Numerics and Boundary Conditions (used in PALM)

PALM group

Institute of Meteorology and Climatology, Leibniz Universität Hannover

last update: 21st September 2015

Overview

PALM is (almost) using simple, standard and fast numerical schemes:

Overview

PALM is (almost) using simple, standard and fast numerical schemes:

- **Spatial and temporal discretization by finite differences**

Overview

PALM is (almost) using simple, standard and fast numerical schemes:

- **Spatial and temporal discretization by finite differences**
- **Explicit timestep methods:**
  - Euler
  - Runge-Kutta, second or third order

Overview

PALM is (almost) using simple, standard and fast numerical schemes:

- **Spatial and temporal discretization by finite differences**
- **Explicit timestep methods:**
  - Euler
  - Runge-Kutta, second or third order
- **Advection method**
  - Upstream
  - Piacsek-Williams (second order central finite differences)
  - Bott-Chlond-scheme (monotone, positiv definit, for scalars only)
  - 5th-order scheme of Wicker and Skamarock, (as used in WRF model)

Overview

PALM is (almost) using simple, standard and fast numerical schemes:

- **Spatial and temporal discretization by finite differences**
- **Explicit timestep methods:**
  - Euler
  - Runge-Kutta, second or third order
- **Advection method**
  - Upstream
  - Piacsek-Williams (second order central finite differences)
  - Bott-Chlond-scheme (monotone, positiv definit, for scalars only)
  - 5th-order scheme of Wicker and Skamarock, (as used in WRF model)
- **Poisson-equation for pressure**
  - Direct FFT-method
  - Multigrid-method

Overview

PALM is (almost) using simple, standard and fast numerical schemes:

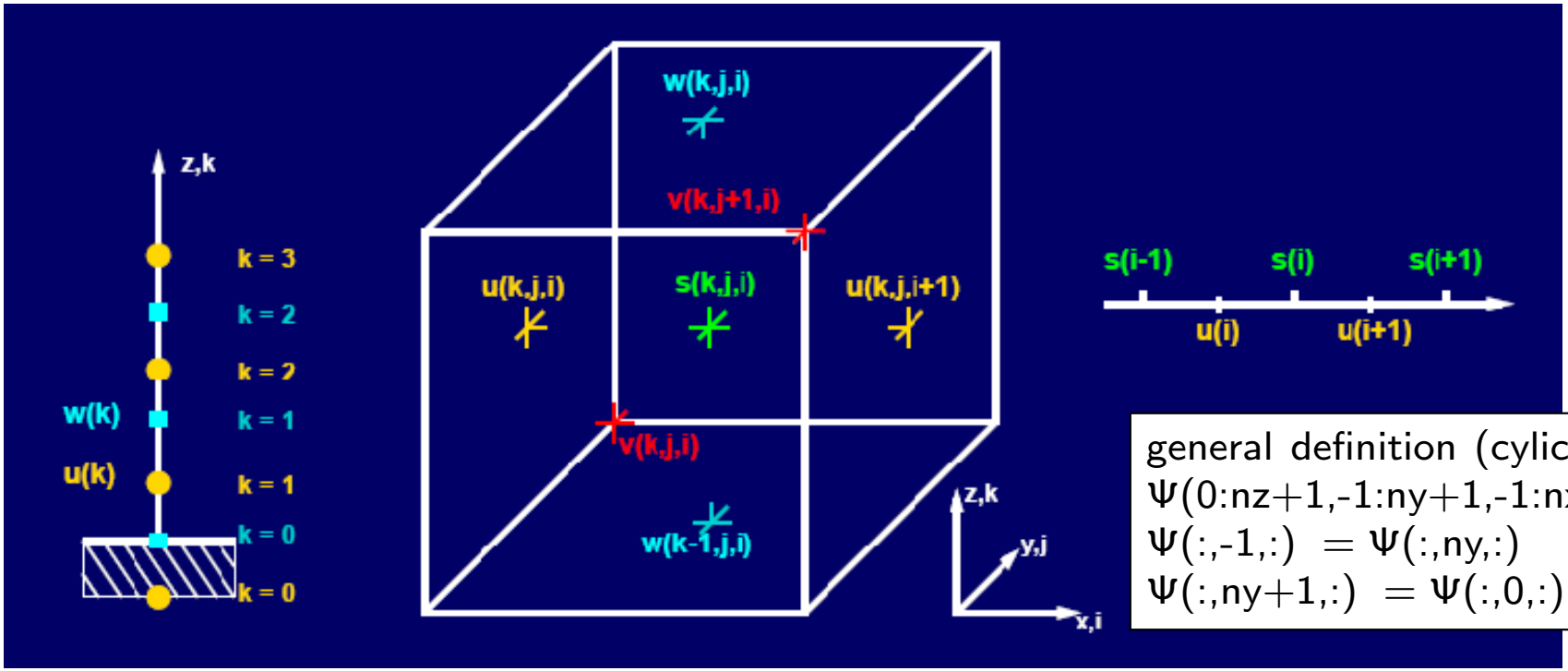
- **Spatial and temporal discretization by finite differences**
- **Explicit timestep methods:**
  - Euler
  - Runge-Kutta, second or third order
- **Advection method**
  - Upstream
  - Piacsek-Williams (second order central finite differences)
  - Bott-Chlond-scheme (monotone, positiv definit, for scalars only)
  - 5th-order scheme of Wicker and Skamarock, (as used in WRF model)
- **Poisson-equation for pressure**
  - Direct FFT-method
  - Multigrid-method
- **Lagrangian particle model included**

Overview

PALM is (almost) using simple, standard and fast numerical schemes:

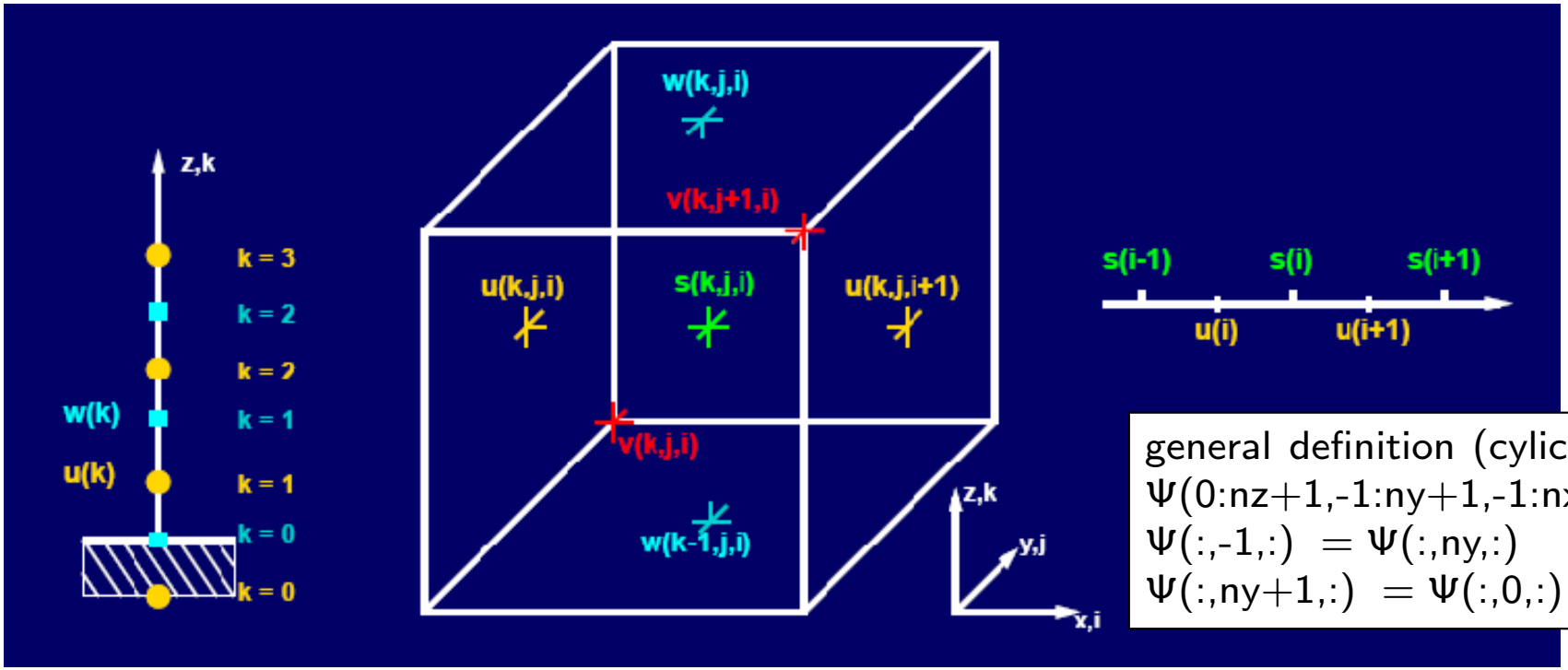
- **Spatial and temporal discretization by finite differences**
- **Explicit timestep methods:**
  - Euler
  - Runge-Kutta, second or third order
- **Advection method**
  - Upstream
  - Piacsek-Williams (second order central finite differences)
  - Bott-Chlond-scheme (monotone, positiv definit, for scalars only)
  - 5th-order scheme of Wicker and Skamarock, (as used in WRF model)
- **Poisson-equation for pressure**
  - Direct FFT-method
  - Multigrid-method
- **Lagrangian particle model included**
- **Boundary conditions:**
  - Cyclic and non-cyclic horizontal boundary conditions
  - Surface layer with Monin-Obukhov similarity
  - Topography
  - Turbulent inflow (for non-cyclic horizontal boundary conditions)

Numerical Grid



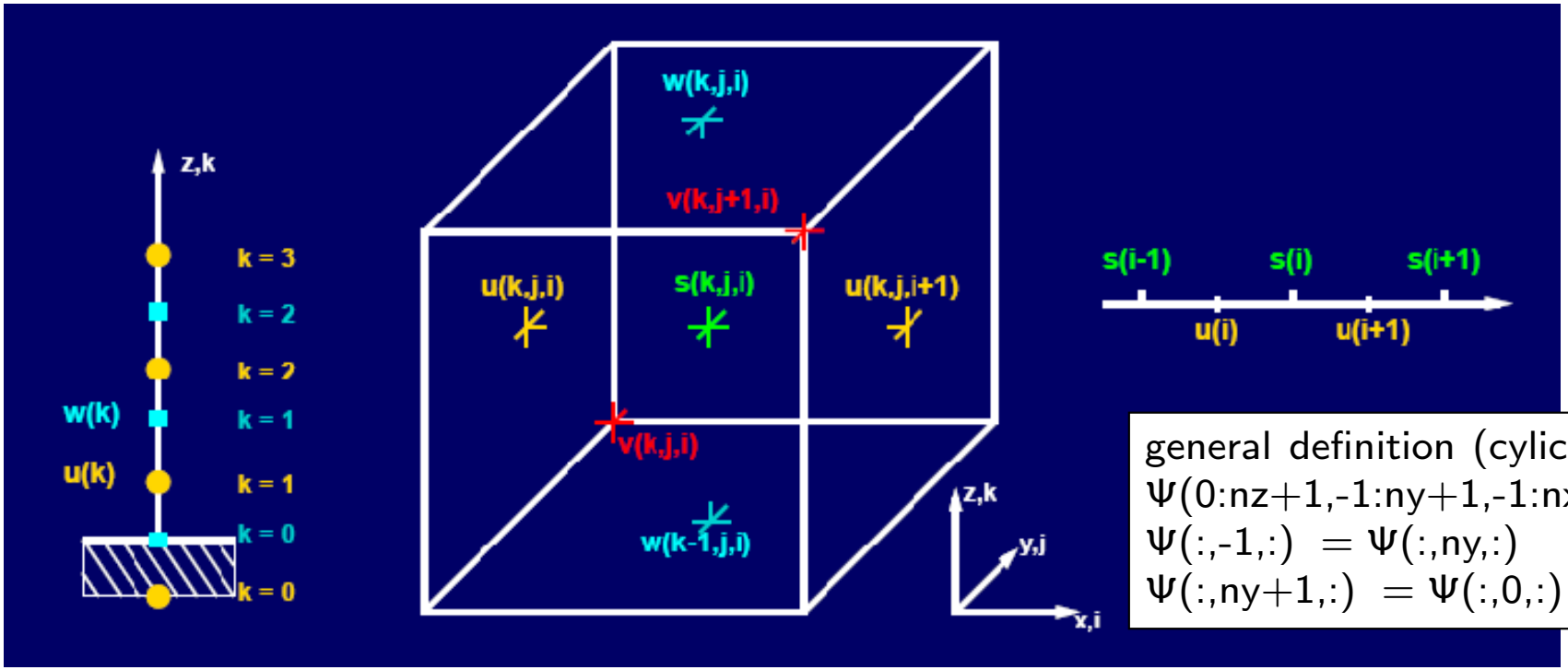
- Equations are spatially discretized on an Arakawa-C grid.

