Lecture series on Kinetic Theory and Applications to Fluid Mechanics

Fabio Guglietta

References

- **Lattice Boltzmann Method:**
  - Krüger, T., Kusumaatmaja, H., Kuzmin, A., Shardt, O., Silva, G., & Viggen, E. M. (2017). The lattice Boltzmann method. Springer International Publishing, 10(978-3), 4-15.
  - Succi, S. (2001). The lattice Boltzmann equation: for fluid dynamics and beyond. Oxford university press.
  - Benzi, R., Succi, S., & Vergassola, M. (1992). The lattice Boltzmann equation: theory and applications. Physics Reports, 222(3), 145-197.
- **Immersed Boundary Method:**
  - Peskin, C. S. (2002). The immersed boundary method. Acta numerica, 11, 479-517.
  - Verzicco, R. (2023). Immersed boundary methods: Historical perspective and future outlook. Annual Review of Fluid Mechanics, 55, 129-155.
- **CUDA Programming**:
  - Ruetsch, G., & Fatica, M. (2013). CUDA Fortran for scientists and engineers: best practices for efficient CUDA Fortran programming. Elsevier.
  - Cook, S. (2012). CUDA programming: a developer's guide to parallel computing with GPUs. Newnes.
  - Sanders, J., & Kandrot, E. (2010). CUDA by example: an introduction to general-purpose GPU programming. Addison-Wesley Professional.

Multiphase and multicomponent flows

Shan-Chen model

Fabio Guglietta

Multiphase and multicomponent flows

- Single-component multi**phase** flows:
  - **liquid** and **gas** phases of the **same substance** are in **coexistence**
  - These two phases can **interconvert** from one to another: gas $\Longleftrightarrow$ liquid

- Multi**component** flows:
  - contain two (or more) **different substances** (e.g., water and oil)
  - substances do not interconvert (diffusion between components)

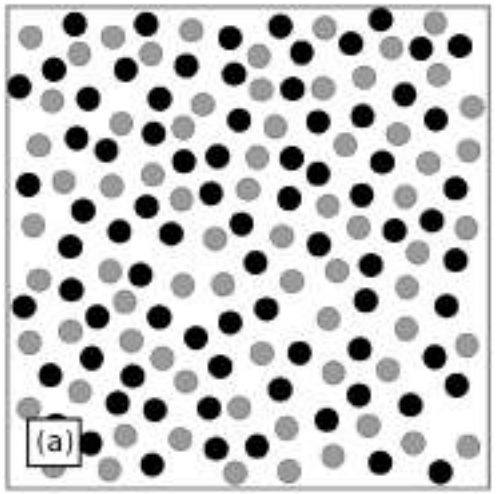
Order parameter

- To **distinguish** two fluid phases or two components

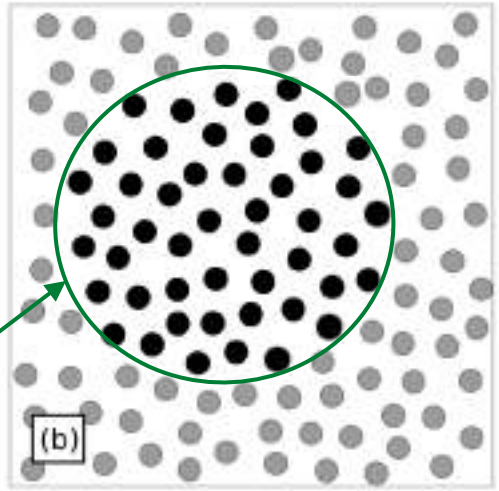
- **Multiphase** flows:
  - density (ρ_g and ρ_l)

- **Multicomponent** flows:
  - density not good (e.g., water and oil have similar densities)
  - $\phi = \dfrac{\rho^{(1)} - \rho^{(2)}}{\rho^{(1)} + \rho^{(2)}}$
  - $\phi = +1$ (pure component 1); $\phi = -1$ (pure component 2).

Miscibile

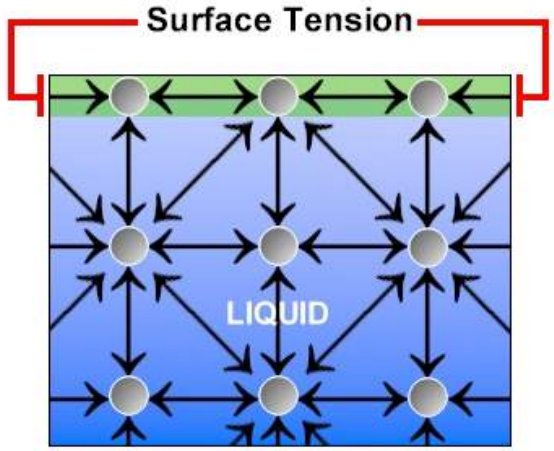


Immiscible



Interface
(surface tension)

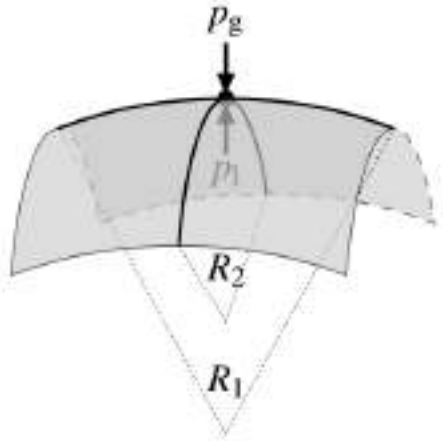
Surface tension and Laplace pressure

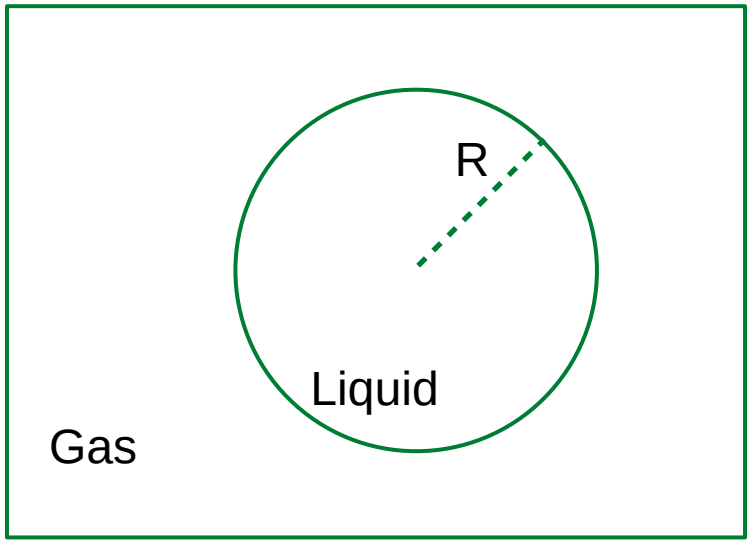


- **Surface tension**: energy per unit area required to form the interface between the two fluid phases or components.

- **Laplace pressure**:
  - Consider a droplet of one fluid (e.g., liquid) suspended in another fluid (e.g., gas)

$$p_l - p_g = \gamma \left(\frac{1}{R_1} + \frac{1}{R_2} \right)$$





$$p_l - p_g = \frac{2\gamma}{R}$$

Surface tension and Laplace pressure

- Consider two fluids
- Surface tension: γ
- $p_2 > p_1$
- Work to increase sphere: $W = \gamma \Delta A$
- We also have $W = p_1 \Delta V_1 + p_2 \Delta V_2$
- $\Delta V_1 = -\Delta V_2$

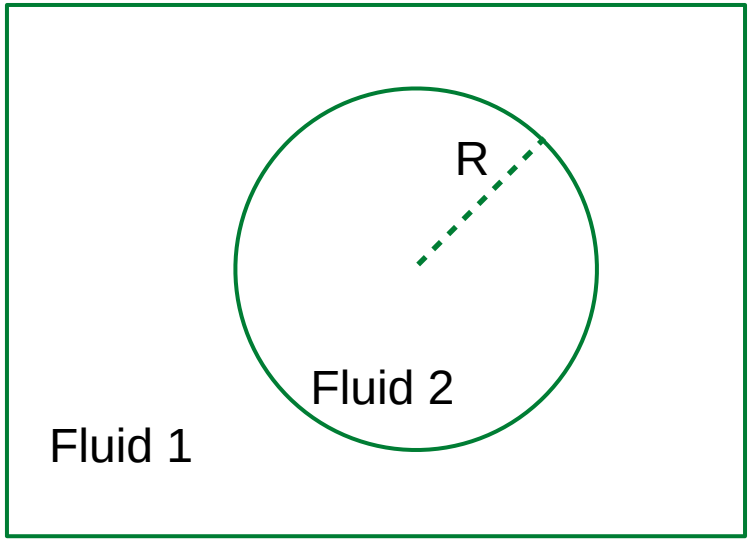
$$\Longrightarrow \Delta V_2 (p_2 - p_1) = \gamma \Delta A$$

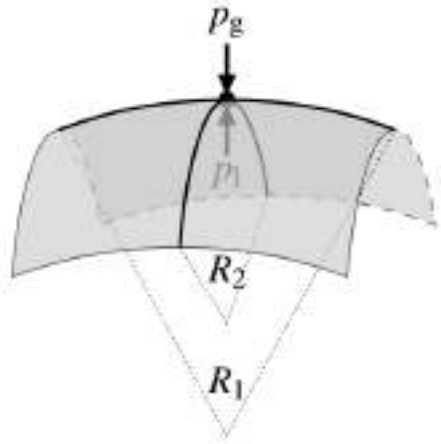
$$\Longrightarrow (p_2 - p_1) = \gamma \frac{\Delta A}{\Delta V_2}$$

$\sim R^2$

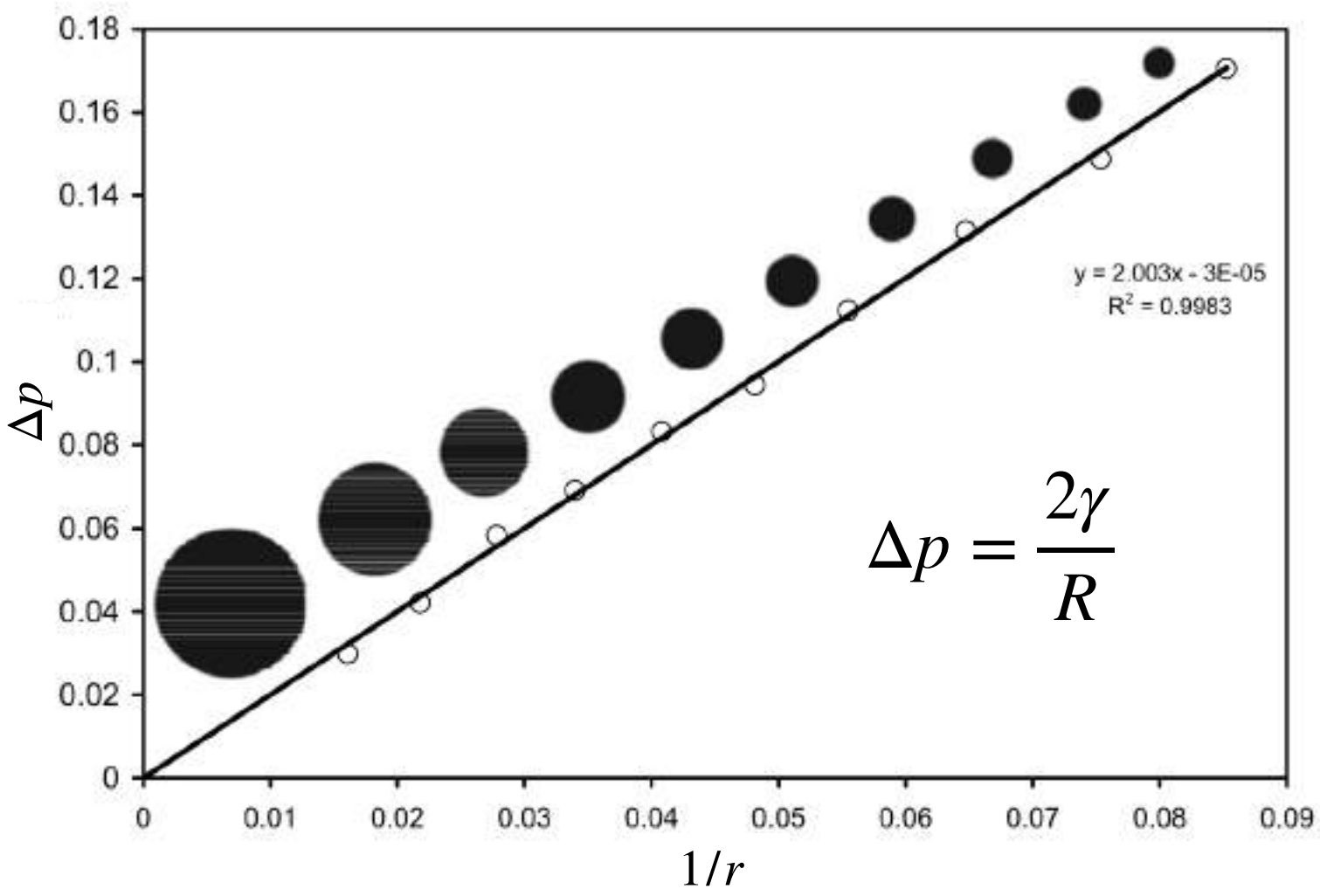
$\sim R^3$

$$\Delta p = \gamma \left(\frac{1}{R_1} + \frac{1}{R_2} \right)$$





Laplace test



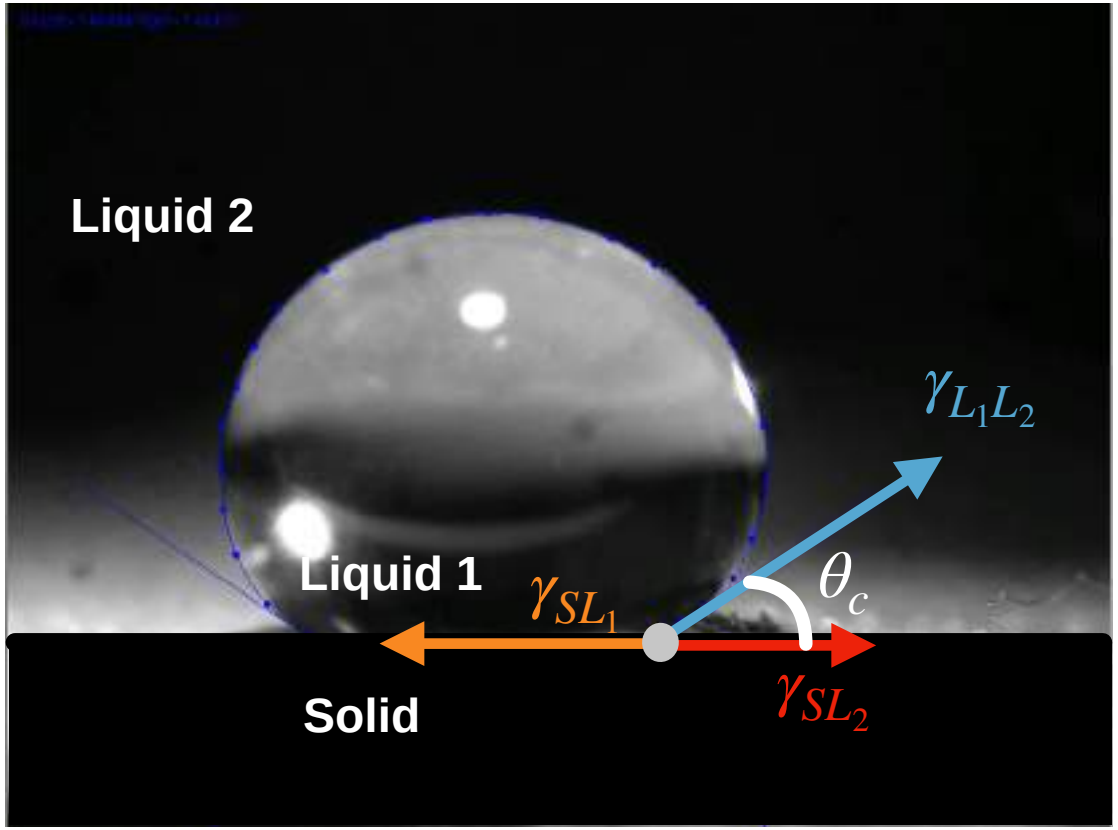
Sukop, M. C., Thorne, D. T. , (2006). Lattice Boltzmann Modeling

Contact angle: Young's equation

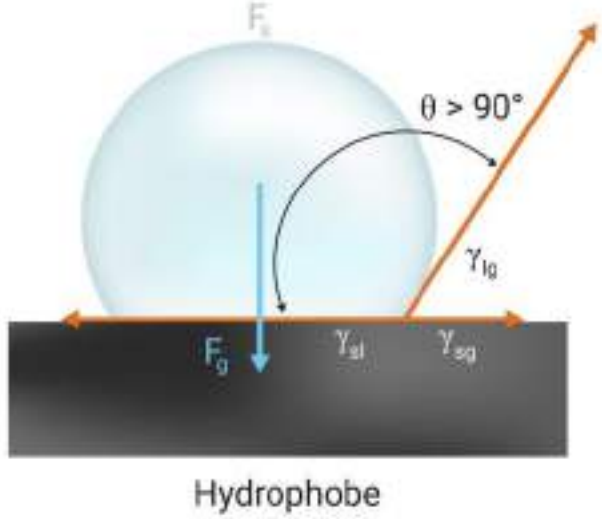
- Thermodynamic equilibrium between three phases:
  - Solid
  - Liquid 1
  - Liquid 2 (gas)

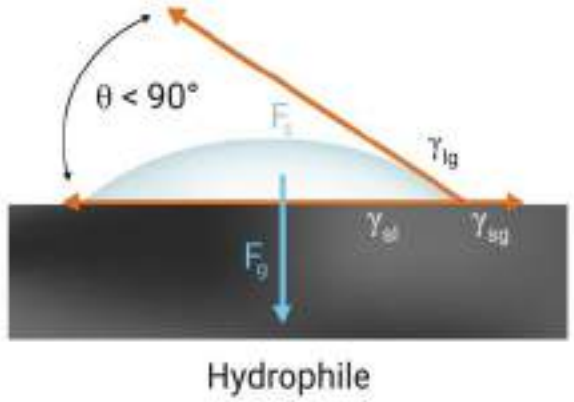
$$\gamma_{SL_1} - \gamma_{SL_2} - \gamma_{L_1L_2}\cos\theta_c = 0$$

$$\cos\theta_c = \frac{\gamma_{SL_1} - \gamma_{SL_2}}{\gamma_{L_1L_2}}$$



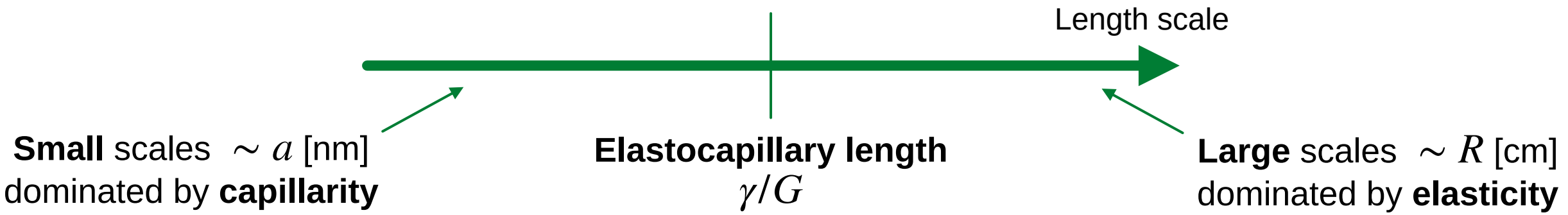
Hydrophobe vs hydrophile





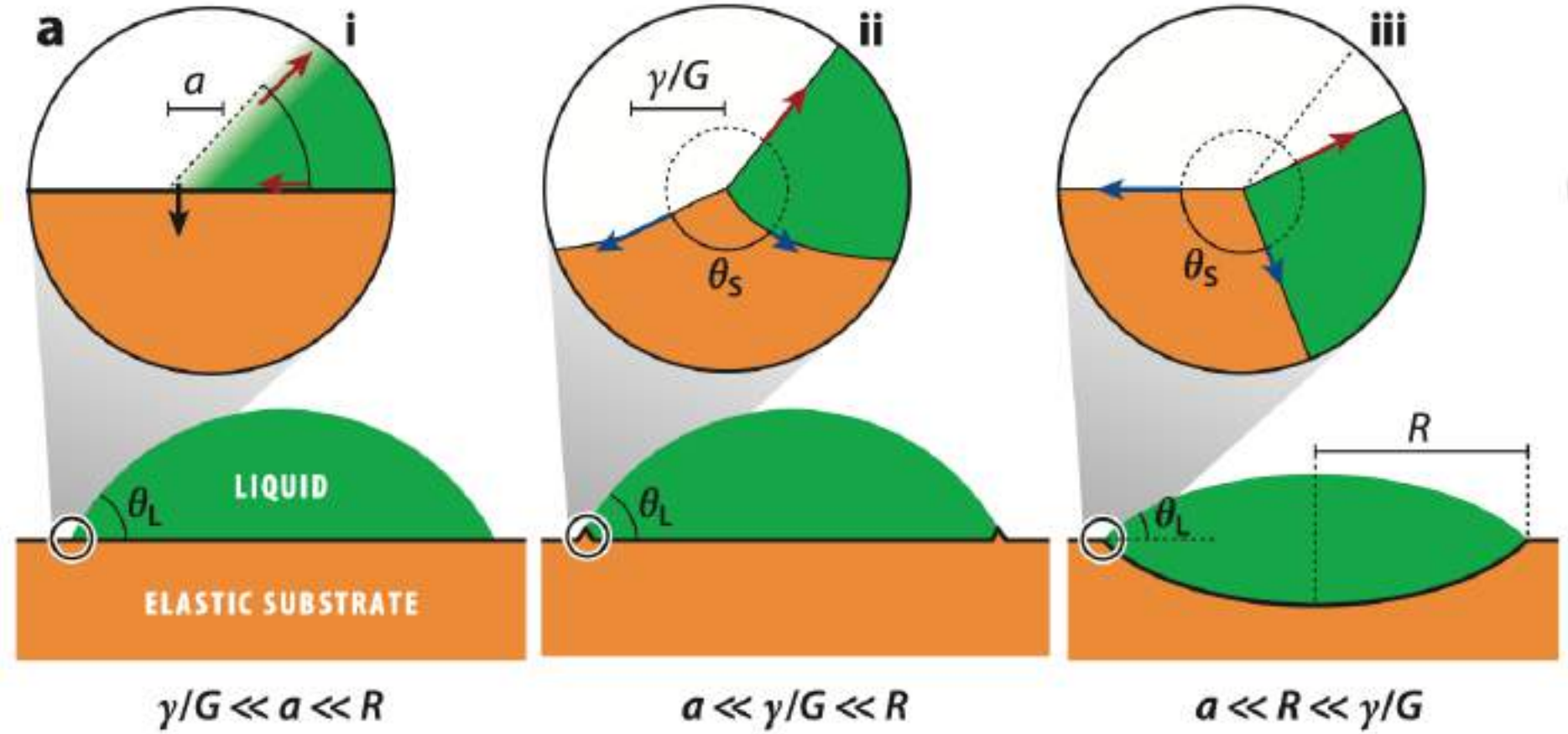
Hard and soft wetting

- Continuum perspective: **elastocapillary** phenomena can be classified in terms of **length scales**



- By tuning the substrate stiffness, γ/G can be varied over order of magnitude

Hard and soft wetting



Small elastocapillary length → Large elastocapillary length

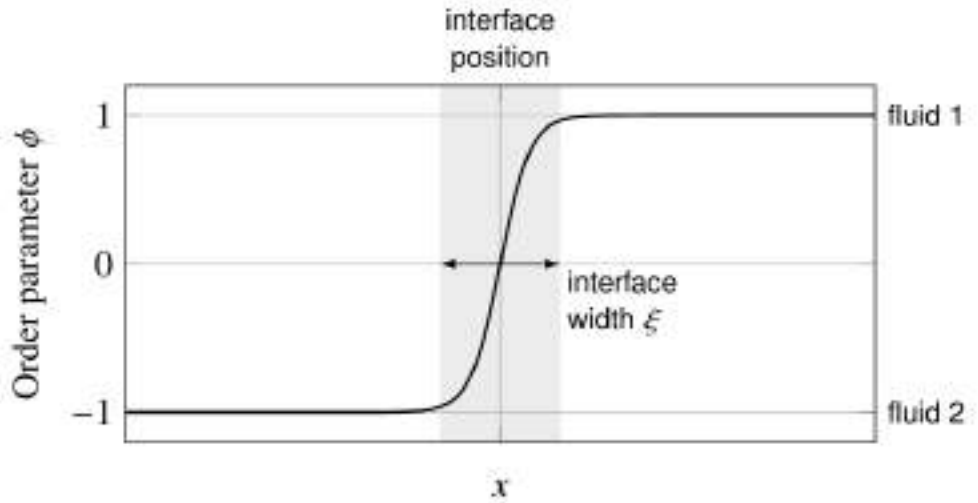
Hard wetting → Soft wetting

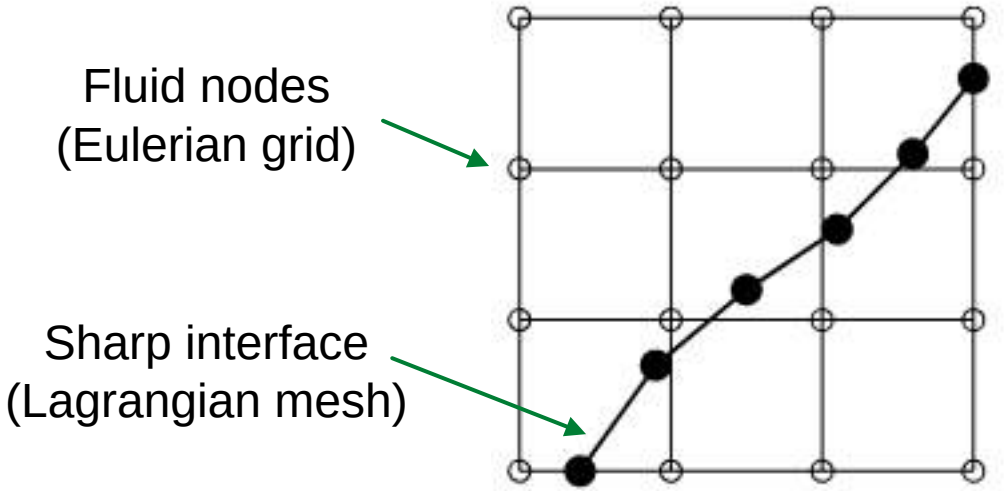
Andreotti, B., & Snoeijer, J. H. (2020). Statics and dynamics of soft wetting. *Annual review of fluid mechanics*, *52*, 285-308.

How do we model interfaces?

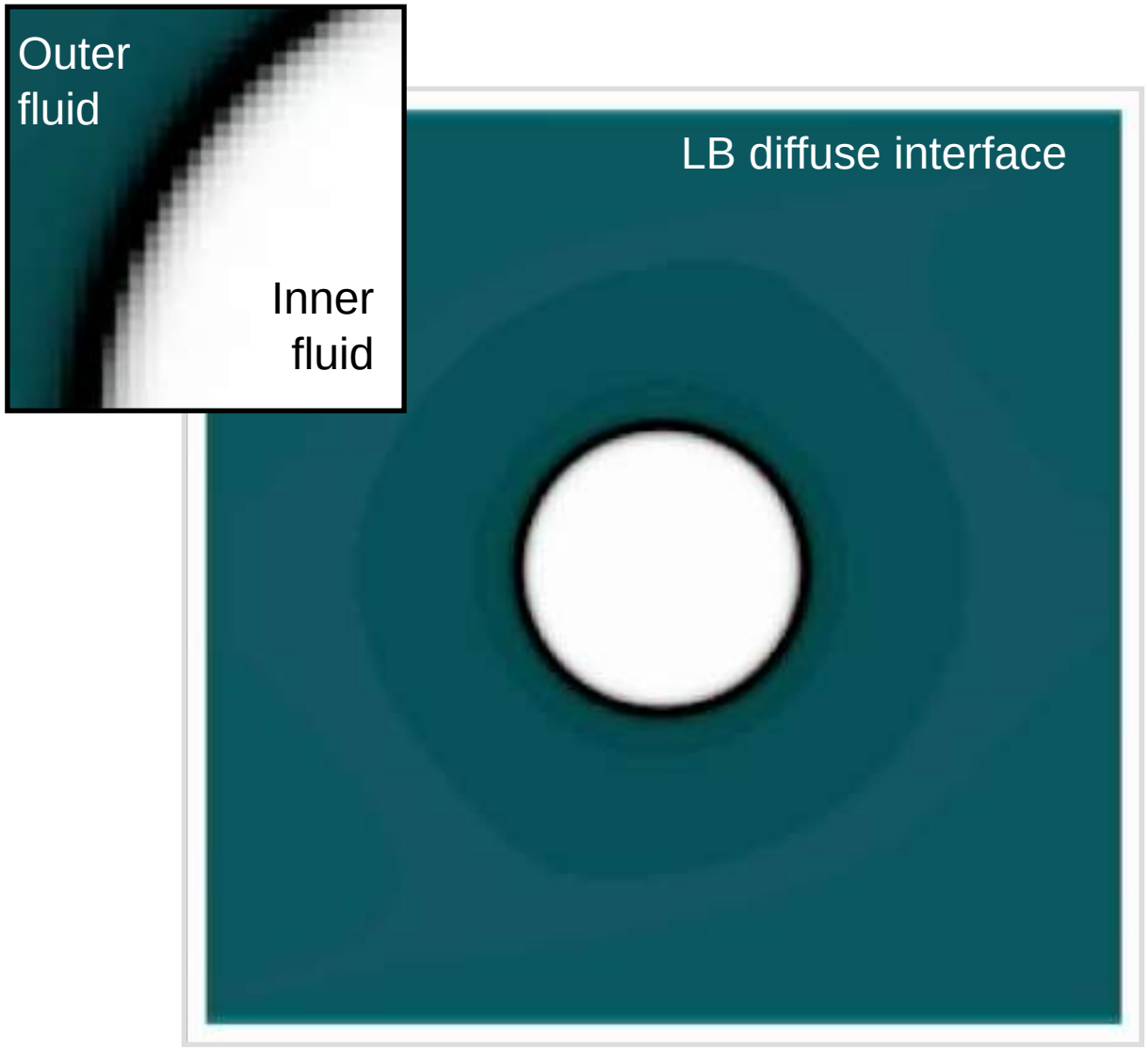
- **Diffuse interface** models (employed in the LB community)
  - 1D order parameter (density for multiphase, ϕ for multicomponent)
  - The order parameter profile **smoothly varies** across the interface between the two bulk values.
  - Realistic values ~ nm
  - Lattice values: several lattice nodes (stability)
  - Pro: no need to track the interface
  - Con: Difficulty to model **viscoelastic properties**

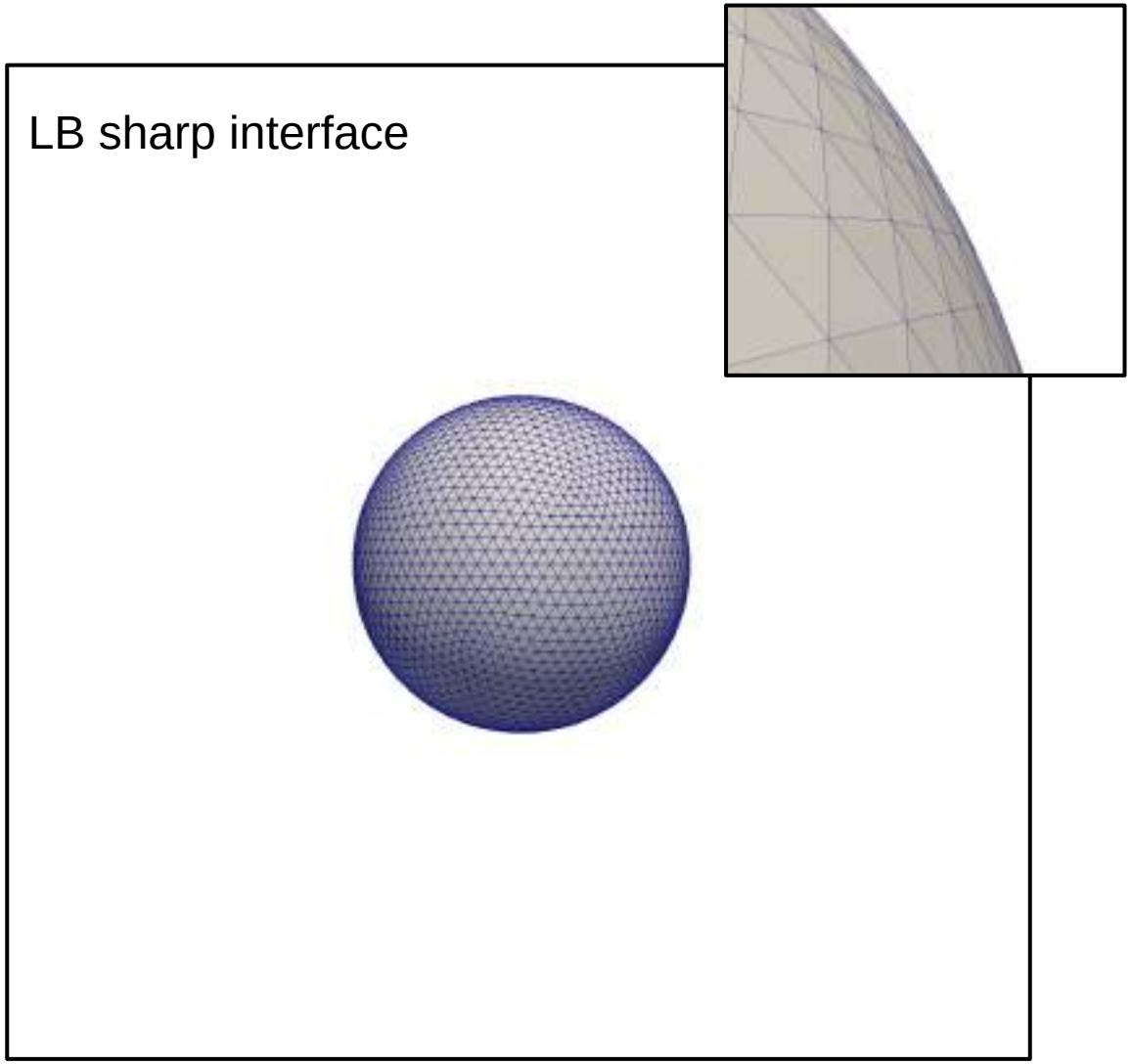
- **Sharp interface** models
  - Interface as a 2D boundary
  - Represented via **Lagrangian mesh**
  - Pro: Easy to implement any kind of **viscoelastic model**
  - Con: Need to introduce another “structure” for the interface





LBM: Diffuse vs sharp interface



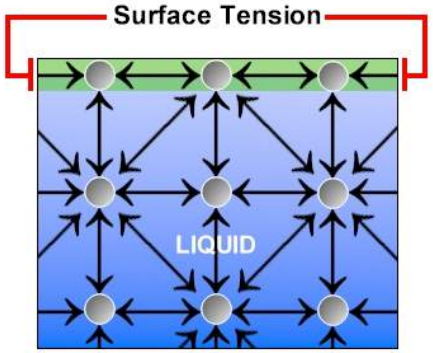


Diffuse interface: Shan-Chen model

- Bottom-up approach:
  - **Postulate** a **microscopic interaction** between fluid elements
  - Interaction **potentials** that lead to phase separation
  - **Surface tension** as an **emergent** effect
    [typical of LBM: based on **mesoscopic rules** with **emergent transport coefficients** (viscosity)]

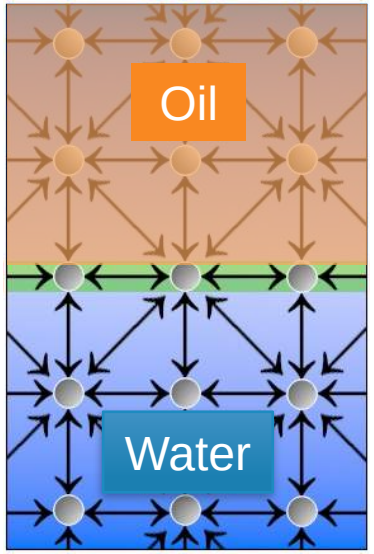
Shan, X., & Chen, H. (1993). Lattice Boltzmann model for simulating flows with multiple phases and components. *Physical review E*, *47*(3), 1815
Shan, X., & Chen, H. (1994). Simulation of nonideal gases and liquid-gas phase transitions by the lattice Boltzmann equation. *Physical Review E*, *49*(4), 2941.

