Introduction to OpenFOAM, part 1: theoretical foundations of finite volume approach

Sibgatullin I.

sibgat@ocean.ru

Moscow Lomonosov State University

27 июня 2016 г.

- To work in CFD, one needs a solid background in both fluid mechanics and numerical analysis; significant errors have been made by people lacking knowledge in one or the other.
- Estimation of numerical errors. A qualitatively incorrect solution of a problem may look reasonable (it may even be a good solution of another problem), the consequences of accepting it may be severe.

Computational Methods for Fluid Dynamics.

Professor Joel H. Ferziger, Dr. Milovan Perić

"OpenFOAM is first and foremost a C++ library, used primarily to create executables, known as applications."

OpenFOAM User Guide

Conservation principles

Conservation laws can be derived by considering a given quantity of matter or *control mass* (CM) and its *extensive* properties, such as mass, momentum and energy.

$$\frac{\mathrm{d}}{\mathrm{d}t}\int\limits_{\Omega_{\mathrm{CM}}} \varphi\rho\,\mathrm{d}\Omega = zero(\mathbf{0}) \quad + \text{sources} + \text{flows through boundaries}$$

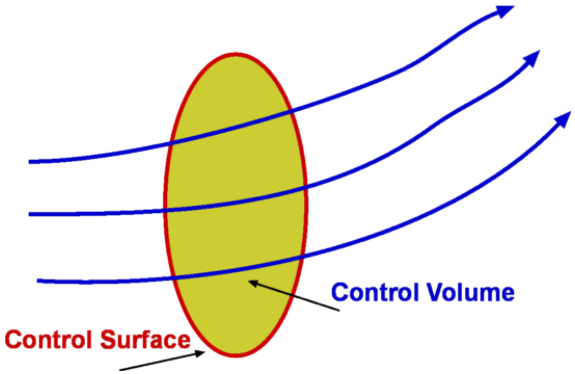
The conservation equation for mass:

$$\frac{\mathrm{d}m}{\mathrm{d}t} = 0 \,. \tag{1}$$

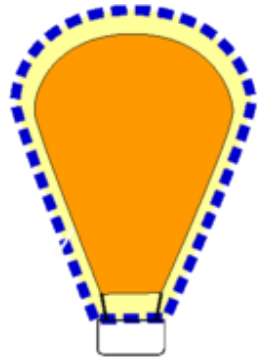
On the other hand, momentum can be changed by the action of forces and its conservation equation is Newton's second law of motion:

$$\frac{\mathrm{d}(m\vec{v})}{\mathrm{d}t} = \sum \vec{F} \,, \tag{2}$$

where t stands for time, m for mass, $\vec{v}$ for the velocity, and $\vec{f}$ for forces acting on the control mass.



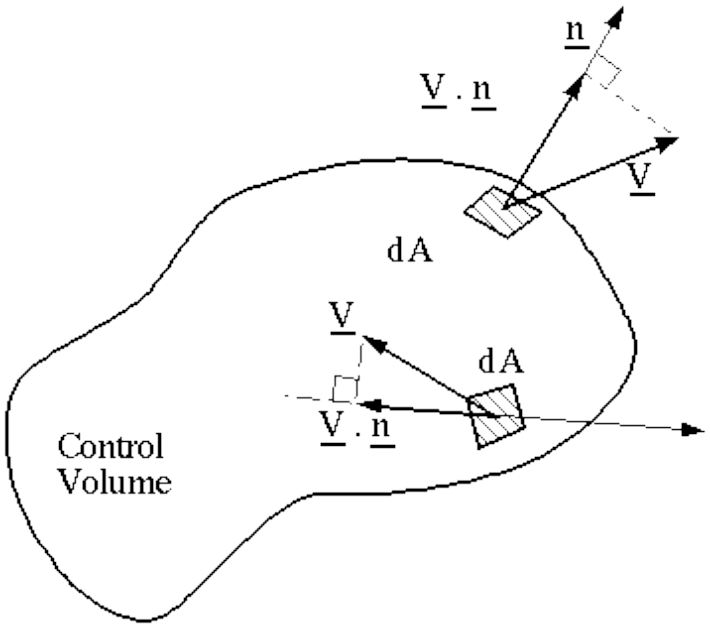
A moving Control Volume around a moving Aeroplane.

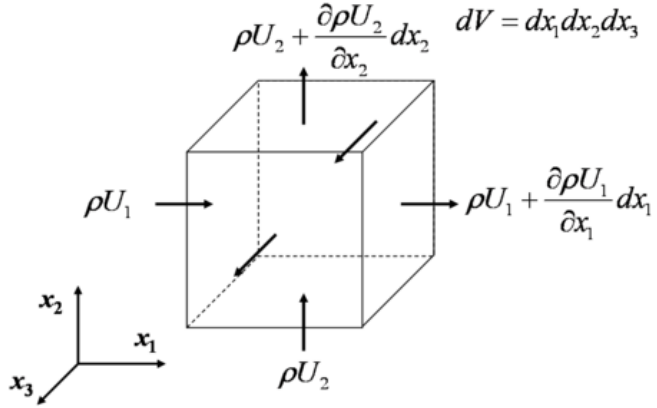


A collapsible Control Volume around a balloon.

- *CM approach* is used to study the dynamics of *solid bodies*, where the CM (sometimes called the *system*) is easily identified.
- In fluid flows, however, it is difficult *to follow a parcel* of matter.
- It is more convenient to deal with the flow within a certain spatial region we call a *control volume* (CV), rather than in a parcel of matter which quickly passes through the region of interest. This method of analysis is called the *control volume approach*.

Closer look to control volume





Reynolds transport theorem

φ – conserved intensive property (for mass conservation, $\varphi = 1$; for momentum conservation, $\varphi = \vec{v}$; for conservation of a scalar, φ represents the conserved property per unit mass). The corresponding extensive property Φ can be expressed as:

$$\Phi = \int\limits_{\Omega_{\mathrm{CM}}} \rho\varphi \,\mathrm{d}\Omega \,, \tag{3}$$

Ω_{CM} stands for volume occupied by the CM.
LHS of each conservation equation for a control volume can be written:

$$\frac{\mathrm{d}}{\mathrm{d}t} \int\limits_{\Omega_{\mathrm{CM}}} \rho\varphi \,\mathrm{d}\Omega = \frac{\mathrm{d}}{\mathrm{d}t} \int\limits_{\Omega_{\mathrm{CV}}} \rho\varphi \,\mathrm{d}\Omega + \int\limits_{S_{\mathrm{CV}}} \rho\varphi \,(\vec{v} - \vec{v}_{\mathrm{b}}) \cdot \vec{n} \,\mathrm{d}S \,, \tag{4}$$

Ω_{CV} is the CV volume, S_{CV} is the surface enclosing CV,
$\vec{n}$ is the unit vector orthogonal to S_{CV} and directed outwards,
$\vec{v}_{\mathrm{b}}$ is the velocity with which the CV surface is moving.
For fixed CV $\vec{v}_{\mathrm{b}} = \vec{0}$, first derivative on the RHS becomes a local (partial).
The last term is usually called the *convective* (or sometimes, advective) flux of φ through the CV boundary.

Reynolds transport theorem if CV is fixed in space

If CV does not change in time $v_b = 0$, $\frac{\partial}{\partial t}\big|_{CV} = \frac{\partial}{\partial t}$:

$$\frac{\mathrm{d}}{\mathrm{d}t}\int\limits_{\Omega_{\mathrm{CM}}} \rho\varphi \,\mathrm{d}\Omega = \int\limits_{\Omega_{\mathrm{CV}}} \frac{\partial}{\partial t}(\rho\varphi)\,\mathrm{d}\Omega + \int\limits_{S_{\mathrm{CV}}} \rho\varphi\,(\vec{v},\vec{n})\,\mathrm{d}S \;= \tag{5}$$

(Homework: Consider "Differentiation under the integral sign" https://en.wikipedia.org/wiki/Differentiation_under_the_integral_sign and describe its connection to differentiation over CM and to Reynolds Transport Theorem.)

$$= \int\limits_{\Omega_{\mathrm{CV}}} \left[\frac{\partial}{\partial t}(\rho\varphi) \;+ \mathrm{div}\,(\rho\varphi\vec{v})\right]\mathrm{d}\Omega \tag{6}$$

Mass Conservation

$$\varphi \to 1 \tag{7}$$

$$\frac{\partial}{\partial t} \int_{\Omega} \rho \, \mathrm{d}\Omega + \int_{S} \rho \vec{v} \cdot \vec{n} \, \mathrm{d}S = 0 \, . \tag{8}$$

By applying the Gauss–Ostrogradsky divergence theorem to the convection term, we can transform the surface integral into a volume integral. Allowing the control volume to become infinitesimally small leads to a differential coordinate-free form of the continuity equation:

$$\frac{\partial \rho}{\partial t} + \mathrm{div} \left(\rho \vec{v} \right) = 0 \, . \tag{9}$$

Scalar Conservation

$$\frac{\mathrm{d}}{\mathrm{d}t}\int\limits_{\Omega_{\mathrm{CM}}}\varphi\rho\,\mathrm{d}v=\int\limits_{\Omega_{\mathrm{CV}}}s_\varphi\rho\,\mathrm{d}v-\int\limits_{\partial\Omega_{\mathrm{CV}}}\vec{q}_\varphi\cdot\overrightarrow{\mathrm{d}s} \tag{10}$$

s_φ – source of φ, $\vec{q}$ – flux of φ through boundaries.

$$\frac{\partial}{\partial t}(\rho\varphi)+\nabla_i(\rho\varphi v^i)=s_\varphi-\nabla_k q^k_\varphi \tag{11}$$

Diffusive transport is always present (even in stagnant fluids), and it is usually described by a gradient approximation, e.g. *Fourier's law* for heat diffusion and *Fick's law* for mass diffusion:

$$\vec{q}_\varphi=-\lambda\nabla\varphi\,,\quad Q_\varphi=-\int_S\lambda\nabla\varphi\cdot\vec{n}\,\mathrm{d}S\,, \tag{12}$$

where λ is the diffusivity for the quantity φ.

Momentum Conservation

$$\varphi \to \vec{v} \tag{13}$$

$$\frac{\partial}{\partial t}\int_{\Omega}\rho\vec{v}\,\mathrm{d}\Omega + \int_{S}\rho\vec{v}(\vec{v},\vec{n})\,\mathrm{d}S = \sum\vec{F}\,. \tag{14}$$

To express the right hand side in terms of intensive properties, one has to consider the forces which may act on the fluid in a CV:

- body forces (gravity, centrifugal and Coriolis forces, electromagnetic forces, etc.).
- surface forces (pressure, normal and shear stresses, surface tension etc.);

Momentum Conservation

$$\frac{\partial}{\partial t}\int_{\Omega}\rho\vec{v}\,\mathrm{d}\Omega+\int_{S}\rho\vec{v}(\vec{v},\vec{n})\,\mathrm{d}S=\int_{\Omega}\vec{f}\rho\,\mathrm{d}\Omega+\int_{S}\vec{\sigma}_{n}\,\mathrm{d}S\,. \tag{15}$$

$\vec{f}$ – body mass forces per unit of mass, $\sigma_{\vec{n}}$ – surface forces per unit of area.
$\sigma_{\vec{n}}=\vec{\sigma}^{i}n_{i}$, $\vec{\sigma}^{i}$ – surface forces per unit of area on i-th coordinate plane.

Momentum Conservation

$$\frac{\partial}{\partial t}\int_{\Omega}\rho\vec{v}\,\mathrm{d}\Omega+\int_{S}\rho\vec{v}(\vec{v},\vec{n})\,\mathrm{d}S=\int_{\Omega}\vec{f}\rho\,\mathrm{d}\Omega+\int_{S}\vec{\sigma}_{n}\,\mathrm{d}S\,. \tag{15}$$

$\vec{f}$ – body mass forces per unit of mass, $\sigma_{\vec{n}}$ – surface forces per unit of area. $\sigma_{\vec{n}}=\vec{\sigma}^{i}n_{i}$, $\vec{\sigma}^{i}$ – surface forces per unit of area on i-th coordinate plane.

$$\frac{\partial}{\partial t}\int_{\Omega}\rho\vec{v}\,\mathrm{d}\Omega+\int_{S}\rho\vec{v}v^{i}n_{i}\,\mathrm{d}S=\int_{\Omega}\vec{f}\rho\,\mathrm{d}\Omega+\int_{S}\vec{\sigma}^{i}n_{i}\,\mathrm{d}S\,. \tag{16}$$

Momentum Conservation

$$\frac{\partial}{\partial t}\int_\Omega \rho\vec{v}\,\mathrm{d}\Omega + \int_S \rho\vec{v}(\vec{v},\vec{n})\,\mathrm{d}S = \int_\Omega \vec{f}\rho\,\mathrm{d}\Omega + \int_S \vec{\sigma}_n\,\mathrm{d}S\,. \tag{15}$$

$\vec{f}$ – body mass forces per unit of mass, $\sigma_{\vec{n}}$ – surface forces per unit of area. $\sigma_{\vec{n}} = \vec{\sigma}^i n_i$, $\vec{\sigma}^i$ – surface forces per unit of area on i-th coordinate plane.

