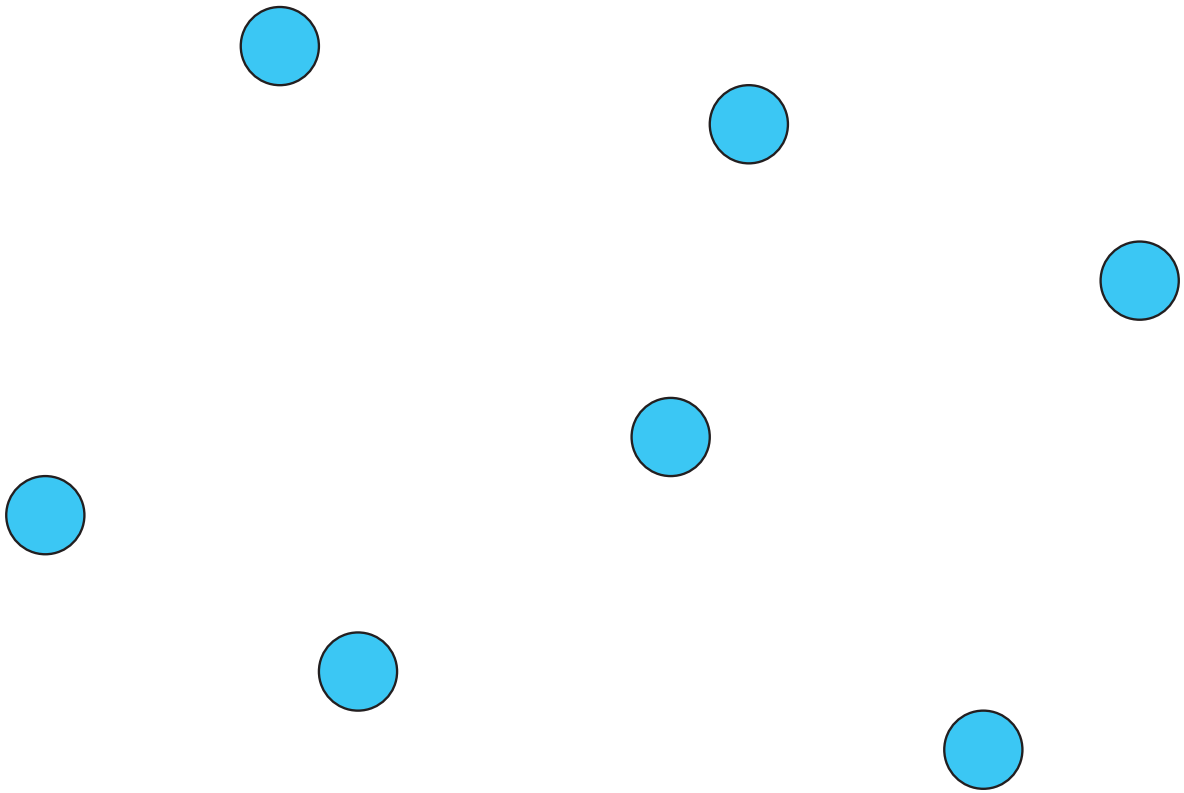
Square Formation by Asynchronous Oblivious Robots

CCCG 2016

Marcello Mamino, Giovanni Viglietta

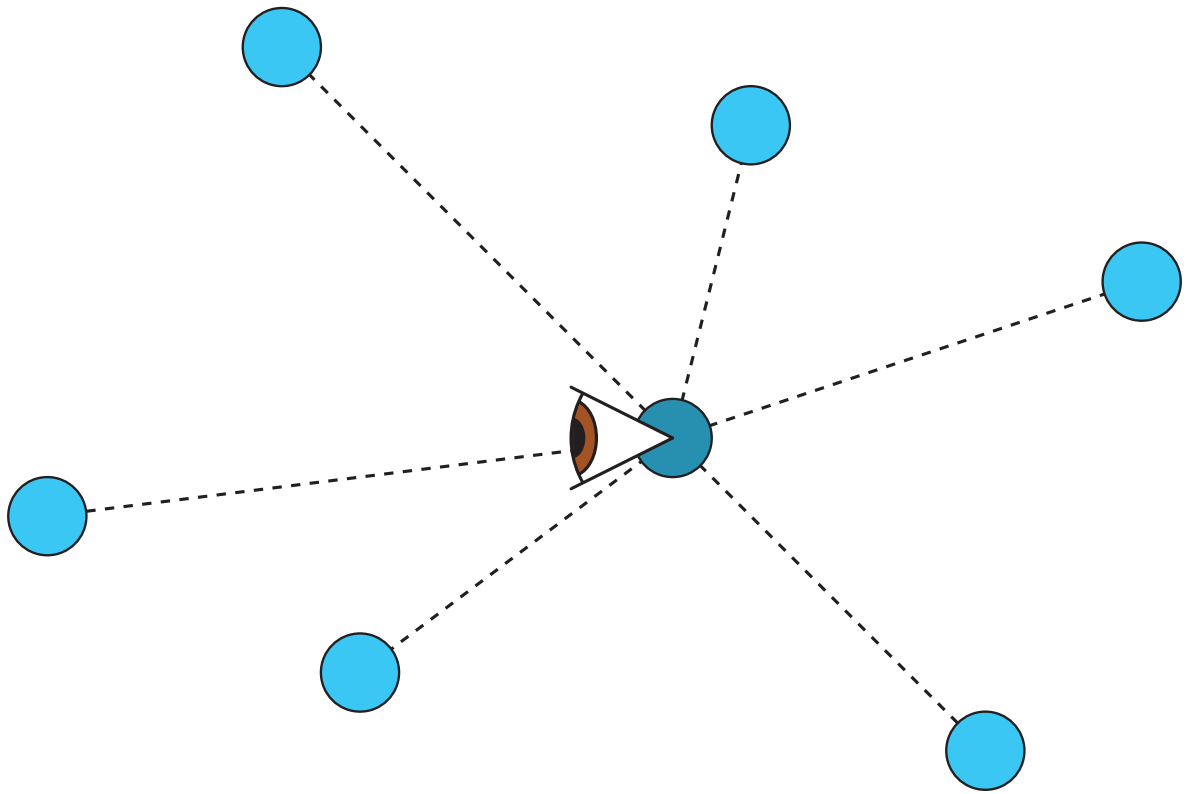
Vancouver – August 3, 2016

Anonymous robots sensing and moving in the plane



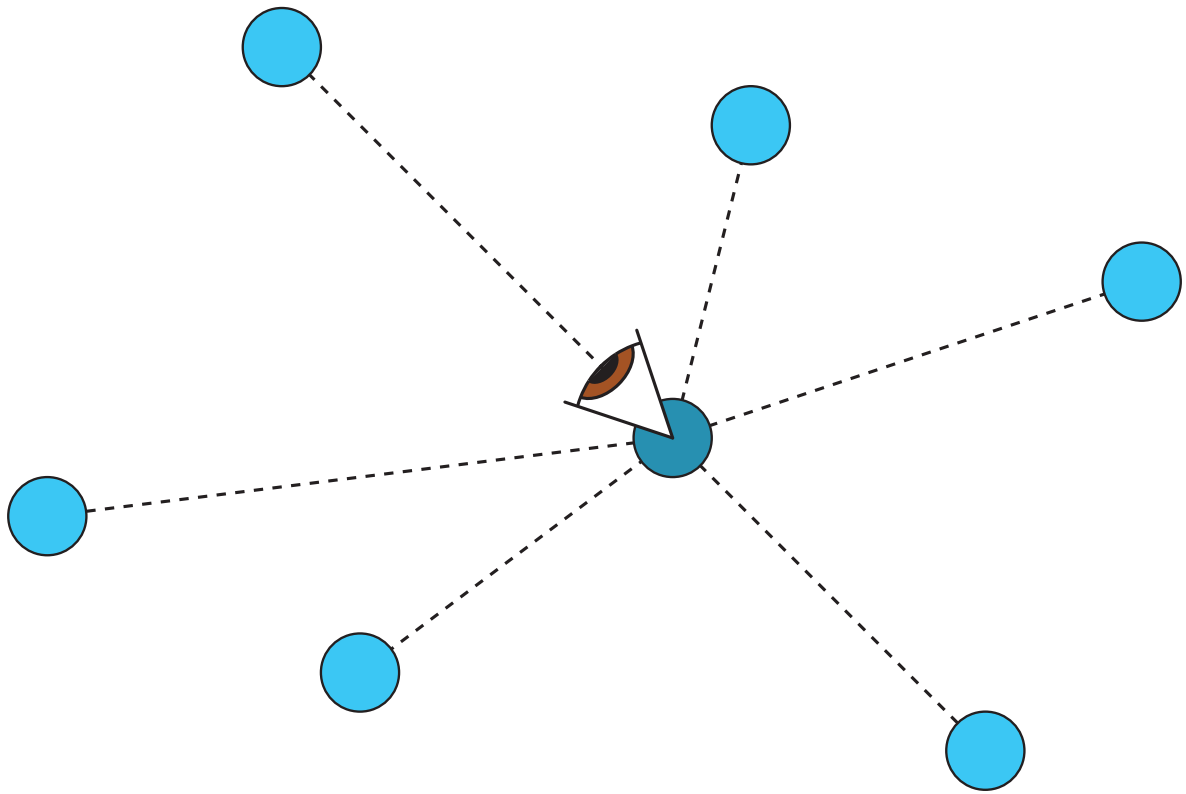
We consider a swarm of anonymous robots in the Euclidean plane

Anonymous robots sensing and moving in the plane



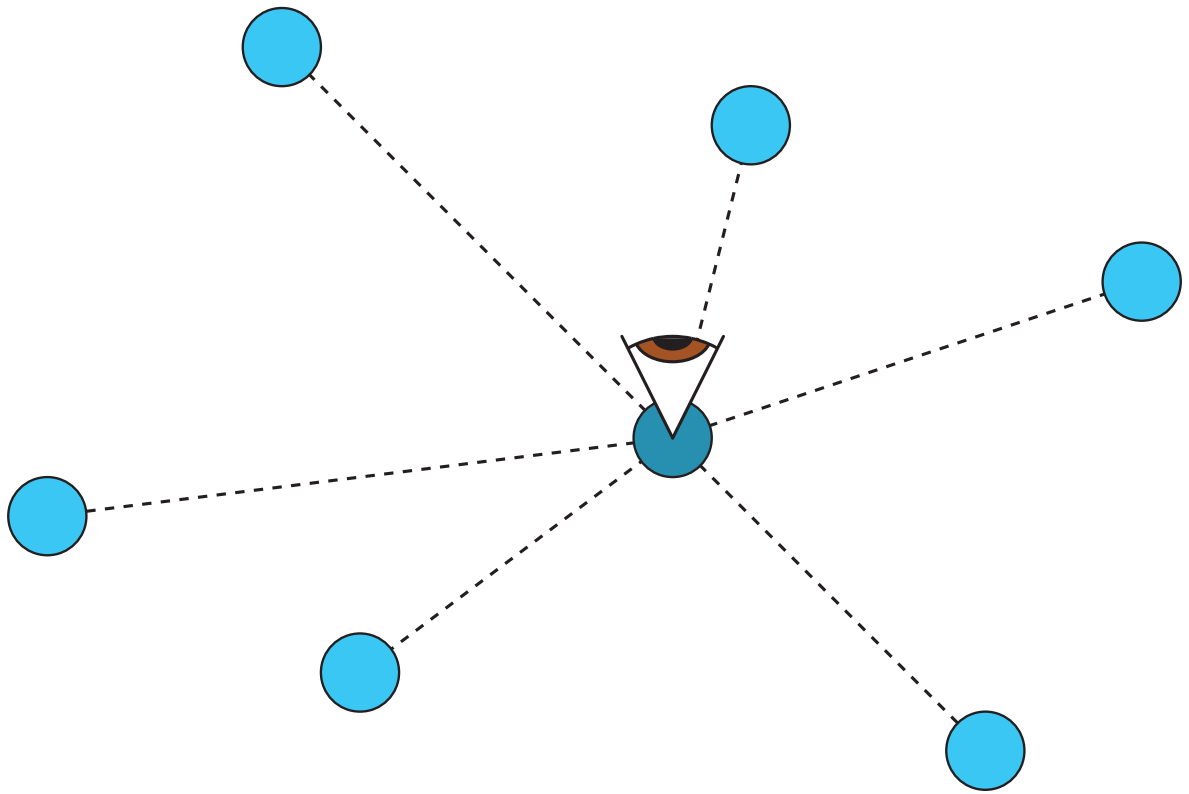
Each robot can sense the positions of all other robots...

Anonymous robots sensing and moving in the plane



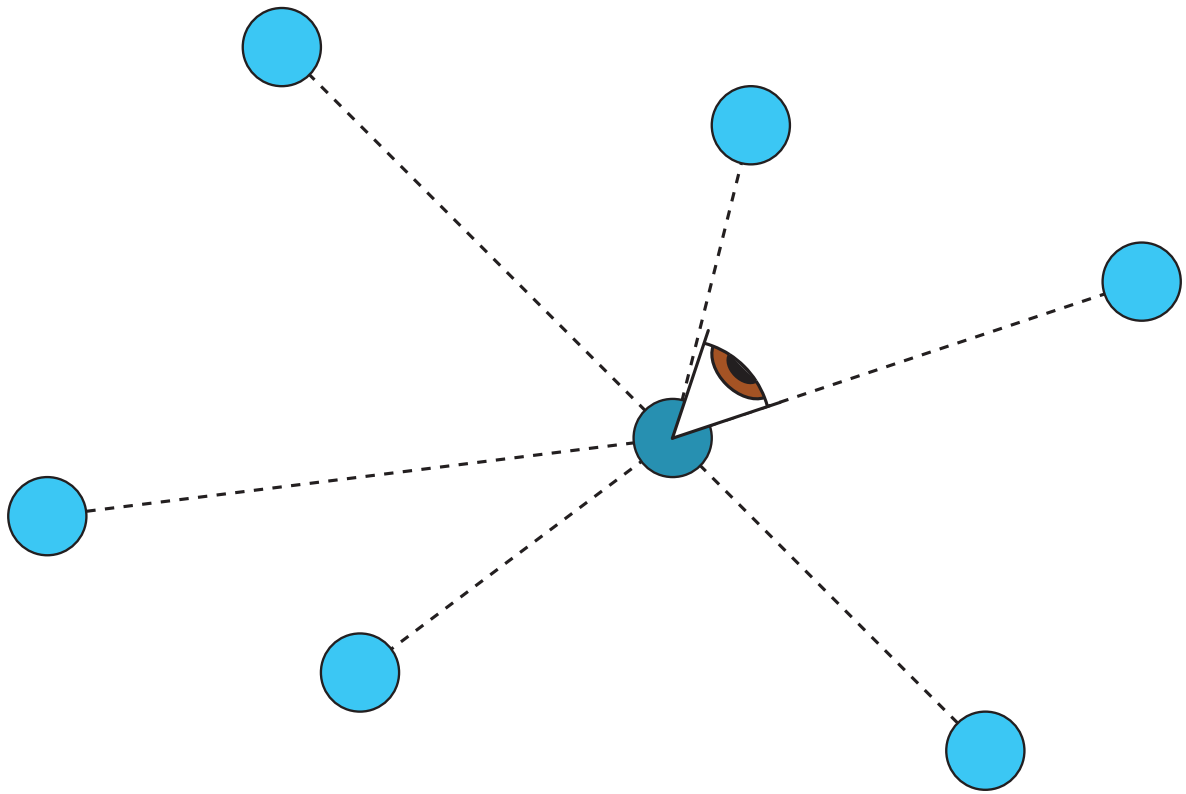
Each robot can sense the positions of all other robots...

Anonymous robots sensing and moving in the plane



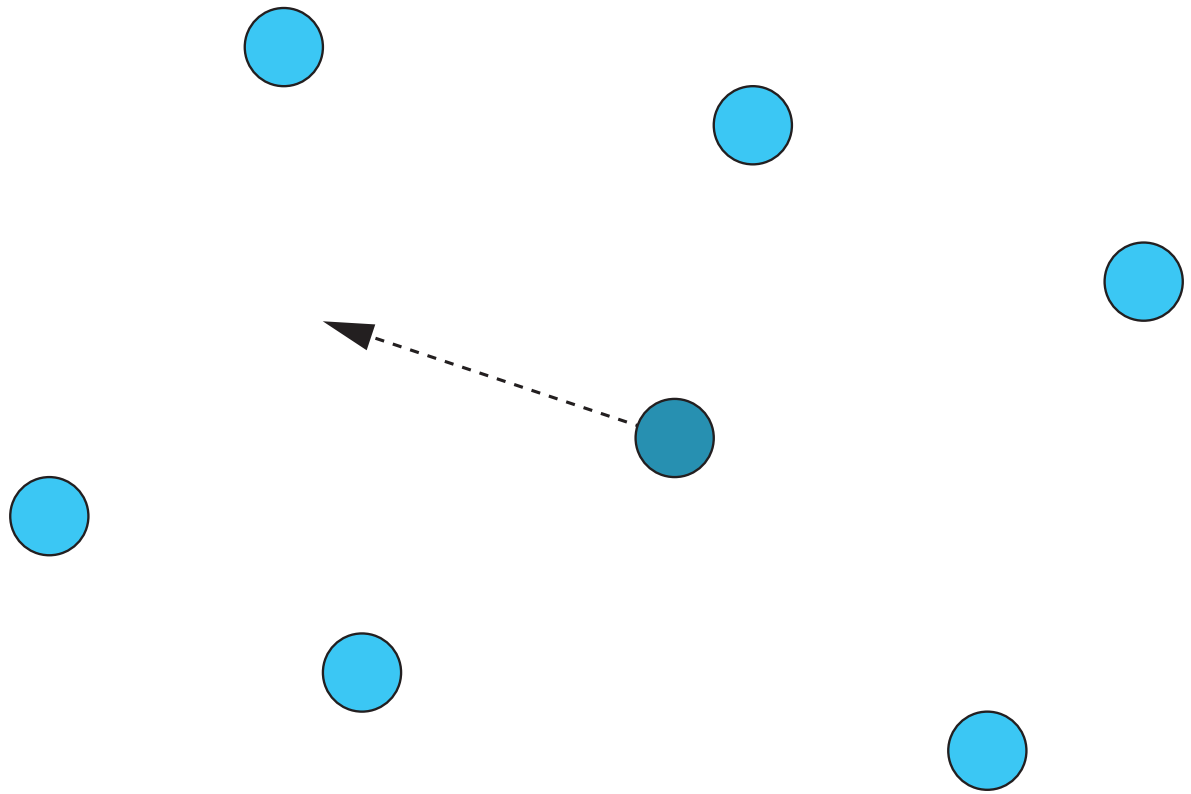
Each robot can sense the positions of all other robots...

Anonymous robots sensing and moving in the plane



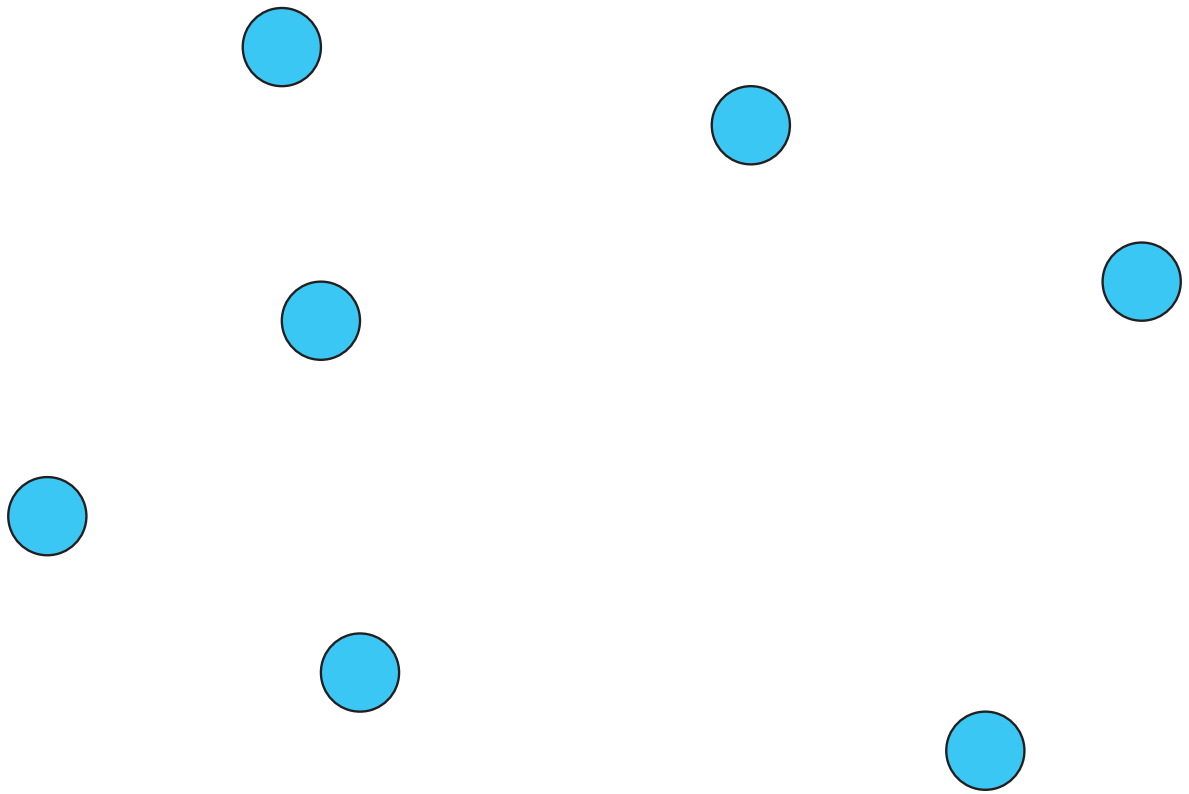
Each robot can sense the positions of all other robots...

Anonymous robots sensing and moving in the plane



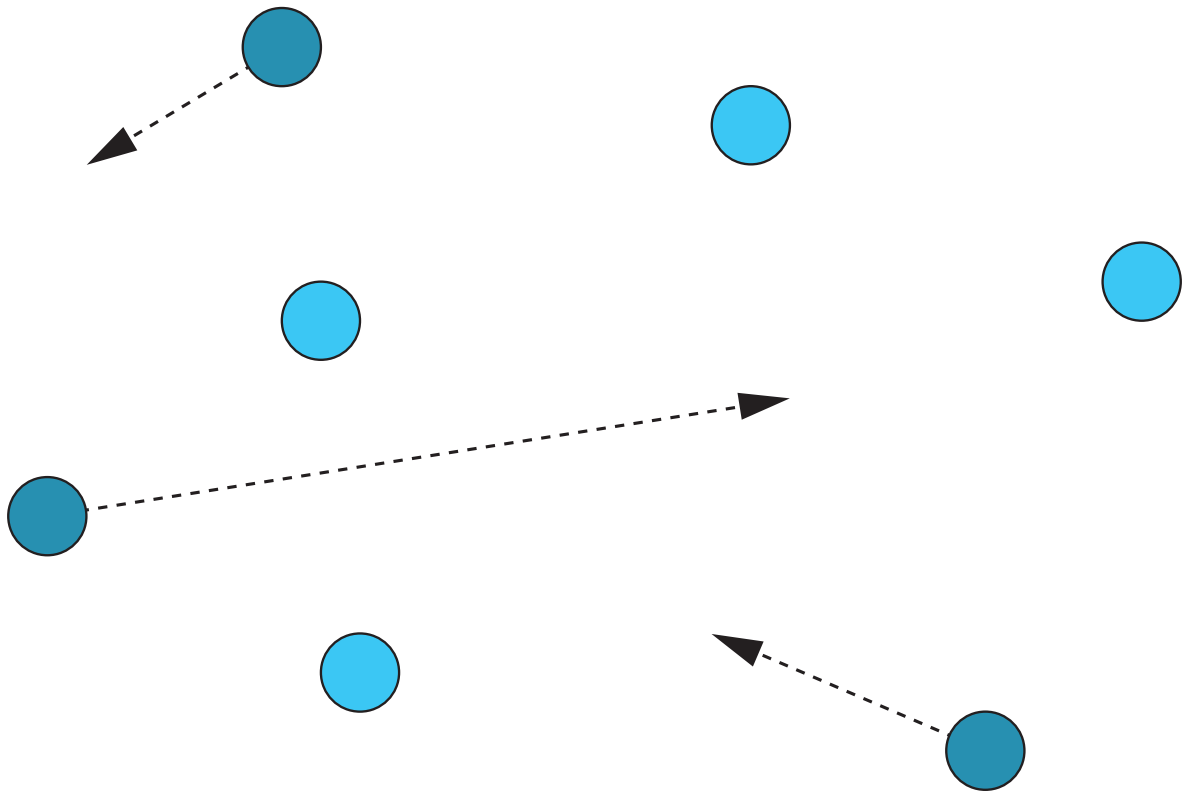
...And move according to a deterministic algorithm

Anonymous robots sensing and moving in the plane



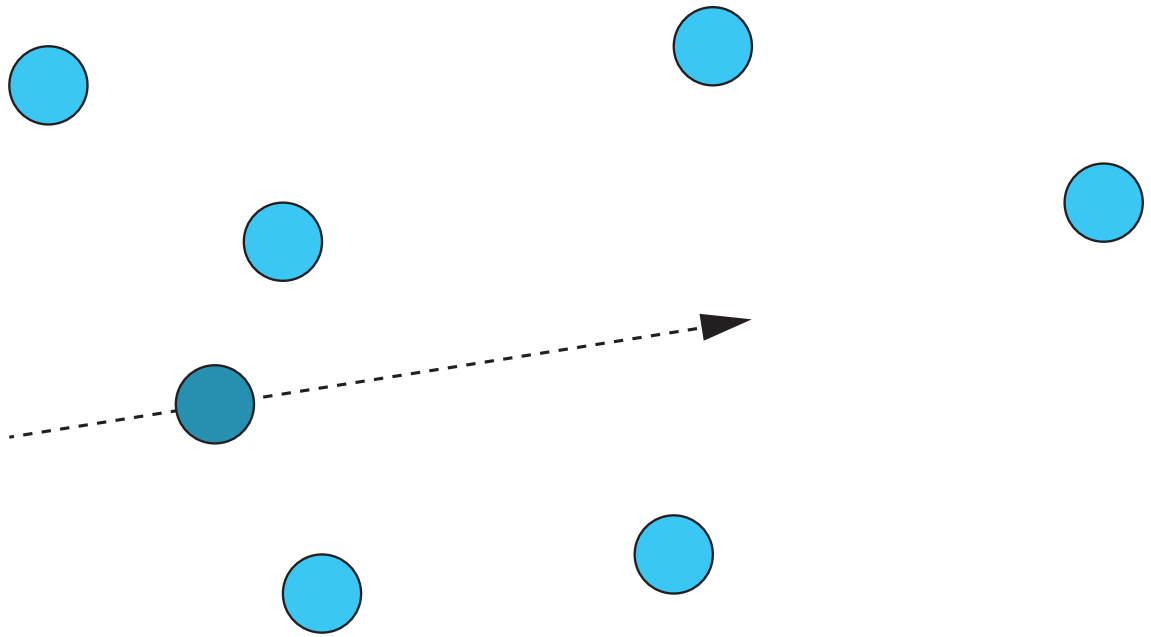
...And move according to a deterministic algorithm

Anonymous robots sensing and moving in the plane



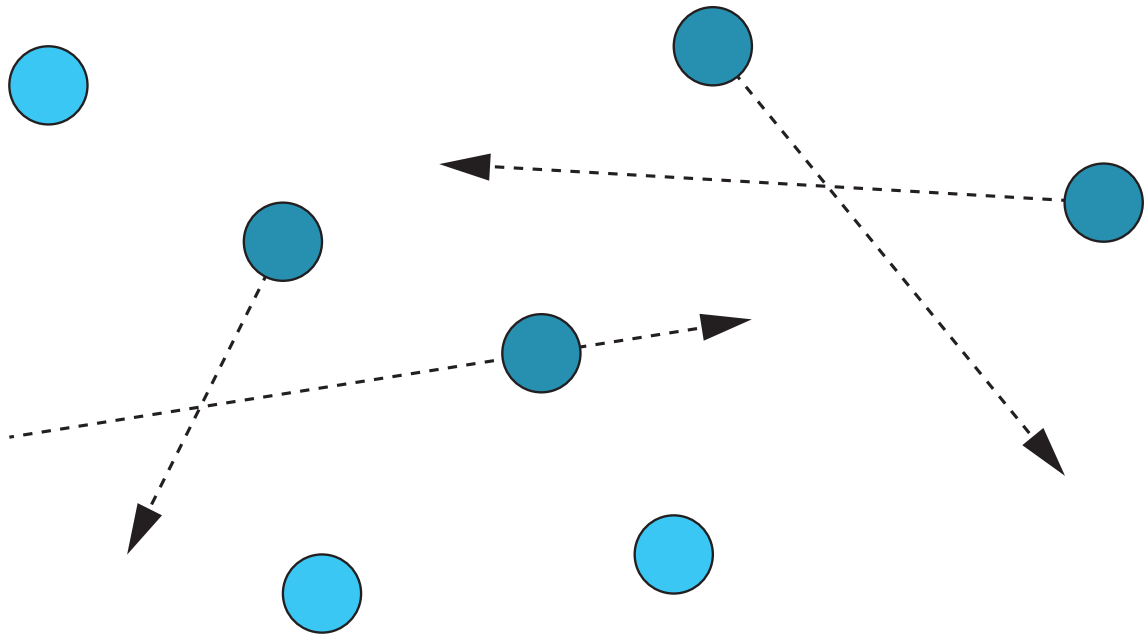
Different robots are activated asynchronously

Anonymous robots sensing and moving in the plane



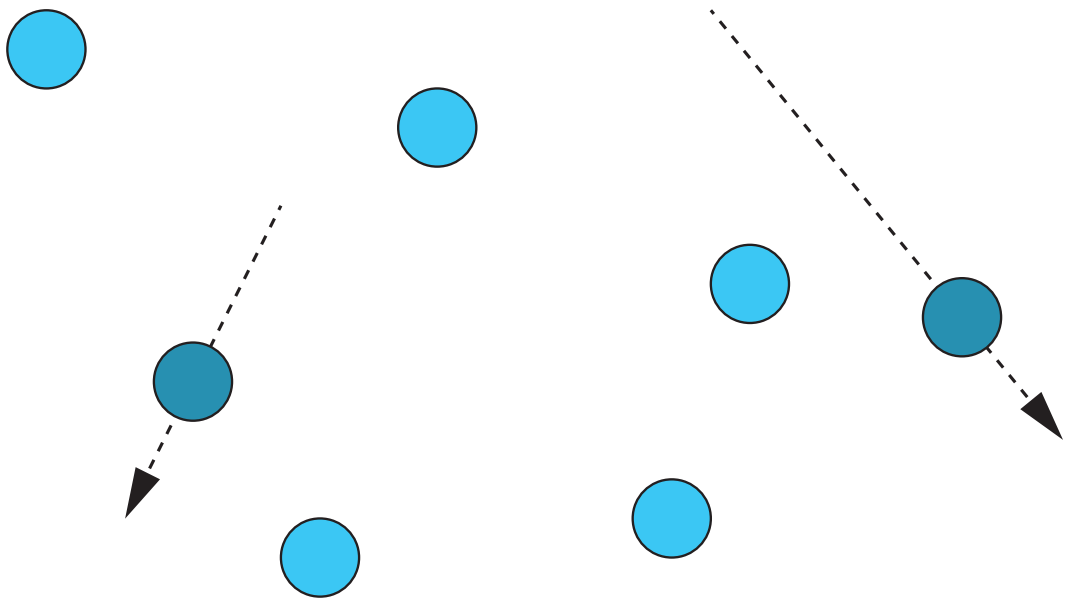
Different robots are activated asynchronously

Anonymous robots sensing and moving in the plane



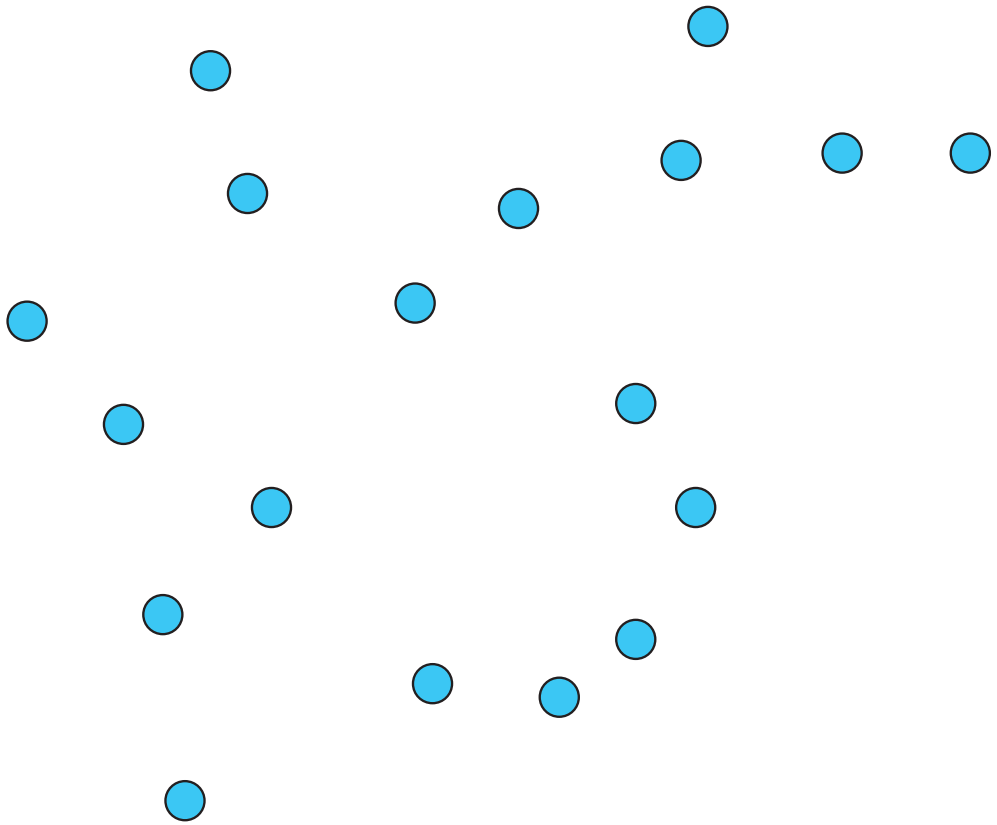
Different robots are activated asynchronously

Anonymous robots sensing and moving in the plane



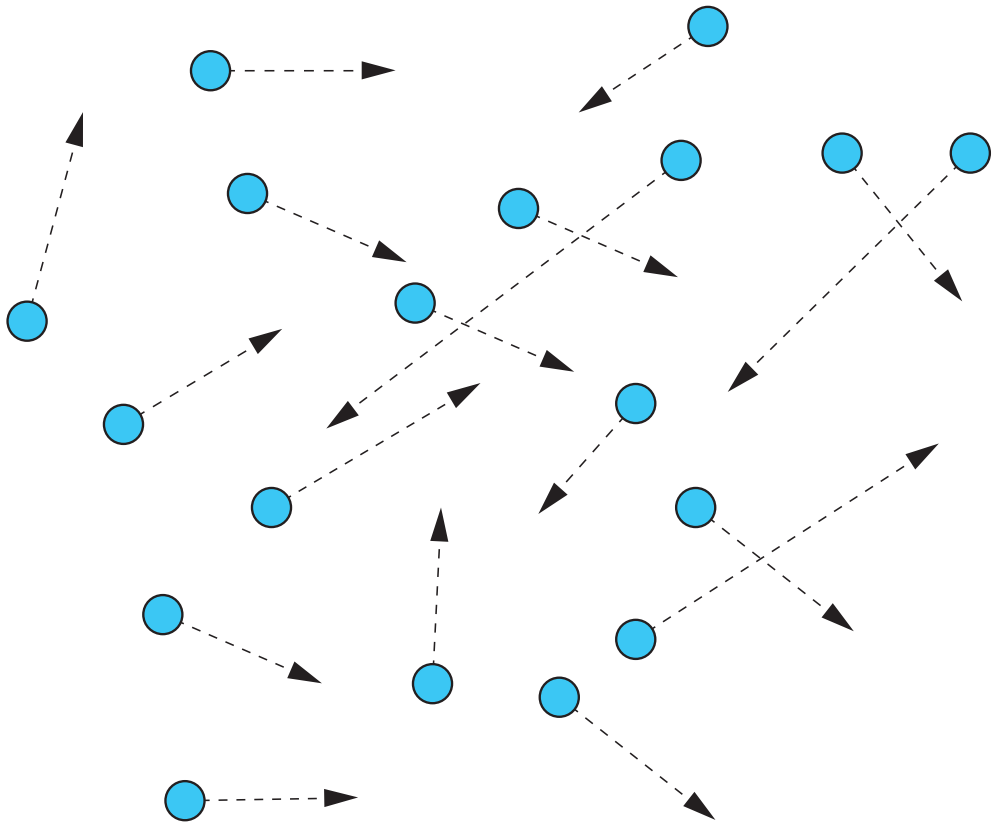
Different robots are activated asynchronously

Pattern Formation problem



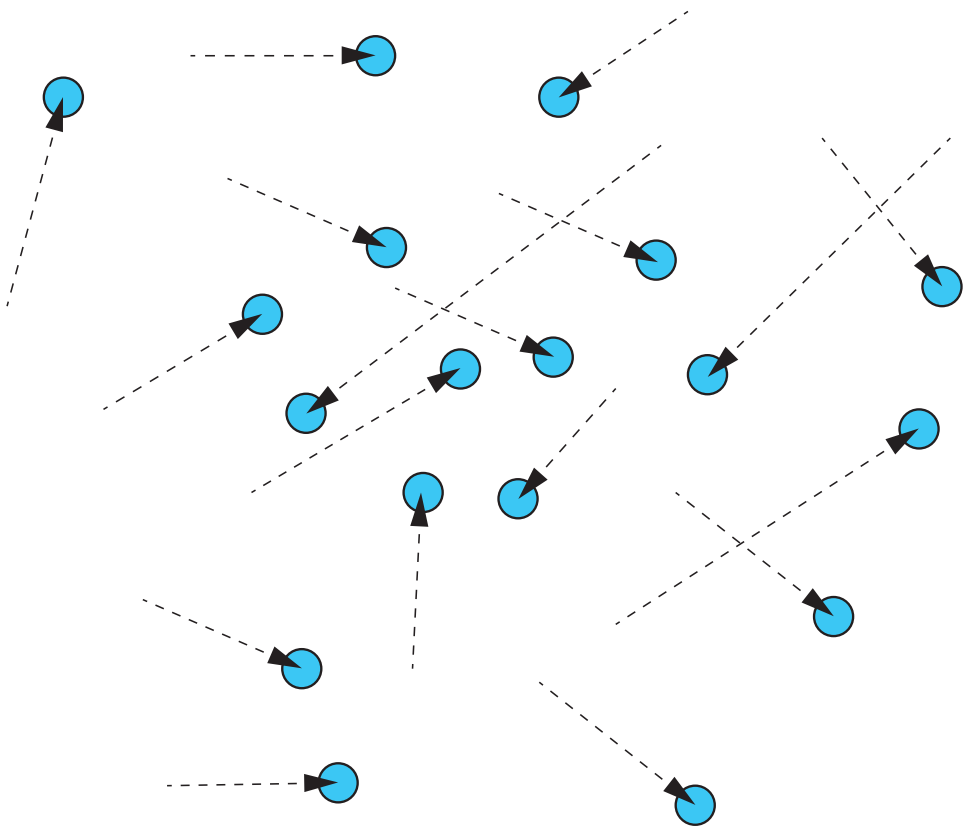
Problem: form a given pattern from any initial configuration

