PSE Molekulardynamik
Sheet 3: XML, Linked-cell algorithm and “falling drop - Wall”

Group A: Daniel Schade, Ashutosh Solanki, Robin Cleve

07.06.2024

Table of contents

XML input

- Creation of **XML schema** file, organized elements in the following way
  - **General parameters** (output name, output frequency)
  - **Simulation parameters** (direct sum, linked cells)
  - **Particles and objects of particles to simulate** (single particles, cuboids, discs)
- Schema **validation** before reading
  - Check if files adhere to the specified XML schema before parsing them
  - Done for every file with the following declaration

```
xsi:noNamespaceSchemaLocation="../../src/fileHandling/reader/XMLHandling/ConfigurationFile.xsd"
```

- Use of **Tree-Mapping**
  - Ease of use, serialization back to DOM or XML, writing XML file back to disc

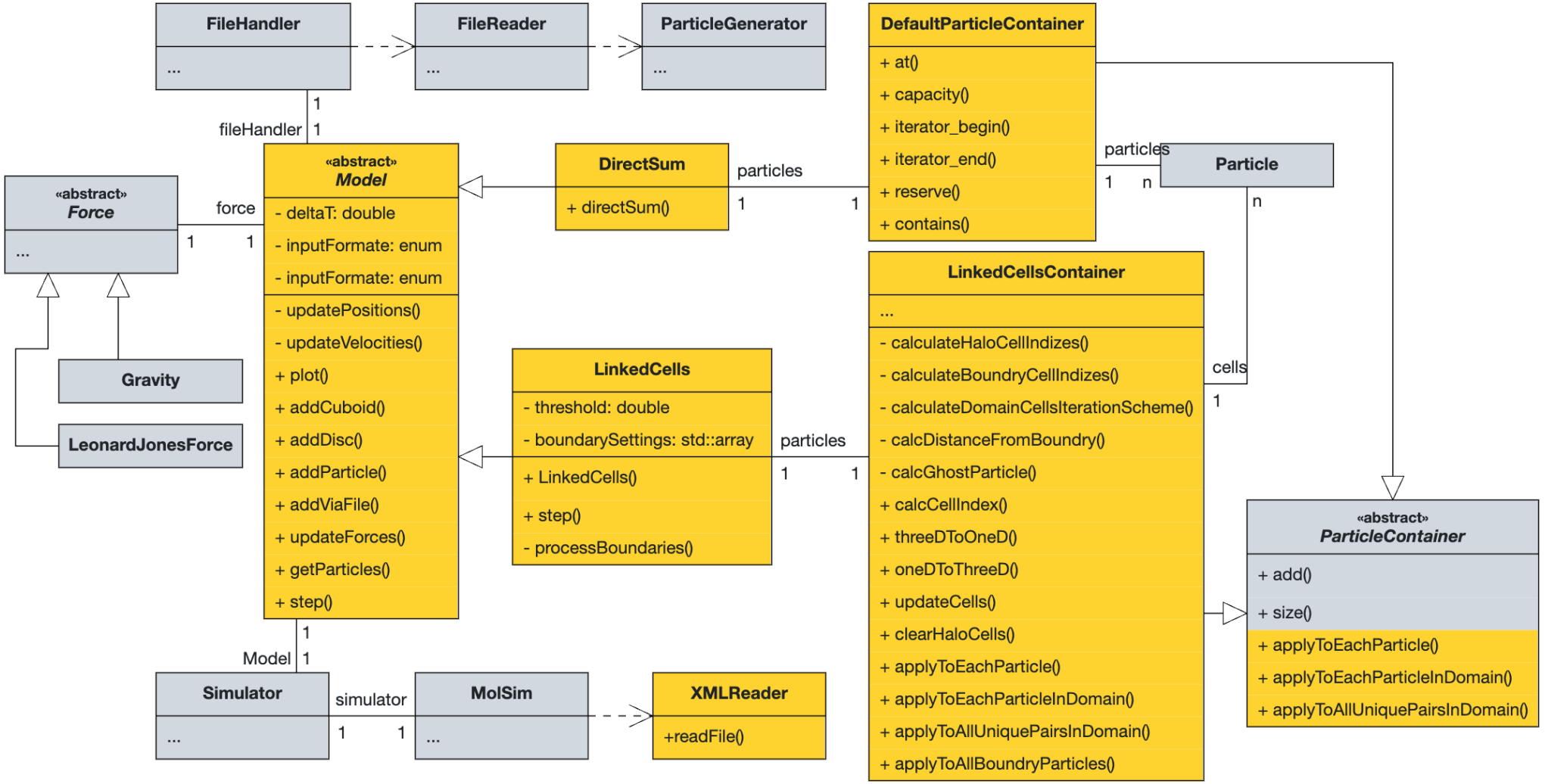
Linked-cell algorithm

- Division of the domain into cells

$$\text{cell\_size} = \frac{\text{domain\_size}}{\text{number of cells}} \qquad \text{number\_of\_cells} = \left\lfloor \frac{\text{domain\_size}}{\text{cut-off\_value}} \right\rfloor$$

