Multigrid Self-Gravity in AthenaK

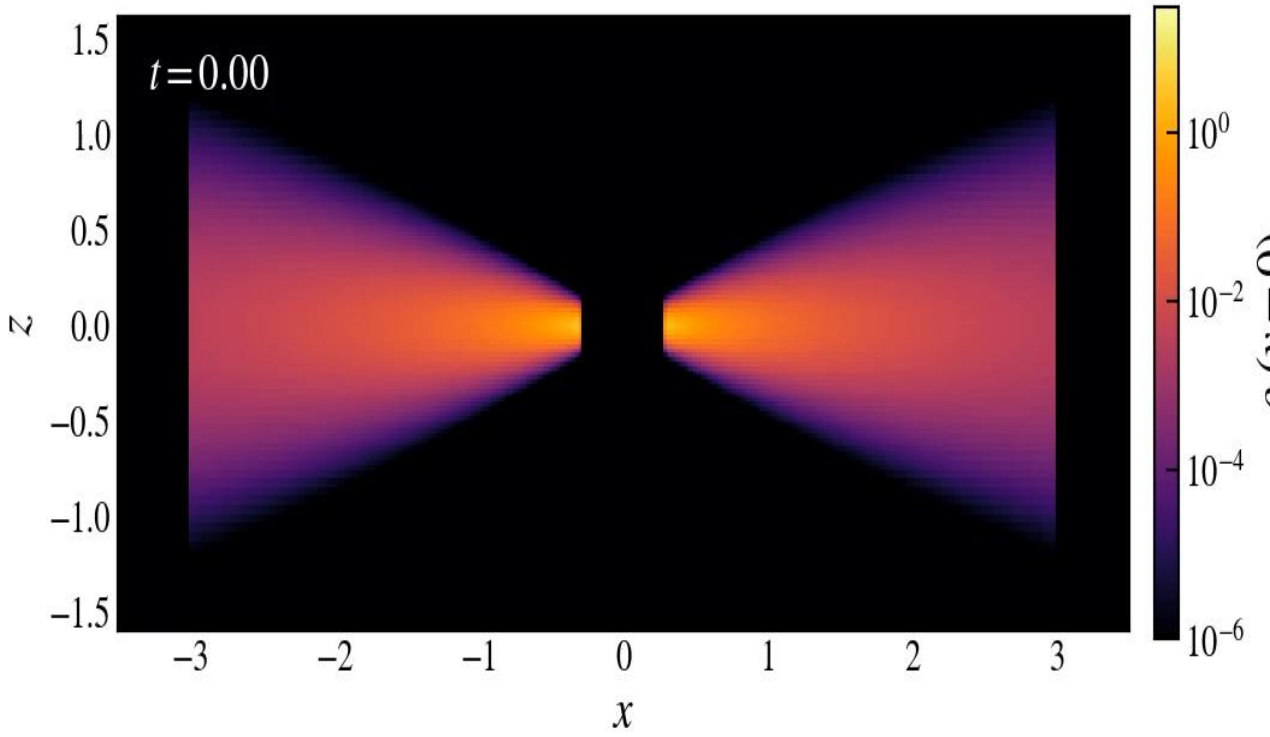
A performance-portable Poisson solver for hydro & MHD

David Velasco · **Institute for Advanced Study**

MOTIVATION

Self-gravity is an elliptic problem at every step

- **Coupling.** Gas feels its own gravity through source terms $-\rho\nabla\Phi$ (momentum) and $-\rho\, v\cdot\nabla\Phi$ (energy).
- **Poisson.** The potential follows $\nabla^2\Phi = 4\pi G\rho$ — an elliptic equation.
- **Global.** Elliptic means every cell couples to every other; there is no local update rule.
- **Hard constraint.** It must be solved accurately, every timestep, in lock-step with the (MHD) evolution.



Video courtesy of Yang Ni: Gravitationally unstable disk computed by the multigrid solver (with SMR).

MOTIVATION

Six ways to solve an elliptic problem

Direct solvers $\gtrsim O(N^2)$

+ exact answer, no iteration, any operator

− memory and cost explode with resolution

Spectral / FFT $O(N \log N)$

+ fast and robust on one uniform box

− periodic, uniform grids only; global transposes

Relaxation (Jacobi, GS, SOR) $\sim O(N^{5/3})$

+ trivially simple, local, matrix-free

− stalls on smooth error — unusable alone

Multigrid $O(N)$

+ optimal cost; AMR-native; mixed BCs; local comms

− more moving parts — the rest of this talk

N = total number of cells. Every family trades generality against cost — only multigrid reaches O(N) while staying compatible with refined meshes and mixed boundary conditions. Next: what AthenaK actually needs.

Deriving the Jacobi update

STEP 1 — From the Poisson equation to a nodal update

$$\nabla^2\Phi = 4\pi G\rho \quad \rightarrow \quad (\Phi_{i+1} - 2\Phi_i + \Phi_{i-1})/h^2 = f_i \quad \rightarrow \quad \Phi_i = \tfrac{1}{2}(\Phi_{i+1} + \Phi_{i-1} - h^2 f_i)$$

Discretize the Laplacian, then solve the stencil for the central node. Sweeping this over every point (from the old values) is one Jacobi iteration.

STEP 2 — The relaxation factor ω

$$\Phi_i \leftarrow \omega \cdot \tfrac{1}{2}(\Phi_{i+1} + \Phi_{i-1} - h^2 f_i) + (1 - \omega) \cdot \Phi_i$$

ω blends the fresh Jacobi estimate with the old value — it controls how far each sweep moves the solution.

ω = 1 · Standard Jacobi

Take the full update. Converges, but removes high-frequency error only slowly.

ω = 2⁄3 · Damped

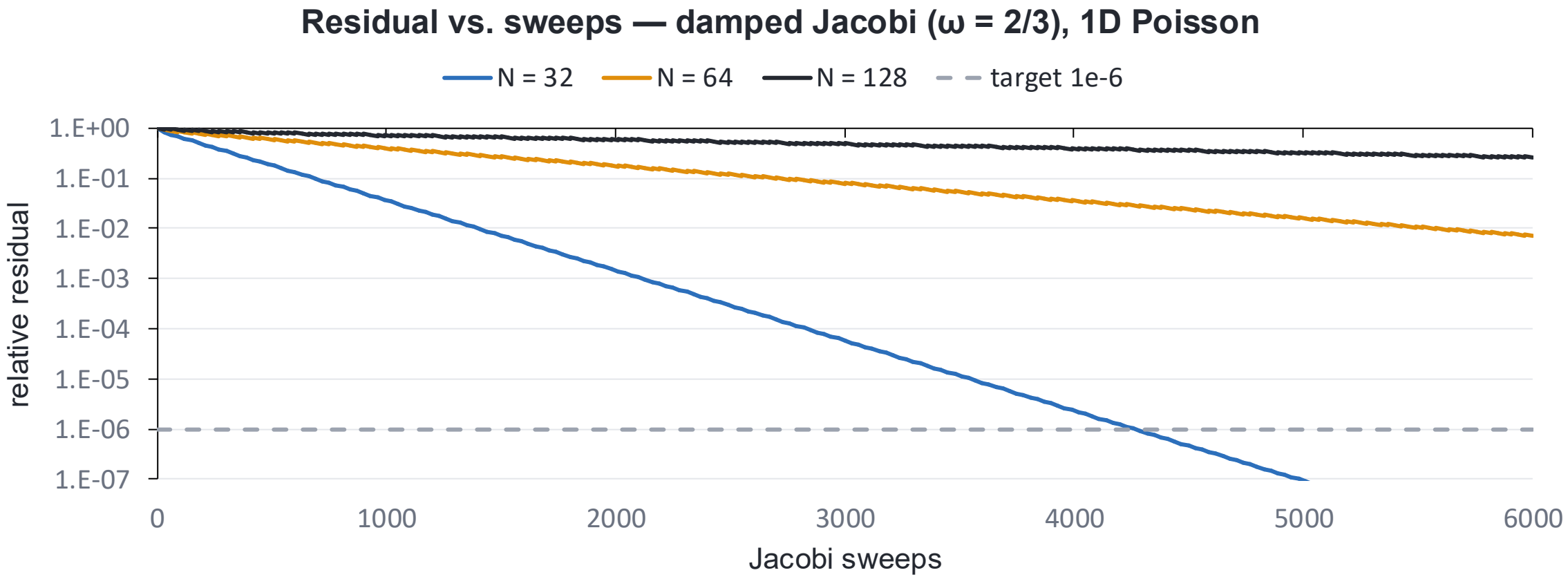
Under-relaxation. Damps oscillatory error fastest — the smoother that makes multigrid work.

ω > 1 · Over-relaxation

Overshoots the update. Faster on smooth error, but diverges if pushed too far.

