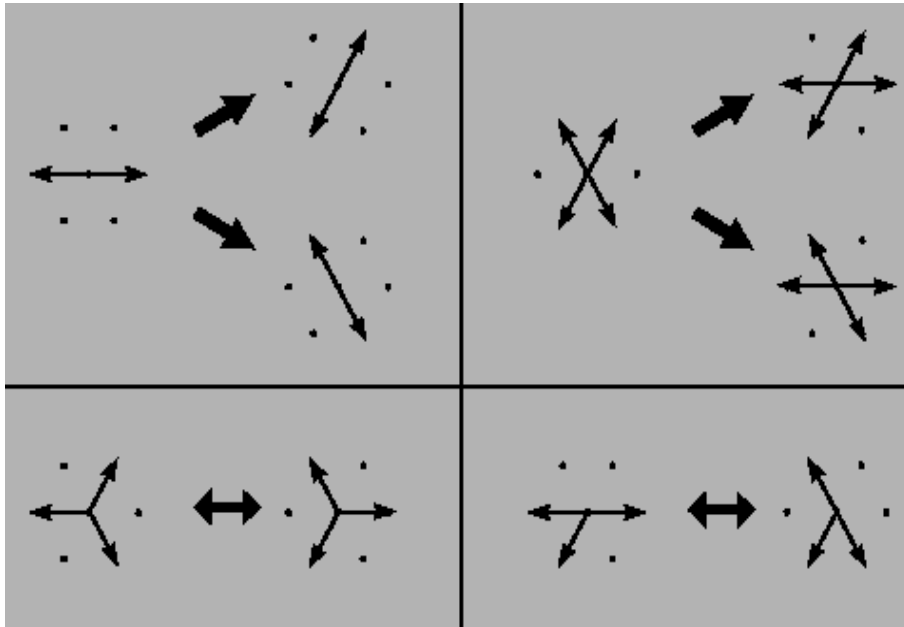


# Lattice gas simulations

Tony Kim  
Spring 2007  
18.354 Project

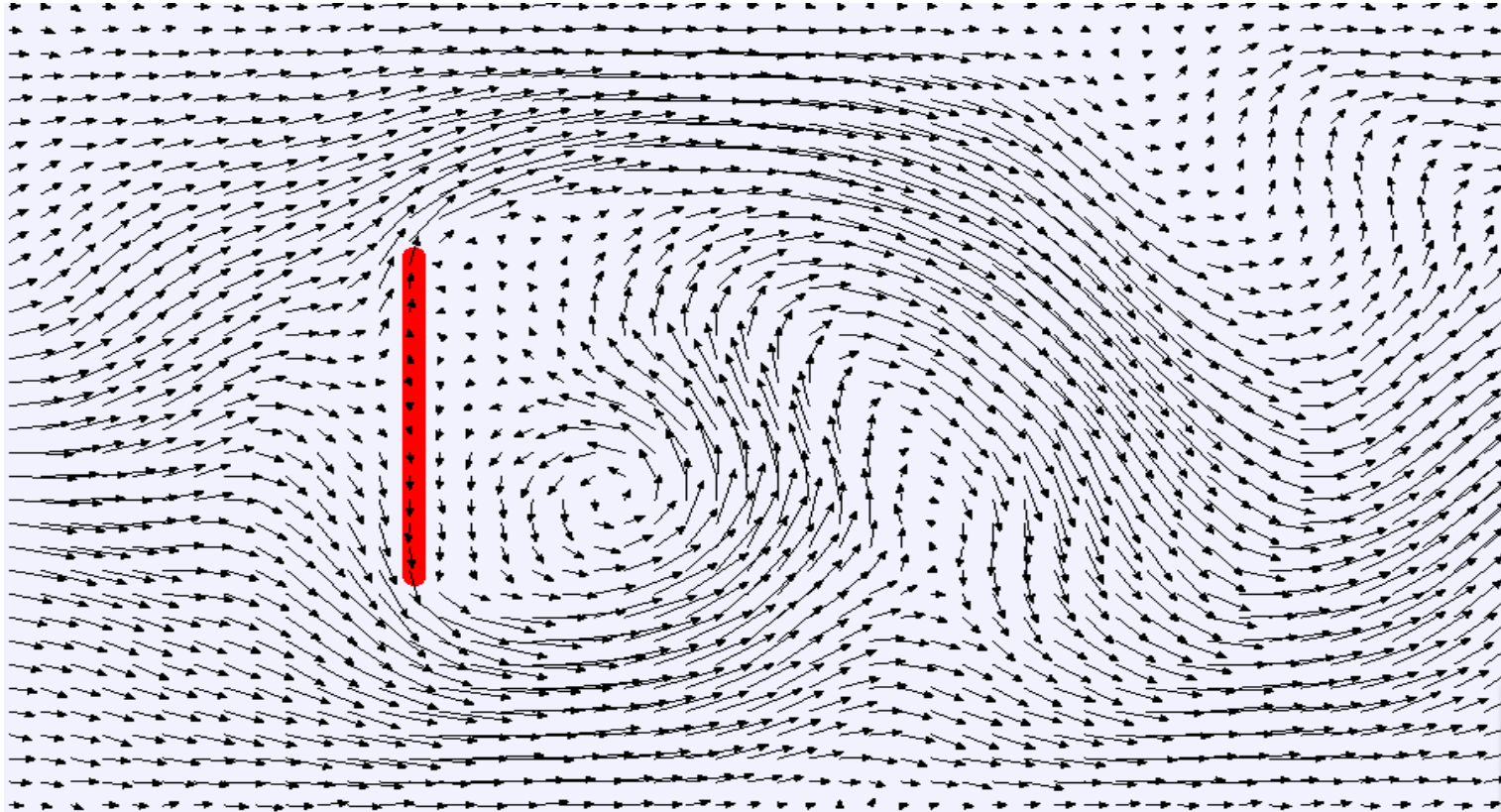
- 1) **Introducing the lattice gas;**
- 2) Analytic description of the lattice gas;
- 3) Program objectives;
- 4) How to implement it (efficiently);
- 5) Demonstrations;

# What is the lattice gas?



- A completely unphysical description of the motion of particles.
- The particle is an entity that hops from point to point on the lattice with each (discrete) time step.

