An efficient SIMD implementation of pseudo Verlet-lists for neighbour interactions in particle-based codes

James Willis, Matthieu Schaller, Pedro Gonnet & Richard Bower
Durham University, ICC

Team

This work is a collaboration between two departments at Durham University (UK):

- The Institute for Computational Cosmology,
- The School of Engineering and Computing Sciences,

with contributions from the astronomy group at the University of St. Andrews, University of Dublin, ETH Lausanne and the DiRAC software team.

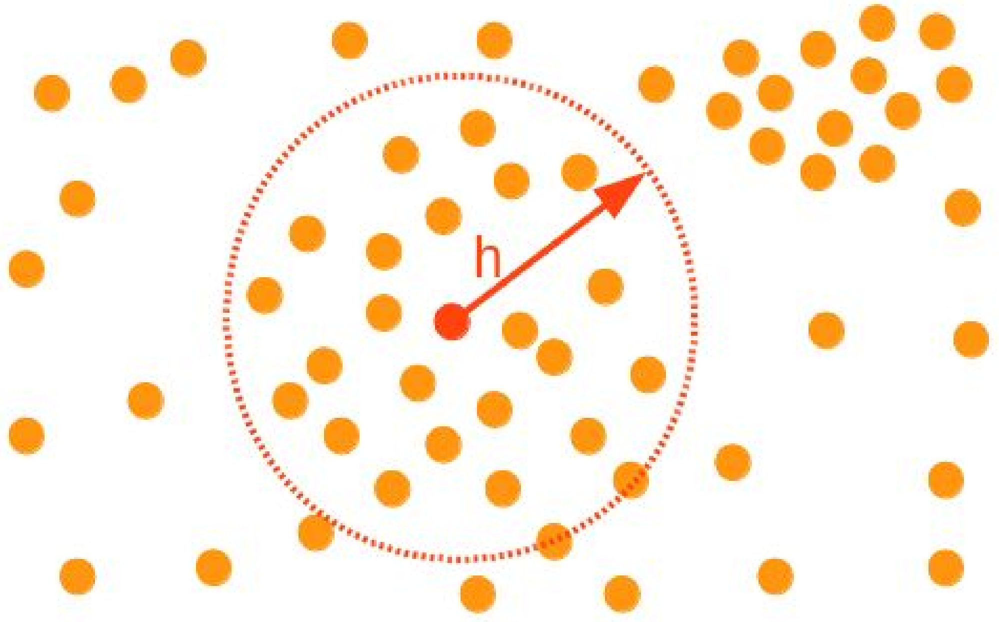
- This research is fully funded by an IPCC.

Overview

- Problem to solve
- Solution
  - Pseudo Verlet list
  - Particle sorting
- SIMD vectorisation strategy for particle based codes (MD, SPH, etc.)
  - Particle caches AoS to SoA
- Strategy applied to SWIFT
- Performance results
- Conclusions

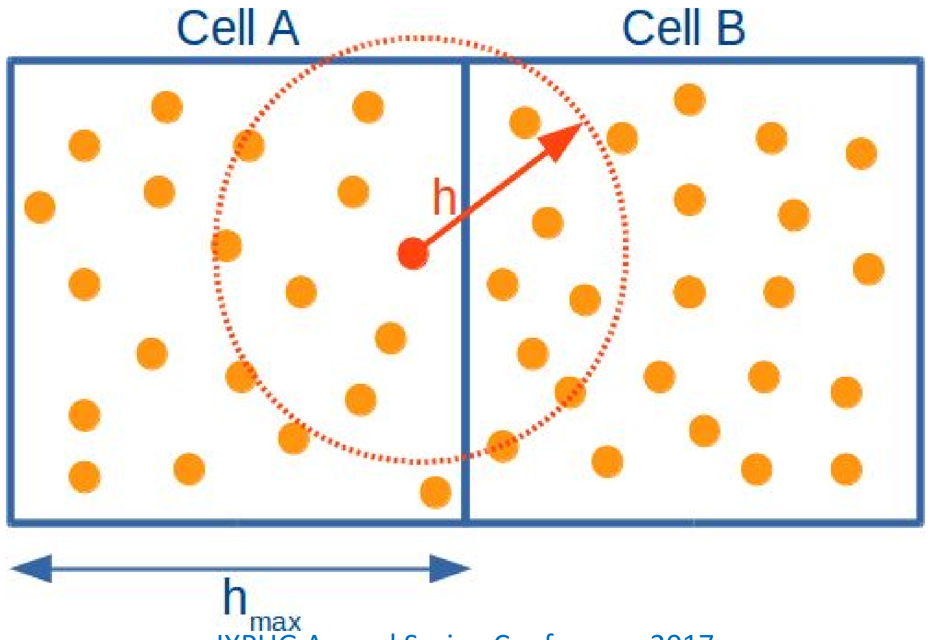
Problem

- We update each particle using SPH (Smoothed-Particle Hydrodynamics)
- Each particle interacts with its neighbours that are within a smoothing length, **h**
- The smoothing length varies depending on the particle density of the region
- Interaction cheap to compute



Problem

- The particles are divided up into cells of edge $\mathbf{h_{max}}$, where $\mathbf{h_{max}}$ is the maximum particle smoothing length in the simulation
- Computing the interactions of particles in two neighbouring cells would require a lot of unnecessary distance calculations
- The majority of particles will not be within range of each other



Naive Solution

<u>Brute Force</u>

- Perform a double for loop over all particles
- Interact particles that are within range of each other, $\mathbf{r^2 < h^2}$

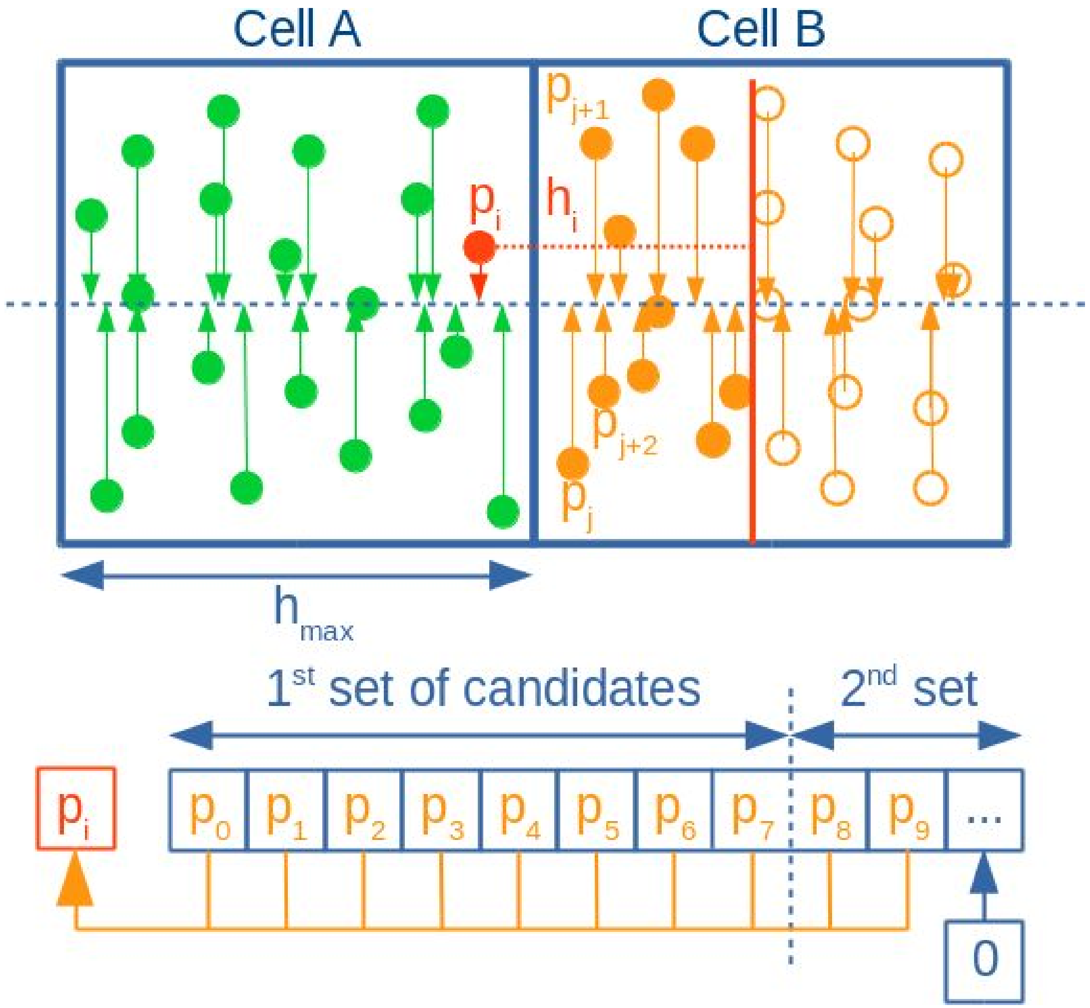
```
for(int i=0; i<count_i; i++) {
  struct part pi = parts_i[i];
  float hi = pi.h;

  for(int j=0; j<count_j; j++) {
    struct part pj = parts_j[j];

    float dist = calc_dist(pi,pj);

    if(dist*dist < h*h) interact(pi,pj);
  }
}
```

