting time s_i of each bike b_i.



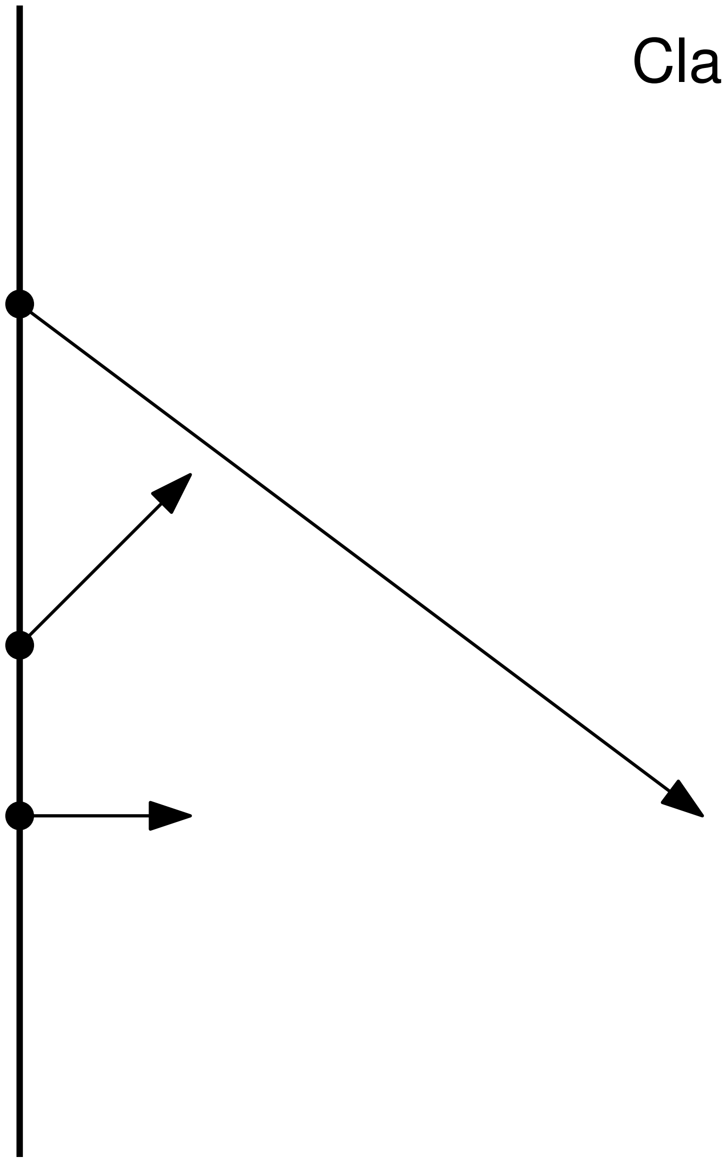
The first biker passed NO more than f time ago, so the second can keep riding.

There is a frustration tolerance f. Whenever b_i meets the tracks of b_j, it stops unless b_j was there no more than f time units ago.

Starting Schedules

We are allowed to modify the starting time s_i of each bike b_i.

Claim: We can solve this instance with $f = 0$.

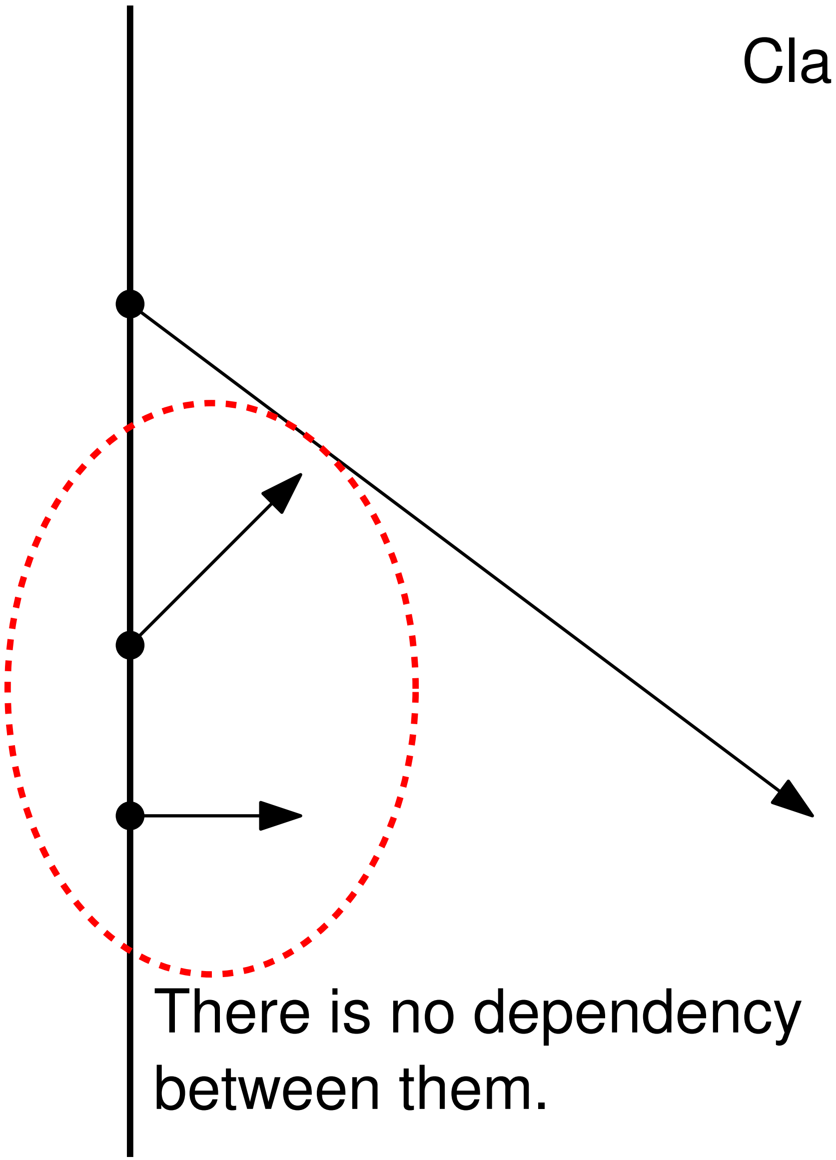


There is a frustration tolerance f. Whenever b_i meets the tracks of b_j, it stops unless b_j was there no more than f time units ago.

Starting Schedules

We are allowed to modify the starting time s_i of each bike b_i.

Claim: We can solve this instance with $f = 0$.



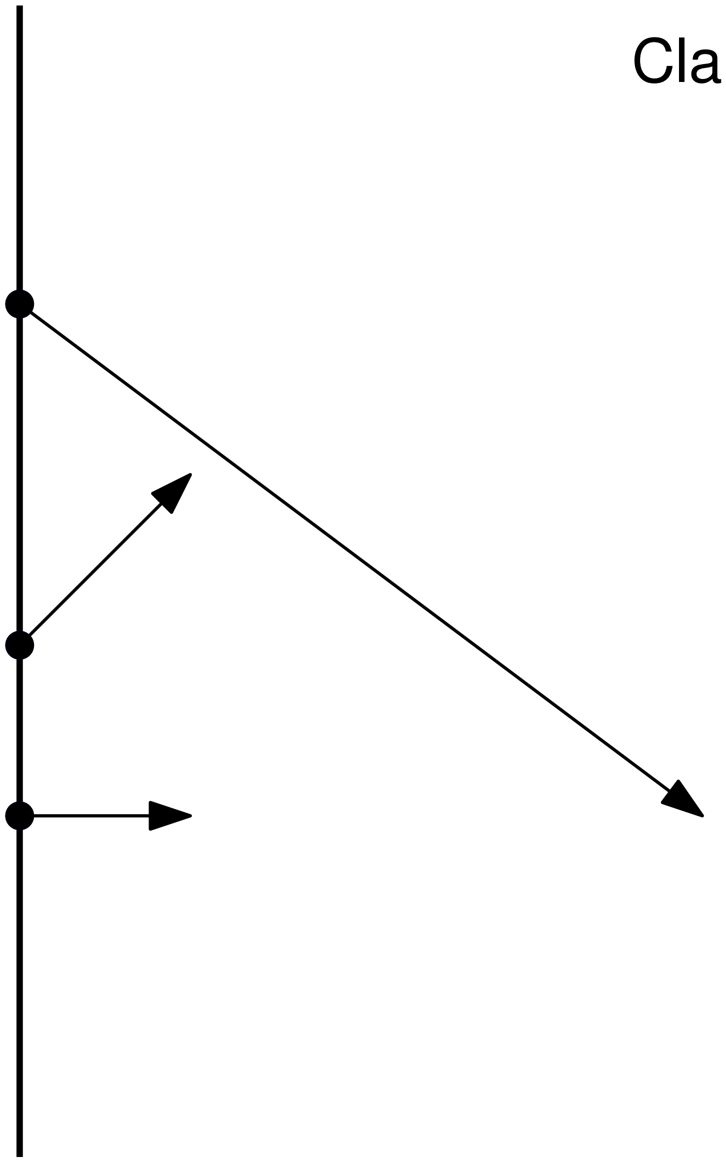
There is no dependency between them.

There is a frustration tolerance f. Whenever b_i meets the tracks of b_j, it stops unless b_j was there no more than f time units ago.

Starting Schedules

We are allowed to modify the starting time s_i of each bike b_i.

Claim: We can solve this instance with $f = 0$.

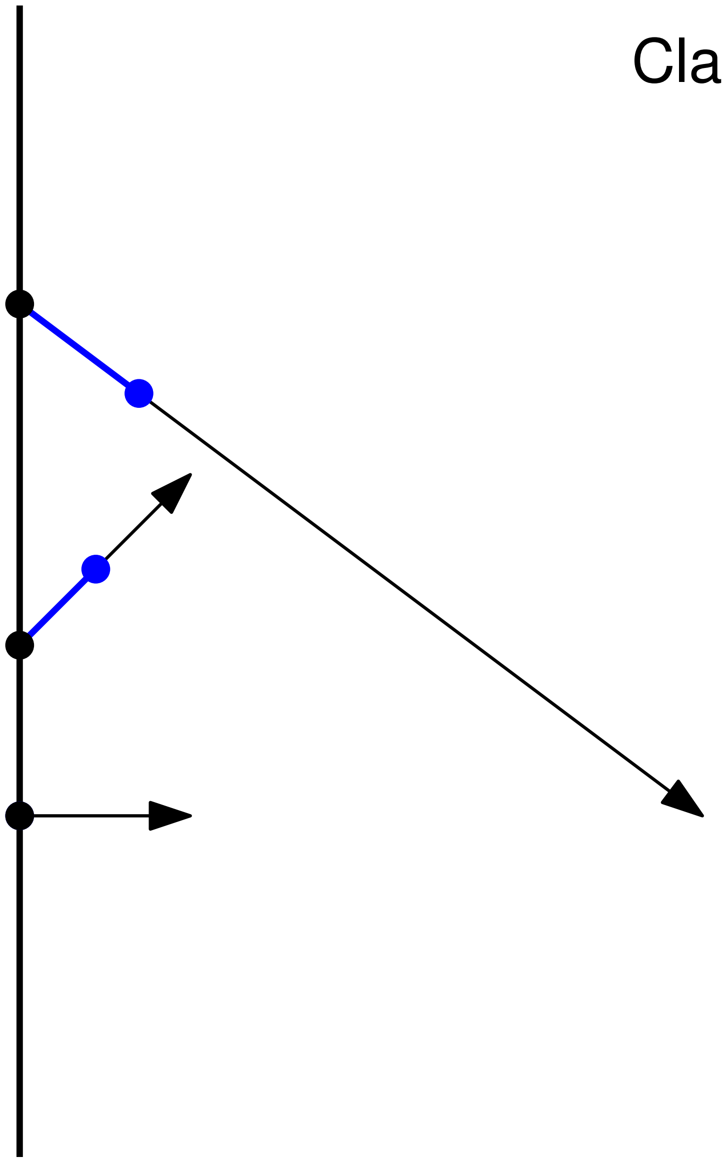


There is a frustration tolerance f. Whenever b_i meets the tracks of b_j, it stops unless b_j was there no more than f time units ago.

Starting Schedules

We are allowed to modify the starting time s_i of each bike b_i.

Claim: We can solve this instance with $f = 0$.

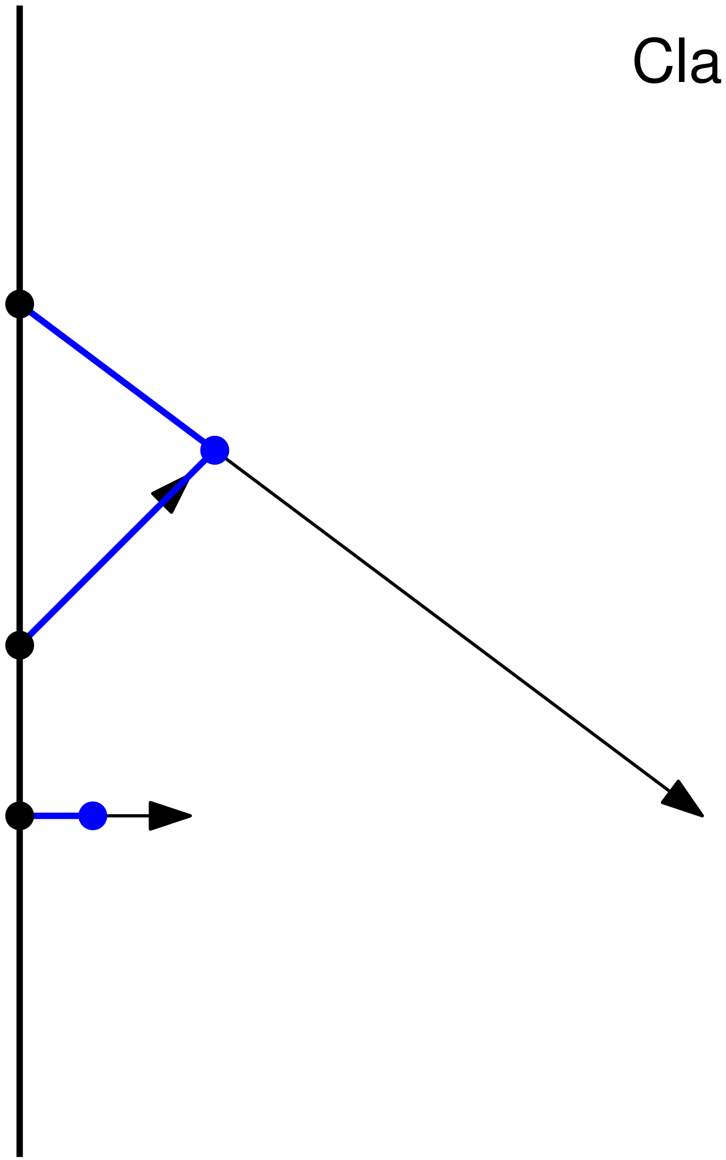


There is a frustration tolerance f. Whenever b_i meets the tracks of b_j, it stops unless b_j was there no more than f time units ago.

Starting Schedules

We are allowed to modify the starting time s_i of each bike b_i.

Claim: We can solve this instance with $f = 0$.

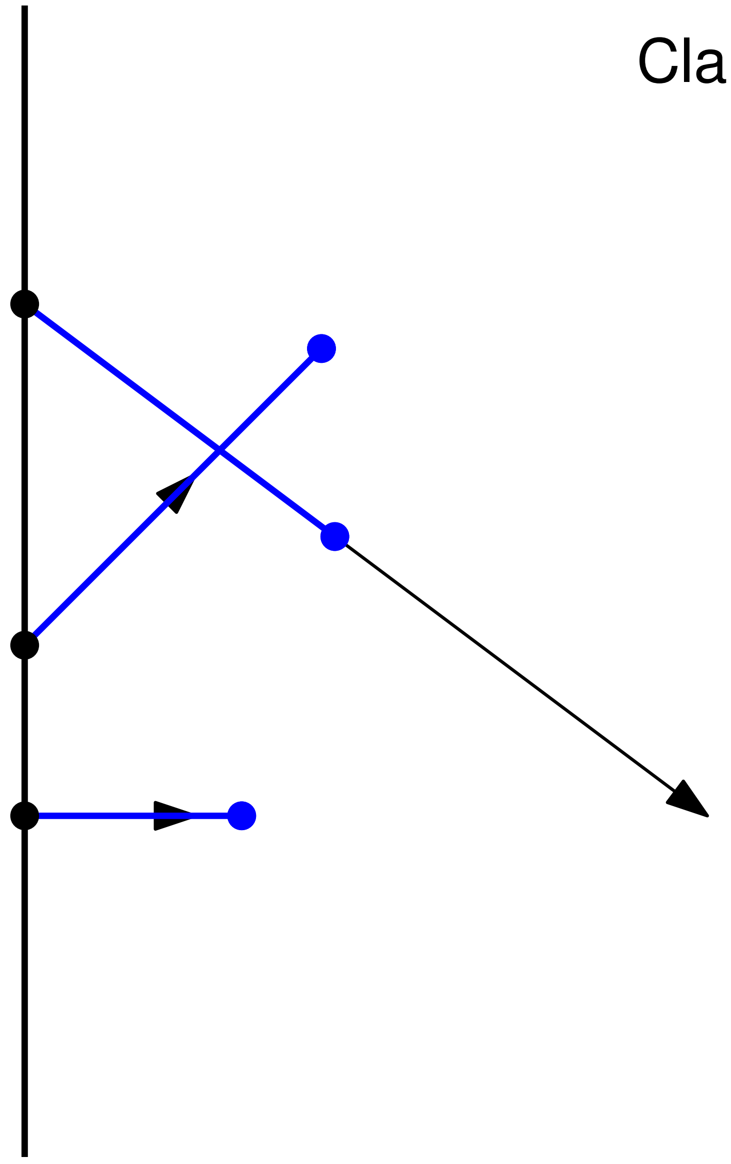


There is a frustration tolerance f. Whenever b_i meets the tracks of b_j, it stops unless b_j was there no more than f time units ago.

Starting Schedules

We are allowed to modify the starting time s_i of each bike b_i.

Claim: We can solve this instance with $f = 0$.

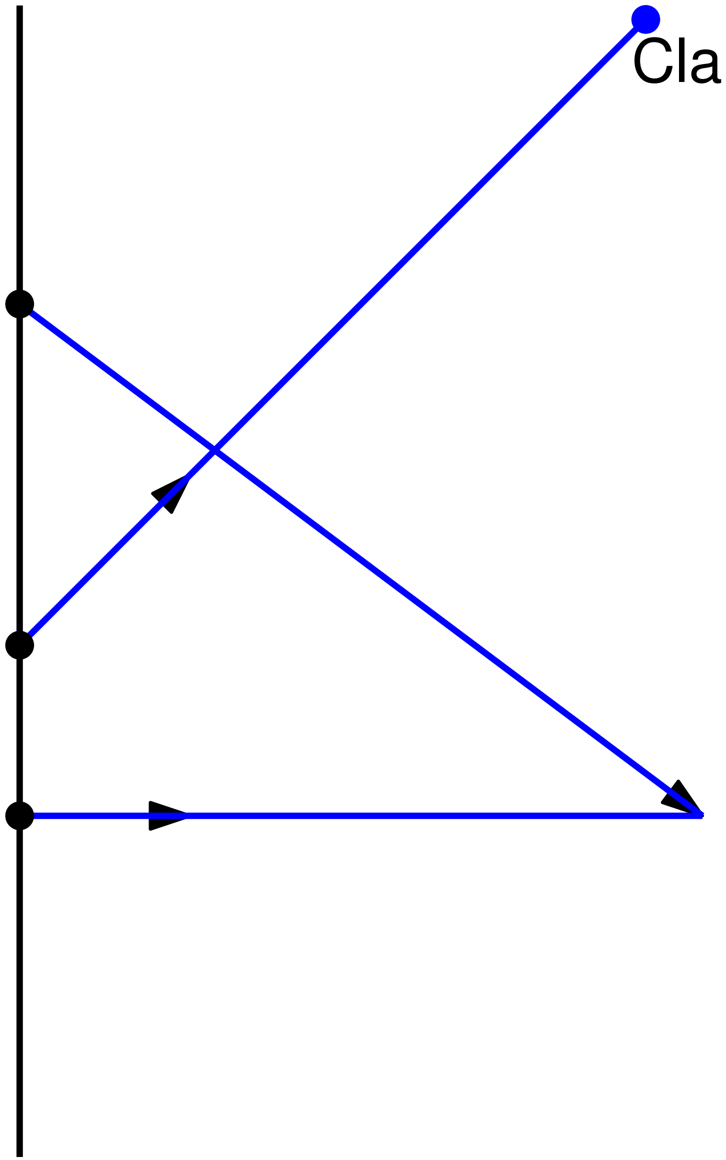


There is a frustration tolerance f. Whenever b_i meets the tracks of b_j, it stops unless b_j was there no more than f time units ago.

Starting Schedules

We are allowed to modify the starting time s_i of each bike b_i.

Claim: We can solve this instance with $f = 0$.



There is a frustration tolerance f.
Whenever b_i meets the tracks of b_j, it stops unless b_j was there no more than f time units ago.

Solving it with Linear Programming

What are the variables?

Solving it with Linear Programming

What are the variables?
The unknowns of the problem are the starting times s_i of each biker and the frustration tolerance f.

Solving it with Linear Programming

What are the variables?
The unknowns of the problem are the starting times s_i of each biker and the frustration tolerance f.

From the specifications, we know that there are at most 101 variables, which is OK to use linear programming.

- It starts with a line that contains a single integer `n` so that $1 \leq n \leq 10^2$. Here n denotes the number of bikers.

Solving it with Linear Programming

What are the variables?
The unknowns of the problem are the starting times s_i of each biker and the frustration tolerance f.

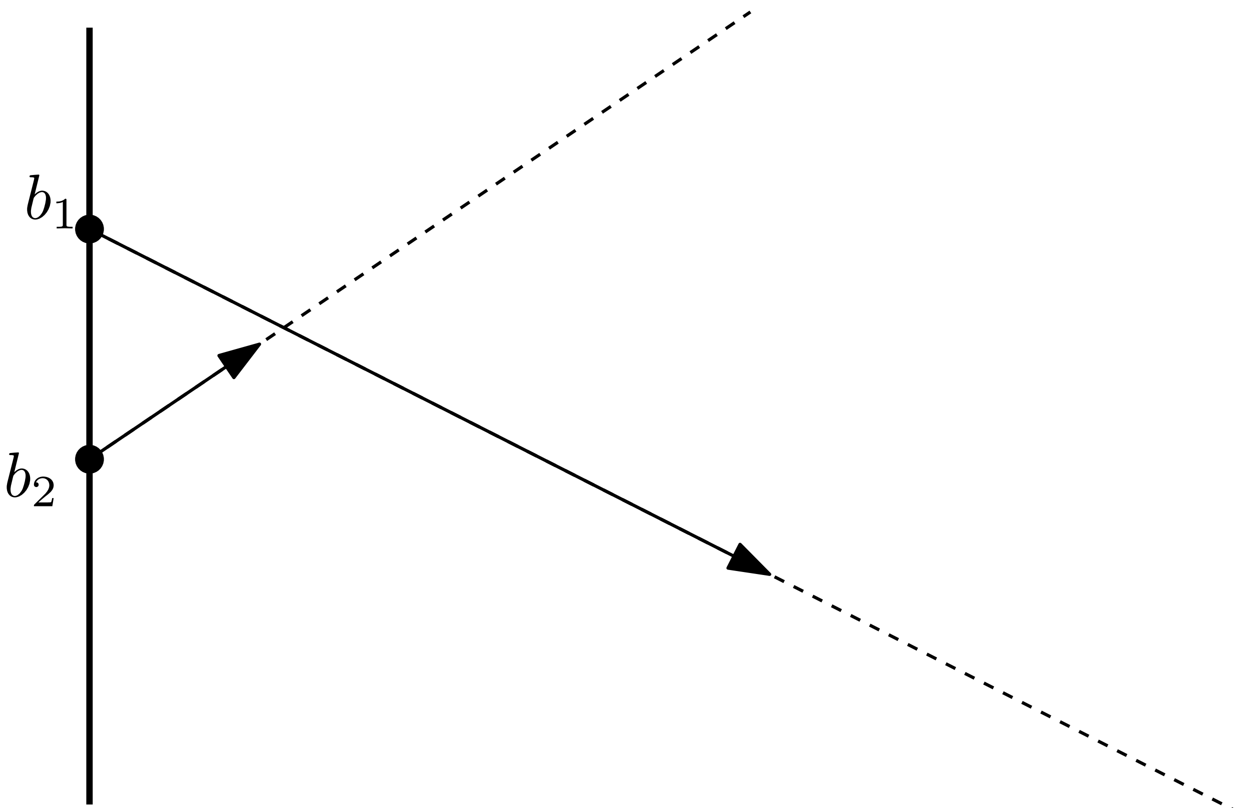
What are the constraints?