# Why do we use it?



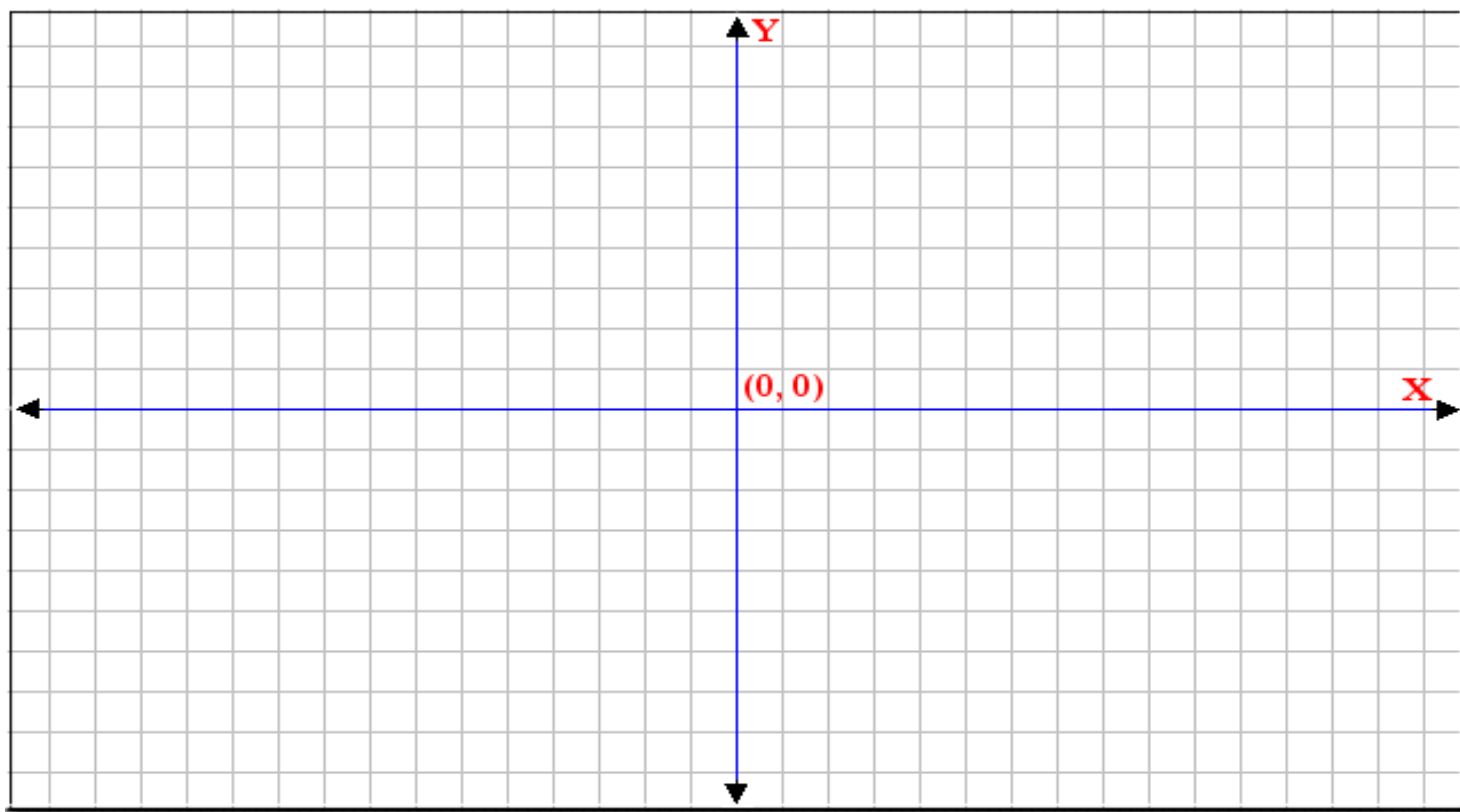
- Gives completely physical results when viewed at large enough scales

# Why do we use it? (2)

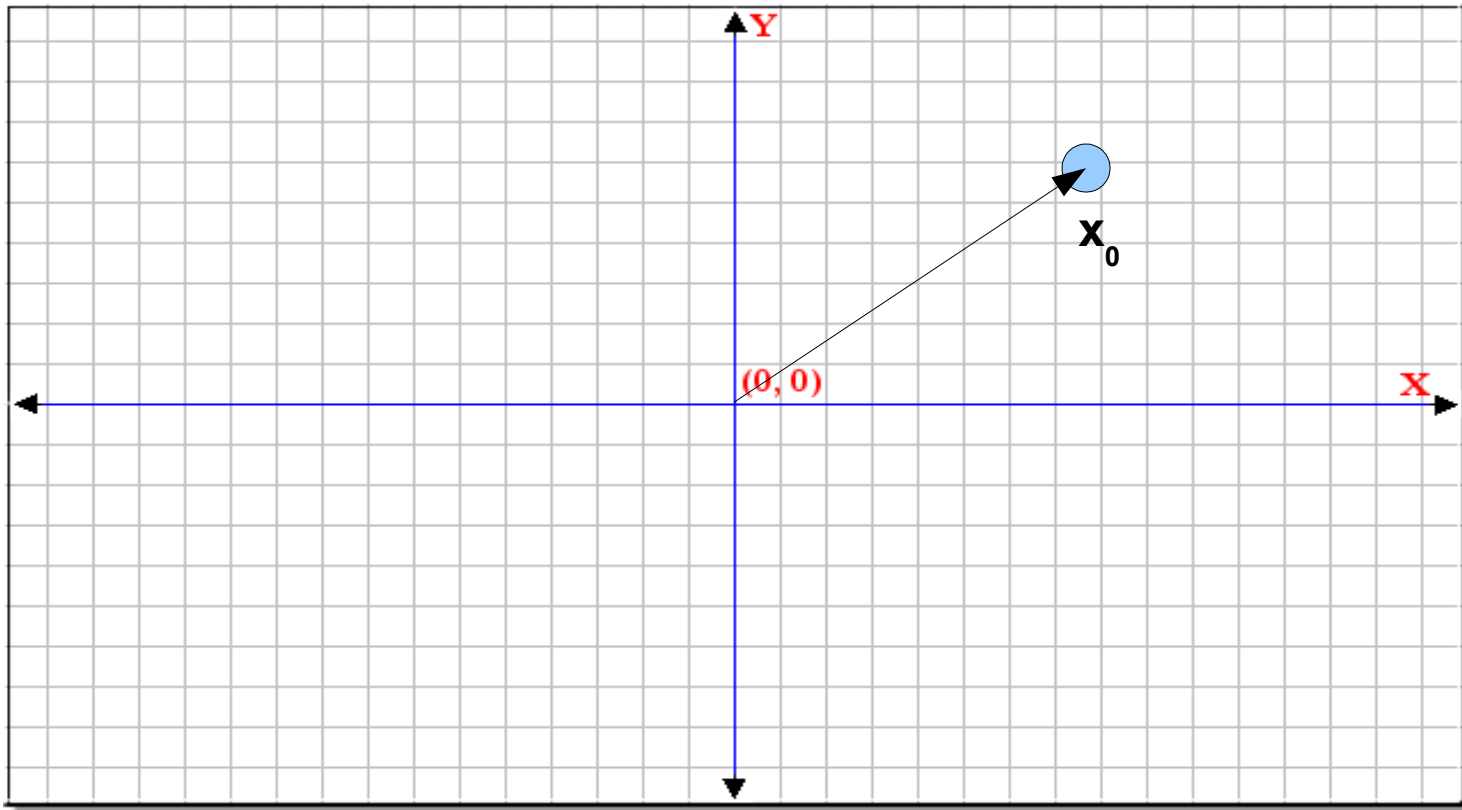
- Because trying to simulate a collection of particles in continuous space is expensive.
- Intuitively, the problem scales as  $O(n^2)$  just for the collisions.
  - For each of the  $n$  particles,
  - we have to test whether it has collided with  $(n-1)$  other particles.
- Will see later that the lattice gas has nice scaling properties in terms of the required computational effort.

