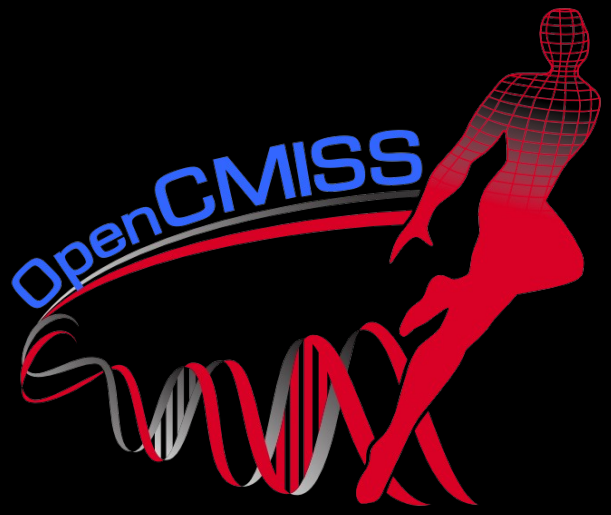
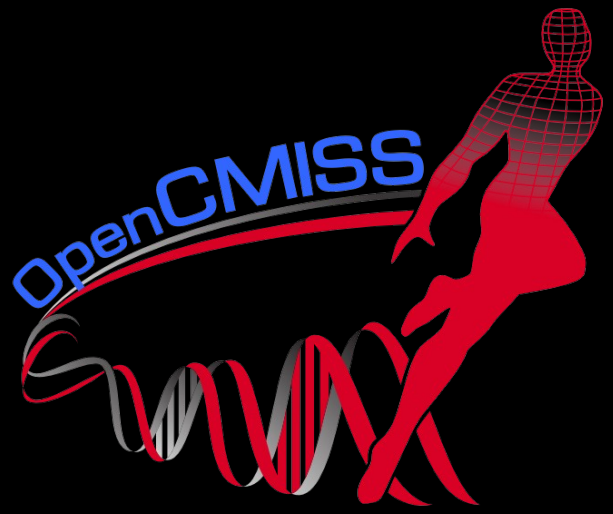


OpenCMISS (Iron)

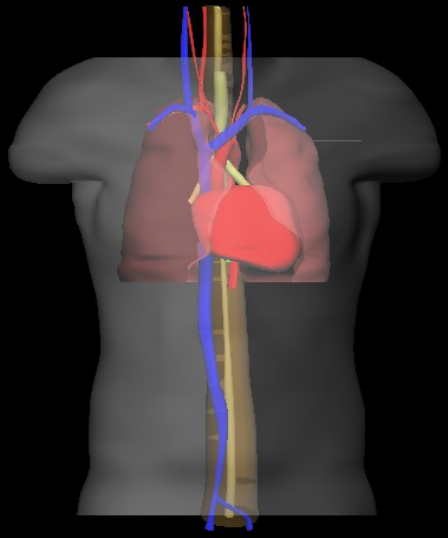
Chris Bradley

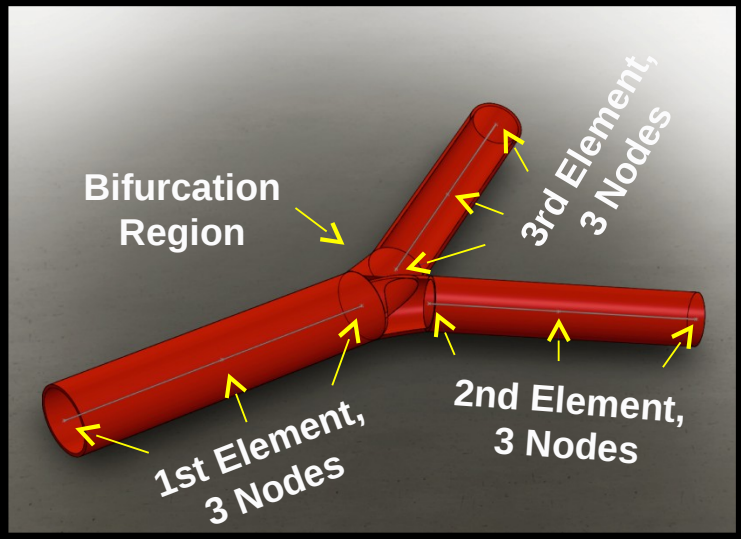
OpenCMISS?

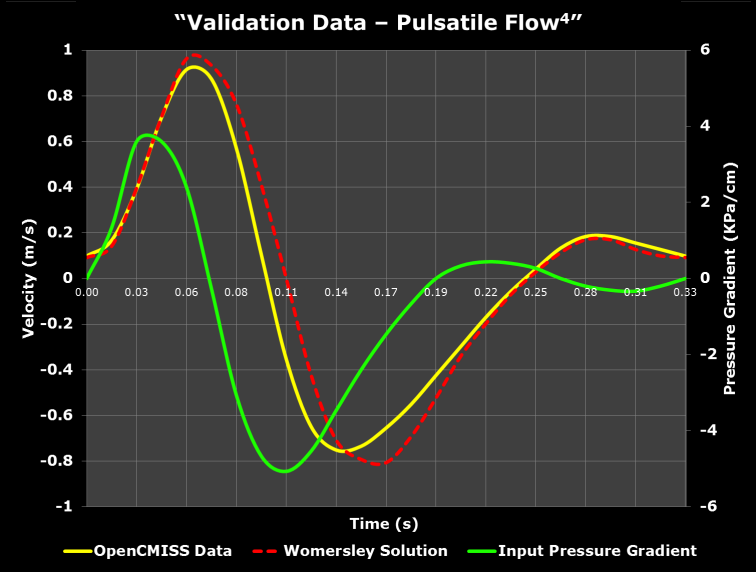


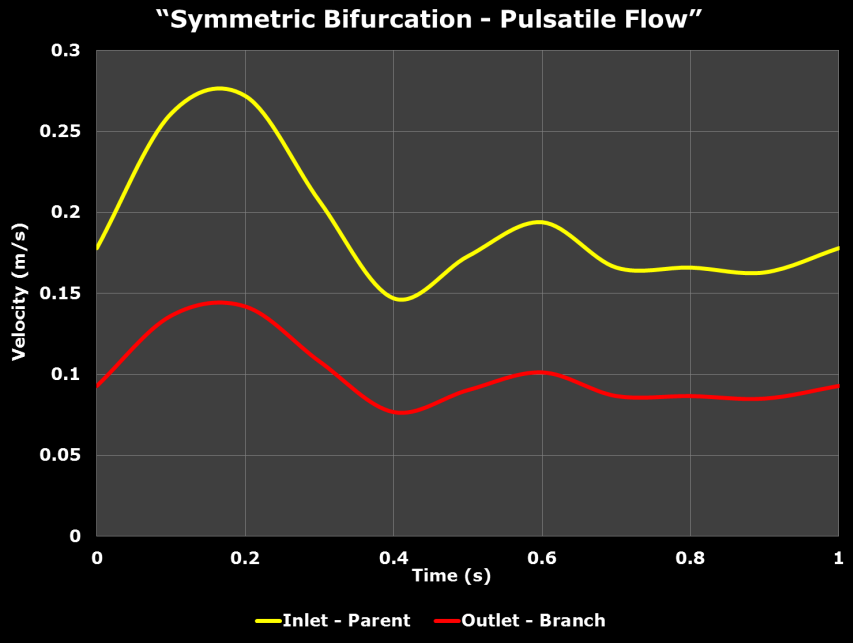
OpenCMISS is a general modelling environment for the solution of mathematical and bioengineering models.

Fluid Mechanics – 1D NS





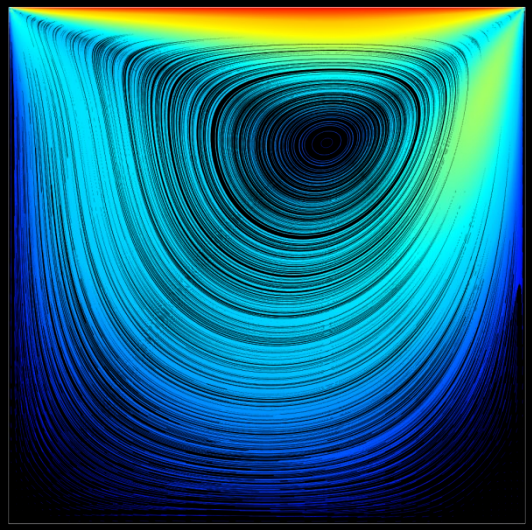




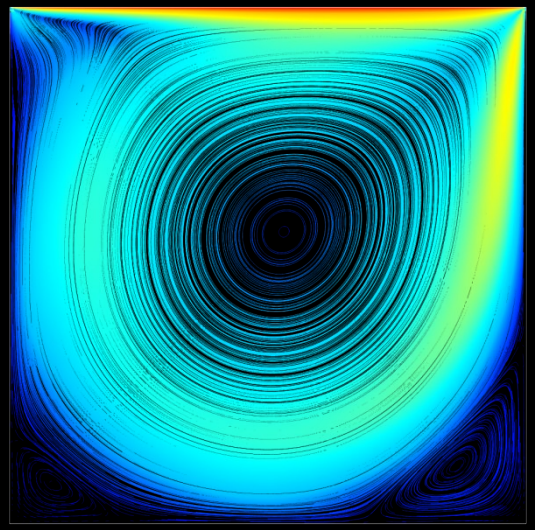
Courtesy of Soroush Safaei

Fluid Mechanics – 2D NS

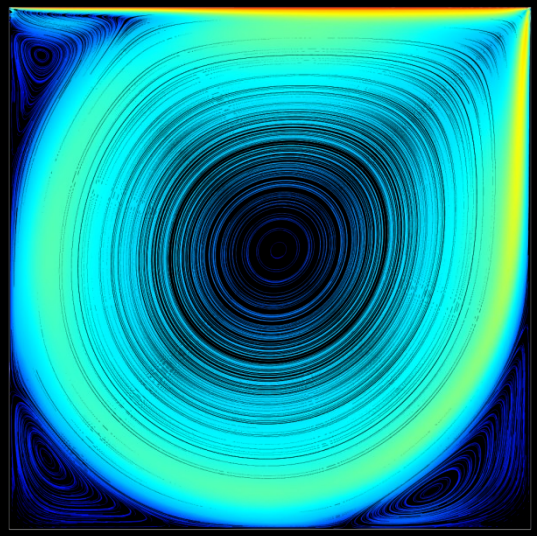
Lid Driven Cavity



Re=100



Re=1000

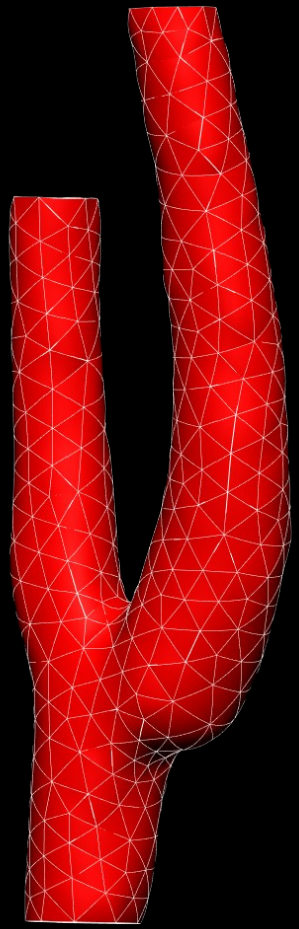


Re=5000

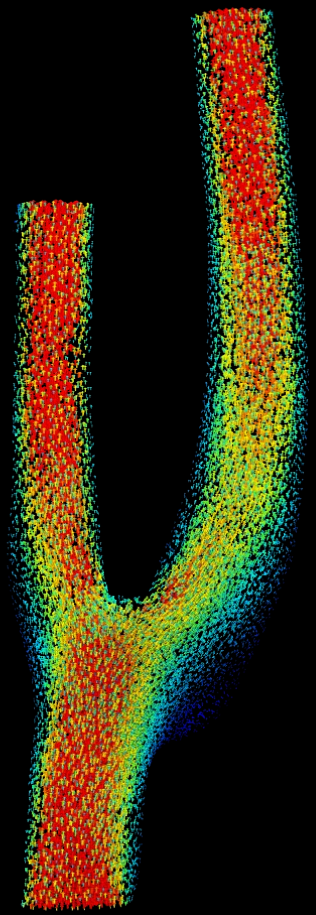
Courtesy of David Ladd

Fluid Mechanics – 3D NS

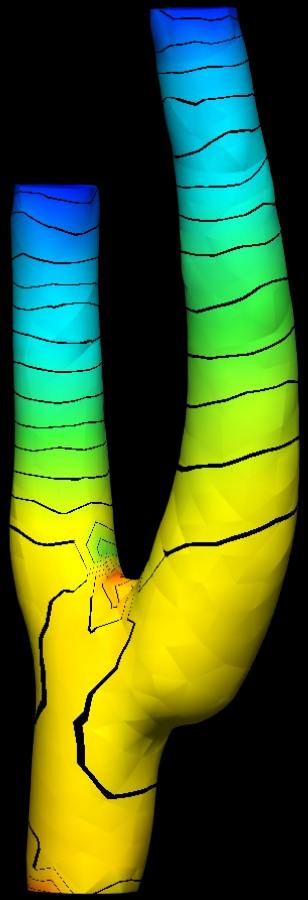
Carotid Artery Flow, Re=100



Mesh



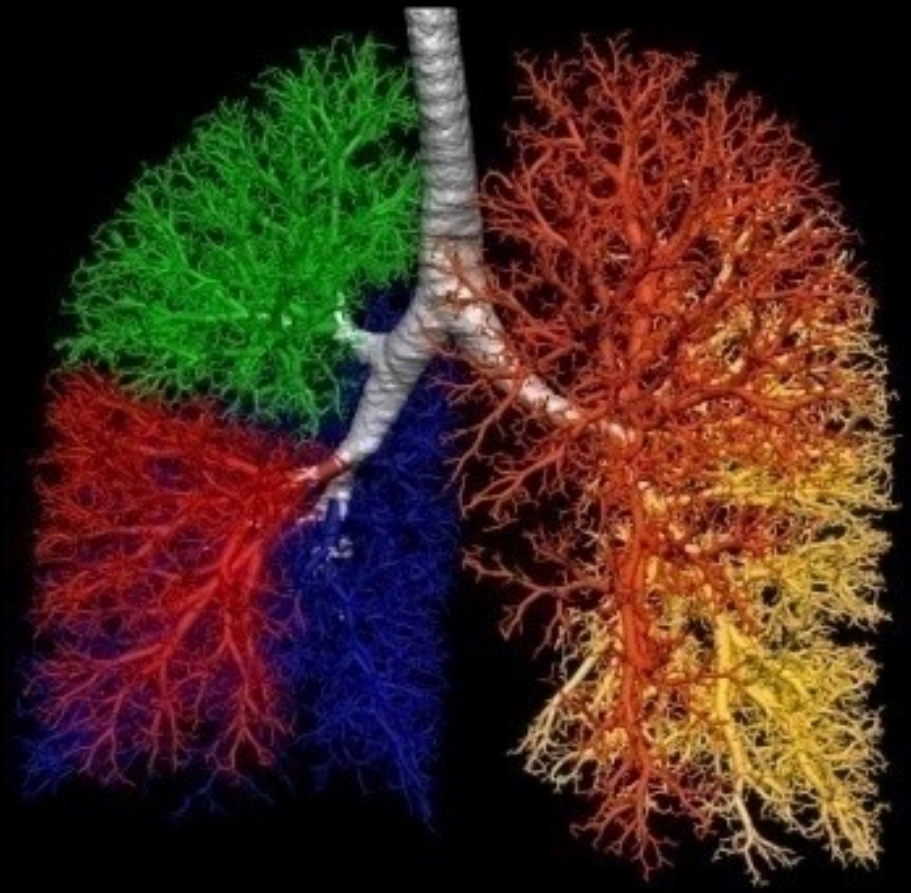
Velocity



Pressure

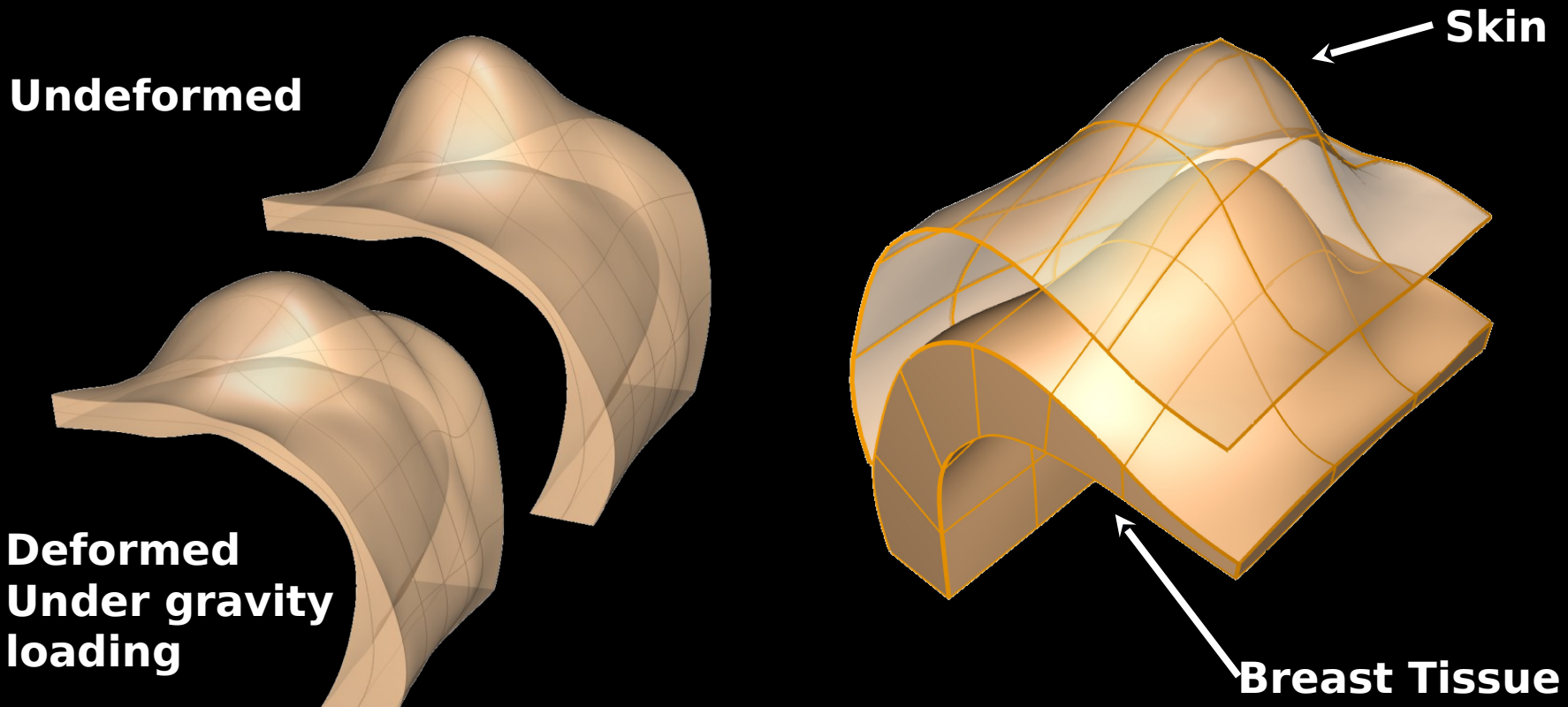
Courtesy of David Ladd

Pulmonary Flow



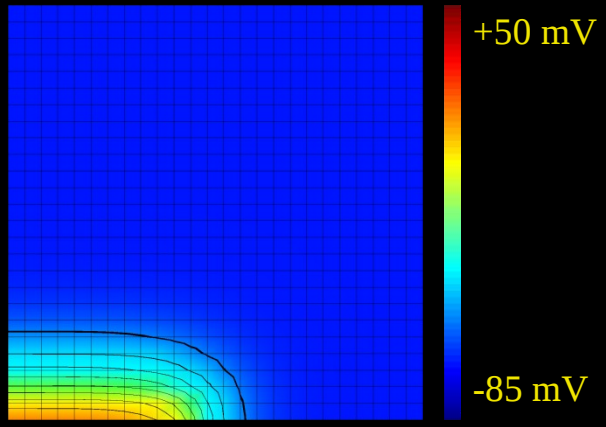
Courtesy of Merryn Tawhai and Jennine Mitchell

Finite Elasticity – Breast Mechanics

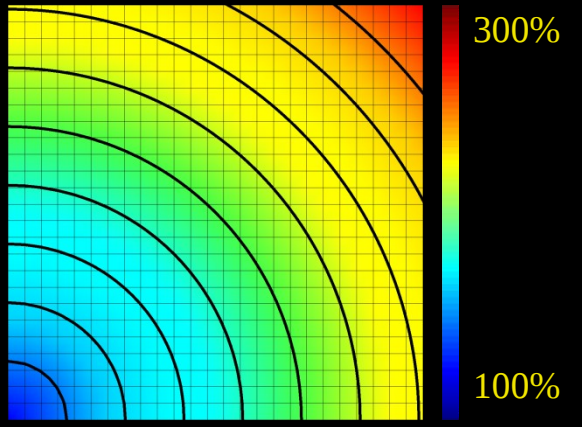


Courtesy of Prasad Gamage

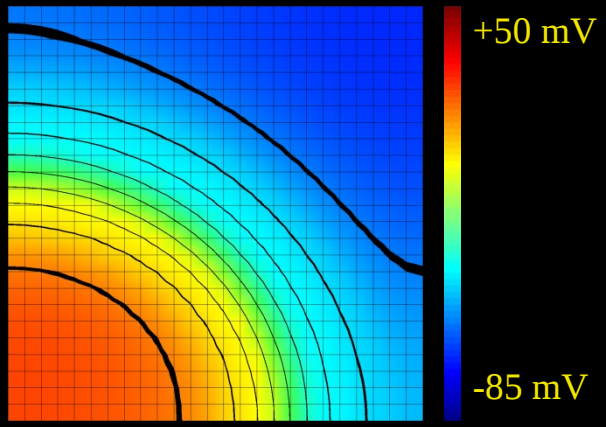
Electrophysiology - CellML



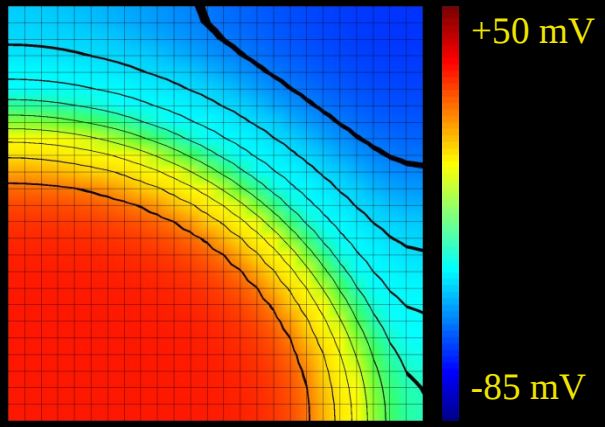
Stimulus off



gNa Distribution

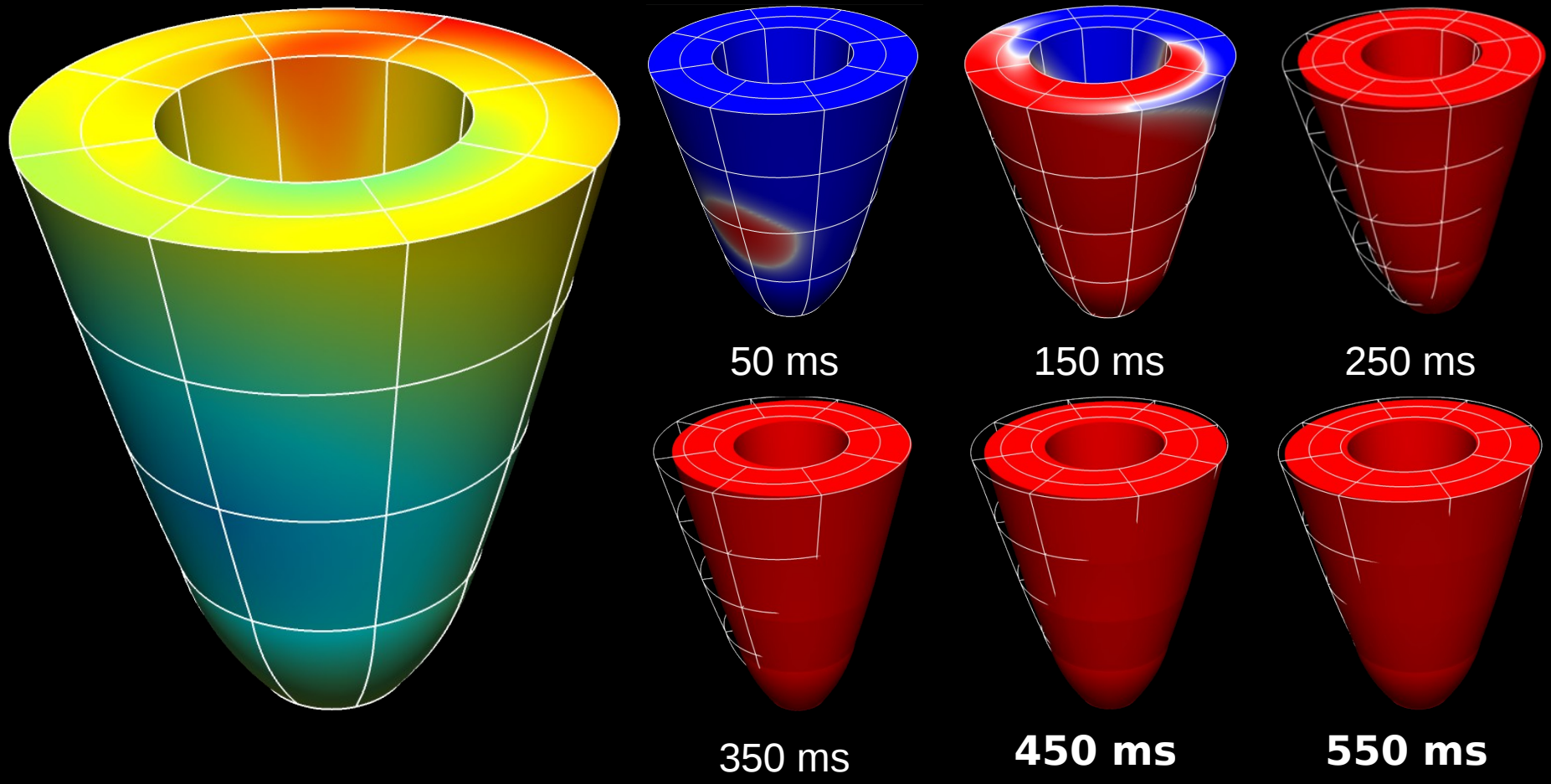


Normal activation



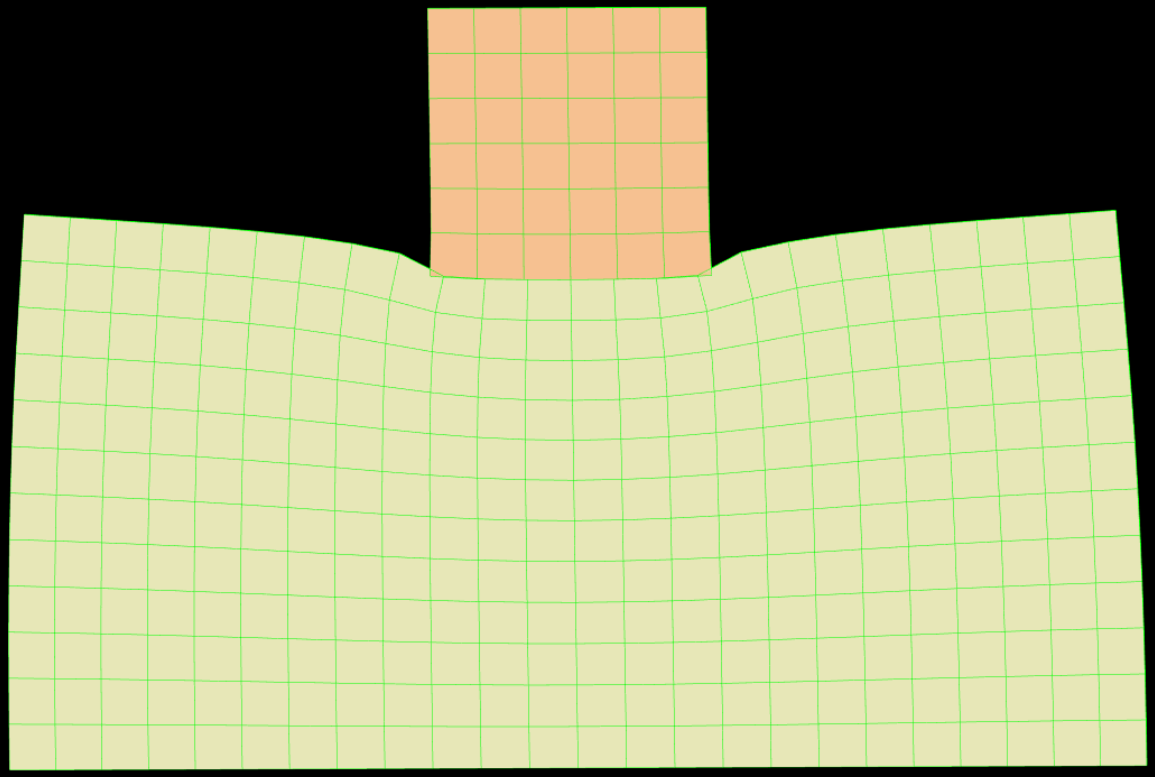
gNa activation

Electromechanics - Eikonal



ourtesy of Ali Pashaei and Martin Steghöfer

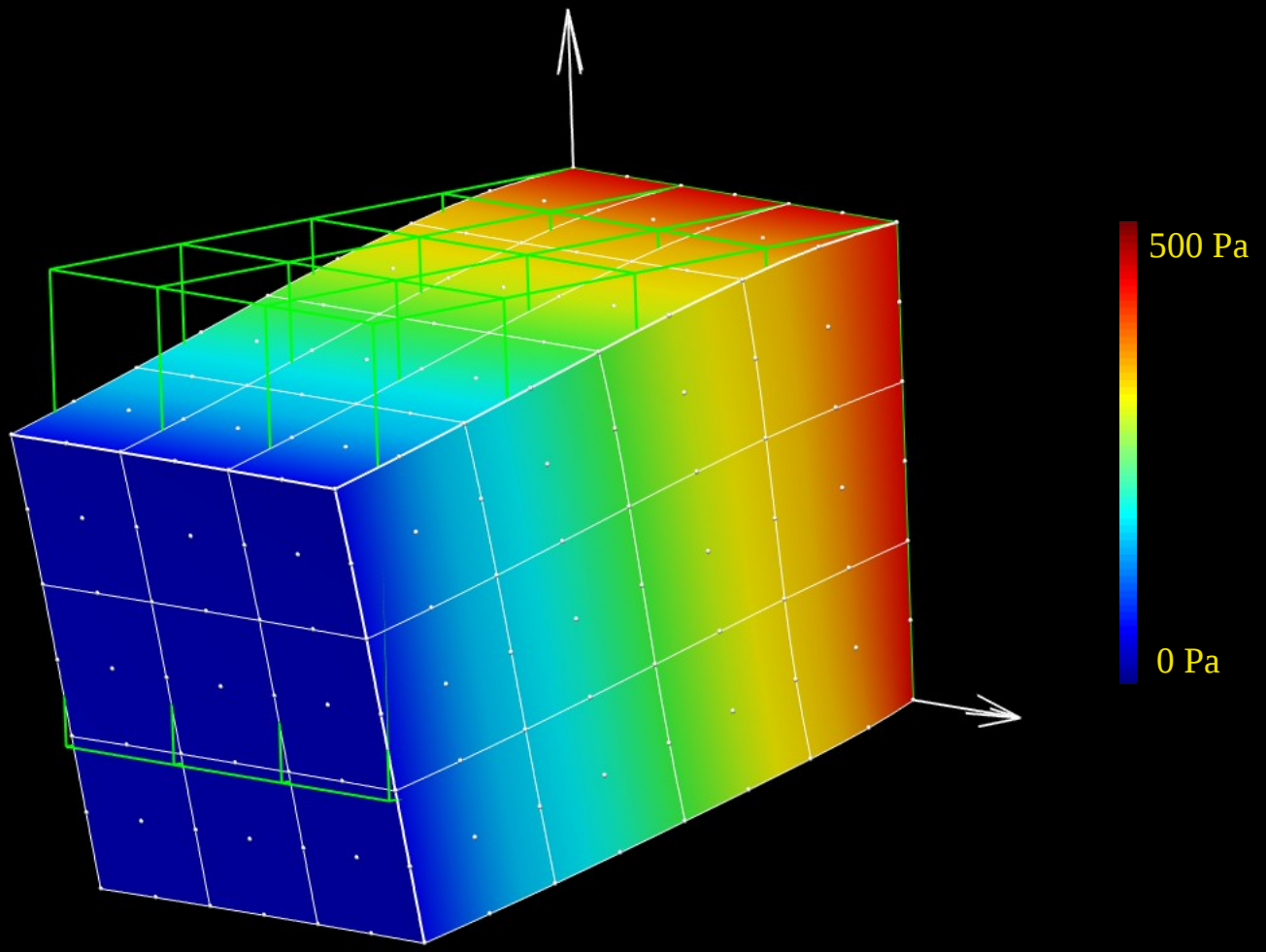
Coupled Elasticity/Elasticity Contact Mechanics



Courtesy of Nancy Yan

Contact between a stiff and soft material

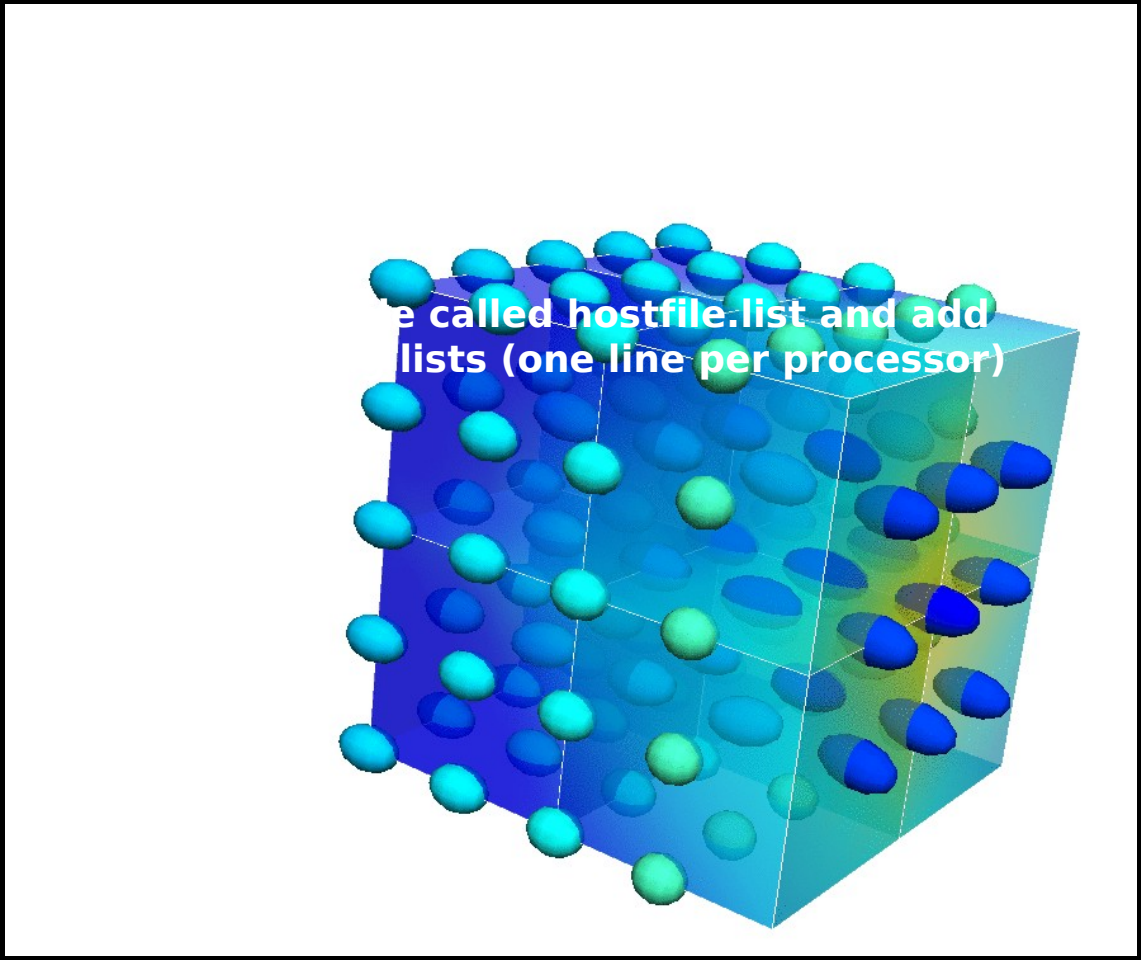
Coupled
Finite Elasticity-Darcy Flow



Courtesy of Adam Reeve

Gravity loading of a porous gel

Coupled Fluid/Mechanics ALE Navier-Stokes flow



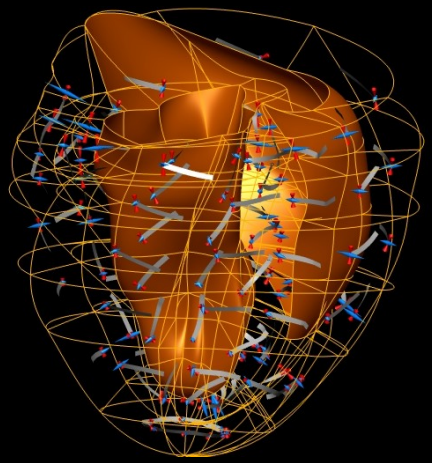
Courtesy of Sebastian Krittian

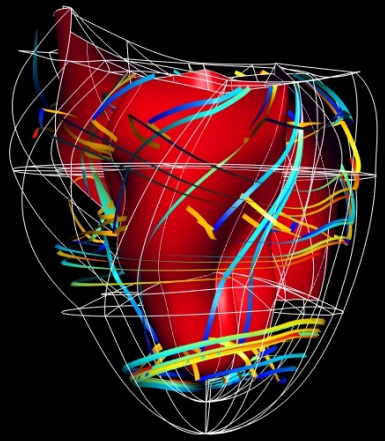
CMISS

- Biological modelling software developed at the University of Auckland, New Zealand
- **C**ontinuum **M**echanics, **I**maging, **S**ignal processing and **S**ystem identification
- **http://www.cmiss.org/**
- Two main components
  - Graphical front end – cmgui
  - Computational engine – cm

Graphical front end - cmgui

- Written in C/C++
- Cross platform e.g. wxwidgets
- Open source on sourceforge
  - http://sourceforge.net/projects/cmiss

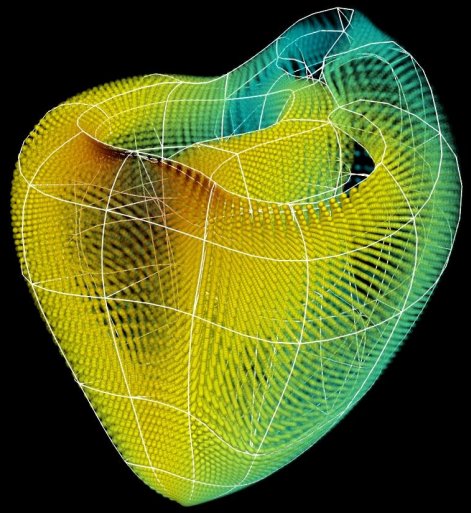




Computational engine - cm

- Developed by a large number of people over the last 30 years.
- Legacy Fortran 77 code.
- ~ 500,000 lines.
- Shared memory (OpenMP) parallelisation.
- Has some licensing limitations.
- Difficult to run large models due to
  - large memory footprint
  - Shared memory parallel speedup limitations.

Why OpenCMISS?



- We need to redevelop legacy CMISS computational code.
- We want to solve larger scale coupled models of the heart, lungs, stomach and other biological systems than we can currently solve
- Take advantage of modern High Performance computer architectures

OpenCMISS(Iron) Design Goals

1. It should be a library based rather than a monolithic application.
2. It should use the Physiome standards FieldML and CellML.
3. It should be a general code.
4. It should be an inherently parallel code.
5. It should be used, developed, and understood by novices and experts alike.
6. It should deal with multiscale and multiphysics applications and with complicated workflows.

