# Exercise 7

## 7.1 Reading

## 7.2 n-Body Problem — Partitioning/Communication Design

### 7.2.1 Memory Layout

```
std::random_device rd;  //Will be used to obtain a seed for the random number engine
std::mt19937 gen(rd()); //Standard mersenne_twister_engine seeded with rd()
std::uniform_real_distribution<> mass_distrib(1e-10, 1e10);
std::uniform_real_distribution<> space_distrib(-1e4, 1e4); // start within 25km of eachother

struct Body {
  union {
    double raw[4];
    struct __attribute__((__packed__)) {
      double m;
      double pos[3];
    };
  };

  Body() {
    m = mass_distrib(gen);
    for (int i=0; i < 3; i++) {
      pos[i] = space_distrib(gen);
    }
  }
}

// to send we just use the raw data
MPI_Send(&b, 8, ...)
```

- this structure will result in contiguous arrays of bytes which can be sent by their `double raw[4]` representation
- velocities will not be sent
- `MPI_Gather()` will then collect blocks of mulitple bodies per rank

### 7.2.2 Partitioning

- the amount of bodies should be a multiple of the available ranks, resulting in equally large messages
- each rank will handle a number of bodies
- the initial positions and masses are scattered over the ranks
- ranks are sequential over nodes (per-slot-mapping) resulting in the least amount of node-to node communication due to circular messaging ($\frac{4}{16}$ of all communication for 16 ranks over 4 nodes)

### 7.2.3 Communication

- the communication will happen in a circular fashion
- for each arriving packet a copy will be saved inside of a buffer
- the now buffered msg can be forwarded to the next rank
- while waiting for the arrival of the next msg, the received data can be used to update the Forces on each body the specific rank is responsible for

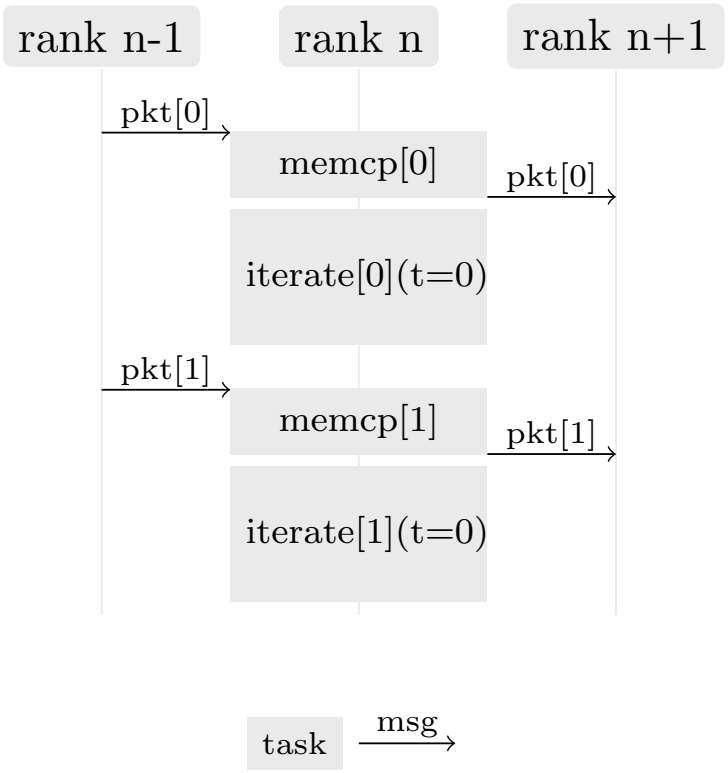


Figure 1: Overalp utilization during ring-wise message passing

Figure 1 shows this sequence for a rank n:

- here rank n starts out with a running `MPI_Irecv()` and waits for `pkt[0]`
- after copying the received data to a buffer, rank n calls `MPI_Isend()` on `pkt[0]` and starts another `MPI_Irecv()`
- after iterating over the received data, rank n will wait again for `pkt[1]`