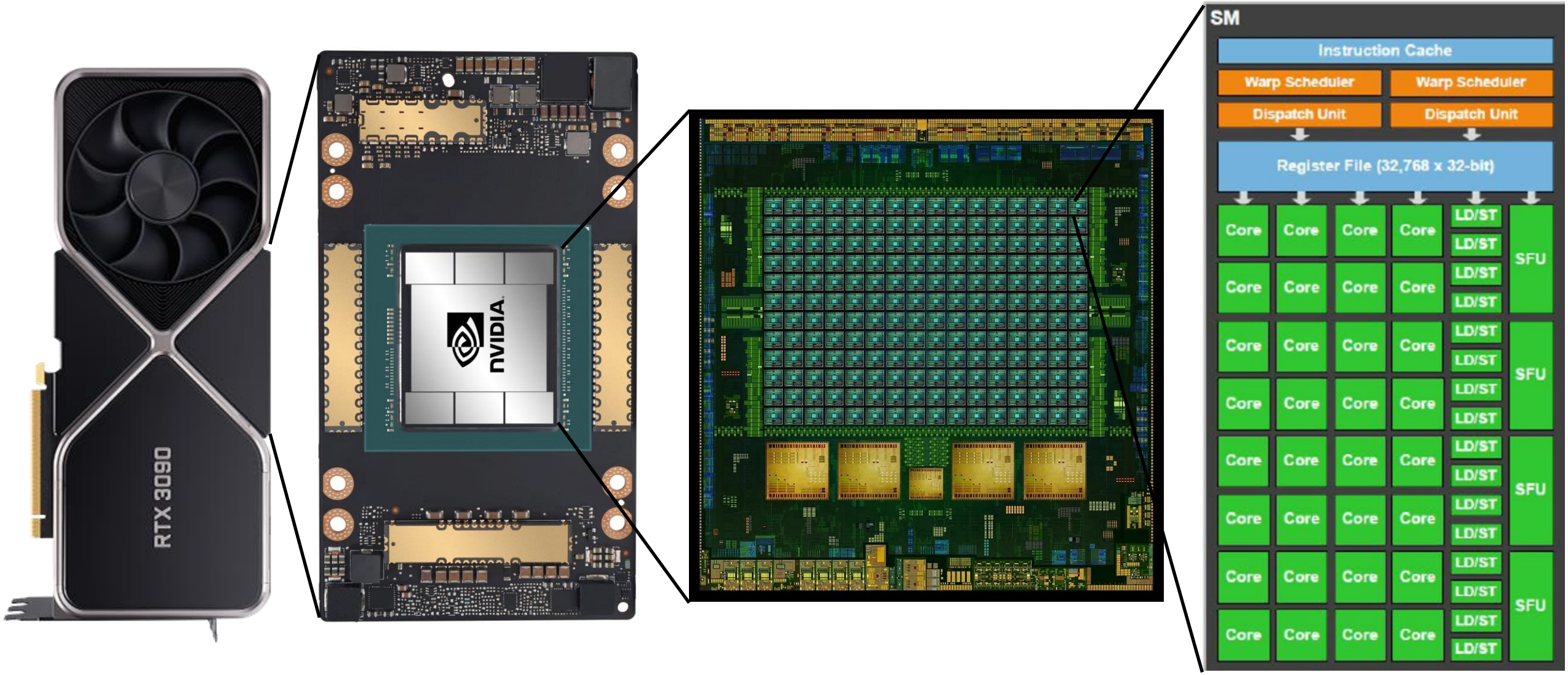
A GLIMPSE ON GPU IN ASTROPHYSICS

Lile Wang （王力乐）
KIAA/PKU


msi



What is a Graphics Processing Unit?

- GPUs: Jet liners

  SIMD (before Turing)

  High latency mem access

  High throughput

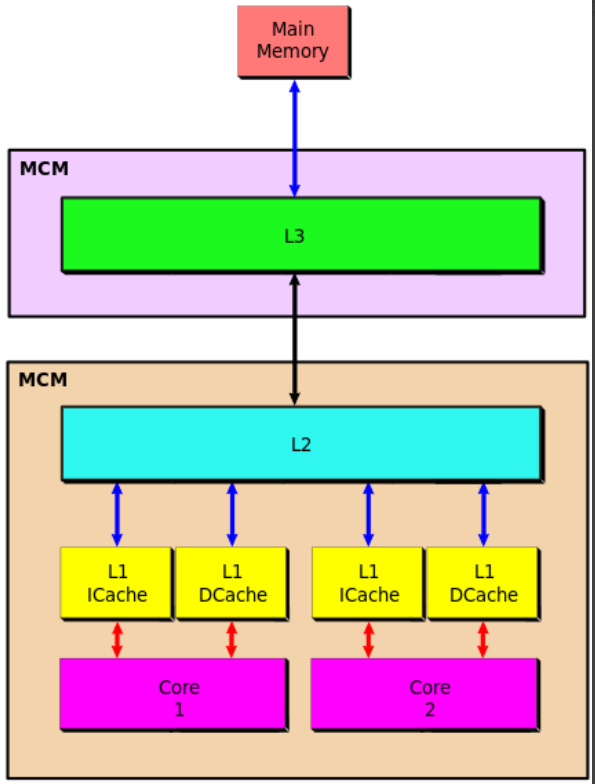
  Lower (?) frequency

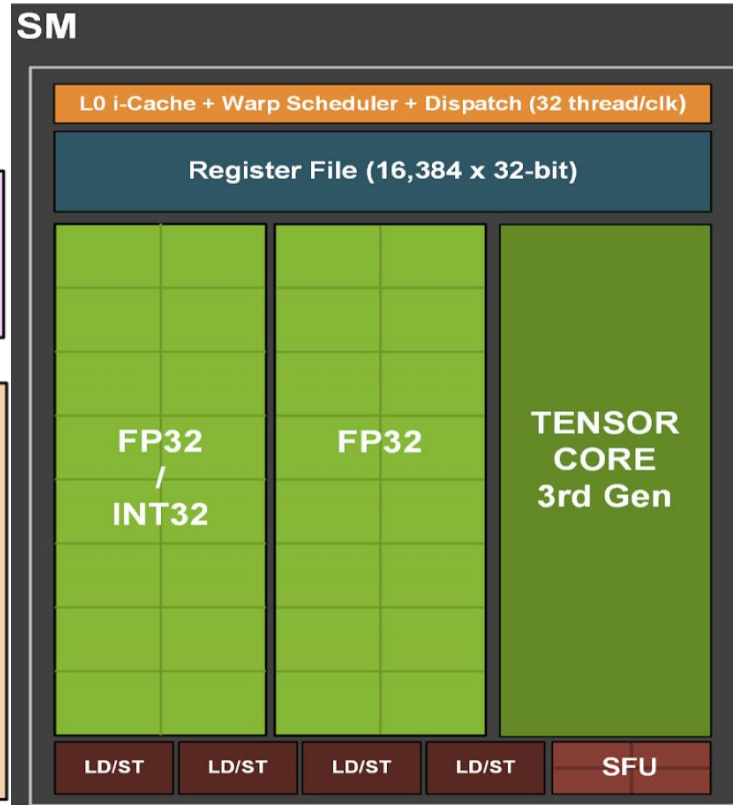
  Worse versatility

- CPUs: Jet fighters

Slow, but Massive

- Typical CPU (AMD Ryzen 7 5800X):
- 8 Cores / CPU
- Register: 64 bits / Unit
- L1 Cache: 64 KB / Core
- L2 Cache: 512 KB / Core
- L3 Cache: 32 MB overall





Typical GPU (NVIDIA RTX 3080):

- 82 SM / GPU
- 16 Units / SM
- Shared I-Cache (!!!)
- Register: 64 kB / SM
- L1 Cache: 128 KB / SM (64 KB / SM <= Turing)
- L2 Cache: 18 MB overall

Advantages and Disadvantages

NVIDIA CUDA

AMD ROCm

Map（映射）

- Distribute computation tasks over all computation units properly

Cache（缓存）

- Minimize global graphics memory access, eliminate unecessary ones

Stream（执行流）

- Coordinate computation and data transfer, avoid waiting

Basic Guidelines

CUDA

- Mature, Reliable (?), Powerful, Modernized (C++-20)
- Limited to NVIDIA

HIP/ROCm

- Almost IDENTICAL to CUDA, e.g. cudaMemcpy vs hipMemcpy
- HIP-CPU: Use MT CPU to “emulate” GPUs (Useful!)

OpenCL

- Deceased support from major manufacturers (and decreasing)
- cHIP-STAR: OpenCL + SPIR-V --> HIP

OneAPI

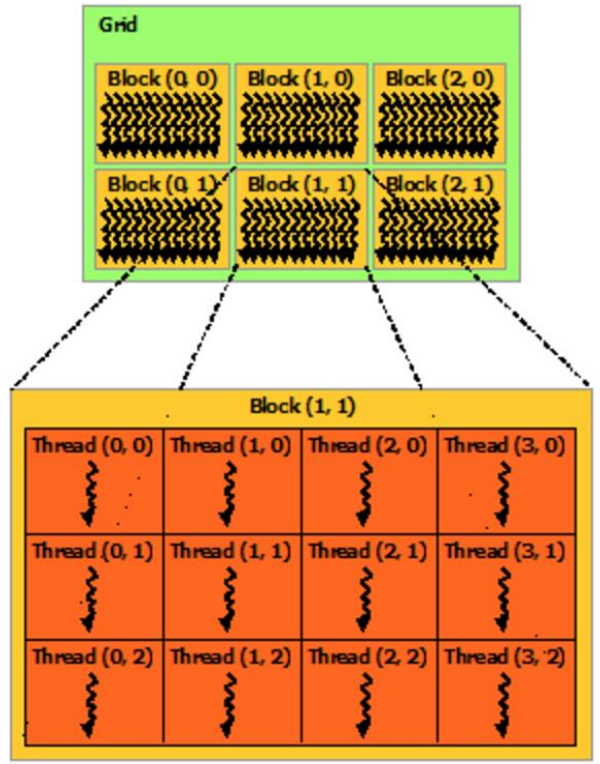
- Premature: programming model, system quality...
- cHIP-STAR: OneAPI --> HIP

MUSA

- Mostly identical to CUDA
- Premature support for computational purposes

Major Programming Models

```
#include <iostream>

__global__ void kernel( float * a, float * b, float * tgt )
{
    const int idx = threadIdx.x + blockDim.x * blockIdx.x;
    c[ idx ]      = a[ idx ] + b[ idx ];
    return;
}

int main(  )
{
    // Memory allocation, initialization
    kernel <<< n_block, n_thread >>> ( a, b, c );
    // Resource releasing and garbage collection
    return 0;
}
```



Major Programming Models for GPUs

Instruction:

res[?] = a[?] + b[?];

Block 0

w/ N Threads

...

Block m

w/ N Threads

Thread 0

? -> 0

Thread 1

? -> 1

...

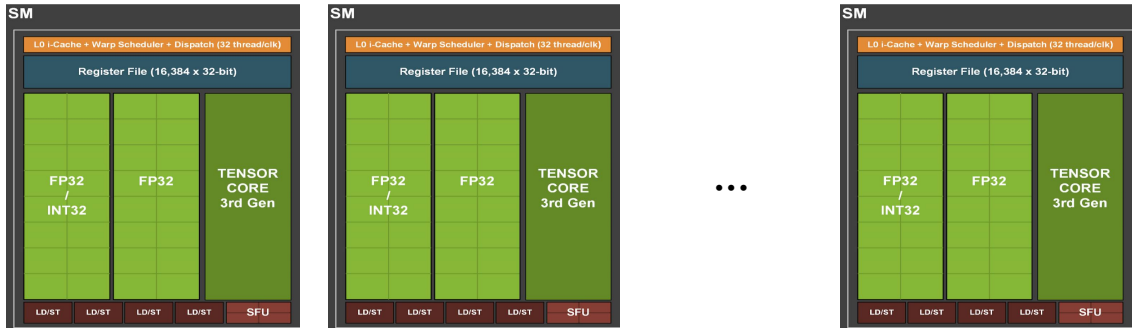
Thread (N-1)

? -> (N-1)

Thread n

? -> m * N + n

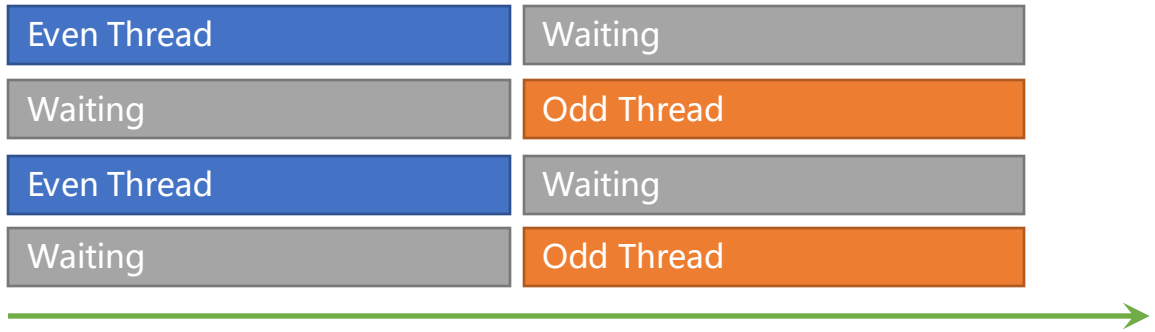
```
template < typename f_T > __global__ void kernel
( f_T * res,  const f_T * a, const f_T * b )
{
    const int idx = blockIdx.x * blockDim.x + threadIdx.x;
    res[ idx ] = a[ idx ] + b[ idx ];
    return;
}
```

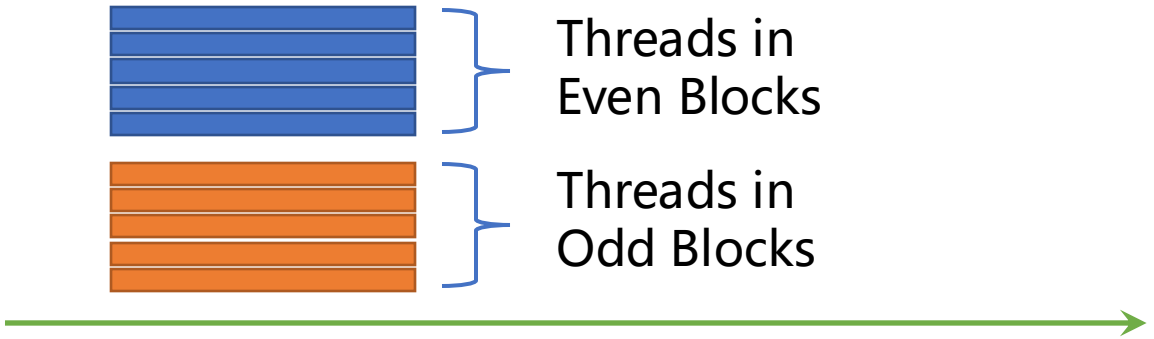


Naive Mapping: Uniform Task Dispatching

```
__global__ void kernel( ... )
{
    const int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if( idx % 2 == 0 )
    {
            // Operations for even indices;
    }
    else
    {
            // Operations for odd  indices;
    }
    return;
}
```

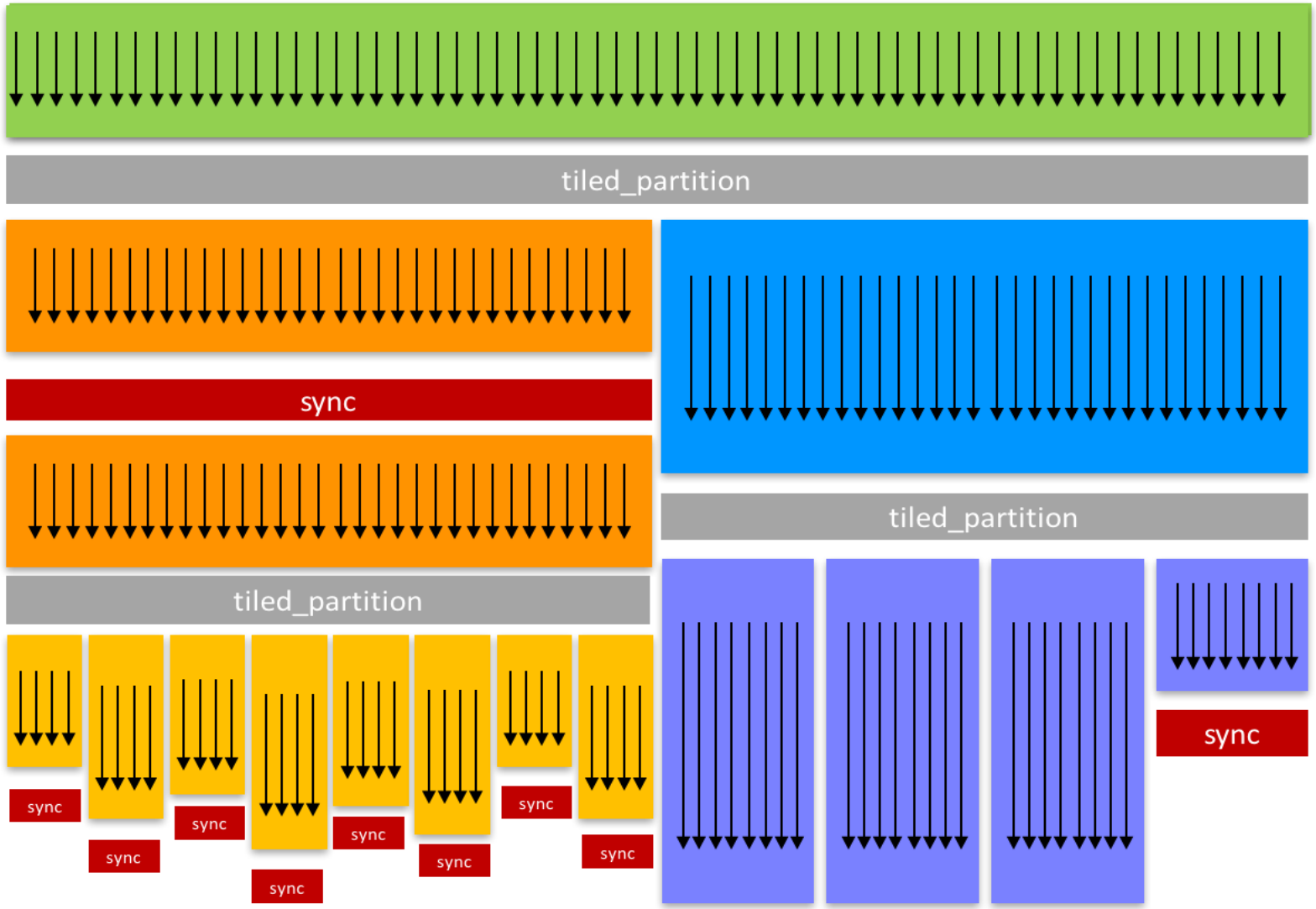
```
__global__ void kernel( ... )
{
    const int idx_0 = blockIdx.x * blockDim.x + threadIdx.x;
    if( blockIdx.x & 1 )
    {
        const int idx = 2 * idx_0;
        // Operations for even indices;
    }
    else
    {
        const int idx = 2 * idx_0 + 1;
        // Operations for odd  indices;
    }
    return;
}
```