Solving it with Linear Programming

What are the variables?
The unknowns of the problem are the starting times s_i of each biker and the frustration tolerance f.

What are the constraints?

For each pair b_i, b_j such that their paths cross, we get a constraint.

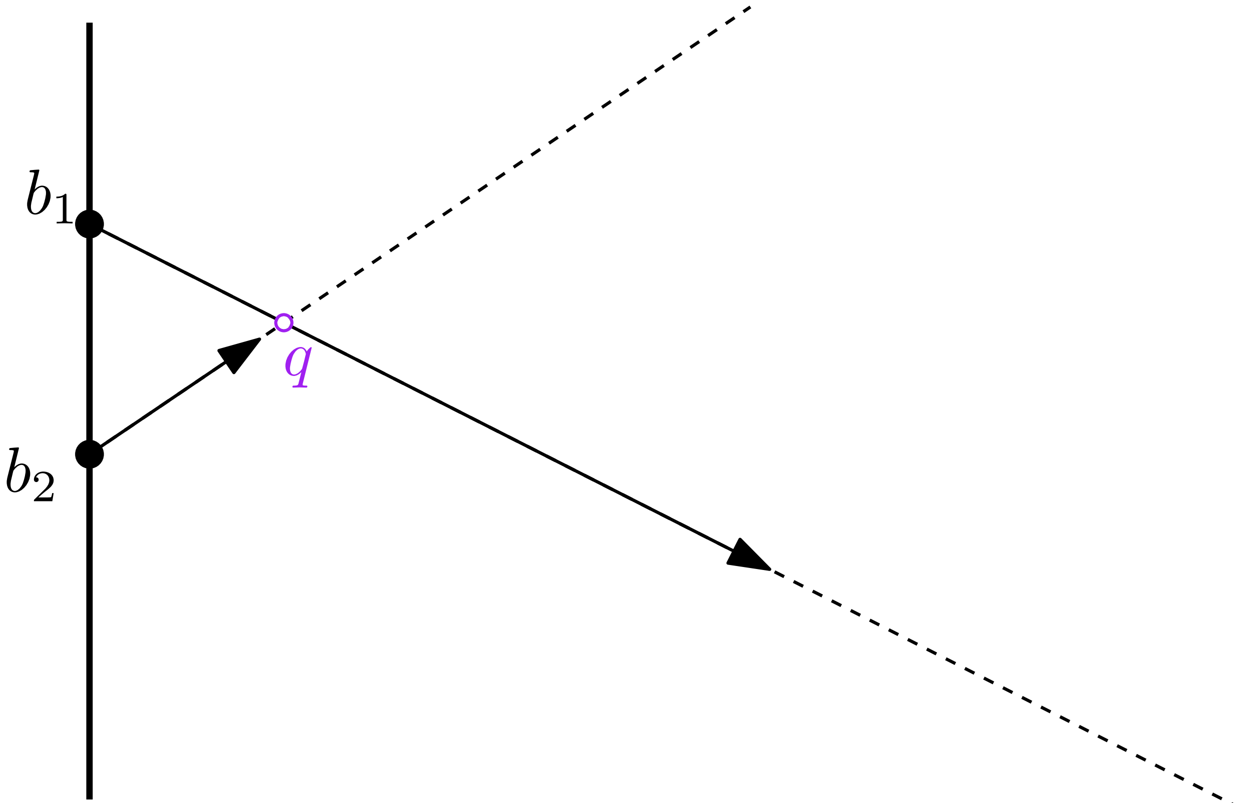


Solving it with Linear Programming

What are the variables?
The unknowns of the problem are the starting times s_i of each biker and the frustration tolerance f.

What are the constraints?

For each pair b_i, b_j such that their paths cross, we get a constraint.



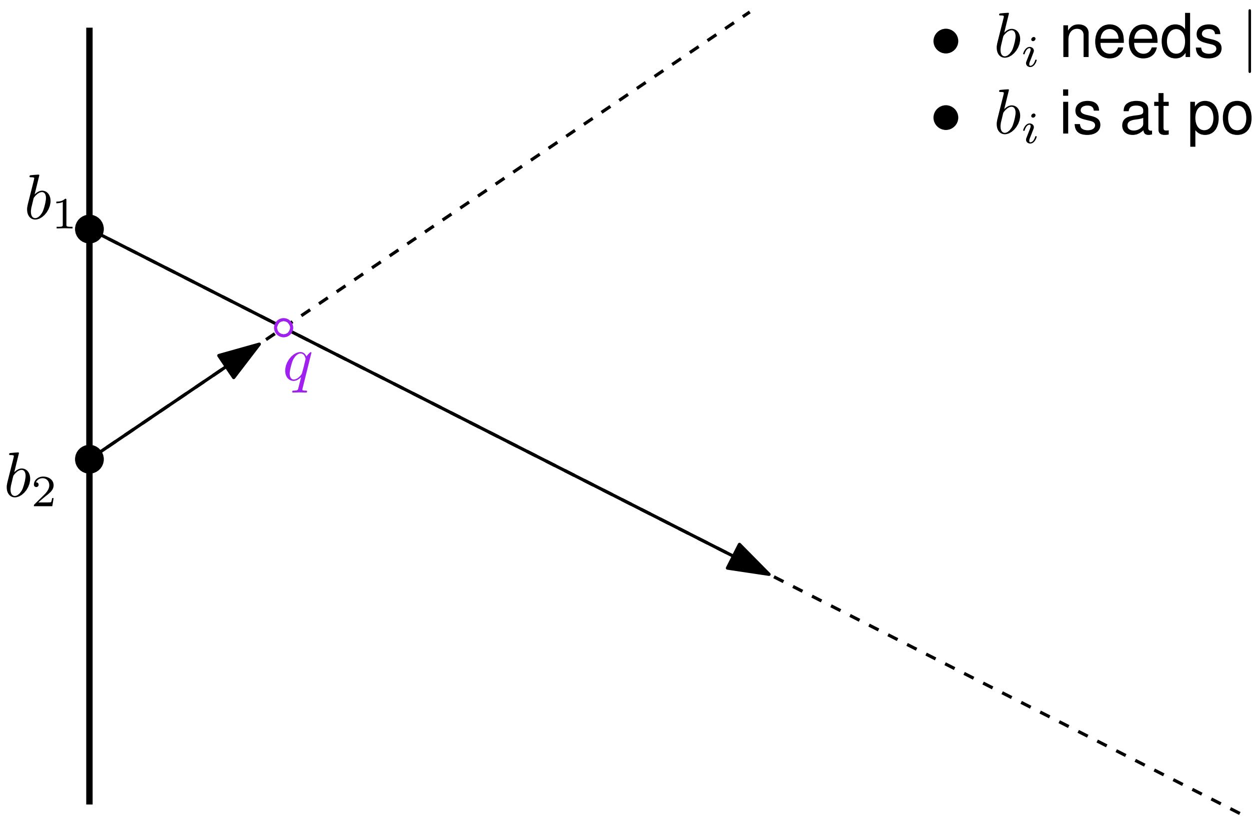
Solving it with Linear Programming

What are the variables?
The unknowns of the problem are the starting times s_i of each biker and the frustration tolerance f.

What are the constraints?

For each pair b_i, b_j such that their paths cross, we get a constraint.

- b_i needs $||b_i - q||$ time to reach q.
- b_i is at position q at time $s_i + ||b_i - q||$.

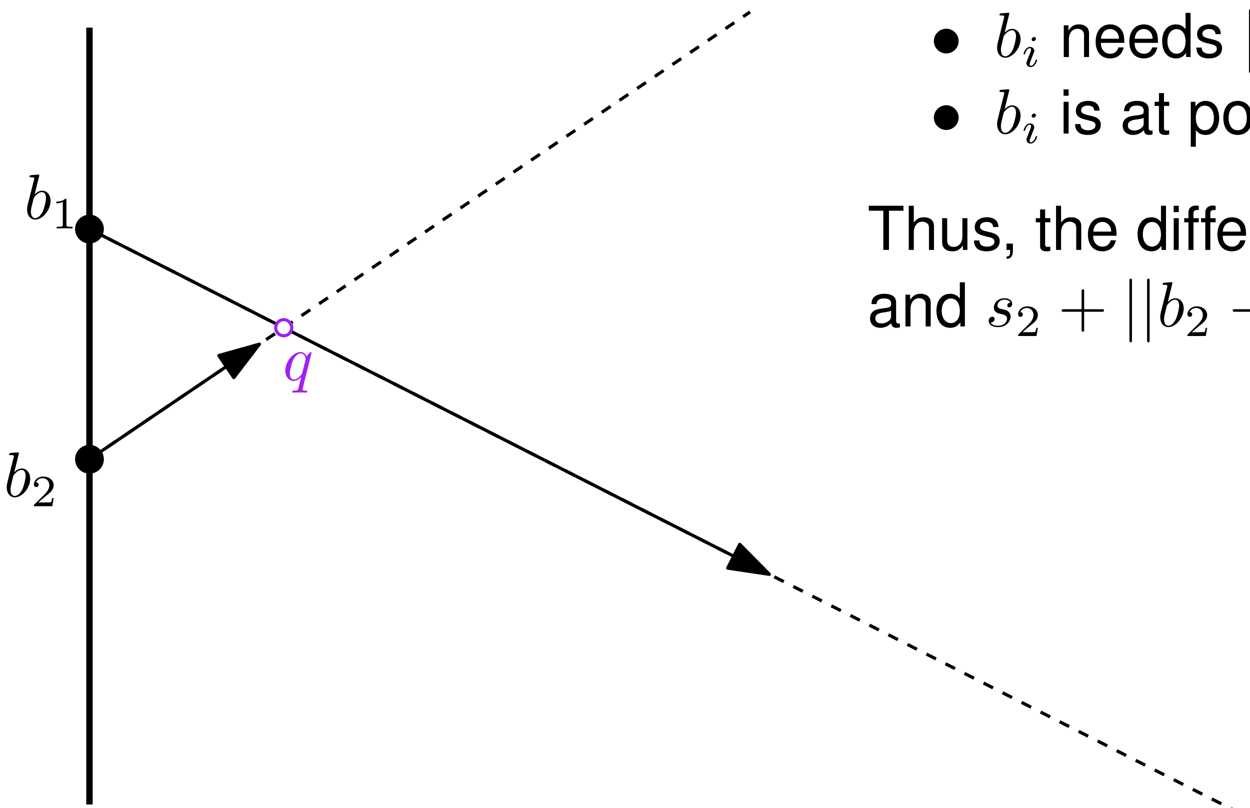


Solving it with Linear Programming

What are the variables?
The unknowns of the problem are the starting times s_i of each biker and the frustration tolerance f.

What are the constraints?

For each pair b_i, b_j such that their paths cross, we get a constraint.



- b_i needs $||b_i - q||$ time to reach q.
- b_i is at position q at time $s_i + ||b_i - q||$.

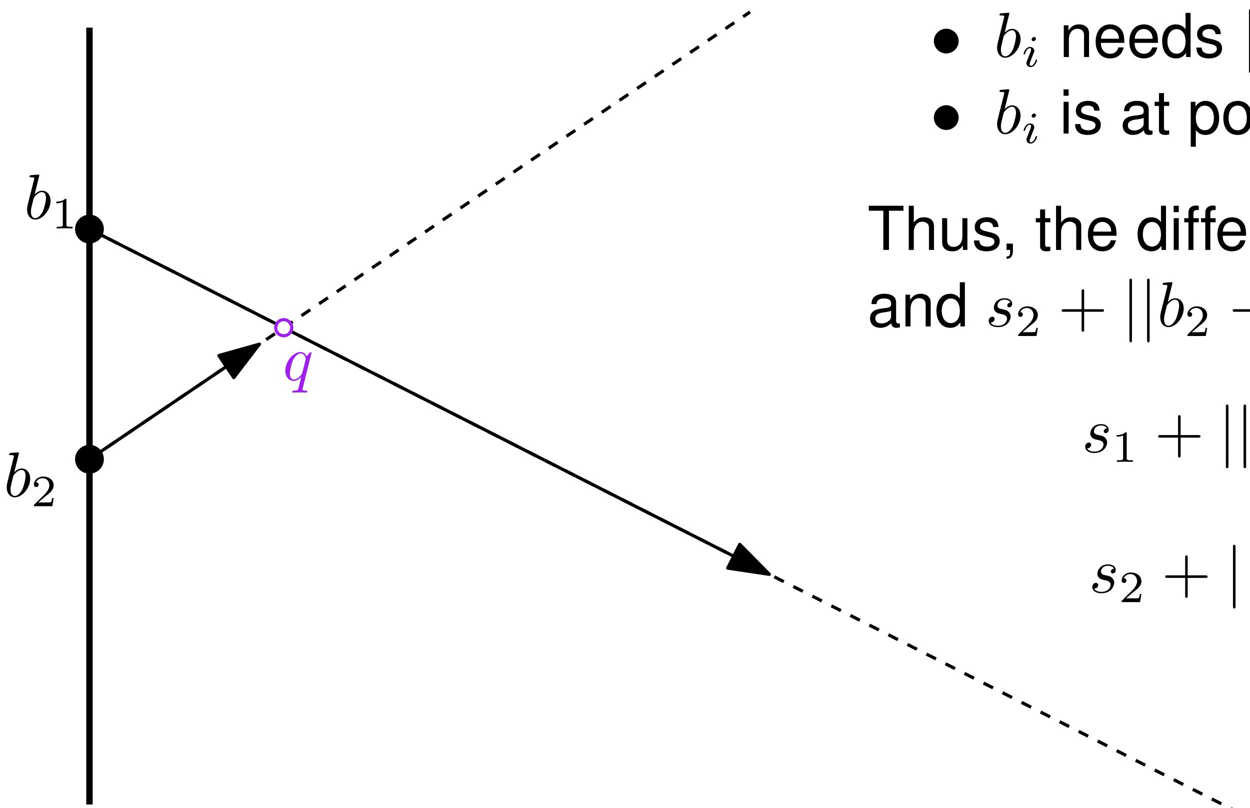
Thus, the difference between $s_1 + ||b_1 - q||$ and $s_2 + ||b_2 - q||$ can be at most f.

Solving it with Linear Programming

What are the variables?
The unknowns of the problem are the starting times s_i of each biker and the frustration tolerance f.

What are the constraints?

For each pair b_i, b_j such that their paths cross, we get a constraint.



- b_i needs $||b_i - q||$ time to reach q.
- b_i is at position q at time $s_i + ||b_i - q||$.

Thus, the difference between $s_1 + ||b_1 - q||$ and $s_2 + ||b_2 - q||$ can be at most f.

$$s_1 + ||b_1 - q|| \leq s_2 + ||b_2 - q|| + f,$$

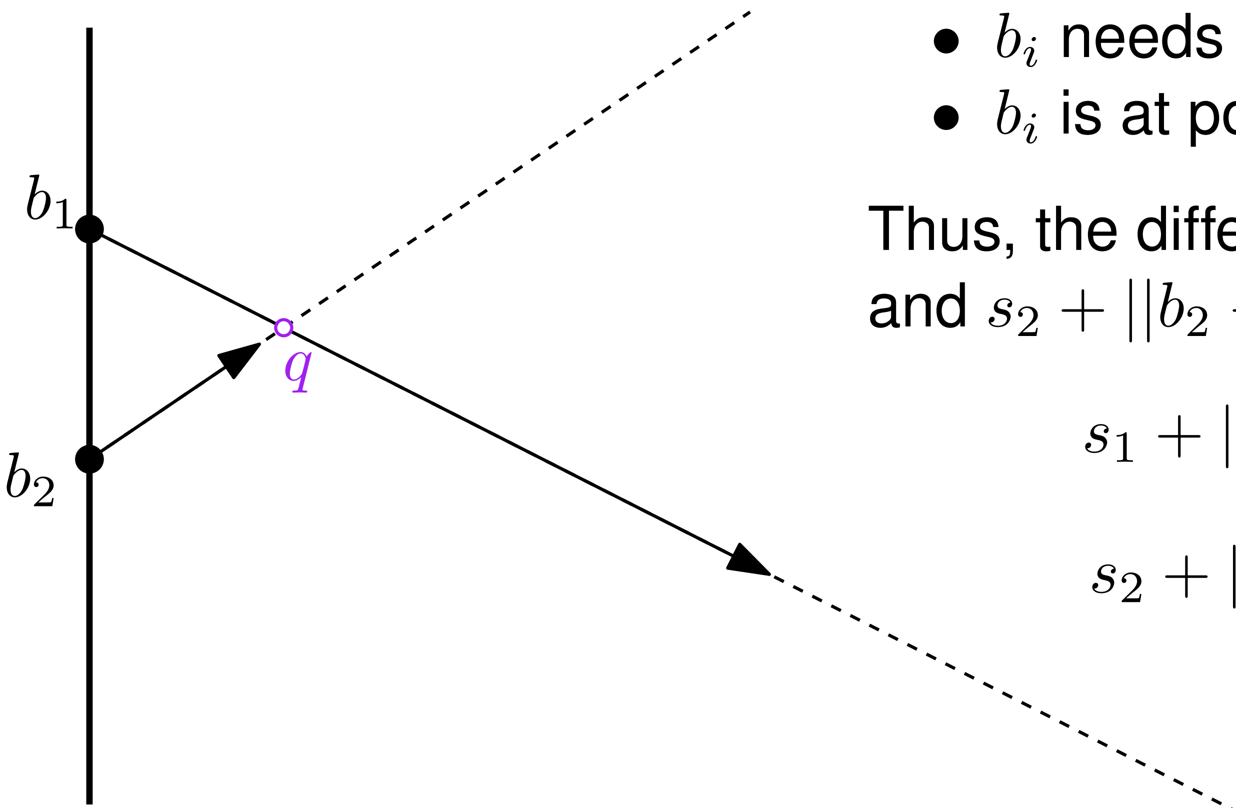
$$s_2 + ||b_2 - q|| \leq s_1 + ||b_1 - q|| + f$$

Solving it with Linear Programming

What are the variables?
The unknowns of the problem are the starting times s_i of each biker and the frustration tolerance f.

What are the constraints?

For each pair b_i, b_j such that their paths cross, we get a constraint.



- b_i needs $||b_i - q||$ time to reach q.
- b_i is at position q at time $s_i + ||b_i - q||$.

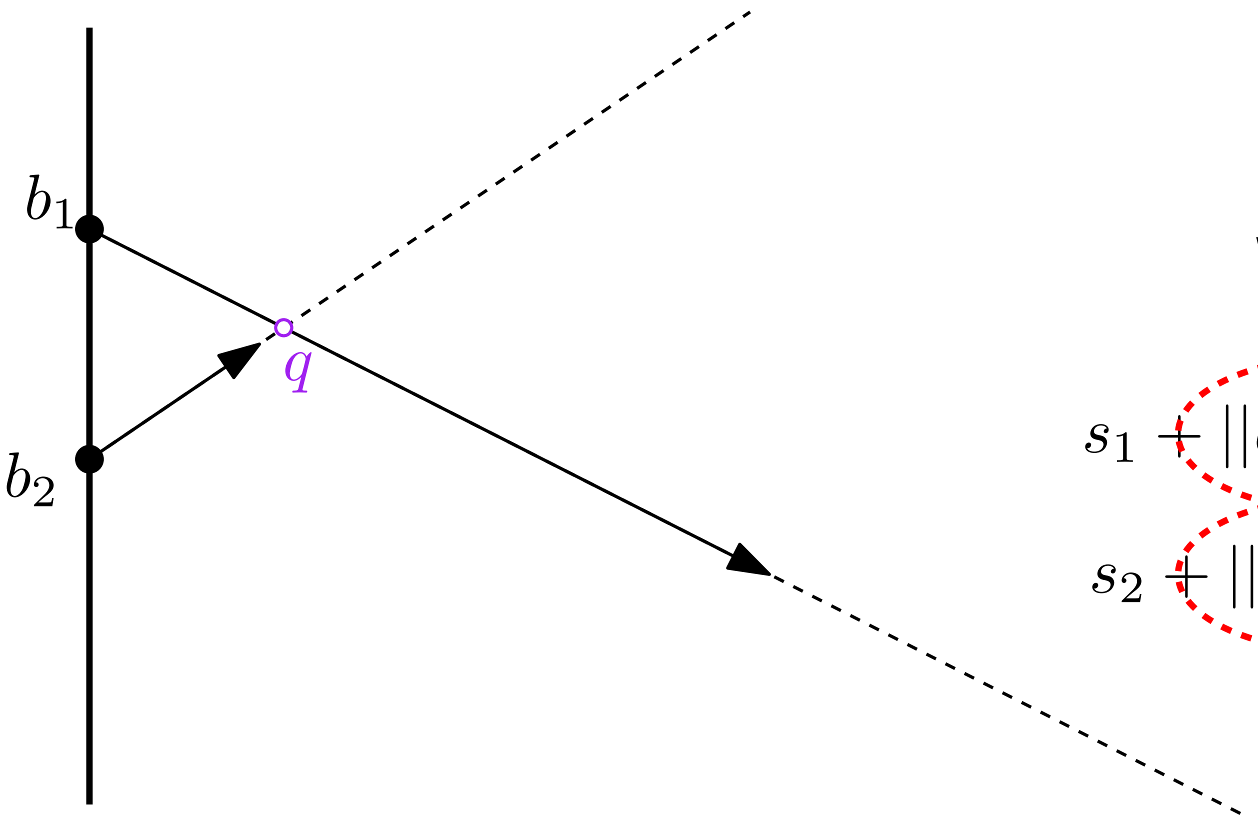
Thus, the difference between $s_1 + ||b_1 - q||$ and $s_2 + ||b_2 - q||$ can be at most f.

$$s_1 + ||b_1 - q|| \leq s_2 + ||b_2 - q|| + f,$$

$$s_2 + ||b_2 - q|| \leq s_1 + ||b_1 - q|| + f$$

There are $O(n^2)$ constraints, which is at most 10,000.

Solving it with Linear Programming



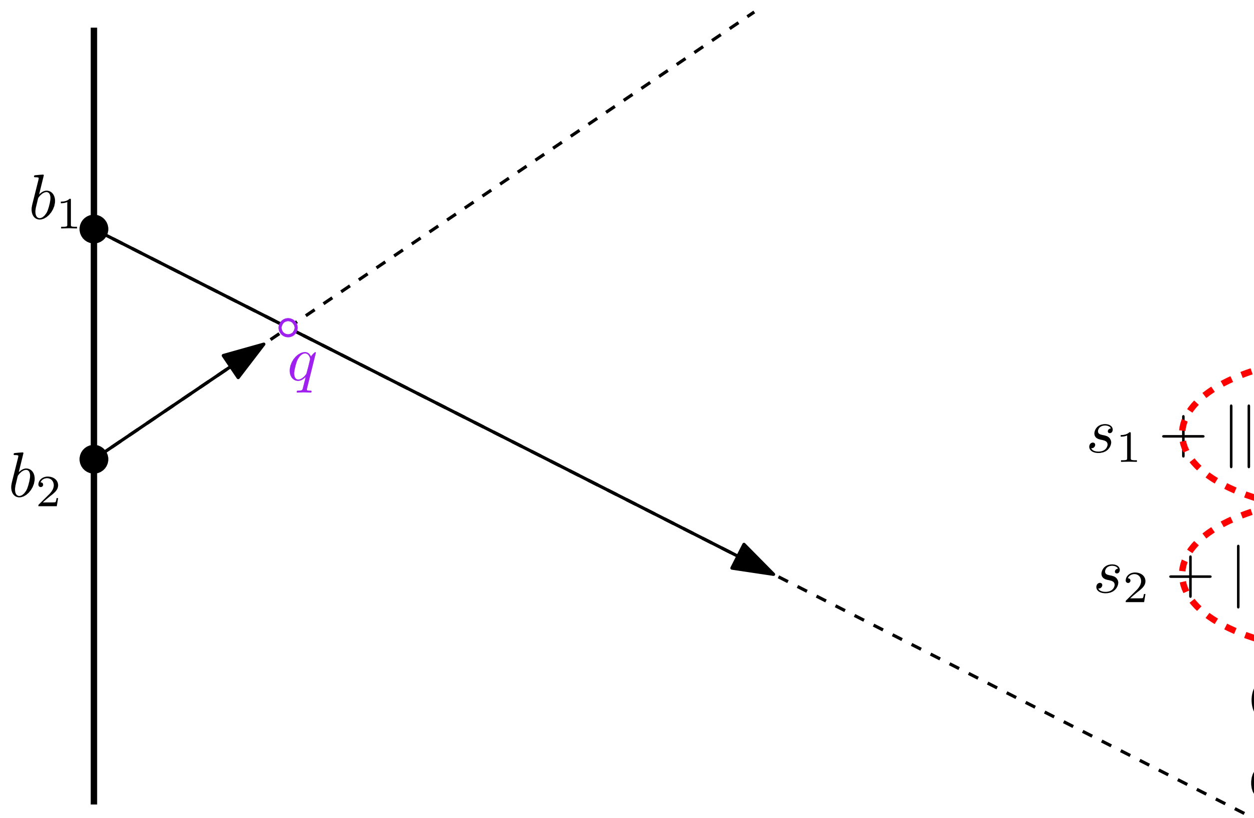
For each pair b_i, b_j such that their paths cross, we get a constraint.