Smart Solution

Particle Sorting

- Place particles into a pseudo Verlet list:
  - Project particles onto the axis joining the center of the two cells
  - Sort the particles on the axis based upon their position on the axis
  - Sorting performed using a merge sort
  - Only occurs when the particles have moved by a certain distance
- Reduces the number of candidates
- These particles are still tested so that they are within the 3D distance

Smart Solution



Smart Solution

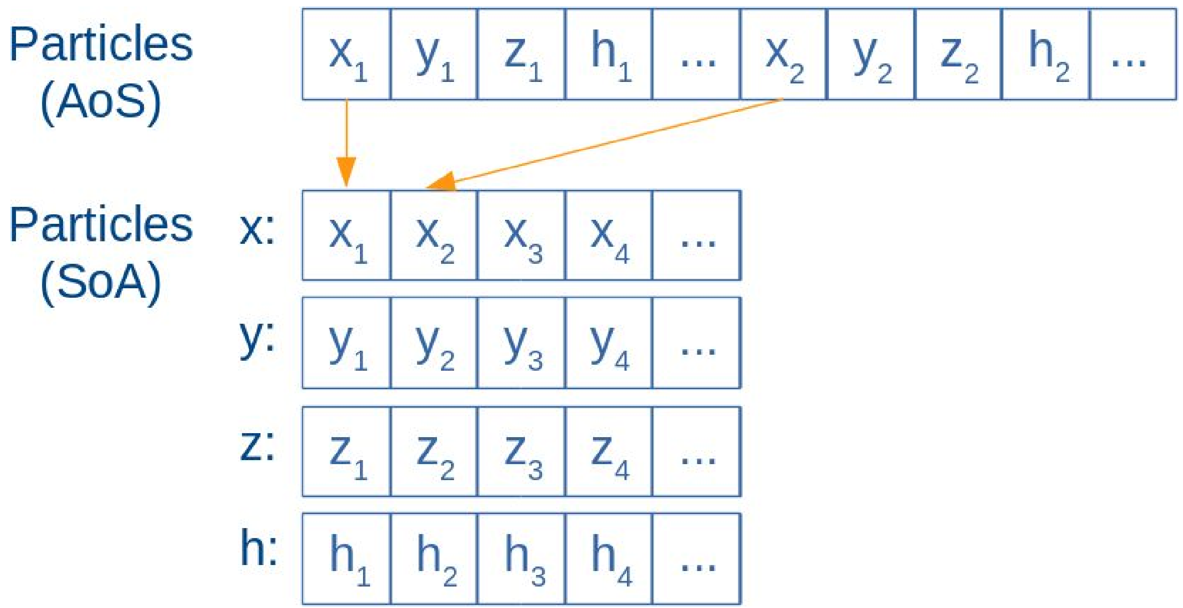
```
// Sort the particle on the axis
sort_parts(parts_i,parts_j);

// Pick particle pi from Cell A
for(int i=0; i<count_i; i++) {
  struct part pi = parts_i[i];
  float hi = pi.h;

  // Only loop over particles in Cell B that are within hi on the axis
  for(int j=0; j<count_j && dj<hi; j++) {
    struct part pj = parts_j[j];

    float dist = calc_dist(pi,pj);

    if(dist*dist < h*h) interact(pi,pj);
  }
}
```

SIMD Optimisations

- Use local particle cache (AoS -> SoA)
- Only read particles into cache that interact
- Calculate all interactions on a particle and store results in a set of intermediate vectors
- Perform horizontal add on intermediate vectors and update the particles with the result
- Pad caches to prevent remainders and mask out the result

Local Particle Cache

- The particles are stored in a global array of structs (AoS)
- Causes strided memory access when vectors are loaded
- We can improve performance by placing the required particle properties into a structure of arrays (SoA)

