Cannonballs in (dusty) turbulence

Drag, Accretion & Concentrations

H. Homann

Laboratoire Lagrange, Université de Nice Sophia Antipolis,
Observatoire de la Côte d'Azur, Nice

in collaboration with

J. Bec, T. Guillot, P. Tanga, F. Laenen
Laboratoire Lagrange, Université de Nice Sophia Antipolis,
Observatoire de la Côte d'Azur, Nice

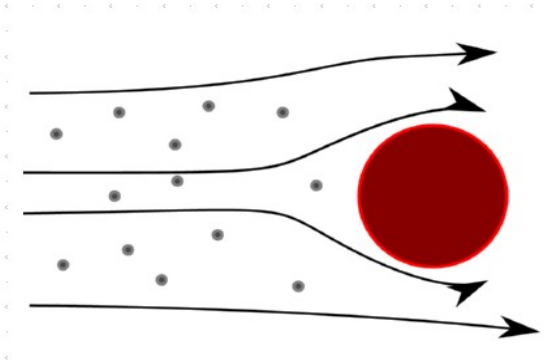
R. Grauer
Institut für theoretische Physik I, Ruhr-Universität Bochum, Germany

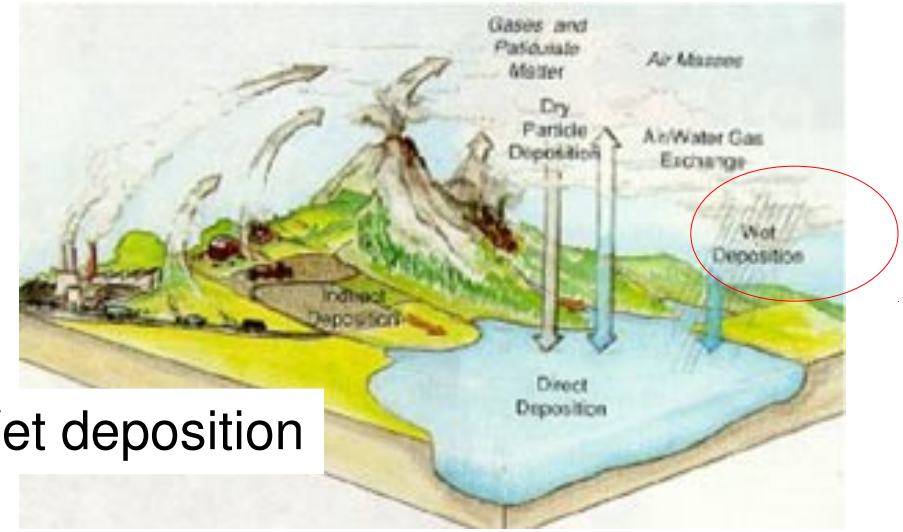
C.W. Ormel
Anton Pannekoek Institute for Astronomy, University of Amsterdam, The Netherlands

S. Ida
Earth-Life Science Institute, Tokyo Institute of Technology, 152-8550 Tokyo, Japan

Motivation - Applications

CONTEXT: Large particle moving through a (turbulent) fluid seeded with small inertial particles:





Wet deposition

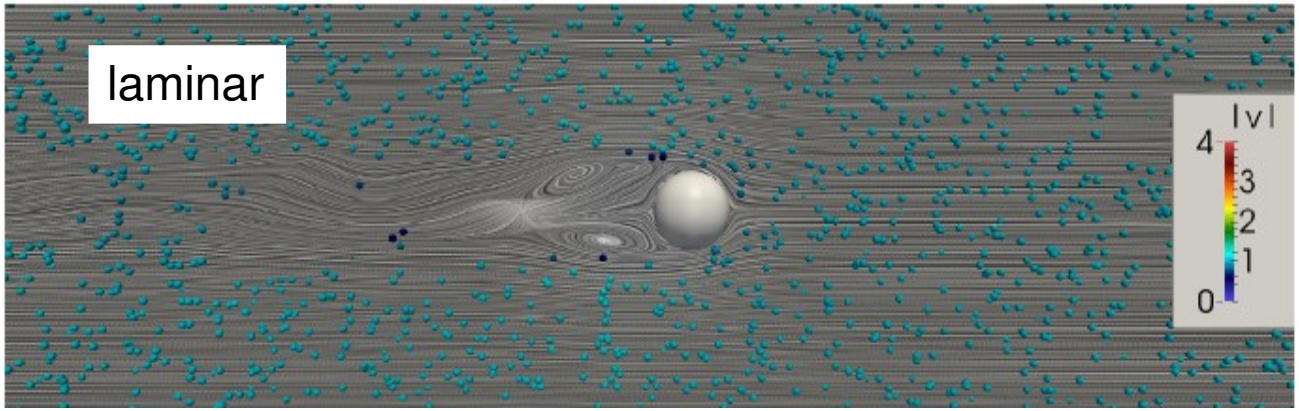
Sekiya and Takeda (2003)

Collision rates of small particles with big body

Motivation – Physical problem

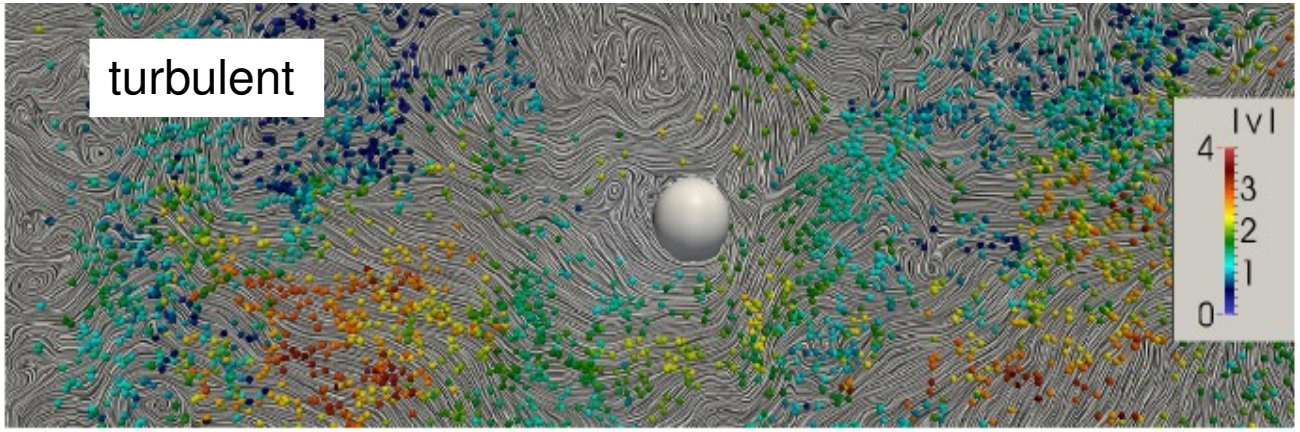
Interaction of big body with stream of small inertial particles

- Small particle collision rates
- Wake interactions

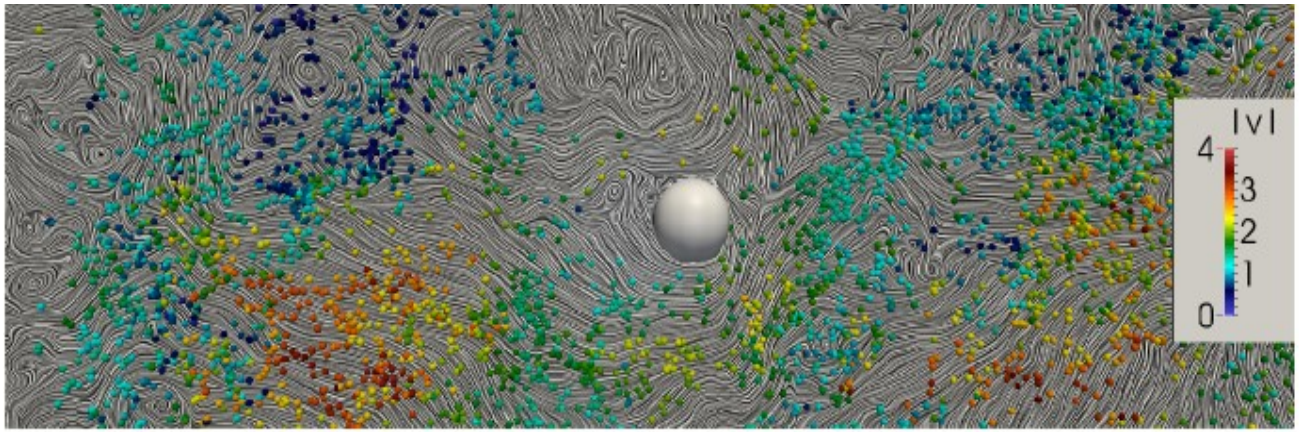


Influence of turbulent velocity fluctuations

- Drag force
- Wake modification
- Collision rates



Numerical challenges

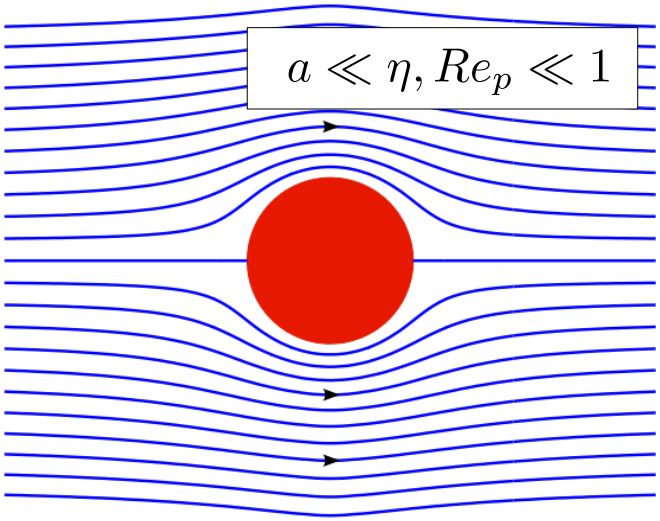


What does we need?

- Equations and solver for fluid motion
  No-slip boundary condition on big particle
  High Reynolds numbers

- Equations for small particle motion
  Integration of many trajectories for statistics

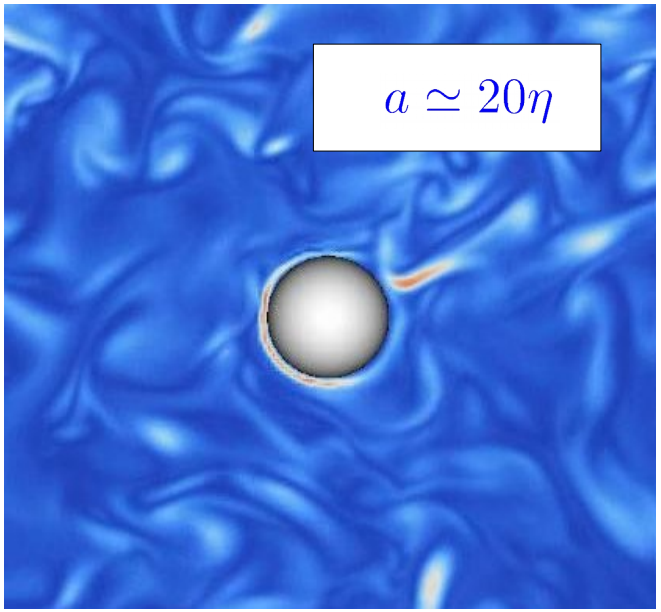
Small vs large particles



Analytically known drag force and flow structure

Stokes drag $\dfrac{d\boldsymbol{v}}{dt} = \dfrac{3\nu}{a^2}(\boldsymbol{u} - \boldsymbol{v})$

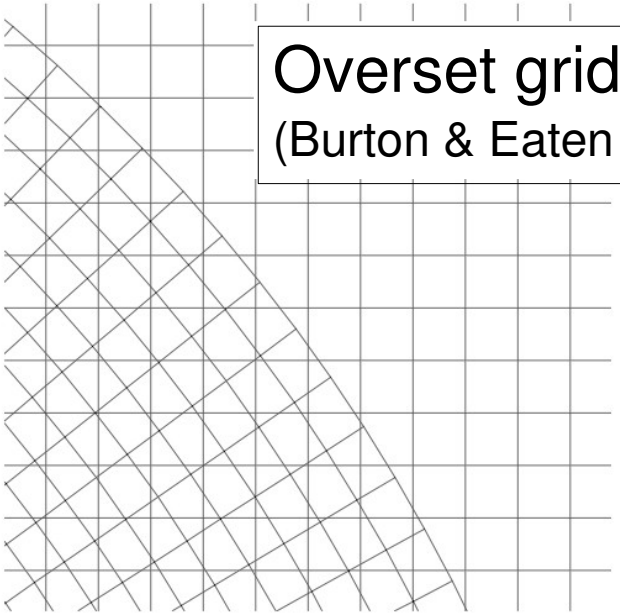
$\boldsymbol{u}$ known → ODE for small particle trajectories



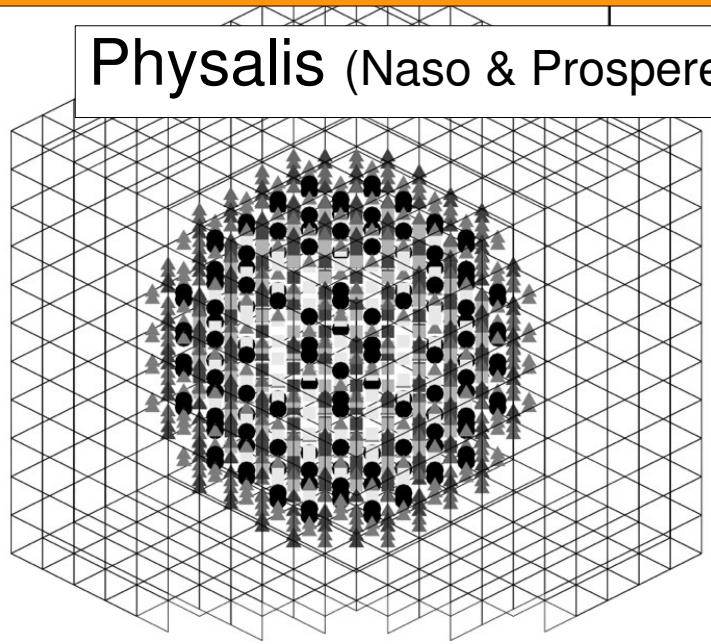
What if particle the particle is bigger?

→ handle the boundary layer!

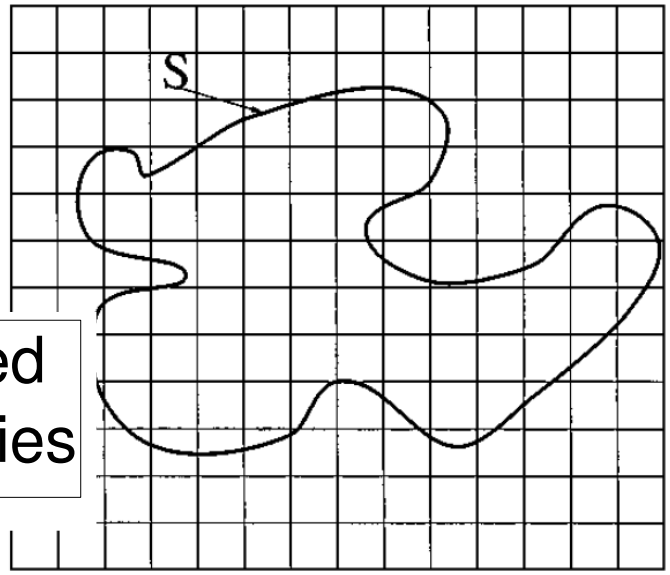
The numerical problem



Overset grid
(Burton & Eaten 2005)



Physalis (Naso & Prosperetti 2010)



Immersed boundaries

FIG. 1. Sketch of the immersed boundary.

- Solving dynamics on complete grid
- Imposing boundary conditions on S

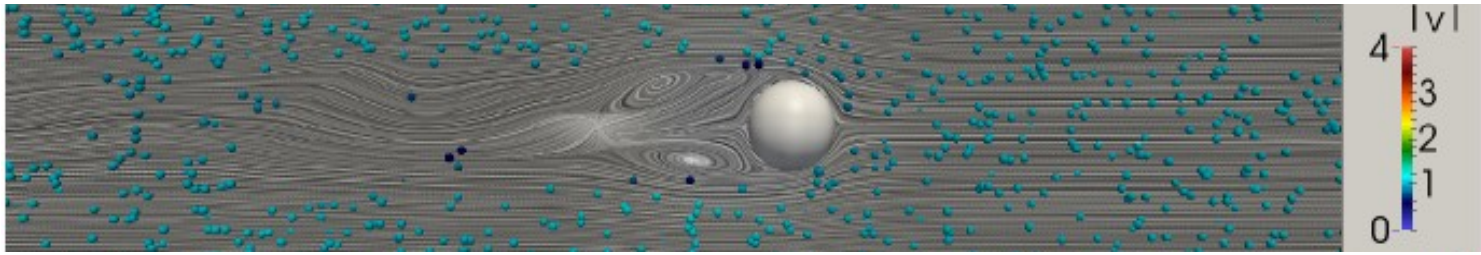
Advantages :

- Arbitrary shapes of S
- Easy to implement
- Compatible with Fourier-spectral codes

Shortcomings :

- Accuracy (order, boundary layer)

Numerical method



- Fluid: Navier-Stokes equations by pseudo-spectral method

$$\partial_t \boldsymbol{u} + \boldsymbol{u}\cdot\nabla\boldsymbol{u} = -\nabla p + \boldsymbol{f} + \nu\nabla^2\boldsymbol{u}, \quad \nabla\cdot\boldsymbol{u} = 0 \qquad \langle\boldsymbol{u}\rangle = U\boldsymbol{e}_x$$

$\boldsymbol{u}$: velocity field, ν: kinematic viscosity, $\boldsymbol{f} = \boldsymbol{f_e} + \boldsymbol{f_b}$: forcing (external + boundary)

- Small particles : heavy particles with low Re → Stokes drag

$$\ddot{\boldsymbol{X}} = \frac{1}{\tau}\left[\boldsymbol{u}(\boldsymbol{X},t) - \dot{\boldsymbol{X}}\right]$$

$\tau = 2\rho_\mathrm{p} a^2/(9\rho_\mathrm{f}\nu)$ particle response time

$\mathrm{St} = \tau/(d/U)$

Effect of inertia based on particle crossing time

- No-slip boundary conditions: penalty method (direct forcing, Fadlun et al. 2000) :

$$\boldsymbol{V} = \boldsymbol{u}|_{sphere} = 0$$

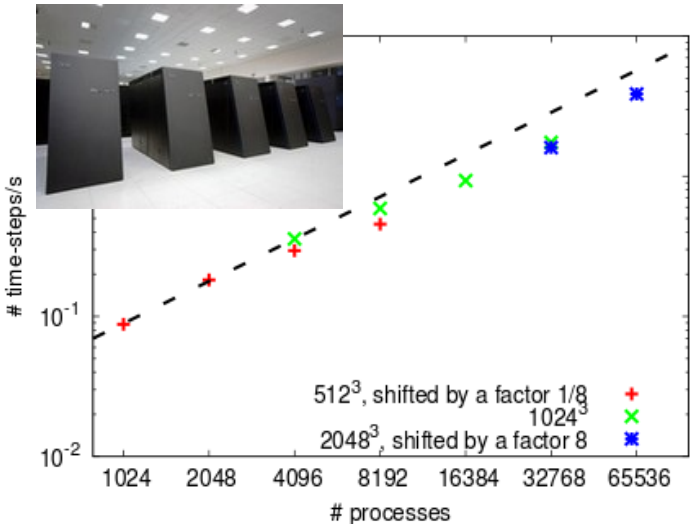
discretized Navier-Stokes equation : $\mathrm{u}^{n+1} - u^n = \Delta t(rhs^n + f_b^n)$

replacing $\mathrm{u}^{n+1} \to V^{n+1}$ yields force $\mathrm{f}_b^n = -rhs^n + (V^{n+1} - u^n)/\Delta t$

Numerical method – bounday conditions

Advantages:

- Standard Fourier-solver for incompressible turbulence
- Massive parallel codes
- Easy to implement



