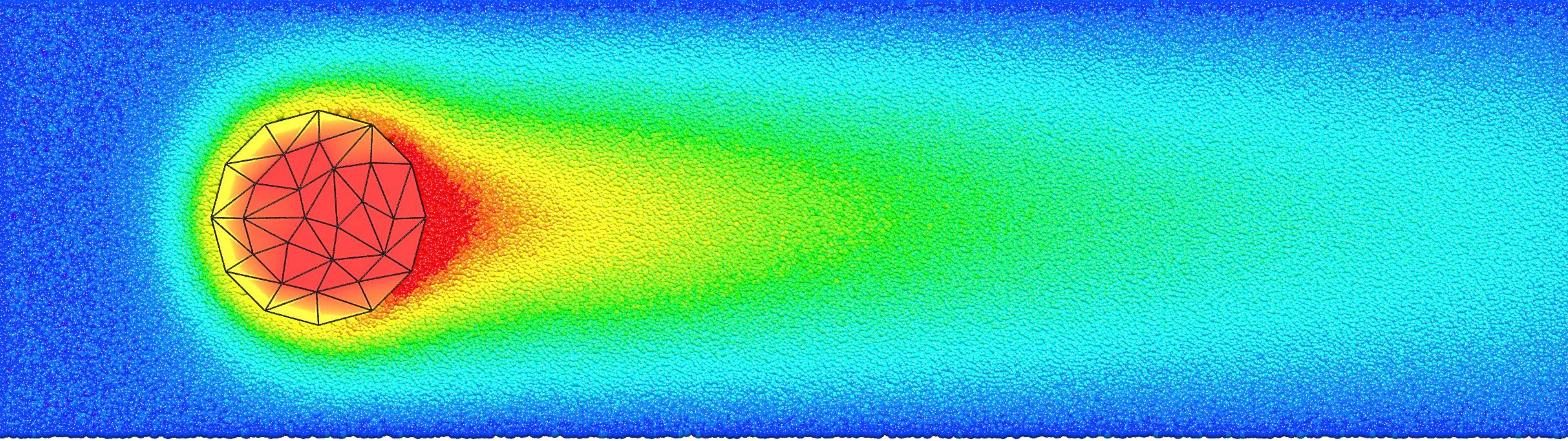


Multiscale Universal Interface

A Concurrent Framework for Coupling Heterogeneous Solvers

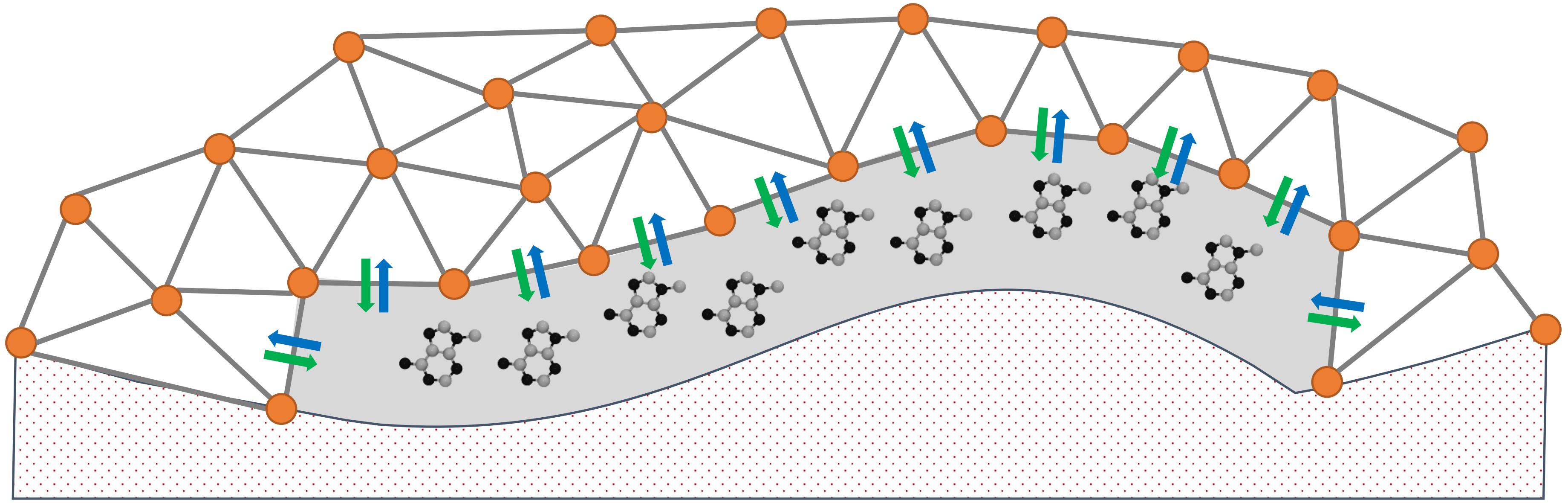
Yu-Hang Tang

Karniadakis Group, Division of Applied Math, Brown University

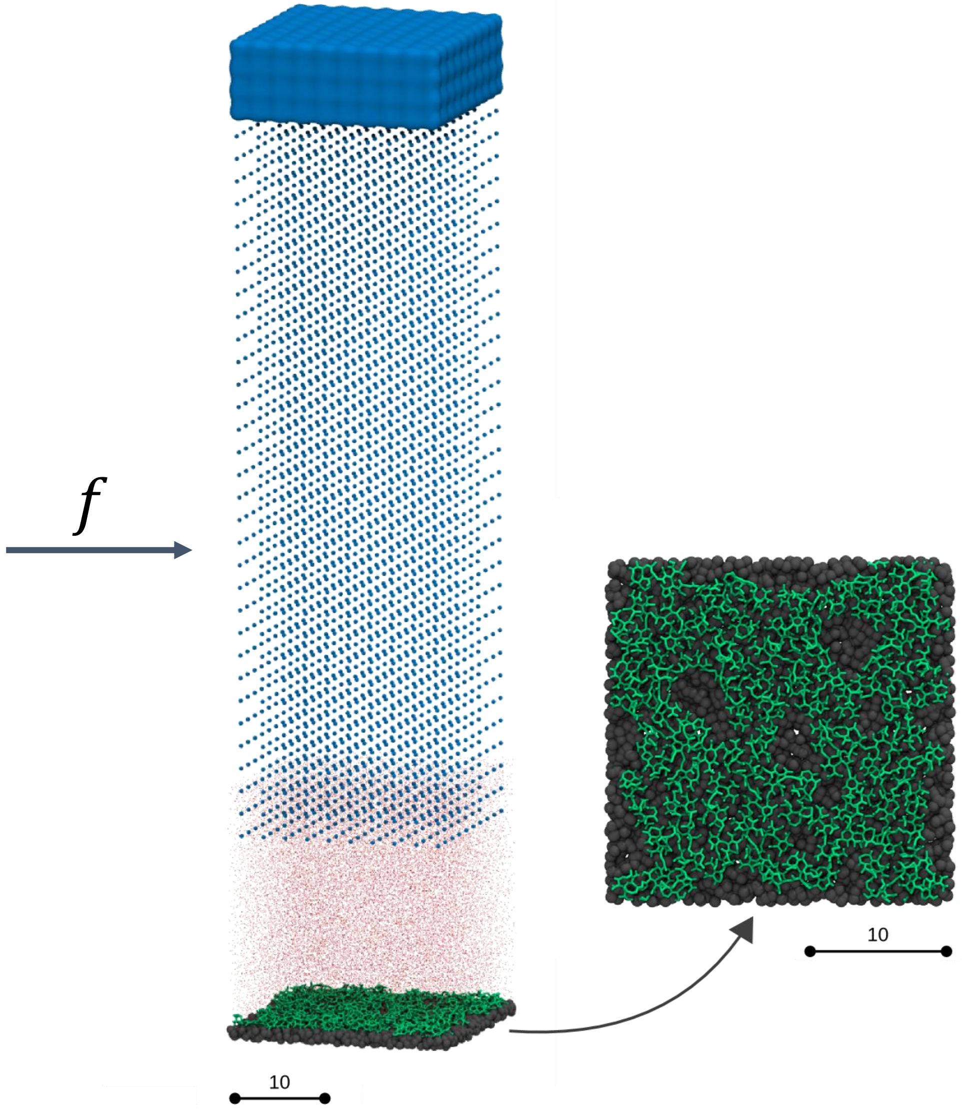
BROWN

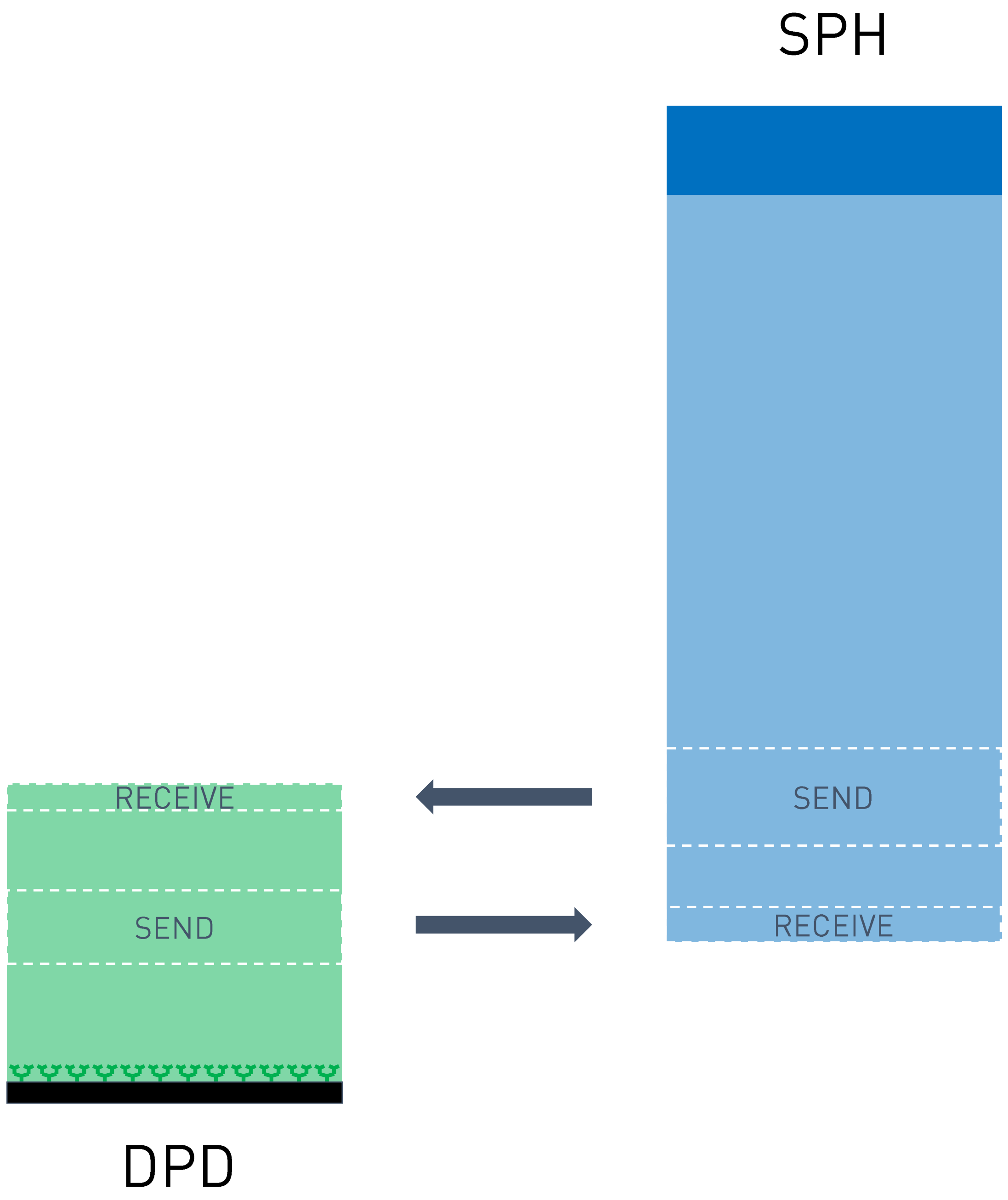
Multiscale Simulations by Domain Decomposition