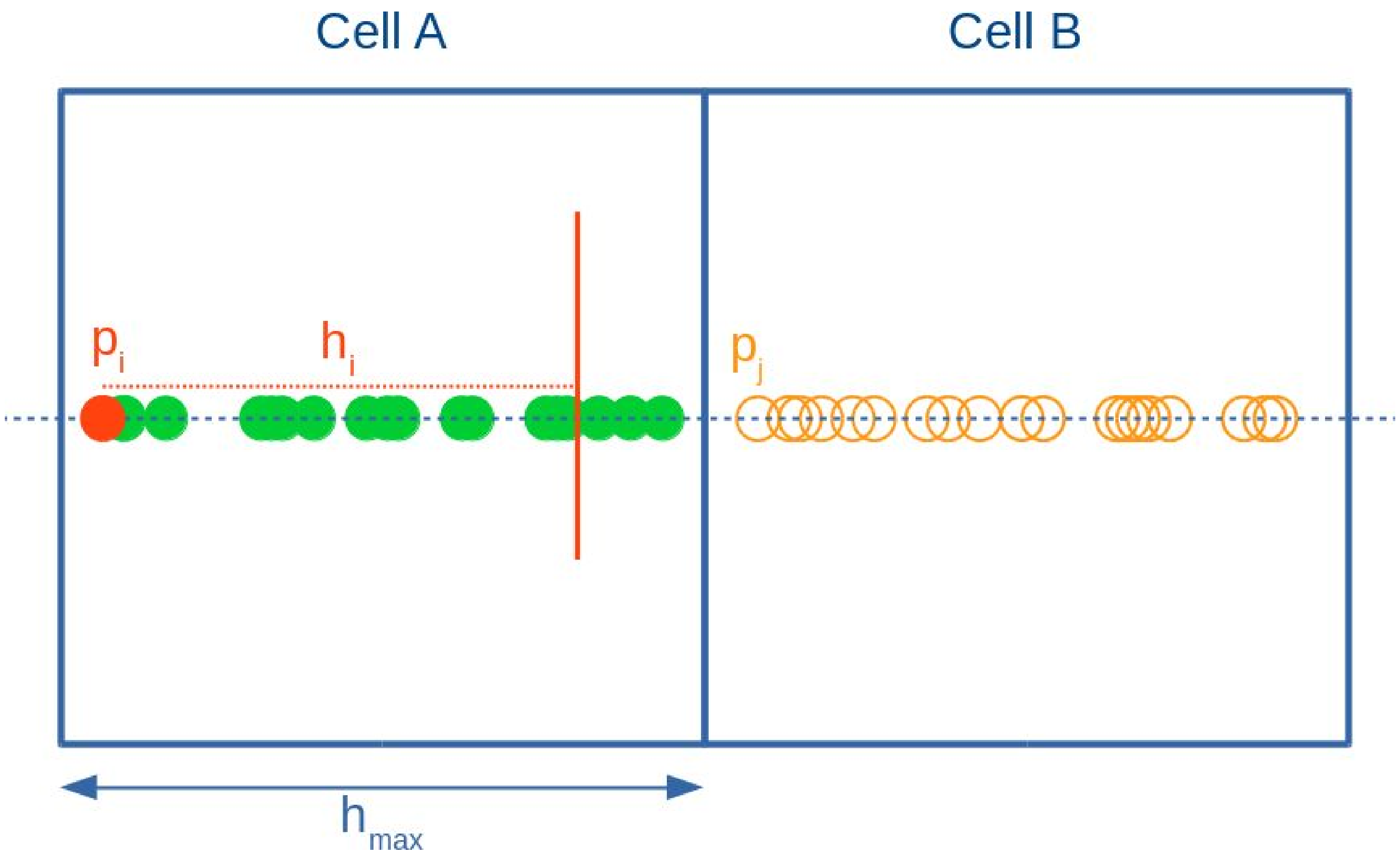


Populate Local Cache

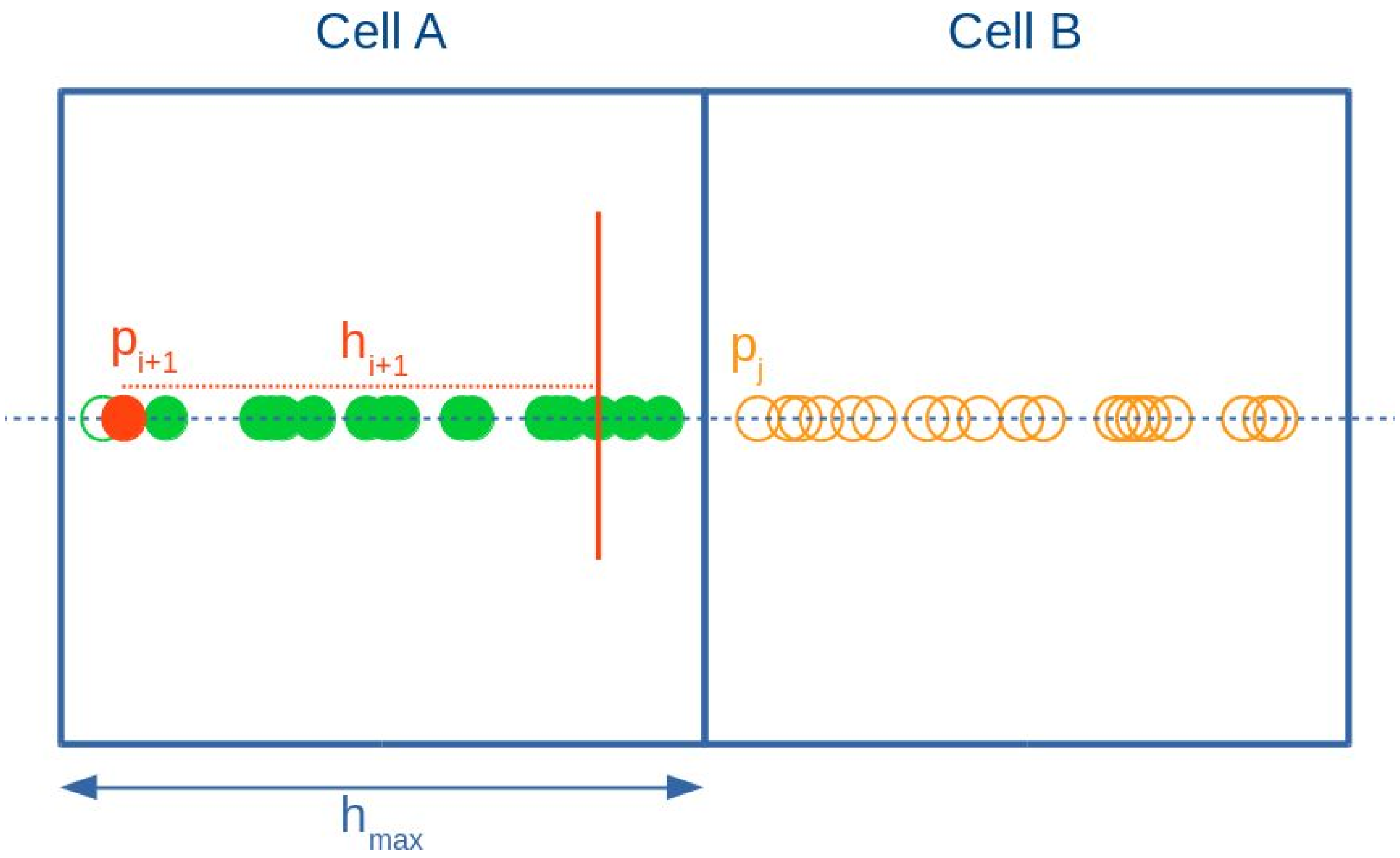
- In a uniform distribution of particles each cell pair orientation has a different number of interactions
- There are three cell pair orientations: *corner*, *edge* and *face*
- Number of interactions: *corner* < *edge* < *face*

- We want to reduce the cache overhead by only reading particles that are within range of each other
- Allows *edge* interactions to speedup instead of slowing down

