## Constructing a neighbour list

- In MD simulations (and actually many other applications) one of the central operations is the calculation of distances between atoms.
  - In MD this is needed in the energy and 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 $N \to \infty$.
  - 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.

## Constructing a neighbour list

- **Solution:** Verlet[1] neighbour list:

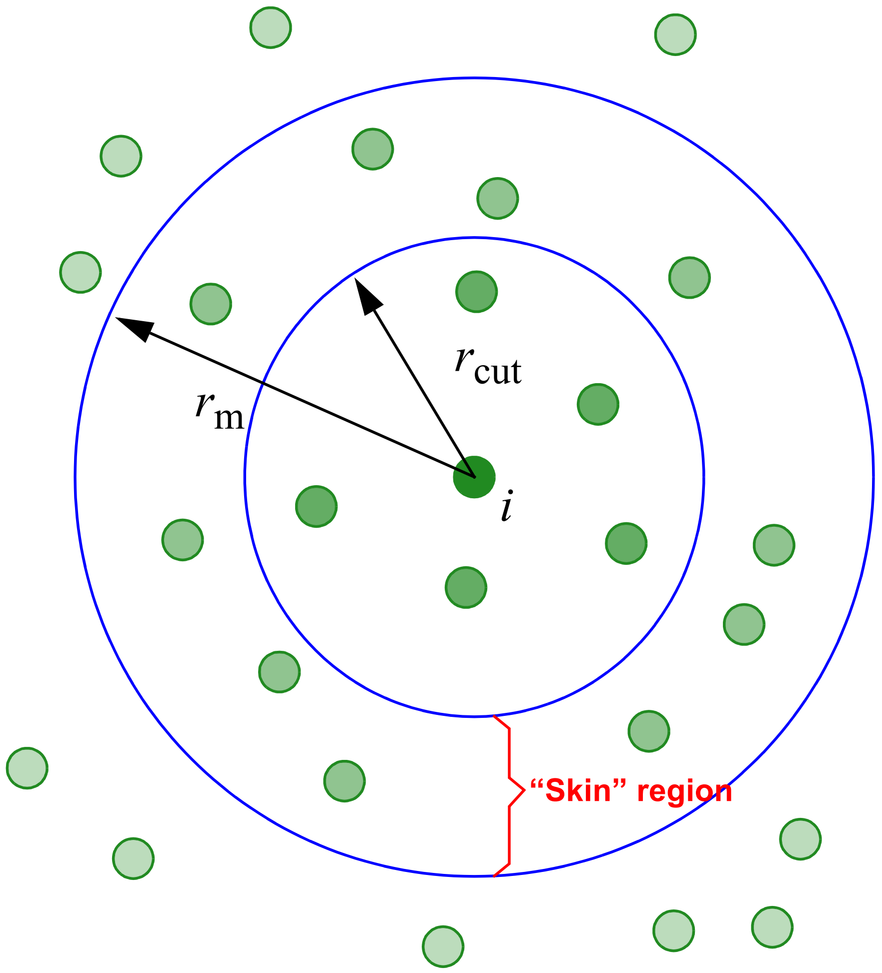
  - Make a list which contains for each atom *i* the indices of all atoms *j* which are closer to *i* than a given distance $r_m$. $r_m > r_{cut}$, the cutoff distance of the potential

  - 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



1. Loup Verlet, *Phys. Rev.* **159** (1967) 98.

## Constructing a neighbour list

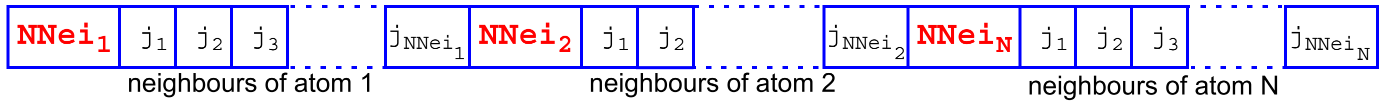
- An even better way to choose when to update the interval: after the neighbour list has been updated, keep a list of the maximum displacement of all atoms:
    - Make a separate table `dxnei(i)`
    - When you move atoms, also calculate `dxnei(i)=dxnei(i)+dx`
    - Calculate the **two** maximal displacements 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 $(\texttt{drneimax+drneimax2}) > r_{\mathrm{m}} - r_{\mathrm{cut}}$ 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.

