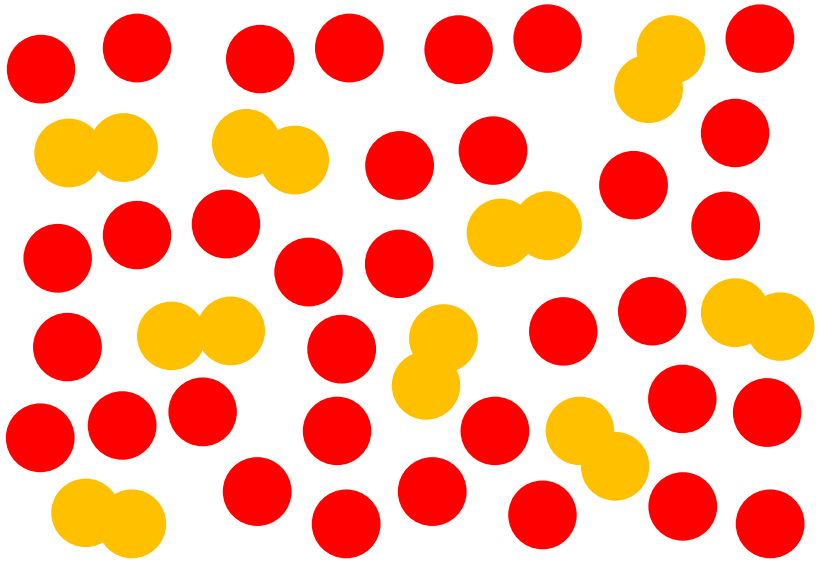
Blazing Fast Neighbor Search with Spatial Hashing

Matthias Müller, Ten Minute Physics
www.matthiasmueller.info/tenMinutePhysics

Problem

- Given: n points



- For all points $\mathbf{p}_i$: Find neighbors $\mathbf{p}_j$ such that $\left|\mathbf{p}_j - \mathbf{p}_i\right| \leq d$
- Special case $d = 2r$: Find overlaps of particles with radius r
- Use case: Fluid, sand, snow simulations

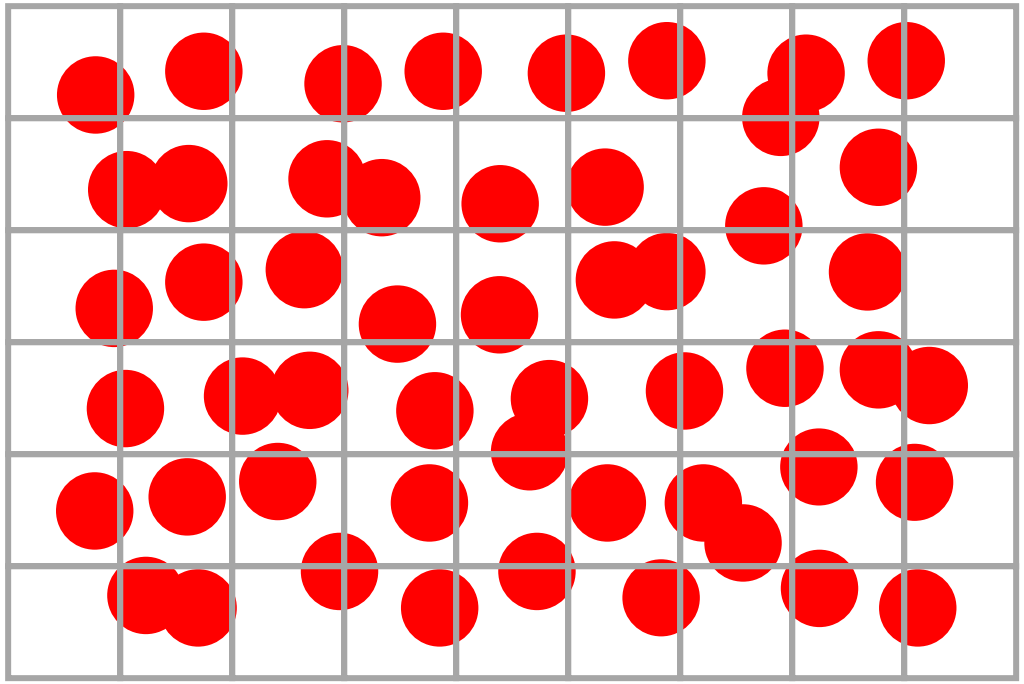
Naïve Solution

for all points $\mathbf{p}_i$
  for all points $\mathbf{p}_j$
    if $\left|\mathbf{p}_j - \mathbf{p}_i\right| \leq d$
      handleOverlap(i, j)

- Problem: the complexity of the algorithm is $O(n^2)$
- For 100,000 points (common) we perform 10,000,000,000 tests!
- In general for the complexity of a simulation algorithm:
  - $O(n)$ is good
  - $O(n \log n)$ is OK (i.e. sorting, $\log 100{,}000 \approx 17$)
  - $O(n^2)$ is not an option!

