Second phase of work in NEPTUNE

David Moxey & Mashy Green

Department of Engineering, King's College London

Bin Liu, Chris Cantwell & Spencer Sherwin

Department of Aeronautics, Imperial College London

UKAEA Workshop, Abingdon; 5th September 2022

Overview

- Our next phase of work on NEPTUNE will focus on three concrete tasks:
    - **implementation optimisation for performance and scalability (task 1):** integrating our initial developments into the codebase and enabling solver development across architectures.
    - **development of solvers (task 2):** implementing a solver for more realistic fluid-based models with particle coupling (system 2-6).
    - **supporting continued development of NEPTUNE (task 3):** increasing support for modern software standards, coupling with other codes and easing implementation with Python interface and developer-level DSL.

	FY22/23							FY23/24											
Task	S	O	N	D	J	F	M	A	M	J	J	A	S	O	N	D	J	F	Lead(s)/assignees
1.1					M1.1														CC/DM + BL
1.2							M1.2												CC/DM + BL
1.3																	M1.3/D1		CC/DM + BL
2.1							M2.1												DM/SJS + MG
2.2											M2.2/D2								DM/SJS + MG
2.3																	M2.3/D3		DM + MG
3.1		M3.1																	CC/DM
3.2																		M3.2/D4	CC/DM + BL
3.3																		M3.3/D5	DM + MG

Solver development (task 2)

Solver development

- System 2-6 was introduced in initial NEPTUNE equations document.
- Initial goal: consider explicit DG implementation
- Several challenges: numerical fluxes, particle interactions, stiffness, mesh generation of guard cells, ...
- Work towards implicit method based on JFNK approach, which we have been developing for implicit compressible N-S solver.

$$\partial_t n_e + \nabla \cdot (n_e \mathbf{u}_e) = S_{n_e} - \frac{n_e}{\tau_{n_e}}$$

$$\partial_t \nabla \cdot \mathbf{E}^+ + \nabla \cdot (\nabla \cdot (\mathbf{u}_i \otimes \mathbf{E}^+)) = \nabla \cdot \left(n_i (\mathbf{u}_{\nabla Bi} + \mathbf{u}_{cx}) - \frac{1}{Z_i} n_e \mathbf{u}_{\nabla Be} \right) + \frac{1}{Z_i} \frac{n_e}{\tau_{n_e}} - \frac{n_i}{\tau_{n_i}} + \nabla \cdot (\nu \nabla_\perp (\nabla \cdot \mathbf{E}^+))$$

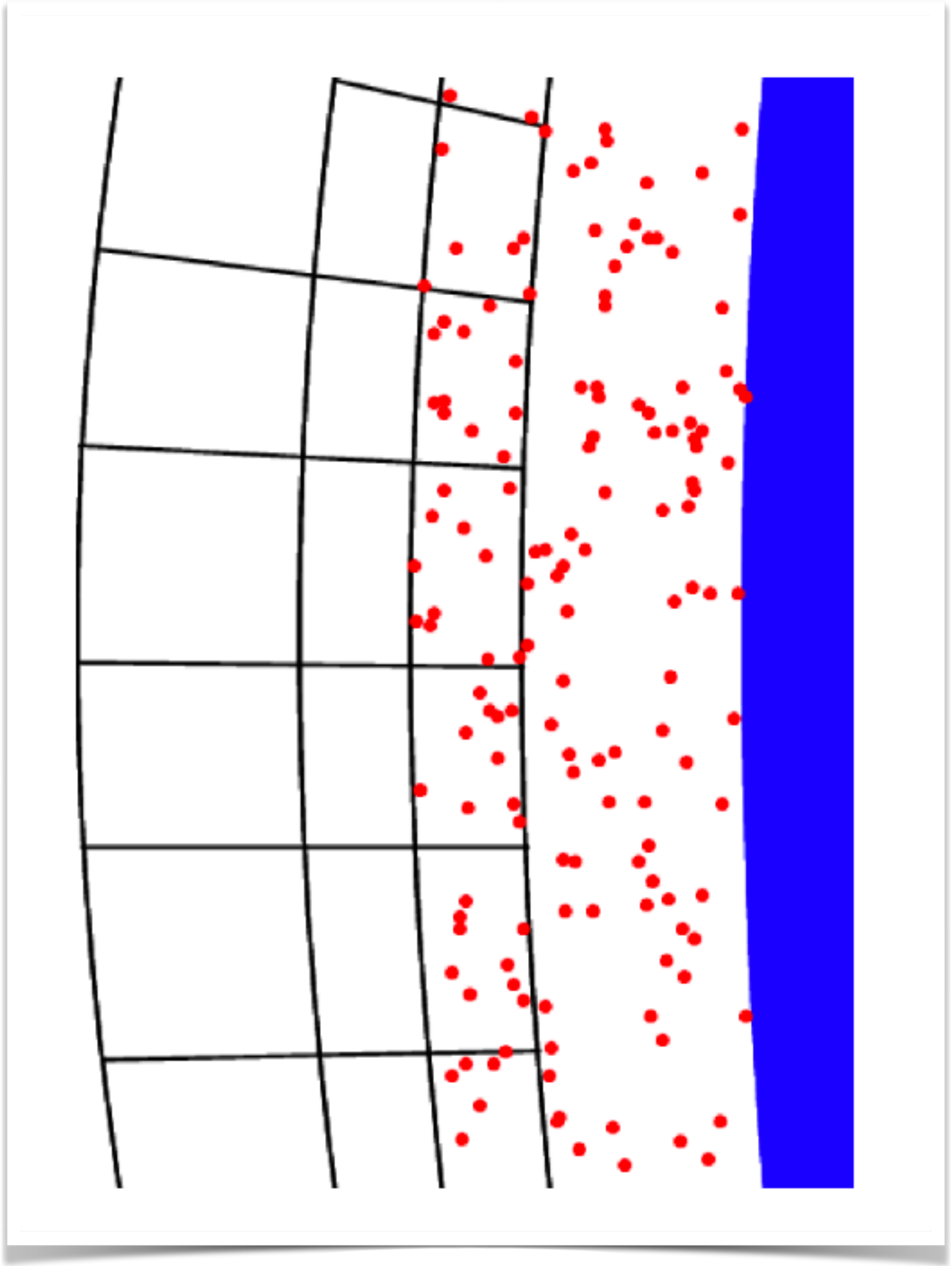
$$\partial_t \mathcal{E}_e + \nabla \cdot (\mathcal{E}_e \mathbf{u}_e + p_e \mathbf{u}_e) = S_{\mathcal{E}_e} - \frac{\mathcal{E}_e}{\tau_{Ee}} + Q_{ie} + \nabla \cdot (\chi_{\perp e} n_e \nabla_\perp T_e)$$

