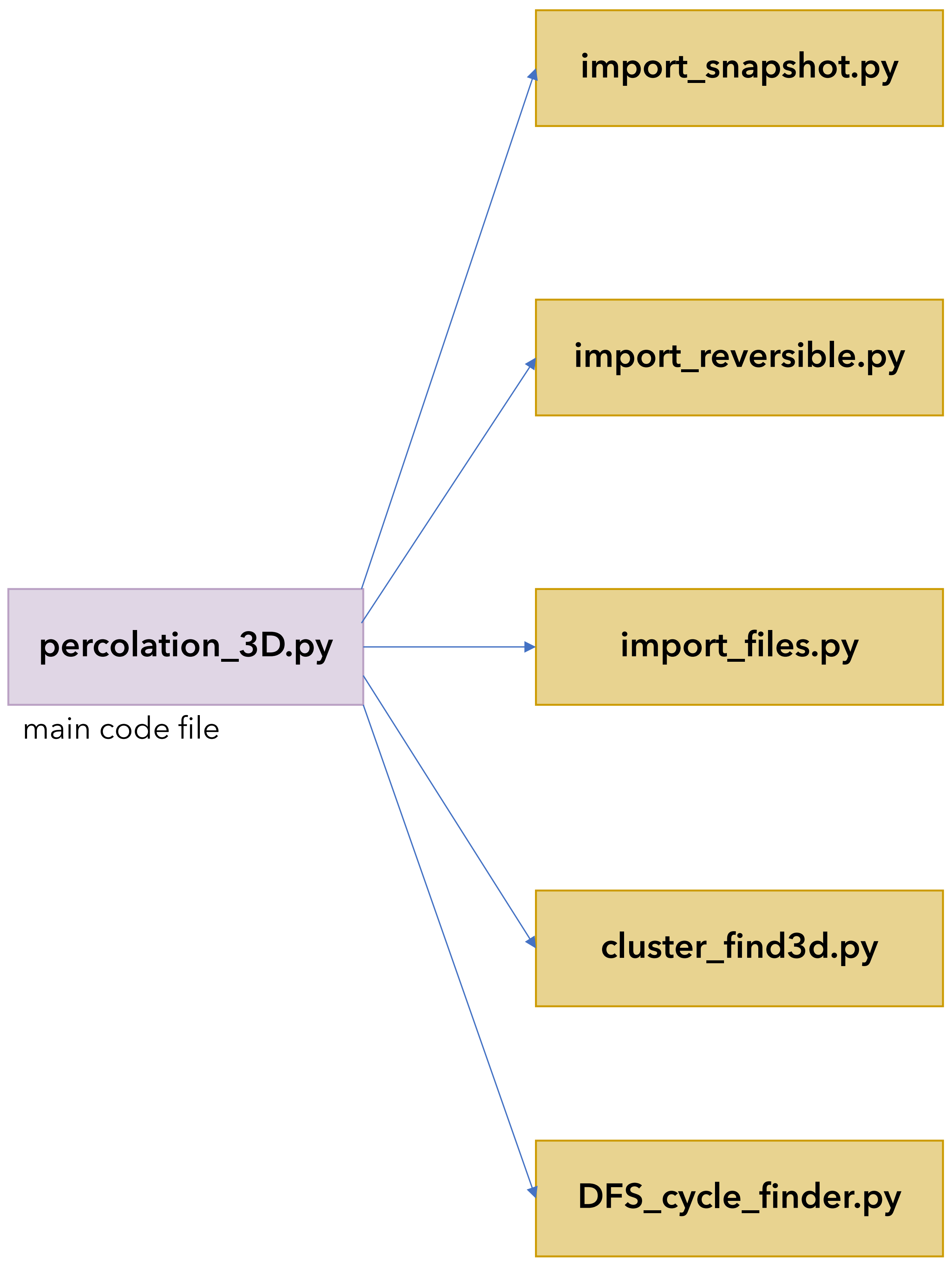
