Zero-overhead Interfaces for High-performance Computing Libraries and Kernels

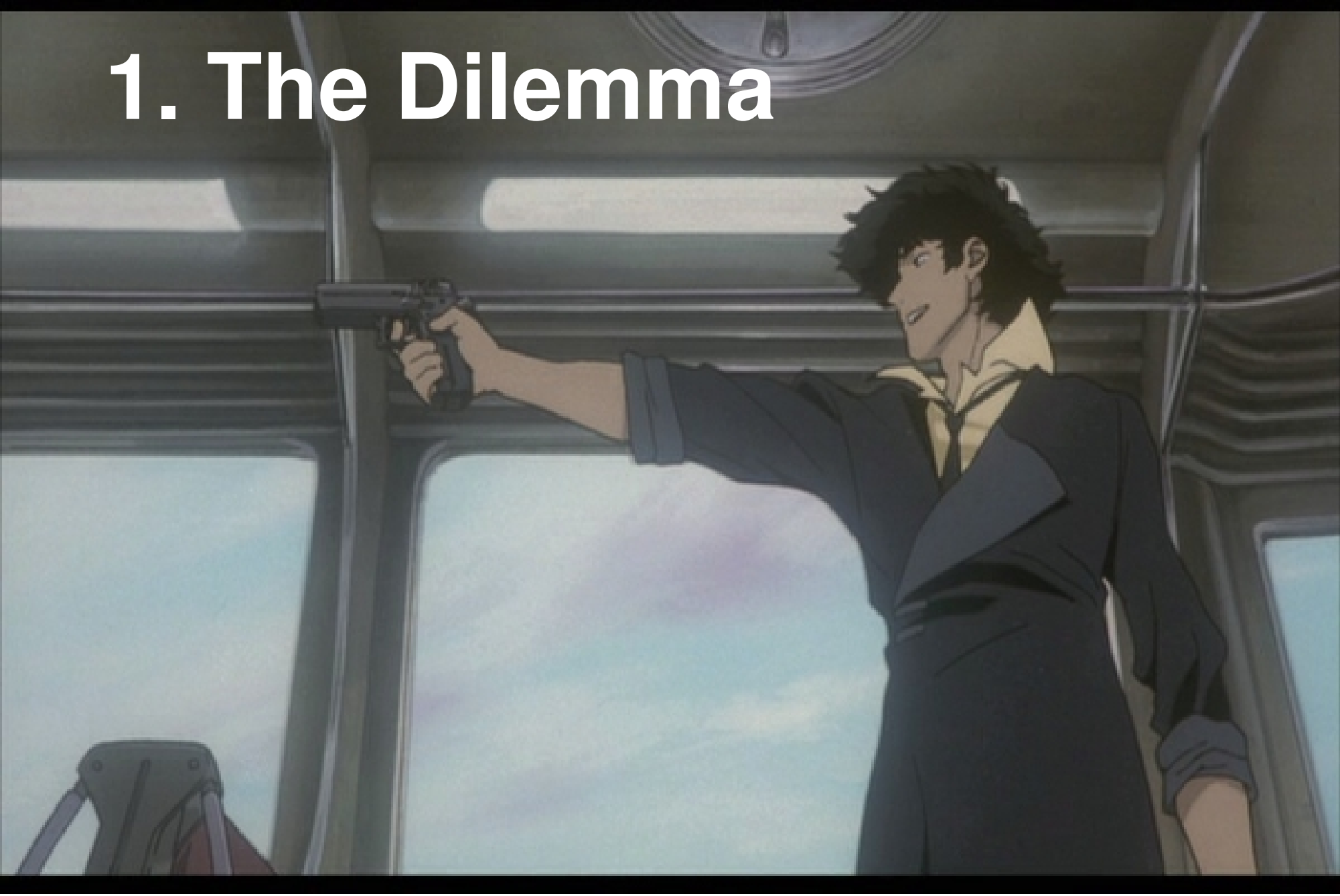
Andreas Schäfer

Friedrich-Alexander-Universität Erlangen-Nürnberg

WOLFHPC 2012 @ SC'12, 2012.11.16

Outline

1. The Dilemma



What this talk is about

- interface: parallel library $\leftrightarrow$ user code
- achieving 0 overhead
- yet providing object-oriented API

