# Constructing a neighbour list

- In MD simulations (and actually many other applications) one of the central operations is the distances between atoms. In MD this is needed in the force calculation

- Trivial calculation of distances between atoms:

```
do i=1,N
   do j=1,N
      if (i==j) cycle
      dx=x(j)-x(i);
      dy=y(j)-y(i);
      dz=z(j)-z(i);
      rsq=dx*dx+dy*dy+dz*dz
      r=sqrt(rsq)
   enddo
enddo
```

- This algorithm is $O(N^2)$, i.e. very slow when N -> ∞.

- But in practice we know the atoms move < 0.2 Å/time step. So a large fraction of the neighbours remain the same during one time step, and it seems wasteful to recalculate which they are every single time.

- **Solution:** Verlet neighbour list:

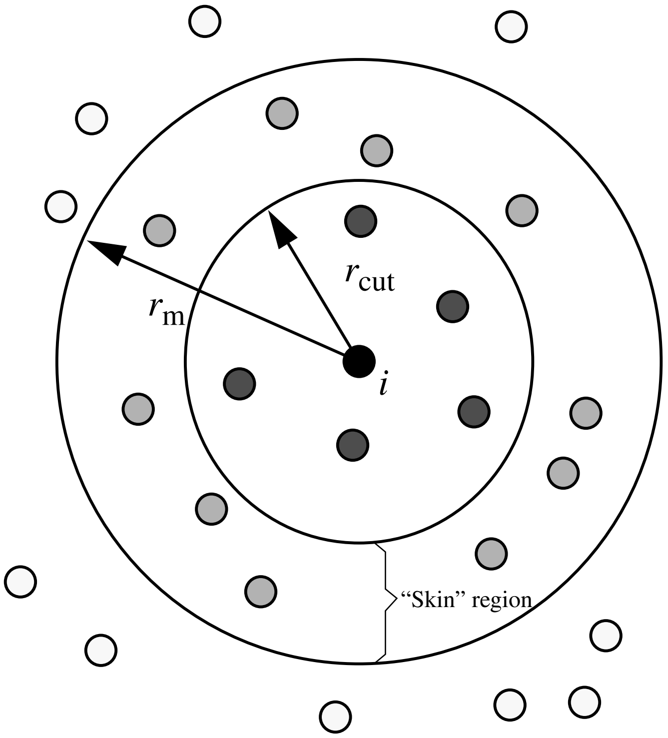
  - Make a list which contains for each atom $i$ the indices of all atoms $j$ which are closer to $i$ than a given distance $r_m$. $r_m > r_{cut}$, the cut-off distance of the potential

  - The list is updated only every $N_m$ time steps.

  - $r_m$ and $N_m$ are chosen such that

    $r_m - r_{cut} > N_m \bar{v} \Delta t$ ,

    where $\bar{v}$ is a typical atom velocity and $\Delta t$ the time step



- An even better way to choose when to update the interval: after the neighbour list has been updated, keep a list of the maximum displacement of all atoms:

  - Make a separate table `dxnei(i)`
  - When you move atoms, also calculate `dxnei(i)=dxnei(i)+dx`
  - Calculate the **two** maximal displacements of all atoms:

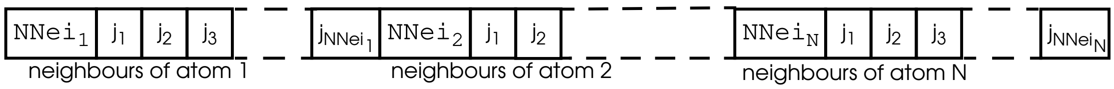
```
drneimax=0.0; drneimax2=0.0
do i=1,N
   drnei=sqrt(dxnei(i)*dxnei(i)+dynei(i)*dynei(i)+dznei(i)*dznei(i)
   if (drnei > drneimax) then
       drneimax2=drneimax
       drneimax=drnei
   else
       if (drnei > drneimax2) then
          drneimax2=drnei
       endif
   endif
enddo
```

  - Now when `drneimax+drneimax2` > $r_m$-$r_c$ the neighbour list has to be updated.