Less Naive Mapping: Dispatch More Wisely

- Finer or broader
- Single-thread granularity
- Why is it possible? (hint: Cache)
- CUDA ONLY (for now)



Finer Thread Manipulation: Cooperative Group

Register (“L0”): Almost No

MIS

L1: < 10 Clock cycles (128 kB/SM)

MIS

L2: Invisible to Programmers (< 1

MIS

“Global” Graphics Memory:
~500 clock cycles

GRAM Data

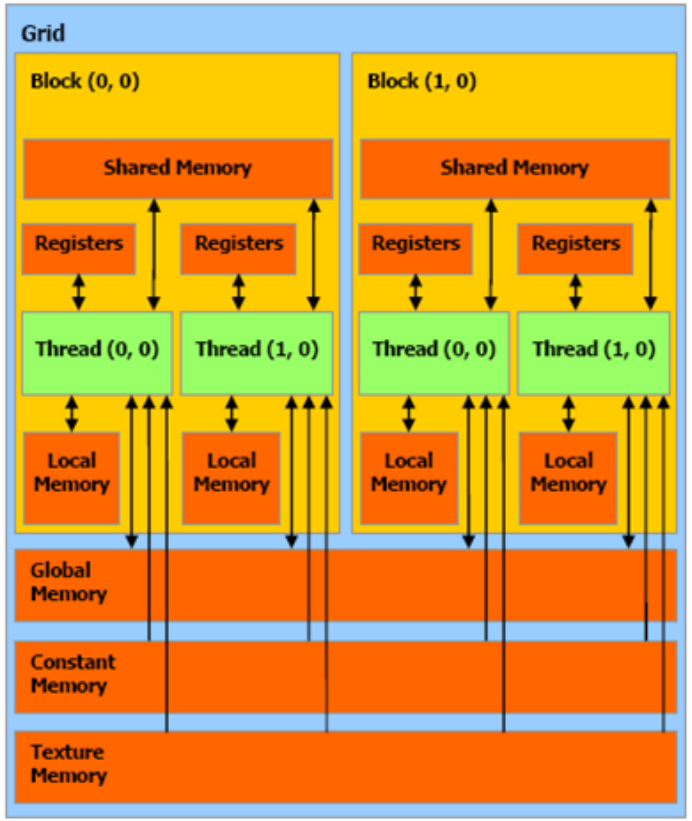
System memory
(or anything via PCI-E buses):

Inter-node
COMM

Inter-proces and Inter-node:
sub-millisecond

Manual L1 Control
(“Shared

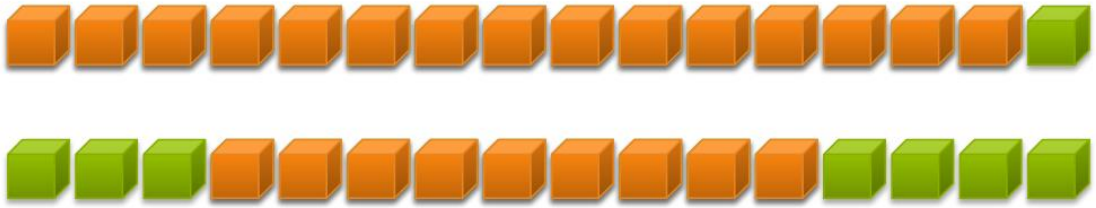
__constant__ and
Kernel Arguments



Cache: Ubiquitous Structures

Example: Convolution, etc. in 1D

- Assuming a logical radius of 3:
  Each data point used 3 times



Example: Minimum Timestep

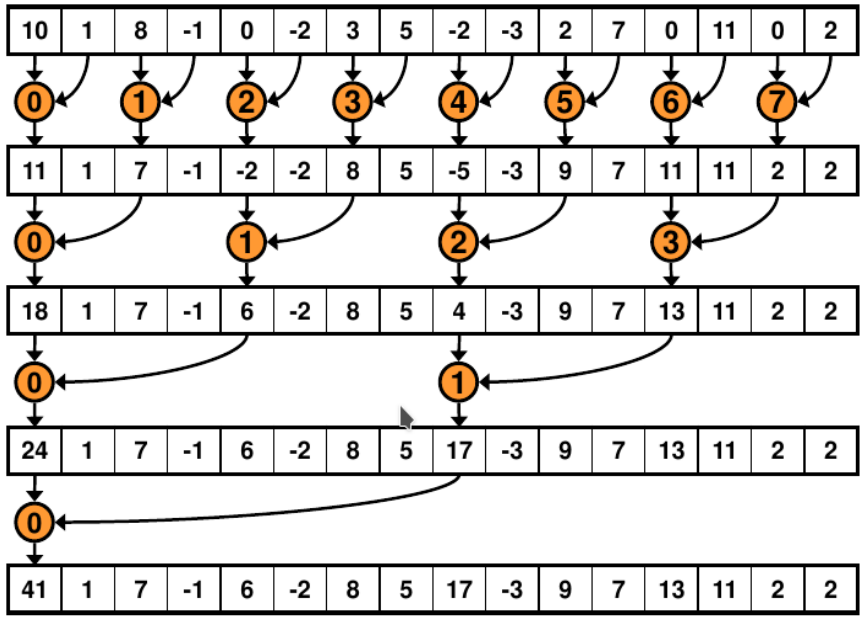
- Binary reduction for the minimum

```
__global__ void kernel_shared_static( ... )
{
   __shared__ float var[ 64 ];
   // Operations...
   return;
}

__global__ void kernel_shared_dynamic( ... )
{
   extern __shared__ float var[ ];
   // Operations...
   return;
}

int main( )
{
   // ...
   kernel_shared_static  <<< n_bl, n_th, 0    >>> ( ... );
   kernel_shared_dynamic <<< n_bl, n_th, n_sh >>> ( ... );
   // ....
}
```

Shared Memory: L1 Cache with Manual Ctrl

Example: Reduction of Arrays

- Reduction for the sum at $O(\log N)$
- Avoid congesting global atomicAdd




```
__global__ void kernel_shared_static( ... )
{
   __shared__ float var[ 64 ];
   // Operations...
   return;
}

__global__ void kernel_shared_dynamic( ... )
{
   extern __shared__ float var[ ];
   // Operations...
   return;
}

int main( )
{
   // ...
   kernel_shared_static  <<< n_bl, n_th, 0    >>> ( ... );
   kernel_shared_dynamic <<< n_bl, n_th, n_sh >>> ( ... );
   // ....
}
```

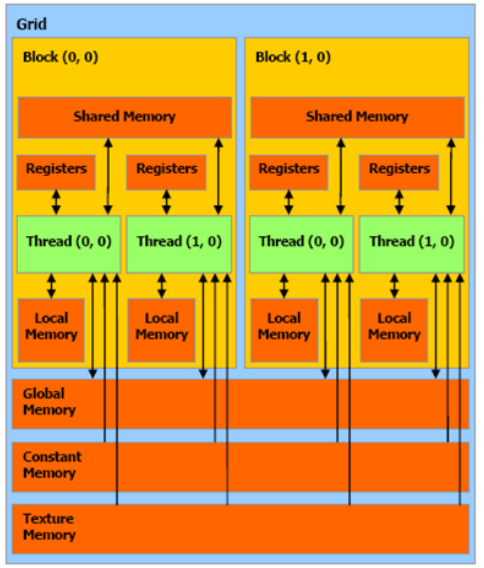
Shared Memory: L1 Cache with Manual Ctrl

Communicate

- The best way that threads communicate with in a block

Race Conditions

- Well-defined shared-memory model: Pay extra attention race conditions!



Local Dynamic Memory

- Shared memory is the only way of reserving dynamic memory locally

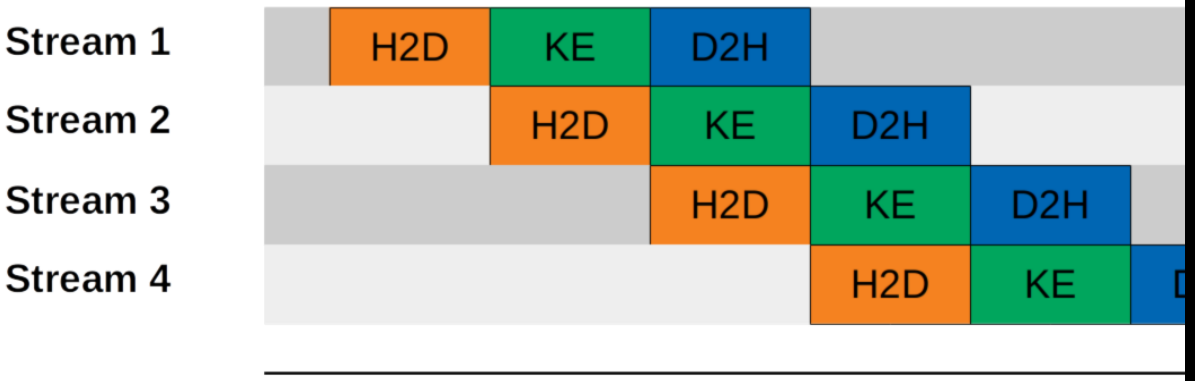
Reduce Usage

- Reduce shared memory usage when possible for more context on an SM

More About Shared Memory

Serial Model

Concurrent Model




```
__global__ void kernel( ... )
{
  // Operations...
  return;
}

int main( )
{
  // ...
  cudaMemcpyAsync( p_device[ 0 ], p_host[ 0 ], n_bytes,
                   cudaMemcpyDefault, stream_0 );
  kernel <<< n_bl, n_th, n_sh, stream_0 >>> ( ... );

  cudaMemcpyAsync( p_device[ 1 ], p_host[ 1 ], n_bytes,
                   cudaMemcpyDefault, stream_1 );
  kernel <<< n_bl, n_th, n_sh, stream_1 >>> ( ... );
  // ....

  cudaStreamSynchronize( stream_0 );
  cudaStreamSynchronize( stream_1 );

  return 0;
}
```

Time

Streams: Higher-Level Coordination

Pinned Memory

- Unpinned memory may be paged out
- Use cudaMallocHost and cudaFreeHost

Distributing and Dispatching Tasks Properly

- If memory exchange is necessary, hide it behind computation
- Divide the computation operations when necessary

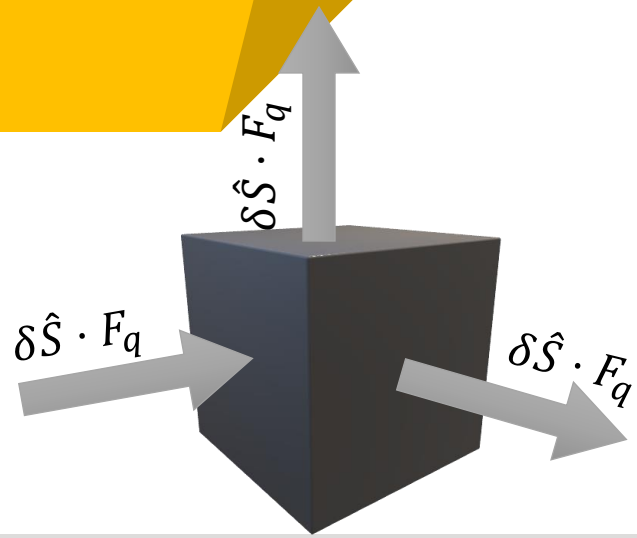
More Possibilities

- “Virtual memory” for the GPUs
- Inter-process or Inter-node communications

More About Streams

Finite Volume Method

- Spatial discretization: Control volume
- Temporal discretization: Explicit(!)



- In a "control volume", the integral of conserved quantity

$$\Delta \int q \, \mathrm{d}V = \sum_{\text{surface}} \int \mathrm{d}t \int \mathrm{d}\hat{\mathrm{S}} \cdot F_q$$

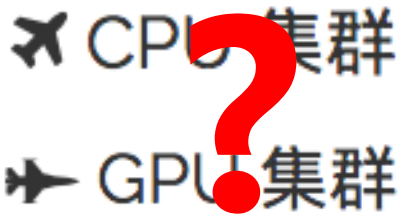
- Finite volume discretization:

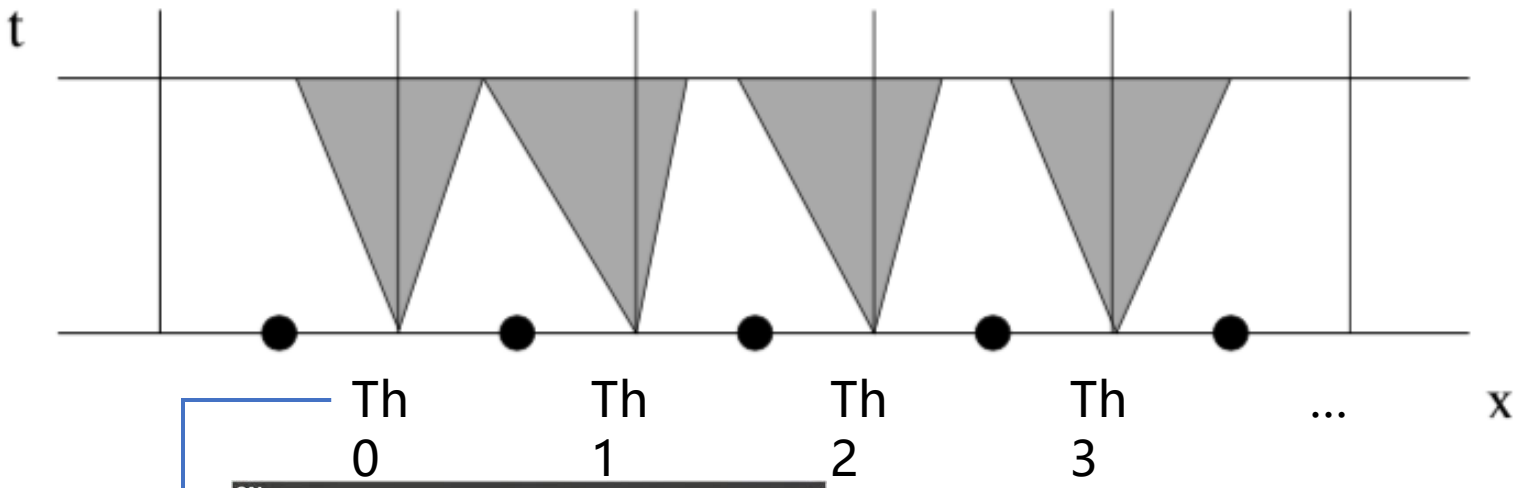
$$\Delta \int q \, \mathrm{d}V \simeq \delta V \Delta q$$

$$\int \mathrm{d}\hat{\mathrm{S}} \cdot F_q \simeq \delta\hat{S} \cdot F_q$$

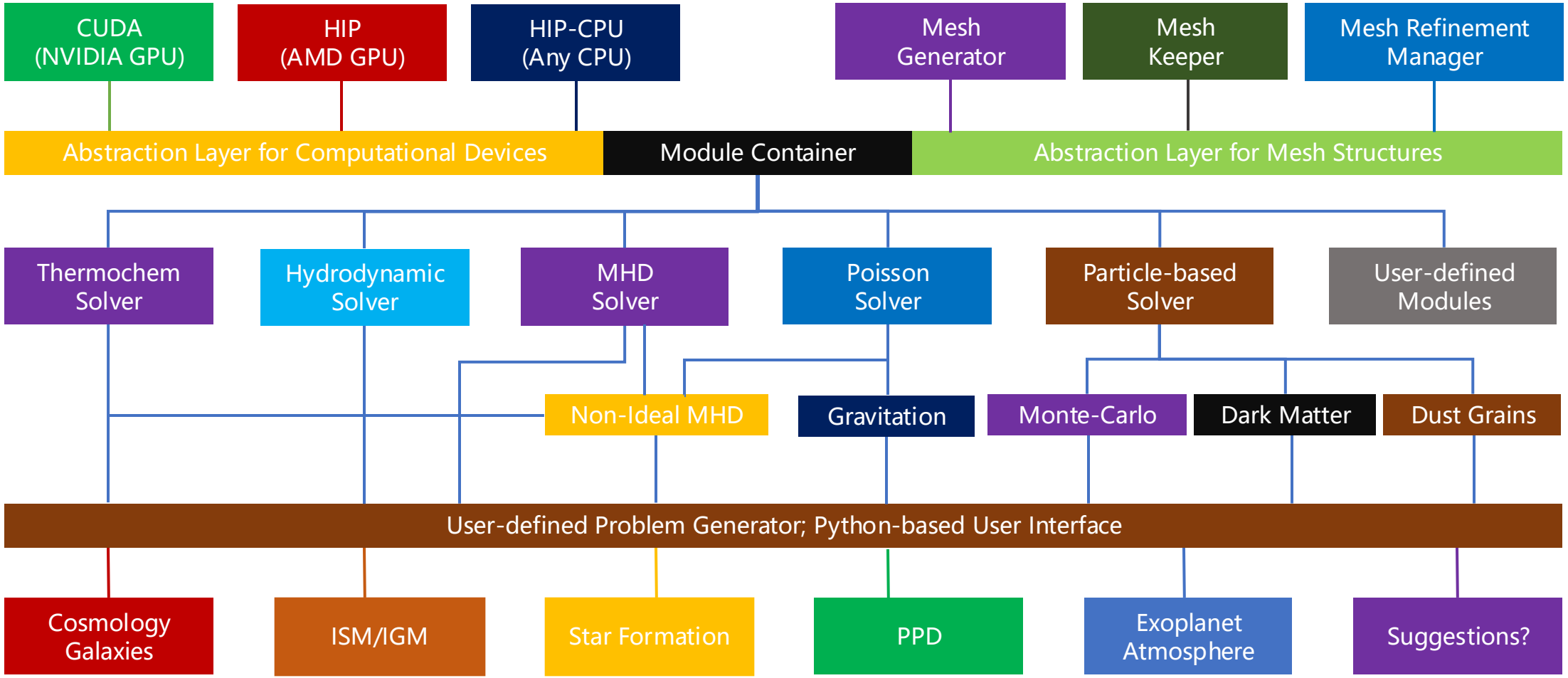
$$\int \mathrm{d}t \rightarrow \text{"Initial Value Problem"}$$