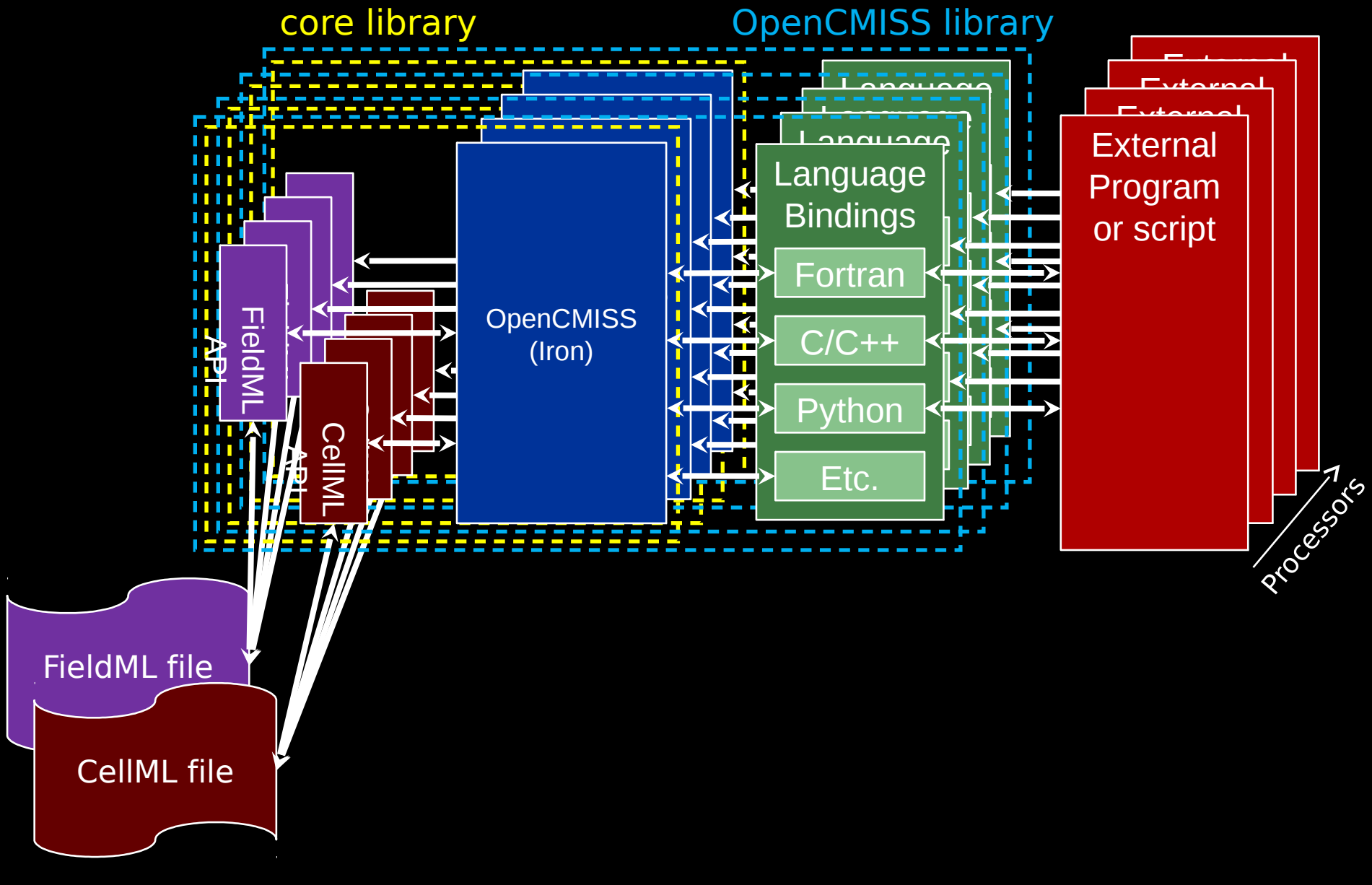
OpenCMISS(Iron)

- Re-engineering of CMISS(cm)
- Main website
  - http://www.opencmiss.org/
- Open source on github
  - https://github.com/organizations/OpenCMISS
- Fortran 95/2003.
- Object “based”

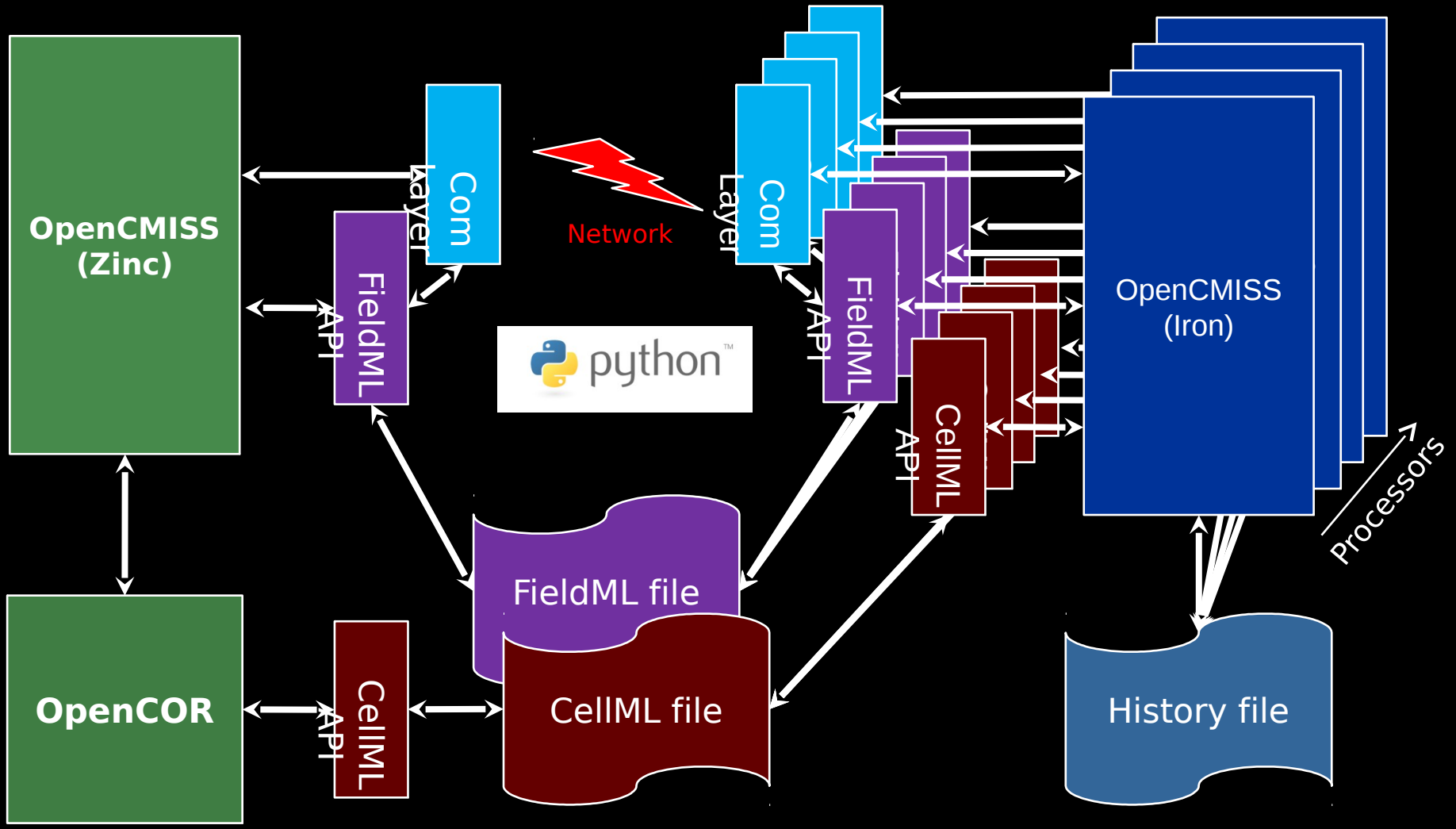
OpenCMISS(Iron) library

- OpenCMISS(Iron) is being developed as a library.
- Aim is to provide a number of bindings for various compiled or scripted languages i.e.,
  - Fortran
  - C/C++
  - Python
- Ultimate plan is to provide an open source OpenCMISS application for Physiome modelling by bringing together a number of tools.

OpenCMISS(Iron) library



OpenCMISS application



OpenCMISS I/O

- As a library, I/O is considered to be outside the scope of OpenCMISS.
- However, the OpenCMISS library will support the I/O of two file formats
  - FieldML
  - Used to read/write the field descriptions inside OpenCMISS.
  - CellML
  - Used to import CellML models as a means of evaluating the values of field degrees-of-freedom.

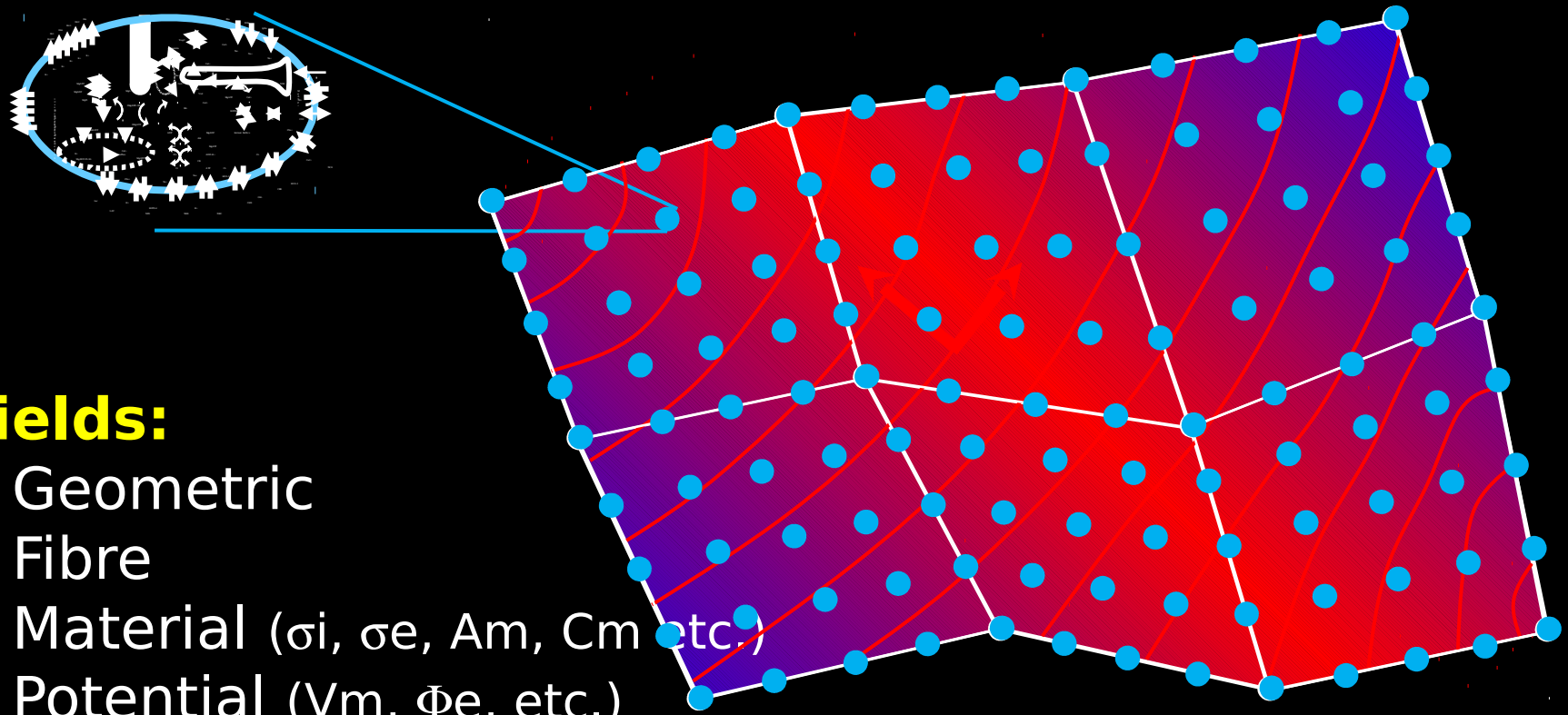
FieldML

- FieldML (Field Modelling/Markup Language) is a declarative language for building hierarchical models represented by generalized mathematical fields.
- Intended to provide a framework for modelling software development and model interchange for bioengineering and general engineering analysis communities.
- Still in an early state of development.
- http://www.fieldml.org/

CellML

- The purpose of CellML is to store and exchange computer-based mathematical models.
- Models are generally point based or “0D”.
- Extensive development of tools and model databases.
- http://www.cellml.org/
- Will interface with OpenCMISS by providing a general way of defining the value of a field degree of freedom via an arbitrary function or model.
- Models will be used to generate code and dynamically linked into OpenCMISS for performance.

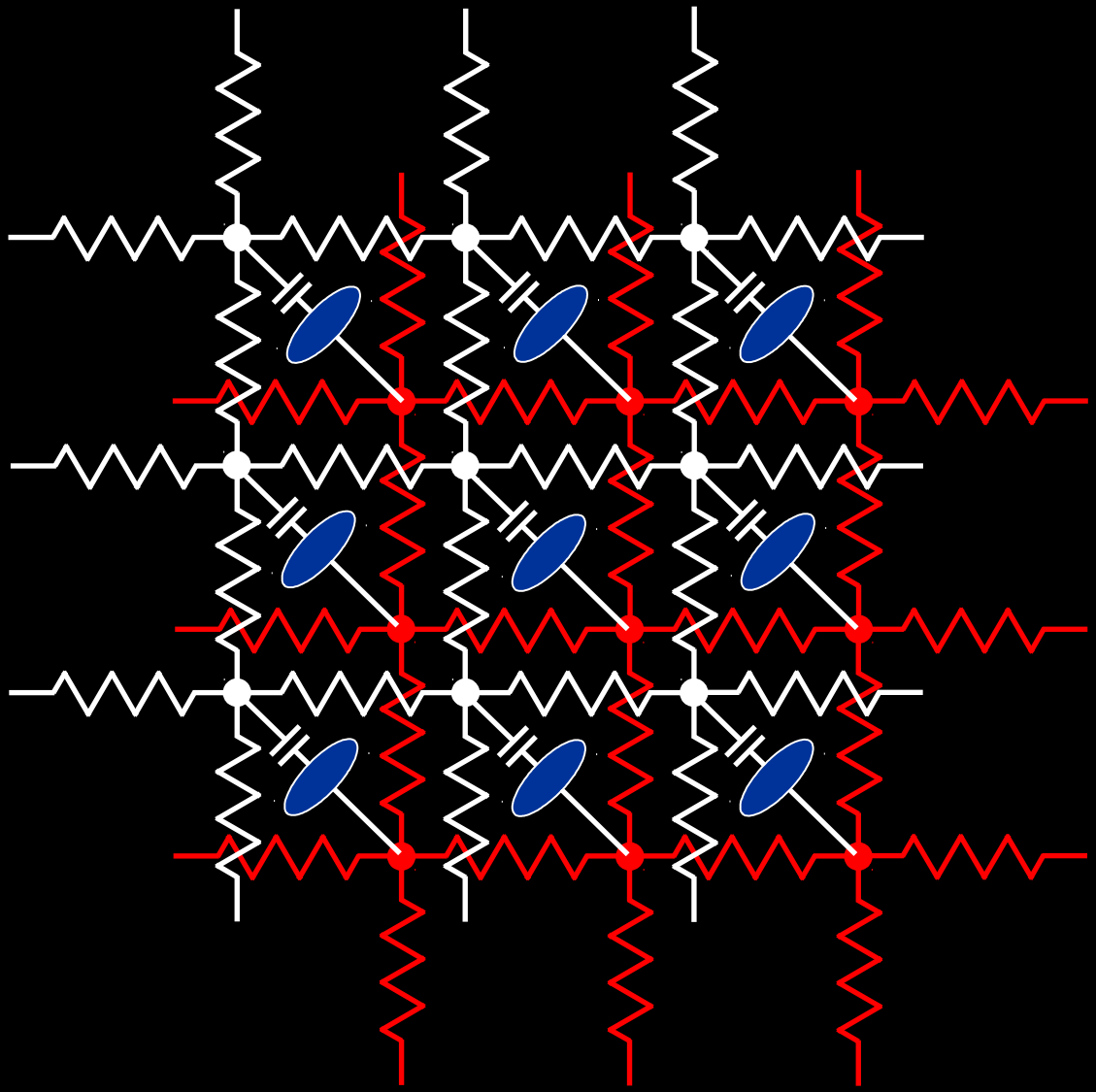
Activation problem



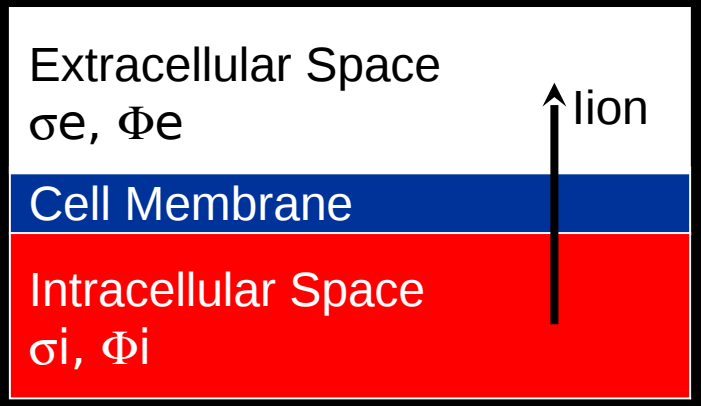
Fields:

- Geometric
- Fibre
- Material (σi, σe, Am, Cm etc.)
- Potential (Vm, Φe, etc.)
- Source (Iion, etc.)

Example – Electrical Activation (Bidomain representation)

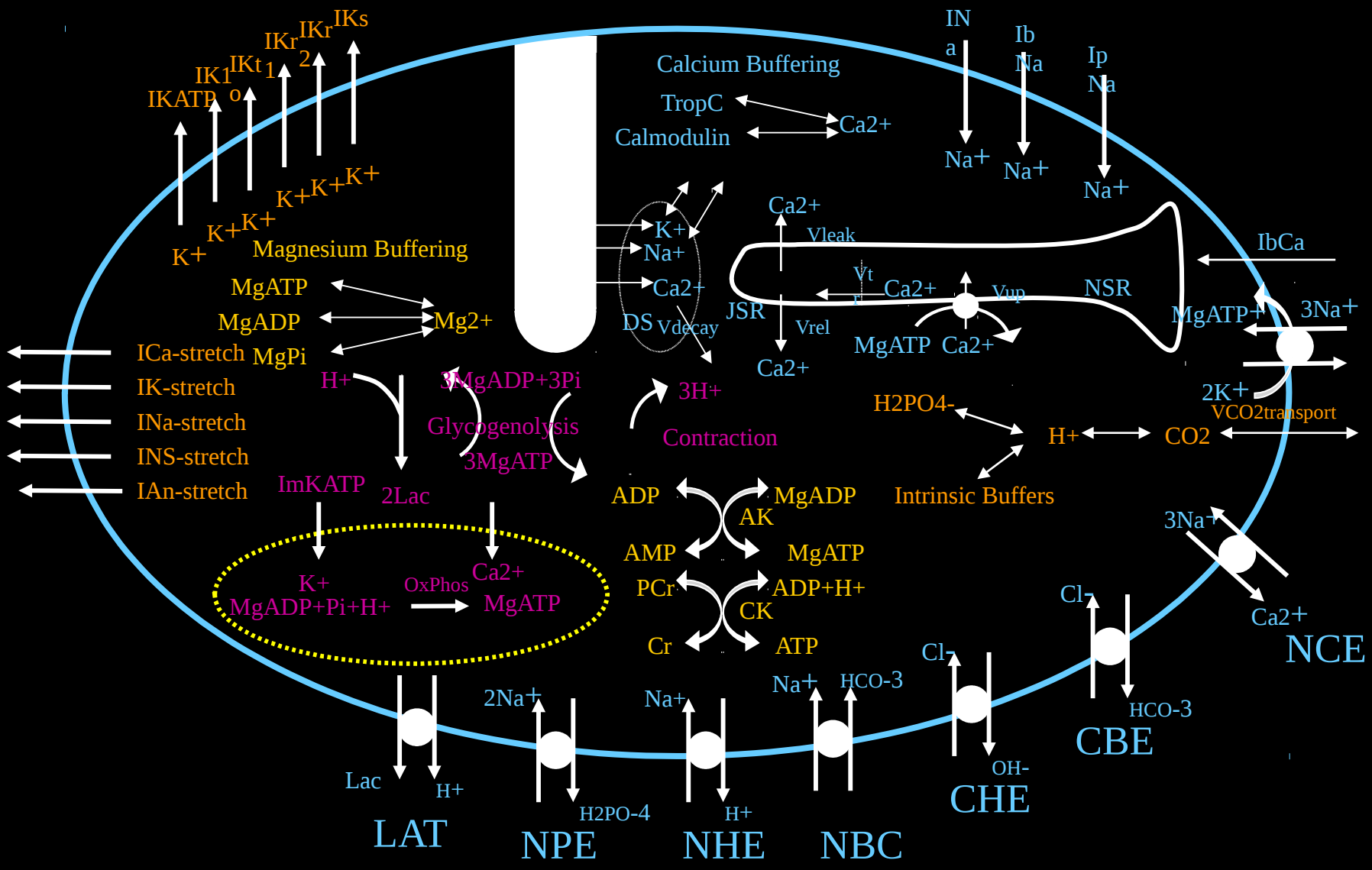


Cell model =



Bidomain equations

Cellular processes



BOTE problem size calculation

- Future electrical activation problem:
- Average human heart is approx 130 mm x 90 mm x 70 mm = 8.19x105 mm3. Assume 50% is ventricle.
- For 100 μm spacing would need 4.23x108 computational points (cp).
- For 30 ODEs/cp we would have 1.27x1010 ODEs to solve at each time instance.
- Assuming 100 flop/ODE we would have 1.27x1012 flop per time instance.
- A 1 ms time step will give 1.27x1015 flop/s real simulation time or 5.26x1016 flop/min real time.

Solution time calculation

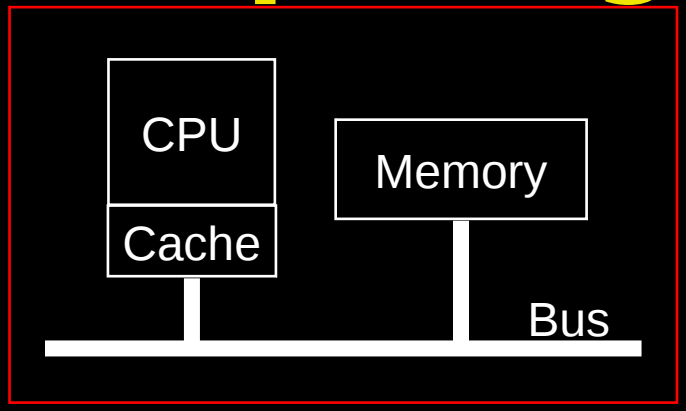
- A 2.8 GHz Intel Core i7 can achieve ~ 18.0 Gflops per core with linpack (best case!)
- To simulate 1 min real time on 1 processor would therefore take 2.92x106 s or 34 days!
- Or, in other words, to get the solution time down to 1 day would require 34 processors assuming perfect speedup (not very likely!)

- We are dealing with very, very big problems that require large parallel computers!

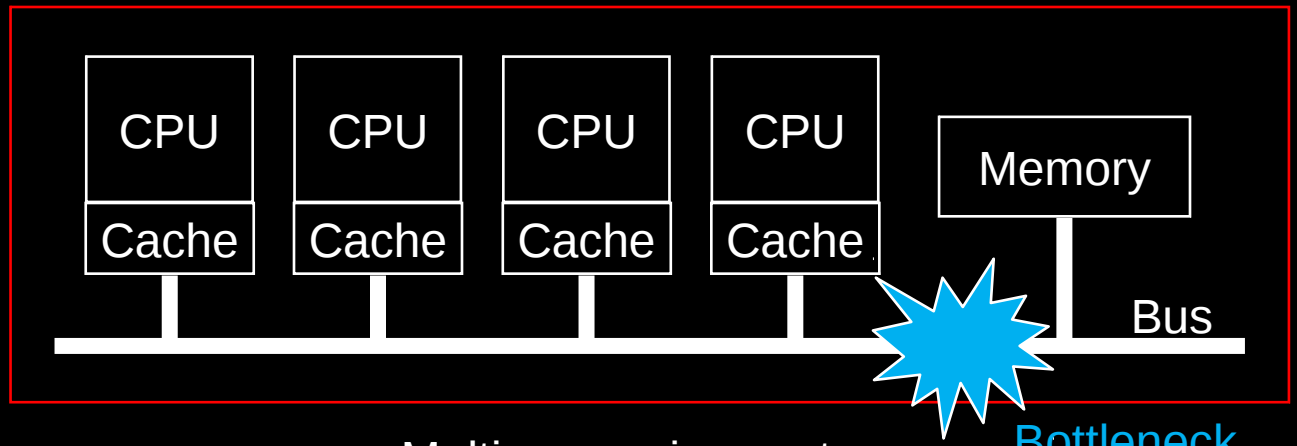
Amount of data

- From the previous example:
- For 100 μm spacing we have 4.23x108 computational points (cp).
- For 30 ODEs/cp each with just one state variable we have 1.27x1010 state variables to store at each time instance.
- To store the all in double precision would require 94.55 GB at each time instance.
- To store 1 minute of simulation at 1 ms time steps would require 5.4 PB!

Shared memory computing

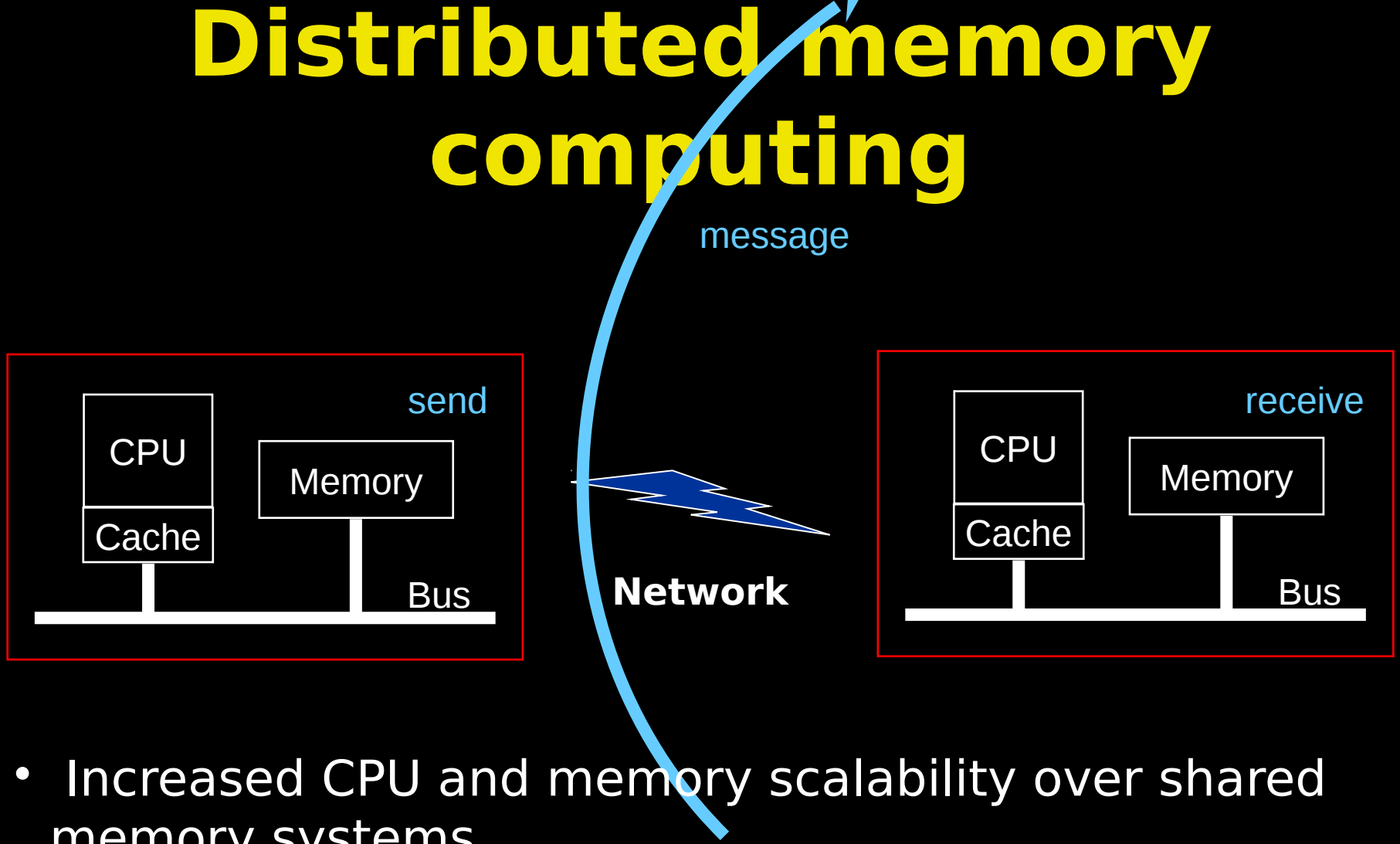


Single CPU system



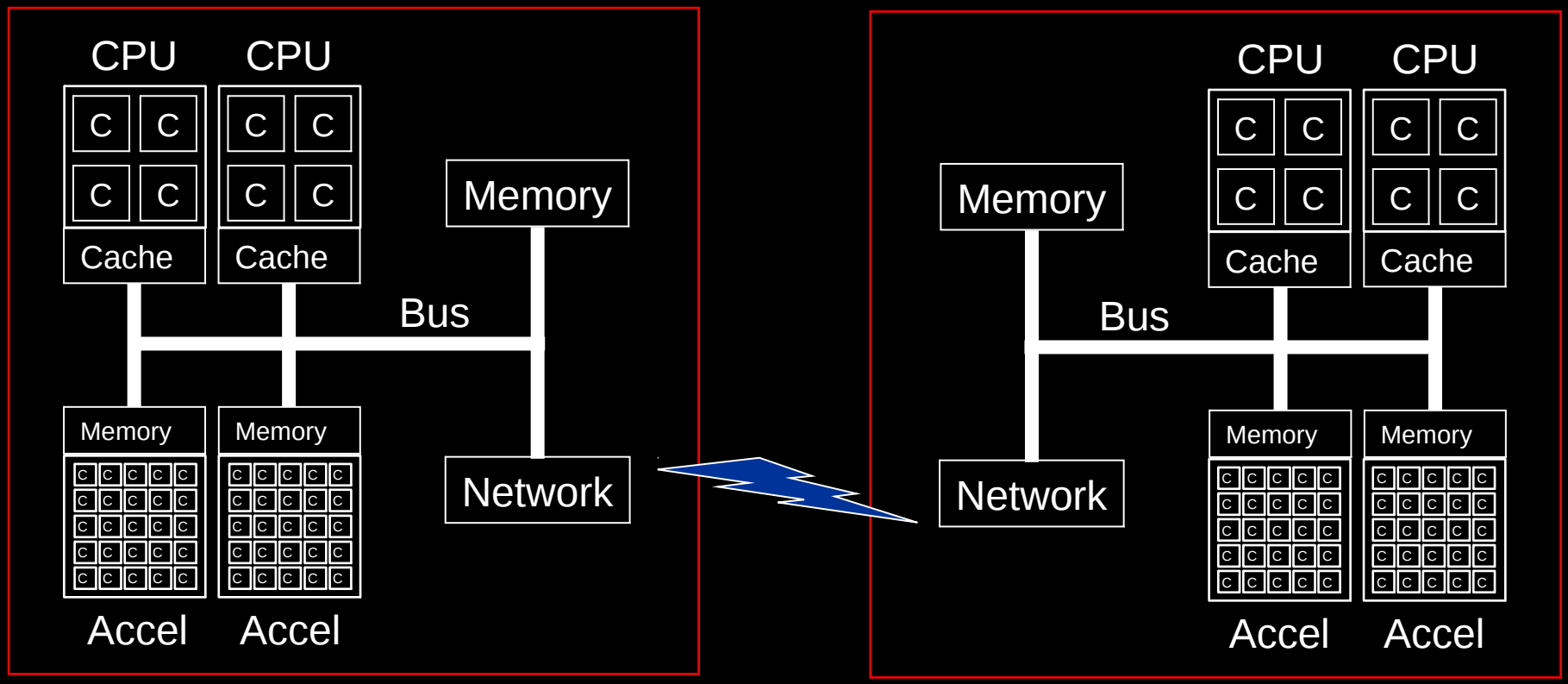
Multiprocessing system

Distributed memory computing



- Increased CPU and memory scalability over shared memory systems.
- Much harder to program!
- MPI standard to handle message parsing.

General parallel environment



Hierarchy of heterogeneous computing elements

Obtaining the Code

- We will now go through obtaining the code and setting up the development environment.
- Main source of information at

http://www.opencmiss.org

Click “Get support”, then “wiki” under “Documentation and source code”.

Obtaining the Code

- **Click on “Information for Users” followed by “Obtaining the code, setting up the development environment, and running the code”.**
- **Most of the information is for compiling OpenCMISSExtras (contains libraries that OpenCMISS uses).**
- **We will use a precompiled OpenCMISSExtras.**

Obtaining the Code

- **For Step 1 just type**

```
mkdir OpenCMISS
```

- **For Steps 2-3 we will use the OpenCMISSExtras located at**

```
/hpc/cmiss/OpenCMISSExtras_hpc5
```

- **Jump to Step 4.**

Programmer Script

If you use the bash shell you can do this by adding the following lines to your "~/.bashrc" file:

```
export OPENCMISS_ROOT=<path to your opencmiss folder>
export OPENCMISSEXTRAS_ROOT=/hpc/cmiss/OpenCMISSExtras_hpc5
export OPENCMISS_USE_GFORTRAN=true
export GNU_COMPILER_VERSION=4.6
if [ -x ${OPENCMISSEXTRAS_ROOT}/utils/scripts/opencmiss_programmer.sh ]; then
    . ${OPENCMISSEXTRAS_ROOT}/utils/scripts/opencmiss_programmer.sh
fi
```

If you use a c to tc shell you can do this by adding the following lines to your "/.cshrc" or "~/.tcshrc" file:

```
setenv OPENCMISS_ROOT <path to your opencmiss folder>
setenv OPENCMISSEXTRAS_ROOT /hpc/cmiss/OpenCMISSExtras_hpc5
setenv OPENCMISS_USE_GFORTRAN true
setenv GNU_COMPILER_VERSION 4.6
if ( -r ${OPENCMISSEXTRAS_ROOT}/utils/scripts/opencmiss_programmer.csh ) then
    source ${OPENCMISSEXTRAS_ROOT}/utils/scripts/opencmiss_programmer.csh
endif
```

Obtaining the Code

- We will now use hpc5 for the rest of the tutorial. SSH to hpc5 via

```
ssh –X hpc5.bioeng.auckland.ac.nz
```

- We are now in a position to obtain the code.
- Jump to step 9.
- Before we obtain the code let us review git.

Github

Go to http://www.github.com/

If you have an account log in.

If you don't have an account create one by clicking "Pricing and Signup". Then click "Create free account".

Now go to https://github.com/OpenCMISS

Git and DVC

GitHub

OpenCMISS

- cm
- examples
- cellml

Fork

Pull request

Your git repo

- cm
- examples
- cellml

Push

Clone

Local Filesystem

Code Changes

Commit

Your local copy

- cm
- examples
- cellml

Git config and cloning

To set up your git environment

```
git config –-global user.name “Your real name”
git config –-global user.email you@somedomain.com
git config –-global core.editor “gedit”
git config –-global colour.ui true
git config –-global http.proxy http://proxy.bioeng.auckland.ac.nz:8080/
```

To get the OpenCMISS code

```
mkdir OpenCMISS
cd OpenCMISS
git clone https://github.com/<username>/cm.git cm
git clone https://github.com/<username>/examples.git examples
git clone https://github.com/<username>/cellml.git cellml
```

Follow the remaining steps in Step 10.

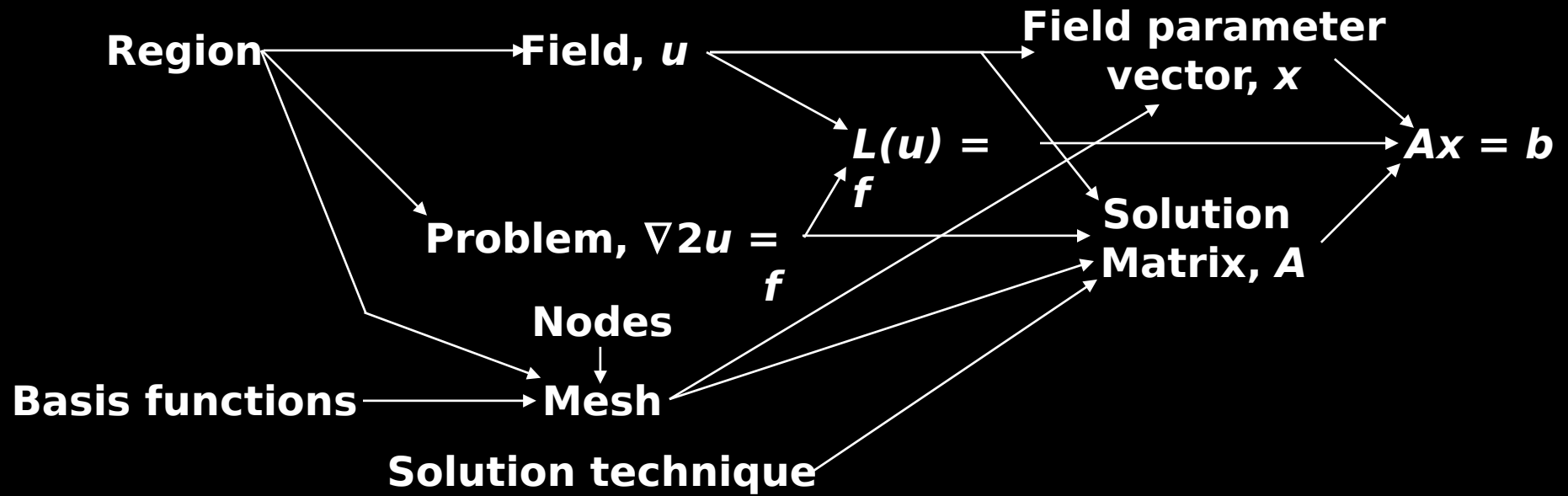
Environment Variables

Environment variables control the programmer setup options (see Step 4). The variables are:

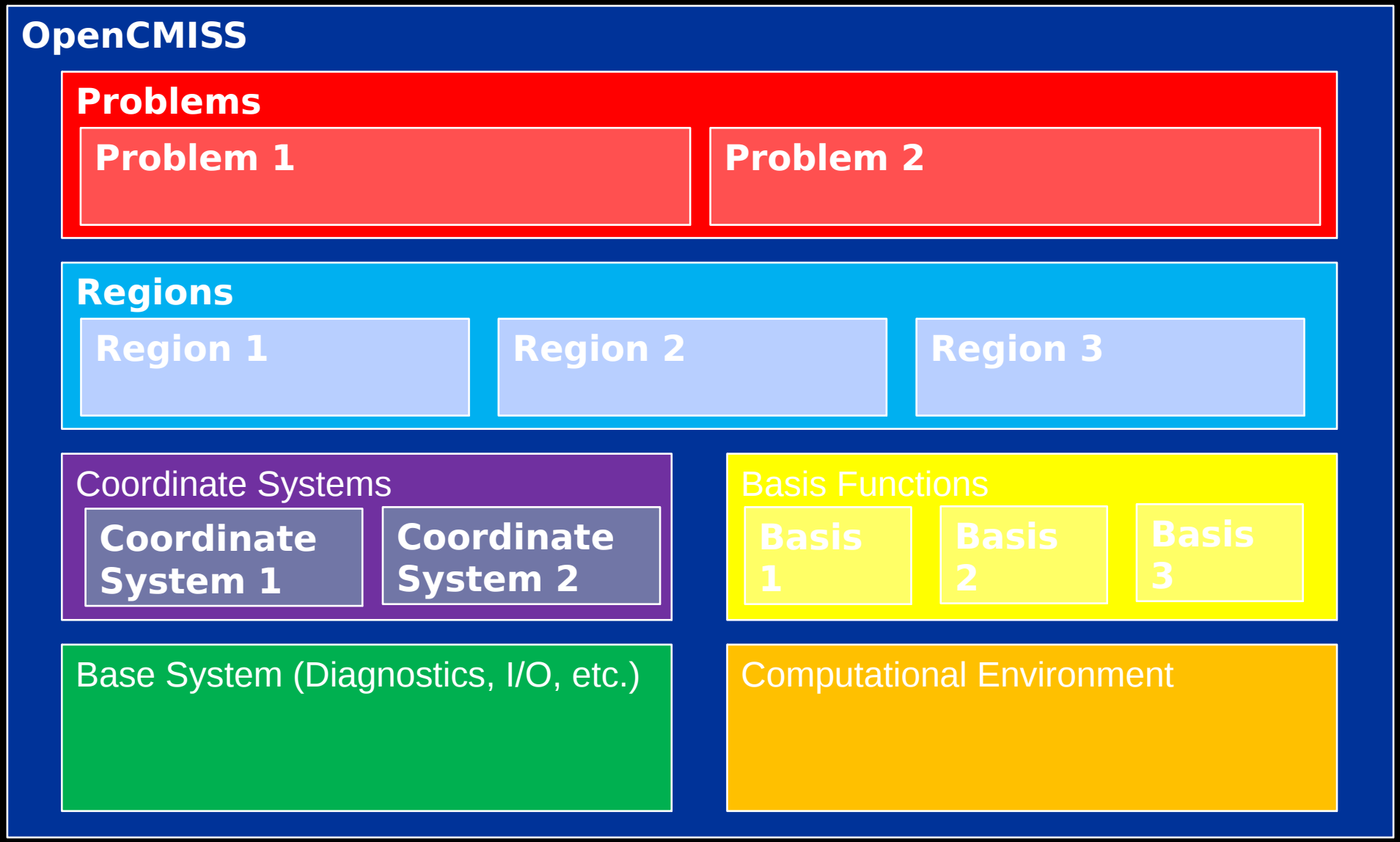
```
COMPILER
GNU_COMPILER_VERSION
OPENCMISSEXTRAS_RUN_CONFIG
MPI
USE_OPENCMISS_GFORTRAN
USE_OPENCMISS_GIT
USE_OPENCMISS_CMAKE
USECELLML
USEFIELDML
INTEL_MPI_VERSION
TOTALVIEW_PATH
TOTALVEW_VERSION
FLEXLM_VERSION
CUDA_ROOT
CUDA_SDK_ROOT
```

5-min break.

General problem overview



OpenCMISS top level



Basis functions

- Basis functions are the key item to specify the field approximation/interpolation and the linking of nodes and elements to form a mesh.
- Currently have two main types:
  - Lagrange-Hermite tensor product.
  - Linear->cubic Lagrange, cubic & quadratic Hermite.
  - Can be arbitrarily collapsed (two or more nodes in the same location) in any one direction or in any two directions to give a degenerate basis.
  - Simplex.
  - Line, triangular and tetrahedral elements.
  - Linear, quadratic or cubic.
  - Arbitrary Gaussian quadrature can integrate from 1st to 5th order (3rd order for lines at the moment).
  - Can only have the same order in each direction at the moment.
- Specifying a basis function automatically generates all necessary line and face basis functions as sub-bases of the basis function.

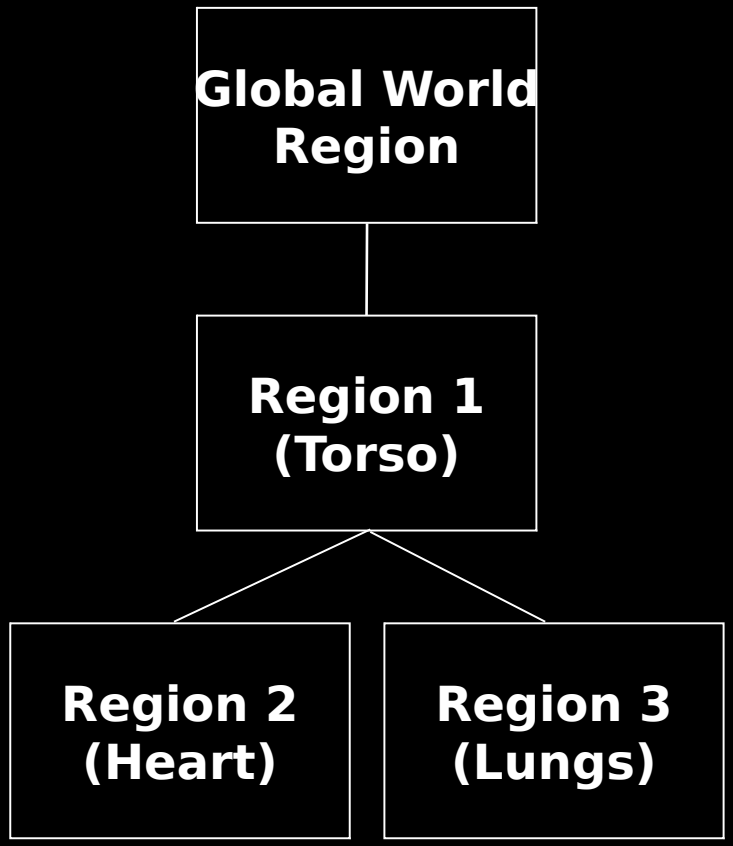
Coordinate systems

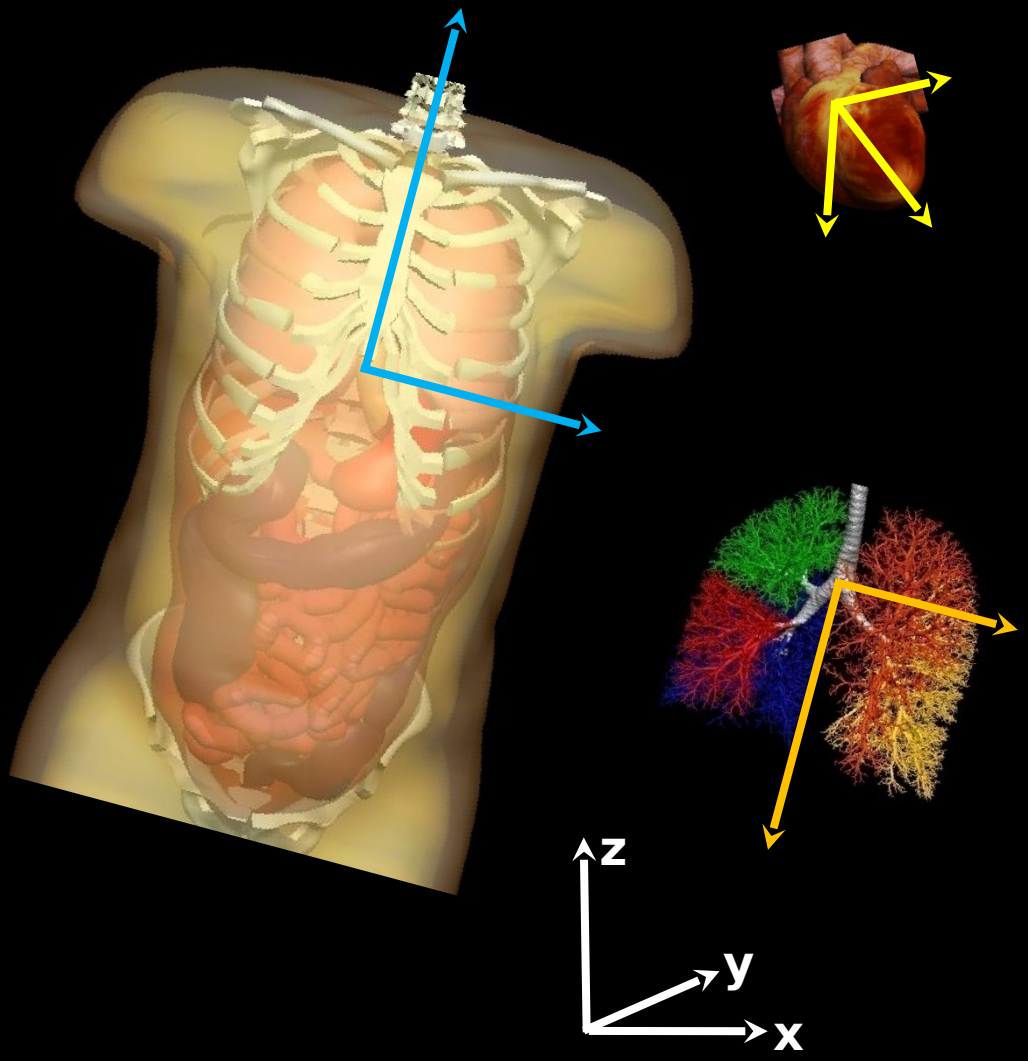
- Anchor the regions within the real world!
- There is a global (world) coordinate system aligned with 3D rectangular Cartesian space.
- Different types
  - Rectangular cartesian
  - Cylindrical polar
  - Spherical polar
  - Prolate spheroidal
  - Oblate spheroidal
- A number of todos!

