PyZoltan: Wrapping Zoltan with Cython

Kunal Puri

Department of Aerospace Engineering,
IIT Bombay

kunalp@iitb.ac.in

December 29, 2012

Outline

The Zoltan Library

- Sandia National Laboratories
- Part of the Trilinos Project (9.0 September 2008)
- Zoltan v3.6 released in September 2011

What can it do?

- Dynamic Load Balancing
- Graph Coloring
- Distributed data directories
- Dynamic memory management

What can it do?

- Dynamic Load Balancing
- Graph Coloring
- Distributed data directories
- Dynamic memory management

Load balancing

Used in ...

- Adaptive simulations (ALE)
- Lagrangian (moving) mesh simulations (FEM, FVM)
- Particle methods (SPH, DSMC, MD)

Used for ...

- Distributing *objects* across processors
- nodes, elements, points ...
- Equalizing work load
- Reducing communication time

Load balancing

Used in ...

- Adaptive simulations (ALE)
- Lagrangian (moving) mesh simulations (FEM, FVM)
- Particle methods (SPH, DSMC, MD)

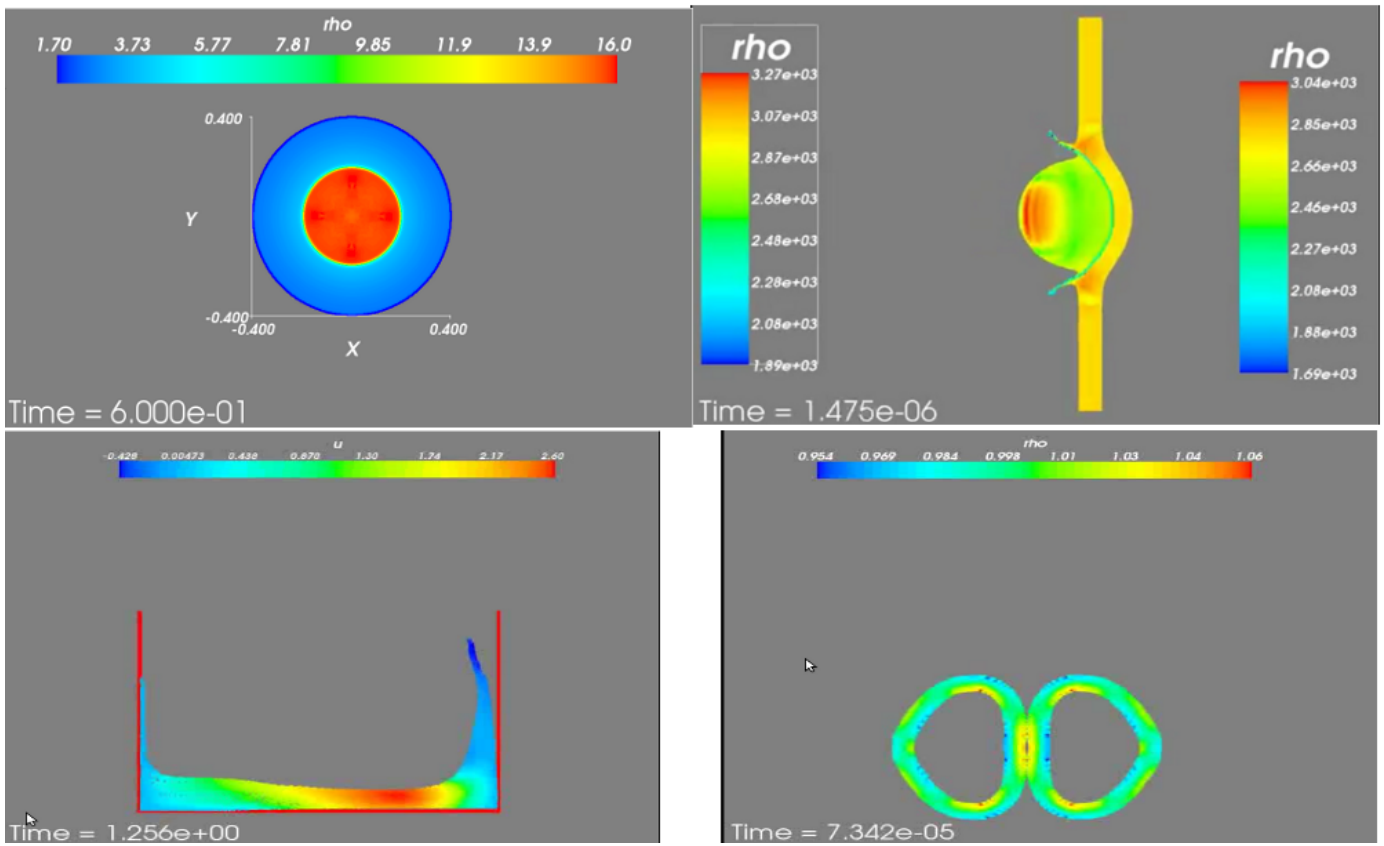
Used for ...

- Distributing *objects* across processors
- nodes, elements, points ...
- Equalizing work load
- Reducing communication time

Particle methods?

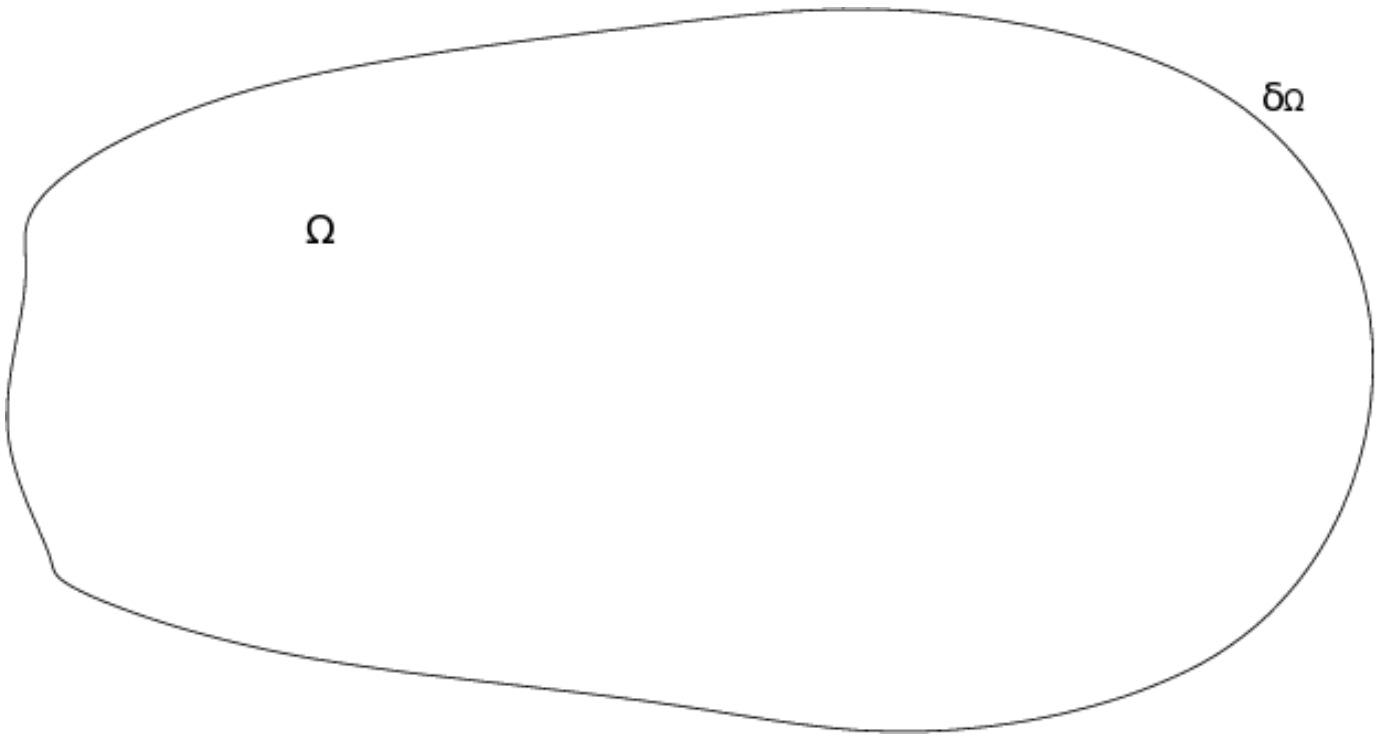
PySPH framework for Smooth Particle Hydrodynamics