- Godunov method naturally suitable for parallelization
- GPU: The de-facto computational device for this
- However: No adequate implements





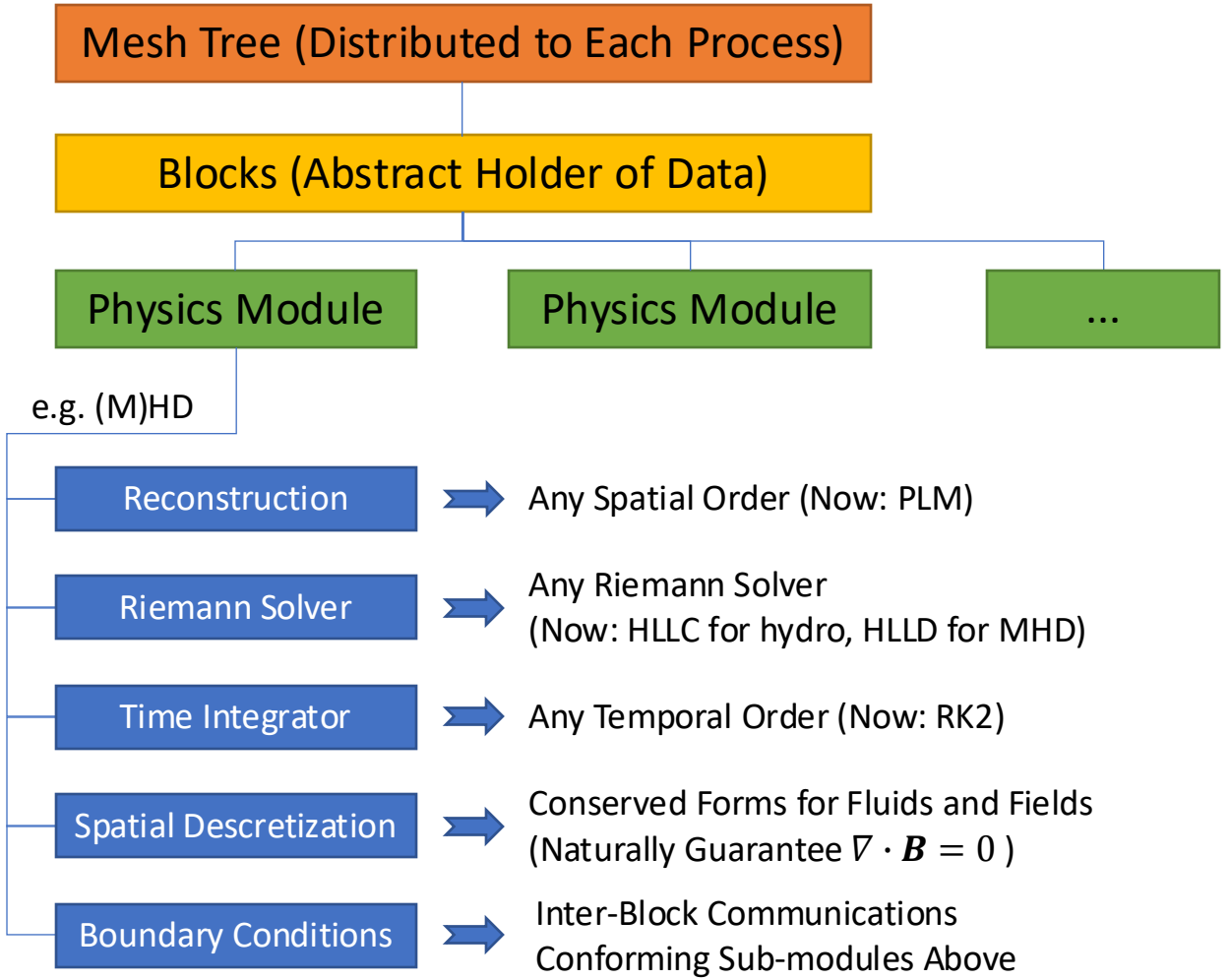
Use GPUs for Godunov Method

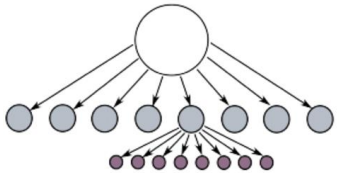


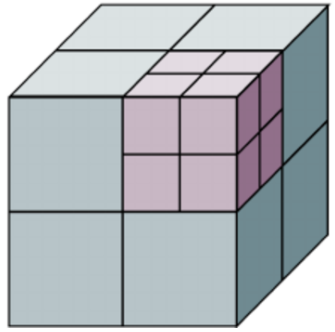
New Heterogeneous Code: Structures

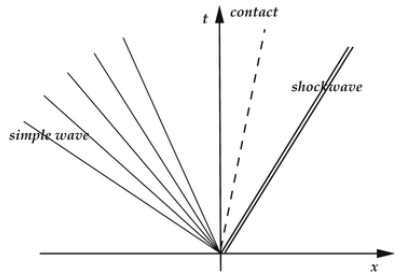
WORK IN PROGRESS

- Baseline: Mesh Management
  - Mesh Tree: **Unaware** of Data
  - Blocks: **Not Specific** to Modules
  - Modules: **Not Specific** to Algorithms
- Physics Modules "Tell" the Mesh Manager about What They Need
  - Easy to Emplace New Modules, Sub-modules, Algorithms
- Abstracted Interfaces
  - With Hardware (GPU, CPU, etc.)
  - With Communication Layer (MPI)
- Optimizations are Specific!
  - Can (and Should) be Accomplished on Sub-module Level
  - Can be Trimmed for Special Needs
- Computer Science Requirements: Type Calculus based on Templates

