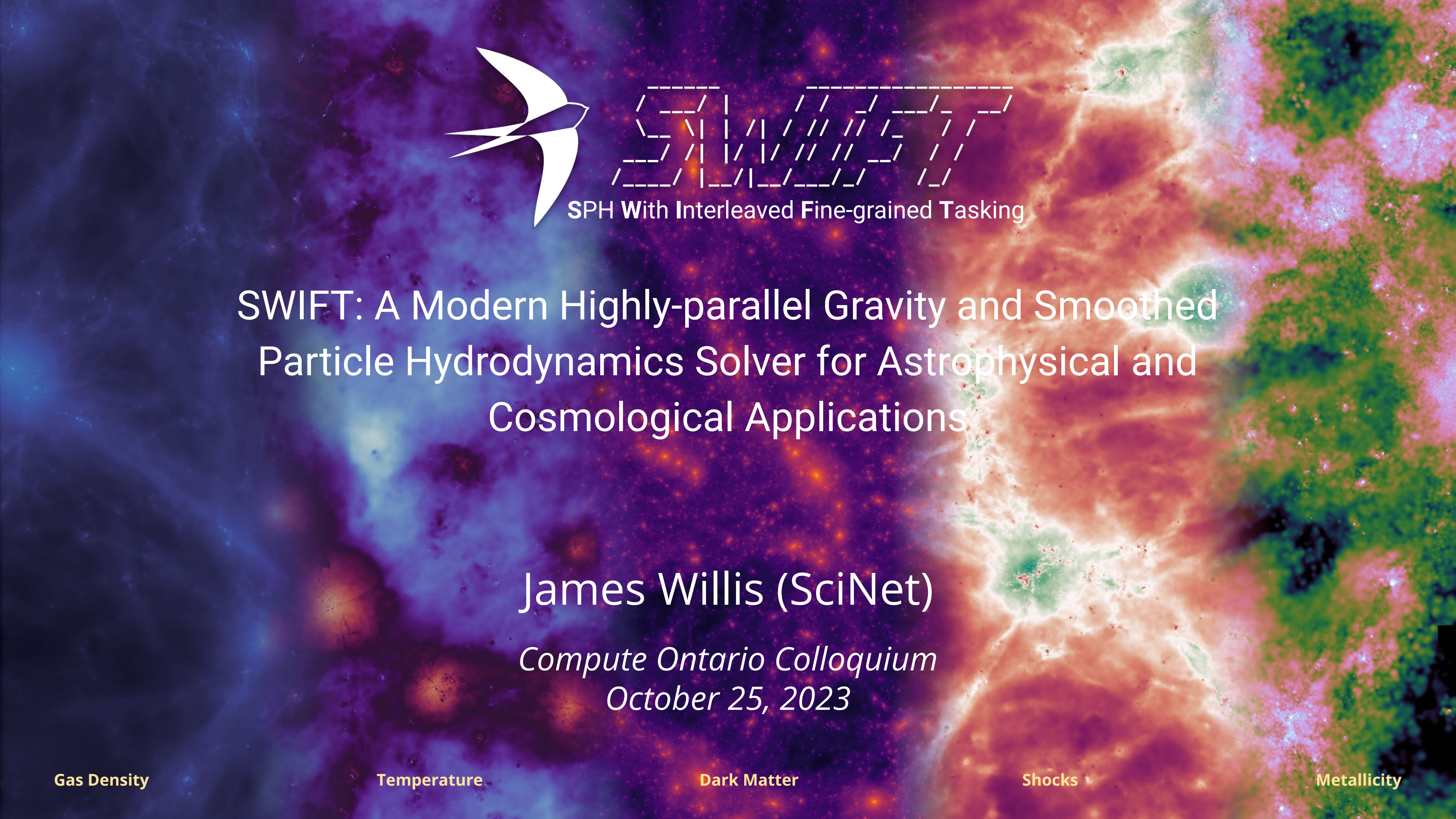
SPH With Interleaved Fine-grained Tasking

SWIFT: A Modern Highly-parallel Gravity and Smoothed Particle Hydrodynamics Solver for Astrophysical and Cosmological Applications

James Willis (SciNet)

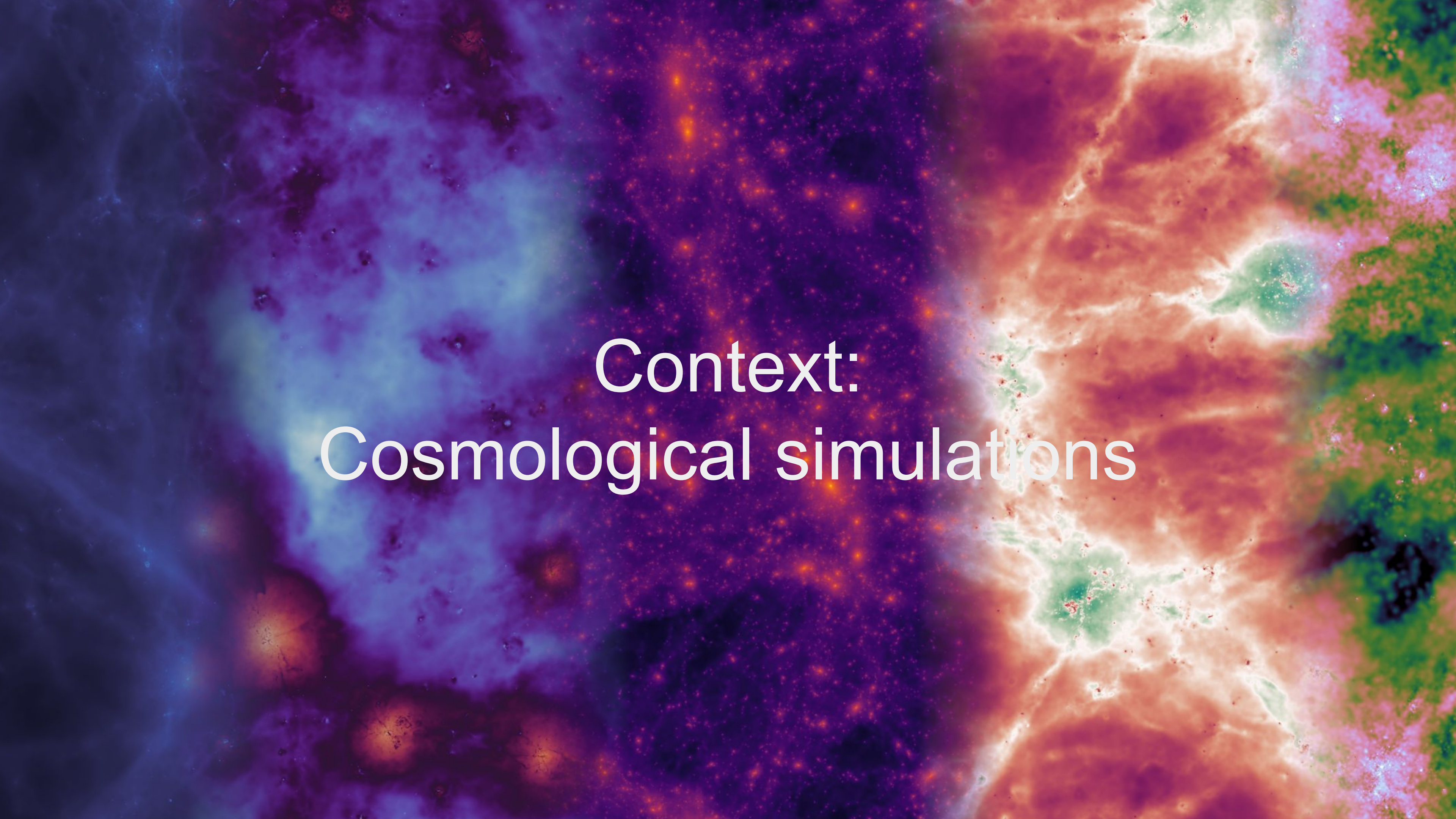
Compute Ontario Colloquium
October 25, 2023

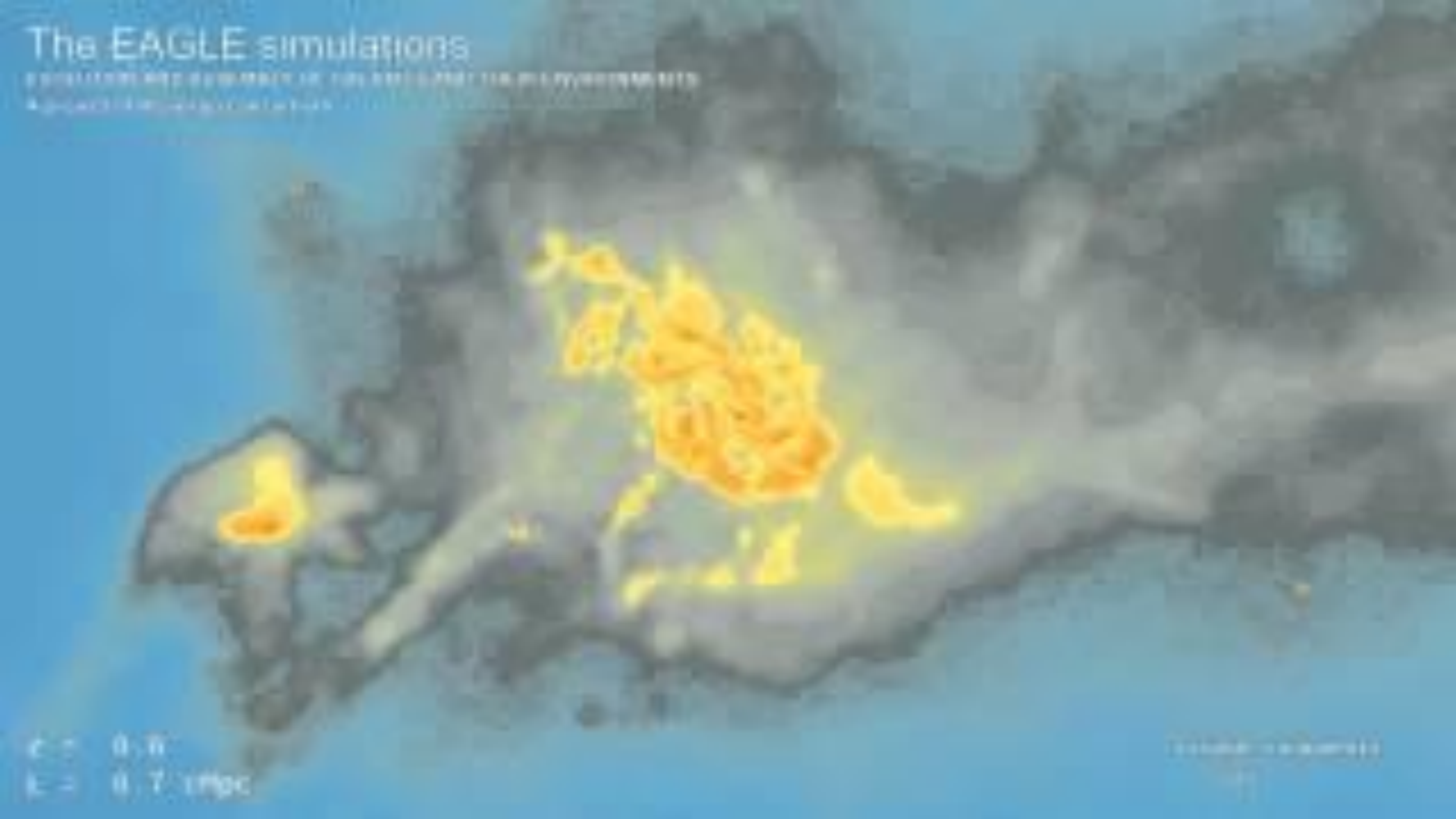


Overview

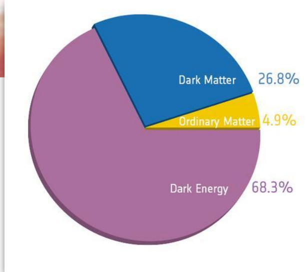
- Motivation
- Problem to solve
- Solution using SPH
- SWIFT implementation using Task-Based Parallelism
- Verlet lists for particle interactions
- Optimising particle timesteps
- Load balancing