`http://pysph.googlecode.com`



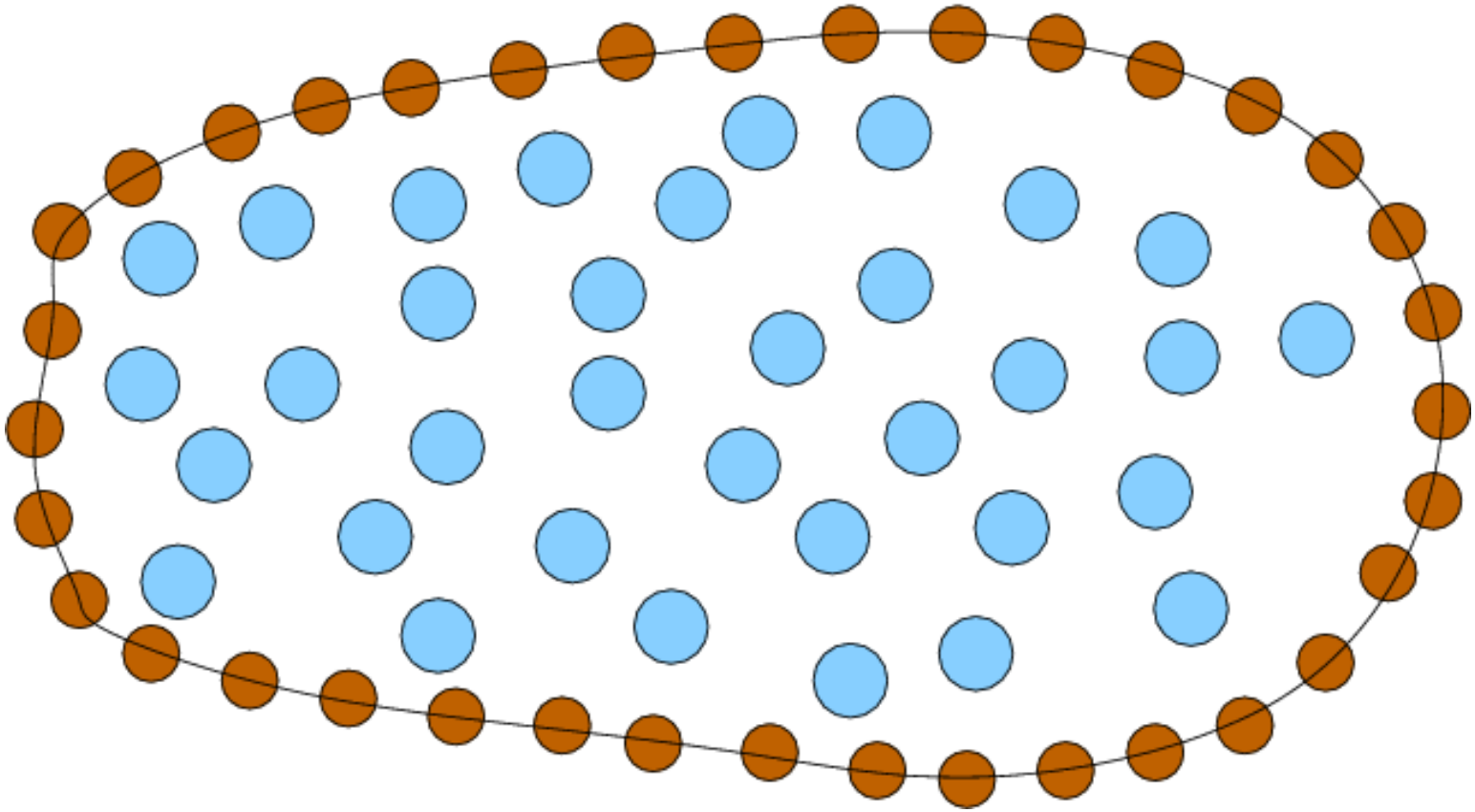
Particle methods in a nutshell

Consider a domain



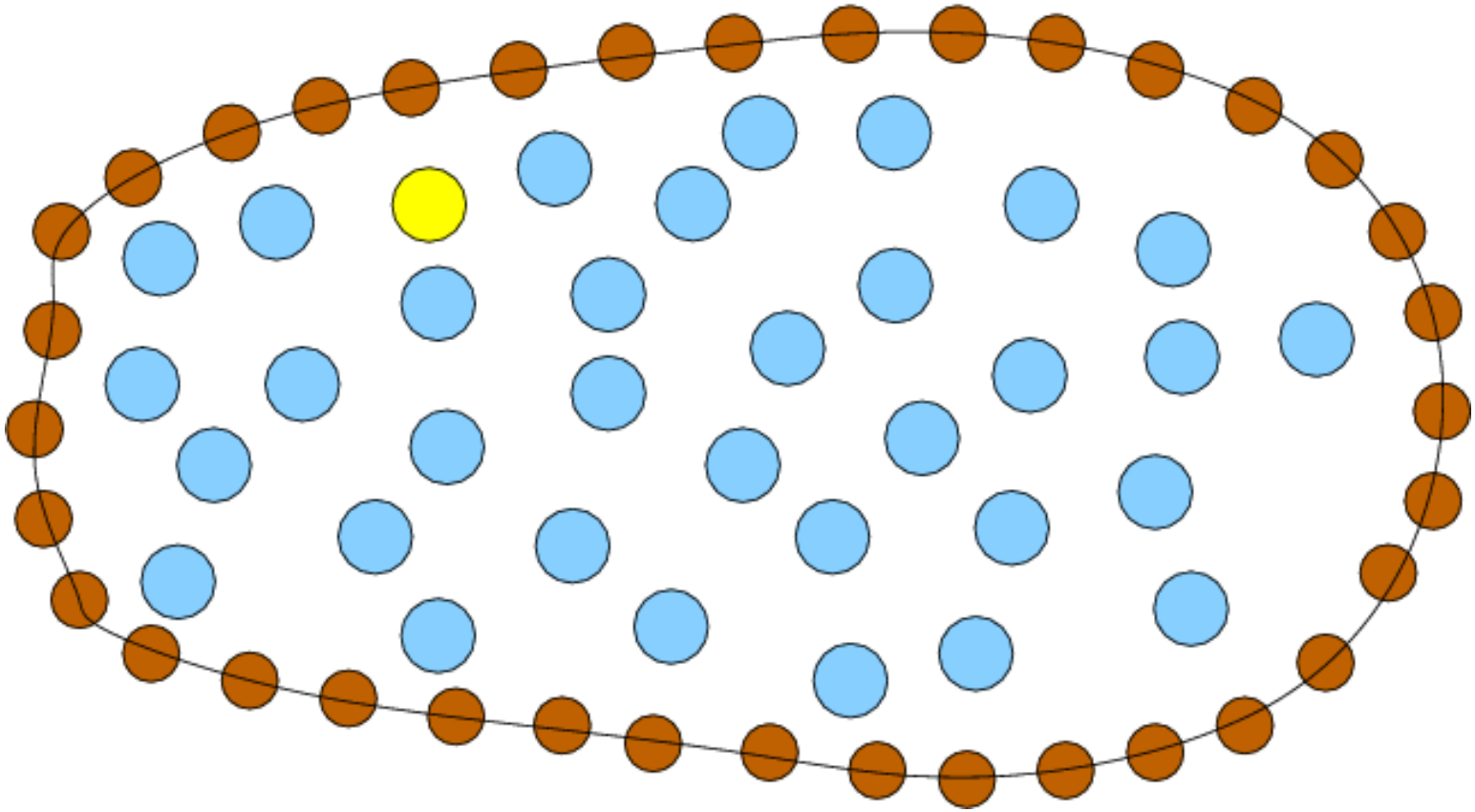
Particle methods in a nutshell

Discretize with "particles"