$$\frac{\partial}{\partial t}\int_\Omega \rho\vec{v}\,\mathrm{d}\Omega + \int_S \rho\vec{v}v^i n_i\,\mathrm{d}S = \int_\Omega \vec{f}\rho\,\mathrm{d}\Omega + \int_S \vec{\sigma}^i n_i\,\mathrm{d}S\,. \tag{16}$$

$$\int_\Omega \left(\frac{\partial}{\partial t}(\rho\vec{v}) + \nabla_i(\rho\vec{v}v^i)\right)\,\mathrm{d}\Omega = \int_\Omega \left(\vec{f}\rho + \nabla_i\vec{\sigma}^i\right)\,\mathrm{d}\Omega\,. \tag{17}$$

Momentum Conservation

$$\frac{\partial}{\partial t}\int_\Omega \rho\vec{v}\,\mathrm{d}\Omega + \int_S \rho\vec{v}(\vec{v},\vec{n})\,\mathrm{d}S = \int_\Omega \vec{f}\rho\,\mathrm{d}\Omega + \int_S \vec{\sigma}_n\,\mathrm{d}S\,. \tag{15}$$

$\vec{f}$ – body mass forces per unit of mass, $\sigma_{\vec{n}}$ – surface forces per unit of area. $\sigma_{\vec{n}} = \vec{\sigma}^i n_i$, $\vec{\sigma}^i$ – surface forces per unit of area on i-th coordinate plane.

$$\frac{\partial}{\partial t}\int_\Omega \rho\vec{v}\,\mathrm{d}\Omega + \int_S \rho\vec{v}v^i n_i\,\mathrm{d}S = \int_\Omega \vec{f}\rho\,\mathrm{d}\Omega + \int_S \vec{\sigma}^i n_i\,\mathrm{d}S\,. \tag{16}$$

$$\int_\Omega \left(\frac{\partial}{\partial t}(\rho\vec{v}) + \nabla_i(\rho\vec{v}v^i)\right)\,\mathrm{d}\Omega = \int_\Omega \left(\vec{f}\rho + \nabla_i\vec{\sigma}^i\right)\,\mathrm{d}\Omega\,. \tag{17}$$

$$\frac{\partial}{\partial t}(\rho\vec{v}) + \nabla_i(\rho\vec{v}v^i) = \vec{f}\rho + \nabla_i\vec{\sigma}^i. \tag{18}$$

Difference between solids and fluids

Fluid is a substance that continually deforms (flows) under an applied shear stress.
Fluid is a substance whose molecular structure cannot resist any shear force applied to it.

Difference between solids and fluids

Fluid is a substance that continually deforms (flows) under an applied shear stress.
Fluid is a substance whose molecular structure cannot resist any shear force applied to it.
In ideal fluid:

$$\vec{\sigma}_{\vec{n}} = -p\vec{n} \tag{20}$$

Difference between solids and fluids

Fluid is a substance that continually deforms (flows) under an applied shear stress.
Fluid is a substance whose molecular structure cannot resist any shear force applied to it.
In ideal fluid:

$$\vec{\sigma}_{\vec{n}} = -p\vec{n} \tag{20}$$

In viscous incompressible fluid: $\vec{\sigma}_{\vec{n}} = \vec{\sigma}^{i} n_i = \sigma^{ik} \vec{e}_k n_i$

$$\sigma^{ik} = -pg^{ik} + 2\mu e^{ik} \tag{21}$$

Difference between solids and fluids

Fluid is a substance that continually deforms (flows) under an applied shear stress.
Fluid is a substance whose molecular structure cannot resist any shear force applied to it.
In ideal fluid:

$$\vec{\sigma}_{\vec{n}} = -p\vec{n} \tag{20}$$

In viscous incompressible fluid: $\vec{\sigma}_{\vec{n}} = \vec{\sigma}^{i} n_i = \sigma^{ik} \vec{e}_k n_i$

$$\sigma^{ik} = -pg^{ik} + 2\mu e^{ik} \tag{21}$$

In viscous compressible fluid (second viscosity assumed to be 0):

$$\sigma^{ik} = -\left(p + \frac{2}{3}\mu \operatorname{div} \vec{v}\right) g^{ik} + 2\mu e^{ik} \tag{22}$$

Navier – Stokes equations

Convective form:

$$\rho\left(\frac{\partial \vec{\mathrm{v}}}{\partial t} + v^k \nabla_k \vec{v}\right) = -\nabla p + \nabla^k \mu \nabla_k \vec{v} + \frac{\mu}{3}\nabla(\nabla \cdot v) + \rho\vec{g}$$

Conservative form:

$$\frac{\partial}{\partial t}(\rho v^k) + \nabla_i(\rho v^k v^i) = -\nabla p + \nabla^k \mu \nabla_k \vec{v} + \frac{\mu}{3}\nabla(\nabla \cdot v) + \rho\vec{g}$$

In case of incompressible fluid with constant viscosity:

$$\frac{\partial}{\partial t}(\rho v^k) + \nabla_i(\rho v^k v^i) = -\nabla p + \mu\Delta\vec{v} + \rho\vec{g}$$

Difference between solids and fluids

An ideal elastic solid will deform under load and, once the load is removed, will return to its original state. Some solids are plastic. These deform under the action of a sufficient load and deformation continues as long as a load is applied, providing the material does not rupture. Deformation ceases when the load is removed, but the plastic solid does not return to its original state.

Difference between solids and fluids

An ideal elastic solid will deform under load and, once the load is removed, will return to its original state. Some solids are plastic. These deform under the action of a sufficient load and deformation continues as long as a load is applied, providing the material does not rupture. Deformation ceases when the load is removed, but the plastic solid does not return to its original state.
In ideal elastic solid stess vector and so stress tensor depends on deformations: $\vec{\sigma}_{\vec{n}} = \vec{\sigma}^i n_i = \sigma^{ik} \vec{e}_k n_i$

$$\sigma^{ik} = \lambda I_1(\varepsilon) g^{ik} + 2\mu\varepsilon^{ik} \tag{23}$$

Difference between solids and fluids

An ideal elastic solid will deform under load and, once the load is removed, will return to its original state. Some solids are plastic. These deform under the action of a sufficient load and deformation continues as long as a load is applied, providing the material does not rupture. Deformation ceases when the load is removed, but the plastic solid does not return to its original state.
In ideal elastic solid stess vector and so stress tensor depends on deformations: $\vec{\sigma}_{\vec{n}} = \vec{\sigma}^i n_i = \sigma^{ik} \vec{e}_k n_i$

$$\sigma^{ik} = \lambda I_1(\varepsilon) g^{ik} + 2\mu\varepsilon^{ik} \tag{23}$$

$$E = \mu \frac{3\lambda + 2\mu}{\lambda + \mu}, \quad \sigma = \frac{\lambda}{2(\lambda + \mu)} \tag{24}$$

Differentianl equations of continuum media: *divergence* form

Mass conservation:

$$\frac{\partial \rho}{\partial t} + \nabla_i(\rho v^i) = 0 \tag{25}$$

Momentum conservation:

$$\frac{\partial}{\partial t}(\rho v^k) + \nabla_i(\rho v^k v^i) = \nabla_i \sigma^{ik} + \rho f^k \tag{26}$$

Scalar conservation:

$$\frac{\partial}{\partial t}(\rho\varphi) + \nabla_i(\rho\varphi v^i) = s_\varphi \rho - \operatorname{div} \vec{q}_\varphi$$

Such a form is somtimes also called *conservation* form since as we will see in FV mehod integral quantities are conserved after space discretisation (decomposition to finite volumes).
So this form of equations is essential for OpenFOAM

Divergence form of differentianl equations in tensor notation

$$\frac{\partial \rho}{\partial t} + \operatorname{div}(\rho \vec{v}) = 0$$

$$\frac{\partial}{\partial t}(\rho \vec{v}) + \operatorname{div}(\rho \vec{v} \vec{v}) = \operatorname{div} \sigma + \rho \vec{f}$$

$$\frac{\partial}{\partial t}(\rho \varphi) + \operatorname{div}(\rho \varphi \vec{v}) = s_\varphi \rho - \operatorname{div} \vec{q}_\varphi$$

$\vec{v}\vec{v}$ means $\vec{v} \otimes \vec{v} = [v_i v_k] = \begin{bmatrix} v_1 v_1 & v_1 v_2 & v_1 v_3 \\ v_2 v_1 & v_2 v_2 & v_2 v_3 \\ v_3 v_1 & v_3 v_2 & v_3 v_3 \end{bmatrix}$

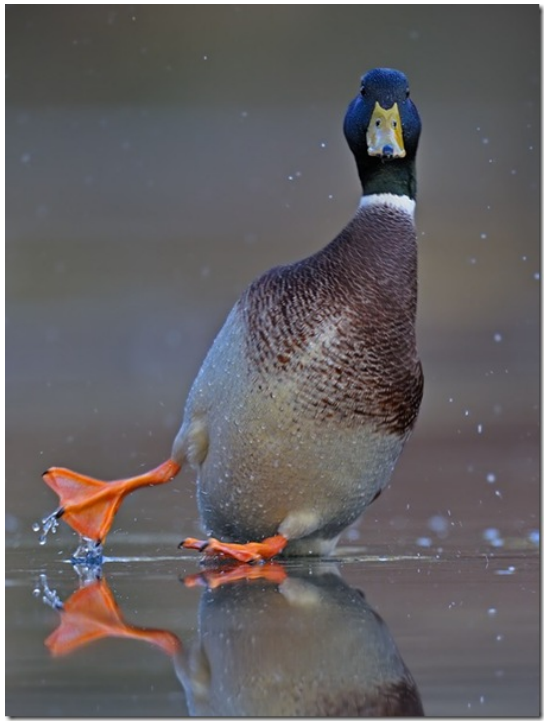
$\operatorname{div}(\rho \vec{v} \vec{v})$ means $\operatorname{div}(\rho v^k \vec{v}) e_k$; $\operatorname{div}(\sigma)$ means $\nabla_i p^{ik} e_k$

Do you think that engeneers who uses OpenFOAM and commercial packages for years always understand it?

Boundary condirions

Slip, No-slip – boring

"Stressfree"



"Stressful"

Dimensionless form of equations

$$t^* = \frac{t}{t_0}\ ; \quad x_i^* = \frac{x_i}{L_0}\ ; \quad u_i^* = \frac{u_i}{v_0}\ ; \quad p^* = \frac{p}{\rho v_0^2}\ ; \quad T^* = \frac{T - T_0}{T_1 - T_0}.$$

$$\frac{\partial u_i^*}{\partial x_i^*} = 0\ , \tag{27}$$

$$\mathrm{St}\frac{\partial u_i^*}{\partial t^*} + \frac{\partial (u_j^* u_i^*)}{\partial x_j^*} = \frac{1}{\mathrm{Re}}\frac{\partial^2 u_i^*}{\partial x_j^{*2}} - \frac{\partial p^*}{\partial x_i^*} + \frac{1}{\mathrm{Fr}^2}\gamma_i\ , \tag{28}$$

$$\mathrm{St}\frac{\partial T^*}{\partial t^*} + \frac{\partial (u_j^* T^*)}{\partial x_j^*} = \frac{1}{\mathrm{Re\ Pr}}\frac{\partial^2 T^*}{\partial x_j^{*2}}\ . \tag{29}$$

In theoretical books on Mechanics of Continua Media you can often see that when they describe the problem, they add asteriscs * to all the varables (dimensional). And later they use usual notation for dimensionless variables. In CFD books this kind of notiation is less popular, since for complex problems and geometry dimensionless form is often useless.

Simplified models

Incompressible flows

$$\operatorname{div} \vec{v} = 0 \ , \tag{30}$$

$$\frac{\partial u_i}{\partial t} + \operatorname{div} (u_i \vec{v}) = \operatorname{div} (\nu \operatorname{grad} u_i) - \frac{1}{\rho} \operatorname{div} (p \vec{i}_i) + b_i \ , \tag{31}$$

Inviscid (Euler) flow:

$$\frac{\partial (\rho u_i)}{\partial t} + \operatorname{div} (\rho u_i \vec{v}) = -\operatorname{div} (p \vec{i}_i) + \rho b_i \ . \tag{32}$$

Simplified models

Potential flows

$$\operatorname{div} \vec{v} = 0 \,, \tag{33}$$

$$\frac{\partial u_i}{\partial t} + \operatorname{div}\left(u_i \vec{v}\right) = \operatorname{div}\left(\nu \operatorname{grad} u_i\right) - \frac{1}{\rho} \operatorname{div}\left(p \vec{i}_i\right) + b_i \,, \tag{34}$$

$$\operatorname{rot} \vec{v} = 0 \,. \tag{35}$$

$$\vec{v} = -\operatorname{grad} \Phi$$

For incompressible flows:

$$\operatorname{div}\left(\operatorname{grad} \Phi\right) = 0 \,.$$

Simplified models

Creeping (Stokes) Flow

When the flow velocity is very small, the fluid is very viscous, or the geometric dimensions are very small (i.e. when the Reynolds number is small), the convective (inertial) terms in the Navier-Stokes equations play a minor role and can be neglected.