Context:
Cosmological simulations




The EAGLE simulations

Simulating a Universe

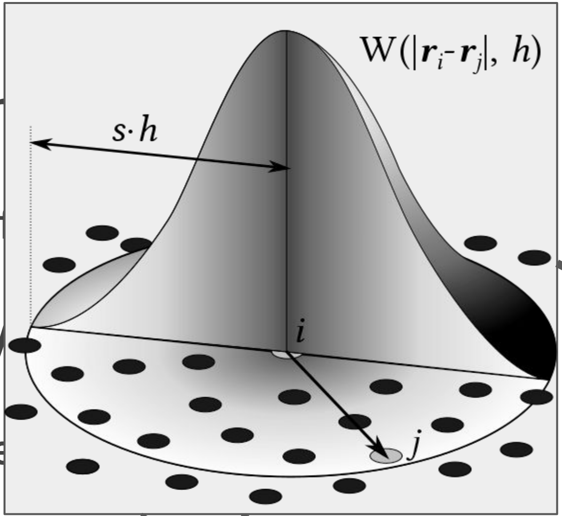


Initial co

- Known

Boundary

- Assume



Forces (physics):

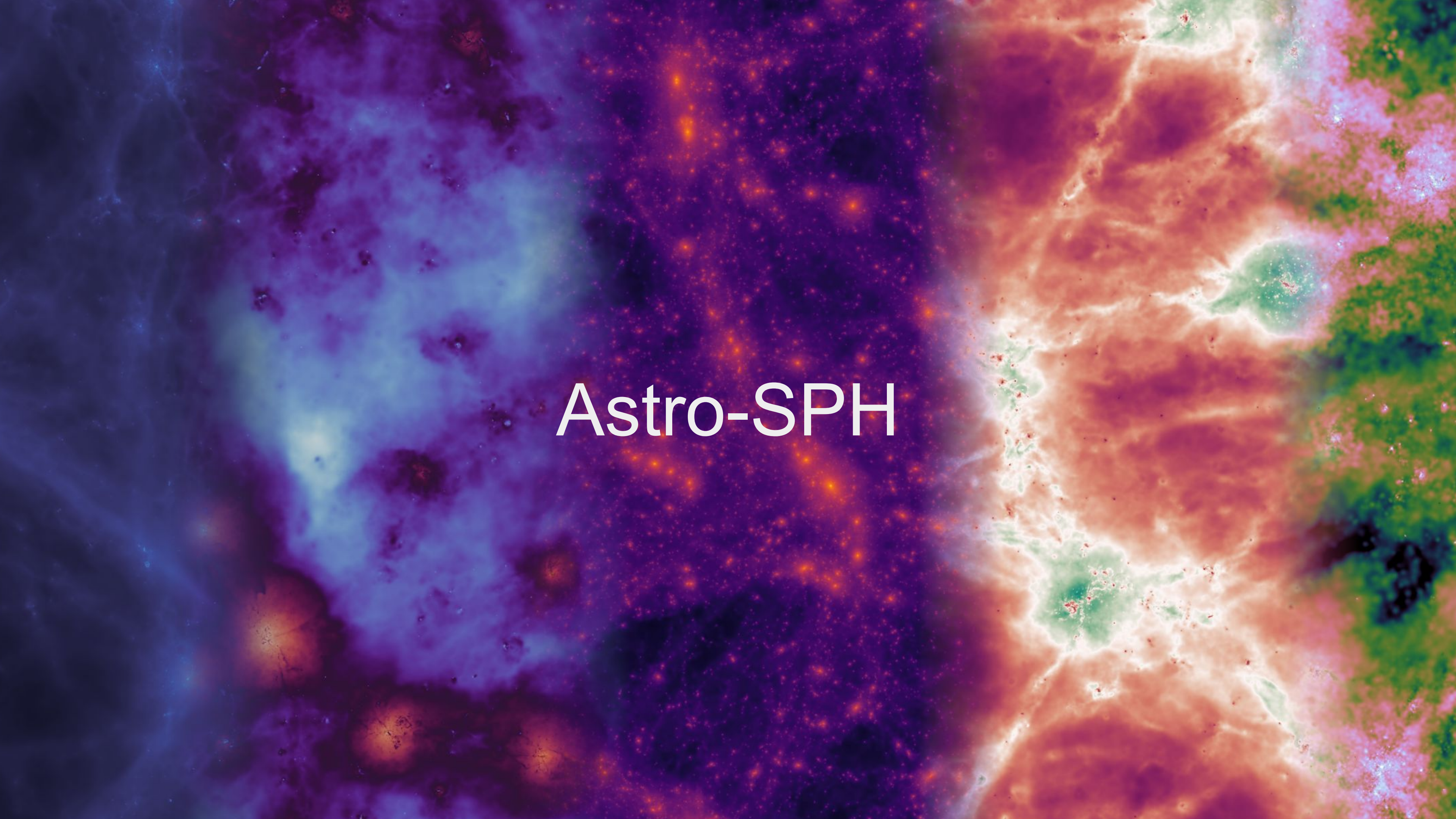
- Gravity, hydrodynamics, magnetic fields (?), radiative transfer (?), cosmic rays (?), ...

Constituents:

“easy”

- *Dark energy*: Absorbed in the choice of coordinates.
- *Dark matter*: Treated as bodies acting gravitationally.
- *Normal matter*: Treated as an ideal (compressible) gas.
- *Others (stars, black holes, neutrinos, dust, planets, ...)*: Sub-grid models.

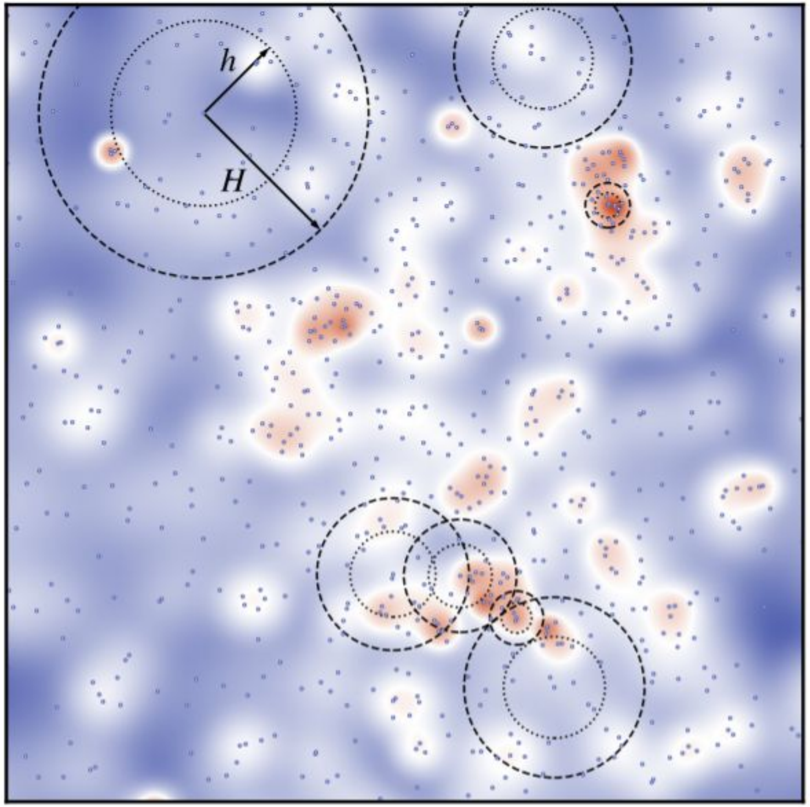
“very hard”


Astro-SPH

Astro-SPH: The basics

- Particles have a fixed mass.
- Density is defined by the (weighted) number of particles in a close neighbourhood.
- Particles loop over their neighbours to compute quantities via a weighting function $W(r, h)$.

