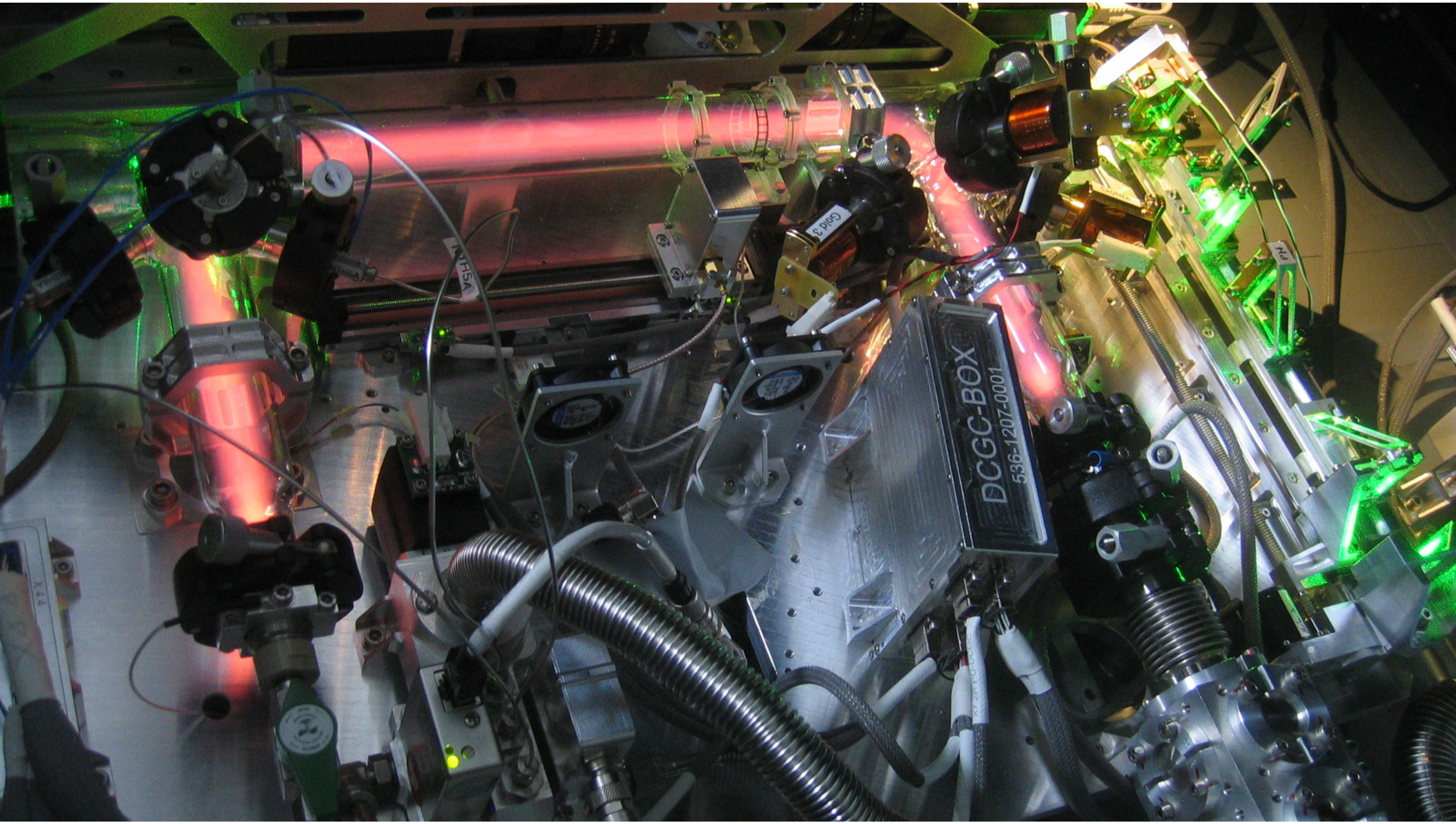
Accelerated Particle in Cell with Monte Carlo Collisions (PIC/MCC) simulation for gas discharge modeling in realistic geometries.

Péter Hartmann, Zoltán Juhász, Zoltán Donkó

Wigner Research Centre for Physics, Budapest, Hungary

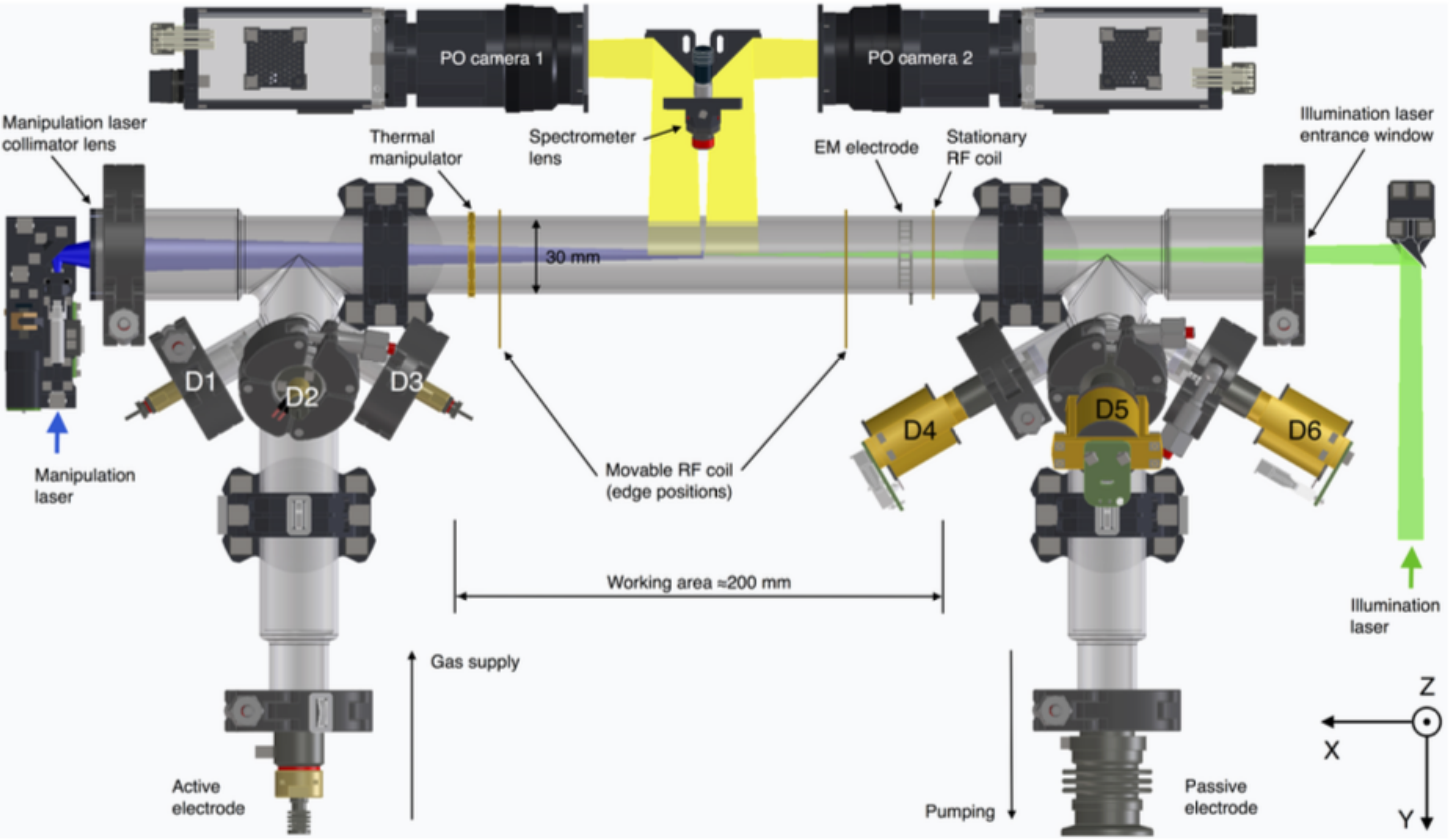
Dept. of Electrical Engineering and Information Systems, University of Pannonia, Veszprem, Hungary