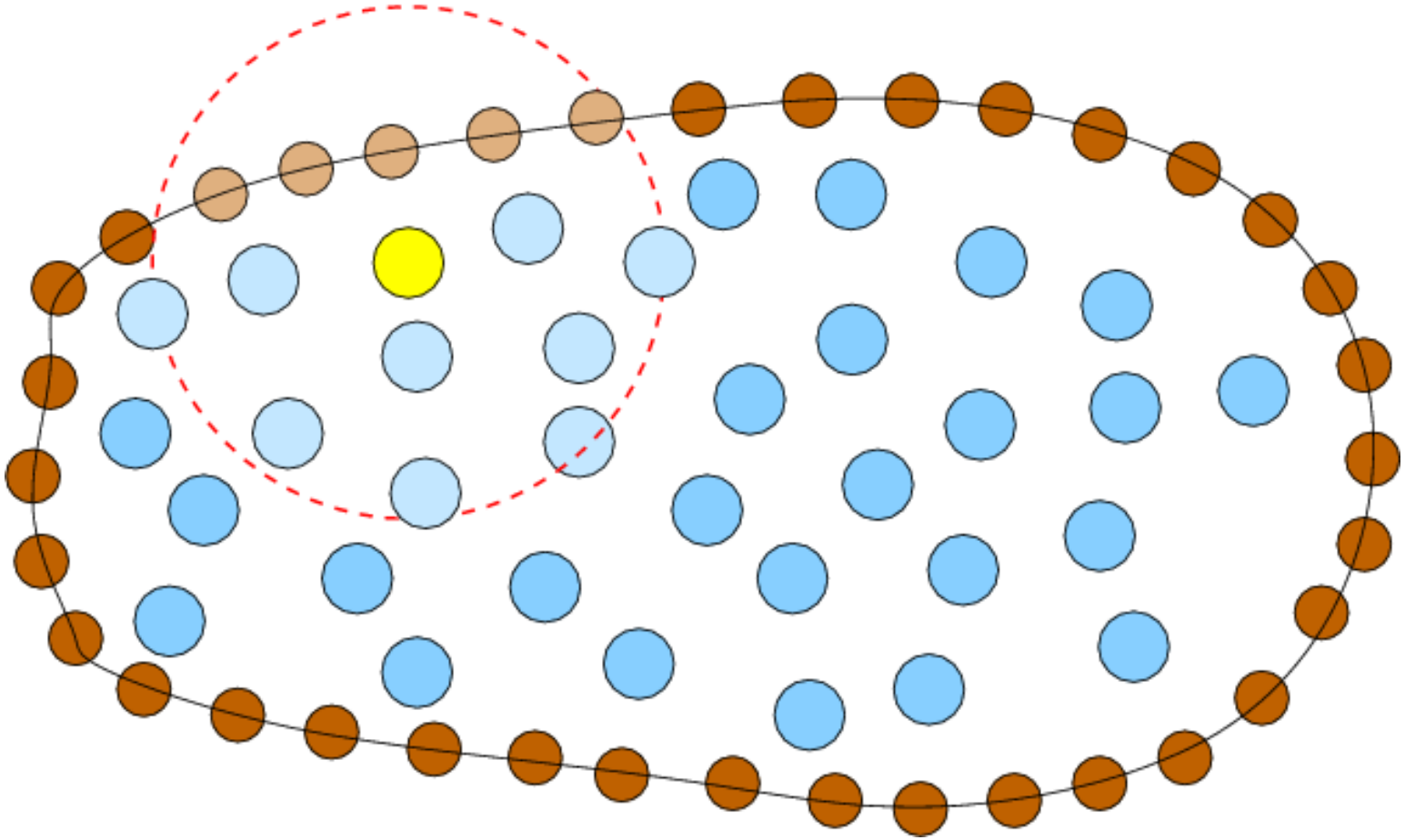
Particle methods in a nutshell

Take a particle



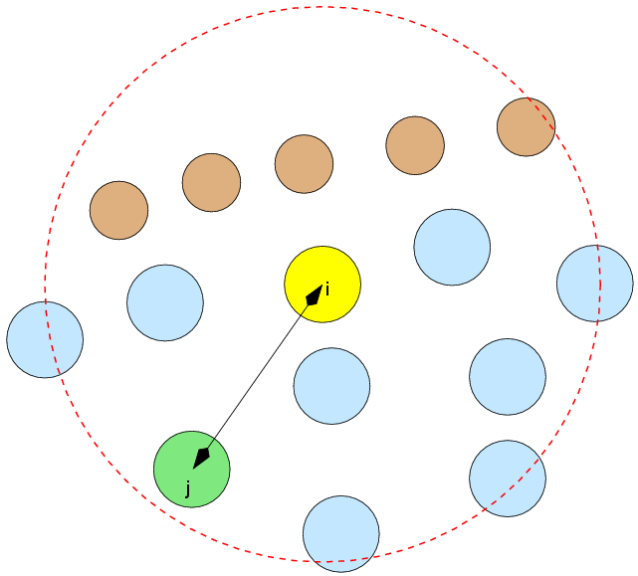
Particle methods in a nutshell

Find it's neighbors



Particle methods in a nutshell

Compute interactions from neighboring particles

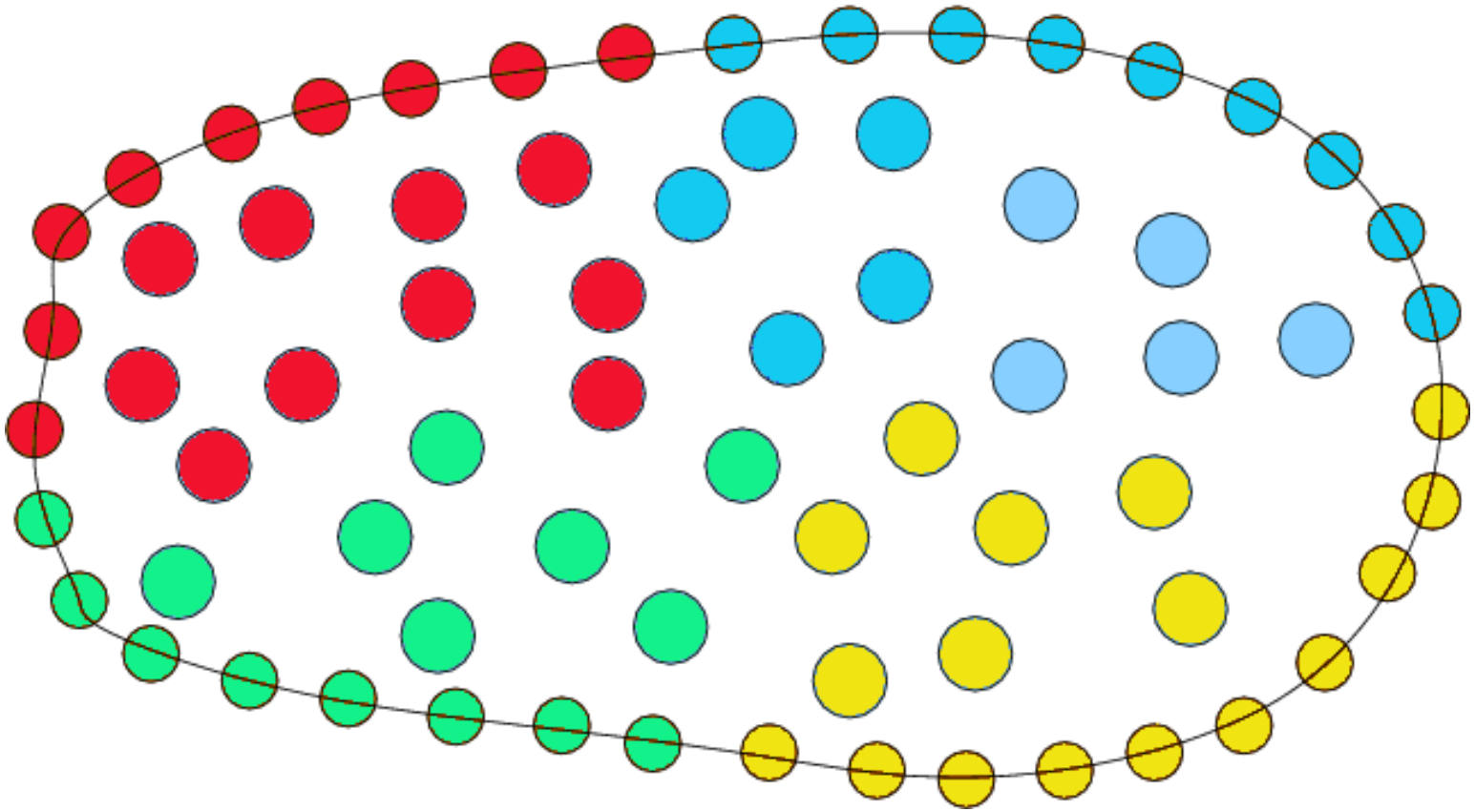


$$< f(x_i) > \approx \sum_j \frac{m_j}{\rho_j} f_j W(x_i - x_j, h_{ij})$$

$$< \nabla f(x_i) > \approx \sum_j \frac{m_j}{\rho_j} f_j \nabla W(x_i - x_j, h_{ij})$$