Numerical Grid



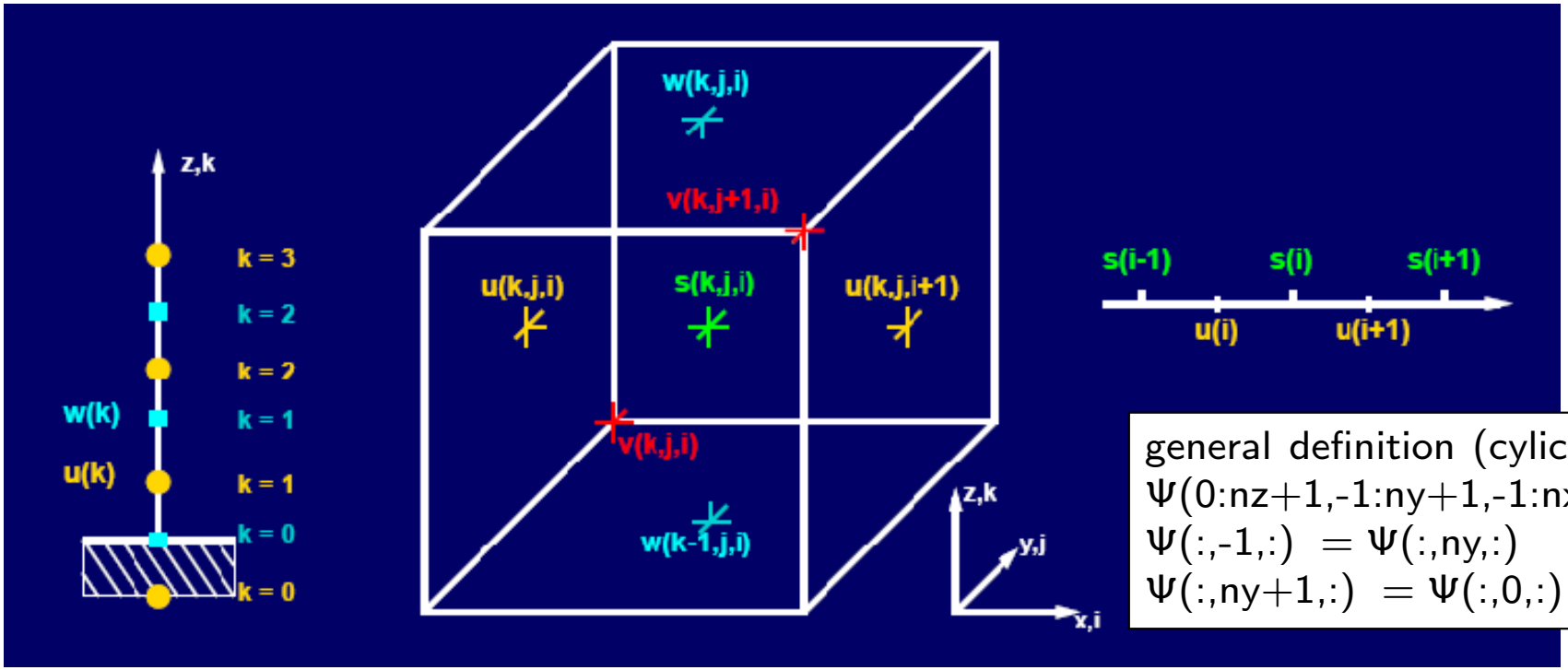
- Equations are spatially discretized on an Arakawa-C grid.
- All scalar variables s (e.g. , p^*, e, K_m, K_h) are defined at the cell centers.

Numerical Grid



- Equations are spatially discretized on an Arakawa-C grid.
- All scalar variables s (e.g. , p^*, e, K_m, K_h) are defined at the cell centers.
- Velocity components (u, v, w) are shifted by half of the grid spacing.

Numerical Grid



- Equations are spatially discretized on an Arakawa-C grid.
- All scalar variables s (e.g. , p^*, e, K_m, K_h) are defined at the cell centers.
- Velocity components (u, v, w) are shifted by half of the grid spacing.
- Spacings are equidistant, stretching along z is possible.

Timestep Methods (I)

- **Euler**

$$\frac{\partial \psi(t)}{\partial t} = F(\psi(t)) \rightarrow \frac{\psi(t+\Delta t) - \psi(t)}{\Delta t} \approx F(\psi(t))$$

$$\psi(t+\Delta t) = \psi(t) + \Delta t \cdot F(\psi(t)) \qquad \mathcal{O}(\Delta t)$$

(used for SGS-TKE in special cases)

Timestep Methods (I)

- **Euler**

$$\frac{\partial \psi(t)}{\partial t} = F(\psi(t)) \rightarrow \frac{\psi(t+\Delta t) - \psi(t)}{\Delta t} \approx F(\psi(t)) \qquad u\frac{\Delta t}{\Delta x} = C < 1$$

for stability

$$\psi(t+\Delta t) = \psi(t) + \Delta t \cdot F(\psi(t)) \qquad \mathcal{O}(\Delta t)$$

(used for SGS-TKE in special cases)

Timestep Methods (I)

- **Euler**

$$\frac{\partial \psi(t)}{\partial t} = F(\psi(t)) \rightarrow \frac{\psi(t+\Delta t) - \psi(t)}{\Delta t} \approx F(\psi(t)) \qquad u\frac{\Delta t}{\Delta x} = C < 1$$

for stability

$$\psi(t+\Delta t) = \psi(t) + \Delta t \cdot F(\psi(t)) \qquad \mathcal{O}(\Delta t)$$

(used for SGS-TKE in special cases)

- **Runge-Kutta, third-order**

$$k_1 = F(\psi(t))$$
$$k_2 = F\left(\psi(t) + \tfrac{1}{3}\Delta t \cdot k_1\right)$$
$$k_3 = F\left(\psi(t) - \tfrac{3}{16}\Delta t \cdot k_1 + \tfrac{15}{16}\Delta t \cdot k_2\right)$$
$$\psi(t+\Delta t) = \psi(t) + \tfrac{1}{30}\Delta t(5k_1 + 9k_2 + 16k_3) \qquad \mathcal{O}(\Delta t^2) \qquad C \leq 0.9$$

Timestep Methods (II)

In the PALM code, the different timestep schemes are treated by one equation using switches:

$$\psi(t+\Delta t) = (1-c_1)\cdot\psi(t-\Delta t)+c_1\cdot\psi(t)+\Delta t\cdot[c_2\cdot F(\psi(t)) + c_3\cdot F(\psi(t-\Delta t))]$$

Timestep Methods (II)

In the PALM code, the different timestep schemes are treated by one equation using switches:

$$\psi(t+\Delta t) = (1-c_1)\cdot\psi(t-\Delta t)+c_1\cdot\psi(t)+\Delta t\cdot[c_2\cdot F(\psi(t)) + c_3\cdot F(\psi(t-\Delta t))]$$

Scheme	$\mathbf{c}_1$	$\mathbf{c}_2$	$\mathbf{c}_3$
Euler	1	1	0
RK (1st step)	1	1/3	0
RK (2nd step)	1	15/16	-25/48
RK (3rd step)	1	8/15	1/15

Timestep Methods (II)

In the PALM code, the different timestep schemes are treated by one equation using switches:

$$\psi(t+\Delta t) = (1-c_1)\cdot\psi(t-\Delta t)+c_1\cdot\psi(t)+\Delta t\cdot[c_2\cdot F(\psi(t)) + c_3\cdot F(\psi(t-\Delta t))]$$

Scheme	$\mathbf{c}_1$	$\mathbf{c}_2$	$\mathbf{c}_3$
Euler	1	1	0
RK (1st step)	1	1/3	0
RK (2nd step)	1	15/16	-25/48
RK (3rd step)	1	8/15	1/15

$$\psi(t-\Delta t) = \psi(t) \qquad \textbf{after each RK substep}$$
$$\psi(t) = \psi(t+\Delta t)$$

Advection Methods (I)

- Piacsek Williams C3 (1970, J. Comput. Phy., 6, 392).

Advection Methods (I)

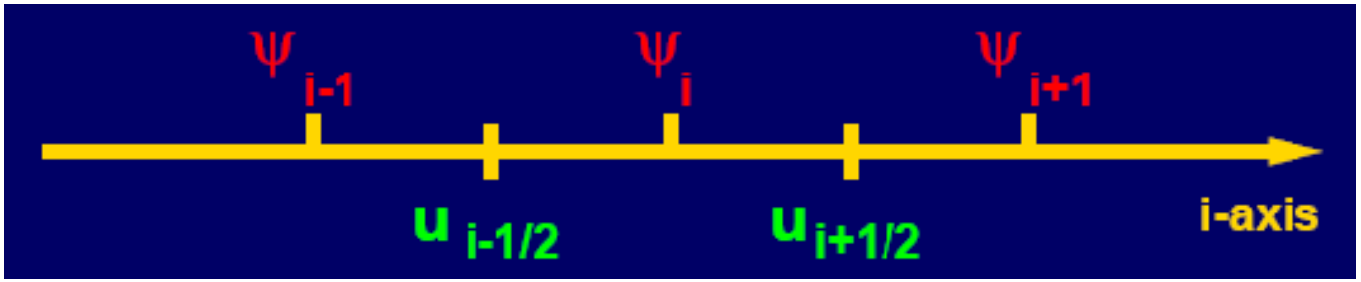
- Piacsek Williams C3 (1970, J. Comput. Phy., 6, 392).
- Scheme of 2nd order accuracy.

Advection Methods (I)

- Piacsek Williams C3 (1970, J. Comput. Phy., 6, 392).
- Scheme of 2nd order accuracy.
- Conserves integrals of linear and quadratic quantities.

