

Seminar 6 – Mobile Robots:  
Gathering and Pattern Formation  
Distributed Computing in Anonymous Dynamic Systems

Giovanni Viglietta

Rome – March 13, 2024

## Syllabus

- Anonymous Networks
  - Introduction and basic algorithms for static networks
  - Dynamicity and history trees
  - Optimal computation in networks with and without leaders
  - Computation in dynamic congested networks
- Population Protocols
  - Introduction and basic algorithmic techniques
  - Leader election in Mediated Population Protocols
- Mobile Robots
  - Gathering and Pattern Formation in the plane
  - Meeting in a polygon by oblivious robots

## Exam

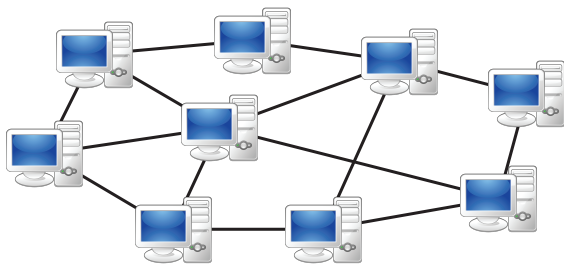
Pre-recorded 10-minute presentation video on one of the papers that will be suggested at the end of the course.

# Today's seminar

- Oblivious mobile robots
- Convergence, Gathering, Near-Gathering
- Uniform Circle Formation problem

# Distributed systems: overview

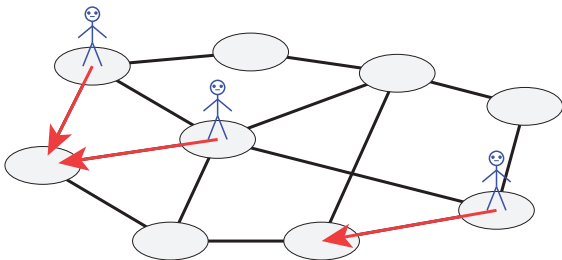
The traditional setting is a network of computers, each of which has its own local memory. Neighboring computers can communicate by sending messages.



A typical problem in this setting is electing a unique leader in a network of anonymous computers (i.e., where all computers are identical and execute the same algorithm).

## Distributed systems: overview

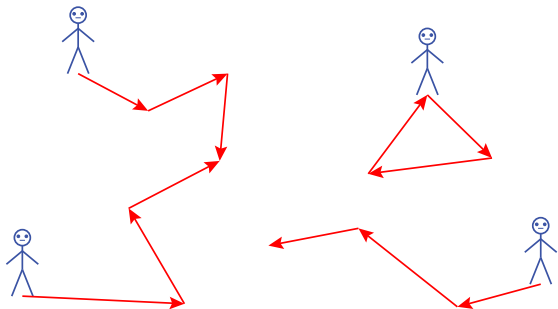
The standard model evolved into a more dynamic one: now the computers can move within the network (they are “robots”) and write messages on shared whiteboards at each node.



A typical problem in this setting is gathering in the same node. Again, robots are anonymous, i.e., they are all identical and execute the same algorithm.

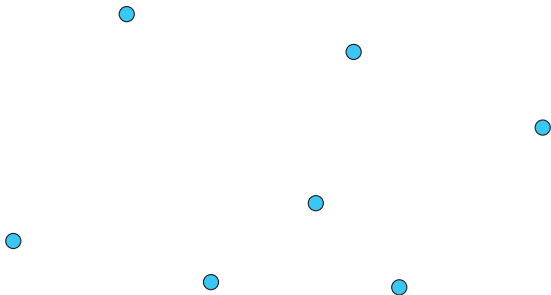
# Distributed systems: overview

A further evolution of the model saw robots no longer constrained to a fixed network, but able to freely move in space.



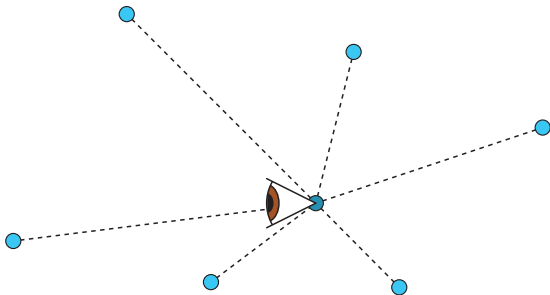
Each robot is now a computational unit that can sense the positions of other robots and compute its next destination.

## Anonymous robots sensing and moving



We consider a *swarm* of anonymous robots in the Euclidean plane.

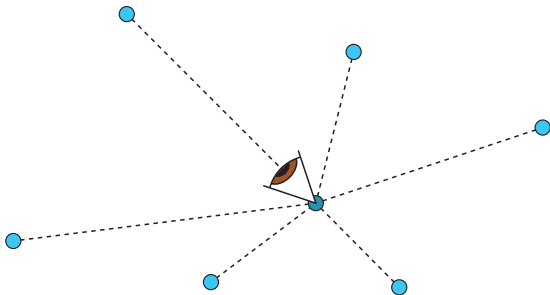
## Anonymous robots sensing and moving



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

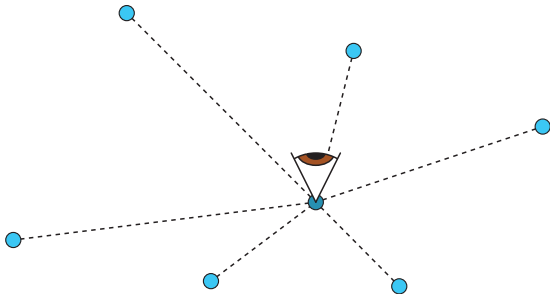


## Anonymous robots sensing and moving



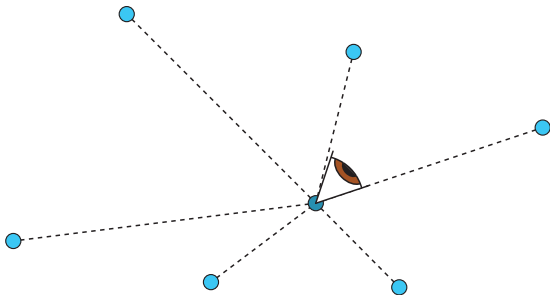
Each robot can see the positions of all other robots...

# Anonymous robots sensing and moving



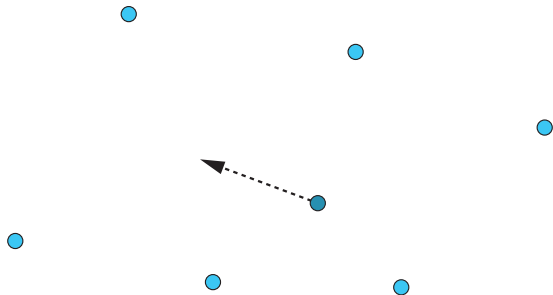
Each robot can see the positions of all other robots...

# Anonymous robots sensing and moving



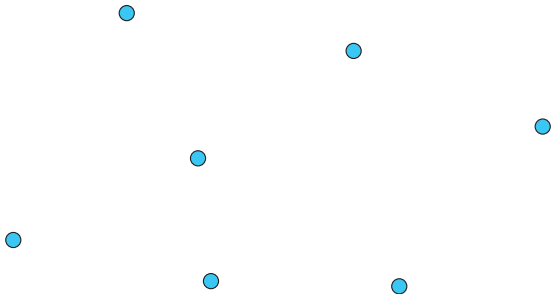
Each robot can see the positions of all other robots...

## Anonymous robots sensing and moving



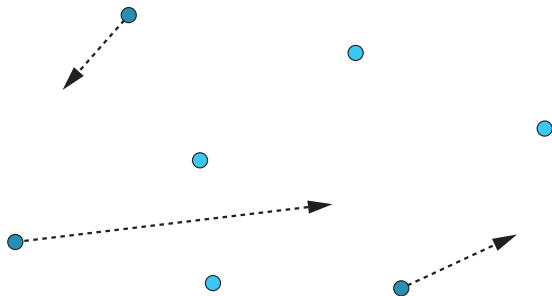
...And move according to a deterministic algorithm.

## Anonymous robots sensing and moving



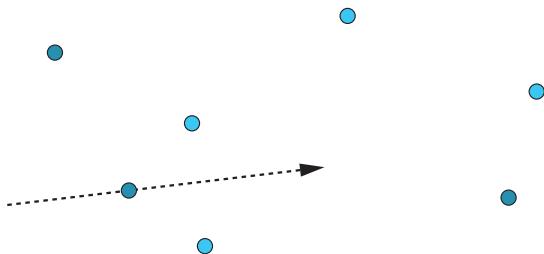
...And move according to a deterministic algorithm.

## Anonymous robots sensing and moving



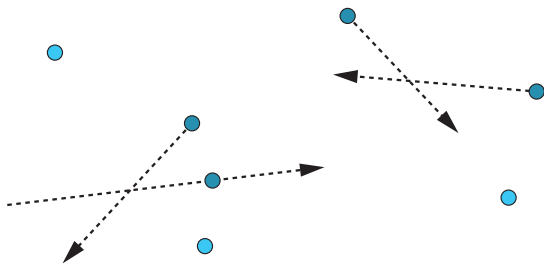
Different robots are activated asynchronously.

## Anonymous robots sensing and moving



Different robots are activated asynchronously.

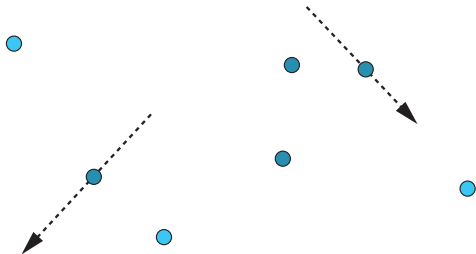
# Anonymous robots sensing and moving



Different robots are activated asynchronously.



## Anonymous robots sensing and moving

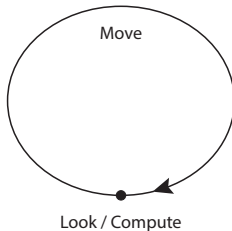


Different robots are activated asynchronously.

Robots are:

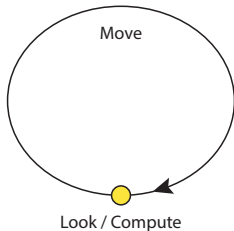
- **Dimensionless** (robots are modeled as geometric points)
- **Anonymous** (no unique identifiers)
- **Homogeneous** (the same algorithm is executed by all robots)
- **Deterministic** (the algorithm has no randomization)
- **Autonomous** (no centralized control)
- **Oblivious** (no memory of past events and observations)
- **Silent** (no explicit way of communicating)
- **Disoriented** (robots do not share a common reference frame)
  - No common unit distance
  - No common compass
  - No common notion of clockwise direction

# Life cycle and asynchrony



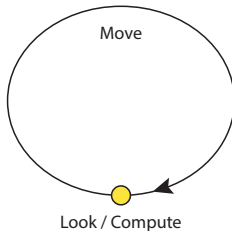
Each robot perpetually repeats a Look/Compute/Move cycle.

# Life cycle and asynchrony



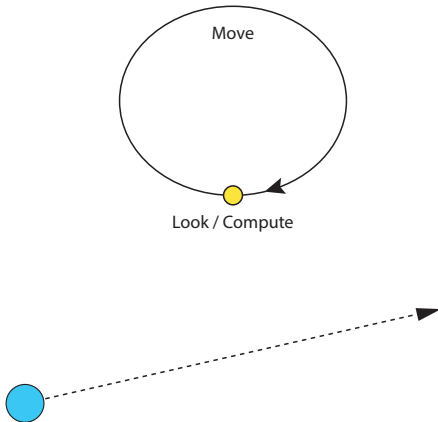
Each robot perpetually repeats a Look/Compute/Move cycle.

# Life cycle and asynchrony



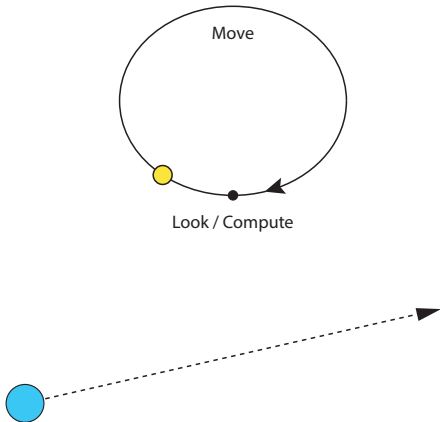
In a Look phase, a snapshot is taken of all visible robots.

# Life cycle and asynchrony



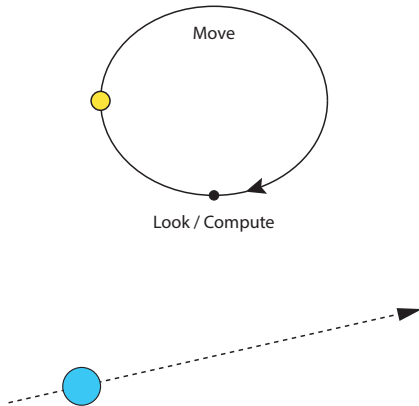
A destination is computed based only on the last snapshot.

# Life cycle and asynchrony



The destination point is approached at unpredictable speed.

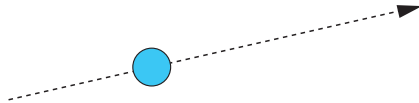
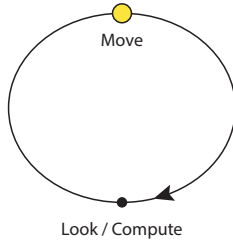
# Life cycle and asynchrony



The destination point is approached at unpredictable speed.

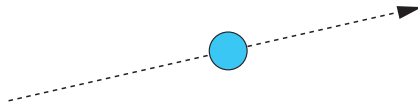
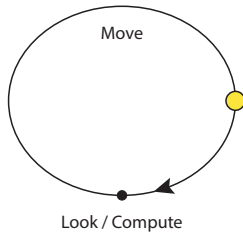


# Life cycle and asynchrony



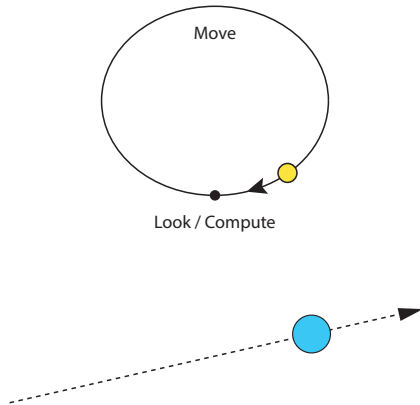
The destination point is approached at unpredictable speed.

# Life cycle and asynchrony



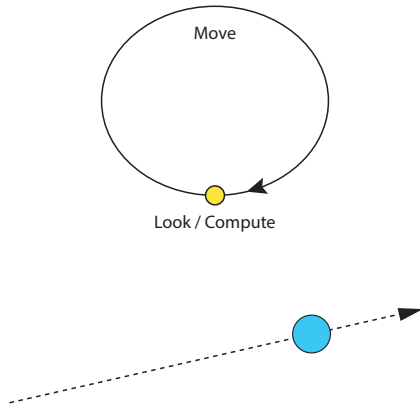
The destination point is approached at unpredictable speed.

# Life cycle and asynchrony



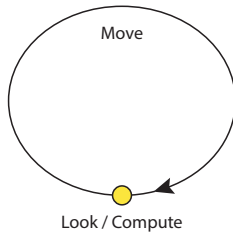
The destination point is approached at unpredictable speed.

# Life cycle and asynchrony



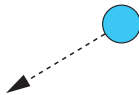
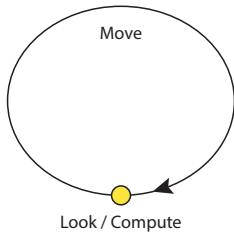
The robot may unpredictably stop before reaching the destination...

# Life cycle and asynchrony



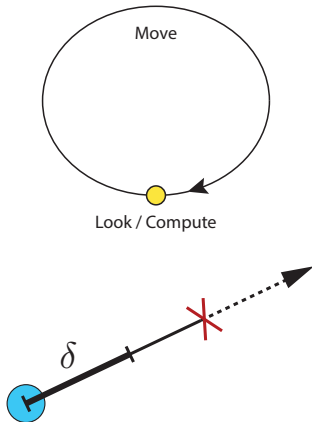
...and execute a new Look/Compute phase.

# Life cycle and asynchrony



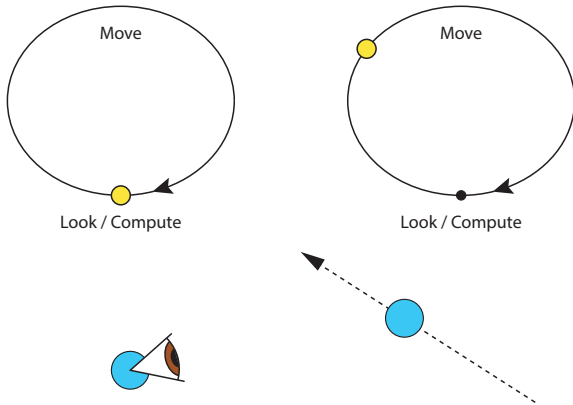
...and execute a new Look/Compute phase.

# Life cycle and asynchrony



At each cycle, a robot is guaranteed to move by at least  $\delta$ .

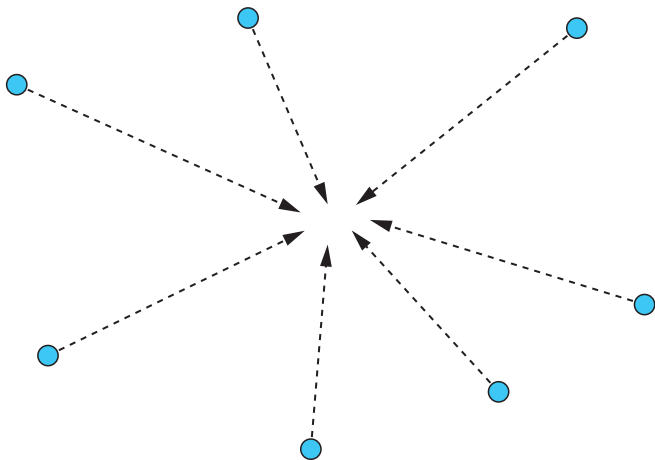
# Life cycle and asynchrony



Different robots execute independent cycles, asynchronously.

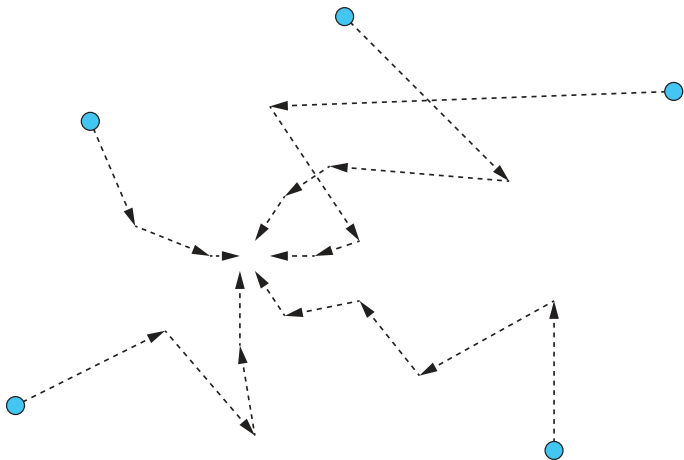


## Gathering-like problems



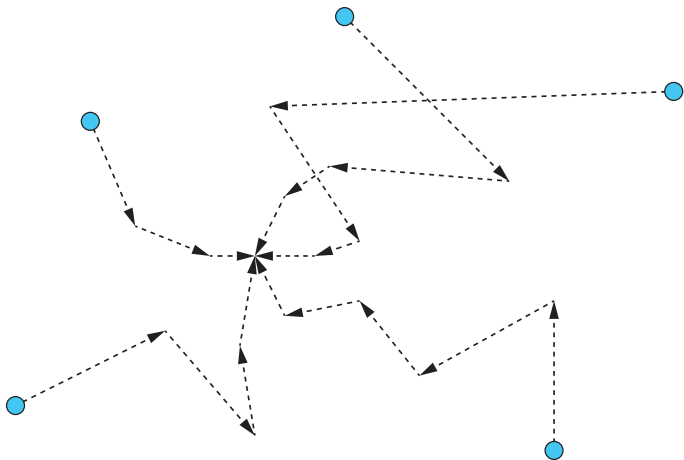
Perhaps the most studied class of problems:  
Design an algorithm that makes all robots “gather”.