What are these terms?

$$s_1 + ||b_1 - q|| \leq s_2 + ||b_2 - q|| + f,$$
$$s_2 + ||b_2 - q|| \leq s_1 + ||b_1 - q|| + f$$

Solving it with Linear Programming

For each pair b_i, b_j such that their paths cross, we get a constraint.



What are these terms?

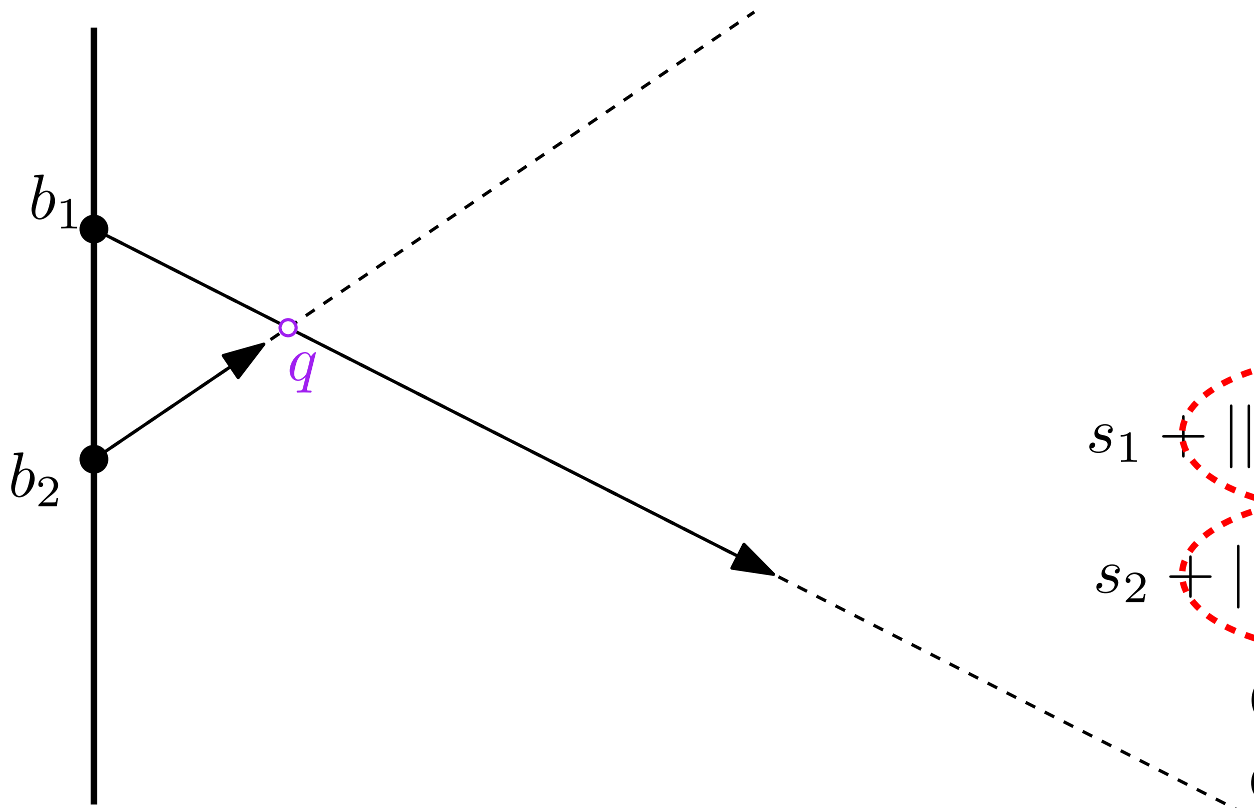
$$s_1 + ||b_1 - q|| \leq s_2 + ||b_2 - q|| + f,$$
$$s_2 + ||b_2 - q|| \leq s_1 + ||b_1 - q|| + f$$

One needs square roots to compute these constants.

Solving it with Linear Programming

In addition, we need all variables to be non-negative. Then, we minimize f subject to these constraints.

For each pair b_i, b_j such that their paths cross, we get a constraint.

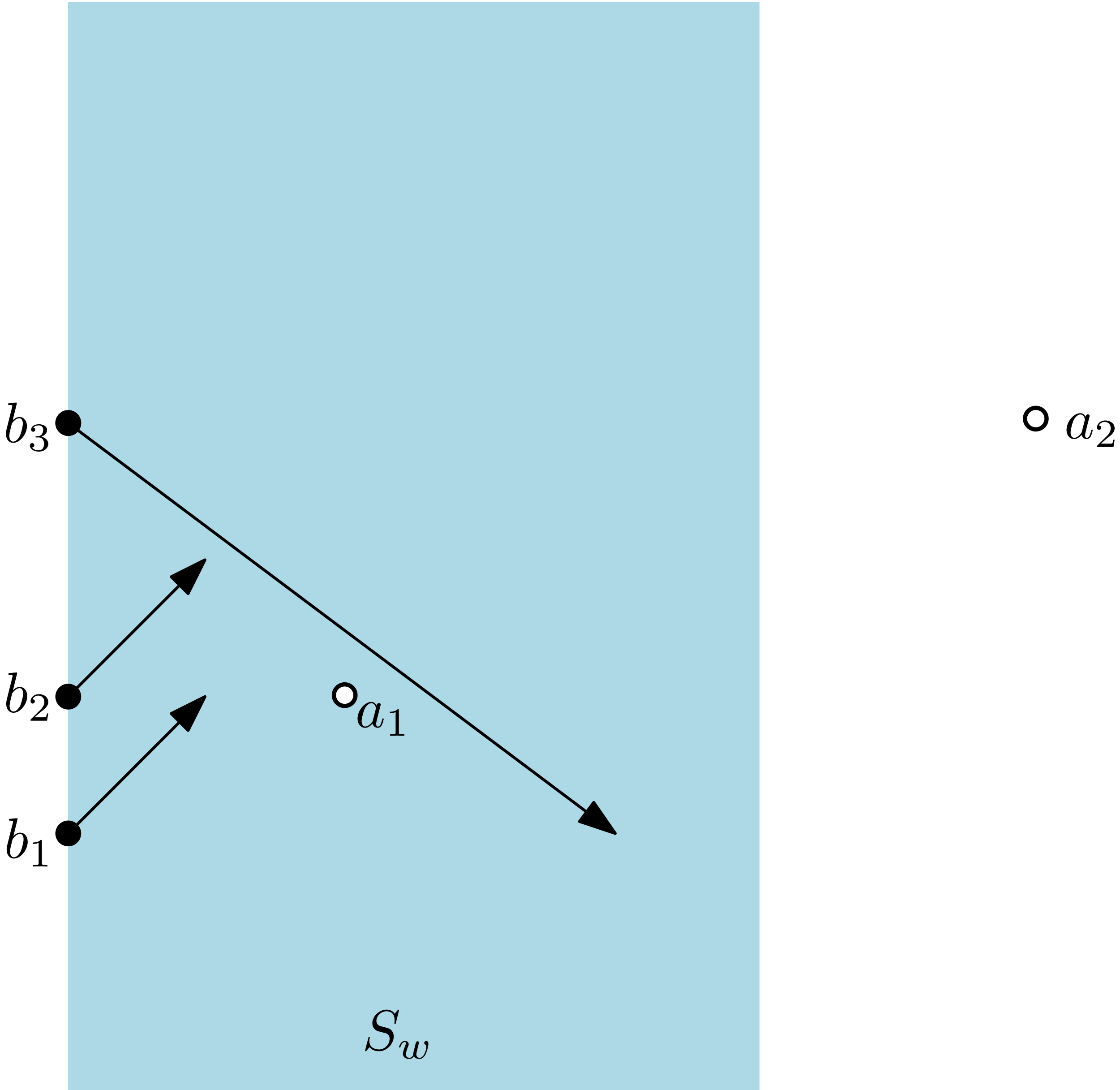


What are these terms?

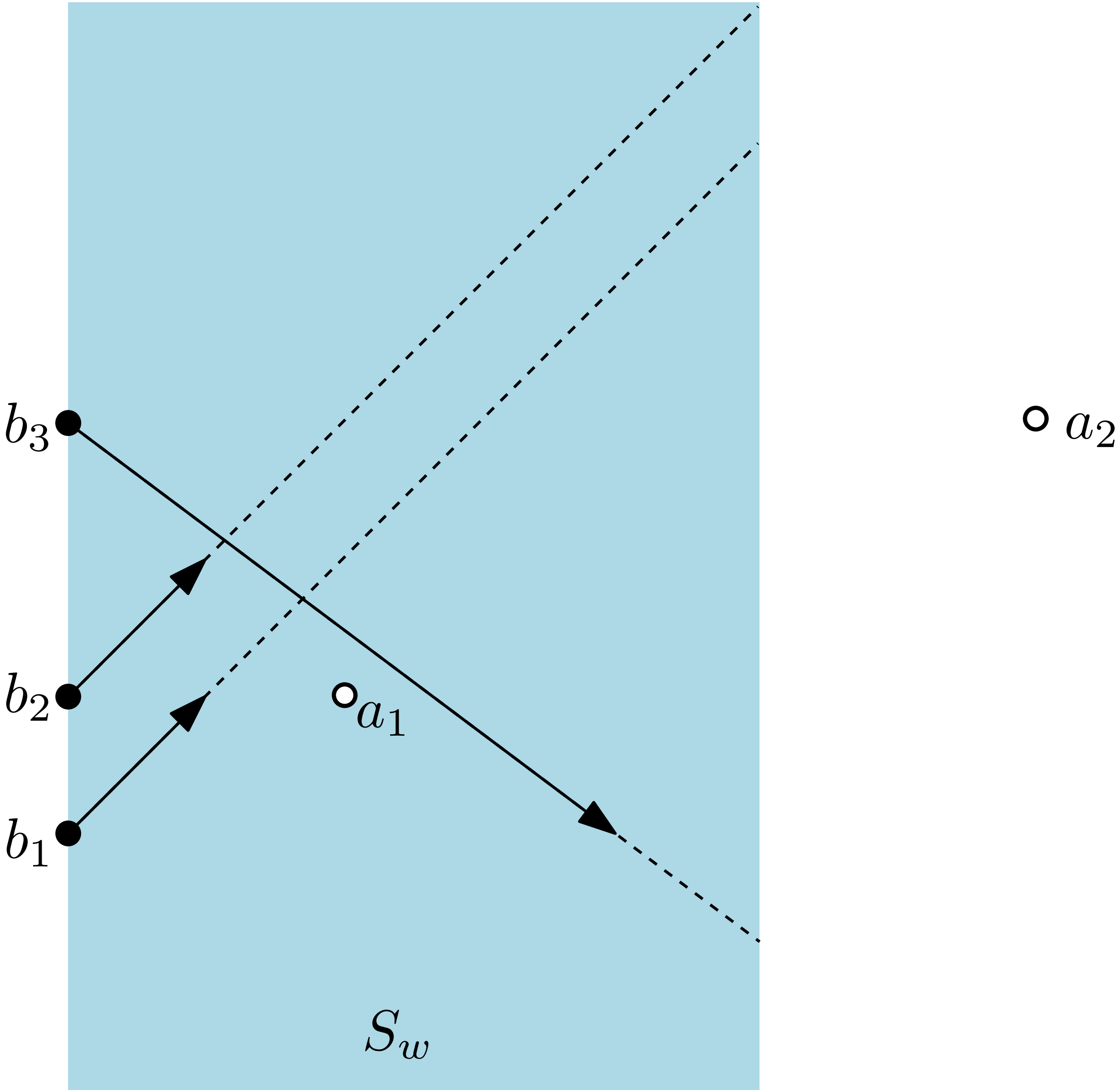
$$s_1 + ||b_1 - q|| \leq s_2 + ||b_2 - q|| + f,$$
$$s_2 + ||b_2 - q|| \leq s_1 + ||b_1 - q|| + f$$

One needs square roots to compute these constants.