Shan-Chen model



- **Coexistence liquid-gas**: attractive force between molecules in the liquid phase

- **Multicomponent** (e.g., oil-water): water-water interaction different from oil-oil interaction



- Could we introduce a **suitable local force** (potential) between **fluid elements**?
- Suitable: **thermodynamically consistence**
  (pressure and equilibrium densities (at given T) should be the same as those derived from thermodynamic principles)
- **Suitable is not enough**: large **surface tensions** or large **liquid-gas density** ratios can lead to numerical **instability**

Shan-Chen model: interaction force

1. **Intermolecular forces** act between **pairs** of molecules and are **additive**

$\Longrightarrow$ Interaction between fluid elements in x and $\tilde{x}$ proportional to $\rho(x)\rho(\tilde{x})$

2. **Interaction** depends on the **distance**

$\Longrightarrow$ Kernel function carrying information on the spatial dependency of the force: $G(x, \tilde{x})$

- We introduce **pseudo potential** $\psi(\rho) = \rho_0(1 - e^{-\rho/\rho_0})$ where ρ_0 is a reference density (typically $\rho_0 = 1$).
  Note: if $\rho \ll \rho_0 \longrightarrow \psi(\rho) \approx \rho$

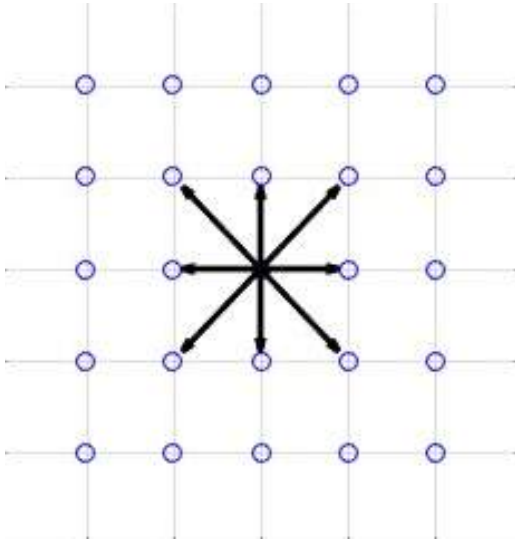
$$\mathbf{F}^{SC} = -\int G(\mathbf{x}, \tilde{\mathbf{x}})\psi(\mathbf{x})\psi(\tilde{\mathbf{x}})(\tilde{\mathbf{x}} - \mathbf{x})d\tilde{\mathbf{x}}$$

Discretised Shan-Chen force

$$\mathbf{F}^{SC} = -\int G(\mathbf{x},\tilde{\mathbf{x}})\psi(\mathbf{x})\psi(\tilde{\mathbf{x}})(\tilde{\mathbf{x}}-\mathbf{x})d\tilde{\mathbf{x}}$$

Discretisation for kernel: $G(\mathbf{x},\tilde{\mathbf{x}}) = \begin{cases} w_i G & \text{for } \tilde{\mathbf{x}} = \mathbf{x} + \mathbf{c}_i\Delta t \\ 0 & \text{otherwise} \end{cases}$

G is a simple scalar that controls the strength of the interaction



$$\mathbf{F}^{SC} = -\psi(\mathbf{x})G\sum_i w_i(\mathbf{x}+\mathbf{c}_i\Delta t)\mathbf{c}_i\Delta t$$

Shan-Chen model for multicomponent flows

σ: fluid component

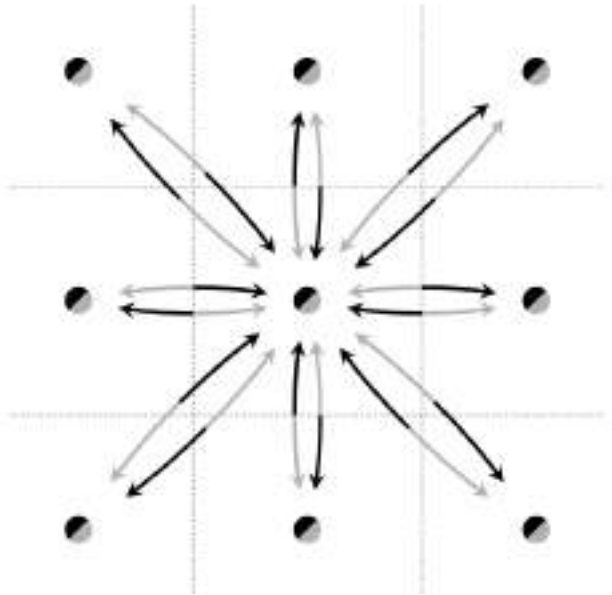
$$\boldsymbol{F}^{\mathrm{SC}(\sigma)}(\boldsymbol{x}) = -\psi^{(\sigma)}(\boldsymbol{x}) \sum_{\tilde{\sigma}\neq\sigma} G_{\sigma\tilde{\sigma}} \sum_i w_i \psi^{(\tilde{\sigma})}(\boldsymbol{x} + \boldsymbol{c}_i \Delta t)\boldsymbol{c}_i \Delta t$$

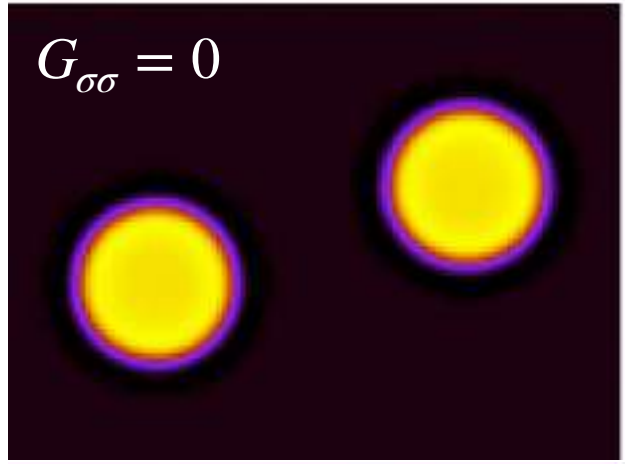
$$f_i^{(\sigma)}(\mathbf{x} + \mathbf{c}_i\Delta t, t + \Delta t) - f_i^{(\sigma)}(\mathbf{x}, t) = -\Delta t \frac{f_i^{(\sigma)}(\mathbf{x}, t) - f_i^{eq(\sigma)}(\mathbf{x}, t)}{\tau} + \left(1 - \frac{\Delta t}{2\tau^{(\sigma)}}\right) F_i^{(\sigma)}(\mathbf{x}, t)\Delta t$$

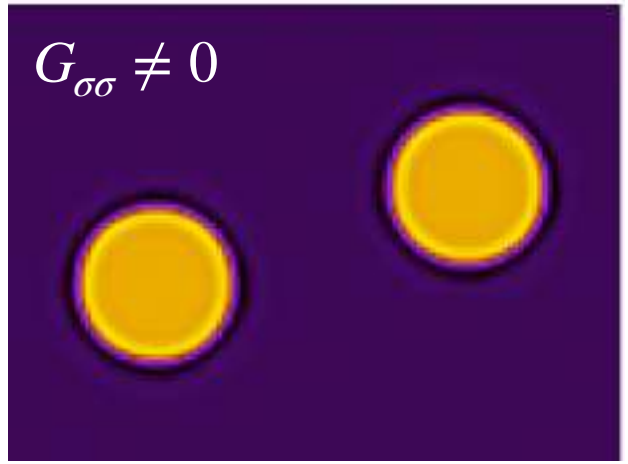
Barycentric velocity

$$\mathbf{u}_b = \frac{1}{\rho}\sum_\sigma \left[\sum_i f_i^{(\sigma)}\mathbf{c}_i + \frac{\mathbf{F}^{SC(\sigma)}\Delta t}{2}\right]$$

$$\rho = \sum_\sigma \rho^{(\sigma)}$$

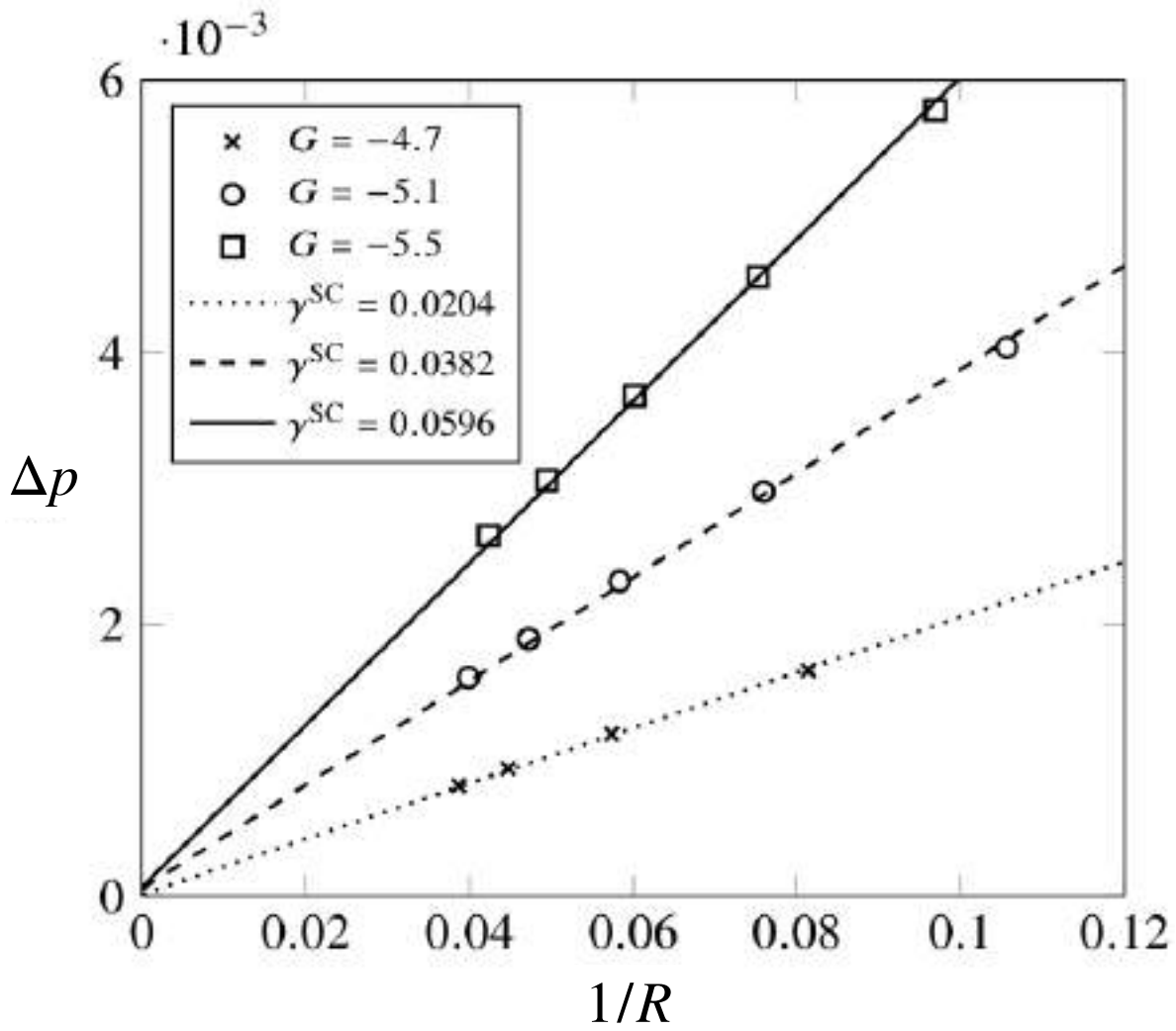






Benzi, R., Chibbaro, S., & Succi, S. (2009). Mesoscopic lattice Boltzmann modeling of flowing soft systems. Physical review letters, 102(2), 026002.

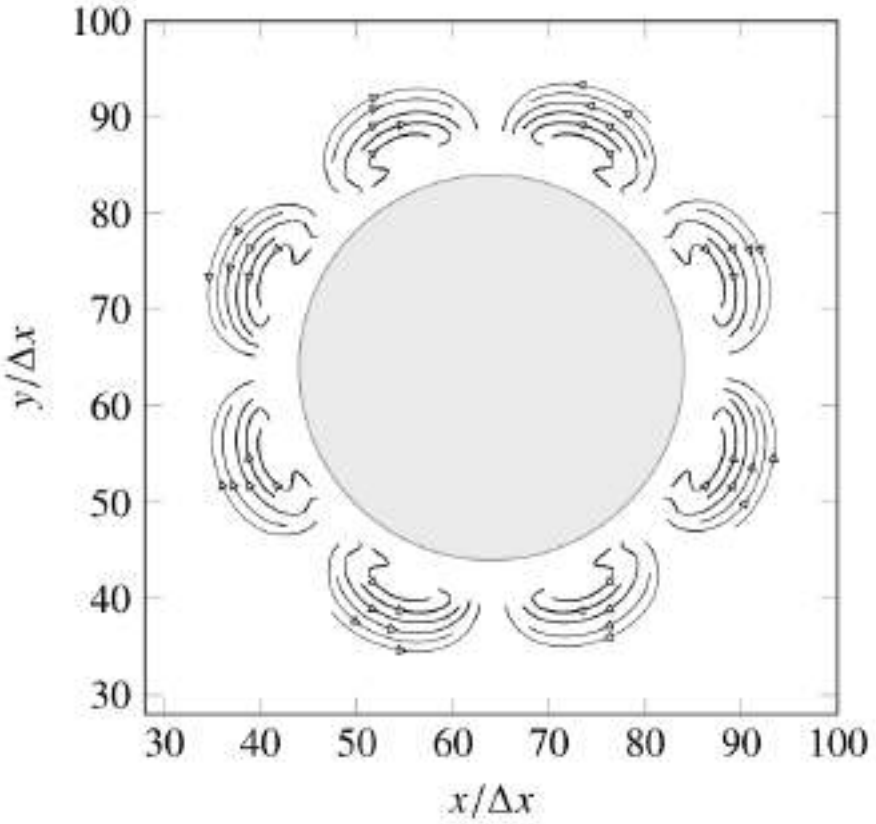
Laplace test with Shan-Chen model



$$\Delta p = \frac{2\gamma}{R}$$

Limitations: Spurious currents

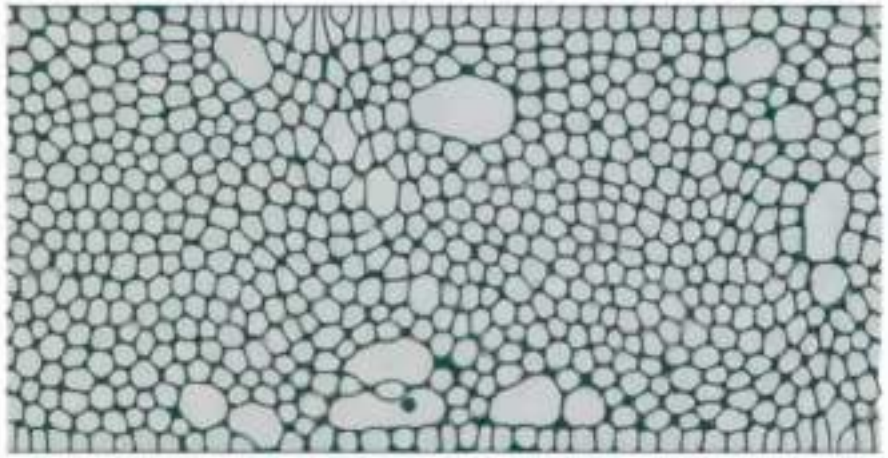
- Steady drop: zero fluid velocity everywhere
- However, Shan-Chen models (and also other models) create **spurious currents**
- Caused by **numerical approximations** of the surface tension force
  (forces do not point exactly towards the centre)
- Characteristic flows vs. Spurious currents



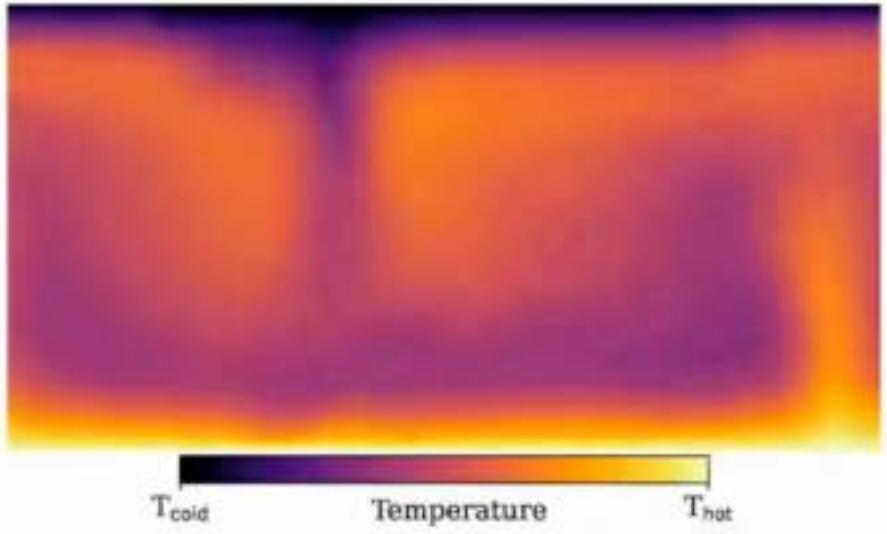
Limitations: surface tension and viscosity ratio

- Range of values limited: **large surface tensions** $\Longrightarrow$ **instability** (spurious currents)
- Interface-governed flows: dimensionless parameters
  - Ca $= \eta u/\gamma$
  - Bo $= \rho g L^2/\gamma$
- Original Shan-Chen model: **max viscosity ratio** = 5

Examples



F. Pelusi, A. Scagliarini, M. Sbragaglia, M. Bernaschi and R. Benzi,
"Intermittent thermal convection in jammed emulsions", *arXiv* (2024)

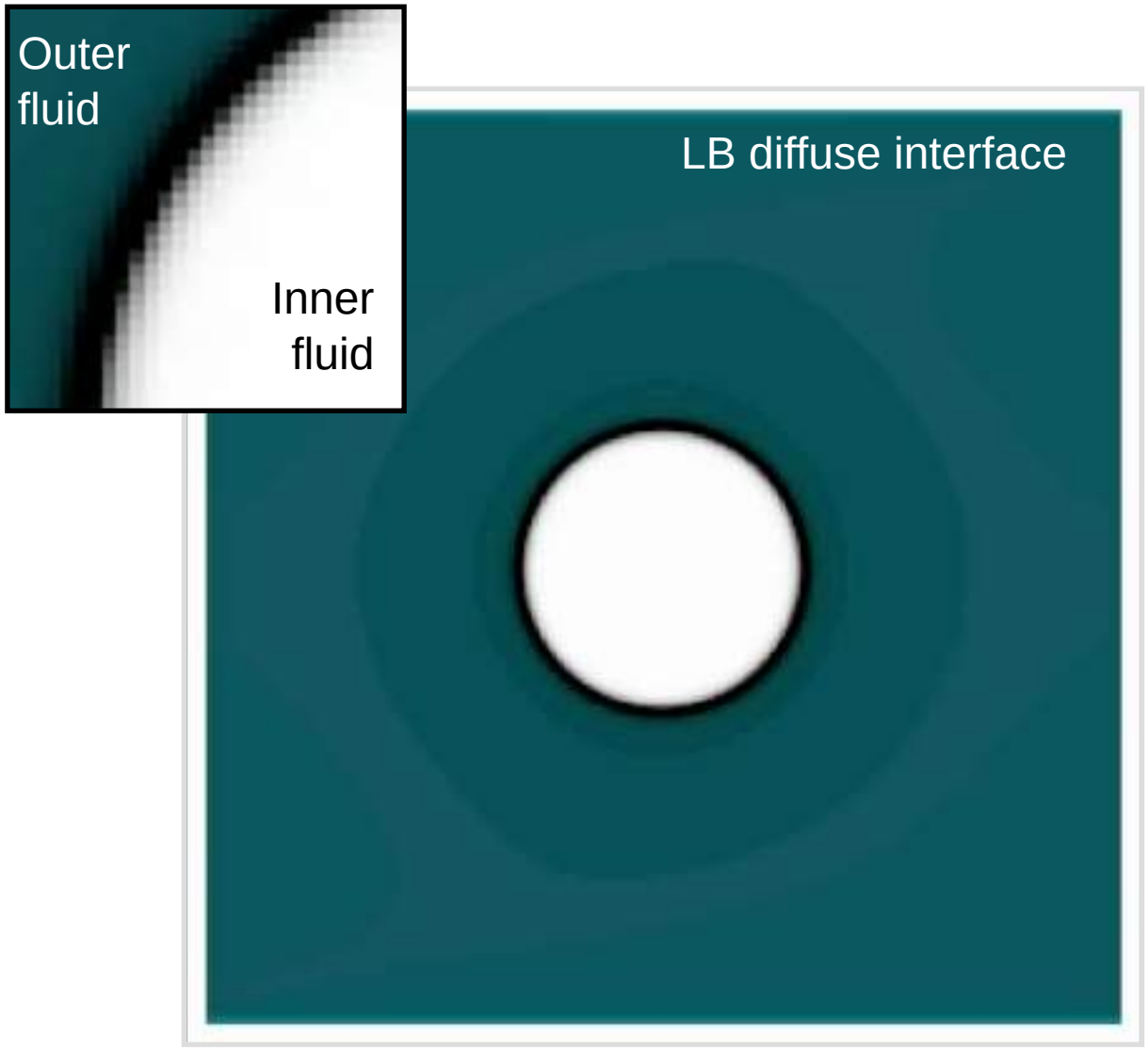


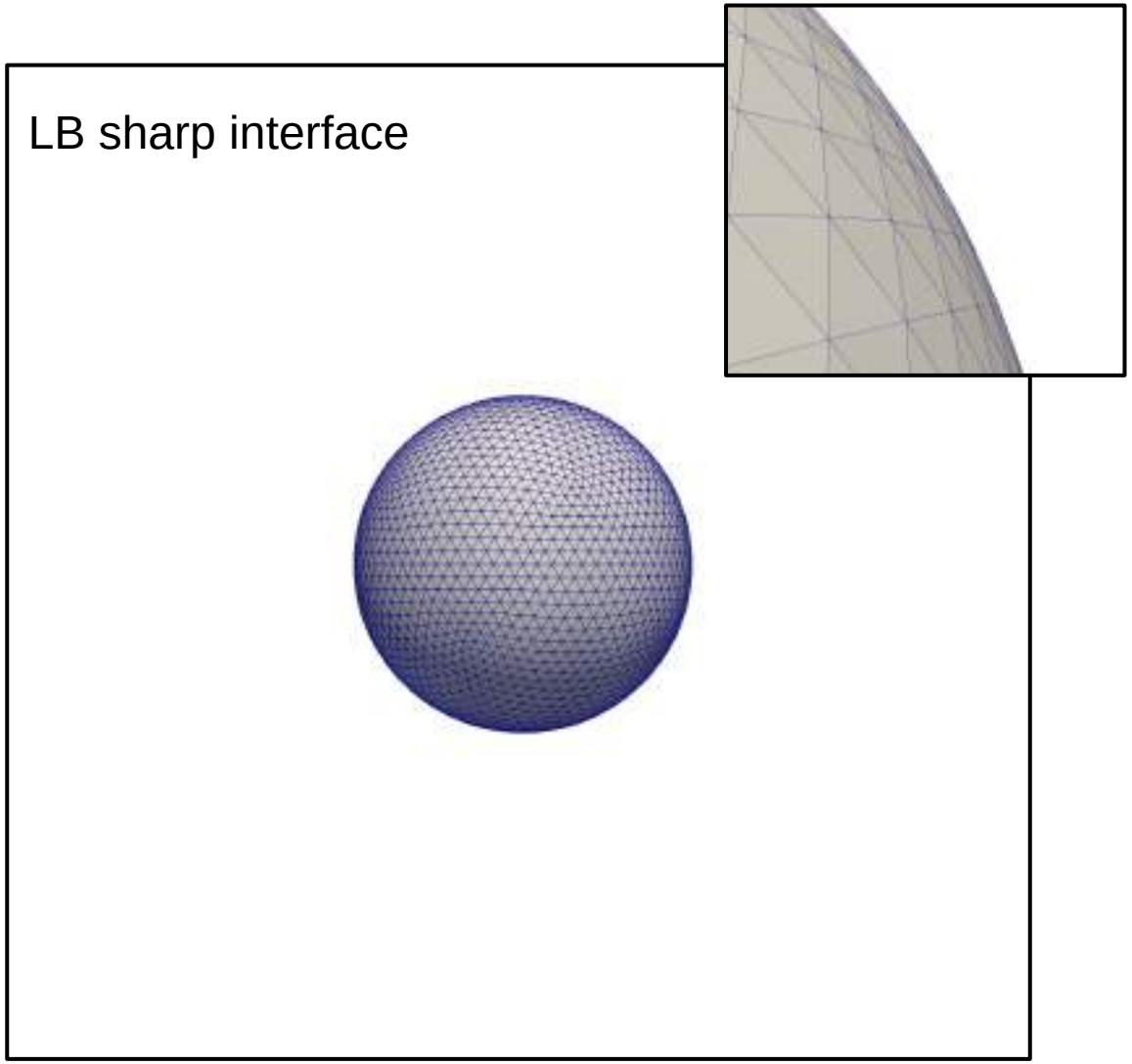
Sharp-interface approach

Immersed boundary - Lattice Boltzmann method (IB-LBM)

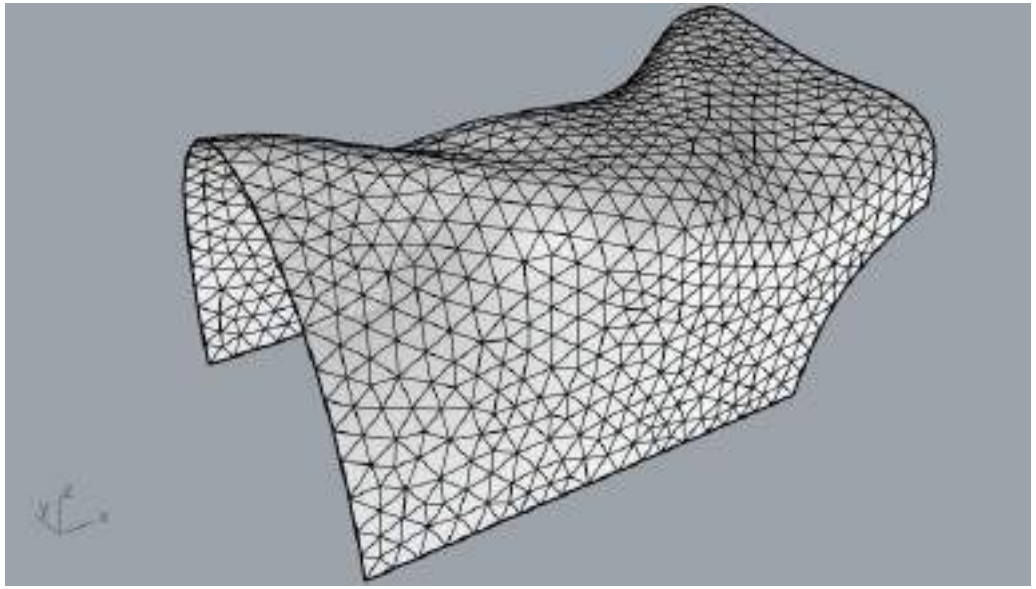
Fabio Guglietta

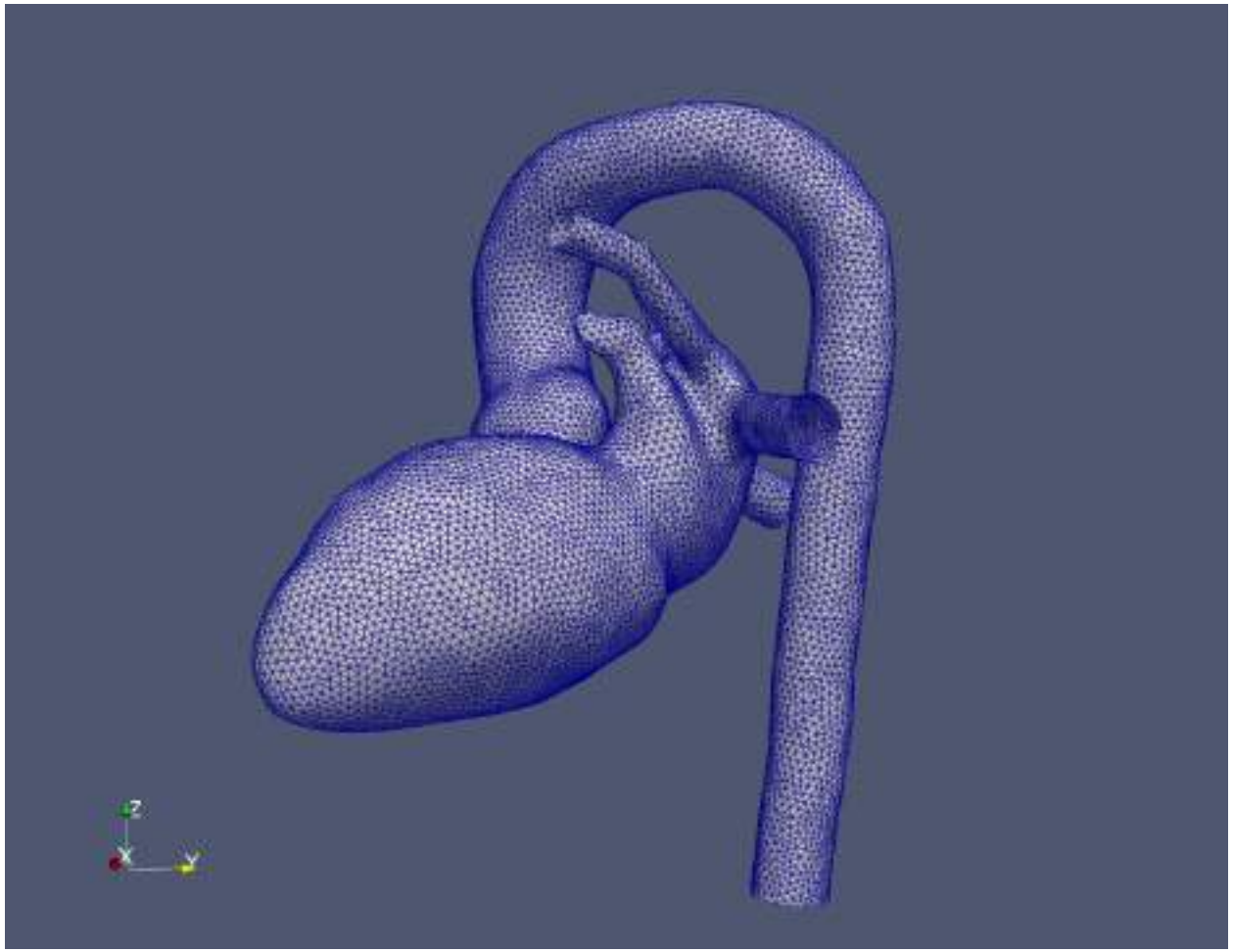
LBM: Diffuse vs sharp interface



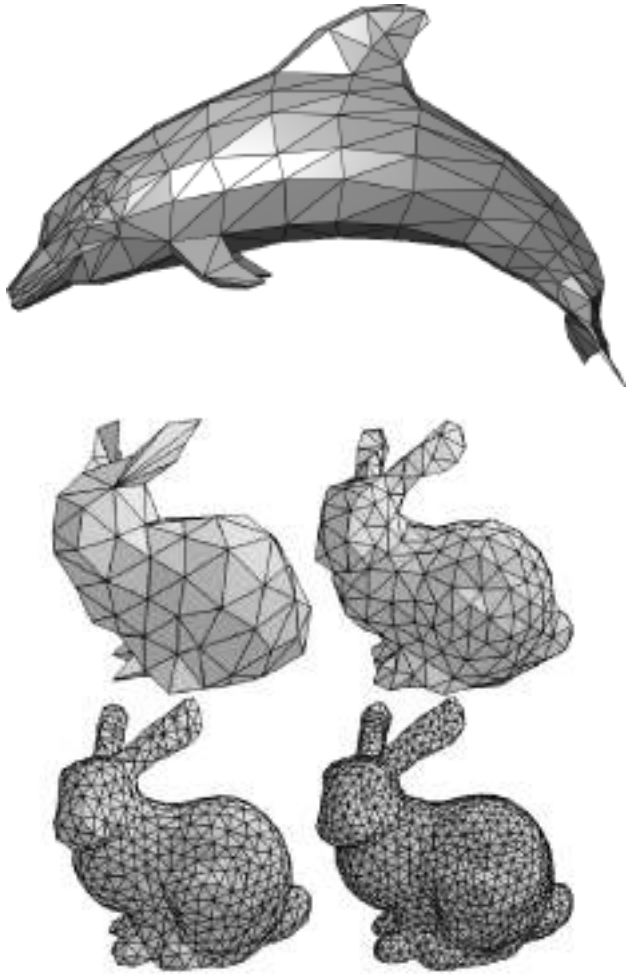


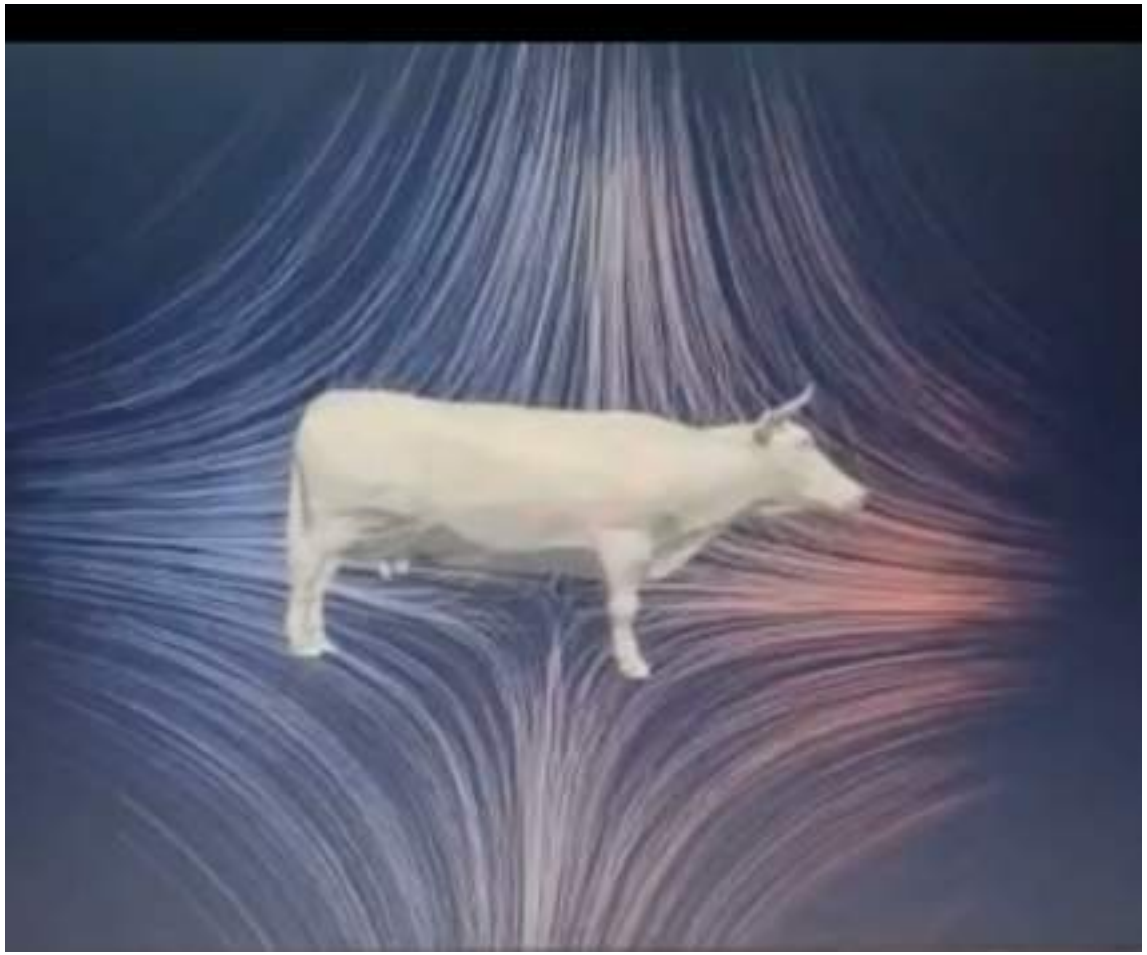
Any kind of triangular mesh



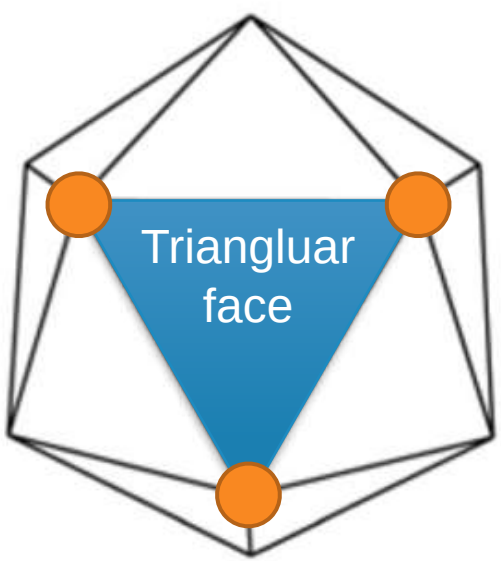


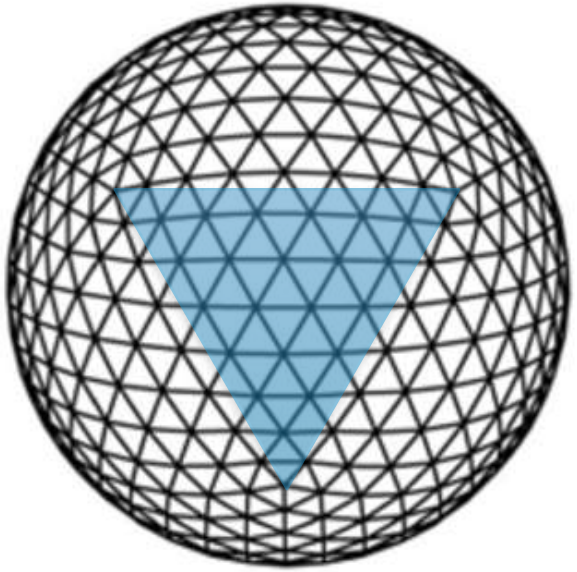
Seriously: any kind





Icosahedral sphere

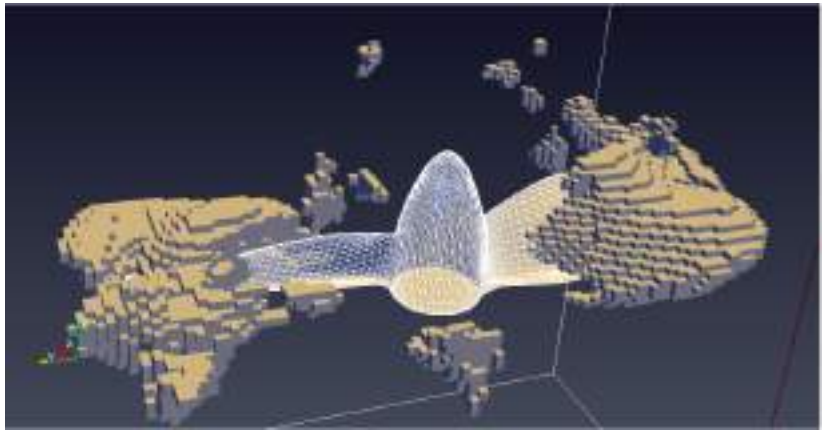




Lagrangian nodes

Mesh refinement

Curved and deformable boundaries



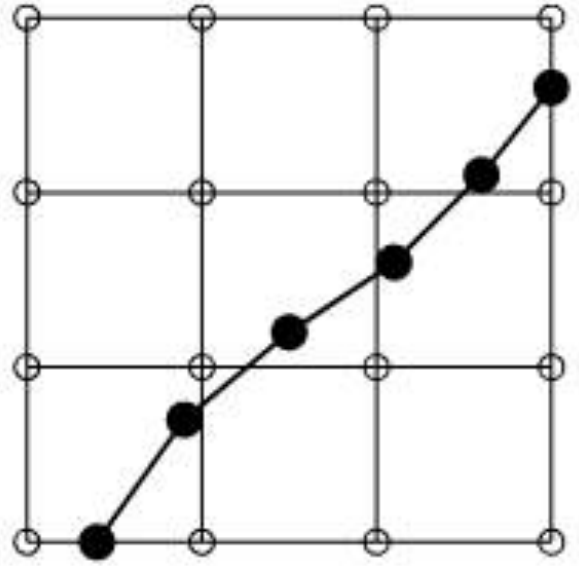
- Most **boundaries** in reality are **curved**:
  - Porous media
  - Curved surfaces of cars and planes in aerodynamics
  - Suspensions (e.g. clay, slurries)
  - Deformable objects (e.g. cells, wings, compliant containers)

- Three main categories:
  - **Stationary rigid** obstacles (e.g. porous media, microfluidic devices, flow over stationary cylinder)
  - **Moving rigid** obstacles (e.g. suspension non-deformable particles, rotating turbine blades)
  - **Moving deformable** obstacles (e.g. flexible wings, living cells, compliant channels)

- **Analytical solutions** are often impossible to obtain $\Longrightarrow$ **computer simulations**

Immersed boundary method (IBM)

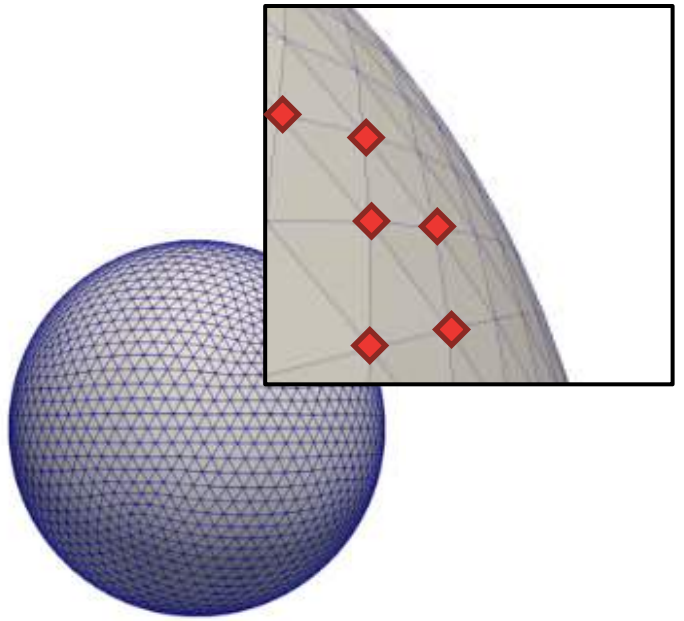
- **Immersed boundary** method (IBM): Peskin, 1972 [1,2]
- **Immersed boundary - lattice Boltzmann** method (IB-LBM): Feng & Michaelides, 2004 [3]
- **LBM:** boundary conditions act at **level of populations**
- **IBM**: use a force density $\mathbf{F}(\mathbf{x}, t)$ at the **level of NSE** to mimic boundary conditions
- Main **advantage** of IBM: boundary **shape known** (it does not have to be reconstructed)

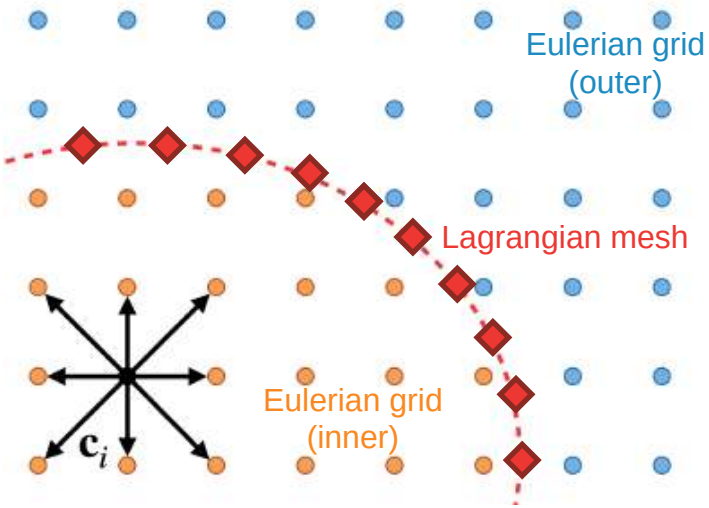


[1] Peskin, C. S. (1972). Flow patterns around heart valves: a digital computer method for solving the equations of motion. Yeshiva University.
[2] Peskin, C. S. (2002). The immersed boundary method. Acta numerica, 11, 479-517.
[3] Feng, Z. G., & Michaelides, E. E. (2004). The immersed boundary-lattice Boltzmann method for solving fluid–particles interaction problems. Journal of computational physics, 195(2), 602-628.

IBM: Mathematical basis

- Eulerian grid and Lagrangian mesh
- Set of Lagrangian nodes $\{\mathbf{r}_j\}$ with $\mathbf{r}_j = \mathbf{r}_j(t)$
- Shape of the Lagrangian mesh independent on the Eulerian grid
- **No-slip** boundary condition:
  - $\dot{\mathbf{r}}(t) = \mathbf{u}(\mathbf{r}(t), t)$
  - First governing equation: $\dot{\mathbf{r}}(t) = \int d\mathbf{x}\ \mathbf{u}(\mathbf{x}, t)\delta(\mathbf{x} - \mathbf{r}(t))$
- **Momentum exchange** (fluid-boundary):
  - force density (per area) on the boundary $\mathbf{F}_A(\mathbf{r}(t), t)$
  - Second governing equation: $\mathbf{F}(\mathbf{x}, t) = \int d^2r\ \mathbf{F}_A(\mathbf{r}(t), t)\delta(\mathbf{x} - \mathbf{r}(t))$
- Velocity field $\mathbf{u}$ known at discrete lattice sites and density force known on discrete Lagrangian nodes.