Pattern Formation problem



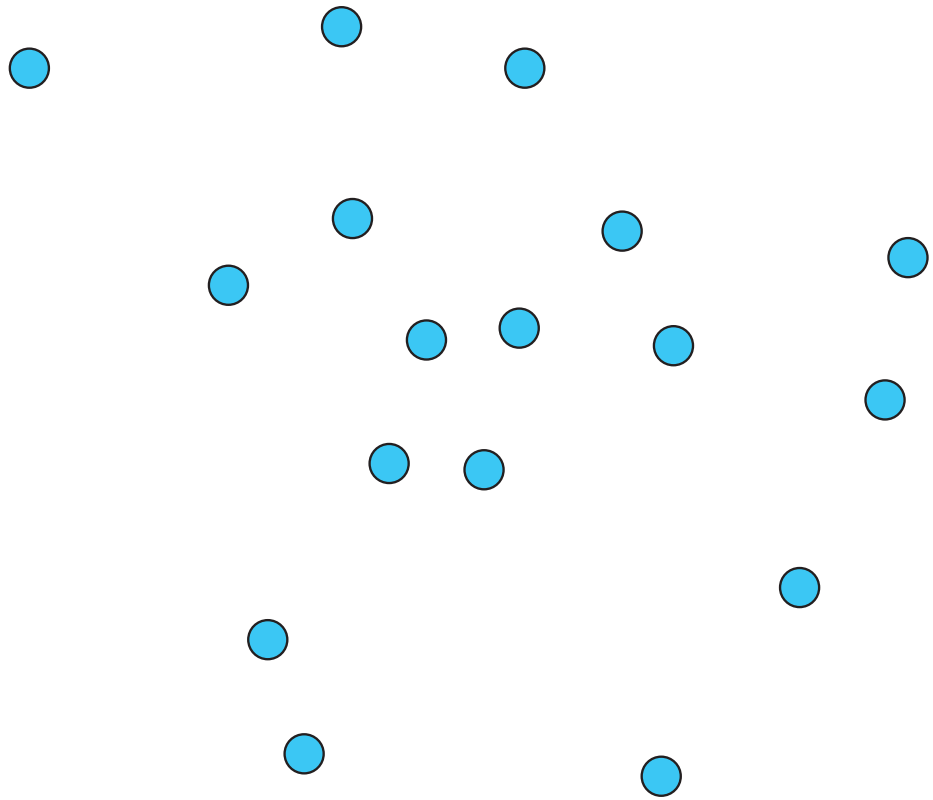
Problem: form a given pattern from any initial configuration

Pattern Formation problem



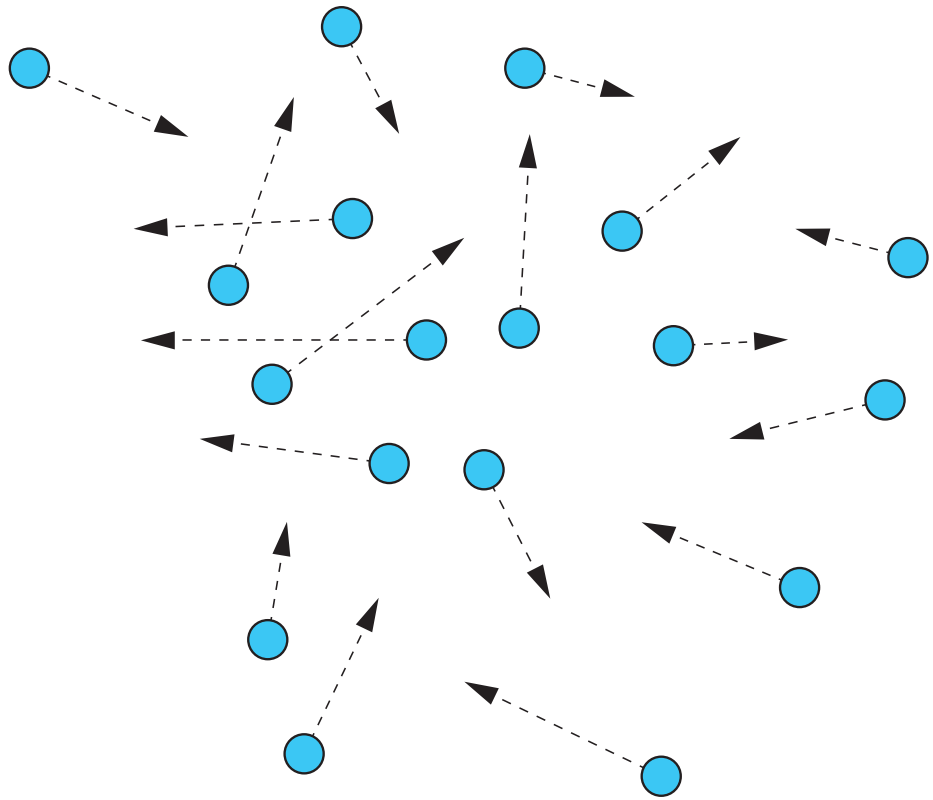
Problem: form a given pattern from any initial configuration

Pattern Formation problem



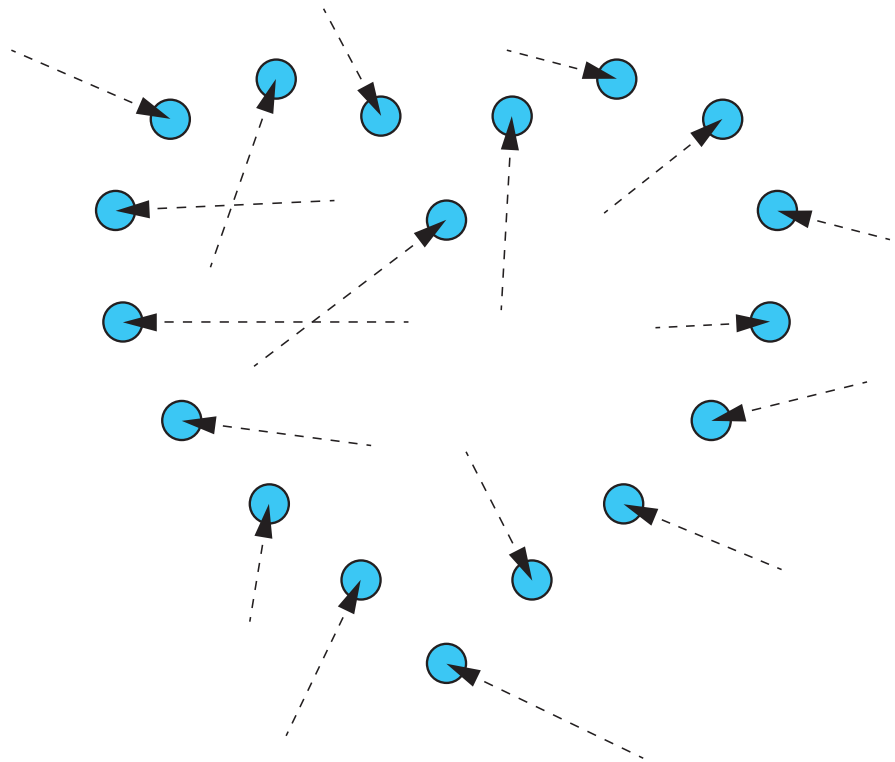
Problem: form a given pattern from any initial configuration

Pattern Formation problem



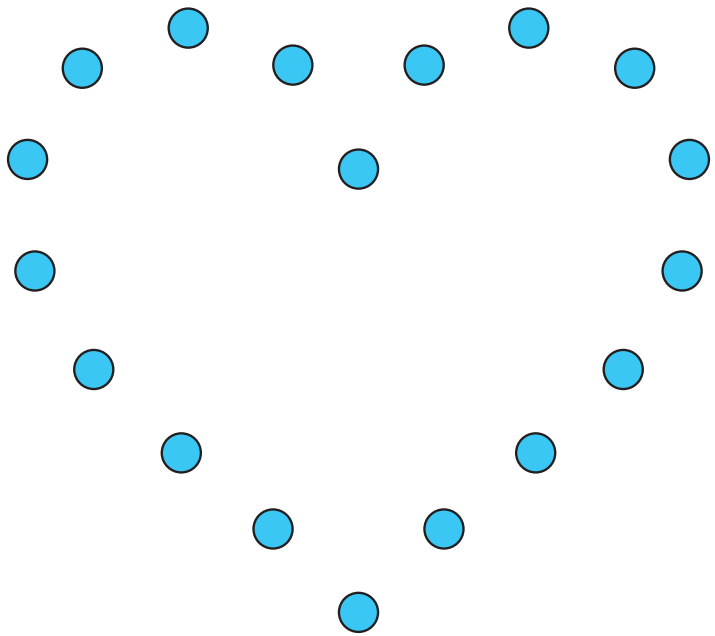
Problem: form a given pattern from any initial configuration

Pattern Formation problem



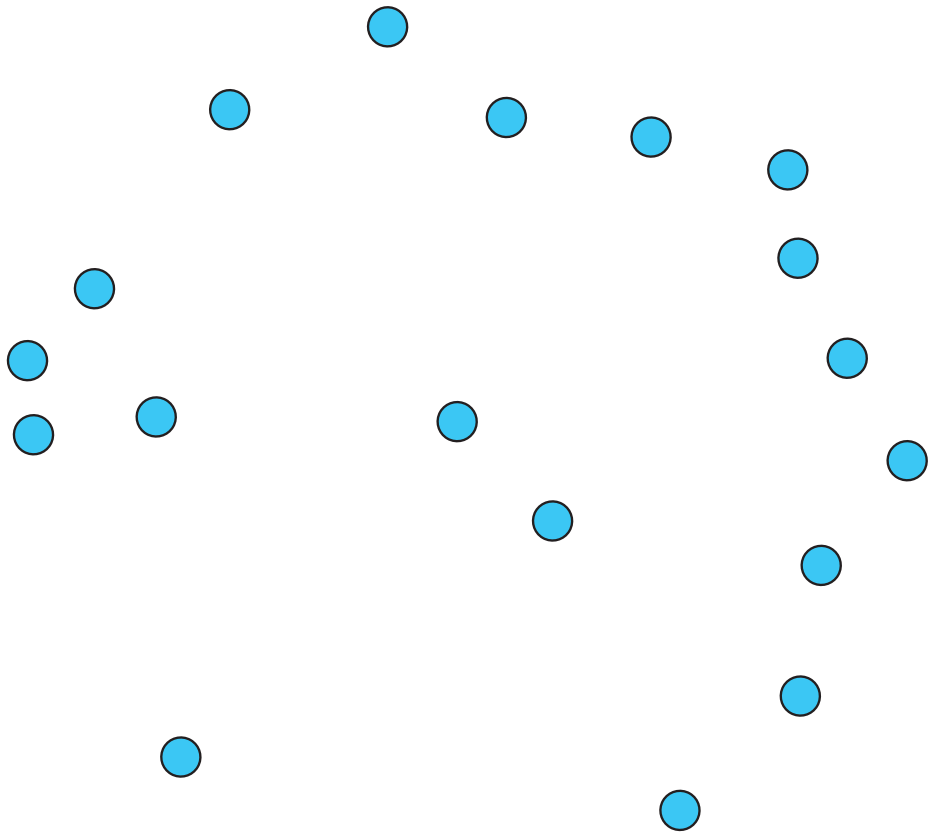
Problem: form a given pattern from any initial configuration

Pattern Formation problem



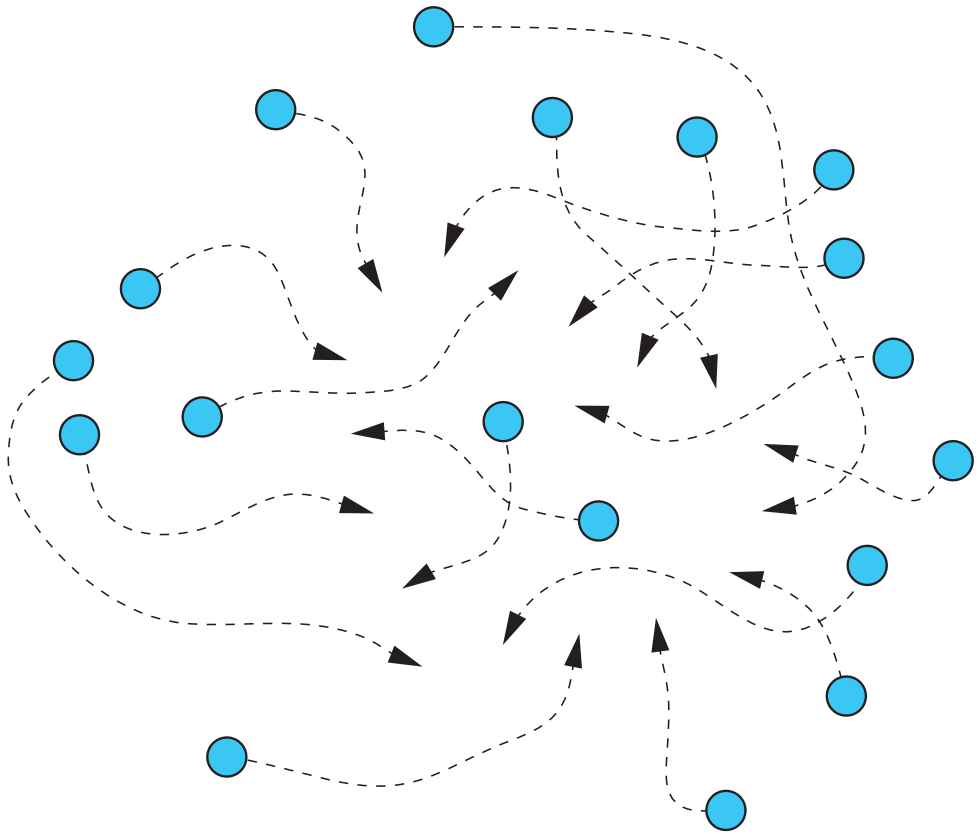
Problem: form a given pattern from any initial configuration

Pattern Formation problem



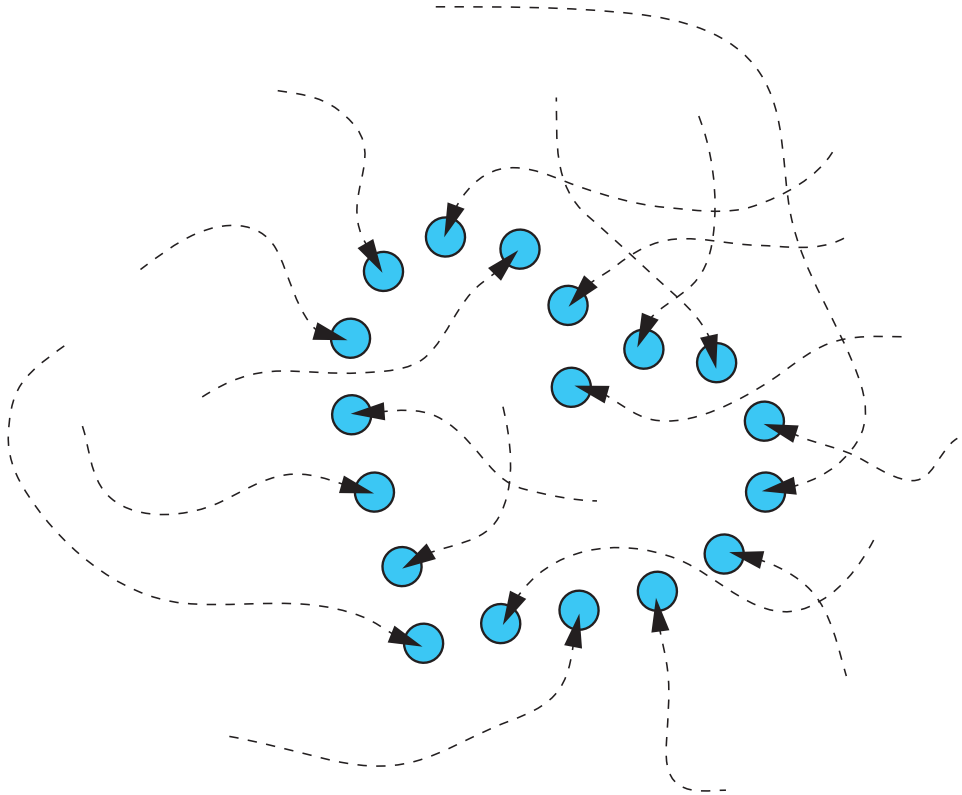
The pattern may be rotated, reflected, and scaled

Pattern Formation problem



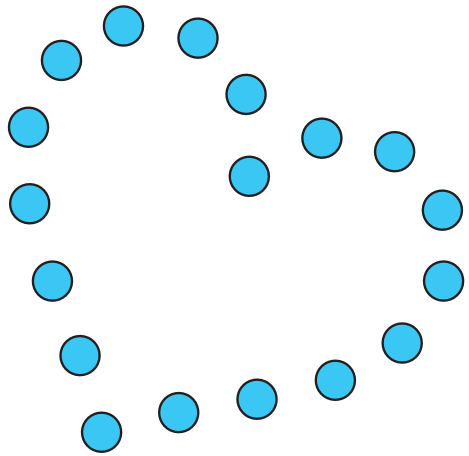
The pattern may be rotated, reflected, and scaled

Pattern Formation problem



The pattern may be rotated, reflected, and scaled

Pattern Formation problem



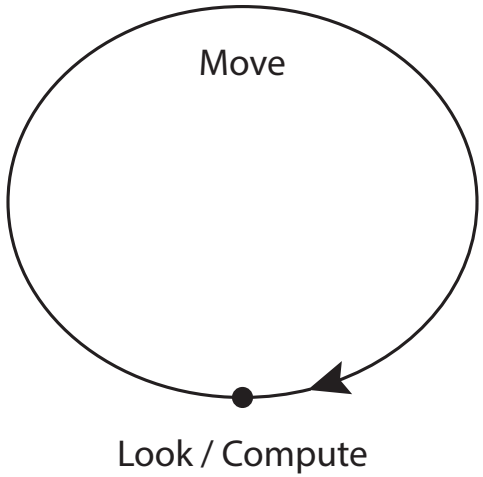
The pattern may be rotated, reflected, and scaled

Model definition

Robots are:

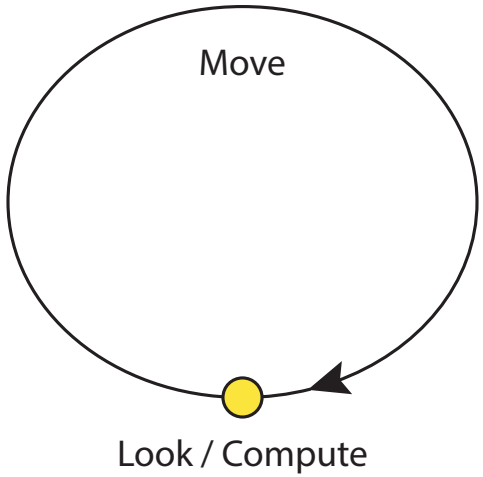
- **Dimensionless** (robots are modeled as geometric points)
- **Anonymous** (no unique identifiers)
- **Homogeneous** (the same algorithm is executed by all robots)
- **Autonomous** (no centralized control)
- **Oblivious** (no memory of past events)
- **Silent** (no explicit way of communicating)
- **Long-sighted** (complete visibility of all other robots)
- **Disoriented** (robots do not share a common reference frame, and a robot's reference frame may change from turn to turn)
  - No common unit distance
  - No common compass
  - No common notion of clockwise direction

Life cycles and asynchronicity



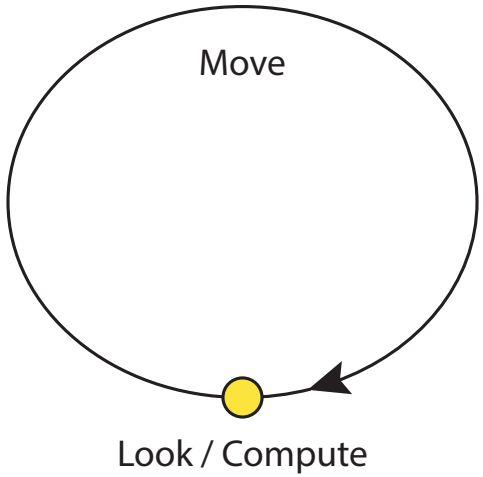
Each robot repeats a Look/Compute/Move cycle

Life cycles and asynchronicity



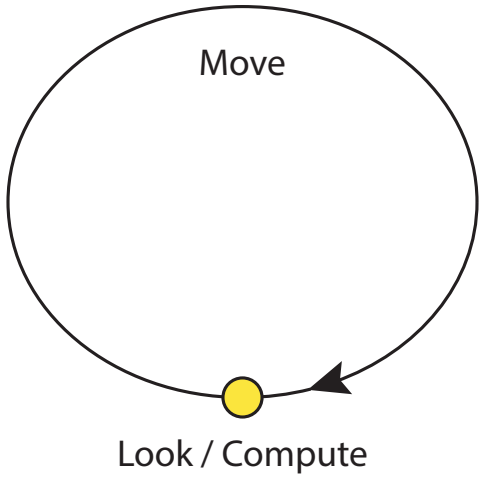
Each robot repeats a Look/Compute/Move cycle

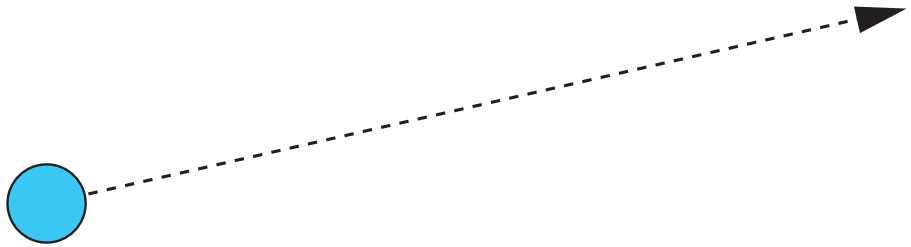
Life cycles and asynchronicity



In a Look phase, an instantaneous snapshot is taken of all robots

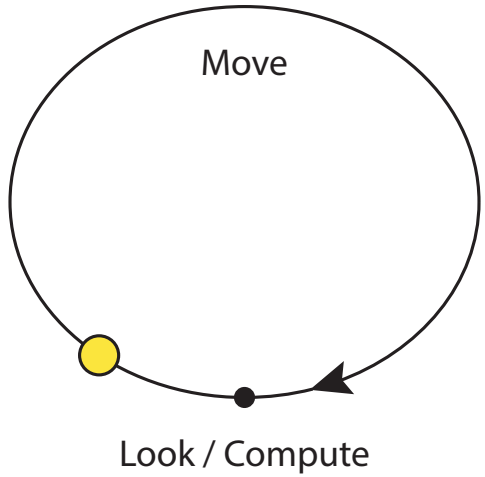
Life cycles and asynchronicity

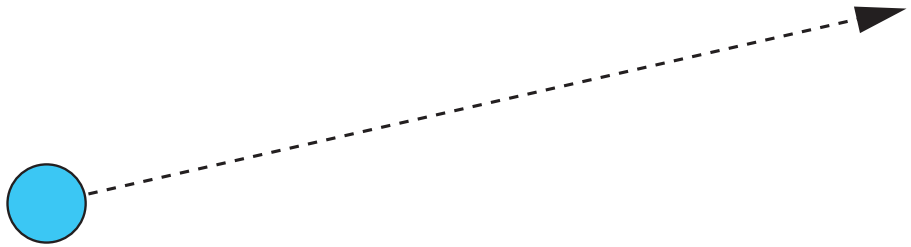




A destination point is computed as a function of the snapshot

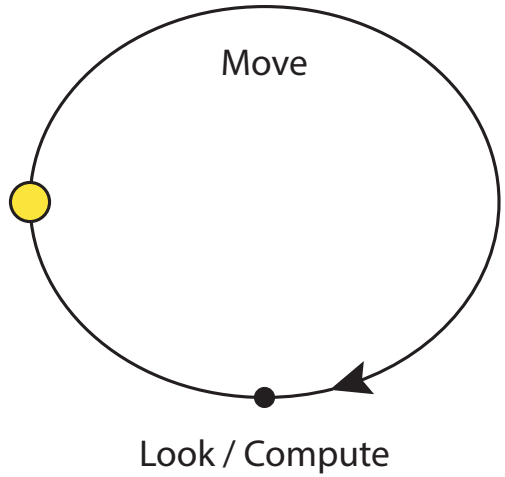
Life cycles and asynchronicity

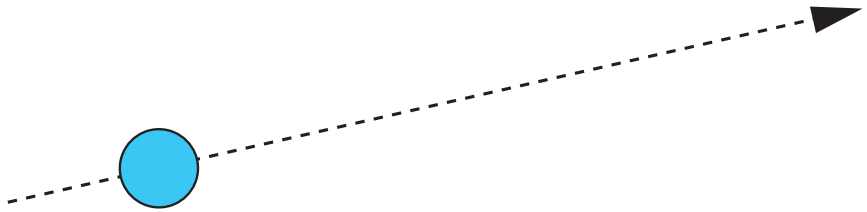




The destination point is approached with unpredictable speed

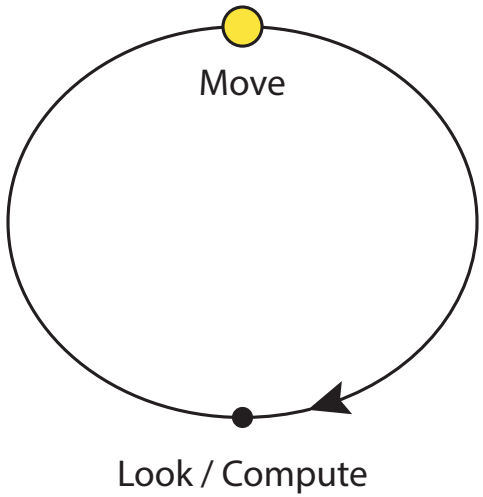
Life cycles and asynchronicity

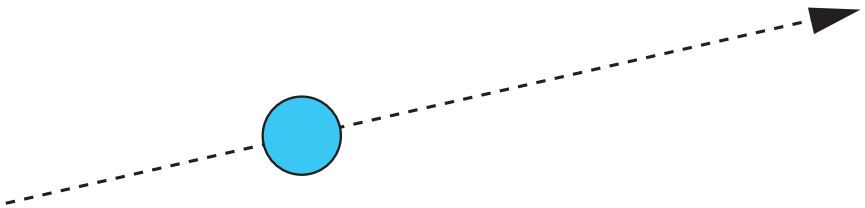




The destination point is approached with unpredictable speed

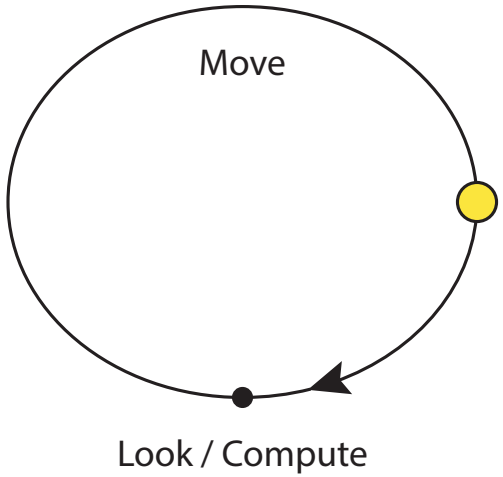
Life cycles and asynchronicity

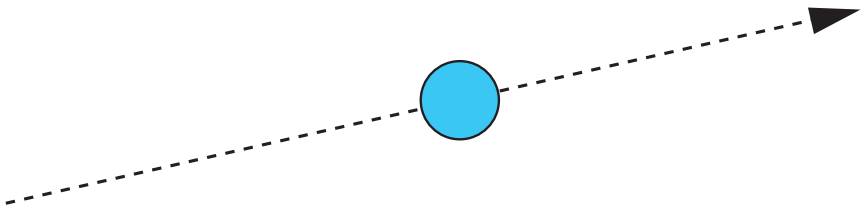




The destination point is approached with unpredictable speed

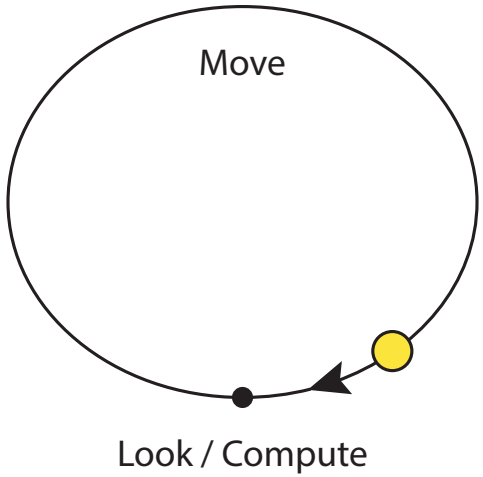
Life cycles and asynchronicity

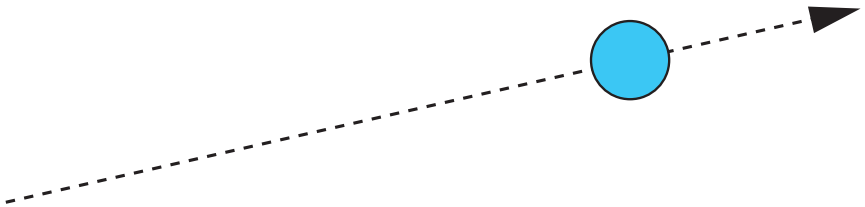




The destination point is approached with unpredictable speed

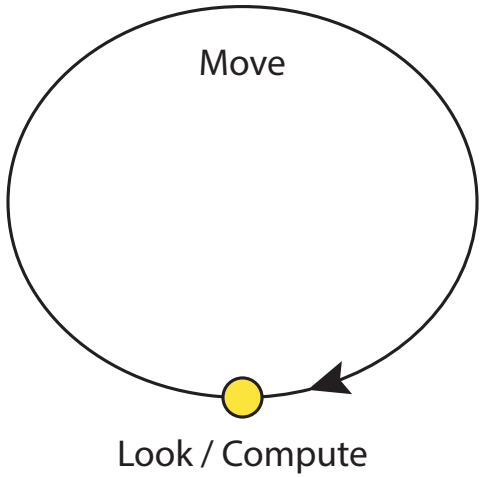
Life cycles and asynchronicity

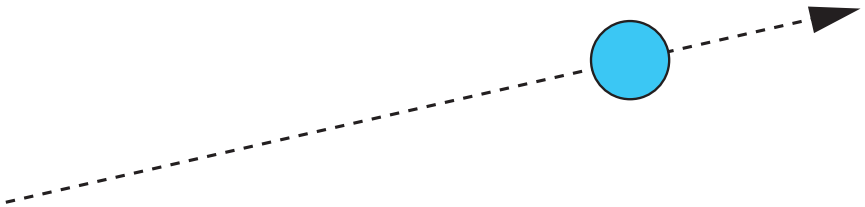




The destination point is approached with unpredictable speed

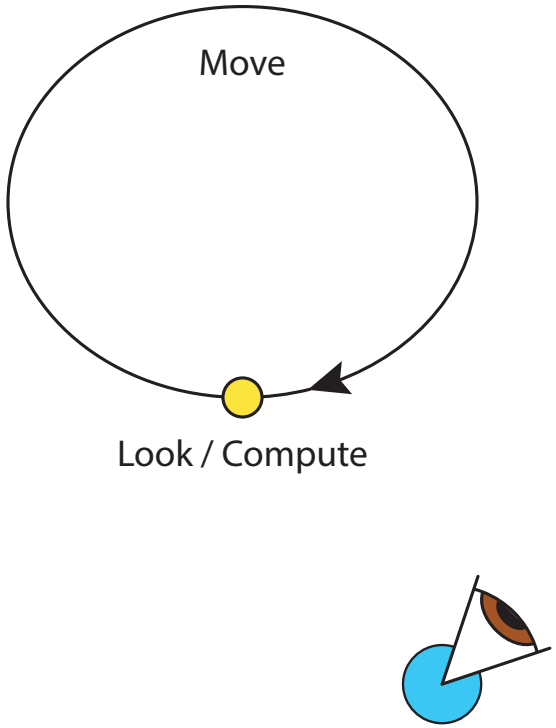
Life cycles and asynchronicity





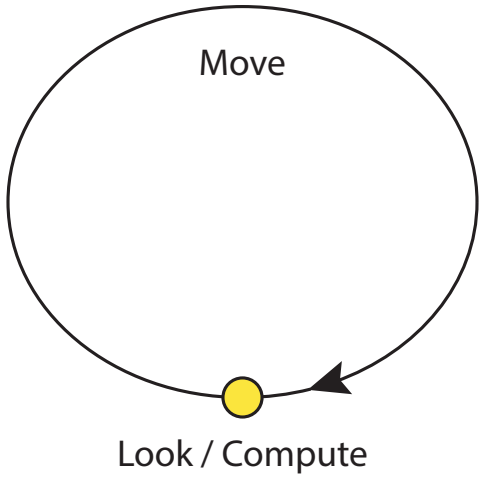
The robot may unpredictably stop before reaching the destination...

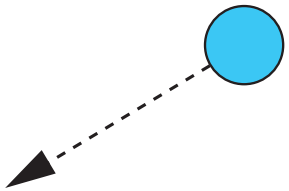
Life cycles and asynchronicity



...and execute a new Look/Compute phase

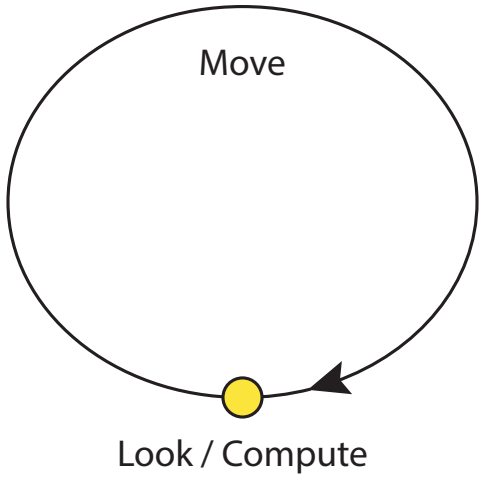
Life cycles and asynchronicity

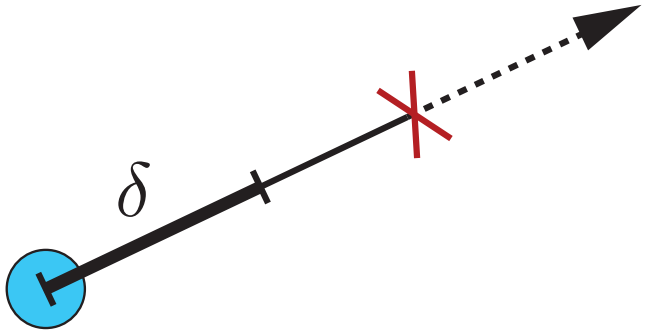




...and execute a new Look/Compute phase

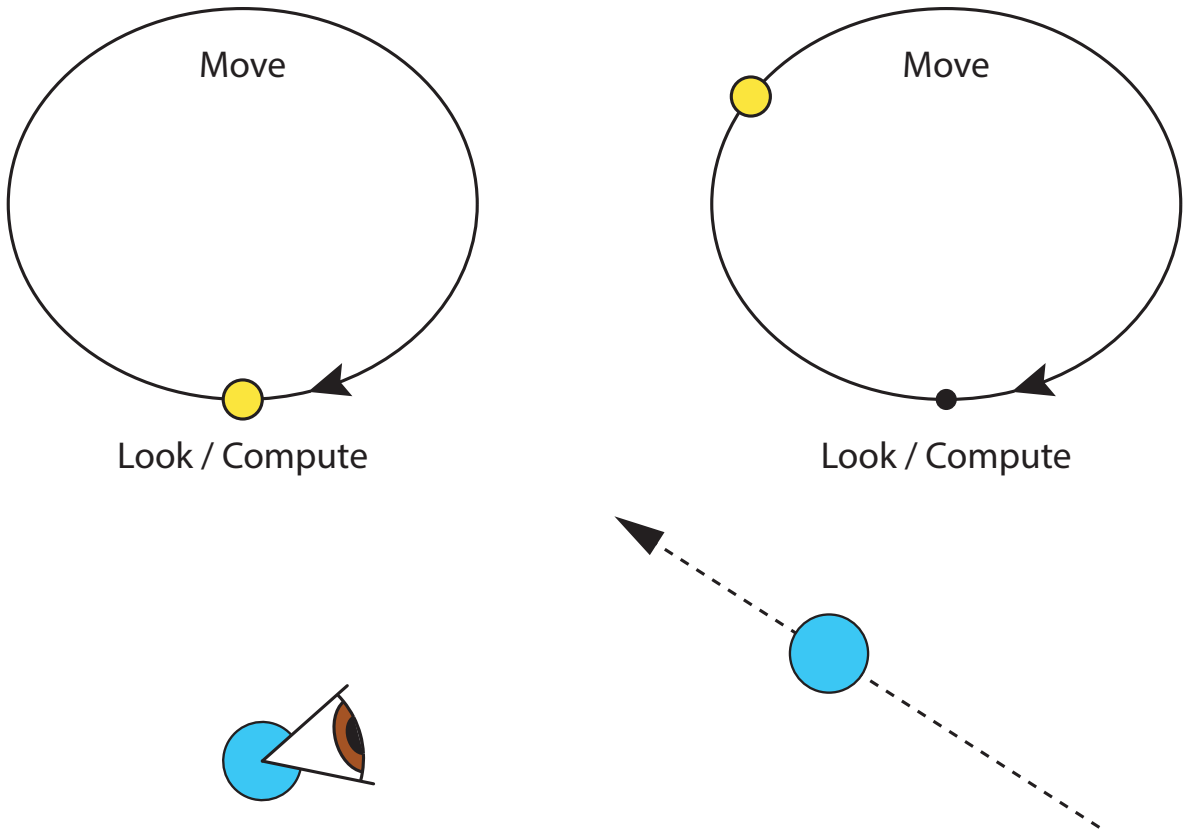
Life cycles and asynchronicity





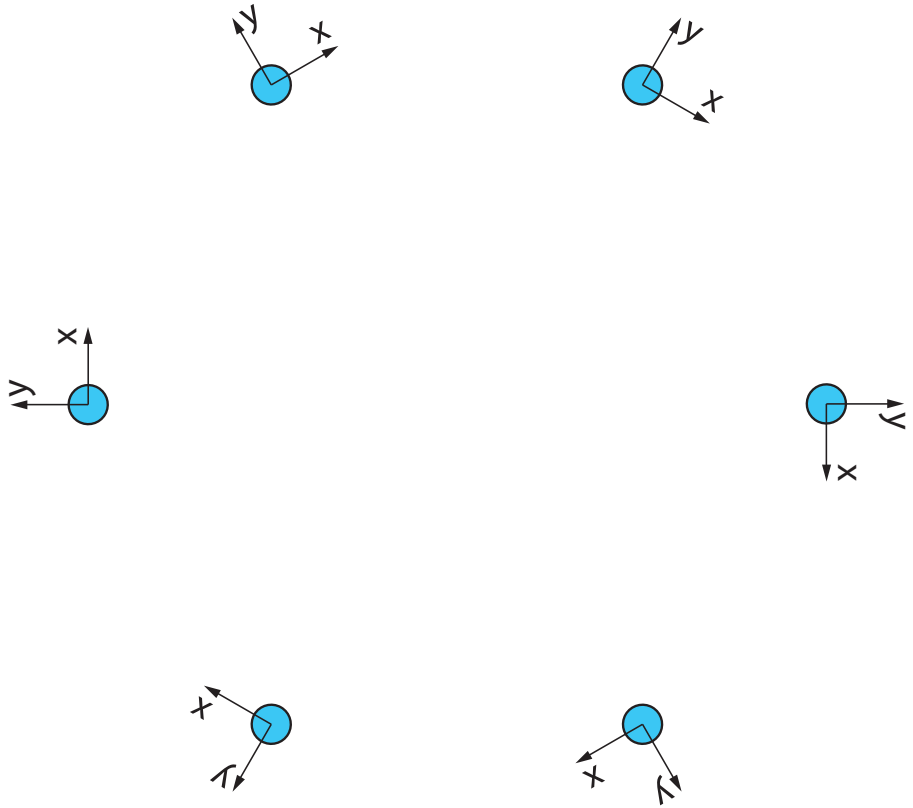
At each cycle, a robot is guaranteed to move by at least δ