$$\partial_t \mathcal{E}_i + \nabla \cdot (\mathcal{E}_i \mathbf{u}_i + p_i \mathbf{u}_i) = S_{\mathcal{E}_i} - \frac{\mathcal{E}_i}{\tau_{Ei}} - Q_{ie} + \nabla \cdot (\chi_{\perp i} n_i \nabla_\perp T_i)$$

$$\partial_t n_n = S_{n_n} + \nabla \cdot (D_n \nabla_\perp p_n)$$

Mesh generation

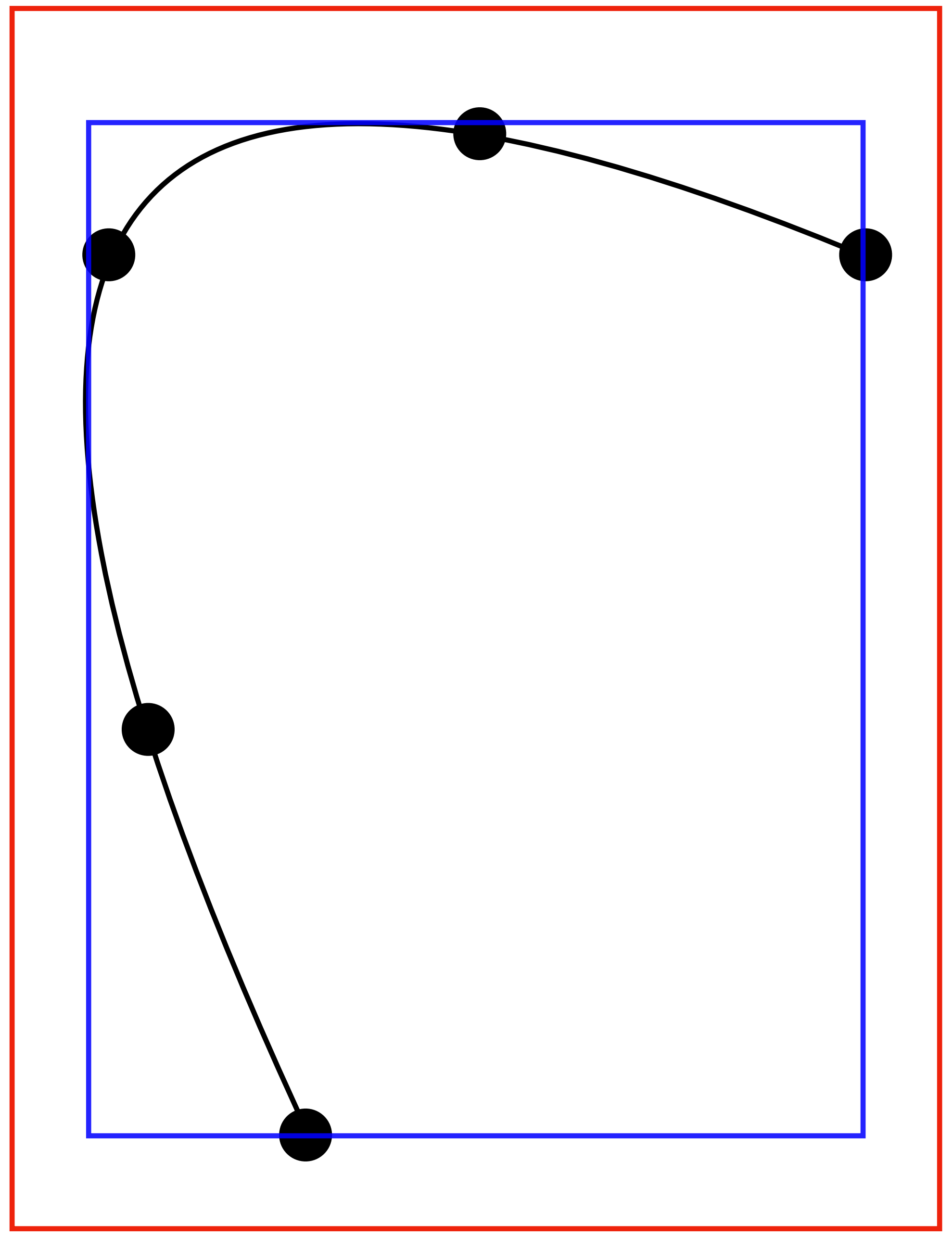
- Various challenges in mesh generation as mentioned in this morning's talk.
- Looking to extend our work to 3D, improve performance/efficiency.
- Other things to consider are related to particle interactions:
  - special treatment around the wall region;
  - creating guard regions around each parallel partition to try to pre-empt transitions between ranks.
- Also looking to improve CAD/mesh/solver workflow for improving interaction with end-users.

