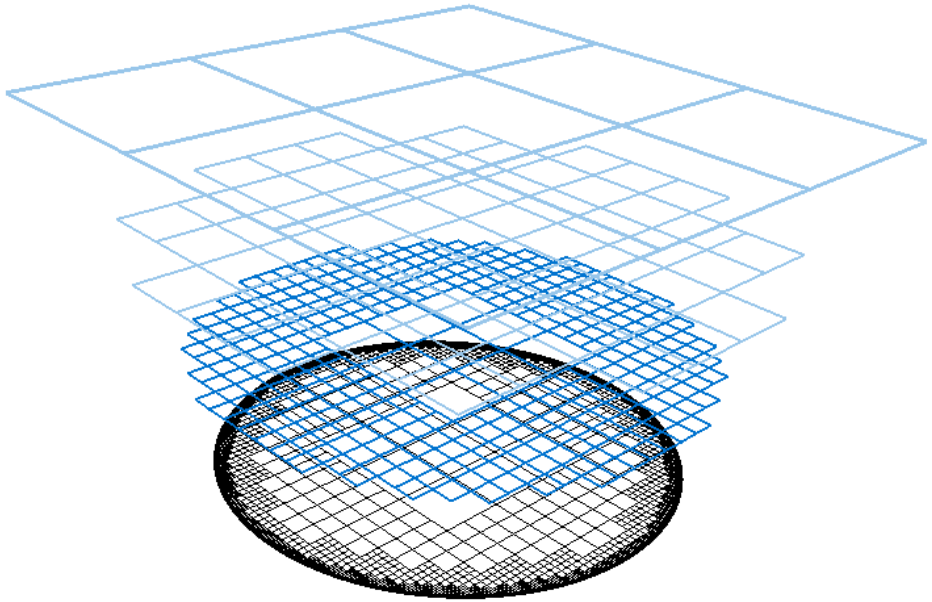
Fast Prototyping and Parallel Computations with Peano

Philipp Neumann

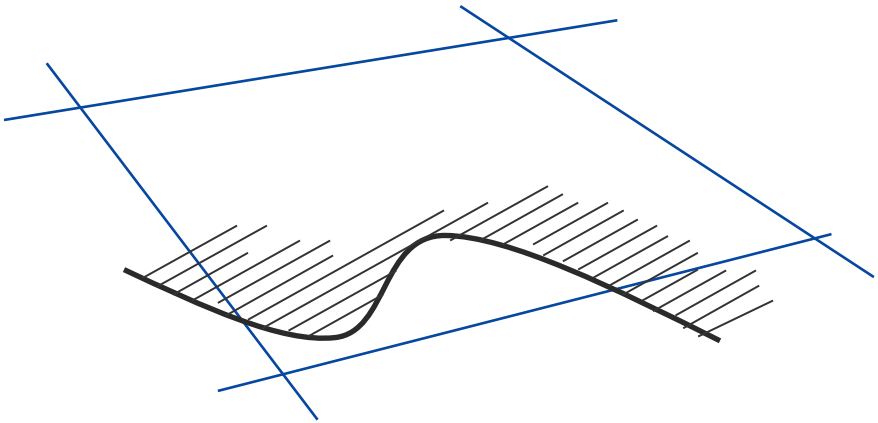
Outline

- Peano's principles
  - Adaptivity
  - Grid traversal
  - Architecture
- Prototyping
  - PeProt: Automatic code generation
  - DaStGen: Data structure generation
- Parallel Computations
  - Domain decomposition model
  - Shared memory parallelisation
  - Distributed memory parallelisation



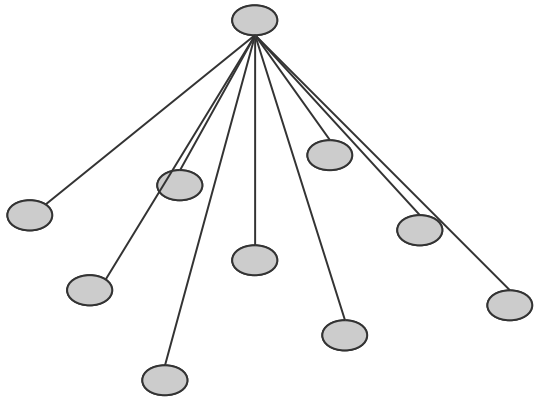
Adaptivity

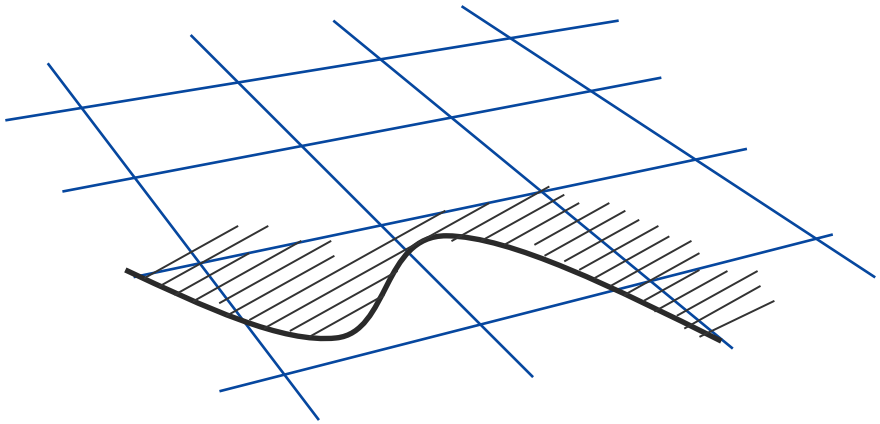
- Generalisation of Octrees
- Construction
  - Embed domain into unit square
  - Refine unit square: Domain is contained in central element
  - Continue recursively
  - Store only tree
- Remarks
  - Any dimension supported
  - k=3 for Peano space-filling curve
  - Low memory requirements
  - Multiple vertices at same position
  - Hanging nodes



Adaptivity

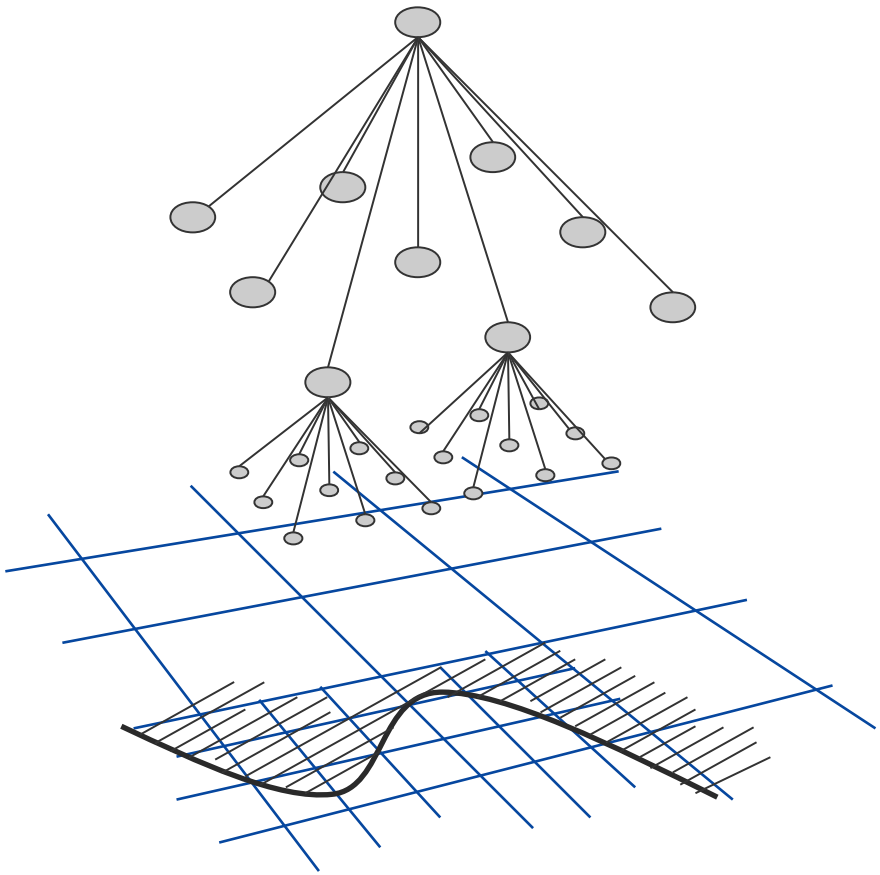
- Generalisation of Octrees
- Construction
  - Embed domain into unit square
  - Refine unit square: Domain is contained in central element
  - Continue recursively
  - Store only tree
- Remarks
  - Any dimension supported
  - k=3 for Peano space-filling curve
  - Low memory requirements
  - Multiple vertices at same position
  - Hanging nodes





Adaptivity

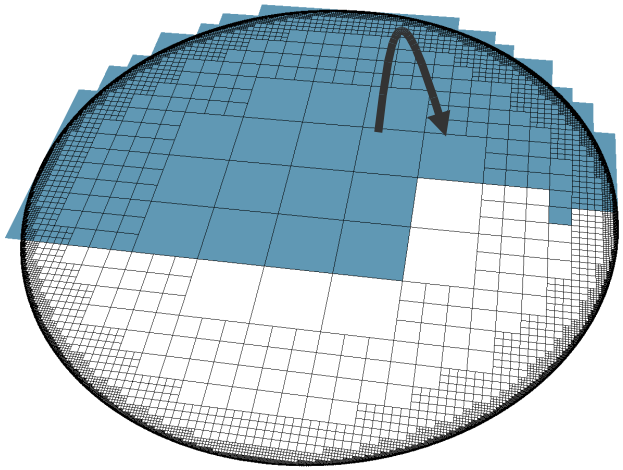
- Generalisation of Octrees
- Construction
  - Embed domain into unit square
  - Refine unit square: Domain is contained in central element
  - Continue recursively
  - Store only tree
- Remarks
  - Any dimension supported
  - k=3 for Peano space-filling curve
  - Low memory requirements
  - Multiple vertices at same position
  - Hanging nodes



Grid Traversal: Don't Call Us – We Call You

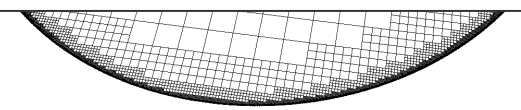
- Call-back with Traversal Events
  - How-to traverse is encapsulated
  - Where to go next is encapsulated
  - Each grid transition triggers event
  - User can plug into events
- Limitations
  - Never traverse subsets of grid
  - User can't control traversal direction or behaviour, but
  - complexity of traversal is hidden and
  - it might run in parallel.