Motivation: PK-4 Experiment on the ISS

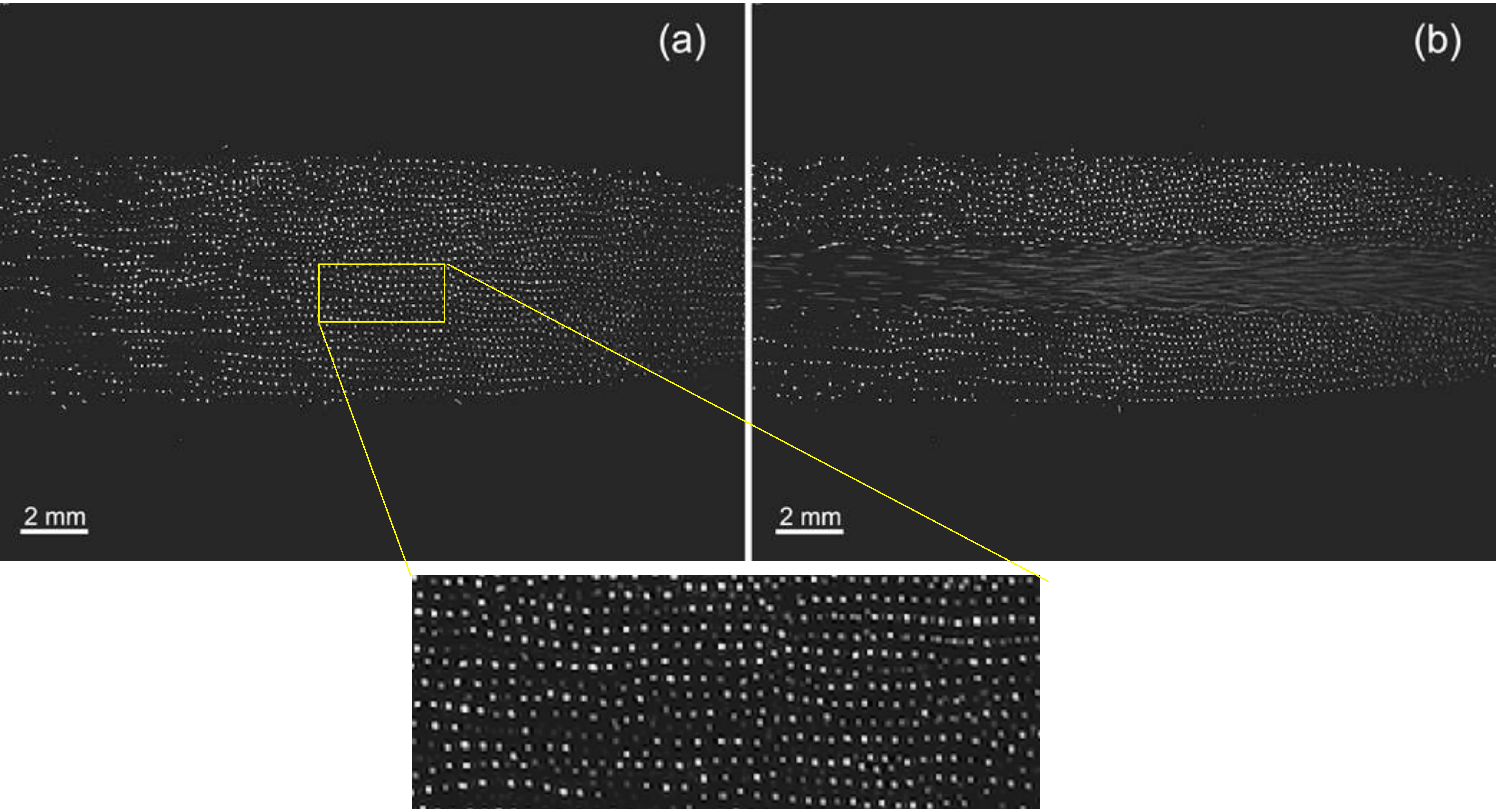


Review of Scientific Instruments **87**, 093505 (2016); doi: 10.1063/1.4962696

Motivation: PK-4 Experiment on the ISS