IBM: discretised governing equations

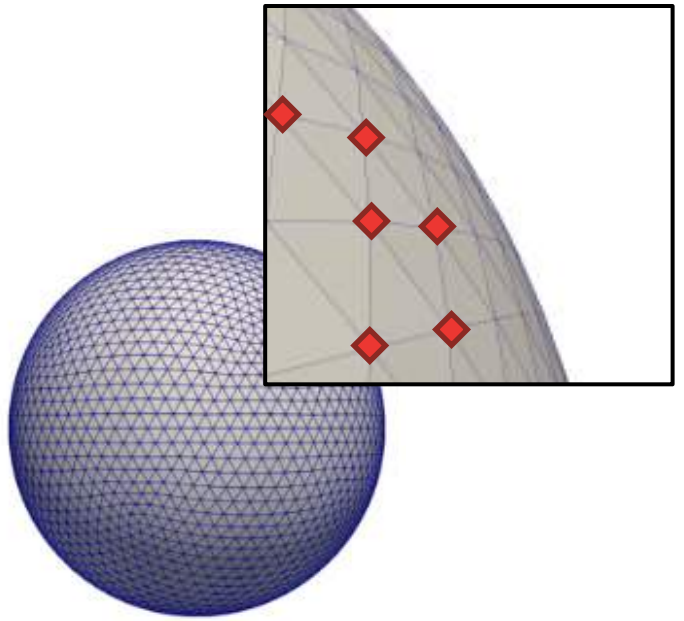
First governing equation (velocity interpolation)

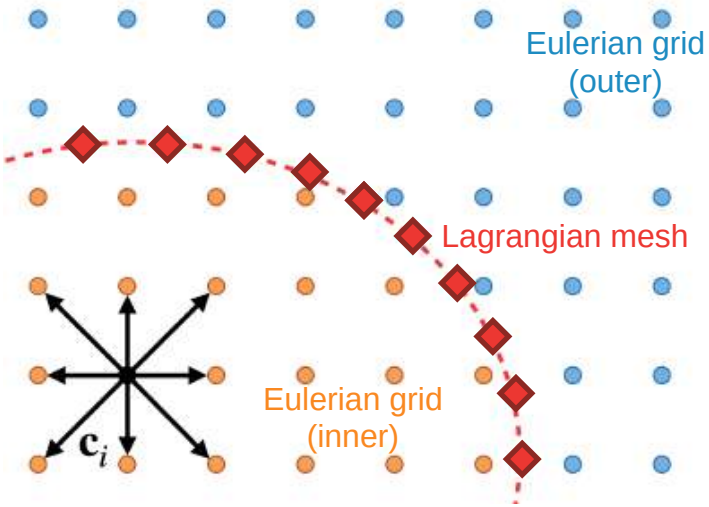
$$\dot{\mathbf{r}}(t) = \int d\mathbf{x}\ \mathbf{u}(\mathbf{x},t)\delta(\mathbf{x}-\mathbf{r}(t)) \quad \Rightarrow \quad \dot{\mathbf{r}}_j(t) = \sum_{\mathbf{x}} \Delta x^3 \mathbf{u}(\mathbf{x},t)\Delta(\mathbf{r}_j(t),\mathbf{x})$$

Second governing equation (force spreading)

$$\mathbf{F}(\mathbf{x},t) = \int d^2r\ \mathbf{F}_A(\mathbf{r}(t),t)\delta(\mathbf{x}-\mathbf{r}(t)) \quad \Rightarrow \quad \mathbf{F}(\mathbf{x},t) = \sum_j \varphi_j(t)\Delta(\mathbf{r}_j(t),\mathbf{x})$$

Total force acting on node $\mathbf{r}_j(t)$





- $\Delta(\mathbf{r}_j,\mathbf{x})$ is a discretised version of Dirac delta distribution
- Assumption: $\Delta(\mathbf{r}_j,\mathbf{x}) = \Delta(\mathbf{r}_j - \mathbf{x})$
- Factorise (not essential): $\Delta(\mathbf{x}) = \phi(x)\phi(y)\phi(z)/\Delta x^3$

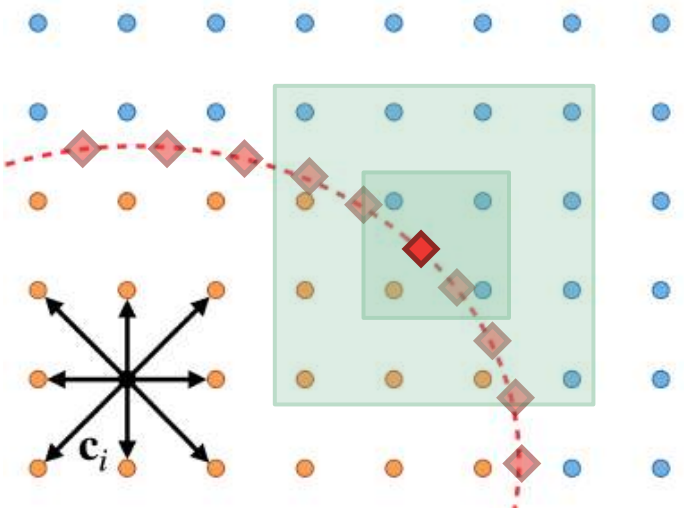
IBM: discretised governing equations

$$\Delta(\mathbf{x}) = \phi(x)\phi(y)\phi(z)/\Delta x^3$$

$$\phi_2(x) = \begin{cases} 1 - |x| & (0 \le |x| \le \Delta x) \\ 0 & (\Delta x \le |x|) \end{cases},$$

$$\phi_3(x) = \begin{cases} \frac{1}{3}\left(1 + \sqrt{1 - 3x^2}\right) & 0 \le |x| \le \frac{1}{2}\Delta x \\ \frac{1}{6}\left(5 - 3|x| - \sqrt{-2 + 6|x| - 3x^2}\right) & \frac{1}{2}\Delta x \le |x| \le \frac{3}{2}\Delta x \\ 0 & \frac{3}{2}\Delta x \le |x| \end{cases},$$

$$\phi_4(x) = \begin{cases} \frac{1}{8}\left(3 - 2|x| + \sqrt{1 + 4|x| - 4x^2}\right) & 0 \le |x| \le \Delta x \\ \frac{1}{8}\left(5 - 2|x| - \sqrt{-7 + 12|x| - 4x^2}\right) & \Delta x \le |x| \le 2\Delta x \\ 0 & 2\Delta x \le |x| \end{cases}.$$



IB-LBM algorithm

1. Compute Lagrangian forces $\varphi_j(t)$ (model dependent: typically, depends on the geometry)
2. IB: **Spread Lagrangian forces** to the Eulerian grid: $\mathbf{F}(\mathbf{x},t) = \sum_j \varphi_j(t)\Delta(\mathbf{r}_j(t),\mathbf{x})$
3. **LB step**: compute equilibrium + collision + streaming
4. Compute physical **fluid velocity** with half-force correction: $\rho\mathbf{u}_f = \sum_i \mathbf{c}_i f_i + \mathbf{F}(\mathbf{x},t)\Delta t/2$
5. IB: **Interpolate fluid velocity $\mathbf{u}_f$** at the Lagrangian positions: $\dot{\mathbf{r}}_j(t) = \sum_{\mathbf{x}} \Delta x^3 \mathbf{u}(\mathbf{x},t)\Delta(\mathbf{r}_j(t),\mathbf{x})$
6. **Advect the Lagrangian nodes** (e.g., forward Euler): $\mathbf{r}_j(t+\Delta t) = \mathbf{r}_j(t) + \dot{\mathbf{r}}_j(t)\Delta t$

Immersed Boundary - Lattice Boltzmann Method

TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

LBM

Lattice Boltzmann Equation

$$f_i(\mathbf{x}+\mathbf{c}_i\Delta t, t+\Delta t) - f_i(\mathbf{x},t) = \Delta t\left[\Omega_i(\mathbf{x},t) + \left(1-\frac{1}{2\tau}\right)F_i(\mathbf{x},t)\right]$$

Fluid Density $\rho(\mathbf{x},t) = \sum_i f_i(\mathbf{x},t)$

Fluid Velocity $\rho(\mathbf{x},t)\mathbf{u}(\mathbf{x},t) = \sum_i \mathbf{c}_i\, f_i(\mathbf{x},t)$

BGK collision operator $\Omega_i^{BGK} = -\frac{1}{\tau}\left(f_i - f_i^{eq}\right)$

Forcing $F_i = w_i\left(\frac{\mathbf{c}_i-\mathbf{u}}{c_s^2} + \frac{(\mathbf{c}_i\cdot\mathbf{u})\mathbf{c}_i}{c_s^4}\right)\cdot\mathbf{F}$

Equilibrium distribution function

$$f_i^{eq}(\mathbf{x},t) = w_i\rho\left(1+\frac{\mathbf{u}\cdot\mathbf{c}_i}{c_s^2} + \frac{(\mathbf{u}\cdot\mathbf{c}_i)^2}{2c_s^4} - \frac{\mathbf{u}\cdot\mathbf{u}}{2c_s^2}\right)$$

IBM

Discrete Dirac Delta: $\Delta(\mathbf{x}) = \frac{\Phi(x)\Phi(y)\Phi(z)}{\Delta x^3}$

$$\Phi_4(x) = \begin{cases} \frac{1}{8}\left(3-2|x|+\sqrt{1+4|x|-4x^2}\right) & 0 \le |x| \\ \frac{1}{8}\left(5-2|x|-\sqrt{-7+12|x|-4x^2}\right) & \Delta x \le |x| \le 2\Delta x \\ 0 & 2\Delta x \le |x| \end{cases}$$

membrane -> fluid

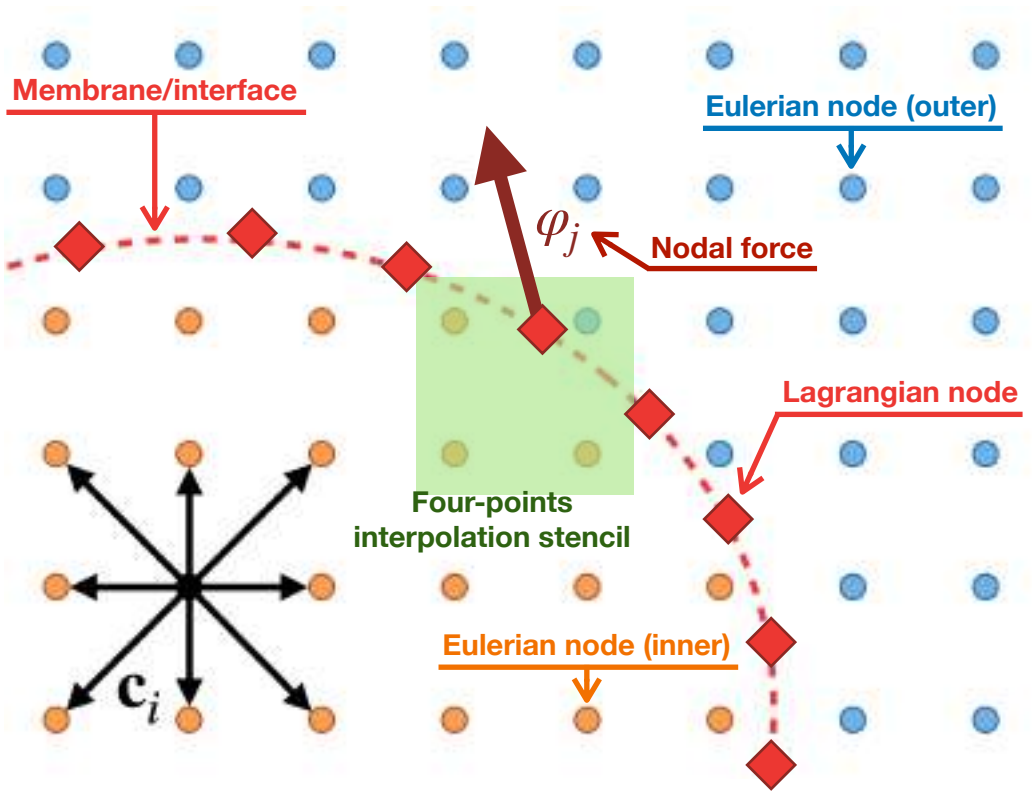
$$\mathbf{F}(\mathbf{x},t) = \sum_j \varphi_j(t)\Delta(\mathbf{r}_j(t),\mathbf{x}) + \mathbf{F}_{other}$$

fluid -> membrane

$$\dot{\mathbf{r}}_j(t) = \sum_{\mathbf{x}} \Delta x^3 \mathbf{u}(\mathbf{x},t)\Delta(\mathbf{r}_j(t),\mathbf{x})$$

$$\mathbf{r}_j(t+\Delta t) = \mathbf{r}_j(t) + \dot{\mathbf{r}}_j(t)$$

T. Krüger et al. "The lattice Boltzmann method", *Springer International Publishing*, 2017



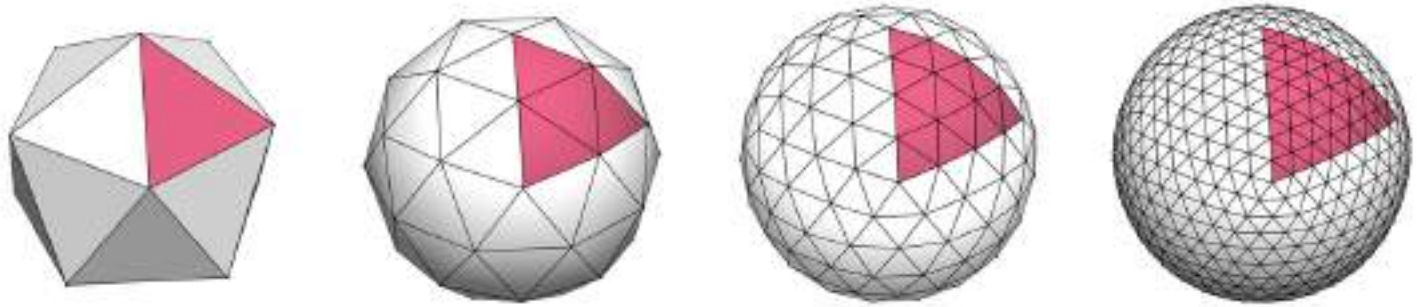
Hoshen-Kopelman algorithm:

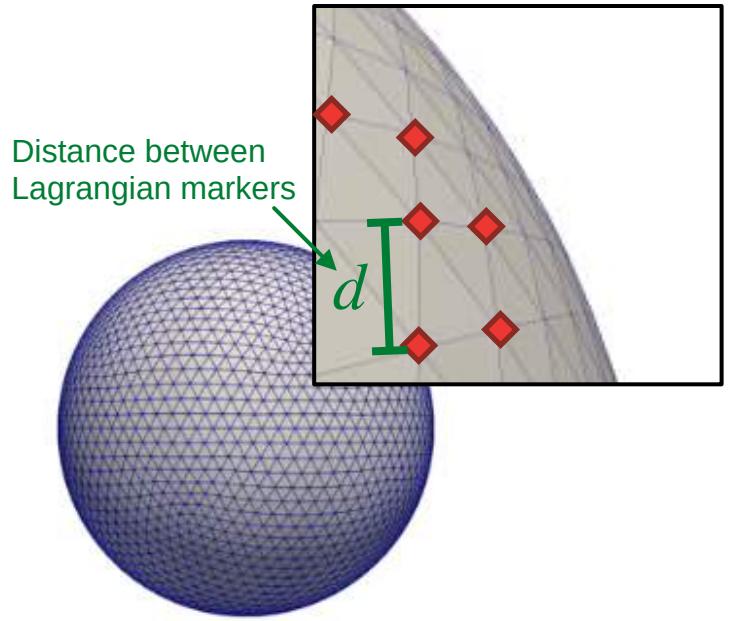
To recognise which lattice sites are inside/ outside the membrane

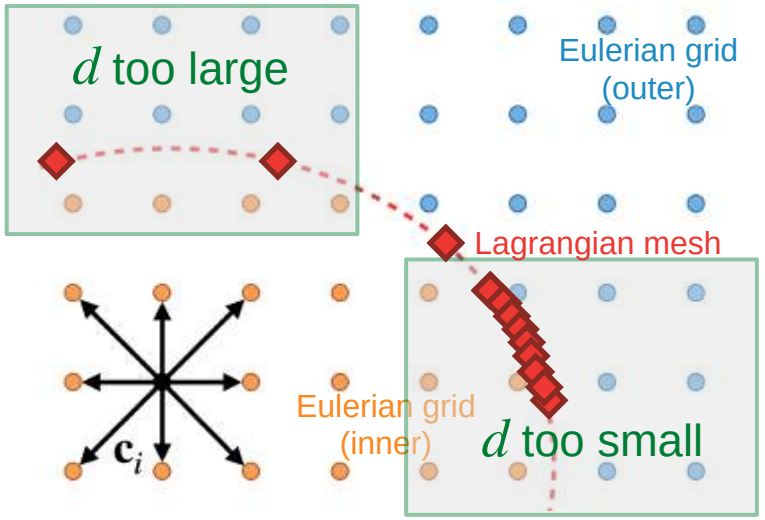
S. Frijters, T. Krüger, J. Harting, "Parallelised Hoshen–Kopelman algorithm for lattice-Boltzmann simulations", *Computer Physics Communications* 189, 92-98, 2015

Distribution of Lagrangian markers

- d too large: holes (fluid can flow between markers)
- d too small (i.e., $d \ll \Delta x$): the markers see the same fluid environment and move with same velocity
- Ideal values: $d \in [0.5,1]\Delta x$





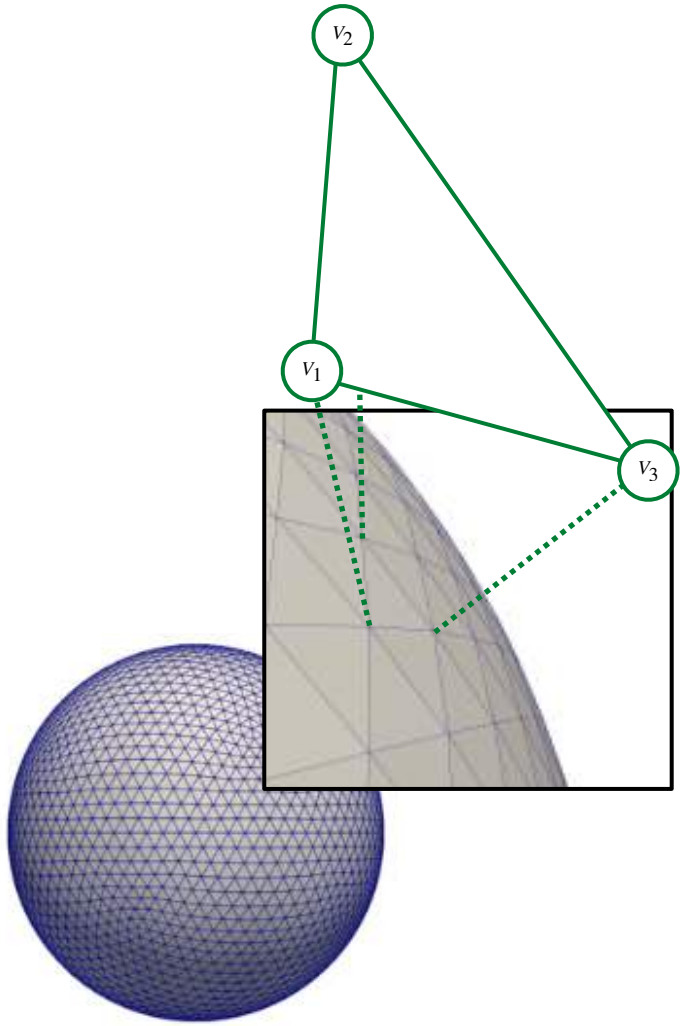


Constitutive models

- They contain all the physics of the boundary deformation

- Hyperelastic models: $\varphi_j(t) = \varphi_j(\{\mathbf{r}(t)\})$
- Viscoelastic models: $\varphi_j(t) = \varphi_j(\{\mathbf{r}(t)\}, \{\dot{\mathbf{r}}(t)\})$

- Define an energy density on each triangle $w = w(\{V_1, V_2, V_3\})$
- Compute the force as $\varphi_j = -\dfrac{\partial w}{\partial \mathbf{V}_j}$



RBC Membrane elasticity - Continuum description

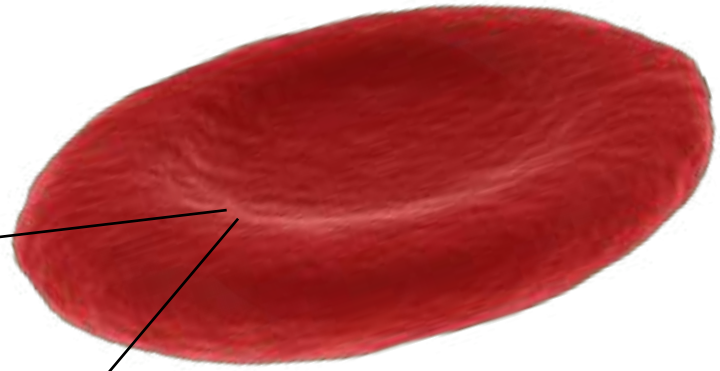
Strain energy (Skalak model):

$$W_S = \frac{1}{12}\int dA \left[\kappa_s \left(I_1^2 + 2I_1 - 2I_2\right) + \kappa_\alpha I_2^2\right]$$

Strain invariants

$$I_1 = \lambda_x^2 + \lambda_y^2 - 2$$
$$I_2 = \lambda_x^2 \lambda_y^2 - 1$$

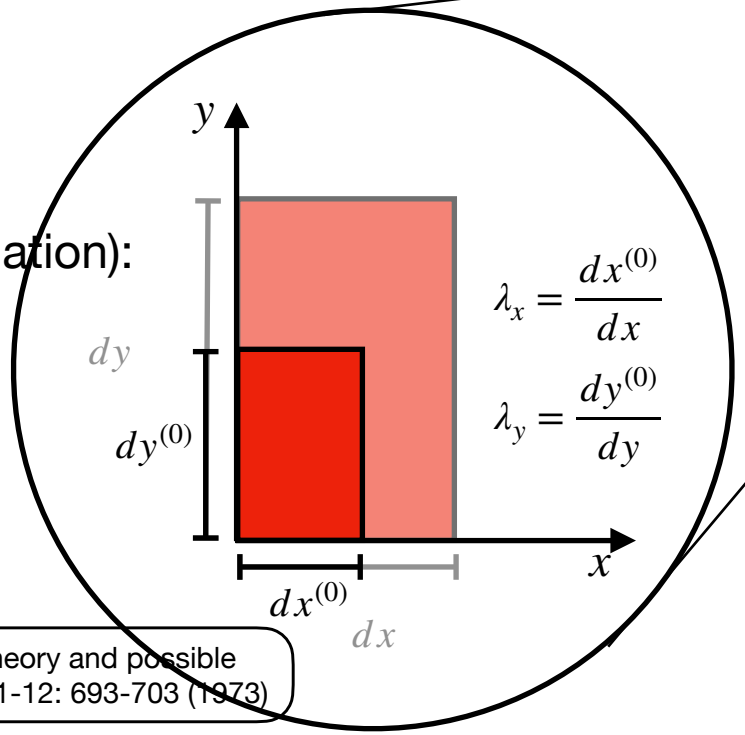
R. Skalak, et al. "Strain energy function of red blood cell membranes." *Biophysical journal* 13.3: 245-264 (1973)



Bending energy (Helfrich formulation):

$$W_B = \frac{\kappa_B}{2}\int dA \left(H - H^{(0)}\right)$$

Trace of the surface curvature tensor



W. Helfrich "Elastic properties of lipid bilayers: theory and possible experiments." *Zeitschrift für Naturforschung C* 28.11-12: 693-703 (1973)

RBC PARAMETERS

Parameter	Value
Radius r	3.91 μm
Area A	133 μm^2
Volume V	93 μm^3
Elastic shear modulus k_S	5.3 μNm^{-1}
Elastic dilatational modulus k_α	50 k_s
Bending modulus k_B	2 10^{-19}Nm
Plasma viscosity μ_{out}	1.2 10^{-3} Pa s
Cytoplasm viscosity μ_{in}	6 10^{-3} Pa s

- **Continuum description** works at the **scale of RBCs.**
- **Skalak** and **Helfrich** models are **validated** and explain **experimental evidences** at the RBC scales.
- **Elastic coefficients** (k_s, k_α and k_B) are experimentally measured with **very high accuracy**.

Other constitutive laws

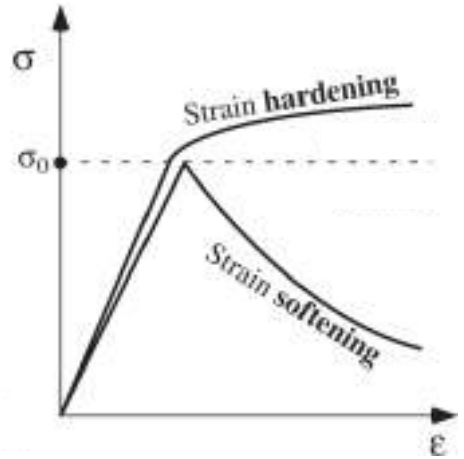
- Neo-Hookean: $w = \frac{G_s}{2}\left(I_1 - 1 + \frac{1}{I_2 + 1}\right)$
  - Volume-incompressible, rubber-like materials
  - It cannot model an area-incompressible membrane

$$W = \int dA\ w(I_1, I_2)$$

$$W = \sum_{i \in \text{triangles}} w(I_1^{(i)}, I_2^{(i)}) A_i$$

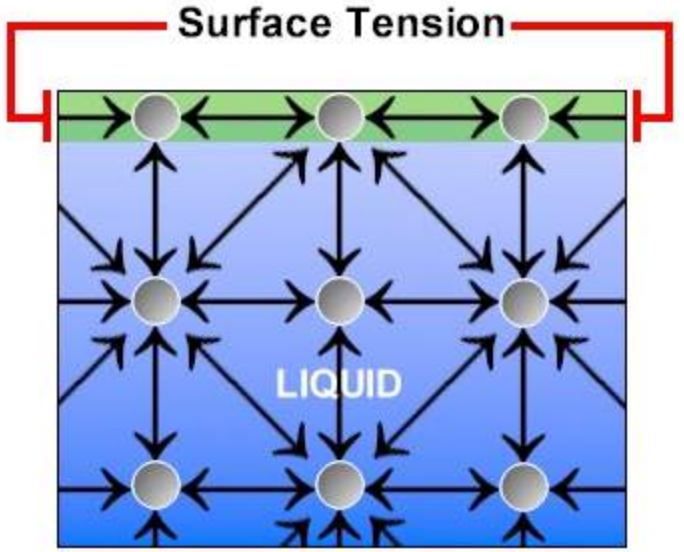
- Evans-Skalak: $w = G_s\left(\frac{I_1 + 2}{2\sqrt{I_2 + 1}} - 1 + \frac{\alpha}{2}\left(I_2 + 2 - 2\sqrt{I_2 + 1}\right)\right)$ with parameter $\alpha > 0$
  - It describes an almost area-incompressible material (appropriate for biological membranes)

- For small deformation, all laws are equivalent.
- For large deformation, they lead to different nonlinear tension-strain relations
  - Neo-Hookean and Evans-Skalak are strain softening under uniaxial stretching
  - Skalak is strain hardening

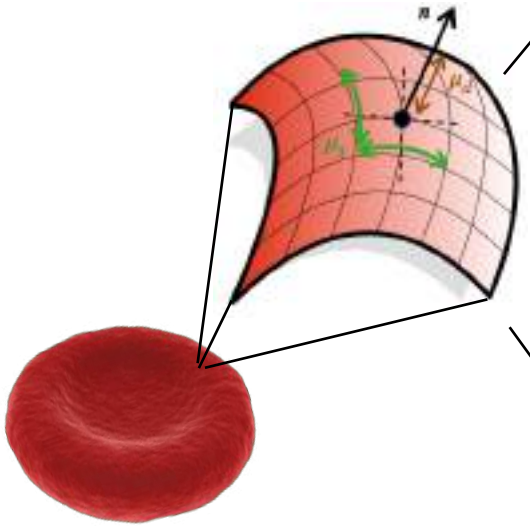


Surface tension

- On each mesh triangle, we can compute:
  - Stress tensor: $\tau = \gamma \mathbf{I}_2$ (the force is retrieved via finite element methods)
  - Energy density: $w = \frac{\gamma}{2} \log(I_2 + 1) \left[1 + \frac{1}{4} \log(I_2 + 1) \right]$



Interface viscosity



Strain tensor $\mathbf{E} = \frac{1}{2}[\nabla \mathbf{u} + \nabla \mathbf{u}^T]$

$$\tau^v = \tau_s^v + \tau_d^v = \mu_s \left[2\mathbf{E} - \mathrm{tr}(\mathbf{E})\mathbf{I}\right] + \mu_d \mathrm{tr}(\mathbf{E})\mathbf{I} = 2\mu_m \mathbf{E}$$

Shear membrane viscosity

Dilation membrane viscosity

L. Scriven "Dynamics of a fluid interface - equation of motion for newtonian surface fluids", Chemical Engineering Science 12,98 (1960)

Volume conservation

- IBM velocity interpolation does not maintain the solenoidal properties of the fluid
  $\Longrightarrow$ Volume of an enclosed region can change in time

- Enforce volume conservation:

$$\boldsymbol{F}_i^{\mathrm{V}} = -\frac{\partial E_{\mathrm{V}}(\{\boldsymbol{x}_i\})}{\partial \boldsymbol{x}_i} \qquad E_{\mathrm{V}} = \frac{\kappa_{\mathrm{V}}}{2}\frac{\left(V - V^{(0)}\right)^2}{V^{(0)}}$$

AQTIVATE

European Research Council
Established by the European Commission
Supporting top researchers from anywhere in the world

TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Immersed Boundary - Lattice Boltzmann Simulations

Red Blood Cells, Drops, Capsules

Fabio Guglietta

What did we do with IB-LBM?

- **Single RBCs**: how IB-LB simulations can improve our understanding [1, 2, 3]
- **Suspension** of spherical **capsules**: non-Newtonian behaviour [4]
- **Wetting dynamics** with sharp interface approaches: why? [5]
- **Dynamics of drops** with sharp interface approaches:
  - Effect of **interface viscosity** on **confined** drops [6]
  - **Reduced order model** for drop deformation in **stationary flows** [7]
  - Dynamics of drops in **turbulent flows** [8]

[1] **F. Guglietta**, M. Behr, L. Biferale, G. Falcucci and M. Sbragaglia, “On the effects of membrane viscosity on transient red blood cell dynamics”, Soft matter, 16(26), 6191-6205 (2020)
[2] **F. Guglietta**, M. Behr, G. Falcucci and M. Sbragaglia, “Loading and relaxation dynamics of a red blood cell”, Soft matter, 17(24), 5978-5990 (2021)
[3] **F. Guglietta**, M. Behr, L. Biferale, G. Falcucci and M. Sbragaglia, “Lattice Boltzmann simulations on the tumbling to tank-treading transition: effects of membrane viscosity”, Philosophical Transactions of the Royal Society A, 379(2208), 20200395 (2021)
[4] **F. Guglietta,** F. Pelusi, M. Sega, O. Aouane and J. Harting, “Suspensions of viscoelastic capsules: Effect of membrane viscosity on transient dynamics”, Journal of Fluid Mechanics, 971, A13 (2023)
[5] F. Pelusi, **F. Guglietta**, M. Sega, O. Aouane & J. Harting “A sharp interface approach for wetting dynamics of coated droplets and soft particles”, Physics of Fluids, 35(8), (2023)
[6] **F. Guglietta**, F. Pelusi “A unified analytical prediction for steady-state behavior of confined drop with interface viscosity under shear flow”, *in peer review*, (2024)
[7] D. Taglienti, **F. Guglietta** & M. Sbragaglia “Reduced model for droplet dynamics in shear flows at finite capillary numbers”, Physical Review Fluids, 8(1), 013603, (2023)
[8] D. Taglienti, **F. Guglietta** & M. Sbragaglia “Droplet dynamics in homogeneous isotropic turbulence with the immersed boundary-lattice Boltzmann method”, *in peer review, (2024)*

TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Red Blood Cell (RBC) simulations with IB-LBM

[1] **F. Guglietta**, M. Behr, L. Biferale, G. Falcucci and M. Sbragaglia, “On the effects of membrane viscosity on transient red blood cell dynamics”, Soft matter, 16(26), 6191-6205 (2020)
[2] **F. Guglietta**, M. Behr, G. Falcucci and M. Sbragaglia, “Loading and relaxation dynamics of a red blood cell”, Soft matter, 17(24), 5978-5990 (2021)
[3] **F. Guglietta**, M. Behr, L. Biferale, G. Falcucci and M. Sbragaglia, “Lattice Boltzmann simulations on the tumbling to tank-treading transition: effects of membrane viscosity”, Philosophical Transactions of the Royal Society A, 379(2208), 20200395 (2021)

Dynamics of complex fluids

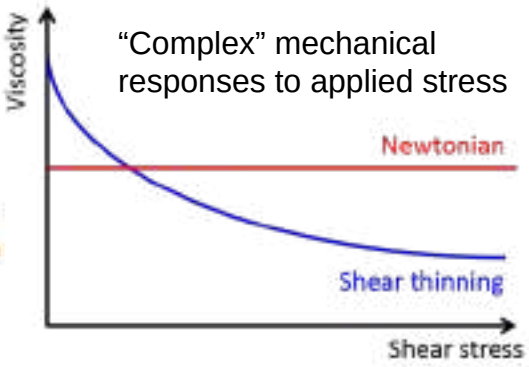
TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Liquid-Liquid

Liquid-Gas

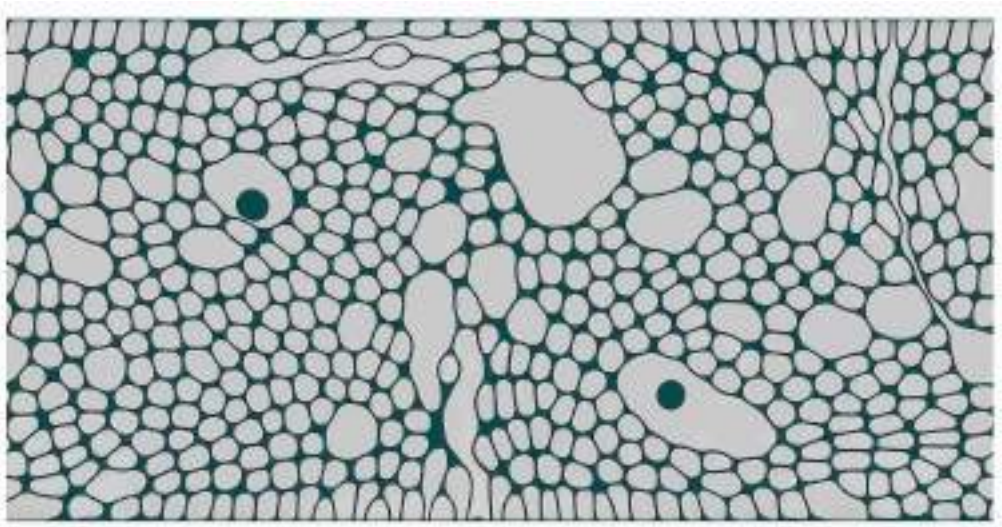
Solid-Gas

Solid-Liquid



"Complex" mechanical responses to applied stress

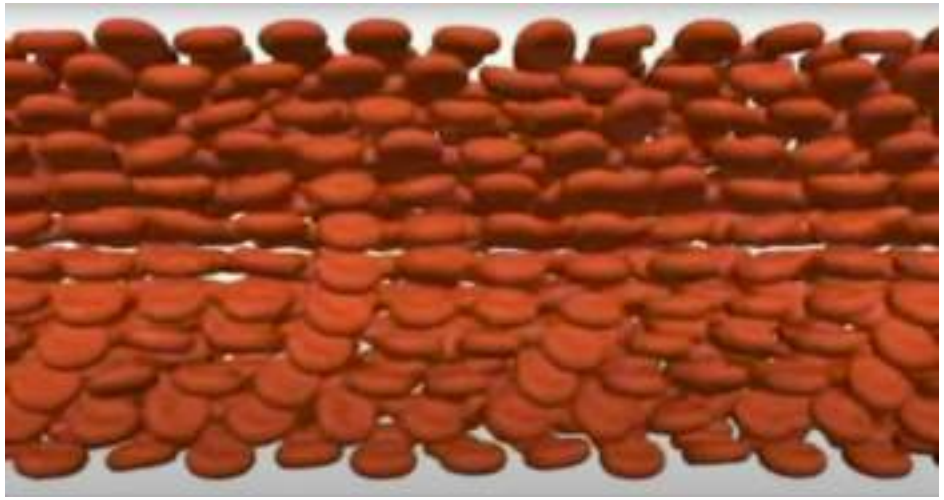
Emulsion



F. Pelusi, A. Scagliarini, M. Sbragaglia, M. Bernaschi and R. Benzi, "Intermittent thermal convection in jammed emulsions", *arXiv* (2024), in peer review

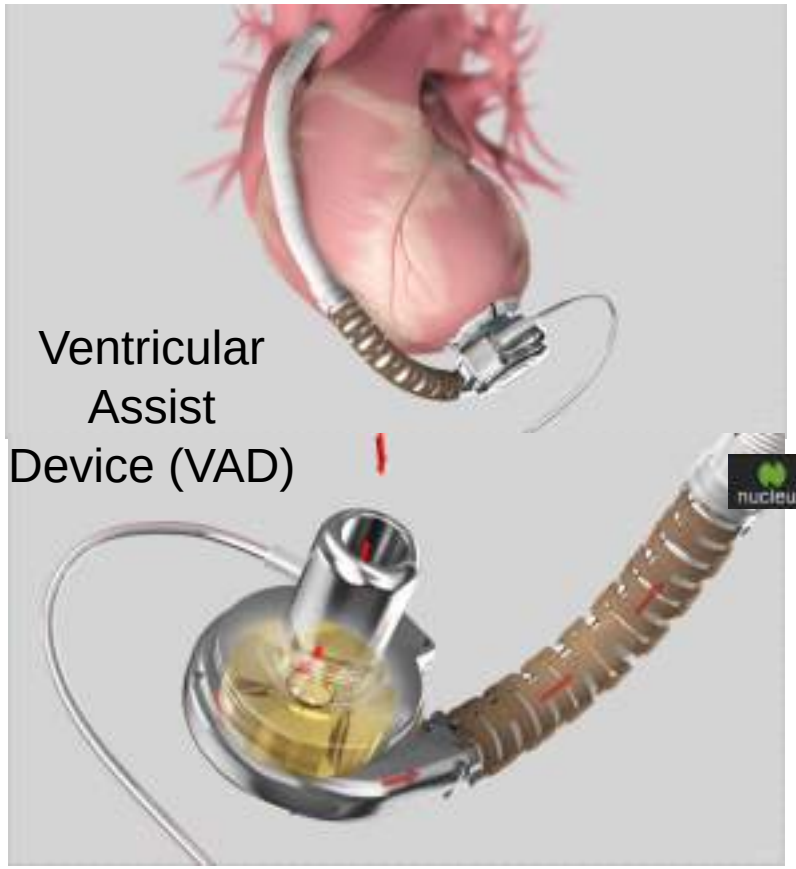
Lattice Boltzmann Simulations

Suspension



T. Krüger, F. Varnik and D. Raabe, "Particle stress in suspensions of soft objects", *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences,* 369(1945), 2414-2421 (2011)

From biomedical devices to a single red blood cell (RBC)



The **high shear rate** in the impeller can cause **high deformation** —> **hemolysis**.
(**rupture** of the RBC **membrane**)

To estimate hemolysis…

Ratio of plasma free hemoglobin to the total hemoglobin

Stress of the RBC

$$\frac{d}{dt}\left(\frac{\Delta Hb}{Hb}\right) \sim \sigma^{2.416}$$

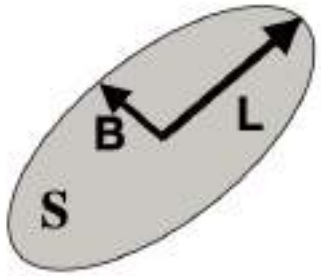
M. Giersiepen et al., "Estimation of shear stress-related blood damage in heart valve prostheses-in vitro comparison of 25 aortic valves." *The International journal of artificial organs* 13.5 (1990): 300-306.