Transition triggers an event, and this event is mapped to (several) event handles – the user-defined functions



Grid Traversal: Don't Call Us – We Call You

```
void MyMapping::touchVertexFirstTime(Vertex &vertex) {
  vertex.setVelocity(1.0);
  vertex.refine();
  […]
}

void MyMapping::enterCell(Vertex vertices[2^D],Cell &cell) {
  […]
}
```



or behaviour, but

- complexity of traversal is hidden and
- it might run in parallel.

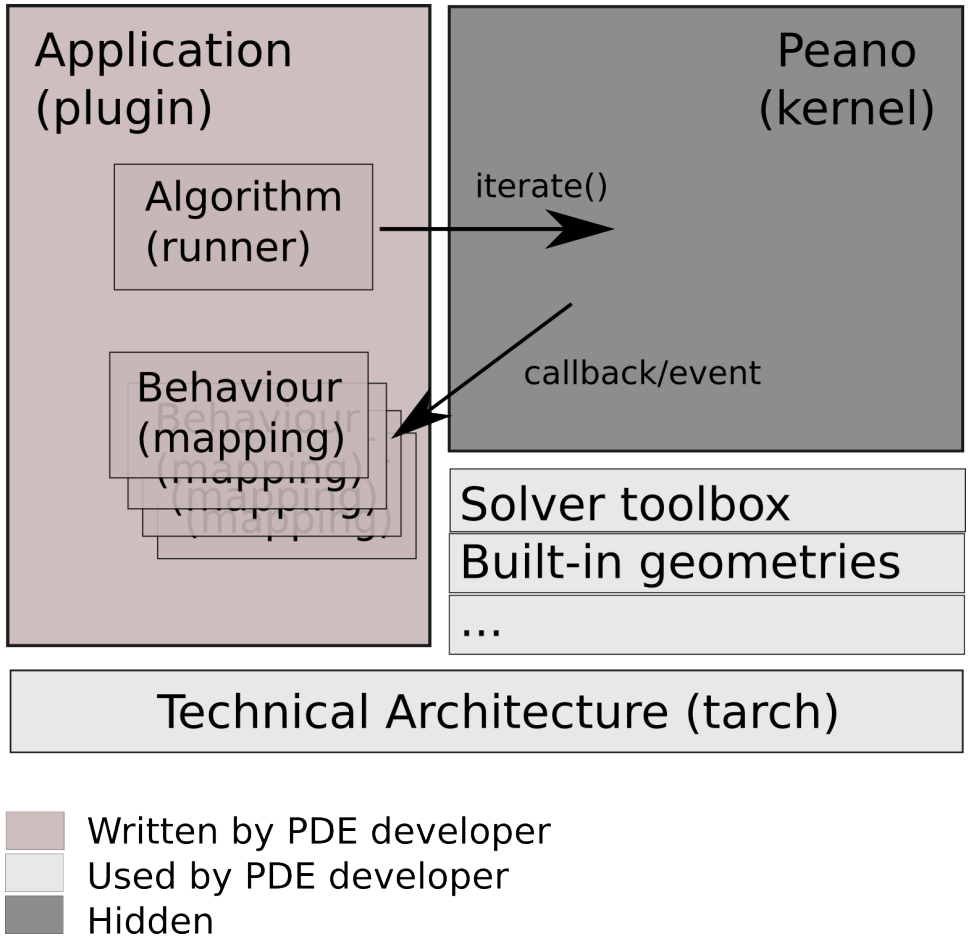
Grid Traversal: Don't Call Us – We Call You

```
void MyMapping::touchVertexFirstTime(Vertex &vertex) {
  vertex.setVelocity(1.0);
  vertex.refine();
  […]
}

void MyMapping::enterCell(Vertex vertices[2^D],Cell &cell) {
  […]
}
```

```
MyRunner::runAsMaster(){
  […]
  myAdapterRepository.switchToMyMappingAdapter();
  myAdapterRepository.iterate();

  […]
}
```

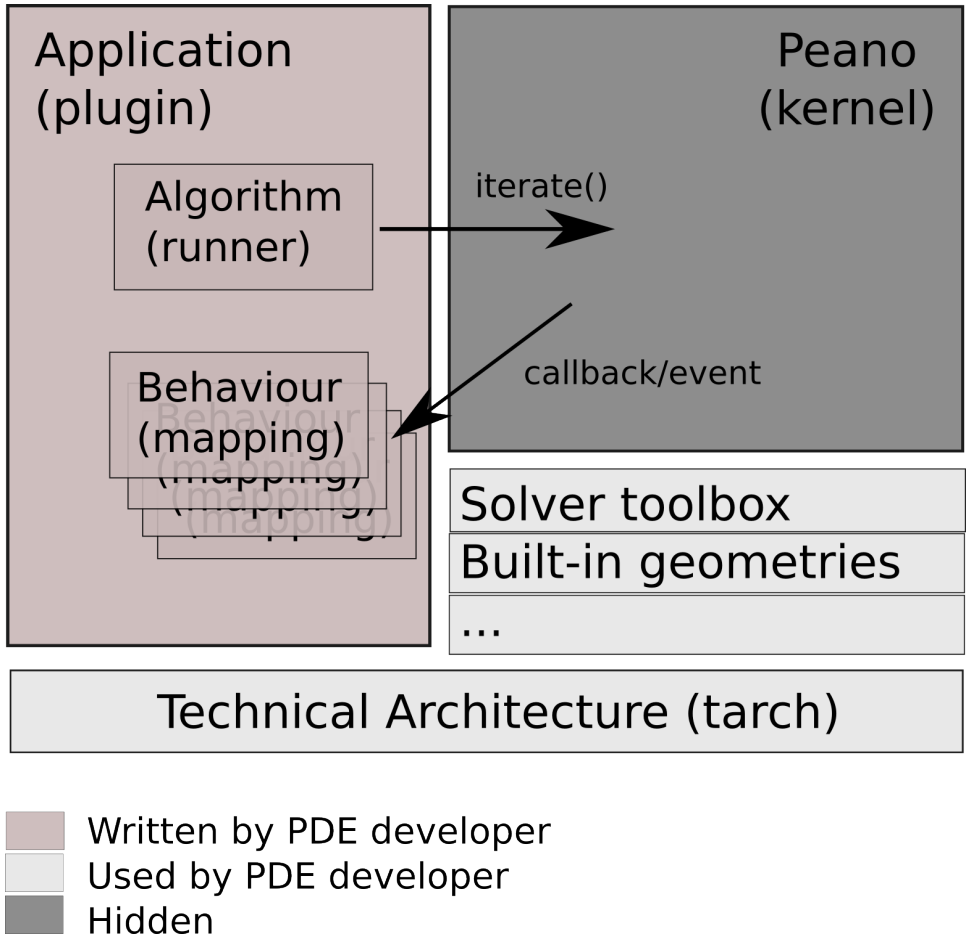
Architecture

- Philosophy
  - **Algorithm = sequence of phases** (implemented in runner)
  - Phase consists of
    - every computing node receives **global state**,
    - grid is traversed once (in parallel), and
    - global state is reduced.
- Repository
  - Repository = big box hiding Peano kernel and phase management
  - Single point of contact for algorithm (SPoC)
  - Two types of operations: iterate() and switchToPhaseXXX()



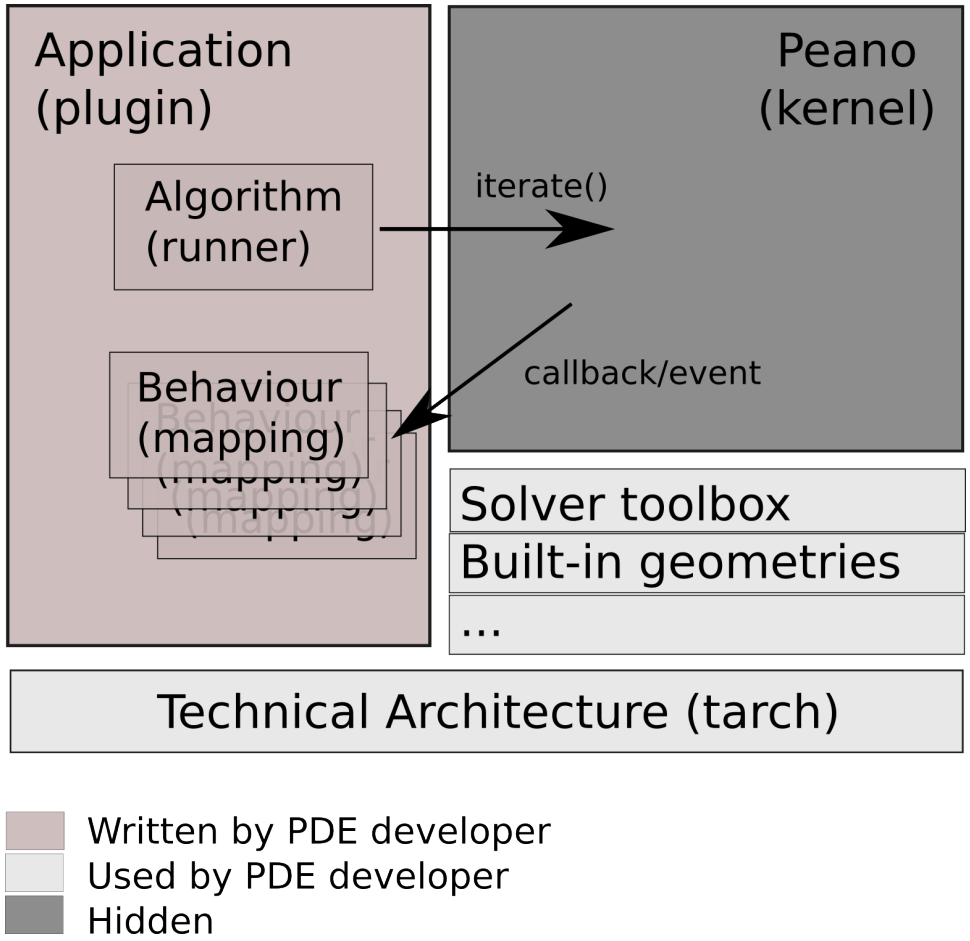
Architecture: What defines a Peano application?