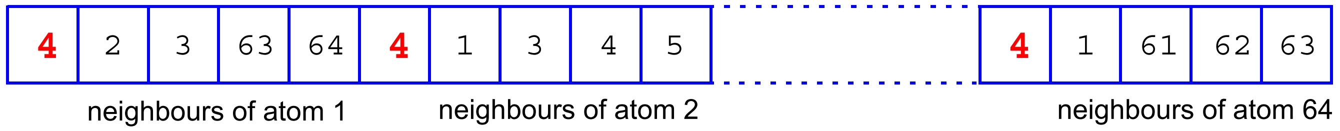
## Constructing a neighbour list

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


$NNei_1$ $j_1$ $j_2$ $j_3$ $j_{NNei_1}$
$NNei_2$ $j_1$ $j_2$ $j_{NNei_2}$
$NNei_N$ $j_1$ $j_2$ $j_3$ $j_{NNei_N}$
neighbours of atom 1
neighbours of atom 2
neighbours of atom N

- Here $NNei_i$ is the number of neighbours of atom $i$.
- $j_1$, $j_2$, ... are the indices of neighbouring atoms (different for different atoms).

- So, if we would have a 64 atom system, where every atom has 4 neighbours, the neighbour list could look like this:



## Constructing a neighbour list

- A practical implementation of creating the list:

```
nlistbeg=1
do i=1,N
    nnei=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
            nnei=nnei+1
            nlist(nlistbeg+nnei)=j
        endif
    enddo
    nlist(nlistbeg)=nnei                      ! Write in number of i's neighbours into list
    nlistbeg=nlistbeg+nnei+1                  ! Set starting position for next atom
enddo
```

Periodic boundaries omitted for brevity. See lecture02 for how to include them in the dx, dy, dz calculations.

- With the neighbour list, we can achieve a savings of a factor $N_\mathrm{m}$ in calculating the distances to neighbours.

- But even using the neighbour list, our algorithm is still $O(N^2)$.

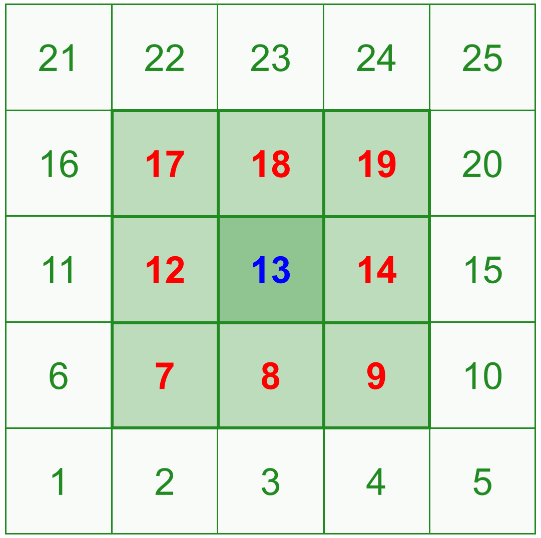
## Constructing a neighbour list

- Remedy: 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

$$l = \frac{L}{M} > r_m,$$

where $L$ = the size 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:



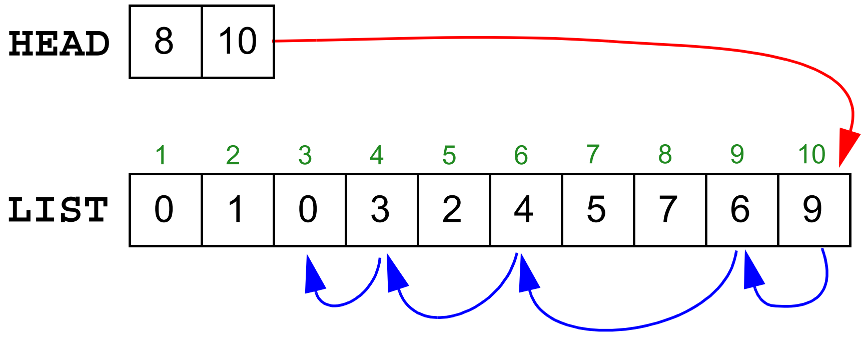
The average number of atoms in a subcell is $N_c = N/M^3$.