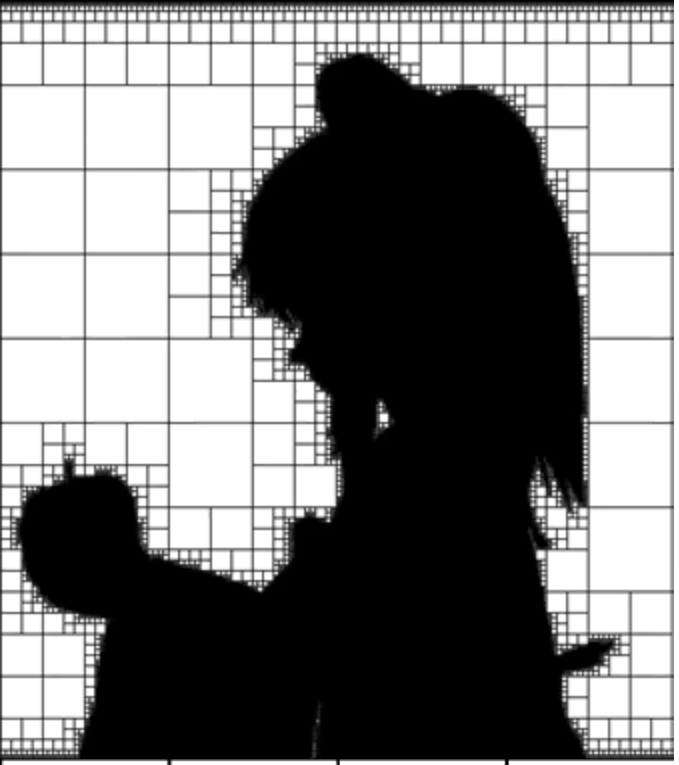


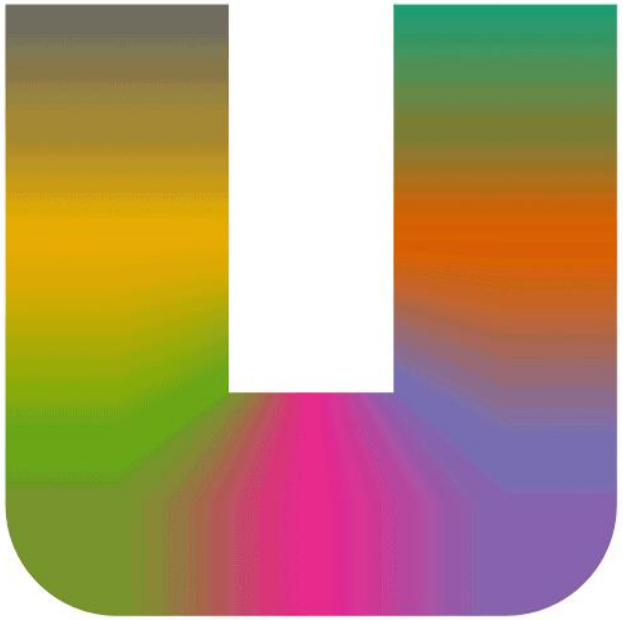


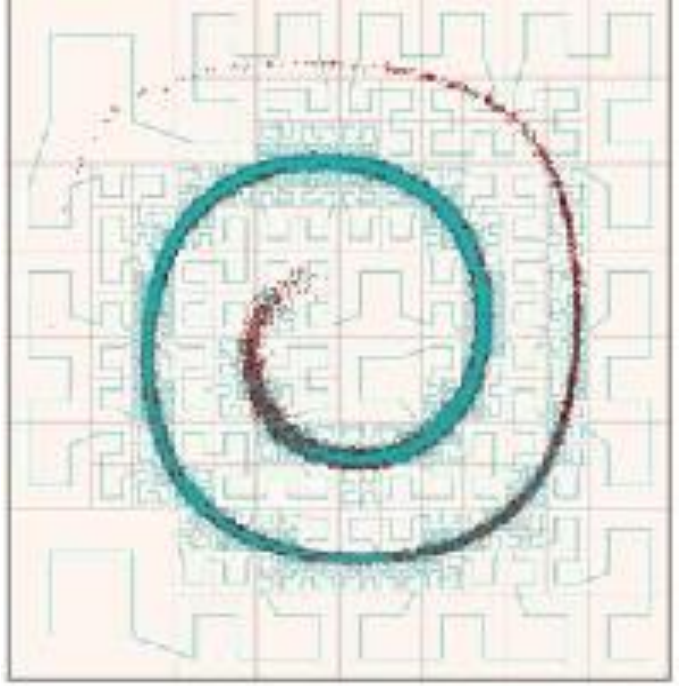


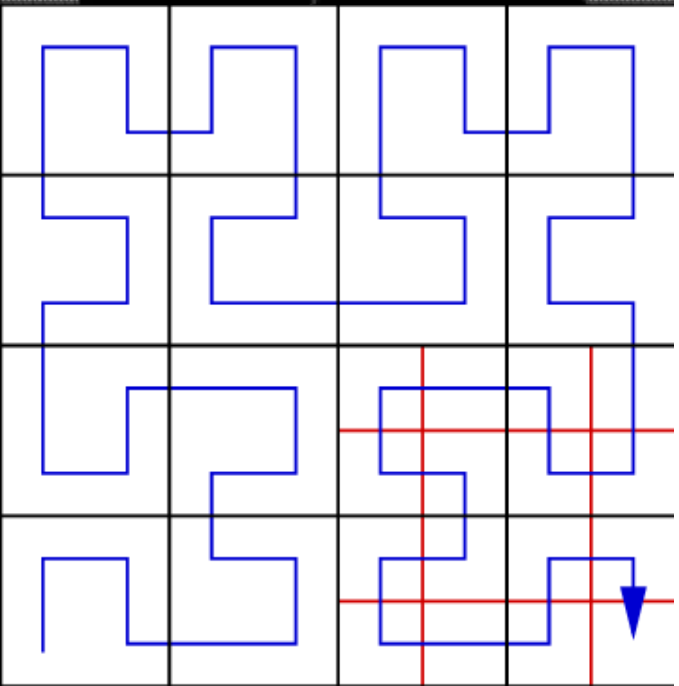


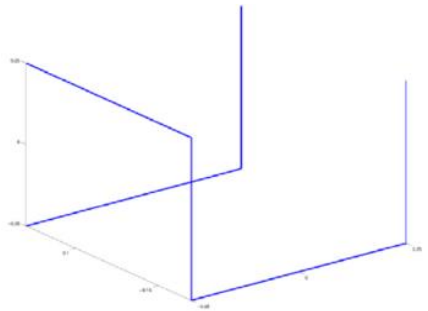
Why KRATOS is Generic and Fast

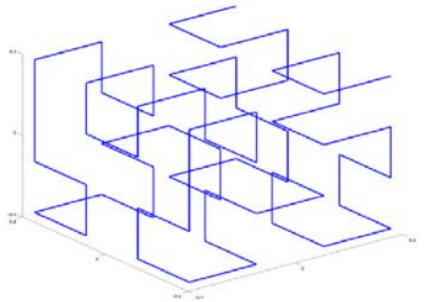


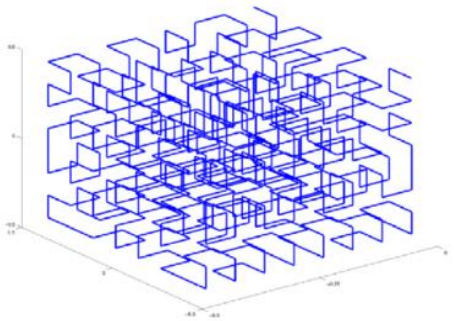


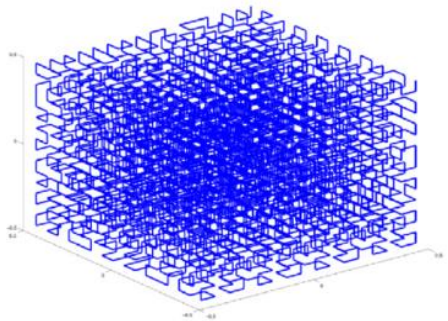


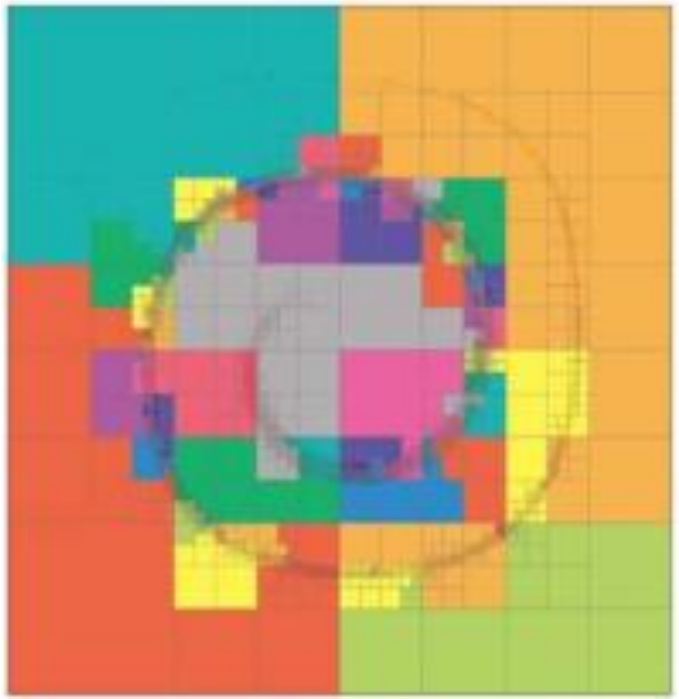




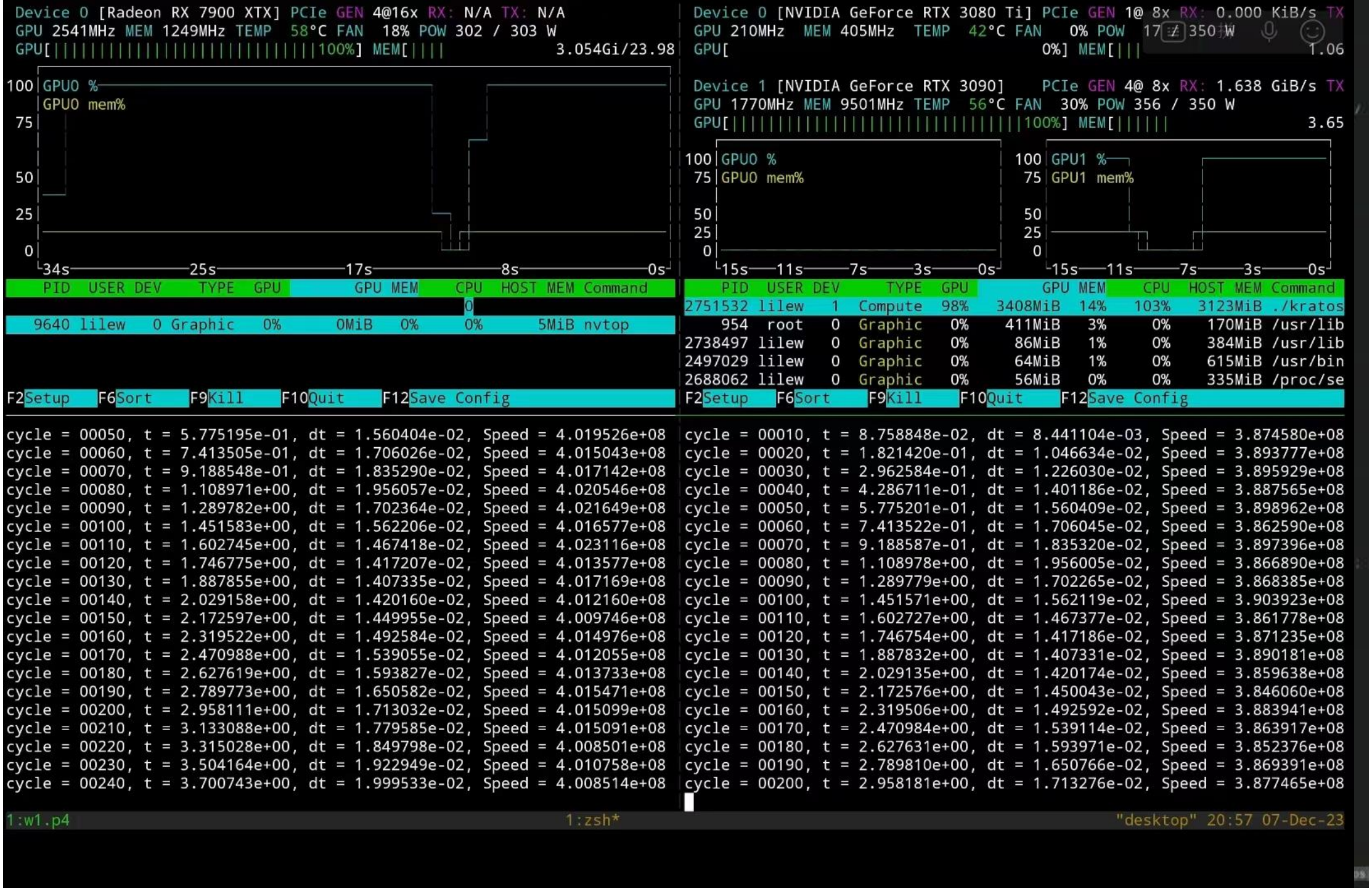




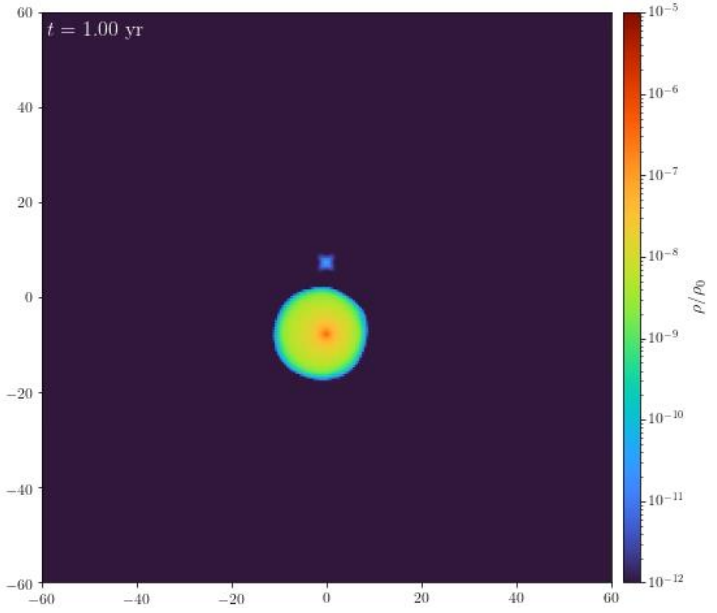


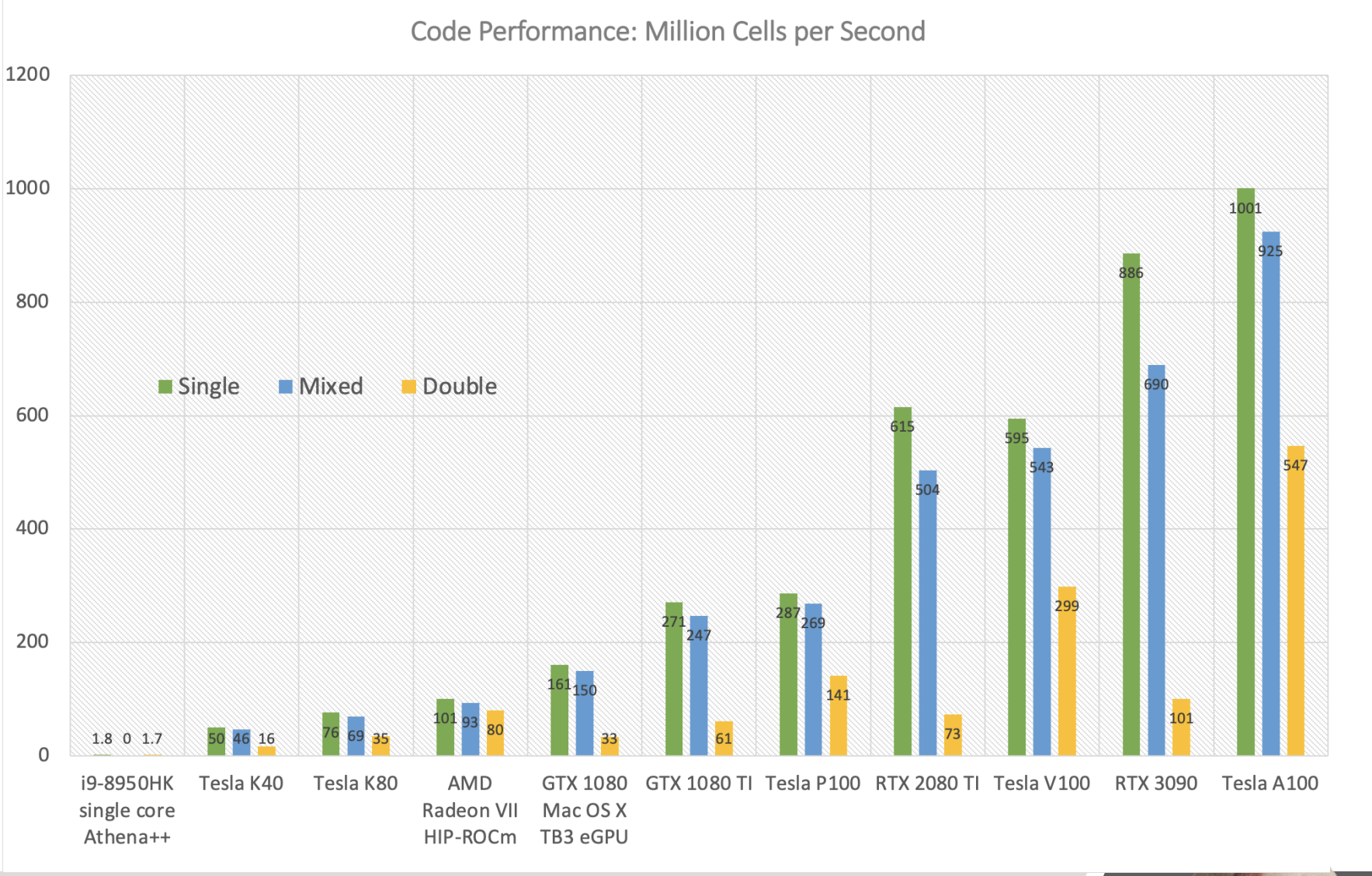


- NVIDIA
- AMD
- RISC-V
- ARM(!)
  - Phone
  - Tablet

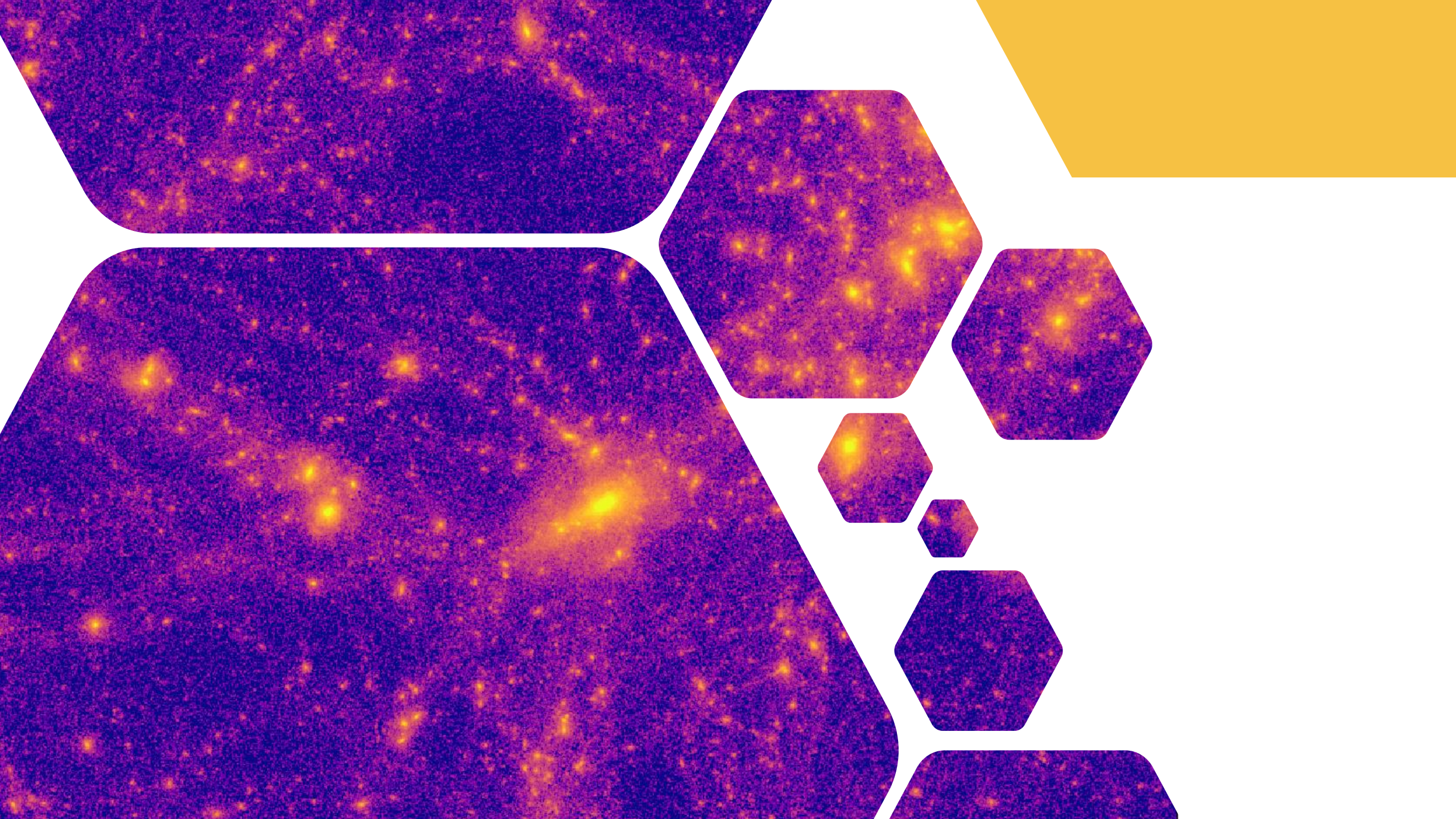


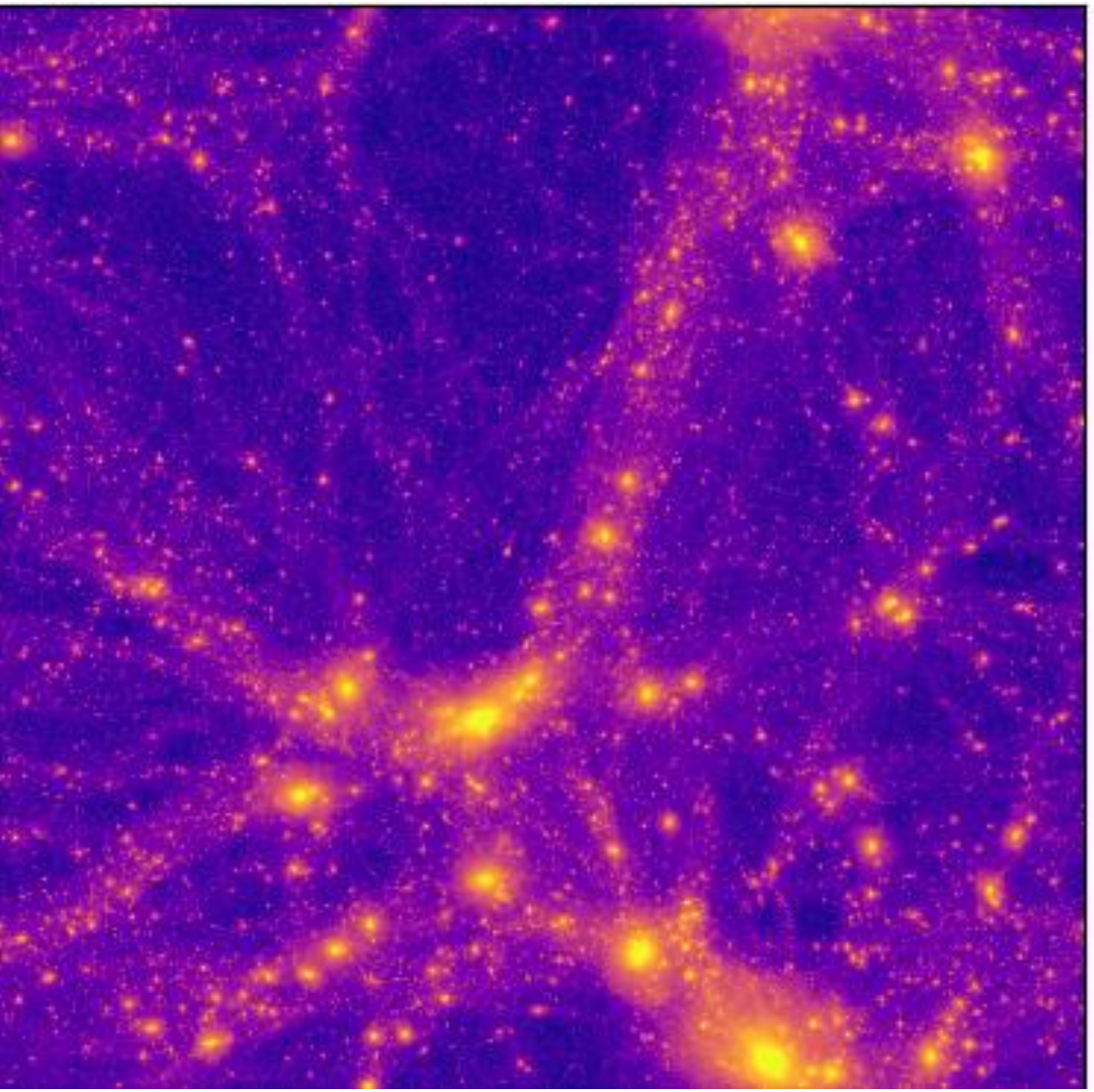
- Test case:
  Binary Outflows
- Single-card:
  10^9 cells/s on A100
- Strong Scaling:
  99.5% on 4xA100