- Vertex/ Cell data stored on the grid
- Global state object
- Event mappings
- Adapters = combinations/ merging of mappings



Architecture: What defines a Peano application?

- Vertex/ Cell data stored on the grid
- Global state object
- Event mappings
- Adapters = combinations/ merging of mappings



Same structure for each application

Peano Prototyping (PeProt): Automatic Code Generation

- PeProt scripting
  - **Define your data structures**
  - Define application-specific mappings
  - Define mapping combinations as adapters
  - Stencil evaluations: Simple definition and automatic code generation for these types of vertex-/ cell-events

PeProt: Automatic Code Generation – Data Structures

```
#include "peano/utils/Globals.h"
[...]

Packed-Type: int;
Constant: LB_BLOCK_NUMBER_OF_CELLS_ON_BLOCKBOUNDARY;
Constant: LB_PDFS_ON_BLOCKBOUNDARY;

class peano::applications::latticeboltzmann::
blocklatticeboltzmann::dastgen::BlockVertexRecord {
#ifdef Parallel
  persistent parallelise double _lbPdfDiff[LB_PDFS_ON_BLOCKBOUNDARY];
  [...]
#endif

  persistent parallelise int _vertexNumber;

  // for dynamic refinement
  enum DynamicRefinementState{ LB_REFINEMENT_DEFAULT,
    LB_REFINEMENT_IS_NEW_PERSISTENT_VERTEX, […] };

  persistent DynamicRefinementState _lbRefinementState;
};
```

PeProt: Automatic Code Generation – Data Structures

- PeProt scripting
  - **Define your data structures**
  - Define application-specific mappings
  - Define mapping combinations as adapters
  - Stencil evaluations: Simple definition and automatic code generation for these types of vertex-/ cell-events

```
component: blocklatticeboltzmann

configuration-tag: blocklatticeboltzmann
repository: BlockLatticeBoltzmannBatchJob
namespace:  peano::applications::latticeboltzmann::
            blocklatticeboltzmann

vertex:
  name: BlockVertex
  dastgen-file: BlockVertex.def


cell:
  name: BlockCell
  dastgen-file: BlockCell.def


state:
  name: BlockState
  dastgen-file: BlockState.def
```

PeProt: Automatic Code Generation – Mappings

- PeProt scripting
  - Define your data structures
  - **Define application-specific mappings**
  - **Define mapping combinations as adapters**
  - Stencil evaluations: Simple definition and automatic code generation for these types of vertex-/ cell-events

```
event-mapping:
  name: RegularBlockSolver

event-mapping:
  name: BlockVTKOutput

adapter:
  name: RegularBlockSolverAdapter
  merge-with-user-defined-mapping: RegularBlockSolver

adapter:
  name: BlockVTKOutputAdapter
  merge-with-user-defined-mapping: BlockVTKOutput

adapter:
  name: RegularBlockSolverAndVTKOutputAdapter
  merge-with-user-defined-mapping: BlockVTKOutput
  merge-with-user-defined-mapping: RegularBlockSolver
```

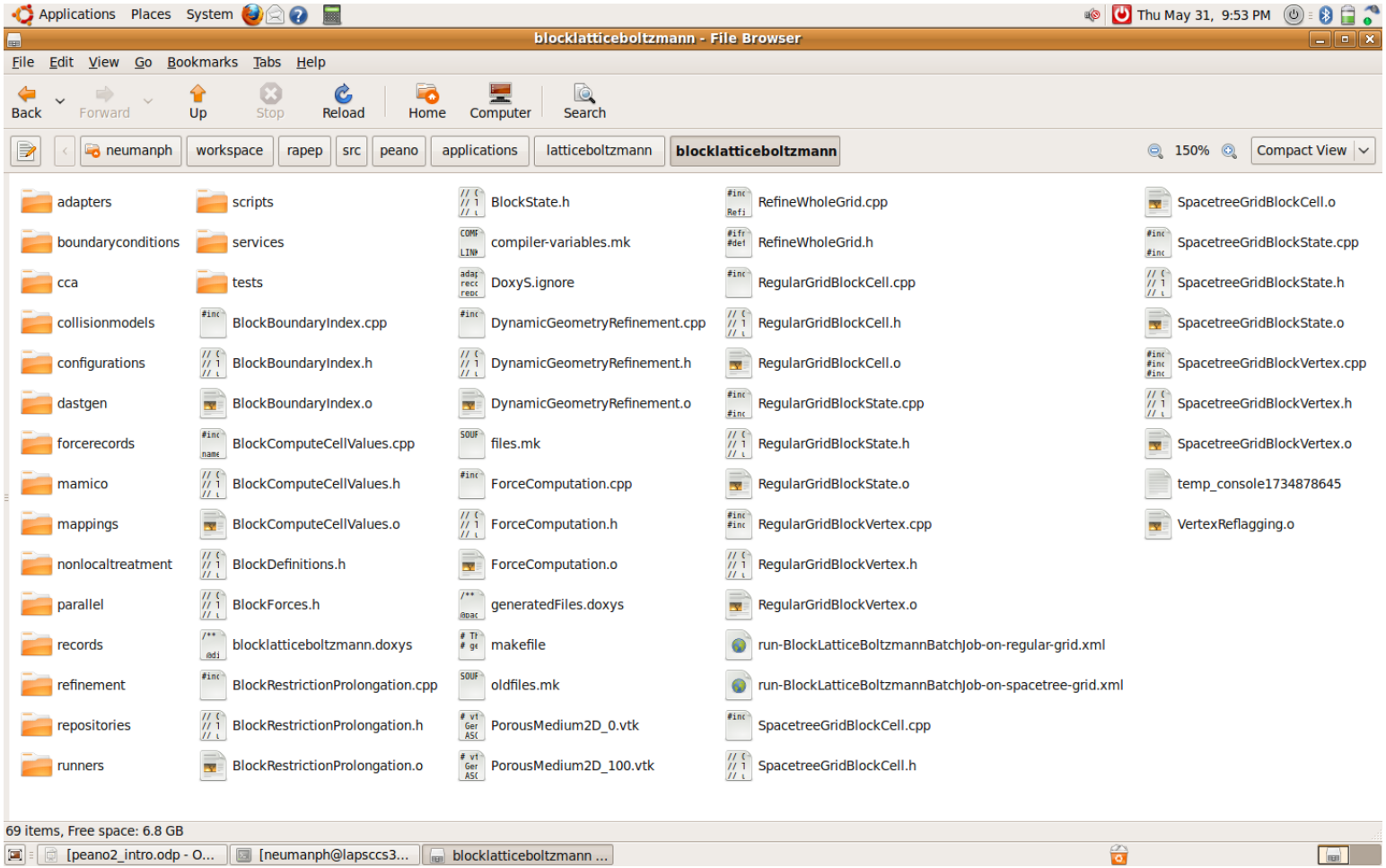
PeProt: Automatic Code Generation

- PeProt scripting
  - Define your data structures
  - **Define application-specific mappings**
  - **Define mapping combinations as adapters**
  - Stencil evaluations: Simple definition and automatic code generation for these types of vertex-/ cell-events

