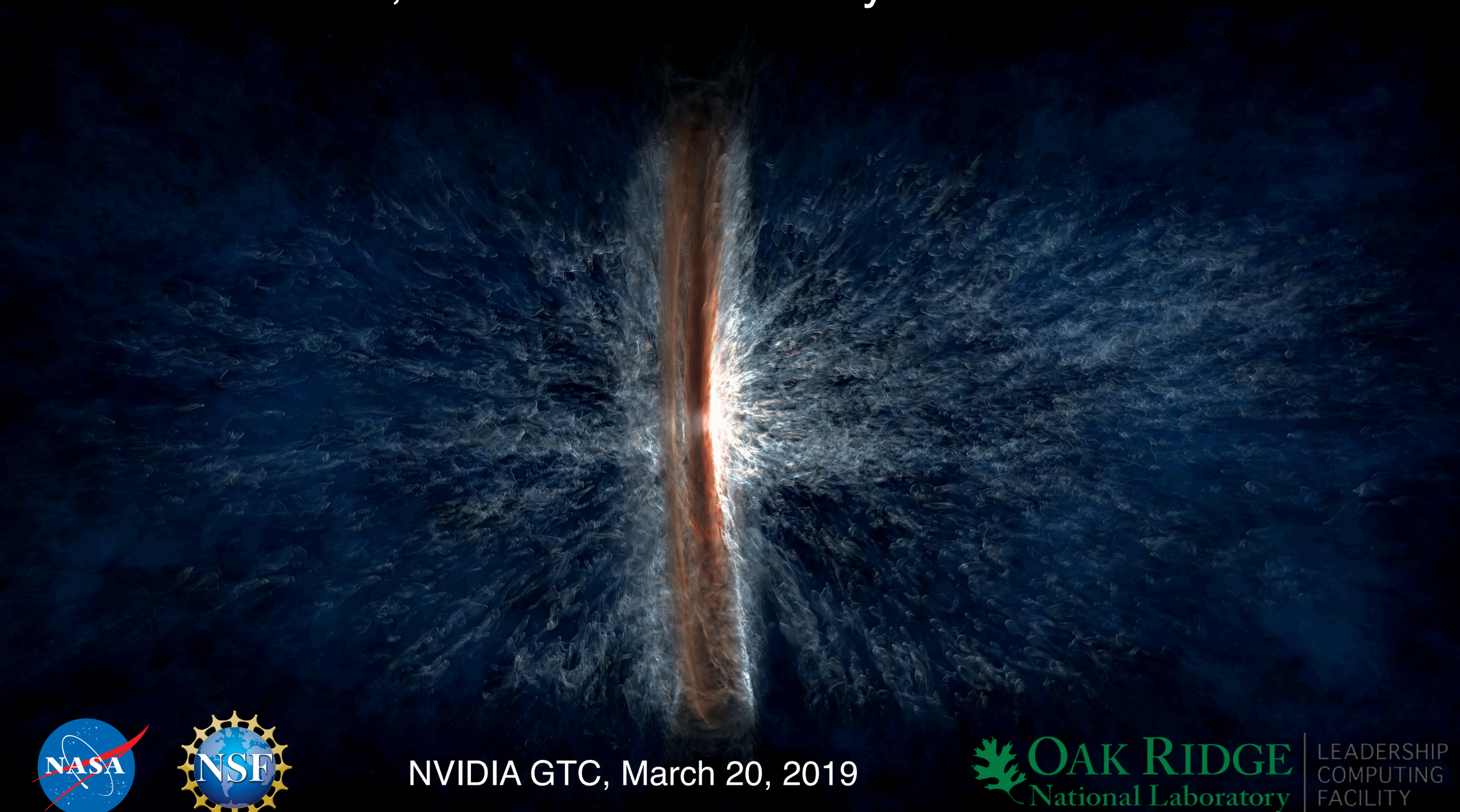
Advancing Astrophysics with the GPU-Native, Massively-Parallel Code, Cholla

Evan Schneider, Princeton University



NVIDIA GTC, March 20, 2019

OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY

Why did we need a new astrophysics code?

Starburst Galaxy M82

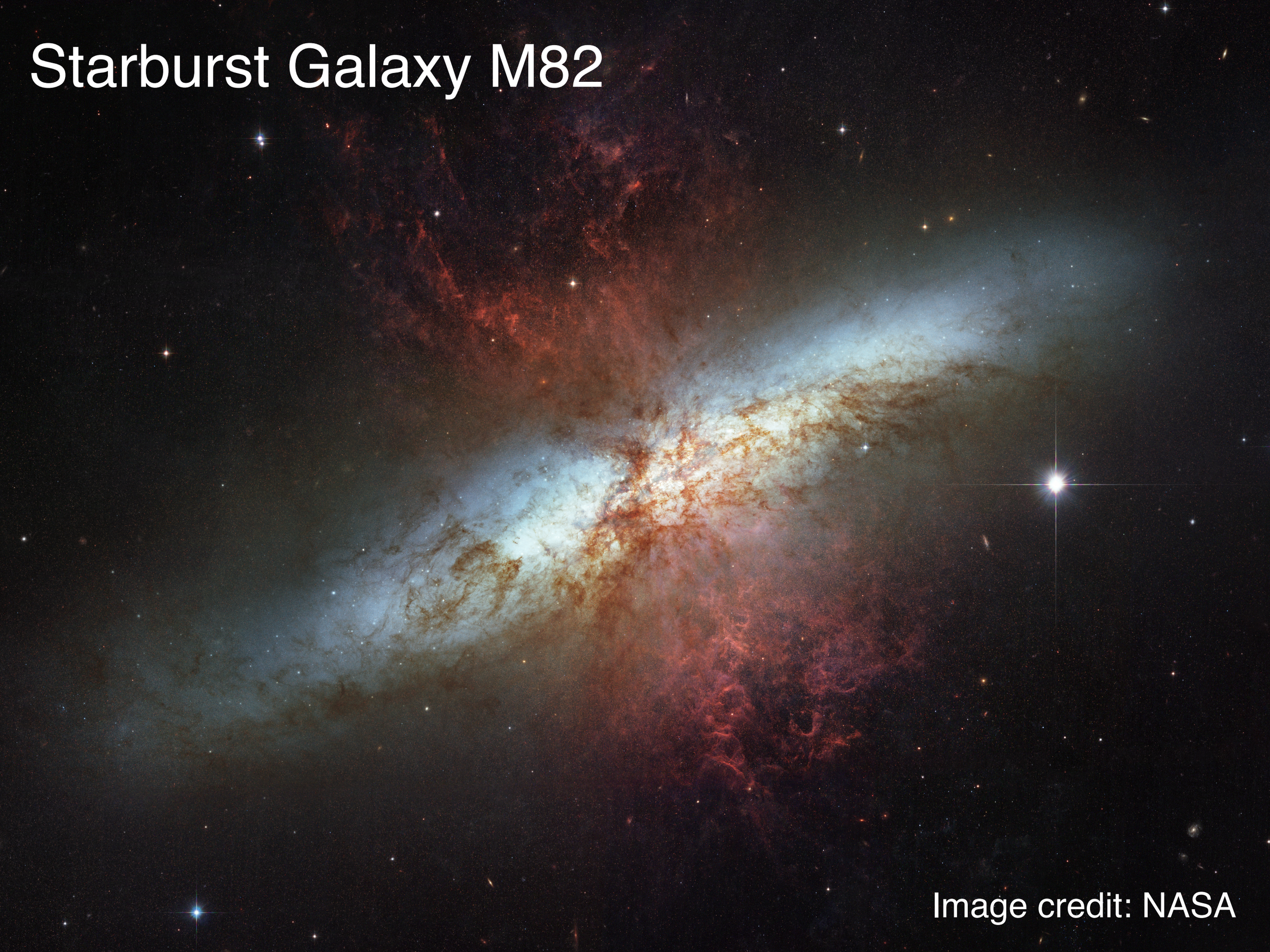


Image credit: NASA

Starburst Galaxy M82

Image credit: NASA

Starburst Galaxy M82

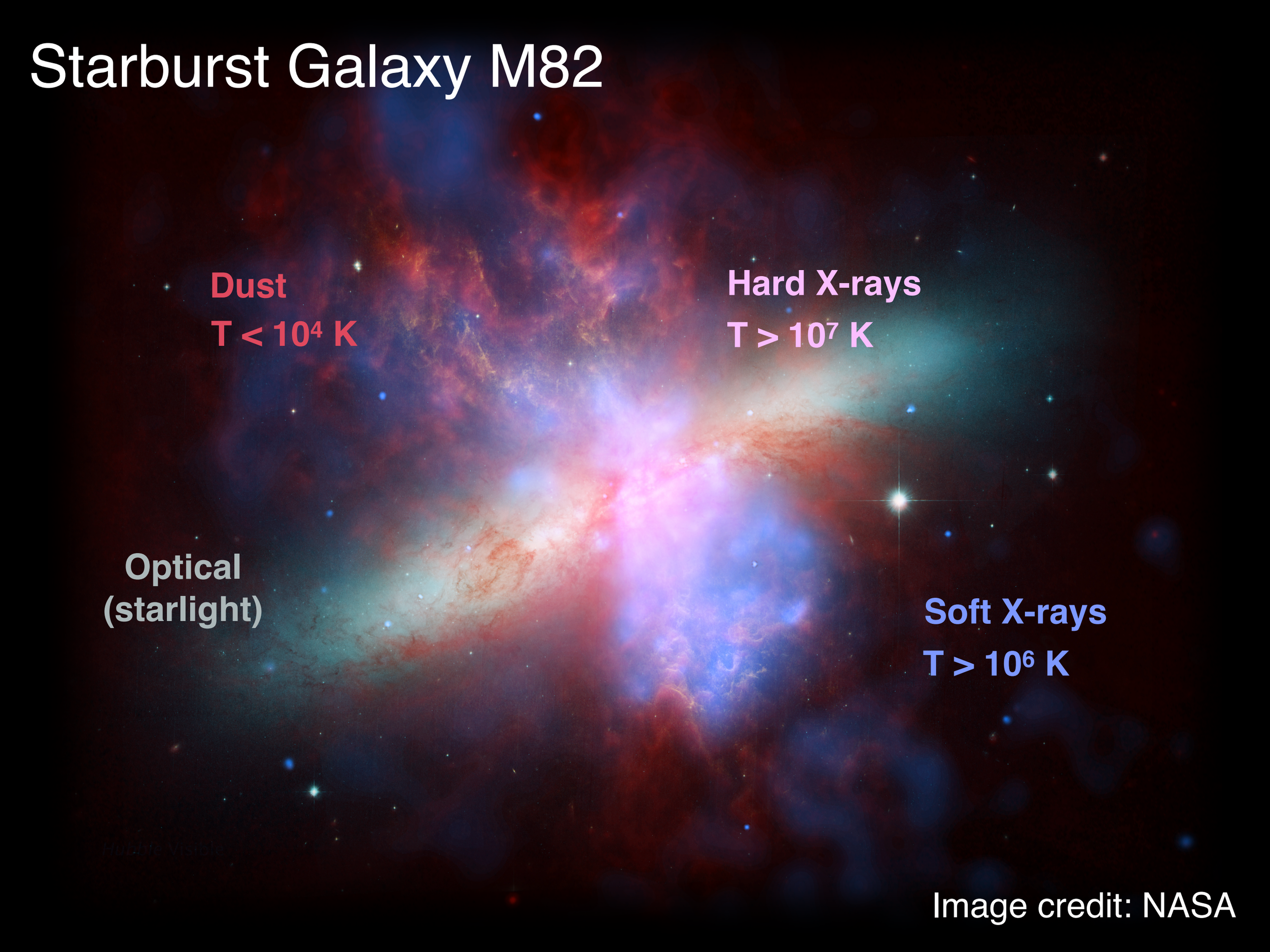
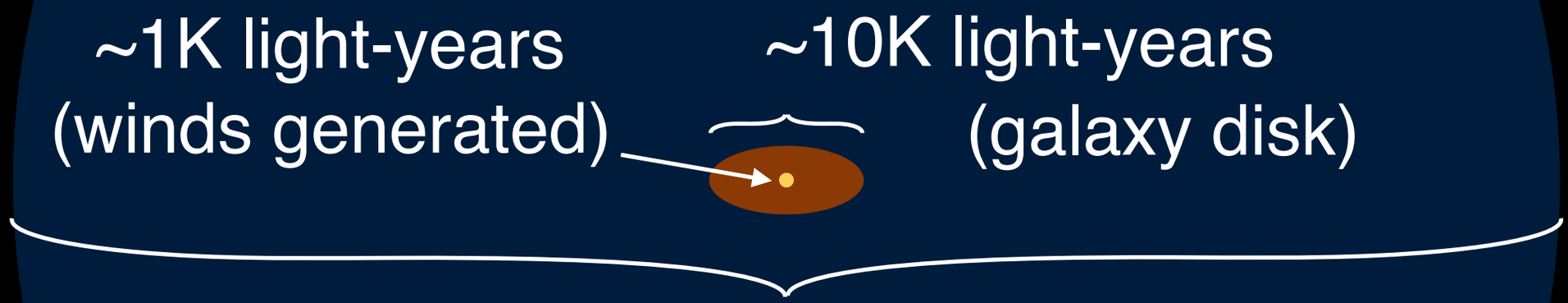


Image credit: NASA

Simulating These Systems Is Computationally Challenging



The scales involved in galactic wind evolution range from 10 light-years (cooling radius of supernova bubbles) to 1 million light years (extent of galaxy halo).