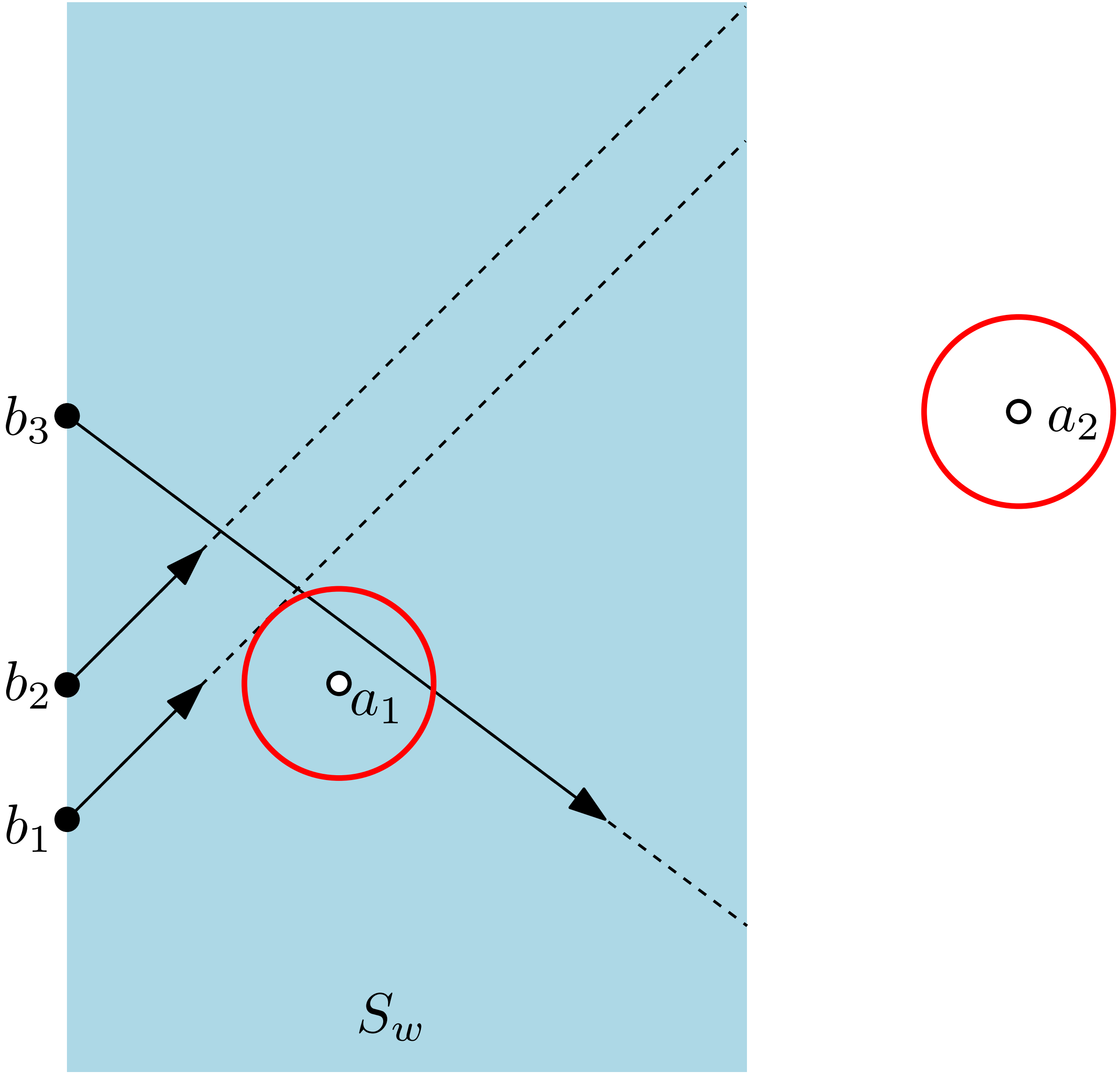
Beethoven



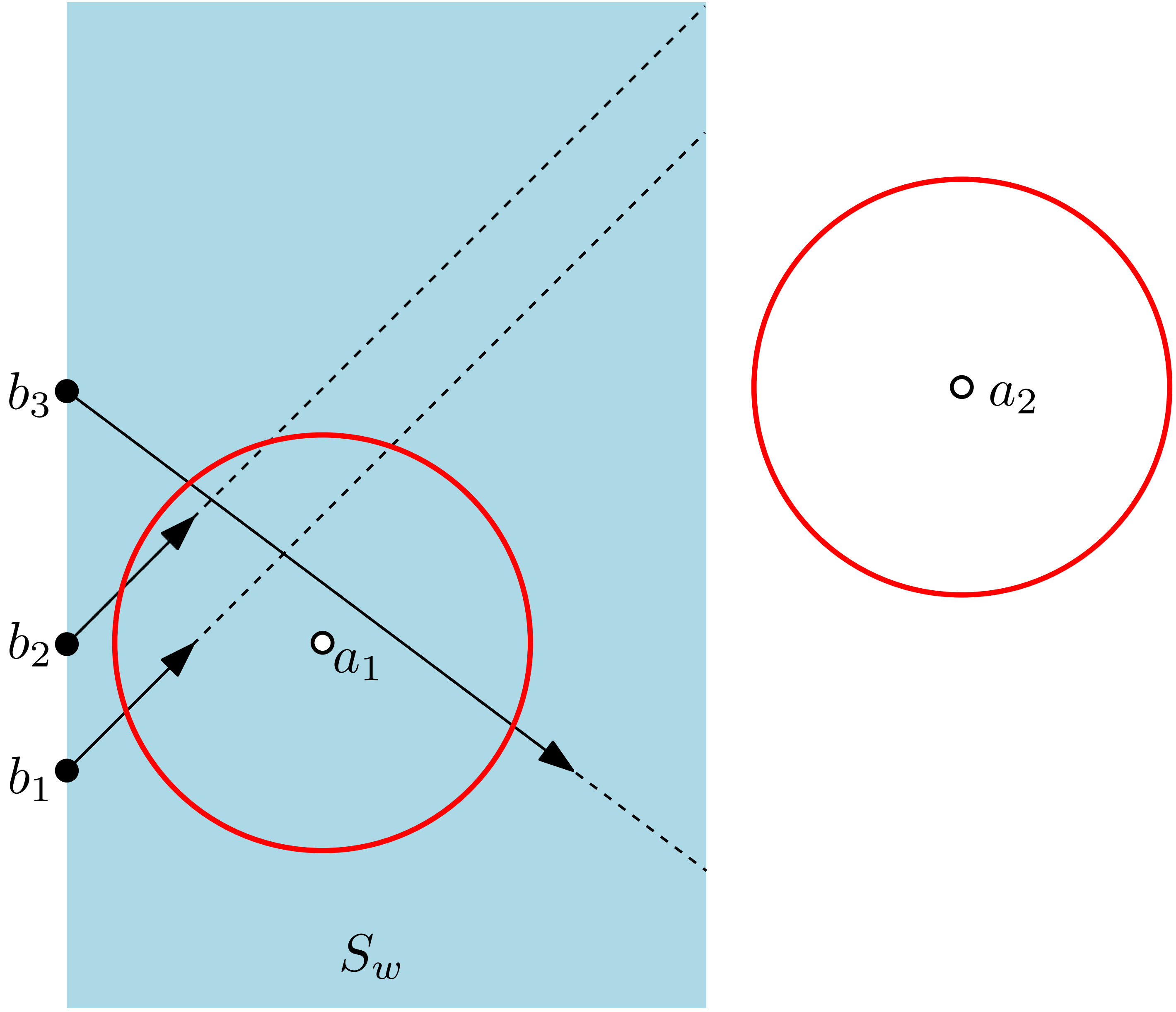
Beethoven



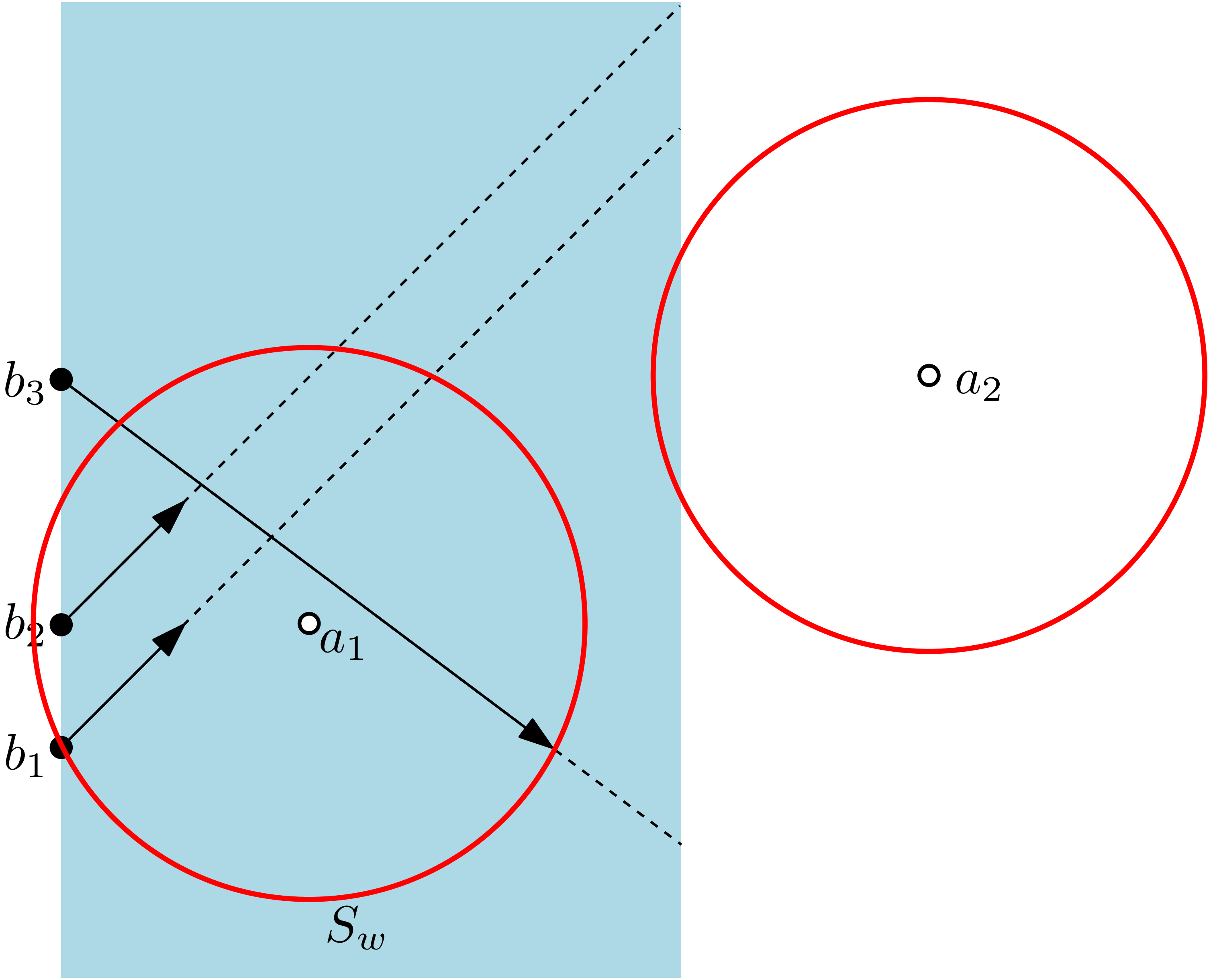
Beethoven



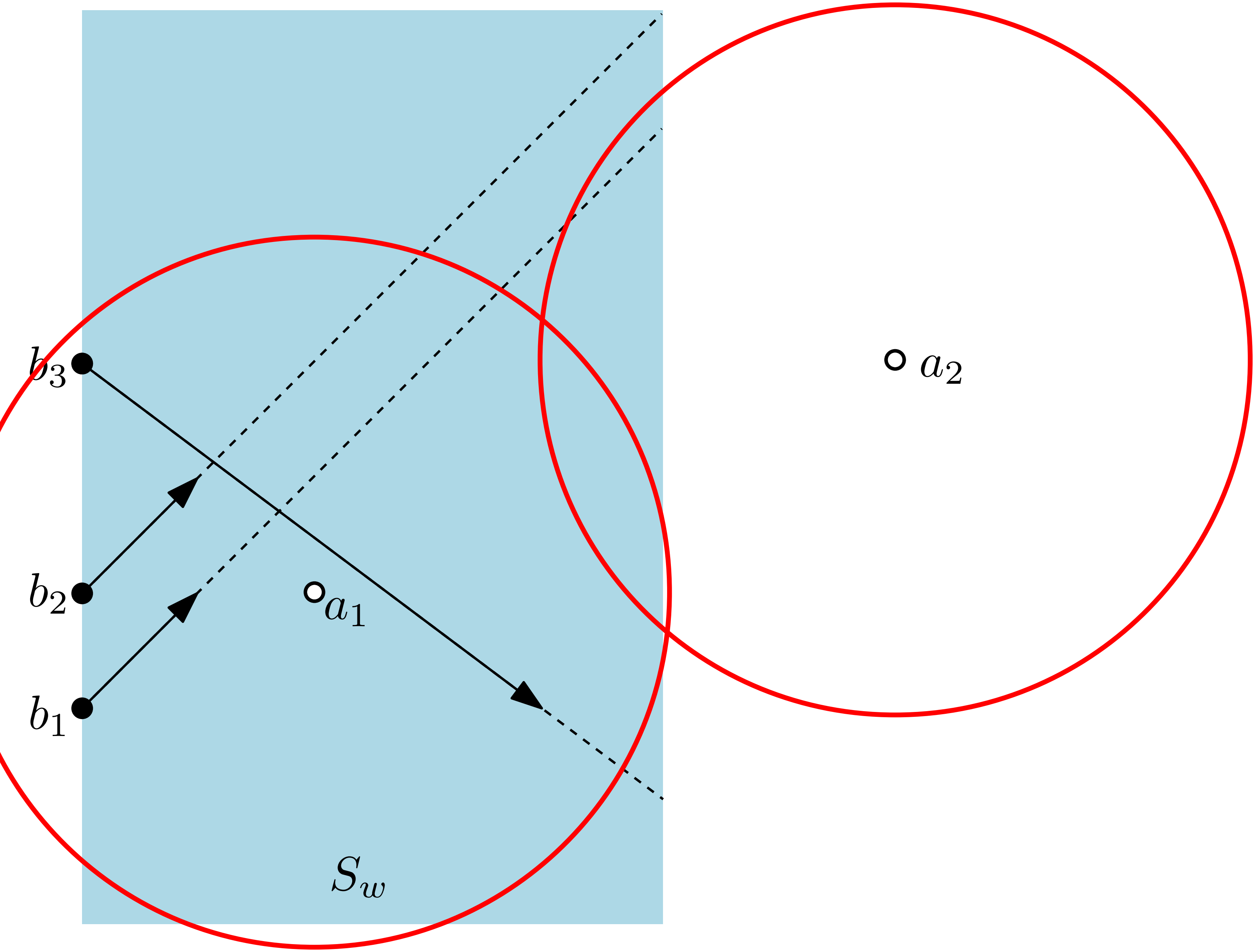
Beethoven



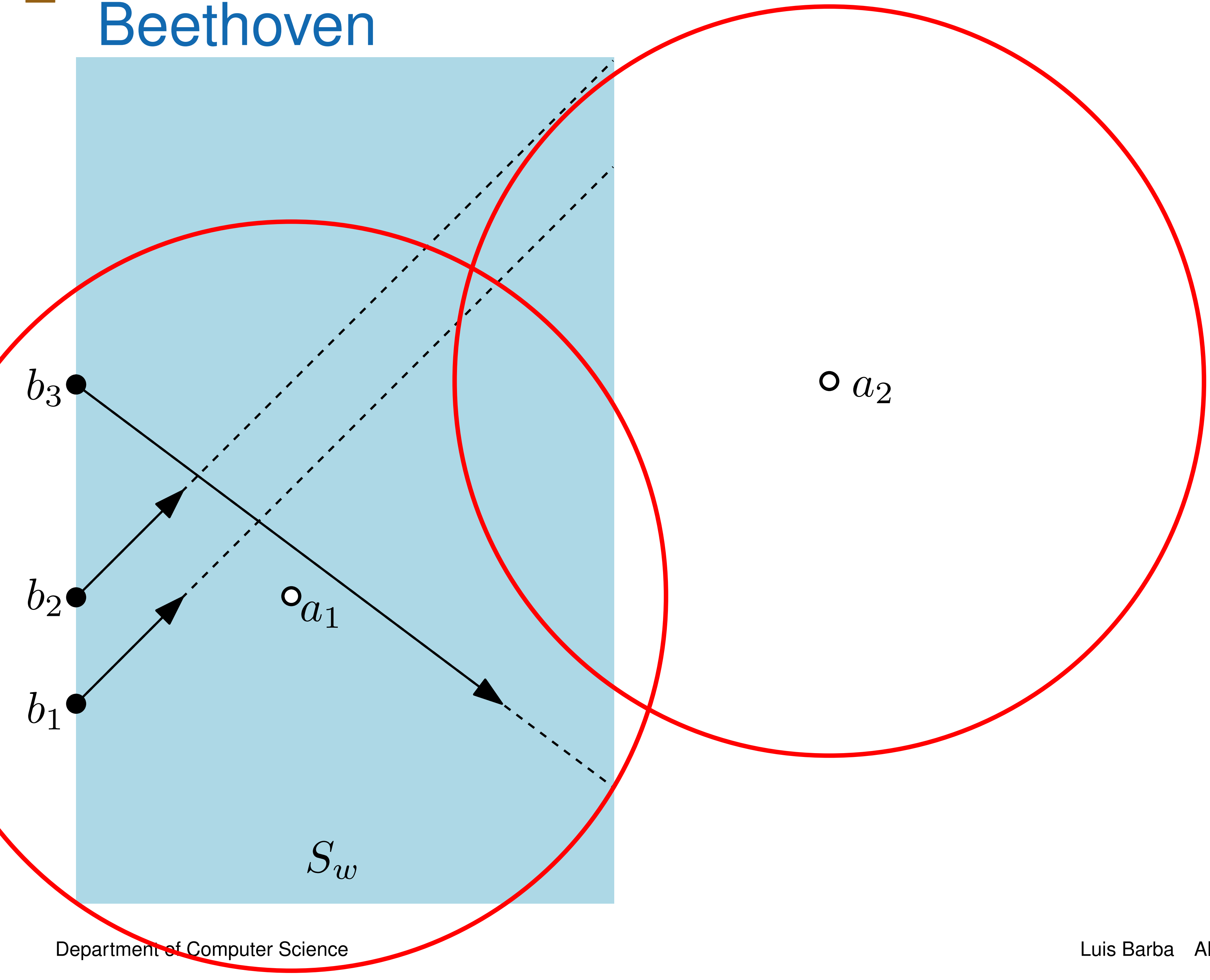
Beethoven



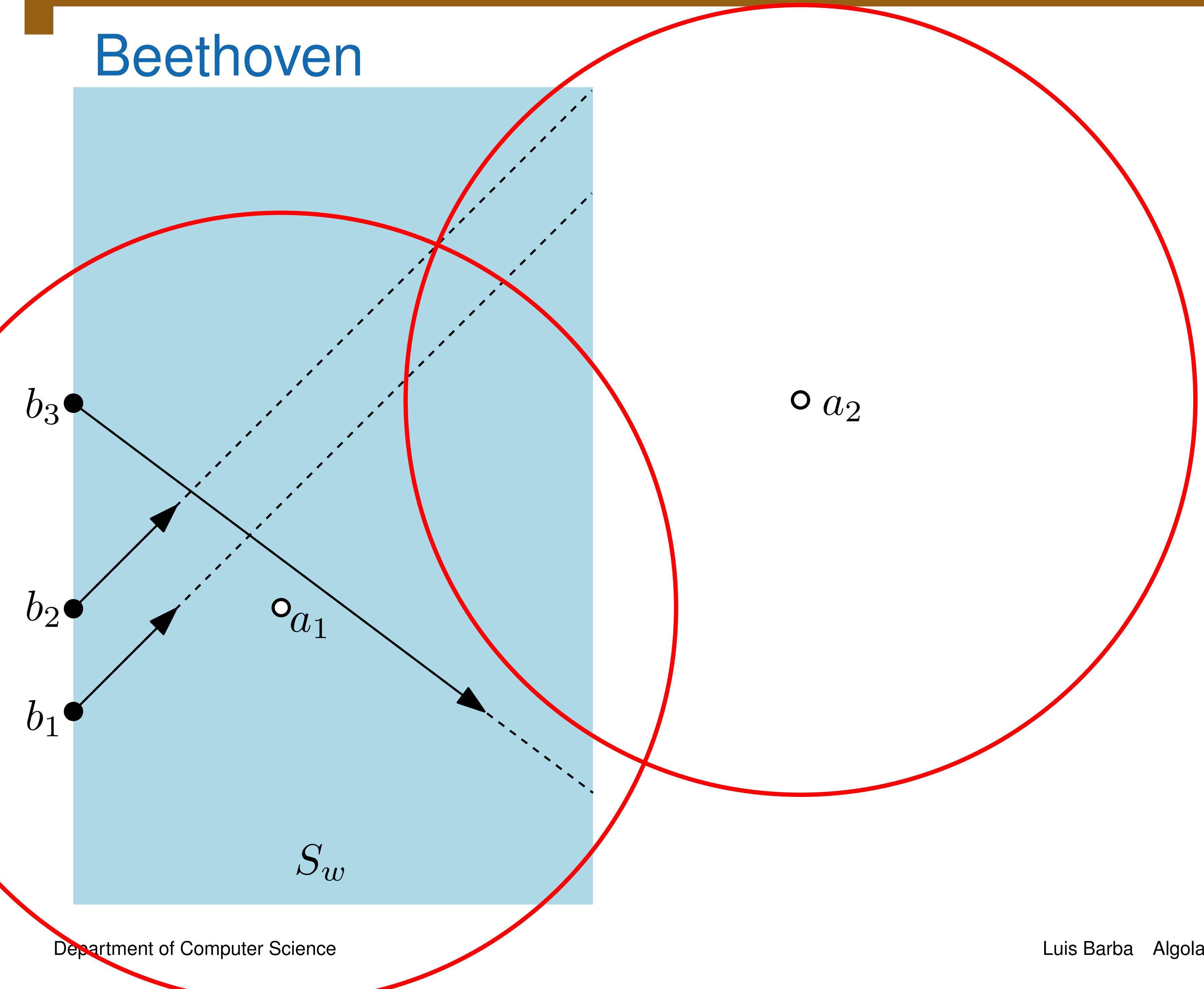
Beethoven



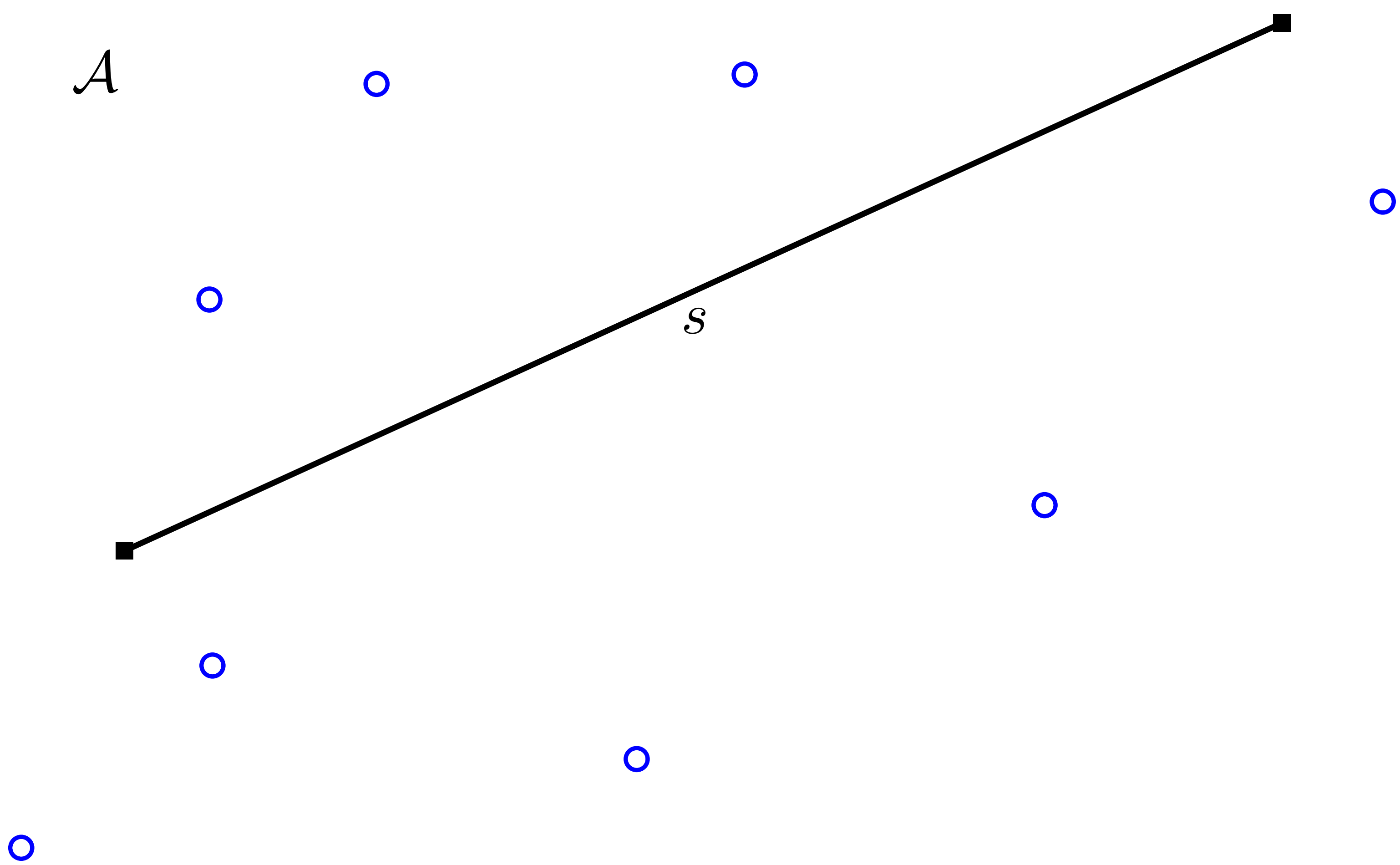
Beethoven



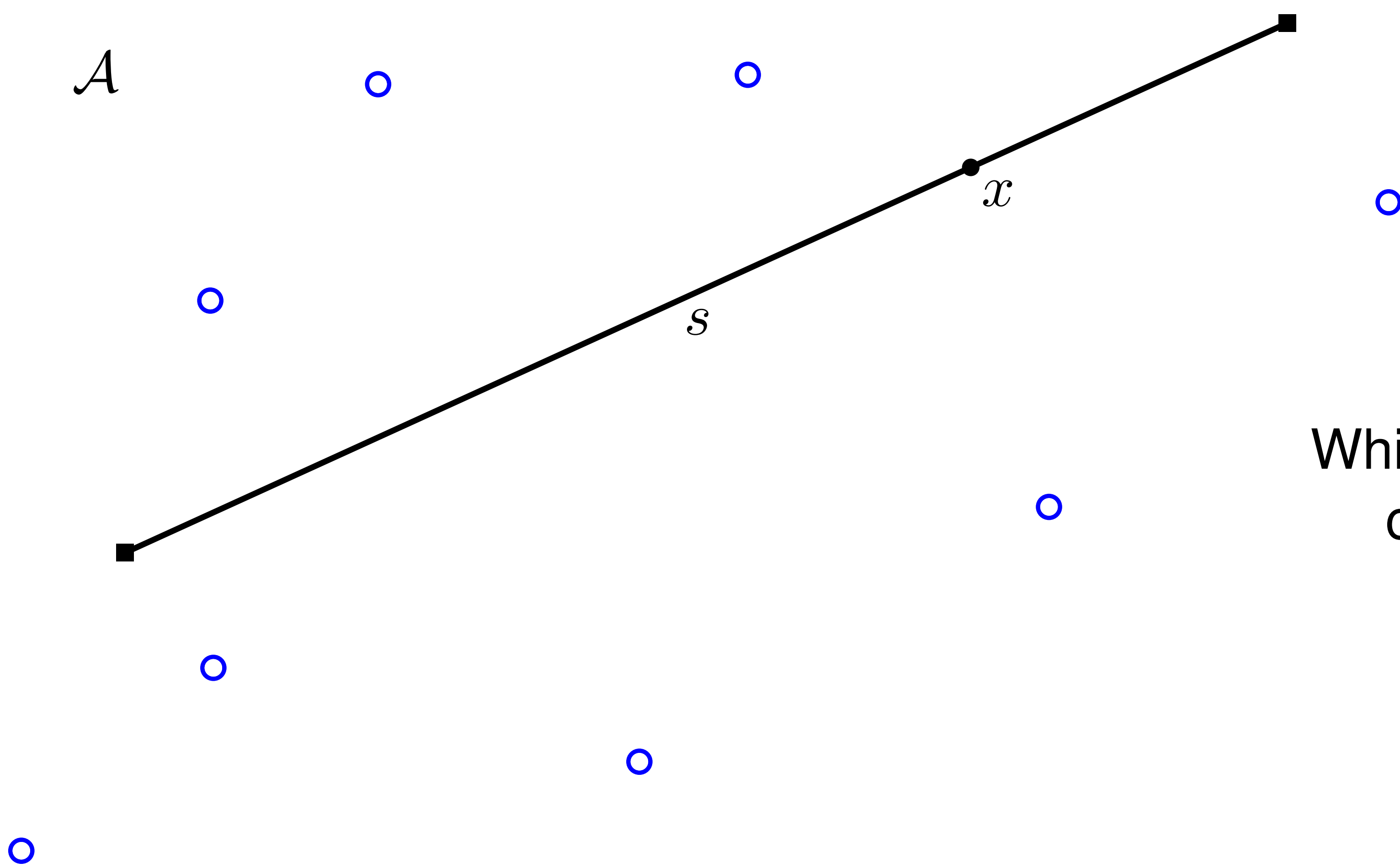
Beethoven



Working with a single segment

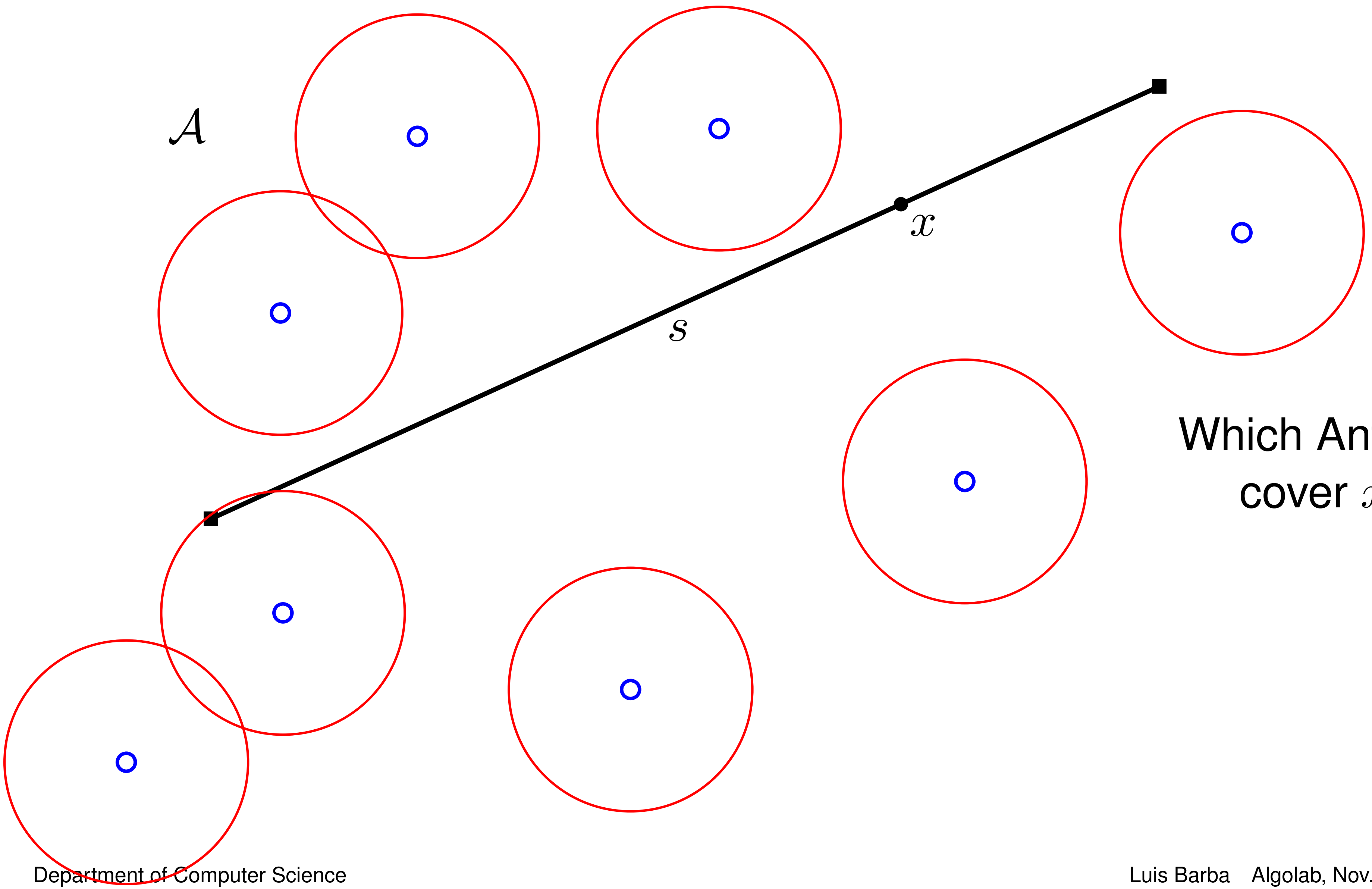


Working with a single segment



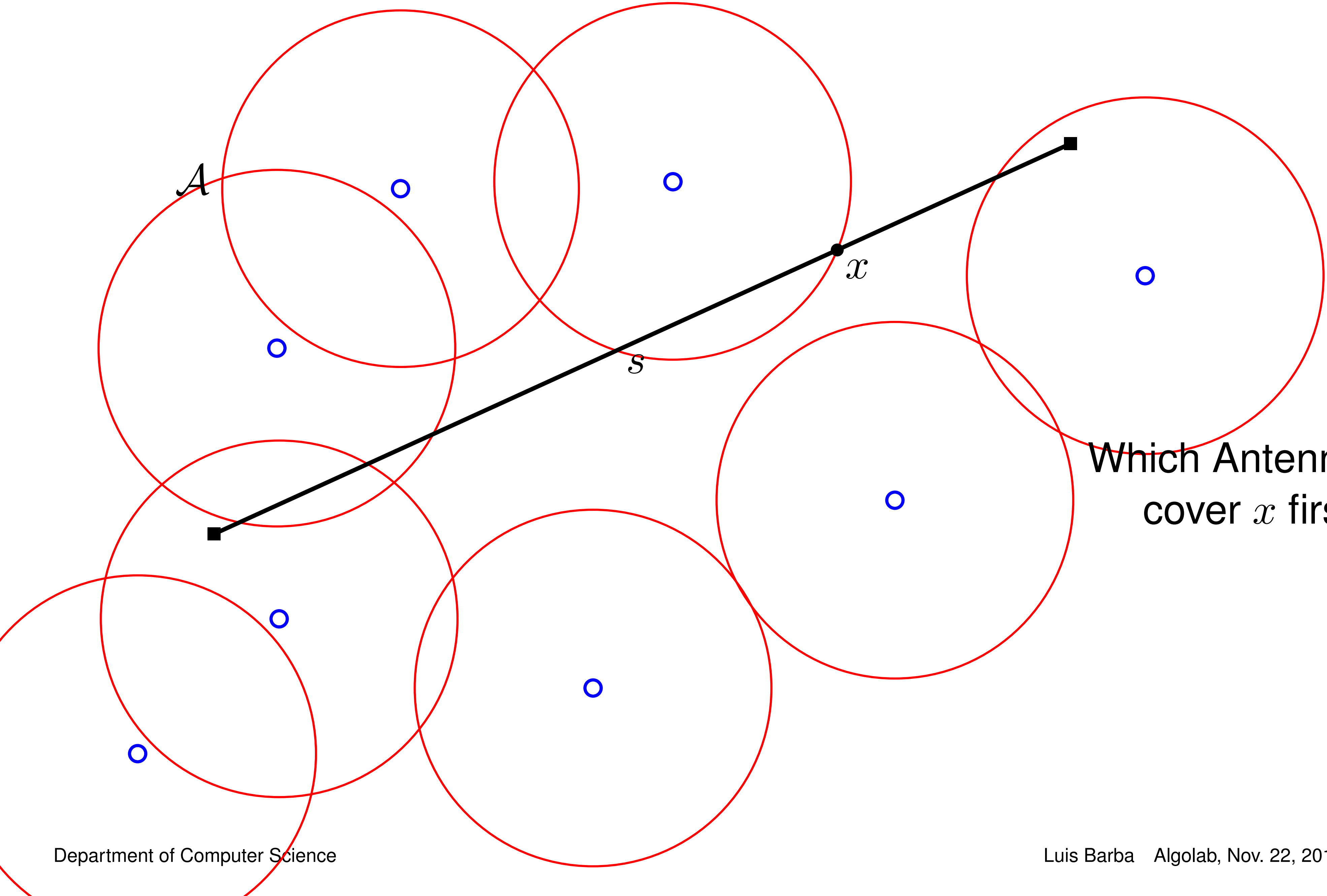
Which Antenna will cover x first?

