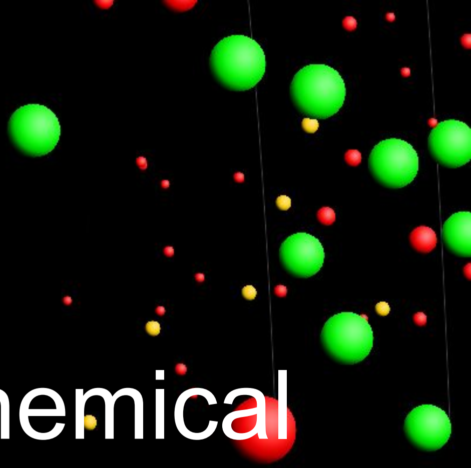
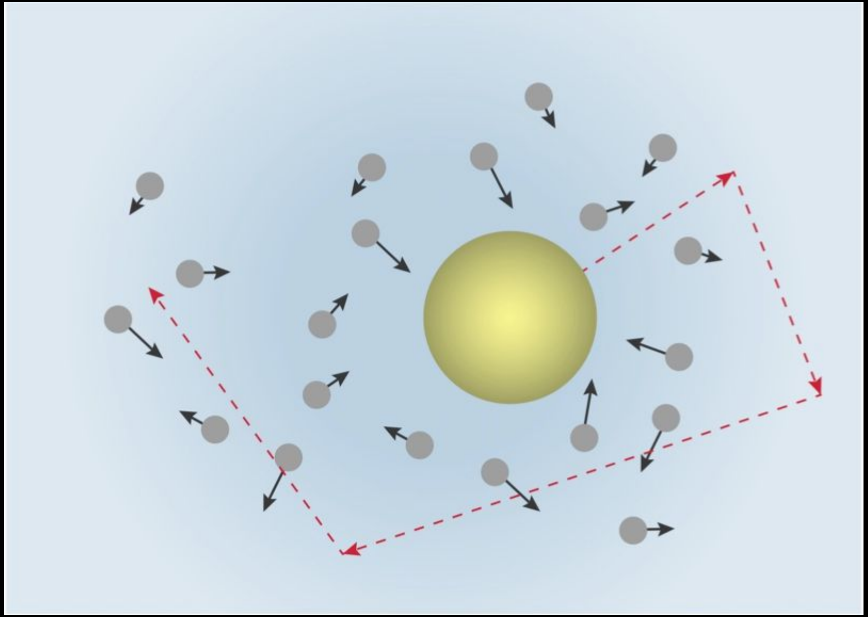
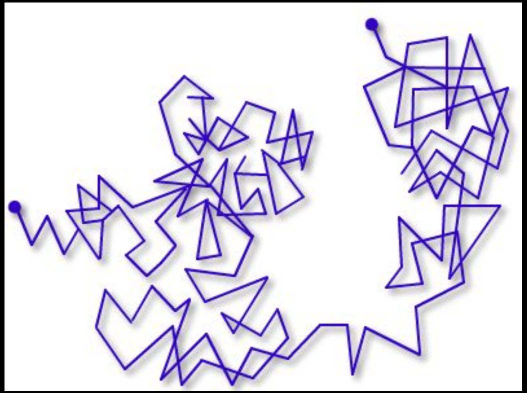
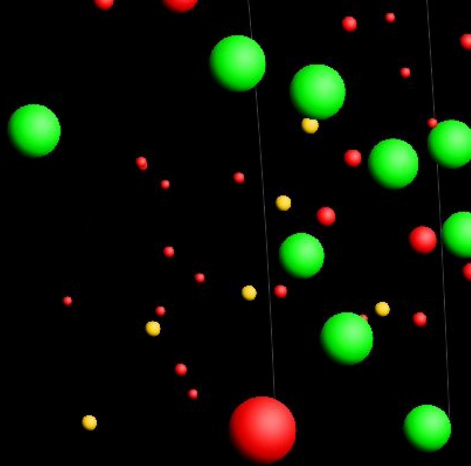