Start simple: relaxation

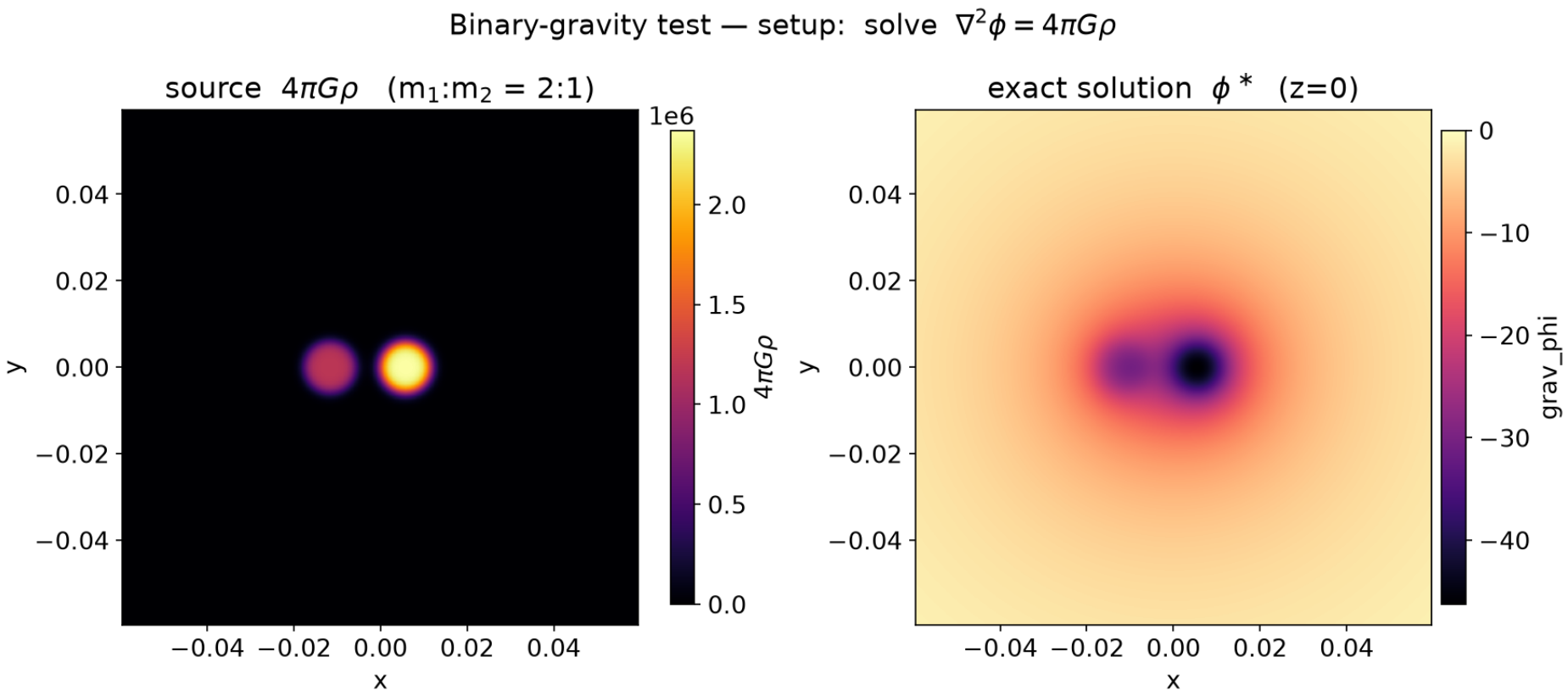
$$\nabla^2\Phi = 4\pi G\rho \;\longrightarrow\; (\Phi_{i+1} - 2\Phi_i + \Phi_{i-1})/h^2 = f_i \;\longrightarrow\; \text{Jacobi: } \Phi_i \leftarrow \omega\cdot\tfrac{1}{2}(\Phi_{i+1} + \Phi_{i-1} - h^2 f_i) + (1-\omega)\Phi_i$$



Classical iterative solvers do converge — but the number of sweeps needed grows as O(N²): doubling the resolution quadruples the work for every digit of accuracy.

Start simple: relaxation

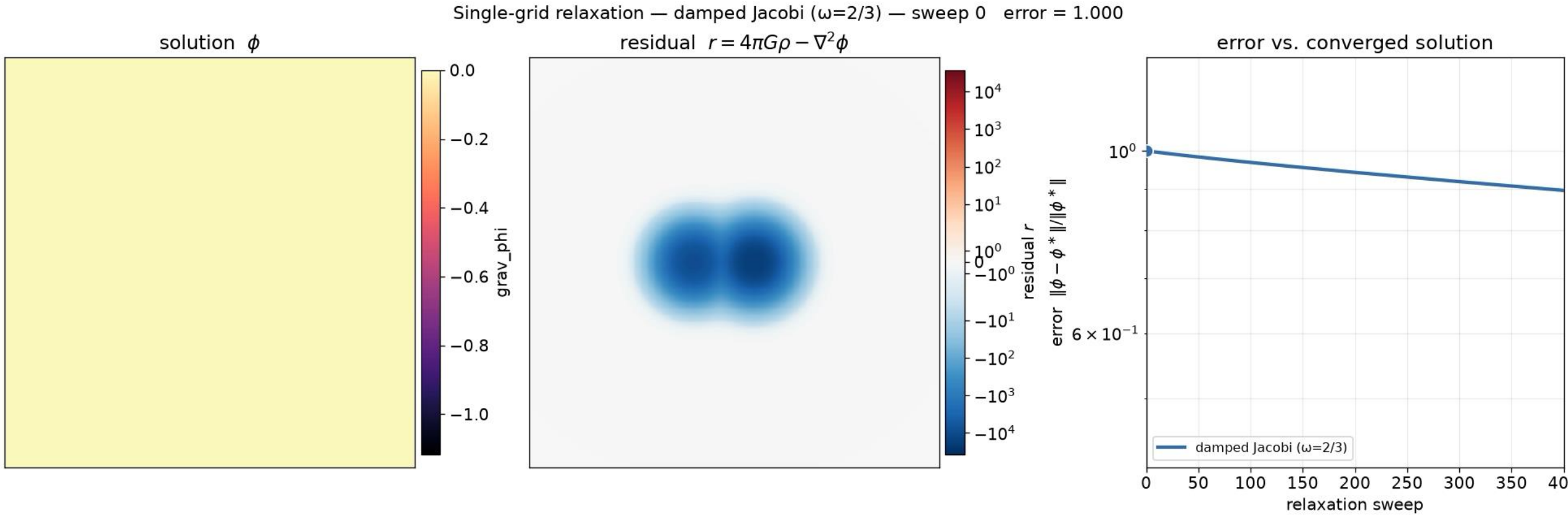
$$\nabla^2\Phi = 4\pi G\rho \;\longrightarrow\; (\Phi_{i+1} - 2\Phi_i + \Phi_{i-1})/h^2 = f_i \;\longrightarrow\; \text{Jacobi: } \Phi_i \leftarrow \omega\cdot\tfrac{1}{2}(\Phi_{i+1} + \Phi_{i-1} - h^2 f_i) + (1-\omega)\Phi_i$$



Classical iterative solvers do converge — but the number of sweeps needed grows as O(N²): doubling the resolution quadruples the work for every digit of accuracy.

Start simple: relaxation

$$\nabla^2\Phi = 4\pi G\rho \;\longrightarrow\; (\Phi_{i+1} - 2\Phi_i + \Phi_{i-1})/h^2 = f_i \;\longrightarrow\; \text{Jacobi: } \Phi_i \leftarrow \omega\cdot\tfrac{1}{2}(\Phi_{i+1} + \Phi_{i-1} - h^2 f_i) + (1-\omega)\Phi_i$$



Classical iterative solvers do converge — but the number of sweeps needed grows as O(N²): doubling the resolution quadruples the work for every digit of accuracy.

From Jacobi to red–black Gauss–Seidel

STEP 1 — Gauss–Seidel: use new values as soon as they exist

Jacobi: $\Phi'_i = \frac{1}{2}(\Phi_{i+1} + \Phi_{i-1} - h^2 f_i)$

Gauss–Seidel: $\Phi'_i = \frac{1}{2}(\Phi_{i+1} + \Phi'_{i-1} - h^2 f_i)$

′ marks values already updated in this sweep. Consuming them immediately roughly halves the error per sweep and needs no second copy of Φ — but it makes each update depend on the previous one.

$$\Phi_i \leftarrow (1-\omega)\cdot\Phi_i + \omega\cdot\Phi_i^{Gs}$$

Jacobi · ω = 2⁄3

Rough error modes overshoot and flip sign every sweep — they must be damped (ω < 1).

Gauss–Seidel · ω = 1

Error modes decay monotonically, no sign flips — there is nothing left to damp.

Red–black GS · ω = 1.15

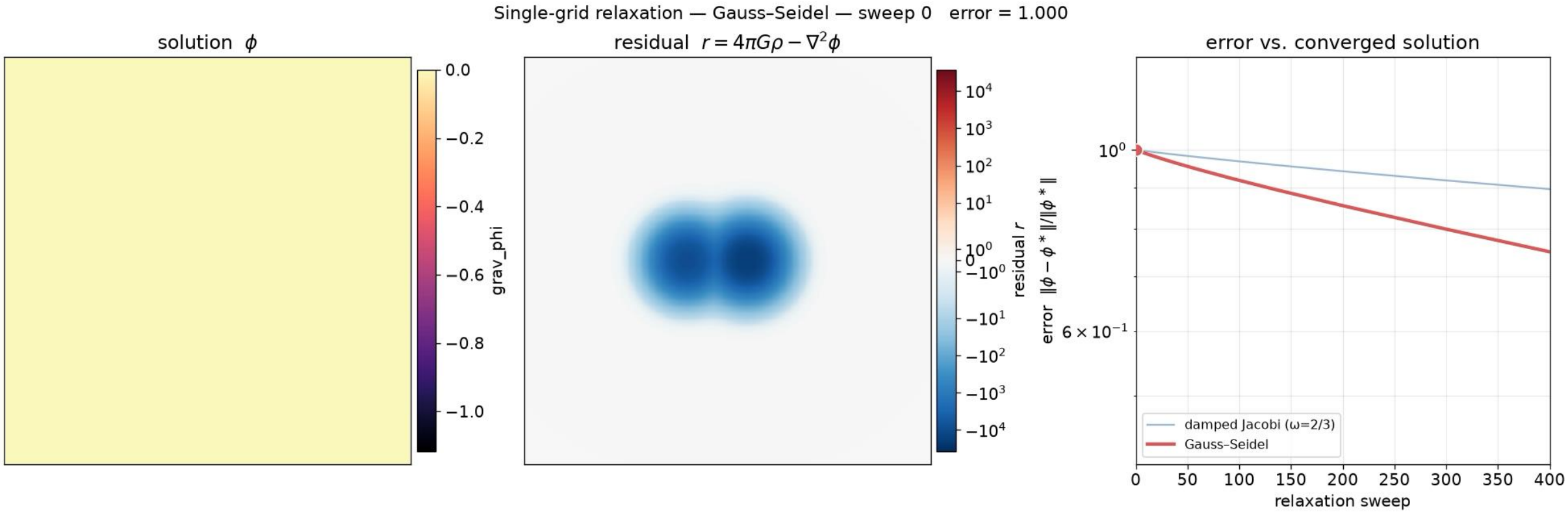
A slight overshoot accelerates the slowest modes — fastest smoothing for Poisson (Yavneh 1996).