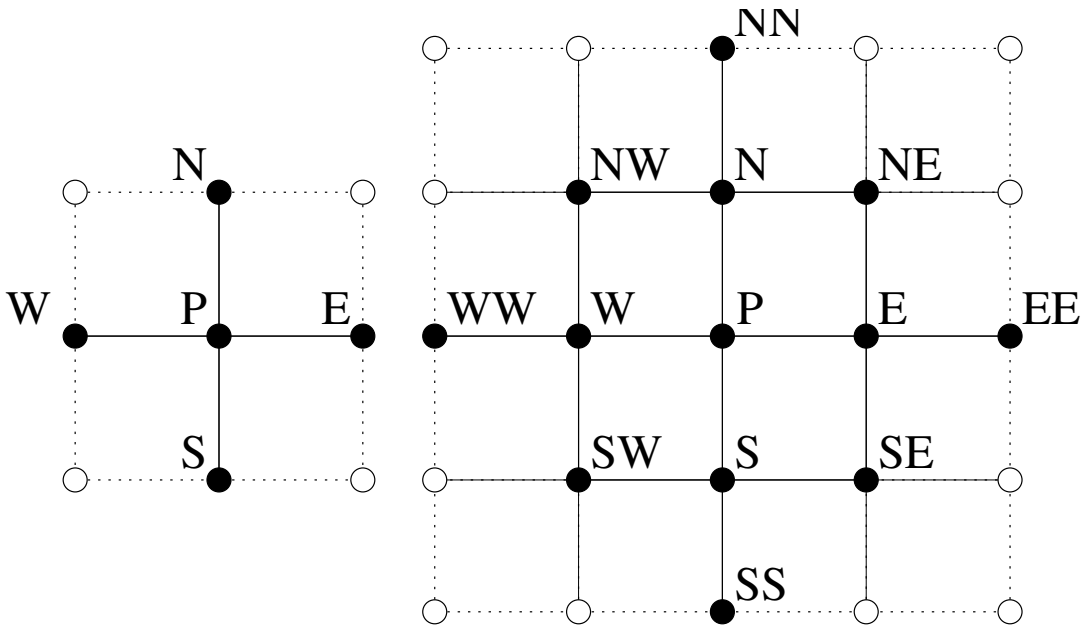
$$\operatorname{div}(\mu \operatorname{grad} u_i) - \frac{1}{\rho} \operatorname{div}(p \vec{i}_i) + b_i = 0 \,. \tag{36}$$

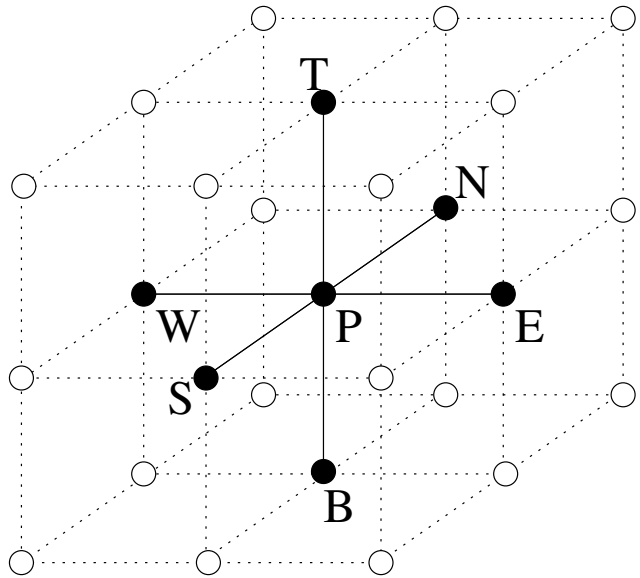
Creeping flows are found in porous media, coating technology, micro-devices etc.

The Algebraic Equation System

$$A_{\mathrm{P}}\phi_{\mathrm{P}} + \sum_{l} A_l \phi_l = Q_{\mathrm{P}} \ , \tag{37}$$

where P denotes the node at which the partial differential equation is approximated and index l runs over the neighbor nodes involved in finite-difference approximations. The node P and its neighbors form the so-called *computational molecule*





The Algebraic Equation System

This system is *sparse*, meaning that each equation contains only a few unknowns. The system can be written in matrix notation as follows:

$$A\vec{\phi} = \vec{Q}\ , \tag{38}$$

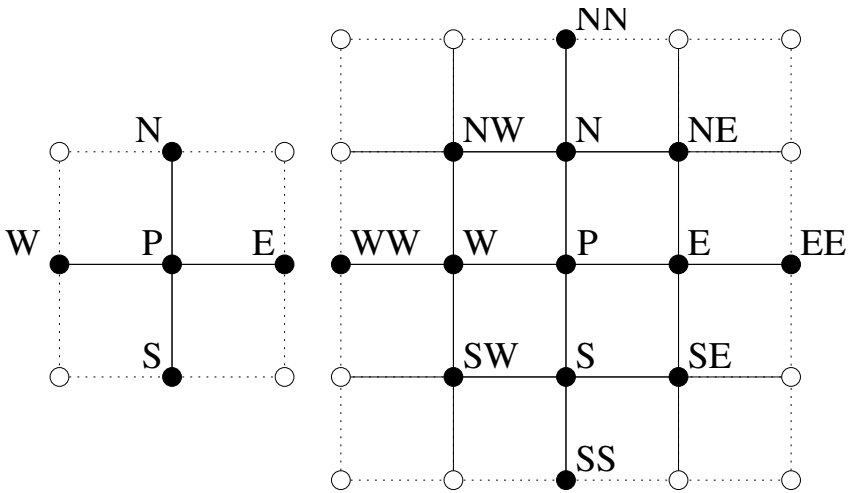
where A is the square sparse coefficient matrix, $\vec{\phi}$ is a vector (or column matrix) containing the variable values at the grid nodes, and $\vec{Q}$ is the vector containing the terms on the right-hand side of Eq. (37).

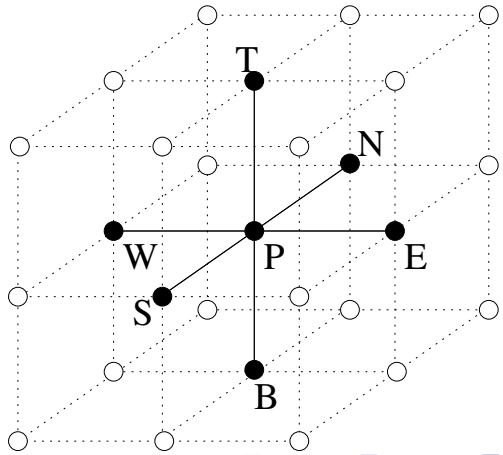
The structure of matrix A depends on the ordering of variables in the vector $\vec{\phi}$. For structured grids, if the variables are labeled starting at a corner and traversing line after line in a regular manner (*lexicographic ordering*), the matrix has a poly-diagonal structure. For the case of a five-point computational molecule, all the non-zero coefficients lie on the main diagonal, the two neighboring diagonals, and two other diagonals removed by N positions from the main diagonal, where N is the number of nodes in one direction. All other coefficients are zero. This structure allows use of efficient iterative solvers.

The Algebraic Equation System

Conversion of grid indices to one-dimensional storage locations for vectors or column matrices

Grid location	Compass notation	Storage location
i, j, k	P	$l = (k-1)N_j N_i + (i-1)N_j + j$
$i-1, j, k$	W	$l - N_j$
$i, j-1, k$	S	$l - 1$
$i, j+1, k$	N	$l + 1$
$i+1, j, k$	E	$l + N_j$
$i, j, k-1$	B	$l - N_i N_j$
$i, j, k+1$	T	$l + N_i N_j$





The Algebraic Equation System

Because the matrix A is sparse, it does not make sense to store it as a two-dimensional array in computer memory (this is standard practice for full matrices). Storing the elements of each non-zero diagonal in a separate array of dimension $1 \times N_i N_j$, where N_i and N_j are the numbers of grid points in the two coordinate directions, requires only $5N_i N_j$ words of storage; full array storage would require $N_i^2 N_j^2$ words of storage. In three dimensions, the numbers are $7N_i N_j N_k$ and $N_i^2 N_j^2 N_k^2$, respectively. The difference is sufficiently large that the diagonal-storage scheme may allow the problem to be kept in main memory when the full-array scheme does not.

The Algebraic Equation System

The linearized algebraic equations in two dimensions can now be written in the form:

$$A_{l,l-N_j}\phi_{l-N_j} + A_{l,l-1}\phi_{l-1} + A_{l,l}\phi_l + A_{l,l+1}\phi_{l+1} + A_{l,l+N_j}\phi_{l+N_j} = Q_l$$

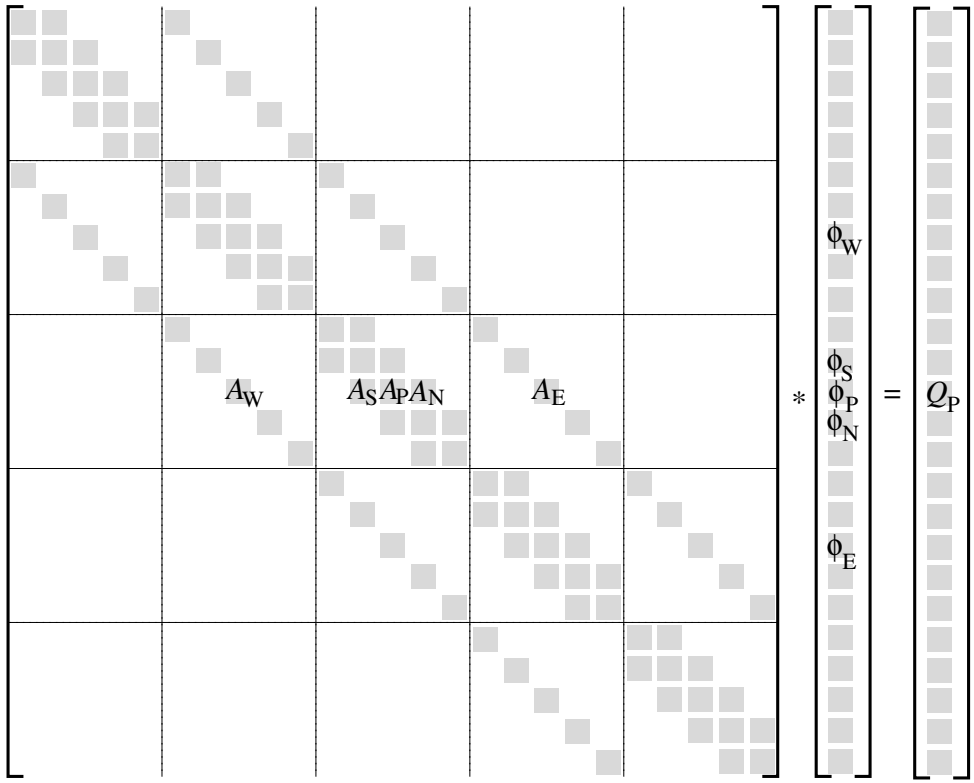


Рис.: Structure of the matrix for a five-point computational molecule (non-zero entries in the coefficient matrix on five diagonals are shaded; each horizontal set of boxes corresponds to one grid line)

The Algebraic Equation System

Matrix storage

As noted above, it makes little sense to store the matrix as an array. If, instead, the diagonals are kept in separate arrays, it is better to give each diagonal a separate name. Since each diagonal represents the connection to the variable at a node that lies in a particular direction with respect to the central node, we shall call them A_{W}, A_{S}, A_{P}, A_{N} and A_{E}; their locations in the matrix for a grid with 5×5 internal nodes are shown in Fig. 3. With this ordering of points, each node is identified with an index l, which is also the relative storage location. In this notation the equation can be written

$$A_{\mathrm{W}}\phi_{\mathrm{W}} + A_{\mathrm{S}}\phi_{\mathrm{S}} + A_{\mathrm{P}}\phi_{\mathrm{P}} + A_{\mathrm{N}}\phi_{\mathrm{N}} + A_{\mathrm{E}}\phi_{\mathrm{E}} = Q_{\mathrm{P}} \ , \tag{39}$$

where the index l, which indicated rows, is understood, and the index indicating column or location in the vector has been replaced by the corresponding letter. We shall use this shorthand notation from now on. When necessary for clarity, the index will be inserted. A similar treatment applies to three-dimensional problems.

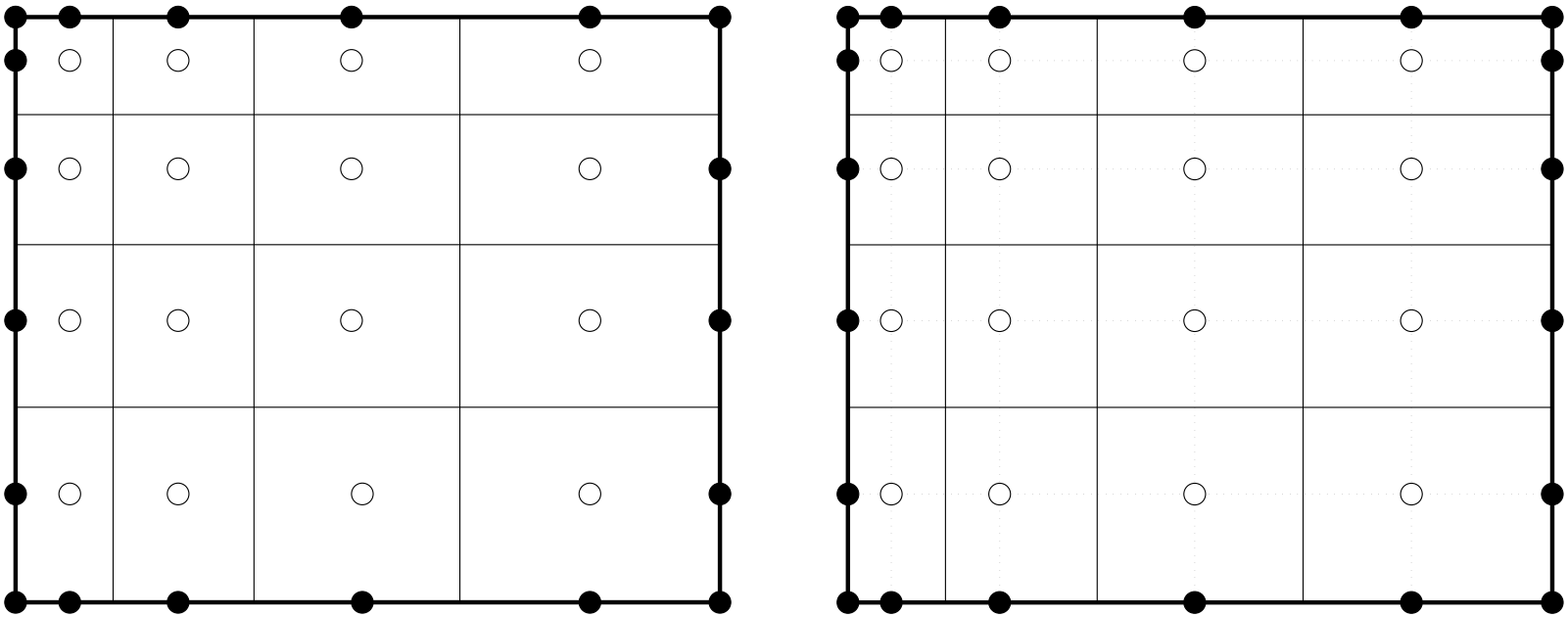
<+++>

Lecture 2. Finite Volume Approximations

Finite Volume Methods

$$\int_S \rho\phi\vec{v}\cdot\vec{n}\,\mathrm{d}S = \int_S \Gamma\,\mathrm{grad}\,\phi\cdot\vec{n}\,\mathrm{d}S + \int_\Omega q_\phi\,\mathrm{d}\Omega\,. \tag{40}$$

The solution domain is subdivided into a finite number of small control volumes (CVs) by a grid which, in contrast to the finite difference (FD) method, defines the control volume boundaries, not the computational nodes.



