scientific programming to develop code of the space-time conservation element and solution element (CESE) method in Python and C++

Yung-Yu Chen

SciPyData Japan 2025

adjunct assistant professor at NYCU

yyc@sciwork.dev

https://yyc.solvcon.net/

scientific computing

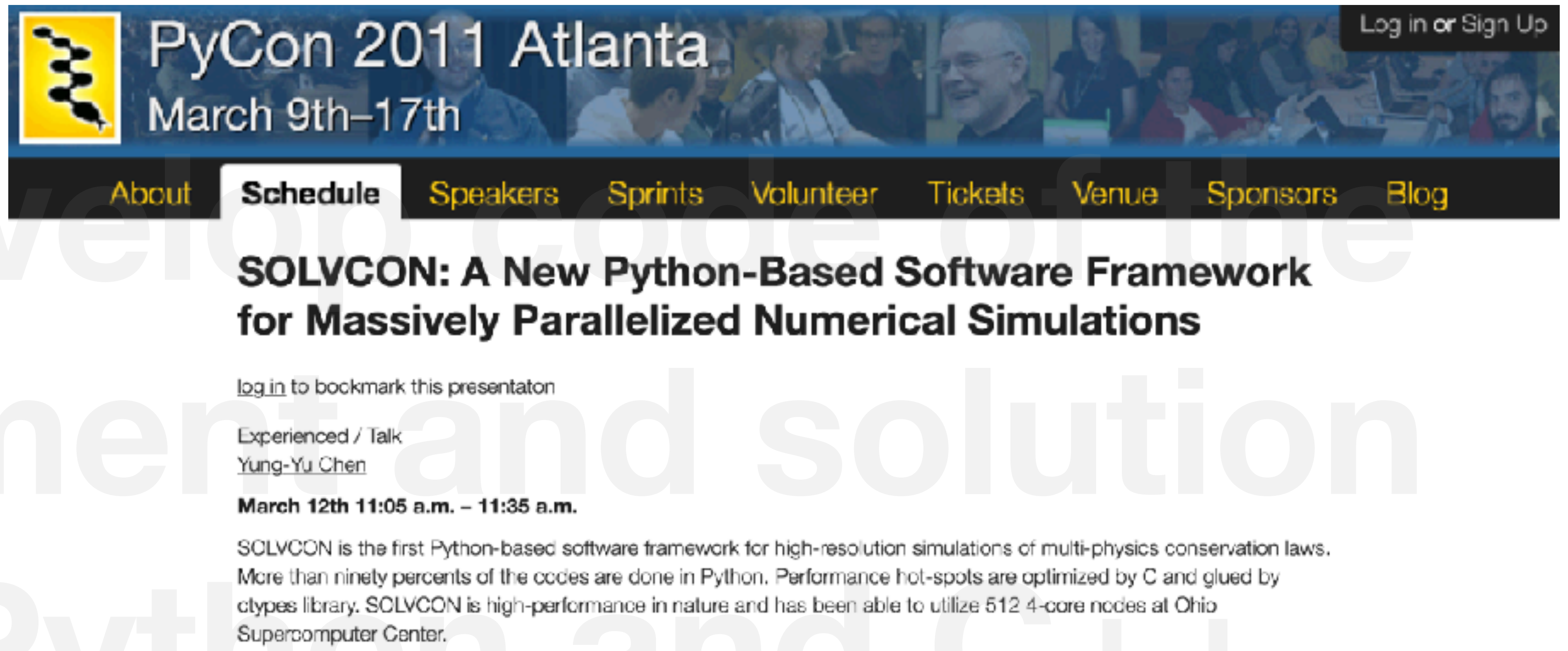
- computing builds this digital world of information
  - Internet, smart phones, virtual reality, all the fancy things
- the term "scientific computing" is ... not 100% satisfying me
  - we can only control everything by writing code ourselves
  - computing without coding misses the details
- the term "scientific programming" emphasizes coding

scientific programming for me

- time-accurate simulations with the continuum assumption
  - first-order, non-linear hyperbolic partial differential equations (PDEs)
- numerical clarity for non-linear systems
  - numerical method needs to handle non-linearity intrinsically
  - fit complex geometry in three-dimensional space
- software clarity for engineering: reproducible results
  - not only for research and applications, but also extension to other areas

it involves so many things

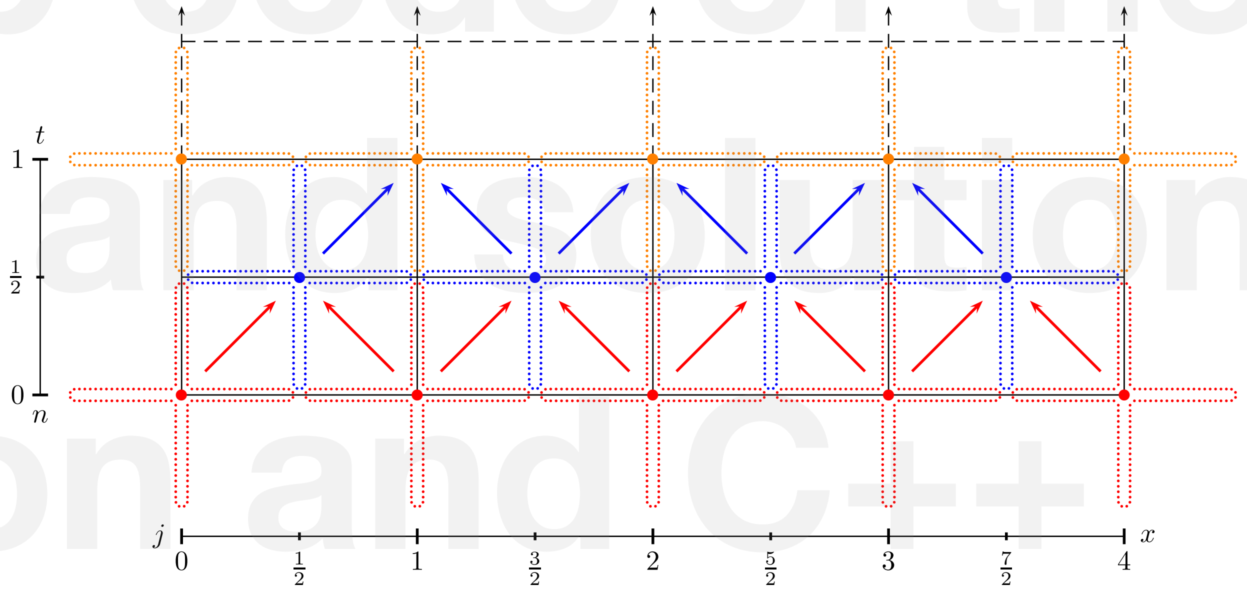
- in the beginning, I thought Python alone suffices
  - but scientific programming is hard
- in reality, much more to handle:
  - hardware, instructions, and performance **computer architecture**
  - resource management **system programming**
  - "bookkeeping" is as important as numerical methods **software architecture**
  - version control, build system, ... and everything **software engineering**



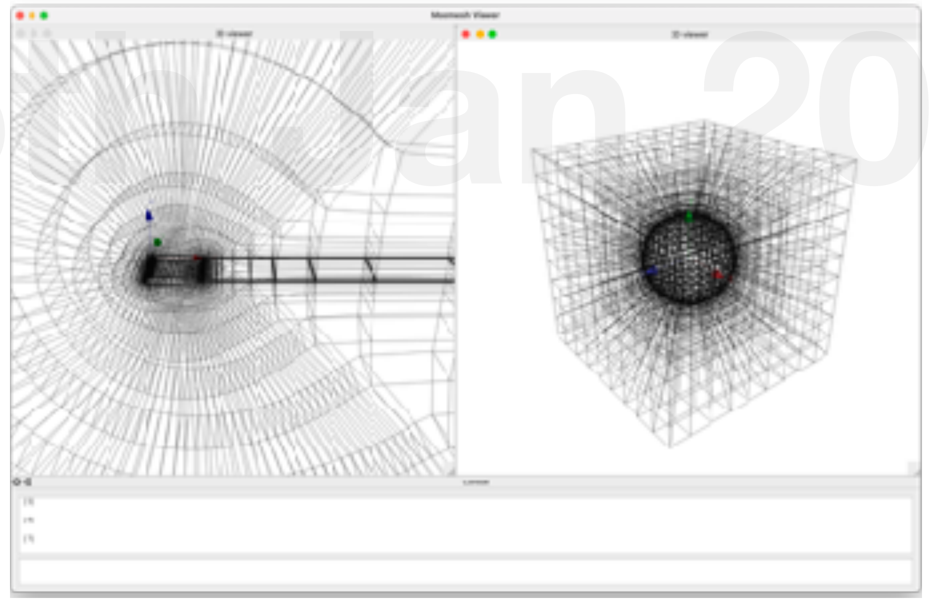
redoing things many times

topics today

- use the space-time conservation element and solution element (CESE) method to solve hyperbolic PDEs



- unstructured meshes of mixed-shape elements



- array library for the numerical method and geometry

solve first-order hyperbolic PDEs

- the space time conservation element and solution element (CESE) method is developed in the 90s to solve first-order, non-linear hyperbolic PDEs

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{\partial \mathbf{F}^{(1)}(\mathbf{u})}{\partial x_1} + \frac{\partial \mathbf{F}^{(2)}(\mathbf{u})}{\partial x_2} + \frac{\partial \mathbf{F}^{(3)}(\mathbf{u})}{\partial x_3} = \underbrace{\boxed{\mathbf{R}(\mathbf{u})}}_{\text{source term}}$$

PDEs in the first-order form

$$\frac{\partial \mathbf{u}}{\partial t} + \sum_{k=1}^{3} \frac{\partial \mathbf{F}^{(k)}(\mathbf{u})}{\partial x_k} = 0$$