- 1) Introducing the lattice gas;
- 2) **Analytic description of the lattice gas;**
- 3) Program objectives;
- 4) How to implement it (efficiently);
- 5) Demonstrations;

# Start with an empty coordinate

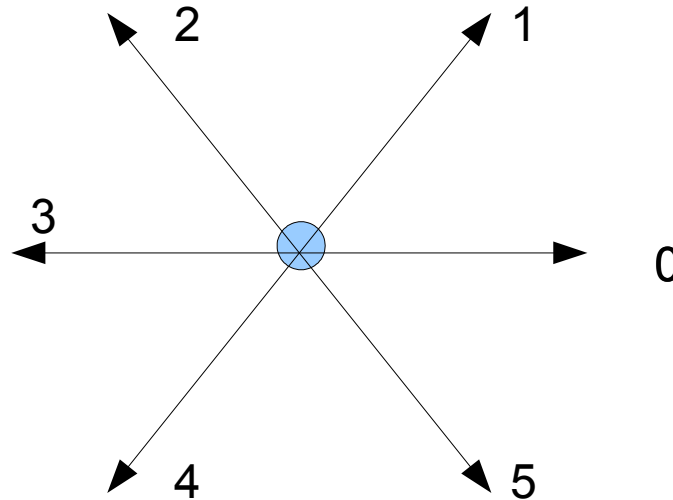


# Place a node at position $\mathbf{x}_0$



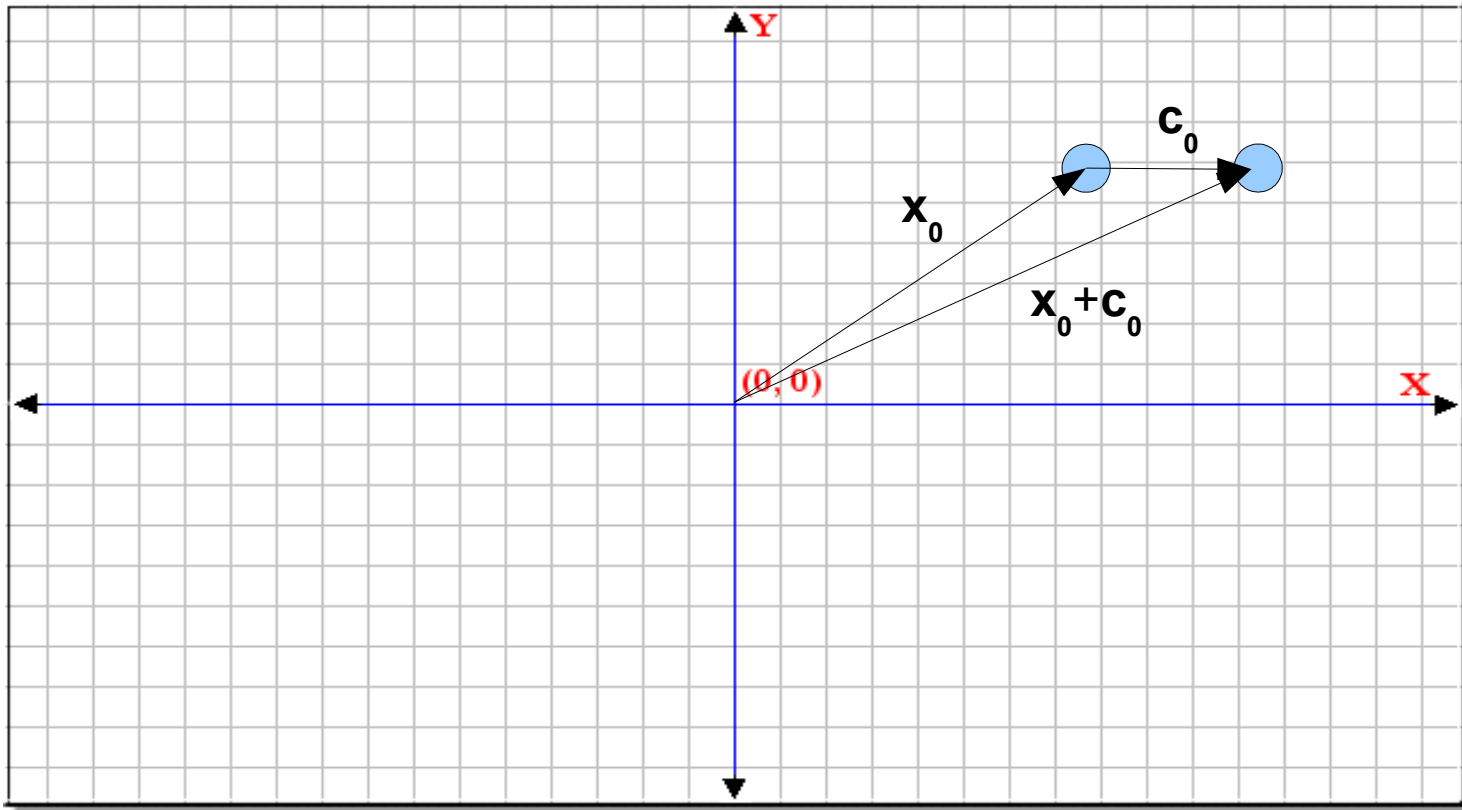
- We now use  $\mathbf{x}_0$  to label the node *at*  $\mathbf{x}_0$ .
  - *We can refer to the node at  $\mathbf{x}_0$  by the vector.*

# One point is no lattice; so let's add some neighbors



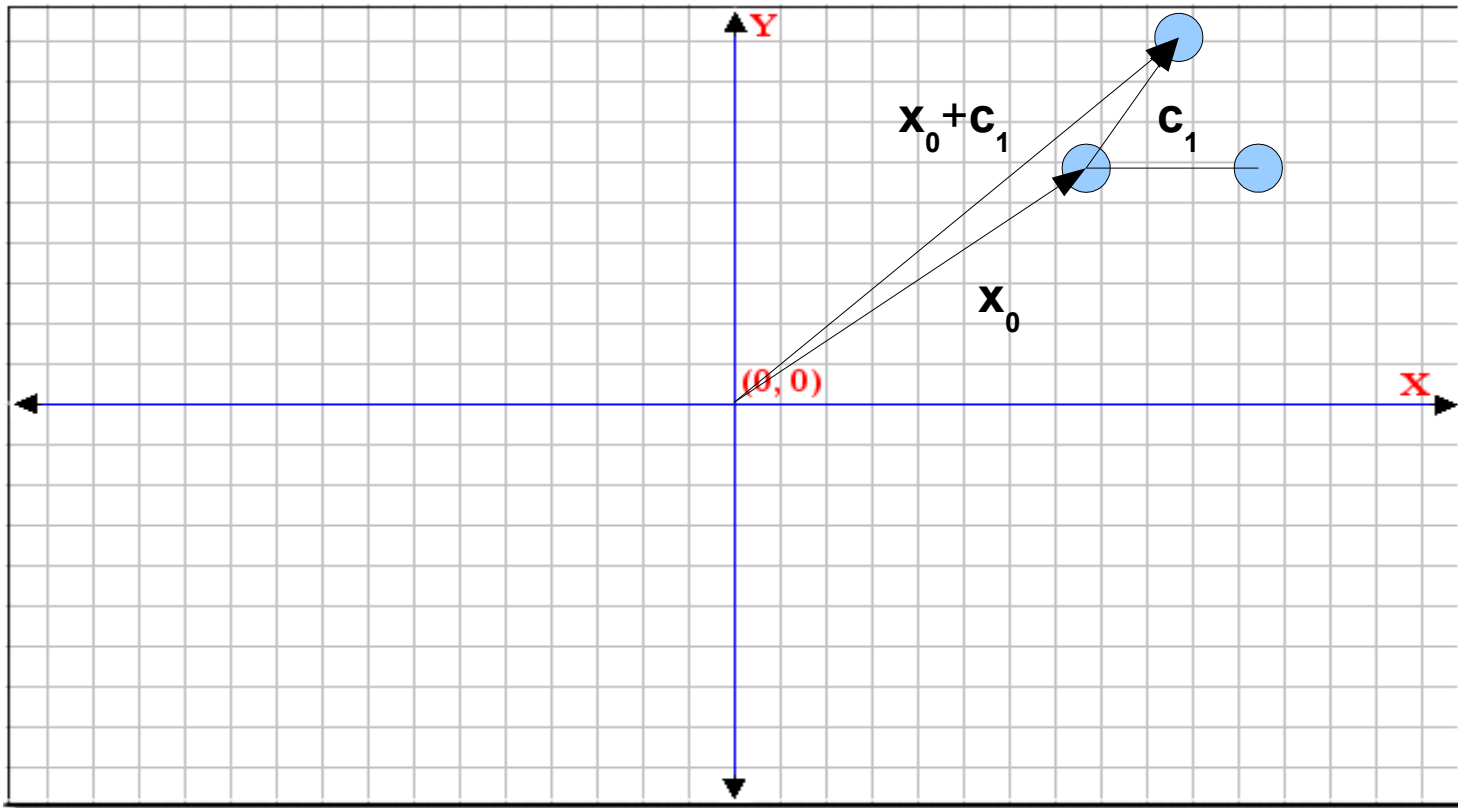
- Let  $\mathbf{c}_i$  denote the vector that connects some node to its neighbor in the  $i$ -th direction,
  - where  $i = 0, 1, 2, 3, 4, 5$

