SFB 1313

Introduction to DuMu^x

Overview and Available Models

The (extended) DuMu^{X} Team (LH^{2})

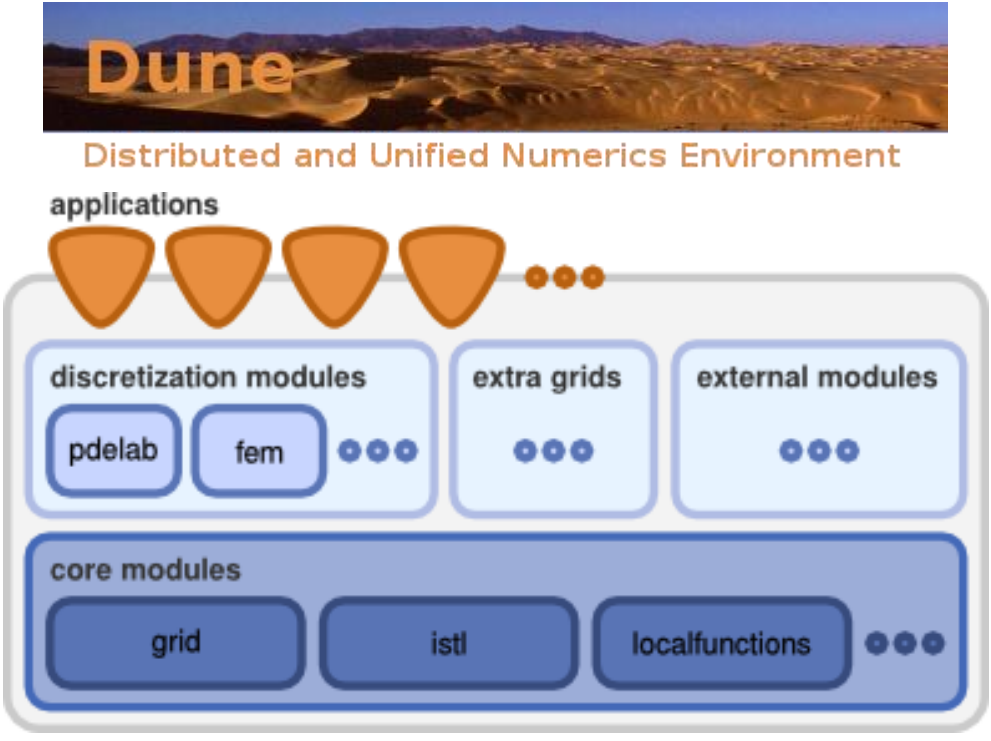
Overview

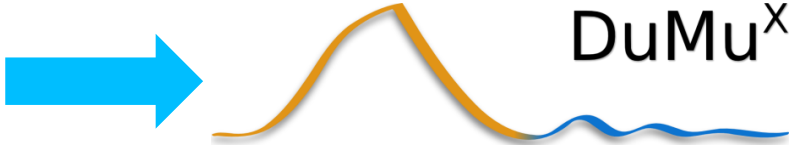
1. History and Structure
2. Available Models
3. Discretization Schemes
4. Model Components
5. Simulation Flow

DuMu^{X} Introduction

1 History and Structure

DuMu^X is a DUNE Module





The DUNE Framework

Distributed and Unified Numerics Environment

- **Developed** by scientists in Aachen, Bergen, Berlin, Dresden, Freiburg, Heidelberg, Münster, Stuttgart and Warwick.
- **Separation** of data structures and algorithms by abstract interfaces.
- Efficient implementation using **generic** programming techniques.
- **Reuse** of existing FE packages with a large body of functionality.
- Current stable release: **2.6** (March 2018).

DUNE core modules:

dune-common: basic classes

dune-geometry: geometric entities

dune-grid: abstract grid/mesh interface.

dune-istl: iterative solver template library

dune-localfunctions: finite element shape functions

Overview

DuMu^x: DUNE for Multi-{Phase, Component, Scale, Physics, ...} flow and transport in porous media. (current stable release 2.12, in transition to 3.0)

- **Goal**: **sustainable** and **consistent framework** for the implementation and application of **model concepts** and **constitutive relations**.
- Successfully applied to **gas storage** scenarios, **radioactive waste** disposal, **remediation** problems, **transport** of therapeutic agents, **fractured** porous media, and **subsurface-atmosphere coupling.**

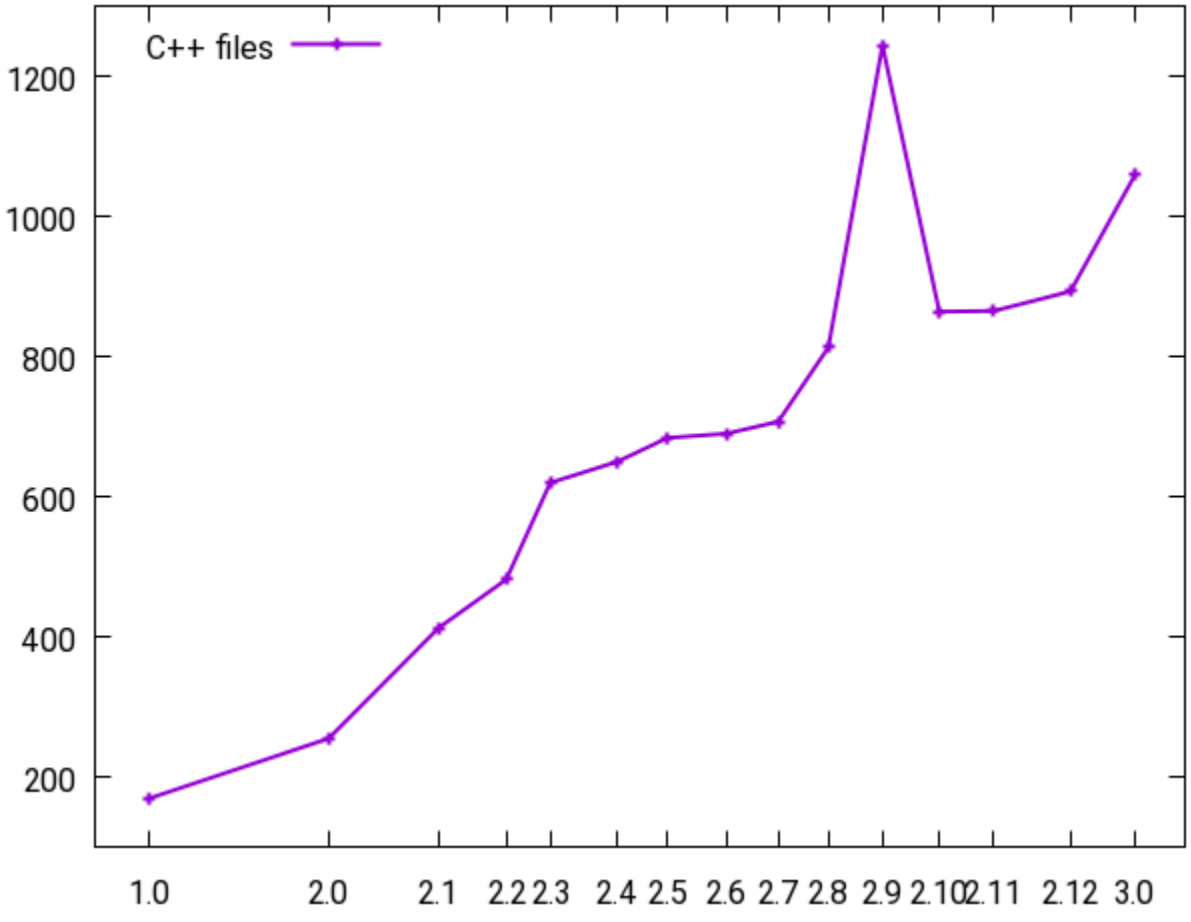
DuMu^x modules:

dumux-lecture: example applications for lectures offered by LH2 in Stuttgart.
dumux-pub: accompany a publication with all code and data to reproduce the results.
dumux-devel, dumux-appl…: current unpublished development and ongoing research