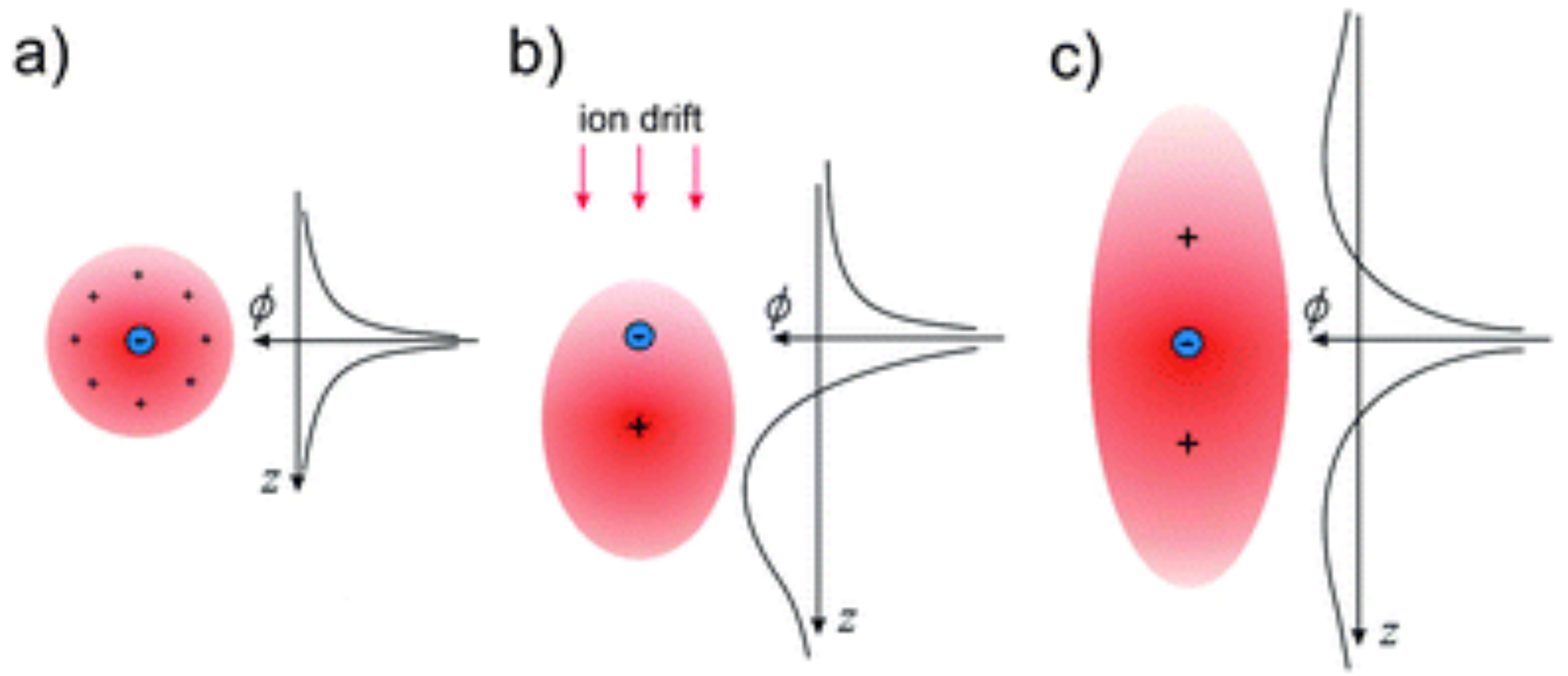
Motivation: PK-4 Experiment on the ISS

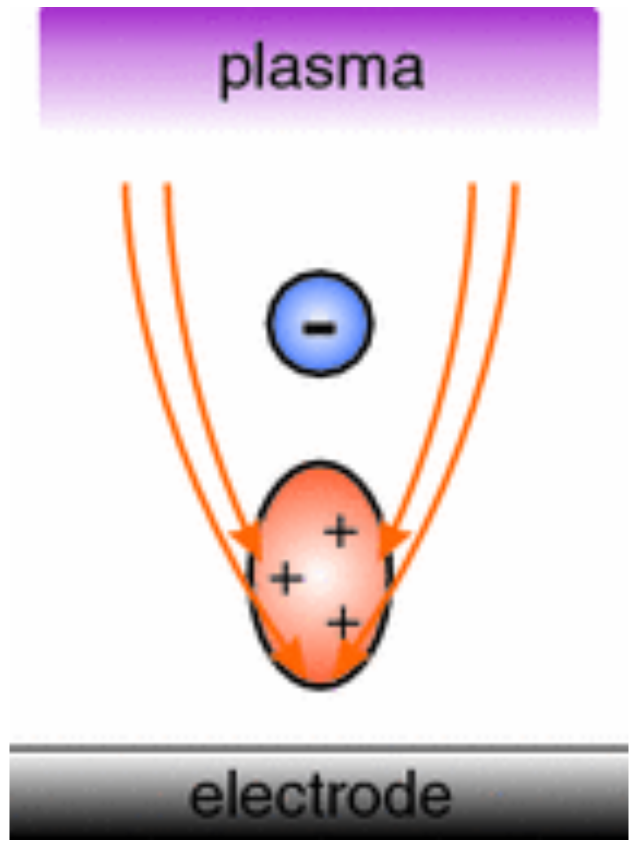


The question: Why do the dust particles align in chains?

The question: Why do the dust particles align in chains?