… we must investigate RBC mechanical (viscoelastic) response

- RBC **deformation** (linked with hemolysis)
- RBC **characteristic times** (loading and relaxation)
- **Loading time** vs. **residence time** in the impeller

Simulate RBC dynamics - Reduced order models

- Maffettone and Minale model [1]:
  - RBC as a drop (shape given by tensor $\mathbf{S}$)
  - Shape always ellipsoidal
  - Two parameters (f_1 and f_2)
  - One-way coupling



Deformation:

$$D = \frac{L - B}{L + B}$$

$$\frac{\partial \mathbf{S}}{\partial t} - [\boldsymbol{\Omega} \cdot \mathbf{S} + \mathbf{S} \cdot \boldsymbol{\Omega}] = -f_1[\mathbf{S} - g(\mathbf{S})\mathbf{I}] + f_2[\mathbf{E} \cdot \mathbf{S} + \mathbf{S} \cdot \mathbf{E}]$$

Vorticity tensor

$$\boldsymbol{\Omega} = \frac{1}{2}[\nabla \mathbf{u} - \nabla \mathbf{u}^T]$$

Volume conservation

$$g(\mathbf{S}) = \frac{6 \det(\mathbf{S})}{\mathrm{tr}(\mathbf{S})^2 - \mathrm{tr}(\mathbf{S}^2)}$$

Strain tensor

$$\mathbf{E} = \frac{1}{2}[\nabla \mathbf{u} + \nabla \mathbf{u}^T]$$

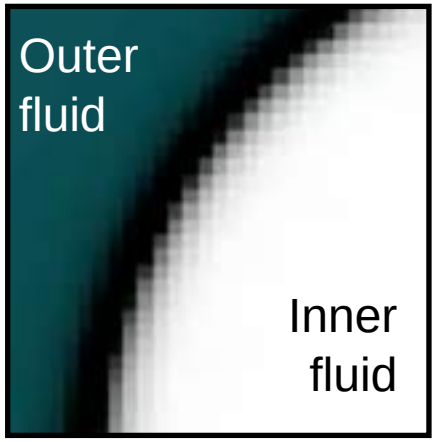
f_1 | Shape-recovery **(Transient-dynamics)**
Behaves like the **inverse of the relaxation time** of the membrane

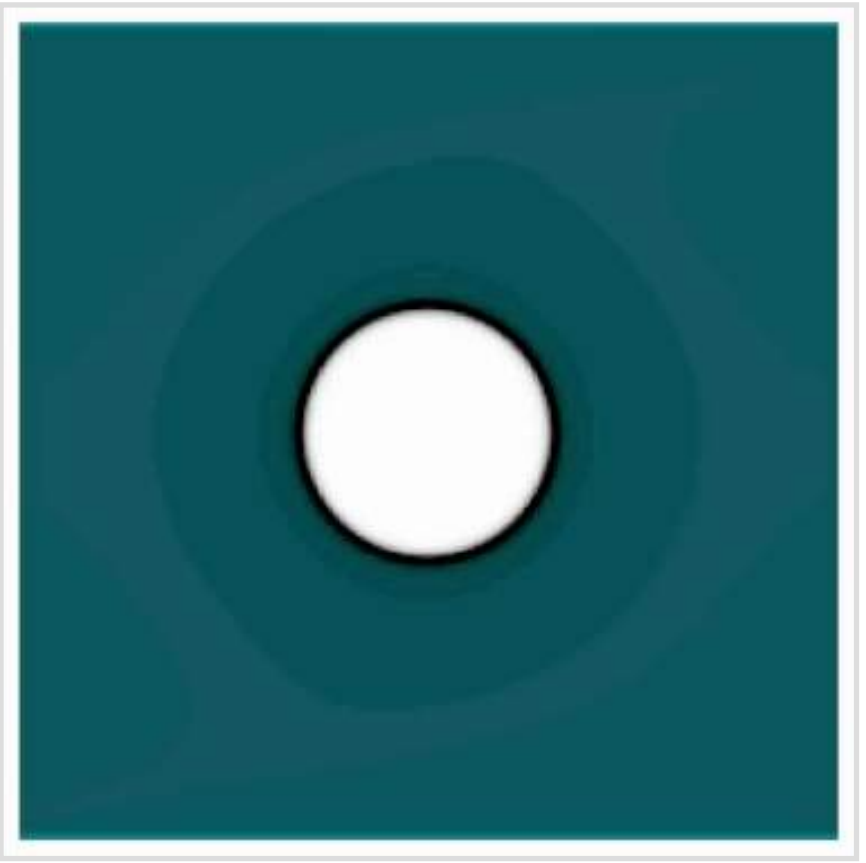
f_2 | Deformation
It is linked with the steady value of the deformation

[1] **P. L. Maffettone and M. Minale,** "Equation of change for ellipsoidal drops in viscous flow", *Journal of non-Newtonian fluid mechanics* 78.2-3, 227-241, 1998

Simulate RBC dynamics

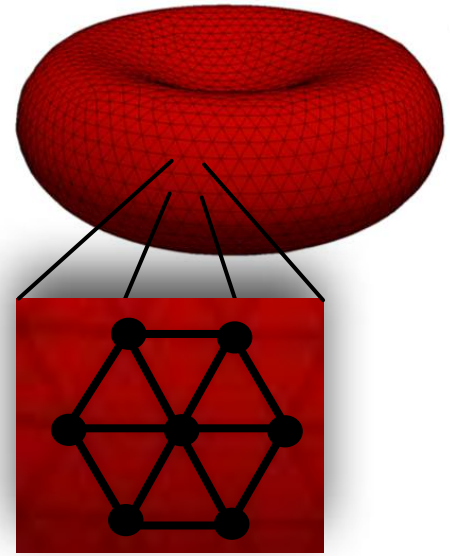
- **Lattice Boltzmann - Diffuse interface** numerical methods:
  - RBC as a drop
  - ~~Shape always ellipsoidal~~
  - ~~Two parameters (f_1 and f_2)~~
  - ~~One-way coupling~~
  - Simulate fluids inside and outside
  - Requires high resolution
  - Cannot control membrane viscoelastic properties (required to simulate RBC membrane)

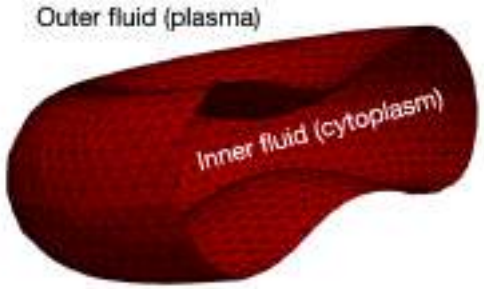




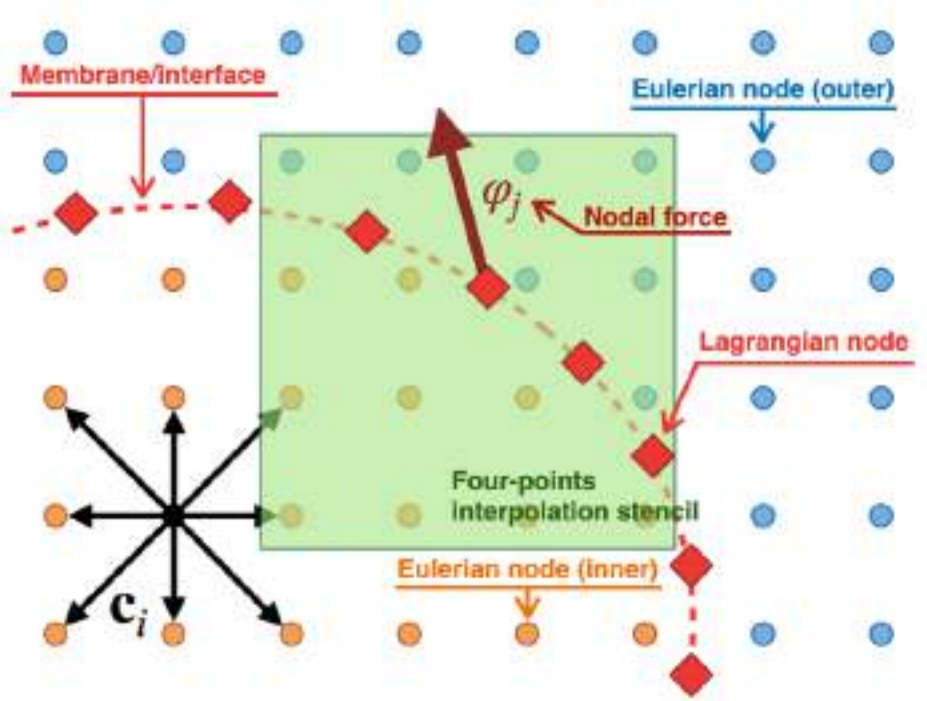
F. Pelusi, M. Lulli, M. Sbragaglia & M. Bernaschi, "TLBfind: a Thermal Lattice Boltzmann code for concentrated emulsions with FINite-size Droplets", *Computer physics communications*, **273**, 108259 (2022)

Simulate RBC dynamics





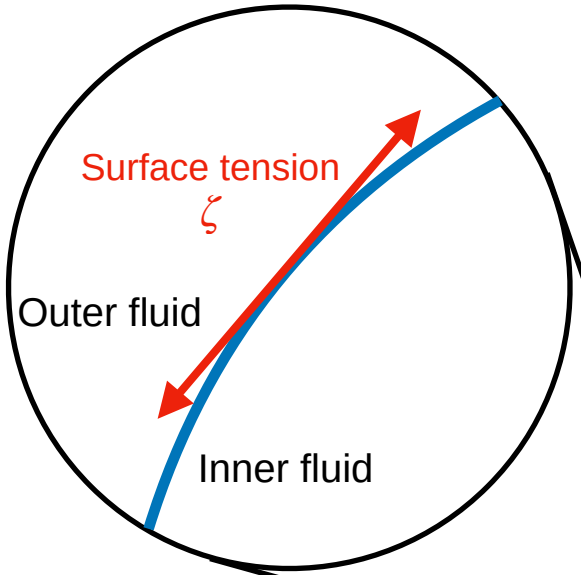
- **Immersed Boundary - Lattice Boltzmann method**:
  - ~~RBC as a drop~~
  - ~~Shape always ellipsoidal~~
  - ~~Two parameters~~ (f_1 and f_2)
  - ~~One-way coupling~~
  - Simulate fluids inside and outside (***Lattice Boltzmann***)
  - ~~Requires high resolution~~
  - ~~Cannot control membrane viscoelastic properties (required to simulate RBC membrane)~~
  - Sharp interface (***Immersed Boundary***) (to accomodate continuum viscoelastic models)

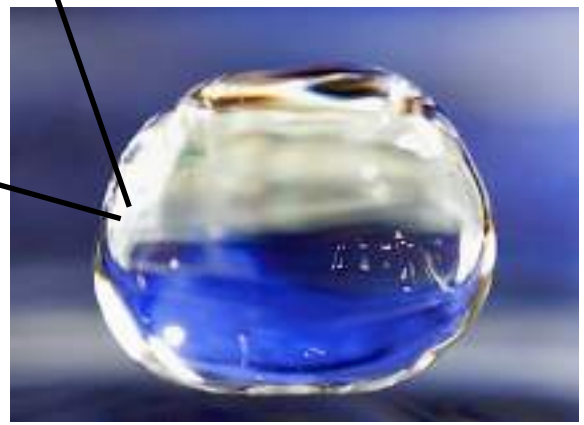


F. Guglietta, M. Behr. L. Biferale, G. Falcucci and M. Sbragaglia,
"On the effects of membrane viscosity on transient red blood cell dynamics",
Soft matter, 16(26), 6191-6205 (2020)

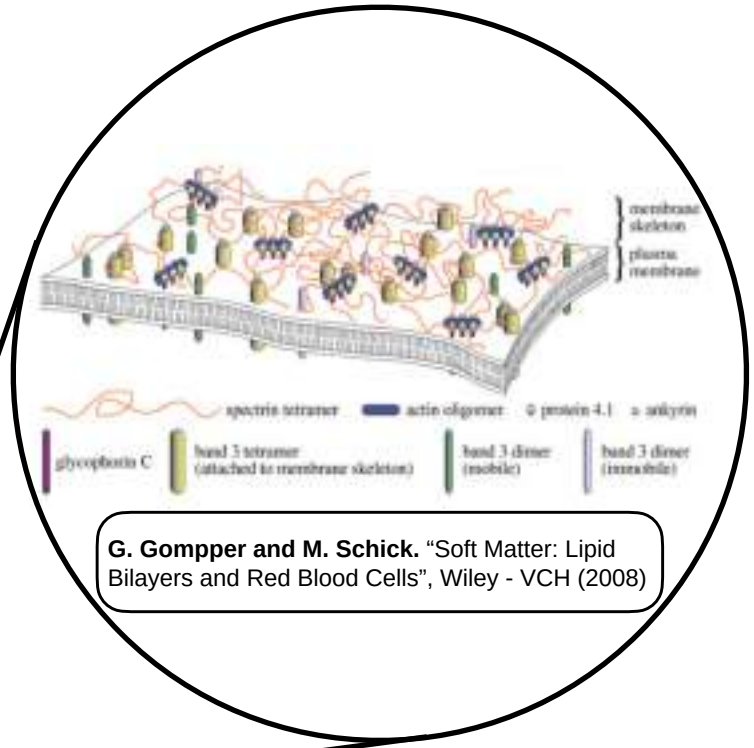
Complex interface

If the RBC membrane was like the drop interface…





- **Interface** between two fluids
- **Surface tension** ζ
- **Spherical shape** (“simple” interface)



G. Gompper and M. Schick. “Soft Matter: Lipid Bilayers and Red Blood Cells”, Wiley - VCH (2008)

- **Membrane** (phospholipid bilayer)
- **Elastic forces** coming from the **complex structure** of the membrane
- **Biconcave shape** (carry more oxygen)

Membrane elasticity - Continuum description

Strain energy
(Skalak model):

Strain invariants:
$I_1 = \lambda_x^2 + \lambda_y^2 - 2$
$I_2 = \lambda_x^2 \lambda_y^2 - 1$

$$W_S = \frac{1}{12}\int dA \left[\kappa_s \left(I_1^2 + 2I_1 - 2I_2\right) + \kappa_\alpha I_2^2\right]$$

Elastic shear modulus

Elastic dilatational modulus

R. Skalak, et al. "Strain energy function of red blood cell membranes." *Biophysical journal* 13.3: 245-264 (1973)

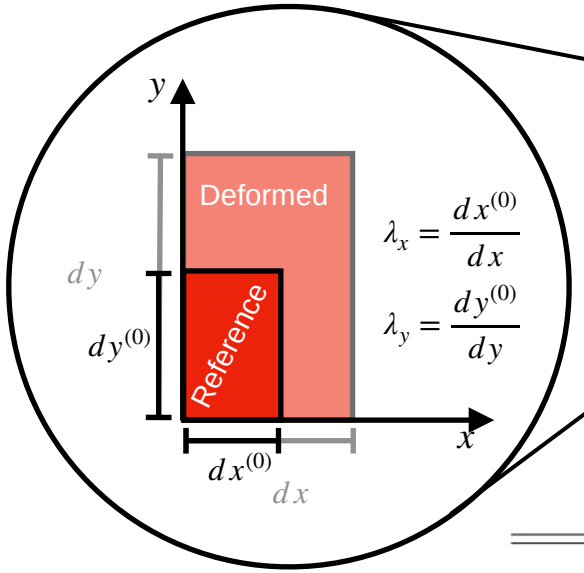
Bending energy
(Helfrich formulation):

Bending modulus

$$W_B = \frac{\kappa_B}{2}\int dA \left(H - H^{(0)}\right)$$

Trace of the surface curvature tensor

W. Helfrich "Elastic properties of lipid bilayers: theory and possible experiments." *Zeitschrift für Naturforschung C* 28.11-12: 693-703 (1973)

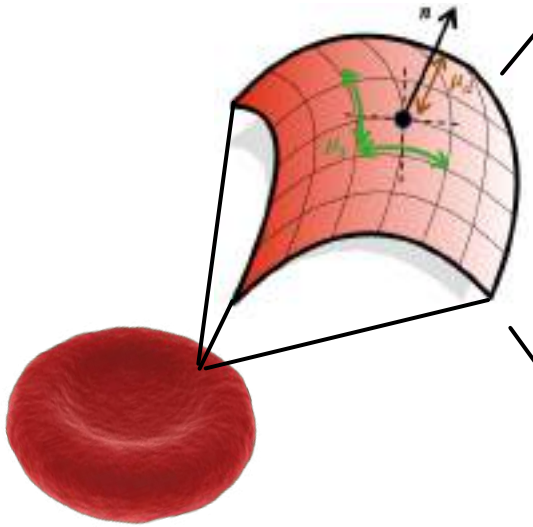


RBC PARAMETERS

Parameter	Value
Radius r	3.91 μm
Area A	133 μm^2
Volume V	93 μm^3
Elastic shear modulus k_S	5.3 μNm^{-1}
Elastic dilatational modulus k_α	50 k_s
Bending modulus k_B	2 10^{-19}Nm
Plasma viscosity μ_{out}	1.2 10^{-3} Pa s
Cytoplasm viscosity μ_{in}	6 10^{-3} Pa s

- **Skalak** and **Helfrich** models are **validated** and explain **experimental evidences** at the RBC scales.
- **Elastic coefficients** (k_s, k_α and k_B) are experimentally measured with **very high accuracy**.

Membrane viscosity - Continuum description



Boussinesq-Scriven law

Strain tensor $\mathbf{E} = \frac{1}{2}[\nabla \mathbf{u} + \nabla \mathbf{u}^T]$

$$\tau^v = \tau_s^v + \tau_d^v = \mu_s \left[2\mathbf{E} - \mathrm{tr}(\mathbf{E})\mathbf{I}\right] + \mu_d \mathrm{tr}(\mathbf{E})\mathbf{I} = 2\mu_m \mathbf{E}$$

Shear membrane viscosity

Dilatational membrane viscosity

L. Scriven "Dynamics of a fluid interface - equation of motion for newtonian surface fluids", Chemical Engineering Science 12,98 (1960)

This experimental uncertainty prompts a parametric investigation

$\mu_m \rightarrow t_R$

Inverse process with respect to experiments. We do not assume any relation between μ_m and t_R

Experiments usually measure **relaxation times** and use **simplified viscoelastic models** to compute $\mu_m = M(t_R)$

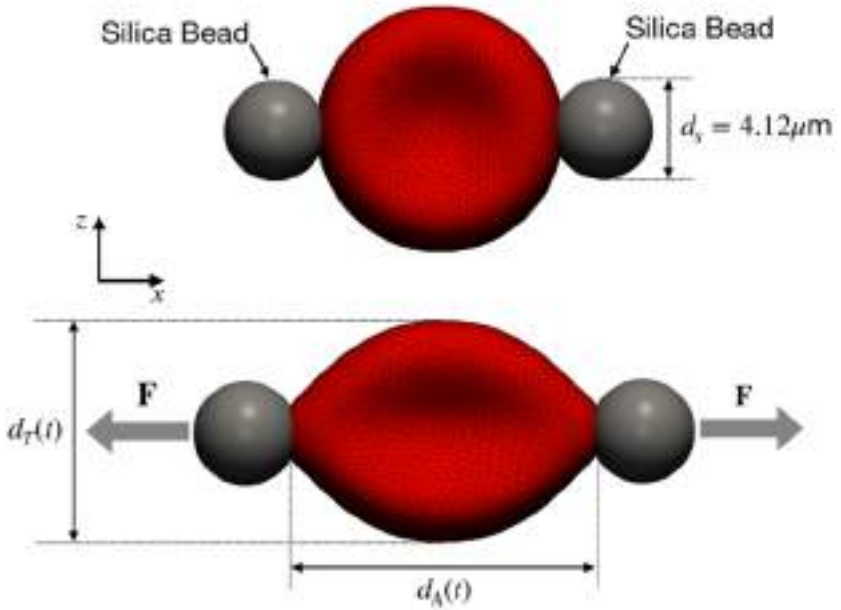
Author	t_R [s]	μ_m [10^{-7} mPa s]	Technique
Evans & Hochmuth 1976	0.300	~1	Micropipette aspiration
Chien *et al.* 1978	0.146 ± 0.055	0.6–4.0	Micropipette aspiration
Hochmuth *et al.* 1979	0.100–0.130	6–8	Micropipette aspiration
Tran-Son-Tay *et al.* 1984	—	0.53–0.96	Tank-treading
Baskurt & Meiselman 1996	0.119 ± 0.017	—	Shear (light reflection)
Baskurt & Meiselman 1996	0.097 ± 0.015	—	Shear (ektacytometry)
Riquelme *et al.* 2000	—	2.7–4.1	Sinusoidal shear stress
Tomaiuolo & Guido 2011	0.100	4.7–10.0	Microchannel deformation
Braunmüller *et al.* 2012	0.100–0.130	~10	Micropipette aspiration
Prado *et al.* 2015	0.08 ± 0.01	0.6–0.9	Numerical and experimental
Fedosov 2010	0.100–0.130	~1[a]	Numerical simulation

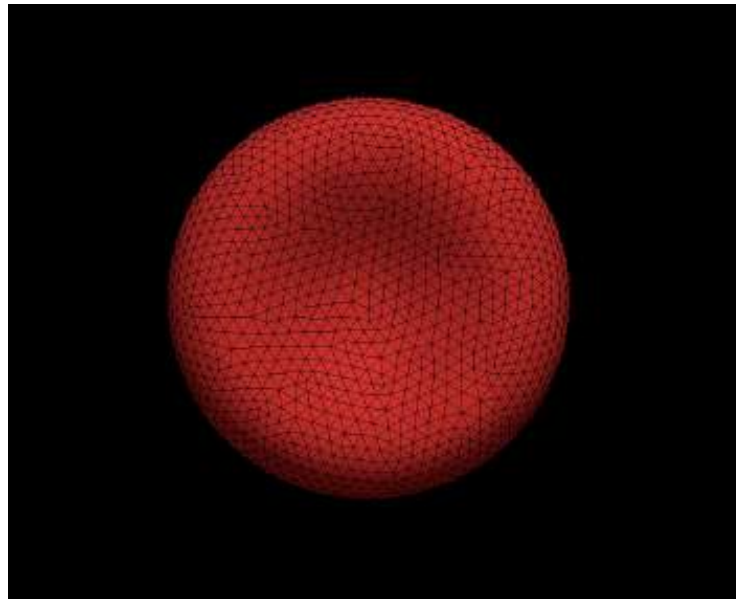
F. Guglietta et al., "On the effects of membrane viscosity on transient red blood cell dynamics", *Soft matter*, 16(26), 6191-6205 (2020)

STretching Simulation (STS)

Stretching in optical tweezers

JPK BioAFM, Bruker Nano GmbH
https://www.youtube.com/watch?v=QkZ95RF_Zf4

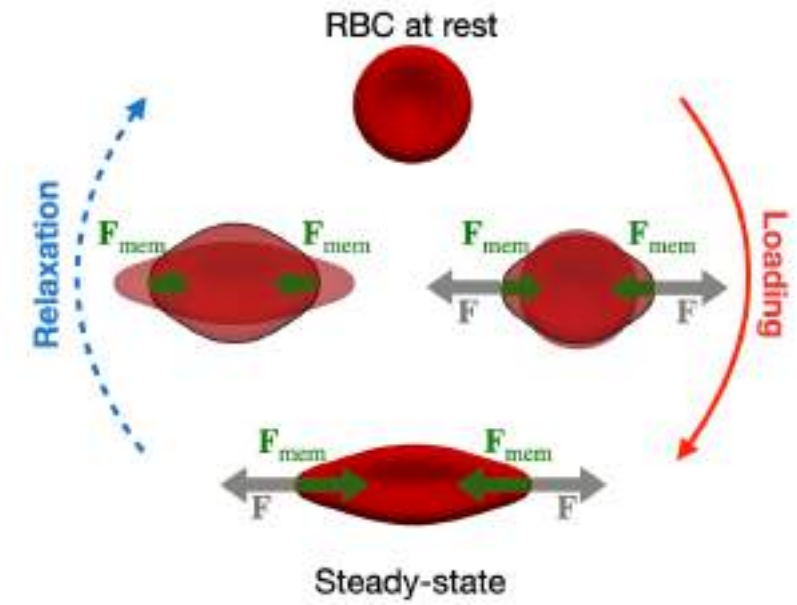




Taylor index

$$D(t) = \frac{d_A(t) - d_T(t)}{d_A(t) + d_T(t)}$$

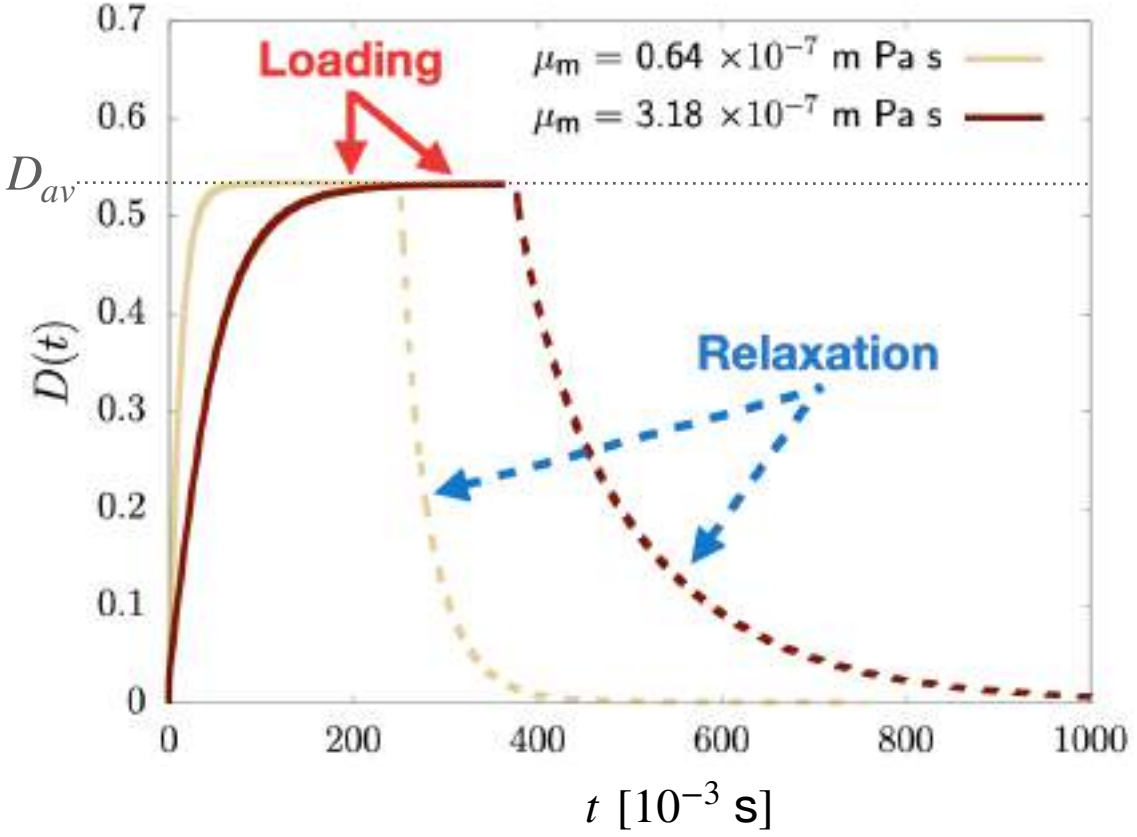
STretching **S**imulation (STS)



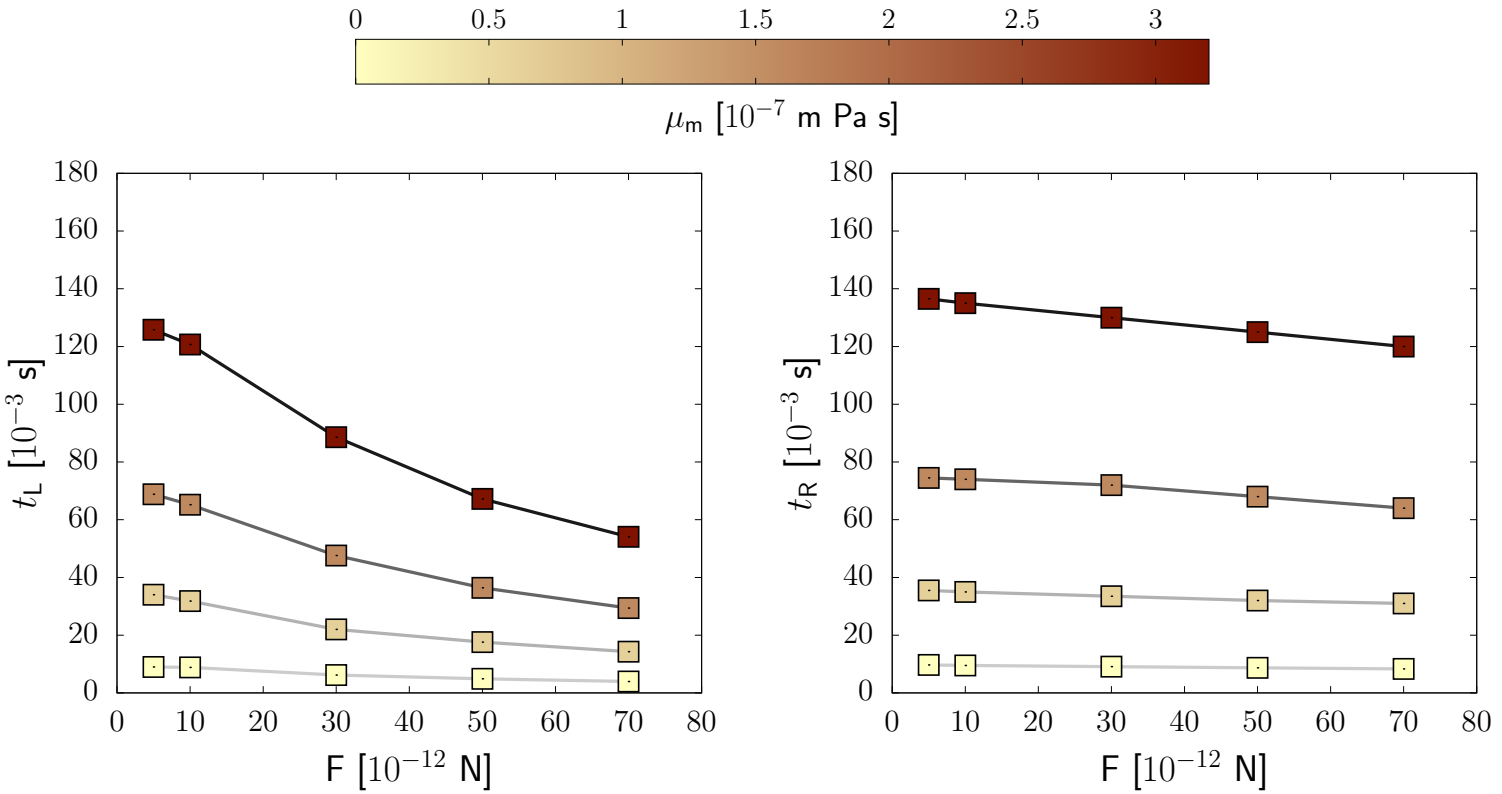
Stretched exponential function fit

Loading: $L(t) = D_{av} - D_{av} \exp\left\{-\left(\frac{t}{t_L}\right)^{\delta}\right\}$

Relaxation: $R(t) = D_{av} \exp\left\{-\left(\frac{t}{t_R}\right)^{\delta}\right\}$



STretching Simulation (STS)



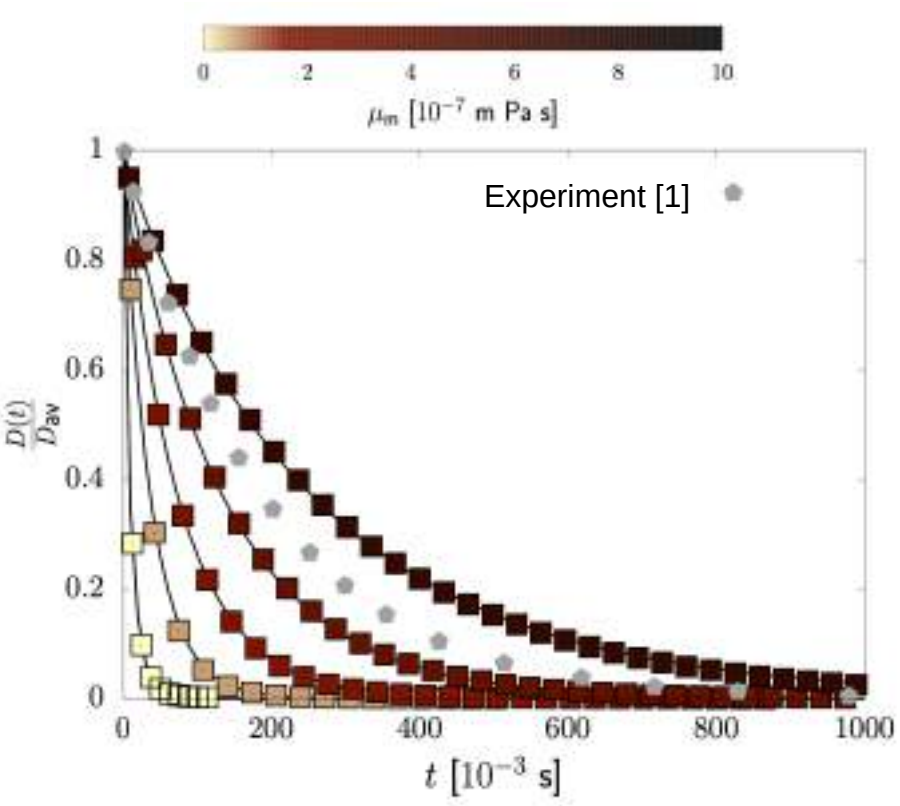
Loading

$$L(t) = D_{av} - D_{av} \exp\left\{-\left(\frac{t}{t_L}\right)^{\delta}\right\}$$

Relaxation

$$R(t) = D_{av} \exp\left\{-\left(\frac{t}{t_R}\right)^{\delta}\right\}$$

Comparison with experiments



- **Membrane viscosity** is needed to **reproduce experimental results**
- **The value of μ_m** is found by **comparing** the corresponding **relaxation time** with experiments

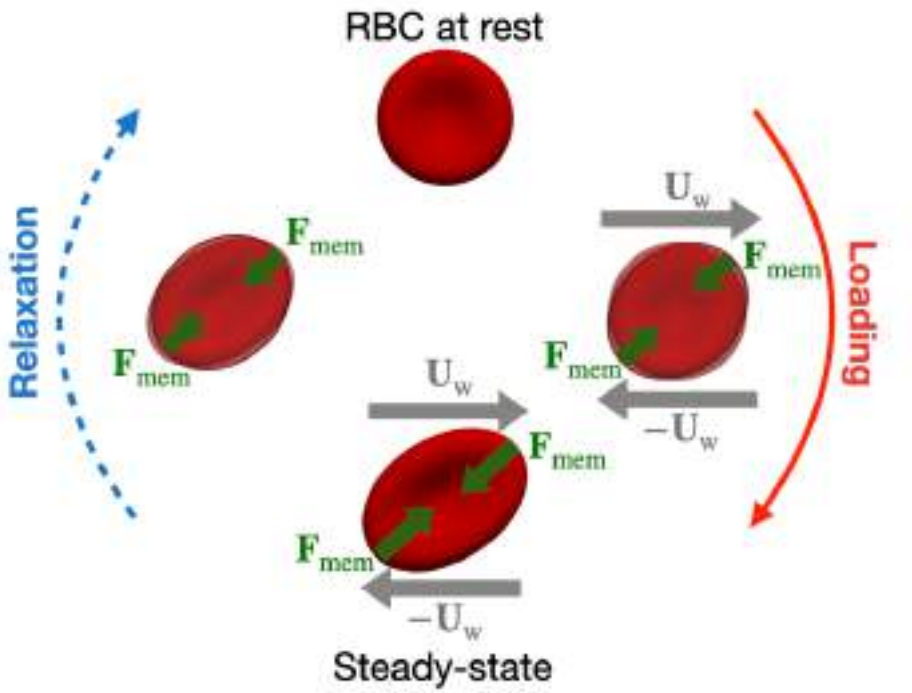
$$\mu_m \approx 5 \times 10^{-7} \text{m Pa s}$$

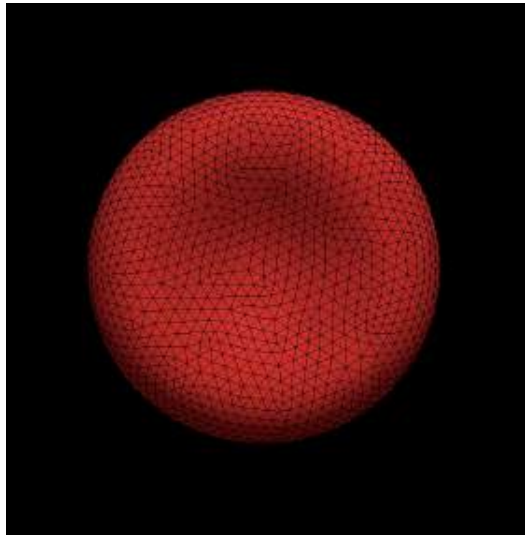
Author	t_R [s]	μ_m [10^{-7} mPa s]	Technique
Evans & Hochmuth 1976	0.300	~1	Micropipette aspiration
Chien *et al.* 1978	0.146 ± 0.055	0.6–4.0	Micropipette aspiration
Hochmuth *et al.* 1979	0.100–0.130	6–8	Micropipette aspiration
Tran-Son-Tay *et al.* 1984	—	0.53–0.96	Tank-treading
Baskurt & Meiselman 1996	0.119 ± 0.017	—	Shear (light reflection)
Baskurt & Meiselman 1996	0.097 ± 0.015	—	Shear (ektacytometry)
Riquelme *et al.* 2000	—	2.7–4.1	Sinusoidal shear stress
Tomaiuolo & Guido 2011	0.100	4.7–10.0	Microchannel deformation
Braunmüller *et al.* 2012	0.100–0.130	~10	Micropipette aspiration
Prado *et al.* 2015	0.08 ± 0.01	0.6–0.9	Numerical and experimental

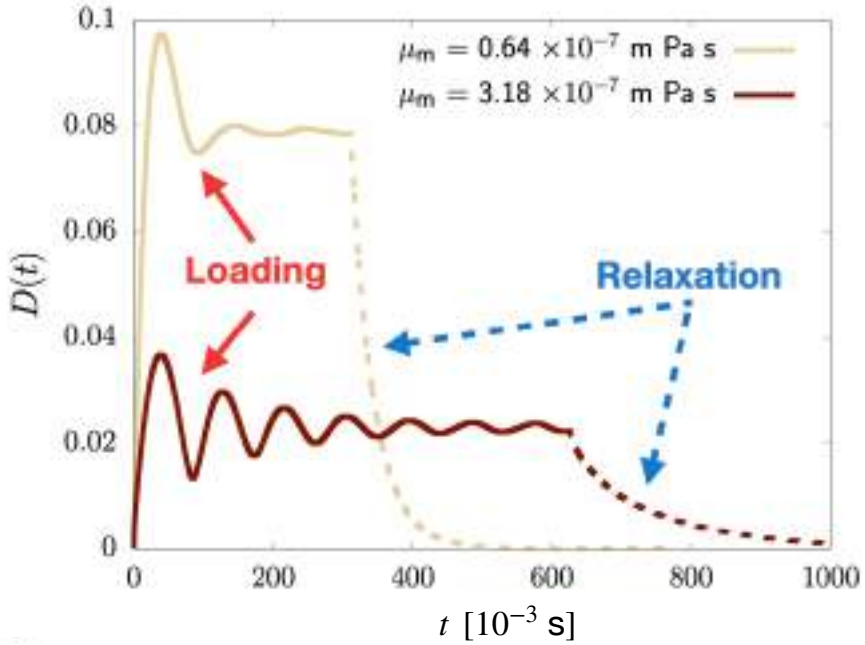
Does t_R depend on the kind of load mechanism?