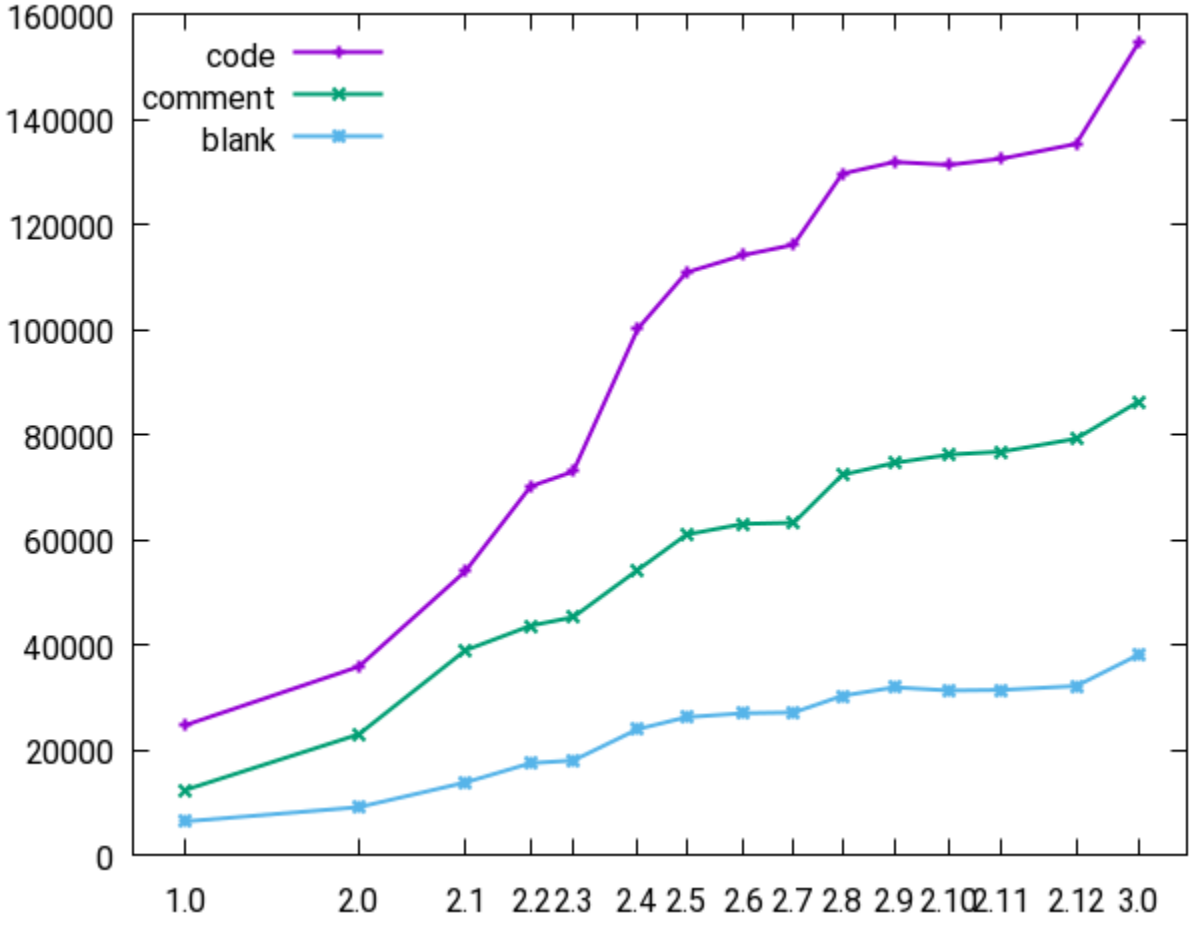
History

- Meanwhile **developed** by more than 20 PhD students and post docs at LH2.
- 1/2007: development **starts**.
- 1/7/2009: release **1.0**.
- 9/2010: **Split** into stable part and development part.
- 12/2010: Anonymous **read access** to the **SVN** trunk of the stable part.
- 25/2/2011: release **2.0**, ..., 31/10/2017: release **2.12**, **3.0-alpha**.
- 28/9/2015: Transition from Subversion to **Git**.
- More than 1000 ``real'' and unique release **downloads**.
- More than 100 peer-reviewed **publications** and PhD theses.

Evolution of C++ Files



Evolution of Code Lines



Available Models

Porous Medium Flow

Fully Implicit

1p, 1p2c
Richards

2p, 2p1c, 2p2c, 2pdfm, 2pminc, 2pnc, 2pncmin, co2

3p, 3p3c, 3pwateroil

mpnc

+ non-isothermal

Sequential

1p

2p, 2p2c

+ non-isothermal

Free Flow

Fully Implicit

stokes
navierstokes
rans (0-Eq, 2-Eq)

+ compositional
+ non-isothermal

Geomechanics

Fully Implicit

el1p2c

el2p

elastic

Multidomain

Fully Implicit

Boundary: Darcy-Darcy, Stokes-Darcy

Facet: Darcy-Darcy 1/2d-2/3d

Embedded: Darcy-Darcy 1/2d-3d

Further Capabilities and Characteristics

Porous Medium Flow

Fully Implicit

Discretization:

- Box method
- Cell-centered FV with TPFA or MPFA

Grid Adaptivity ☑

Parallel ☑

Sequential

Discretization:

- Cell-centered FV with TPFA, MPFA-L, MPFA-0 (2p), MFD (2p)

Grid Adaptivity ☑

Parallel ☑

Free Flow

Fully Implicit

Discretization:

- Staggered grid (MAC) method

Geomechanics

Fully Implicit

Discretization:

- Cell-centered method for flow
- Box for displacement

Parallel ☑

Multidomain

Fully Implicit

Discretization:

- Stokes-Darcy: cell-centered for PM flow, staggered for free flow
- …

What is DuMu^X?

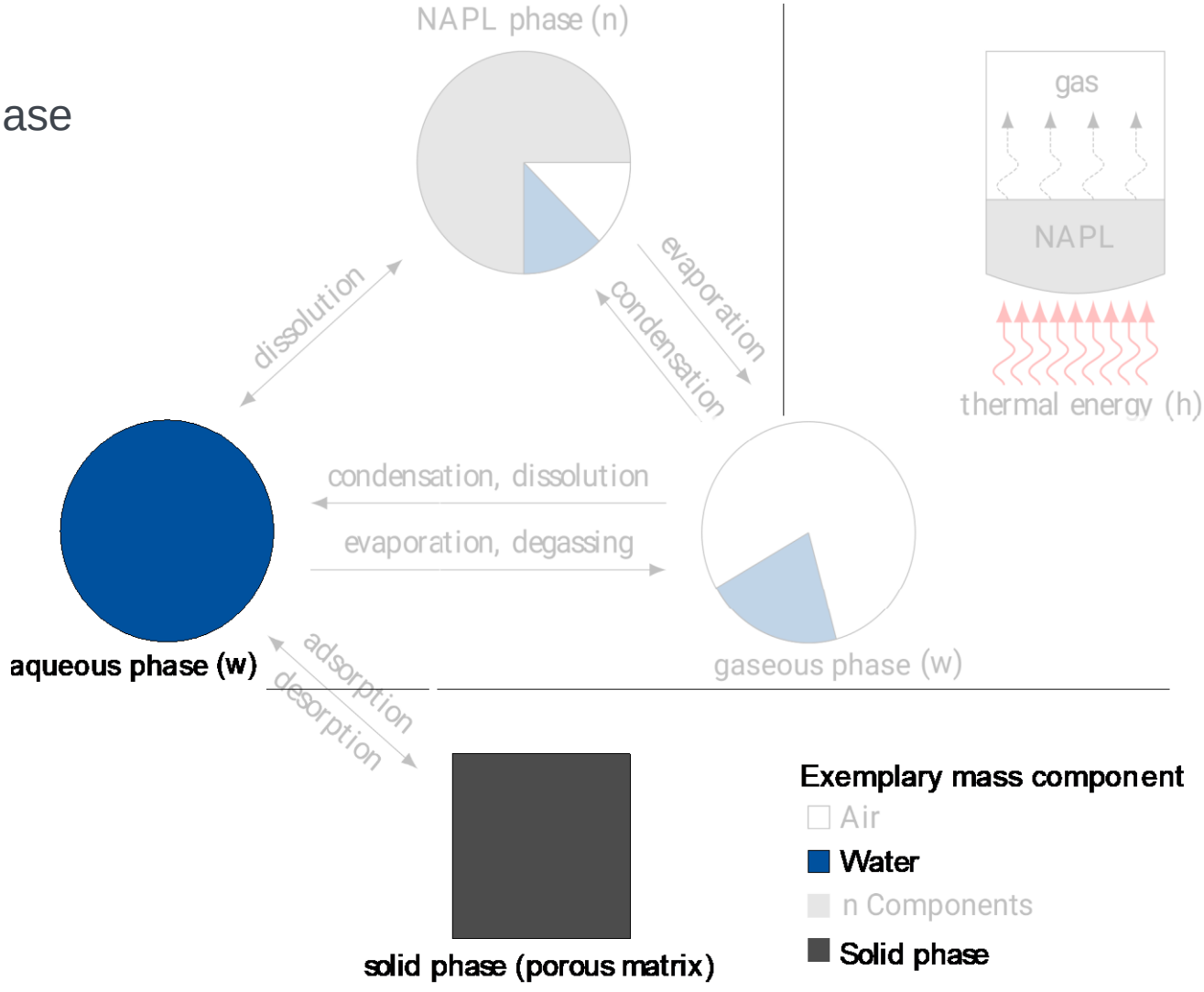
Where to get help? Where to report problems? How to stay updated?

