MueLu: The next-generation Trilinos multigrid package

Tobias Wiesner,
Jeremie Gaidamour,
Jonathan Hu,
Ray Tuminaro,
Chris Siefert,
Michael Gee

1st European Trilinos User Group Meeting

Lausanne, Switzerland, June, 4th, 2012

Agenda

- Motivation for a new multigrid software package
- Current status of MueLu
- Object-oriented design for a flexible multigrid code
- User perspective of MueLu
- Examples
- Conclusions

MueLu - Objectives

1

Existing Multigrid Capabilities in Trilinos

ML 5.0

PRO

- Supports:
  - Smoothed aggregation
  - Petrov-Galerkin
  - AMG for H (curl)
- Mature/stable software
- Robust capabilities
- Fast implementation (target: performance/HPC)
- broad user base

CONTRA

- Non 64-bit integer
  - Only scalar type „double“
  - Only Epetra and C-based interface
- Lack of modularity (extensibility)
- Lack of tests (no unit tests)
- Duplication of functionality with other Trilinos packages

Objectives for a new Multigrid library – MueLu

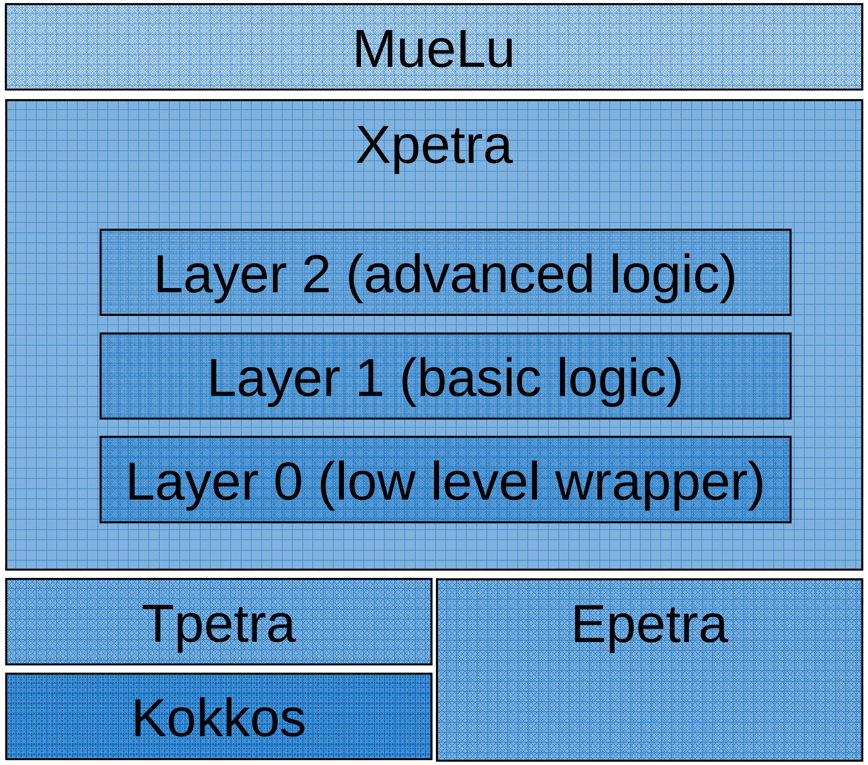
- Support for a variety of scalar types (e.g. complex, float)
- Hybrid parallelism: MPI, MPI+threads, MPI+MPI
- Support for many-core architectures
- Extensibility: facilitate development of other algorithms
  - Energy minimization methods
  - Ruge Stueben AMG
  - Geometric MG
- Preconditioner reuse to reduce setup expense

- Epetra/Tpetra look & feel for algorithms

MueLu & Xpetra - Status

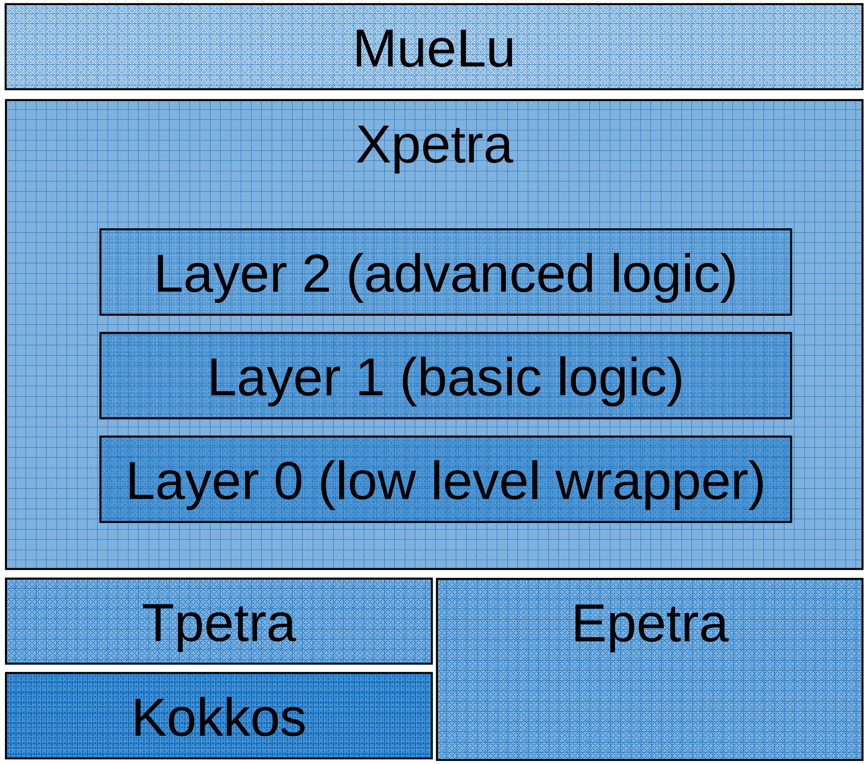
Xpetra – *petra goes future?

- Wrapper for Epetra and Tpetra
  - Look & feel of Epetra and Tpetra
  - Intended to be used for porting existing Epetra-based software
  - Intended to write new Tpetra-based applications
  - Based on Tpetra user interface
- Layer concept:
  - Layer 0: low level wrapper for Epetra and Tpetra (automatically generated code)
  - Layer 1: operator views
  - Layer 2: support for blocked operators
- Future: independent Trilinos linear Algebra wrapper package (beside of Thyra?)



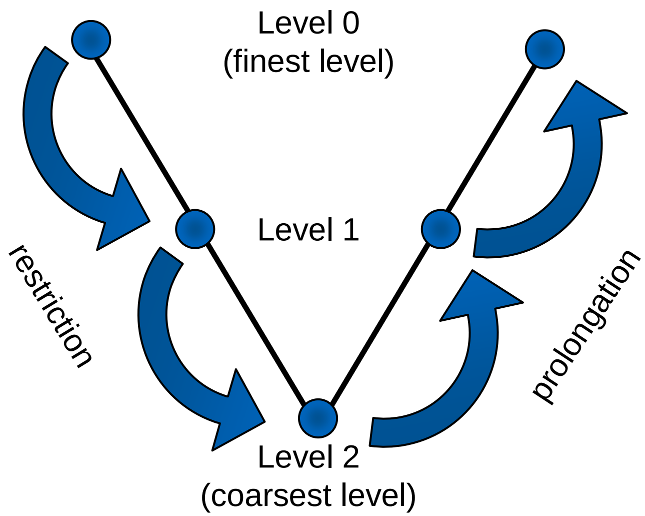
Status - Xpetra

- Layer 0: Epetra/Tpetra wrapper for
  - Maps
  - (Multi)Vectors
  - CrsGraph
  - CrsMatrix
  - Export/Import
- Layer 1: basic logic
  - (Crs)Operator
  - Operator Views
  - Strided Maps
- Layer 2: extensions
  - BlockedCrsOperator