Regions

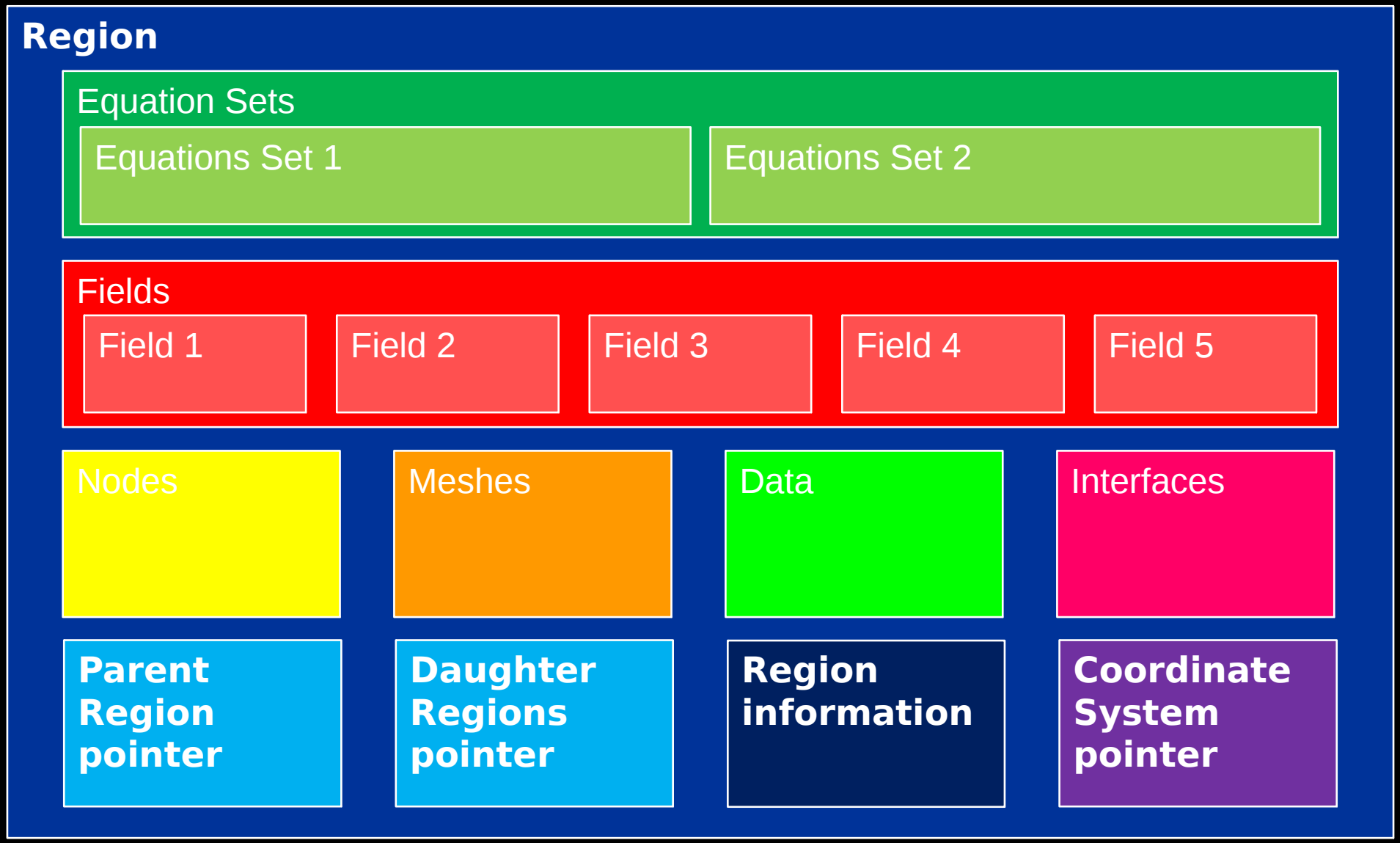
- Regions are one of the primary objects in openCMISS.
- Regions are hierarchical in nature in that a region can have one parent region and a number of daughter sub-regions.
- Daughter regions are related in space to parent regions by an origin and an orientation of the regions coordinate system.
- Daughter regions may only have the same or fewer dimensions as the parent region.
- There is a global (world) region (number 0) that has the global (world) coordinate system.

Regions


Global World Region
Region 1 (Torso)
Region 2 (Heart)
Region 3 (Lungs)


z
y
x

Region sub-objects

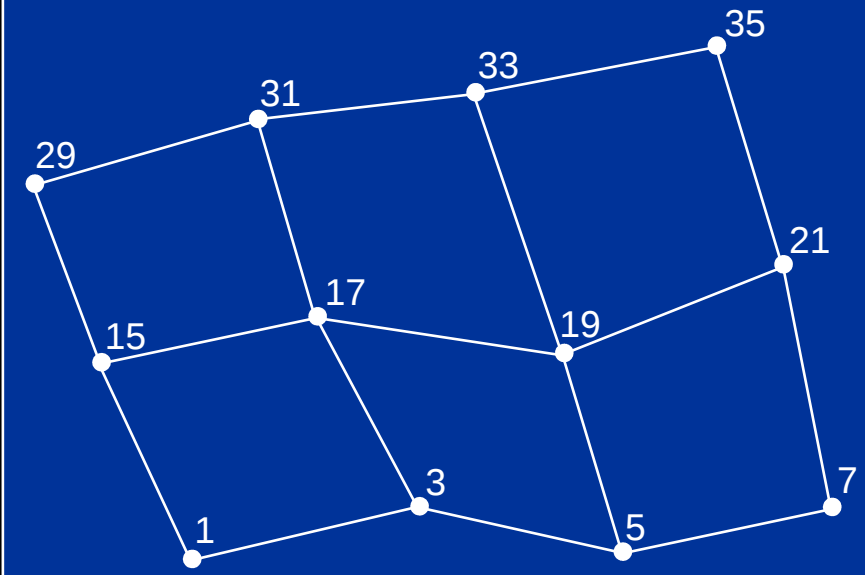


Meshes

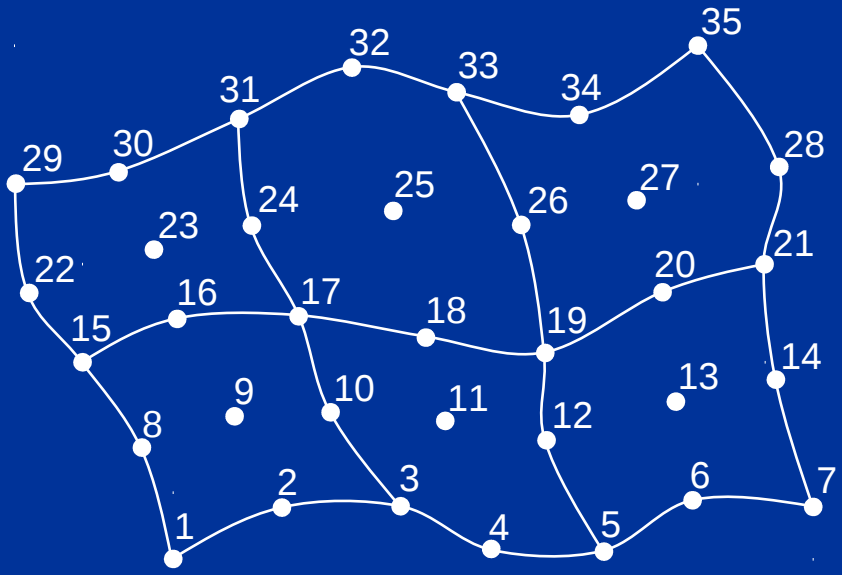
- Meshes are topological constructs within a region which fields use to define themselves.
- Meshes are made up of a number of mesh components.
- Mesh components are made up from nodes, elements and basis functions.
- A new mesh component is required for each different form of interpolation e.g., one mesh component is bilinear Lagrange and another is biquadratic Lagrange.
- All mesh components in a mesh must “conform”, that is have the same number of elements, Xi directions etc. etc.

Mesh components

Mesh

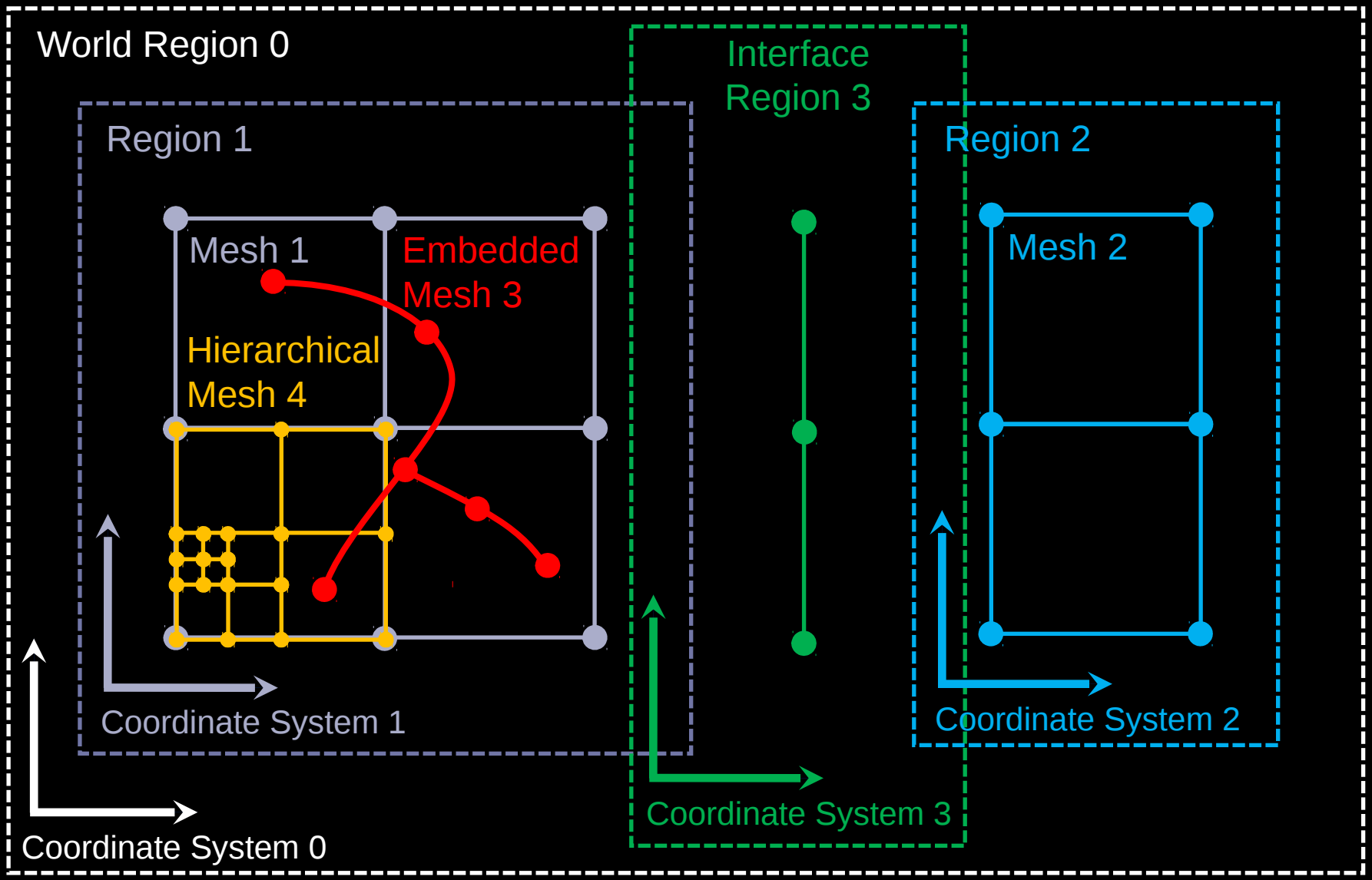


Component 1 –
bilinear Lagrange



Component 2 –
biquadratic Lagrange

Generalised Structure



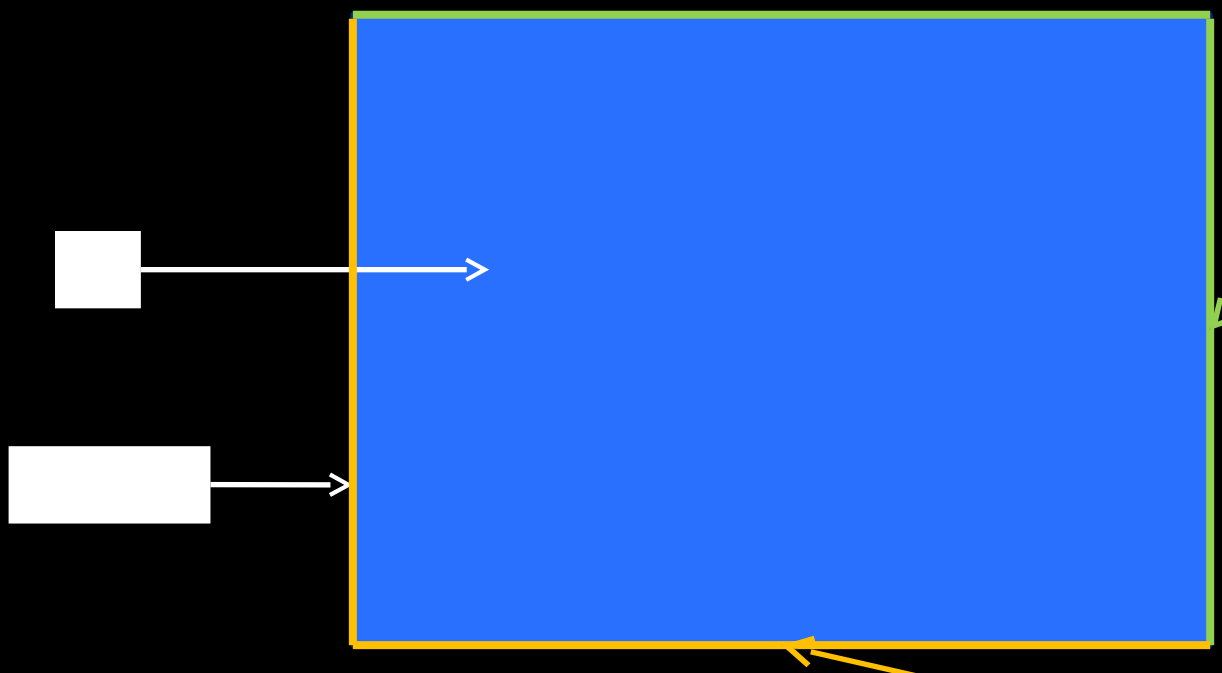
Let us look at some code

```
cd ${OPENCMISS_ROOT}/examples/ClassicalField/Laplace/Laplace
```

- Choose what language Fortran/C/Python you are most comfortable with and change directory to that language directory.
- Edit either
  - src/FortranExample.f90
  - src/CExample.f90
  - LaplaceExample.py

Laplace's equation

$$\phi(\boldsymbol{x}) = \phi_d(\boldsymbol{x}), \boldsymbol{x} \in \Gamma_d$$



$$q(\boldsymbol{x}) = \nabla\phi(\boldsymbol{x}) \cdot \boldsymbol{n} = q_n(\boldsymbol{x}), \boldsymbol{x} \in \Gamma_n$$

Object interface

- How to define objects for the library?
- Want:
  - Simple interface for a variety of scripting and programming languages.
  - User friendly in that the library should sensibly set default parameters to minimise the amount of information the programmer/user has to send.
  - Should be extensible so that extra parameters can be added at a later stage without causing problems.

Object interface

- Objects identified by a unique user number.
- Objects are initialised with a `Object_TypeInitialise` call.
- Objects are finalised with a `Object_TypeFinalise` call.
- Objects are created with a pair of `Object_CreateStart` and `Object_CreateFinish` calls.
- Objects are destroyed with a `Object_Destroy` call.
- Object parameters are set with a number of `Object_ParameterSet` calls inside the START and FINISH calls.
- START call initialises OBJECT and sets default parameters. SET calls modify default parameters. FINISH call finalises the object.

Fortran Example, basis functions

```
TYPE(CMISSBasisType) :: Basis
!Initialise the basis type.
CALL CMISSBasis_Initialise(Basis,Err)
!Start the creation of a basis with a user number of 1.
!The default is tri-linear Lagrange
CALL CMISSBasis_CreateStart(1,Basis,Err)
!Set the number of xi directions to 2.
CALL CMISSBasis_NumberOfXiSet(Basis,2,Err)
!Set the interpolation to be cubic Hermite*quadratic Lagrange.
CALL CMISSBasis_InterpolationXiSet(Basis, &
 & [CMISSCubicHermiteInterpolation, &
 & CMISSQuadraticLagrangeInterpolation],Err)
!Finish the creation of the basis.
CALL CMISSBasis_CreateFinish(Basis,Err)
!Destroy the basis with the user number 1.
CALL CMISSBasis_Destroy(Basis,Err)
!Finalise the basis functions.
CALL CMISSBasis_Finalise(Basis,Err)
```

C Example, basis functions

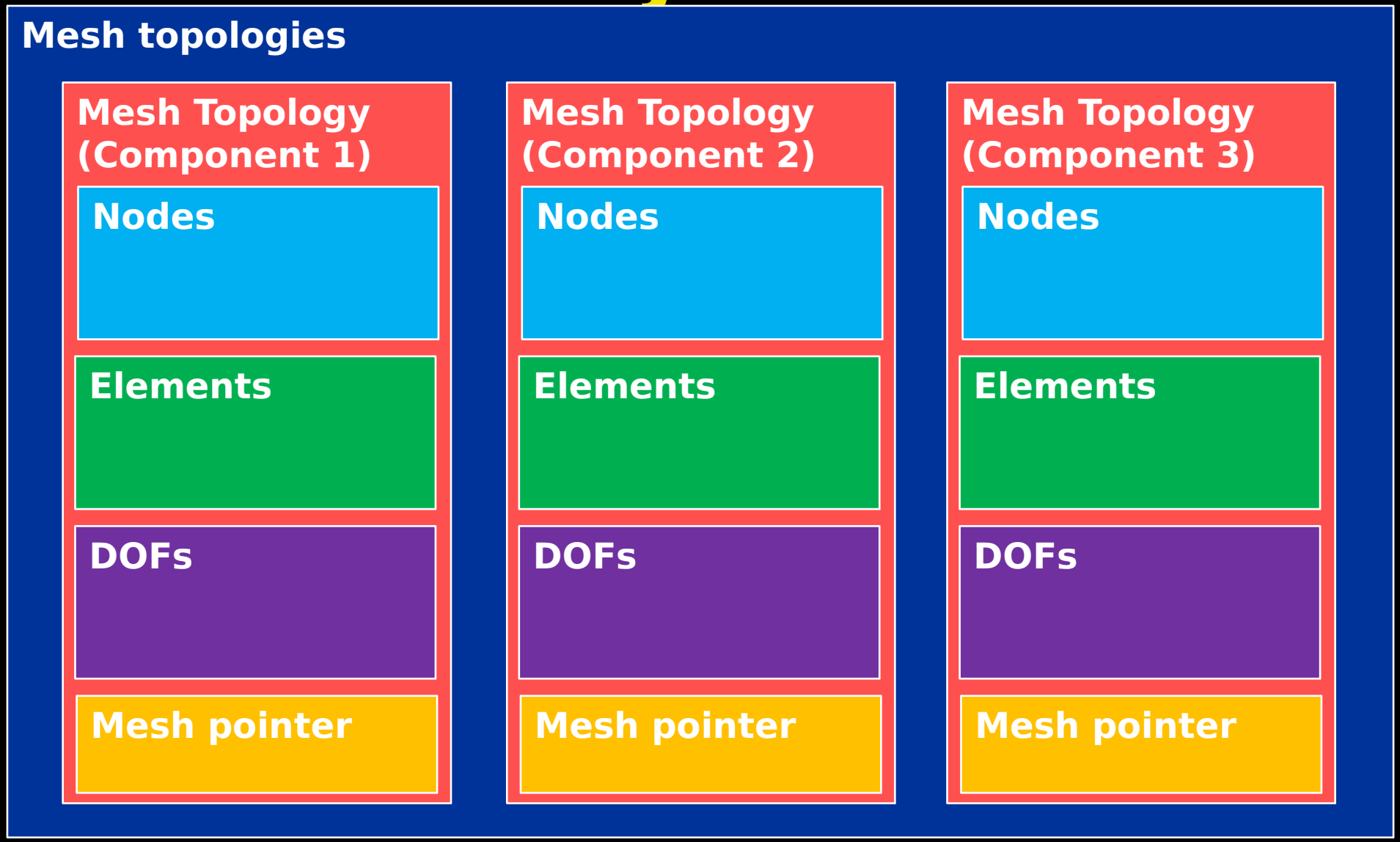
```
CMISSBasisType Basis = (CMISSBasisType)NULL;
/* Initialise the basis type */
Err = CMISSBasis_Initialise(&Basis);
/* Start the creation of a basis with a user number of 1.
   The default is tri-linear Lagrange */
Err = CMISSBasis_CreateStart(1,Basis);
/* Set the number of xi directions to 2 */
Err = CMISSBasis_NumberOfXiSet(Basis,2);
/* Set the interpolation to be cubic Hermite*quadratic Lagrange */
BasisInterpolation[0]=CMISS_BASIS_CUBIC_HERMITE_INTERPOLATION;
BasisInterpolation[1]=CMISS_BASIS_LINEAR_LAGRANGE_INTERPOLATION;
Err = CMISSBasis_InterpolationXiSet(Basis,2,BasisInterpolation);
/* Finish the creation of the basis */
Err = CMISSBasis_CreateFinish(Basis);
/* Destroy the basis with the user number 1 */
Err = CMISSBasis_Destroy(Basis);
/* Finalise the basis functions */
Err = CMISSBasis_Finalise(&Basis);
```

Python, basis functions

```
# Initialise the basis type
basis = CMISS.Basis()
# Start the creation of a basis with a user number of 1.
# The default is tri-linear Lagrange
basis.CreateStart(1)
# Set the number of xi directions to 2
basis.NumberOfXiSet(2)
# Set the interpolation to be cubic Hermite*quadratic Lagrange
basis.interpolationXi=[CMISS_BASIS_CUBIC_HERMITE_INTERPOLATION,
   CMISS_BASIS_LINEAR_LAGRANGE_INTERPOLATION]
# Finish the creation of the basis
basis.CreateFinish()
# Destroy the basis
basis.Destroy();
# Finalise the basis functions
basis.Finalise();
```

Review Example

Mesh topologies sub-objects



Memory issues

- Most computational nodes used in distributed computing do not have a large amount of memory per node.
- If running on a large “constellation type” machine it would be inefficient to allocate the whole problem multiple times on each node.
- Want to be able to just allocate what the computational node needs to do its calculations.

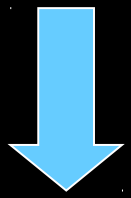
- Require a distributed mesh and problem.

Memory issues II

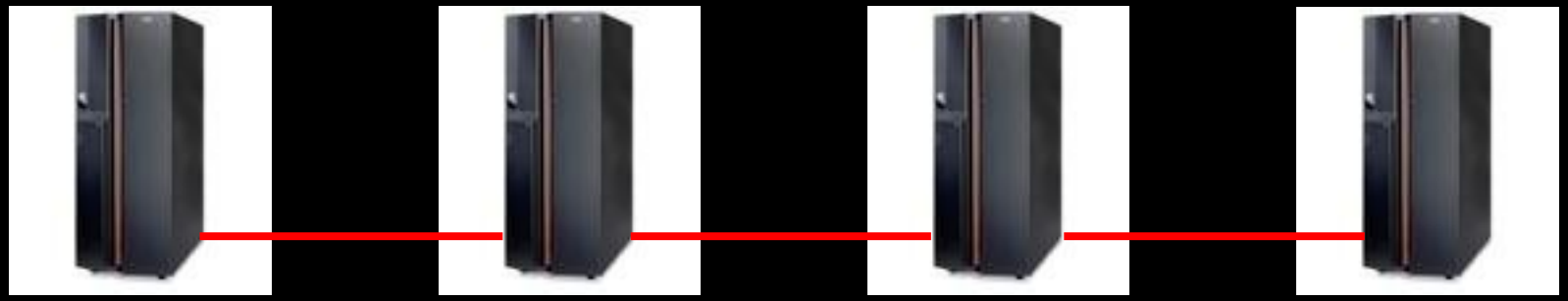
- Computer architectures moving towards multiple cores per CPU/Memory unit.
- Multiple cores are equivalent to mini shared memory systems. Similar memory efficiency problems to constellation type systems.
- Need to still allow for shared memory programming for some problems e.g., BEM.

- Aim for a general $n_x p(n) x e(p)$ parallel environment where n=number of computation nodes, p(n) is the number of processors per node and e(p) the number of accelerators per processor

Parallelisation?



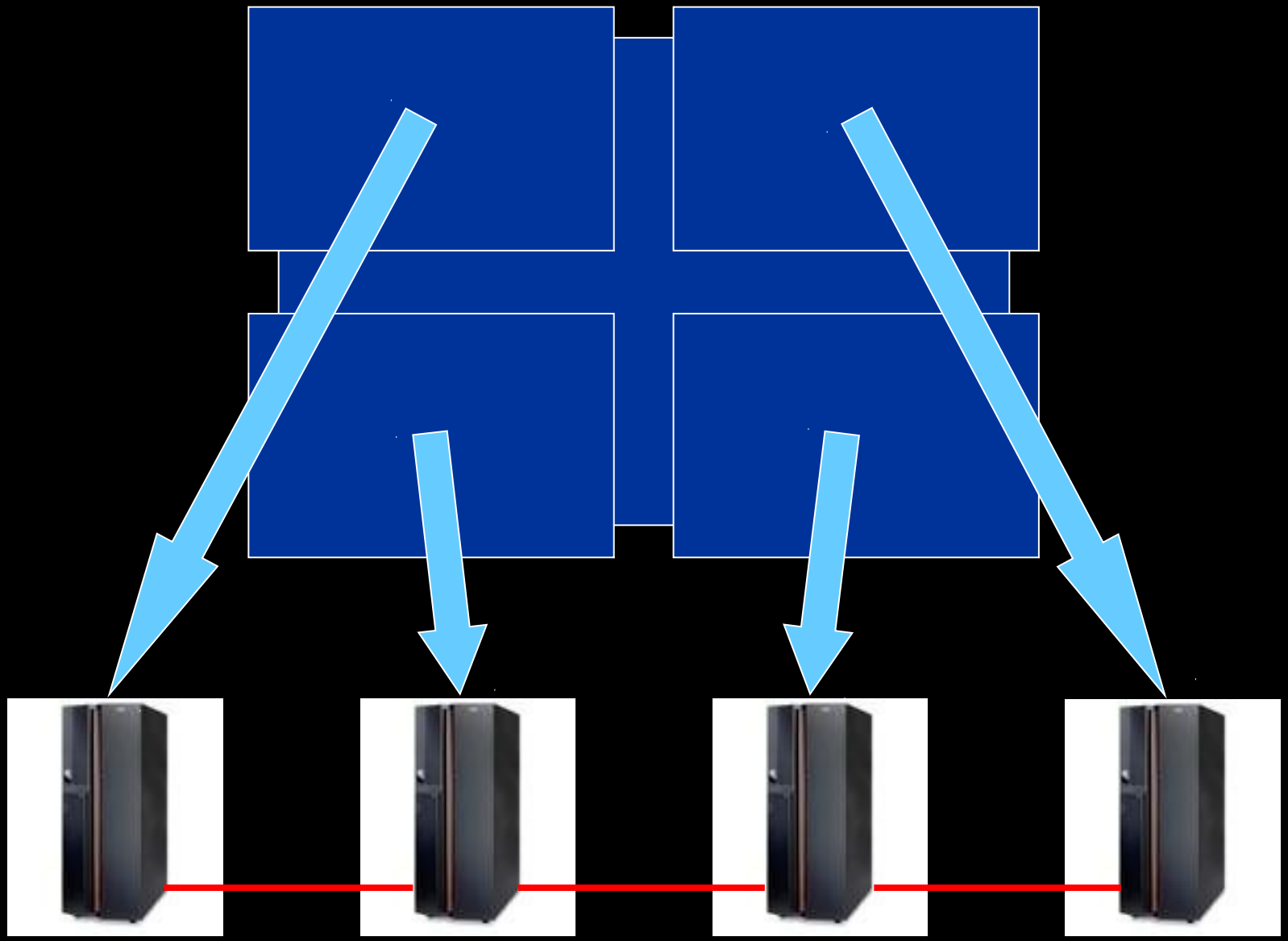
?



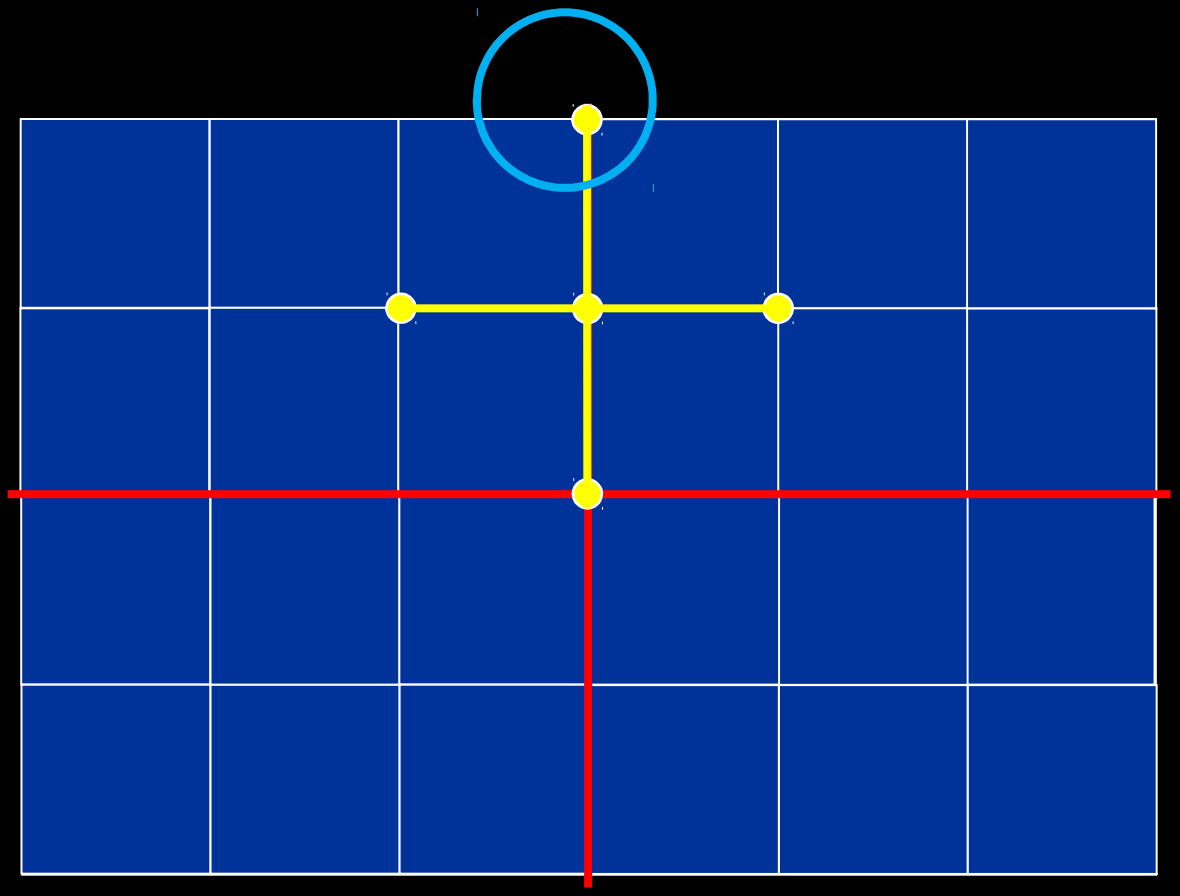
Parallelisation strategies

- Data orientated (vector)
- Producer/consumer (or master/slave)
- Domain decomposition
- Which one???
- Work-to-communication ratio.
  - Typical network time period ~ ms.
  - Typical computer clock period ~ ns.
- Need to maximise computational work to communication ratio

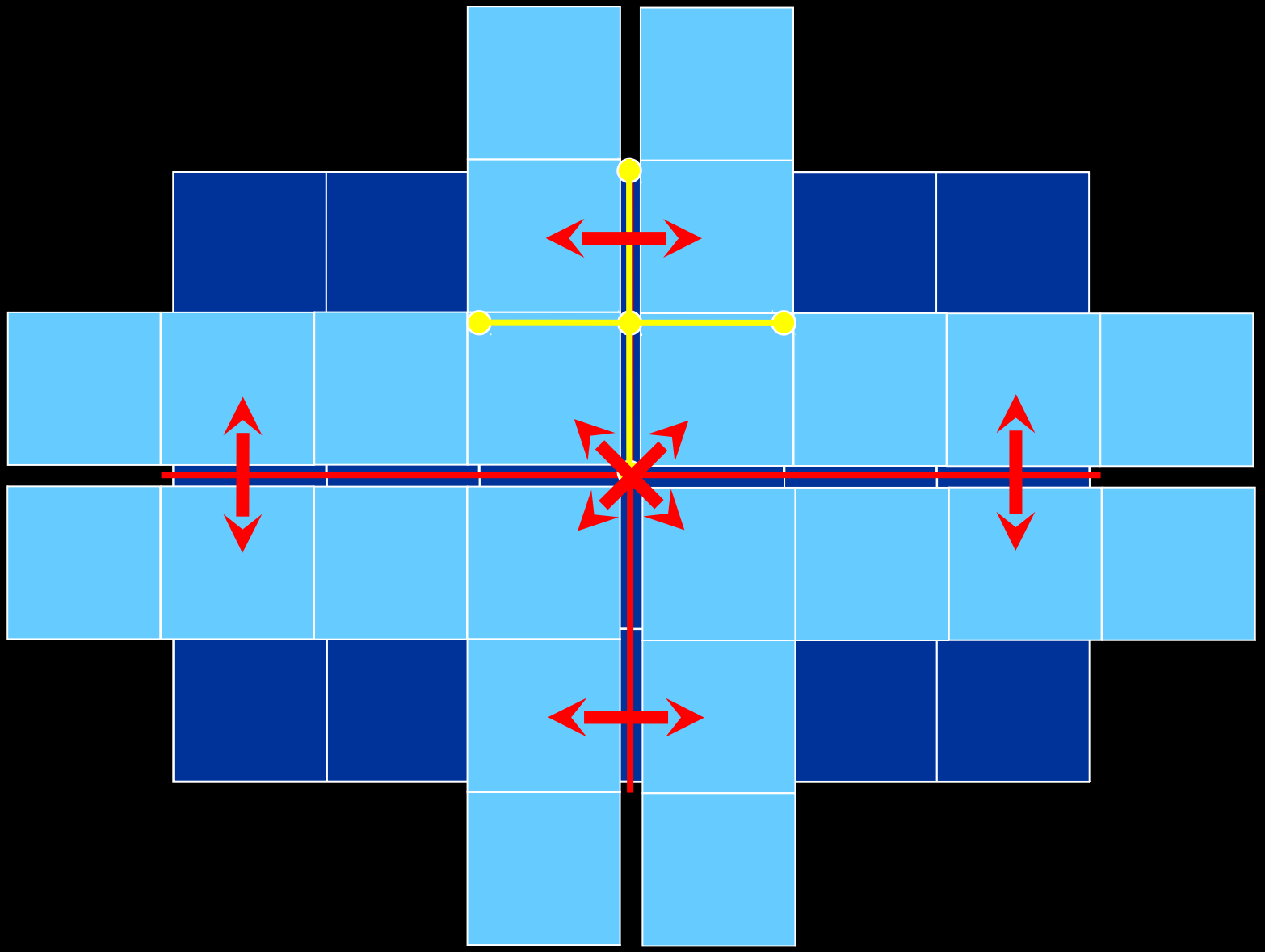
Domain decomposition



Domain decomposition



Ghosting



Mesh decomposition

- How to break up the mesh?
- We desire:
  - Roughly equal “sizes” to allow for good load balancing (may need to “weight” sizes to allow for different computational loads).
  - We want to minimise the amount of “cuts” or breaks in the mesh as this will minimise the communication between parts of the broken mesh.

- Use the ParMETIS parallel graph partitioning library to break up the mesh into a number of domains.

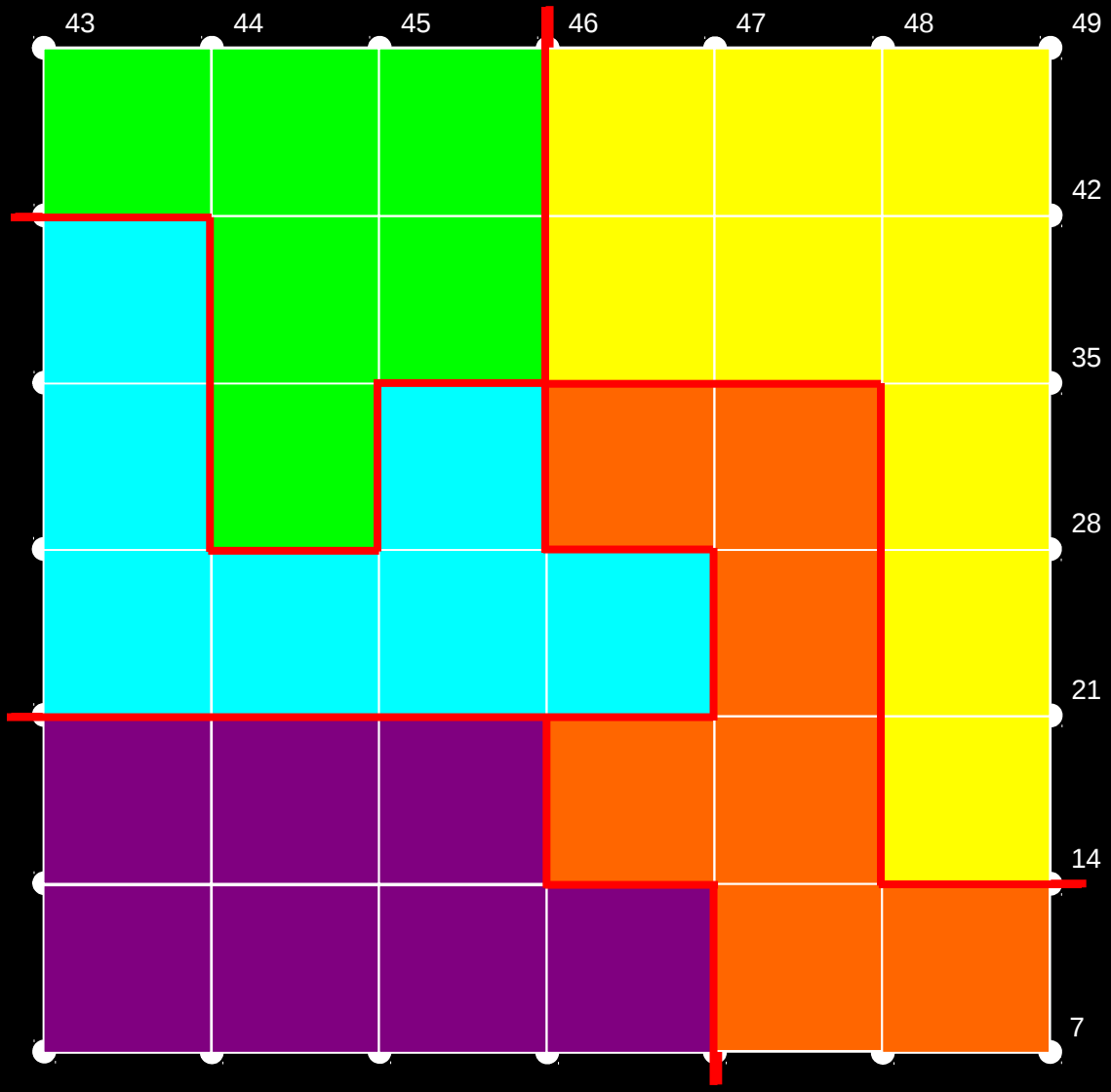
Mesh decomposition

Break the mesh into 5 pieces by element?

