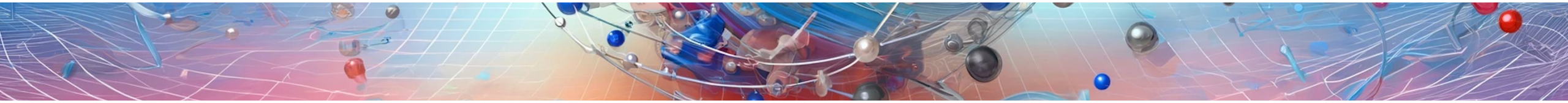
PSE Molekulardynamik
Sheet 2: Collision of two bodies



Group A: Daniel Schade, Ashutosh Solanki, Robin Cleve

17.05.2024

Overview

1. **Unit Tests**
   - 1.1 Google Test setup
   - 1.2 What we implemented
2. **Continuous Integration**
3. **Logging**
4. **Collision of Bodies**
   - 4.1 What is new?
   - 4.2 Particle Generator
   - 4.3 Brownian Motion
   - 4.4 Leonard Jones Force
   - 4.5 Animation
5. **Build Project on Mac**

Unit Tests – Google Test setup

- Separate **`google-test.cmake`** file
- No system wide installations:

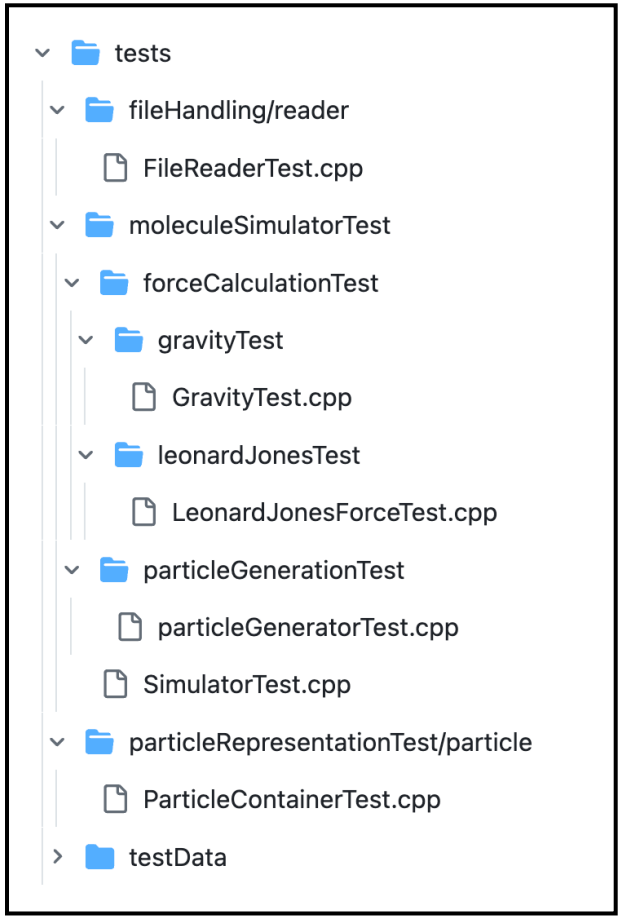
```
include(FetchContent)
FetchContent_Declare(
        googletest
        URL https://github.com/google/googletest/archive/03597a01ee50ed33e9dfd640b249b4be3799d395.zip
)
```

- Automatically discover and configure tests

```
gtest_discover_tests(MolSimTests)
```