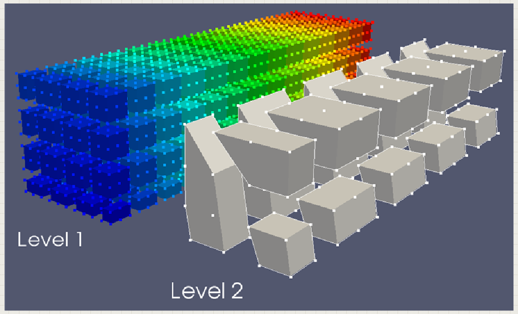


*A*lgebraic *M*ulti*G*rid methods

- construct a hierarchy of coarser representations
- use information from finest level problem A only
- apply smoothing methods for reducing high oscillatory error components on each level



Aggregation method



Transfer operators

- prolongation operator P
- restriction operator R
- define coarse levels

$$A_{\ell+1} = RA_{\ell}P$$

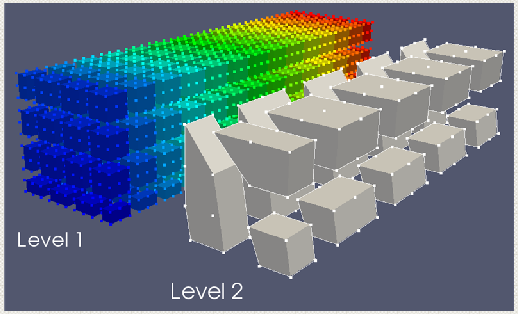
Level smoothers

- cheap smoothers for fine and intermedium levels
- direct solver on coarsest level

Status – MueLu Aggregation

- „standard“ aggregation method (with proc overlapping aggregates) ✓
- Simple coalescing (support for strided maps) ✓
- Other methods?

Aggregation method



Transfer operators

- prolongation operator P
- restriction operator R
- define coarse levels

$$A_{\ell+1} = R A_\ell P$$

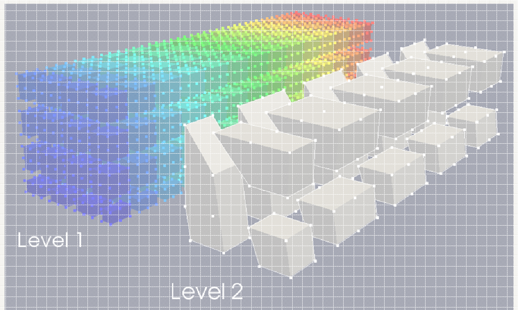
Level smoothers

- cheap smoothers for fine and intermedium levels
- direct solver on coarsest level

Status – MueLu Transfer operators

- – Nonsmoothed aggregation (tentative prolongator) ✓
- – Smoothed aggregation (SA-AMG) ✓
- – Petrov-Galerkin smoothed aggregation (PG-AMG) ✓
- – Energy-minimization methods (e.g. SchurComp) ✗
- – Segregated transfer operators for block systems (under construction)

Aggregation method



Transfer operators

- prolongation operator P
- restriction operator R
- define coarse levels

$$A_{\ell+1} = RA_\ell P$$

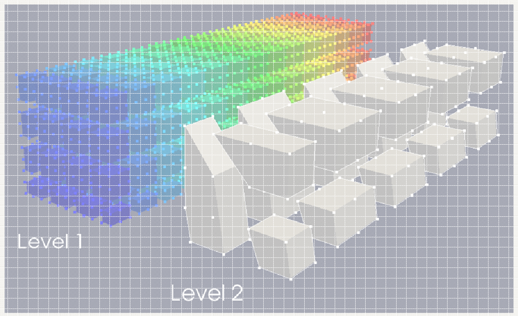
Level smoothers

- cheap smoothers for fine and intermedium levels
- direct solver on coarsest level

Status – MueLu Smoothers

- Smoother: Ifpack/Ifpack2 (Jacobi, GaussSeidel, ILU…) ✓
- Direct solver: Amesos/Amesos2 (KLU, Umfpack, SuperLu) ✓
- GaussSeidel (test implementation) ✓
- Block Smoothers
  - Block Gauss Seidel
  - Braess Sarazin
  - Teko interface ✗

Aggregation method



Transfer operators

- prolongation operator P
- restriction operator R
- define coarse levels

$$A_{\ell+1} = R A_\ell P$$

Level smoothers

- cheap smoothers for fine and intermedium levels
- direct solver on coarsest level

Status – MueLu Smoothers

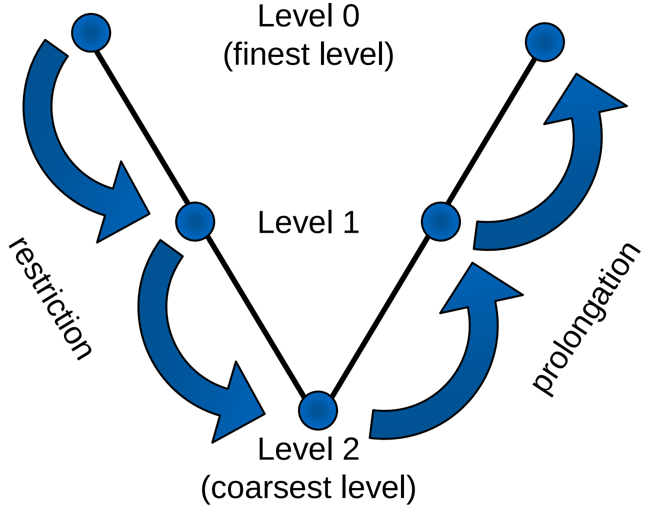
- Load balancing: Zoltan
- Usage as preconditioners
  - Adapter to Belos
  - Adapter to AztecOO
- Accessible via Stratimikos

to be continued…

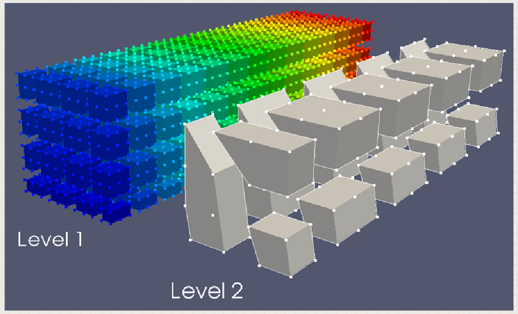
MueLu – OO design

*A*lgebraic *M*ulti*G*rid methods

- Construct a hierarchy of coarser representations
- Use information from finest level problem A only
- Apply smoothing methods for reducing high oscillatory error components on each level



Aggregation method



Transfer operators

- prolongation operator P
- restriction operator R
- define coarse levels

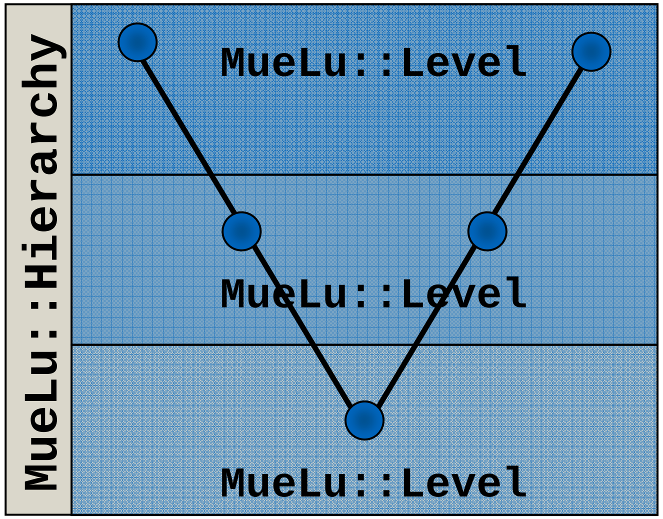
$$A_{\ell+1} = RA_\ell P$$

Level smoothers

- cheap smoothers for fine and intermediate levels
- direct solver on coarsest level