# Creating neighbors



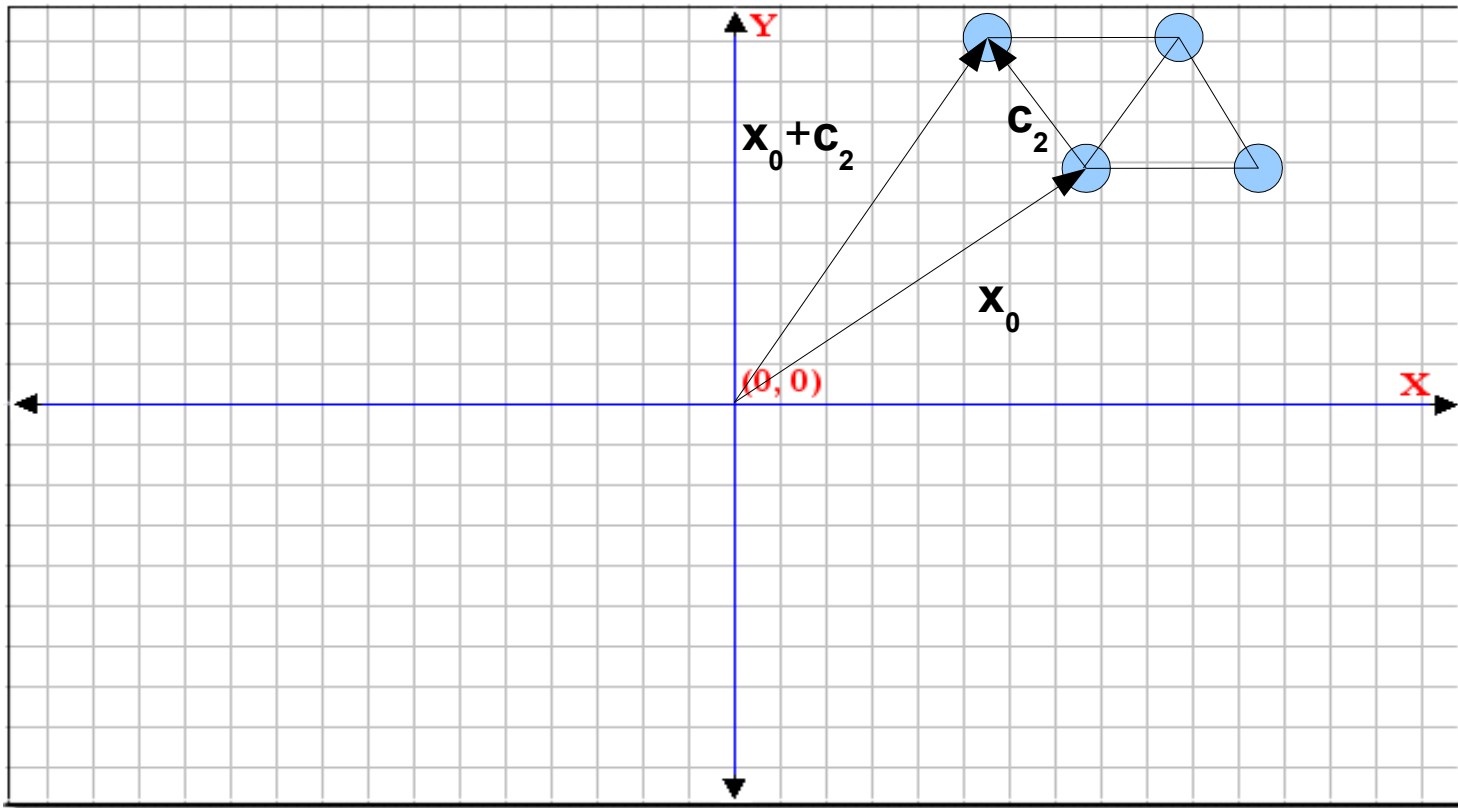
- So we can add a new node at  $x_0 + c_0$
- The new node too can be identified by its position in the coordinate system:  $x_0 + c_0$