first-order hyperbolic PDEs

- propagating wave

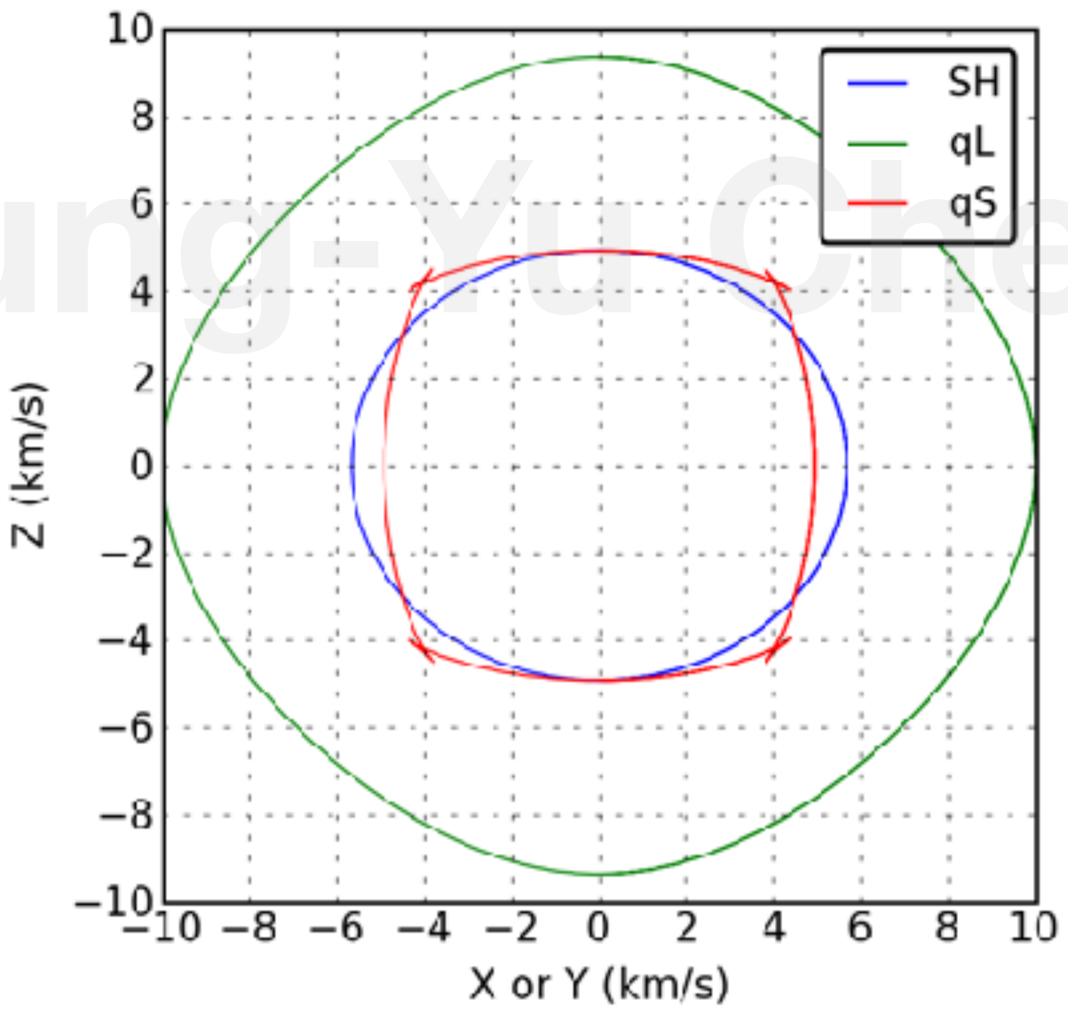
join #solvcon on discord

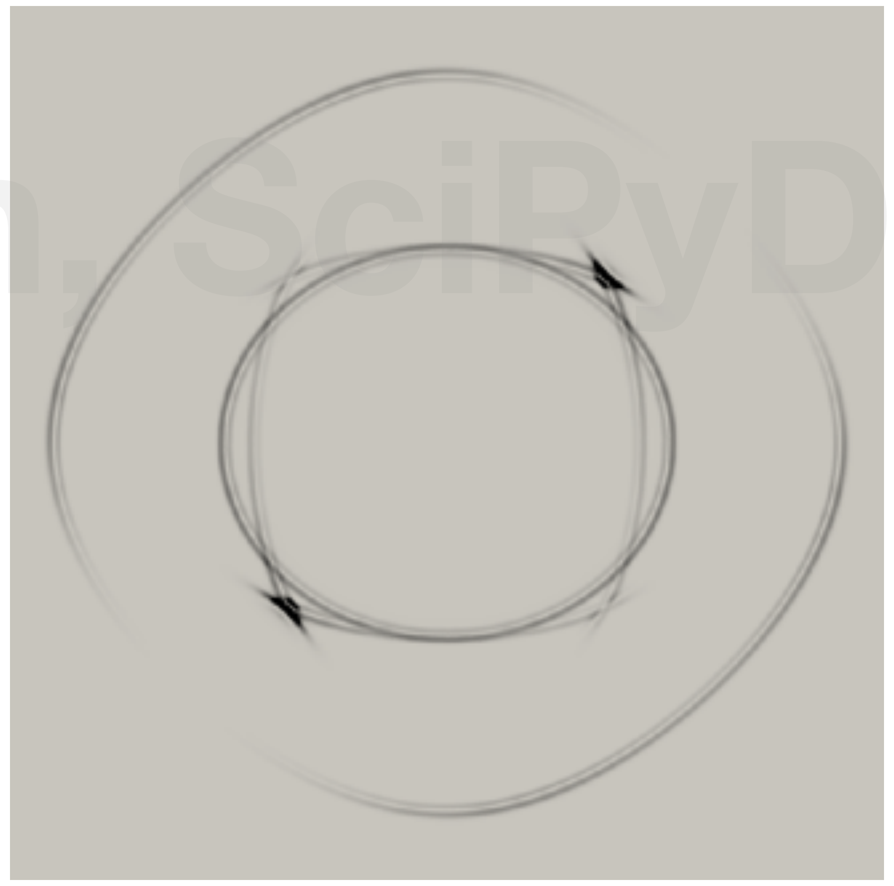
https://discord.gg/P9U7PFv

old results (2011)

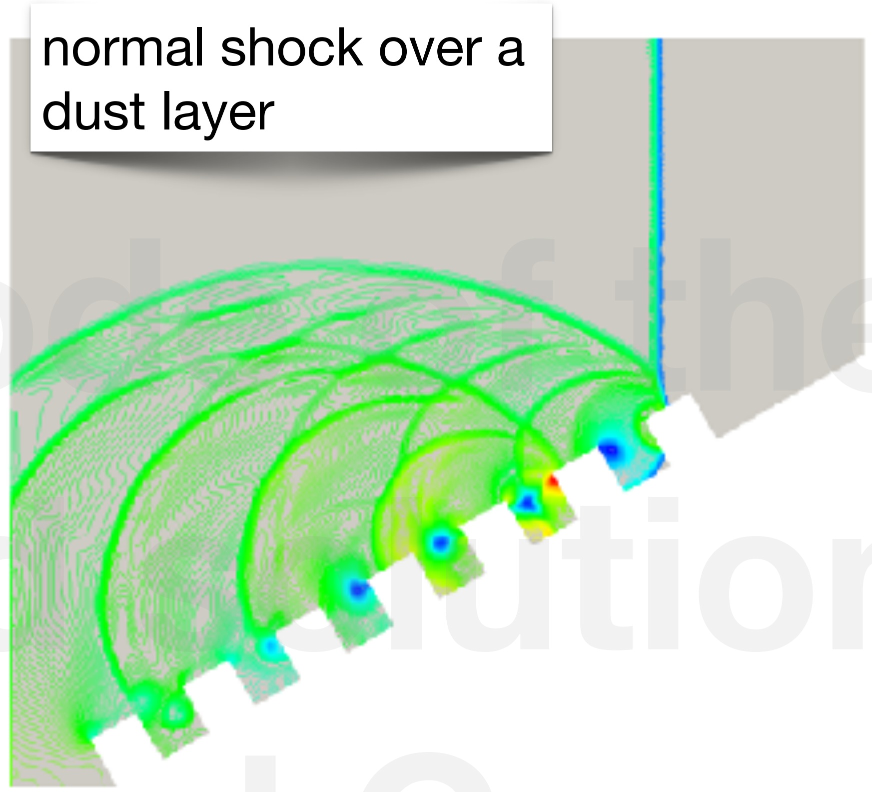
conservation laws: $\dfrac{\partial \mathbf{u}}{\partial t} + \sum_{k=1}^{3} \dfrac{\partial \mathbf{F}^{(k)}(\mathbf{u})}{\partial x_k} = 0$

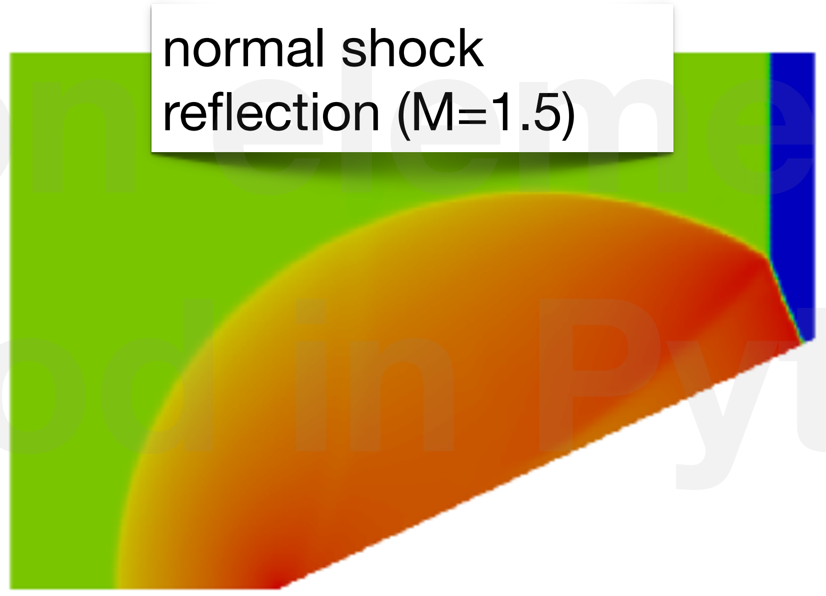
stress waves in anisotropic solids (hexagonal symmetry)

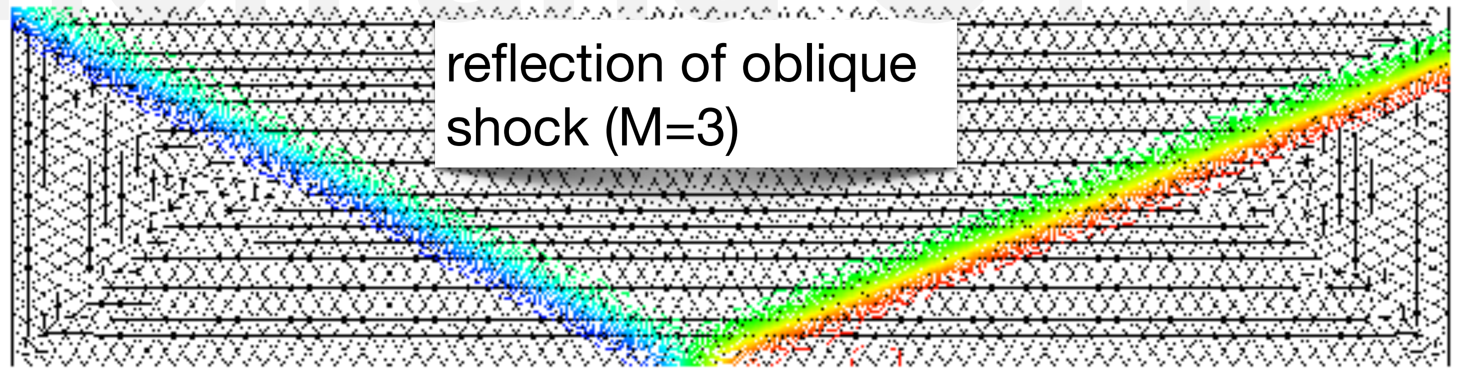


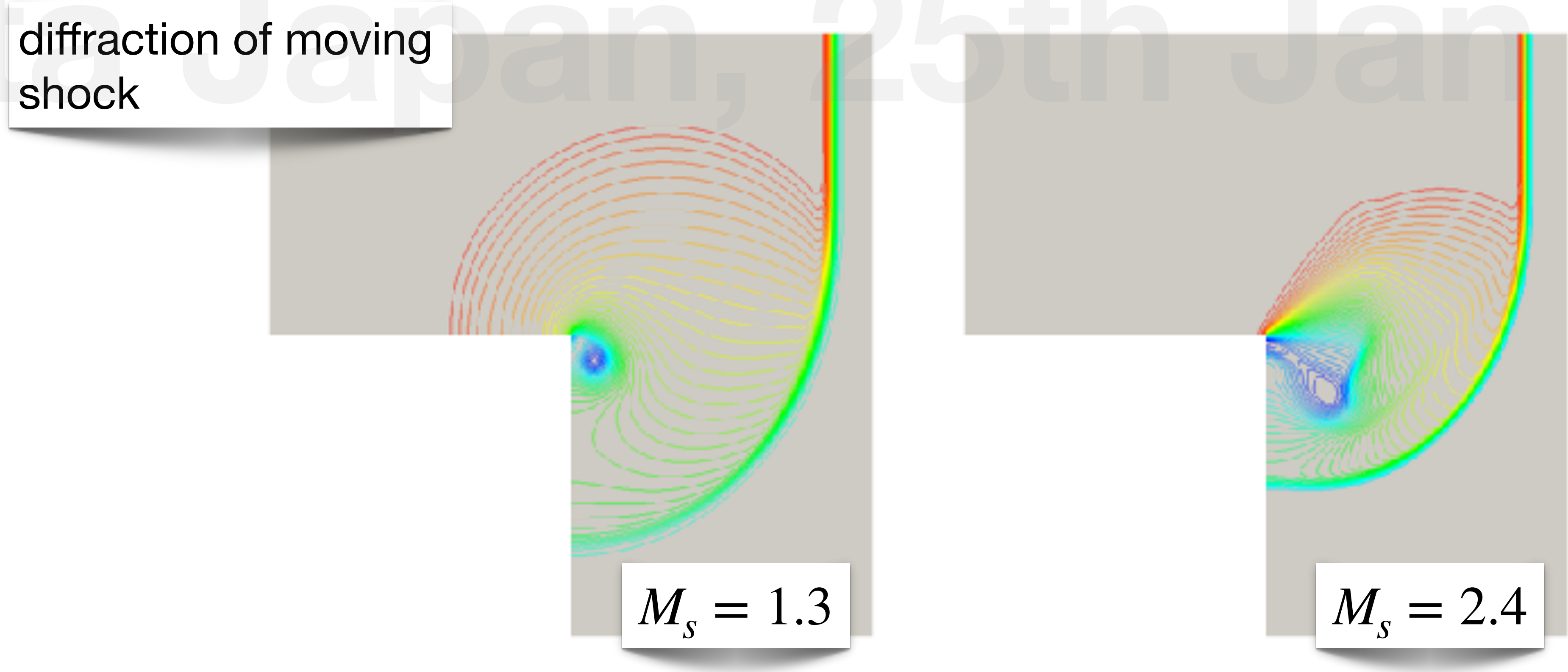


two-dimensional compressible flow









join #solvcon on discord

https://discord.gg/P9U7PFv

integral equation in (x, t)

differential equation

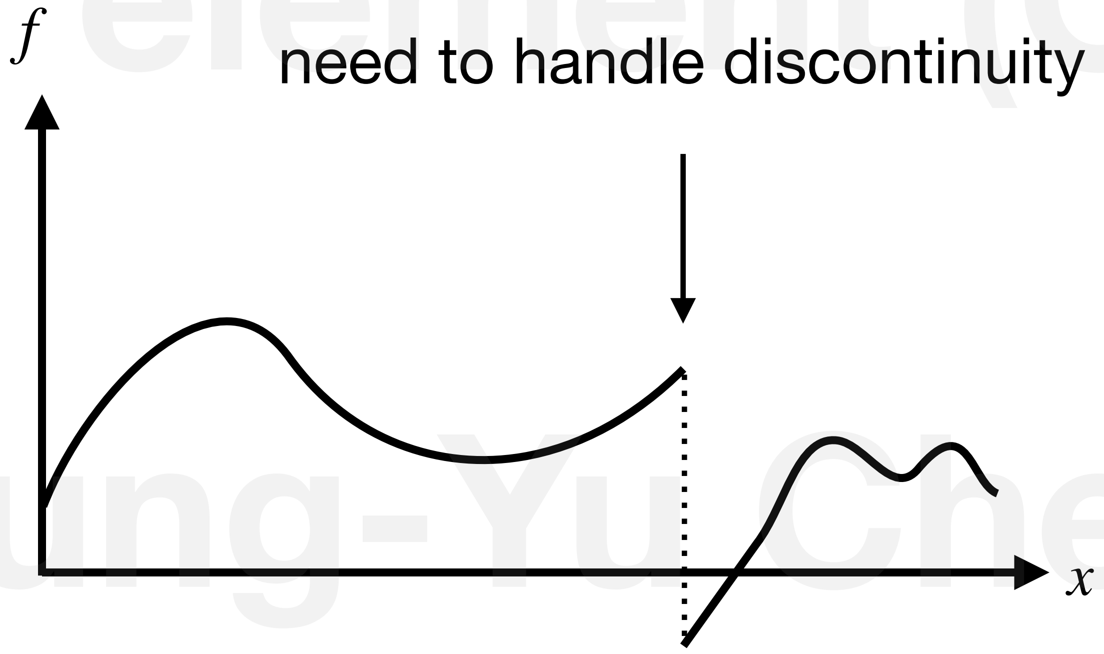
$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0$$

by $\mathbf{h} \overset{\text{def}}{=} \big(f(u), u\big)$

in Euclidean 2-space (x, t)