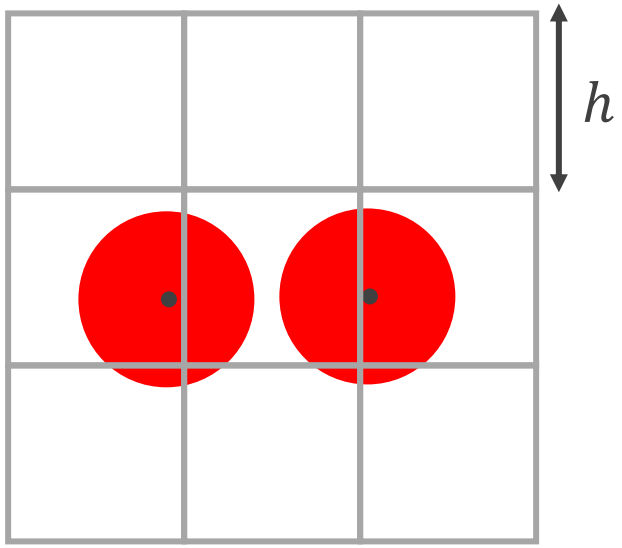
Acceleration Idea

- Store particles in a regular grid



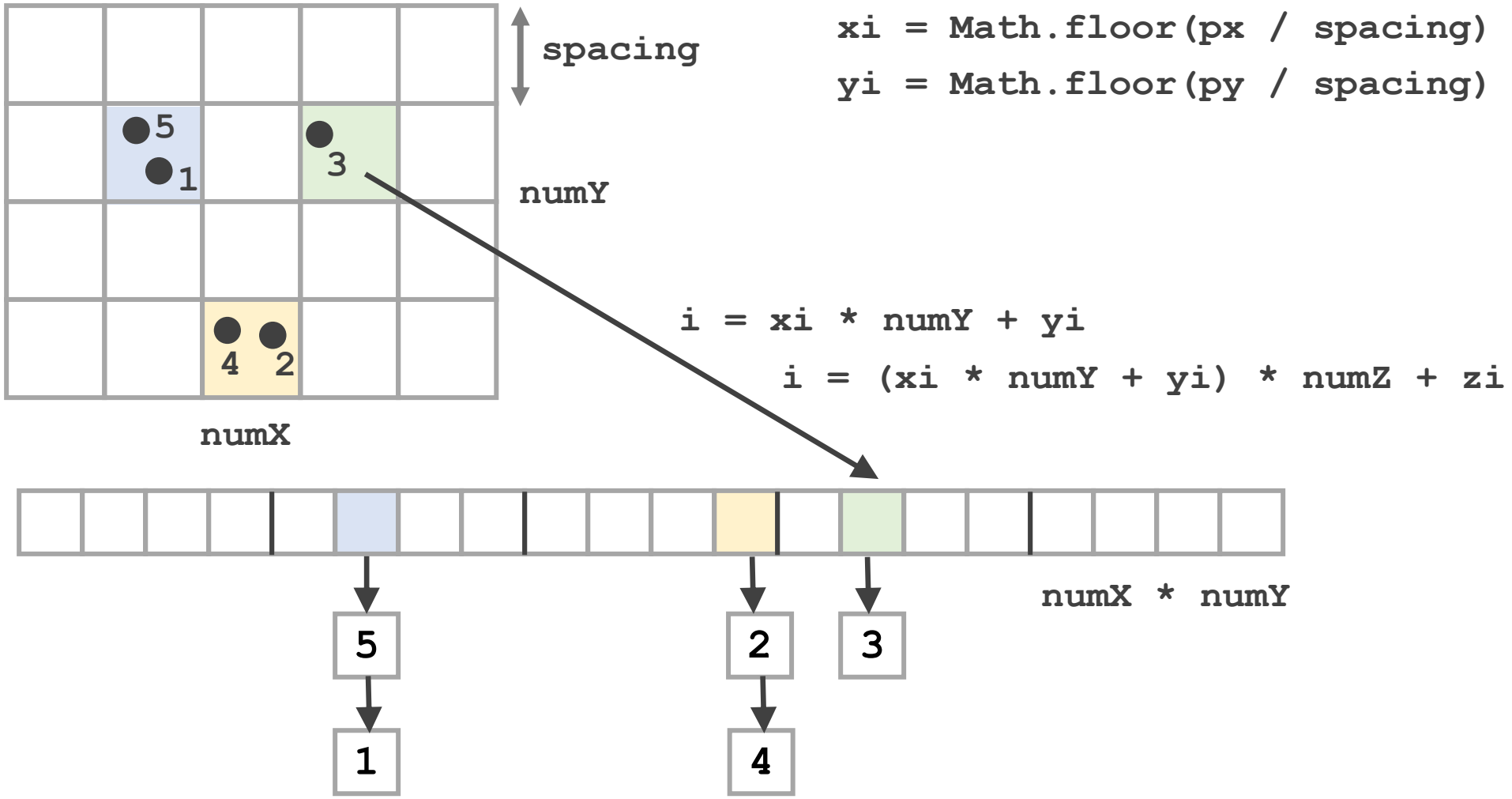
- Only check close cells

Grid Layout

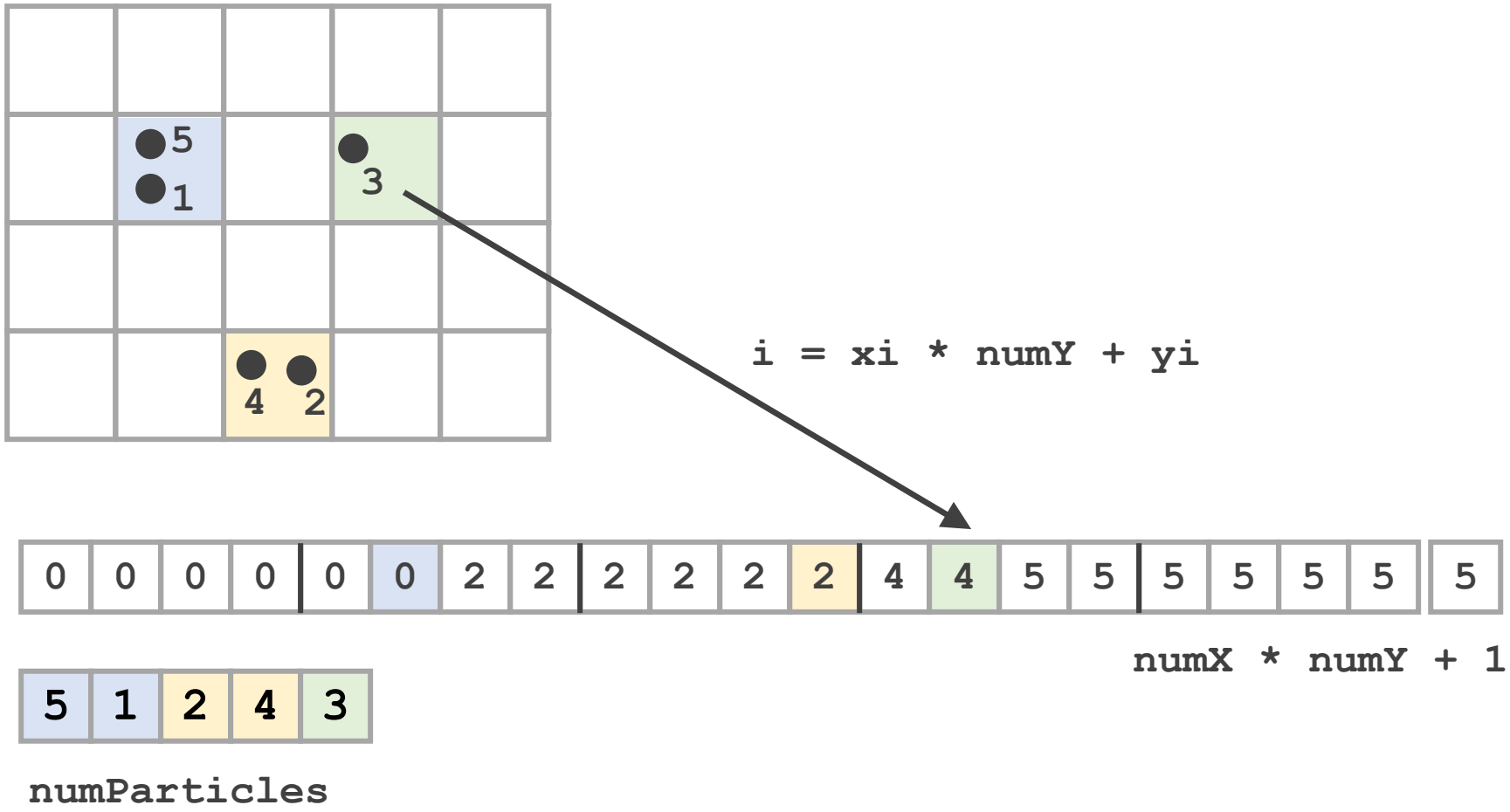


- Store particles once where their center is located.
- If we choose $h = 2r$
  we only need to check the containing and the surrounding cells
- 9 in 2 dimensions, 27 in 3 dimensions