Surface integrals over inner CV faces cancel out.

The net flux through the CV boundary is the sum of integrals over the four (in 2D) or six (in 3D) CV faces:

$$\int_S f \, \mathrm{d}S = \sum_k \int_{S_k} f \, \mathrm{d}S \, , \tag{41}$$

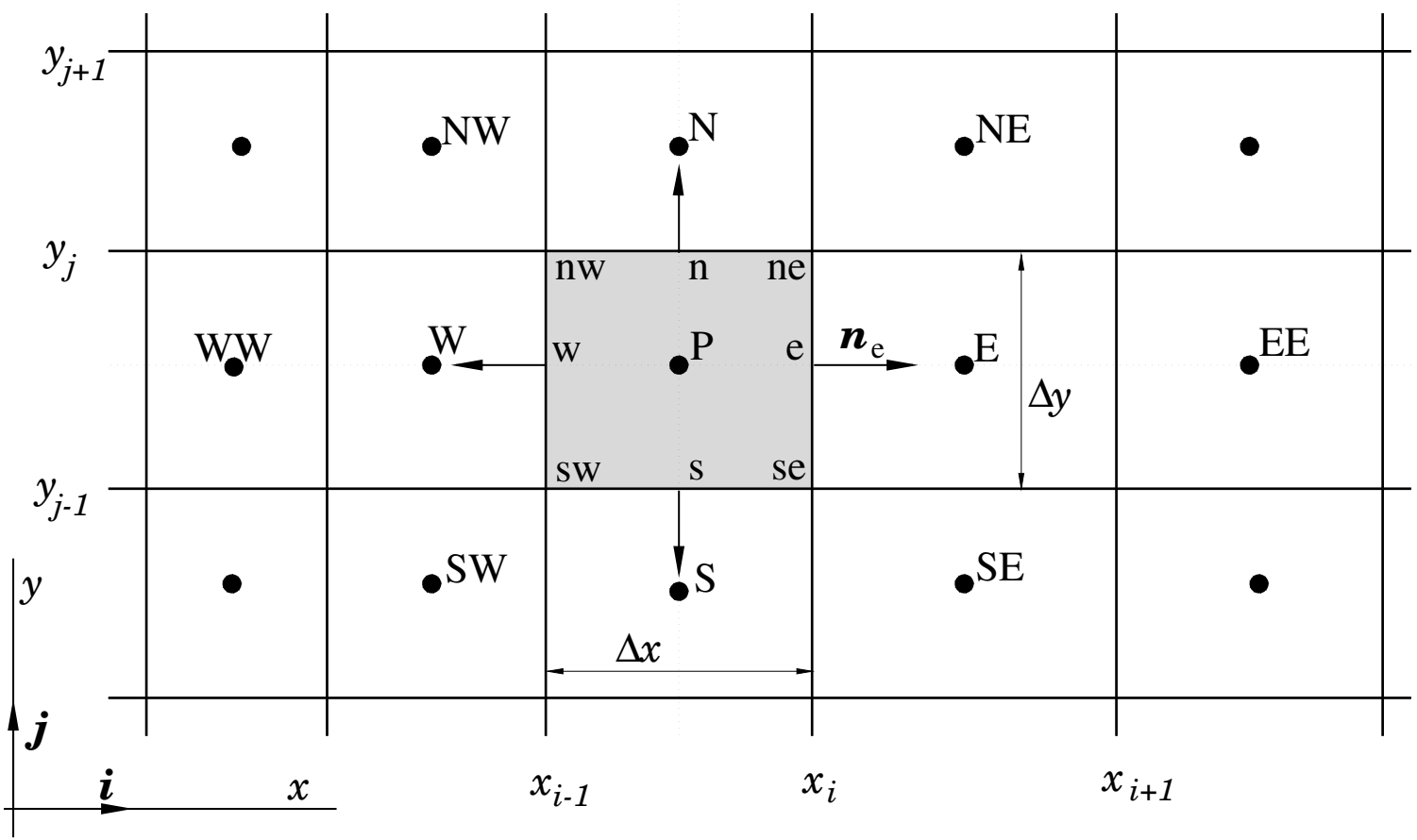


Рис.: A typical CV and the notation used for a Cartesian 2D grid

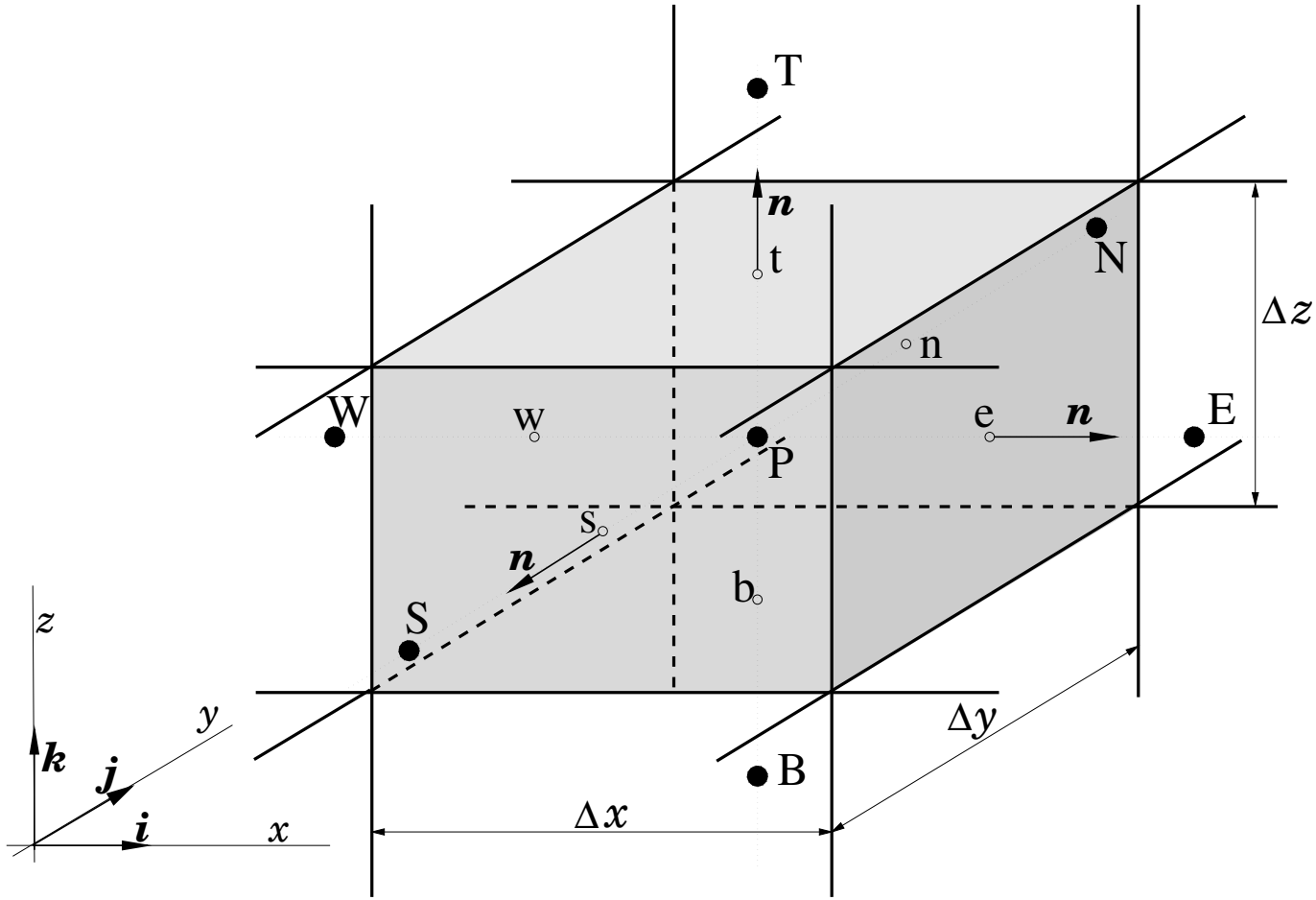


Рис.: A typical CV and the notation used for a Cartesian 3D grid

To calculate the surface integral exactly, one would need to know the integrand f everywhere on the surface S_{e}. This information is not available, as only the nodal (CV center) values of ϕ are calculated so an approximation must be introduced. This is best done using two levels of approximation:

- the integral is approximated in terms of the variable values at one or more locations on the cell face;
- the cell-face values are approximated in terms of the nodal (CV center) values.

The simplest approximation to the integral is the midpoint rule: the integral is approximated as a product of the integrand at the cell-face center (which is itself an approximation to the mean value over the surface) and the cell-face area:

$$F_{\mathrm{e}} = \int_{S_{\mathrm{e}}} f \, \mathrm{d}S = \overline{f}_{\mathrm{e}} S_{\mathrm{e}} \approx f_{\mathrm{e}} S_{\mathrm{e}} \, . \tag{42}$$

This approximation of the integral – provided the value of f at location 'e' is known – is of second-order accuracy.

Another second-order approximation of the surface integral in 2D is the trapezoid rule, which leads to:

$$F_{\mathrm{e}} = \int_{S_{\mathrm{e}}} f \, \mathrm{d}S \approx \frac{S_{\mathrm{e}}}{2} \left(f_{\mathrm{ne}} + f_{\mathrm{se}} \right) \ . \tag{43}$$

In this case we need to evaluate the flux at the CV corners.

Approximation of Volume Integrals

The simplest second-order accurate approximation is to replace the volume integral by the product of the mean value of the integrand and the CV volume and approximate the former as the value at the CV center:

$$Q_{\mathrm{P}} = \int_{\Omega} q \, \mathrm{d}\Omega = \overline{q} \, \Delta\Omega \approx q_{\mathrm{P}} \, \Delta\Omega \, , \tag{44}$$

where q_{P} stands for the value of q at the CV center.

Interpolation and Differentiation Practices

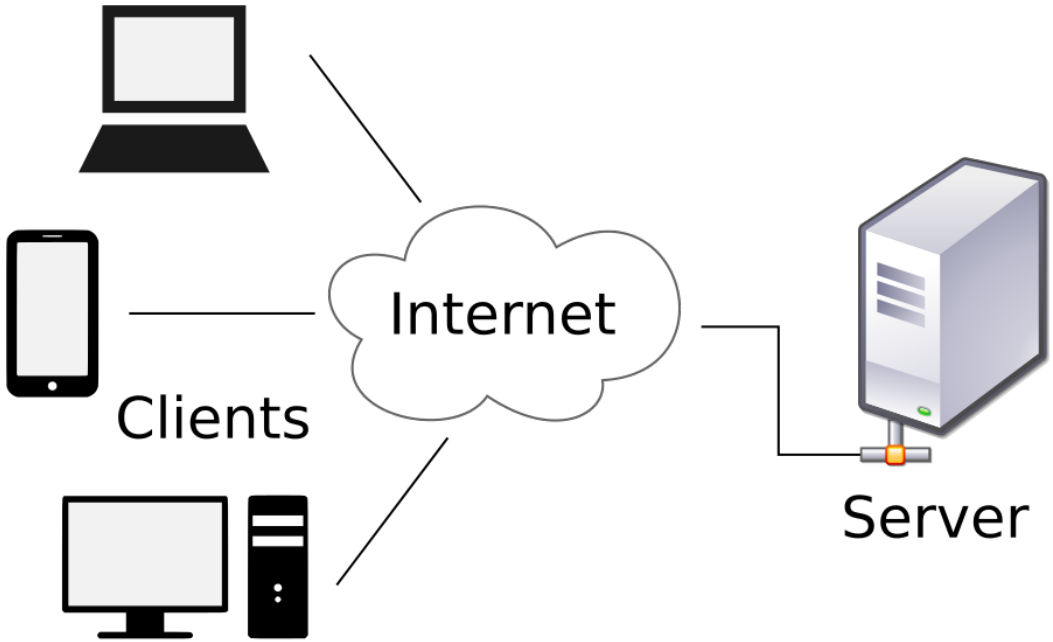
The approximations to the integrals require the values of variables at locations other than computational nodes (CV centers).
$f^{\mathrm{c}} = \rho\phi\vec{v} \cdot \vec{n}$ for the convective flux
$f^{\mathrm{d}} = \Gamma \mathrm{grad}\,\phi \cdot \vec{n}$ for the diffusive flux
We assume that the velocity field and the fluid properties ρ and Γ are known at all locations.
To calculate the convective and diffusive fluxes, the value of ϕ and its gradient normal to the cell face at one or more locations on the CV surface are needed. Volume integrals of the source terms may also require these values. They have to be expressed in terms of the nodal values by interpolation.
How the value of ϕ and its normal derivative at cell face 'e' can be approximated?