# Convergence problem



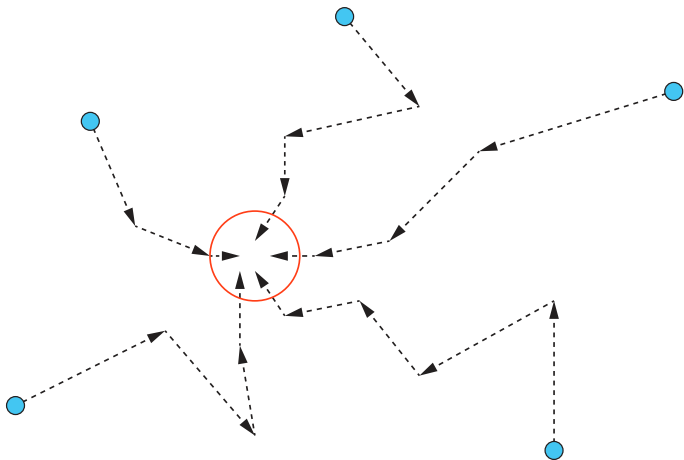
**Convergence:** Make all robots converge to the same point (possibly never actually reaching it, and possibly colliding).

# Gathering problem



**Gathering:** Make all robots reach the same point in finite time.

# Near-Gathering problem



**Near-Gathering:** Make all robots reach a small-enough area, avoiding collisions.

## Case study: two robots



Suppose the swarm consists of only two robots.

## Case study: two robots



**Strategy:** Move to the midpoint.

## Case study: two robots



It solves the Gathering problem if robots are fully synchronous...

## Case study: two robots



...But it only solves Convergence if robots are asynchronous.



## Case study: two robots



...But it only solves Convergence if robots are asynchronous.

## Case study: two robots



...But it only solves Convergence if robots are asynchronous.

## Case study: two robots



...But it only solves Convergence if robots are asynchronous.

## Case study: two robots



...But it only solves Convergence if robots are asynchronous.

## Case study: two robots



...But it only solves Convergence if robots are asynchronous.

## Case study: two robots



...But it only solves Convergence if robots are asynchronous.

## Case study: two robots



...But it only solves Convergence if robots are asynchronous.

## Case study: two robots



**Modified strategy:** Reduce the distance by a fraction of  $1/3$ .



## Case study: two robots



It solves Near-Gathering if robots are fully synchronous...

## Case study: two robots



It solves Near-Gathering if robots are fully synchronous...

## Case study: two robots



It solves Near-Gathering if robots are fully synchronous...

## Case study: two robots



...But the robots may collide if they are asynchronous.

## Case study: two robots



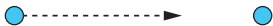
...But the robots may collide if they are asynchronous.

## Case study: two robots



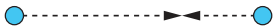
...But the robots may collide if they are asynchronous.

## Case study: two robots



...But the robots may collide if they are asynchronous.

## Case study: two robots



...But the robots may collide if they are asynchronous.



## Case study: two robots



...But the robots may collide if they are asynchronous.

# Simple Convergence algorithm



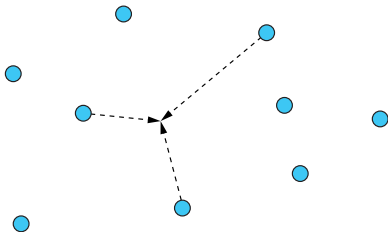
R. Cohen, D. Peleg

Convergence Properties of the Gravitational Algorithms in Asynchronous Robot Systems

SIAM Journal on Computing 34(6):1516–1528, 2005

**Strategy:** Move to the center of gravity.

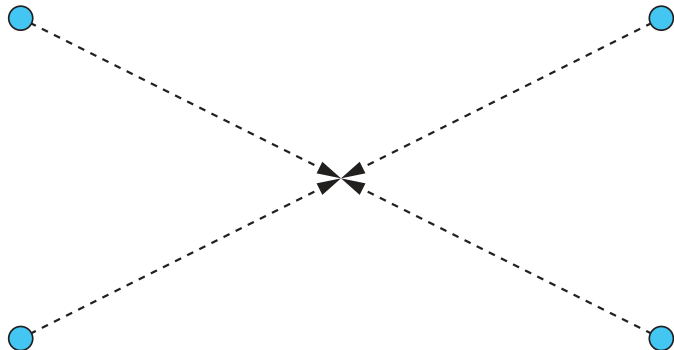
**Requirement:** The visibility range must be infinite.



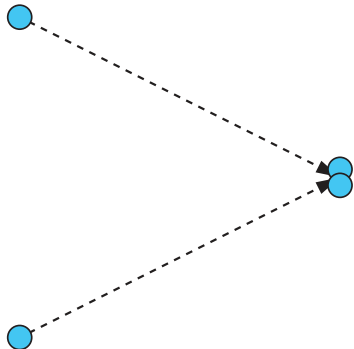
## Simple Convergence algorithm: example



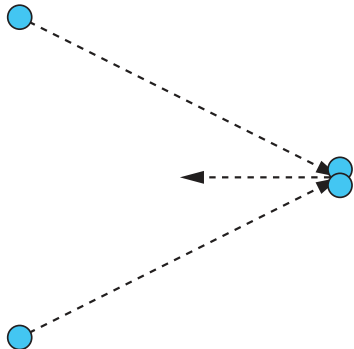
## Simple Convergence algorithm: example



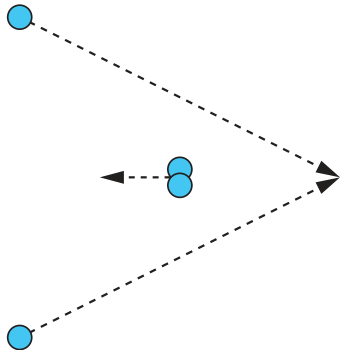
## Simple Convergence algorithm: example



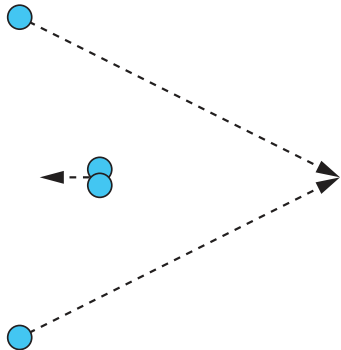
## Simple Convergence algorithm: example



# Simple Convergence algorithm: example

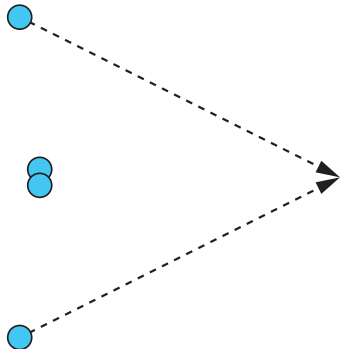


## Simple Convergence algorithm: example





## Simple Convergence algorithm: example



## Simple Convergence algorithm: example



## Simple Convergence algorithm: example



## Simple Convergence algorithm: example





H. Ando, Y. Oasa, I. Suzuki, M. Yamashita

Distributed Memoryless Point Convergence Algorithm for Mobile Robots with Limited Visibility

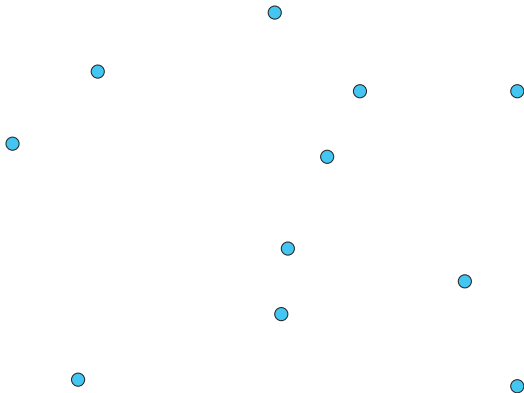
IEEE Trans. Robot. Autom. 15(5):818–828, 1999

**Strategy:** Move to the center of the smallest enclosing circle of the visible robots, without going too far from any of them.

**Remark:** It also works for robots with *limited visibility*, i.e., robots with a finite (and equal) visibility range.

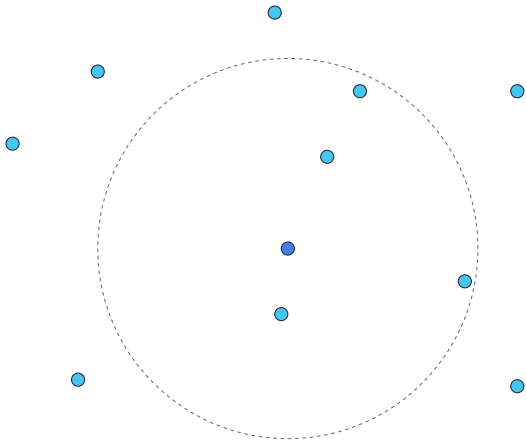
**Requirement:** Each move must be “instantaneous” (i.e., no robot can see another robot in the middle of a movement).

## Convergence with limited visibility: details



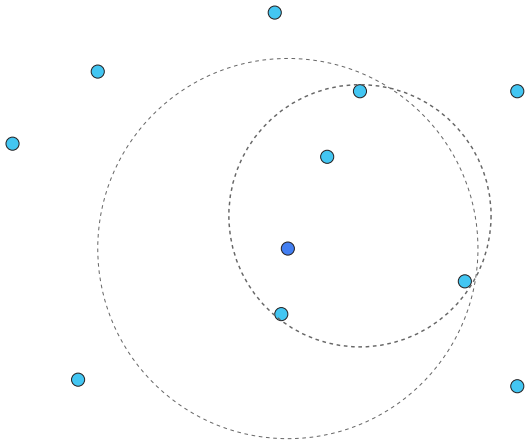
Each robot moves in the direction of the center of the smallest enclosing circle of the visible robots.

## Convergence with limited visibility: details



Each robot moves in the direction of the center of the smallest enclosing circle of the visible robots.

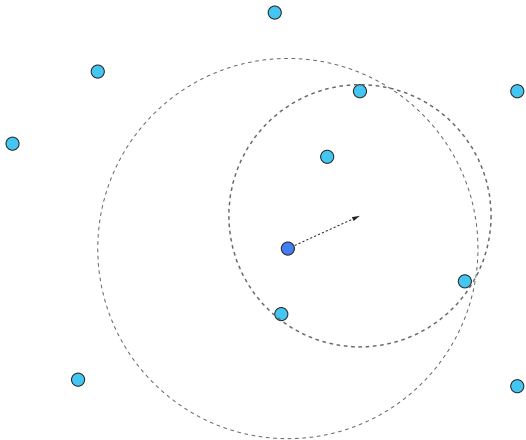
## Convergence with limited visibility: details



Each robot moves in the direction of the center of the smallest enclosing circle of the visible robots.

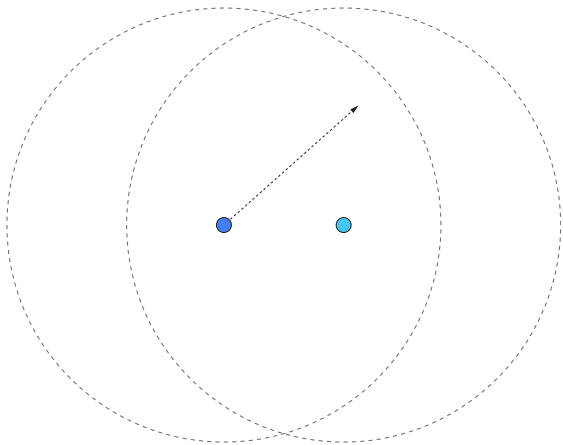


## Convergence with limited visibility: details



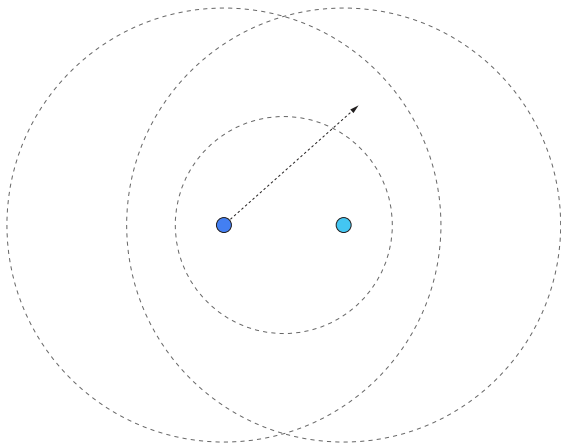
Each robot moves in the direction of the center of the smallest enclosing circle of the visible robots.

## Convergence with limited visibility: details



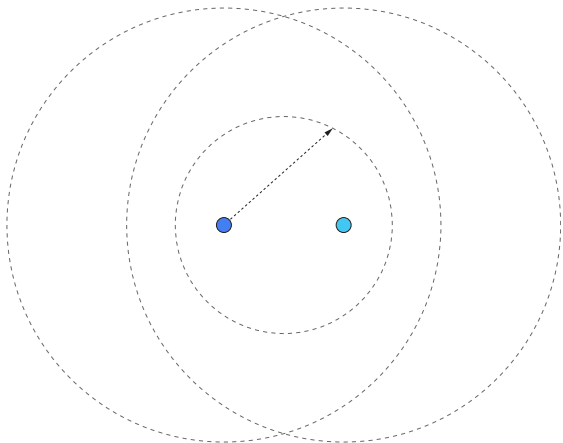
However, it does not get too far from other visible robots.

## Convergence with limited visibility: details



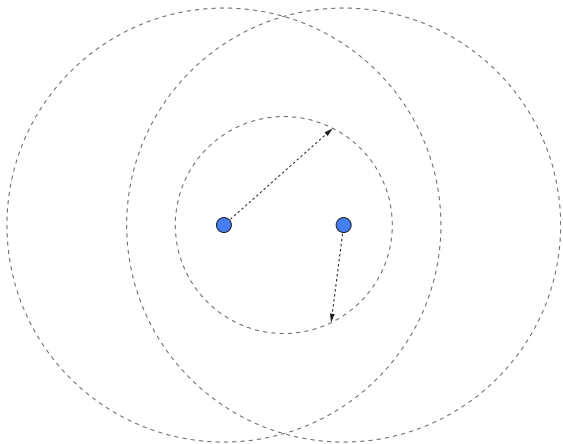
For each other visible robot, it remains within the circle centered at their midpoint and with diameter equal to their radius of visibility.

## Convergence with limited visibility: details



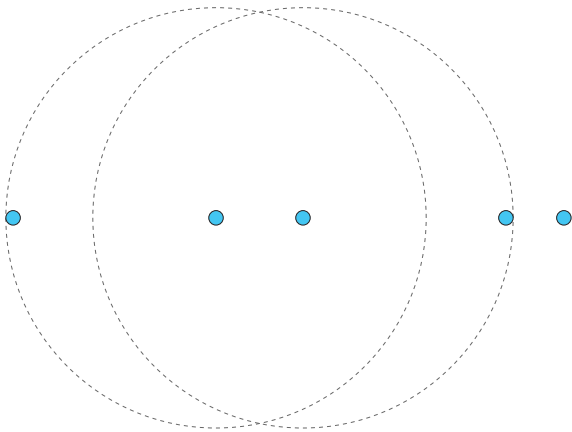
For each other visible robot, it remains within the circle centered at their midpoint and with diameter equal to their radius of visibility.

## Convergence with limited visibility: details



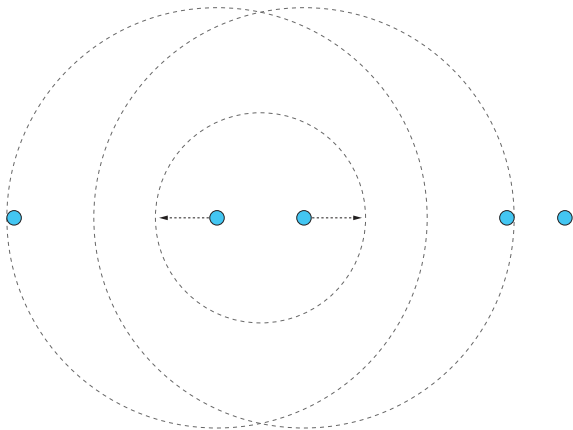
This way, if two robots see each other, they will keep seeing each other after they move.

# The algorithm may fail if moves are not instantaneous



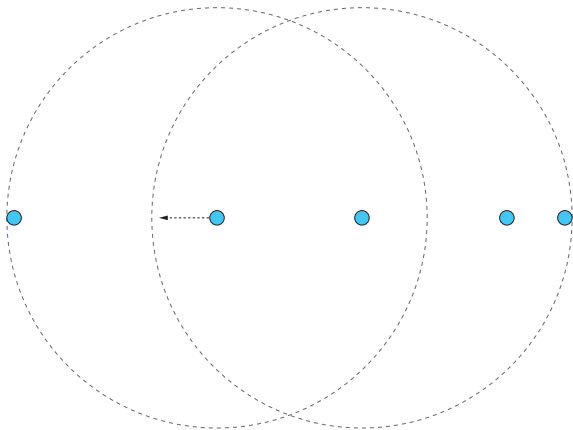
If moves are not instantaneous, some robots may lose vision of each other and converge to different points.

# The algorithm may fail if moves are not instantaneous



If moves are not instantaneous, some robots may lose vision of each other and converge to different points.

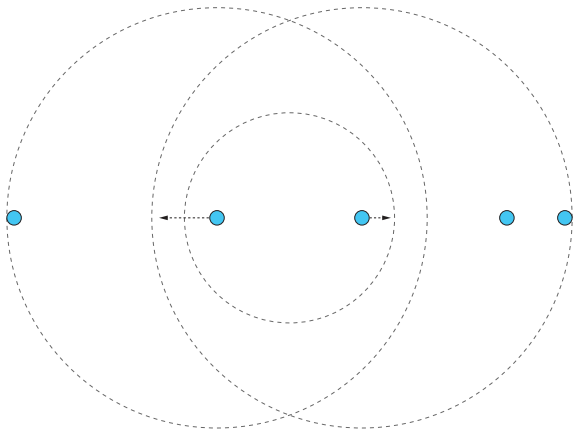
## The algorithm may fail if moves are not instantaneous



If moves are not instantaneous, some robots may lose vision of each other and converge to different points.

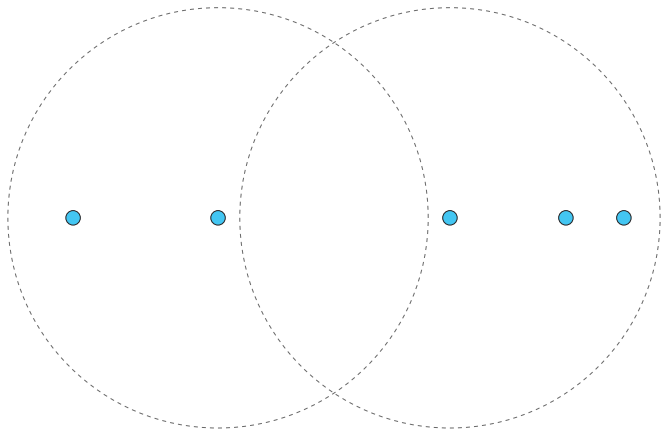


# The algorithm may fail if moves are not instantaneous



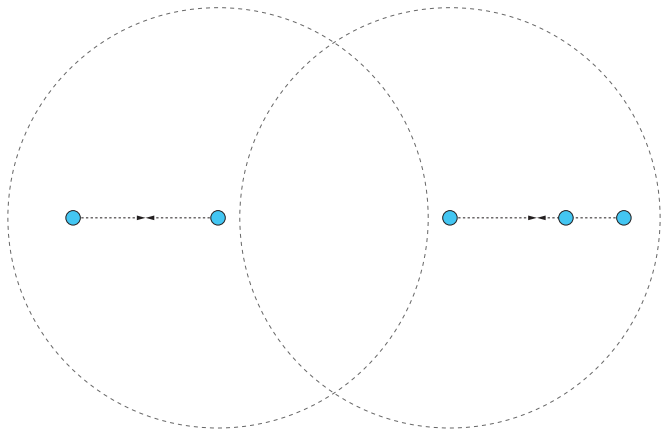
If moves are not instantaneous, some robots may lose vision of each other and converge to different points.

## The algorithm may fail if moves are not instantaneous



If moves are not instantaneous, some robots may lose vision of each other and converge to different points.

# The algorithm may fail if moves are not instantaneous



If moves are not instantaneous, some robots may lose vision of each other and converge to different points.

## The algorithm may fail if moves are not instantaneous



If moves are not instantaneous, some robots may lose vision of each other and converge to different points.

# Gathering with limited visibility



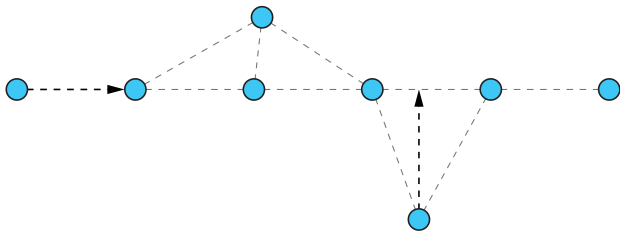
P. Flocchini, G. Prencipe, N. Santoro, P. Widmayer

Gathering of Asynchronous Robots with Limited Visibility

Theoretical Computer Science 337:147–168, 2005

**Strategy:** If you see only robots “above” you, move upward.  
Otherwise, if you see only robots to the “right”, move rightward.  
Make sure you do not lose vision with other robots.