- **Mailing lists** of DUNE (dune@dune-project.org) and DuMu^X (dumux@listserv.uni-stuttgart.de)
- Get **GitLab** accounts (non-anonymous) for better access
  - Dune GitLab (https://gitlab.dune-project.org/core)
  - DuMu^X GitLab (https://git.iws.uni-stuttgart.de/dumux-repositories/dumux)
- GitLab **Issue Tracker** (https://git.iws.uni-stuttgart.de/dumux-repositories/dumux/issues)
- **Doxygen** code **documentation** of
  - DUNE (http://dune-project.org/doc/doxygen.html)
  - DuMu^X (http://dumux.org/doxygen-stable/html-2.12/)
- DuMu^X **Handbook** (http://dumux.org/documentation.php)
- **Further information → http://dumux.org/**

2 Available Models

Models

1p – 1 phase



Models

1p – 1 phase

- Standard Darcy approach for the conservation of momentum

$$v = -\frac{\mathbf{K}}{\mu}\left(\mathbf{grad}\, p - \varrho \mathbf{g}\right)$$

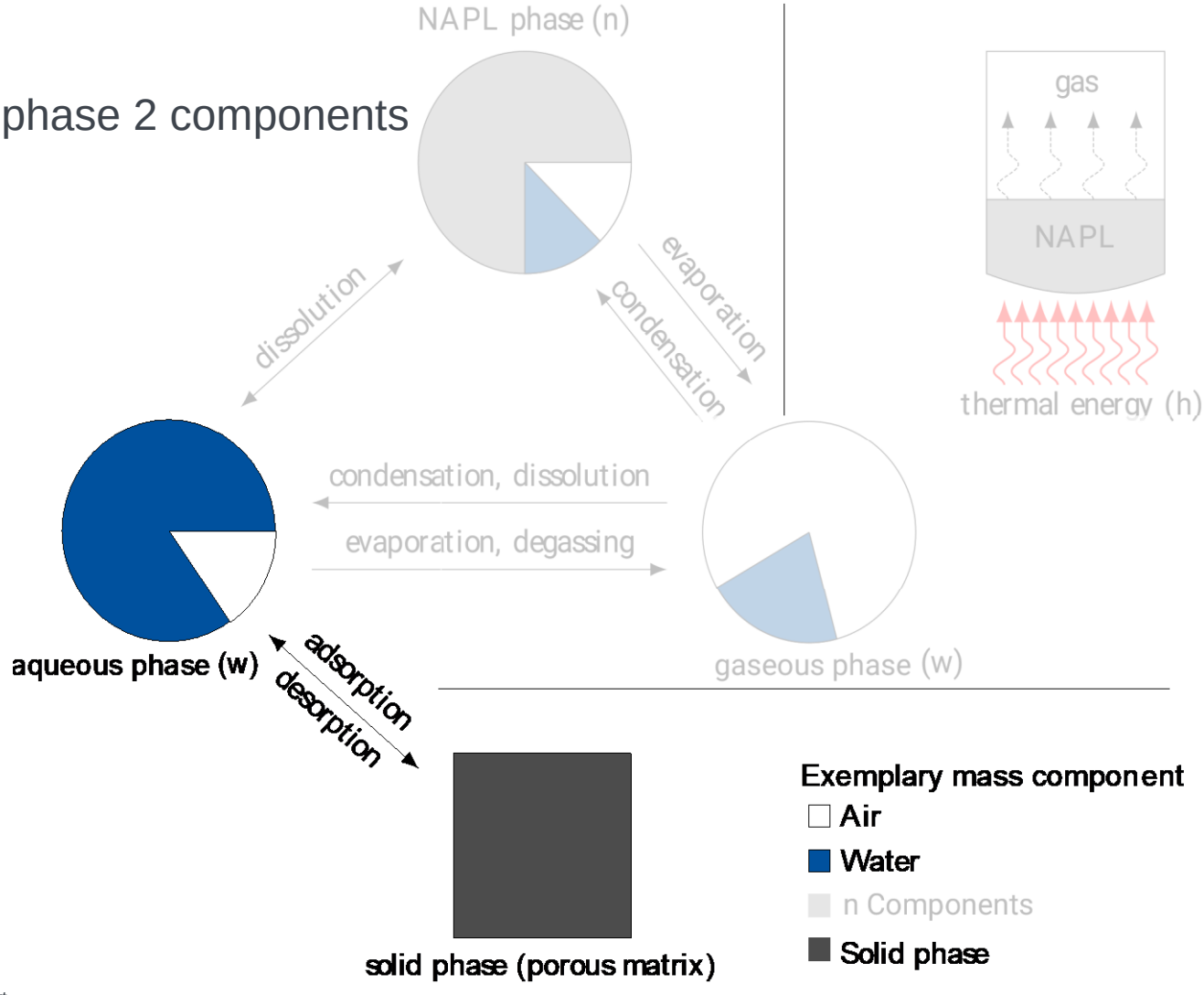
- Balance equation

$$\phi \frac{\partial \varrho}{\partial t} + \operatorname{div}\left\{-\varrho \frac{\mathbf{K}}{\mu}\left(\mathbf{grad}\, p - \varrho \mathbf{g}\right)\right\} = q$$

- Primary variable: p

Models

1p2c – 1 phase 2 components



Models

1p2c – 1 phase 2 components

- Continuity equation (including Darcy‘s law)

$$\phi \frac{\partial \varrho}{\partial t} - \operatorname{div} \left\{ \varrho \frac{\mathbf{K}}{\mu} \left(\mathbf{grad}\, p - \varrho \mathbf{g} \right) + \sum_{\kappa} \varrho D_{\mathrm{pm}}^{\kappa} \frac{M^{\kappa}}{M_{\alpha}} \mathbf{grad} x^{\kappa} \right\} = q, \ \kappa \in \{w, a\}$$

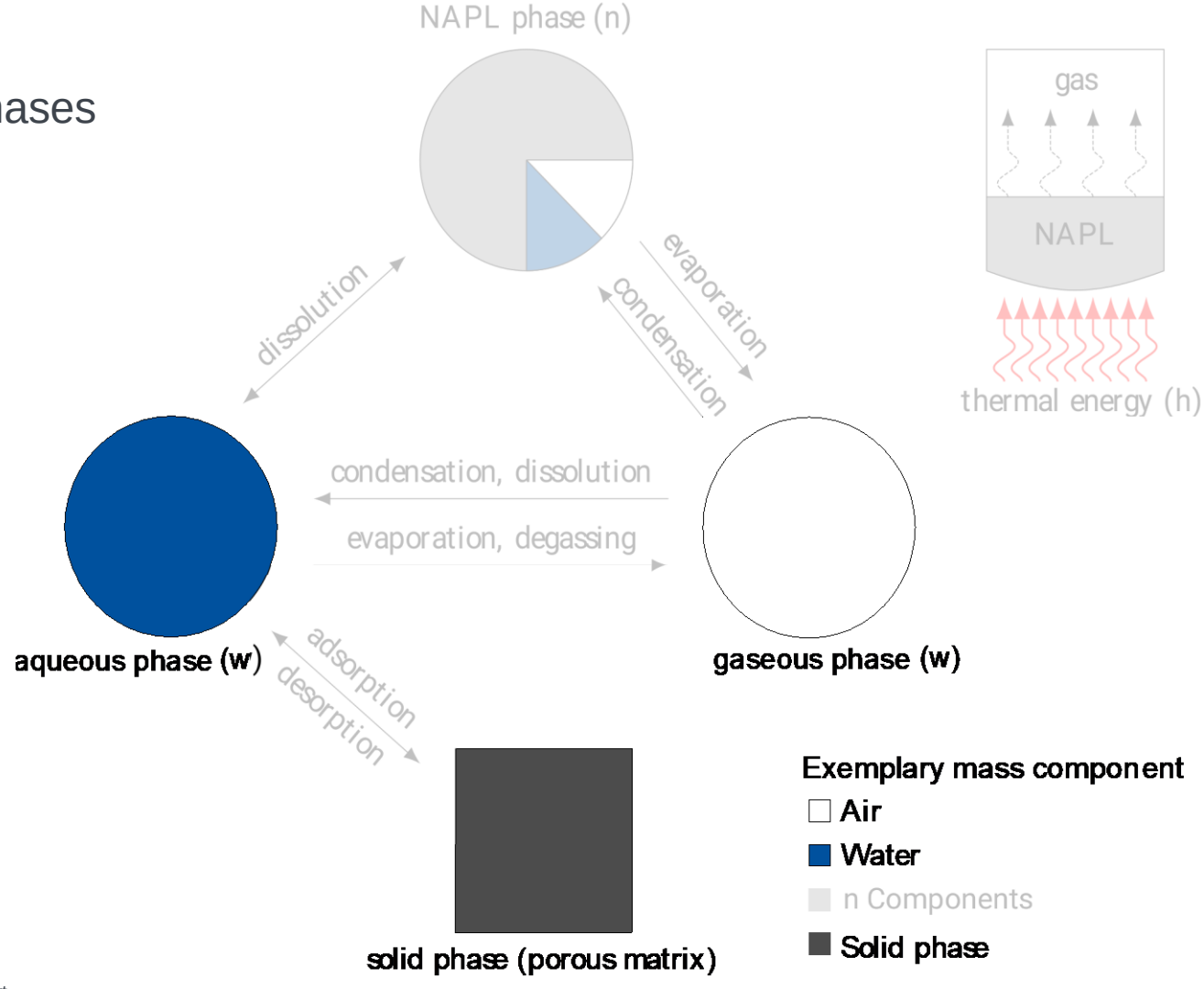
- Component balance

$$\phi \frac{\partial \varrho X^{\kappa}}{\partial t} - \operatorname{div} \left\{ \varrho X^{\kappa} \frac{\mathbf{K}}{\mu} \left(\mathbf{grad}\, p - \varrho \mathbf{g} \right) + \varrho D_{\mathrm{pm}}^{\kappa} \frac{M^{\kappa}}{M_{\alpha}} \mathbf{grad} x^{\kappa} \right\} = q$$