Typical case: 50-100 neighbours

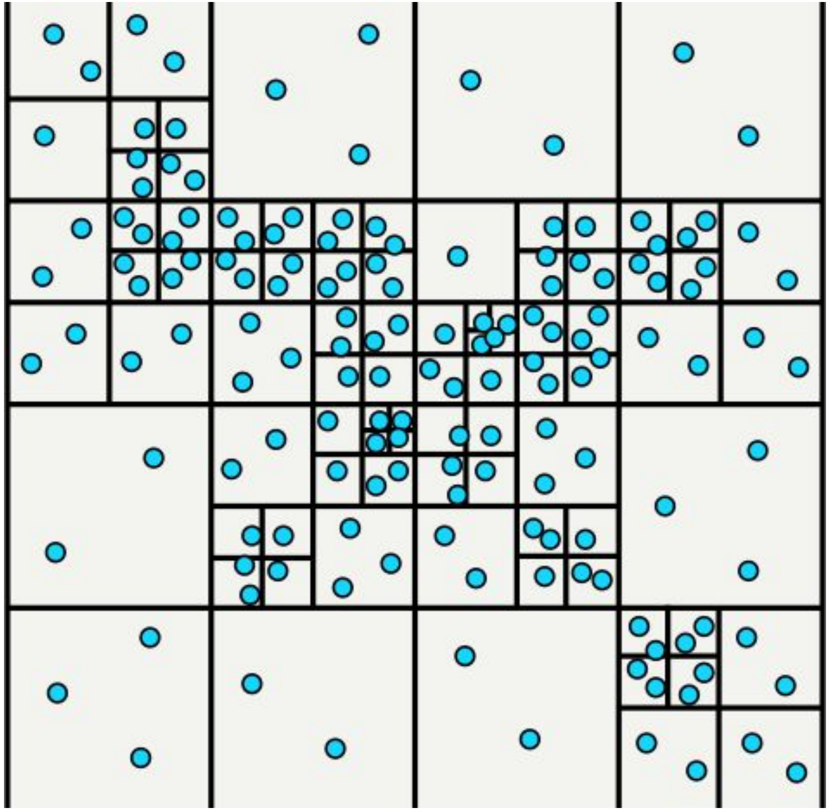


Borrow, MS, +2021

Astro-SPH: The traditional method

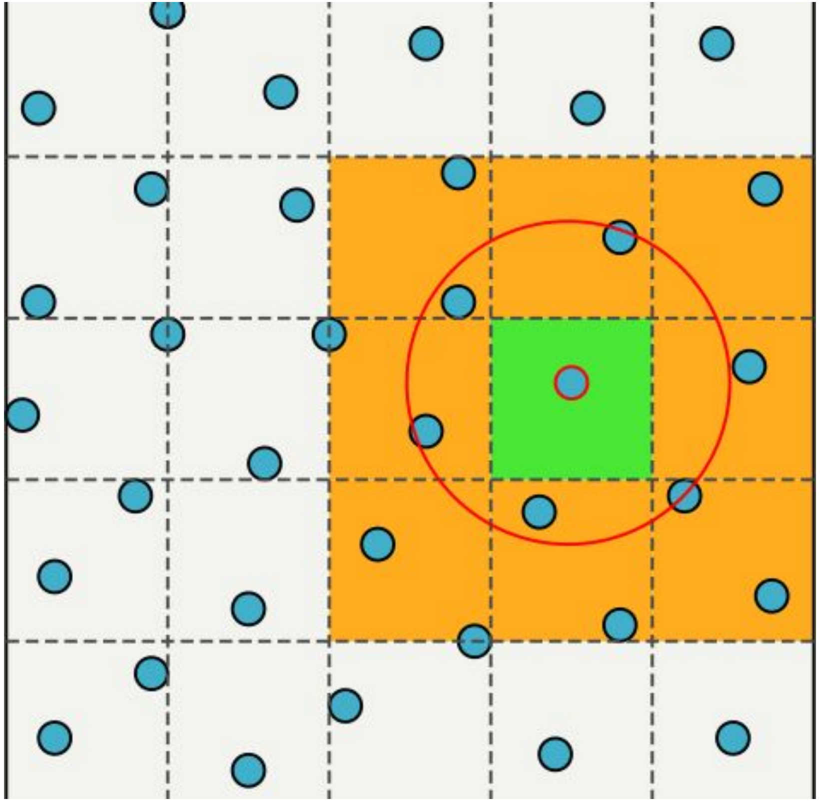
- TREESPH (Hernquist & Katz 1989)
- Neighbour search based on oct-tree.
- Technique used a lot: Gadget-[234], Gasoline-[12], PHANTOM, ...

This makes sense. Easy, robust, and parallelisable.



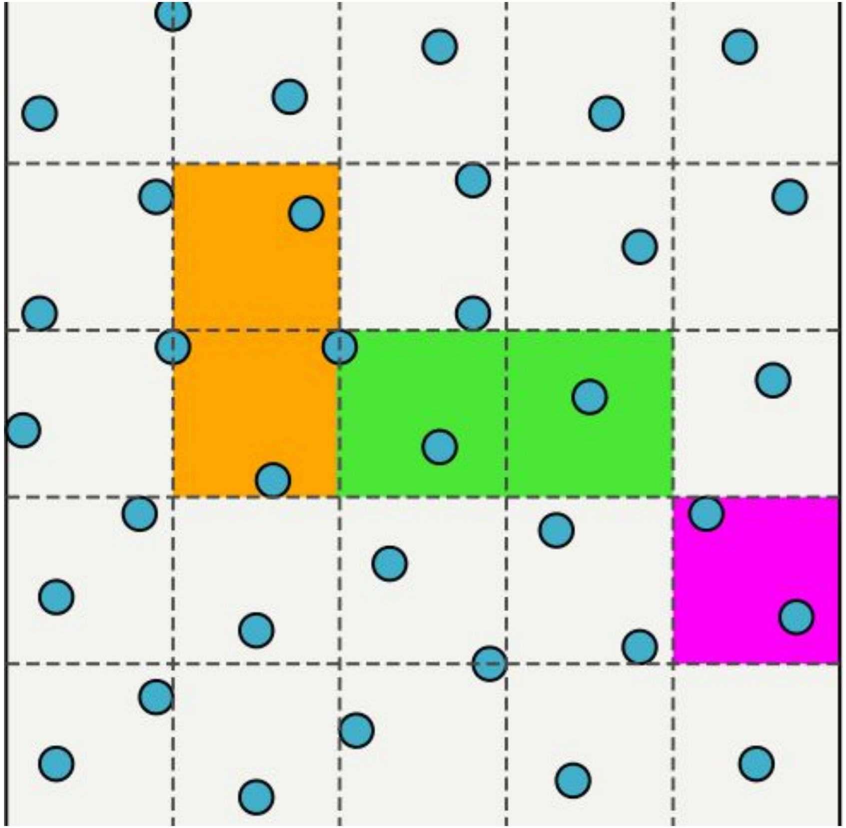
Astro-SPH: The SWIFT way

- Target ~500 particles per cell via adaptive mesh refinement.
- Cell size naturally matches particle neighbour search radius.
- Particles only interact with particles in the same cell or any direct neighbouring cell.



Astro-SPH: The SWIFT way

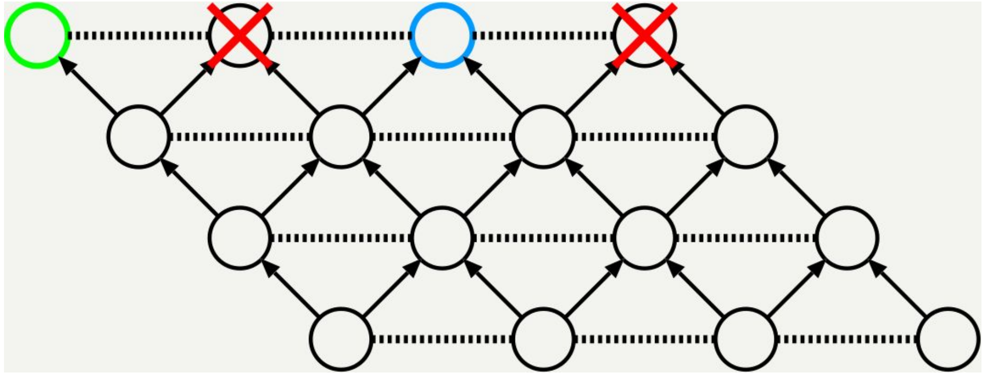
- Cells pairs do not need to be processed in any pre-defined order.
- Only need to make sure two threads do not work on the same cell.
- Cell pairs can have vastly different work-loads.



→ **Need runtime dynamic scheduling**

Task-based parallelism

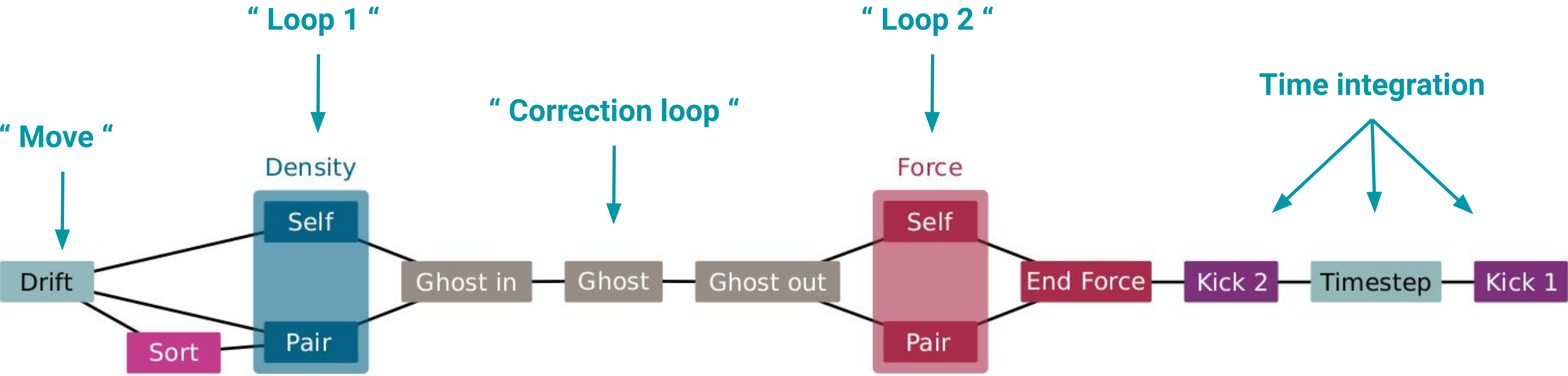
- Shared-memory parallel programming paradigm in which the computation is formulated in an implicitly parallelizable way that automatically avoids most of the problems associated with concurrency and load-balancing.



- We first reduce the problem to a set of inter-dependent tasks.
- For each task, we need to know:
  - Which tasks it depends on,
  - Which tasks it conflicts with.
- Each thread then picks up a task which has no unresolved dependencies or conflicts and computes it.
- We use our own Open-source library `QuickSched` (arXiv:1601.05384)

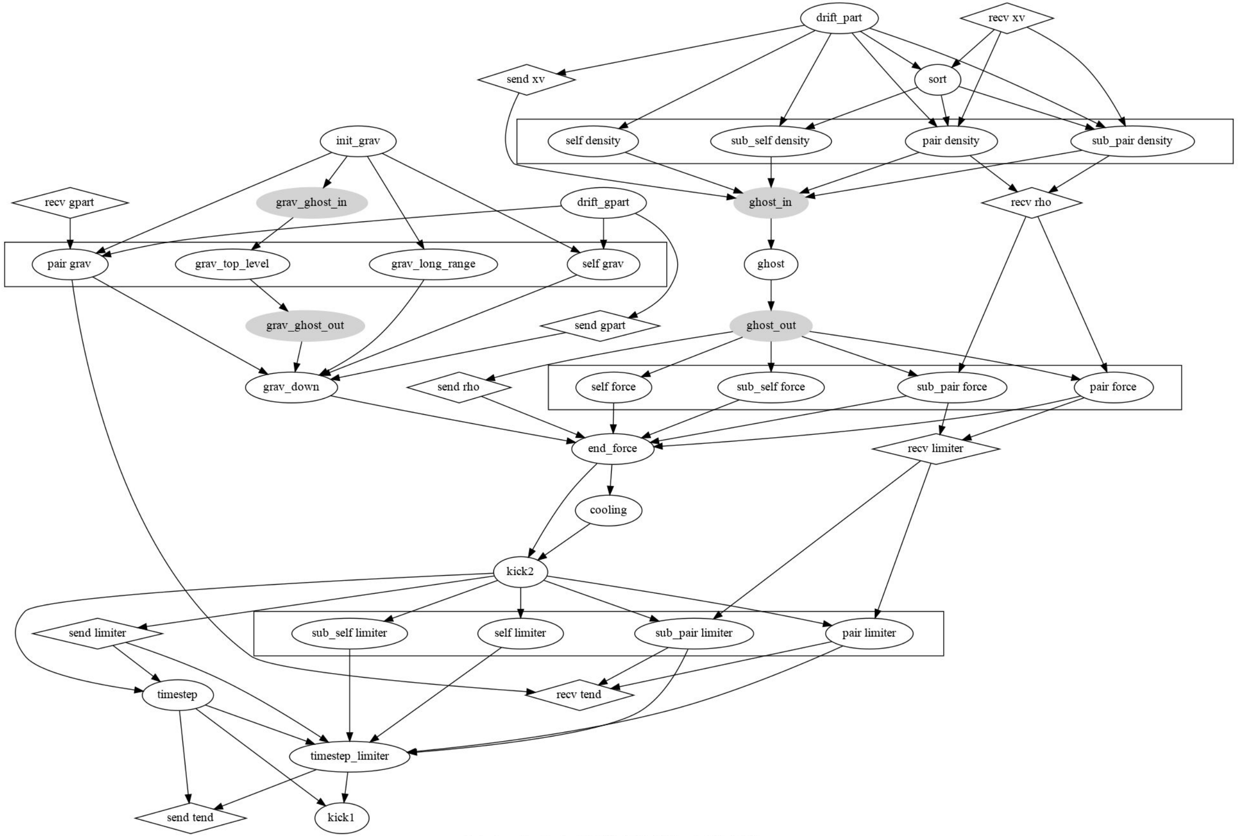
Task-based parallelism for SPH

What happens to one cell "bundle" of particles during one time-step:



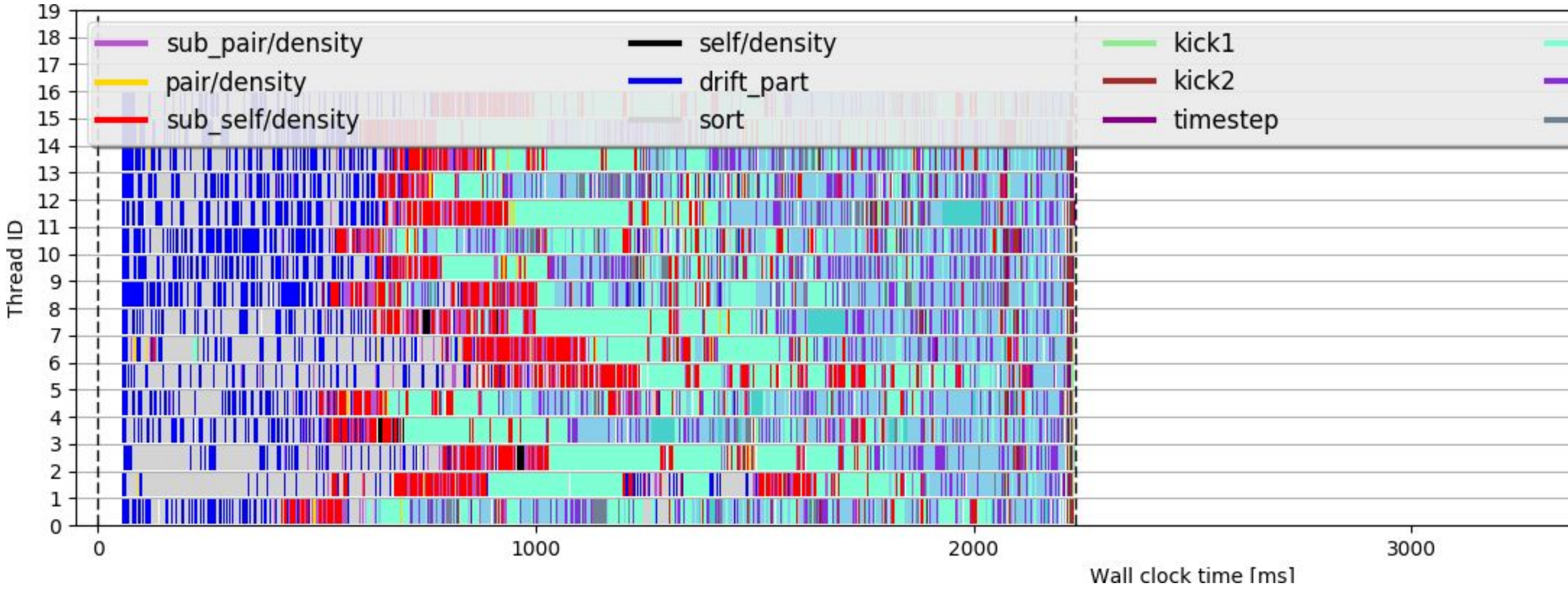
All the code within a task is very simple. No need for deep C knowledge
—> Easy to extend the code

With all the physics

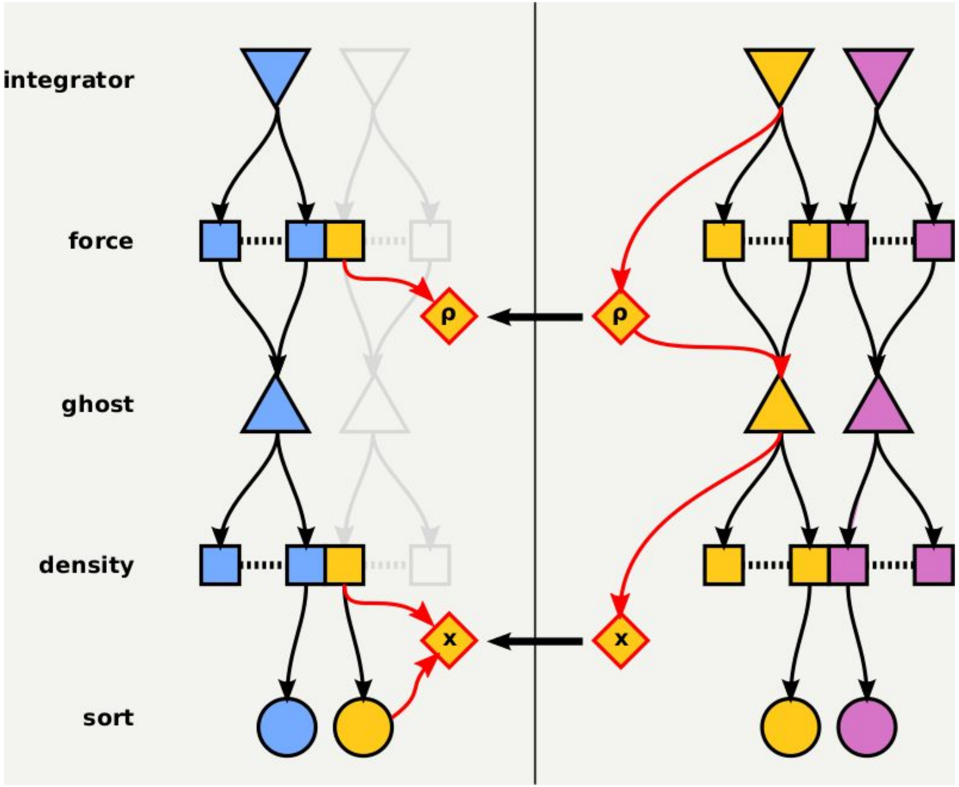


Task dependencies for SWIFT v0.6.0-1090-ga8e380a5-dirty

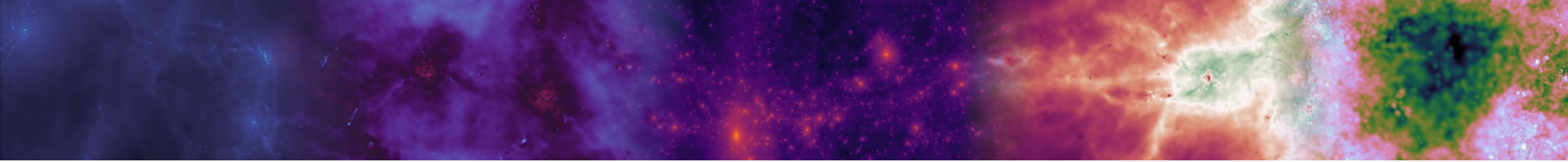
Task-based parallelism in action



How about multiple nodes?

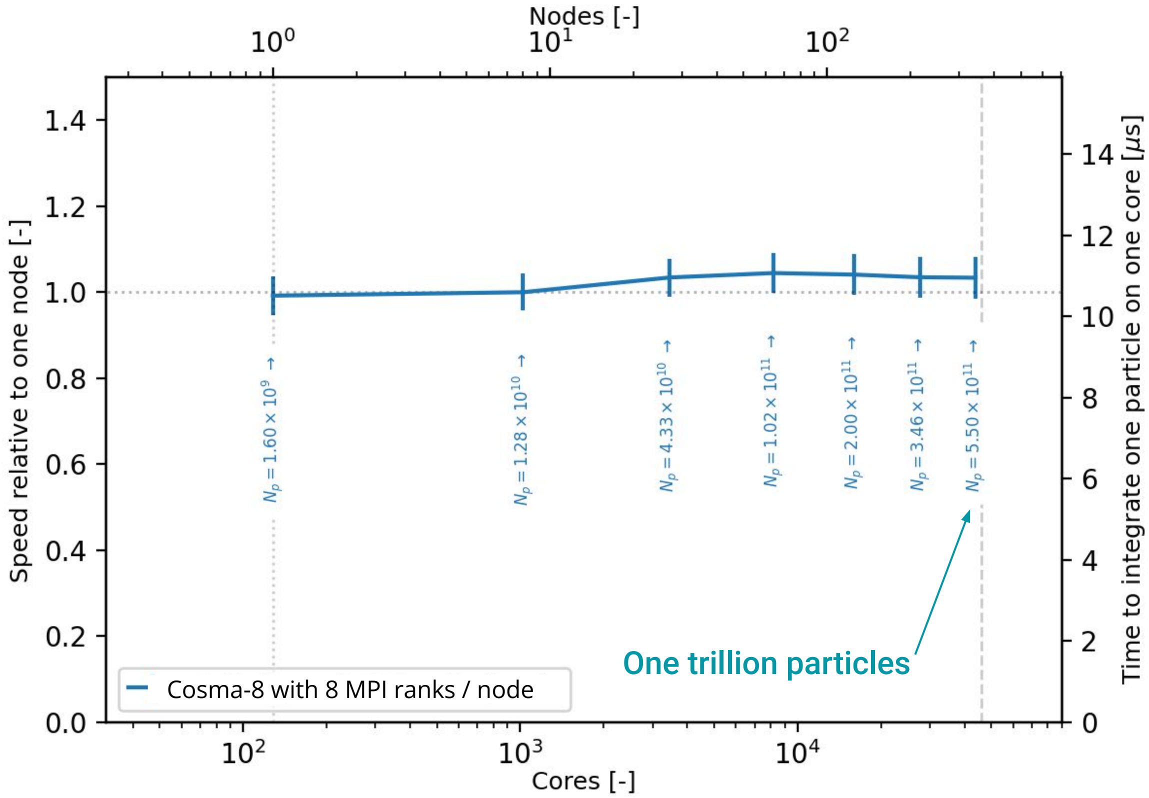


- Instead of sending all the particles and *then* compute, do it at the same time.
- Sending/receiving data is just another task type, and can be executed in parallel with the rest of the computation.
- Once the data has arrived, the scheduler unlocks the tasks that needed the data.