**Requirement:** All robots must agree on up and right directions.



# Near-Gathering with limited visibility



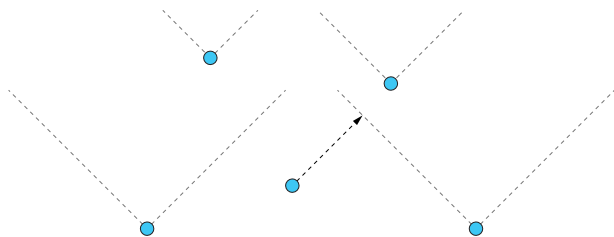
L. Pagli, G. Prencipe, G. Viglietta

Getting Close Without Touching: Near-Gathering for Autonomous Mobile Robots

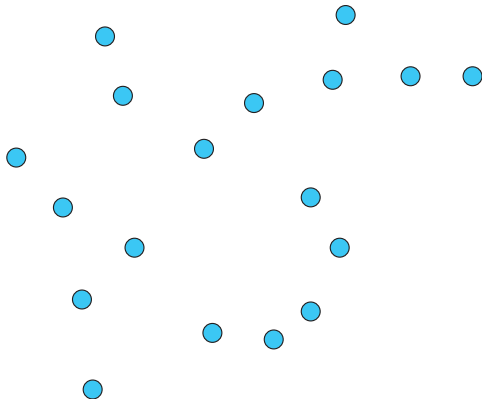
Distributed Computing, 28(5):333–349, 2015

**Strategy:** Move up-right or up-left without going too far and without entering the “move space” of other robots.

**Requirement:** All robots must agree on the up direction.

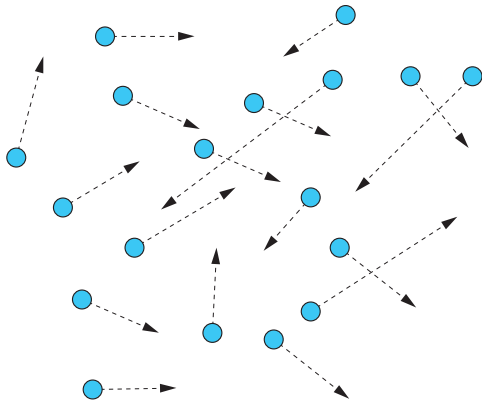


# Pattern Formation problem



**Pattern Formation:** form a pattern (given as input)  
from any initial configuration

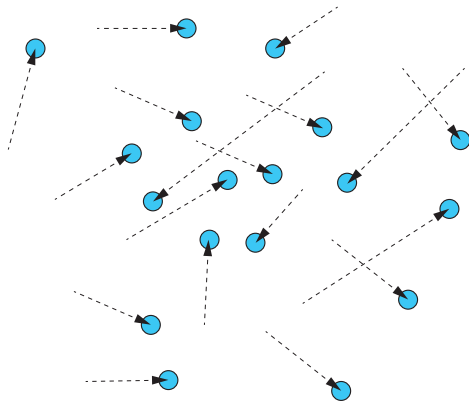
# Pattern Formation problem



**Pattern Formation:** form a pattern (given as input)  
from any initial configuration

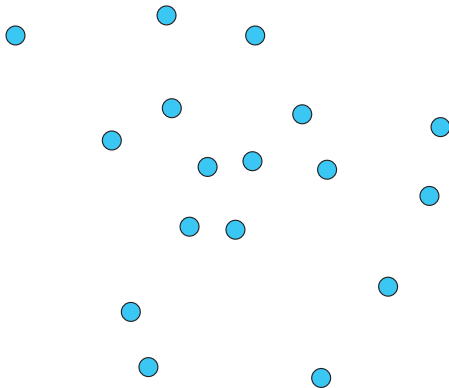


# Pattern Formation problem



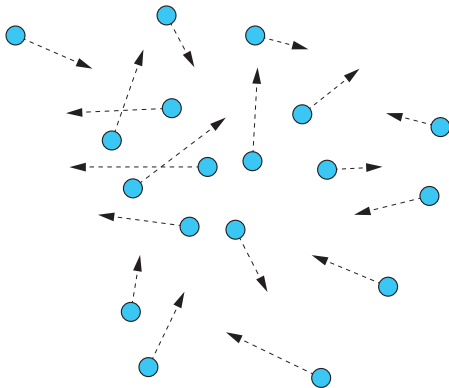
**Pattern Formation:** form a pattern (given as input)  
from any initial configuration

## Pattern Formation problem



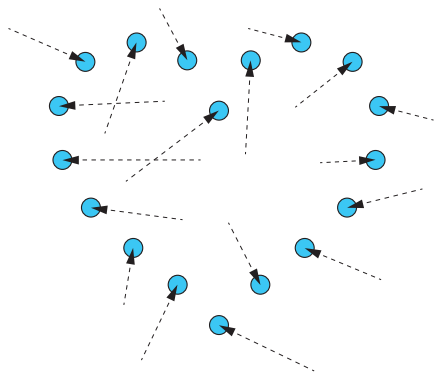
**Pattern Formation:** form a pattern (given as input)  
from any initial configuration

# Pattern Formation problem



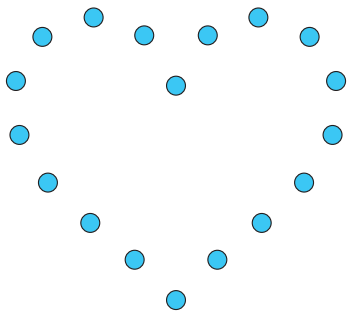
**Pattern Formation:** form a pattern (given as input)  
from any initial configuration

# Pattern Formation problem



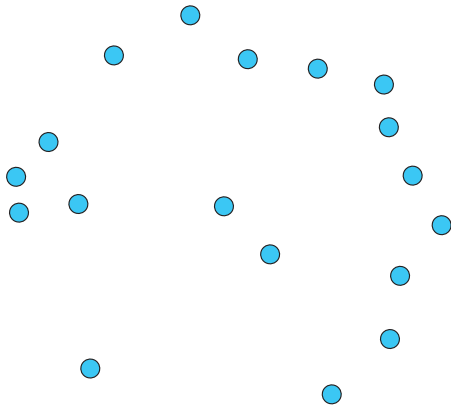
**Pattern Formation:** form a pattern (given as input)  
from any initial configuration

# Pattern Formation problem



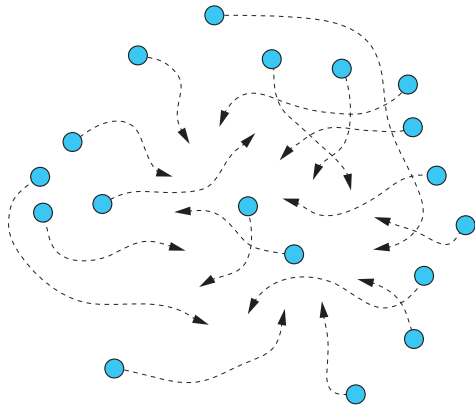
**Pattern Formation:** form a pattern (given as input)  
from any initial configuration

# Pattern Formation problem



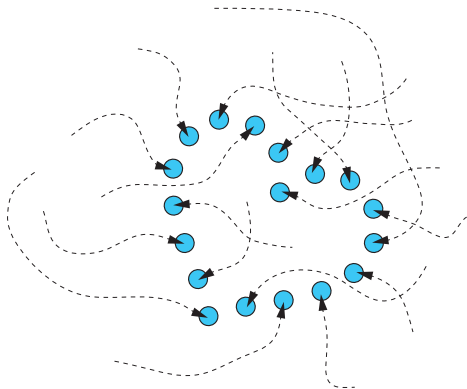
The pattern may be rotated, reflected, and scaled

# Pattern Formation problem



The pattern may be rotated, reflected, and scaled

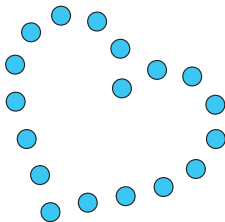
# Pattern Formation problem



The pattern may be rotated, reflected, and scaled

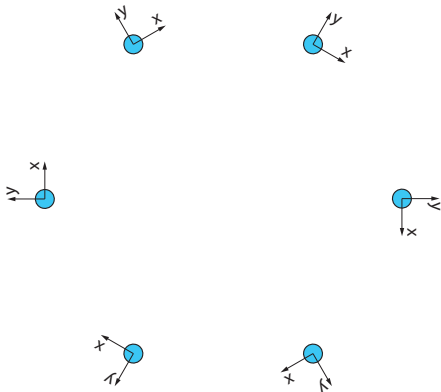


# Pattern Formation problem



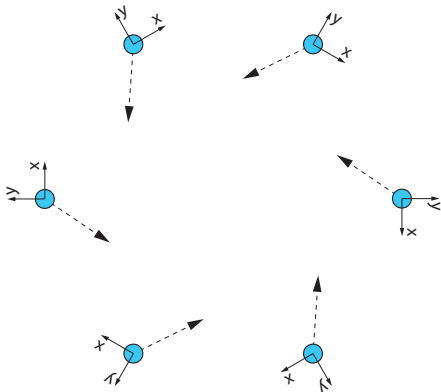
The pattern may be rotated, reflected, and scaled

# Pattern Formation problem



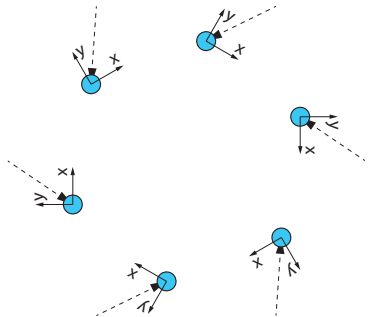
Let the initial configuration be rotationally symmetric

# Pattern Formation problem



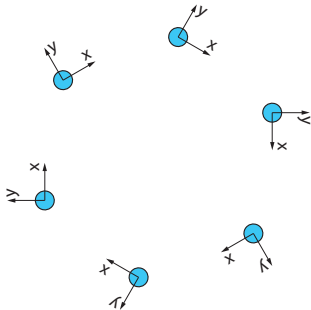
All robots have the same view and compute symmetric destinations

# Pattern Formation problem



If they are all activated synchronously, they remain symmetric

# Pattern Formation problem



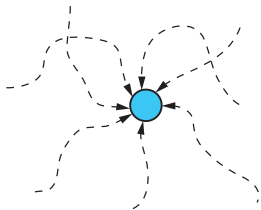
Hence Pattern Formation is unsolvable if the pattern is asymmetric

# Pattern Formation problem

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

- **Single point** (aka Gathering problem)

⇒ Solved (Cieliebak-Flocchini-Prencipe-Santoro, 2012)

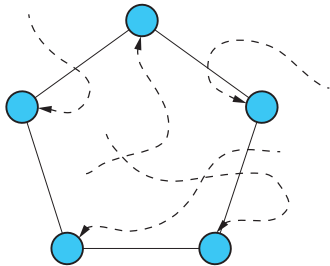
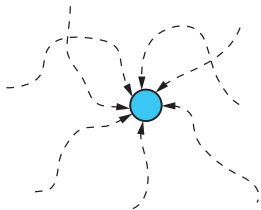


# Pattern Formation problem

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

- **Single point** (aka Gathering problem)

⇒ Solved (Cieliebak-Flocchini-Prencipe-Santoro, 2012)



- **Regular polygon** (aka Uniform Circle Formation problem)

⇒ Today's seminar

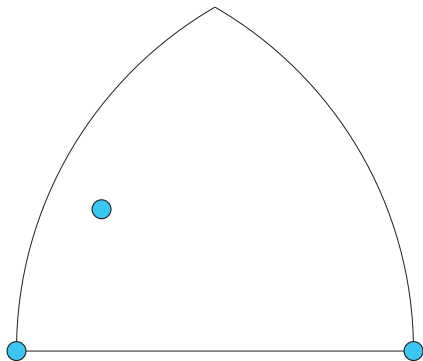
## Uniform Circle Formation for 3 robots



Suppose the triangle formed by the robots is scalene

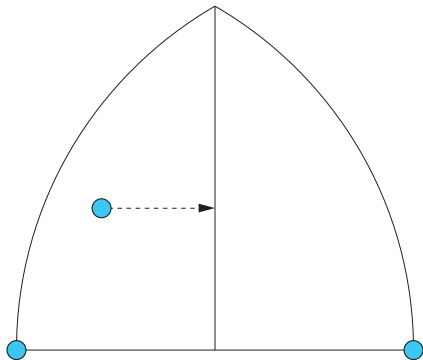


## Uniform Circle Formation for 3 robots



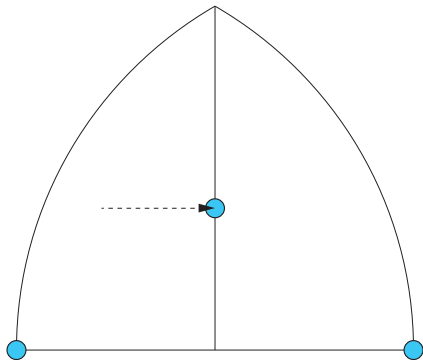
Identify the longest edge

## Uniform Circle Formation for 3 robots



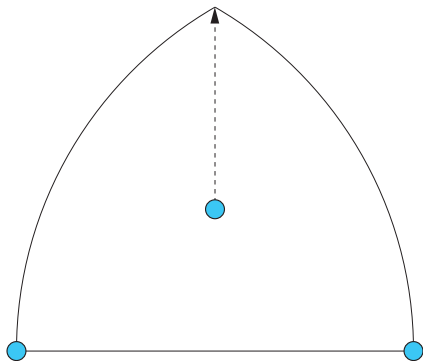
Move the third vertex parallel to the longest edge...

## Uniform Circle Formation for 3 robots



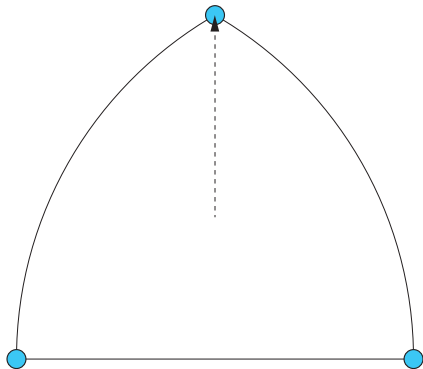
...until the triangle is isosceles

## Uniform Circle Formation for 3 robots



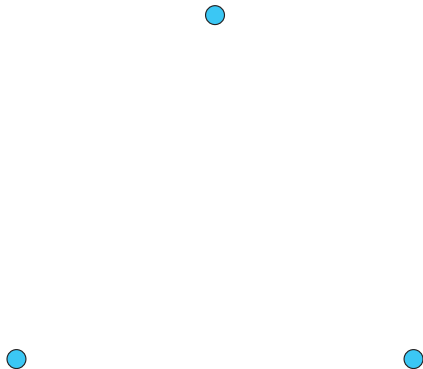
Then move the apex to the final position

## Uniform Circle Formation for 3 robots



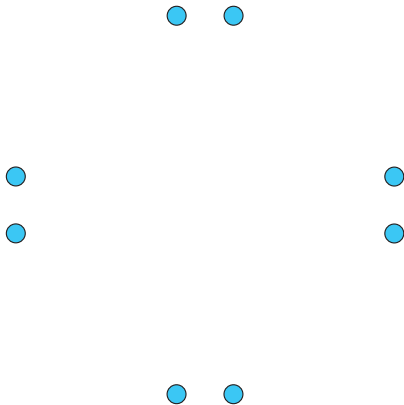
Then move the apex to the final position

## Uniform Circle Formation for 3 robots



**Correctness:** only one robot is ever allowed to move

## Resolving Biangular configurations



In a **Biangular** configuration all robots may have the same view

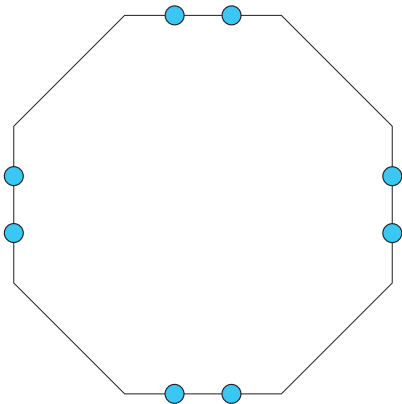
# Resolving Biangular configurations



In a **Biangular** configuration all robots may have the same view

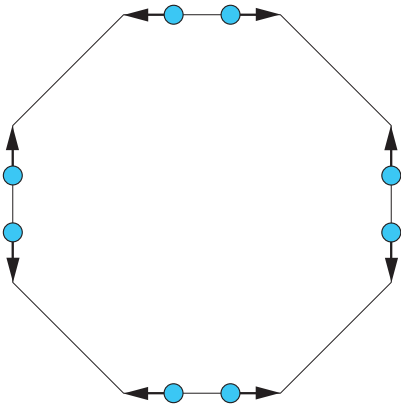


## Resolving Biangular configurations



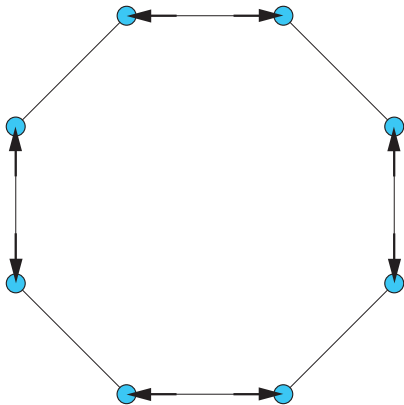
**Solution:** identify a “supporting polygon”

## Resolving Biangular configurations



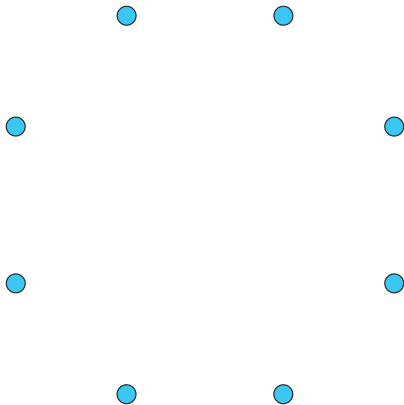
Each robot moves to the closest vertex

## Resolving Biangular configurations



During the motion, the supporting polygon remains fixed (if  $n > 4$ )

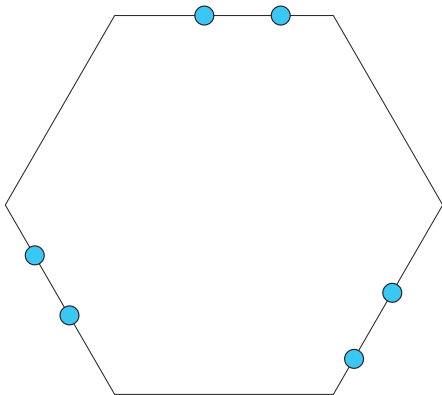
## Resolving Biangular configurations



During the motion, the supporting polygon remains fixed (if  $n > 4$ )

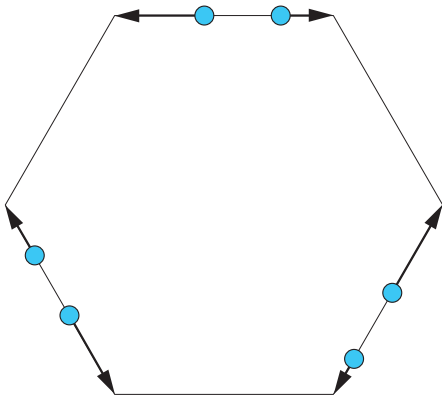
# Pre-regular configurations

The possible (asynchronous) evolutions of a Biangular configuration are called **Pre-regular** configurations



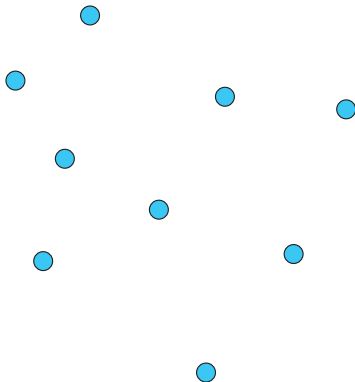
# Pre-regular configurations

The possible (asynchronous) evolutions of a Biangular configuration are called **Pre-regular** configurations