And so on  $(\mathbf{x}_0 + \mathbf{c}_1) \dots$

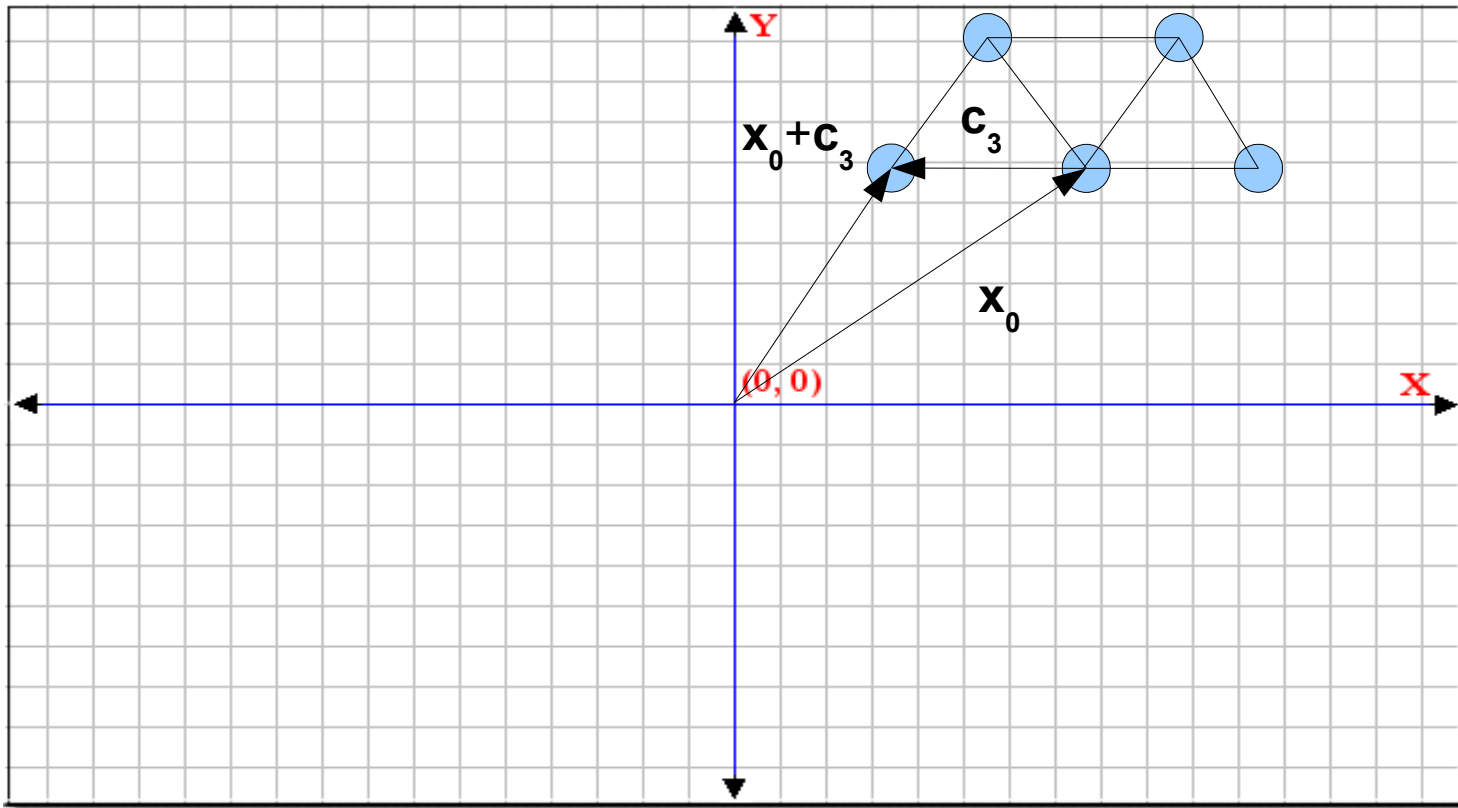


- The solid line indicates a “lattice connection” between the node at  $\mathbf{x}_0$  and  $\mathbf{x}_0 + \mathbf{c}_0$

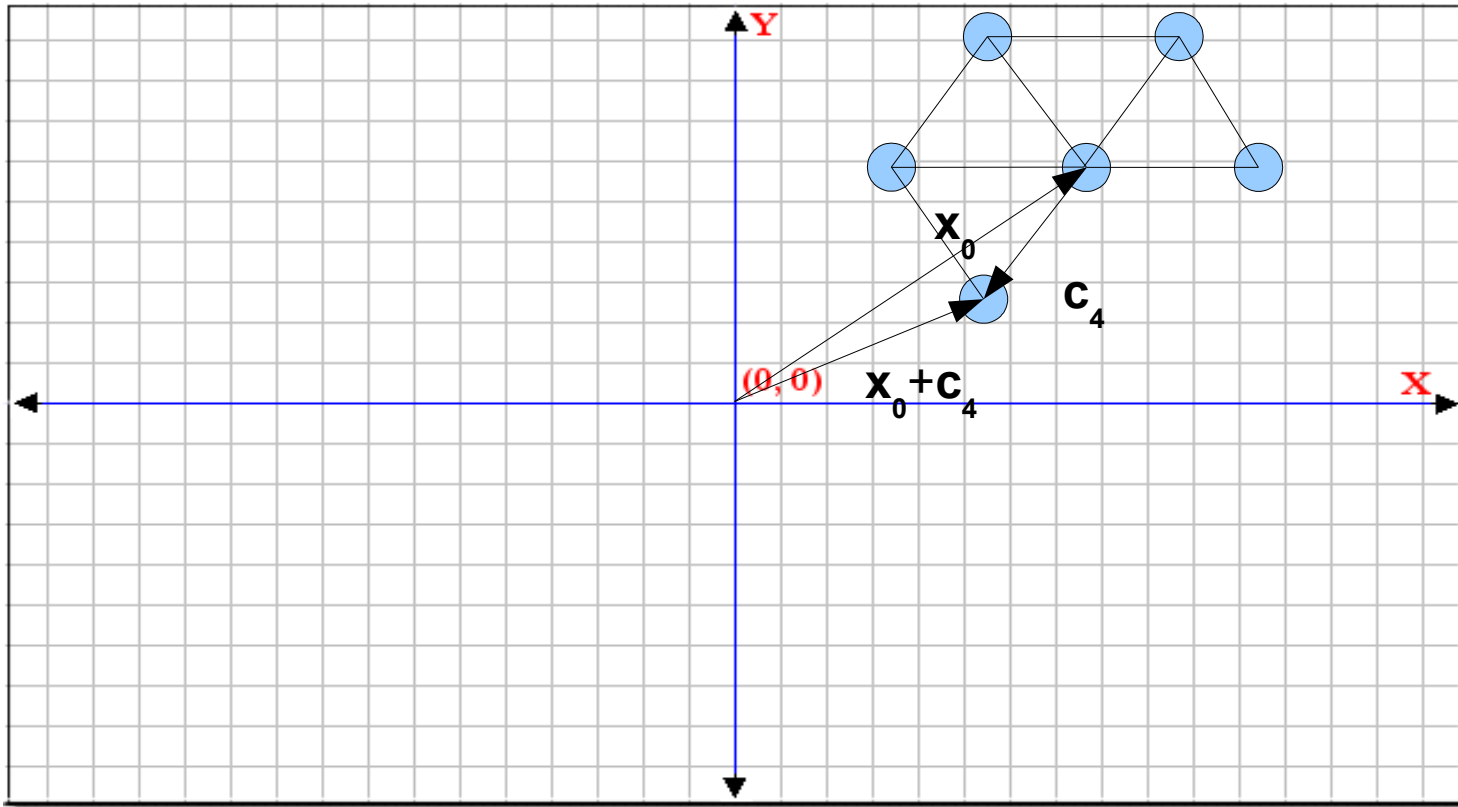
And so on  $(\mathbf{x}_0 + \mathbf{c}_2) \dots$