Advection Methods (I)

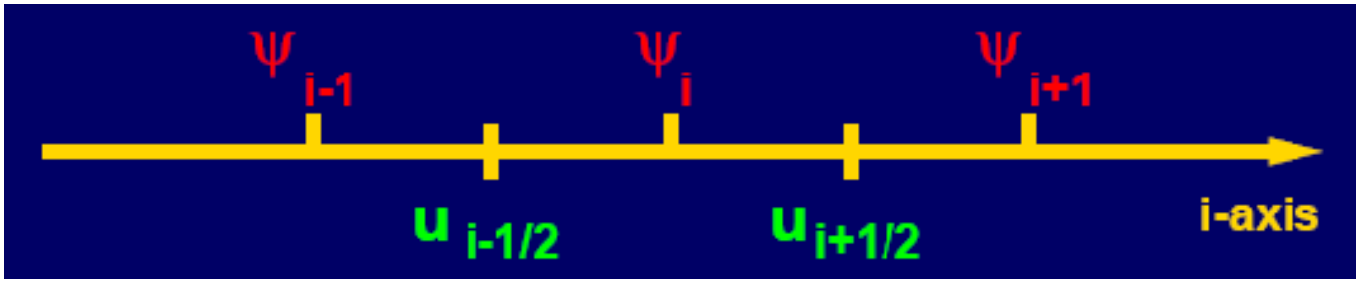
- Piacsek Williams C3 (1970, J. Comput. Phy., 6, 392).
- Scheme of 2nd order accuracy.
- Conserves integrals of linear and quadratic quantities.
- Requires incompressibility $\rightarrow$ flux form of advection term.



$$\left.\frac{\partial(u\psi)}{\partial x}\right|_i = \frac{1}{2\Delta x}\left(u_{i+\frac{1}{2}}\psi_{i+1} - u_{i-\frac{1}{2}}\psi_{i-1}\right)$$

Advection Methods (I)

- Piacsek Williams C3 (1970, J. Comput. Phy., 6, 392).
- Scheme of 2nd order accuracy.
- Conserves integrals of linear and quadratic quantities.
- Requires incompressibility $\rightarrow$ flux form of advection term.



$$\left.\frac{\partial(u\psi)}{\partial x}\right|_i = \frac{1}{2\Delta x}\left(u_{i+\frac{1}{2}}\psi_{i+1} - u_{i-\frac{1}{2}}\psi_{i-1}\right)$$

- In case of momentum advection (e.g. $\psi = u$), u_{i-1} and u_{i+1} have to be obtained by linear interpolation.
- May cause $2\Delta x$ wiggles in case of sharp gradients.

Advection Methods (II)

- Bott-Chlond
  - Chlond (1994)

Advection Methods (II)

- Bott-Chlond
  - Chlond (1994)
  - Monotone, positive definit. Can only be used for scalars

Advection Methods (II)

- Bott-Chlond
  - Chlond (1994)
  - Monotone, positive definit. Can only be used for scalars
  - Conserves sharp gradients

Advection Methods (II)

- Bott-Chlond
  - Chlond (1994)
  - Monotone, positive definit. Can only be used for scalars
  - Conserves sharp gradients
  - Numerically expensive

Advection Methods (II)

- Bott-Chlond
  - Chlond (1994)
  - Monotone, positive definit. Can only be used for scalars
  - Conserves sharp gradients
  - Numerically expensive
  - Not optimized for use on cache-based machines.

Advection Methods (II)

- Bott-Chlond
  - Chlond (1994)
  - Monotone, positive definit. Can only be used for scalars
  - Conserves sharp gradients
  - Numerically expensive
  - Not optimized for use on cache-based machines.

- Default: Wicker and Skamarock scheme (5th order)
  - Much better accuracy than Piacsek Williams

Advection Methods (II)

- Bott-Chlond
  - Chlond (1994)
  - Monotone, positive definit. Can only be used for scalars
  - Conserves sharp gradients
  - Numerically expensive
  - Not optimized for use on cache-based machines.

- Default: Wicker and Skamarock scheme (5th order)
  - Much better accuracy than Piacsek Williams
  - Much simpler algorithm than Bott-Chlond

Advection Methods (II)

- Bott-Chlond
  - Chlond (1994)
  - Monotone, positive definit. Can only be used for scalars
  - Conserves sharp gradients
  - Numerically expensive
  - Not optimized for use on cache-based machines.

- Default: Wicker and Skamarock scheme (5th order)
  - Much better accuracy than Piacsek Williams
  - Much simpler algorithm than Bott-Chlond
  - Requires additional ghost layers

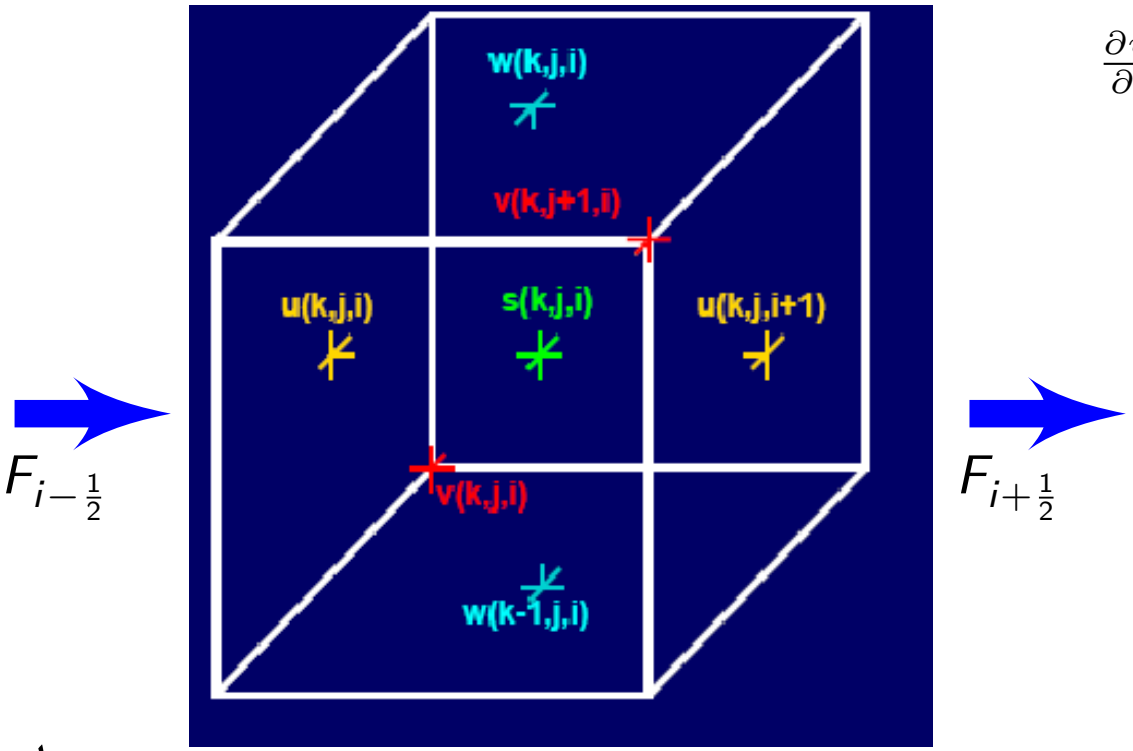
Advection Methods (II)

- Bott-Chlond
  - Chlond (1994)
  - Monotone, positive definit. Can only be used for scalars
  - Conserves sharp gradients
  - Numerically expensive
  - Not optimized for use on cache-based machines.

- Default: Wicker and Skamarock scheme (5th order)
  - Much better accuracy than Piacsek Williams
  - Much simpler algorithm than Bott-Chlond
  - Requires additional ghost layers
  - Adds additional numerical dissipation

Advection Methods – Wicker/Skamarock (I)

- Wicker and Skamarock (2002, Mon. Wea. Rev. 130, 2088 – 2097).
- Based on flux form of advection term
- Difference of fluxes at the edge of the grid cell is used to discretise advection term



$$\frac{\partial \psi}{\partial t} = -\nabla(u_i \psi) \approx -\frac{F_{i+\frac{1}{2}} - F_{i-\frac{1}{2}}}{\Delta x}$$

Advection Methods – Wicker/Skamarock (II)

Finite difference approximation of 6^{th} order

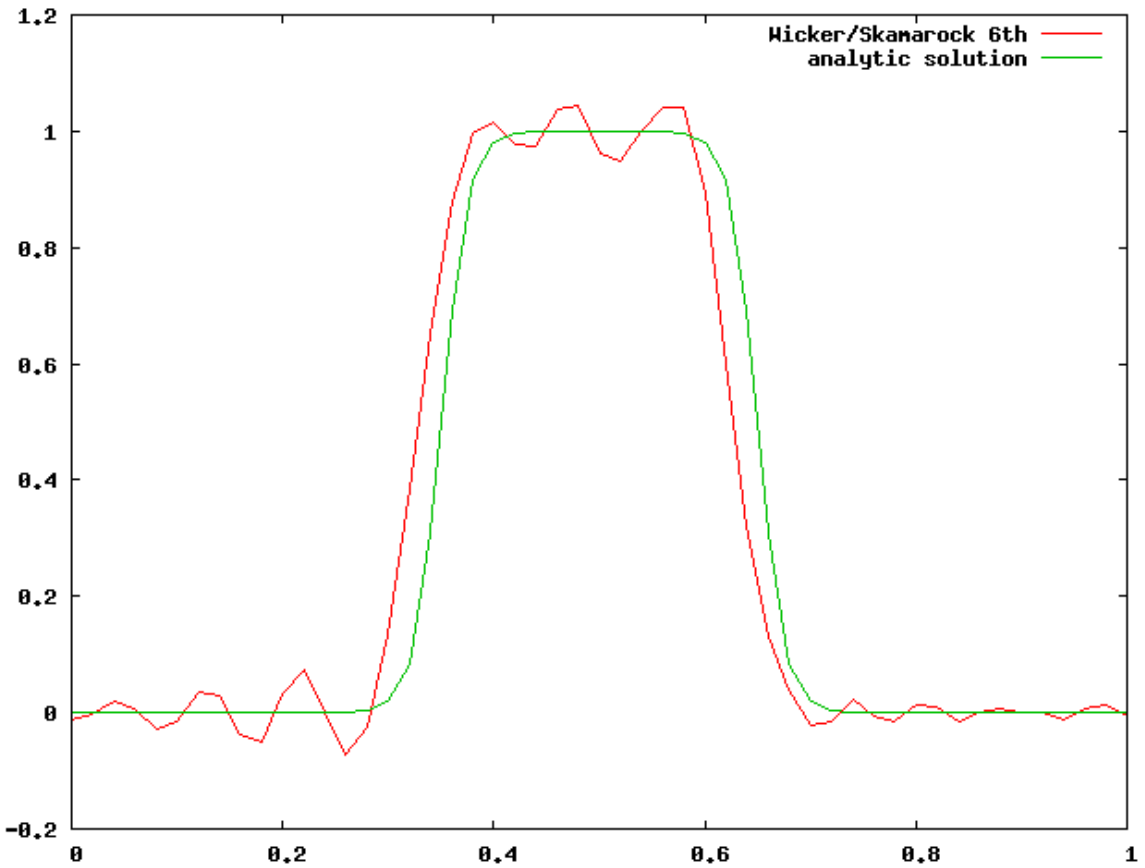
$$F^{6\text{th}}_{i-\frac{1}{2}} = \frac{1}{60} u_{i-\frac{1}{2}} \left(37(\Psi_i + \Psi_{i-1}) - 8(\Psi_{i+1} + \Psi_{i-2}) + (\Psi_{i+2} + \Psi_{i-3})\right)$$

Artificially added numerical dissipation term

$$-\frac{1}{60} \left| u_{i-\frac{1}{2}} \right| \left(10(\Psi_i - \Psi_{i-1}) - 5(\Psi_{i+1} - \Psi_{i-2}) + (\Psi_{i+2} - \Psi_{i-3})\right)$$

Advection Methods – Wicker/Skamarock (III)