Particle methods in parallel

Particles colored according to processor assignment



General requirements in parallel

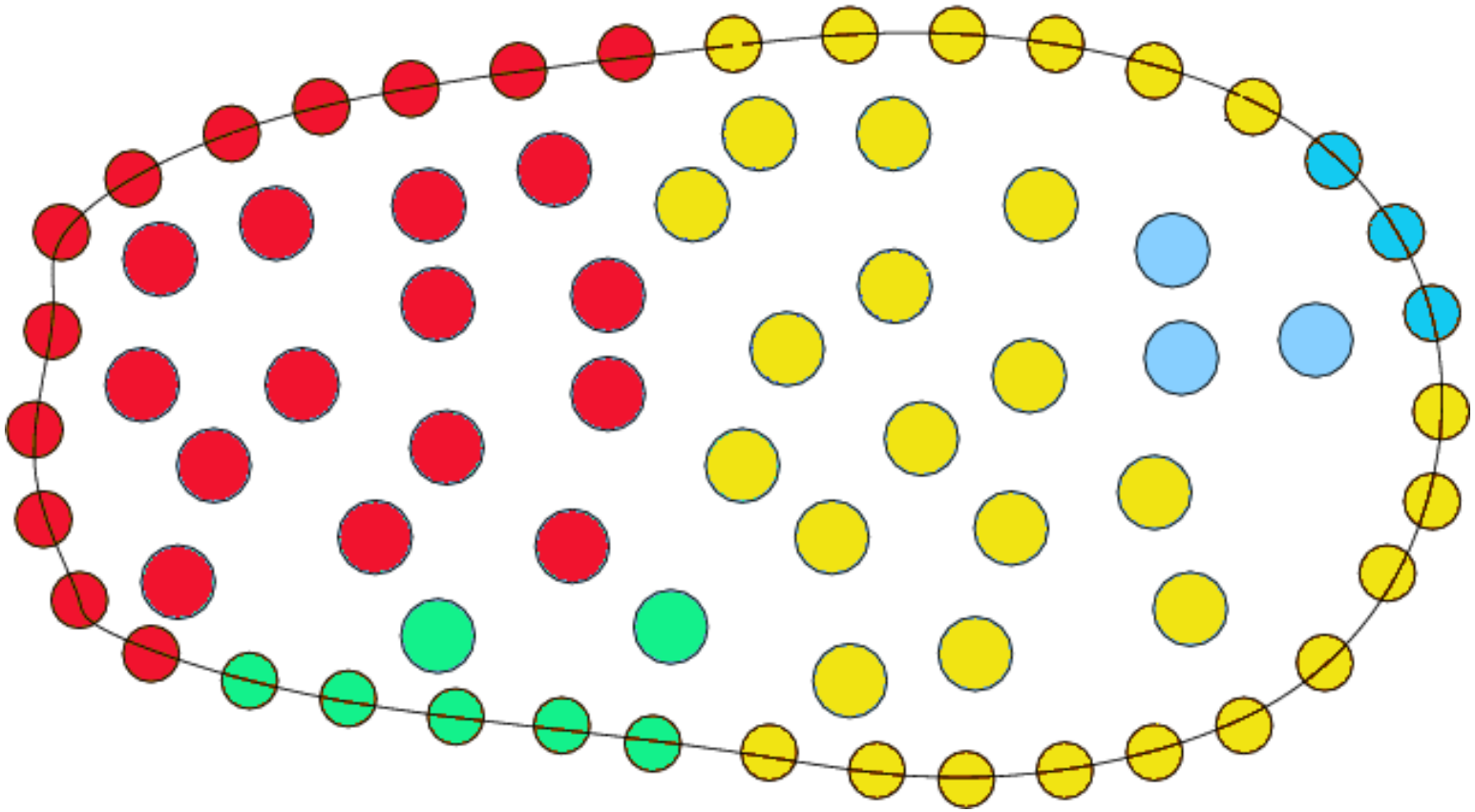
- Equal distribution of work load (volume constraint)
- Minimum communication (surface constraint)
- *Dynamic*

General requirements in parallel

- Equal distribution of work load (volume constraint)
- Minimum communication (surface constraint)
- *Dynamic*