The Algebraic Equation System

By summing all the flux approximations and source terms, we produce an algebraic equation which relates the variable value at the center of the CV to the values at several neighbor CVs.
The numbers of equations and unknowns are both equal to the number of CVs so the system is well-posed.
This is true only for structured grids with quadrilateral or hexahedral CVs; for other geometries, the matrix structure will be more complex but it will always be sparse.
The maximum number of elements in any row is equal to the number of near neighbors for second order approximations.
For higher-order approximations, it depends on the number of neighbors used in the scheme.

Client – server model

The providers of a resource or service are called servers and service requesters are called clients.

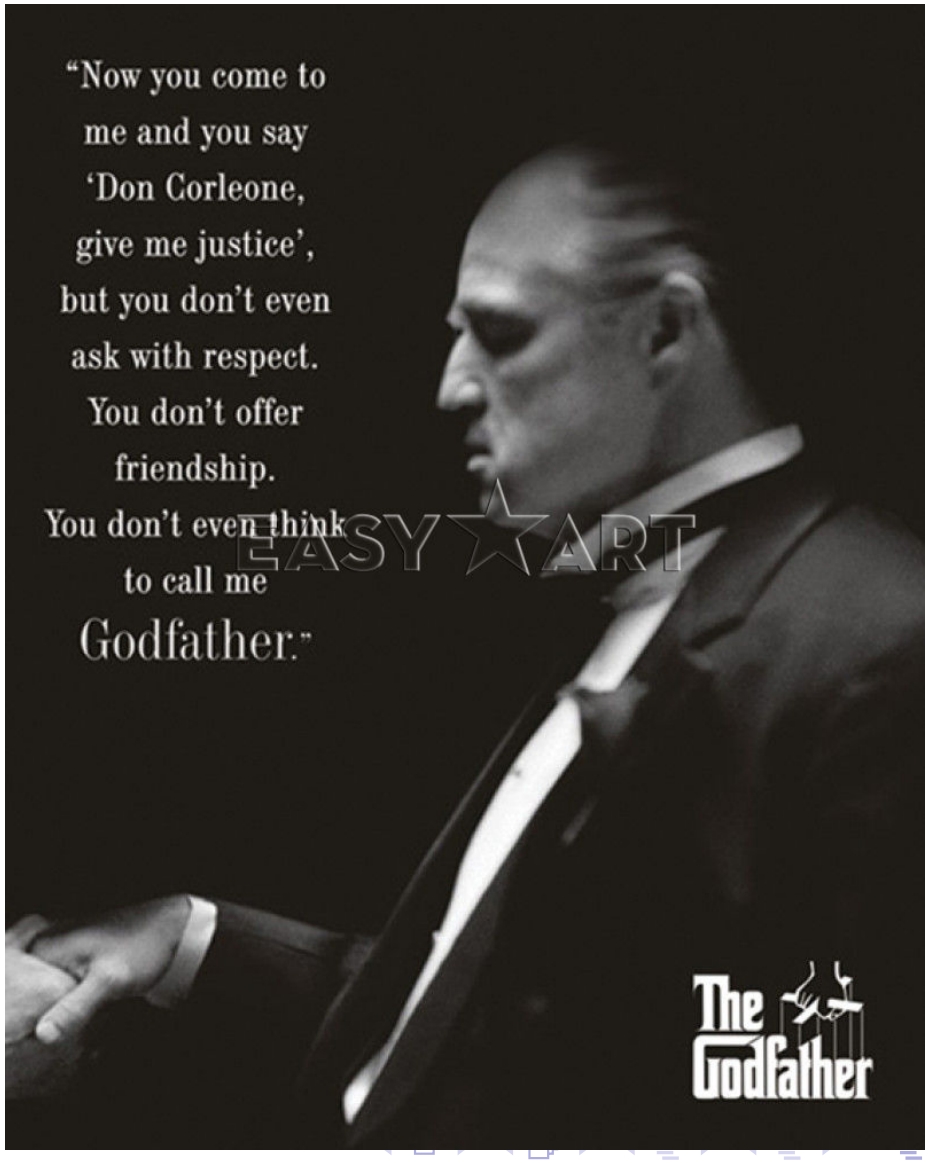


What is relationship of "Client – Server" to "Client – Patron" in Rome (clientela)?

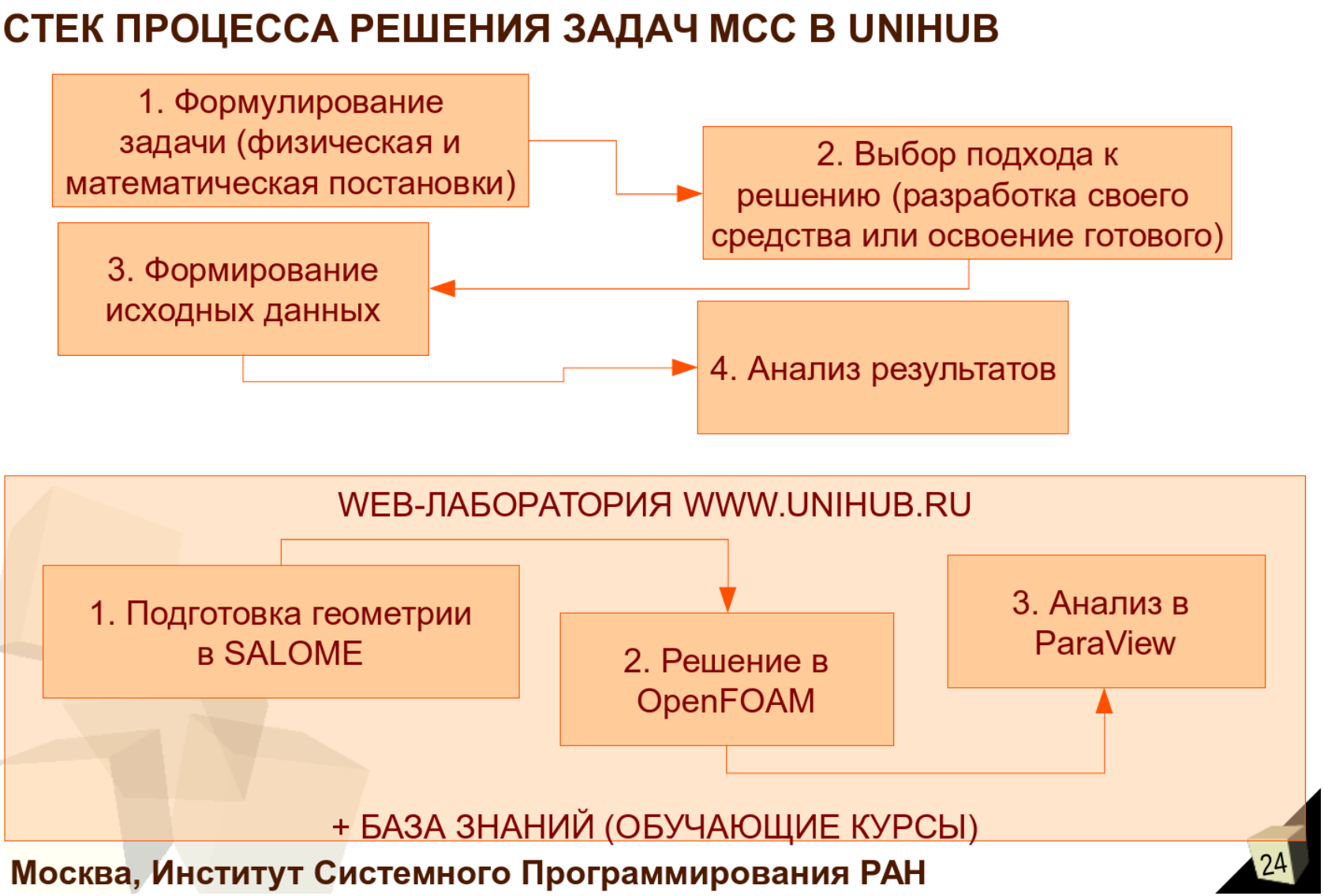
Question asked by Mech. & Math. faculty auditorium – obviously it is not Cibernetics faculty.

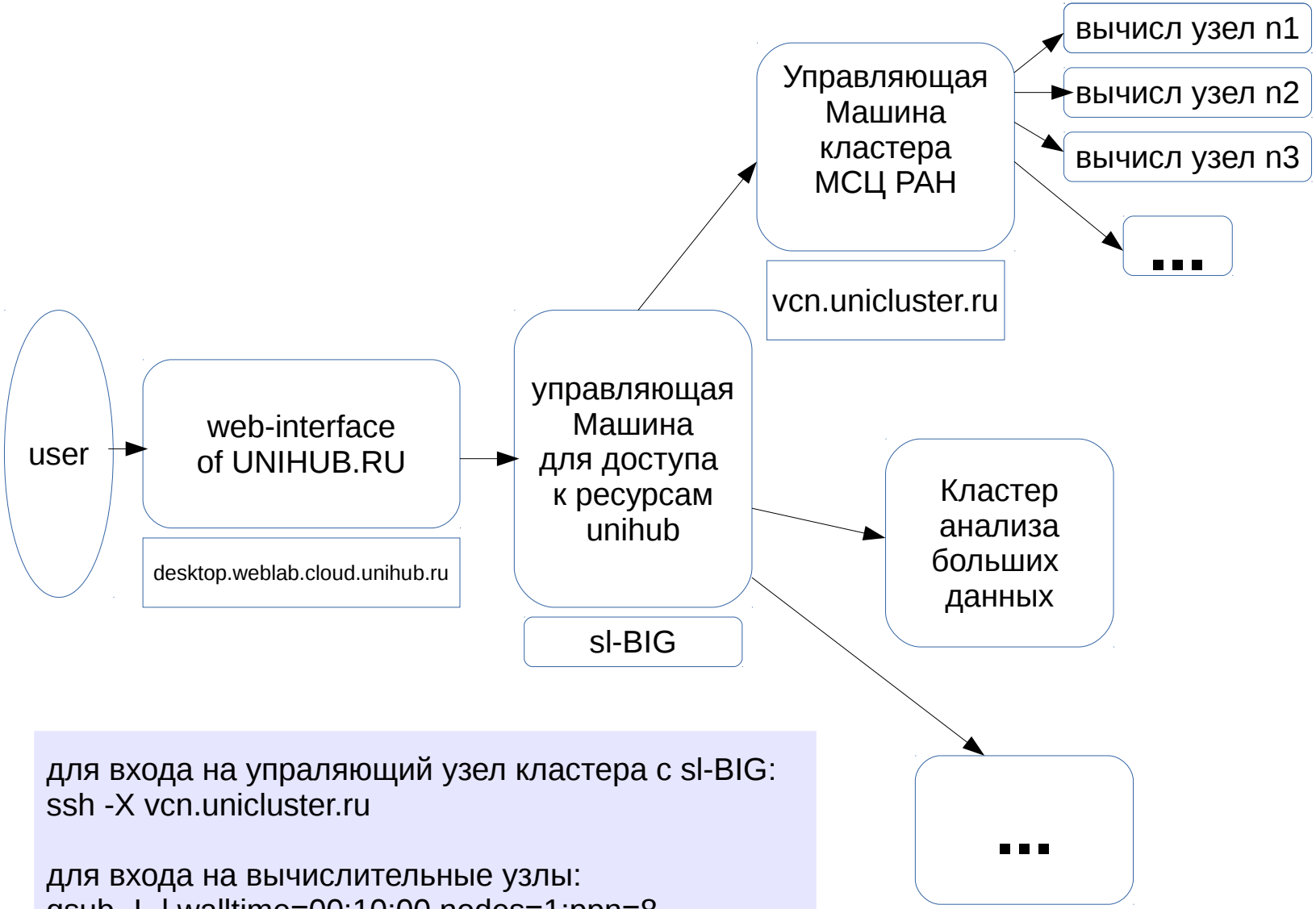
Modern Clientela – Italian Mafia.

At this moment Don Corleone demands: call me a **Server**



Process of numerical simulation on Unihub.ru





для входа на упраляющий узел кластера с sl-BIG:
ssh -X vcn.unicluster.ru

для входа на вычислительные узлы:
qsub -I -l walltime=00:10:00,nodes=1:ppn=8

Access to computational clusters

To enter on the management node of the cluster:

```
ssh -X vcn.unicluster.ru
```

Cluster management

Batch queuing for HPC clusters
An important niche for job schedulers is managing the job queue for a cluster of computers. Typically, the scheduler will schedule jobs from the queue as sufficient resources (cluster nodes) become idle. Some widely used cluster batch systems are Moab, Argent Job Scheduler®, Univa Grid Engine, Portable Batch System, LoadLeveler, Condor, OAR, Simple Linux Utility for Resource Management (SLURM), OpenLava, and IBM's Platform LSF.
TORQUE *The Terascale Open-source Resource and QUEue Manager (TORQUE)* is a distributed resource manager providing control over batch jobs and distributed compute nodes.