Starburst Galaxy M82

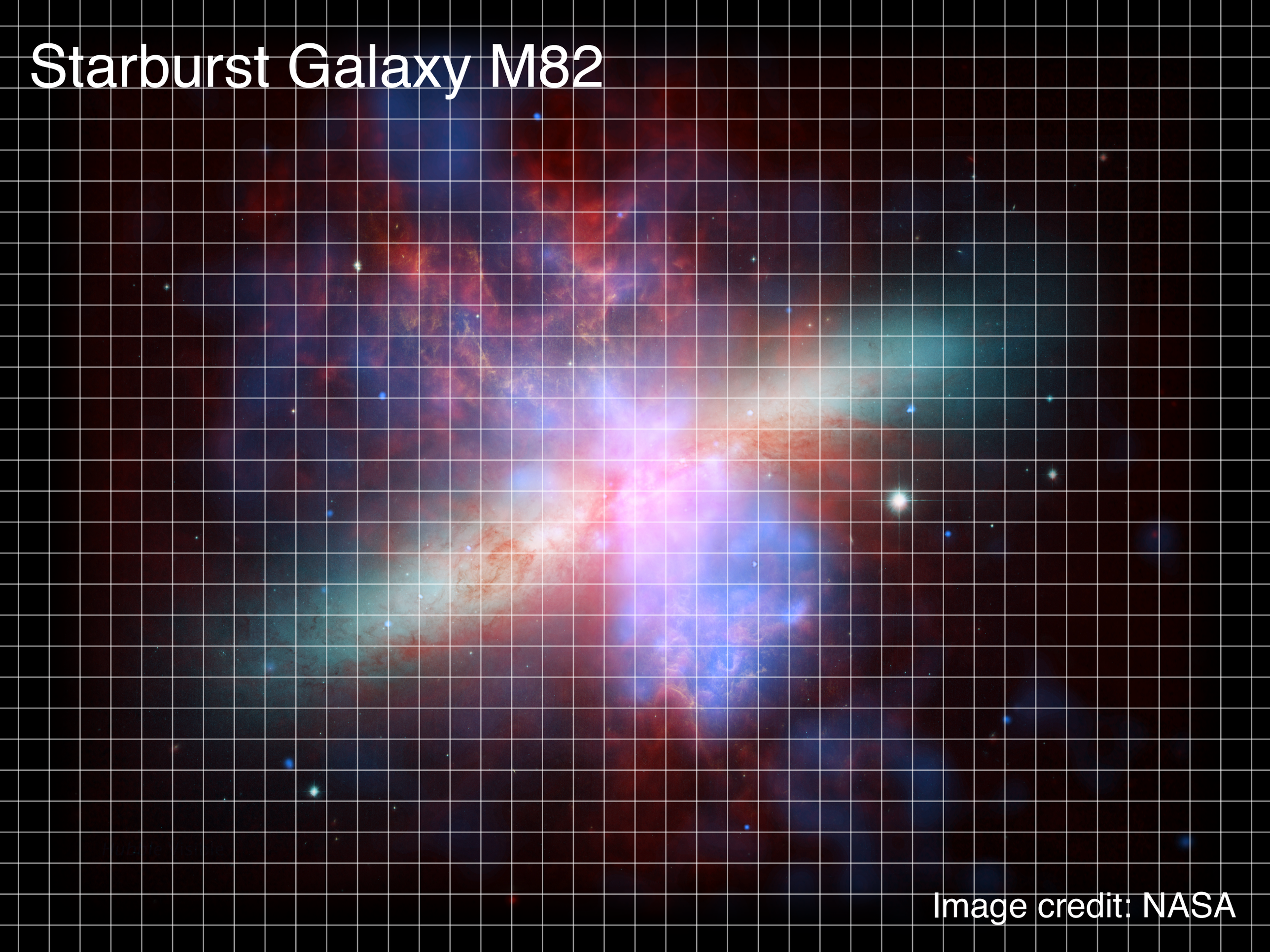
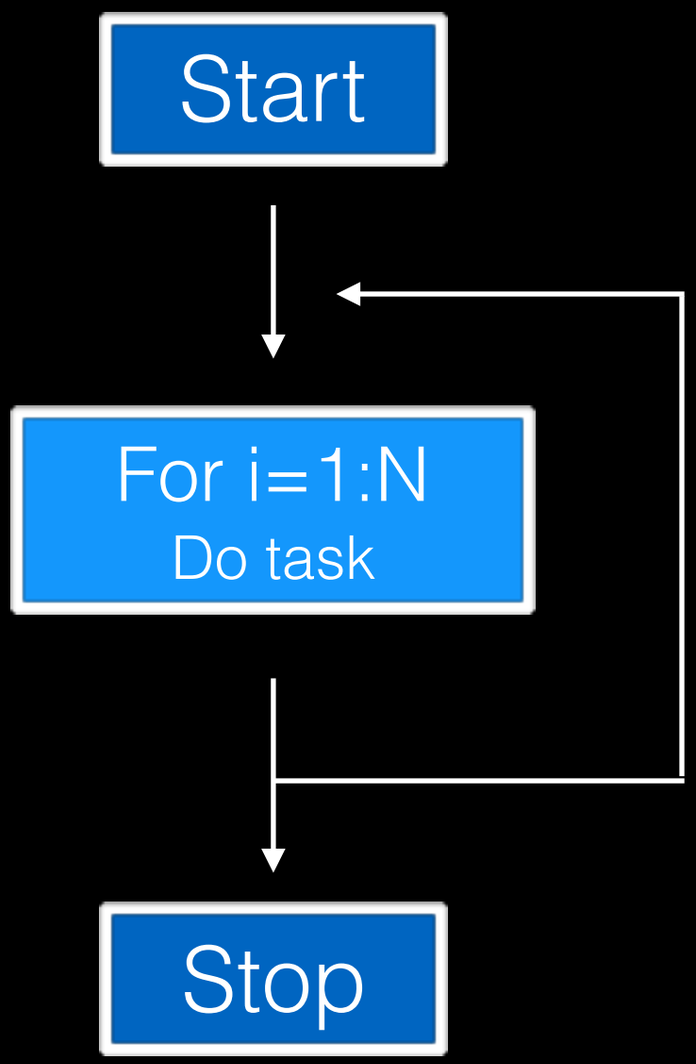


Image credit: NASA

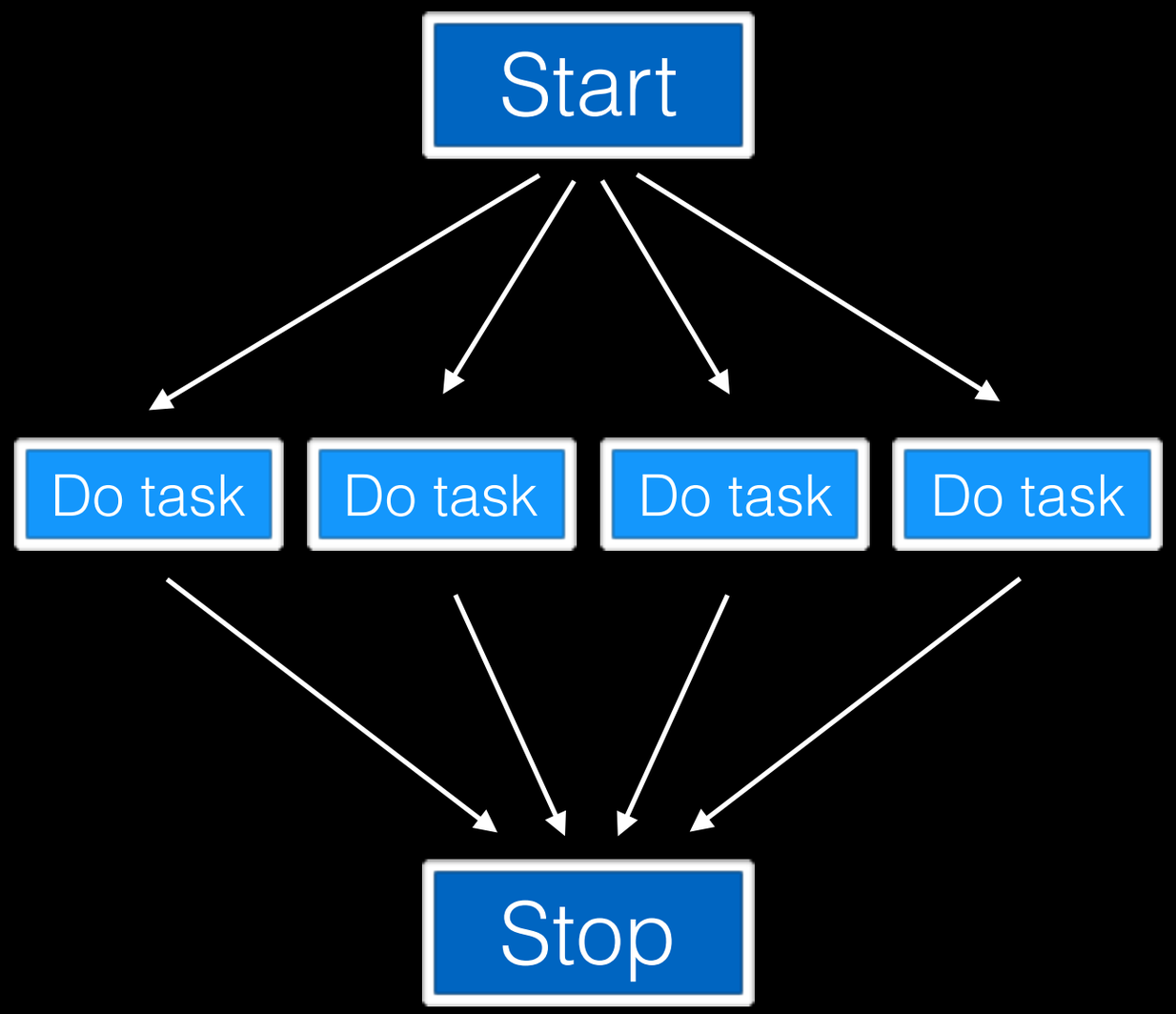
How can we improve this situation?