- Each solver handles a subdomain and use the other as boundary

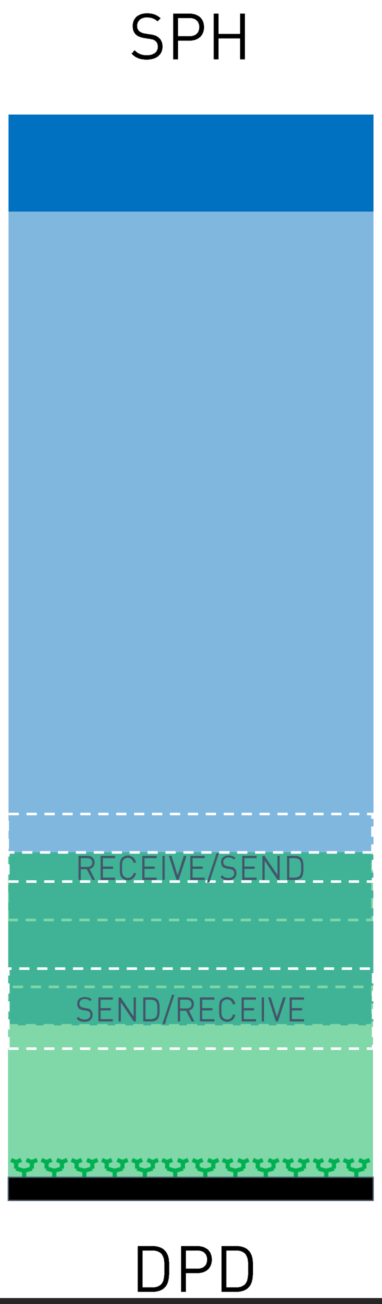


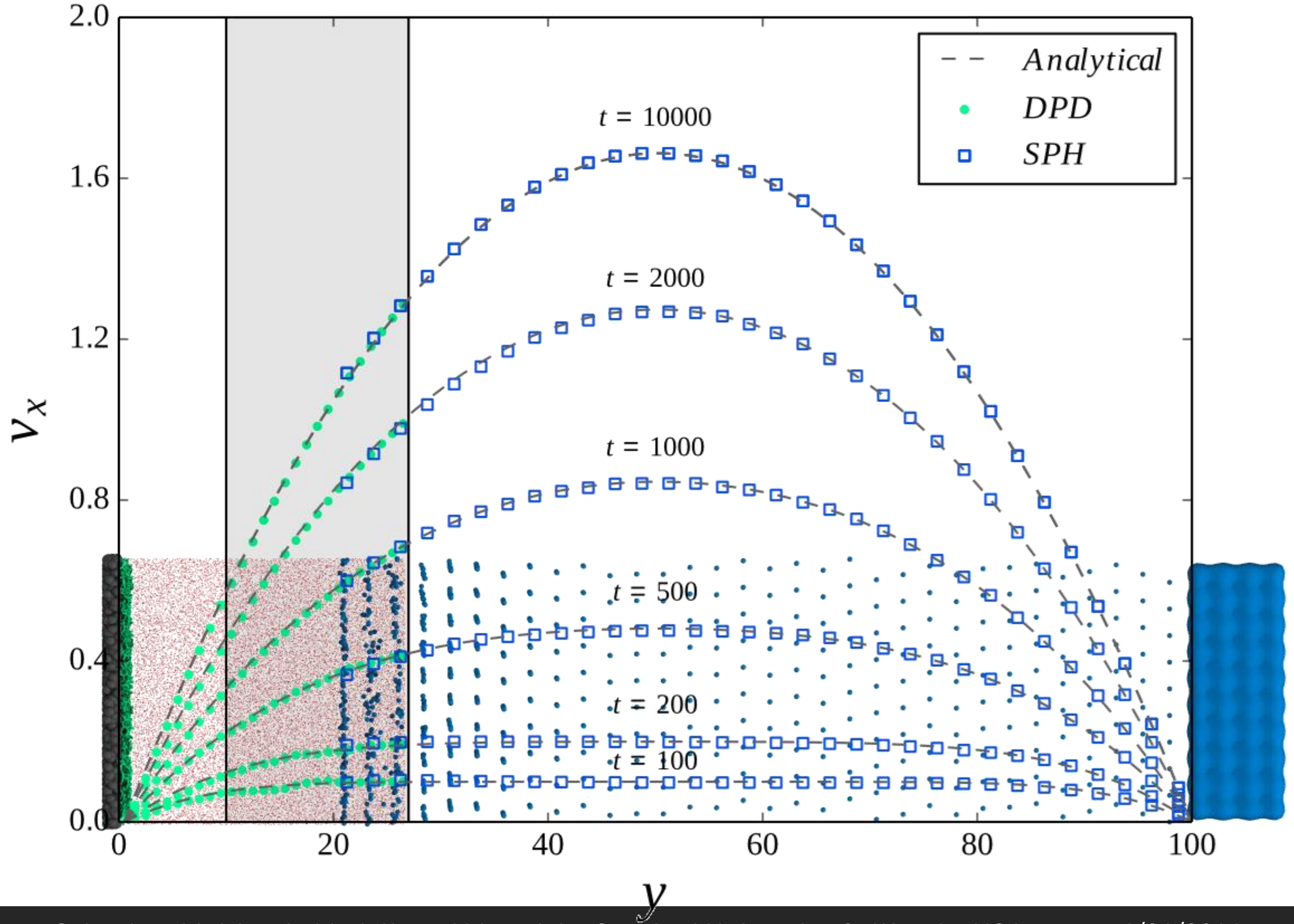
Example: Grafted Surface



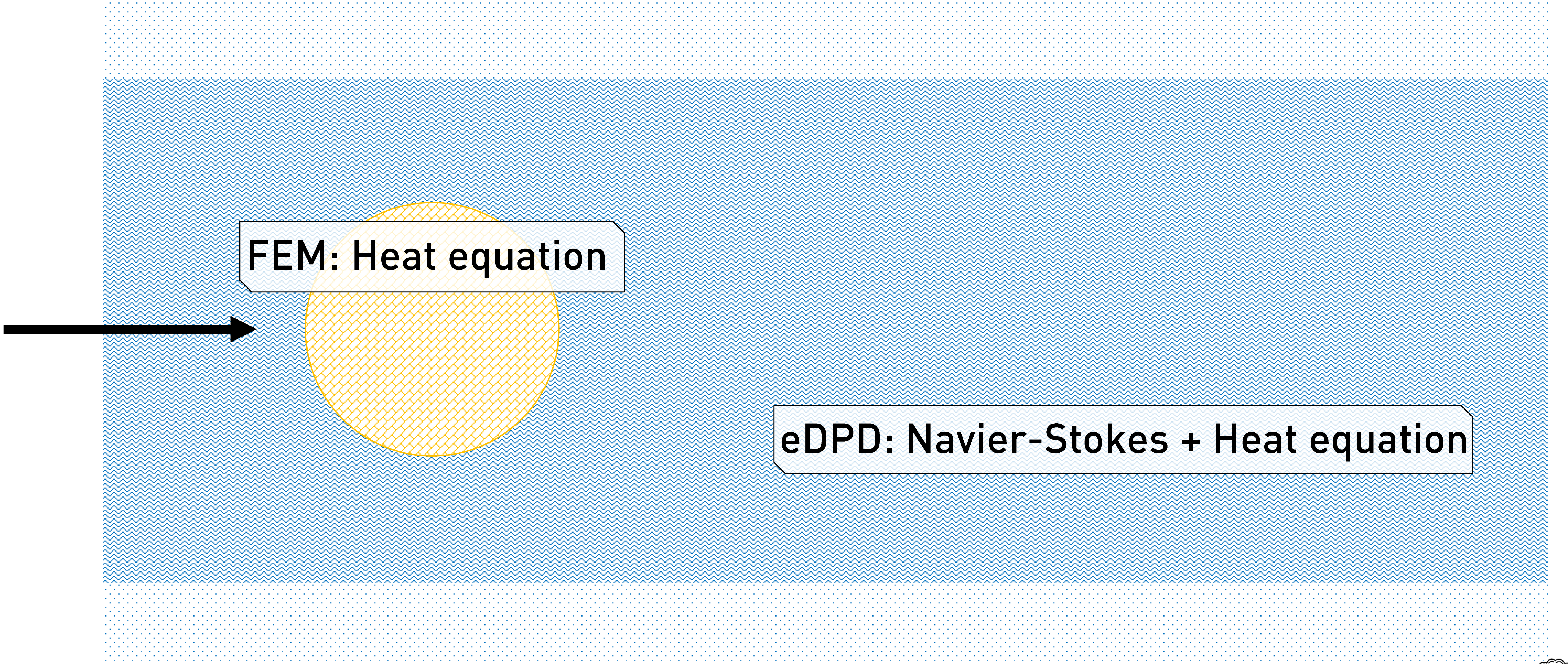


Example : Grafted Surface

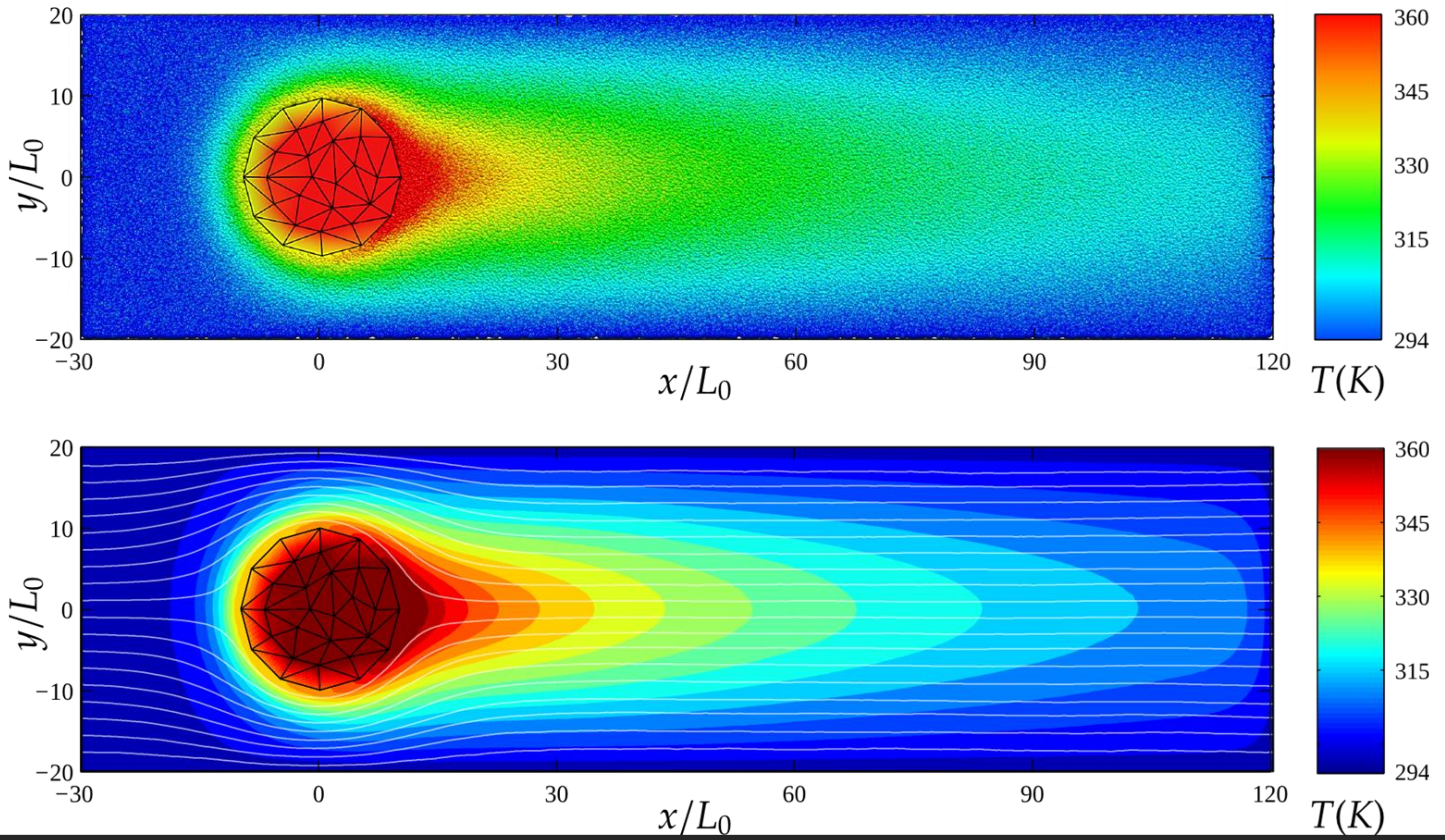




Example : Conjugate Heat Transfer

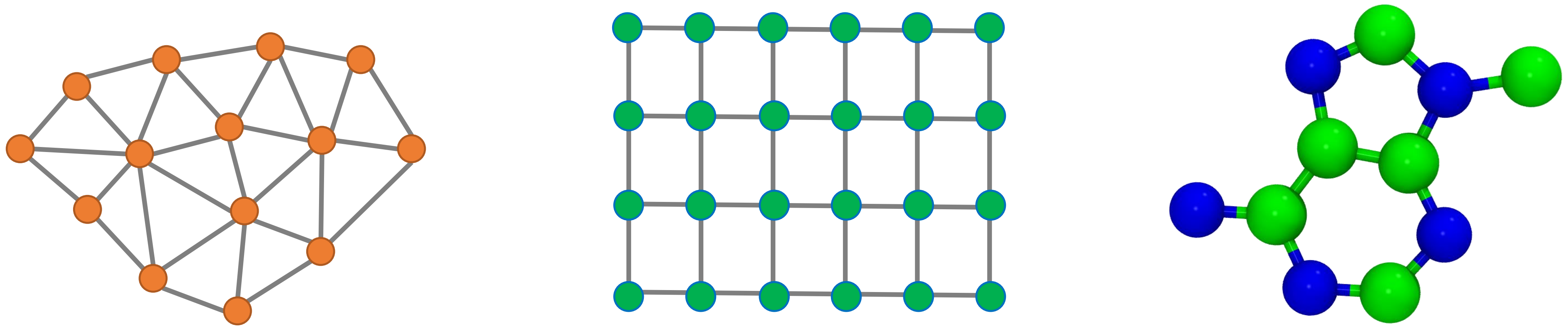


Example : Conjugate Heat Transfer

