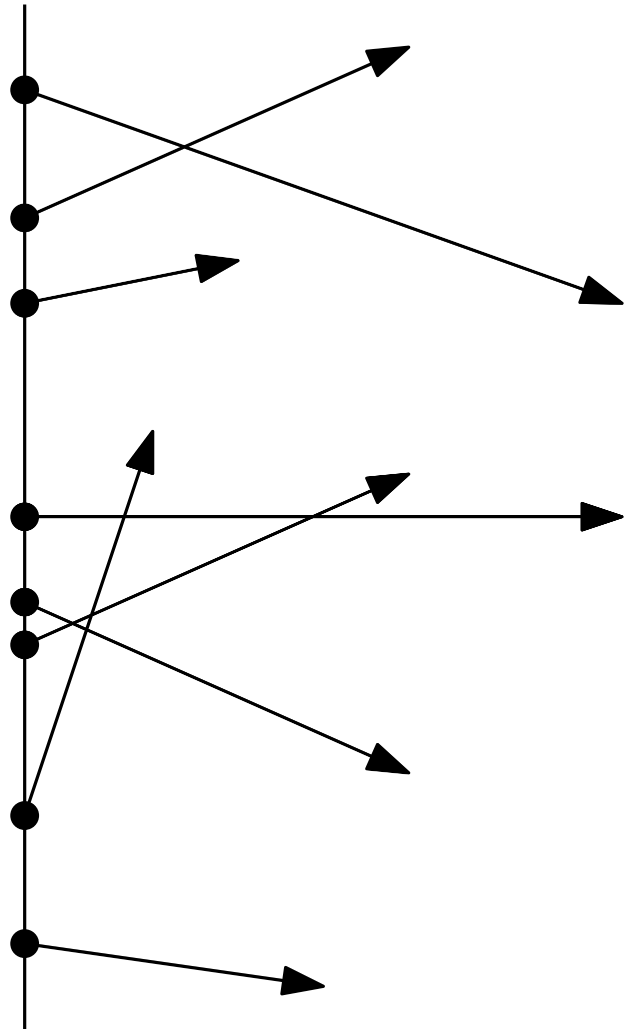
# Motorcycles

Bikers start at the same time at unit speed. If a biker runs into the tracks of another it stops.

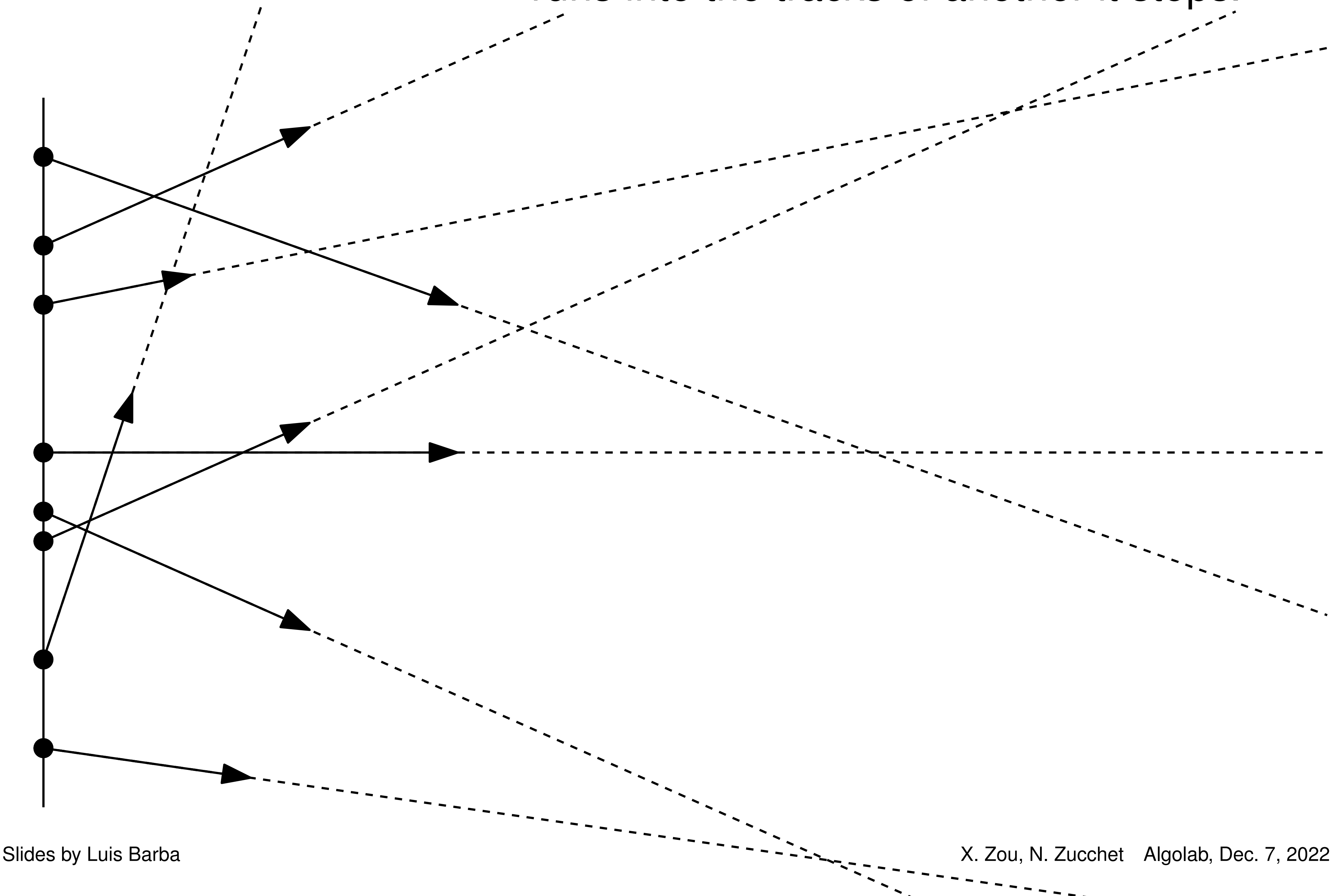
# Motorcycles

Bikers start at the same time at unit speed. If a biker runs into the tracks of another it stops.



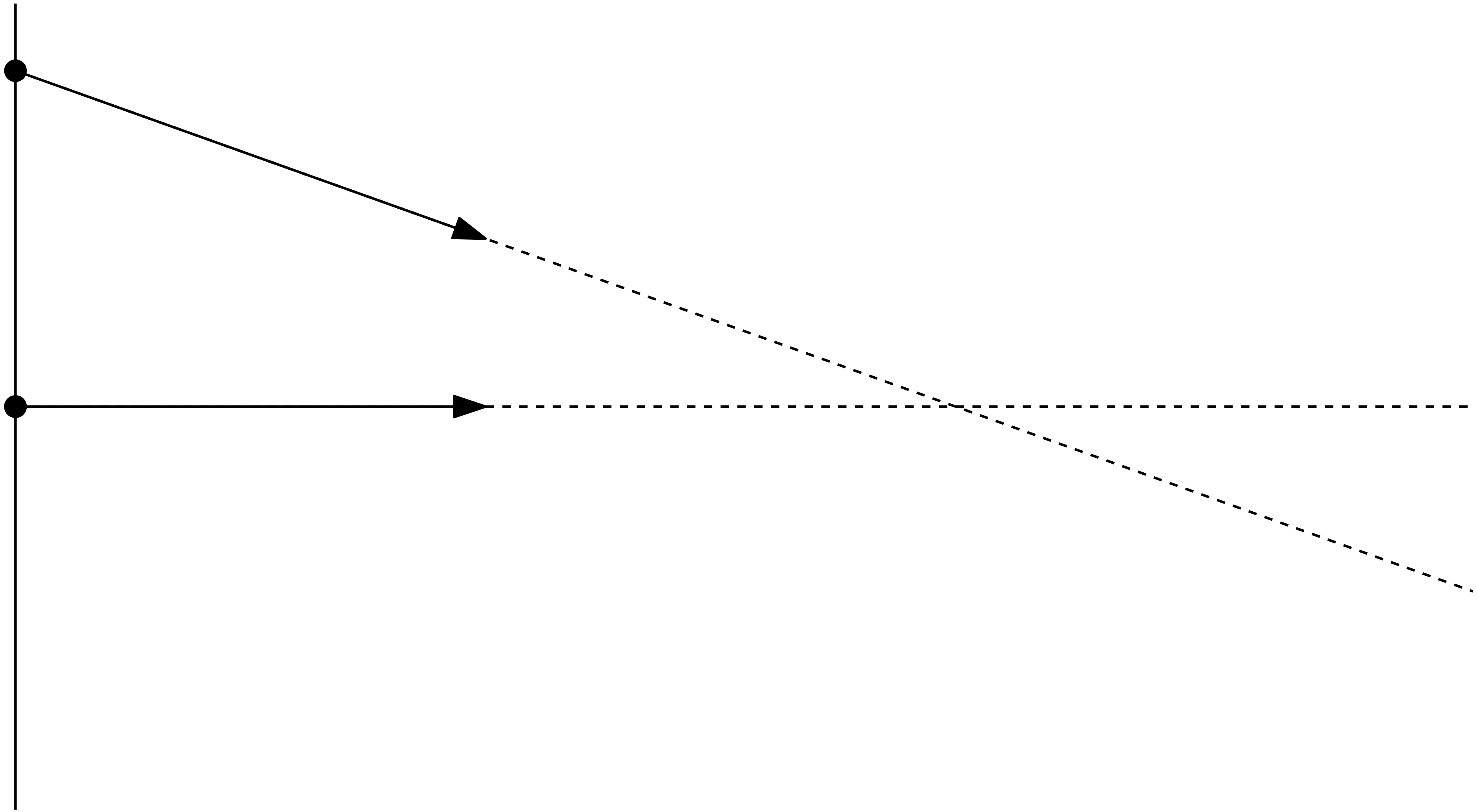
# Motorcycles

Bikers start at the same time at unit speed. If a biker runs into the tracks of another it stops.



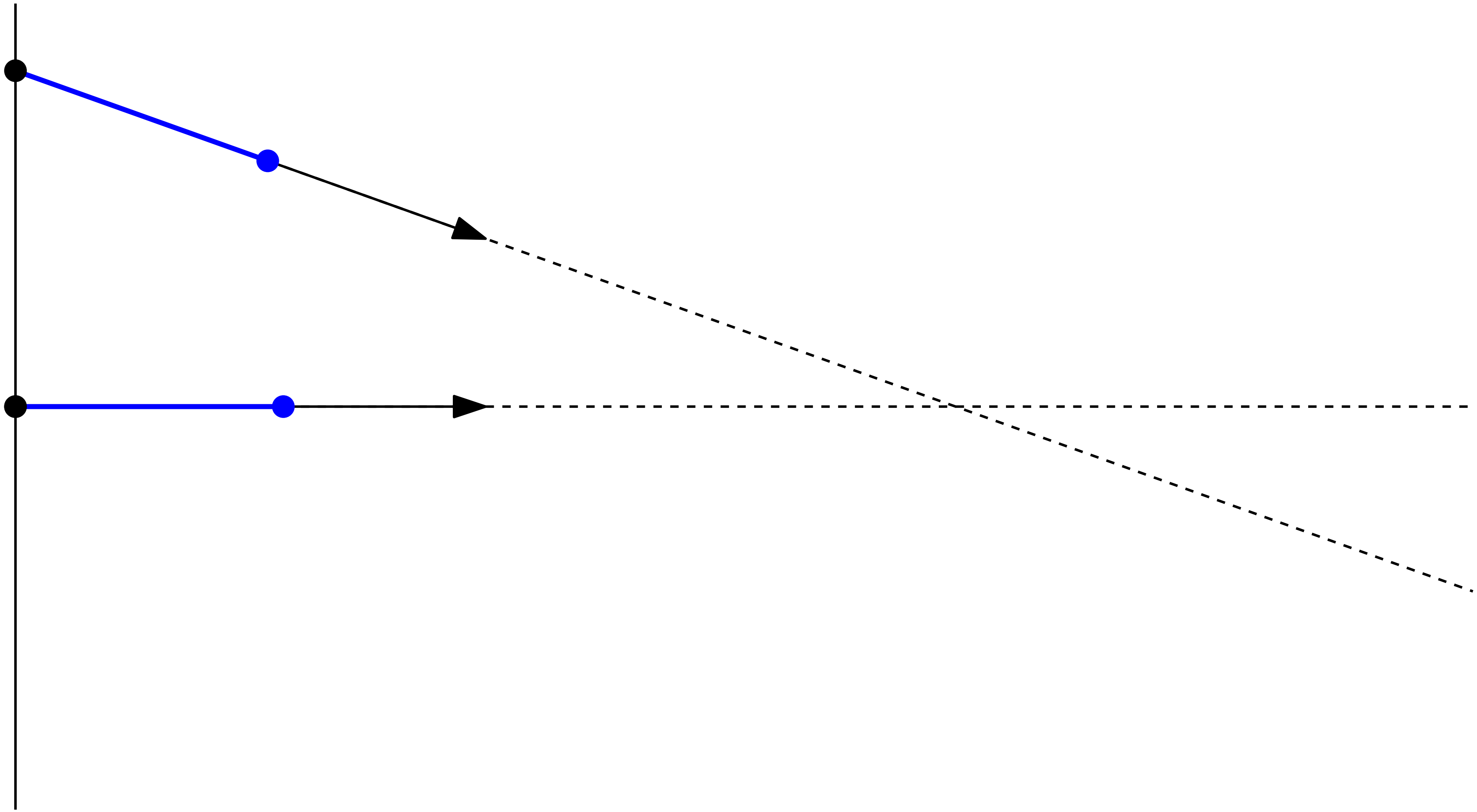
# Motorcycles

Bikers start at the same time at unit speed. If a biker runs into the tracks of another it stops.



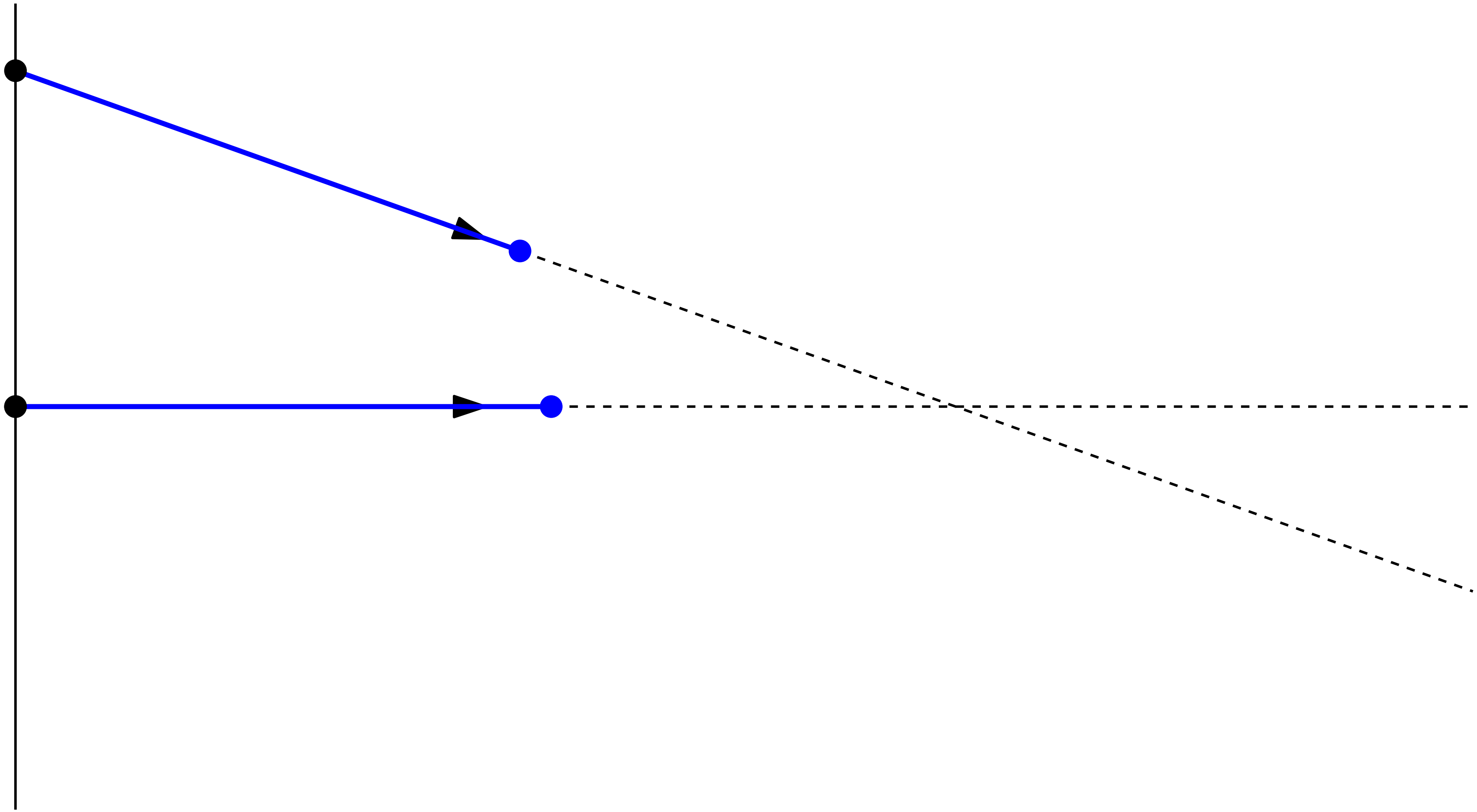
# Motorcycles

Bikers start at the same time at unit speed. If a biker runs into the tracks of another it stops.



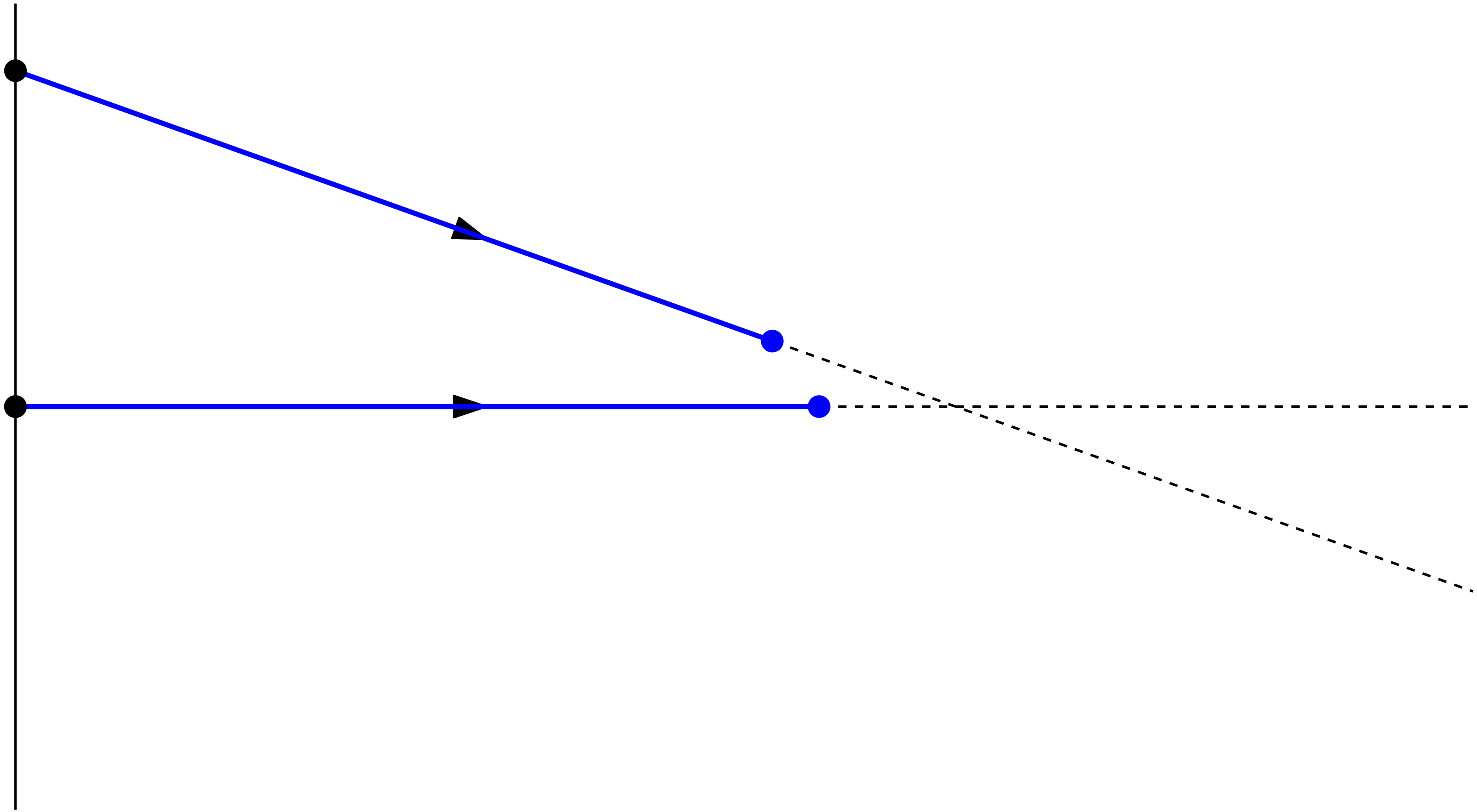
# Motorcycles

Bikers start at the same time at unit speed. If a biker runs into the tracks of another it stops.



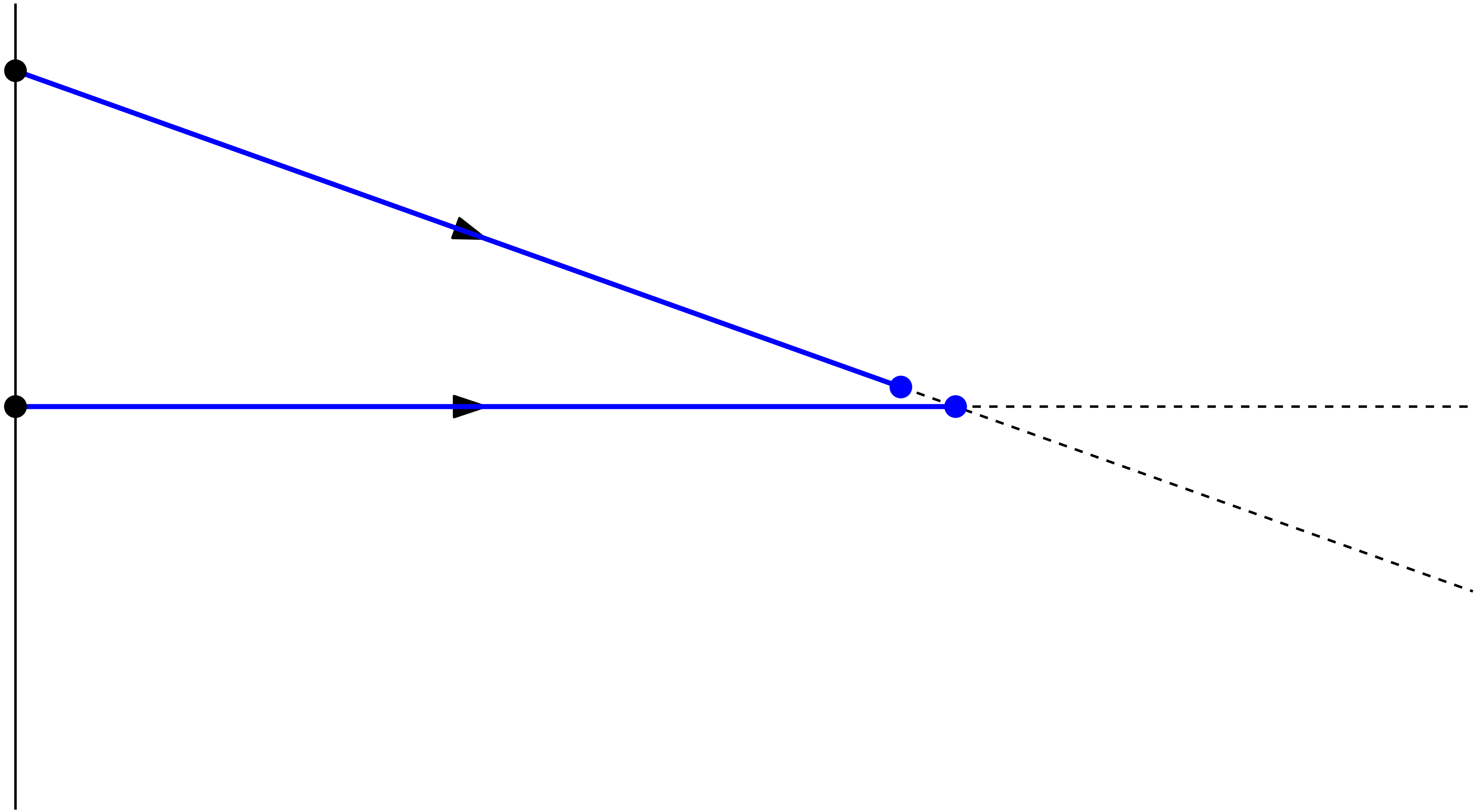
# Motorcycles

Bikers start at the same time at unit speed. If a biker runs into the tracks of another it stops.



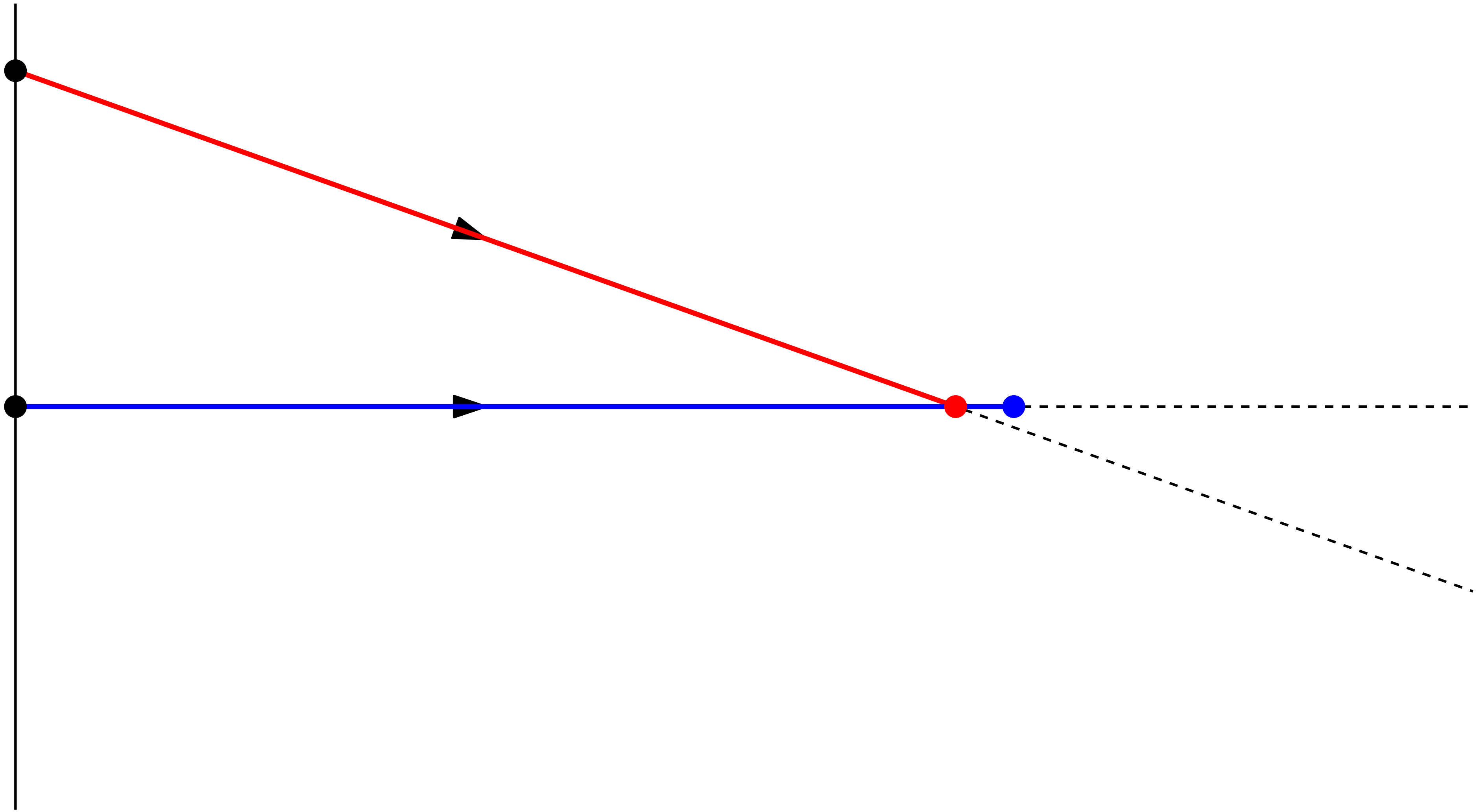
# Motorcycles

Bikers start at the same time at unit speed. If a biker runs into the tracks of another it stops.



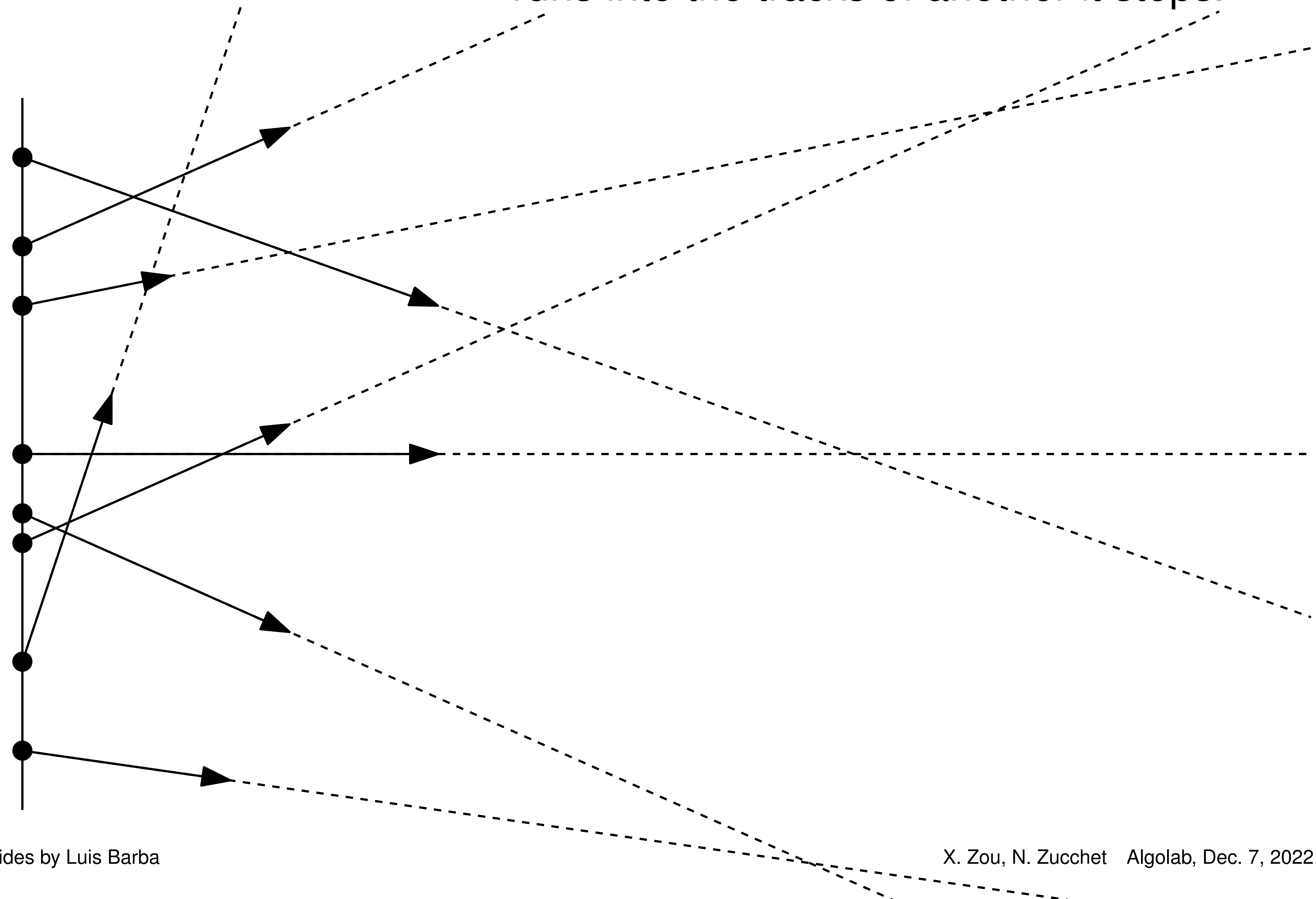
# Motorcycles

Bikers start at the same time at unit speed. If a biker runs into the tracks of another it stops.



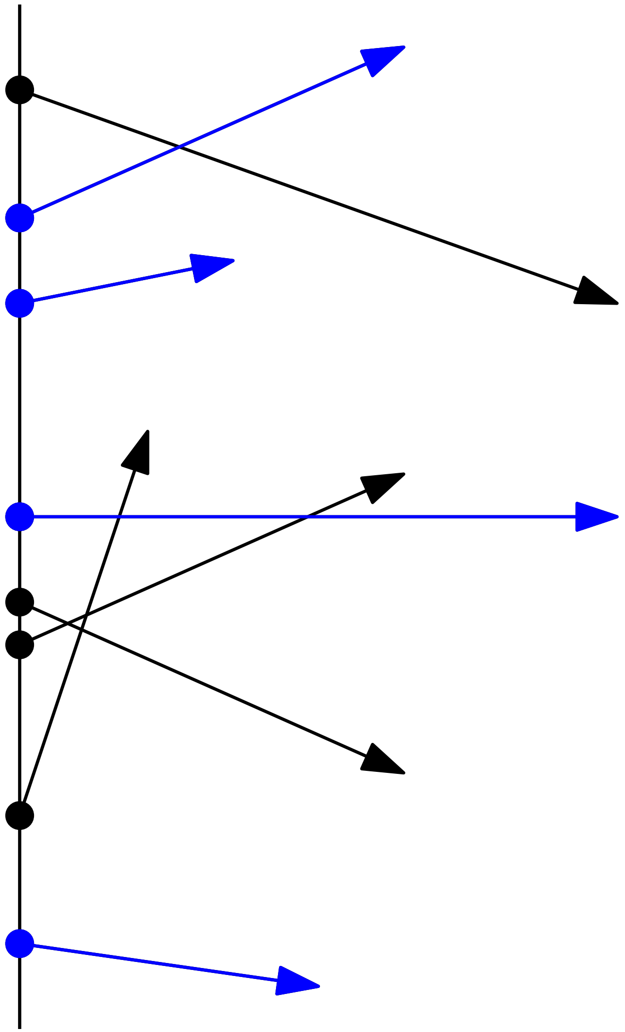
# Motorcycles

Bikers start at the same time at unit speed. If a biker runs into the tracks of another it stops.



# Motorcycles

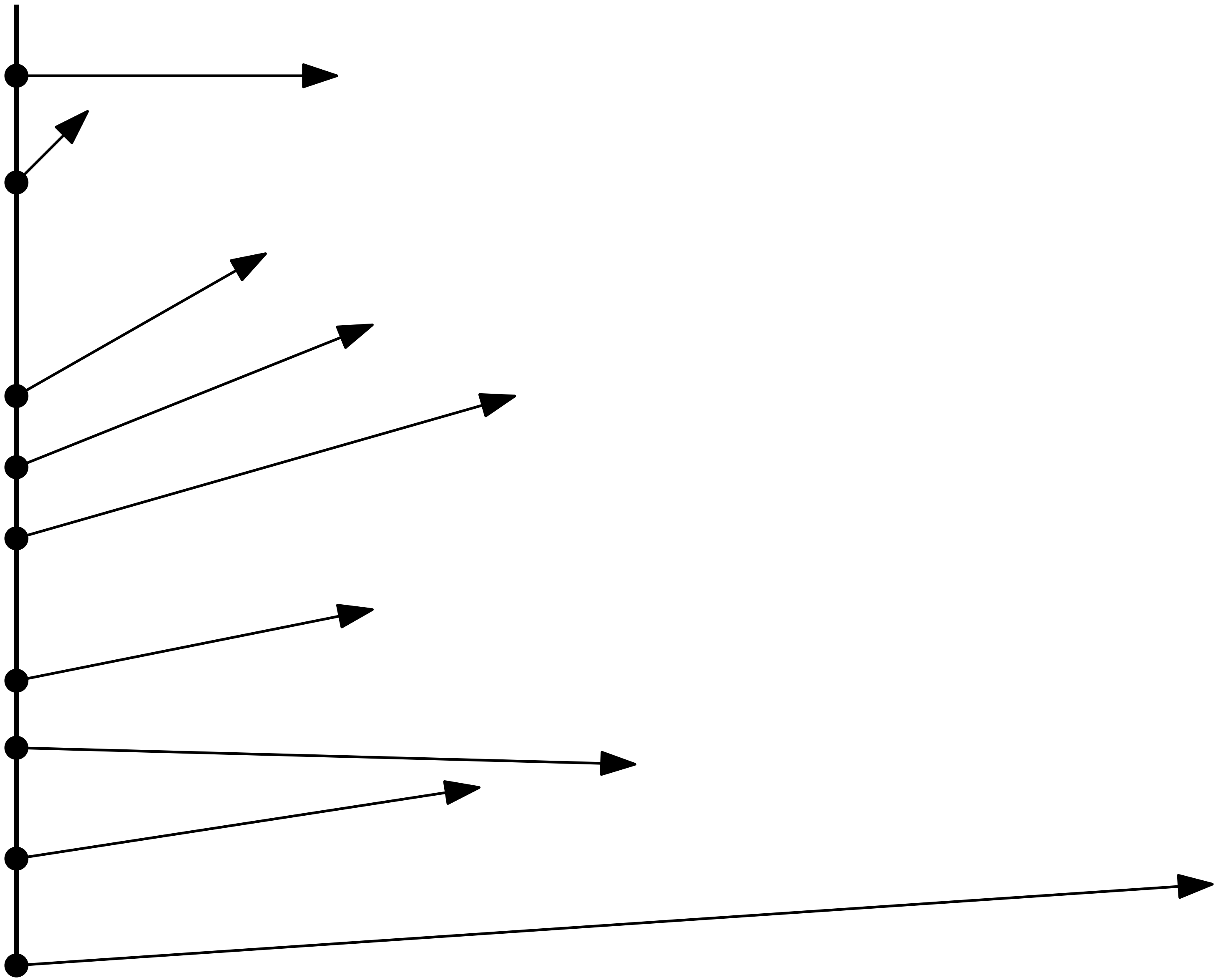
Bikers start at the same time at unit speed. If a biker runs into the tracks of another it stops.