Graph of the dual of the mesh

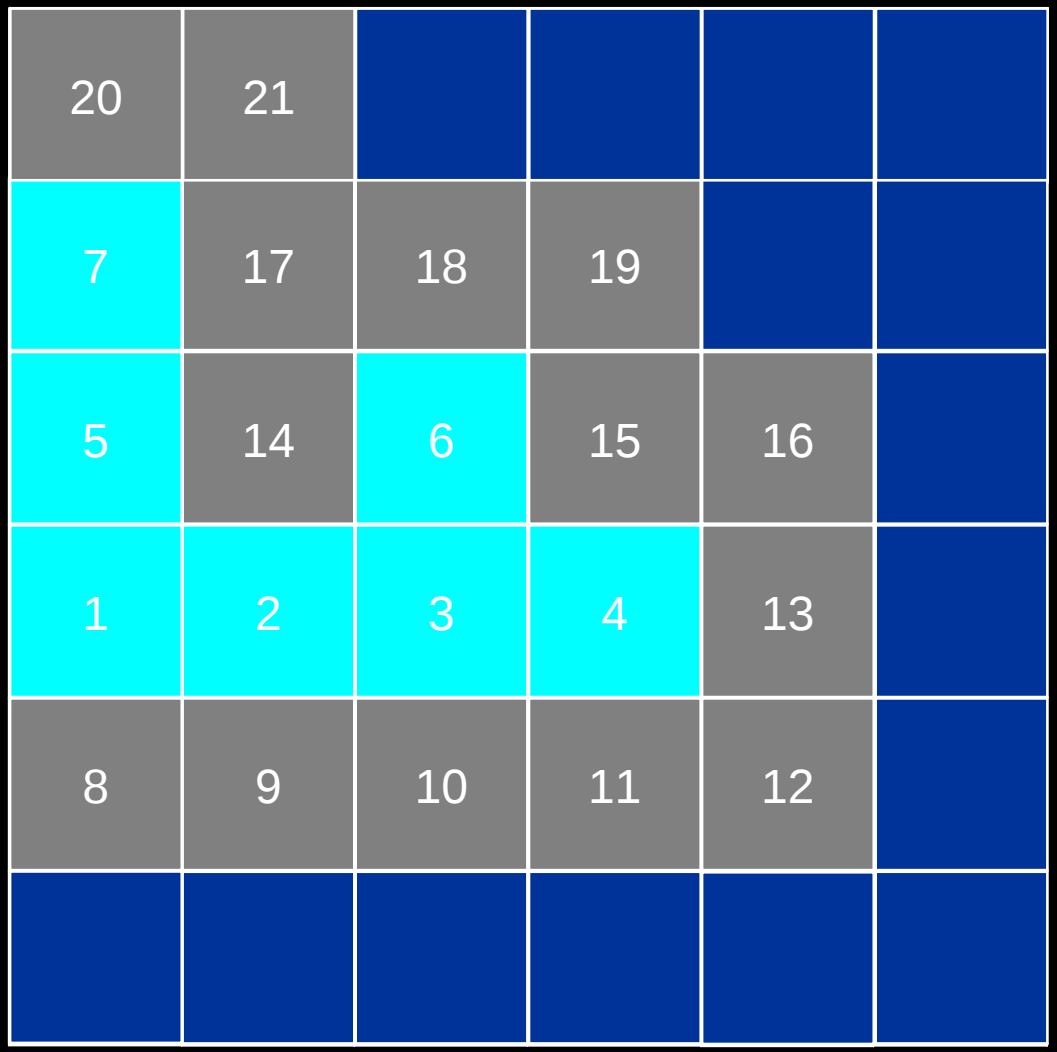
Graph cuts

Domains



Mesh numbering

Local Mesh Numbering



Mesh sub-objects

Mesh

Mesh topologies

- Mesh Topology (Component 1)
- Mesh Topology (Component 2)
- Mesh Topology (Component 3)

Decompositions

- **Decomposition 1**
- **Decomposition 2**

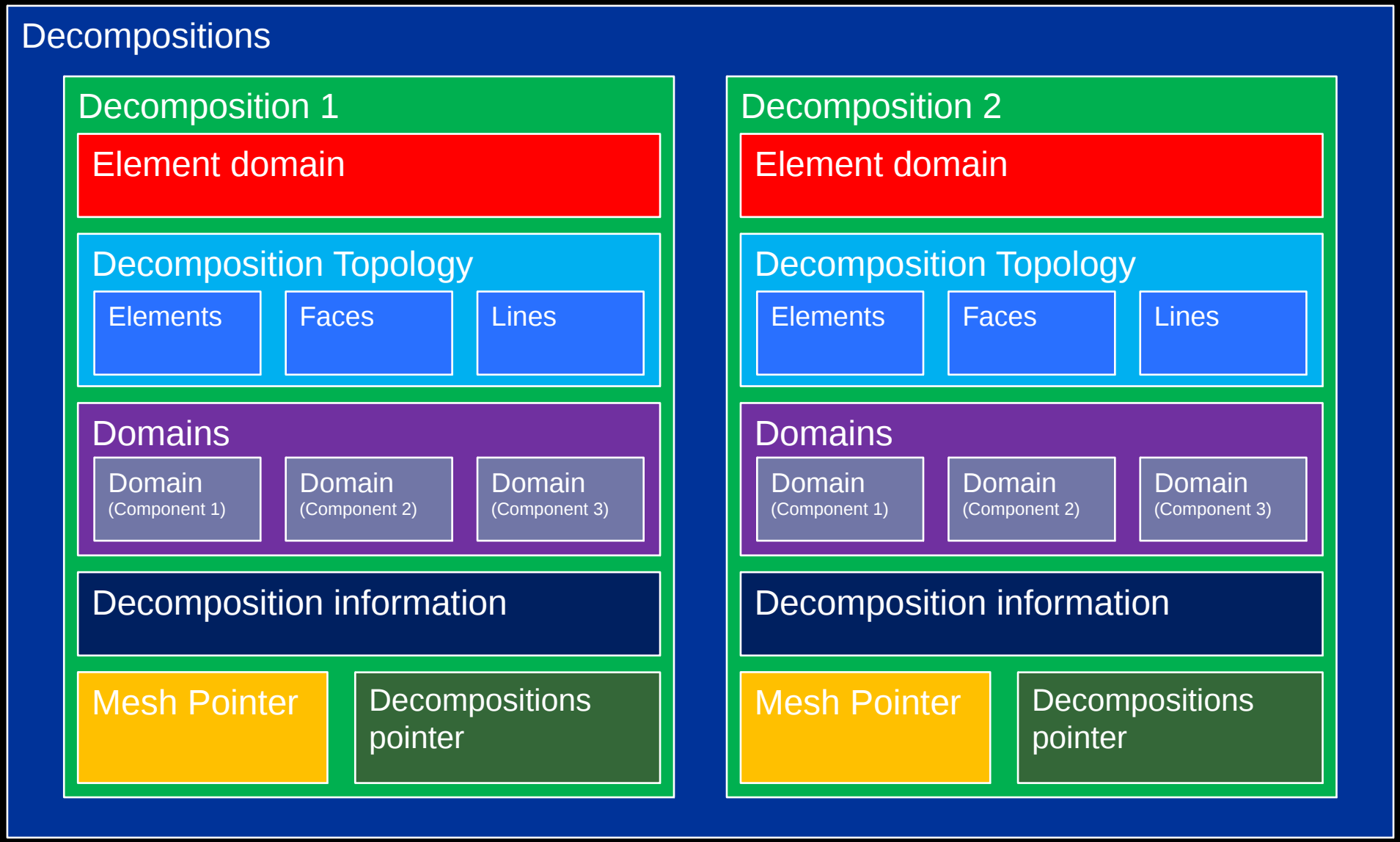
Region pointer

Embedded Mesh/Hierarchical Mesh information

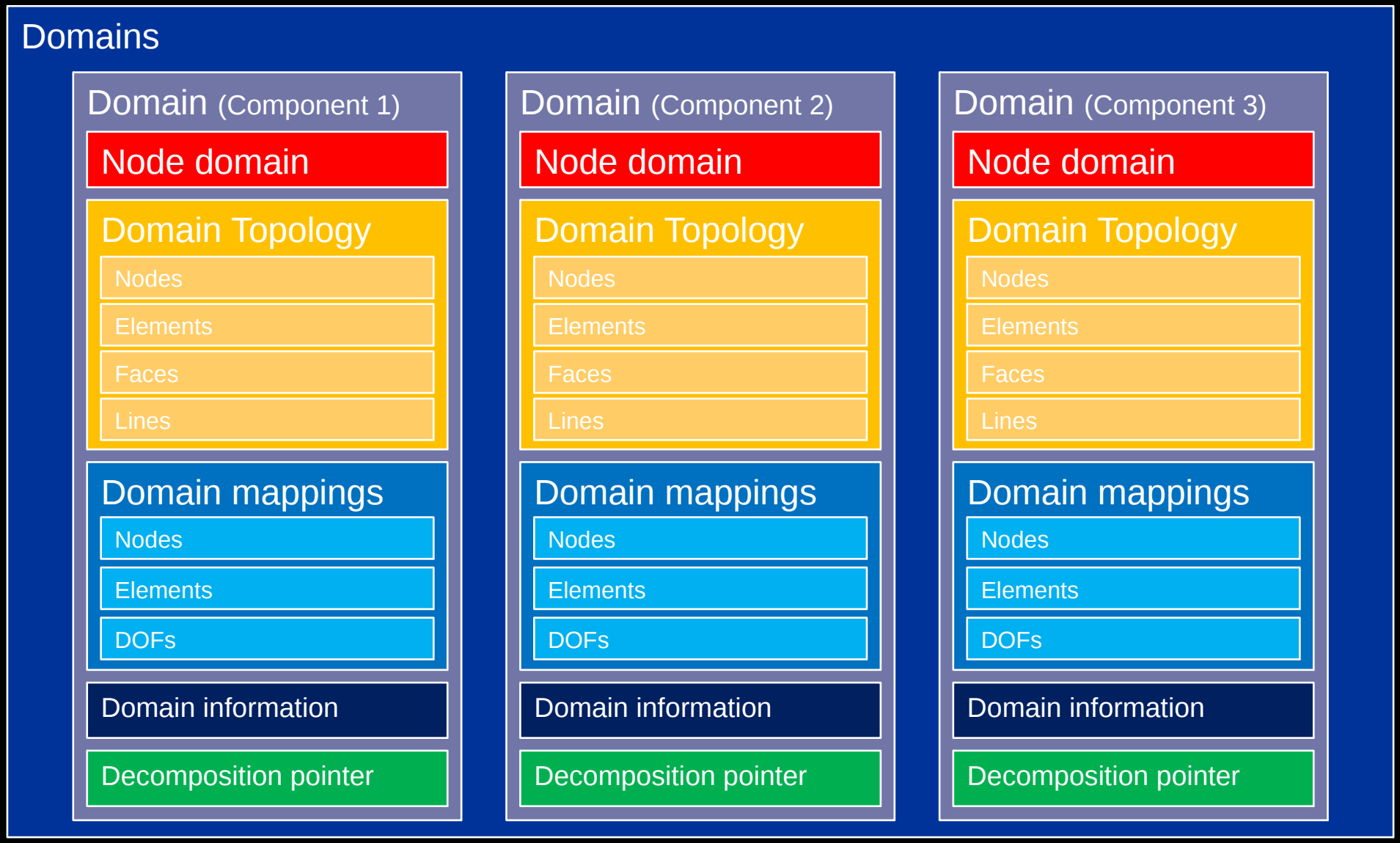
Mesh information

- Identifying Number
- Number of Components
- …

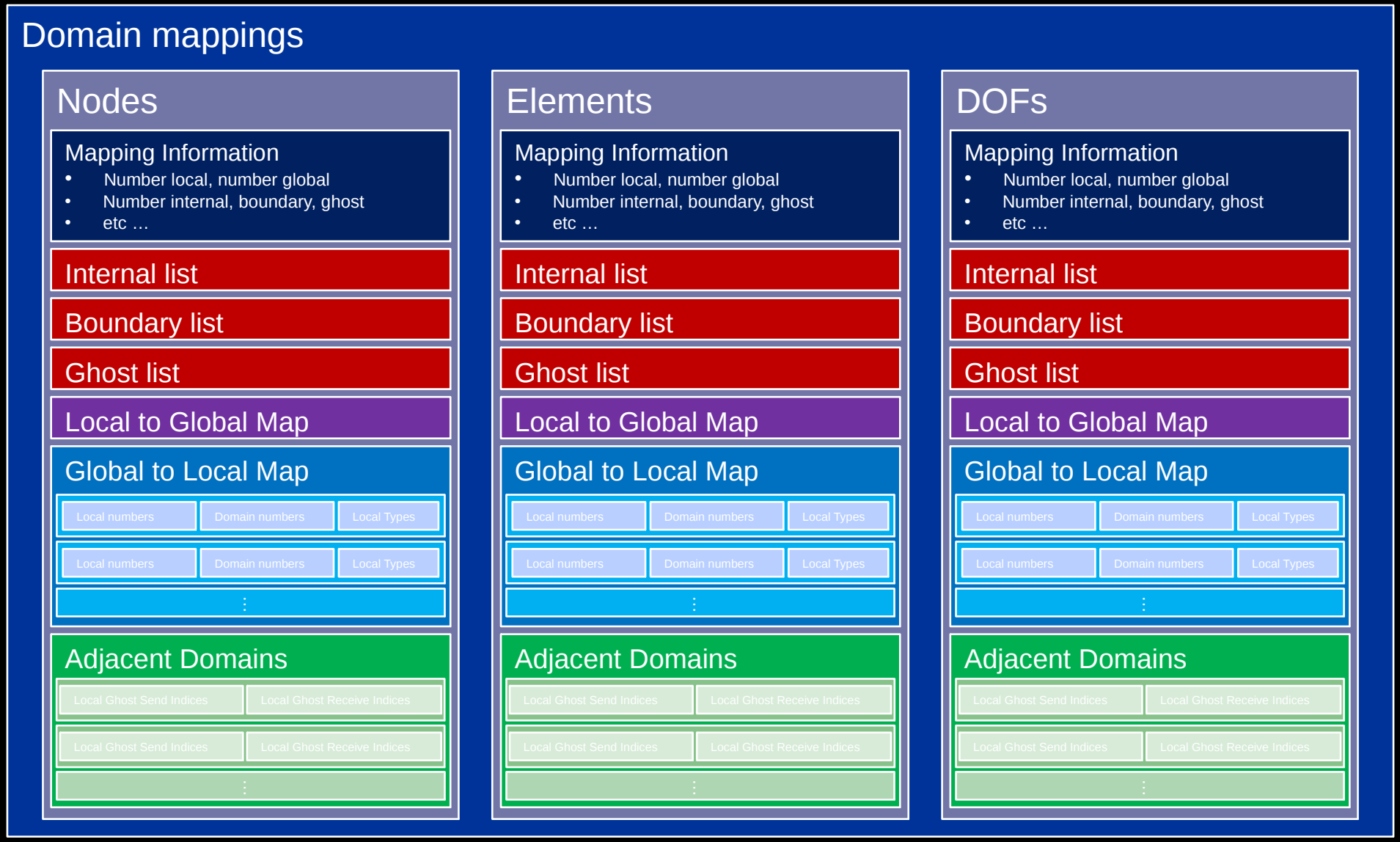
Decomposition sub-objects



Domains


Domains
Domain (Component 1)
Node domain
Domain Topology
Nodes
Elements
Faces
Lines
Domain mappings
Nodes
Elements
DOFs
Domain information
Decomposition pointer
Domain (Component 2)
Node domain
Domain Topology
Nodes
Elements
Faces
Lines
Domain mappings
Nodes
Elements
DOFs
Domain information
Decomposition pointer
Domain (Component 3)
Node domain
Domain Topology
Nodes
Elements
Faces
Lines
Domain mappings
Nodes
Elements
DOFs
Domain information
Decomposition pointer

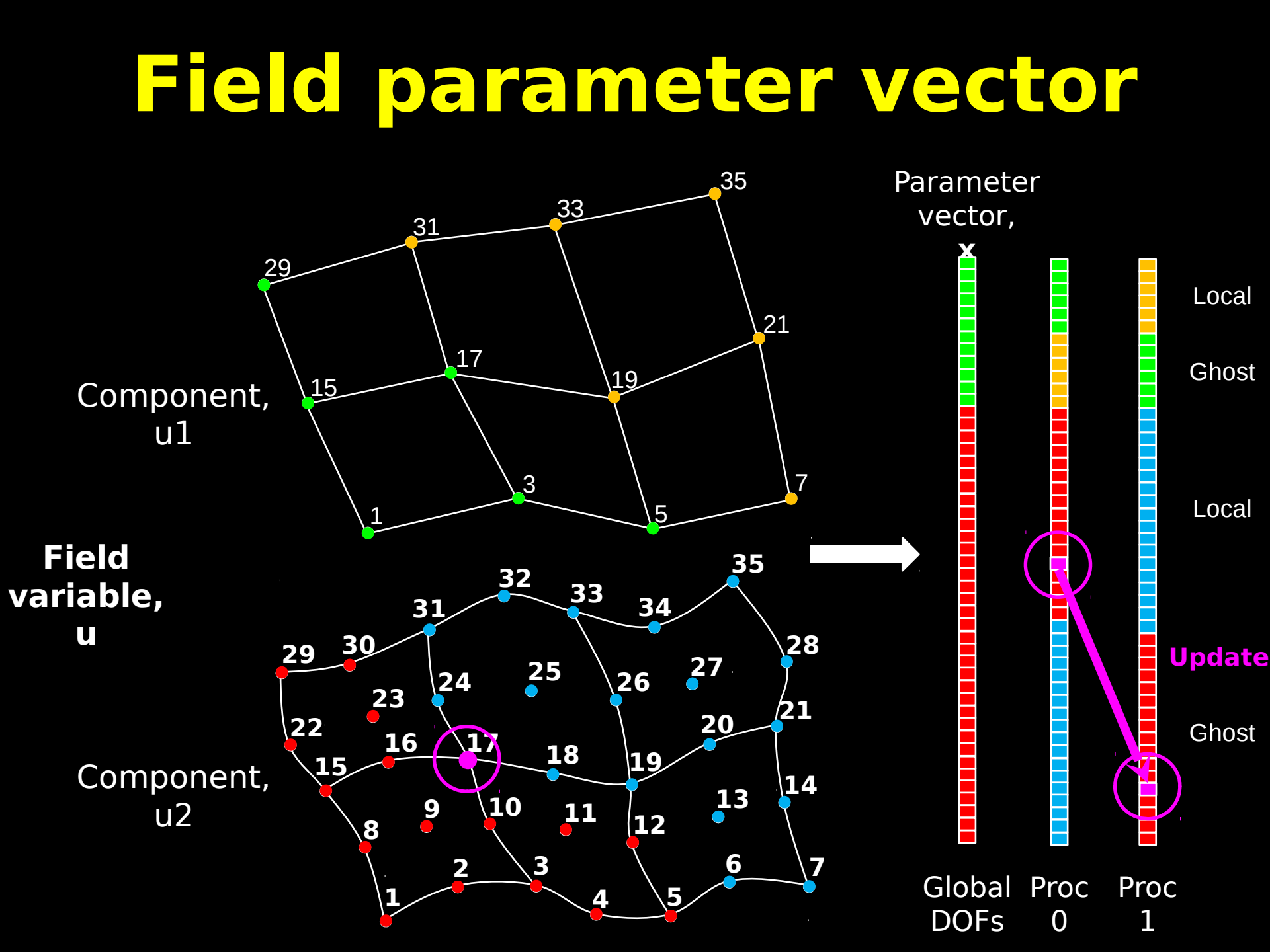
Domain mappings



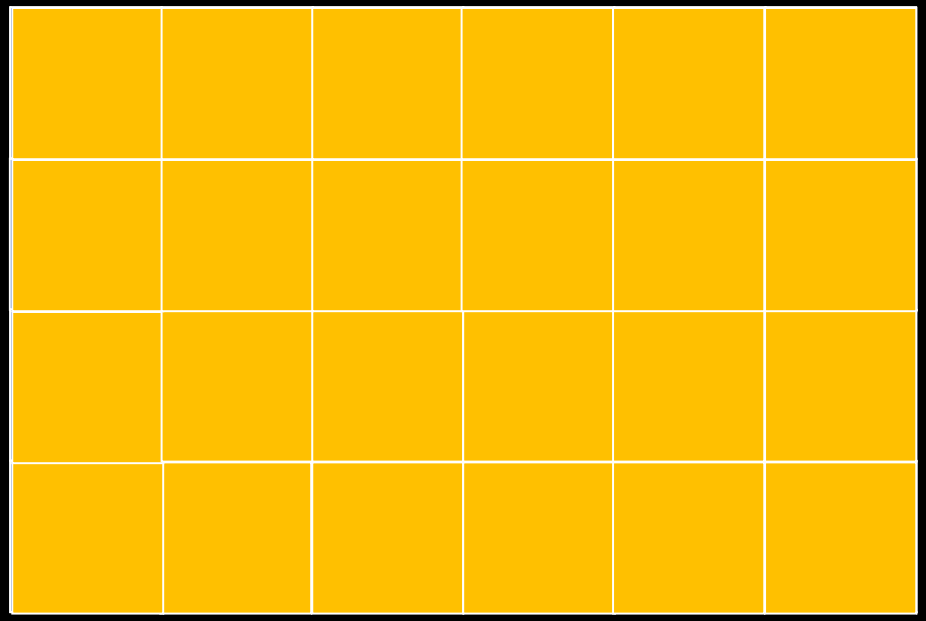
Fields

- Fields are the central object for storing information and framing the problem.
- Fields have a number of field variables i.e., u, ∂u/∂n, ∂u/∂t, ∂2u/∂t2, v, ∂v/∂n....
- Each field variable has a number of components.
- A field is defined on a decomposed mesh.
- Each field variable component is defined on a decomposed mesh component.

Field parameter vector



Generic temporal-spatial problem

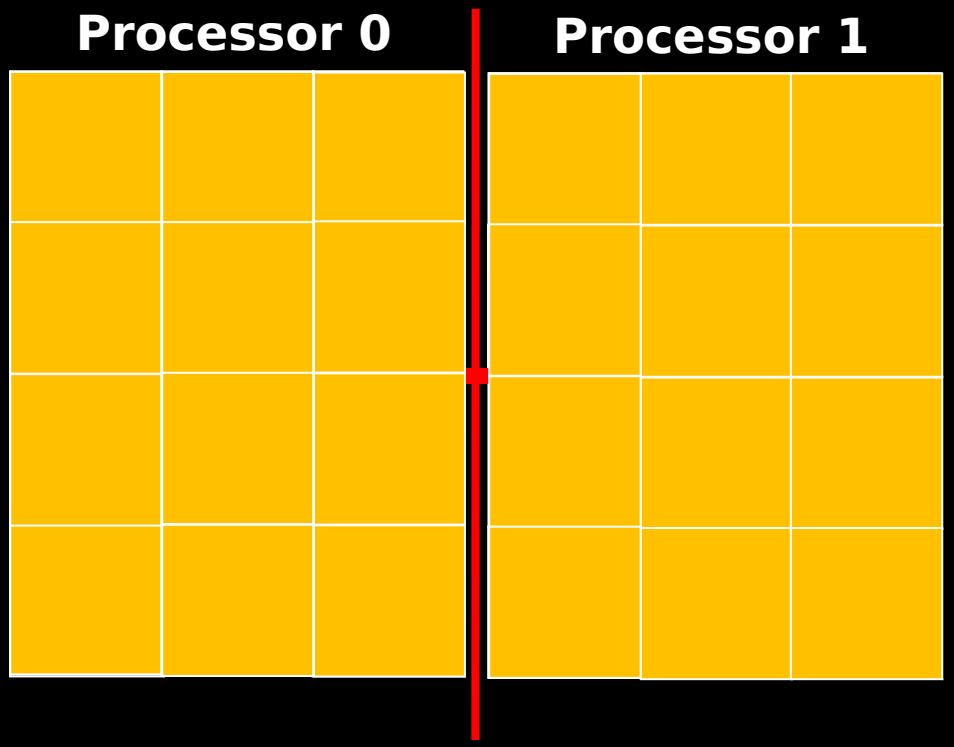


Loop over time
  Update current field parameters
    from previous (or initial) time step

  Loop over elements
    Calculate element equations
  Endloop

  Assemble global equation system
  Solve spatial system
  Calculate field parameters at next
    time step
Endloop

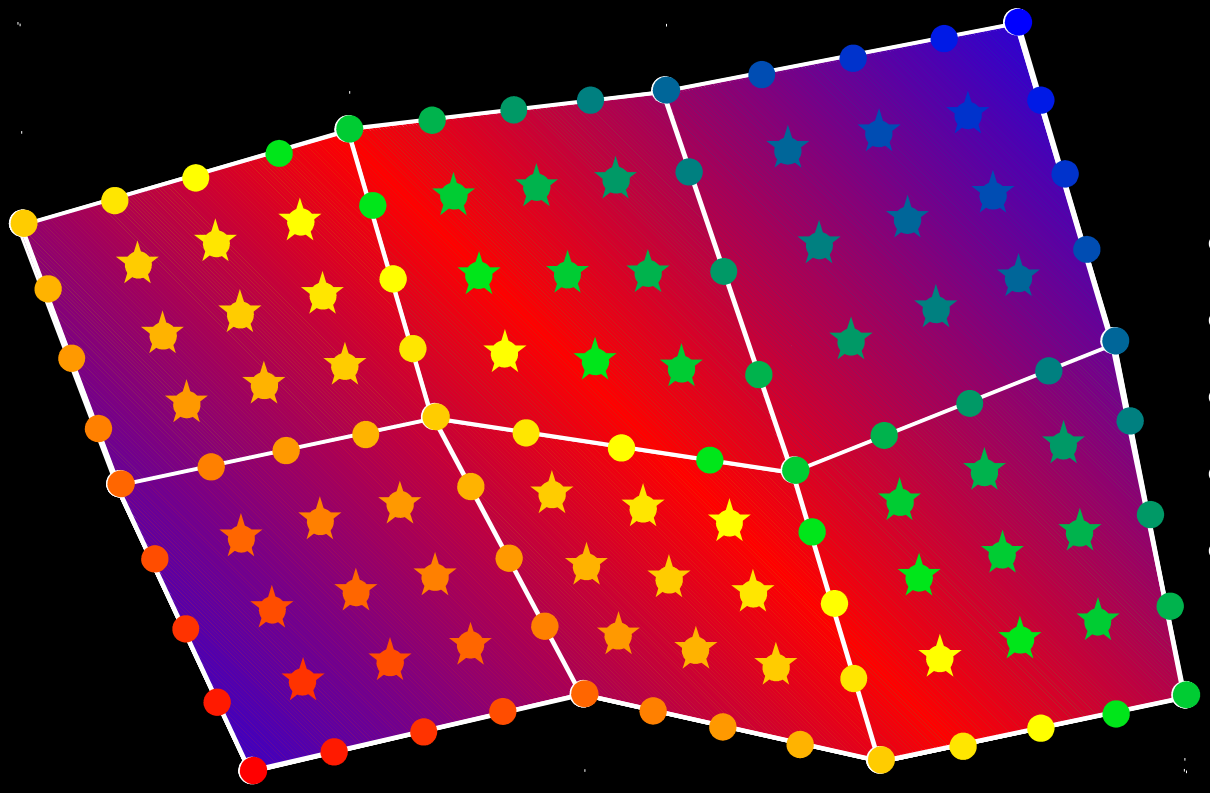
Generic temporal-spatial problem (distributed)



- Loop over time
- Update current field parameters
- from previous (or initial) time step
- Start ghost transfer of previous time step solution
- Loop over internal elements
- Calculate element equations
- Endloop
- Wait until transfer has finished
- Loop over boundary+ghost elements
- Calculate element equations
- Endloop
- Assemble global equation system
- Solve spatial system
- Calculate field parameters at next time step
- Endloop

Still allows for OpenMP style shared memory parallelism or serial execution

Field interpolation



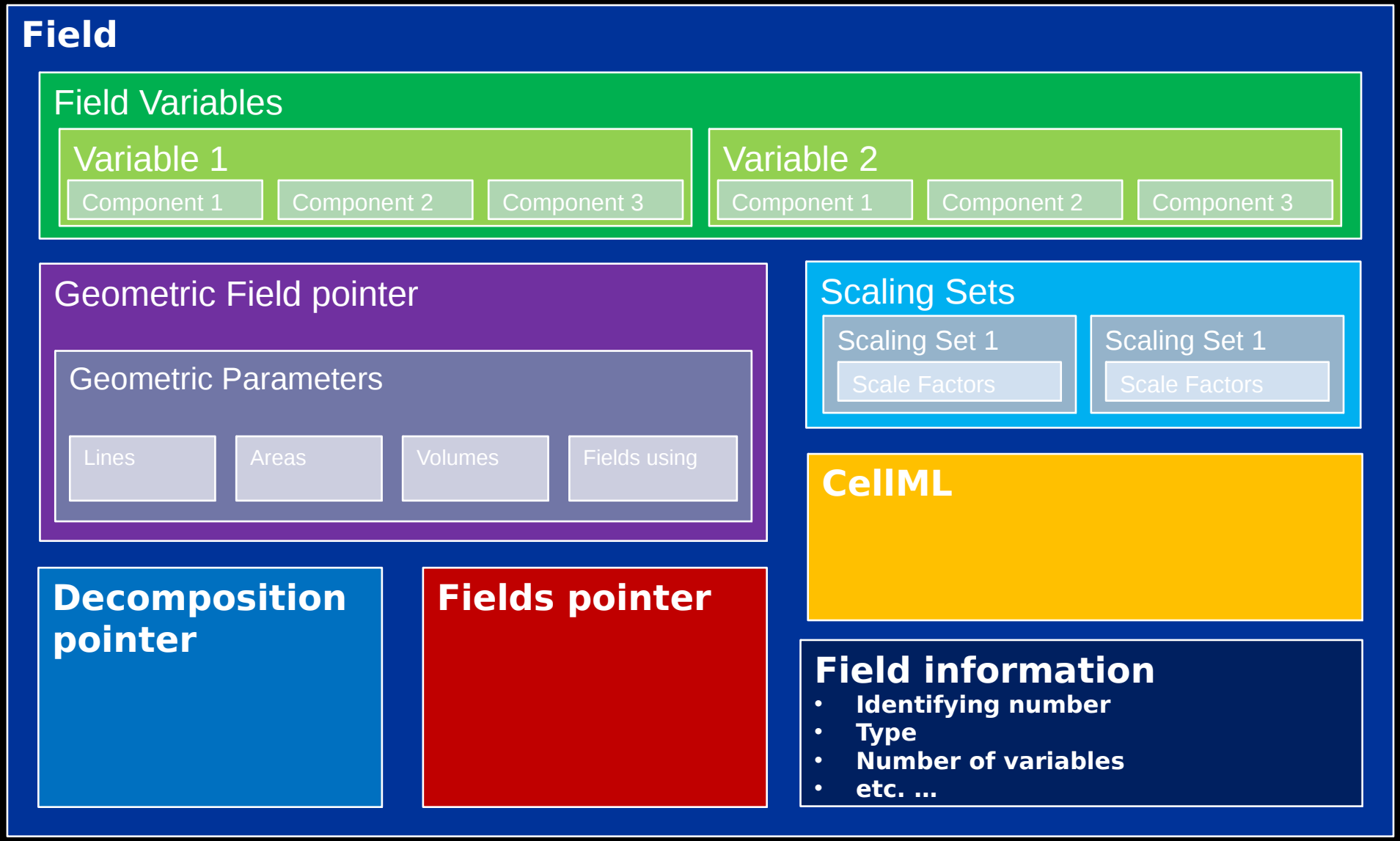
- Constant
- Element based
- Node based
- Grid point based
- Gauss point based

Types of fields

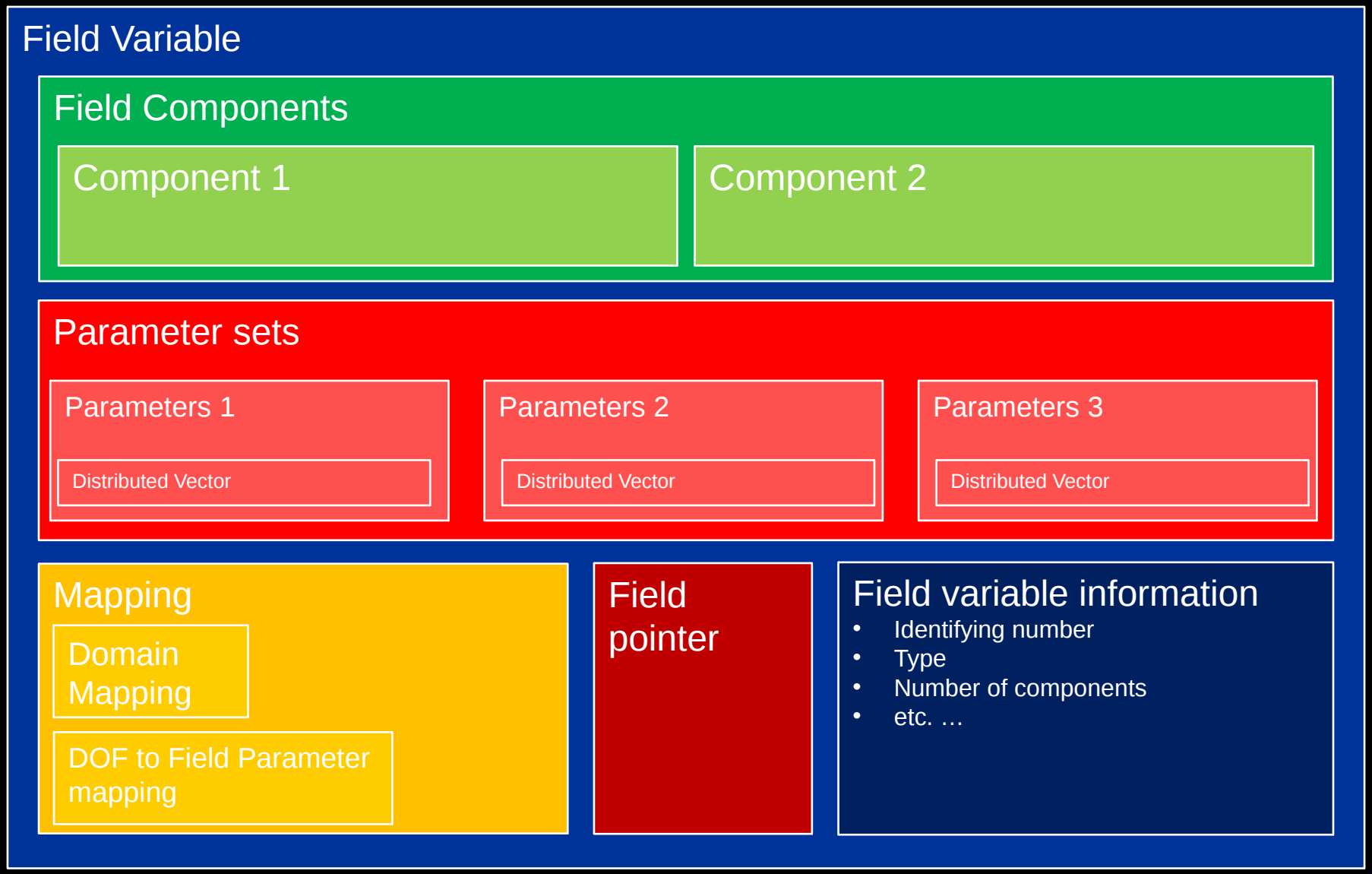
- Geometric
- Fibre
- General
- Material
- Source

- Independent
- Dependent

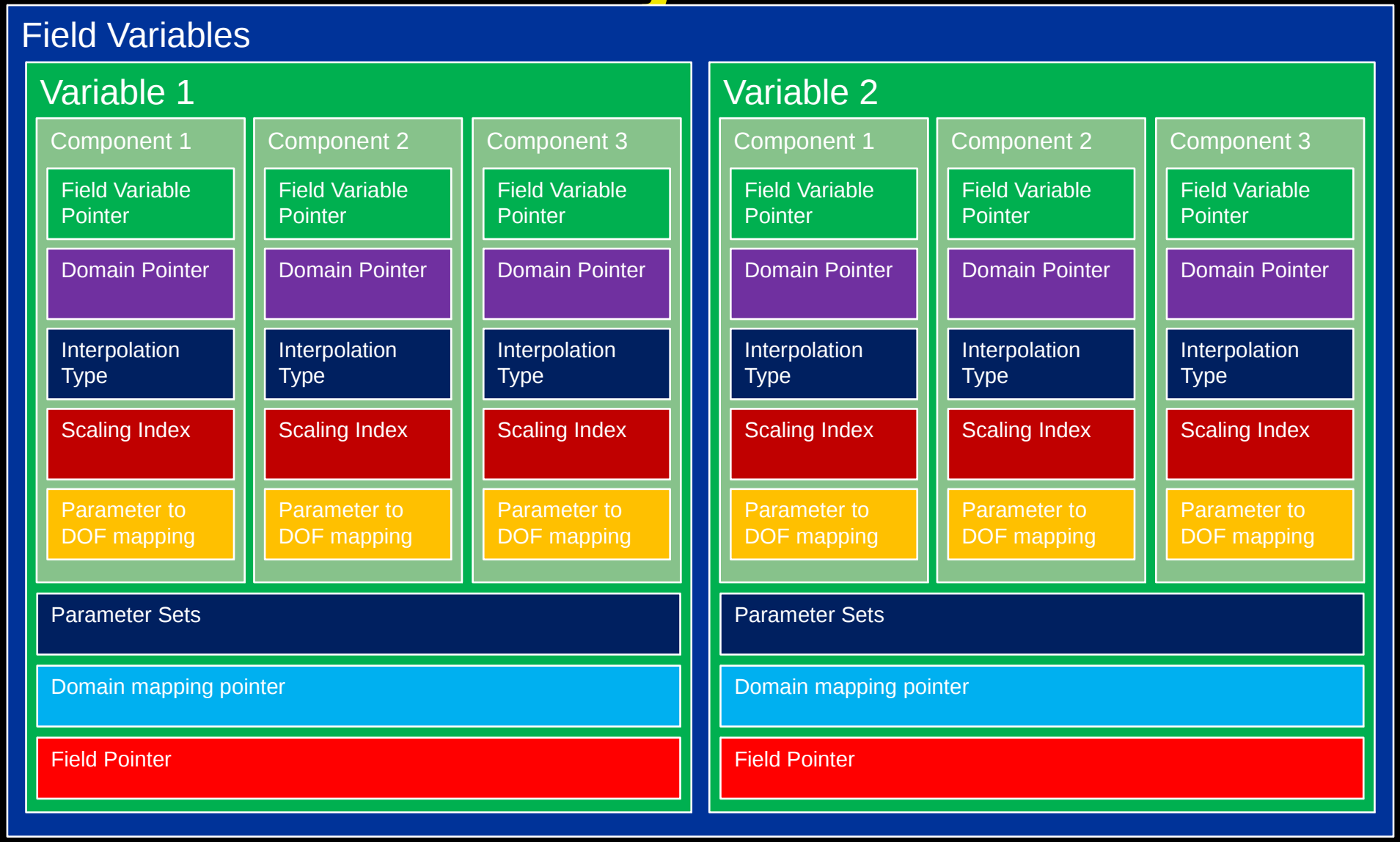
Field sub-objects



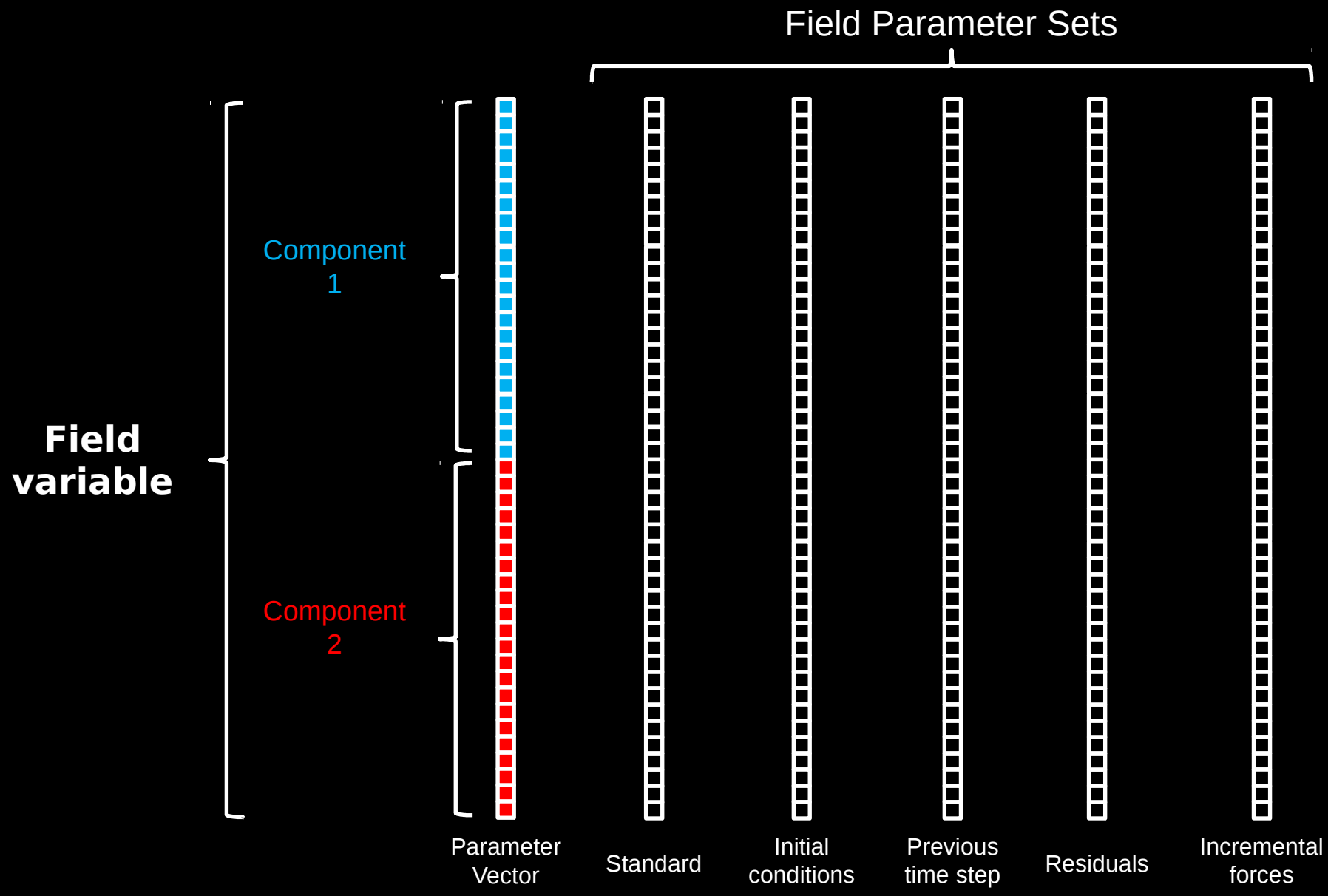
Field Variable sub-objects



Field Components sub-objects



Field variable parameter sets



Basic Structure of Code

CMISSInitialise (For Fortran and C)

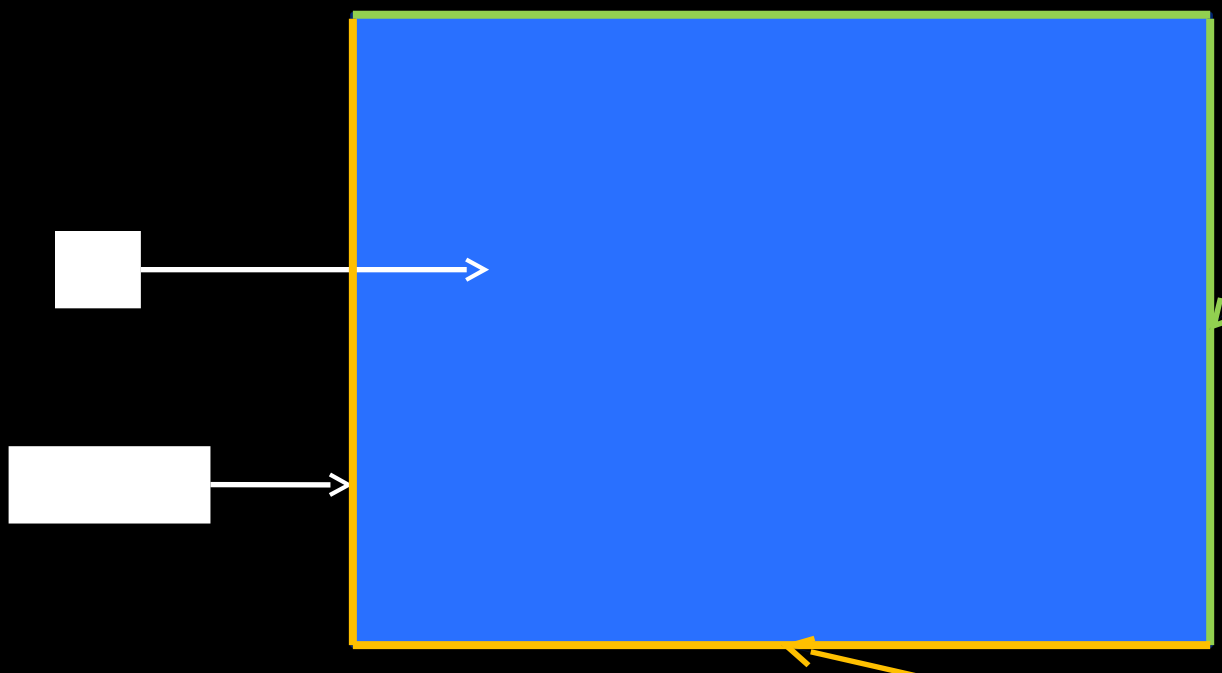
1. **(System setup)**
2. **CoordinateSystem Objects**
3. **Region Objects**
4. **Basis Objects**
5. **(GeneratedMesh Objects)**
6. **Mesh Objects**
7. **Decomposition Objects**
8. **Field Objects**
9. **…**

CMISSFinalise

Review Example

Laplace's equation

$$\phi(\boldsymbol{x}) = \phi_d(\boldsymbol{x}), \boldsymbol{x} \in \Gamma_d$$



$$q(\boldsymbol{x}) = \nabla\phi(\boldsymbol{x}) \cdot \boldsymbol{n} = q_n(\boldsymbol{x}), \boldsymbol{x} \in \Gamma_n$$

Generalised Laplace equation

Laplace’s equation

Laplace equation

Laplace’s equation

Laplace’s equation

Laplace’s equation

5-min break.

Laplace’s equation

Equations sets

- Aim to have multiple classes
  - e.g. Elasticity, Fluid mechanics, Electromagnetics, General field problems, Fitting, Optimisation.
- Different equations within each class
  - e.g. Bidomain, Navier-stokes etc.
- Different solution techniques
  - e.g. FEM, BEM, FD, GFEM.
- Associated with a region
- Built using the fields defined on the region.
- Different forms for linear/nonlinear and dynamic/static equations.