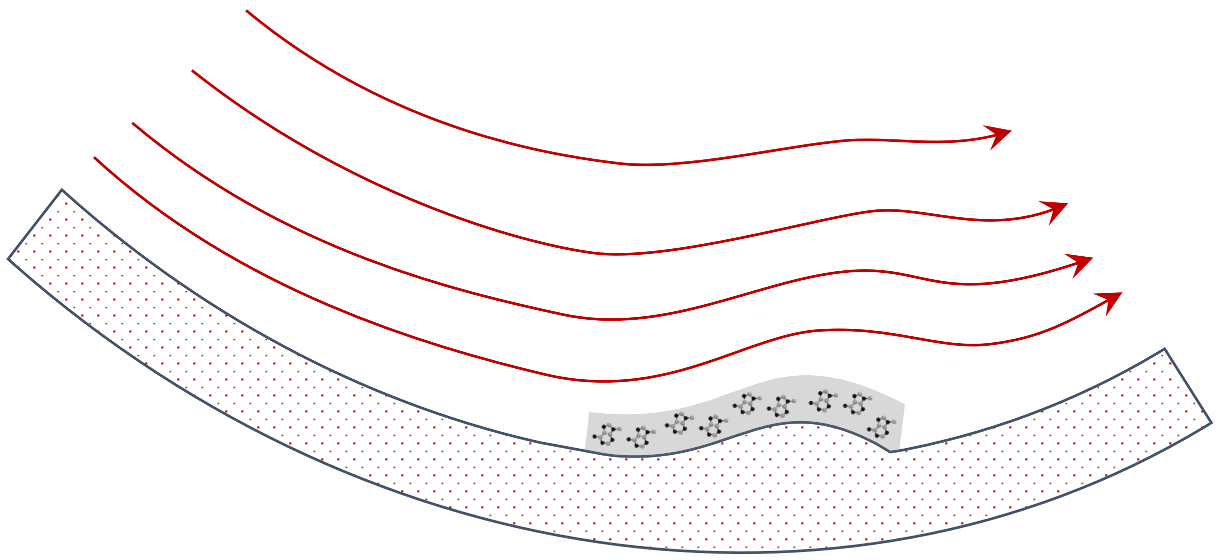


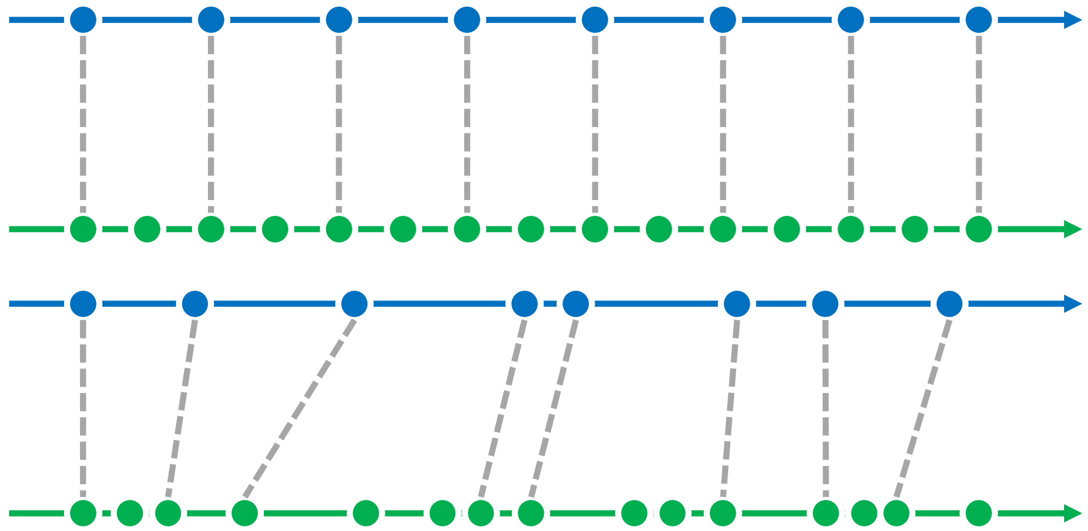
Diversity in Current Coupling - I

- Equation
  - Newton's
  - Schrödinger's
  - etc.



- Discretization / Geometry
- Time stepping: uniform, staggered, variable



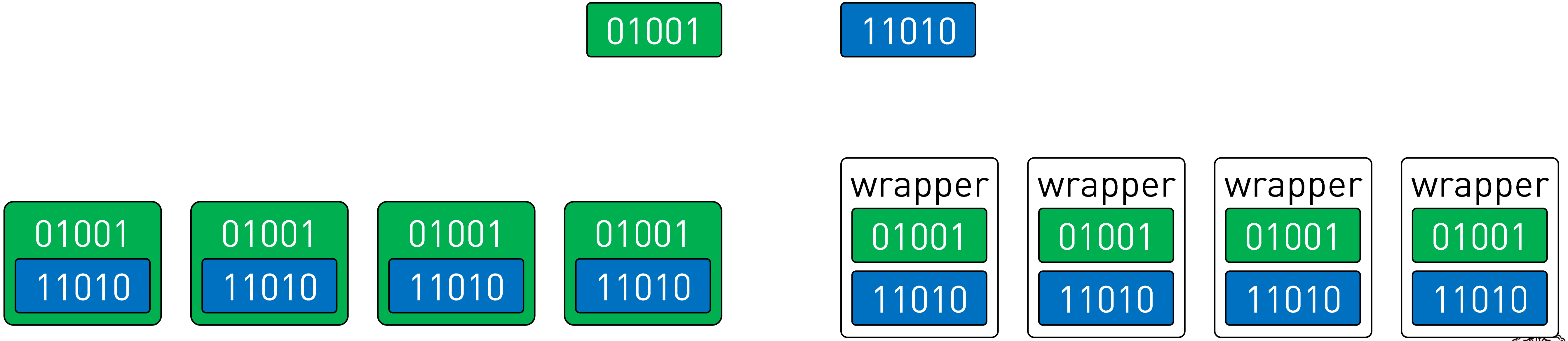


Diversity in Concurrent Coupling - II

- Solver: C, C++, Fortran, Python, …
- Scheme
- Parallelization: Serial, OpenMP, MPI, …