Computer Architectures Have Changed

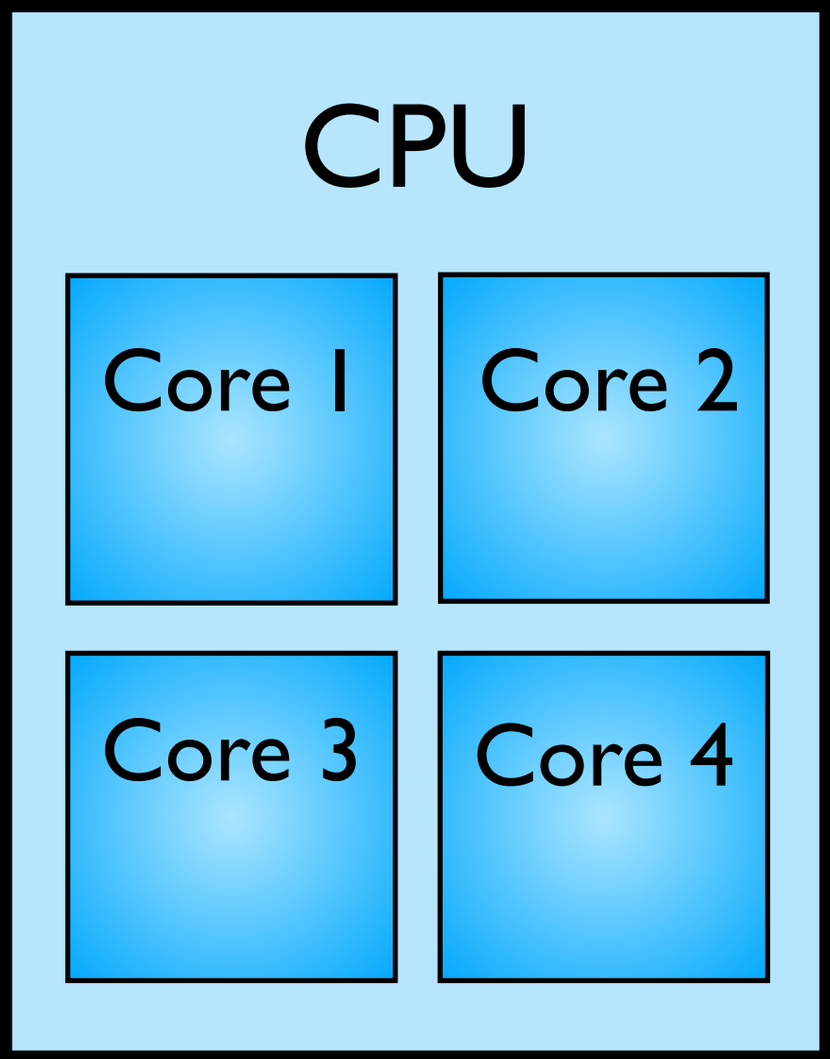
Serial Approach



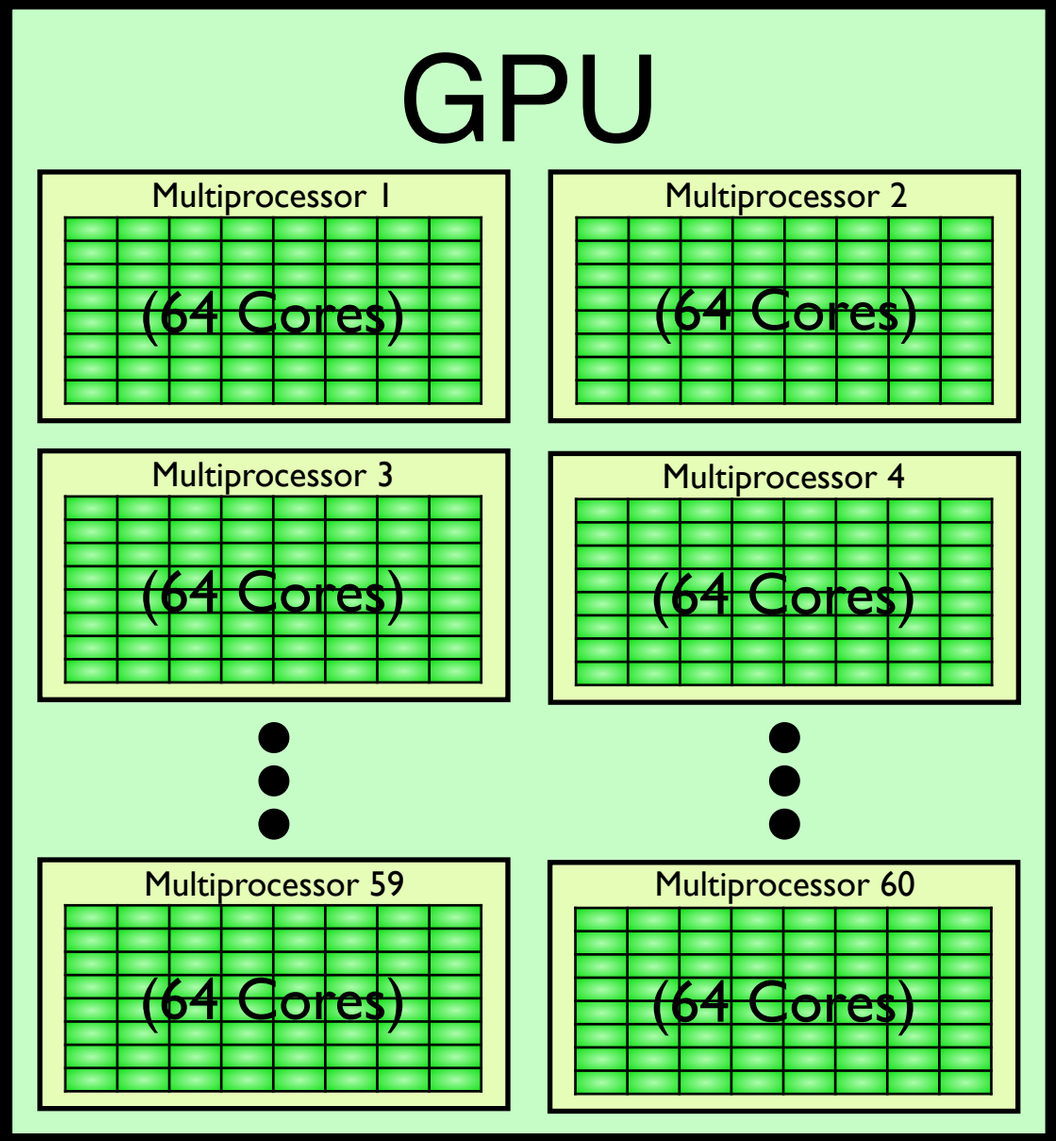
Parallel Approach



Computer Architectures Have Changed



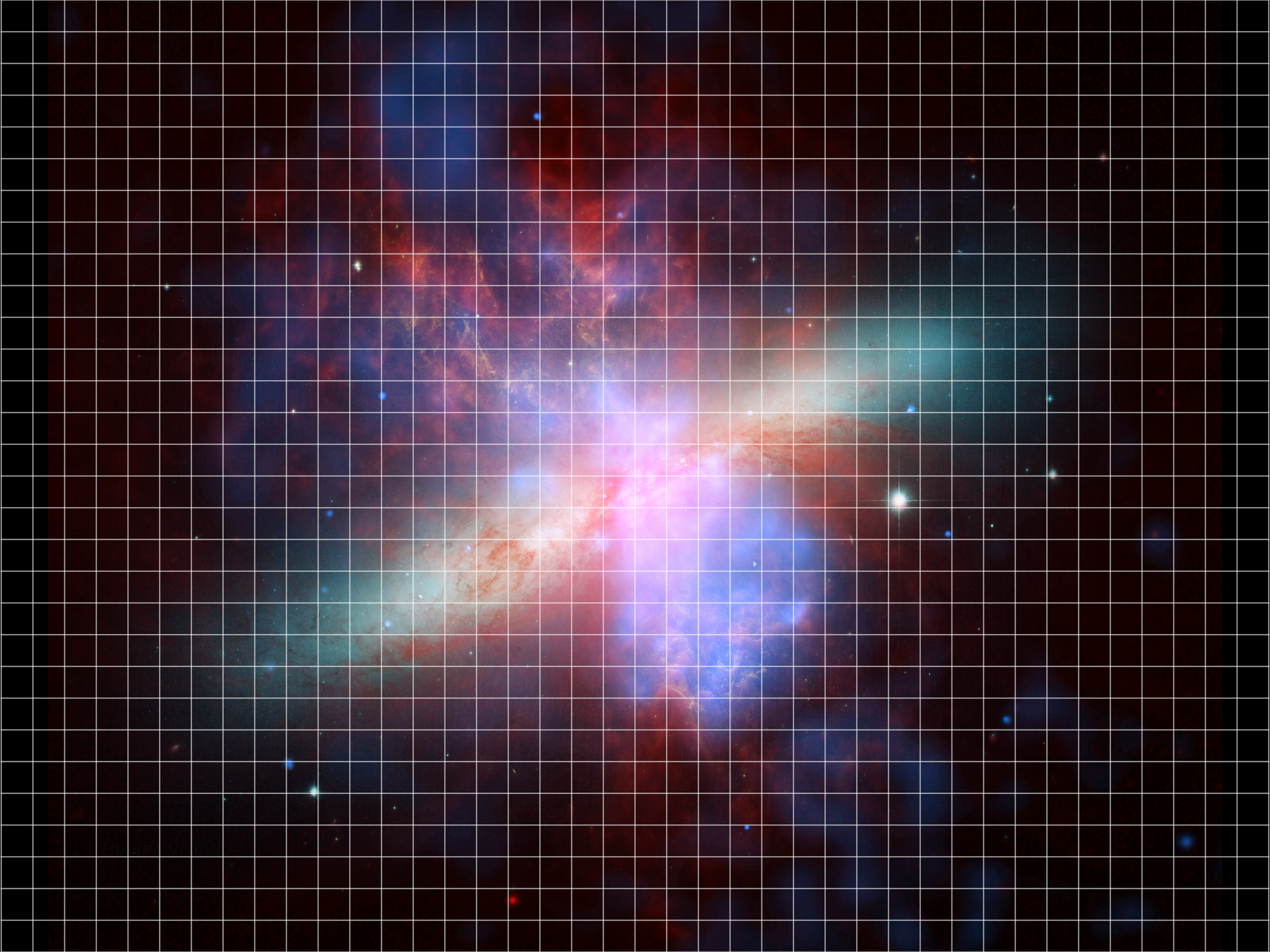
Optimized for Serial Tasks



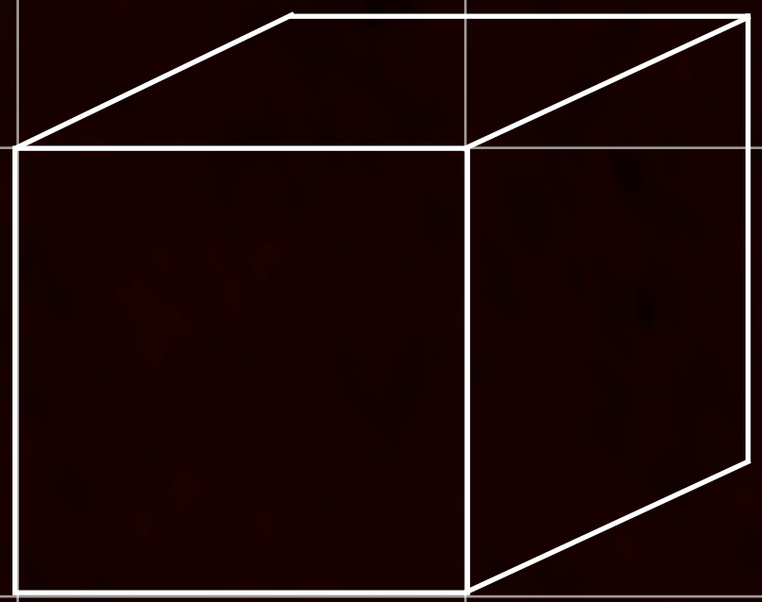
Optimized for Parallel Tasks

So, the goal was to build a *new* code, that could:

- achieve high resolution throughout the simulation volume (run simulations with large numbers of cells)
- take full advantage of new computing architectures
- address the limitations of the previous generation of astrophysics codes.

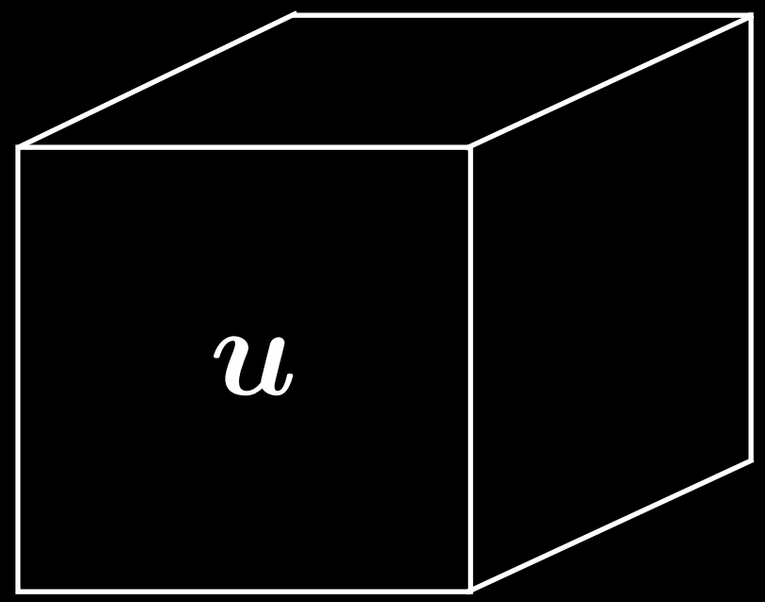


A (brief) introduction to finite-volume methods



A (brief) introduction to finite-volume methods

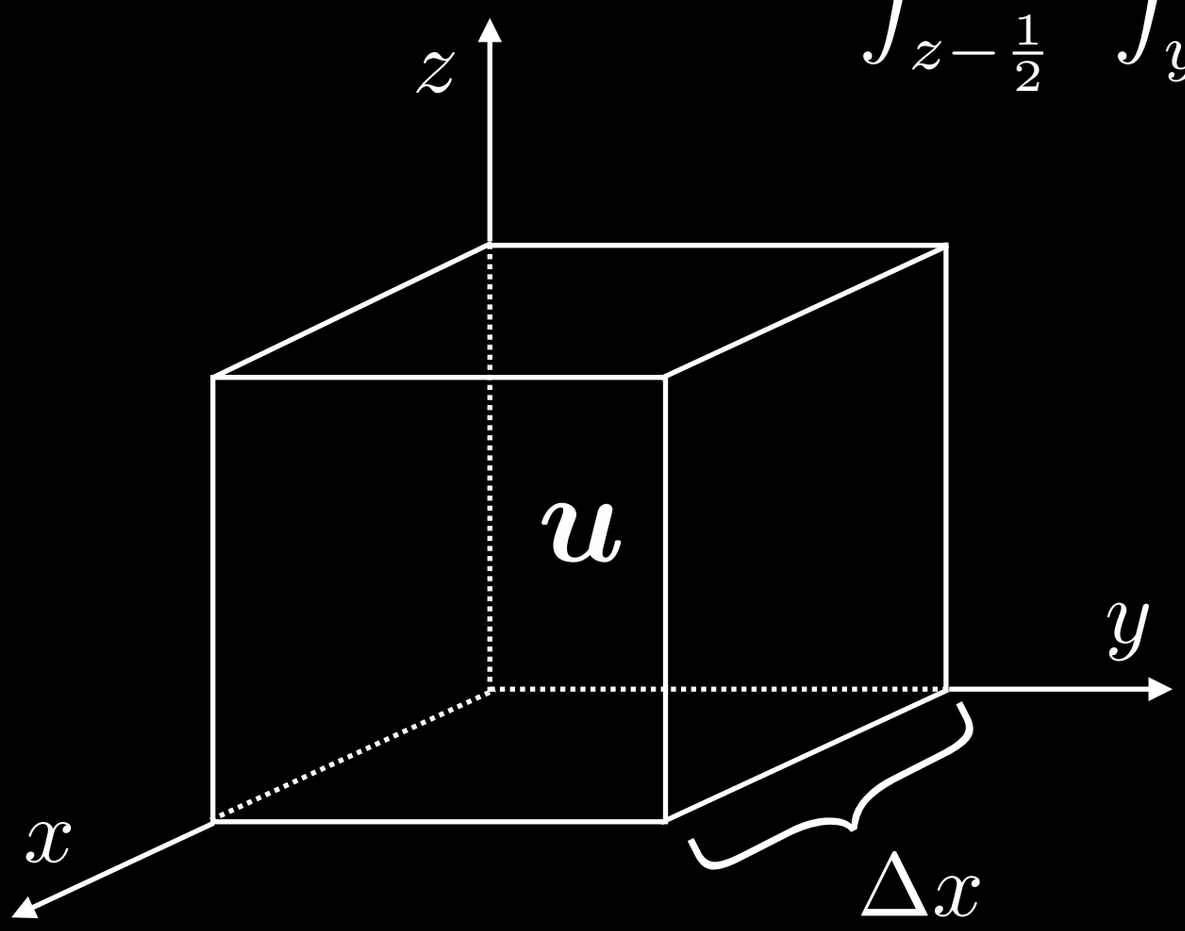
$\boldsymbol{u} = [\rho, \rho u, \rho v, \rho w, E]^{\mathrm{T}}$, a vector of conserved quantities



A (brief) introduction to finite-volume methods

$\boldsymbol{u} = [\rho, \rho u, \rho v, \rho w, E]^{\mathrm{T}}$, a vector of conserved quantities

$$\boldsymbol{u} = \int_{z-\frac{1}{2}}^{z+\frac{1}{2}} \int_{y-\frac{1}{2}}^{y+\frac{1}{2}} \int_{x-\frac{1}{2}}^{x+\frac{1}{2}} \frac{\boldsymbol{u}(x,y,z)}{\Delta x \Delta y \Delta z} \mathrm{d}x\mathrm{d}y\mathrm{d}z$$

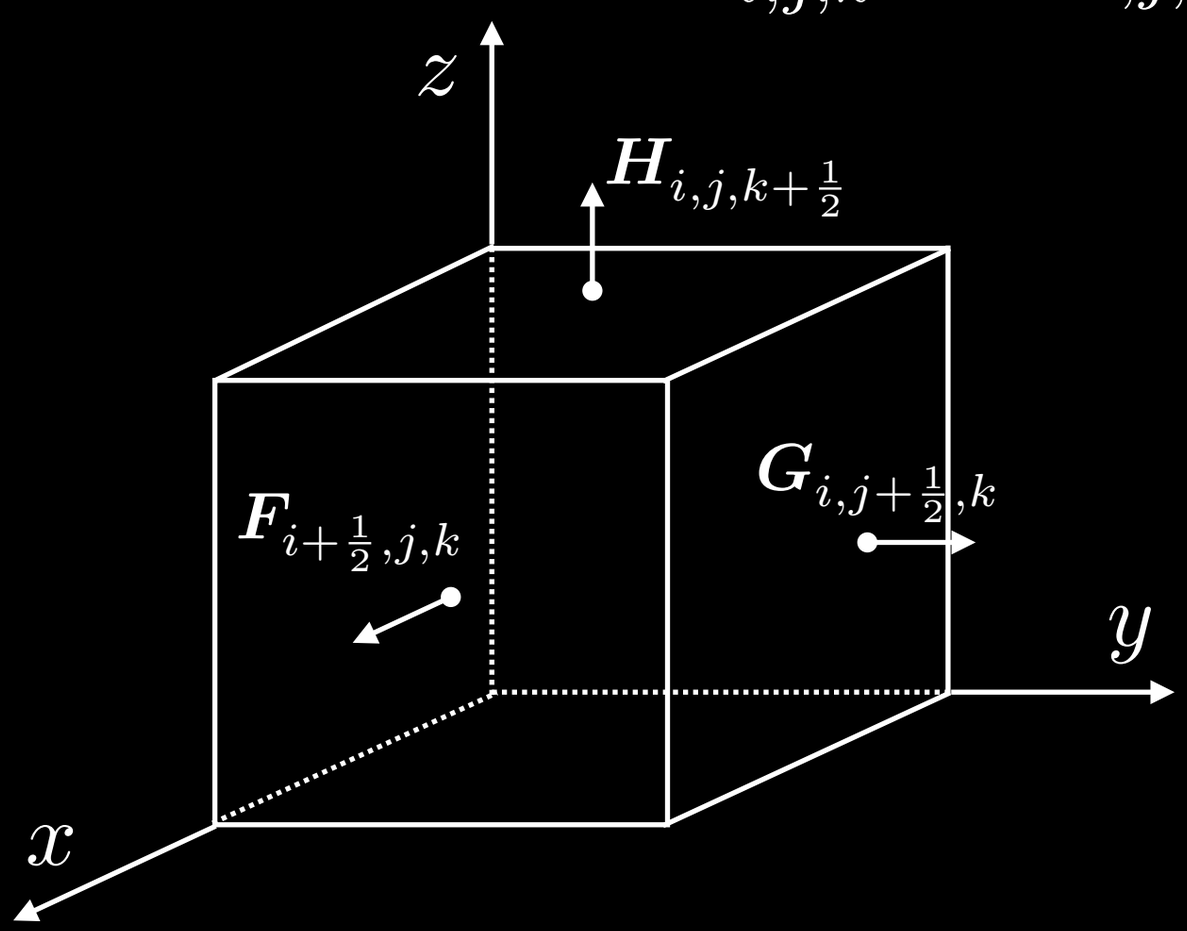


We want to go from ***u*** at time *t*, to ***u*** at time *t* + △*t*.

A (brief) introduction to finite-volume methods

$\boldsymbol{u} = [\rho, \rho u, \rho v, \rho w, E]^{\mathrm{T}}$, a vector of **conserved** quantities

$$\boldsymbol{u}_{i,j,k}^{t+\Delta t} = \boldsymbol{u}_{i,j,k}^{t} + \frac{\Delta t}{\Delta x}\left(F_{i+\frac{1}{2},j,k} - F_{i-\frac{1}{2},j,k}\right) + \frac{\Delta t}{\Delta y}\left(G_{i,j+\frac{1}{2},k} - G_{i,j-\frac{1}{2},k}\right) + \frac{\Delta t}{\Delta z}\left(H_{i,j,k+\frac{1}{2}} - H_{i,j,k-\frac{1}{2}}\right)$$



And that's it!