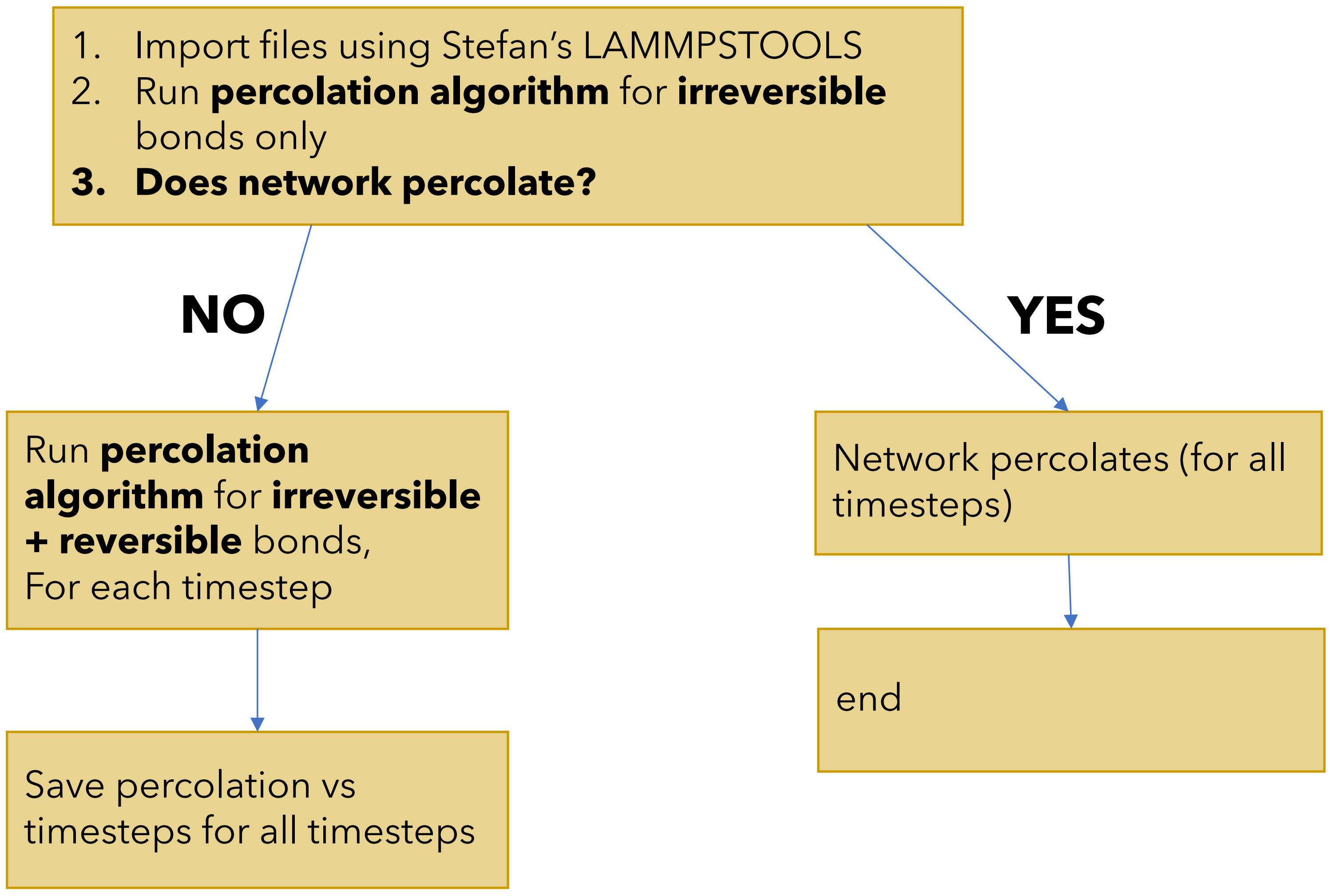
Percolation algorithm