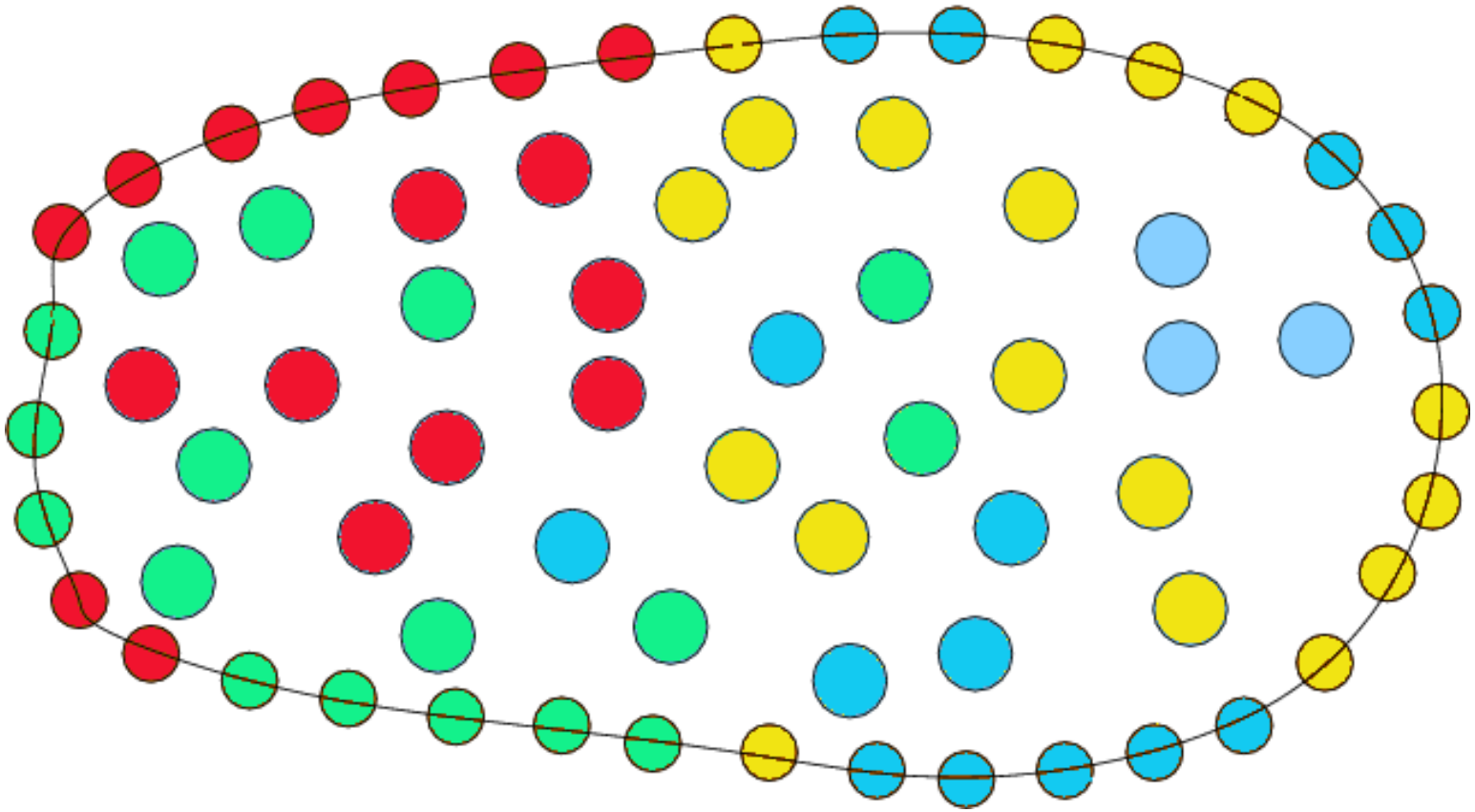
Equal distribution of work load

Unequal load

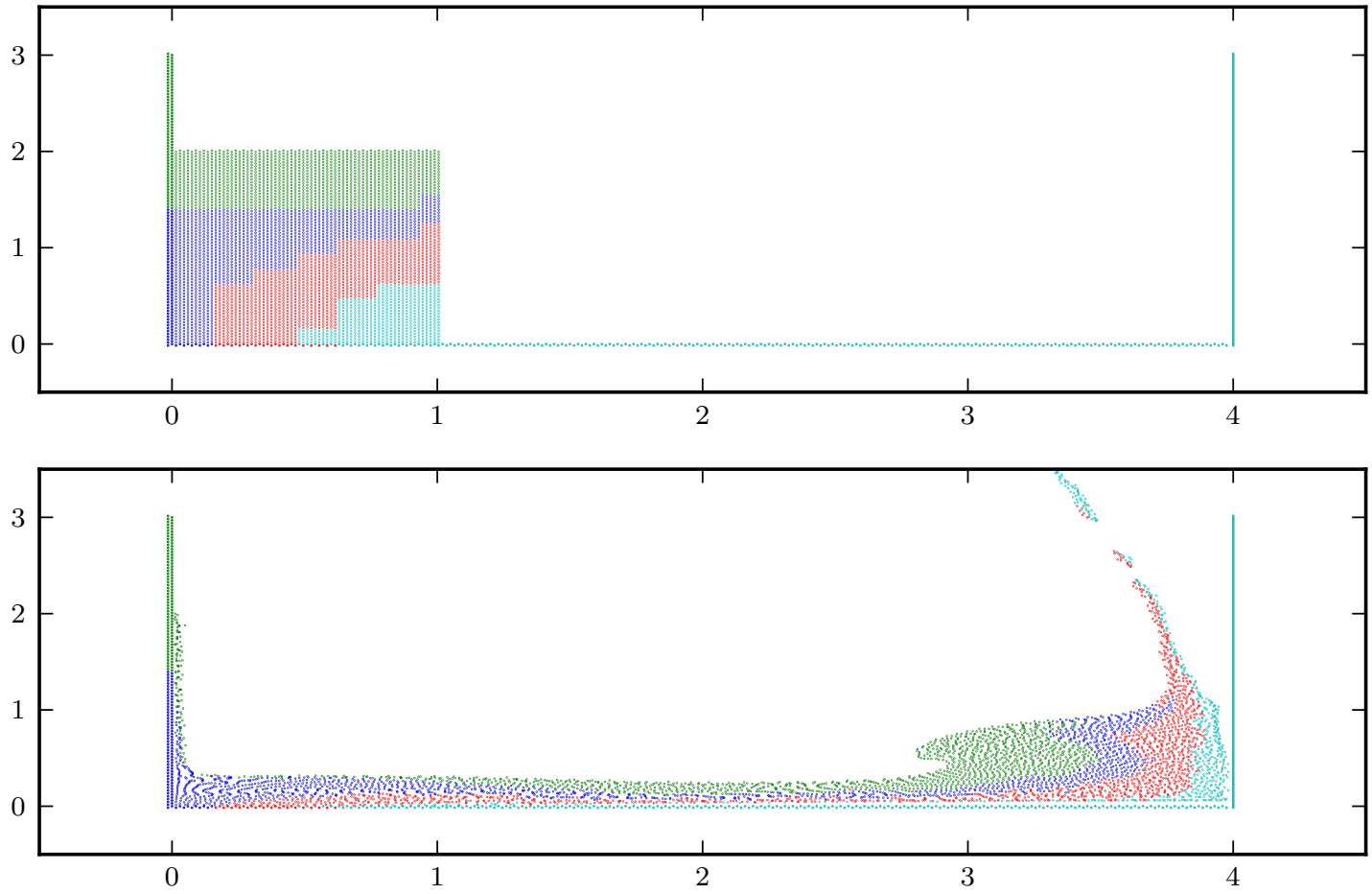


Minimum communication

Bad processor assignment

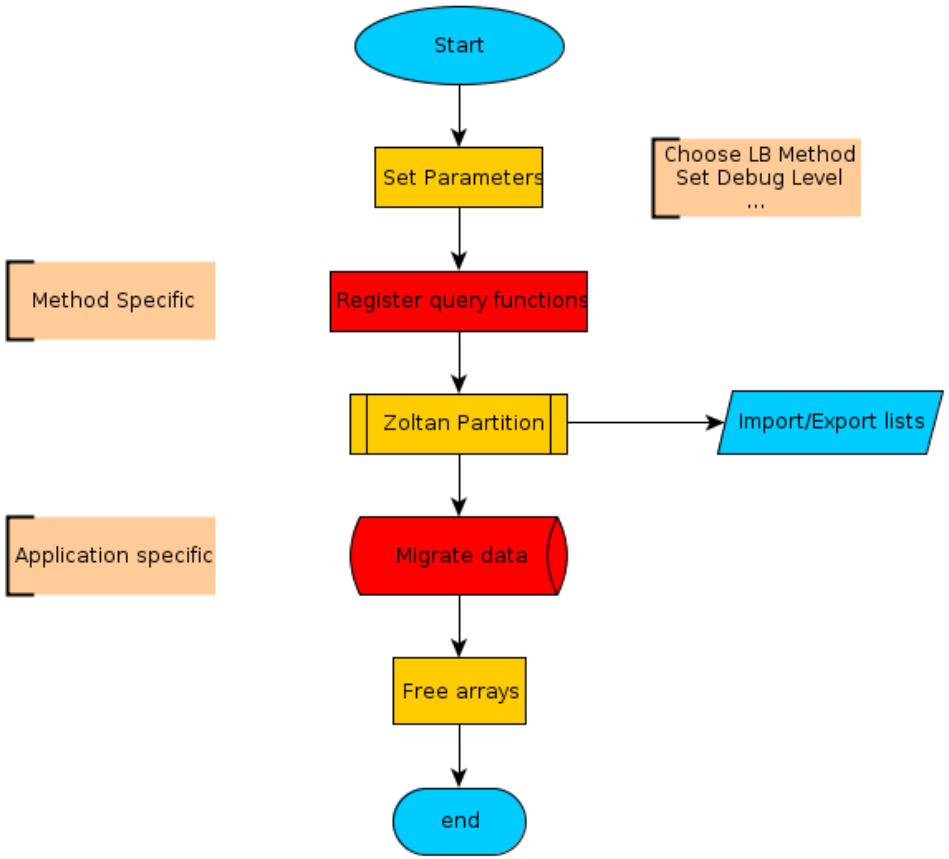


Dynamic



- Parallel load balancing
- Geometric + Graph partitioning
- Plugins (ParMetis, PTScotch)