Equations Sets

Equations Sets

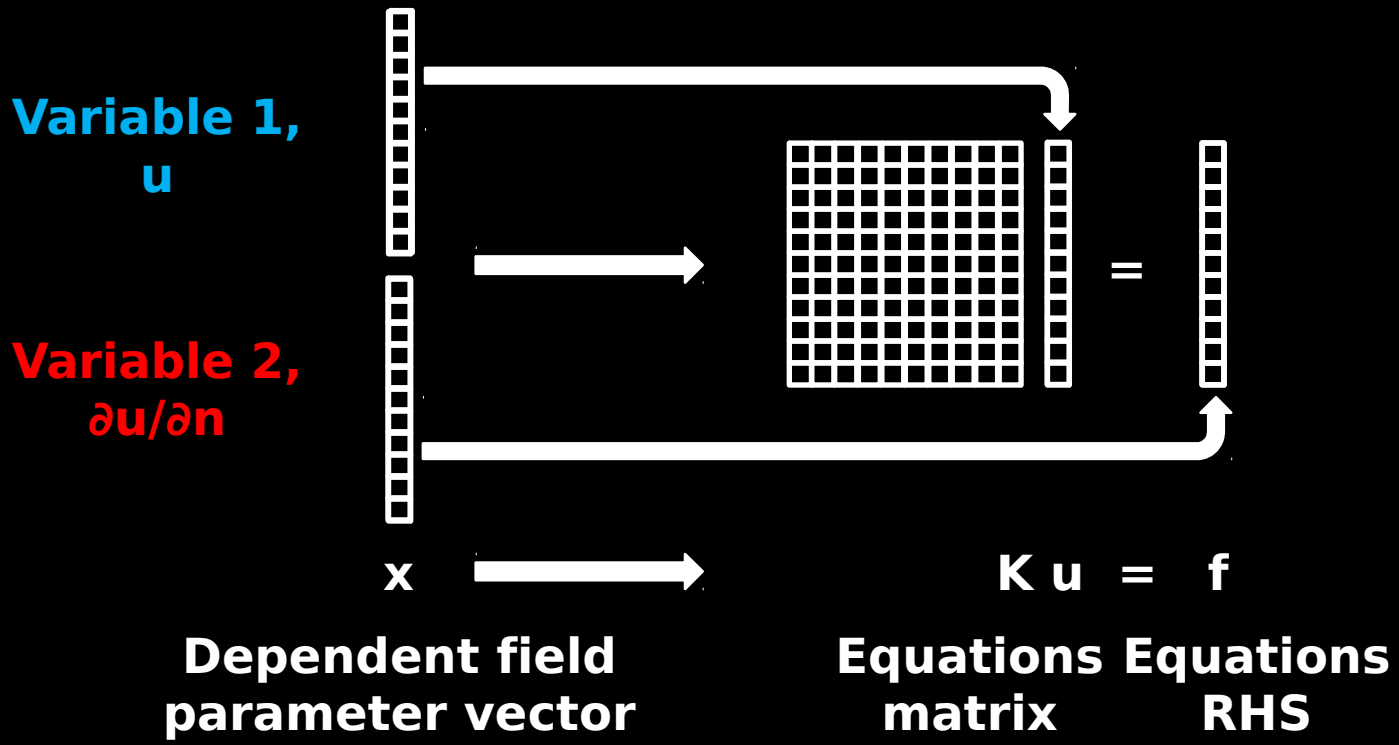
$$\boldsymbol{M\ddot{u} + C\dot{u} + Ku} + \sum_{i=1}^{N} \boldsymbol{A_i u_i} + \sum_{j=1}^{M} \boldsymbol{g_j}(u_k, u_l, \cdots) + \boldsymbol{s} = \boldsymbol{f}$$

Dynamic **Linear** **Nonlinear** **Source** **RHS**

Linear Equations

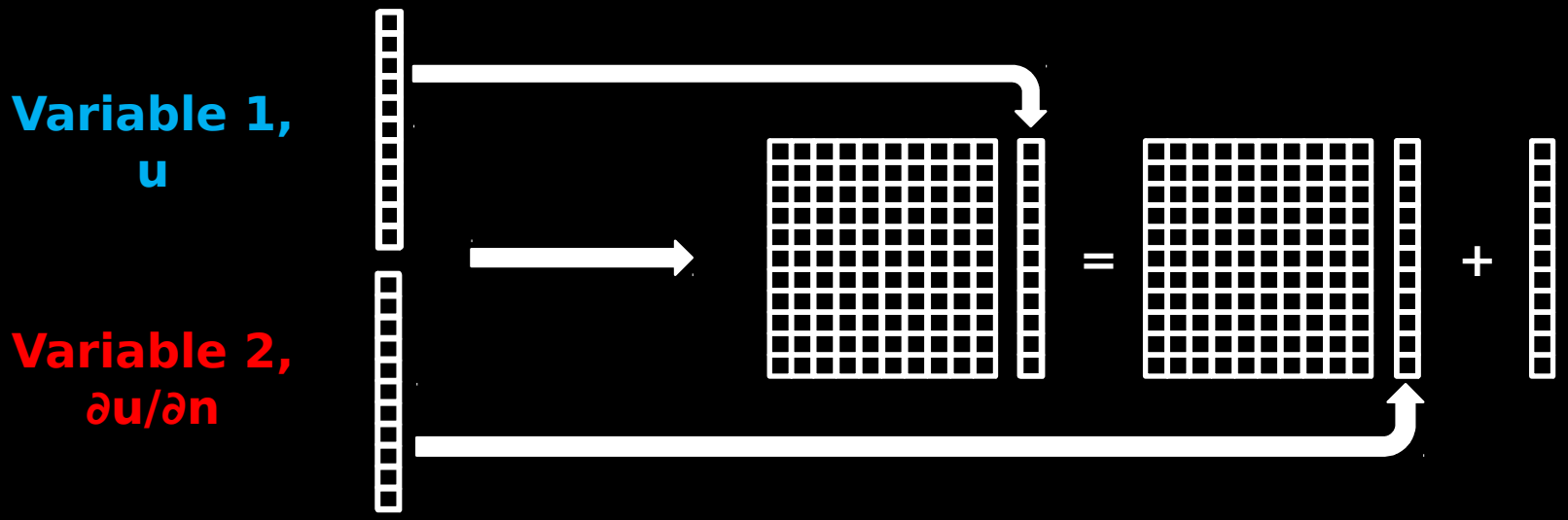
Field Variable → Matrix Mappings

- Map each field variable onto the equations matrices or RHS vector.
- e.g. Laplace (FEM):- 2 variables, 1 component



Field Variable → Matrix Mappings

- e.g. Laplace (BEM):- 2 variables, 1 component



x → $A\,u = B\,q + 0$

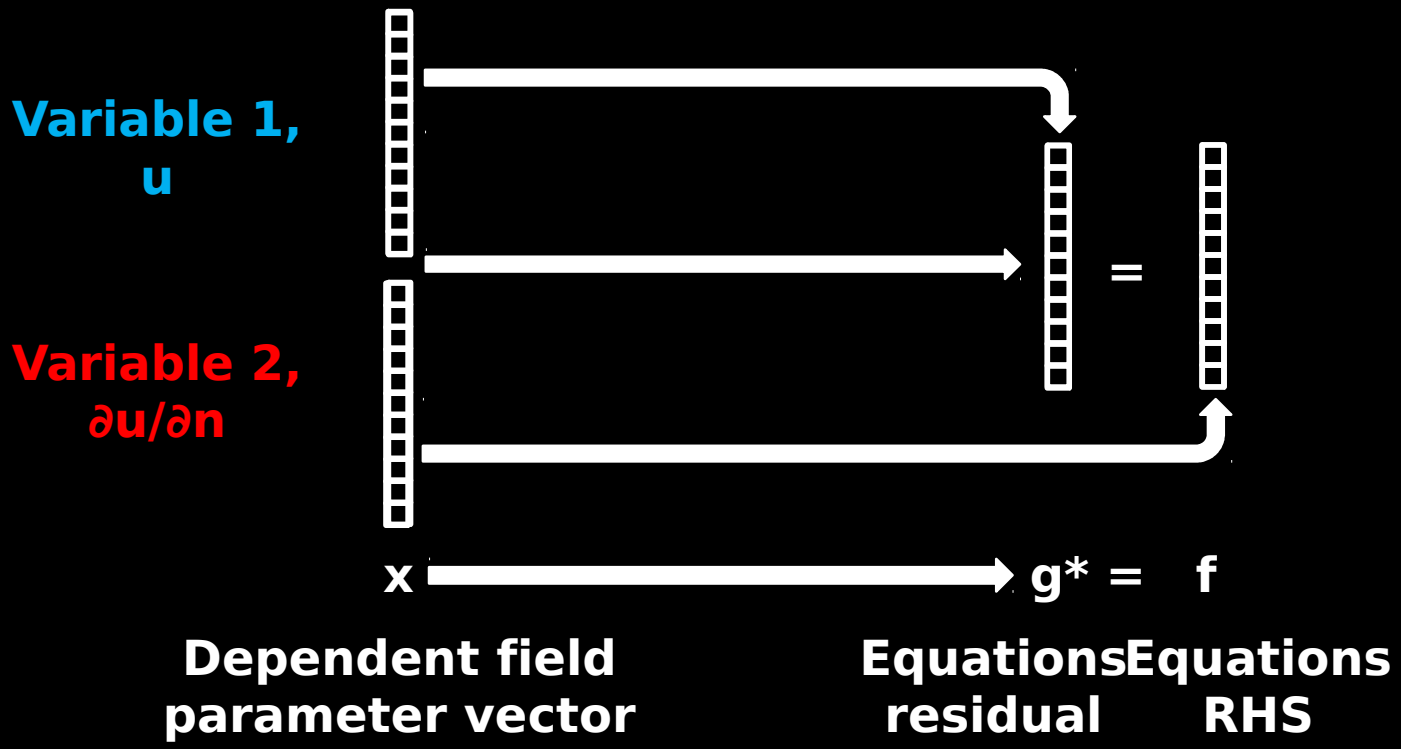
$A\,u - B\,q = 0$

Nonlinear Equations

Nonlinear Equations

Field Variable → Matrix Mappings

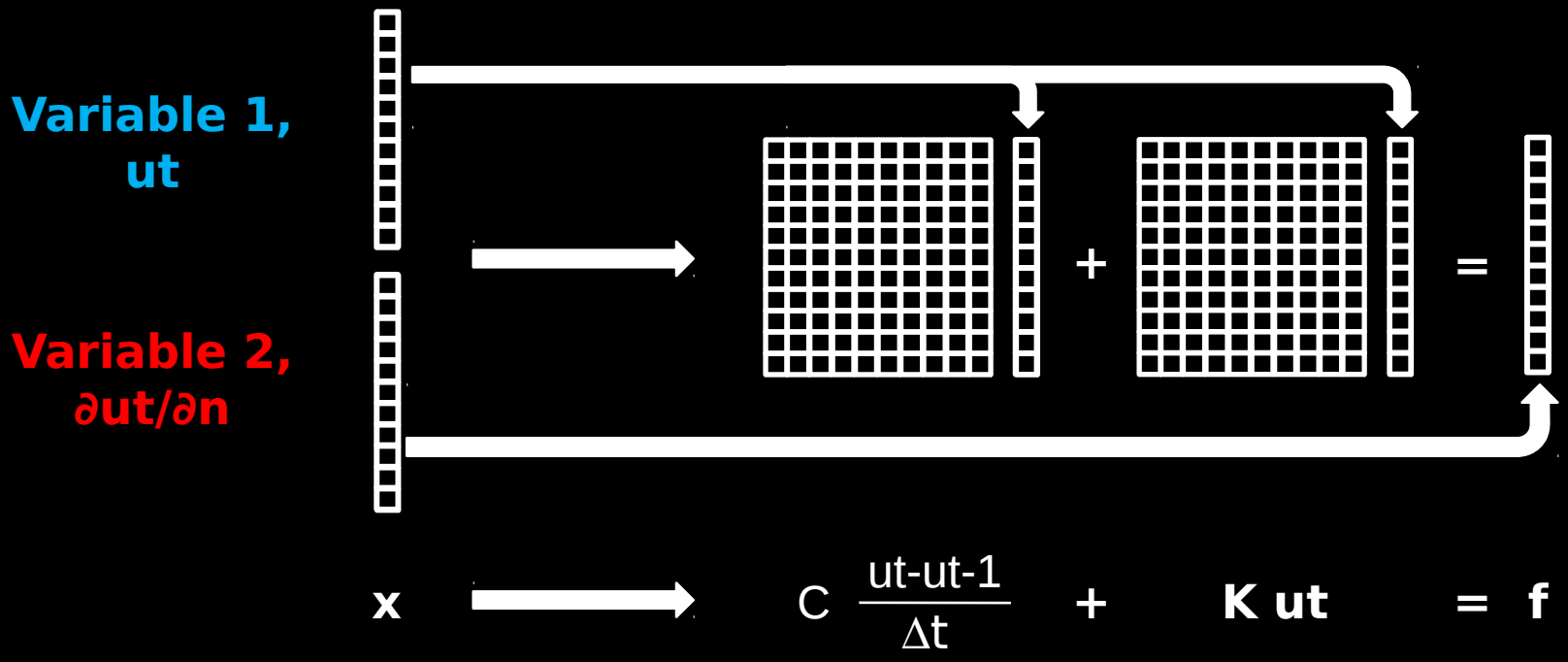
- Map each field variable onto the equations matrices, equations residual vector or RHS vector.
- e.g. Finite elasticity (FEM):- 2 variables, n+1 components, no linear parts.



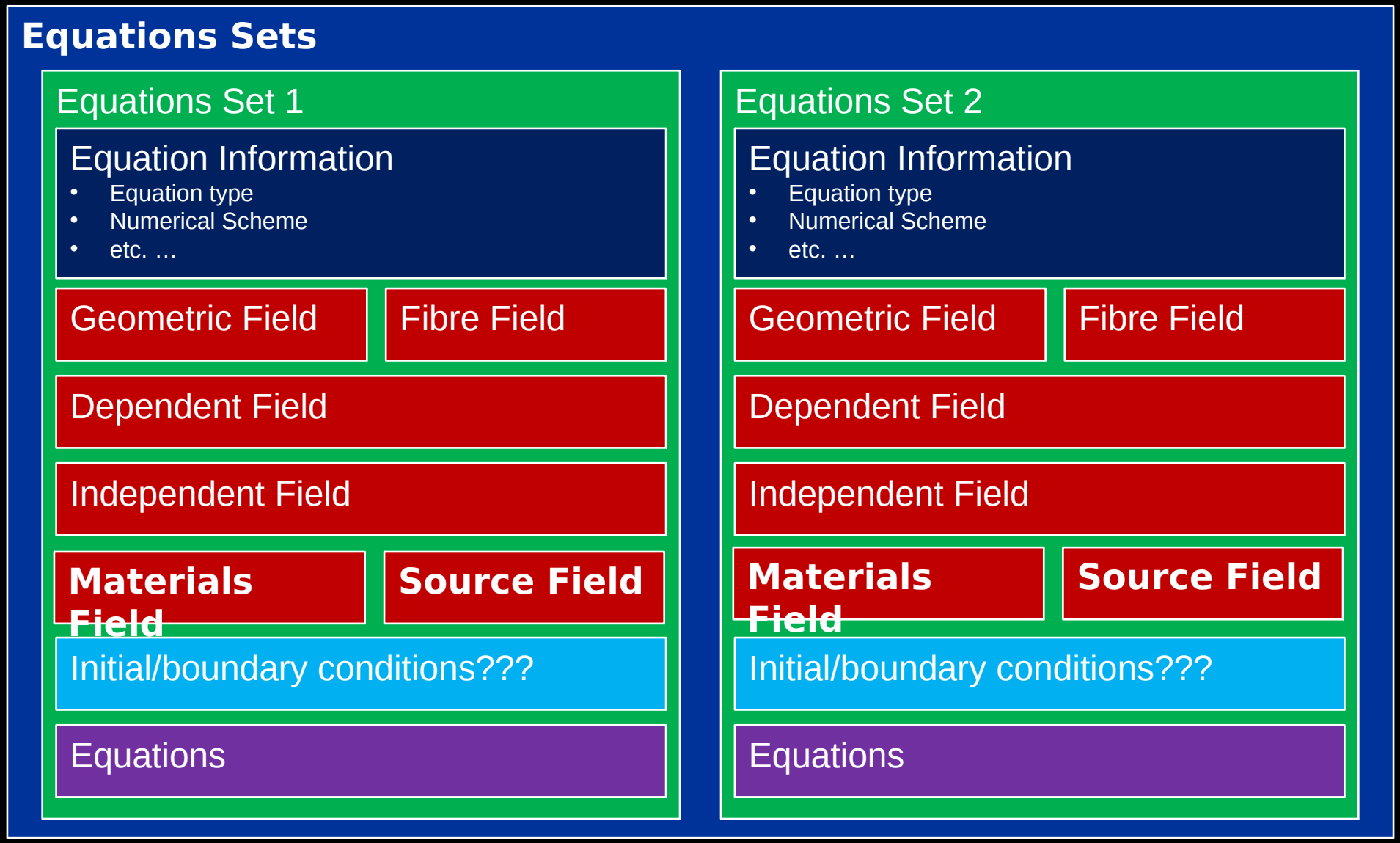
Dynamic Equations

Field Variable → Matrix Mappings

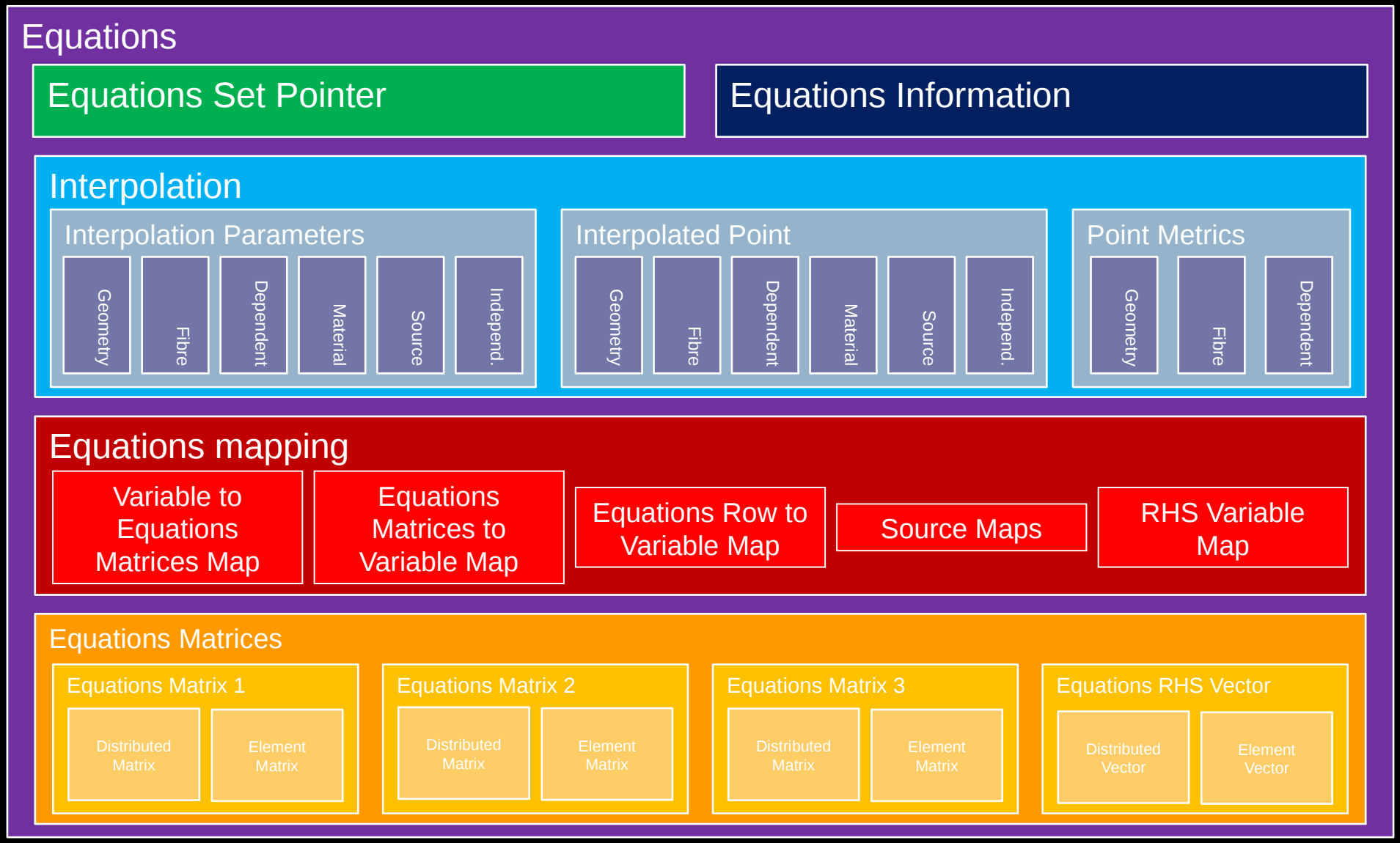
- e.g. Heat equation (explicit time/FEM space) :- 2 variables, 1 component



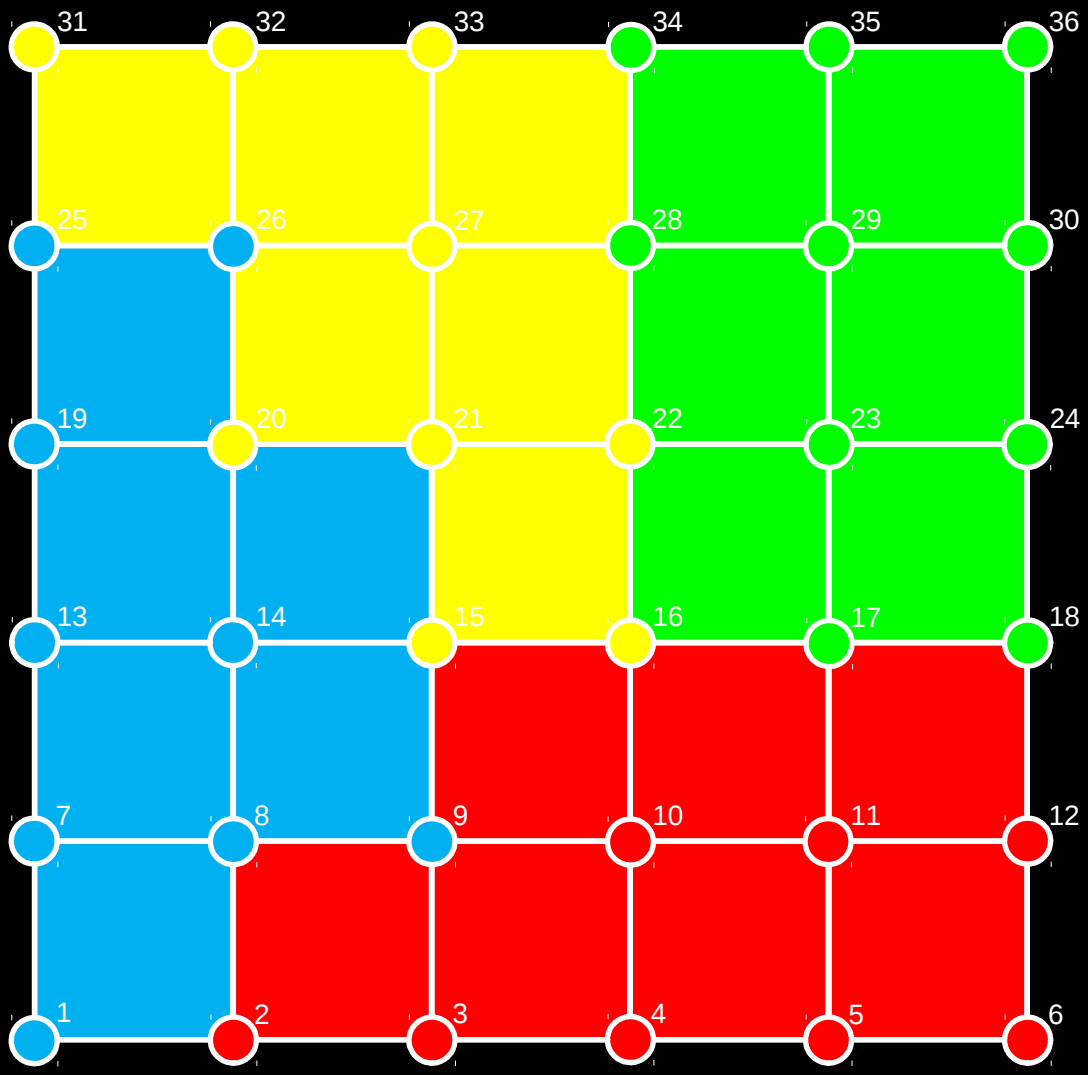
Equations Sets



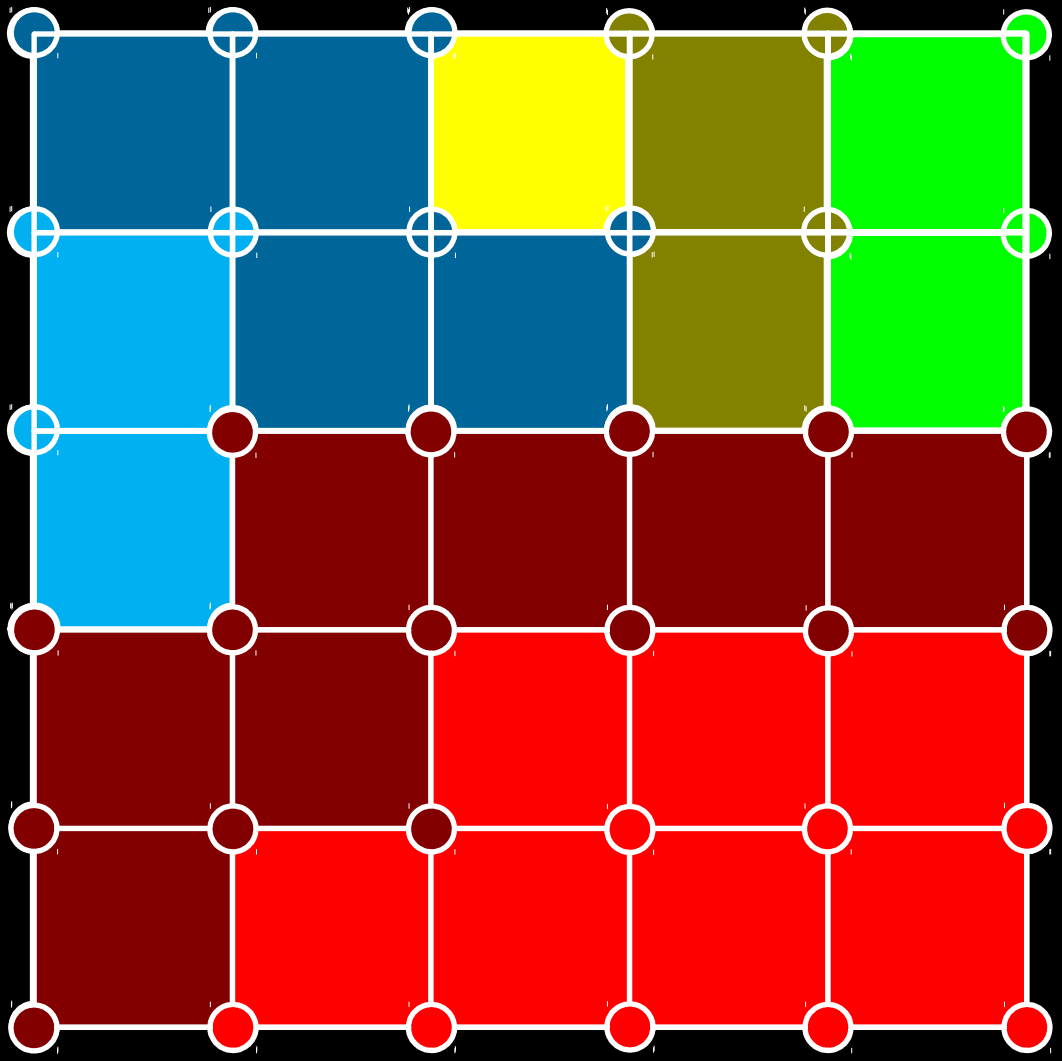
Equations


Equations
Equations Set Pointer
Equations Information
Interpolation
Interpolation Parameters
Geometry
Fibre
Dependent
Material
Source
Independ.
Interpolated Point
Geometry
Fibre
Dependent
Material
Source
Independ.
Point Metrics
Geometry
Fibre
Dependent
Equations mapping
Variable to Equations Matrices Map
Equations Matrices to Variable Map
Equations Row to Variable Map
Source Maps
RHS Variable Map
Equations Matrices
Equations Matrix 1
Distributed Matrix
Element Matrix
Equations Matrix 2
Distributed Matrix
Element Matrix
Equations Matrix 3
Distributed Matrix
Element Matrix
Equations RHS Vector
Distributed Vector
Element Vector

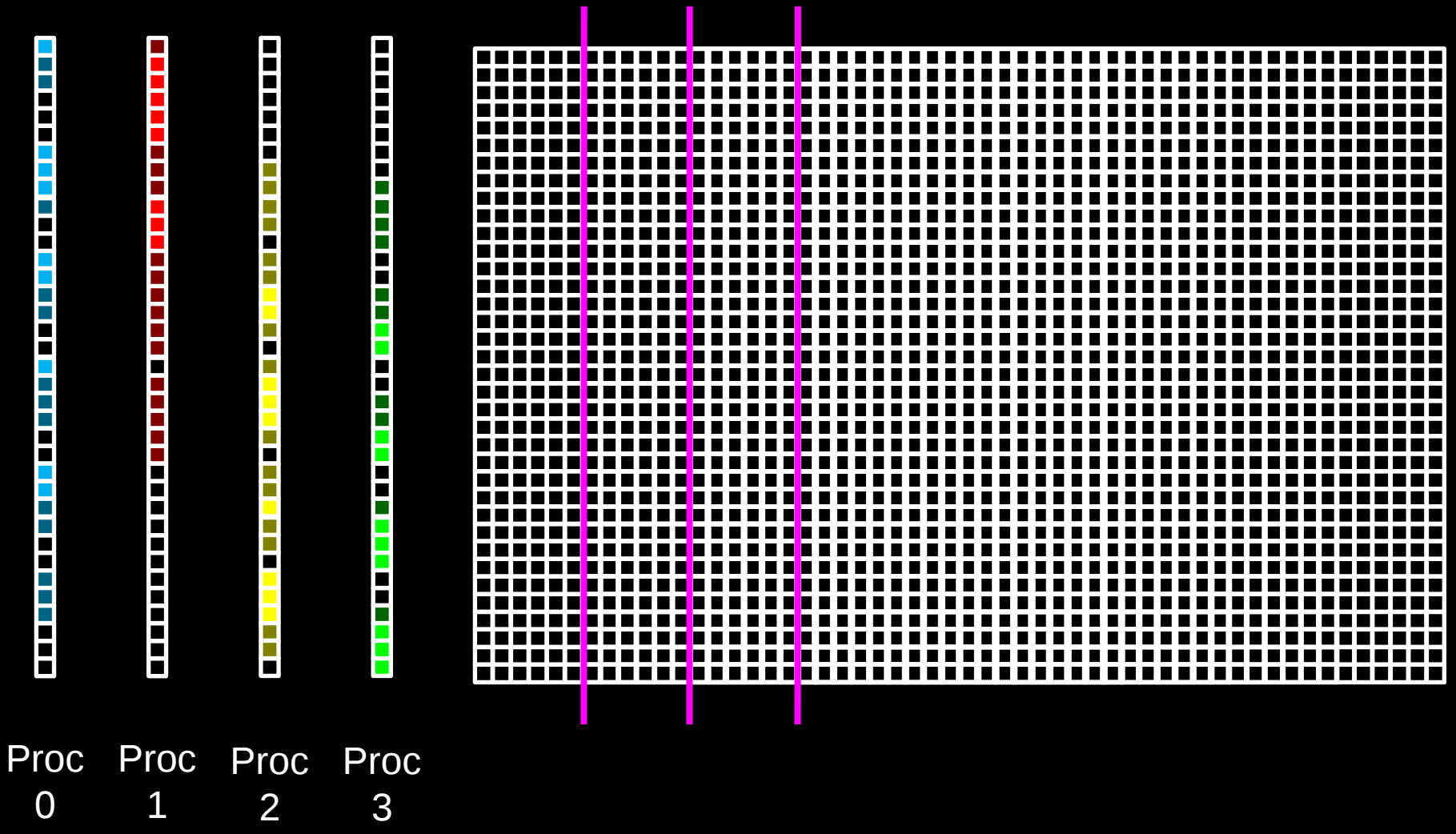
Matrix Distribution



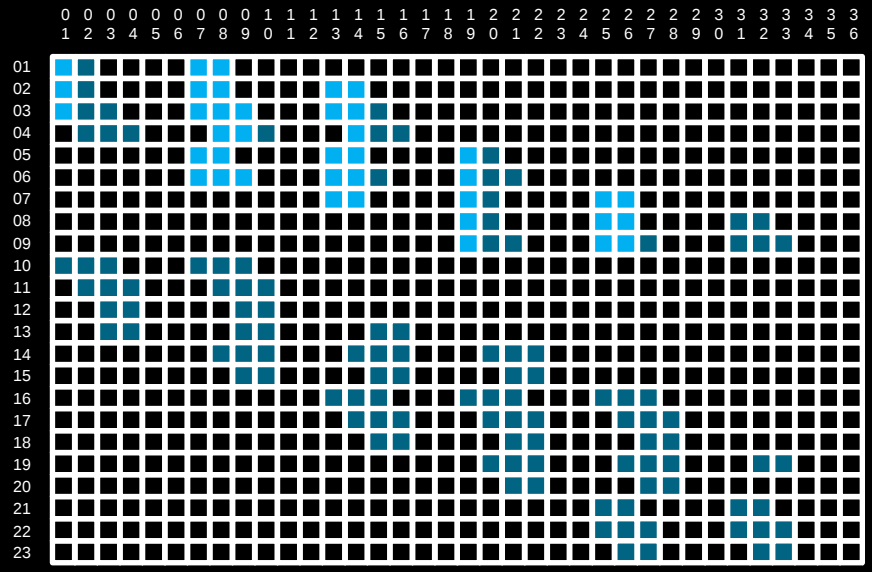
Matrix Distribution

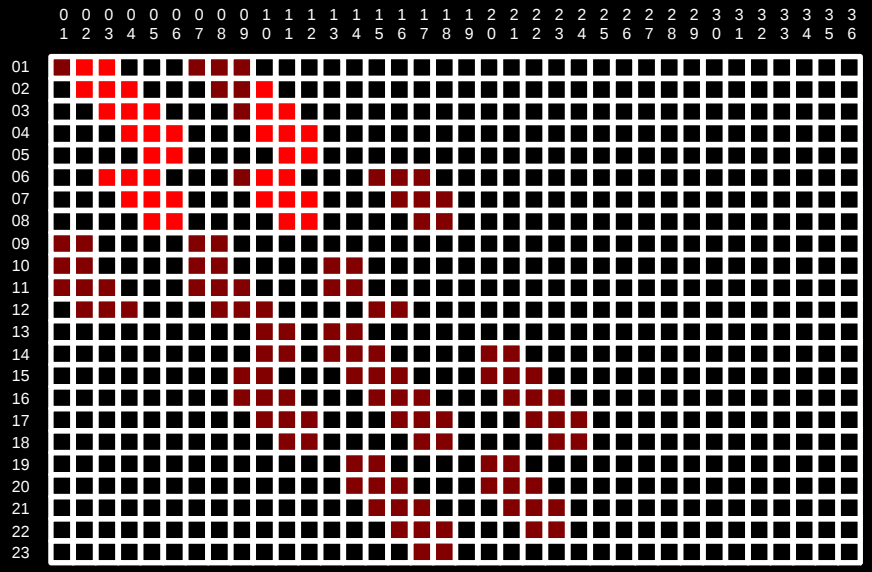


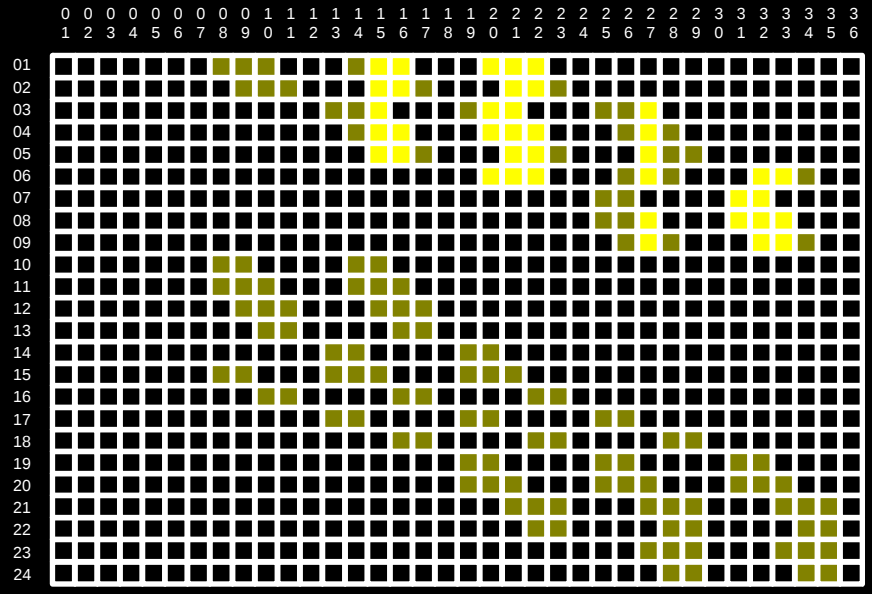
Matrix distribution

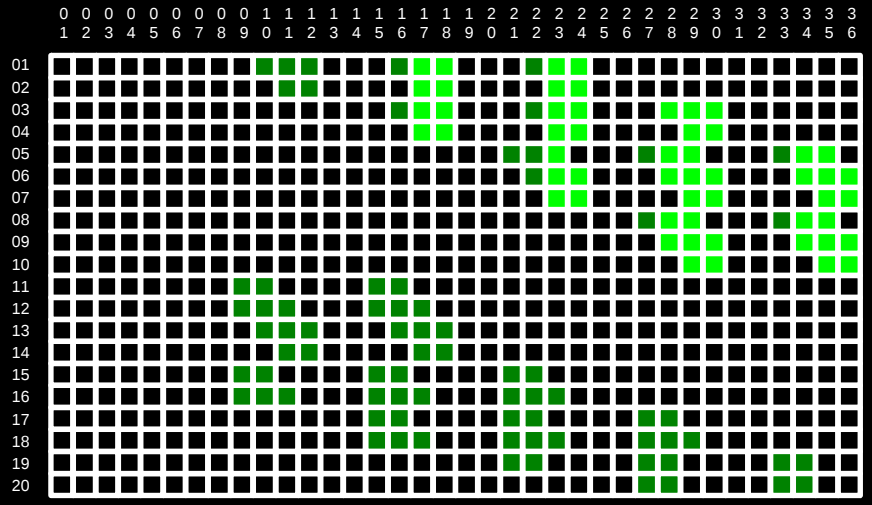


Matrix distribution

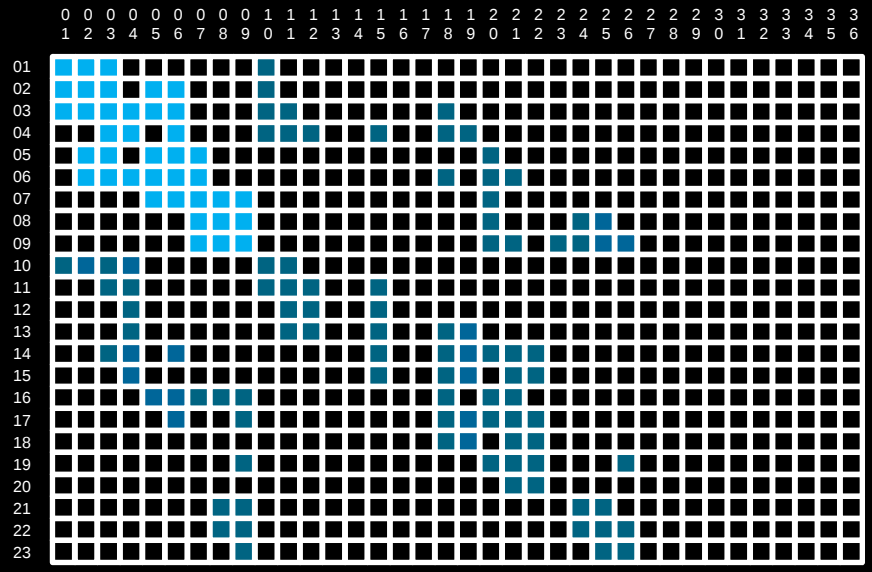


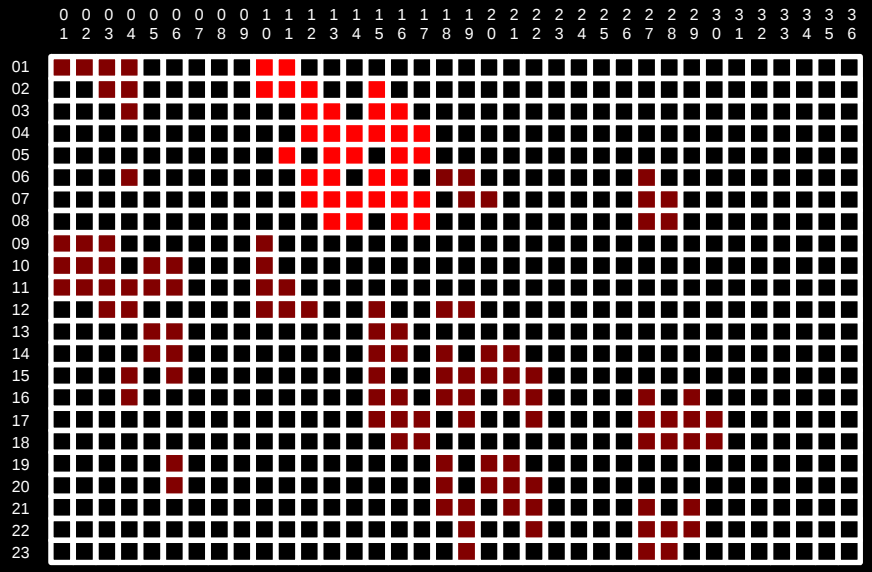


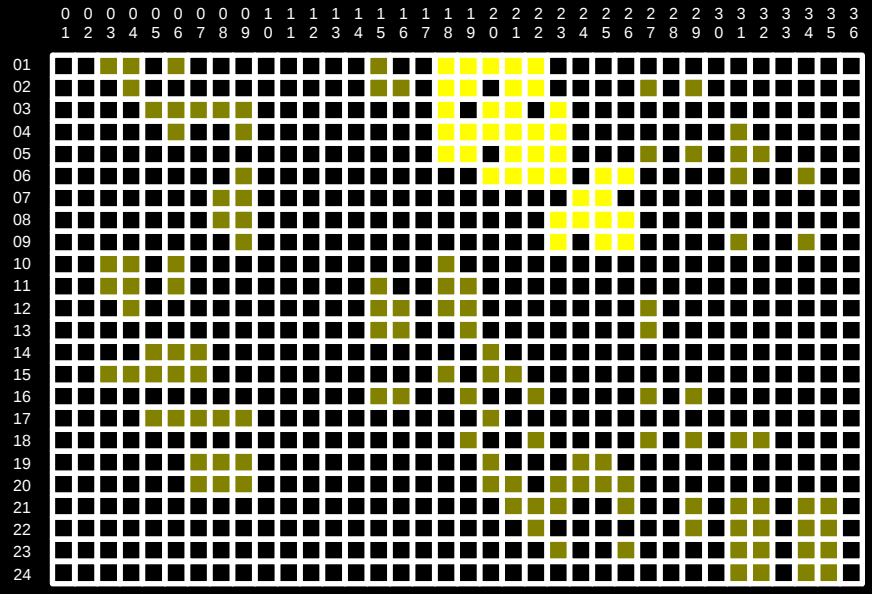


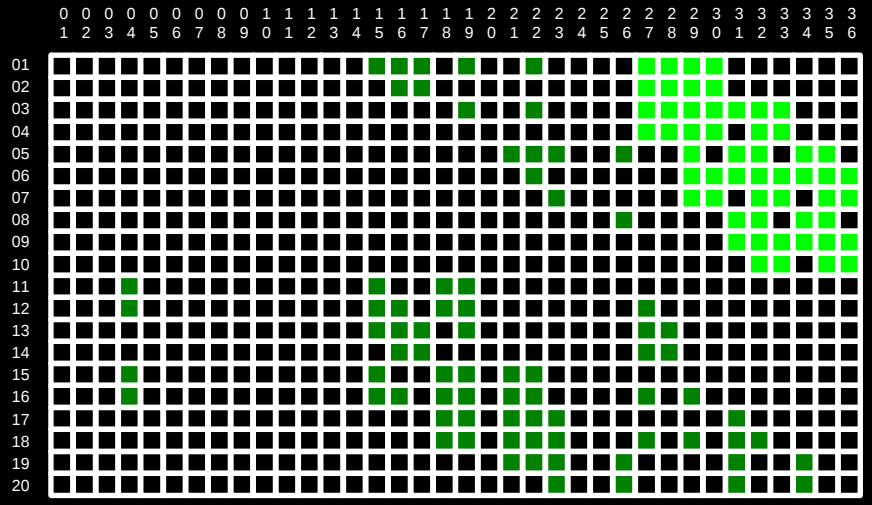


Matrix distribution

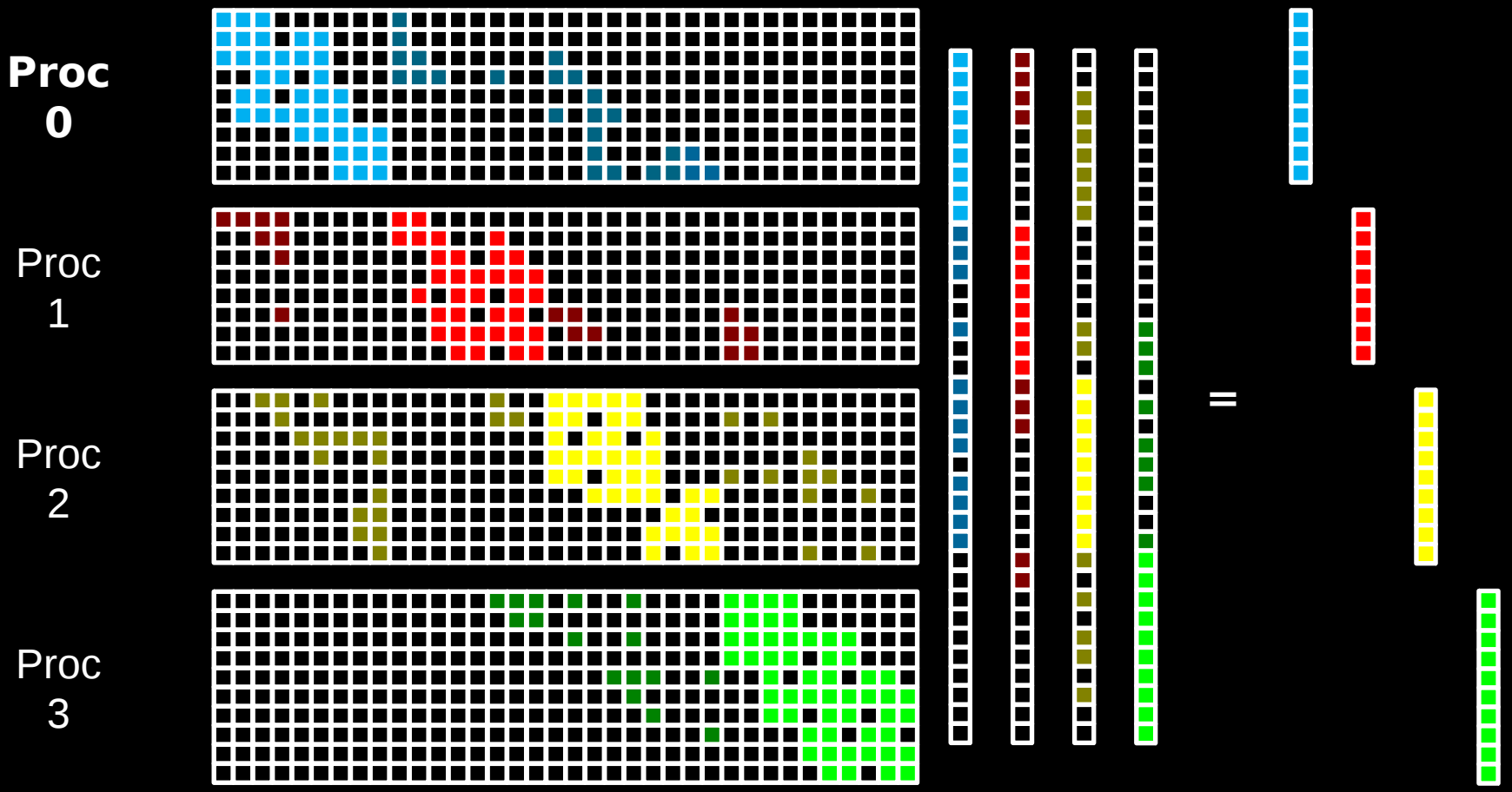








Matrix distribution



Basic Structure of Code

CMISSInitialise (For Fortran and C)

7. …
8. Decomposition Objects
9. Field Objects
10. Equations Sets
    1. Dependent Field
    2. (Materials Field, Source Field, etc.)
    3. Equations
11. ….