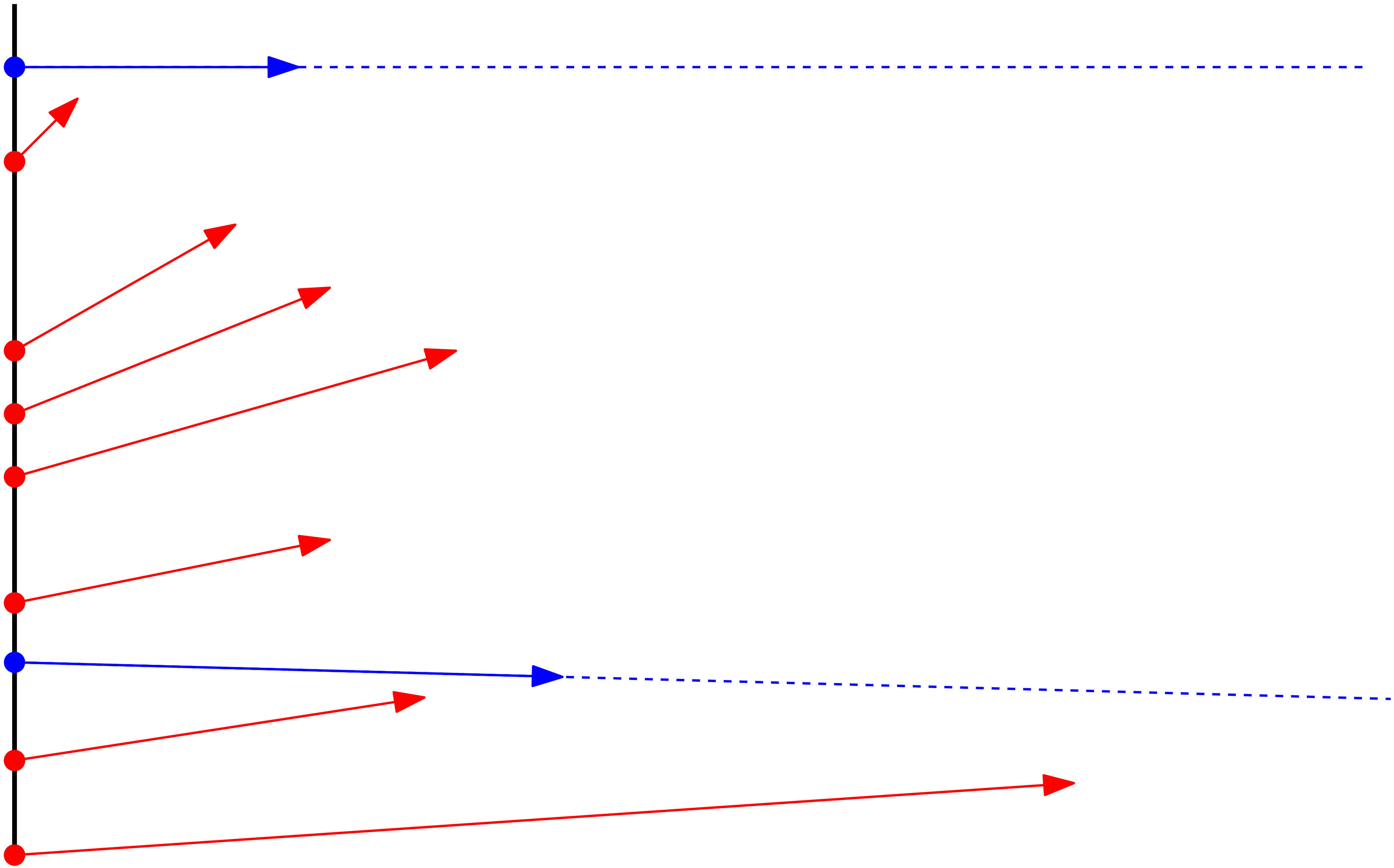
Which bikers get to ride forever?

# Admission Control

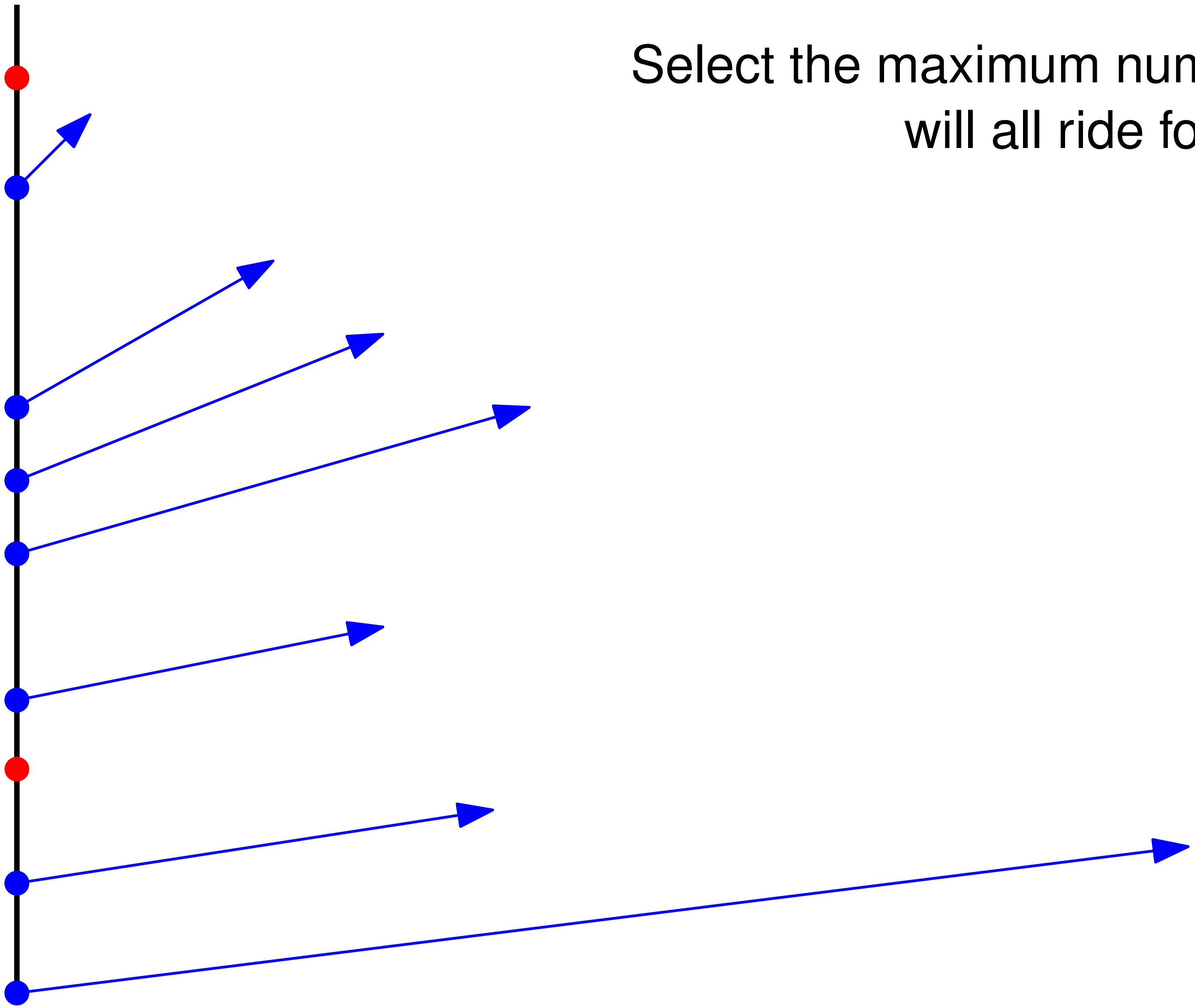
# Admission Control



# Admission Control

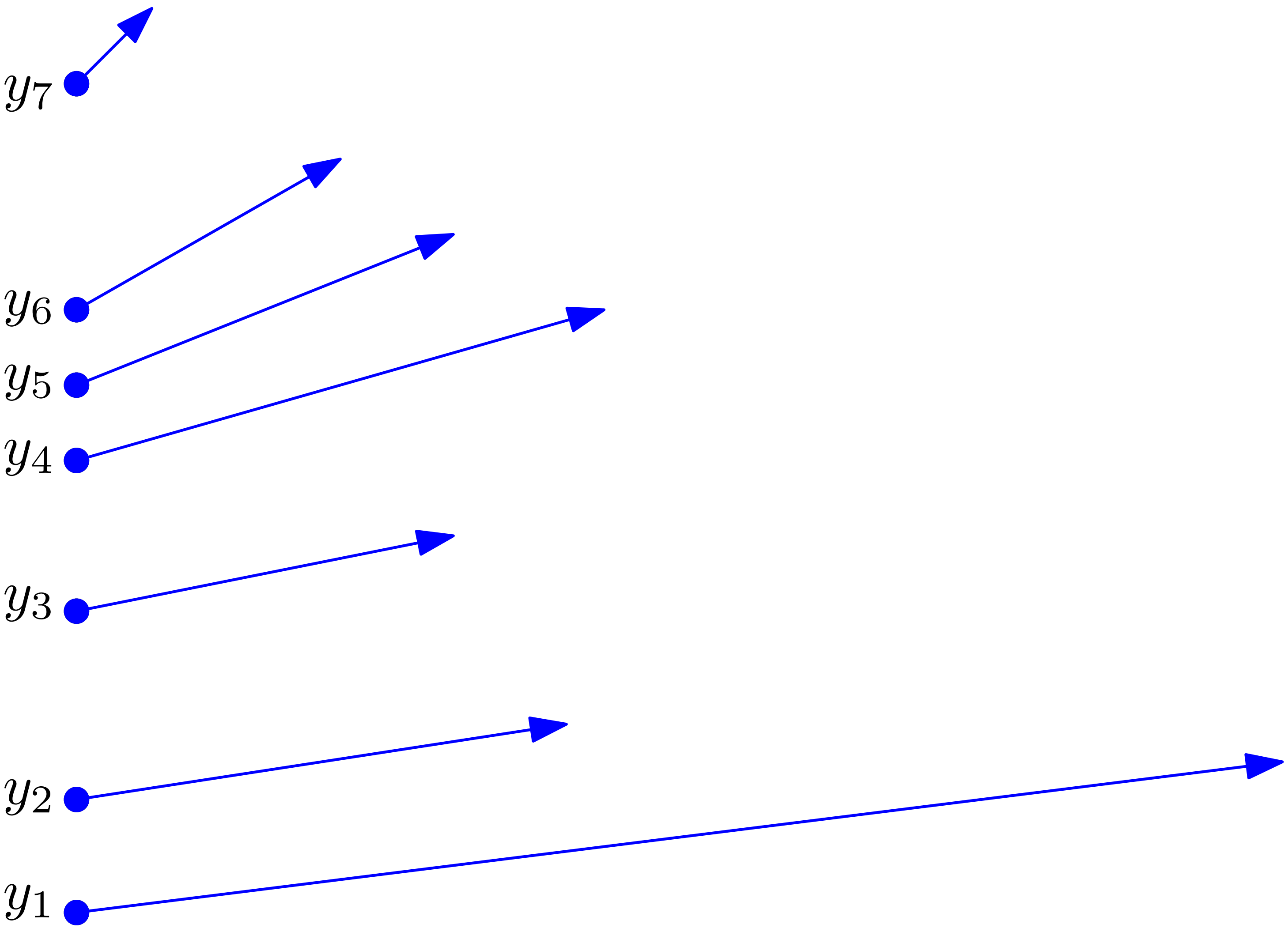


# Admission Control

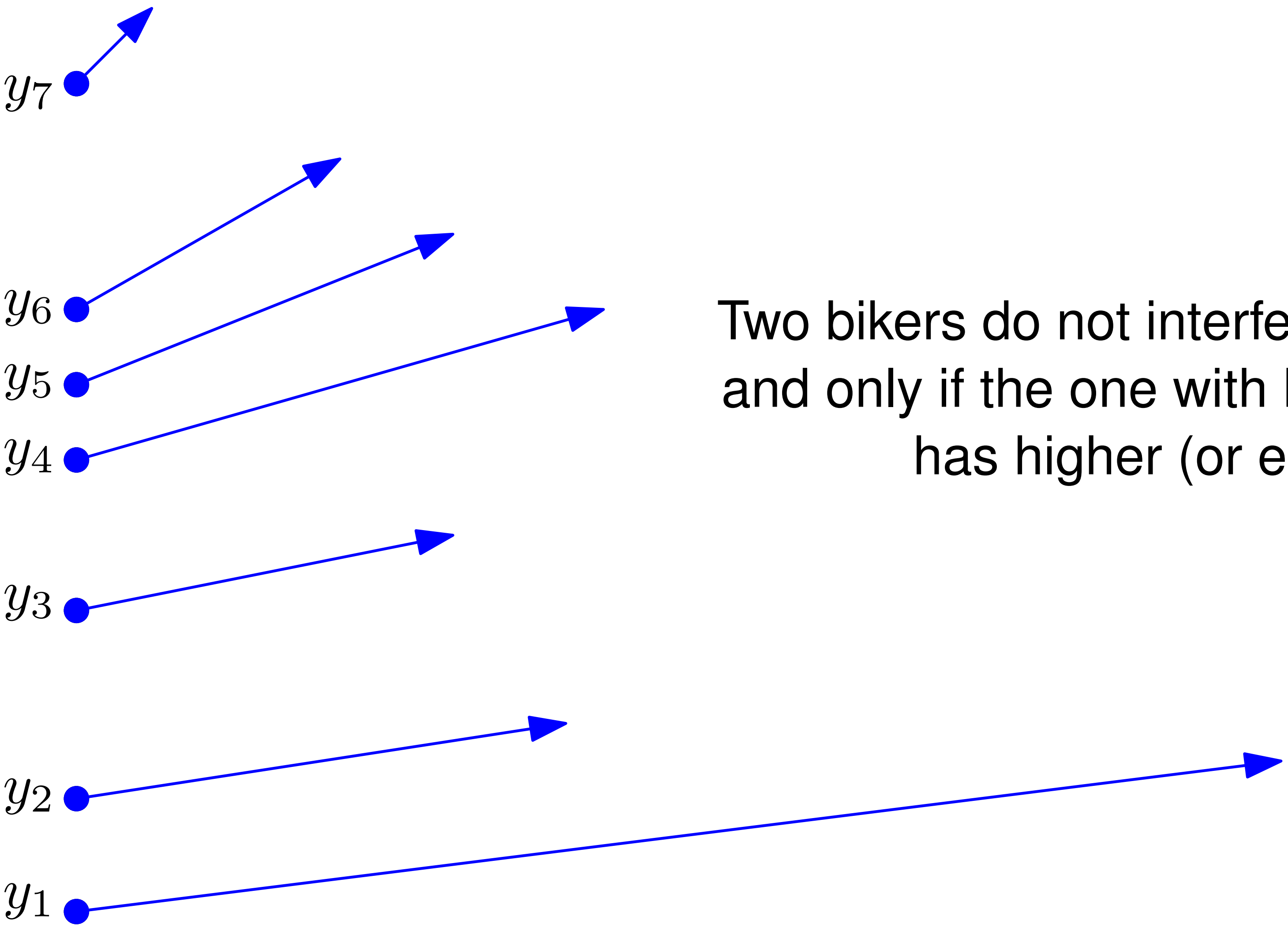


Select the maximum number of bikers that will all ride forever.

# Admission Control

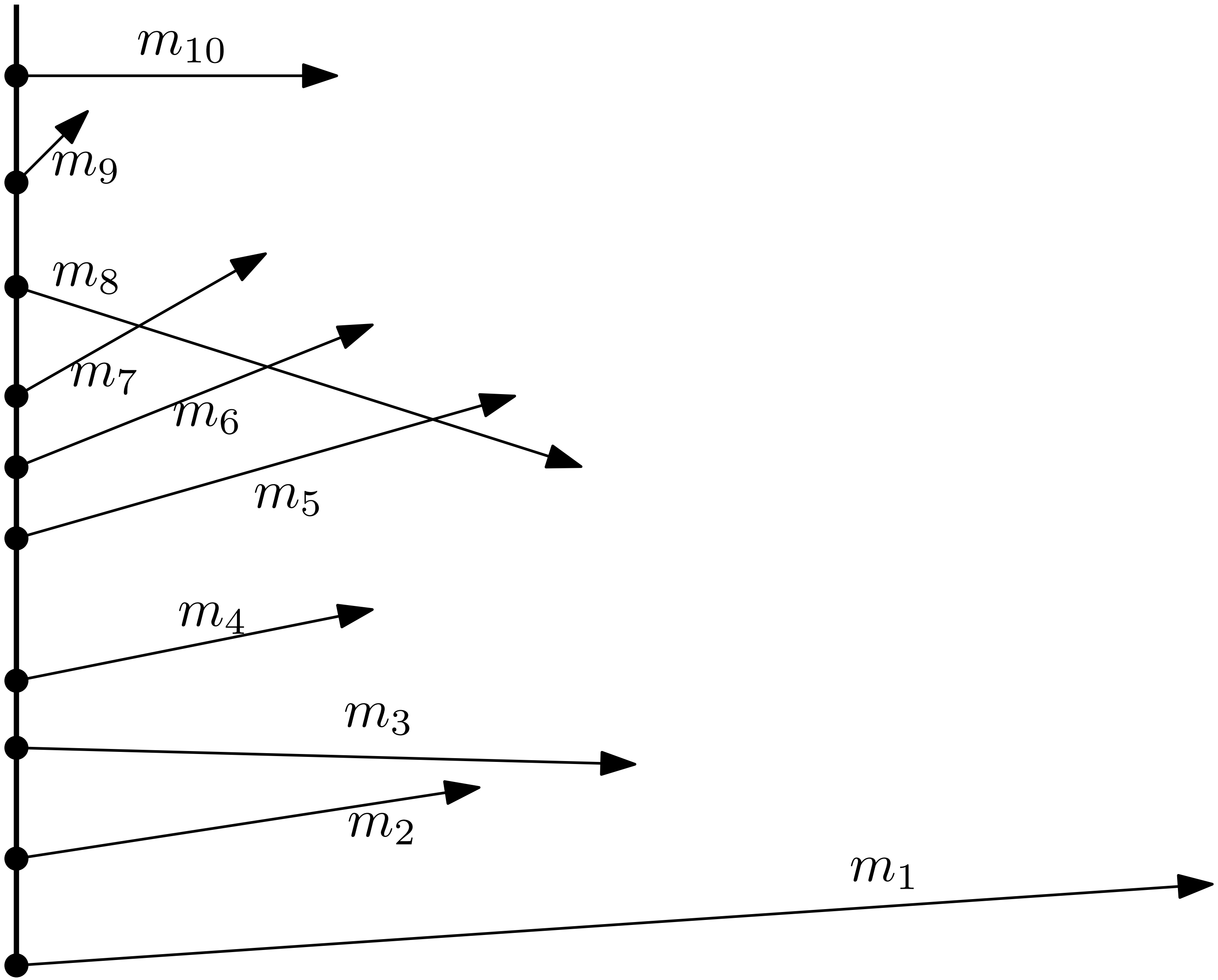


# Admission Control

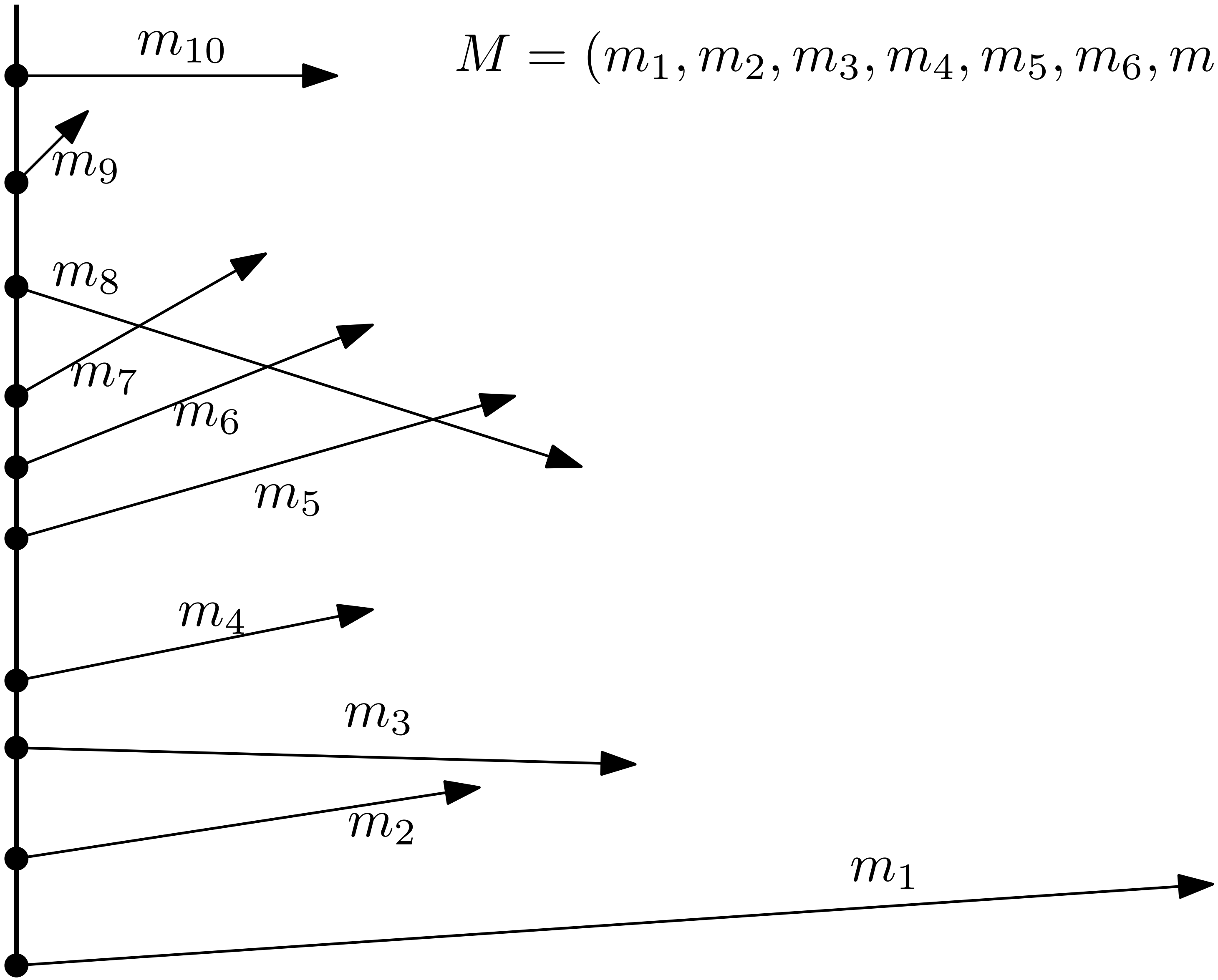



Two bikers do not interfere with each other if and only if the one with higher starting point has higher (or equal) slope.

# Admission Control

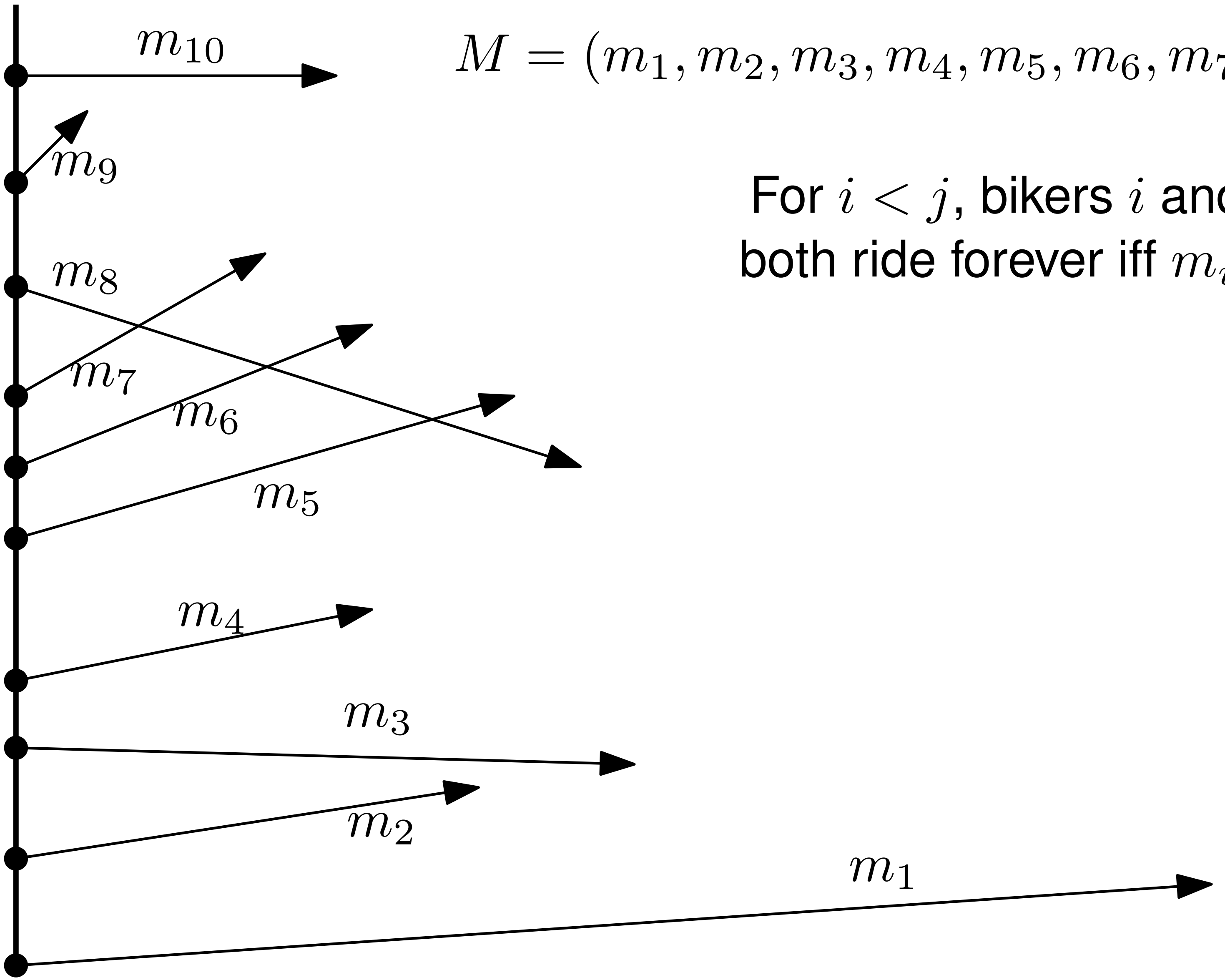


# Admission Control




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

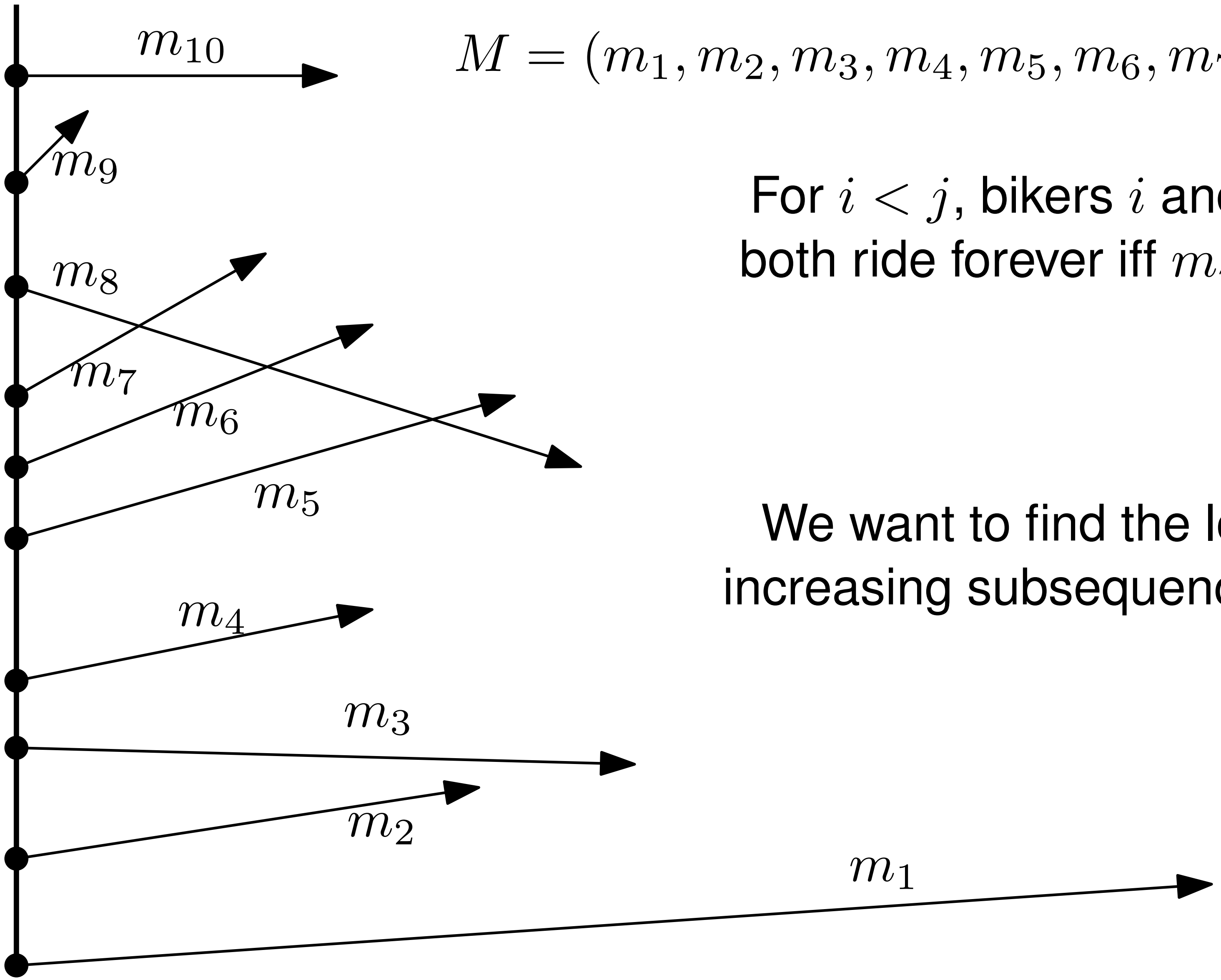
# Admission Control




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

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

We want to find the longest increasing subsequence in $M$.

# Longest increasing subsequence

- Classic Dynamic programming leads to $O(n^2)$ time.
- Possible in $O(n \log n)$ time, however.

# Longest increasing subsequence

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

$L1$

$L2$

$L3$

$L4$

$L5$

$L6$

# Longest increasing subsequence

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

$L1$ $0$ $\leftarrow$ increasing subsequence of length 1

$L2$

$L3$

$L4$

$L5$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 8$ $\leftarrow$ increasing subsequence of length 2

$L3$

$L4$

$L5$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 4$ $\leftarrow$ increasing subsequence of length 2 (with smallest possible last entry)

$L3$

$L4$

$L5$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 4$

$L3$ $0, 4, 12$ $\leftarrow$ increasing subsequence of length 3

$L4$

$L5$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 4$

$L3$ $0, 4, 12$

$L4$

$L5$

$L6$

# Longest increasing subsequence

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

$L1$ $0$ $\leftarrow$ cannot improve since $L1$ ends in $0$ ($< 2$)

$L2$ $0, 4$

$L3$ $0, 4, 12$

$L4$

$L5$

$L6$

# Longest increasing subsequence

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

$L1$ $0$ $\leftarrow$ cannot improve since $L1$ ends in $0$ ($< 2$)

$L2$ $0, 4$

$L3$ $0, 4, 12$ $\leftarrow$ cannot improve since $L2$ ends in $4$ ($> 2$)

$L4$

$L5$

$L6$

# Longest increasing subsequence

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

$L1$ $0$ $\leftarrow$ cannot improve since $L1$ ends in $0$ ($< 2$)

$L2$ $0, 4$ $\leftarrow$ improves by adding $2$ at the end of $L1$

$L3$ $0, 4, 12$ $\leftarrow$ cannot improve since $L2$ ends in $4$ ($> 2$)

$L4$

$L5$

$L6$

# Longest increasing subsequence

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

$L1$ $0$ $\leftarrow$ cannot improve since $L1$ ends in $0$ ($< 2$)

$L2$ $0, 2$ $\leftarrow$ improves by adding $2$ at the end of $L1$

$L3$ $0, 4, 12$ $\leftarrow$ cannot improve since $L2$ ends in $4$ ($> 2$)

$L4$

$L5$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 2$

$L3$ $0, 4, 12$ $\leftarrow$ improves by adding $10$ at the end of $L2$

$L4$

$L5$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 2$

$L3$ $0, 2, 10$ $\leftarrow$ improves by adding $10$ at the end of $L2$

$L4$

$L5$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 2$

$L3$ $0, 2, 10$ $\leftarrow$ improves by adding $6$ at the end of $L2$

$L4$

$L5$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 2$

$L3$ $0, 2, 6$ $\leftarrow$ improves by adding $6$ at the end of $L2$

$L4$

$L5$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 2$

$L3$ $0, 2, 6$

$L4$ $\leftarrow$ “improves” by adding $14$ at the end of $L3$

$L5$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 2$

$L3$ $0, 2, 6$

$L4$ $0, 2, 6, 14$ $\leftarrow$ "improves" by adding $14$ at the end of $L3$

$L5$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 2$ $\leftarrow$ improves by adding $1$ at the end of $L1$

$L3$ $0, 2, 6$

$L4$ $0, 2, 6, 14$

$L5$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 1$ $\leftarrow$ improves by adding $1$ at the end of $L1$

$L3$ $0, 2, 6$

$L4$ $0, 2, 6, 14$

$L5$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 1$

$L3$ $0, 2, 6$

$L4$ $0, 2, 6, 14$ $\leftarrow$ improves by adding $9$ at the end of $L3$

$L5$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 1$

$L3$ $0, 2, 6$

$L4$ $0, 2, 6, 9$ $\leftarrow$ improves by adding $9$ at the end of $L3$

$L5$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 1$

$L3$ $0, 2, 6$ $\leftarrow$ improves by adding $5$ at the end of $L2$

$L4$ $0, 2, 6, 9$

$L5$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 1$

$L3$ $0, 1, 5$ $\leftarrow$ improves by adding $5$ at the end of $L2$

$L4$ $0, 2, 6, 9$

$L5$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 1$

$L3$ $0, 1, 5$

$L4$ $0, 2, 6, 9$

$L5$ $\leftarrow$ "improves" by adding $13$ at the end of $L4$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 1$

$L3$ $0, 1, 5$

$L4$ $0, 2, 6, 9$

$L5$ $0, 2, 6, 9, 13$ $\leftarrow$ "improves" by adding $13$ at the end of $L4$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 1$

$L3$ $0, 1, 5$ $\leftarrow$ improves by adding $3$ at the end of $L2$

$L4$ $0, 2, 6, 9$

$L5$ $0, 2, 6, 9, 13$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 1$

$L3$ $0, 1, 3$ $\leftarrow$ improves by adding $3$ at the end of $L2$

$L4$ $0, 2, 6, 9$

$L5$ $0, 2, 6, 9, 13$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 1$

$L3$ $0, 1, 3$

$L4$ $0, 2, 6, 9$

$L5$ $0, 2, 6, 9, 13$ $\leftarrow$ improves by adding $11$ at the end of $L4$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 1$

$L3$ $0, 1, 3$

$L4$ $0, 2, 6, 9$

$L5$ $0, 2, 6, 9, 11$ $\leftarrow$ improves by adding $11$ at the end of $L4$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 1$

$L3$ $0, 1, 3$

$L4$ $0, 2, 6, 9$ $\leftarrow$ improves by adding $7$ at the end of $L3$

$L5$ $0, 2, 6, 9, 11$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 1$

$L3$ $0, 1, 3$

$L4$ $0, 1, 3, 7$ $\leftarrow$ improves by adding $7$ at the end of $L3$

$L5$ $0, 2, 6, 9, 11$

$L6$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 1$

$L3$ $0, 1, 3$

$L4$ $0, 1, 3, 7$

$L5$ $0, 2, 6, 9, 11$

$L6$ $\leftarrow$ "improves" by adding $15$ at the end of $L5$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 1$

$L3$ $0, 1, 3$

$L4$ $0, 1, 3, 7$

$L5$ $0, 2, 6, 9, 11$

$L6$ $0, 2, 6, 9, 11, 15$ $\leftarrow$ “improves” by adding $15$ at the end of $L5$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 1$