From Jacobi to red–black Gauss–Seidel

STEP 1 — Gauss–Seidel: use new values as soon as they exist

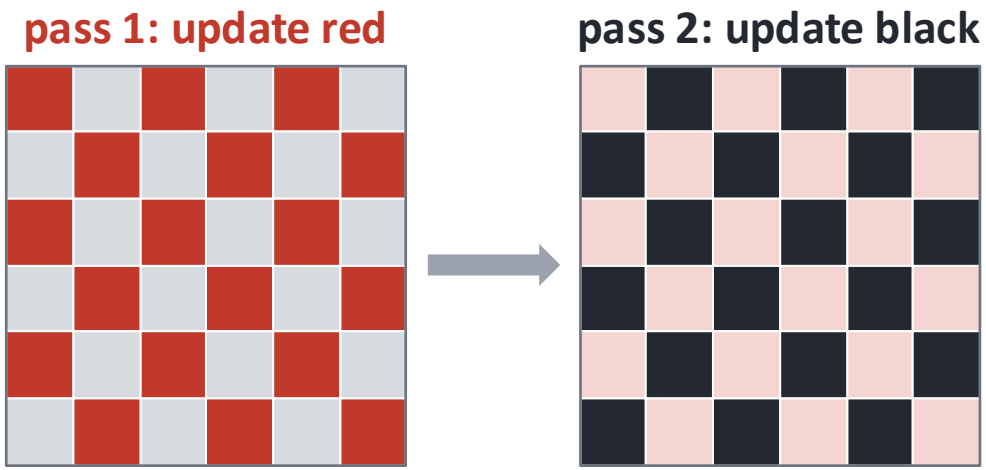
Jacobi: $\Phi'_i = \frac{1}{2}(\Phi_{i+1} + \Phi_{i-1} - h^2 f_i)$

Gauss–Seidel: $\Phi'_i = \frac{1}{2}(\Phi_{i+1} + \Phi'_{i-1} - h^2 f_i)$



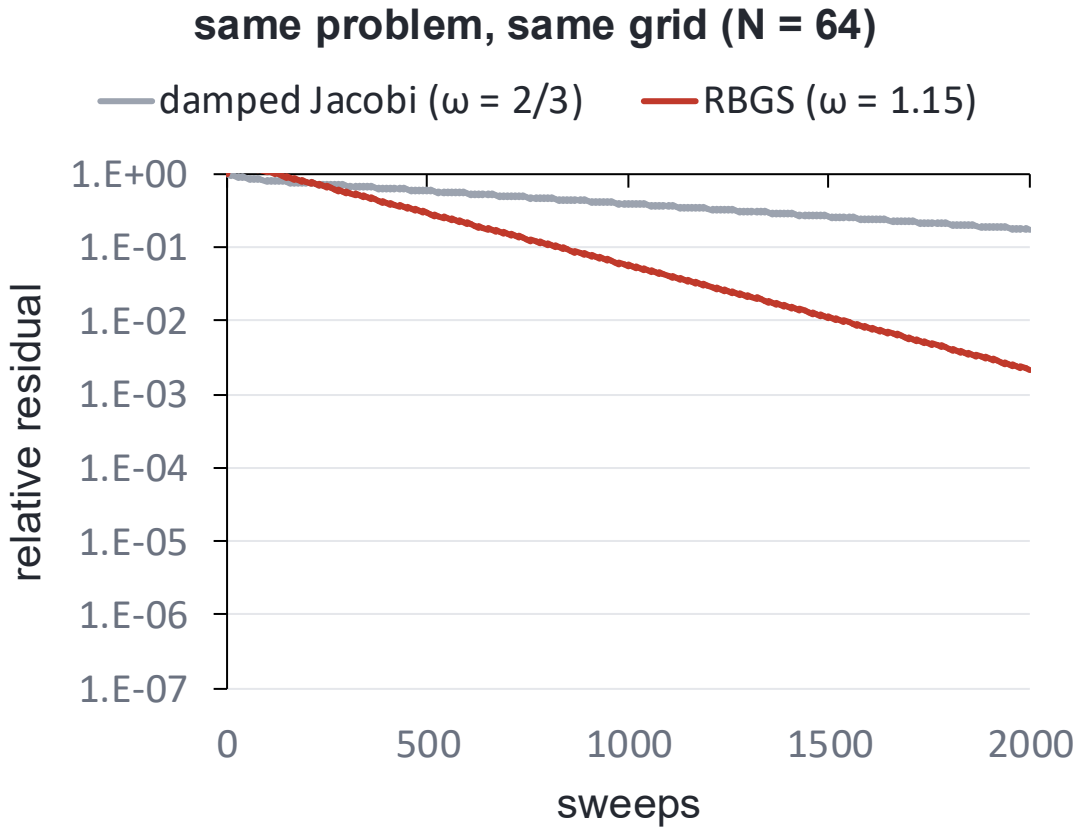
The smoother Athena uses: red–black Gauss–Seidel

S: $\Phi_{ijk} \leftarrow (1-\omega)\cdot\Phi_{ijk} + (\omega/6)\cdot(\,\Sigma \text{ six neighbours} - h^2 f_{ijk}\,)$ · red half-sweep first, then black · $\omega = 1.15$



each half-sweep touches only one colour — no data races, one kernel launch per colour

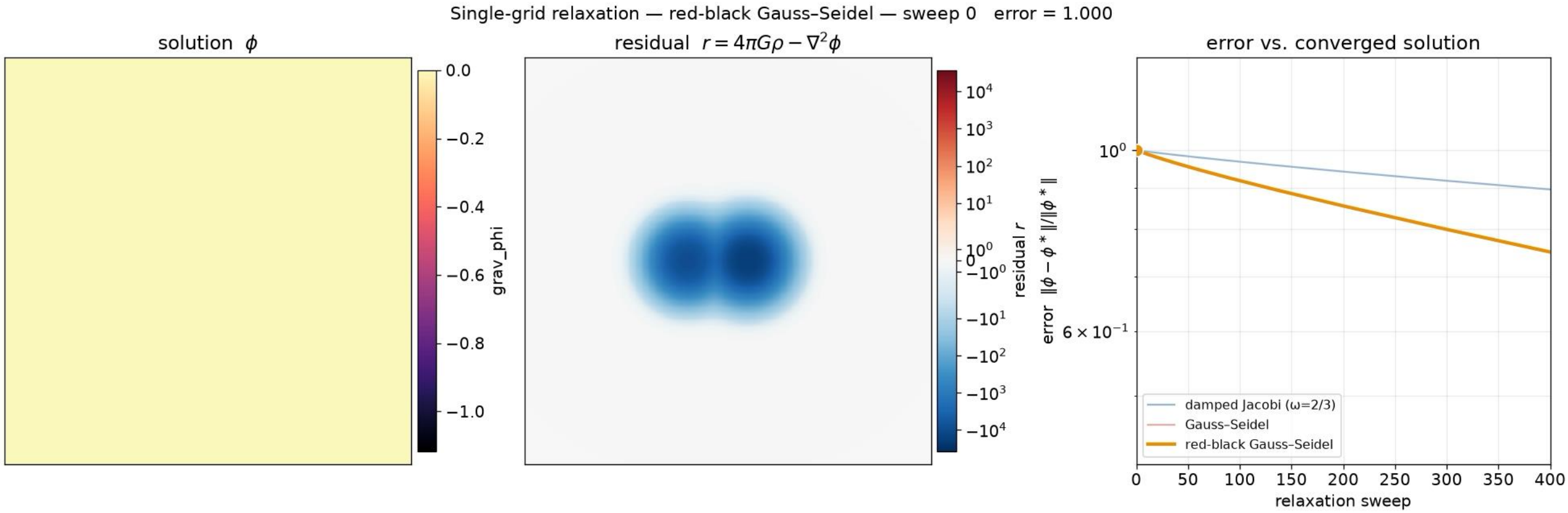
ω = 1.15 over-relaxation: tuned to damp high-frequency error fastest (Yavneh 1996)



Colour the cells like a checkerboard: every red cell's neighbours are black, so each half-sweep updates its cells independently — perfectly parallel, ideal for. With over-relaxation ω = 1.15 it damps rough error about twice as fast as damped Jacobi.