Life cycles and asynchronicity



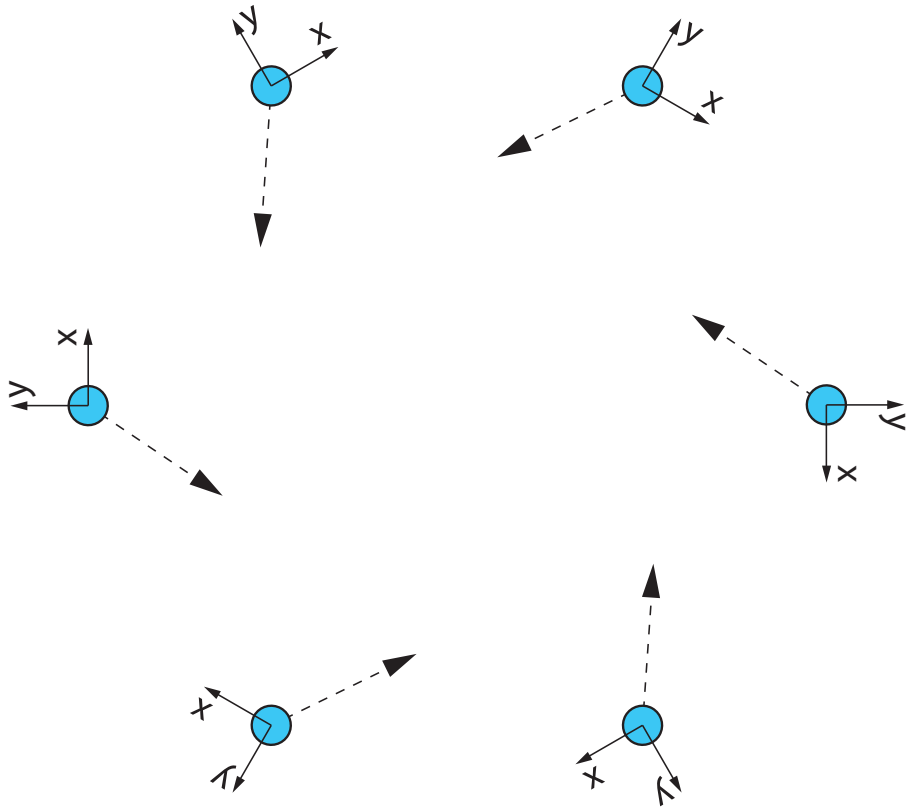
Different robots execute independent cycles, asynchronously

Pattern Formation problem: counterexample



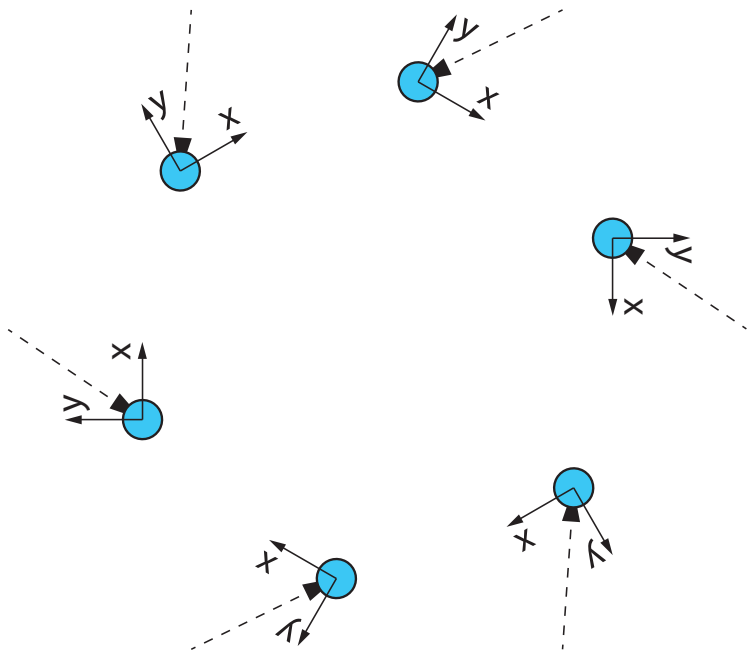
Let the initial configuration be rotationally symmetric

Pattern Formation problem: counterexample



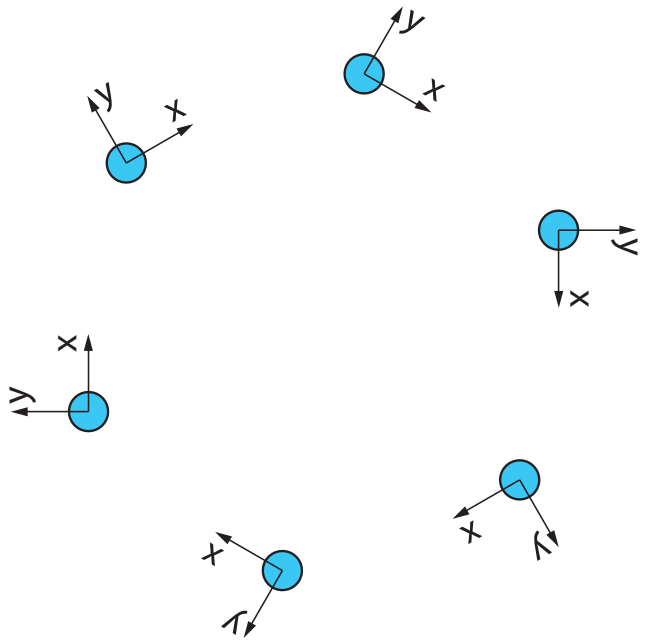
All robots have the same view and compute symmetric destinations

Pattern Formation problem: counterexample



If they are all activated synchronously, they remain symmetric

Pattern Formation problem: counterexample

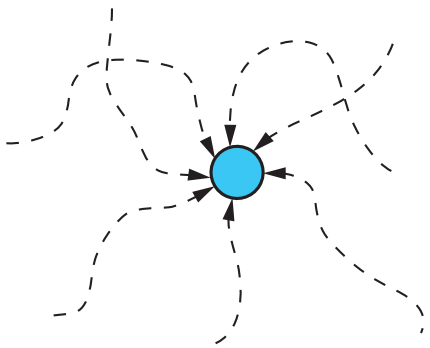


Hence Pattern Formation is *unsolvable* if the pattern is *asymmetric*

Pattern Formation problem: state of the art

No pattern is formable from every possible initial configuration, except:

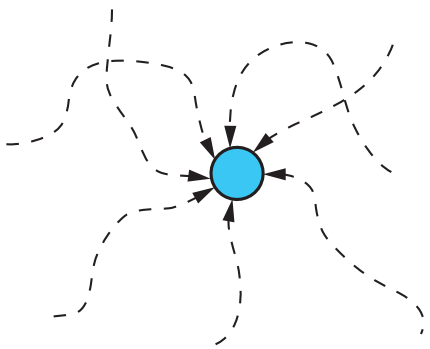
- **Single point** (aka Gathering problem)
  $\Longrightarrow$ Solved [Cieliebak-Flocchini-Prencipe-Santoro, 2012]

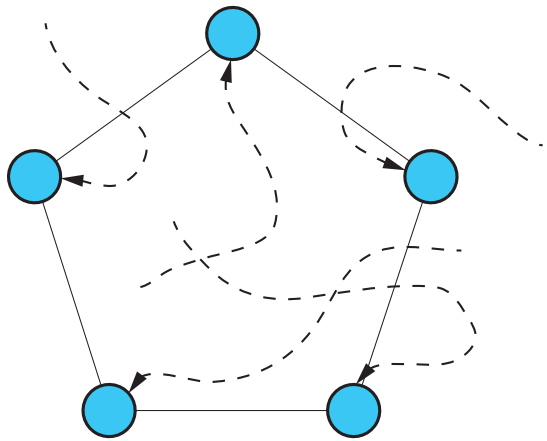


Pattern Formation problem: state of the art

No pattern is formable from every possible initial configuration, except:

- **Single point** (aka Gathering problem)
  $\Longrightarrow$ Solved [Cieliebak-Flocchini-Prencipe-Santoro, 2012]



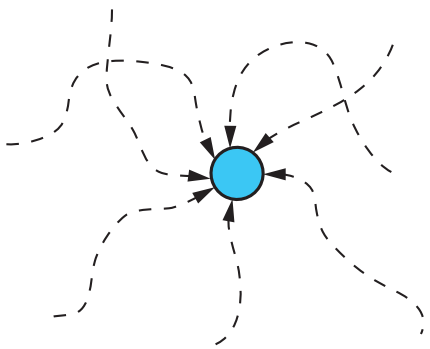


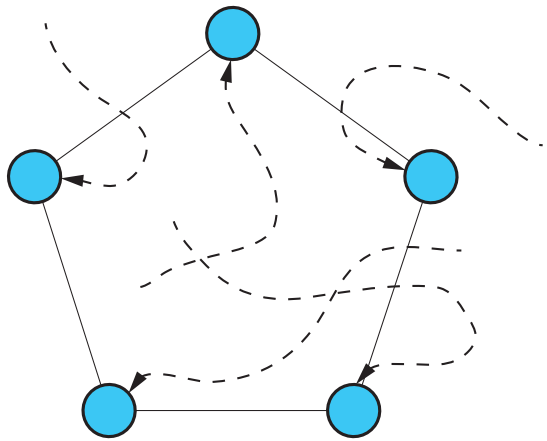
- **Regular polygon**
  $\Longrightarrow$ Solved... [Flocchini-Prencipe-Santoro-Viglietta, 2014–15]

Pattern Formation problem: state of the art

No pattern is formable from every possible initial configuration, except:

- **Single point** (aka Gathering problem)
  $\Longrightarrow$ Solved [Cieliebak-Flocchini-Prencipe-Santoro, 2012]

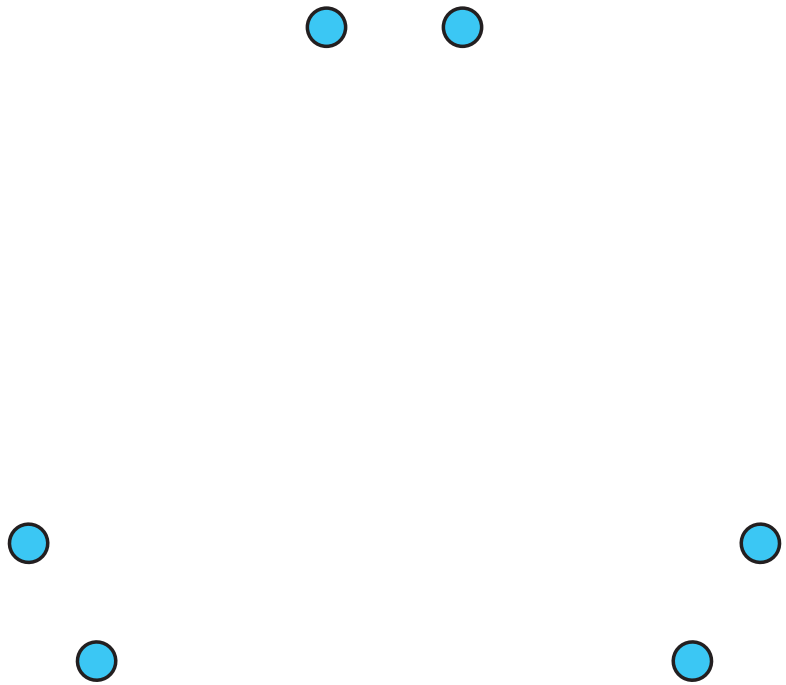




- **Regular polygon**
  $\Longrightarrow$ Solved... [Flocchini-Prencipe-Santoro-Viglietta, 2014–15]

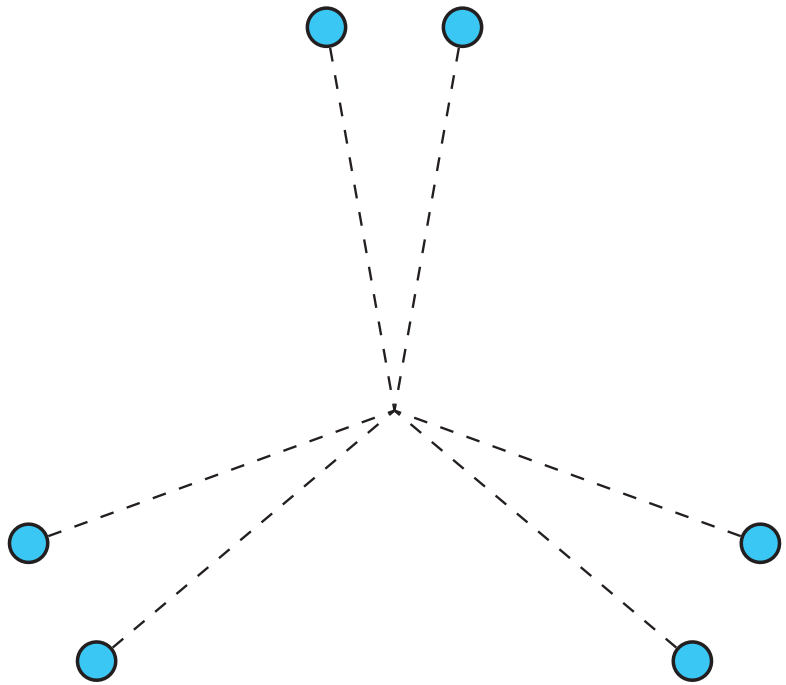
...*except for 4 robots!* (aka Square Formation problem)

General approach to forming a regular polygon



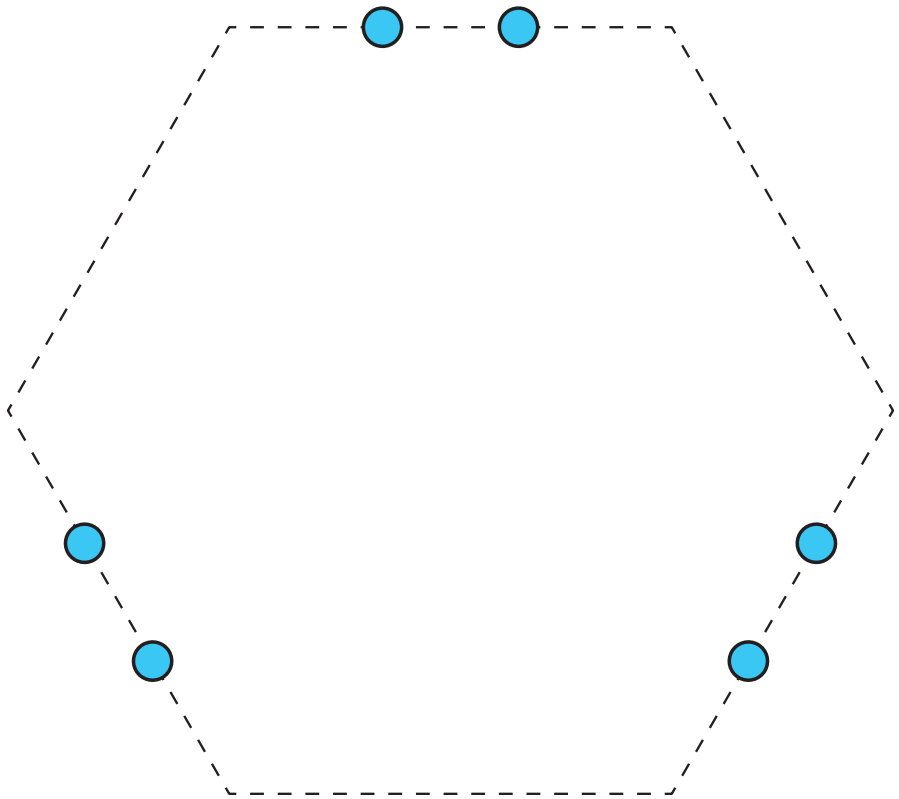
An important configuration is the *biangular* one

General approach to forming a regular polygon



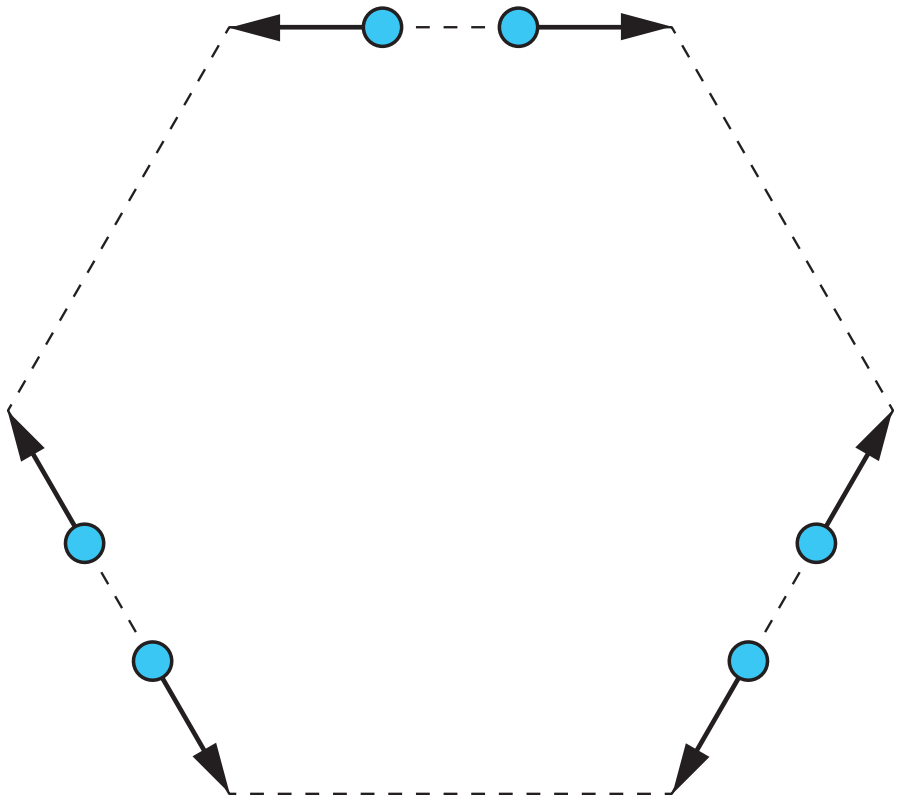
An important configuration is the *biangular* one

General approach to forming a regular polygon



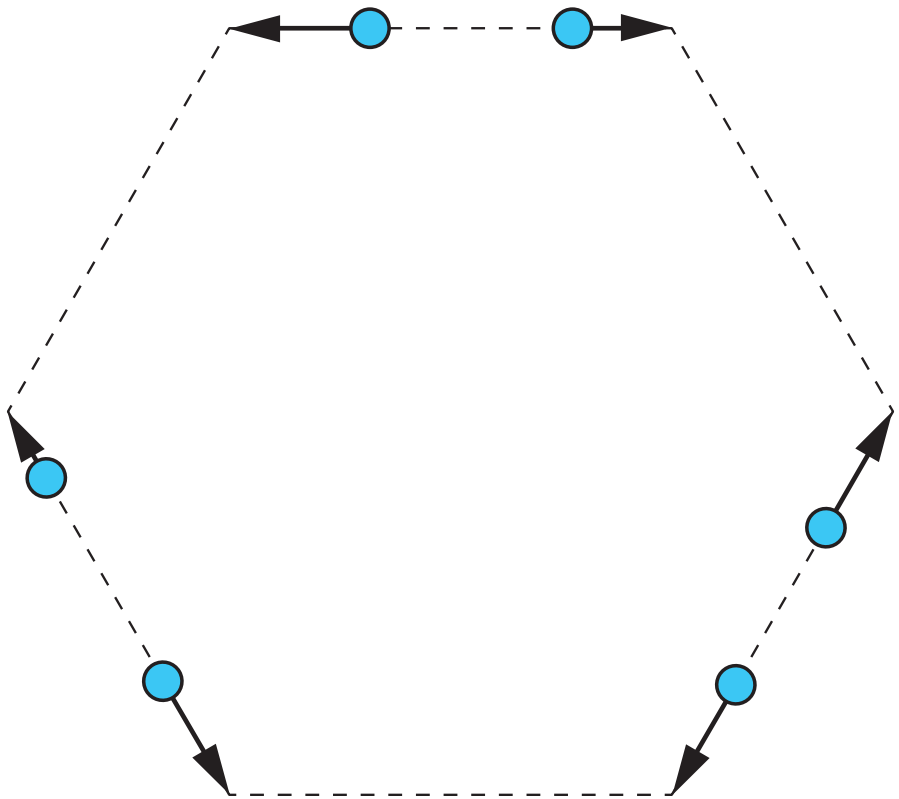
The general algorithm identifies a *supporting polygon*...

General approach to forming a regular polygon



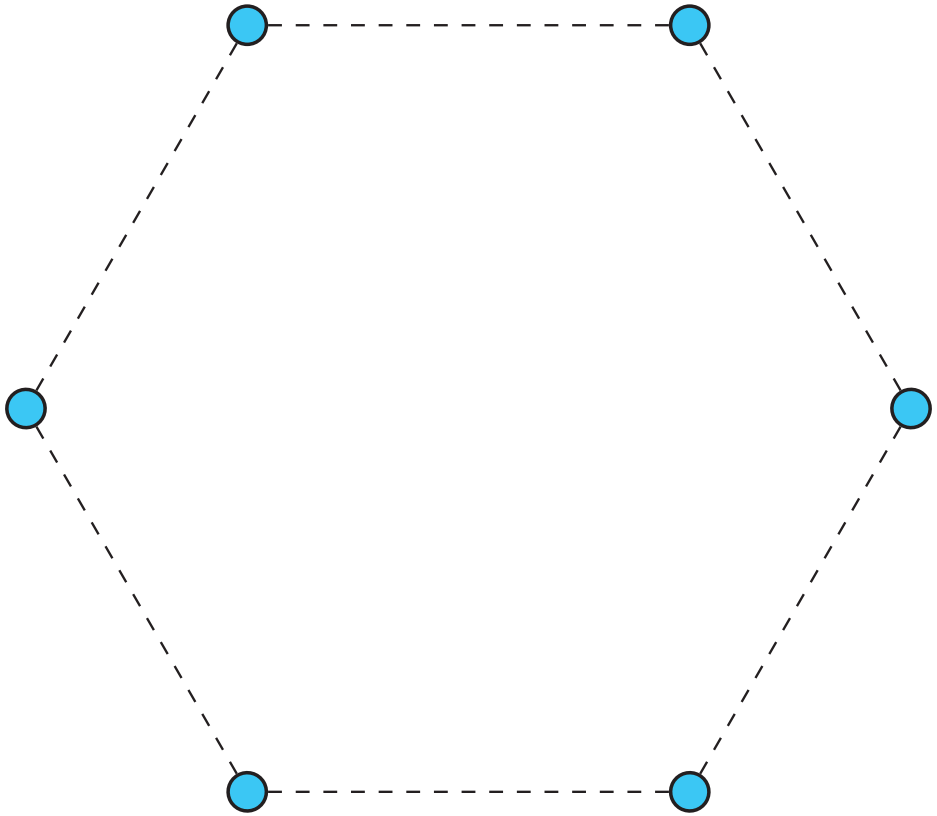
...And makes each robot move to the closest vertex

General approach to forming a regular polygon



As robots move, the supporting polygon is preserved

General approach to forming a regular polygon

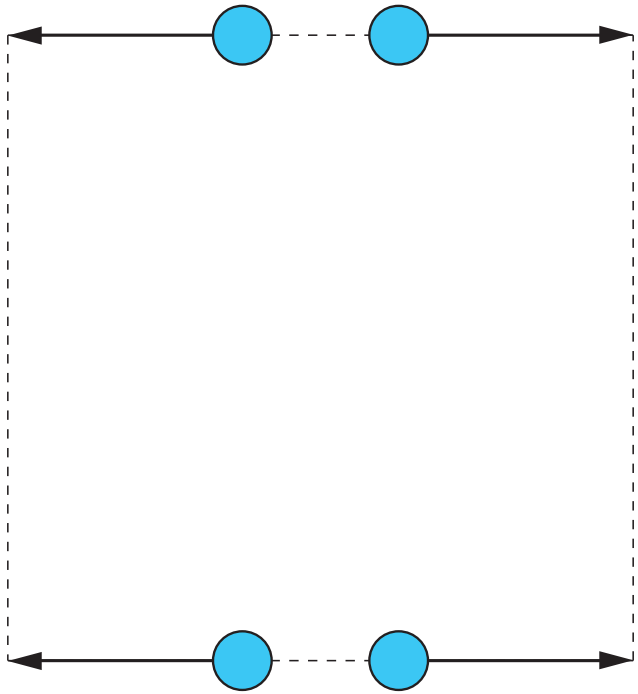


As robots move, the supporting polygon is preserved

Why the general approach fails with 4 robots

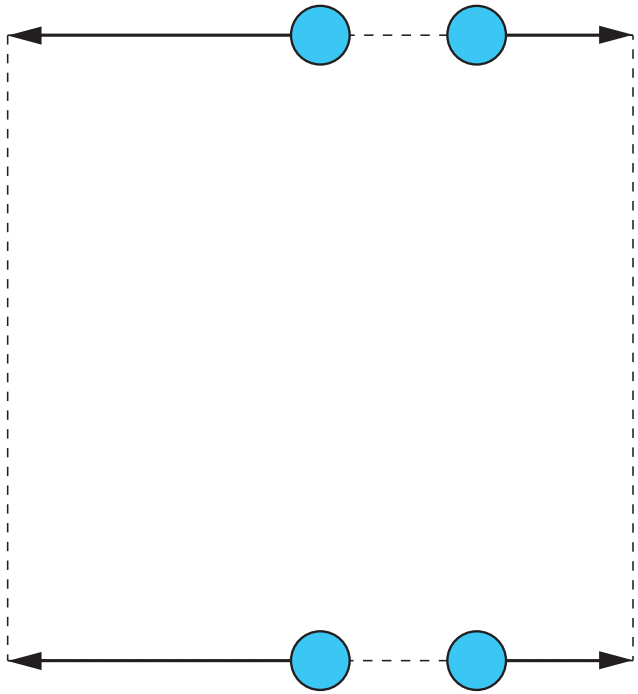
With 4 robots, biangular configurations are rectangles

Why the general approach fails with 4 robots



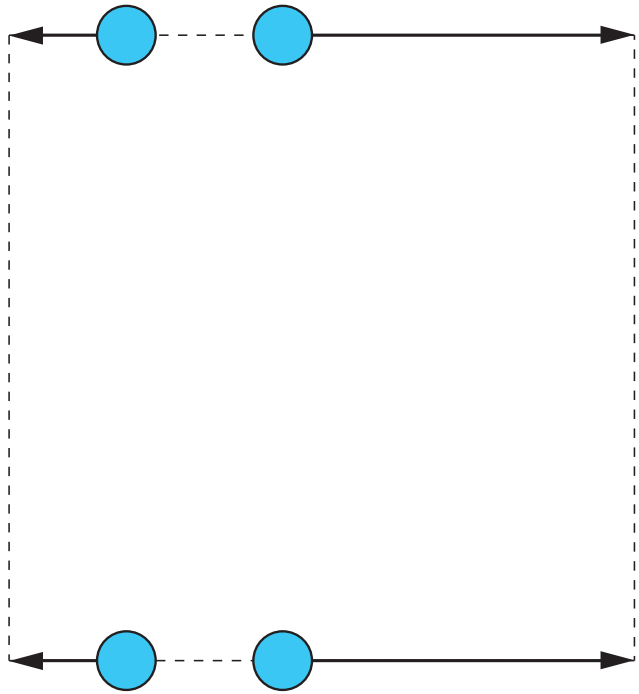
We can still identify a supporting square...

Why the general approach fails with 4 robots



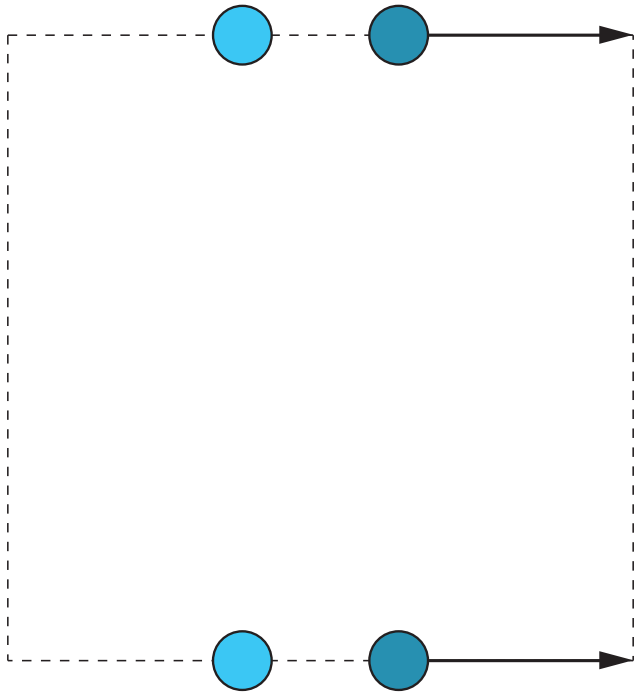
...But it is not unique!

Why the general approach fails with 4 robots



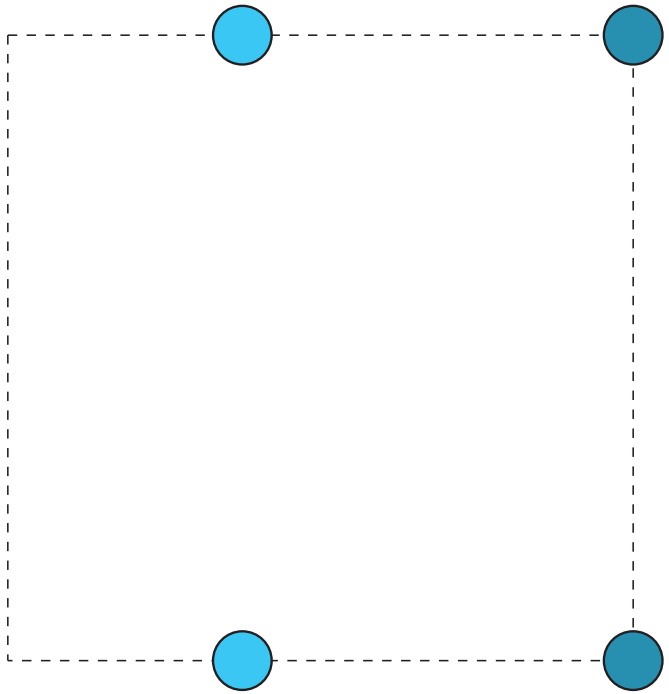
...But it is not unique!

Why the general approach fails with 4 robots



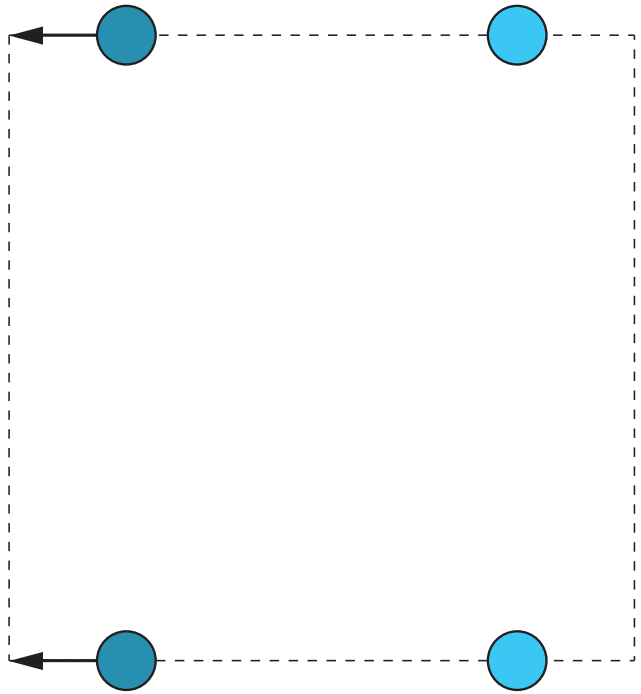
The “central” supporting polygon may be chosen...

Why the general approach fails with 4 robots



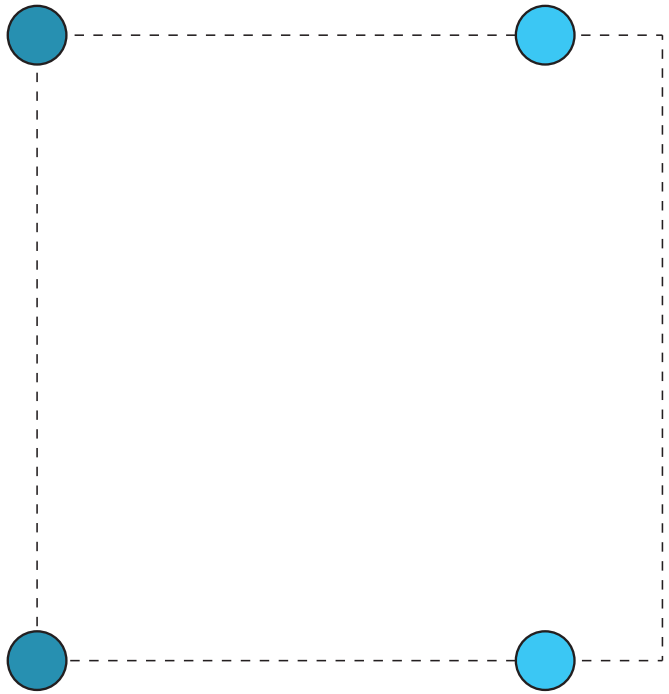
...But asynchronous robots may never manage to form a square

Why the general approach fails with 4 robots



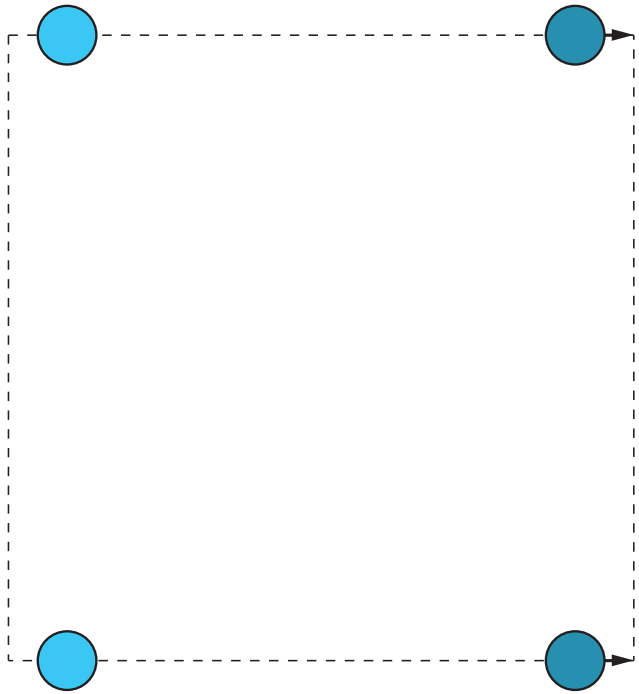
...But asynchronous robots may never manage to form a square

Why the general approach fails with 4 robots



...But asynchronous robots may never manage to form a square

Why the general approach fails with 4 robots

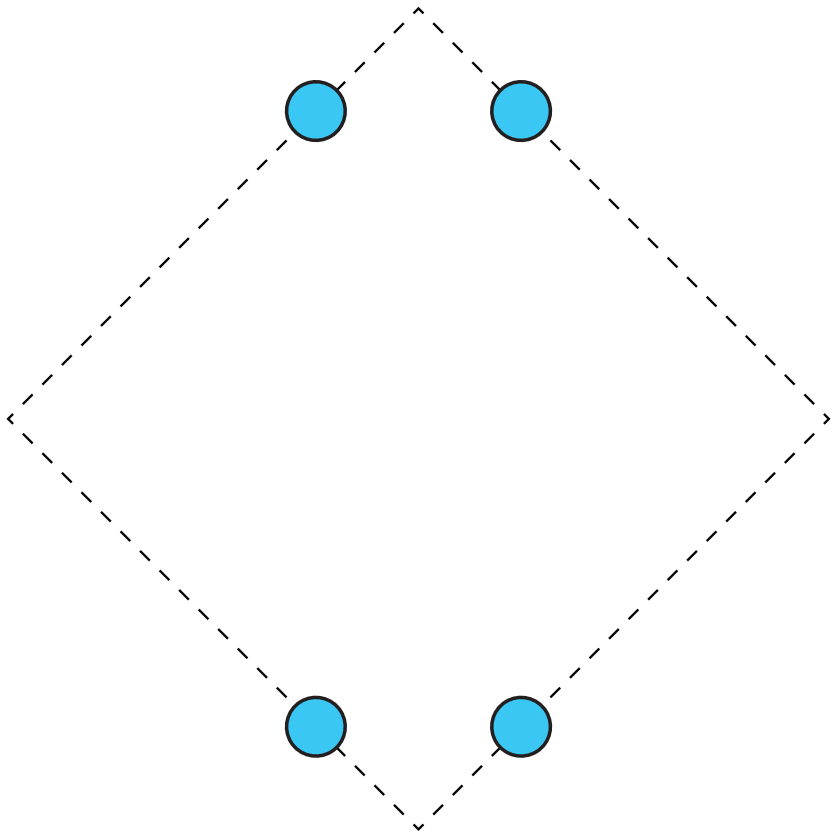


...But asynchronous robots may never manage to form a square

How to solve the rectangle

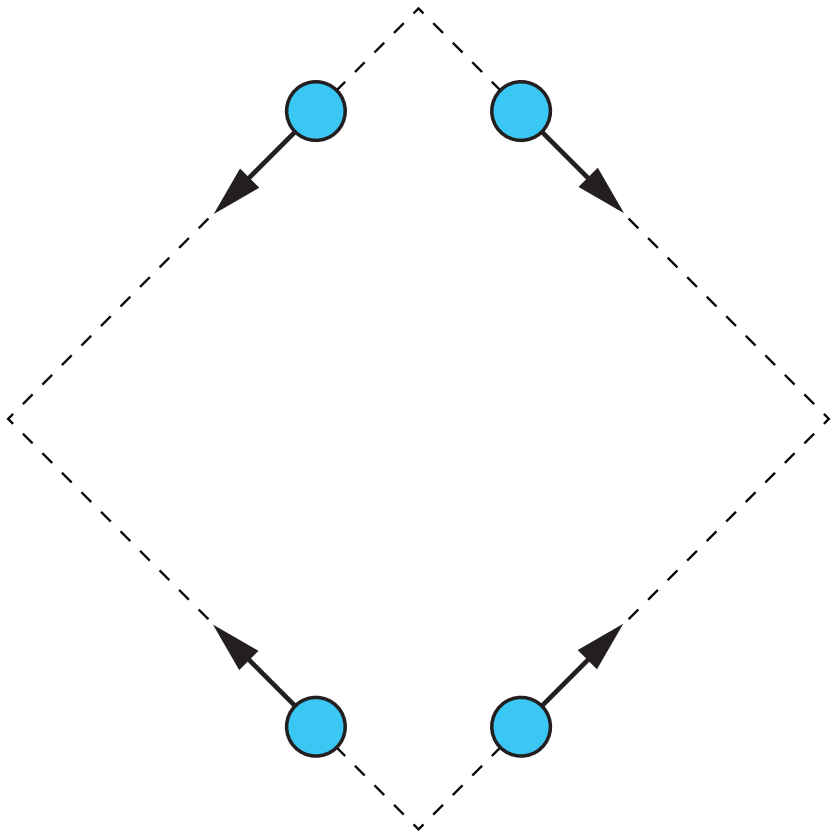
How do we solve the rectangular case?