$L3$ $0, 1, 3$

$L4$ $0, 1, 3, 7$

$L5$ $0, 2, 6, 9, 11$

$L6$ $0, 2, 6, 9, 11, 15$

In every iteration, exactly one row changes $\Rightarrow$ time $O(n^2)$

# Longest increasing subsequence

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

$L1$ $0$

$L2$ $0, 1$

$L3$ $0, 1, 3$

$L4$ $0, 1, 3, 7$

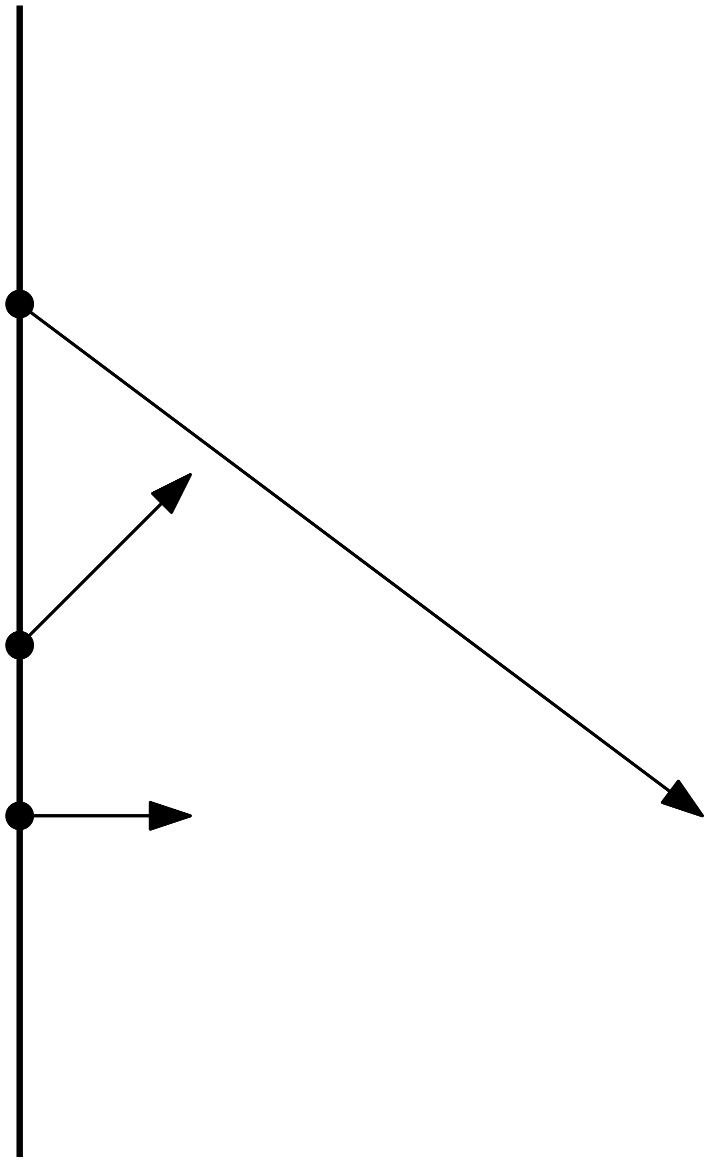
$L5$ $0, 2, 6, 9, 11$

$L6$ $0, 2, 6, 9, 11, 15$

In every iteration, exactly one row changes $\Rightarrow$ time $O(n^2)$

We can also compute only the main diagonal $\Rightarrow$ time $O(n \log n)$

# Starting Schedules

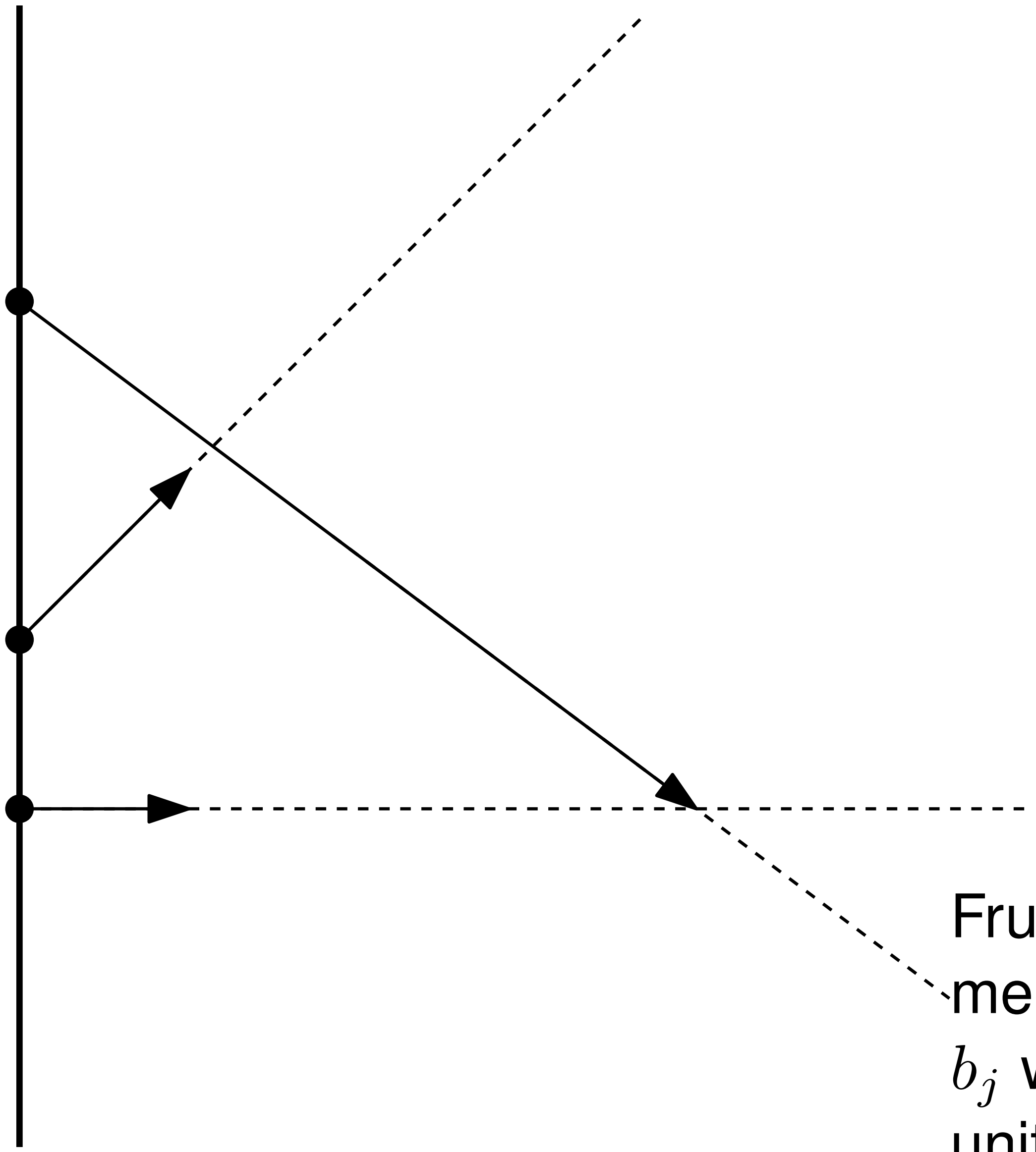


We are allowed to modify the starting time $s_i$ of each bike $b_i$.

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

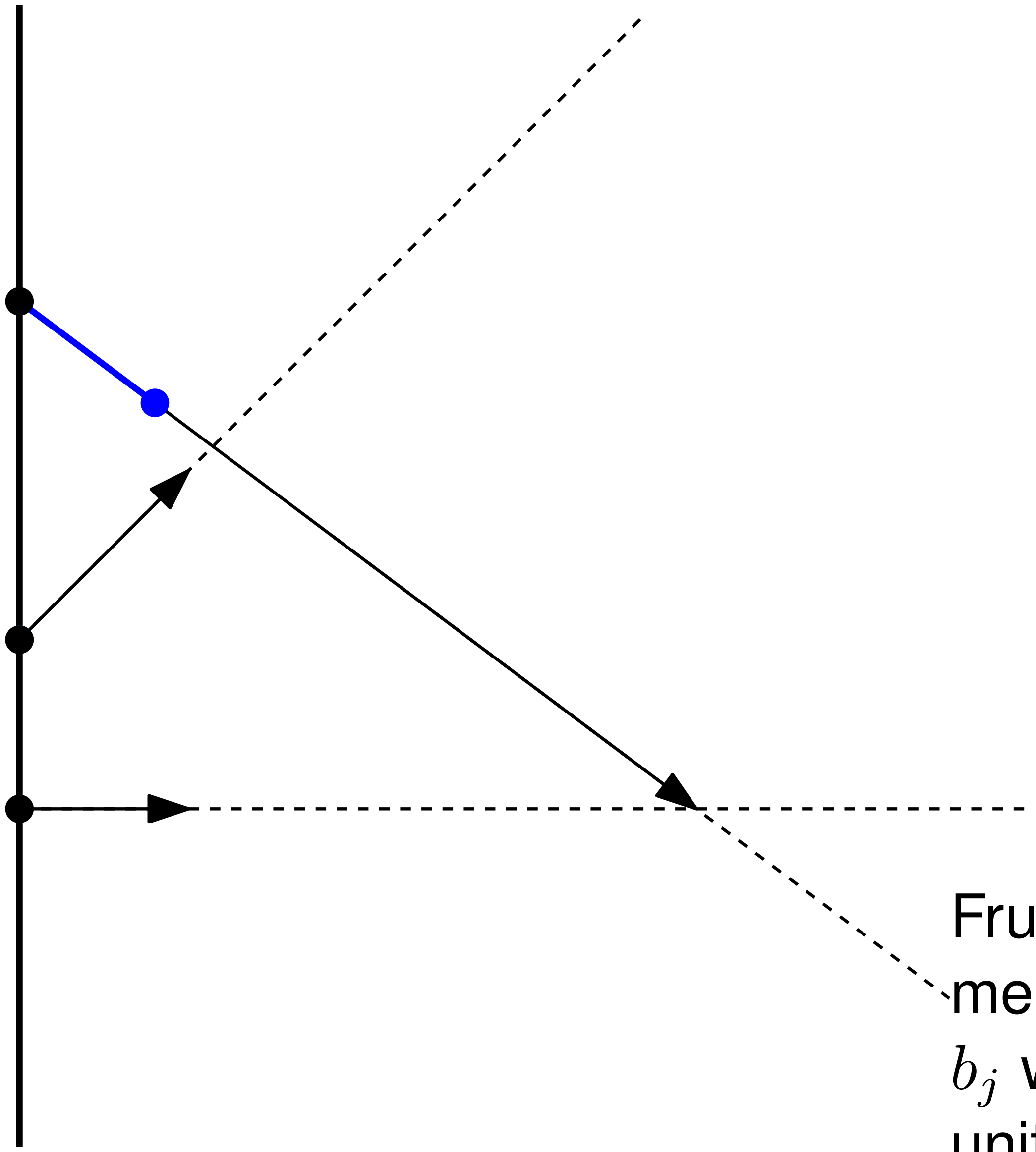
We are allowed to modify the starting time $s_i$ of each bike $b_i$.



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

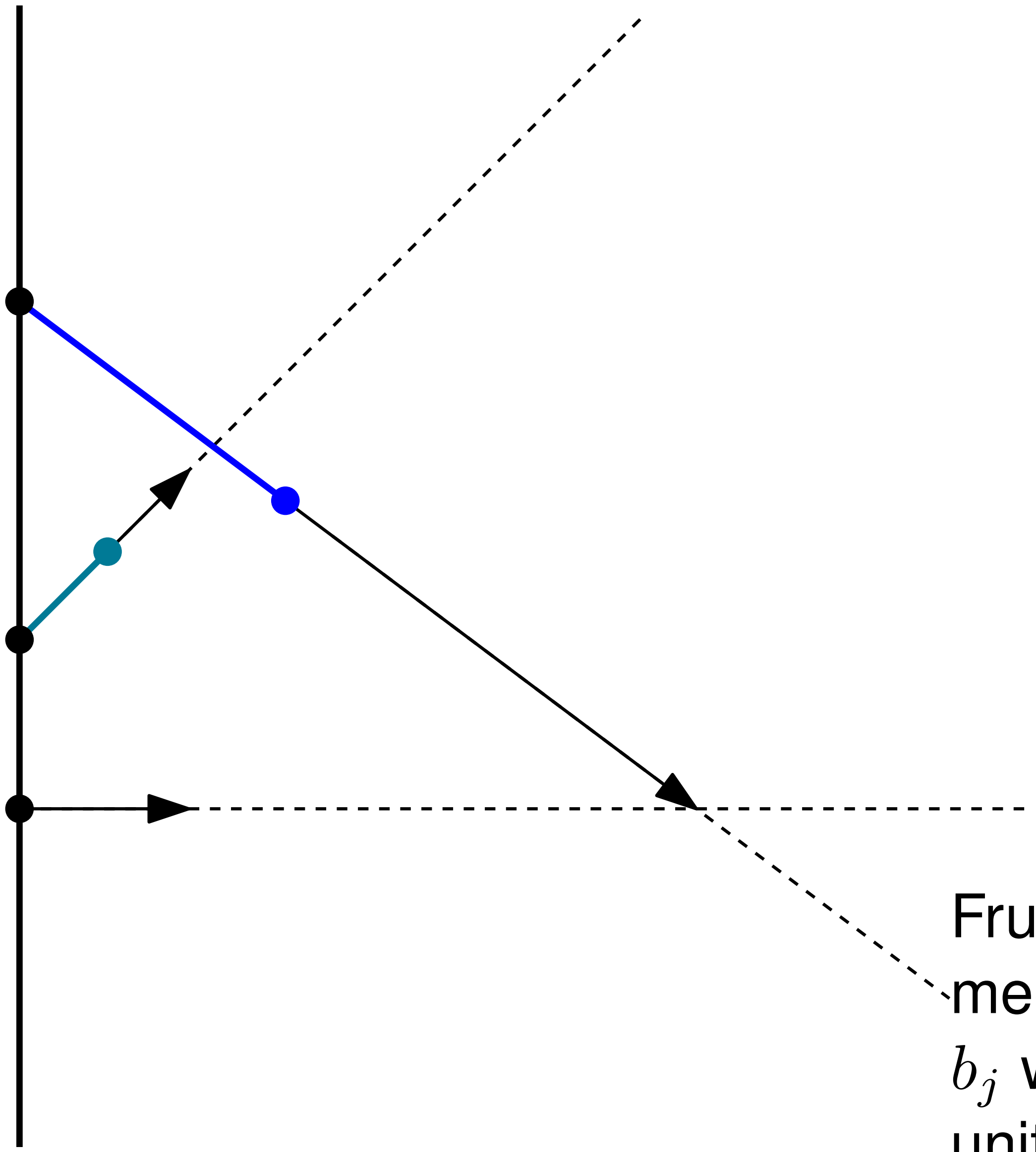
We are allowed to modify the starting time $s_i$ of each bike $b_i$.



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.

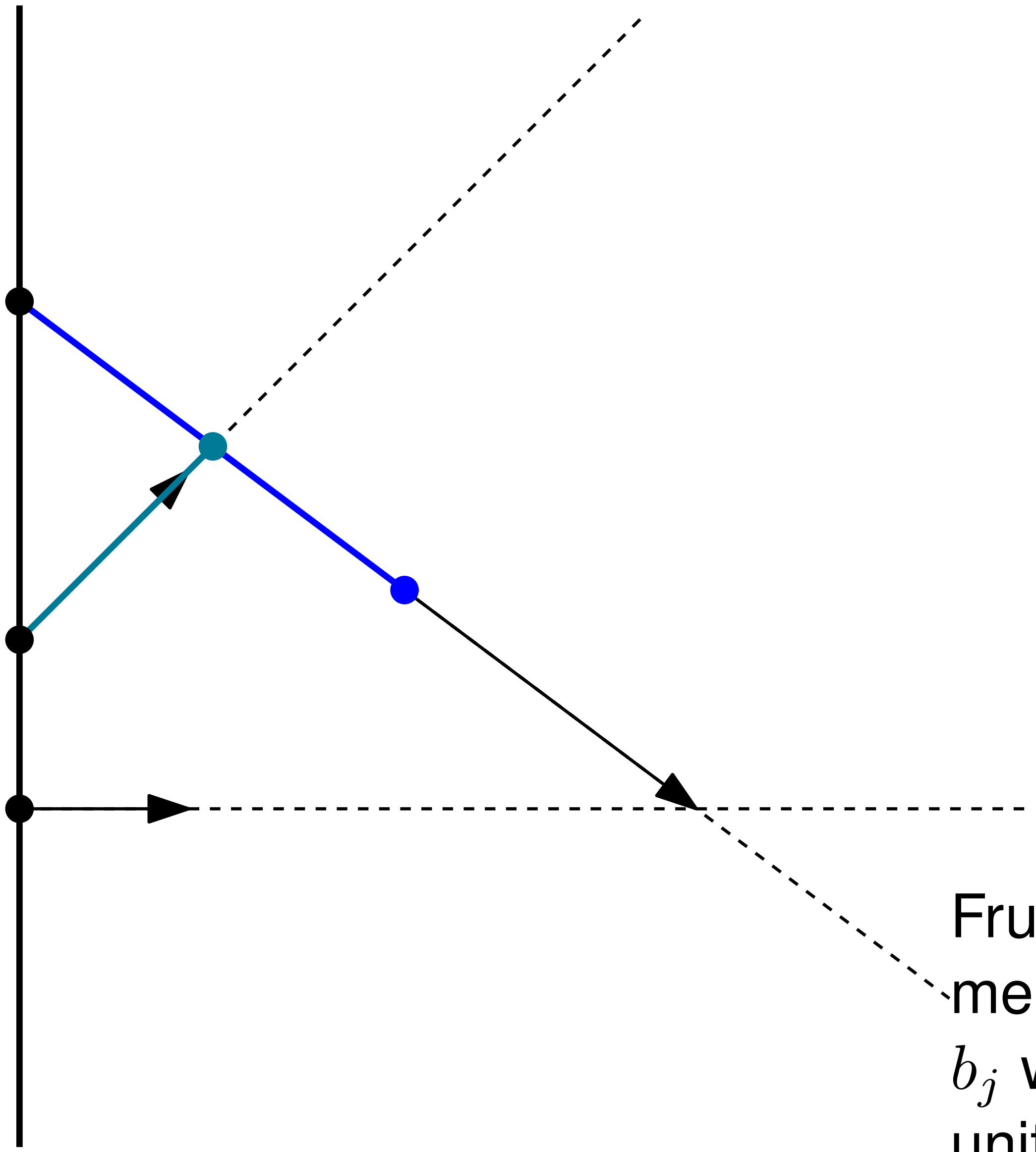
# Starting Schedules

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



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

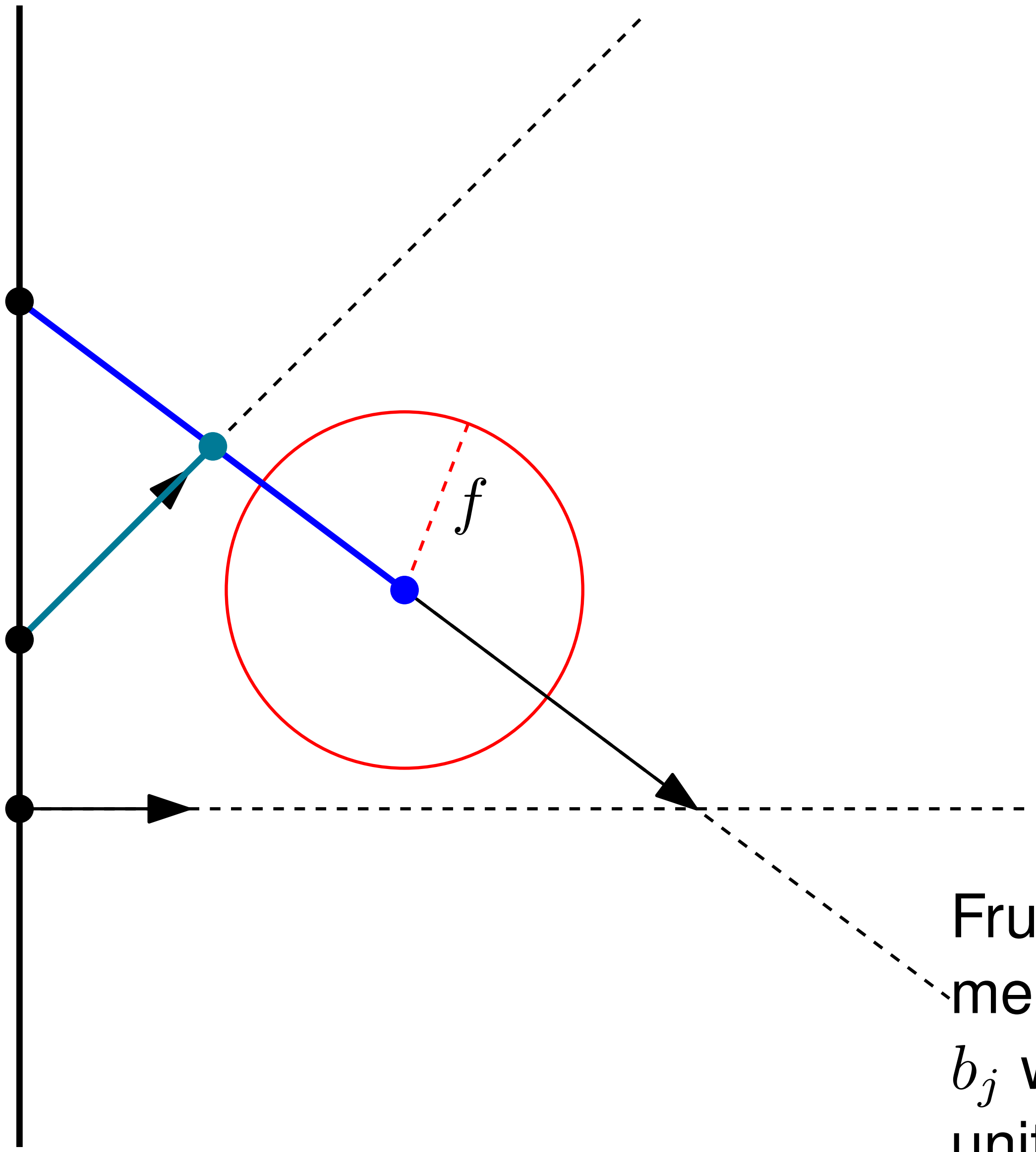


We are allowed to modify the starting time $s_i$ of each bike $b_i$.

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

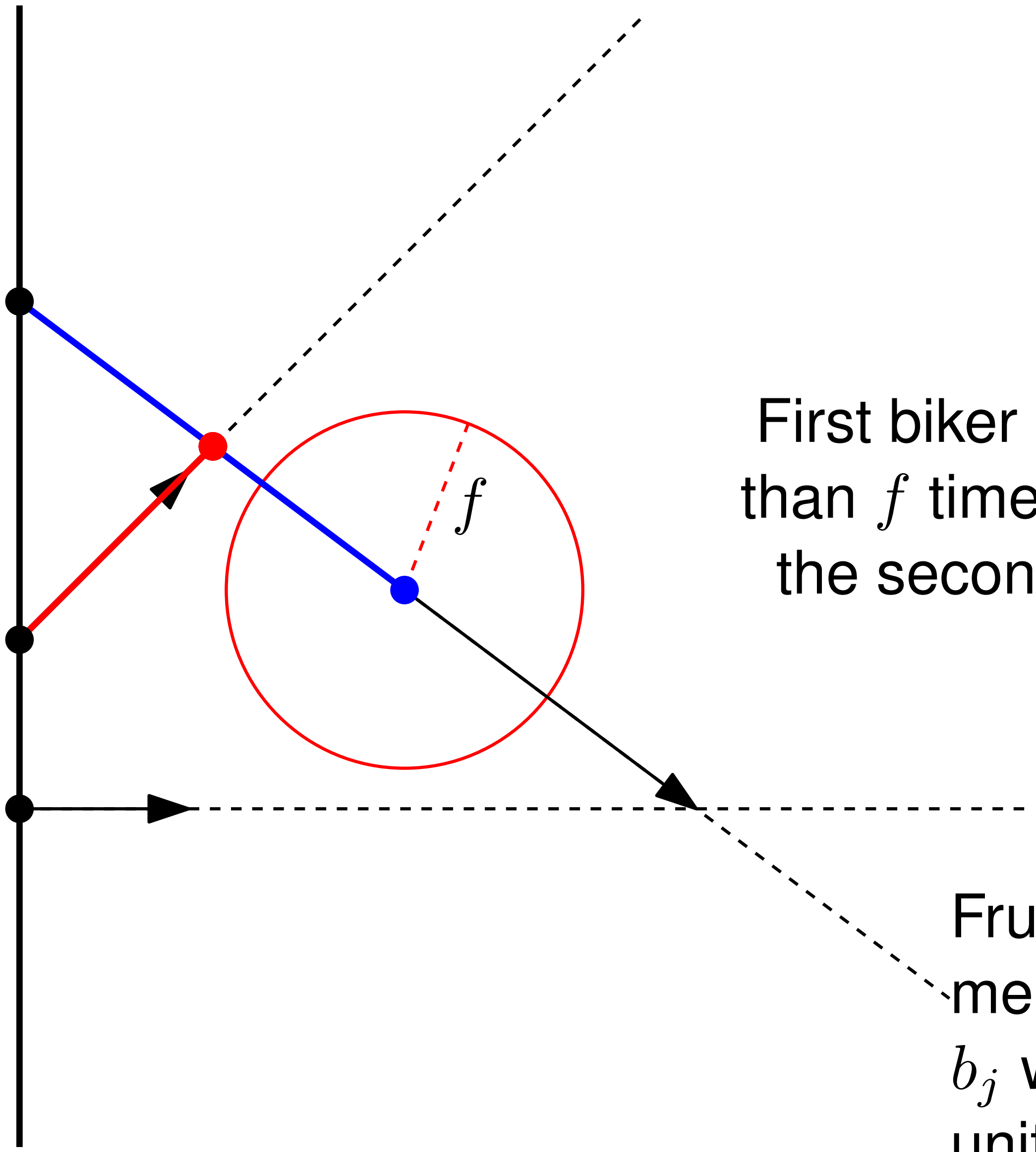
We are allowed to modify the starting time $s_i$ of each bike $b_i$.




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 $b_i$.

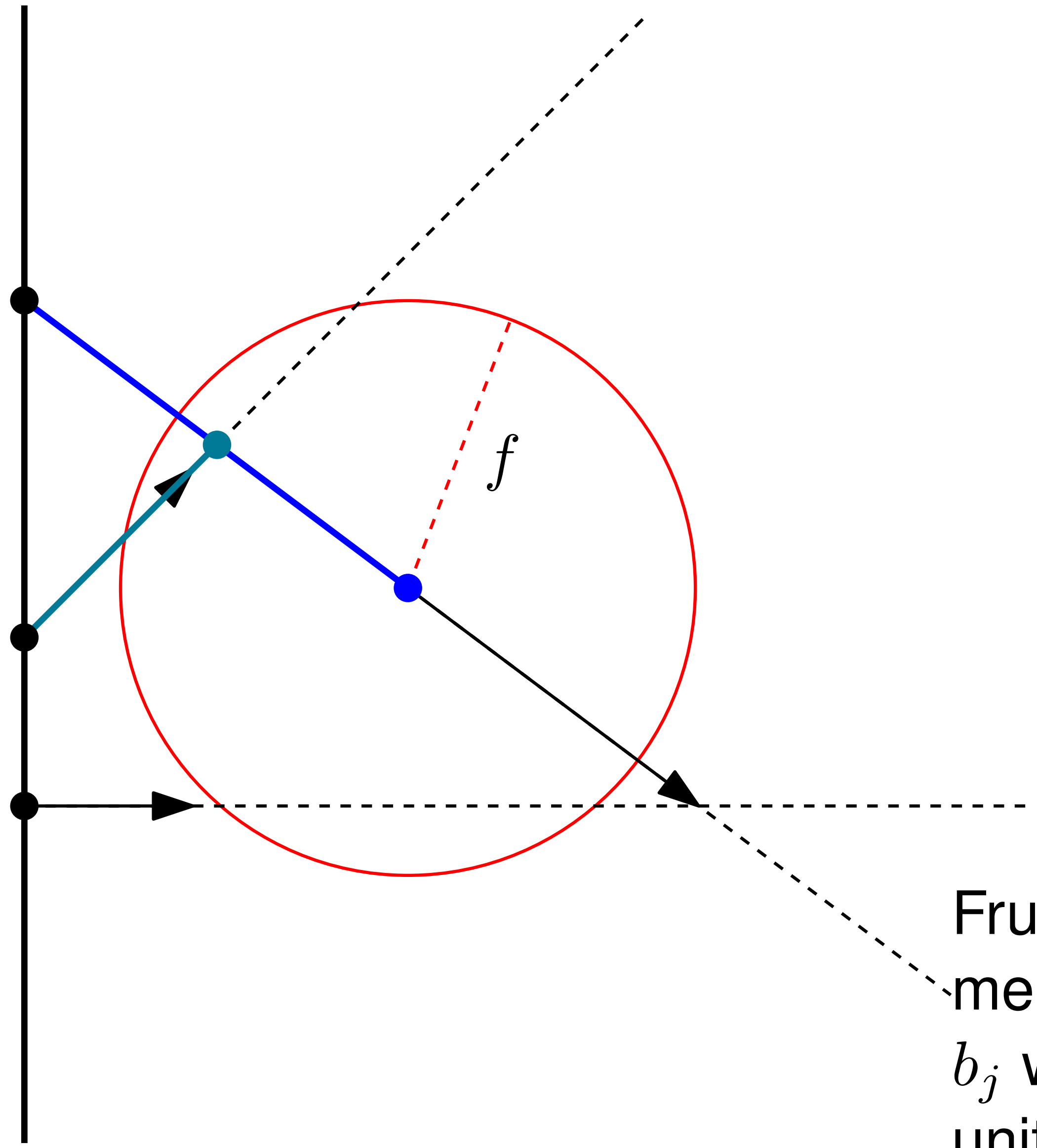



First biker passed more than $f$ time units ago, so the second must stop.

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

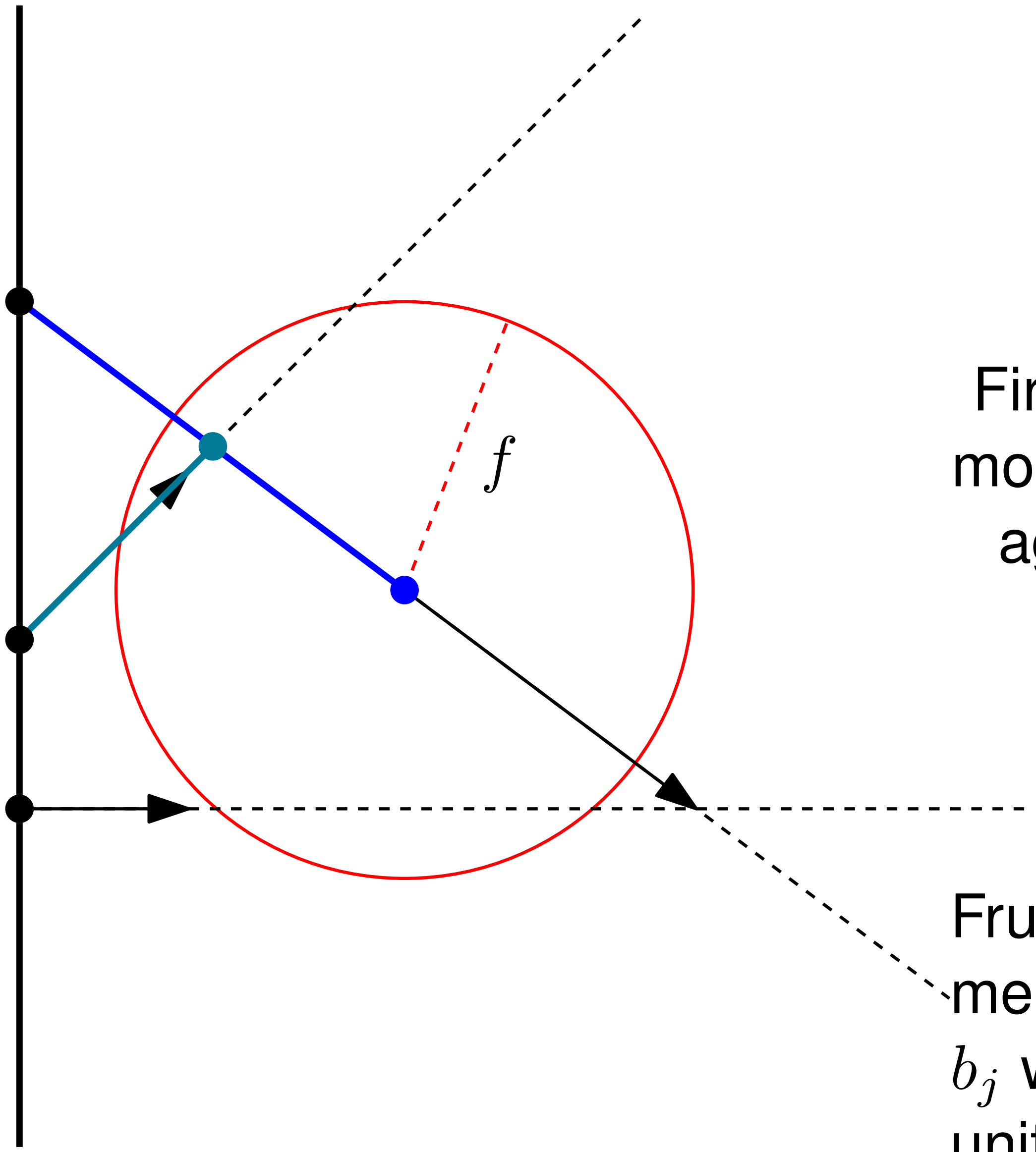
We are allowed to modify the starting time $s_i$ of each bike $b_i$.




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 $b_i$.

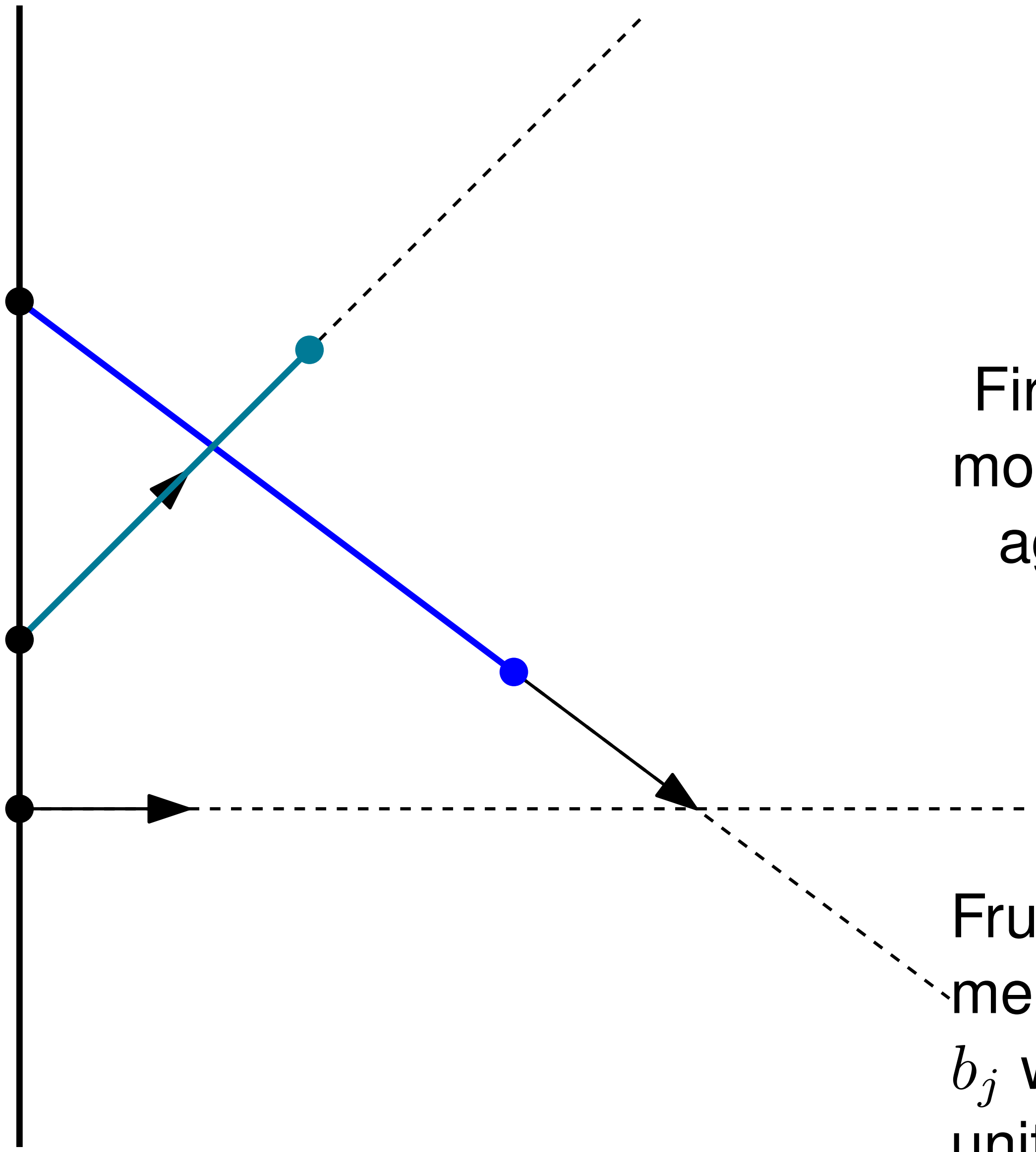



First biker passed no more than $f$ time units ago, so the second keeps riding.