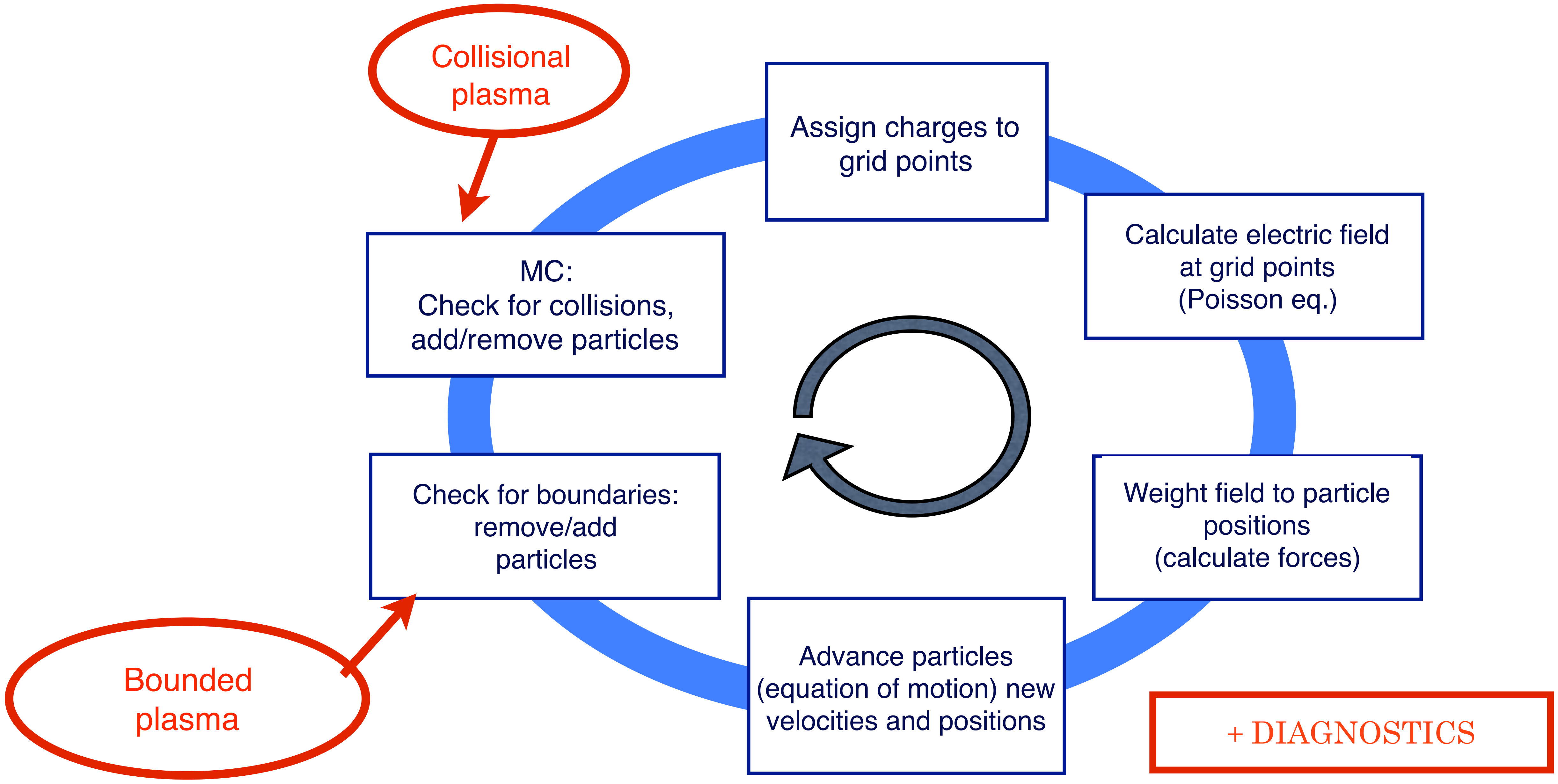
Known mechanism: ion drift and wake field formation:



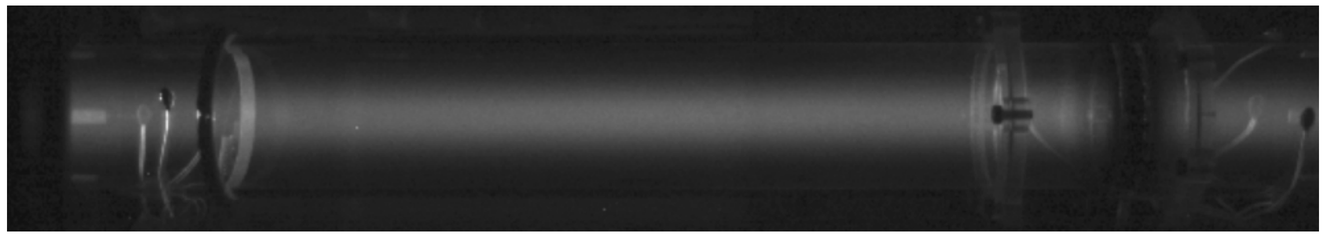


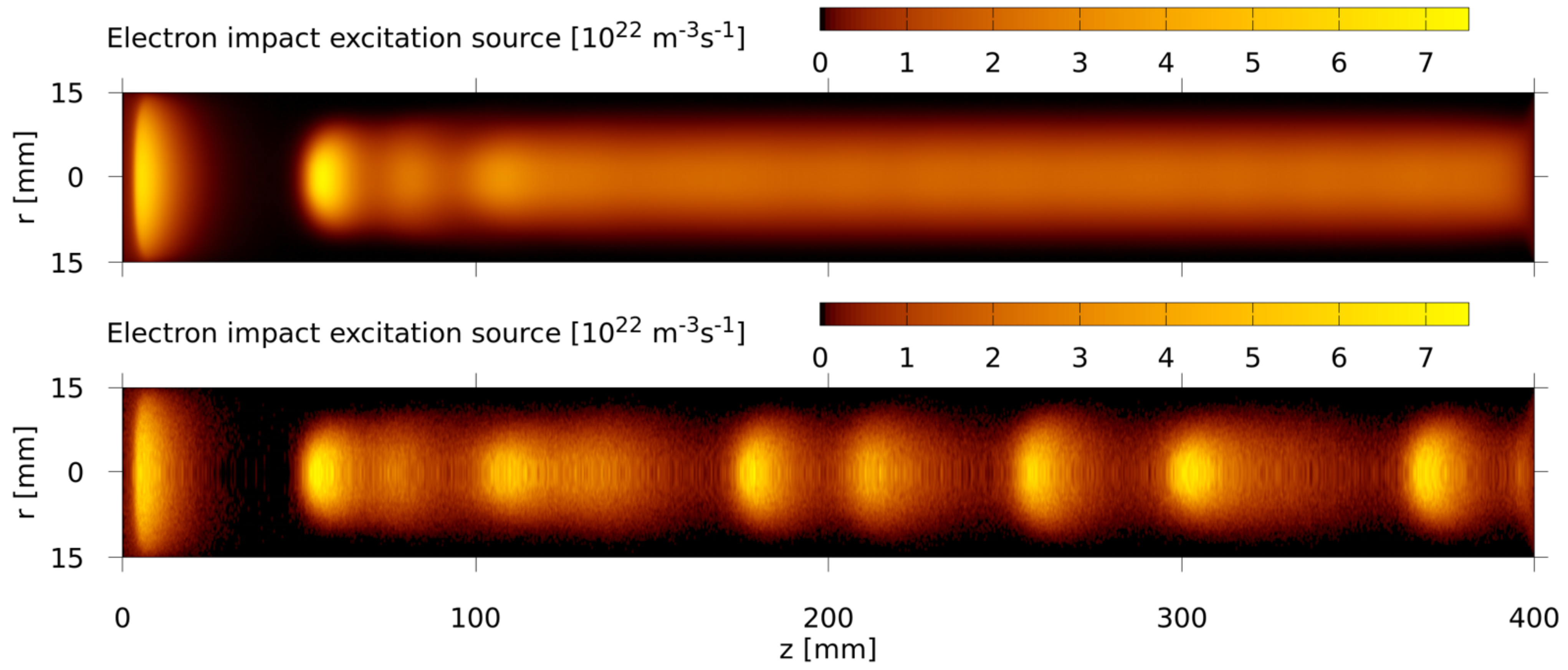
BUT: In the PK-4 discharge the electric field is very low (~3 V/cm) and the ion drift velocity is not enough!!!