pair/density
sub_pair/density
sub_self/density
self/density
self/grav
pair/grav
grav_mm
sort
kick2
timestep
drift_part
ghost
drift_gpart
init_grav
grav_down
kick1
grav_mesh
end_force
grav_long_range
sub_pair/force
sub_self/force
pair/force
self/force
send/tend
send/xv
send/rho
send/gpart
recv/tend
recv/gpart
recv/rho
recv/xv
Thread ID
Wall clock time [ms]

Weak-scaling to large systems



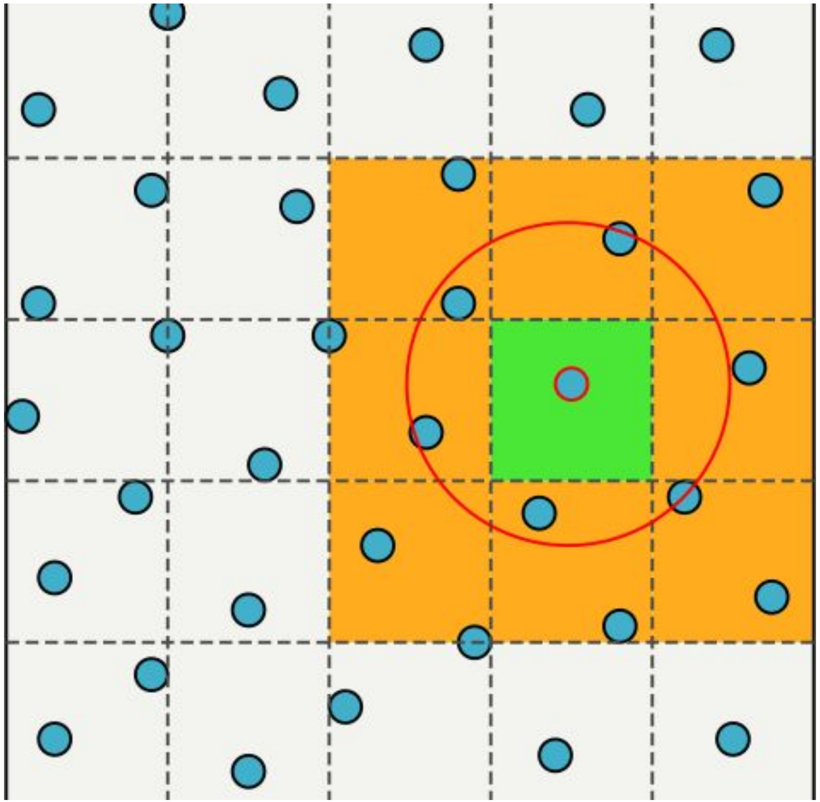
`DiRAC Cosma-8` **system @ Durham.**

360 nodes with
- **2x AMD 7H12**
- **1 TB of RAM**
- **HDR Inter-connect**

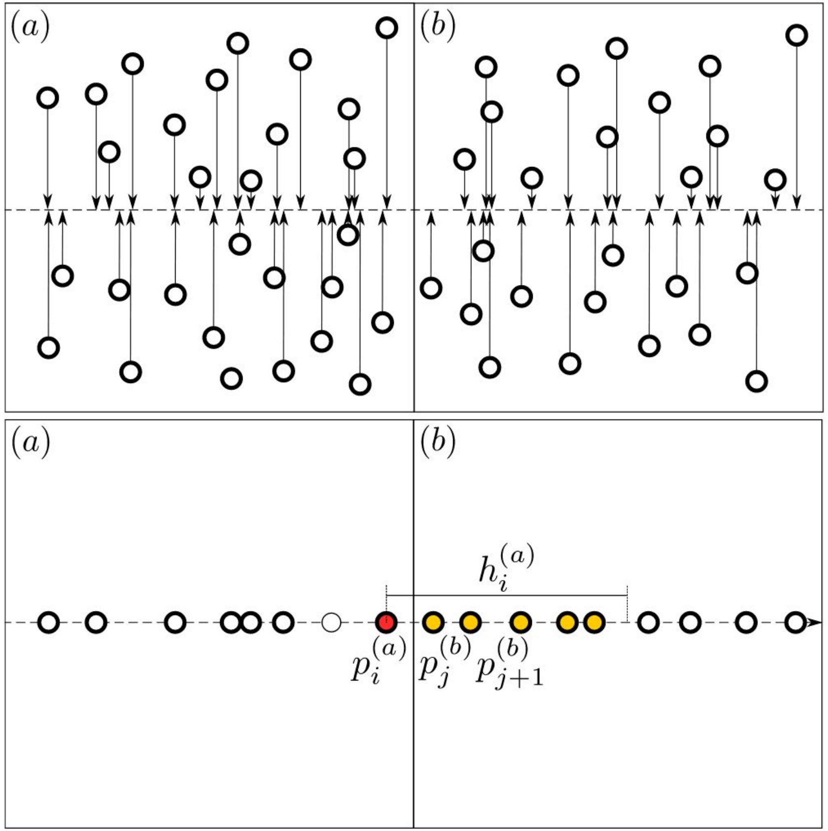
Rogers+2022

Verlet list detail

Particle Interactions



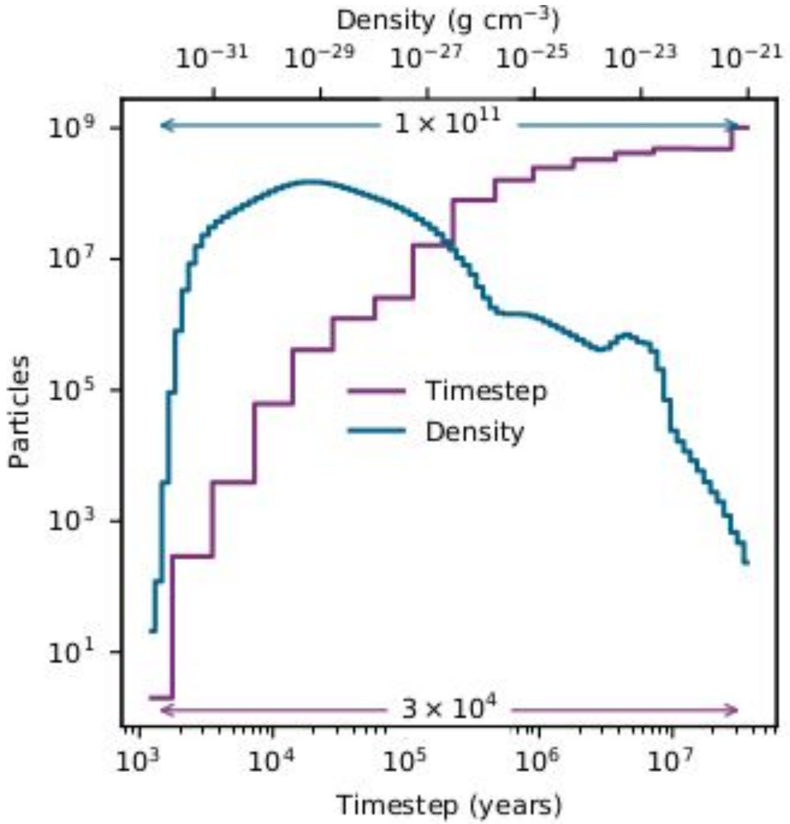
pseudo-Verlet lists

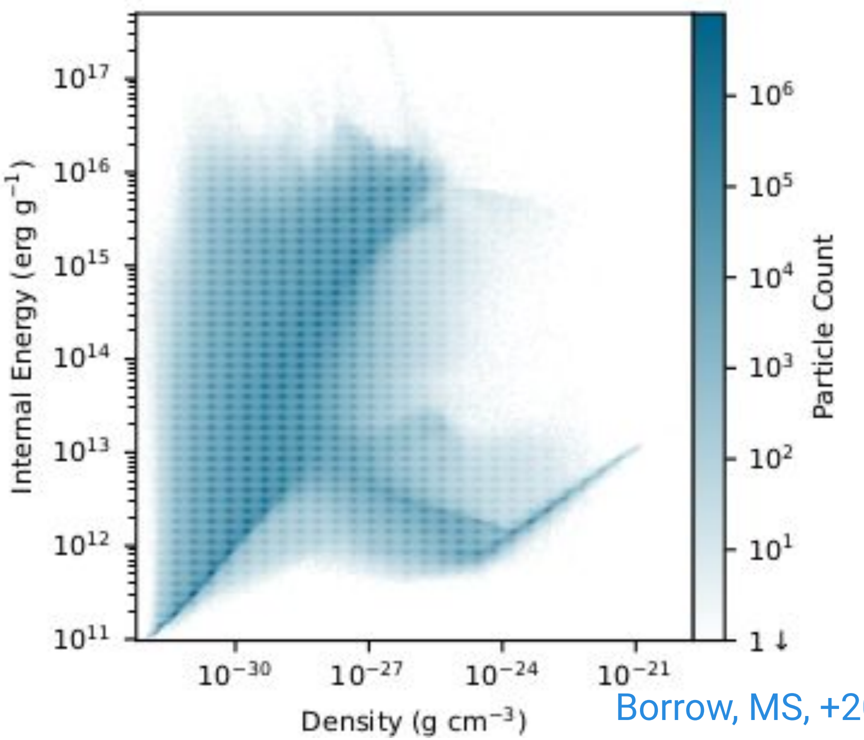


Willis, MS, +2018


Local Δt “fun”

Localized time-stepping





Borrow, MS, +2018

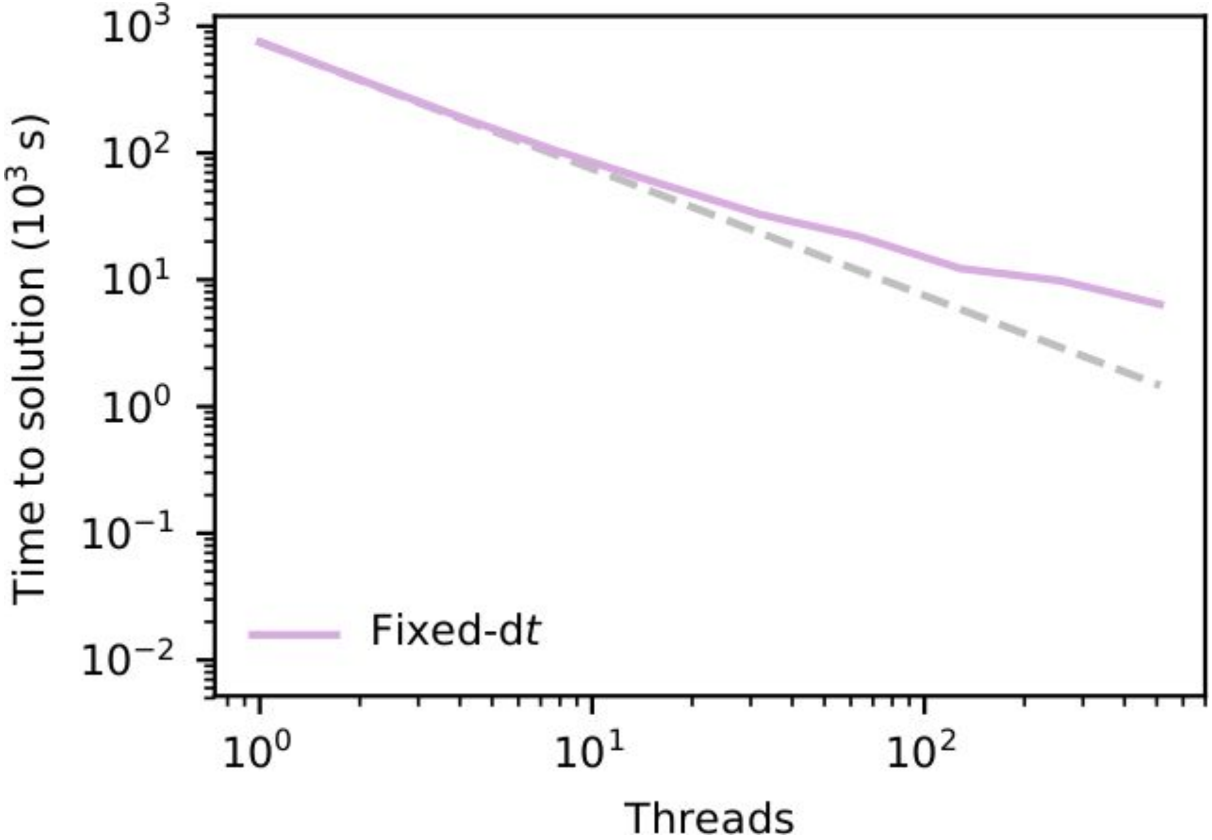
Time integration

Classic *leapfrog (Velocity Verlet)*:

$K(\Delta t/2) \times D(\Delta t) \times K(\Delta t/2)$

Where each particle has the same timestep, i.e. the smallest Δt in the simulation

Efficiency



Time to solution (10^3 s)
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
Fixed-dt
10^0
10^1
10^2
Threads

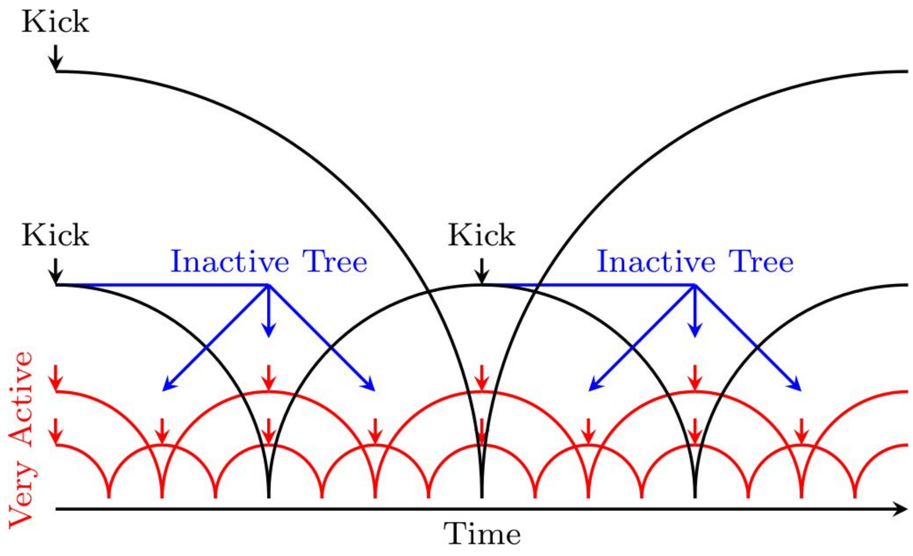
Time integration operator splitting

Classic *leapfrog (Velocity Verlet)*:

$$K(\Delta t/2) \times D(\Delta t) \times K(\Delta t/2)$$

Splitting the "drift":

$$K(\Delta t/2) \times D(\Delta t/2^n) \cdots D(\Delta t/2^n) \times K(\Delta t/2)$$



Potter+2017

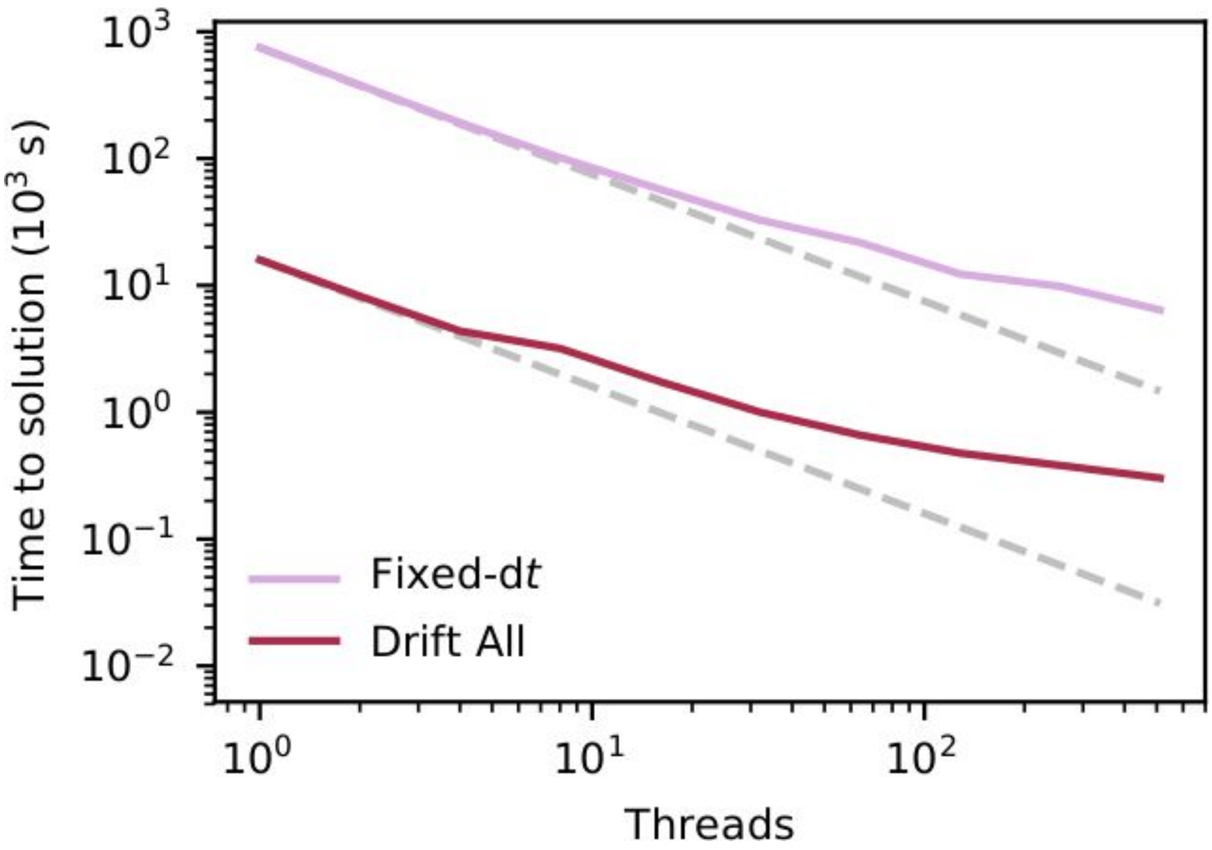
More implementation problems...

Localized time-stepping is great. But...

Need to select particles to update, or maintain lists, or sort, ...

→ ***Proportionally***, more “logic” and less “compute”.

Efficiency



Time to solution (10^3 s)
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
Fixed-dt
Drift All
10^0
10^1
10^2
Threads

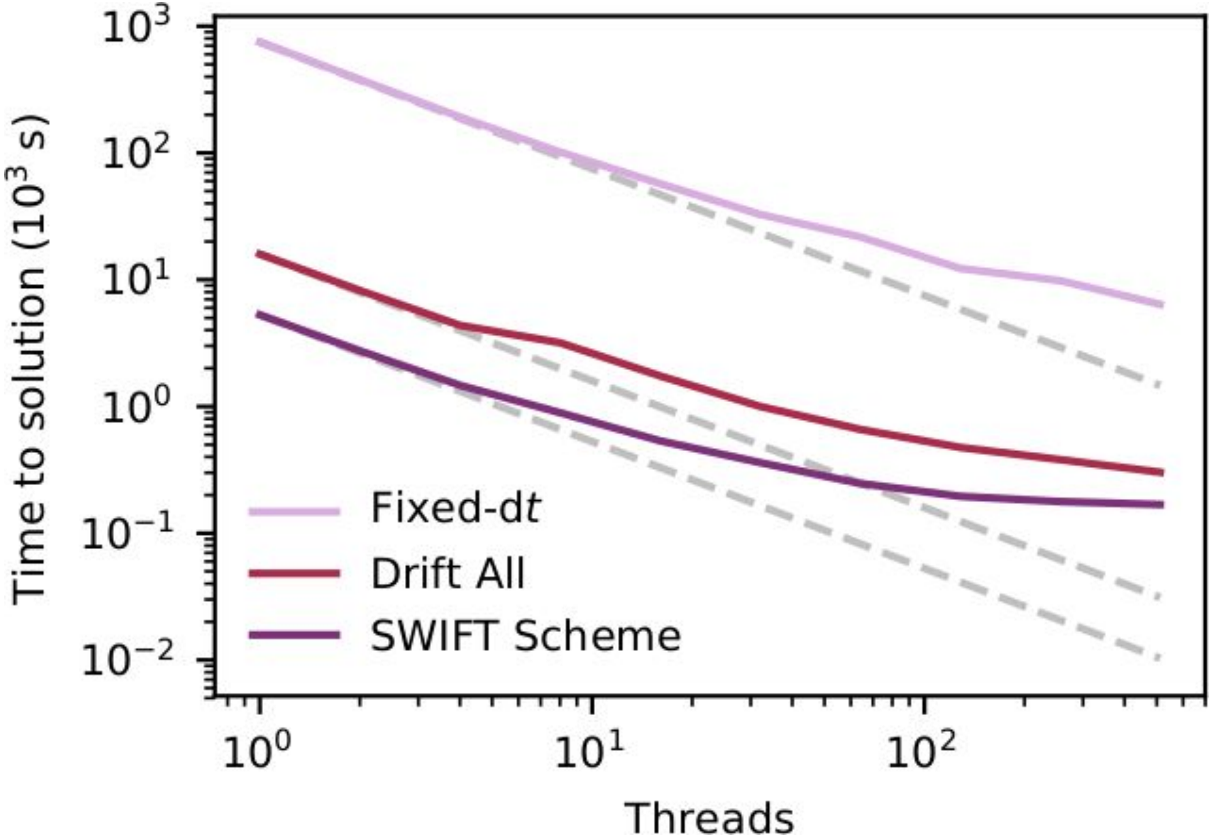
Going beyond

The canonical algorithm drifts **all** the particles to the current point in time.

Do we need that? No! Only the particles that are neighbours of an active particle need to be moved forward.