Working with a single segment



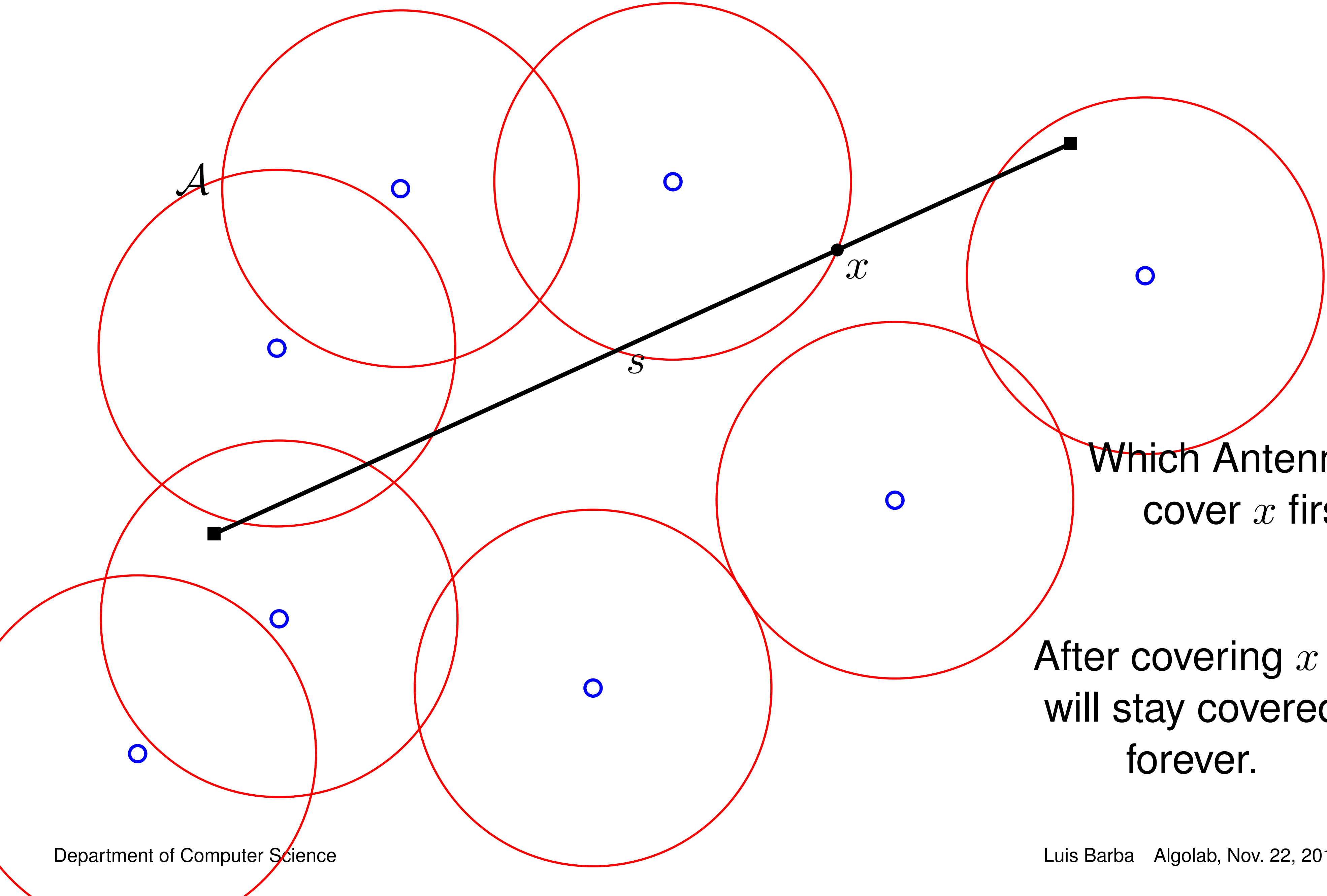
Which Antenna will cover x first?

Working with a single segment



Which Antenna will cover x first?

Working with a single segment



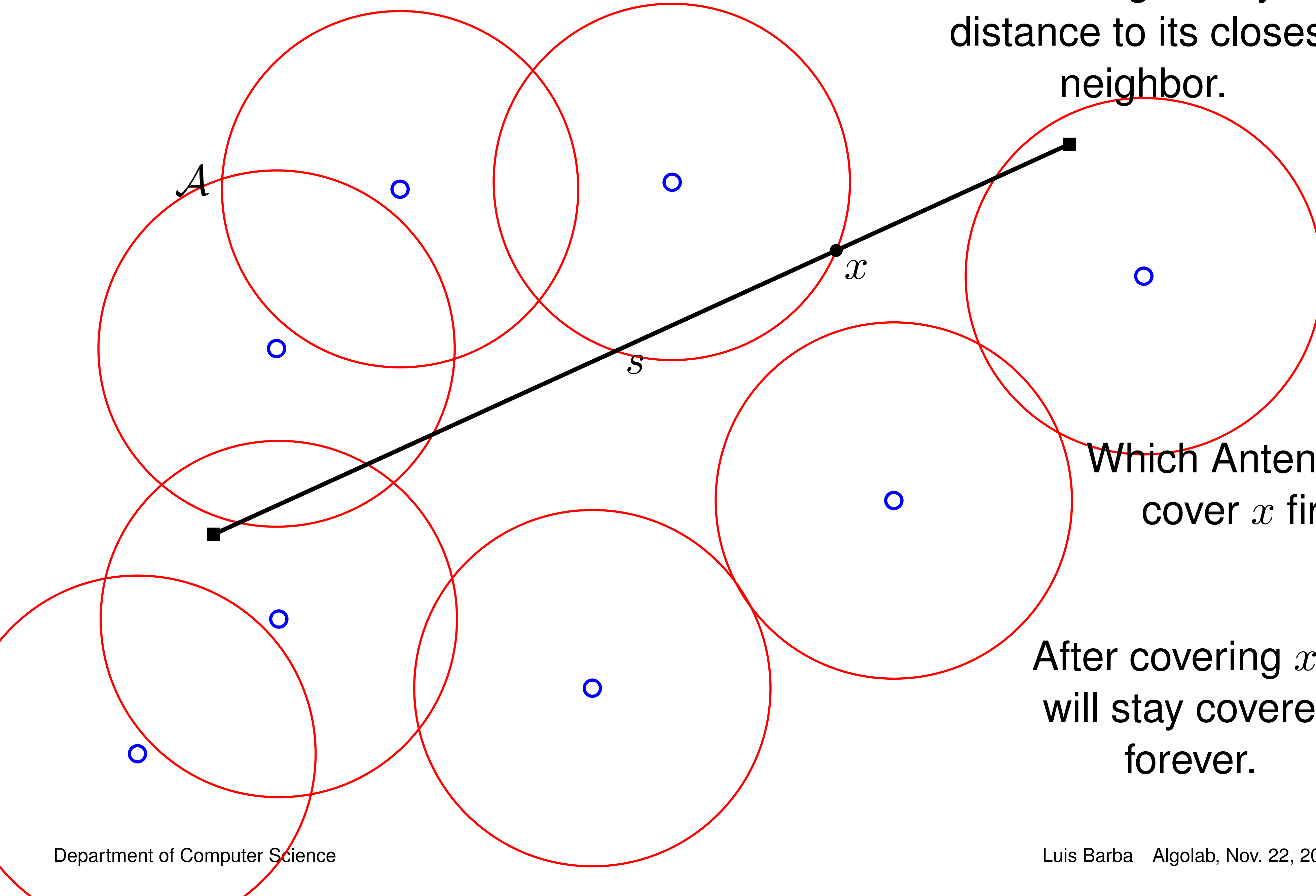
Which Antenna will cover x first?

After covering x it will stay covered forever.

ETH zürich

Working with a single segment

The minimum radius to cover x is given by the distance to its closest neighbor.

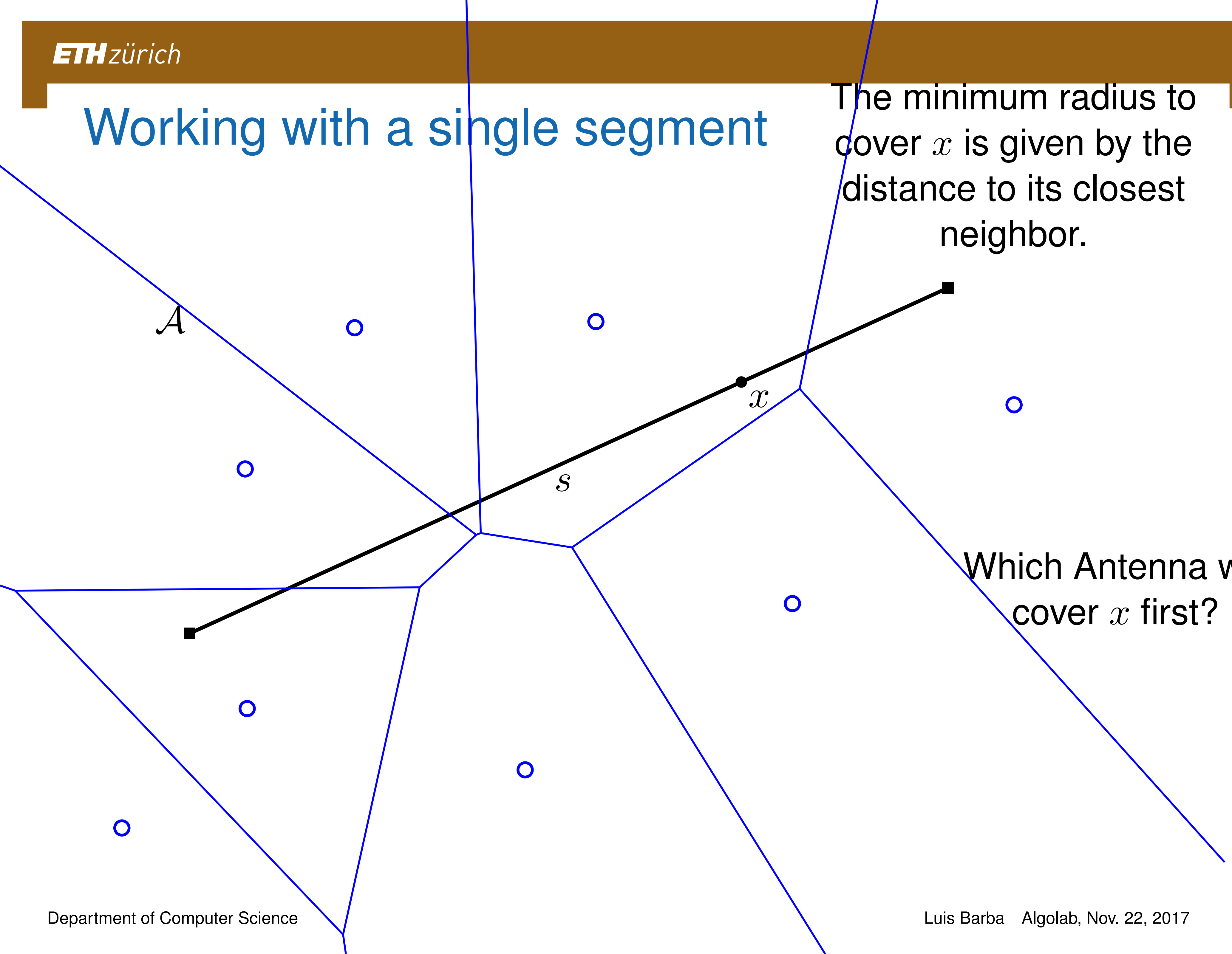


Which Antenna will cover x first?

After covering x it will stay covered forever.

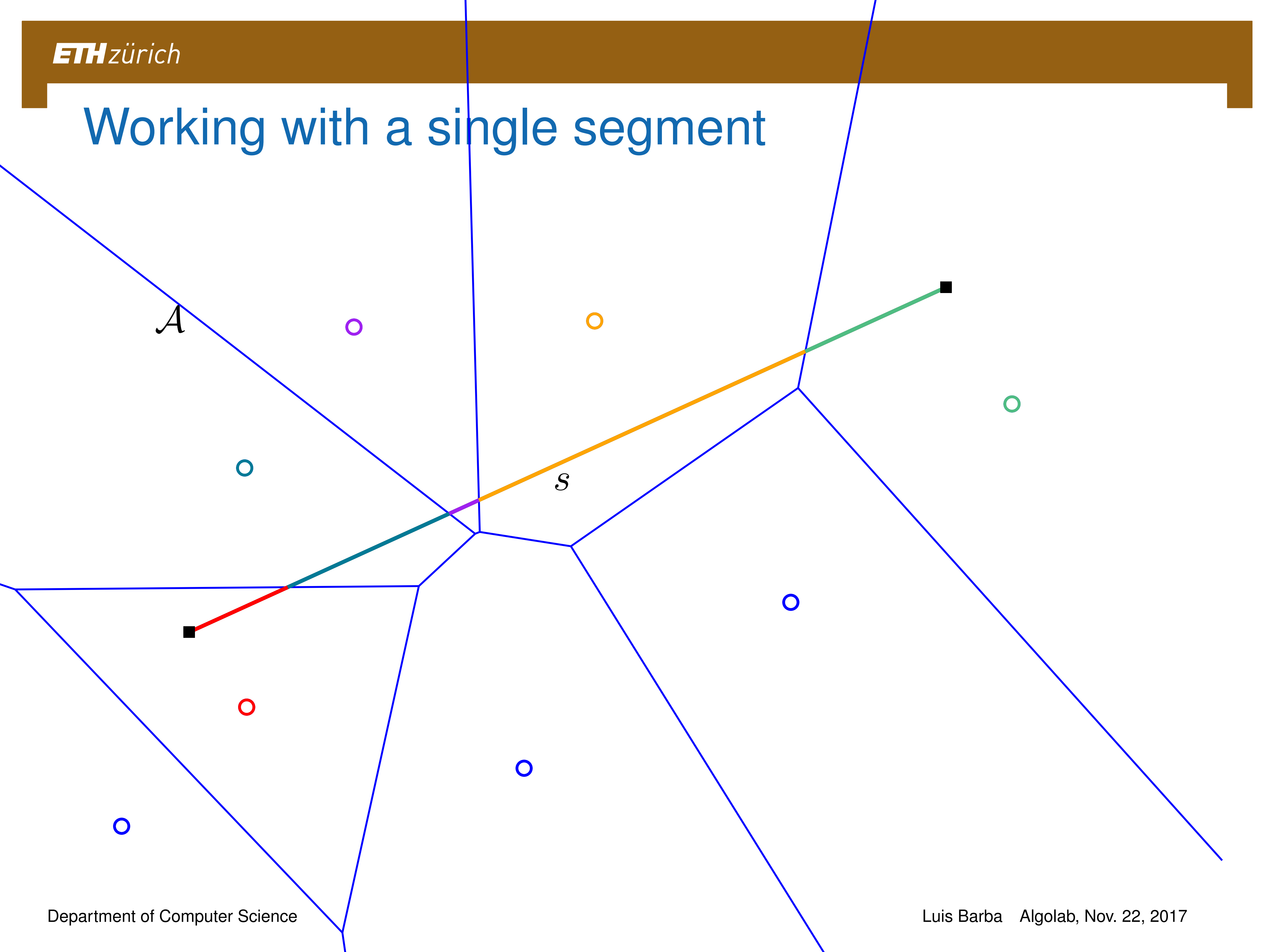
Working with a single segment

The minimum radius to cover x is given by the distance to its closest neighbor.



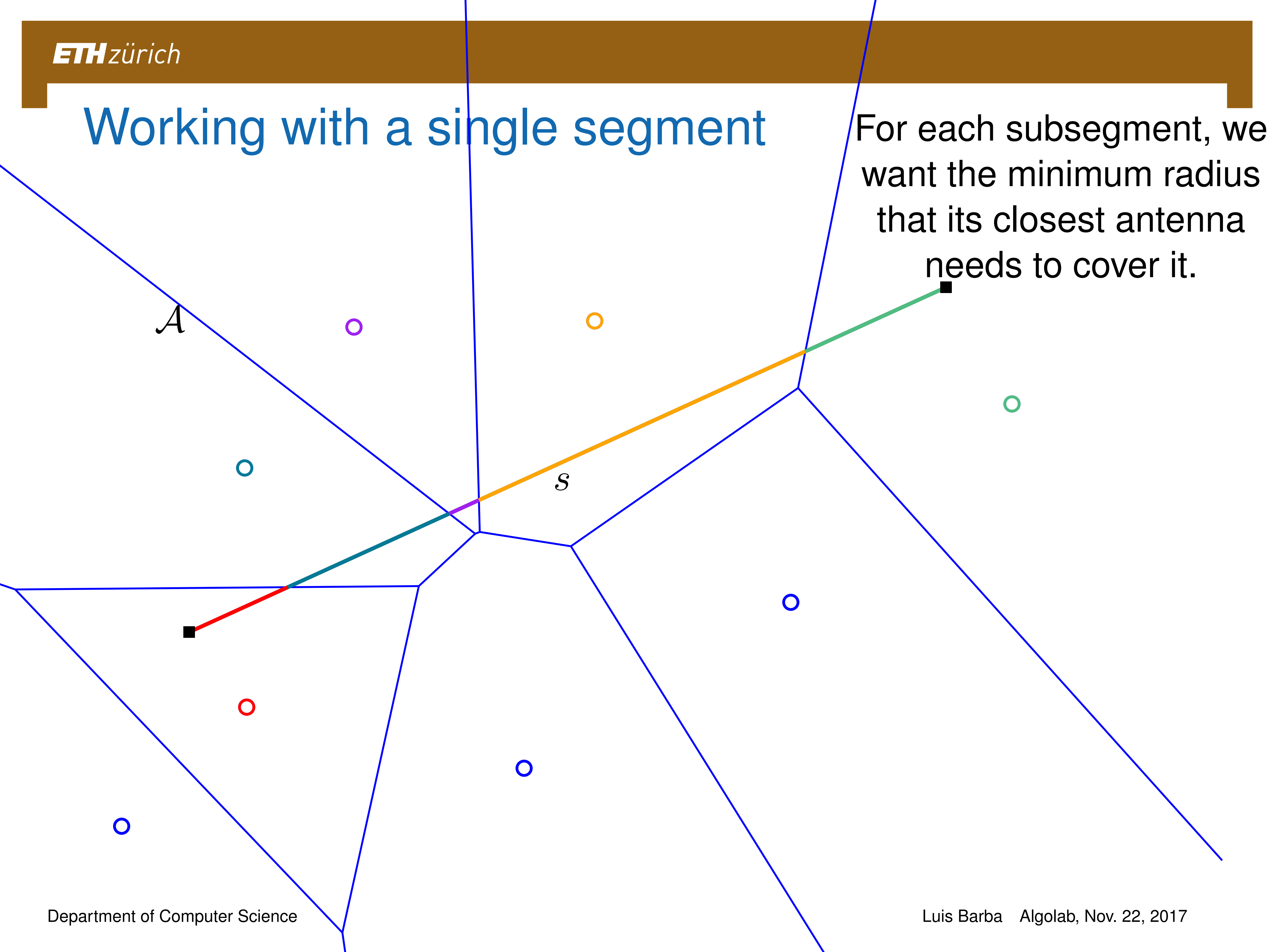
Which Antenna will cover x first?

Working with a single segment



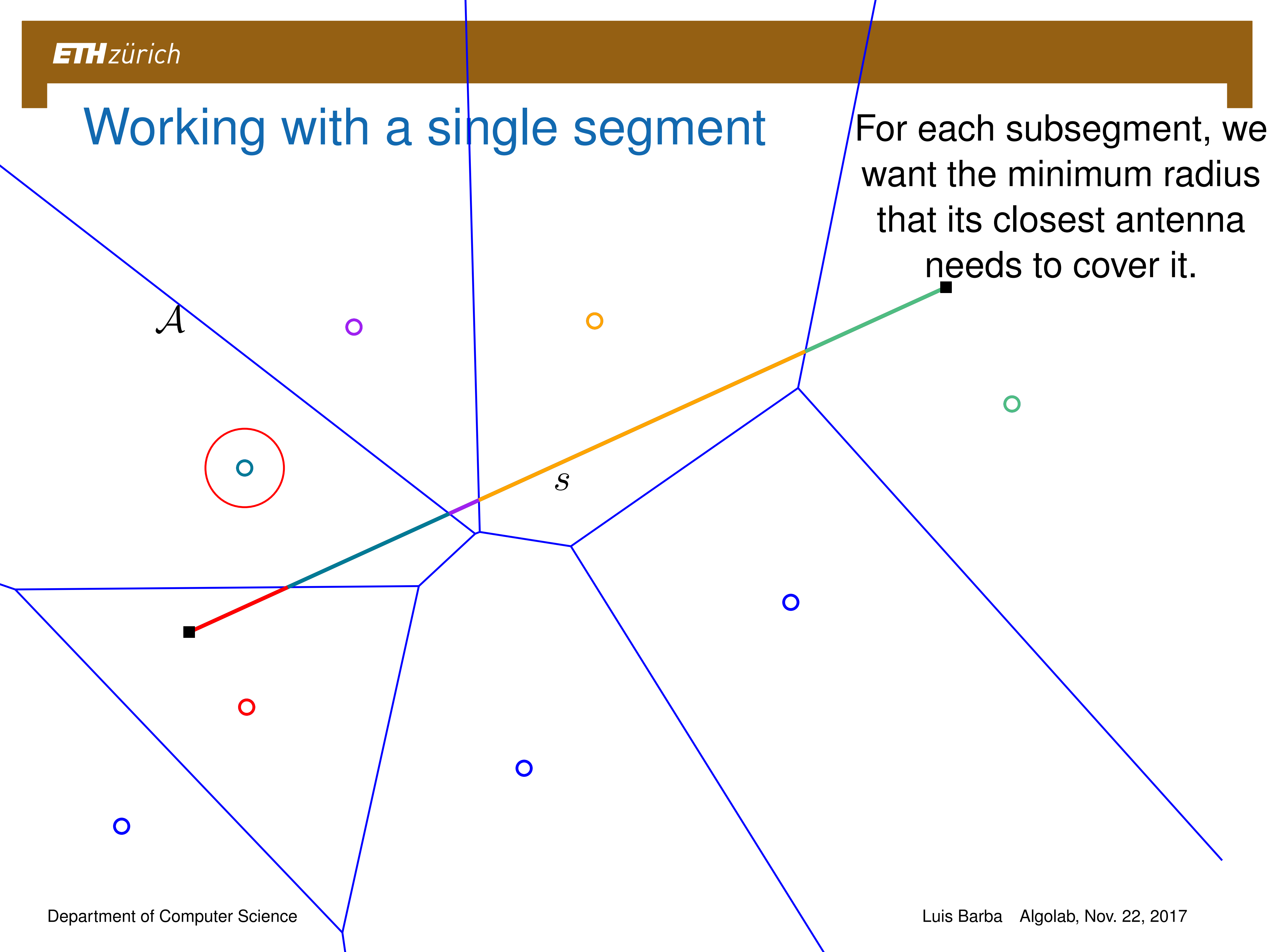
Working with a single segment

For each subsegment, we want the minimum radius that its closest antenna needs to cover it.