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 $b_i$.

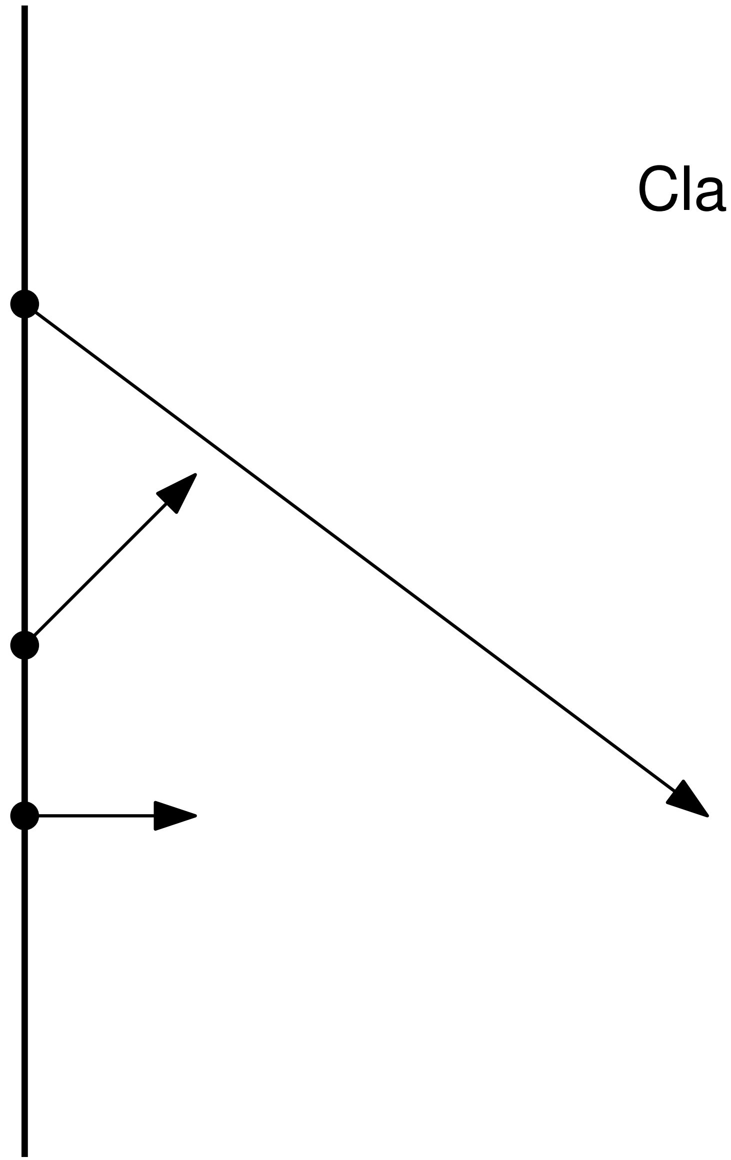


First biker passed no more than $f$ time units ago, so the second keeps riding.

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 $b_i$.



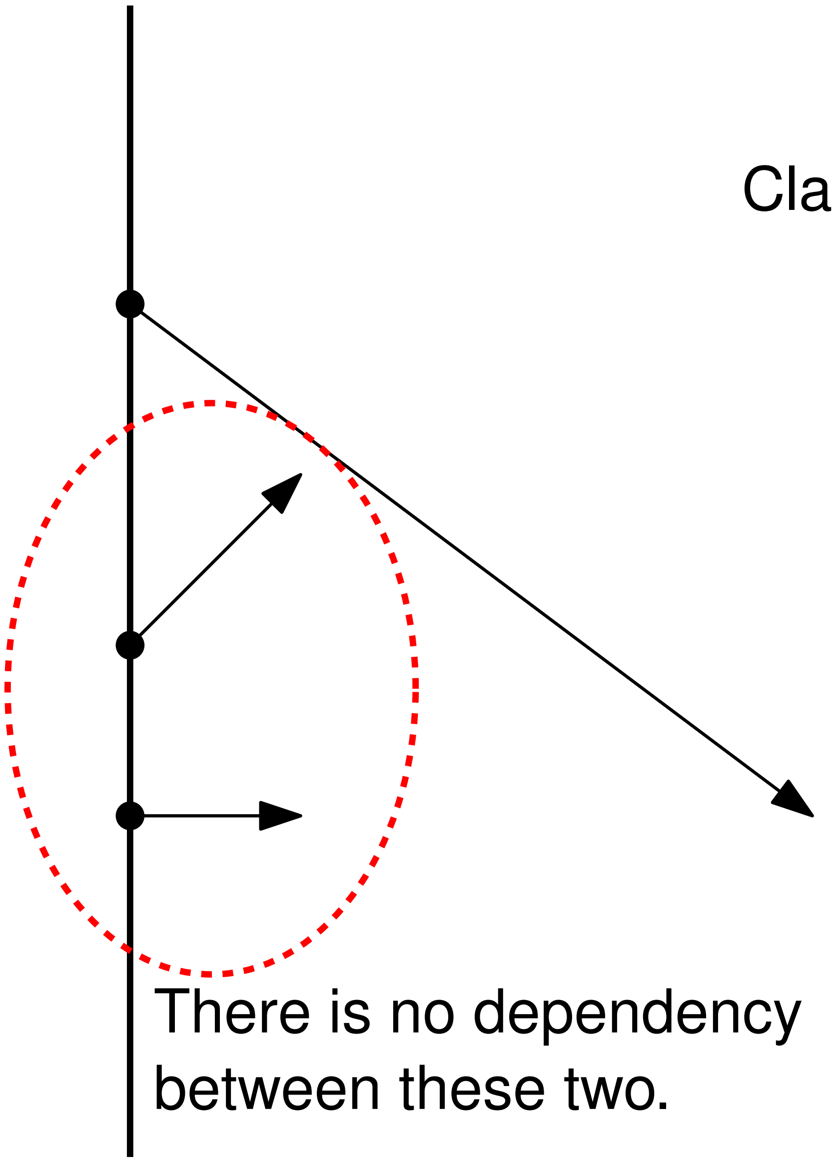
Claim: We can solve this instance with $f = 0$.

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 $f = 0$.

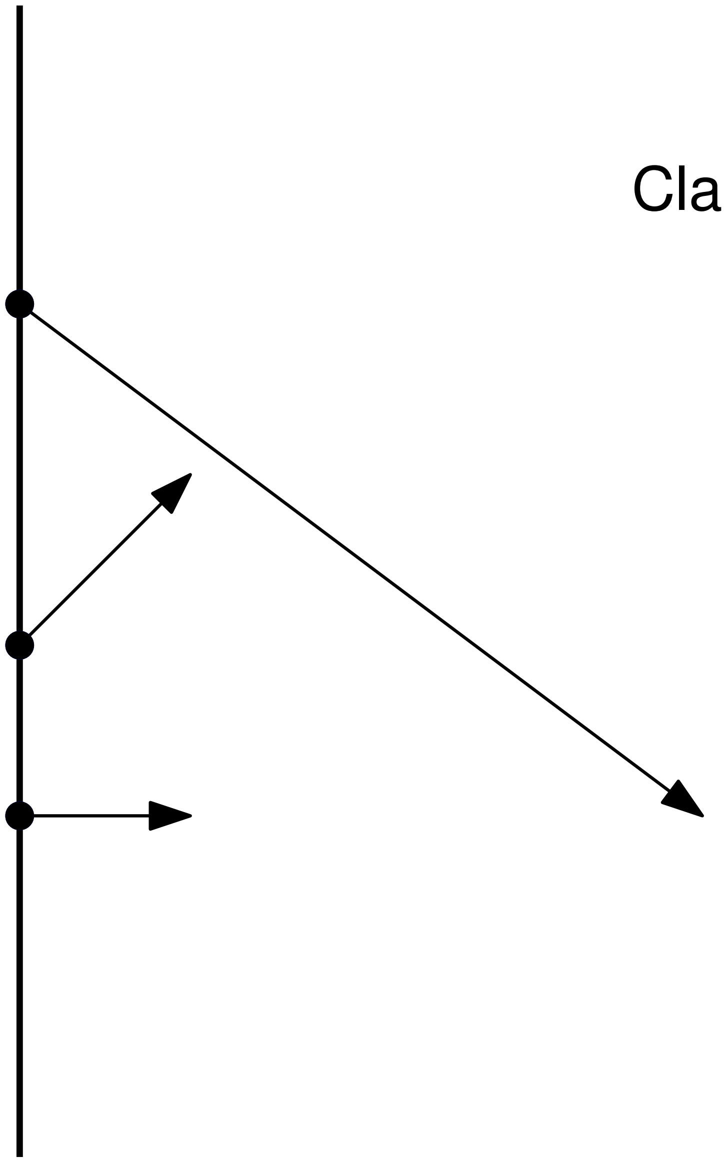


There is no dependency between these two.

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 $b_i$.



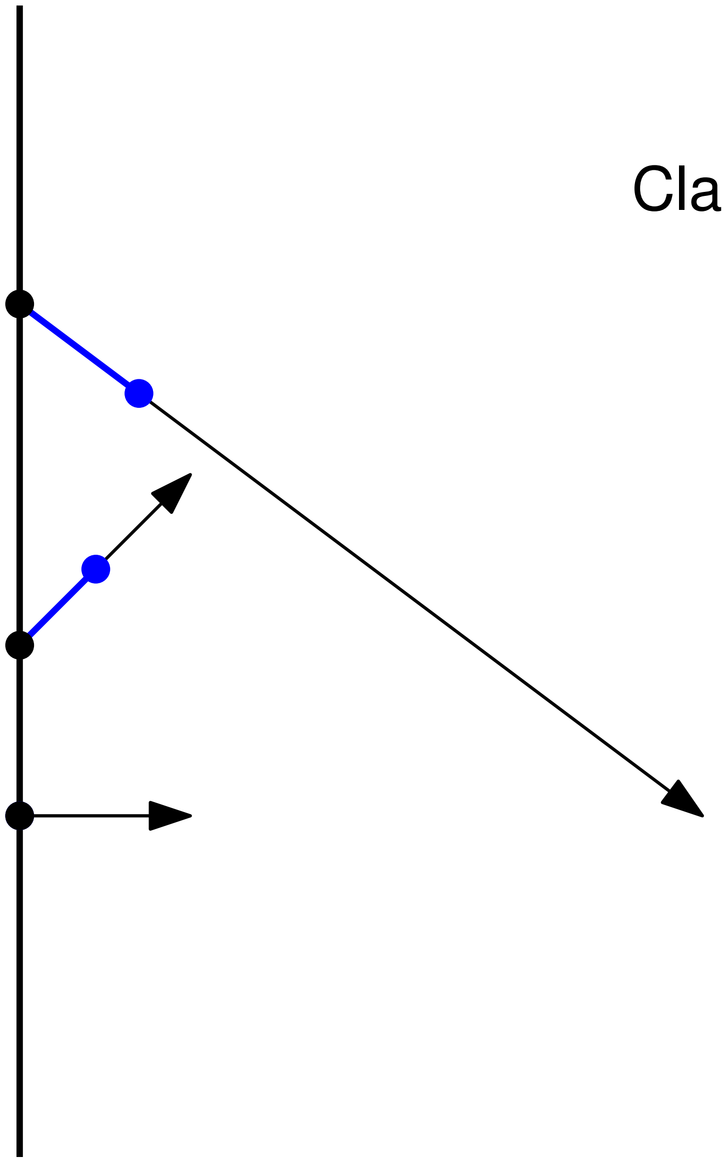
Claim: We can solve this instance with $f = 0$.

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 $b_i$.

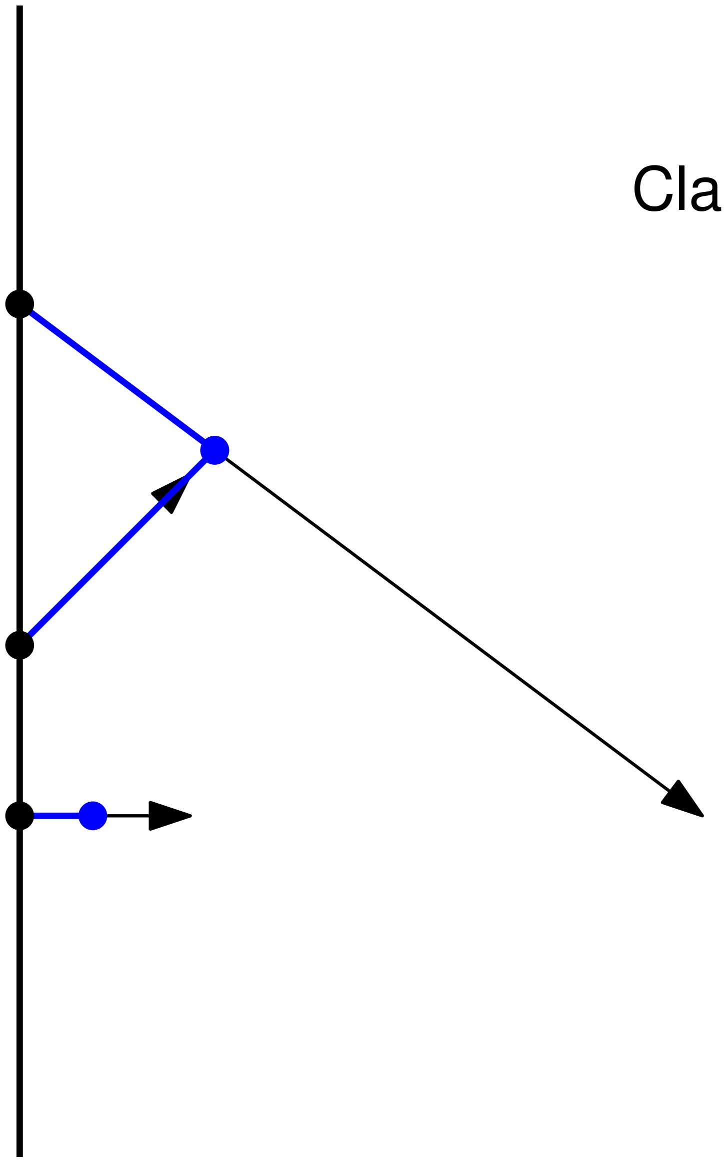
Claim: We can solve this instance with $f = 0$.



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 $b_i$.

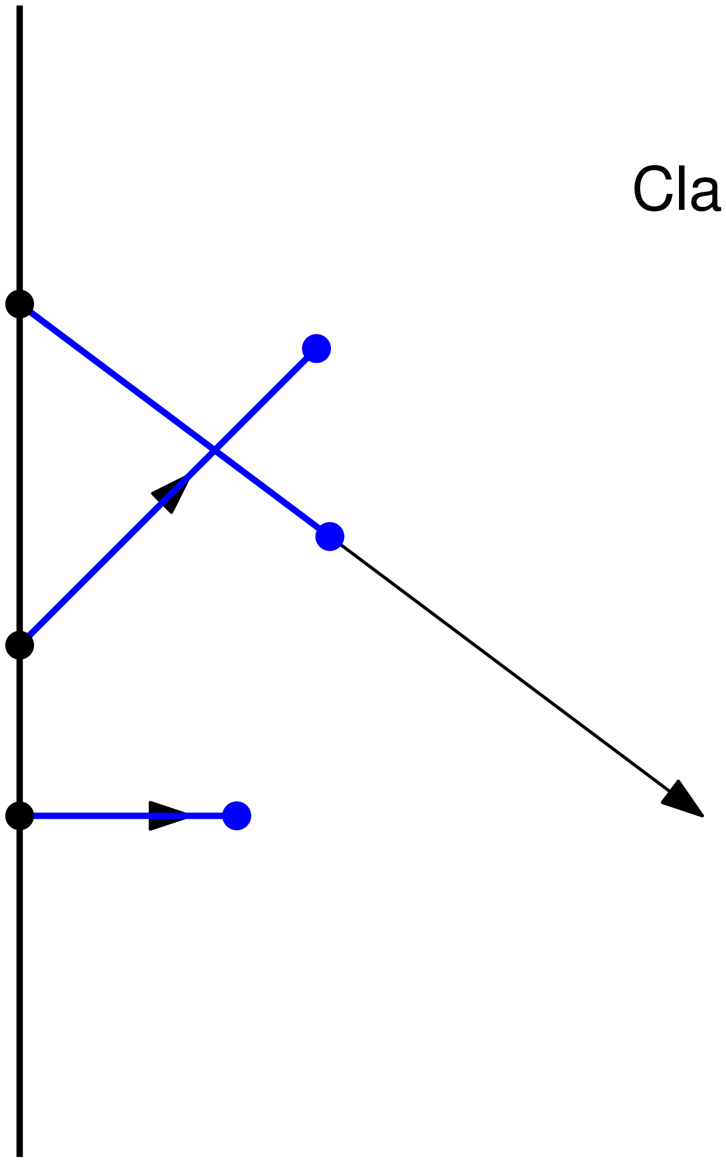


Claim: We can solve this instance with $f = 0$.

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 $b_i$.

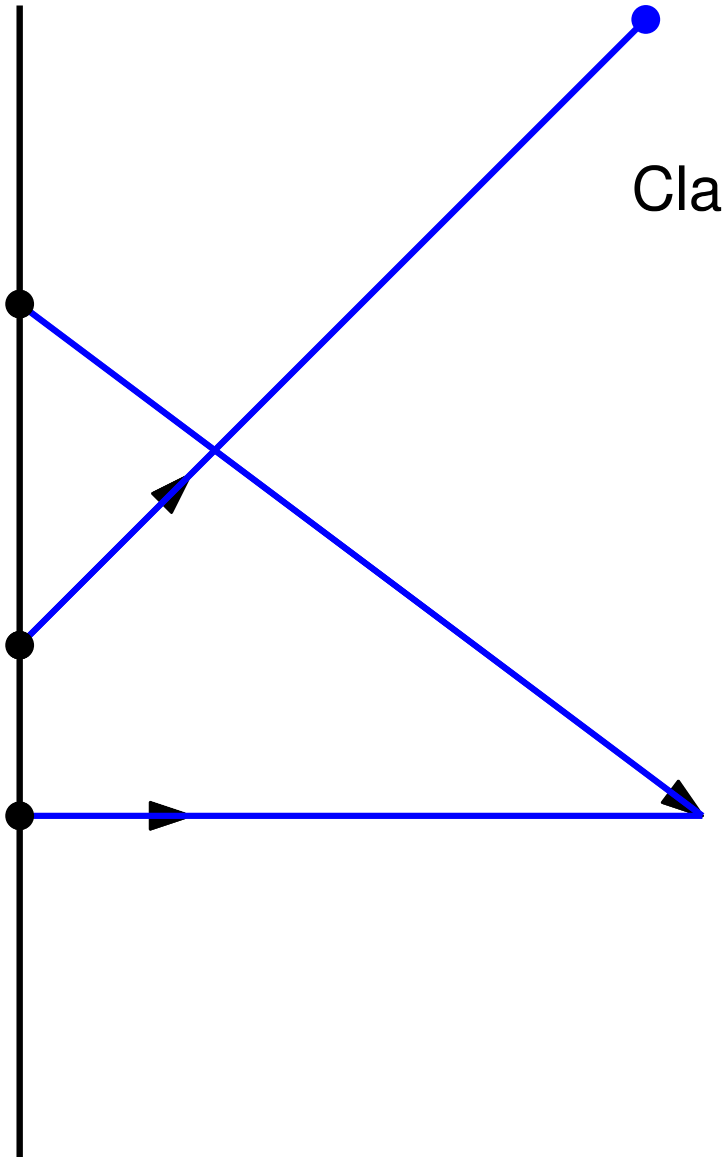


Claim: We can solve this instance with $f = 0$.

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 $f = 0$.

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?

- Starting time $s_i$ of each biker
- Frustration tolerance $f$

# Solving it with Linear Programming

What are the variables?

- Starting time $s_i$ of each biker
- Frustration tolerance $f$

From the specification, we know there are at most 101 variables, which is still OK for 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?

- Starting time $s_i$ of each biker
- Frustration tolerance $f$

What are the constraints?

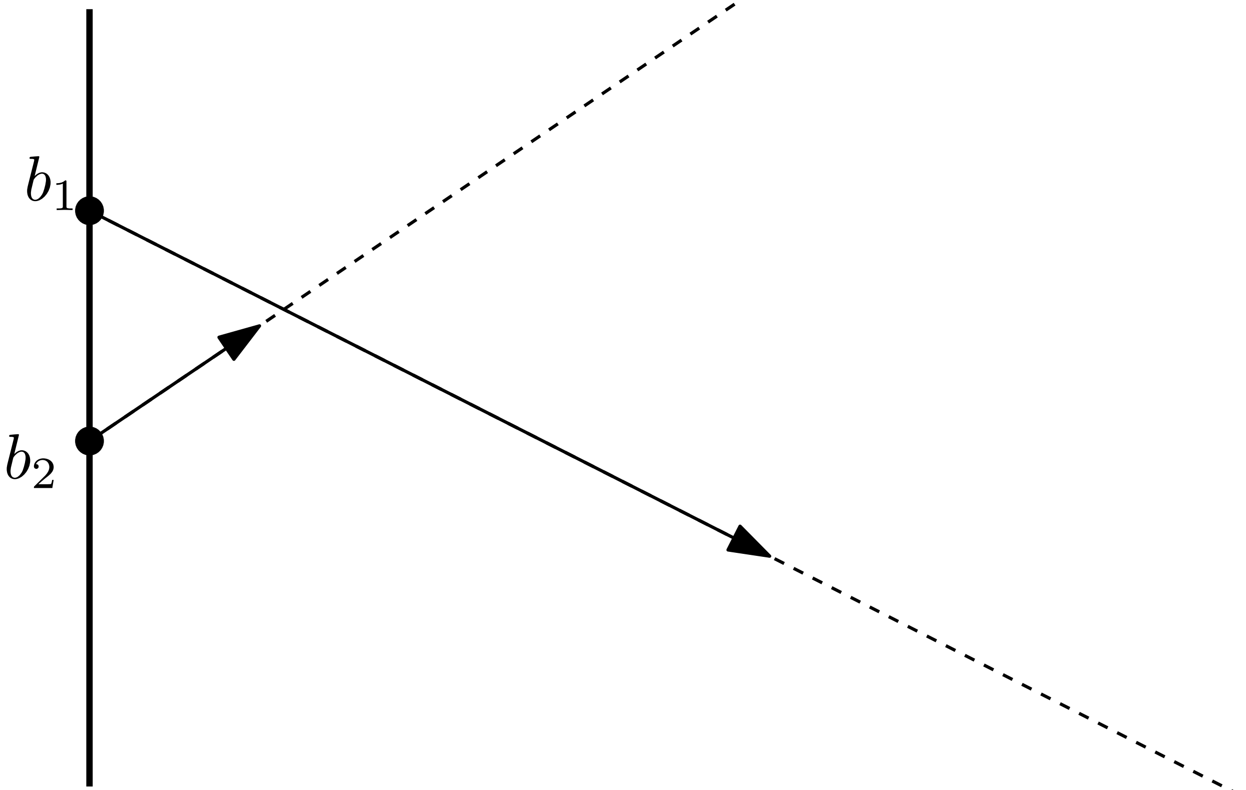
# Solving it with Linear Programming

What are the variables?

- Starting time $s_i$ of each biker
- Frustration tolerance $f$

One constraint for each pair $b_i, b_j$ with crossing paths.

What are the constraints?

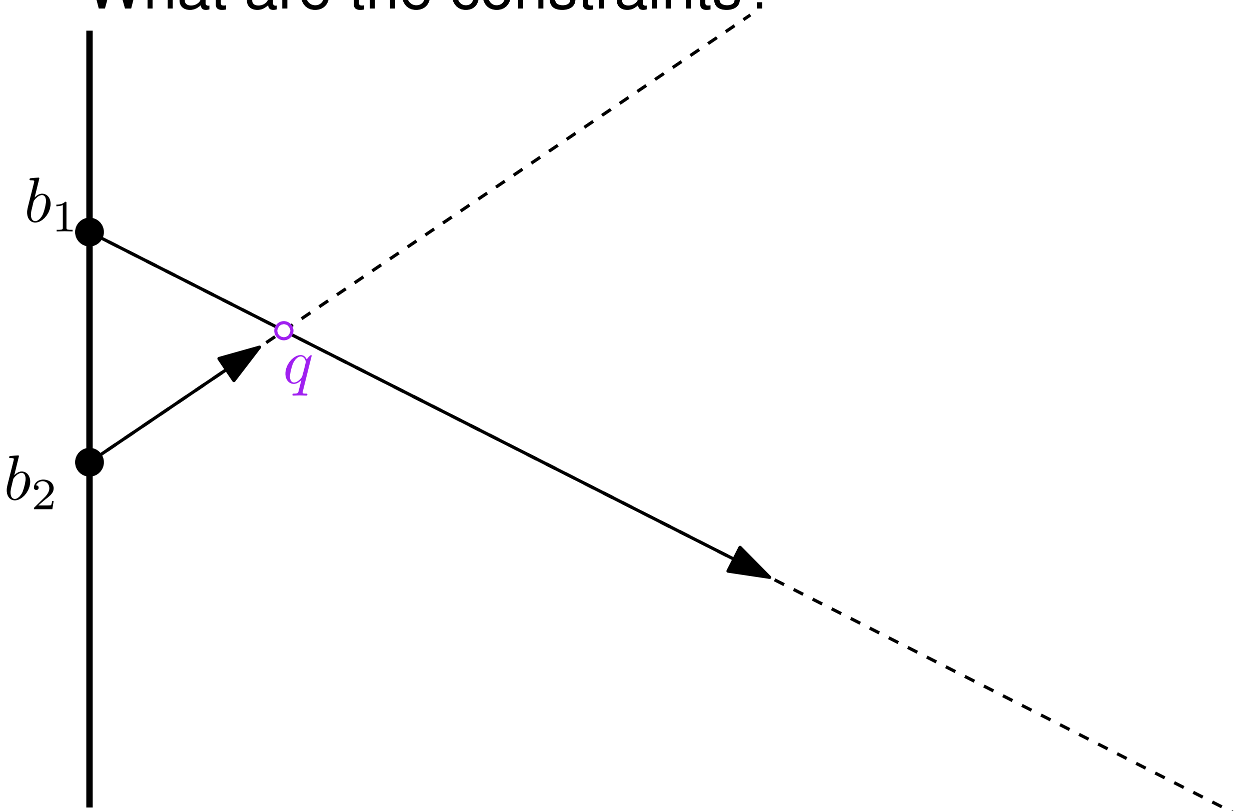


# Solving it with Linear Programming

What are the variables?

- Starting time $s_i$ of each biker
- Frustration tolerance $f$

What are the constraints?




One constraint for each pair $b_i, b_j$ with crossing paths.

# Solving it with Linear Programming

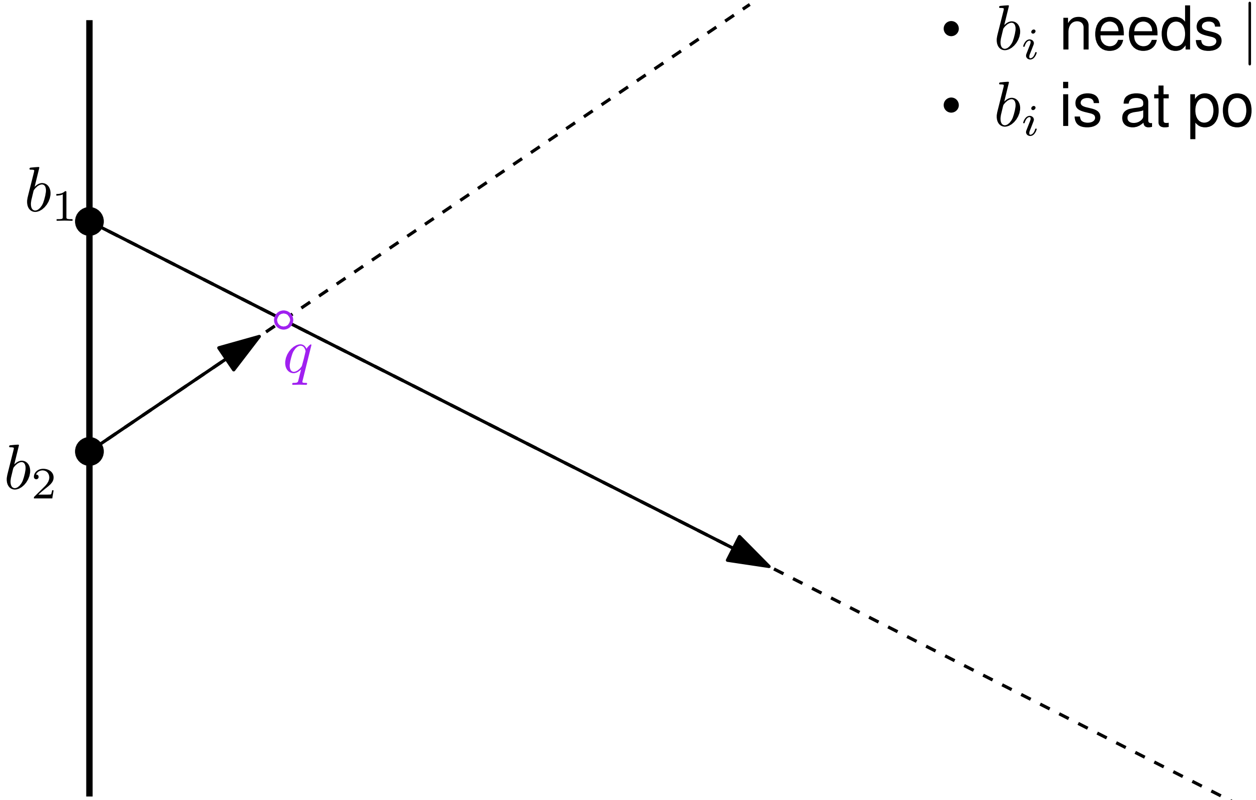
What are the variables?

- Starting time $s_i$ of each biker
- Frustration tolerance $f$

What are the constraints?

One constraint for each pair $b_i, b_j$ with crossing paths.

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

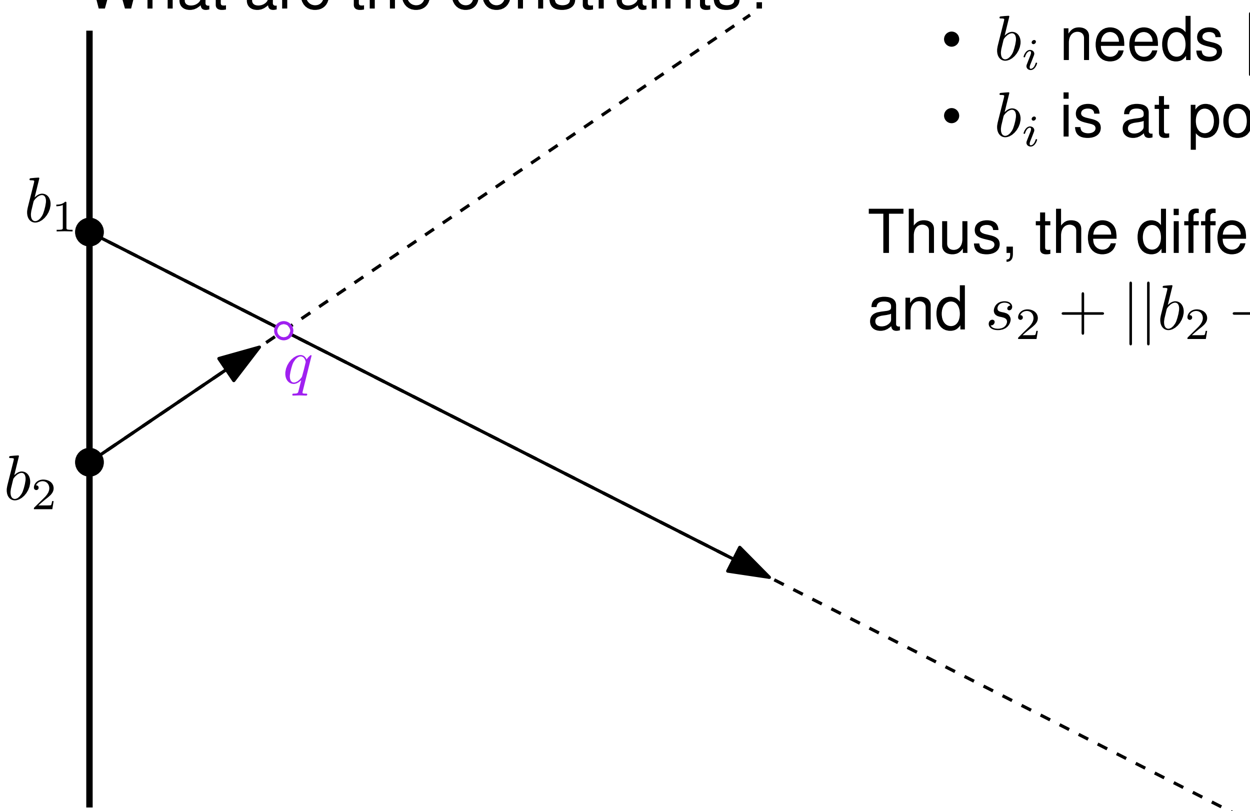


# Solving it with Linear Programming

What are the variables?

- Starting time $s_i$ of each biker
- Frustration tolerance $f$

What are the constraints?




One constraint for each pair $b_i, b_j$ with crossing paths.

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

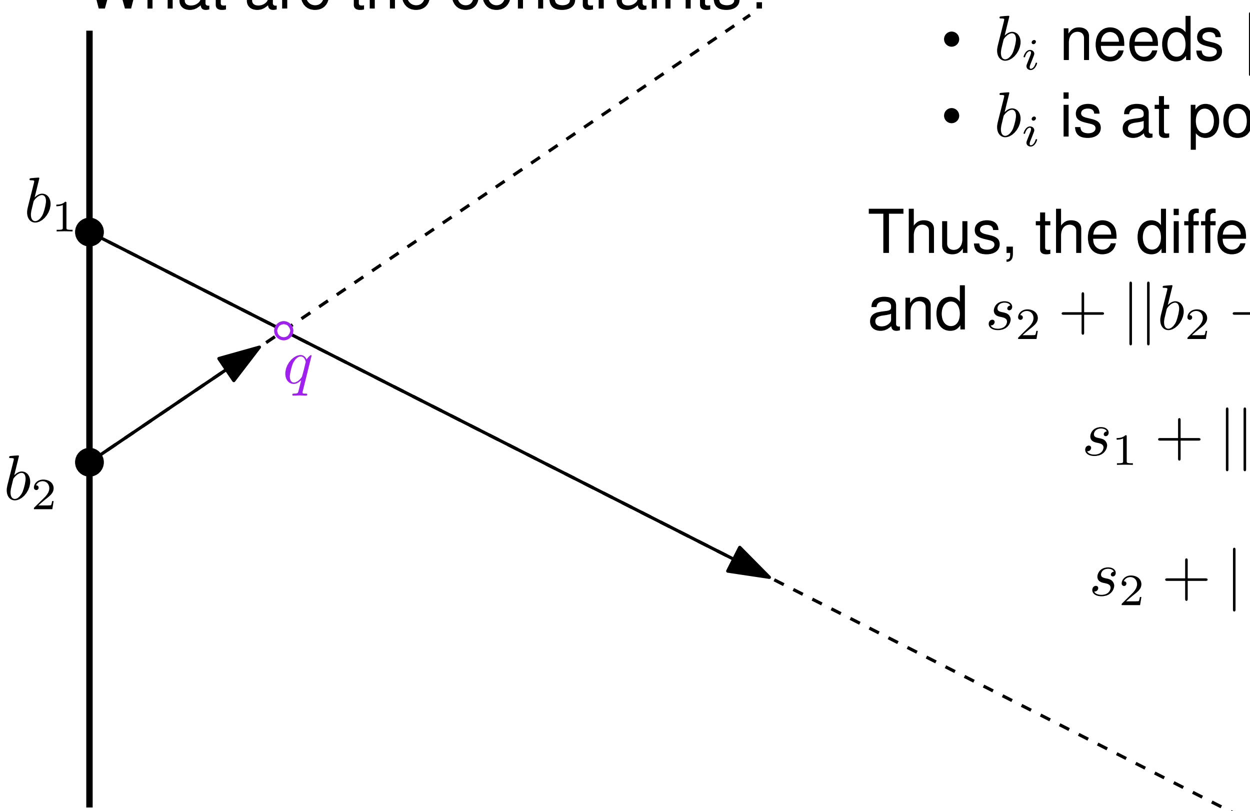
Thus, the difference between $s_1 + ||b_1 - q||$ and $s_2 + ||b_2 - q||$ can be at most $f$.

# Solving it with Linear Programming

What are the variables?

- Starting time $s_i$ of each biker
- Frustration tolerance $f$

What are the constraints?




One constraint for each pair $b_i, b_j$ with crossing paths.

- $b_i$ 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 $f$.