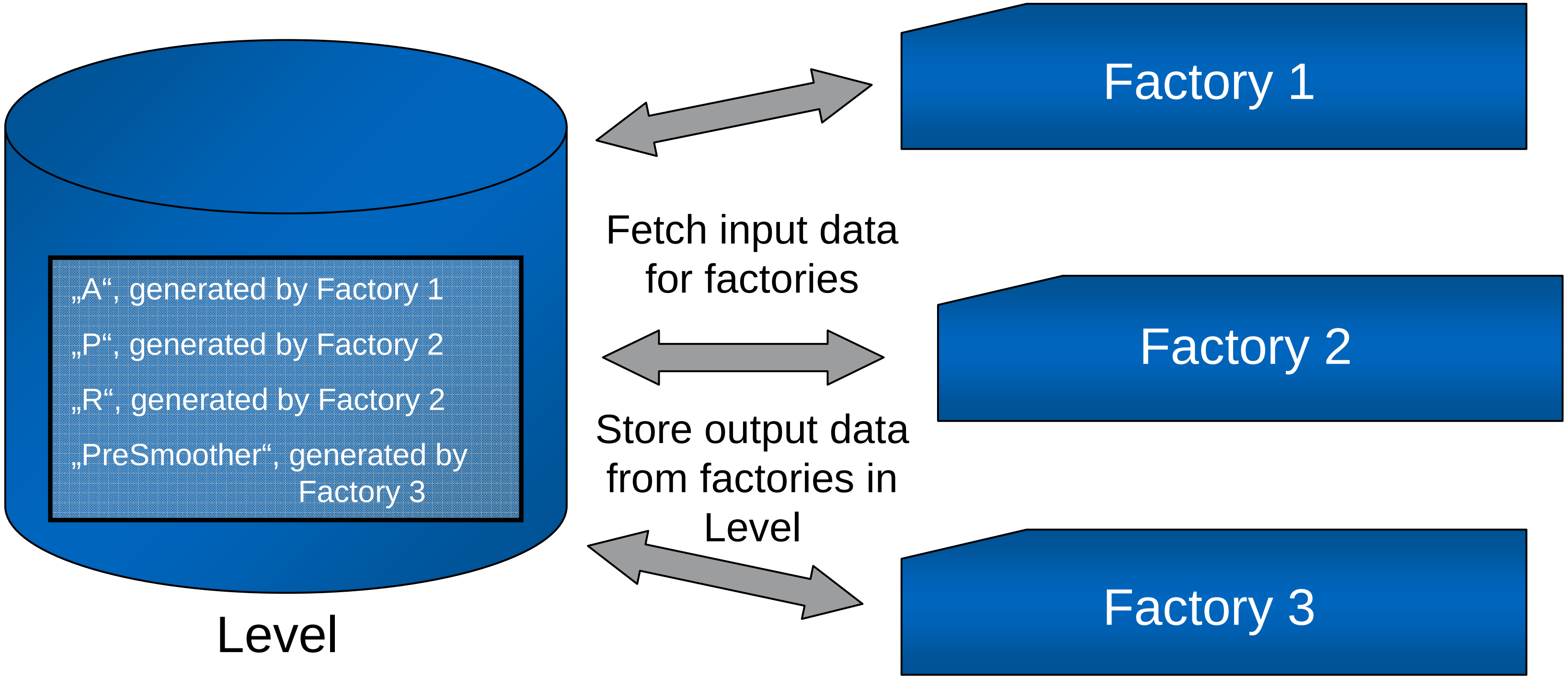
*A*lgebraic *M*ulti*G*rid methods – OO design

- Hierarchy object
  - Generates and stores multigrid levels
  - Provides multigrid cycles (e.g. V-cycle)
- Factory pattern
  - Factories generate components of multigrid algorithm
  - „FactoryManager“ manages dependencies between factories



Aggregation method	Transfer operators	Level smoothers
AggregationFactory Provide routines to build aggregates from graph of a matrix.	transfer operator factories Distinction between prolongation and restriction.	SmootherFactory Prototype concept: define a smoother prototype, clone a smoother prototype in the SmootherFactory

Design concept



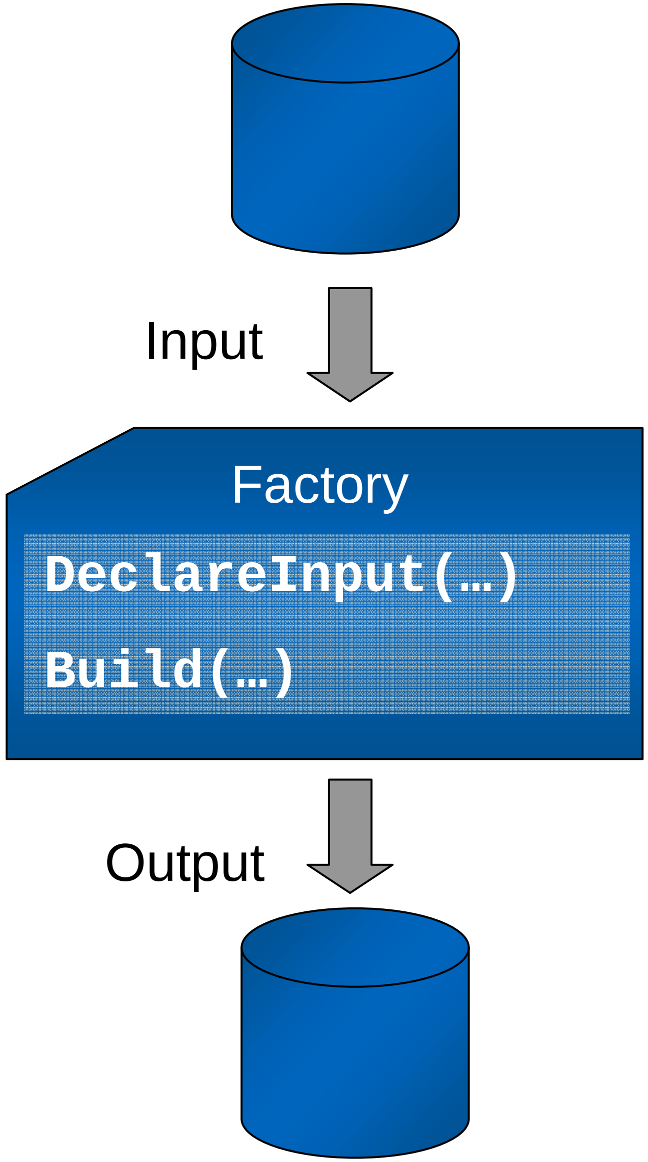
- Level: data storage
- Data uniquely determined by „variable name“ and „generating factory“
- Data automatically released/destroyed as soon as possible

- Factories generate data using information from Level data storage
- Store generated data in Level data storage
- Dependencies handled/resolved by FactoryManager