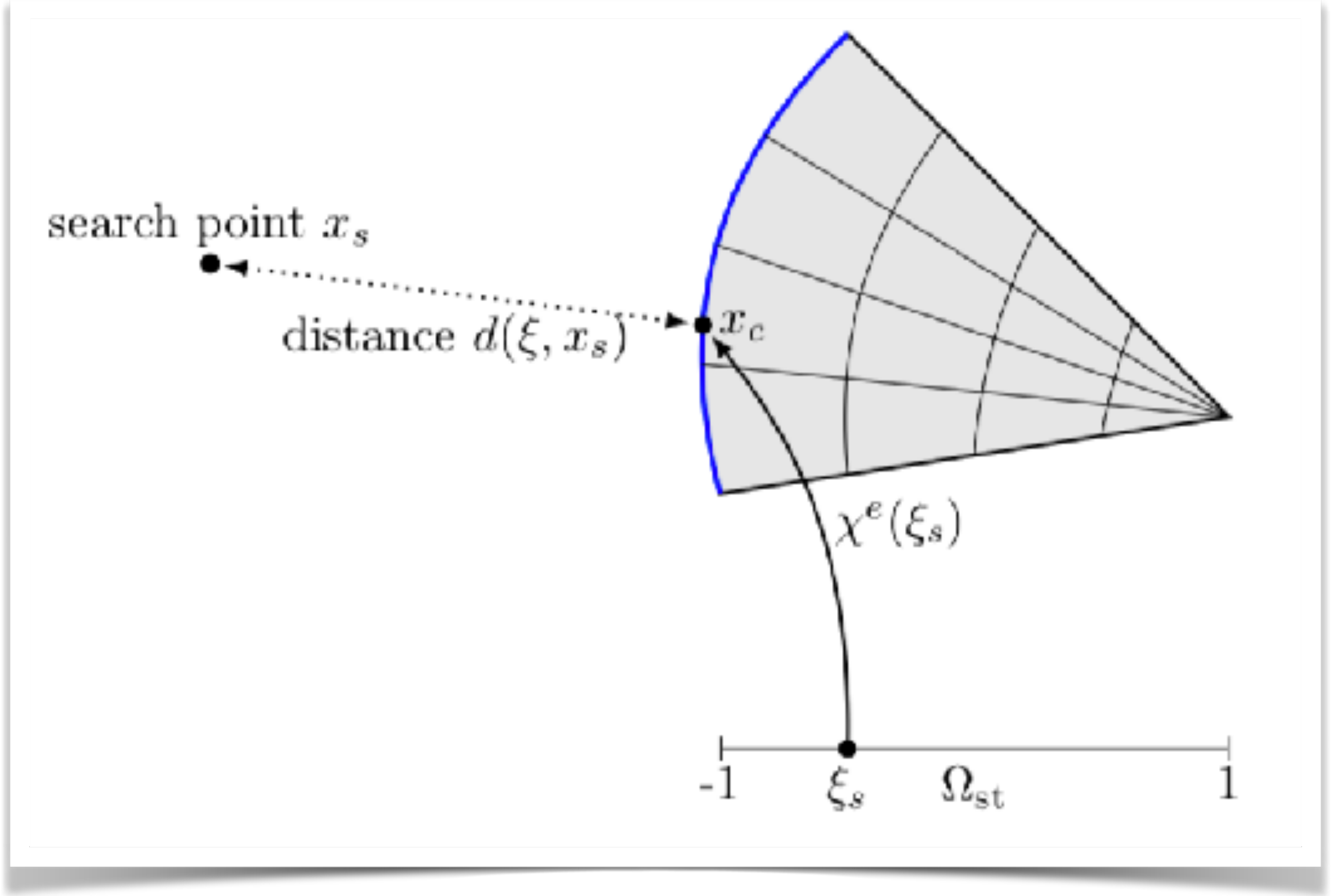


schematic of particles (red) interacting with FE mesh and wall (blue)

Current progress in particle coupling

- Nektar++ provides routines for location to element position.
- Enhanced by using bounding box of element + *r*-tree.
- For straight sided elements quite expensive, curvilinear moreso as requires non-linear search for position (and also bounding box).
- Looking at ways to mitigate costs as far as possible: barycentric interpolation for polynomial evaluation; better initial guesses for nonlinear search.