$\Rightarrow$ We have to go through $27NN_c$ 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.

## Constructing a neighbour list

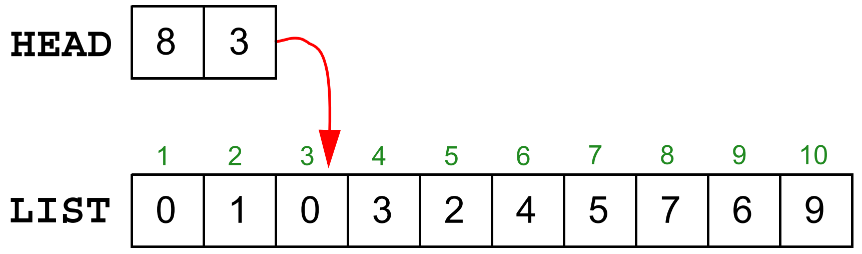
- A practical implementation:
  - array **HEAD**:
    - size = $M^3$
    - contains pointers to the table **LIST**
    - tells where the neighbours in subcell $m$ start
  - array **LIST**:
    - size = $N$
    - element $j$ tells where the next atom index of atoms in this cell is



- So the example below means that subcell 2 contains atoms 10, 9, 6, 4, and 3
- This representation is indeed enough to give all the atoms in all cells.
- A two dimensional array would of course also work, but would require much more memory, or dynamic allocation, both of which are less efficient.

## Constructing a neighbour list

- Building the list:
  - 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.

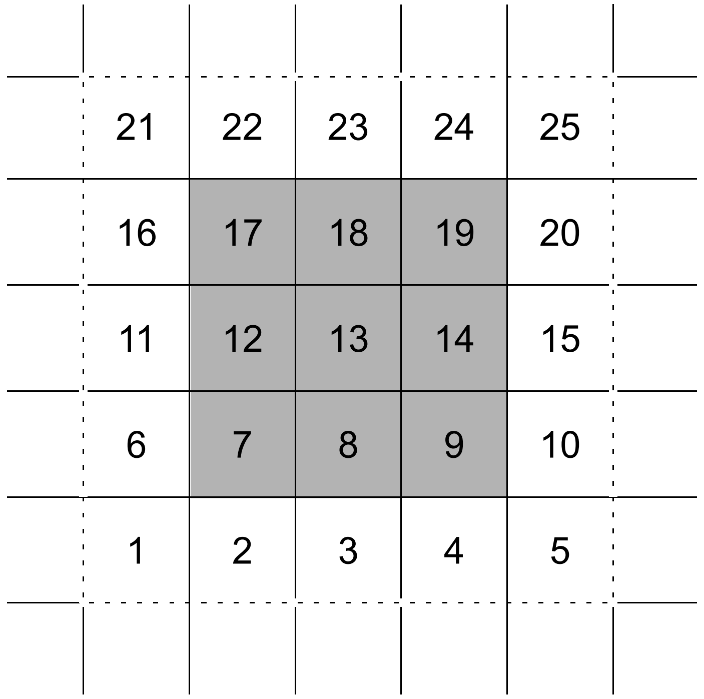
- The above algorithm requires periodic boundaries. If the boundaries are open, an atom may get outside the cell borders, and the **`icell`** may point to the wrong cell.