- Primary variables: $p, \ x^{\kappa}$

Models

2p – 2 phases



Models

2p – 2 phases

- Momentum conservation via standard multiphase Darcy approach:

$$v_\alpha = -\frac{k_{r\alpha}}{\mu_\alpha}\mathbf{K}\left(\mathbf{grad}\, p_\alpha - \varrho_\alpha \mathbf{g}\right)$$

- Continuity equation with Darcy‘s law included

$$\phi\frac{\partial \varrho_\alpha S_\alpha}{\partial t} - \mathrm{div}\left\{\varrho_\alpha \frac{k_{r\alpha}}{\mu_\alpha}\mathbf{K}\left(\mathbf{grad}\, p_\alpha - \varrho_\alpha \mathbf{g}\right)\right\} - q_\alpha = 0,\ \alpha \in \{w, n\}$$

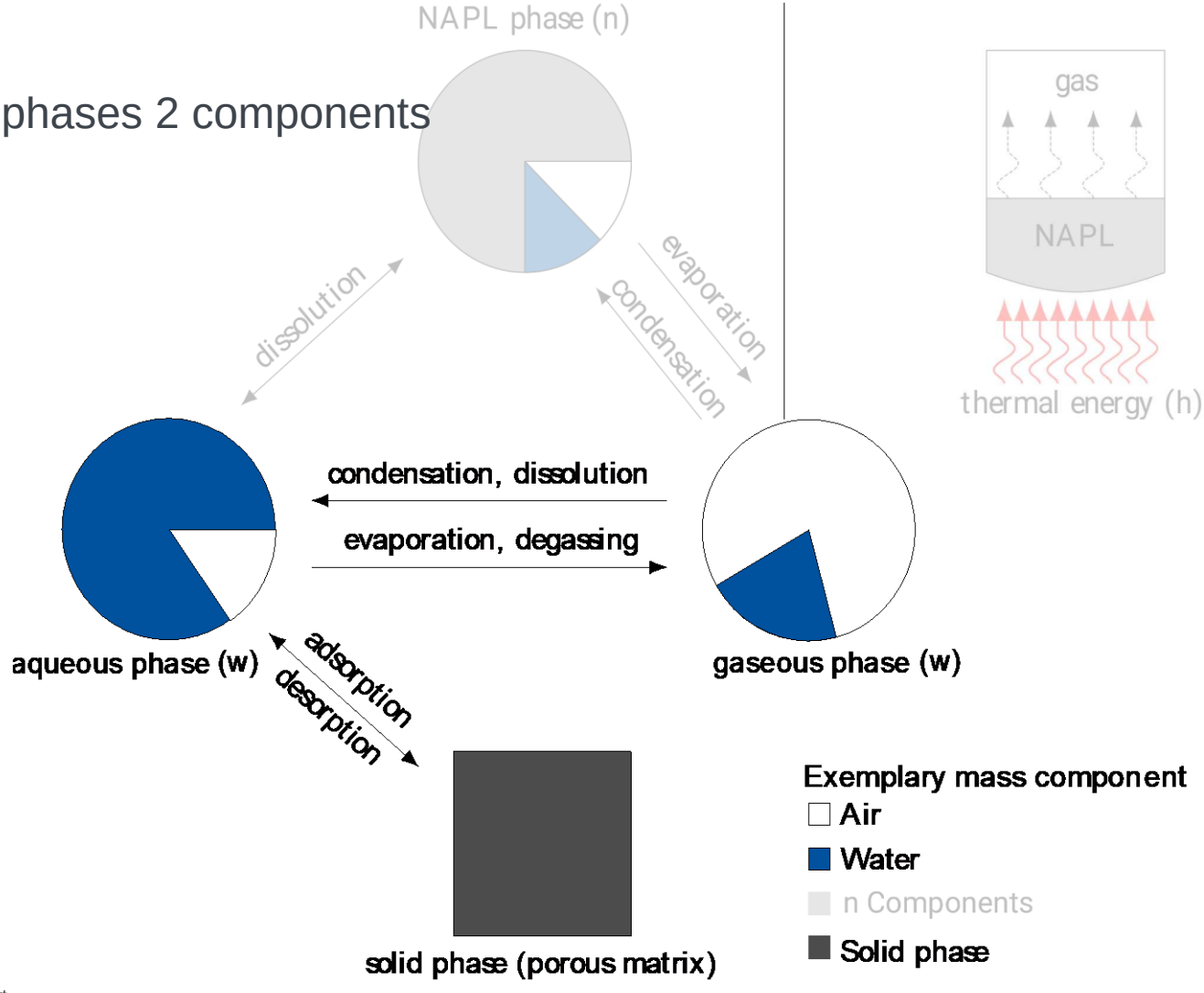
- Constitutive relations: $p_c = p_n - p_w$

$$S_w + S_n = 1$$

- Primary variables: $p_w, S_n \ / \ p_n, S_w$

Models

2p2c – 2 phases 2 components



Models

2p2c – 2 phases 2 components

- 2 phases $\alpha \in \{w, n\}$ each composed of 2 components $\kappa \in \{w, n\}$
- Standard multiphase Darcy for the conservation of momentum

$$v_\alpha = -\frac{k_{r\alpha}}{\mu_\alpha}\mathbf{K}\left(\mathbf{grad}\, p_\alpha - \varrho_\alpha \mathbf{g}\right)$$

- Balance equation for each component:

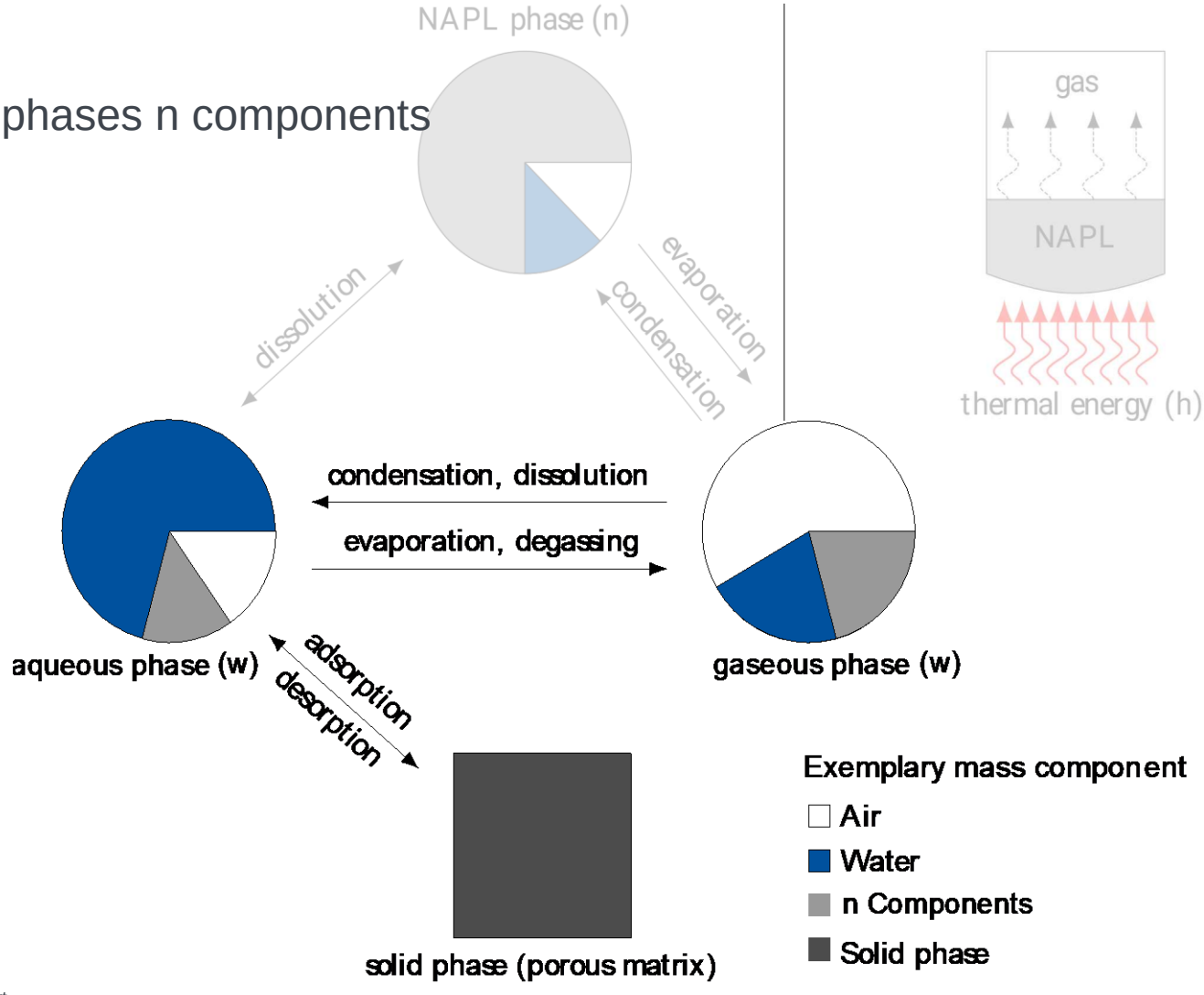
$$\frac{\partial \sum_\alpha \varrho_\alpha X_\alpha^\kappa \phi S_\alpha}{\partial t} - \sum_\alpha \mathrm{div}\left\{\varrho_\alpha X_\alpha^\kappa v_\alpha\right\} - \sum_\alpha \mathrm{div}\left\{D_{\alpha,\mathrm{pm}}^\kappa \varrho_\alpha \mathbf{grad} X_\alpha^\kappa\right\} - \sum_\alpha q_\alpha^\kappa = 0$$