[1] **J. P. Mills et al.,** "Nonlinear elastic and viscoelastic deformation of the human red blood cell with optical tweezers." *Molecular & Cellular Biomechanics* 1.3: 169 (2004)

SHear Simulation (SHS)



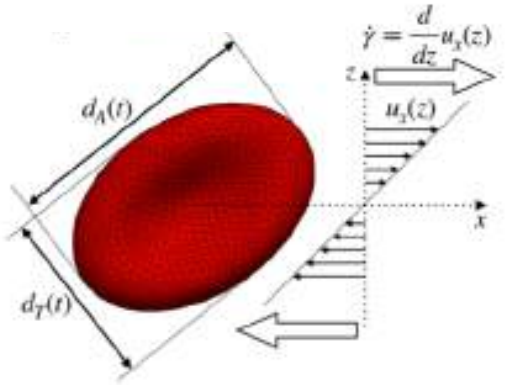




Stretched exponential function fit

Loading: $$L(t) = D_{av} - D_{av} \exp\left\{-\left(\frac{t}{t_L}\right)^{\delta}\right\} \cos\left(\frac{t}{t_L^{cos}}\right)$$

Relaxation: $$R(t) = D_{av} \exp\left\{-\left(\frac{t}{t_R}\right)^{\delta}\right\}$$

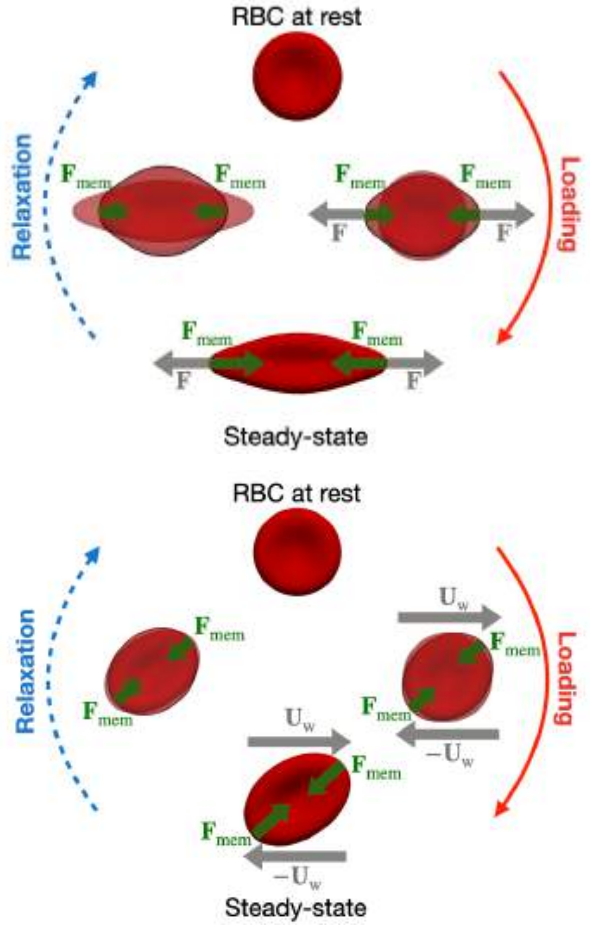


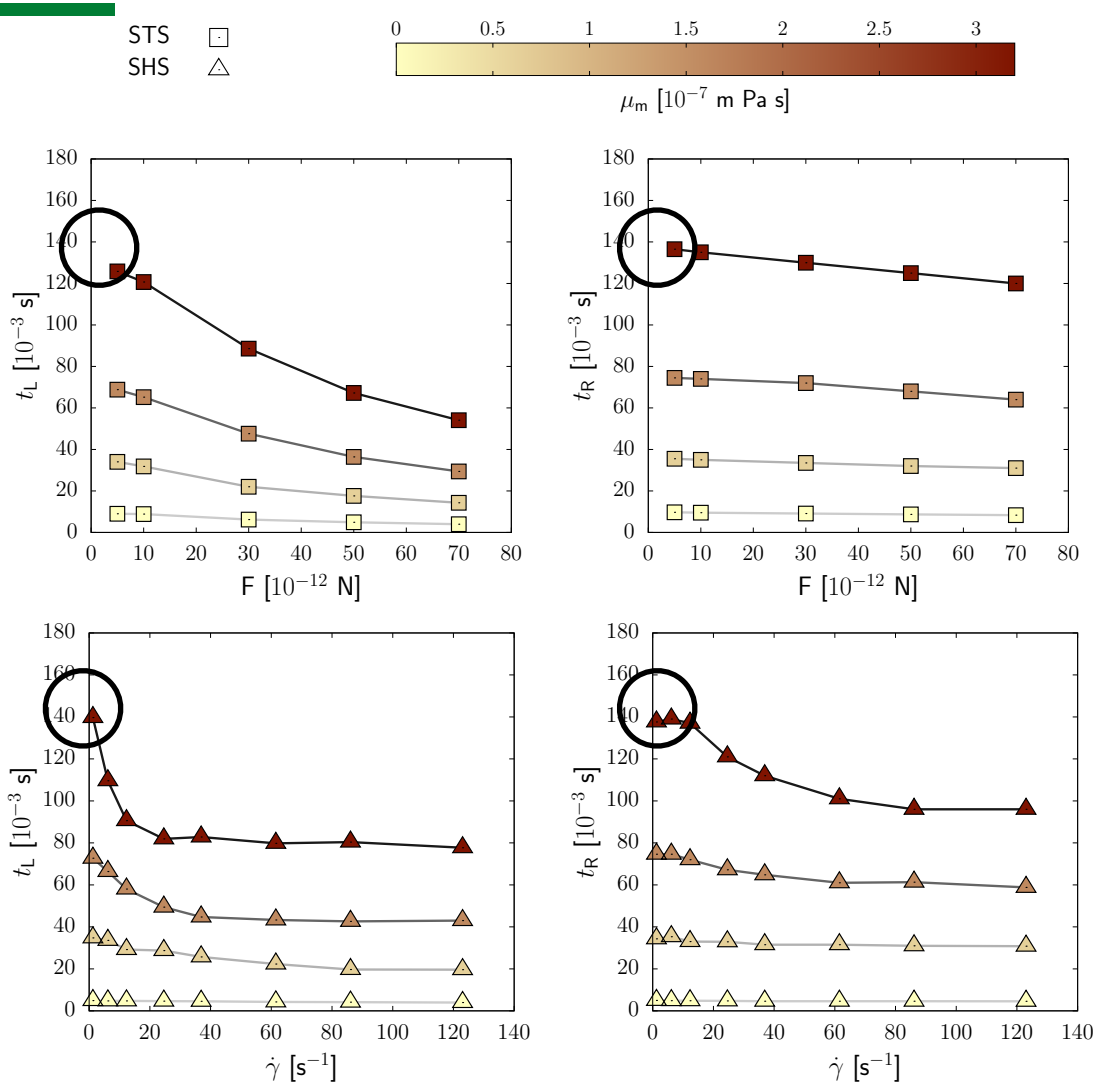
- Time scale of the **oscillation** is $\dot{\gamma}^{-1}$
- Time scale of the **deformation** $\sim \mu_m$

Comparison: STS vs. SHS

Stretching Simulation (STS)

Shear Simulation (SHS)



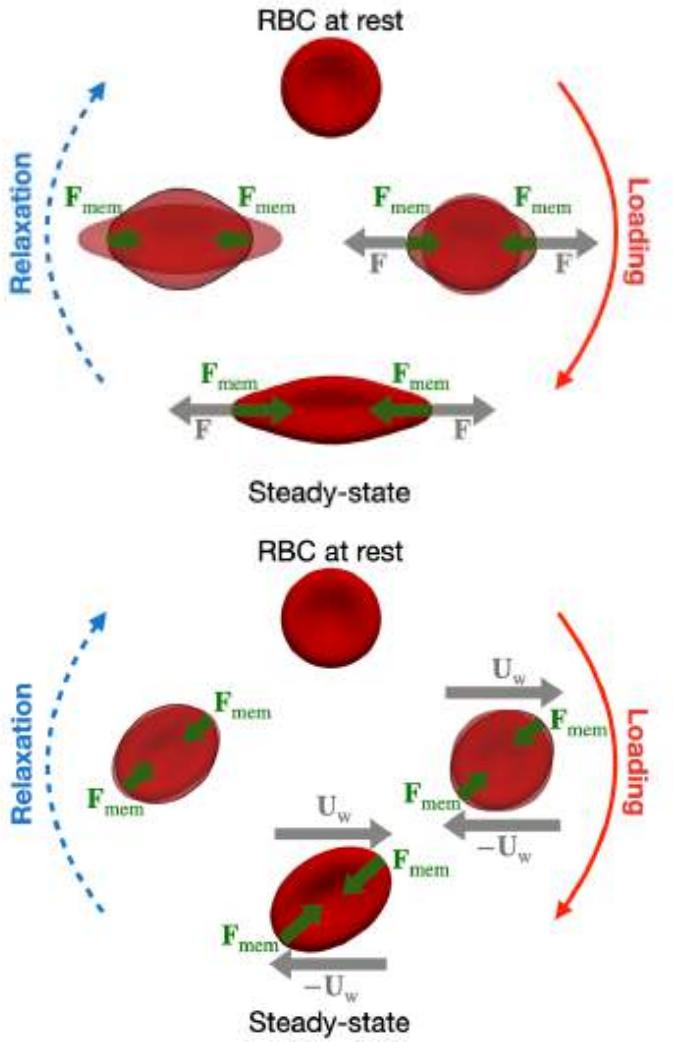


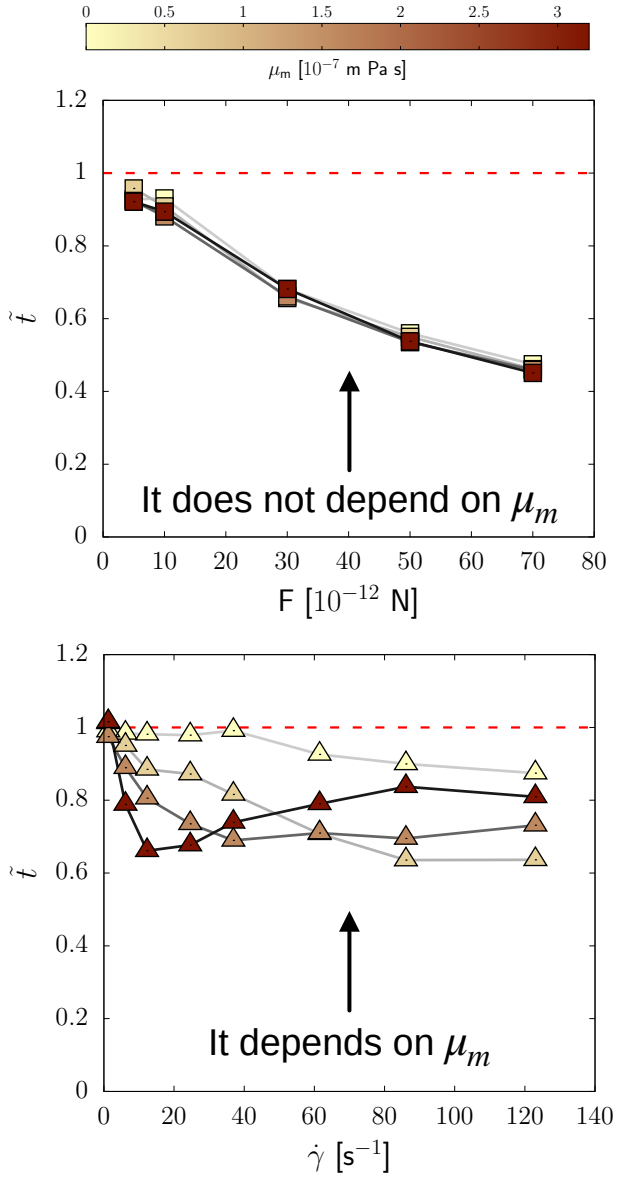
Loading

Relaxation

- Effect of membrane viscosity is **NOT universal**
- For small values of external force, the **intrinsic properties** of the membrane arise.
- Are loading and relaxation dynamics symmetrical?

Loading vs. relaxation: Symmetry?





$$\tilde{t} = \frac{t_L}{t_R} < 1$$

Loading is faster than relaxation
($t_L < t_R$)

Energetic motivation:
Loading: Viscoelastic force + External force
Relaxation: Viscoelastic force

Why is Loading-Relaxation Symmetry so different?

Why so different?

a. Membrane does ***not*** rotate
b. Forced ***directly*** on the membrane

a. Membrane ***does*** rotate
b. Forced ***indirectly*** on the membrane
(i.e. forced by the fluid)

STS

F. Guglietta et al., "Loading and relaxation dynamics for a red blood cell", *Soft matter*, 17, 5978-5990, 2021.

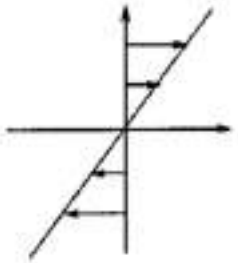
SHS

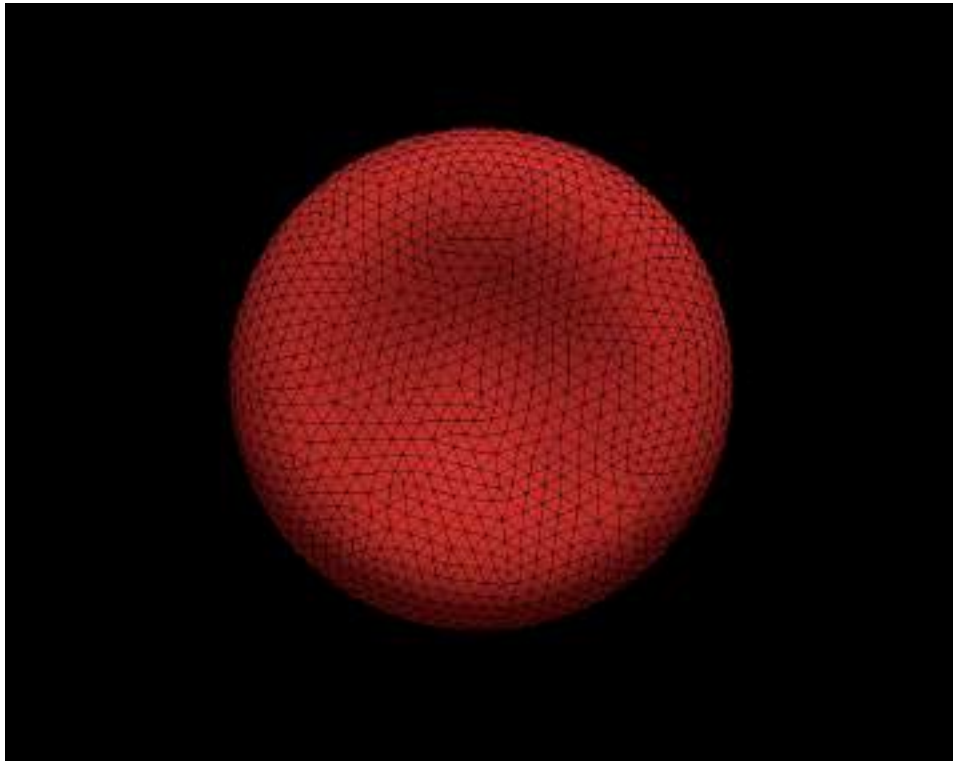
Why so different?

$$\frac{\partial \mathbf{u}}{\partial \mathbf{x}} = \begin{pmatrix} 0 & \dot{\gamma} \\ 0 & 0 \end{pmatrix}$$

Rotation

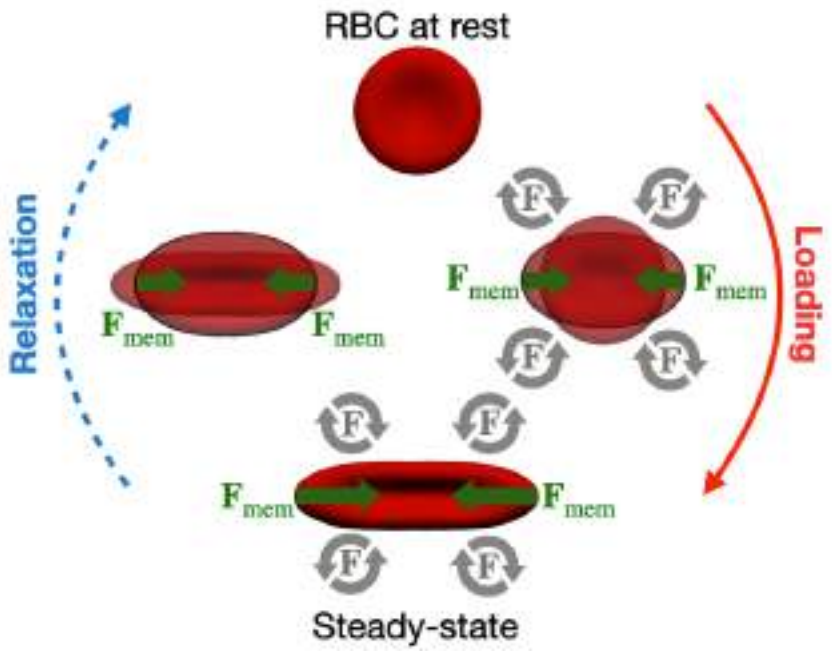
Elongation

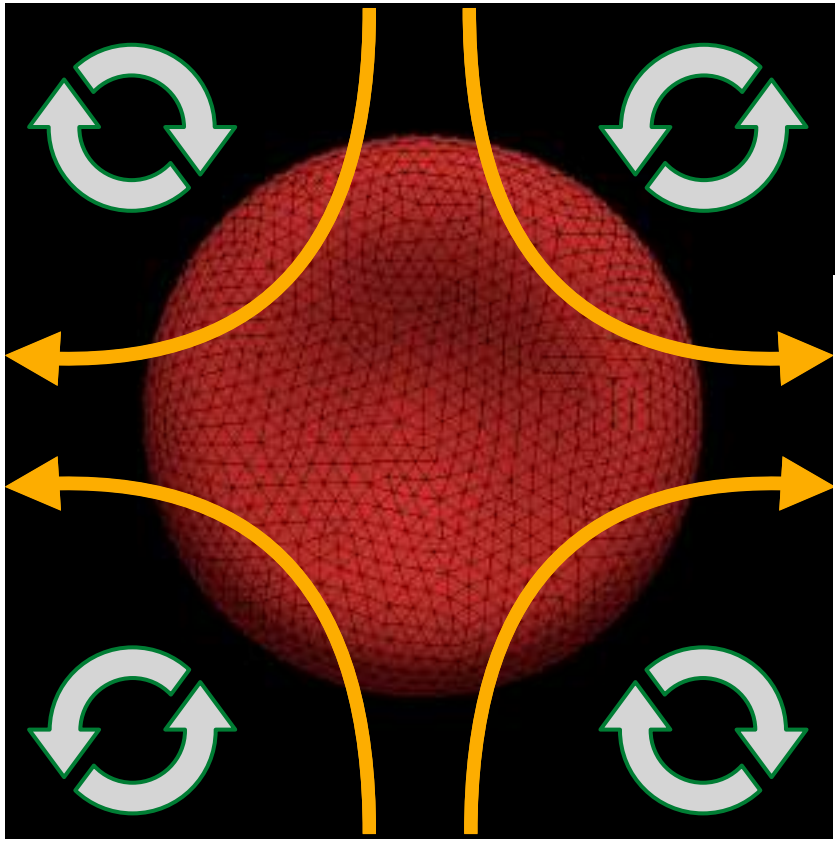




F. Guglietta et al., “Loading and relaxation dynamics for a red blood cell”, *Soft matter*, 17, 5978-5990, 2021.

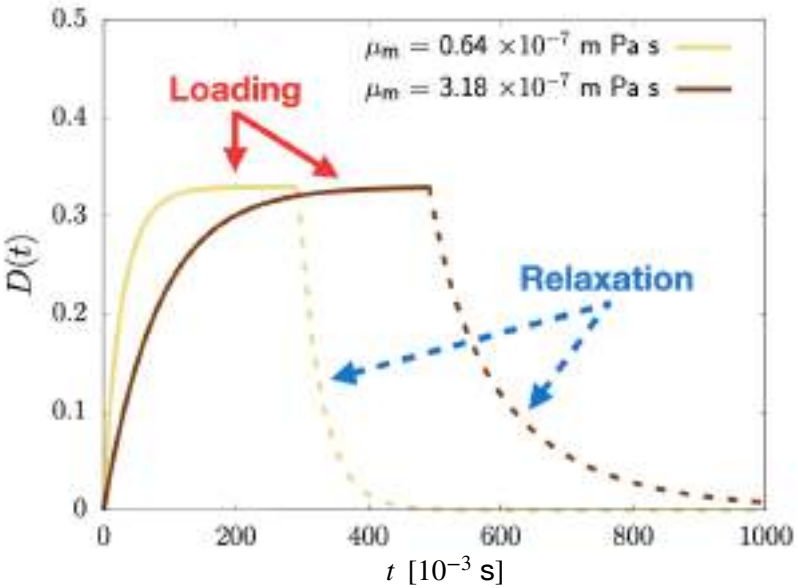
Four-Roll Mill Simulation (FRMS)





$$\mathbf{F}(x,y) = 2k\mu\dot{\gamma}_{FRMS}\begin{pmatrix}\sin(kx)\cos(ky)\\ -\cos(kx)\sin(ky)\\ 0\end{pmatrix}$$

$$k = \frac{2\pi}{L}$$



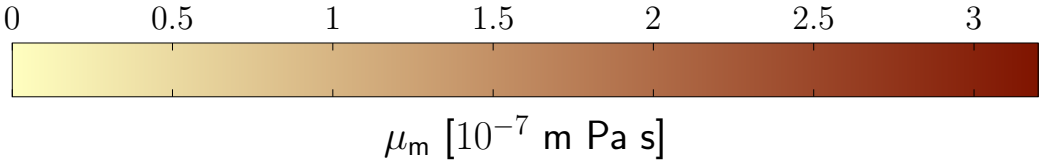
Stretched exponential function fit

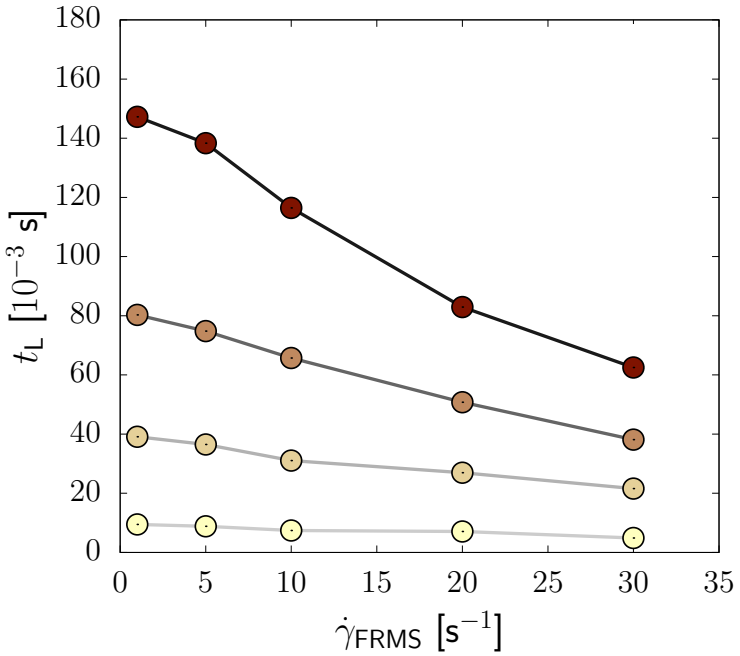
Loading: $$L(t) = D_{av} - D_{av}\ \exp\left\{-\left(\frac{t}{t_L}\right)^{\delta}\right\}$$

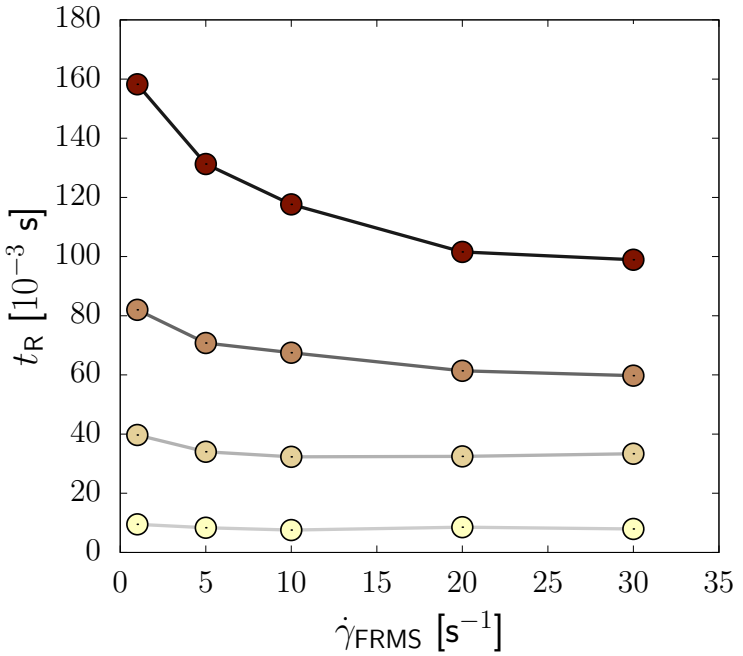
Relaxation: $$R(t) = D_{av}\ \exp\left\{-\left(\frac{t}{t_R}\right)^{\delta}\right\}$$

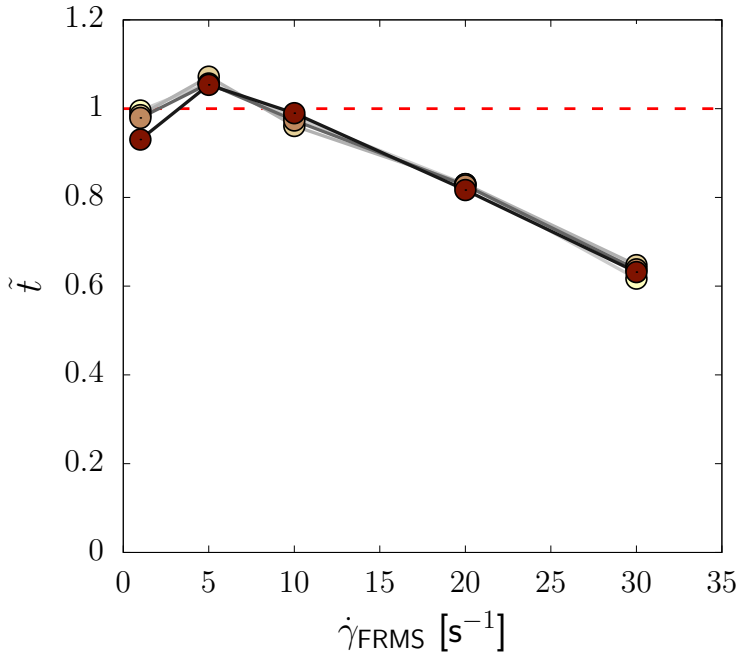
a. Membrane does not rotate (like **STS**)
b. Forced indirectly on the membrane (like **SHS**)

Four-Roll Mill Simulation (FRMS)









Loading

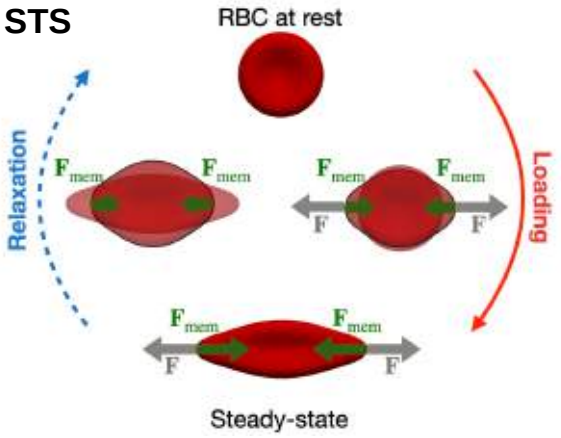
$$L(t) = D_{av} - D_{av} \exp\left\{-\left(\frac{t}{t_L}\right)^{\delta}\right\}$$

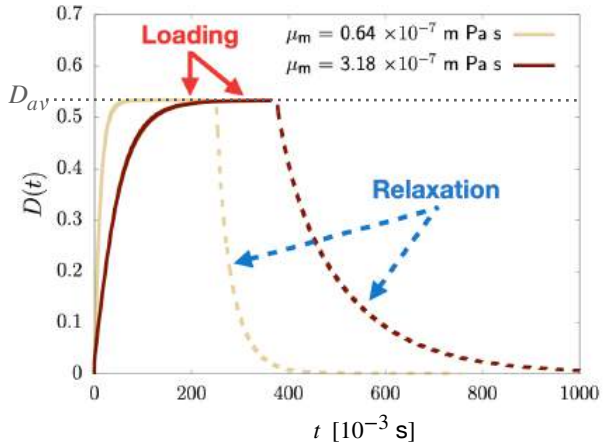
Relaxation

$$R(t) = D_{av} \exp\left\{-\left(\frac{t}{t_R}\right)^{\delta}\right\}$$

F. Guglietta et al., "Loading and relaxation dynamics for a red blood cell", *Soft matter*, 17, 5978-5990, 2021.

STS





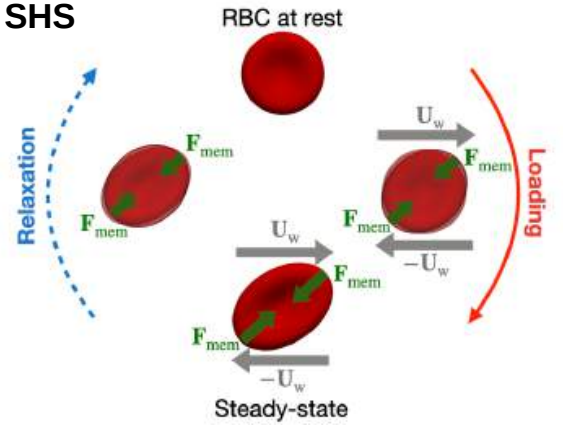
Energy injected

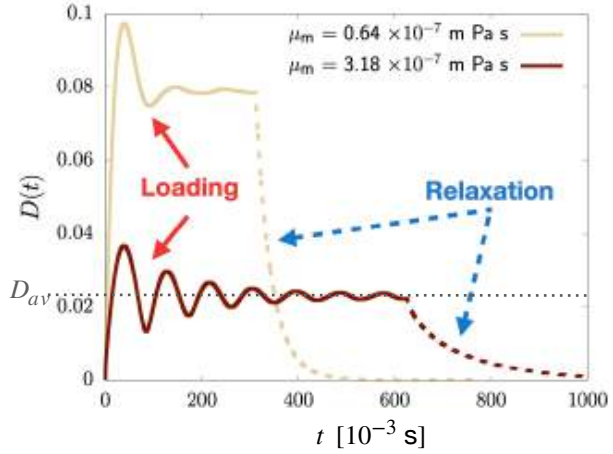
Deformation

Energy released

Deformation

SHS





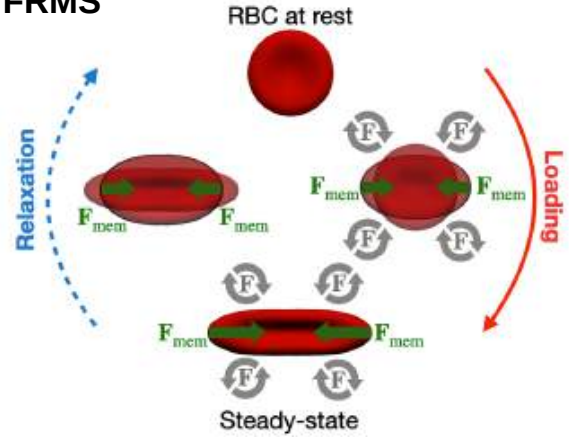
Energy injected

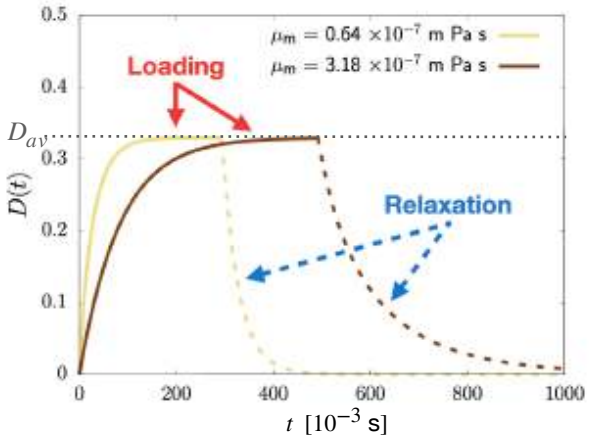
Deformation + Rotation

Energy released

Deformation

FRMS



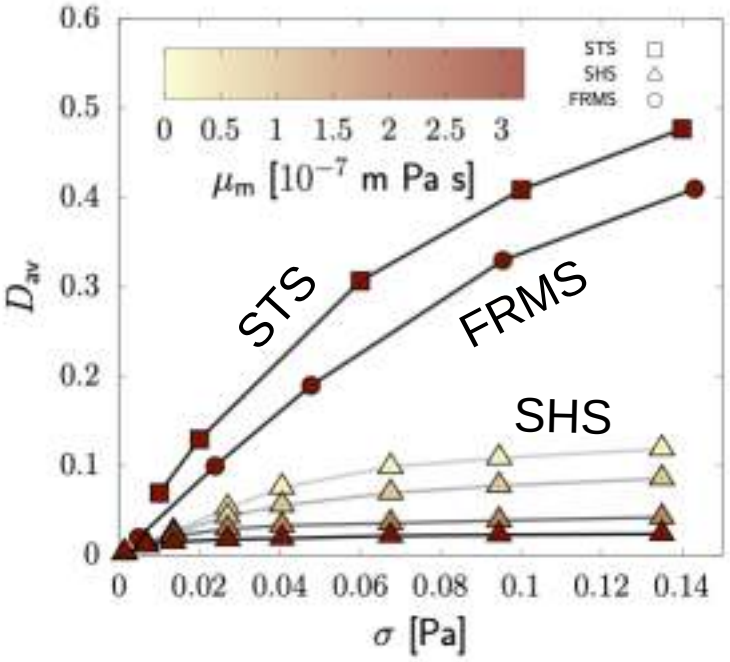


Energy injected

Deformation

Energy released

Deformation



- **The steady value of deformation** may depend on the value of **membrane viscosity**
- **The rotation** of the membrane **dissipates** part of the energy otherwise used to **deform** the membrane.

TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Suspension of capsules

[4] **F. Guglietta,** F. Pelusi, M. Sega, O. Aouane and J. Harting, “Suspensions of viscoelastic capsules: Effect of membrane viscosity on transient dynamics”, Journal of Fluid Mechanics, 971, A13 (2023)

Membrane model

Energy:

$W = W_s + W_v$

Skalak model (resistance against strain) [1,2]

$$W_s = \frac{1}{12}\int \left[k_s \left(I_1^2 + 2I_1 - 2I_2 \right) + k_\alpha I_2^2 \right] dS$$

k_s: Elastic *shear* modulus; k_α: Elastic *dilatational* modulus

Strain invariants:

$$I_1 = \lambda_x^2 + \lambda_y^2 - 2$$
$$I_2 = \lambda_x^2 \lambda_y^2 - 1$$

λ_x and λ_y are the principal stretch ratios

Global volume conservation [2]

$$W_v = \frac{k_v}{2}\frac{(V - V_0)^2}{V_0}$$

Volume modulus: k_v

Initial volume: V_0

Viscous tensor:

τ^v

Boussinesq-Scriven law [3,4]

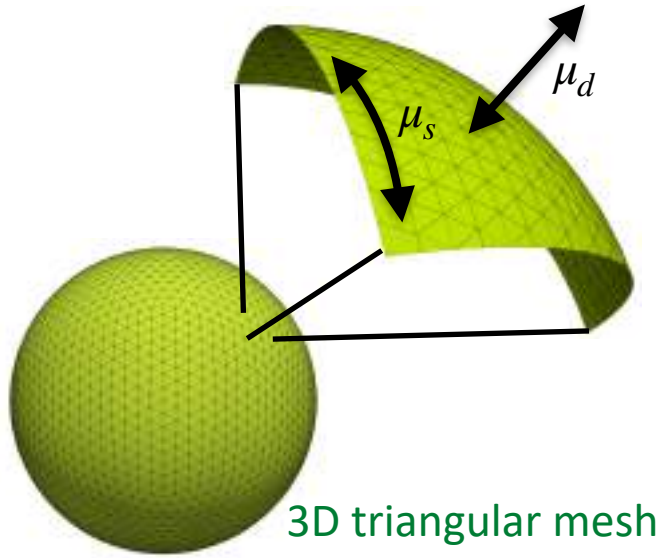
$$\tau^v = \mu_s \left[2\mathbf{E} - \mathrm{tr}(\mathbf{E})\mathbf{I} \right] + \mu_d \mathrm{tr}(\mathbf{E})\mathbf{I} = 2\mu_m \mathbf{E}$$

$\mathbf{E} = \frac{1}{2}[\nabla \mathbf{u} + \nabla \mathbf{u}^T]$

Shear membrane viscosity; *Dilatational* membrane viscosity; Membrane viscosity

Repulsive force [5]:

$$\vec{\varphi}_{ij} = \begin{cases} \bar{\epsilon} \left[\left(\frac{\Delta x}{d_{ij}} \right)^2 - \left(\frac{\Delta x}{\delta_0} \right)^2 \right] \hat{d}_{ij} & \text{if } d_{ij} < \delta_0 \,, \\ 0 & \text{if } d_{ij} \geq \delta_0 \,, \end{cases}$$



3D triangular mesh (5120 faces)

(1) Skalak, R., Tozeren, A., Zarda, R. P., & Chien, S. (1973). Strain energy function of red blood cell membranes. Biophysical journal, 13(3), 245-264.
(2) Krüger, T. (2012). Computer simulation study of collective phenomena in dense suspensions of red blood cells under shear. Springer Science & Business Media.
(3) Barthès-Biesel, D., & Sgaier, H. (1985). Role of membrane viscosity in the orientation and deformation of a spherical capsule suspended in shear flow. Journal of Fluid Mechanics, 160, 119-135.
(4) Li, P., & Zhang, J. (2019). A finite difference method with subsampling for immersed boundary simulations of the capsule dynamics with viscoelastic membranes. Int. J. Numer. Methods Biomed. Eng., 35(6), e3200.
(5) Aouane, O., Scagliarini, A., & Harting, J. (2021). Structure and rheology of suspensions of spherical strain-hardening capsules. Journal of Fluid Mechanics, 911.

Single capsule in simple shear flow

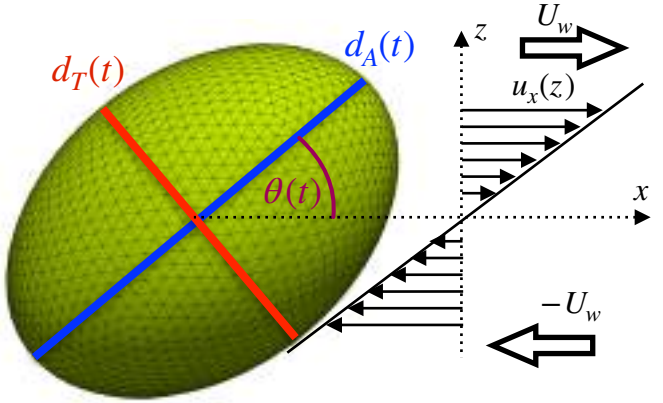
Reynolds number: $\mathrm{Re} = \dfrac{\dot{\gamma}\rho r^2}{\mu} = 10^{-2}$

Capillary number: $\mathrm{Ca} = \dfrac{\dot{\gamma}\mu r}{k_s} = \dot{\gamma} t^* \in [0.1,1]$

Boussinesq number: $\mathrm{Bq} = \dfrac{\mu_m}{\mu r} \in [0,50]$

Radius: r
Shear rate: $\dot{\gamma}$
Fluid viscosity: μ
Membrane viscosity: μ_m
Intrinsic time: $t^* = \mu r / k_s$

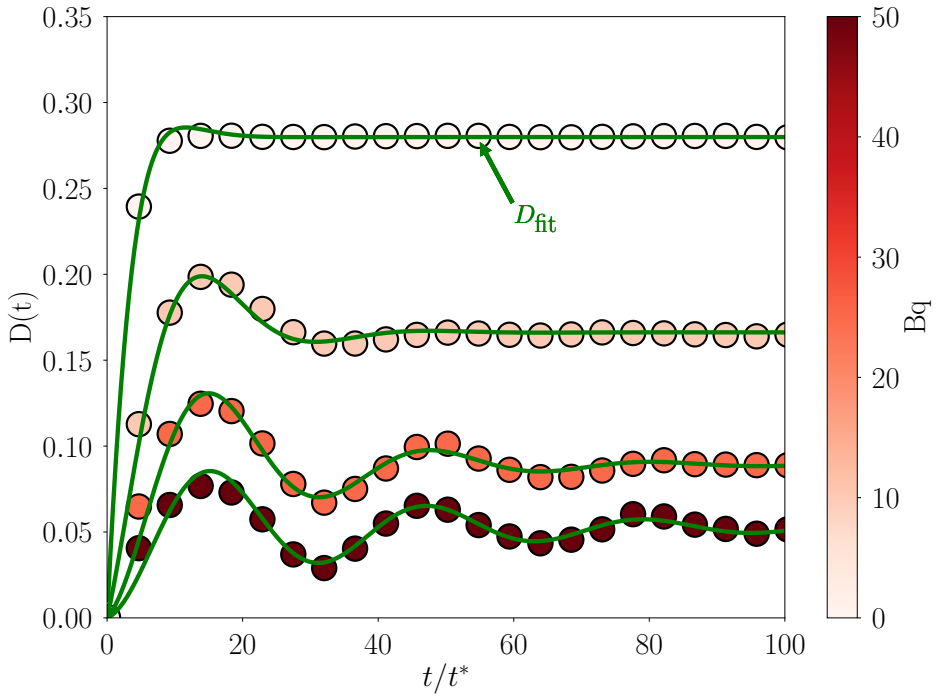
$$D(t) = \frac{d_A(t) - d_T(t)}{d_A(t) + d_t(t)}$$



Fit function:

$$D_{\mathrm{fit}}\left(\frac{t}{t^*}\right) = \bar{D}\left[1 - \exp\left(-\frac{t}{t^* t_L}\right)\cos\left(\omega\frac{t}{t^*}\right)\right]$$

Fit parameters: t_L, ω



Bq = 0

Bq = 50

Suspension of viscoelastic capsules

Capillary number: $\mathrm{Ca} = \frac{\dot{\gamma}\mu r}{k_s} \in [0.1,1]$

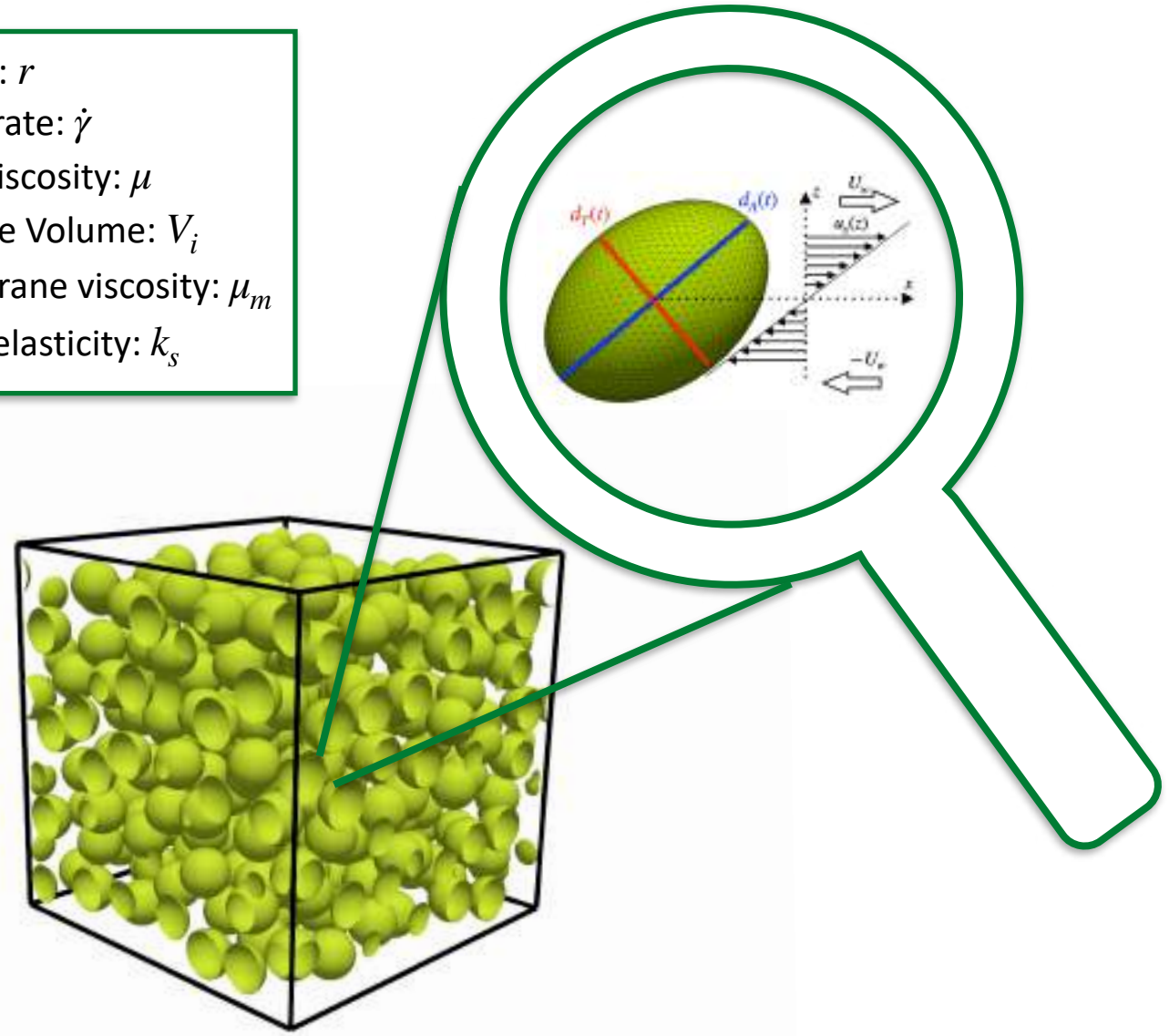
Viscous Force / Elastic Force

Boussinesq number: $\mathrm{Bq} = \frac{\mu_m}{\mu r} \in [0,50]$

Volume fraction: $\phi = \frac{\sum_i V_i}{L^3} \in [0.001,0.4]$

Deformation: $\langle D \rangle = \frac{1}{N} \sum_i D_i(t)$

Radius: r
Shear rate: $\dot{\gamma}$
Fluid viscosity: μ
Capsule Volume: V_i
Membrane viscosity: μ_m
Shear elasticity: k_s