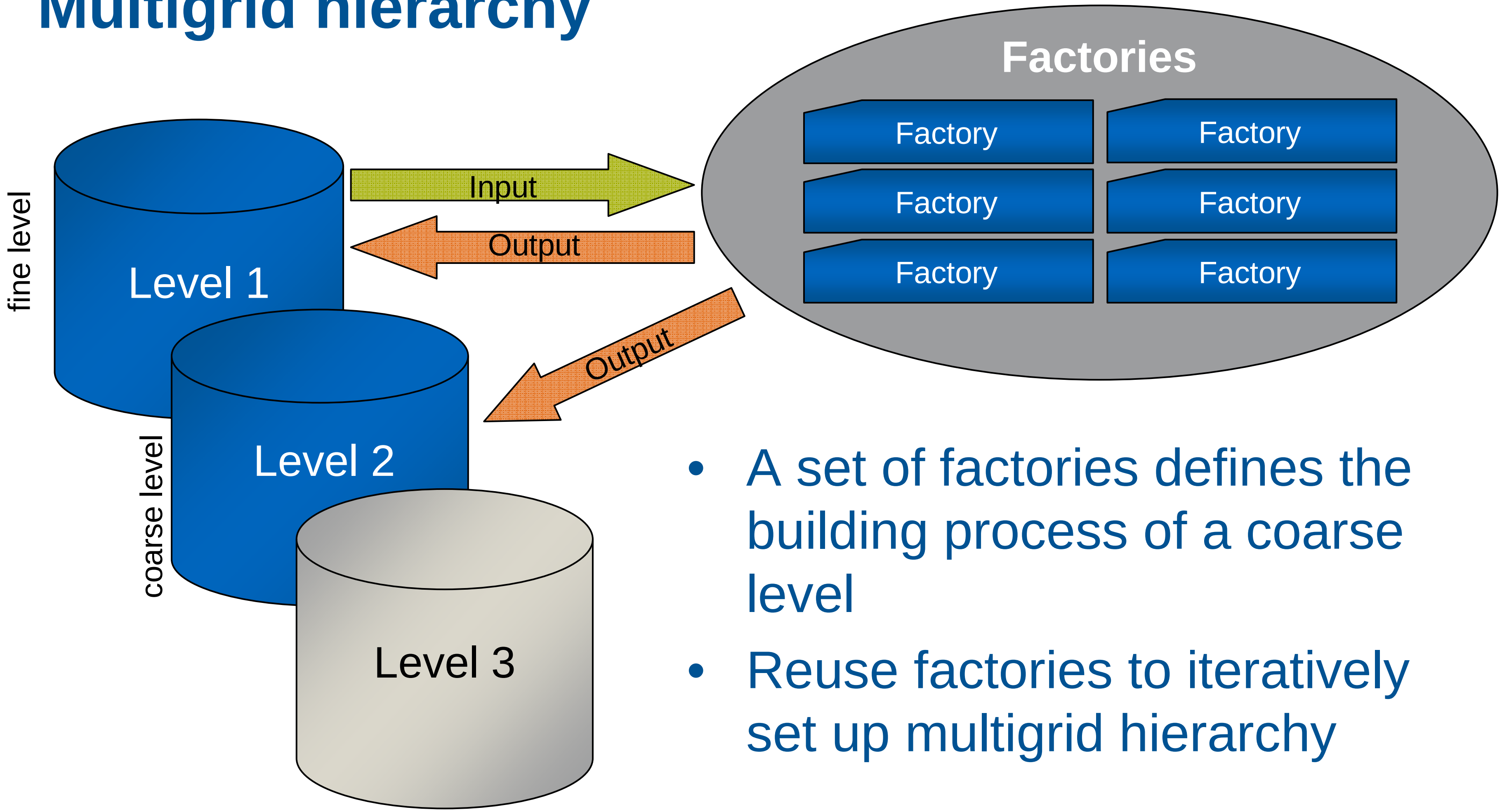
Factories

- Factory processes input data (from Level) and generates some output data (stored in Level)
- Distinction between
  - SingleLevelFactories: e.g. Level Smoothers, AggregationFactory…
  - TwoLevelFactories: e.g. transfer factories

    → output is stored on next coarser level
- Factory can generate more than one output variables (e.g. „Ptent“ and „Nullspace“)