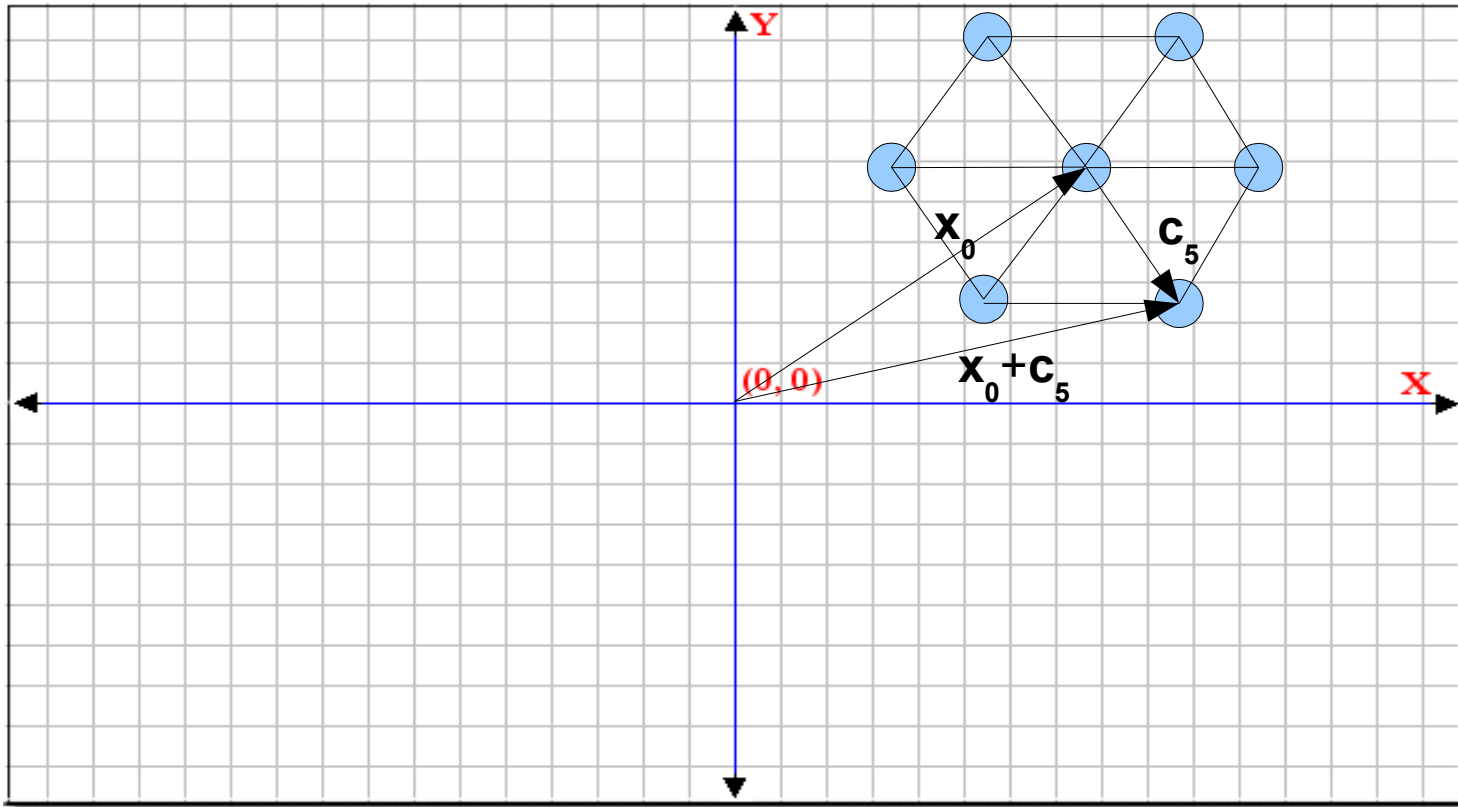
And so on  $(\mathbf{x}_0 + \mathbf{c}_3) \dots$



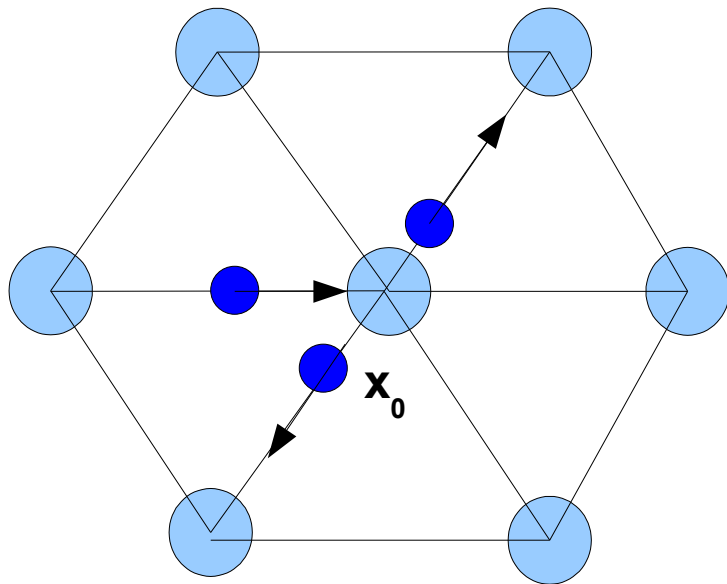
And so on  $(\mathbf{x}_0 + \mathbf{c}_4) \dots$



And so on  $(\mathbf{x}_0 + \mathbf{c}_5) \dots$

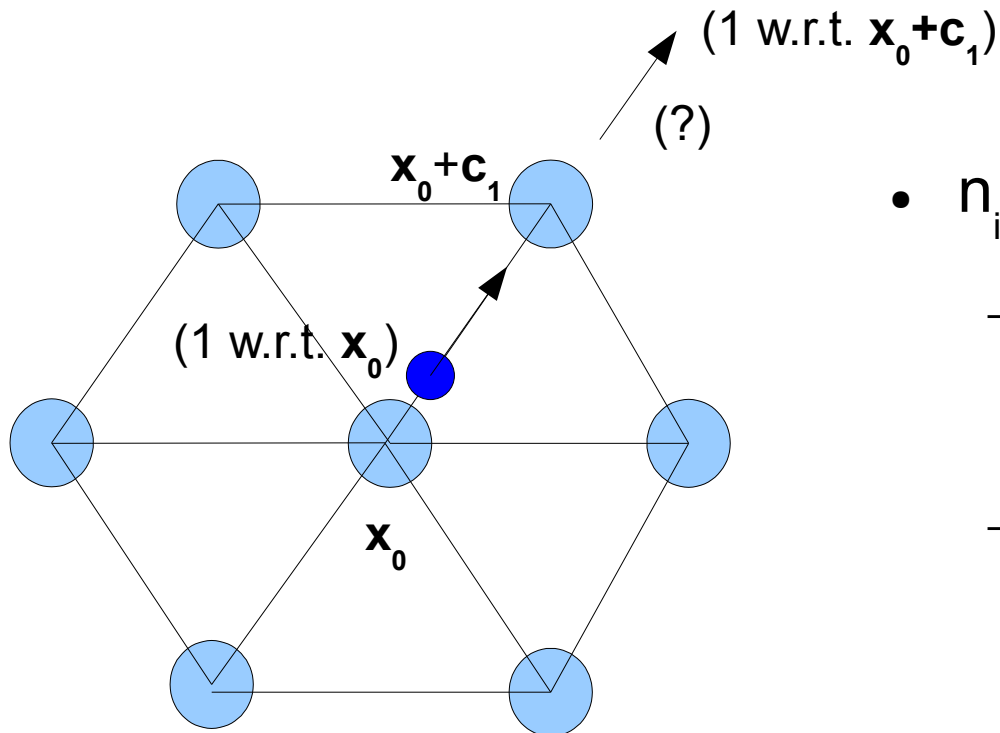


# Denoting particles at a node



- Now we have a node at  $\mathbf{x}_0$  with all six neighbors.
- We denote the presence of a particle at the node  $\mathbf{x}_0$  heading towards the  $i$ -th direction with  $n_i(\mathbf{x}_0)$
- $n_i(\mathbf{x}_0)$  takes boolean values (0 or 1) depending on the occupancy.