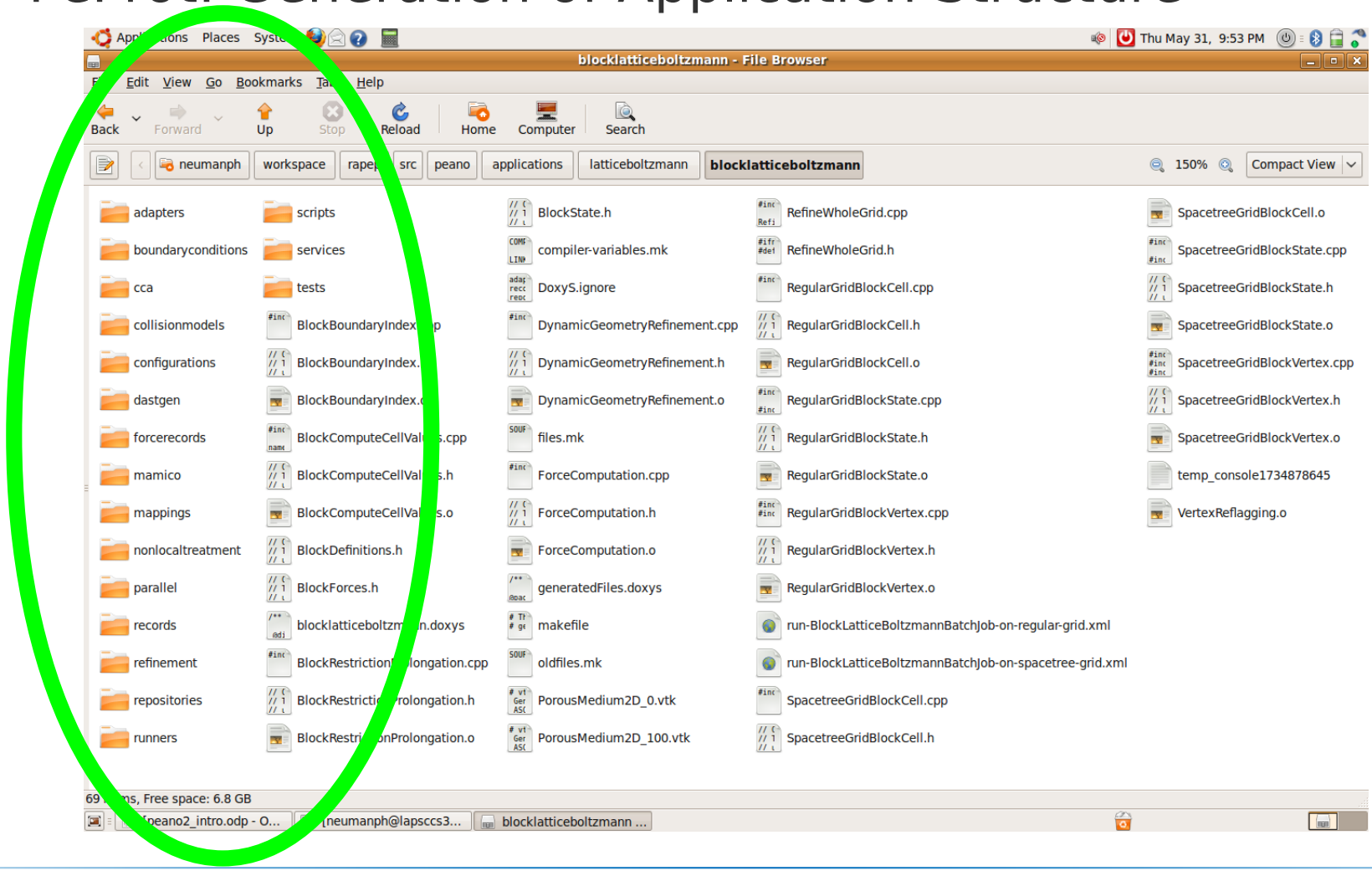
```
event-mapping:
  name: RegularBlockSolver

event-mapping:
  name: BlockVTKOutput

adapter:
  name: RegularBlockSolverAdapter
  merge-with-user-defined-mapping: RegularBlockSolver

adapter:
  name: BlockVTKOutputAdapter
  merge-with-user-defined-mapping: BlockVTKOutput

adapter:
  name: RegularBlockSolverAndVTKOutputAdapter
  merge-with-user-defined-mapping: BlockVTKOutput
  merge-with-user-defined-mapping: RegularBlockSolver
```

```
MYPATH="/home/neumanph/workspace/rapep/src/"

java -classpath PeProt.jar:DaStGen.jar de.tum.peano.peprot.PeProt
./SetupBlockLatticeBoltzmann $MYPATH ./templates $MYPATH/peano/kernel spacetreegrid
```

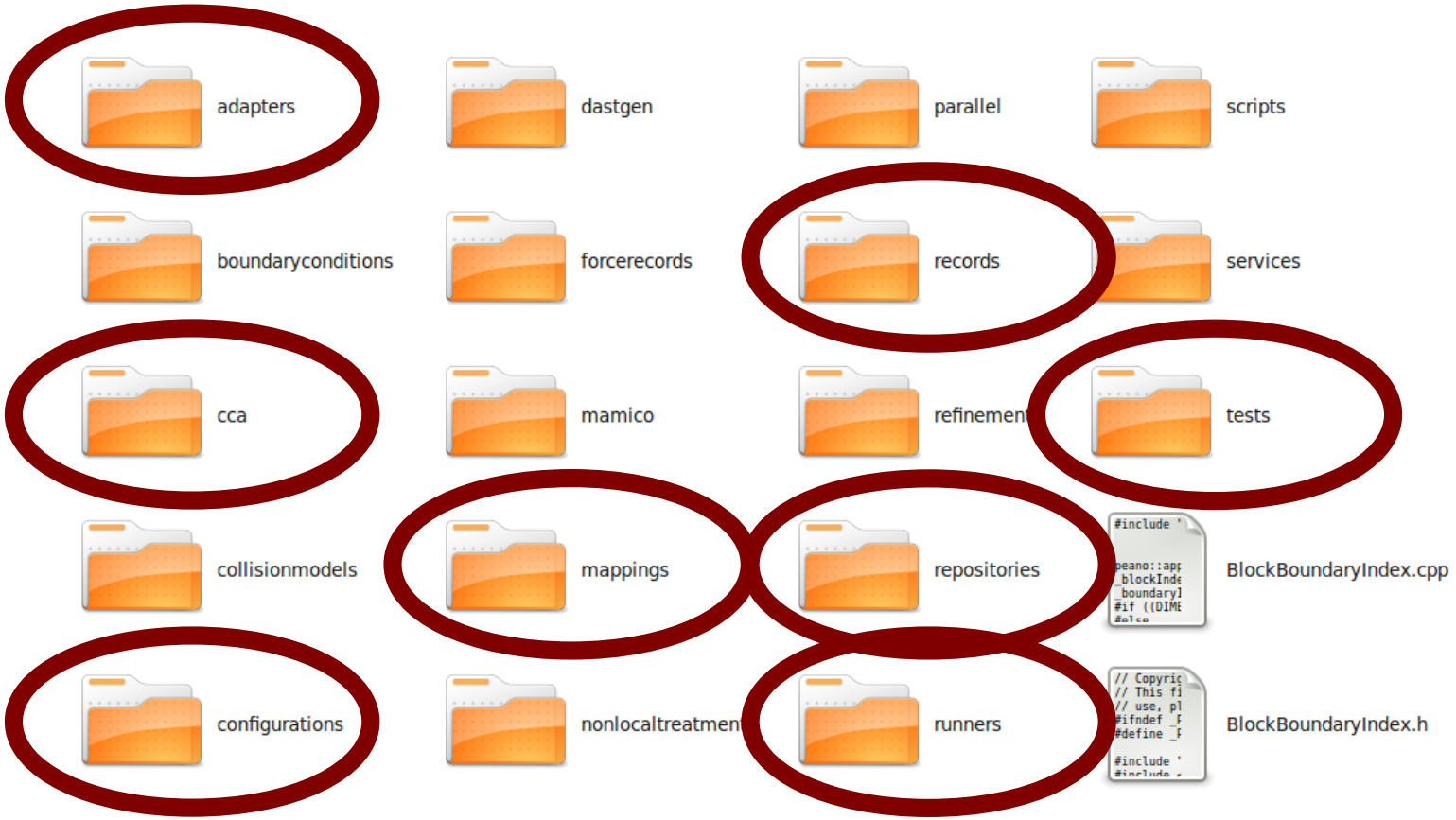
PeProt: Generation of Application Structure



PeProt: Generation of Application Structure

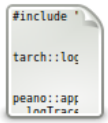


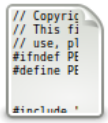
PeProt: Generation of Application Structure