A stochastic particle-based chemical system simulator for the web

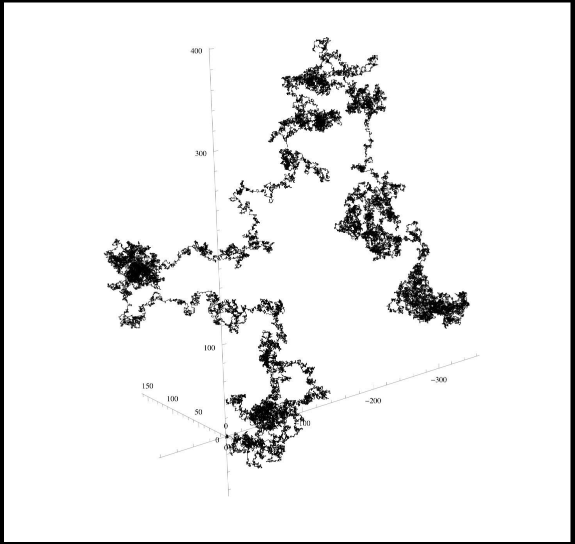
By Herman Bergwerf

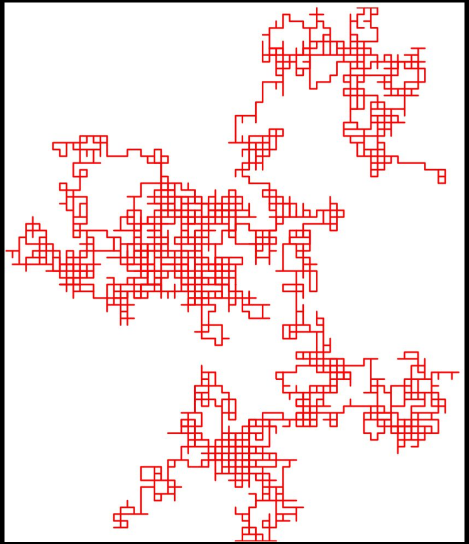
Brownian motion

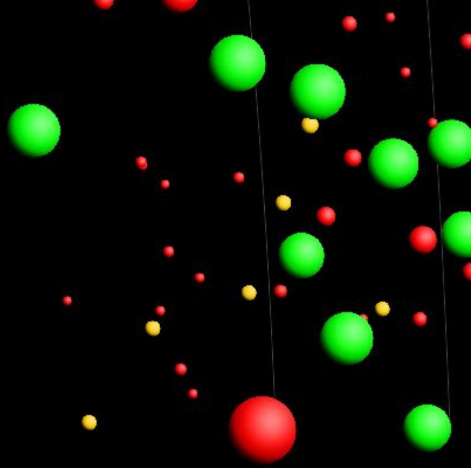
Random walk

Every step, choose a random new direction and speed*

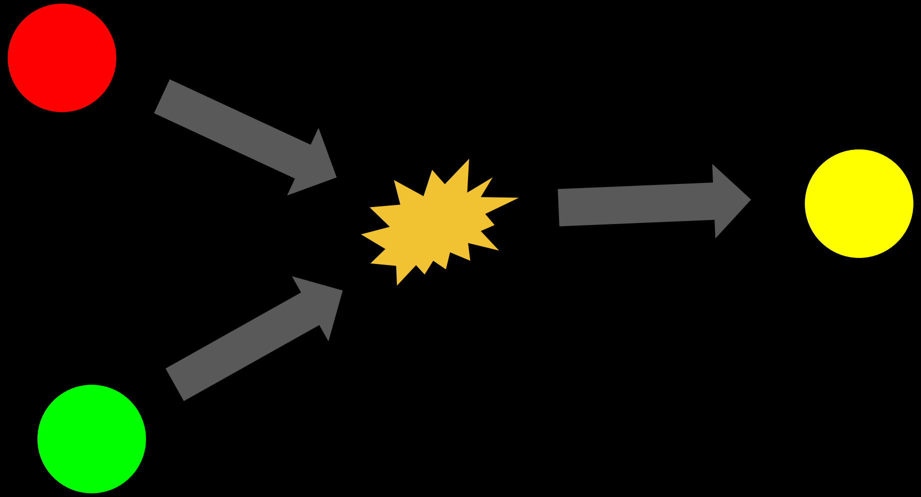


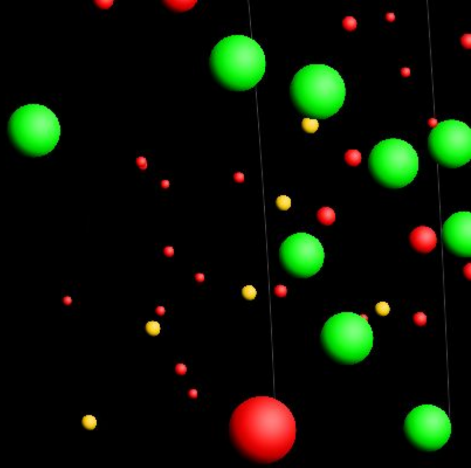