Let's use numerical simulations to find discharge conditions: PIC/MCC:


Collisional plasma
Assign charges to grid points
Calculate electric field at grid points (Poisson eq.)
Weight field to particle positions (calculate forces)
Advance particles (equation of motion) new velocities and positions
Check for boundaries: remove/add particles
MC: Check for collisions, add/remove particles
Bounded plasma
+ DIAGNOSTICS

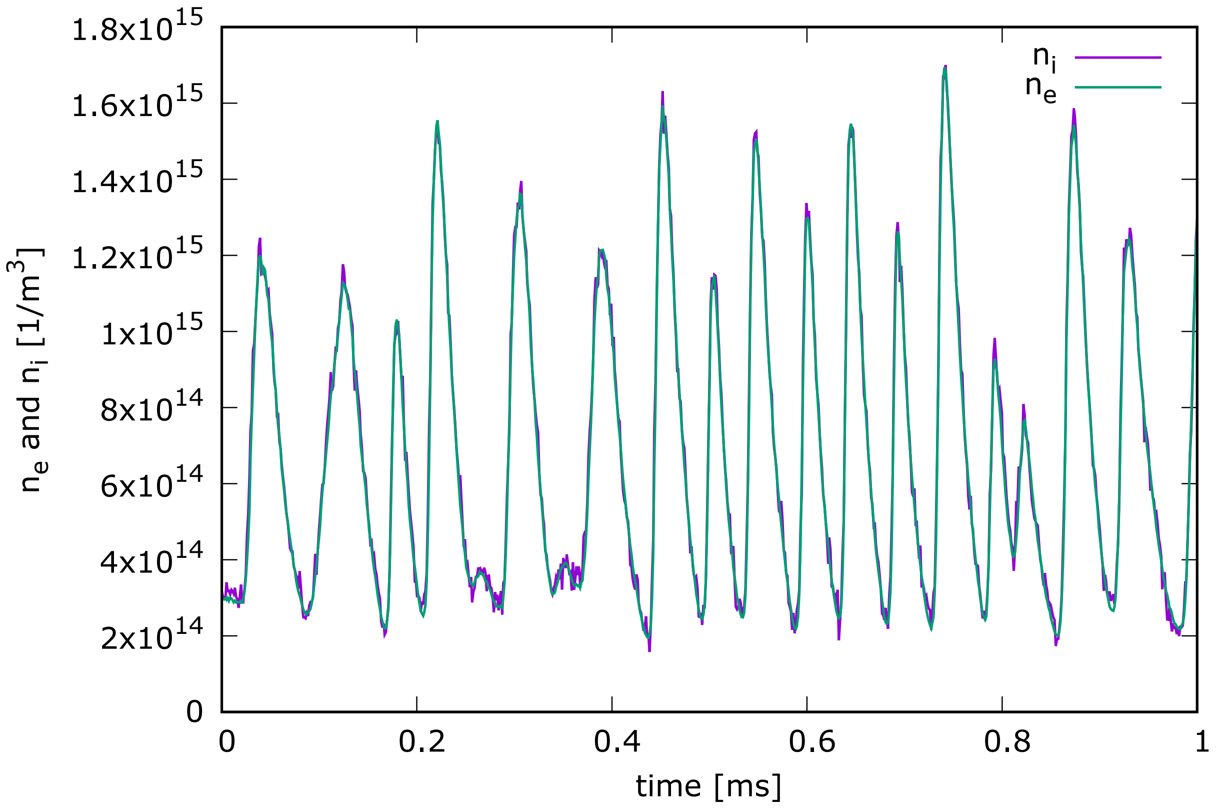
Ionization waves:

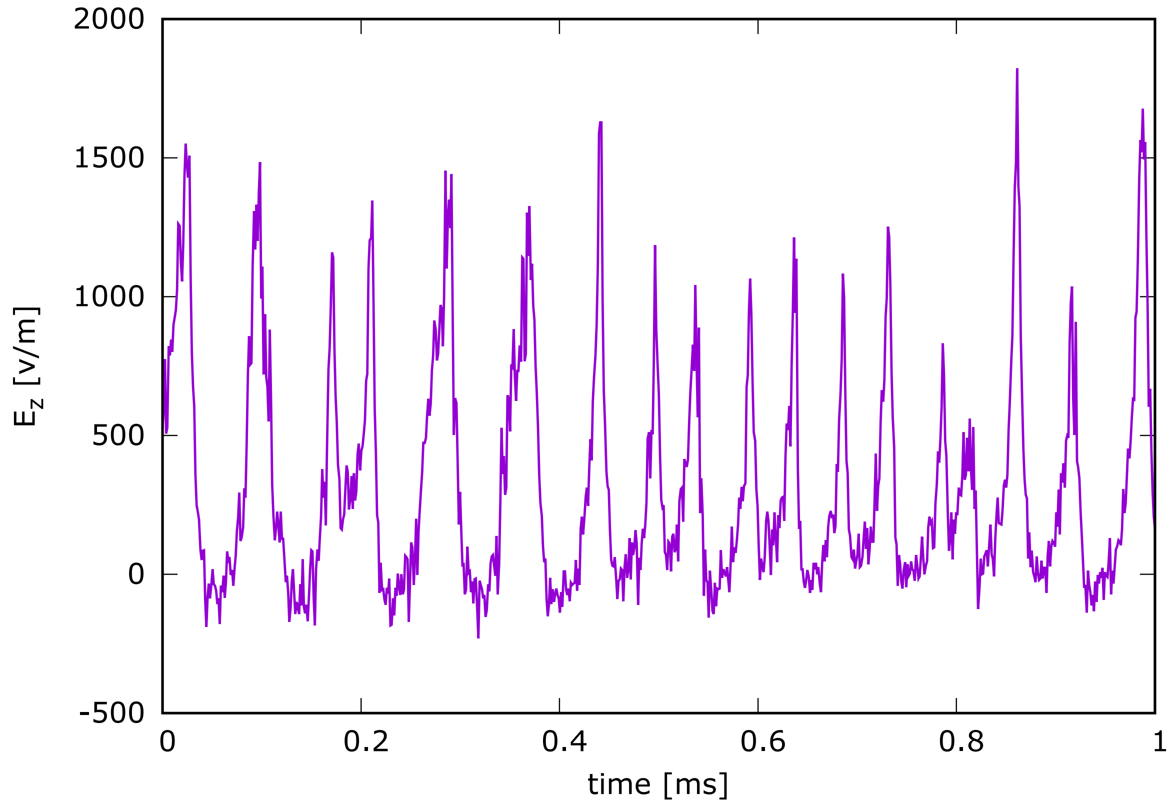

experiment



PIC
1 ms

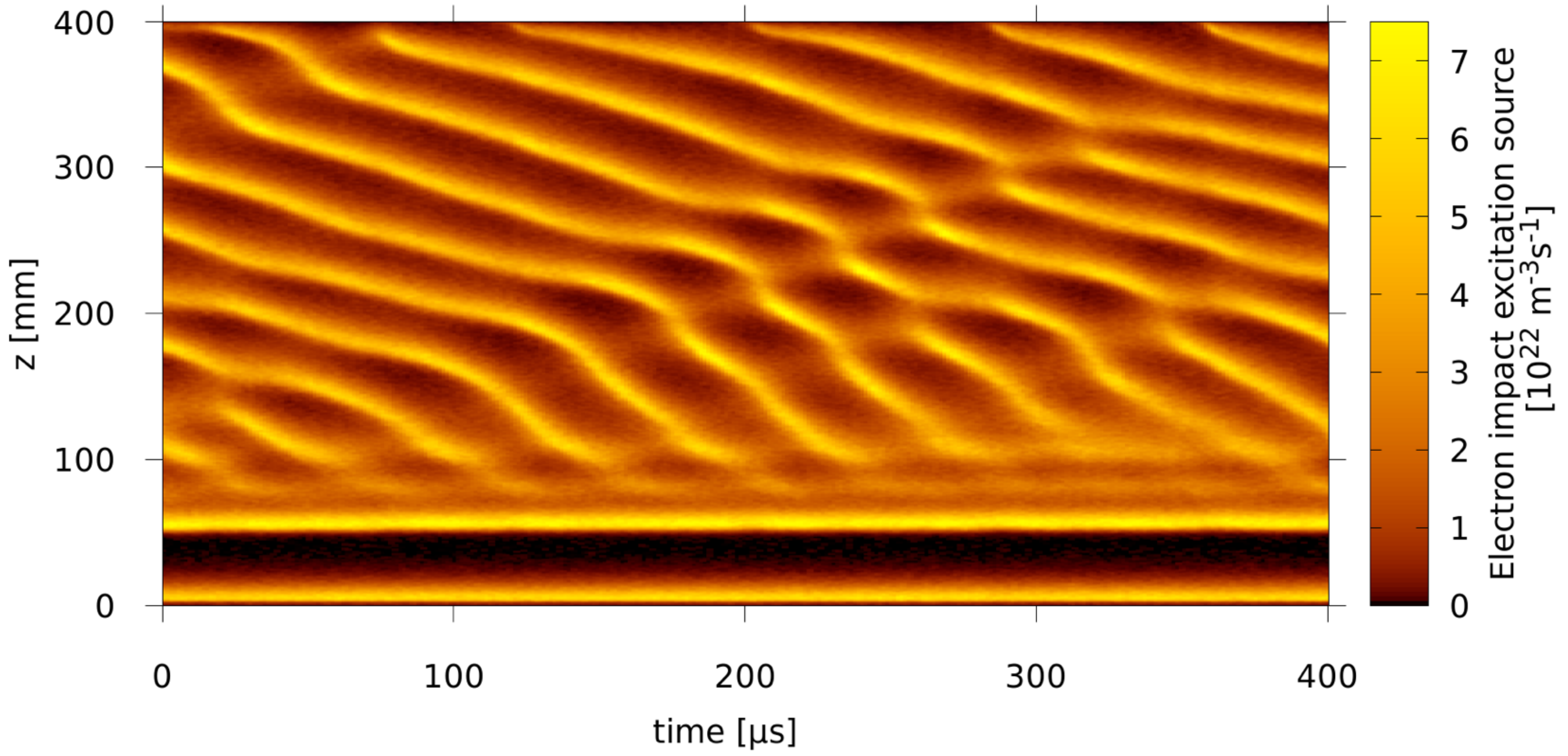
PIC
1.2 μs



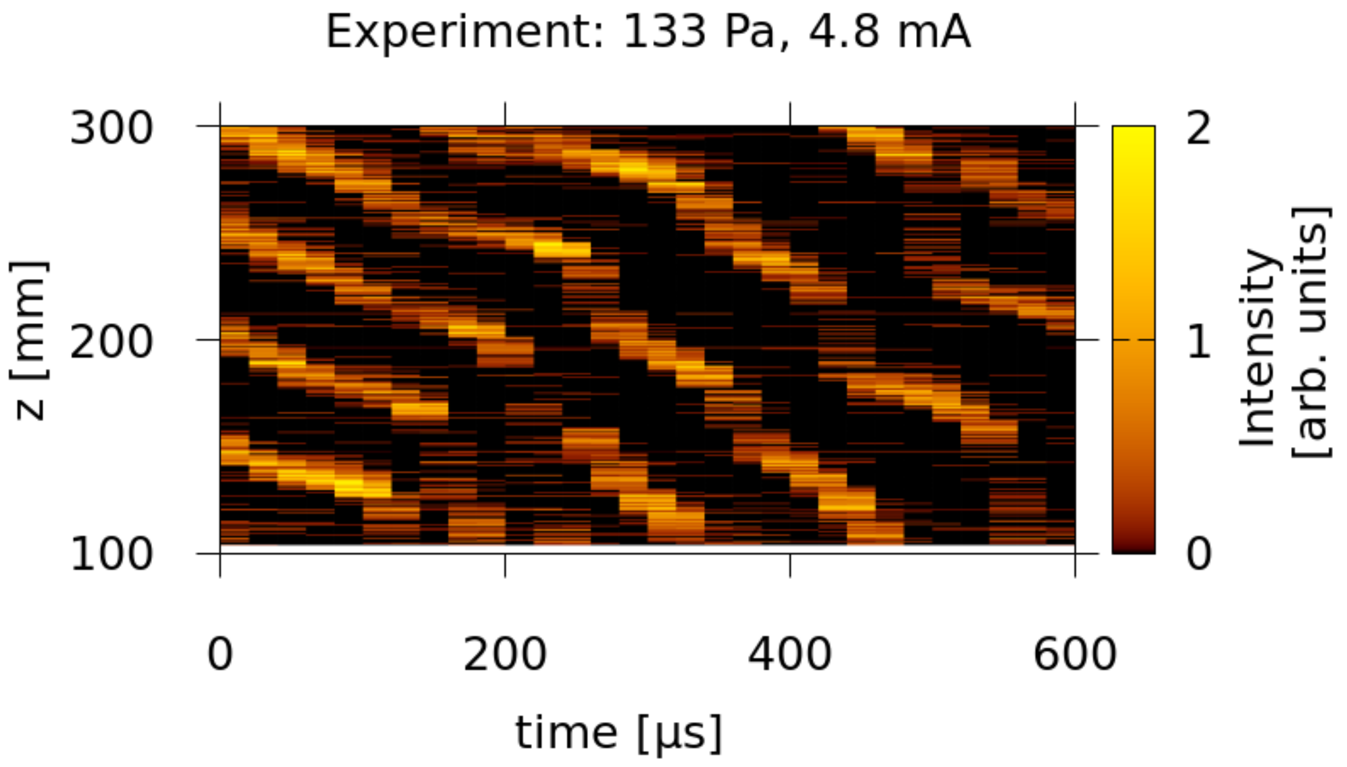


Ionization waves:

PIC



ground based experiment



Simulation details:

Model parameters:

- super-particle weight: ~10^5
- super-particle number: 10^6 - 10^7
- particle pusher: leap-frog algorithm
- field solver: black-red successive over-relaxation (SOR)
- Mesh size: 128(r) x 4096(z)

System parameters:

- Neon gas (electrons: elastic scattering, excitation, ionization; Ne^+ ions: isotropic