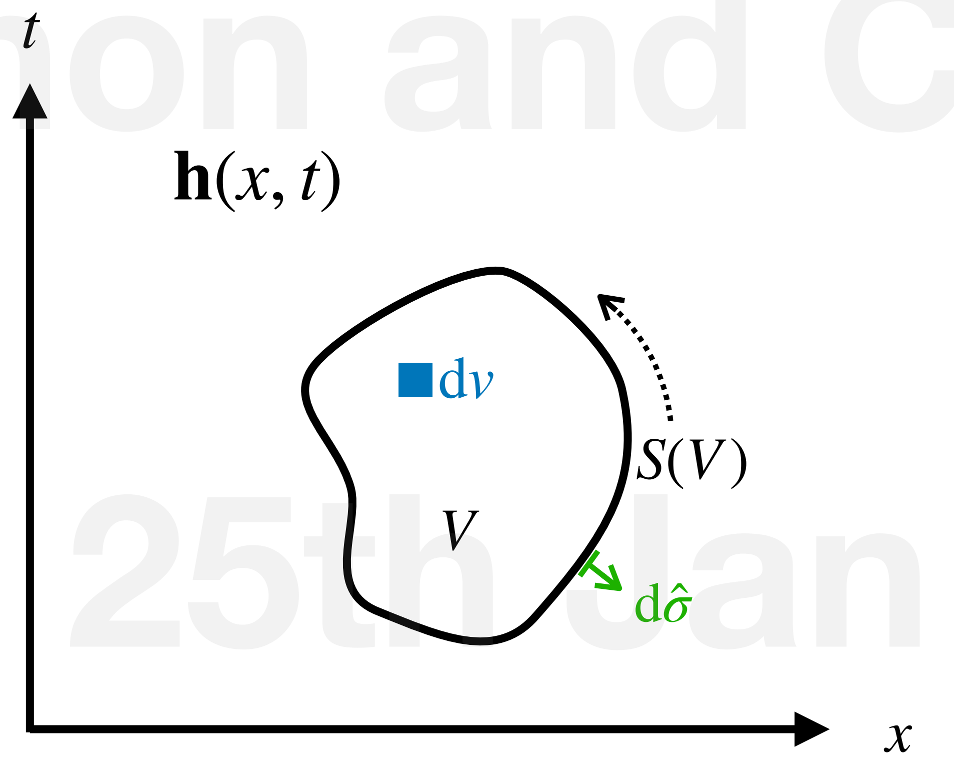
integral equation

$$\int_V \nabla \cdot \mathbf{h} \, \mathrm{d}v = 0$$



$$\nabla \cdot \mathbf{h} = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial t}\right) \cdot \big(f(u), u\big)$$

$$= \frac{\partial f(u)}{\partial x} + \frac{\partial u}{\partial t}$$



divergence theorem

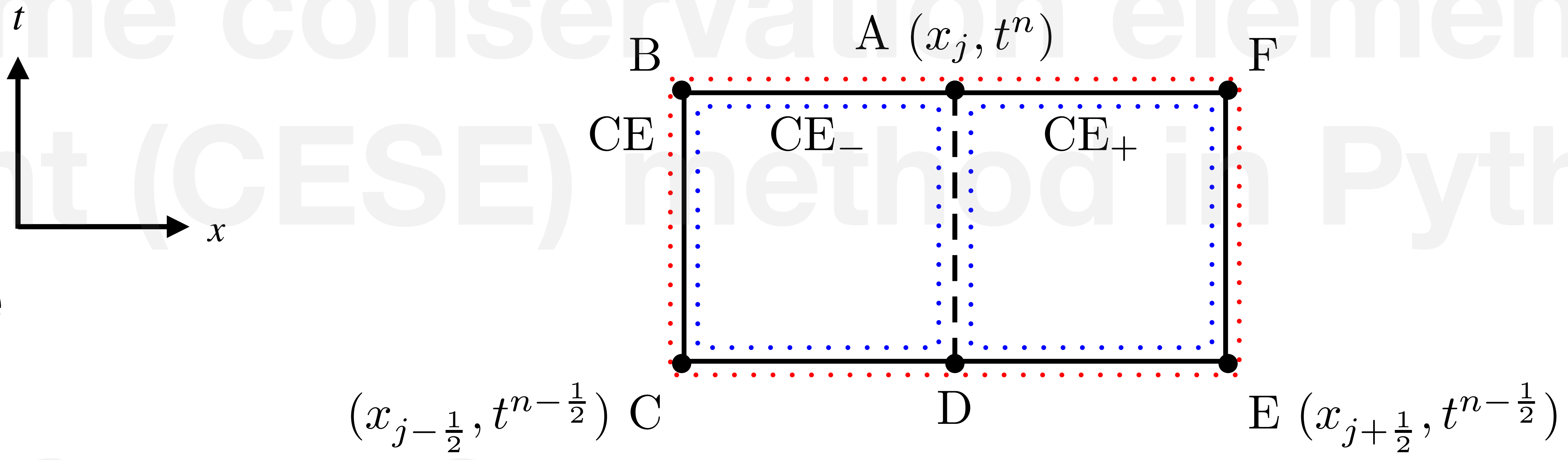
$$\oint_{S(V)} \mathbf{h} \cdot \mathrm{d}\hat{\sigma} = 0$$

join #solvcon on discord

https://discord.gg/P9U7PFv

conservation element

compound conservation element (CCE): $\square$ BCEF defines the control volume CE and the control surface $S(\mathrm{CE})$



space-time

$$\oint_{S(\mathrm{CE}_\pm)} \mathbf{h}^* \cdot d\hat{\sigma} = 0$$

$\mathbf{h}^*$ is the approximation of $\mathbf{h}$ by using solution elements

basic conservation element (BCE):

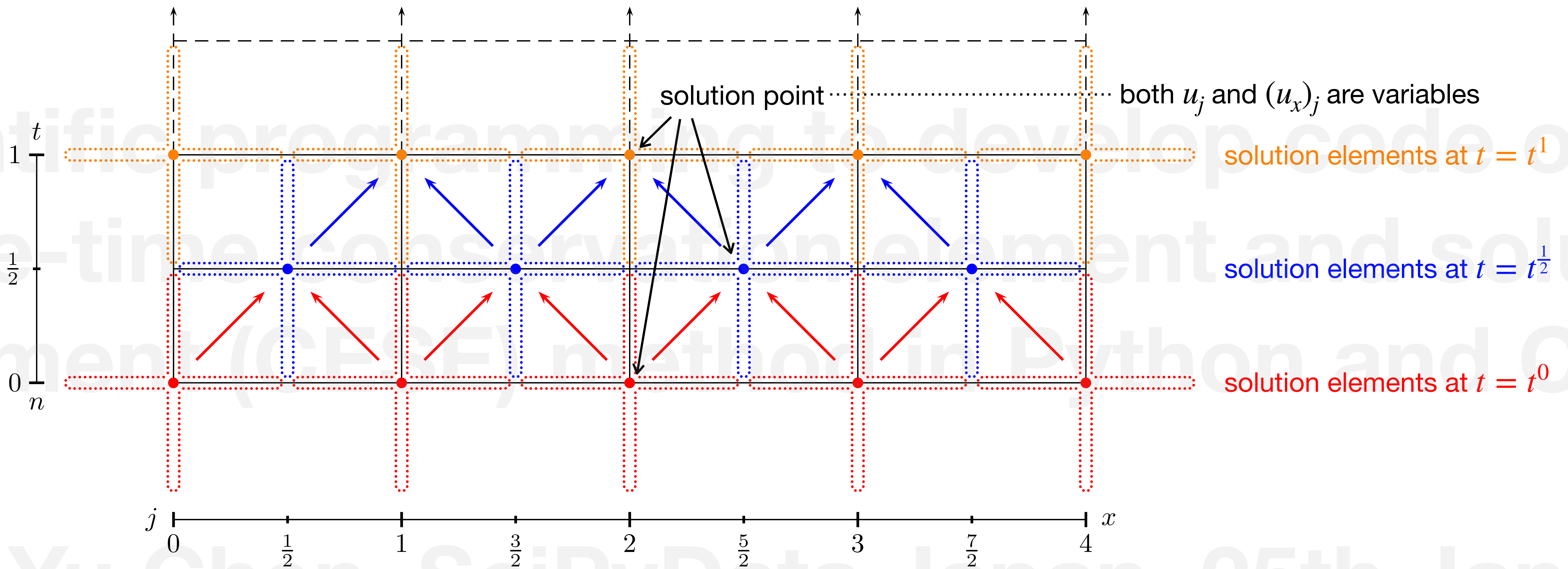
$\square$ ABCD defines the control volume CE_ and the control surface S(CE_)

$\square$ ADEF defines the control volume CE_+ and the control surface $S(\mathrm{CE}_+)$

join #solvcon on discord

https://discord.gg/P9U7PFv

solution element approximation



in the solution element, define:

$$u^*(x, t; j, n) = u_j^n + (u_x)_j^n (x - x_j) + (u_t)_j^n (t - t^n)$$

$$f^*(x, t; j, n) = f_j^n + (f_x)_j^n (x - x_j) + (f_t)_j^n (t - t^n)$$

write $\mathbf{h}^*(x, t; j, n)$ using only the variables defined on the solution points $u_j, (u_x)_j, f_j, (f_u)_j$:

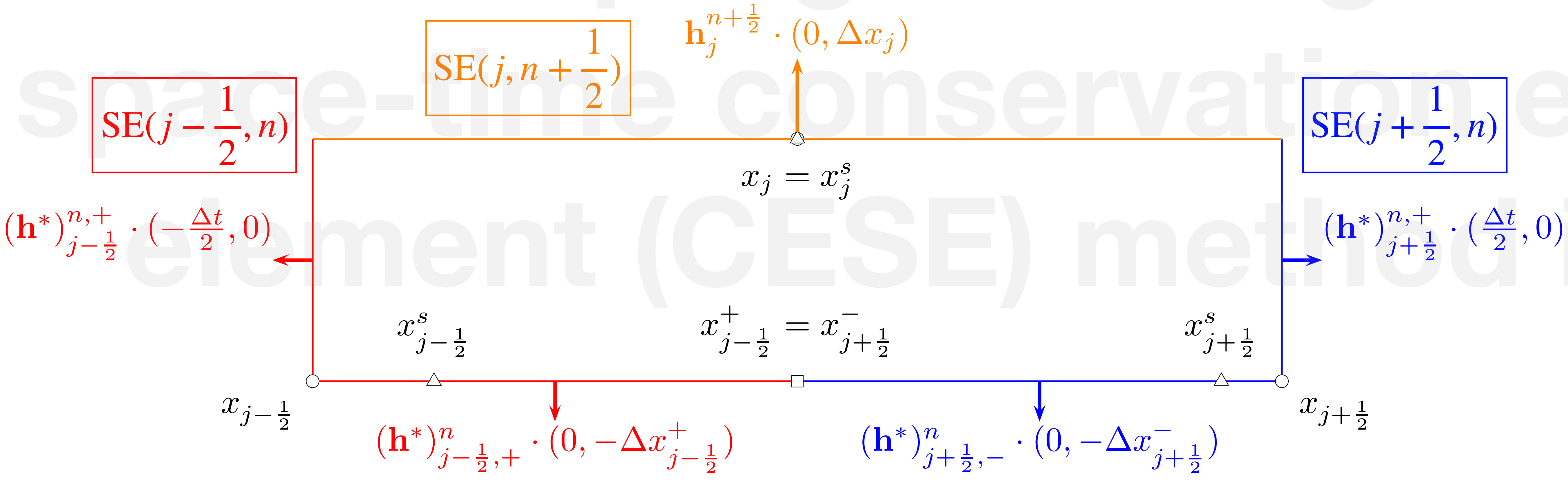
$$f^*(x, t; j, n) = f_j^n + (f_u)_j^n (u_x)_j^n \left[(x - x_j) - (f_u)_j^n (t - t^n) \right]$$

$$u^*(x, t; j, n) = u_j^n + (u_x)_j^n \left[(x - x_j) - (f_u)_j^n (t - t^n) \right]$$

flux conservation

the half-step time-marching formula can be derived by using the space-time flux conservation

$$\oint_{S(\mathrm{CE})} \mathbf{h}^* \cdot d\hat{\sigma} = 0 \text{ around } \mathrm{CE}(j - \frac{1}{2}, n) \text{ and } \mathrm{CE}(j + \frac{1}{2}, n)$$



recall

$$f^*(x, t; j, n) = f_j^n + (f_u)_j^n (u_x)_j^n \Big[(x - x_j^s) - (f_u)_j^n (t - t^n) \Big]$$

$$u^*(x, t; j, n) = u_j^n + (u_x)_j^n \Big[(x - x_j^s) - (f_u)_j^n (t - t^n) \Big]$$

use $\mathrm{SE}(j, n + \frac{1}{2})$, $\mathrm{SE}(j - \frac{1}{2}, n)$, $\mathrm{SE}(j + \frac{1}{2}, n)$ and $\Delta x_j = \Delta x_{j-\frac{1}{2}}^+ + \Delta x_{j+\frac{1}{2}}^-$ to obtain:

$$u_j^{n+\frac{1}{2}} = \frac{1}{\Delta x_j} \left\{ (u^*)_{j-\frac{1}{2},+}^n \Delta x_{j-\frac{1}{2}}^+ + (u^*)_{j+\frac{1}{2},-}^n \Delta x_{j+\frac{1}{2}}^- + \frac{\Delta t}{2} \left[(f^*)_{j-\frac{1}{2}}^{n,+} - (f^*)_{j+\frac{1}{2}}^{n,+} \right] \right\}$$

join #solvcon on discord

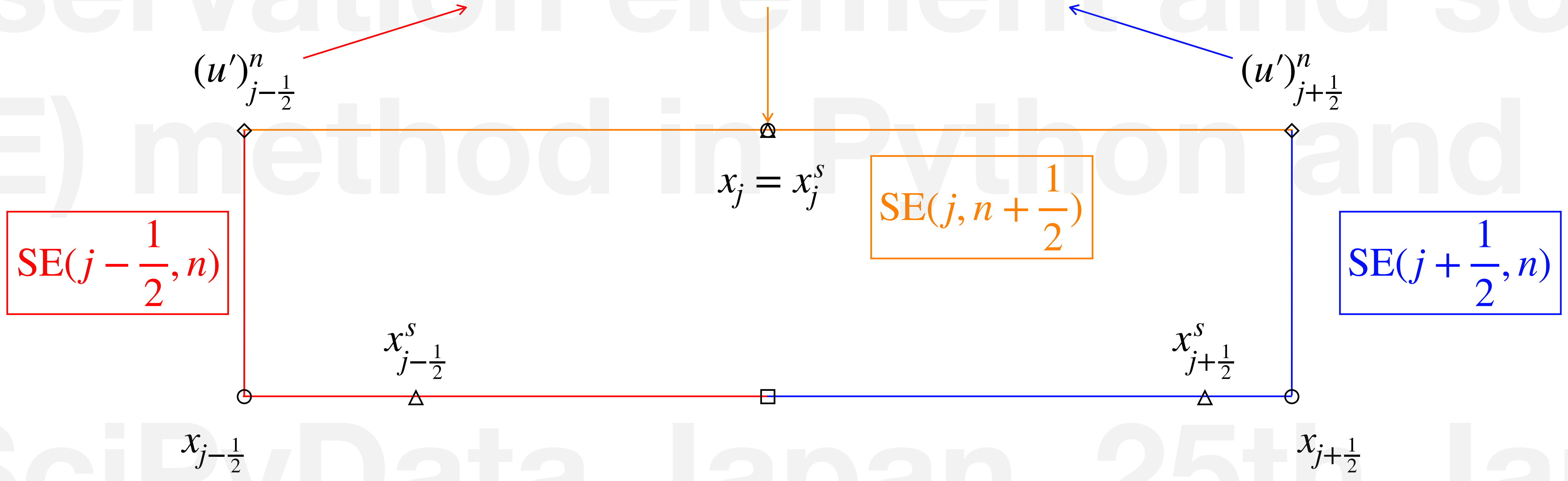
https://discord.gg/P9U7PFv

calculate gradient

the c scheme uses the central differencing to approximate $(u_x)_j^{n+\frac{1}{2}}$

Note: both u and u_x are unknowns

$$(u_x)_j^{n+\frac{1}{2}} = \frac{(u')_{j+\frac{1}{2}}^n - (u')_{j-\frac{1}{2}}^n}{\Delta x_j}$$



$(u')_{j\pm\frac{1}{2}}^n \overset{\text{def}}{=} u_{j\pm\frac{1}{2}}^n + (u_x)_{j\pm\frac{1}{2}}^n \left[x_{j\pm\frac{1}{2}} - x_{j\pm\frac{1}{2}}^s - (f_u)_{j\pm\frac{1}{2}}^n \frac{\Delta t}{2}\right]$ is the first-order Taylor expansion with respect to $\mathrm{SE}(j \pm \frac{1}{2}, n)$

Note: This is the treatment of the c scheme. It is developed based on the inviscid a scheme and the viscous a-ε scheme. The c scheme is also the foundation of the CFL-insensitive c-τ scheme.

join #solvcon on discord

https://discord.gg/P9U7PFv

remarks on gradient approximation

- the derivation to determine gradient looks magical and less rigorous
  - it is an alternate description of the a-ε scheme when $\varepsilon = 0.5$
  - the special case is called the c scheme
  - the c scheme is extended to the c-τ scheme, which is CFL-insensitive
- the central differencing only work for linear equation
  - for non-linear equation, a re-weighting function is used

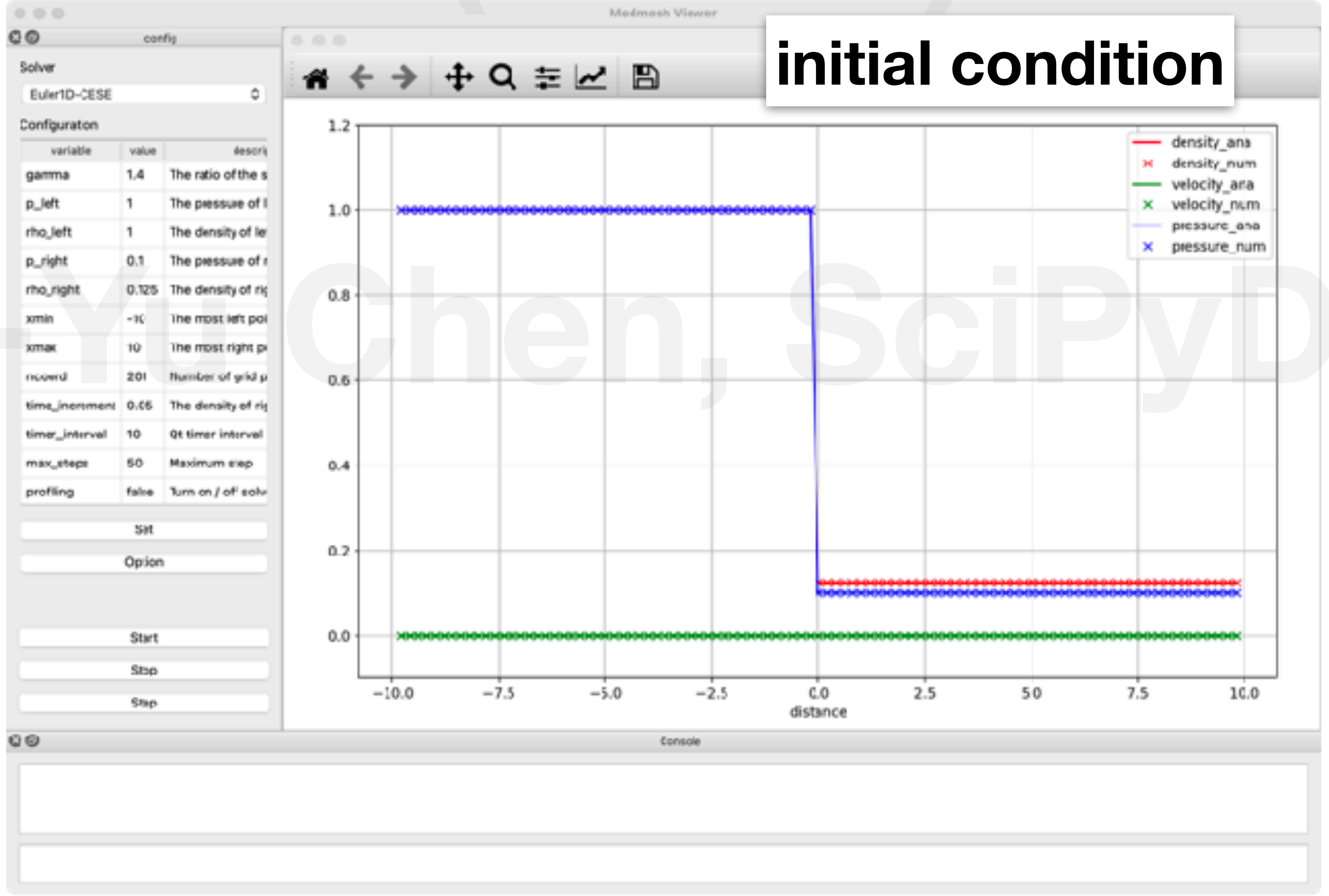
GUI integrating everything

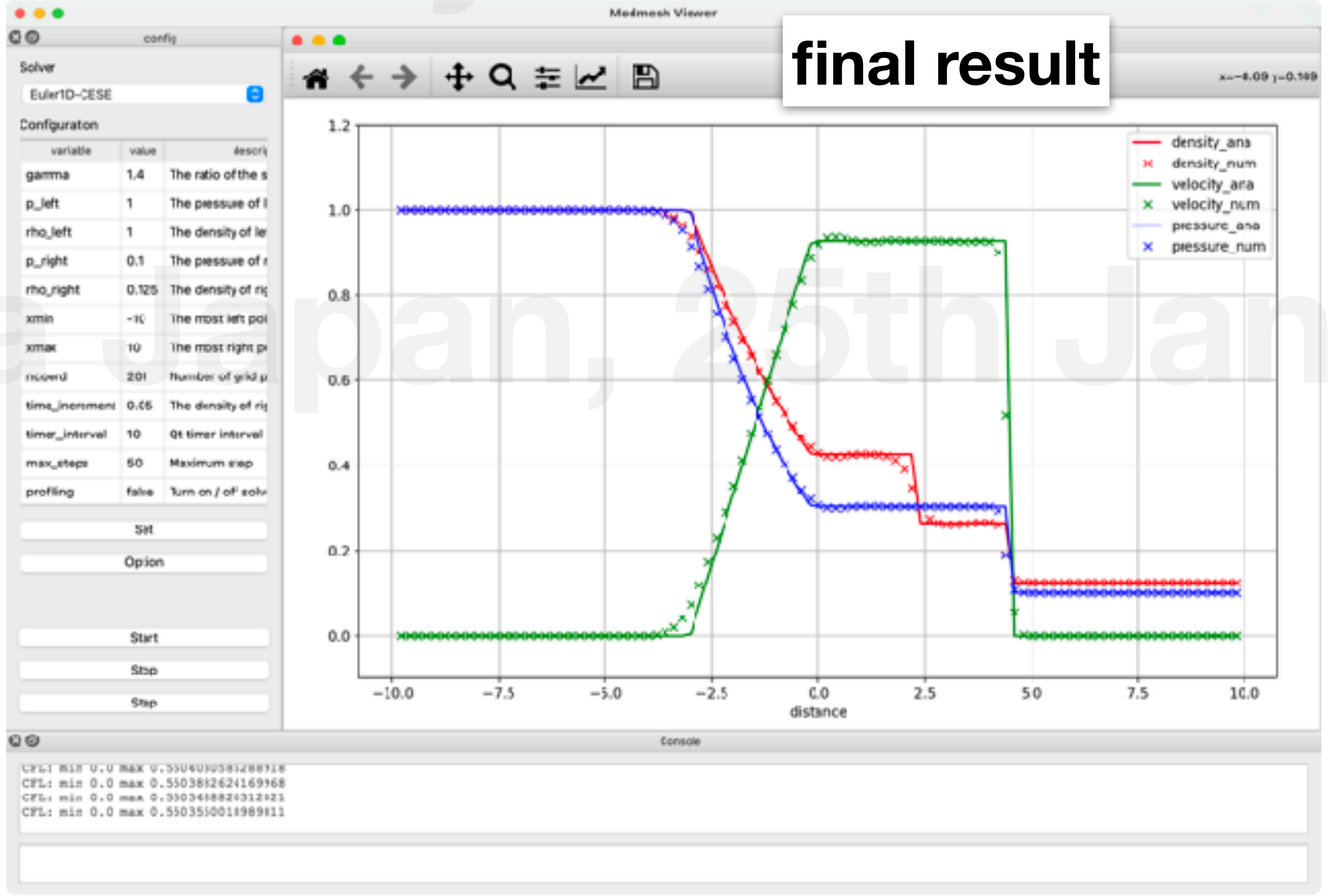
- clean numerical method

$$u_j^{n+\frac{1}{2}} = \frac{1}{\Delta x_j}\left\{(u^*)^n_{j-\frac{1}{2},+}\Delta x^+_{j-\frac{1}{2}} + (u^*)^n_{j+\frac{1}{2},-}\Delta x^-_{j+\frac{1}{2}} + \frac{\Delta t}{2}\left[(f^*)^{n,+}_{j-\frac{1}{2}} - (f^*)^{n,+}_{j+\frac{1}{2}}\right]\right\}$$

$$(u_x)_j^{n+\frac{1}{2}} = \frac{(u')^n_{j+\frac{1}{2}} - (u')^n_{j-\frac{1}{2}}}{\Delta x_j}$$