Difficulties:

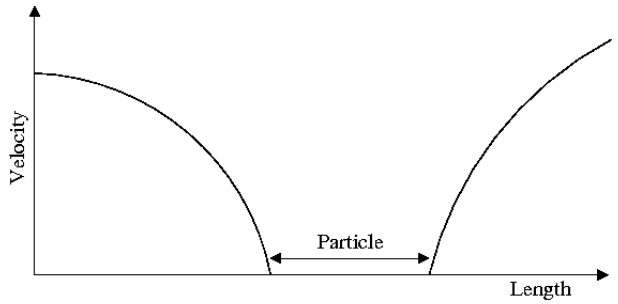
- Truncated Fourier representation and discontinuities
- No-slip condition and divergence-freeness at the boundaries

---

Discontinuities in the derivative of the velocity field

→ lost spectral accuracy (non-smooth)

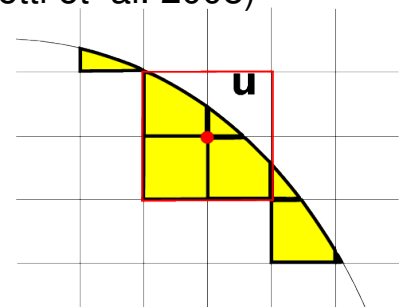
→ Gibbs oscillations at the discontinuities



Smoothed particle: volume-fraction scheme (Fadlun et al. 2000, Pasquetti et al. 2008)

- reduces Gibbs-oszillation
- allows for a moving particle
- in the spirit of a volumic forcing

$$\chi = \frac{1}{2h} \int_{(i-1)h}^{(i+1)h} \chi dx$$



Numerical method – no-slip & solenoidal

Main troubles :

- Operations boundary conditions (B) and divergence-freeness (P) do not commute !
- Fourier : poisson-equation for p with periodic boundary condtions

Solution : pressure predictor (Brown et al. 2001)

$$\tilde{u}^{n+1} = u^n + \Delta t\,[-(u^n \cdot \nabla)u^n - \nu \Delta u^n - \nabla p^n] \qquad \Delta p^n = \nabla \cdot [(u^n \cdot \nabla)u^n]$$

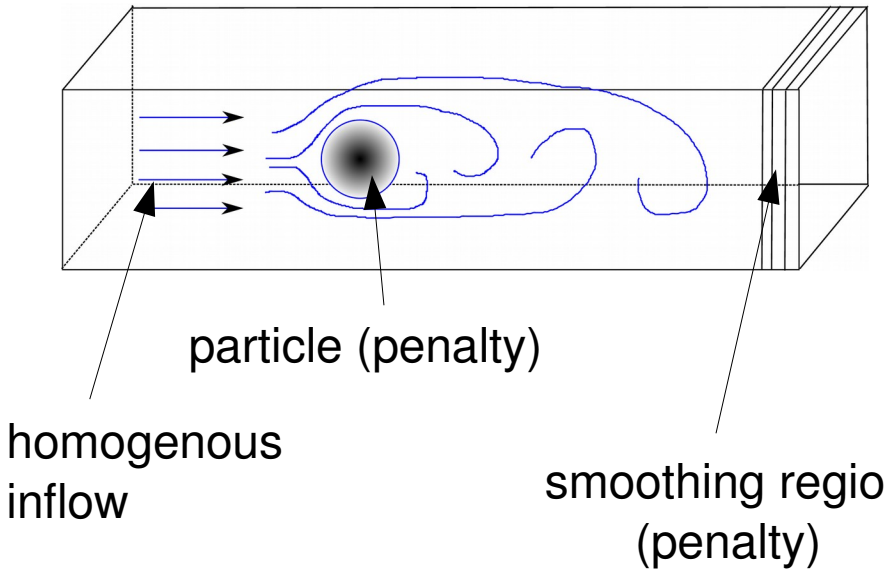
$$\tilde{\tilde{u}}^{n+1} = B(\tilde{u}^{n+1}) \equiv \tilde{u}^{n+1} + f_b^n$$

$$u^{n+1} = P(\tilde{\tilde{u}}^{n+1})$$

Usually no predictor ∇p – only projection

Benchmark-setup :

Numerical windtunnel



- regular grid with e.g. 256x256x1024 grid-points
- periodic boundary conditions at the surfaces
- penalty for particle and smoothing region

Numerical method – no-slip & solenoidal

Main troubles :

- Operations boundary conditions (B) and divergence-freeness (P) do not commute !
- Fourier : poisson-equation for p with periodic boundary condtions

Solution : pressure predictor (Brown et al. 2001)

$$\tilde{u}^{n+1} = u^n + \Delta t\,[-(u^n \cdot \nabla)u^n - \nu \Delta u^n - \nabla p^n] \qquad \Delta p^n = \nabla \cdot [(u^n \cdot \nabla)u^n]$$

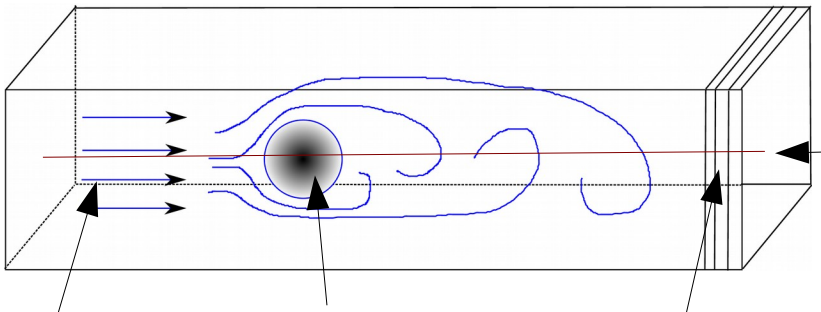
$$\tilde{\tilde{u}}^{n+1} = B(\tilde{u}^{n+1}) \equiv \tilde{u}^{n+1} + f_b^n$$

$$u^{n+1} = P(\tilde{\tilde{u}}^{n+1})$$

Usually no predictor ∇p – only projection

Benchmark-setup :

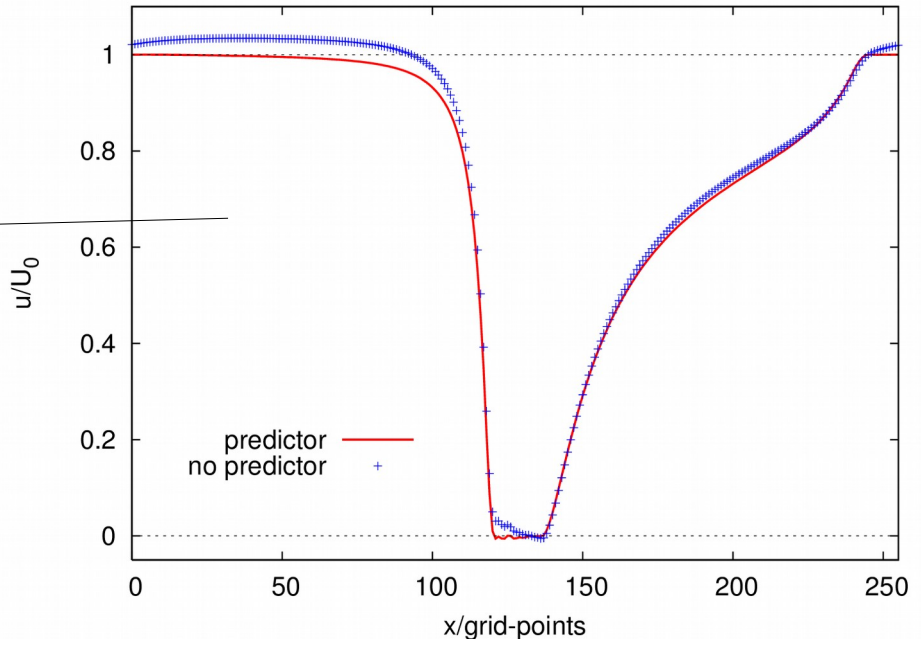
Numerical windtunnel



particle (penalty)

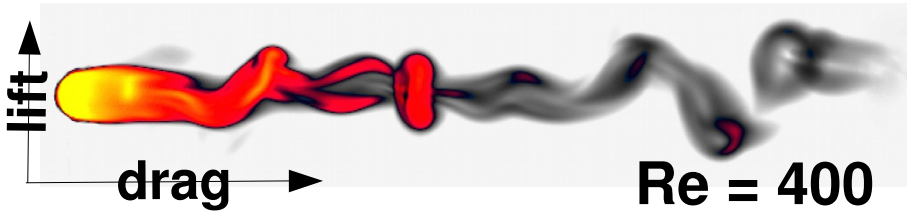
homogenous inflow

smoothing region (penalty)

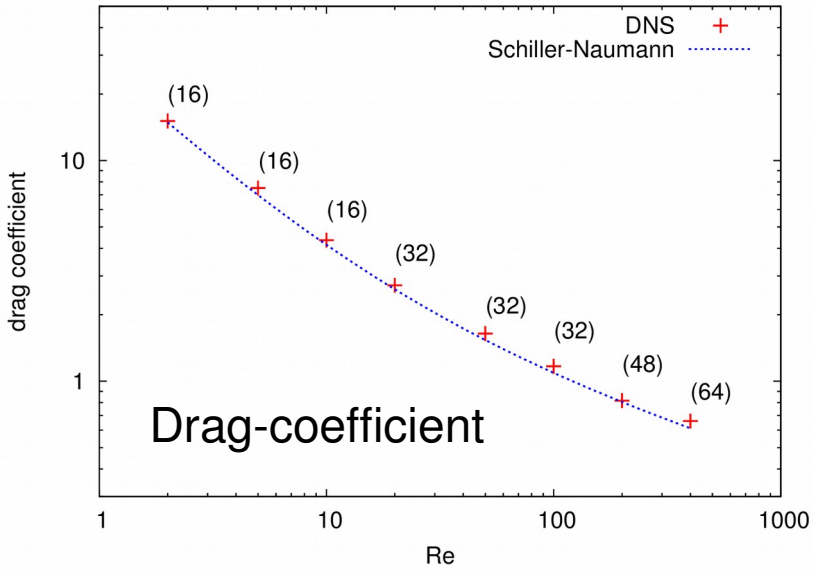


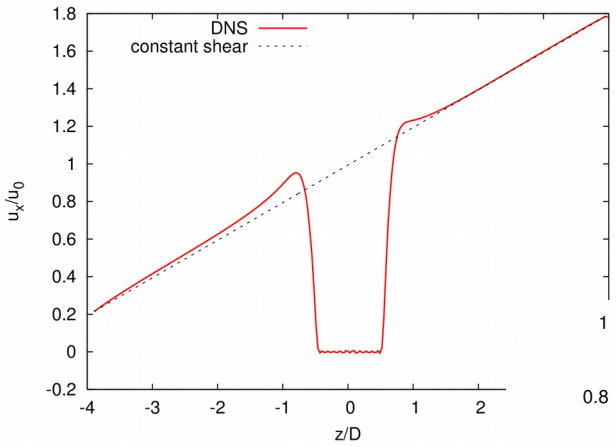
Numerical method - benchmarks

Forces on the particles



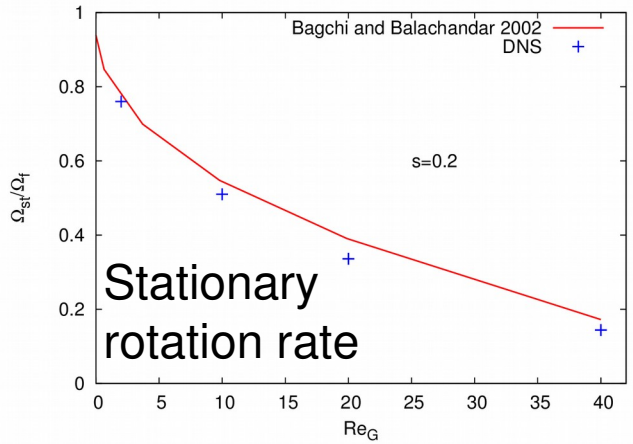
Drag-coefficient $C_D = F_D/(0.5\,U_\infty^2 \pi r^2)$

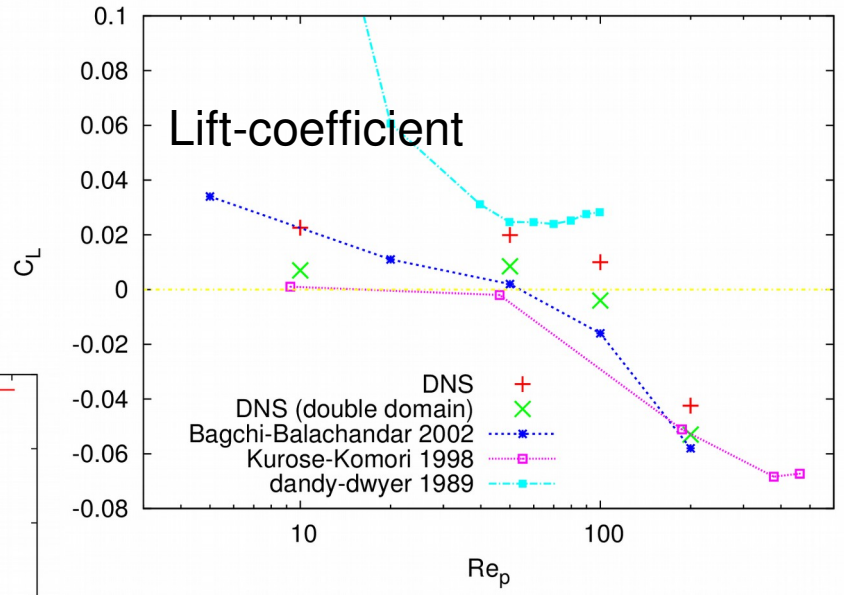




Constant shear:

$s = \nabla u\, D/u_\infty$





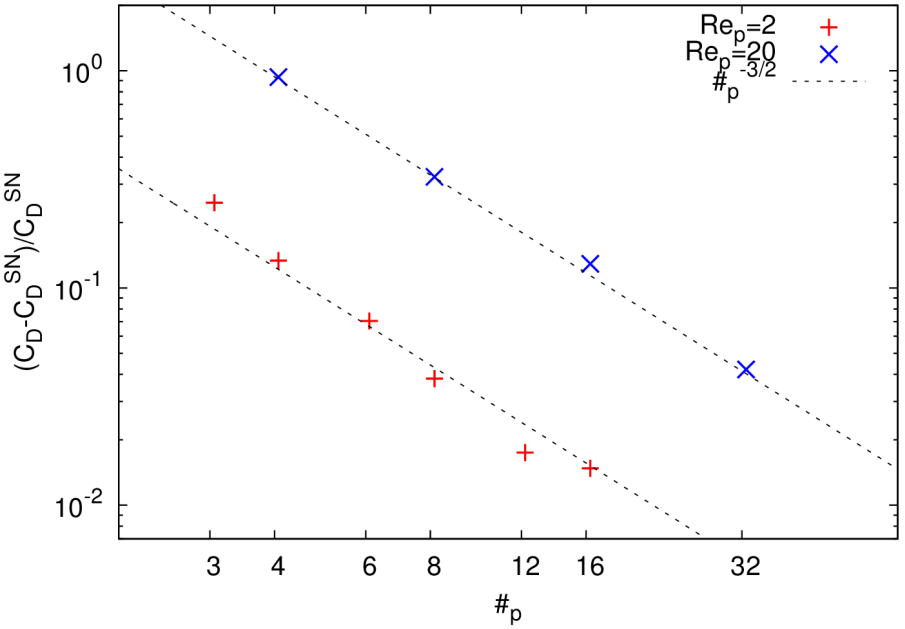
Numerical method - convergence

Convergence rate:

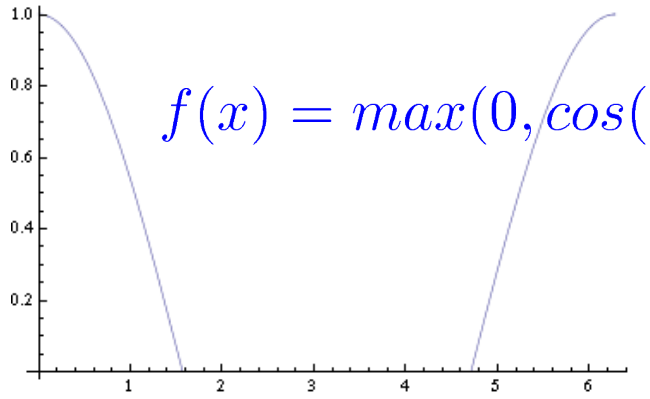
Boundary layer thickness $\delta_b \sim Re^{-1/2}$

Resolution : $D/\Delta_x \sim \delta_b/\Delta_x$

for $Re = const$

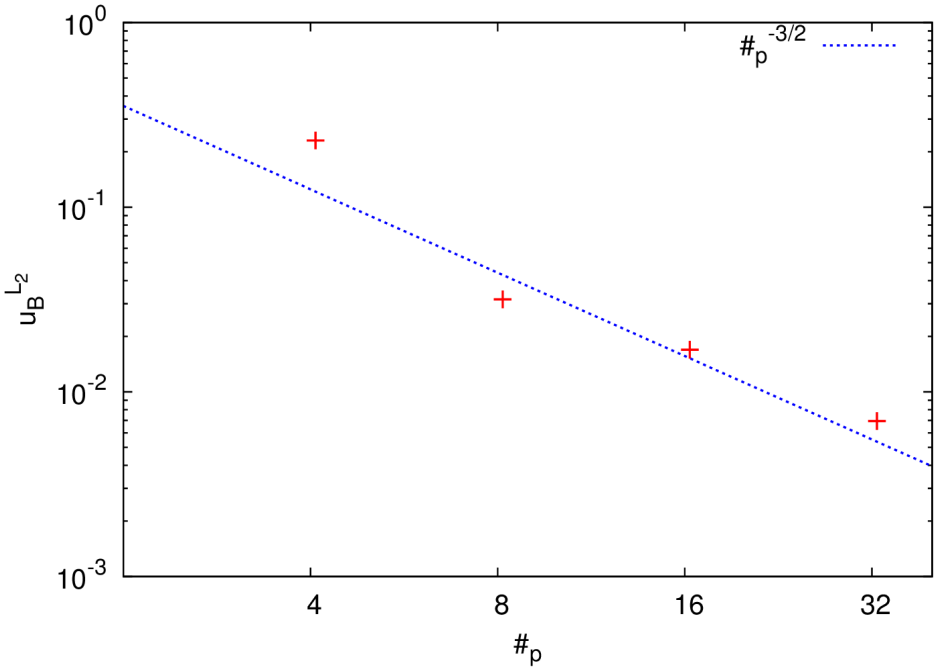


Why -3/2?