## Constructing a neighbour list

- To account for possibly open 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:



## Constructing a neighbour list

- 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
```

## MD code `mdmorse`

- A simplified MD code **`mdmorse`** has been written for this course:
  - **`mdmorse`** simulates atom motion in a variety of metals (but only one metal at a time) with a simple Morse pair potential model.

$$V(r) = D[e^{-2\alpha(r-r_0)} - 2e^{-\alpha(r-r_0)}]$$

  - The code has a Verlet neighbour list (but not a linked list) and the equations of motion are solved with the velocity Verlet method.
  - The code is given in Fortran90 and C.
- The code can be downloaded from the course web page.
  - The code has the input parameter and output routines included.
  - Physically interesting subroutines have been removed from the code, so it does not work.
  - During the course exercises, you get the task of writing the missing subroutines.
  - Solutions will be provided and explained during the exercise sessions.
  - You may either use your own or the provided solutions afterwards.

## Structure of the `mdmorse` code

- Program files:

| | |
|---|---|
| `main.f90` | Main program |
| `inout.f90` | Miscellaneous input and output stuff |
| `modules.f90` | Global variables |
| `physical.f90`* | Calculating $T$ and $E$, and random number generators |
| `neighbourlist.f90`* | Getting the neighbour list |
| `solve.f90`* | Solving the equations of motion |
| `forces.f90`* | Calculating the forces |
| `Makefile` | Makefile (If you have used Unix or Linux systems you should know how to **`make`** programs.) |

  - Files marked with * contain the subroutines which are to be filled up during the exercises

- C version: `*.c` $\rightarrow$ `*.f90`
  `modules.f90` $\rightarrow$ `global.h`

- Compiling the code:

```
make
```

  - This has been tested to work at least on Linux systems with a GNU compilers (`gfortran` and `gcc`).
  - You may have to change the compiler command in `Makefile`.

## Structure of the `mdmorse` code

- Input files (file names are hardcoded):

| | |
|---|---|
| `mdmorse.in` | Miscellaneous parameters |
| `atoms.in` | Atom coordinates in XYZ format |

- Running the program:

`./mdmorse` (or if you don't want to disturb other users `nice ./mdmorse`)

  - Should be done in the same directory where the input files are.

- Output files:

| | |
|---|---|
| `standard output` | $T$, $E$, $P$ and other interesting output |
| `atoms.out` | Atom coordinates at regular intervals |

  - Note also that during the program running, the code writes out a large number of atom coordinates to a file `atoms.out`, which may grow very large.

## Structure of the `mdmorse` code

- Input file `mdmorse.in`

```
 Sample input file for mdmorse md program
 File format: -identifier, then value. Rest is arbitrary comments
 Lines which do not begin with "-" are all ignored

Identifier     | Value

-initialT       600.0     Initial temperature

-desiredT     300.0        Variables for temperature control
-btctau       0.0          If btctau=0 no effect

-bpctau       0.0          Variables for pressure control
-bpcbeta      7.0e-4       If bpctau=0 no effect
-desiredP     0.0

-mass           63.546       For Cu

-xsize          18.126900793    Box size in each dimension
-ysize          18.126900793
-zsize          18.126900793

-periodicx      1               1 = periodic, 0 = non
-periodicy      1
-periodicz      1

-morseDe        0.3429       Morse potential parameters for Cu
-morsealpha     1.3588
-morseRe        2.866

-rpotcut        5.0          Potential cutoff
-rskincut       6.0          Neighbour list cutoff, must be > rpotcut

-nupdate        5               Number of steps between neighbour list updates

-nmovieoutput   100             Interval between atom movie output

-deltat         2.0          Time step in simulation in fs
-tmax           10000.0     Total simulation time
```

## Structure of the `mdmorse` code

- Input file `atoms.in`
  - The file is a normal XYZ atom coordinate file:

```
500
FCC cell made by makeFCC with a= 3.615 n= 5 5 5
Cu      -8.13375      -8.13375      -8.13375
Cu      -6.32625      -6.32625      -8.13375
```

*...and so forth the remaining 498 atom coordinates....*

```
Cu       6.32625       8.13375       8.13375
Cu       8.13375       6.32625       8.13375
```

- Note that the cell is centered on the origin.

## Structure of the `mdmorse` code

- Standard output (for the working code; F90 version):