  - When the update is done, do `dxnei(i)=0.0` !

  - This alternative has two major advantages: the simulation does not screw up if one atom suddenly starts to move much faster than the average, and if the system cools down, the neighbour list update interval keeps increasing.

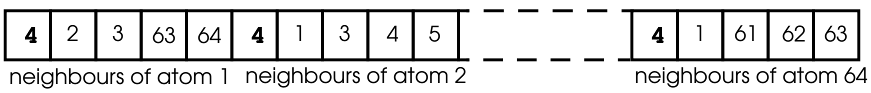
- In practice the neighbour list can look e.g. like the following:




  - Here $\mathrm{NNei}_1$ is the number of neighbours of atom *i*

  - $j_1$, $j_2$, ... are the indices of neighbouring atoms

- So, if we would have a 64 atom system, where every atom has exactly 4 neighbours, the neighbour list could e.g. look like the following:



- A practical implementation of creating the list:

```
startofineighbourlist=1
do i=1,N
   nneighboursi=0
   do j=1,N
      if (i==j) cycle
      dx=x(j)-x(i);
      dy=y(j)-y(i);
      dz=z(j)-z(i);
      rsq=dx*dx+dy*dy+dz*dz

      if (rsq <= rskincutsq) then
         nneighboursi=nneighboursi+1
         neighbourlist(startofineighbourlist+nneighboursi)=j
      endif

   enddo
   ! Write in number of i's neighbours into list
   neighbourlist(startofineighbourlist)=nneighboursi
   ! Set starting position for next atom
   startofineighbourlist=startofineighbourlist+nneighboursi+1
enddo
```

- Note that periodic boundaries have been omitted here for brevity!
- With the neighbour list, we can achieve a savings of a factor $N_m$ in calculating the distances to neighbours.
- But even using the neighbour list, our algorithm is still $O(N^2)$: When $N$ is growing, at some point creating the neighbour list will take a substantial fraction of the total simulation time.

## Linked list / cellular method

- Using a linked list and cellular division of the simulation cell, we can make the algorithm truly *O(N)*:

  - Let's divide the MD cell into smaller subcells: $M \times M \times M$ cells

  - The size of one subcell $l$ is chosen so that

$$l = \frac{L}{M} > r_{\mathrm{m}} \qquad ,$$

  where $L$ = the length of the MD cell, and $r_m$ is as above.

  - Now when we look for neighbours of atom $i$ we only have to look through the subcell where $i$ is, and its neighbouring subcells, but not the whole simulation cell! For instance if atom $i$ is in cell 13:

| | | | | |
|---|---|---|---|---|
| 21 | 22 | 23 | 24 | 25 |
| 16 | 17 | 18 | 19 | 20 |
| 11 | 12 | 13 | 14 | 15 |
| 6 | 7 | 8 | 9 | 10 |
| 1 | 2 | 3 | 4 | 5 |

- The average number of atoms in a subcell is $N_{\mathrm{c}} = N / M^3$.

  $\Rightarrow$ We have to go through $27 N N_{\mathrm{c}}$ atom pairs instead of $N(N-1)$ !

- For some interaction potentials (symmetric *ij* pairs) it is actually enough to calculate every second neighbour pair (e.g. `i > j`) whence the number of pairs is further reduced by a factor of 2.

- A practical implementation:

- array HEAD: - size = $M^3$
  - contains pointers to the table LIST
  - tells where the neighbours in subcell $m$ start

- array LIST: - size = $N$
  - element $j$ tells where the next atom index of atoms in this cell is

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| HEAD: | 8 | 10 | | | | | | | | |
| LIST: | 0 | 1 | 0 | 3 | 2 | 4 | 5 | 7 | 6 | 9 |

- So the above example means that subcell 2 contains atoms 9, 6, 4, 3 and 0
- If you think about it a bit, you'll realize that this representation is indeed enough to give all the atoms in all cells. A twodimensional array would of course also work, but would require much more memory, or dynamic allocation, both of which are less efficient.