Typical usage



Outline

Motivation

PySPH framework for Smooth Particle Hydrodynamics

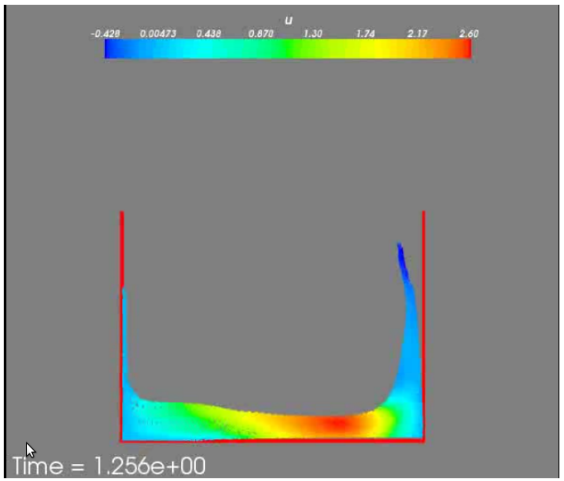
`http://pysph.googlecode.com`

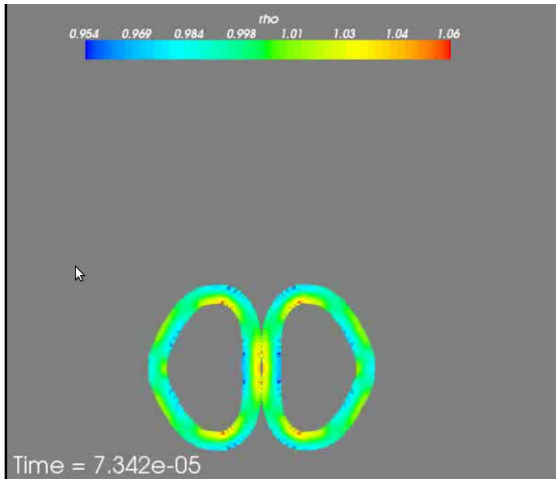
Features

- Python
- Multiple solvers
- Parallel

Limitations

- Robust parallel module
- Dynamic load balancing
- Scalable





Motivation

PySPH framework for Smooth Particle Hydrodynamics

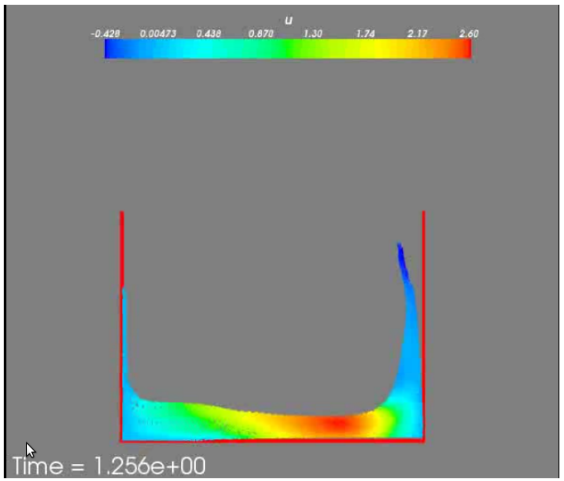
`http://pysph.googlecode.com`

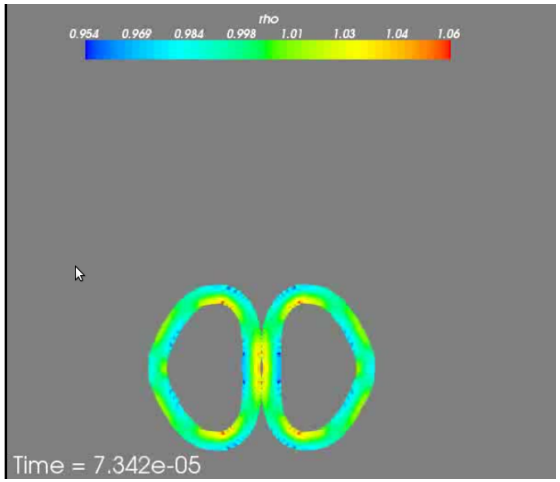
Features

- Python
- Multiple solvers
- Parallel

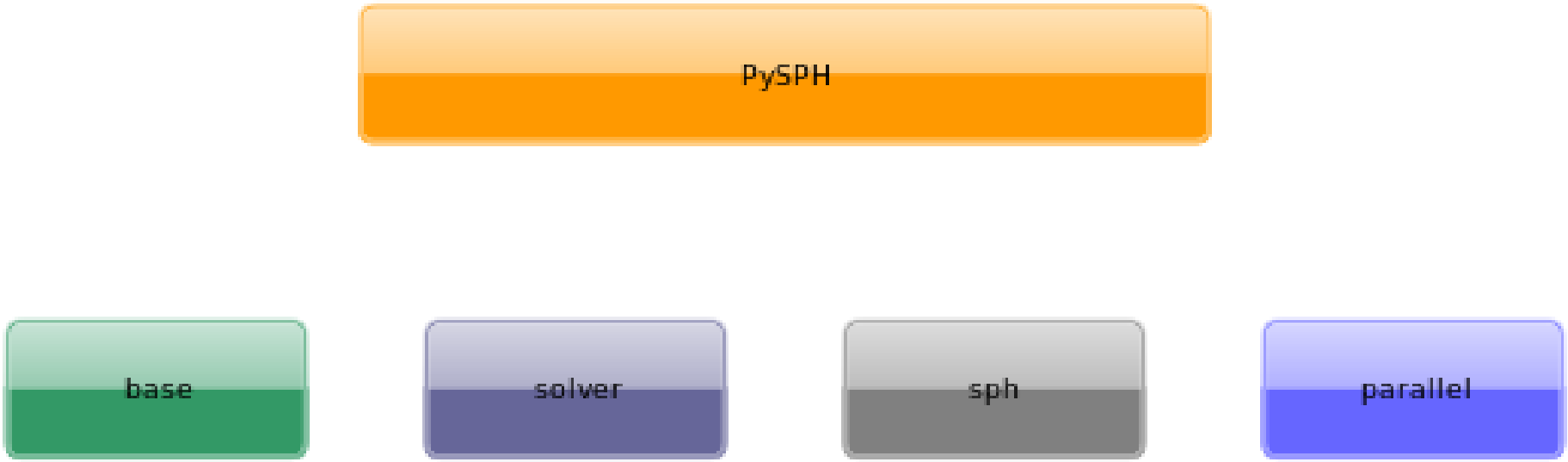
Limitations

- Robust parallel module
- Dynamic load balancing
- Scalable

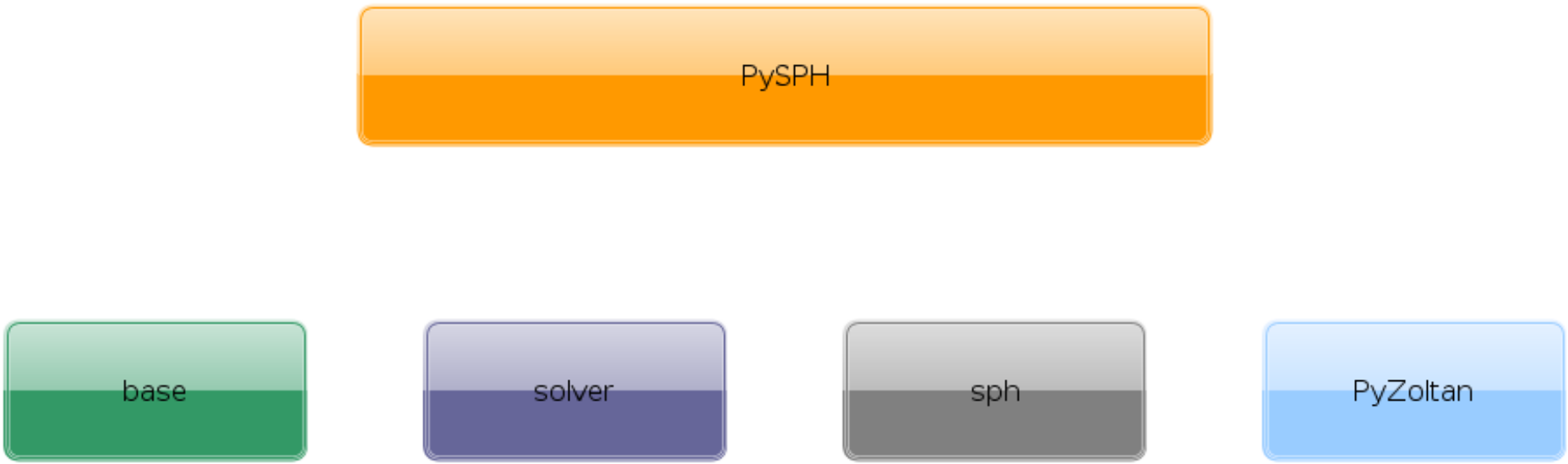




Motivation



Motivation



Justification

- Zoltan is already written!
- I can concentrate on the Physics
- PyZoltan would be kinda nice

Justification

- Zoltan is already written!
- I can concentrate on the Physics
- PyZoltan would be kinda nice

Justification

- Zoltan is already written!
- I can concentrate on the Physics
- PyZoltan would be kinda nice

What is Cython?