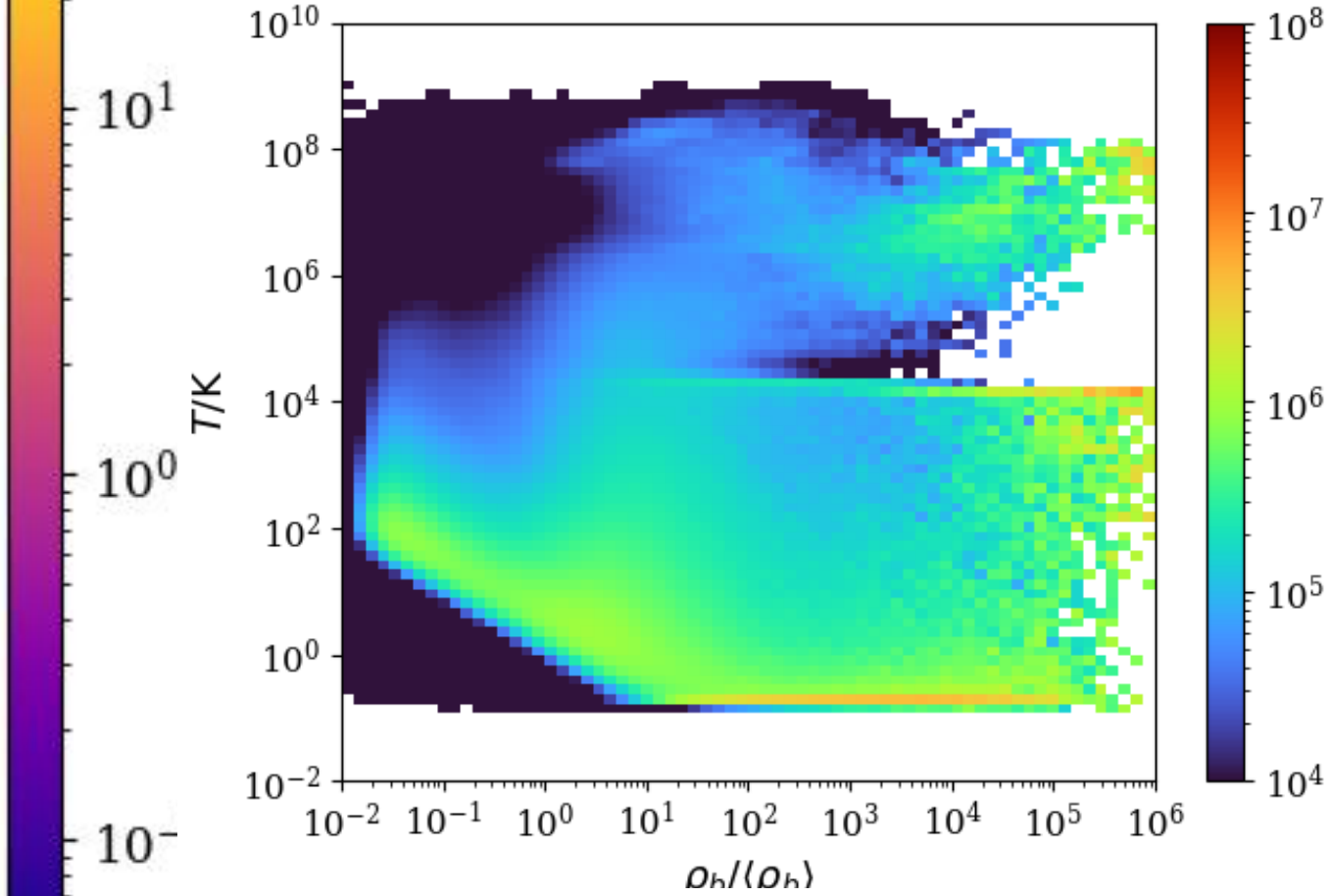


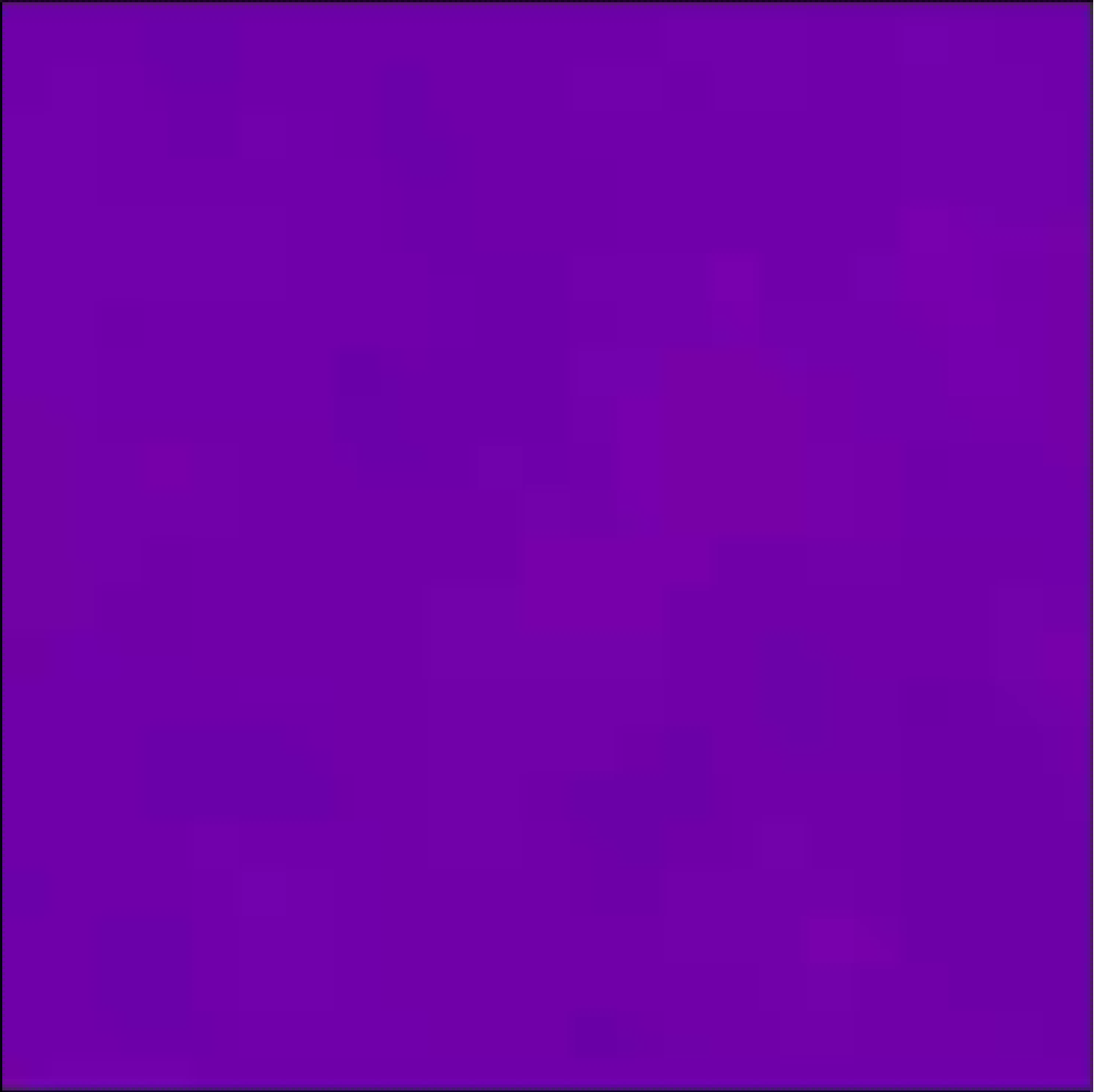


Usages and Tests: Hydrodynamics & MHD

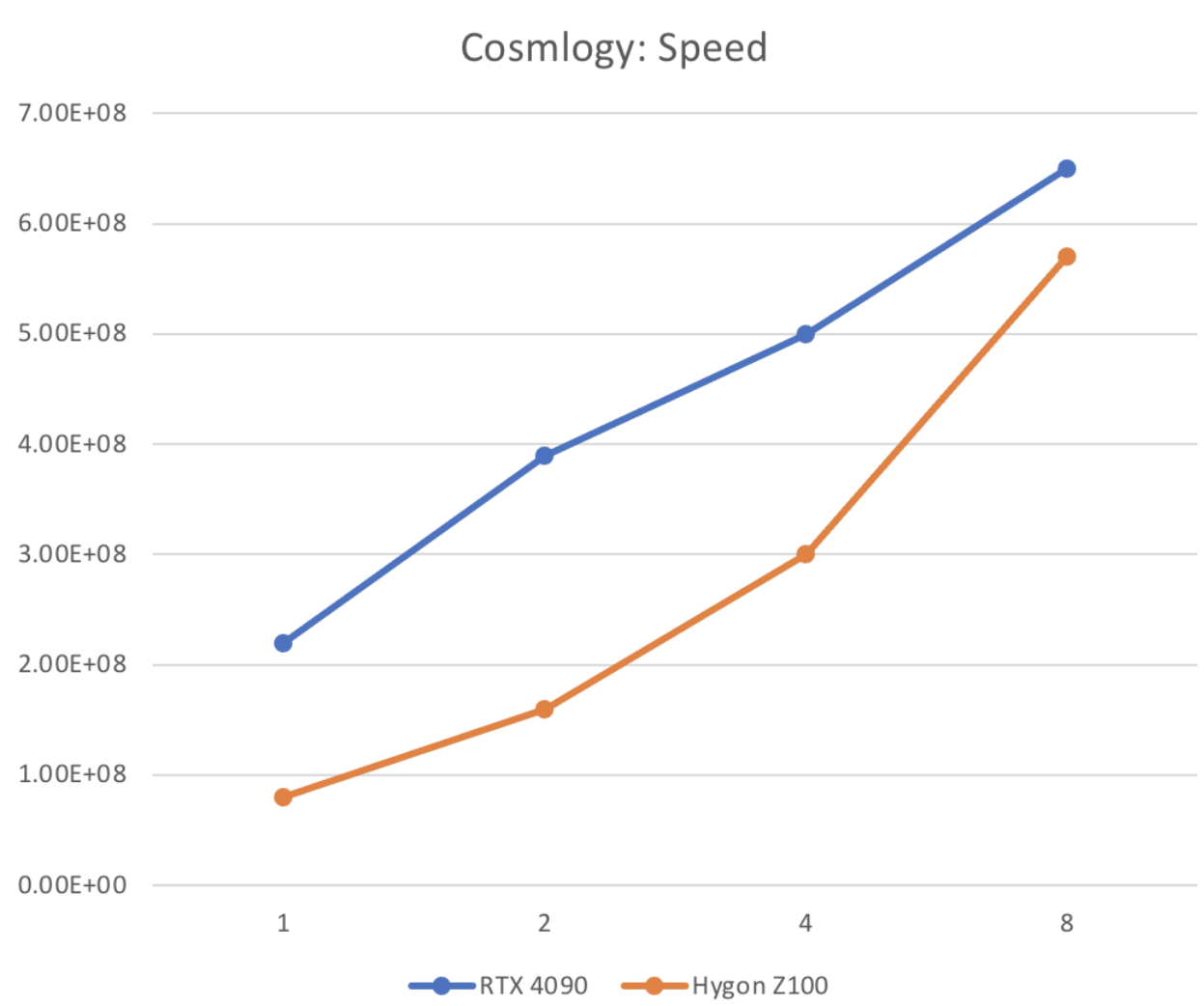


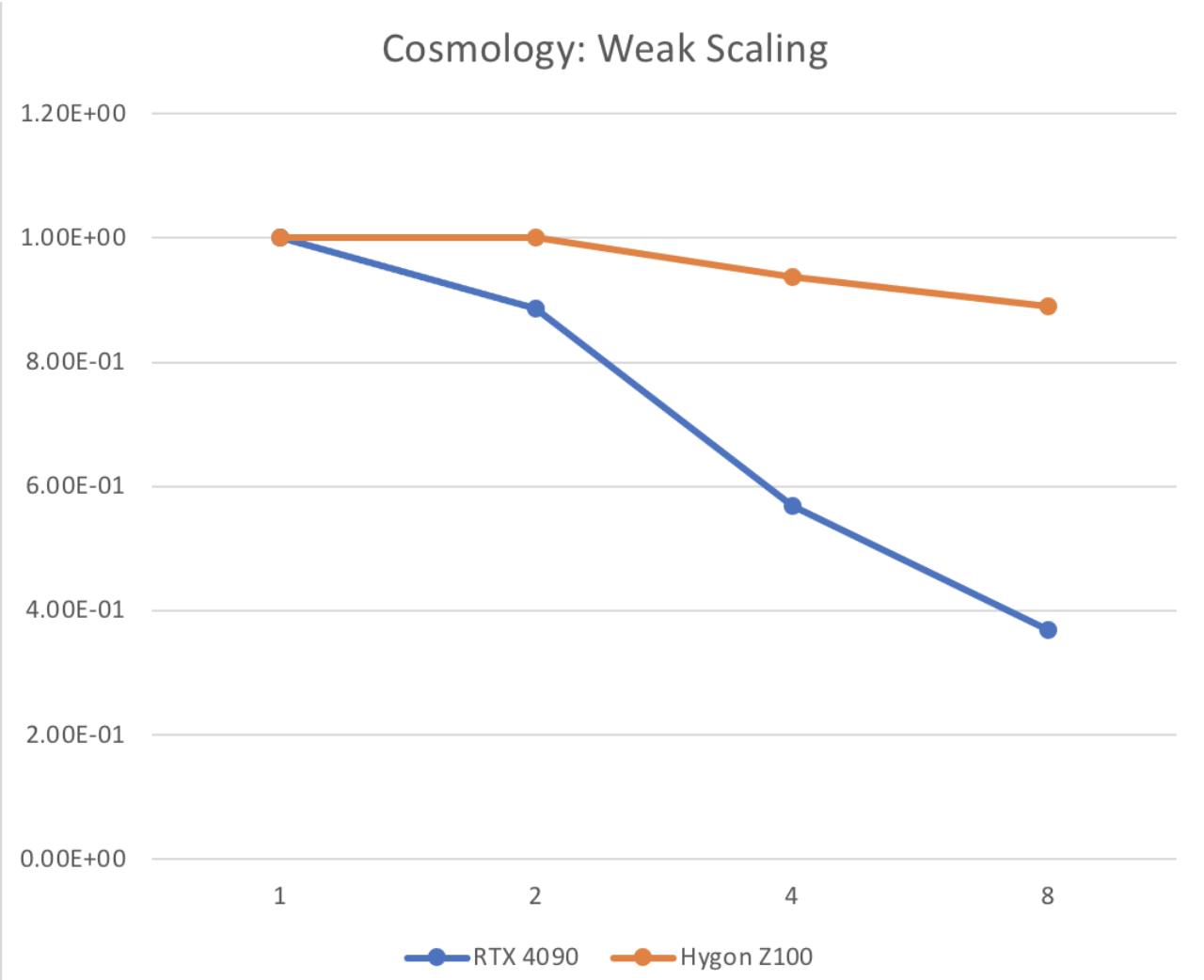


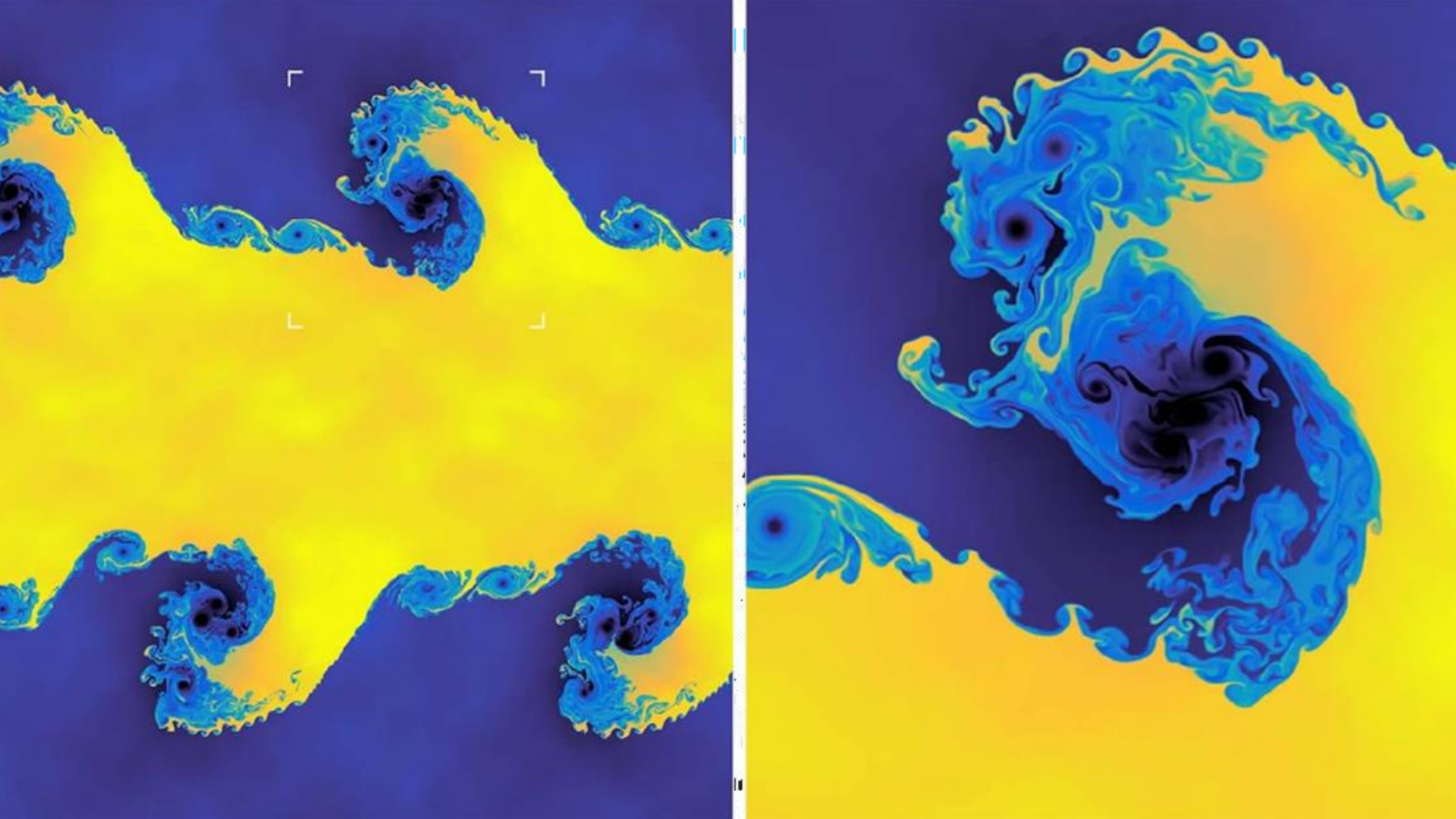

10^{10}
10^{8}
10^{6}
10^{4}
10^{2}
10^{0}
10^{-2}
T/K
10^{-2} 10^{-1} 10^{0} 10^{1} 10^{2} 10^{3} 10^{4} 10^{5} 10^{6}
$\rho_b/\langle\rho_b\rangle$
10^{8}
10^{7}
10^{6}
10^{5}
10^{4}