What it's not about:

- stencil codes implementations

What this talk is about

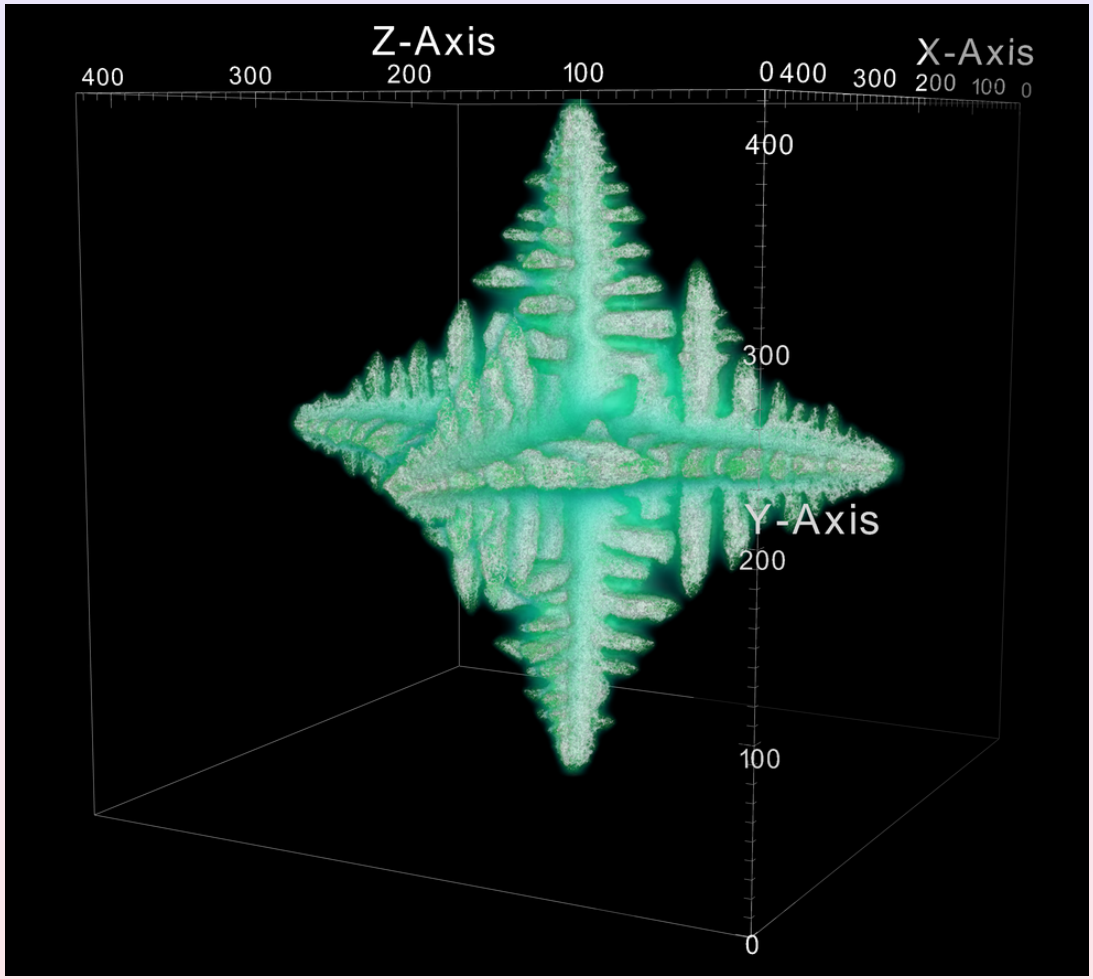
- interface: parallel library $\leftrightarrow$ user code
- achieving 0 overhead
- yet providing object-oriented API

What it's not about:

- stencil codes implementations

Increasingly Complex Simulation Models

- crystal growth in Al/Cu alloys
- $O(1\ TB)$ output data
- 800 B per cell



Increasingly Complex Simulation Models (cont.)