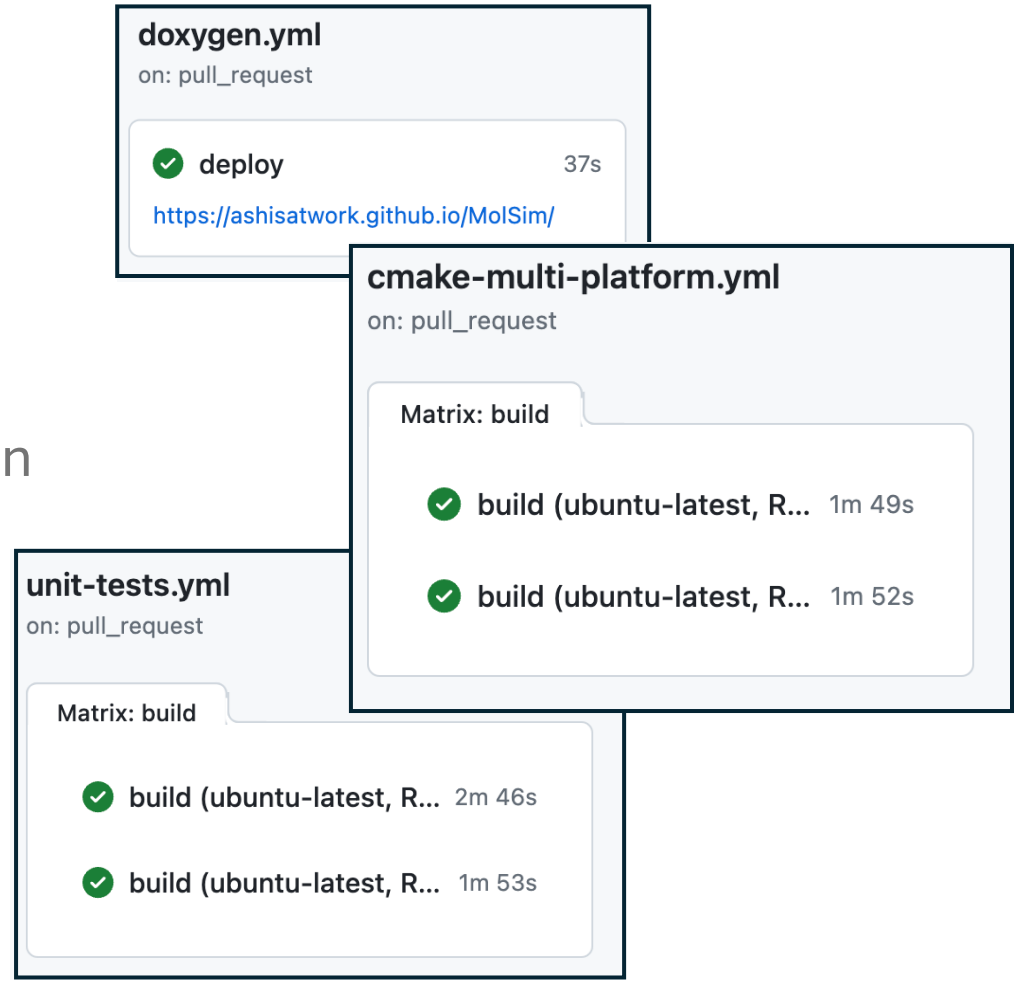
Unit Tests – What we implemented

- Test classes:
  - FileReaderTest.cpp
  - GravityTest.cpp
  - LeonardJonesForceTest.cpp
  - ParticleGeneratorTest.cpp
  - SimulatorTest.cpp
  - ParticleContainerTest.cpp
- Every method of the classes above is tested
- Structure of **tests** is identical to that of **src**
- Fixures for test environment in every test class



Continuous Integration

- We use Git Hub action workflows
- Our workflows
  1. Checks if codes builds and compiles
  2. Automatically runs the unit tests
  3. Build an deploy Doxygen documentation (hosted on GitHub pages)
- Enabled branch protection