elastic and charge transfer collisions; Ne^m metastables: diffusion and Penning ionization; Biagi cross-section database)
- absorbing electrodes
- cylindrical geometry with floating dielectric wall, wall charging calculation is included

Implementation:

- Massively parallel implementation on NVIDIA GPUs using the CUDA-C language extension.
- Speedup factor ~100 with respect to our CPU version using MPI parallelizations, reducing simulation execution time from 3 months to 1 day

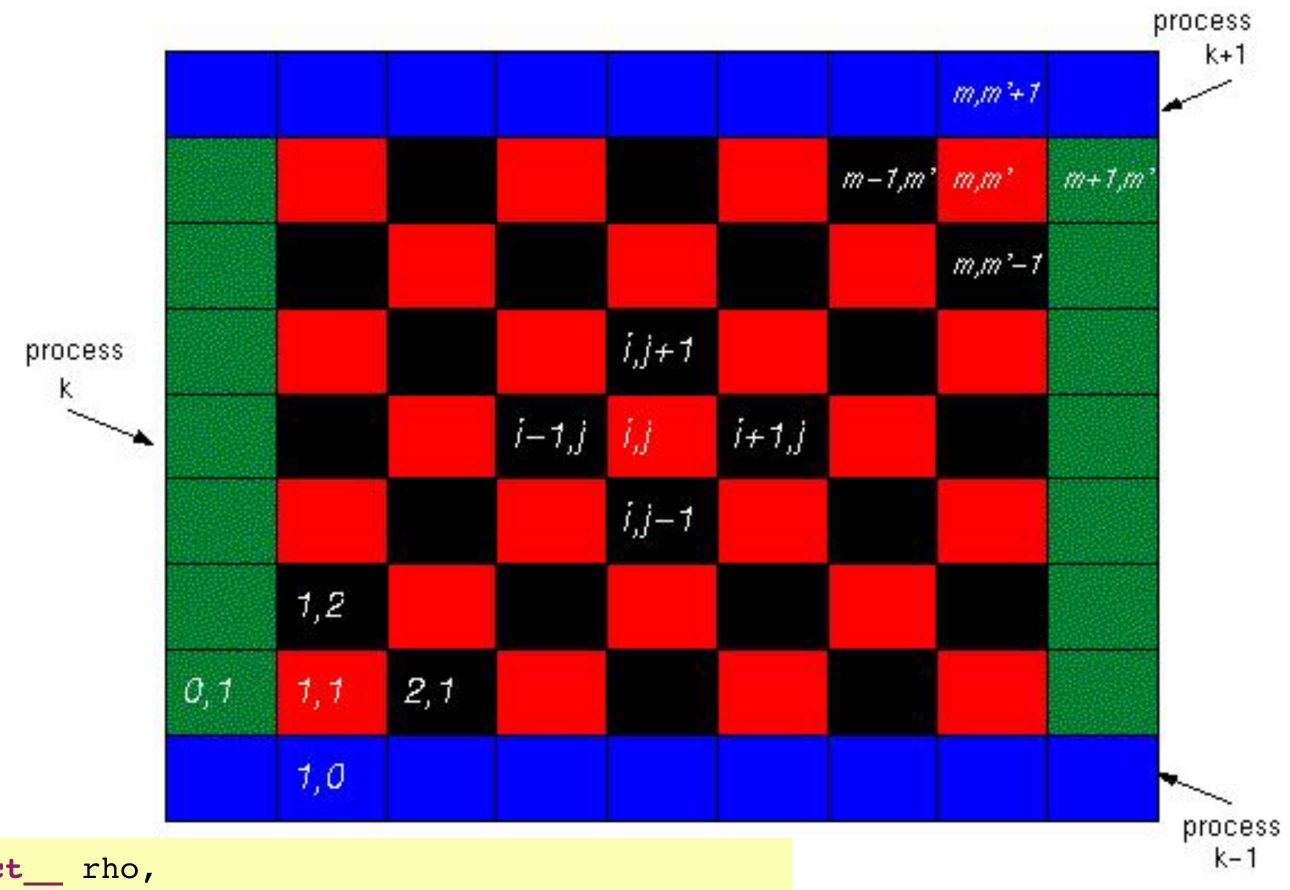
Implementation details: critical features