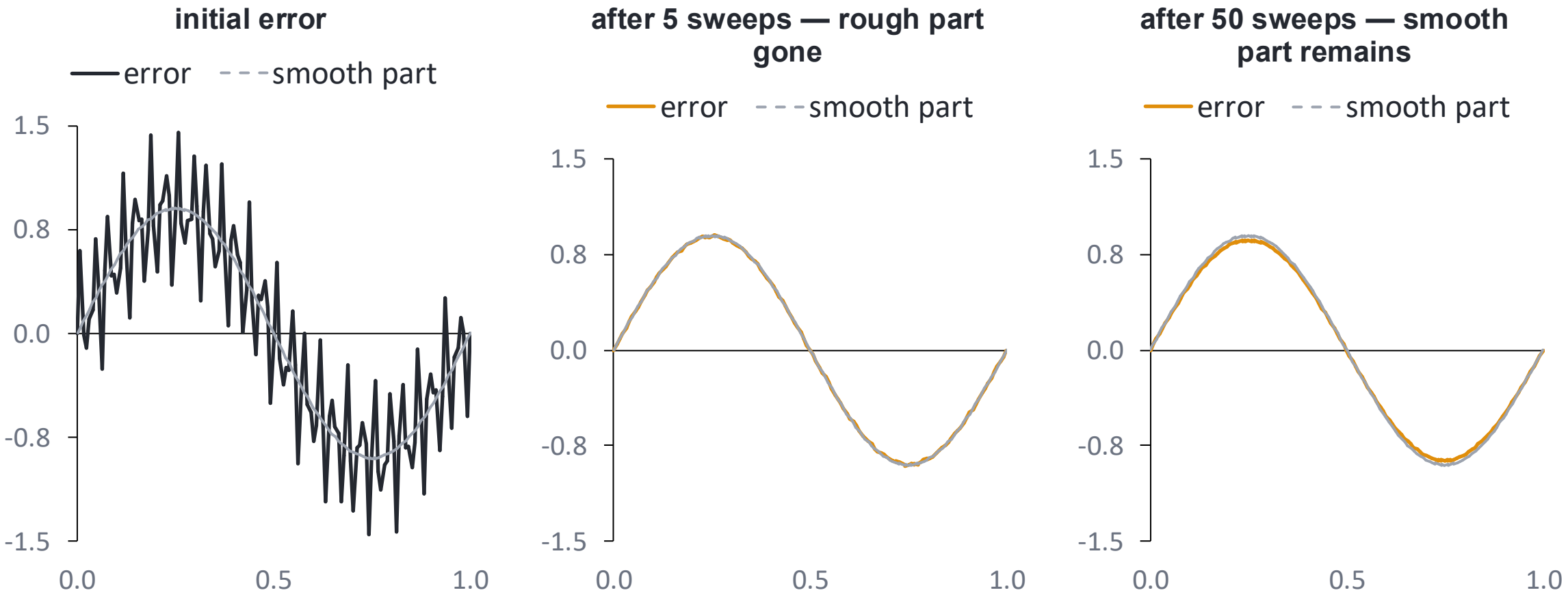
The smoother Athena uses: red–black Gauss–Seidel

S: $\Phi_{ijk} \leftarrow (1-\omega)\cdot\Phi_{ijk} + (\omega/6)\cdot(\,\Sigma \text{ six neighbours} - h^2 f_{ijk}\,)$ · red half-sweep first, then black · $\omega = 1.15$



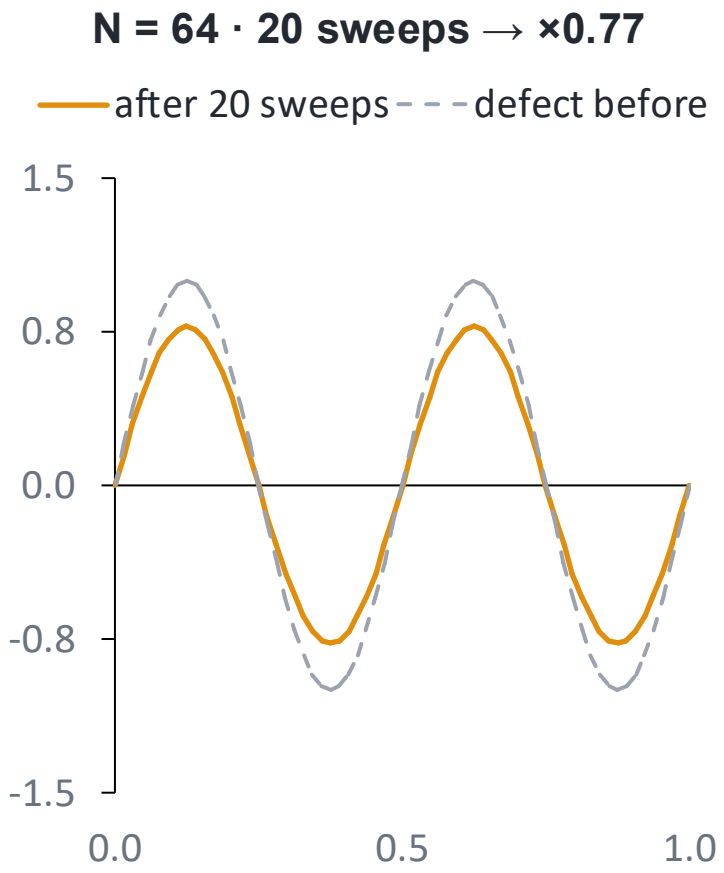
Colour the cells like a checkerboard: every red cell's neighbours are black, so each half-sweep updates its cells independently — perfectly parallel, ideal for. With over-relaxation ω = 1.15 it damps rough error about twice as fast as damped Jacobi.

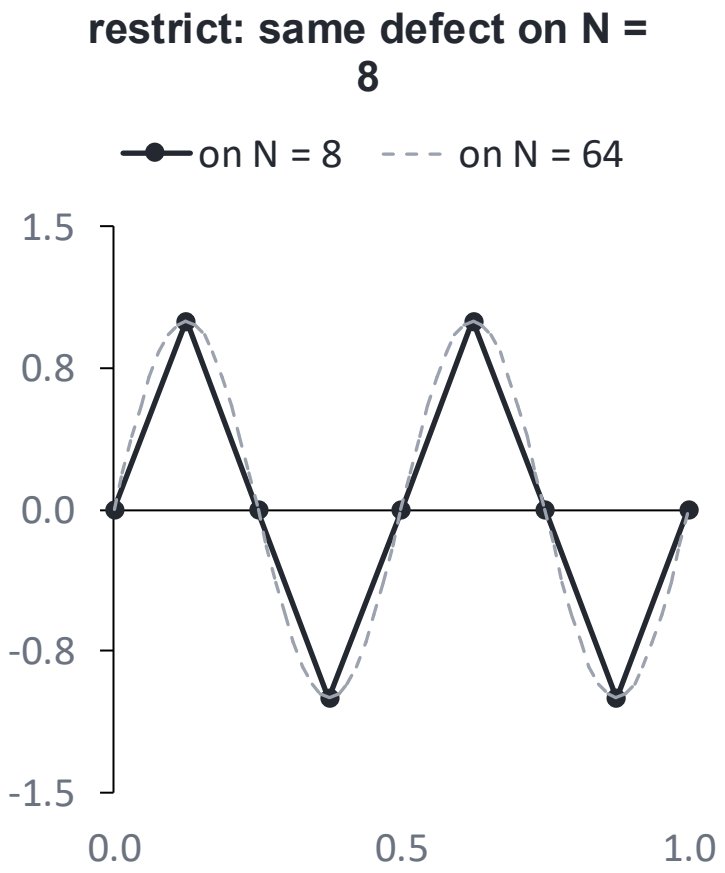
Relaxation is local — it only kills rough error

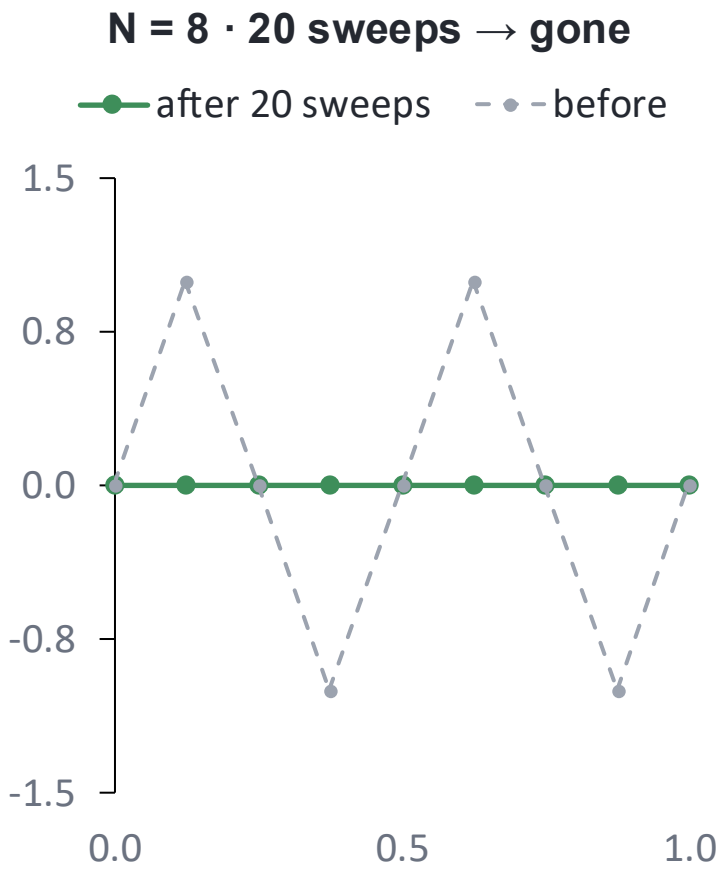


One damped-Jacobi sweep only mixes neighbouring cells: wavelengths of a few cells vanish within ~5 sweeps, while the smooth component (dashed) is barely touched — exactly why convergence stalls.