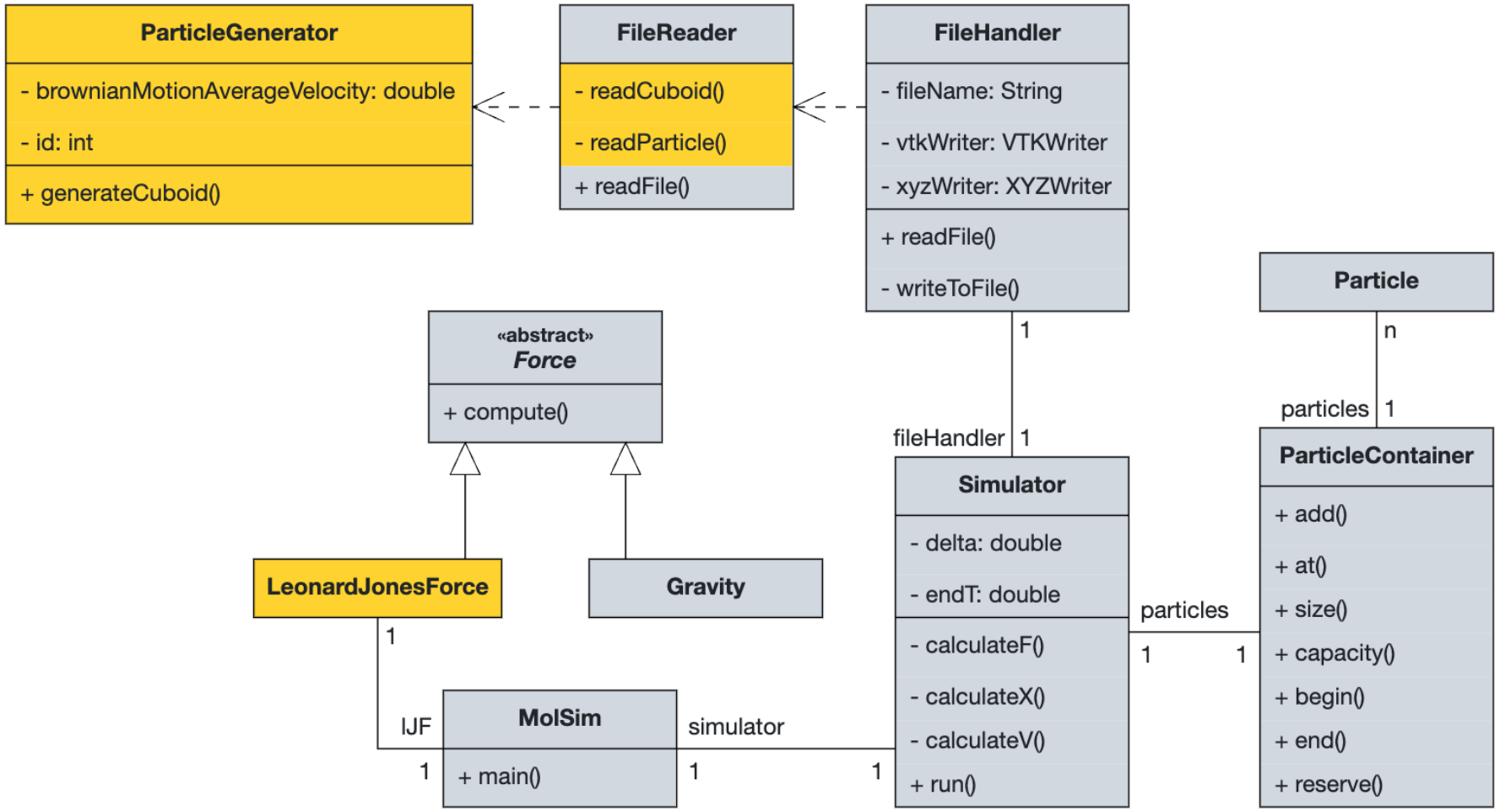


Logging

- We went with log functions instead of log macros
  - Change log level without recompiling
  - Not significantly longer execution times due to optimization
  - Multiple output formats
- Default log level is **info**
- Log level **error**

```
if (datastream.eof()) {
    spdlog::error("Error reading file: eof reached unexpectedly reading from line {}", i);
    return -1;
}
```

Collision of two bodies – What is new?