Finding minimum distance between a point and an element's edge

Community development (task 3)

Overview

- In this task, we will provide general support efforts to facilitate the use of Nektar++ within the wider NEPTUNE code.
- In part this will be through direct interaction with UKAEA team, but also the following actions:
  - improving exposure to modern software standards (C++17)
  - integration of coupling with other frameworks (e.g. MUI).
  - extend Python interface to support creation of solvers and a simpler DSL-like interface;
  - other performance improvements (increasing memory locality).

Python interface

- Current Python interface provides access to pre-processing (NekMesh modules), post-processing (FieldConvert modules) and the core library API.
- Extend Python interface to facilitate solver integration with other software:
  - Avoid requirement for XML input (in progress).
  - Extend support to include SolverUtils library, in particular EquationSystem (in progress).
  - Wrap filters used to post-process during simulations (in progress).
  - Wrap time-integration schemes.
  - Enable construction of solvers entirely from Python in a simplified and more DSL-like manner.
- Some initial developments: https://gitlab.nektar.info/dmoxey/nektar/-/commits/feature/solver-plugins

C++17 transition

- `std::filesystem` and `std::thread` (reduced exposure to Boost)
- Polymorphic allocators - enable containers with different allocators to be treated as the same data type - useful for multiple device / memory-space support.
- `std::aligned_alloc` (memory aligned allocation) - important for vectorisation
- Nested namespaces, (e.g. `Nektar::LibUtilities { }`), new attributes: `[[fallthrough]]` (switch statements), `[[maybe unused]]`
- Some initial exploration: https://gitlab.nektar.info/dmoxey/nektar/-/commits/feature/c++17

Other planned improvements

- Automatic code formatting (completed July 2022)
    - Clang-format v11, included .clang-format configuration & should be applied on all MRs before merging
- Improved error-checking
    - Static registration of names/parameters with SessionReader
    - Registration at point-of-use - avoid coupling of code & enable new options to be defined at solver-level but still validated.
- Updated session file format, including better validation
    - Make better use of XML syntax, more intuitive structure of input configuration options
- XML-free solver usage: simplify integration with other software tools, allows all aspects of simulation to be programmatically specified as required

Implementation optimisation (task 1)

Current challenges

CPU-focused design

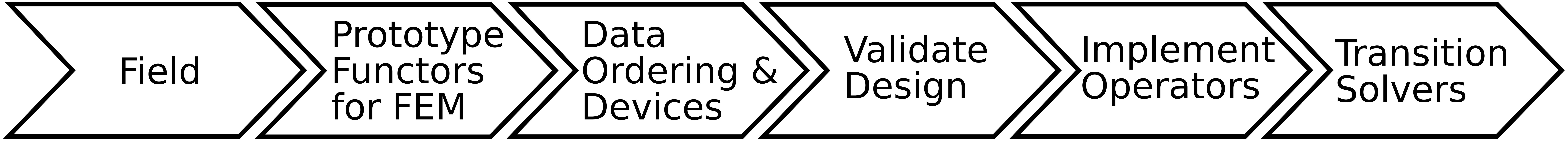
- Code was designed during the CPU era
- Flat storage assuming coarse-grained MPI parallelism
- Support for vectorisation and GPUs came later but did not fit the design - target specific regions of code
- Inefficient need to re-order data between sections of code
- Prevents competitive use of GPU systems and modern CPU systems

Restricted flexibility

- Alignment with mathematical foundations limits programmatic freedom
- Coupling of data structures with algorithms, e.g. `ExpList`
- Operators forced to be in the library
- Hampers creation of new solvers, requiring library changes

Redevelopment Pathway

1. Implement a new container for storing Field data
2. Create a hierarchy of functor classes to implement core FEM operations
3. Extend Field storage class to support re-ordering and multiple devices
4. Validate design for multi-architecture performance
5. Incrementally implement functors for each operator and transition ExpList to use them
6. Transition solvers to use functors directly and clean up ExpList



Redevelopment Benefits

- Support for AVX2, AVX512, GPU, ARM as well as CPU across all core operators
- Multiple data layouts to align with architecture requirements to achieve performance / cache coherency
- Flexibility to implement new operators for a solver outside the library and link into the underlying infrastructure without needing to modify the core libraries.

1. Field storage

StorageType
- ePhys
- eCoeff

Field<type TData, StorageType TStype>

GetData()
UpdateData()
GetData1D()
GetExpList()