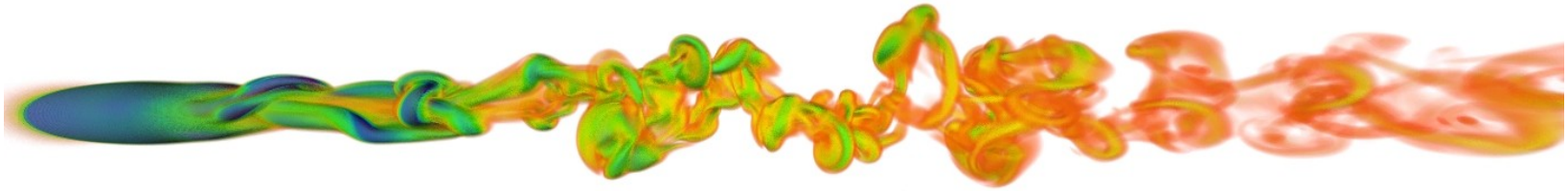


$$f(x) = max(0, cos(x))$$

L_2-Norm $\left(\int_B f_K(x)^2 dx\right)^{1/2} \sim K^{-3/2}$

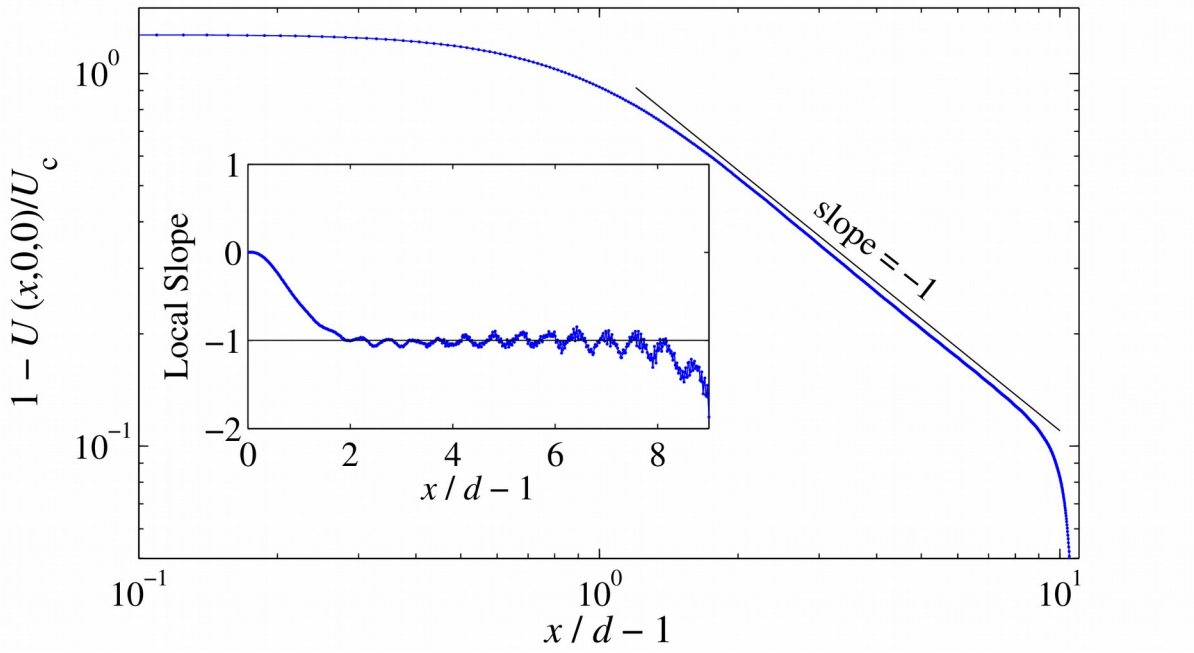


Benchmark - wake



Mean velocity deficit : $\Delta U = 1 - U(x,0,0)/U_c \sim x^{-1}$

(Wu & Faeth 1994, Bagchi & Balachandar 2004)



Results

1) Particle collision rates
2) Particle-wake interactions
3) Drag force

Small particle collision rates

Problem Parameters :

Particle Reynolds number

$$100 \leq Re_p = \frac{d\,U_c}{\nu} \leq 1000$$

Fluid Reynolds number

$$0 \leq Re = \frac{u_{rms}\,L}{\nu} \leq 3000$$

Turbulent intensity

$$0 \leq I = \frac{u_{rms}}{U_c} \leq 1.2$$

Dust Stokes number

$$0 \leq \tau \leq 80$$

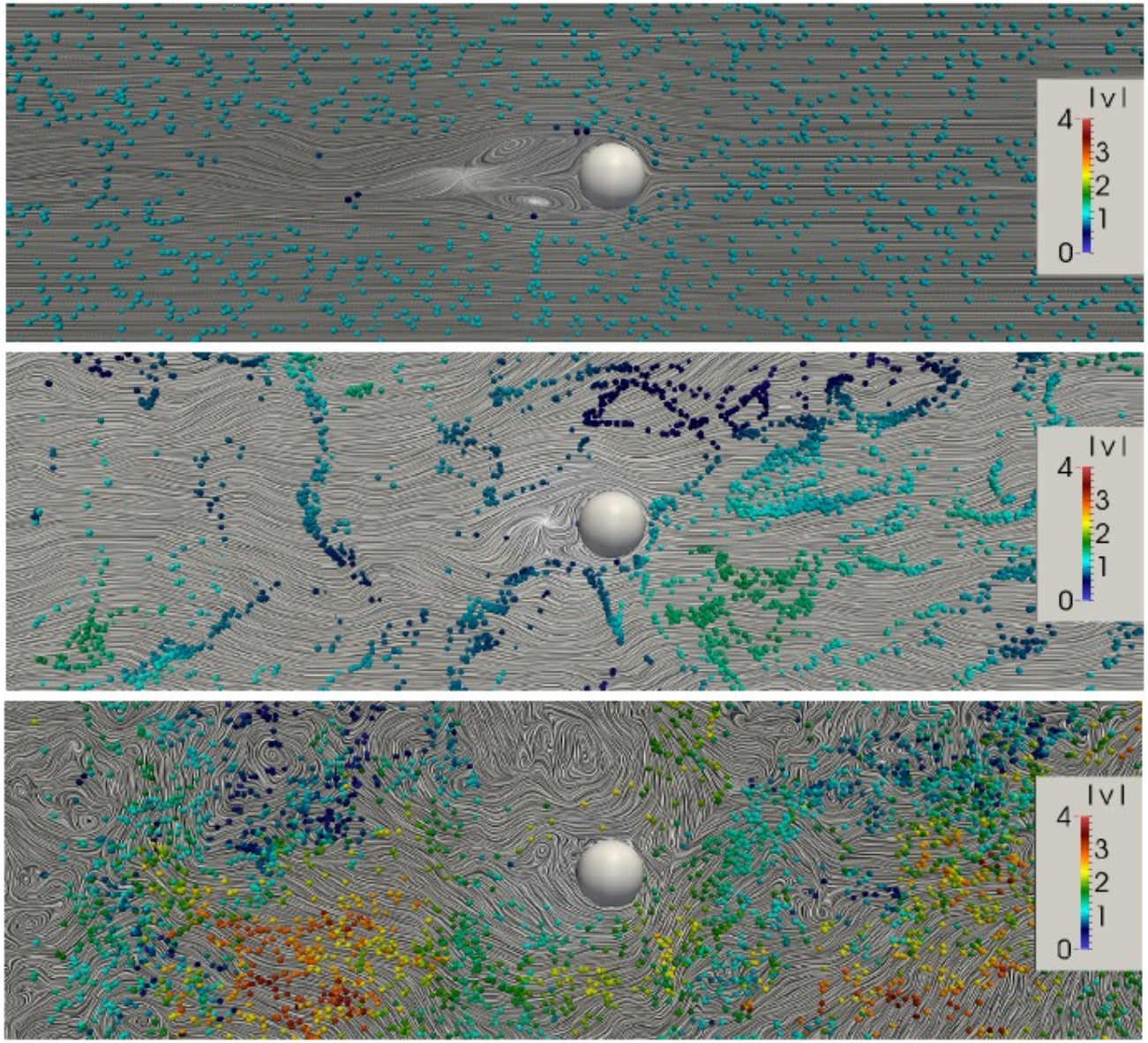
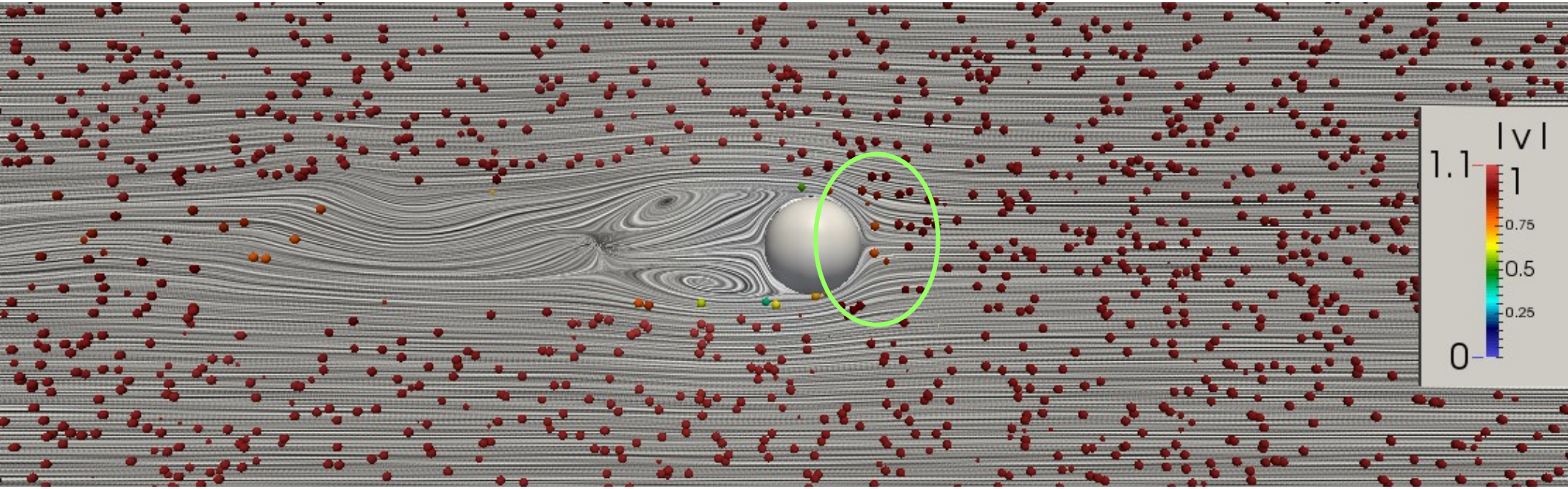


Fig. 1. Instantaneous streamlines of the gas flow and dust particles ($St = 0.8$) for I=0 (laminar), $I = 0.29$ (weakly turbulent), and $I = 1.18$ (strongly turbulent) from top to bottom. Note that while $St = 0.8$ for all shown cases, $St_\eta = 1.25$ for $I = 0.29$ and $St_\eta = 9.8$ for $I = 1.18$ so that the clustering properties of the shown particles change from one I to another.

Small particle collision rates

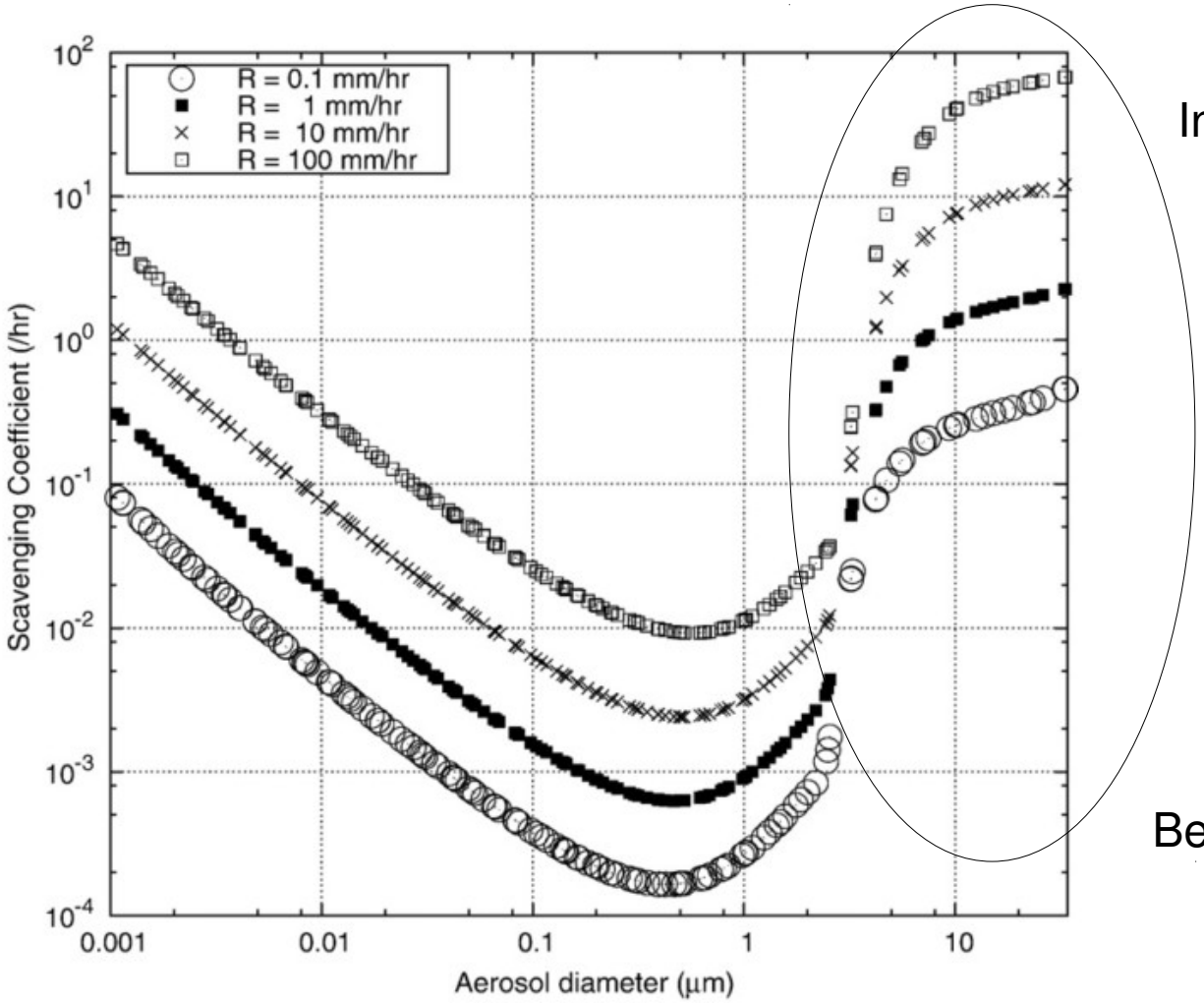
Laminar (uniform) inflow



Small particle collision rates

Application: Wet deposition of aerosols by falling raindrops

$$\gamma(d) = \int_0^\infty \pi/4\, U_t(D)\, E(d,D)\, n(D) dD$$

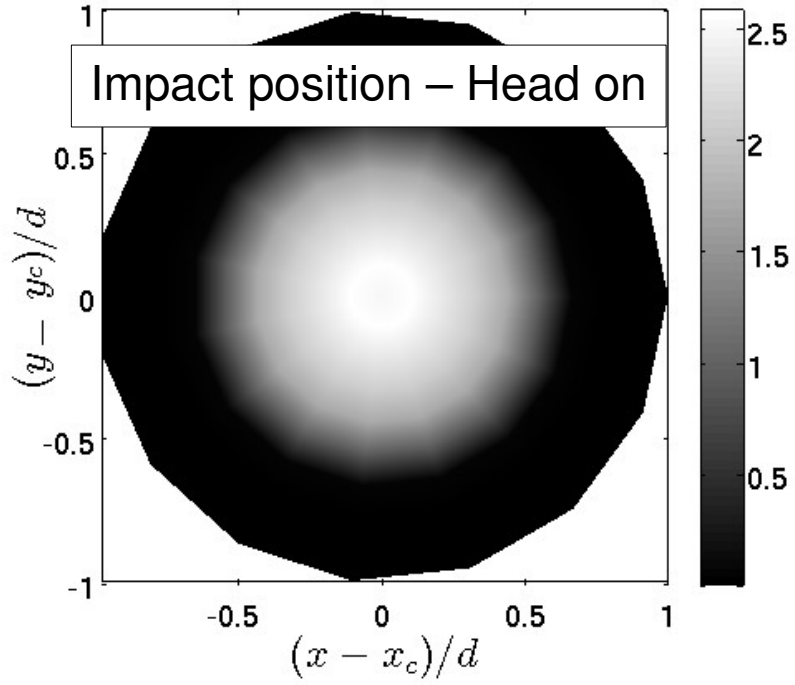


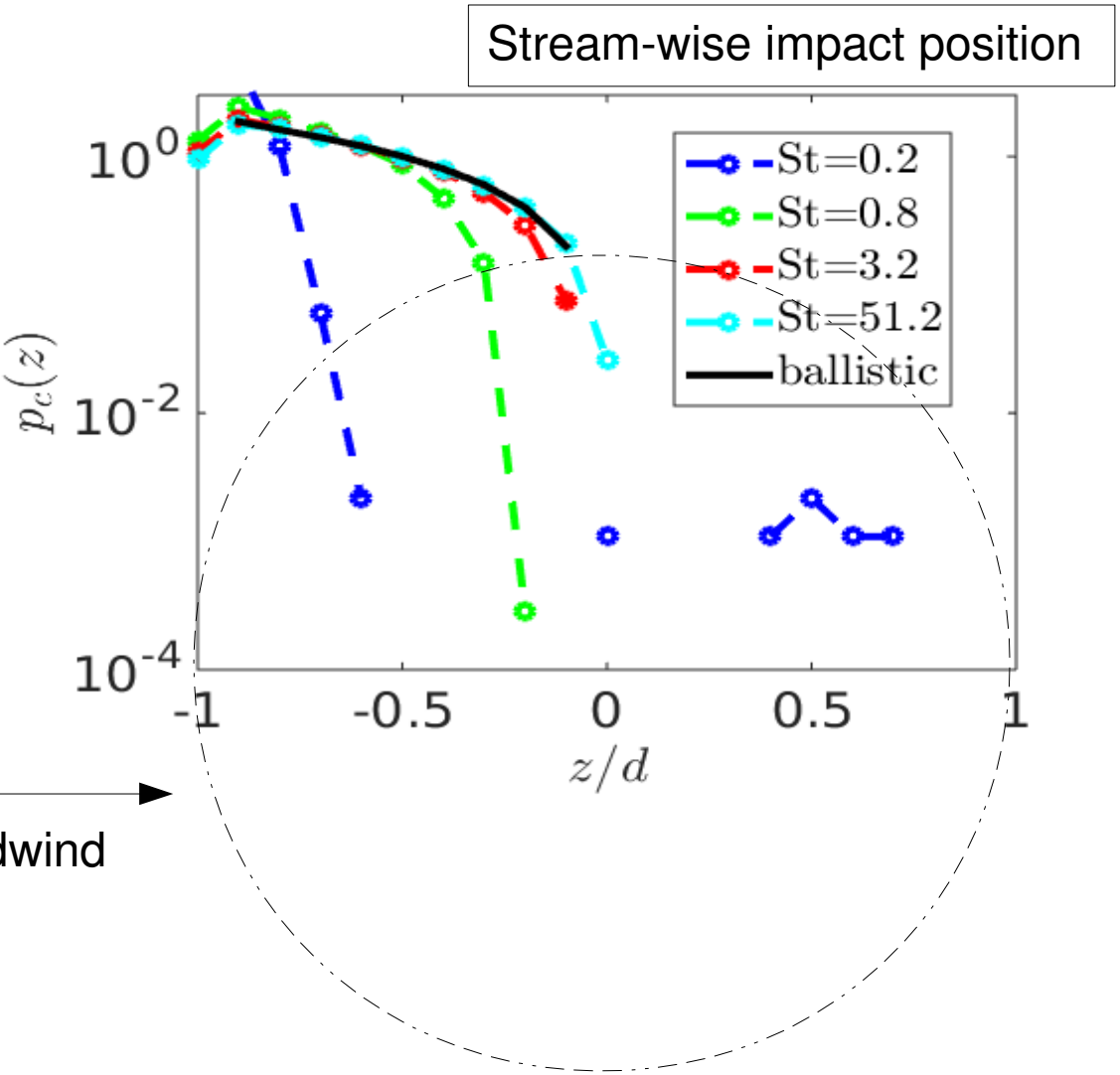
Inertial impaction

Berthet - Leriche et al. (2010)

LAMINAR collision statistics

Where do collisions happen ?