Too smooth to relax? Restrict the defect

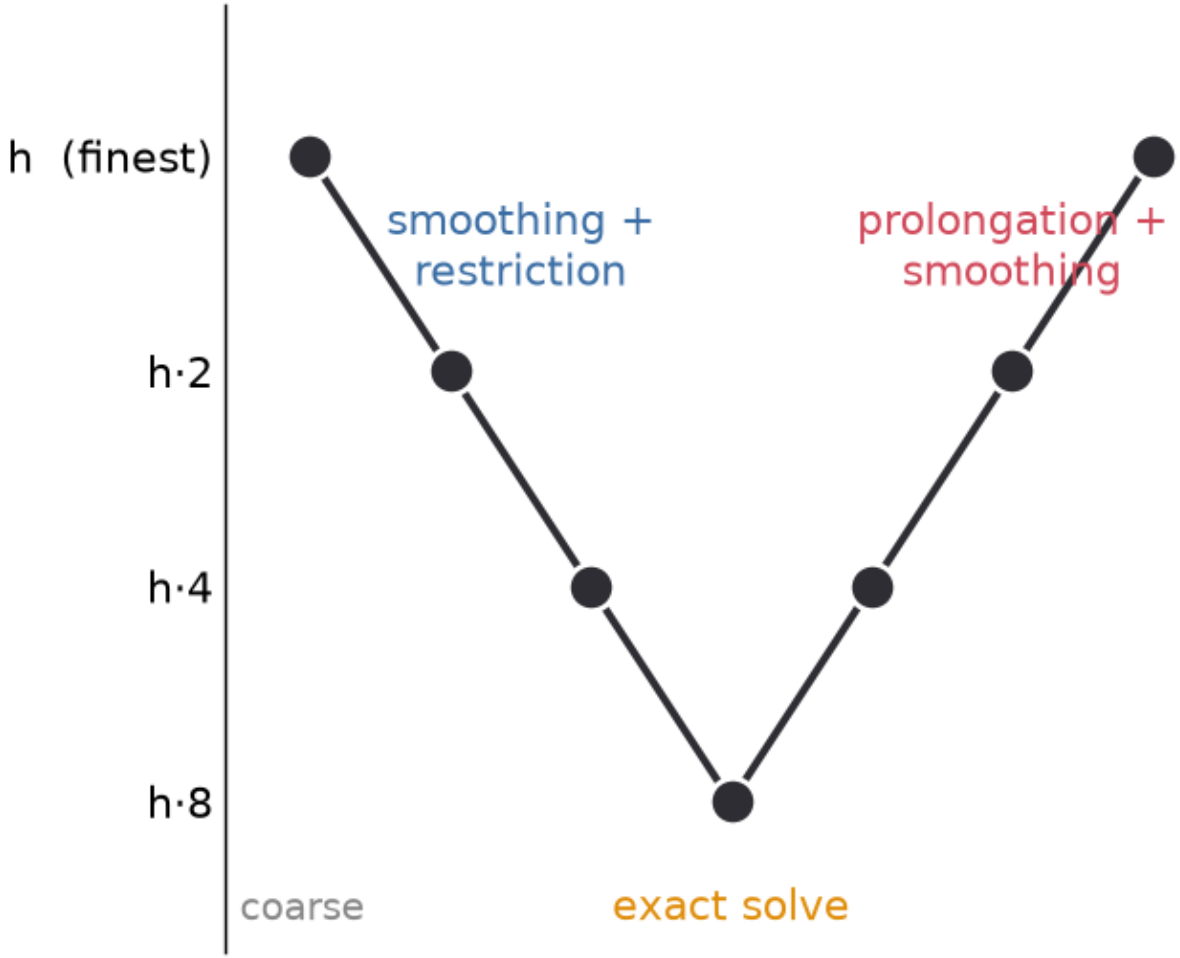






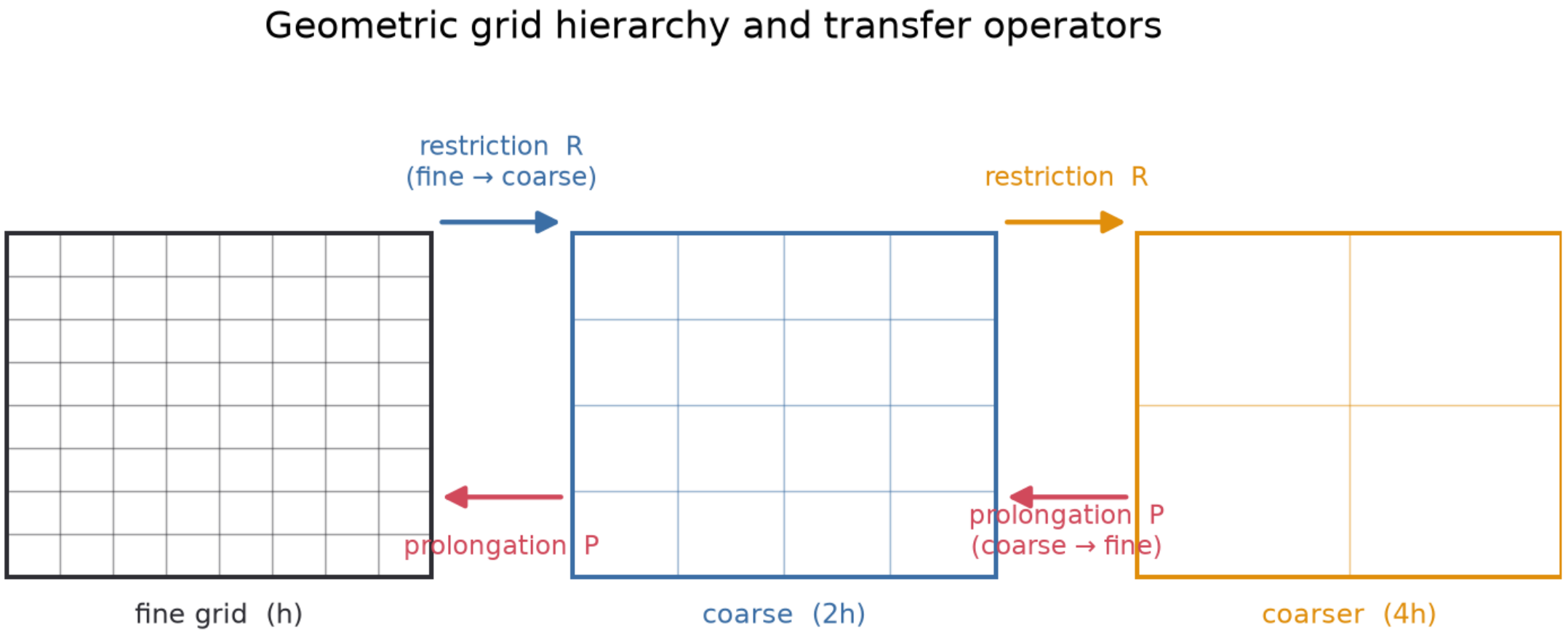
The defect $d = f - L\Phi$ carries the surviving low-frequency error. Restricted to a coarse grid, the same mode is rough relative to the grid: 20 sweeps that achieved nothing at $N = 64$ wipe it out at $N = 8$ — and each coarse sweep costs a fraction of a fine one.

Recursion over resolution: V-cycle



Smooth, restrict the residual to a coarser grid, solve the coarse problem recursively, prolong the correction back, smooth again.

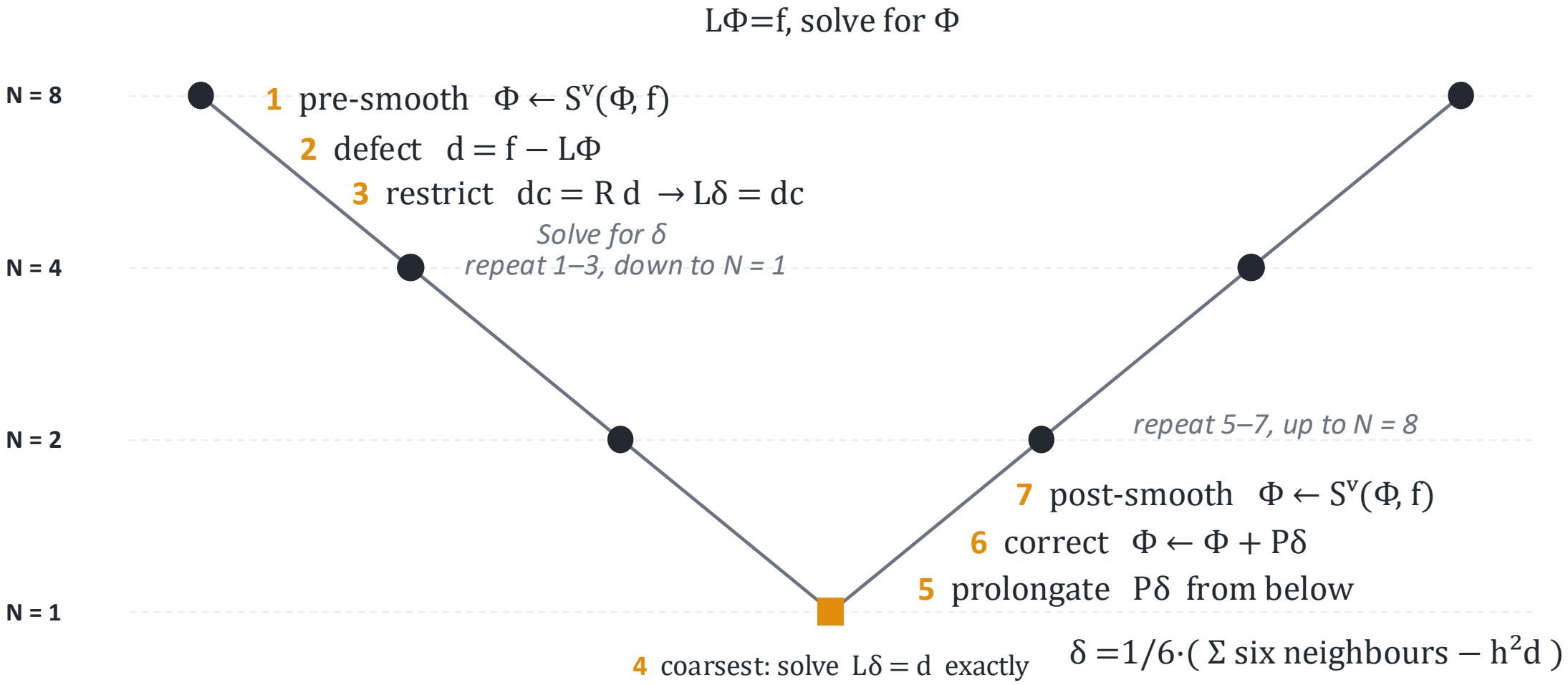
Transfer operators: restriction and prolongation



Restriction R (full-weighting) carries residuals to coarser grids; prolongation P (bilinear interpolation) carries corrections back to finer grids.