Public API

std::vector<ExpListFieldInterface> m_expIF

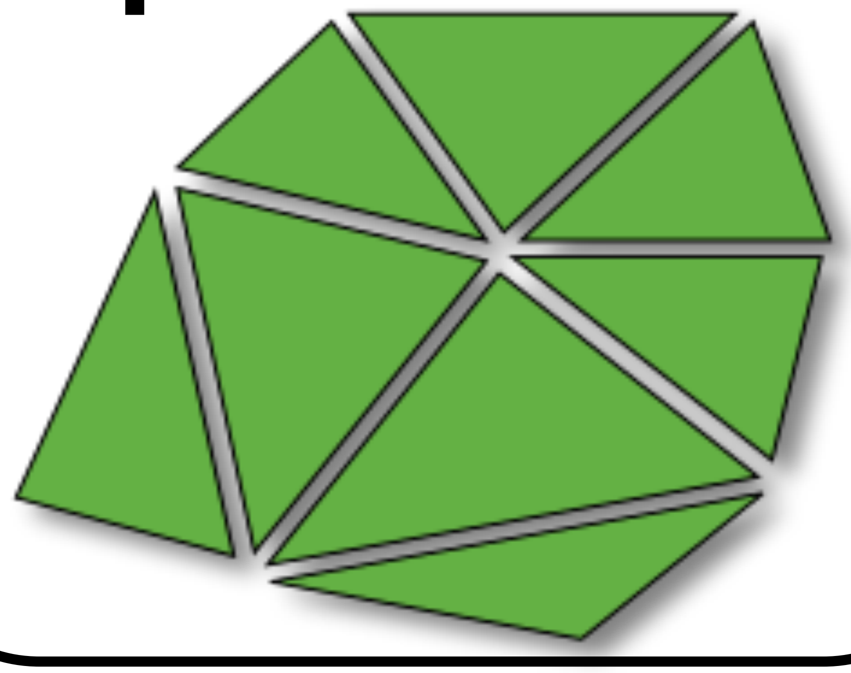
std::size_t m_numVariables

std::vector<TData> m_storage

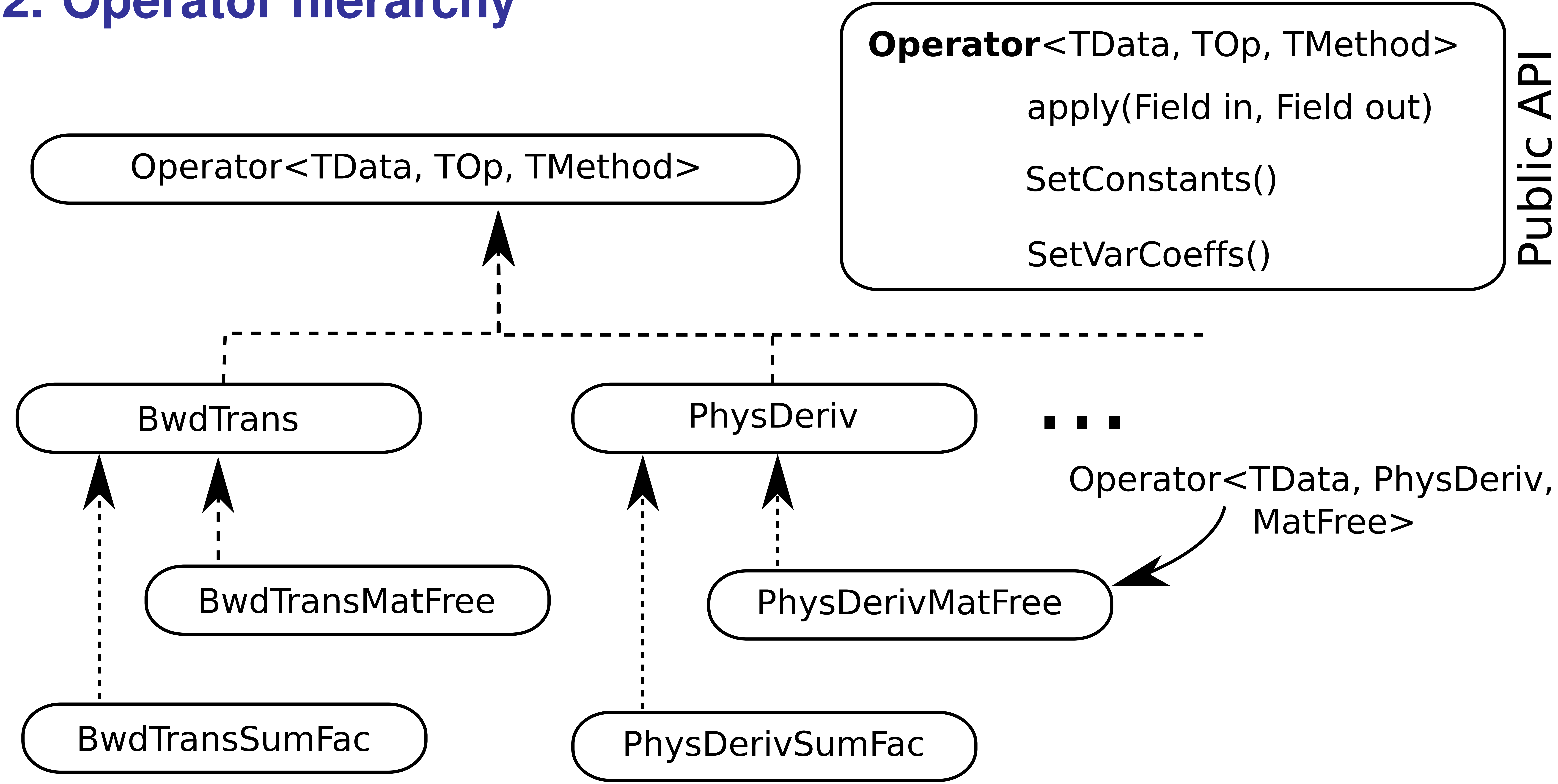
SYCL buffer?