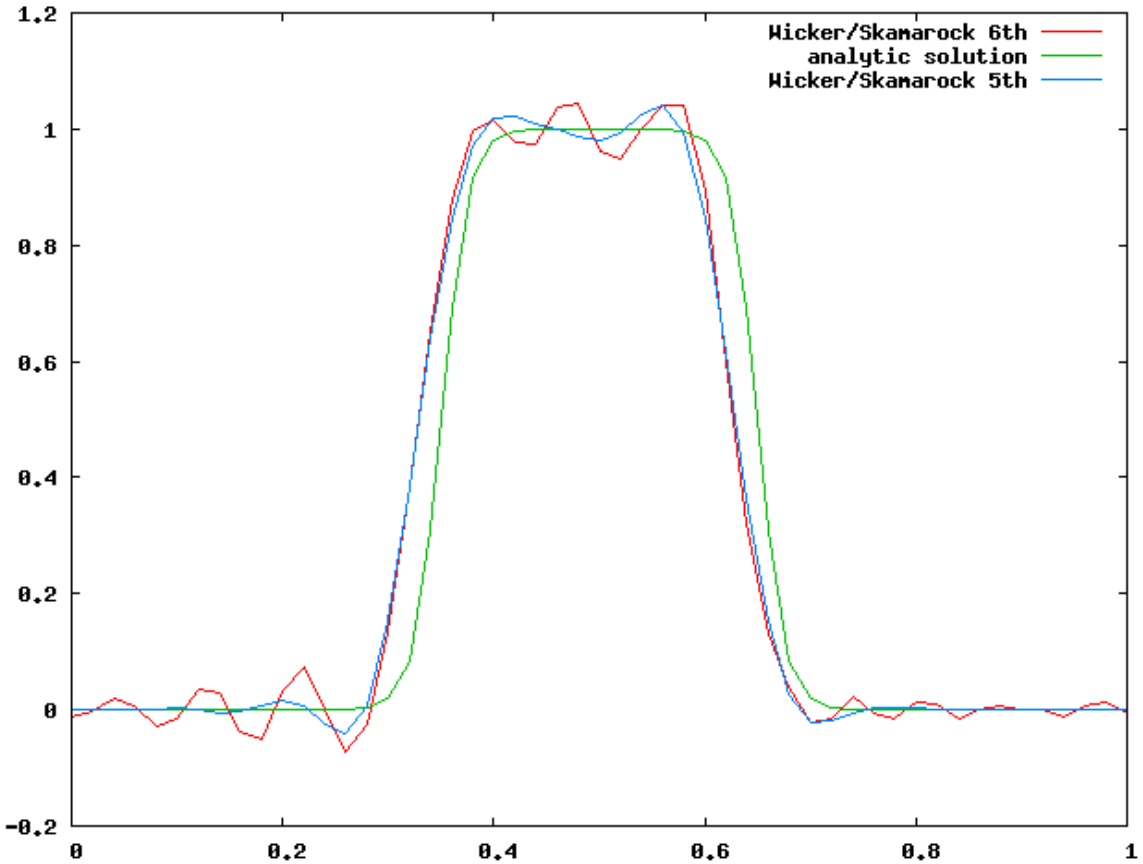
$$F^{6th}_{i-\frac{1}{2}} = \frac{1}{60} u_{i-\frac{1}{2}} \left(37(\Psi_i + \Psi_{i-1}) - 8(\Psi_{i+1} + \Psi_{i-2}) + (\Psi_{i+2} + \Psi_{i-3})\right)$$



Centered Finite Differences produces numerical oscillations ("wiggles") near sharp gradients.

Advection Methods – Wicker/Skamarock (IV)

$$F^{5th}_{i-\frac{1}{2}} = F^{6th}_{i-\frac{1}{2}} - \frac{1}{60}\left|u_{i-\frac{1}{2}}\right|\left(10(\Psi_i - \Psi_{i-1}) - 5(\Psi_{i+1} - \Psi_{i-2}) + (\Psi_{i+2} - \Psi_{i-3})\right)$$



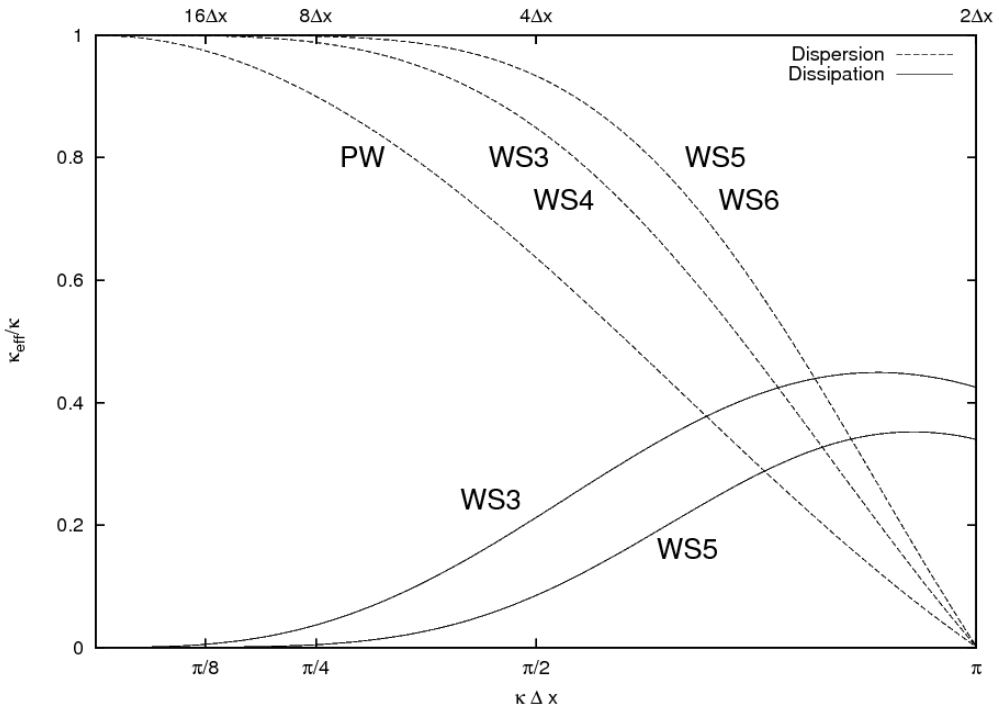
Advantage

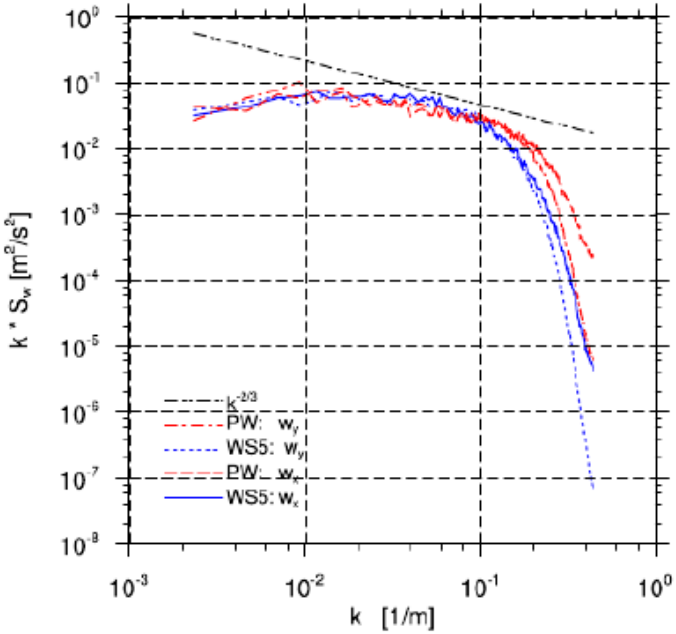
Numerical Dissipation damps small scale oscillations.

Disadvantage

In a turbulent flow numerical dissipation removes energy from small scales.

Advection Methods – Wicker/Skamarock (V)





- Better resolution of larger scales ($> 8\,\Delta x$) and hence less numerical energy transfer from larger to smaller scales compared to lower order schemes.
- Less spectral energy at smaller scales.

Pressure Solver (I)

- Governing equations of PALM require incompressibility

Pressure Solver (I)

- Governing equations of PALM require incompressibility
- Incompressibility is reached by a predictor-corrector method
  1. Momentum equations are solved without the pressure term giving a provisional velocity field which is not free of divergence.

$$\overline{u}_{i_{\mathrm{prov}}}^{t+\Delta t} = \overline{u}_i^t + \Delta t \left(-\frac{\partial}{\partial x_k} \overline{u}_k^t \overline{u}_i^t - (\varepsilon_{ijk} f_j \overline{u}_k^t - \varepsilon_{i3k} f_3 u_{g_k}) + g \frac{\overline{\theta^*}^t}{\theta_0} \delta_{i3} - \frac{\partial}{\partial x_k} \overline{u_k' u_i'}^t \right)$$

Pressure Solver (I)

- Governing equations of PALM require incompressibility
- Incompressibility is reached by a predictor-corrector method

1. Momentum equations are solved without the pressure term giving a provisional velocity field which is not free of divergence.

$$\overline{u}_{i_{\mathrm{prov}}}^{t+\Delta t} = \overline{u}_i^t + \Delta t \left(-\frac{\partial}{\partial x_k} \overline{u}_k^t \overline{u}_i^t - (\varepsilon_{ijk} f_j \overline{u}_k^t - \varepsilon_{i3k} f_3 u_{g_k}) + g \frac{\overline{\theta^*}^t}{\theta_0} \delta_{i3} - \frac{\partial}{\partial x_k} \overline{u_k' u_i'}^t \right)$$

2. Assign all remaining divergences to the (perturbation) pressure p^* so that the new corrected velocity field is the sum of the provisional, divergent field and the perturbation pressure term.

$$\overline{u}_i^{t+\Delta t} = \overline{u}_{i_{\mathrm{prov}}}^{t+\Delta t} + \Delta t \left(-\frac{1}{\rho_0} \frac{\partial \overline{p^*}^t}{\partial x_i} \right)$$

Pressure Solver (I)

- Governing equations of PALM require incompressibility
- Incompressibility is reached by a predictor-corrector method

1. Momentum equations are solved without the pressure term giving a provisional velocity field which is not free of divergence.

$$\overline{u}_{i_{\mathrm{prov}}}^{t+\Delta t} = \overline{u}_i^t + \Delta t \left(-\frac{\partial}{\partial x_k} \overline{u}_k^t \overline{u}_i^t - (\varepsilon_{ijk} f_j \overline{u}_k^t - \varepsilon_{i3k} f_3 u_{g_k}) + g \frac{\overline{\theta^*}^t}{\theta_0} \delta_{i3} - \frac{\partial}{\partial x_k} \overline{u_k' u_i'}^t \right)$$

2. Assign all remaining divergences to the (perturbation) pressure p^* so that the new corrected velocity field is the sum of the provisional, divergent field and the perturbation pressure term.

$$\overline{u}_i^{t+\Delta t} = \overline{u}_{i_{\mathrm{prov}}}^{t+\Delta t} + \Delta t \left(-\frac{1}{\rho_0} \frac{\partial \overline{p^*}^t}{\partial x_i} \right)$$

3. The divergence operator is applied to this equation. Demanding a corrected velocity field free of divergence, this leads to a Poisson equation for the perturbation pressure.

$$\frac{\partial^2 \overline{p^*}^t}{\partial x_i^2} = \frac{\rho_0}{\Delta t} \frac{\partial \overline{u}_{i_{\mathrm{prov}}}^{t+\Delta t}}{\partial x_i}$$