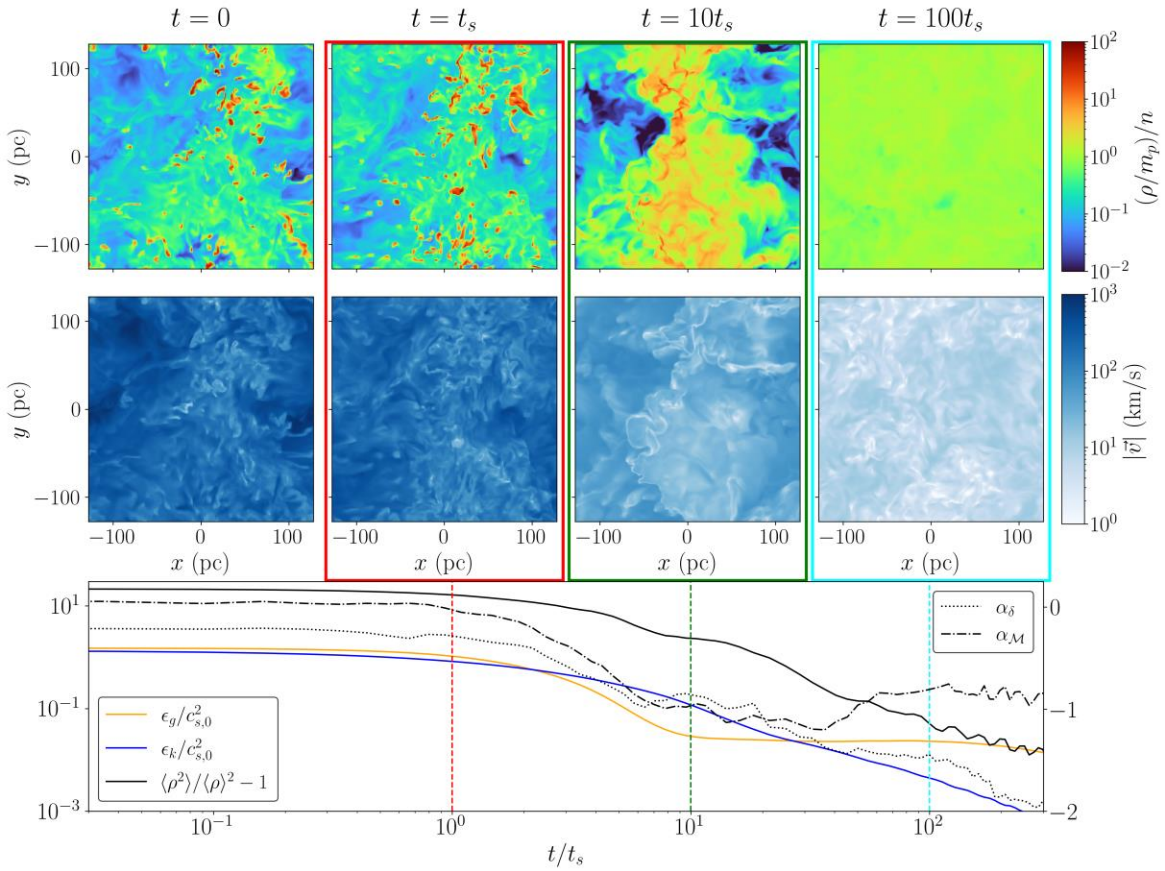


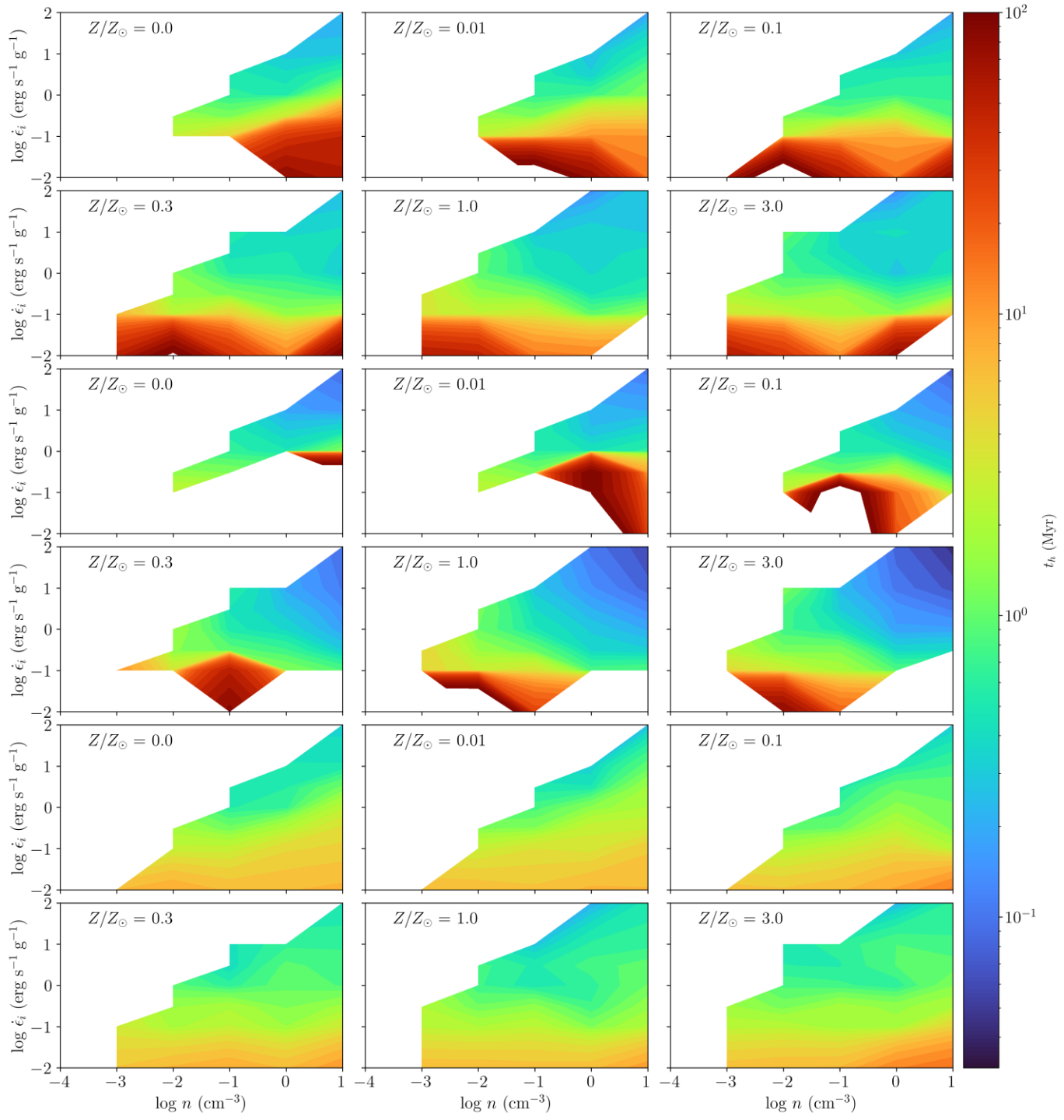
- MUSIC generated intial conditions
  https://www-n.oca.eu/ohahn/MUSIC/
- Particle module for DM (1 or 2 order LF)
- VL2 mixed-precision hydro with dual-energy formalism (Why VL2?)
- Multi-grid Poisson solver
- How fast is fast?


Cosmlogy: Speed
7.00E+08
6.00E+08
5.00E+08
4.00E+08
3.00E+08
2.00E+08
1.00E+08
0.00E+00
1
2
4
8
RTX 4090
Hygon Z100

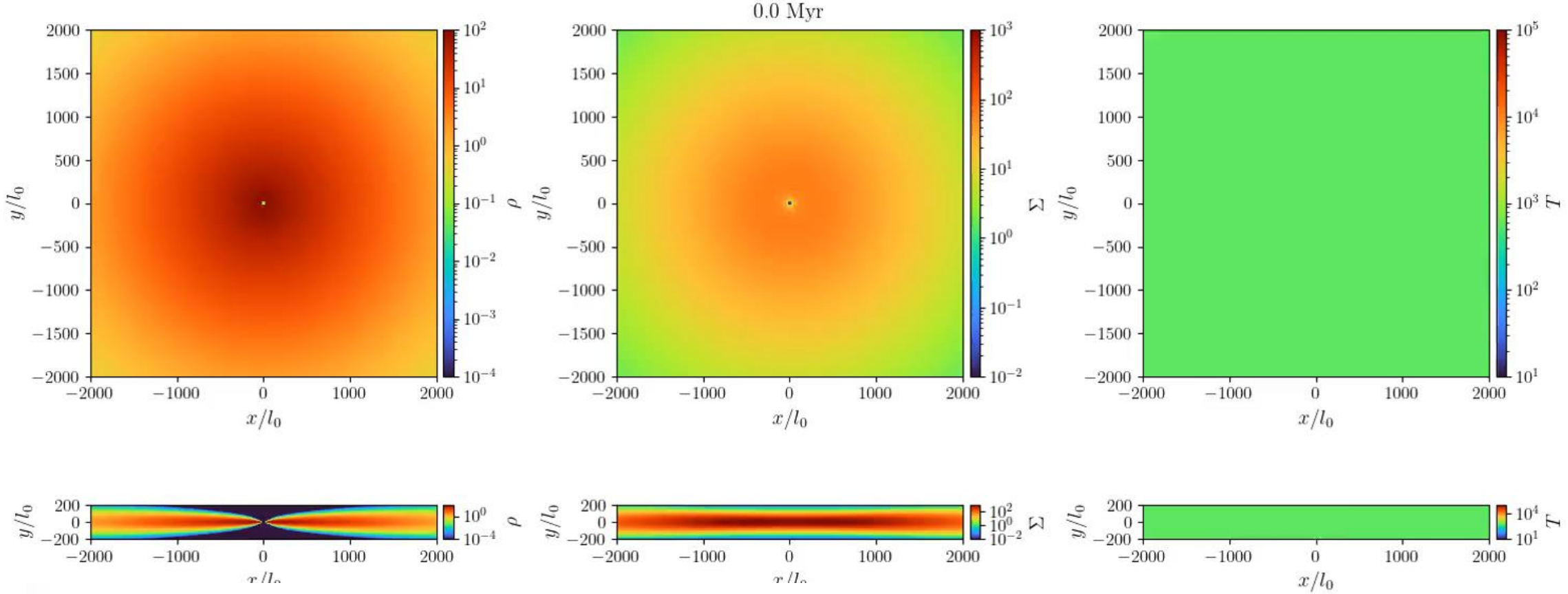

Cosmology: Weak Scaling
1.20E+00
1.00E+00
8.00E-01
6.00E-01
4.00E-01
2.00E-01
0.00E+00
1
2
4
8
RTX 4090
Hygon Z100

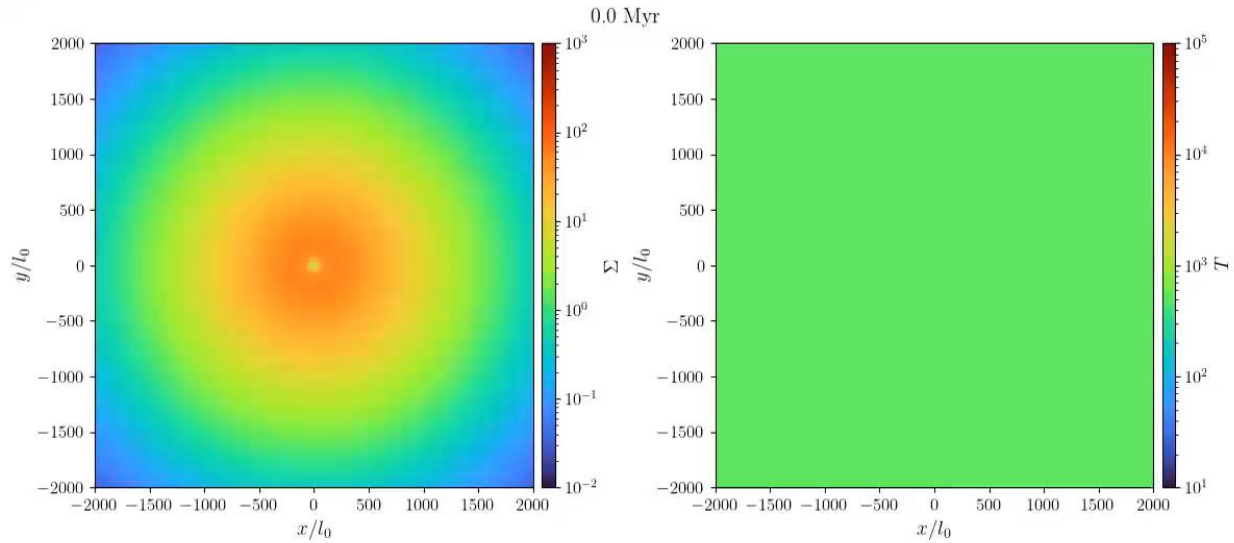






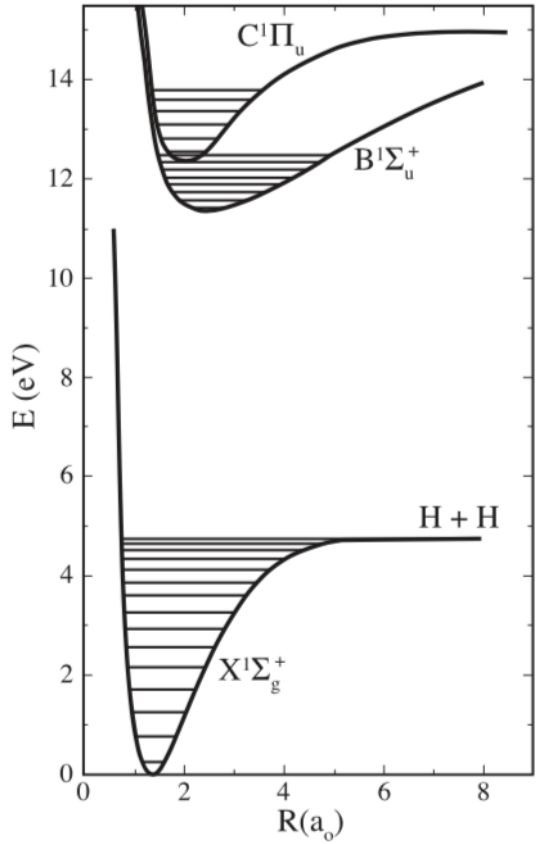
- 252 sims
- 1.5 days
- On this guy
- arXiv:2406.18920

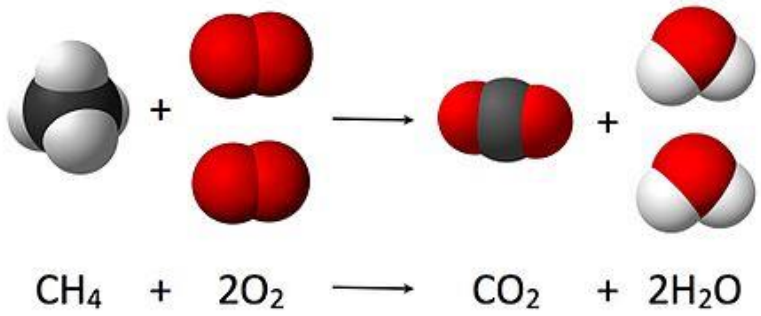


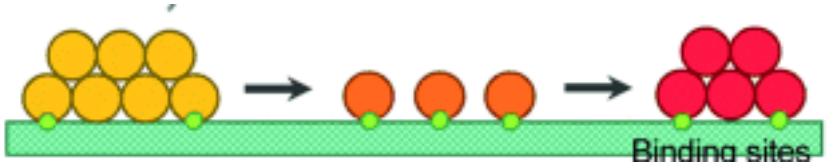


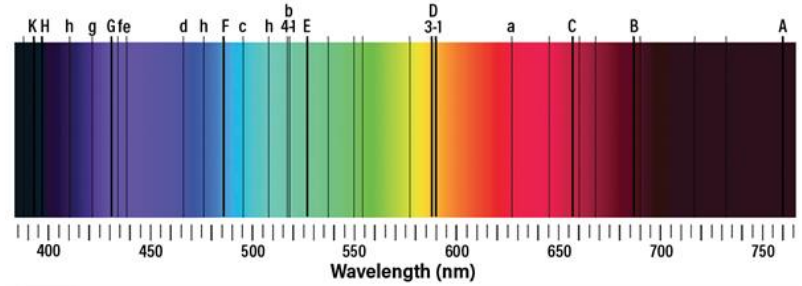
Lv, Wang, Ho et al., in prep.

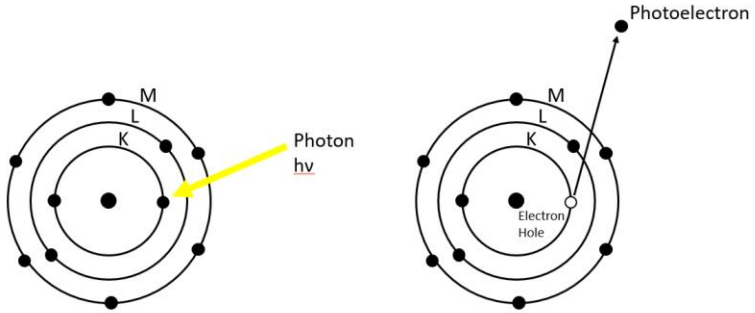
Microphysics











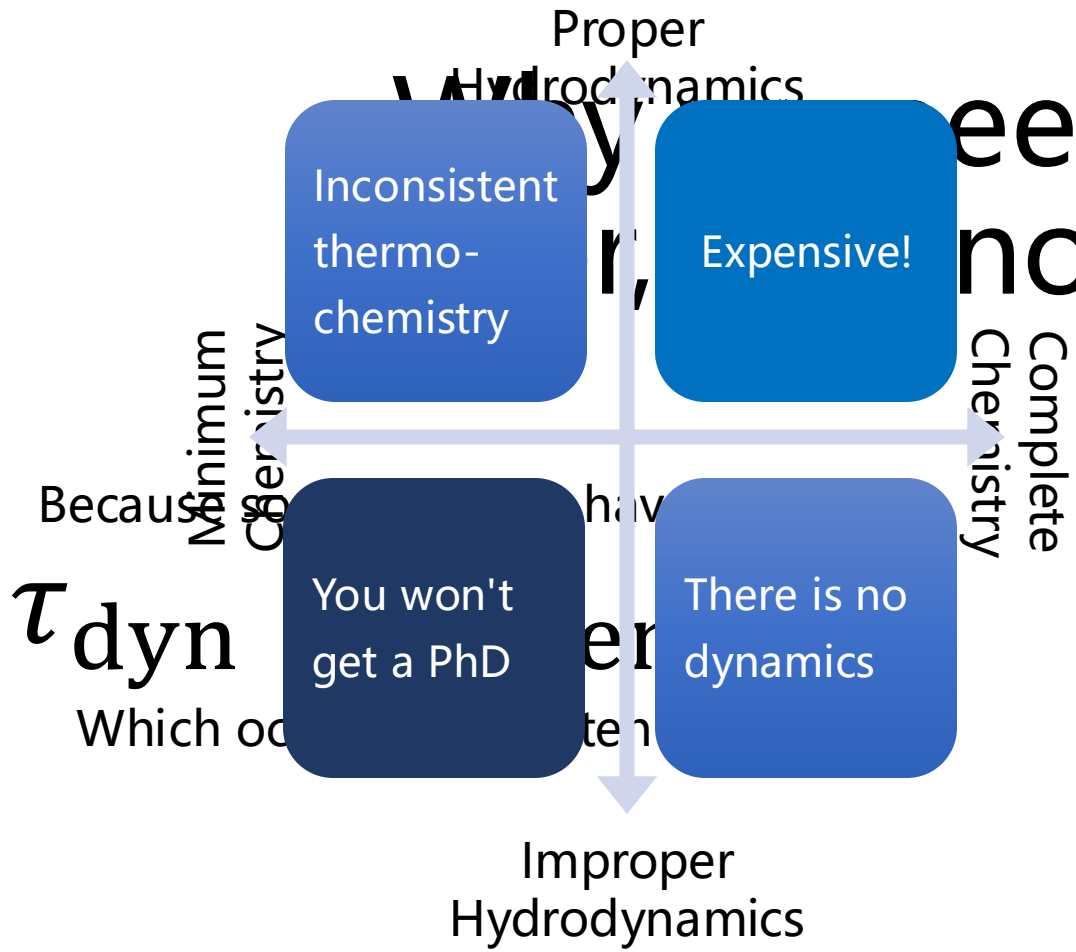
Chemistry (“Microphysics”) is Necessary

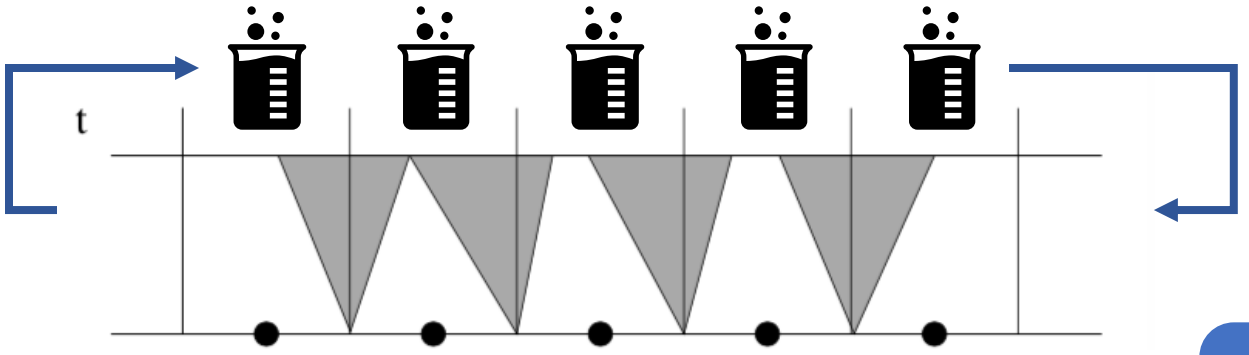
Why need it
r, not equilibrium?