Python files structure



Steps (general)



1. Import files using Stefan's LAMMPSTOOLS
2. Run **percolation algorithm** for **irreversible** bonds only
3. **Does network percolate?**

NO

Run **percolation algorithm** for **irreversible + reversible** bonds,
For each timestep

Save percolation vs timesteps for all timesteps

YES

Network percolates (for all timesteps)

end

Steps: 1

1. **`snapshot_map`** (in **`import_snapshot`**): makes a map from the endbead id to the star id; the resulting array is a `[NatomsTot * 1]` array: the `star_id (-1)` of a bead is found at `i=bead_id -1` in the array

2. **`map_coordinates`** (in **`import_files`**): function makes a map from the star id to its center coordinates (coordinates of the center bead of the star); the resulting array is a `[Nstars * 3]` array: the position `[x,y,z]` of a star with `star_id=i` is found at `star_coords[star_id - 1, :]`

3. **`store_crosslinks1`** (in **`import_files`**): function makes an `[N_stars*N_stars]` array where counter (0 or 1) `c(i,j)` shows if molecule `i` is bound to molecule `j`; **output**:
   - `crosslinks` - `[N_stars*N_stars]` array
   - `crosslink_boundaries` - `[N_stars*N_stars * 3]` array that shows, for each bond `(i,j)`, whether it is bound across the [x,y,z] box boundaries (direction +1 or -1) *uses **`crossings`**

`crossings` (in **`import_files`**): function checks distance between the center coordinates of molcule I and j across each dimension (x,y,z) and outputs +/- 1 if their distance is > abs(boxsize/2)

4. **`drop`** (in **`cluster_find3d`**): function checks the `crosslink_boundaries` array and makes a temporary crosslinks () array of ; **output**:
   - `crosslink_tmp` - `[N_stars*N_stars]` array (`crosslinks`) where crosslinks across the boundaries have been removed
   - `dropped_list` - `[N_dropped_bonds/2 * 5]` array that lists, for each dropped bond `(i,j)`, whether it is bound across the [x,y,z] box boundaries (direction +1 or -1) – `[i, j, x, y , z]`

Dropped bonds?

no → no percolation (END)

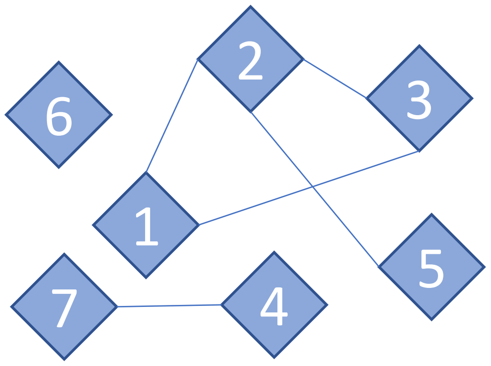
yes → proceed

Steps : neighbours/clusters

5. **`store_neighbours`** (in **`cluster_find3d`**): using the `crosslinks` (or `crosslinks_tmp`) info, makes an array with a list of neighbours for every star (see example)

number of arms (maximum bonds per star)

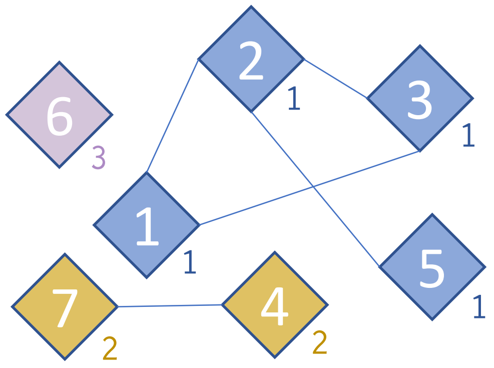
star ID ↓	bound star ID (1)	bound star ID (2)	bound star ID (...)	bound star ID (max)
1	2	3	-1	-1
2	1	3	5	-1
3	1	2	-1	-1
4	7	-1	-1	-1
5	2	-1	-1	-1
6	-1	-1	-1	-1
7	4	-1	-1	-1



*colouring algorithm:

6. **`cluster_find`** (in **`cluster_find3d`**): function makes an `[N_stars]` array where each star `i` is assigned a `colorID` ; **output**:
   - `list_color, Ncolors` – *uses **`coloring`**

`coloring` (in **`import_files`**): recursive function, assigns the same `colorID` to stars that are connected (i.e. part of the same cluster)