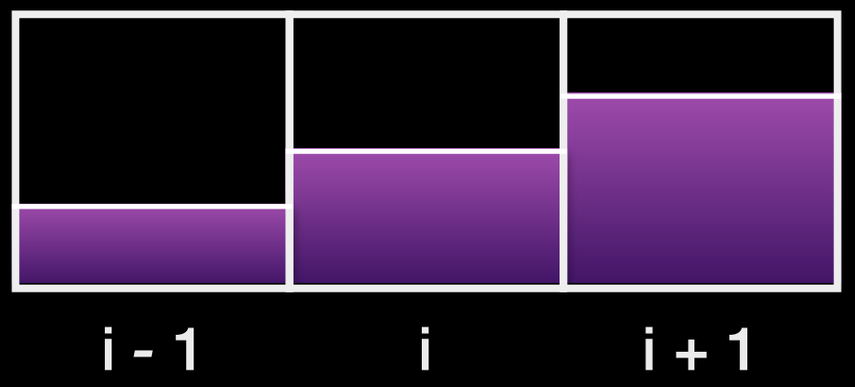
Well, not so fast…

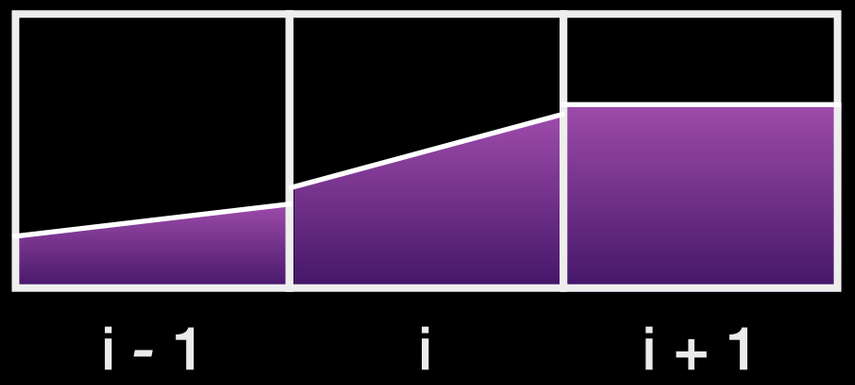
How do we do it?

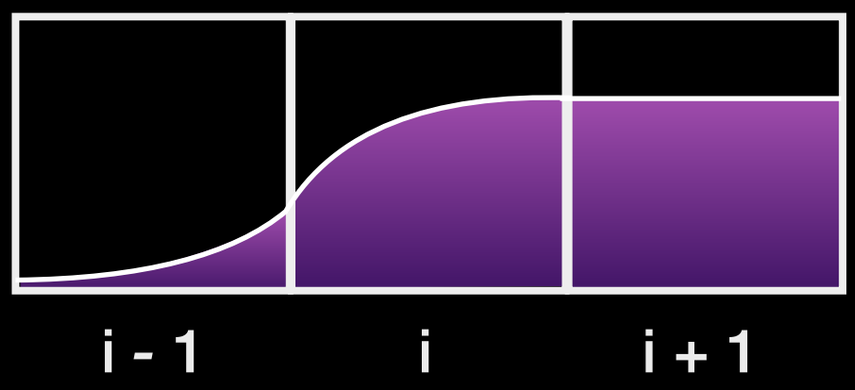
Reconstruct interface values using cell averages.

Choose either piecewise constant, piecewise linear, or piecewise parabolic reconstruction.

Piecewise linear and piecewise parabolic reconstruction can be done in either the primitive variables or the characteristic variables.

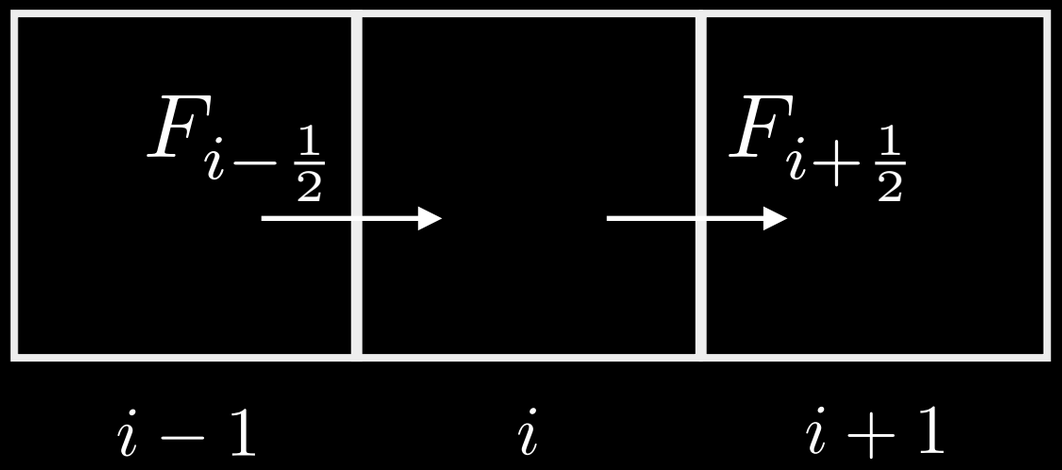


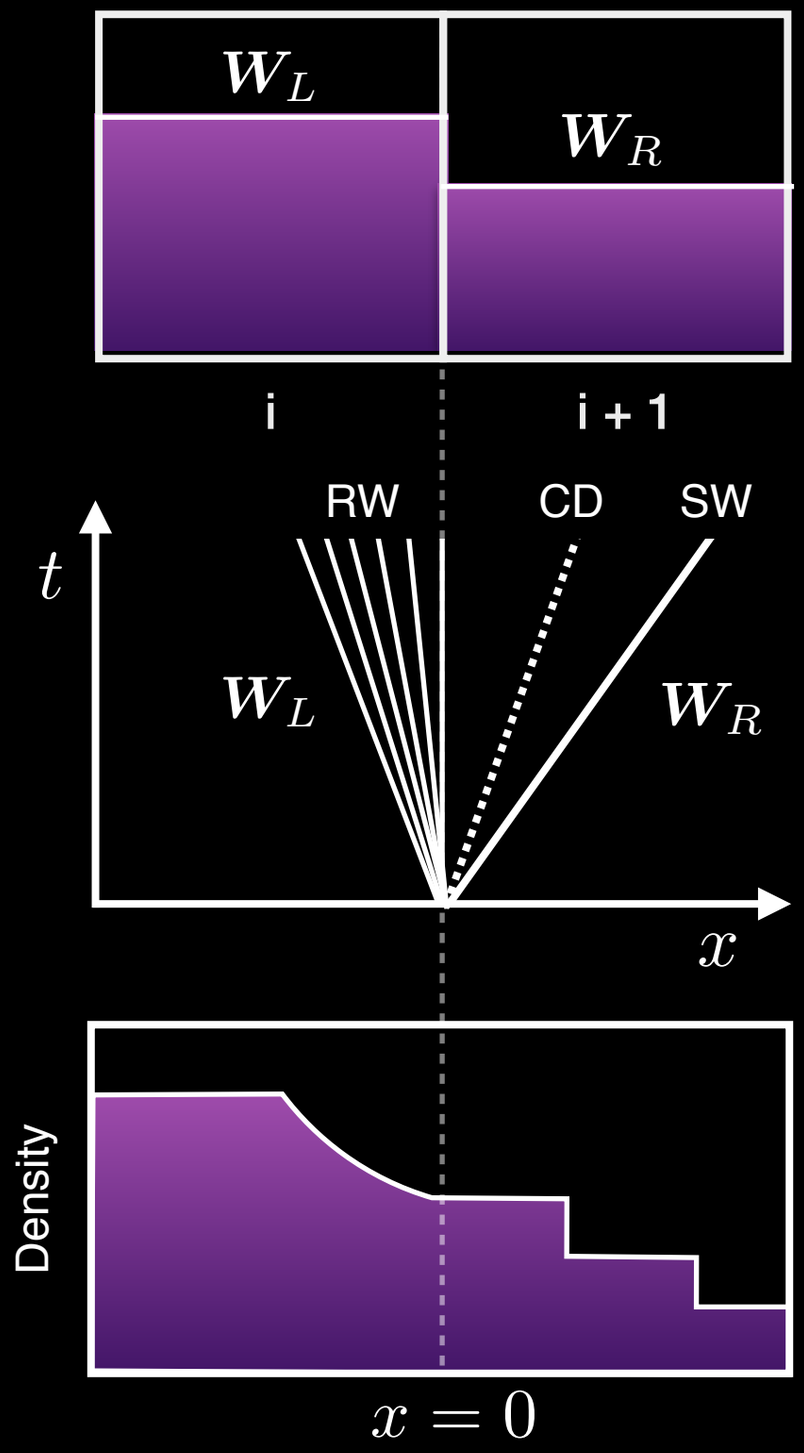




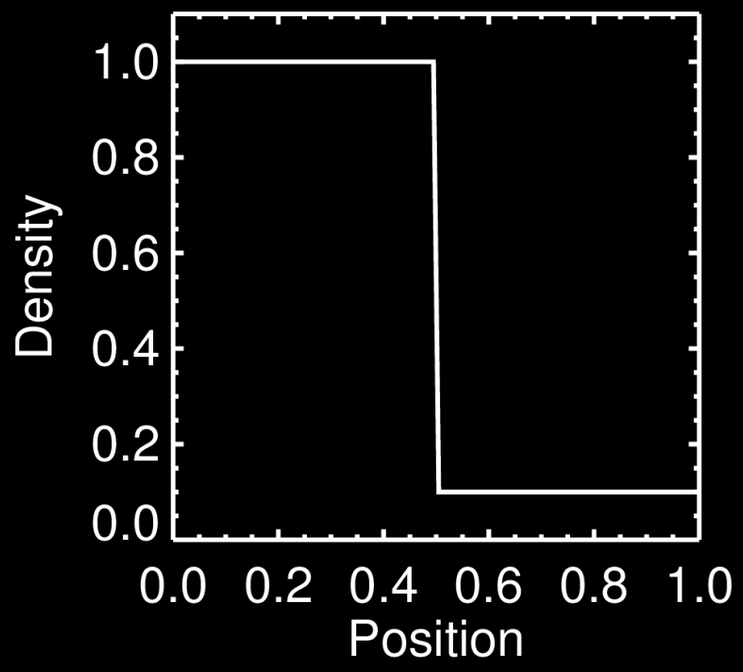
How do we do it?

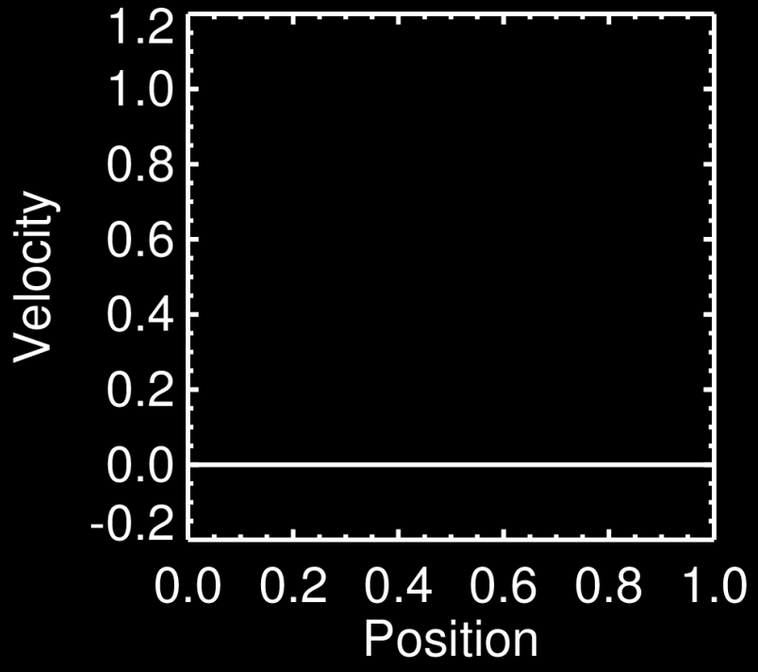
Calculate fluxes across cell interfaces using reconstructed interface values.