Portable Batch System (or simply PBS) is the name of computer software that performs job scheduling. Its primary task is to allocate computational tasks, i.e., batch jobs, among the available computing resources. It is often used in conjunction with UNIX cluster environments.

Background execution (batch mode)

1) Создать задание (myjob):

```
#PBS -l walltime=HH:MM:SS,nodes=[# of nodes]:ppn=[# of cores]
#PBS -q workq@master
#PBS -M samov@ispras.ru
#PBS -m abe
#PBS -N jobname
#!/bin/sh
```

cd [path to the programm/data]

```
mpirun -np [nodes]x[cores] -machinefile $PBS_NODELIST [programs]
```

2) Submit a job in queue for execution

```
qsub myjob
```

3) Job status

```
qstat
qstat -f [ID задания]
```

Interactive execution on computational nodes

1) Submit an interactive job to the queue for execution

```
[master@jx0]$qsub -I -l walltime=HH:MM:SS,nodes=[nodes]:ppn=[cores]
[job ID].[job name]

[master@jx64]$
```

2) Execute a programm

```
[master@jx64]$  cd [path to a programm/data]
[master@jx64]$ mpirun -np [nodes]x[cores] -machinefile $PBS_NODELIS
```

Environmental variables:

```
echo $PBS_ENVIRONMENT
echo $PBS_JOBNAME
echo $PBS_O_PATH
echo $PBS_QUEUE
echo $PBS_JOBID
echo $PBS_NODEFILE
echo $PBS_O_HOST
echo $PBS_O_WORKDIR
...
```

Important Environment Variables

```
$WM_PROJECT_DIR - path to the OpenFOAM installation
$WM_PROJECT_USER_DIR - OpenFOAM user directory
$FOAM_TUTORIALS - OpenFOAM tutorials
$FOAM_SRC - source-tree of OpenFOAM libraries
$FOAM_APP - source-tree of OpenFOAM applications
$FOAM_APPBIN - directory with the applications
$FOAM_USER_APPBIN - directory with the applications created by the u
$FOAM_LIBBIN - directory with the libraries provided by OpenFOAM
$FOAM_USER_LIBBIN - directory with the libraries created by the user
$FOAM_RUN - directory where the user can put his/her cases
```

Important Shell-Aliases

```
run - cd to $FOAM_RUN
src - cd to $FOAM_SRC
app - cd to $FOAM_APP
util - cd to $FOAM_APP/utilities
sol - cd to $FOAM_APP/solvers
```

Basic Case Structure (Базовая структура примера)

Case – boundary value problemm with initial conditions. Задача с граничными условиями и начальными данными.

case – relative or absolute path to the case

case/ – the case directory

+ 0/ – содержит начальные и граничные условия
+ constant/ – constant data (данные и константы)
+ polyMesh/ – содержит данные сетки
+ transportProperties – вязкость
+ system/ – run-time control / numerics
+ controlDict – run-time control (параметры для контроля задачи)
+ fvSchemes – numerical schemes (расчетные схемы)
+ fvSolution – решатели для СЛАУ

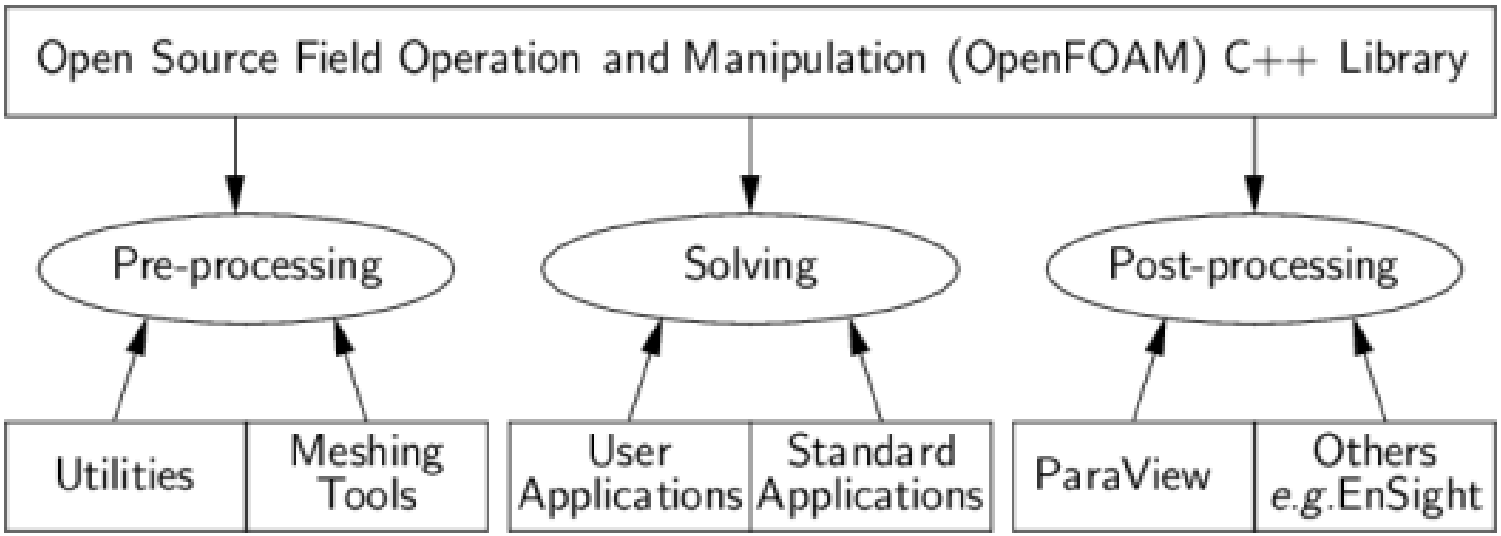
case/0/ – contains for each variable a fle defning the initial and boundary conditions. May also contain initial and boundary conditions for a moving grid.

case/constant/polyMesh/ – contains the grid data for a non-moving grid. The fles are: boundary, faces, neighbour, owner, points.

case/constant/transportProperties – defnes the viscosity (also for

OpenFOAM structure

OpenFOAM consists of solvers and utilities.
Solvers are designed to solve a specific kind of problem in continuum mechanics.
Utilities are designed for data manipulation.

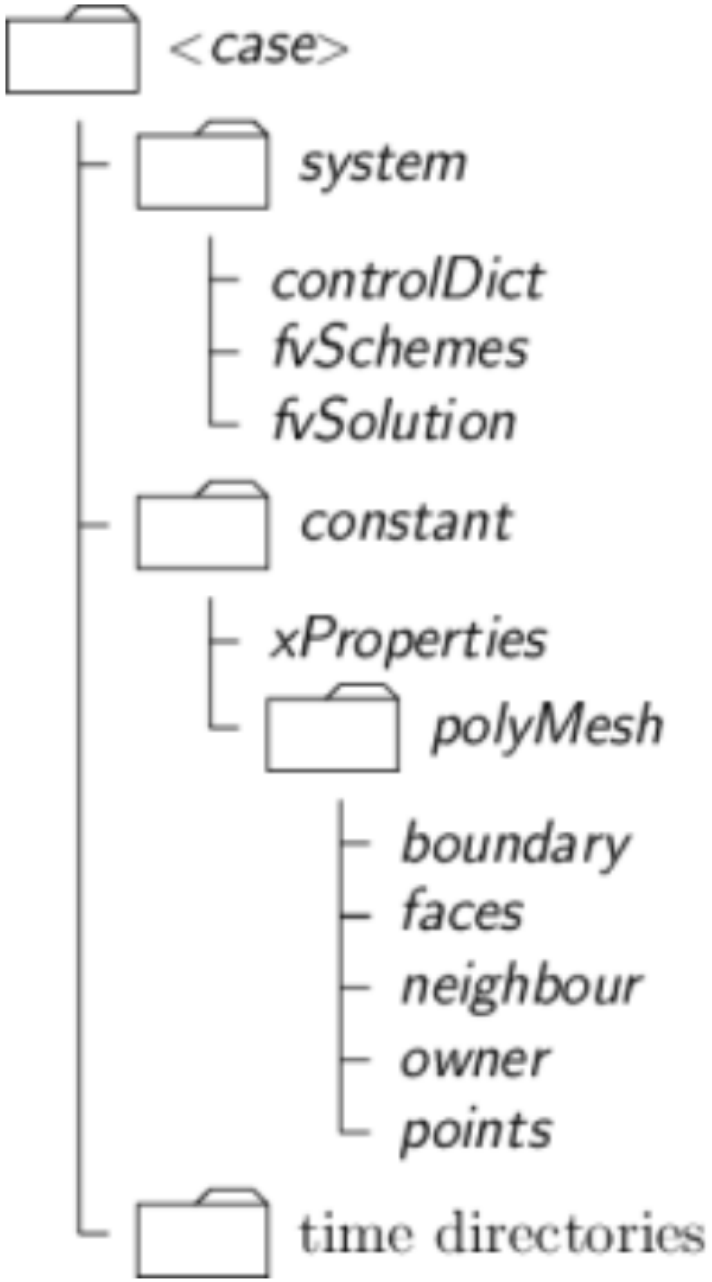


OpenFOAM tutorials

Tutorials are located in directory, indicated by `$FOAM_TUTORIALS` environment variable. Tutorials are grouped by directories with descriptive names:

```
DNS
basic
combustion
compressible
discreteMethods
electromagnetics
financial
heatTransfer
incompressible
lagrangian
mesh
multiphase
resources
stressAnalysis
```

Case structure

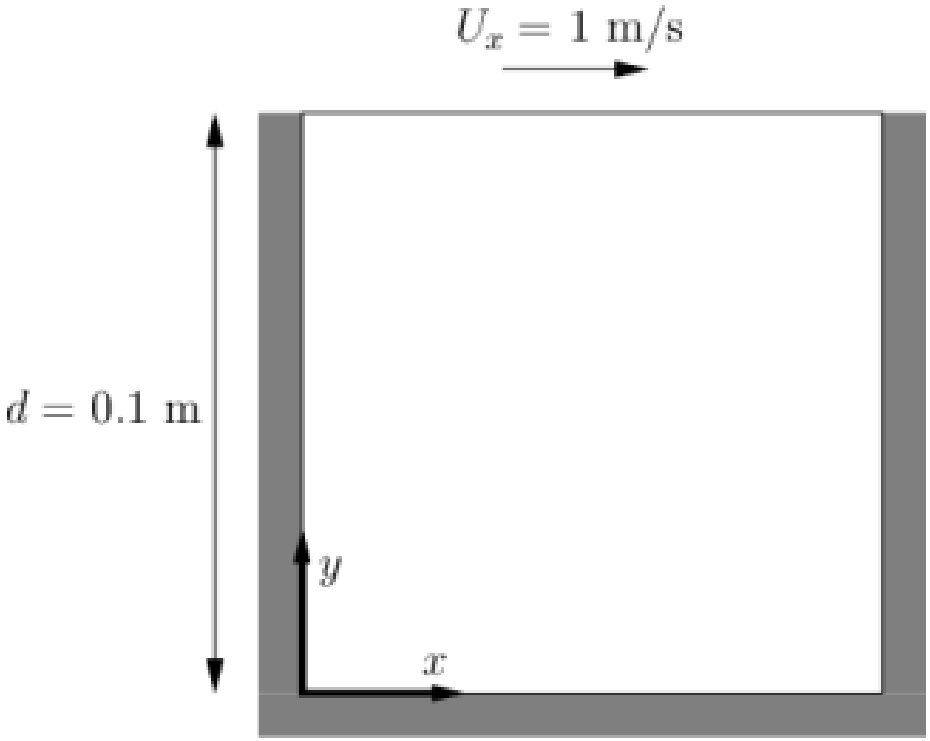


Constant directory contains a description of the case mesh in a subdirectory polyMesh and files specifying physical properties for the application concerned, e.g.transportProperties.

System directory for setting parameters associated with the solution procedure itself. It contains at least the following 3 files: controlDict where run control parameters are set including start/end time, time step and parameters for data output; fvSchemes where discretisation schemes used in the solution may be selected at run-time; and, fvSolution where the equation solvers, tolerances and other algorithm controls are set for the run.

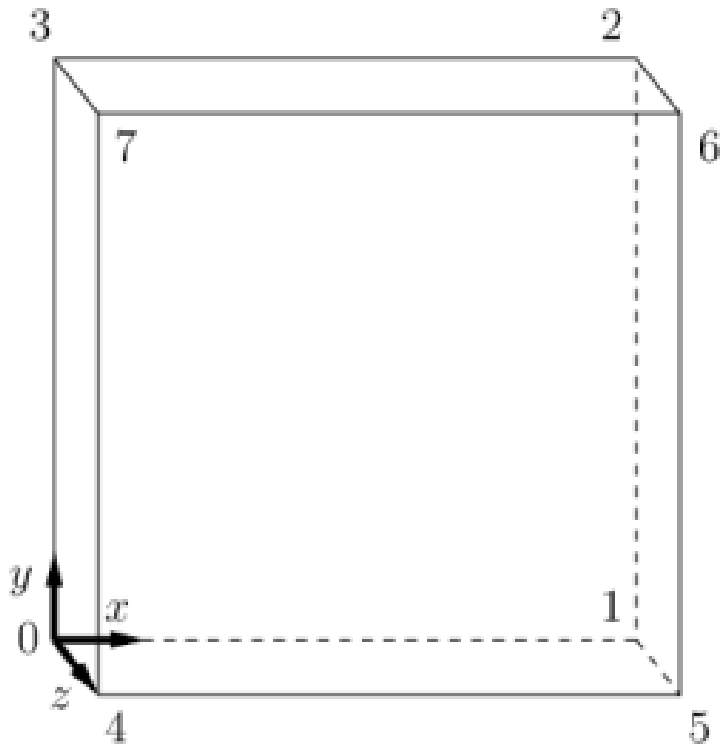
The 'time' directories Initial values and boundary conditions that the user must specify to define the problem; or, results written to file by OpenFOAM.