Object-oriented Model (Array of Structs)

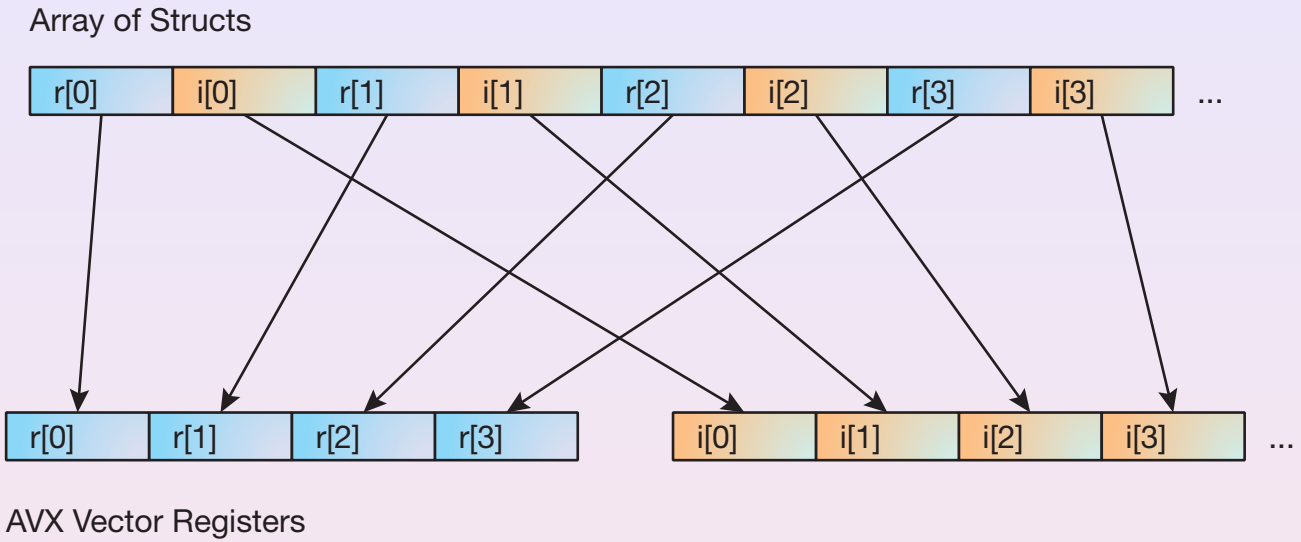
```
class TCell { ...
  double deltaNEff;
  Tvector Z;
  Neumann<double> fluctuation;
  // Moore<double> fluctuation;
};
Grid<TCell, 3> grid;
```

C-style Model (Struct of Arrays, clumsy!)