- **Data Structure:** flattened 2D or 3D cell structure into one dimensional vector
- **2D Mode:** save memory in 2D simulations by omitting halo cells from third dimension
- **Iteration routes:** pre calculate iterations routs in constructor
- **Grouping of halo and boundary cells:** *front, back, left, right, top, bottom*

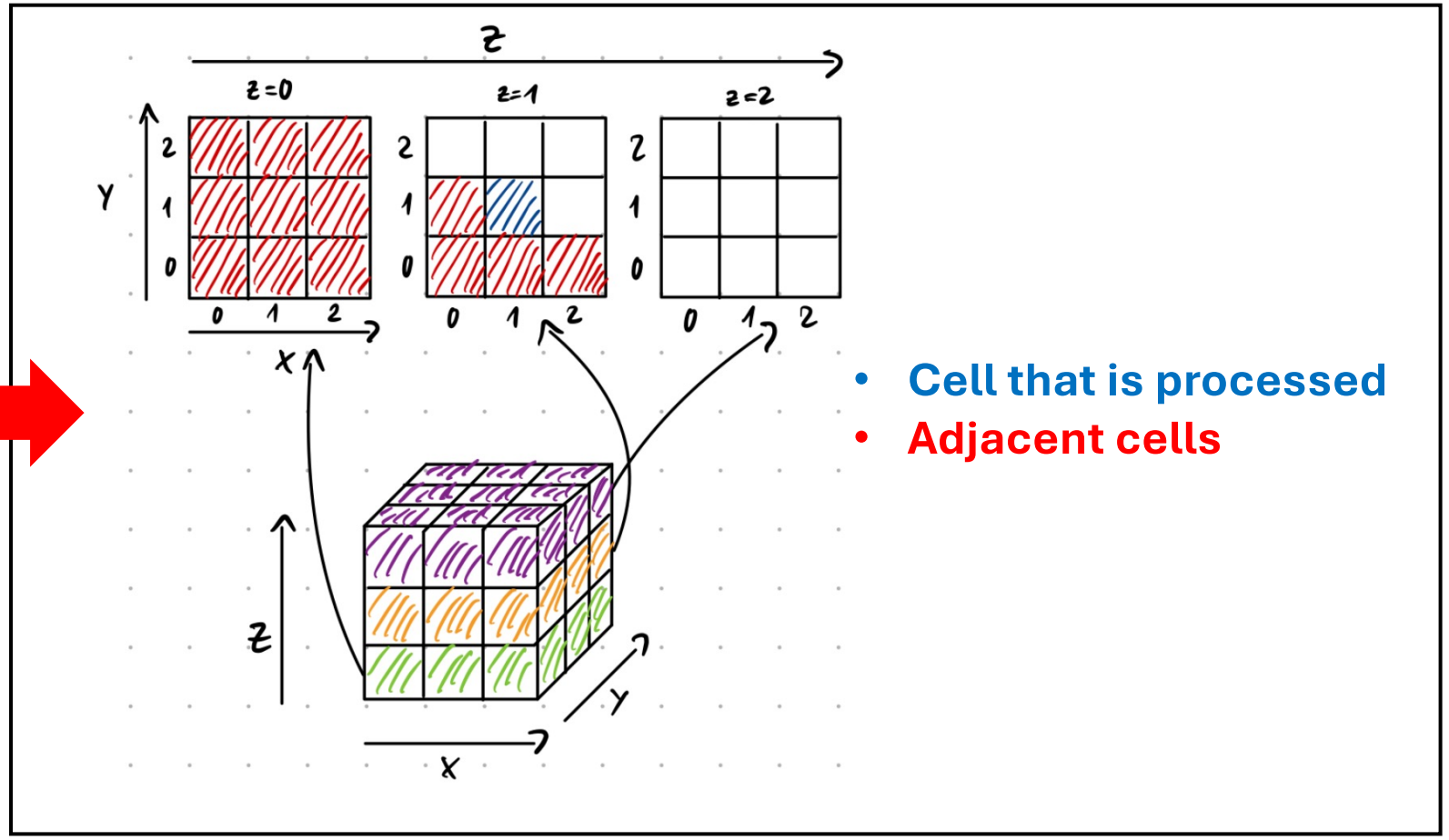
Linked-cell algorithm – Refactoring



Linked-cell algorithm – Iteration

Challenge:

“Iterate over unique pairs of particles”



- **Cell that is processed**
- **Adjacent cells**

Linked-cell algorithm – Implementation

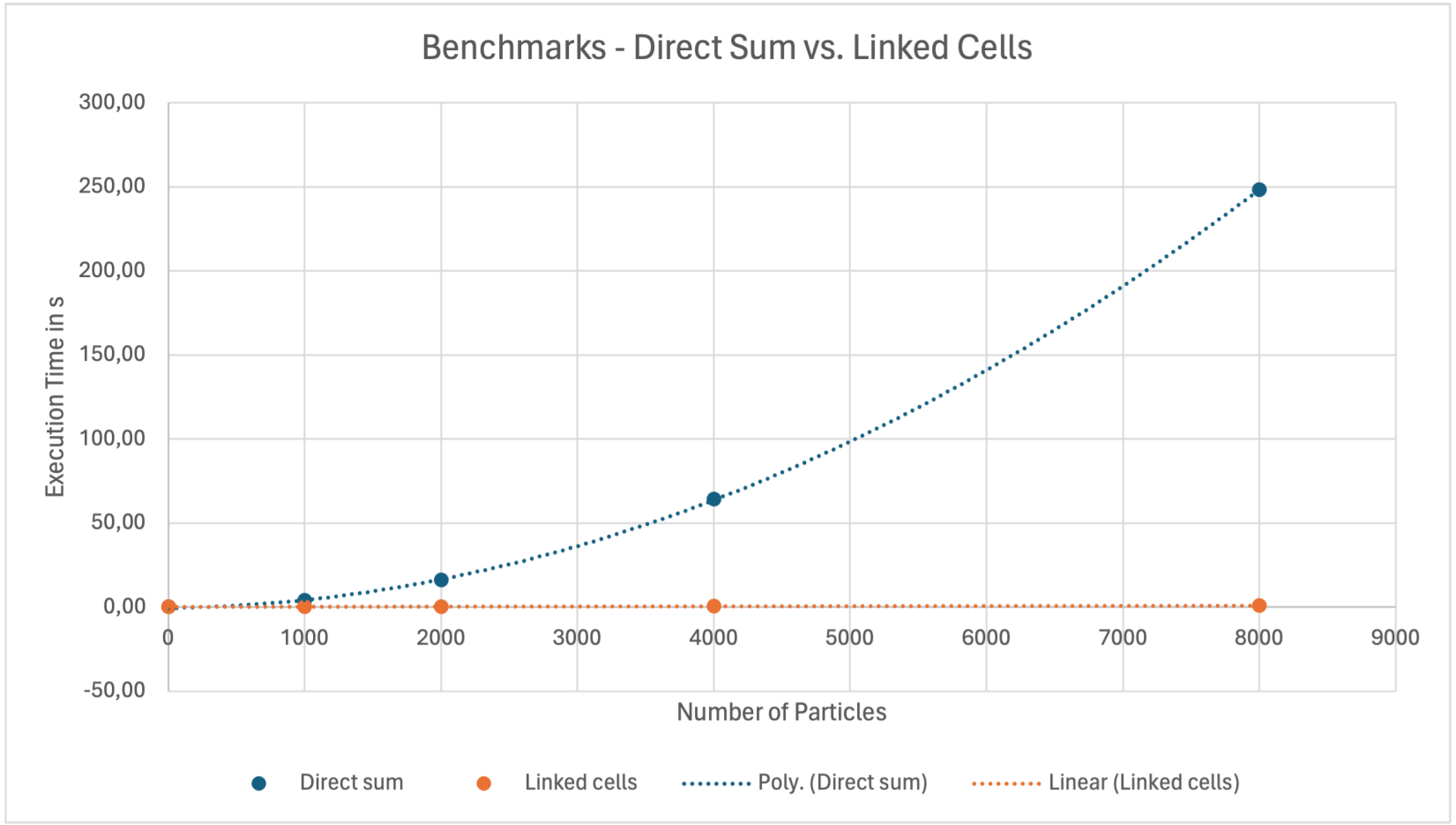
```
void LinkedCellsContainer::applyToAllUniquePairsInDomain(const std::function<void(Particle &, Particle &)> &function)
    for (auto& cellGroup: domainCellIterationScheme) {
        //First, consider all pairs within the cell that distance is smaller or equal then the cutoff radius
        for (auto p_i = cells[cellGroup[0]].begin(); p_i != cells[cellGroup[0]].end(); std::advance(p_i, 1)) {
            for (auto p_j = std::next(p_i); p_j != cells[cellGroup[0]].end(); std::advance(p_j, 1)) {
                if (ArrayUtils::L2Norm(p_i->getX() - p_j->getX()) <= rCutOff) {
                    function(*p_i, *p_j);
                }
            }
        }
        //Then, consider all relevant neighbour cells

        for (auto neighbour = cellGroup.begin() + 1; neighbour != cellGroup.end(); std::advance(neighbour, 1)) {
            for (auto &p_i: cells[cellGroup[0]]) {
                for (auto &p_j: cells[*neighbour]) {
                    if (ArrayUtils::L2Norm(p_i.getX() - p_j.getX()) <= rCutOff) {
                        function(p_i, p_j);
                    }
                }
            }
        }
    }
}
```