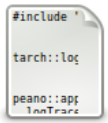


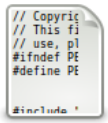
PeProt: Generation of Mapping Templates

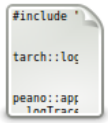
PeProt: Generation of Mapping Templates

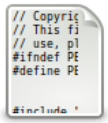

SpacetreeGrid2InitialiseSpacetreeGrid.cpp

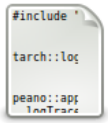

SpacetreeGrid2InitialiseSpacetreeGrid.h

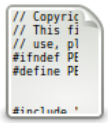

SpacetreeGrid2ReceiveInformationFromMaMiCo.cpp

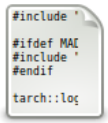

SpacetreeGrid2ReceiveInformationFromMaMiCo.h

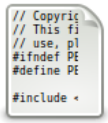

SpacetreeGrid2RefineWholeGrid.cpp

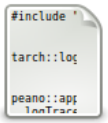

SpacetreeGrid2RefineWholeGrid.h

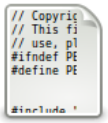

SpacetreeGrid2Reflag4StaticGeometryModification.cpp


SpacetreeGrid2Reflag4StaticGeometryModification.h

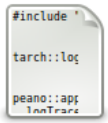

SpacetreeGrid2RegularBlockSolver.cpp

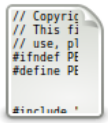

SpacetreeGrid2RegularBlockSolver.h

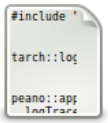

SpacetreeGrid2SendInformation2MaMiCo.cpp

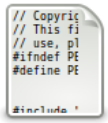

SpacetreeGrid2SendInformation2MaMiCo.h

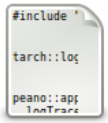
Developer's Task: Fill the Templates

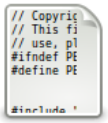

SpacetreeGrid2InitialiseSpacetreeGrid.cpp


SpacetreeGrid2RefineWholeGrid.h

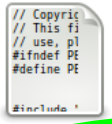

SpacetreeGrid2SendInformation2MaMiCo.cpp

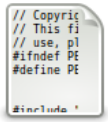

SpacetreeGrid2InitialiseSpacetreeGrid.h


SpacetreeGrid2Reflag4StaticGeometryModification.cpp

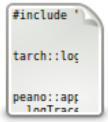

SpacetreeGrid2SendInformation2MaMiCo.h

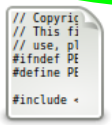

SpacetreeGrid2ReceiveInformationFromMaMiCo.cpp


SpacetreeGrid2Reflag4StaticGeometryModification.h


SpacetreeGrid2ReceiveInformationFromMaMiCo.h


SpacetreeGrid2RegularBlockSolver.cpp


SpacetreeGrid2RefineWholeGrid.cpp


SpacetreeGrid2RegularBlockSolver.h

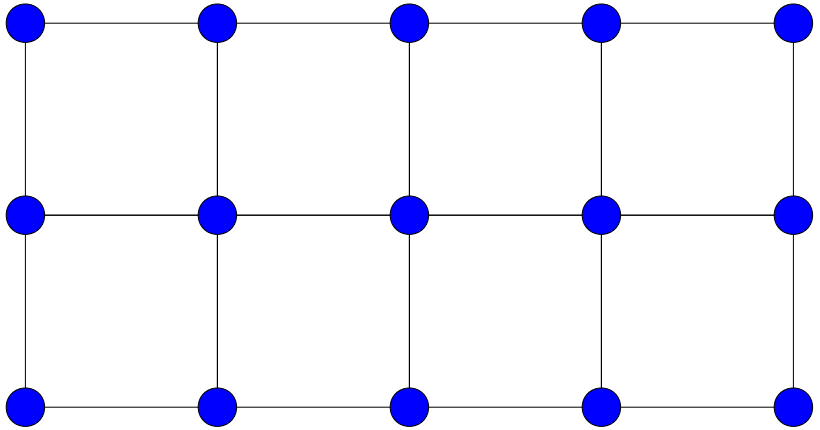
Developer's Task: Fill the Templates

```
logDebug("touchVertexFirstTime()","Do collision");
_blockCollisionModelManager->collide(level,inner,density,velocity,pdf,hasMinusOneEntries);

// apply external forces
peano::applications::latticeboltzmann::blocklatticeboltzmann::services::ExternalForceService::
getInstance().applyForce(level,*_multiLevelSimData.get(),inner,pdf,hasMinusOneEntries);

// TODO check, if hasMinusOneEntries is needed in force computation as well
logDebug("touchVertexFirstTime()", "Store post-collision pdfs at moving-obstacle boundaries");
_forceComputation->storePostCollisionPdfs(boundaryDataBuffer,boundaryDataIndex,pdf);

// if it is not the coarsest level, do prolongation if needed
if (level != _currentLevel){

  _blockRestrictionProlongation.prolongateToFineLevel(
    fineGridPositionOfVertex,coarsePdf,pdf,density,velocity,hasMinusOneEntries
  );

  // if the vertex is a new created vertex, remove new persistent flag
  if (fineGridVertex.isNewPersistentVertex()){
    fineGridVertex.switchToDefault();
  }
}

logDebug("touchVertexFirstTime()","Treat boundary after collision");
_blockBoundaryConditionManager->treatBoundaryAfterCollision(
  level,boundaryDataBuffer,boundaryDataIndex,density,velocity,pdf
);
```

Putting It All Together: The Application's Runner

```
while(time < t_end){
  repository.switchToControlTimeStep();                repository.iterate();
  repository.switchToSetVelocitiesBoundary();          repository.iterate();
  repository.switchToSetScenarioBoundary();            repository.iterate();
  repository.switchToComputeVelocitiesDerivatives();   repository.iterate();
  repository.switchToComputeRightHandSide();           repository.iterate();

  runLatticeBoltzmannSimulation(repository);

  int it = 0;
  repository.getSpacetreeGridState().setResidual(1.0+eps);

  while((it<itermax)&& repository.getSpacetreeGridState().getResidual()>eps)){
    repository.switchToSORStep();                      repository.iterate();
    repository.switchToComputeResidualNormAndSetPressureBoundary();
    repository.iterate();
    it++;
  }

  repository.switchToComputeVelocities();              repository.iterate();
  repository.switchToMoveParticles();                  repository.iterate();

  time += repository.getSpacetreeGridState().getTimeStepSize();
  repository.getSpacetreeGridState().setCurrentTime(time);
  logInfo("runAsMaster()","time t: " << "#(solver iterations): " << it);
}
```

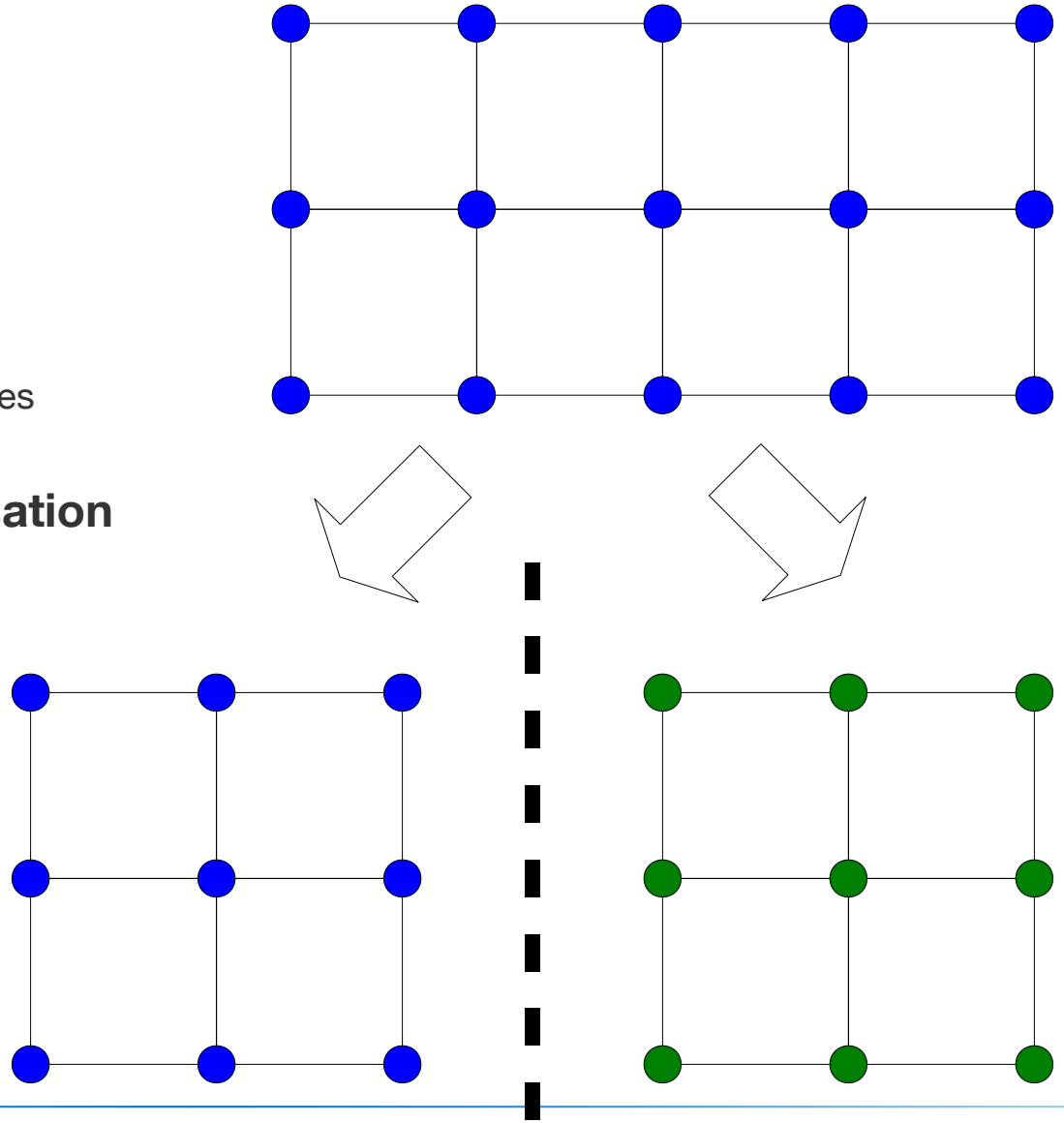
Parallelisation



- Shared memory parallelisation
  - OpenMP
  - Intel threading building blocks
  - Patch-based acceleration techniques
  - Autotuning mechanisms
- **Distributed memory parallelisation**
  - Domain decomposition