How to solve the rectangle



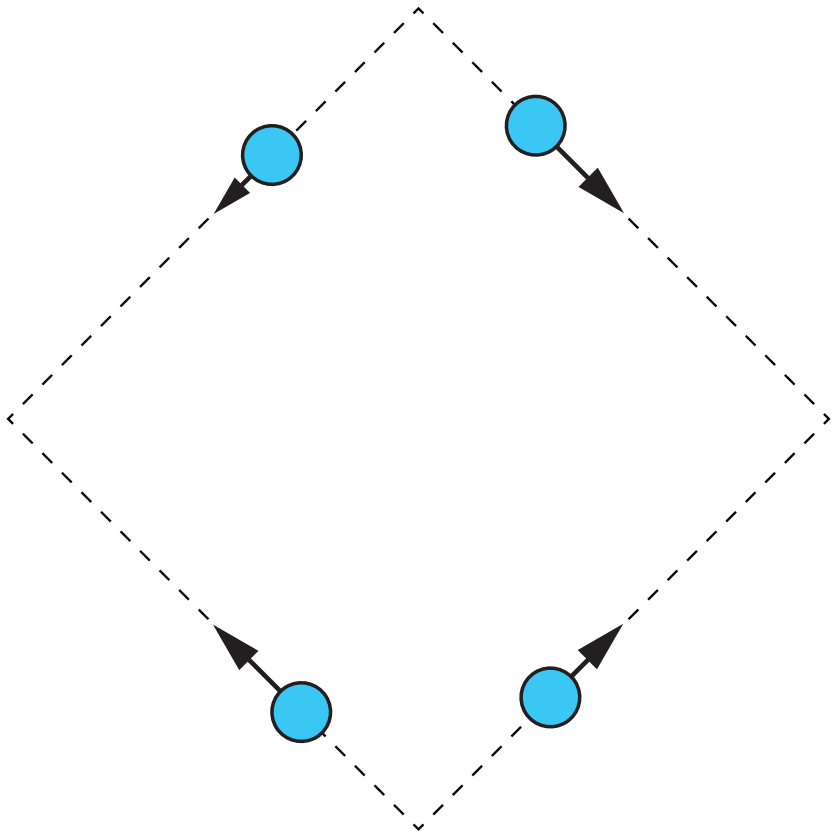
Choose a supporting square that is tilted by 45°...

How to solve the rectangle



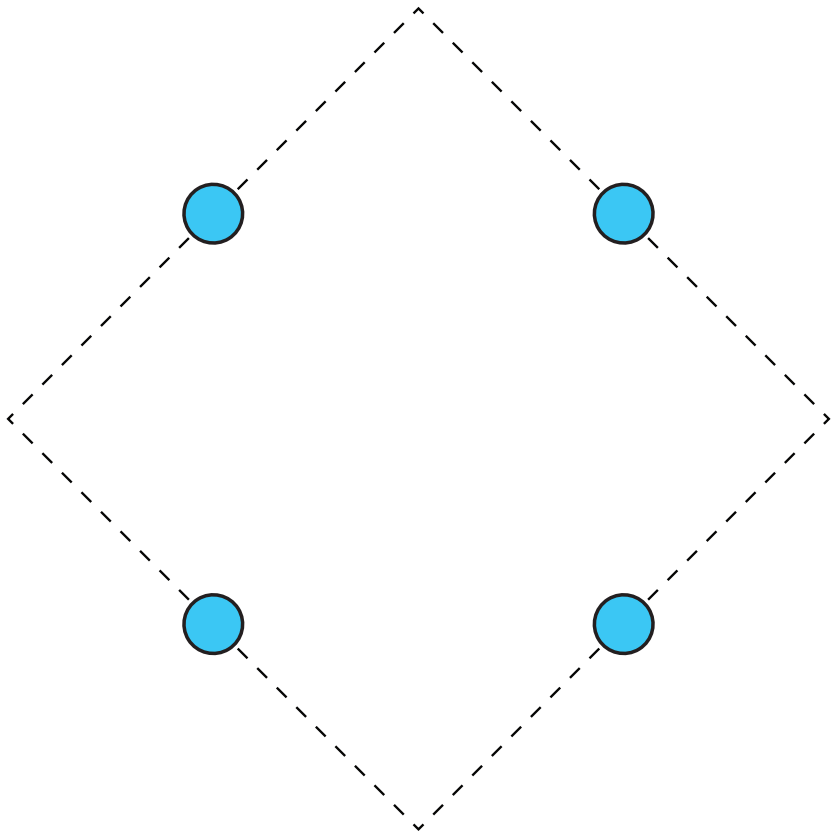
...And make the robots move to the midpoints of its edges

How to solve the rectangle



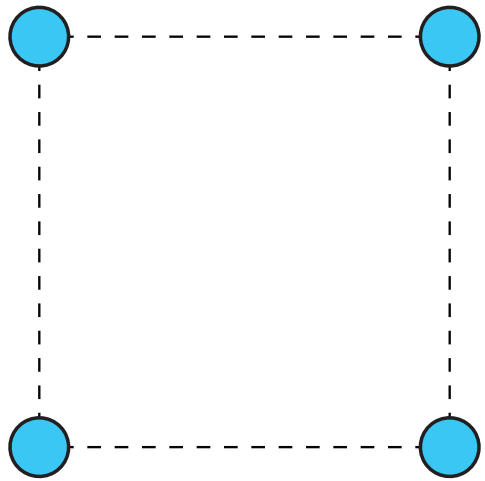
Again, the supporting square is preserved as the robots move

How to solve the rectangle



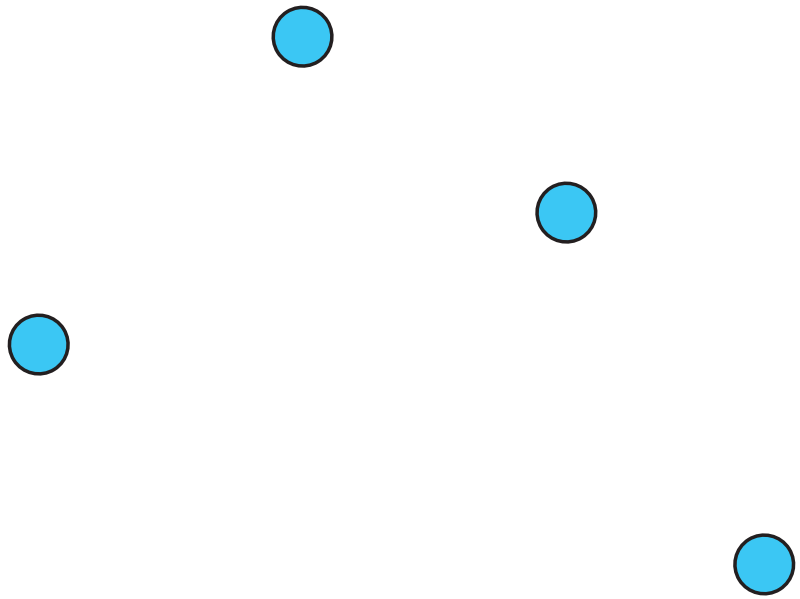
Again, the supporting square is preserved as the robots move

How to solve the rectangle



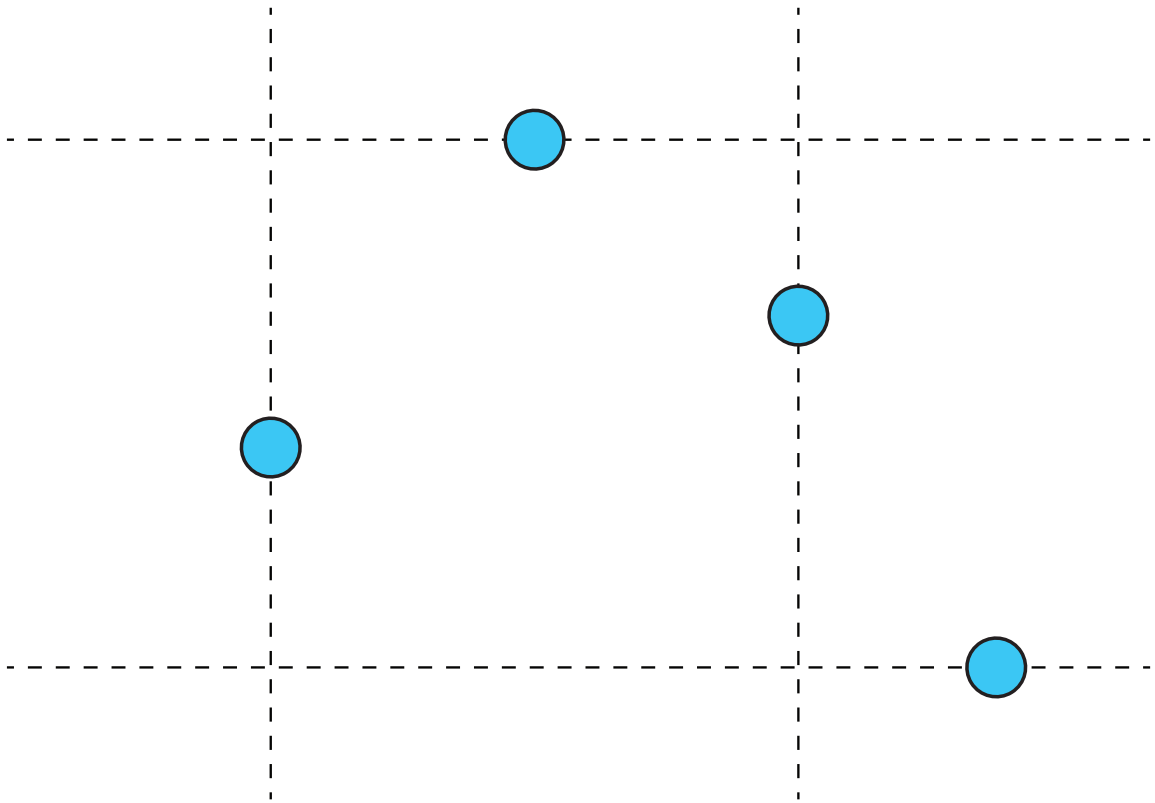
When they reach the midpoints, they form a square

Identifying the supporting square



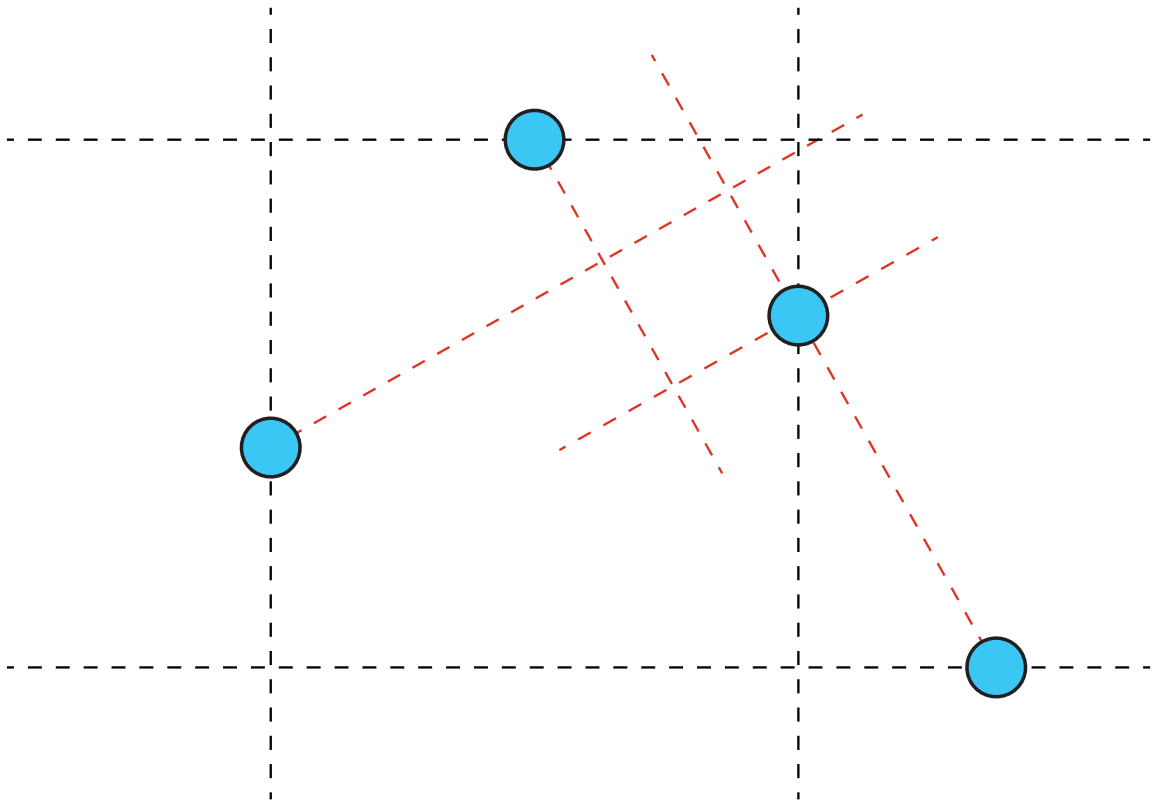
In general, we can also identify a supporting square...

Identifying the supporting square



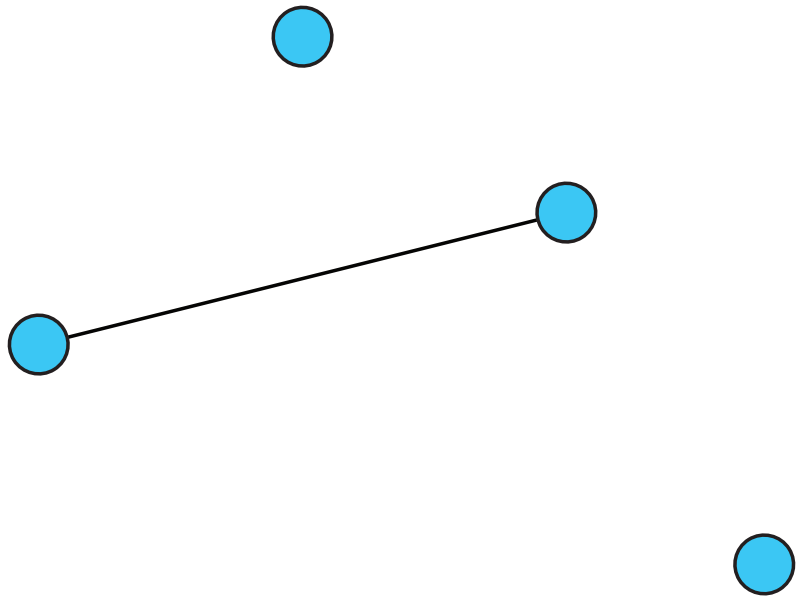
...Having a robot on each (extended) edge

Identifying the supporting square



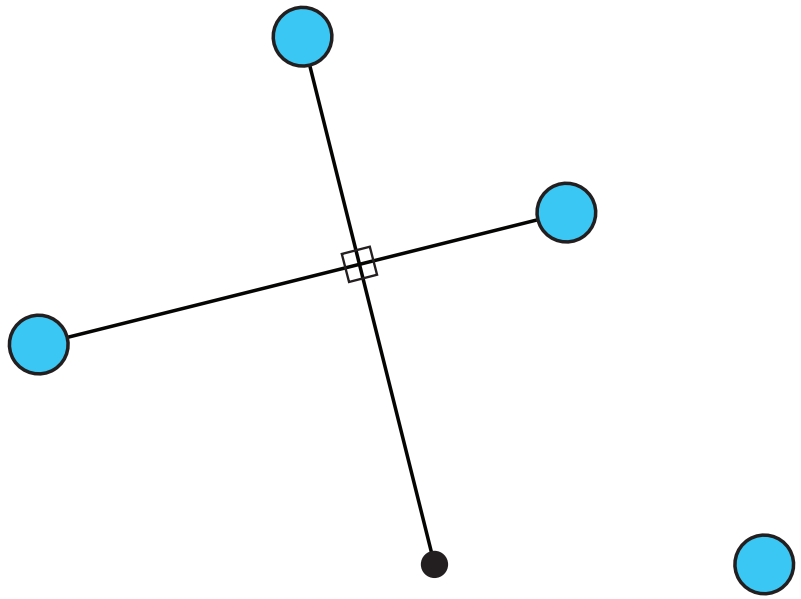
But once again, the supporting square is not unique!

Identifying the supporting square



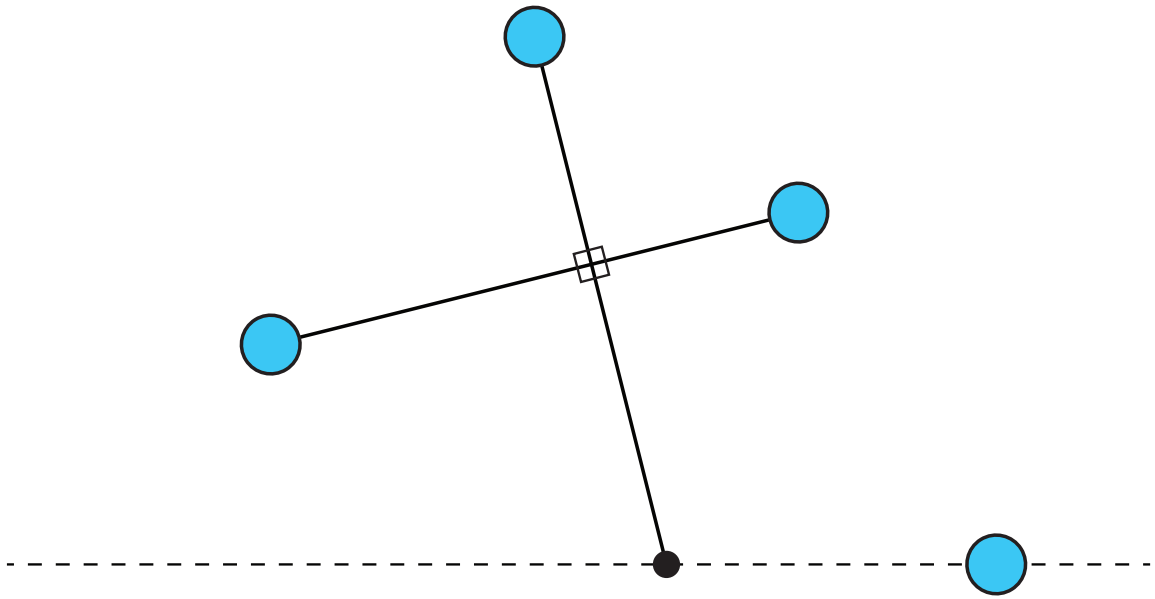
However, there is a geometric construction that identifies one

Identifying the supporting square



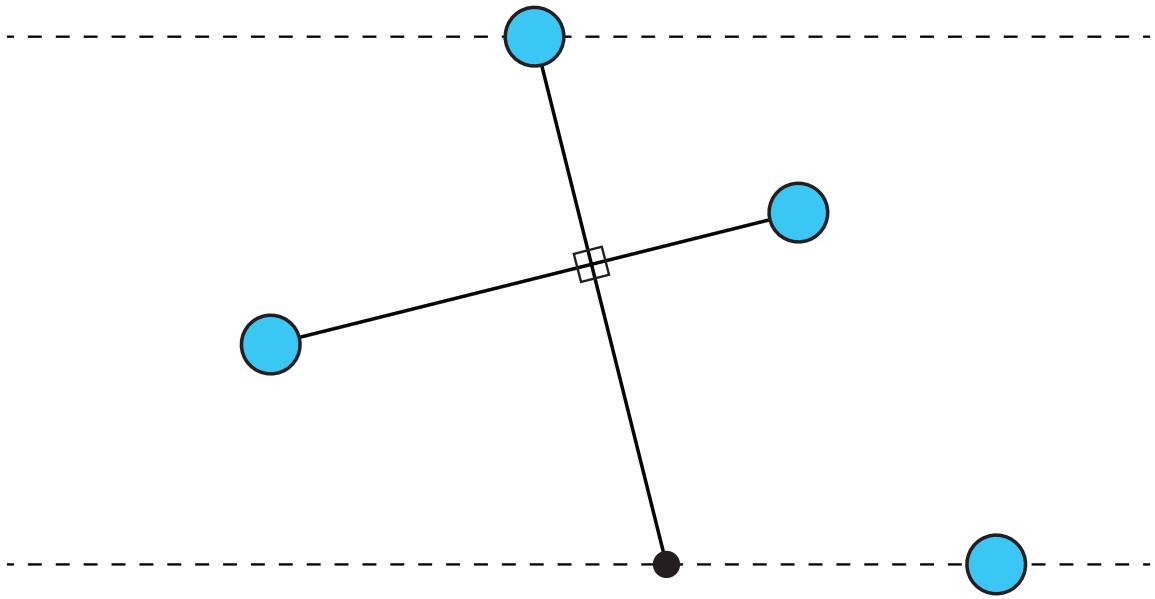
However, there is a geometric construction that identifies one

Identifying the supporting square



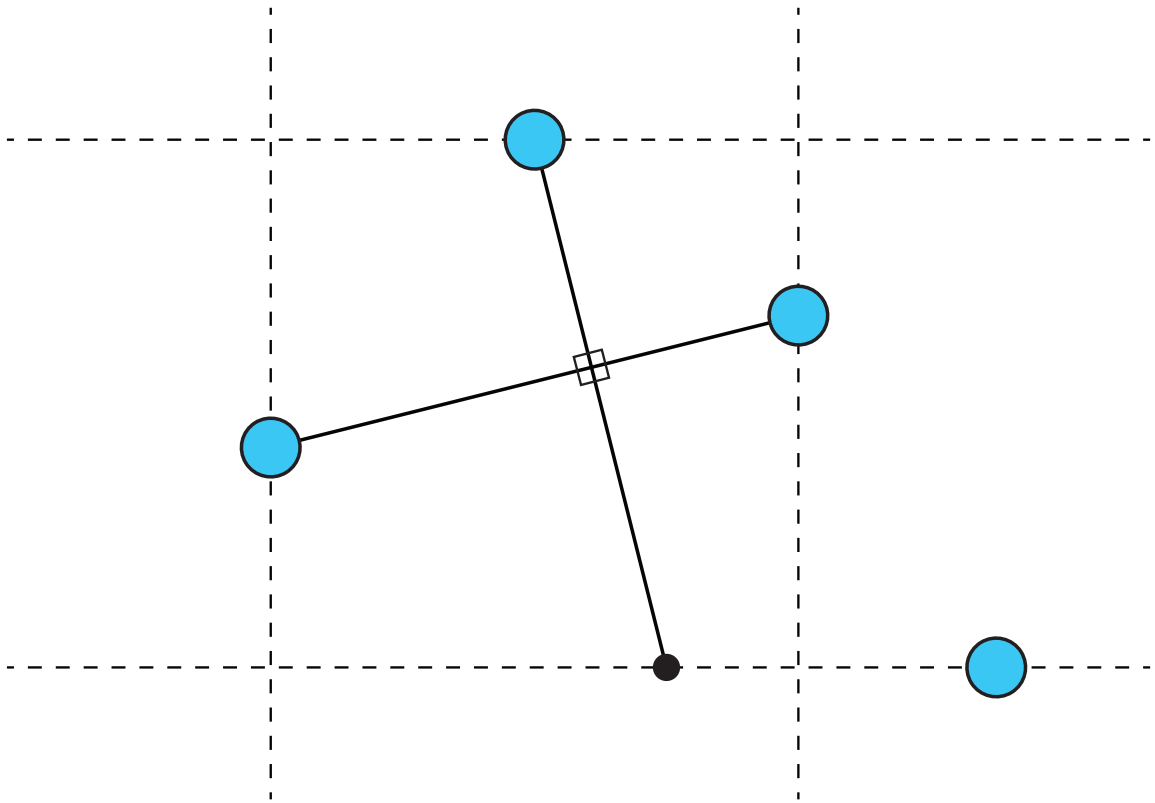
However, there is a geometric construction that identifies one

Identifying the supporting square



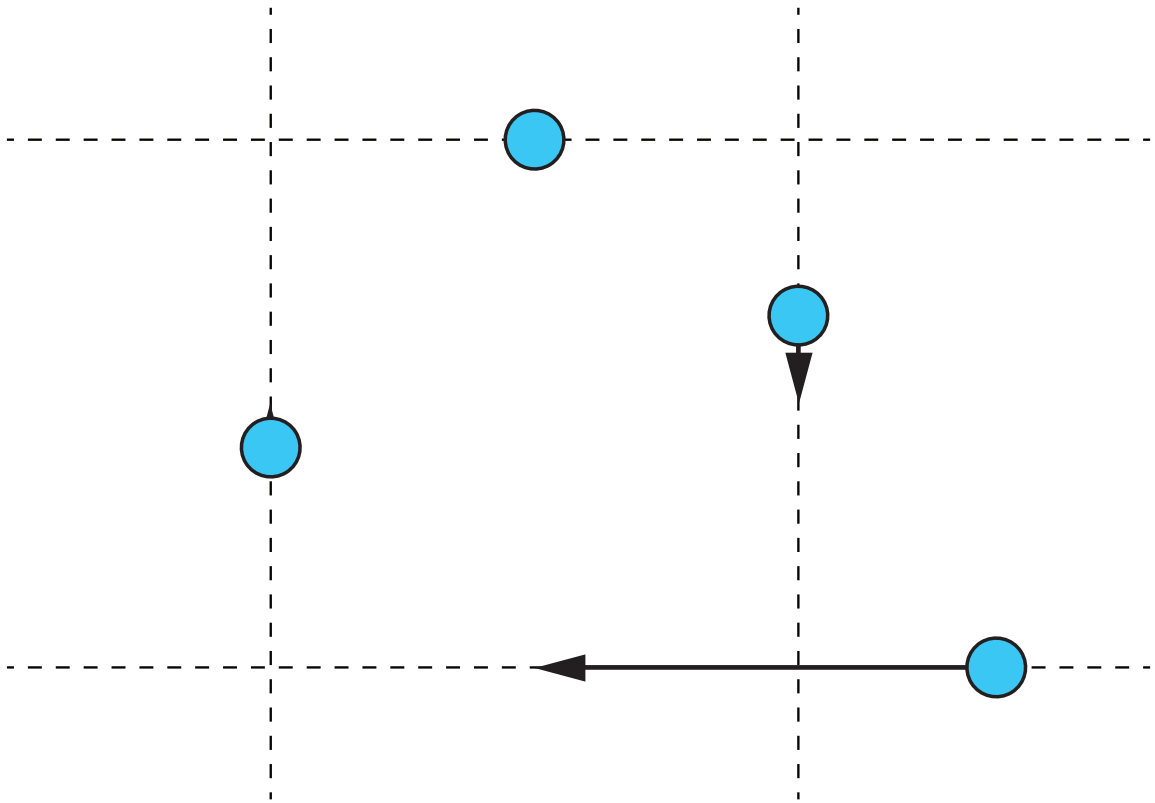
However, there is a geometric construction that identifies one

Identifying the supporting square



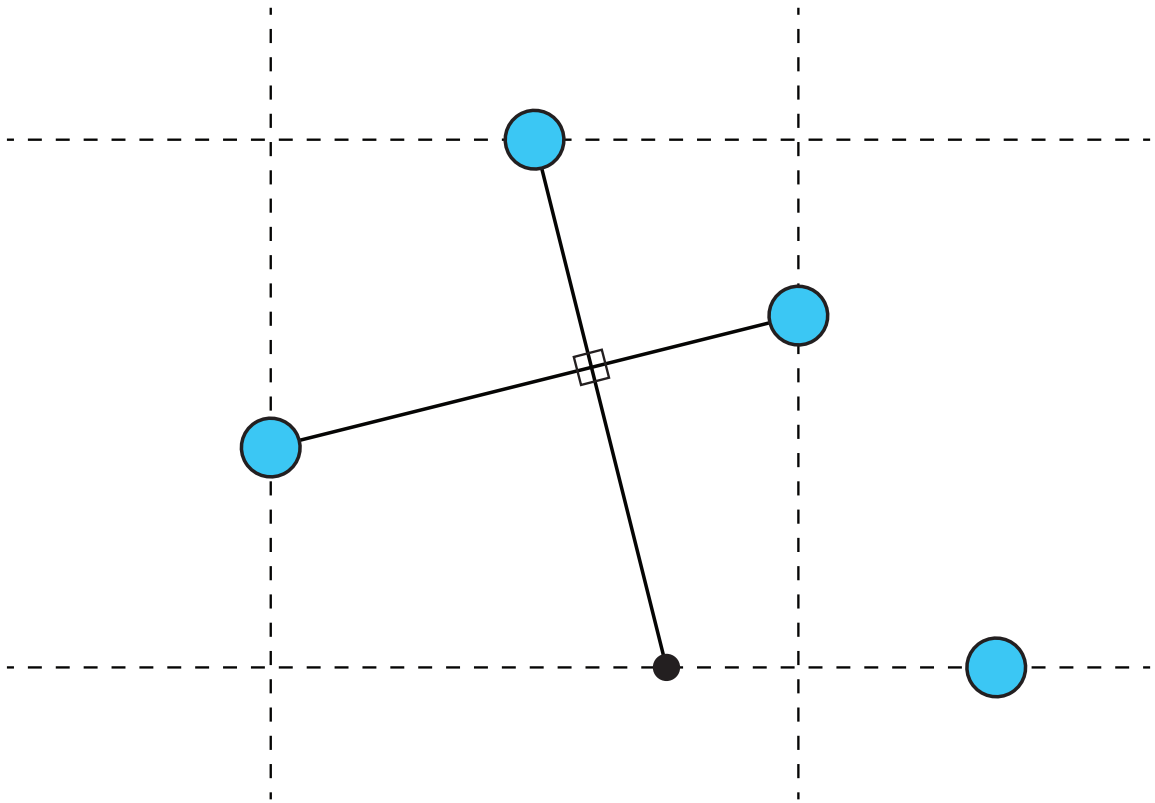
However, there is a geometric construction that identifies one

Identifying the supporting square



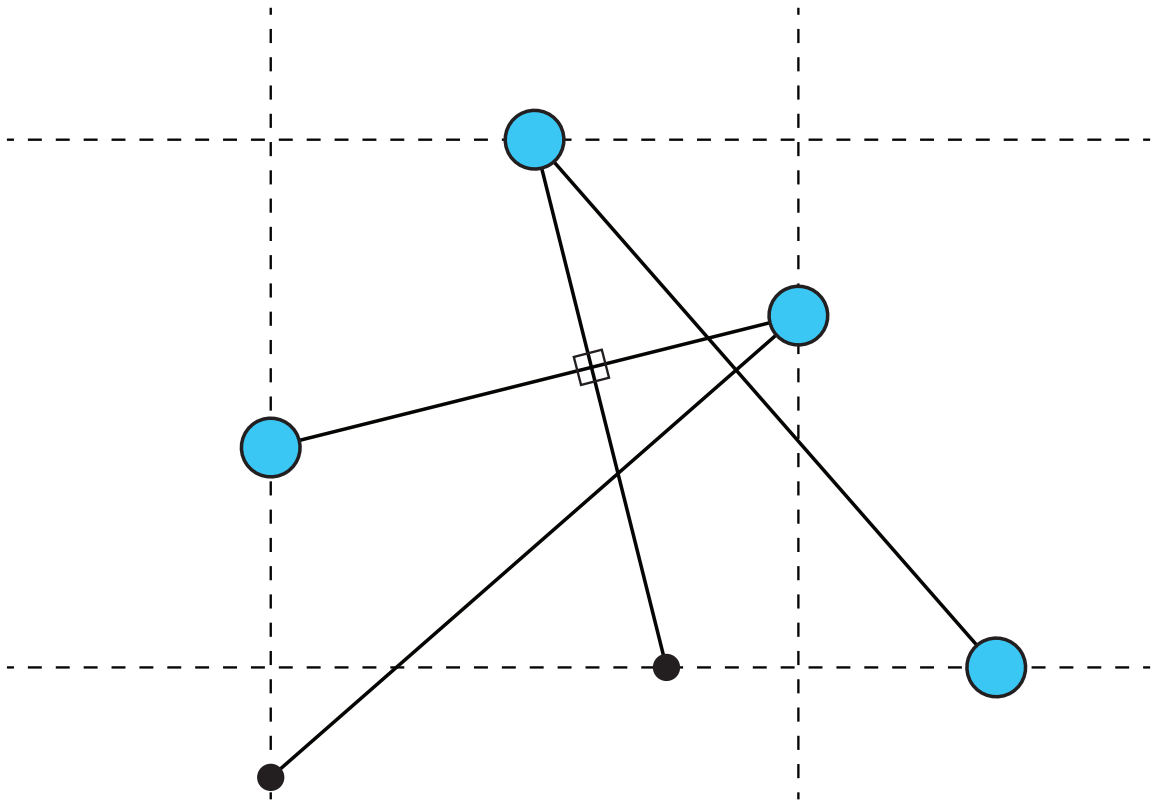
However, there is a geometric construction that identifies one

Identifying the supporting square



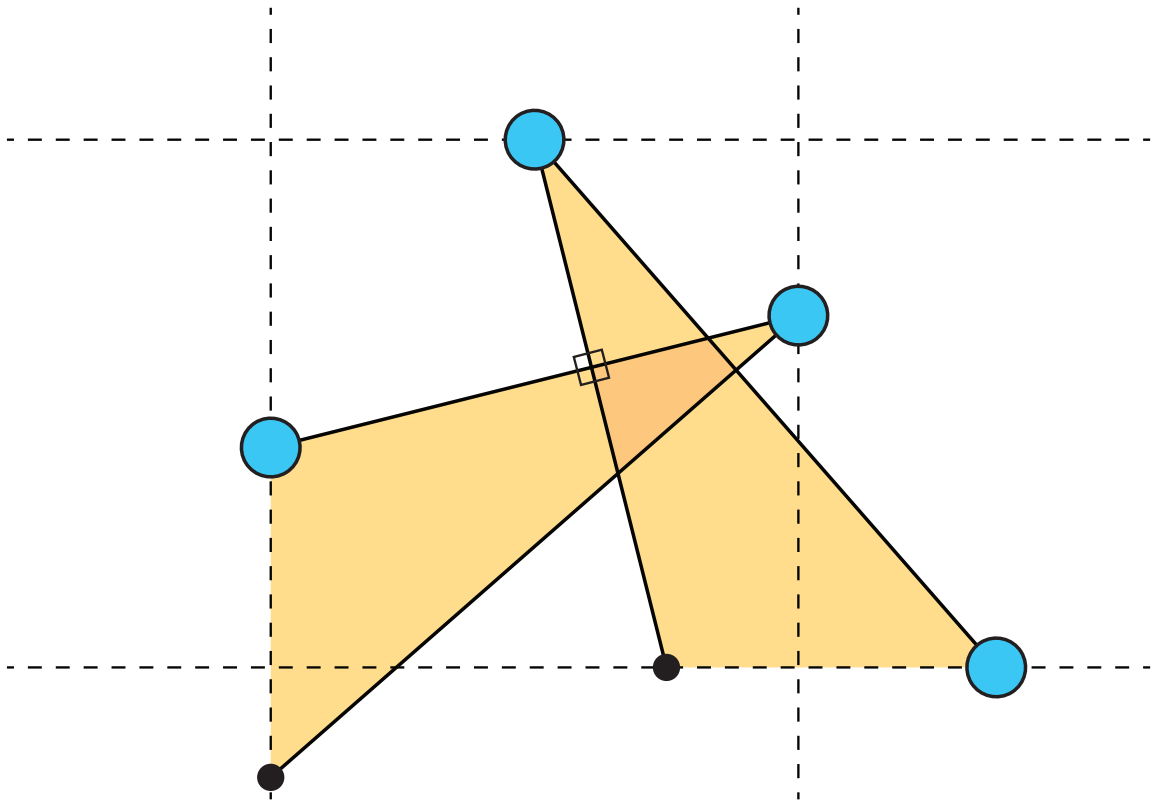
All robots automatically agree on the same supporting square!

Identifying the supporting square



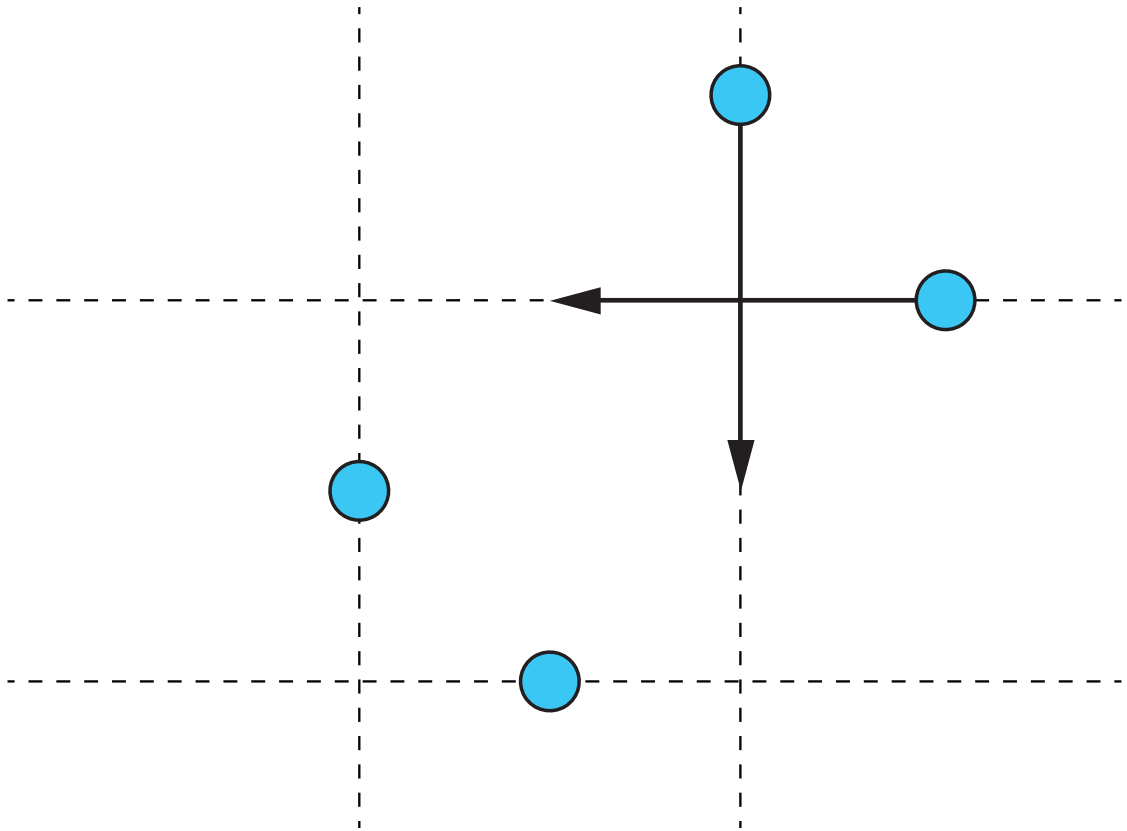
All robots automatically agree on the same supporting square!