Because s ha

τ_{dyn} er

Which oc te

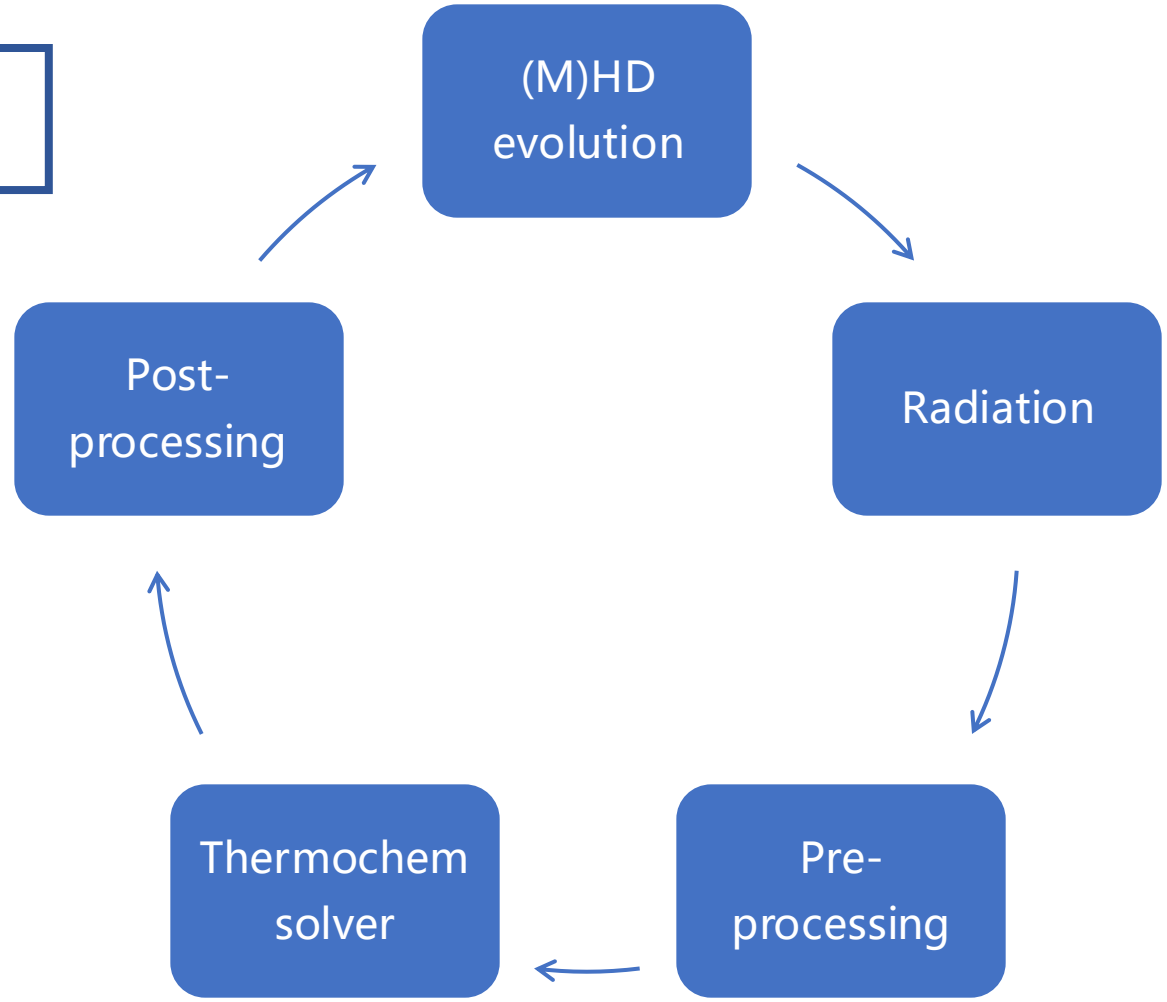




Post-processing

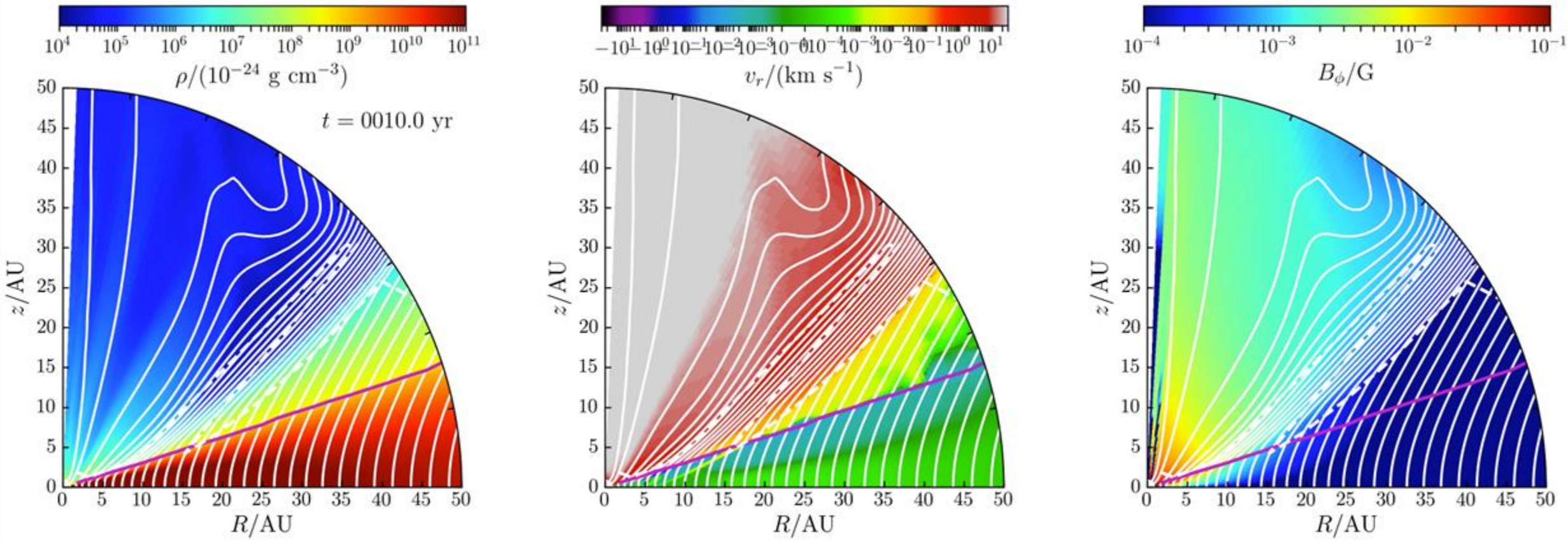
Eliminate inconsistency before data rejoin (M)HD procedures

- Fix violations of conservations in elements and charge
- Detect non-converged chemistry, retry/reset
- Update energy density and chemical abundances
- Update non-ideal MHD diffusivities



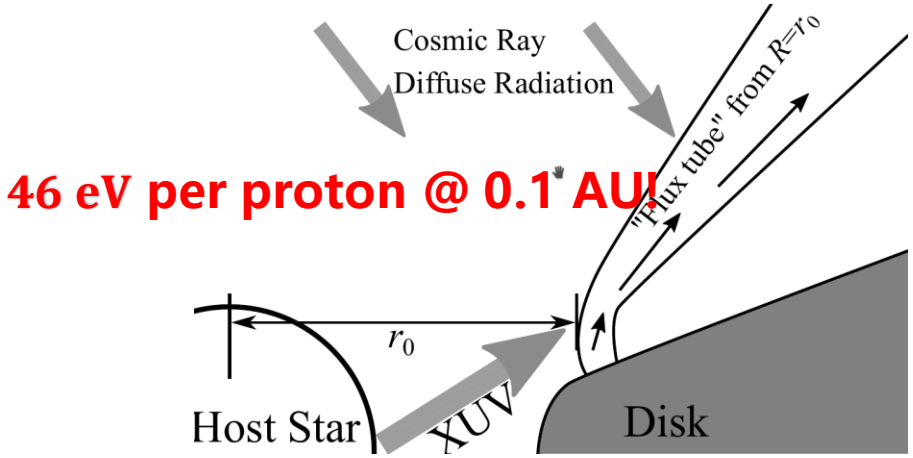
Realtime GPU Microphysics

L. Wang
PhD Thesis
(2018)



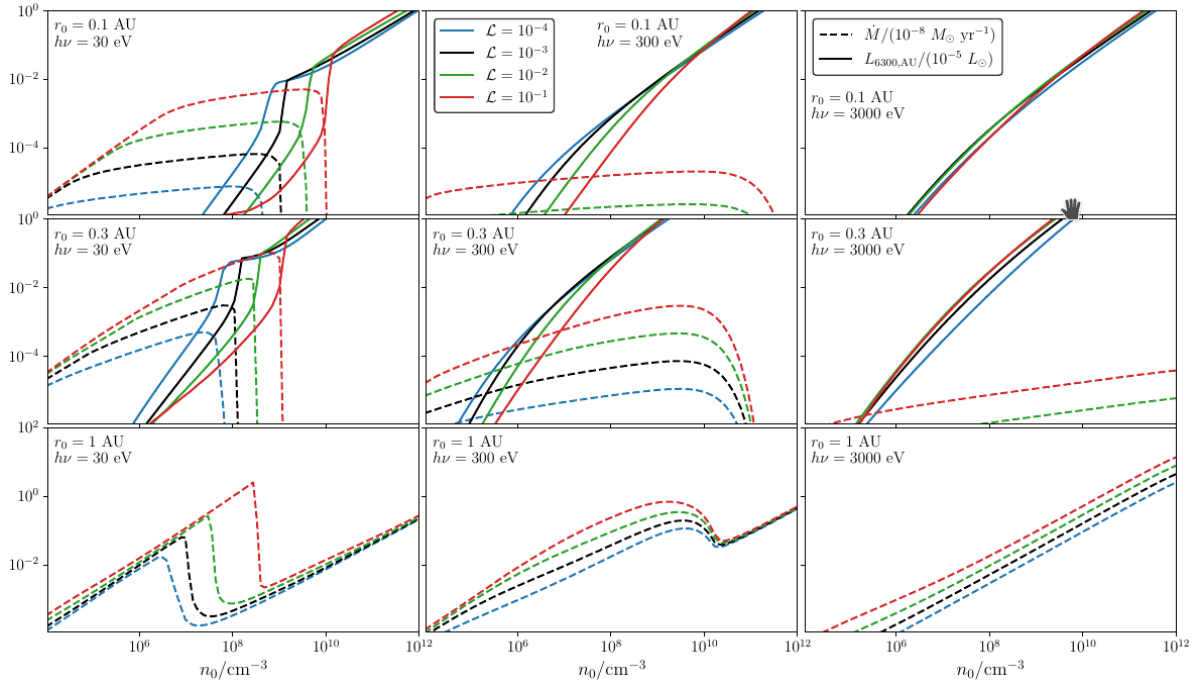
Wind-driven Accretion: Consistent Microphysics

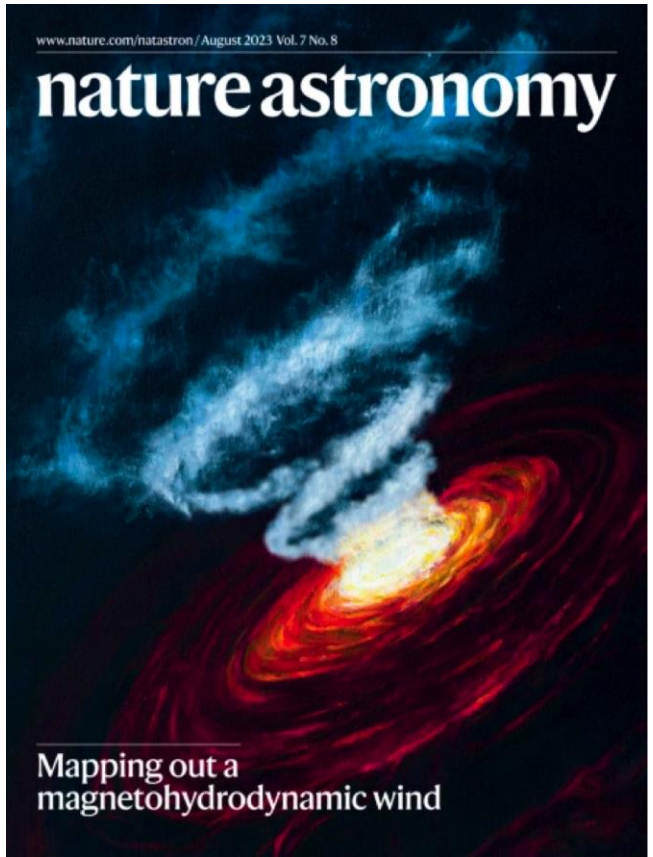
Wang, Bai & Goodman
2019 ApJ, 874, 90

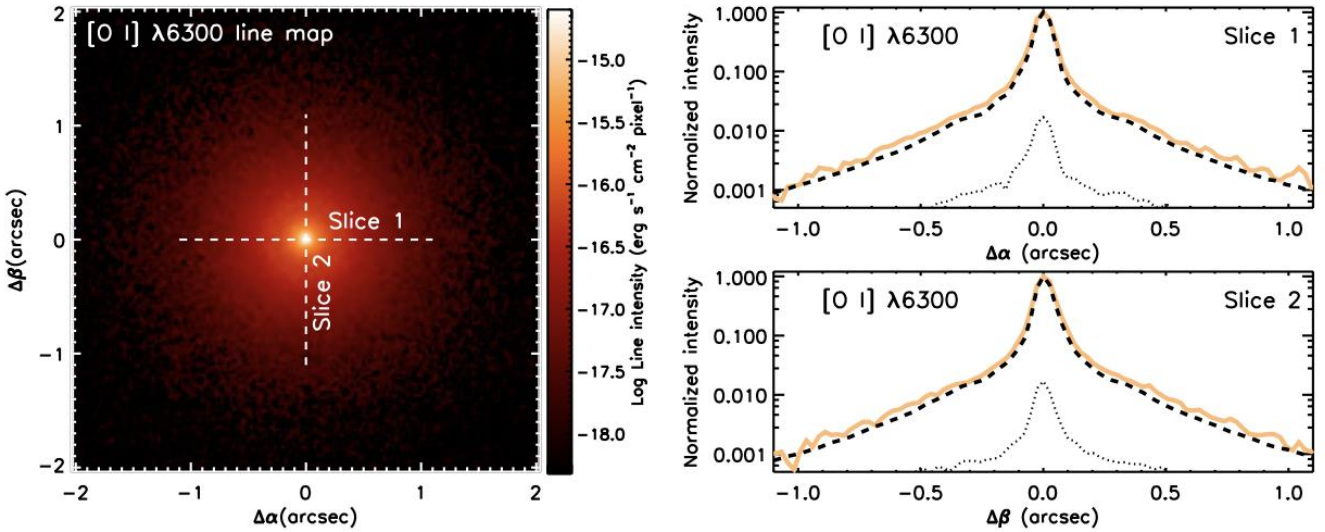


46 eV per proton @ 0.1 AU

Lin, Wang, Fang, Ahmad & Goodman (arXiv:2401.15419)