- Poisson Solver - GPU
- particle pusher - GPU
- data reduction - GPU atomic
- table search - GPU
- random number generator - GPU
- collision branching - GPU
- particle creation - GPU
- particle removal - CPU periodically

Implementation details:

Poisson solver: black-red successive over-relaxation

Superposition principle: only space-charge contribution



```
__global__ void cudo_half_SOR(float *__restrict__ pot, float *__restrict__ rho,
    int *__restrict__ boundary, int color, int calc_residual, float *__restrict__ residual){

  int Nz = Params.Nz;
  for(int ii = 0; ii < Params.BRn; ii++){

    int idx = blockIdx.x * Nz + 2*(threadIdx.x + ii*blockDim.x) + (color + blockIdx.x) % 2;

    if( (idx < Params.N) && (boundary[idx] == 0) ){

      int i = idx / Nz;
      int j = idx % Nz;
      float ir2 = Params.ir2;
      float iz2 = Params.iz2;
      float irdr = 0.0f;
      if (i > 0) irdr = 0.5f * ir2 / (float) i;
      float w = Params.w;
      //float sf = fmaxf((float)i, 0.125);
      float sf = 1.0f;
      int ln = abs(i-1);
      int rn = (i+1);
      float newpot = (1.0-w)*pot[idx]
                          + w * ((ir2-irdr)*pot[ln*Nz+j] + (ir2+irdr)*pot[rn*Nz+j] + iz2*pot[idx-1] +
        iz2*pot[idx+1]
                         + Params.Poisson_factor/sf*rho[idx]) / (2.0*ir2 + 2.0*iz2);
      if(calc_residual == 1) residual[idx] = fabsf(newpot - pot[idx]);
      pot[idx] = newpot;
      //if (threadIdx.x == 0) printf("%.2e   %.2e\n", ddd, newpot);
    }
  }
}
```

Implementation details:

Random number generator

```
__device__ float curand32_float(particle_type * seed){
  unsigned int u = seed[0].RS_u;
  unsigned int v = seed[0].RS_v;
  unsigned int w1 = seed[0].RS_w1;
  unsigned int w2 = seed[0].RS_w2;
  u = u * (unsigned int) 2891336453 + (unsigned int) 1640531513;
  v ^= v >> 13; v ^= v << 17; v ^= v >> 5;
  w1 = 33378 * (w1 & 0xffff) + (w1 >> 16);
  w2 = 57225 * (w2 & 0xffff) + (w2 >> 16);
  unsigned int x = u ^ (u << 9); x ^= x >> 17; x ^= x << 6;
  unsigned int y = w1 ^ (w1 << 17); y ^= y >> 15; y ^= y << 5;
  seed[0].RS_u = u;
  seed[0].RS_v = v;
  seed[0].RS_w1 = w1;
  seed[0].RS_w2 = w2;
  w1 = (x + v) ^ (y + w2);
  return 2.32830641E-10 * w1;
}
```

based on Numerical Recipes 3rd Ed.
CURAND turned out to be way to slow

Implementation details: Data storage

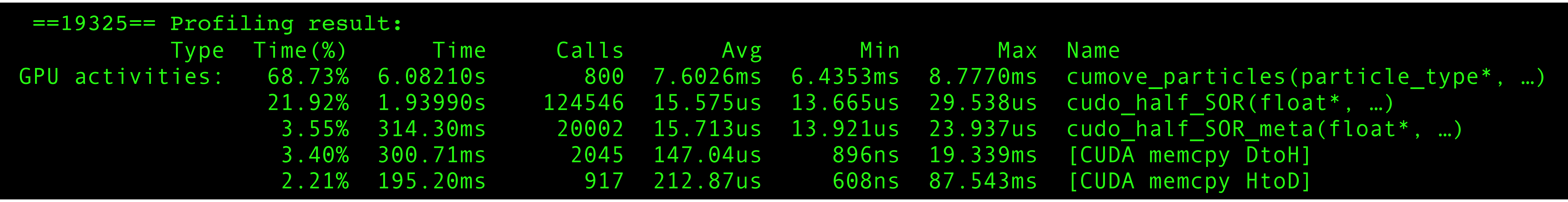
particles: AoS

```
struct particle_type{
  float x, y, r, z;         // m
  float vx, vy, vz;         // m/s
  float S;                  // Monte Carlo scatterint integral
  int   coll;               // collision type;
  unsigned int RS_u;        // seed variables for RNG
  unsigned int RS_v;        // seed variables for RNG
  unsigned int RS_w1;       // seed variables for RNG
  unsigned int RS_w2;       // seed variables for RNG
};
```

system parameters: constant memory in structure

```
struct PIC_Params_Type
{
  int Nr, Nz, N;
  float Dr, Dz;
  float iz2, ir2;
  float w, dt;
  float Lr, Lz;
  float ion_gamma;
  float Poisson_factor;
  float Metastable_factor;
  float cs_lE_min, cs_ldE;
  float temperature;
  float s2epm;
  int   MAX_particles;
  int   BRn;
  float weight;
  float charge_over_mass[N_species];
  int   N_reactions[N_species];
  float mass_ratio[N_species];
```

Performance



```
 ==19325== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
GPU activities:    68.73%  6.08210s       800  7.6026ms  6.4353ms  8.7770ms  cumove_particles(particle_type*, …)
                   21.92%  1.93990s    124546  15.575us  13.665us  29.538us  cudo_half_SOR(float*, …)
                    3.55%  314.30ms     20002  15.713us  13.921us  23.937us  cudo_half_SOR_meta(float*, …)
                    3.40%  300.71ms      2045  147.04us     896ns  19.339ms  [CUDA memcpy DtoH]
                    2.21%  195.20ms       917  212.87us     608ns  87.543ms  [CUDA memcpy HtoD]
```

Poisson Flops: 64 x 2048 x 22 / 0.000015 ≈ 200 G
Push Flops: 4.6M x 280 / 0.0076 ≈ 200 G
Data transfer per step: 4 MB

Real life performance:

Convergence can be reached in 1-2 days in contrast to the MPI-CPU version with a convergence time of 3 month.