Programmable Matter: From Fractal Formation to Genetic-Programming Solutions

Dagstuhl Seminar 23091

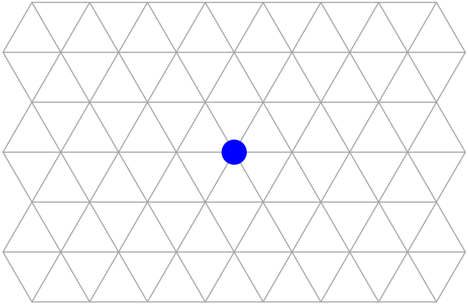
Giovanni Viglietta

Schloss Dagstuhl – March 2, 2023

Talk Overview

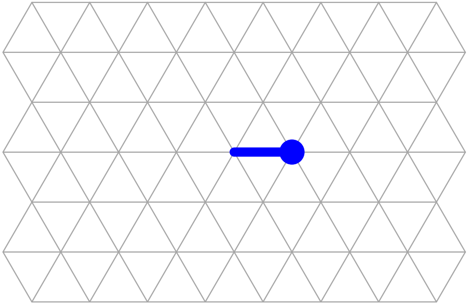
- “Naive approach” to Shape Formation by Amoebots
    - Leader election
    - Handedness agreement
    - Line formation
    - Simulation of a Counter Machine
    - General shape formation
- Shortcomings of the “naive approach”
- Genetic-Programming approach
    - Preliminary results
    - Open problems

Amoebots



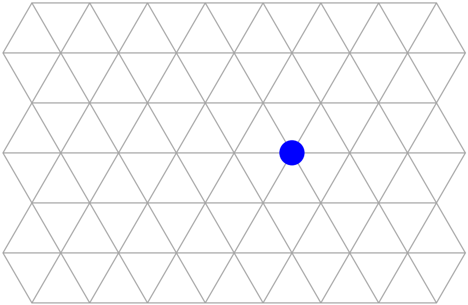
In this model, particles occupy nodes of a triangular grid.

Amoebots



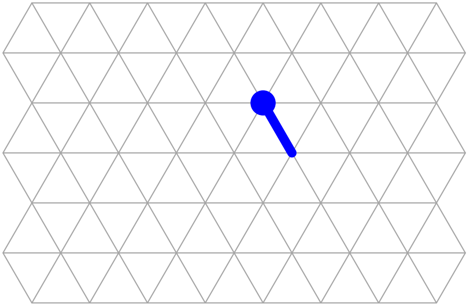
A particle can move by *expanding* and *contracting*.

Amoebots



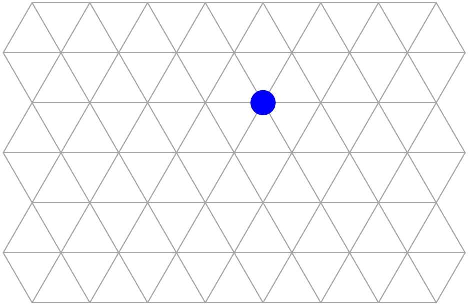
A particle can move by *expanding* and *contracting*.

Amoebots



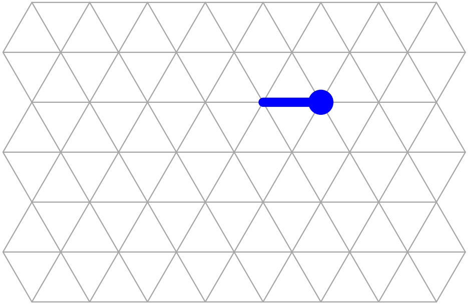
A particle can move by *expanding* and *contracting*.

Amoebots



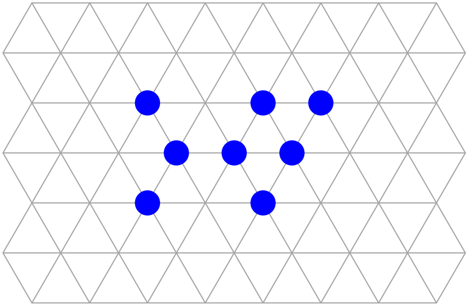
A particle can move by *expanding* and *contracting*.

Amoebots



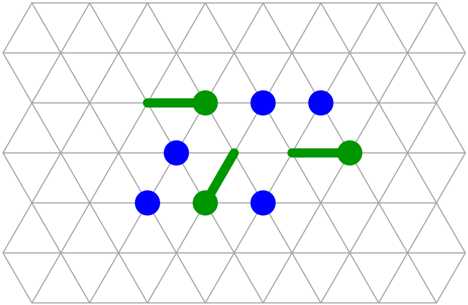
A particle can move by *expanding* and *contracting*.

Amoebots



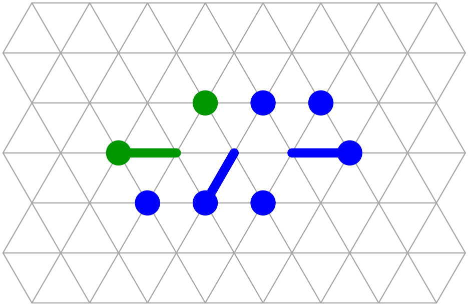
A *system* of particles is given.

Amoebots



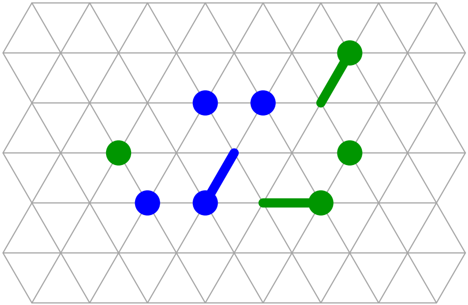
Particles move *asynchronously* following an algorithm.

Amoebots



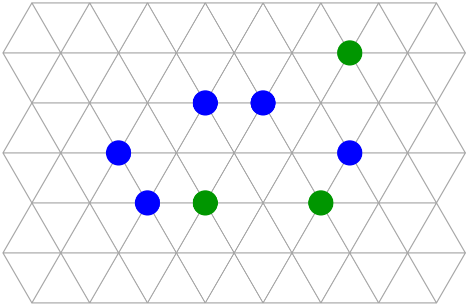
Particles move *asynchronously* following an algorithm.

Amoebots



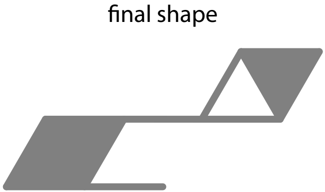
At each step, any set of particles is activated by an *adversary*.

Amoebots



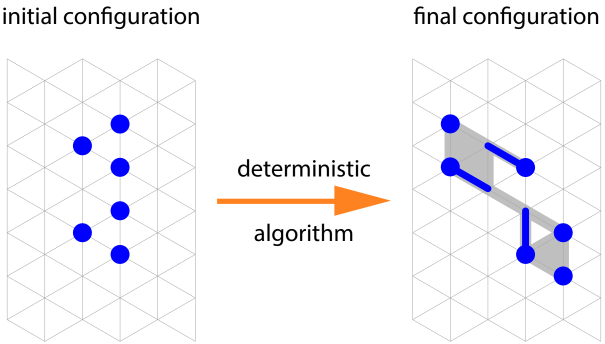
At each step, any set of particles is activated by an *adversary*.

Shape Formation



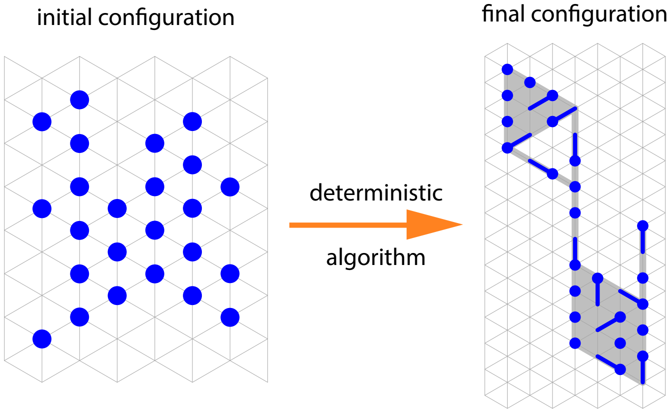
The goal is to form a *shape* that is given as input to all particles.

Shape Formation



The shape-formation algorithm should be *deterministic*.

Shape Formation

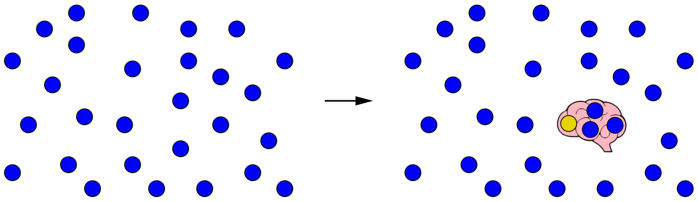


The shape can be scaled up depending on the size of the system.

Shape Formation: Naive Approach

Theorem (*Euro-Par 2020 / Dist. Comp., 2020*)

There is a distributed algorithm for finite-state Amoebots that allows them to form any Turing-computable shape.

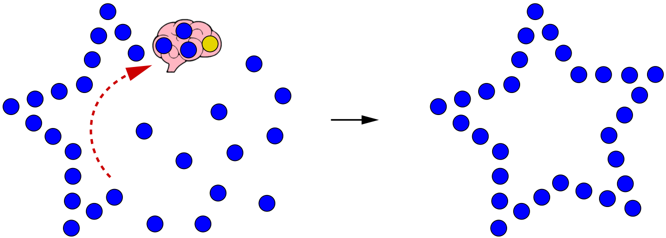


The algorithm starts with a deterministic Leader Election phase. The leader then recruits some particles to simulate a "moving Counter Machine" that travels across the system and instructs every particle on where to go to form the final shape.

Shape Formation: Naive Approach

Theorem (*Euro-Par 2020 / Dist. Comp., 2020*)

There is a distributed algorithm for finite-state Amoebots that allows them to form any Turing-computable shape.



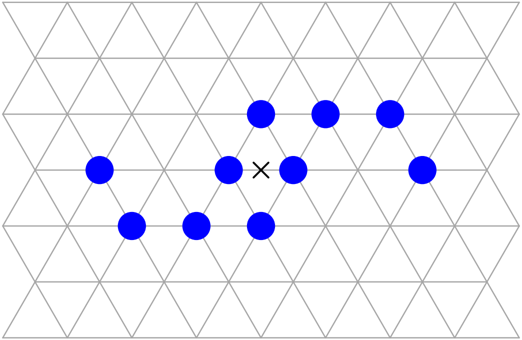
The algorithm starts with a deterministic Leader Election phase. The leader then recruits some particles to simulate a "moving Counter Machine" that travels across the system and instructs every particle on where to go to form the final shape.

Particle Model

The n particles in the system:

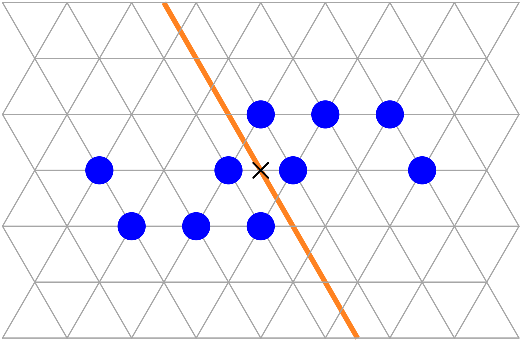
- initially form any simply connected shape
- know the final shape but do *not* know n
- have a constant amount of internal memory
- are anonymous and start in the same state
- can only see and communicate with adjacent particles
- do not have a *compass*
- may not agree on a *clockwise direction*
- are activated asynchronously (actually, semi-synchronously)
- execute the same deterministic algorithm
- cannot occupy the same node

Unbreakable Symmetry



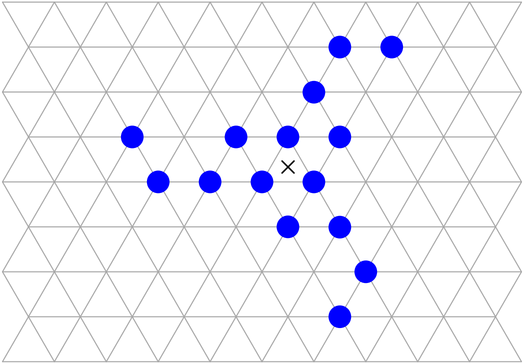
If the system has a center of symmetry not on a grid node...

Unbreakable Symmetry



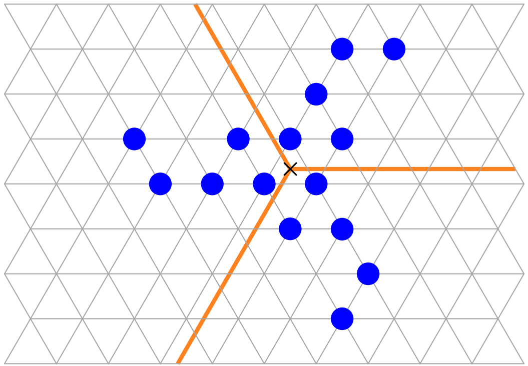
Then this symmetry is impossible to break.

Unbreakable Symmetry



The same holds for systems with a 3-fold rotational symmetry.

Unbreakable Symmetry



If the center is not on a grid node, the symmetry is unbreakable.

Statement of Results

Observation

If the system initially has an unbreakable 2- or 3-symmetry, it cannot form shapes that do not have the same type of symmetry.

Theorem

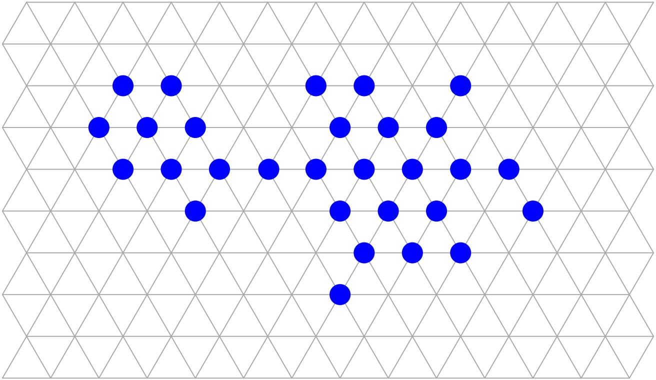
For all other cases, there is a universal shape-formation algorithm, provided that the system initially forms a simply connected shape, and the final shape and its scaled-up copies are Turing-computable (with some bland extra assumptions).

The extra assumptions are satisfied by connected shapes, so:

Corollary

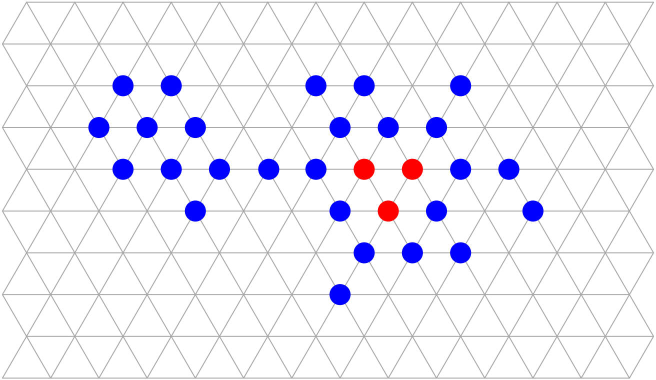
A system that initially forms a simply connected shape can form a final shape whose scaled-up copies are Turing-computable and connected if and only if this does not contradict the Observation.

Universal Shape-Formation Algorithm



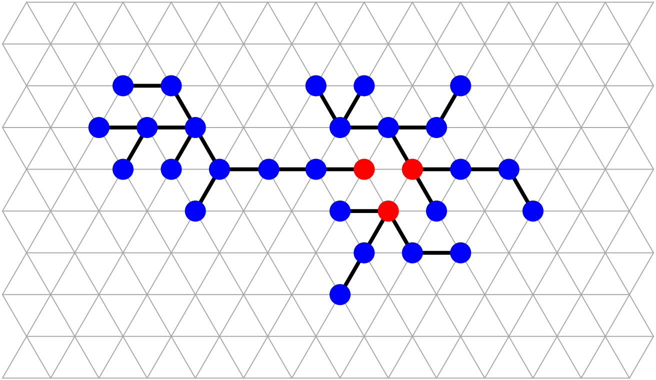
Start with a sufficiently large simply connected system.

Universal Shape-Formation Algorithm



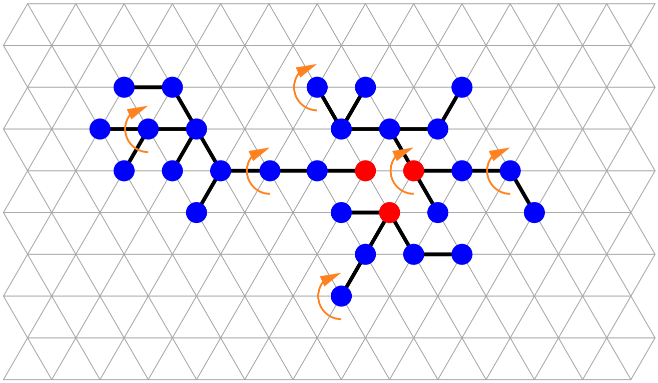
Phase 1: attempt to elect a leader.

Universal Shape-Formation Algorithm



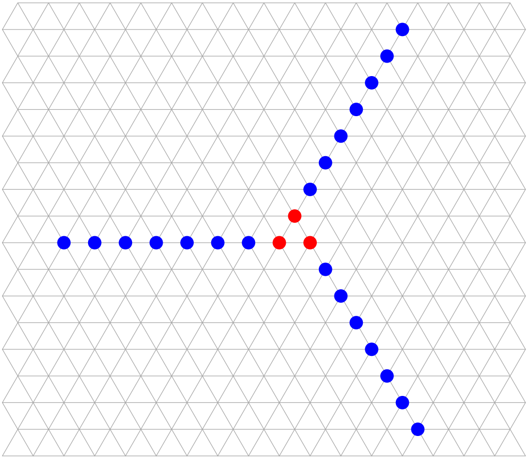
Phase 2: construct a spanning forest.

Universal Shape-Formation Algorithm



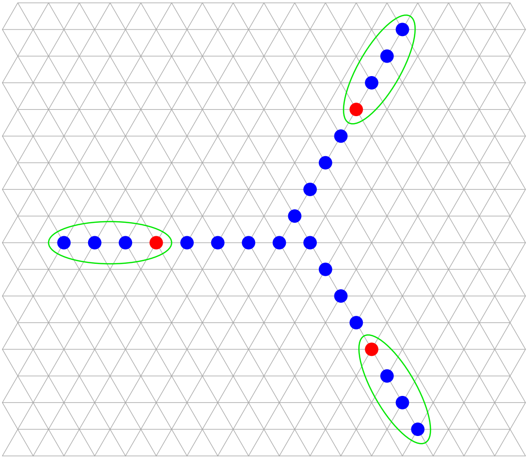
Phase 3: agree on a clockwise direction.

Universal Shape-Formation Algorithm



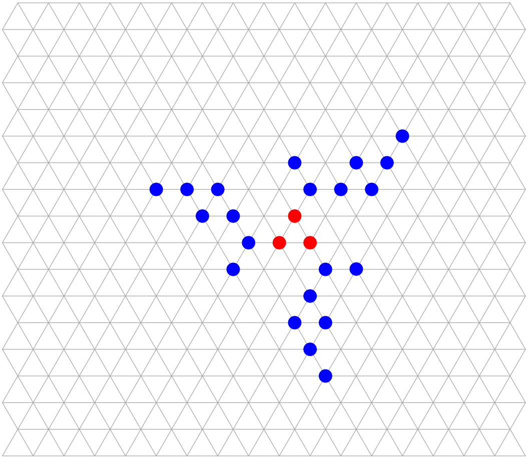
Phase 4: form one line per leader.

Universal Shape-Formation Algorithm



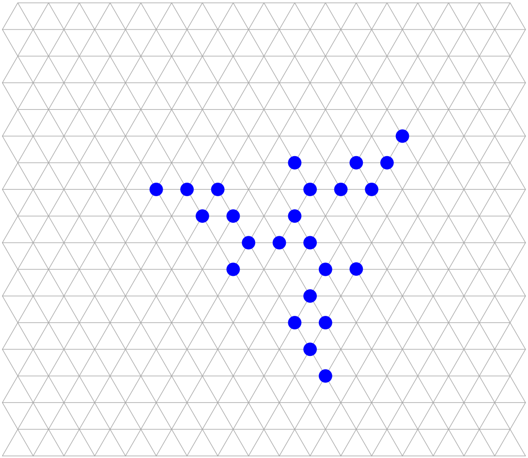
Phase 5: simulate Counter Machines to compute the final shape.

Universal Shape-Formation Algorithm



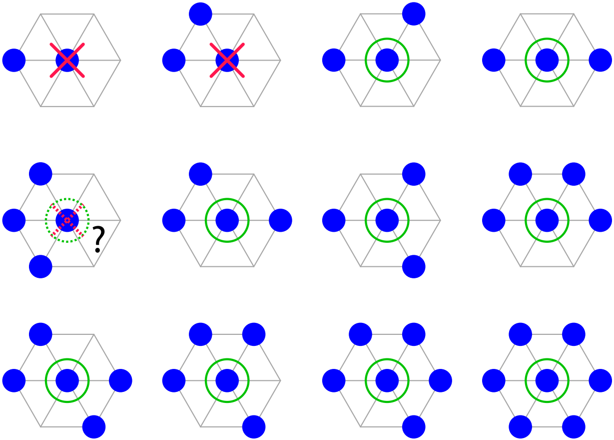
Phase 6: keep computing while forming the final shape.

Universal Shape-Formation Algorithm



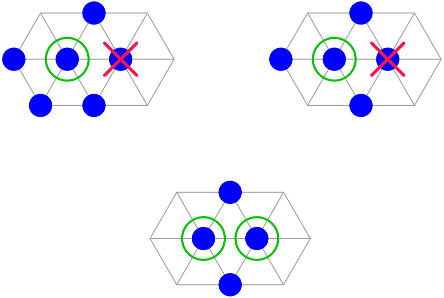
Phase 6: keep computing while forming the final shape.

Lattice Consumption Phase



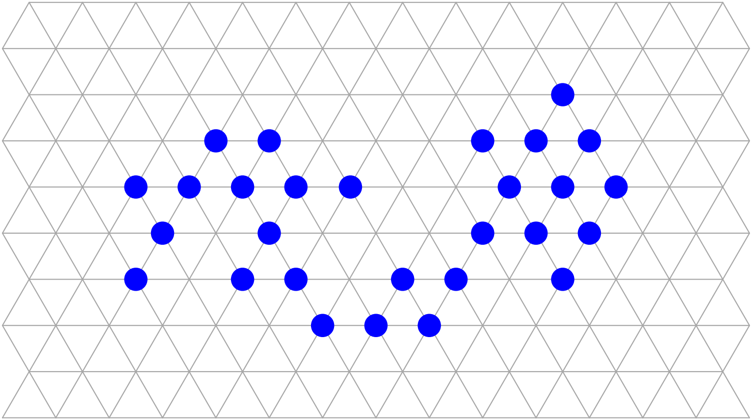
All particles are initially *eligible*. Depending on its eligible neighbors, a particle may decide to *eliminate* itself or stay eligible.

Lattice Consumption Phase



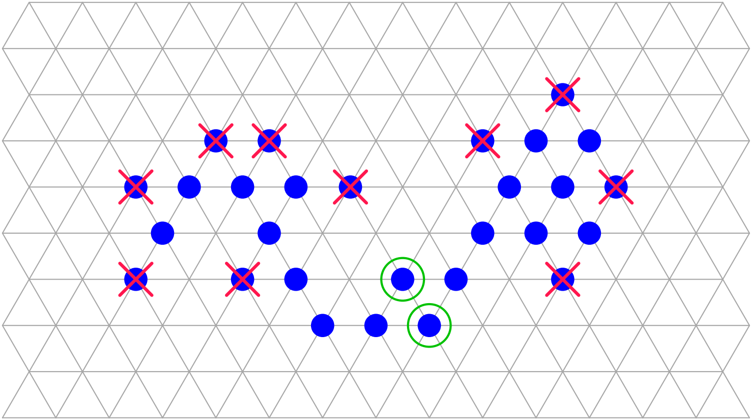
There is just one special case, where the particle has to communicate with a neighbor to ensure that its elimination would not disconnect the set of eligible particles.

Lattice Consumption Phase



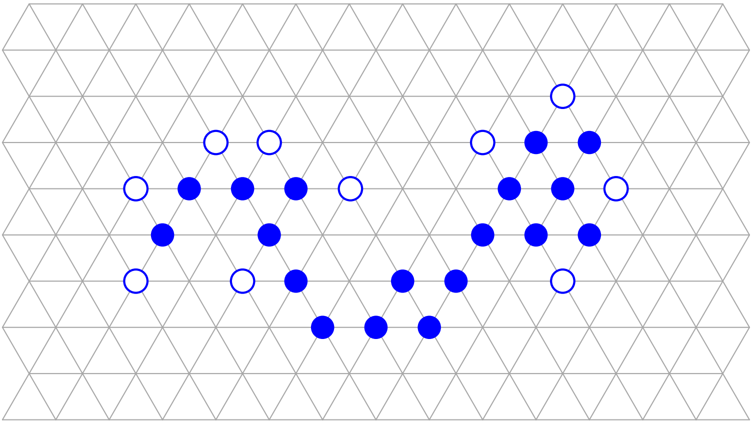
Following this protocol, the set of eligible particles remains simply connected, even if activations occur asynchronously.

Lattice Consumption Phase



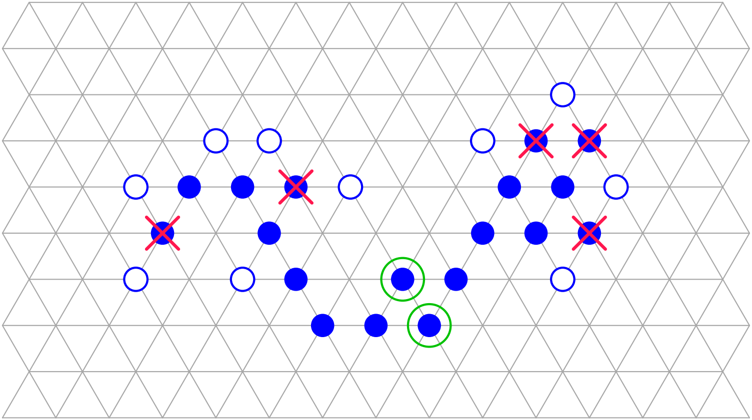
Following this protocol, the set of eligible particles remains simply connected, even if activations occur asynchronously.

Lattice Consumption Phase



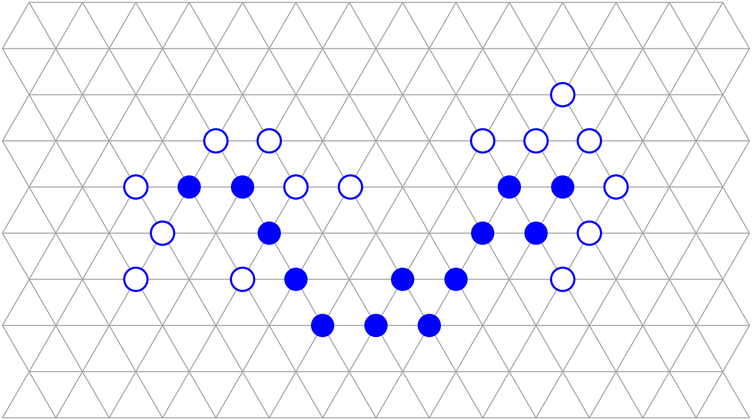
Following this protocol, the set of eligible particles remains simply connected, even if activations occur asynchronously.