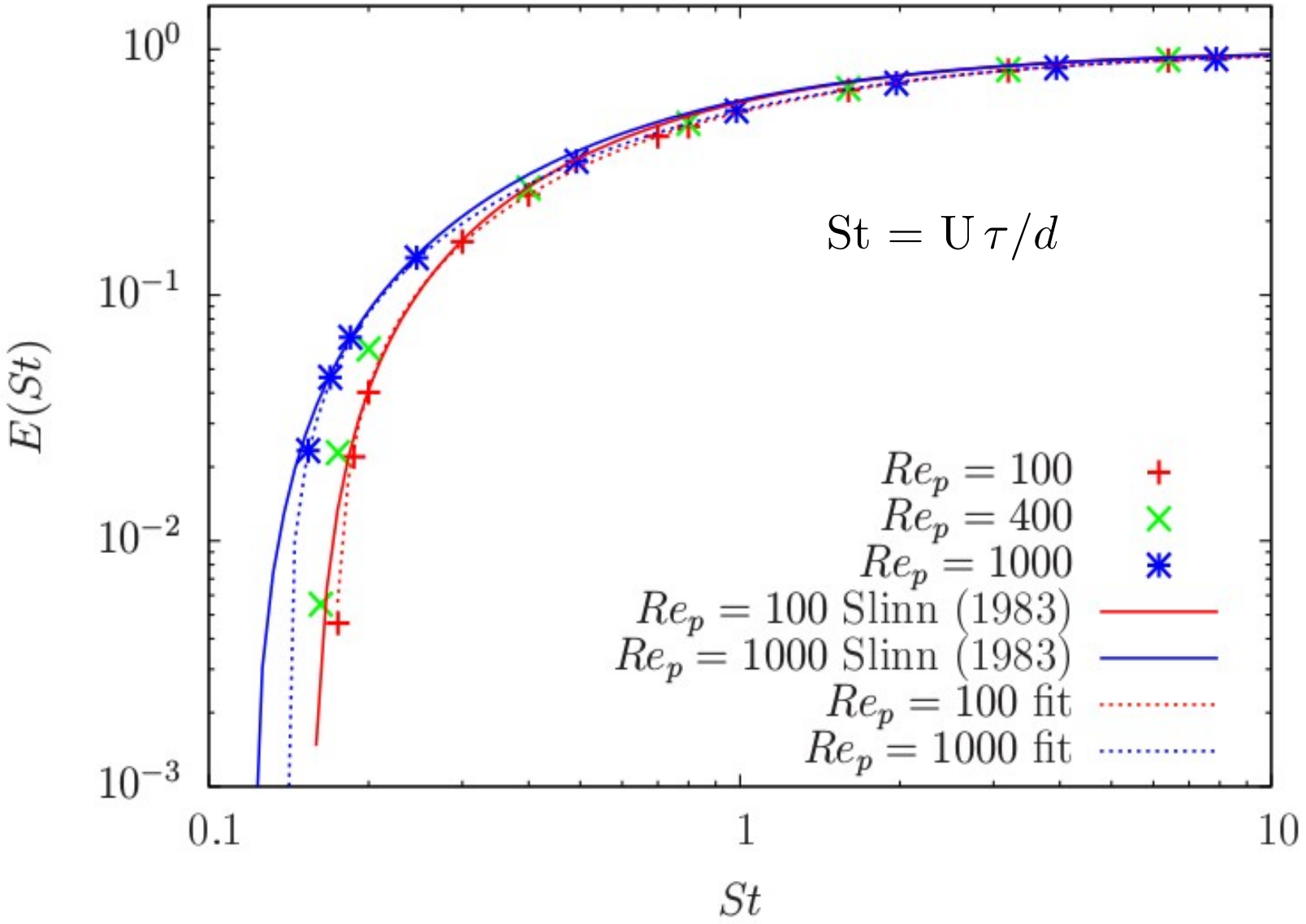




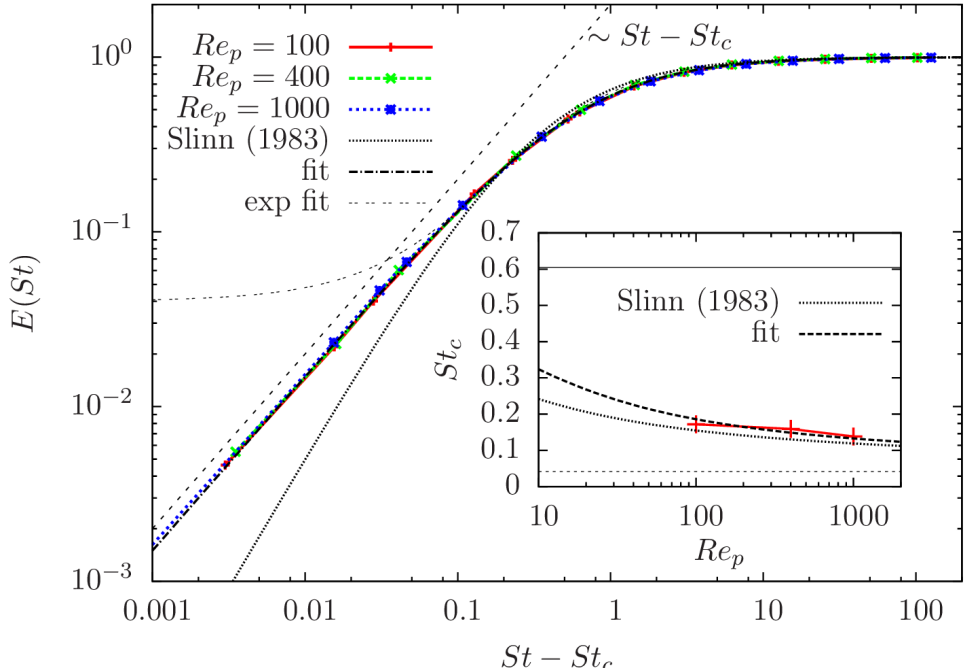
- Collisions are concentrated around the axis of symmetry
- Virtually no backward collision happen

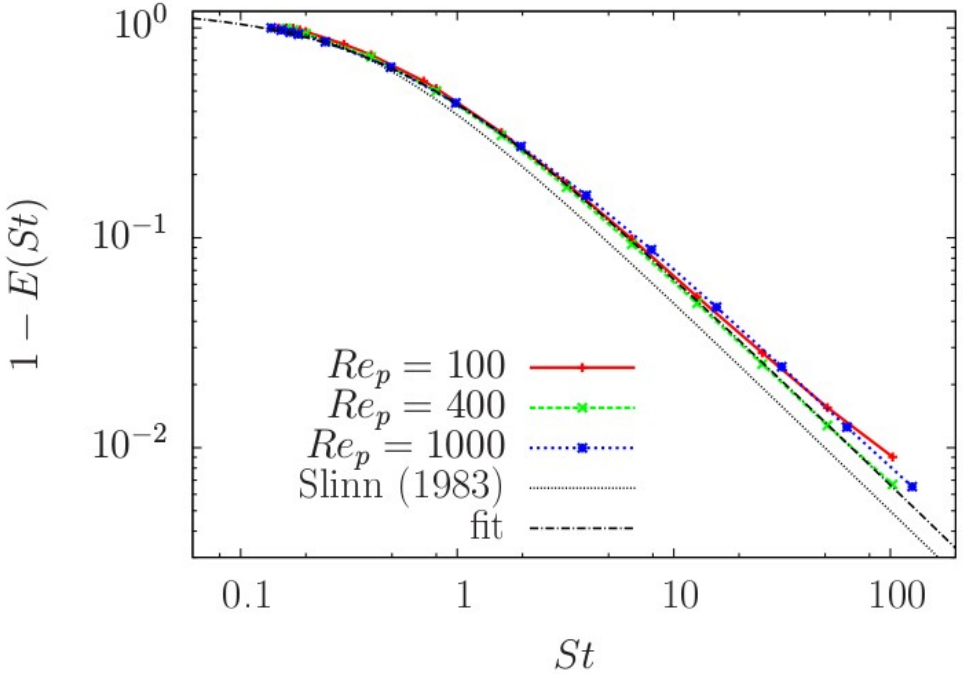
LAMINAR Collision statistics

Collision rate



LAMINAR Collision statistics





Small St limit :

- Critical Stokes number St_c
- Euler flow : $St_c = 1/24$
- Stokes flow : $St_c = 0.6$
- Re-collapse with linear slope

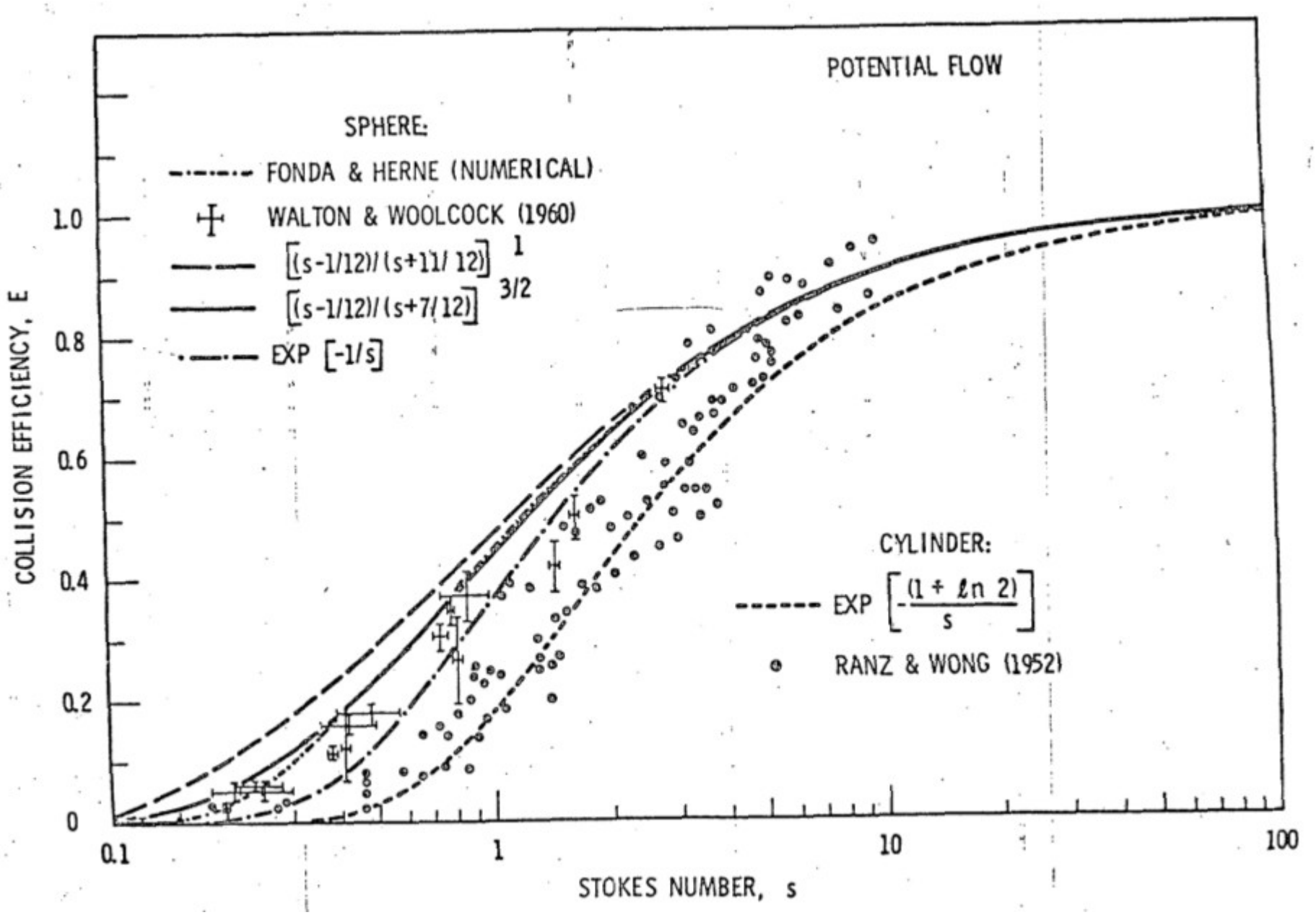
fit : $\frac{St-St_c}{St-St_c+2/3}$

Slinn : $\left(\frac{St-St_c}{St-St_c+2/3}\right)^{3/2}$

Large St limit :

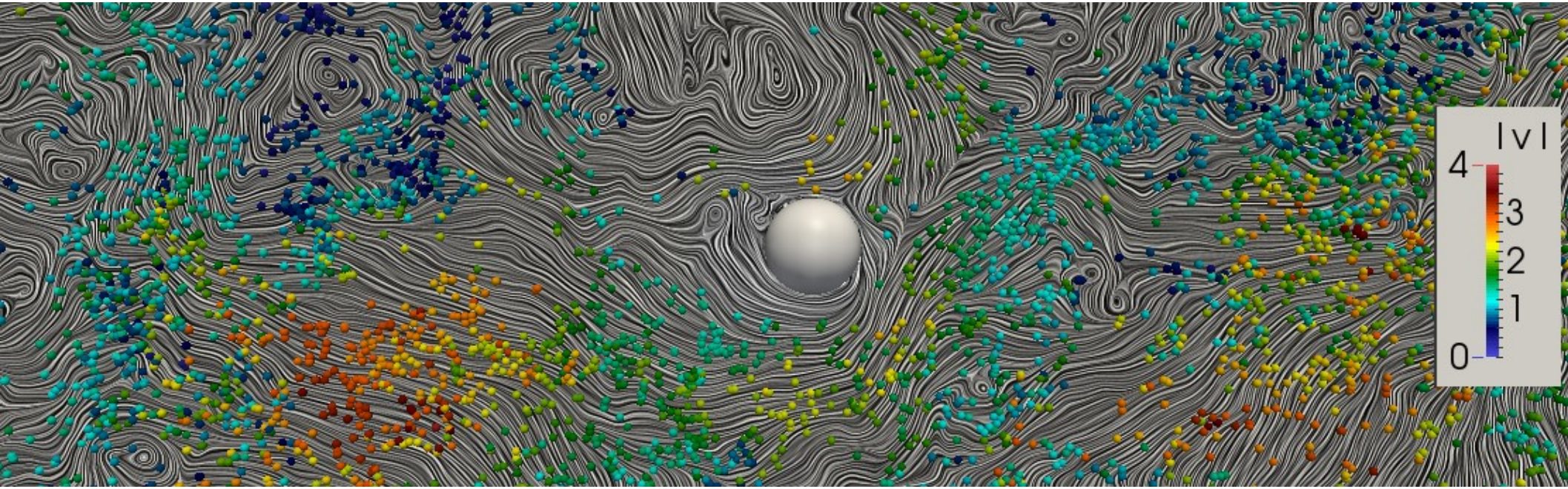
- Re-collapse with $\mathrm{E(St)} \sim 1 - 1/St$

Slinn's fit (1974)

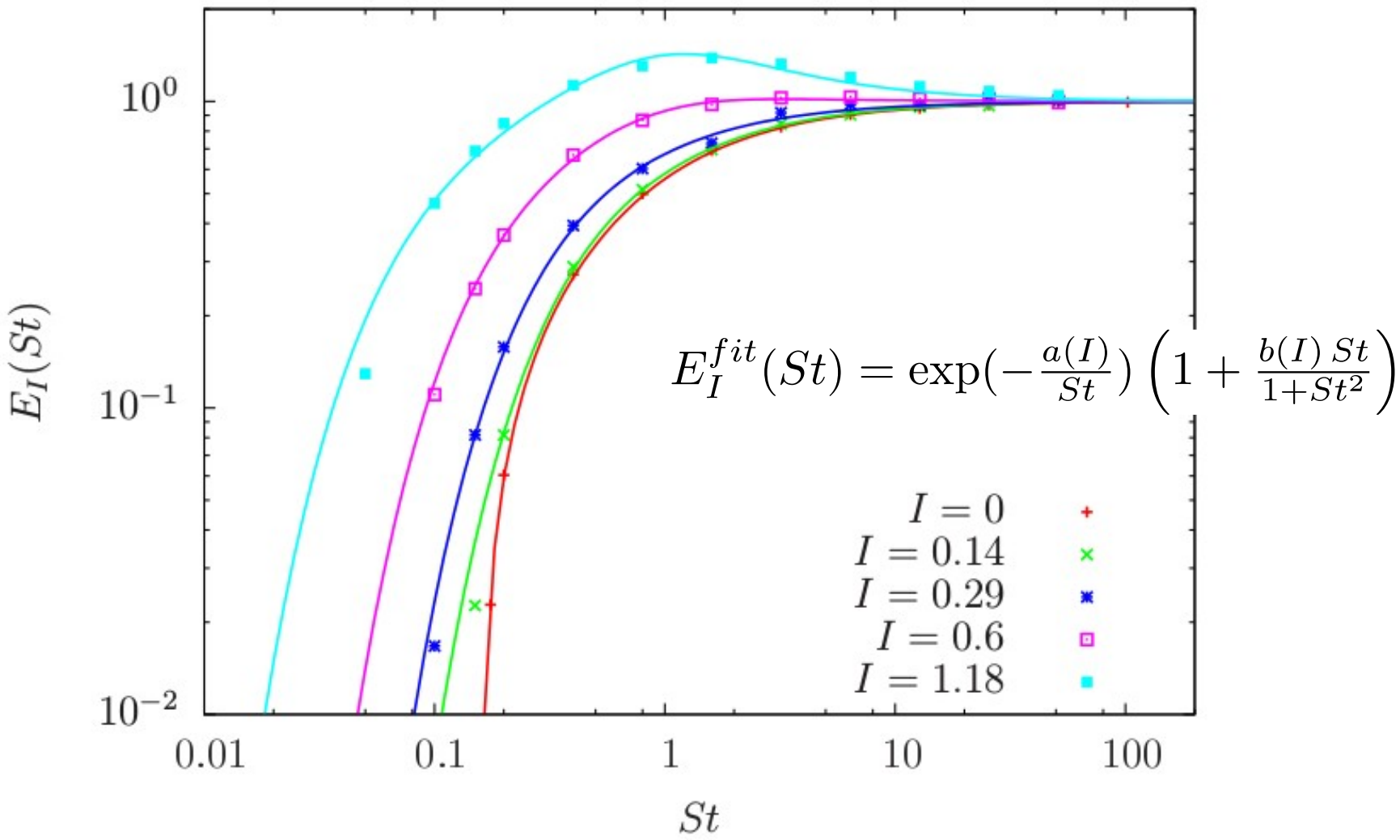


Small particle collision rates

Turbulent inflow



TURBULENT Collision Statistics



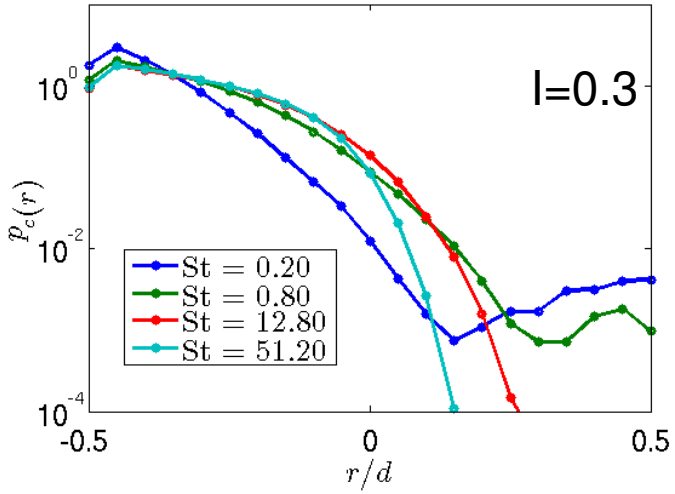
- Turbulence increases significantly the collision efficiency
- Collision efficiency can be larger than unity
- Probably no critical Stokes number

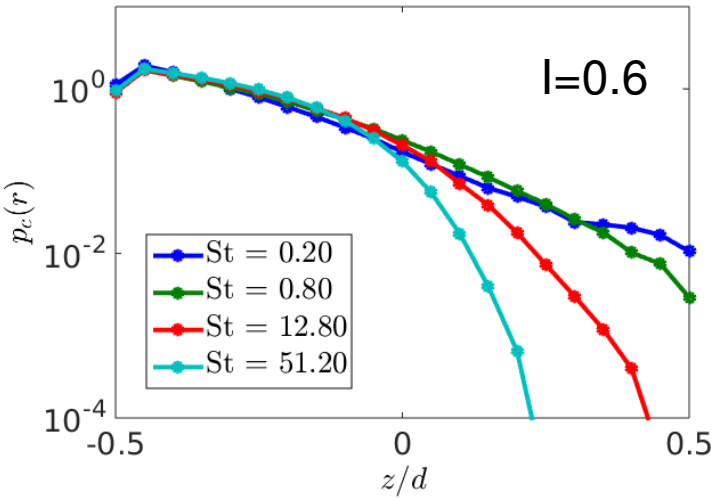
Turbulence effects on collision rate I

1) Fluctuating head wind

- Increase of collision rate (higher Re → higher collision rate)
- Velocity distribution → no critical Stokes number