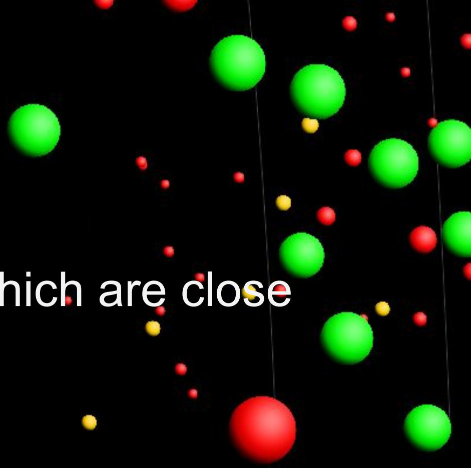
**Distributed around an average value (similar to Gaussian distribution)*



Reactions



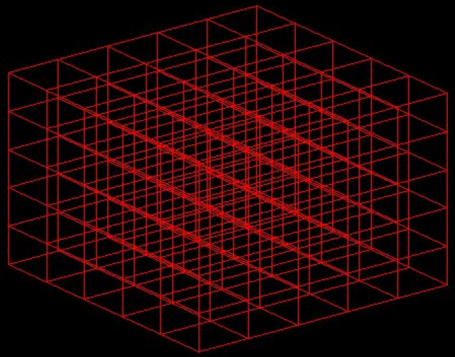




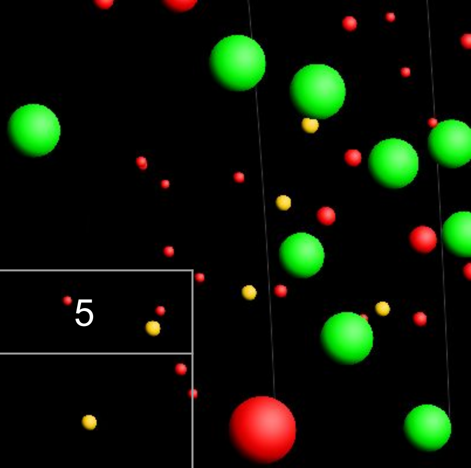
Collision algorithm

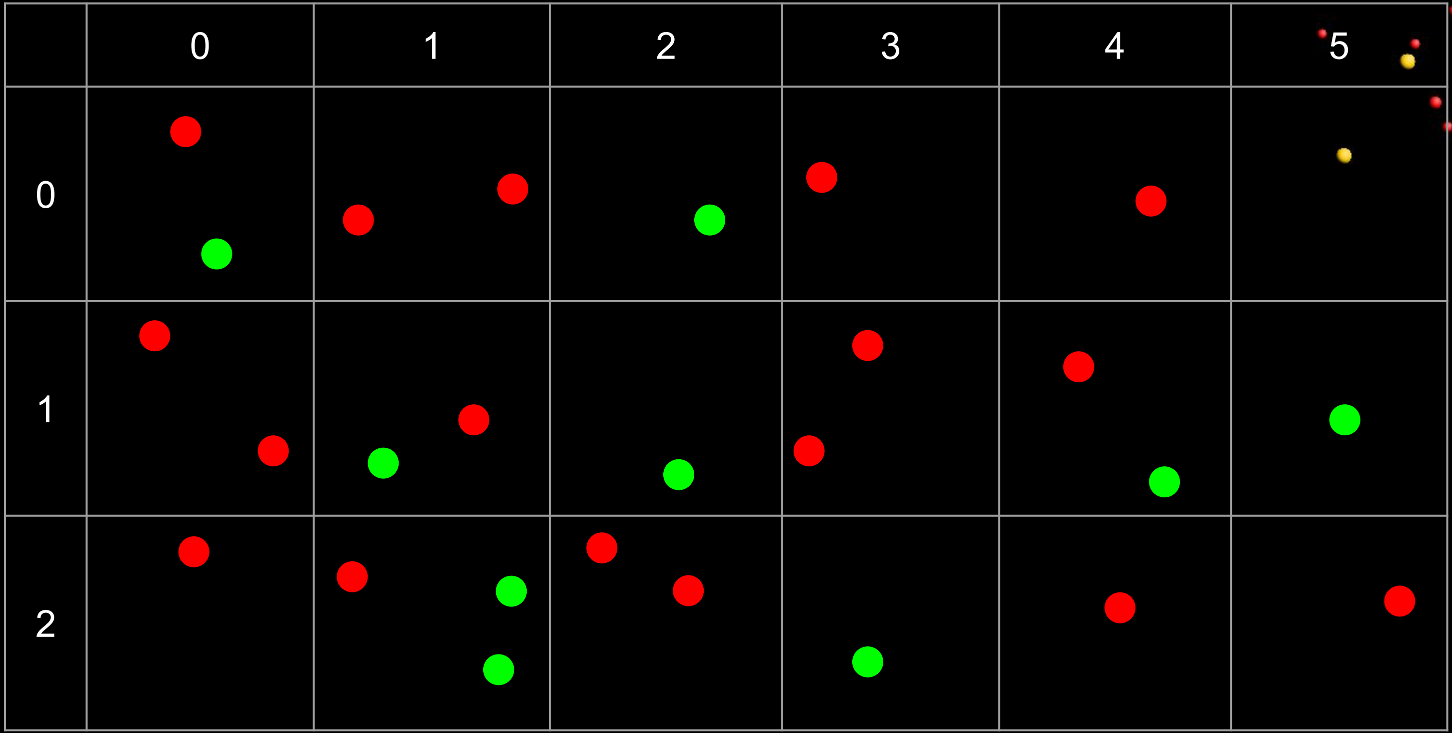
Problem: computing the distances between all particles to see which are close enough to collide is very slow (N^2), so:

1. Compute a voxel for each particle
2. Only reaction with other particles in the same voxel
3. Less accurate, but way faster

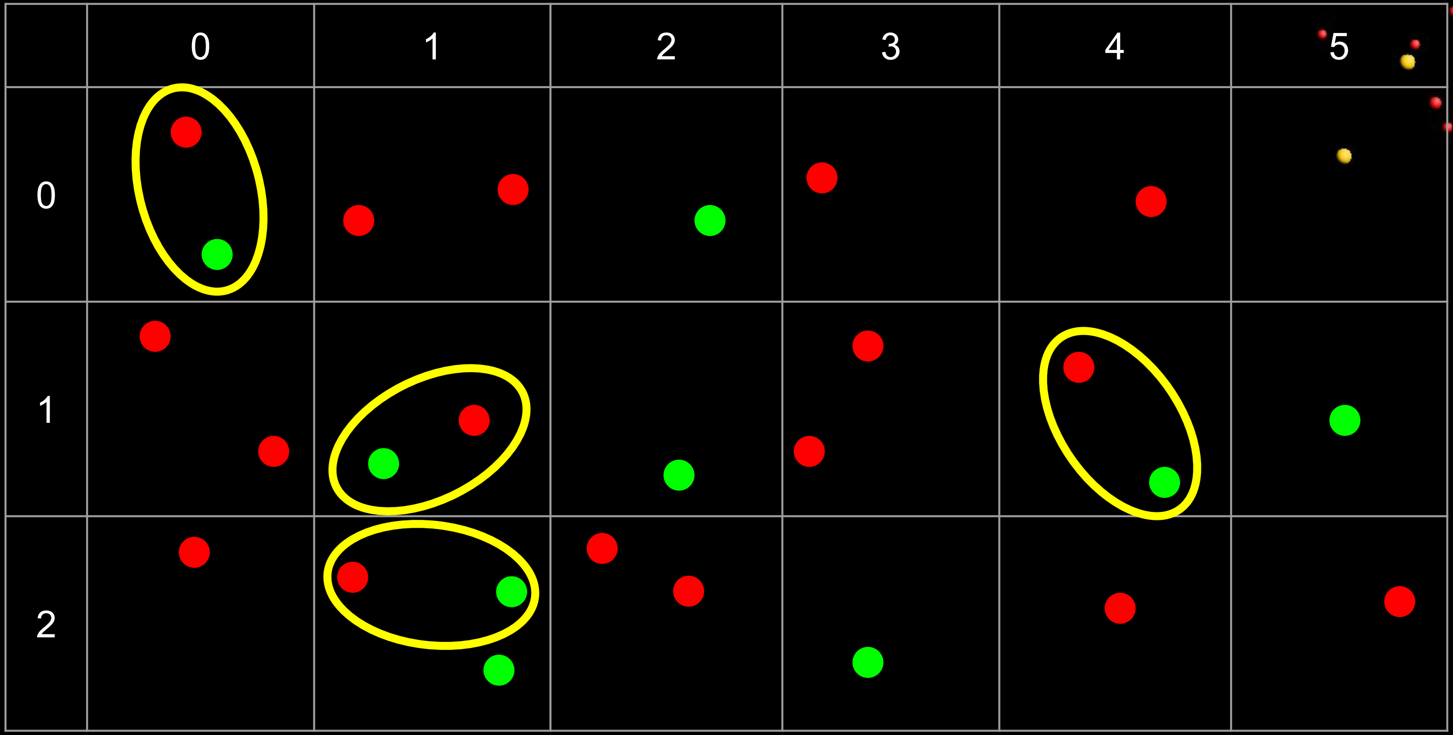


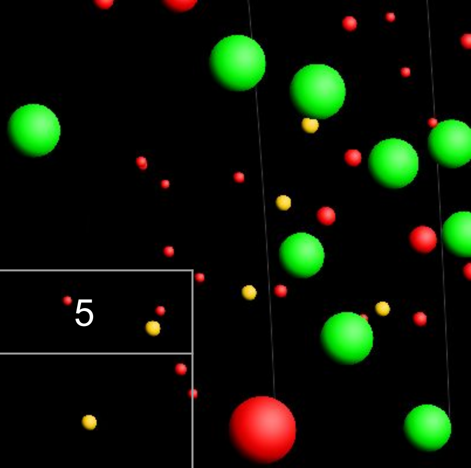
Particles in voxels: 2D example



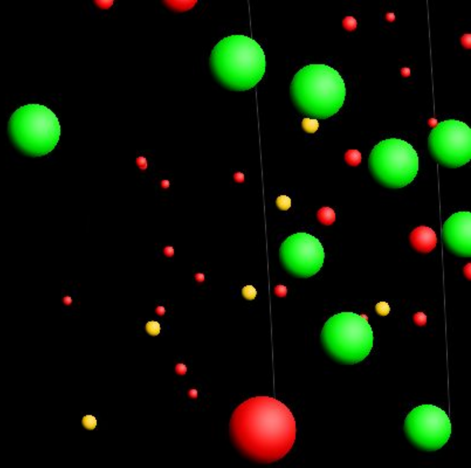