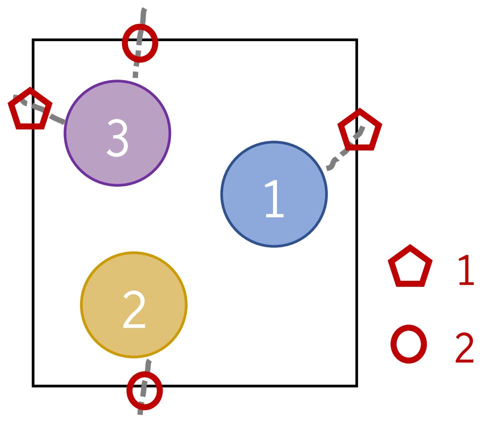
Steps : neighbours/clusters

*reintroduce dropped bonds, make neighbours list:

7. **`molid_to_clusterid`** (in **`cluster_find3d`**): uses `list_color` to change the starID in `dropped_list` (see 4.) to `clusterID` (i.e. color); returns `dropped_list`
8. **`neighbour_clusters`** (in **`cluster_find3d`**): similar to 5., uses `dropped_list` to form a neighbour list of clusters connected across the boundaries; ; **output**:
   - `Cneighbors` – `[Nclusters * max_links]` array with a list of neighbours for every cluster
   - `Ccrossings` - `[Nclusters * max_links * 3]` array that shows across what [x,y,z] box boundaries (direction +1 or -1) each "bond" is, for each "bond" between two clusters i and j (`[x,y,z]` crossing is found at position `[i,j]` in the array)
   - `edges_list – [Nclusters * max_links]` array, same shape as `Cneighbors`, but contains the id of the linkers (edges) between clusters (across the boundary)

`Cneighbors`

cluster ID	bound cluster ID (1)	bound cluster ID (2)	bound cluster ID (max)
1	3	-1	-1
2	3	-1	-1
3	1	2	-1



`Ccrossings`

cluster ID	bound cluster ID (1)	bound cluster ID (2)	bound cluster ID (max)
1	[1, 0]	[0,0]	[0,0]
2	[0,-1]	[0,0]	[0,0]
3	[-1, 0]	[0,1]	[0,0]

`edges_list`

cluster ID ↓	link ID (1)	link ID (2)	link ID (max)
1	1	-1	-1
2	2	-1	-1
3	1	2	-1

Steps : DFS algorithm

9. **`launch_dfs`** (in **`DFS_cycle_finder`**):
   `clusters=nodes`, `linkers=edges`
   i. initialises "blank" (everything set to -1) arrays: `visited_nodes` (length:`Nclusters`) and `visited_edges` (length: number of edges)
   ii. for all nodes in range `Nclusters`, checks if node was already visited:
      - ➢ YES: skip, next node
      - ➢ NO: proceed:
        - `crossing_sum = [[0,0,0]]` - every time I start visiting a new network (that has never been visited before), I set the sum of crossings to 0 (this is a counter of how many times I'm crossing the boundaries and in which dimension and direction, as I go through the search)
        - Also set to 0: `current_crossing`, `loops_temp` (temporary list of loops found in the current network), `discovery_timeC` (counter for when the cluster/node was "discovered"/touched), `discovery_timeE` (counter for when the an edge was "discovered"/touched),
        - START **`dfs`** function (recursive) – ends at the end of the tree
        - **`dfs`** updates `loops,discovery_timeC,discovery_timeE`
        - update `discovery_timeC`, add +1
        - `END: use` **`linearly_independent`** (in **`DFS_cycle_finder`**) to get the number of independent loops

`dfs` (in **`DFS_cycle_finder`**) – see code

10. **`launch_dfs`** returns the number of independent loops; does the network percolate in 3 independent directions?
    - ➢ YES: percolation!
    - ➢ NO: no percolation > add REVERSIBLE and check percolation for each timestep

Extra: reversible bonds

1. combine ends_to_star mas for reversible and irreversible ends
2. **`synchro`** (in **`import_files`**): synchronize timesteps of all files
3. find desired timestep
4. **`import_rev`** (in **`import_reversible`**): make list of end_beads ID that are bound (extracted from the "revBonds_count" file from LAMMPS) > output: `reversible_bonds, bound`
5. **`map_coords`** (in **`import_reversible`**): makes a map from the bead id to its center coordinates

6. **`make_revbondlist`** (in **`reversible_bonds`**): makes list of bonds by checking distance between all endbeads in `revList` (uses **`make_revbondlist`** (in **`reversible_bonds`**))
7. make total list for reversible + irreversible bonds
8. same as for irreversible bonds, check percolation