Pressure Solver (I)

- Governing equations of PALM require incompressibility
- Incompressibility is reached by a predictor-corrector method

1. Momentum equations are solved without the pressure term giving a provisional velocity field which is not free of divergence.

$$\overline{u}_{i_{\mathrm{prov}}}^{t+\Delta t} = \overline{u}_i^t + \Delta t \left(-\frac{\partial}{\partial x_k} \overline{u}_k^t \overline{u}_i^t - (\varepsilon_{ijk} f_j \overline{u}_k^t - \varepsilon_{i3k} f_3 u_{g_k}) + g \frac{\overline{\theta^*}^t}{\theta_0} \delta_{i3} - \frac{\partial}{\partial x_k} \overline{u_k' u_i'}^t \right)$$

2. Assign all remaining divergences to the (perturbation) pressure p^* so that the new corrected velocity field is the sum of the provisional, divergent field and the perturbation pressure term.

$$\overline{u}_i^{t+\Delta t} = \overline{u}_{i_{\mathrm{prov}}}^{t+\Delta t} + \Delta t \left(-\frac{1}{\rho_0} \frac{\partial \overline{p^*}^t}{\partial x_i} \right)$$

3. The divergence operator is applied to this equation. Demanding a corrected velocity field free of divergence, this leads to a Poisson equation for the perturbation pressure.

$$\frac{\partial^2 \overline{p^*}^t}{\partial x_i^2} = \frac{\rho_0}{\Delta t} \frac{\partial \overline{u}_{i_{\mathrm{prov}}}^{t+\Delta t}}{\partial x_i}$$

4. After solving the Poisson equation, the final velocity field is calculated as given in step 2.

Pressure Solver (II)

- FFT-solver
  1. Discretization of the Poisson-equation by central differences

Pressure Solver (II)

- FFT-solver
  1. Discretization of the Poisson-equation by central differences
  2. 2D discrete FFT in both horizontal directions

Pressure Solver (II)

- FFT-solver
  1. Discretization of the Poisson-equation by central differences
  2. 2D discrete FFT in both horizontal directions
  3. Solving the resulting tridiagonal set of linear equations

Pressure Solver (II)

- FFT-solver
  1. Discretization of the Poisson-equation by central differences
  2. 2D discrete FFT in both horizontal directions
  3. Solving the resulting tridiagonal set of linear equations
  4. Inverse 2D discrete FFT in both horizontal directions leading to the perturbation pressure

Pressure Solver (II)

- FFT-solver
  1. Discretization of the Poisson-equation by central differences
  2. 2D discrete FFT in both horizontal directions
  3. Solving the resulting tridiagonal set of linear equations
  4. Inverse 2D discrete FFT in both horizontal directions leading to the perturbation pressure

  - Very fast and accurate, $\mathcal{O}(n \log n)$, n: number of gridpoints

Pressure Solver (II)

- FFT-solver
  1. Discretization of the Poisson-equation by central differences
  2. 2D discrete FFT in both horizontal directions
  3. Solving the resulting tridiagonal set of linear equations
  4. Inverse 2D discrete FFT in both horizontal directions leading to the perturbation pressure

  - Very fast and accurate, $\mathcal{O}(n \log n)$, n: number of gridpoints
  - CPU requirement $< 50\%$ of total CPU time

Pressure Solver (II)

- FFT-solver
  1. Discretization of the Poisson-equation by central differences
  2. 2D discrete FFT in both horizontal directions
  3. Solving the resulting tridiagonal set of linear equations
  4. Inverse 2D discrete FFT in both horizontal directions leading to the perturbation pressure

  - Very fast and accurate, $\mathcal{O}(n \log n)$, n: number of gridpoints
  - CPU requirement $< 50\%$ of total CPU time
  - Due to non-locality of the FFT, transpositions are required on parallel computers

Pressure Solver (II)

- FFT-solver
  1. Discretization of the Poisson-equation by central differences
  2. 2D discrete FFT in both horizontal directions
  3. Solving the resulting tridiagonal set of linear equations
  4. Inverse 2D discrete FFT in both horizontal directions leading to the perturbation pressure

  - Very fast and accurate, $\mathcal{O}(n \log n)$, n: number of gridpoints
  - CPU requirement $< 50\%$ of total CPU time
  - Due to non-locality of the FFT, transpositions are required on parallel computers
  - Requires periodic boundary conditions and uniform grids along x and y

Pressure Solver (III)

- Multigrid-method
  - Iterative solver
    basic idea: Poisson equation is transformed to a fixed point problem:
    $\vec{p}^{k+1} = T \cdot \vec{p}^{k} + \vec{c}^{k}$

Pressure Solver (III)

- Multigrid-method
  - Iterative solver
    basic idea: Poisson equation is transformed to a fixed point problem:
    $\vec{p}^{k+1} = T \cdot \vec{p}^{k} + \vec{c}^{k}$

    starting from a first guess, the solution will be improved by repeated execution of the fixed point problem:

$$\begin{aligned} \vec{p}^{1} &= T \cdot \vec{p}^{0} + \vec{c}^{0} \\ \vec{p}^{2} &= T \cdot \vec{p}^{1} + \vec{c}^{1} \\ &\vdots \\ \vec{p}^{k} &= T \cdot \vec{p}^{k-1} + \vec{c}^{k-1} \\ \vec{p}^{k+1} &= T \cdot \vec{p}^{k} + \vec{c}^{k} \end{aligned}$$

Pressure Solver (III)

- Multigrid-method
  - Iterative solver
    basic idea: Poisson equation is transformed to a fixed point problem:
    $\vec{p}^{k+1} = T \cdot \vec{p}^{k} + \vec{c}^{k}$

    starting from a first guess, the solution will be improved by repeated execution of the fixed point problem:

$$\begin{aligned} \vec{p}^{1} &= T \cdot \vec{p}^{0} + \vec{c}^{0} \\ \vec{p}^{2} &= T \cdot \vec{p}^{1} + \vec{c}^{1} \\ &\vdots \\ \vec{p}^{k} &= T \cdot \vec{p}^{k-1} + \vec{c}^{k-1} \\ \vec{p}^{k+1} &= T \cdot \vec{p}^{k} + \vec{c}^{k} \end{aligned}$$

    Depending on the structure of the matrix T and vector c different iterative solvers can be defined, e.g.: Jacobi-scheme (here on 2D-uniform grid):
    $p_{i,j}^{k+1} = \frac{1}{4} \cdot \left(p_{i-1,j}^{k} + p_{i+1,j}^{k} + p_{i,j-1}^{k} + p_{i,j+1}^{k} - \Delta x^2 f(i,j,k) \right)$

Pressure Solver (III)

- Multigrid-method
  - Iterative solver
    basic idea: Poisson equation is transformed to a fixed point problem:
    $\vec{p}^{k+1} = T \cdot \vec{p}^{k} + \vec{c}^{k}$

    starting from a first guess, the solution will be improved by repeated execution of the fixed point problem:

$$\begin{aligned} \vec{p}^{1} &= T \cdot \vec{p}^{0} + \vec{c}^{0} \\ \vec{p}^{2} &= T \cdot \vec{p}^{1} + \vec{c}^{1} \\ &\vdots \\ \vec{p}^{k} &= T \cdot \vec{p}^{k-1} + \vec{c}^{k-1} \\ \vec{p}^{k+1} &= T \cdot \vec{p}^{k} + \vec{c}^{k} \end{aligned}$$

    Depending on the structure of the matrix T and vector c different iterative solvers can be defined, e.g.: Jacobi-scheme (here on 2D-uniform grid):
    $p_{i,j}^{k+1} = \frac{1}{4} \cdot \left(p_{i-1,j}^{k} + p_{i+1,j}^{k} + p_{i,j-1}^{k} + p_{i,j+1}^{k} - \Delta x^2 f(i,j,k) \right)$
  - With each iteration step k the improved solution converges towards the exact solution.

Pressure Solver (III)

- Multigrid-method
  - Iterative solver
    basic idea: Poisson equation is transformed to a fixed point problem:
    $\vec{p}^{k+1} = T \cdot \vec{p}^{k} + \vec{c}^{k}$

    starting from a first guess, the solution will be improved by repeated execution of the fixed point problem:

$$\begin{aligned} \vec{p}^{1} &= T \cdot \vec{p}^{0} + \vec{c}^{0} \\ \vec{p}^{2} &= T \cdot \vec{p}^{1} + \vec{c}^{1} \\ &\vdots \\ \vec{p}^{k} &= T \cdot \vec{p}^{k-1} + \vec{c}^{k-1} \\ \vec{p}^{k+1} &= T \cdot \vec{p}^{k} + \vec{c}^{k} \end{aligned}$$

    Depending on the structure of the matrix T and vector c different iterative solvers can be defined, e.g.: Jacobi-scheme (here on 2D-uniform grid):
    $p_{i,j}^{k+1} = \frac{1}{4} \cdot \left(p_{i-1,j}^{k} + p_{i+1,j}^{k} + p_{i,j-1}^{k} + p_{i,j+1}^{k} - \Delta x^2 f(i,j,k) \right)$
  - With each iteration step k the improved solution converges towards the exact solution.
  - Iterative schemes are 'local schemes' $\rightarrow$ information is needed only from neighboring grid-points.

Pressure Solver (III)

- Multigrid-method
  - Iterative solver
    basic idea: Poisson equation is transformed to a fixed point problem:
    $\vec{p}^{k+1} = T \cdot \vec{p}^{k} + \vec{c}^{k}$

    starting from a first guess, the solution will be improved by repeated execution of the fixed point problem:

$$\begin{aligned} \vec{p}^{1} &= T \cdot \vec{p}^{0} + \vec{c}^{0} \\ \vec{p}^{2} &= T \cdot \vec{p}^{1} + \vec{c}^{1} \\ &\vdots \\ \vec{p}^{k} &= T \cdot \vec{p}^{k-1} + \vec{c}^{k-1} \\ \vec{p}^{k+1} &= T \cdot \vec{p}^{k} + \vec{c}^{k} \end{aligned}$$

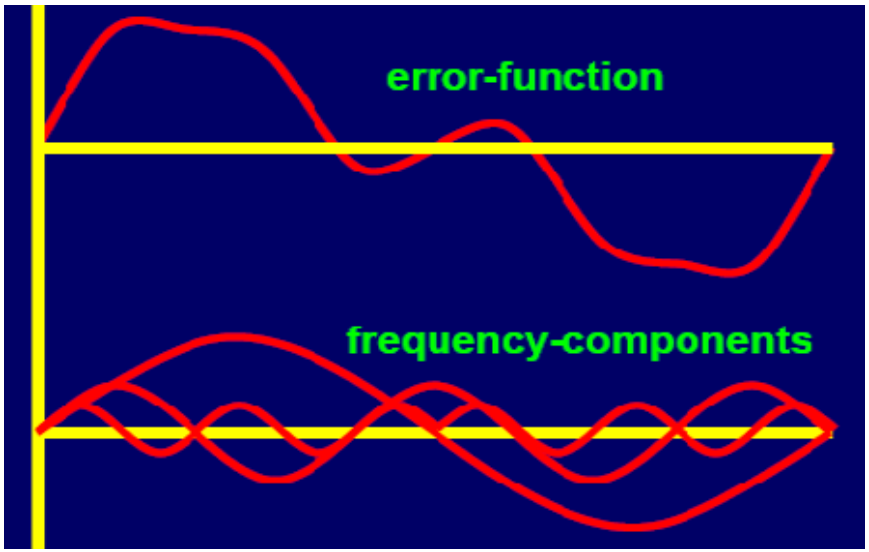
    Depending on the structure of the matrix T and vector c different iterative solvers can be defined, e.g.: Jacobi-scheme (here on 2D-uniform grid):
    $p_{i,j}^{k+1} = \frac{1}{4} \cdot \left(p_{i-1,j}^{k} + p_{i+1,j}^{k} + p_{i,j-1}^{k} + p_{i,j+1}^{k} - \Delta x^2 f(i,j,k) \right)$
  - With each iteration step k the improved solution converges towards the exact solution.
  - Iterative schemes are 'local schemes' $\rightarrow$ information is needed only from neighboring grid-points.
  - Very low convergence: $\mathcal{O}(n^2)$.