- The majority of existing code
  - was not developed to be coupled
  - need refactoring/invasive development

- Existing solutions largely rely on embedding or metacode



EMBEDDED METACODE

Multiscale Universal Interface (MUI) 01001 MUI

IS

A plug-and-play platform for testing ideas on multiscale coupling.

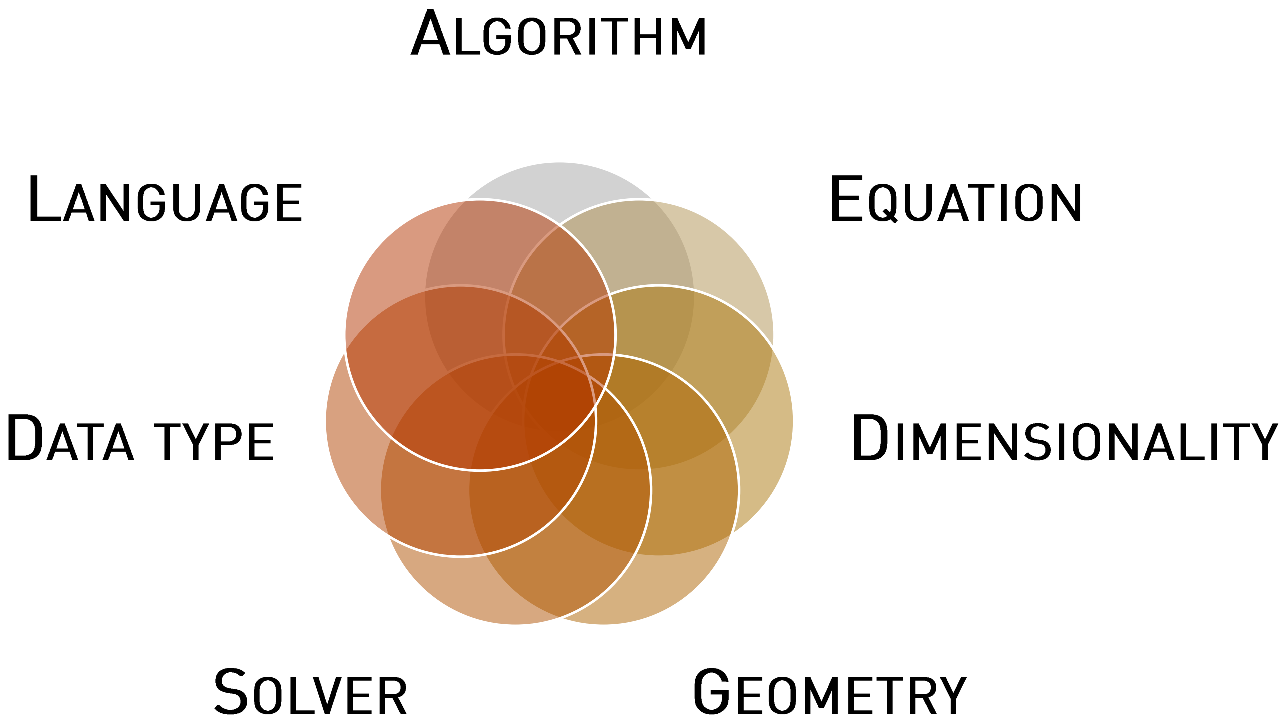
A communication layer for multi-solver information exchange.