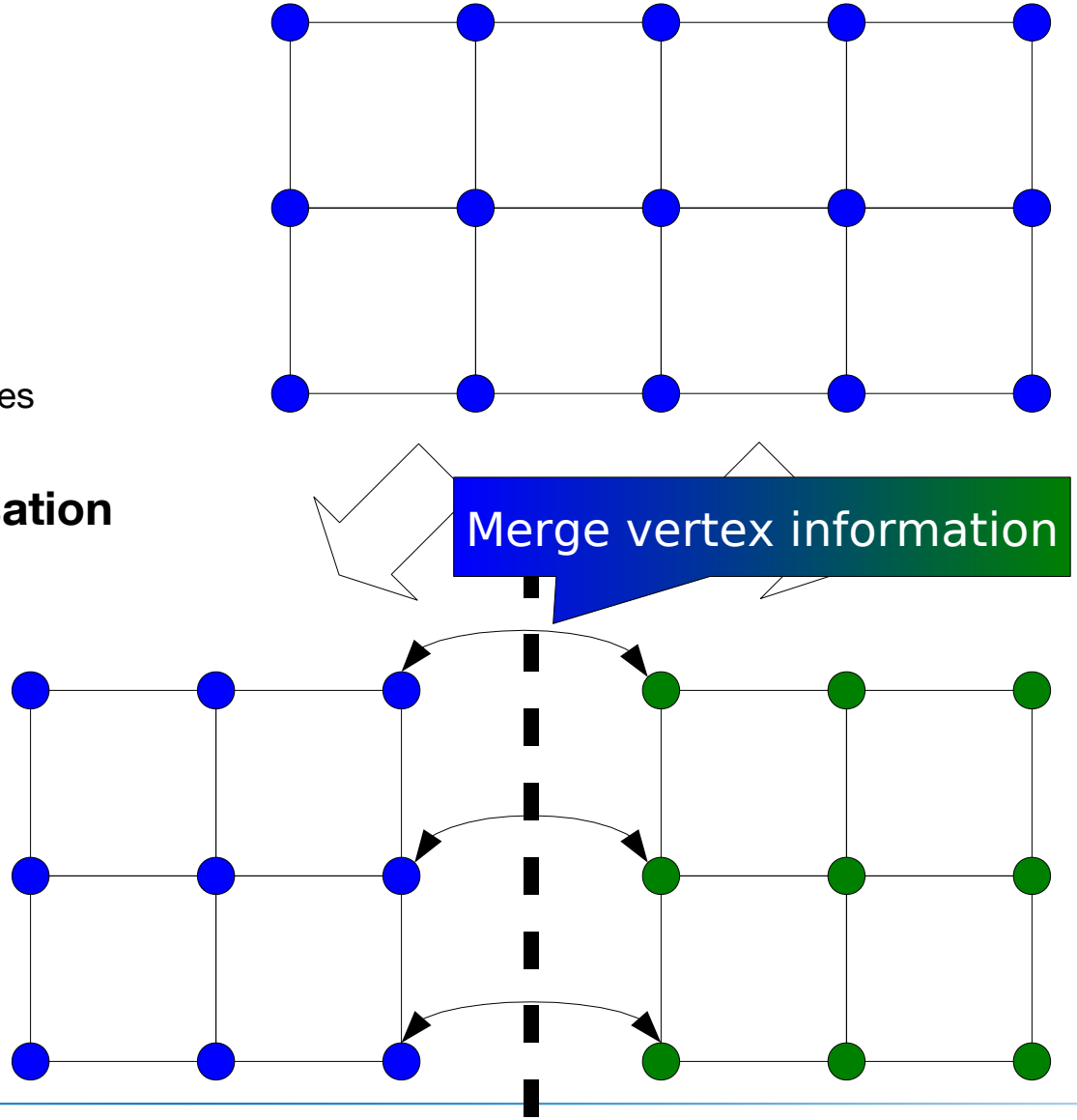
Parallelisation

- Shared memory parallelisation
  - OpenMP
  - Intel threading building blocks
  - Patch-based acceleration techniques
  - Autotuning mechanisms
- **Distributed memory parallelisation**
  - Domain decomposition



Parallelisation

- Shared memory parallelisation
  - OpenMP
  - Intel threading building blocks
  - Patch-based acceleration techniques
  - Autotuning mechanisms
- **Distributed memory parallelisation**
  - Domain decomposition
  - Exchange of vertices between grid iterations

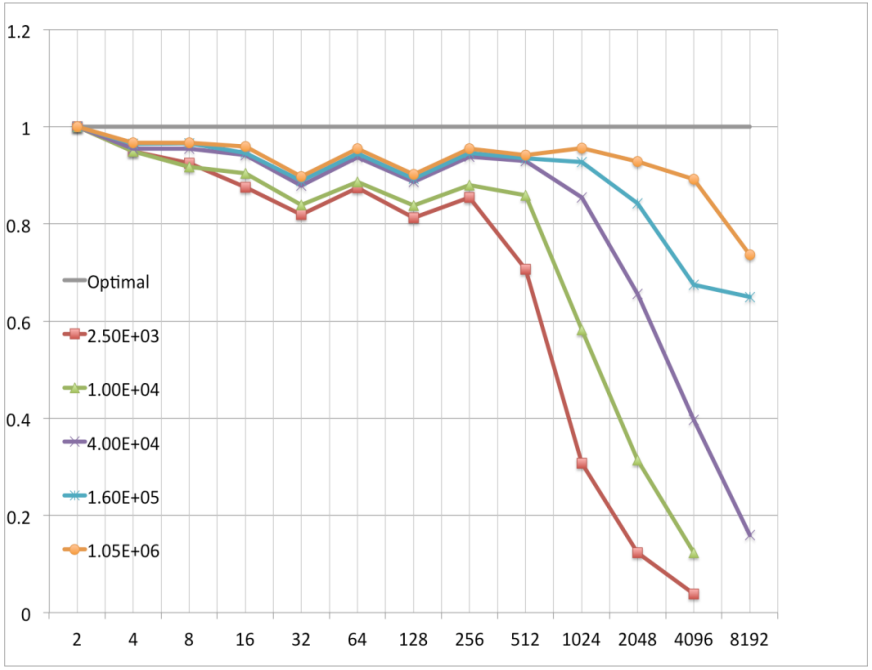


Parallelisation: Don't Call Us – We Call You

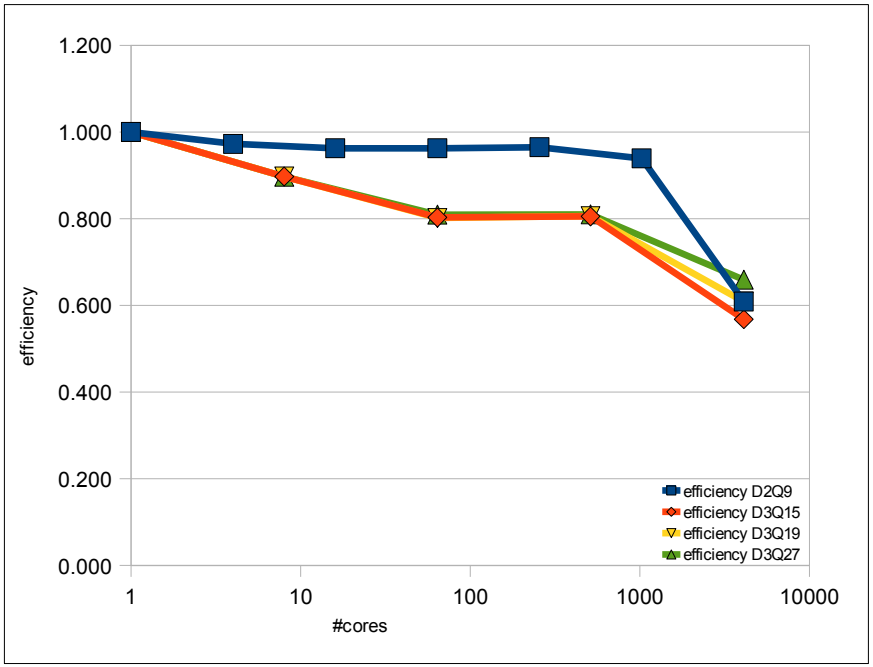
```
void peano::applications::navierstokes::prototype1::mappings::
RegularGrid2MergePressureGradientUpdate::prepareSendToNeighbour(
  RegularGridFluidVertexEnhancedDivFreeEulerExplicit&  vertex,
  int  toRank
) {
 logTraceInWith2Arguments("prepareSendToNeighbour(...)",vertex,toRank);
 // @todo Insert your code here
 logTraceOut("prepareSendToNeighbour(...)");
}
```

```
void peano::applications::navierstokes::prototype1::mappings::
  RegularGrid2MergePressureGradientUpdate::mergeWithNeighbour(
  RegularGridFluidVertexEnhancedDivFreeEulerExplicit&  vertex,
  const RegularGridFluidVertexEnhancedDivFreeEulerExplicit&  neighbour,
  int  fromRank
) {
 logTraceInWith2Arguments("mergeWithNeighbour(...)",vertex,neighbour);
 vertex.mergeGradPUpdate(neighbour);
 logTraceOut("mergeWithNeighbour(...)");
}
```

Parallelisation: Measurements



Navier-Stokes: Weak scaling on Shaheen (Blue Gene/P)



Lattice-Boltzmann: Weak scaling on Shaheen (Blue Gene/P)

Parallelisation: Don't Call Us – We Call You

```
void peano::applications::navierstokes::prototype1::mappings::
RegularGrid2MergePressureGradientUpdate::prepareSendToNeighbour(
  RegularGridFluidVertexEnhancedDivFreeEulerExplicit&  vertex,
  int                                                  toRank
) {
 logTraceIn("prepareSendToNeighbour(...)");
 // @todo Insert your code here
 logTraceOut("prepareSendToNeighbour(...)");
}
```

MPI-Usage **hidden** from applications

```
void peano::applications::navierstokes::prototype1::mappings::
RegularGrid2MergePressureGradientUpdate::mergeWithNeighbour(
  RegularGridFluidVertexEnhancedDivFreeEulerExplicit& vertex,
  const RegularGridFluidVertexEnhancedDivFreeEulerExplicit& neighbour,
  int                                                  fromRank,
  const tarch::la::Vector<DIMENSIONS,double>&          fromX,
  const tarch::la::Vector<DIMENSIONS,double>&          fromH,
  int                                                  level
) {
 logTraceInWith2Arguments("mergeWithNeighbour(...)",vertex,neighbour);
 vertex.mergeGradPUpdate(neighbour);
 logTraceOut("mergeWithNeighbour(...)");
}
```

Parallelisation: Don't Call Us – We Call You

```
void peano::applications::navierstokes::prototype1::mappings::
RegularGrid2MergePressureGradientUpdate::prepareSendToNeighbour(
  RegularGridFluidVertexEnhancedDivFreeEulerExplicit&  vertex,
  int                                                  toRank
) {
 logT                                                          toRank);
 // @
 logT
}
```

MPI-Usage **hidden** from applications

→ Extend this to “arbitrary“ send-/ receive-operations for Peano cells and vertices

```
void
  Reg
  Reg
  con                                                          ghbour,
  int
) {
 logTraceInWith2Arguments("mergeWithNeighbour(...)",vertex,neighbour);
 vertex.mergeGradPUpdate(neighbour);
 logTraceOut("mergeWithNeighbour(...)");
}
```

Parallelisation: Don't Call Us – We Call You

```
void peano::applications::navierstokes::prototype1::mappings::
RegularGrid2MergePressureGradientUpdate::prepareSendToNeighbour(
  RegularGridFluidVertexEnhancedDivFreeEulerExplicit&  vertex,
  int                                                  toRank
) {
 logT                                                          toRank);
 // @
 logT
}
```