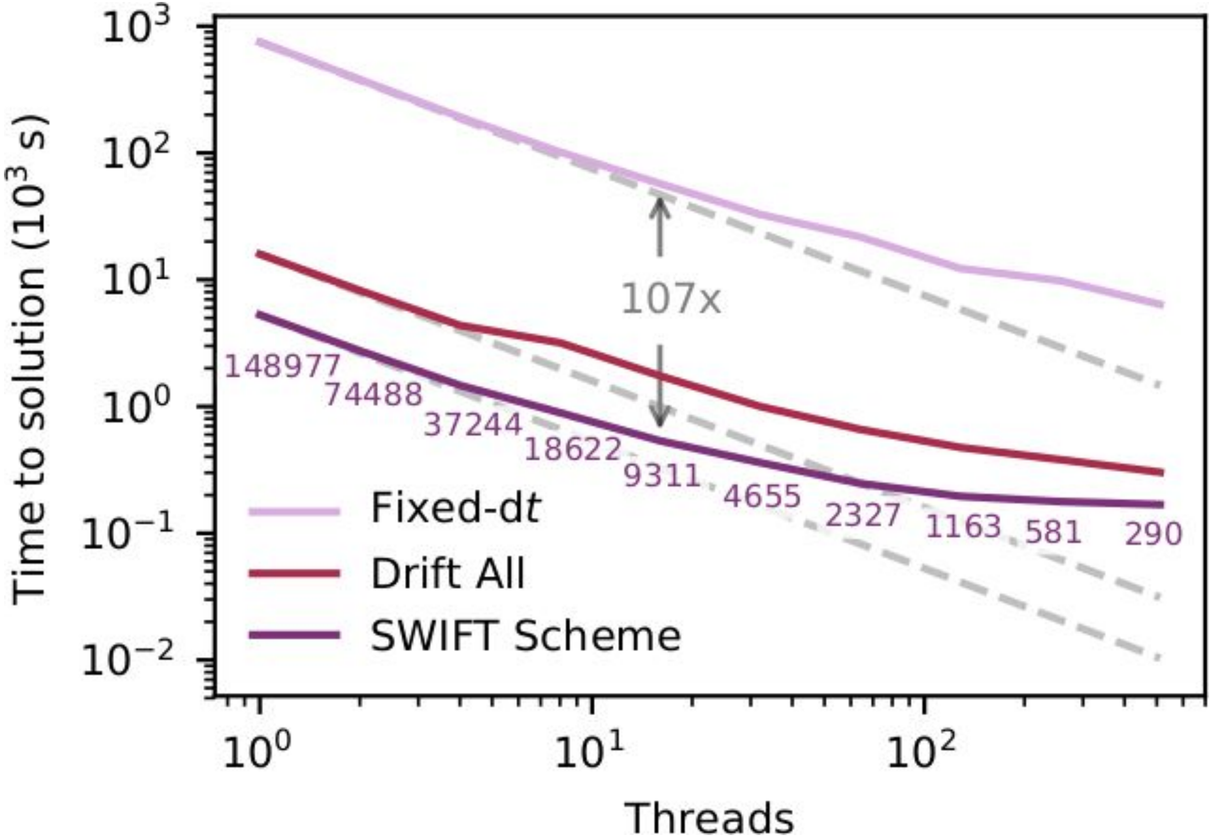
-> Tree-walk “activating” the tasks in parts of the domain that need to. Followed by the actual calculation.

Efficiency

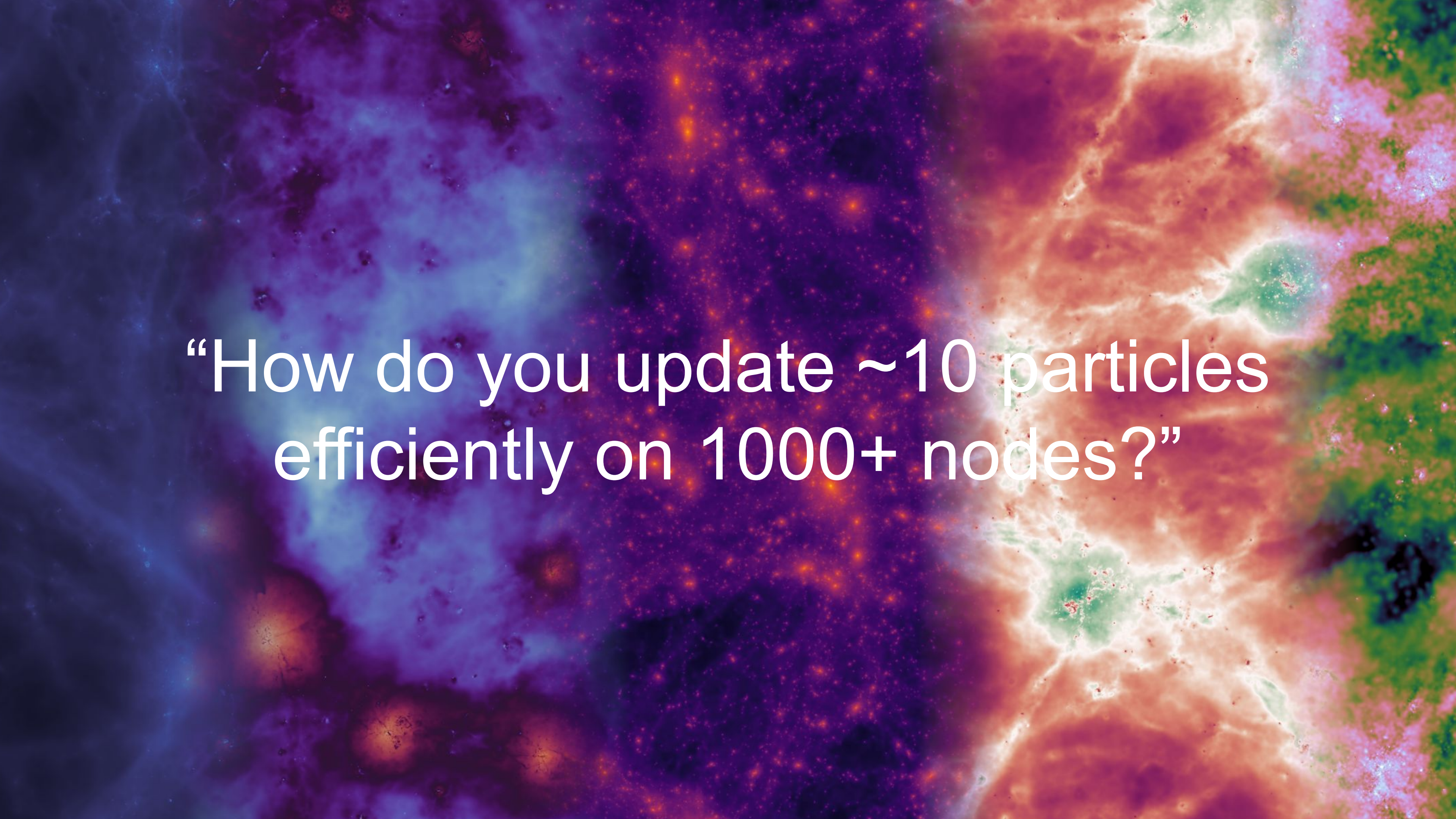


Time to solution (10^3 s)
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
Fixed-dt
Drift All
SWIFT Scheme
10^0
10^1
10^2
Threads

Not enough stuff to do

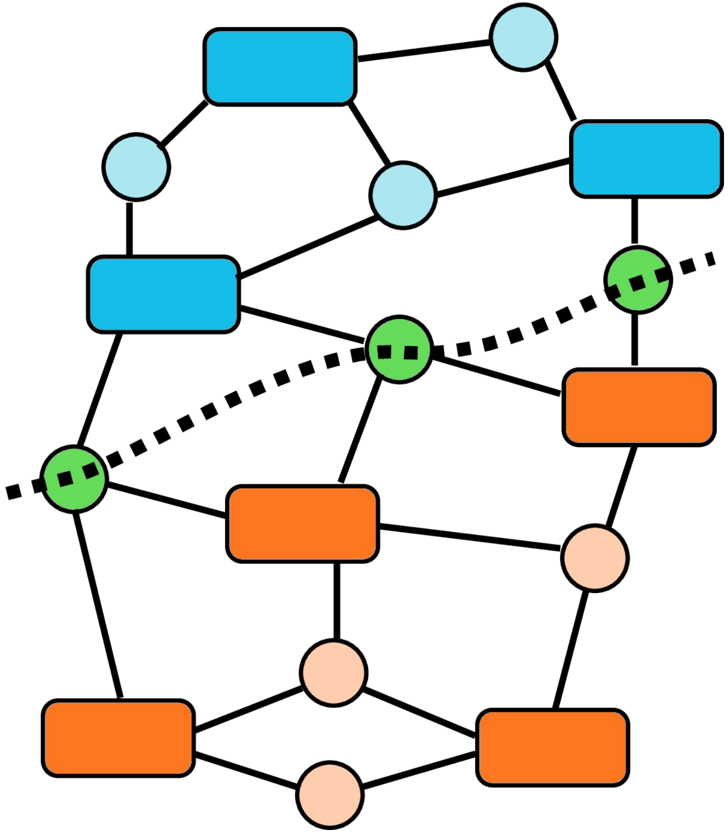



How do we load-balance this efficiently?



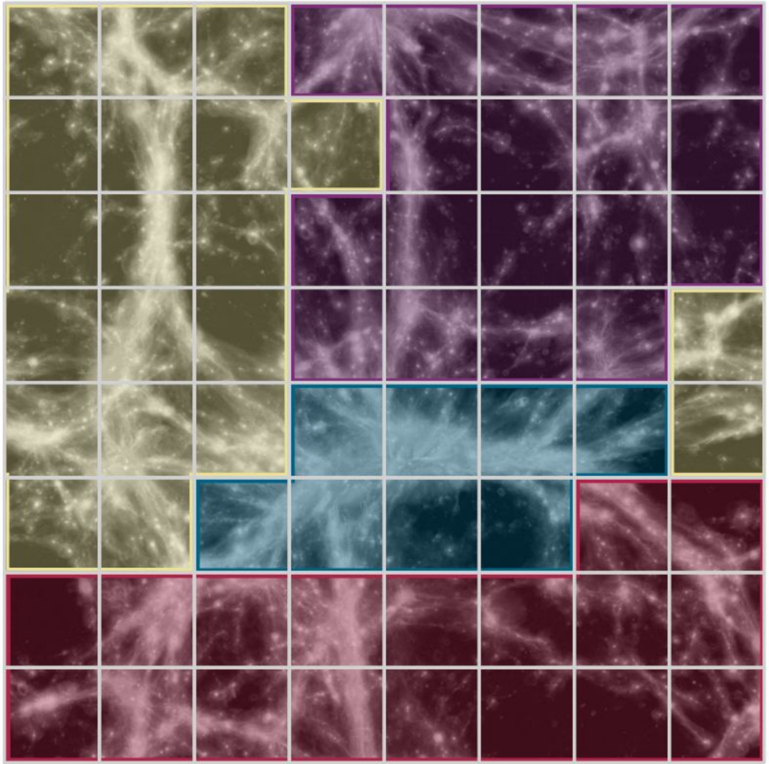
“How do you update ~10 particles efficiently on 1000+ nodes?”