Lattice Consumption Phase



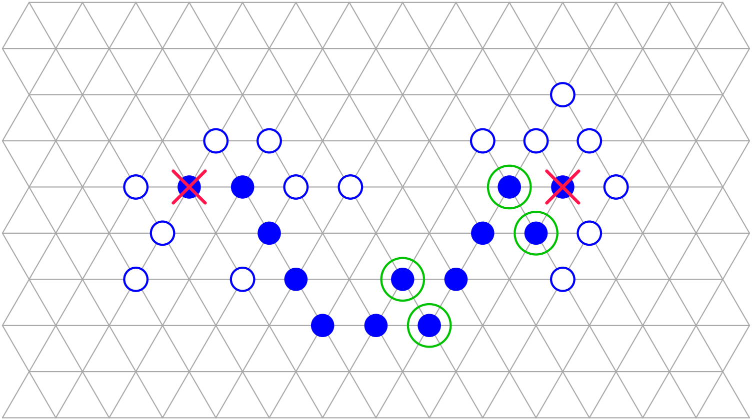
Following this protocol, the set of eligible particles remains simply connected, even if activations occur asynchronously.

Lattice Consumption Phase



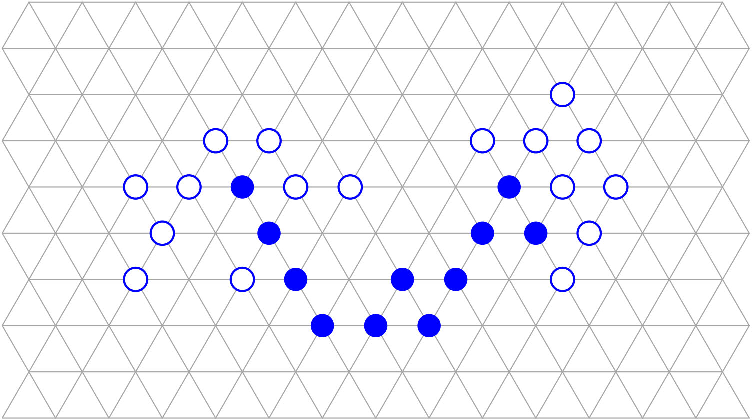
Following this protocol, the set of eligible particles remains simply connected, even if activations occur asynchronously.

Lattice Consumption Phase



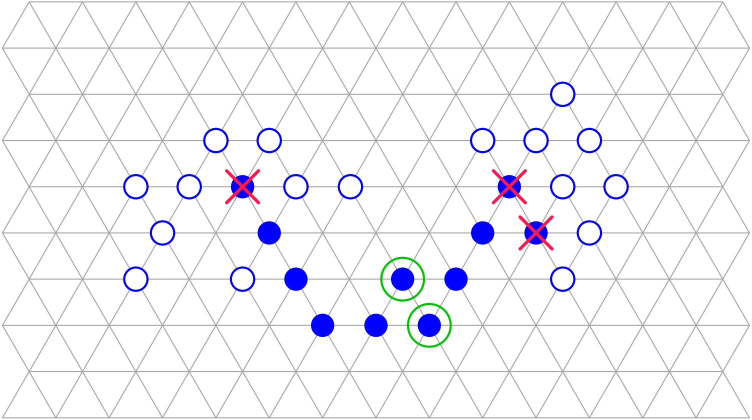
Following this protocol, the set of eligible particles remains simply connected, even if activations occur asynchronously.

Lattice Consumption Phase



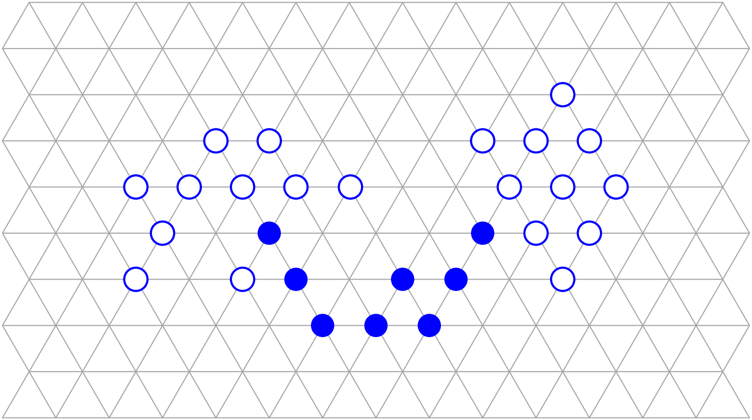
Following this protocol, the set of eligible particles remains simply connected, even if activations occur asynchronously.

Lattice Consumption Phase



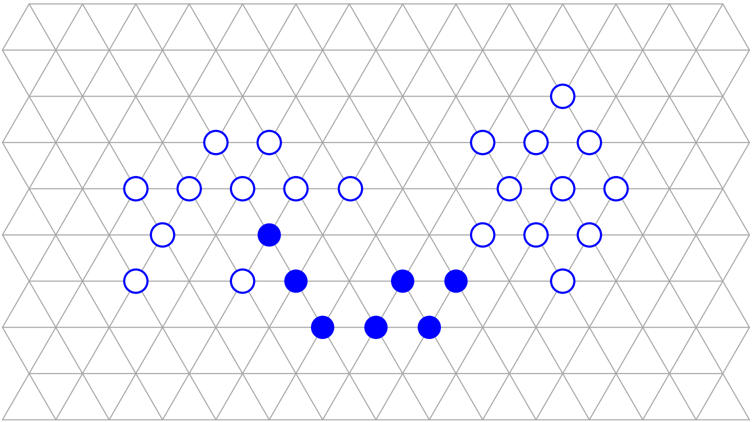
Following this protocol, the set of eligible particles remains simply connected, even if activations occur asynchronously.

Lattice Consumption Phase



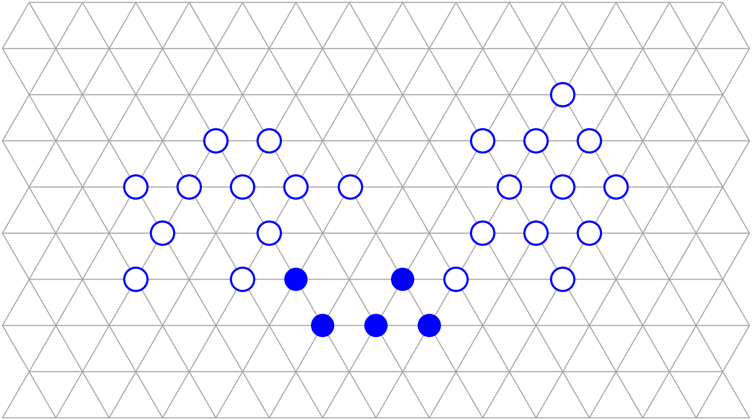
Following this protocol, the set of eligible particles remains simply connected, even if activations occur asynchronously.

Lattice Consumption Phase



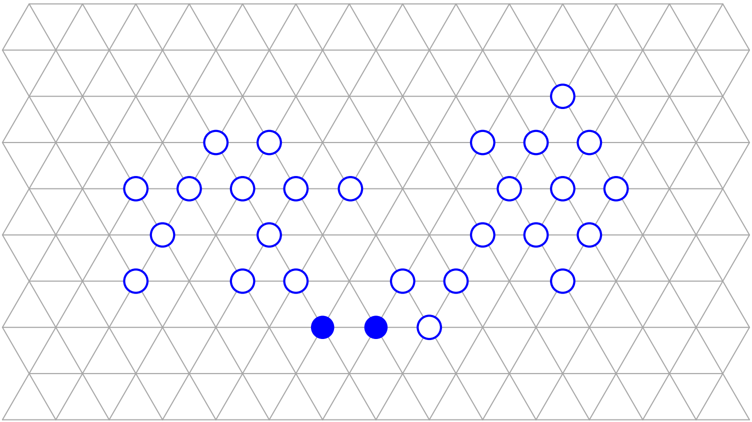
Following this protocol, the set of eligible particles remains simply connected, even if activations occur asynchronously.

Lattice Consumption Phase



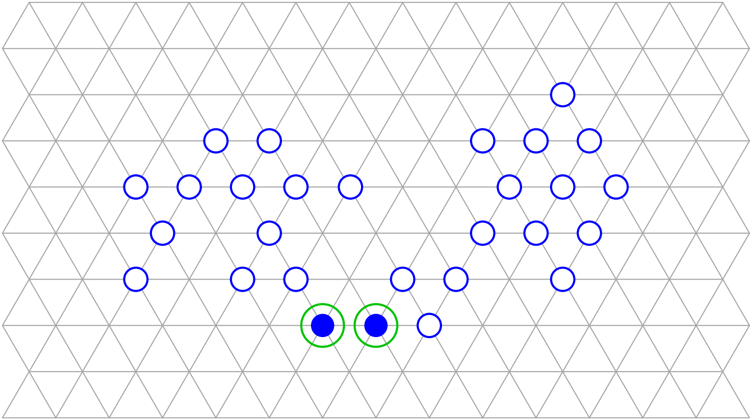
Following this protocol, the set of eligible particles remains simply connected, even if activations occur asynchronously.

Lattice Consumption Phase



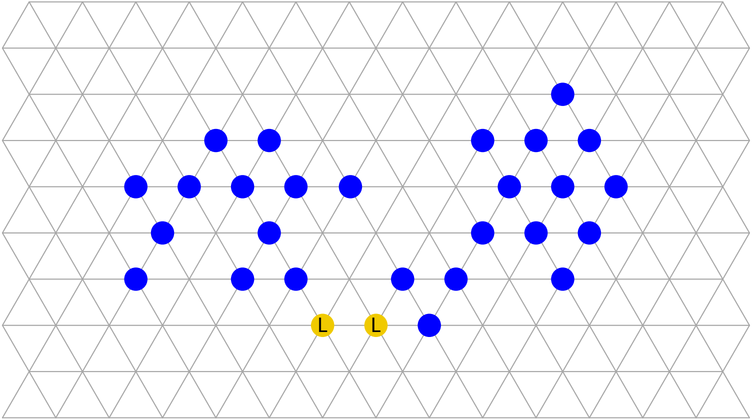
Following this protocol, the set of eligible particles remains simply connected, even if activations occur asynchronously.

Lattice Consumption Phase



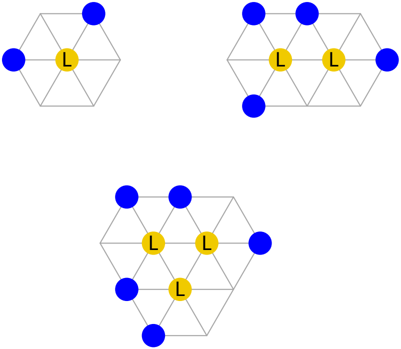
When the process ends, the particles that are still eligible become *candidate leaders*.

Lattice Consumption Phase



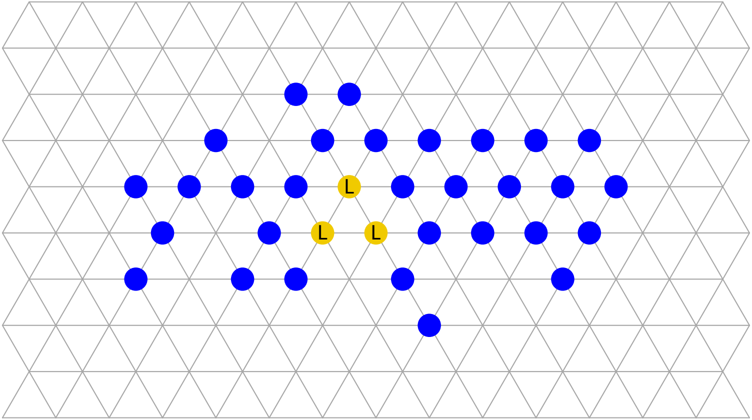
When the process ends, the particles that are still eligible become *candidate leaders*.

Lattice Consumption Phase



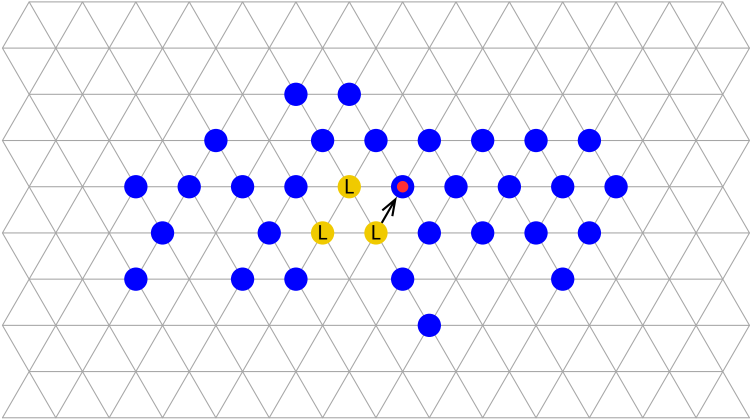
The candidate leaders are all adjacent, and can be at most 3.

Spanning Forest Construction Phase



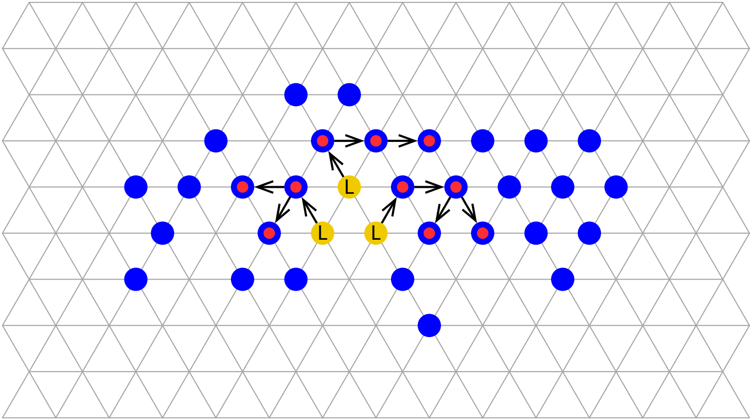
Each candidate leader starts constructing a tree.

Spanning Forest Construction Phase



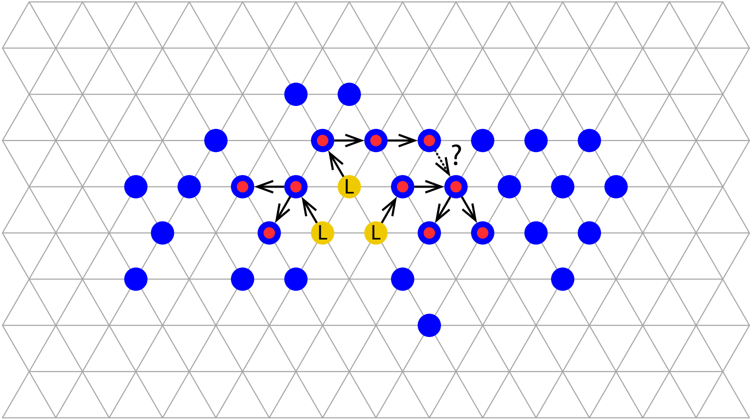
Each candidate leader starts constructing a tree.

Spanning Forest Construction Phase



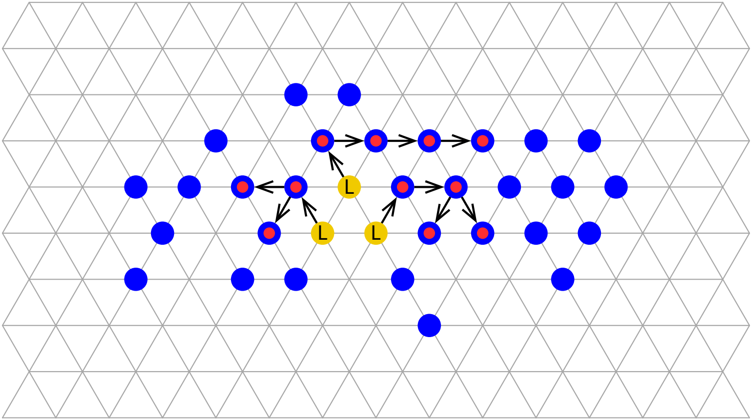
Each node of a tree tries to extend the tree in all directions by sending *merge requests* to its neighbors.

Spanning Forest Construction Phase



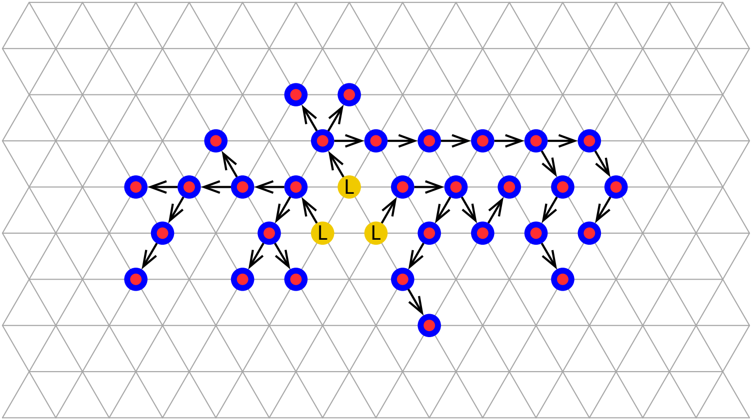
If a node is already part of a tree, it refuses further merge requests.

Spanning Forest Construction Phase



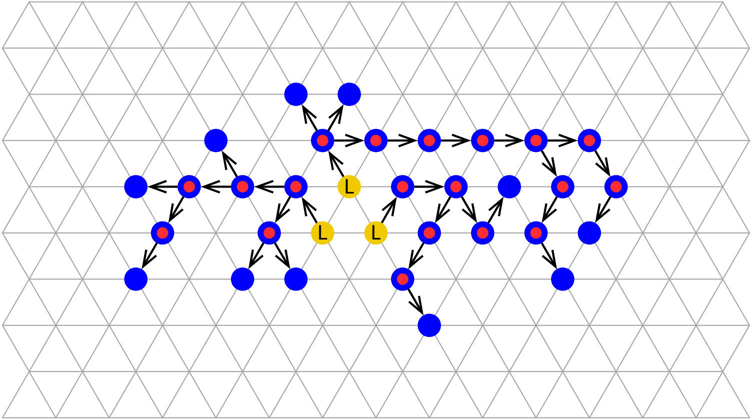
Otherwise, it sets a *parent* variable to the *port number* corresponding to a neighbor that sent a request.

Spanning Forest Construction Phase



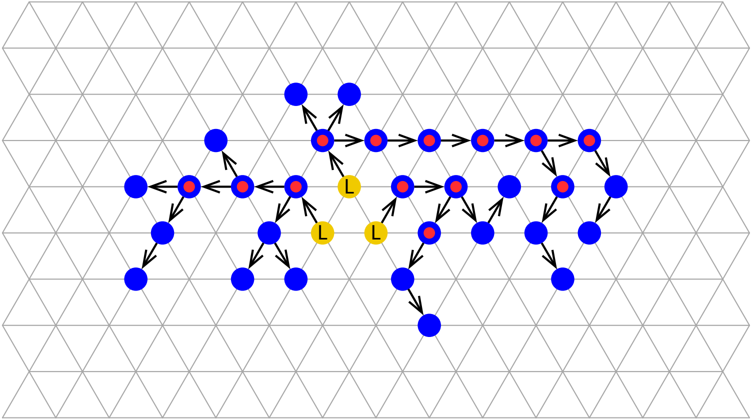
Since the shape is connected, eventually a spanning forest is constructed.

Spanning Forest Construction Phase



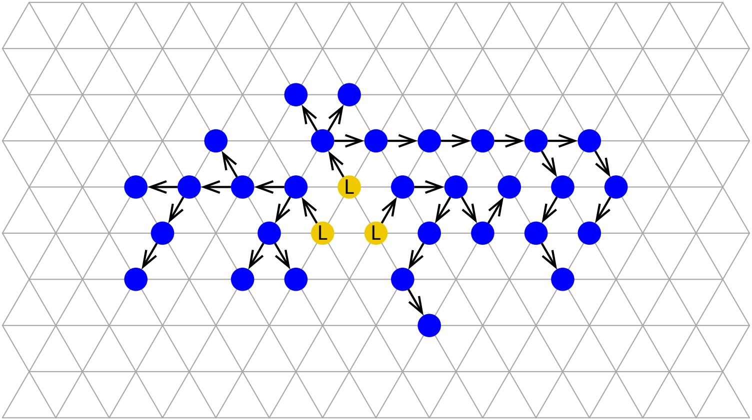
Nodes that cannot expand anymore send a *termination message* to their parents, starting from the leaves.

Spanning Forest Construction Phase



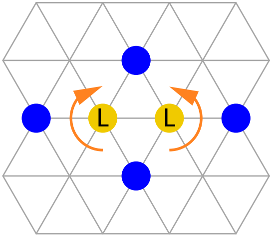
Nodes that cannot expand anymore send a *termination message* to their parents, starting from the leaves.

Spanning Forest Construction Phase



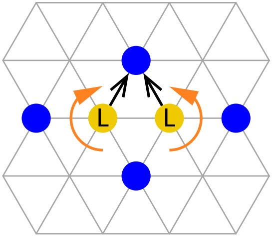
Eventually, the termination messages reach the candidate leaders, and the phase ends.

Handedness Agreement Phase



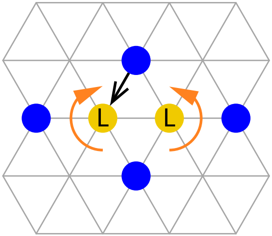
We want two candidate leaders to agree on the same handedness.

Handedness Agreement Phase



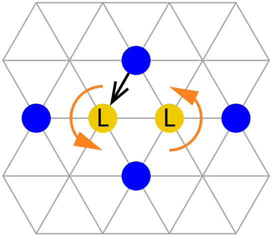
If they have a common neighbor, they send a message to it.
If the same neighbor receives both messages, it means that the candidate leaders have opposite handedness.

Handedness Agreement Phase



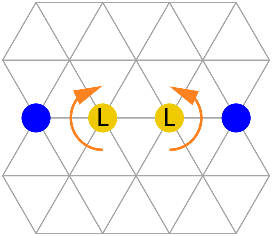
So, the neighbor decides which candidate leader has to change its handedness, and sends it a message.

Handedness Agreement Phase



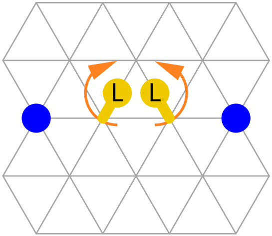
So, the neighbor decides which candidate leader has to change its handedness, and sends it a message.

Handedness Agreement Phase



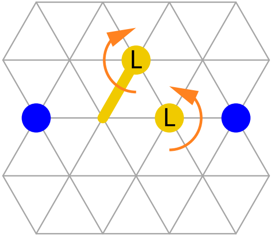
If the candidate leaders have no common neighbor,
they try to expand to a neighboring location.

Handedness Agreement Phase



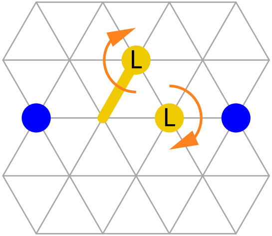
If the candidate leaders have no common neighbor, they try to expand to a neighboring location.

Handedness Agreement Phase



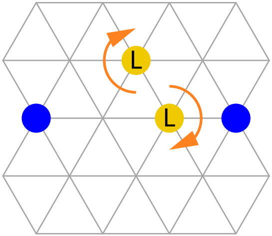
If one of them fails to reach it, it means that they have opposite handedness.

Handedness Agreement Phase



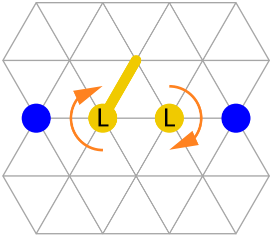
So, the candidate leader that fails to expand changes its own handedness.

Handedness Agreement Phase



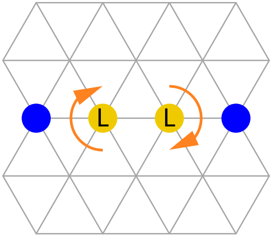
If a candidate leader succeeds to expand, it then contracts and moves back to its original location.

Handedness Agreement Phase



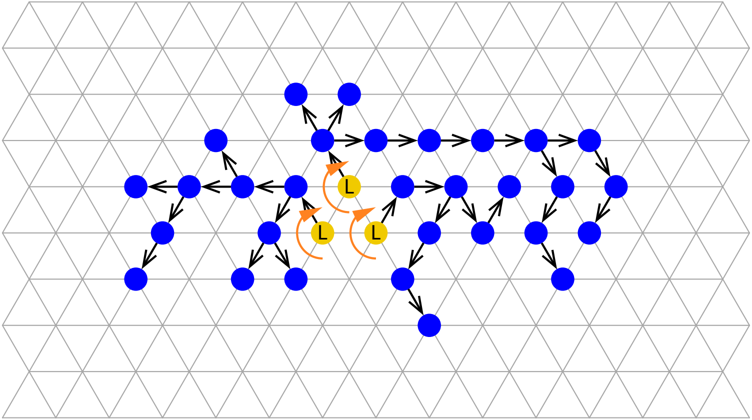
If a candidate leader succeeds to expand, it then contracts and moves back to its original location.

Handedness Agreement Phase



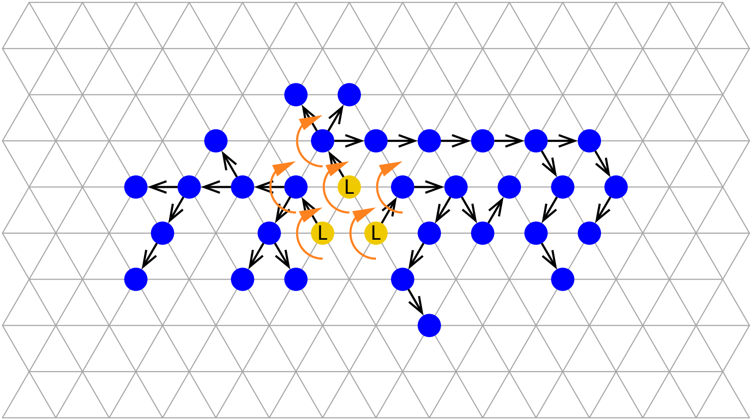
If a candidate leader succeeds to expand, it then contracts and moves back to its original location.

Handedness Agreement Phase



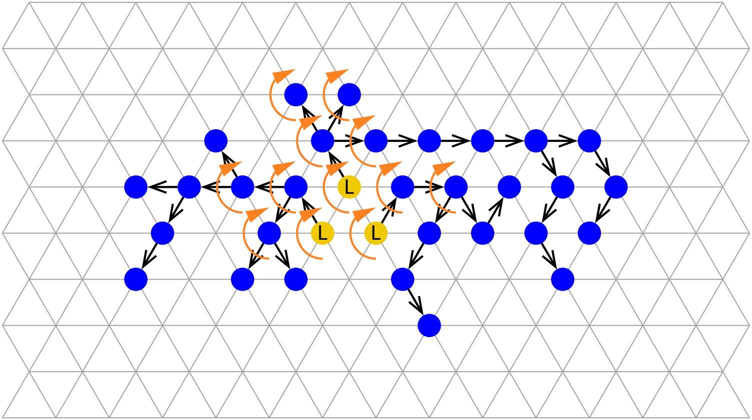
Eventually, all candidate leaders get the same handedness.