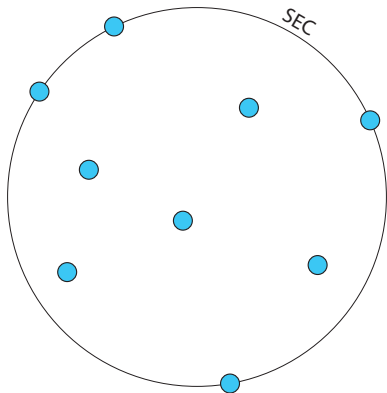
For consistency, they must be resolved in the same fashion

## Tool: Smallest Enclosing Circle (SEC)



There always exists a unique **SEC**, which is easily computable

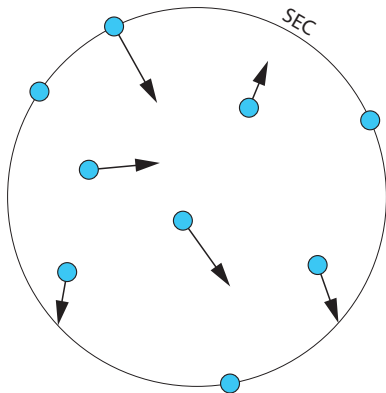
## Tool: Smallest Enclosing Circle (SEC)



There always exists a unique **SEC**, which is easily computable

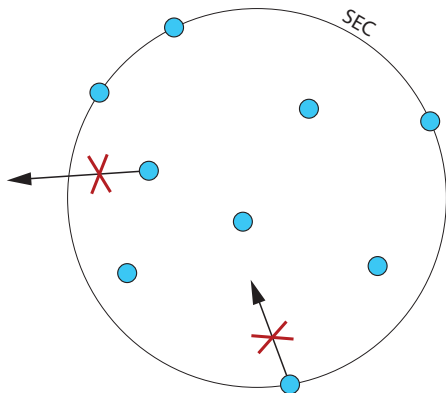


## Tool: Smallest Enclosing Circle (SEC)



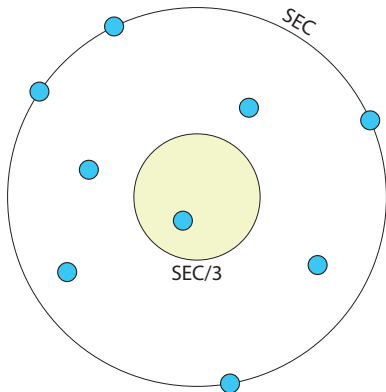
**Strategy:** always preserve the SEC, until a Pre-regular is formed

## Tool: Smallest Enclosing Circle (SEC)



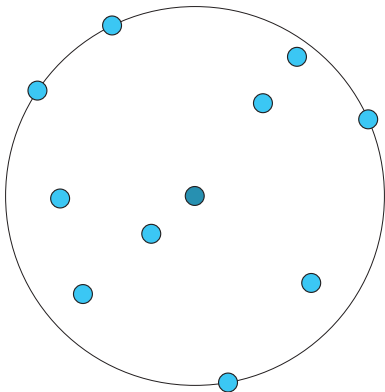
**Strategy:** always preserve the SEC, until a Pre-regular is formed

## Tool: Smallest Enclosing Circle (SEC)



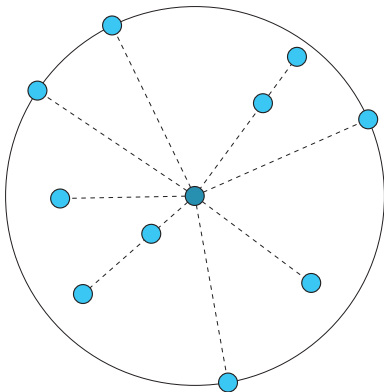
Another circle plays an important role: **SEC/3**

## Central configurations



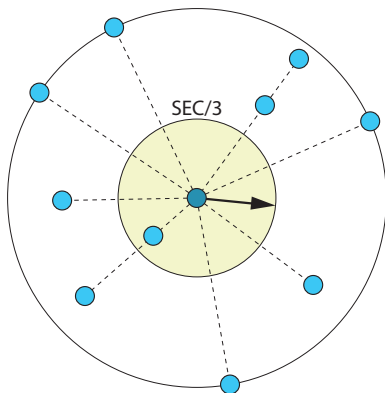
If there is a robot at the center of the SEC

# Central configurations



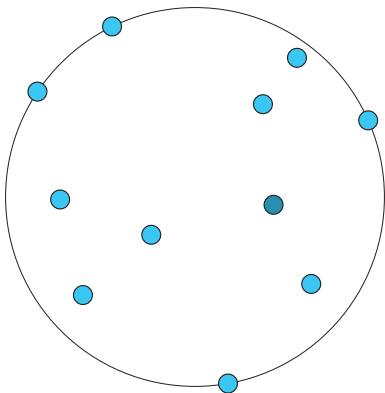
Identify the occupied radii

# Central configurations



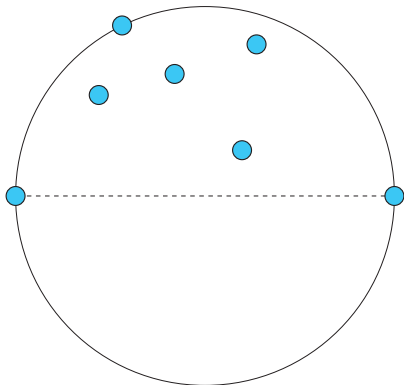
Move to an unoccupied radius all the way to SEC/3

# Central configurations



No Central configuration will ever be formed again

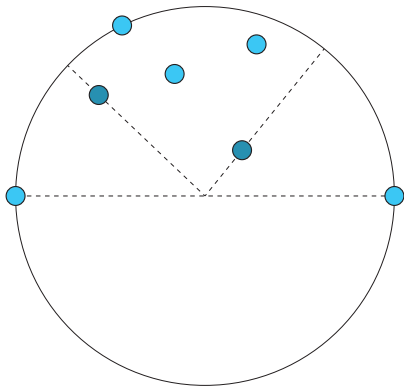
## Half-disk configurations



If all the robots lie in one half of the SEC

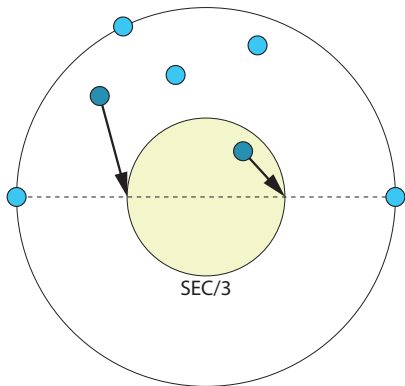


## Half-disk configurations



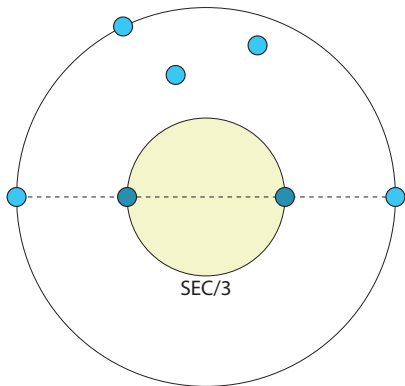
Identify the robots “angularly closest” to the diameter

# Half-disk configurations



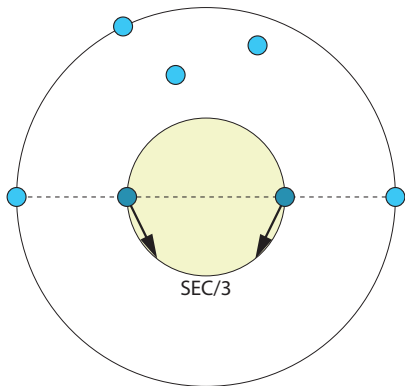
Move them to the diameter and make sure they are in SEC/3

# Half-disk configurations



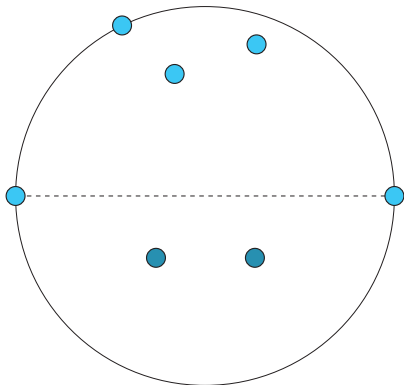
Move them to the diameter and make sure they are in SEC/3

# Half-disk configurations



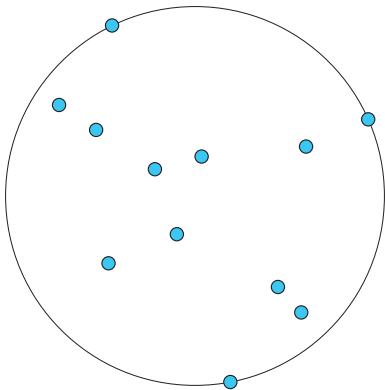
Make them cross the diameter but remain in SEC/3

## Half-disk configurations



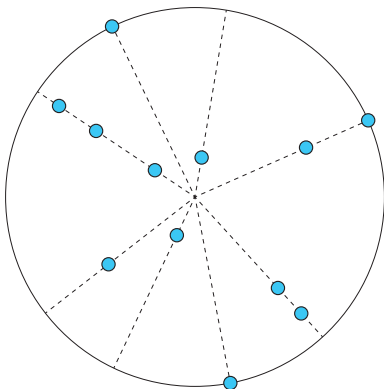
No half-disk configuration will ever be formed again

## Co-radial configurations



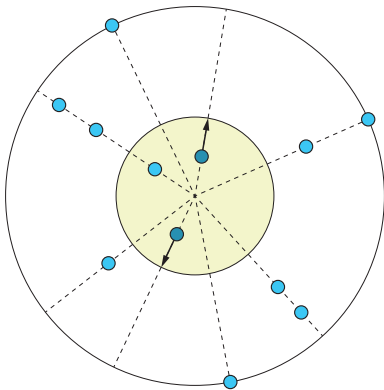
If there are co-radial robots

# Co-radial configurations



Identify the occupied radii

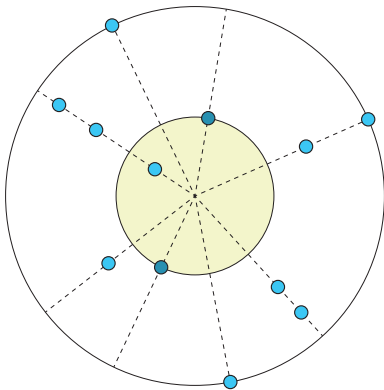
# Co-radial configurations



Move radially to  $SEC/3$  the non-co-radial robots that lie inside it

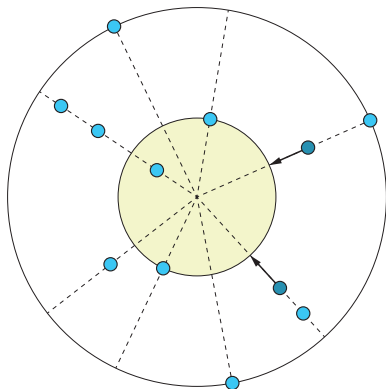


# Co-radial configurations



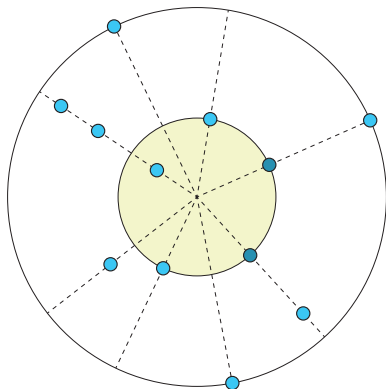
Move radially to  $SEC/3$  the non-co-radial robots that lie inside it

# Co-radial configurations



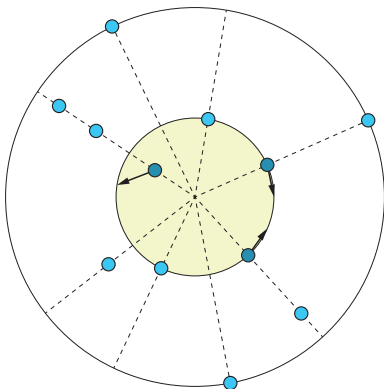
Move to  $SEC/3$  the innermost co-radial robots that lie outside

# Co-radial configurations



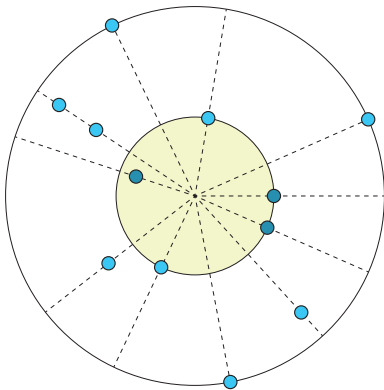
Move to  $SEC/3$  the innermost co-radial robots that lie outside

# Co-radial configurations



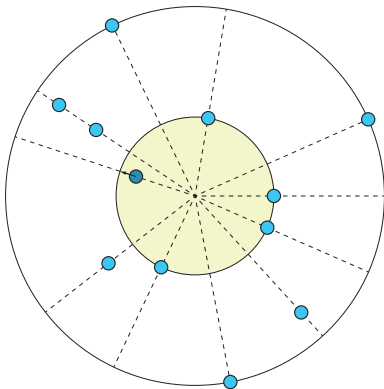
Move the innermost co-radial robots laterally by a small angle

# Co-radial configurations



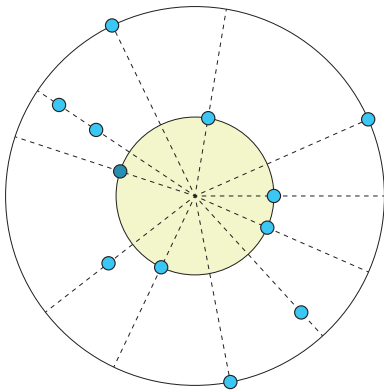
If moves are small enough, no new co-radialities are formed

# Co-radial configurations



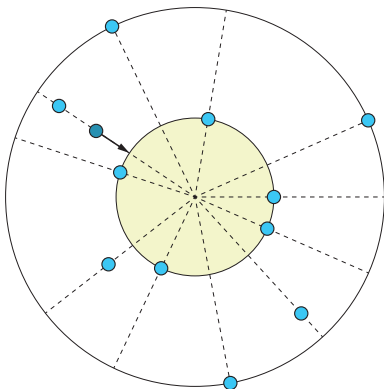
Move radially to  $SEC/3$  the non-co-radial robots that lie inside it

# Co-radial configurations



Move radially to  $SEC/3$  the non-co-radial robots that lie inside it

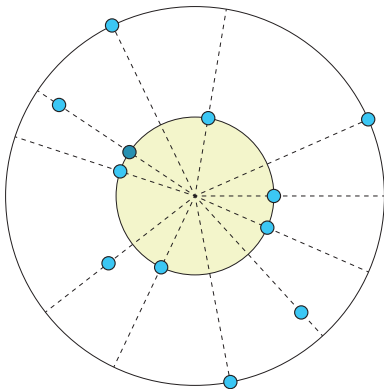
# Co-radial configurations



Move to  $SEC/3$  the innermost co-radial robots that lie outside

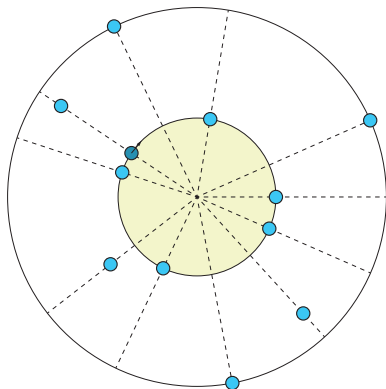


# Co-radial configurations



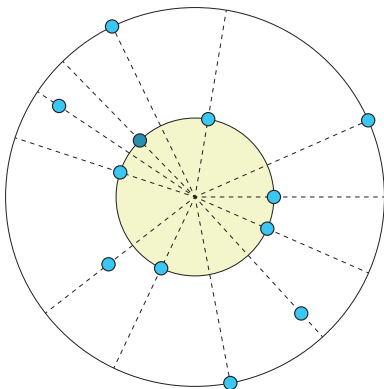
Move to  $SEC/3$  the innermost co-radial robots that lie outside

# Co-radial configurations



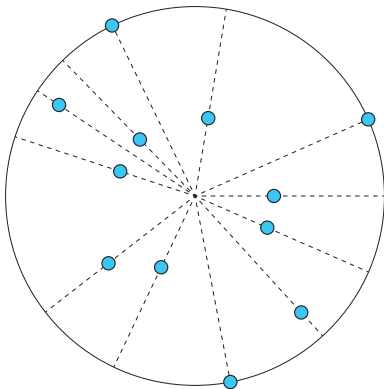
Move the innermost co-radial robots laterally by a small angle

# Co-radial configurations



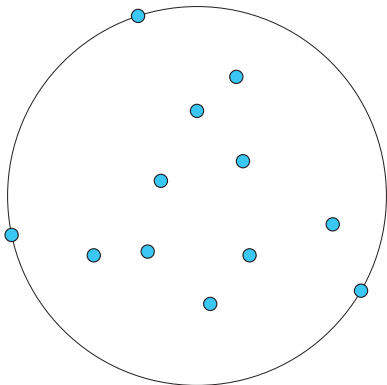
Move the innermost co-radial robots laterally by a small angle

## Co-radial configurations



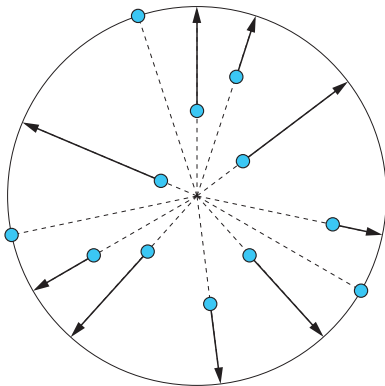
No Co-radial configuration will ever be formed again

## General case: overview



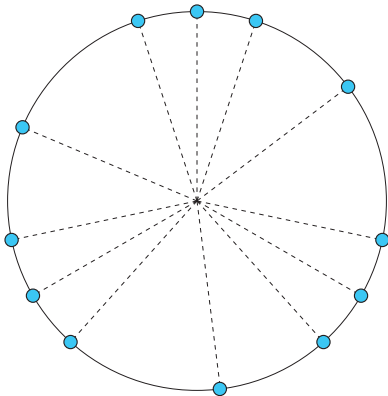
If we are not in one of the previous special cases

## General case: overview



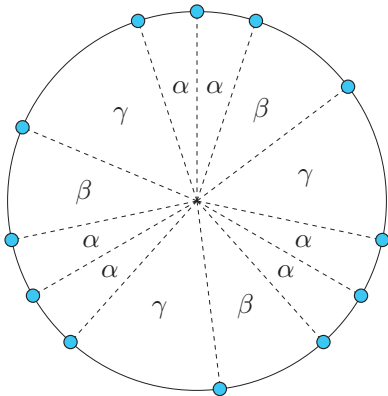
All robots move radially to SEC

## General case: overview



All robots move radially to SEC

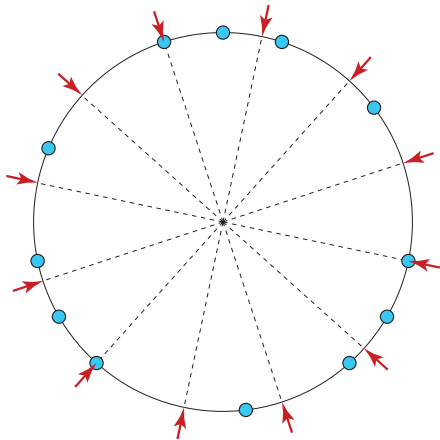
## General case: overview



Consider the angle sequence

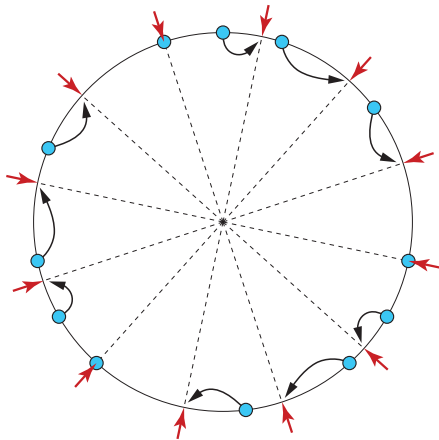


## General case: overview



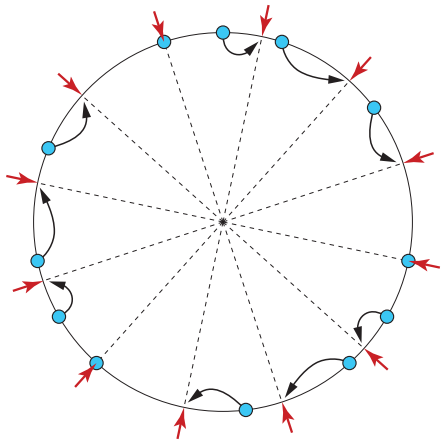
To solve Uniform Circle Formation, all angles must become equal

## General case: overview



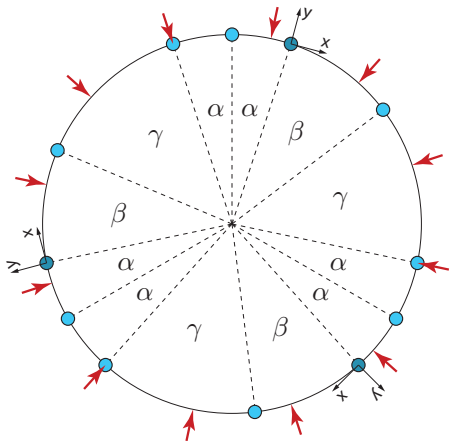
Identify a **target** for each robot

## General case: overview



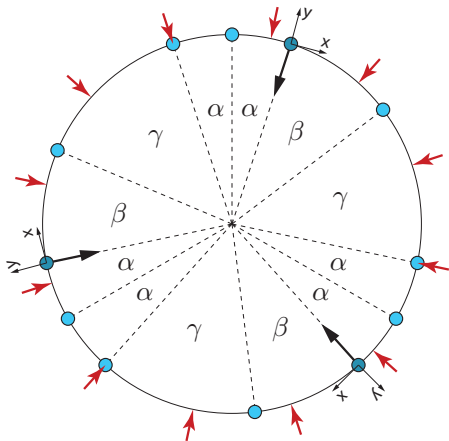
If all robots move together, they may forget their targets!