CMISSFinalise

Review Example

Multiphysics

- In order to accurately model biological systems OpenCMISS must model multiple physical systems at different time and space scales.
- OpenCMISS deals with multiphysics via two mechanisms:
  - A flexible system to allow for multiple solvers and workflows.
  - Methods for coupling different physical systems together.

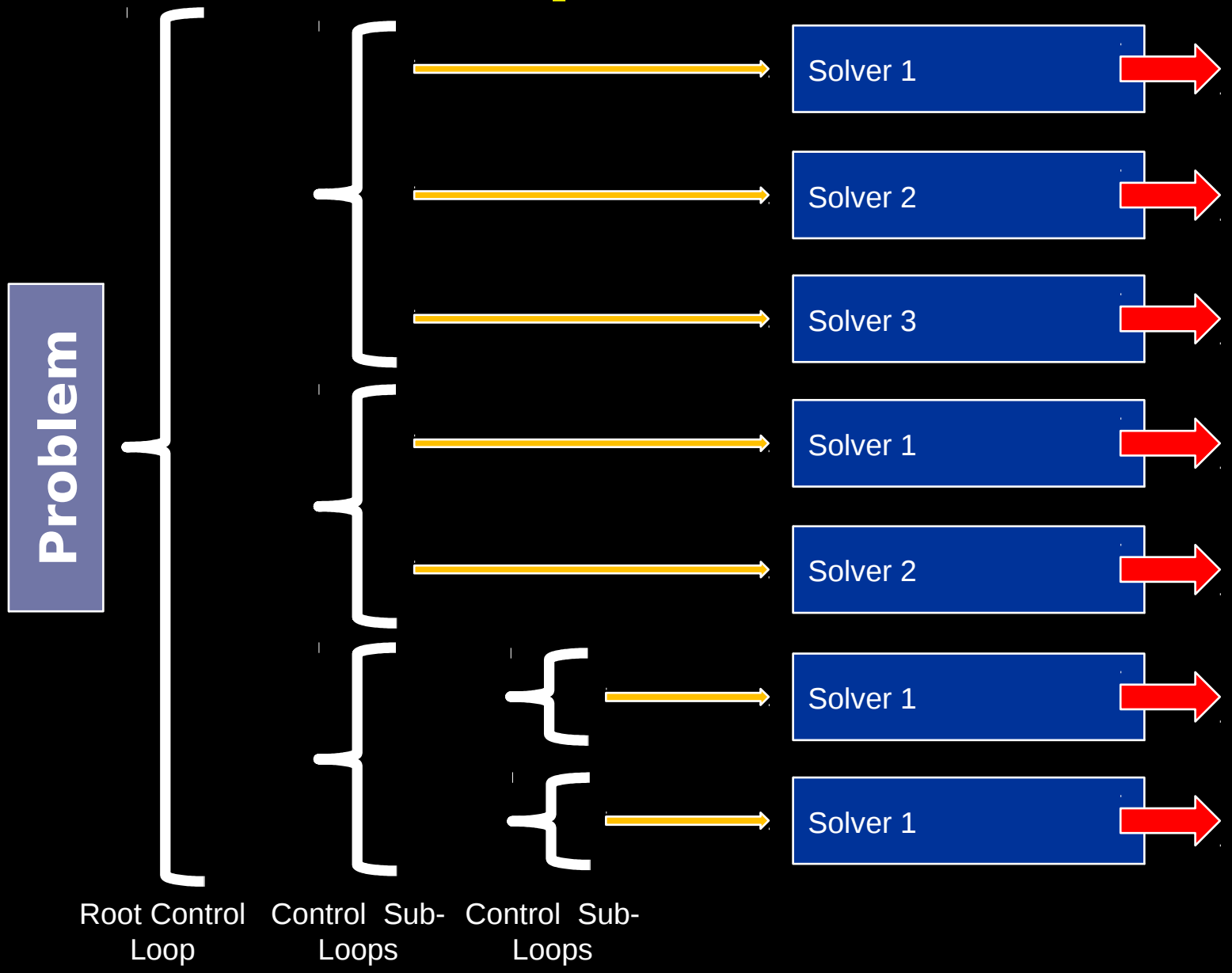
Multiphysics: Problem workflows

- Aim to have multiple classes
  - e.g. Elasticity, Fluid mechanics, Electromagnetics, General field problems, Fitting, Optimisation.
- Different problems within each class
  - e.g. Bidomain, Navier-stokes etc.
- To deal with problem complexity a problem has a number of nested control loops and each control loop has a number of solvers.

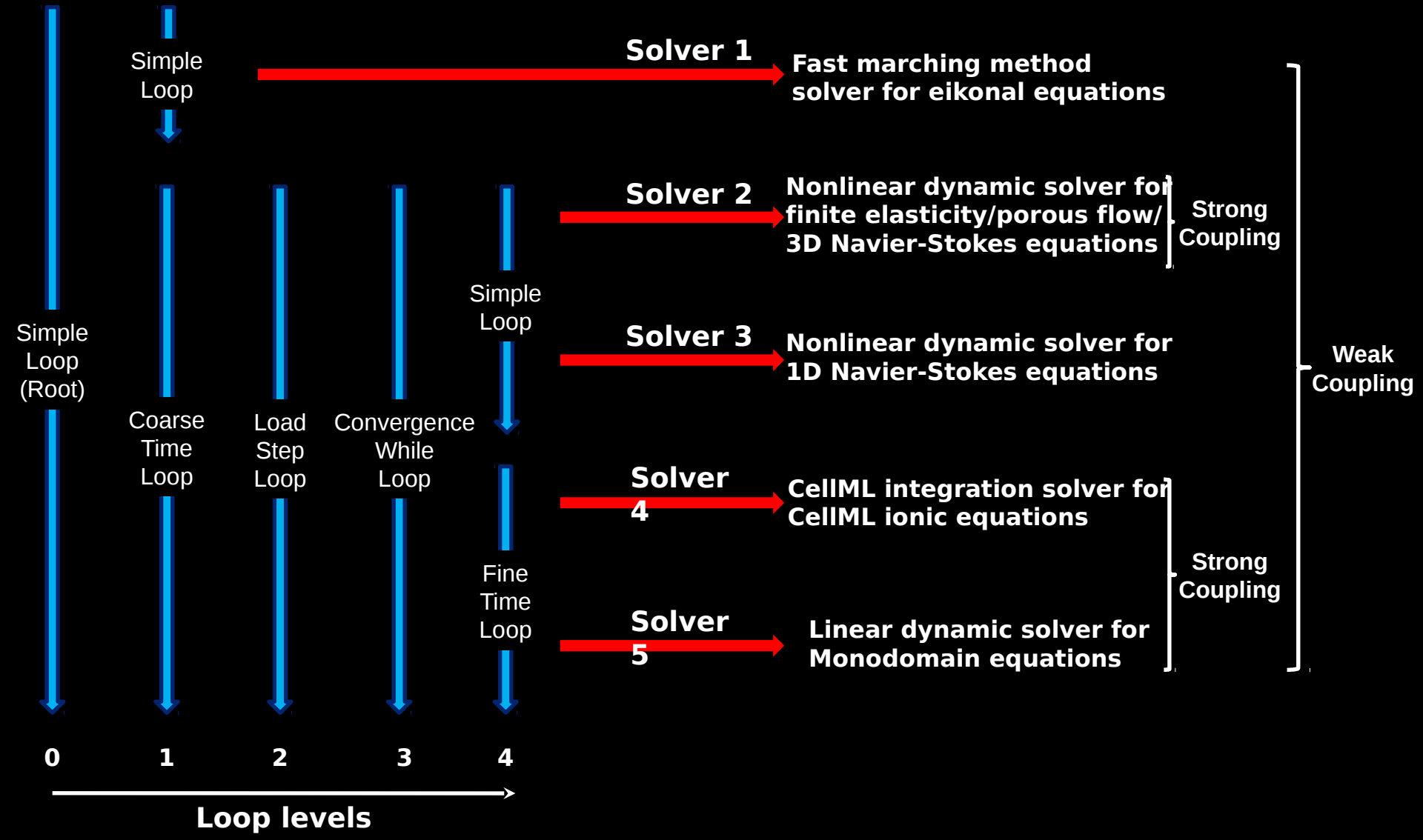
Control Loops

- Each control loops can be of three types
  - Simple – loop executes once
  - Iterative – loop executes a specified number of times. There are additional parameters this loop i.e., a control variable of a start value plus an increment.
  - Conditional – loop executes until a condition is met e.g., a convergence condition.
- Each control loop has a number of sub loops.
- If a control loop has no sub loops it can have a number of solvers.
- Each solver deals with a particular numerical problem solver.

Control Loops and Solvers



Control Loops and Solvers



Solvers

- Each solver just deals with a particular numerical problem.
- The numerical problem is formed by adding in equations sets (which can come from different regions or physical problems).
- Solver mapping defines how the rows and columns of the equations sets map to the rows and columns of the solution rows and columns.
- Solver mapping defined by mapping equations set variables onto solver matrices and vectors.

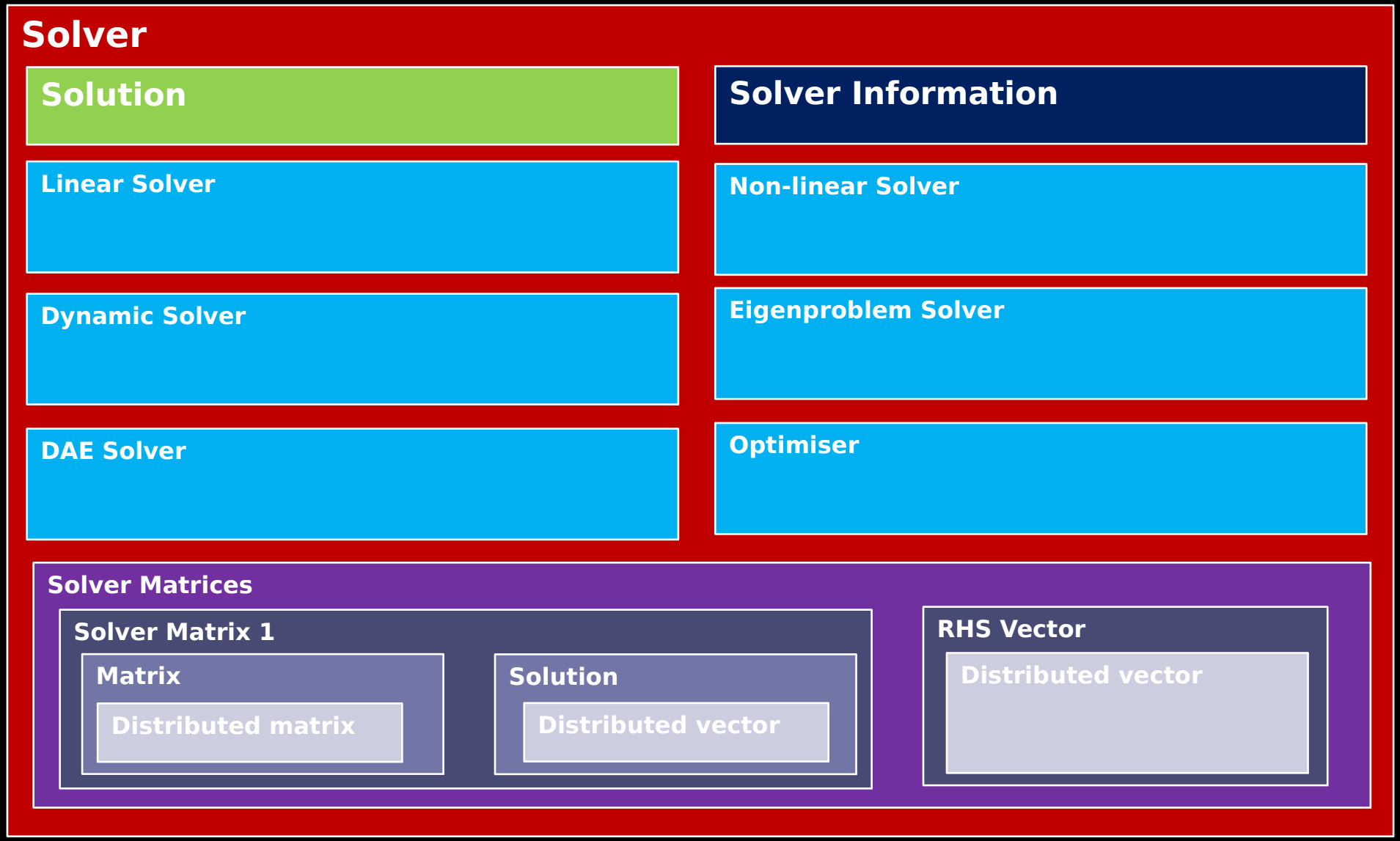
Solvers

- A solver solves a numerical problem.
- Main solver types:
  - Linear solver
  - Direct solver
  - Iterative solver
  - Nonlinear solver
  - Newton methods
    - Line search
    - Trust region
  - Dynamic solver
  - Explicit/Implicit – Euler, Backward Euler, Crank Nicholson, Newmark
  - Linear, quadratic, cubic time stepping
  - ODE integration solver
  - Euler, Improved Euler, Runga-Kutta, Adams-Moulton, ...
  - Eigenproblem solver
  - Optimisation solver

Solvers

- Solvers have a number of solver matrices and solver vectors.
- There are a number of solver libraries for each solver type.
- Main libraries:
  - CMISS (self coded)
  - PETSc
  - Sundials
  - Hypre
  - Trilinos
- Aim to be able to include new libraries and have OpenCMISS produce the solver matrices and vectors in the correct form.

Solver


Solver
Solution
Solver Information
Linear Solver
Non-linear Solver
Dynamic Solver
Eigenproblem Solver
DAE Solver
Optimiser
Solver Matrices
Solver Matrix 1
Matrix
Distributed matrix
Solution
Distributed vector
RHS Vector
Distributed vector

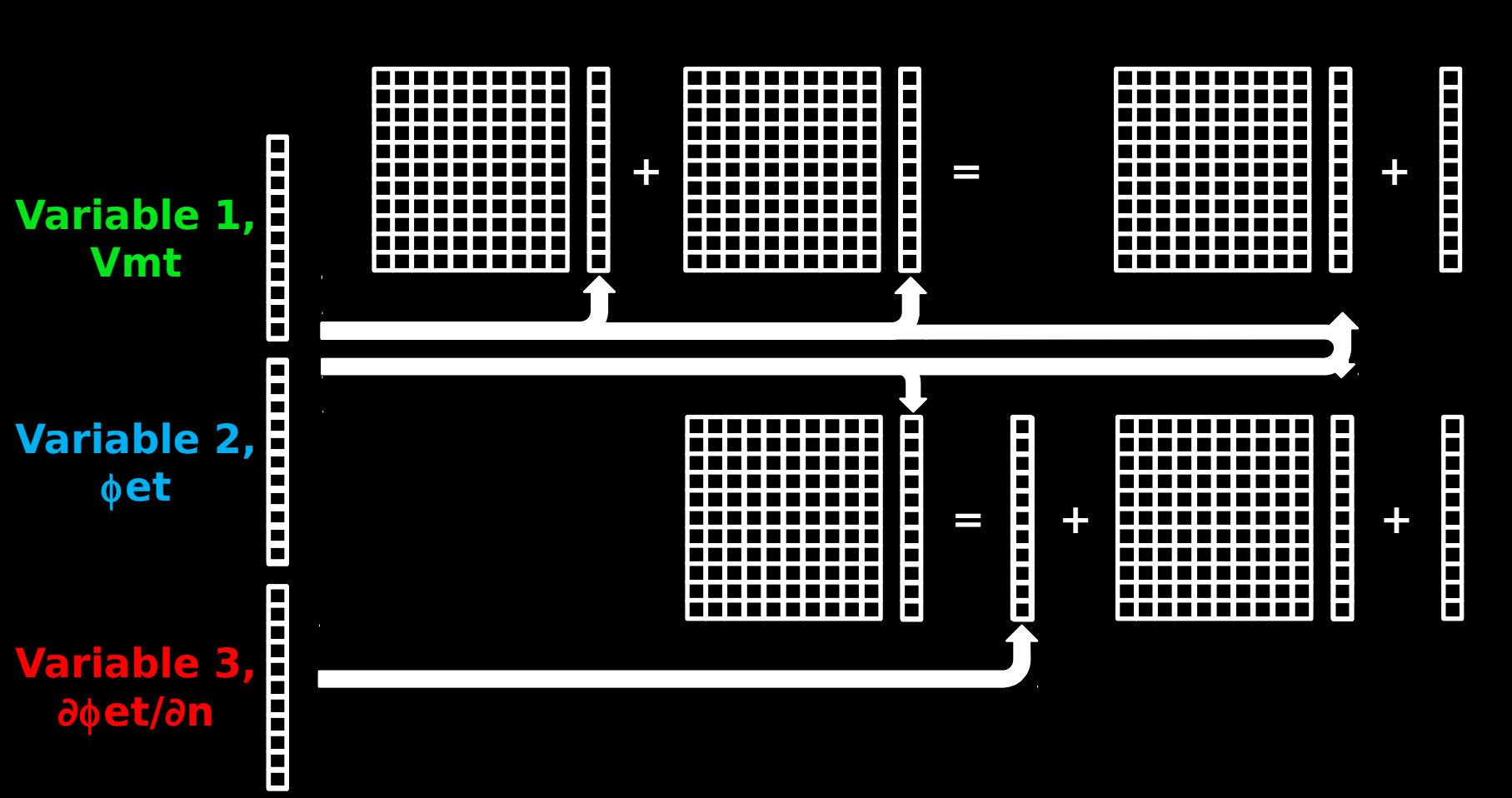
Multiphysics: Coupled Systems

- There are three main types of coupled systems:
  1. Coupled PDE systems within a region
  2. Strongly coupled systems between regions
  3. Weakly coupled systems between regions.

Coupled PDE systems

- Coupled PDE systems within the same region/mesh are handled with multiple field variables.
- For example consider the previously stated bidomain equations.
- For a FEM formulation the three field variables of interest are thus, $\mathbf{V}m$, ϕe and $\nabla \phi e \bullet \mathbf{n}$.
- Use two equations sets for the two different PDE equations (i.e., parabolic and elliptic equation).

Coupled PDE systems



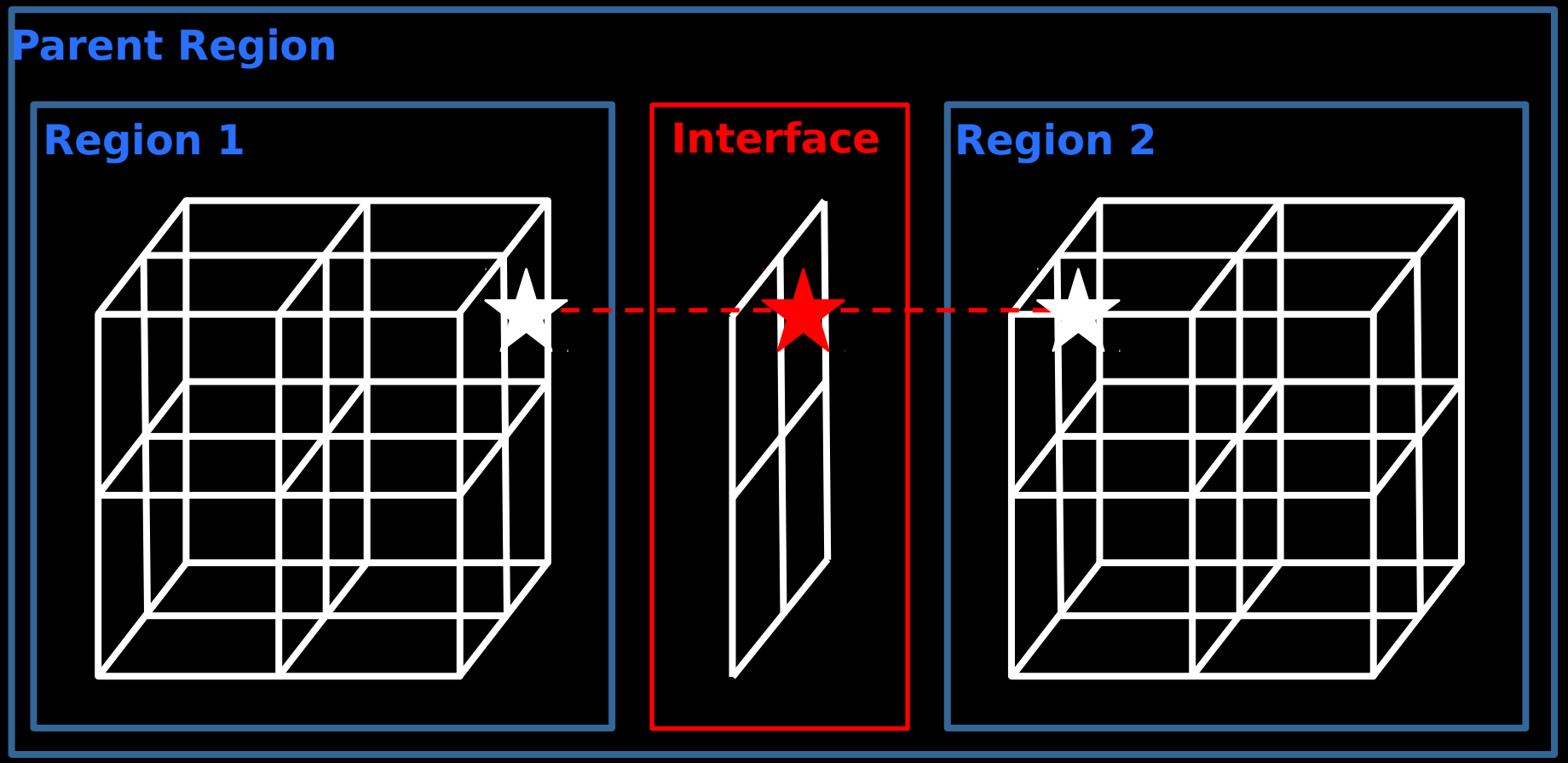
$$x \longrightarrow C\,\frac{Vmt - Vmt\text{-}1}{\Delta t} + Ki\,Vmt = Ki\,\phi et\text{-}1 + s1$$

$$\longrightarrow Ki{+}e\,\phi et = f + Ki\,Vmt + s2$$

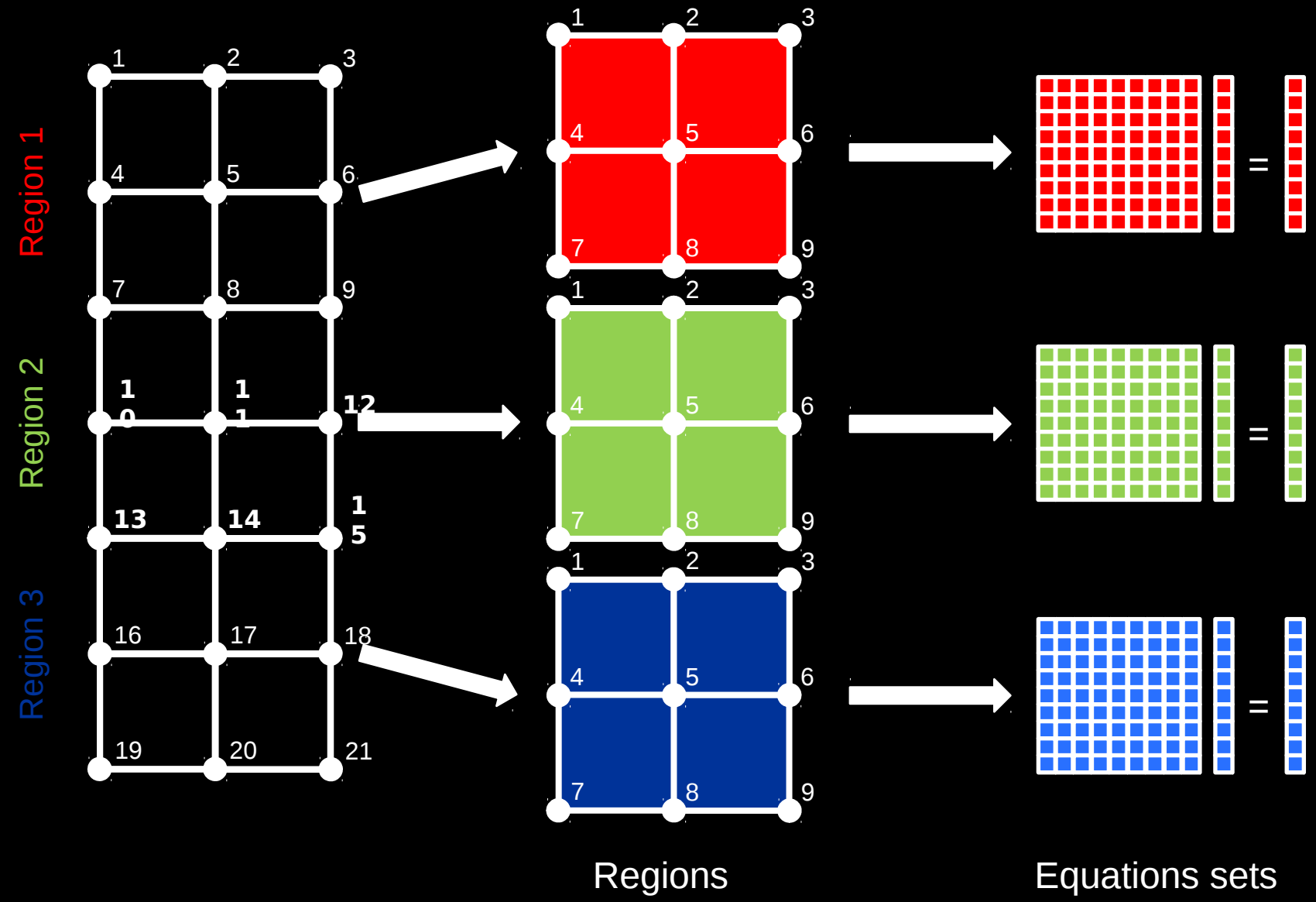
Inter region coupled systems

- To achieve a coupled system between regions/meshes interfaces are used.
- Interfaces are created on a parent region containing the two interface regions or meshes to coupled together.
- Interface nodes and an interface mesh is then defined on the interface and mapped to the surfaces of the two meshes.

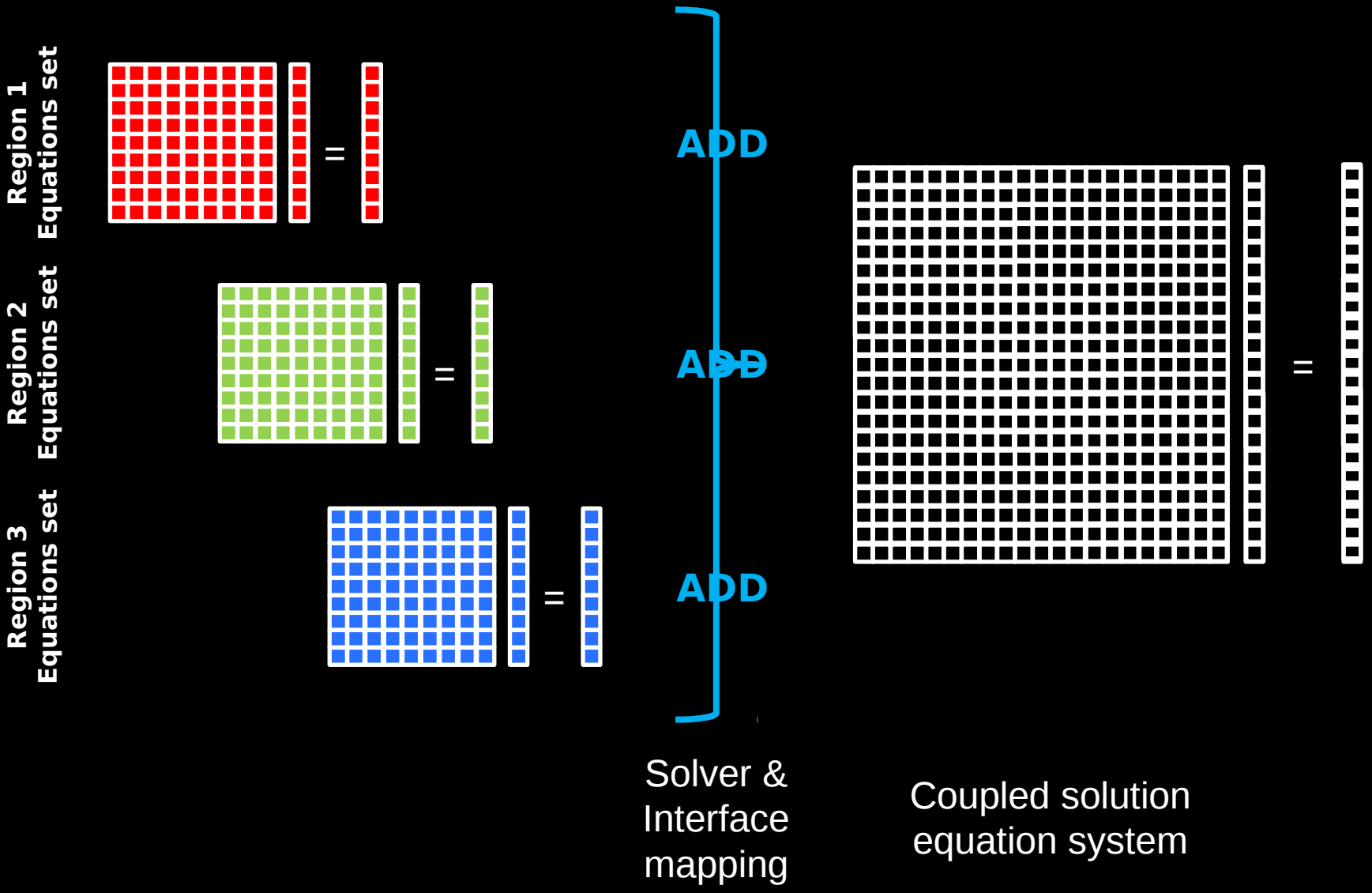
Interfaces



Strongly coupled system



Strongly coupled system



Interface Conditions

- Once an interface has been defined a number of interface conditions can be defined that specify the coupling.
- Coupling methods include
  - Linear DOF coupling
  - Lagrange Multipliers
  - Agumented Lagrange and Penalty methods
- For Lagrange and Penalty methods the coupling is defined via an interface operator.

Interface Conditions – Lagrange multipliers

- If the equations set on the first mesh is $\mathbf{A}1\mathbf{x}1=\mathbf{b}1$ and the equations set on the second mesh is $\mathbf{A}2\mathbf{x}2=\mathbf{b}2$.
- Define a Lagrange multiplier field on the interface mesh - λ.
- Define the interface operator which relates the two equations sets.

Interface Conditions – Lagrange multipliers

- To couple **A**1**x**1=**b**1 and **A**2**x**2=**b**2 we form the system

- Where **M**, **N** and **V** are the coupling matrix and α is the penalty parameter.

$$\begin{bmatrix} A1 & 0 & M \\ 0 & A2 & N \\ MT & NT & \frac{1}{\alpha}V \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ \lambda \end{bmatrix} = \begin{bmatrix} b1 \\ b2 \\ b3 \end{bmatrix}$$

Basic Structure of Code

CMISSInitialise (For Fortran and C)

9. …
10. Equations Sets
11. Problems
12. Control Loops
13. Solvers
14. Solver Equations
15. Boundary Conditions
16. Problem Solution
17. Export of Solution

CMISSFinalise

Review Example

Compiling Code

There are a few steps to compile the OpenCMISS library.

```
cd ${OPENCMISS_ROOT}/cellml
make
cd ${OPENCMISS_ROOT}/cm
make
make python
```

Compiling Examples

You should now be ready to compile (or run) the Laplace example).

```
cd ${OPENCMISS_ROOT}/ClassicalField/Laplace/Laplace/Fortran
make
make run
```

Or

```
cd ${OPENCMISS_ROOT/ClassicalField/Laplace/Laplace/Python
Python LaplaceExample.py
```

Running in parallel – mpd setup

```
cd ~/.
touch .mpd.conf
```

Edit .mpd.conf and add

```
MPD_SECRETWORD=<your secret word>
```

DO NOT use your password for your secret word. Now

```
chmod 600 .mpd.conf
```

Now edit a file called hostfile.list and add the following lines (one line per processor)

```
hpc5.bioeng.auckland.ac.nz
```

Finally add the following to your login script

```
setenv MP_HOSTFILE <path to your hostfile.list file>
```

Running in parallel

For bash:

```
cd ${OPENCMISS_ROOT}/ClassicalField/Laplace/Laplace/Fortran
```

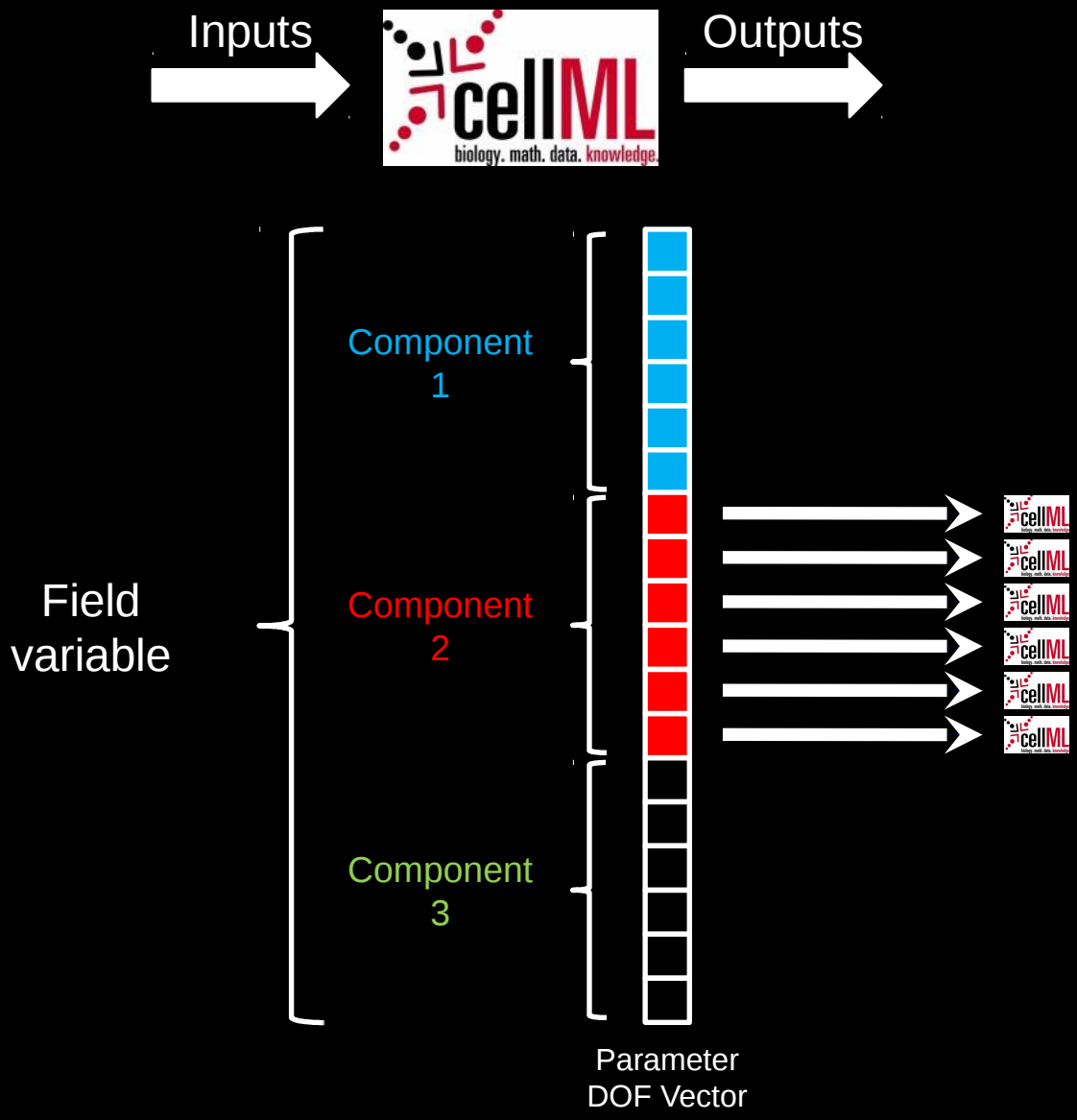
Run a MPI example using mpiexec i.e.,

```
mpd &
mpiexec.mpd –n #processors binary
```

To see what is happening get another terminal up on hpc5 and type top.

```
mpiexec.mpd –n 4
   bin/x86_64-linux/mpich2/gnu_4.6/FortranExample-debug
   10 10 10 1
```

CellML as a DOF black box



OpenCMISS CellML Import

CellML Environment

CellML Model

CellML Variables

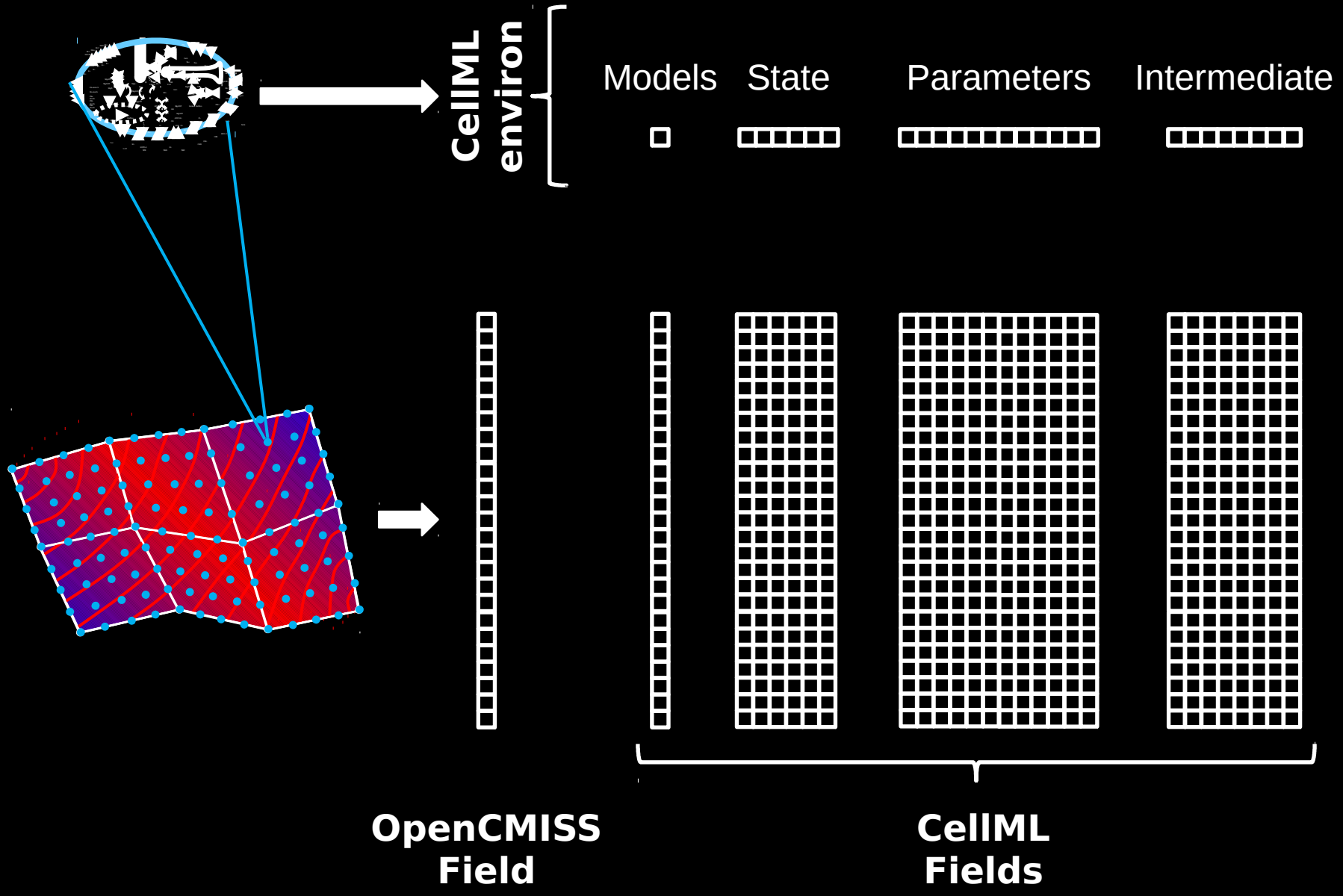
State Parameters Intermediate

State Parameters Intermediate

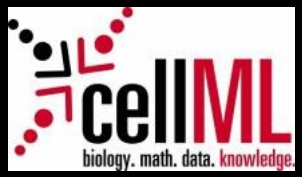
```
void cor  teRates(        double VOI,
                          double *PARAMETERS,
                          double *RATES,
                          double *STATES,
                          double *INTERMEDIATE);
{
                          const double FIXED[..];

                          :
                          :
}
```