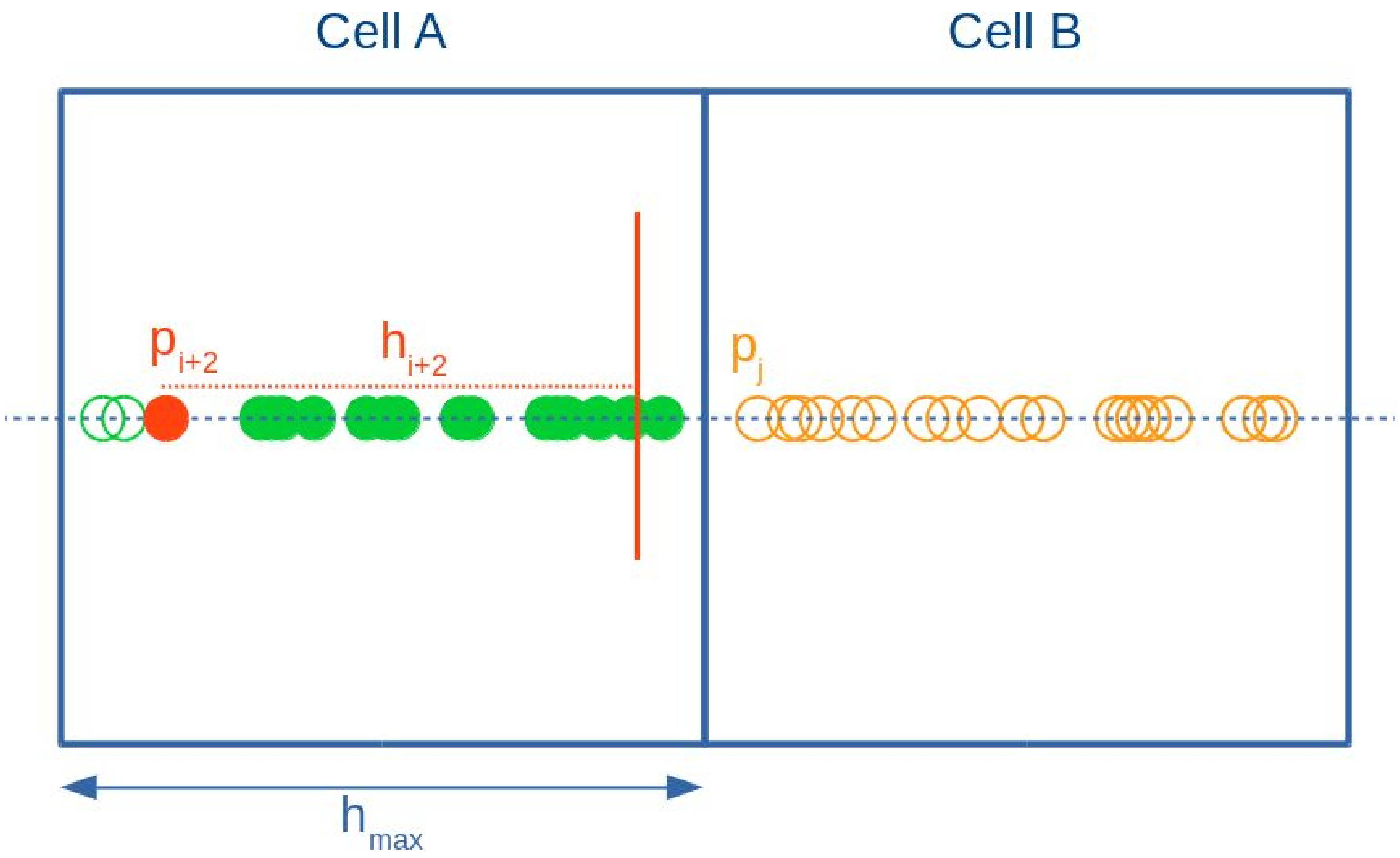
Populate Local Cache



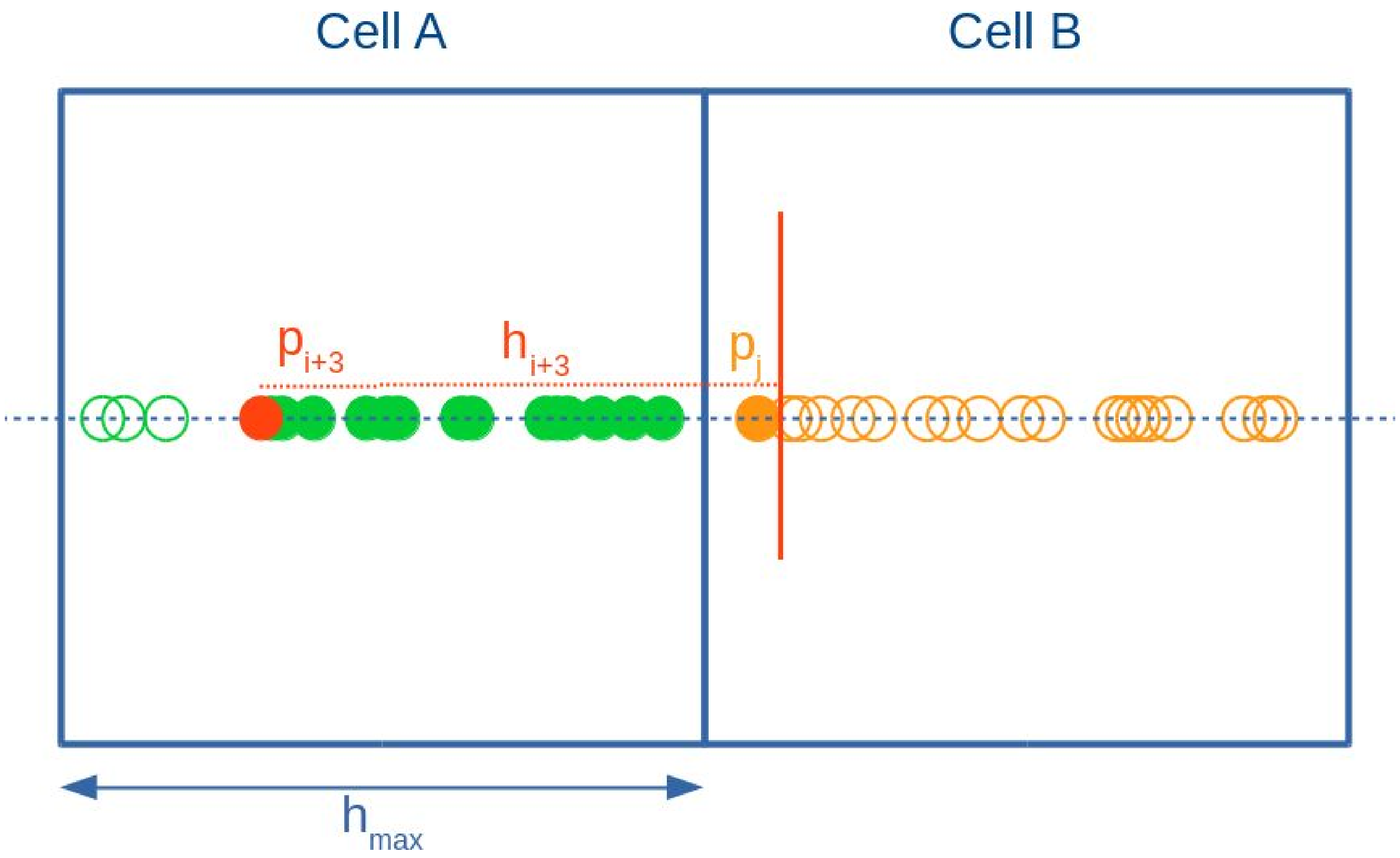
Populate Local Cache



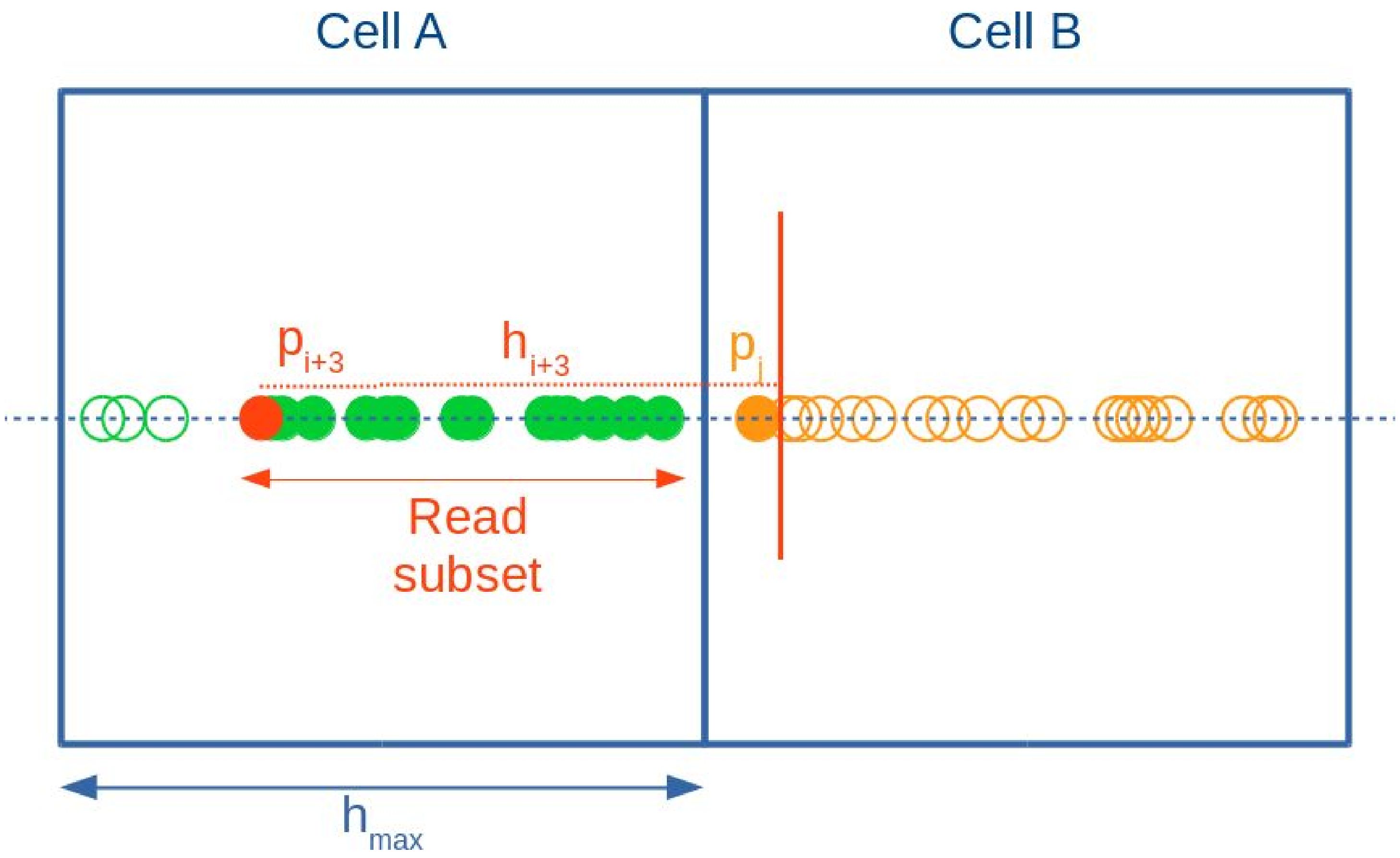
Populate Local Cache



Populate Local Cache



Populate Local Cache

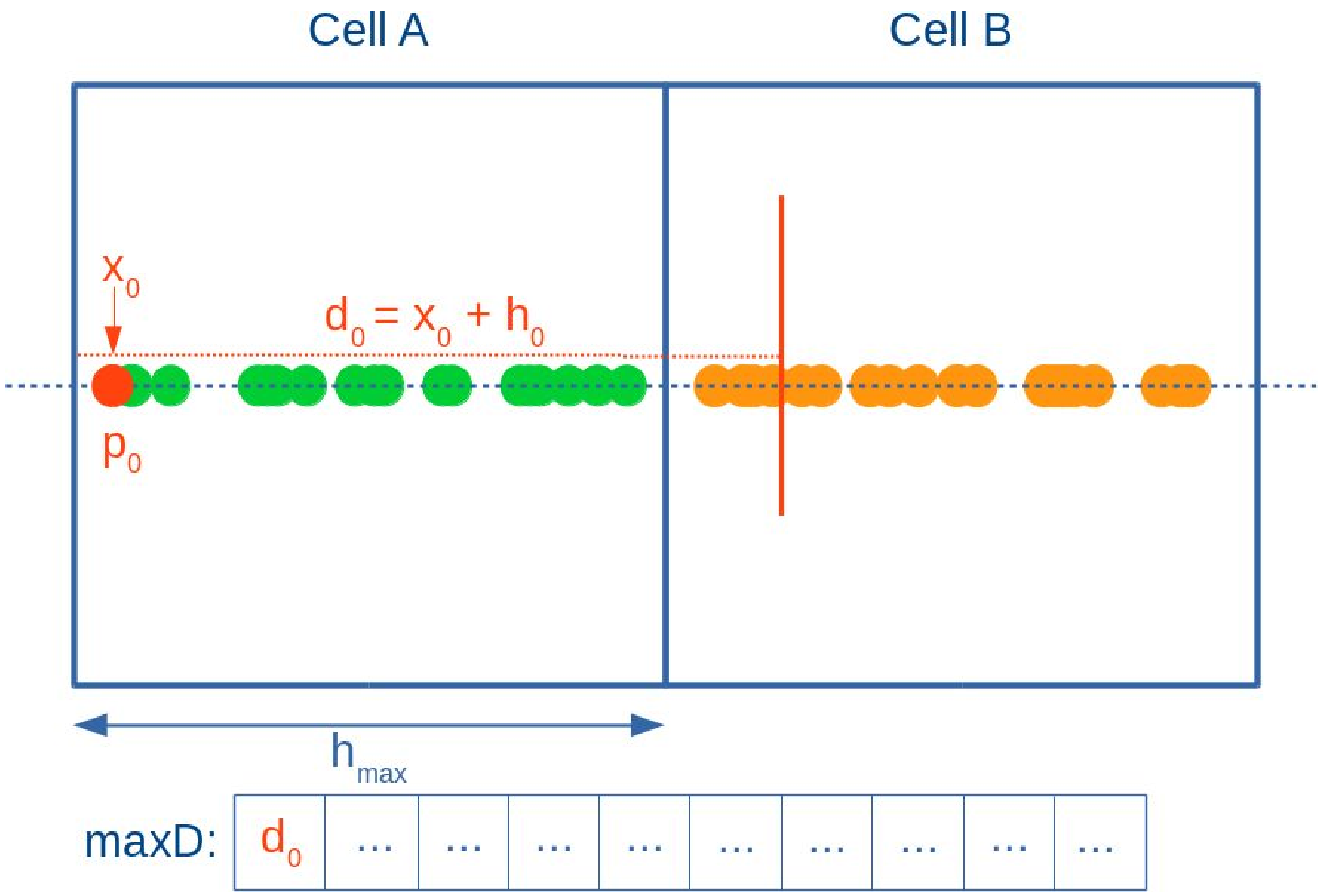


Limit Loop Bounds

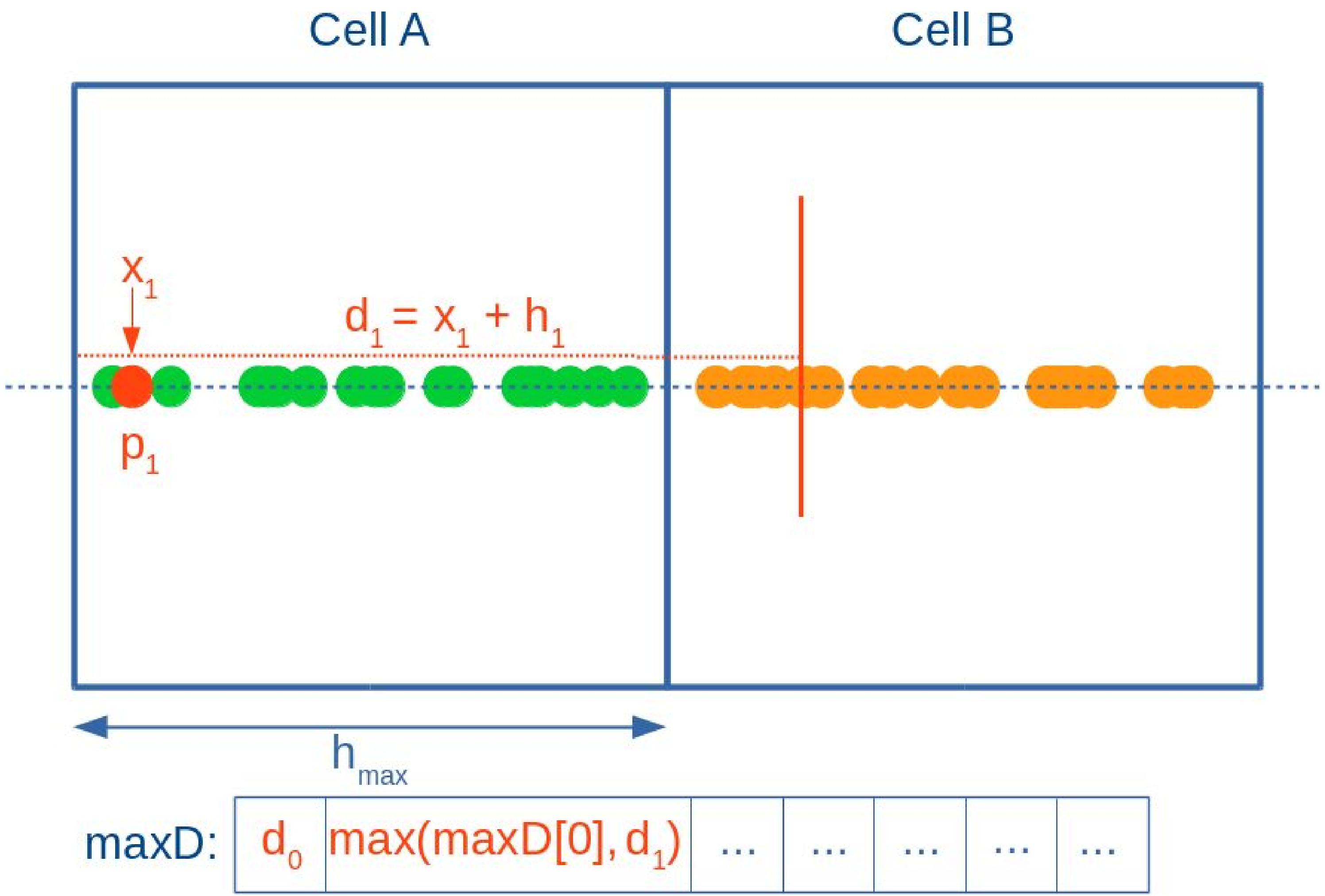
- For each particle we loop over every candidate in the neighbouring cell
- Even though most will be out of range as we move further from the interface between the two cells
- We want to reduce the number of distance calculations even further

- Form array of maximum distances into neighbouring cell
- Use array to limit the number of particles looped over in the neighbouring cell