THE ALGORITHM

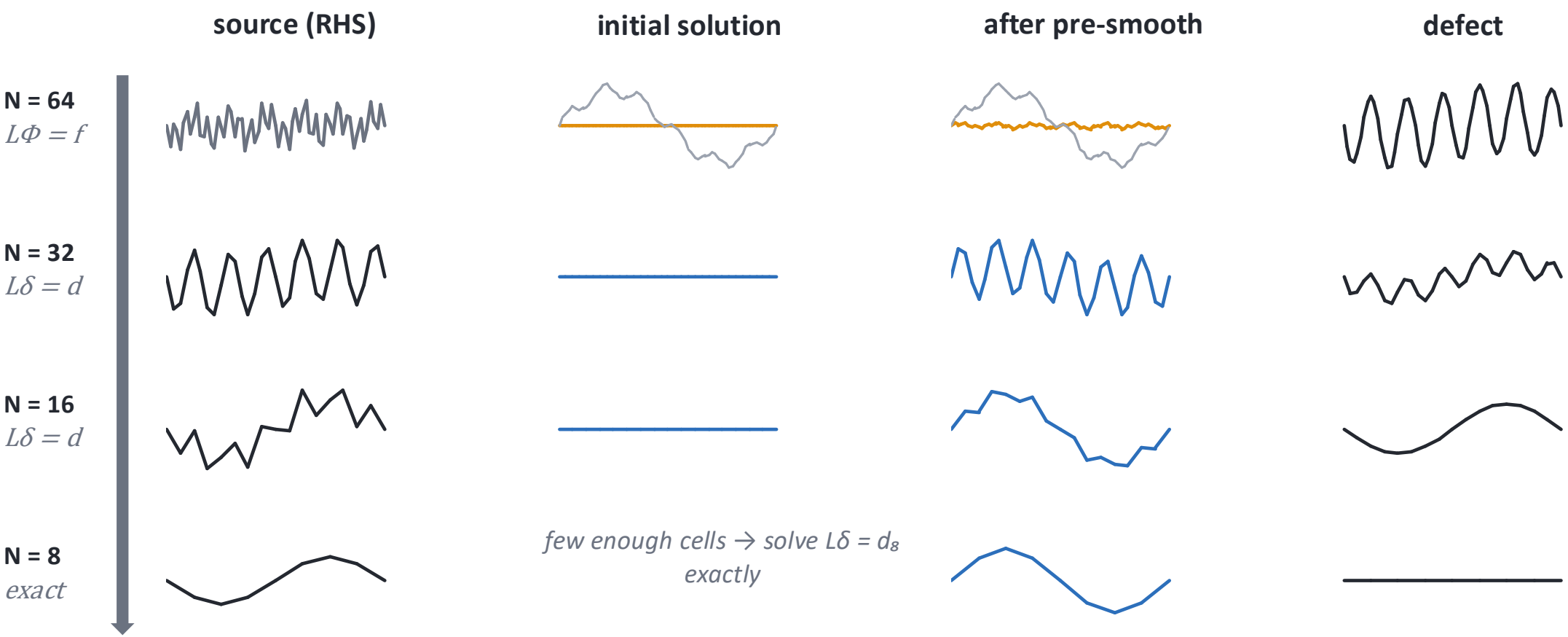
The V-cycle, operator by operator



S = RBGS smoother (ν sweeps), L = Laplacian, R = restriction, P = prolongation. S kills only rough error — which is why the pre-smoothed Φ on the next slide looks half-finished; the defect carries the smooth remainder to the coarser grids, and post-smoothing cleans the ripples P introduces.

Going down: the defect becomes the next source

pre-smooth from zero · d = RHS − L·solution · restrict d → next RHS

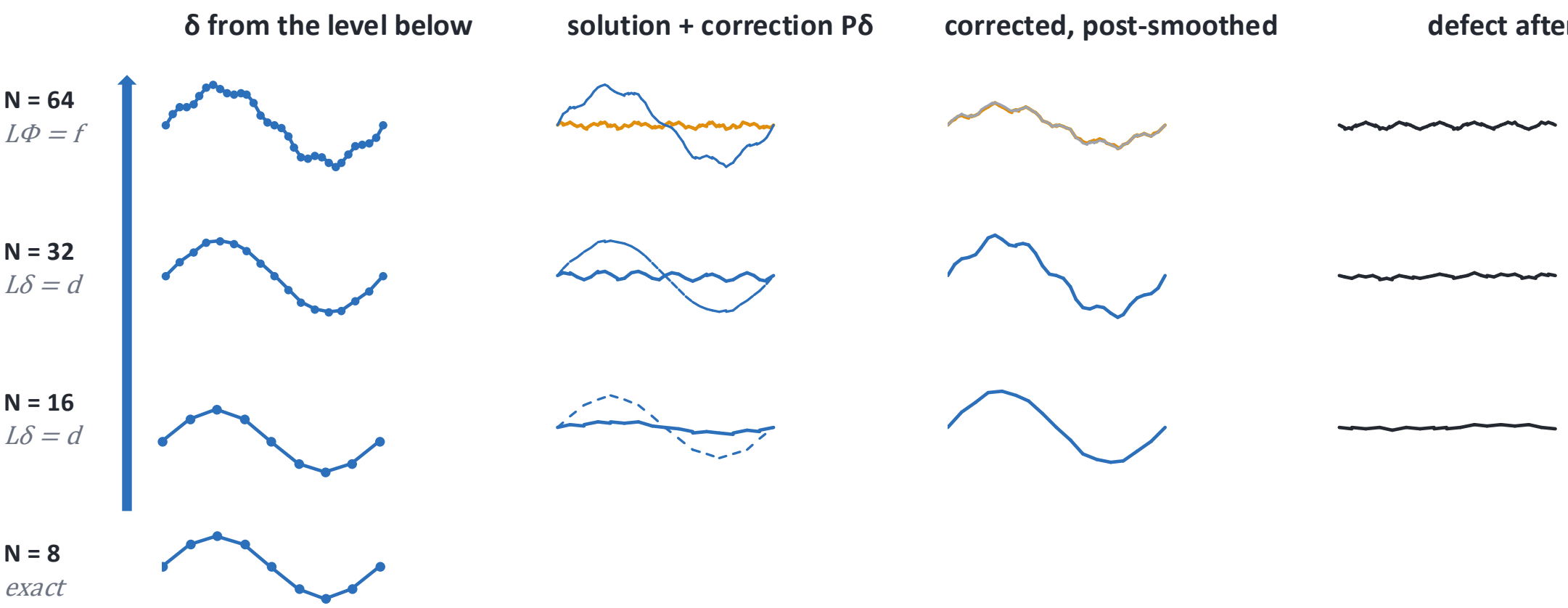


Descending the V on LΦ = f: every level starts from zero (second column), smooths a few sweeps, and hands its remaining defect down as the next level's right-hand side — until N = 8, where the exact solve leaves (almost) no defect. Dashed gray in the top row: the exact solution Φ, the target we are converging to.*