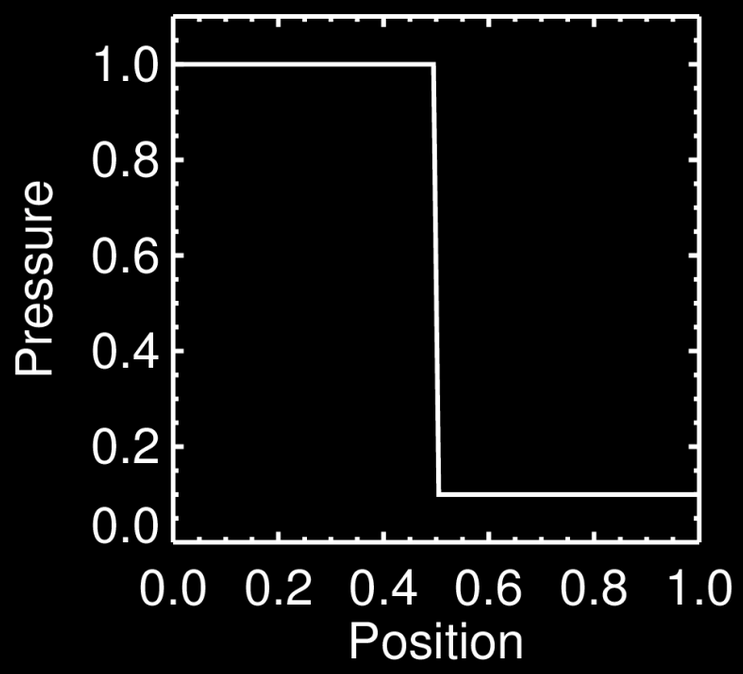


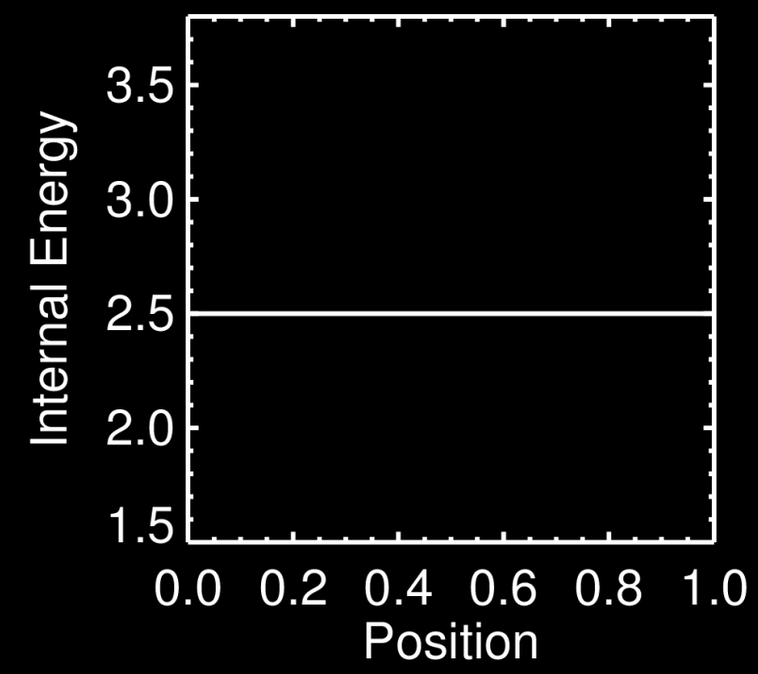


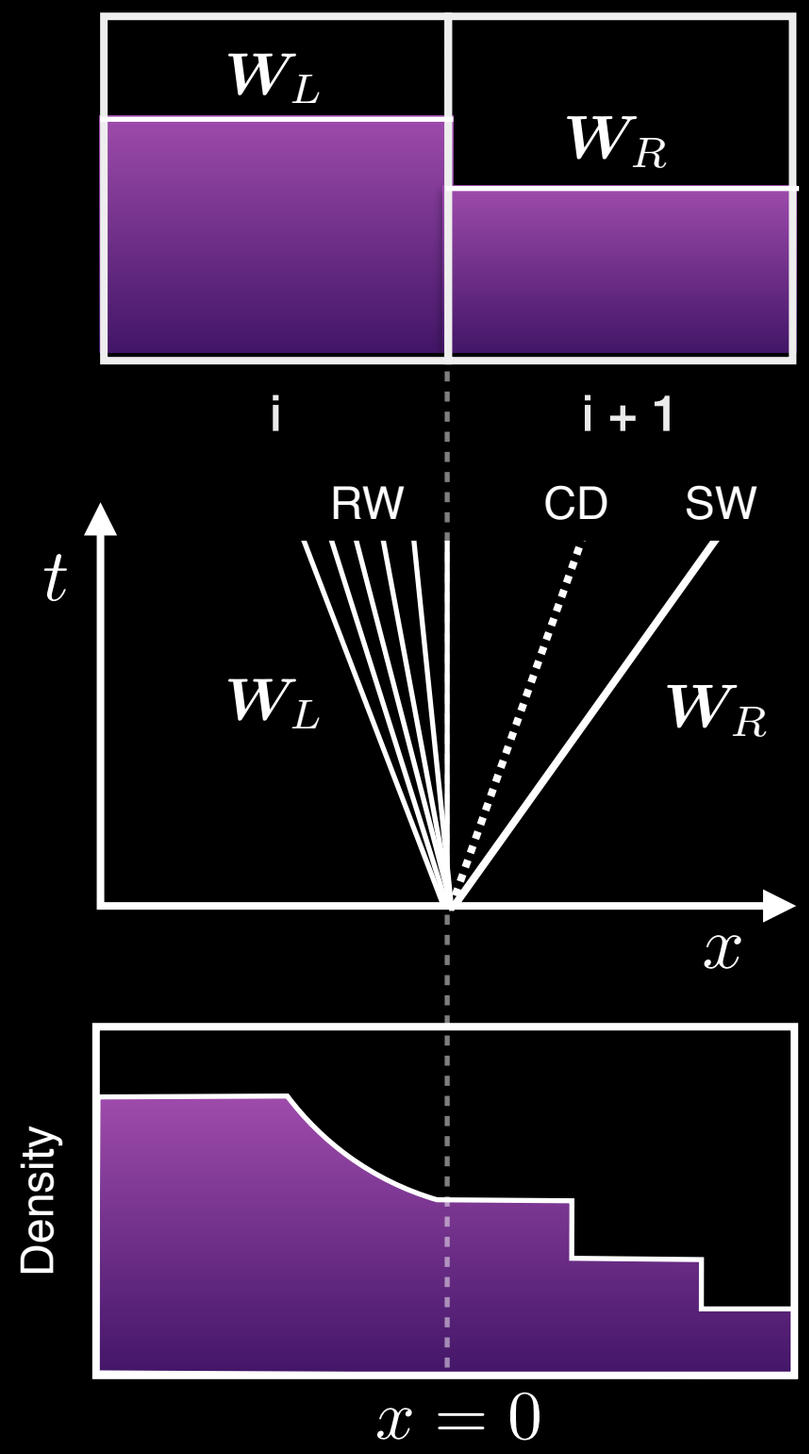
How do we do it?





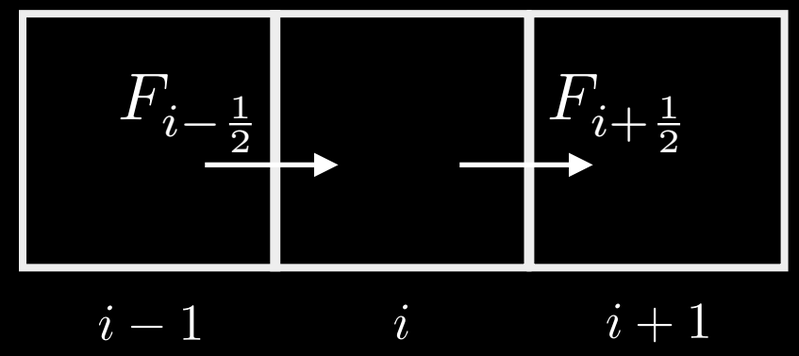




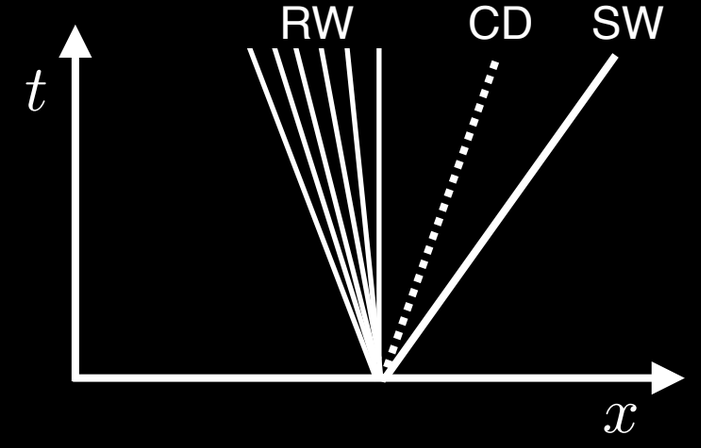


What's the GPU advantage?

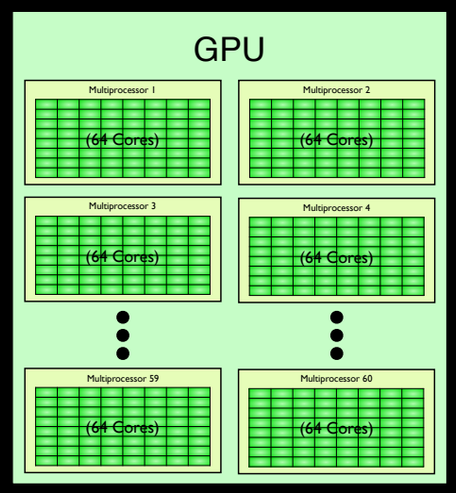
- Grid-based hydro codes are eminently parallelizable - each cell needs data from only a few nearby cells to reconstruct interface values, calculate fluxes across interfaces, and update conserved quantities.



- Hydro solvers are computationally expensive. Many unsplit algorithms require 6 Riemann problems per cell, per timestep (in 3D).

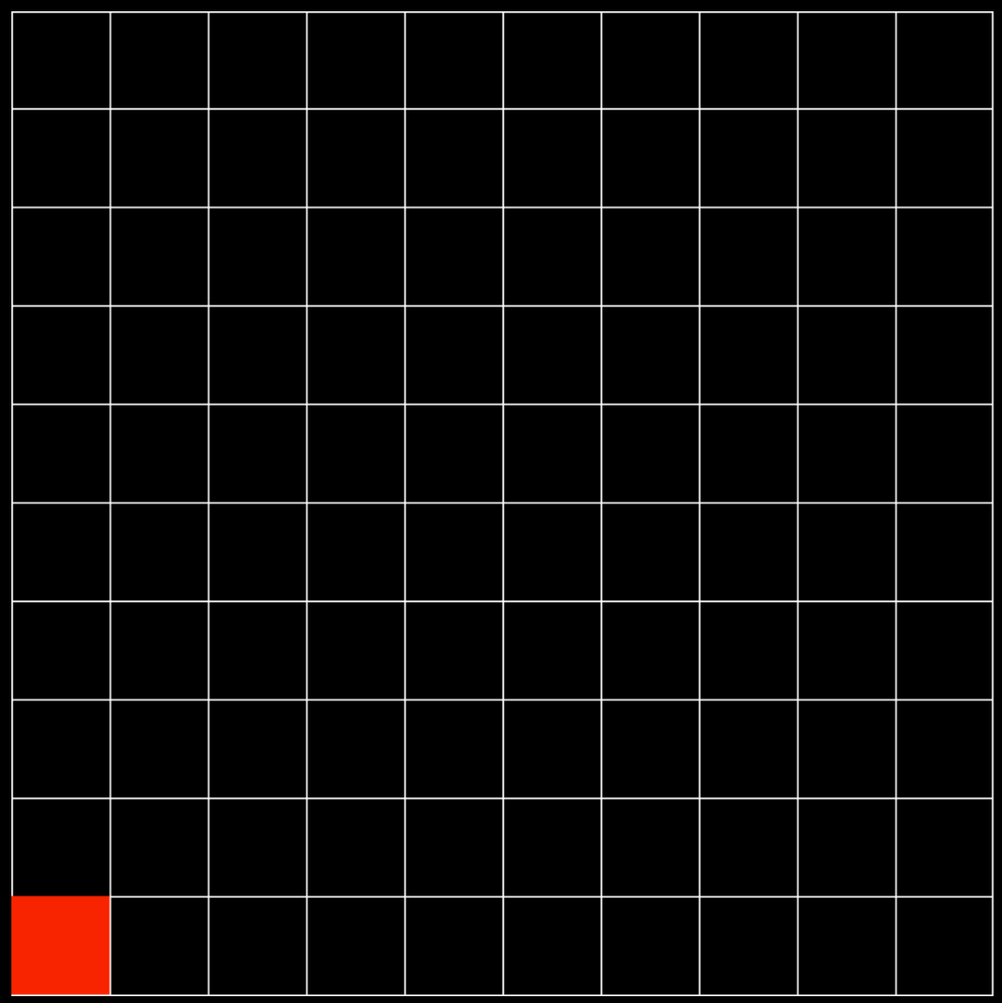


- With GPUs, we massively parallelize the calculation across many cores, allowing us to speed up computation by an order of magnitude as compared to similarly intensive CPU codes.

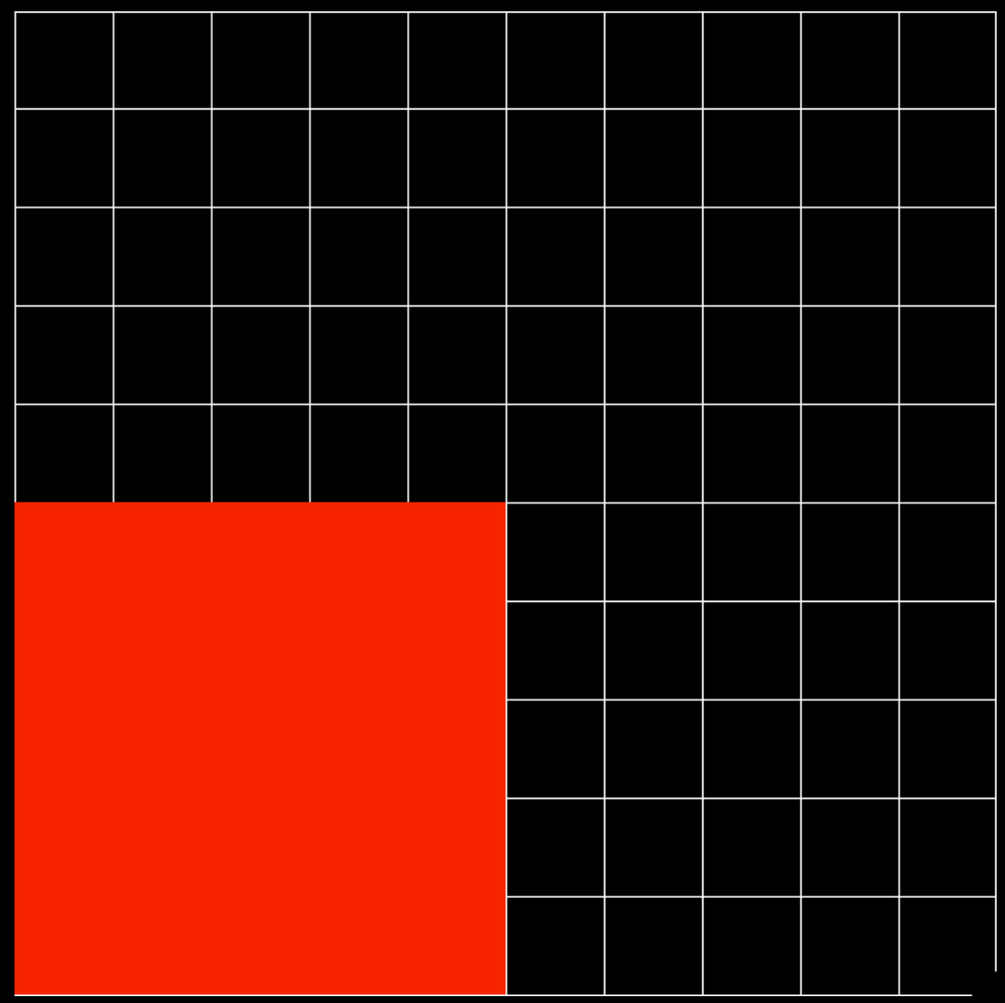


What's the GPU advantage?

Grid code on a CPU

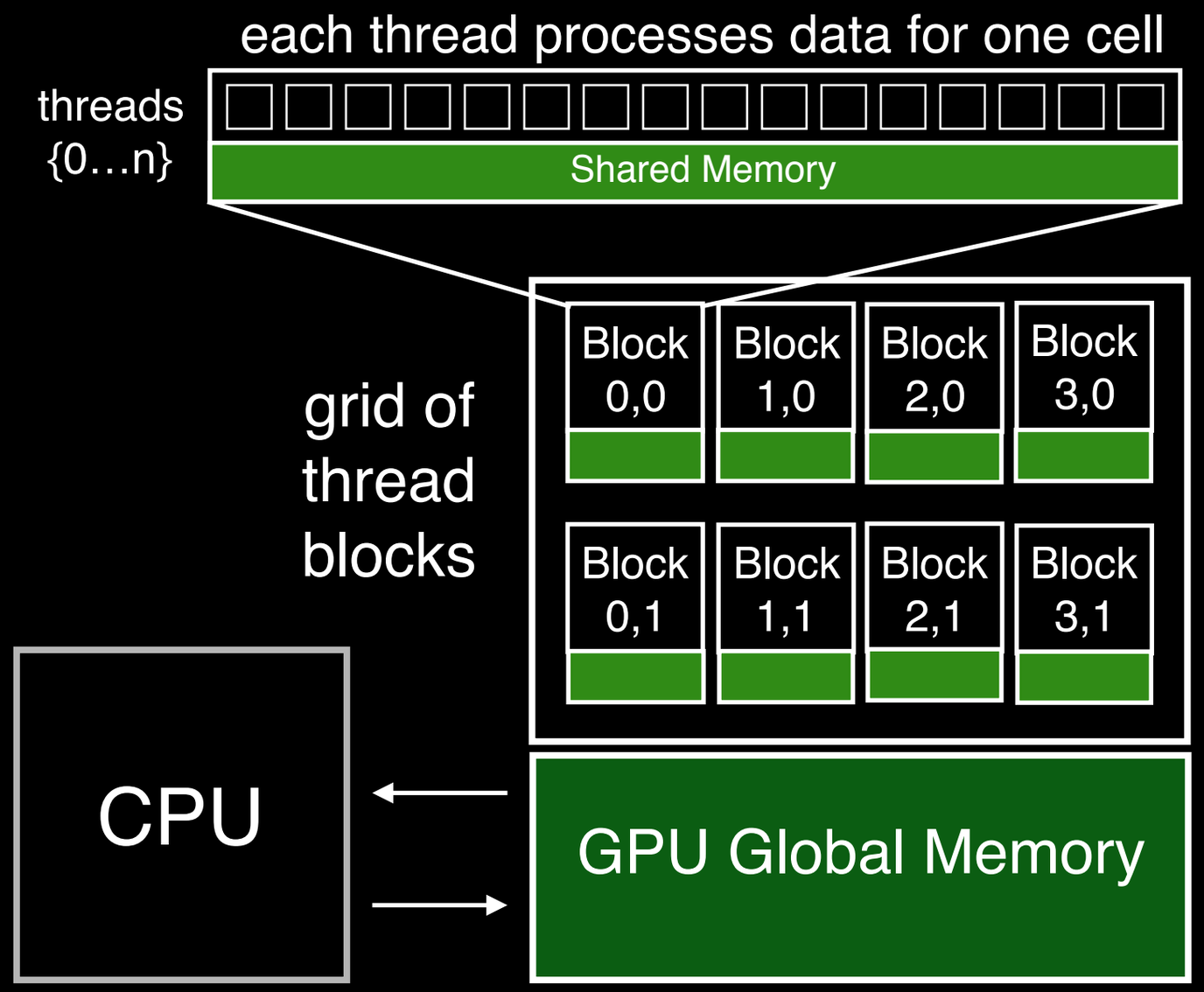


Grid code on a GPU



How does it work?