Case study: Lid-driven cavity flow



Meshing

The most basic and simple mesh generation tool for OpenFOAM is blockMesh utility. It can create a structured mesh in a rectangular geometry. The mesh is described in the file blockMeshDict located in `system` directory of current case.



```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  3.0.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.1;

vertices
(
    (0 0 0)
    (1 0 0)
    (1 1 0)
    (0 1 0)
    (0 0 0.1)
    (1 0 0.1)
    (1 1 0.1)
    (0 1 0.1)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (1 1 1)
);

edges
(
```

```
edges
(
);

boundary
(
    movingWall
    {
        type wall;
        faces
        (
            (3 7 6 2)
        );
    }
    fixedWalls
    {
        type wall;
        faces
        (
            (0 4 7 3)
            (2 6 5 1)
            (1 5 4 0)
        );
    }
    frontAndBack
    {
        type empty;
        faces
        (
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);
```

Boundary conditions

Boundary and initial conditions are usually set in the subdirectory 0 of the case in file, whose name correspond to names of variables.
To set dimensions of variables `dimensionSet` entry is used.

No.	Property	SI unit	USCS unit
1	Mass	kilogram (kg)	pound-mass (lbm)
2	Length	metre (m)	foot (ft)
3	Time	— — — — second (s)	— — — —
4	Temperature	Kelvin (K)	degree Rankine (°R)
5	Quantity	kilogram-mole (kgmol)	pound-mole (lbmol)
6	Current	— — — — ampere (A)	— — — —
7	Luminous intensity	— — — — candela (cd)	— — — —

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  3.0.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    movingWall
    {
        type            zeroGradient;
    }

    fixedWalls
    {
        type            zeroGradient;
    }

    frontAndBack
    {
        type            empty;
    }
}
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  3.0.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    movingWall
    {
        type            fixedValue;
        value           uniform (1 0 0);
    }

    fixedWalls
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    frontAndBack
    {
        type            empty;
```

Physical properties

Physical properties are set in constant subdirectory of a case in files, with names ending by `..Properties`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  3.0.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

nu              [0 2 -1 0 0 0 0] 0.01;


// ************************************************************************* //
```

Solution control

Start and end- time of execution, timestep and write control are set in `controlDict` file of `constant` subdirectory
Ensure that Courant number is less than 1:

$$Co = \frac{\delta t}{\delta x}|U|$$

$$\delta x = \frac{d}{n} = \frac{0.1}{20} = 0.005m$$

$$\delta t < \frac{0.005}{|U| = 1} = 0.005$$

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  3.0.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     icoFoam;
startFrom       startTime;
startTime       0;
stopAt          endTime;
endTime         0.5;
deltaT          0.005;
writeControl    timeStep;
writeInterval   20;
purgeWrite      0;
writeFormat     ascii;
writePrecision  6;
writeCompression off;
timeFormat      general;
timePrecision   6;
runTimeModifiable true;
```

Discretisation and linear-solver settings

Finite volume discretisation schemes are set in the fvSchemes dictionary in the system directory.
The specification of the linear equation solvers and tolerances and other algorithm controls is made in the fvSolution dictionary,

Postprocessing

- Viewing the mesh
- Isosurface and contour plots
- Vector plots
- Streamline plots

Increasing the mesh resolution. Creating the finer mesh

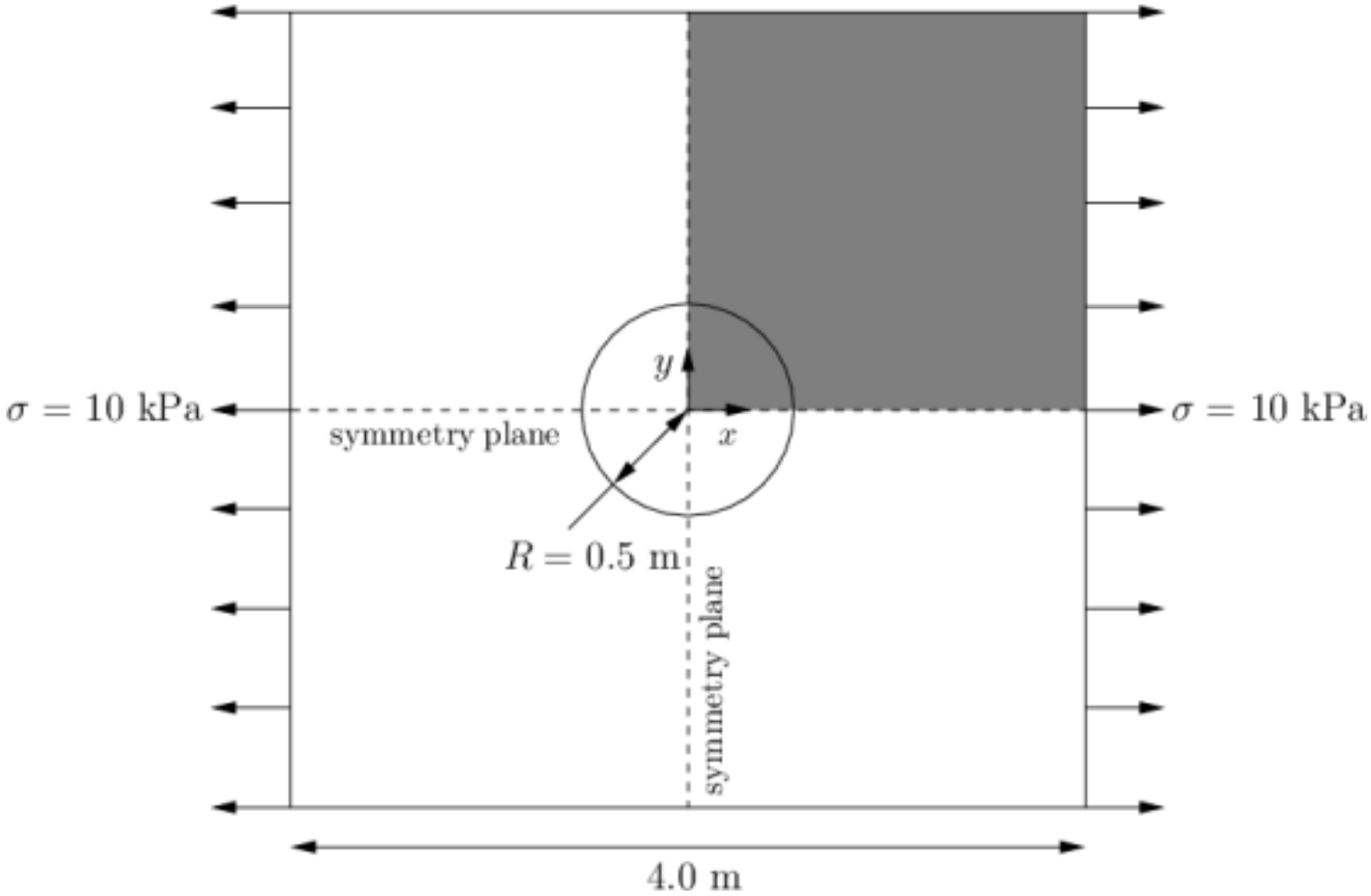
Mapping the coarse mesh results onto the fine mesh
`mapFields` utility maps one or more fields relating to a given geometry onto the corresponding fields for another geometry.
The field data that mapFields maps is read from the time directory specified by startFrom/startTime in the controlDict of the target case, i.e. those into which the results are being mapped.
Control adjustments: `deltaT` -> 0.0025, `writeControl`: `timeStep` -> `runTime`, `writeInterval` -> 0.1

Plotting graphs

```
foamCalc <calcType> <fieldName1 ... fieldNameN>
```

```
cd $FOAM_RUN/tutorials/incompressible/icoFoam/cavity
foamCalc components U
```

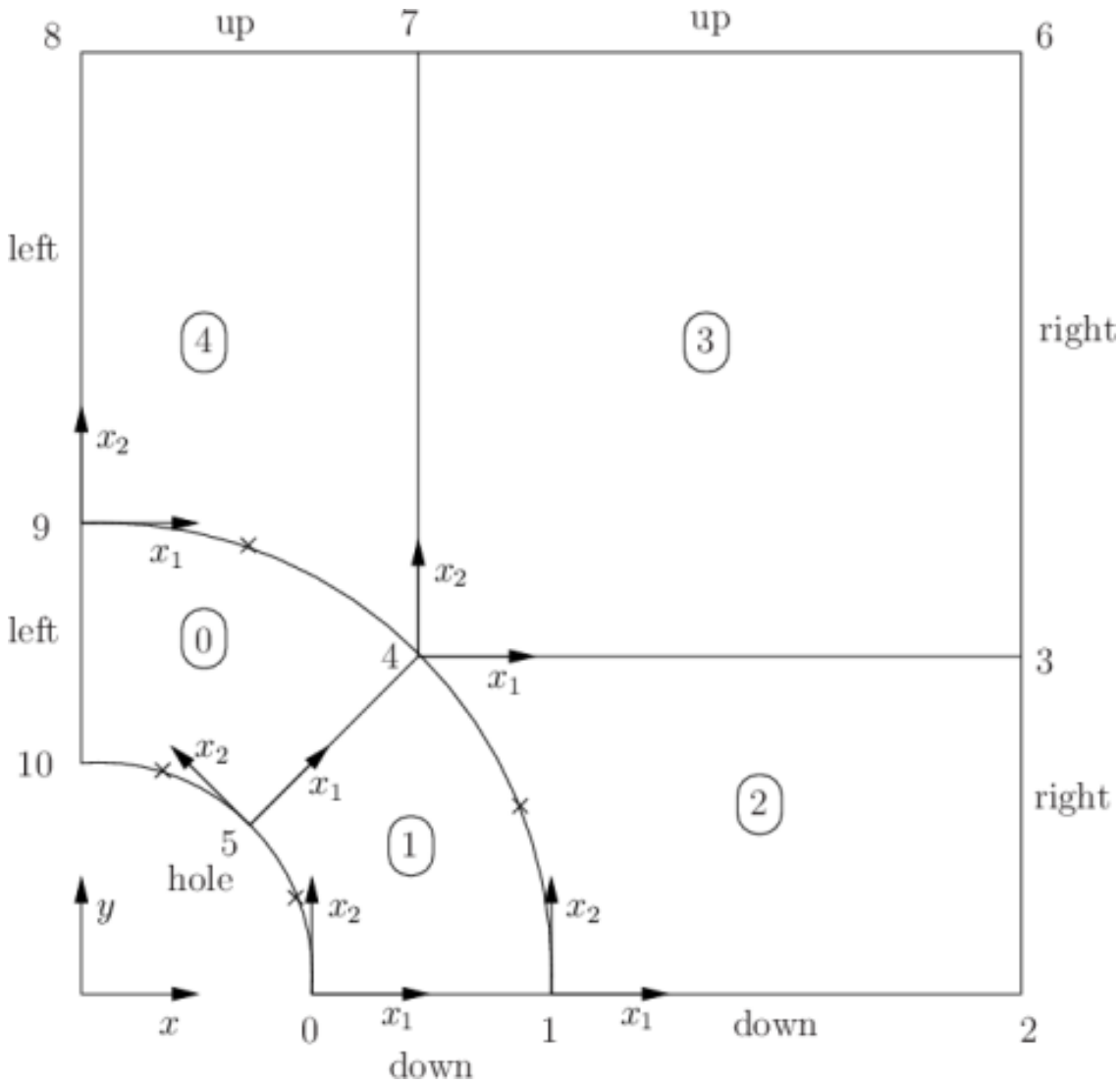
Stress analysis of a plate with a hole



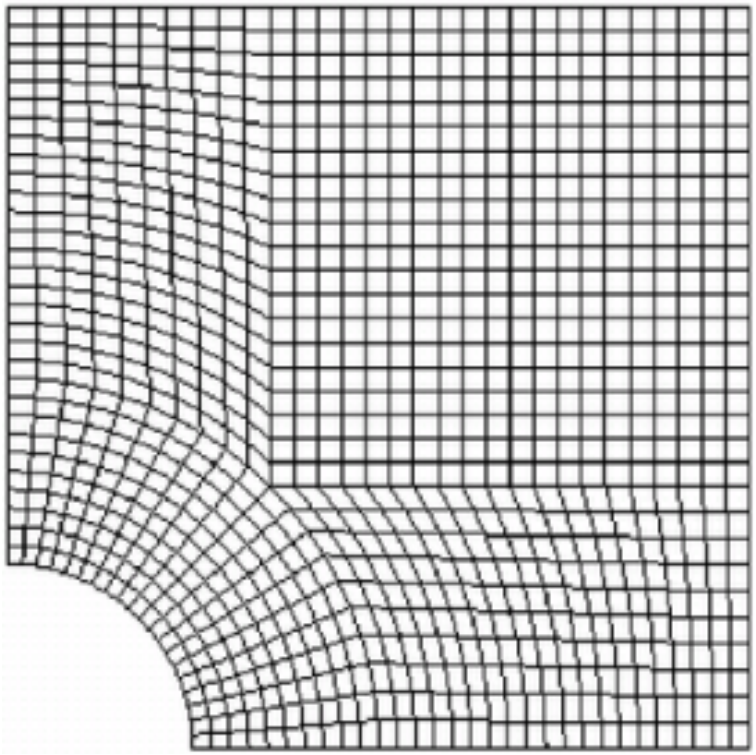
Stress analysis of a plate with a hole

$$(\sigma_{xx})_{x=0} = \begin{cases} \sigma\left(1 + \dfrac{R^2}{2y^2} + \dfrac{3R^4}{2y^4}\right) & \text{for } |y| \geq R \\ 0 & \text{for } |y| < R \end{cases}$$

Stress analysis of a plate with a hole



Stress analysis of a plate with a hole



Stress analysis of a plate with a hole

blockMeshDict I