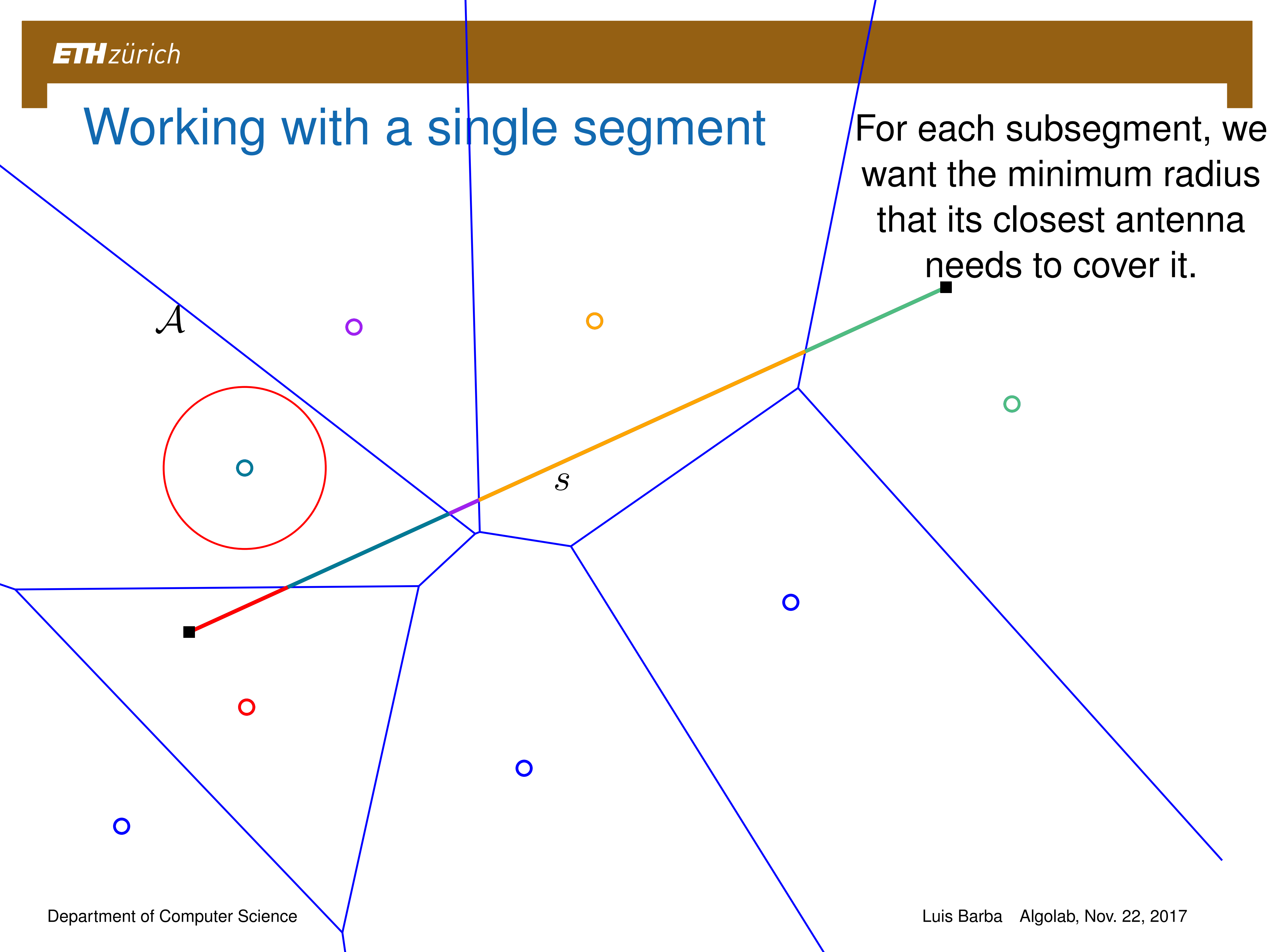
Working with a single segment

For each subsegment, we want the minimum radius that its closest antenna needs to cover it.



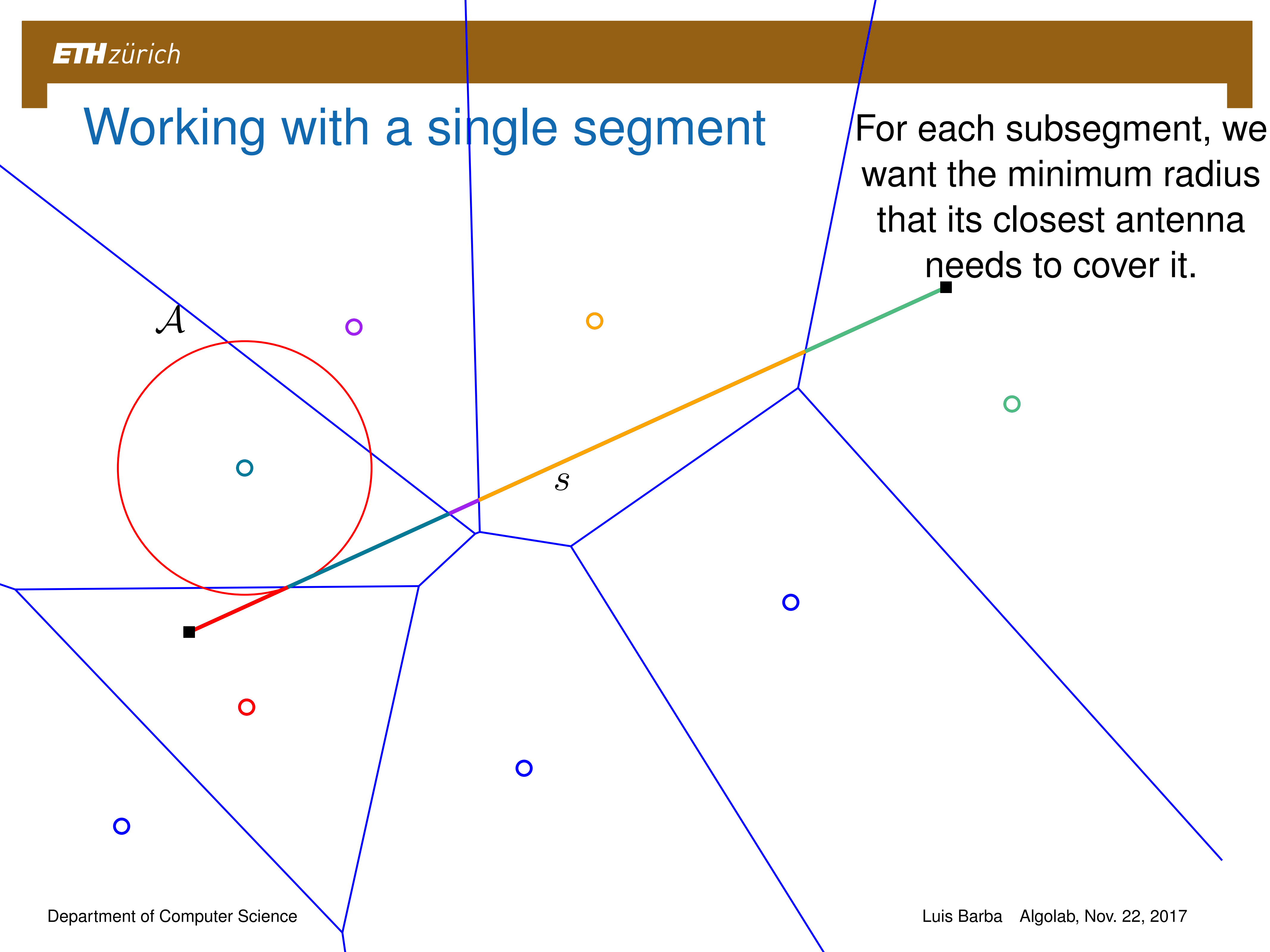
Working with a single segment

For each subsegment, we want the minimum radius that its closest antenna needs to cover it.



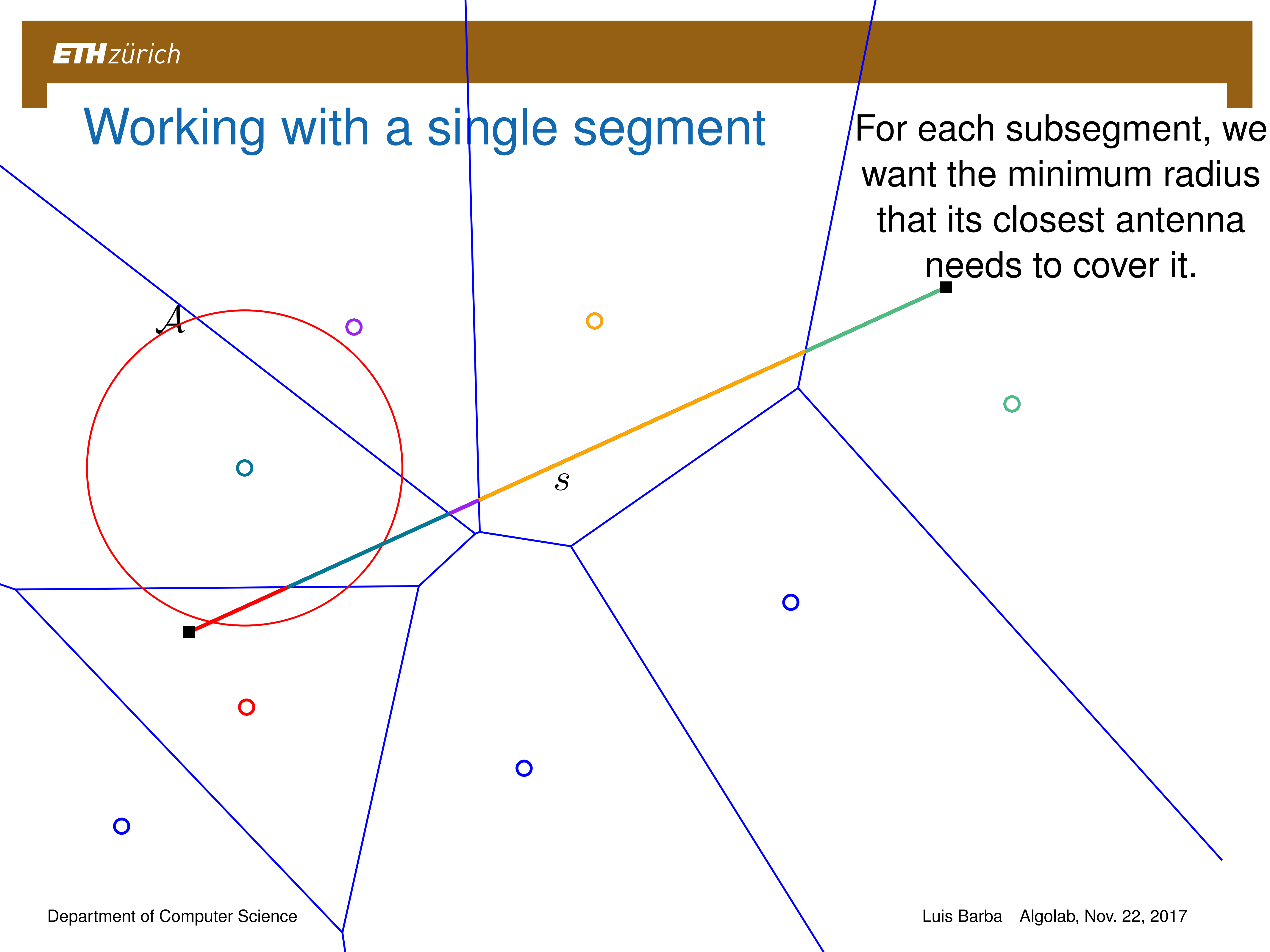
Working with a single segment

For each subsegment, we want the minimum radius that its closest antenna needs to cover it.



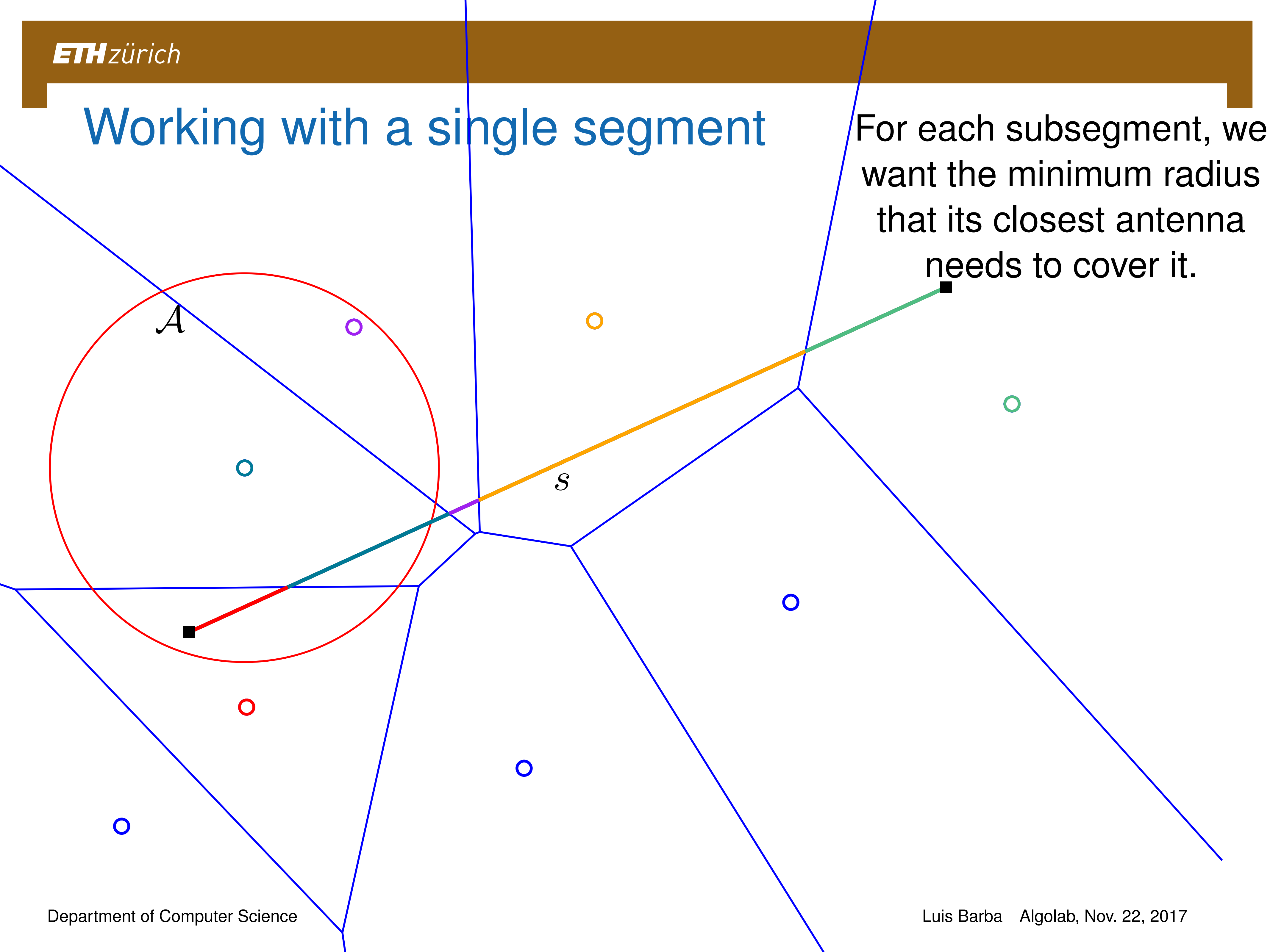
Working with a single segment

For each subsegment, we want the minimum radius that its closest antenna needs to cover it.



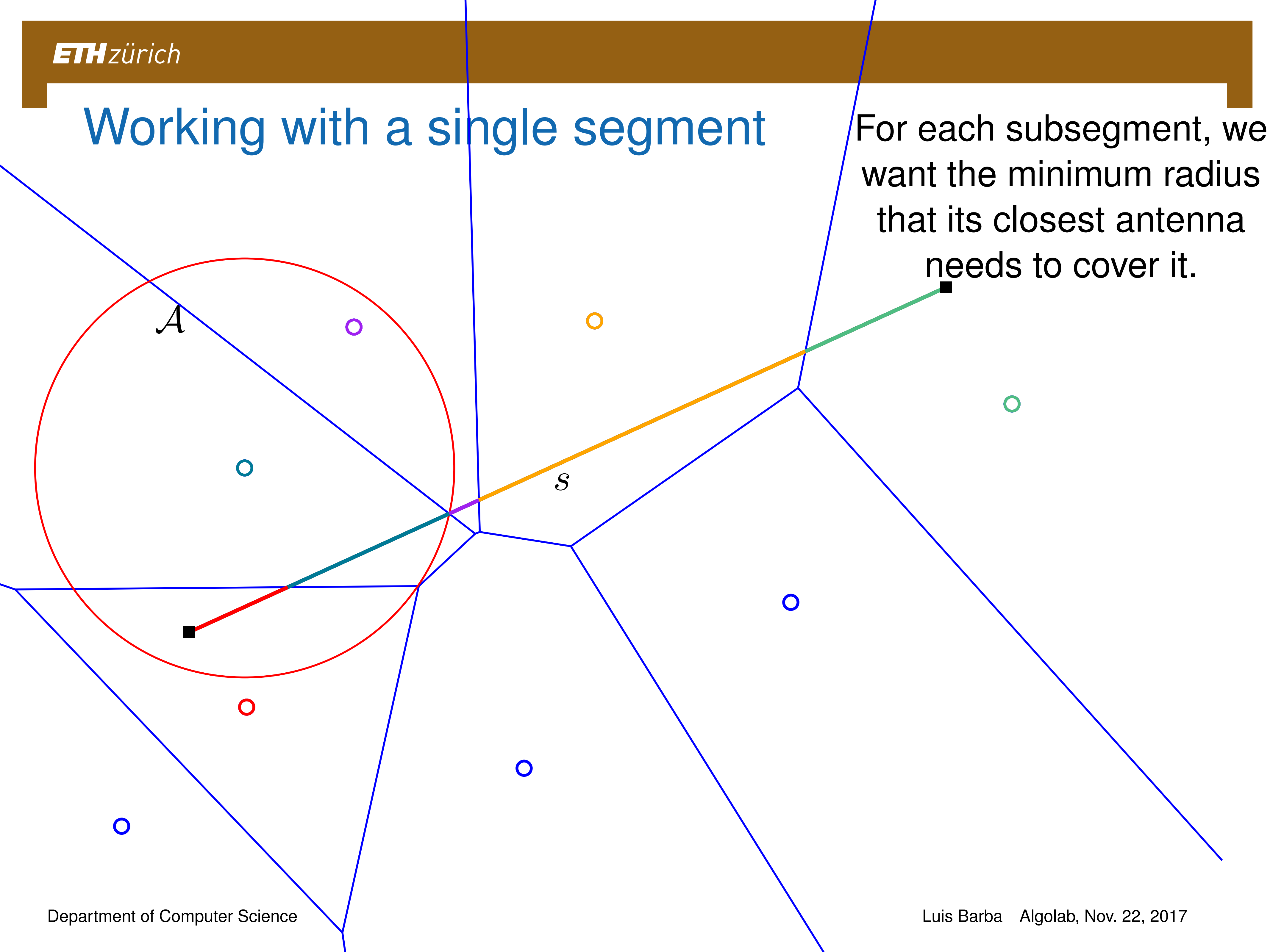
Working with a single segment

For each subsegment, we want the minimum radius that its closest antenna needs to cover it.



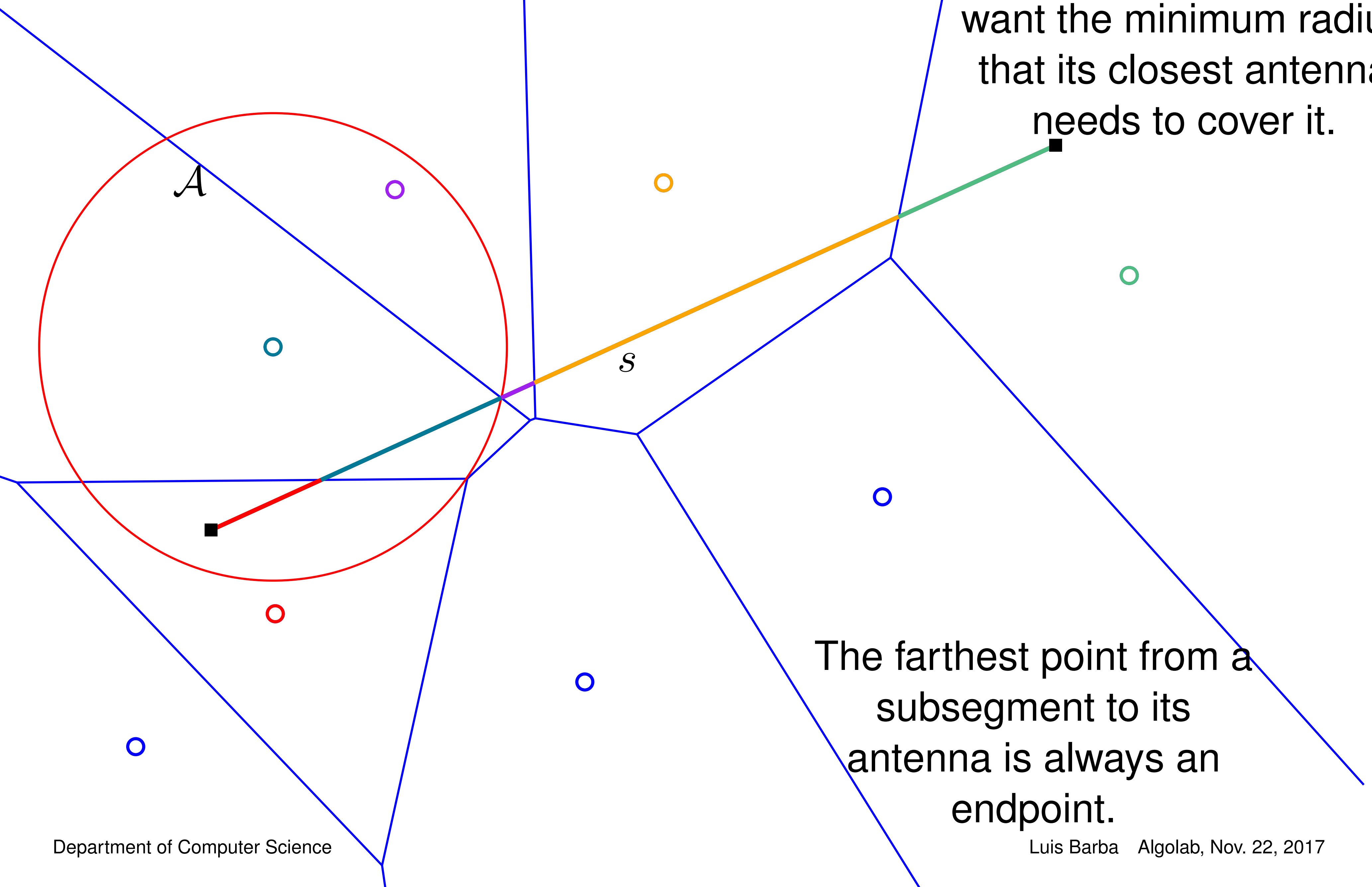
Working with a single segment

For each subsegment, we want the minimum radius that its closest antenna needs to cover it.