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

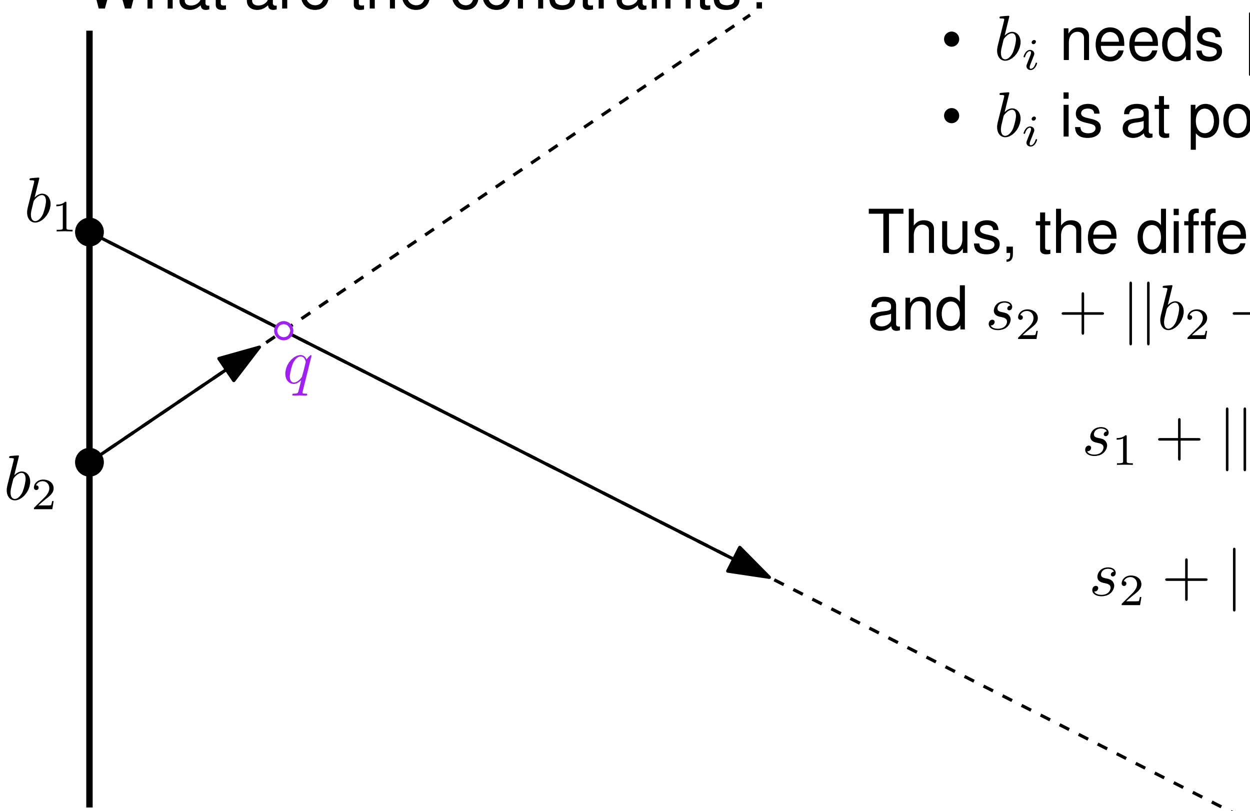
$$s_2 + ||b_2 - q|| \leq s_1 + ||b_1 - q|| + f$$

# Solving it with Linear Programming

What are the variables?

- Starting time $s_i$ of each biker
- Frustration tolerance $f$

What are the constraints?




One constraint for each pair $b_i, b_j$ with crossing paths.

- $b_i$ 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 $f$.

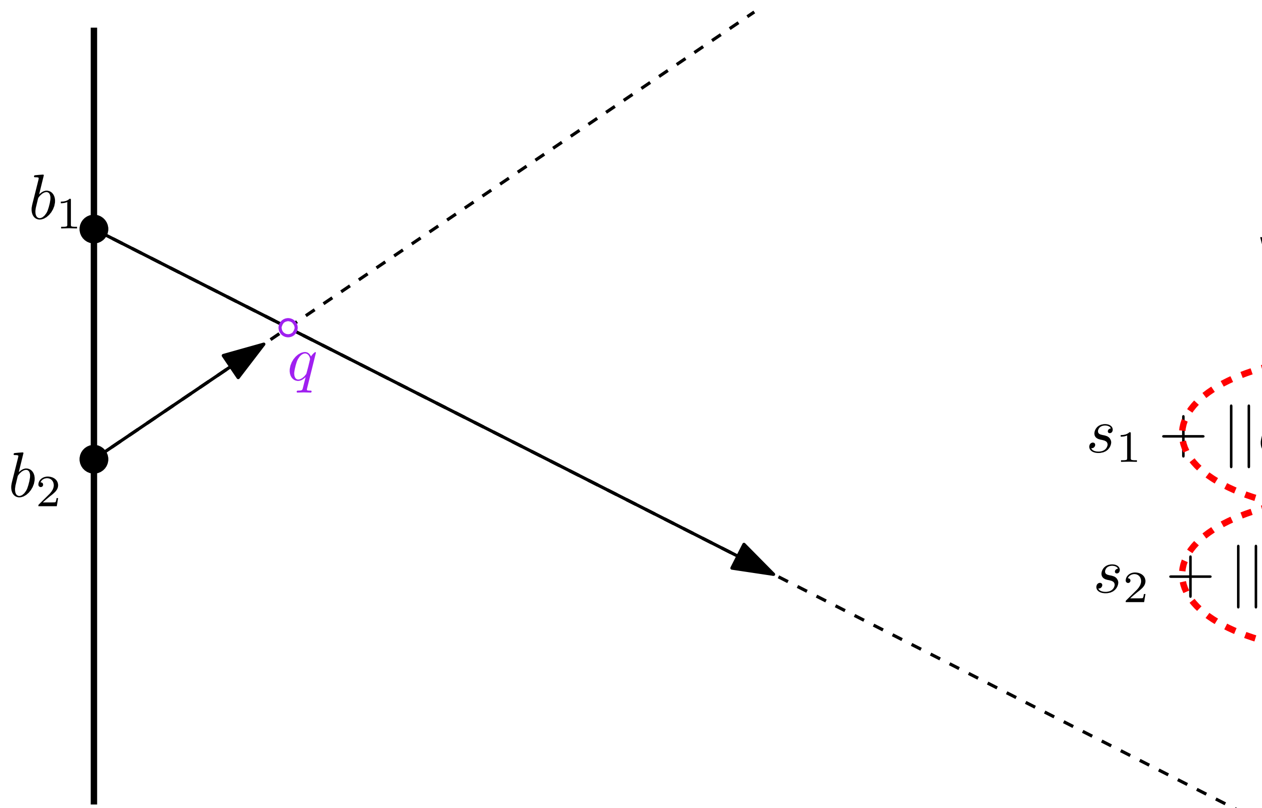
$$s_1 + ||b_1 - q|| \leq s_2 + ||b_2 - q|| + f,$$

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

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

# Solving it with Linear Programming

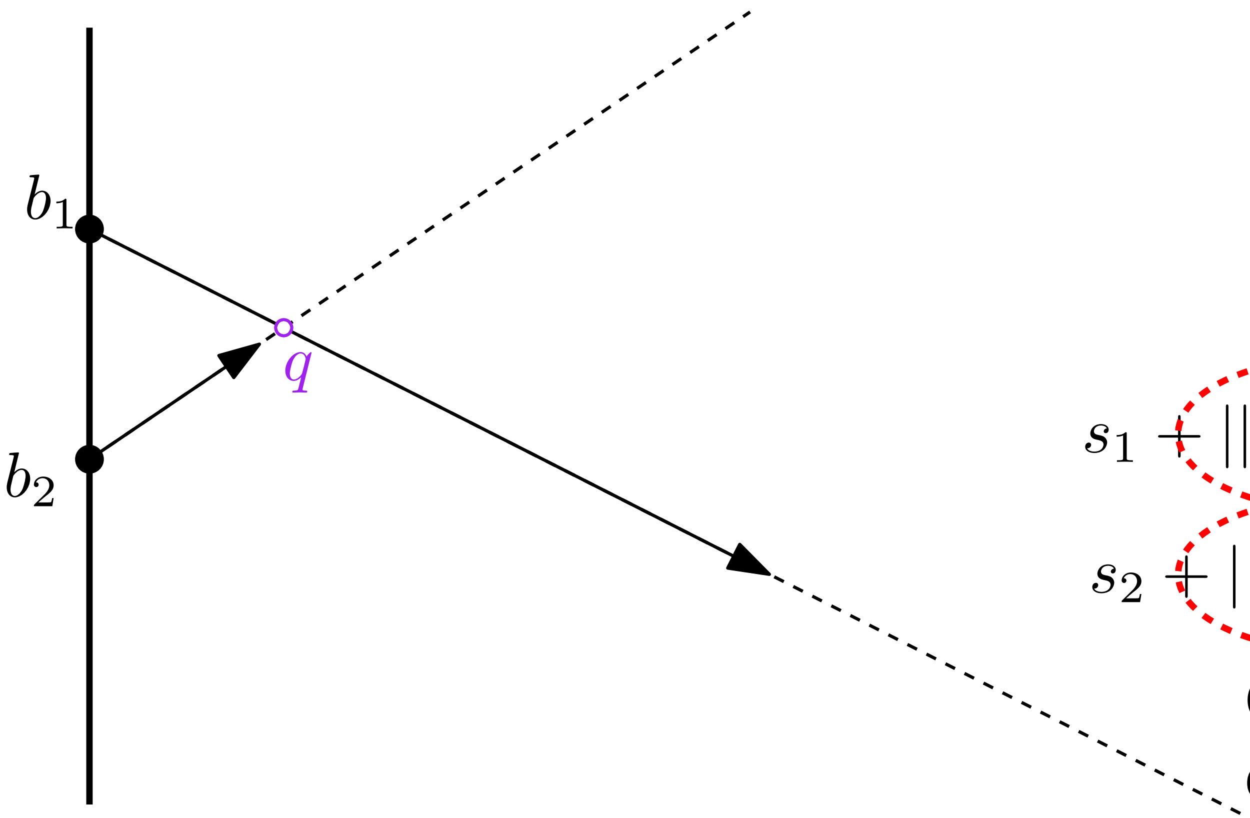
One constraint for each pair $b_i, b_j$ with crossing paths.




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




One constraint for each pair $b_i, b_j$ with crossing paths.

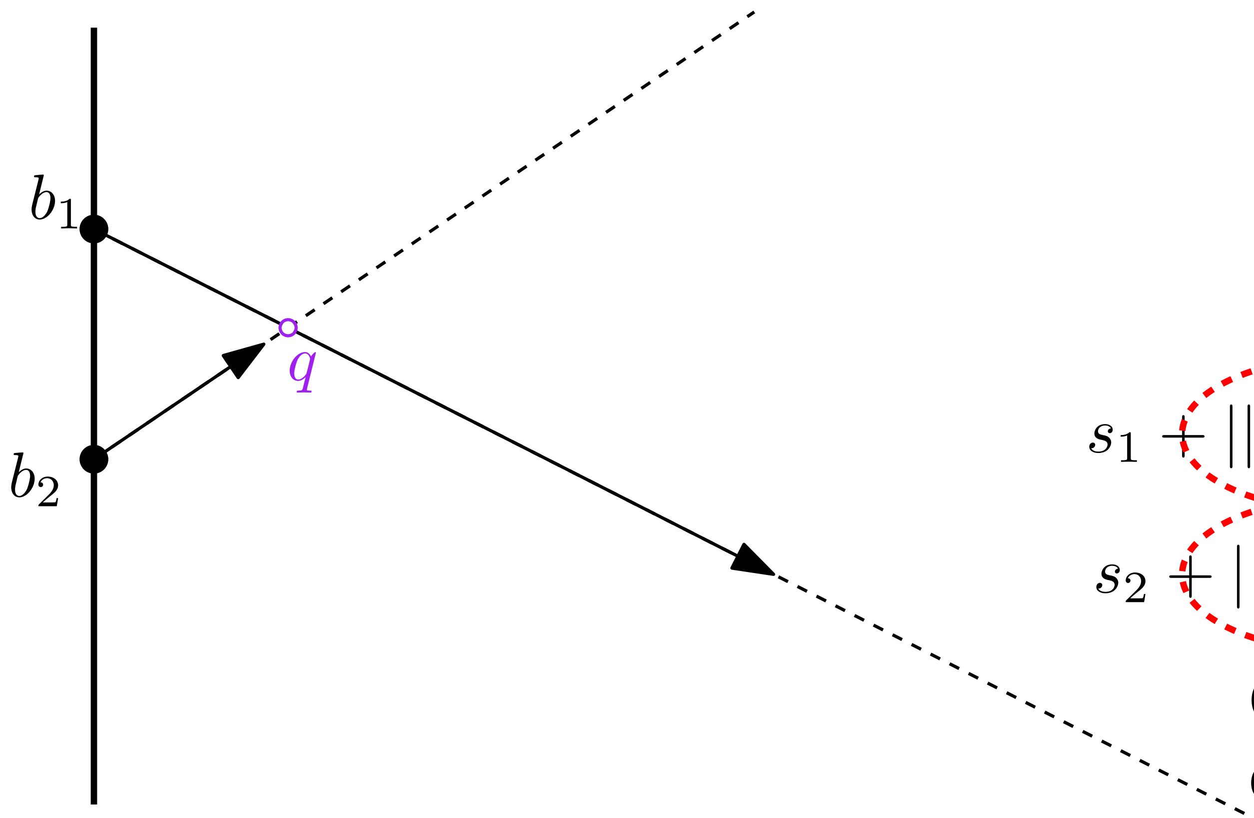
What are these terms?

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

One needs square roots to compute these constants.

# Solving it with Linear Programming

In addition, all variables are non-negative.




One constraint for each pair $b_i, b_j$ with crossing paths.

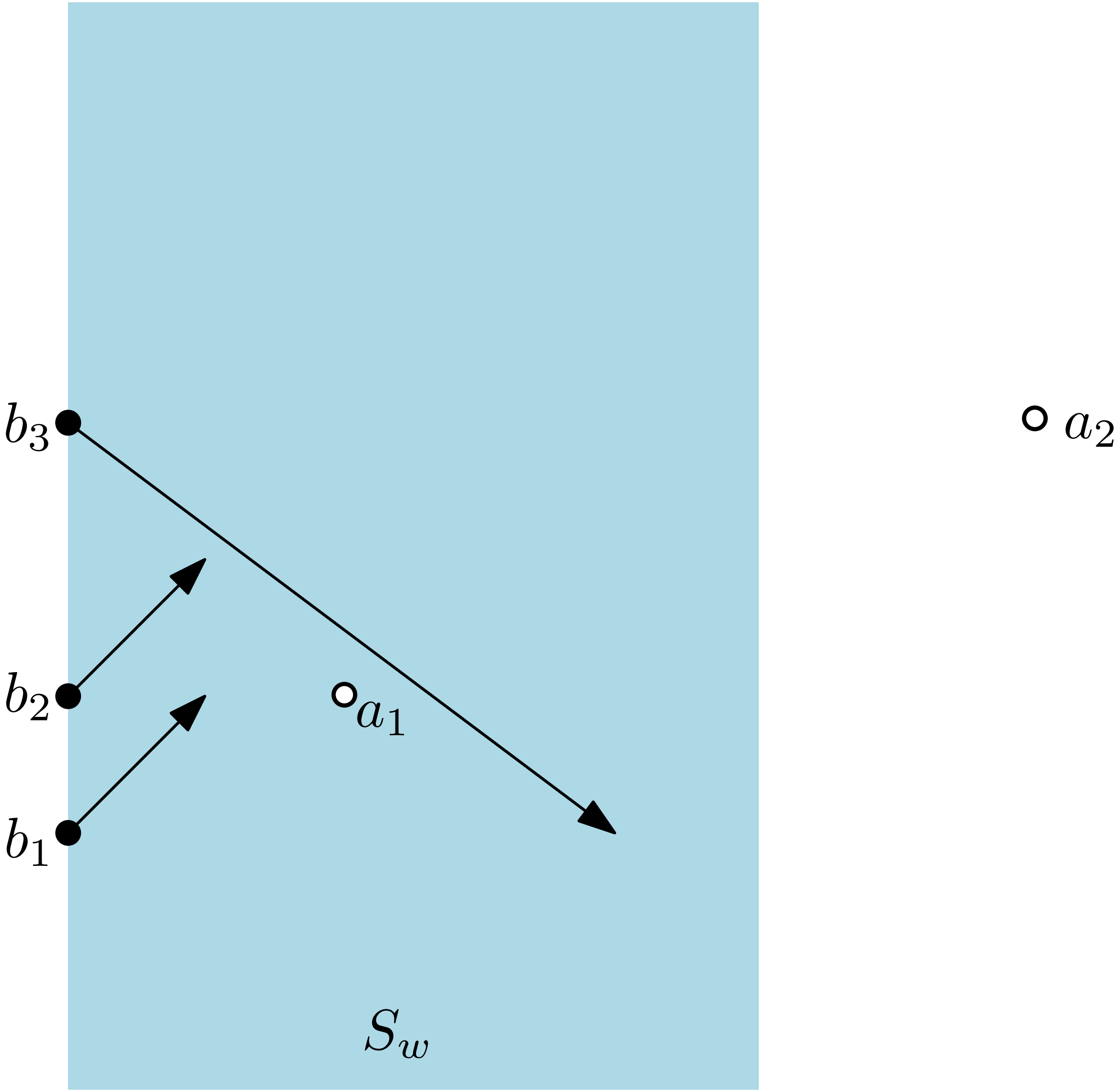
What are these terms?

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

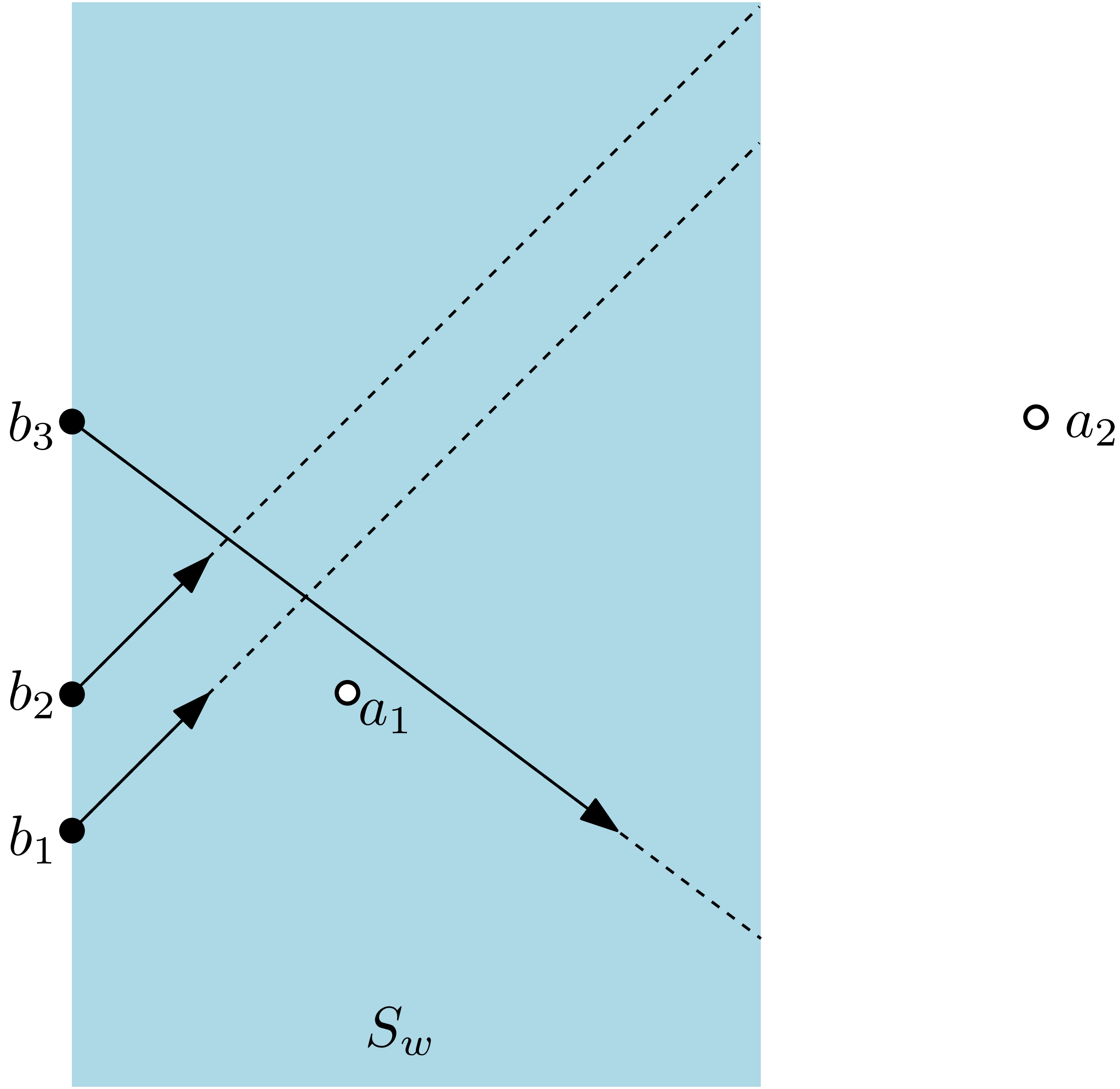
$$s_2 + ||b_2 - q|| \leq s_1 + ||b_1 - q|| + f$$

One needs square roots to compute these constants.