Pressure Solver (IV)

- Multigrid-method
  - Due to their locality, iterative solvers show a frequency-dependent reduction of the residual: low frequencies are reduced slower than high frequencies.

Pressure Solver (IV)

- Multigrid-method
  - Due to their locality, iterative solvers show a frequency-dependent reduction of the residual: low frequencies are reduced slower than high frequencies.
  - The main idea of the multigrid method is to reduce errors of different frequencies on grids with different grid spacing:
    - errors of high frequency are reduced on fine grids
    - errors of low frequency are reduced on coarse grids.

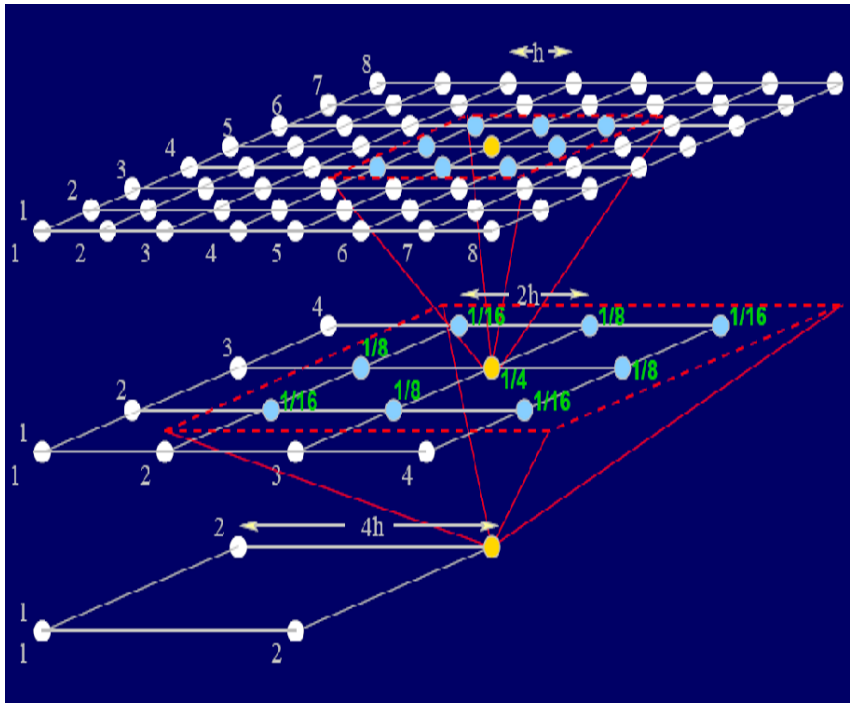


Pressure Solver (V)

- Multigrid-method
  - On each grid-level an approximate solution of the fixed point equation is obtained performing a few iterations.

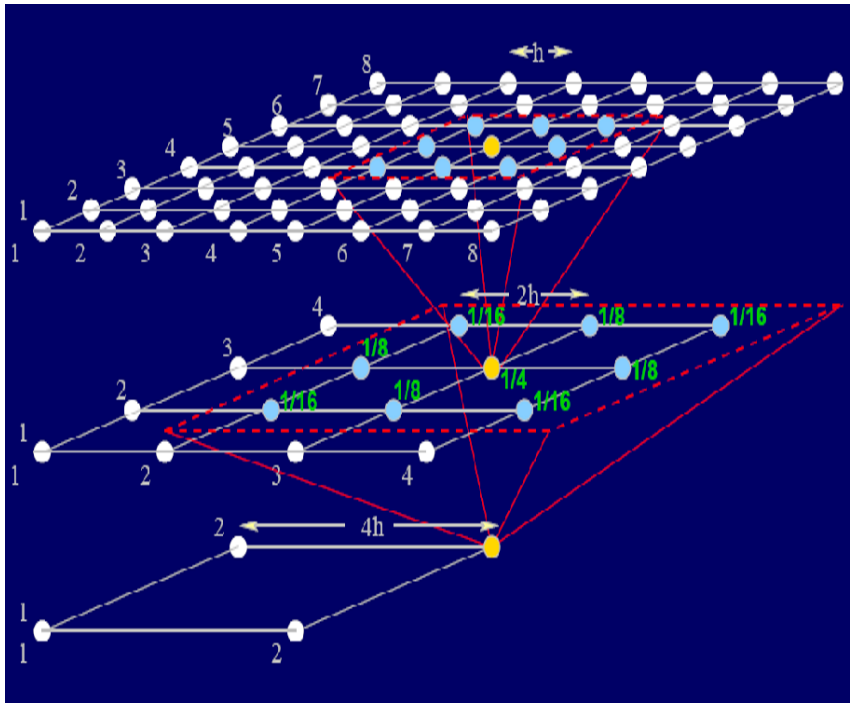
Pressure Solver (V)

- Multigrid-method
  - On each grid-level an approximate solution of the fixed point equation is obtained performing a few iterations.
  - The solution is transmitted to the next coarser grid-level where it is used as the first guess to solve the fixed point problem.



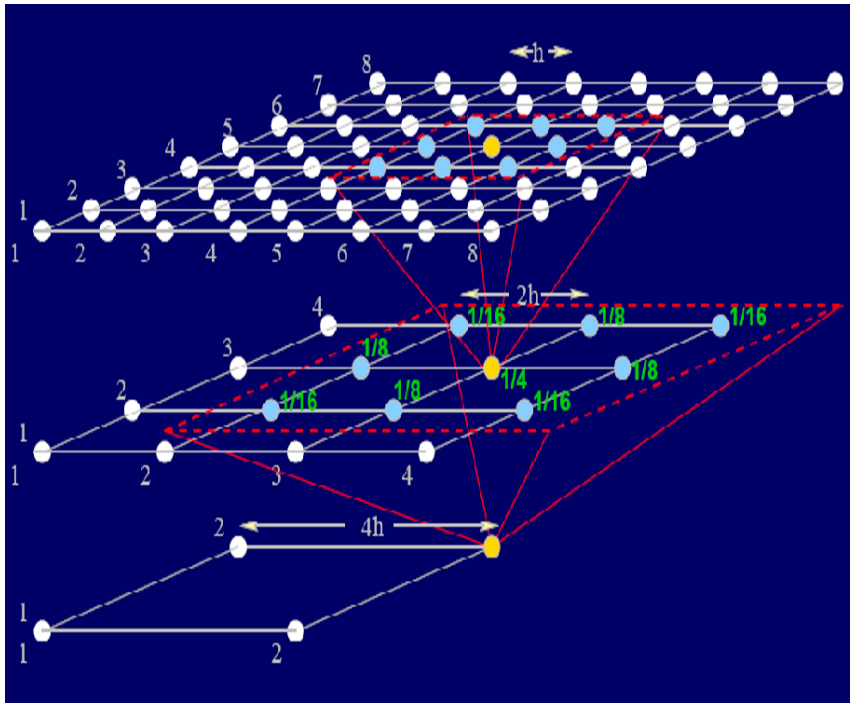
Pressure Solver (V)

- Multigrid-method
  - On each grid-level an approximate solution of the fixed point equation is obtained performing a few iterations.
  - The solution is transmitted to the next coarser grid-level where it is used as the first guess to solve the fixed point problem.
  - This procedure is performed up to the coarsest grid-level containing two grid-points in each direction.



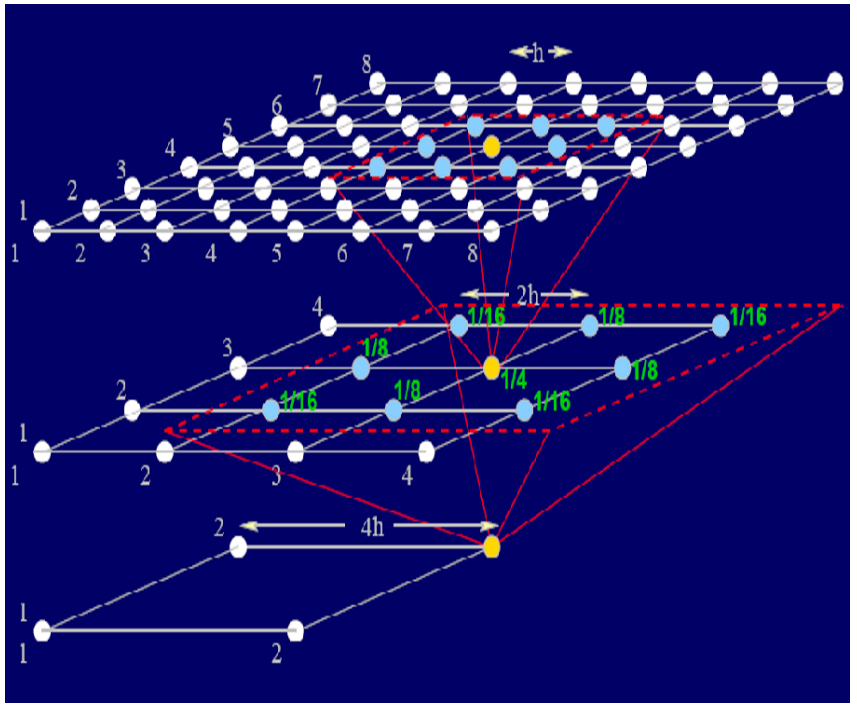
Pressure Solver (V)

- Multigrid-method
  - On each grid-level an approximate solution of the fixed point equation is obtained performing a few iterations.
  - The solution is transmitted to the next coarser grid-level where it is used as the first guess to solve the fixed point problem.
  - This procedure is performed up to the coarsest grid-level containing two grid-points in each direction.
  - From the coarsest grid-level the procedure is passed in backward order to get the final solution.



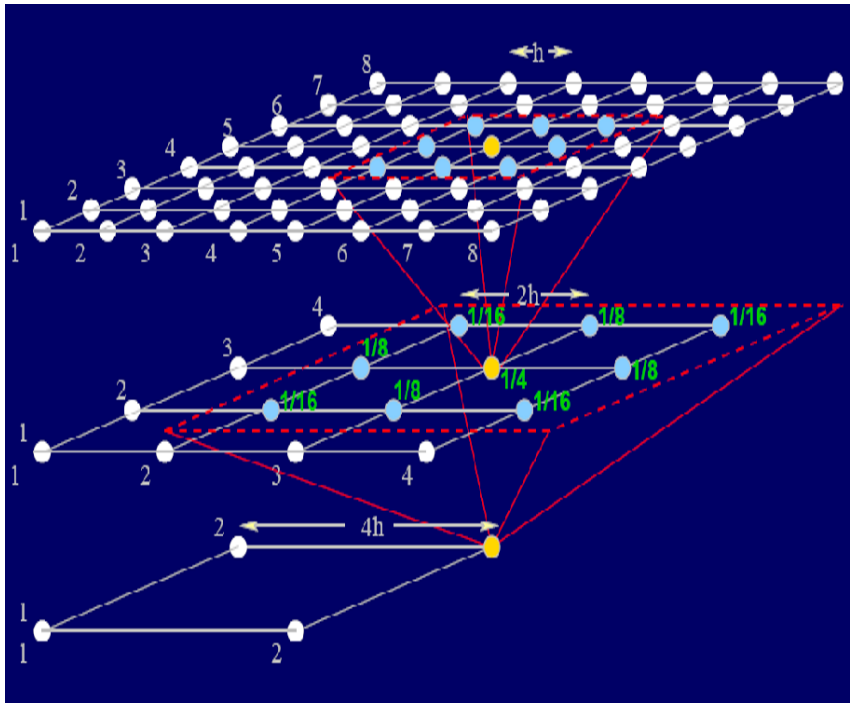
Pressure Solver (V)

- Multigrid-method
  - On each grid-level an approximate solution of the fixed point equation is obtained performing a few iterations.
  - The solution is transmitted to the next coarser grid-level where it is used as the first guess to solve the fixed point problem.
  - This procedure is performed up to the coarsest grid-level containing two grid-points in each direction.
  - From the coarsest grid-level the procedure is passed in backward order to get the final solution.
  - For large grids faster than FFT method.