Import

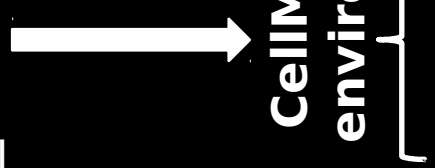
CellML Interface



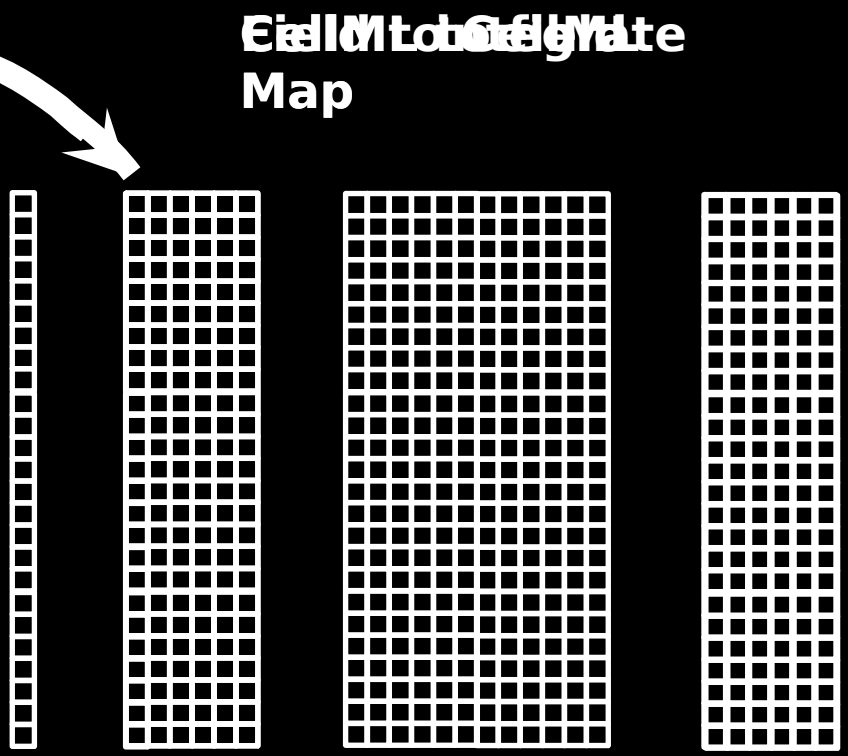
CellML/Field Maps



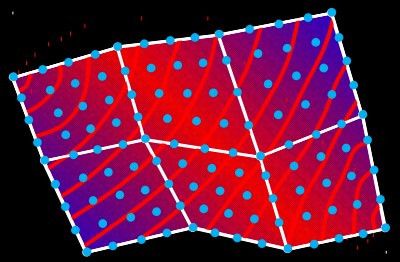
Cellular model



Models State Parameters Intermediate

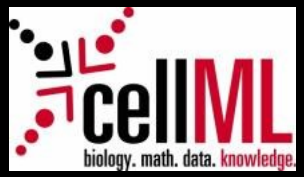


Map



Transmembrane
Voltage Field

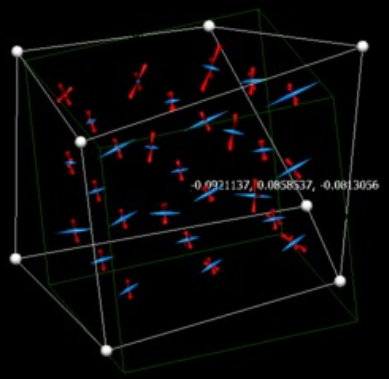
CellML/Field Maps

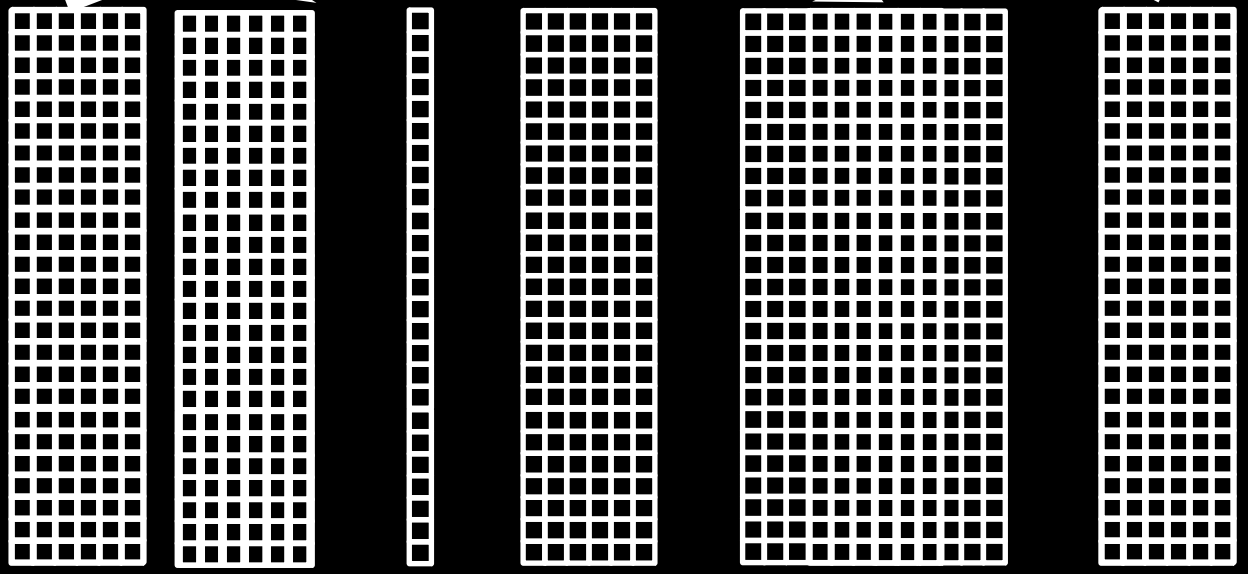


Constitutive Law

CellML environ

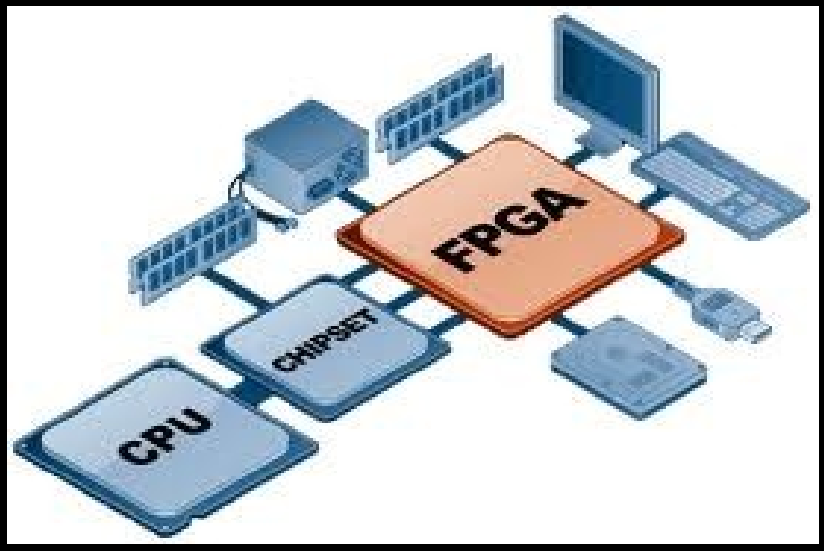
Models State Parameters Intermediate





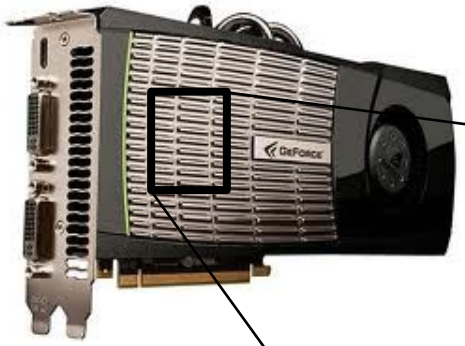
StrainStress Field Field

FPGA Acceleration

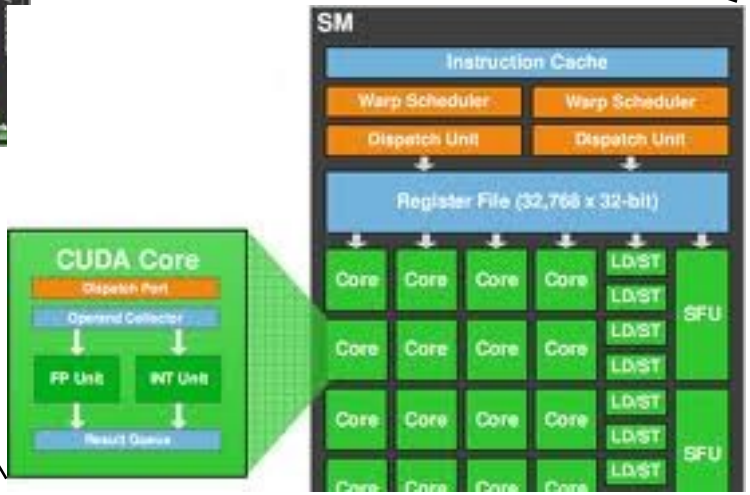


Investigating using Field Programmable Gate Array (FPGAs) as accelerators for CellML equations and/or fast special function (e.g., exponential) acceleration.

GPU Acceleration



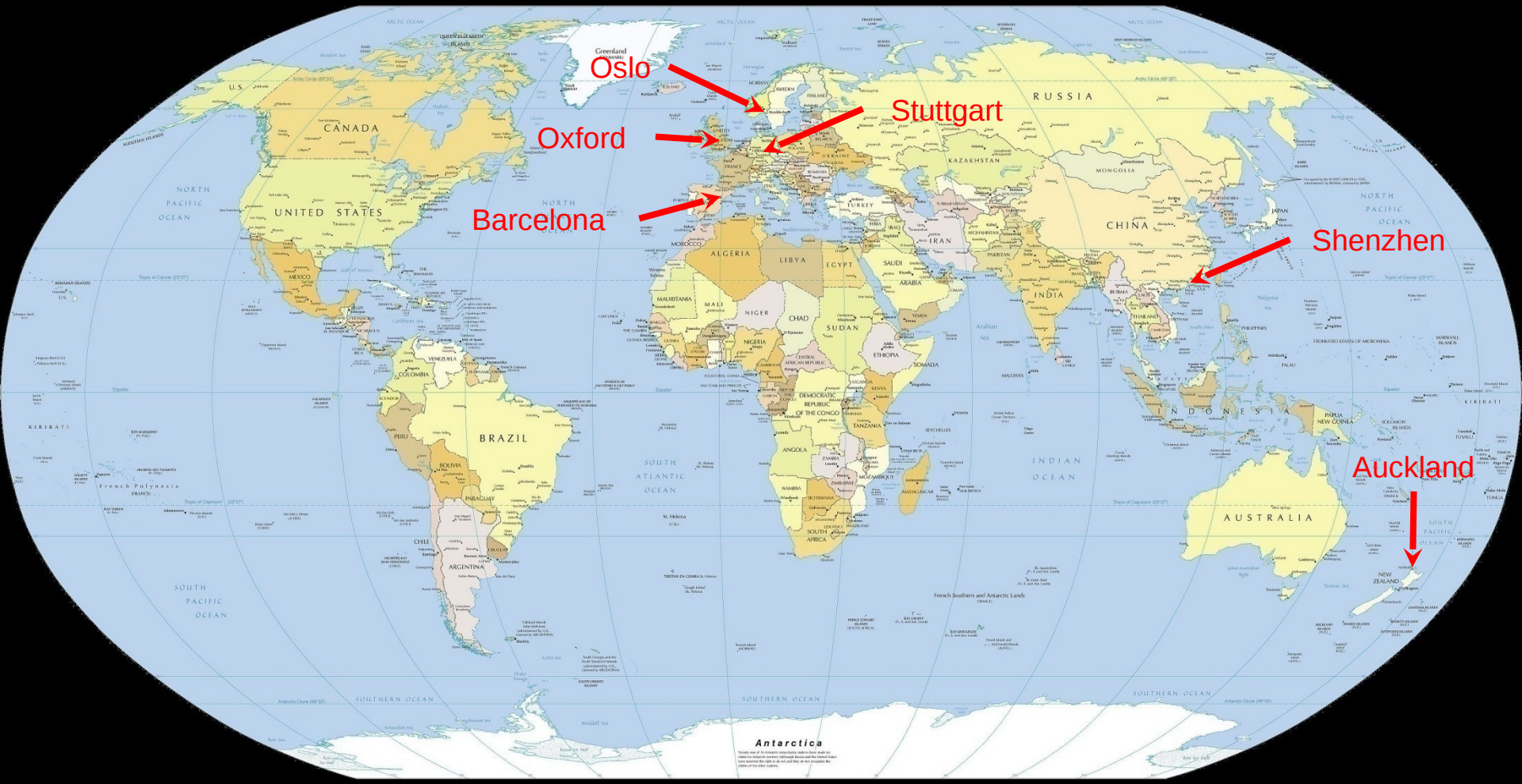
NVIDIA CUDA



OpenCL

cellML
biology. math. data. knowledge.

Current OpenCMISS development



Acknowledgements

- A (hopefully complete) list of those individuals who have contributed to OpenCMISS (so far….)

- Bojan Blazevic
- Andy Bowery
- Chris Bradley
- Randall Britten
- Vincent Budelmann
- Phani Chichapatnam
- Richard Christie
- Andrew Cookson
- Andrew Crozier
- Prasad Gamage
- Arne Gjuvsland
- Thomas Heidlauf
- Alice Hung
- Peter Hunter
- Jagir Hussan
- Chloe Irwin Whitney
- Jessica Jor
- Sebastian Krittian
- David Ladd
- Sander Land
- Jack Lee
- Caton Little
- Xaio Bo Lu
- Kumar Mithraratne
- Christian Michler
- Jennine Mitchell
- Martyn Nash
- David Nickerson
- Steven Niederer
- Poul Nielsen
- Øyvind Nordbø
- David Nordsletten
- Stig Omholt
- Ali Pashaei
- Vijayaraghavan Rajagopal
- Adam Reeve
- Oliver Rohle
- Ishani Roy
- Ole W. Saastad
- Soroush Safaei
- Vickie Shim
- Matt Sinclair
- Nic Smith
- Martin Steghofer
- Merryn Tawhai
- Mark Trew
- Jon Olav Vik
- Tim Wu
- Robert Jiahe Xi
- Nancy (Xiani) Yan
- Ting Yu
- Heye Zhang
- Ju Zhang

Object interface

- How to define objects for the library?
- Want:
  - Simple interface for a variety of scripting and programming languages.
  - User friendly in that the library should sensibly set default parameters to minimise the amount of information the programmer/user has to send.
  - Should be extensible so that extra parameters can be added at a later stage without causing problems.

Object interface

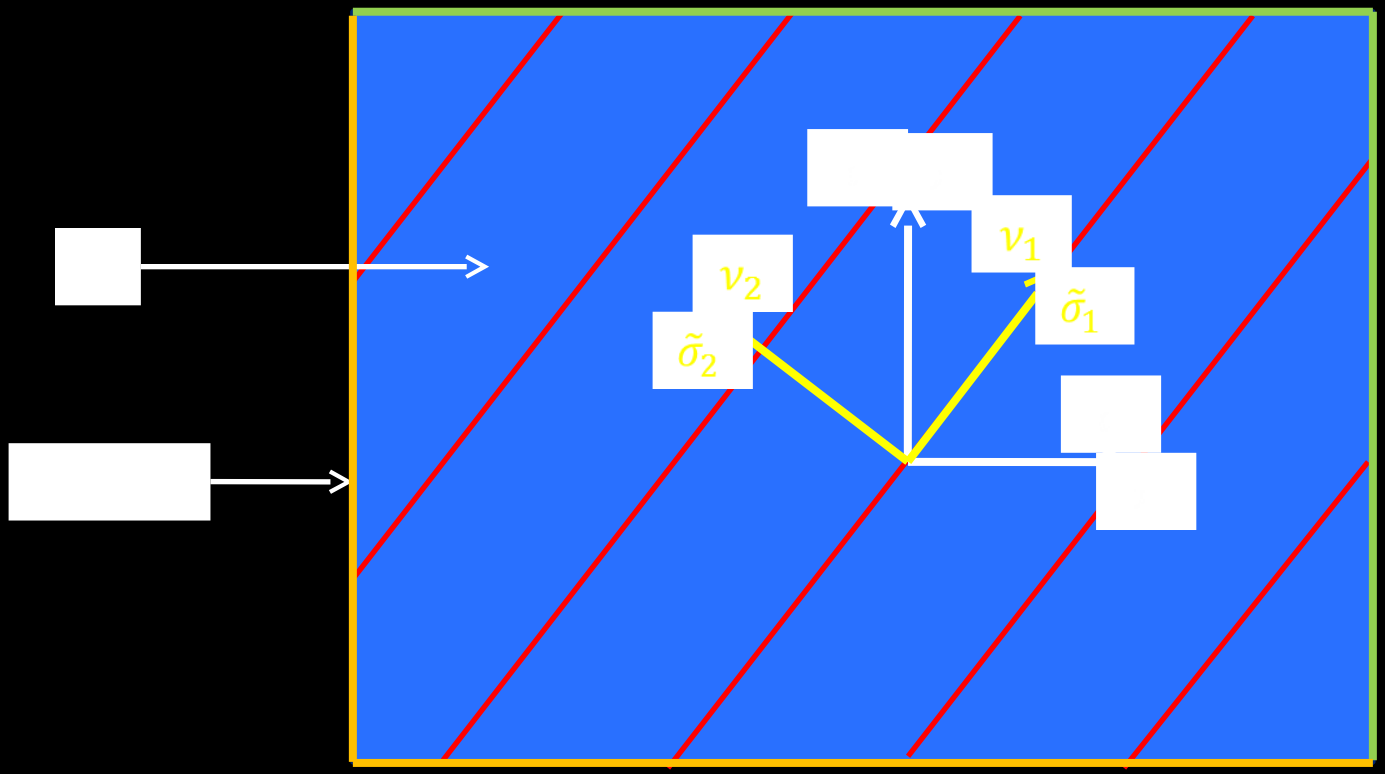
- Objects identified by a unique user number.
- Objects are initialised with a `Object_TypeInitialise` call.
- Objects are finalised with a `Object_TypeFinalise` call.
- Objects are created with a pair of `Object_CreateStart` and `Object_CreateFinish` calls.
- Objects are destroyed with a `Object_Destroy` call.
- Object parameters are set with a number of `Object_ParameterSet` calls inside the START and FINISH calls.
- START call initialises OBJECT and sets default parameters. SET calls modify default parameters. FINISH call finalises the object.

Example, basis functions

```
!Initialise the basis type.
CALL CMISSBasis_TypeInitialise(Basis,Err)
!Start the creation of a basis with a user number of 1.
!The default is tri-linear Lagrange
CALL CMISSBasis_CreateStart(1,Basis,Err)
!Set the number of xi directions to 2.
CALL CMISSBasis_NumberOfXiSet(Basis,2,Err)
!Set the interpolation to be cubic Hermite*quadratic Lagrange.
CALL CMISSBasis_InterpolationXiSet(Basis, &
 & [CMISSCubicHermiteInterpolation, &
 & CMISSQuadraticLagrangeInterpolation],Err)
!Finish the creation of the basis.
CALL CMISSBasis_CreateFinish(Basis,Err)
!Destroy the basis with the user number 1.
CALL CMISSBasis_Destroy(Basis,Err)
!Finalise the basis functions.
CALL CMISSBasis_TypeFinalise(Basis,Err)
```

Generalised Laplace equation

$$\phi(\boldsymbol{x}) = \phi_d(\boldsymbol{x}), \boldsymbol{x} \in \Gamma_d$$



$$q(\boldsymbol{x}) = \boldsymbol{\sigma}(\boldsymbol{x})\nabla\phi(\boldsymbol{x}) \cdot \boldsymbol{n} = q_n(\boldsymbol{x}), \boldsymbol{x} \in \Gamma_n$$

Generalised Laplace equation

Generalised Laplace equation

Generalised Laplace equation

Generalised Laplace equation

Generalised Laplace equation

Generalised Laplace equation

Compiling OpenCMISS

To compile the main OpenCMISS library

```
cd ${OPENCMISS_ROOT}/cm
make
```

To compile an example (e.g., Laplace)

```
cd ${OPENCMISS_ROOT}/examples/ClassicalField/Laplace/Laplace
Make USECELLML=false
```

To run you can simply

```
make run
```

Or execute the appropriate binary file in the bin directories.

Diffusion equation

The generalised diffusion equation is

The weak form of this equation is

Or, in tensorial form after applying the divergence theorem

Diffusion equation

Diffusion equation

Diffusion equation

That is

Diffusion equation

Or, taking the values that are constant outside the integrals

Diffusion equation

Which is an equation of the form

where

With the elemental matrices given by

Finite elasticity equations

Finite elasticity equations

Finite elasticity equations

Finite elasticity equations

Finite elasticity equations

Finite elasticity equations

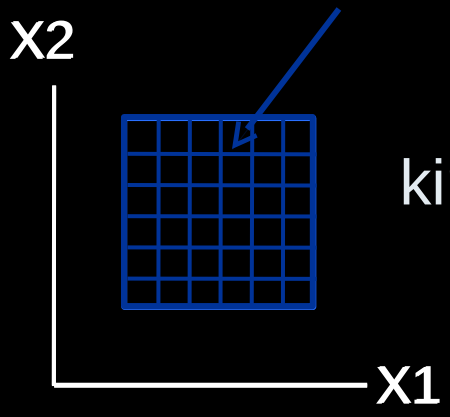
Finite elasticity equations

Finite elasticity equations

Finite elasticity equations

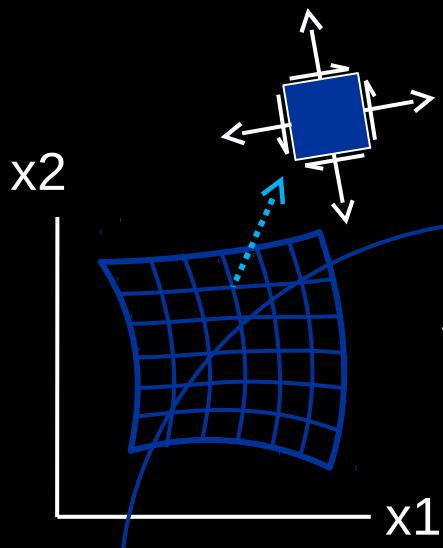
Consider a small block of tissue

define material coordinates



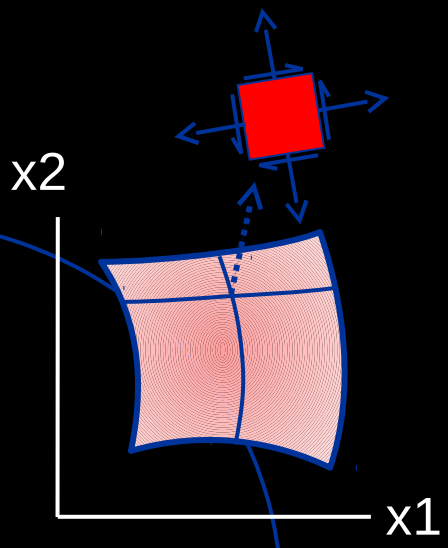
kinematics

strain tensor
relative to the
material axes



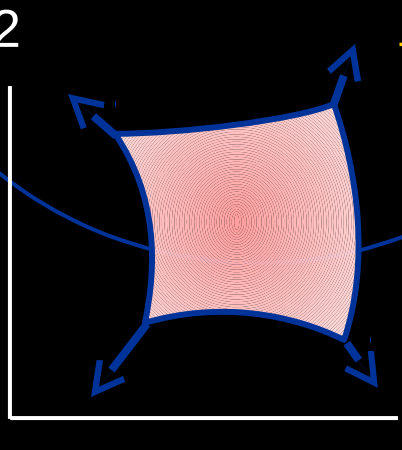
stress

stress tensor
relative to the
material axes



requires constitutive law
(defines material properties)

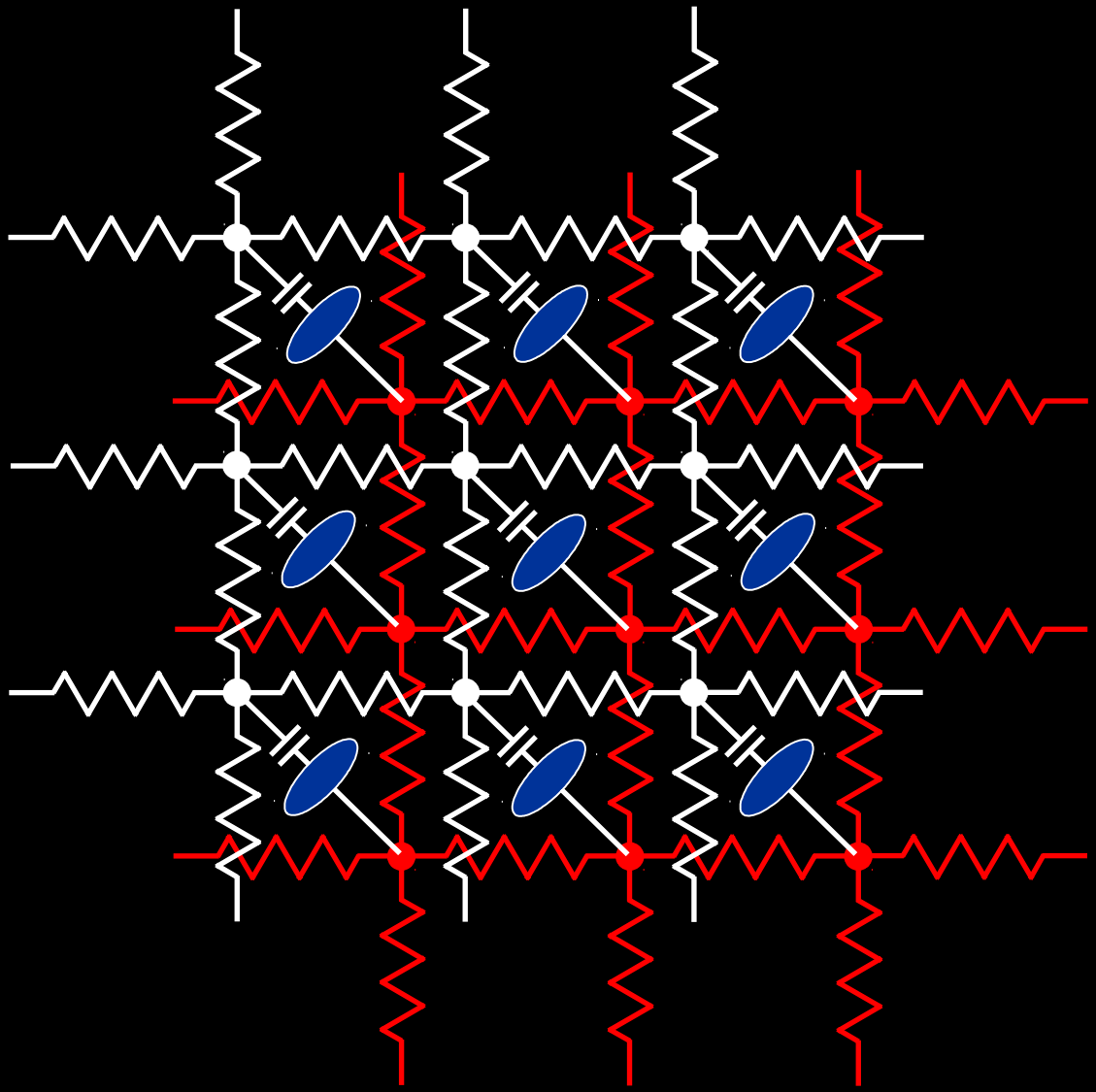
Solve the equations
of motion derived
from physical
conservation laws



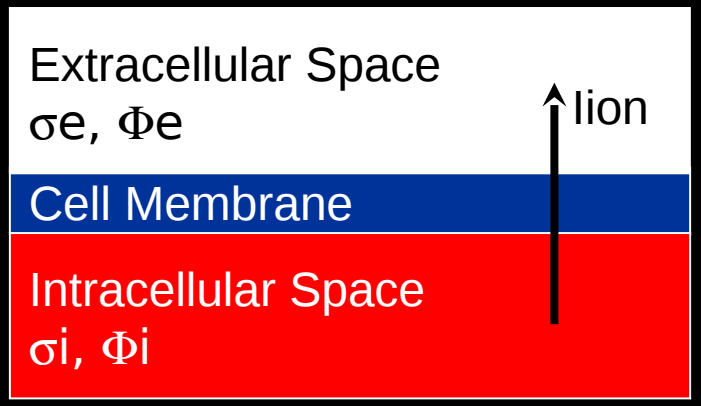
Finite elasticity equations

Finite elasticity equations

Example – Electrical Activation (Bidomain representation)

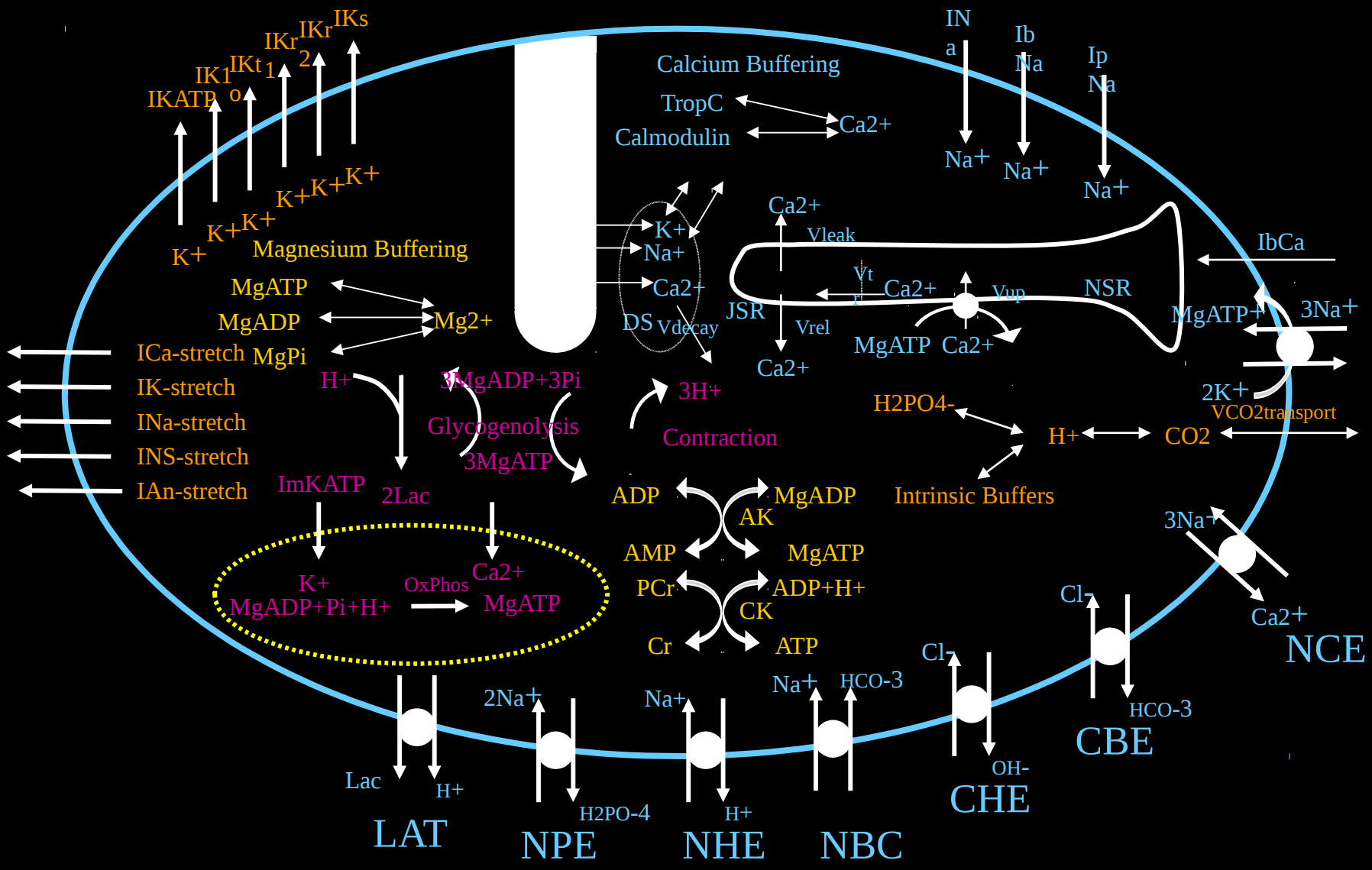


Cell model =



Bidomain equations

Cellular processes



Boundary conditions

No current flow from the intracellular domain to the extramyocardial domain i.e.,

or

Current flow from the extracellular domain is matched by the extramyocardial domain i.e.,

If there are any applied voltages then

Monodomain equations

Operator Splitting

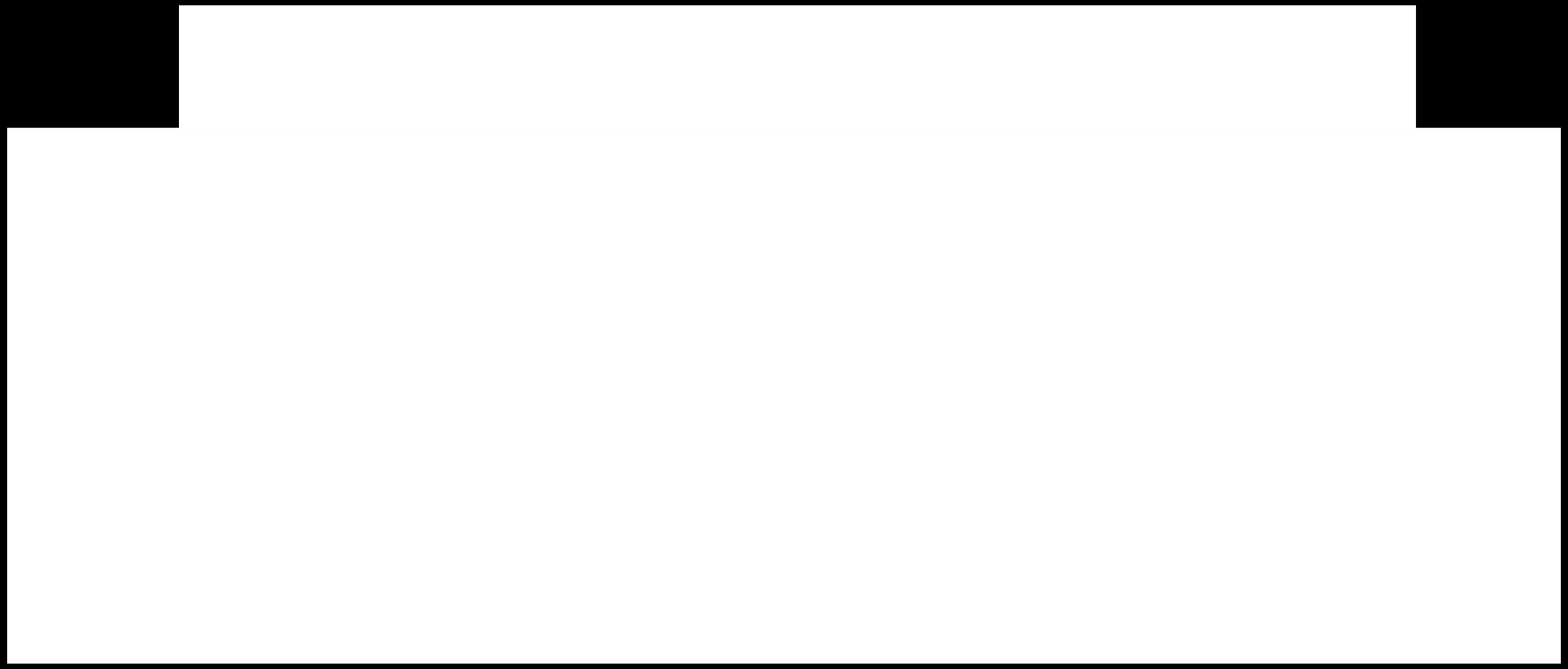
Gudunov Operator Splitting

Strang Operator Splitting 1

Strang Operator Splitting 2

Dynamic Solver

The general form of dynamic equations is



Note: 2nd order dynamic equations have mass terms, 1st order dynamic equations only have damping terms

Dynamic Solver

where

and

is an unknown vector

Dynamic Solver

Substituting and taking a weighted residual approach

Dynamic Solver

Dynamic Solver

Dynamic Solver

Dynamic Solver

Dynamic Solver

Dynamic Solver

Dynamic Solver

Dynamic Solver Example

CellML field interface

```
!Create the equations set source field
CALL EQUATIONS_SET_SOURCE_CREATE_START(SOURCE_FIELD_USER_NUMBER,EQUATIONS_SET, &
    & ERR,ERROR,*999)
CALL EQUATIONS_SET_SOURCE_CREATE_FINISH(EQUATIONS_SET,ERR,ERROR,*999)
CALL EQUATIONS_SET_SOURCE_FIELD_GET(EQUATIONS_SET,SOURCE_FIELD,ERR,ERROR,*999)

!Setup the CellML environment on the host source field
CALL CELLML_CREATE_START(CELLML_USER_NUMBER,SOURCE_FIELD,CELLML,ERR,ERROR,*999)
!Finish creating CellML environment
CALL CELLML_CREATE_FINISH(CELLML,ERR,ERROR,*999)

!Setup the CellML models
CALL CELLML_MODELS_CREATE_START(CELLML,ERR,ERROR,*999)
!Import the variants
CALL CELLML_MODELS_IMPORT(MODEL_USER_NUMBER,CELLML,URI,ERR,ERROR,*999)
!Finish creating the variants.
CALL CELLML_MODELS_CREATE_FINISH(CELLML,ERR,ERROR,*999)

!Setup the CellML model field i.e., the model variable number which is used to
!evaluate the value at each of the host field dofs.
CALL CELLML_MODELS_FIELD_CREATE_START(MODEL_FIELD_USER_NUMBER,CELLML,ERR,ERROR,*999)
CALL CELLML_MODELS_FIELD_CREATE_FINISH(CELLML,ERR,ERROR,*999)
```

```
!Assign model variable numbers to the CellML models field
CALL CELLML_MODELS_FIELD_GET(CELLML,CELLML_MODELS_FIELD,ERR,ERROR,*999)
CALL CELLML_MODELS_VARIABLE_NUMBER_GET(CELLML,MODEL_USER_NUMBER, &
    MODEL_VARIABLE_URI,MODEL_VARIABLE_NUMBER,ERR,ERROR,*999)
CALL FIELD_PARAMETER_SET_UPDATE_GRID(CELLML_MODELS_FIELD,FIELD_VALUES_SET_TYPE, &
    & 1,1,1,FIELD_STANDARD_VARIABLE_TYPE,MODEL_USER_VARIABLE_NUMBER,ERR,ERROR,*999)

!Setup the CellML state model field
CALL CELLML_MODEL_STATE_FIELD_CREATE_START(STATE_FIELD_USER_NUMBER,CELLML, &
    & ERR,ERROR,*999)
!Finish creating the state variable field. N.b. Default field values are the
!default for the state variable.
CALL CELLML_MODEL_STATE_FIELD_CREATE_FINISH(CELLML,ERR,ERROR,*999)
!Change initial conditions etc. e.g., [Ca2+] to 1.0 mMol
CALL CELLML_MODELS_STATE_FIELD_GET(CELLML,CELLML_MODELS_STATE_FIELD,ERR,ERROR,*999)
CALL CELLML_MODELS_STATE_VARIABLE_COMPONENTS_GET(CELLML,MODEL_USER_NUMBER, &
    & Ca_URI,VARIABLE_COMPONENT_NUMBER,ERR,ERROR,*999)
CALL FIELD_PARAMETER_SET_UPDATE_GRID(CELLML_MODELS_STATE_FIELD, &
    & FIELD_VALUES_SET_TYPE,1,1,VARIABLE_COMPONENT_NUMBER, &
    & FIELD_STANDARD_VARIABLE_TYPE,1.0_DP,ERR,ERROR,*999)
```

```
!Setup varying parameters for each intermediate
CALL CELLML_MODEL_INTERMEDIATE_FIELD_CREATE_START(INTERMEDIATE_FIELD_USR_NUMBER,&
    & CELLML,ERR,ERROR,*999)
!Set the varying parameters
CALL CELLML_MODEL_INTERMEDIATE_ADD(CELLML,INTERMEDIATE_NUMBER/URI,IKr_URI, &
    & ERR,ERROR,*999)
!Finish the creation of the varying parameters.
CALL CELLML_MODEL_INTERMEDIATE_FIELD_CREATE_FINISH(CELLML,ERR,ERROR,*999)

!Setup varying parameters for each variant
CALL CELLML_MODELS_PARAMETERS_CREATE_START(CELLML,ERR,ERROR,*999)
!Set the varying parameters
CALL CELLML_MODELS_PARAMETERS_VARYING_ADD(CELLML,MODEL_NUMBER,gKr_URI, &
    & ERR,ERROR,*999)
!Finish the creation of the varying parameters.
CALL CELLML_MODELS_PARAMETER_CREATE_FINISH(CELLML,ERR,ERROR,*999)


!Setup the CellML variants parameter values field. The number of components in
this field will be equal to the sum of the number of varying parameters across all
variants.
CALL CELLML_MODELS_PARAMETER_FIELD_CREATE_START(PARAM_FIELD_USER_NUMBERS,CELLML, &
    & ERR,ERROR,*999)
!Get the CellML state variable field
CALL CELLML_MODELS_PARAMETER_FIELD_GET(CELLML,MODEL_USER_NUMBER/URI,PARAM_FIELD, &
    & ERR,ERROR,*999)
!Get gKr parameter component
CALL CELLML_MODELS_PARAMETER_FIELD_COMPONENT_GET(CELLML,MODELS_USER_NUMBER/URI, &
    & gKr_USER_NUMBER/gKr_URI,gKr_COMPONENT_NUMBER,ERR,ERROR,*999)
!Set field properties
CALL FIELD_COMPONENT_INTERPOLATION_SET(PARAM_FIELD,gKr_COMPONENT_NUMBER, &
    & NODE_BASED_INTERPOLATION,ERR,ERROR,*999)
```

```
!Finish creating the variants parameter field. This will then create an embedded
!field which contains the working data at the level of the source dofs. It will
!contain, for example, the values of the parameters interpolated at the host
!dofs and any extra components for integrator working data.
CALL CELLML_MODELS_PARAMETER_FIELD_CREATE_FINISH(CELLML,ERR,ERROR,*999)
!Change default parameters etc.
CALL FIELD_PARAMETER_SET_UPDATE_NODE(PARAM_FIELD,FIELD_VALUES_SET_TYPE,1,1, &
    & gKr_COMPONENT_NUMBER,FIELD_STANDARD_VARIABLE_TYPE,0.05_DP, &
    & ERR,ERROR,*999)


!Generate/instatiate the CellML i.e., compile, link etc. Would first check for !
valid setup.
CALL CELLML_GENERATE(CELLML,ERR,ERROR,*999)
```