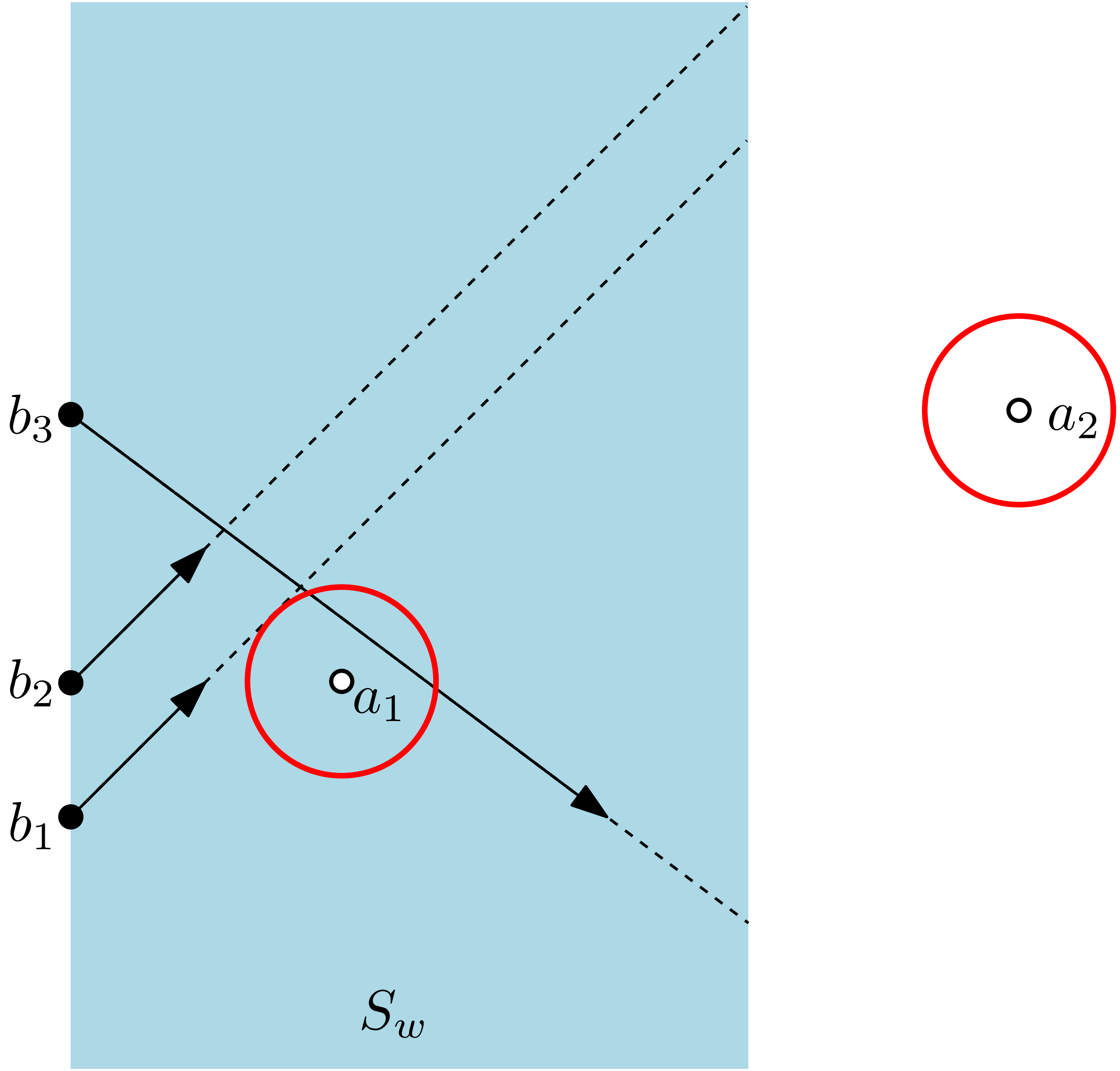
# Beethoven



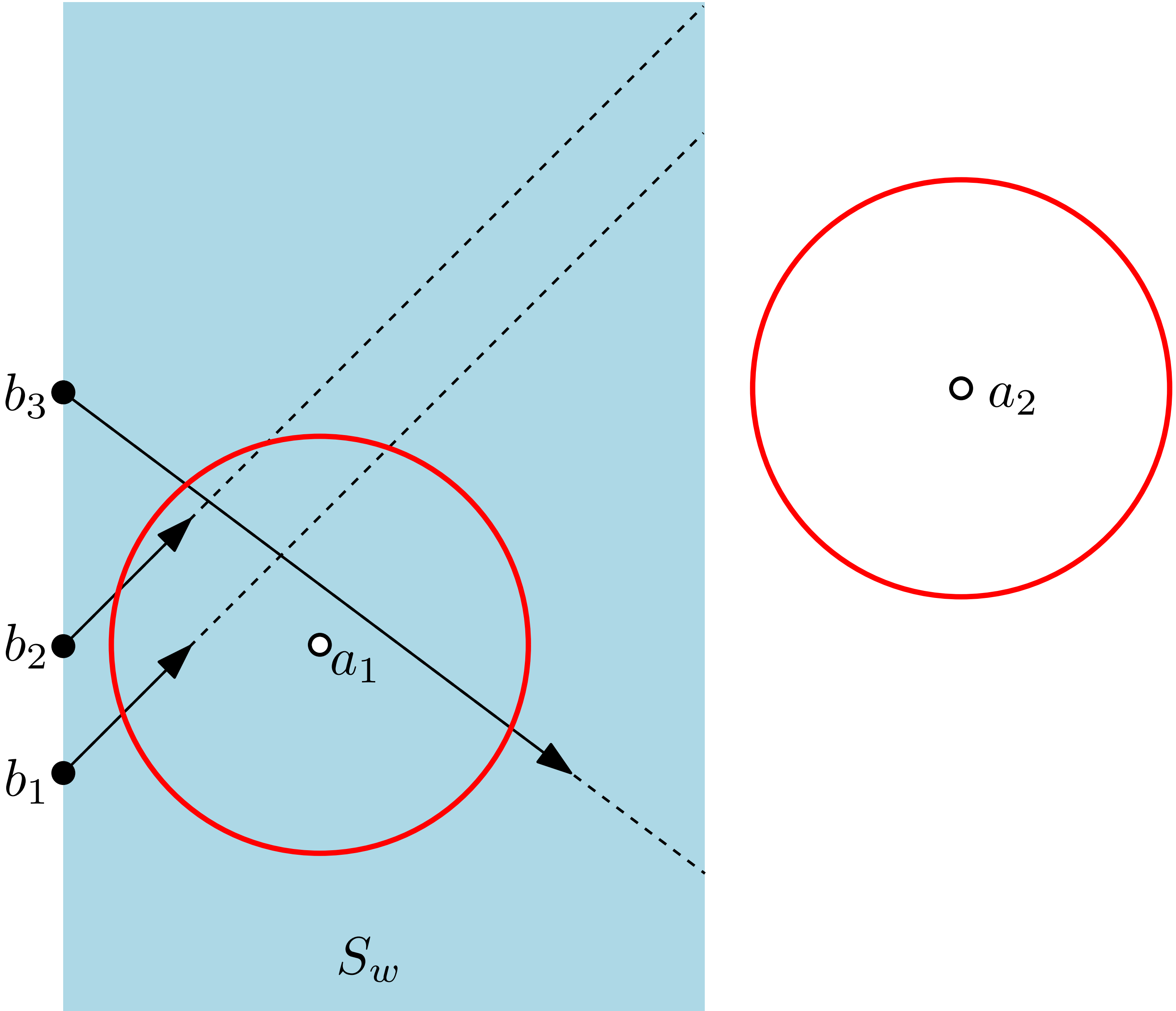
# Beethoven



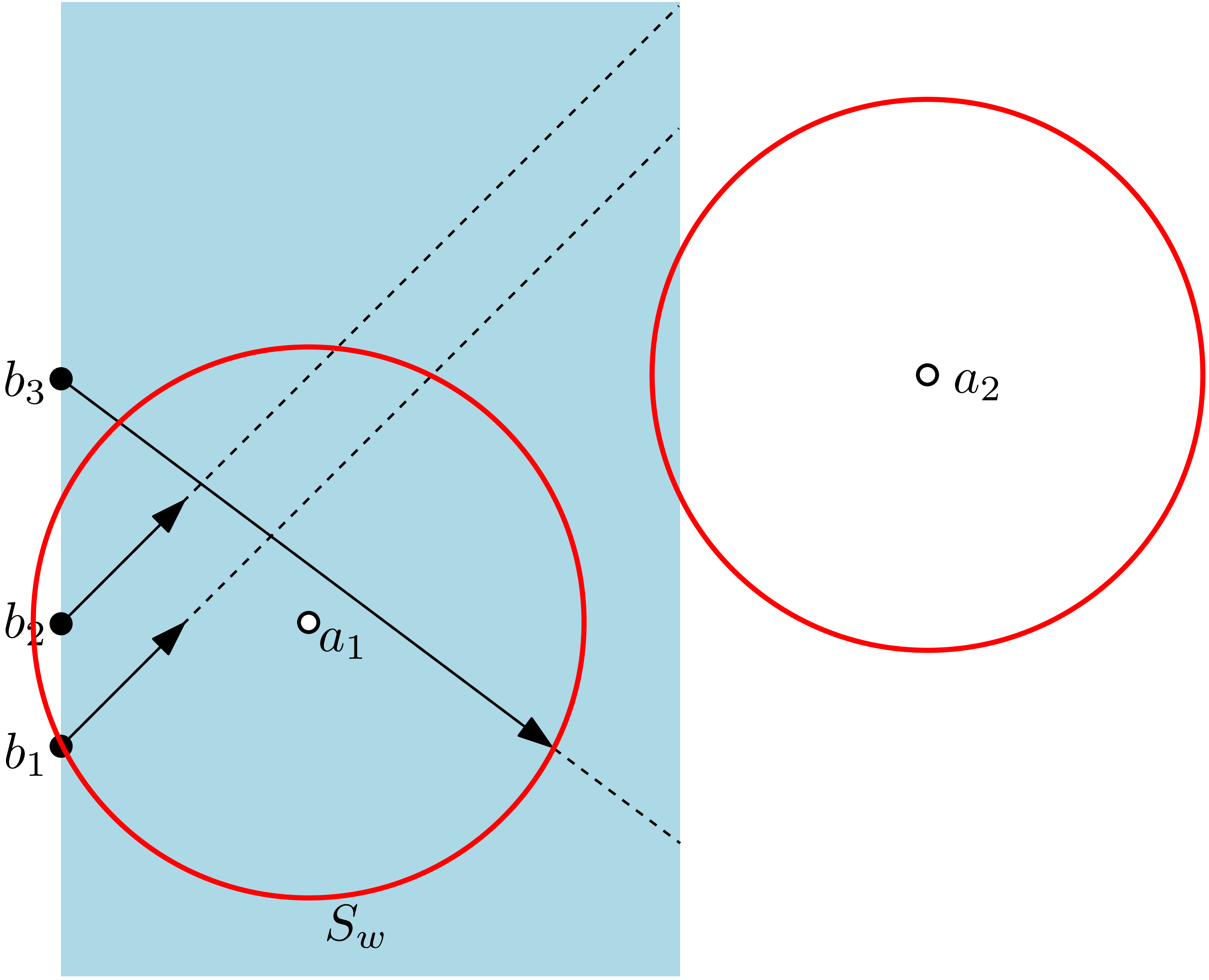
# Beethoven



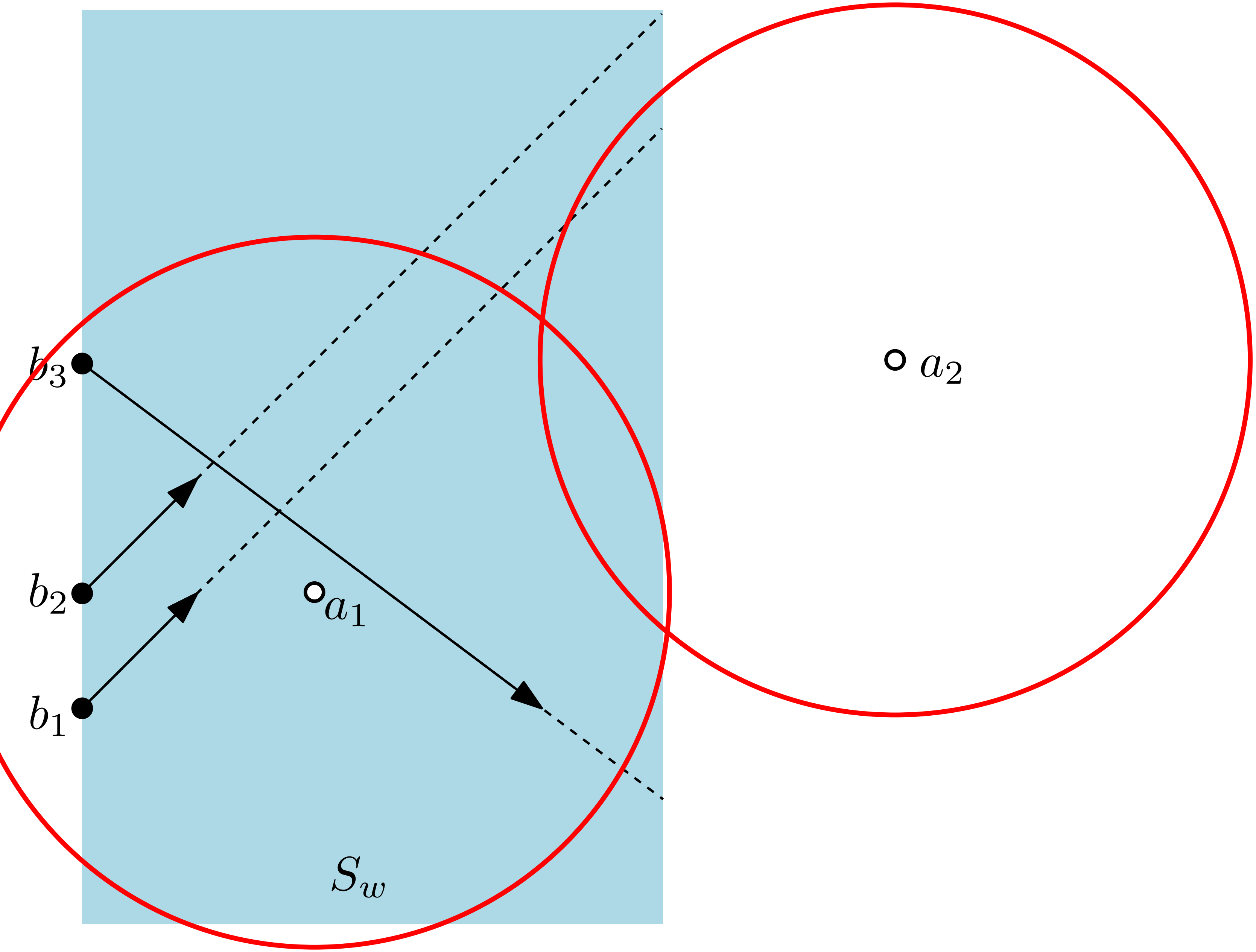
# Beethoven



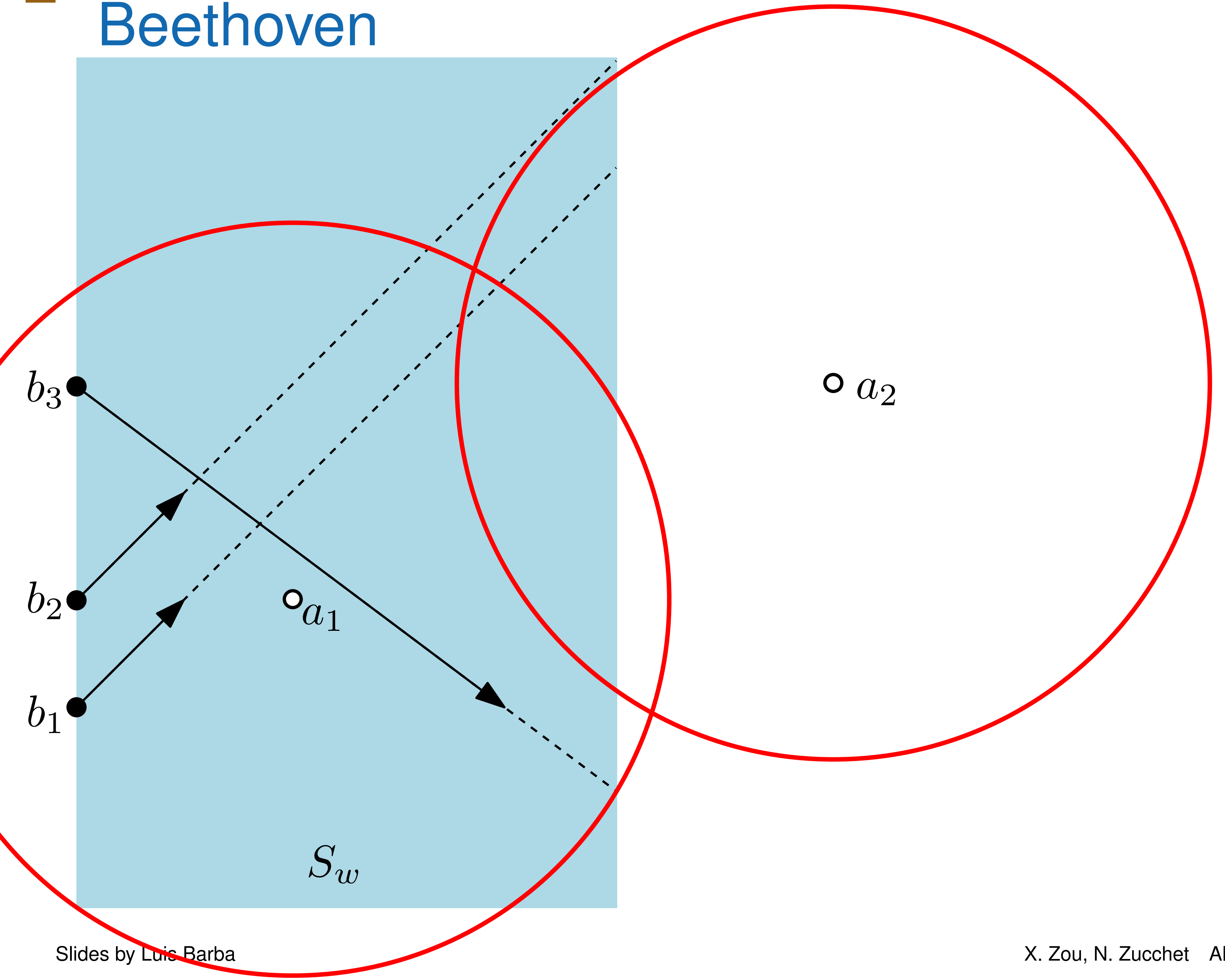
# Beethoven



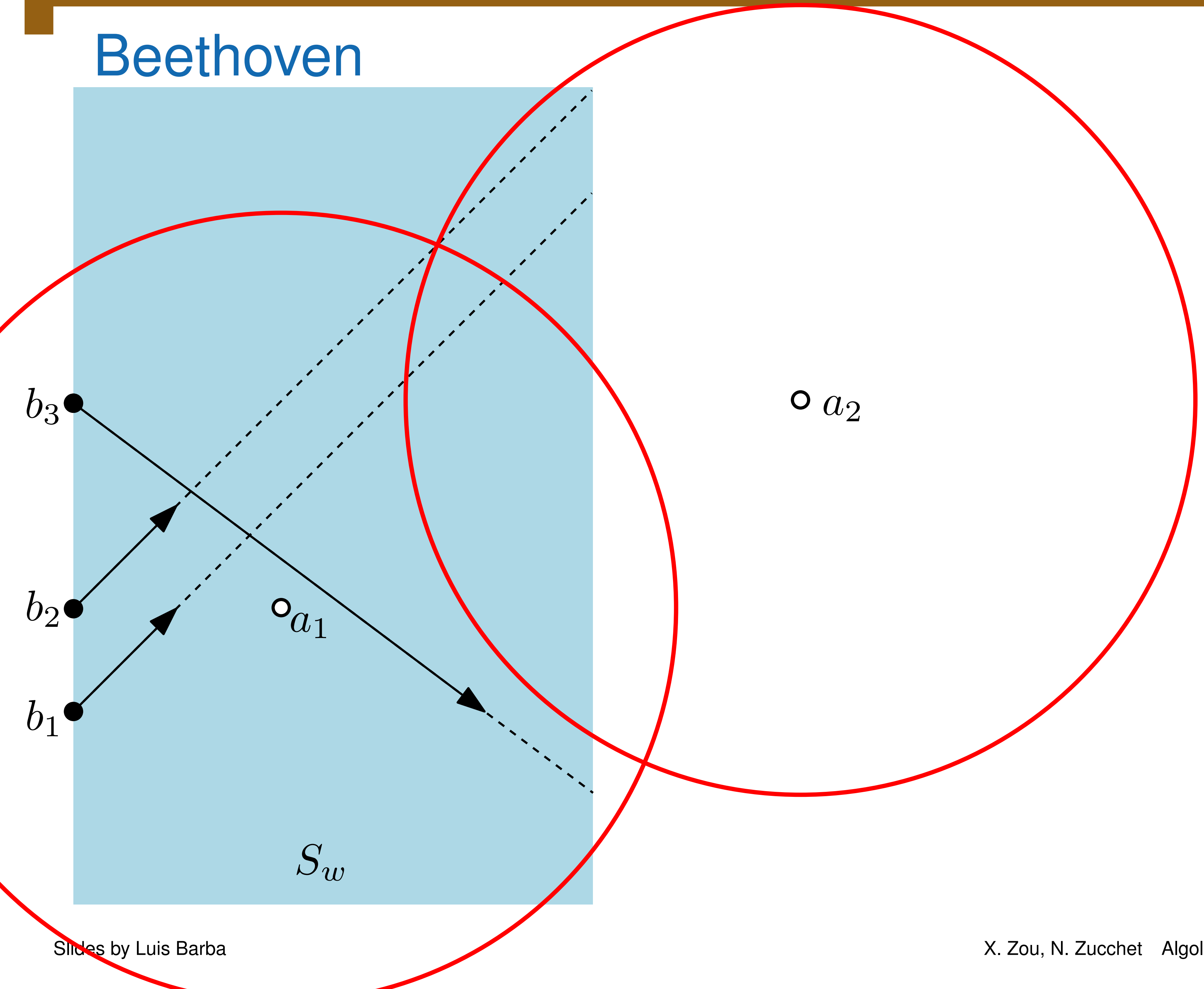
# Beethoven



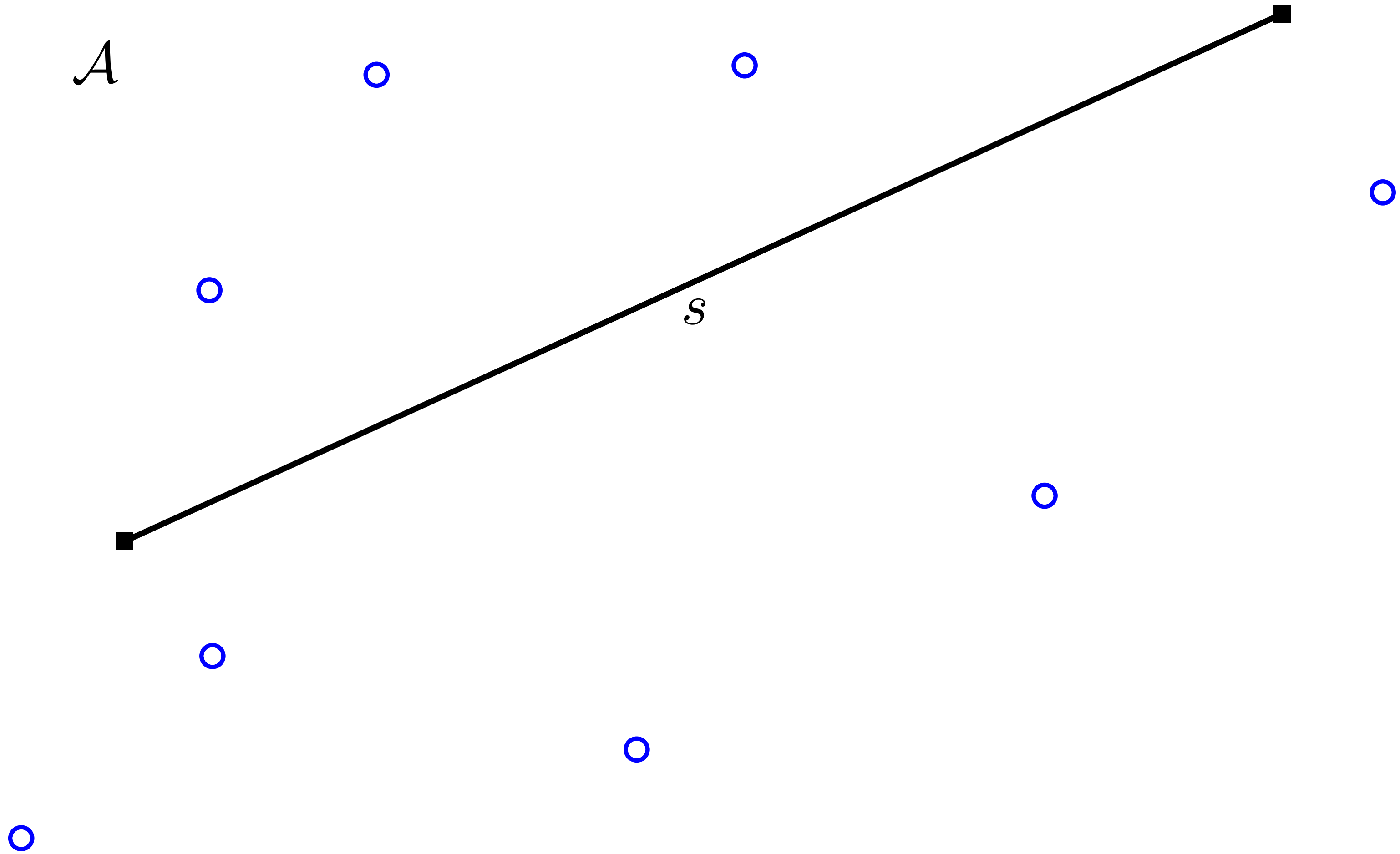
# Beethoven



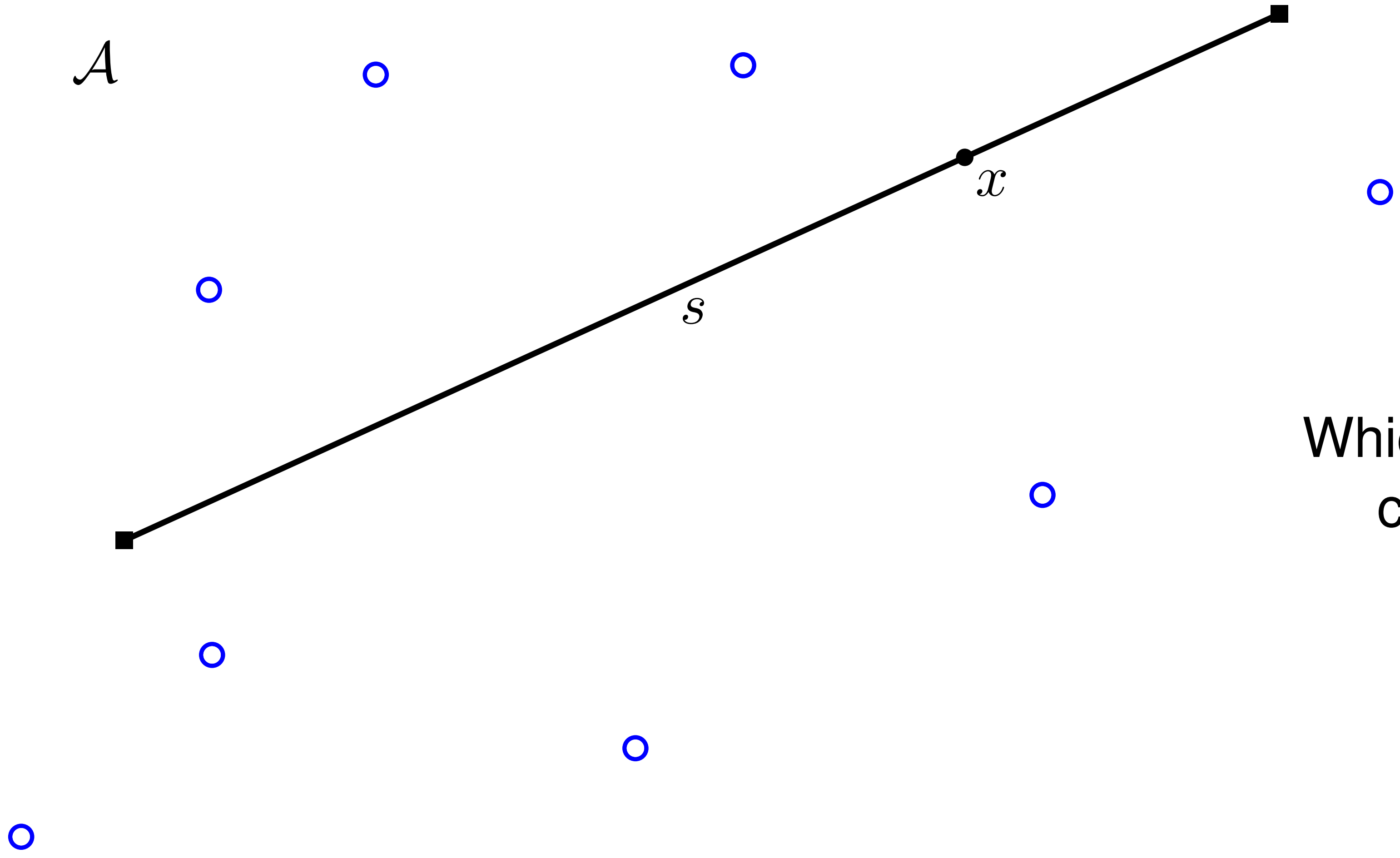
# Beethoven



# Working with a single segment

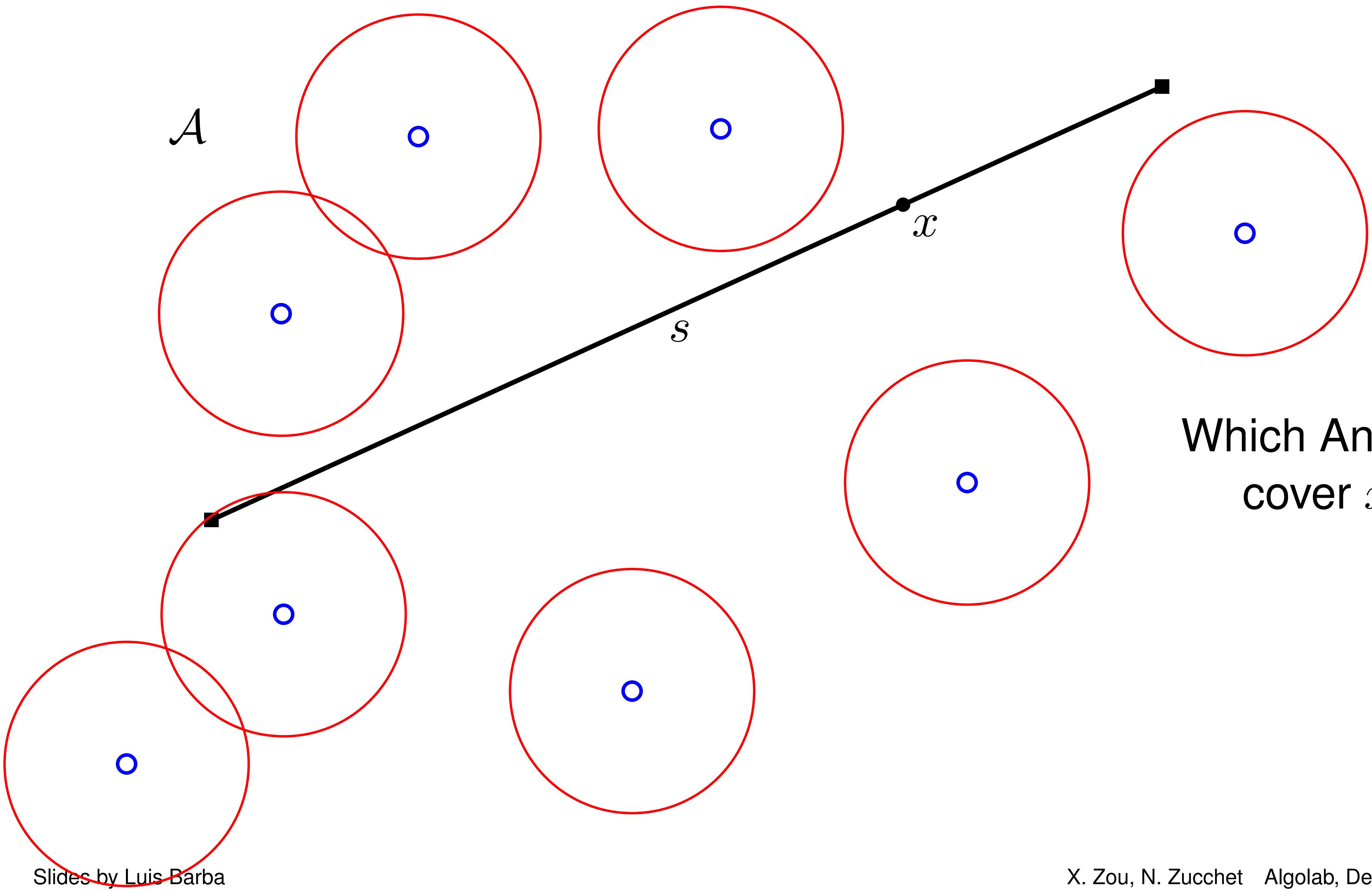


# Working with a single segment



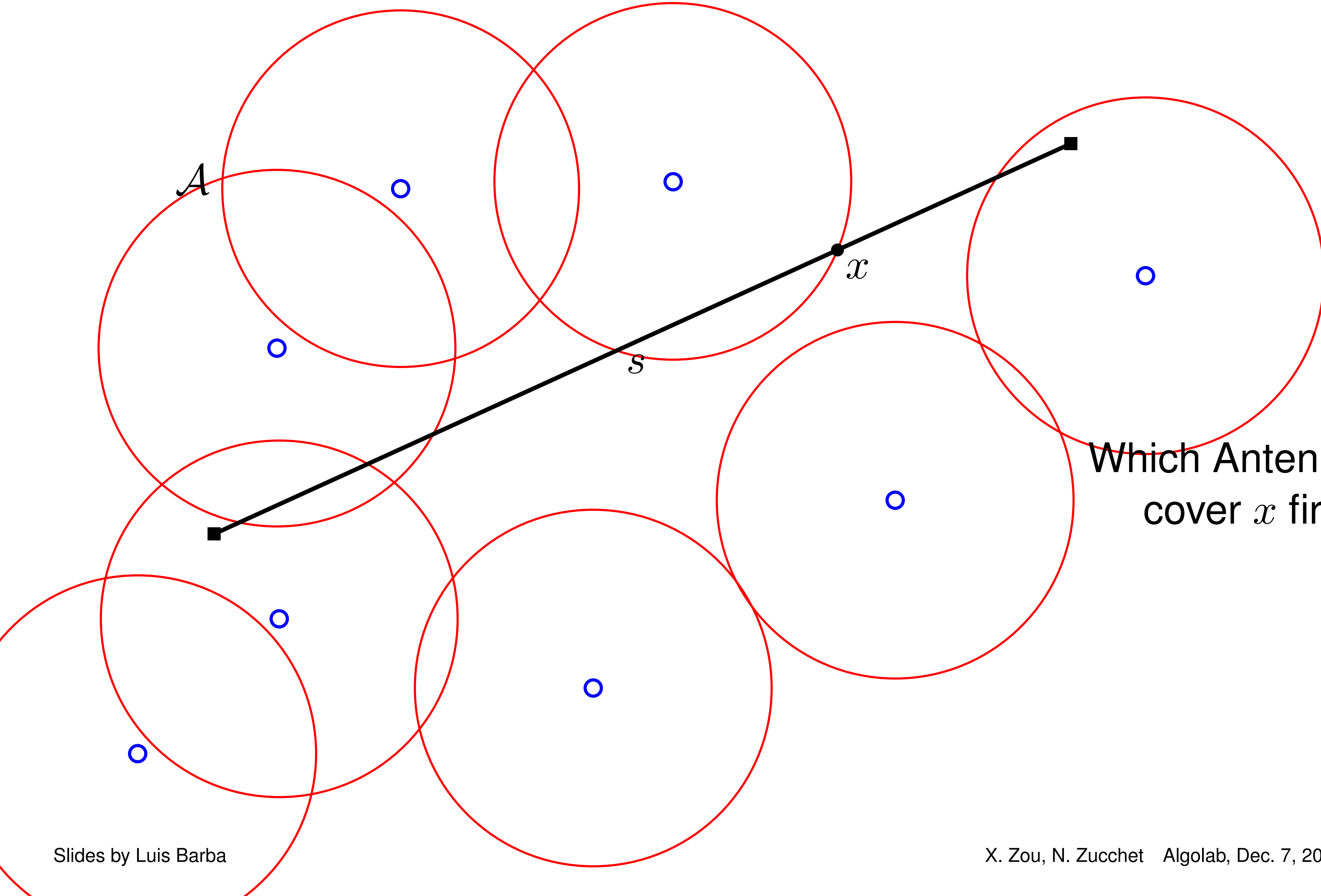

Which Antenna will cover $x$ first?

# Working with a single segment



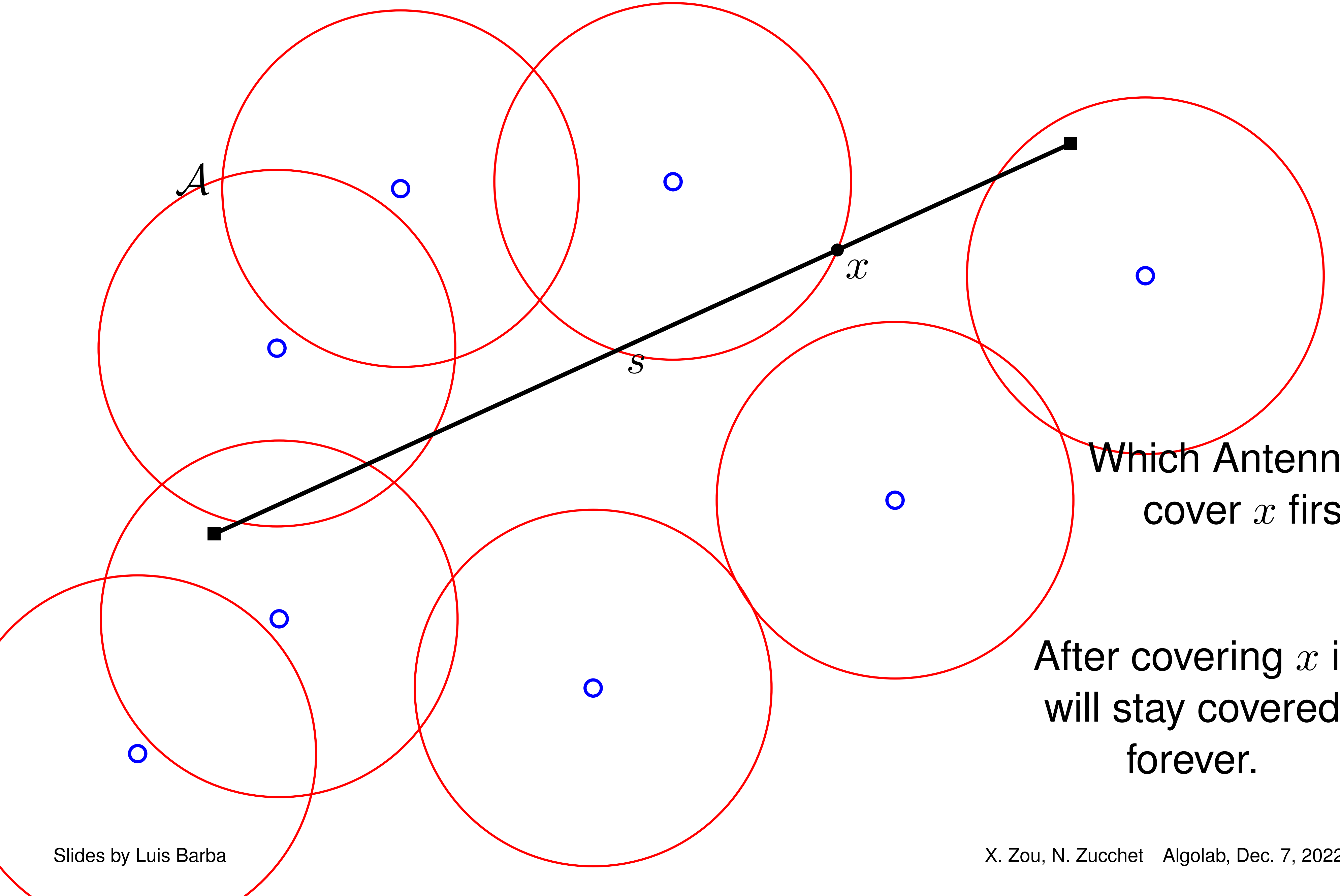

Which Antenna will cover $x$ first?

# Working with a single segment




Which Antenna will cover $x$ first?

# Working with a single segment

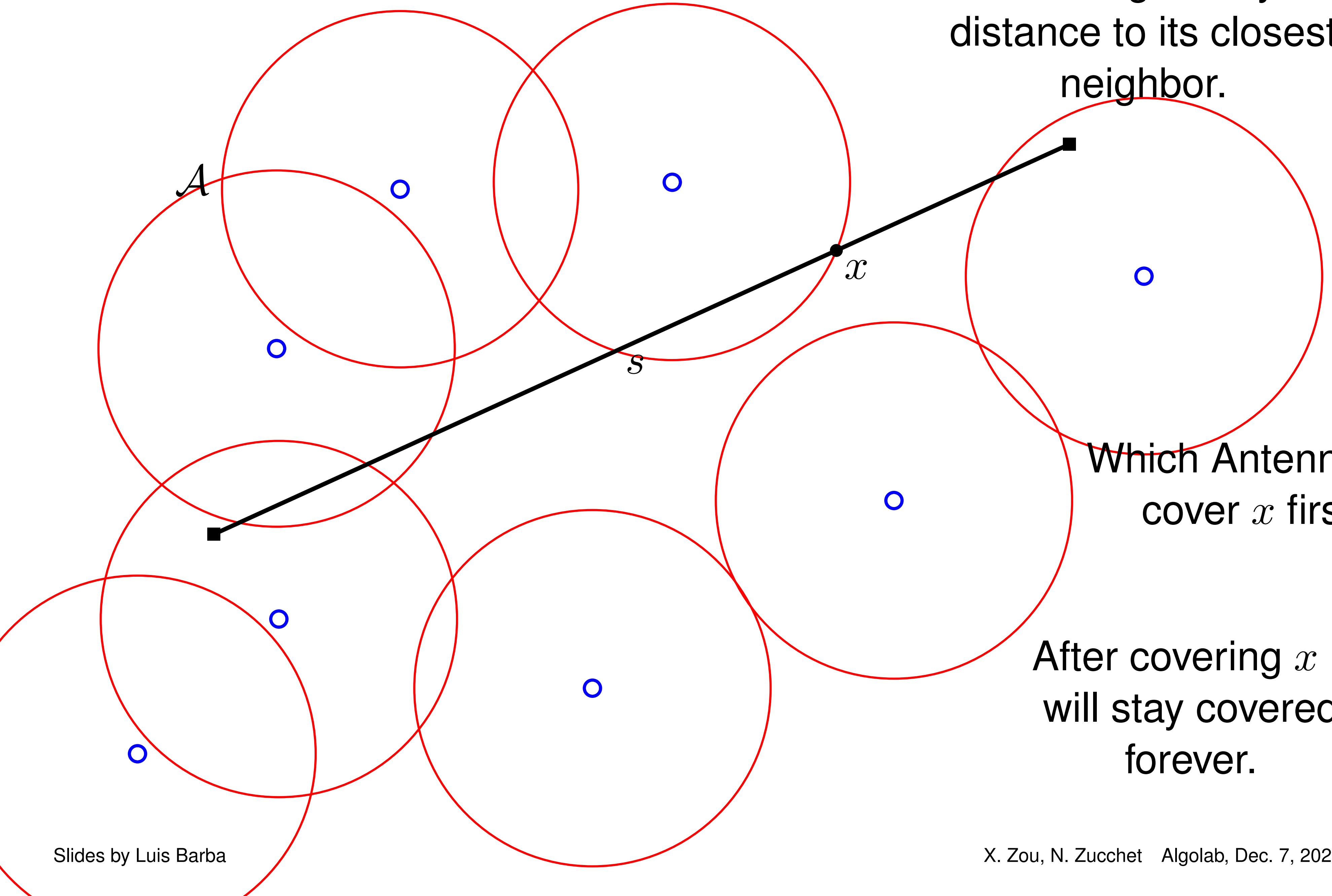



Which Antenna will cover $x$ first?

After covering $x$ it will stay covered forever.

# Working with a single segment

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

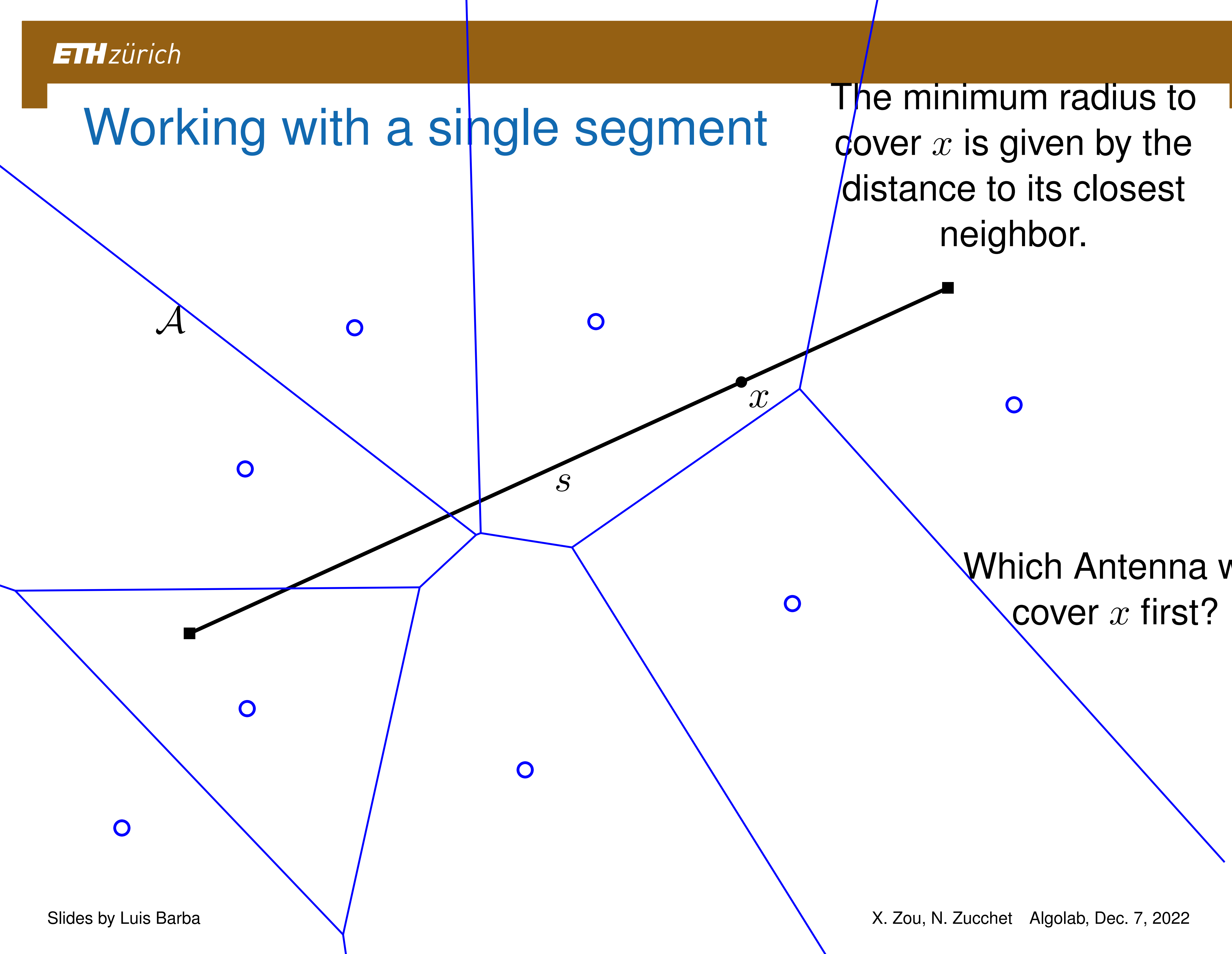



Which Antenna will cover $x$ first?

After covering $x$ it will stay covered forever.

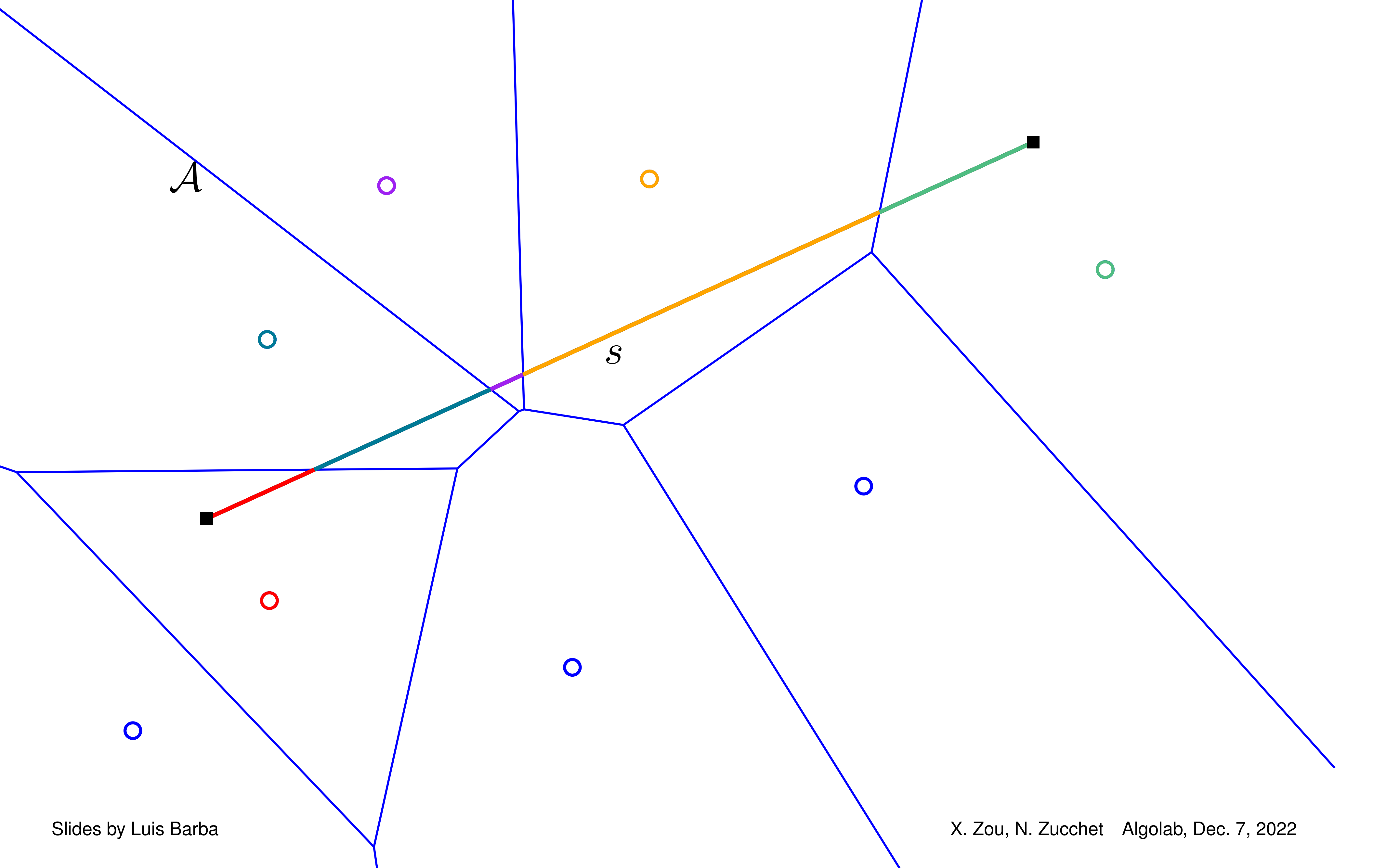
# Working with a single segment

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



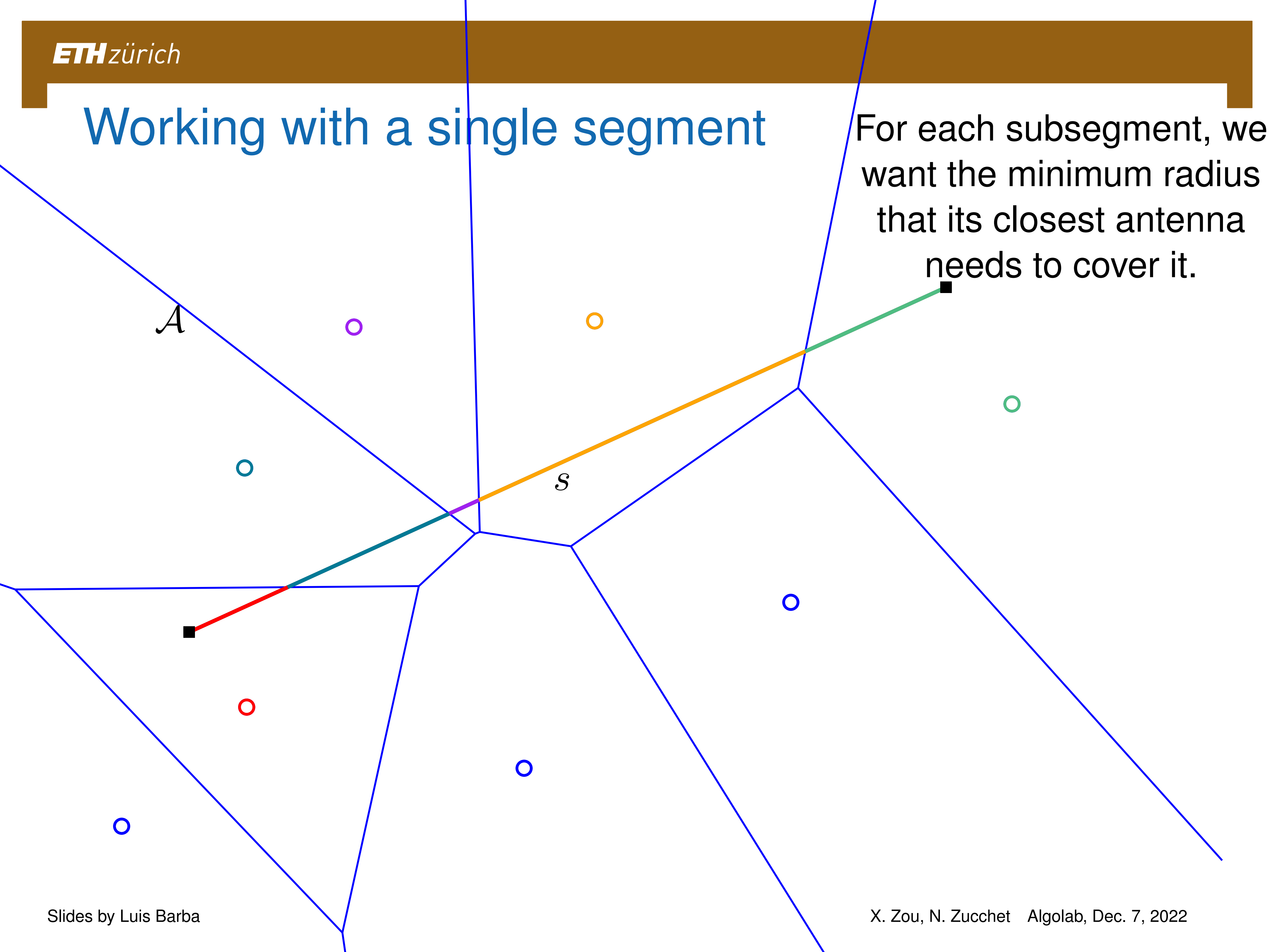

Which Antenna will cover $x$ first?