THE ALGORITHM

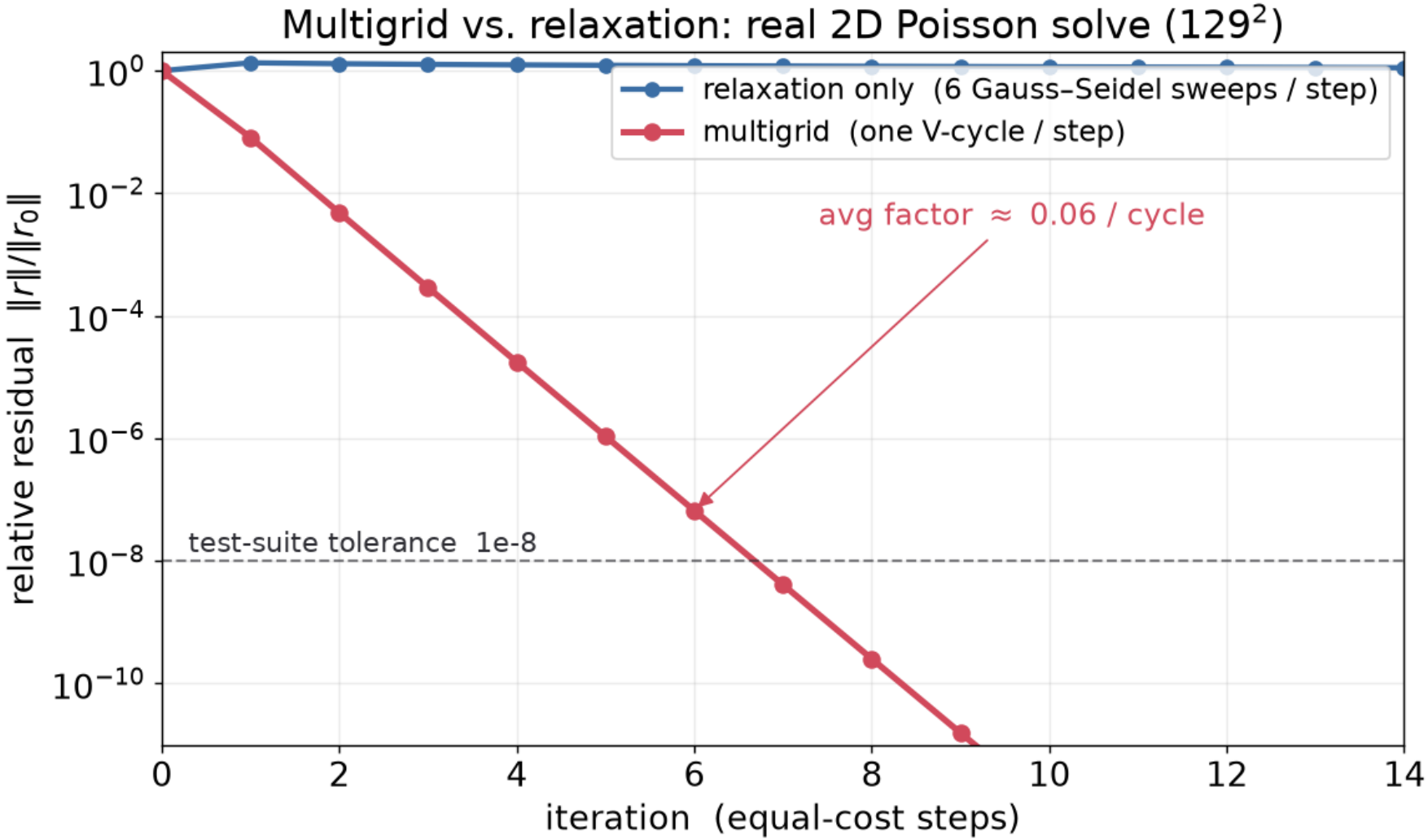
Going up: corrections repair the solution

$$\delta = \text{coarse solution of } L\delta = d \;\longrightarrow\; \Phi \leftarrow \Phi + P\delta \;\longrightarrow\; \text{post-smooth}$$



The correction is the coarser level's own solution of Lδ = d (first column, coarse points), prolongated and overlaid on this level's solution (second column, dashed blue over solid), added, and post-smoothed. Dashed gray in the top row: the exact solution Φ — after the cycle, Φ lands on it and the defect has dropped ×15.*

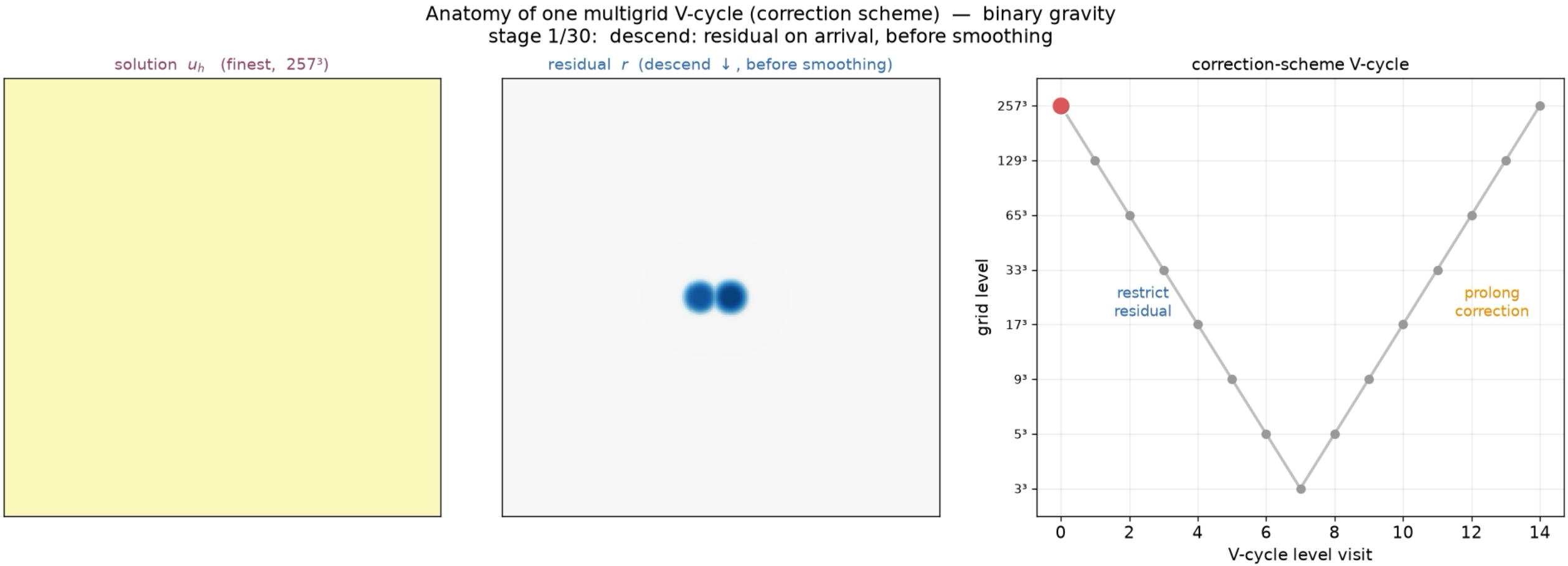
The payoff: grid-independent convergence



A real 2D Poisson solve. Multigrid contracts the residual by ~0.06 per cycle to machine precision; relaxation alone is flat on this scale.

APPLICATIONS

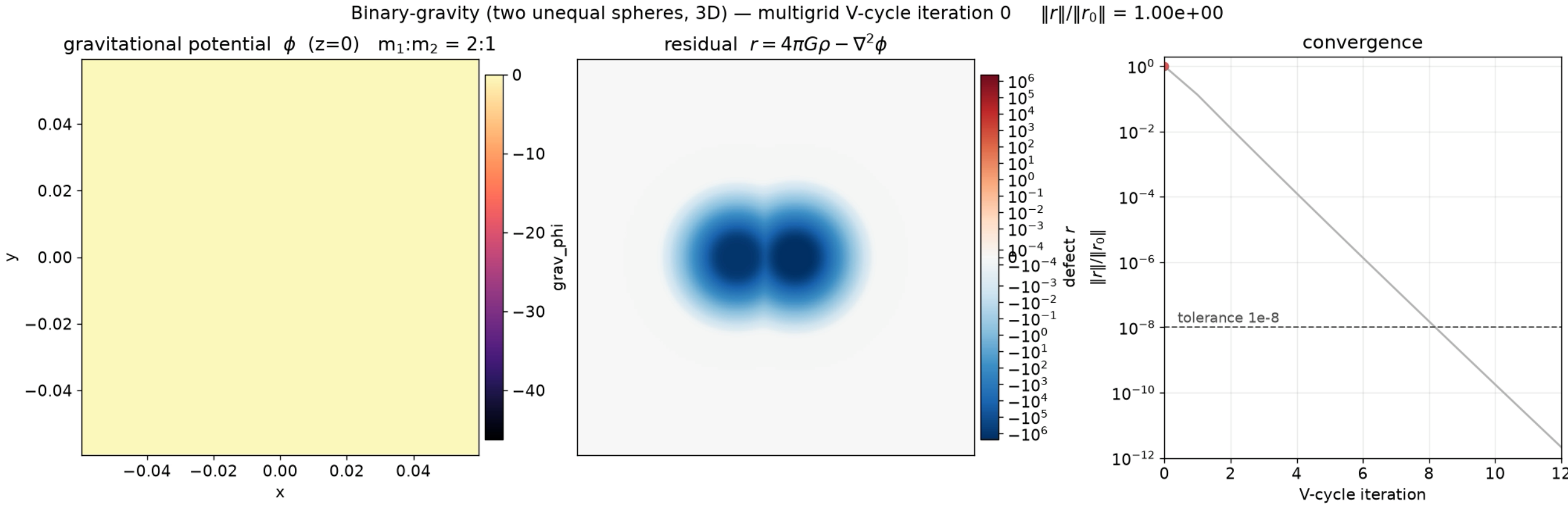
Binary self-gravity with static mesh refinement



The potential of two point masses resolved with an SMR patch — the fine grid captures the steep central wells while the coarse grid covers the far field.