## General case: overview



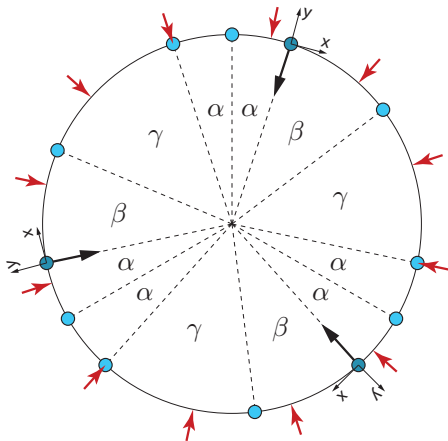
Robots that “see” the same angle sequence are **analogous**

## General case: overview



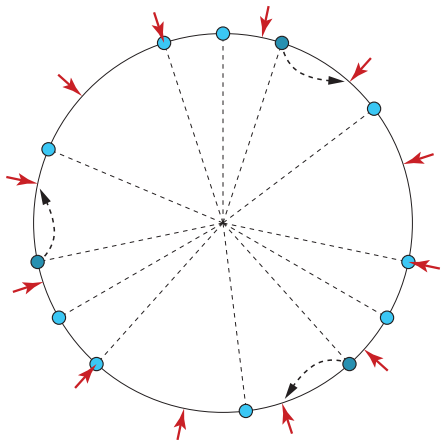
The scheduler can force analogous robots to move together

## General case: overview



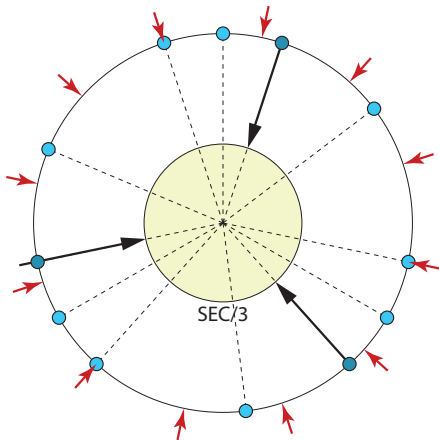
Hence the algorithm will move one **analogy class** at a time

## General case: overview



**Strategy:** choose analogous robots that can “see” their targets

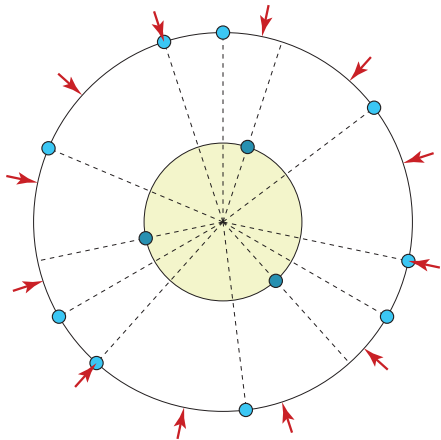
## General case: overview



Move these **walkers** radially to SEC/3

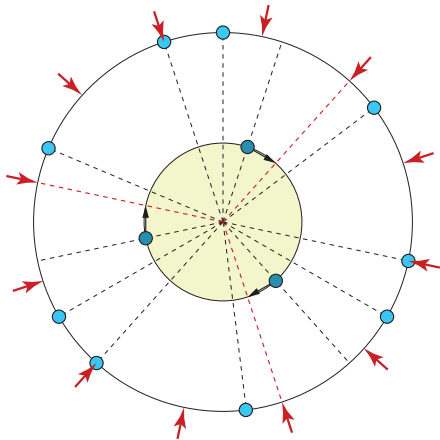


## General case: overview



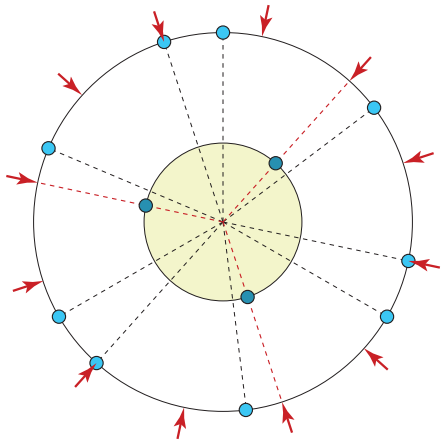
Move these **walkers** radially to SEC/3

## General case: overview



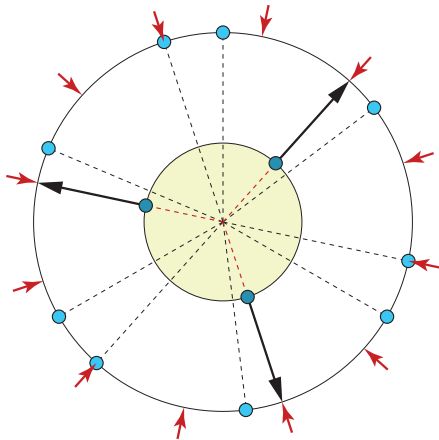
Move them laterally to their target radii

## General case: overview



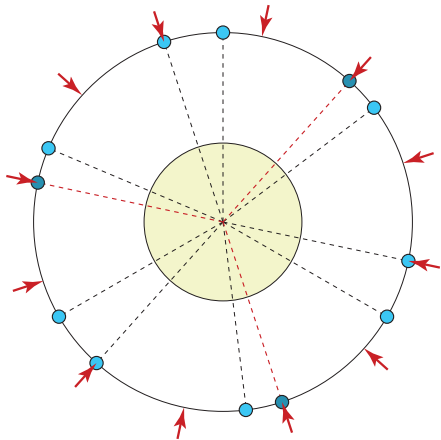
Move them laterally to their target radii

## General case: overview



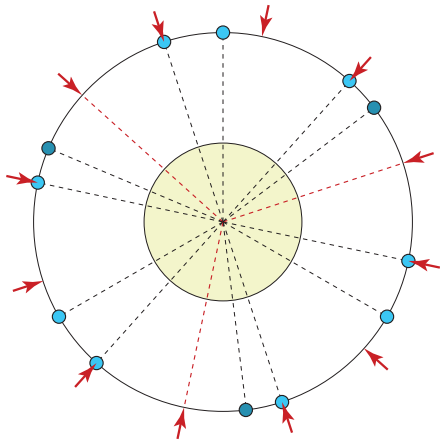
Move them back to SEC

## General case: overview



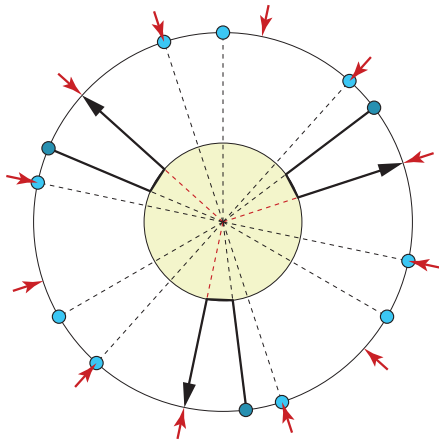
Move them back to SEC

## General case: overview



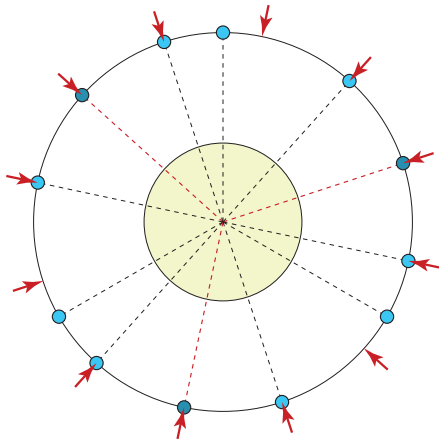
Repeat with another analogy class, until all targets are reached

## General case: overview



Repeat with another analogy class, until all targets are reached

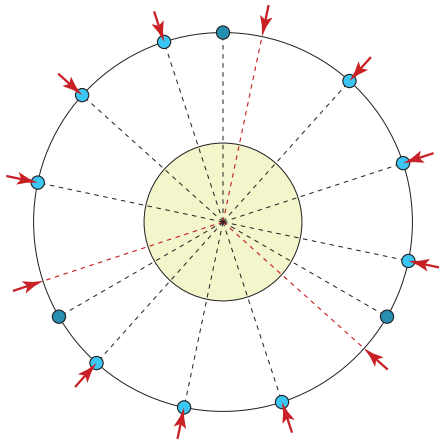
## General case: overview



Repeat with another analogy class, until all targets are reached

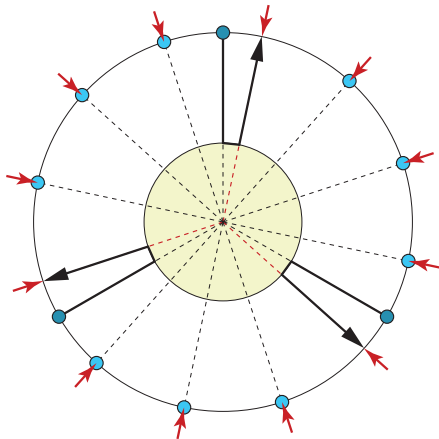


## General case: overview



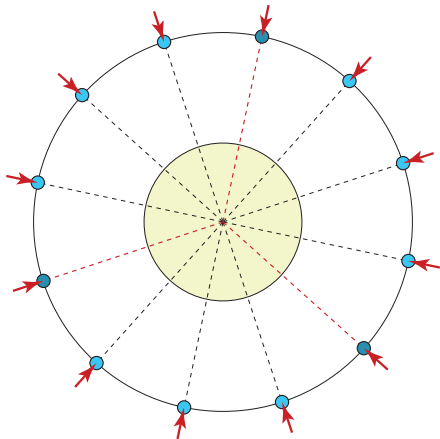
Repeat with another analogy class, until all targets are reached

## General case: overview



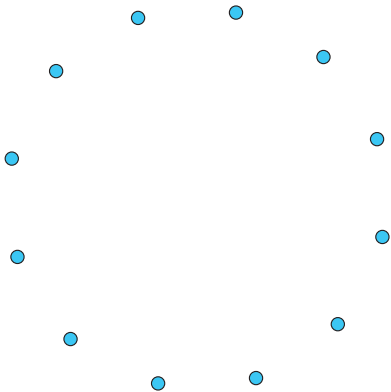
Repeat with another analogy class, until all targets are reached

## General case: overview



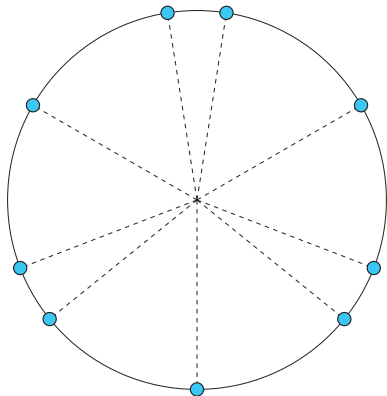
Repeat with another analogy class, until all targets are reached

## General case: overview



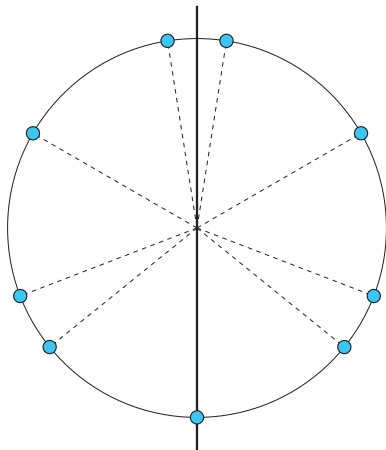
Repeat with another analogy class, until all targets are reached

## How to identify the target set



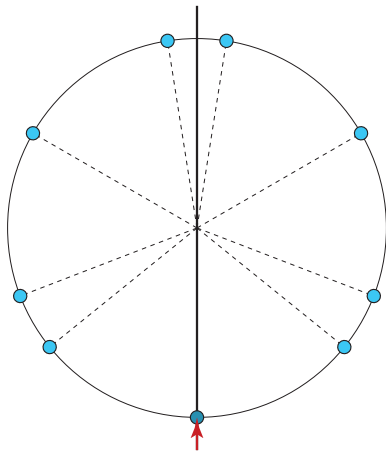
Suppose the configuration has an axis of symmetry

## How to identify the target set



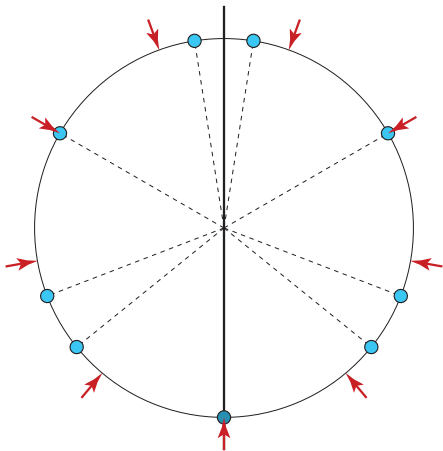
Suppose the configuration has an axis of symmetry

## How to identify the target set



If a robot lies on the axis, it is on its target by definition

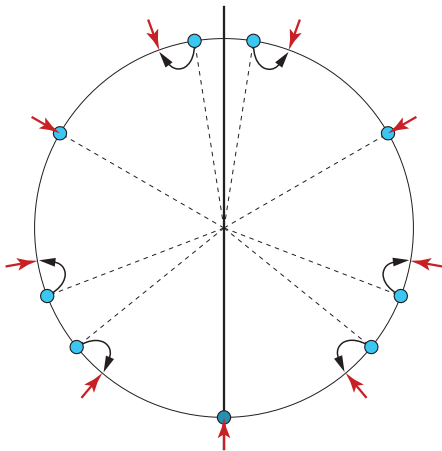
## How to identify the target set



The other targets are determined accordingly

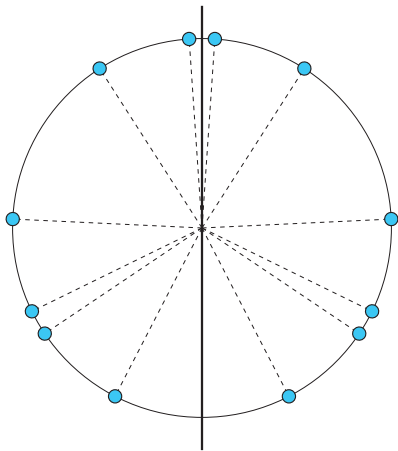


## How to identify the target set



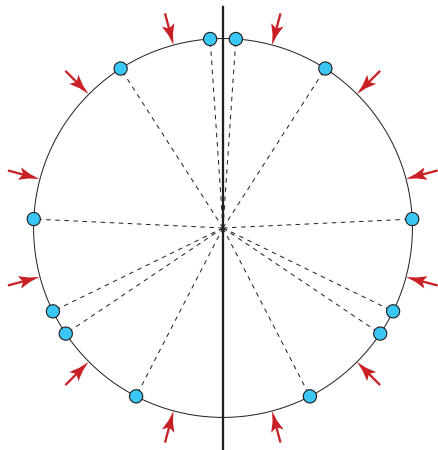
The other targets are determined accordingly

## How to identify the target set



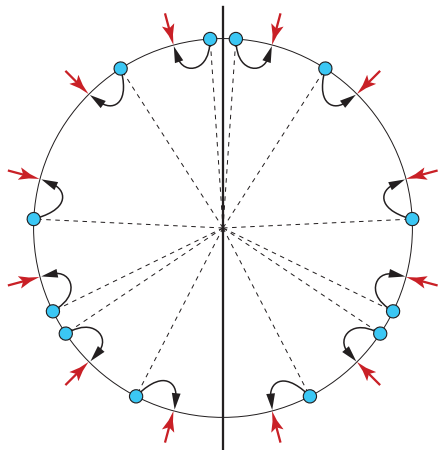
Suppose that no robot lies on the axis of symmetry

## How to identify the target set



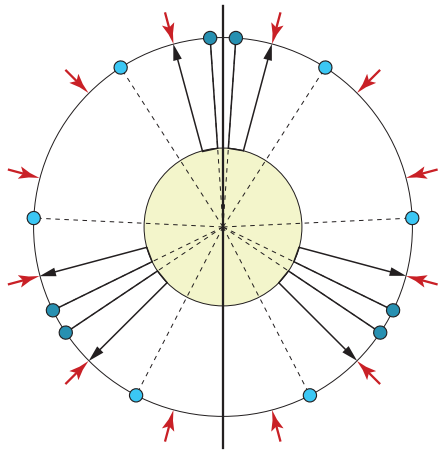
Then no target lies on the axis of symmetry, either

## How to identify the target set



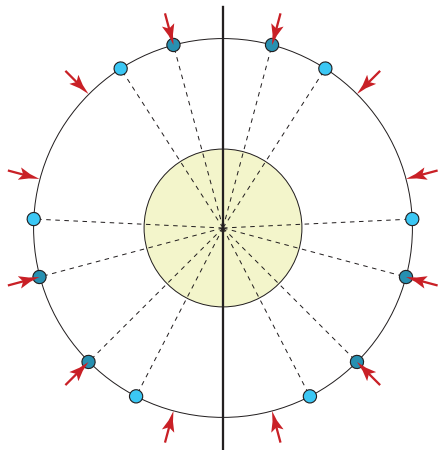
Then no target lies on the axis of symmetry, either

## How to identify the target set



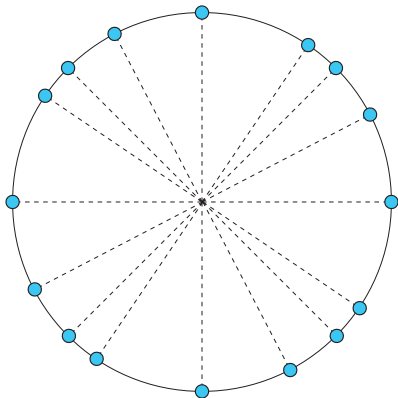
When an analogy class moves, the axis of symmetry is preserved

## How to identify the target set



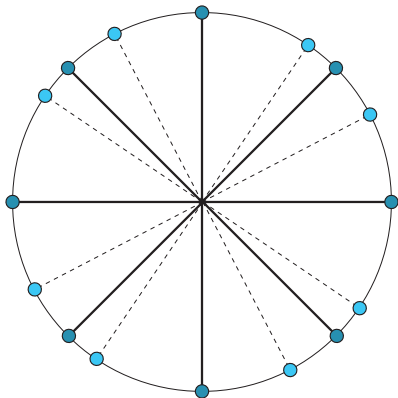
Hence also the targets are preserved

## How to identify the target set



Suppose the configuration has no axis of symmetry

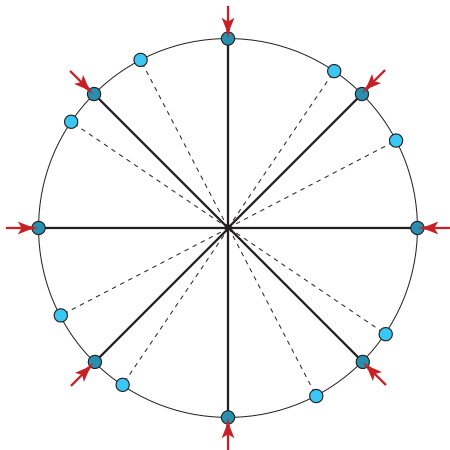
## How to identify the target set



Robots having the “correct” angular distance are **concordant**

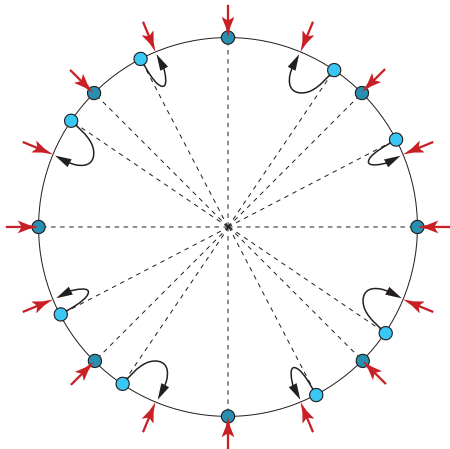


## How to identify the target set



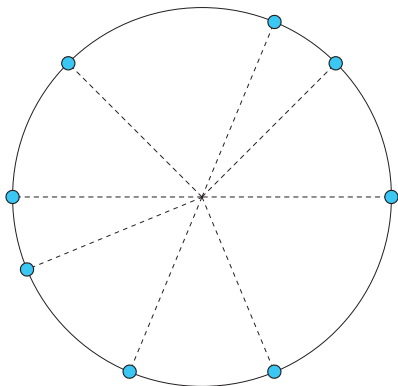
The targets are determined by the largest **concordance class**

# How to identify the target set



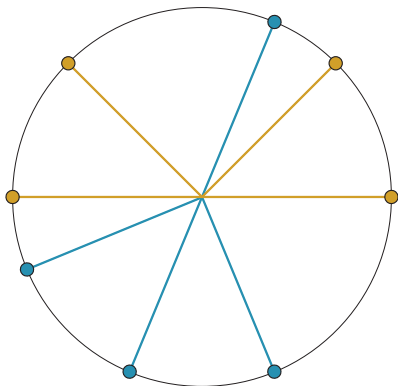
The targets are determined by the largest **concordance class**

## How to identify the target set



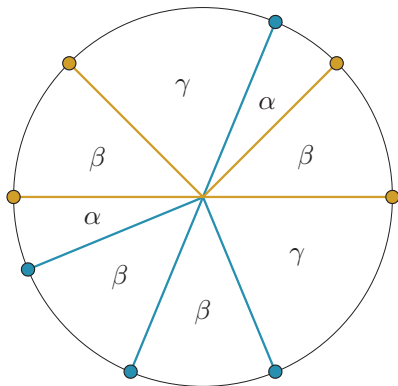
If there is more than one largest concordance class...