C in Python

- Python like language
- C/C++ extensions for Python
- C/C++ wrapper

What is Cython?

C in Python

- Python like language
- C/C++ extensions for Python
- C/C++ wrapper

Why Cython?

Other tools

- SWIG
- ctypes
- boost python

Cython

- Straightforward
- PySPH uses Cython extensions

Why Cython?

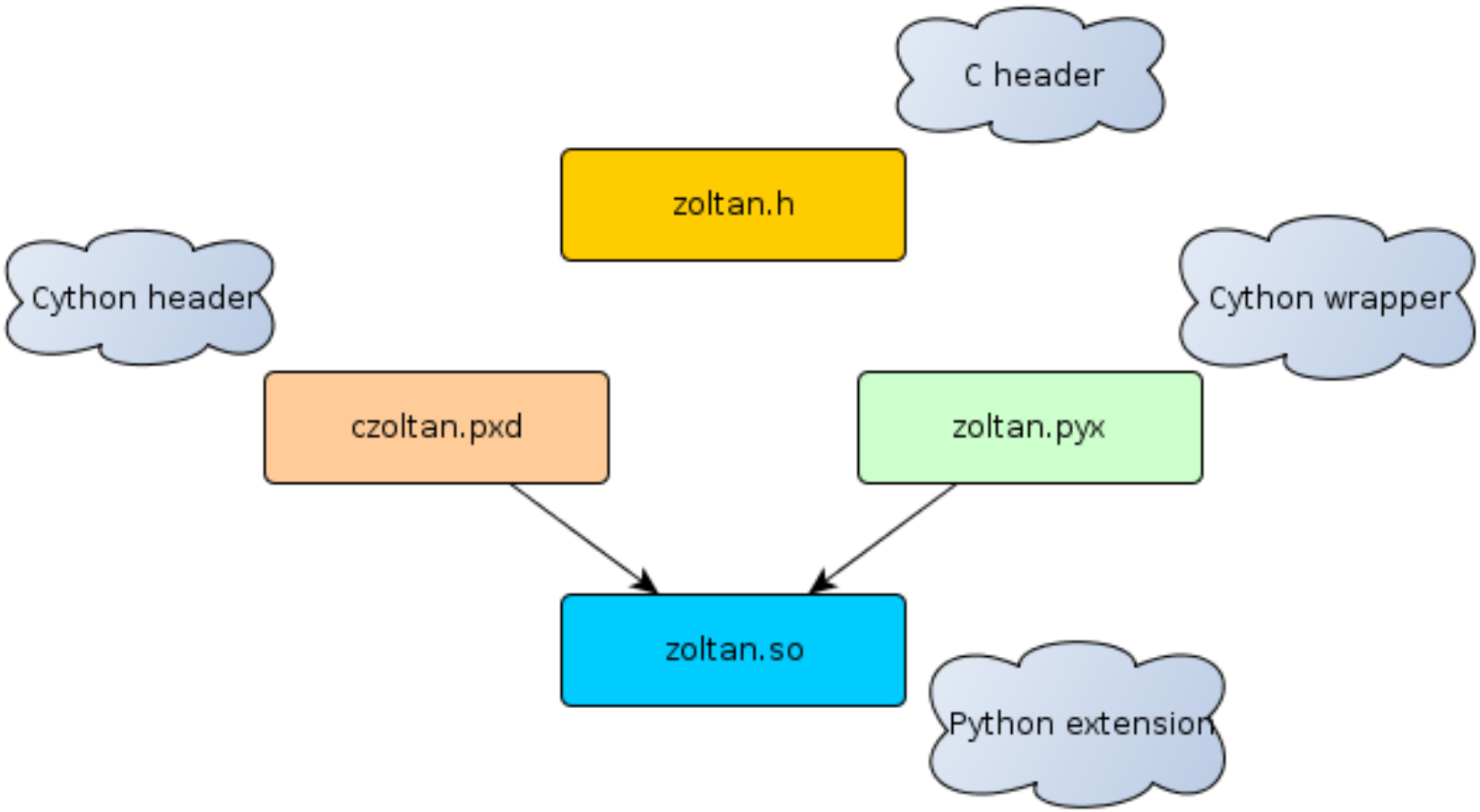
Other tools

- SWIG
- ctypes
- boost python

Cython

- Straightforward
- PySPH uses Cython extensions

General workflow



Outline

zoltan.h

```
#define ZOLTAN_VERSION_NUMBER    3.6

enum Zoltan_Fn_Type {ZOLTAN_NUM_EDGES_FN_TYPE,
             ZOLTAN_NUM_EDGES_MULTI_FN_TYPE};

struct Zoltan_Struct;
extern struct Zoltan_Struct *Zoltan_Create(MPI_Comm communicator);

extern int Zoltan_Initialize(
  int argc, char **argv, float *ver);

typedef int ZOLTAN_NUM_OBJ_FN(
  void *data, int *ierr);

extern int Zoltan_Set_Num_Obj_Fn(
  struct Zoltan_Struct *zz, ZOLTAN_NUM_OBJ_FN *fn_ptr,  void *data_ptr);
```

czoltan.pxd

```
cdef extern from "zoltan.h":

    # Zoltan version number
    float ZOLTAN_VERSION_NUMBER

    enum Zoltan_Fn_Type:
        ZOLTAN_NUM_EDGES_FN_TYPE
        ZOLTAN_NUM_EDGES_MULTI_FN_TYPE

    struct Zoltan_Struct: pass

    extern Zoltan_Struct* Zoltan_Create(MPI_Comm)

    # Initialize Zoltan
    extern int Zoltan_Initialize(int, char**, float* ver)

    ctypedef int ZOLTAN_NUM_OBJ_FN(void *data, int *ierr)

    extern int Zoltan_Set_Num_Obj_Fn(Zoltan_Struct *zz, ZOLTAN_NUM_OBJ_FN *fn_ptr, void↩
          *data_ptr)
```