MPI-Usage **hidden** from applications

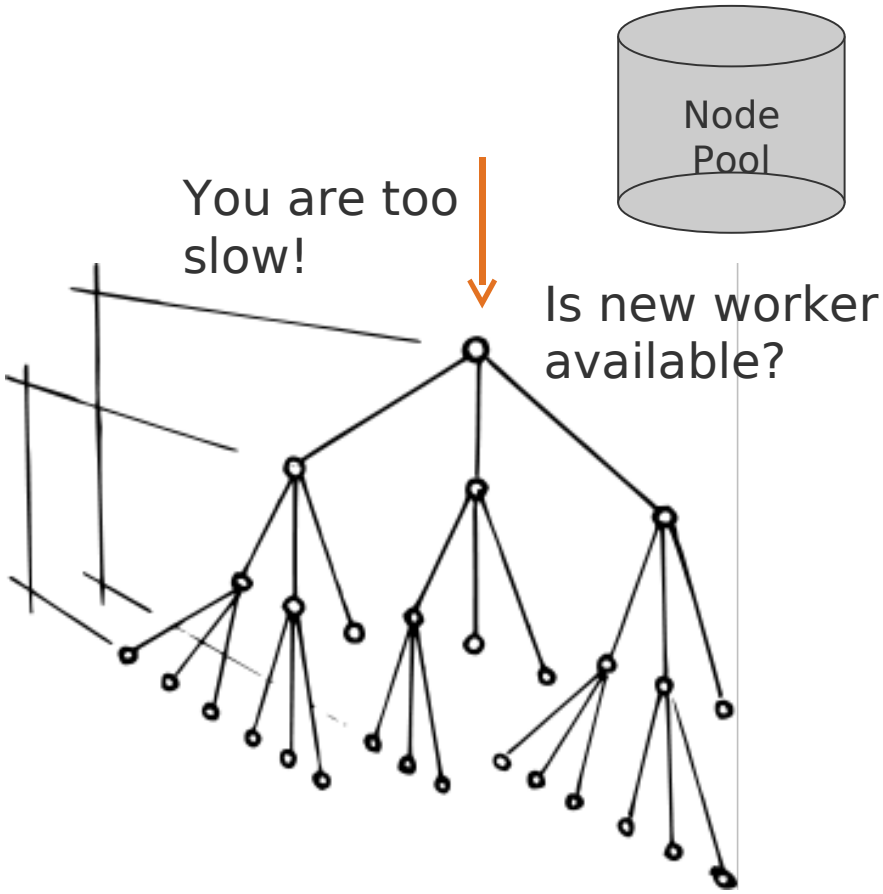
→ Extend this to “arbitrary“ send-/ receive-operations for Peano cells and vertices

→ **Dynamic Load Balancing**

```
void
  Reg
  Reg                                                          ghbour,
  con
  int
) {
 logTraceInWith2Arguments("mergeWithNeighbour(...)",vertex,neighbour);
 vertex.mergeGradPUpdate(neighbour);
 logTraceOut("mergeWithNeighbour(...)");
}
```

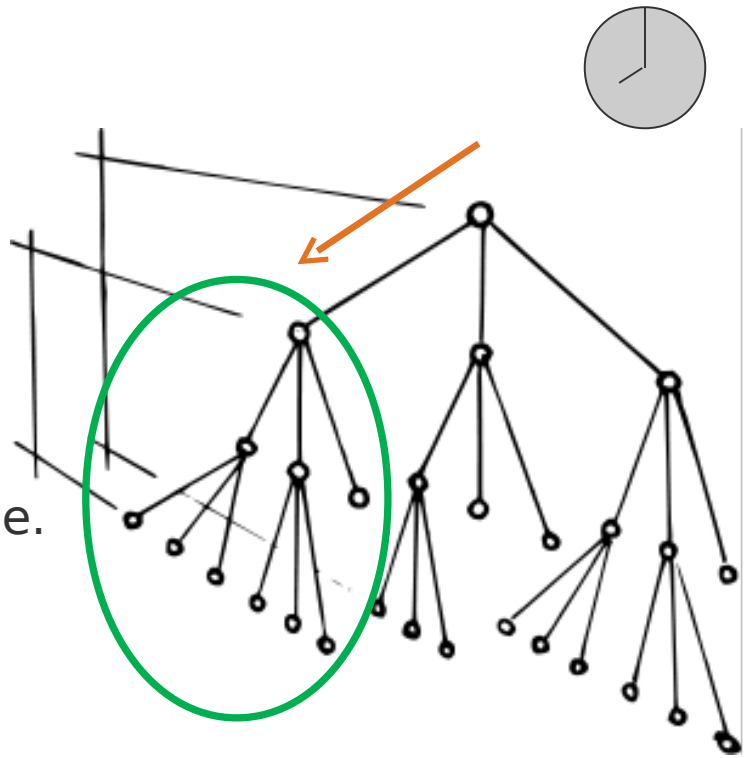
Dynamic Load Balancing

- Master-Worker Concept
  - Sub-trees are distributed among processes
  - Process may be worker **and** master
- Extension of callbacks
- Dynamic Load Balancing
  - Plug into tree reduction
  - Measure wait time and define busy worker and lazy master
  - Deploy join and fork decision to external interface (black-box)



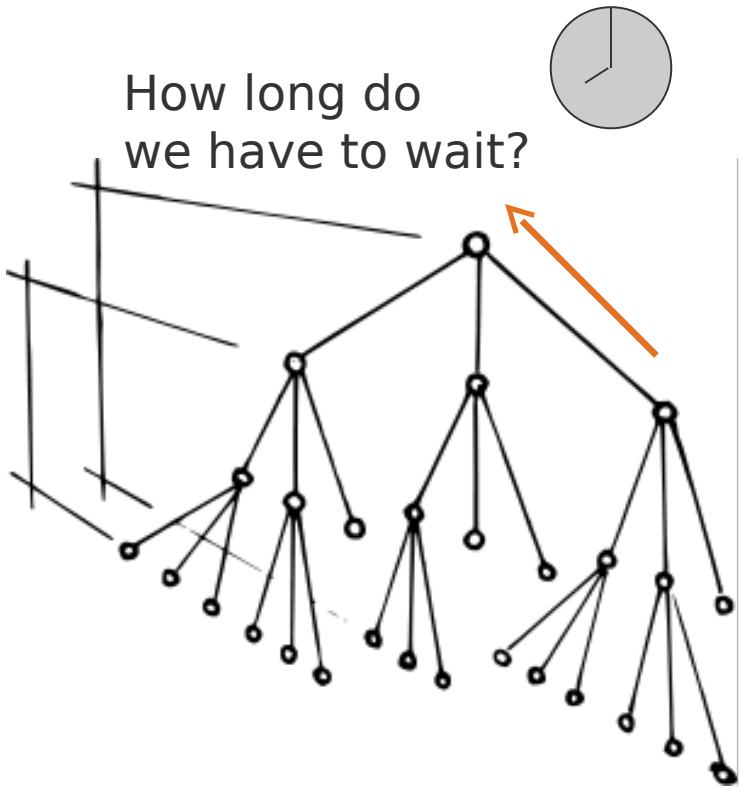
Dynamic Load Balancing

- Master-Worker Concept
  - Sub-trees are distributed among processes
  - Process may be worker **and** master
- Extension of callbacks
- Dynamic Load Balancing
  - Plug into tree reduction
  - Measure wait time and define busy worker and lazy master
  - Deploy join and fork decision to external interface (black-box)



Dynamic Load Balancing

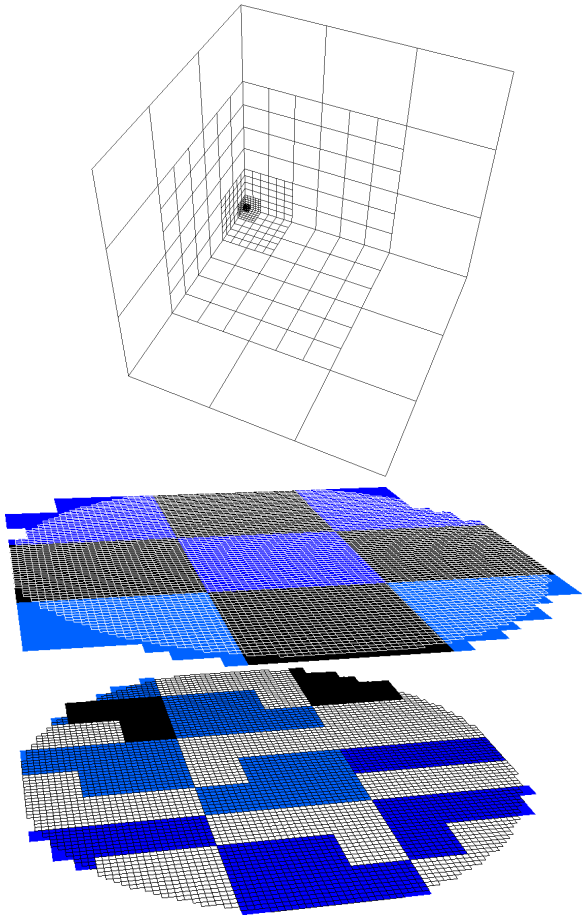
- Master-Worker Concept
  - Sub-trees are distributed among processes
  - Process may be worker **and** master
- Extension of callbacks
- Dynamic Load Balancing
  - Plug into tree reduction
  - Measure wait time and define busy worker and lazy master
  - Deploy join and fork decision to external interface (black-box)