## How to identify the target set



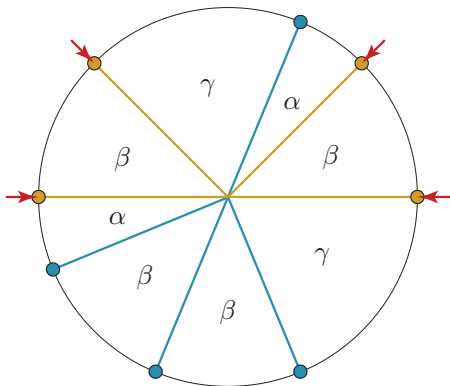
If there is more than one largest concordance class...

## How to identify the target set



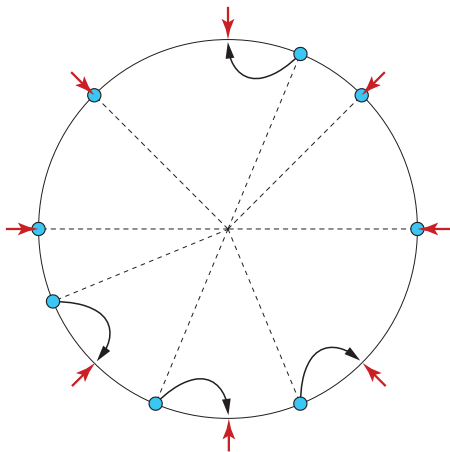
...they are all “non-equivalent”, so one can always be chosen

## How to identify the target set



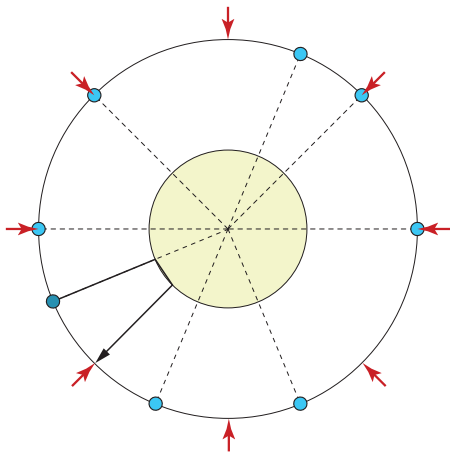
...they are all “non-equivalent”, so one can always be chosen

## How to identify the target set



...they are all “non-equivalent”, so one can always be chosen

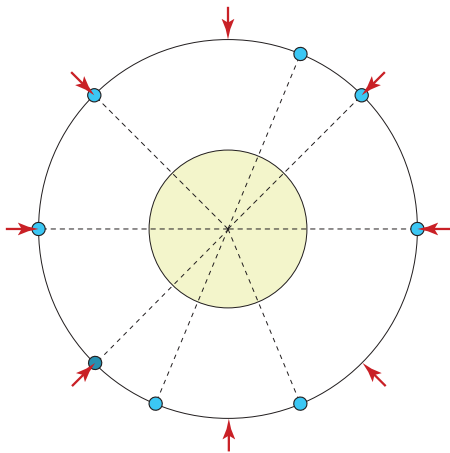
## How to identify the target set



After a move, the chosen concordance class becomes the largest

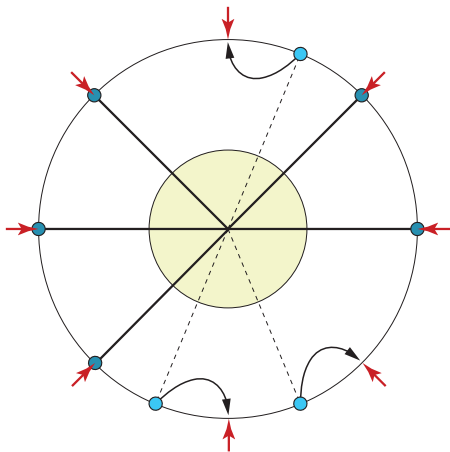


## How to identify the target set



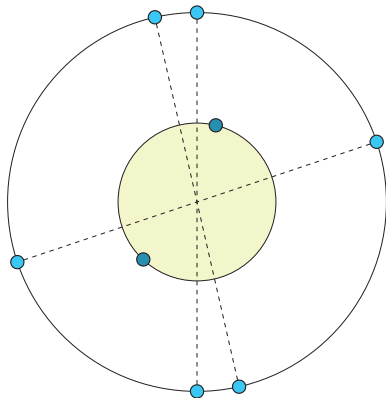
After a move, the chosen concordance class becomes the largest

## How to identify the target set



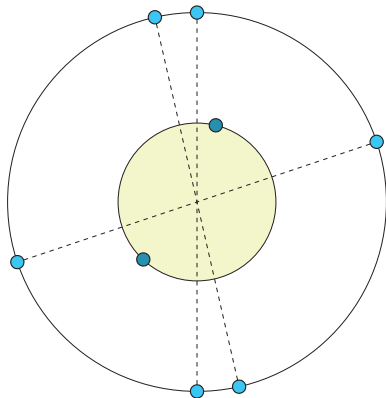
So the targets are preserved, until an axis of symmetry is created

## How the walkers “remember” their targets



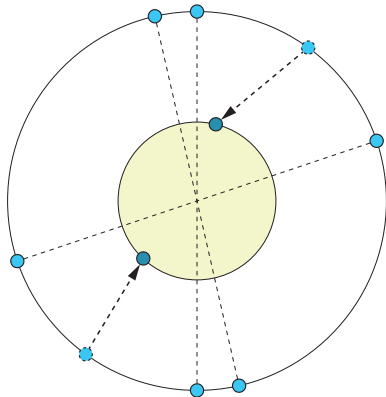
Target locations depend on the walkers' initial positions on SEC

## How the walkers “remember” their targets



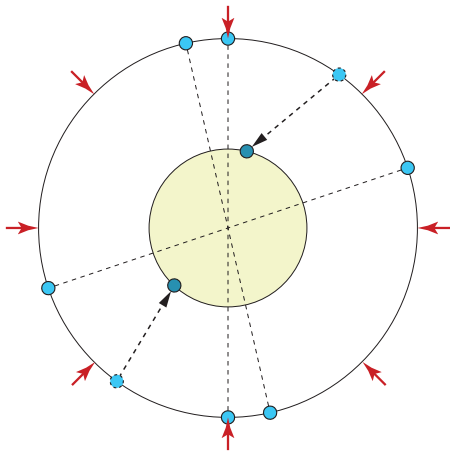
Once in SEC/3, they can only “guess” the initial positions

## How the walkers “remember” their targets



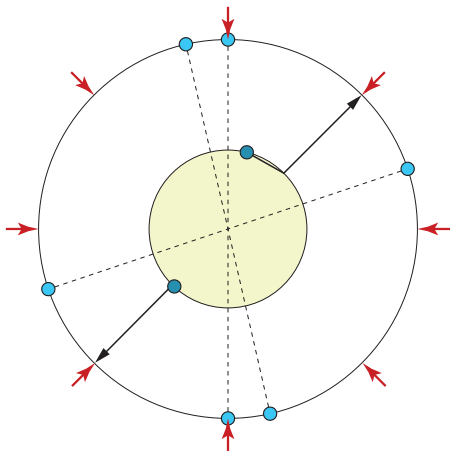
Reasonable guess: we were equidistant from our adjacent robots

# How the walkers “remember” their targets



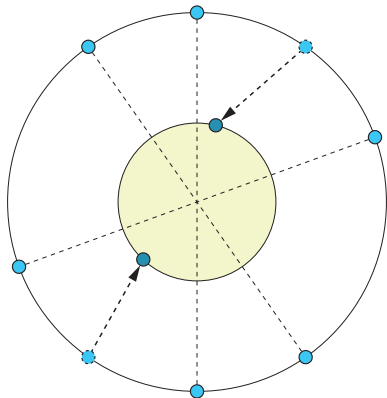
Now they can reconstruct a consistent set of targets...

# How the walkers “remember” their targets



...and move to the appropriate ones

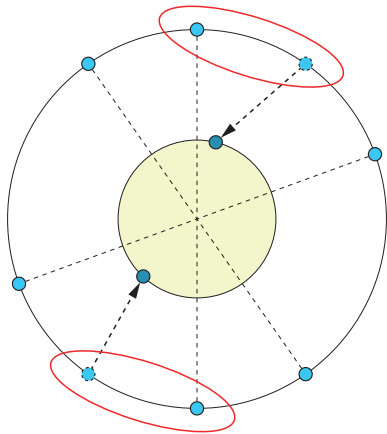
## How the walkers “remember” their targets



**Problem:** the guessed locations may not form an analogy class...

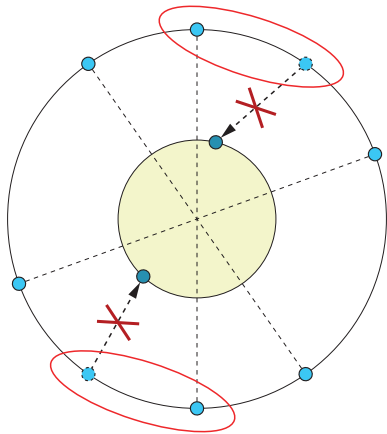


# How the walkers “remember” their targets



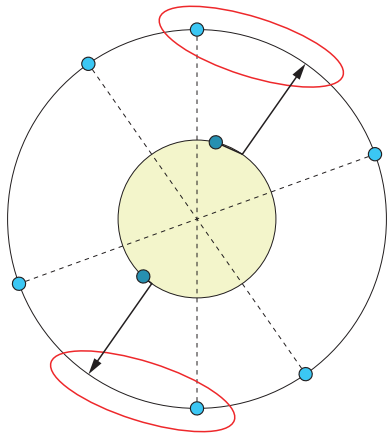
...they may be a proper subset of one!

# How the walkers “remember” their targets



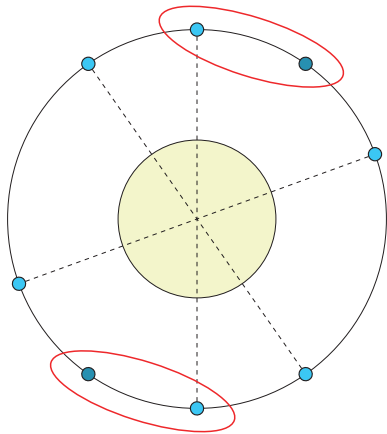
Wrong guess: the walkers are supposed to form an analogy class

# How the walkers “remember” their targets



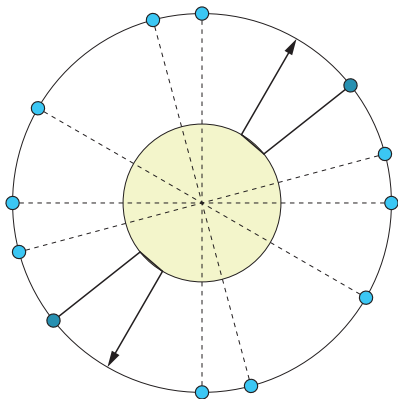
**Solution:** move to the guessed positions!

## How the walkers “remember” their targets



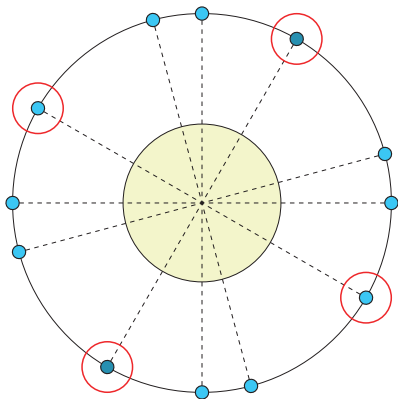
This causes two analogy classes to merge

## Making steady progress



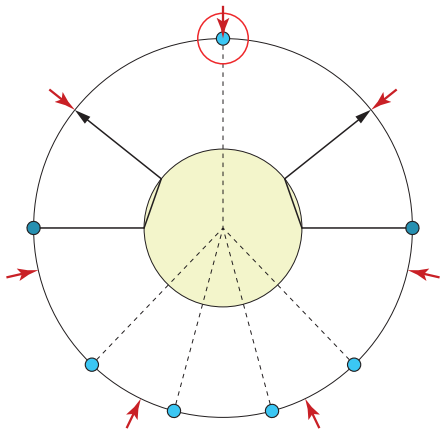
When the walkers complete their journey, two things can happen

## Making steady progress



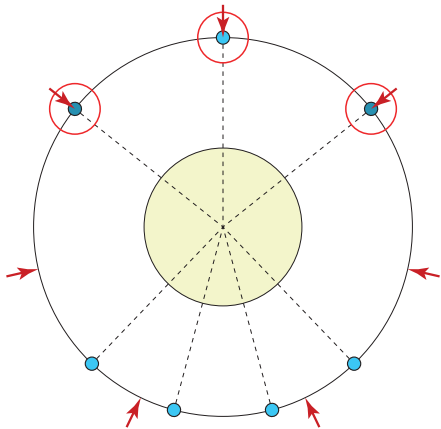
Either two analogy classes merge (and the targets may change)...

## Making steady progress



...or more robots reach their targets

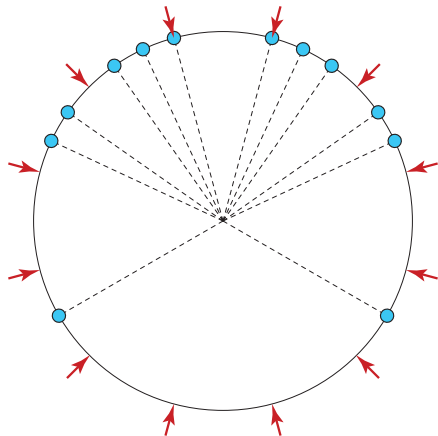
## Making steady progress



Eventually all robots will reach their targets

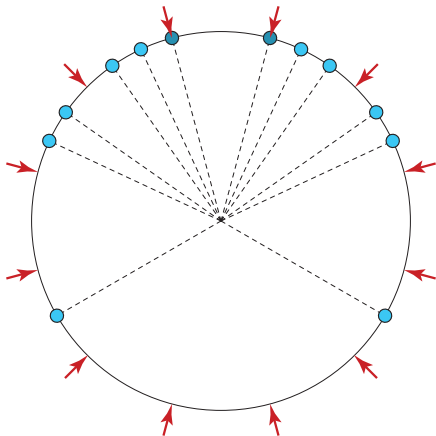


# Locked configurations



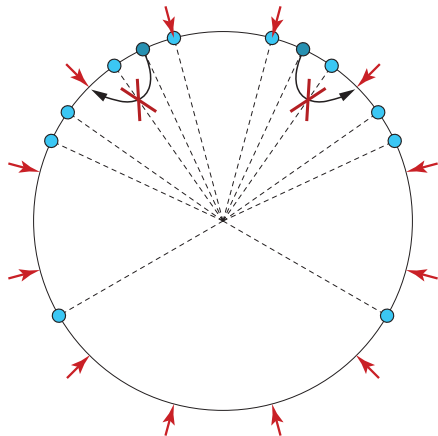
Sometimes no analogy class is able to move to reach its targets

# Locked configurations



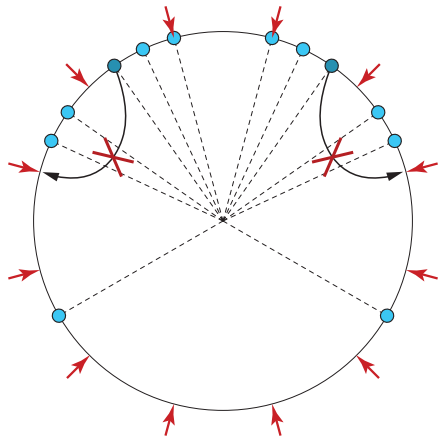
Either because it is already there...

# Locked configurations



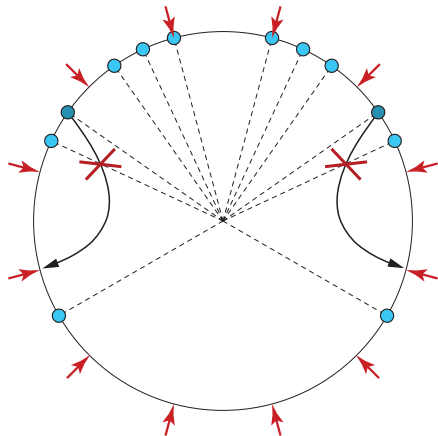
...or because it would form a Co-radial configuration...

# Locked configurations



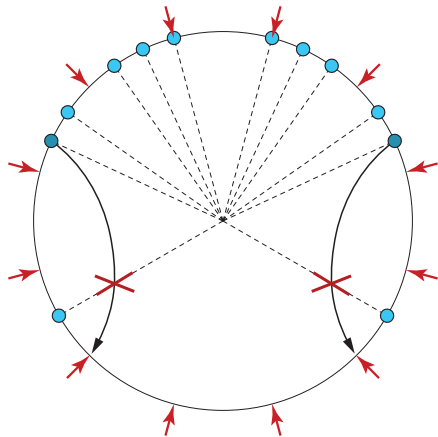
...or because it would form a Co-radial configuration...

# Locked configurations



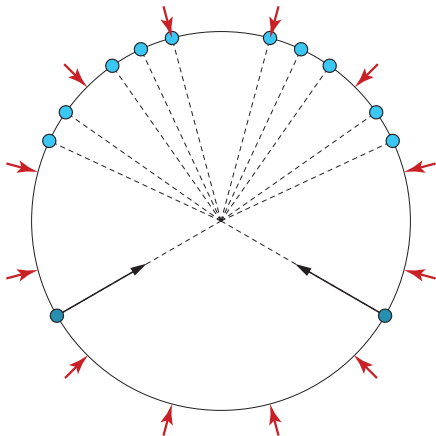
...or because it would form a Co-radial configuration...

# Locked configurations



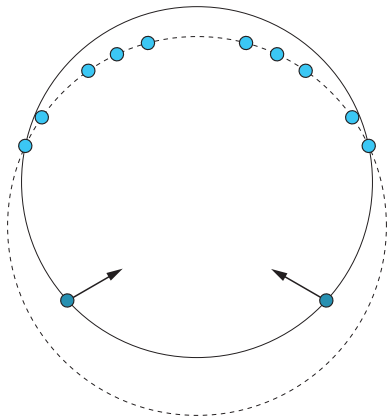
...or because it would form a Co-radial configuration...