Handedness Agreement Phase



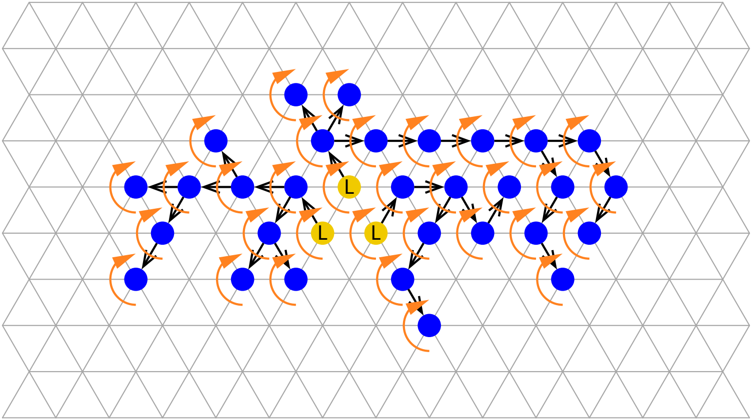
By a similar protocol, the agreed-upon handedness is communicated along the trees until all particles agree.

Handedness Agreement Phase



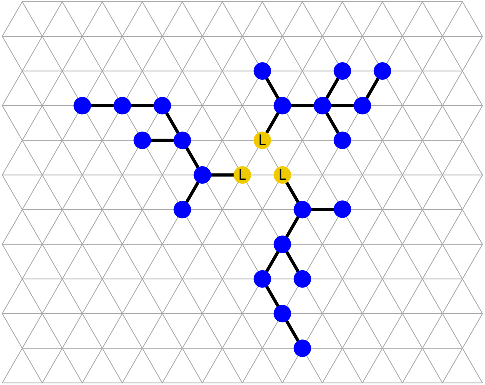
By a similar protocol, the agreed-upon handedness is communicated along the trees until all particles agree.

Handedness Agreement Phase



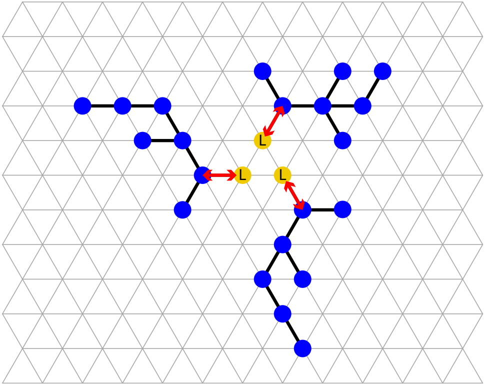
Since several instances of the protocol are taking place across the network, appropriate *locking* and *unlocking* strategies have to be implemented, and the absence of *deadlocks* has to be proven.

Leader Election Phase



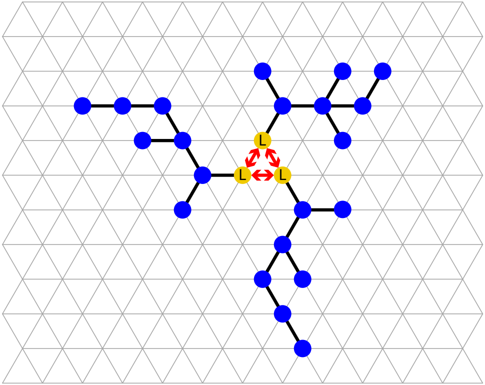
The candidate leaders want to compare their respective trees, in an attempt to break symmetry.

Leader Election Phase



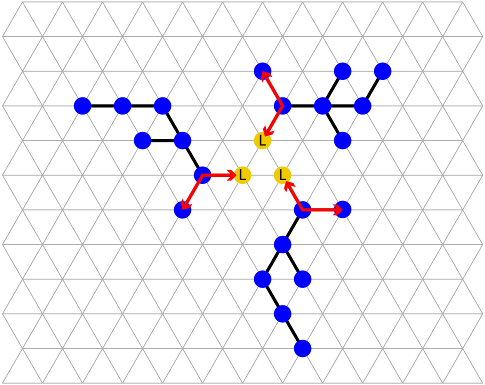
They do so by a breadth-first search, forwarding a message to a node and waiting for it to reply with a representation of its neighborhood.

Leader Election Phase



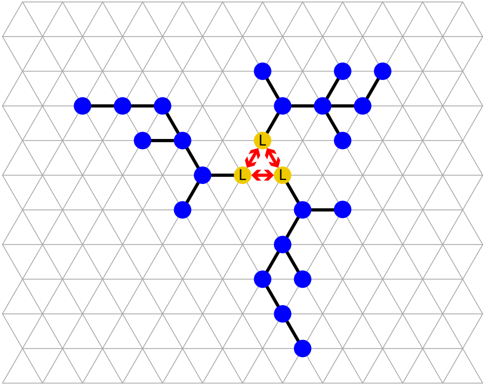
When all candidate leaders have received a reply from a node, they compare it to see if the symmetry can be broken.

Leader Election Phase



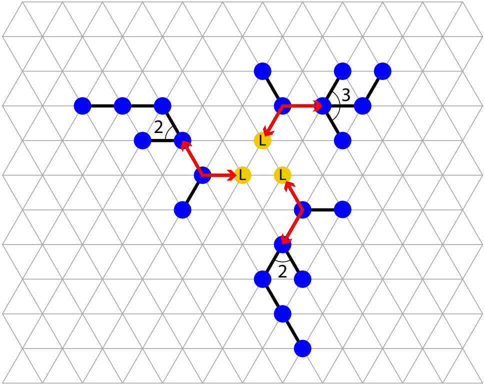
If the replies are all equal, they proceed with the next node.

Leader Election Phase



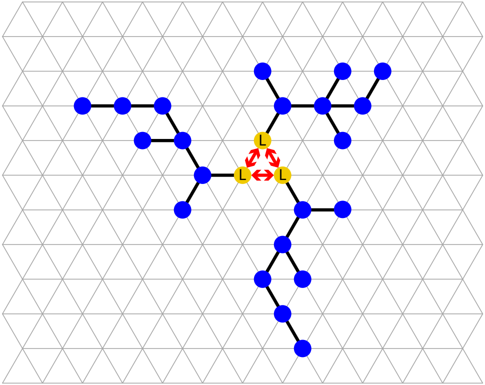
If the replies are all equal, they proceed with the next node.

Leader Election Phase



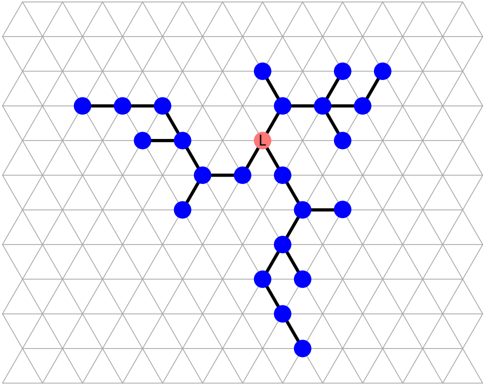
As soon as the replies are not all equal, a unique leader is elected, and the other candidate leaders become its children.

Leader Election Phase



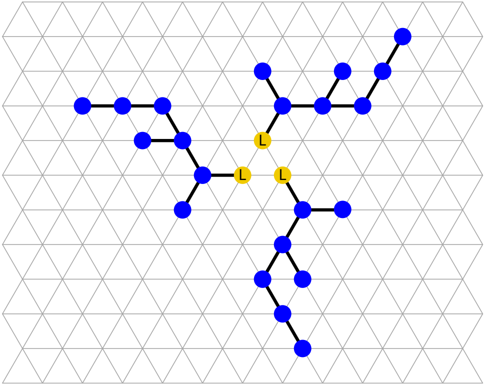
As soon as the replies are not all equal, a unique leader is elected, and the other candidate leaders become its children.

Leader Election Phase



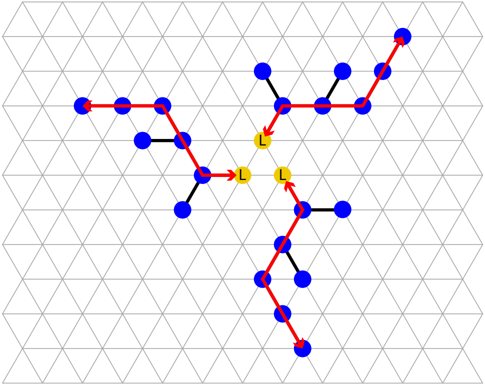
As soon as the replies are not all equal, a unique leader is elected, and the other candidate leaders become its children.

Leader Election Phase



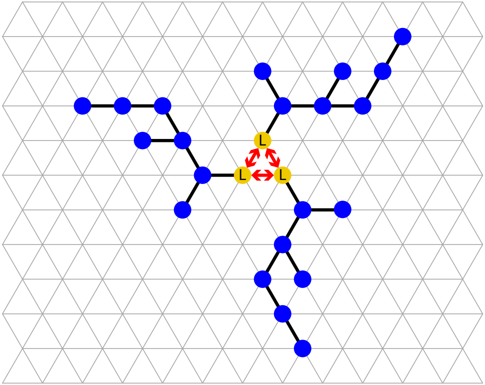
If the last node of each tree has been reached and the replies are still all equal, then the trees must be equal and equally oriented (because all particles agree on the same handedness).

Leader Election Phase



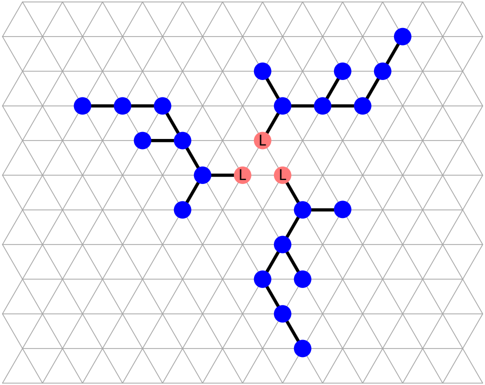
If the last node of each tree has been reached and the replies are still all equal, then the trees must be equal and equally oriented (because all particles agree on the same handedness).

Leader Election Phase



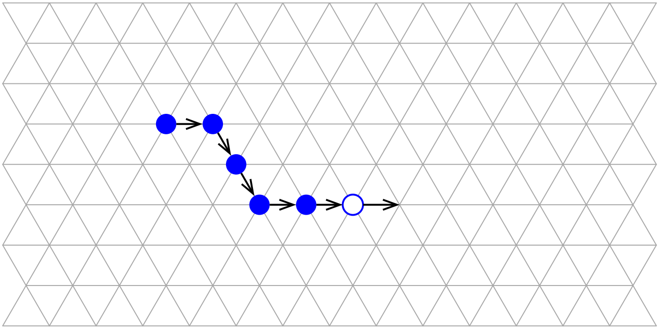
If the last node of each tree has been reached and the replies are still all equal, then the trees must be equal and equally oriented (because all particles agree on the same handedness).

Leader Election Phase



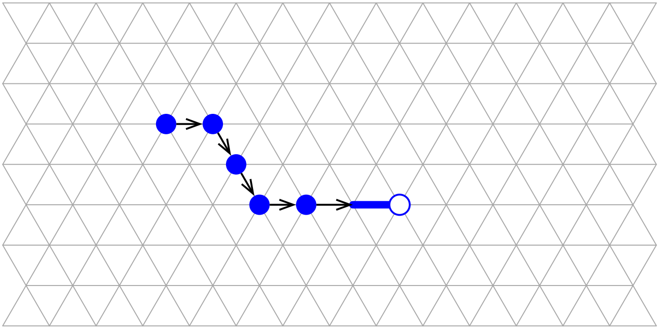
In this case the initial shape has an unbrekable symmetry, and all candidate leaders become leaders.

Basic Locomotion Protocol



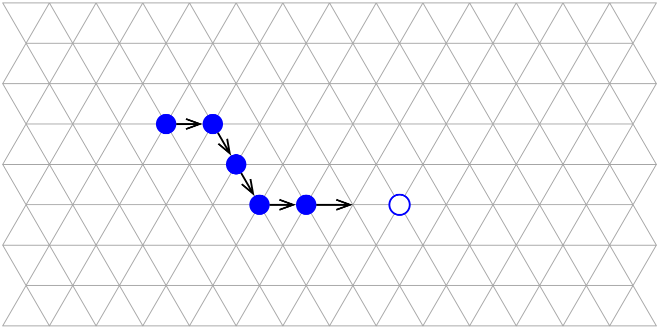
This protocol allows a chain of particles, led by a *pioneer*, to move around without leaving particles behind.

Basic Locomotion Protocol



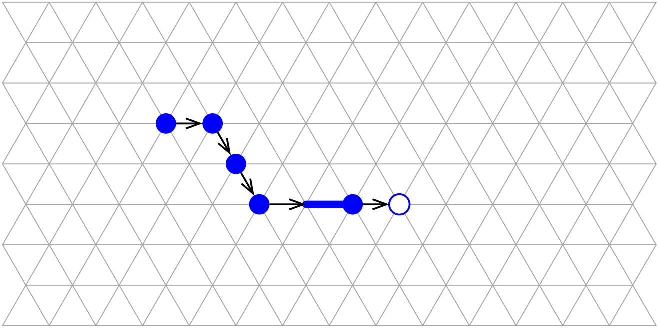
The pioneer expands in some direction and then contracts.

Basic Locomotion Protocol



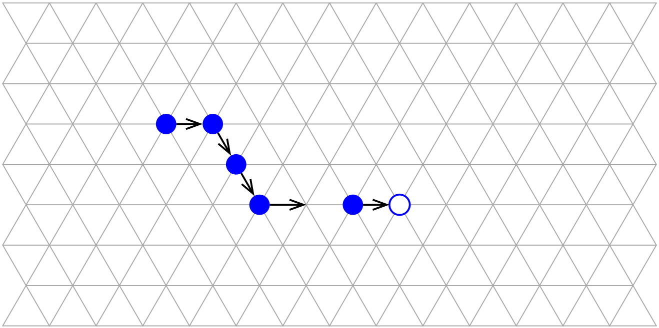
The pioneer expands in some direction and then contracts.

Basic Locomotion Protocol



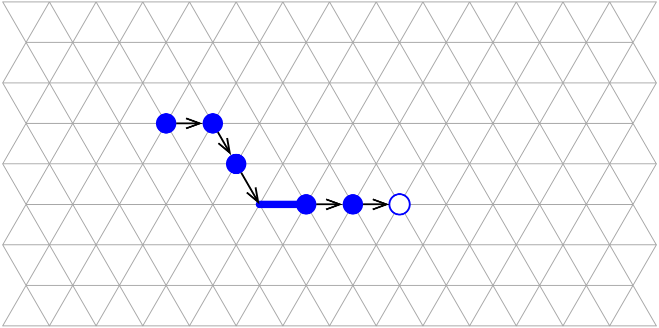
The next particle notices the absence of its parent and moves to the location where it used to be.

Basic Locomotion Protocol



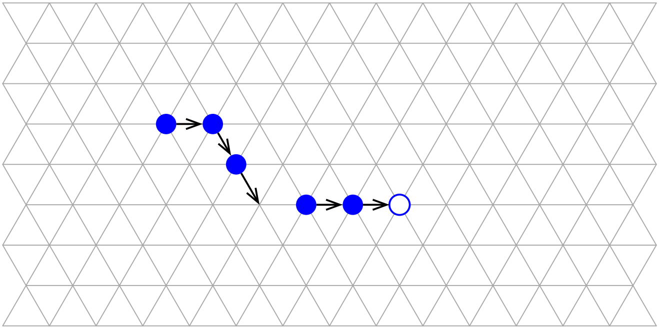
The next particle notices the absence of its parent and moves to the location where it used to be.

Basic Locomotion Protocol



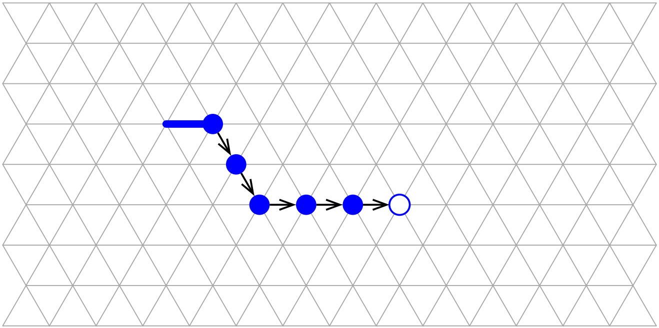
The protocol continues until the last particle has moved.

Basic Locomotion Protocol



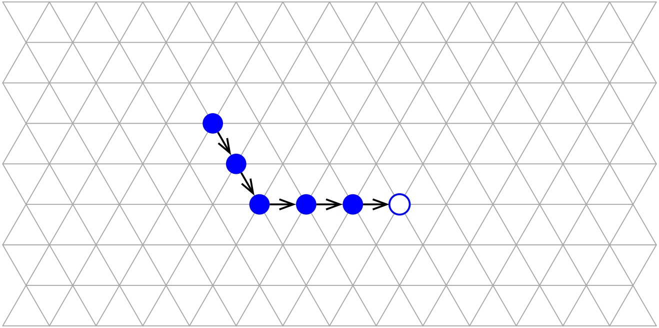
The protocol continues until the last particle has moved.

Basic Locomotion Protocol



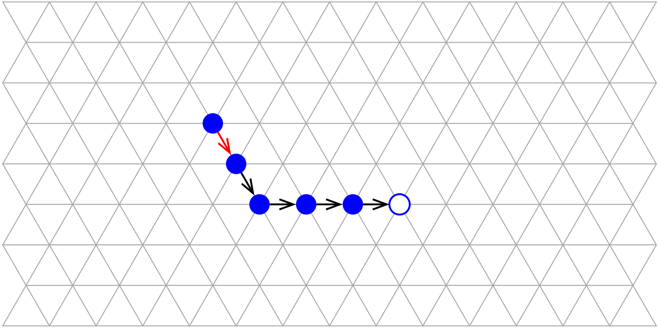
The protocol continues until the last particle has moved.

Basic Locomotion Protocol



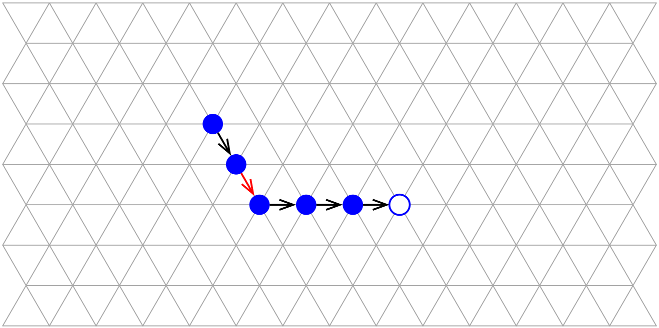
The protocol continues until the last particle has moved.

Basic Locomotion Protocol



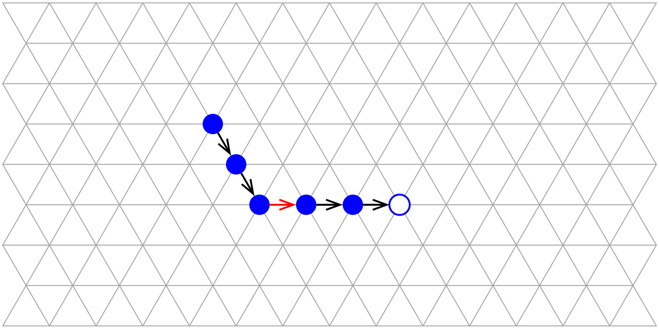
The last particle forwards a *termination* message to the pioneer.

Basic Locomotion Protocol



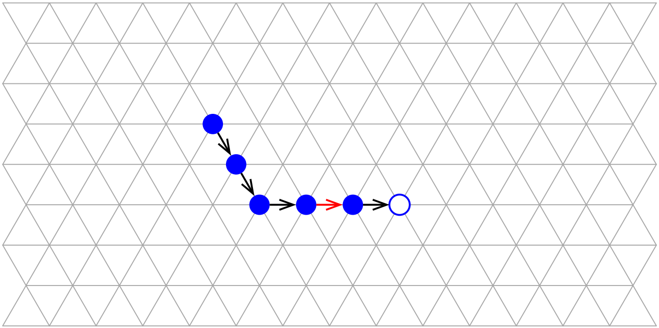
The last particle forwards a *termination* message to the pioneer.

Basic Locomotion Protocol



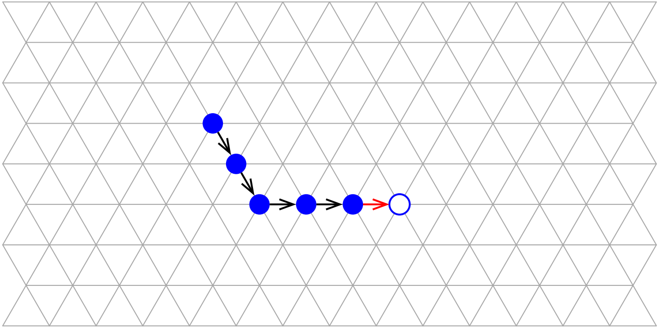
The last particle forwards a *termination* message to the pioneer.

Basic Locomotion Protocol



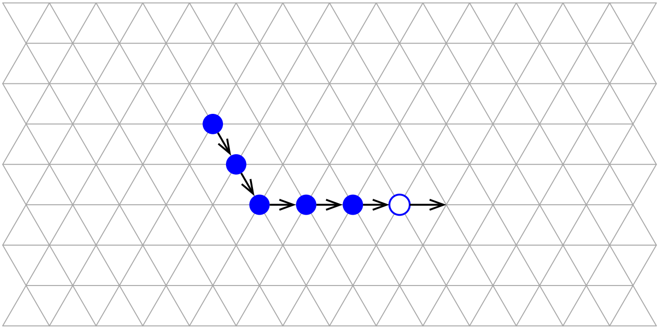
The last particle forwards a *termination* message to the pioneer.

Basic Locomotion Protocol



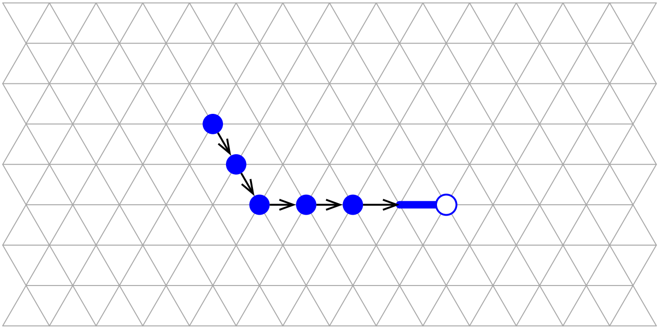
The last particle forwards a *termination* message to the pioneer.

Basic Locomotion Protocol



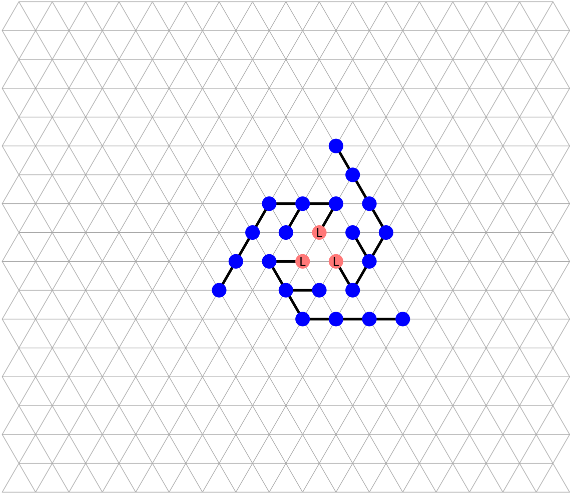
When the pioneer receives the termination message, it moves again, and the protocol repeats.

Basic Locomotion Protocol



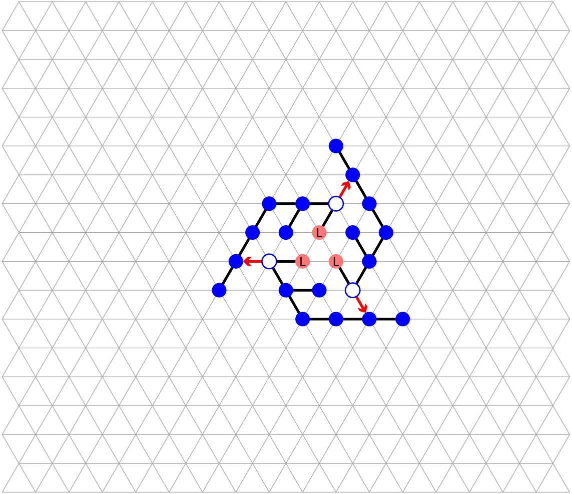
When the pioneer receives the termination message,
it moves again, and the protocol repeats.

Straightening Phase



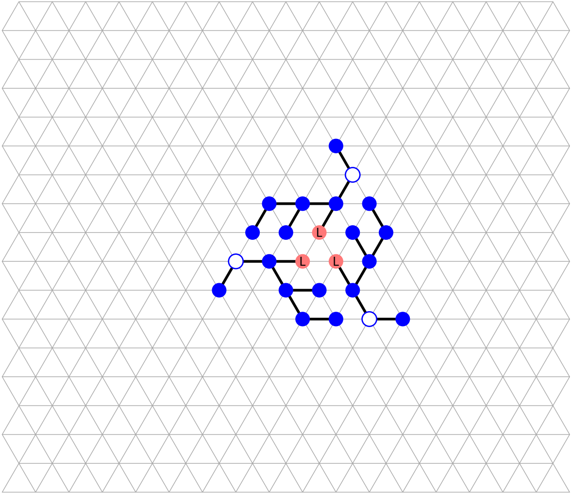
Each leader wants to transform its tree into a line segment.

Straightening Phase



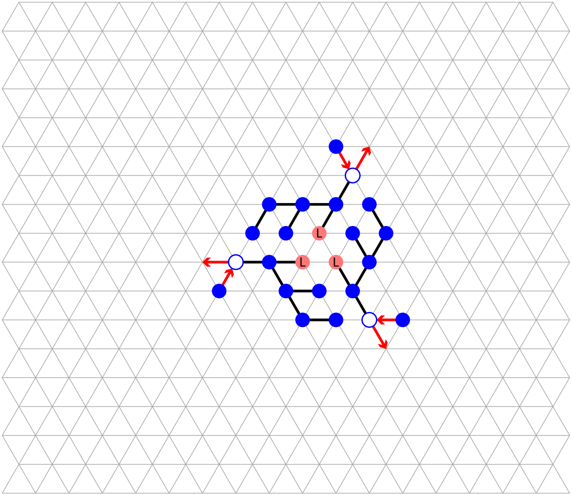
A *pioneer* is sent forth to the designated direction.

Straightening Phase



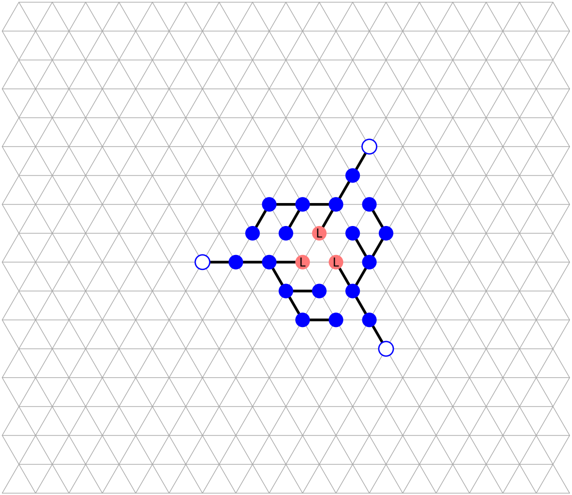
When a pioneer hits another particle,
it becomes its parent and then swaps states with it.