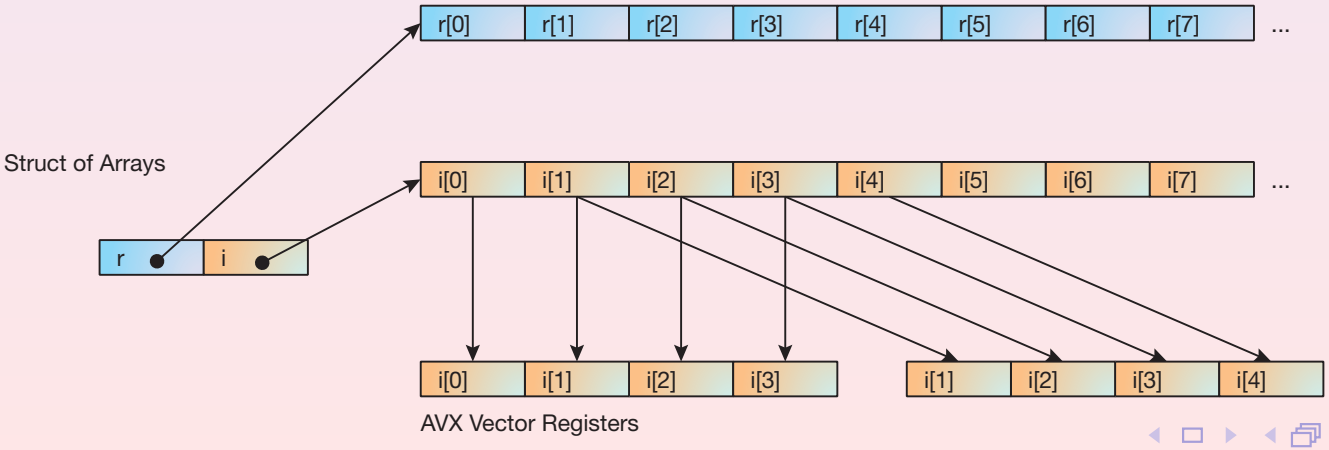
```
struct TGrid {
  double deltaNEff[DIM_Z][DIM_Y][DIM_X];
  double Z[DIM_Z][DIM_Y][DIM_X][3];
  double fluctuation[DIM_Z][DIM_Y][DIM_X][6];
  //double fluctuation[DIM_Z][DIM_Y][DIM_X][27];
};
```

Memory Layout: Arrays of Structs vs. Struct of Arrays

Array of Structs:



Structs of Array:

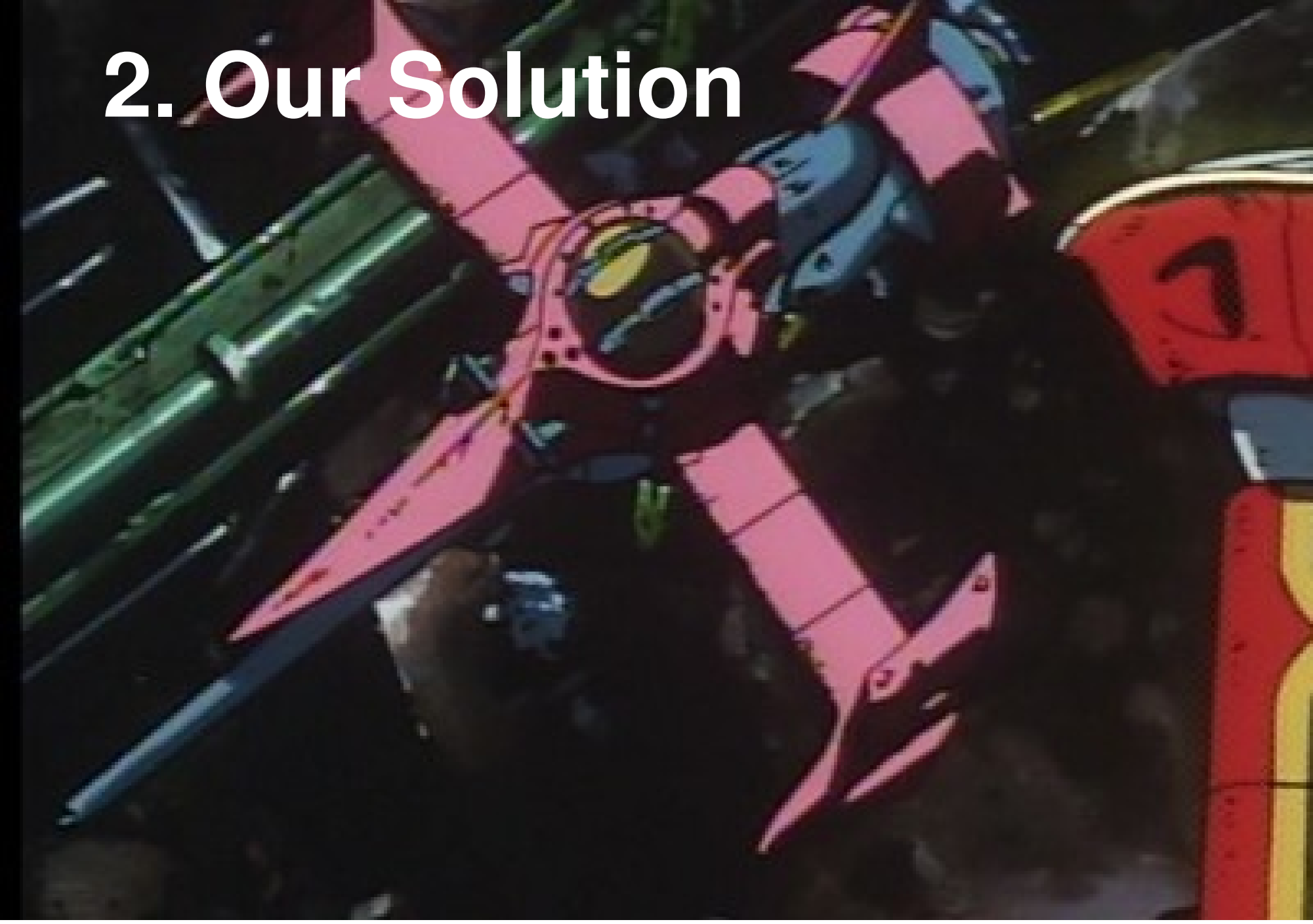


What's Taking so Long?

```
for(Uint z=1; z<zSize-1; ++z)
  for(Uint y=1; y<ySize-1; ++y){
    for(Uint x=1; x<xSize-1; ++x) {
      Real velX, velY, velZ;
      velX =
        src->GET_COMP(x-1,y,z,E) +
        src->GET_COMP(x-1,y-1,z,NE) +
        src->GET_COMP(x-1,y+1,z,SE) +
        src->GET_COMP(x-1,y,z-1,TE) +
        src->GET_COMP(x-1,y,z+1,BE);
...
```

- compute?
- data transfer?
- address computation!