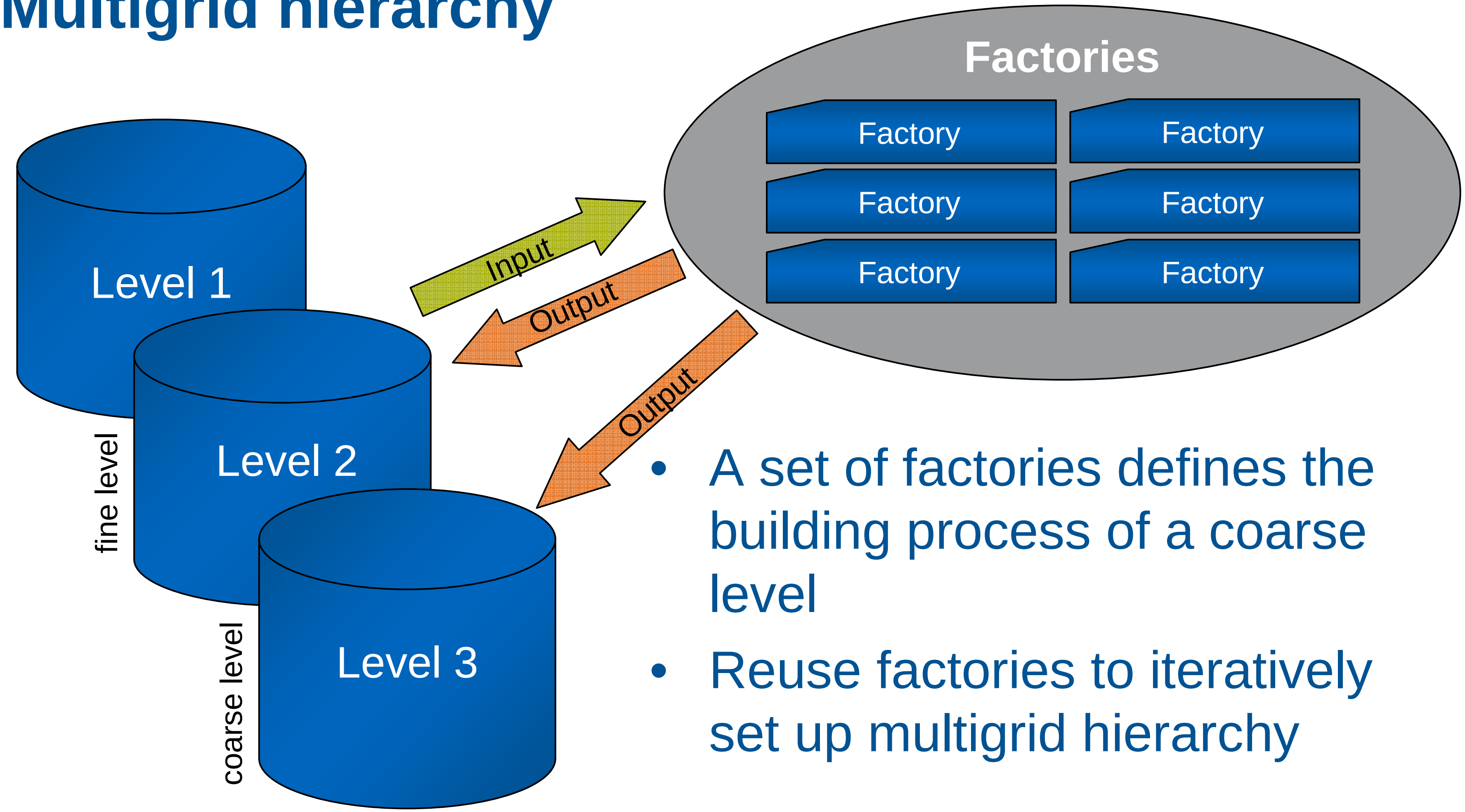


Multigrid hierarchy



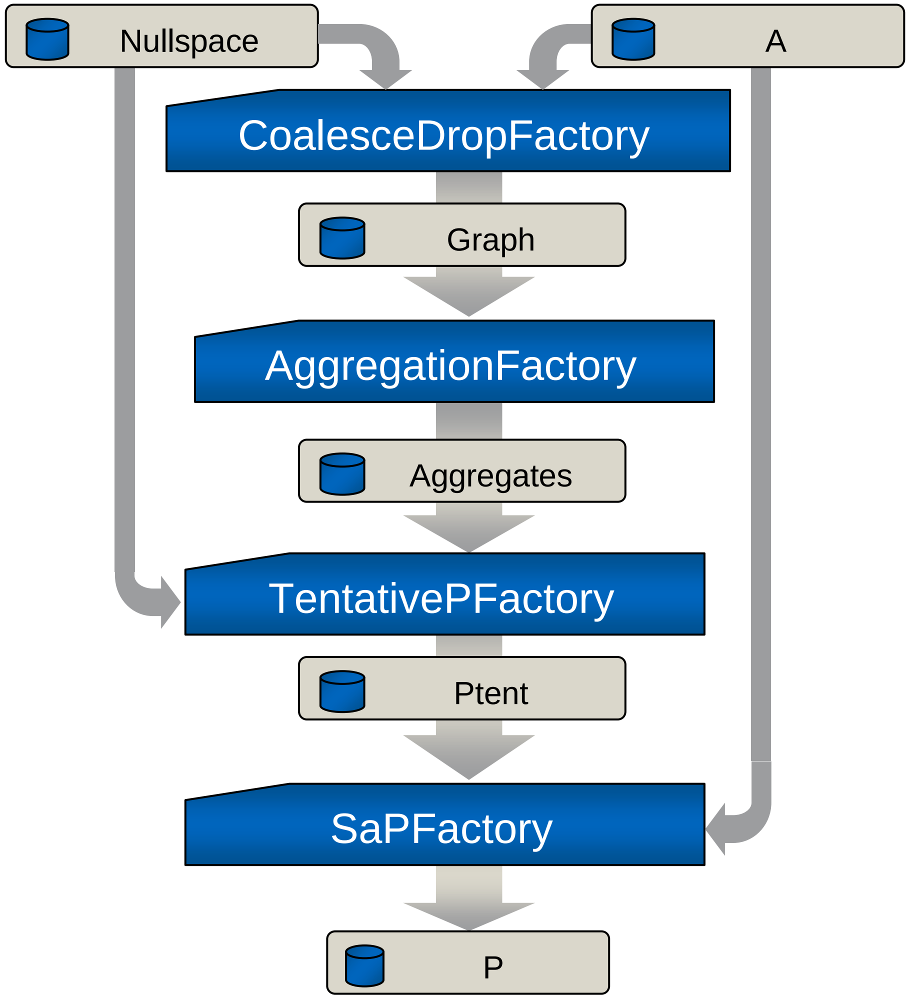
- A set of factories defines the building process of a coarse level
- Reuse factories to iteratively set up multigrid hierarchy

Multigrid hierarchy



- A set of factories defines the building process of a coarse level
- Reuse factories to iteratively set up multigrid hierarchy

Chaining factories – example: transfer operators



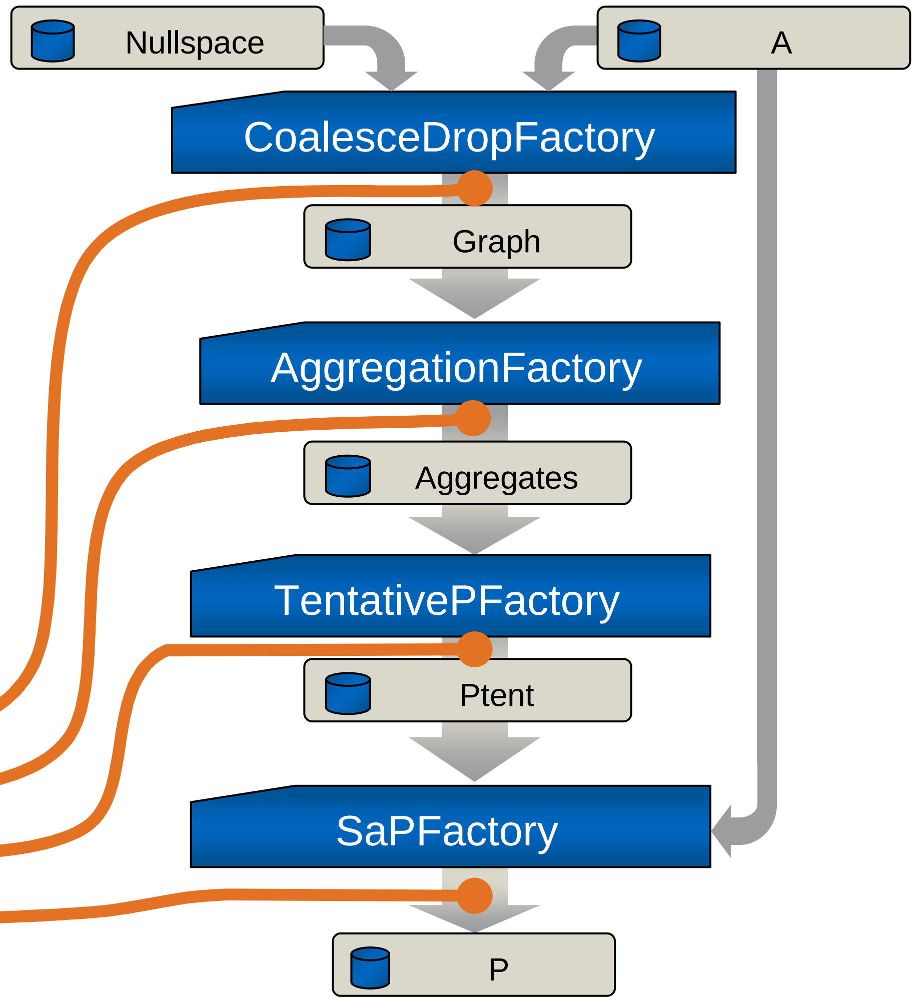

```
RCP<MueLu::Level> Finest = H->GetLevel();
Finest->Set("A", A);
Finest->Set("Nullspace", nullspace);

RCP<CoalesceDropFactory> dropFact =
          rcp(new CoalesceDropFactory());

RCP<UCAggregationFactory> UCAggFact =
      rcp(new UCAggregationFactory(dropFact));

RCP<TentativePFactory> PtentFact =
        rcp(new TentativePFactory(UCAggFact));

RCP<SaPFactory> SaPfact =
             rcp( new SaPFactory(PtentFact) );
```

Factory manager

- Holds default factories to be used during multigrid setup
- Can have one FactoryManager per level
- User can selectively specify alternatives
- The hierarchy set up process queries the FactoryManager for proper factory for each algorithmic component

FactoryManager

Variable	Default Factory
Graph	CoalesceDropFactory
Aggregates	AggregationFactory
Ptent	TentativePFactory
P	SaPFactory