Working with a single segment

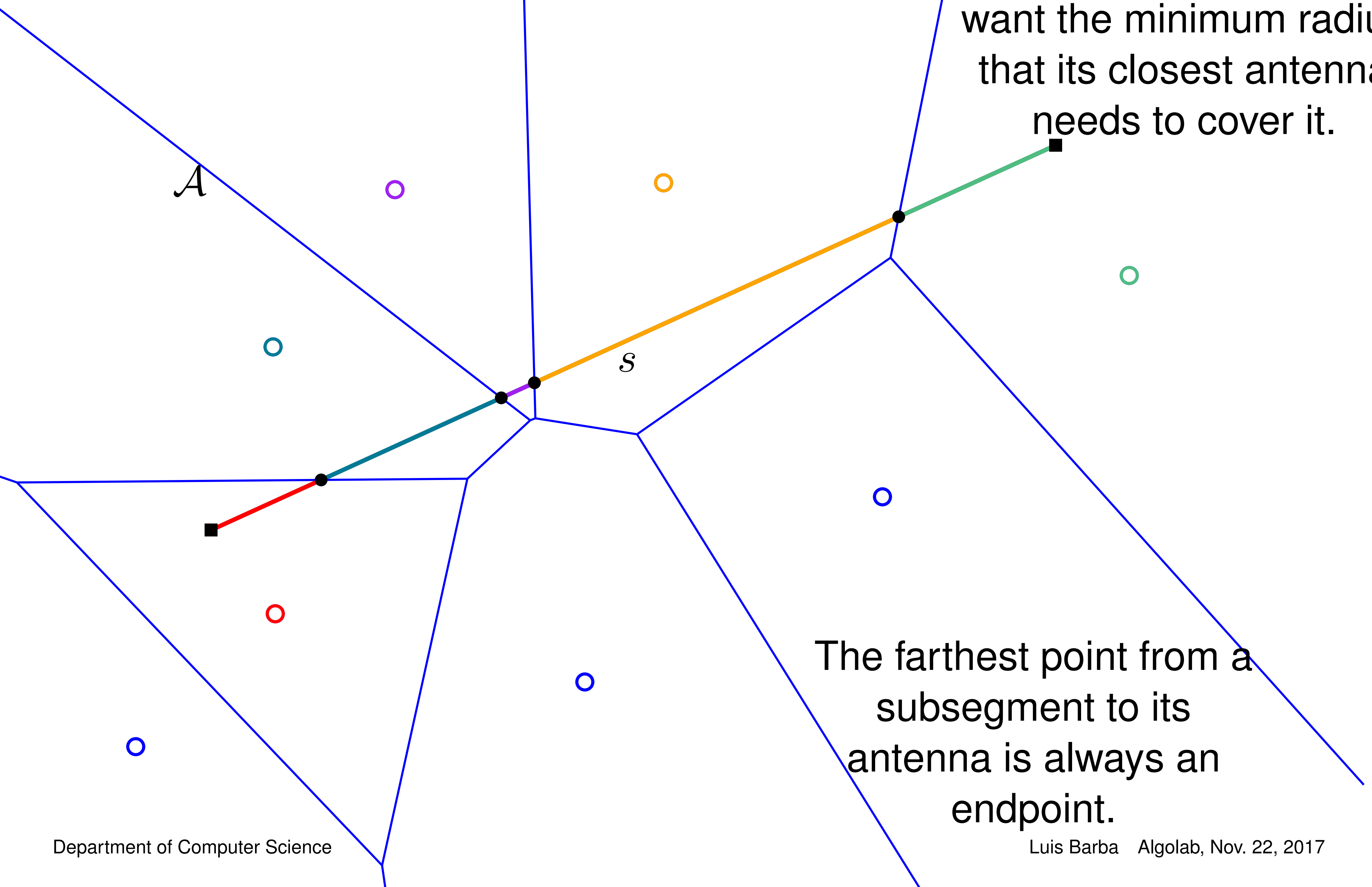
For each subsegment, we want the minimum radius that its closest antenna needs to cover it.



The farthest point from a subsegment to its antenna is always an endpoint.

Working with a single segment

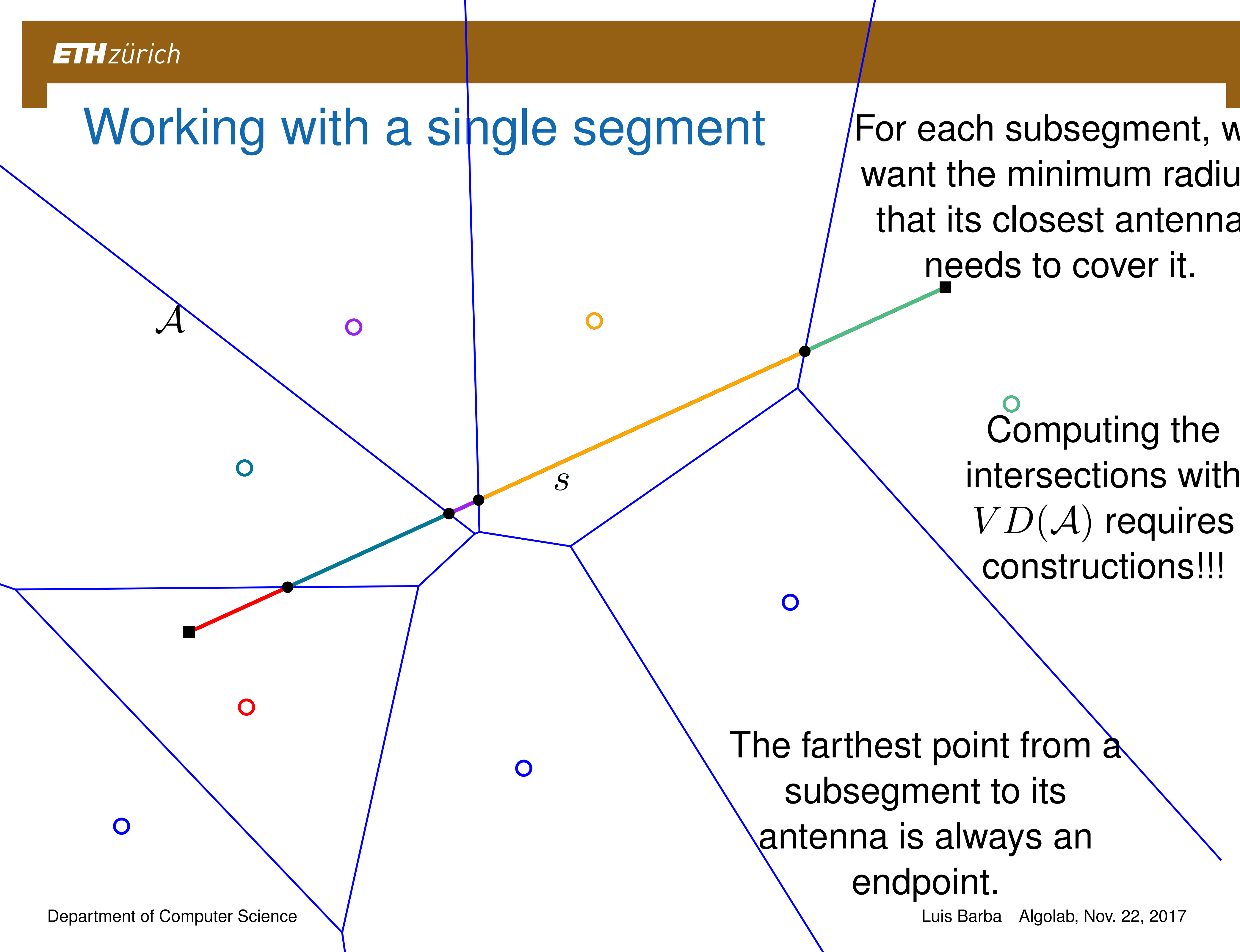
For each subsegment, we want the minimum radius that its closest antenna needs to cover it.



The farthest point from a subsegment to its antenna is always an endpoint.

Working with a single segment

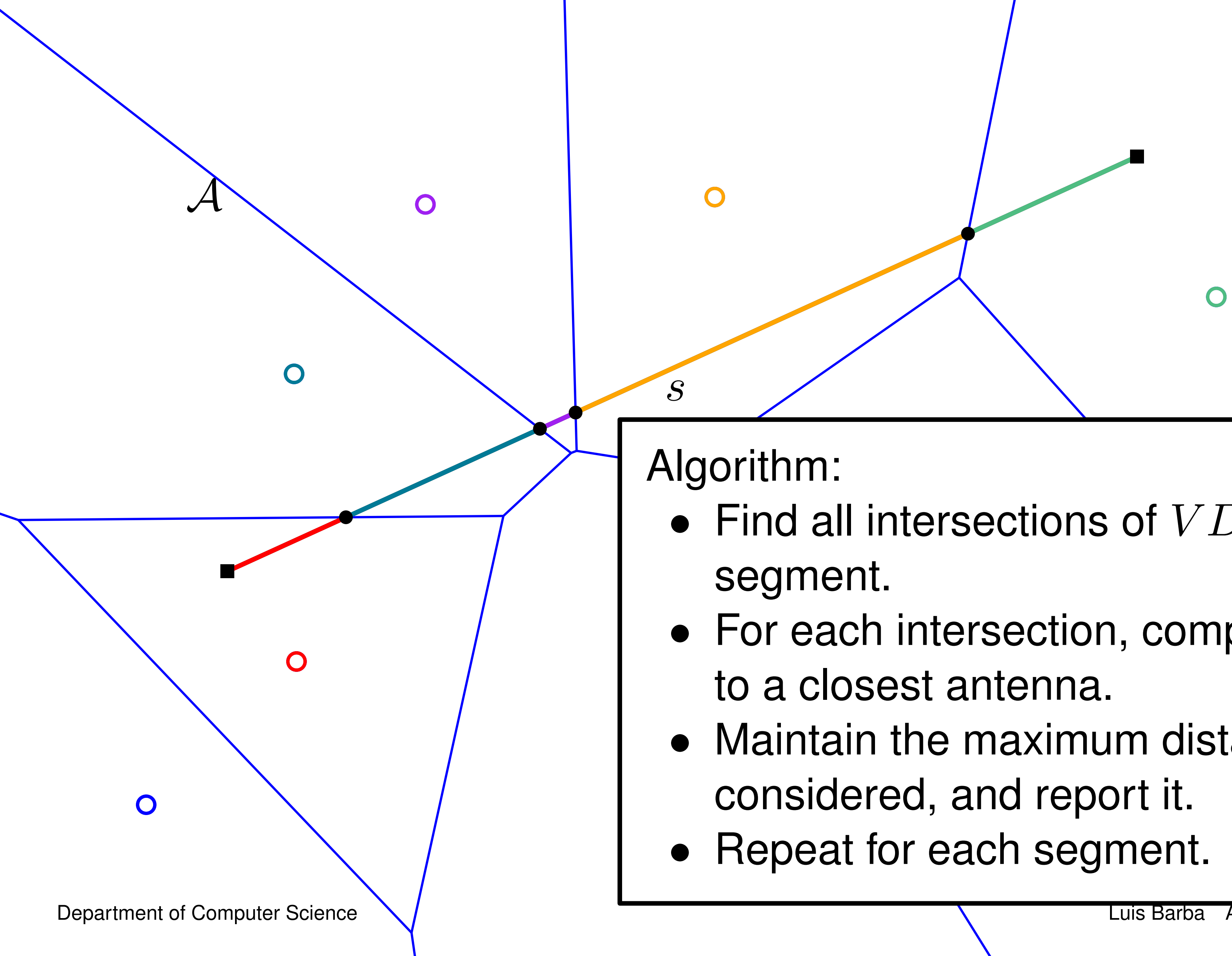
For each subsegment, we want the minimum radius that its closest antenna needs to cover it.



Computing the intersections with $VD(\mathcal{A})$ requires constructions!!!

The farthest point from a subsegment to its antenna is always an endpoint.

Working with a single segment

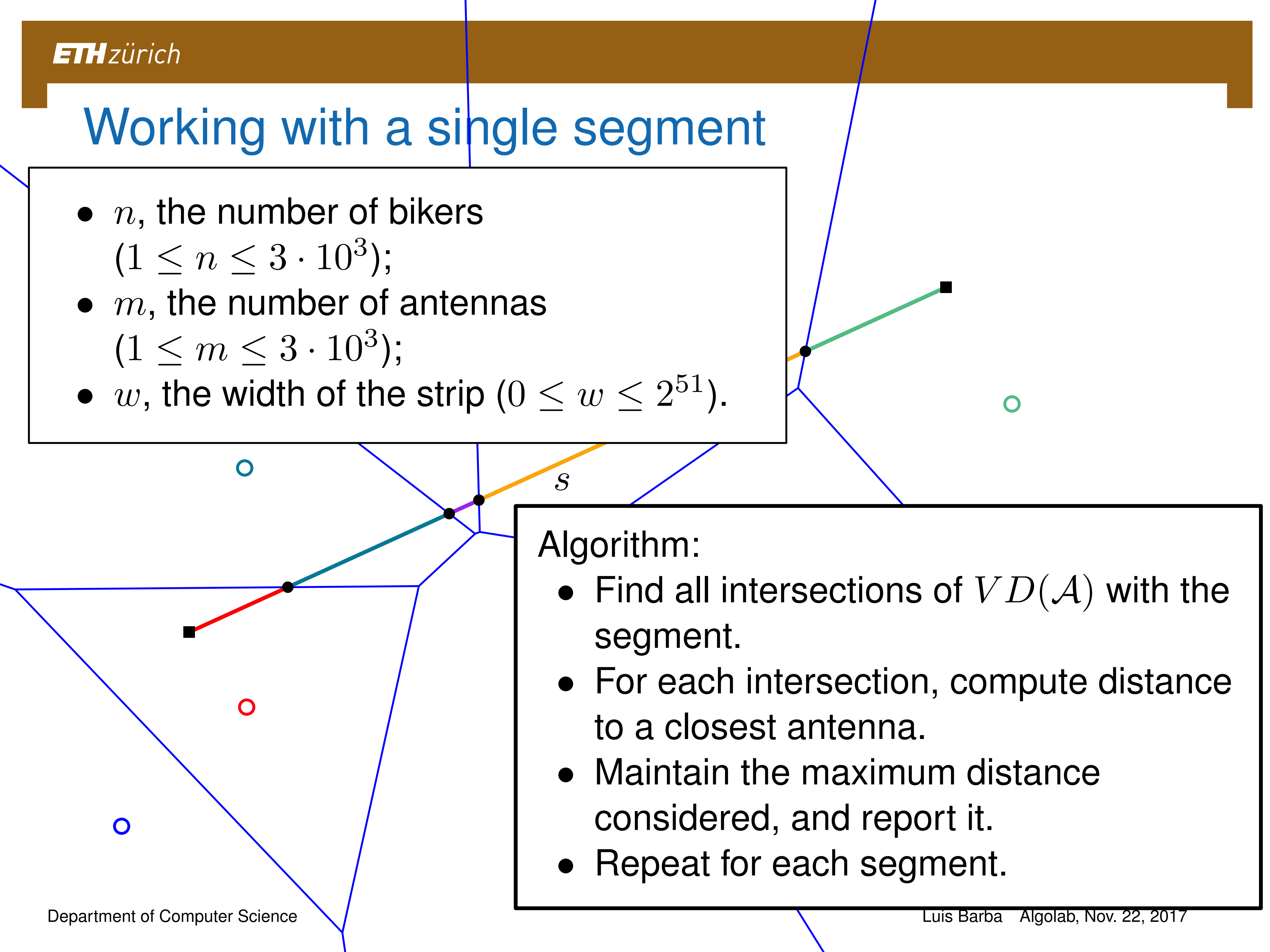


Algorithm:

- Find all intersections of $VD(\mathcal{A})$ with the segment.
- For each intersection, compute distance to a closest antenna.
- Maintain the maximum distance considered, and report it.
- Repeat for each segment.

Working with a single segment

- n, the number of bikers $(1 \leq n \leq 3 \cdot 10^3)$;
- m, the number of antennas $(1 \leq m \leq 3 \cdot 10^3)$;
- w, the width of the strip $(0 \leq w \leq 2^{51})$.



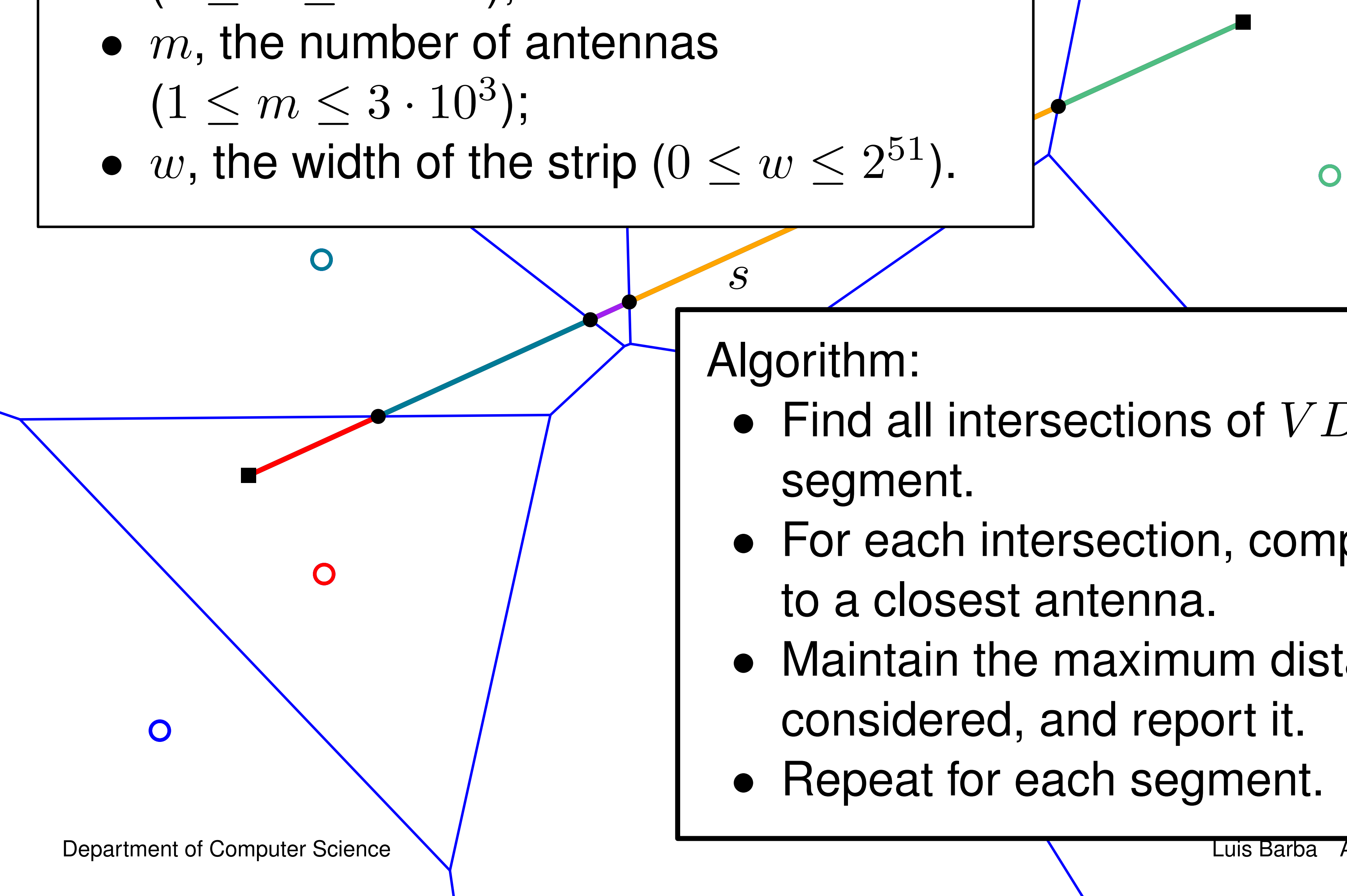
Algorithm:

- Find all intersections of $VD(\mathcal{A})$ with the segment.
- For each intersection, compute distance to a closest antenna.
- Maintain the maximum distance considered, and report it.
- Repeat for each segment.

Working with a single segment

- n, the number of bikers $(1 \leq n \leq 3 \cdot 10^3)$;
- m, the number of antennas $(1 \leq m \leq 3 \cdot 10^3)$;
- w, the width of the strip $(0 \leq w \leq 2^{51})$.

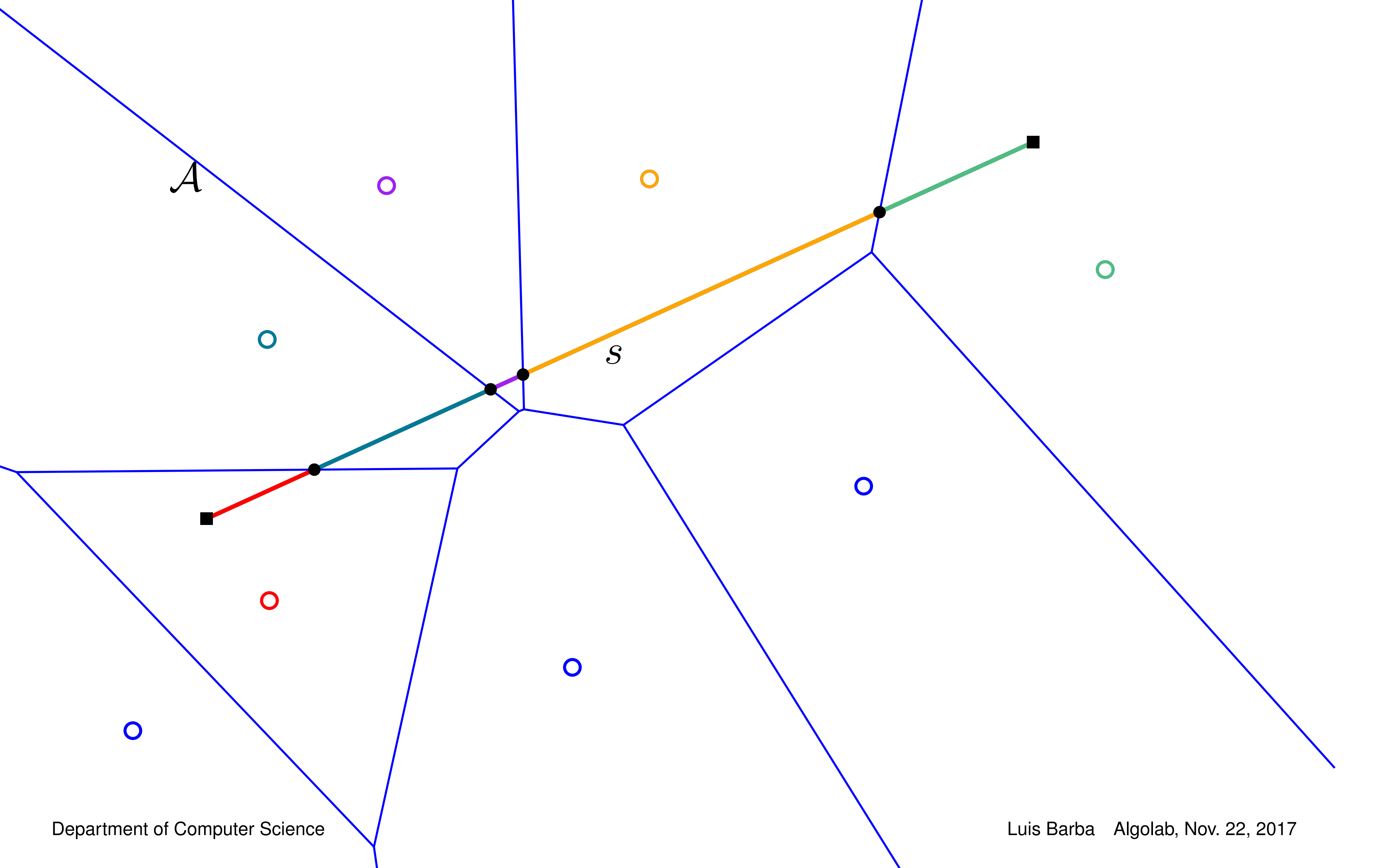
- $O(m \log m)$ time to Compute $VD(\mathcal{A})$.
- $O(m)$ time per segment.
- $O(nm)$ time in total.