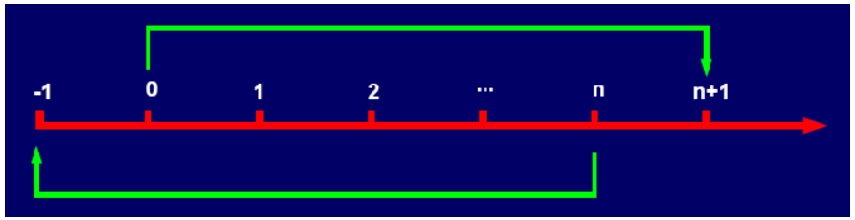
Pressure Solver (V)

- Multigrid-method
  - On each grid-level an approximate solution of the fixed point equation is obtained performing a few iterations.
  - The solution is transmitted to the next coarser grid-level where it is used as the first guess to solve the fixed point problem.
  - This procedure is performed up to the coarsest grid-level containing two grid-points in each direction.
  - From the coarsest grid-level the procedure is passed in backward order to get the final solution.
  - For large grids faster than FFT method.
  - V- and W-cycles are implemented.



Boundary Conditions (I)

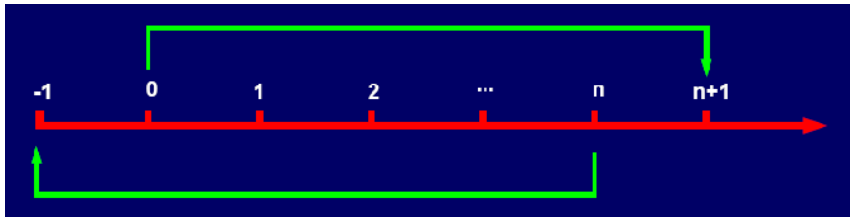
- Lateral (xy) boundary conditions:
  - Cyclic by default, allowing undisturbed evolution / advection of turbulence.



$$\Psi(-1) = \Psi(n)$$
$$\Psi(n+1) = \Psi(0)$$

Boundary Conditions (I)

- Lateral (xy) boundary conditions:
  - Cyclic by default, allowing undisturbed evolution / advection of turbulence.

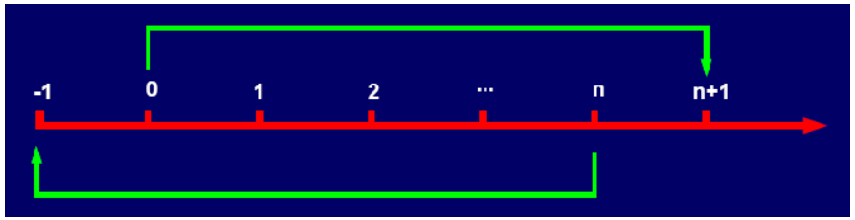


$$\Psi(-1) = \Psi(n)$$
$$\Psi(n+1) = \Psi(0)$$

  - Dirichlet (inflow) and radiation (outflow) conditions are allowed along either x- or y-direction.

Boundary Conditions (I)

- Lateral (xy) boundary conditions:
  - Cyclic by default, allowing undisturbed evolution / advection of turbulence.

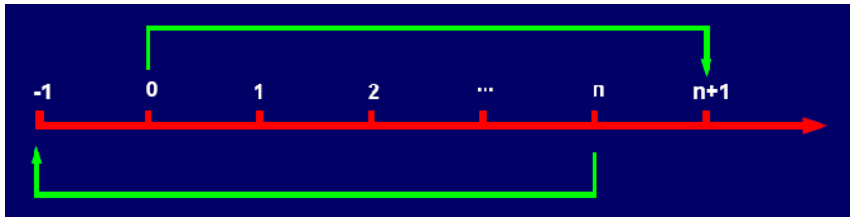


$$\Psi(-1) = \Psi(n)$$
$$\Psi(n+1) = \Psi(0)$$

  - Dirichlet (inflow) and radiation (outflow) conditions are allowed along either x- or y-direction.
  - In case of a Dirichlet condition, the inflow is laminar (by default) and the domain has to be extended to allow for the development of a turbulent state, if neccessary.

Boundary Conditions (I)

- Lateral (xy) boundary conditions:
  - Cyclic by default, allowing undisturbed evolution / advection of turbulence.



$$\Psi(-1) = \Psi(n)$$
$$\Psi(n+1) = \Psi(0)$$

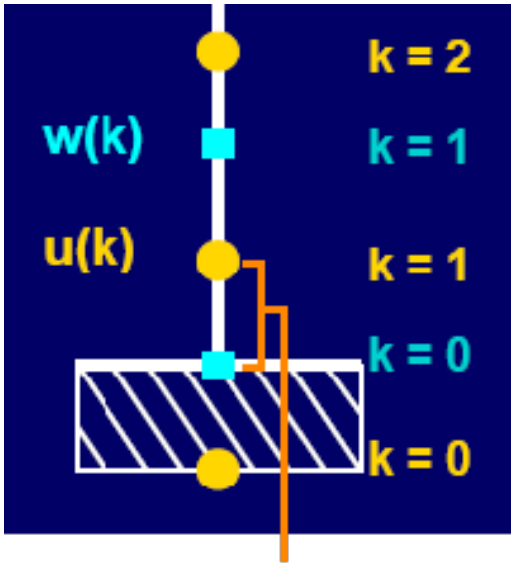
  - Dirichlet (inflow) and radiation (outflow) conditions are allowed along either x- or y-direction.
  - In case of a Dirichlet condition, the inflow is laminar (by default) and the domain has to be extended to allow for the development of a turbulent state, if neccessary.
  - Non-cyclic lateral conditions require the use of the multigrid-method for solving the Poisson-equation.

Boundary Conditions (II)

- Surface boundary condition:
  - Monin-Obukhov-similarity is used by default, i.e. a Prandtl-layer is assumed between the surface and the first grid layer.

$$\frac{\partial \overline{u}}{\partial z} = \frac{u_*}{\kappa z}\Phi_{\mathrm{m}}; \qquad u_* = \sqrt{-\overline{w'u'_0}} = \sqrt{\frac{\tau_0}{\overline{\rho}}}$$

$$\frac{\partial \overline{\theta}}{\partial z} = \frac{\vartheta_*}{\kappa z}\Phi_{\mathrm{h}}; \qquad \vartheta_* = \frac{\overline{w'\theta'_0}}{u_*}$$



Prandtl-layer

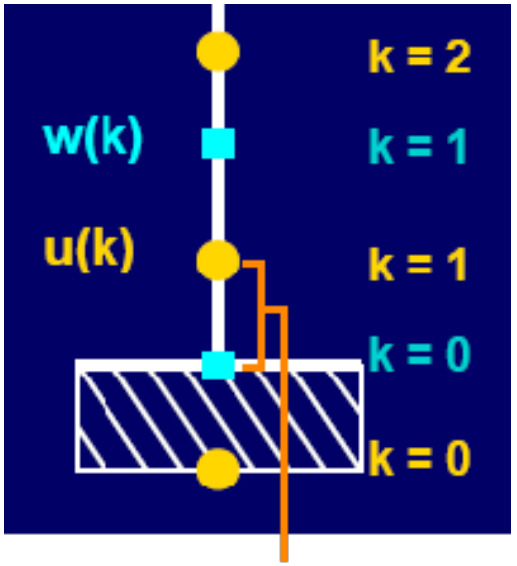
Boundary Conditions (II)

- Surface boundary condition:
  - Monin-Obukhov-similarity is used by default, i.e. a Prandtl-layer is assumed between the surface and the first grid layer.

$$\frac{\partial \overline{u}}{\partial z} = \frac{u_*}{\kappa z}\Phi_\mathrm{m}; \qquad u_* = \sqrt{-\overline{w'u'_0}} = \sqrt{\frac{\tau_0}{\overline{\rho}}}$$

$$\frac{\partial \overline{\theta}}{\partial z} = \frac{\vartheta_*}{\kappa z}\Phi_\mathrm{h}; \qquad \vartheta_* = \frac{\overline{w'\theta'_0}}{u_*}$$

  - Integration between $z = z_0$ (roughness height) and $z = z_\mathrm{p}$ (top of Prandtl-layer, $k = 1$) gives the only unknowns u_* and θ_* which then define the surface momentum and heat flux, used as the real boundary conditions.



Prandtl-layer

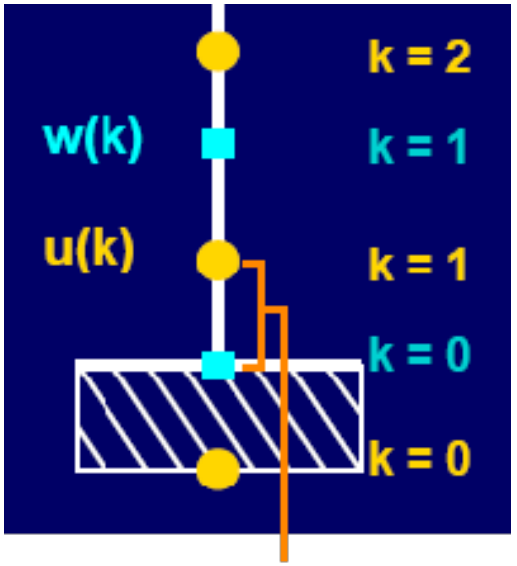
Boundary Conditions (II)

- Surface boundary condition:
  - Monin-Obukhov-similarity is used by default, i.e. a Prandtl-layer is assumed between the surface and the first grid layer.

$$\frac{\partial \overline{u}}{\partial z} = \frac{u_*}{\kappa z}\Phi_\mathrm{m}; \qquad u_* = \sqrt{-\overline{w'u'_0}} = \sqrt{\frac{\tau_0}{\overline{\rho}}}$$

$$\frac{\partial \overline{\theta}}{\partial z} = \frac{\vartheta_*}{\kappa z}\Phi_\mathrm{h}; \qquad \vartheta_* = \frac{\overline{w'\theta'_0}}{u_*}$$

  - Integration between $z = z_0$ (roughness height) and $z = z_\mathrm{p}$ (top of Prandtl-layer, $k = 1$) gives the only unknowns u_* and θ_* which then define the surface momentum and heat flux, used as the real boundary conditions.
  - Φ_m, Φ_h: Dyer-Businger functions



Prandtl-layer

Boundary Conditions (II)