2) Randomization of impact direction and position (→ backward collisions)



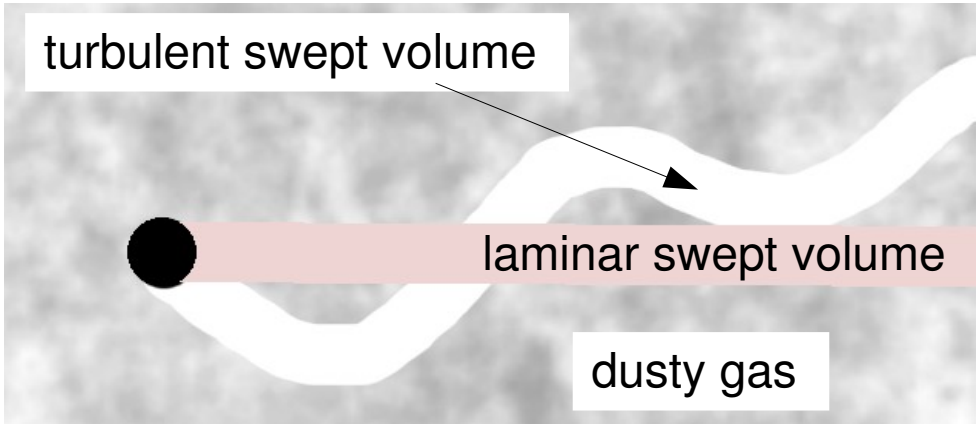


3) Increase of swept volume

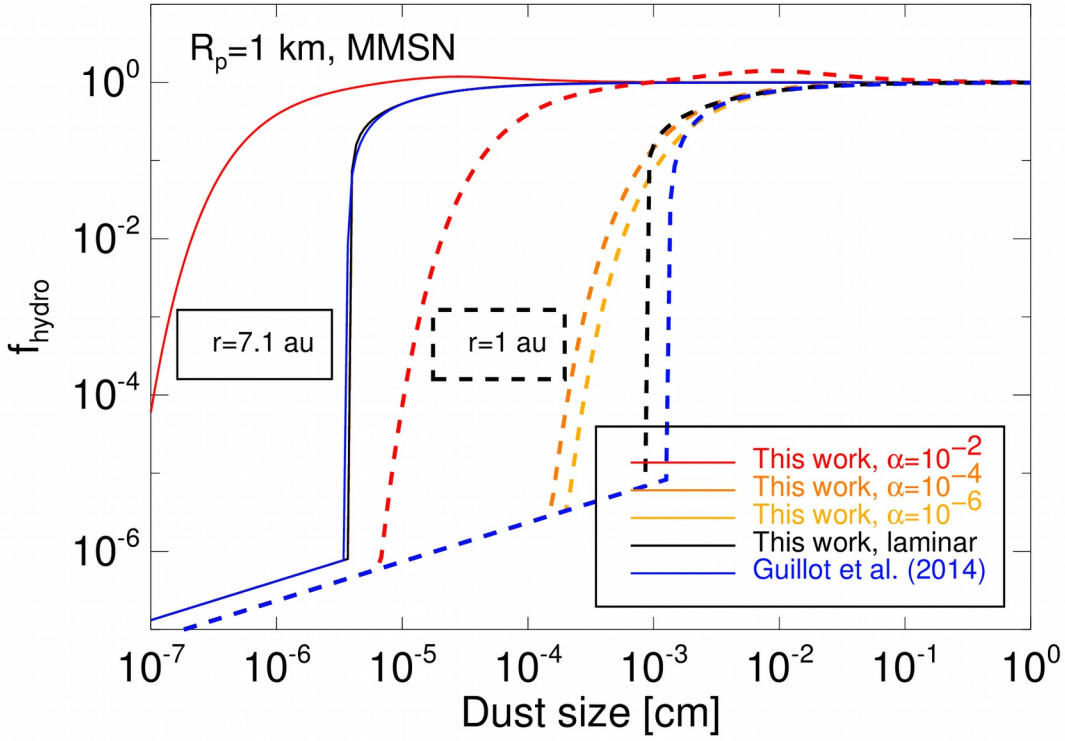
turbulent volume > laminar volume
simple computation for sinusoidal motion:

$$E_I(St) = E_0(St)(1 + \frac{I^2}{1+St^2})$$

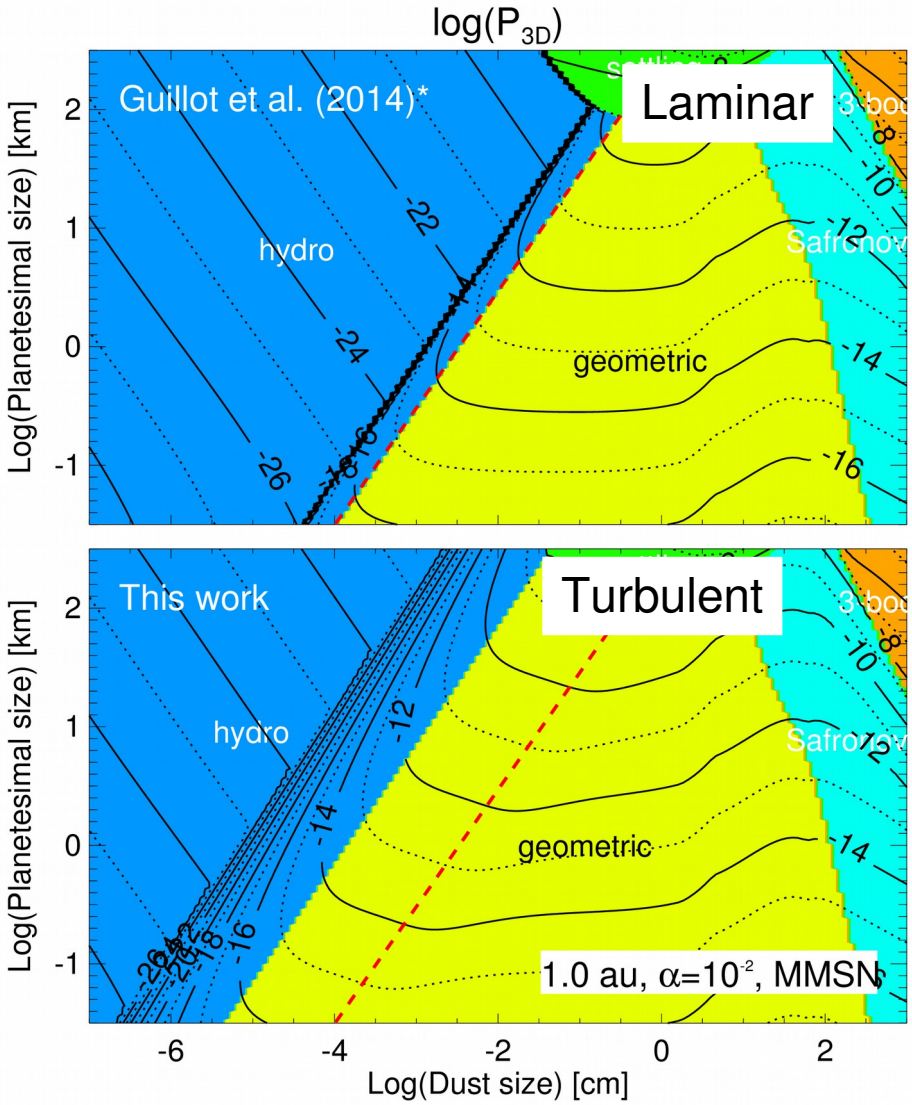
Allows for collision rate > 1!



Consequences for planet formation



Extension of the geometric regime to smaller dust sizes by turbulence



Collision efficiency for different orbital distances and turbulent intensities

alpha parameter: Turbulent eddy viscosity

$$\nu_t = \alpha\, c_s\, H$$

Importance of collision speeds

Different collision scenarios

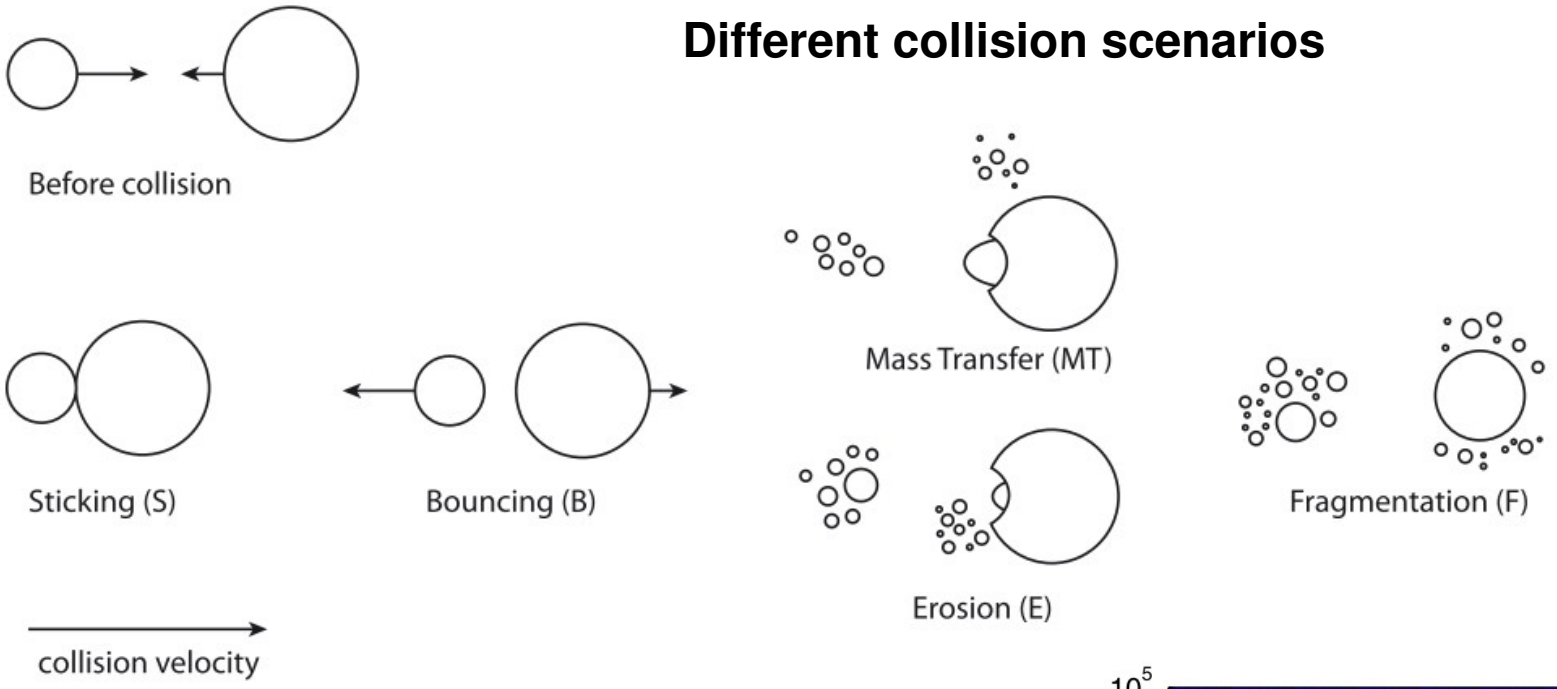
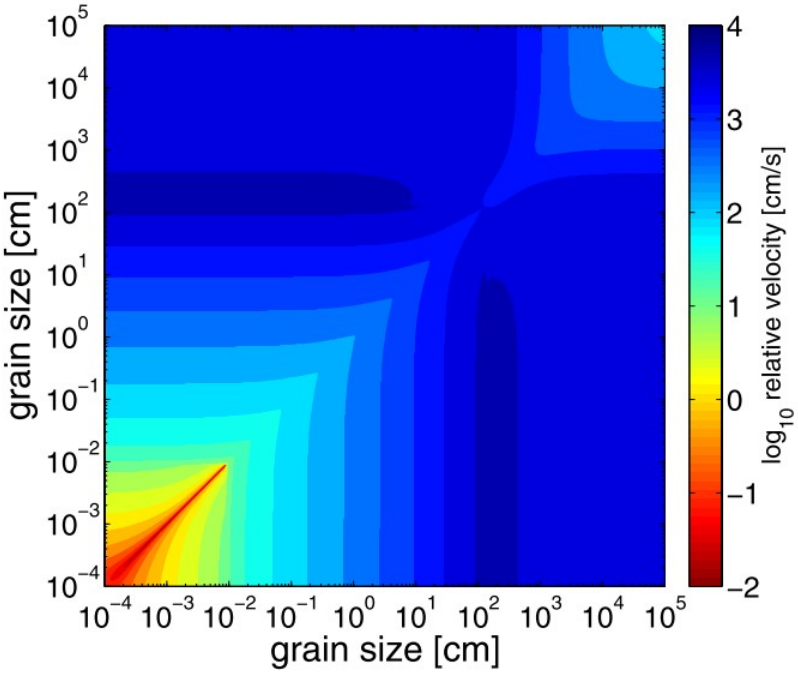


Fig. 2. Sketch of the five possible outcomes described in Sect. 3 sorted in rough order of collision
neously in a collision, and we define a mass transfer collision as leading to net growth for the tar
loss. This outcome is extremely dependent on the mass-ratio of the particles, thus adding a secon

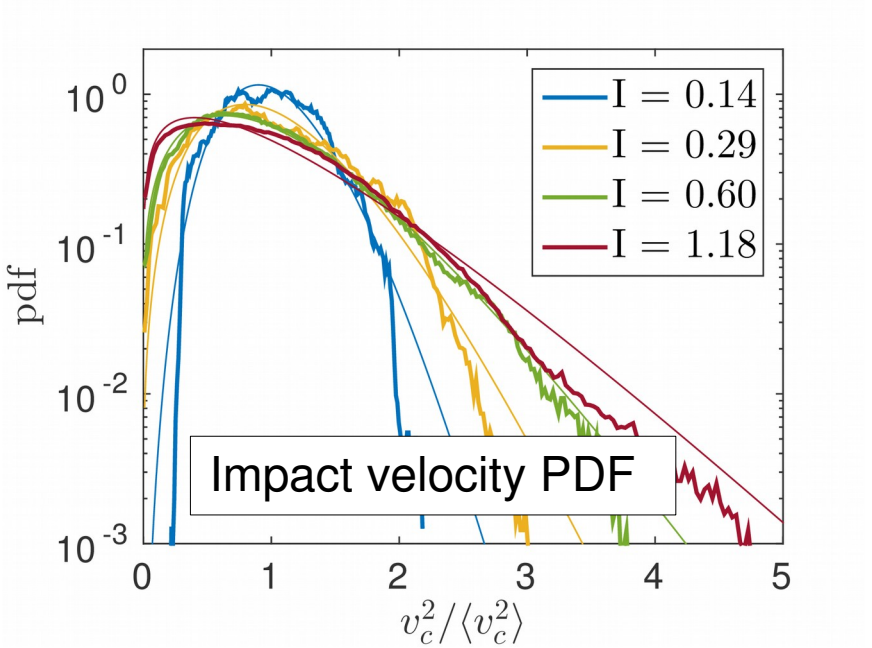
Population evolution models use velocity maps:

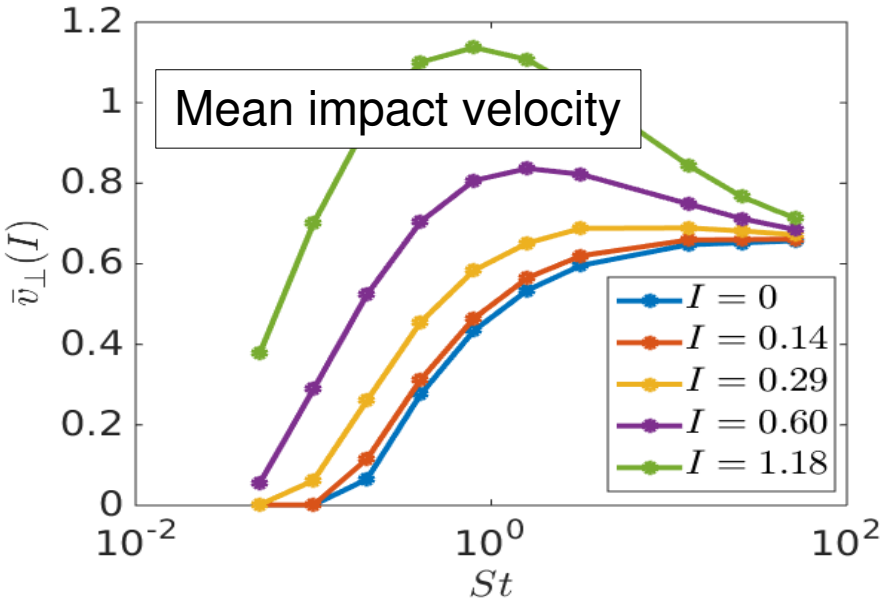
Windmark et al. (2012)

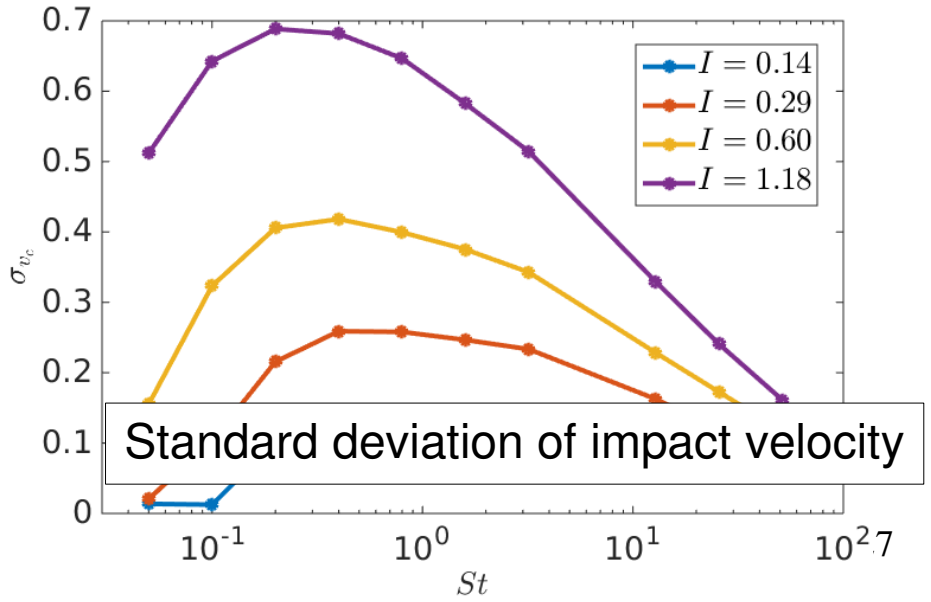


Measured collision speeds

- Turbulent fluctuations lead to velocity distribution
- Distribution fairly modeled by non-central chi-squared distribution
- Turbulence increases mean impact speed with maximum close to St=1
- Standard deviation approx. proportional to turbulence intensity

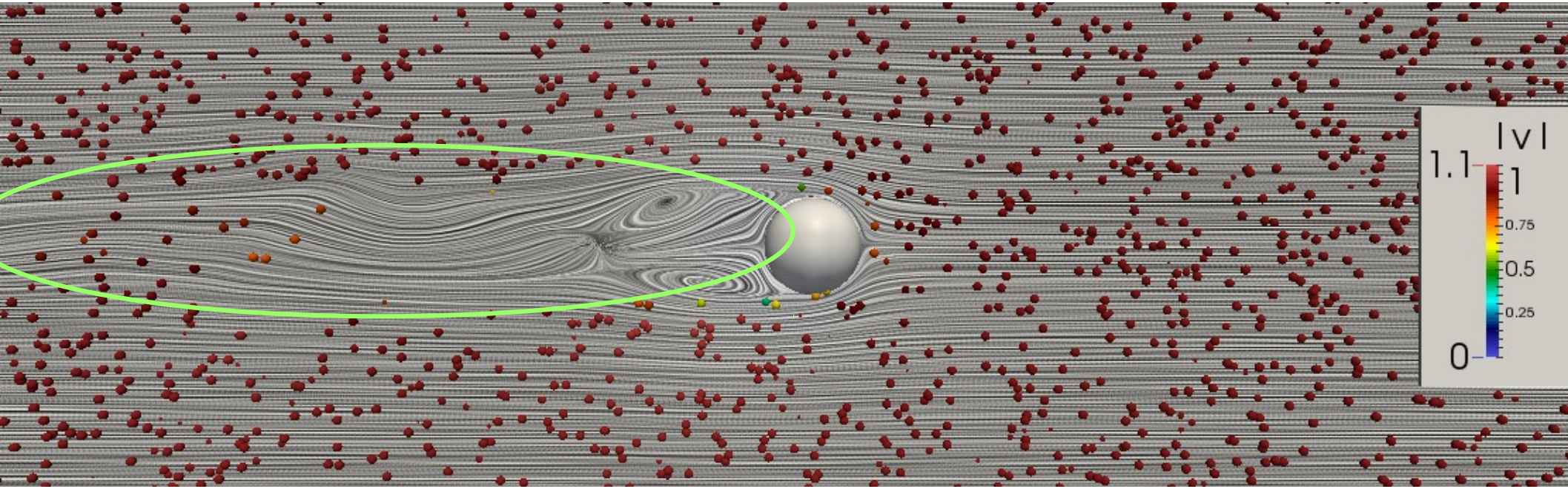






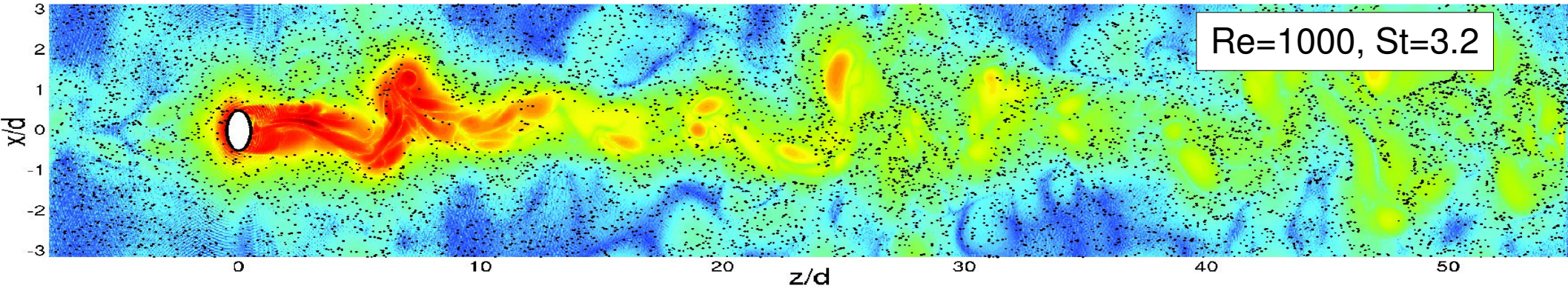
Small particle wake interactions

Laminar (uniform) inflow



Dust in the wake

What happens to particles that passed the sphere ?