# Locked configurations



...or because it would alter the SEC

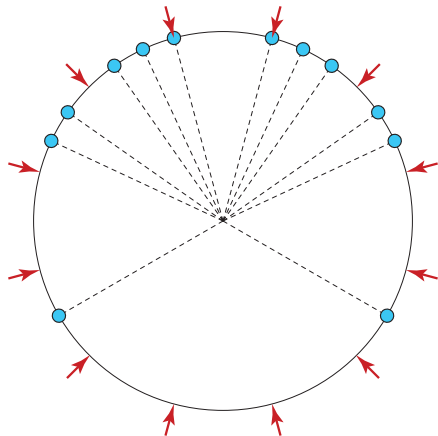
## Locked configurations



...or because it would alter the SEC

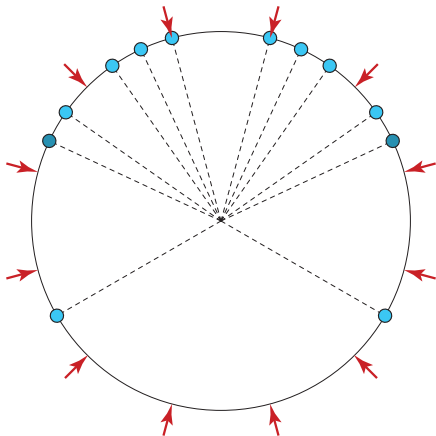


# Locked configurations



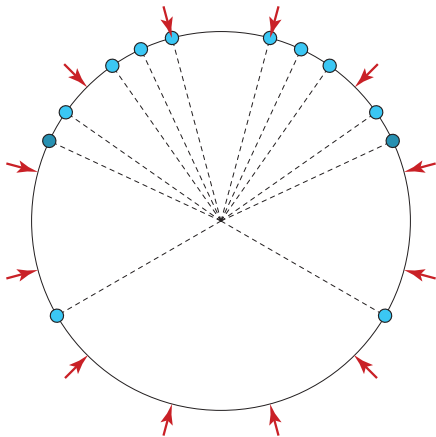
In this case the configuration is **locked**

# Locked configurations



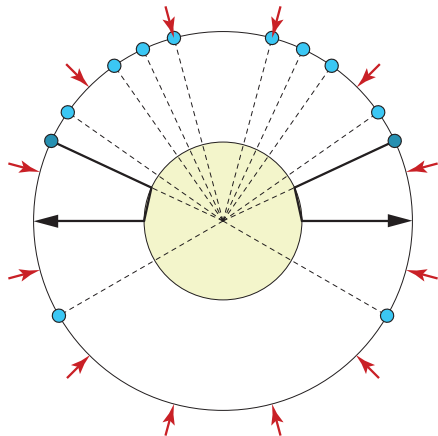
**Strategy:** identify an **unlocking** analogy class

## Locked configurations



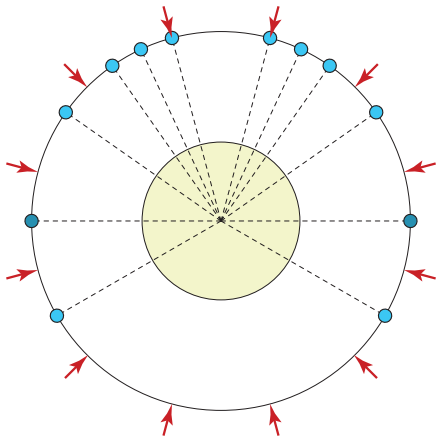
It exists because the configuration is not Half-disk!

# Locked configurations



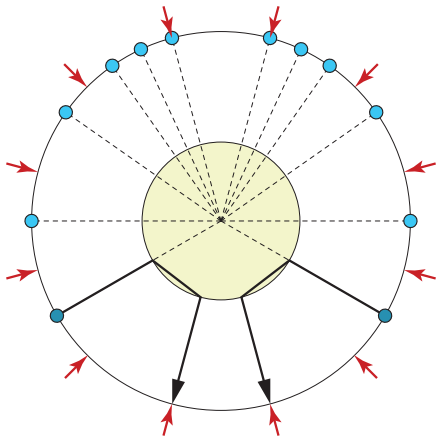
The unlocking class makes a preliminary move...

# Locked configurations



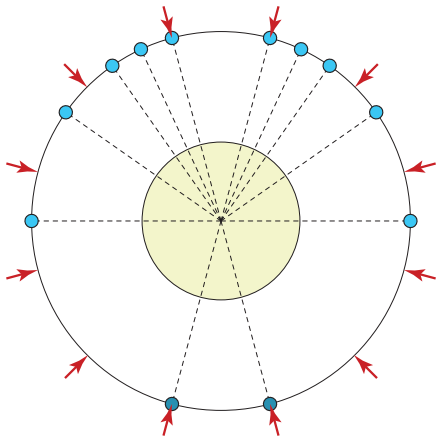
The unlocking class makes a preliminary move...

# Locked configurations



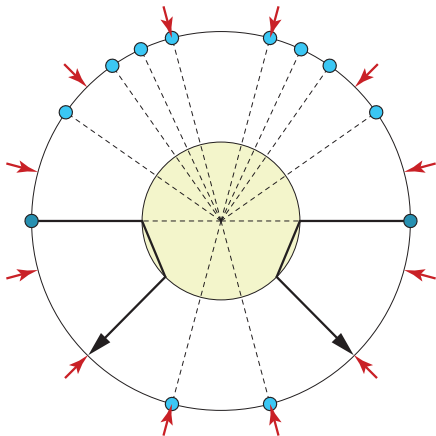
...and the previously unmovable class becomes free to move

# Locked configurations



...and the previously unmovable class becomes free to move

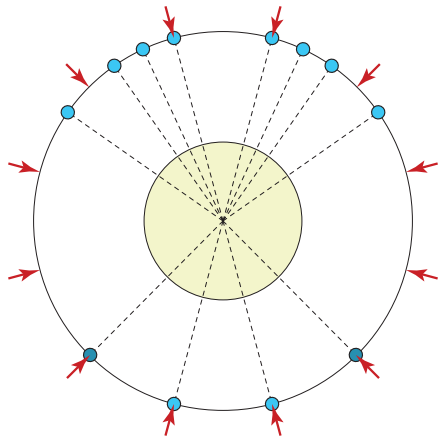
# Locked configurations



The unlocking step does not make robots lose their progress

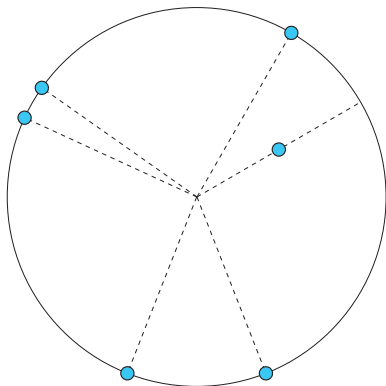


## Locked configurations



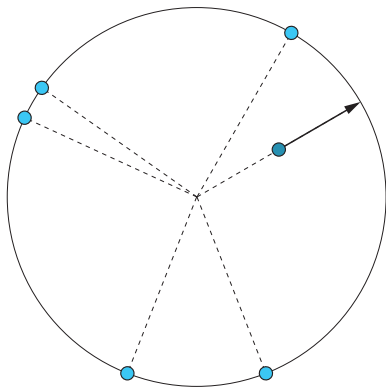
And after the unlocking step, steady progress is made again

## Accidental formation of Pre-regular configurations



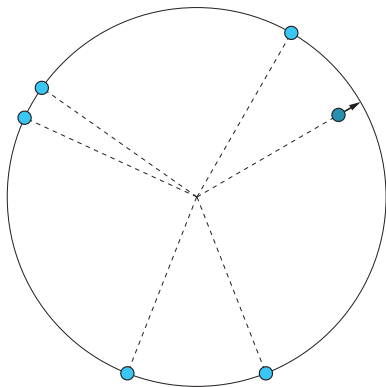
What if a Pre-regular configuration is formed “accidentally”?

# Accidental formation of Pre-regular configurations



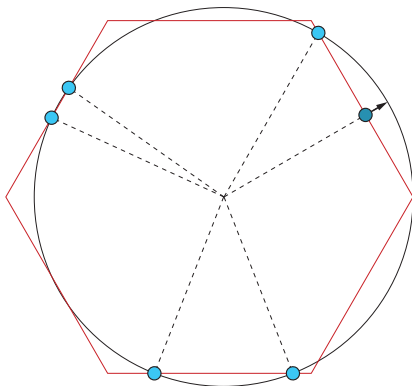
What if a Pre-regular configuration is formed “accidentally”?

## Accidental formation of Pre-regular configurations



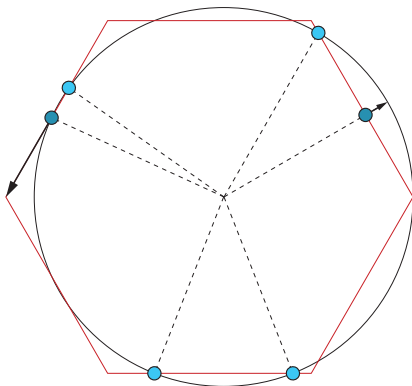
What if a Pre-regular configuration is formed “accidentally”?

## Accidental formation of Pre-regular configurations



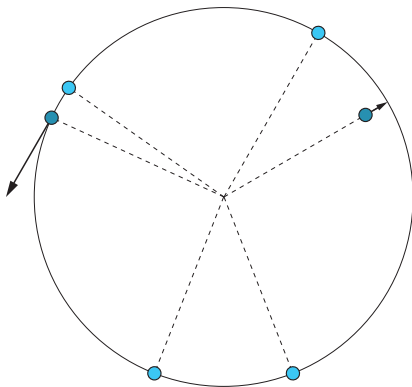
What if a Pre-regular configuration is formed “accidentally”?

## Accidental formation of Pre-regular configurations



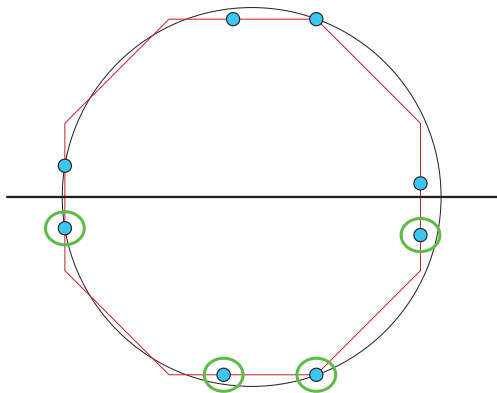
Due to asynchronicity, the behavior may be inconsistent

## Accidental formation of Pre-regular configurations



Due to asynchronicity, the behavior may be inconsistent

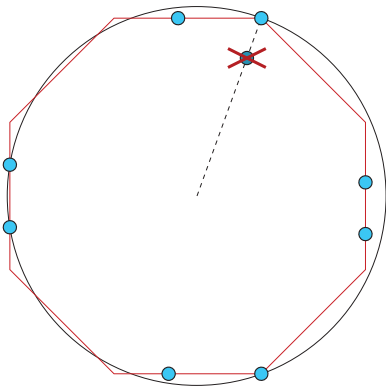
# Accidental formation of Pre-regular configurations



**Lemma:** no Pre-regular configuration is Half-disk

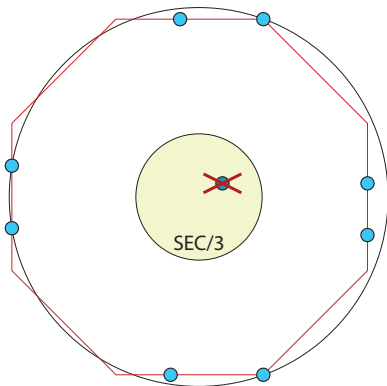


# Accidental formation of Pre-regular configurations



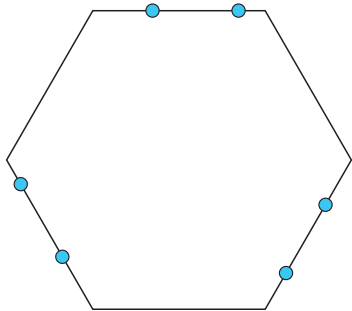
**Lemma:** no Pre-regular configuration is Co-radial

# Accidental formation of Pre-regular configurations



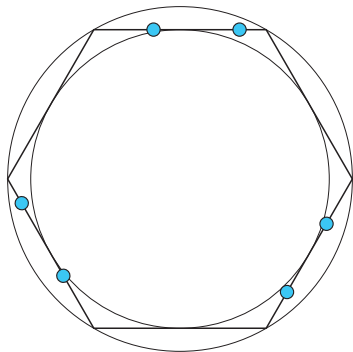
**Lemma:** no Pre-regular configuration has robots in SEC/3

# Accidental formation of Pre-regular configurations



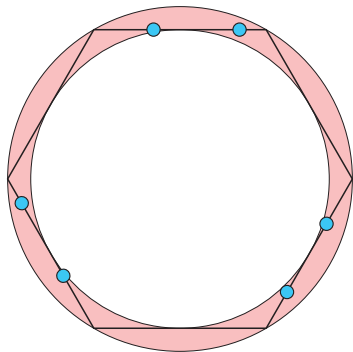
**Proof:** suppose the configuration is Pre-regular

# Accidental formation of Pre-regular configurations



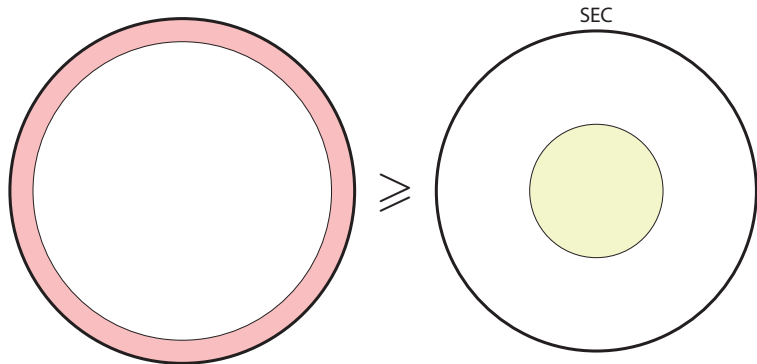
All robots lie in a thin annulus...

# Accidental formation of Pre-regular configurations



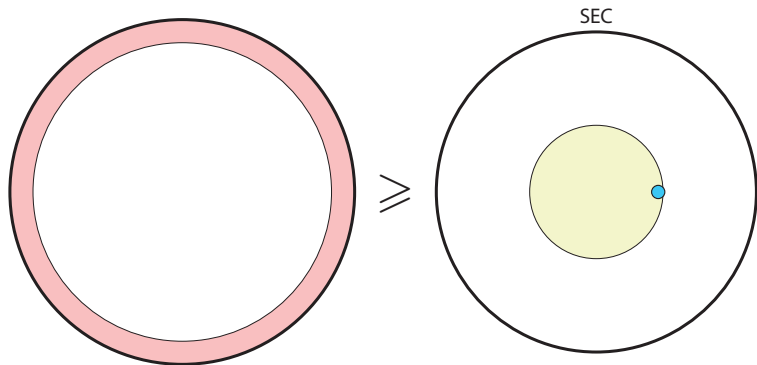
All robots lie in a thin annulus...

# Accidental formation of Pre-regular configurations



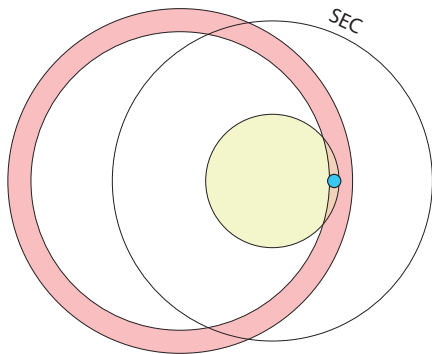
...whose outer circle is at least as large as the SEC

# Accidental formation of Pre-regular configurations



Suppose there is a robot in SEC/3

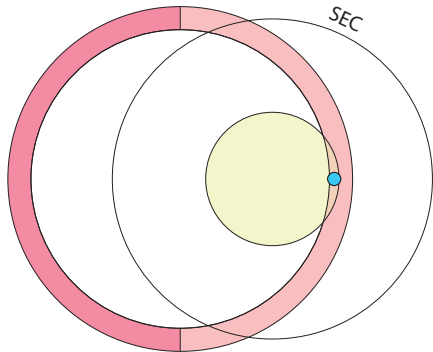
# Accidental formation of Pre-regular configurations



Then the annulus and  $SEC/3$  must overlap

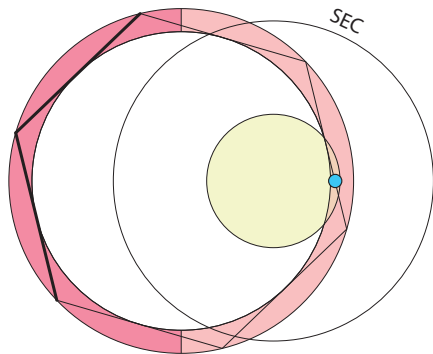


# Accidental formation of Pre-regular configurations



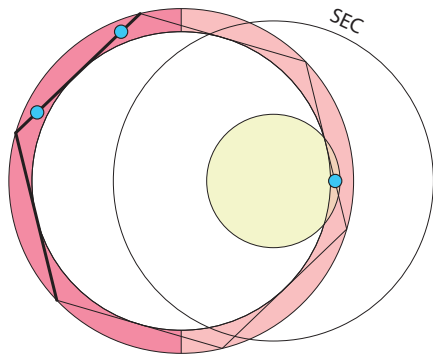
Hence a half-annulus lies outside SEC...

# Accidental formation of Pre-regular configurations



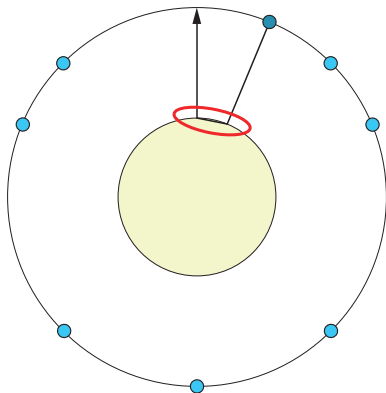
...but it contains two consecutive edges of the polygon...

# Accidental formation of Pre-regular configurations



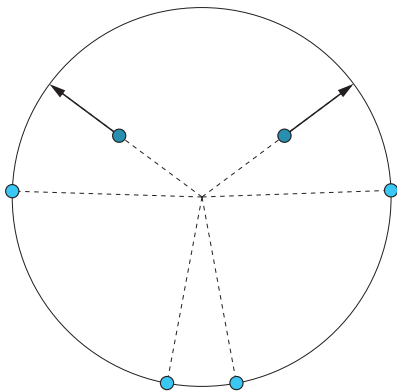
...and one of them must contain robots: contradiction!