A Graph-based strategy

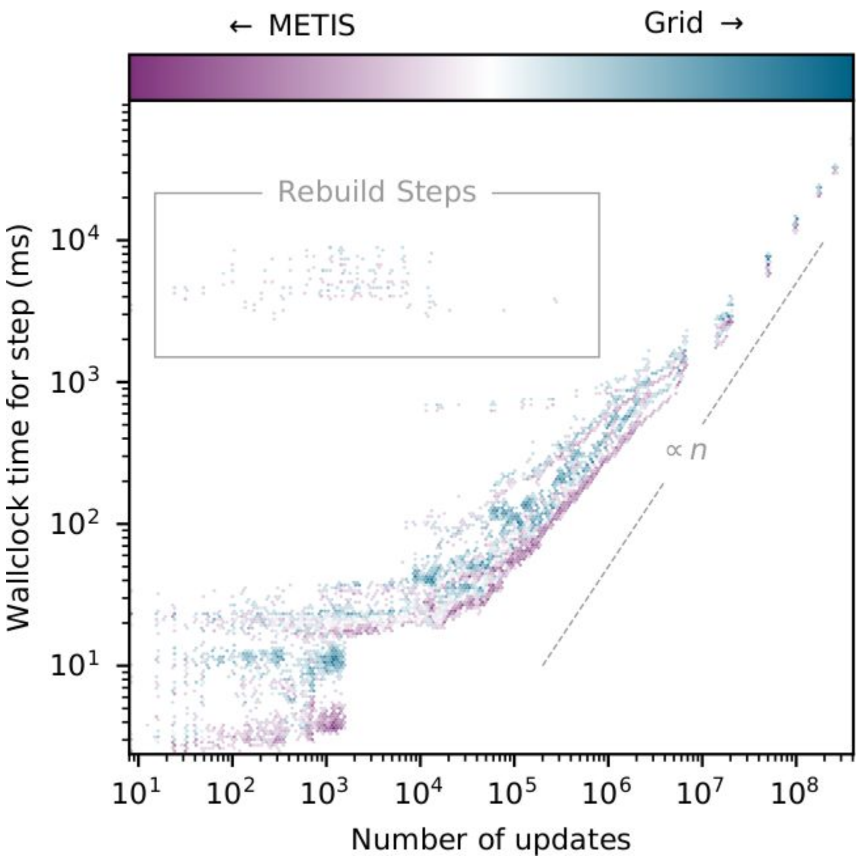


- For each task, we compute the amount of work (=runtime) required.
- We build a graph where the data are nodes and tasks are hyper-edges.
- Extra cost added for communication tasks to minimise them.
- `METIS` is used to split the graph such that the work (not the data!) is balanced.

Metis Domain Decomposition



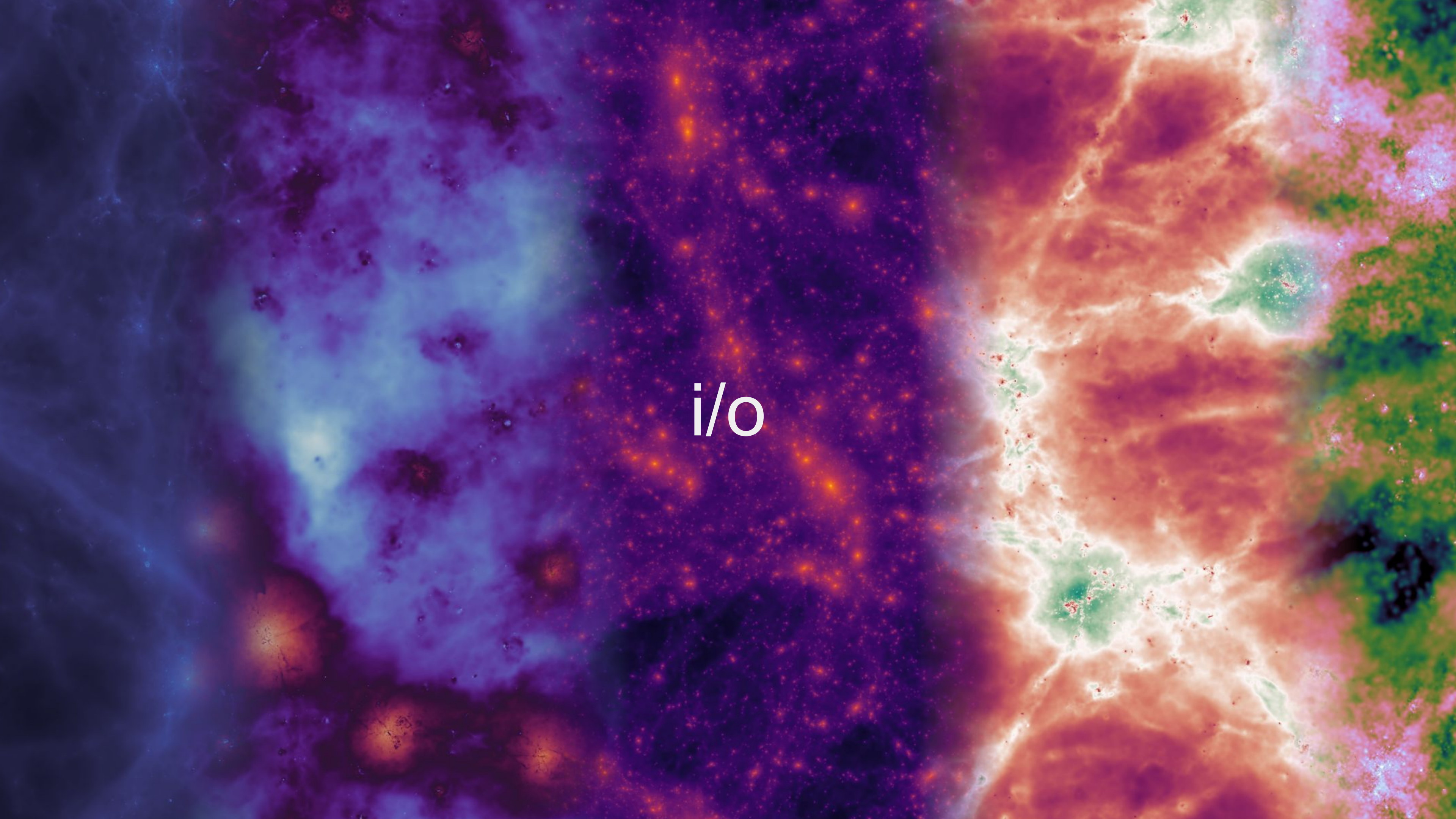
A diagnostic



400 × 10^6 particles

4 nodes

16 cores/node


i/o

Two strategies

`hdf5`-based snapshots.

- Exploits MPI-io under the hood.
- Compression (lossy and lossless)
- Achieves ~30% of peak on cosma's lustre system (when not compressing).

Continuous Simulation Data Stream

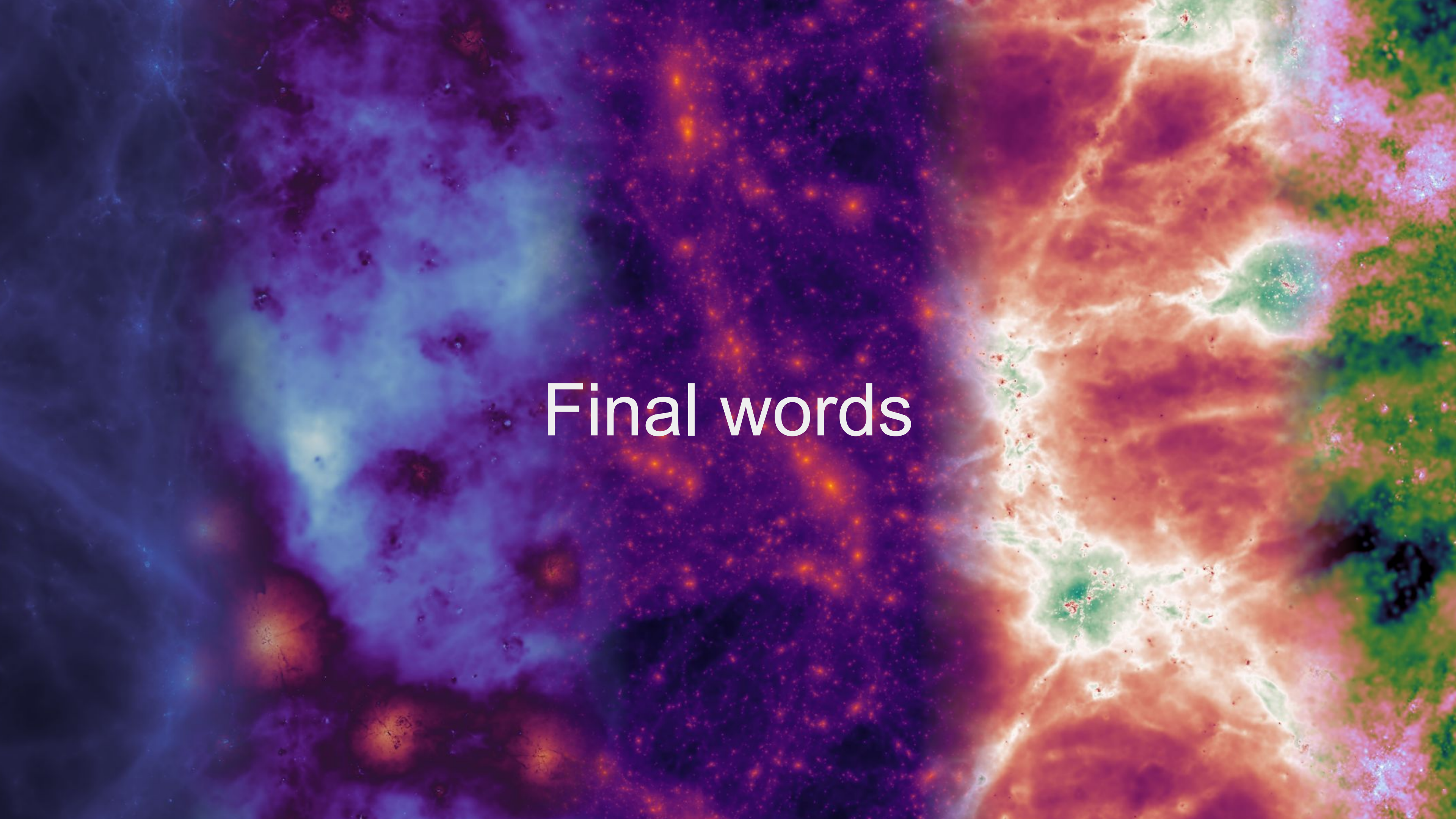
- Write particle ***changes*** to a per-node log file.
- Takes place as a task.
- Use memory-mapped files and the OS lazily writes to disk in the background.

Hausamman, MS, +2022

Planetary Impacts

J. A. Kegerreis *et al* 2022 *ApJL* **937** L40

Final words



References

- Website: www.swiftsim.com
- Paper: https://ui.adsabs.harvard.edu/abs/2023arXiv230513380S/abstract
- Source code: https://github.com/SWIFTSIM/SWIFT