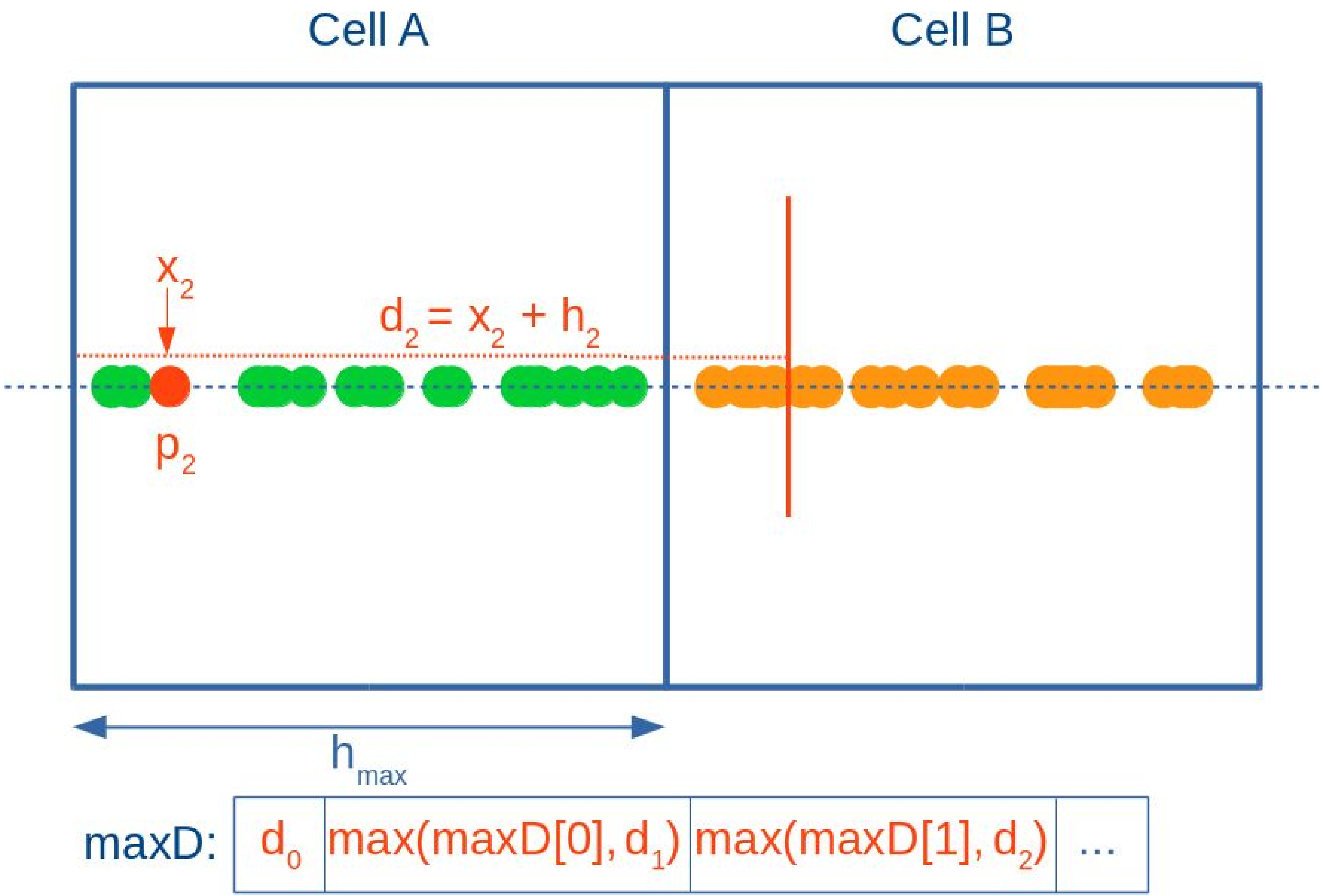
Limit Loop Bounds



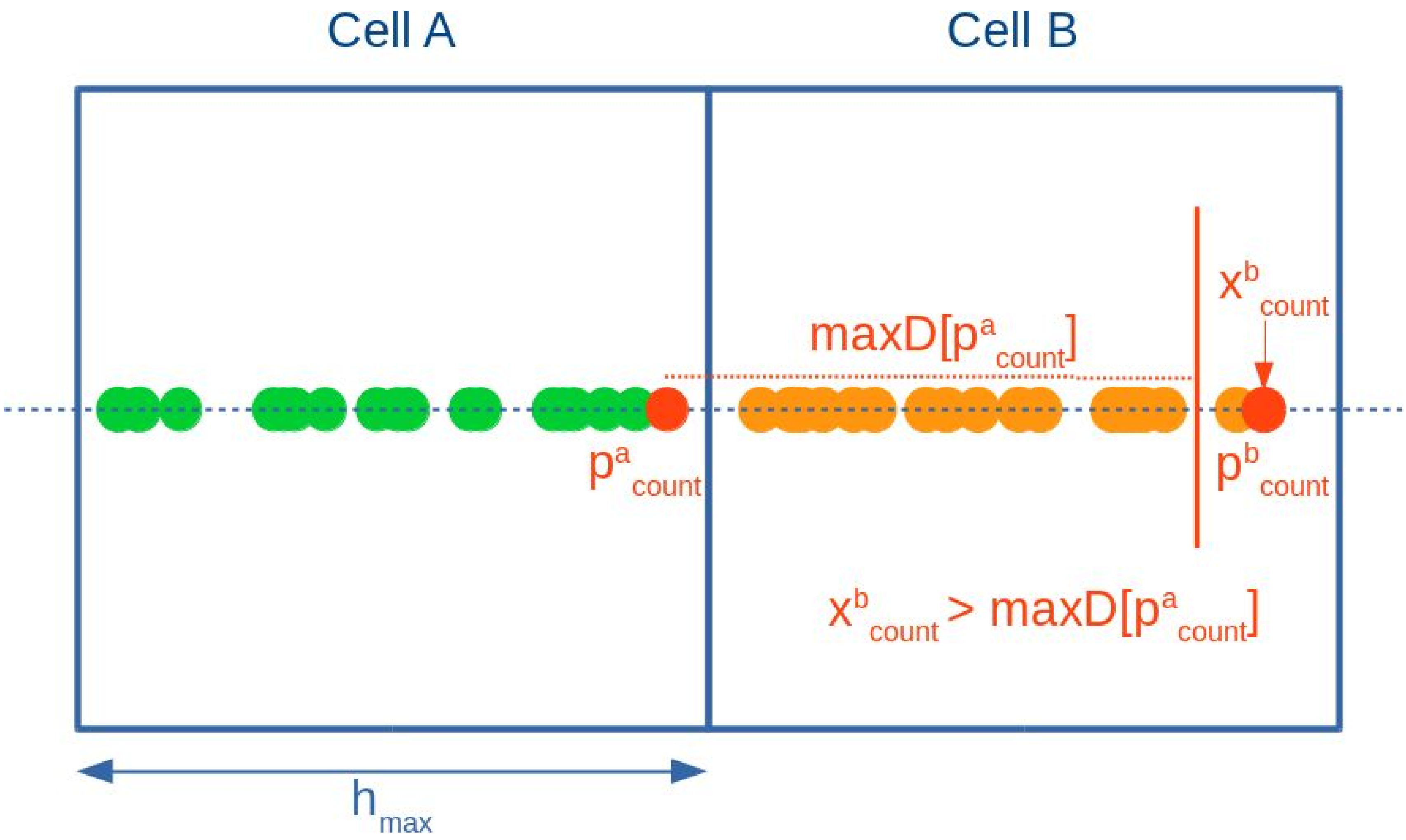
Limit Loop Bounds



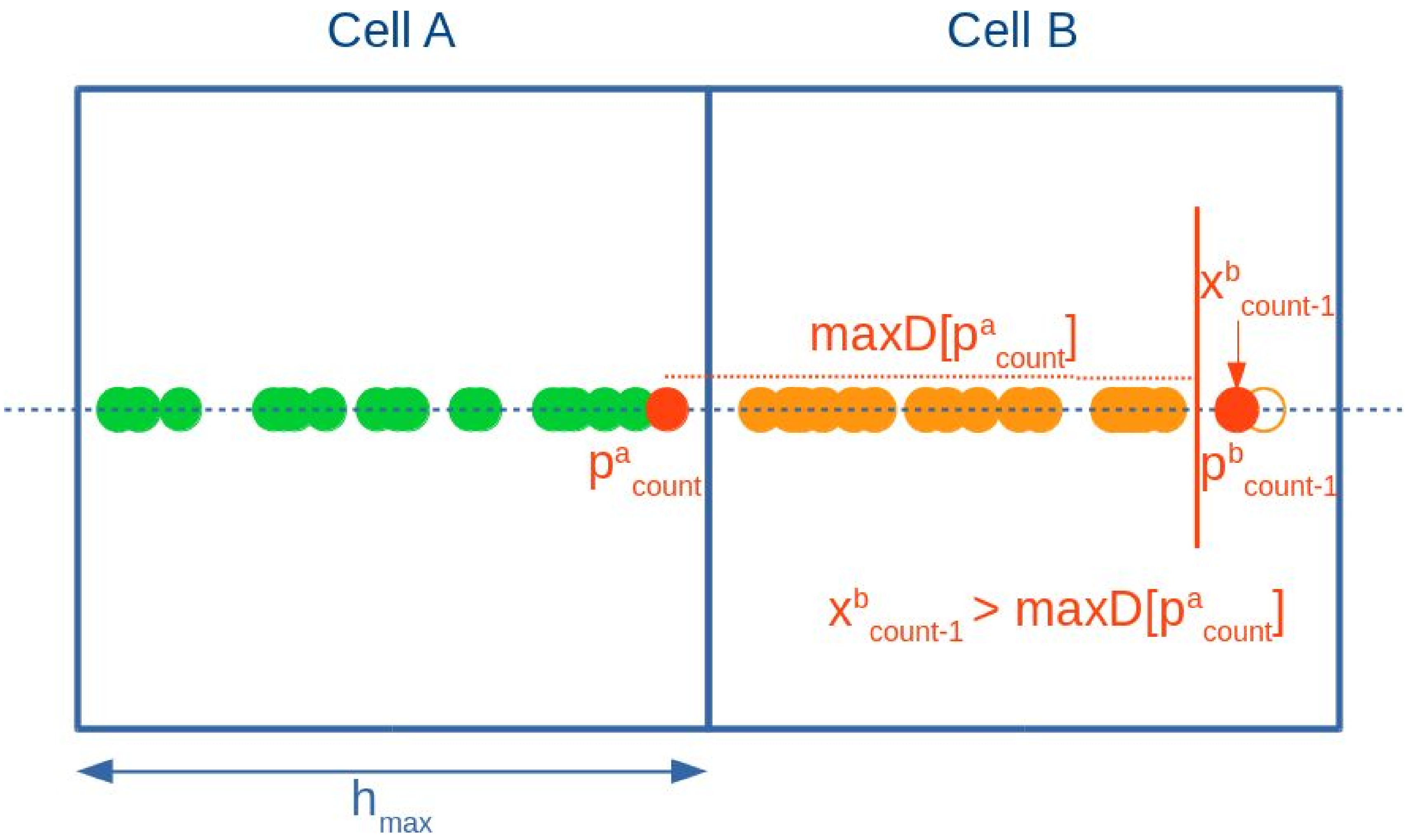
Limit Loop Bounds



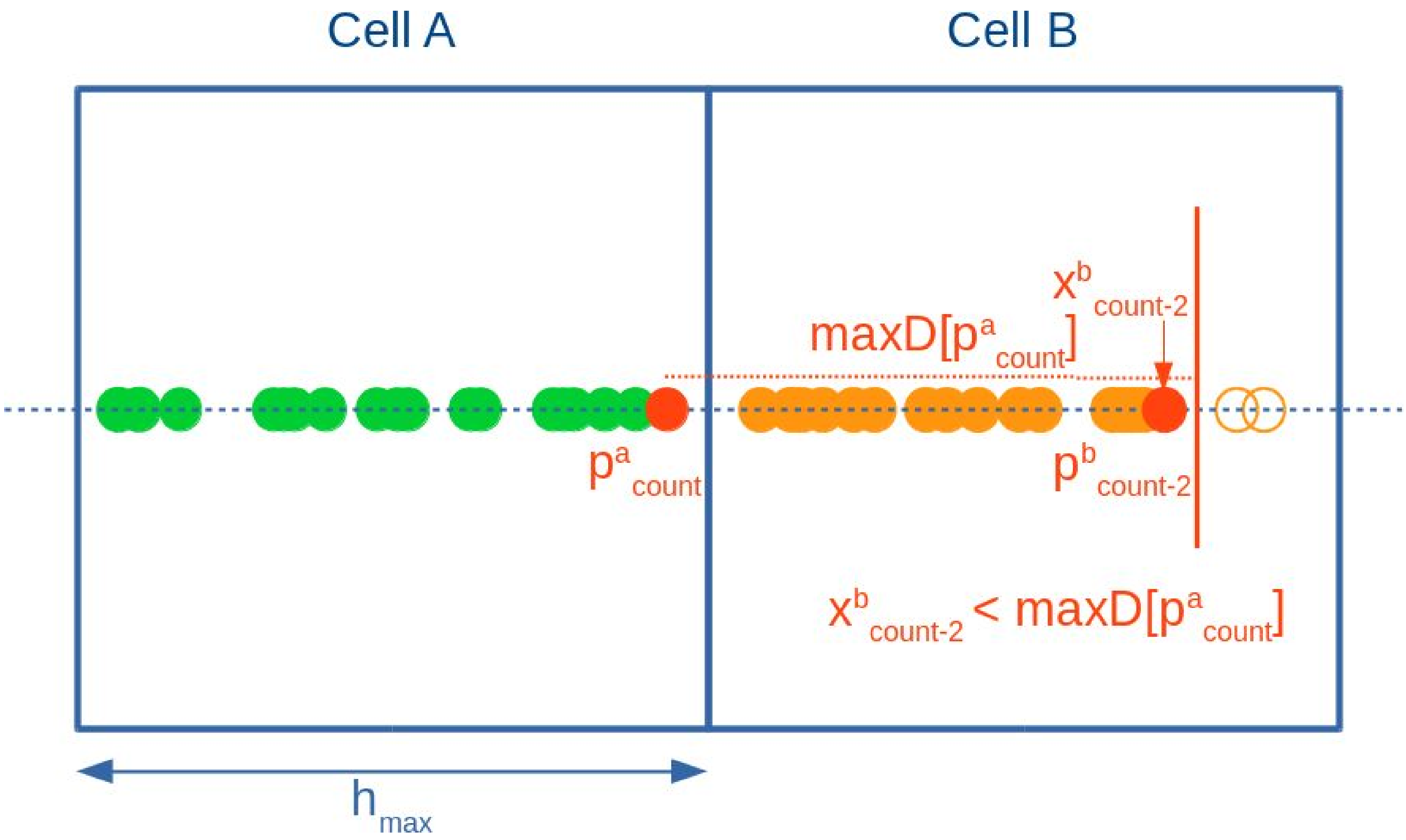
Limit Loop Bounds