Straightening Phase



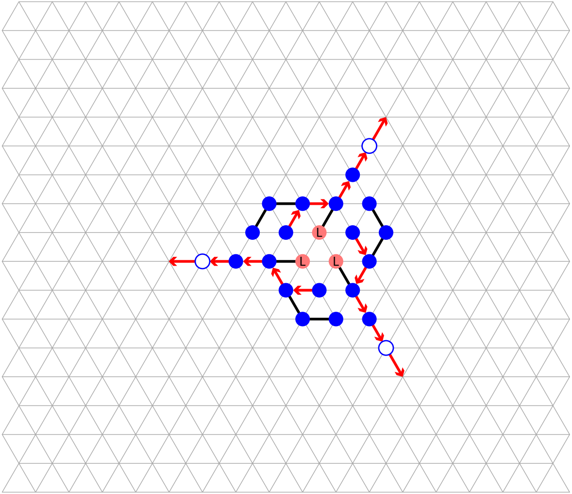
Then each pioneer keeps pulling chains of particles with the basic locomotion protocol, until its tree is straightened.

Straightening Phase



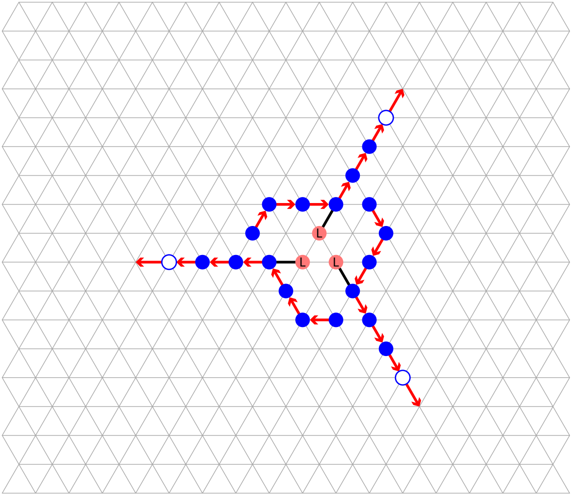
Then each pioneer keeps pulling chains of particles with the basic locomotion protocol, until its tree is straightened.

Straightening Phase



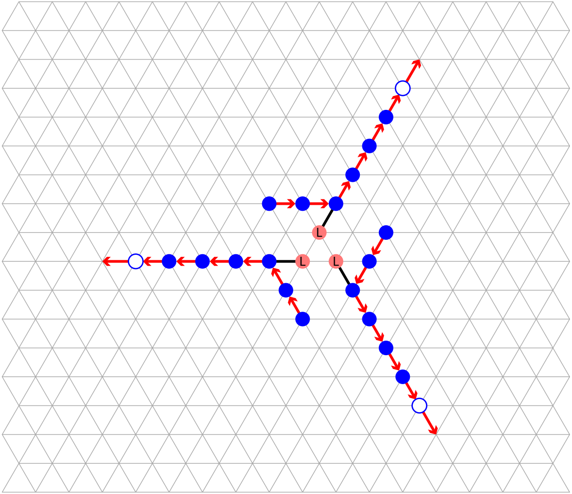
Then each pioneer keeps pulling chains of particles with the basic locomotion protocol, until its tree is straightened.

Straightening Phase



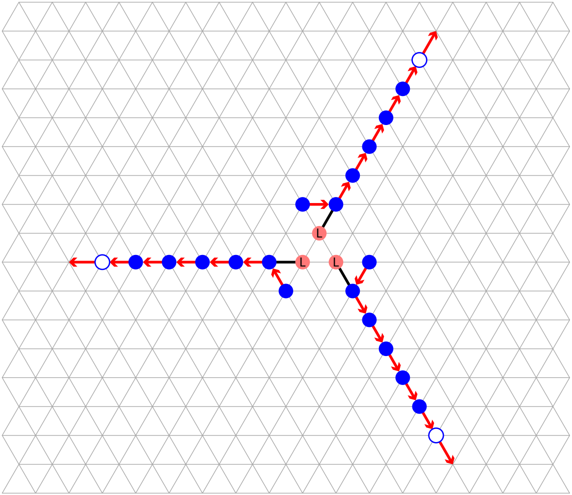
Then each pioneer keeps pulling chains of particles with the basic locomotion protocol, until its tree is straightened.

Straightening Phase



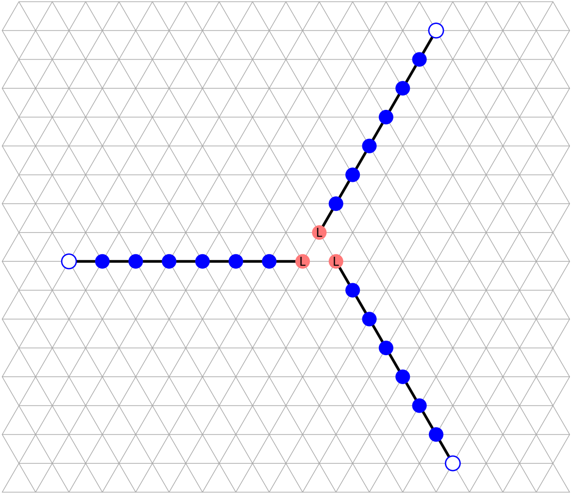
Then each pioneer keeps pulling chains of particles with the basic locomotion protocol, until its tree is straightened.

Straightening Phase



Then each pioneer keeps pulling chains of particles with the basic locomotion protocol, until its tree is straightened.

Straightening Phase



The lines must have the same length, and the leaders communicate with each other to make their pioneers move at the same pace.

Counter Machines

A Counter Machine is a model of computation with:

- some *registers*, each storing a non-negative integer
- a finite *program* consisting of only 3 types of instructions:
  - increment the value stored in a register by 1
  - if the value stored in a register is positive, decrement it by 1
  - test the value of a register and branch if it is 0

Counter Machines

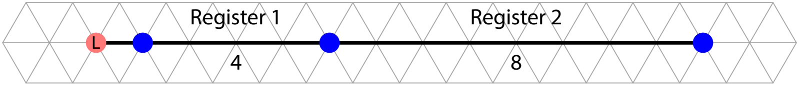
A Counter Machine is a model of computation with:

- some *registers*, each storing a non-negative integer
- a finite *program* consisting of only 3 types of instructions:
  - <u>increment</u> the value stored in a register by 1
  - if the value stored in a register is positive, <u>decrement</u> it by 1
  - <u>test</u> the value of a register and <u>branch</u> if it is 0

Theorem (Minsky, 1967)

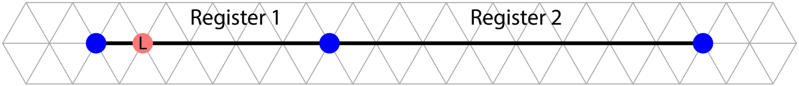
Any Turing machine can be simulated by a Counter Machine with only 2 registers, the first of which initially contains the input.

Simulating a Counter Machine with 2 Registers



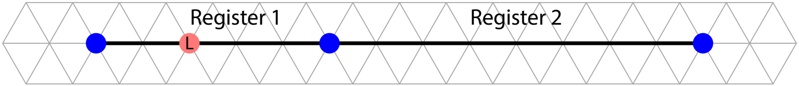
A Counter Machine with 2 registers can be simulated by 4 particles: a *leader*, which executes the program, and 3 particles whose distances correspond to the values stored in the 2 registers.

Simulating a Counter Machine with 2 Registers



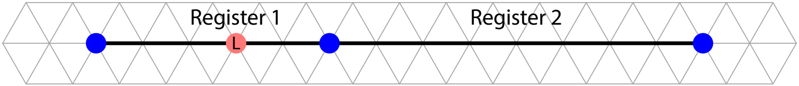
If the leader has to increment the value of the first register, it pulls the last two particles to the right by one step, and then goes back to its original position.

Simulating a Counter Machine with 2 Registers



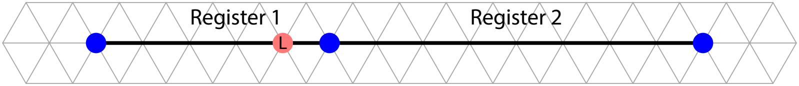
If the leader has to increment the value of the first register,
it pulls the last two particles to the right by one step,
and then goes back to its original position.

Simulating a Counter Machine with 2 Registers



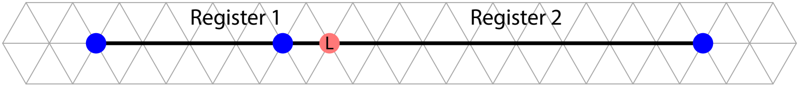
If the leader has to increment the value of the first register,
it pulls the last two particles to the right by one step,
and then goes back to its original position.

Simulating a Counter Machine with 2 Registers



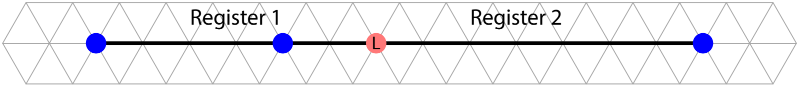
If the leader has to increment the value of the first register, it pulls the last two particles to the right by one step, and then goes back to its original position.

Simulating a Counter Machine with 2 Registers



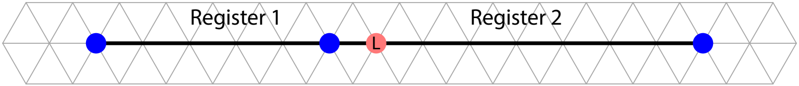
If the leader has to increment the value of the first register, it pulls the last two particles to the right by one step, and then goes back to its original position.

Simulating a Counter Machine with 2 Registers



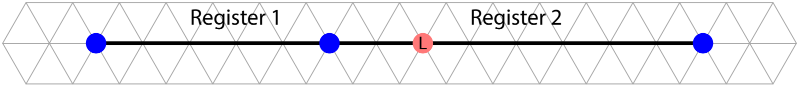
If the leader has to increment the value of the first register, it pulls the last two particles to the right by one step, and then goes back to its original position.

Simulating a Counter Machine with 2 Registers



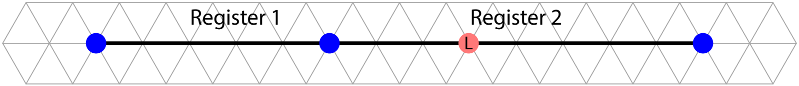
If the leader has to increment the value of the first register, it pulls the last two particles to the right by one step, and then goes back to its original position.

Simulating a Counter Machine with 2 Registers



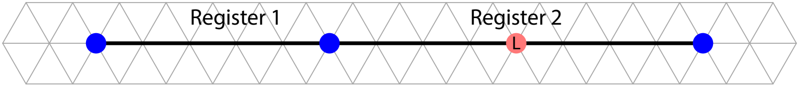
If the leader has to increment the value of the first register,
it pulls the last two particles to the right by one step,
and then goes back to its original position.

Simulating a Counter Machine with 2 Registers



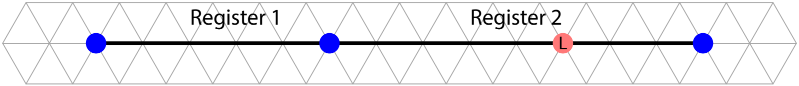
If the leader has to increment the value of the first register, it pulls the last two particles to the right by one step, and then goes back to its original position.

Simulating a Counter Machine with 2 Registers



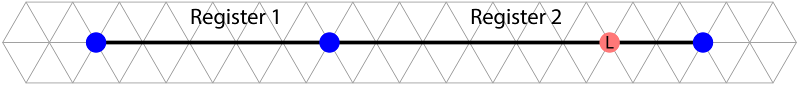
If the leader has to increment the value of the first register,
it pulls the last two particles to the right by one step,
and then goes back to its original position.

Simulating a Counter Machine with 2 Registers



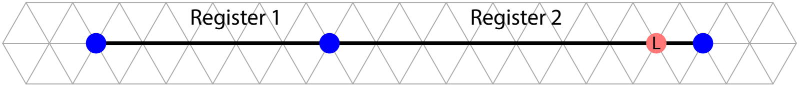
If the leader has to increment the value of the first register,
it pulls the last two particles to the right by one step,
and then goes back to its original position.

Simulating a Counter Machine with 2 Registers



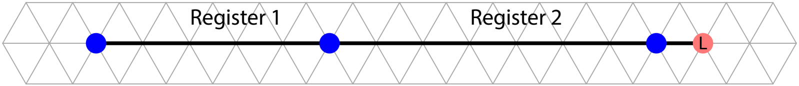
If the leader has to increment the value of the first register,
it pulls the last two particles to the right by one step,
and then goes back to its original position.

Simulating a Counter Machine with 2 Registers



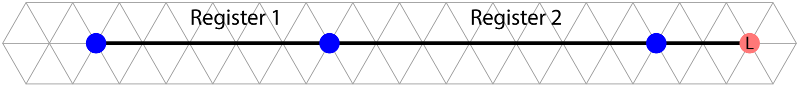
If the leader has to increment the value of the first register,
it pulls the last two particles to the right by one step,
and then goes back to its original position.

Simulating a Counter Machine with 2 Registers



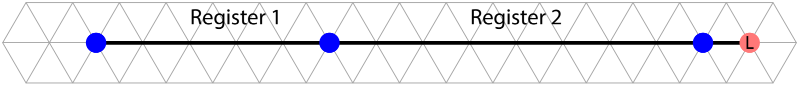
If the leader has to increment the value of the first register, it pulls the last two particles to the right by one step, and then goes back to its original position.

Simulating a Counter Machine with 2 Registers



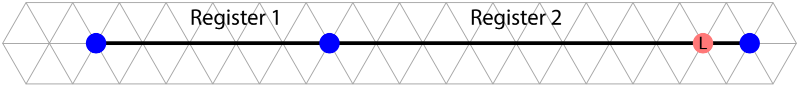
If the leader has to increment the value of the first register,
it pulls the last two particles to the right by one step,
and then goes back to its original position.

Simulating a Counter Machine with 2 Registers



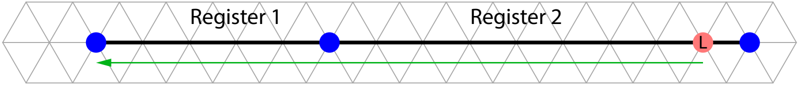
If the leader has to increment the value of the first register, it pulls the last two particles to the right by one step, and then goes back to its original position.

Simulating a Counter Machine with 2 Registers



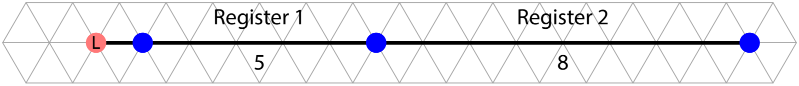
If the leader has to increment the value of the first register,
it pulls the last two particles to the right by one step,
and then goes back to its original position.

Simulating a Counter Machine with 2 Registers



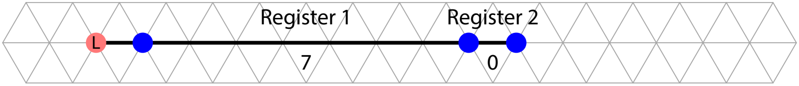
If the leader has to increment the value of the first register, it pulls the last two particles to the right by one step, and then goes back to its original position.

Simulating a Counter Machine with 2 Registers



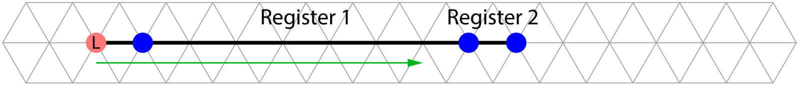
If the leader has to increment the value of the first register,
it pulls the last two particles to the right by one step,
and then goes back to its original position.

Simulating a Counter Machine with 2 Registers



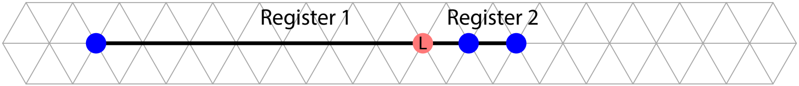
If the leader has to test if the value of the second register is 0, it reaches the second-to-last particle and exchanges messages with it, asking if the last particle is adjacent to it.

Simulating a Counter Machine with 2 Registers



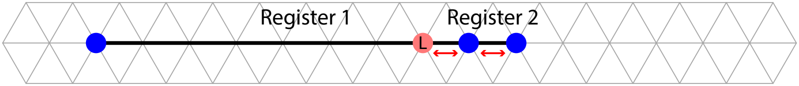
If the leader has to test if the value of the second register is 0, it reaches the second-to-last particle and exchanges messages with it, asking if the last particle is adjacent to it.

Simulating a Counter Machine with 2 Registers



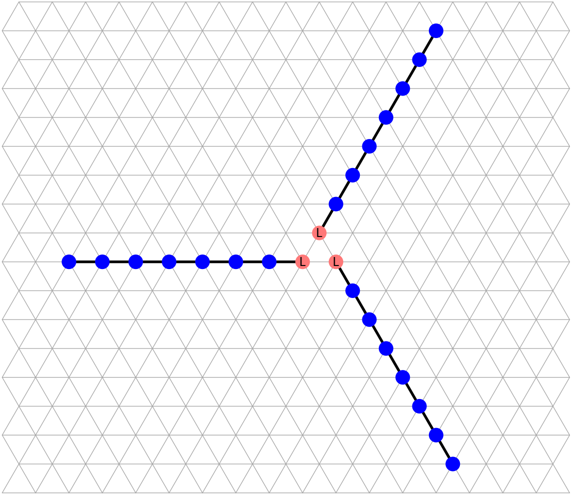
If the leader has to test if the value of the second register is 0, it reaches the second-to-last particle and exchanges messages with it, asking if the last particle is adjacent to it.

Simulating a Counter Machine with 2 Registers



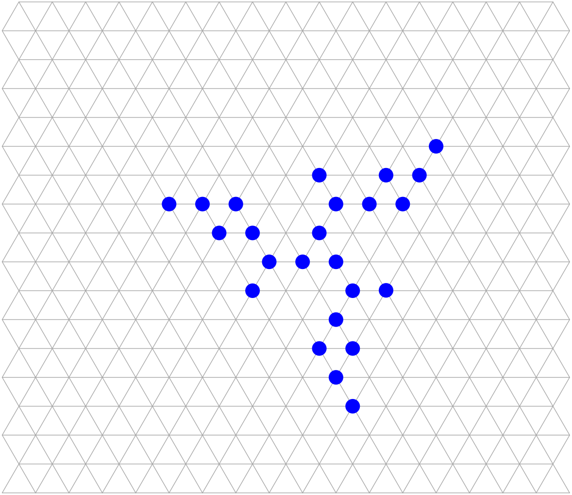
If the leader has to test if the value of the second register is 0, it reaches the second-to-last particle and exchanges messages with it, asking if the last particle is adjacent to it.

Shape-Formation Phase



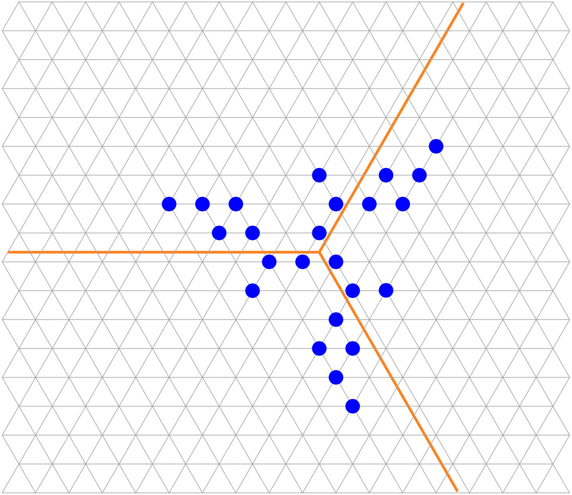
If $k > 1$ leaders have been elected in the previous phases, it means that the initial shape has an unbreakable k-fold symmetry.

Shape-Formation Phase



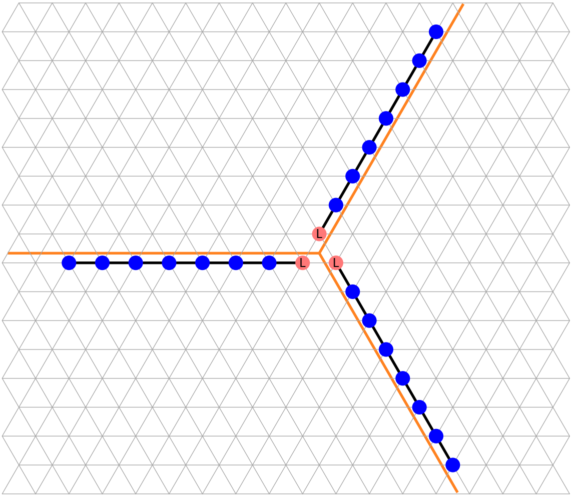
Hence, we may assume that also the shape to be formed has the same k-fold symmetry.

Shape-Formation Phase



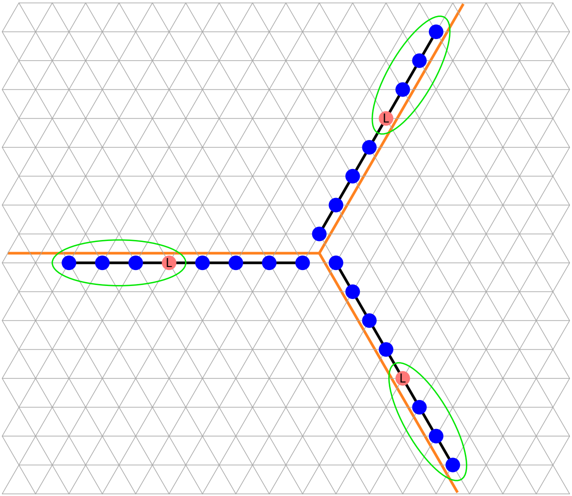
The plane is partitioned into k sectors, and each leader is tasked with forming the part of the shape that falls into its sector.

Shape-Formation Phase



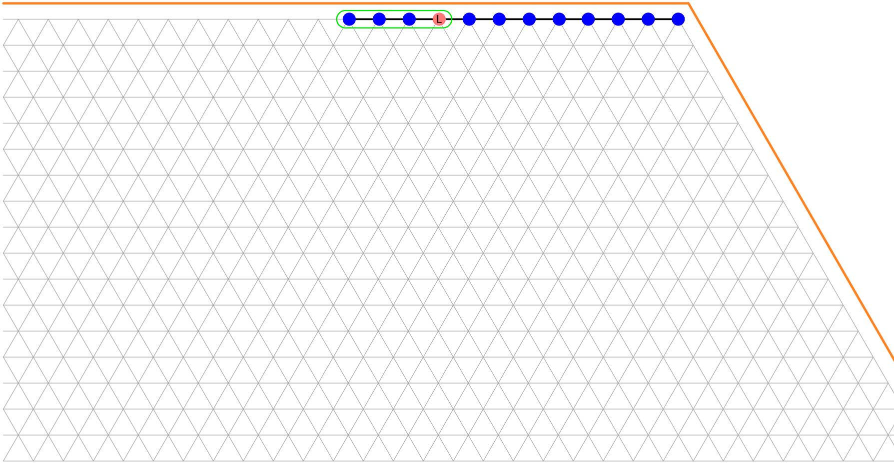
The plane is partitioned into k sectors, and each leader is tasked with forming the part of the shape that falls into its sector.

Shape-Formation Phase



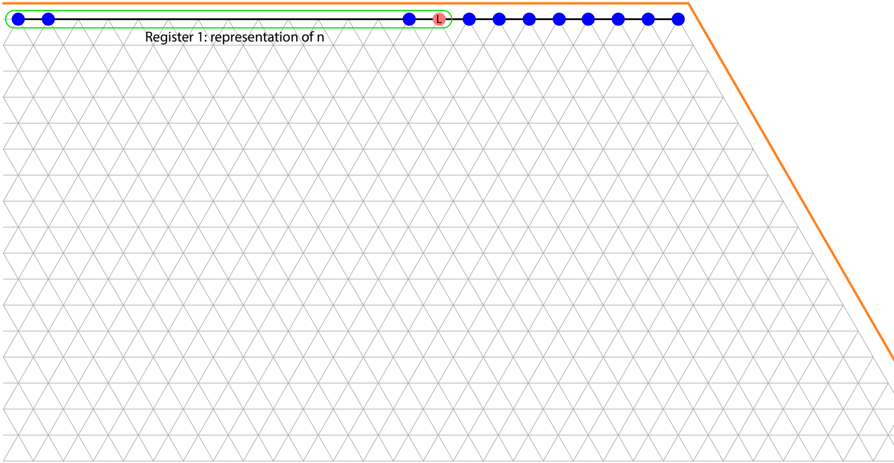
Assume there is an algorithm that, given n, generates the points of the shape. Let each leader simulate a Counter Machine for that algorithm.

Shape-Formation Phase



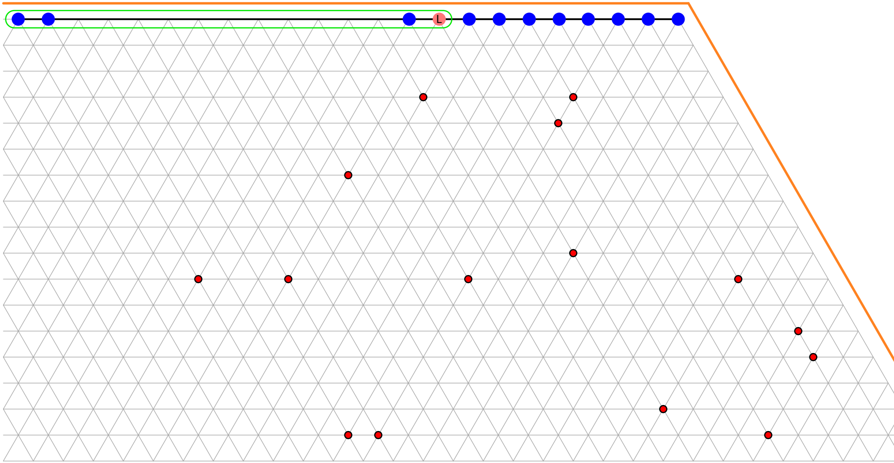
The leader takes position at the beginning of the simulated Counter Machine.

Shape-Formation Phase



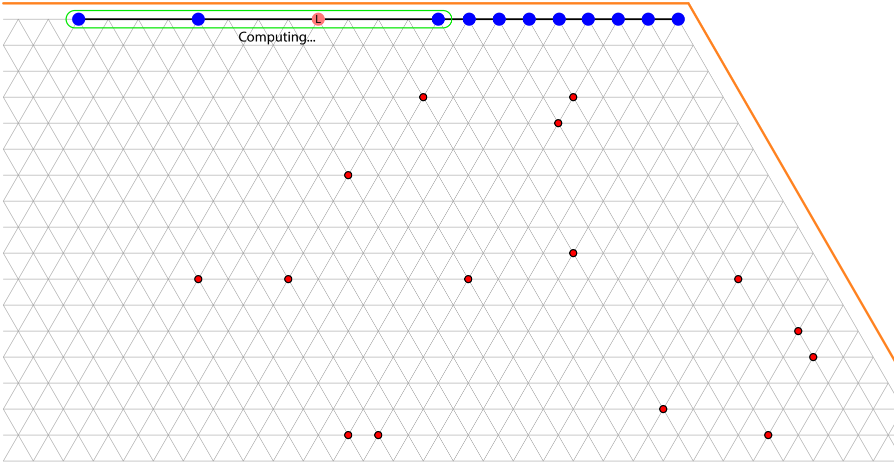
By scanning the previous part of the chain, it constructs a representation of n in the first register, which serves as the input.