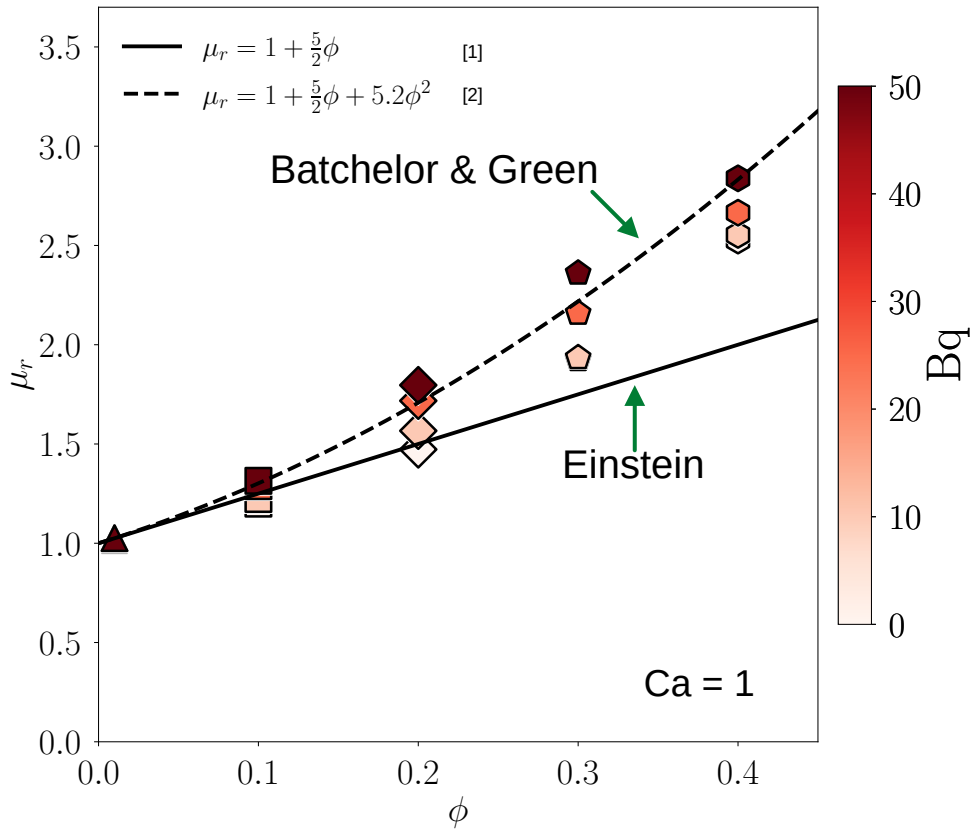


Rheology

Relative viscosity : $\mu_r = \frac{\mu_s}{\mu}$

Suspension viscosity

Fluid viscosity



Fluid viscosity

$$\mathrm{Ca} = \frac{\mu \dot{\gamma} R}{k_s}$$

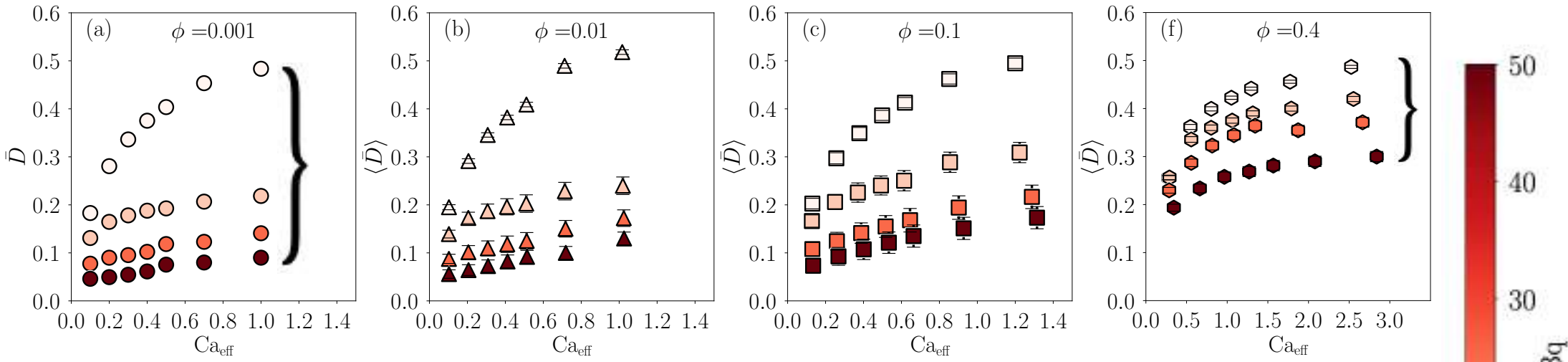
Suspension viscosity

$$\mathrm{Ca_{eff}} = \frac{\mu_s \dot{\gamma} R}{k_s} = \mathrm{Ca}\mu_r$$

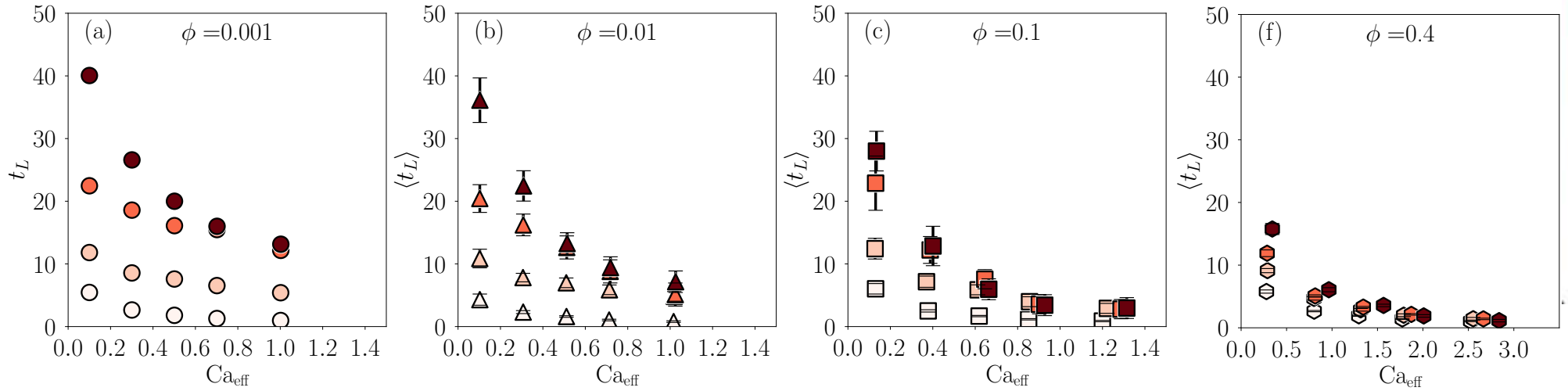
[1] **A. Einstein,** "Eine neue bestimmung der moleküldimensionen". Doctoral dissertation (1905)
[2] **G. K. Batchelor and J. Green**, The determination of the bulk stress in a suspension of spherical particles to order c2. *Journal of Fluid Mechanics*, *56*(3), 401-427, (1972)

Deformation and Loading time

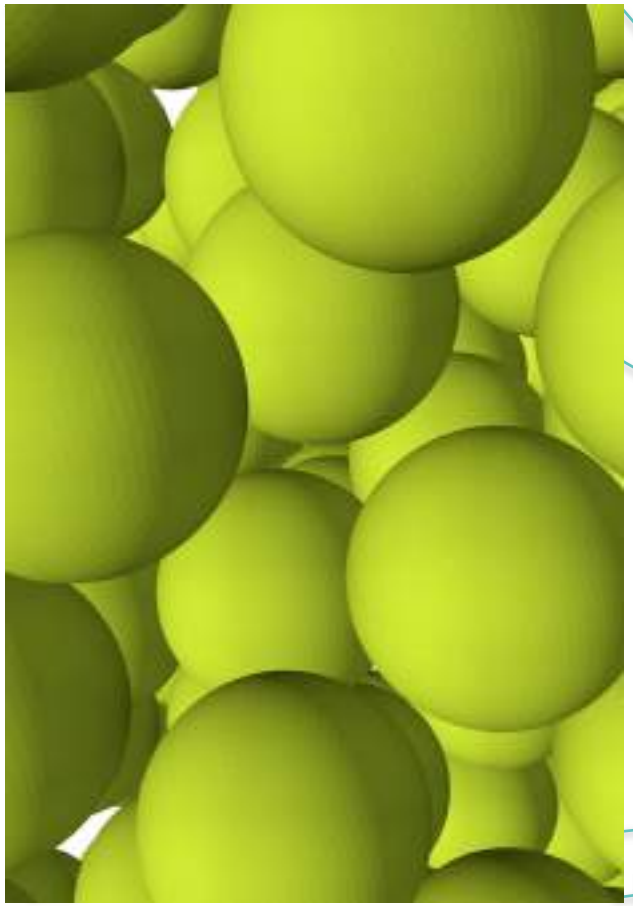
Deformation:



Loading time:

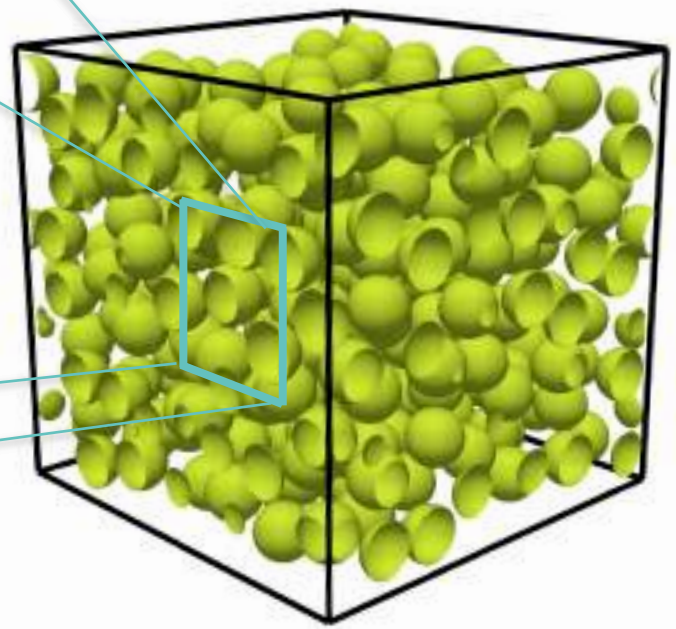


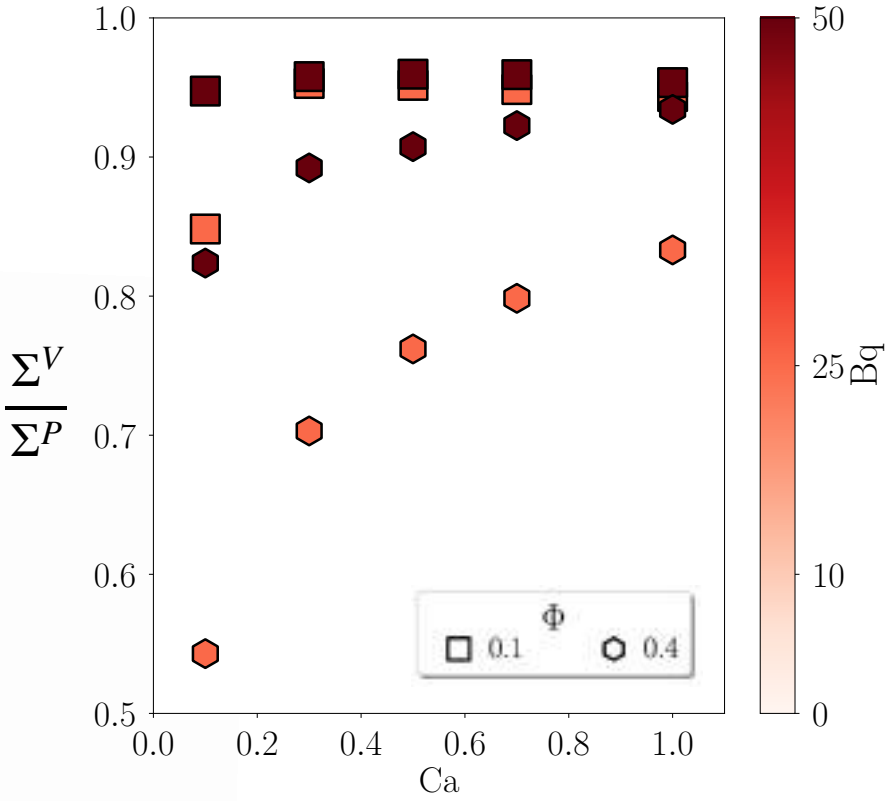
Membrane viscosity and volume fraction



Boussinesq-Scriven law

$$\tau^v = 2\mu_m \mathbf{E} = \mu_m[\nabla \mathbf{u} + \nabla \mathbf{u}^T]$$



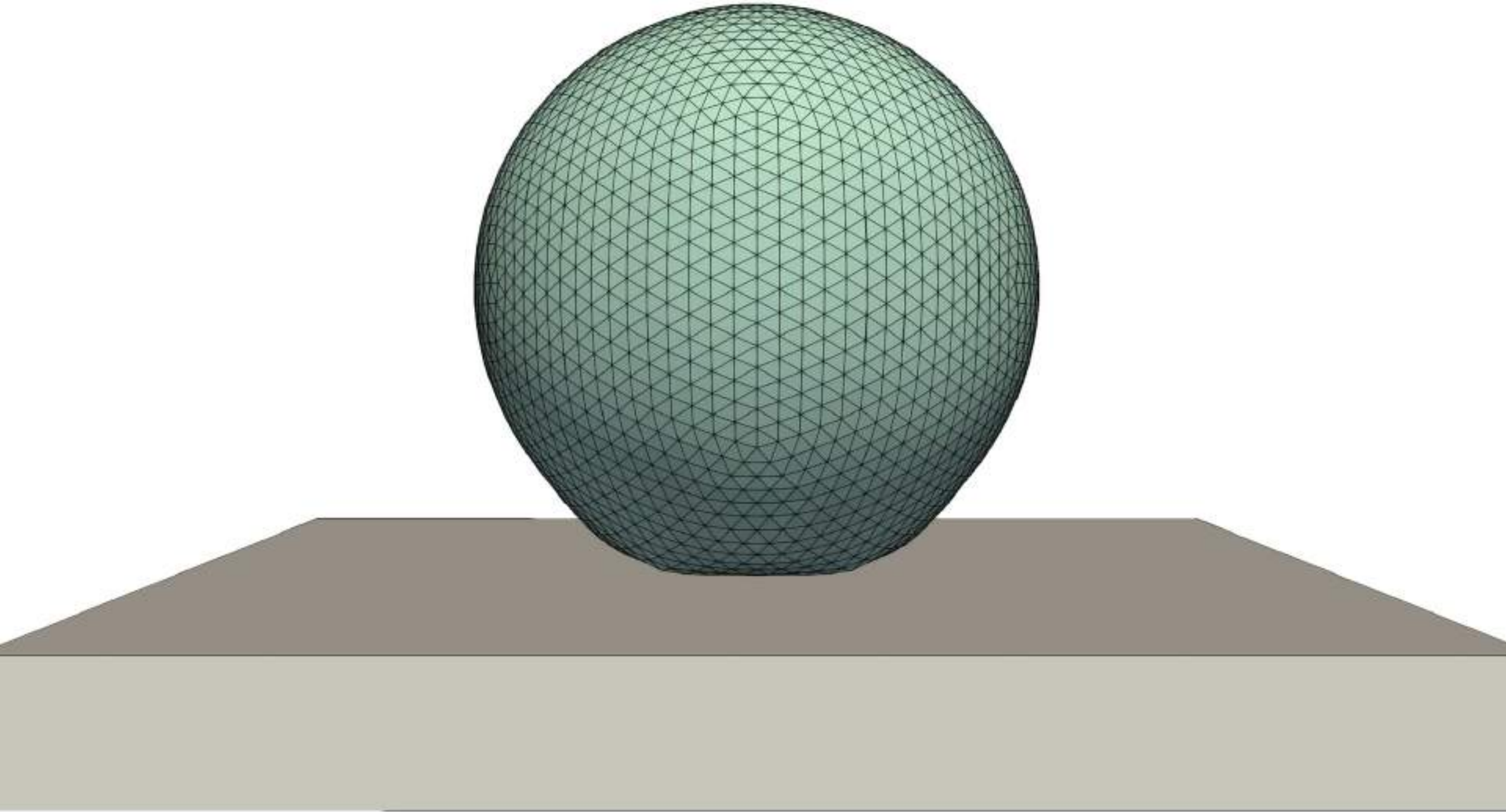


When the **volume fraction increases**, the **viscous dissipation reduces**

TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Wetting dynamics with sharp interface approach

[5] F. Pelusi, **F. Guglietta**, M. Sega, O. Aouane & J. Harting “A sharp interface approach for wetting dynamics of coated droplets and soft particles”, Physics of Fluids, 35(8), (2023)



Viscoelastic drop model

- Elastic energy density:

$$w = \frac{1}{2}(\alpha_1 - \alpha_3)\log(I_2 + 1) + \frac{1}{8}(\alpha_1 + \alpha_2)\log^2(I_2 + 1) + \alpha_3 \left[\frac{1}{2}(I_1 + 2) - 1\right]$$

- Volume conservation.

- Drop-wall interaction (Lennard-Jones):

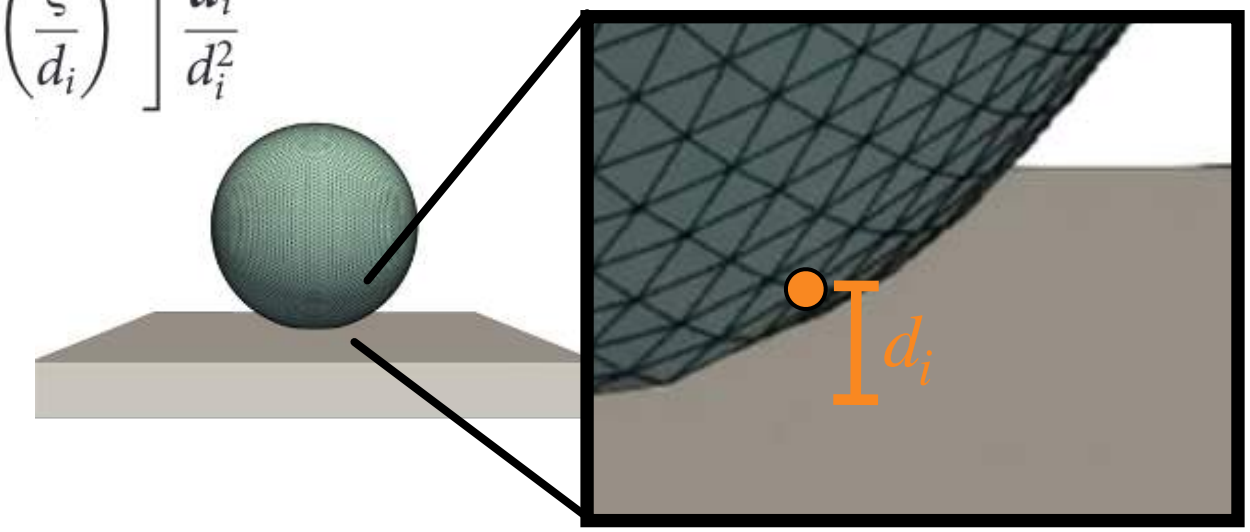
$$\boldsymbol{\varphi}_i^W = 48\epsilon\left[\left(\frac{\xi}{d_i}\right)^{12} - \frac{1}{2}\left(\frac{\xi}{d_i}\right)^{6}\right]\frac{\boldsymbol{d}_i}{d_i^2}$$

Pre-stressed

$\alpha = (\alpha_1, \alpha_2, \alpha_3)$	Model
$\alpha = \bar{\alpha}(1, 0, 0)$	Pure droplet
$\alpha = \bar{\alpha}(1, 0, 1)$	Softly coated droplet
$\alpha = \bar{\alpha}(1, 1, 1)$	Rigidly coated droplet

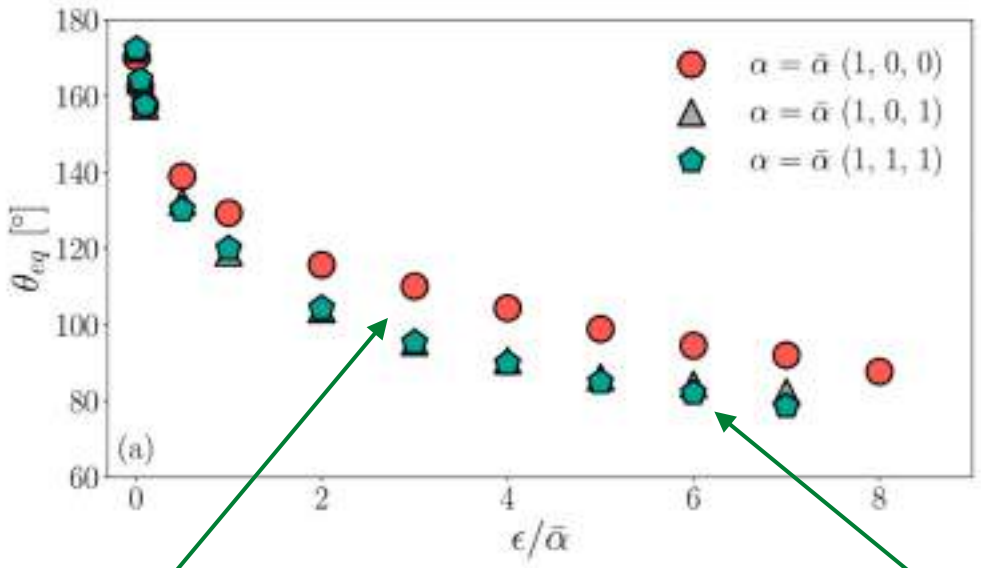
Not Pre-stressed

$\alpha = (\alpha_1, \alpha_2, \alpha_3)$	Model
$\alpha = \bar{\alpha}(0, 0, 1)$	Pure elastic capsule
$\alpha = \bar{\alpha}(0, 1, 1)$	Non-pre-stressed capsule

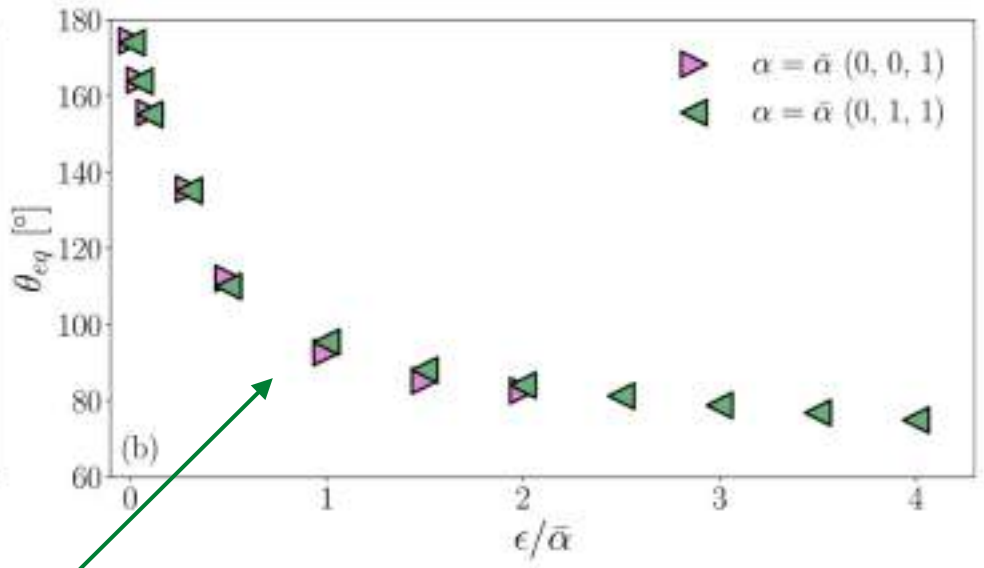


Equilibrium contact angle

Pre-stressed



Not pre-stressed



Effect of resistance against shear deformation (α_3)

No effect of resistance against area dilatation (α_2)

- Compare numerical simulations with experiment to find the real values of elastic coefficients α_i

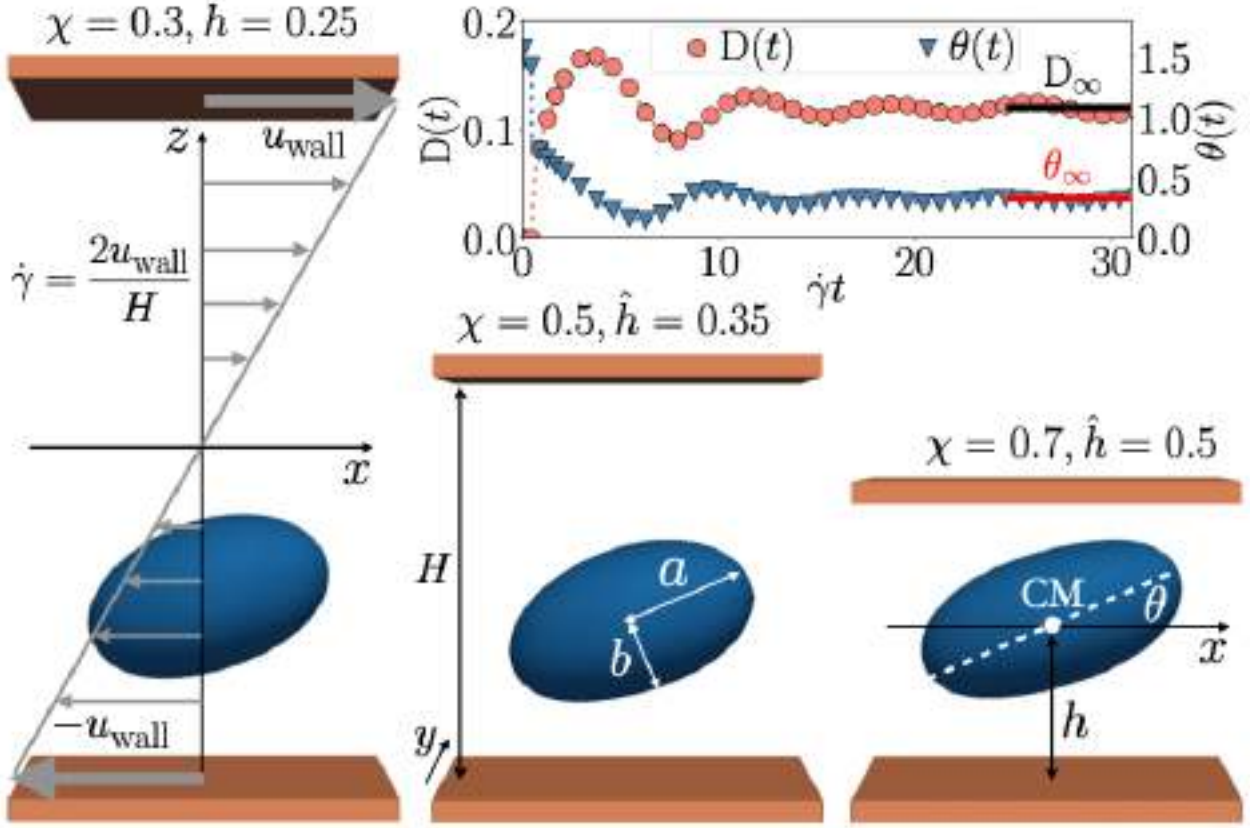
TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

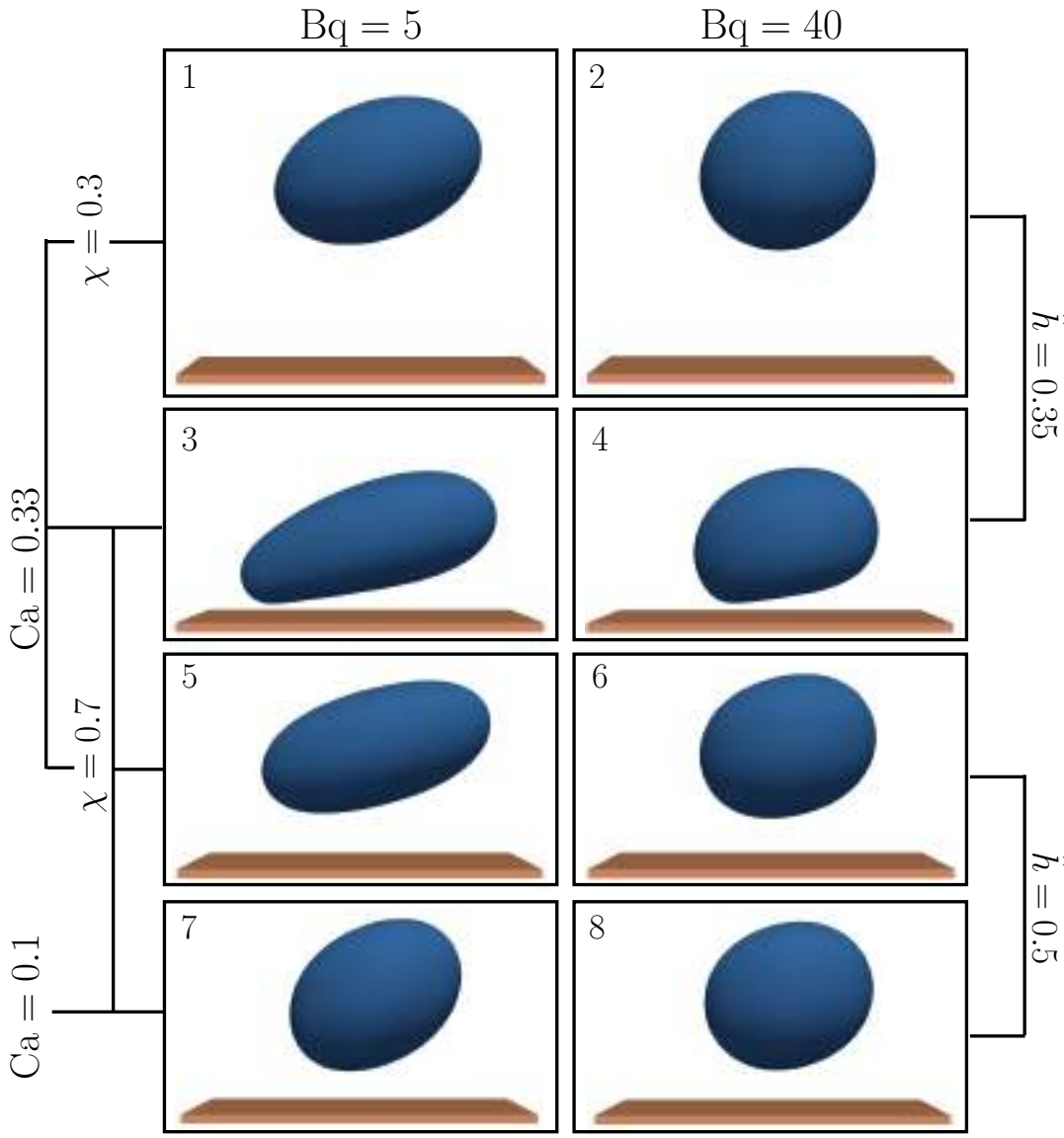
Confined drop in simple shear flow: effect of interface viscosity

[6] **F. Guglietta**, F. Pelusi “A unified analytical prediction for steady-state behavior of confined drop with interface viscosity under shear flow”, *in peer review*, (2024)

Numerical setup

Confinement degree: $\chi = \dfrac{2R}{H}$ Capillary number: $\mathrm{Ca} = \dfrac{\dot{\gamma}\mu R}{\sigma}$





Analytical results on drop deformation

	Without interface viscosity	With interface viscosity
Unconfined	Taylor [1] 1934	Flumerfelt [2] 1980
Confined	Shapira & Haber [3] 1990	Our Work [4] 2024

[1] G. I. Taylor, The formation of emulsions in definable fields of flow, Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character 146, 501 (1934).
[2] R. W. Flumerfelt, Effects of dynamic interfacial proper- ties on drop deformation and orientation in shear and extensional flow fields, Journal of Colloid and Interface Science 76, 330 (1980).
[3] M. Shapira and S. Haber, Low Reynolds number motion of a droplet in shear flow including wall effects, Interna- tional Journal of Multiphase Flow 16, 305 (1990)
[4] F. Guglietta, F. Pelusi “A unified analytical prediction for steady-state behavior of confined drop with interface viscosity under shear flow”, *in peer review*, (2024)

Analytical results on drop deformation

	Without interface viscosity	With interface viscosity
Unconfined	$D_T = \frac{19\lambda + 16}{16\lambda + 16}\mathrm{Ca}$ [1]	$D_F = \frac{19\lambda + 16 + 32\mathrm{Bq}}{(16\lambda + 16 + 32\mathrm{Bq})\sqrt{\mathrm{Ca}^{-2} + \frac{19F}{20}(\lambda + 2\mathrm{Bq})}}$ [2]
Confined	$D_{SH} = D_T(\lambda, \mathrm{Ca})\Psi(\lambda, \hat{h}, \chi)$ [3]	$D = D_F(\lambda, \mathrm{Ca}, \mathrm{Bq})\Psi(\lambda, \hat{h}, \chi)$ [4]

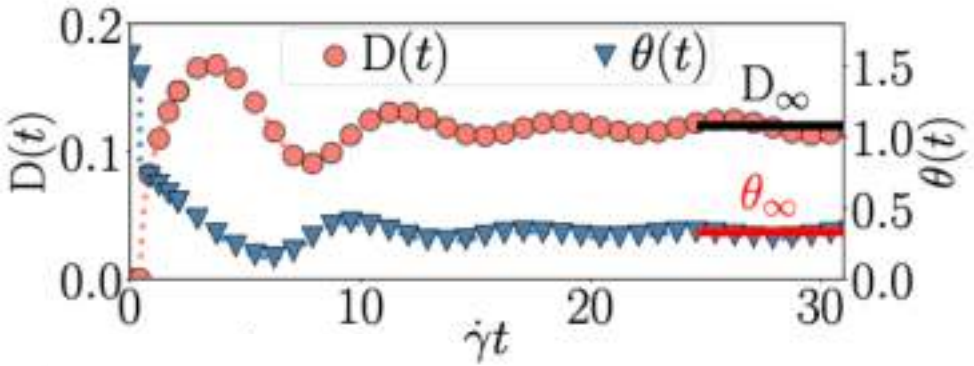
$$\Psi(\lambda, \hat{h}, \chi) = 1 + C_s(\hat{h})\left(\frac{\chi}{2}\right)^3 \frac{1 + 2.5\lambda}{1 + \lambda}$$

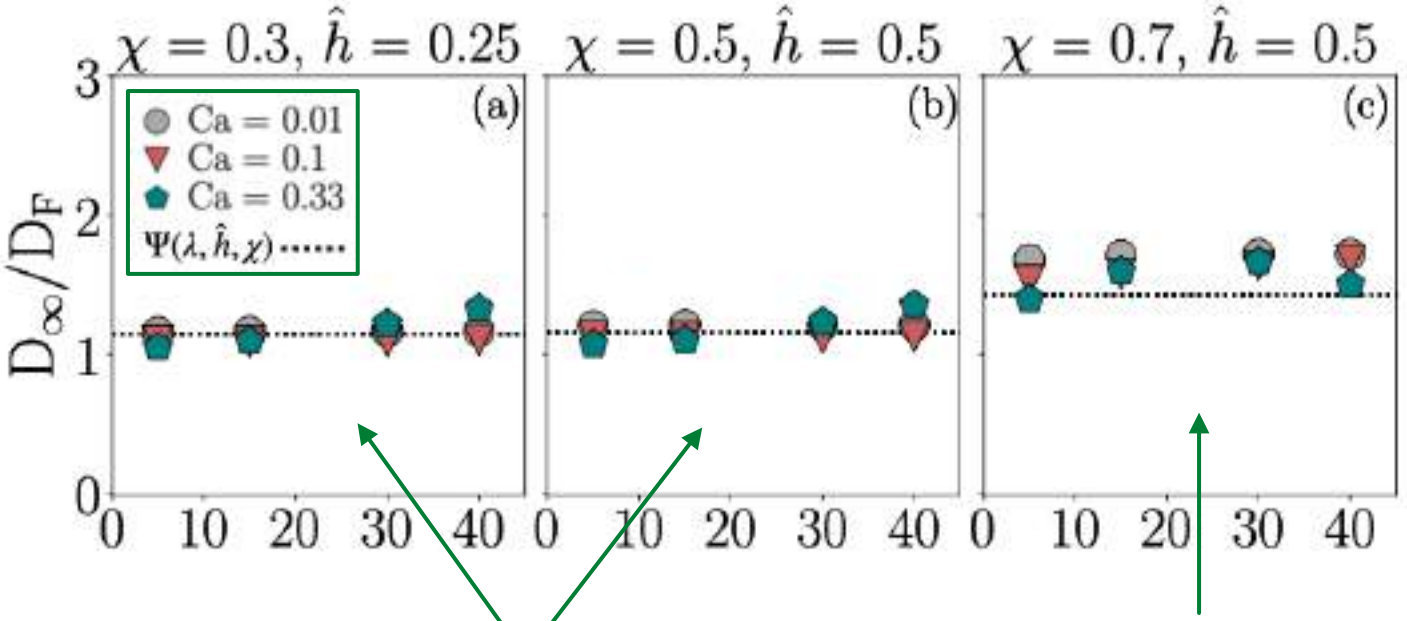
$$F = 1 - \frac{9\lambda + 18\mathrm{Bq} - 2}{8(\lambda + 2\mathrm{Bq})^2}$$

[1] G. I. Taylor, The formation of emulsions in definable fields of flow, Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character 146, 501 (1934).
[2] R. W. Flumerfelt, Effects of dynamic interfacial proper- ties on drop deformation and orientation in shear and extensional flow fields, Journal of Colloid and Interface Science 76, 330 (1980).
[3] M. Shapira and S. Haber, Low Reynolds number motion of a droplet in shear flow including wall effects, Interna- tional Journal of Multiphase Flow 16, 305 (1990)
[4] F. Guglietta, F. Pelusi “A unified analytical prediction for steady-state behavior of confined drop with interface viscosity under shear flow”, *in peer review*, (2024)

Deformation can be factorised

$$D \stackrel{?}{=} D_F(\lambda, \mathrm{Ca}, \mathrm{Bq})\Psi(\lambda, \hat{h}, \chi)$$





Do not depend on Bq

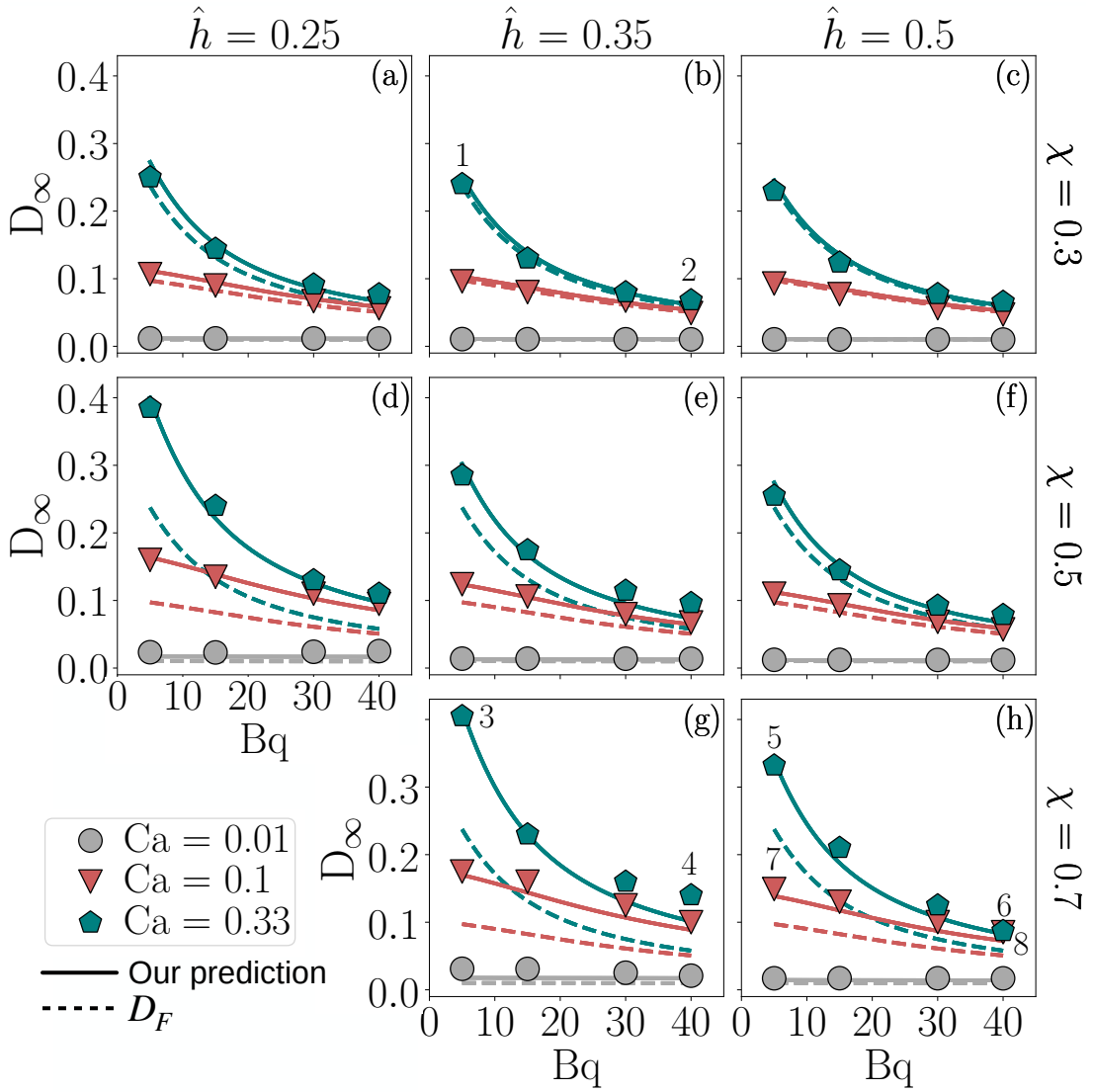
Slightly depends on Bq when Ca=0.33

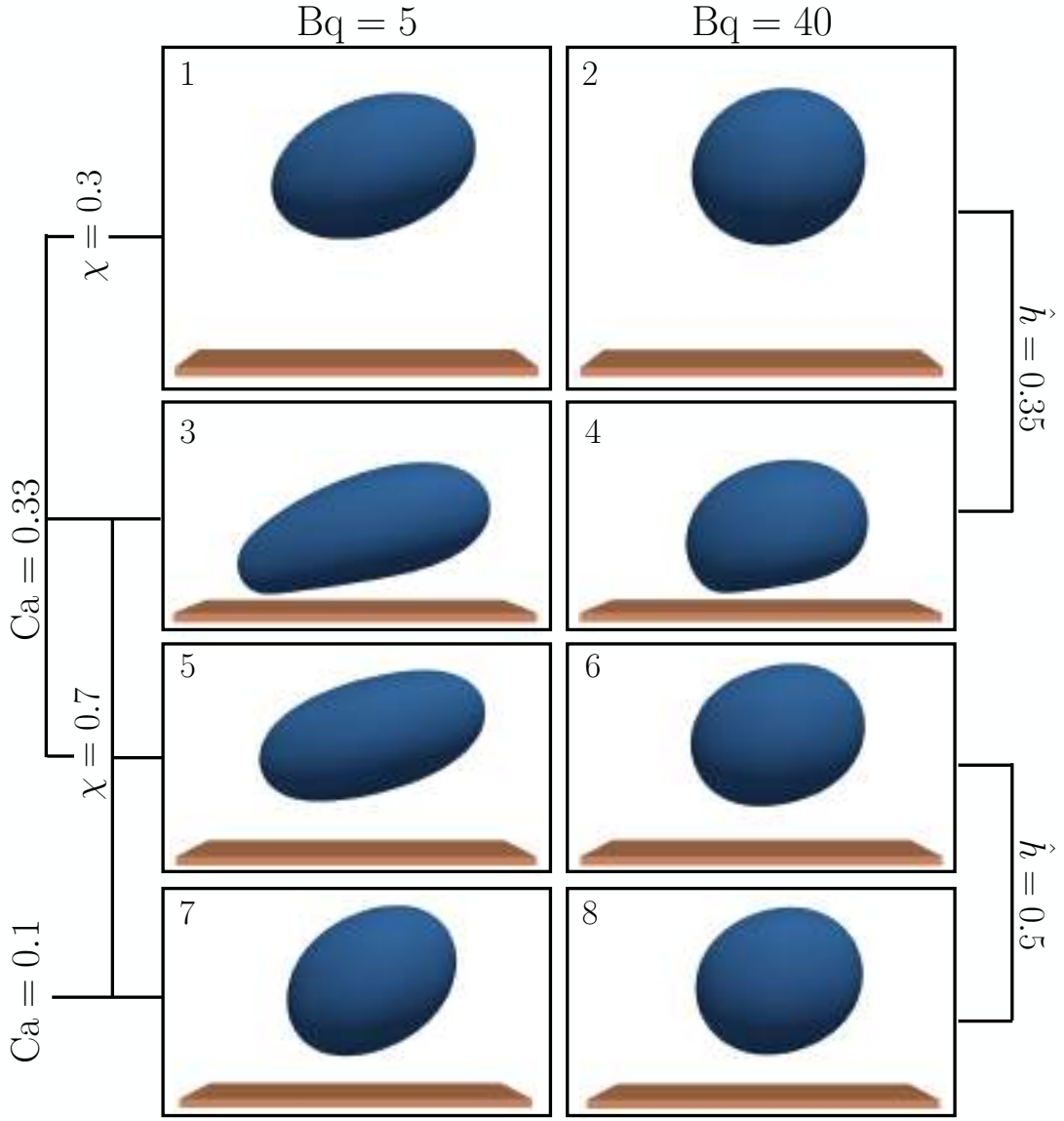
$$D_F = \frac{19\lambda + 16 + 32\mathrm{Bq}}{(16\lambda + 16 + 32\mathrm{Bq})\sqrt{\mathrm{Ca}^{-2} + \frac{19F}{20}(\lambda + 2\mathrm{Bq})}}$$

$$F = 1 - \frac{9\lambda + 18\mathrm{Bq} - 2}{8(\lambda + 2\mathrm{Bq})^2}$$

$$\Psi(\lambda, \hat{h}, \chi) = 1 + C_s(\hat{h})\left(\frac{\chi}{2}\right)^3 \frac{1 + 2.5\lambda}{1 + \lambda}$$