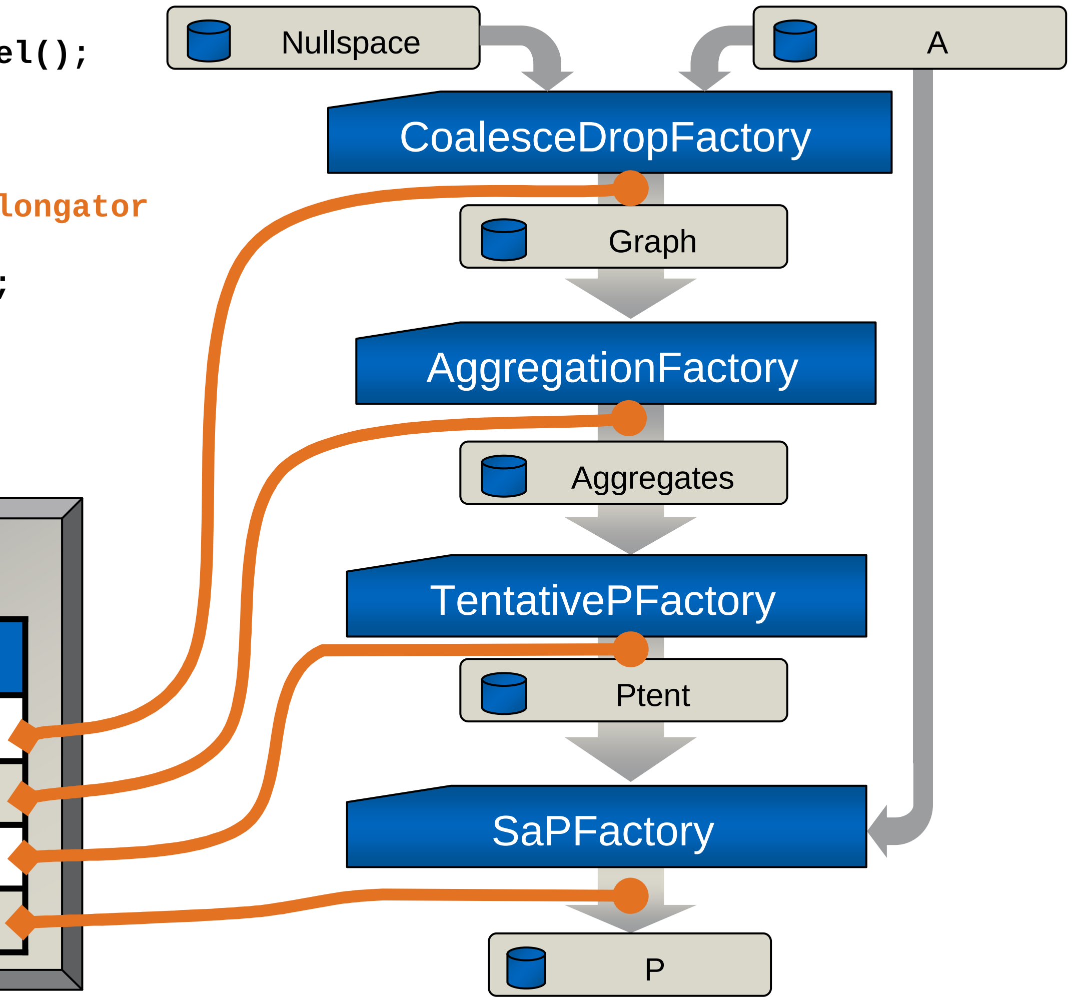


Factory manager

```
RCP<MueLu::Level> Finest = H->GetLevel();
Finest->Set("A", A);
Finest->Set("Nullspace", nullspace);

// generate smoothed aggregation prolongator
RCP<SaPFactory> SaPfact =
            rcp( new SaPFactory() );
```

FactoryManager

Variable	Default Factory
Graph	CoalesceDropFactory
Aggregates	AggregationFactory
Ptent	TentativePFactory
P	SaPFactory



MueLu – user interfaces

4

User interfaces

- MueLu can be customized using
  - XML input files
  - Parameter lists (key-value pairs, limited backwards compatibility for ML)
  - C++ interfaces
- New/casual users
  - Minimal interface
  - Sensible defaults provided automatically
- Advanced users
  - Can customize or replace any component

MueLu – simple user interface

```
Hierarchy H(fineA);  // generate hierarchy using fine level matrix
H.Setup();           // call multigrid setup: create hierarchy
H.Iterate(B,nits,X); // perform nits iterations with multigrid
                     // algorithm (V-cycle)
```

- Generate smoothed aggregation multigrid preconditioner
- Uses reasonable defaults:
  - Symmetric Gauss-Seidel (1 sweep, no damping) as pre-/postsmoother
  - Direct solver on coarsest level

Customizing the preconditioner

```
Hierarchy H(fineA);  // generate hierarchy using fine level matrix
RCP<TentativePFactory> PFact = rcp(new TentativePFactory());
FactoryManager M;    // generate a factory manager
M.SetFactory(„P“, PFact); // define tentative prolongator factory
                          // as default factory for generating P
H.Setup(M);          // call multigrid setup: create hierarchy
H.Iterate(B,nits,X); // perform nits iterations with multigrid
                     // algorithm (V-cycle)
```

- Use tentative (non-smoothed) prolongator instead of smoothed prolongation operator
- Register changes with FactoryManager and pass to Setup

Customizing the preconditioner

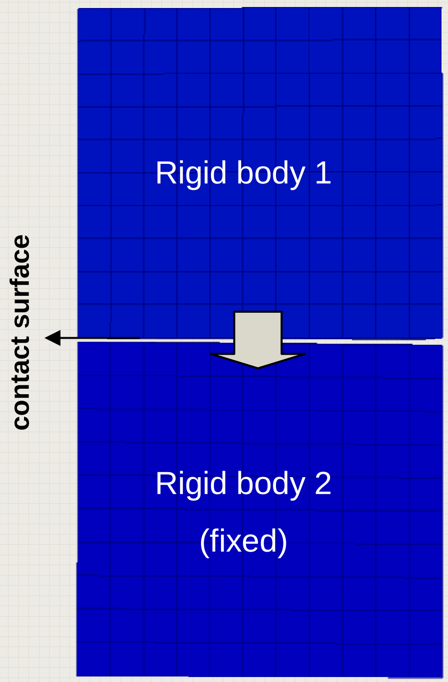
```
Hierarchy H(fineA);
Teuchos::ParamterList smootherParams;
smootherParams.set(„Chebyshev: degree“, 3);
RCP<SmootherPrototype> smooProto =
       rcp(new TrilinosSmoother(„Chebyshev“,smootherParams) );
RCP<SmootherFactory> SmooFact =
       rcp(new SmootherFactory(smooProto));
FactoryManager M;
M.SetFactory(„Smoother“, SmooFact);
H.Setup(M);
H.Iterate(B,nits,X);
```

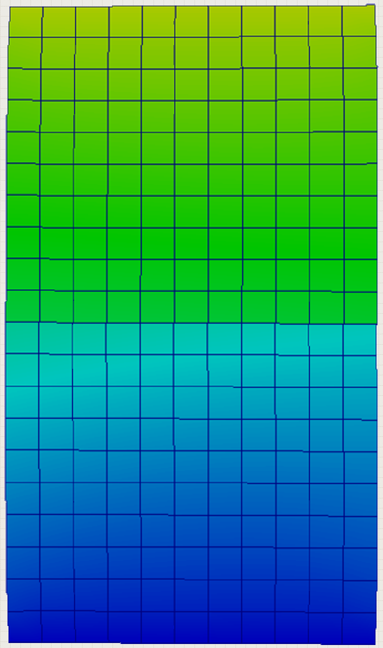
Use Chebyshev level smoother instead of SGS