# Evolution equation



- $n_i(\mathbf{x} + \mathbf{c}_i, t+1) = n_i(\mathbf{x}, t) + \Delta[\mathbf{n}(\mathbf{x}, t)]$ 
  - Where  $\Delta[\mathbf{n}(\mathbf{x}, t)]$  is the “momentum operator” acting on the configuration state  $\mathbf{n}(\mathbf{x}, t)$ .
  - The sophistication of the model depends on the nature of  $\Delta$  chosen. In my project I deal with only 2- and 3-body collisions.

e.g. Consider  $i = 1$  case:

- $n_1(\mathbf{x} + \mathbf{c}_1, t+1) = n_1(\mathbf{x}, t) + \Delta[\mathbf{n}(\mathbf{x}, t)]$
- $n_1(\mathbf{x} + \mathbf{c}_1, t+1) = n_1(\mathbf{x}, t)$  (Presumably)
- $n_1(\mathbf{x} + \mathbf{c}_1, t+1) = 1$  (i.e. The particle continues on its path.)

- 1) Introducing the lattice gas;
- 2) Analytic description of the lattice gas;
- 3) **Program objectives;**
- 4) How to implement it (efficiently);
- 5) Demonstrations;

# Objectives

- Malleable lattice points; so that I can create the node network “on the fly”
  - This requires some thought into the underlying data structure. The simple two-dimensional array will not suffice for the dynamic network.
- Ability to simulate  $N > 1000$  particles at acceptable speeds.
  - This is very modest. The field picture at the introduction contains tens of thousands of particles at each “arrow.”

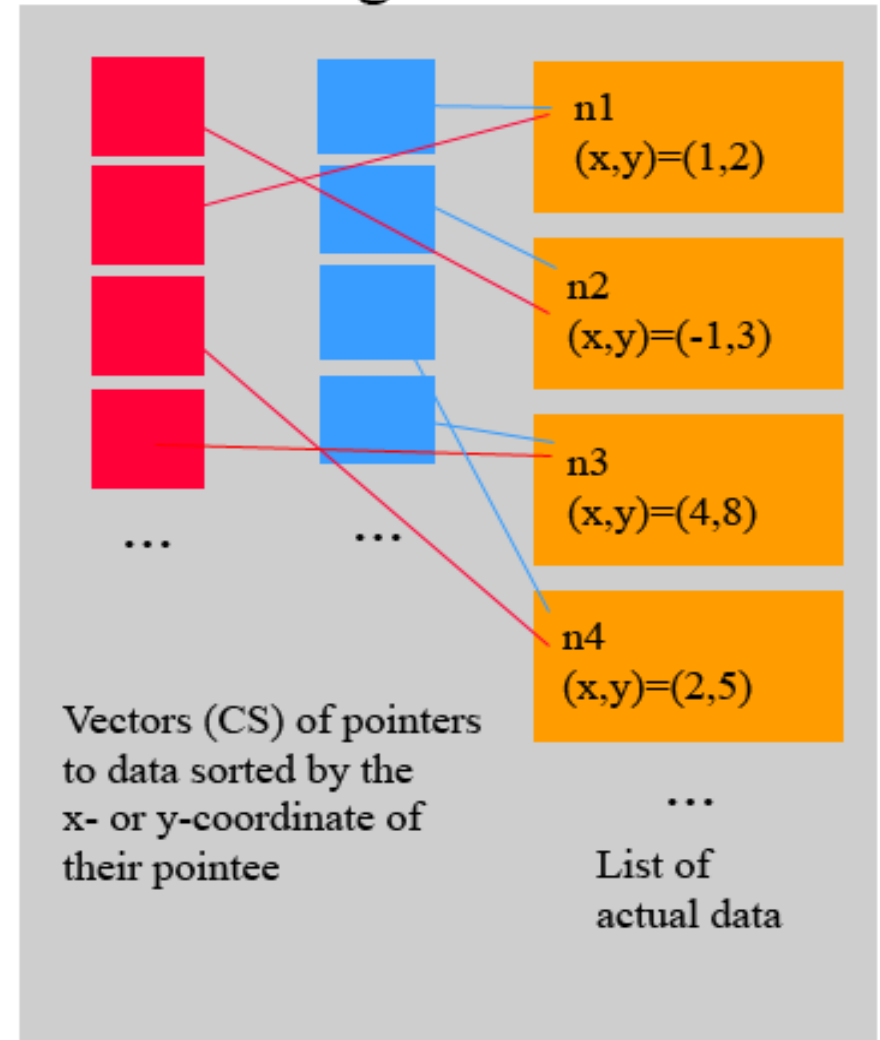
Demo 0:

- 1) Introducing the lattice gas;
- 2) Analytic description of the lattice gas;
- 3) Program objectives;
- 4) **How to implement it (efficiently);**
- 5) Demonstrations;

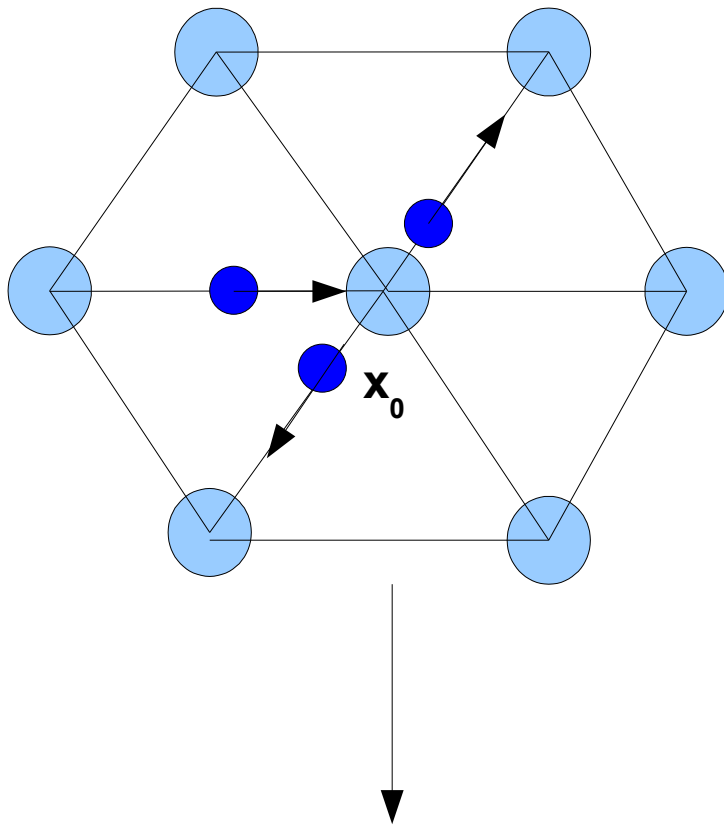
# Node management

- Dynamic node generation:
  - How does each node – acting very independently – know about and connect to its neighbors?
- How do we achieve this faster than  $O(n^2)$ , which is why we moved away from the continuous space calculation?
  - Coming up with this solution and its implementation was the most challenging part of this assignment.