$\forall \kappa, \forall \alpha$

- Constitutive relations: $S_w + S_n = 1,\ p_c = p_n - p_w$
- Primary variables: $p_w, S_n \ / \ p_n, S_w$

Models

2pnc – 2 phases n components



Models

2pnc – 2 phases n components

- 2 phases $\alpha \in \{w, n\}$ each composed of up to n components $\kappa \in \{w, n, ...\}$
- Standard multiphase Darcy for the conservation of momentum

$$v_\alpha = -\frac{k_{r\alpha}}{\mu_\alpha}\mathbf{K}\left(\mathbf{grad}\, p_\alpha - \varrho_\alpha \mathbf{g}\right)$$

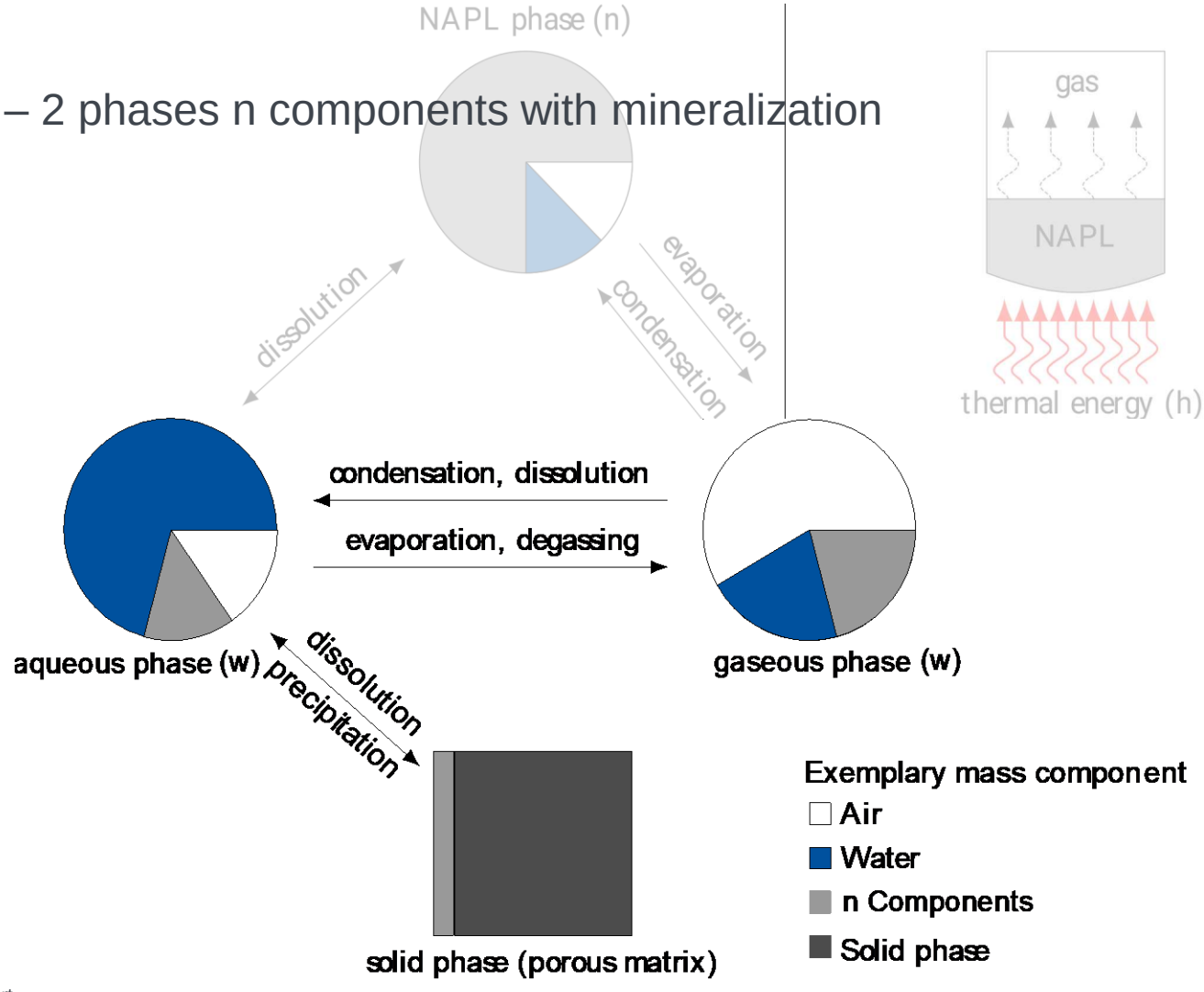
- Balance equation for each component:

$$\frac{\partial \sum_\alpha \varrho_\alpha X_\alpha^\kappa \phi S_\alpha}{\partial t} - \sum_\alpha \mathrm{div}\left\{\varrho_\alpha X_\alpha^\kappa v_\alpha\right\} - \sum_\alpha \mathrm{div}\left\{D_{\alpha,\mathrm{pm}}^\kappa \varrho_\alpha \mathbf{grad} X_\alpha^\kappa\right\} - \sum_\alpha q_\alpha^\kappa = 0$$

- Constitutive relations: $S_w + S_n = 1,\ p_c = p_n - p_w$
- Primary variables: $p_w, S_n, X_\alpha \ / \ p_n, S_w, X_\alpha$

Models

2pncmin – 2 phases n components with mineralization



Models

2pncmin – 2 phases n components with mineralization

- 2 phases $\alpha \in \{w, n\}$ each composed of up to n components $\kappa \in \{w, n, ...\}$
- Standard multiphase Darcy for the conservation of momentum

$$v_\alpha = -\frac{k_{r\alpha}}{\mu_\alpha}\mathbf{K}\left(\mathbf{grad}\, p_\alpha - \varrho_\alpha \mathbf{g}\right)$$

- Balance equation for each component:

$$\frac{\partial \sum_\alpha \varrho_\alpha X_\alpha^\kappa \phi S_\alpha}{\partial t} - \sum_\alpha \mathrm{div}\left\{\varrho_\alpha X_\alpha^\kappa v_\alpha\right\} - \sum_\alpha \mathrm{div}\left\{D_{\alpha,\mathrm{pm}}^\kappa \varrho_\alpha \mathbf{grad} X_\alpha^\kappa\right\} - \sum_\alpha q_\alpha^\kappa = 0$$