Okubo-Weiss parameter

$$\Delta = \left(\frac{\det[\hat{\sigma}]}{2}\right)^2 - \left(\frac{\mathrm{tr}[\hat{\sigma}^2]}{6}\right)^3$$

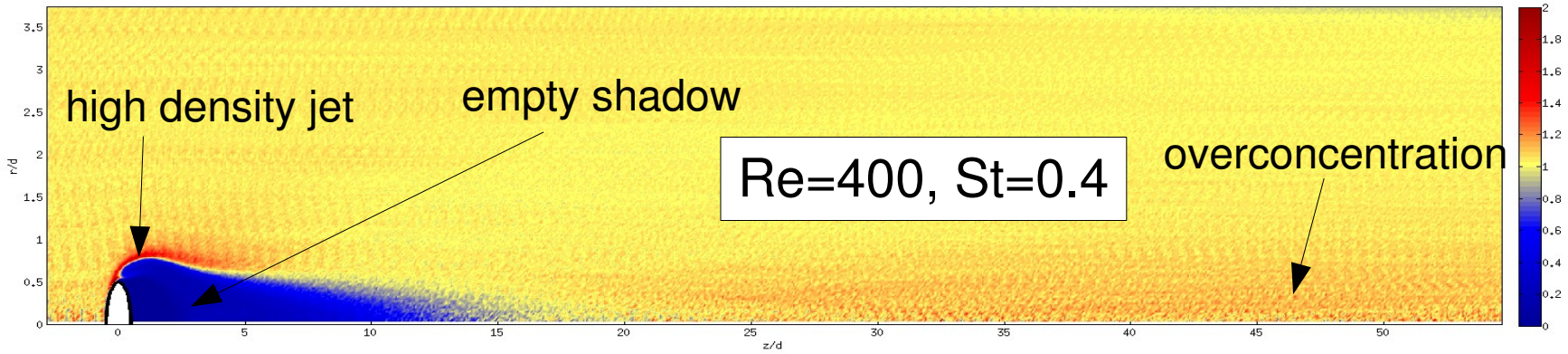
$\Delta < 0$ straining regions

$\Delta > 0$ rotational regions

$\hat{\sigma} = \partial_i u_j$ strain matrix

Particles interact with the shed structures – they are ejected

Mean particle densities



Experimental measurement (Jacober and Matteson 1990)

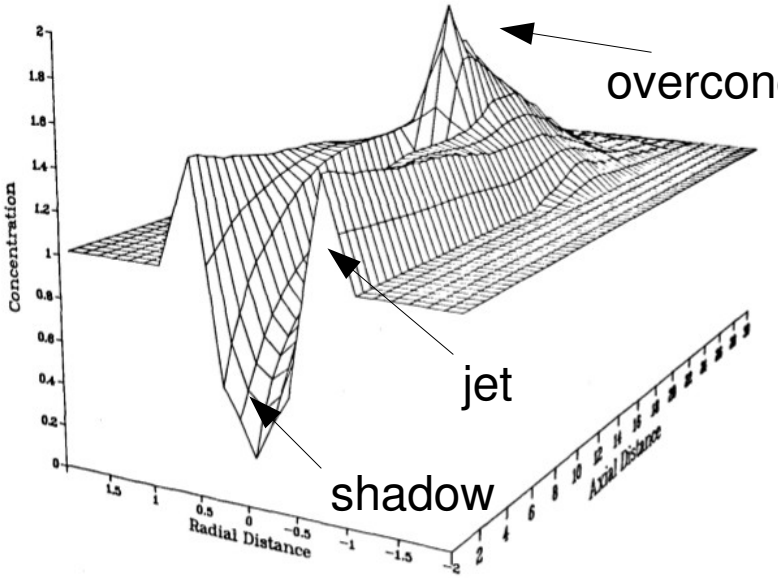
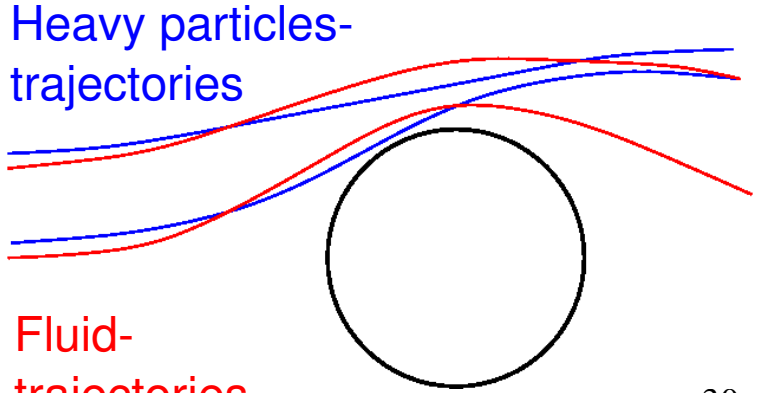
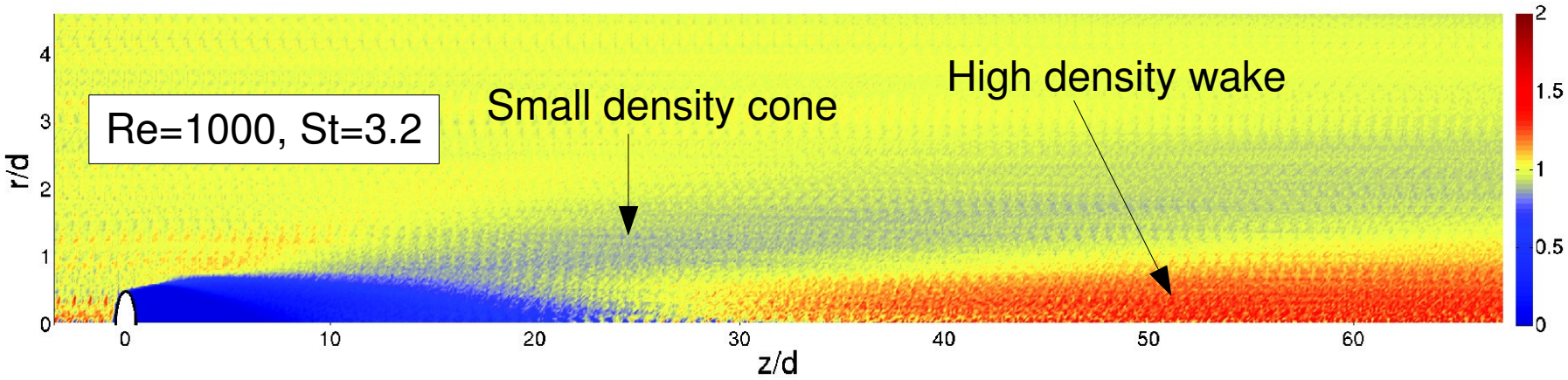


FIGURE 7. Axial and radial particle concentration distribution in the sphere wake for 10-μm particles and a free stream velocity of 6.1 m/s (radial and axial distances in radii).

Jet creation mechanism

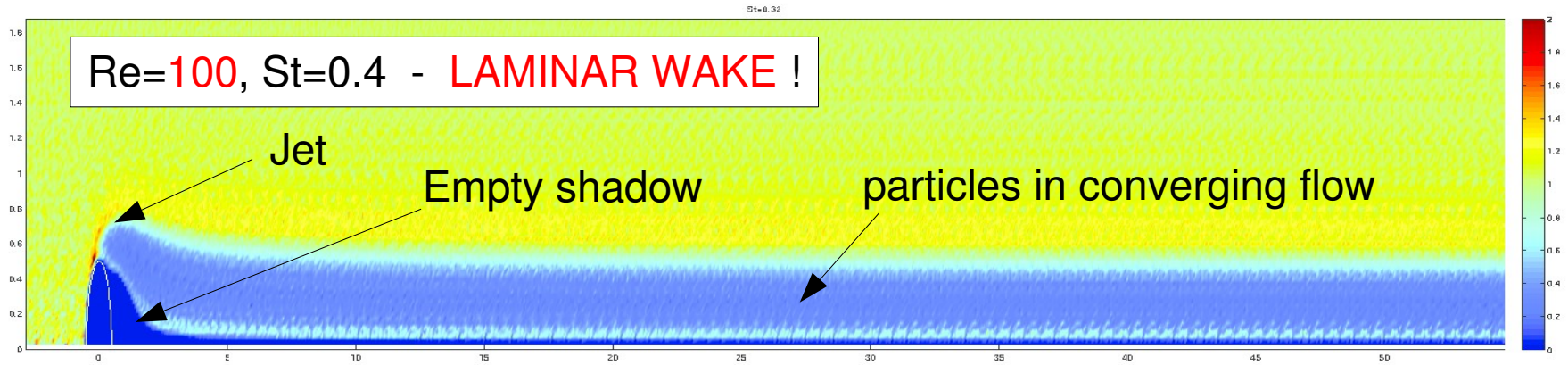


The wake : Mean particle densities



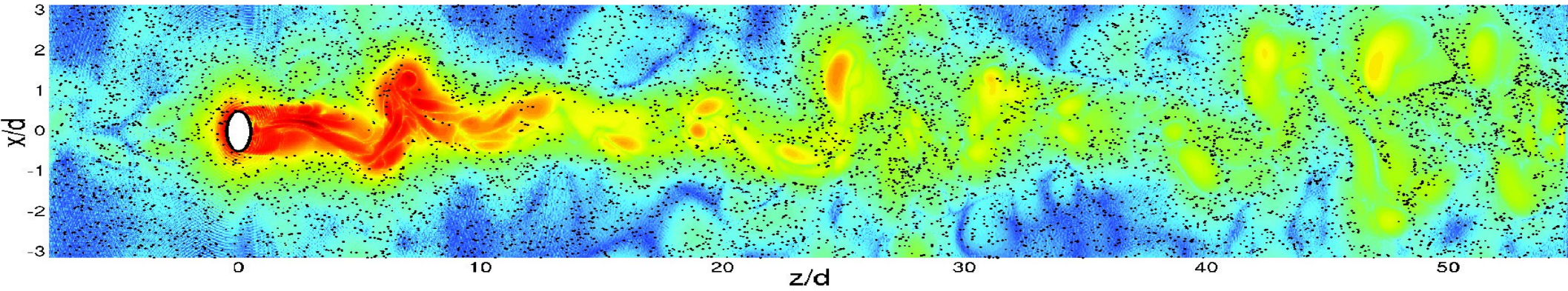
Jacober and Matteson : wake concentrations from collapsing jet

But: Concentrations only for turbulent wakes !

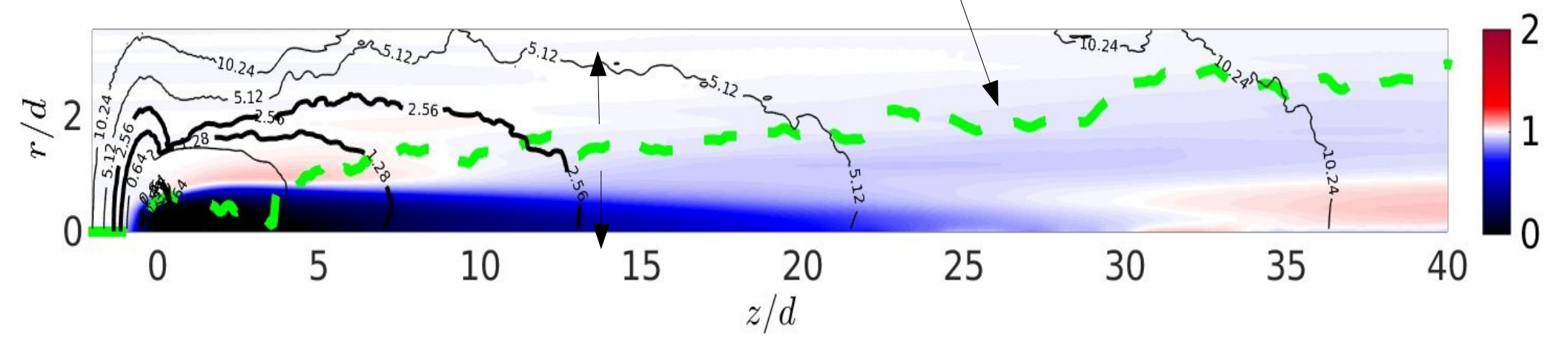


Concentration in the wake due to turbulent fluctuations

Wake concentration mecanism



Maximum of Δ depends on stream-wise position



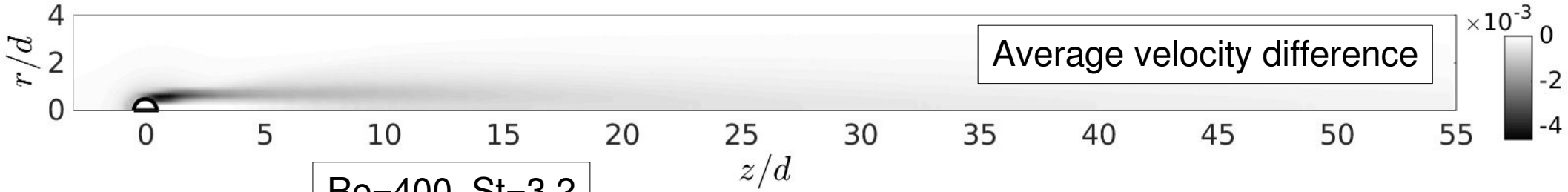
Resonant ($\mathrm{St} \simeq 1$) ejection from vortices determines starting point of overconcentration

Inter-particle collisions

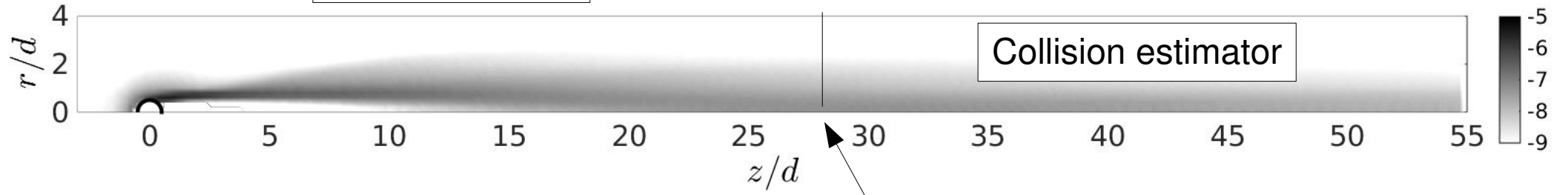
Rough collision estimator : $p_c(x) = -\langle n(x)^2 \rangle \langle w^-(x) \rangle$