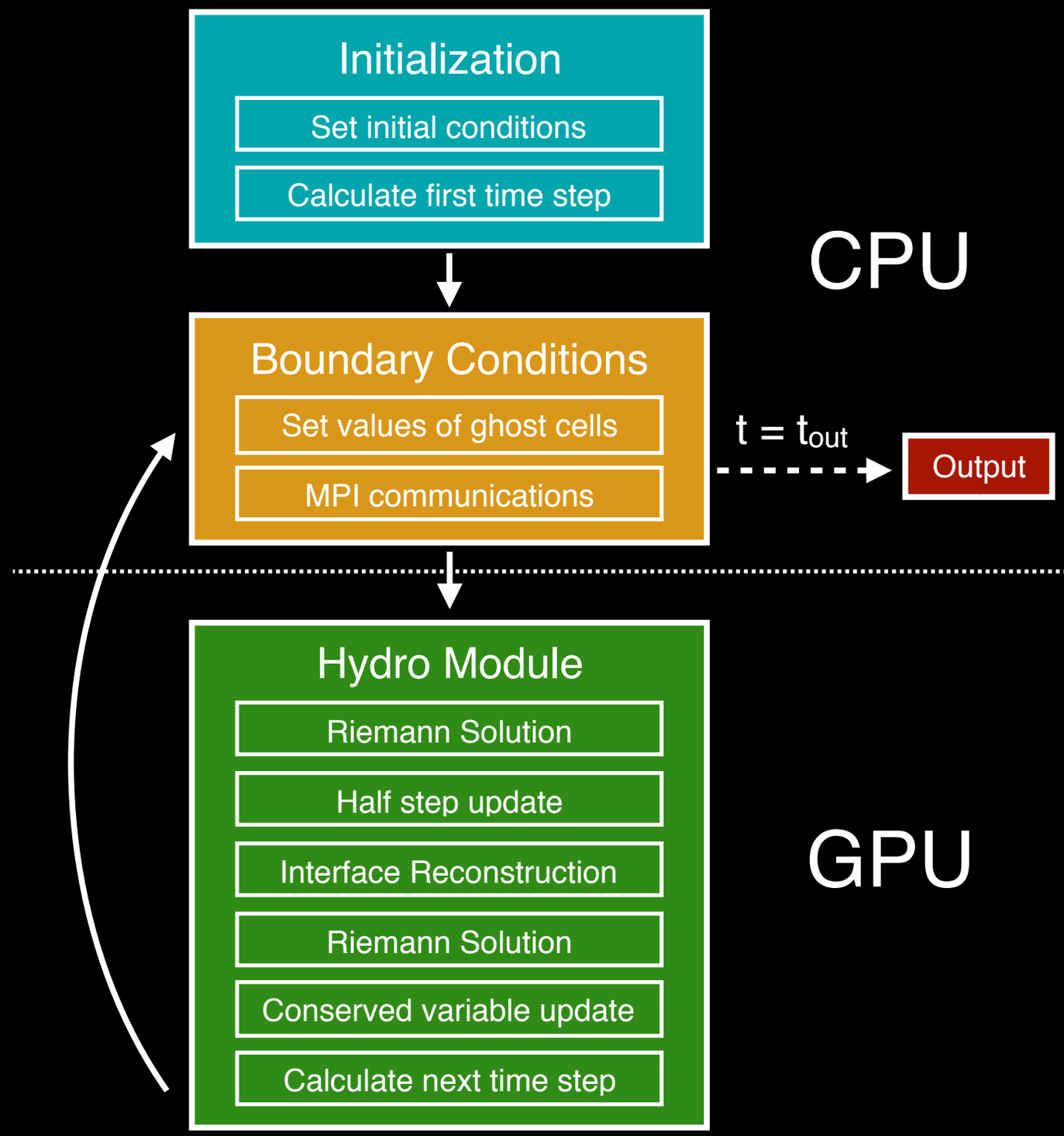
- GPU functions execute as CUDA kernels on a grid of thread blocks - each cell in the simulation is mapped to a single thread.
- Our design minimizes memory transfers between the CPU and GPU, reducing computational overhead.



How does it work?

Serial parts of code execute on the CPU

Parallel portions execute on the GPU

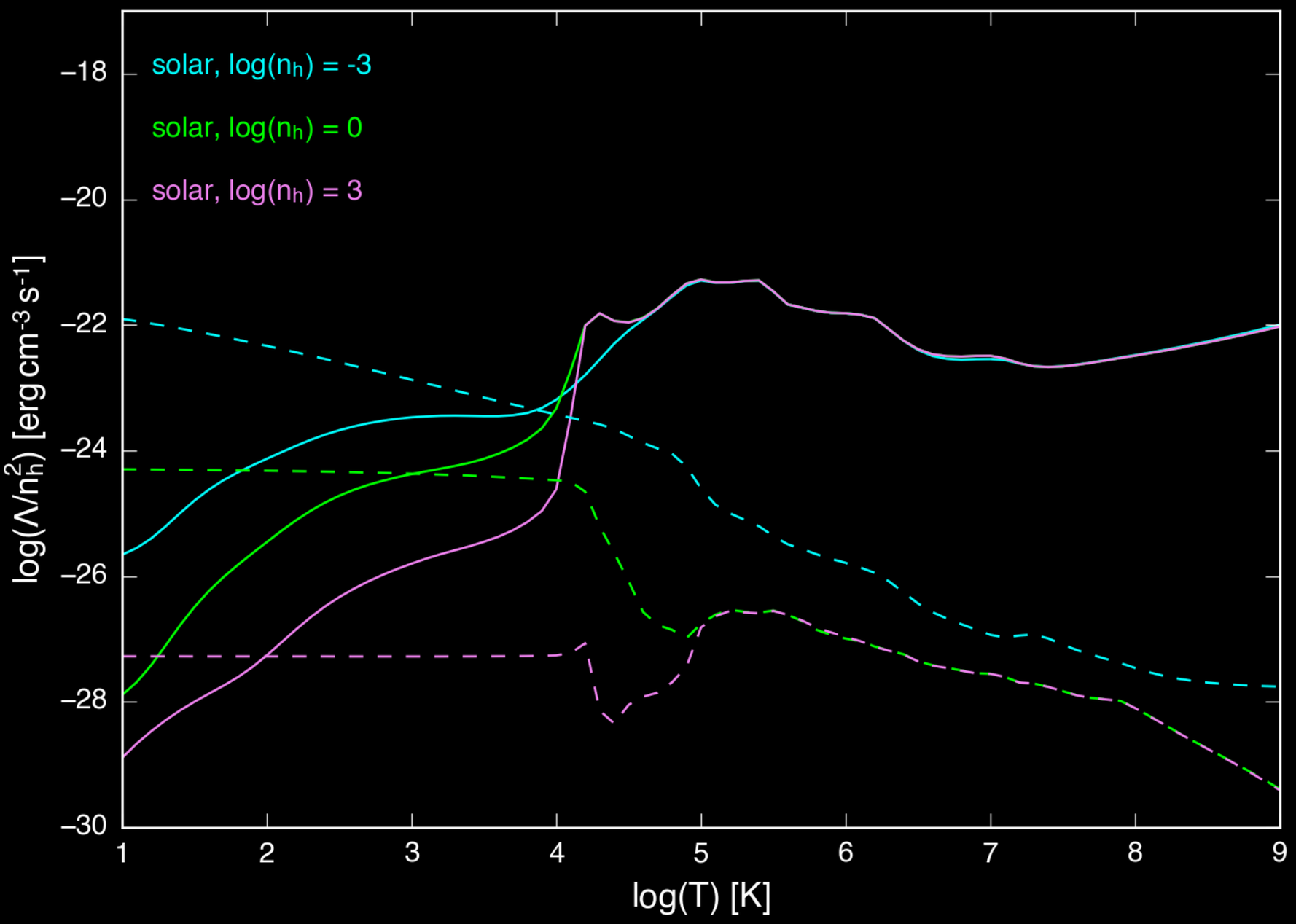


An aside: using texture mapping to accelerate cooling calculations.

What is texture mapping?

Skyrim, Bethesda Game Studios

Many astrophysical cooling calculations rely on multidimensional table lookups to calculate radiative cooling / heating rates.



Texture mapping is a natural fit to the cooling implementation.

1. Copy 2D cooling tables to texture memory on the GPU
2. Calculate density and temperature of gas
3. “Fetch” cooling and heating rates from the texture - bilinear interpolation comes for free!

```
__device__ double Cloudy_cool(double n, double T)
{
  double lambda = 0.0; // cooling rate, erg s^-1 cm^3
  double H = 0.0;      // heating rate, erg s^-1 cm^3
  double cool = 0.0;   // cooling per unit volume, erg / s / cm^3

  // fetch cooling and heating rates
  lambda = tex2D<float>(coolTexObj, T, n);
  H      = tex2D<float>(heatTexObj, T, n);

  // cooling rate per unit volume
  cool = n*n*(lambda - H);

  return cool;
}
```

How does it compare to the CPU version?

Loops/ Threads	GSL (CPU)	CUDA (GPU)
10^2	0.006 ms	0.023 ms
10^4	0.31 ms	0.023 ms
10^6	30 ms	0.023 ms

So basically, texture mapping makes our cooling calculations free.

Cholla:
Computational hydrodynamics on ll architectures

Cholla are also a group of cactus species that grows in the Sonoran Desert of southern Arizona.

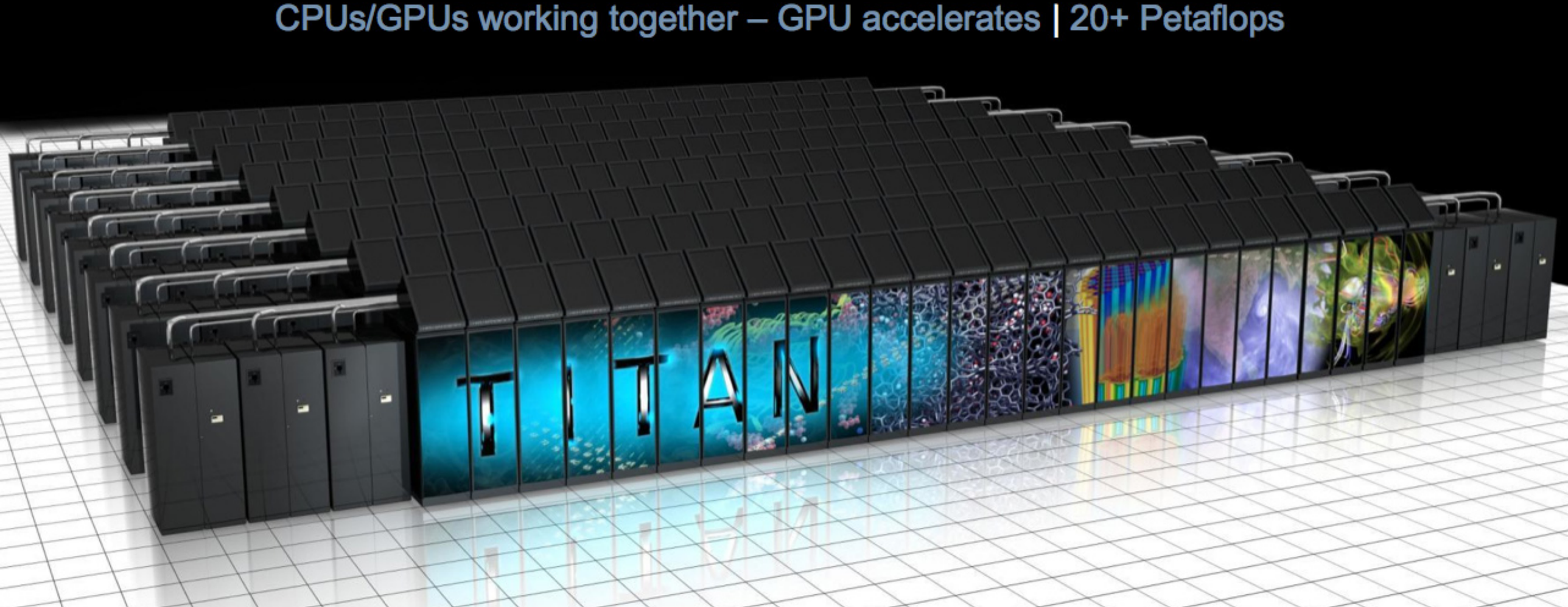
- A GPU-native, massively-parallel, grid-based hydrodynamics code (publicly available at github.com/cholla-hydro/cholla)
- Incorporates state-of-the-art hydrodynamics algorithms (unsplit integrators, 3rd order spatial reconstruction, precise Riemann solvers, dual energy formulation, etc.)
- Also includes optically-thin cooling and photoionization heating based on precomputed rate tables, and static gravity.

Schneider & Robertson (2015, 2017)

Cholla takes advantage of the world's most powerful supercomputers.