Shape-Formation Phase



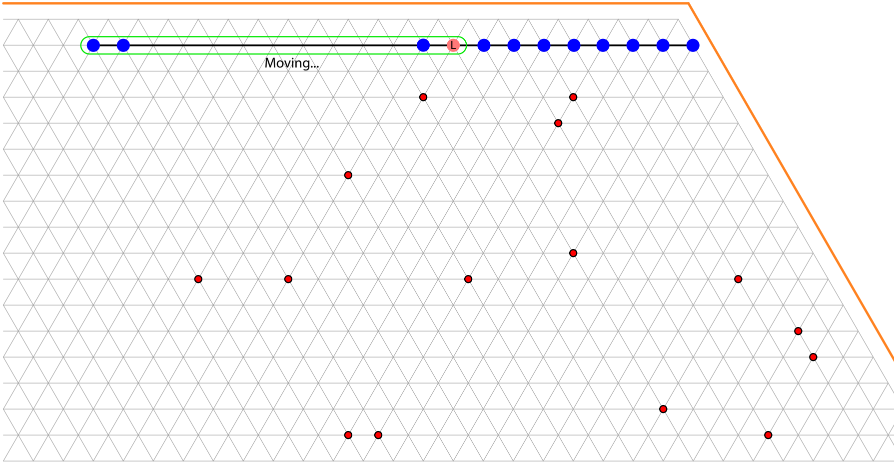
The simulated Counter Machine will generate all the points of the shape and the sequence of moves necessary to reach them.

Shape-Formation Phase



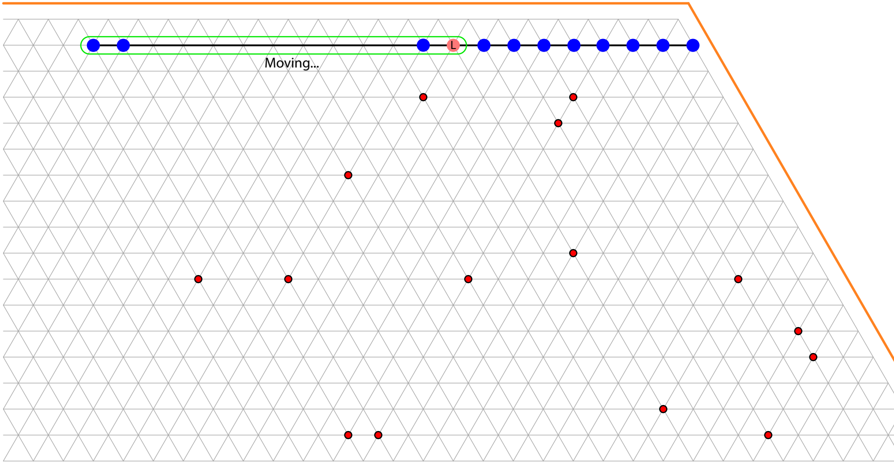
The simulated Counter Machine computes the first point of the shape, while the rest of the chain does not move.

Shape-Formation Phase



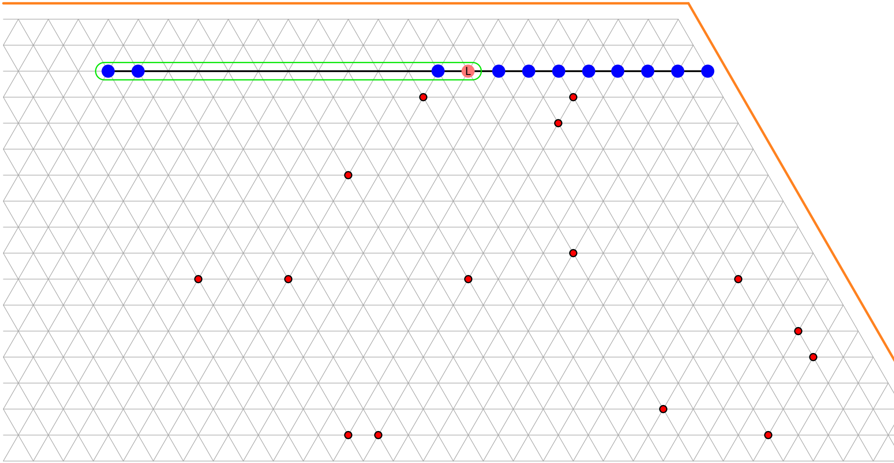
When the Counter Machine has finished, the value of the first register indicates that the chain has to move in some direction.

Shape-Formation Phase



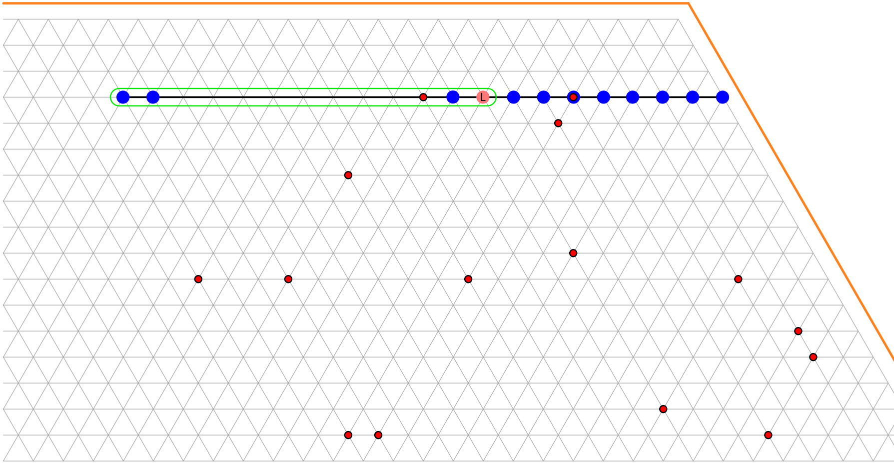
(The movement of the whole chain is coordinated by the leader, and takes place one particle at a time.)

Shape-Formation Phase



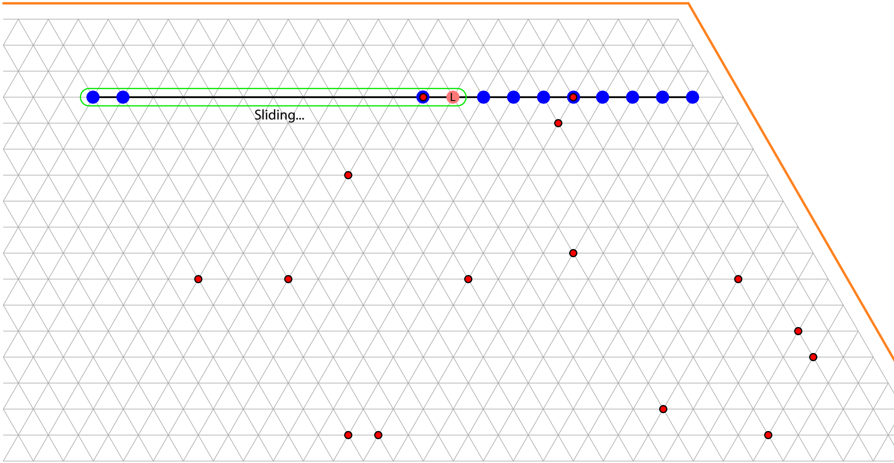
The Counter Machine computes the next movement,
and the whole chain moves as soon as the computation is finished.

Shape-Formation Phase



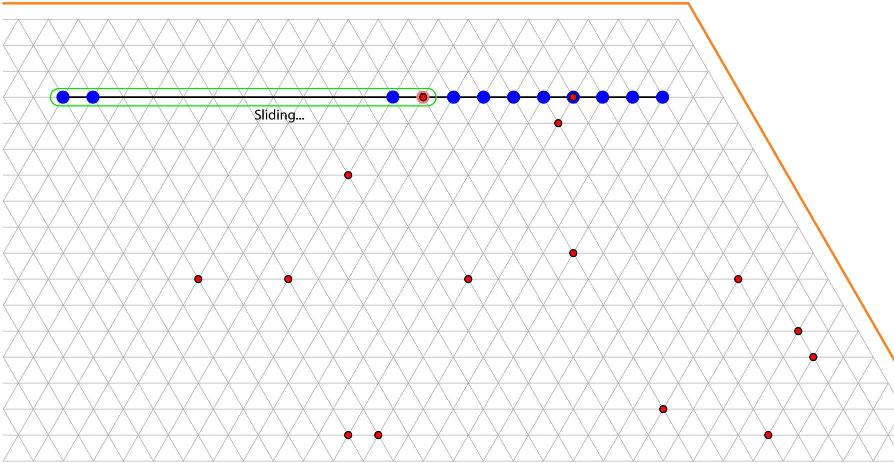
The Counter Machine computes the next movement,
and the whole chain moves as soon as the computation is finished.

Shape-Formation Phase



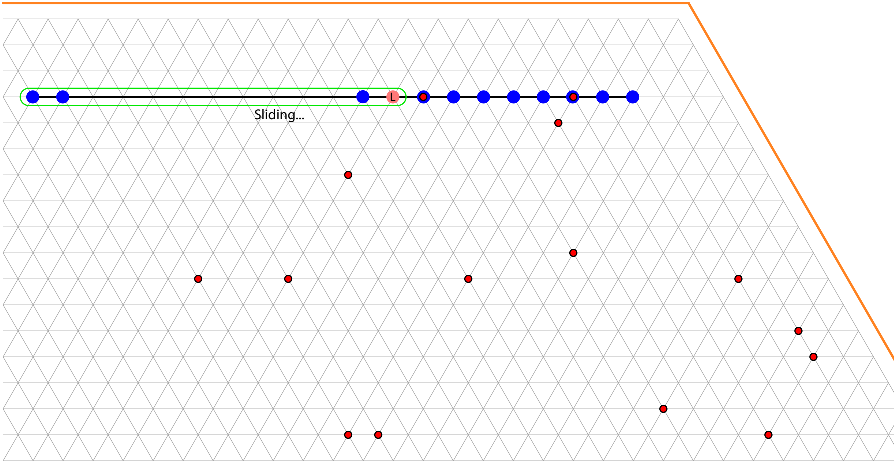
When the chain is on the same line as the first point of the shape, it slides until the last particle of the chain coincides with the point.

Shape-Formation Phase



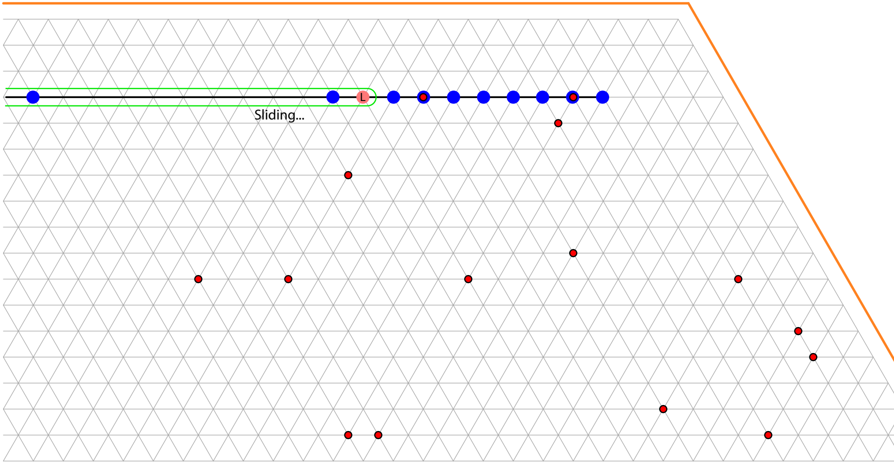
When the chain is on the same line as the first point of the shape, it slides until the last particle of the chain coincides with the point.

Shape-Formation Phase



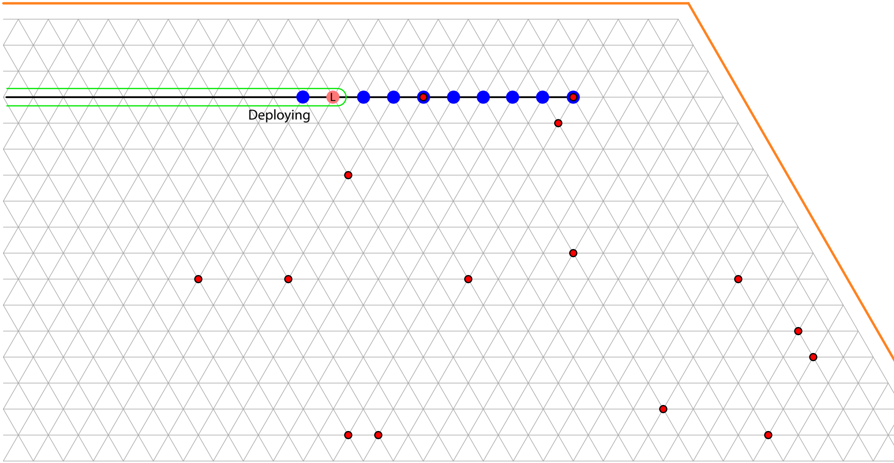
When the chain is on the same line as the first point of the shape, it slides until the last particle of the chain coincides with the point.

Shape-Formation Phase



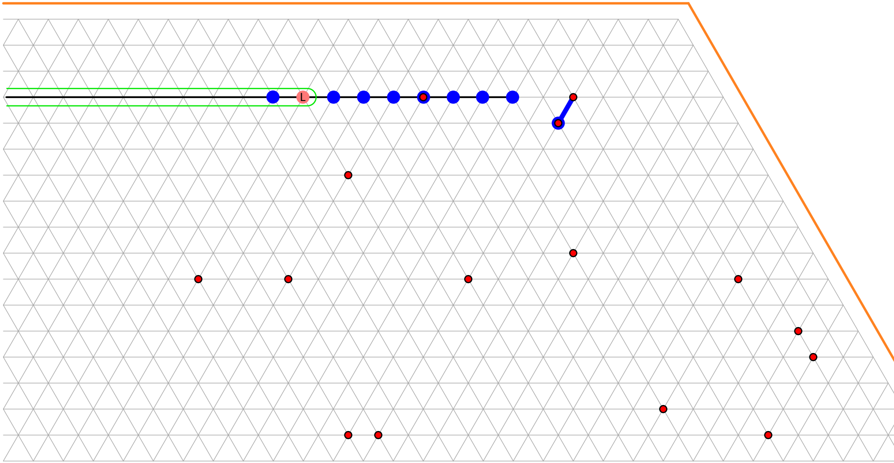
When the chain is on the same line as the first point of the shape, it slides until the last particle of the chain coincides with the point.

Shape-Formation Phase



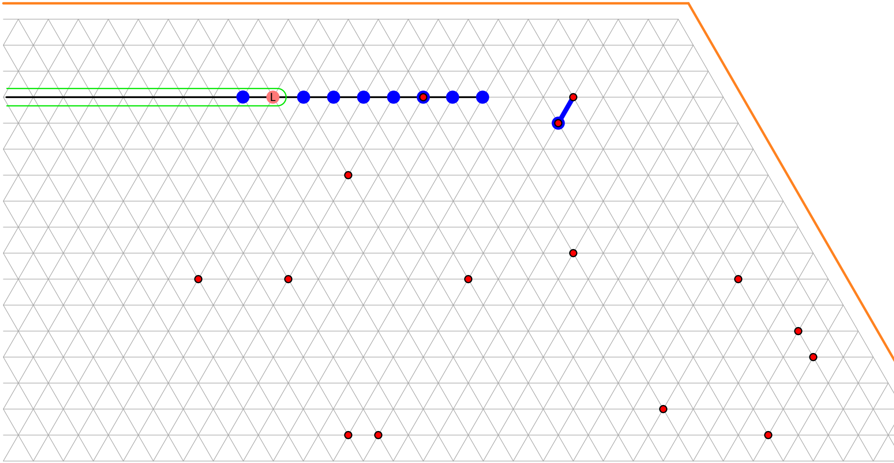
A message is forwarded to the last particle, telling it to stay there, and perhaps expand in some direction to cover two points.

Shape-Formation Phase



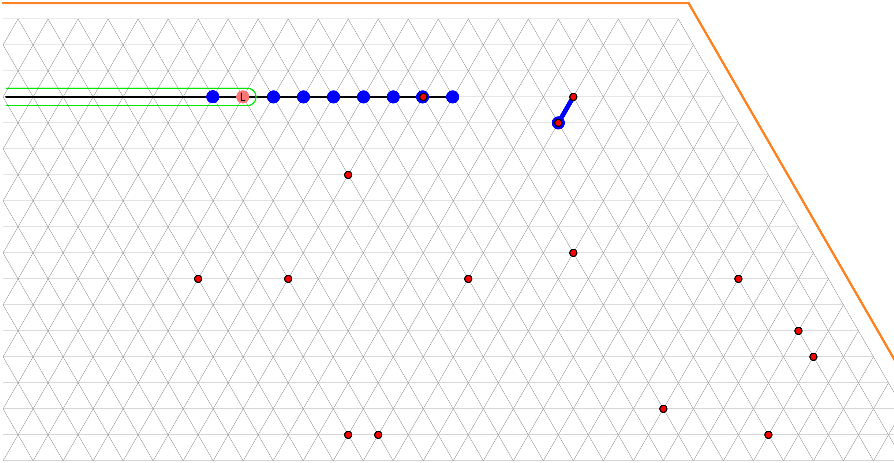
A message is forwarded to the last particle, telling it to stay there, and perhaps expand in some direction to cover two points.

Shape-Formation Phase



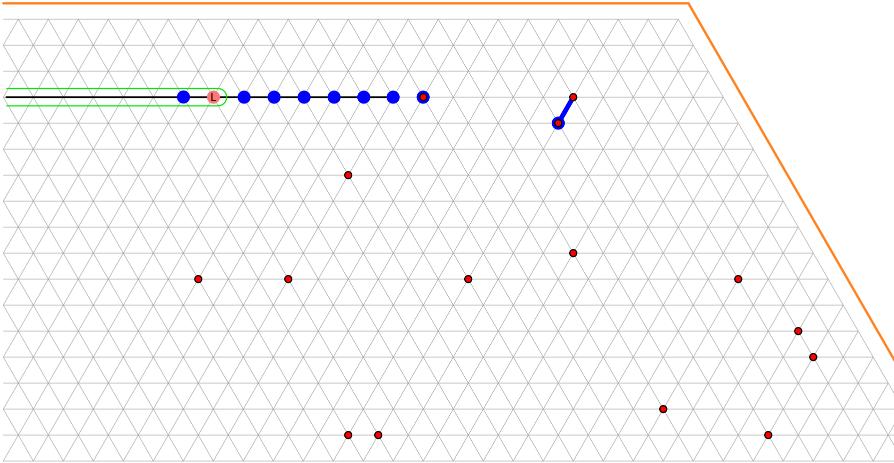
The protocol proceeds in the same fashion with the other points of the shape.

Shape-Formation Phase



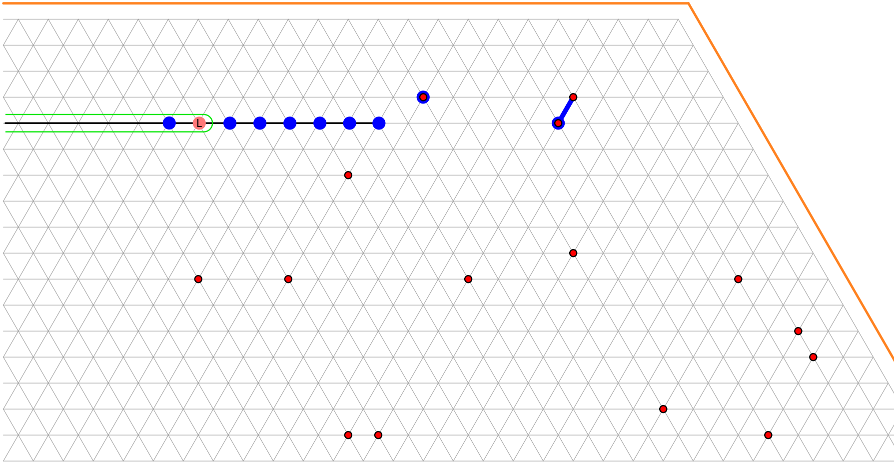
The protocol proceeds in the same fashion with the other points of the shape.

Shape-Formation Phase



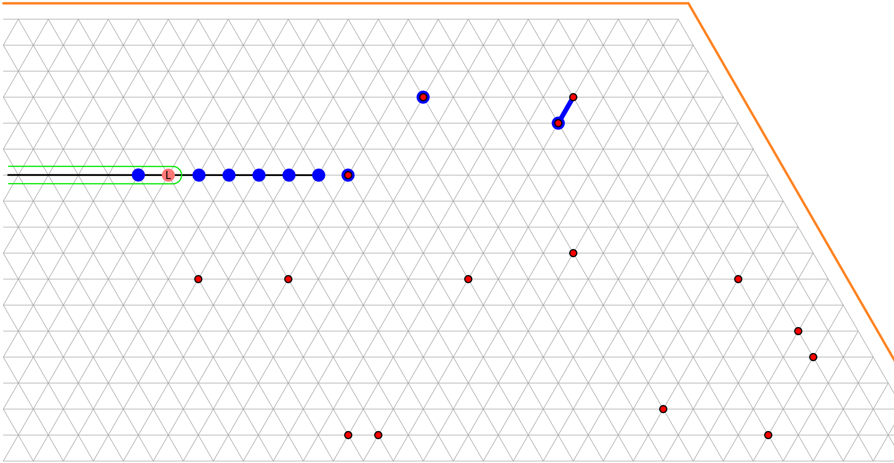
The protocol proceeds in the same fashion with the other points of the shape.

Shape-Formation Phase



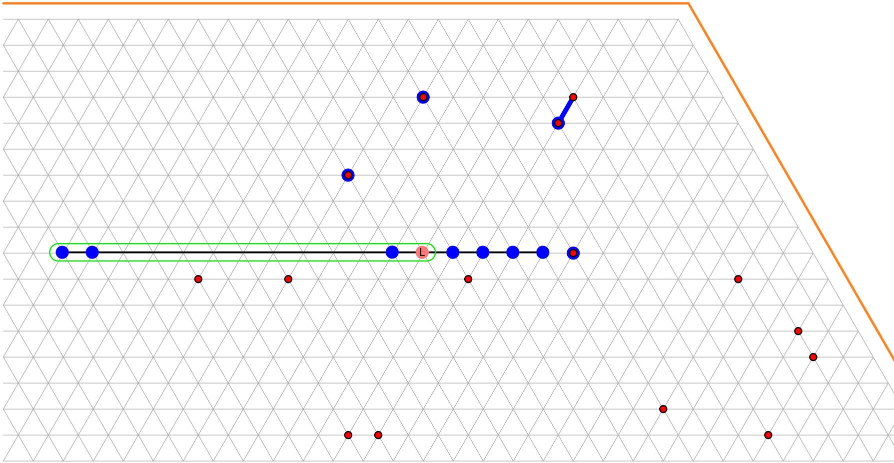
The protocol proceeds in the same fashion with the other points of the shape.

Shape-Formation Phase



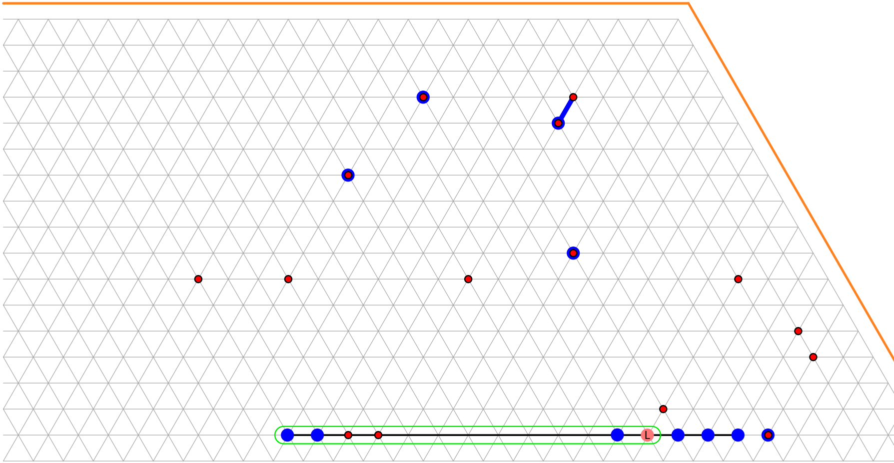
The protocol proceeds in the same fashion with the other points of the shape.

Shape-Formation Phase



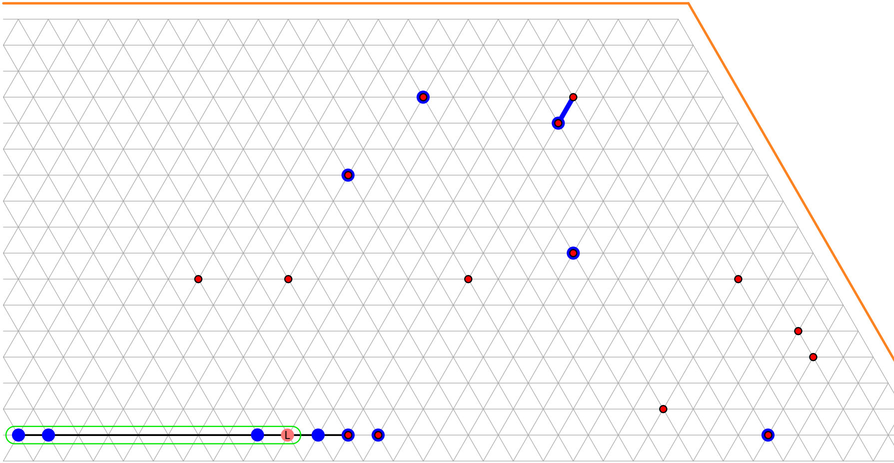
The protocol proceeds in the same fashion with the other points of the shape.

Shape-Formation Phase



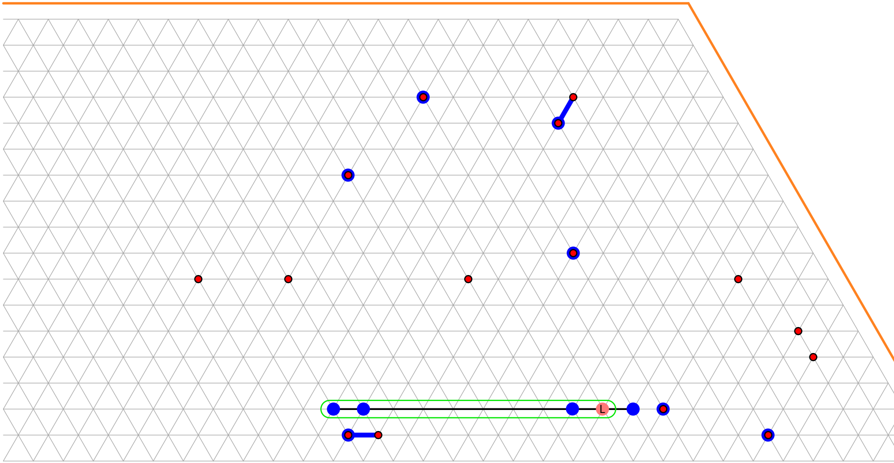
The protocol proceeds in the same fashion with the other points of the shape.