## NodeManager



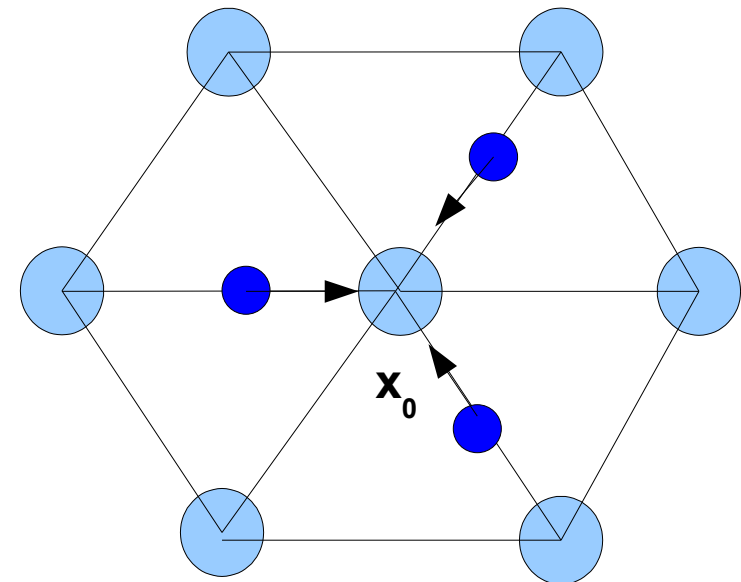
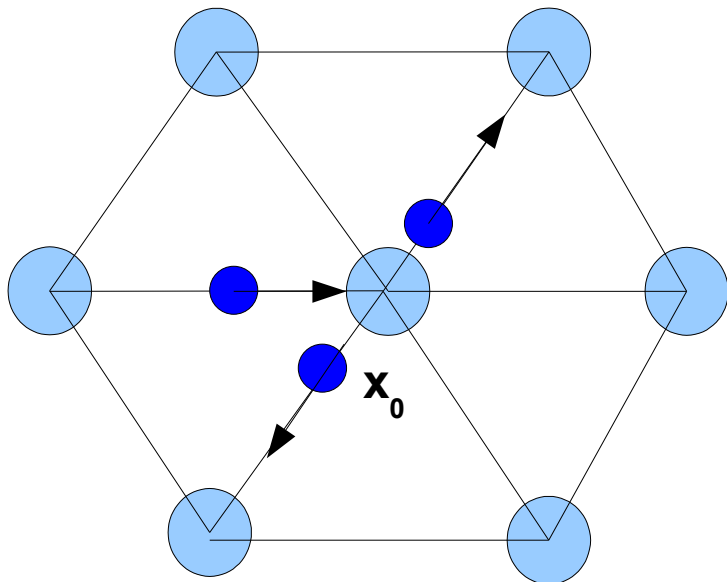
# Boolean operations at the bit level



- Take advantage of the fact that occupancy is represented by a boolean variable (0 or 1).
- Represent occupancy by using 6-bits of a byte.



# Example of a three-body head-on collision calculation



x	x	0	1	0	0	1	1
7	6	5	4	3	2	1	0

x	x	0	1	0	1	0	1
7	6	5	4	3	2	1	0

Does the scenario on the left correspond to a three-body head-on collision (see right)?

# Three-body head-on collision

Actual configuration:

x	x	0	1	0	0	1	1
7	6	5	4	3	2	1	0

Heads-on three-body collision state:

x	x	0	1	0	1	0	1
7	6	5	4	3	2	1	0

Bitwise AND (&):

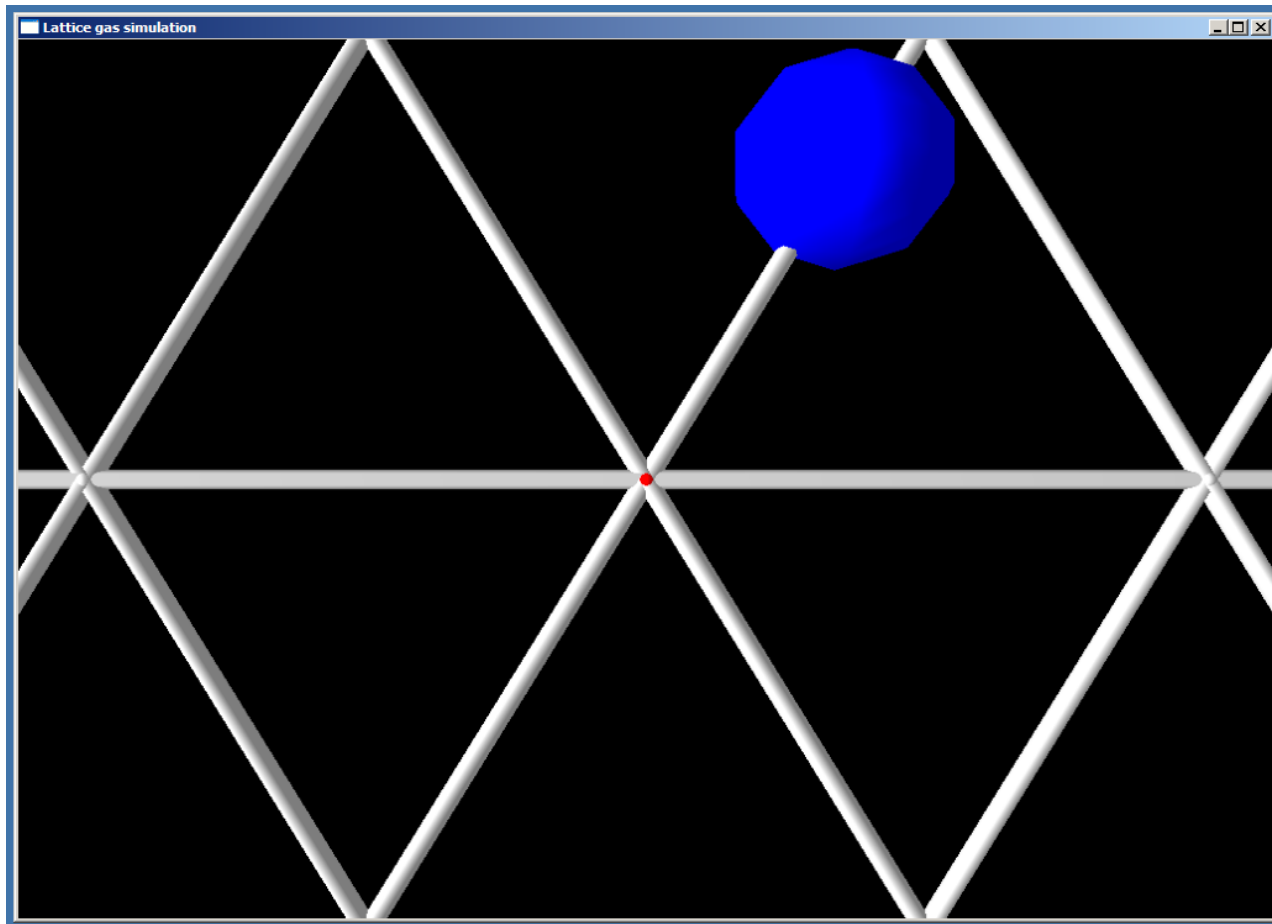
x	x	0	1	0	0	0	1
7	6	5	4	3	2	1	0

- 1) “Mask” (bitwise AND) the actual configuration with the candidate scenario;
- 2) Bitwise comparison of the masked result to the candidate scenario.
- 3) If equivalent, then we have a three-body head-on collision as shown.

# Huge advantage over continuous simulations

- Calculation of a *three-body collision* has been reduced to two primitive operations:
  - Masking;
  - Equality checking;
- Furthermore, the number of calculations per frame is **NOT** dependent on particle number!
  - Instead, it depends on the number of *nodes*;
  - The number of computations increases only **linearly**, once the network is configured!

# So what do we do with the “extra” processing power?



- Squander it on rendering!

- 1) Introducing the lattice gas;
- 2) Analytic description of the lattice gas;
- 3) Program objectives;
- 4) How to implement it (efficiently);
- 5) **Demonstrations;**