Identifying the supporting square



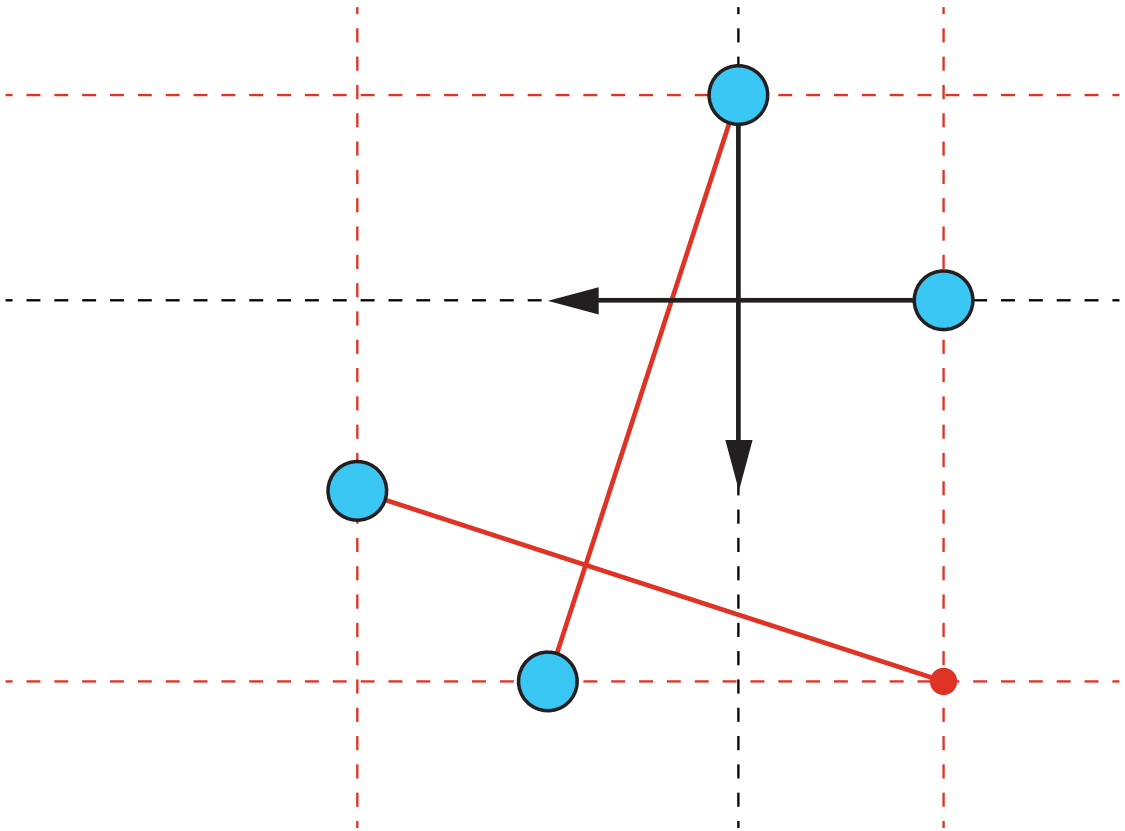
All robots automatically agree on the same supporting square!

Identifying the supporting square



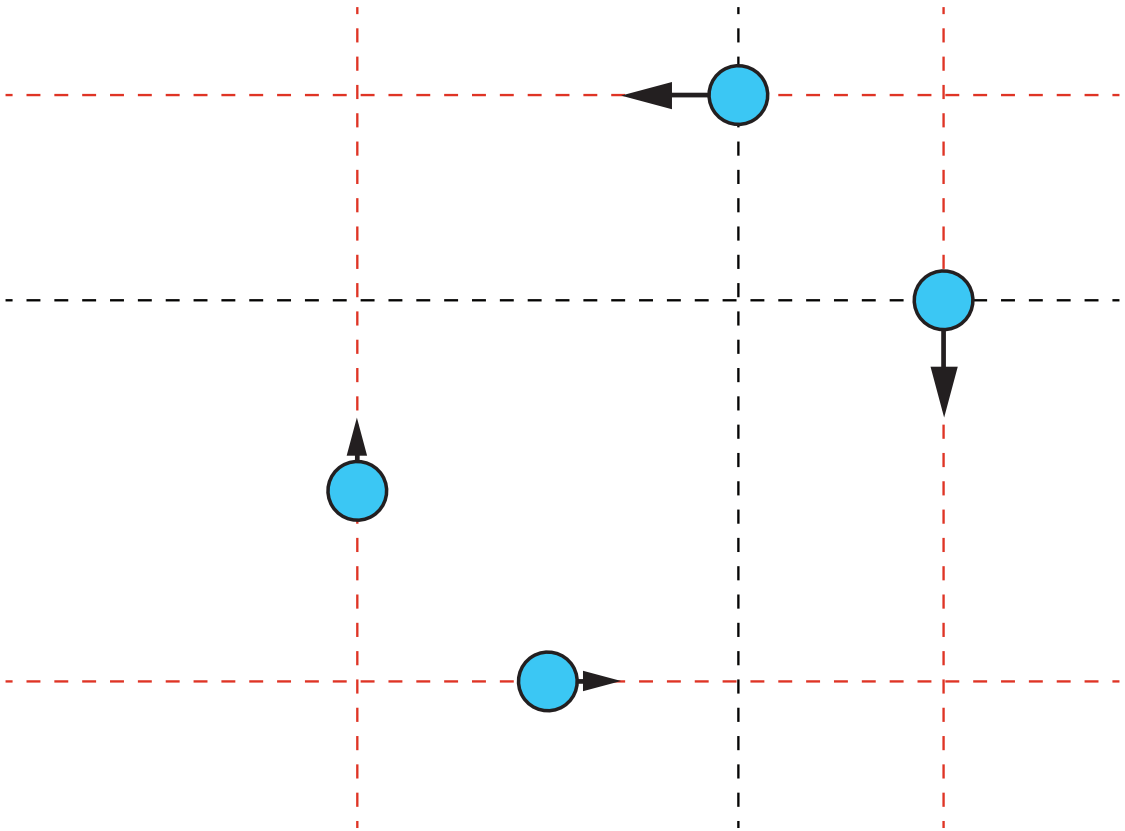
No two robots have intersecting pathways!

Identifying the supporting square



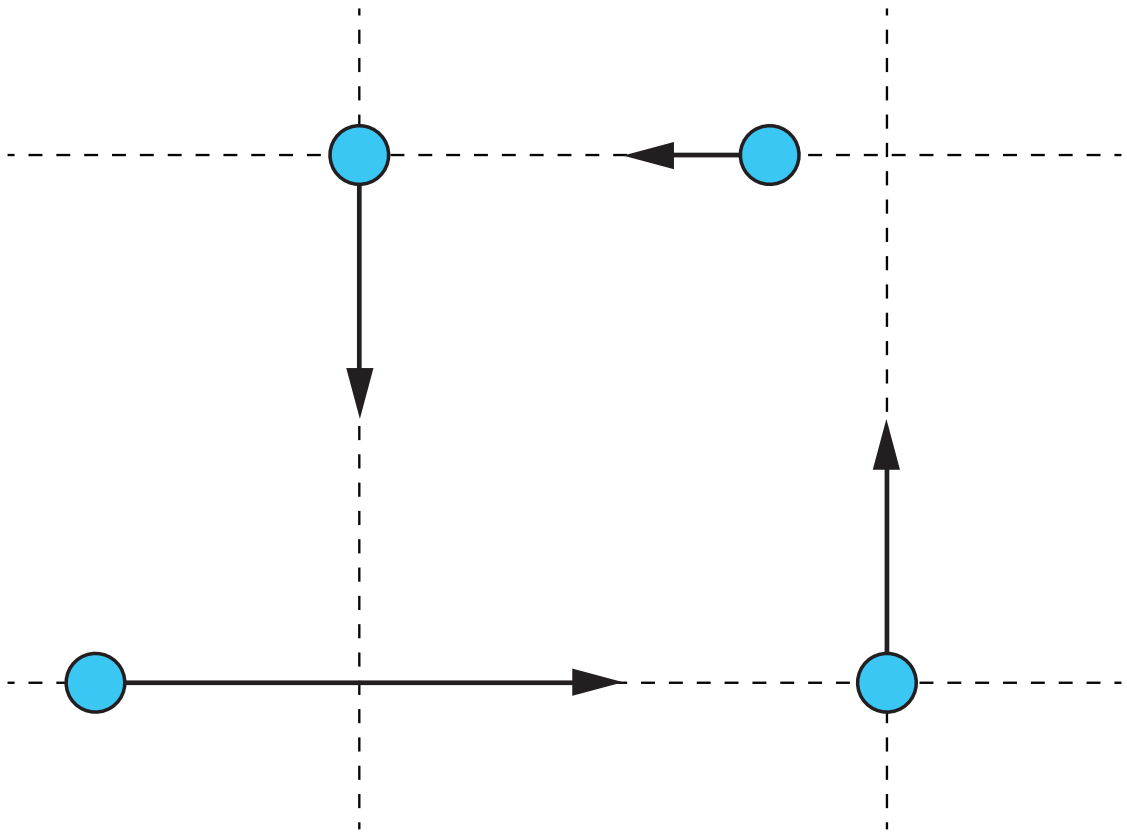
No two robots have intersecting pathways!

Identifying the supporting square



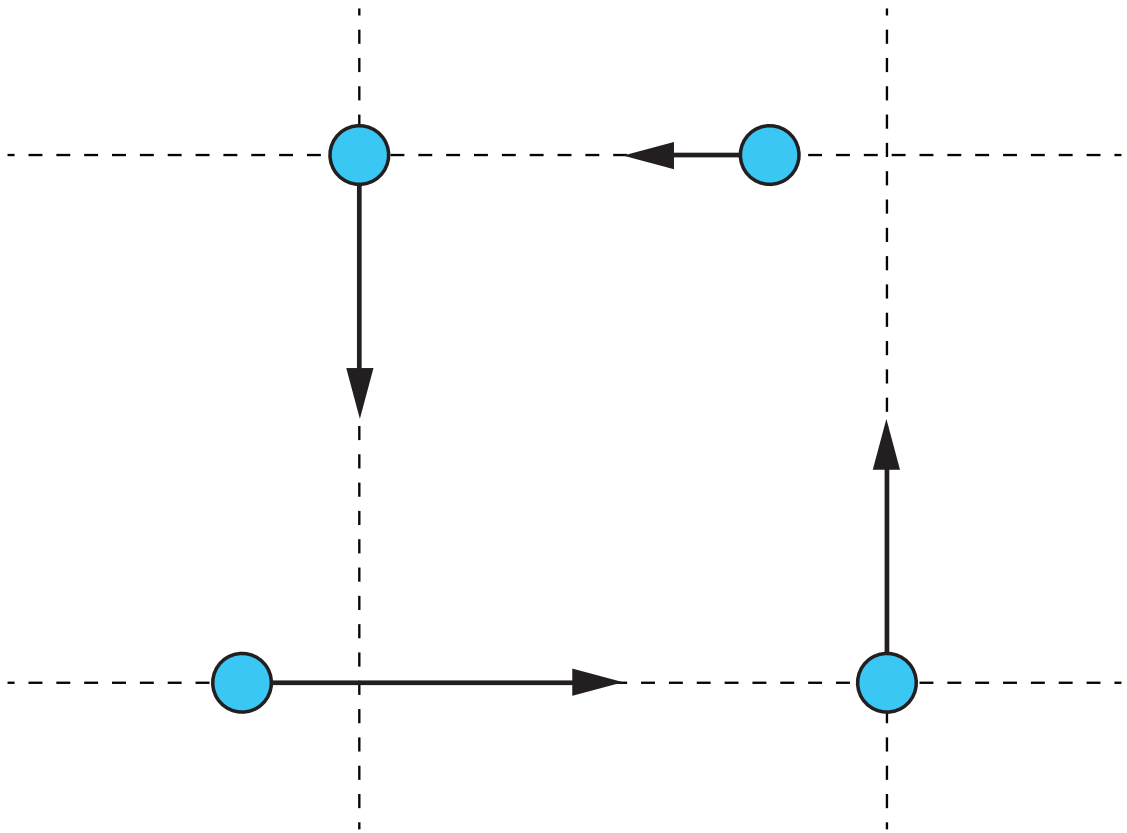
No two robots have intersecting pathways!

Problem: orthogonal diagonals



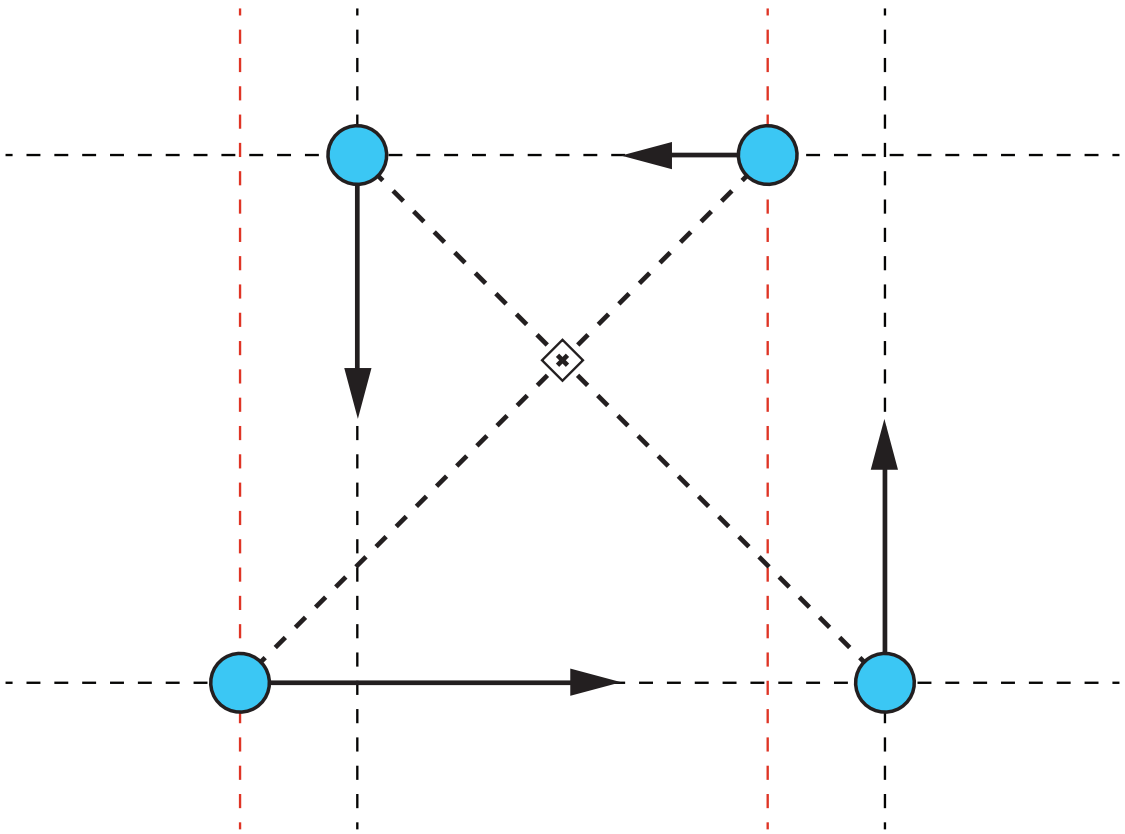
Suppose the two diagonals "accidentally" become orthogonal

Problem: orthogonal diagonals



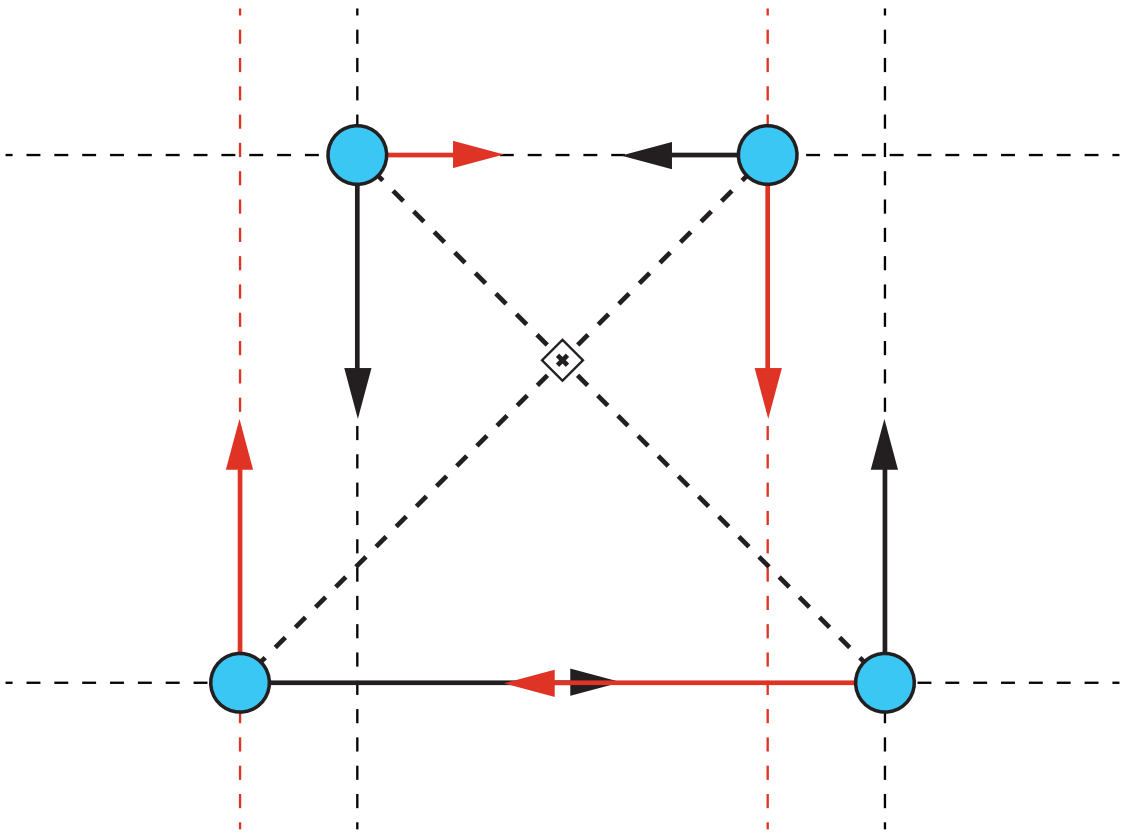
Suppose the two diagonals "accidentally" become orthogonal

Problem: orthogonal diagonals



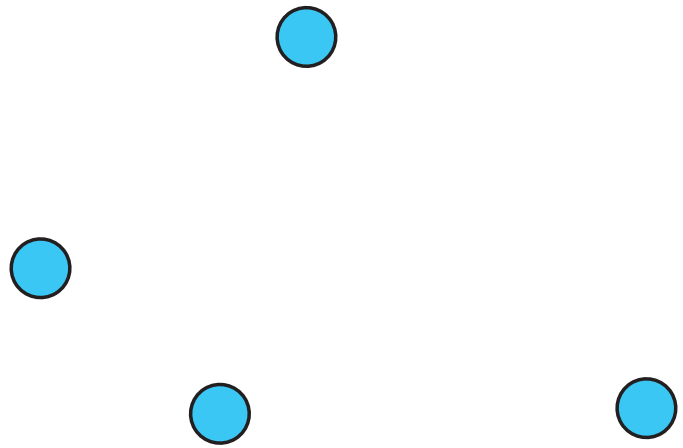
Then our construction does not work

Problem: orthogonal diagonals



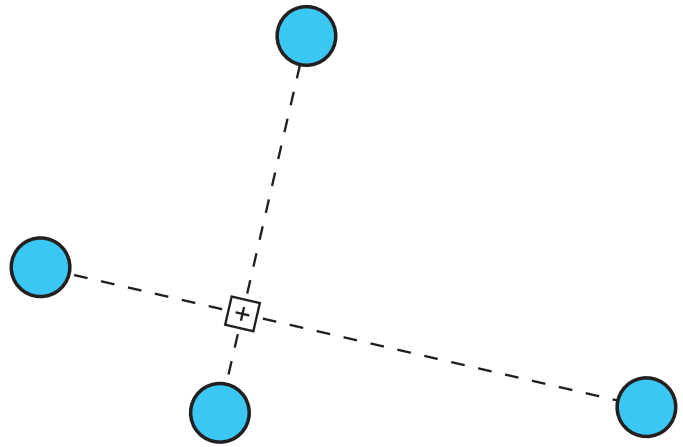
The robots may not agree on a supporting square

Special strategy for orthogonal diagonals



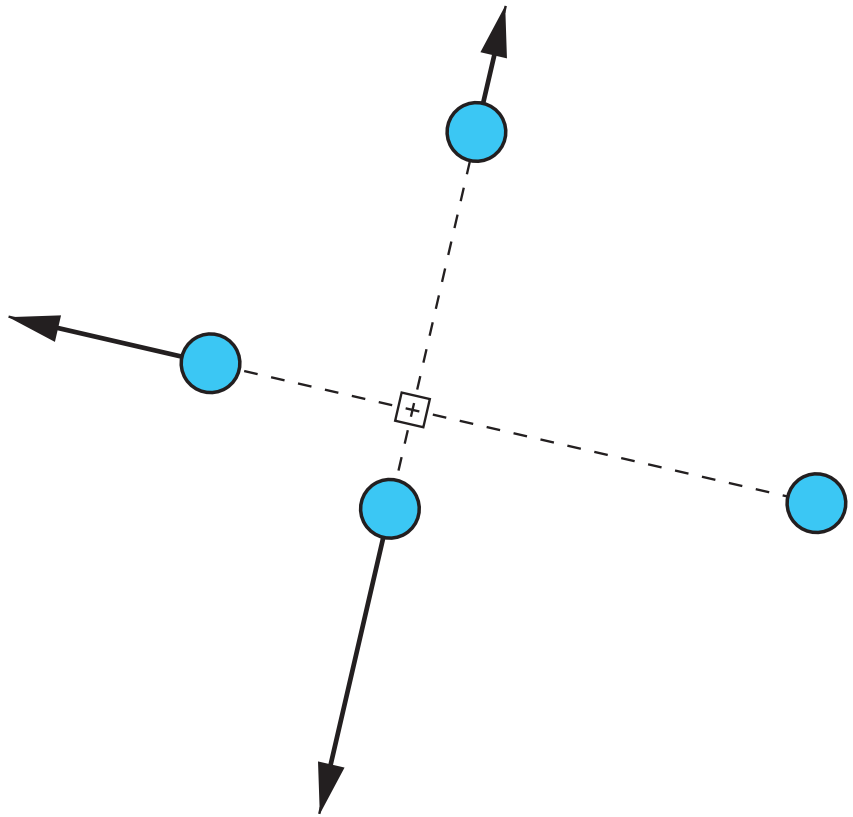
If the diagonals are orthogonal, we use a different approach

Special strategy for orthogonal diagonals



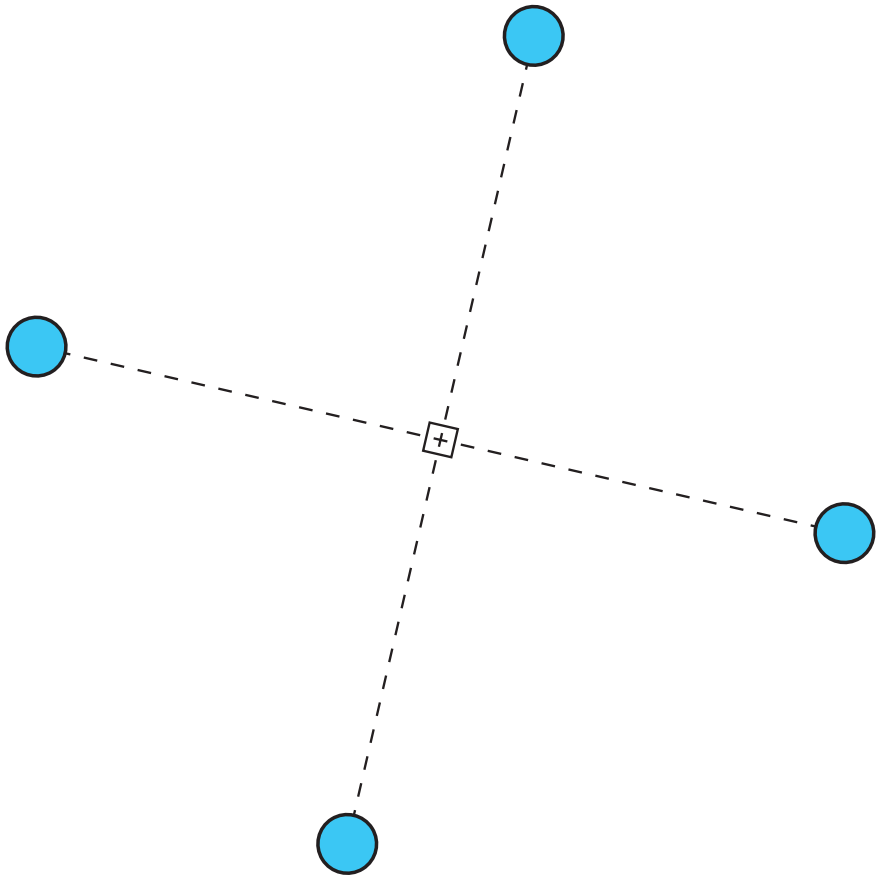
If the diagonals are orthogonal, we use a different approach

Special strategy for orthogonal diagonals



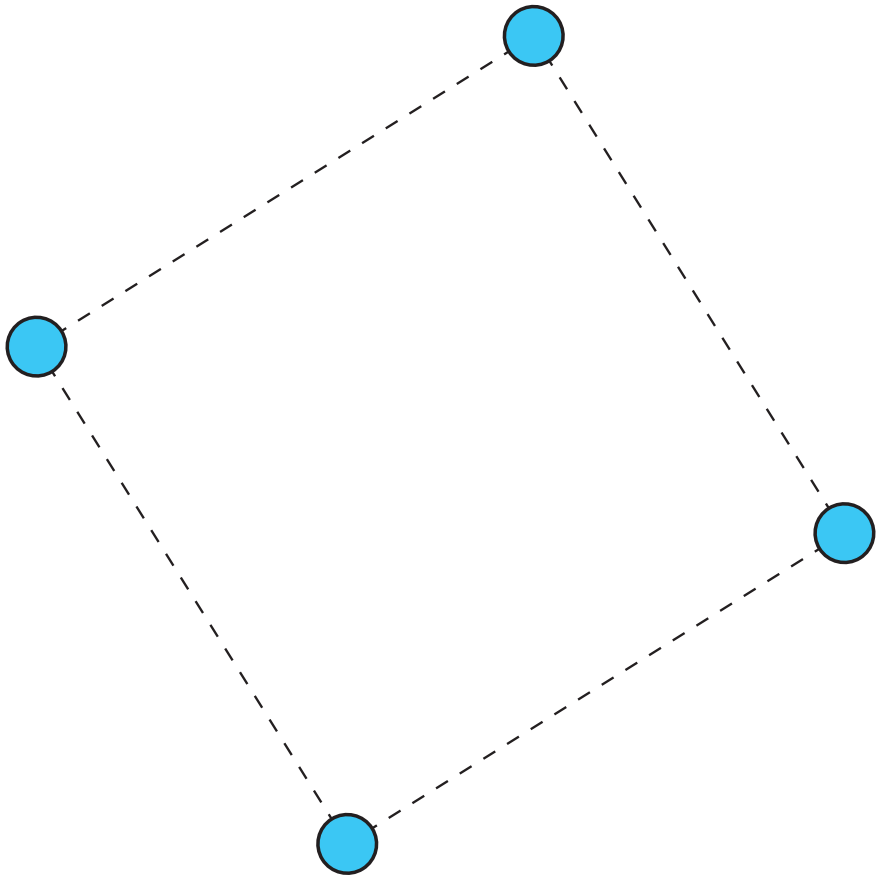
The robots that are closest to the center move away from it

Special strategy for orthogonal diagonals



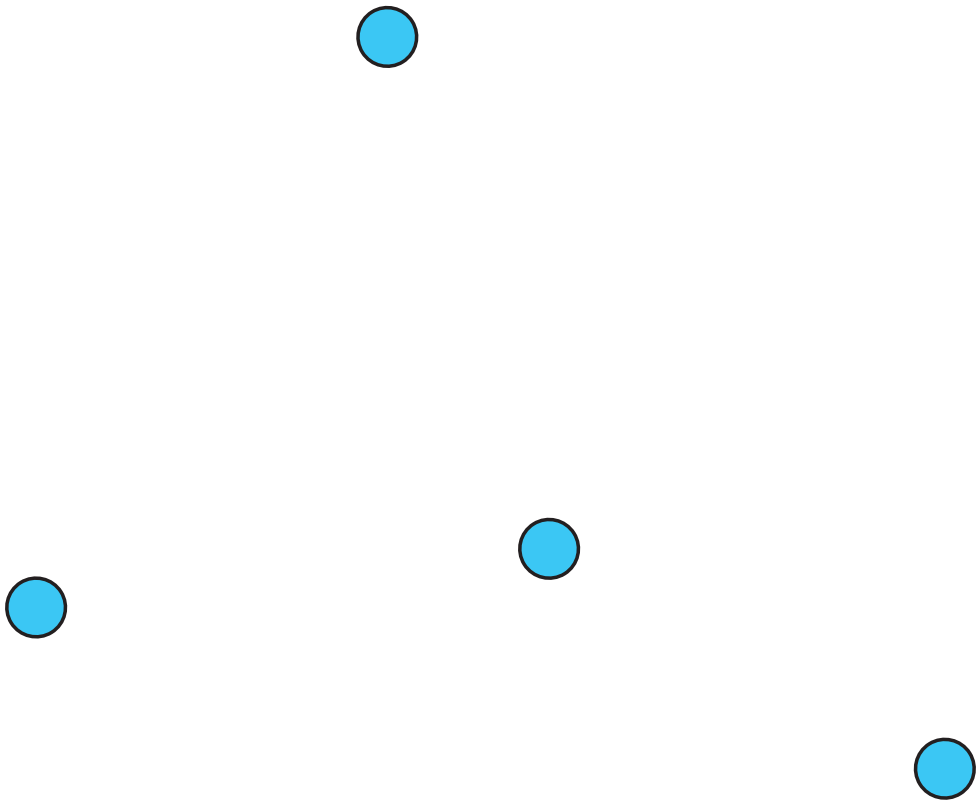
The robots that are closest to the center move away from it

Special strategy for orthogonal diagonals



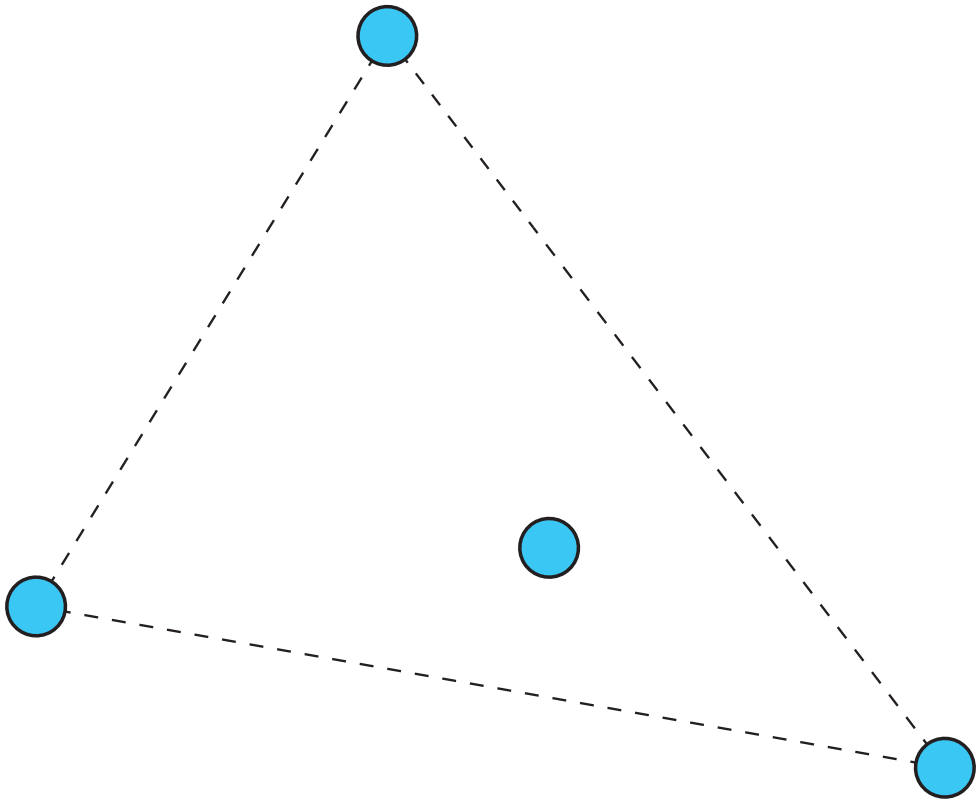
The robots that are closest to the center move away from it

Special strategy for non-convex configurations



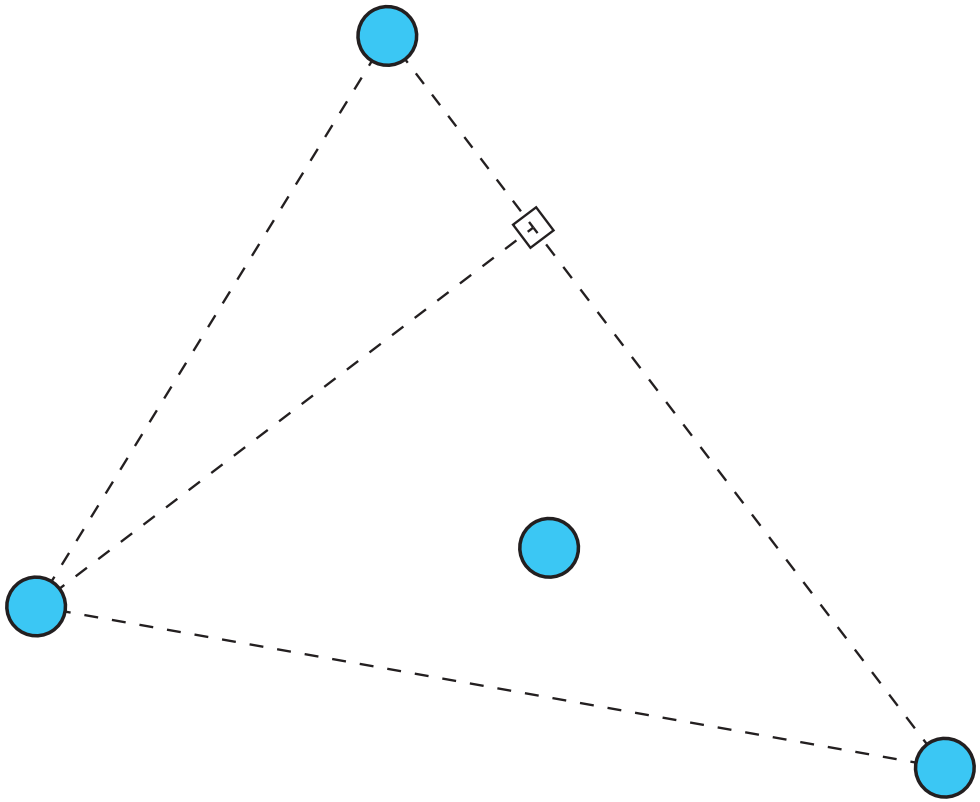
For non-convex configurations, our construction does not work...

Special strategy for non-convex configurations



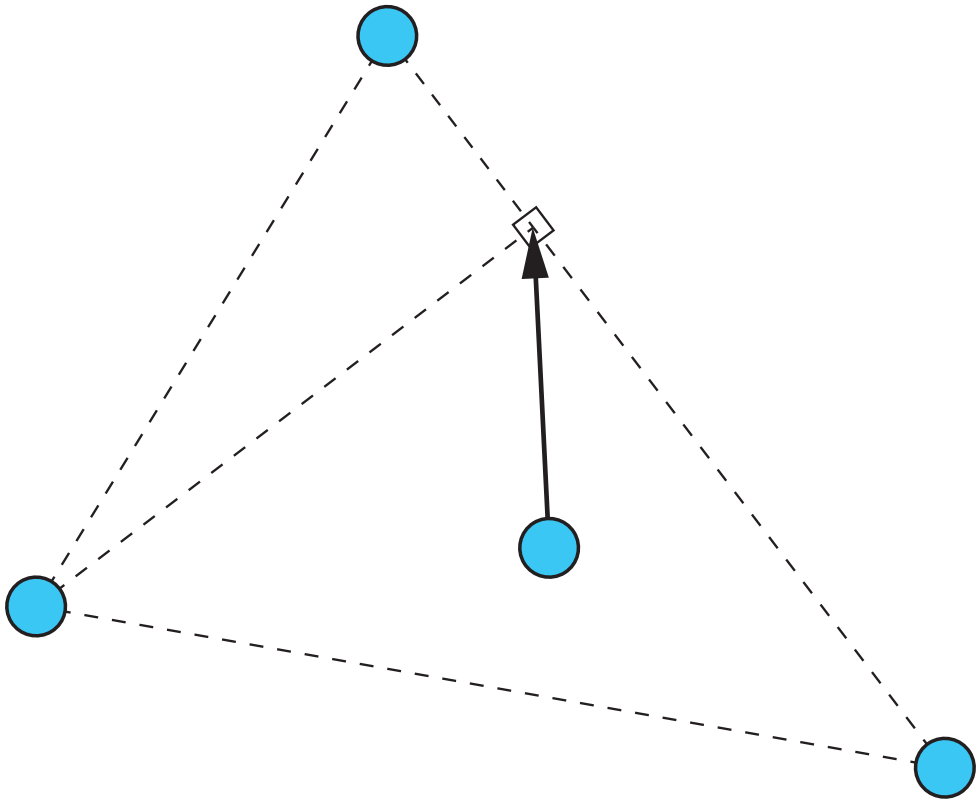
...Because the diagonals are not well defined

Special strategy for non-convex configurations



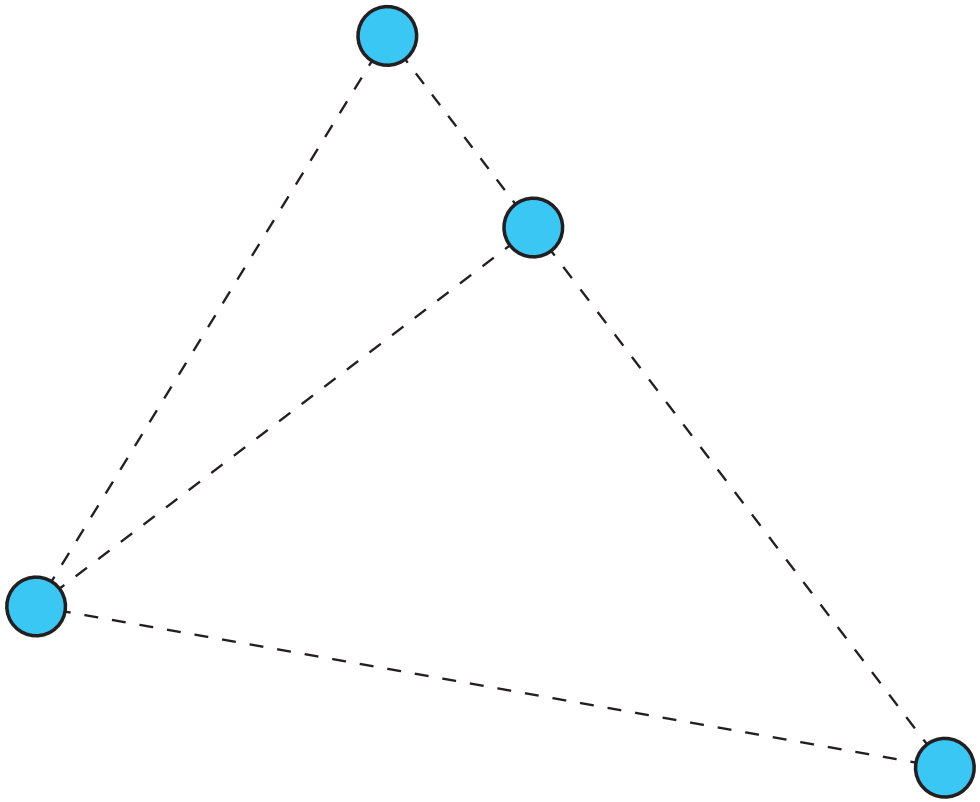
In this case, the internal robot moves...