- 1D example: Sod's shock tube problem
- 2D problems in the pipeline





join #solvcon on discord

https://discord.gg/P9U7PFv

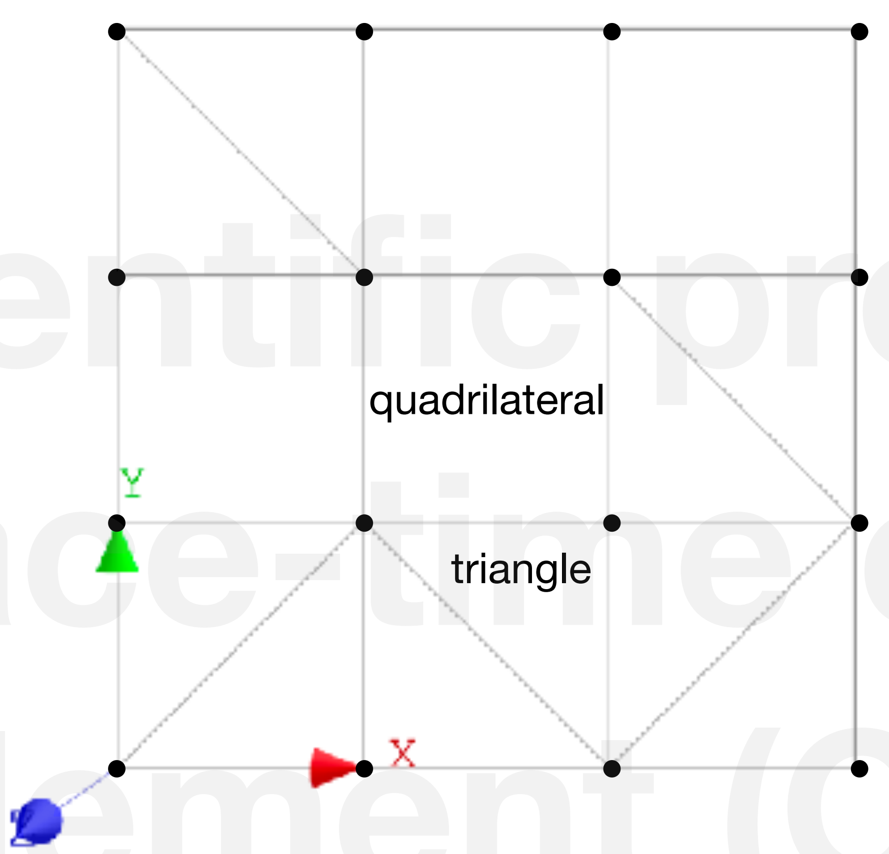
numerical and code clarity

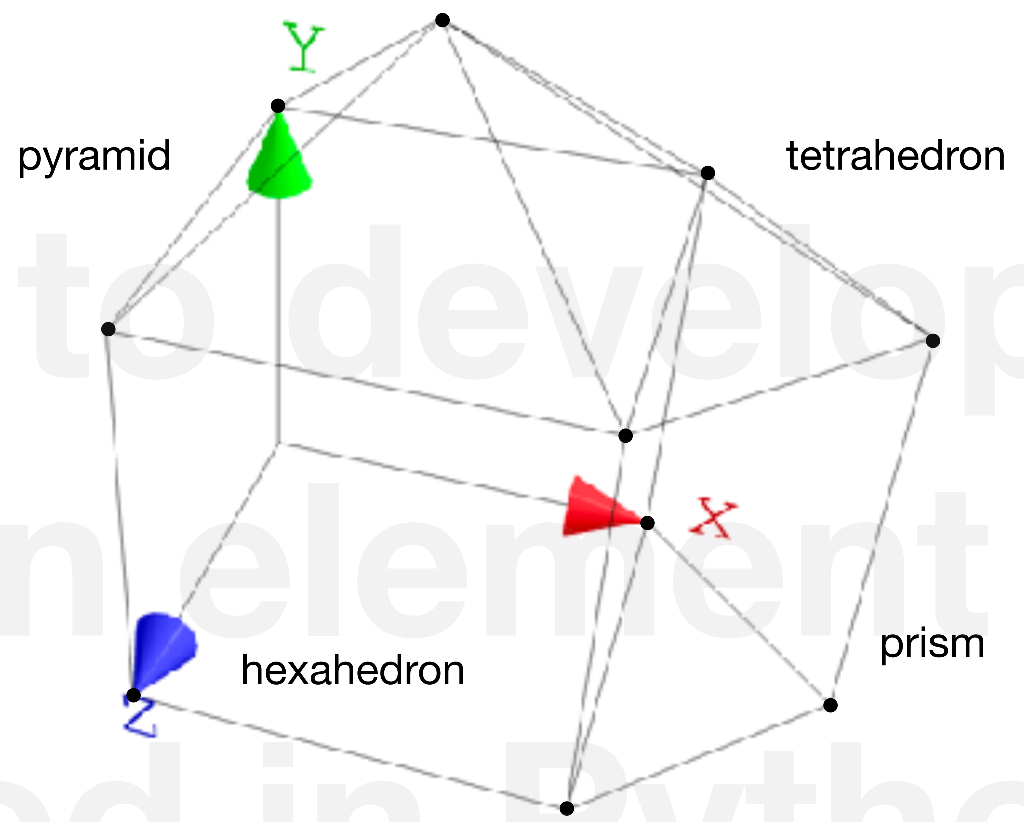
- do I make it?
  - I do have open-source code that follows the equations closely
- for scientific and engineering applications, results and speed matter more
  - clarity is subjective, varies by applications and teams
  - my production code (not this side project of the CESE method) is much uglier than my open-source code
- the value is ease of understanding
  - community members of zero background can contribute using spare time