Shape-Formation Phase



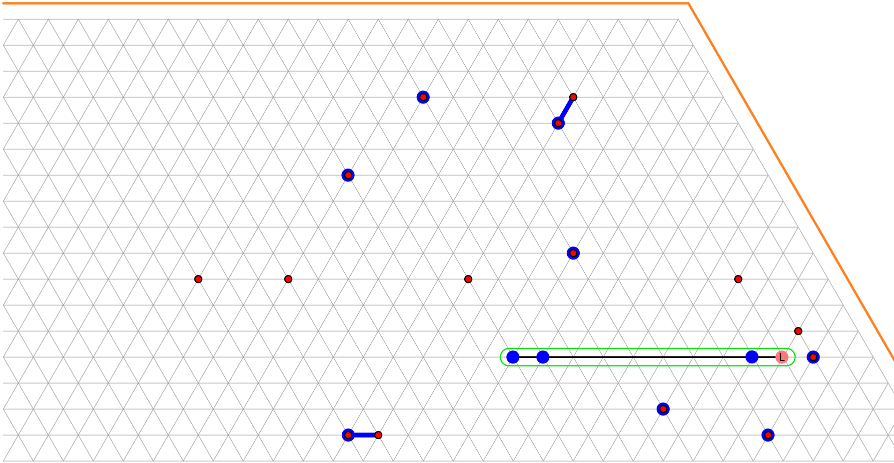
The protocol proceeds in the same fashion with the other points of the shape.

Shape-Formation Phase



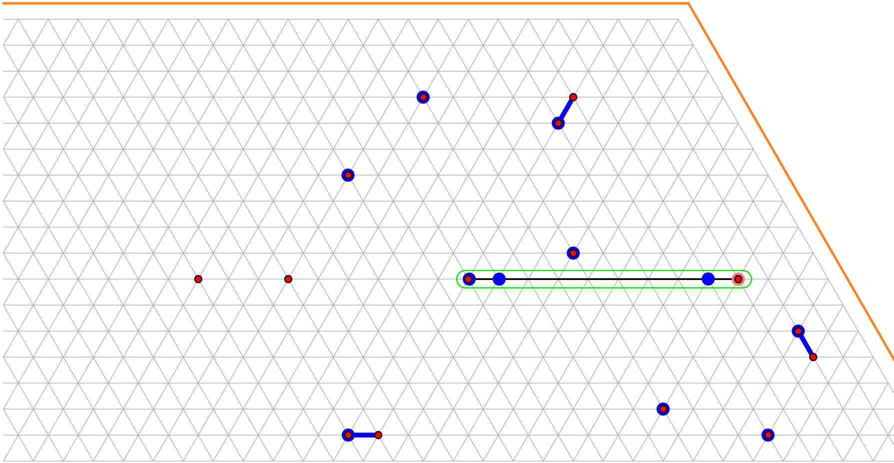
The protocol proceeds in the same fashion with the other points of the shape.

Shape-Formation Phase



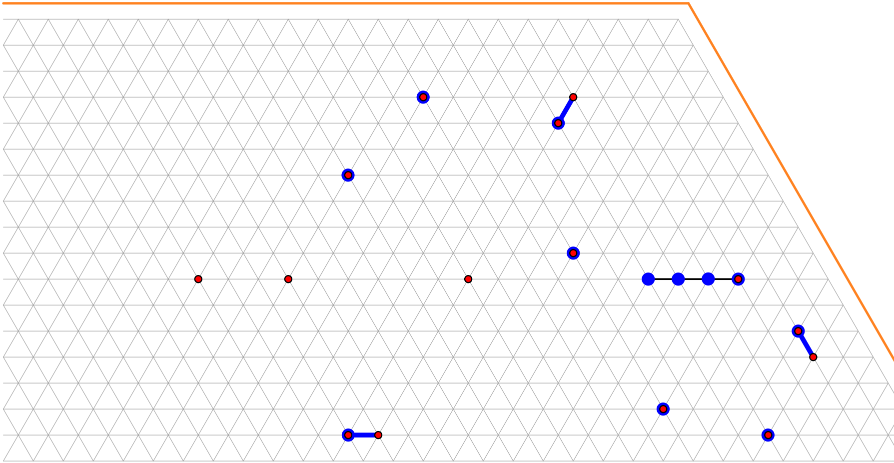
The protocol proceeds in the same fashion with the other points of the shape.

Shape-Formation Phase



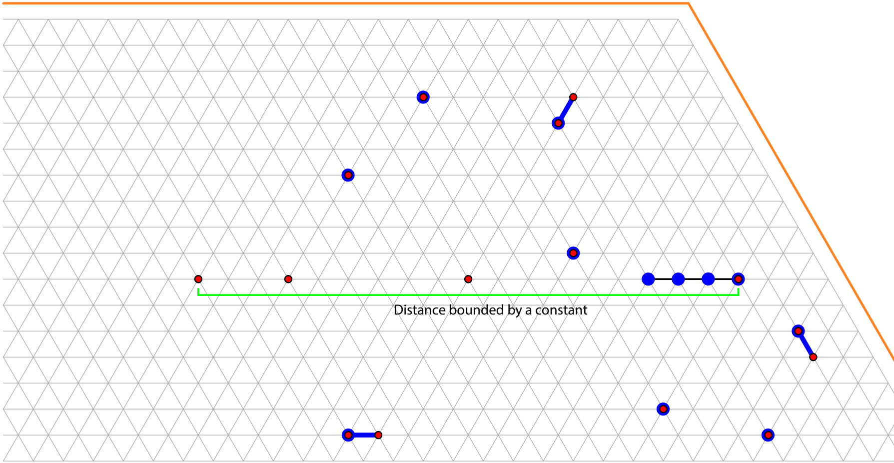
The algorithm ensures that the last 4 points of the shape are “in the same neighborhood”.

Shape-Formation Phase



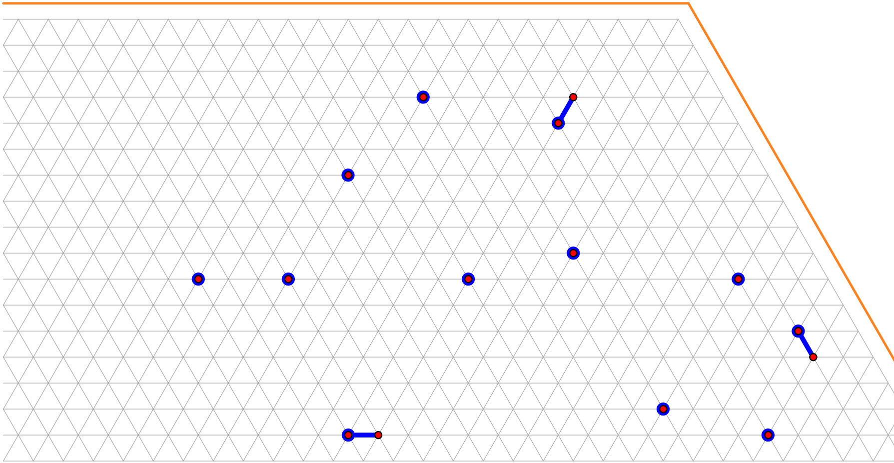
When the leader is on the first of these 4 points,
it makes the Counter Machine contract, erasing the registers.

Shape-Formation Phase



Assuming that the distance of the other 3 points is bounded by a constant, the particles can reach them using constant memory.

Shape-Formation Phase



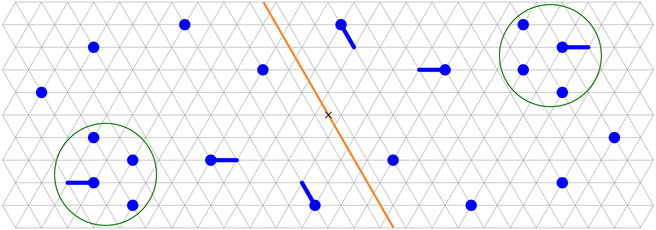
Assuming that the distance of the other 3 points is bounded by a constant, the particles can reach them using constant memory.

Dismantling the Mobile Counter Machines

Also, each mobile Counter Machine used to form the shape has to be <u>dismantled</u> when it has finished computing:

Assumption

For each scaled-up copy S_n of the final shape, there exists a configuration C_f of n particles that forms S_n (such that C_f is unbreakably k-symmetric if S_n has to be formed from an unbreakably k-symmetric initial configuration) and, for each symmetric component C'_f of C_f, there exists a ball of diameter independent of n that contains at least four particles of C'_f.



Results

Theorem

Under the previous Assumption, any Turing-computable shape is formable from any simply connected initial configuration.

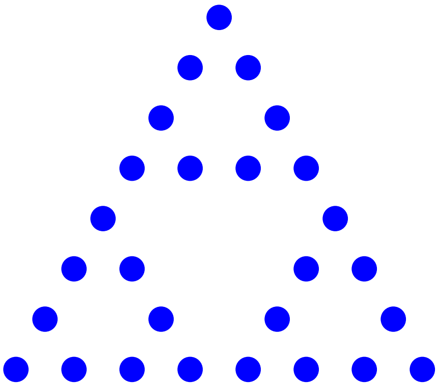
Only very sparse shapes fail to satisfy the Assumption.
In particular, connected shapes abundantly satisfy it.

Corollary

A necessary and sufficient condition for a connected Turing-computable shape to be formable from a simply connected initial configuration is that, if the initial configuration is unbreakably k-symmetric, then also the corresponding scaled-up copy of the shape is unbreakably k-symmetric.

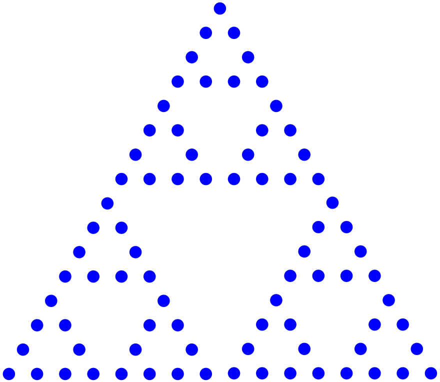
Open problem: can we keep the system connected if the shape to be formed is connected?

Fractal and Curved Shapes



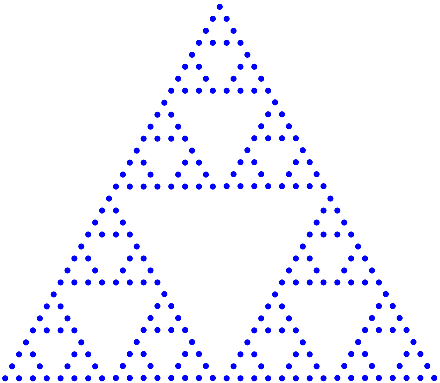
This protocol allows the system to form shapes that scale up like fractals (e.g., the Sierpinski triangle), as well as curved objects, which are better approximated as the number of particles increases.

Fractal and Curved Shapes



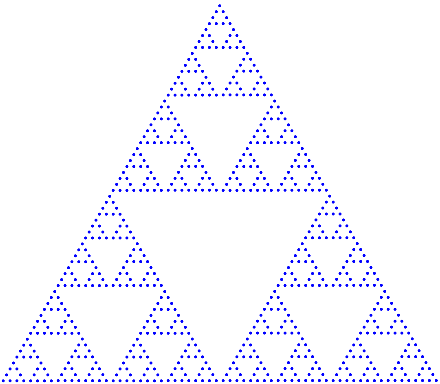
This protocol allows the system to form shapes that scale up like fractals (e.g., the Sierpinski triangle), as well as curved objects, which are better approximated as the number of particles increases.

Fractal and Curved Shapes



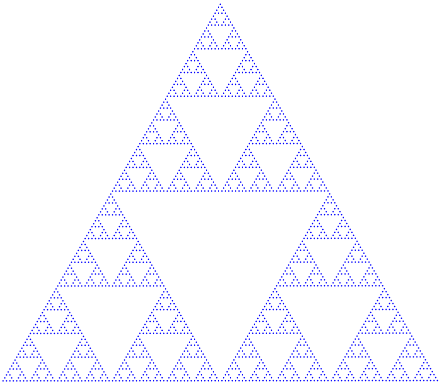
This protocol allows the system to form shapes that scale up like fractals (e.g., the Sierpinski triangle), as well as curved objects, which are better approximated as the number of particles increases.

Fractal and Curved Shapes



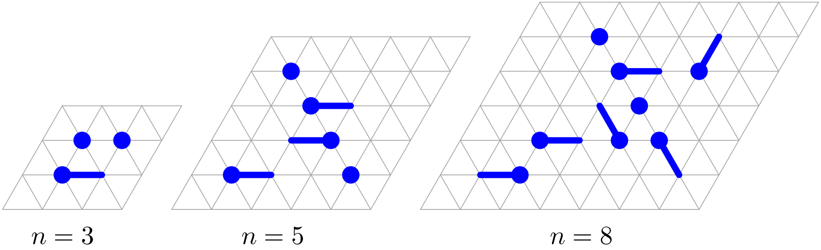
This protocol allows the system to form shapes that scale up like fractals (e.g., the Sierpinski triangle), as well as curved objects, which are better approximated as the number of particles increases.

Fractal and Curved Shapes



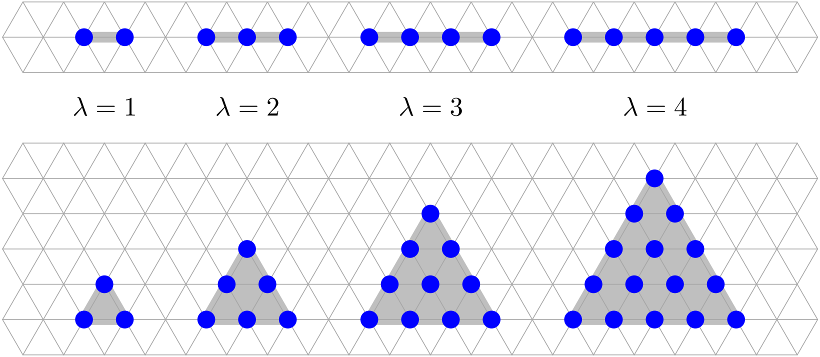
This protocol allows the system to form shapes that scale up like fractals (e.g., the Sierpinski triangle), as well as curved objects, which are better approximated as the number of particles increases.

General Computable Shapes



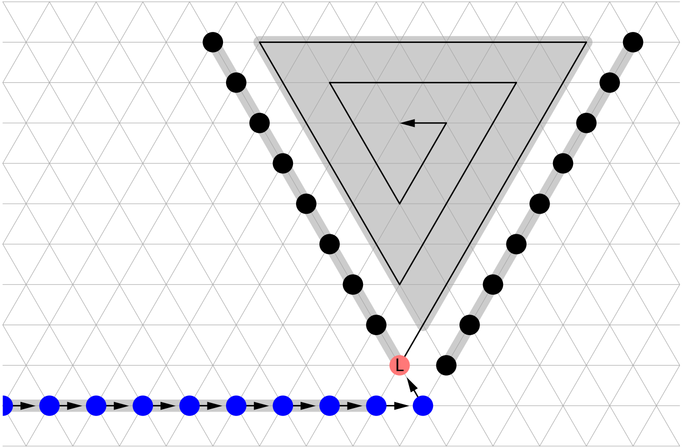
More generally, the protocol only requires the existence of a computable function that, on input n, produces a configuration of n particles that forms a suitably scaled-up copy of the final shape.

Forming Segments and Full Triangles



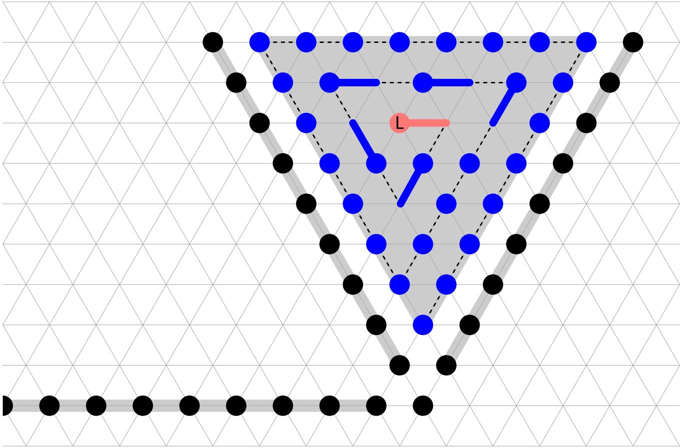
If the shape to be formed is made up of "blocks" that scale up like segments and full triangles, the last phase can be optimized.

Forming Segments and Full Triangles



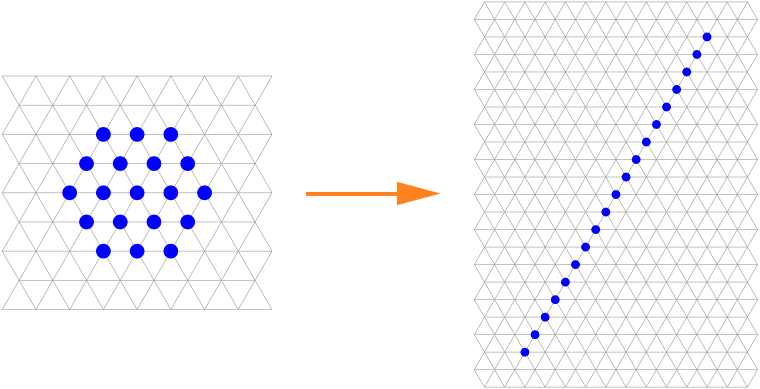
For this special case, there is a protocol that allows to form the shape in $O(n^2)$ total moves.

Forming Segments and Full Triangles



For this special case, there is a protocol that allows to form the shape in $O(n^2)$ total moves.

Matching Lower Bound



This example shows that $O(n^2)$ total moves are optimal.

Further Results

Theorem

A necessary and sufficient condition for a connected Turing-computable shape to be formable from a simply connected initial configuration is that, if the initial configuration is unbreakably k-symmetric, then also the corresponding scaled-up copy of the shape is unbreakably k-symmetric.

The running time depends on how fast a Turing machine can compute the shape. However, there is a special case:

Theorem

If the shape to be formed consists only of segments and full triangles, the system can form it in $O(n^2)$ moves (optimally) and $O(n^2)$ rounds.

Open problem: are $O(n^2)$ rounds optimal?

Shortcomings of the Naive Approach

The previous approach has at least two major **problems**:

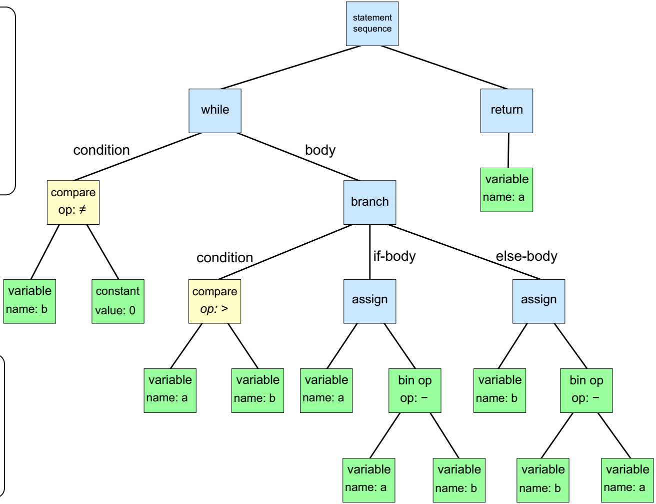
- The algorithm is vulnerable to **crash failures**: if the leader malfunctions, the whole system fails to carry out the task.
- Simulating a Counter Machine introduces a **bottleneck** that sequentializes the execution and fails to take advantage of the parallel nature of Programmable Matter.

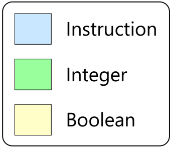
To cope with these problems, a different approach based on Genetic Programming has been explored:

- Designed and developed a Programmable Matter simulator endowed with a general-purpose Genetic-Programming framework that allows particles to autonomously discover algorithms for any given task.
- Tested this approach on several Programmable Matter tasks by devising suitable fitness functions and running the Genetic-Programming framework on a supercomputer.

Abstract Syntax Trees

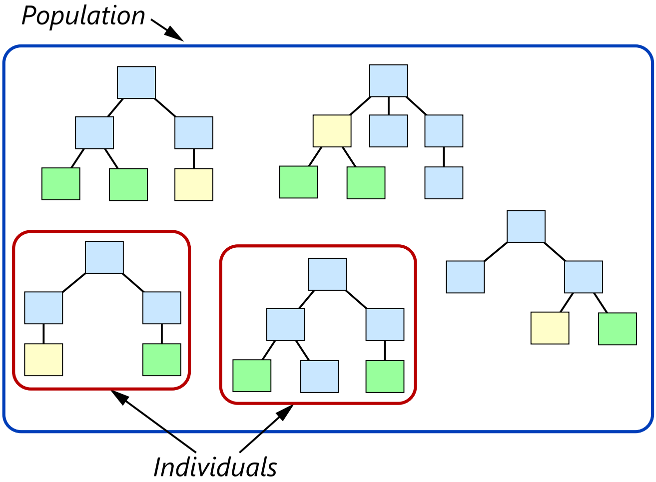
```
while b ≠ 0:
    if a > b:
        a := a - b
    else:
        b := b - a
return a
```





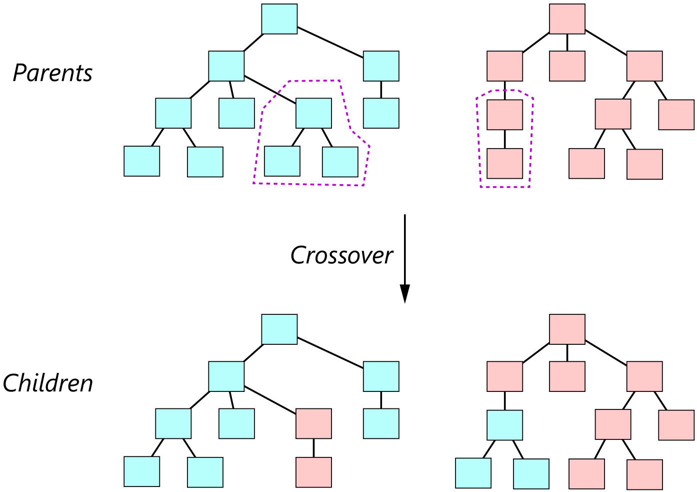
An Abstract Syntax Tree (AST) is a representation of the logical structure of a program. Each node has a type.

Genetic Programming



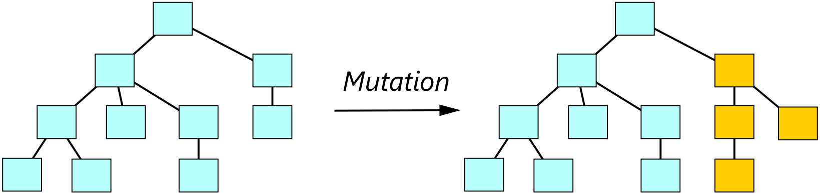
Genetic Programming is an extension of Genetic Algorithms where individuals are ASTs. The goal of Genetic Programming is to find a "good" program that solves a given problem.

Genetic Programming



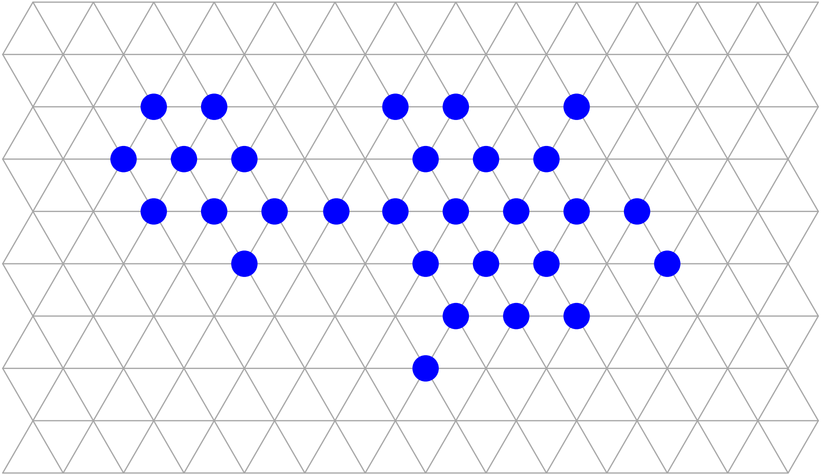
When mating, the two parents' ASTs are combined by exchanging some randomly selected subtrees (having same-type roots).

Genetic Programming



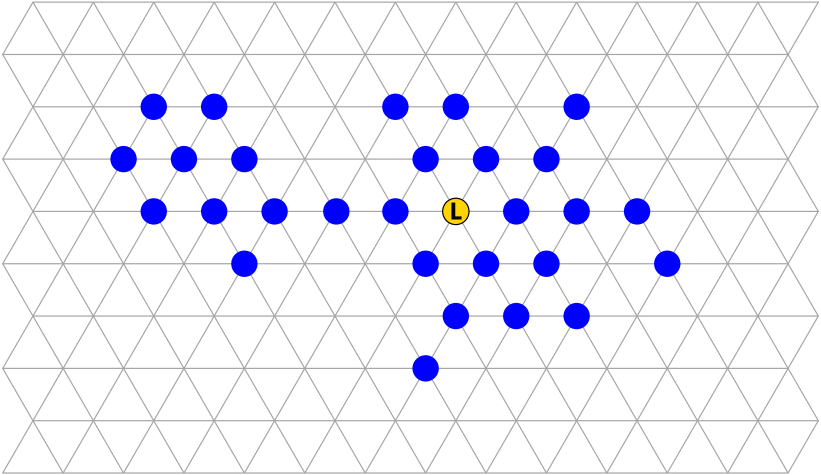
Mutation is done by replacing a randomly selected subtree with a randomly generated (well-formed) AST.

Basic Tasks



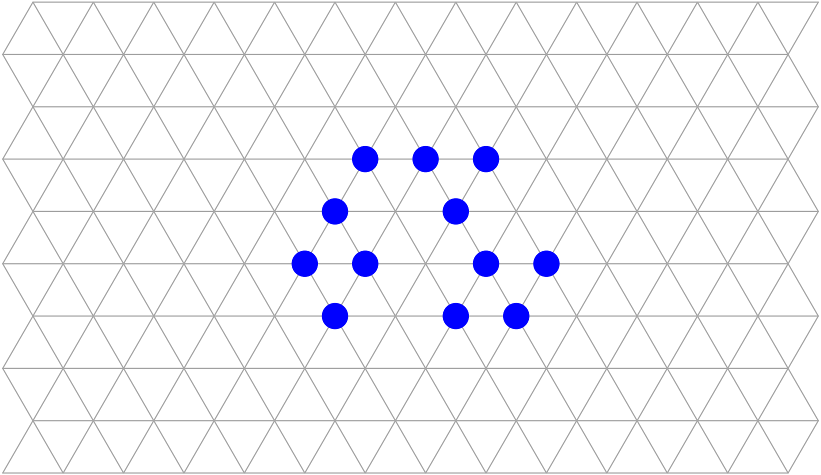
Leader Election: The particles must elect a unique leader without moving. All particles start in the same state.