ExpListField Interface

ExpList



2. Operator hierarchy



3. Data re-ordering and multi-device support

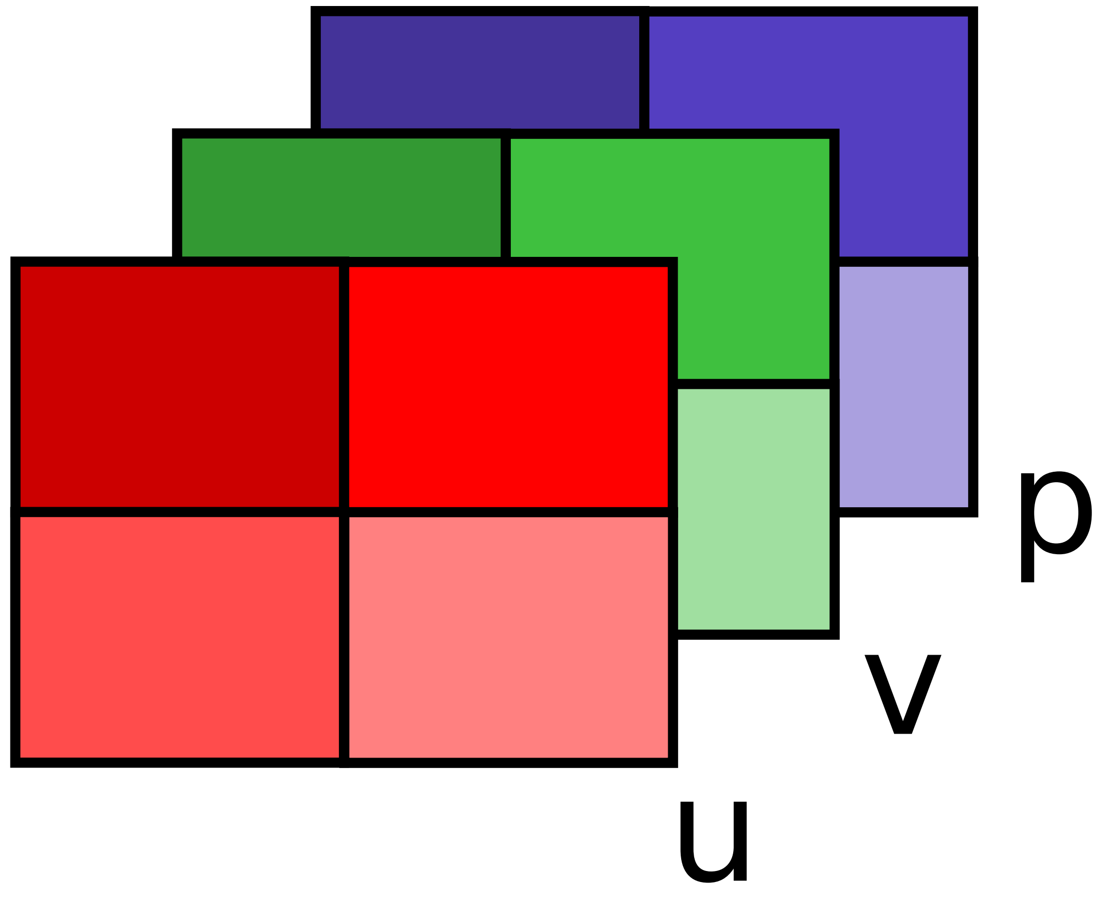
Data reordering

- Data may be ordered by variable, element, or DOF (interleaved)
- Different operators may be more performant in different orderings (e.g. by variable for HelmSolve, by element/DOF for advection)
- Need to avoid unnecessary re-ordering to avoid performance overhead
- Allow operators to support multiple re-orderings - encode the knowledge of cost/benefit of reordering into the operators

Multi-device support

- Generally concerned with GPU support
- Transparently (but smartly!) move data between host and device when needed.
- Investigate efficacy of SYCL buffers, or direct (e.g) CUDA