- Building the list:
  - For simplicity let's assume a cubic case:
  - MD cell size = size(3)
  - Size of subcell =size()/M
  - MD cell centered on origin

```
do i=1,N
      head(i) = 0
enddo
do i=1,N
      icell = 1 + int((x(i)+size(1)/2)/size(1)*M) &
                  int((y(i)+size(2)/2)/size(2)*M) * M &
                  int((z(i)+size(2)/2)/size(3)*M) * M * M
      list(i) = head(icell)
      head(icell) = i
enddo
```

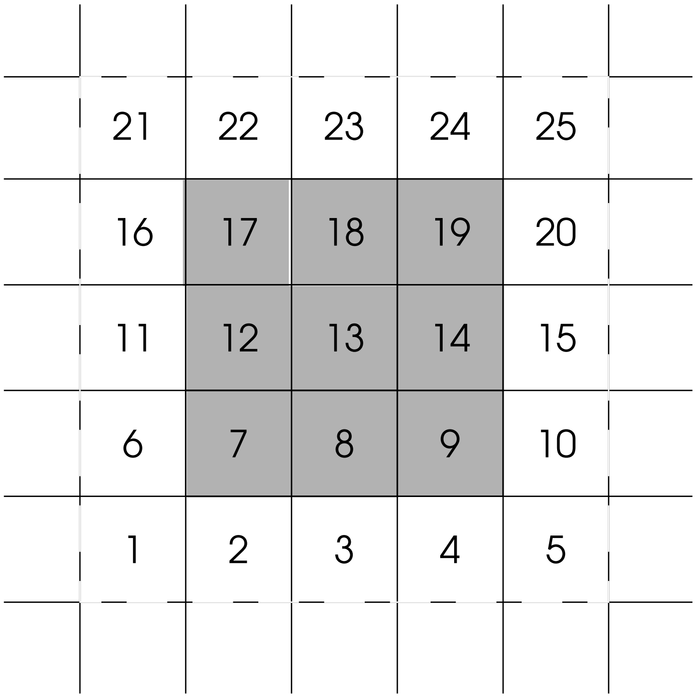
- So the list “LIST” is filled in reverse order to the picture above.

- The above algorithm requires periodic boundaries. If the boundaries are open, an atom may get outside the cell borders, and the `icell` may point to the wrong cell.

- To account for possibly open periodic boundaries properly things get a bit trickier:
  - MD Cell size `size(3)`
  - MD cell centered on origin
  - Number of cells in different dimensions `Mx, My, Mz`
  - Cell range `0 - Mx-1` and same in y and z

```
do i=1,N
      dx=x(i)+size(1)/2
      ! Check that we are really inside boundaries
      if (periodic(1) == 1 .and. dx < 0.0) dx=dx+size(1)
      if (periodic(1) == 1 .and. dx > size(1)) dx=dx-size(1)
      ix=int((dx/size(1))*Mx)
      ! If not periodic, let border cells continue to infinity
      if (periodic(1) == 0) then
           if (ix < 0) ix=0
           if (ix >= Mx) ix=Mx-1
      endif
      (and same thing for y and z)
      icell=(iz*My+iy)*Mx+ix
      list(i)=head(icell)
      head(icell)=i
enddo
```

  - So the subcells at open boundaries continue out to infinity:



- Usually the linked list (LIST, HEAD) is used to generate a Verlet list

- Decoding a linked list into a Verlet-list, as pseudocode:
  - Cell size `size(3)`
  - Number of cells `Mx, My, Mz`

```
do i=1,N
   do (Loop over 27 neighbouring cells: inx iny inz)
      icell=(inz*My+iny)*Mx+inx
      ! Get first atom in cell
      j=head(icell)
      do
         if (j==0) exit ! exit from innermost loop
         (get distance r between atoms i and j)
         if (r <= rneicut) then
            (accept neighbour)
         endif
         j=list(j)
      enddo
   enddo
enddo
```