Linked-cell algorithm – Benchmarks

Operating system
Ubuntu 22.04.4 LTS

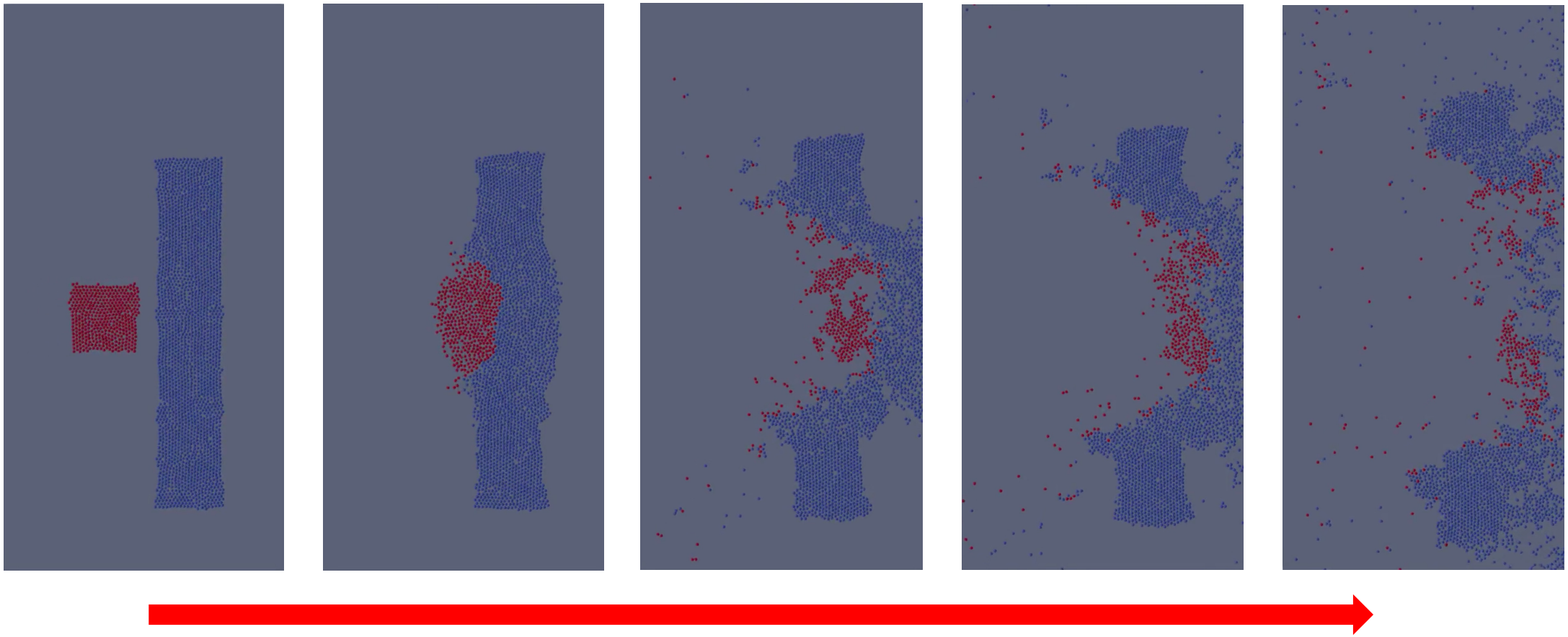
Processor
Intel Core i5-6500 CPU @3.20Hz

Memory
16 GiB



Boundary conditions – Reflective

Note: Boundary conditions can also be set for each side independently (top, bottom, etc.)



Simulation of a falling drop – Wall