Limit Loop Bounds



Limit Loop Bounds



Calculating Interactions

- Particle density interactions are calculated using:

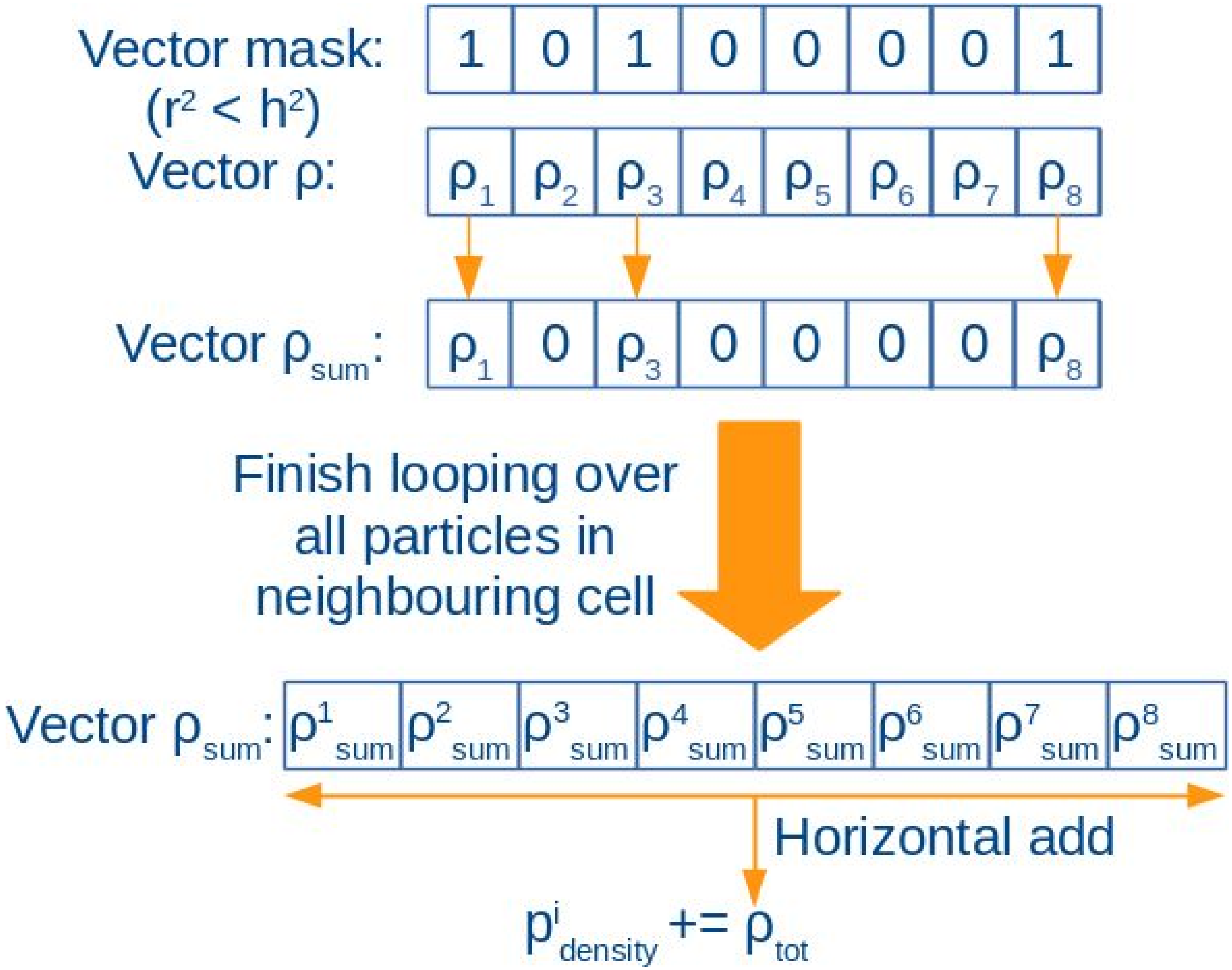
$$\rho(\mathbf{r}) = \sum_{b=1}^{N_{neigh}} m_b W(\mathbf{r} - \mathbf{r}_b, h)$$