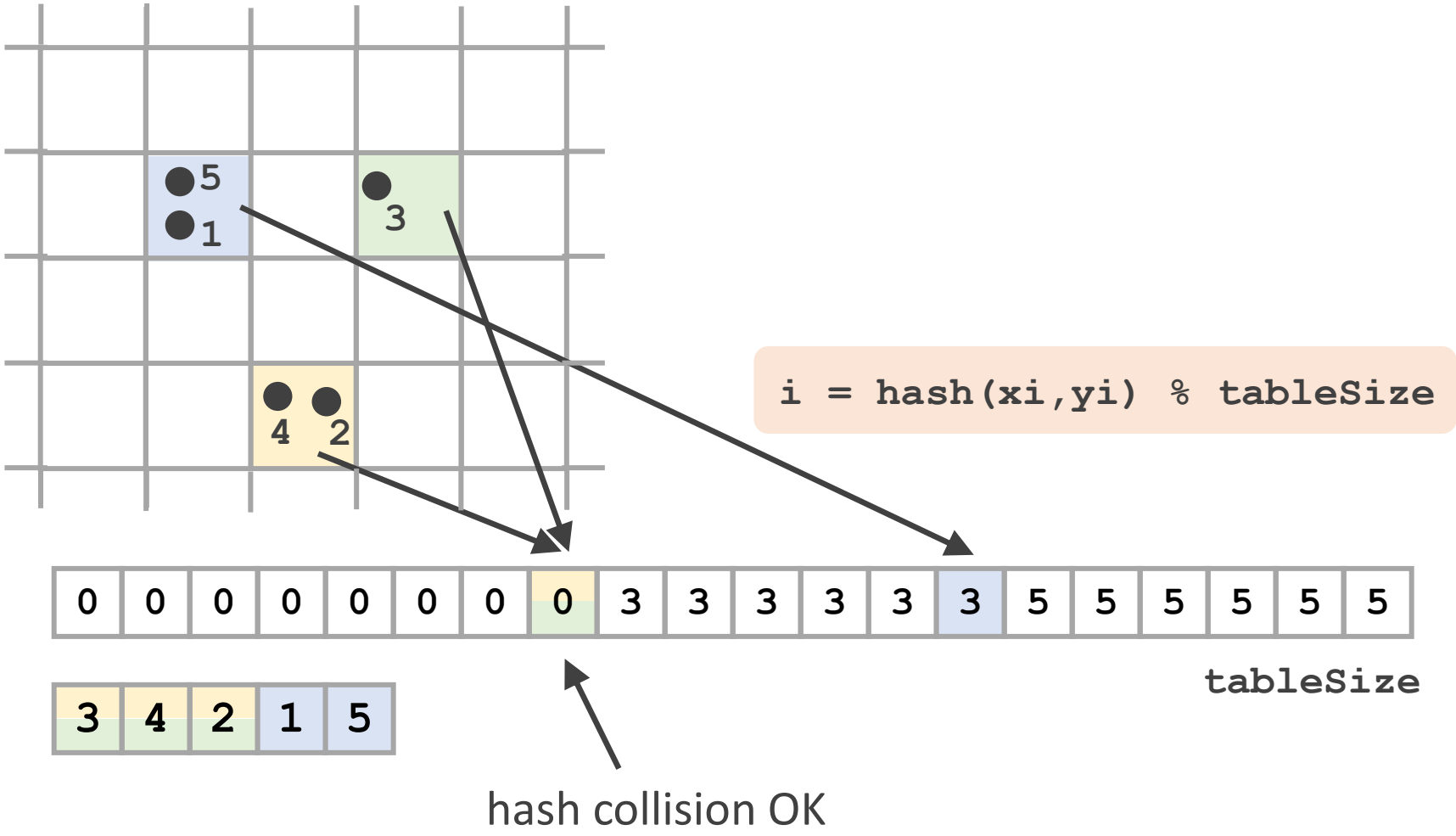
Storing the Grid



Dense Grid Representation



Handle Unbounded Grid with Hashing



Hashing Remarks

- The hash function must return a value that distributes the cells evenly

```
hashCoords(xi, yi, zi) {
    var h = (xi * 92837111) ^ (yi * 689287499) ^ (zi * 283923481);
    return Math.abs(h) % this.numCells;
}
```

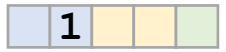
- Large hash table size: fewer collisions, fewer tests, faster
- Choosing **`tableSize = numParticles`** works well

Creating the Data Structure

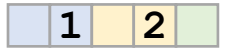
- Count

- Partial sums

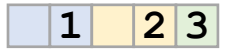
- Fill in
  - 1



  - 2



  - 3



  - 4

  - 5

Let’s implement it…