A header-only C++ library that can be dropped into existing codes easily

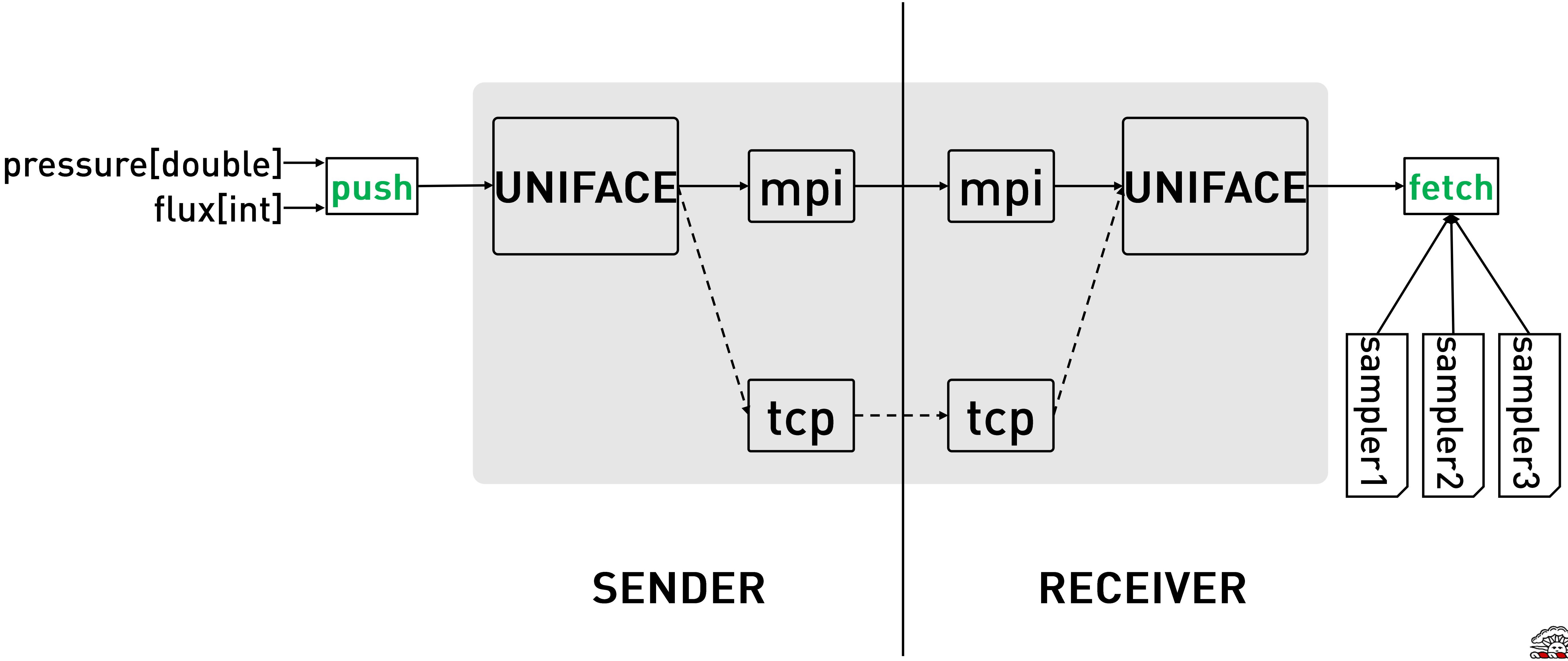
IS NOT

A specific coupling method that dictates which and how physical quantities get coupled.

A driver/wrapper that requires the exposure of certain programming interfaces from the solver.

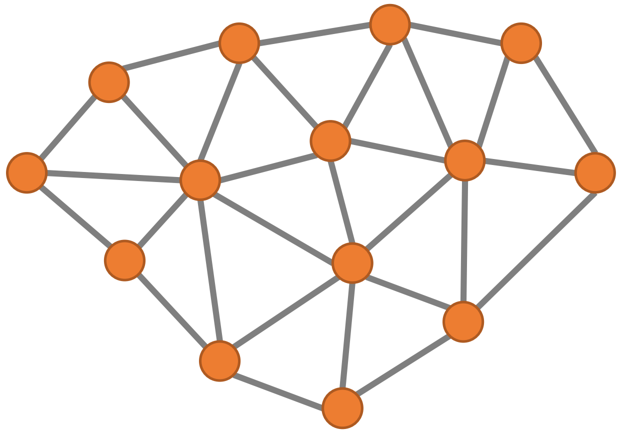


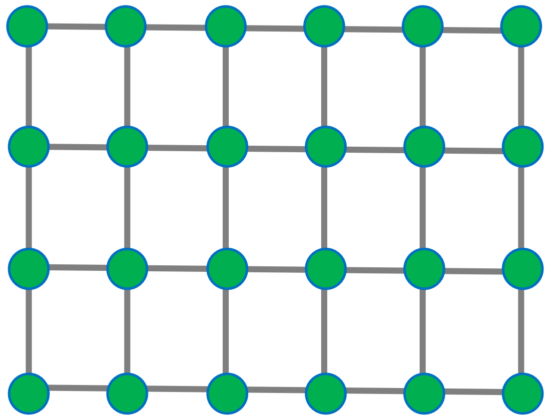
Workflow Overview

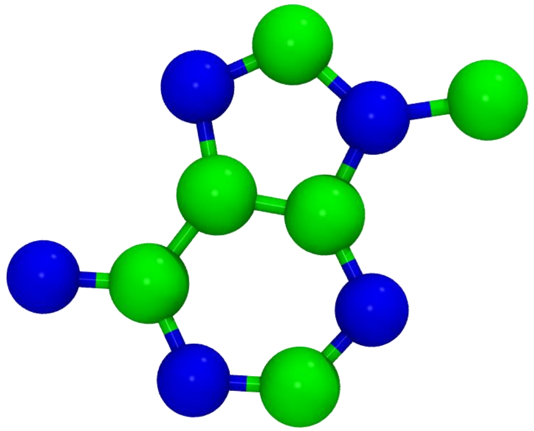


Abtraction: Data points

- Data point ≔ (location, type, value)



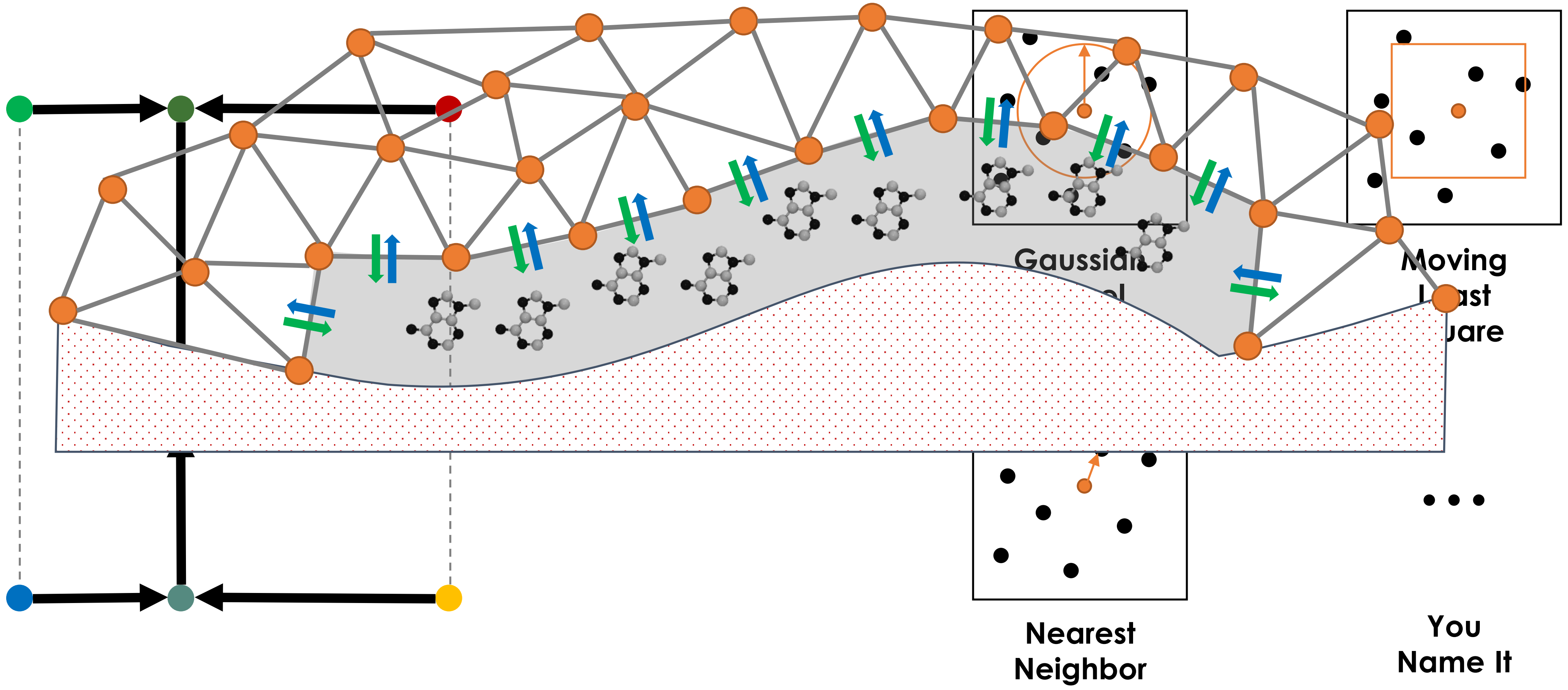




- Location: Vector Expression Templates
  - arbitrary dimension
  - real/complex coordinate
  - automatic SIMDization
- Value: arbitrary type
  - C++ templates
  - Type list metaprogramming

Abstraction: Sampling

- Texture Sampler
  - Hardware-implemented
  - Interpolate continuous color surface from discrete pixels
- MUI Data sampler
  - C++ functors
  - Can implement any interpolation