2. Our Solution



Our Solution

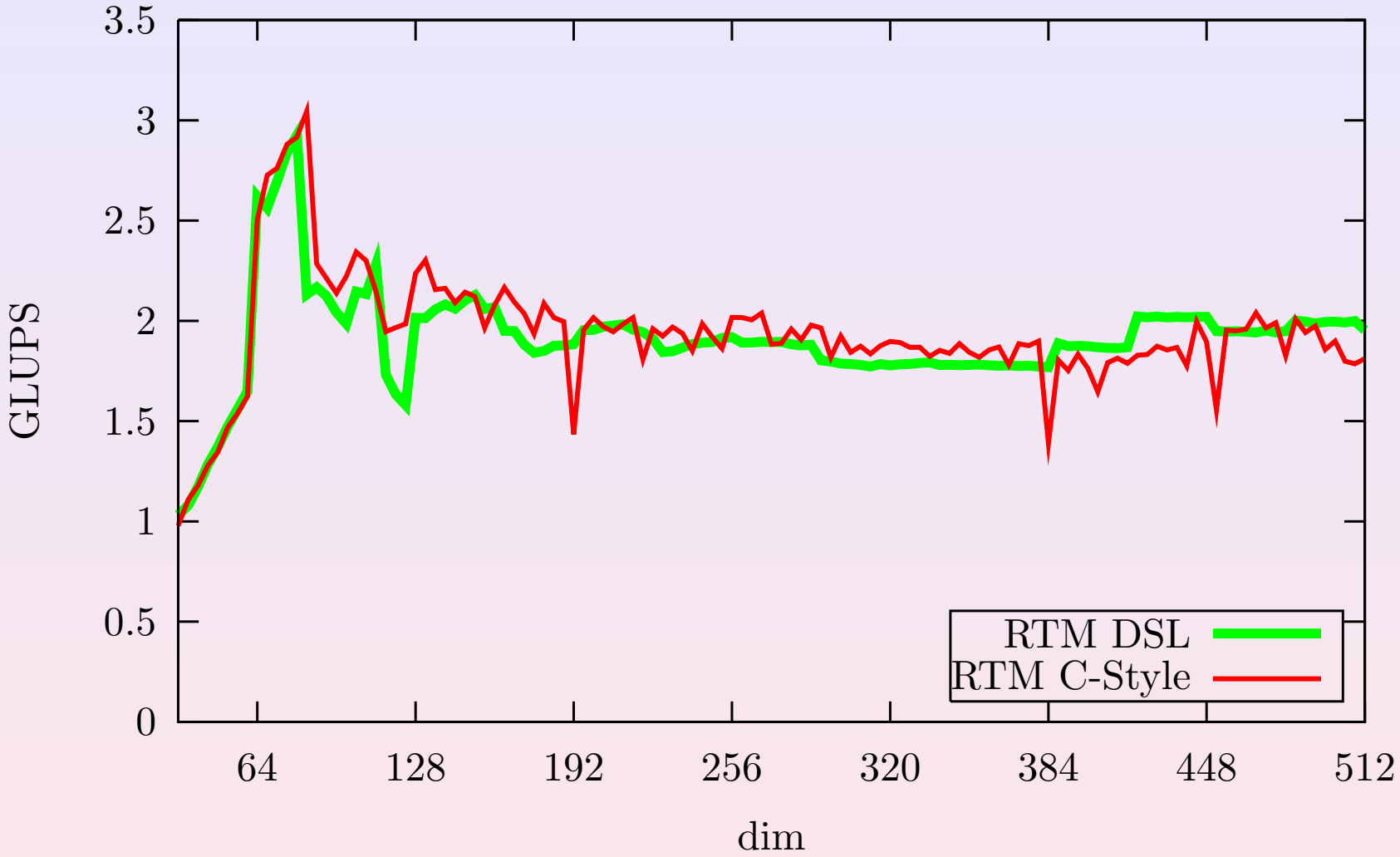
- C++ templates and Macros
- store data in *Struct of Arrays* layout
- provide *Arrays of Structs* interface (object-oriented)
- proxy-objects removed by compiler (fast!)
- offset computation at compile time (fast!)
- works with CPUs and GPUs

Example

```
class Cell {
public:
#define hoody(X, Y) hood[FixedCoord<X, Y, 0>()]

  template<typename CELL, typename HOOD>
  static void updateLine(CELL& c, const HOOD& hoody...) {
  for (*x = startX; *x < endX; ++(x)) {
    c.r() =
     (hoody( 0, -1).r() +
      hoody(-1,  0).r() +
      hoody( 0,  0).r() +
      hoody( 1,  0).r() +
      hoody( 0,  1).r()) * (1.0 / 4.0) +
      hoody(0, 0).i() *  hoody(0, 0).i();
  }

  double r;
  double i;
};

LIBGEODECOMP_REGISTER_SOA(Cell, ((double)(r))((double)(i)))
```