MueLu – examples

Flexibility of MueLu framework

Example: Rigid body contact





- Two rigid-body contact problem
- Matrix structure before contact

$$K_S = \begin{pmatrix} K_{11} & \\ & K_{22} \end{pmatrix}$$

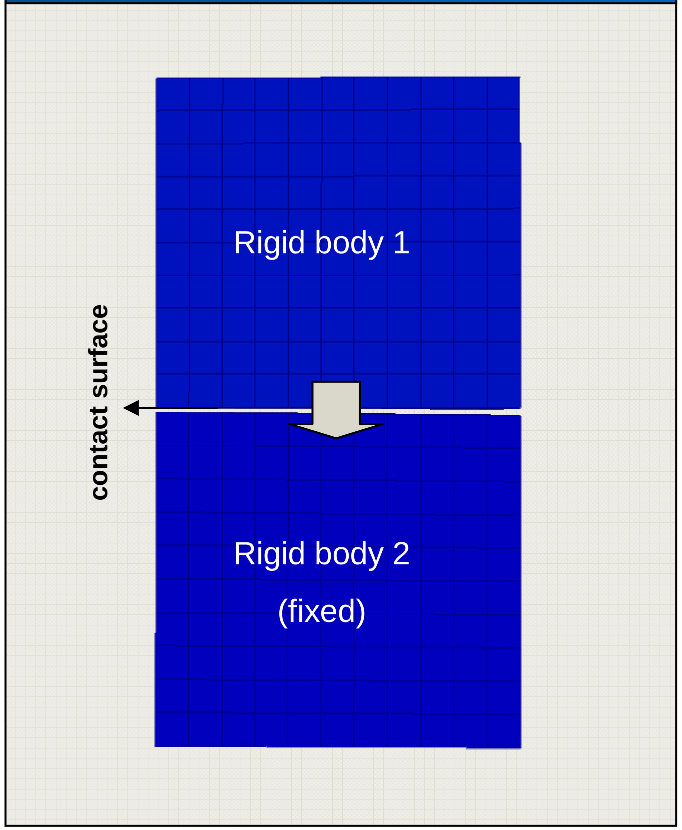
- Matrix structure after contact

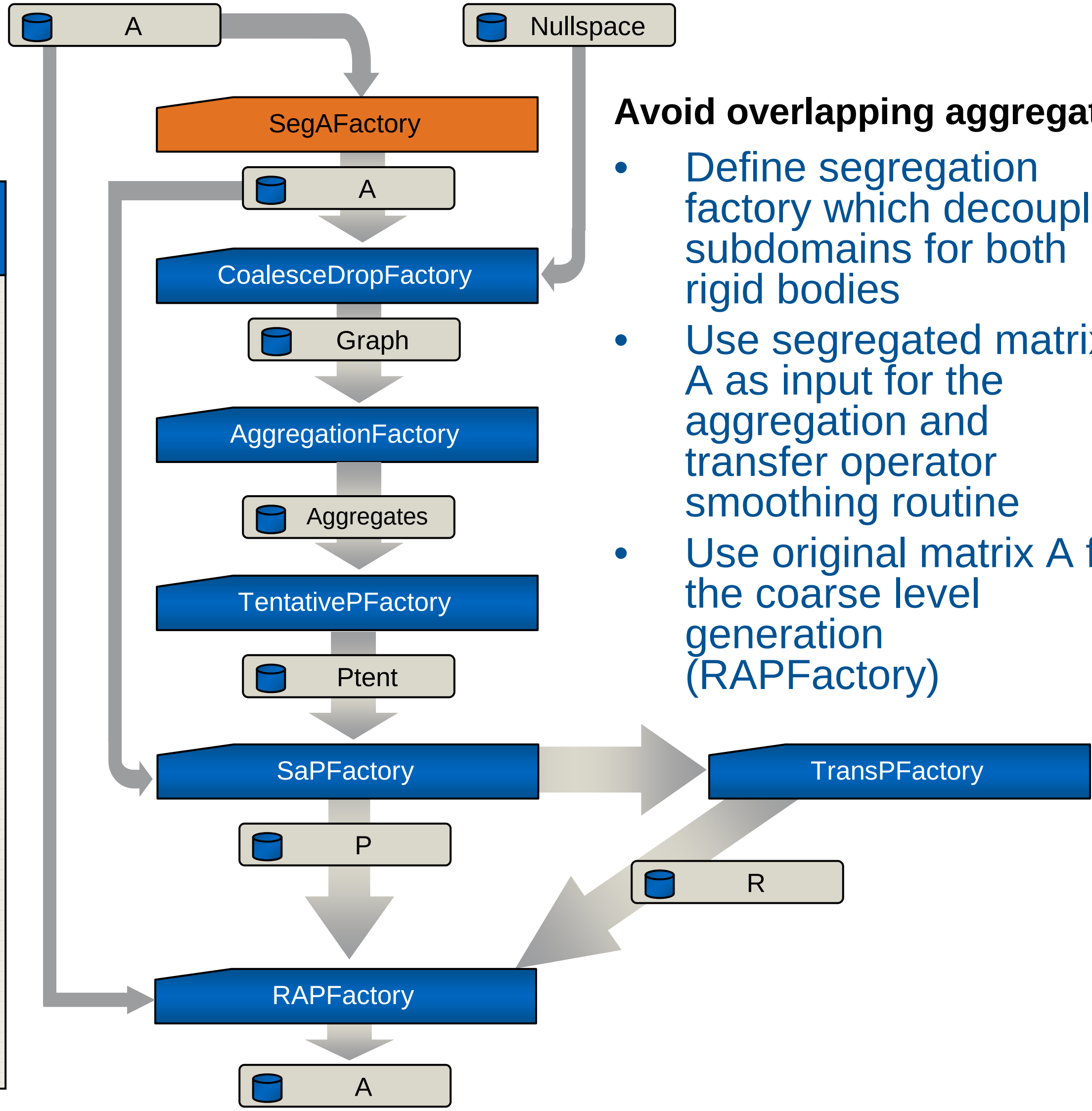
$$K_C = \begin{pmatrix} K_{11} + C_{11} & C_{21} \\ C_{12} & K_{22} + C_{22} \end{pmatrix}$$