Special strategy for non-convex configurations



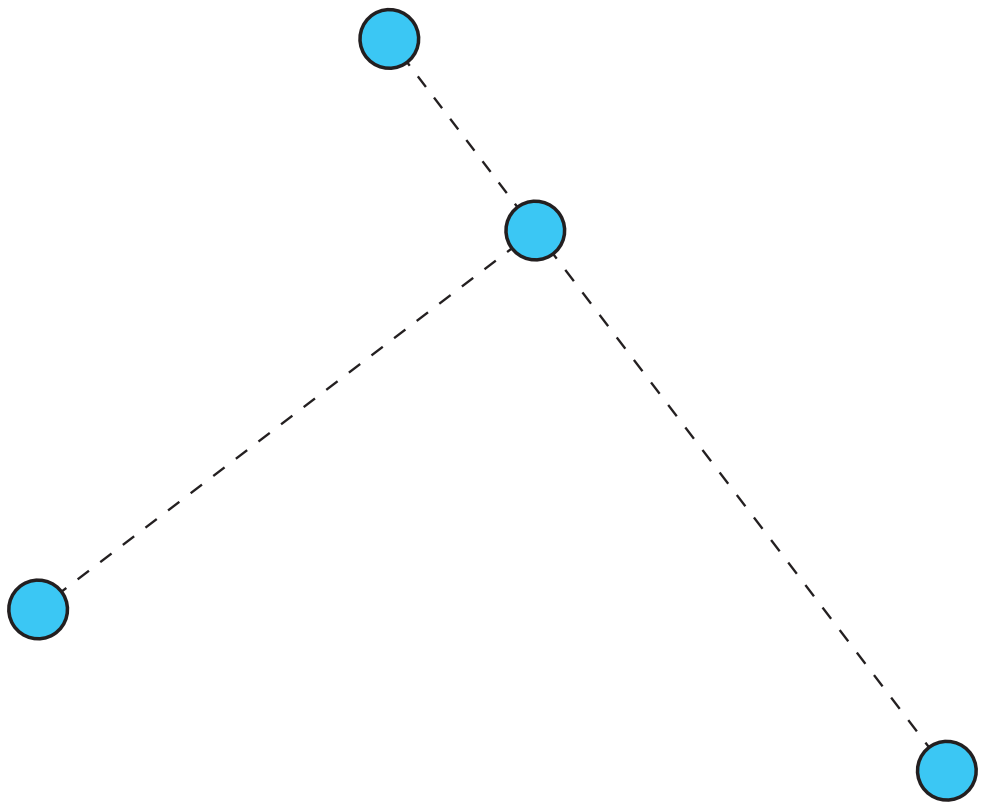
In this case, the internal robot moves...

Special strategy for non-convex configurations



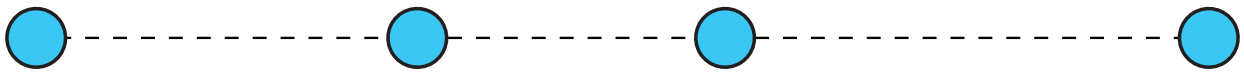
...So to make the diagonals orthogonal...

Special strategy for non-convex configurations



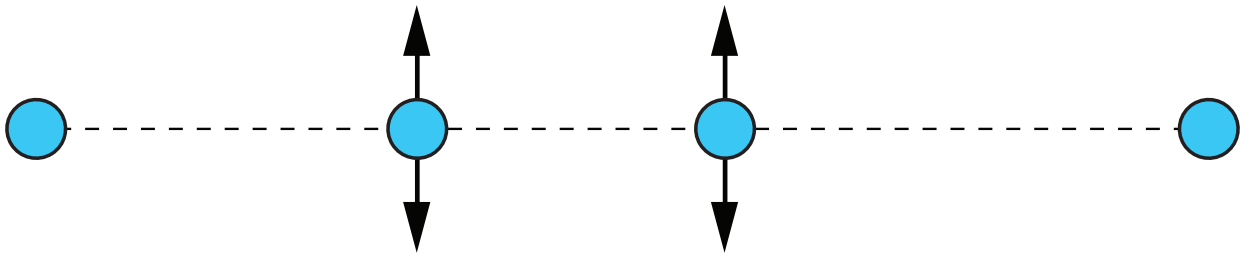
...And reduce the problem to the previous case

Special strategy for collinear configurations



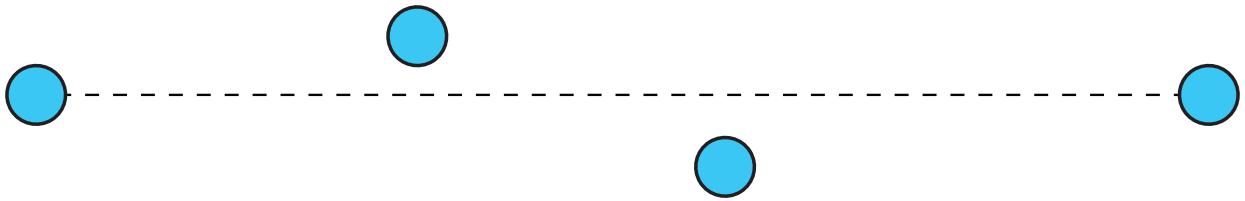
If the robots are collinear, the previous approach does not work

Special strategy for collinear configurations



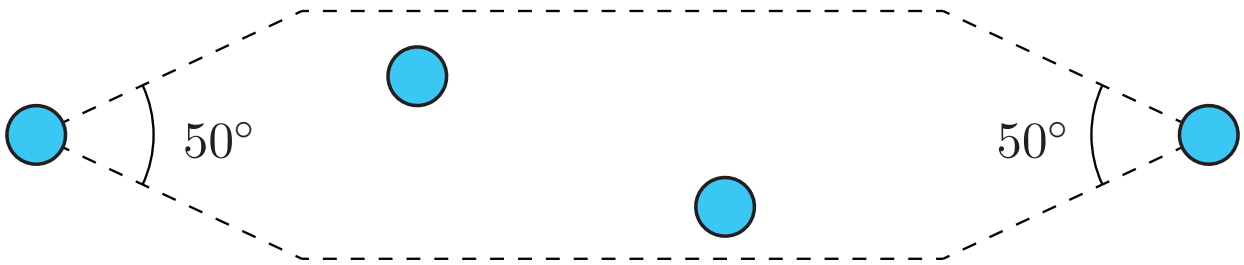
In this case, the internal robots move to either side of the line

Special strategy for collinear configurations



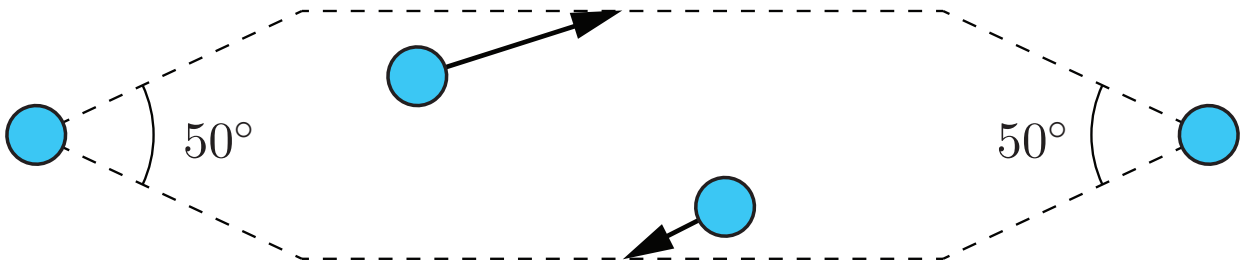
As they asynchronously move, their supporting square may change

Special strategy for collinear configurations



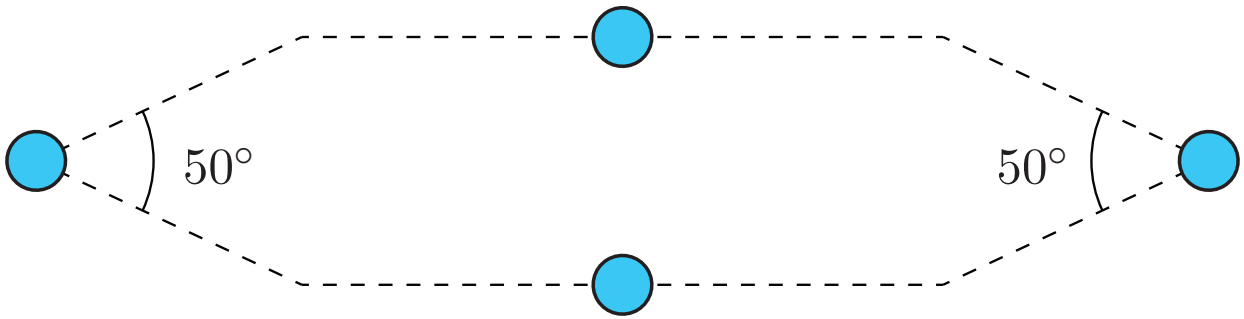
So we must identify a "safe region", e.g., a thin hexagon

Special strategy for collinear configurations



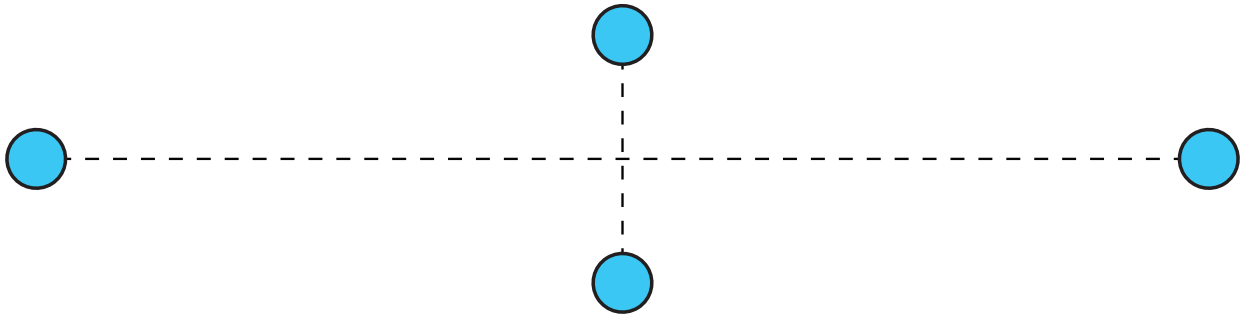
If the robots are in a thin hexagon, they follow a special algorithm

Special strategy for collinear configurations



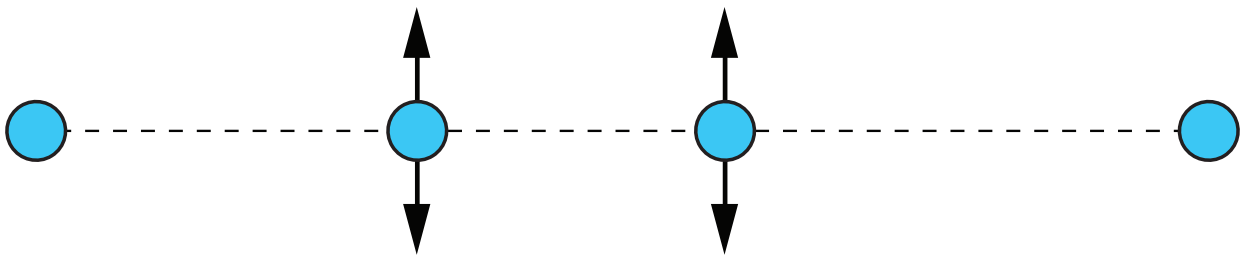
If they end up on opposite sides of the long diagonal...

Special strategy for collinear configurations



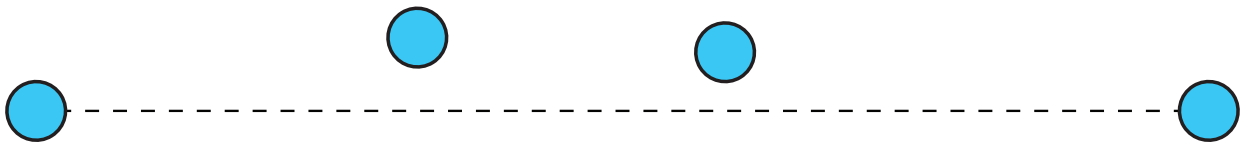
...We make them form a configuration with orthogonal diagonals

Special strategy for collinear configurations



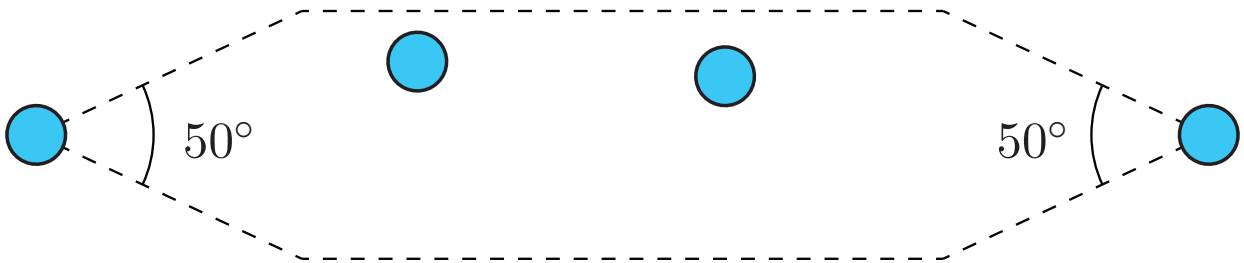
Otherwise, they move on two vertices and wait for each other

Special strategy for collinear configurations



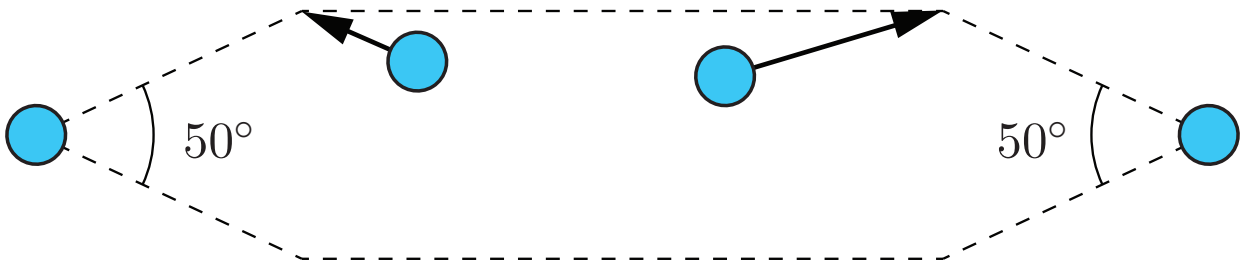
Otherwise, they move on two vertices and wait for each other

Special strategy for collinear configurations



Otherwise, they move on two vertices and wait for each other

Special strategy for collinear configurations

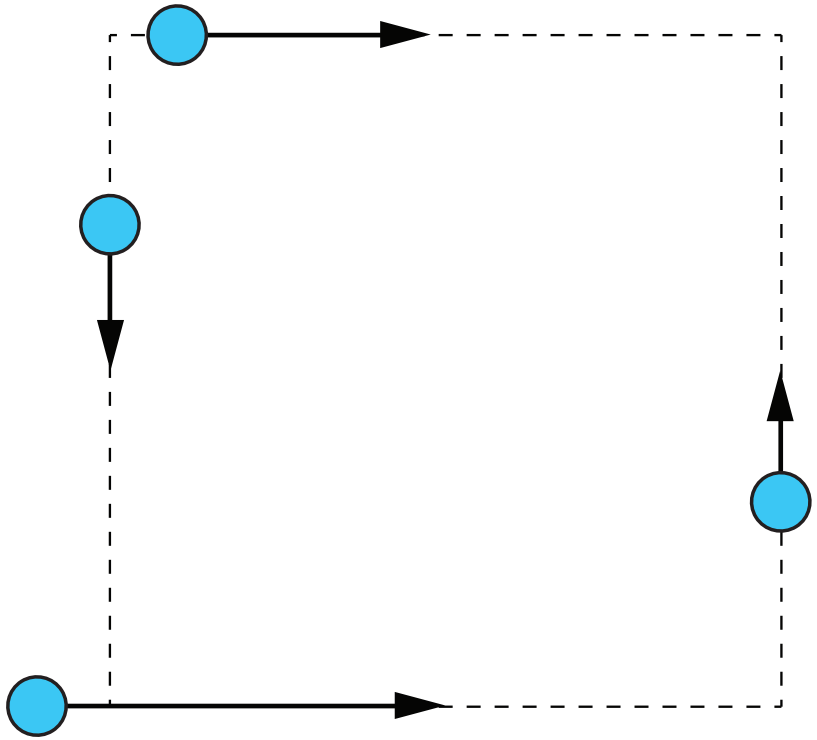


Otherwise, they move on two vertices and wait for each other

Special strategy for collinear configurations

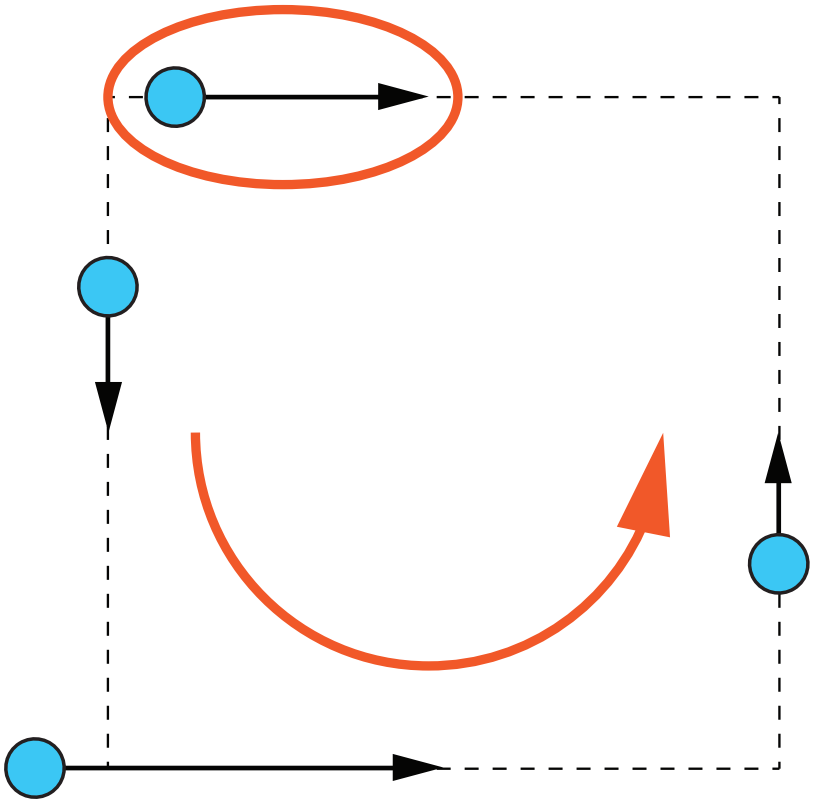
Now that they are not moving, they agree on a supporting square

General algorithm: one discordant robot



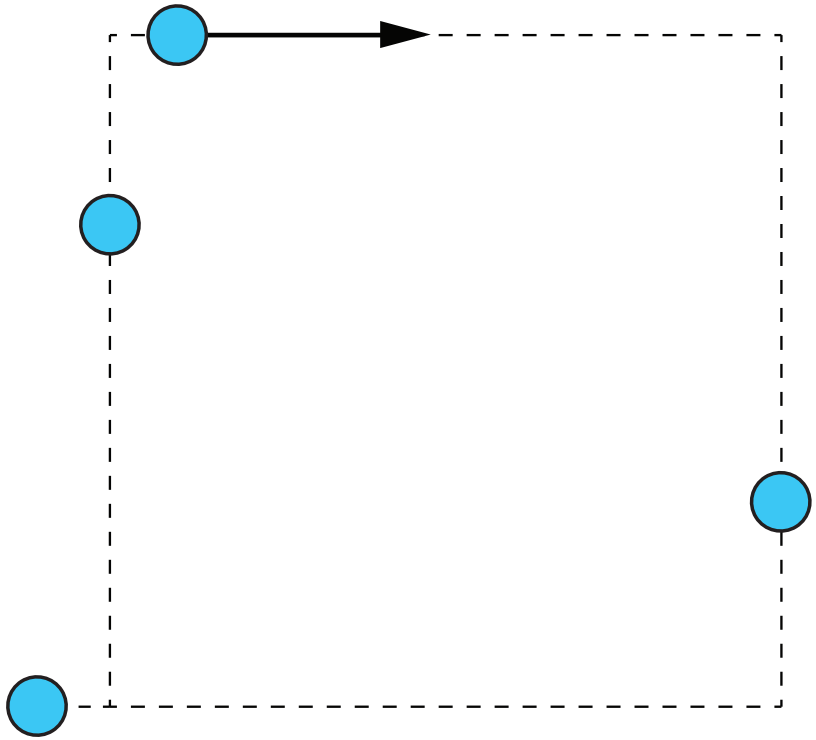
Suppose one robot is “discordant” with all the others

General algorithm: one discordant robot



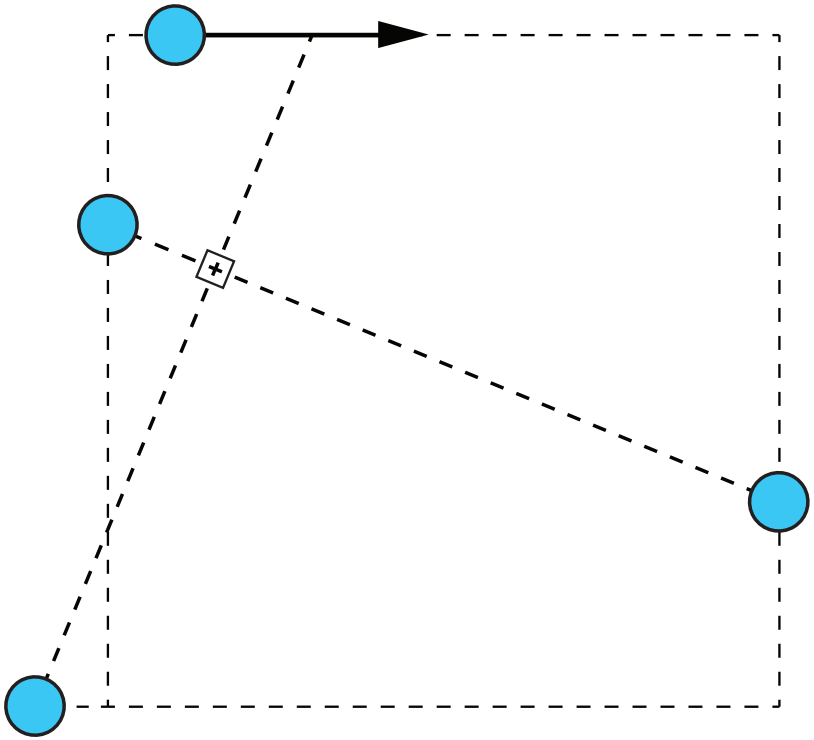
Suppose one robot is “discordant” with all the others

General algorithm: one discordant robot



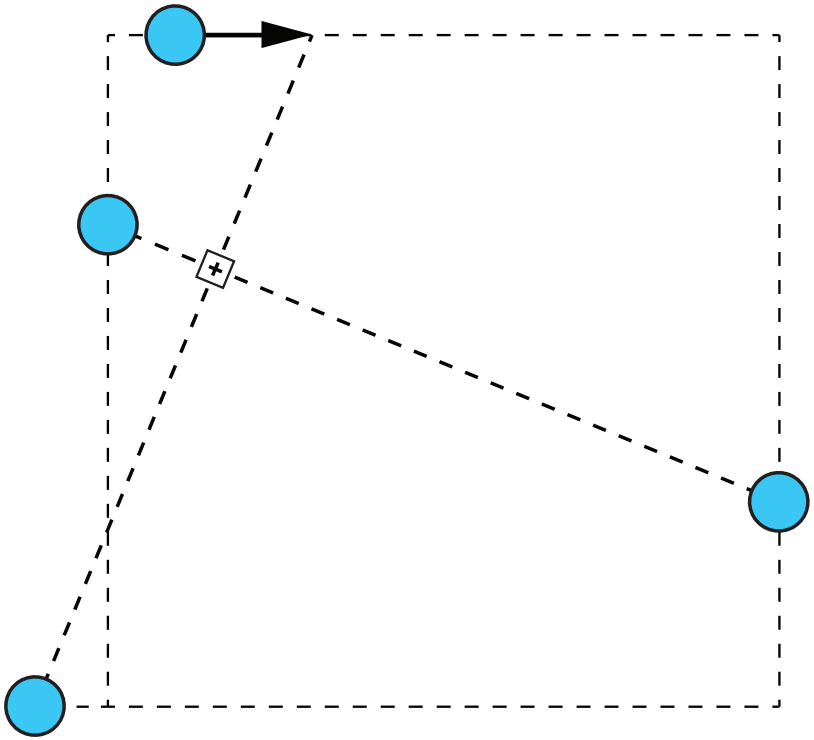
We let only the discordant robot move toward its final destination

General algorithm: one discordant robot



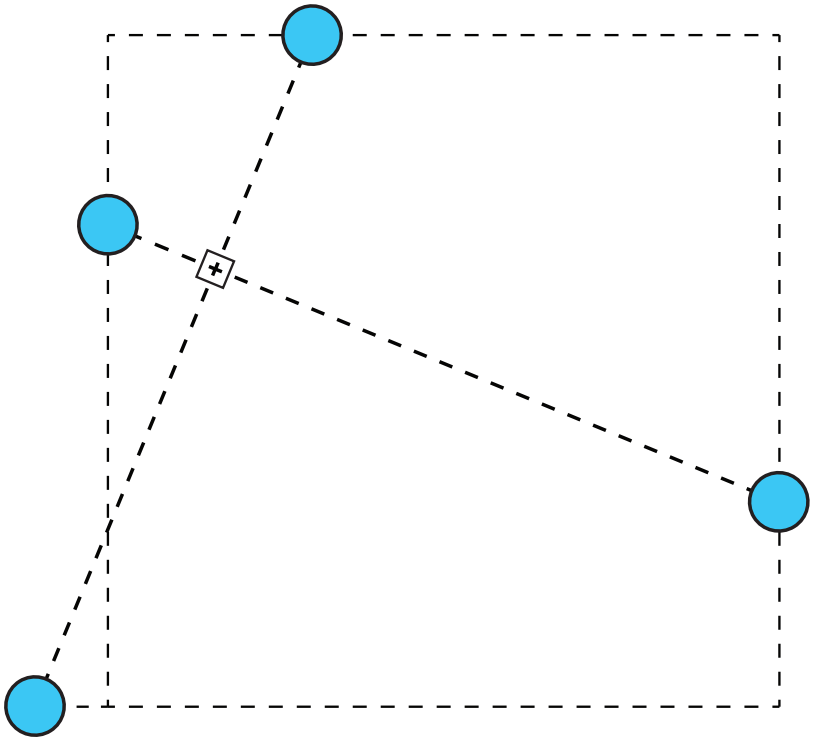
As it moves, it may cause the diagonals to become orthogonal!

General algorithm: one discordant robot



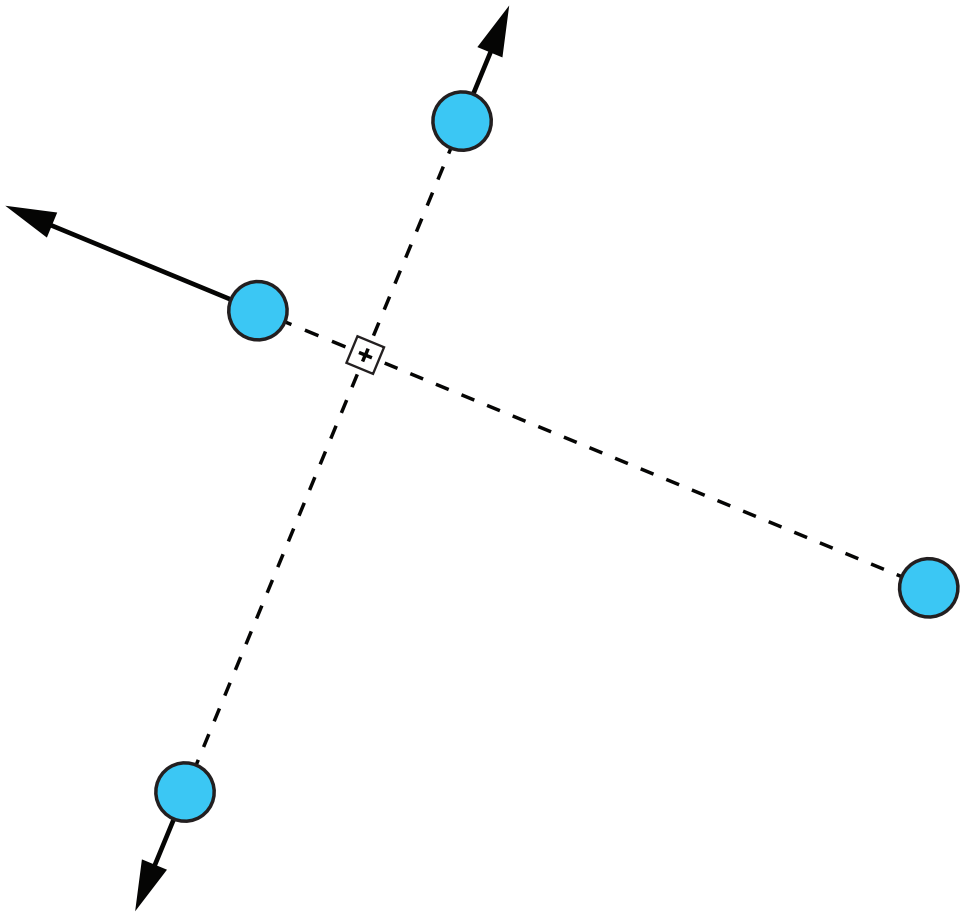
In this case, it has to stop at the point of orthogonality...

General algorithm: one discordant robot



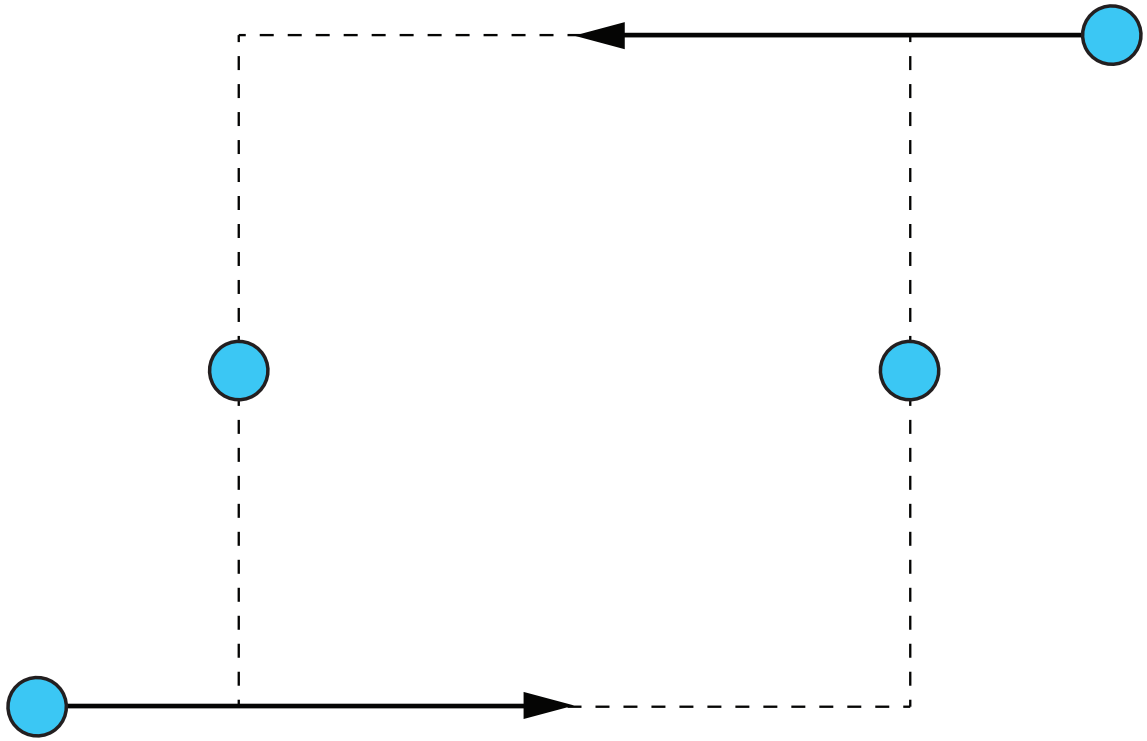
In this case, it has to stop at the point of orthogonality...

General algorithm: one discordant robot



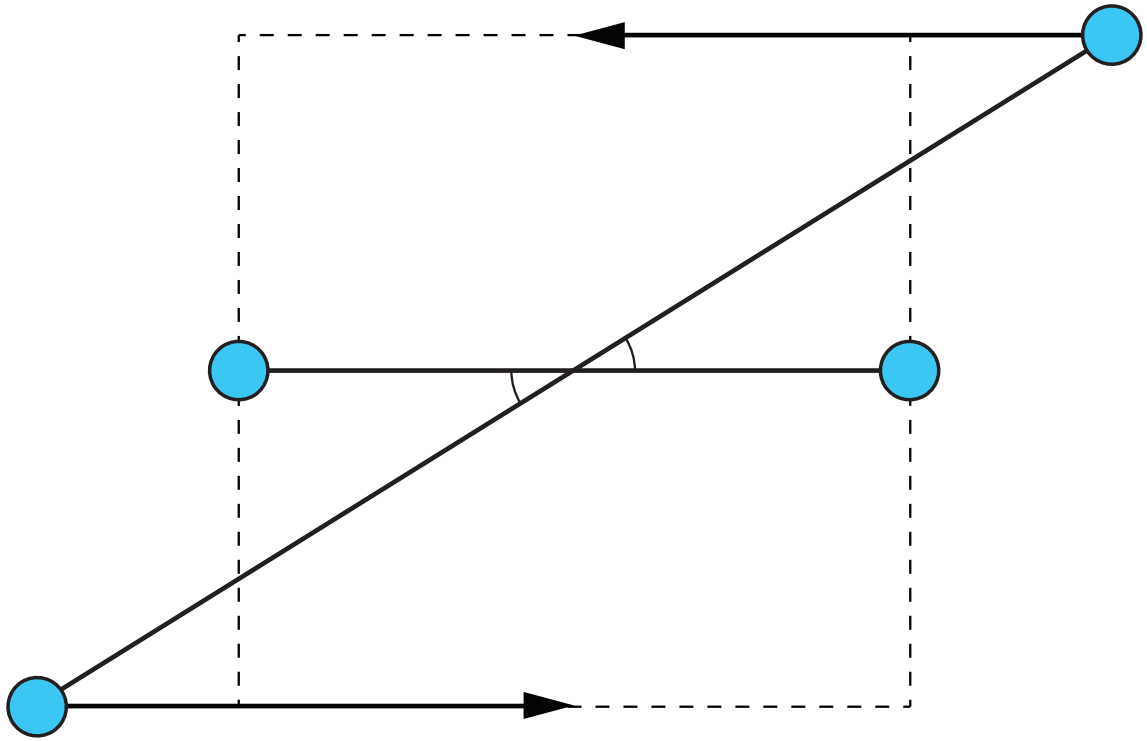
...So all robots will behave coherently, despite asynchronicity

General algorithm: two opposite concordant, two finished



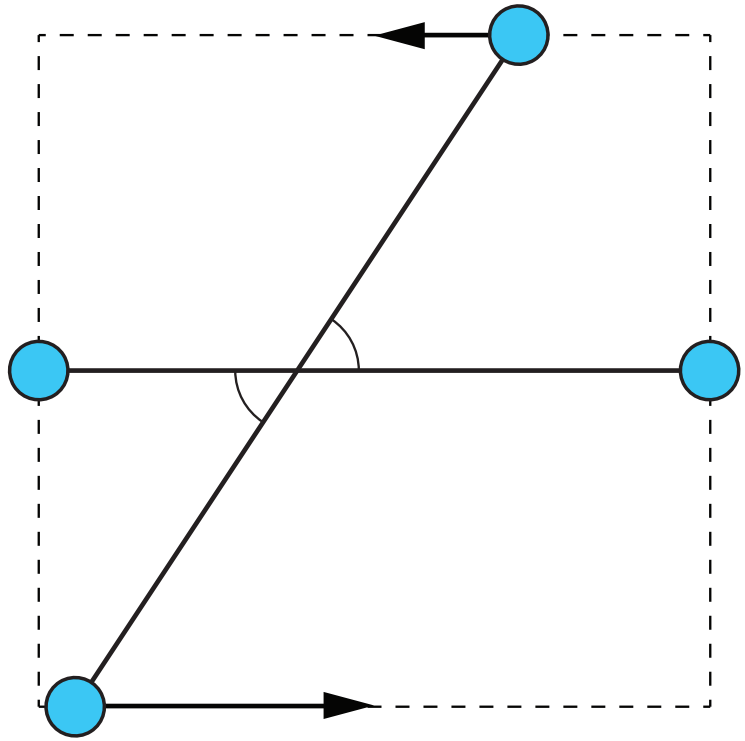
We let the two opposite robots move

General algorithm: two opposite concordant, two finished



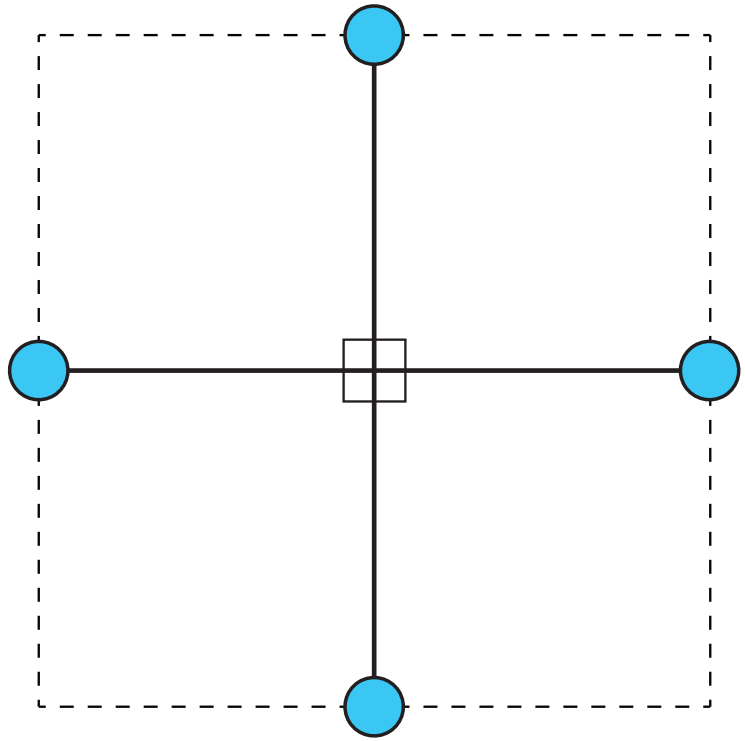
The diagonals can never become orthogonal by accident