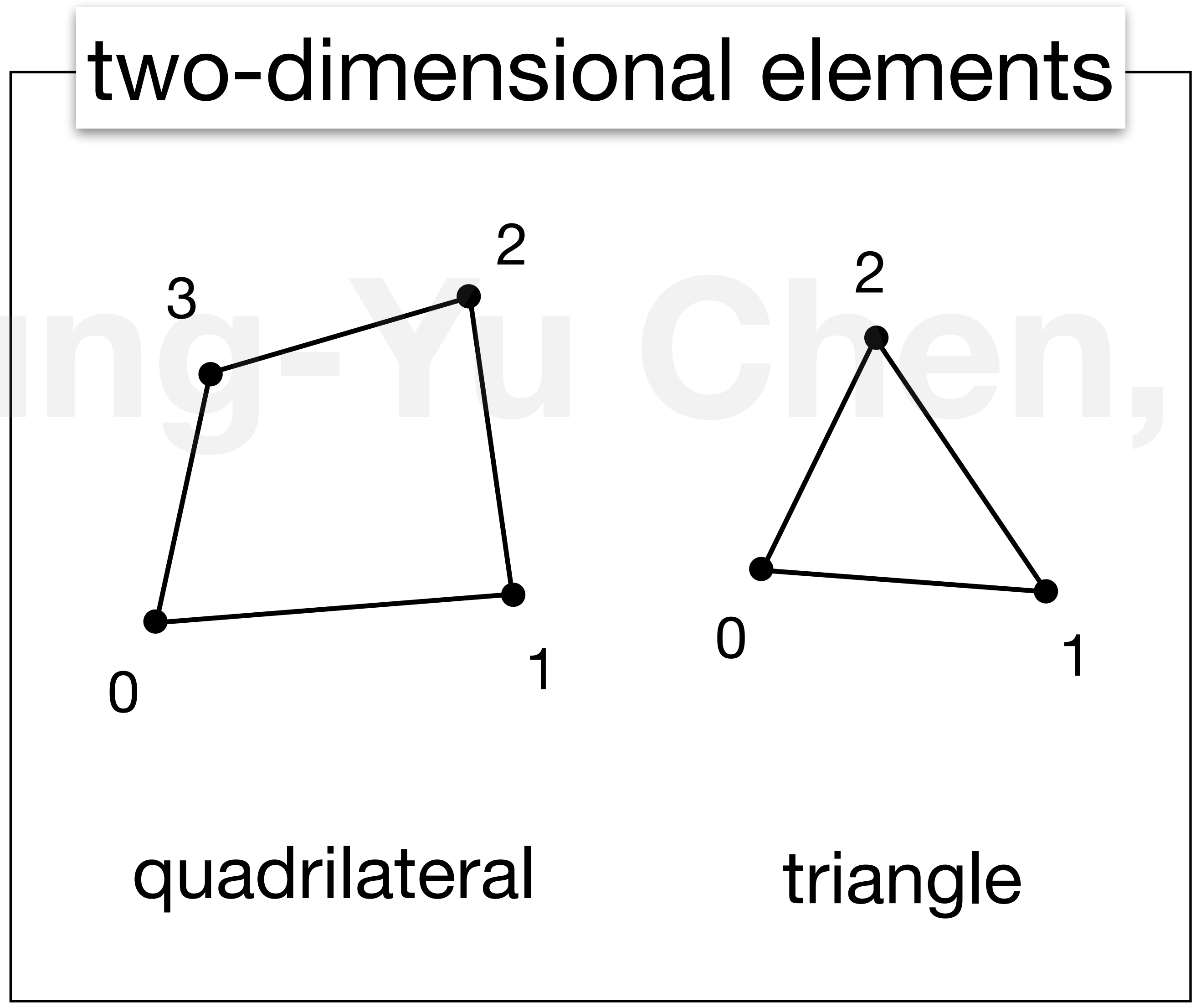
join #solvcon on discord

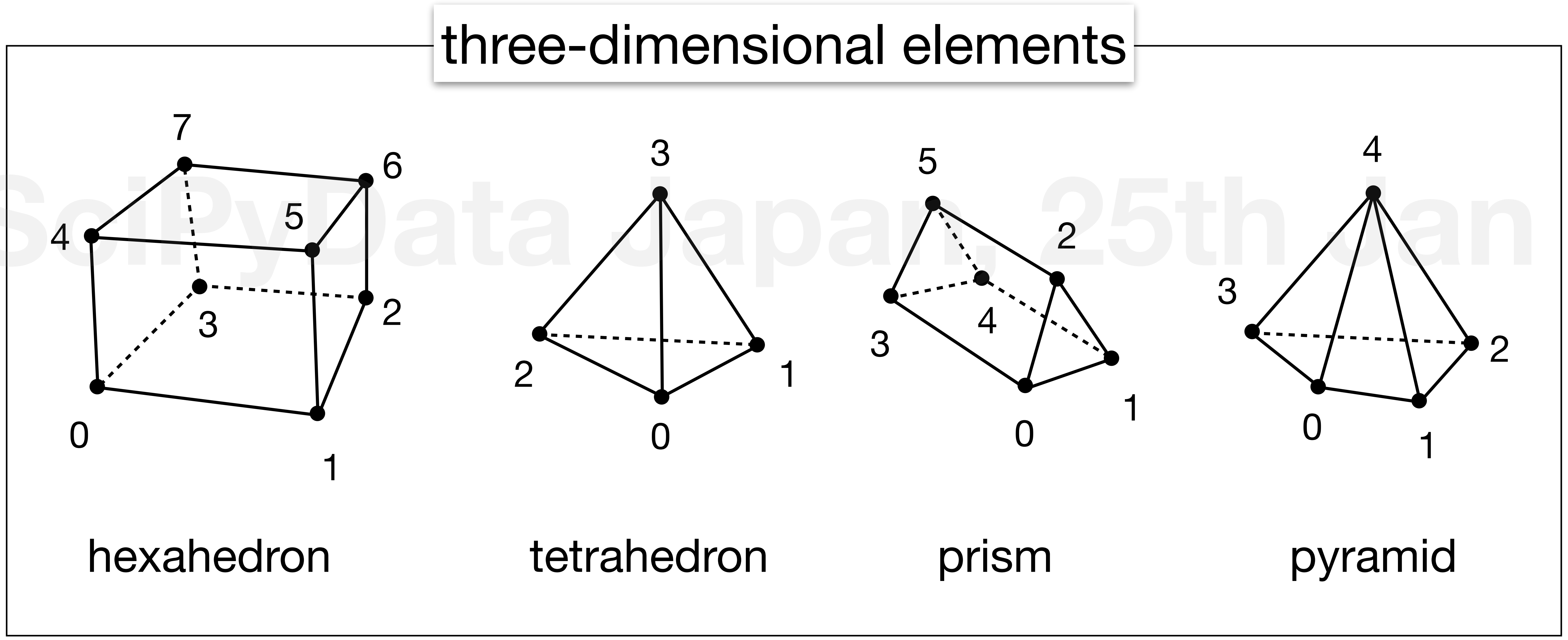
https://discord.gg/P9U7PFv

mixed-shape unstructured meshes





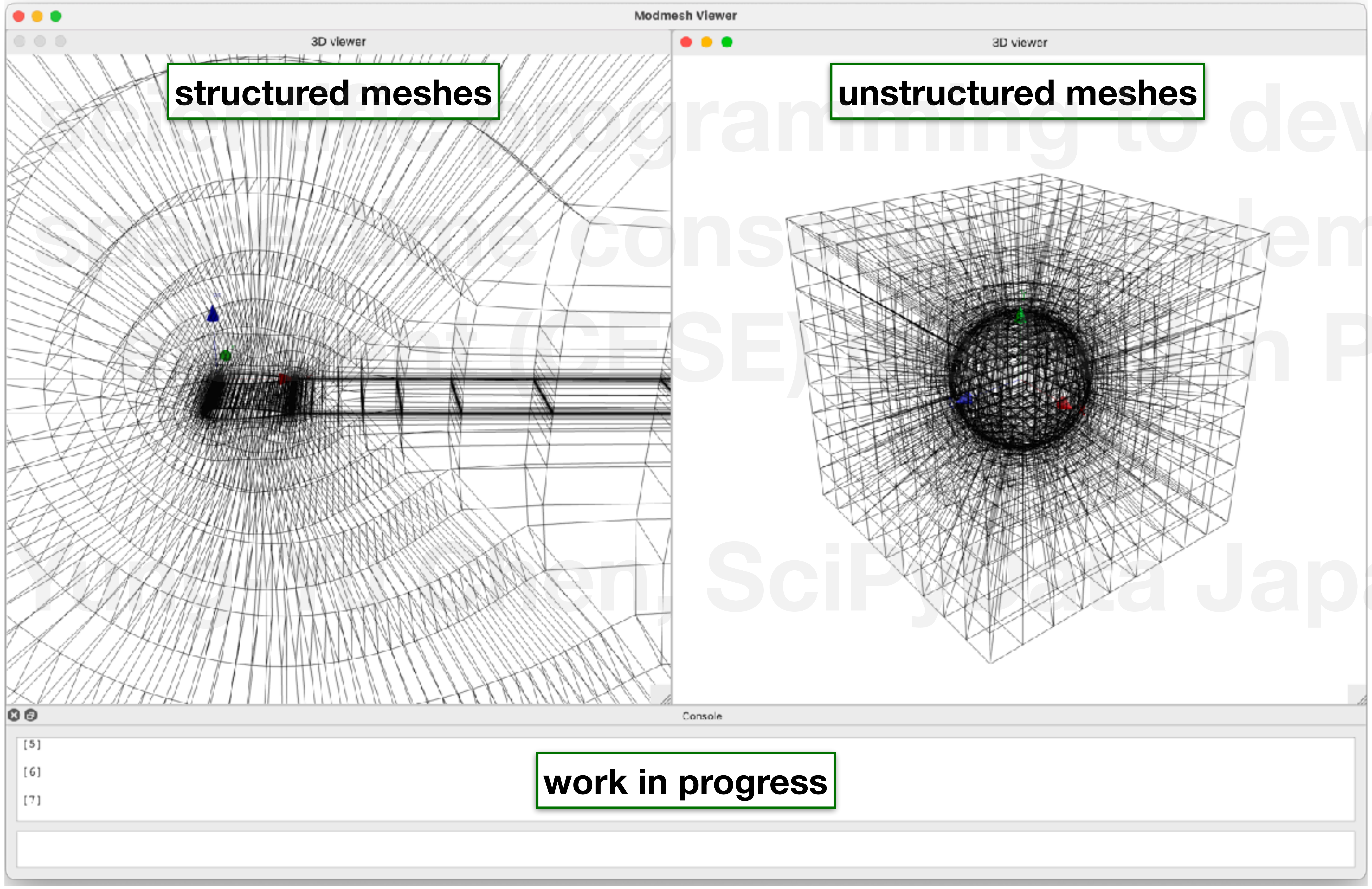




join #solvcon on discord

https://discord.gg/P9U7PFv

complex geometry for 2/3D solver

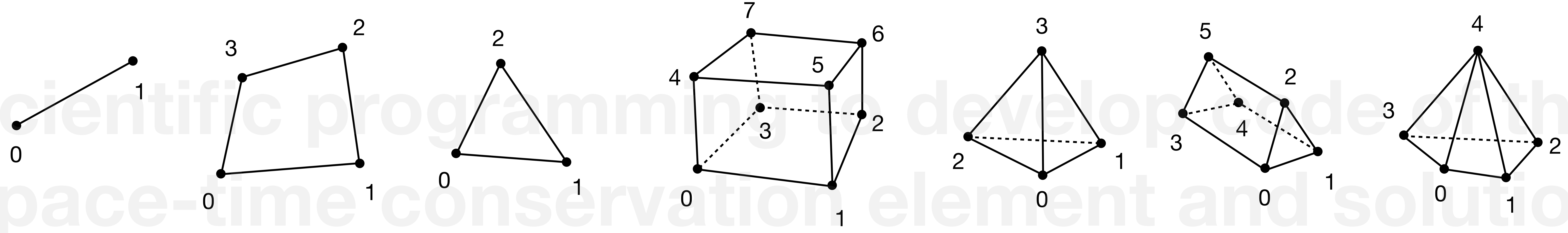


- C++ code for Qt3D
  - wrapped to Python using pybind11
- Qt6 and PySide6 for the interactive GUI
  - interoperation between Shiboken6 (PySide6) and pybind11

join #solvcon on discord

https://discord.gg/P9U7PFv

connectivity



line quadrilateral triangle hexahedron tetrahedron prism pyramid

name	type id	dimension	# node	# edge	# surface	# face
point	1	0	1	n/a	n/a	n/a
line	2	1	2	n/a	n/a	n/a
quadrilateral	3	2	4	4	n/a	4
triangle	4	2	3	3	n/a	3
hexahedron	5	3	8	12	6	6
tetrahedron	6	3	4	6	4	4
prism	7	3	6	9	5	5
pyramid	8	3	5	8	5	5

- face connects two cells
  - line in 2D
  - surface in 3D

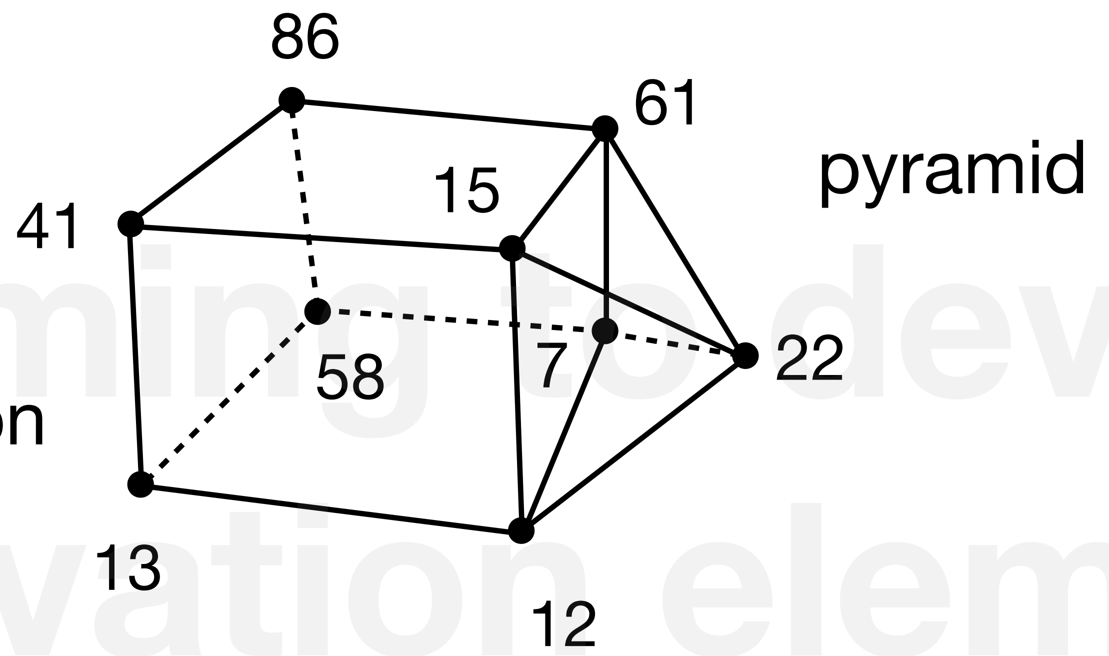
```
#define MM_DECL_SWITCH_CELL_TYPE(TYPE, NDIM, NNODE, NEDGE, NSURFACE) \
    case TYPE: return CellType(TYPE, NDIM, NNODE, NEDGE, NSURFACE); break;
    switch (id)
    {
        //                        id, ndim, nnode, nedge, nsurface
        MM_DECL_SWITCH_CELL_TYPE(  0,    0,     0,     0,        0 ) // non-type
        MM_DECL_SWITCH_CELL_TYPE(  1,    0,     1,     0,        0 ) // point/node/vertex
        MM_DECL_SWITCH_CELL_TYPE(  2,    1,     2,     0,        0 ) // line/edge
        MM_DECL_SWITCH_CELL_TYPE(  3,    2,     4,     4,        0 ) // quadrilateral
        MM_DECL_SWITCH_CELL_TYPE(  4,    2,     3,     3,        0 ) // triangle
        MM_DECL_SWITCH_CELL_TYPE(  5,    3,     8,    12,        6 ) // hexahedron/brick
        MM_DECL_SWITCH_CELL_TYPE(  6,    3,     4,     6,        4 ) // tetrahedron
        MM_DECL_SWITCH_CELL_TYPE(  7,    3,     6,     9,        5 ) // prism
        MM_DECL_SWITCH_CELL_TYPE(  8,    3,     5,     8,        5 ) // pyramid
        default: return CellType{}; break;
    }
#undef MM_DECL_SWITCH_CELL_TYPE
```

join #solvcon on discord

https://discord.gg/P9U7PFv

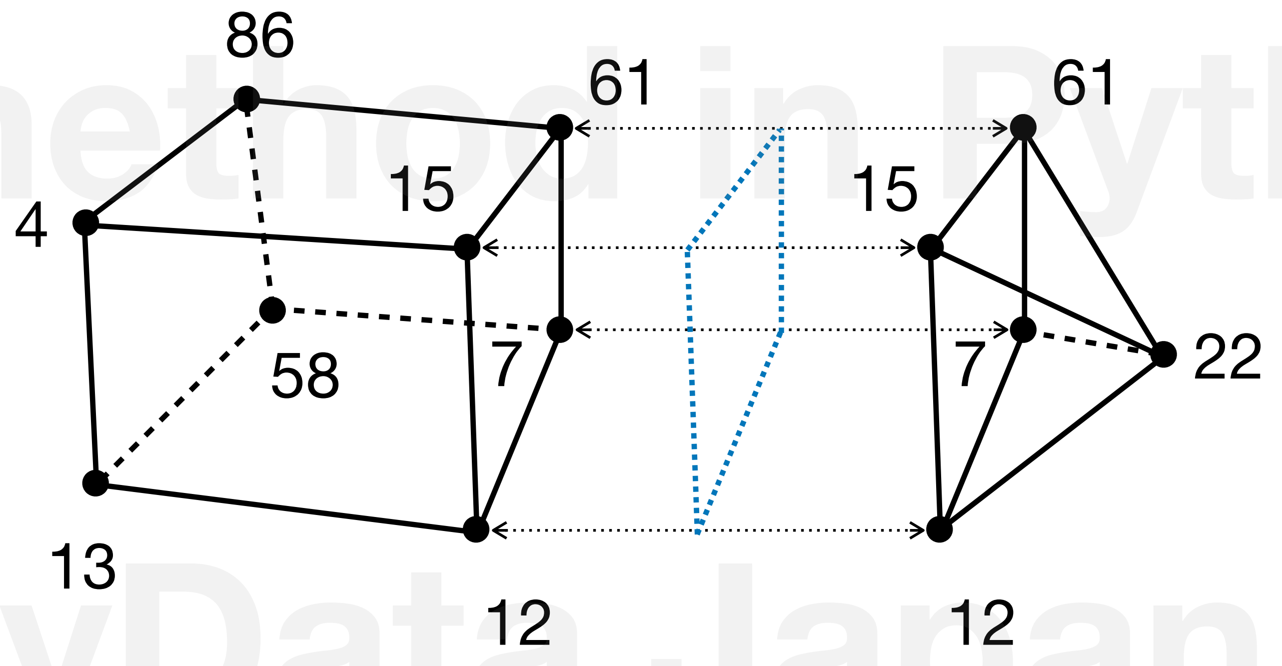
cells connect by sharing face

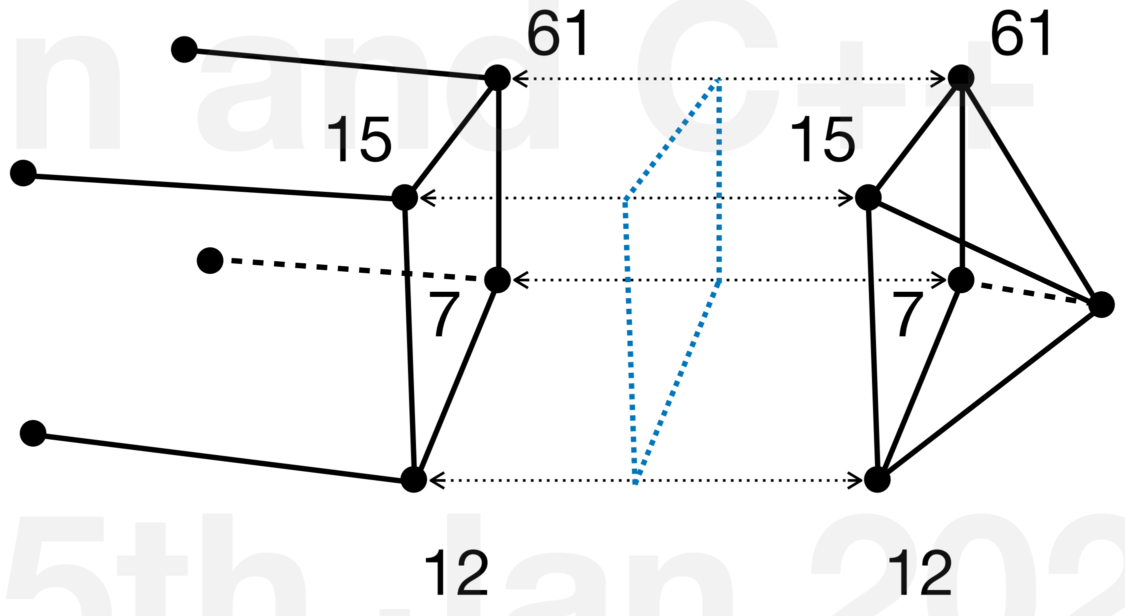
global indices



# of node

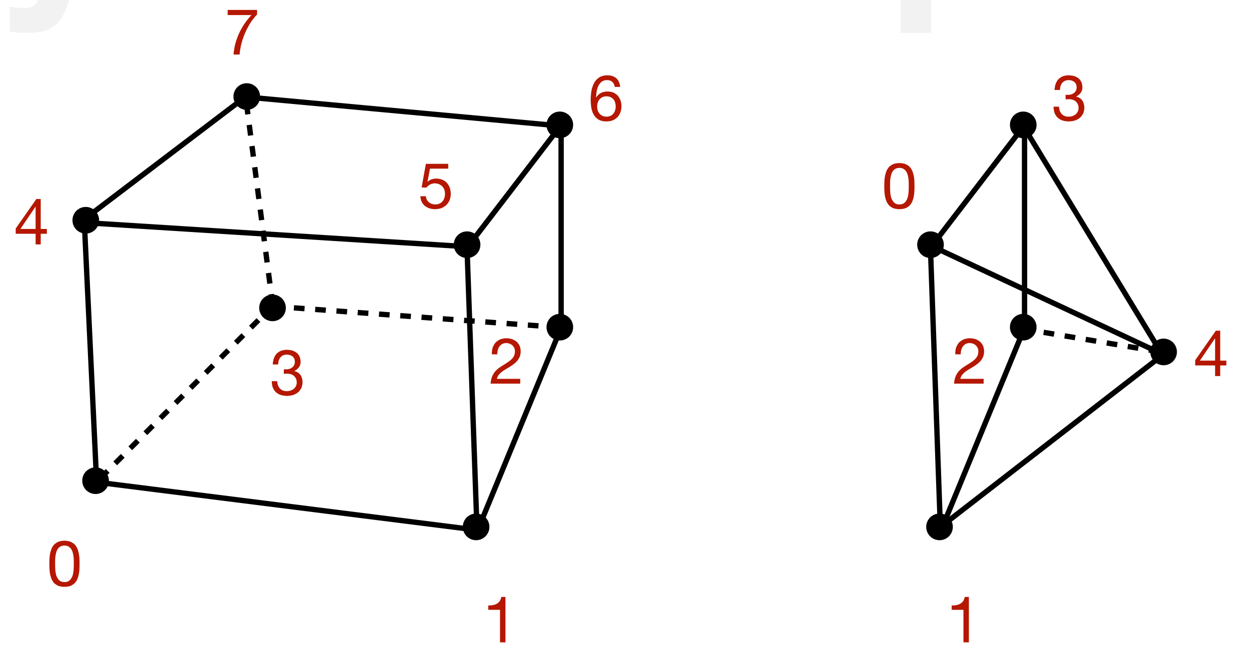
	# of node								
hexahedron	8	13	12	7	58	4	15	61	86
pyramid	5	15	12	7	61	22	-	-	-
		0	1	2	3	4	5	6	7