Particles in voxels: 2D example





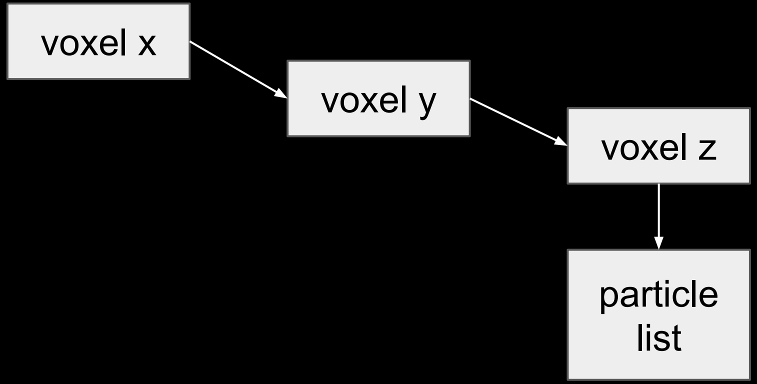
Data structure



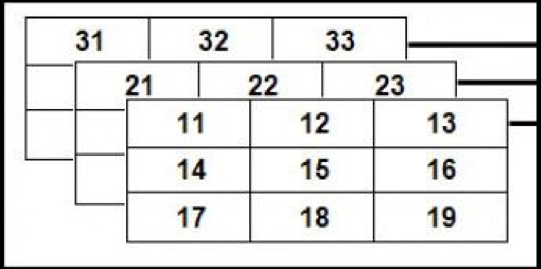
1. **First pass:** assign voxel position to each particle
2. **Second pass:** find particles that are in the same voxel

Naïve data structure for this:

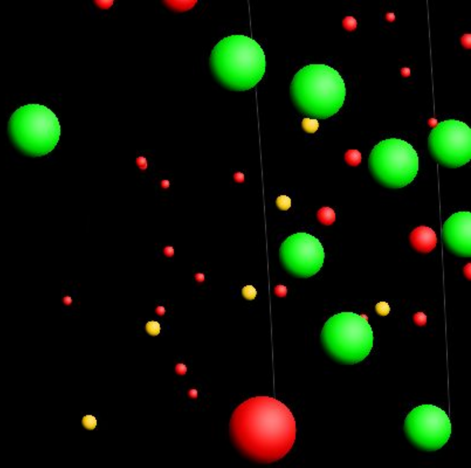
```
HashMap<int x,
  HashMap<int y,
    HashMap<int z,
      List<Particle>>>>
```



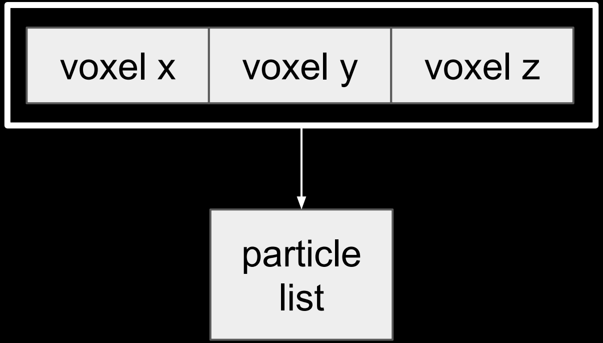
Alternative



- 10 bits per position
- packed together: 30 bits to represent a voxel position.

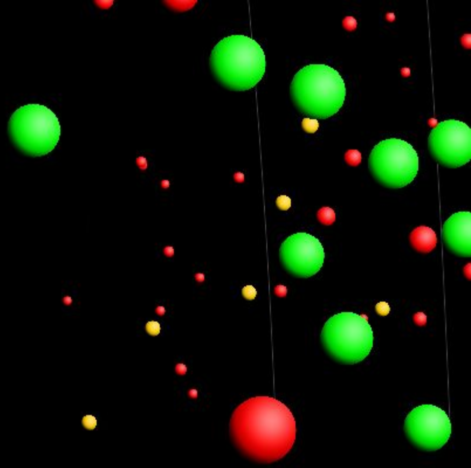


```
HashMap<int position, List<Particle>>
```



Alternative

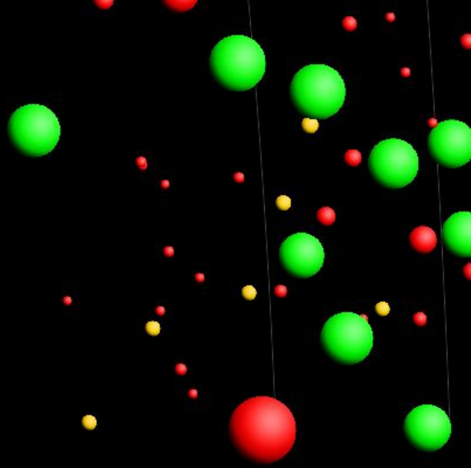
- 10 bits per position
- packed together: 30 bits to represent a voxel position.



```
List<Tuple2<int position, Particle>>
```

position	5	20	8	30	5	25	16	8	9	4
particle ID	0	1	2	3	4	5	6	7	8	9
particle type	🟩	🟩	🟥	🟥	🟥	🟥	🟩	🟩	🟥	🟥