~~Titan~~: ~~Fastest Open Science Supercomputer in the US~~

Flagship accelerated computing system | 200-cabinet Cray XK7 supercomputer |
18,688 nodes (AMD 16-core Opteron + NVIDIA Tesla K20 GPU) |
CPUs/GPUs working together – GPU accelerates | 20+ Petaflops



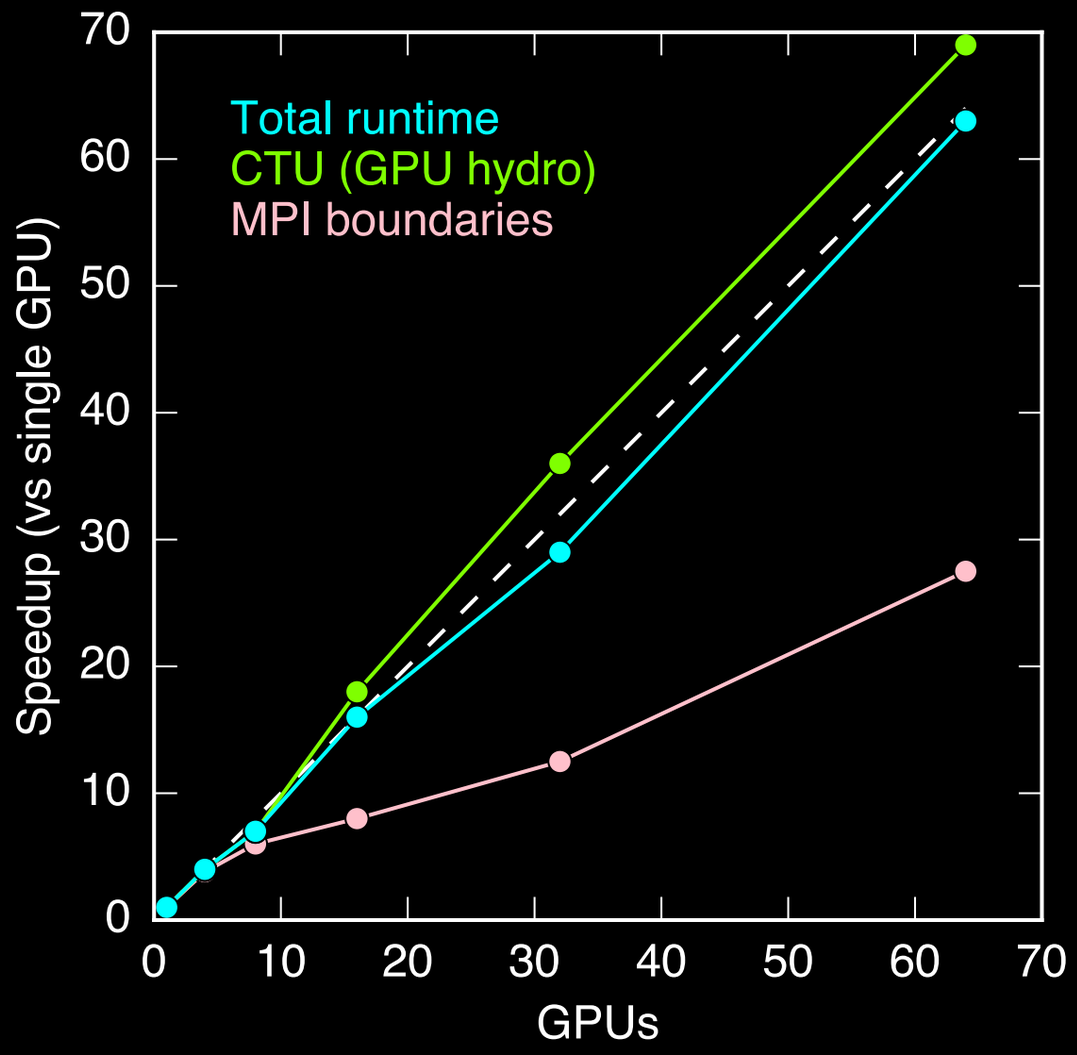
Cholla takes advantage of the world's most powerful supercomputers.

Summit:
Fastest Open Science Supercomputer in the World

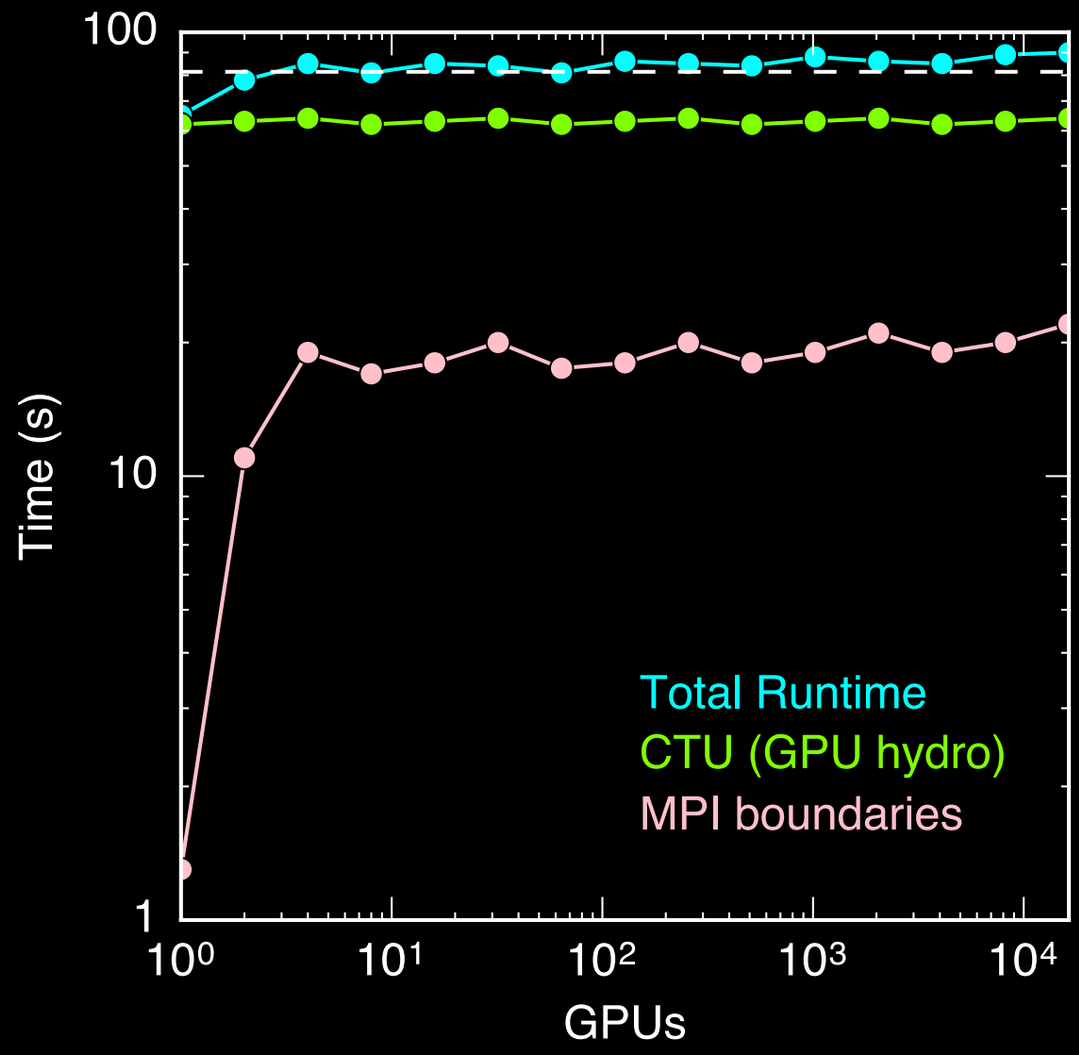
- **200 petaflops**
- **4600 nodes**
- **27,600 NVIDIA V100 GPUs**

Cholla Achieves Excellent Scaling

Strong Scaling test, 512^3 cells



Weak Scaling test, ~322^3 cells / GPU



Schneider & Robertson (2015)

Graphics Processors as a Scientific Tool

Advantages

- Optimized for fast execution of parallel tasks
- Blocked architecture easily transitions to new hardware models
- Specialized hardware functions are FAST
- Offloading computation to GPU leaves CPU free to perform other tasks
- Energy efficient!

Challenges

- Limited memory on GPU
- Need lots of computation to make up for data transfer and memory latency
- Blocked architecture not optimal for some problems

Graphics Processors as a Scientific Tool

Advantages

- Optimized for fast execution of parallel tasks
- Blocked architecture easily transitions to new hardware models
- Specialized hardware functions are FAST
- Offloading computation to GPU leaves CPU free to perform other tasks
- Energy efficient!

Challenges

- ~~Limited memory on GPU~~
- Need lots of computation to make up for data transfer and memory latency
- Blocked architecture not optimal for some problems

Graphics Processors as a Scientific Tool

Advantages

- Optimized for fast execution of parallel tasks
- Blocked architecture easily transitions to new hardware models
- Specialized hardware functions are FAST
- Offloading computation to GPU leaves CPU free to perform other tasks
- Energy efficient!

Challenges

- ~~Limited memory on GPU~~
- Need lots of computation to make up for data transfer and memory latency ☑️
- Blocked architecture not optimal for some problems

Graphics Processors as a Scientific Tool

Advantages

- Optimized for fast execution of parallel tasks
- Blocked architecture easily transitions to new hardware models
- Specialized hardware functions are FAST
- Offloading computation to GPU leaves CPU free to perform other tasks
- Energy efficient!

Challenges

- ~~Limited memory on GPU~~
- Need lots of computation to make up for data transfer and memory latency ☑️
- ~~Blocked architecture not optimal for some problems~~

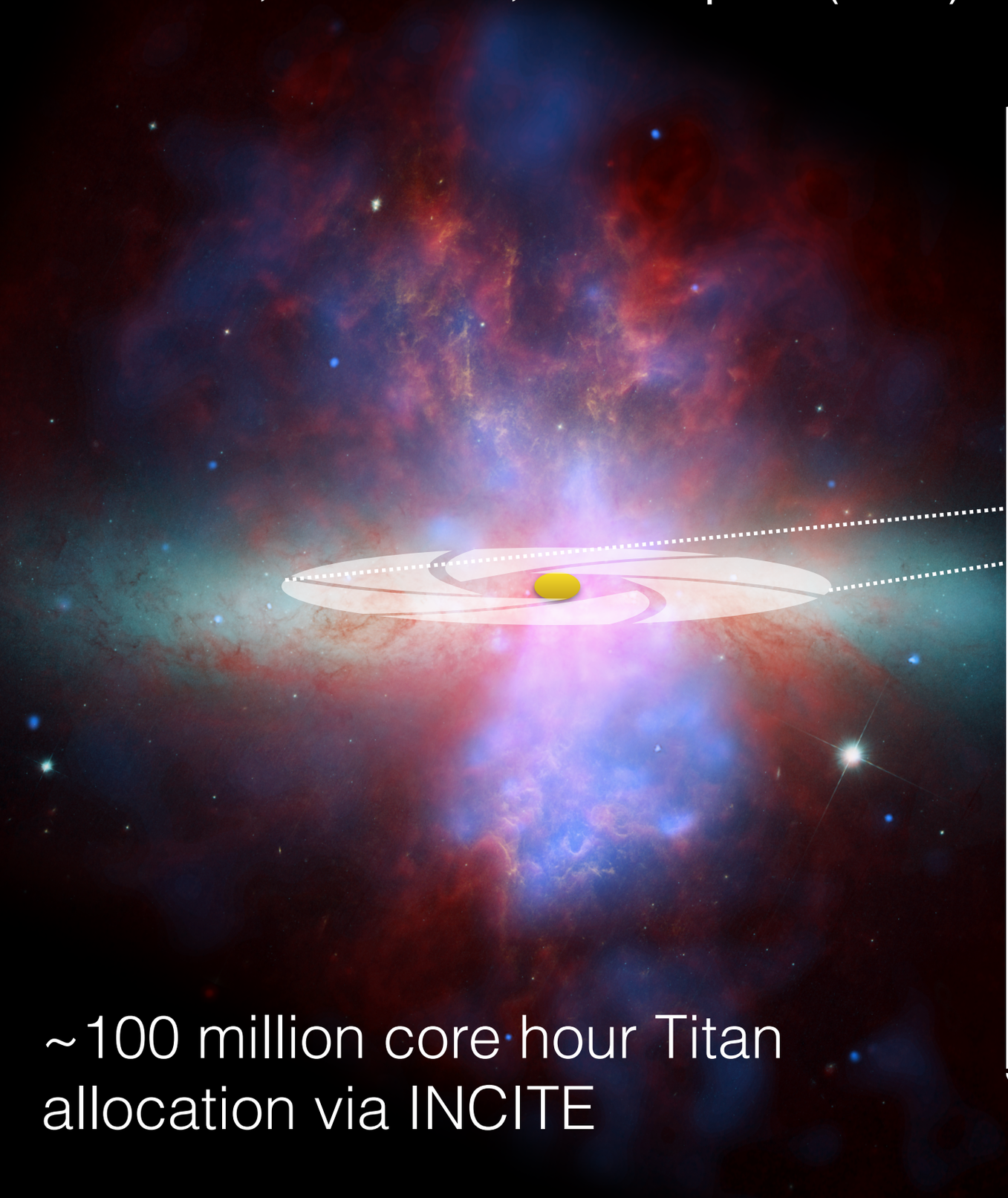
Starburst Galaxy M82

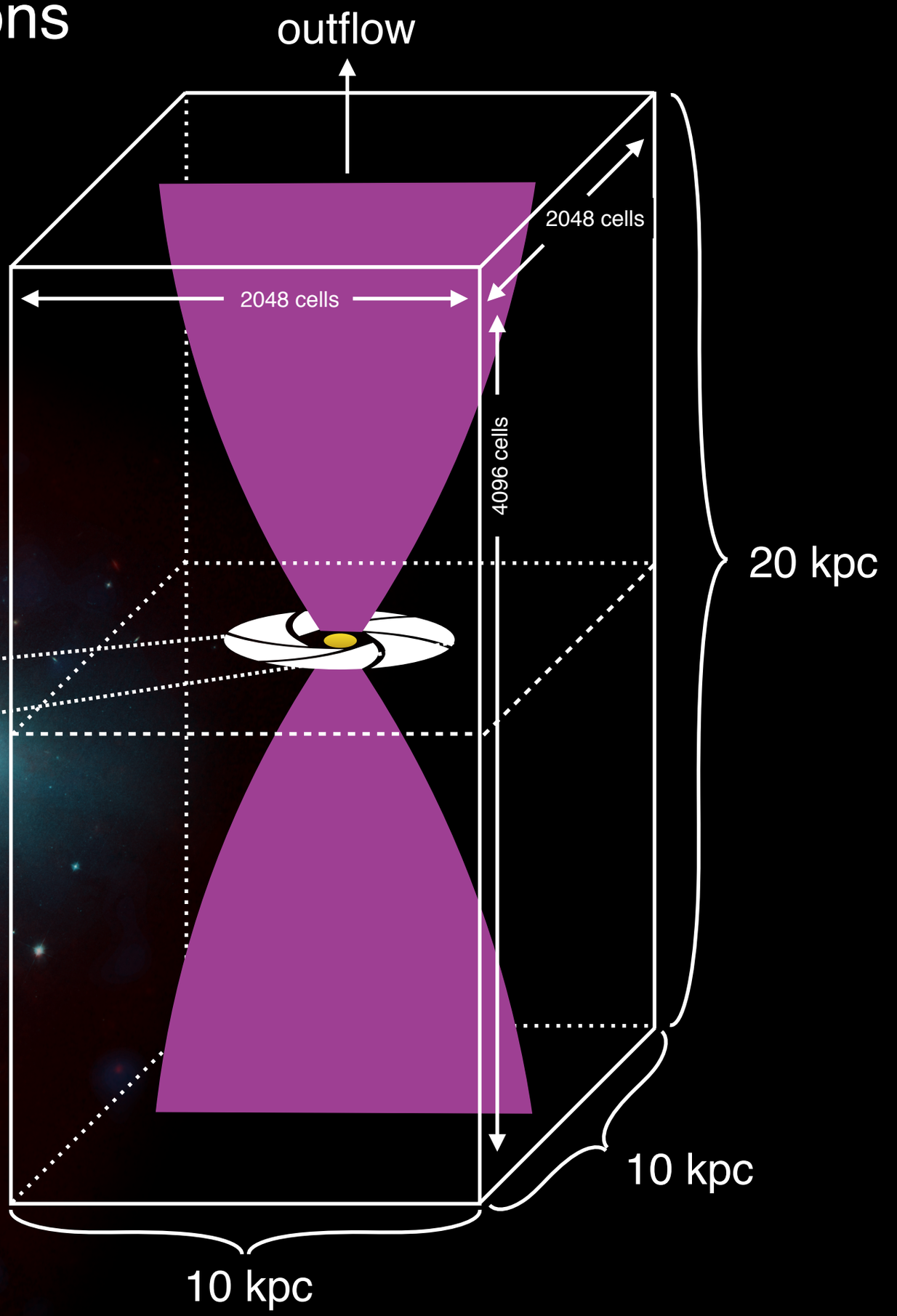
Image credit: NASA

Cholla Galactic OutfLow Simulations

Schneider & Robertson (2018)