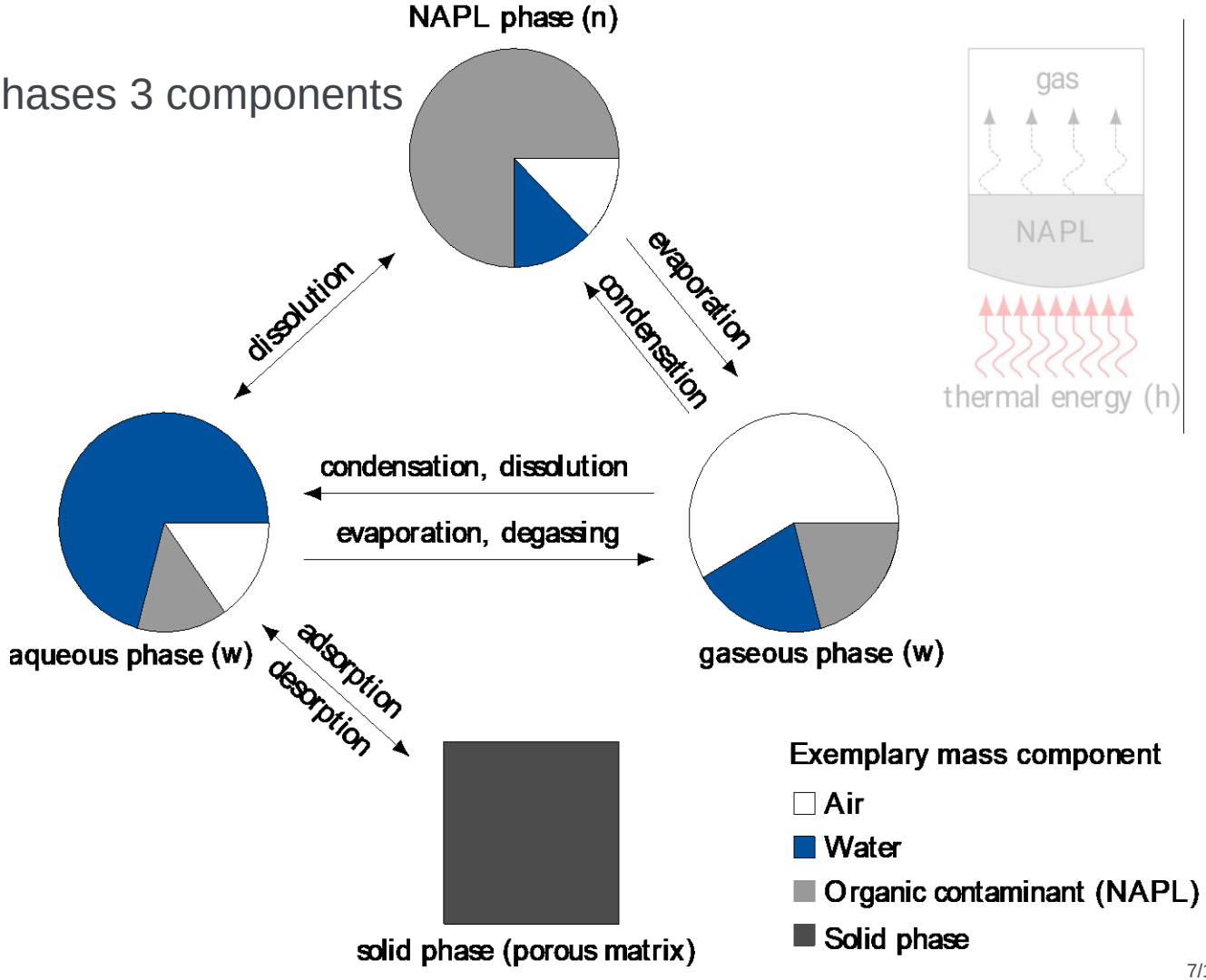
$\forall \kappa, \forall \alpha$

- Mass balance for solid/mineral phase λ:

$$\frac{\partial(\varrho_\lambda \phi_\lambda)}{\partial t} = q_\lambda, \ \forall \lambda$$

Models

3p3c – 3 phases 3 components



Models

3p3c – 3 phases 3 components

- 3 phases $\alpha \in \{w, g, n\}$ each composed of up to three components $\kappa \in \{w, a, n\}$
- standard multiphase Darcy for the conservation of momentum

$$v_\alpha = -\frac{k_{r\alpha}}{\mu_\alpha}\mathbf{K}\left(\mathbf{grad}\, p_\alpha - \varrho_\alpha \mathbf{g}\right)$$

- Balance equation

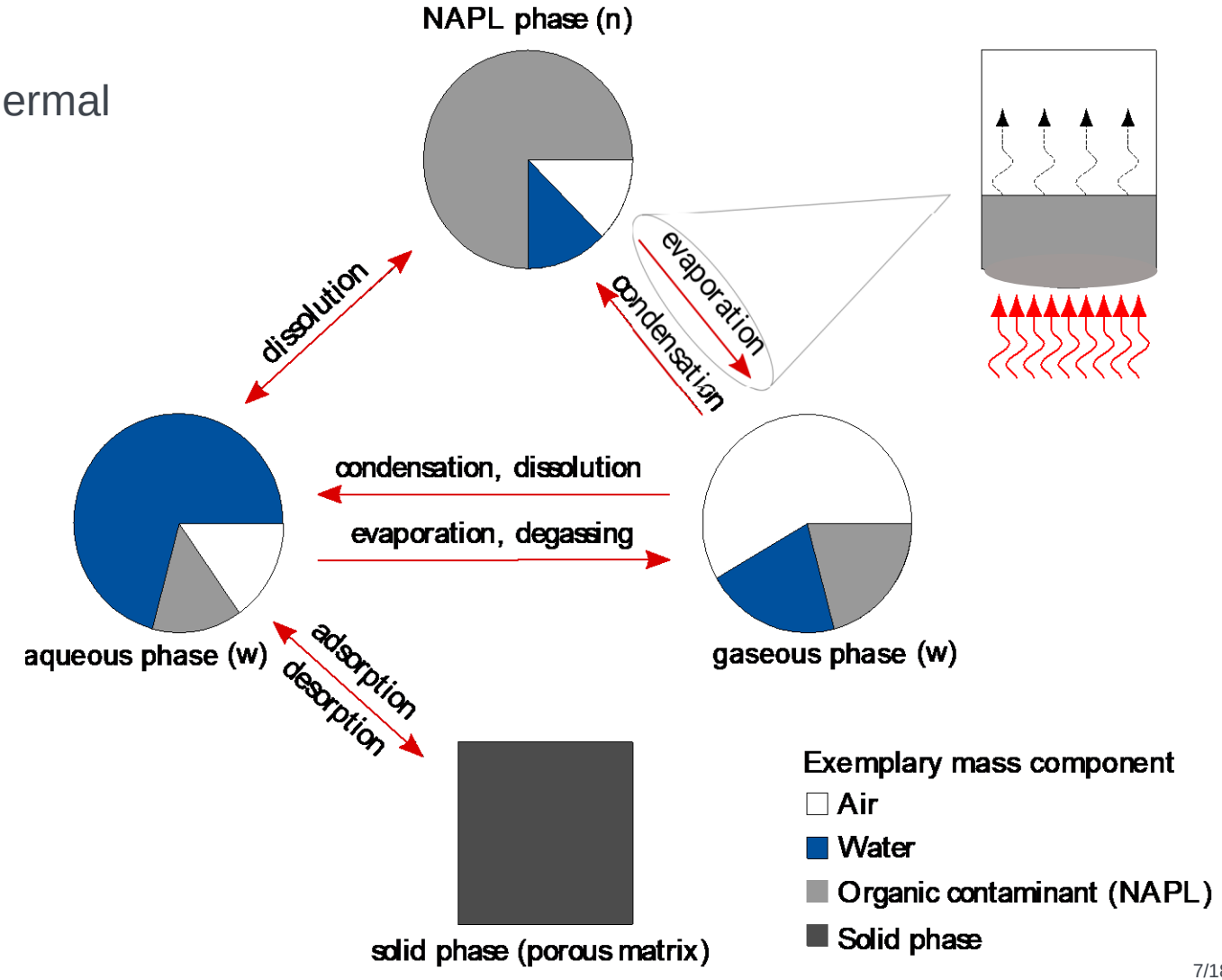
$$\phi \frac{\partial(\sum_\alpha \varrho_{\alpha,mol} x_\alpha^\kappa S_\alpha)}{\partial t} - \sum_\alpha \mathrm{div}\left\{\varrho_{\alpha,mol} x_\alpha^\kappa v_\alpha\right\}$$

$$- \sum_\alpha \mathrm{div}\left\{D_{\mathrm{pm}}^\kappa \varrho_{\alpha,mol} \mathbf{grad} x_\alpha^\kappa\right\} - q^\kappa = 0 \qquad \forall \kappa,\ \forall \alpha$$

- auxiliary conditions: $S_w + S_n + S_g = 1\ ,\ \ x_\alpha^w + x_\alpha^a + x_\alpha^c = 1$
- Primary variables: S_w, S_n, p_g

Models

Non-Isothermal



Models

Non-Isothermal

- Assuming local thermal equilibrium → one energy conservation equation for the porous solid matrix and the fluids:

$$\phi \frac{\partial \sum_\alpha \varrho_\alpha u_\alpha S_\alpha}{\partial t} + (1-\phi)\frac{\partial (\varrho_s c_s T)}{\partial t} - \sum_\alpha \operatorname{div}\left\{\varrho_\alpha h_\alpha \frac{k_{r\alpha}}{\mu_\alpha}\mathbf{K}\left(\mathbf{grad}\, p_\alpha - \varrho_\alpha \mathbf{g}\right)\right\} - \operatorname{div}\left(\lambda_{pm}\mathbf{grad}\, T\right) - q^h = 0.$$