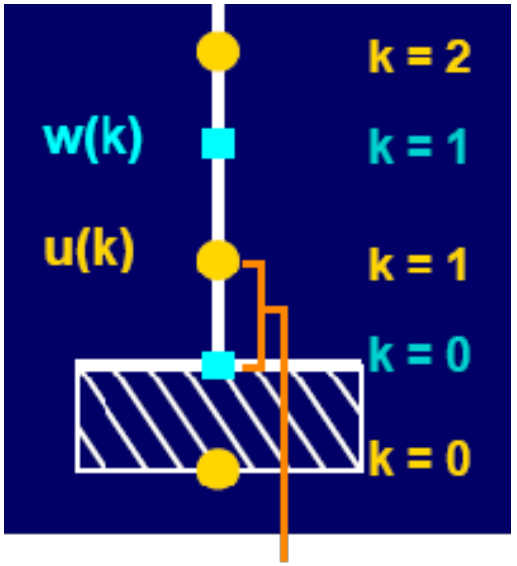
- Surface boundary condition:
  - Monin-Obukhov-similarity is used by default, i.e. a Prandtl-layer is assumed between the surface and the first grid layer.

$$\frac{\partial \overline{u}}{\partial z} = \frac{u_*}{\kappa z}\Phi_\mathrm{m}; \qquad u_* = \sqrt{-\overline{w'u'_0}} = \sqrt{\frac{\tau_0}{\overline{\rho}}}$$

$$\frac{\partial \overline{\theta}}{\partial z} = \frac{\vartheta_*}{\kappa z}\Phi_\mathrm{h}; \qquad \vartheta_* = \frac{\overline{w'\theta'_0}}{u_*}$$

  - Integration between $z = z_0$ (roughness height) and $z = z_\mathrm{p}$ (top of Prandtl-layer, $k = 1$) gives the only unknowns u_* and θ_* which then define the surface momentum and heat flux, used as the real boundary conditions.
  - Φ_m, Φ_h: Dyer-Businger functions

$$\Phi_\mathrm{m} = \begin{cases} 1 + 5\,\mathrm{Rif} & \text{stable} \\ 1 & \text{neutral} \\ (1 - 16\,\mathrm{Rif})^{-1/4} & \text{unstable} \end{cases}$$



Prandtl-layer

Boundary Conditions (II)

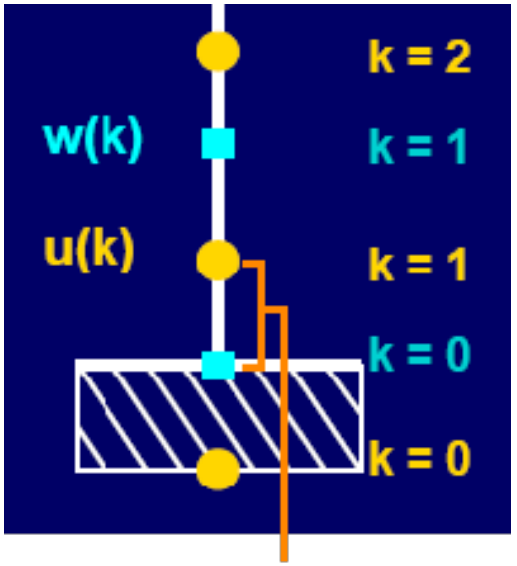
- Surface boundary condition:
  - Monin-Obukhov-similarity is used by default, i.e. a Prandtl-layer is assumed between the surface and the first grid layer.

$$\frac{\partial \overline{u}}{\partial z} = \frac{u_*}{\kappa z}\Phi_\mathrm{m}; \qquad u_* = \sqrt{-\overline{w'u'_0}} = \sqrt{\frac{\tau_0}{\overline{\rho}}}$$

$$\frac{\partial \overline{\theta}}{\partial z} = \frac{\vartheta_*}{\kappa z}\Phi_\mathrm{h}; \qquad \vartheta_* = \frac{\overline{w'\theta'_0}}{u_*}$$

  - Integration between $z = z_0$ (roughness height) and $z = z_\mathrm{p}$ (top of Prandtl-layer, $k = 1$) gives the only unknowns u_* and θ_* which then define the surface momentum and heat flux, used as the real boundary conditions.
  - Φ_m, Φ_h: Dyer-Businger functions

$$\Phi_\mathrm{m} = \begin{cases} 1 + 5\,\mathrm{Rif} & \text{stable} \\ 1 & \text{neutral} \\ (1 - 16\,\mathrm{Rif})^{-1/4} & \text{unstable} \end{cases}$$



Prandtl-layer

$$\mathrm{Rif} = \frac{\frac{g}{\tilde{\theta}}\overline{w'\theta'_0}}{\overline{w'u'}\frac{\partial \overline{u}}{\partial z}}$$

Richardson flux number

Boundary Conditions (III)

- Surface boundary condition:
  - Monin-Obukhov-similarity is only valid for a horizontal surface with homogeneous conditions.

Boundary Conditions (III)

- Surface boundary condition:
  - Monin-Obukhov-similarity is only valid for a horizontal surface with homogeneous conditions.
  - The surface temperature has to be prescribed. Alternatively, the surface heat flux can be prescribed.

Boundary Conditions (III)

- Surface boundary condition:
  - Monin-Obukhov-similarity is only valid for a horizontal surface with homogeneous conditions.
  - The surface temperature has to be prescribed. Alternatively, the surface heat flux can be prescribed.
  - Instead of MO-similarity, no-slip conditions or free-slip conditions can be used

$$u(z=0)=0, \quad v(z=0)=0 \qquad \frac{\partial u}{\partial z}=0, \quad \frac{\partial v}{\partial z}=0$$

realized by

$$u(k=0)=-u(k=1) \qquad u(k=0)=u(k=1)$$
$$v(k=0)=-v(k=1) \qquad v(k=0)=v(k=1)$$

Boundary Conditions (III)

- Surface boundary condition:
  - Monin-Obukhov-similarity is only valid for a horizontal surface with homogeneous conditions.
  - The surface temperature has to be prescribed. Alternatively, the surface heat flux can be prescribed.
  - Instead of MO-similarity, no-slip conditions or free-slip conditions can be used

$$u(z=0)=0, \quad v(z=0)=0 \qquad \frac{\partial u}{\partial z}=0, \quad \frac{\partial v}{\partial z}=0$$

realized by

$$u(k=0)=-u(k=1) \qquad u(k=0)=u(k=1)$$
$$v(k=0)=-v(k=1) \qquad v(k=0)=v(k=1)$$

  - Pressure boundary condition: $\frac{\partial p}{\partial z}=0$ in order to guarantee $w(z=0)=0$

Boundary Conditions (III)

- Surface boundary condition:
  - Monin-Obukhov-similarity is only valid for a horizontal surface with homogeneous conditions.
  - The surface temperature has to be prescribed. Alternatively, the surface heat flux can be prescribed.
  - Instead of MO-similarity, no-slip conditions or free-slip conditions can be used

$$u(z=0)=0, \quad v(z=0)=0 \qquad \frac{\partial u}{\partial z}=0, \quad \frac{\partial v}{\partial z}=0$$

realized by

$$u(k=0)=-u(k=1) \qquad u(k=0)=u(k=1)$$
$$v(k=0)=-v(k=1) \qquad v(k=0)=v(k=1)$$

  - Pressure boundary condition: $\frac{\partial p}{\partial z}=0$ in order to guarantee $w(z=0)=0$
  - SGS-TKE condition $\frac{\partial e}{\partial z}=0$

Boundary Conditions (IV)

- Boundary conditions at the top (default)
  - Dirichlet conditions for velocities: $u = u_{\mathrm{g}}, \quad v = v_{\mathrm{g}}, \quad w = 0$

Boundary Conditions (IV)

- Boundary conditions at the top (default)
  - Dirichlet conditions for velocities: $u = u_{\mathrm{g}}, \quad v = v_{\mathrm{g}}, \quad w = 0$
  - Neumann conditions (temporal constant gradients) for scalars:

$$\frac{\partial \theta}{\partial z} = \left.\frac{\partial \theta}{\partial z}\right|_{t=0}$$

Boundary Conditions (IV)

- Boundary conditions at the top (default)
  - Dirichlet conditions for velocities: $u = u_{\mathrm{g}}, \quad v = v_{\mathrm{g}}, \quad w = 0$
  - Neumann conditions (temporal constant gradients) for scalars:

$$\frac{\partial \theta}{\partial z} = \left. \frac{\partial \theta}{\partial z} \right|_{t=0}$$

  - Pressure: Dirichlet $p = 0$ (Neumann $\frac{\partial p}{\partial z} = 0$ is better)

Boundary Conditions (IV)

- Boundary conditions at the top (default)
  - Dirichlet conditions for velocities: $u = u_{\mathrm{g}}, \quad v = v_{\mathrm{g}}, \quad w = 0$
  - Neumann conditions (temporal constant gradients) for scalars:

$$\frac{\partial \theta}{\partial z} = \left. \frac{\partial \theta}{\partial z} \right|_{t=0}$$

  - Pressure: Dirichlet $p = 0$ (Neumann $\frac{\partial p}{\partial z} = 0$ is better)
  - SGS-TKE: Neumann $\frac{\partial e}{\partial z} = 0$

Boundary Conditions (IV)

- Boundary conditions at the top (default)
  - Dirichlet conditions for velocities: $u = u_{\mathrm{g}}, \quad v = v_{\mathrm{g}}, \quad w = 0$
  - Neumann conditions (temporal constant gradients) for scalars:

$$\frac{\partial \theta}{\partial z} = \left. \frac{\partial \theta}{\partial z} \right|_{t=0}$$

  - Pressure: Dirichlet $p = 0$ (Neumann $\frac{\partial p}{\partial z} = 0$ is better)
  - SGS-TKE: Neumann $\frac{\partial e}{\partial z} = 0$
  - A damping layer can be switched on in order to absorb gravity waves.

Initial Conditions

All 3D-arrays are initialized with vertical profiles (horizontally homogeneous).

Two different profiles can be chosen:

Initial Conditions

All 3D-arrays are initialized with vertical profiles (horizontally homogeneous).

Two different profiles can be chosen:

- **constant (piecewise linear) profiles**
  - **e.g.** $u = 0, v = 0, \frac{\partial \theta}{\partial z} = 0$ **up to** $z = 1000\,\mathrm{m}, \frac{\partial \theta}{\partial z} = +1.0$ **up to top**

Initial Conditions

All 3D-arrays are initialized with vertical profiles (horizontally homogeneous).

Two different profiles can be chosen:

- **constant (piecewise linear) profiles**
  - **e.g.** $u = 0, v = 0, \frac{\partial \theta}{\partial z} = 0$ **up to** $z = 1000\,\mathrm{m}, \frac{\partial \theta}{\partial z} = +1.0$ **up to top**
- **velocity profiles calculated by a 1D-model (which is a part of PALM)**
  - **constant (piecewise linear) temperature profile is used for the 1D-model**

Initial Conditions

All 3D-arrays are initialized with vertical profiles (horizontally homogeneous).

Two different profiles can be chosen:

- **constant (piecewise linear) profiles**
  - **e.g.** $u = 0, v = 0, \frac{\partial \theta}{\partial z} = 0$ **up to** $z = 1000\,\mathrm{m}, \frac{\partial \theta}{\partial z} = +1.0$ **up to top**
- **velocity profiles calculated by a 1D-model (which is a part of PALM)**
  - **constant (piecewise linear) temperature profile is used for the 1D-model**

Under horizontally homogeneous initial conditions, random fluctuations have to be added in order to generate turbulence!