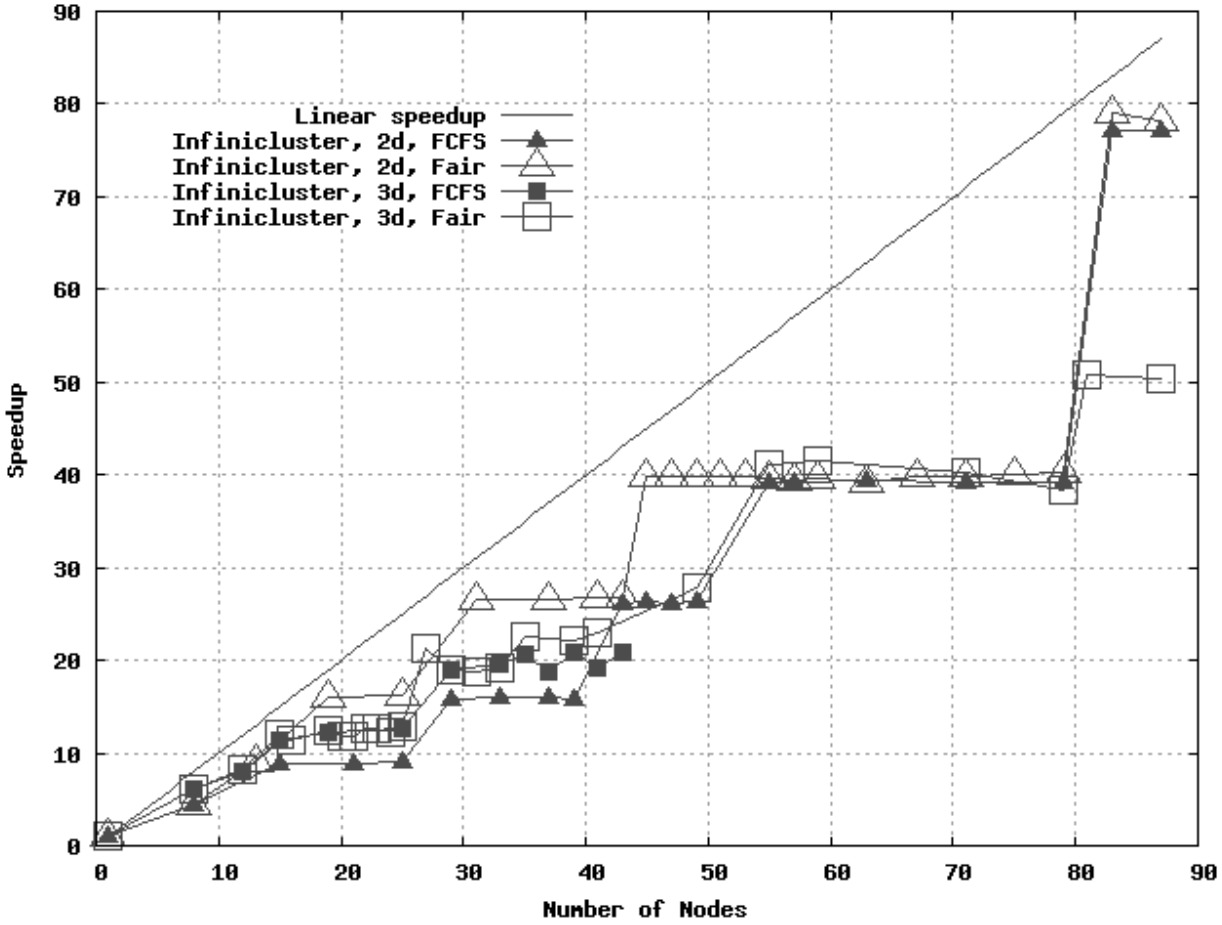


Dynamic Load Balancing

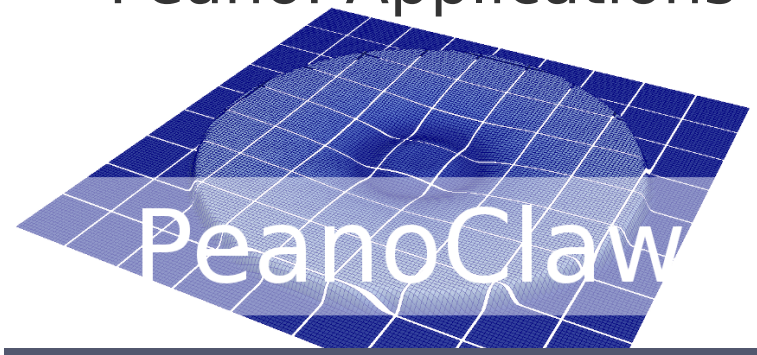
```
class MyMapping {
  [...]

  void prepareCopyToRemoteNode(MyVertex& localVertex,int toRank);
  void prepareCopyToRemoteNode(MyCell &localCell,int toRank);

  void mergeWithRemoteDataDueToForkOrJoin(MyVertex& localVertex,
    const MyVertex&  masterOrWorkerVertex, int  fromRank);
  void mergeWithRemoteDataDueToForkOrJoin(MyCell&  localCell,
    const MyCell&  masterOrWorkerCell,int fromRank);

  [...]
};
```

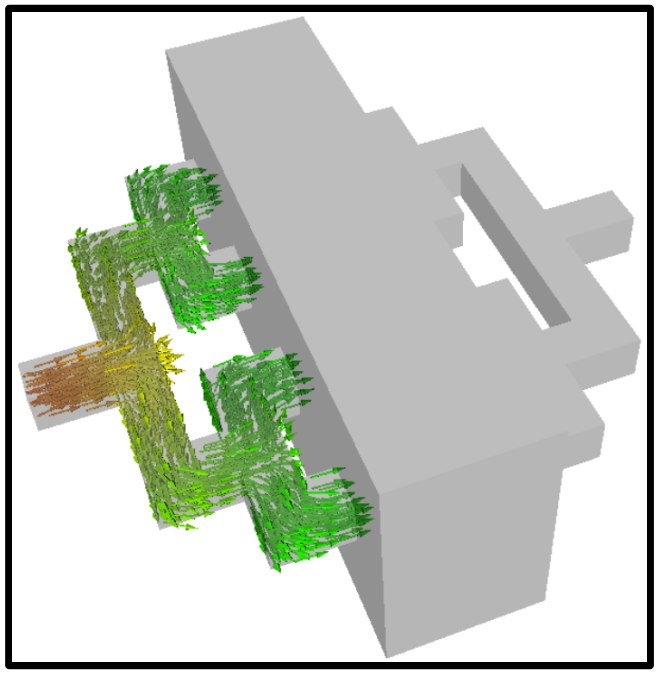
Dynamic Load Balancing

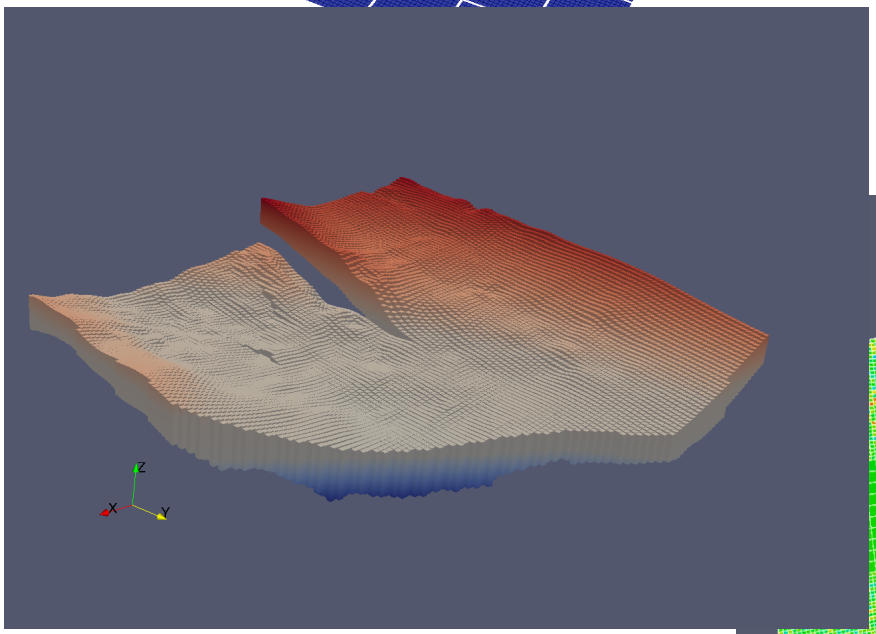


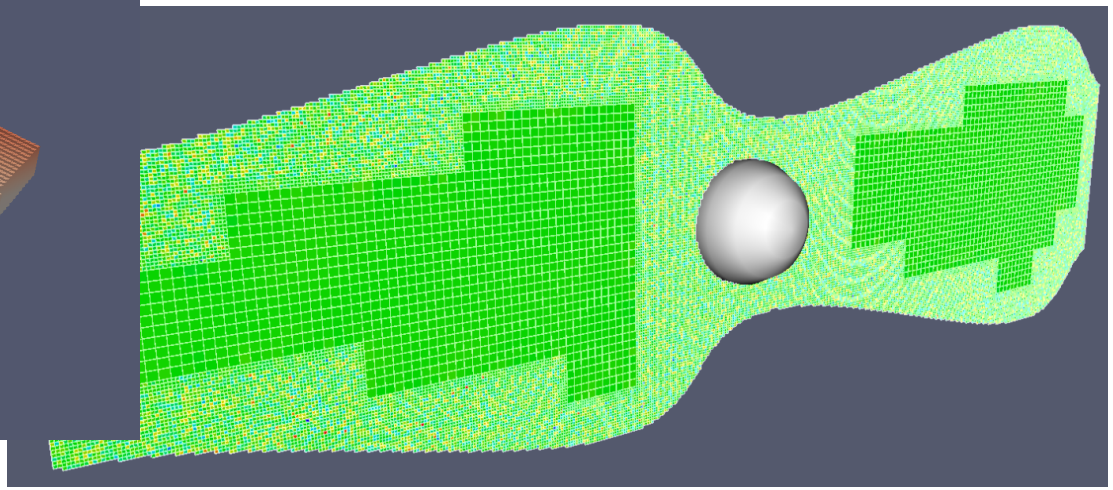


Peano: Applications









The Peano Crew

Atanas Atanasov
Denis Jarema
Michael Lieb
Tobias Neckel
Philipp Neumann
Marco Seravalli
Kristof Unterweger
Tobias Weinzierl

Thank you!

Congratulations, DUNE! :-)