Alternative

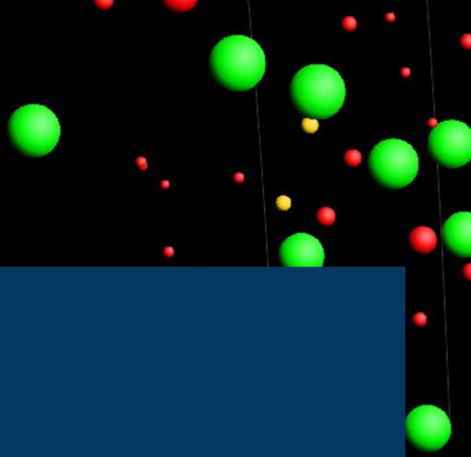


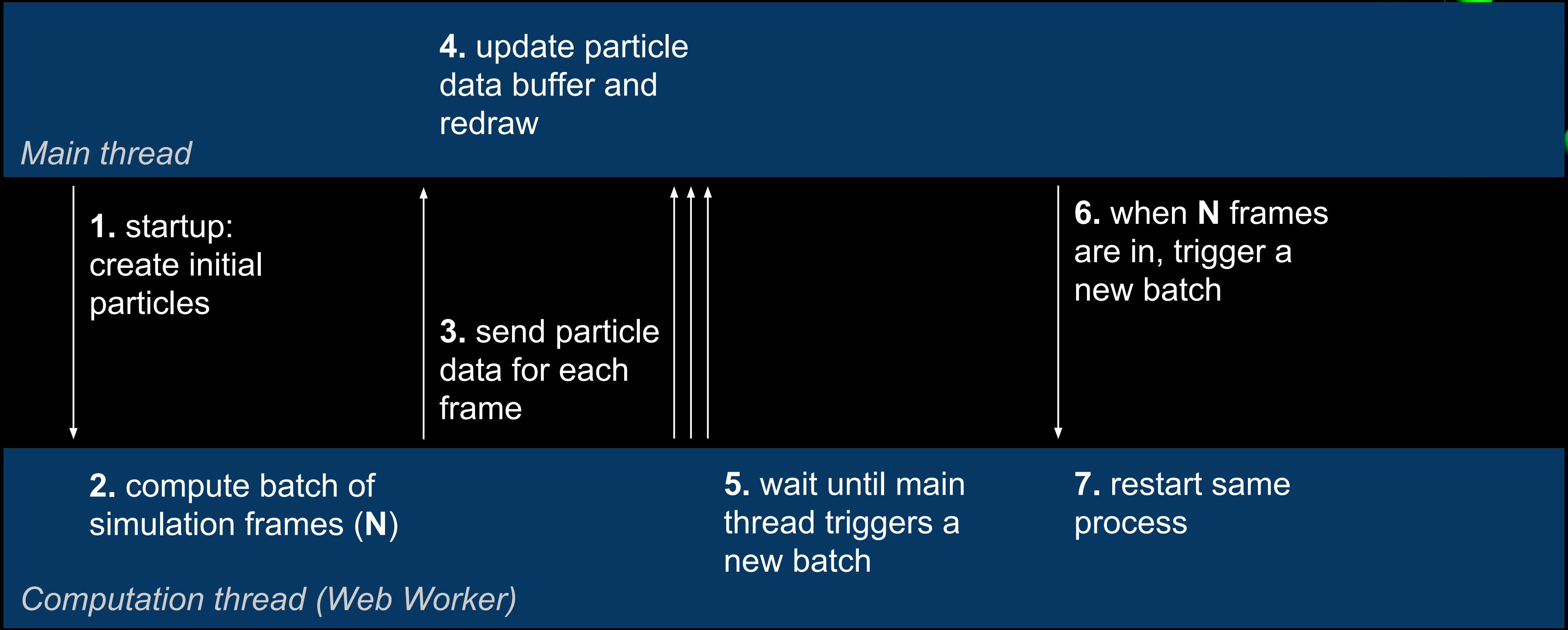
- 10 bits per position
- packed together: 30 bits to represent a voxel position.

```
List<Tuple2<int position, Particle>>
```

position	4	5	5	8	8	9	16	20	25	30
particle ID	9	0	4	2	7	8	6	1	5	3
particle type	🟥	🟩	🟥	🟥	🟩	🟥	🟩	🟩	🟥	🟥

Web Workers (multi-threading, kinda)





Main thread

4. update particle data buffer and redraw

1. startup: create initial particles

3. send particle data for each frame

6. when **N** frames are in, trigger a new batch

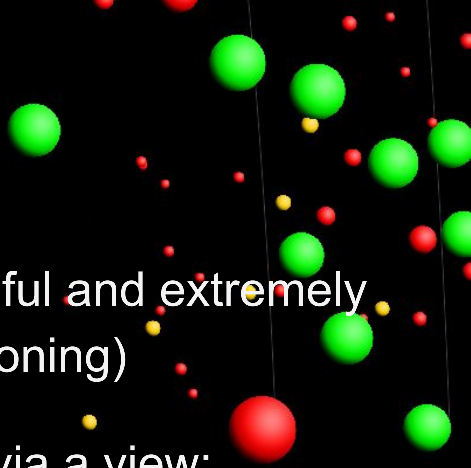
Computation thread (Web Worker)

2. compute batch of simulation frames (**N**)