## Accidental formation of Pre-regular configurations



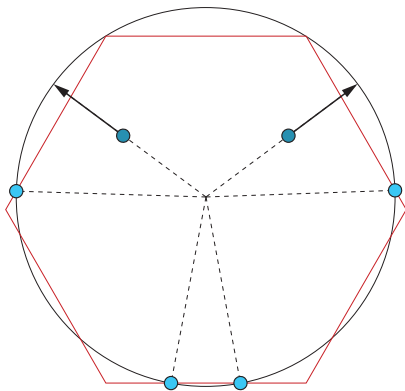
Hence all non-radial moves are safe, as they happen in  $SEC/3$

# Accidental formation of Pre-regular configurations



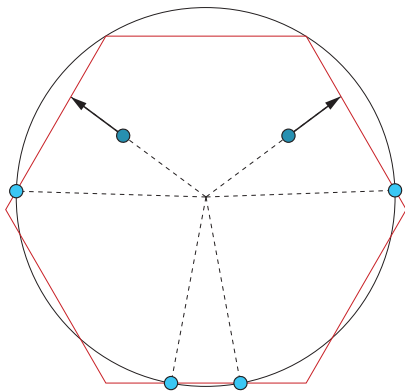
Radial moves are not safe, even if only one analogy class moves

# Accidental formation of Pre-regular configurations



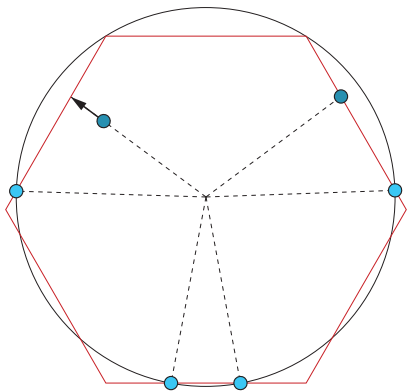
Radial moves are not safe, even if only one analogy class moves

# Accidental formation of Pre-regular configurations



**Strategy:** add **critical points** corresponding to Pre-regulars...

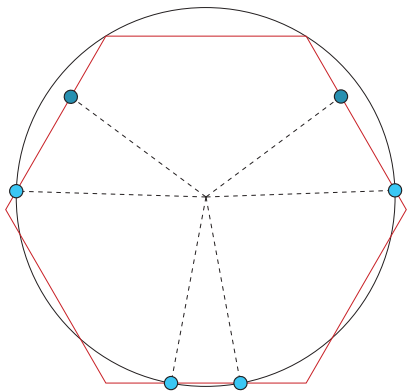
# Accidental formation of Pre-regular configurations



...stop at the next critical point and wait for each other

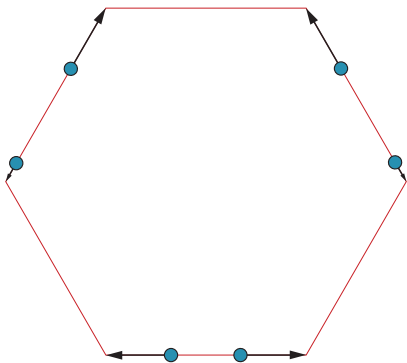


# Accidental formation of Pre-regular configurations



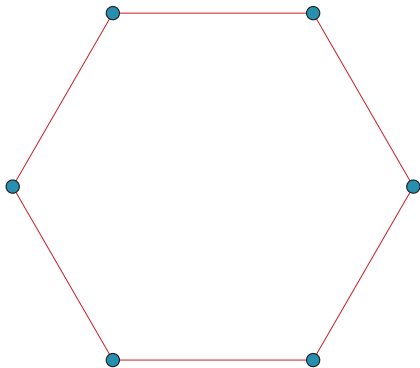
Whenever a Pre-regular is formed, all robots are stopped

# Accidental formation of Pre-regular configurations



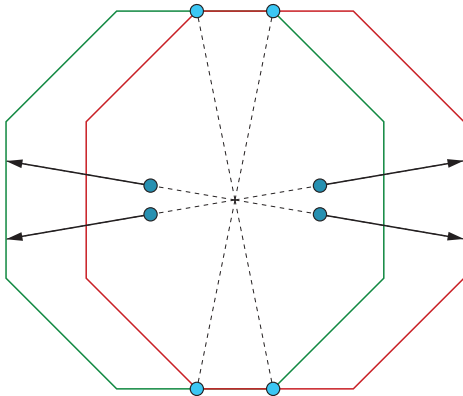
So they correctly transition

## Accidental formation of Pre-regular configurations



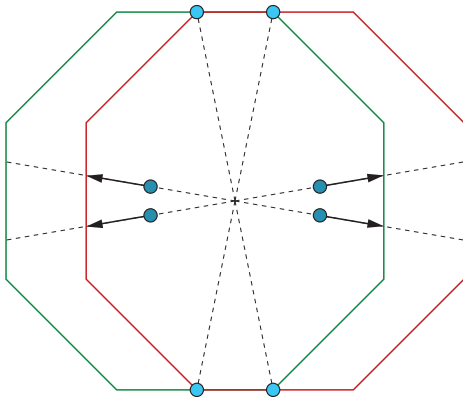
So they correctly transition

# Accidental formation of Pre-regular configurations



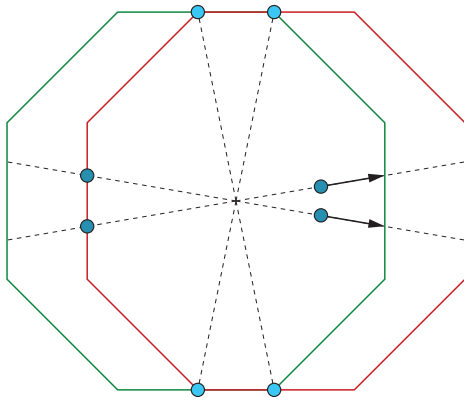
Corresponding critical points may lie on different Pre-regulars

# Accidental formation of Pre-regular configurations



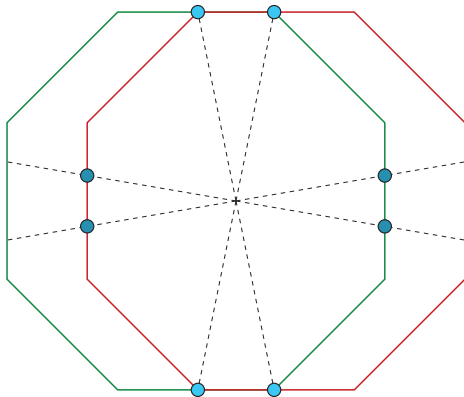
Even if robots wait for each other, they may get confused

# Accidental formation of Pre-regular configurations



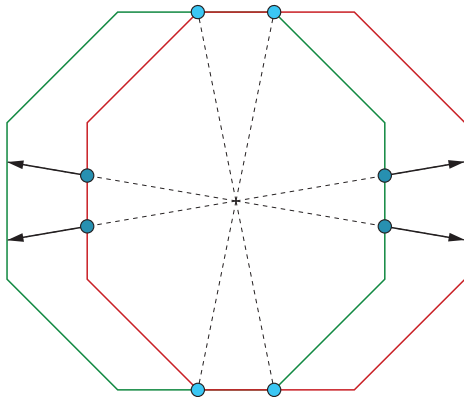
Even if robots wait for each other, they may get confused

# Accidental formation of Pre-regular configurations



Even if robots wait for each other, they may get confused

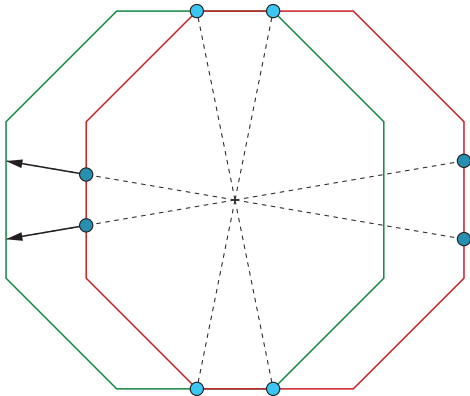
# Accidental formation of Pre-regular configurations



Even if robots wait for each other, they may get confused

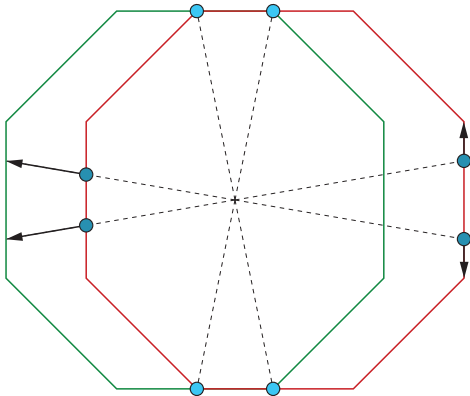


# Accidental formation of Pre-regular configurations



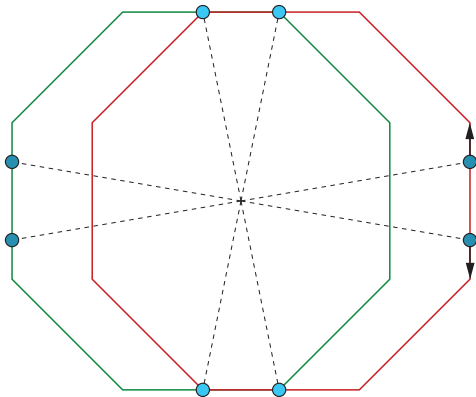
Even if robots wait for each other, they may get confused

# Accidental formation of Pre-regular configurations



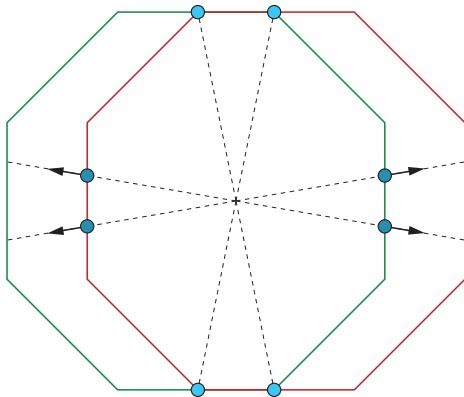
Even if robots wait for each other, they may get confused

# Accidental formation of Pre-regular configurations



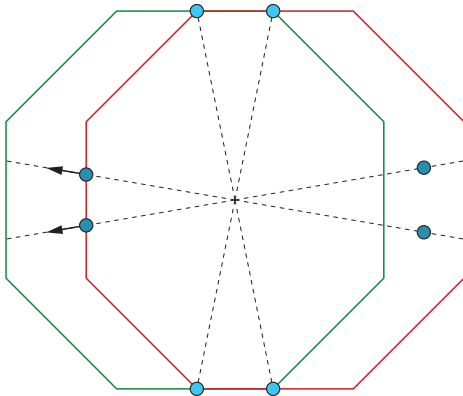
Even if robots wait for each other, they may get confused

# Accidental formation of Pre-regular configurations



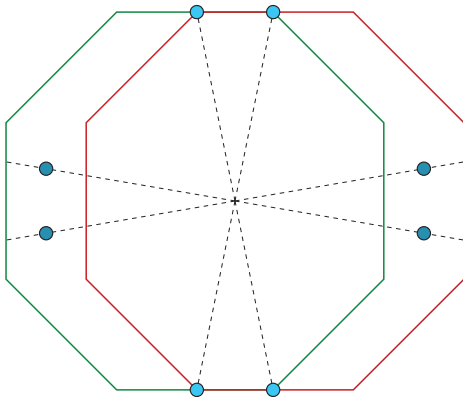
**Strategy:** add extra critical points between Pre-regulars

# Accidental formation of Pre-regular configurations



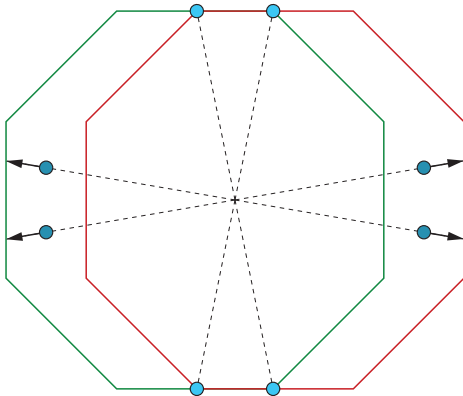
With this extra step, no Pre-regular is formed

# Accidental formation of Pre-regular configurations



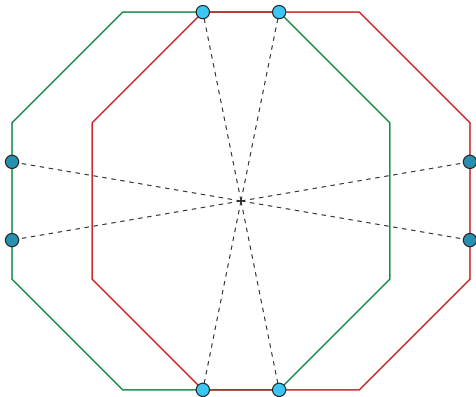
With this extra step, no Pre-regular is formed

# Accidental formation of Pre-regular configurations



With this extra step, no Pre-regular is formed

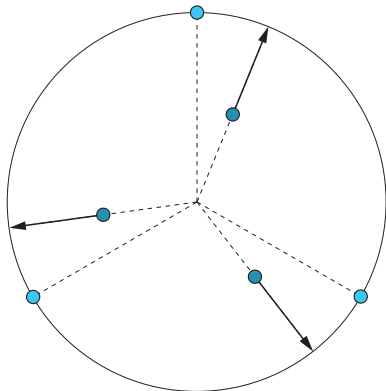
# Accidental formation of Pre-regular configurations



With this extra step, no Pre-regular is formed

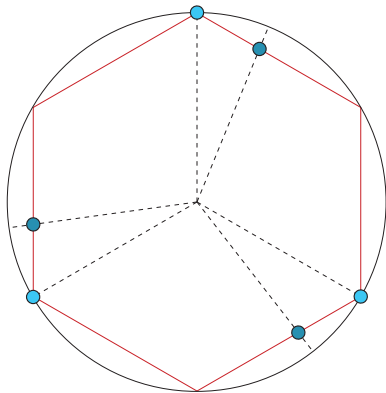


# Accidental formation of Pre-regular configurations



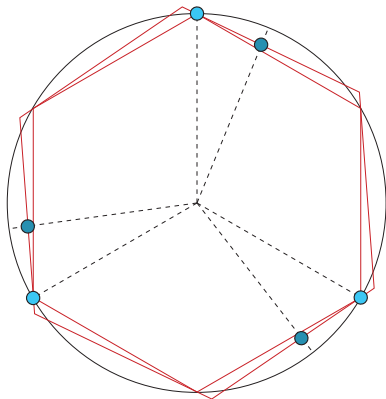
Formable Pre-regulars may be infinitely many!

# Accidental formation of Pre-regular configurations



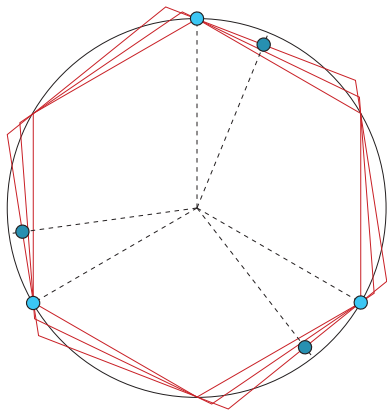
Formable Pre-regulars may be infinitely many!

# Accidental formation of Pre-regular configurations



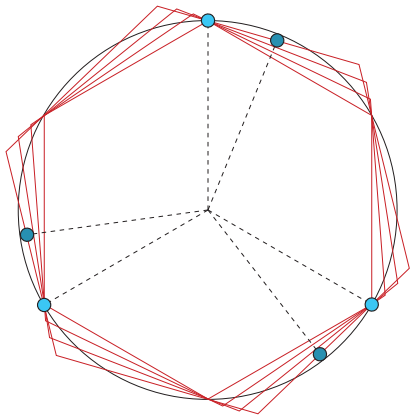
Formable Pre-regulars may be infinitely many!

# Accidental formation of Pre-regular configurations



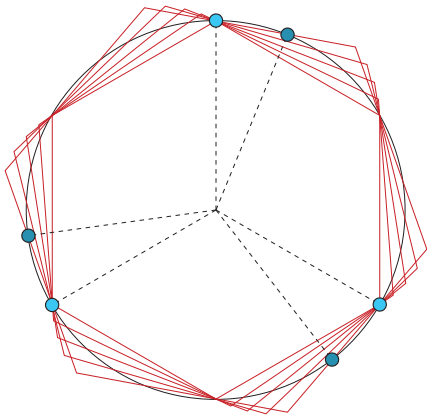
Formable Pre-regulars may be infinitely many!

# Accidental formation of Pre-regular configurations



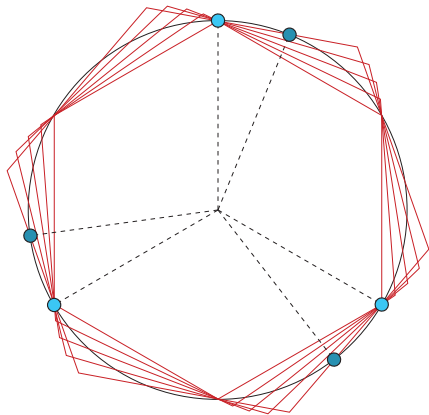
Formable Pre-regulars may be infinitely many!

# Accidental formation of Pre-regular configurations



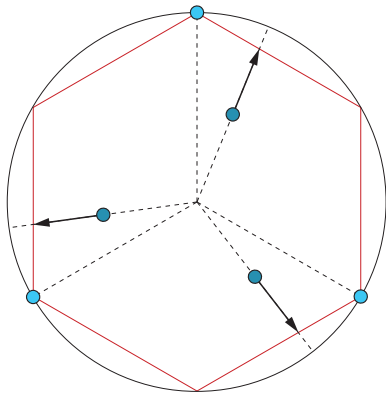
Formable Pre-regulars may be infinitely many!

# Accidental formation of Pre-regular configurations



But we cannot have infinitely many critical points

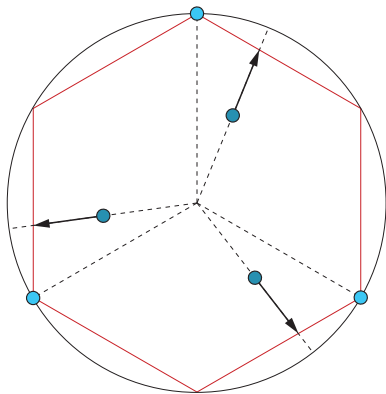
# Accidental formation of Pre-regular configurations



**Lemma:** a unique Pre-regular is formable before the others

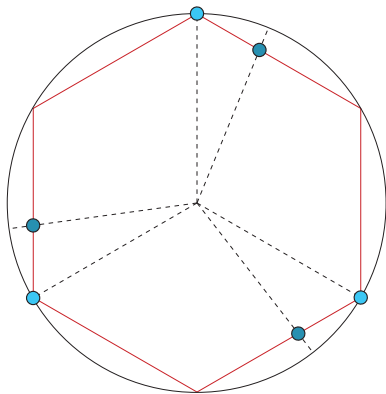


# Accidental formation of Pre-regular configurations



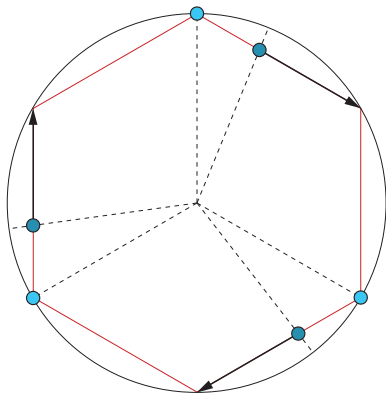
Hence finitely many critical points are sufficient

## Accidental formation of Pre-regular configurations



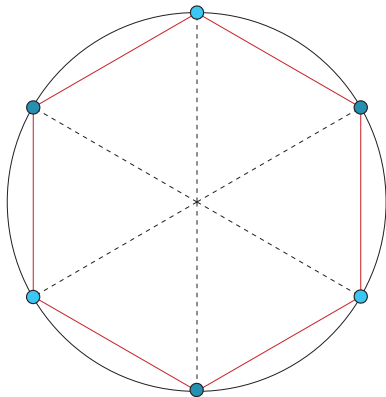
Hence finitely many critical points are sufficient

# Accidental formation of Pre-regular configurations



Hence finitely many critical points are sufficient

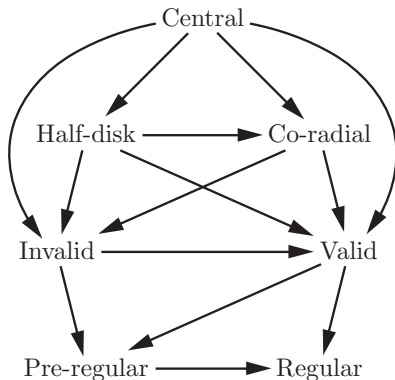
## Accidental formation of Pre-regular configurations



Hence finitely many critical points are sufficient

## Algorithm flow

As the algorithm proceeds, the current configuration transitions according to this diagram:



There are no loops in the diagram, and the only sink is the Regular configuration. So, in finite time a Regular configuration is reached.

## Summary and concluding remarks

The only solvable Pattern Formation problems are:

- **Gathering** problem (for  $n \neq 2$  robots)
- **Uniform Circle Formation** problem  
(today we saw a solution for  $n \neq 4$  robots:  
the case with 4 robots is solvable with an ad-hoc algorithm)

This is true even if

- robots are fully synchronous
- robots have a common notion of “clockwise”
- robots cannot be stopped in the middle of a movement

⇒ For Pattern Formation problems, these features  
are computationally irrelevant!



P. Flocchini, G. Prencipe, and N. Santoro

*Distributed Computing by Oblivious Mobile Robots*

Morgan & Claypool, 2012



G. Viglietta

“Uniform Circle Formation”

In *Distributed Computing by Mobile Entities*, Springer, 2019

# Assignment

In this assignment we study the *Line Formation* problem for asynchronous robots: the goal is to reach a configuration where all robots are on the same straight line. Robots should never collide, and they should permanently stop as soon as they form a line.

We limit our analysis to a swarm of only 3 robots:

1. Prove that no distributed algorithm can solve the Line Formation problem from an initial configuration in which the robots form an equilateral triangle.
2. Give a distributed algorithm that solves the Line Formation problem from any initial configuration that is not an equilateral triangle (assuming that no two robots are initially in the same location), and prove the algorithm's correctness.