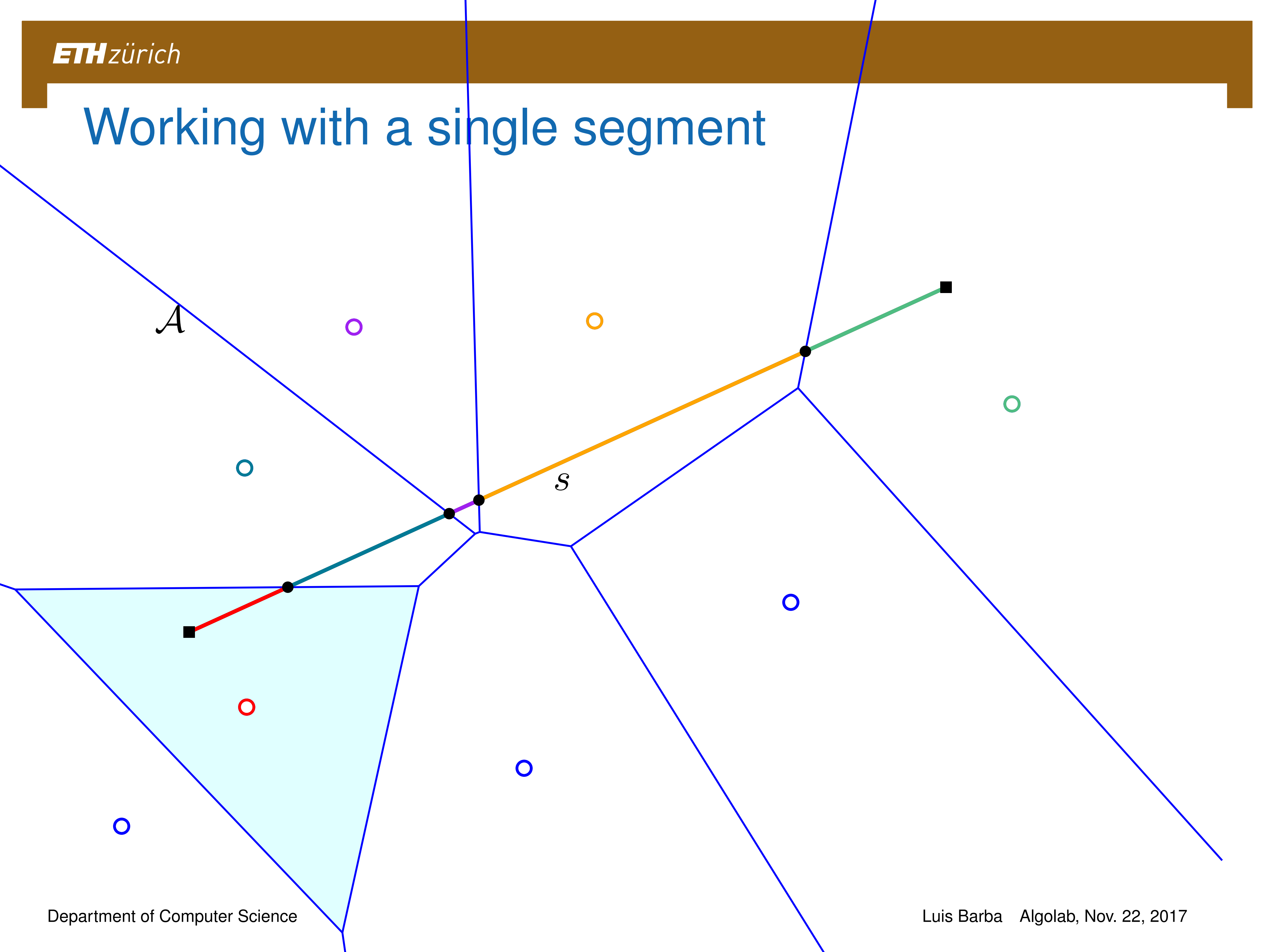
Algorithm:

- Find all intersections of $VD(\mathcal{A})$ with the segment.
- For each intersection, compute distance to a closest antenna.
- Maintain the maximum distance considered, and report it.
- Repeat for each segment.

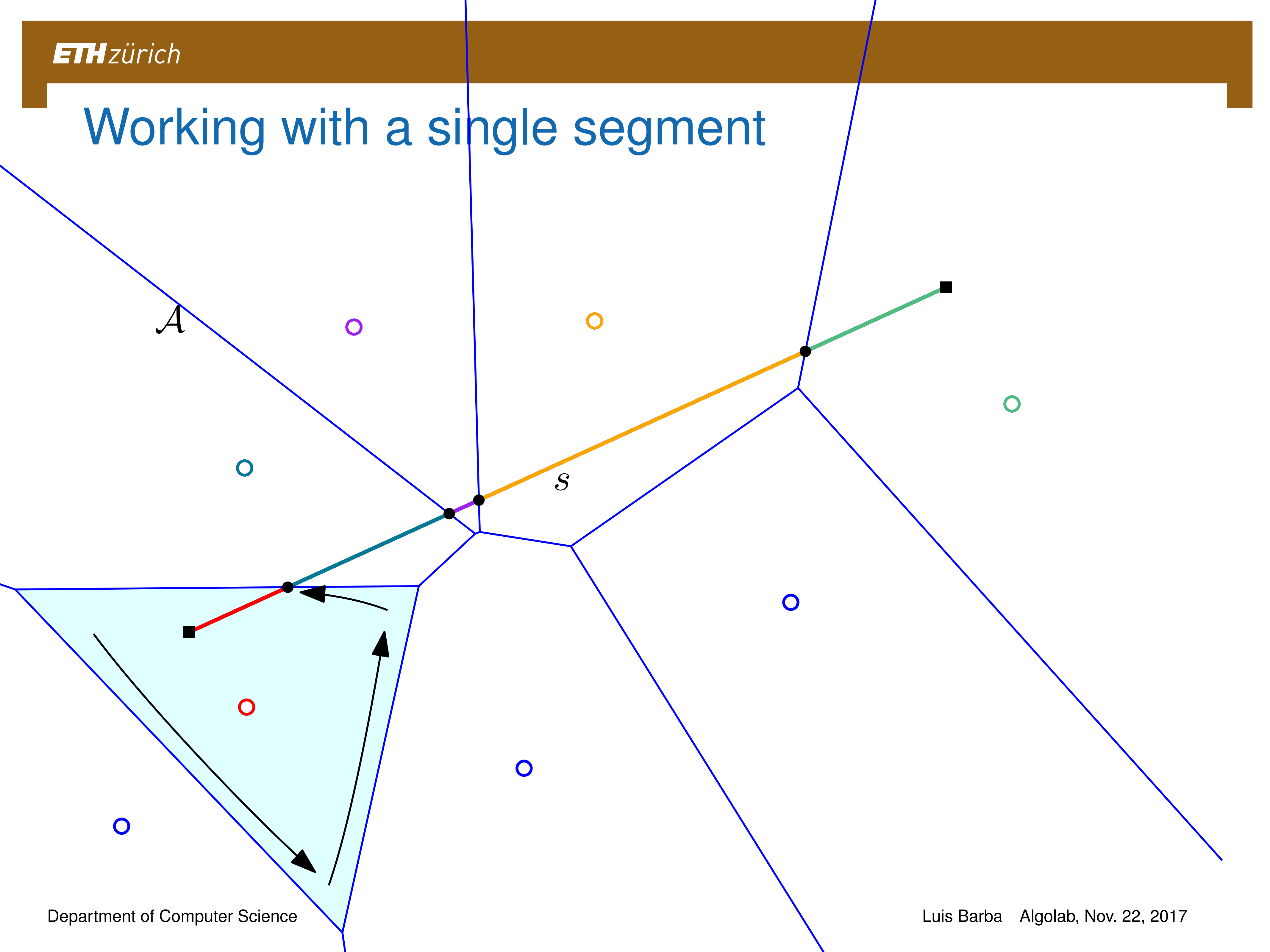
Working with a single segment



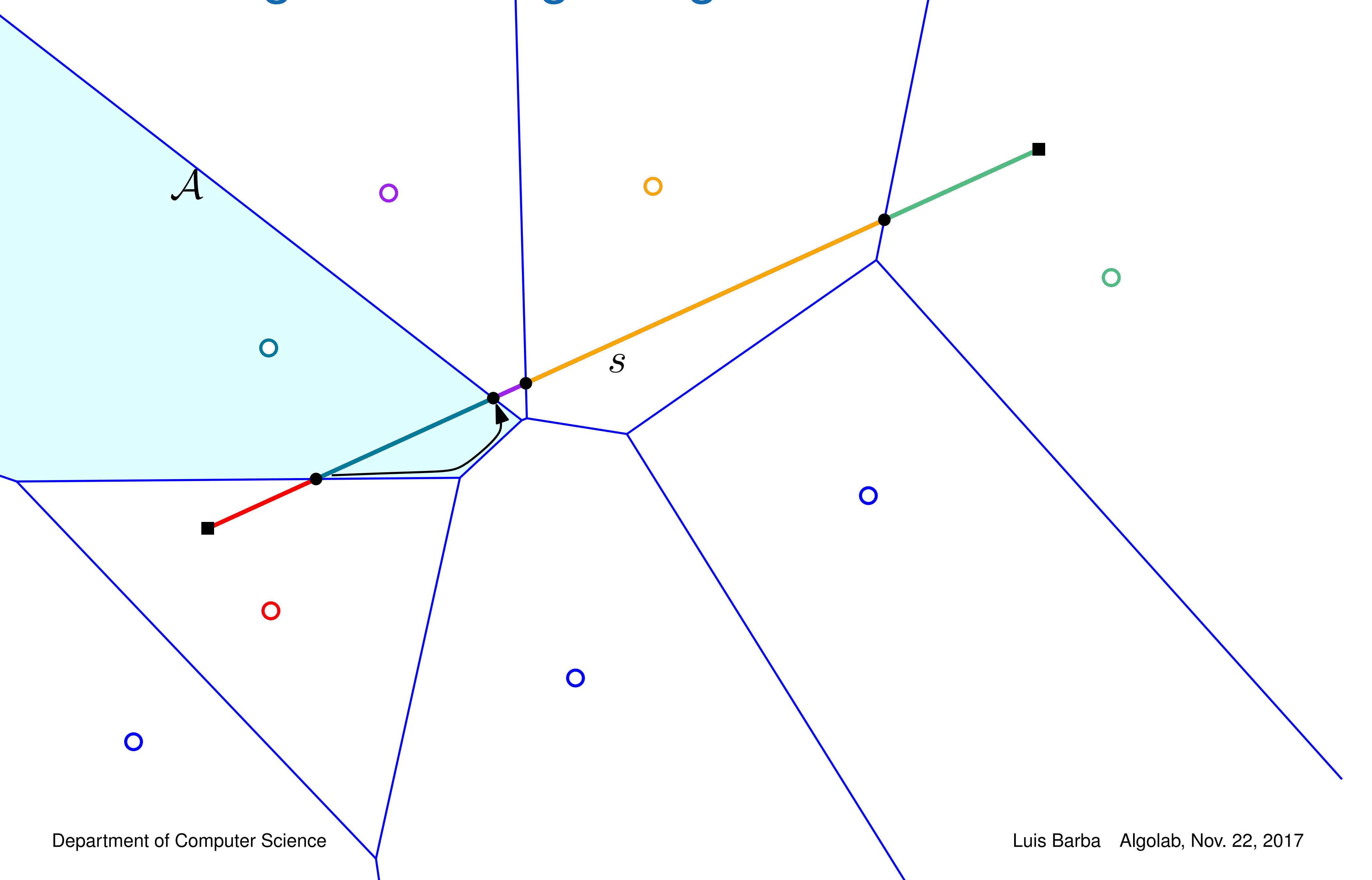
Working with a single segment



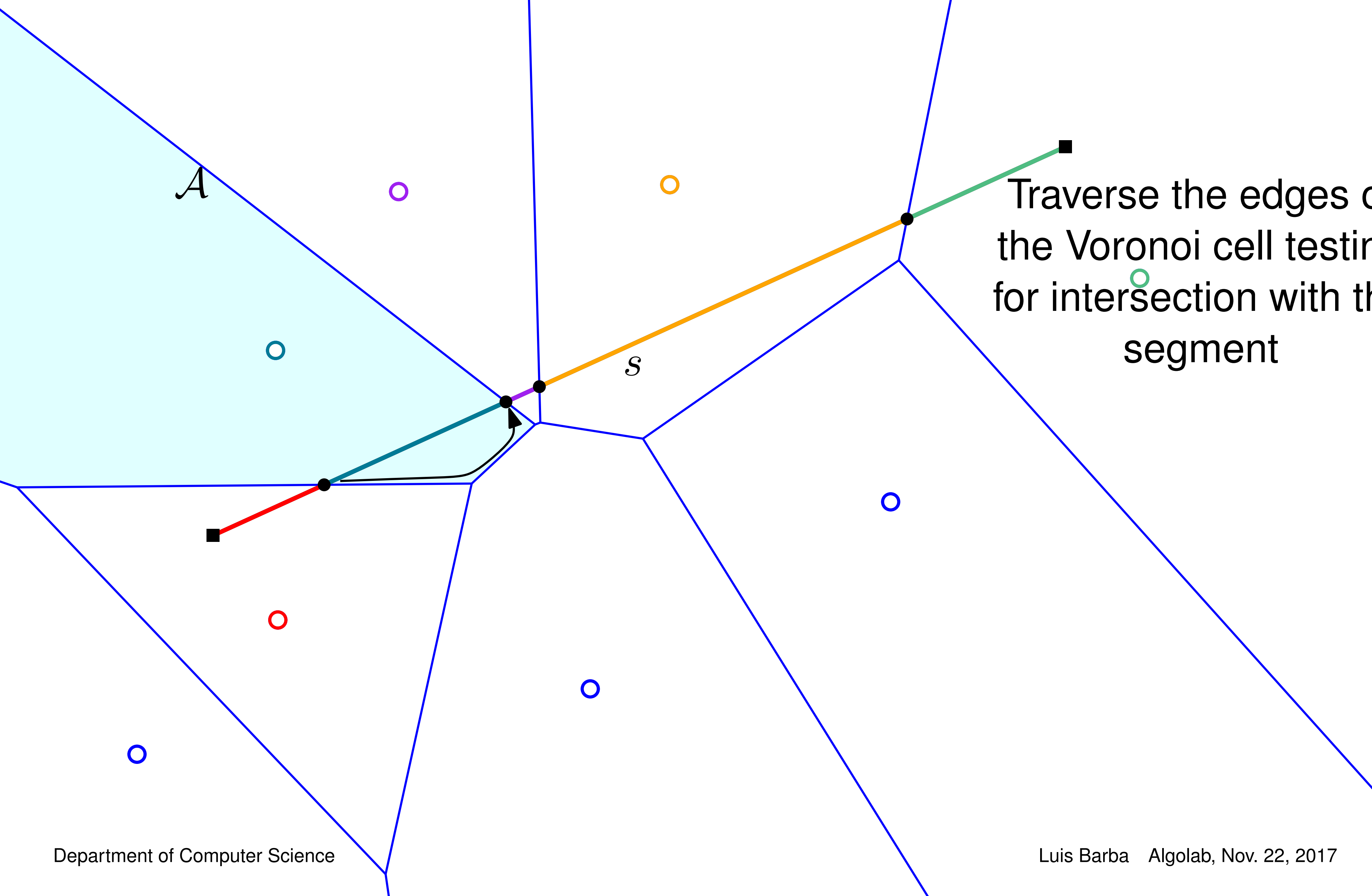
Working with a single segment



Working with a single segment

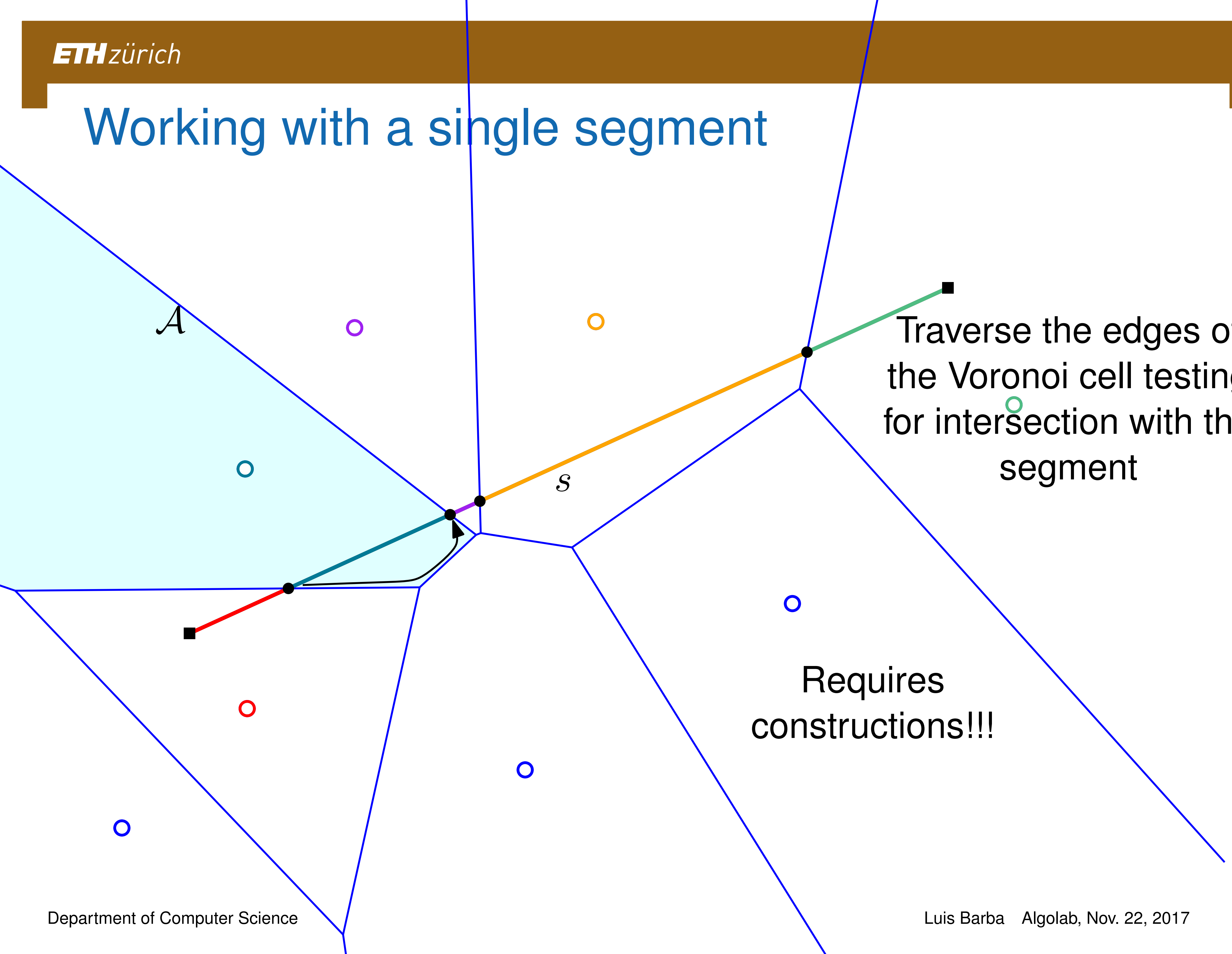


Working with a single segment



Traverse the edges of the Voronoi cell testing for intersection with the segment

Working with a single segment

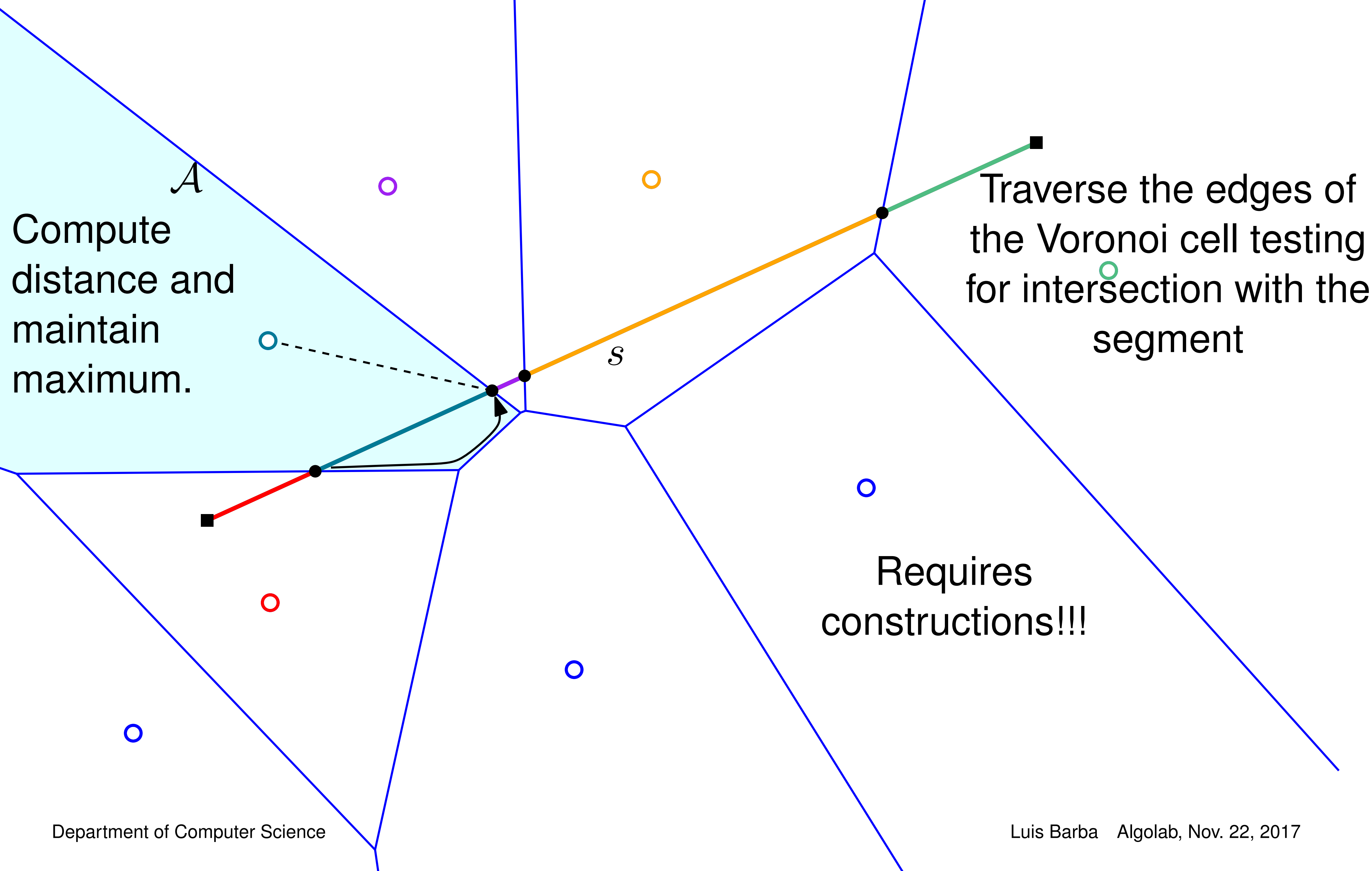


Traverse the edges of the Voronoi cell testing for intersection with the segment

Requires constructions!!!

ETH zürich

Working with a single segment



Compute distance and maintain maximum.

Traverse the edges of the Voronoi cell testing for intersection with the segment

Requires constructions!!!

Working with a single segment