```
 --------------- mdmorse V1.0 --------------------

Read in parameter -initialT        value  1000.00
Read in parameter -desiredT        value  2500.00
Read in parameter -btctau          value  300.000
Read in parameter -bpctau          value  3000.00
Read in parameter -bpcbeta         value 0.700000E-03
Read in parameter -desiredP        value  0.00000
Read in parameter -mass            value  63.5460
Read in parameter -xsize           value  18.1269
Read in parameter -ysize           value  18.1269
Read in parameter -zsize           value  18.1269
Read in parameter -periodicx       value  1.00000
Read in parameter -periodicy       value  1.00000
Read in parameter -periodicz       value  1.00000
Read in parameter -morseDe         value 0.342900
Read in parameter -morsealpha      value  1.35880
Read in parameter -morseRe         value  2.86600
Read in parameter -rpotcut         value  7.00000
Read in parameter -rskincut        value  8.00000
Read in parameter -nupdate         value  5.00000
Read in parameter -nmovieoutput    value  100.000
Read in parameter -deltat          value  5.00000
Read in parameter -tmax            value  50000.0
Using periodics (1=on, 0=off) 1 1 1
Morse potential parameters: De alpha Re    0.342900    1.358800    2.866000
Movie output selected every     100 steps

Reading in     500 atoms described as FCC cell made by makeFCC with a= 3.62538
 Initial atom temperature is    1970.4541462944828
Neighbour list update found    176.00     neighbours per atom
ec       5.000   1890.175     0.24432    -3.48740    -3.24307
bpc       5.000       26.025014  5956.400065997    18.127    18.127    18.127
Outputting atom movie at t =       5.000
ec      10.000   1652.943     0.21366    -3.45507    -3.24141
bpc      10.000       33.853085  5956.635315608    18.127    18.127    18.127
ec      15.000   1318.804     0.17047    -3.40893    -3.23846
bpc      15.000       43.555081  5956.937997643    18.128    18.128    18.128
```

## Structure of the `mdmorse` code

- And so on. Here most things are self-explanatory.
  - The "`ec`" and "`bpc`" lines contain the physically most interesting stuff in the following format:

|  | time(fs) | $T$ (K) | $E_{kin}$/at. | $E_{pot}$/at. | $E_{tot}$/at. | $P$(kbar) | *(energies in eV)* |
|---|---|---|---|---|---|---|---|
| `ec` | `4.000` | `594.069` | `0.07538` | `-3.03868` | `-2.96330` | `163.82195` |  |

|  | time(fs) | $b_x$(Å) | $b_y$(Å) | $b_z$(Å) | $V$(Å$^3$) | $P$(kbar) | $\mu_{Berendsen}$ |
|---|---|---|---|---|---|---|---|
| `bpc` | `4.000` | `18.132452` | `18.132452` | `18.132452` | `5961.69346` | `163.82195` | `1.00015` |

- Output file `atoms.out`
  - This file is in the XYZ format, but with the exception that column 5 contains the atom potential energy:

```
500
mdmorse atom output at time       2.000 fs boxsize      18.1269      18.1269      18.1269
Cu     -9.053407   -9.061041   -9.048299  -3.085270
Cu     -7.236810   -7.239921   -9.048988  -3.033905
Cu     -7.241191   -9.049845   -7.246436  -3.035222
Cu     -9.038484   -7.238137   -7.241429  -3.031141
.
.
.
```

## Structure of the `mdmorse` code

- Testing the incomplete code:
  - Even though the code is not complete, it should compile and run in the intermediate stages.
  - The output should look something like:

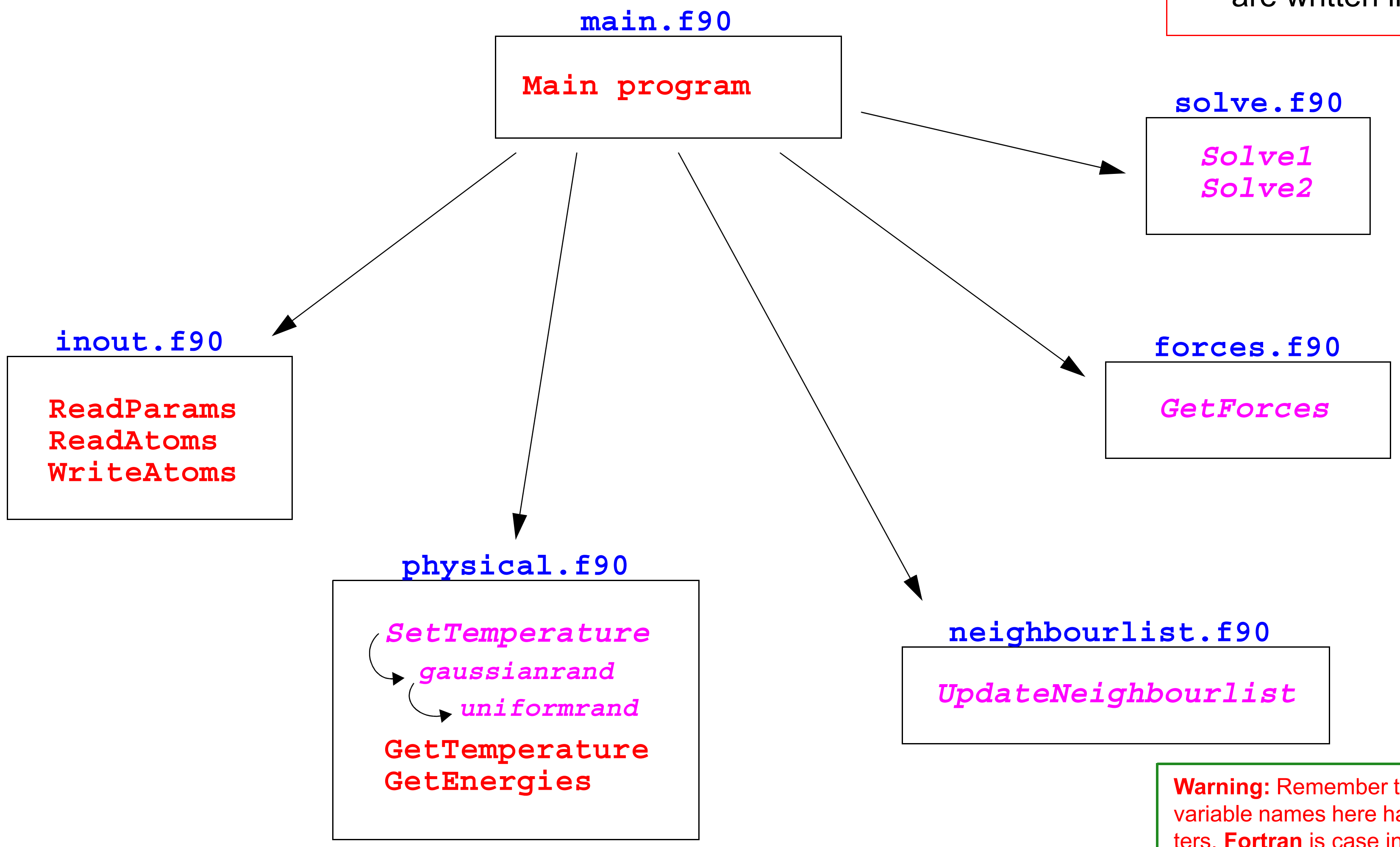
```
Reading in     500 atoms described as FCC Cu; boxsize      18.1000     18.1000
Initial atom temperature is   0.000000000000000
Neighbour list update found   0.26928E+06 neighbours per atom
ec       2.000      0.000     0.00000     0.00000     0.00000     0.00000
Outputting atom movie at t =      2.000
ec       4.000      0.000     0.00000     0.00000     0.00000     0.00000
```

  - I.e. the number of neighbours is nonsense, and the temperature is 0.

  - When you start doing the exercises, this should change and interesting things will start to happen.

- Structure of the program

Routines printed in *magenta* are written in exercises.



**Warning:** Remember that although routine and variable names here have small and capital letters, **Fortran** is case insensitive. I.e. symbols

SetTemperature
settemperature

refer to same routine (or variable).