2 cells share a face

local indices

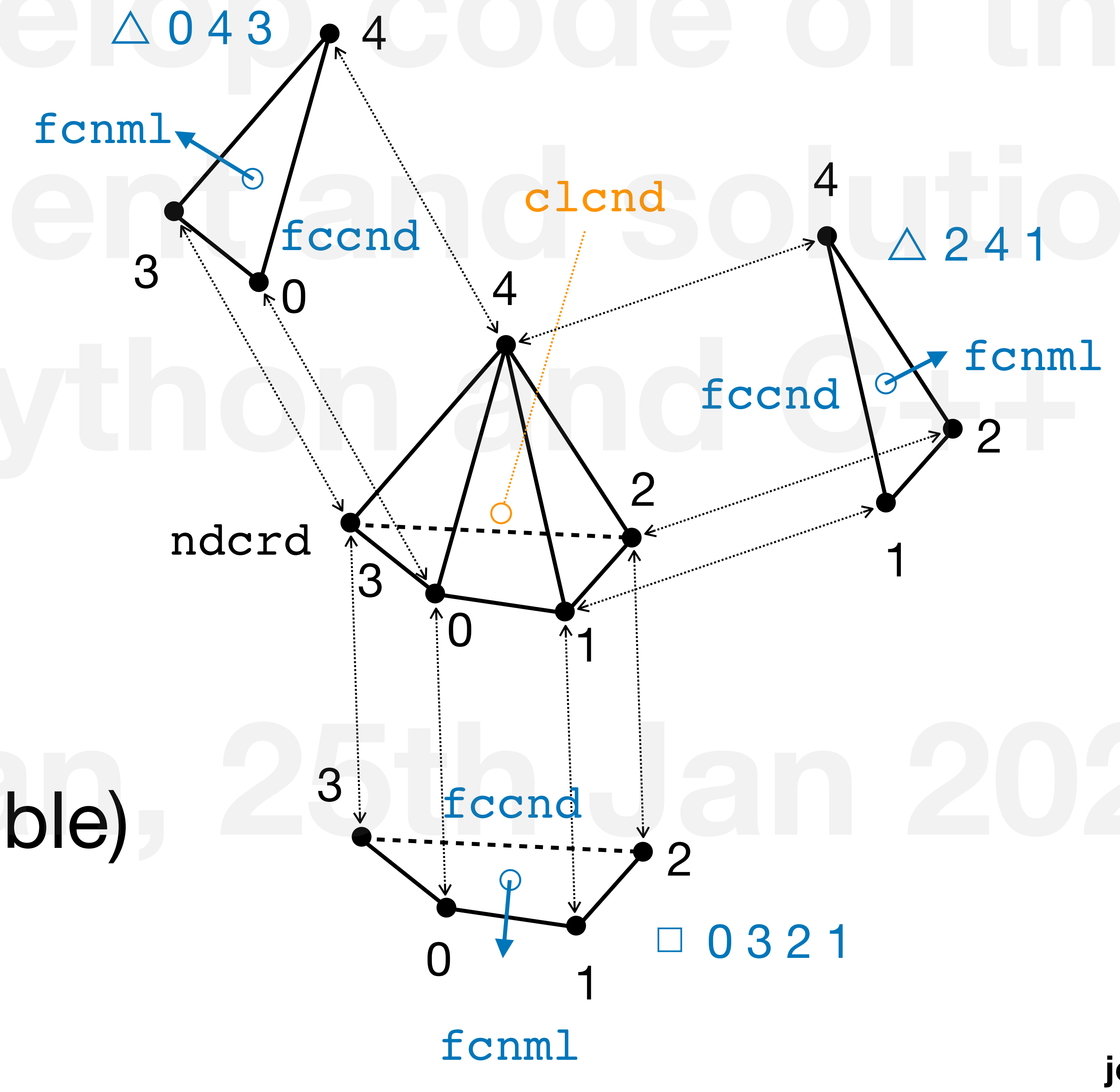


join #solvcon on discord

https://discord.gg/P9U7PFv

metric data

metrics	name in code	length (shape[0])	shape[1]	data source
node coordinate	`ndcrd`	nnode	ndim (2/3)	input
face center coordinate	`fccnd`	nface	ndim (2/3)	derived
face unit normal vector	`fcnml`	nface	ndim (2/3)	derived
face area	`fcara`	nface	n/a	derived
cell center coordinate	`clcnd`	ncell	ndim (2/3)	derived
cell volume	`clvol`	ncell	n/a	derived



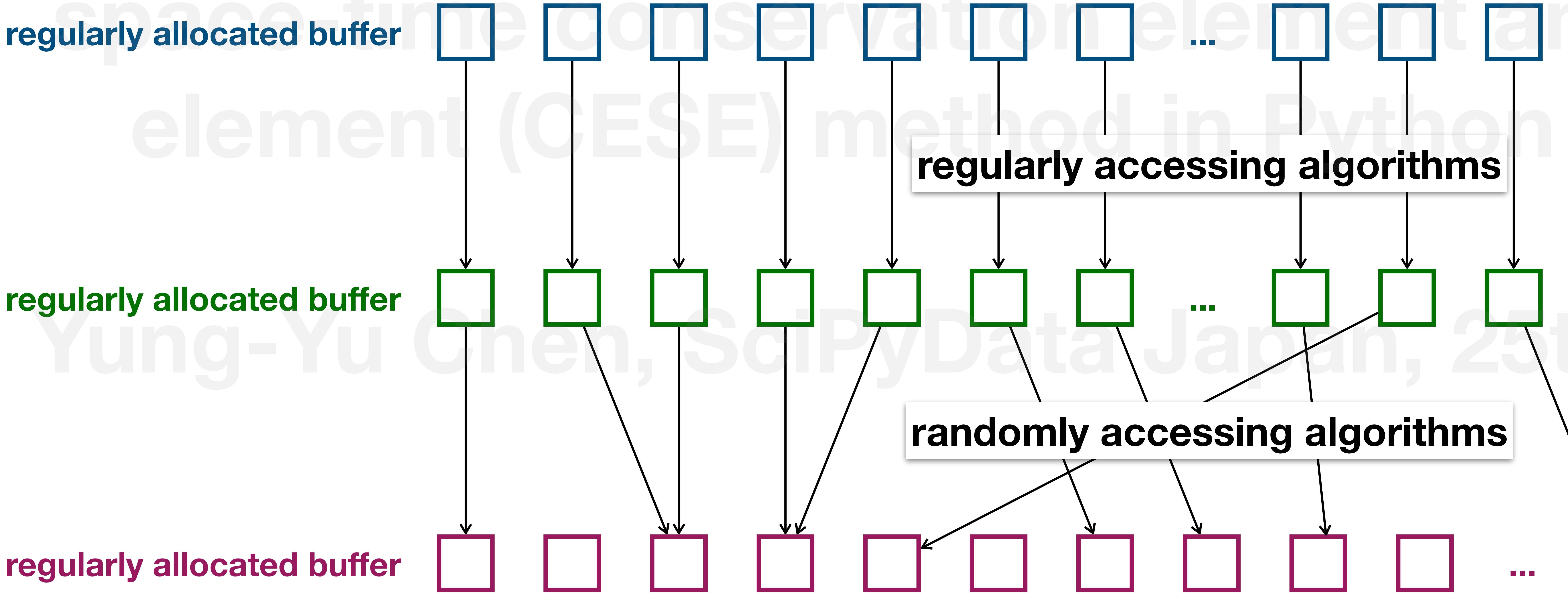
- face and cell center may use centroid (other center is possible)
- surface normal of a triangle can be determined by cross product of two edge vectors
- normal of a quadrilateral needs to be averaged

join #solvcon on discord

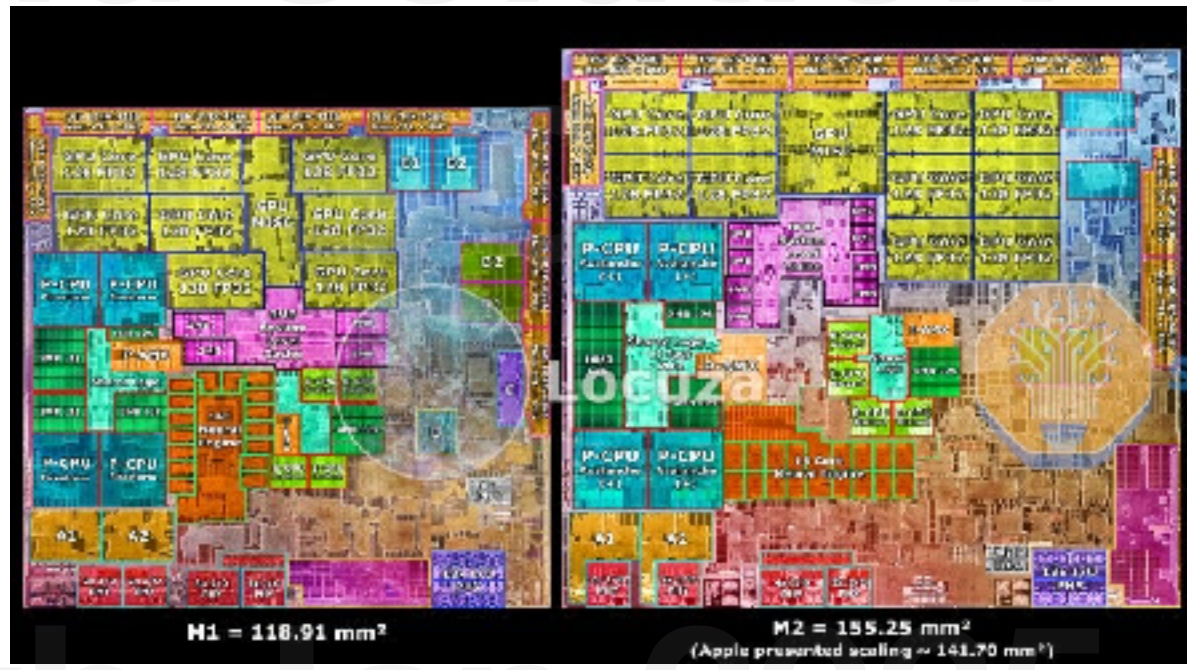
https://discord.gg/P9U7PFv

make array library

the data structure for cache optimization, data parallelism, SIMD, GPU, all the techniques of high-performance computing (HPC)



Apple Silicons



quoted from SemiAnalysis
https://www.semianalysis.com/p/apple-m2-die-shot-and-architecture