3. Data re-ordering and multi-device support

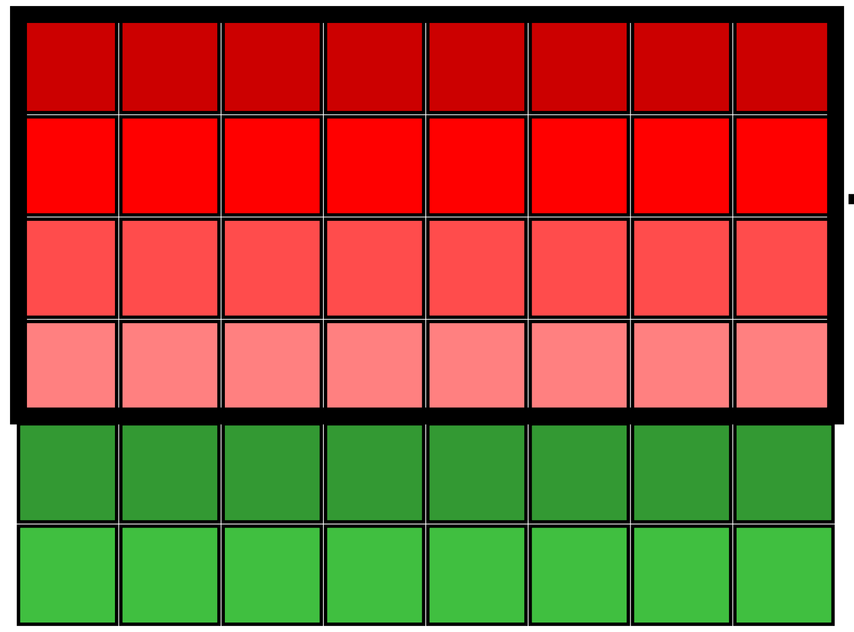


3 Variables

4 Elements

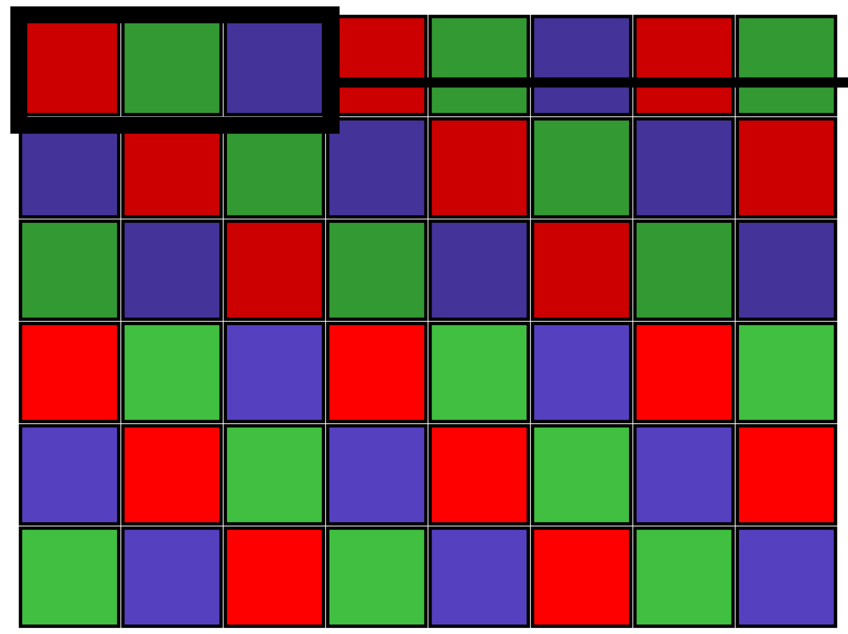
8 DOFs/Element

By-variable, by-element



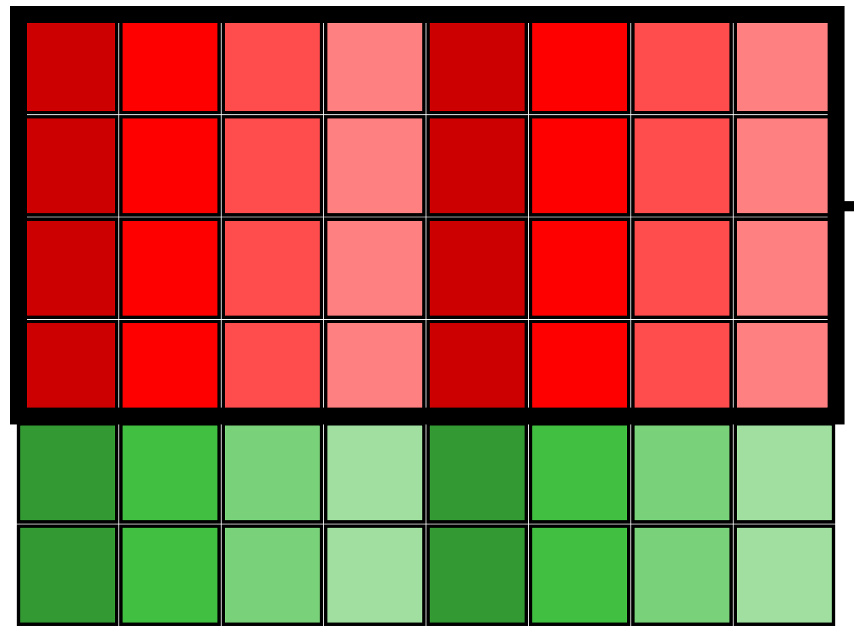
$$\nabla^2 u - \lambda u = f$$

By-element, by-dof



$$(\mathbf{u}.\nabla)\mathbf{u}$$

By-variable, by-element (vectorised, VW=4)



$$\mathbf{u} = \mathbf{B}\hat{\mathbf{u}}$$

3. Data re-ordering and multi-device support

StorageType
- ePhys
- eCoeff

Field<type TData, StorageType TStype>

GetData()
UpdateData()
GetData1D()
GetExpList()