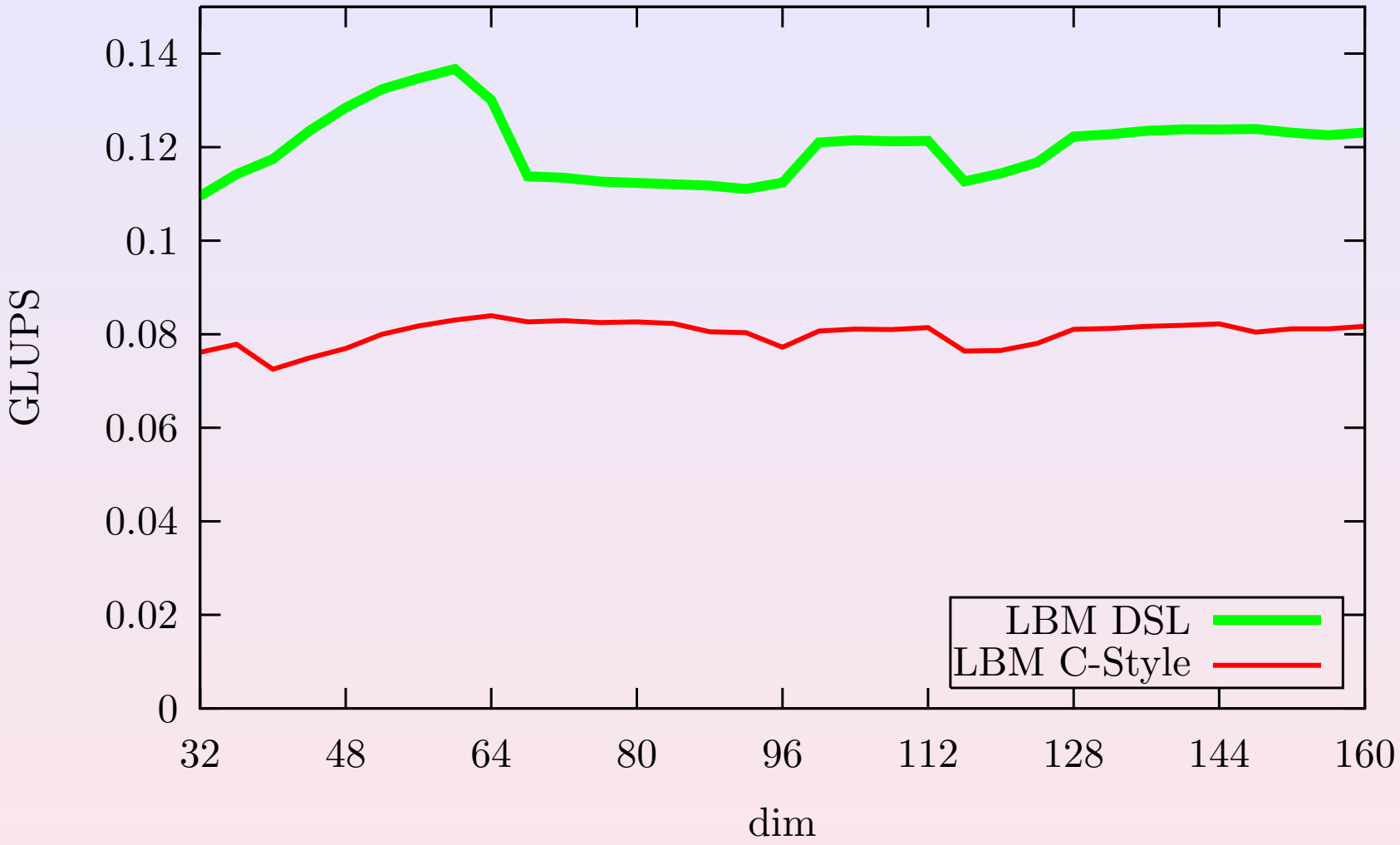
3. Results

Benchmark Results: 3D Reverse Time Migration



- measured on Tesla C2050
- no performance gain for DSL

Benchmark Results: 3D Lattice Boltzmann



- measured on Tesla C2050
- approx. 50 % speedup

Summary

- complex models need objects
- vectorization vs. objects
- we generate
  - SoA data-structures
  - highly efficient proxy objects
- get best of both worlds!
- available for download (free, open-source)
  `http://www.libgeodecomp.org`