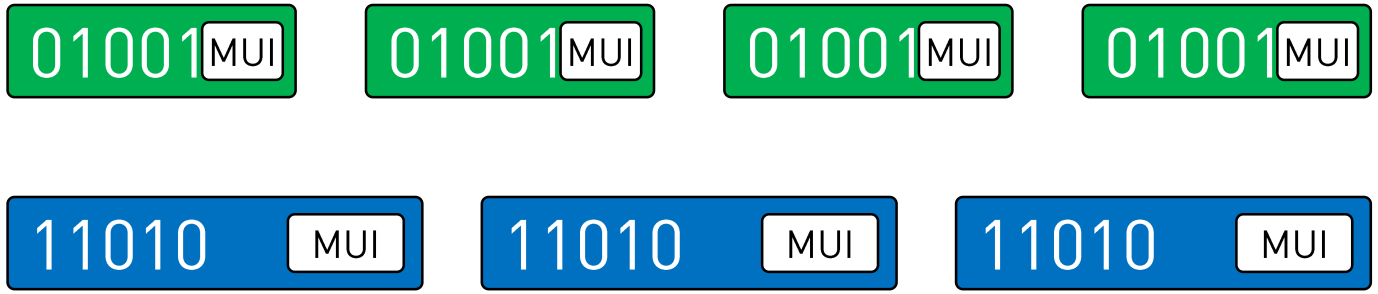


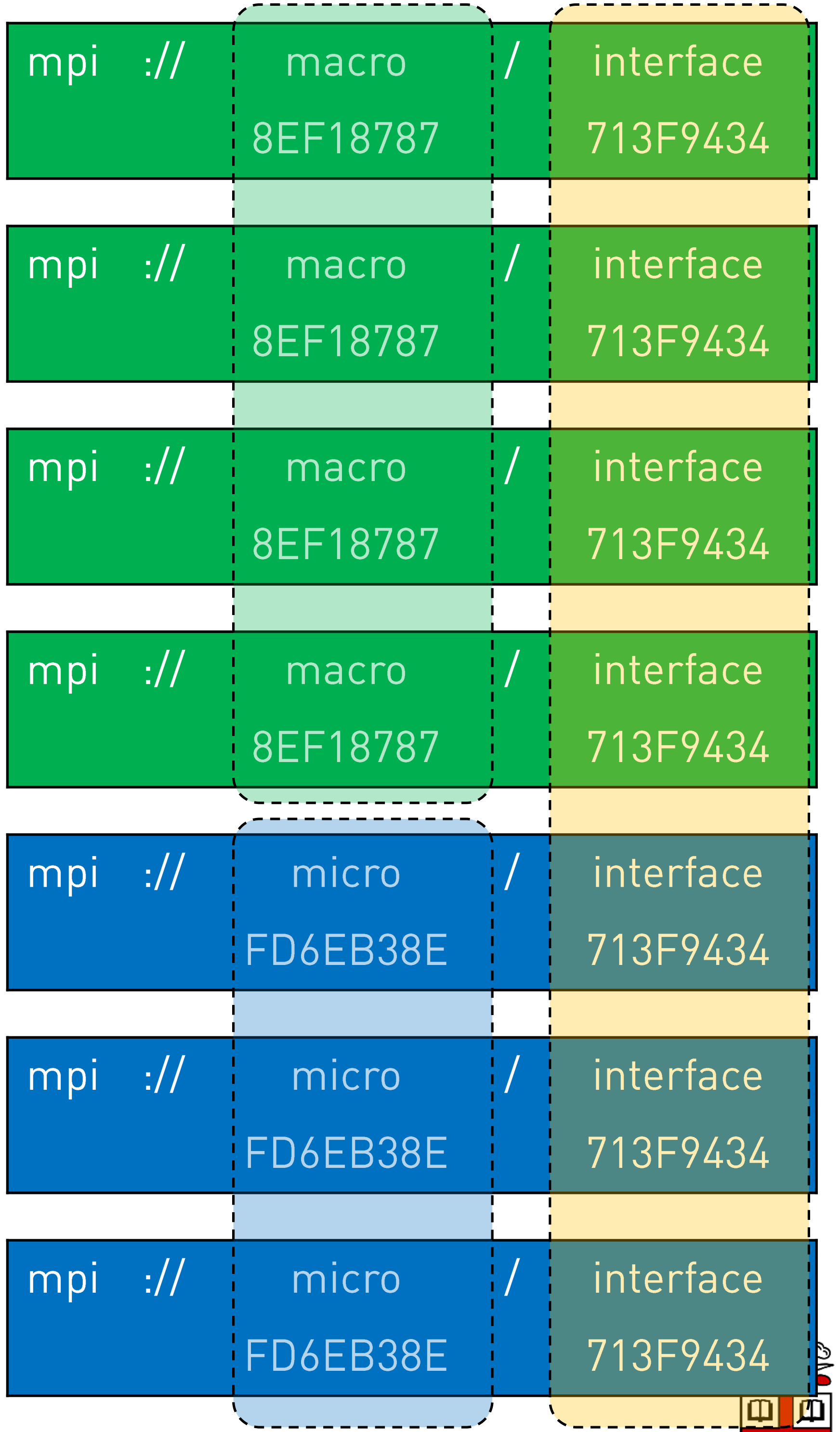
Sampler Design

```
template<typename O_TP, typename I_TP=O_TP, typename CONFIG=default_config>
class sampler_gauss {
public:
  using OTYPE      = O_TP;
  using ITYPE      = I_TP;
  using REAL       = typename CONFIG::REAL;
  using INT        = typename CONFIG::INT;
  using point_type = typename CONFIG::point_type;

  sampler_gauss( REAL r_, REAL h_ ) :
    r(r_), h(h_),
    nh(std::pow(2*PI*h,-0.5*CONFIG::D)) {}

  template<template<typename,typename> class CONTAINER>
  inline OTYPE filter( point_type focus,
                       const CONTAINER<ITYPE,CONFIG> &data_points ) const {
    REAL  wsum = 0;
    OTYPE vsum = 0;
    for(INT i = 0 ; i < data_points.size() ; i++) {
      auto d = (focus-data_points[i].first).normsq();
      if ( d < r*r ) {
        REAL w = nh * std::exp( (-0.5/h) * d );
        vsum += data_points[i].second * w;
        wsum += w;
      }
    }
    if ( wsum ) return vsum / wsum;
    else return 0.;
```

Parallelization: MPI MPMD

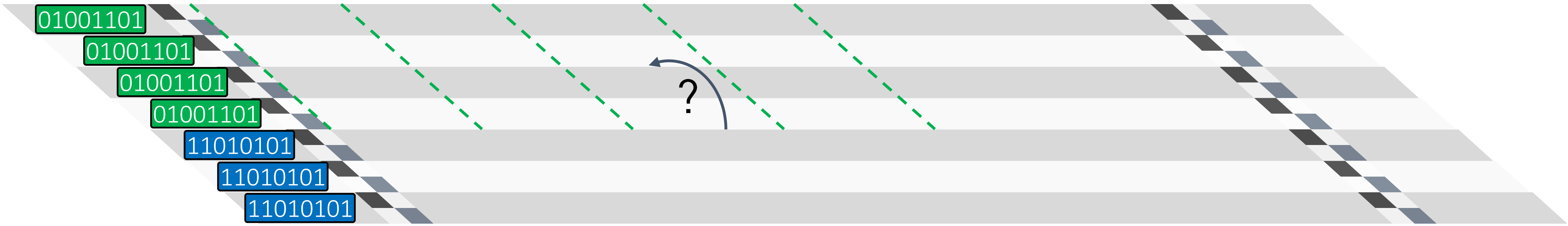
- Solvers compiled separately, runs concurrently
- MPMD syntax: **mpirun -np N1 solver1 : -np n2 solver2**
- URI: `protocol://domain/interface`
  - Use hash function to digitize the string
- Fetch method thread-safe