Example – Laplace equation library calls

```
!Initialise CMISS.
CALL CMISS_INITIALISE(ERR,ERROR,*999)

!Start the creation a coordinate system (default 3D RC)
CALL COORDINATE_SYSTEM_CREATE_START(1,COORDINATE_SYSTEM,ERR,ERROR,*999)
!Set the coordinate system to be 2D
CALL COORDINATE_SYSTEM_DIMENSION_SET(COORDINATE_SYSTEM,2,ERR,ERROR,*999)
!Finish the creation a coordinate system
CALL COORDINATE_SYSTEM_CREATE_FINISH(COORDINATE_SYSTEM,ERR,ERROR,*999)

!Start the creation of a region with a coordinate system.
CALL REGION_CREATE_START(1,COORDINATE_SYSTEM,REGION,ERR,ERROR,*999)
!Finish the creation of a region
CALL REGION_CREATE_FINISH(REGION,ERR,ERROR,*999)

!Start the creation of a basis (default tri-linear Lagrange)
CALL BASIS_CREATE_START(1,BASIS,ERR,ERROR,*999)
!Set the number of Xi directions to 2 (bi-linear Lagrange)
CALL BASIS_NUMBER_XI_SET(BASIS,2,ERR,ERROR,*999)
!Finish the creation of a basis
CALL BASIS_CREATE_FINISH(BASIS,ERR,ERROR,*999)
```

```
!Start the creation of a generated mesh in the region
CALL GENERATED_MESH_CREATE_START(1,REGION,GENERATED_MESH,ERR,ERROR,*999)
!Set up a regular 100x100 mesh
CALL GENERATED_MESH_TYPE_SET(GENERATED_MESH,GENERATED_MESH_REGULAR_TYPE, &
       & ERR,ERROR,*999)
CALL GENERATED_MESH_BASIS_SET(GENERATED_MESH,BASIS,ERR,ERROR,*999)
CALL GENERATED_MESH_EXTENT_SET(GENERATED_MESH,(/2.0_DP,1.0_DP/),ERR,ERROR,*999)
CALL GENERATED_MESH_NUMBER_OF_ELEMENTS_SET(GENERATED_MESH,(/100,100/), &
       & ERR,ERROR,*999)
!Finish the creation of a generated mesh in the region
CALL GENERATED_MESH_CREATE_FINISH(GENERATED_MESH,MESH,ERR,ERROR,*999)

!Start the creation a decomposition on a mesh
CALL DECOMPOSITION_CREATE_START(1,MESH,DECOMPOSITION,ERR,ERROR,*999)
!Set the decomposition so that the domains are calculated
CALL DECOMPOSITION_TYPE_SET(DECOMPOSITION,DECOMPOSITION_CALCULATED_TYPE, &
       & ERR,ERROR,*999)
CALL DECOMPOSITION_NUMBER_OF_DOMAINS_SET(DECOMPOSITION,5,ERR,ERROR,*999)
!Finish the creation of decomposition
CALL DECOMPOSITION_CREATE_FINISH(DECOMPOSITION,ERR,ERROR,*999)
```

```
!Start the creation of a field on a region (default geometric)
CALL FIELD_CREATE_START(1,REGION,GEOMETRIC_FIELD,ERR,ERROR,*999)
!Set the decomposition the field will use
CALL FIELD_MESH_DECOMPOSITION_SET(GEOMETRIC_FIELD,DECOMPOSITION,ERR,ERROR,*999)
!Set mesh components each of the field variable components will use. NB. These
!are shown for example as each field variable component will default to the
!first mesh component.
CALL FIELD_COMPONENT_MESH_COMPONENT_SET(GEOMETRIC_FIELD, &
       & FIELD_STANDARD_VARIABLE_TYPE,1,1,ERR,ERROR,*999)
CALL FIELD_COMPONENT_MESH_COMPONENT_SET(GEOMETRIC_FIELD, &
       & FIELD_STANDARD_VARIABLE_TYPE,2,1,ERR,ERROR,*999)
!Finish the creation of a field
CALL FIELD_CREATE_FINISH(GEOMETRIC_FIELD,ERR,ERROR,*999)

!Set a field value in a parameter set
CALL FIELD_PARAMETER_SET_UPDATE_NODE(GEOMETRIC_FIELD, &
       & FIELD_VALUES_SET_TYPE,1,1,1,FIELD_STANDARD_VARIABLE_TYPE, &
       & 1.0_DP,ERR,ERROR,*999)
!Start the update of field values in a parameter set
CALL FIELD_PARAMETER_SET_UPDATE_START(GEOMETRIC_FIELD, &
       & FIELD_VALUES_SET_TYPE,ERR,ERROR,*999)
!!DO SOME CALCULATIONS
!Finish the update of field values in a parameter set
CALL FIELD_PARAMETER_SET_UPDATE_FINISH(GEOMETRIC_FIELD, &
       & FIELD_VALUES_SET_TYPE,ERR,ERROR,*999)
```

```
!Start the creation of an equations set on a region
CALL EQUATIONS_SET_CREATE_START(1,REGION,GEOMETRIC_FIELD,EQUATIONS_SET, &
       & ERR,ERROR,*999)
!Set the equations set to be standard Laplace's equation problem
CALL EQUATIONS_SET_SPECIFICATION_SET(EQUATIONS_SET, &
       &
   EQUATIONS_SET_CLASSICAL_FIELD_CLASS,EQUATIONS_SET_LAPLACE_EQUATION_TYPE, &
       & EQUATIONS_SET_STANDARD_LAPLACE_SUBTYPE,ERR,ERROR,*999)
!Finish the creation of an equations set
CALL EQUATIONS_SET_CREATE_FINISH(EQUATIONS_SET,ERR,ERROR,*999)

!Start the creation of the equations set dependent field
CALL EQUATIONS_SET_DEPENDENT_CREATE_START(EQUATIONS_SET,ERR,ERROR,*999)
!Finish the create of the problems dependent field
CALL EQUATIONS_SET_DEPENDENT_CREATE_FINISH(EQUATIONS_SET,ERR,ERROR,*999)

!Start the creation of the equations set fixed conditions
CALL EQUATIONS_SET_FIXED_CONDITIONS_CREATE_START(EQUATIONS_SET,ERR,ERROR,*999)
!Set BC's
CALL EQUATIONS_SET_FIXED_CONDITIONS_SET_DOF(EQUATIONS_SET,1, &
       & EQUATIONS_SET_FIXED_BOUNDARY_CONDITION,0.0_DP,ERR,ERROR,*999)
CALL EQUATIONS_SET_FIXED_CONDITIONS_SET_DOF(PROBLEM,10, &
       & EQUATIONS_SET_FIXED_BOUNDARY_CONDITION,1.0_DP,ERR,ERROR,*999)
!Finish the create of the equations set fixed conditions
CALL EQUATIONS_SET_FIXED_CONDITIONS_CREATE_FINISH(EQUATIONS_SET,ERR,ERROR,*999)
```

```
!Start the creation of the equations for the equations set
CALL EQUATIONS_SET_EQUATIONS_CREATE_START(EQUATIONS_SET,ERR,ERROR,*999)
!Set the equations matrices sparsity type
CALL EQUATIONS_SET_SPARSITY_TYPE_SET(EQUATIONS_SET,EQUATIONS_SET_SPARSE_MATRICES, &
   & ERR,ERROR,*999)
!Set the equations output type
CALL EQUATIONS_SET_OUTPUT_TYPE_SET(EQUATIONS_SET,EQUATIONS_SET_TIMING_OUTPUT, &
       & ERR,ERROR,*999)
!Finish the creation of the equations for the equations set
CALL EQUATIONS_SET_EQUATIONS_CREATE_FINISH(EQUATIONS_SET,ERR,ERROR,*999)

!Start the creation of a problem
CALL PROBLEM_CREATE_START(1,PROBLEM,ERR,ERROR,*999)
!Set the problem set to be standard Laplace's equation problem
CALL PROBLEM_SPECIFICATION_SET(PROBLEM,PROBLEM_CLASSICAL_FIELD_CLASS, &
       &
   PROBLEM_LAPLACE_EQUATION_TYPE,PROBLEM_STANDARD_LAPLACE_SUBTYPE,ERR,ERROR,*999)
!Finish the creation of a problem
CALL PROBLEM_CREATE_FINISH(PROBLEM,ERR,ERROR,*999)

!Start the creation of the control for a problem
CALL PROBLEM_CONTROL_CREATE_START(PROBLEM,ERR,ERROR,*999)
!Finish the creation of the control a problem
CALL PROBLEM_CONTROL_CREAT_CREATE_FINISH(PROBLEM,ERR,ERROR,*999)
```

```
!Start the creation of the solutions for the problem
CALL PROBLEM_SOLUTIONS_CREATE_START(PROBLEM,ERR,ERROR,*999)
!Add in the equations set
CALL PROBLEM_SOLUTIONS_EQUATIONS_SET_ADD(PROBLEM,CONTROL_LOOP_NODE,1,EQUATIONS_SET, &
        & EQUATIONS_SET_INDEX,ERR,ERROR,*999)
!Finish the creation of the solutions for the problem
CALL PROBLEM_SOLUTIONS_CREATE_FINISH(PROBLEM,ERR,ERROR,*999)

!Start the creation of the solvers for the problem
CALL PROBLEM_SOLVER_CREATE_START(PROBLEM,ERR,ERROR,*999)
!Get the solvers for the problem
CALL PROBLEM_SOLVER_GET(PROBLEM,CONTROL_LOOP_NODE,1,SOLVER,ERR,ERROR,*999)
!Set the type of preconditioner
CALL SOLVER_ITERATIVE_PRECONDITIONER_TYPE_SET(SOLVER, &
        & SOLVER_ITERATIVE_INCOMPLETE_LU_PRECONDITIONER,ERR,ERROR,*999)
!Set the number of iterations
CALL SOLVER_ITERATIVE_MAX_ITERATIONS_SET(SOLVER,100,ERR,ERROR,*999)
!Set the output type
CALL SOlVER_OUTPUT_TYPE_SET(SOLVER,SOLVER_TIMING_OUTPUT,ERR,ERROR,*999)
!Finish the creation of the solvers for the problem
CALL PROBLEM_SOLVER_CREATE_FINISH(PROBLEM,ERR,ERROR,*999)

!Solve the problem
CALL PROBLEM_SOLVE(PROBLEM,ERR,ERROR,*999)

!Finalise CMISS.
CALL CMISS_FINALISE(ERR,ERROR,*999)
```

Example – Code Snippets

Laplace Stiffness Matrix

```
CALL FIELD_INTERPOLATION_PARAMETERS_ELEMENT_GET(FIELD_VALUES_SET_TYPE, &
       & ELEMENT_NUMBER,PROBLEM%SOLUTION%INTERPOLATION
   %GEOMETRIC_INTERP_PARAMETERS, & & ERR,ERROR,*999)
!Loop over gauss points
DO ng=1,QUADRATURE_SCHEME%NUMBER_OF_GAUSS
  CALL FIELD_INTERPOLATE_GAUSS(FIRST_PART_DERIV,BASIS_DEFAULT_QUADRATURE_SCHEME,
   &
      & ng,PROBLEM%SOLUTION%INTERPOLATION%GEOMETRIC_INTERP_POINT,ERR,ERROR,*999)
  CALL FIELD_INTERPOLATED_POINT_METRICS_CALCULATE(GEOMETRIC_BASIS%NUMBER_OF_XI, &
      & PROBLEM%SOLUTION%INTERPOLATION%GEOMETRIC_INTERP_POINT_METRICS, &
      & ERR,ERROR,*999)
  !Calculate RWG.
  RWG=PROBLEM%SOLUTION%INTERPOLATION%GEOMETRIC_INTERP_POINT_METRICS%JACOBIAN* &
       & QUADRATURE_SCHEME%GAUSS_WEIGHTS(ng)
  !Loop over field components
  mhs=0
  DO mh=1,FIELD_VARIABLE%NUMBER_OF_COMPONENTS
    !Loop over element rows
    DO ms=1,DEPENDENT_BASIS%NUMBER_OF_ELEMENT_PARAMETERS
      mhs=mhs+1
      nhs=0
```

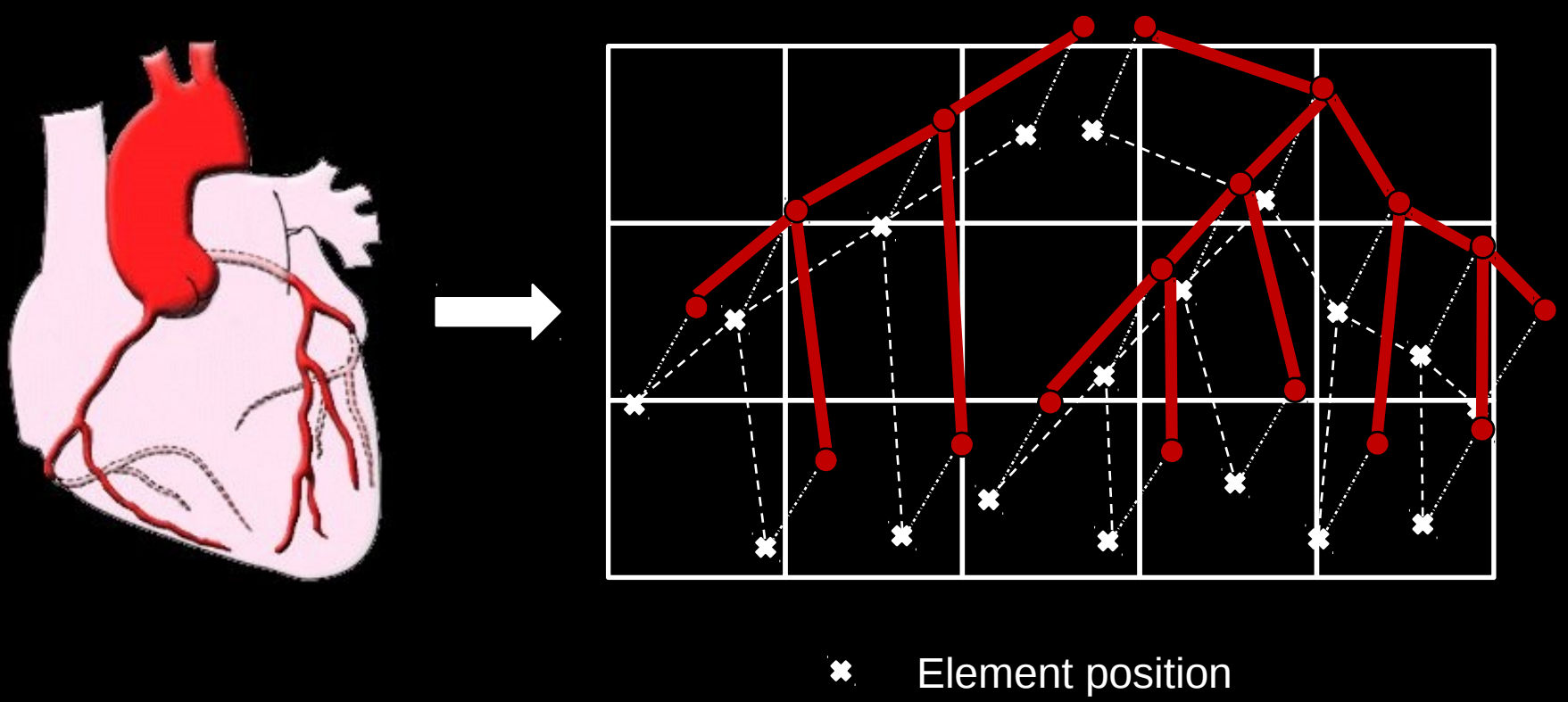
```
      IF(EQUATIONS_MATRICES%MATRICES(1)%UPDATE_MATRIX) THEN
        !Loop over element columns
        DO nh=1,FIELD_VARIABLE%NUMBER_OF_COMPONENTS
          DO ns=1,DEPENDENT_BASIS%NUMBER_OF_ELEMENT_PARAMETERS
            nhs=nhs+1
            DO ni=1,DEPENDENT_BASIS%NUMBER_OF_XI
              PGMSI(ni)=QUADRATURE_SCHEME%GAUSS_BASIS_FNS(ms, &
                    & PARTIAL_DERIVATIVE_FIRST_DERIVATIVE_MAP(ni),ng)
              PGNSI(ni)=QUADRATURE_SCHEME%GAUSS_BASIS_FNS(ns, &
                    & PARTIAL_DERIVATIVE_FIRST_DERIVATIVE_MAP(ni),ng)
            ENDDO !ni
            SUM=0.0_DP
            DO mi=1,DEPENDENT_BASIS%NUMBER_OF_XI
              DO ni=1,DEPENDENT_BASIS%NUMBER_OF_XI
                SUM=SUM+PGMSI(mi)*PGNSI(ni)*PROBLEM%SOLUTION%INTERPOLATION% &
                        & GEOMETRIC_INTERP_POINT_METRICS%GU(mi,ni)
              ENDDO !ni
            ENDDO !mi
            EQUATIONS_MATRICES%MATRICES(1)%ELEMENT_MATRIX%MATRIX(mhs,nhs)= &
                & EQUATIONS_MATRICES%MATRICES(1)%ELEMENT_MATRIX%MATRIX(mhs,nhs)+SUM*RWG
          ENDDO !ns
        ENDDO !nh
      ENDIF
      IF(EQUATIONS_MATRICES%UPDATE_VECTOR) THEN
        EQUATIONS_MATRICES%ELEMENT_VECTOR%VECTOR(mhs)=0.0_DP
      ENDIF
    ENDDO !ms
  ENDDO !mh
ENDDO !ng
```

Line Lengths Calculation

```
COORDINATE_SYSTEM=>FIELD%REGION%COORDINATE_SYSTEM
!Iterate to find the line lengths as line lengths depend on the scaling factors and vise versa.
CALL FIELD_INTERPOLATION_PARAMETERS_INITIALISE(FIELD,FIELD_STANDARD_VARIABLE_TYPE, &
    & INTERPOLATION_PARAMETERS,ERR,ERROR,*999)
CALL FIELD_INTERPOLATED_POINT_INITIALISE(INTERPOLATION_PARAMETERS,INTERPOLATED_POINT, &
        & ERR,ERROR,*999)
ITERATE=.TRUE.
ITERATION_NUMBER=0
LAST_MAXIMUM_LENGTH_DIFFERENCE=0.0_DP
DO WHILE(ITERATE.AND.ITERATION_NUMBER<=LINES_MAXIMUM_NUMBER_OF_ITERATIONS)
  MAXIMUM_LENGTH_DIFFERENCE=0.0_DP
  MAXIMUM_DIFFERENCE_LINE=1
  !Loop over the lines
  DO nl=1,FIELD%DECOMPOSITION%TOPOLOGY%LINES%NUMBER_OF_LINES
    CALL FIELD_INTERPOLATION_PARAMETERS_LINE_GET(FIELD_VALUES_SET_TYPE,nl, &
        & INTERPOLATION_PARAMETERS,ERR,ERROR,*999)
    OLD_LINE_LENGTH=FIELD%GEOMETRIC_FIELD_PARAMETERS%LENGTHS(nl)
    LINE_LENGTH=0.0_DP
    !Integrate || dr(xi)/dt || from xi=0 to 1 to determine the arc length.
    DO ng=1,NUMBER_OF_GAUSS_POINTS
      XI(1)=XIG(GAUSS_START(NUMBER_OF_GAUSS_POINTS)+ng)
      W=WIG(GAUSS_START(NUMBER_OF_GAUSS_POINTS)+ng)
      CALL FIELD_INTERPOLATE_XI(FIRST_PART_DERIV,XI,INTERPOLATED_POINT,ERR,ERROR,*999)
      CALL COORDINATE_DERIVATIVE_NORM(COORDINATE_SYSTEM,PART_DERIV_S1,INTERPOLATED_POINT, &
          & DERIV_NORM,ERR,ERROR,*999)
      LINE_LENGTH=LINE_LENGTH+W*DERIV_NORM
    ENDDO !ng
```

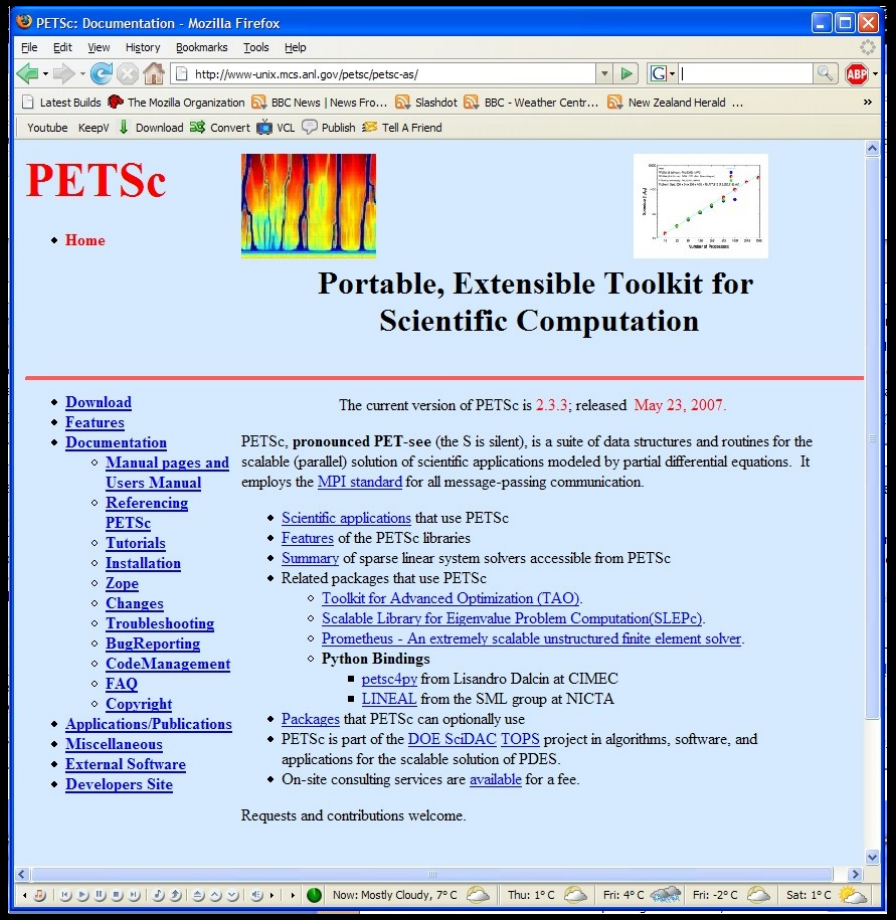
```
    FIELD%GEOMETRIC_FIELD_PARAMETERS%LENGTHS(nl)=LINE_LENGTH
    LENGTH_DIFFERENCE=ABS(LINE_LENGTH-OLD_LINE_LENGTH)/(1.0_DP+OLD_LINE_LENGTH)
    IF(LENGTH_DIFFERENCE>MAXIMUM_LENGTH_DIFFERENCE) THEN
      MAXIMUM_LENGTH_DIFFERENCE=LENGTH_DIFFERENCE
      MAXIMUM_DIFFERENCE_LINE=nl
    ENDIF
  ENDDO !nl
  ITERATE=MAXIMUM_LENGTH_DIFFERENCE>LINE_INCREMENT_TOLERANCE
  IF(ITERATE) THEN
    IF(ITERATION_NUMBER==1) THEN
      LAST_MAXIMUM_LENGTH_DIFFERENCE=MAXIMUM_LENGTH_DIFFERENCE
    ELSE IF(MAXIMUM_LENGTH_DIFFERENCE<LOOSE_TOLERANCE.AND. &
        & MAXIMUM_LENGTH_DIFFERENCE>=LAST_MAXIMUM_LENGTH_DIFFERENCE) THEN
      !Seems to be at a numerical limit
      ITERATE=.FALSE.
    ELSE
      LAST_MAXIMUM_LENGTH_DIFFERENCE=MAXIMUM_LENGTH_DIFFERENCE
    ENDIF
  ENDIF
  ITERATION_NUMBER=ITERATION_NUMBER+1
  IF(.NOT.ITERATE.OR.ITERATION_NUMBER==LINES_MAXIMUM_NUMBER_OF_ITERATIONS) THEN
    UPDATE_FIELDS_USING=.TRUE.
  ELSE
    UPDATE_FIELDS_USING=.FALSE.
  ENDIF
  CALL FIELD_GEOMETRIC_PARAMETERS_SCALE_FACTORS_UPDATE(FIELD,UPDATE_FIELDS_USING,ERR,ERROR,*999)
ENDDO !iterate
CALL FIELD_INTERPOLATED_POINT_FINALISE(INTERPOLATED_POINT,ERR,ERROR,*999)
CALL FIELD_INTERPOLATION_PARAMETERS_FINALISE(INTERPOLATION_PARAMETERS,ERR,ERROR,*999)
```

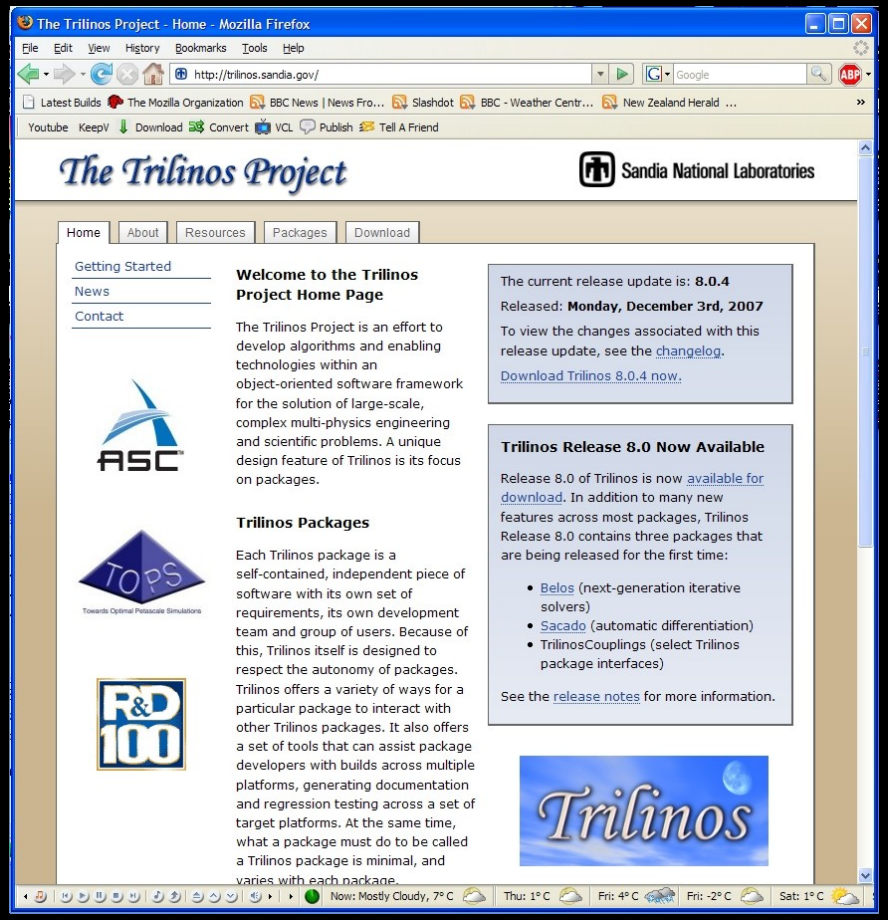
Embedded meshes



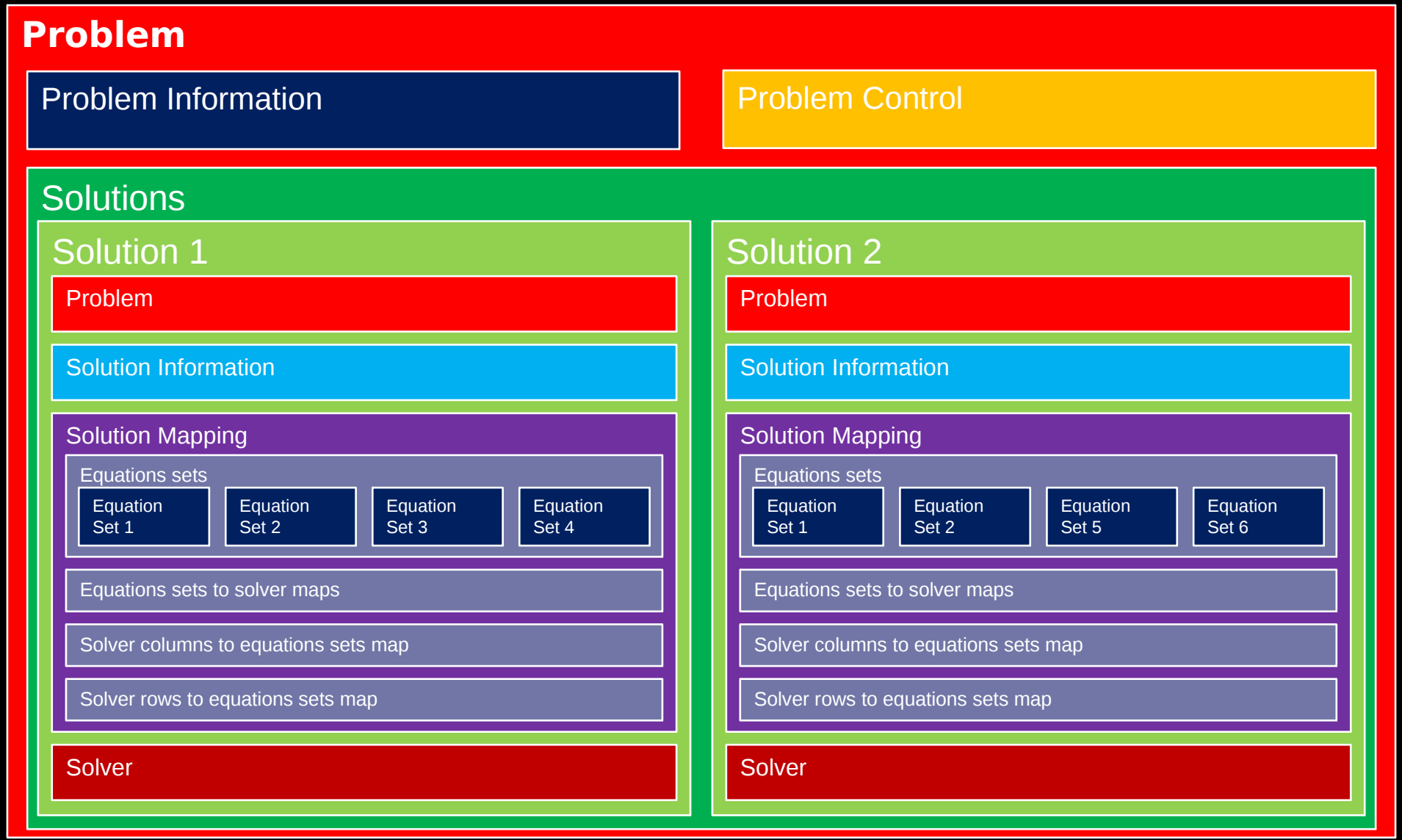
Element position

Distributed solvers





Problem


Problem
Problem Information
Problem Control
Solutions
Solution 1
Problem
Solution Information
Solution Mapping
Equations sets
Equation Set 1
Equation Set 2
Equation Set 3
Equation Set 4
Equations sets to solver maps
Solver columns to equations sets map
Solver rows to equations sets map
Solver
Solution 2
Problem
Solution Information
Solution Mapping
Equations sets
Equation Set 1
Equation Set 2
Equation Set 5
Equation Set 6
Equations sets to solver maps
Solver columns to equations sets map
Solver rows to equations sets map
Solver

Native History I/O

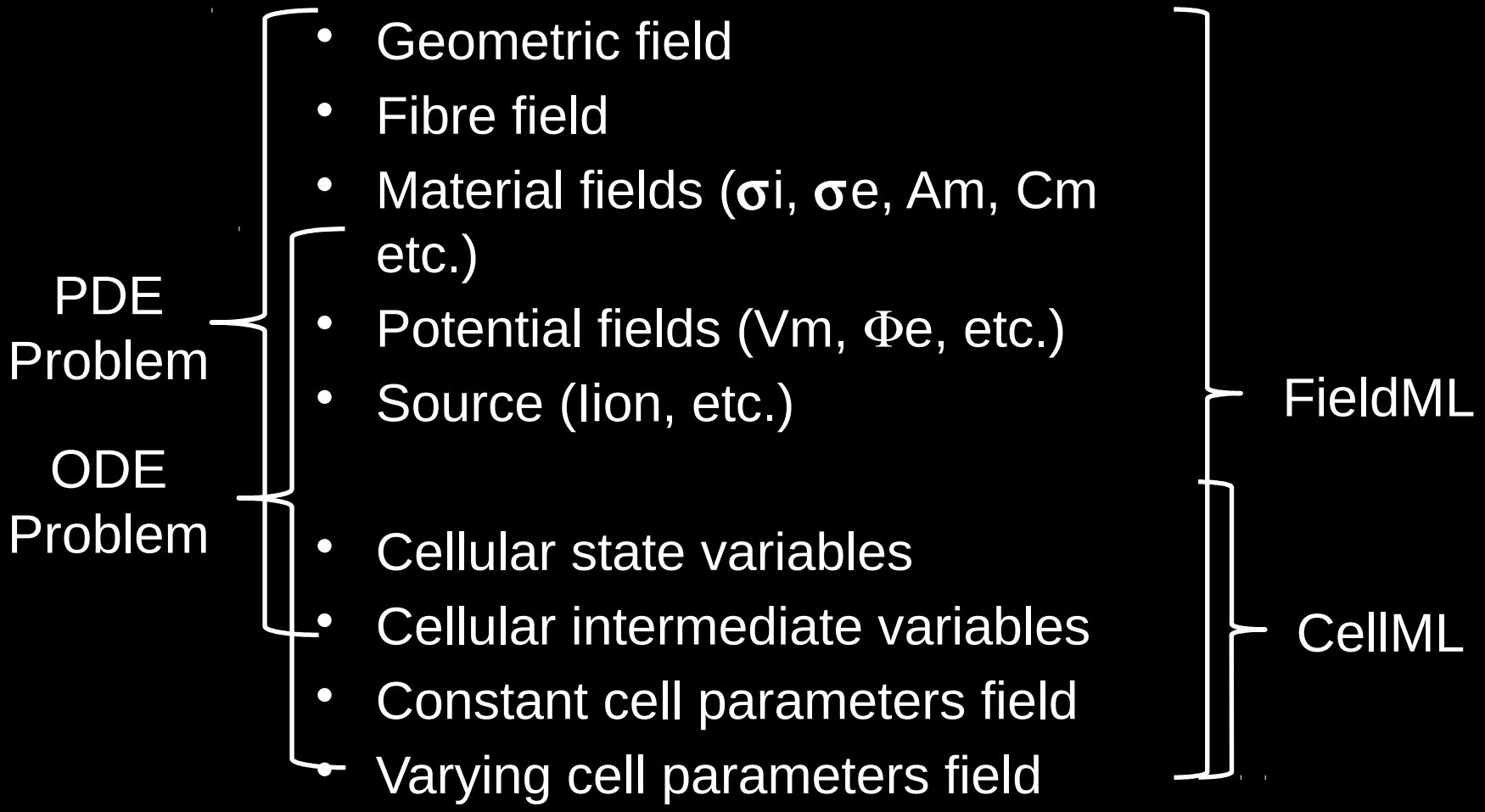
- Both FieldML and CellML files are primarily designed for information interchange. As such their format is not optimised for speed.
- During a solve it is often necessary to write out intermediate results or periodic results during a time stepping solution (history file).
- As the history I/O is inside a computational loop it is essential that the I/O is a) high performance and b) parallel otherwise overall speed and scalability will be severely affected.
- Parallel high performance history I/O being developed in collaboration with University of Oslo.

Example – Electrical Activation Problems

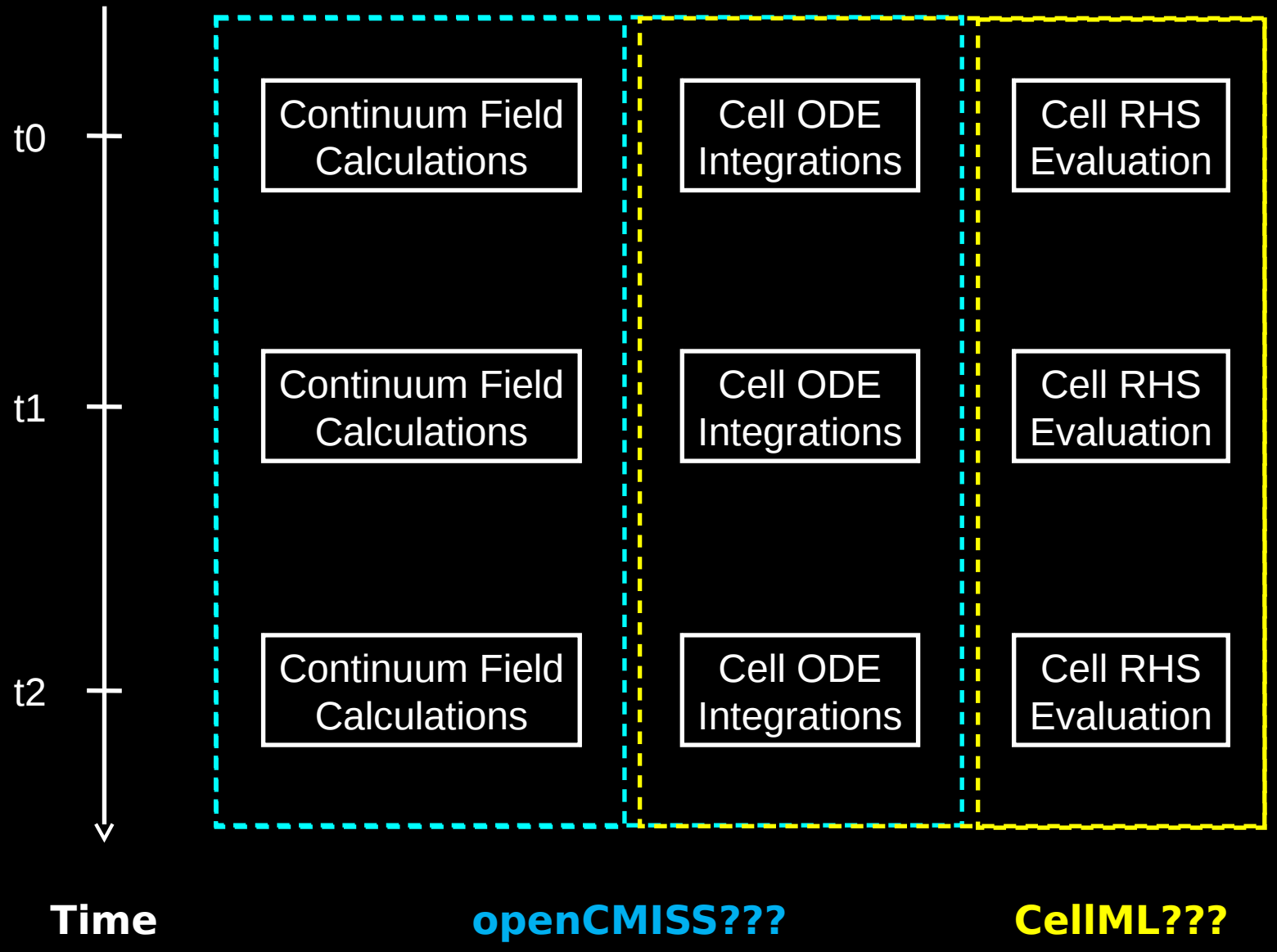
Cellular parameter field

- Want to be able to vary any cell parameter spatially
- Two options
  - Interpolate every parameter before evaluating a cell model.
    - Most general
    - Computationally expensive as most parameters are constant
    - Only allow certain parameters to vary and interpolate them

Fields Involved



Cell Model Evaluation Scope



Field variable component sub-objects

- Interpolation type
- Mesh component index
- Scaling index
- Domain pointer
- Mappings (Field parameters → DOF)

Nonlinear Poisson equation

Applying Green-Gauss theorem and rearranging gives

Adopting a Galerkin approach we set the weight function to be a basis function

Nonlinear Poisson equation

And interpolating the dependent variable using basis functions we have

The equation thus becomes is

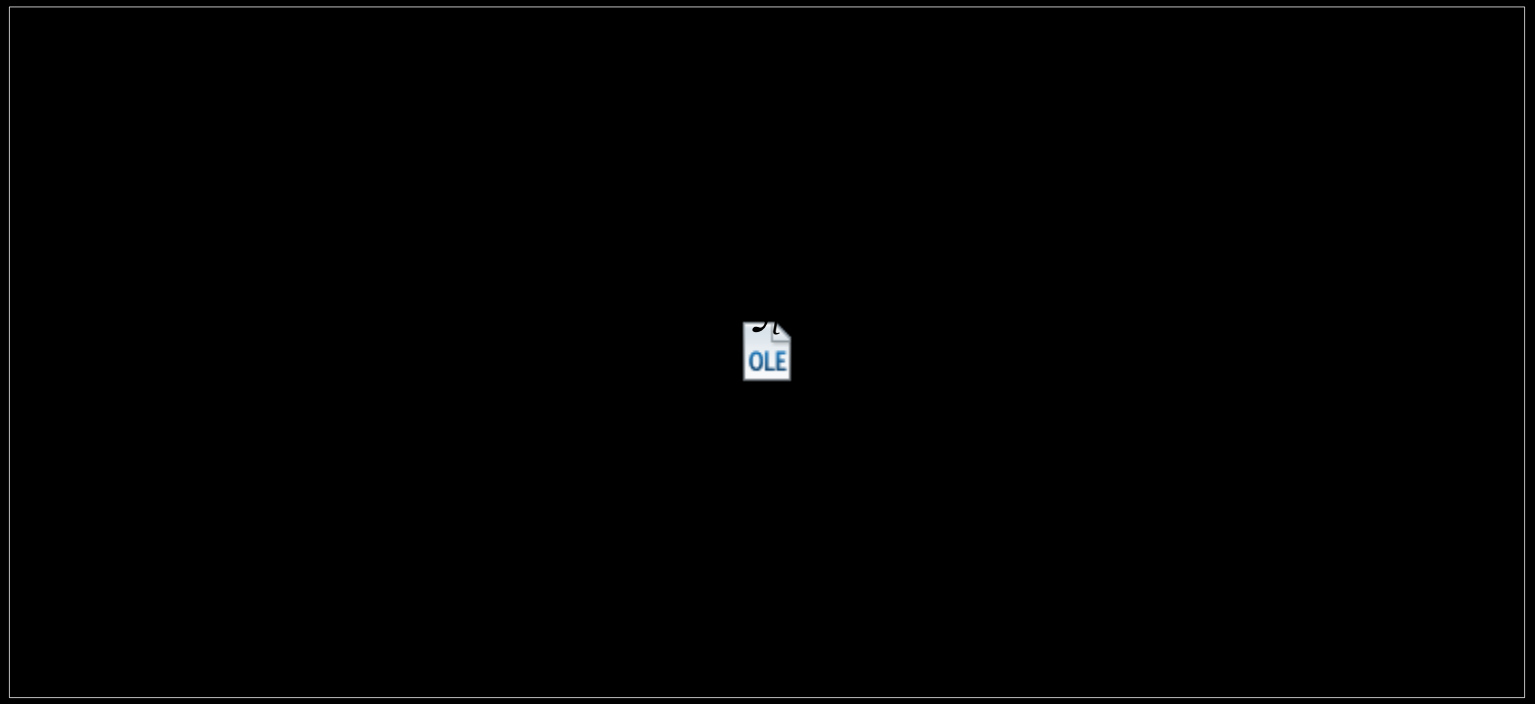
After discretising into elements we can obtain the set of nonlinear equations

or

Where **g** is a nonlinear vector function in terms of the vector of nodal potentials, ϕ, **K** is the stiffness matrix, **f** is the forcing vector and s is the source vector

Nonlinear Poisson equation

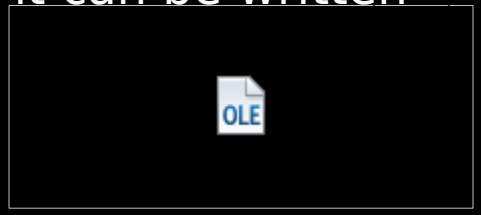
The individual components are given by



Nonlinear Poisson equation

In general a system of nonlinear equations takes the form of

The standard method for solving this equation is Newtons method. Newtons method is an iterative method for finding the root of equations. In basic form it can be written



Where **J** is the Jacobian of the function **F** i.e.,

Nonlinear Poisson equation

Comparing to the system of nonlinear Poisson equations we see

We can just use Newton’s method i.e.,

Where **J** is given by