With the specific internal energy of the phase α : $u_\alpha = h_\alpha - p_\alpha/\varrho_\alpha$

- Primary variable : T
- can currently be used on top of **most porous media models**

Models

Free Flow – (Reynolds-averaged) Navier-Stokes (1pncni)

$$\frac{\partial\left(\varrho_{\mathrm{g}}\bar{X}_{\mathrm{g}}^{\kappa}\right)}{\partial t}+\nabla\cdot\left(\varrho_{\mathrm{g}}\bar{X}_{\mathrm{g}}^{\kappa}\bar{\mathbf{v}}_{\mathrm{g}}\right)+\nabla\cdot\mathbf{j}_{\mathrm{mass,eff}}^{\kappa,\mathrm{ff}}=0$$

$$\mathbf{j}_{\mathrm{mass,eff}}^{\kappa,\mathrm{ff}}=-\left(D_{\mathrm{g}}^{\kappa}+D_{\mathrm{t}}\right)\varrho_{\mathrm{mol,g}}M^{\kappa}\nabla\bar{x}_{\mathrm{g}}^{\kappa}$$

$$\frac{\partial\left(\varrho_{\mathrm{g}}\bar{\mathbf{v}}_{\mathrm{g}}\right)}{\partial t}+\nabla\cdot\left(\varrho_{\mathrm{g}}\bar{\mathbf{v}}_{\mathrm{g}}\bar{\mathbf{v}}_{\mathrm{g}}^{\mathsf{T}}\right)-\nabla\cdot\boldsymbol{\tau}_{\mathrm{eff}}+\nabla\cdot\left(\bar{p}_{\mathrm{g}}\mathbf{I}\right)-\varrho_{\mathrm{g}}\mathbf{g}=0$$

$$\boldsymbol{\tau}_{\mathrm{eff}}=\varrho_{\mathrm{g}}\left(\nu_{\mathrm{g}}+\nu_{\mathrm{t}}\right)\left(\nabla\bar{\mathbf{v}}_{\mathrm{g}}+\nabla\bar{\mathbf{v}}_{\mathrm{g}}^{\mathsf{T}}\right)$$

$$\frac{\partial\left(\varrho_{\mathrm{g}}\bar{u}_{\mathrm{g}}\right)}{\partial t}+\nabla\cdot\left(\varrho_{\mathrm{g}}\bar{h}_{\mathrm{g}}\bar{\mathbf{v}}_{\mathrm{g}}\right)+\sum_{\kappa\in\{\mathrm{w,a}\}}\nabla\cdot\left(\bar{h}_{\mathrm{g}}^{\kappa}\mathbf{j}_{\mathrm{mass,eff}}^{\kappa,\mathrm{ff}}\right)+\nabla\cdot\mathbf{j}_{\mathrm{cond,eff}}^{\mathrm{ff}}=0$$

$$\mathbf{j}_{\mathrm{cond,eff}}^{\mathrm{ff}}=-\left(\lambda_{\mathrm{g}}+\lambda_{\mathrm{t}}\right)\nabla\bar{T}$$

- Primary variable : p, v, X, T, …

DuMu^{X} Introduction

3 Discretization Schemes

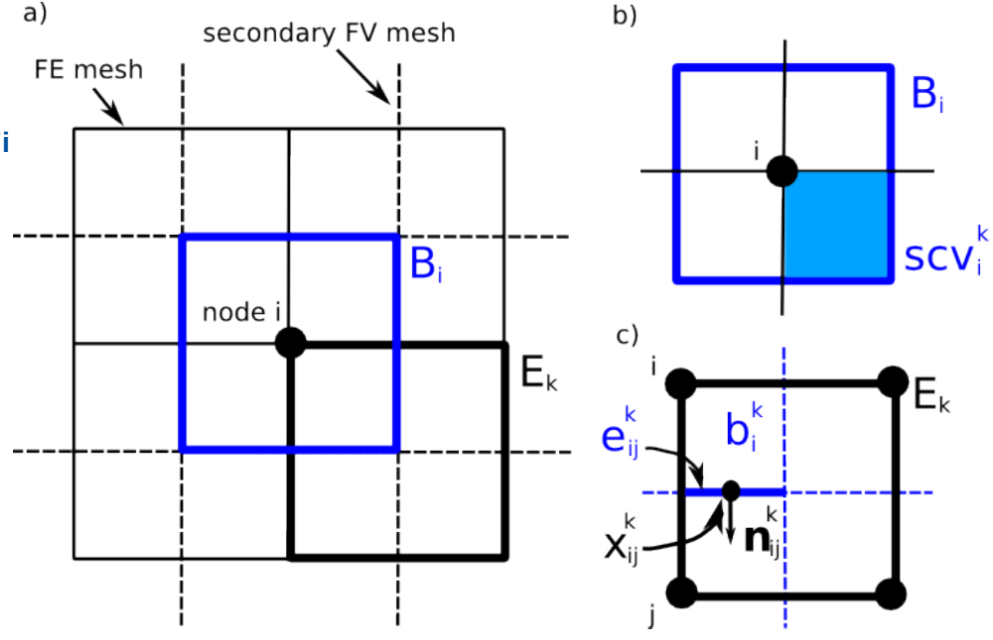
Spatial Discretization Schemes

Box method

- Discretize model domain with **FE** mesh creating **elements E_k**
- Construct secondary **FV** mesh → **Box B_i**
- FE mesh divides Box into subcontrolvolumes (**scv's**) **b_i^k** inside **E_k**
- Subcontrolvolumefaces (scvf's) are needed between **scv's** **b_i^k** and **b_j^k** with **$|e_{ij}^k|$** as the length of the scvf

→ **unstructured grid** (from FE method)

→ **mass conservative** (FV method)



Spatial Discretization Schemes

Cell Centered Finite Volume Method (CC)

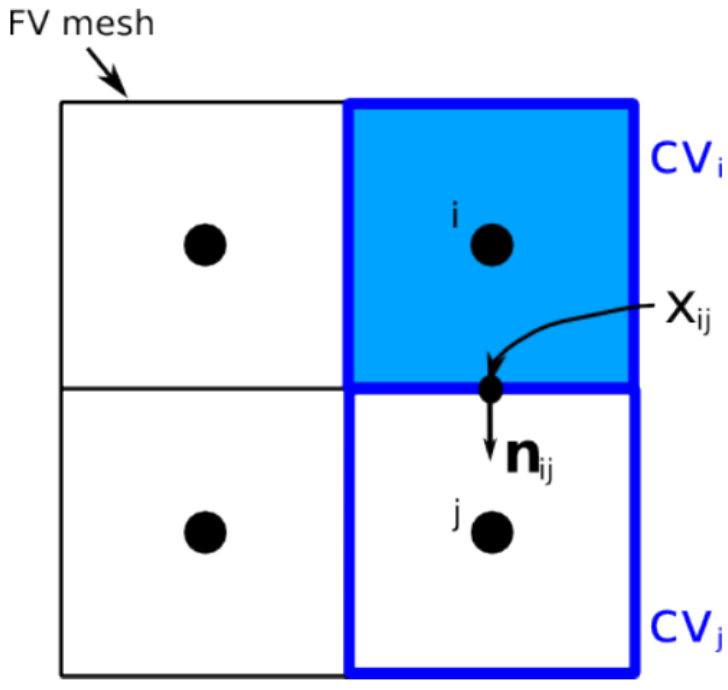
- Use elements of the grid as control volumes
- Discrete **values** are determined at the element/control volume **center** i
- **Mass/energy** fluxes are evaluated at the integration point $\mathbf{x}_{ij}$ (at the **mid** of **cvf's**)

  → **two-point flux approximation (TPFA)**

→ **robust** and **mass conservative**

→ Should be applied for **k-orthogonal grids** only

→ Extension to general grids with **MPFA**

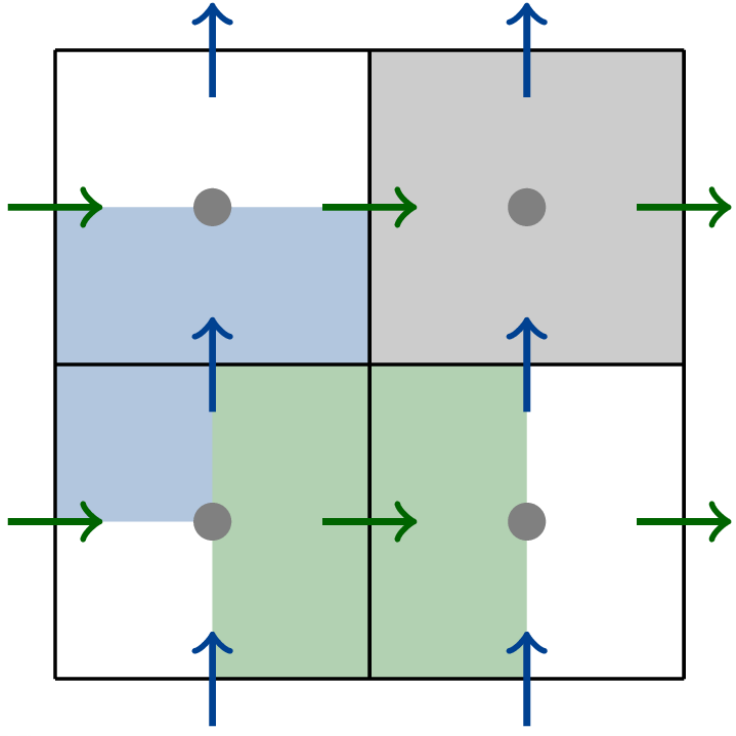


Spatial Discretization Schemes

Staggered Grid or Marker-and-Cell Method (MAC)