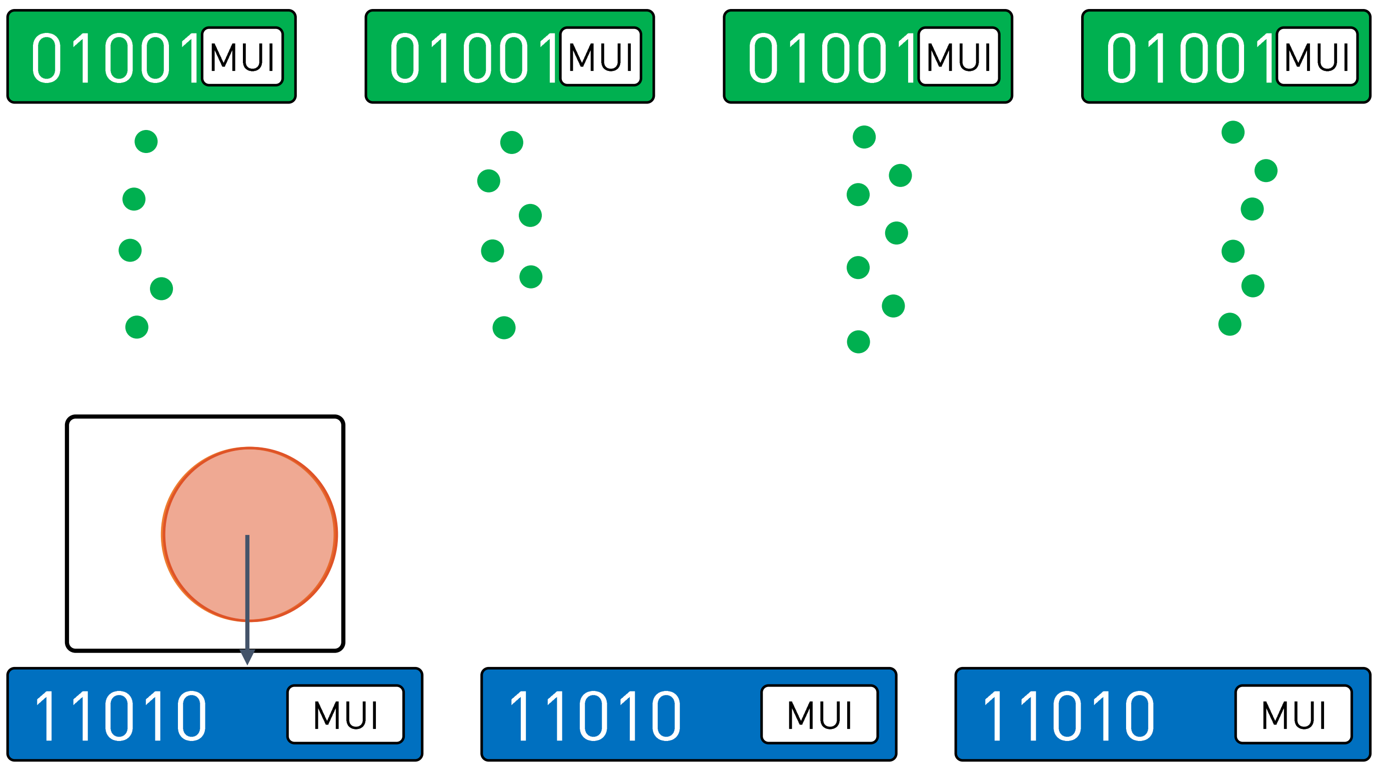
Time coherence

- MUI does not implicitly enforce solvers synchronization
  - In addition: time stepping may not align

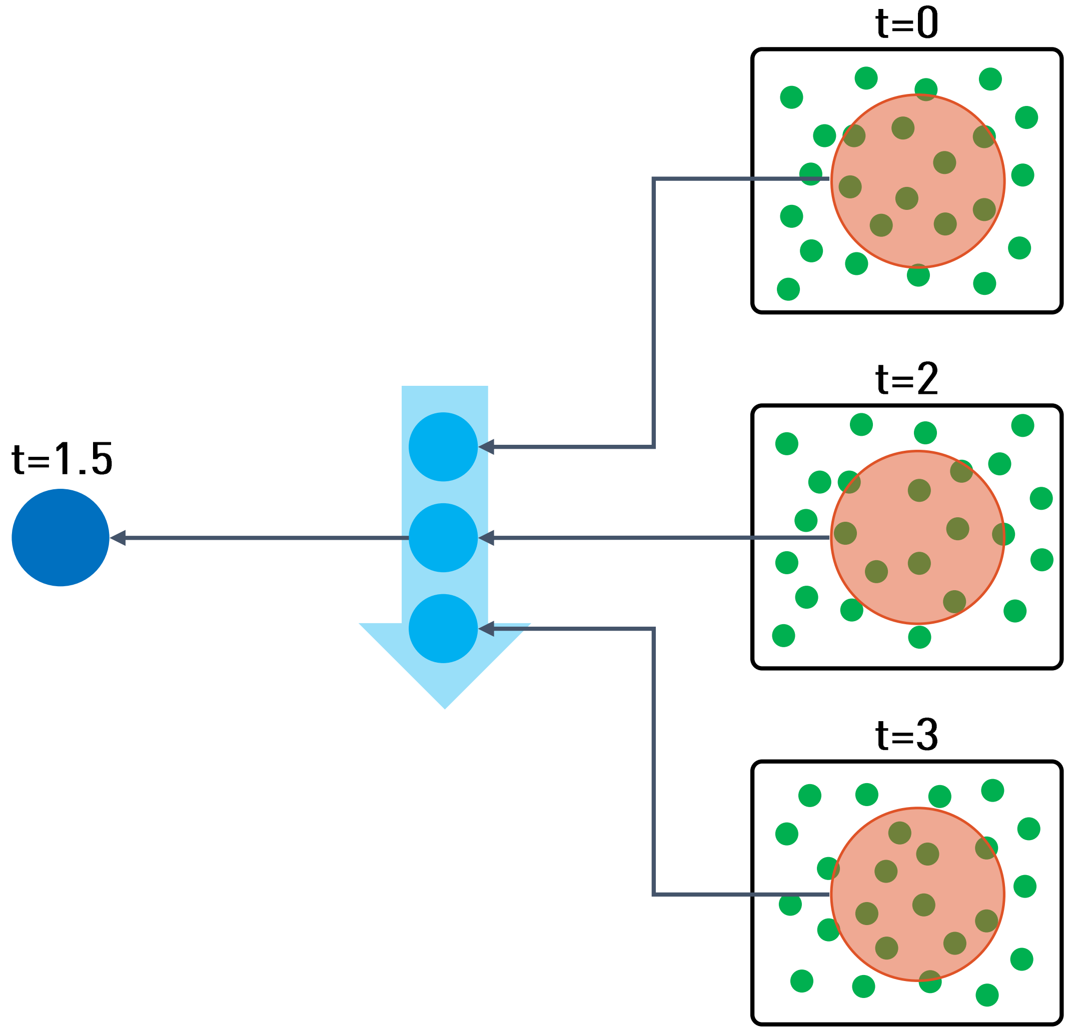


Time coherence

- Time frames
  - points of same timestamp merged as frames
  - tagged by timestamp
  - sampling performed on frames