5. wait until main thread triggers a new batch

7. restart same process

ByteBuffer



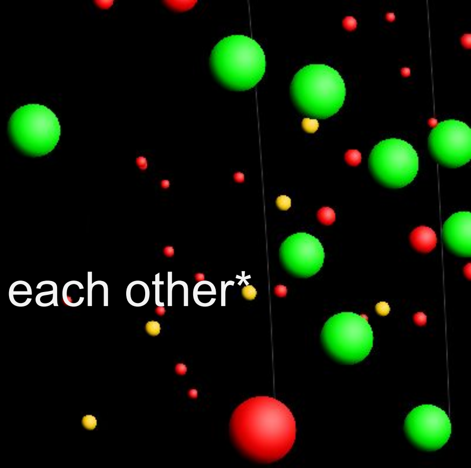
Passing an array with 100.000 particles to another thread is painful and extremely slow in JavaScript, but fortunately we have ArrayBuffer (faster cloning)

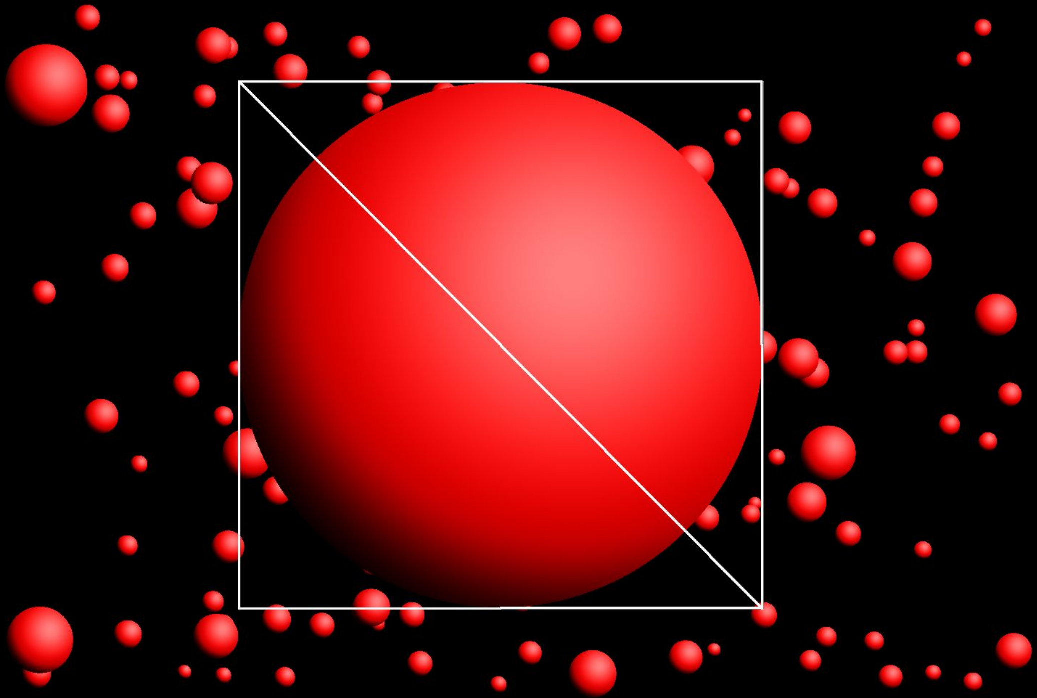
All data is constantly packed into an ArrayBuffer, and accessed via a view:

adius	**Position**			**Color**			**Radius**	
r	x	y	z	R	G	B	r	x

Fast spheres using shaders

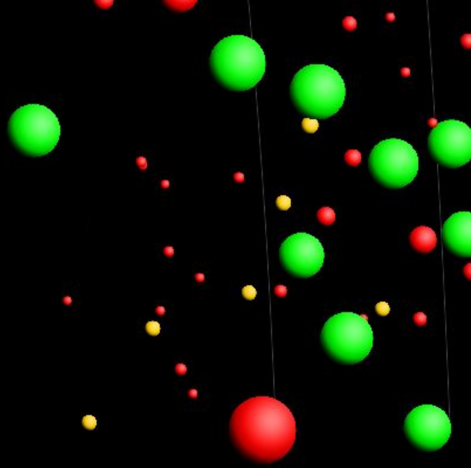
How to quickly render spheres? Throw some gradients on top of each other*





**and a neat trick with* `glDrawElementsInstanced`

More information



Source code:

github.com/molview/bromium (written in the Dart language)

Live demo:

molview.github.io/bromium-deploy/