- Use elements of the grid as control volumes for the Scalar quantities
- Control volumes for the velocity components are shifted half a cell in each direction
- **TPFA: Mass/energy** fluxes at **mid** of **cvf's**, **momentum** fluxes at **mid** of **staggered-cvf's**
- **robust** and **mass conservative**

→ Should be applied for **k-orthogonal grids** only



DuMu^{X} Introduction

4 Model Components

DuMu^x Material

Constitutive equations (material laws, fluidsystems, …)

- Two of the **biggest challenges** in porous media simulation: highly heterogeneous distribution of **parameters** and the complex nonlinear **material laws**.
- **Difficult** to achieve modularity due to the strong interconnection of these properties.
- Try to achieve a **modular structure** by a separation into the following parts:
  - Fluid states
  - Fluid systems
  - Constraint solvers
  - Components
  - Fluid-matrix interactions
  - Spatial parameters
- See the course block starting **Thursday at 11:00 h**.

DuMu^x Problem

Configuring and running a simulation scenario

- A DuMu^x **problem**
  - **implements** a **specific model scenario**
  - **configures** a model for the scenario using
    - **properties** (see the course block starting **Thursday at 9:00 h**)
    - and **parameters** (see the course block starting **today at 17:00 h**)
  - defines **boundary** conditions
  - defines **initial** conditions
  - defines **source/sink** terms
- The **spatial parameters**
  - define spatial parameters of the **porous material** (permeability, porosity, material laws)
- See the course block starting **today at 13:30 h**.

Newton´s method

The basic ingredient for solving nonlinear PDEs

- All equations are considered **non-linear** → Newton scheme

$$\mathbf{J}(\mathbf{U_n})(\mathbf{U_{n+1}} - \mathbf{U_n}) = -\mathbf{R}(\mathbf{U_n})$$

- Where the **residual** ***R*** has the general form

$$\mathbf{R}(\mathbf{U}) = \frac{\partial \mathbf{S}(\mathbf{U})}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{U}) + \mathbf{Q}(\mathbf{U}) = \mathbf{0}$$

- with storage ***S***, fluxes ***F***, sources ***Q*** and solution vector ***U***

- Computing the Jacobian → Numeric differentiation (deflect ***U***, recompute ***R***)

Model components

Necessary ingredients

- `…LocalResidual`, *…localresidual.hh*
- `{Model}VolumeVariables`, *{model}/volumevariables.hh*
- `…FluxVariables`, *…fluxvariables.hh*
- `…PrimaryVariables`, possibly customized in terms of a formulation, *{model}/model.hh*
- `…ModelTraits`, *{model}/model.hh*
- `…Indices`, *{model}/model.hh*
- `{Model}VtkOutputFields`, *{model}/vtkoutputfields.hh*

Model components

The local residual

- The local residual computes the **residual per element** for all associated degree of freedoms (located at element center (CCFV) / vertex positions (Box))
- Every model uses a …*localresidual.hh* with a `…LocalResidual` class, e.g., `ImmiscibleLocalResidual`, `CompositionalLocalResidual`
- The `LocalResidual` class consists of (most importantly):
  - a `computeStorage` method computing $\boldsymbol{S}(\boldsymbol{U})$
  - a `computeFlux` method computing $\boldsymbol{F}(\boldsymbol{U}) \cdot \boldsymbol{n}$ using the `FluxVariables` class
  - a `computeSource` method computing $\boldsymbol{Q}(\boldsymbol{U})$ forwarding to the problem‘s `source` / `sourceAtPos` methods

Model components

The volume variables: From primary to secondary variables

- Where to get the variables needed to compute storage ***S***(***U***), flux ***F***(***U***) · ***n*** *and source* ***Q***(***U***)?
- Every model has a *volumevariables.hh* with a `{Model}VolumeVariables` class
- Volume variables are all **variables** defined in a **control volume** (e.g. pressure, density, porosity, saturation, viscosity, …)
- The `VolumeVariables` class
  - takes primary variables of the current Newton step (e.g. pressure *p*)
  - computes and gathers all secondary variables (e.g. density $\rho(p, T)$)
- The `ElementVolumeVariables` class is a collection of all `VolumeVariables` needed to compute the local residual of an element (stencil)

Model components

The flux variables: Flux over the control volume boundaries

- A `FluxVariables` class object exists for every (sub) **control volume face**
- Helper class to compute volume fluxes ***f(u)*** using the respective discretization scheme

$$f(u) = \mathbf{F}(\mathbf{U}) \cdot \mathbf{n}$$

- **Box** method – **fluxes** over **inner** sub control volume boundaries
- **CCFV** method – **fluxes** over **intersections** (facets / codim 1 entities) with neighboring elements, i.e. (sub) control volume boundaries

Model components

Primary variables and equations

- A model has the same number of **primary variables** and **equations** (`numEq`)
- Therefore the same container is used (`PrimaryVariables`)
- It is usually a `Dune::Fieldvector<Scalar, numEq>` class (array with some calculus abilities)
- Primary variables of models with a primary-variable switch also have a `state`
- (`Scalar` is the type used for all scalar values, it defaults to `double` but may be changed to compute with types of higher / lower precision)

Model components

The Model Traits

- Each model provides its …`ModelTraits`, a struct containing

```
using Indices = …;
static constexpr int numEq() {…}
static constexpr int numPhases() {…}
static constexpr int numComponents() {…}
…
static constexpr bool enableAdvection() {…}
static constexpr bool enableMolecularDiffusion() {…}
static constexpr bool enableEnergyBalance() {…}
…
static constexpr … priVarFormulation() {…}
```

- To be used as, e.g., `ModelTraits::numEq()`.

Model components

Indices: organizing primary variables and equations

- A model has an `Indices` class providing named access to the array‘s entries for
  - **primary variables** (e.g. setting Dirichlet boundary conditions per PV)

    Example:

    ```
    • PrimaryVariables values(0.0); // initialize to zero
    • values[pressureIdx] = 1e5; // pressure in Pa
    ```

  - **equations** (e.g. setting source terms for certain equations)

    Example:

    ```
    • PrimaryVariables sources(0.0); // initialize to zero
    • sources[contiOEqIdx] = -1e-5; // mass balance sink term in kg/(s*m³)
    ```

Model components

Vtk Output Fields

- Each model provides …`VtkOutputFields`, a class where the **default** output fields are set by employing a `VtkOutputModule vtk` via, e.g.,

```
vtk.addVolumeVariable( [](const auto& v)
                       { return v.saturation(FS::phase1Idx); },
                       "S_n“ );
```

- In addition to the default fields, custom fields can be added to the output by defining a function

  `void addFieldsToWriter(VtkWriter& vtk)`

  in the problem class.

DuMu^{X} Introduction

5 Simulation flow

Simulation Flow

1. main
2. time step
3. Newton
4. element

Initialize
foreach time step
    prepare update
    foreach NEWTON iteration
        foreach element
            - calculate element residual vector and Jacobian matrix
            - assemble into global residual vector and Jacobian matrix
        endfor
        solve linear system
        update solution
        check for NEWTON convergence
    endfor
    - adapt time step size, possibly redo with smaller step size
    - write result
endfor
finalize

Simulation Flow: see course block starting today at 13:30 h

1. main
2. time step
3. Newton
4. element

Initialize
foreach time step
    prepare update
    foreach NEWTON iteration
        foreach element
            - calculate element residual vector and Jacobian matrix
            - assemble into global residual vector and Jacobian matrix
        endfor
        solve linear system
        update solution
        check for NEWTON convergence
    endfor
    - adapt time step size, possibly redo with smaller step size
    - write result
endfor
finalize

University of Stuttgart
Institute for Modelling Hydraulic and Environmental Systems

Thank you!

DuMu^x

http://dumux.org/
https://dune-project.org/

e-mail dumux@listserv.uni-stuttgart.de
phone +49 (0) 711 685-69162
fax +49 (0) 711 685-60430

University of Stuttgart
Institute for Modelling Hydraulic and Environmental Systems
Pfaffenwaldring 61, 70569 Stuttgart

SFB 1313