# Working with a single segment



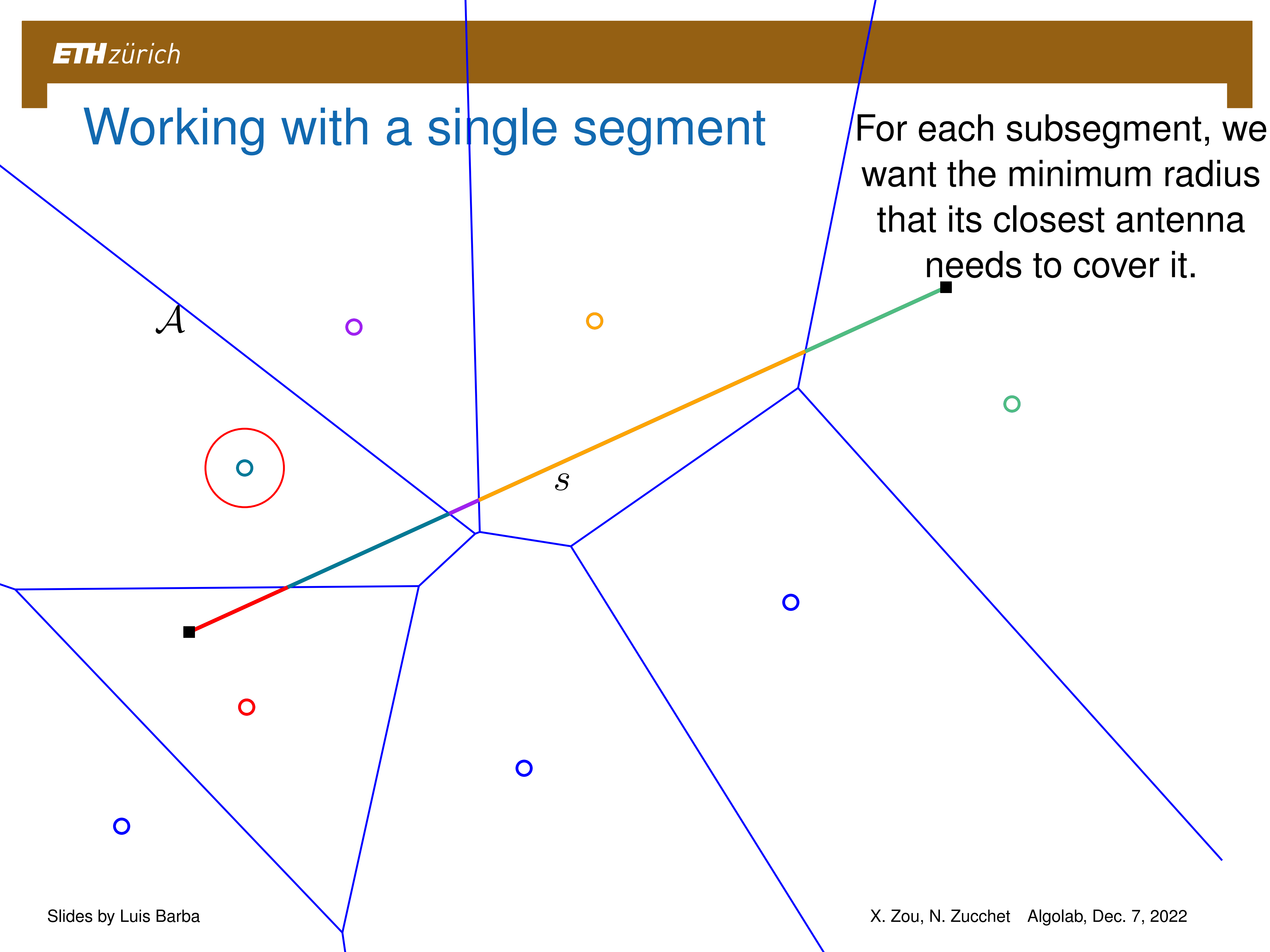
# Working with a single segment

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



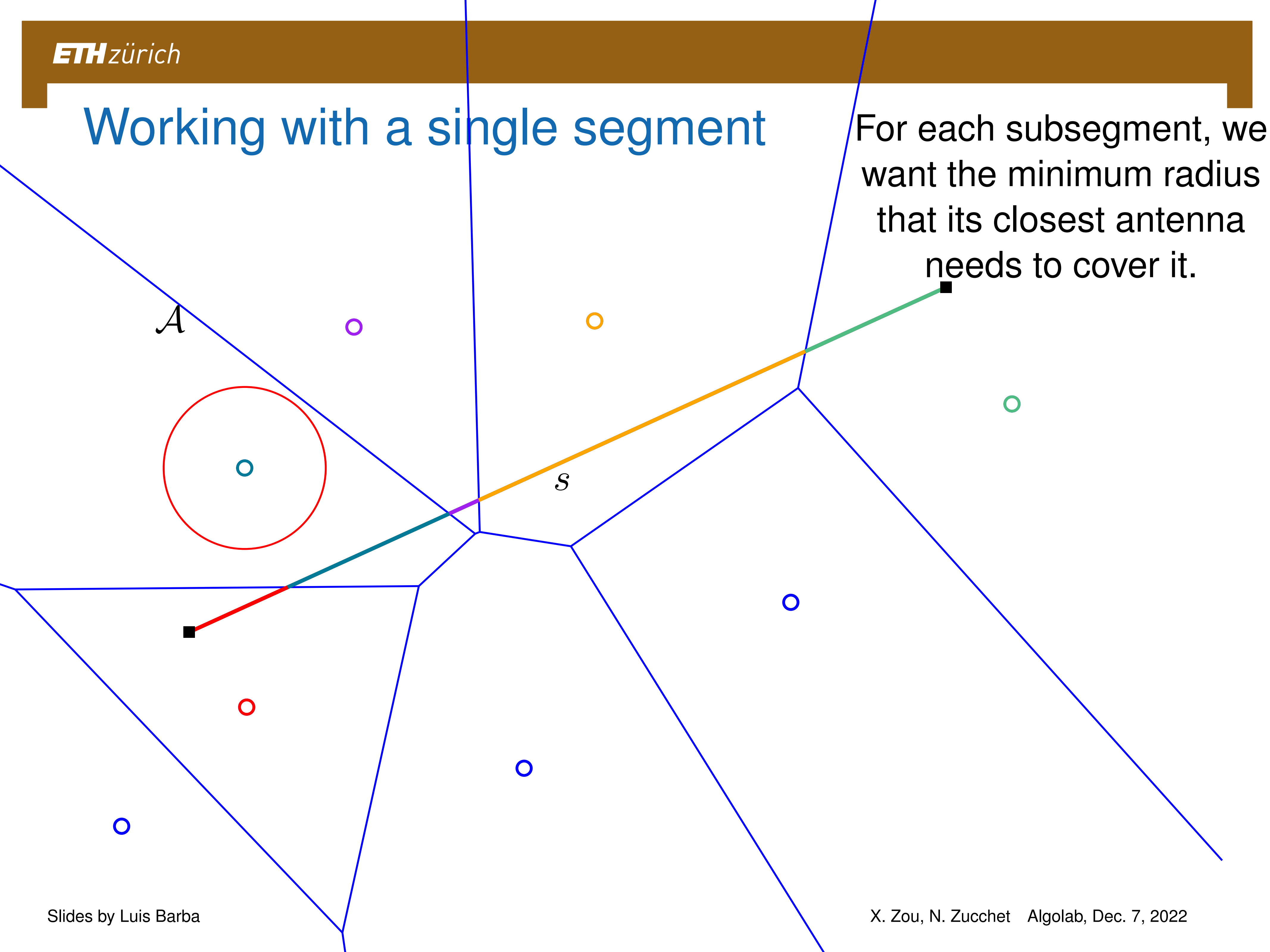
# Working with a single segment

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



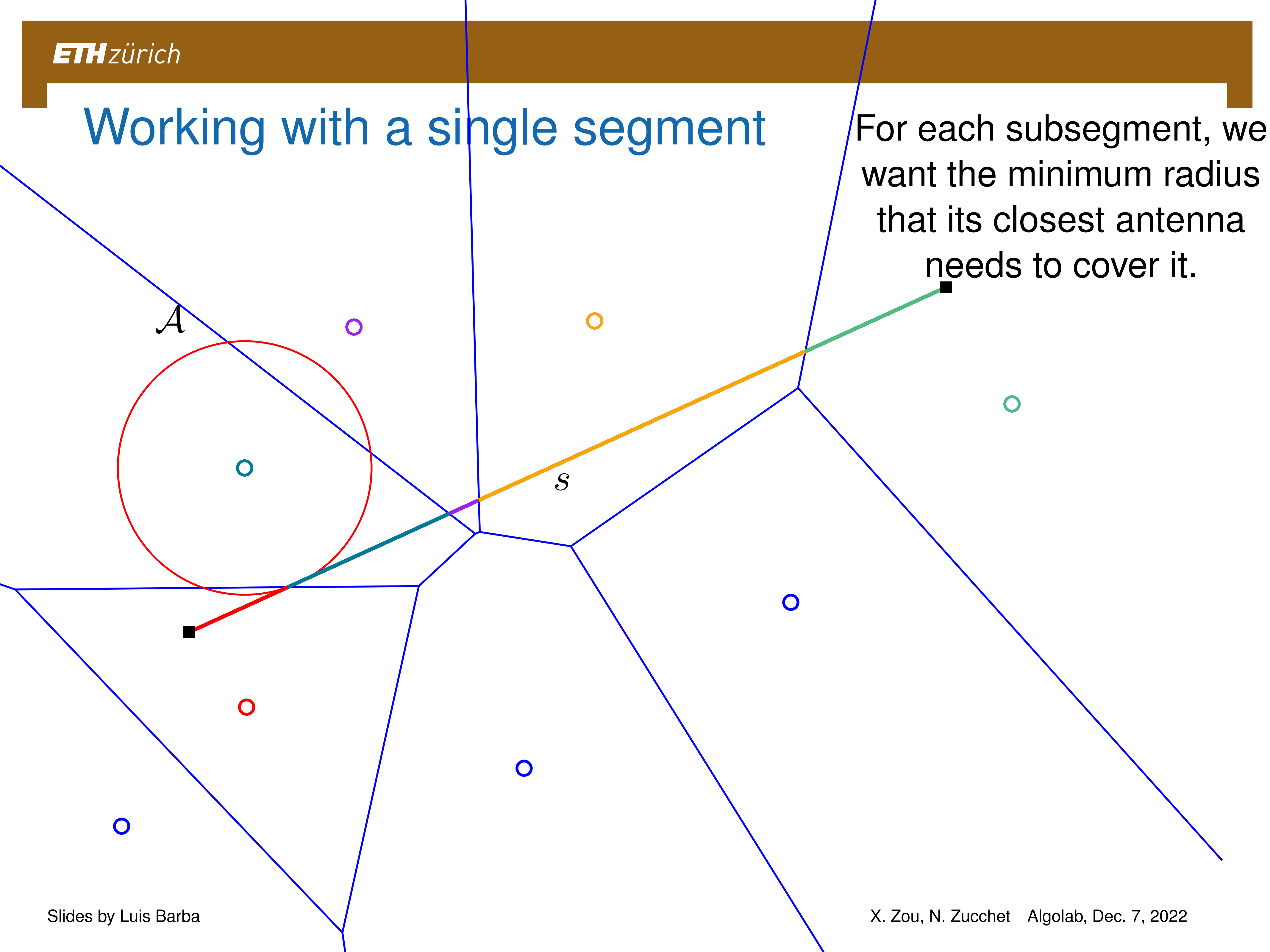
# Working with a single segment

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



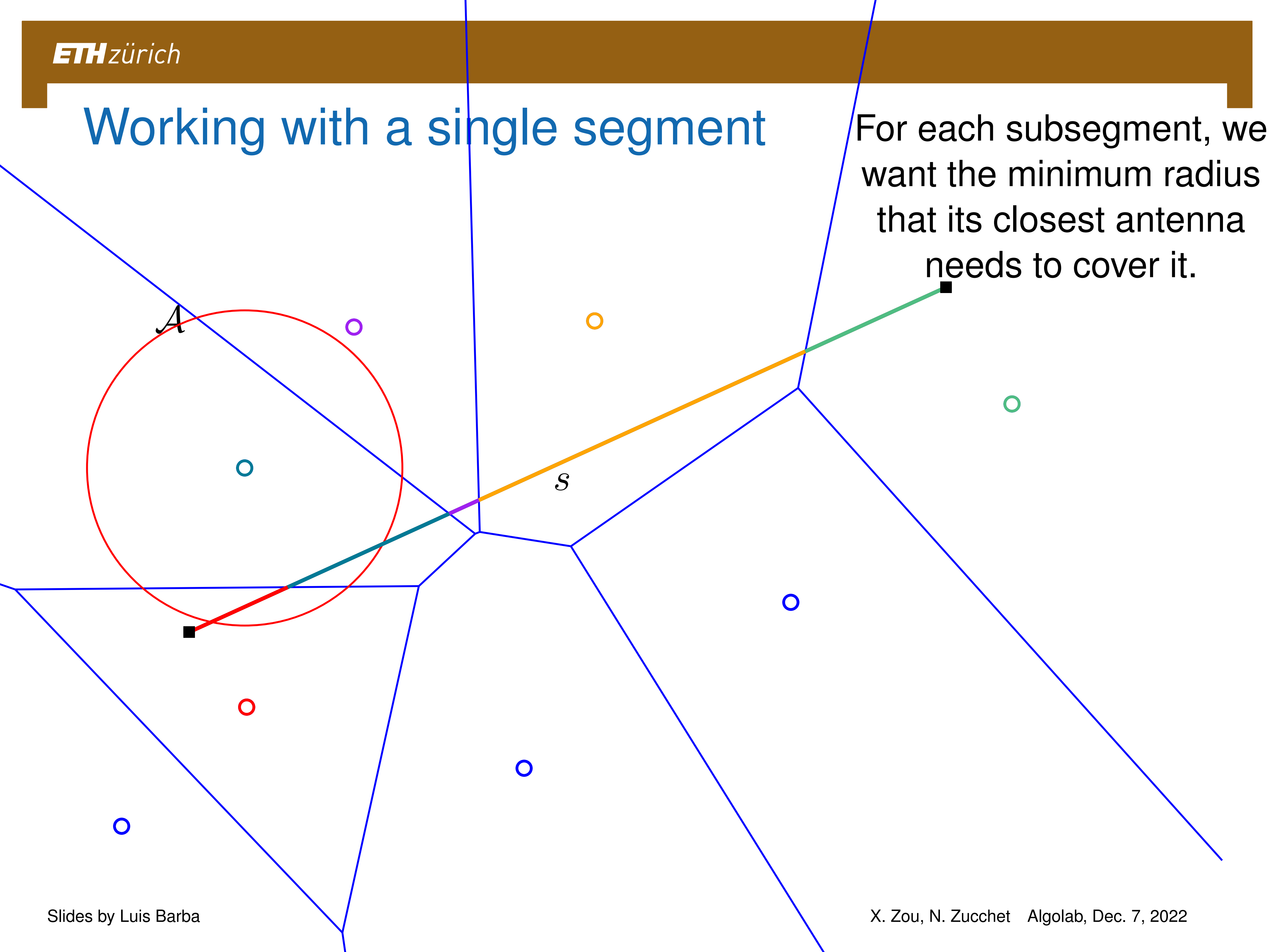
# Working with a single segment

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



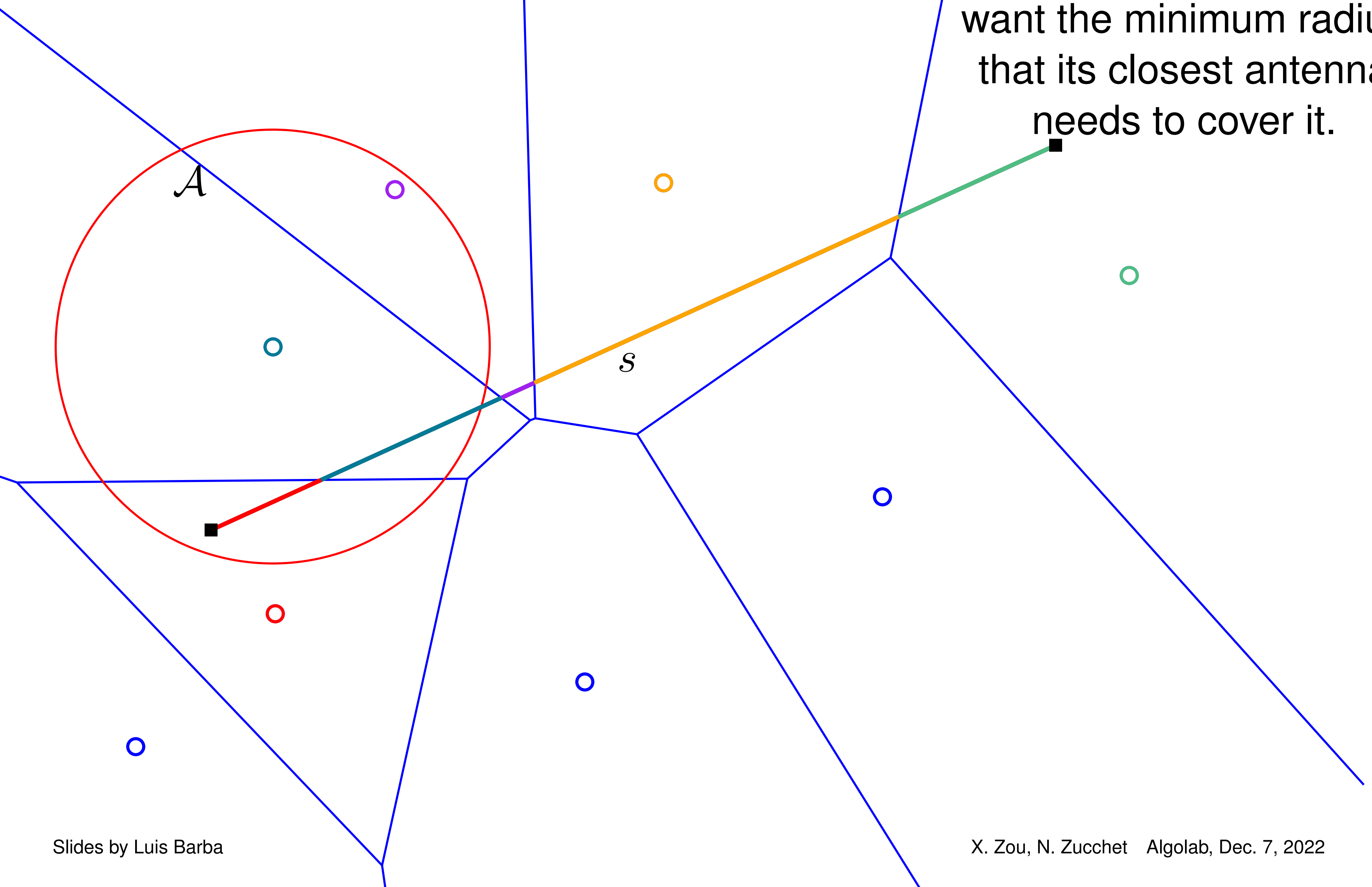
# Working with a single segment

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



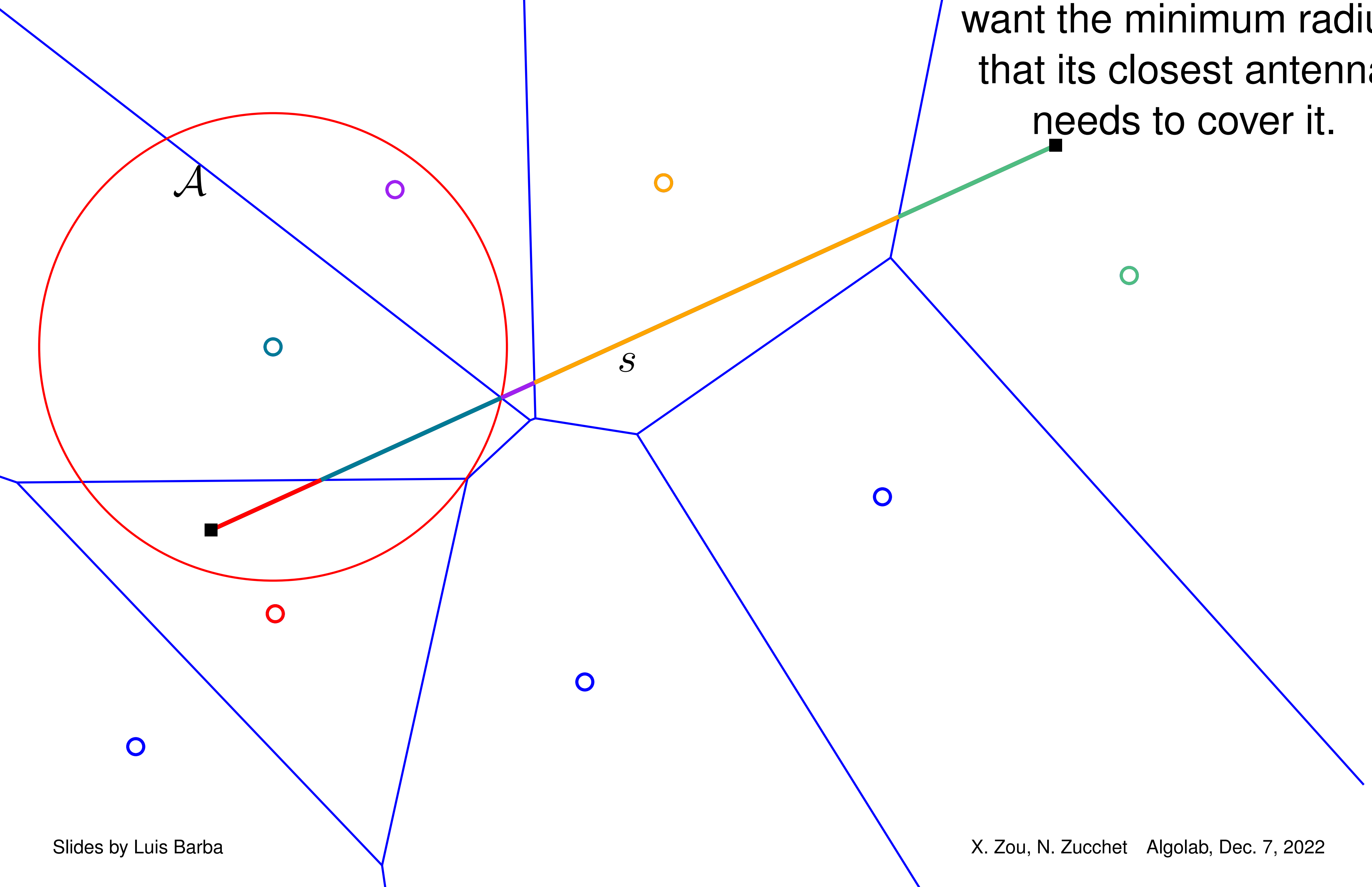
# Working with a single segment

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



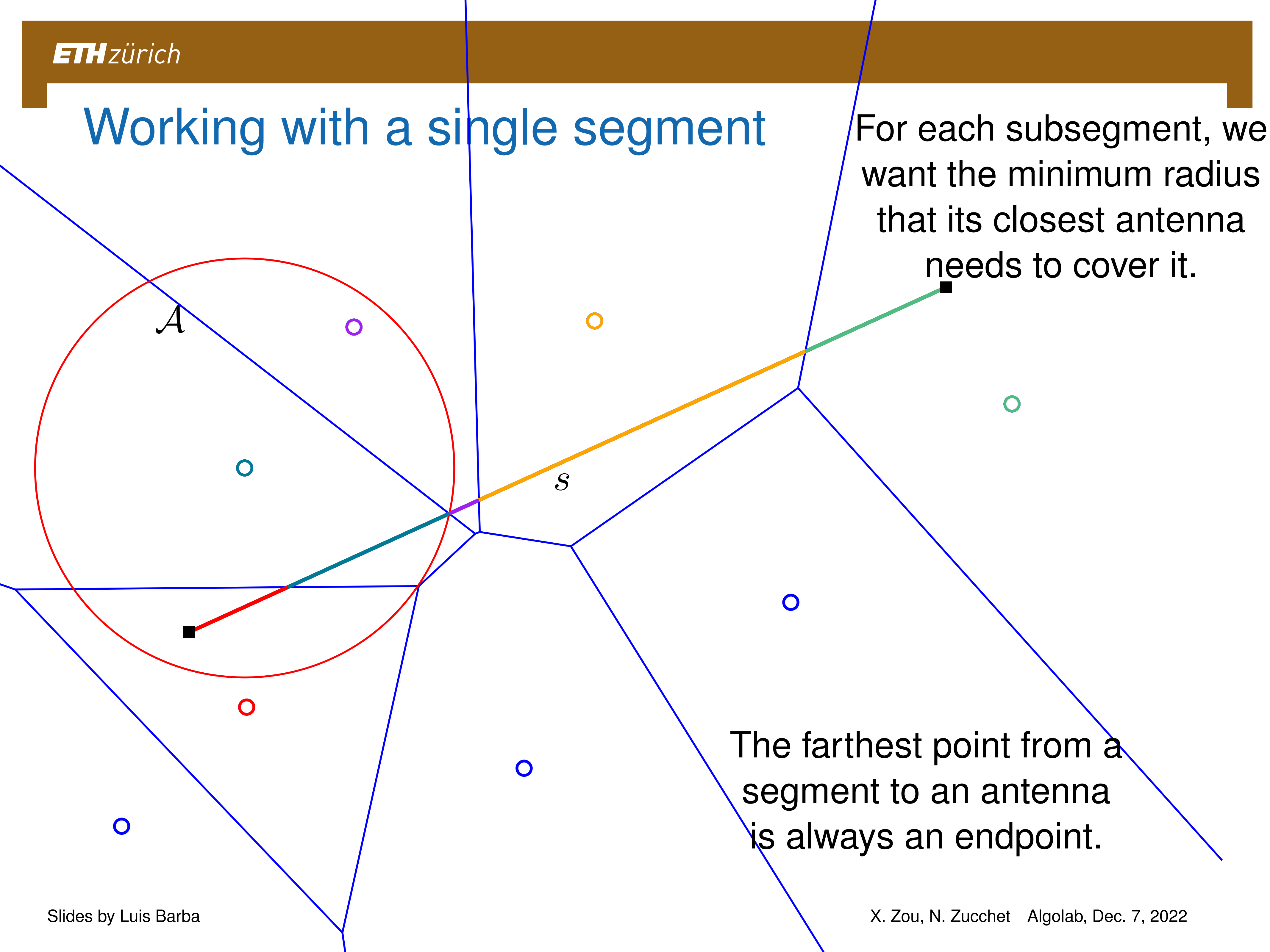
# Working with a single segment

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



# Working with a single segment

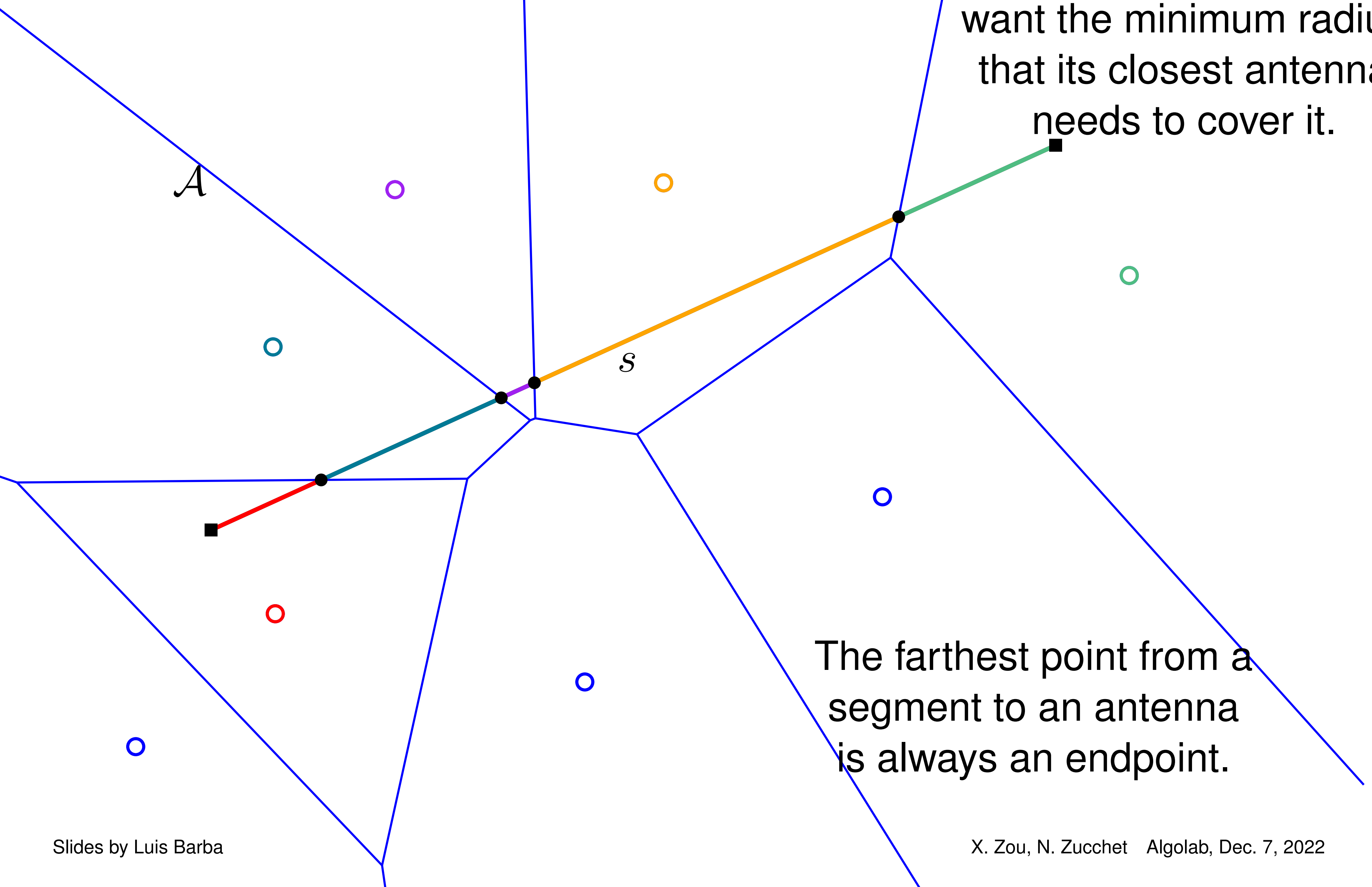
For each subsegment, we want the minimum radius that its closest antenna needs to cover it.




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

# Working with a single segment

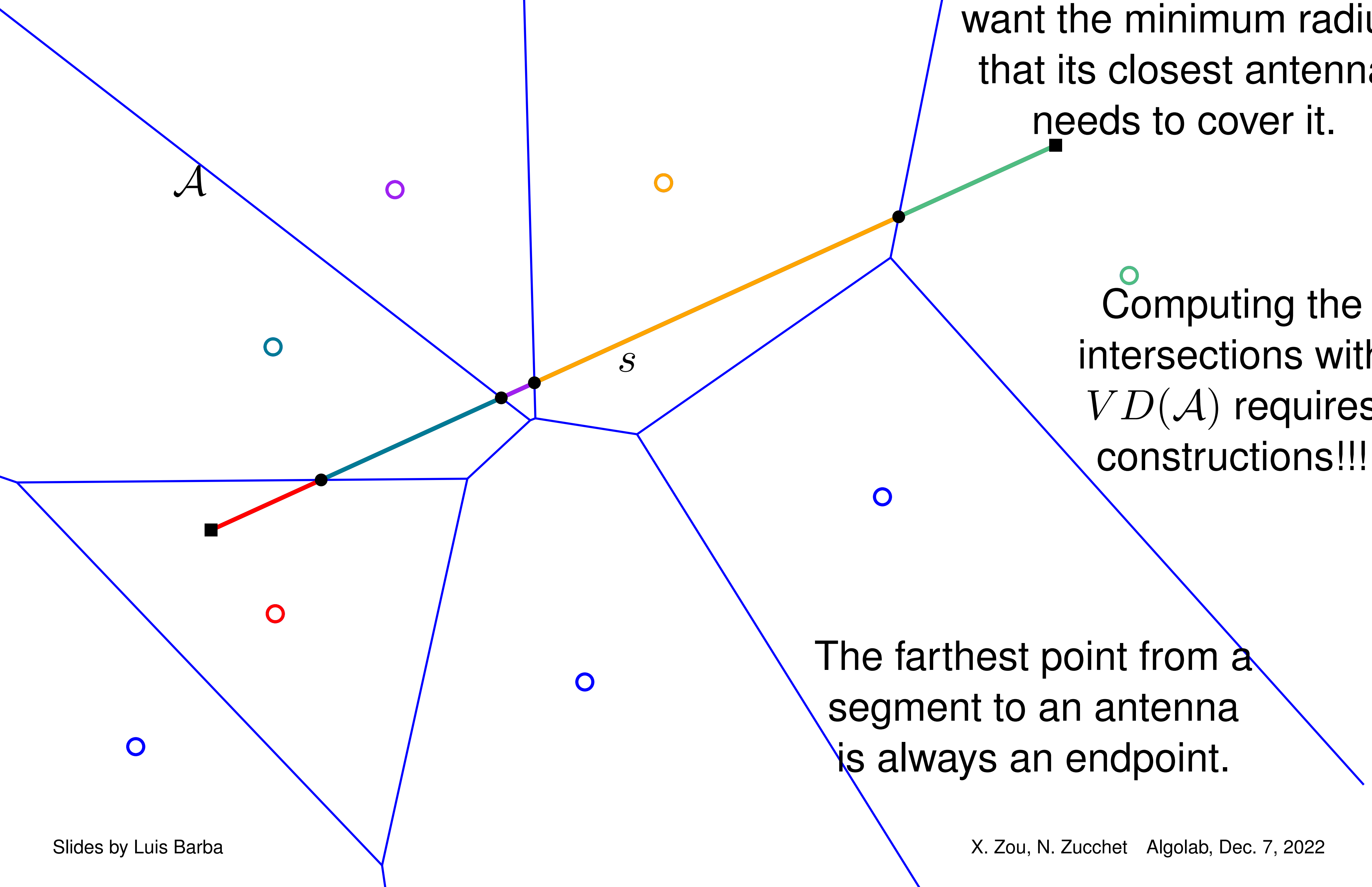
For each subsegment, we want the minimum radius that its closest antenna needs to cover it.