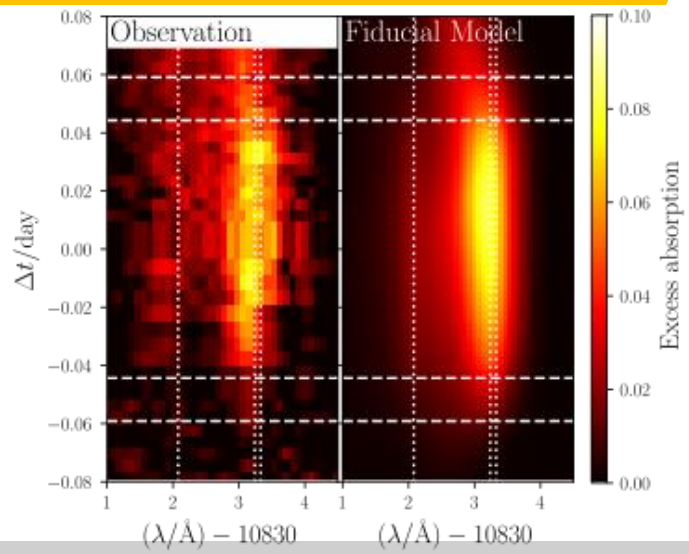


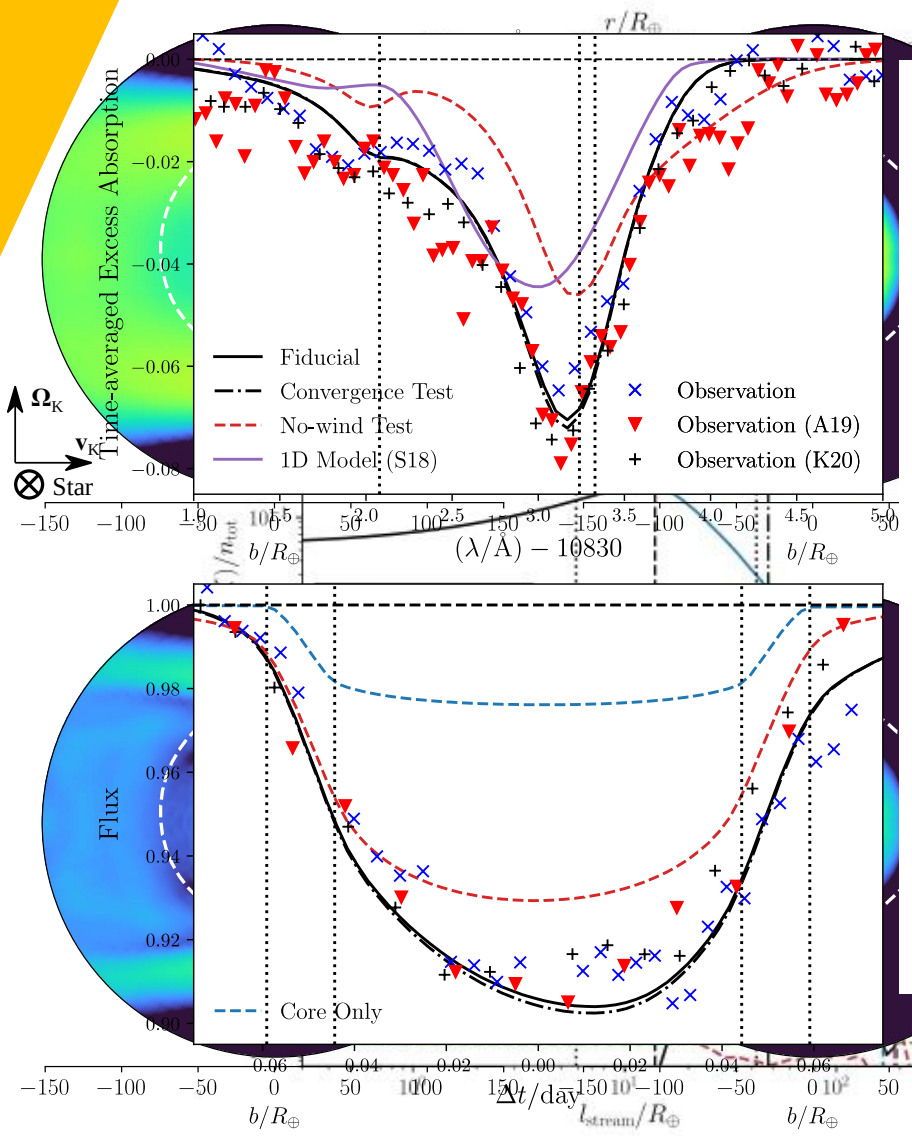




Fang, Wang, Herczeg et al., Nature Astronomy (2023 Aug Cover Paper)

- $\tau_{\rm dyn}$~1 hr, τ(He 2^3S)~1 hr
- He I 10830 A absorption: ionized outflows
- Full hydro simulations for observed spectra





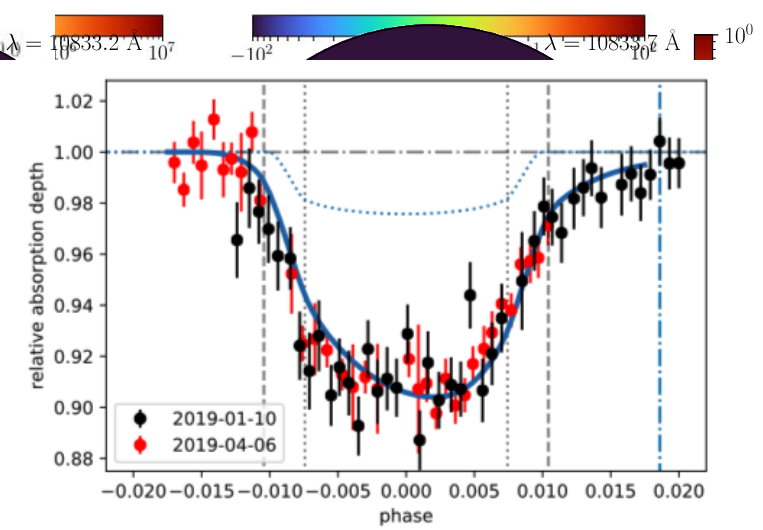
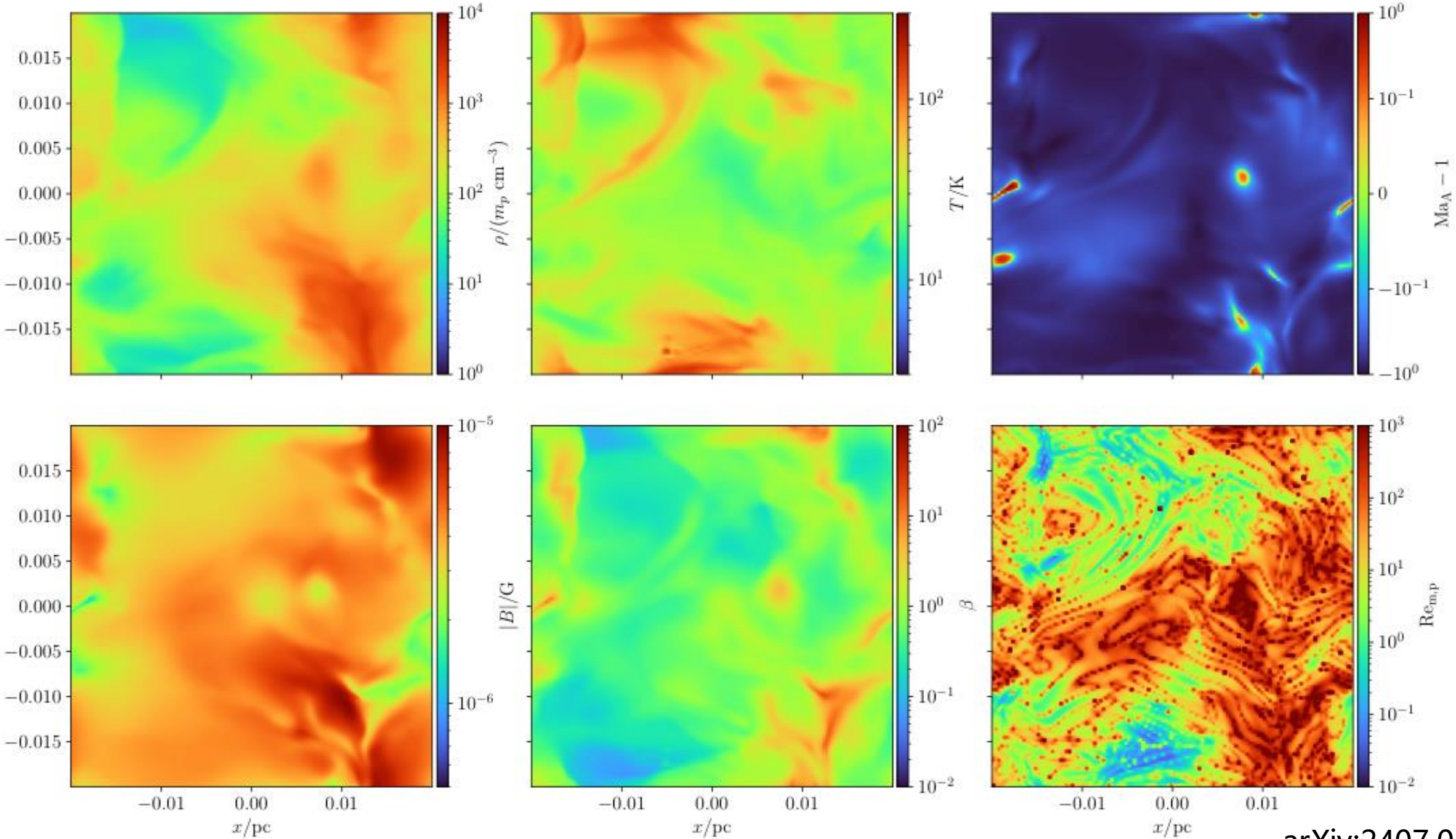


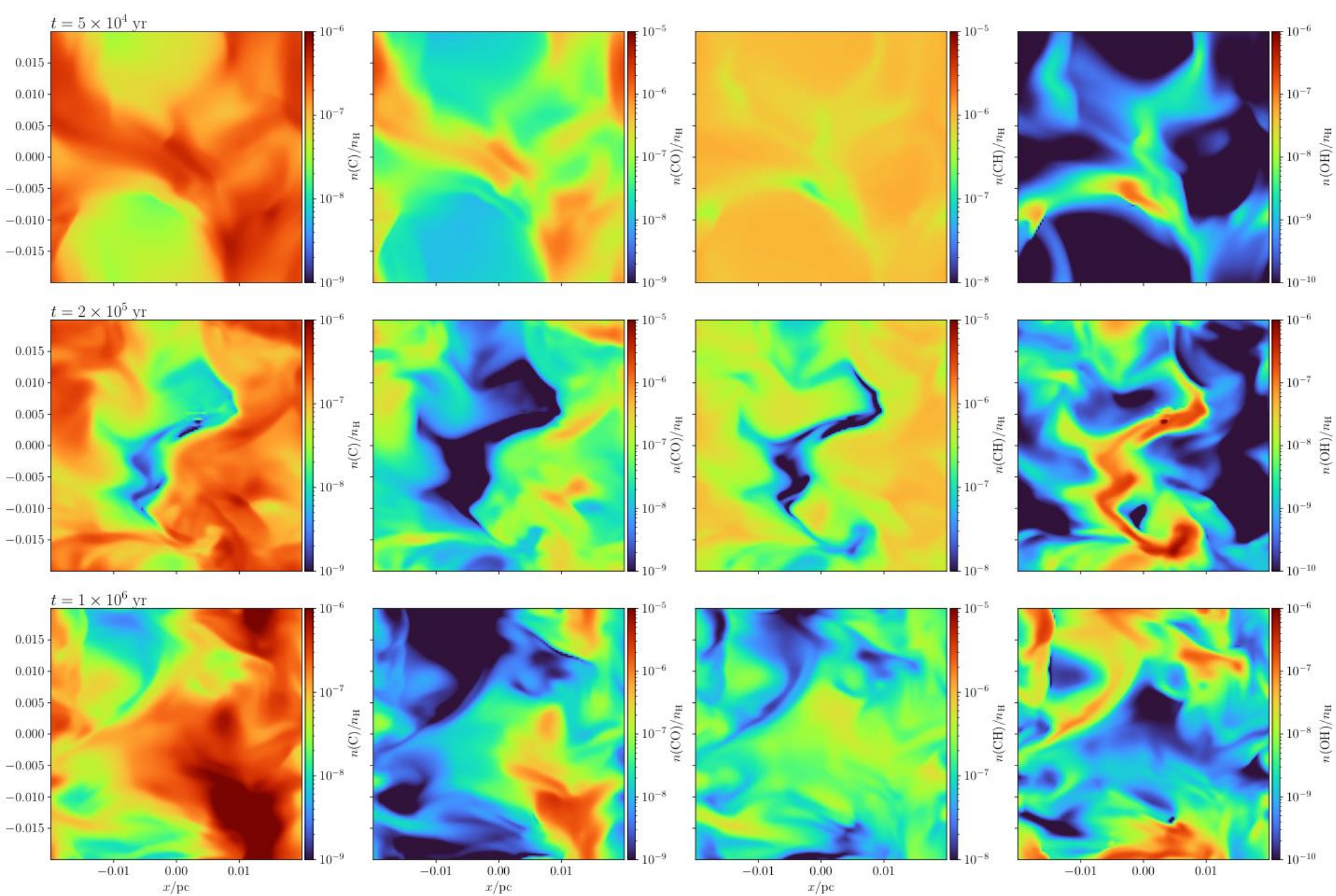
Figure 4. Top panel: Light curves showing excess helium absorption integrated over wavelengths between 1082.983 nm and 1083.058 nm. The gray(red) area shows phases used for the construction of the out-of-transit spectrum for the January(April) night. Bottom panel: same as top panel, but both light curves are added to the white-light curve from Spake et al. (2018) (dotted blue curve), and then normalized so that the transit depths match. Solid blue curve is the light curve based on the fiducial hydrodynamic simulation from Wang & Dai (2020). Dash-dotted vertical line indcates the end of the post-transit absorption.

Direct Evidence of Dispersal: A He I 10830

Wang & Dai 2021 ApJ, 914, 99

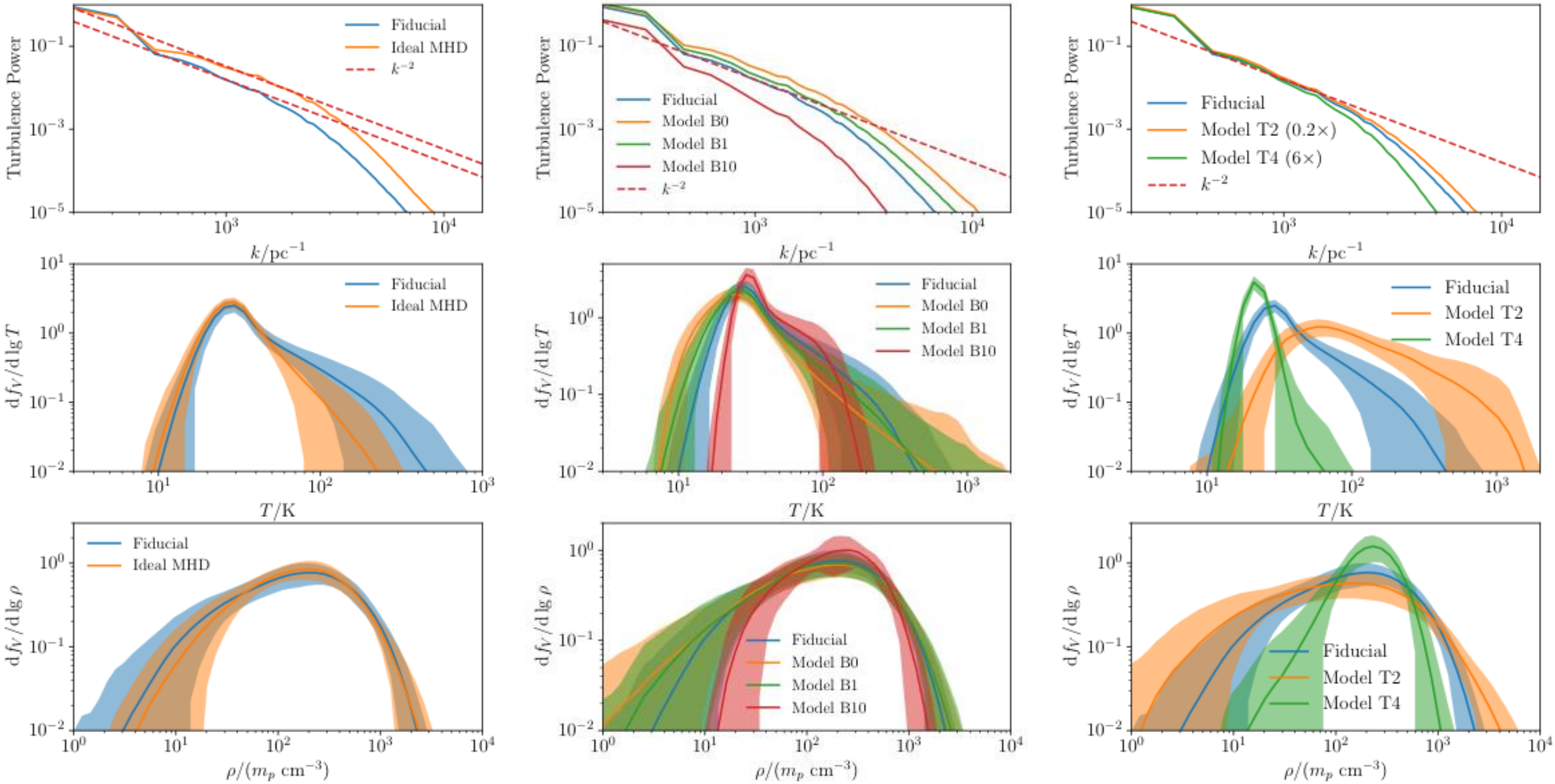


arXiv:2407.02306



arXiv:2407.02306

(ApJ accepted)



arXiv:2407.02306 (ApJ accepted)

- Any CUDA tutorials--Simply search “CUDA tutorial” with Google/Bing. Specifically, please search these topics:
  - Stream (graphs), Shared Memory, Multi-thread Algorithms
  - cuda-gdb, cuda-memcheck
  - Template metaprogramming of C++
- Documents of CUDA and HIP/ROCm
- https://github.com/ROCm-Developer-Tools/HIP-CPU
- Write your own GPU project and play with it!

Search-Engine-Oriented Programming

Questions?