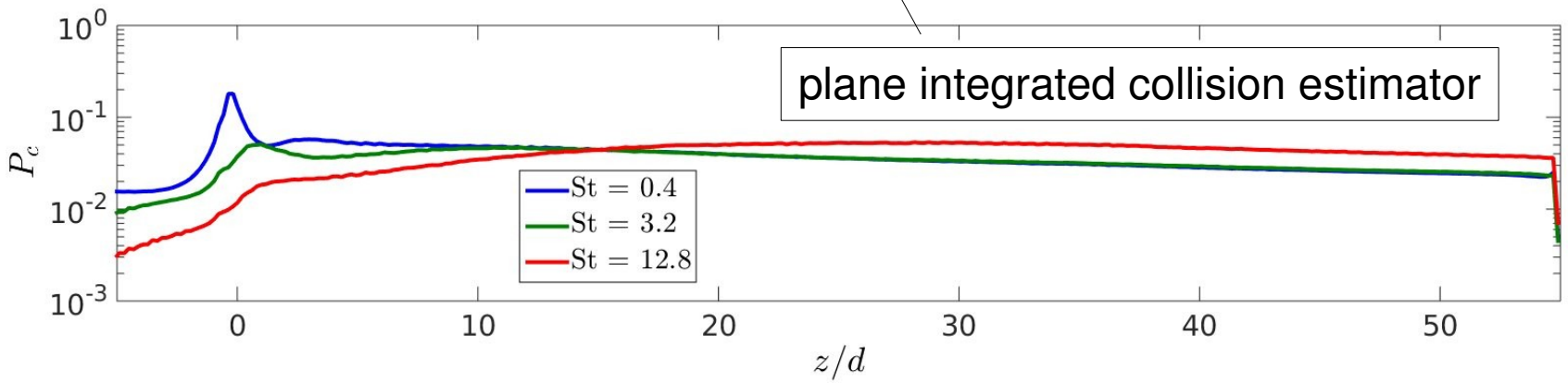
$n(x)^2$: number of particle pairs

approaching particles → $w^-(x)$ longitudinal velocity difference

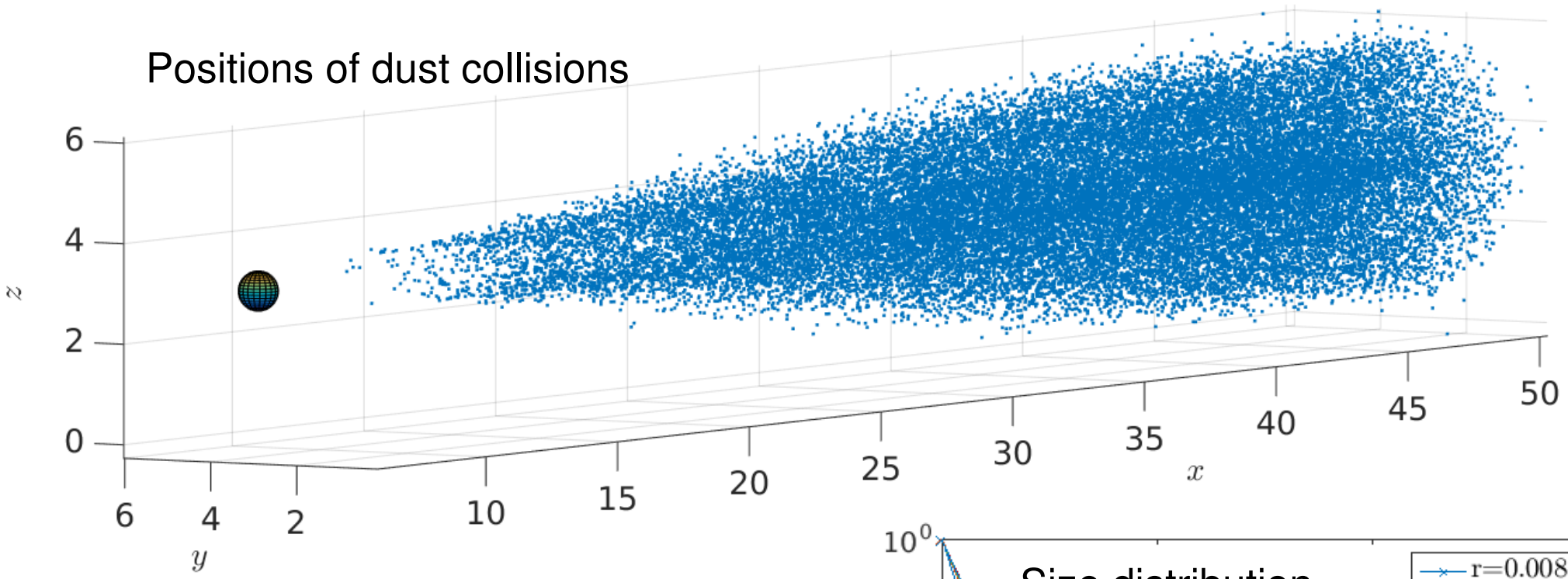


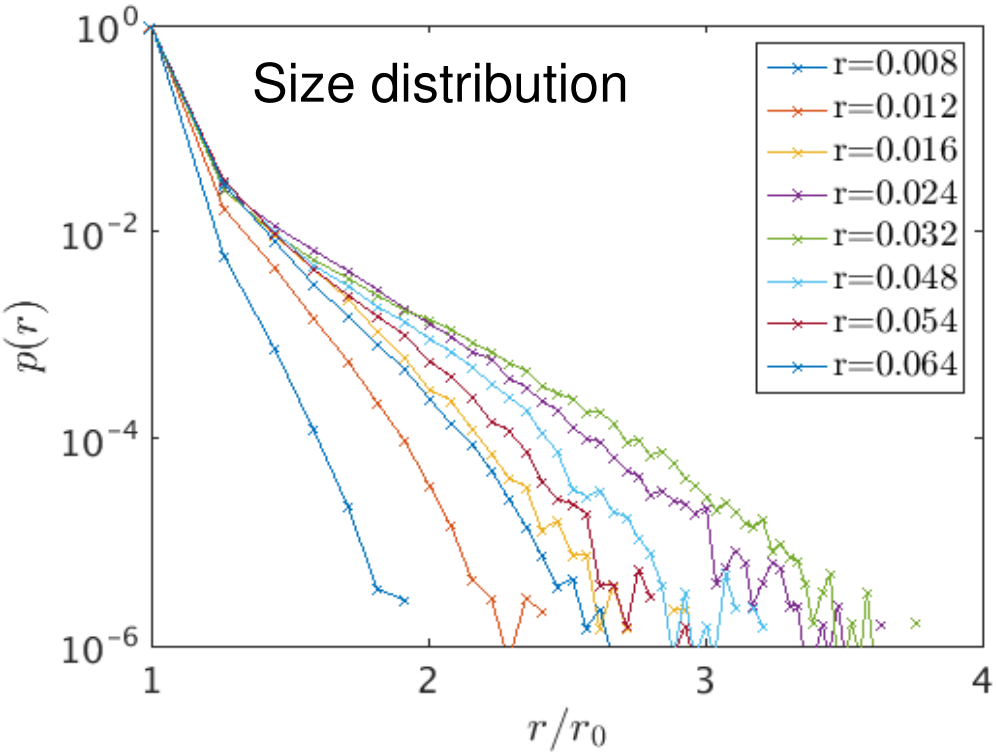
Re=400, St=3.2





Soon with “real” inter-dust collisions

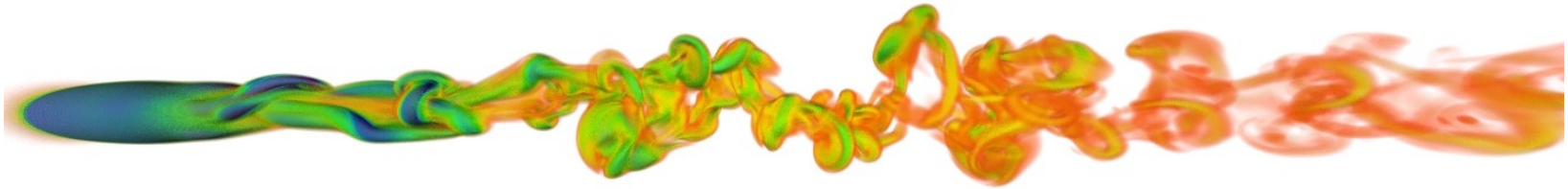




- Functional form of size-distribution
- Estimate processing efficiency of swarm of big particles (planetesimals)

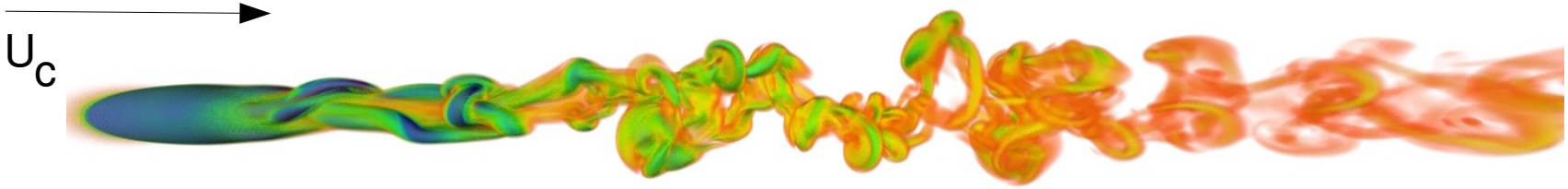
The drag force

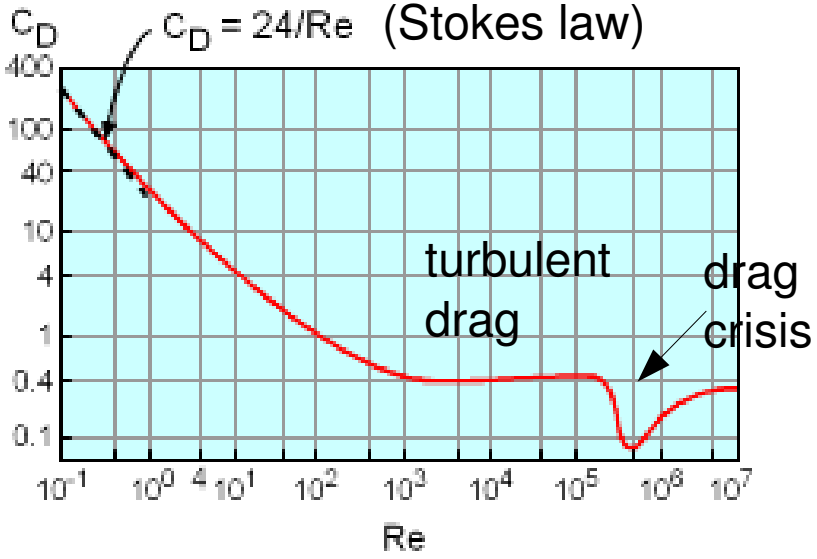
Particle in a laminar flow



Particle in a turbulent flow

The drag coefficient and flow profiles



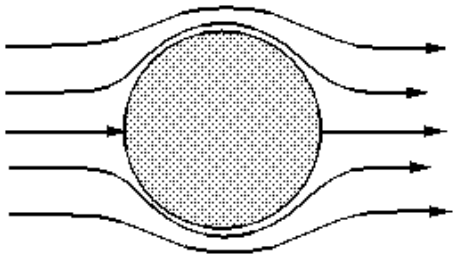


Drag-coefficient

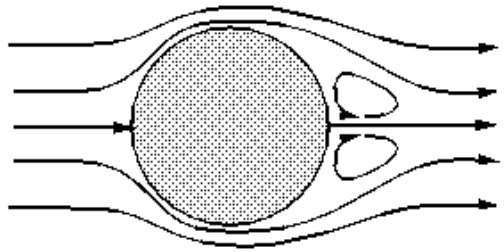
$$C_D = \frac{8\,F}{\rho\, U_c^2 \pi d^2}$$

$$Re_p = \frac{d\,U_c}{\nu}$$

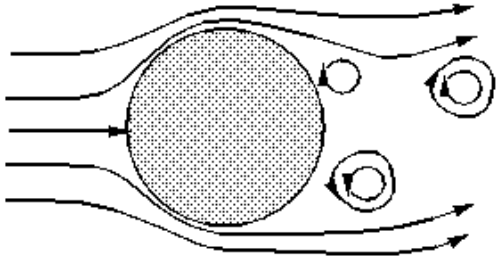
F	drag force
U_c	inflow speed
d	sphere diameter
ρ	fluid density
ν	fluid viscosity



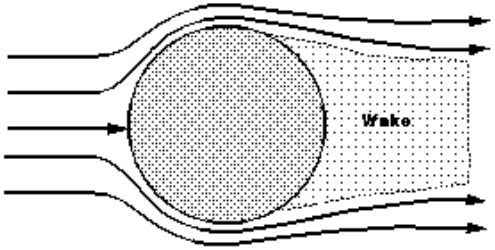
$Re_p \lesssim 23$



$23 \lesssim Re_p \lesssim 350$



$350 \lesssim Re_p \lesssim 10^5$



$Re_p \gtrsim 10^5$

Complicated drag force and boundary layer already for a constant inflow

The drag coefficient and flow profiles

NOW: Large sphere moving through turbulent fluid

- How does the drag force depend on turbulent fluctuations?
- How is the fluid affected?

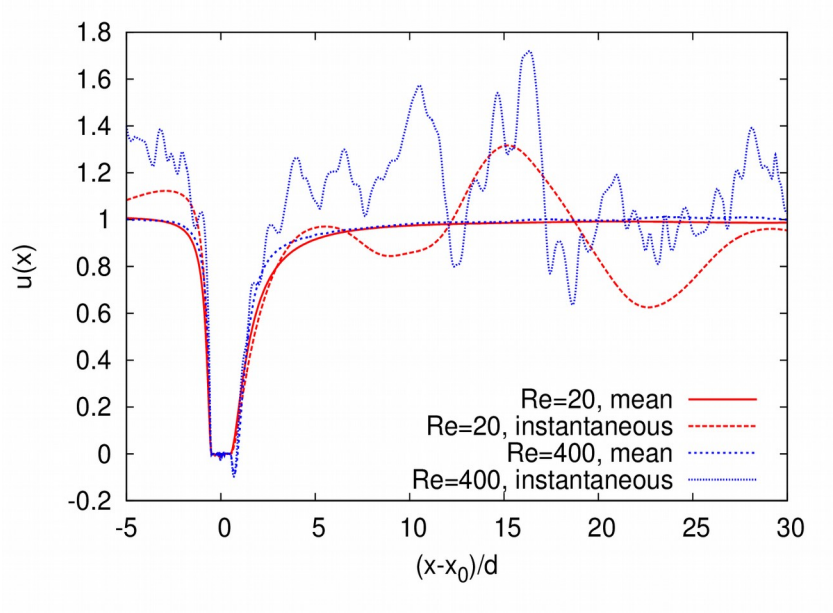
Dimensionless parameters :

- Particle Reynolds number $20 \leq Re_p = \frac{d\,U_c}{\nu} \leq 400$
- Fluid Reynolds number $0 \leq Re = \frac{u_{rms}\,L}{\nu} \leq 930$
- Turbulent intensity $0 \leq I = \frac{u_{rms}}{U_c} \leq 0.6$

Drag force from immersed boundary method: $\int_S \nabla \cdot \boldsymbol{T} dV = \int_S \boldsymbol{f_b} dV$

$\boldsymbol{T} = -p\boldsymbol{I}_3 + \nu(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T)$ stress-tensor

Drag force – Large scale effects



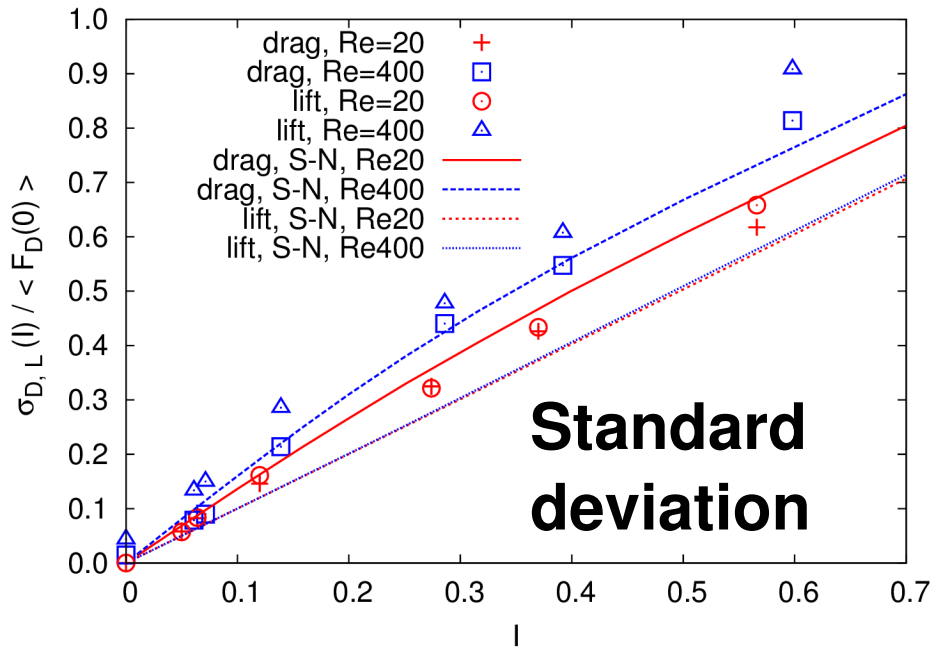
Drag is not linear

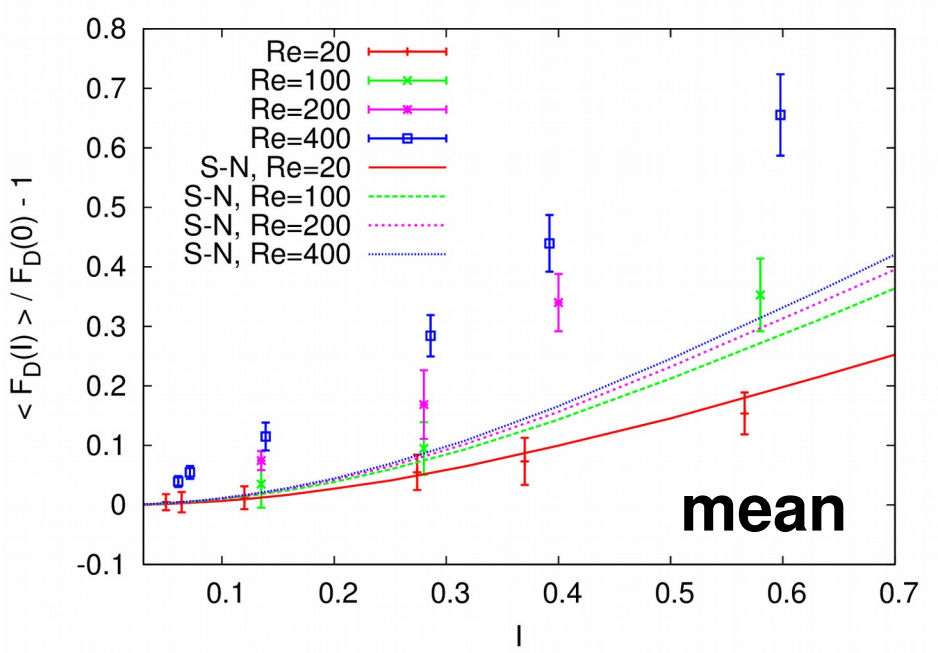
→ Effective drag force (Bagchi and Balachandar, 2003 & Kim and Balachandar, 2010)

Fitting formular of Schiller and Naumann (1933)

$$\boldsymbol{F} \approx \boldsymbol{F}^{\mathrm{SN}}(\boldsymbol{U}) = 3\pi\, d\, \nu \left[1 + 0.15 \left(\frac{|\boldsymbol{U}|\, d}{\nu}\right)^{0.687}\right] \boldsymbol{U}.$$

Test: $$\overline{\boldsymbol{F}}^{\mathrm{SN}}(I) = \frac{1}{\left(\sqrt{2\pi}\, u_{\mathrm{rms}}\right)^3} \int \boldsymbol{F}^{\mathrm{SN}}(\boldsymbol{U})\, \mathrm{e}^{-|\mathcal{U}|^2/(2u_{\mathrm{rms}}^2)}\, \mathrm{d}^3\mathcal{U}$$





Drag force – Small scale effects

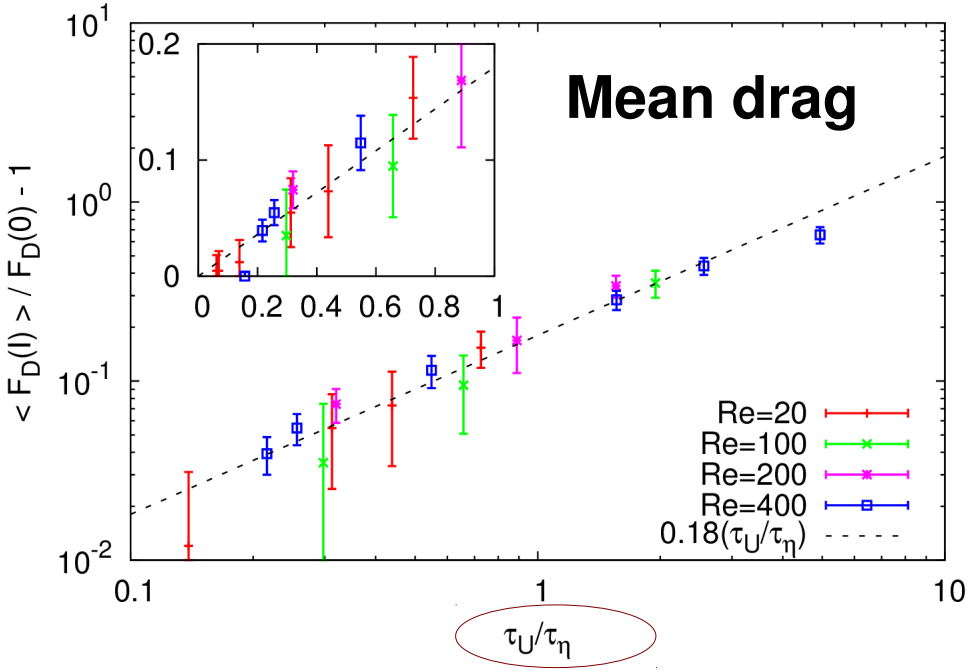
Another approach:
Drag increase by **modified velocity gradients** at the particle surface