Analytical results vs. IB-LB simulations





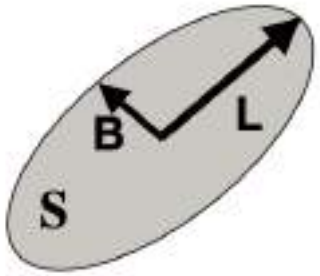
TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Reduced order model for drop deformation in stationary flows

[7] D. Taglienti, **F. Guglietta** & M. Sbragaglia “Reduced model for droplet dynamics in shear flows at finite capillary numbers”, Physical Review Fluids, 8(1), 013603, (2023)

Reduced order model for drop dynamics

- Maffettone and Minale (MM) model [1]:
  - Drop shape given by tensor $\mathbf{S}$
  - Shape always ellipsoidal
  - Two parameters (f_1 and f_2)
  - One-way coupling



Deformation:

$$D = \frac{L - B}{L + B}$$

[1] **P. L. Maffettone and M. Minale,** "Equation of change for ellipsoidal drops in viscous flow", *Journal of non-Newtonian fluid mechanics* 78.2-3, 227-241, 1998

$$\frac{\partial \mathbf{S}}{\partial t} - [\boldsymbol{\Omega} \cdot \mathbf{S} + \mathbf{S} \cdot \boldsymbol{\Omega}] = -f_1[\mathbf{S} - g(\mathbf{S})\mathbf{I}] + f_2[\mathbf{E} \cdot \mathbf{S} + \mathbf{S} \cdot \mathbf{E}]$$

Vorticity tensor

$$\boldsymbol{\Omega} = \frac{1}{2}[\nabla \mathbf{u} - \nabla \mathbf{u}^T]$$

Volume conservation

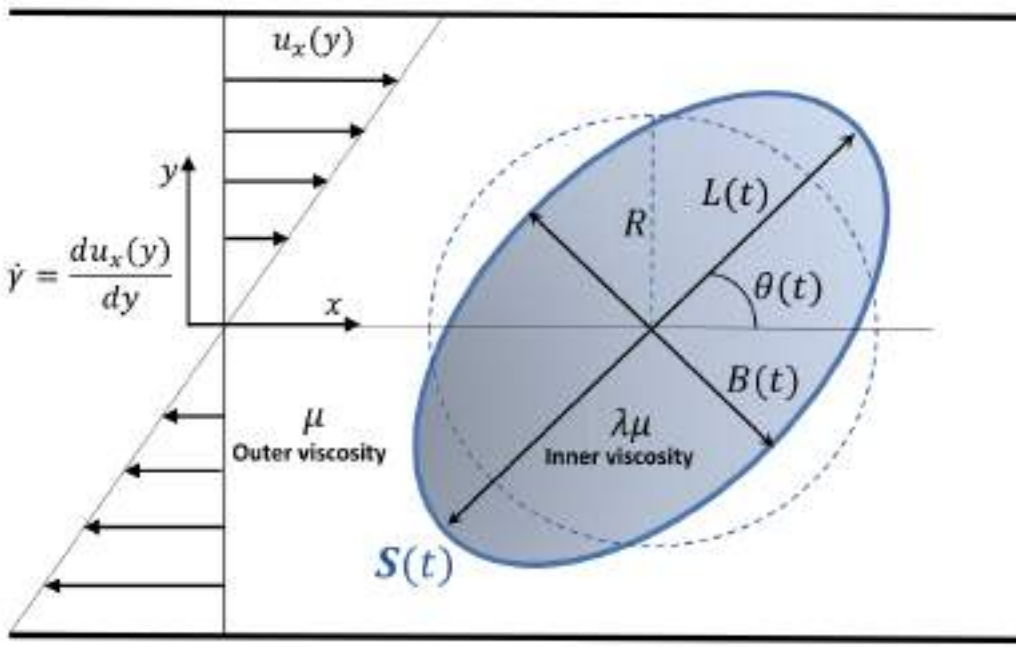
$$g(\mathbf{S}) = \frac{6 \det(\mathbf{S})}{\mathrm{tr}(\mathbf{S})^2 - \mathrm{tr}(\mathbf{S}^2)}$$

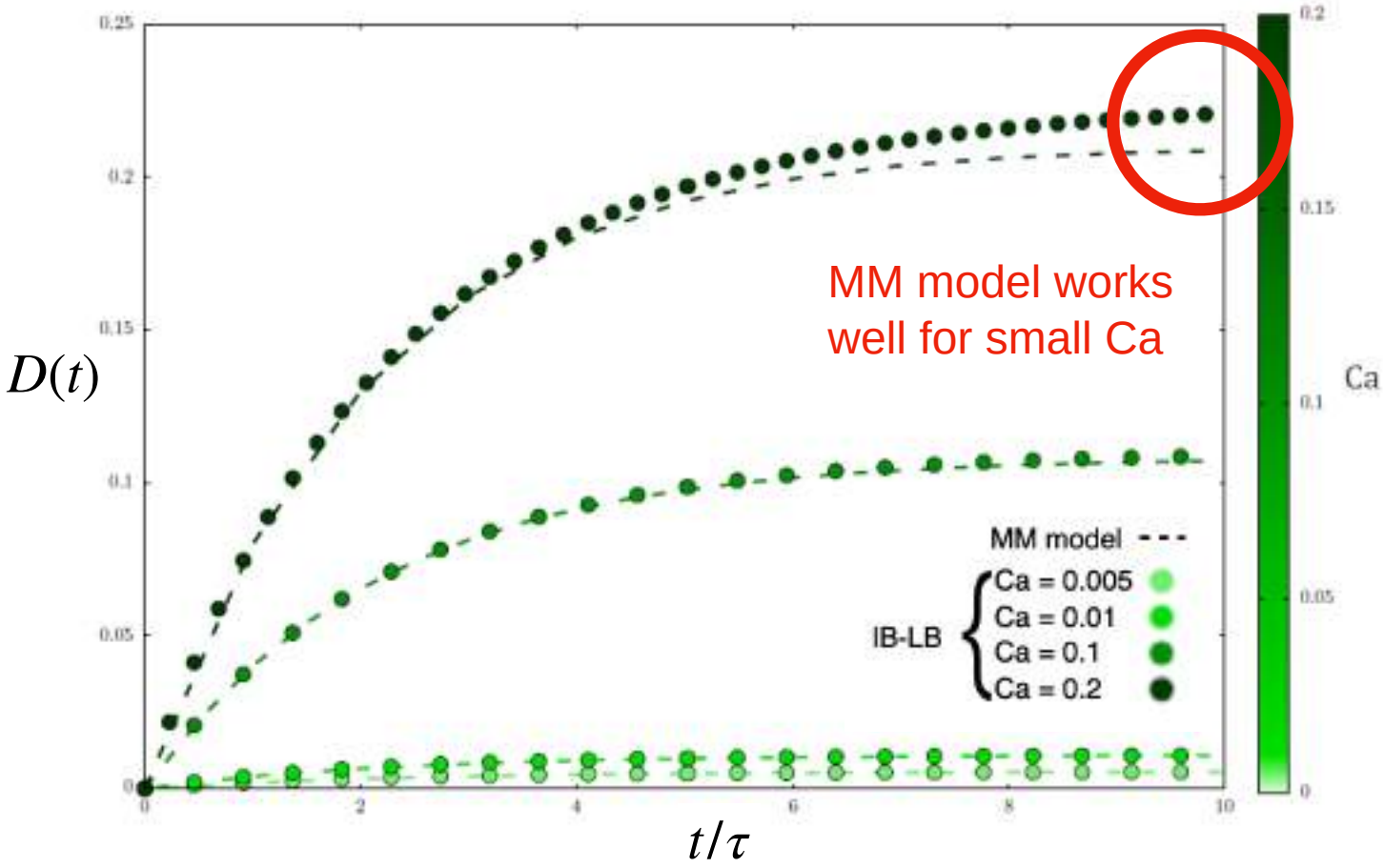
Strain tensor

$$\mathbf{E} = \frac{1}{2}[\nabla \mathbf{u} + \nabla \mathbf{u}^T]$$

$$f_1 = f_1(\lambda) = \frac{40(\lambda + 1)}{(2\lambda + 3)(19\lambda + 16)} \qquad f_2 = f_2(\lambda) = \frac{5}{2\lambda + 3}$$

MM model vs. IB-LB simulations





Drop breakup

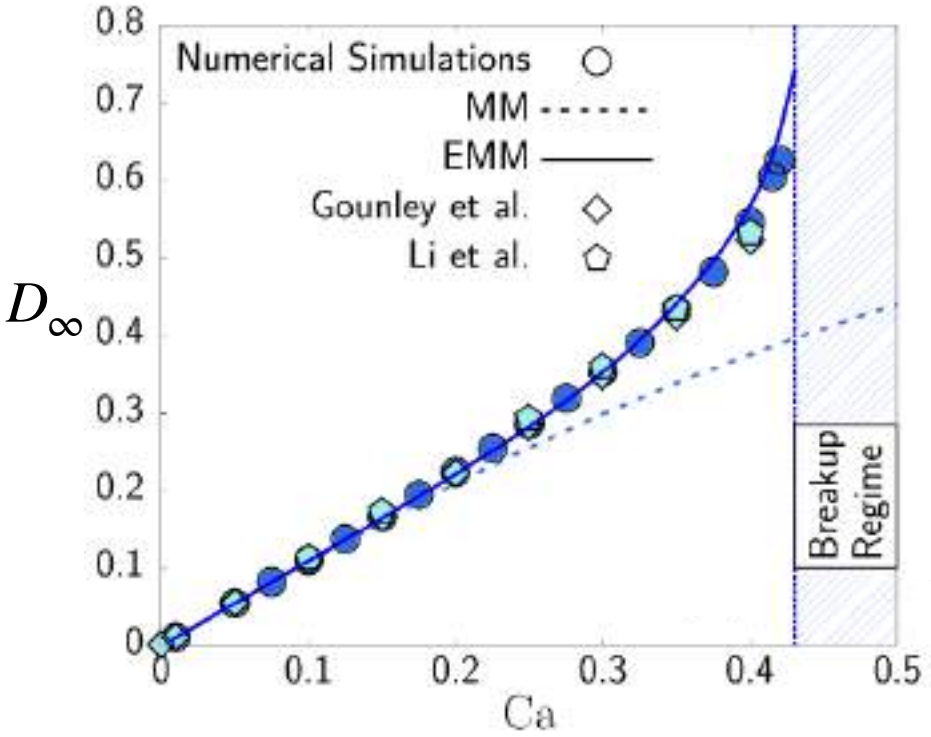
LB (Shan-Chen) simulations of drop breakup in confined shear flow [1]

t_1 t_2 t_3

- IB-LBM cannot simulate drops breakup.

[1] Gupta, A., & Sbragaglia, M. (2014). Deformation and breakup of viscoelastic droplets in confined shear flow. *Physical Review E*, *90*(2), 023305.

MM model vs. IB-LB simulations



- We proposed an extended MM (EMM) model:

$$\frac{\partial \mathbf{S}}{\partial t} - [\boldsymbol{\Omega}\cdot\mathbf{S} + \mathbf{S}\cdot\boldsymbol{\Omega}] = -f_1^{(EMM)}[\mathbf{S} - g(\mathbf{S})\mathbf{I}] + f_2^{(EMM)}[\mathbf{E}\cdot\mathbf{S} + \mathbf{S}\cdot\mathbf{E}]$$

$$f_{1,2}^{(EMM)} = \sum_i a_i^{(1,2)}(\lambda) Ca^i$$ [Polynomial expansion in Ca]

	$\lambda = 1$	
i	$a_i^{(1)}$	$a_i^{(2)}$
0	0.457	1
1	0.235	0
2	−4.546	0
3	20.536	0
4	−34.798	0

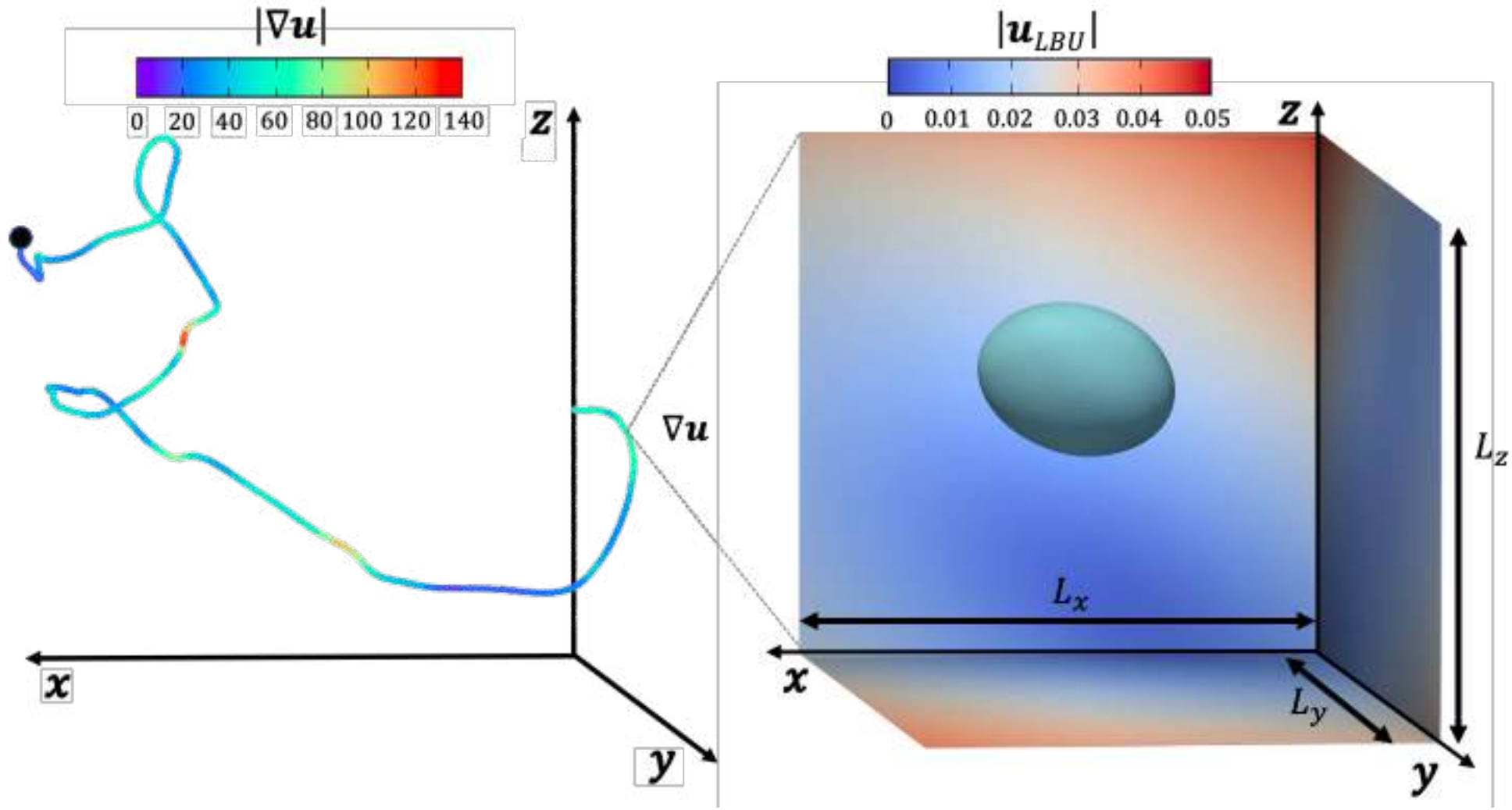
[1] Gounley, J., Boedec, G., Jaeger, M., & Leonetti, M. (2016). Influence of surface viscosity on droplets in shear flow. *Journal of Fluid Mechanics*, *791*, 464-494.
[2] J. Li, Y. Y. Renardy, and M. Renardy, Numerical simulation of breakup of a viscous drop in simple shear flow through a volume-of-fluid method, Phys. Fluids 12, 269 (2000).

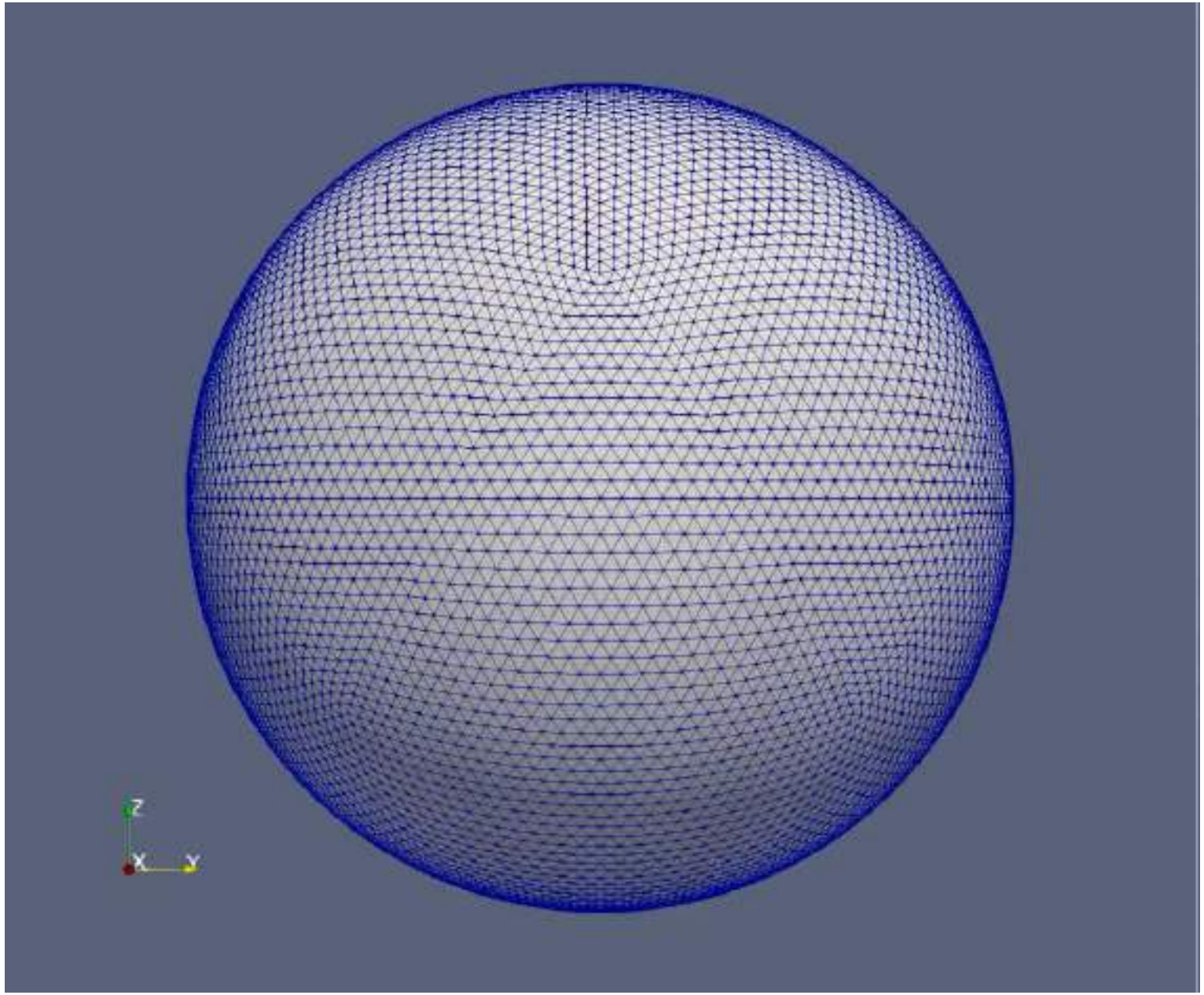
TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Drop dynamics in time-dependent flows

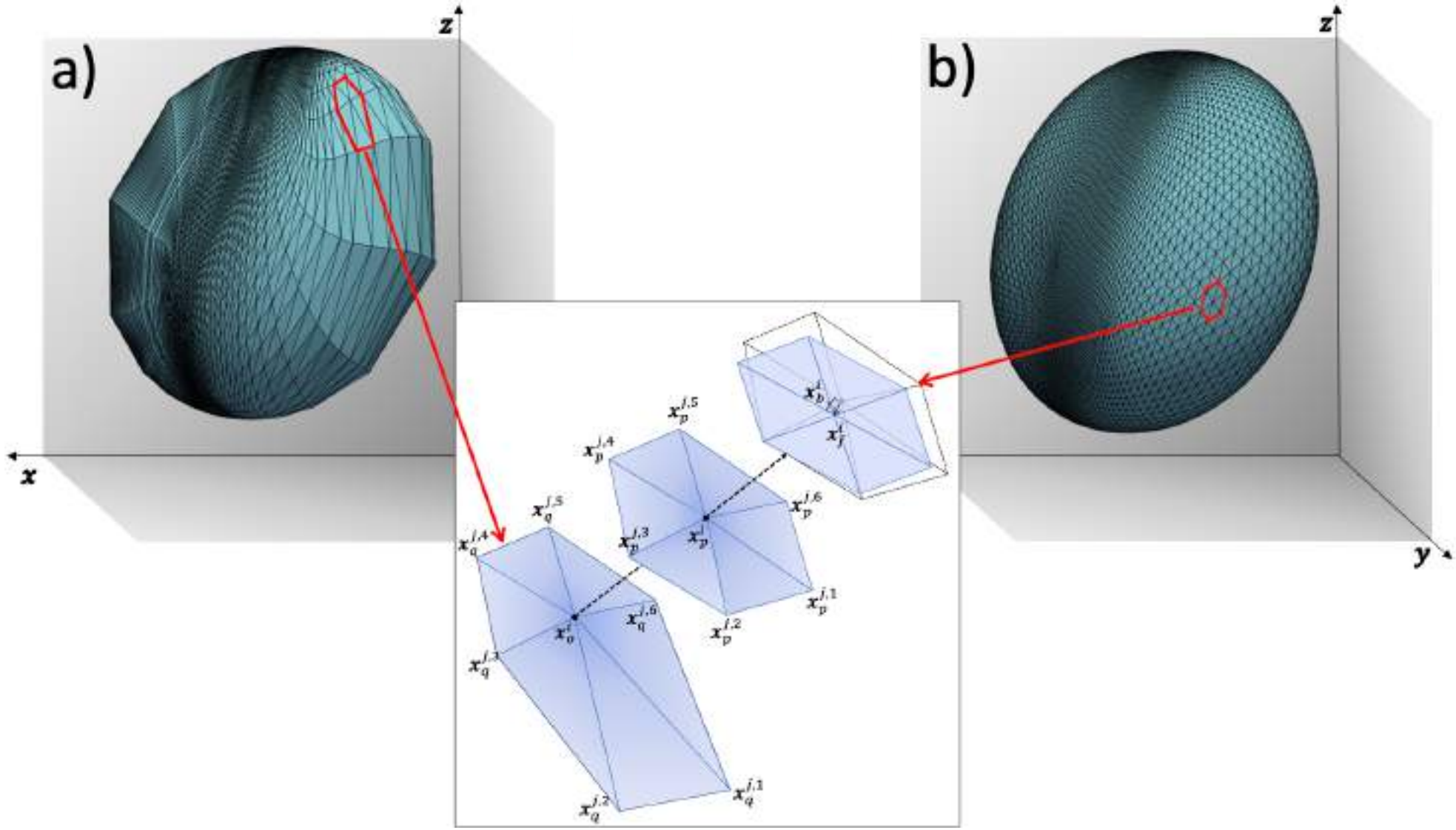
[8] D. Taglienti, **F. Guglietta** & M. Sbragaglia “Droplet dynamics in homogeneous isotropic turbulence with the immersed boundary-lattice Boltzmann method”, *in peer review, (2024)*

From Lagrangian turbulence to drop dynamics

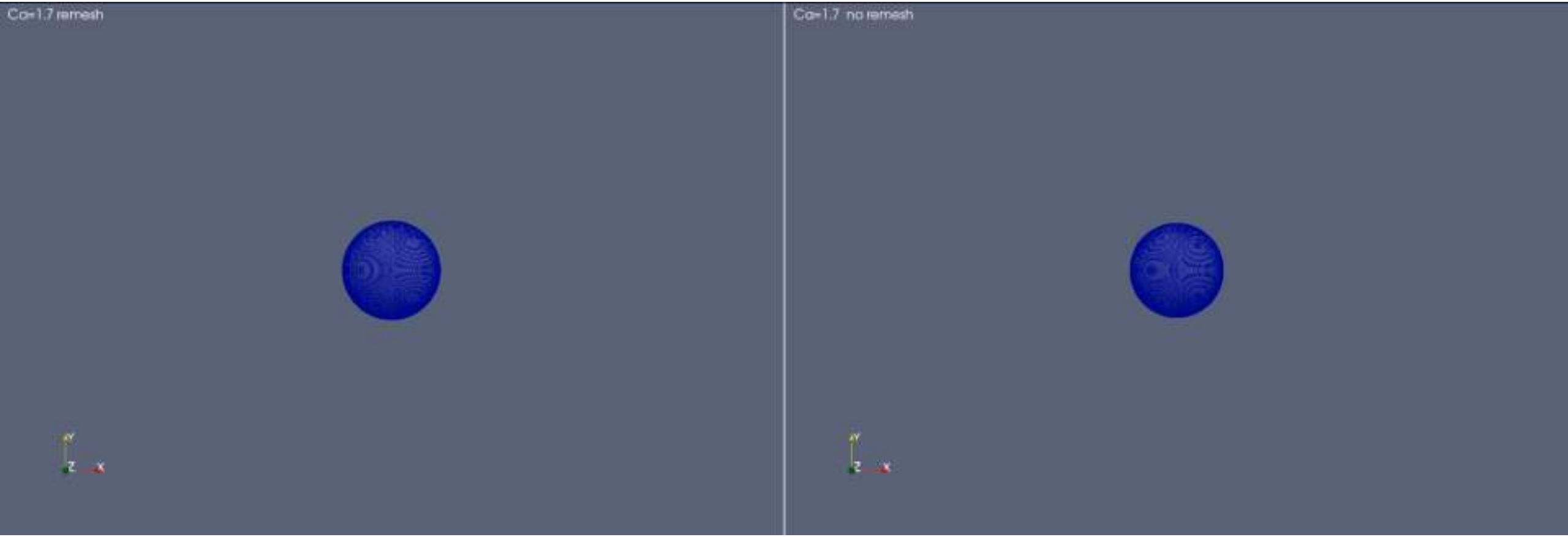



Z
X
Y

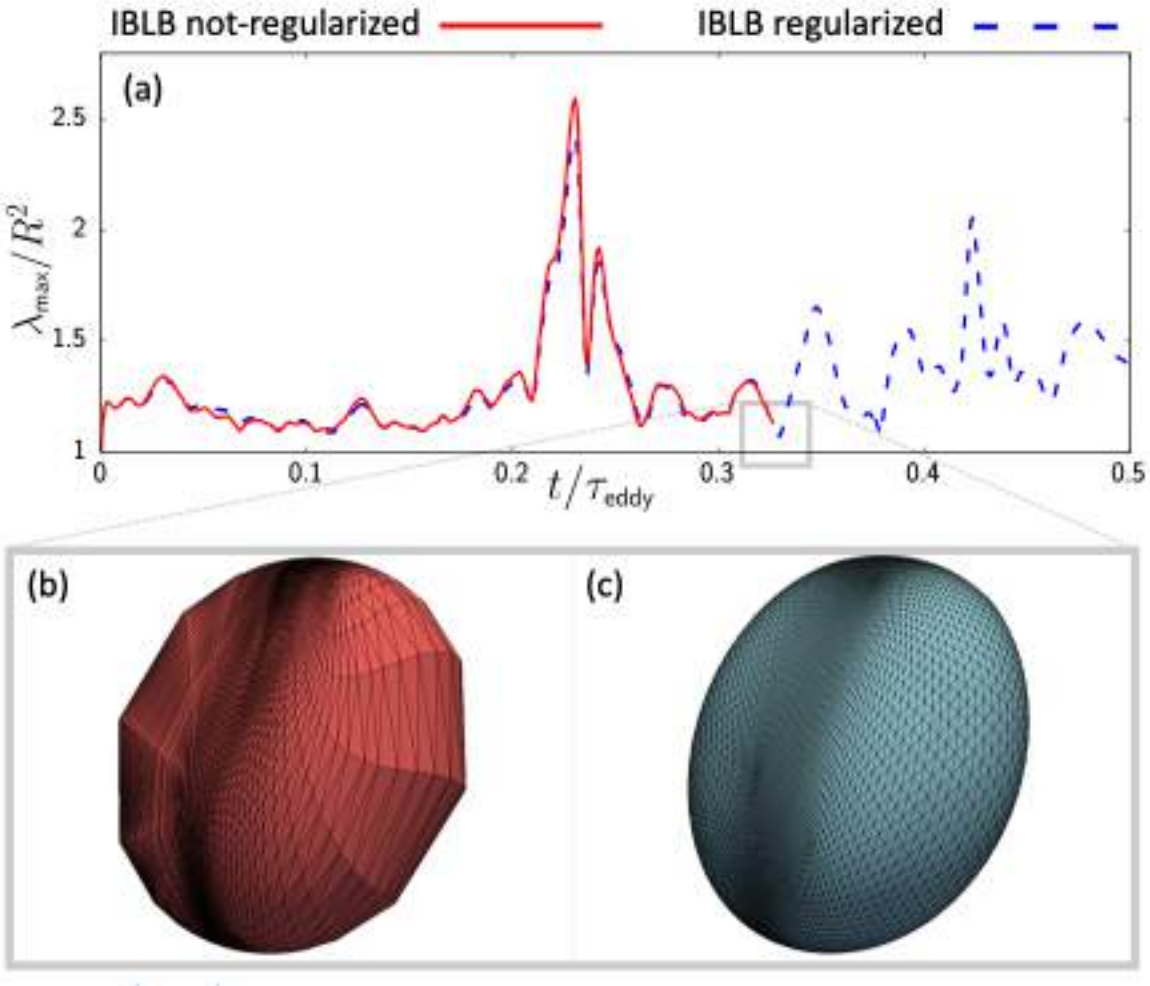
Laplacian smoothing



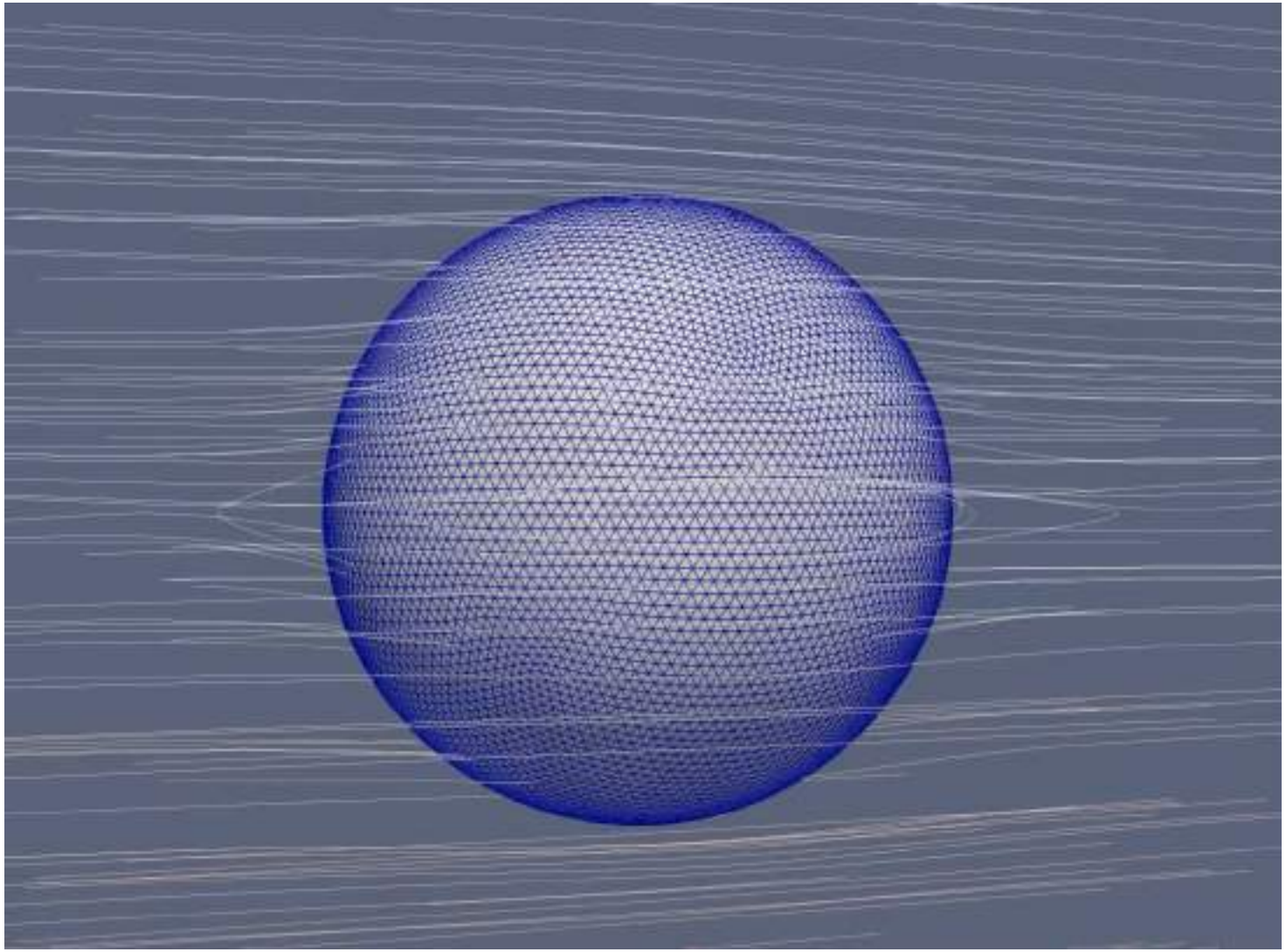
Effect of smoothing



Effect of smoothing



Final result



AQTIVATE

European Research Council
Established by the European Commission
Supporting top researchers from anywhere in the world

TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

A crash course on GPU programming in CUDA Fortran

Fabio Guglietta

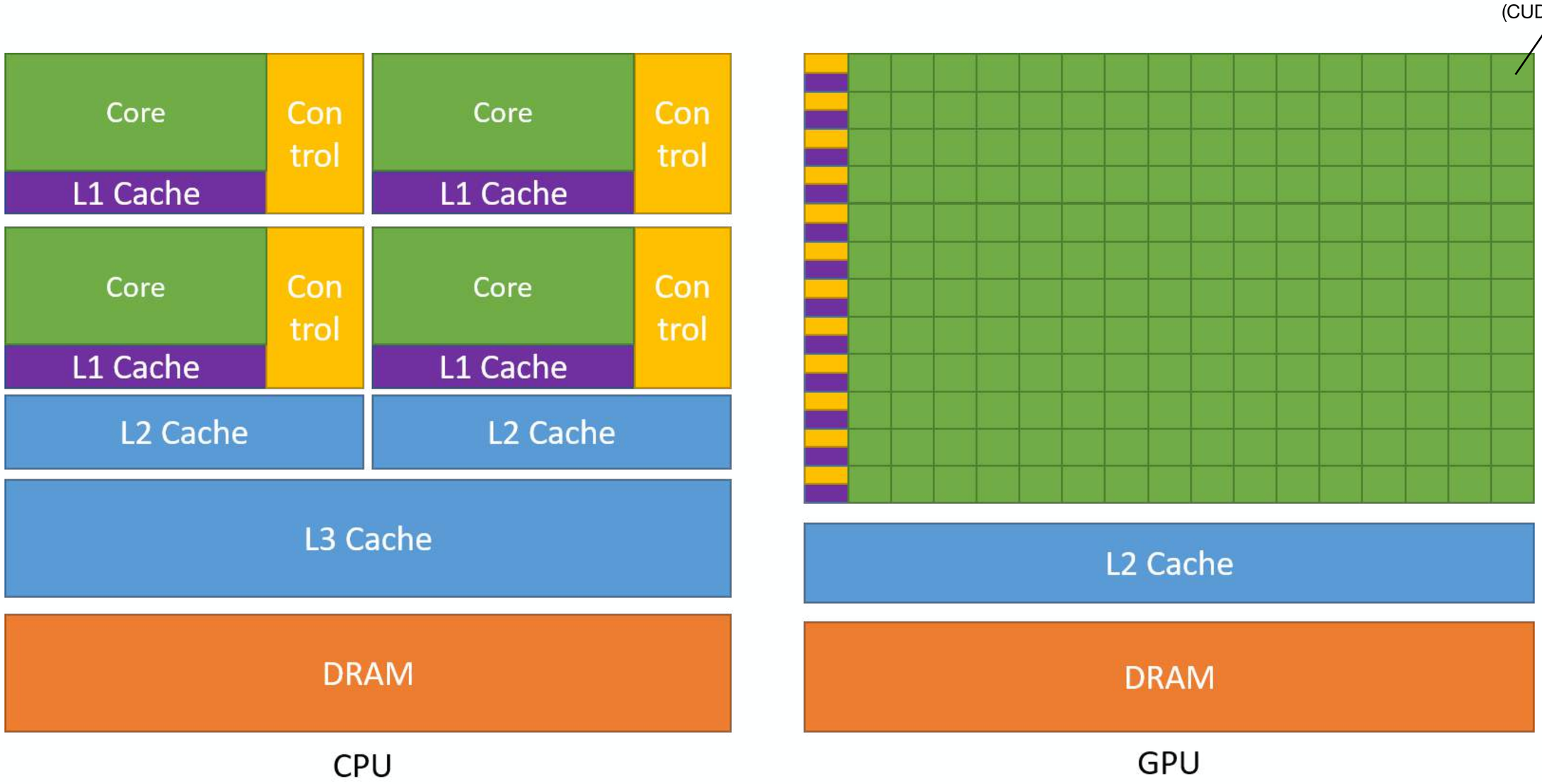
A crash course on GPU programming

in CUDA (Fortran)

Fabio Guglietta

CPU vs GPU in a very small nutshell

fabio.guglietta@roma2.infn.it



CPU code: serial

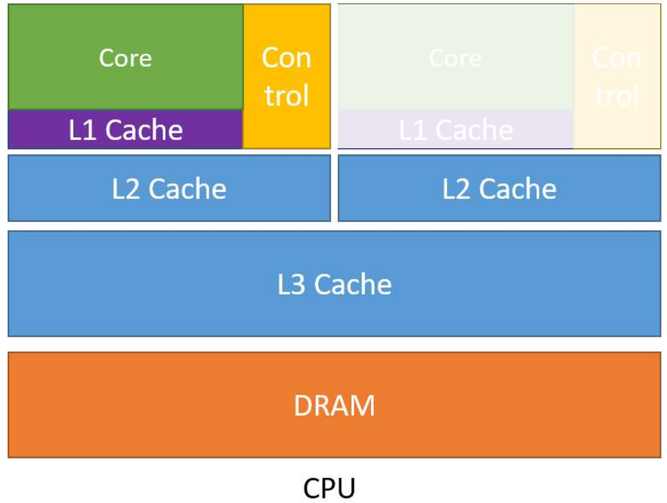
fabio.guglietta@roma2.infn.it

```
 1 module simpleOps_m
 2 contains
 3   subroutine increment(a,b)
 4     implicit none
 5     integer, intent(inout) :: a(:)
 6     integer, intent(in) :: b
 7     integer :: i,n
 8     n = size(a)
 9     do i = 1, n
10       a(i) = a(i) + b
11     end do
12   end subroutine
13 end module
14
15 program ex1
16   use simpleOps_m
17   implicit none
18   integer, parameter :: n = 256
19   integer :: a(n), b
20
21   a = 1
22   b = 3
23   call increment(a,b)
24
25   if(any(a/=4))then
26     print*, "Program Failed"
27   else
28     print*, "Program Passed"
29   endif
30 end program
```

```
do i = 1, size(a)
```

$$b = \boxed{3}$$

$$a = \boxed{4}\ \boxed{4}\ \boxed{4}\ \boxed{4}\ \cdots\ \boxed{4}\ \boxed{4}\ \boxed{4}\ \boxed{4}$$