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

# Working with a single segment

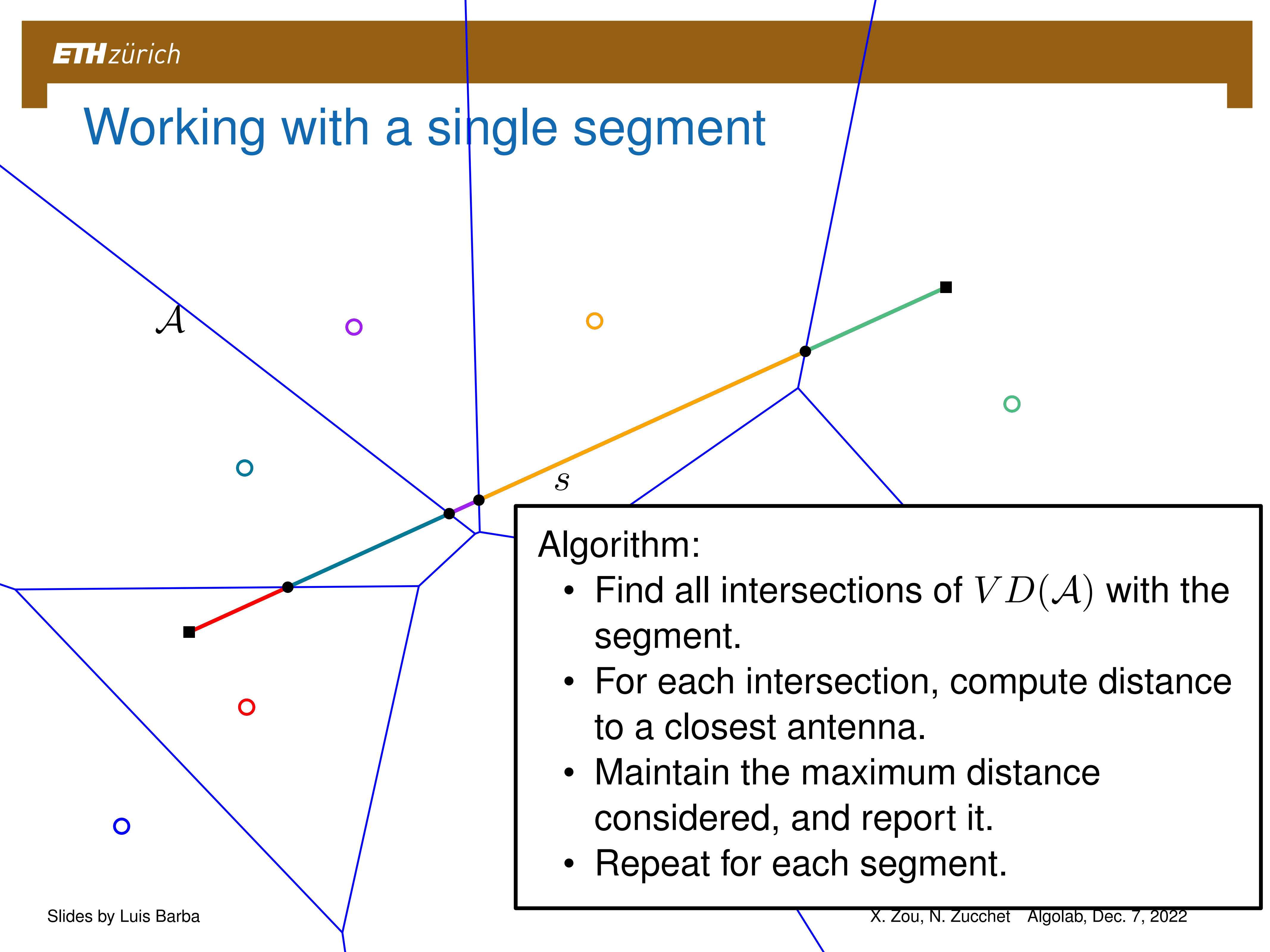
For each subsegment, we want the minimum radius that its closest antenna needs to cover it.




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

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

# Working with a single segment

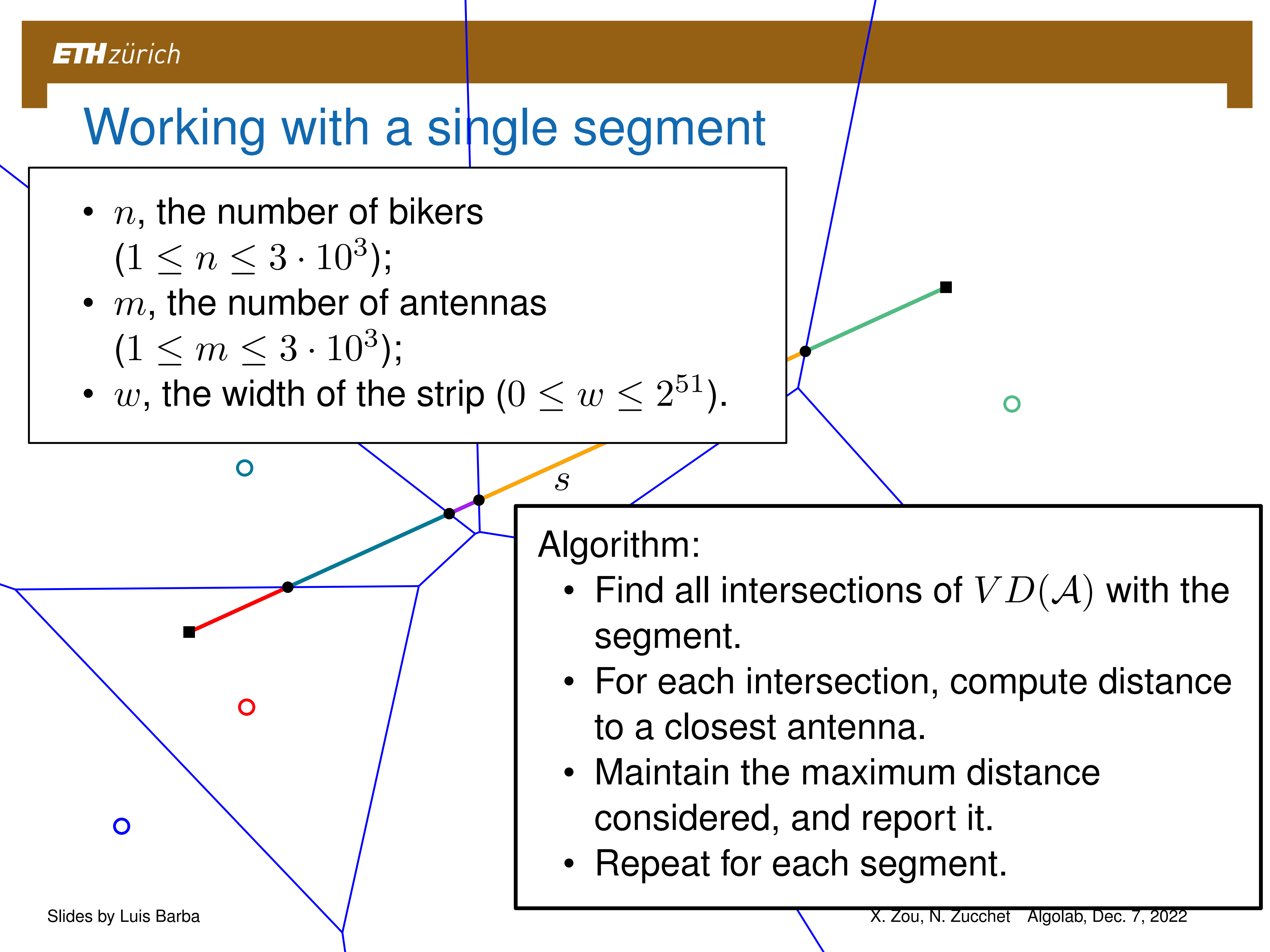



Algorithm:

- 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 ($0 \leq w \leq 2^{51}$).

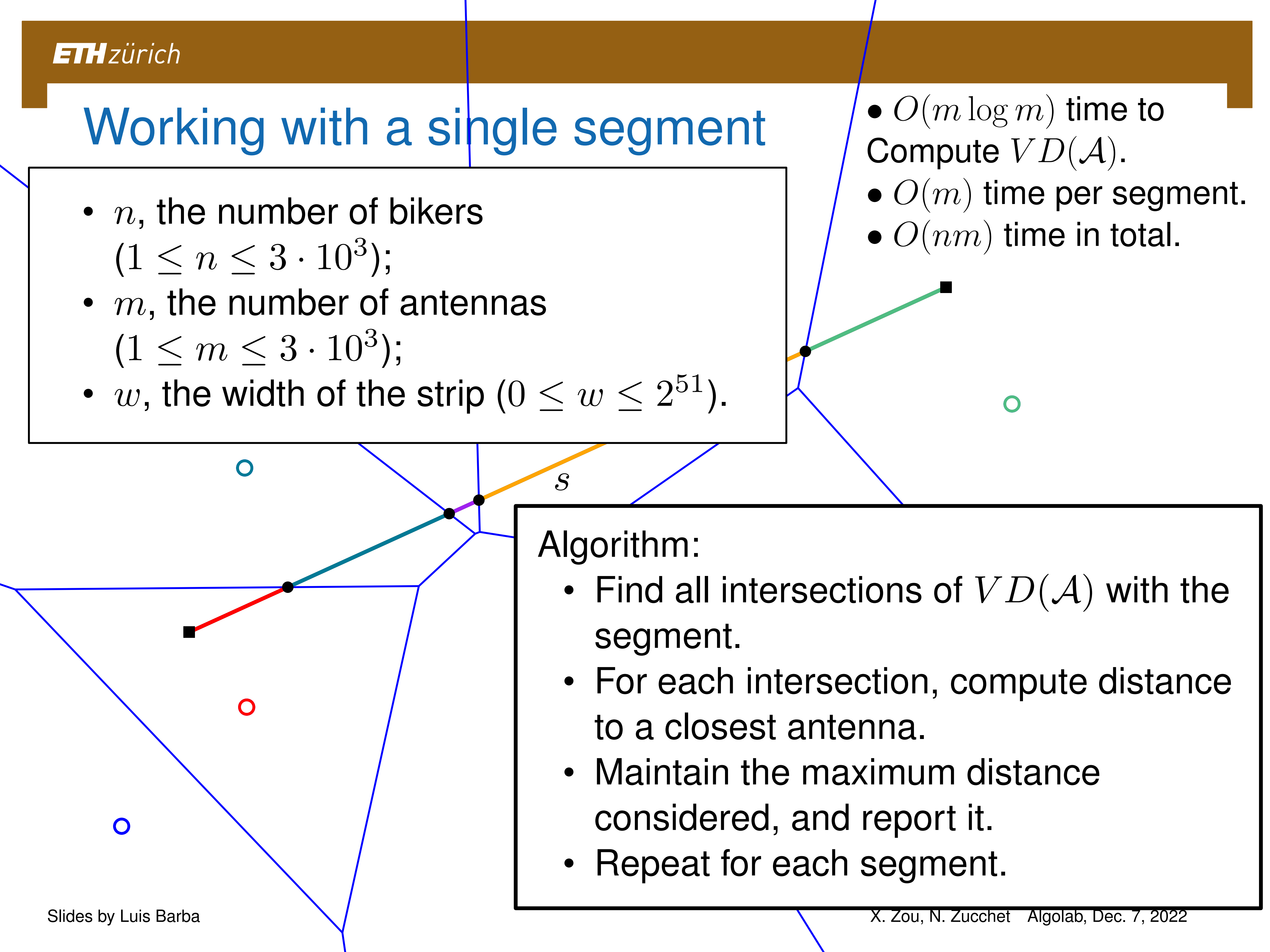



Algorithm:
- Find all intersections of $VD(\mathcal{A})$ with the segment.
- For each intersection, compute distance to a closest antenna.
- Maintain the maximum distance considered, and report it.
- Repeat for each segment.

# 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 $(0 \leq w \leq 2^{51})$.

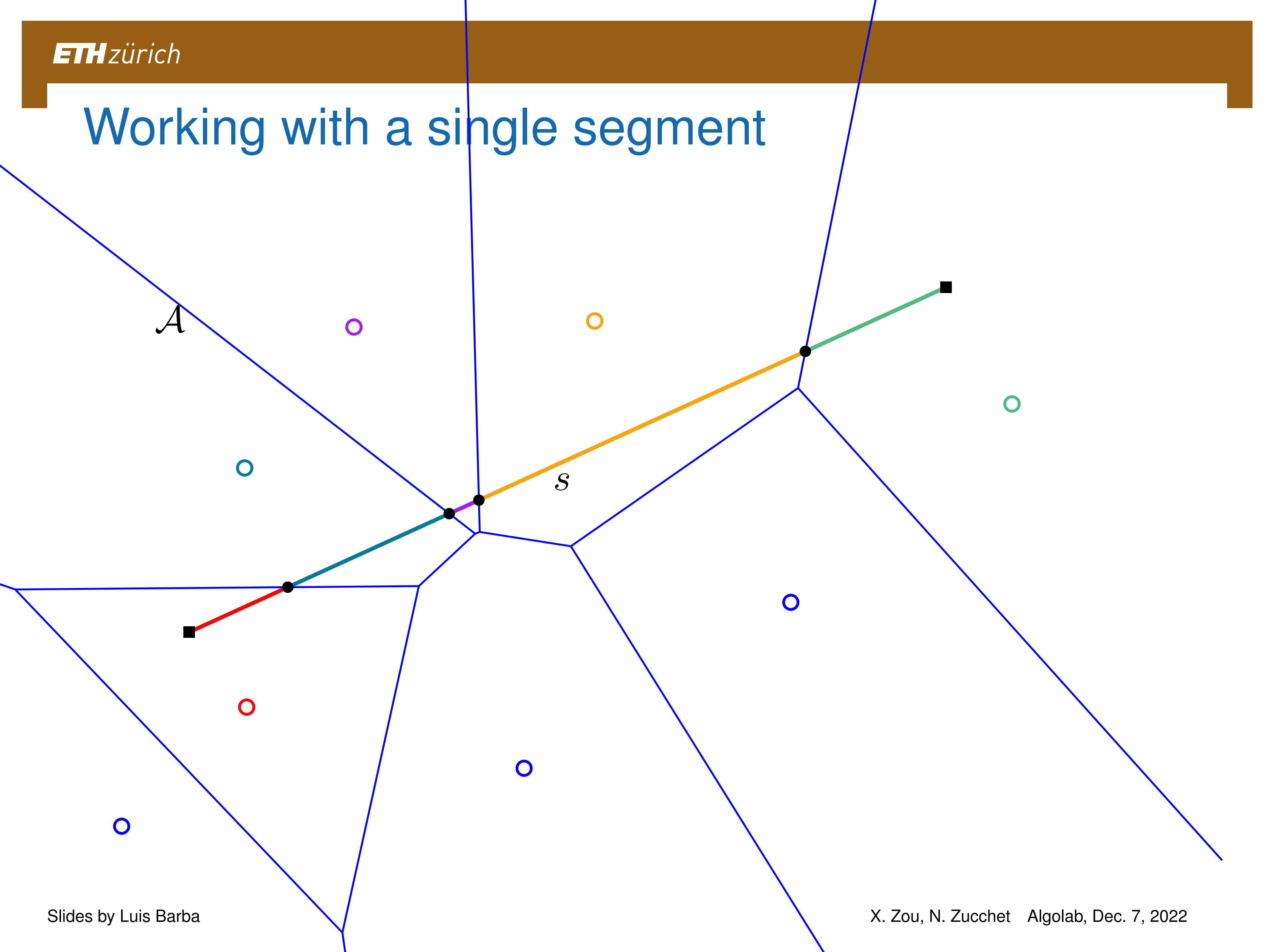
- $O(m \log m)$ time to Compute $VD(\mathcal{A})$.
- $O(m)$ time per segment.
- $O(nm)$ time in total.



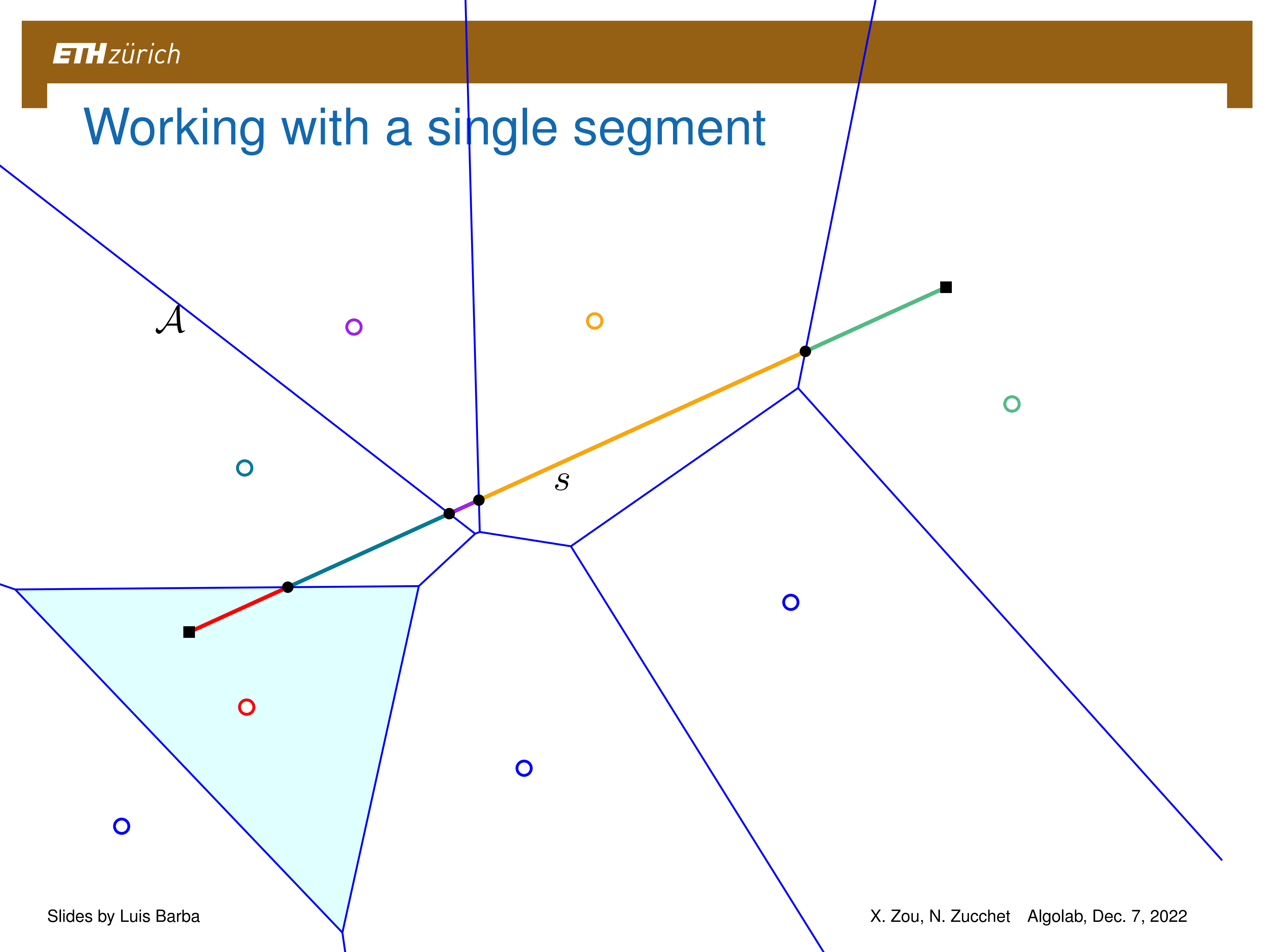

Algorithm:

- 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.

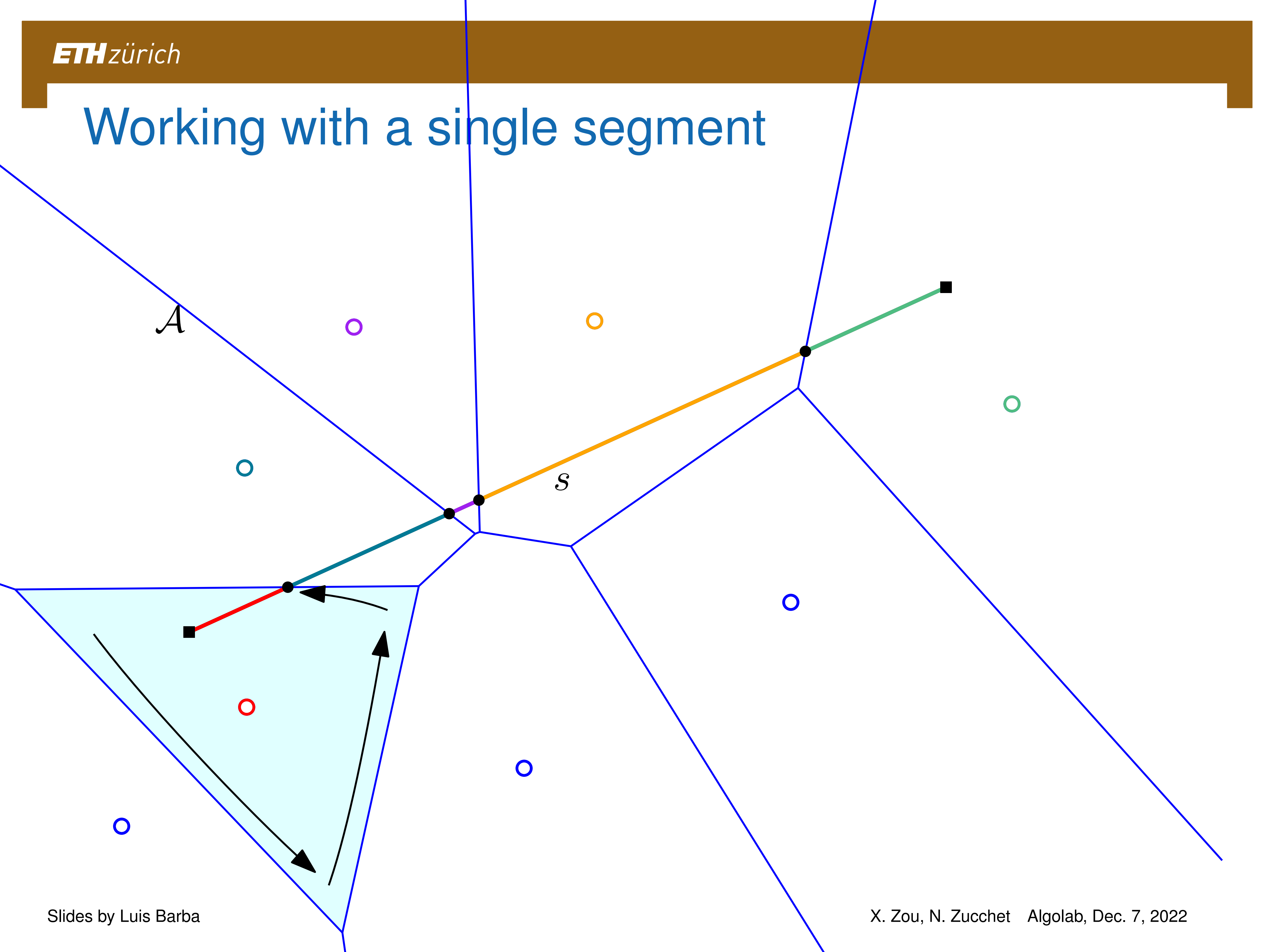
# Working with a single segment



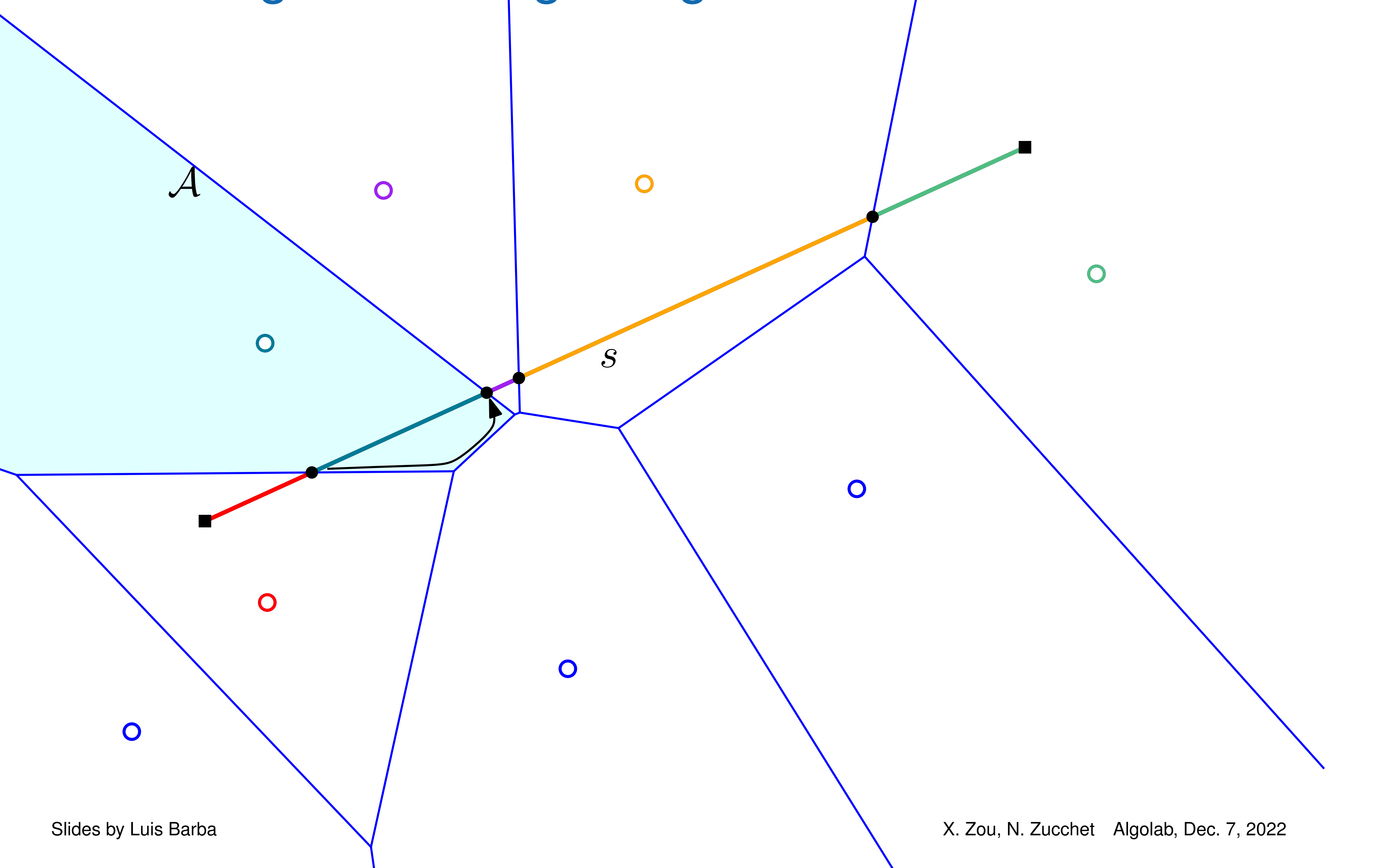
# Working with a single segment



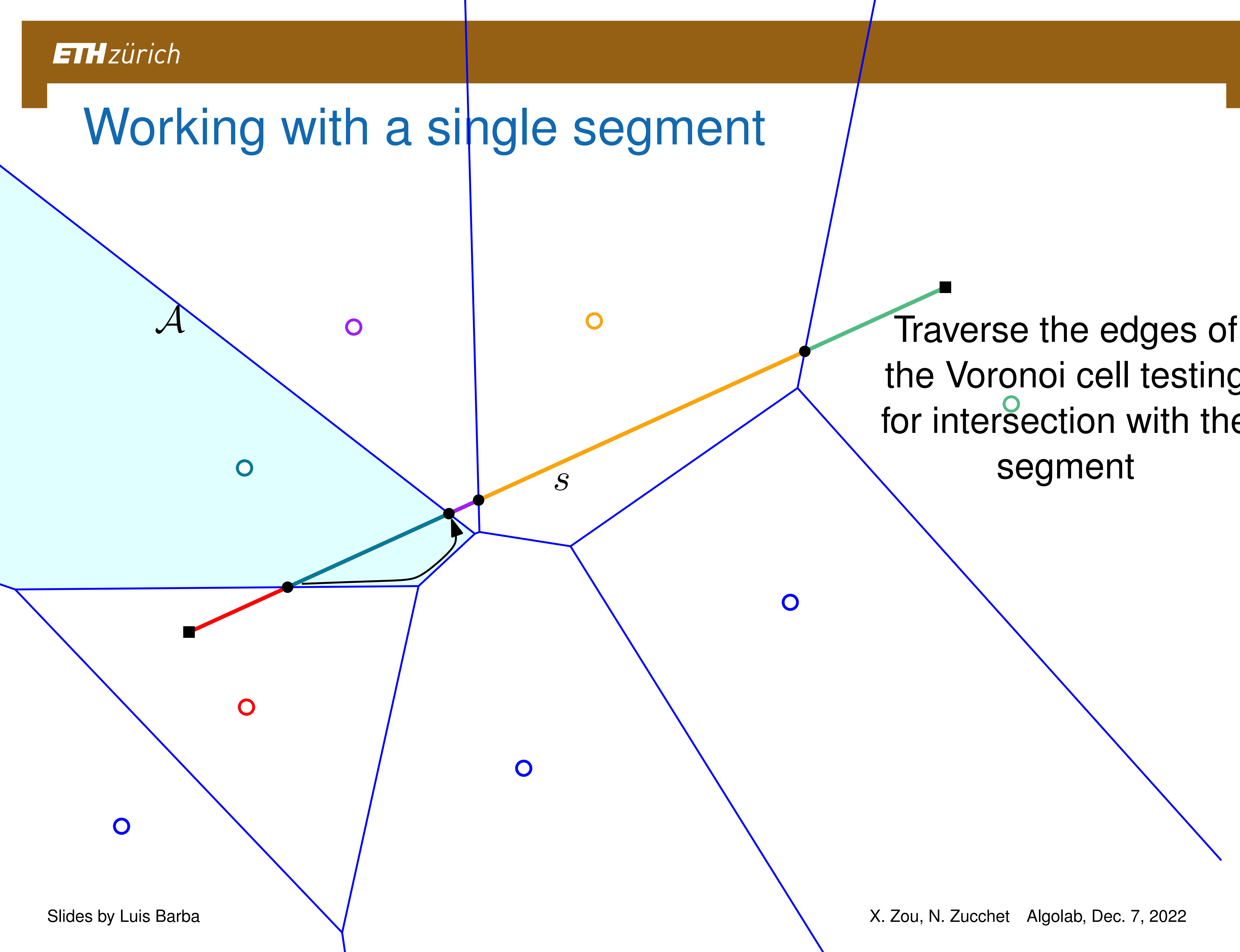
# Working with a single segment



# Working with a single segment

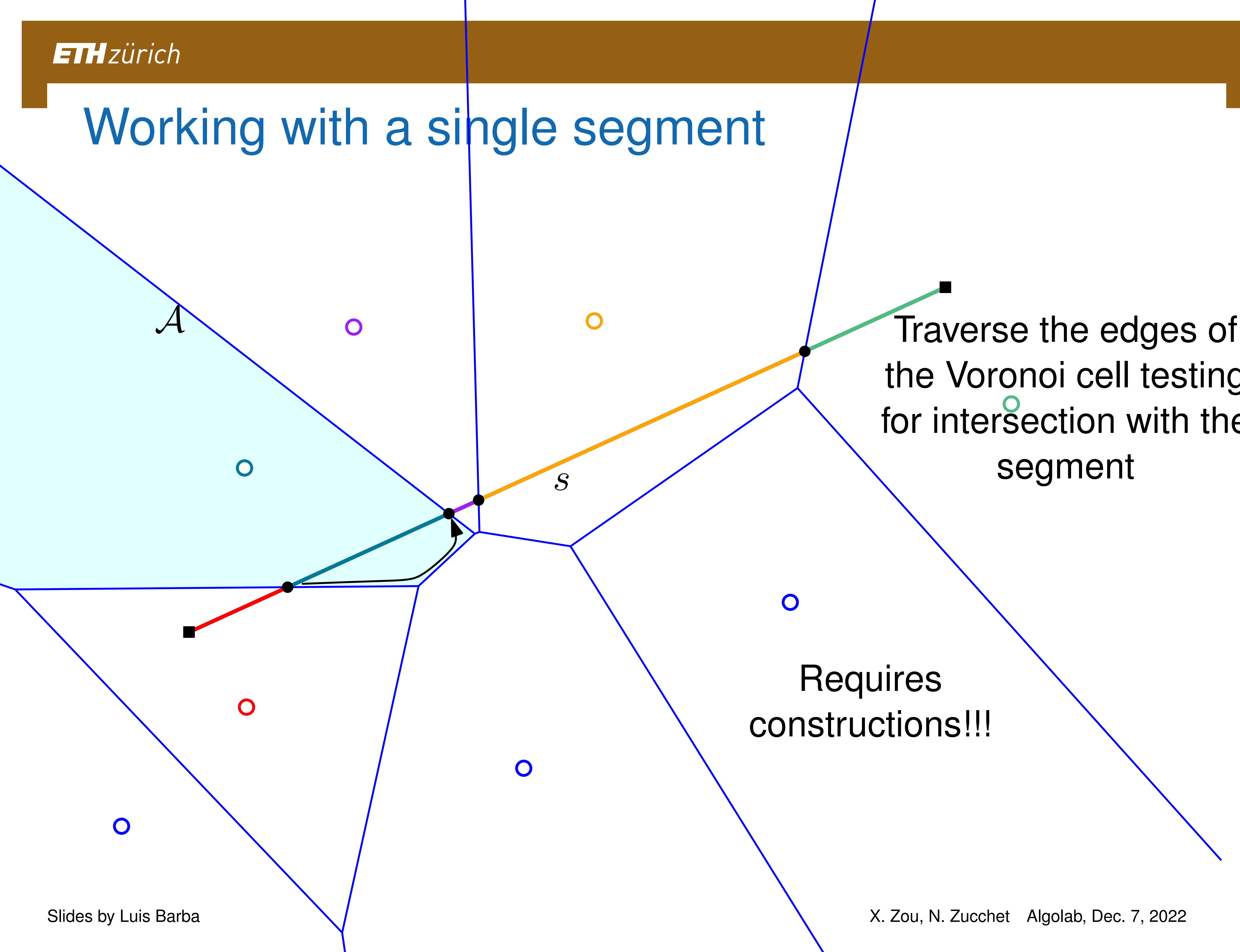


# Working with a single segment




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

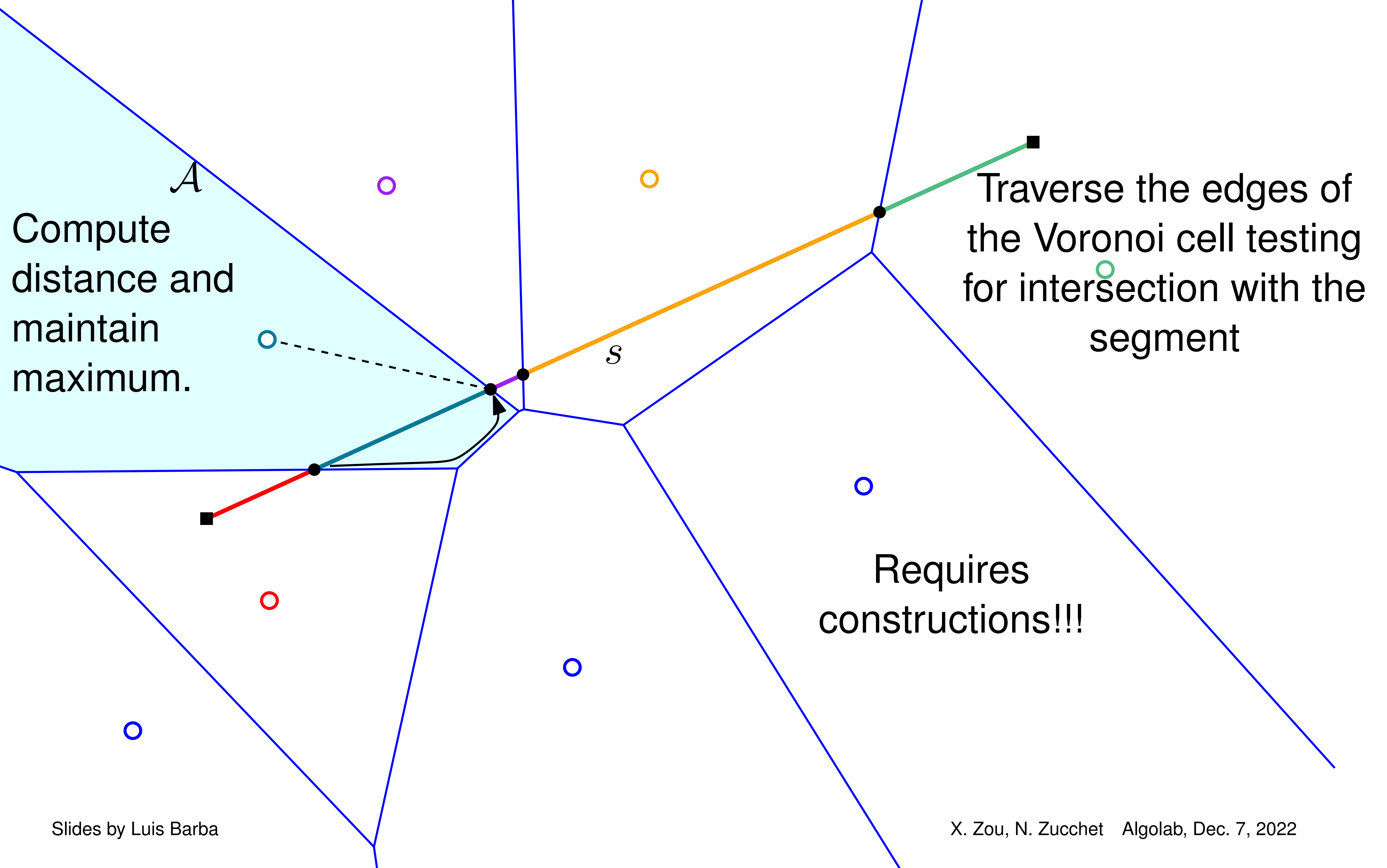
# Working with a single segment




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

Requires constructions!!!

# 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