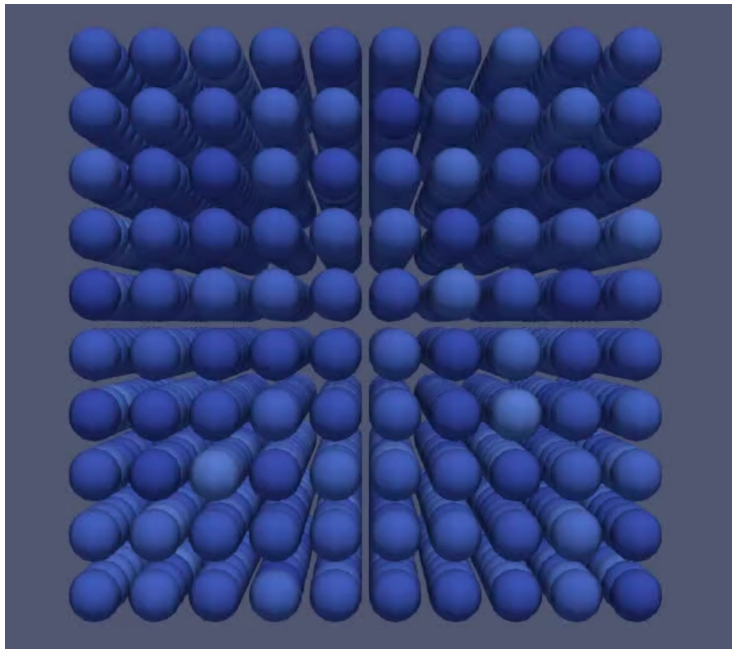


Collision of two bodies – Particle Generator

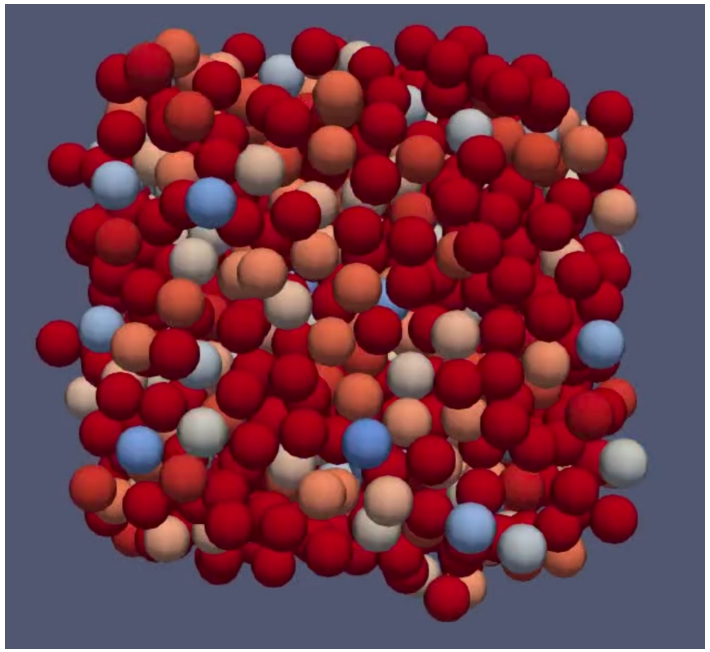
Structure of the new input file:

```
# position        N1     N2     N3     h          m     velocity          brownianMotion
Cuboid
2
0.0 0.0 0.0       40     8      1      1.1225     1.0   0.0   0.0 0.0     0.1
15.0 15.0 0.0      8     8      1      1.1225     1.0   0.0 -10.0 0.0     0.1
```