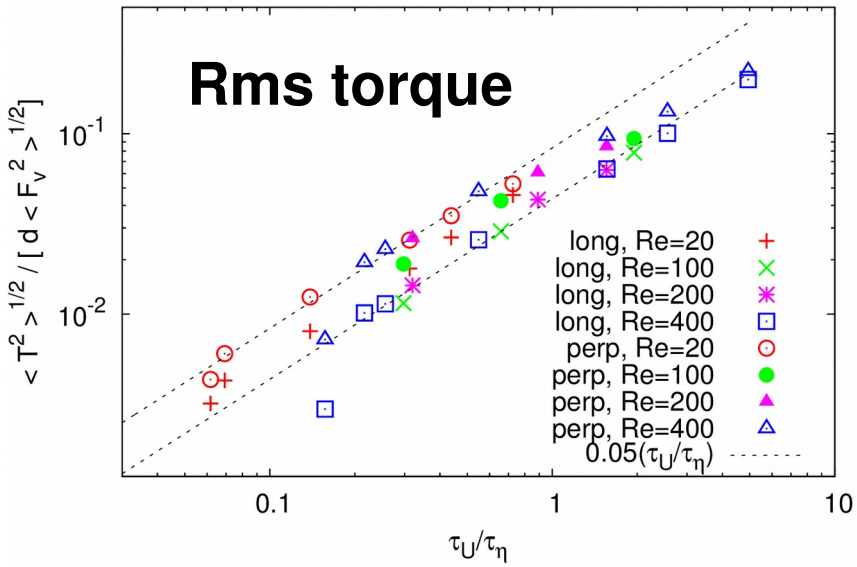
Laminar gradient $\sim U_c/d$

Turbulent gradient $u_\eta/\eta = \tau_\eta^{-1} = \nu^{-1/2}\varepsilon^{1/2}$

$\tau_\mathrm{U} = d/U_c$ Sweeping time

$\tau_\eta = (\nu/\varepsilon)^{1/2}$ Kolmogorov time scale

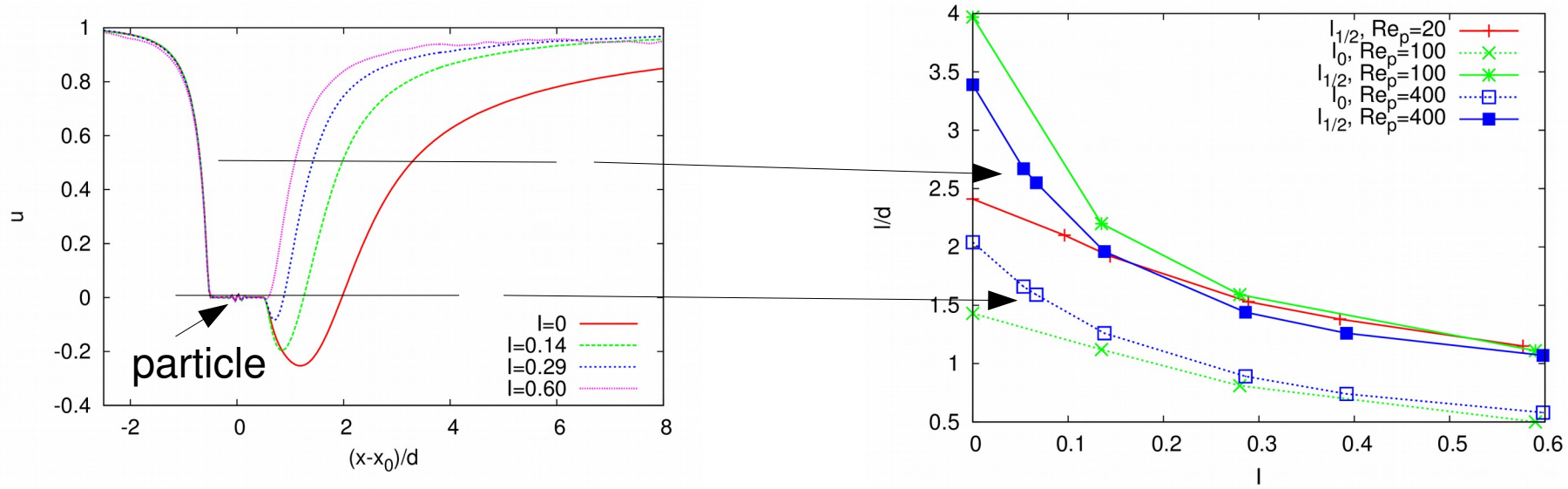




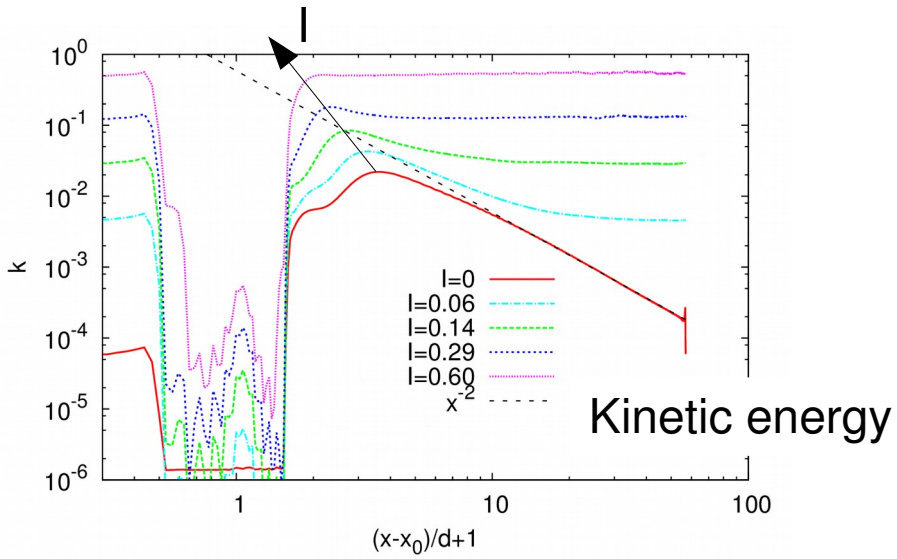
Small and large scales increase the drag on the particle

Shortening of the wake

Average velocity profile (Re=400)



Turbulent fluctuations significantely reduce wake length



Kinetic energy

- falls off quadratically
- maximum shifts to the particle surface

Conclusions

Outer turbulence leads to
- more dust – particle collisions
- higher dust impact speeds
- increased drag forces
- shortening of the wake

Turbulent wake leads to
- dust concentrations behind the sphere
- inter-dust collisions and heavy modification of the dust size distribution

Homann, Bec, Grauer (2013) JFM 721:155
Homann & Bec (2015) Phys. Fluids 27:053301
Homann, Guillot, Bec, Ormel, Ida, Tanga (2016) to appear in A&A

Perspectives
- Effect of rotation of sphere?
- Gravitational focusing?
- Effect of back-reaction of the particles?

Numerical implementation aspects

How to handle millions (or billions!) of particles in a numerical code?

Particle storage in HPC simulations

Array of Structure (AoS) vs Structure of Array (SoA)

```
struct Particle
{
  int id;

  double pos[3];
  double vel[3];

  float mass;
};

Particle particle;
particle.id = 1;

list<Particle> particleList;
particleList.push_back(Particle());
```

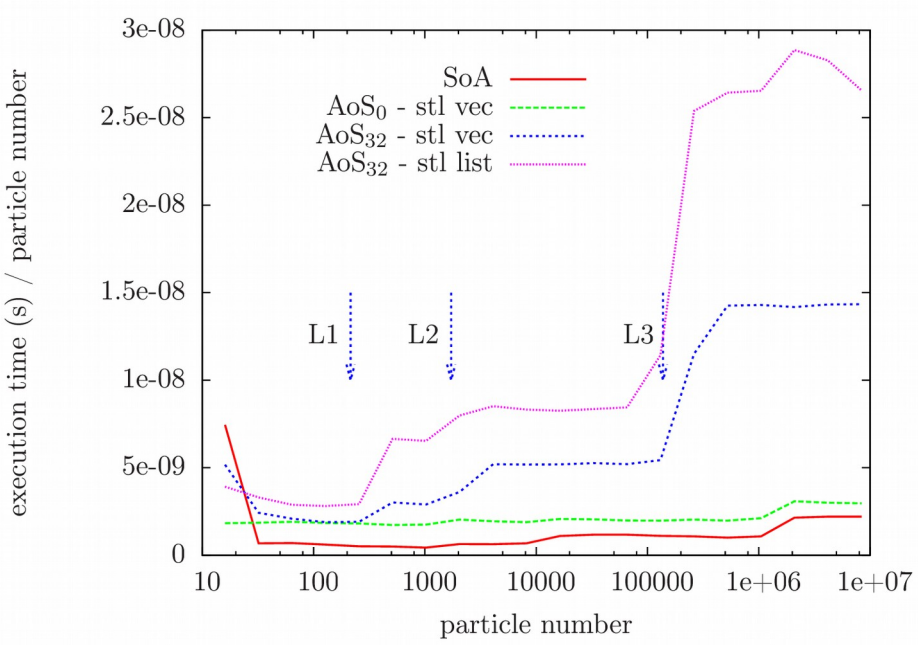
```
struct ParticleArray
{
  vector<int> id;

  vector<double> posX;
  vector<double> velX;

  vector<float> mass;

  void resize(int i) {
    id.resize(i);
    posX.resize(i);
    ...
  }

  void init() { ... }
  void output() { ... }
  ...
};
```

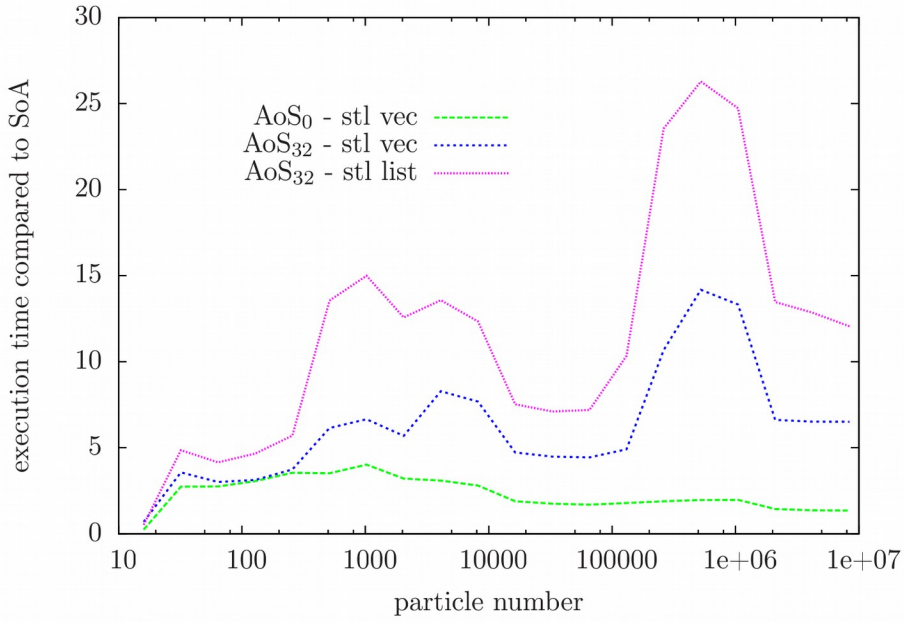
AoS (Particle = Object) is convenient:

- Modification of particle properties (charge, history)
- Creation/Deletion of particles
- I/O and MPI

Particle storage: Benchmark for CPUs

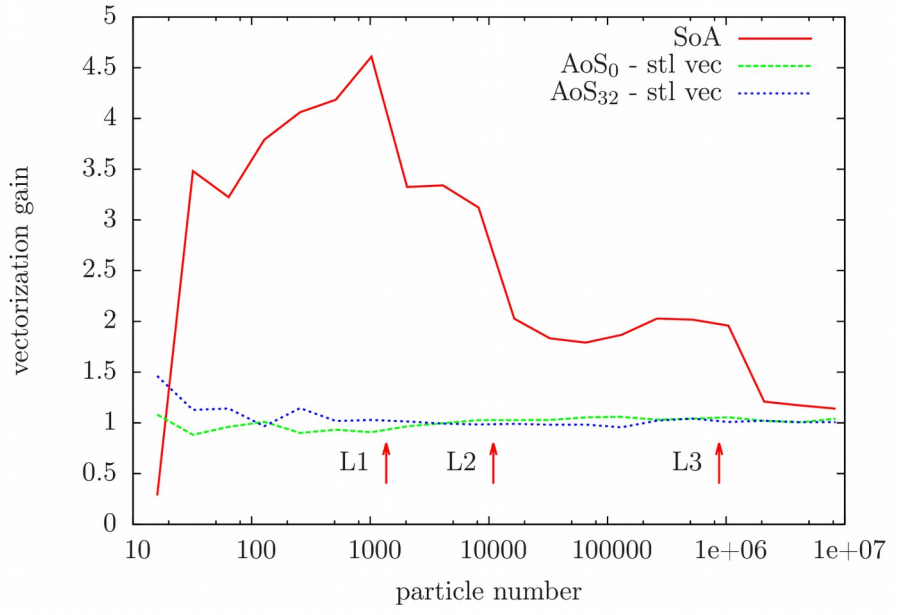
Euler-type for-loop over all particles: $\vec{x} += dt\,\vec{v}$





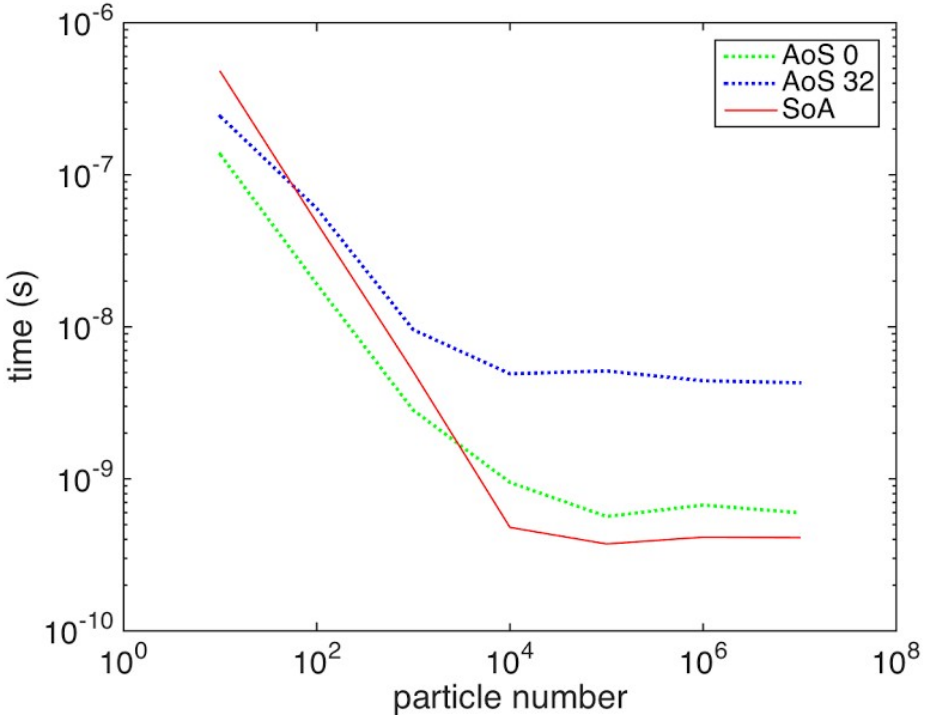
AoS are (much) slower than SoA!

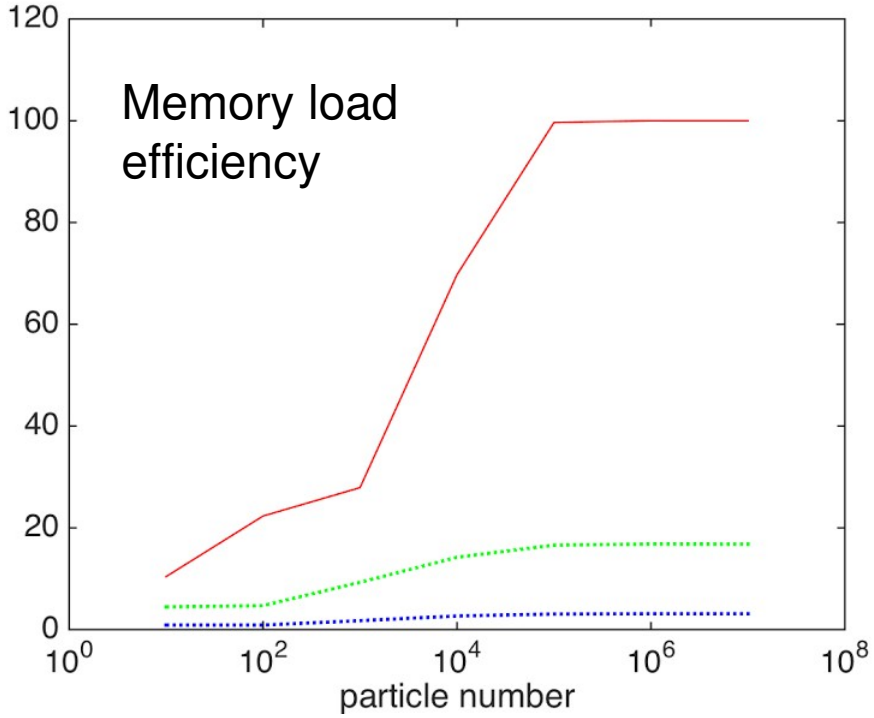
- Vectorization
  accelerating operations on cached data
- Cache
  Pollution from unused particle properties



Particle storage: Benchmark for **GPUs**

Euler-type for-loop over all particles: $\vec{x} += dt\,\vec{v}$





AoS are slower than SoA for large particle numbers

- Memory bandwidth: Pollution from unused particle properties
- Cache benefits at small particle numbers

A fast and flexible library (SoAX)

```
// Define attributes by SOAX macro
SOAX_ATTRIBUTE(id, 'N');        // identity (number)
SOAX_ATTRIBUTE(pos, 'P');       // position
SOAX_ATTRIBUTE(vel, 'V');       // velocity
SOAX_ATTRIBUTE(mass, 'M');      // mass

// std::tuple holding attributes
// of specified type and dimension
typedef std::tuple<id<int,1>,
                   pos<double,3>,
                   vel<double,3>,
                   mass<float,1>> Attributes;

// create structure of array for 42 particles
Soax<Attributes> soa(42);

// access attributes:
soa.id(0) = 0;           // set identity
soa.pos(i,0) = 100;      // set x-coordinate

// array-wise operations using
// expression templates (x = vy - vz)
soa.posArr(0) = soa.velArr(1) - soa.velArr(2);

// extract and treat a particle as an object
auto particle = soax.getElement(7);
particle.id() = 42;
particle.pos(0) = 3.14;
soax.push_back(particle);
```

- Optimal Performance (same as c-arrays)
- Handyness of Arrays of Structure (particle = object)

Implementation of SoAx

- Inheritance
- C++11
- **Template meta-programming**
  Simple example: Compute the factorial at compile time

```
template <int N>
struct Factorial
{
    enum { value = N * Factorial<N - 1>::value };
};

template <>
struct Factorial<0>
{
    enum { value = 1 };
};
```

`Factorial<11>::value` is a compile time constant

Implementation of SoAx

Template meta-programming

Second example: User provided function on arrays

Set all particle data to a given value (here 42):

Code to write by user:
+
Function object (SetToValue):
=
Code generated at compile time

```
p.apply<SetToValue>(42);
```

```
struct SetToValue
{
  template<class T, class Type>
  static void doIt(T& t, Type value) {
    auto tRef = *t;
    for(int i=0;i<t->size();i++)
    t[i] = value;
  }
};
```

Implementation of SoAx

Application of Functor (SetToValue) to all arrays by recursive instantiation:

```
template<class Tuple, std::size_t N, class DoItClass>
struct TupleDo {

  template<class... Args>
  static void doIt(Tuple& t, Args... args)
  {
    TupleDo<Tuple, N-1, DoItClass>::doIt(t,args...);
    DoItClass::doIt(std::get<N-1>(t),args...);
  }
};

template<class Tuple, class DoItClass>
struct TupleDo<Tuple, 1, DoItClass> {

  template<class... Args>
  static void doIt(Tuple& t, Args... args)
  {
     DoItClass::doIt(std::get<0>(t),args...);
  }
};
```

Conclusions

Use Structures of Arrays!

Homann & Laenen (2016) in preparation for Comp. Phys. Comm.

Thank you for your attention!