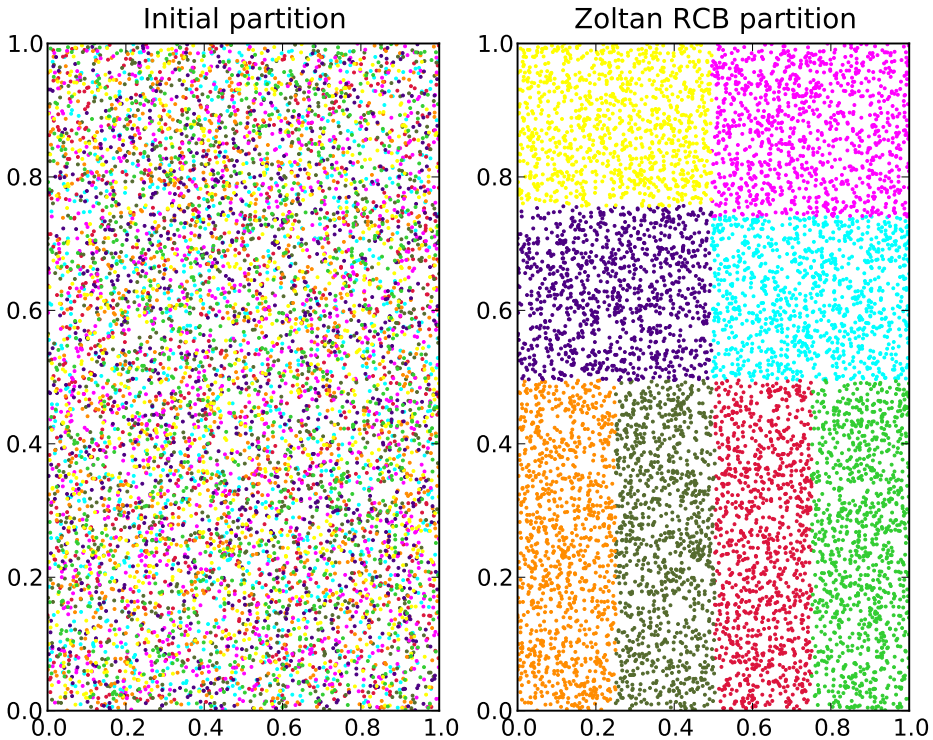
zoltan.pyx

```
# Import the Cython header
cimport czoltan

def zoltan_version_number():
    return czoltan.ZOLTAN_VERSION_NUMBER

cdef _zoltan_create(mpi.MPI_Comm comm):
    cdef Zoltan_Struct* zz = czoltan.Zoltan_Create(comm)

cdef _zoltan_initialize( int argc, args, float* version ):
    cdef char **c_argv
    args = [ bytes(x) for x in args ]
    c_argv = <char**>malloc( sizeof(char*) *len(args) )
    if c_argv is NULL:
        raise MemoryError()
    try:
        for idx, s in enumerate( args ):
            c_argv[idx] = s
    finally:
        free( c_argv )

    error_code = cython.declare(cython.int)
    error_code = czoltan.Zoltan_Initialize(len(args), c_argv, version)
    return error_code
```

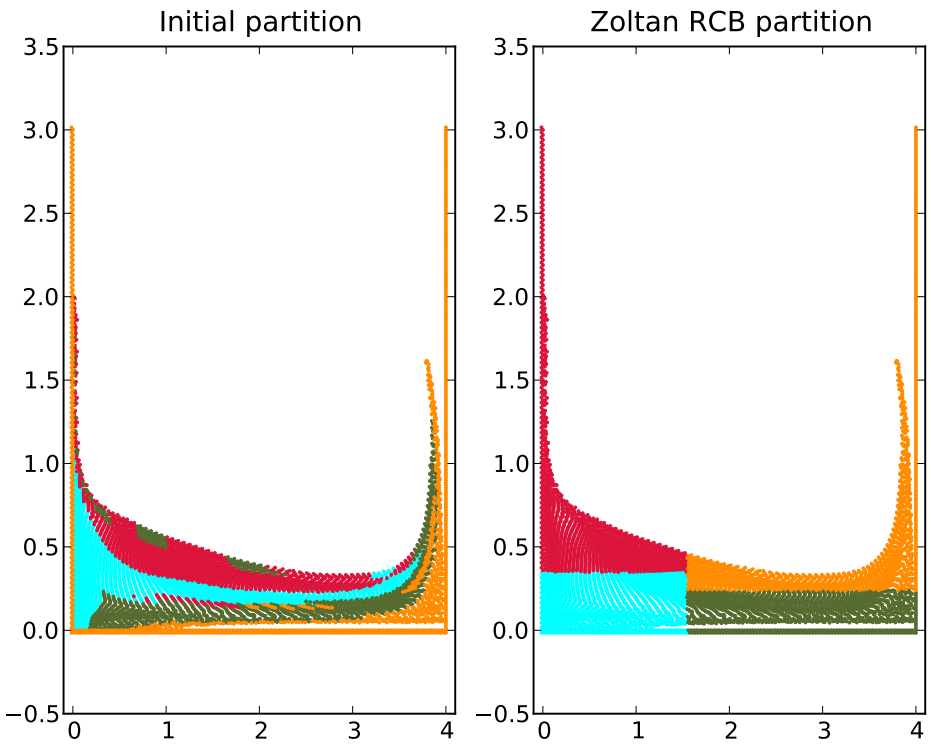
setup.py

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext
import commands, mpi4py, os

zoltan_include_dirs = [ os.environ['ZOLTAN_INCLUDE'] ]
zoltan_library_dirs = [ os.environ['ZOLTAN_LIBRARY'] ]

mpic = 'mpicc'
mpi_include_dirs = [ commands.getoutput( mpic + ' --showme:incdirs' ) ]
mpi_include_dirs.append(mpi4py.get_include())
mpi_library_dirs = [ commands.getoutput( mpic + ' --showme:link' ) ]

include_dirs = zoltan_include_dirs + mpi_include_dirs
library_dirs = zoltan_library_dirs + mpi_library_dirs

ext_modules = [
    Extension( name="zoltan", sources=['zoltan.pyx'], libraries=['zoltan', 'mpi'],
               include_dirs=include_dirs, library_dirs=library_dirs, pyrex_dbg=True),

setup(name="PyZoltan", cmdclass = {'build_ext': build_ext},
      ext_modules = ext_modules)
```

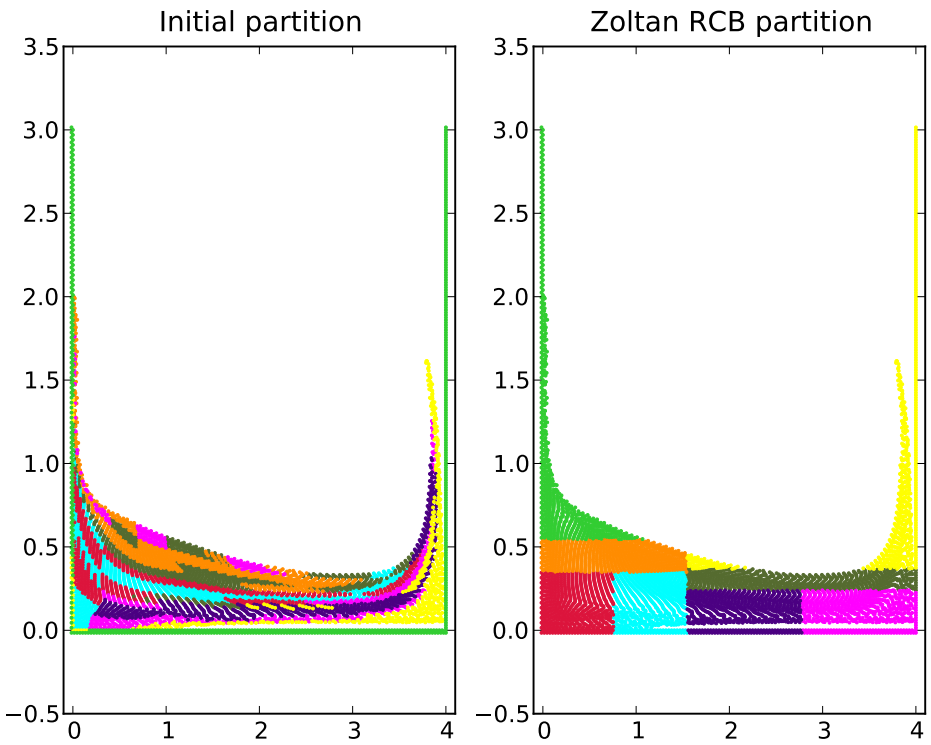
Particles in a box



Dam break problem: 4 Processors



Dam break problem: 8 Processors



Outline

Summary

- Python wrapper around Zoltan (Infancy)
- Explore different partitioners

Appeal

- Applications (SPH, MD, ALE, FEM)
- Anybody with wrapping experience
- Load balancing pitfalls

Summary

- Python wrapper around Zoltan (Infancy)
- Explore different partitioners

Appeal

- Applications (SPH, MD, ALE, FEM)
- Anybody with wrapping experience
- Load balancing pitfalls

Thank you