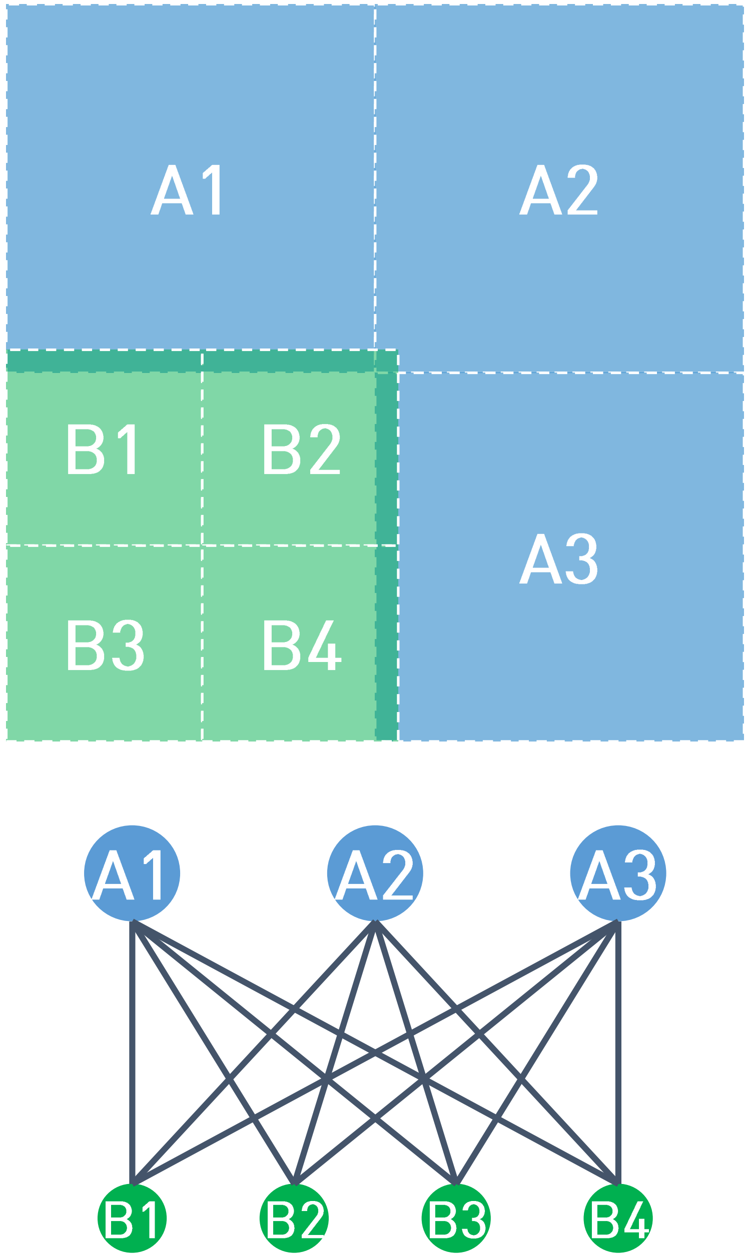


- Chrono sampler
  - Interpolate spatial results from time frames

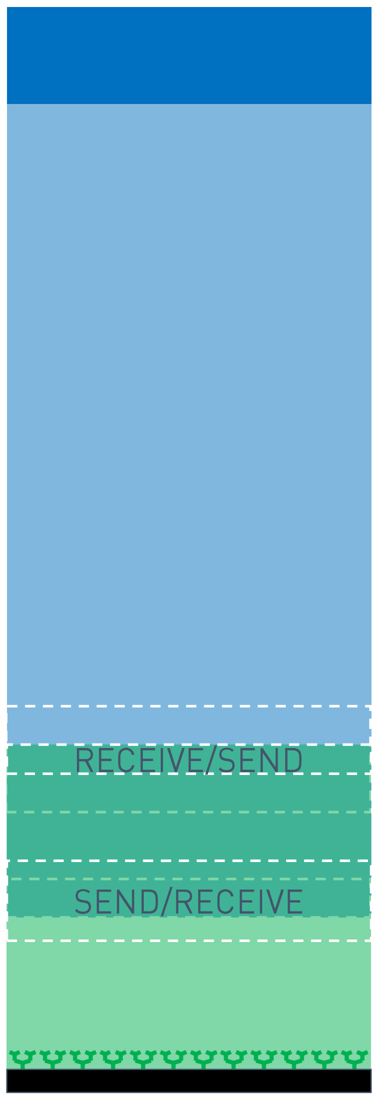


Selective Communication

- By default MUI broadcast all data points to all peer ranks
  - $O(N^2)$ messages!
- In many situations the interpolation algorithm is local
- Regions of interest
  - Hint for MUI to cut unnecessary messages
  - Arbitrary Boolean operations of boxes, spheres, and points
  - Use validity period for moving boundary



Example Revisited: Grafted Surface




```
/********** DPD **********/
For t = 0:dt:T
  For each particle i
    If WithinSendRegion(i)
      MUI::Push("vx", coord[i], velx[i])
  MUI::Commit(t)

  Force Eval, Integrate...

  tSPH = Floor(t,50dt)
  For each particle i
    If WithinReceiveRegion(i)
      Ss = Quintic(rSPH, hSPH)
      St = ExactTime
      vx[i] = MUI::Fetch
              ("vx", coord[i], tSPH, Ss, St)
  If t % 50dt = 0
    MUI::Forget(t-50dt)
```

```
/********** SPH **********/
For t = 0:50dt:T
  For each particle i
    If WithinSendRegion(i)
      MUI::Push("vx", coord[i], velx[i])
  MUI::Commit(t)

  Force Eval, Integrate...

  For each particle i
    If WithinReceiveRegion(i)
      Ss = Quintic(rDPD, hDPD)
      St = AverageOver(50dt)
      vx[i] = MUI::Fetch
              ("vx", coord[i], t, Ss, St)

  MUI::Forget(t)
```

Warming Up: Common C++/C++11 Techniques in MUI

Templates

Template programming is the mechanism in C++ for writing code that is independent of any particular type.

Function template

```
template<class T> swap( T &a, T &b ) {
    T tmp = a; a = b; b = tmp;
}
```

Class template

```
template<class T> struct adder {
    T sum = 0;
    T accumulate( T v ) {
        return sum +=  v;
    }

};
```

The using keyword

The `using` keyword can either by used as an enhanced version of `typedef`, or to bring classes/functions from another namespace into the current one.

```
// brings in iostream etc.
using namespace std;

// create type alias
using real = double;

// create class alias
using real_adder = adder<real>;

// create templated alias
template<class T, int N> struct fft;
template<class T> using fft_48pts =
fft<T,48>;
```

Functors

A functor is basically a class which defines the `operator ()`. It can instantiate objects which **look like** functions, while in addition can store states.

```
struct RNG {
    int state = 0x0;
    double operator () (bool peek =
false) {
        if (peek) return state;
        else return peek = peek *
0x1234 + 0x4321;
    }
};


auto log3 = [&] (double x) {
    return log(x) / log(3.0);
};
double a = log3( 9.0 ); // a = 2
```

Hands-on Practice: Environment Setup

Pre-configured Amazon EC2 Server

ssh -CX user@ytang.dyndns.org OR 52.53.243.39

user = summer00, summer01, ..., summer47 **pick your lucky number!**

initial password: cm4summerschool **change it upon login to claim your account!**

Once logged in: `cp -r /opt/hands_on ~`

Alternatively: use any system with a C++11 compiler and an MPI implementation and download example via SCP:

```
scp -r summer@ytang.dyndns.org:/opt/hands_on local_path
```

Hands-on Practice: MUI Library Structure

LICENSE-Apache-v2
LICENSE-GPL-v3
wrapper_c
wrapper_f
sampler_exact.h
sampler_gauss.h
sampler.h
sampler_mov_avg.h
sampler_nn.h
sampler_null.h
sampler_pseudo_n2_linear.h
sampler_pseudo_nn.h
sampler_shepard_quintic.h
sampler_sph_quintic.h
sampler_sum_quintic.h
chrono_sampler_exact.h

chrono_sampler_gauss.h
chrono_sampler_mean.h
chrono_sampler_null.h
chrono_sampler_sum.h
comm_factory.h
comm.h
comm_mpi.h
comm_mpi_nxn.h
comm_mpi_smart.h
comm_tcp.h
config.h
dim.h
dynstorage.h
exception.h
geometry.h
lib_dispatcher.h

lib_factory.h
lib_mpi_hepler.h
lib_mpi_multidomain.h
lib_mpi_split.h
lib_singleton.h
lib_uri.h
message.h
mui.h
point.h
reader.h
reader_variable.h
README.md
span.h
spatial_storage.h
stream.h

stream_ordered.h
stream_string.h
stream_tuple.h
stream_unordered.h
stream_vector.h
uniface.h
util.h
bin.h
virtual_container.h

Hands-on Practice: Hello World

```
#include "../mui/mui.h"                                   // Include the MUI header file
int main( int argc, char ** argv ) {
    using namespace mui;                                  // Bring in the MUI classes and functions
    uniface1d interface( argv[1] );                       // Instantiate a interface object which implicitly initialize MPI
    printf( "domain %s pushed value %s\n", argv[1], argv[2] );
    interface.push( "data", 0, atof( argv[2] ) );         // Push a single data point at coordinate 0
    interface.commit( 0 );                                // Commit the data points as frame 0
    double v = interface.fetch( "data", 0, 0,             // Fetch the data point from coordinate 0 and time frame 0
                                sampler_exact1d<double>(),// as provided by the other 'solver'
                                chrono_sampler_exact1d() );
    printf( "domain %s fetched value %lf\n", argv[1], v );
    return 0;                                             // Note how MUI implicitly handles MPI finalization
}
```

mpic++ -std=c++11 hello.cpp –o hello

mpirun –np 1 ./hello mpi://solver1/ifs 0.618 : -np 1 ./hello mpi://solver2/ifs 1.414

MUI Key Classes & Methods

mui::uniface[1d|2d|3d]

The MUI controller class that manages all coupling activities

mui::uniface::push(name, position, value)

Push data into the interface buffer

mui::uniface::commit(timestamp)

Send all data points in buffer as a time frame

mui::uniface::fetch(name, position, time, sampler_spatial, sampler_temporal)

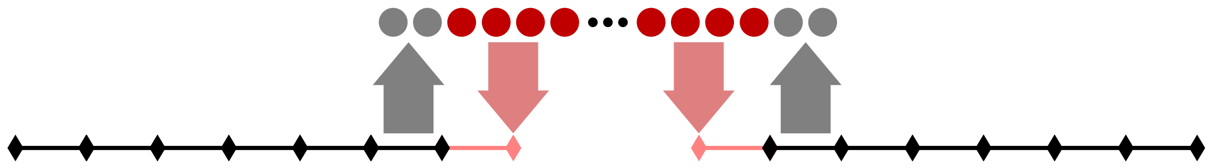
Interpolate data from a timeframe / across several time frames

sampler_[gauss|exact|moving_avg|...][1d|2d|3d]

chrono_sampler_[gauss|exact|mean|...]

The MUI samplers

Hands-on Practice: 1D FDM-SPH Coupling



Finite Difference

```
for k steps
| push 'send' grid points
| commit
| fetch 'receive' grid points
| for all i:
| | u_i = u_i + k / h^2 * Du
| end
end
```

Smoothed Particle Hydrodynamics

```
for k steps
| push 'send' particles
| commit
| fetch 'receive' particles
| for all i
| | for all j
| | | flux_ij = ...
| | end
| end
end
```

Hands-on Practice: 1D FDM-FDM Coupling

Fine

```
for k steps
| push 4 points near the boundary
| commit
| fetch 1 point from the coarse solver
| for all i:
| | u_i = u_i + k / h^2 * Du
| end
end
```

Coarse

```
for k steps
| push 1 point near the boundary
| commit
| interpolate 1 point from the fine
solver using a Gaussian kernel
| for all i:
| | u_i = u_i + k / h^2 * Du
| end
end
```

Thank you!

Acknowledgement

YHT acknowledges partial financial support from the 2015-2016 IBM Ph.D. Scholarship.

Research supported by the Department of Energy Collaboratory on Mathematics for Mesoscopic Modeling of Materials (CM4).

Reference

Tang, Kudo, Bian, Li & Karniadakis. *Journal of Computational Physics*, 2015