Basic Tasks



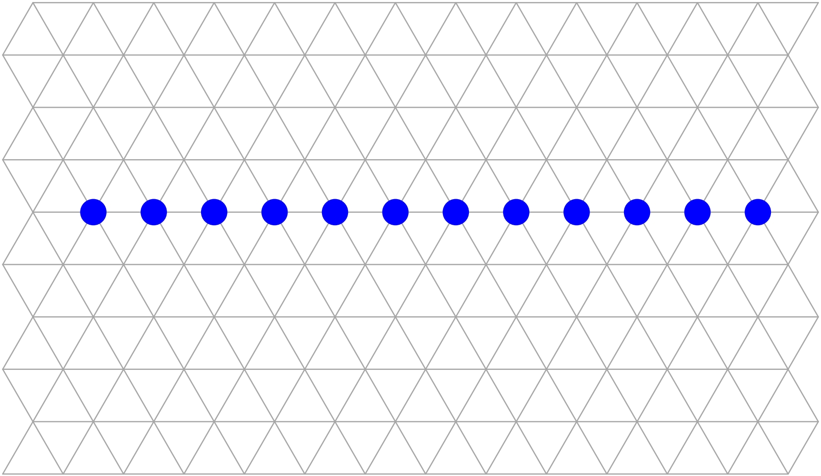
Leader Election: The particles must elect a unique leader without moving. All particles start in the same state.

Basic Tasks



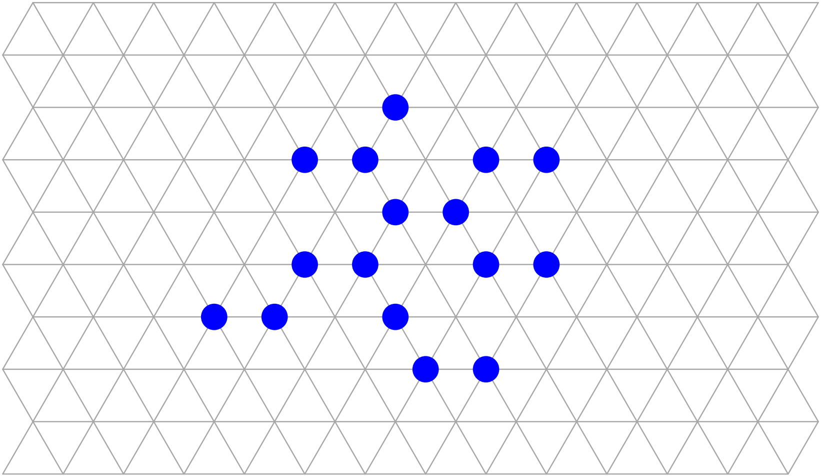
Line Formation: The particles must form a straight line. The initial configuration is assumed to be connected.

Basic Tasks



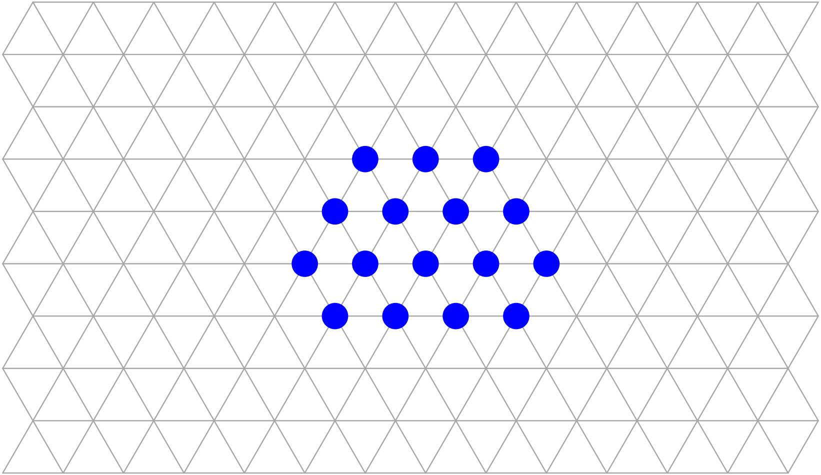
Line Formation: The particles must form a straight line. The initial configuration is assumed to be connected.

Basic Tasks



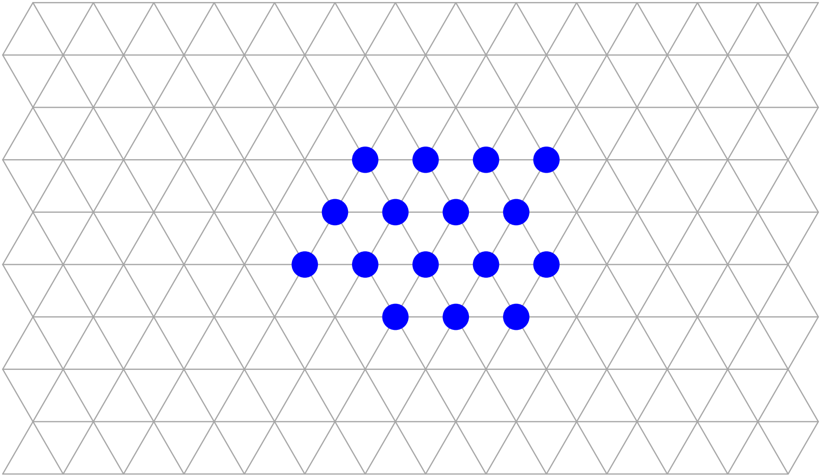
Compaction: The particles must form a configuration of minimum diameter. The initial configuration is assumed to be connected.

Basic Tasks



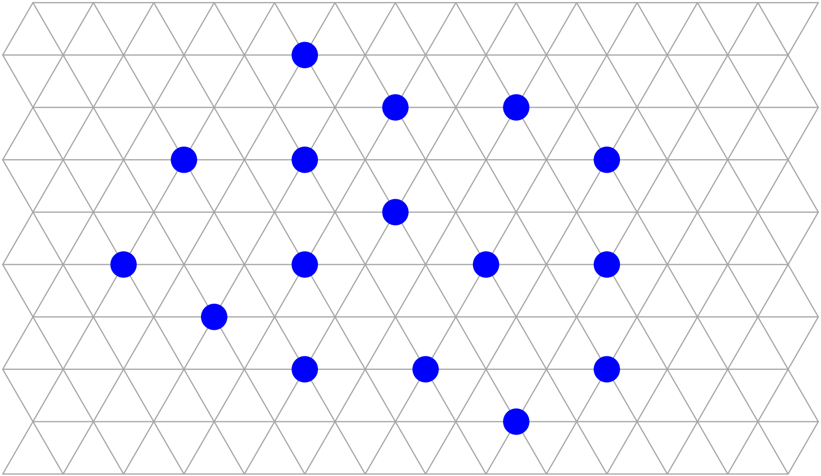
Compaction: The particles must form a configuration of minimum diameter. The initial configuration is assumed to be connected.

Basic Tasks



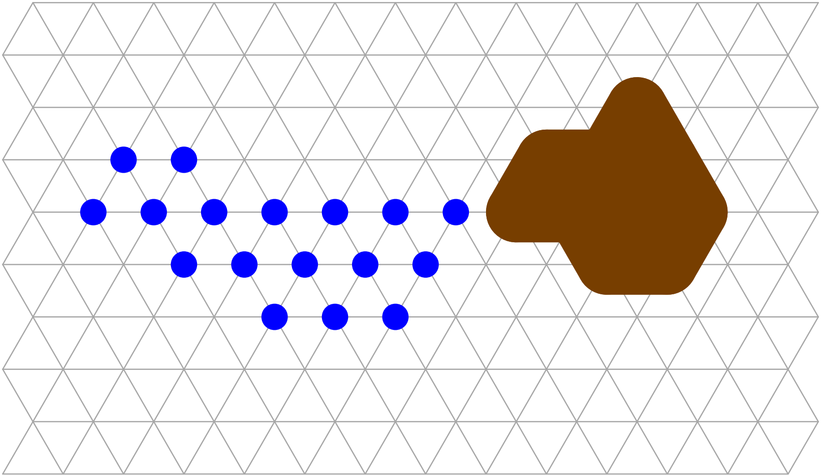
Scattering: The system must reach a configuration where no two particles are adjacent and no particle is moving.

Basic Tasks



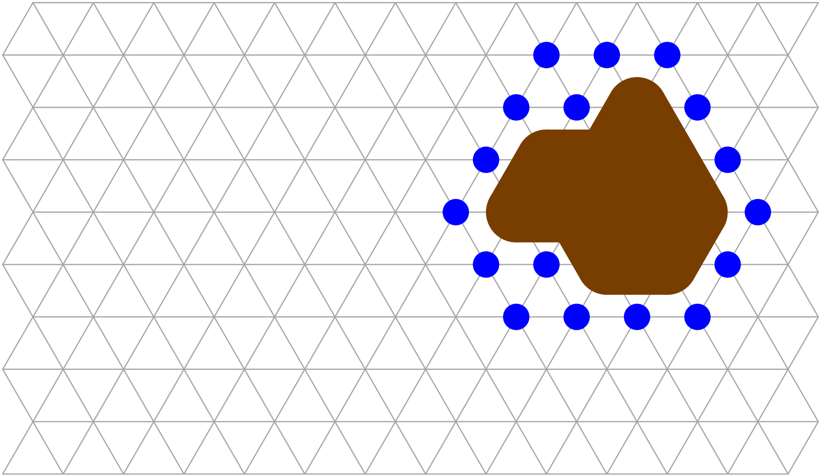
Scattering: The system must reach a configuration where no two particles are adjacent and no particle is moving.

Basic Tasks



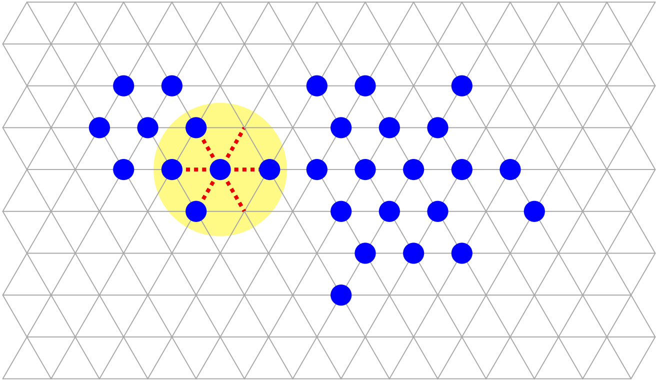
Coating: The particles must completely surround an object of unknown shape. Initially, only one particle is touching the object.

Basic Tasks



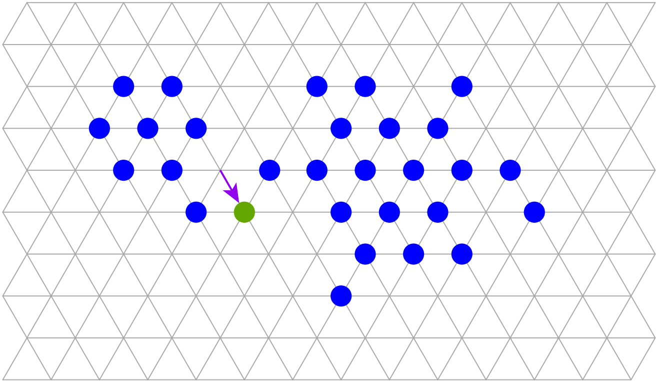
Coating: The particles must completely surround an object of unknown shape. Initially, only one particle is touching the object.

Algorithm Model



A local algorithm is a function that takes as input a particle's **internal state** and **list of neighbors**, each of which may be an empty node or a particle with a certain state. The output is the particle's **new state** and a **direction of movement**.

Algorithm Model



A local algorithm is a function that takes as input a particle's **internal state** and **list of neighbors**, each of which may be an empty node or a particle with a certain state. The output is the particle's **new state** and a **direction of movement**.

Primitive Set

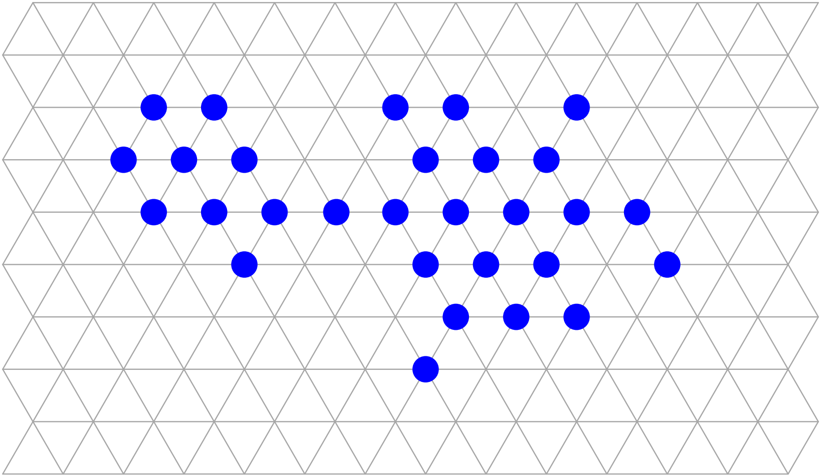
Let us take these "primitives" as building blocks of our algorithms:

- **Basic Instructions**
  - Concatenate [Instruction] and [Instruction]
  - If [Boolean] then [Instruction] else [Instruction]
  - Set state [Integer]
  - Set direction [Integer]
- **Integer Terminals**
  - Get state
  - Get neighbor [Integer]
  - Integer constants
- **Integer Operators**
  - Add [Integer] [Integer]
  - Subtract [Integer] [Integer]
  - Max [Integer] [Integer]
  - Min [Integer] [Integer]

Primitive Set

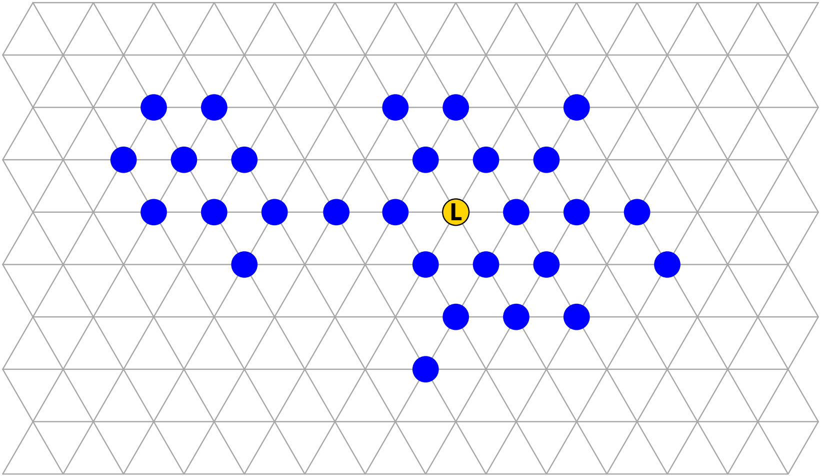
- **Boolean Terminals**
  - True
  - False
- **Boolean Operators**
  - Not [Boolean]
  - And [Boolean] [Boolean]
  - Or [Boolean] [Boolean]
  - Xor [Boolean] [Boolean]
  - Equals [Integer] [Integer]
  - Greater than [Integer] [Integer]
  - Less than [Integer] [Integer]
- **Counter Operations**
  - Set counter [Integer]
  - Get counter
  - Increment counter
  - Decrement counter

Fitness Functions



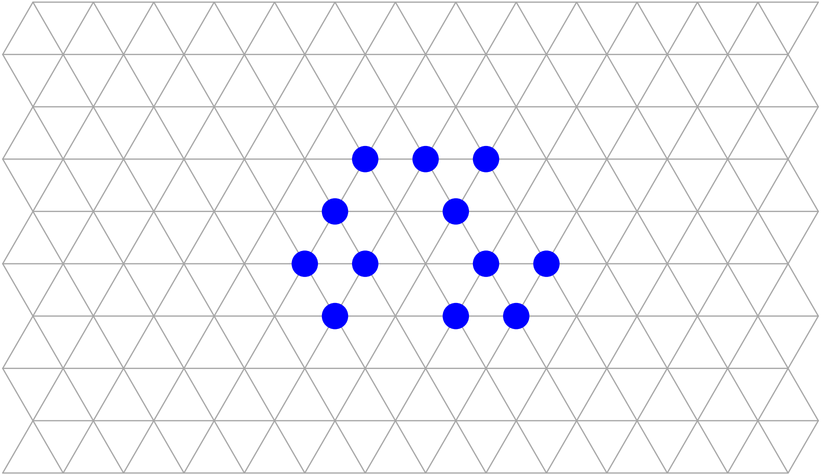
Leader Election: Give a large penalty if there is no leader in the system and a small penalty for having more than one leader.

Fitness Functions



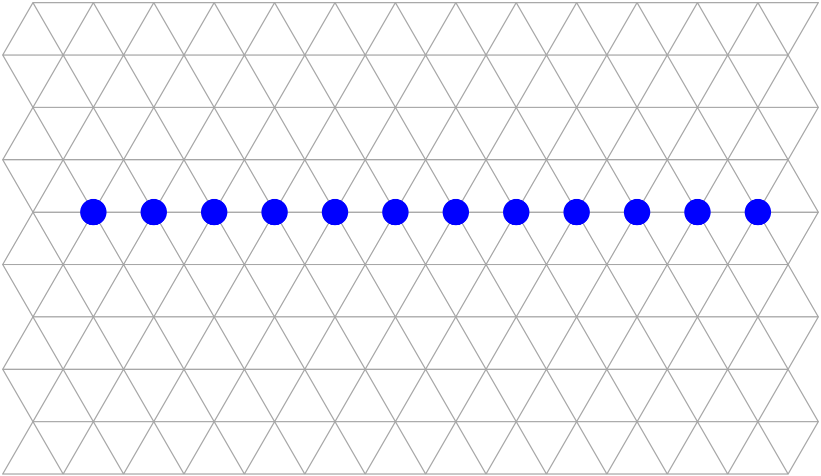
Leader Election: Give a large penalty if there is no leader in the system and a small penalty for having more than one leader.

Fitness Functions



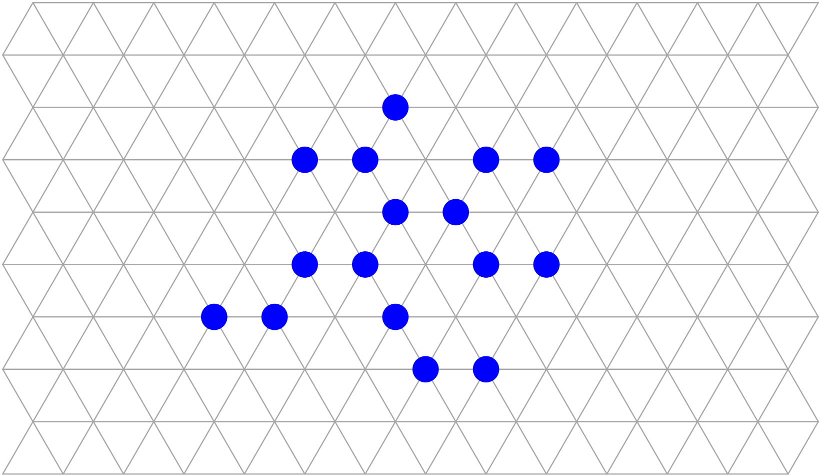
Line Formation: Give a penalty for every particle that does not have exactly two neighbors on opposite sides.

Fitness Functions



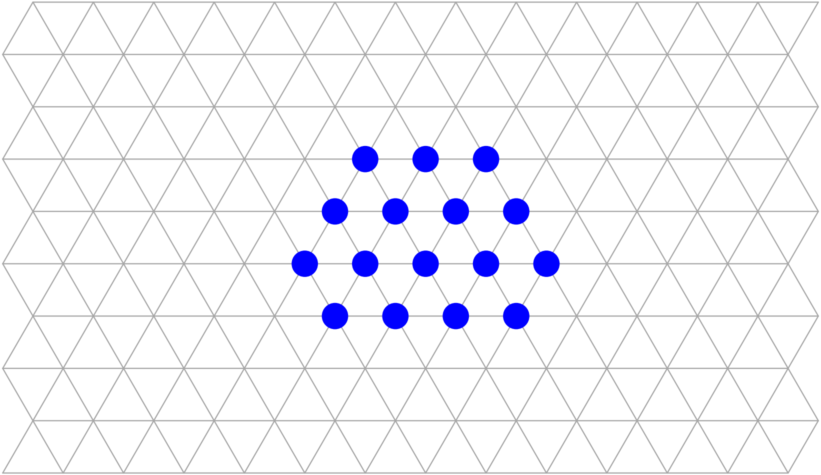
Line Formation: Give a penalty for every particle that does not have exactly two neighbors on opposite sides.

Fitness Functions



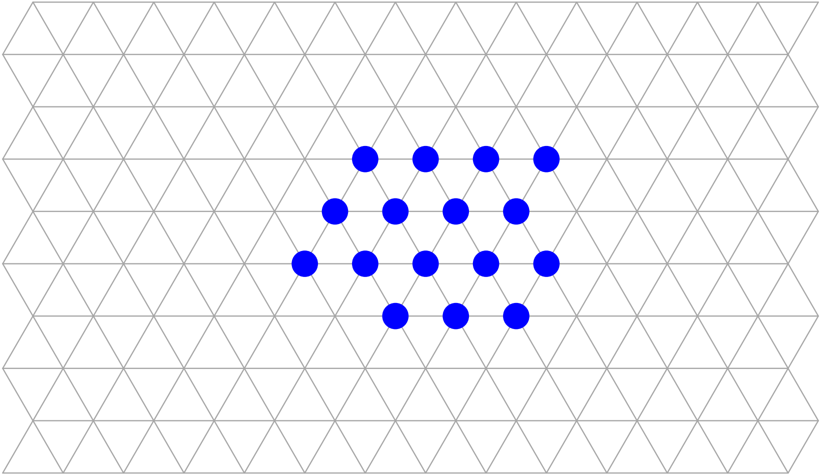
Compaction: Give a penalty for every particle that is not completely surrounded by other particles.

Fitness Functions



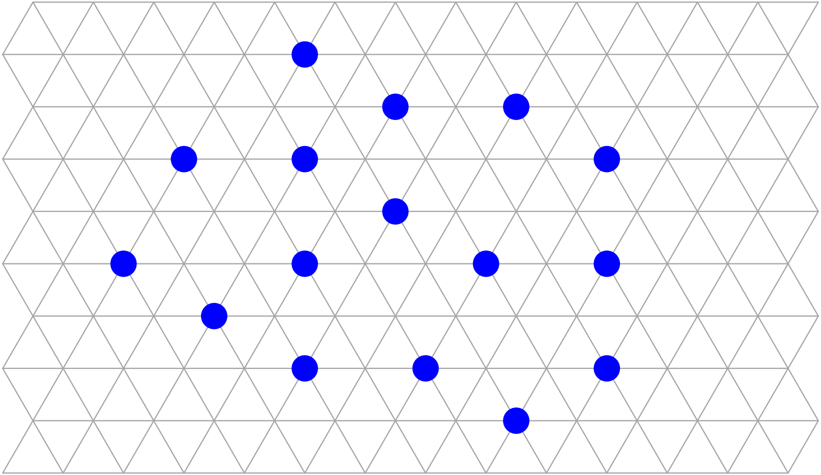
Compaction: Give a penalty for every particle that is not completely surrounded by other particles.

Fitness Functions



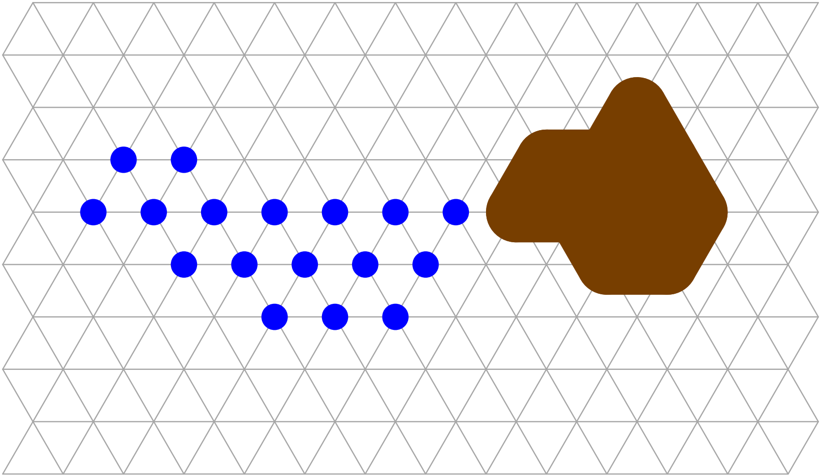
Scattering: Give a large penalty for every two neighboring particles, and a small penalty for particles that are too far apart.

Fitness Functions



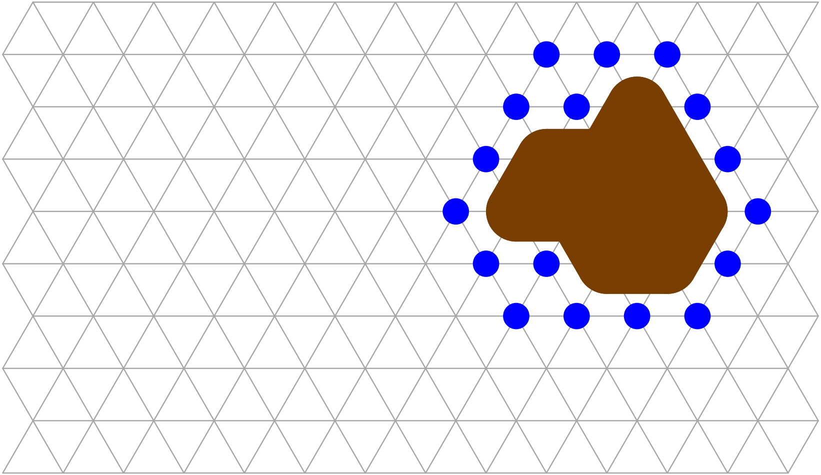
Scattering: Give a large penalty for every two neighboring particles, and a small penalty for particles that are too far apart.

Fitness Functions



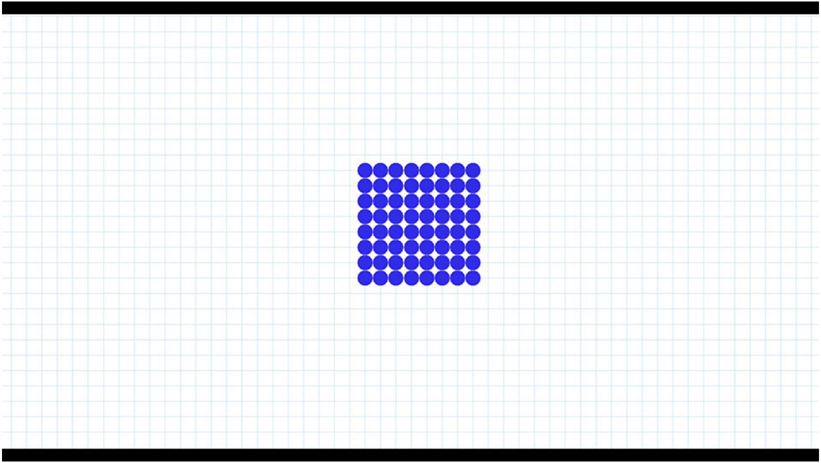
Coating: Give a penalty for every point on the object's surface that is not occupied by a particle.