Collision of two bodies – Brownian Motion



thermal energy



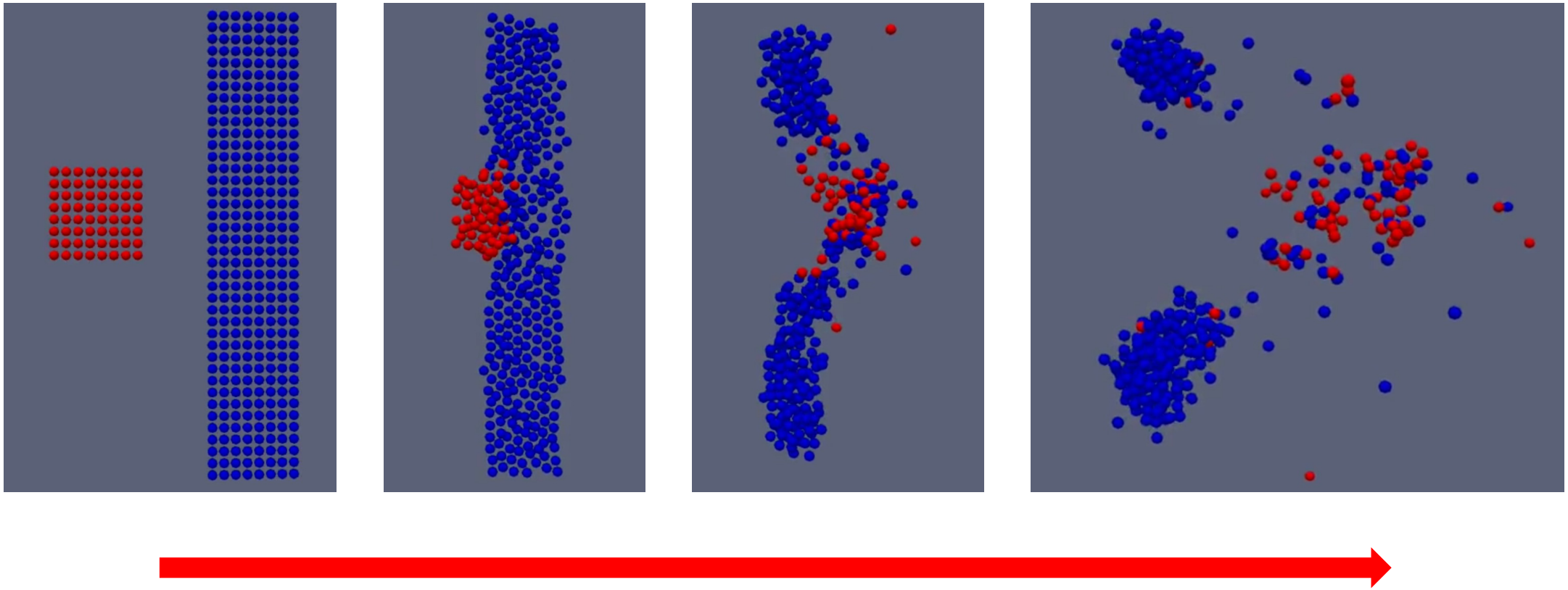
Collision of two bodies – Leonard Jones

$$F_{ij} = -\frac{24\varepsilon}{(\|\mathbf{x}_i - \mathbf{x}_j\|_2^2)^2}\left(\left(\frac{\sigma}{\|\mathbf{x}_i - \mathbf{x}_j\|_2}\right)^6 - 2\left(\frac{\sigma}{\|\mathbf{x}_i - \mathbf{x}_j\|_2}\right)^{12}\right)(\mathbf{x}_i - \mathbf{x}_j)$$

$$F_{ij} = \frac{24\varepsilon}{(\|\mathbf{x}_i - \mathbf{x}_j\|_2^2)^2}\left(\left(\frac{\sigma^2}{(\|\mathbf{x}_i - \mathbf{x}_j\|_2^2)^2}\right)^3 - 2\left(\left(\frac{\sigma^2}{(\|\mathbf{x}_i - \mathbf{x}_j\|_2^2)^2}\right)^3\right)^2\right)(\mathbf{x}_j - \mathbf{x}_i)$$

```
std::array<double, 3> LeonardJonesForce::compute(Particle &target, Particle &source) {
    auto difference = source.getX() - target.getX();
    double squared_distance = std::pow(ArrayUtils::L2Norm(difference), 2);
    double c1 = std::pow(sigma * sigma / squared_distance, 3);
    double c2 = 2 * c1 * c1;
    return ((24 * epsilon) / squared_distance) * (c1 - c2) * difference;
}
```

Collision of two bodies – Animation



Build Project on Mac – with Docker

- **Issue:** Mac users in our team encountered difficulties building the project on their local machines
- **Solution:**
  - Use of **Docker**, a free software for running code in lightweight containers
  - Container is an isolated environment for building and running the code
  - Not native configuration of environment on native operating system
  - **Docker file** and **step-by-step guide**

Thank you for listening!