→ Overlapping aggregates

Framework

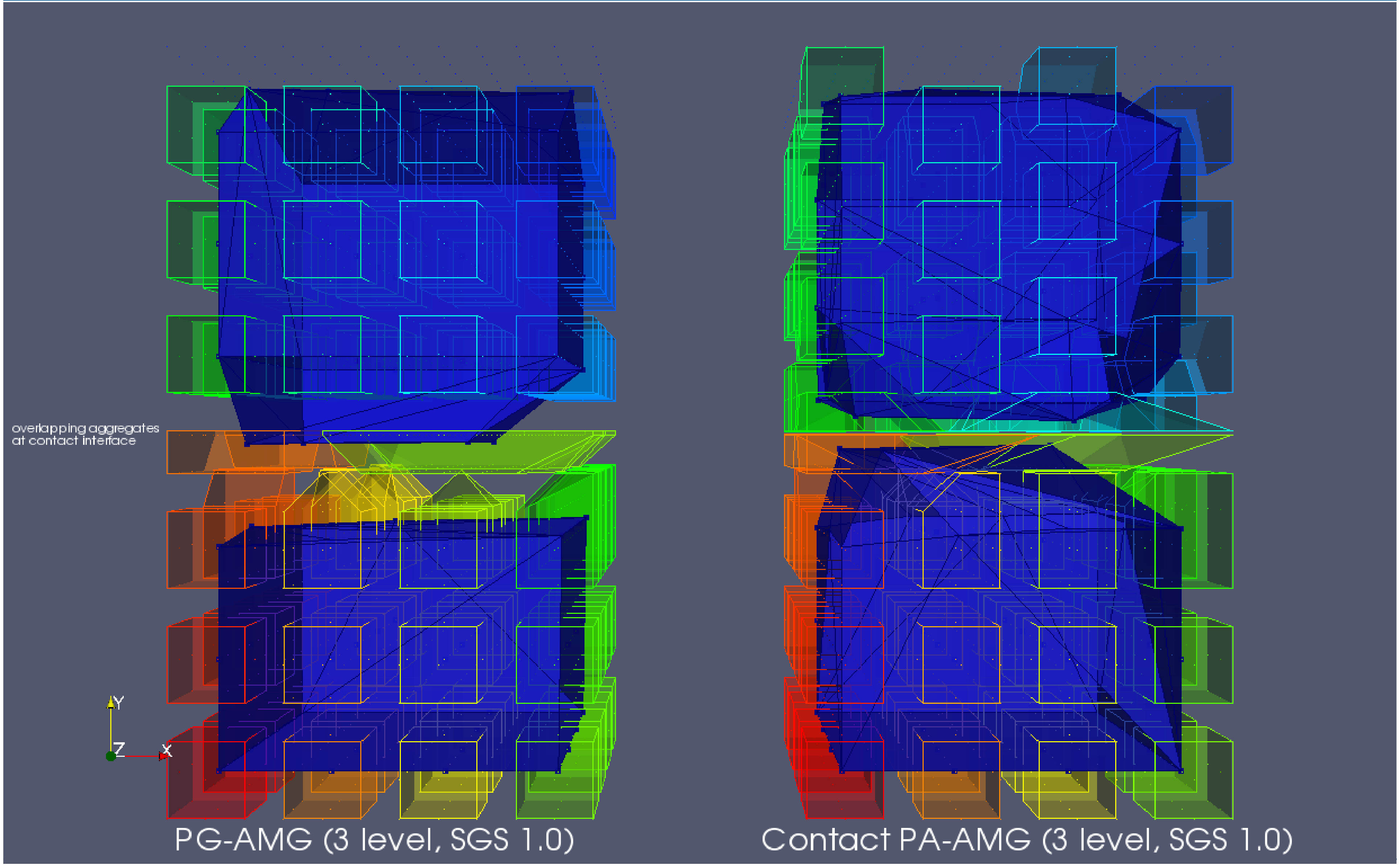
Example: Rigid body contact



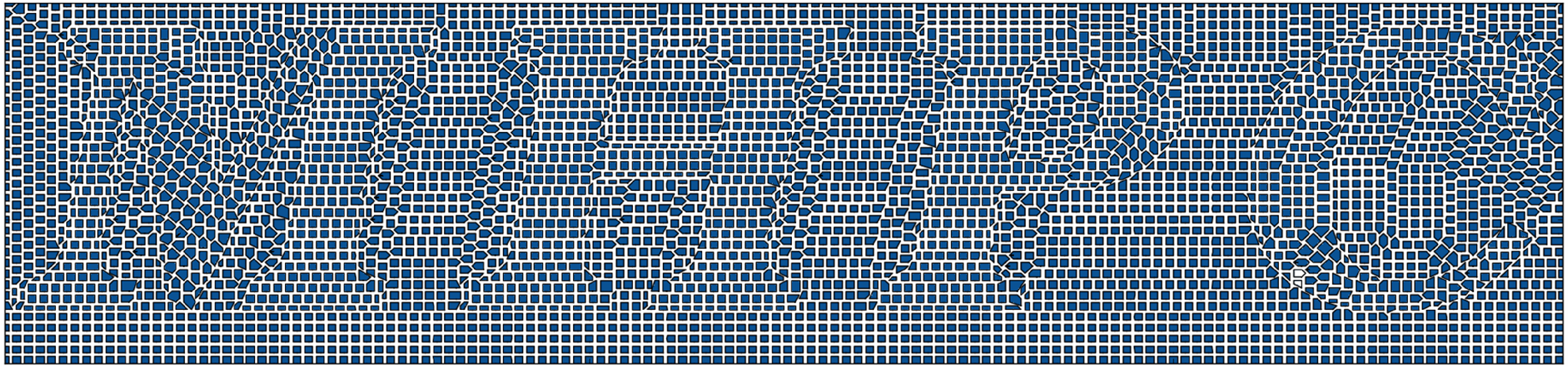


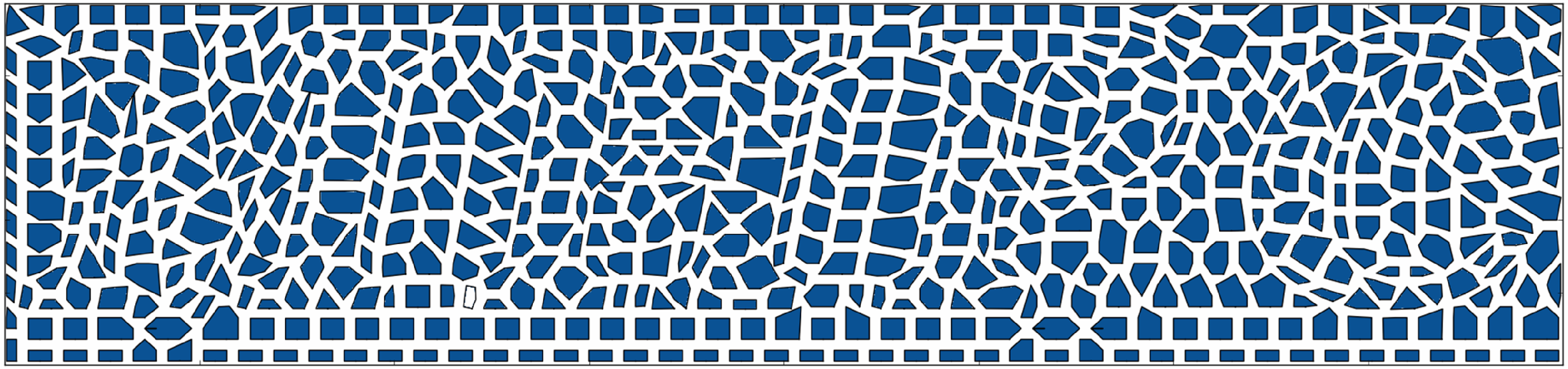
Avoid overlapping aggregates

- Define segregation factory which decouples subdomains for both rigid bodies
- Use segregated matrix A as input for the aggregation and transfer operator smoothing routine
- Use original matrix A for the coarse level generation (RAPFactory)

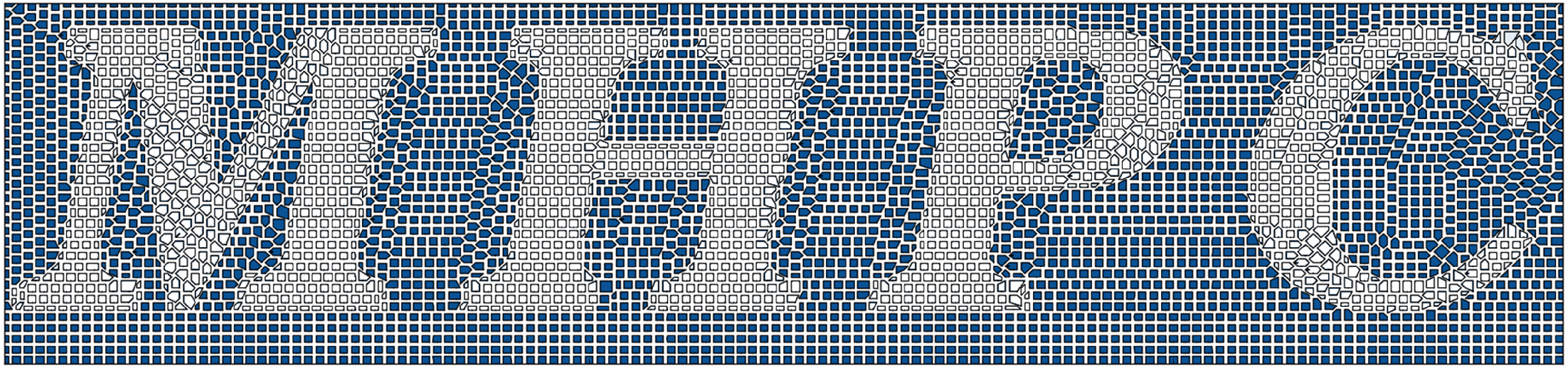

overlapping aggregates at contact interface
Y
Z
X
PG-AMG (3 level, SGS 1.0)
Contact PA-AMG (3 level, SGS 1.0)

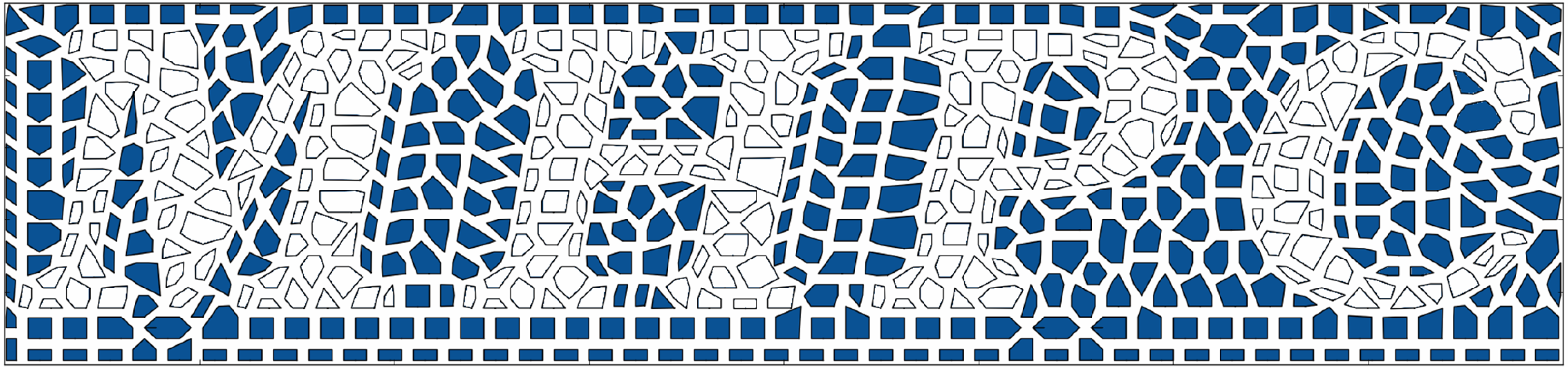
Example 2D





Example 2D – MHPC group logo





Factory concept

- Maximum of flexibility by „chaining“ factories
- Minimum code redundancy
- construction kit for AMG preconditioners
- Future: building blocks for multiphysics AMG preconditioners

Outlook

- Improve performance of some parts
- Implement energy minimization based transfer operators
- Extend Xpetra functionality

Questions?