- *W* is the weight function which is a low order polynomial

<u>SIMD Implementation</u>

- Use intermediate vectors to accumulate sum of particle updates in interaction function
- Perform horizontal add on these vectors and update the particles
- Decreases the amount of writes to memory

Calculating Interactions



Calculating Interactions

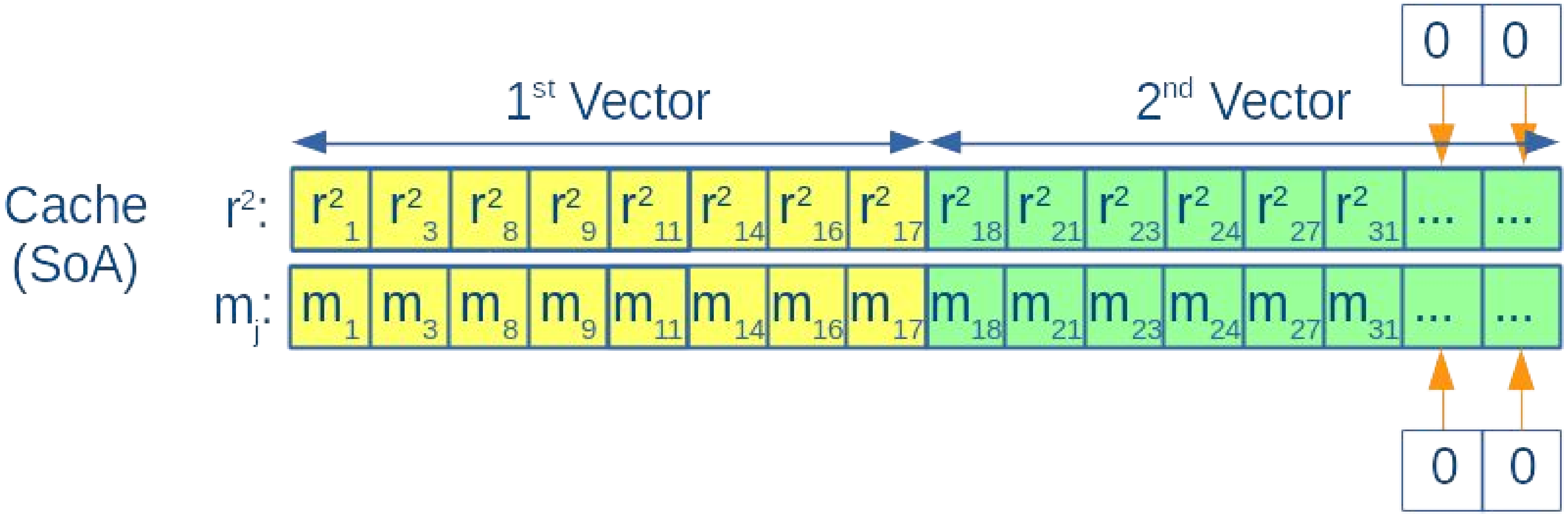
```
vector densitySum;
density = setzero();

for (int pjd = 0; pjd < icount; pjd+=VEC_SIZE) {
  INTERACT(&r2[pjd], &dx[pjd], &dy[pjd],
           &dz[pjd], &m[pjd], &v[pjd],
           &densitySum);
}

VEC_HADD(densitySum,pi); // _mm_hadd_ps
```

Calculating Interactions

```
// AVX intrinsics
vector interactionMask
vector v_densitySum;
vector v_mj, v_wi, v_r2, v_hi2;

// Form mask
interactionMask = _mm256_cmp_ps(v_r2, v_hi2, _CMP_LT_OQ);

// Mask and add to density sum
v_densitySum = _mm256_add_ps(v_densitySum,_mm256_and_ps(interactionMask, _mm256_mul_ps(v_mj, v_wi));

// AVX-512 intrinsics
__mmask16 interactionMask

// Form mask
interactionMask = _mm512_cmp_ps_mask(v_r2, v_hi2, _CMP_LT_OQ);

// Mask and add to density sum
v_densitySum = _mm512_mask_add_ps(v_densitySum, interactionMask, _mm512_mul_ps(v_mj, v_wi), v_densitySum);
```

Padding Local Cache

- Pad vectors to remove the serial remainders and mask the result



Performance Results

- Vectorisation performance was measured using AVX, AVX2 and AVX-512 instruction sets on the following hardware:

  - Intel Xeon CPU E5-4640 @ 2.4GHz (Sandy Bridge)

  - Intel Xeon CPU E5-2697 @ 2.6GHz (Haswell)

  - Intel Xeon Phi CPU 7210 @ 1.3GHz (Knights Landing)
    - 64 cores, configured in quadrant-cache mode
- Intel Compiler 17.0.1 20161005

Performance Results

CFLAGS	Speed-up of raw particle interactions over serial version	Speed-up over unsorted brute force solution	Speed-up over sorted solution
-O3 -xAVX -no-prec-sqrt -fp-model fast=2	6.50x	9.04x	2.20x
-O3 -xCORE-AVX2 -no-prec-sqrt -fp-model fast=2	6.65x	9.35x	2.32x
-O3 -xMIC-AVX512 -no-prec-sqrt -fp-model fast=2	17.35x	13.68x	3.90x

Performance Results

CFLAGS	Speed-up of raw particle interactions over serial version	Speed-up over unsorted brute force solution	Speed-up over sorted solution
-O3 -xAVX -no-prec-sqrt -fp-model fast=2	6.50x	9.04x	2.20x
-O3 -xCORE-AVX2 -no-prec-sqrt -fp-model fast=2	6.65x	9.35x	2.32x
-O3 -xMIC-AVX512 -no-prec-sqrt -fp-model fast=2	17.35x	13.68x	3.90x

Performance Results

CFLAGS	Speed-up of raw particle interactions over serial version	Speed-up over unsorted brute force solution	Speed-up over sorted solution
-O3 -xAVX -no-prec-sqrt -fp-model fast=2	6.50x	9.04x	2.20x
-O3 -xCORE-AVX2 -no-prec-sqrt -fp-model fast=2	6.65x	9.35x	2.32x
-O3 -xMIC-AVX512 -no-prec-sqrt -fp-model fast=2	17.35x	13.68x	3.90x

Performance Results

CFLAGS	Speed-up of raw particle interactions over serial version	Speed-up over unsorted brute force solution	Speed-up over sorted solution
-O3 -xAVX -no-prec-sqrt -fp-model fast=2	6.50x	9.04x	2.20x
-O3 -xCORE-AVX2 -no-prec-sqrt -fp-model fast=2	6.65x	9.35x	2.32x
-O3 -xMIC-AVX512 -no-prec-sqrt -fp-model fast=2	17.35x	13.68x	3.90x

Conclusions and Insights

- Increased performance of algorithm using a pseudo Verlet list and particle sorting
- Implemented a local particle cache (SoA)
- Implemented a vectorisation strategy
  - Only read particles into cache that interact
  - Calculate all interactions on a particle and store results in a set of intermediate vectors
  - Perform horizontal add on intermediate vectors and update the particles with the result
  - Pad caches to prevent remainders and mask out the result
- Obtained speed-up on AVX, AVX2 and AVX512 instruction sets

Future Work

- Reduce the impact of overheads even further
- Improve vectorisation efficiency to obtain speedup closer to 8x and 16x for AVX and AVX-512 instruction sets

Questions

- Thank you for your attention
- Any questions?
- Website: www.icc.dur.ac.uk/swift/