CPU code: parallel (MPI)

fabio.guglietta@roma2.infn.it

```
1  module simpleOps_m
2  contains
3    subroutine increment(a,b)
4      implicit none
5      integer, intent(inout) :: a(:)
6      integer, intent(in) :: b
7      integer :: i,n
8      n = size(a)
9      do i = 1, n
10       a(i) = a(i) + b
11     end do
12   end subroutine
13 end module
14
15 program ex1
16   use mpi
17   use simpleOps_m
18   implicit none
19   integer, parameter  :: n = 256
20   integer             :: n_red
21   integer             :: num_procs
22   integer             :: a_global(n)
23   integer,allocatable :: a(:)
24   integer             :: b
25   integer             :: myrank
26   integer             :: ierr
27
28   !NEW SECTION
29   call MPI_init(ierr)
30   call MPI_comm_size(MPI_COMM_WORLD, num_procs,ierr)
31   call MPI_comm_rank(MPI_COMM_WORLD,myrank,ierr)
32   n_red = n/num_procs
33   allocate(a(n_red))
34   if(mod(n,num_procs).ne.0)stop "n = 256: use a number of processor that divides n"
35   !
36   a = 1
37   b = 3
38   call increment(a,b)
39   !
40   !NEW SECTION
41   a_global = 0
42   call MPI_gather(a,n_red,MPI_INT,a_global,n_red,MPI_INT,0,MPI_COMM_WORLD,ierr)
43   !
44   if(myrank.eq.0)then
45     if(any(a_global/=4))then
46       print*, "Program Failed"
47     else
48       print*, "Program Passed"
49     endif
50   endif
51   call MPI_Finalize(ierr)
52 end program
```

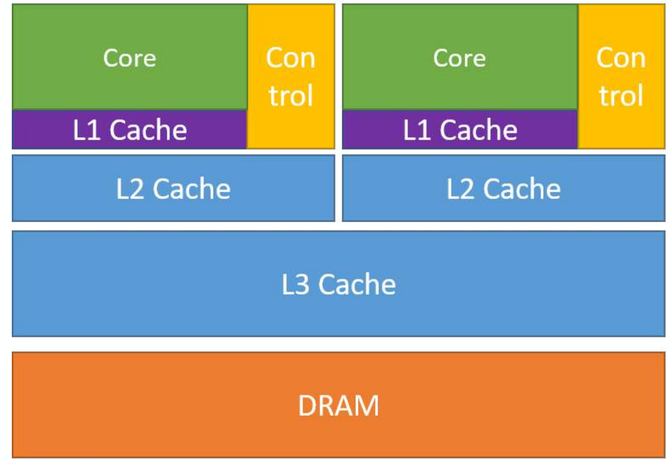
`do i = 1, size(a)` | `do i = 1, size(a)` | `do i = 1, size(a)`

$b = \boxed{3}$ $b = \boxed{3}$

$a = \boxed{4}\boxed{4}\boxed{\cdots}$ $a = \boxed{4}\boxed{4}\boxed{\cdots}$ $a = \boxed{4}\boxed{4}\boxed{\cdots}$

$a_{global} = \boxed{4}\boxed{4}\boxed{\cdots}\boxed{4}\boxed{4}\boxed{\cdots}\boxed{4}\boxed{4}\boxed{\cdots}$

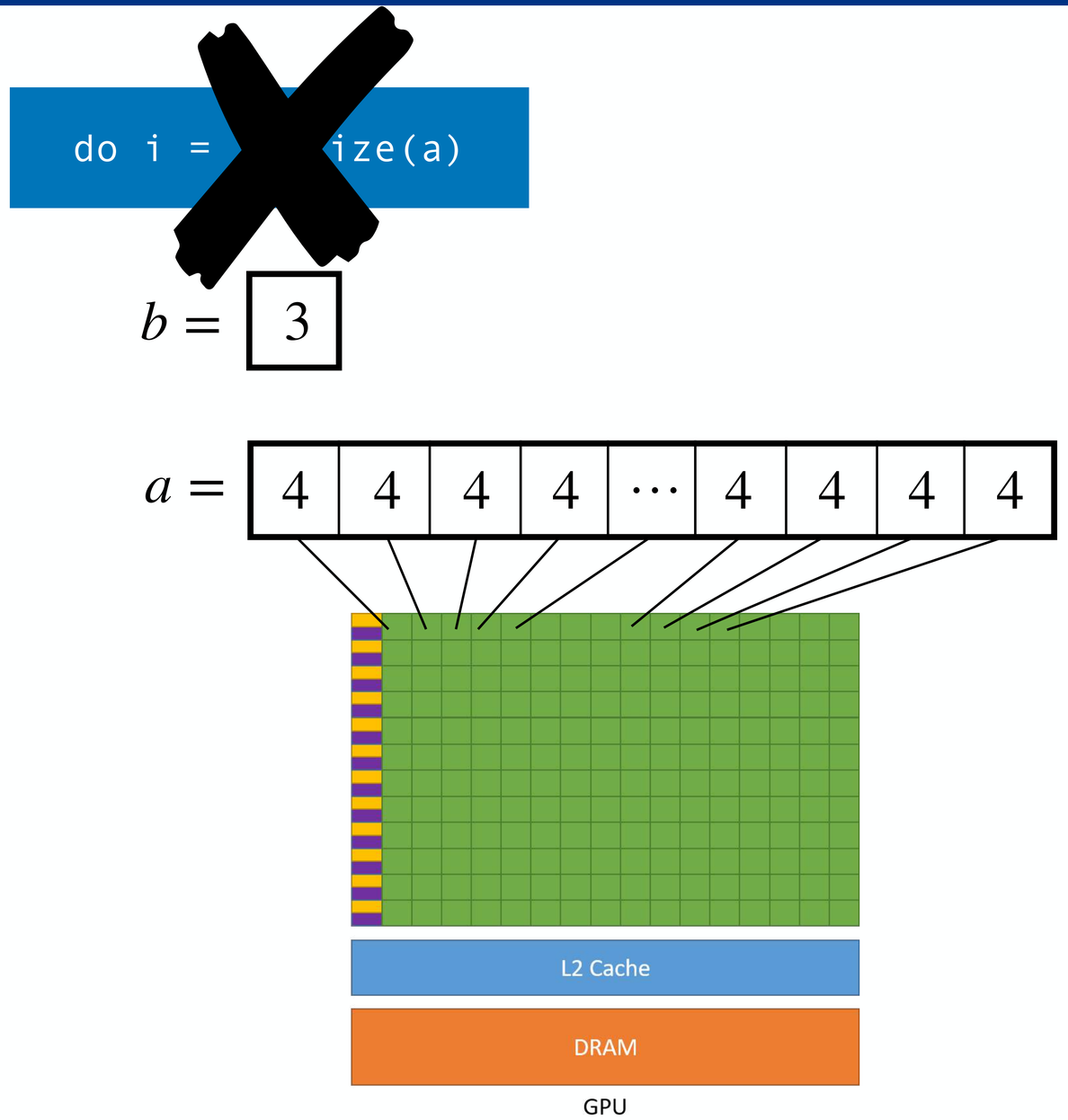
How can we make it faster?



GPU code: CUDA

fabio.guglietta@roma2.infn.it

```
1  module simpleOps_m
2  contains
3    attributes(global) subroutine increment(a, b)
4      implicit none
5      integer, intent(inout) :: a(:)
6      integer, value :: b
7      integer :: i
8
9      i = threadIdx%x
10     a(i) = a(i)+b
11
12   end subroutine increment
13 end module simpleOps_m
14
15 program incrementTest
16   use cudafor
17   use simpleOps_m
18   implicit none
19   integer, parameter :: n = 256
20   integer :: a(n), b
21   integer, device :: a_d(n)
22
23   a = 1
24   b = 3
25
26   a_d = a
27   call increment<<<1,n>>>(a_d, b)
28   a = a_d
29
30   if (any(a /= 4)) then
31      write(*,*) '**** Program Failed ****'
32   else
33      write(*,*) 'Program Passed'
34   endif
35 end program incrementTest
```



GPU code: CUDA


```
1  module simpleOps_m
2  contains
3    attributes(global) subroutine increment(a, b)
4      implicit none
5      integer, intent(inout) :: a(:)
6      integer, value :: b
7      integer :: i
8
9      i = threadIdx%x
10     a(i) = a(i)+b
11
12   end subroutine increment
13 end module simpleOps_m
14
15 program incrementTest
16   use cudafor
17   use simpleOps_m
18   implicit none
19   integer, parameter :: n = 256
20   integer :: a(n), b
21   integer, device :: a_d(n)
22
23   a = 1
24   b = 3
25
26   a_d = a
27   call increment<<<1,n>>>(a_d, b)
28   a = a_d
29
30   if (any(a /= 4)) then
31      write(*,*) '**** Program Failed ****'
32   else
33      write(*,*) 'Program Passed'
34   endif
35 end program incrementTest
```

The attribute **global** indicates that the code is to **run on the device** but is **called from the host**.
(The term global, as with all **subroutine attributes**, describes the scope; the subroutine is seen from both the host and the device.)

In the CUDA Fortran version, the subroutine is executed by many GPU threads concurrently. Each **thread identifies itself via the built-in threadIdx** variable that is available in all device code and uses this variable as an index of the array.

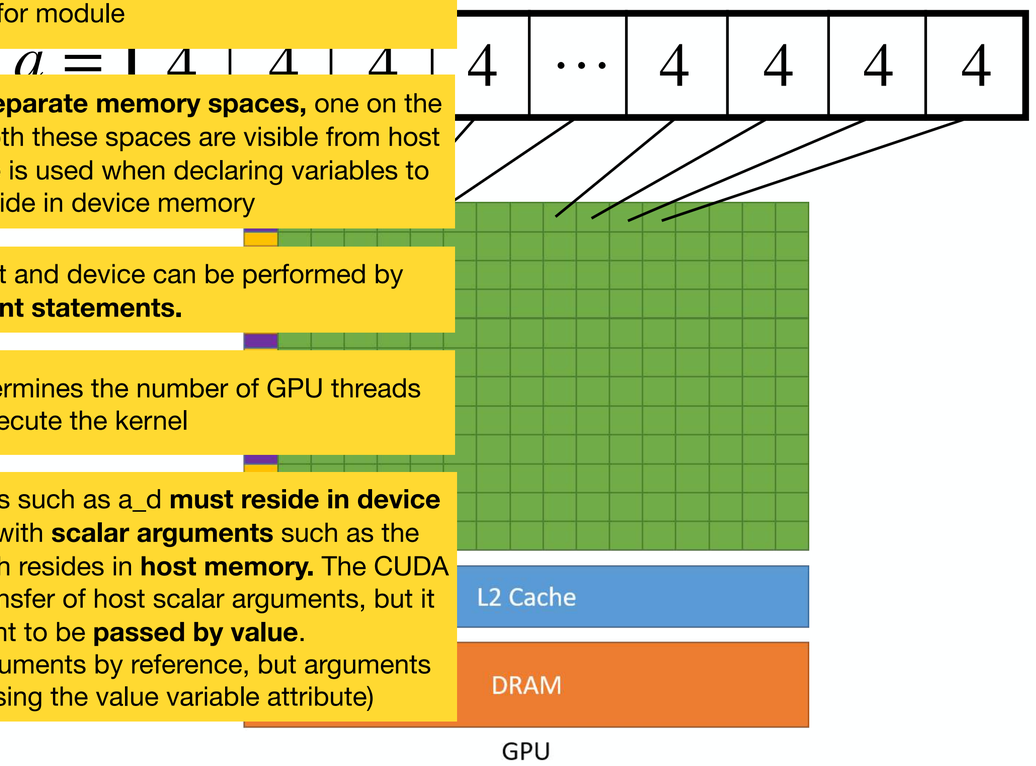
The CUDA Fortran definitions and derived types are contained in the cudafor module

CUDA Fortran deals with **two separate memory spaces,** one on the **host** and one on the **device**. Both these spaces are visible from host code, and the **device attribute** is used when declaring variables to indicate they reside in device memory

Data transfers between host and device can be performed by **assignment statements.**

Execution configuration determines the number of GPU threads used to execute the kernel

Although kernel array arguments such as a_d **must reside in device memory**, this is not the case with **scalar arguments** such as the second kernel argument b, which resides in **host memory.** The CUDA runtime will take care of the transfer of host scalar arguments, but it expects the argument to be **passed by value**.
(By default, Fortran passes arguments by reference, but arguments can be passed by value using the value variable attribute)



Pass by value vs pass by reference

- Mechanism of **copying** the function parameter value to another variable
- **Changes** made inside the function **are not reflected** in the original value
- **Makes a copy** of the actual parameter
- Function **gets a copy** of the actual content
- **Requires more memory**
- **Requires more time as it involves copying values**

- Mechanism of **passing** the actual parameters to the function
- **Changes** made inside the function **are reflected** in the original value
- **Address** of the actual parameter passes to the function
- Function **accesses** the original variable's content
- **Requires less memory**
- **Requires a less amount of time as there is no copying**

Synchronisation Host-Device


```
 1 module simpleOps_m
 2 contains
 3   attributes(global) subroutine increment(a, b)
 4     implicit none
 5     integer, intent(inout) :: a(:)
 6     integer, value :: b
 7     integer :: i
 8
 9     i = threadIdx%x
10     a(i) = a(i)+b
11
12   end subroutine increment
13 end module simpleOps_m
14
15 program incrementTest
16   use cudafor
17   use simpleOps_m
18   implicit none
19   integer, parameter :: n = 256
20   integer :: a(n), b
21   integer, device :: a_d(n)
22
23   a = 1
24   b = 3
25
26   a_d = a
27   call increment<<<1,n>>>(a_d, b)
28   a = a_d
29
30   if (any(a /= 4)) then
31      write(*,*) '**** Program Failed ****'
32   else
33      write(*,*) 'Program Passed'
34   endif
35 end program incrementTest
```

For this program to execute correctly, we need to know that:

- the **host-to-device data transfer** on line 26 completes before the kernel begins execution and
- that the **kernel completes** before the device-to-host transfer on line 28 commences.

We are assured of such behaviour because the **data transfers** via assignment statements on lines 26 and 28 are **blocking or synchronous transfers**.

Such **transfers do not initiate until all previous operations on the GPU are complete,** and subsequent operations on the GPU will not begin until the data transfer is complete. The blocking nature of these data transfers is helpful in implicitly synchronizing the CPU and GPU.

KEEP IN MIND!

The **data transfers** via assignment statements are blocking or synchronous operations, whereas **kernel launches** are nonblocking or asynchronous.

Take-home messages

fabio.guglietta@roma2.infn.it

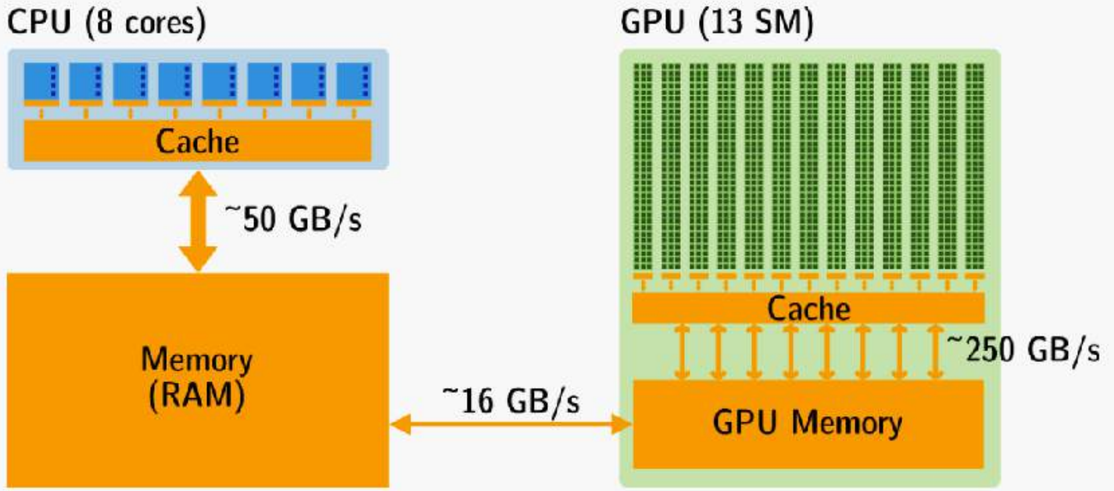
- How to structure a GPU-code:
  - Declare host (CPU) and device (GPU) variables.
  - Transfer from CPU -> GPU
  - Invoke CUDA Kernel
  - Transfer from GPU -> CPU (if necessary!!!)
    [**minimize** data transfers between **host and device** whenever possible and, when such transfers are necessary, make sure they are **optimized**!]

```
program incrementTest
  use cudafor
  use simpleOps_m
  implicit none
  integer, parameter :: n = 256
  integer :: a(n), b
  integer, device :: a_d(n)
```

```
a_d = a
call increment<<<1,n>>>(a_d, b)
a = a_d
```

KEEP IN MIND!
The **data transfers** via assignment statements are blocking or synchronous operations, whereas **kernel launches** are nonblocking or asynchronous.

GPU programming model



"Offload" model of programming

- CPU starts program (runs `main()`)
- CPU copies data to GPU memory (over e.g. PCIe, ~16 GB/s)
- CPU dispatches "kernels" for execution on GPU
  - Kernels read/write to GPU memory (~250 GB/s)
  - Kernels run on GPU threads (thousands) which share *fast* memory [$\mathrm{O}(10)$ times faster compared to GPU memory]
- Kernel completes; CPU copies data back from GPU (over e.g. PCIe, ~16 GB/s)

How easy is it?

Did you understand how to do it?

So do it!

Exercise

- Write a code running on the GPU which increments an array of dimension 256 initialised to 1. Check for correctness.

$$\vec{a} \rightarrow \vec{a} + b$$

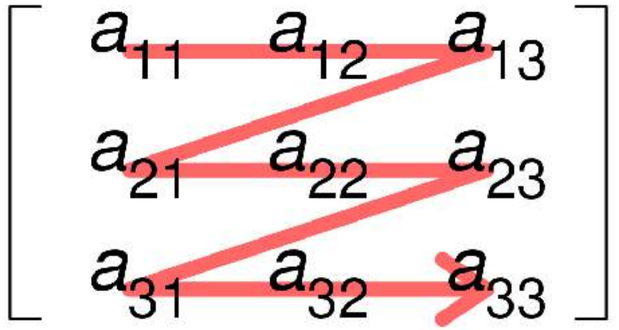
- File format: example.cuf
- To compile it: pgf90 example.cuf
- ssh -J guglietta@newturb.roma2.infn.it guglietta@hydrosoft

Multidimensional arrays

fabio.guglietta@roma2.infn.it

```fortran
module simpleOps_m
contains
  subroutine increment(a,b)
    implicit none
    integer, intent(inout) :: a(:,:)
    integer, intent(in) :: b
    integer :: i,j,nx,ny
    nx = size(a,1)
    ny = size(a,2)
    do j = 1, ny
      do i = 1, nx
        a(i,j) = a(i,j) + b
      end do
    end do
  end subroutine
end module

program ex1
  use simpleOps_m
  implicit none
  integer, parameter :: nx = 1024
  integer, parameter :: ny = 512
  integer :: a(nx,ny), b

  a = 1
  b = 3
  call increment(a,b)

  if(any(a/=4))then
    print*, "Program Failed"
  else
    print*, "Program Passed"
  endif
end program
```

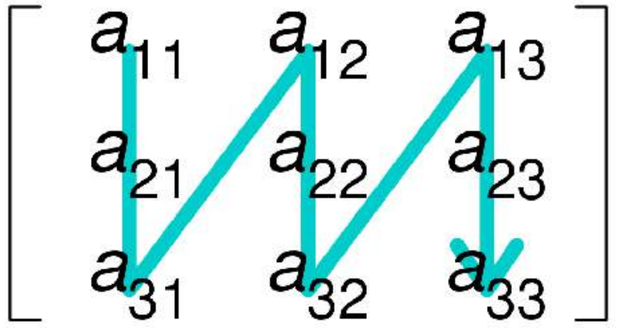
Row-major order



C: row-major order (lexicographical access order), zero-based indexing

Address	Access	Value
0	`A[0][0]`	a_{11}
1	`A[0][1]`	a_{12}
2	`A[0][2]`	a_{13}
3	`A[1][0]`	a_{21}
4	`A[1][1]`	a_{22}
5	`A[1][2]`	a_{23}

Column-major order

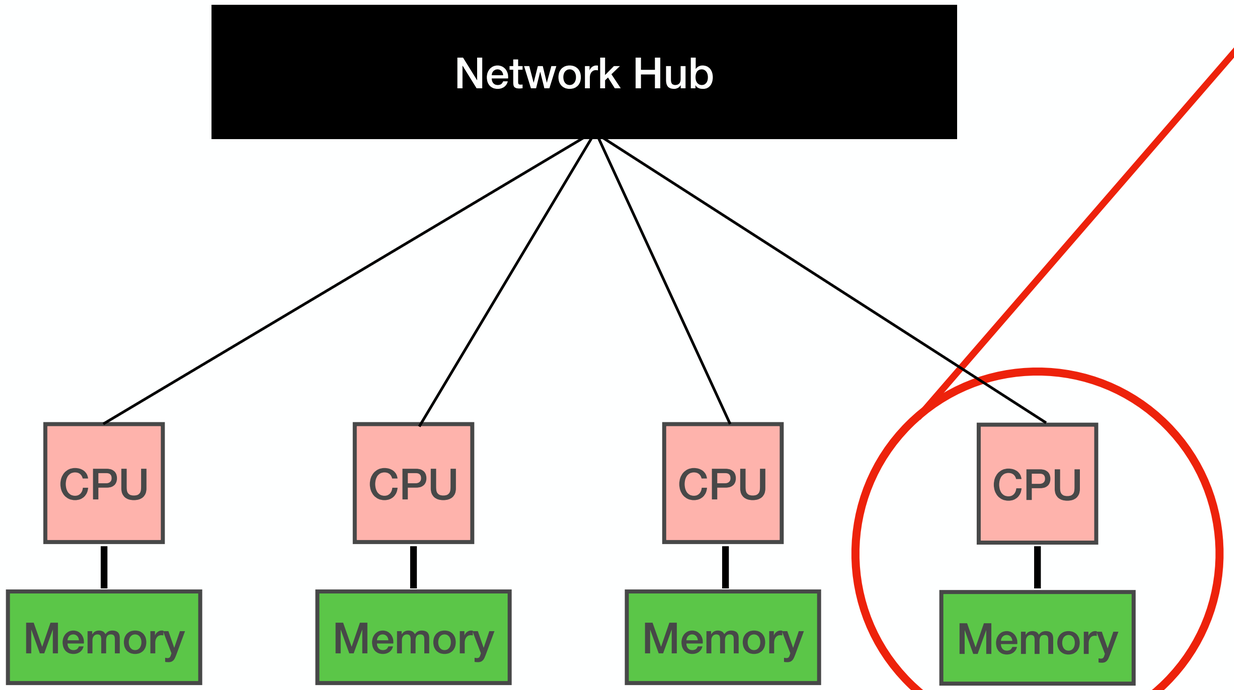


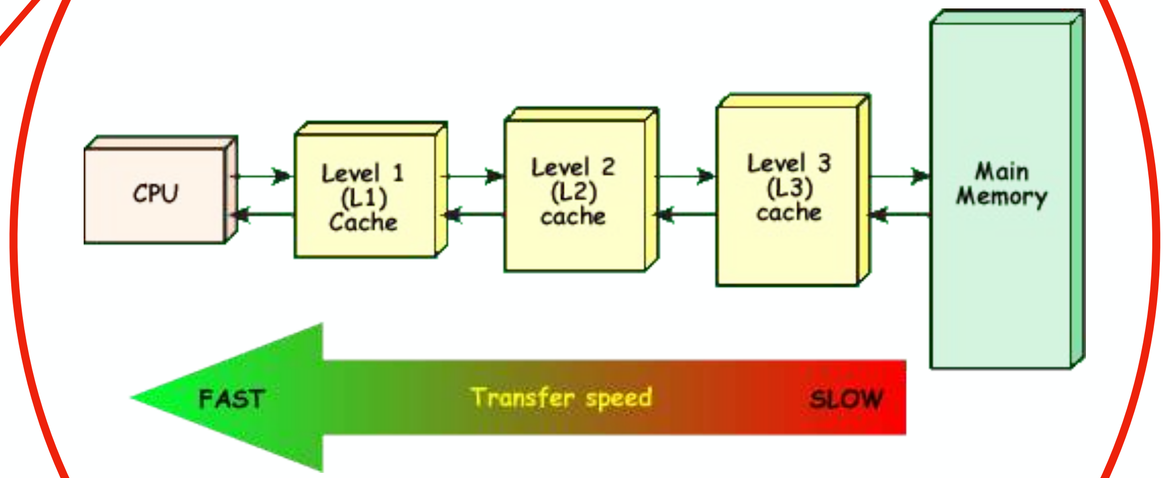
Fortran: column-major order (colexicographical access order), one-based indexing

Address	Access	Value
1	`A(1,1)`	a_{11}
2	`A(2,1)`	a_{21}
3	`A(1,2)`	a_{12}
4	`A(2,2)`	a_{22}
5	`A(1,3)`	a_{13}
6	`A(2,3)`	a_{23}

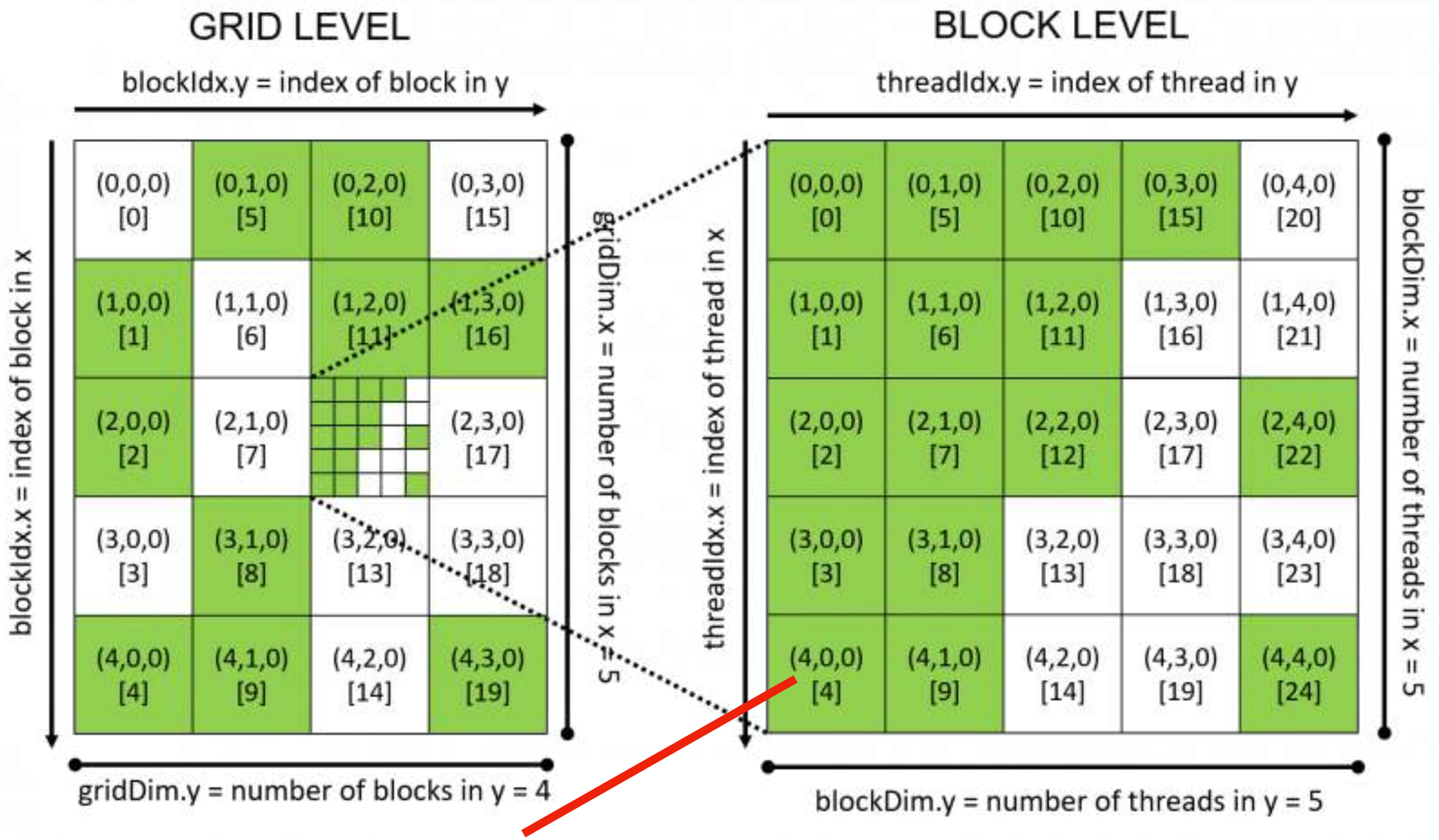
Multidimensional arrays

$$\text{Cache hit ratio} = \frac{\text{Number of cache hits}}{\text{Number of cache hits + Number of cache miss}}$$





```
 1 module simpleOps_m
 2 contains
 3   subroutine increment(a,b)
 4     implicit none
 5     integer, intent(inout) :: a(:,:)
 6     integer, intent(in) :: b
 7     integer :: i,j,nx,ny
 8     nx = size(a,1)
 9     ny = size(a,2)
10     do j = 1, ny
11       do i = 1, nx
12         a(i,j) = a(i,j) + b
13       end do
14     end do
15   end subroutine
16 end module
17
18 program ex1
19   use simpleOps_m
20   implicit none
21   integer, parameter :: nx = 1024
22   integer, parameter :: ny = 512
23   integer :: a(nx,ny), b
24
25   a = 1
26   b = 3
27   call increment(a,b)
28
29   if(any(a/=4))then
30     print*, "Program Failed"
31   else
32     print*, "Program Passed"
33   endif
34 end program
```



This thread has coordinates (4,0,0) in the block.
The block has coordinates (2,2,0) in the grid.

What are its “global” (unique) coordinates?

X = (blockIdx%x-1)*blockDim%y + threadIdx%x
Y = (blockIdx%y-1)*blockDim%y + threadIdx%y

Multidimensional arrays

fabio.guglietta@roma2.infn.it

```
module simpleOps_m
contains
  subroutine increment(a,b)
    implicit none
    integer, intent(inout) :: a(:,:)
    integer, intent(in) :: b
    integer :: i,j,nx,ny
    nx = size(a,1)
    ny = size(a,2)
    do j = 1, ny
      do i = 1, nx
        a(i,j) = a(i,j) + b
      end do
    end do
  end subroutine
end module

program ex1
  use simpleOps_m
  implicit none
  integer, parameter :: nx = 1024
  integer, parameter :: ny = 512
  integer :: a(nx,ny), b

  a = 1
  b = 3
  call increment(a,b)

  if(any(a/=4))then
    print*, "Program Failed"
  else
    print*, "Program Passed"
  endif
end program
```

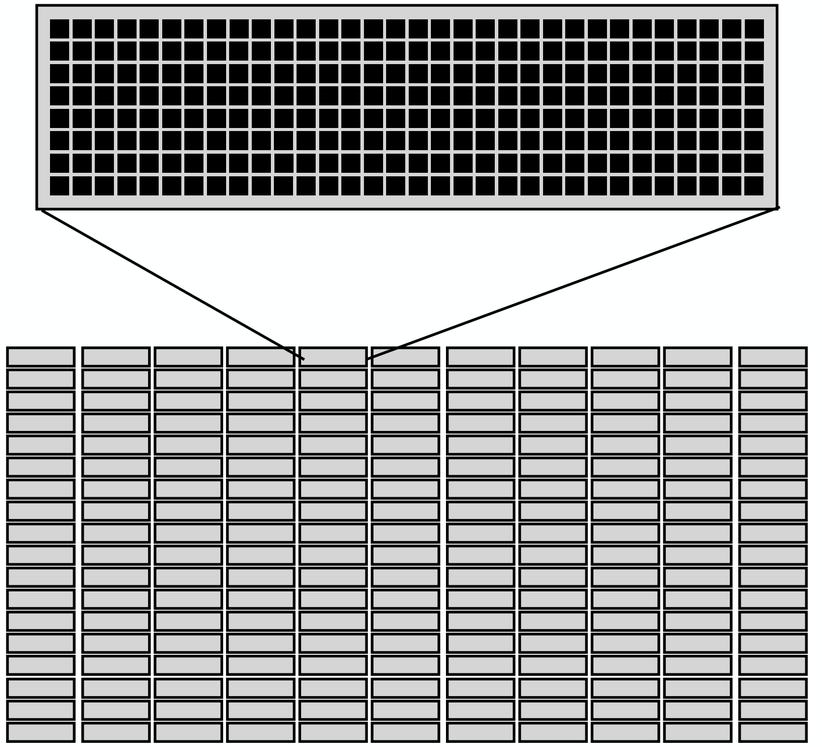
```
module simpleOps_m
contains
  attributes(global) subroutine increment(a, b)
    implicit none
    integer :: a(:,:)
    integer, value :: b
    integer :: i, j, nx,ny
    i = (blockIdx%x-1)*blockDim%x + threadIdx%x
    j = (blockIdx%y-1)*blockDim%y + threadIdx%y
    nx = size(a,1)
    ny = size(a,2)
    if (i<=nx .and. j<=ny) a(i,j) = a(i,j) + b
  end subroutine increment
end module simpleOps_m

program incrementTest
  use cudafor
  use simpleOps_m
  implicit none
  integer, parameter :: nx=1024, ny=512
  integer :: a(nx,ny), b
  integer, device :: a_d(nx,ny)
  type(dim3) :: grid, tBlock

  a = 1
  b = 3

  tBlock = dim3(32,8,1)
  grid = dim3(ceiling(real(nx)/tBlock%x), &
              ceiling(real(ny)/tBlock%y), 1)
  a_d = a
  call increment<<<grid,tBlock>>>(a_d, b)
  a = a_d

  if (any(a /= 4)) then
     print*, "Program Failed"
  else
     print*, "Program Passed"
  endif
end program incrementTest
```

Check if i and j are in the correct range

dim3 is a CUDA Fortran **provided data type** that has 3 dimensions

ceiling(A) returns the least integer greater than or equal to A.

BLOCK: 32 x 8 threads



GRID: 32 x 64 blocks

Exercise

- Initialise (on the GPU) a 3D array which stores the ids of a 3D lattice with dimension (128,128,64)
- Ids go from 1 to 128x128x64
- Check for correctness

Time for… timing

Nvprof

- nvprof ./a.out
- Try profiling the previous exercise.
- Try to increment the number of operations / number of invocations.

Theoretical and effective bandwidth

- **Theoretical bandwidth** can be calculated using **hardware specifications** available in the product literature.
  For example, the NVIDIA Tesla C2050 GPU uses DDR (double data rate) RAM with a **memory clock rate of 1500 MHz** and a **384-bit wide memory interface**. Then:

$$\text{BWTheoretical} = 1500 \times 10^{6} \times \frac{384}{8} \times 2 \times 10^{-9} = 144 \text{ GB/s}$$

- Effective bandwidth is calculated by **timing specific program activities** and using our knowledge of **how data is accessed by the program**.

$$\text{BWEffective} = \frac{R_B + W_B}{t} 10^{-9}$$

- Almost all changes to code should be made in the context of how they affect bandwidth.

- Memory bound or computational bound?
- -Mcuda=fastmath
- Measure the bandwidth (rate at which data can be transferred)
- Bandwidth can be dramatically affected by the choice of memory in which data are stored, how the data are laid out, and the order in which they are accessed, as well as other factors.

Compute capability

- The compute capability of a CUDA-enabled device indicates the **architecture** and is given in **Major.Minor format**
  - The **Major component** of the compute capability reflects the **generation** of the architecture;
  - The **Minor component** reflects the **revision** within that generation.
- We can target a compute capability of X.Y with the compiler option -Mcuda=ccXY.

	"Fermi"	"Fermi"	"Kepler"	"Kepler"	"Maxwell"	"Pascal"
Tesla GPU	**GF100**	**GF104**	**GK104**	**GK110**	**GM200**	**GP100**
Compute Capability	2.0	2.1	3.0	3.5	5.3	6.0
Streaming Multiprocessors (SMs)	16	16	8	15	24	56
FP32 CUDA Cores / SM	32	32	192	192	128	64
FP32 CUDA Cores	512	512	1536	2880	3072	3584
FP64 Units	–	–	512	960	96	1792
Threads / Warp	32	32	32	32	32	32
Max Warps / Multiprocessor	48	48	64	64	64	64
Max Threads / Multiprocessor	1536	1536	2048	2048	2048	2048
Max Thread Blocks / Multiprocessor	8	8	16	16	32	32
32-bit Registers / Multiprocessor	32768	32768	65536	65536	65536	65536
Max Registers / Thread	63	63	63	255	255	255
Max Threads / Thread Block	1024	1024	1024	1024	1024	1024
Shared Memory Size Configurations	16 KB	16 KB	16 KB	16 KB	96 KB	64 KB
	48 KB	48 KB	32 KB	32 KB		
			48 KB	48 KB		
Hyper-Q	No	No	No	Yes	Yes	Yes
Dynamic Parallelism	No	No	No	Yes	Yes	Yes
Unified Memory	No	No	No	No	No	Yes
Pre-Emption	No	No	No	No	No	Yes

LBM on GPU

$$f_i^{eq} = \omega_i \rho \left(1 + \frac{u_\alpha \, c_{i\alpha}}{c_s^2} + \frac{(u_\alpha \, c_{i\alpha})^2}{2c_s^4} - \frac{u_\alpha \, u_\alpha}{2c_s^2} \right)$$

$$t_i = u_\alpha c_{i\alpha} / c_s^2$$

$$f_i^{eq} = \omega_i \rho \left(1 + t_i + 0.5 t_i^2 - usq \right)$$

$$S_i = \left(1 - \frac{\Delta t}{2\tau} \right) \omega_i \left(\frac{c_{i\alpha}}{c_s^2} + \frac{(c_{i\alpha} c_{i\beta} - c_s^2 \delta_{\alpha\beta}) u_\beta}{c_s^4} \right) F_\alpha$$

$$s_{uF} = u_\alpha F_\alpha / c_s^2$$

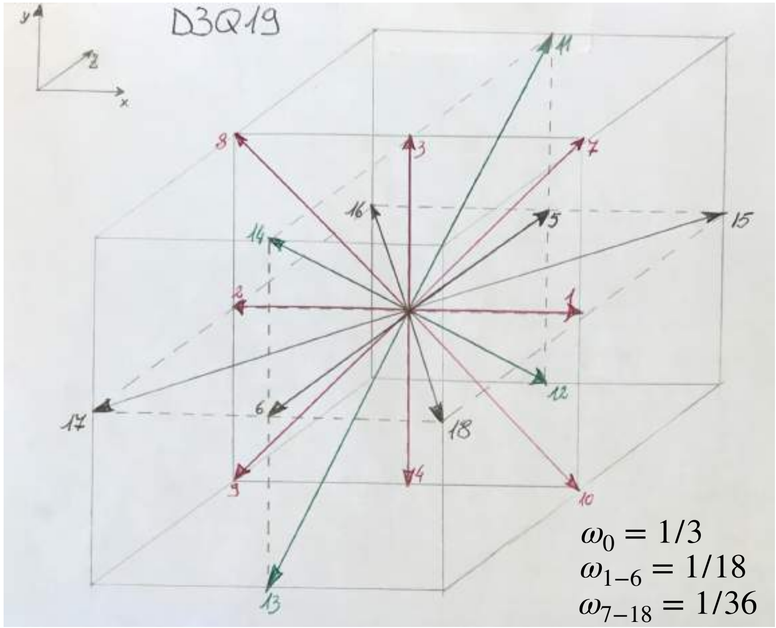
$$c_i^{(F)} = c_{i\alpha} F_\alpha$$

$$S_i = \left(1 - \frac{\Delta t}{2\tau} \right) \omega_i \left(c_i^{(F)} (1 + t_i) - s_{uF} \right)$$

$$f_i^* = f_i \left(1 - \frac{\Delta t}{\tau} \right) + f_i^{eq} \frac{\Delta t}{\tau} + S_i \Delta t$$

$$\rho = \sum_i f_i \qquad u_\alpha = \frac{1}{\rho} \sum_i f_i c_{i\alpha} + \frac{f_\alpha \Delta t}{2}$$

```
pop_out(ix,iy,iz,pop ) = ft_pop * OneMdtTau_inv +
dtTau_inv * rho * omega_pop  * (1._rk_LB + t_pop + .5_rk_LB* t_pop *t_pop  - usq) + s_pop
```



	X	Y	Z
0	0	0	0
1	+	0	0
2	-	0	0
3	0	+	0
4	0	-	0
5	0	0	+
6	0	0	-
7	+	+	0
8	-	+	0
9	-	-	0
10	+	-	0
11	0	+	+
12	0	-	+
13	0	-	-
14	0	+	-
15	+	0	+
16	-	0	+
17	-	0	-
18	+	0	-

```
grid = dim3(ceiling(real(nx_g)/tBlock%x),ceiling(real(ny_g)/tBlock%y),ceiling(real(nz_g)/tBlock%z))
```

References

- Ruetsch, Gregory, and Massimiliano Fatica. *CUDA Fortran for scientists and engineers: best practices for efficient CUDA Fortran programming.* Elsevier, 2013.