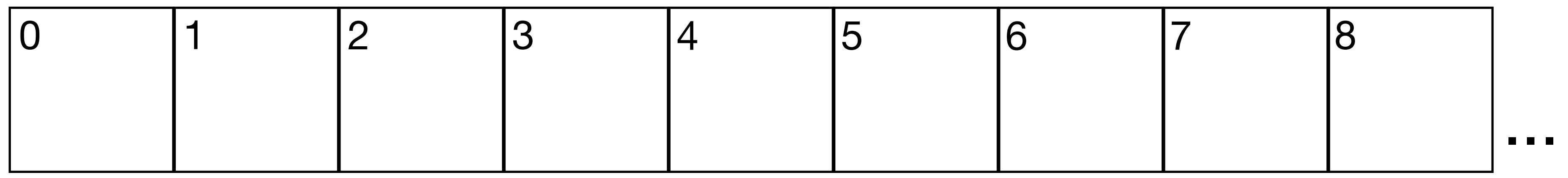
join #solvcon on discord

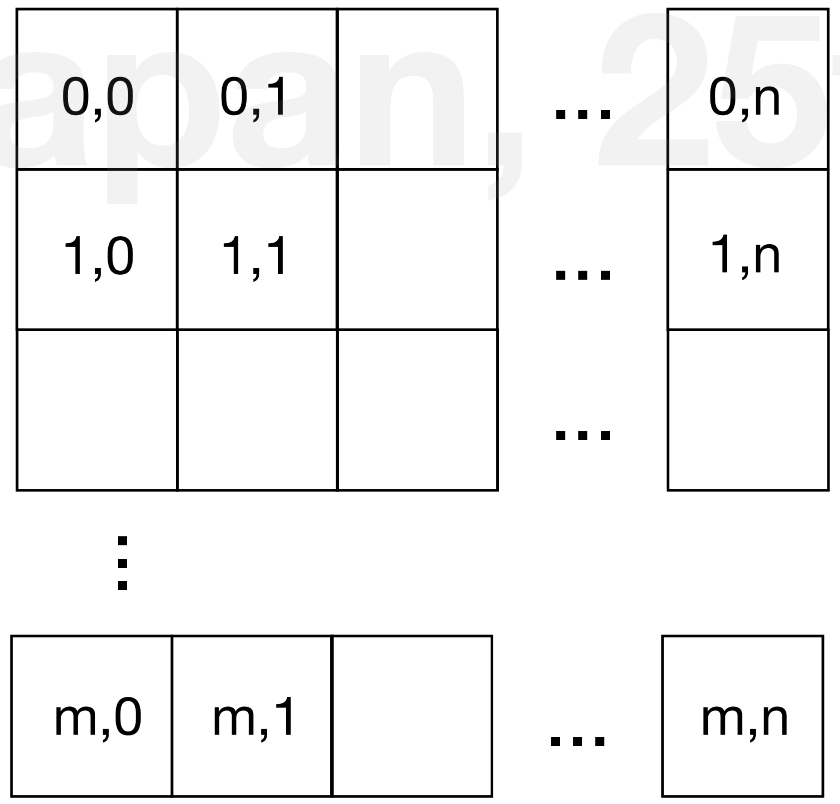
https://discord.gg/P9U7PFv

multi-dimensional array

- `SimpleArray` is a *class template*
  - holds a contiguous memory buffer
  - provides multi-dimensional accessors to its elements
- designed for fundamental types; might be used for POD types (untested)

1D array





$m \times n$ array

join #solvcon on discord

https://discord.gg/P9U7PFv

`SimpleArray` is not C++ vector

`SimpleArray`

`SimpleArray` is fixed size

- only allocate memory on construction

multi-dimensional access:

`operator()`

`std::vector`

`std::vector` is variable size

- buffer may be invalidated
- implicit memory allocation (reallocation)

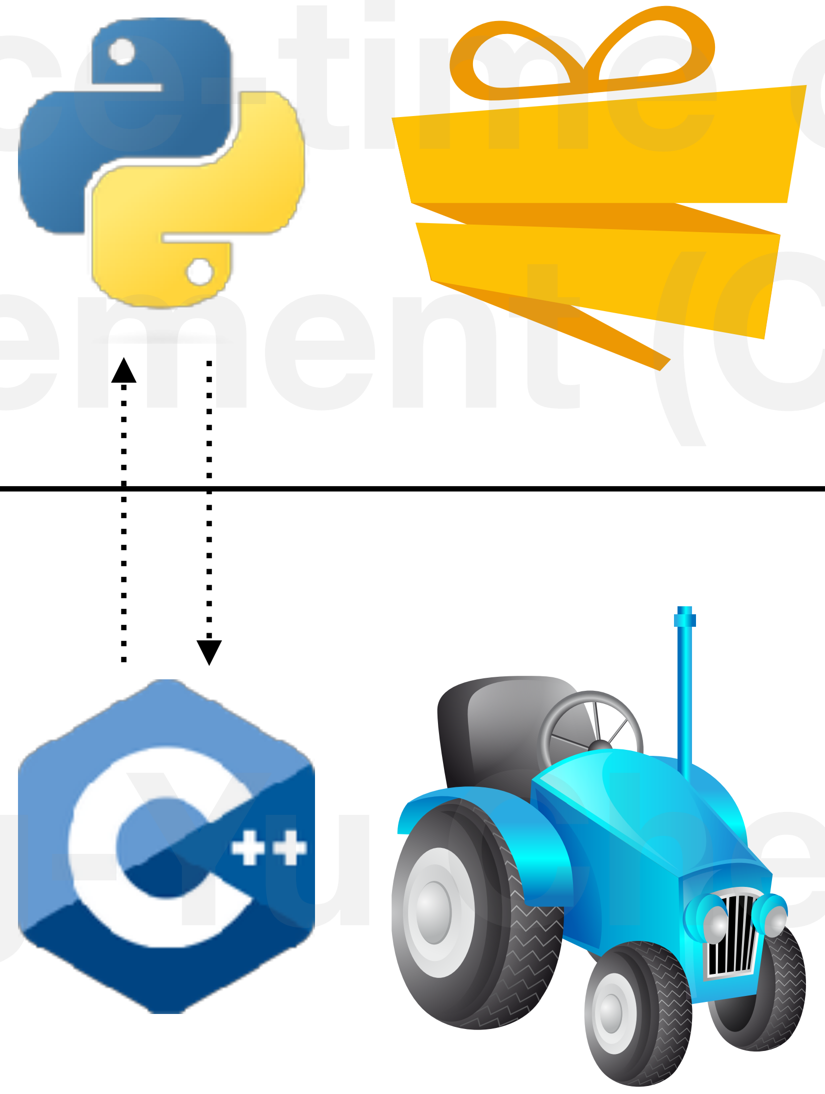
one-dimensional access:

`operator[]`

join #solvcon on discord

https://discord.gg/P9U7PFv

manage lifecycle



the Python wrapping layer bridge the object lifecycle between `PyObject` and the C++ engine

the C++ engine defines a management framework

join #solvcon on discord

https://discord.gg/P9U7PFv

management framework

buffer

```
class ConcreteBuffer
    : public std::enable_shared_from_this<ConcreteBuffer>
    , public BufferBase<ConcreteBuffer>
{
private:
    using data_deleter_type = detail::ConcreteBufferDataDeleter;
public:
    using remover_type = detail::ConcreteBufferRemover;
    using unique_ptr_type = std::unique_ptr<int8_t, data_deleter_type>;
private:

    static unique_ptr_type allocate(size_t nbytes)
    {
        unique_ptr_type ret(nullptr, data_deleter_type());
        if (0 != nbytes)
        {
            ret = unique_ptr_type(new int8_t[nbytes], data_deleter_type());
        }
        return ret;
    }

    size_t m_nbytes;
    unique_ptr_type m_data;
};
```

use unique pointer to manage lifecycle

deleter

```
struct ConcreteBufferDataDeleter
{
    using remover_type = ConcreteBufferRemover;

    ConcreteBufferDataDeleter(ConcreteBufferDataDeleter const &) = delete;
    ConcreteBufferDataDeleter & operator=(ConcreteBufferDataDeleter const &) = delete;

    ConcreteBufferDataDeleter() = default;
    ConcreteBufferDataDeleter(ConcreteBufferDataDeleter &&) = default;
    ConcreteBufferDataDeleter & operator=(ConcreteBufferDataDeleter &&) = default;
    ~ConcreteBufferDataDeleter() = default;

    explicit ConcreteBufferDataDeleter(std::unique_ptr<remover_type> && remover_in)
      : remover(std::move(remover_in))
    {}

    void operator()(int8_t * p) const
    {
        if (!remover) { delete[] p; }

        else { (*remover)(p); }
    }

    std::unique_ptr<remover_type> remover{nullptr};
};
```

allocate by constructor: no remover use array new/delete

```
ConcreteBuffer(size_t nbytes
             , const ctor_passkey &)
    : BufferBase<ConcreteBuffer>()
    , m_nbytes(nbytes)
    , m_data(allocate(nbytes))
    { /* ... */ }
```

```
ConcreteBuffer(size_t nbytes, int8_t * data
             , std::unique_ptr<remover_type> && remover
             , const ctor_passkey &)
    : BufferBase<ConcreteBuffer>()
    , m_nbytes(nbytes)
    , m_data(data, data_deleter_type(std::move(remover)))
    { /* ... */ }
```

foreign pointer needs a remover to manage lifetime

what happens in Python stays in Python

standard remover follows the default behavior of the deleter

```
struct ConcreteBufferRemover
{
    virtual ~ConcreteBufferRemover() = default;
    virtual void operator()(int8_t * p) const { delete[] p; }
};
```

```
struct ConcreteBufferDataDeleter
{
    using remover_type = ConcreteBufferRemover;

    ConcreteBufferDataDeleter(ConcreteBufferDataDeleter const &) = delete;
    ConcreteBufferDataDeleter & operator=(ConcreteBufferDataDeleter const &) = delete;

    ConcreteBufferDataDeleter() = default;
    ConcreteBufferDataDeleter(ConcreteBufferDataDeleter &&) = default;
    ConcreteBufferDataDeleter & operator=(ConcreteBufferDataDeleter &&) = default;
    ~ConcreteBufferDataDeleter() = default;

    explicit ConcreteBufferDataDeleter(std::unique_ptr<remover_type> && remover_in)
      : remover(std::move(remover_in))
    {}

    void operator()(int8_t * p) const
    {
        if (!remover) { delete[] p; }

        else { (*remover)(p); }
    }

    std::unique_ptr<remover_type> remover{nullptr};
};
```

Python remover uses PyObject to manage lifecycle

```
struct ConcreteBufferNdarrayRemover : ConcreteBuffer::remover_type
{
    ConcreteBufferNdarrayRemover() = delete;
    explicit ConcreteBufferNdarrayRemover(pybind11::array arr_in)
        : ndarray(std::move(arr_in))
    {}

    void operator()(int8_t *) const override {}

    pybind11::array ndarray;
};
```

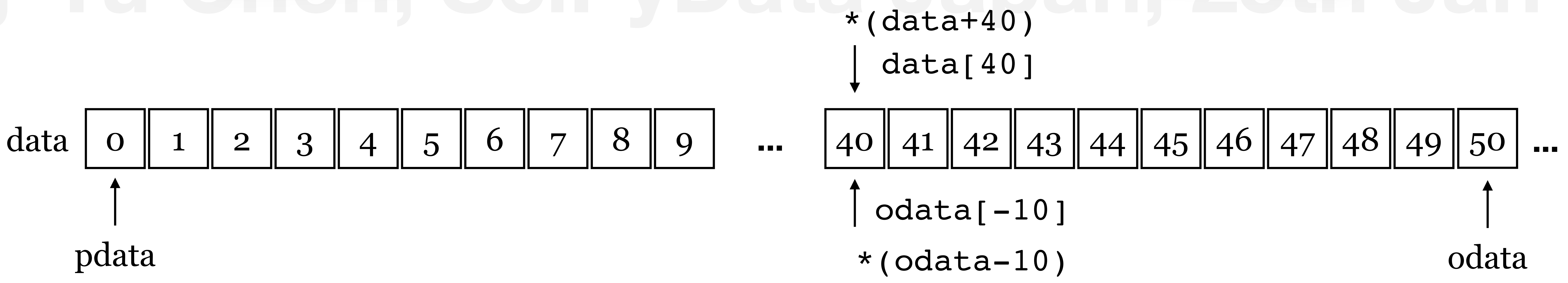
do nothing

Remover releases PyObject on destruction

ghost (negative) index

- ghost: elements indexed by negative integer
  - similar to the negative index for the POD array
- no overhead for normal arrays that start with 0 index


```
// C-style POD array.
int32_t data[100];
// Make a pointer to the head address of the array.
int32_t * pdata = data;
// Make another pointer to the 50-th element from the head of the array.
int32_t * odata = pdata + 50;
```



join #solvcon on discord
https://discord.gg/P9U7PFv

implement first-dimension ghost

keep ghost number and body address

```
template <typename T>
class SimpleArray
{
private:
    // Number of ghost elements
    size_t m_nghost = 0;
    // Starting address of non-ghost
    value_type * m_body = nullptr;
};
```

calculate body address

```
static T * calc_body(T * data, shape_type const & stride,
                     size_t nghost)
{
    if (nullptr == data || stride.empty() || 0 == nghost)
    {
        // Do nothing.
    }
    else
    {
        shape_type shape(stride.size(), 0);
        shape[0] = nghost;
        data += buffer_offset(stride, shape);
    }
    return data;
}
```

assign ghost and body information in constructors

```
SimpleArray(SimpleArray const & other)
    : m_buffer(other.m_buffer->clone())
    , m_shape(other.m_shape)
    , m_stride(other.m_stride)
    , m_nghost(other.m_nghost)
    , m_body(calc_body(m_buffer->data<T>(),
                       m_stride, other.m_nghost))
{}

SimpleArray(SimpleArray && other) noexcept
    : m_buffer(std::move(other.m_buffer))
    , m_shape(std::move(other.m_shape))
    , m_stride(std::move(other.m_stride))
    , m_nghost(other.m_nghost)
    , m_body(other.m_body)
{}
```

more to do

numerics

- numerical methods
  - ✅ the space-time CESE method
    - ✅ 1D Euler solver
    - 🚜 (work in progress) 2/3D Euler solver
    - 🪧 (to be planned) more equations
  - 🪧 (to be planned) FVM, FEM, etc.
- mixed-shape unstructured meshes
  - ✅ geometry data for consumption from solver
  - 🪧 (to be planned) create the spatial discretization (meshing)

computer

- 🚜 visualization (very basic)
  - 🪧 (to be planned) reasonable UI
- ✅ array operations and buffer management
  - 🚜 columnar arrays
- 🚜 profiling
  - 🪧 (to be planned) install in applications
- 🚜 parameter database
  - 🪧 (to be planned) install in applications
- ✅ build, test, and deploy
  - 🚜 continuously improvement

... and more

join #solvcon on discord

https://discord.gg/P9U7PFv

end

join #solvcon on discord

https://discord.gg/P9U7PFv