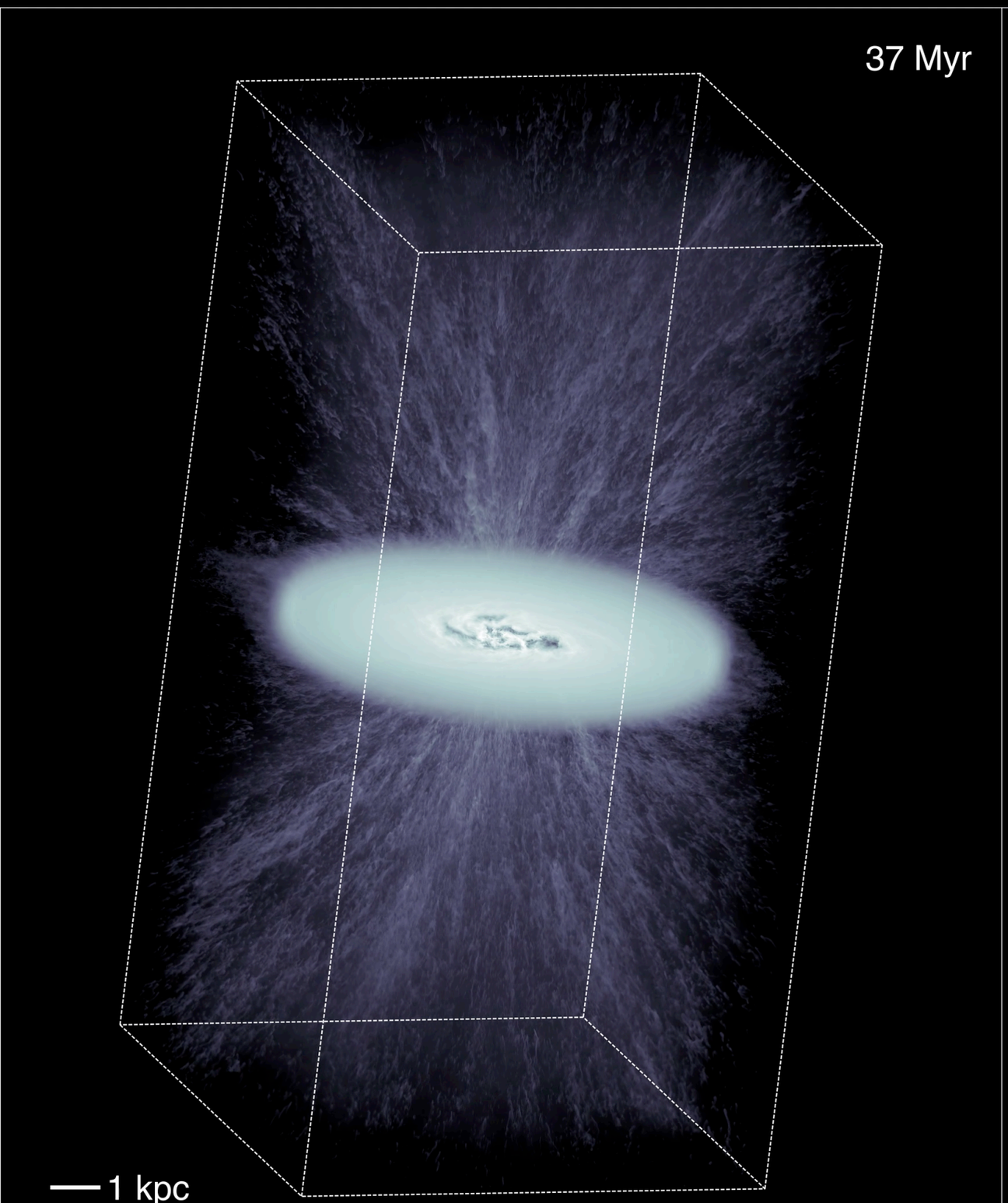
Schneider, Robertson, & Thompson (2018)



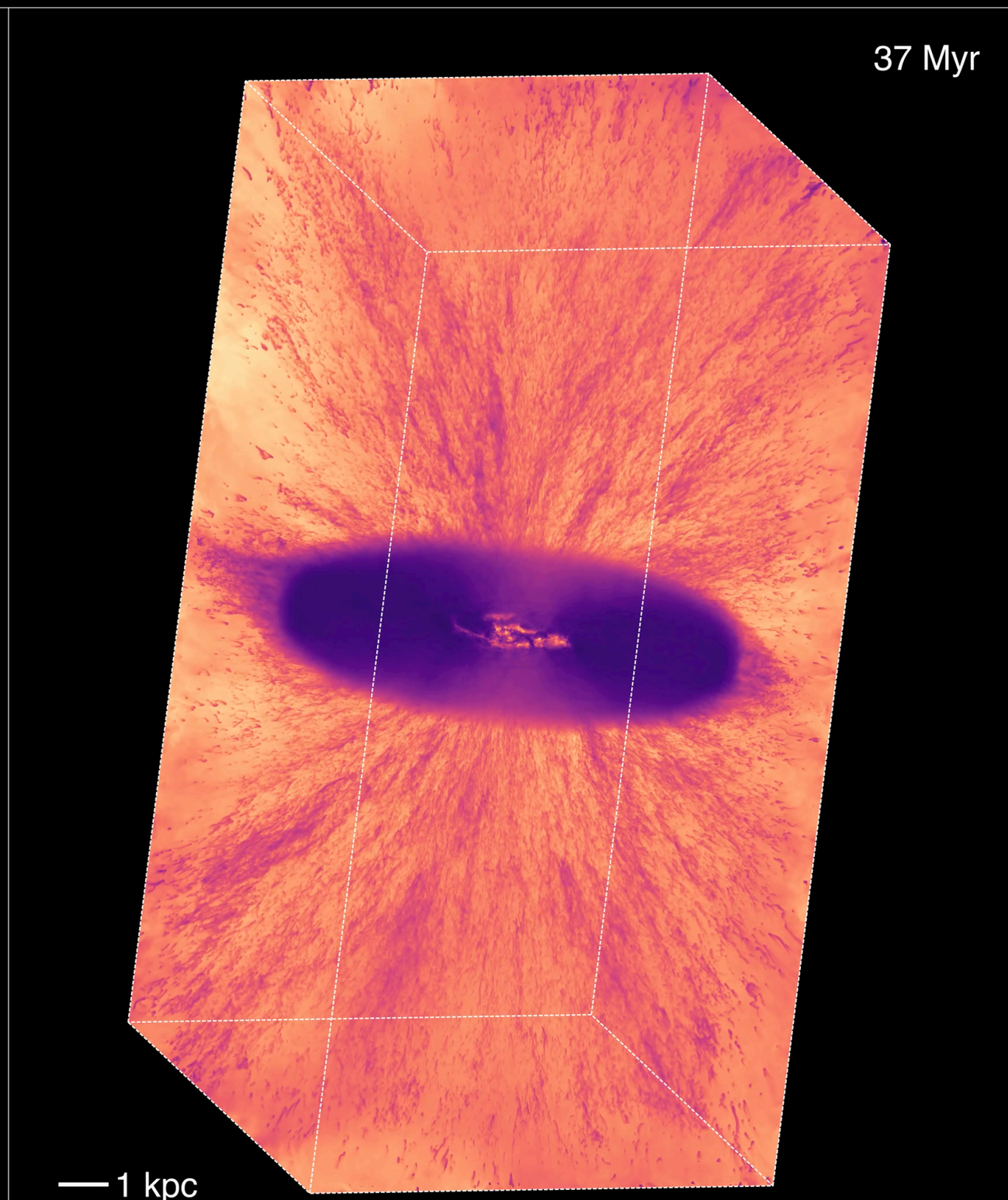


~100 million core hour Titan allocation via INCITE

CGOLS Simulation - 17 billion cells

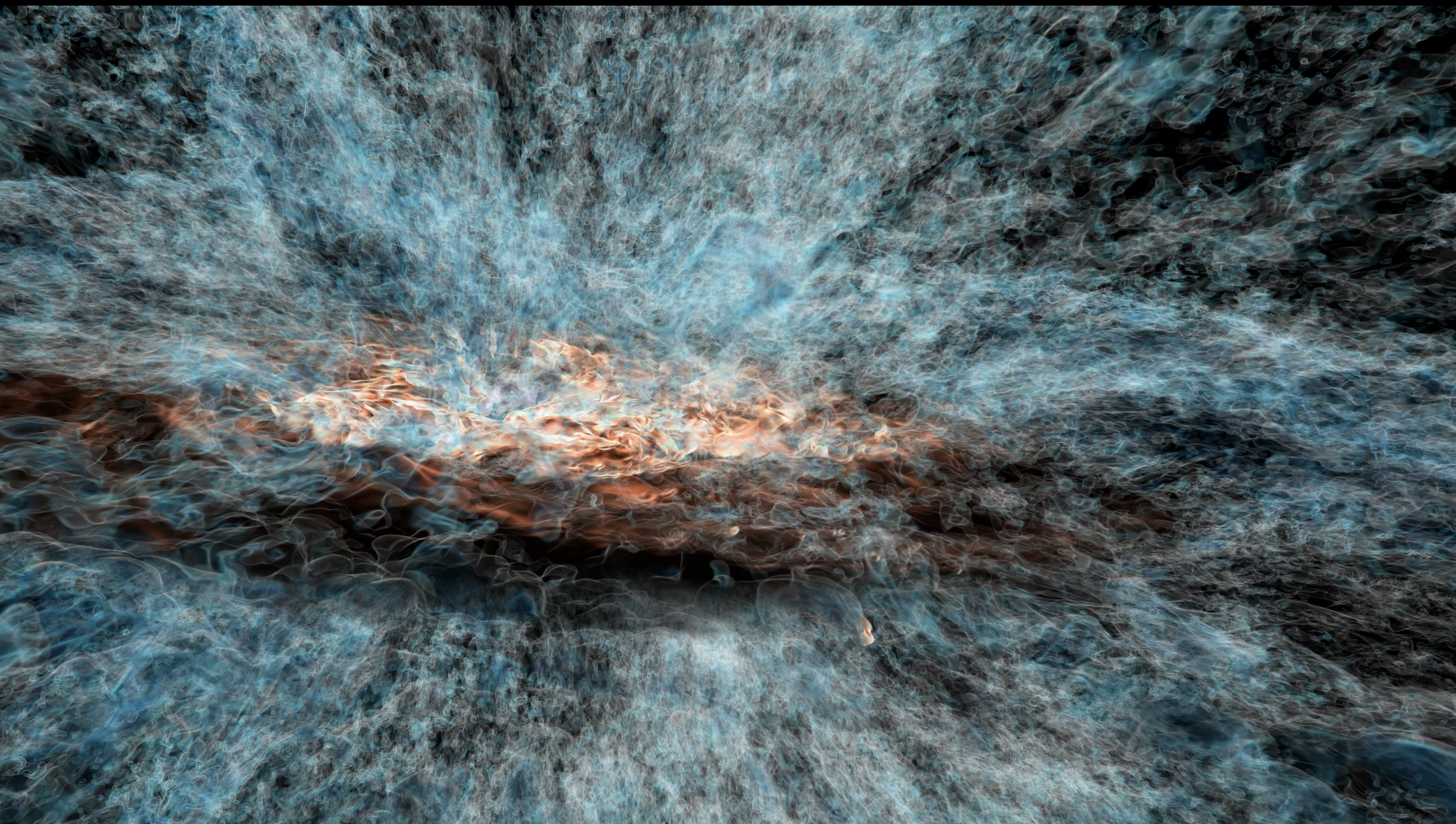


Density Projection



Temperature Projection

Cholla plus IndeX - Next Generation Data Visualization



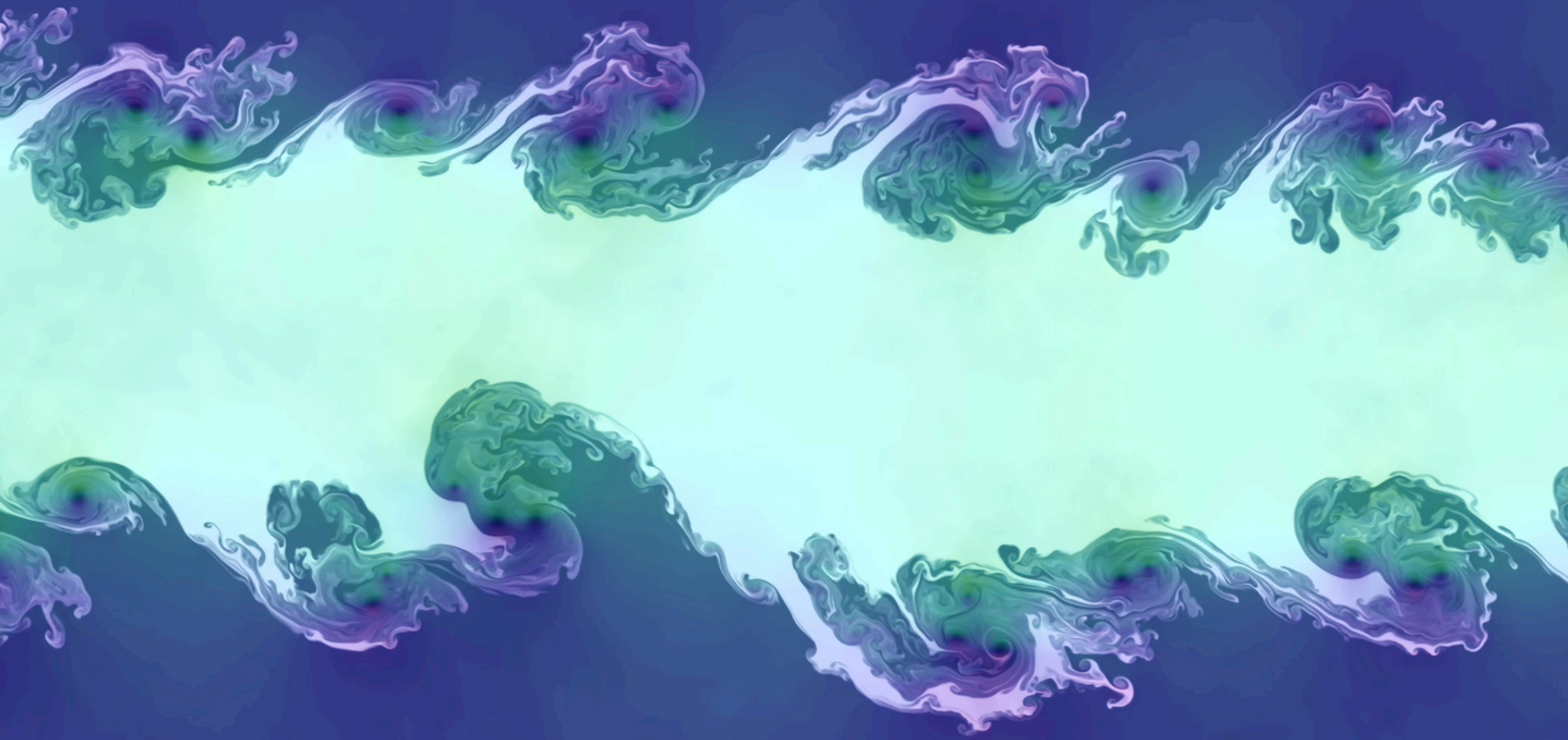
Cholla plus IndeX - Next Generation Data Visualization

M82, real data

M82, CGOLS simulation

2D Kelvin Helmholtz test

3840 x 2160 Resolution



Thanks!