General algorithm: two opposite concordant, two finished



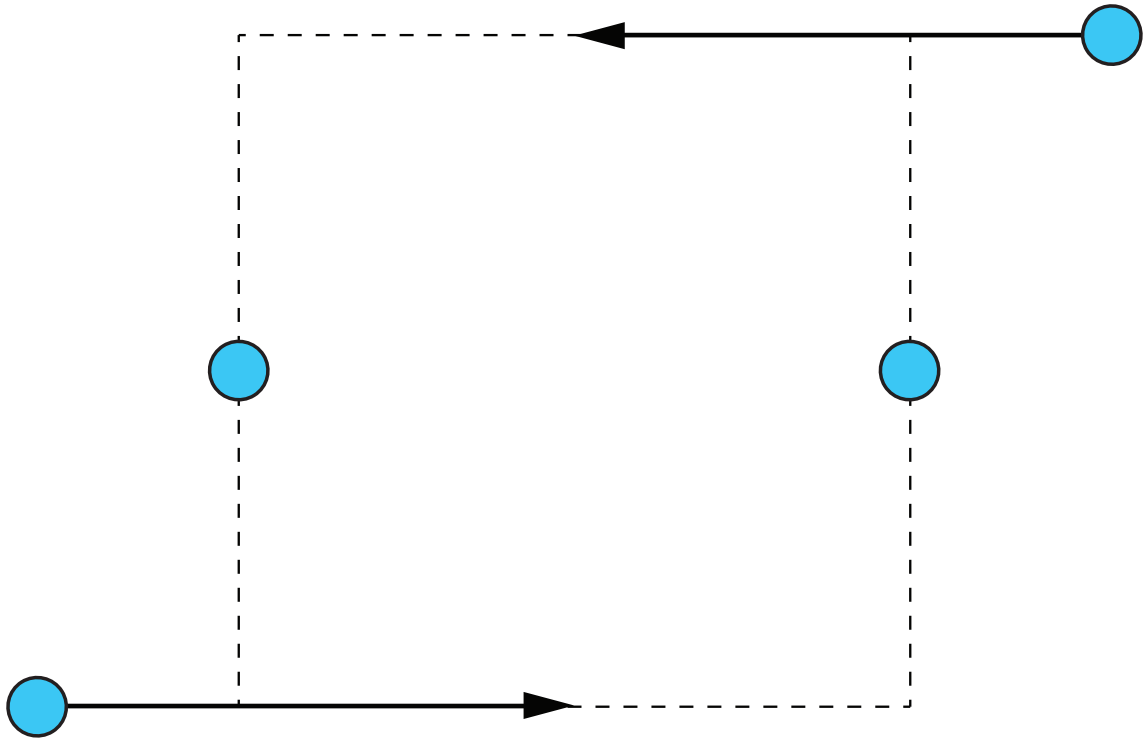
The diagonals can never become orthogonal by accident

General algorithm: two opposite concordant, two finished



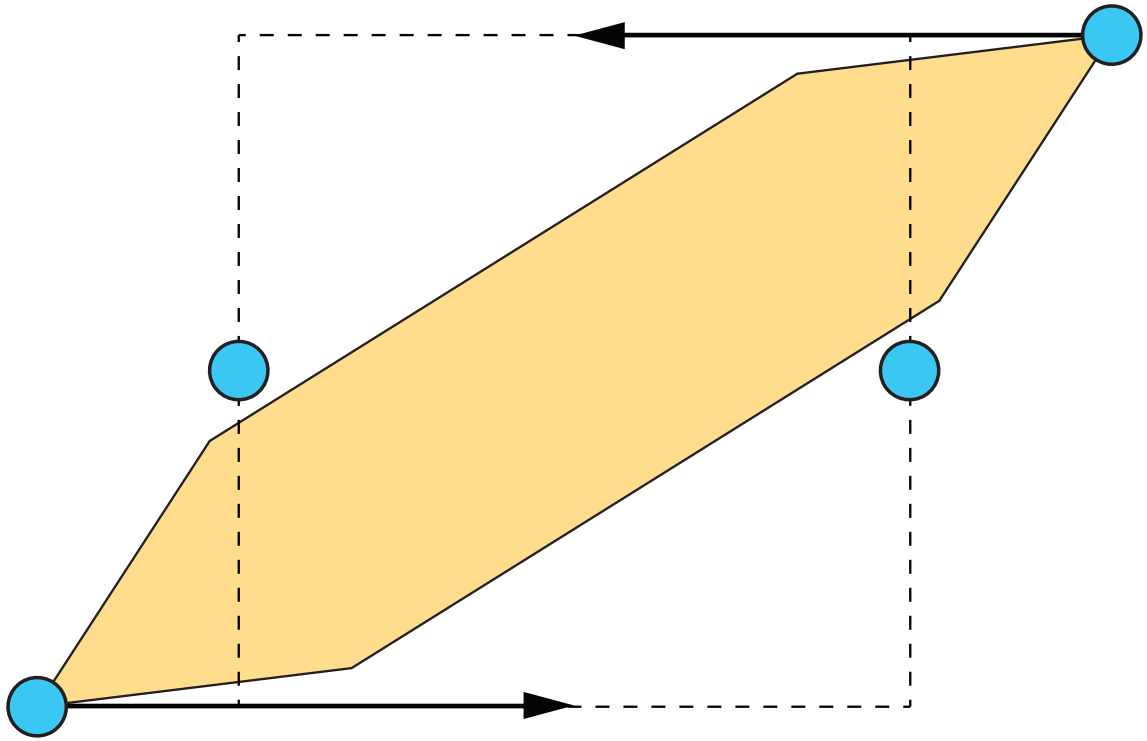
The diagonals can never become orthogonal by accident

General algorithm: two opposite concordant, two finished



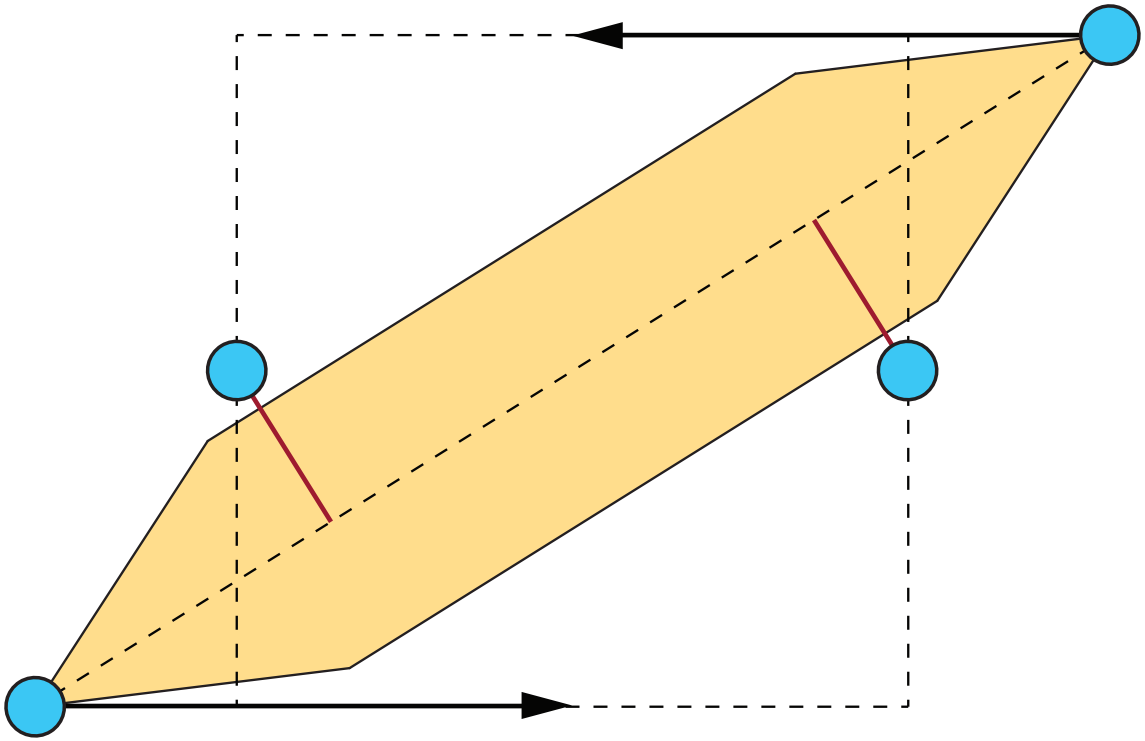
No thin hexagon can be formed by accident...

General algorithm: two opposite concordant, two finished



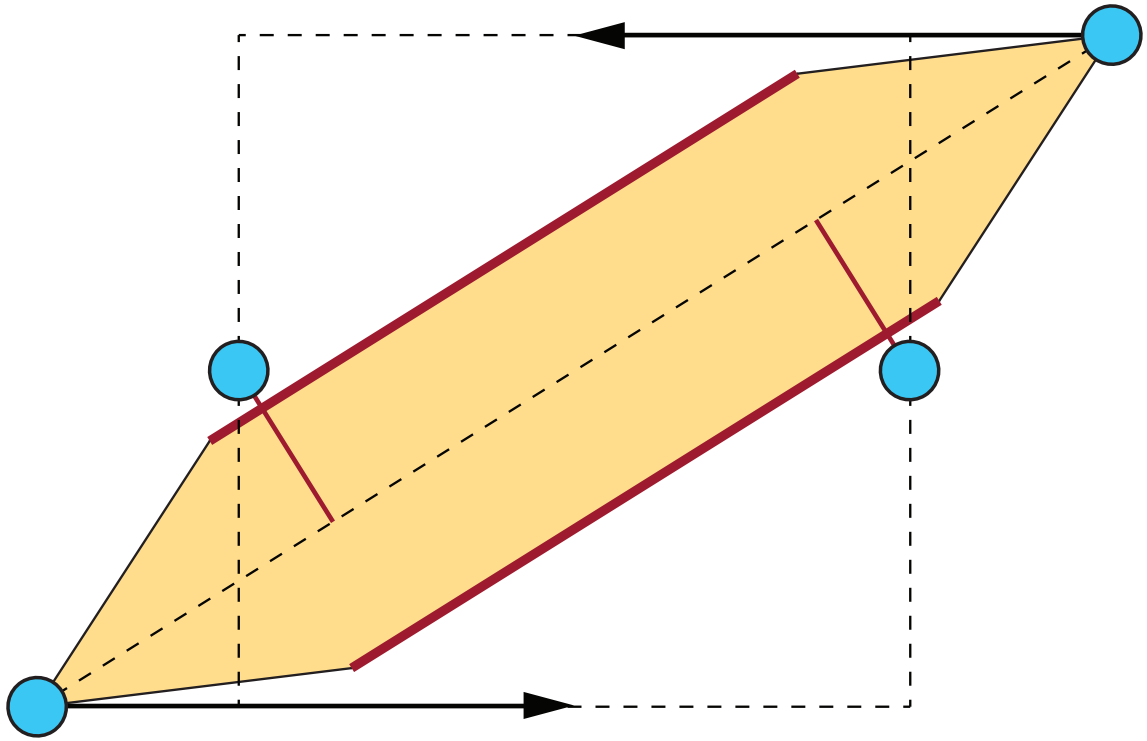
No thin hexagon can be formed by accident...

General algorithm: two opposite concordant, two finished



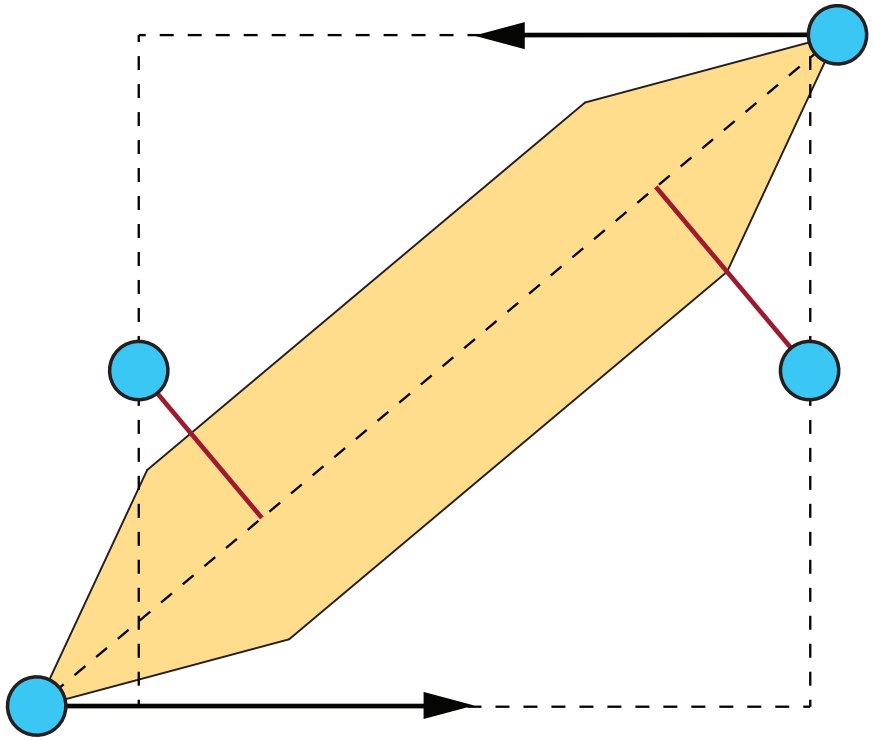
...Because the sum of distances from the long diagonal is too large

General algorithm: two opposite concordant, two finished



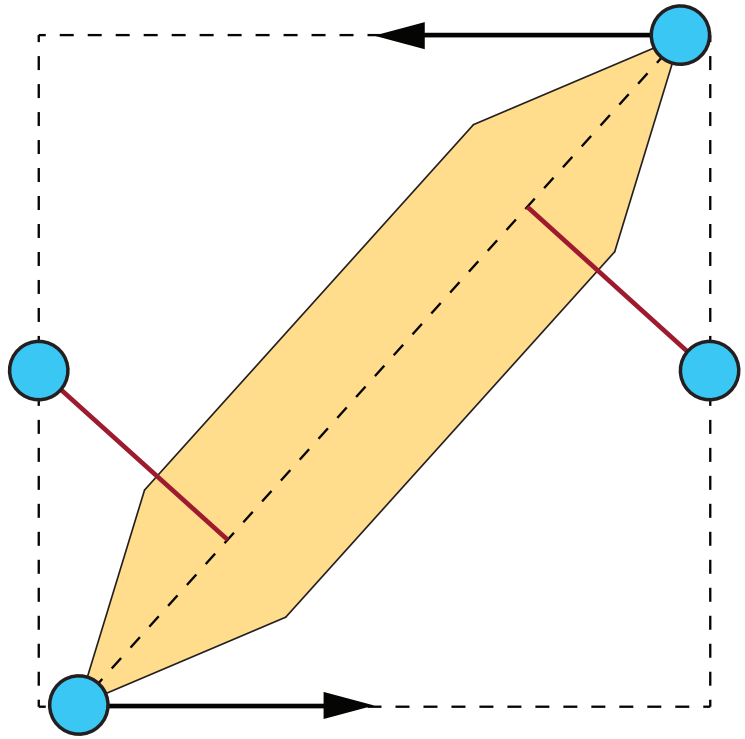
...Because the sum of distances from the long diagonal is too large

General algorithm: two opposite concordant, two finished



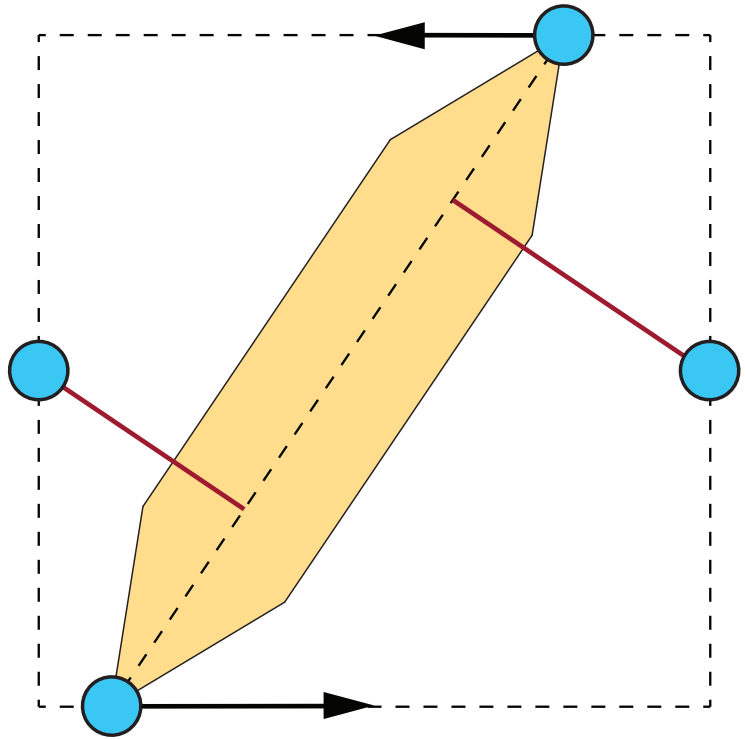
...Because the sum of distances from the long diagonal is too large

General algorithm: two opposite concordant, two finished



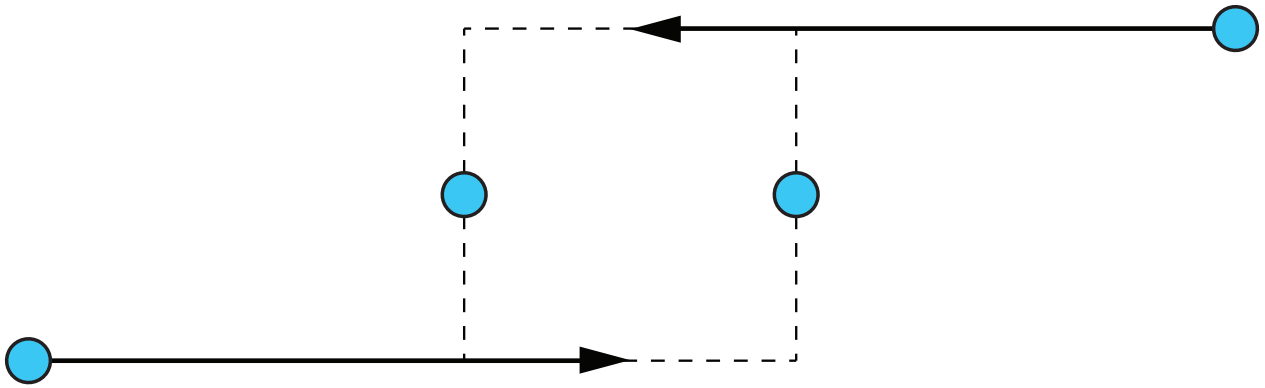
...Because the sum of distances from the long diagonal is too large

General algorithm: two opposite concordant, two finished



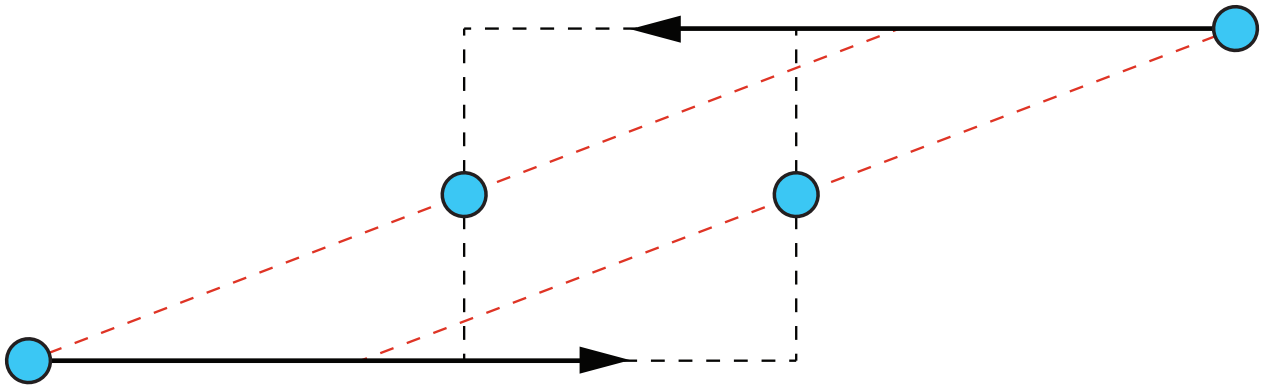
...Because the sum of distances from the long diagonal is too large

General algorithm: two opposite concordant, two finished



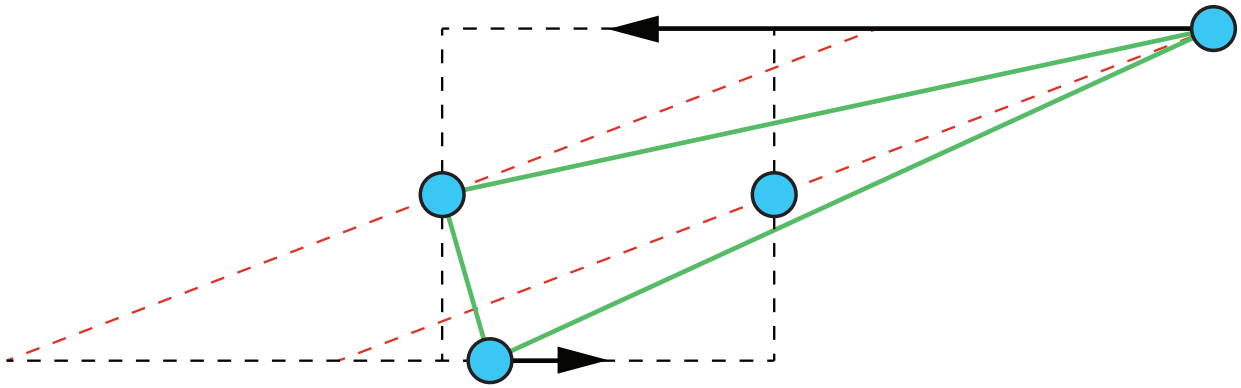
But the configuration may become non-convex by accident!

General algorithm: two opposite concordant, two finished



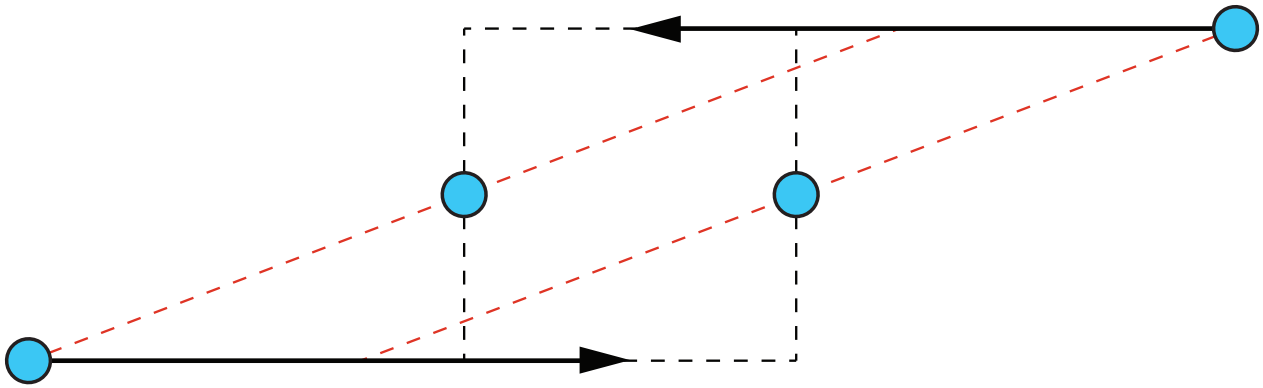
But the configuration may become non-convex by accident!

General algorithm: two opposite concordant, two finished



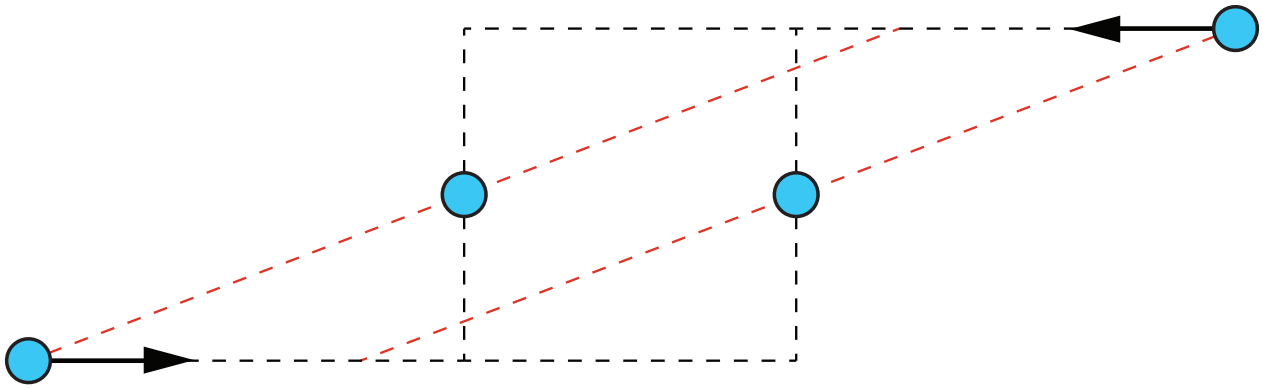
But the configuration may become non-convex by accident!

General algorithm: two opposite concordant, two finished



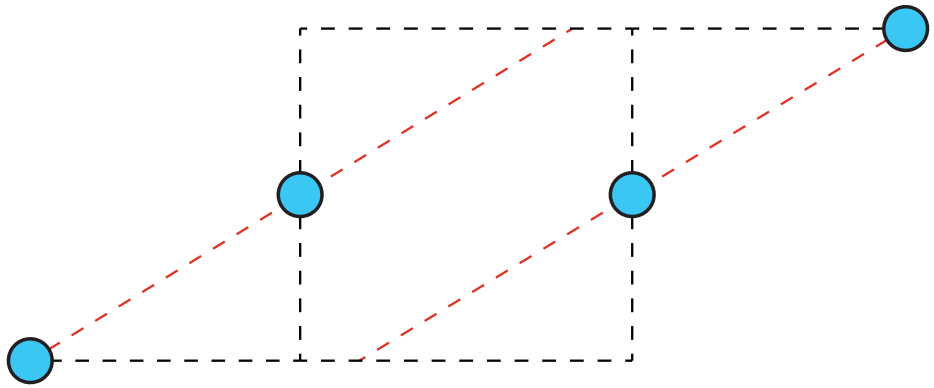
This can be prevented by making several shorter moves

General algorithm: two opposite concordant, two finished



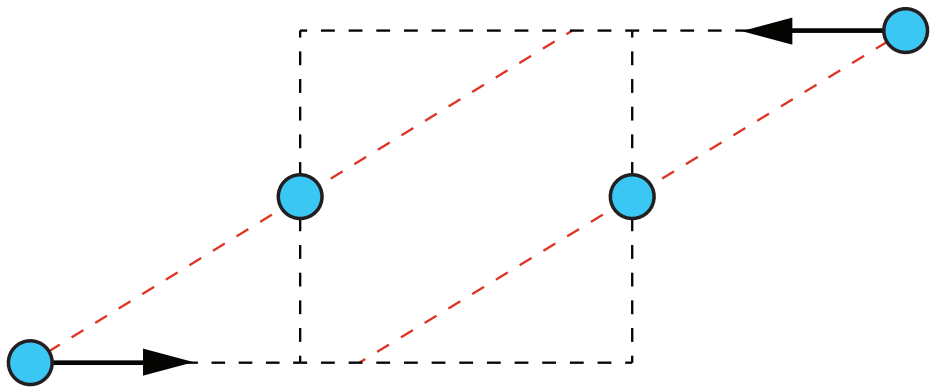
This can be prevented by making several shorter moves

General algorithm: two opposite concordant, two finished



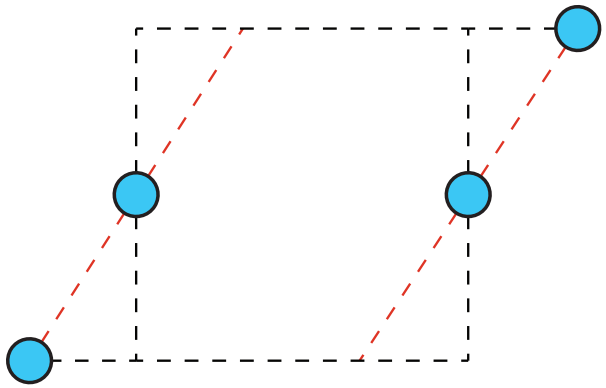
This can be prevented by making several shorter moves

General algorithm: two opposite concordant, two finished



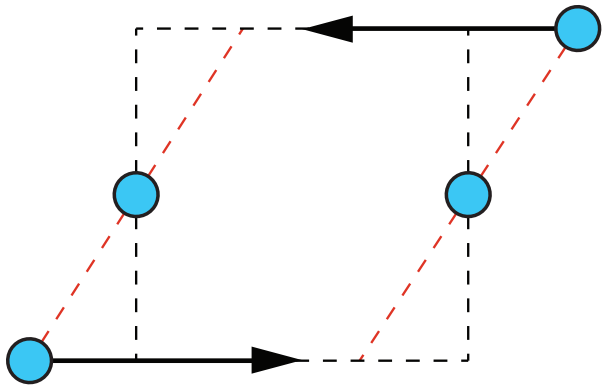
This can be prevented by making several shorter moves

General algorithm: two opposite concordant, two finished



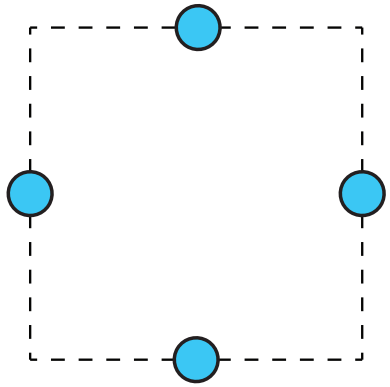
This can be prevented by making several shorter moves

General algorithm: two opposite concordant, two finished



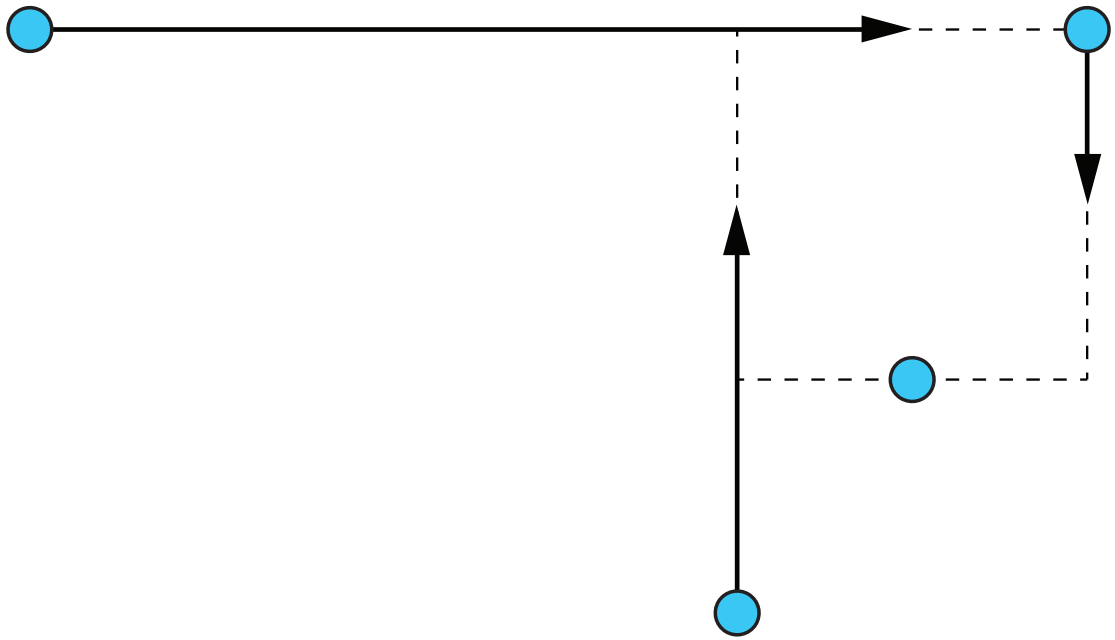
This can be prevented by making several shorter moves

General algorithm: two opposite concordant, two finished



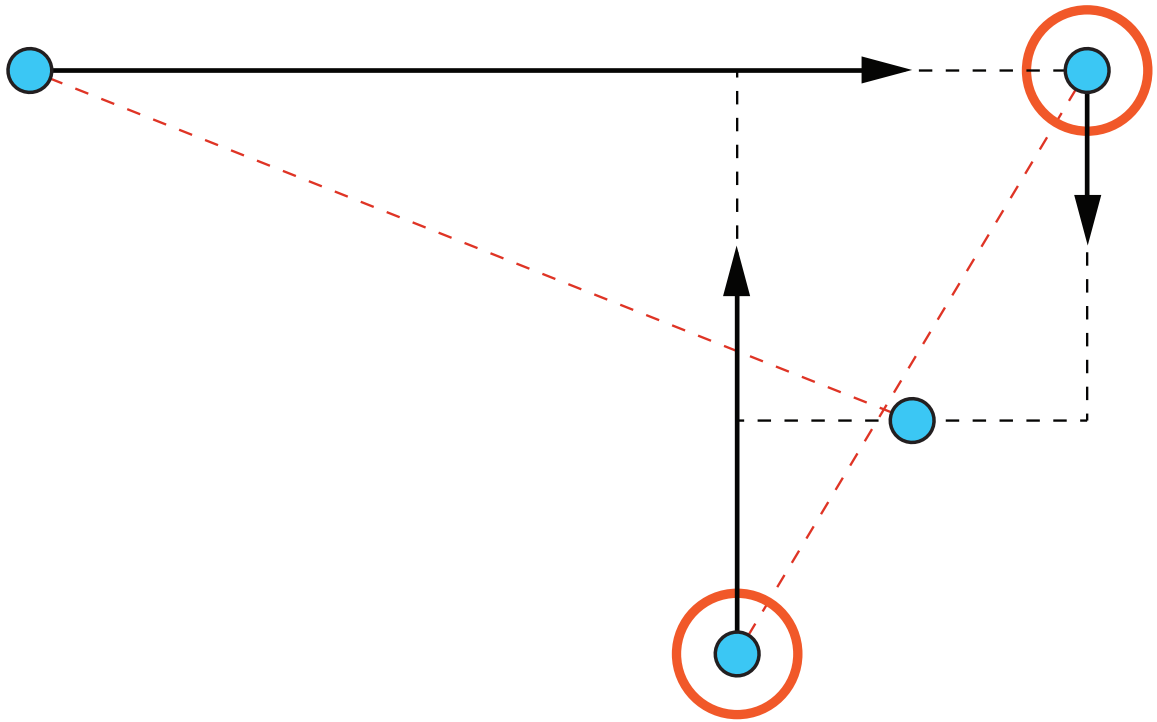
This can be prevented by making several shorter moves

General algorithm: all concordant



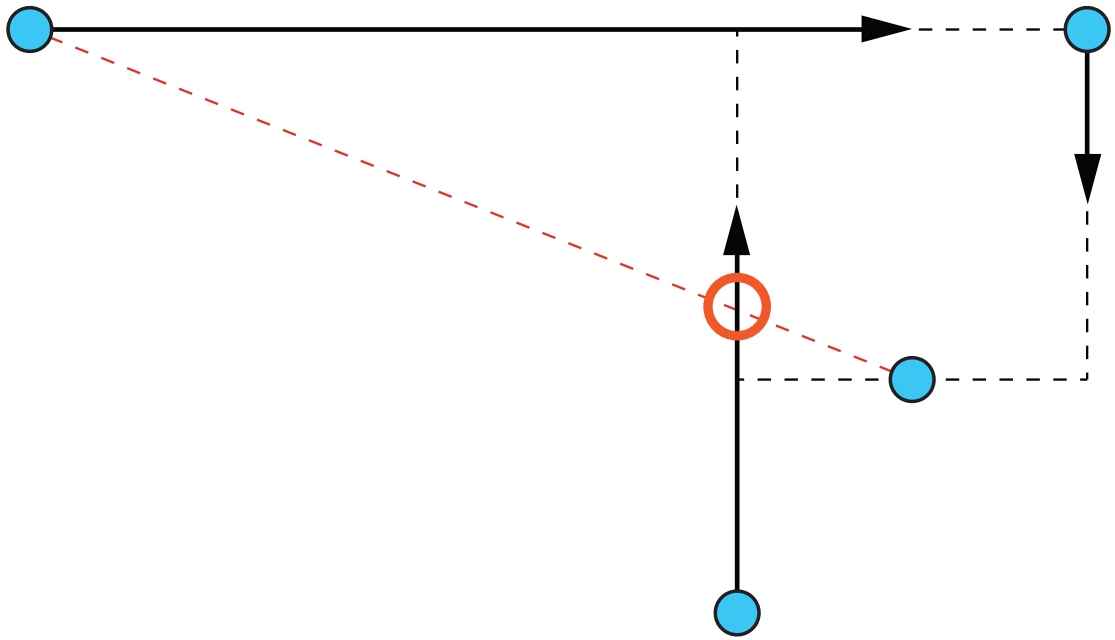
We let only the robots on the shortest diagonal move...