```
convertToMeters 1;
vertices
(
    (0.5 0 0)
    (1 0 0)
    (2 0 0)
    (2 0.707107 0)
    (0.707107 0.707107 0)
    (0.353553 0.353553 0)
    (2 2 0)
    (0.707107 2 0)
    (0 2 0)
    (0 1 0)
    (0 0.5 0)
    (0.5 0 0.5)
    (1 0 0.5)
    (2 0 0.5)
    (2 0.707107 0.5)
    (0.707107 0.707107 0.5)
    (0.353553 0.353553 0.5)
    (2 2 0.5)
    (0.707107 2 0.5)
    (0 2 0.5)
    (0 1 0.5)
    (0 0.5 0.5)
);
blocks
(
    hex (5 4 9 10 16 15 20 21) (10 10 1) simpleGrading (1 1 1)
    hex (0 1 4 5 11 12 15 16) (10 10 1) simpleGrading (1 1 1)
    hex (1 2 3 4 12 13 14 15) (20 10 1) simpleGrading (1 1 1)
    hex (4 3 6 7 15 14 17 18) (20 20 1) simpleGrading (1 1 1)
```

Stress analysis of a plate with a hole

blockMeshDict II

```
edges
(
    arc 0 5 (0.469846 0.17101 0)
    arc 5 10 (0.17101 0.469846 0)
    arc 1 4 (0.939693 0.34202 0)
    arc 4 9 (0.34202 0.939693 0)
    arc 11 16 (0.469846 0.17101 0.5)
    arc 16 21 (0.17101 0.469846 0.5)
    arc 12 15 (0.939693 0.34202 0.5)
    arc 15 20 (0.34202 0.939693 0.5)
);
boundary
(
    left
    {
        type symmetryPlane;
        faces
        (
            (8 9 20 19)
            (9 10 21 20)
        );
    }
    right
    {
        type patch;
        faces
        (
            (2 3 14 13)
            (3 6 17 14)
        );
    }
    down
```

Applications and libraries

solvers are each designed to solve a specific problem in computational continuum mechanics;

utilities perform simple pre-and post-processing tasks, mainly involving data manipulation and algebraic calculations.

The programming language of OpenFOAM

OpenFOAM is written in object-oriented C++

Equations representations:

$$\frac{\partial \rho U}{\partial t} + \nabla \cdot (\phi U) - \nabla \cdot (\mu \nabla U) = -\nabla p$$

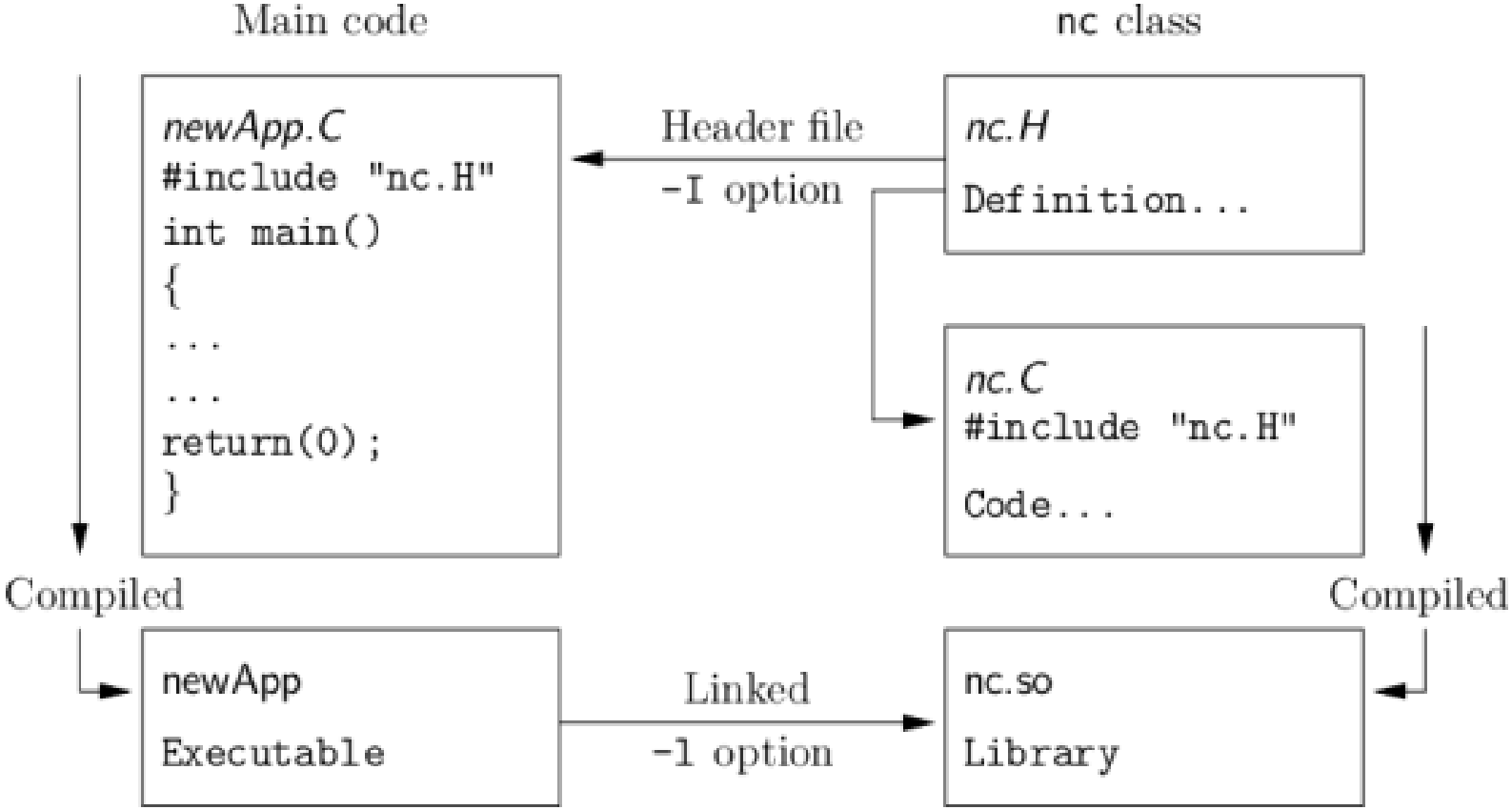
```
solve
  (
      fvm::ddt(rho, U)
    + fvm::div(phi, U)
    - fvm::laplacian(mu, U)
      ==
    - fvc::grad(p)
  );
```

Solver codes are documented with Doxygen at

`$WM_PROJECT_DIR/doc/Doxygen/html/index.html`

OpenFOAM classes can be compiled to shared object library with .so extensions.

Later they can be user in other programms or classed with dynamic linking



Class declaration

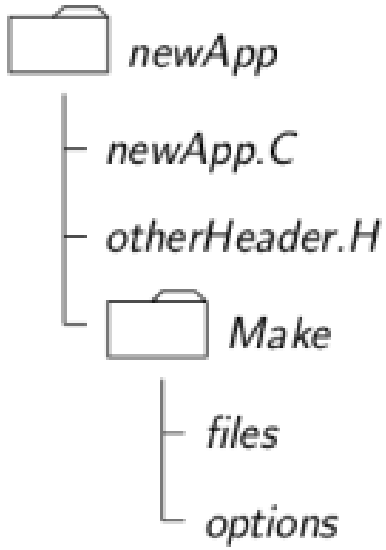
Each class requires a class declaration, contained in a header file with a .H file extension, e.g.nc.H, that includes the names of the class and its functions.
This file is included at the beginning of any piece of code using the class, including the class declaration code itself.
.C code can resource any number of classes and must begin with all the .H files required to declare these classes.
The classes in turn can resource other classes and begin with the relevant .H files. By searching recursively down the class hierarchy we can produce a complete list of header files for all the classes on which the top level .C code ultimately depends; these .H files are known as the dependencies.
With a dependency list, a compiler can check whether the source files have been updated since their last compilation and selectively compile only those that need to be.
Header files are included in the code using # include statements, e.g.

```
#include "otherHeader.H";
```

Building with wmake

- Automatic generation and maintenance of file dependency lists, i.e. lists of files which are included in the source files and hence on which they depend.
- Multi-platform compilation and linkage, handled through appropriate directory structure.
- Multi-language compilation and linkage, e.g. C, C++, Java.
- Multi-option compilation and linkage, e.g. debug, optimised, parallel and profiling.
- Support for source code generation programs, e.g. lex, yacc, IDL, MOC.
- Simple syntax for source file lists.
- Automatic creation of source file lists for new codes.
- Simple handling of multiple shared or static libraries.
- Extensible to new machine types.
- Extremely portable, works on any machine with: make; sh, ksh or csh; lex, cc.
- Has been tested on Apollo, SUN, SGI, HP (HPUX), Compaq (DEC), IBM (AIX), Cray, Ardent, Stardent, PC Linux, PPC Linux, NEC, SX4, Fujitsu VP1000.

Compiling with wmake



Including headers: trough `Make/options`

```
EXE_INC = \
    -I<directoryPath1> \
    -I<directoryPath2> \
    ...                \
    -I<directoryPathN>
```

Linking to libraries

Make/options file

```
EXE_LIBS = \
    -L<libraryPath1> \
    -L<libraryPath2> \
    ...              \
    -L<libraryPathN> \
    -l<library1>     \
    -l<library2>     \
    ...              \
    -l<libraryN>
```

Source files to be compiled

The full list of .C source files must be included in the Make/files file
Most simple case:

```
newApp.C

EXE = $(FOAM_USER_APPBIN)/newApp
```

To run wmake:

```
wmake <optionalArguments> <optionalDirectory>
```

Options:

- all wmake all subdirectories, running Allwmake files if present
- exe Compile statically linked executable
- lib Compile statically linked archive lib (.a)
- libo Compile statically linked lib (.o)
- libso Compile dynamically linked lib (.so)
- dep Compile lnInclude and dependencies only

Removing dependency lists: wclean and rmdepall

```
wclean <optionalArguments> <optionalDirectory>
```

Adding temperature transport to icoFoam

Let's check where `$FOAM_SOLVERS` enviroment variable points to:

```
echo $FOAM_SOLVERS
ls $FOAM_SOLVERS
```

Yes, it does point to the residence of all the OpenFOAM default solvers. Now we will first go to the directory where we want to keep the source codes of our own solvers, f.e. `$HOME/calc/source/OpenFOAM` and we will copy icoFoam solver to that direcory

```
cp -rf $FOAM_SOLVERS/incompressible/icoFoam icoFoamHeat
```

Now change the filename of the `.C` source file of the new solver:

```
mv icoFoam.C icoFoamHeat.C
```

Next, you have to change `Make/files` (3! changes):

```
icoFoam.C -> icoFoamHeat.C
EXE = $(FOAM_USER_APPBIN)/icoFoam
            -> EXE = $(FOAM_USER_APPBIN)/icoFoamHeat
```

After you have compiled your new solver, you have, *of course*, to change the name of the solver in `system/controlDict`

SIMPLE algorithm, SIMPLE is an acronym for Semi-Implicit Method for Pressure Linked Equations.

1. Set the boundary conditions.
2. Compute the gradients of velocity and pressure.
3. Solve the discretized momentum equation to compute the intermediate velocity field.
4. Compute the uncorrected mass fluxes at faces.
5. Solve the pressure correction equation to produce cell values of the pressure correction.
6. Update the pressure field: $p^{k+1} = p^k + \text{urf} \cdot p^{'}$ where urf is the under-relaxation factor for pressure.
7. Update the boundary pressure corrections $p_b^{'}$.
8. Correct the face mass fluxes: $\dot{m}_f^{k+1} = \dot{m}_f^* + \dot{m}_f^{'}$
9. Correct the cell velocities: $\vec{v}^{k+1} = \vec{v}^* - \frac{\text{Vol } \nabla p^{'}}{\vec{a}_P^v}; where \nabla p^{'}$ is the gradient of the pressure corrections, $\vec{a}_P^v$ is the vector of central coefficients for the discretized linear system representing the velocity equation and Vol is the cell volume.

PISO algorithm (Pressure Implicit with Splitting of Operator)

1. Set the boundary conditions.
2. Solve the discretized momentum equation to compute an intermediate velocity field.
3. Compute the mass fluxes at the cells faces.
4. Solve the pressure equation.
5. Correct the mass fluxes at the cell faces.
6. Correct the velocities on the basis of the new pressure field.
7. Update the boundary conditions.
8. Repeat from 3 for the prescribed number of times.
9. Increase the time step and repeat from 1.