Public API

std::vector<ExpListFieldInterface> m_expIF

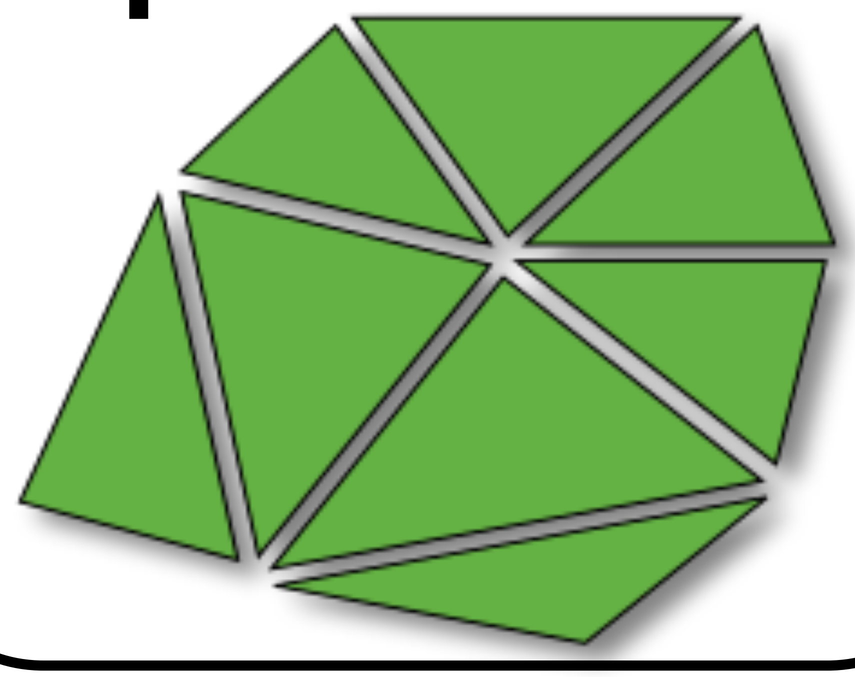
std::size_t m_numVariables

std::vector<TData> m_storage

SYCL buffer?

ExpListField Interface

ExpList



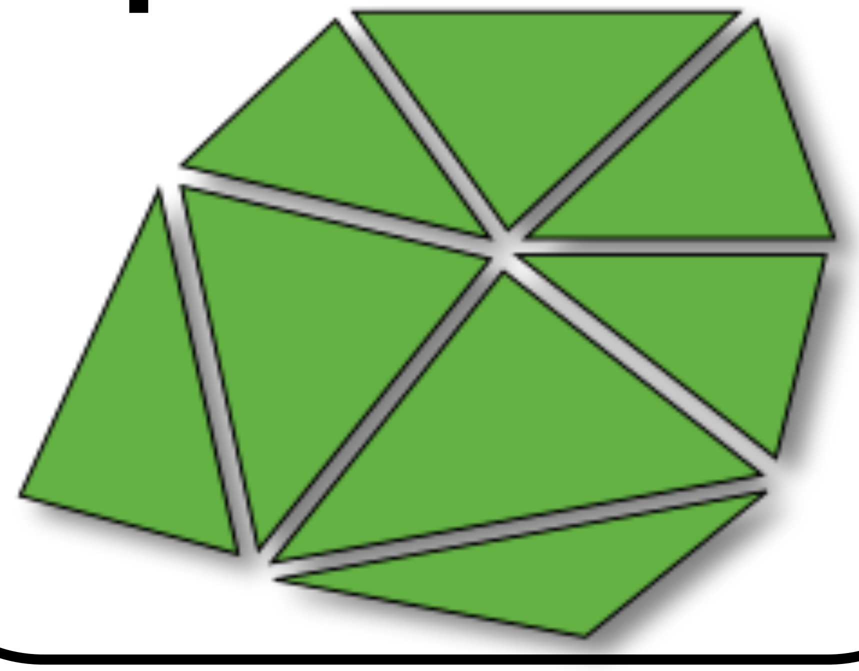
3. Data re-ordering and multi-device support

StorageType
- ePhys
- eCoeff

DataLayout
- eVarElmtDof
- eVarDofElmt
- eElmtDofVar
- ...

ExpListField Interface

ExpList



Field<type TData, StorageType TStype>

DataLayout m_layout

std::size_t m_vectorWidth

std::vector<ExpListFieldInterface> m_expIF

std::size_t m_numVariables

std::vector<TData> m_storage

SYCL buffer?

Public API
- GetData()
- UpdateData()
- GetData1D()
- GetExpList()
- GetLayout()
- SetLayout()
- GetVectorWidth()
- SetVectorWidth()

4. Validation

- BwdTrans
- Derivative app (PhysDeriv)
- Projection app (BwdTrans + IProductWRTBase + InvMassMatrix)
- Helmholtz app (HelmSolve + BwdTrans)
- DG Advection app (BwdTrans + IProductWRTBase + InvElementalMassMatrix + IProductWRTDerivBase)

5. Transition solvers to new design (Pseudo-code)

```
int main(int argc, char* argv[]) {
    LibUtilities::SessionReader session(argc, argv);
    SpatialDomains::MeshGraph   mesh(session);
    MultiRegions::ContField     exp(session, mesh);

    Field<NekDouble, ePhys>  ic;
    Field<NekDouble, eCoeff> coeffs;
    Field<NekDouble, ePhys>  sol;

    // Project initial condition
    Operator<IProductWRTBase>::create() (ic, coeffs);
    Operator<BwdTrans>::create()        (coeffs, sol);

    // Helmsolve
    FactorMap   fmap = { {"lambda", session->GetParameter("lamda")} };
    Operator<Zero>::create()            (coeffs);
    Operator<HelmSolve>::create(fmap) (ic, coeffs);
    Operator<BwdTrans>::create()      (coeffs, sol);

    LibUtilities::FieldIO fld = LibUtilities::FieldIO::CreateDefault(
        session);
    fld->Write(sol);
}
```

Support for multiple platforms