General algorithm: all concordant



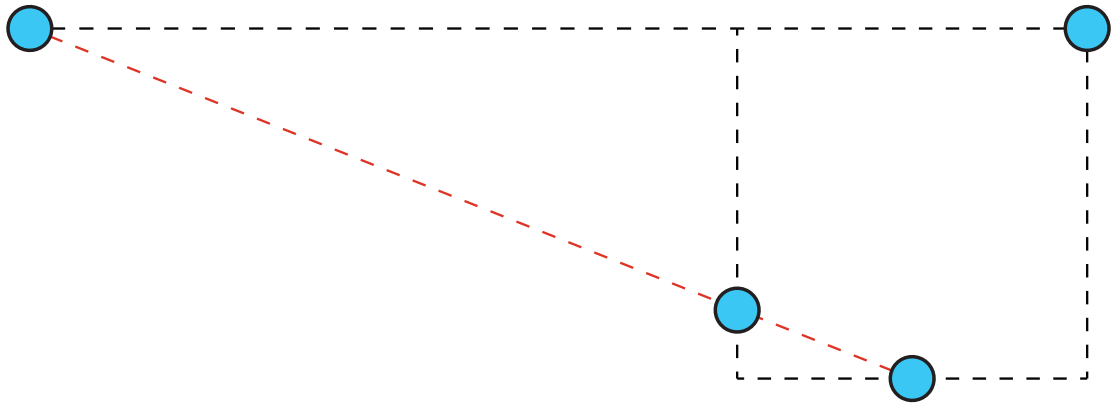
...Because it will remain the shortest as they move

General algorithm: all concordant



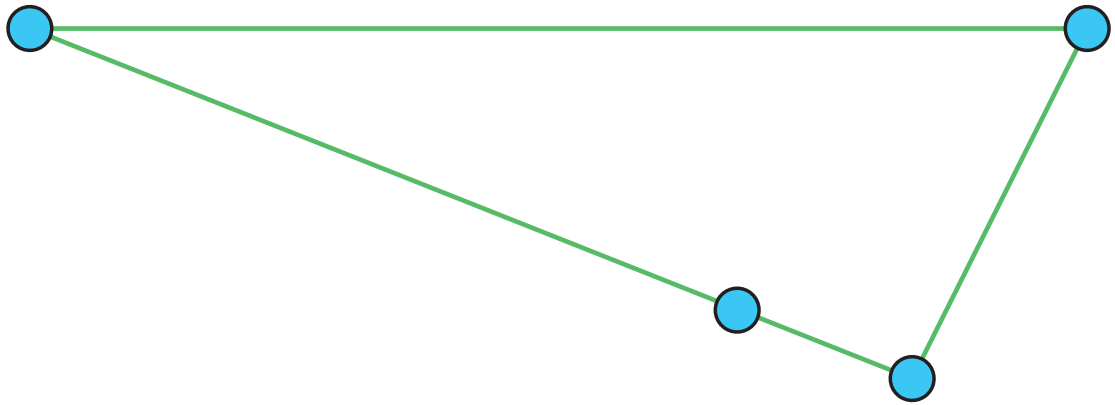
But one robot (not both!) may be "blocked" by the other diagonal

General algorithm: all concordant



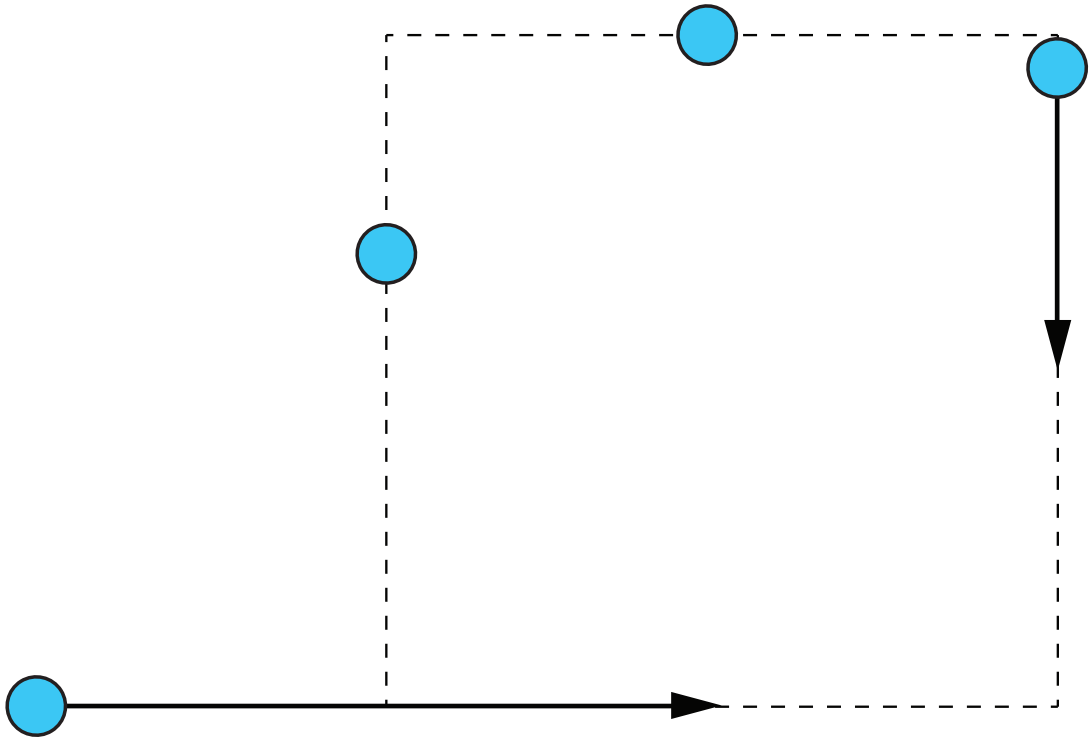
If so, only the blocked robot moves, and stops on the diagonal

General algorithm: all concordant



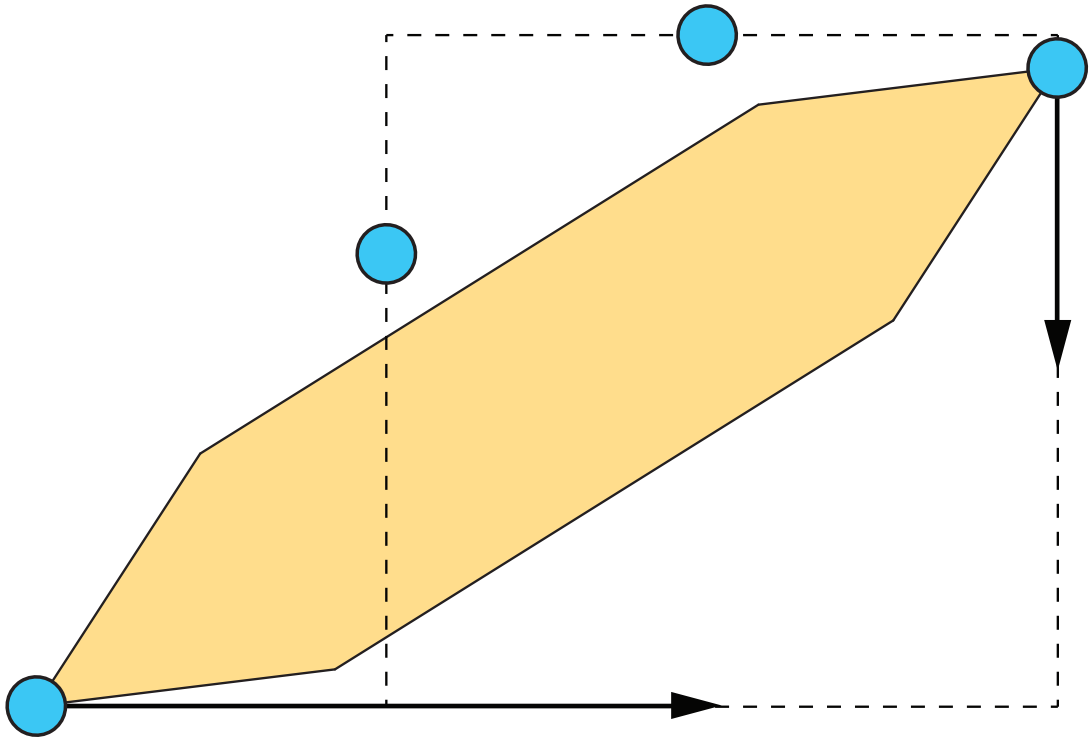
Then all robots behave coherently as in the non-convex case

General algorithm: two convergent robots



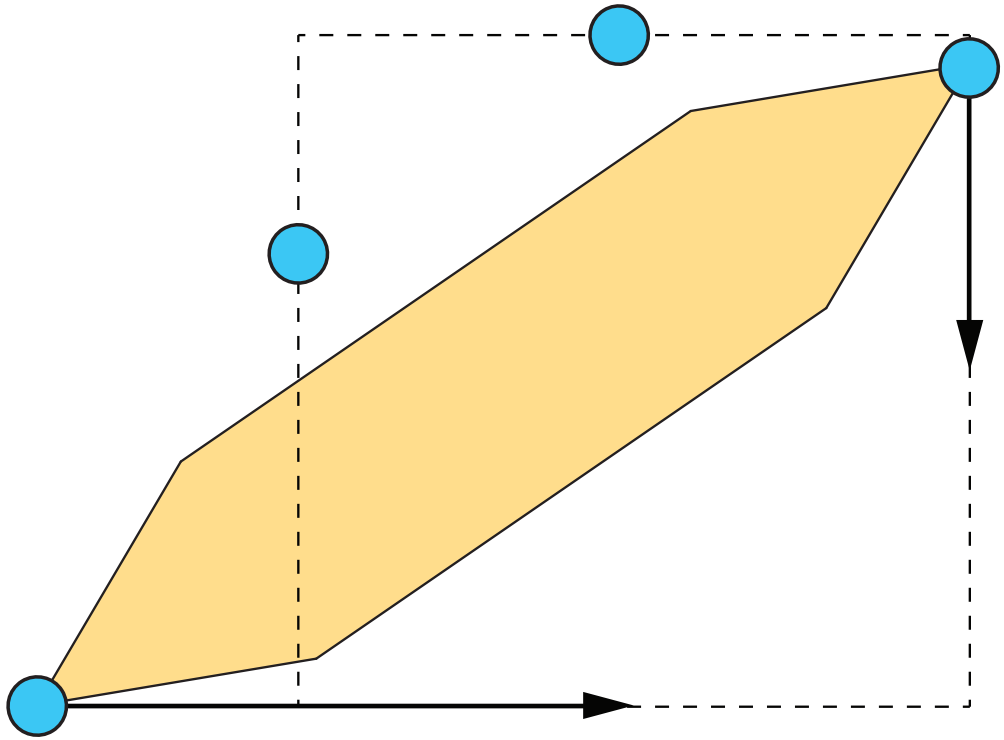
The convergent robots move, while the others wait

General algorithm: two convergent robots



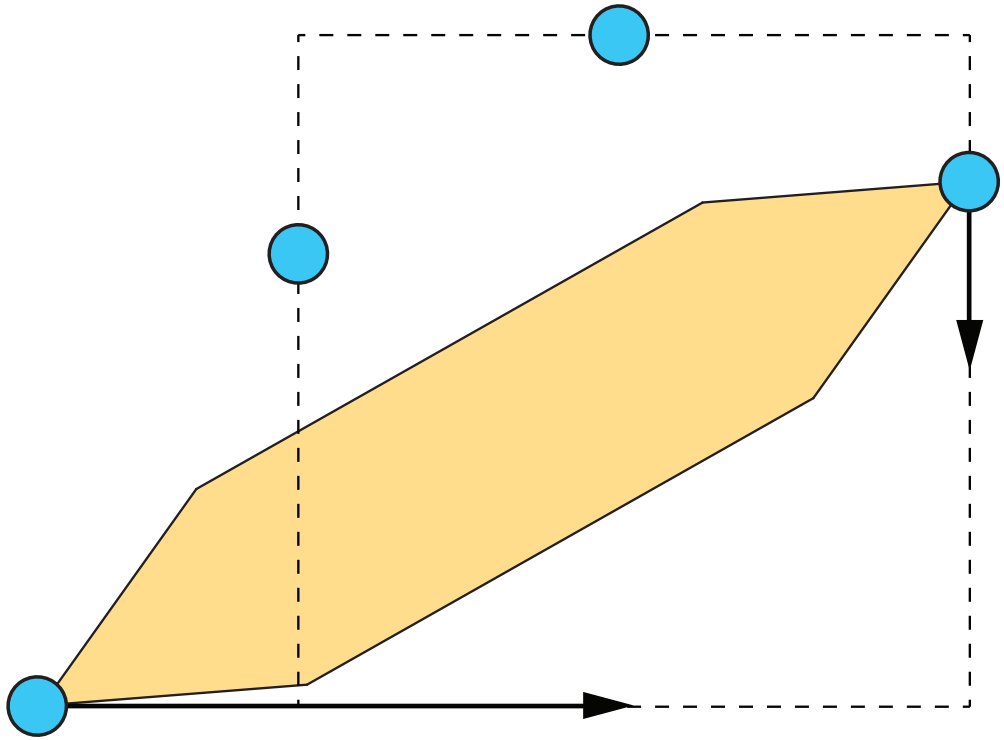
No thin hexagon can be formed by accident

General algorithm: two convergent robots



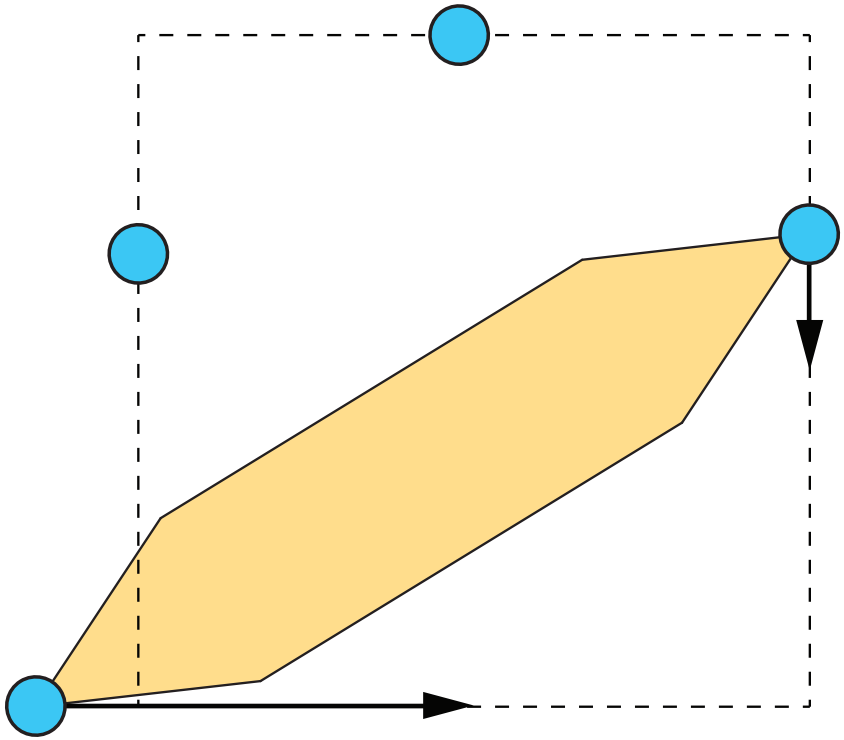
No thin hexagon can be formed by accident

General algorithm: two convergent robots



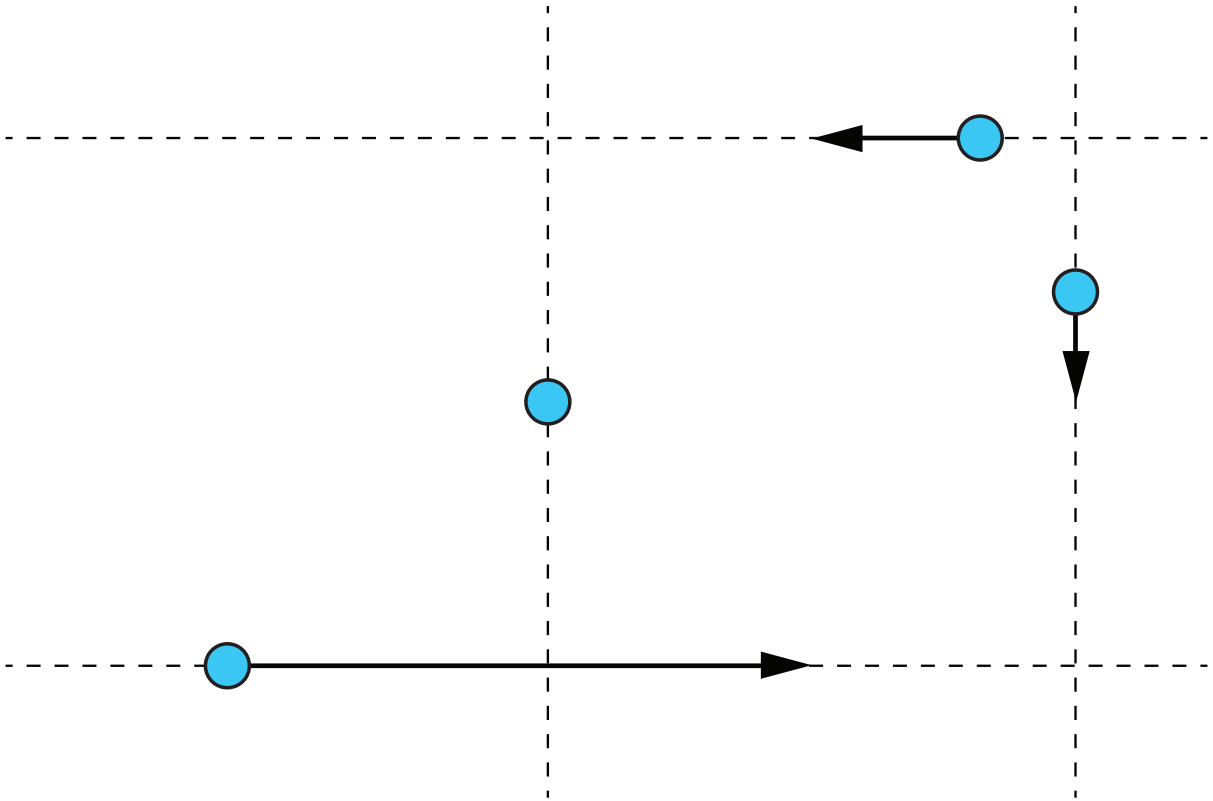
No thin hexagon can be formed by accident

General algorithm: two convergent robots



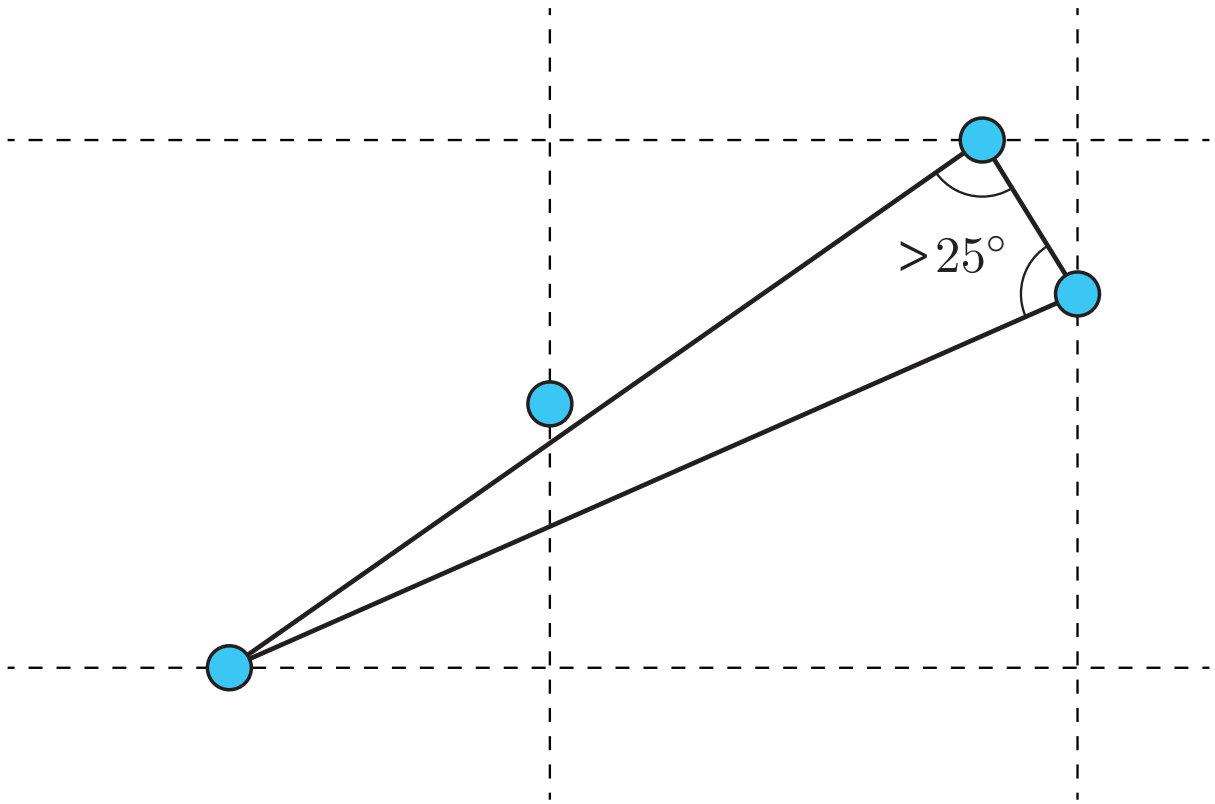
No thin hexagon can be formed by accident

General algorithm: last case



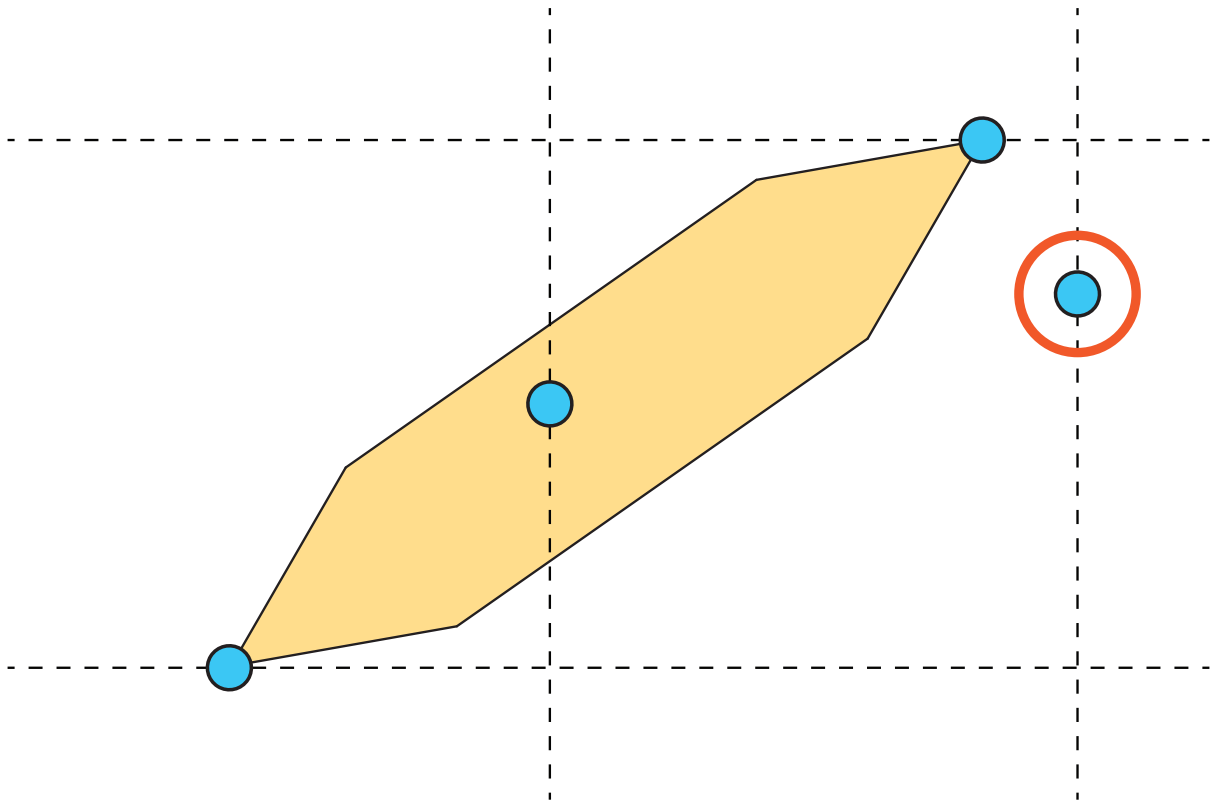
If only one robot is external...

General algorithm: last case



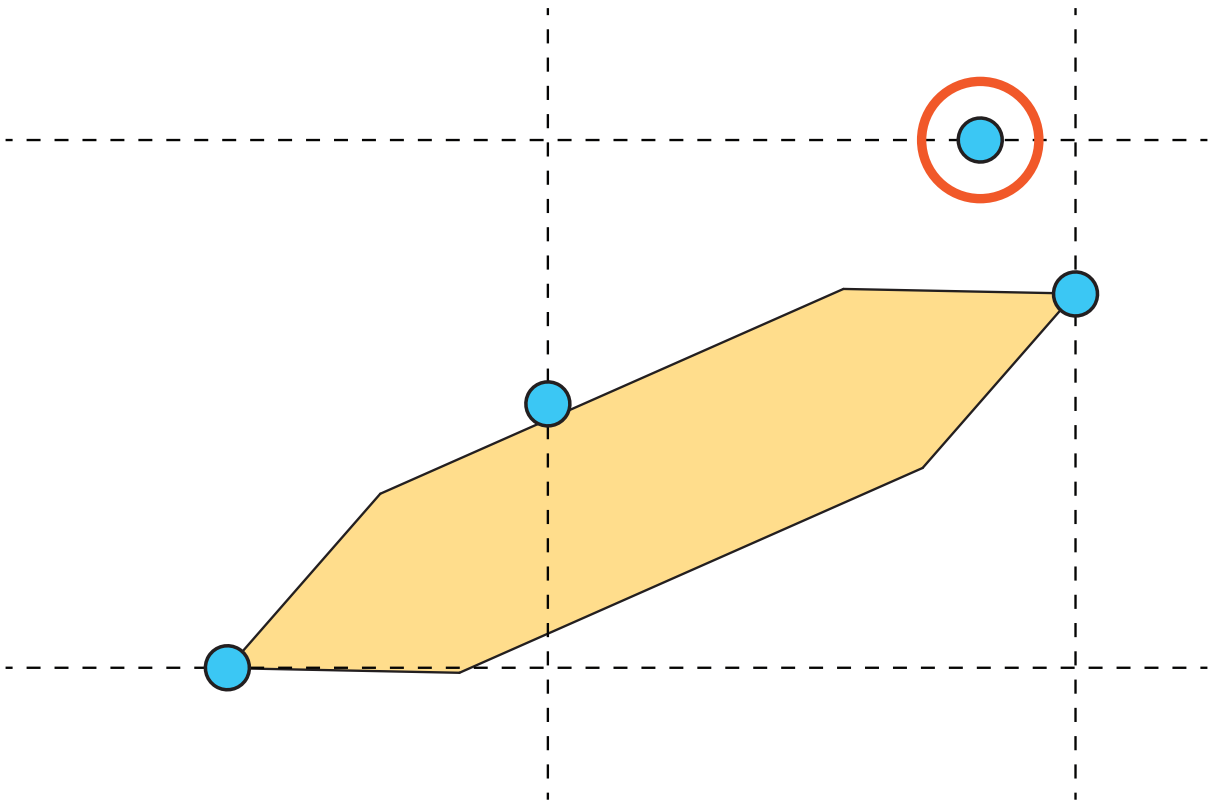
...The angles it forms with the two far robots are $< 25°$

General algorithm: last case



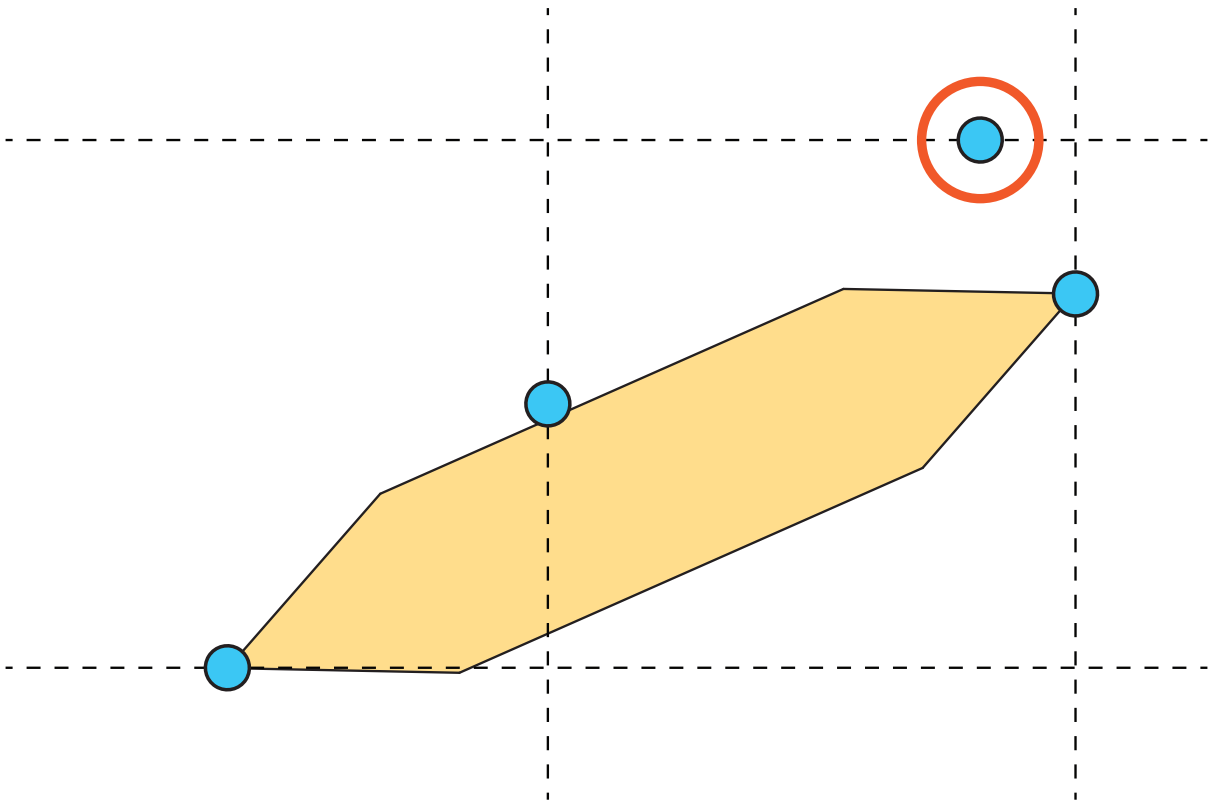
So a thin hexagon cannot be formed, because its angles are 50°

General algorithm: last case



So a thin hexagon cannot be formed, because its angles are 50°

General algorithm: last case



This yields a simple coordination protocol for the robots in all cases

Algorithm summary

The configuration is checked against each possible class, in the correct order!

1. Orthogonal diagonals
2. Thin hexagon
3. Non-convex
4. All concordant
5. Two convergent, two divergent
6. Two divergent, two divergent
7. One discordant

Algorithm summary

The configuration is checked against each possible class, in the correct order!

1. Orthogonal diagonals
2. Thin hexagon
3. Non-convex
4. All concordant
5. Two convergent, two divergent
6. Two divergent, two divergent
7. One discordant

Ensure that, when a class transition occurs,

- No robot is moving (to prevent inconsistent behaviors!)
- The resulting class has lower index (in the list above)

Algorithm summary

The configuration is checked against each possible class, in the correct order!

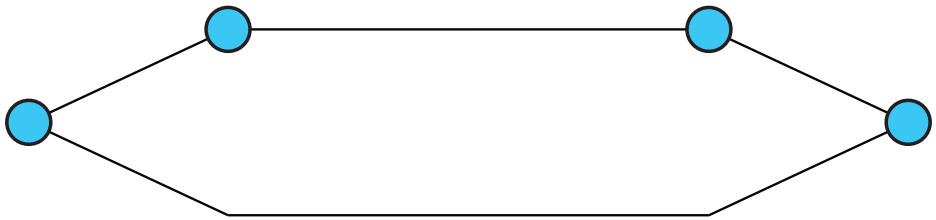
1. Orthogonal diagonals
2. Thin hexagon
3. Non-convex
4. All concordant
5. Two convergent, two divergent
6. Two divergent, two divergent
7. One discordant

Ensure that, when a class transition occurs,

- No robot is moving (to prevent inconsistent behaviors!)
- The resulting class has lower index (in the list above)

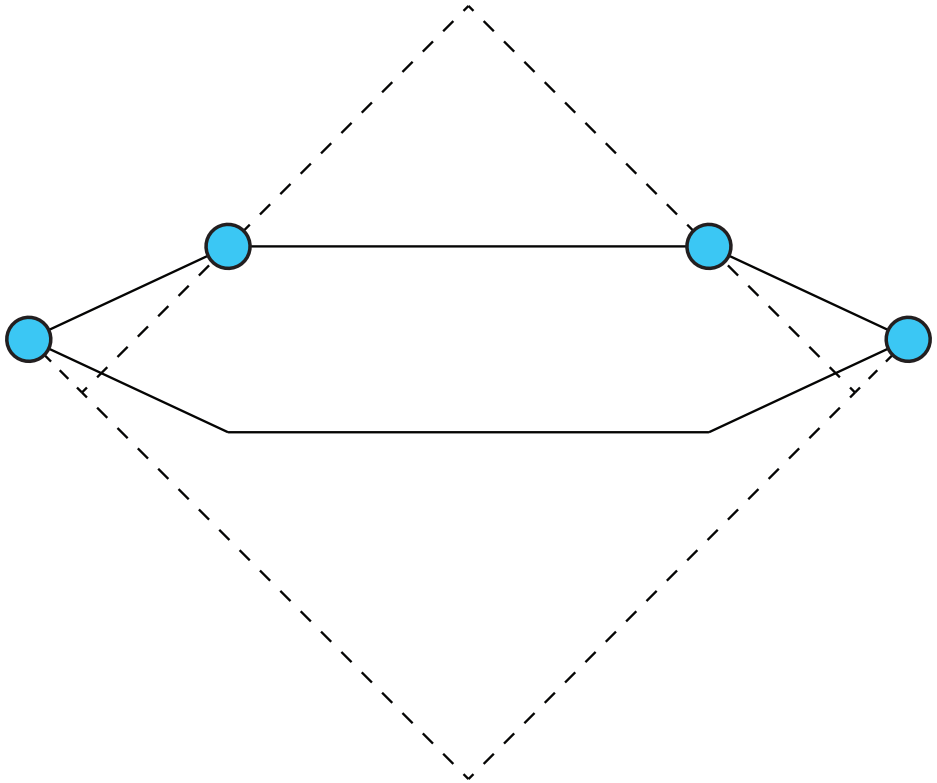
The last rule is broken in only one case!

Resolving the anomaly



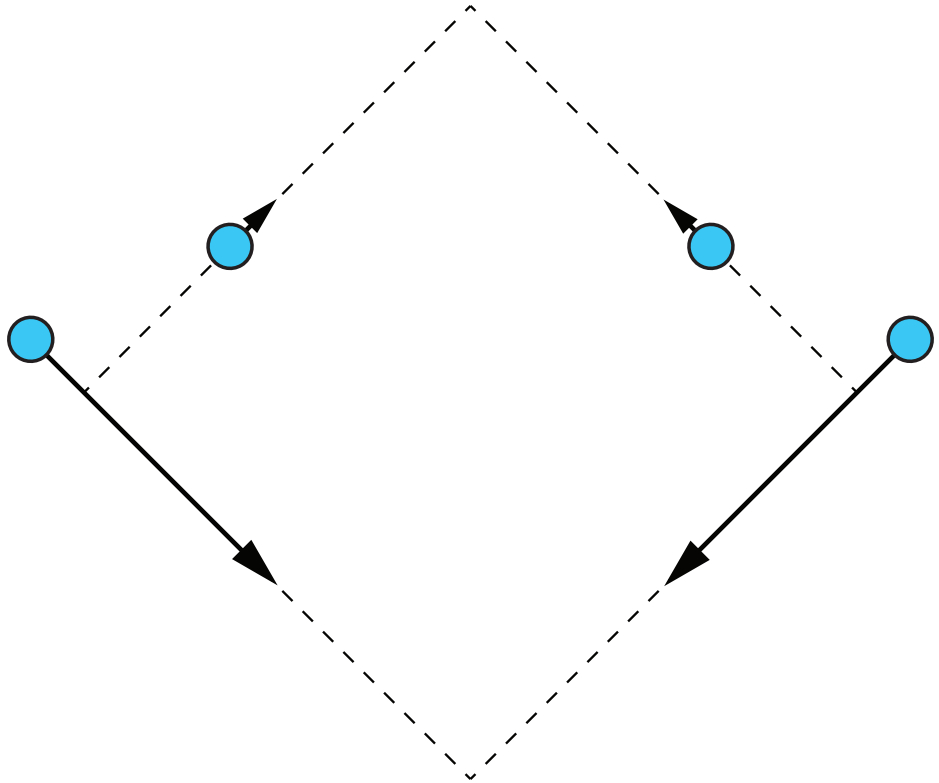
When all robots are on the same side of a thin hexagon...

Resolving the anomaly



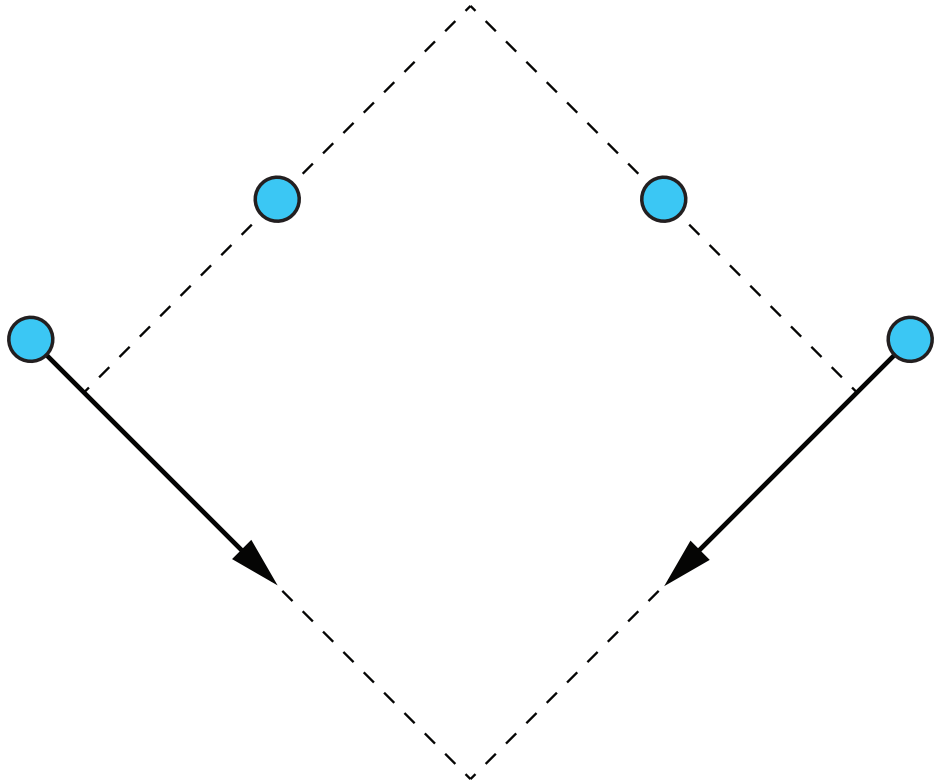
...They move to the vertices, and then apply the general algorithm

Resolving the anomaly



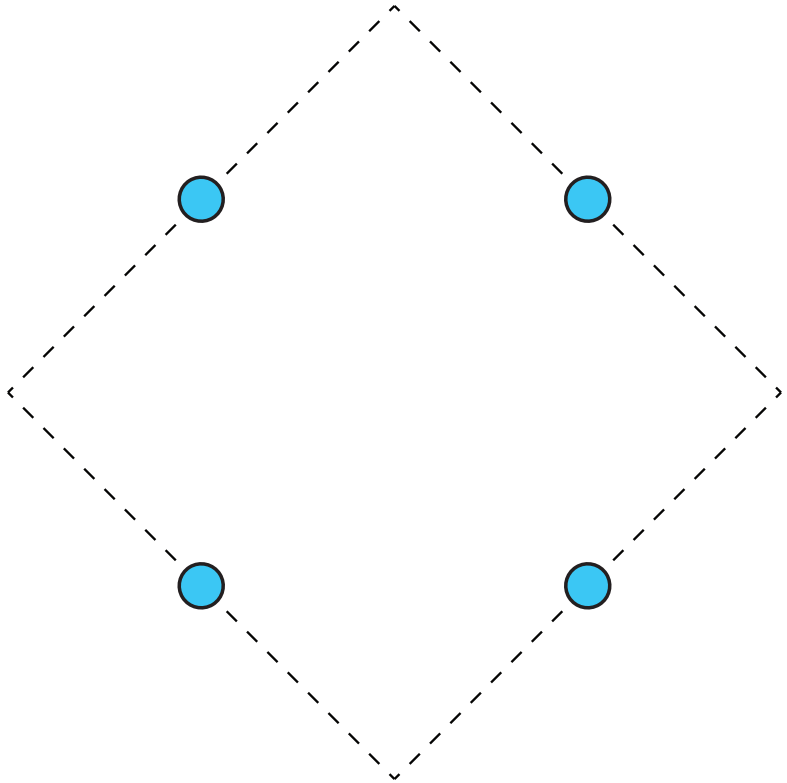
As a consequence, the internal robots move first

Resolving the anomaly



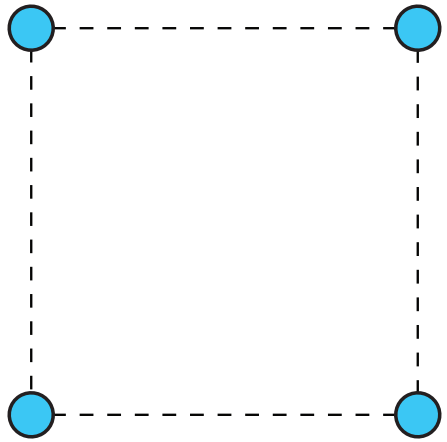
As a consequence, the internal robots move first

Resolving the anomaly



And finally the external robots move...

Resolving the anomaly



...Thus forming a square

Concluding remarks

The only solvable Pattern Formation problems for n robots are:

- **Single point** (except the case $n = 2$, which is unsolvable)
- **Regular n-gon** (now also for $n = 4$!)

Concluding remarks

The only solvable Pattern Formation problems for n robots are:

- **Single point** (except the case $n = 2$, which is unsolvable)
- **Regular n-gon** (now also for $n = 4$!)

For $n > 2$, this is true even if

- Robots are fully synchronous
- Robots have a common notion of "clockwise" (chirality)
- Robots always reach their destination (rigidity)

$\Longrightarrow$ For Pattern Formation problems, these features are computationally irrelevant!