Fitness Functions



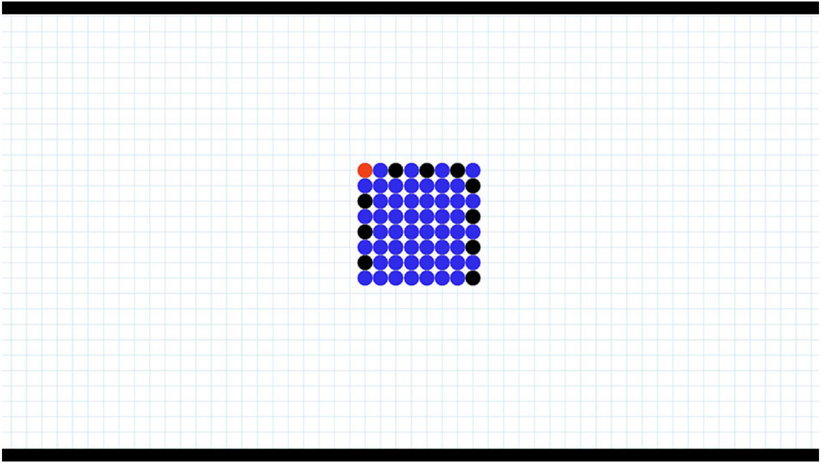
Coating: Give a penalty for every point on the object's surface that is not occupied by a particle.

Experimental Results



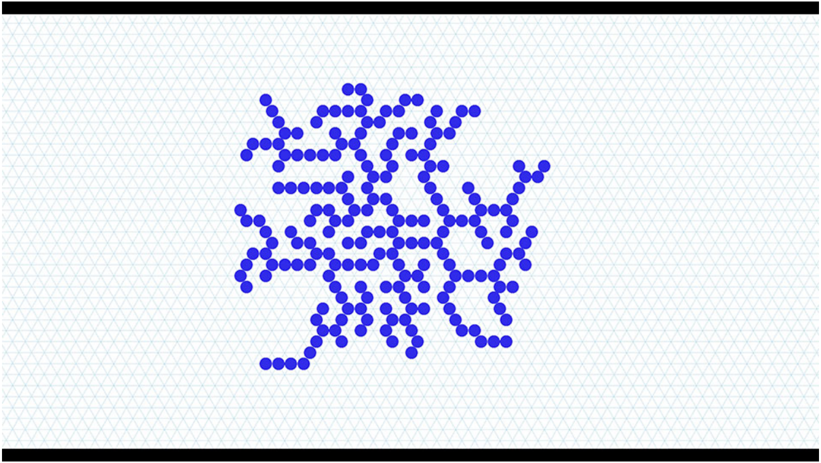
Leader Election in a rectangle

Experimental Results



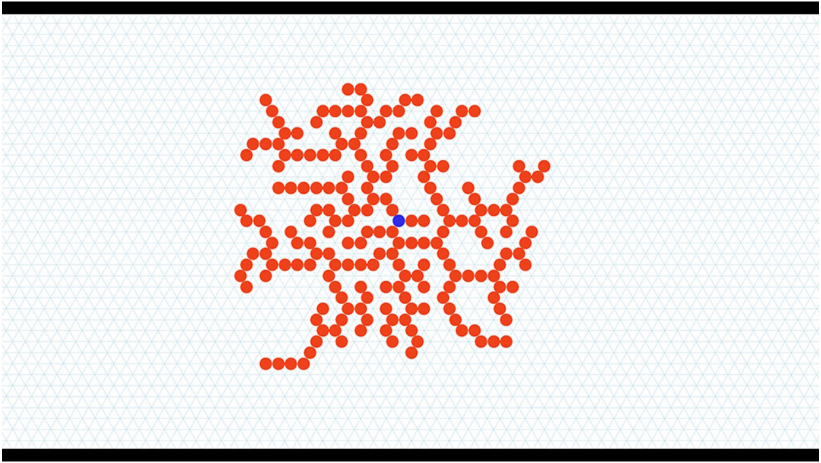
Leader Election in a rectangle

Experimental Results



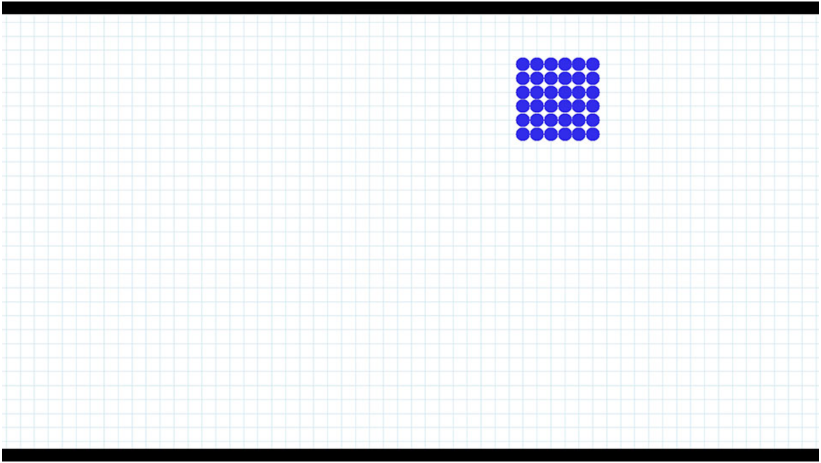
Leader Election in a tree

Experimental Results



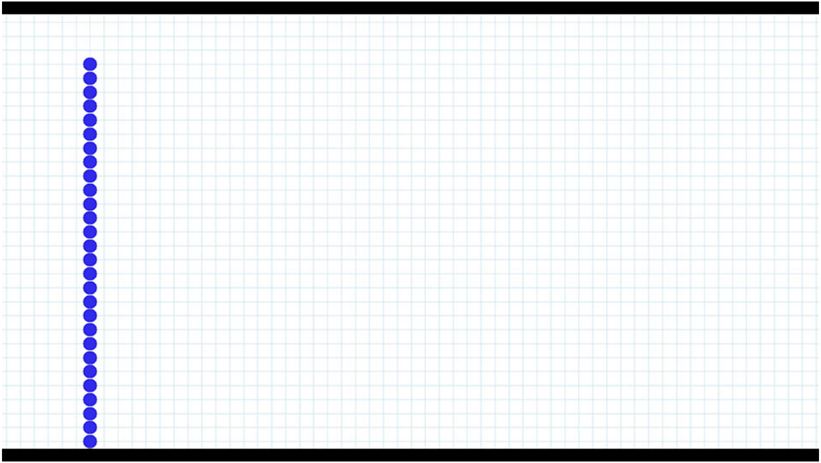
Leader Election in a tree

Experimental Results



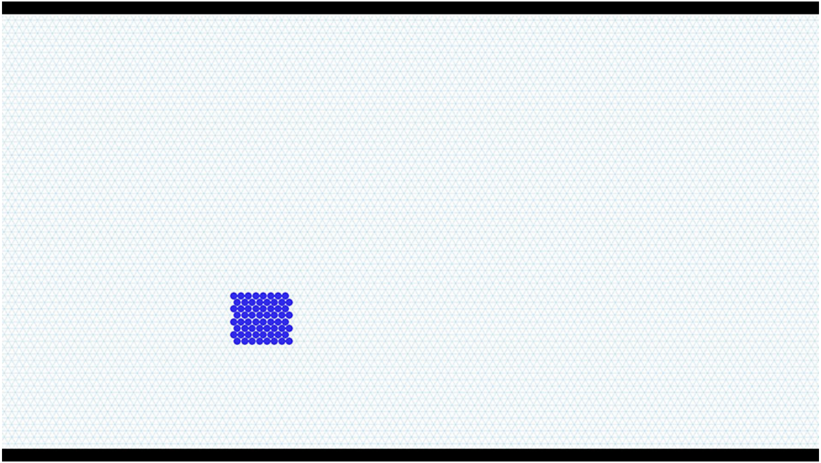
Line Formation from a rectangle in a square grid

Experimental Results



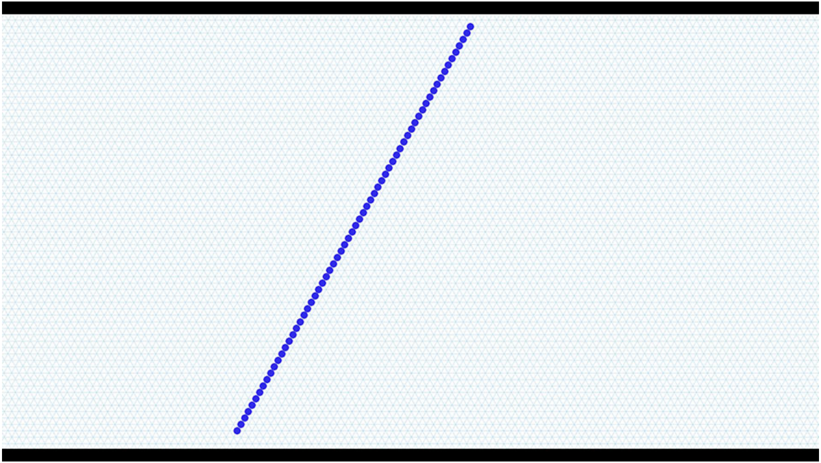
Line Formation from a rectangle in a square grid

Experimental Results



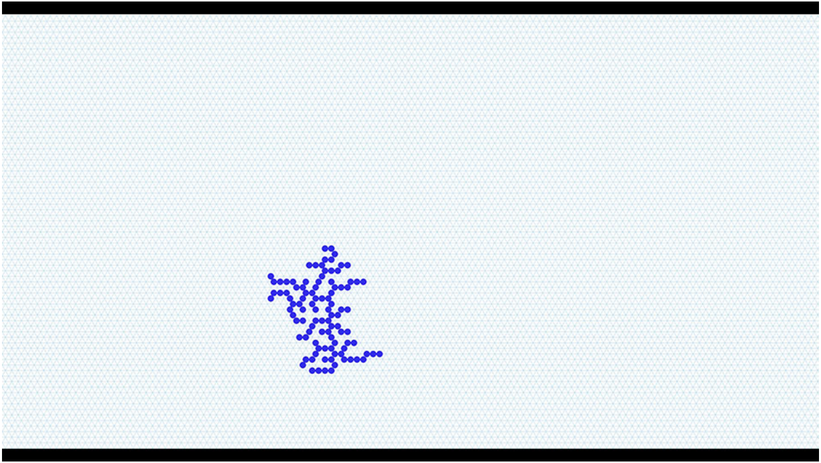
Line Formation from a box in a triangular grid

Experimental Results



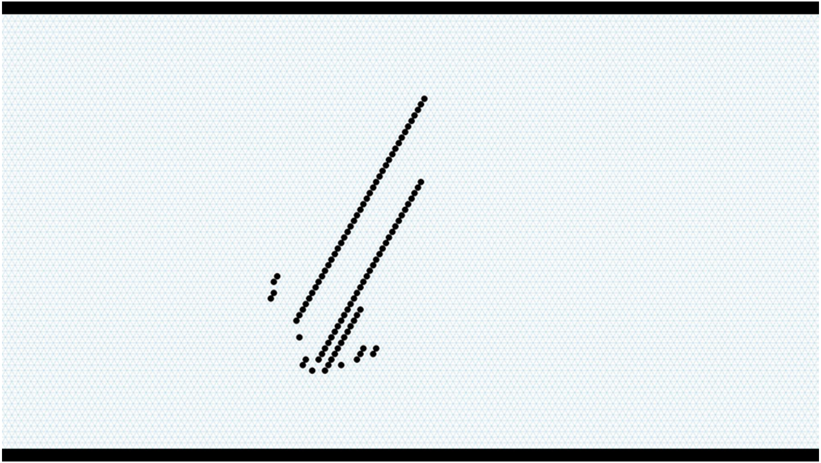
Line Formation from a box in a triangular grid

Experimental Results



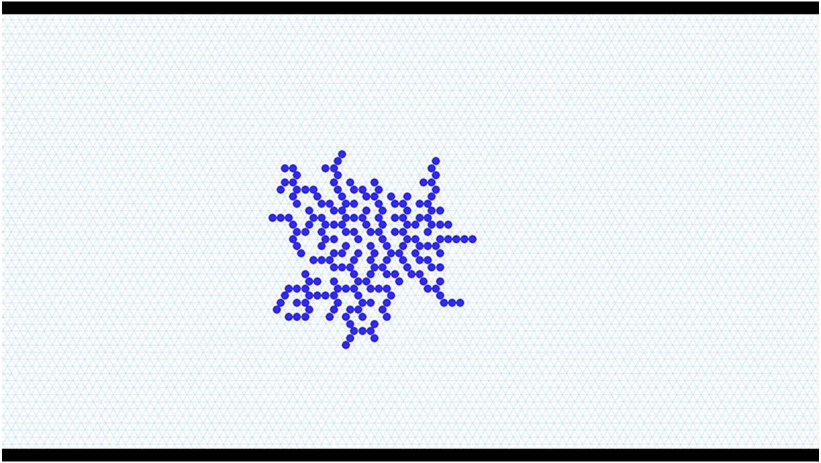
Line Formation from a tree

Experimental Results



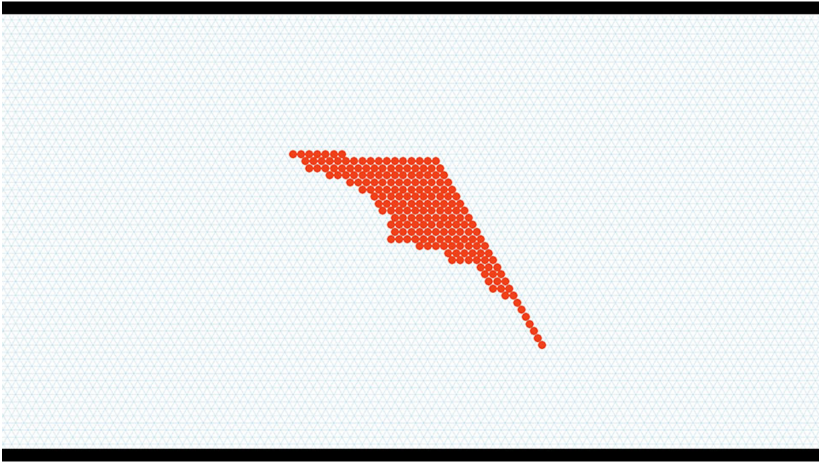
Line Formation from a tree

Experimental Results



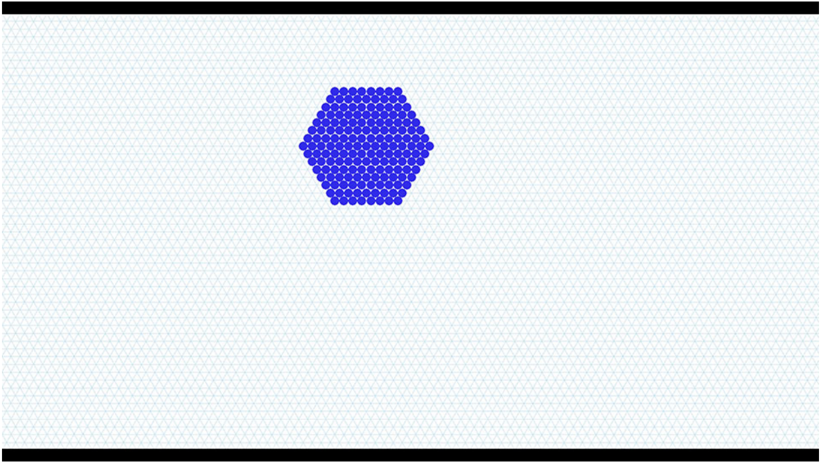
Compaction of a tree

Experimental Results



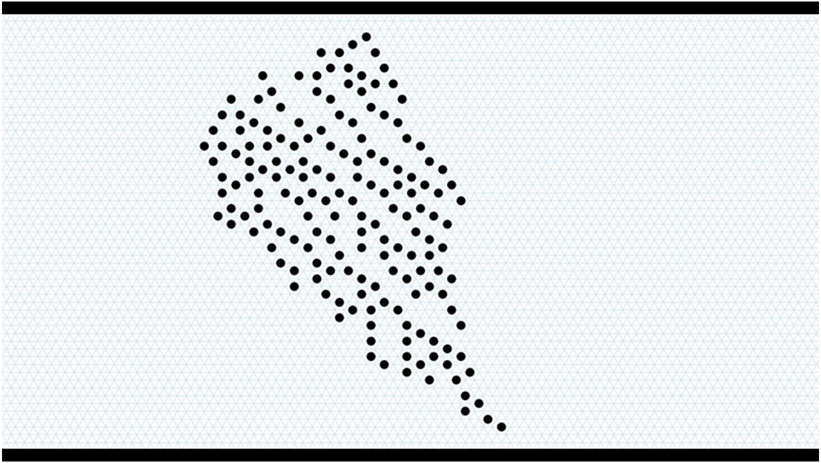
Compaction of a tree

Experimental Results



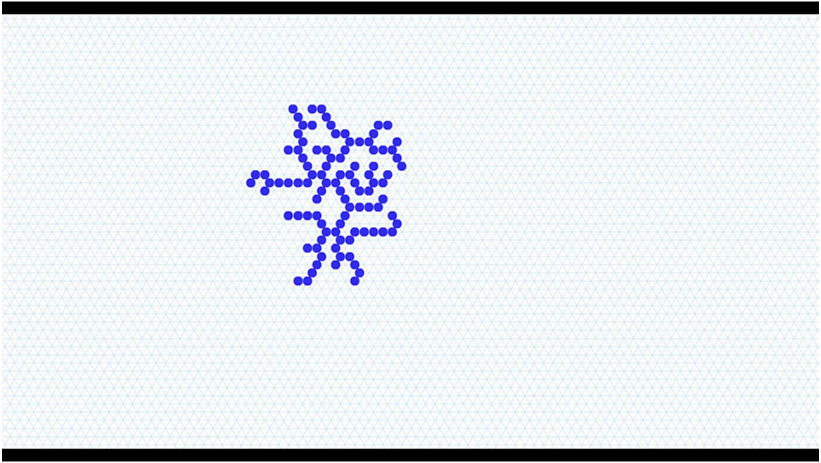
Scattering from a hexagon

Experimental Results



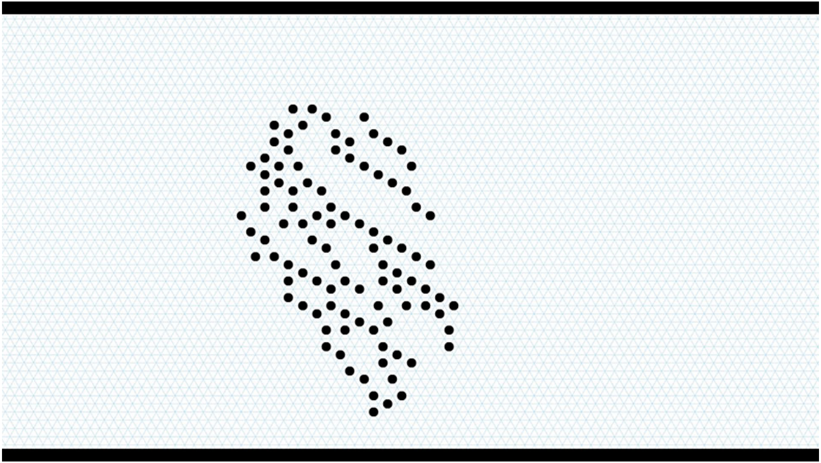
Scattering from a hexagon

Experimental Results



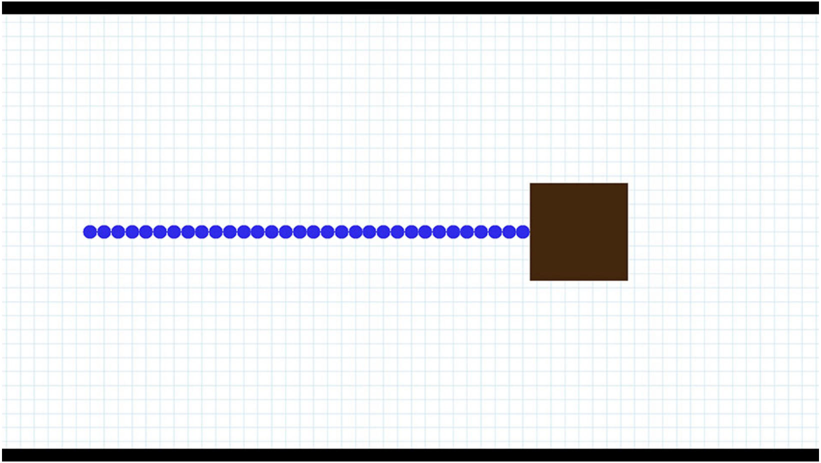
Scattering from a tree

Experimental Results



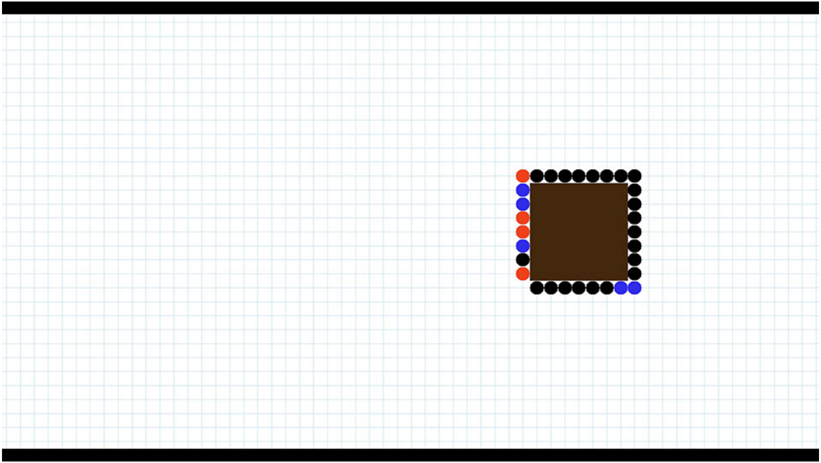
Scattering from a tree

Experimental Results



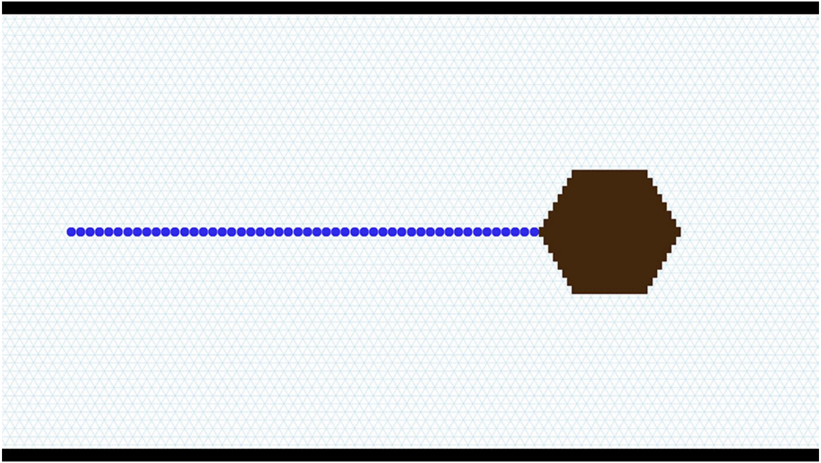
Coating of a rectangle

Experimental Results



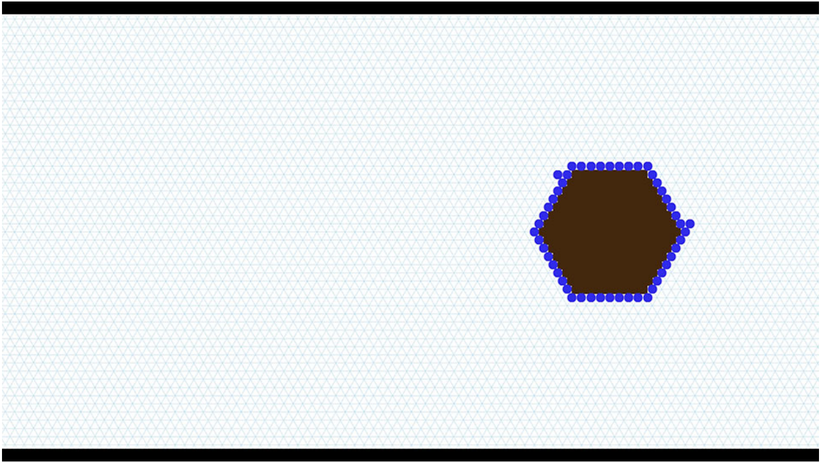
Coating of a rectangle

Experimental Results



Coating of a hexagon

Experimental Results



Coating of a hexagon

Conclusion

Summary:

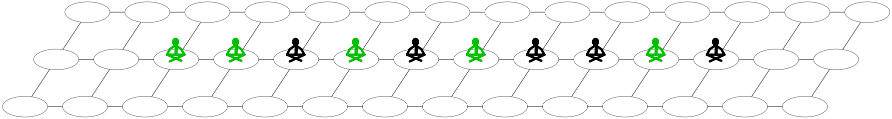
- Developed our Programmable Matter simulator and Genetic Programming framework in Python by extending the DEAP library.
- Evolved our programs on a pair of AMD EPYC 7502 2.5 GHz 32C/64T processors with 16x32 GB DDR4 3200 MHz RAM and a 6 TB Hard Disk.
- The evolved programs can perform fundamental Programmable Matter tasks in some basic settings, and have also re-discovered known techniques such as Saturation.

Future work:

- Design more sophisticated and meaningful primitive functions.
- Perform harder tasks from more general initial configurations.
- Introduce faulty particles and implement fault tolerance.
- Produce humanly understandable algorithms for all tasks.

Line Reconstruction

An earlier attempt to deal with fault tolerance in a special case:



Assume that the initial configuration consists of a line of particles, some of which are *faulty* and unable to move and communicate.

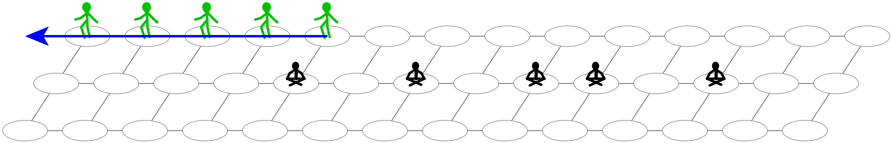
Theorem (*FUN 2016*)

The Line Reconstruction problem is solvable deterministically.

Open problem: what about *Byzantine particles*?

Line Reconstruction

An earlier attempt to deal with fault tolerance in a special case:



The goal is for all the non-faulty particles to "regroup" and form a new line of contiguous particles.

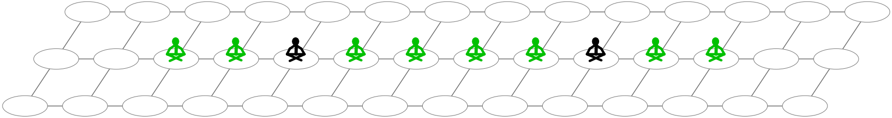
Theorem (*FUN 2016*)

The Line Reconstruction problem is solvable deterministically.

Open problem: what about *Byzantine particles*?

Line Reconstruction

An earlier attempt to deal with fault tolerance in a special case:



If the initial configuration is symmetric, it is acceptable to form two lines instead of one.

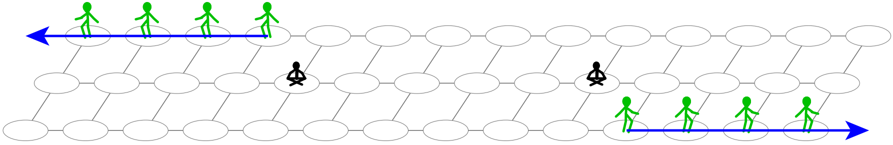
Theorem (*FUN 2016*)

The Line Reconstruction problem is solvable deterministically.

Open problem: what about *Byzantine particles*?

Line Reconstruction

An earlier attempt to deal with fault tolerance in a special case:



If the initial configuration is symmetric, it is acceptable to form two lines instead of one.

Theorem (*FUN 2016*)

The Line Reconstruction problem is solvable deterministically.

Open problem: what about *Byzantine particles*?