APPLICATIONS

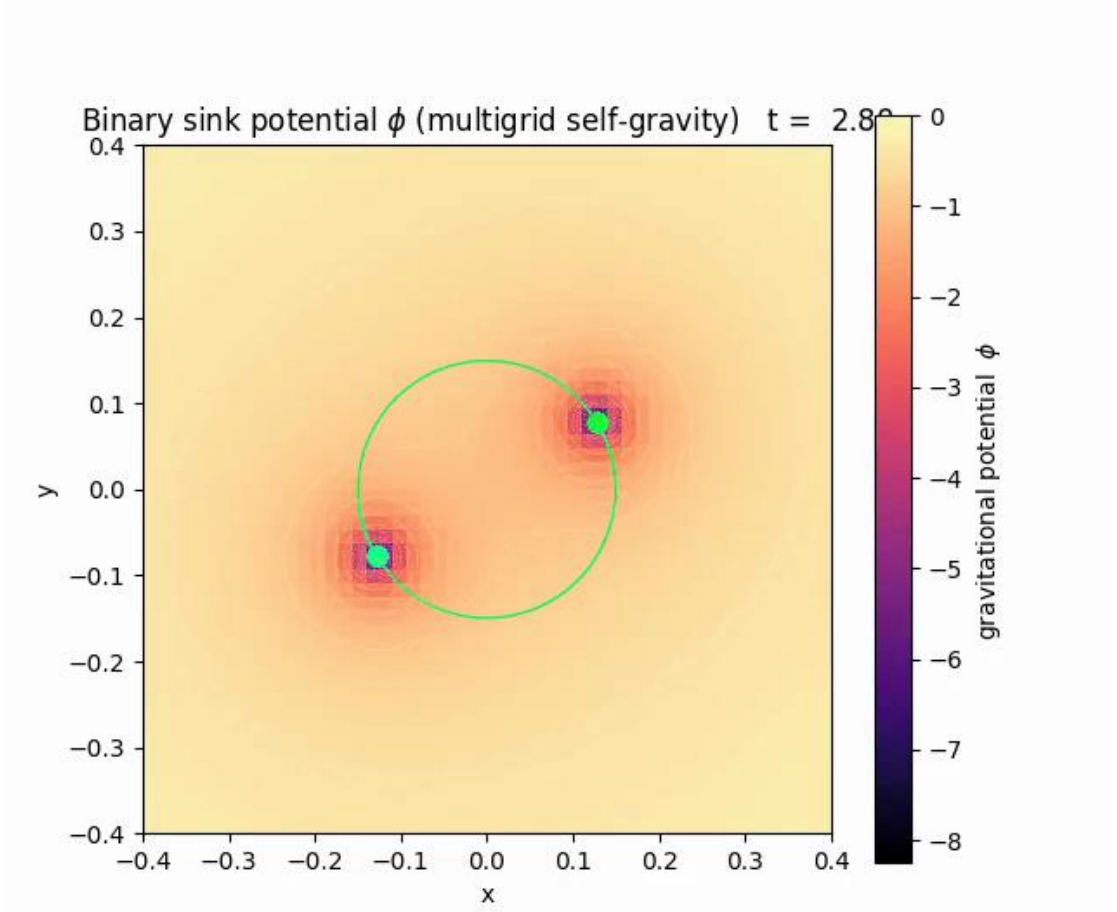
Binary self-gravity with static mesh refinement



The potential of two point masses resolved with an SMR patch — the fine grid captures the steep central wells while the coarse grid covers the far field.

APPLICATIONS

A binary orbit, evolved self-consistently



Gravitational potential over a full binary orbit; the self-gravity solver is called every step and feeds back on the gas dynamics.

THE ALGORITHM

Two flavours: correction scheme vs FAS

Correction scheme — vanilla multigrid

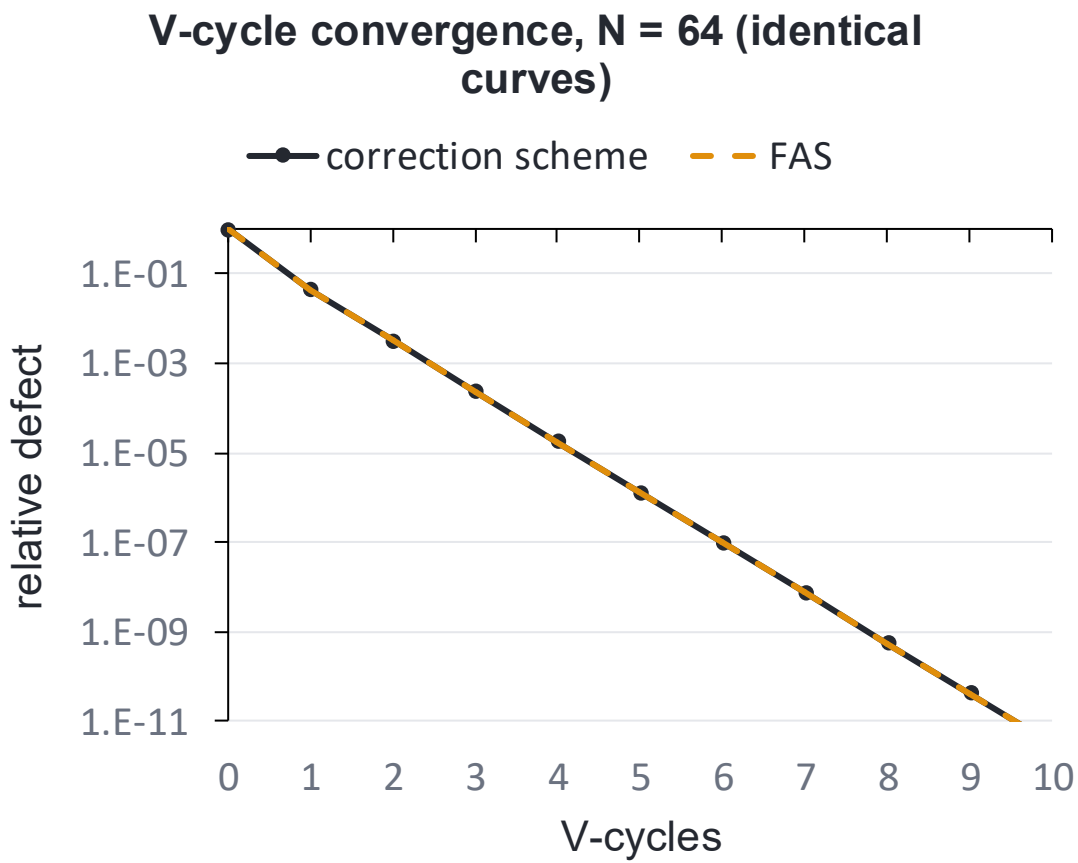
coarse levels solve $L\delta = d$, starting from $\delta = 0$

Coarse levels see only the error: restrict the defect, solve for the correction, prolongate and add it. Everything shown so far used this. Perfect for linear problems on uniform grids — minimal storage.

Full Approximation Scheme (FAS)

coarse levels solve $L\Phi c = R\,d + L(R\,\Phi)$;
correction $= \Phi c - R\Phi$

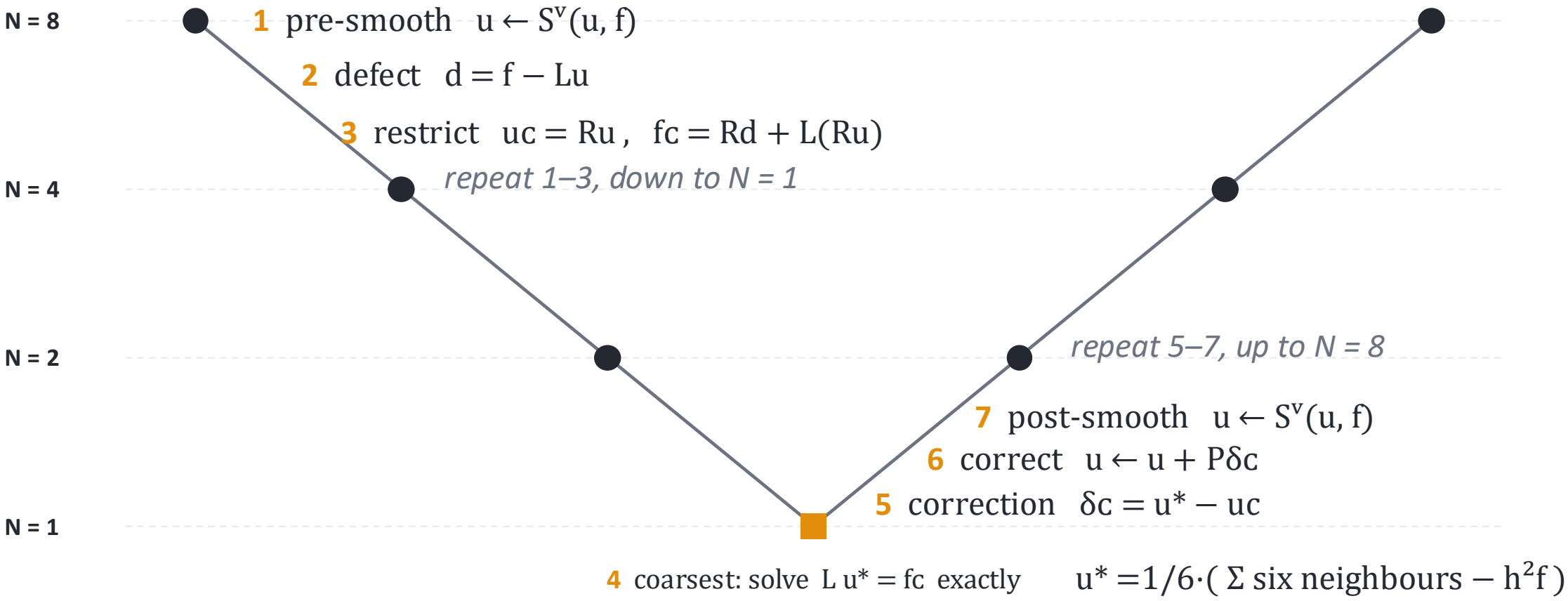
Coarse levels carry the full solution, not just the error: the restricted solution enters the right-hand side (the τ-correction). This stays consistent when the operator is nonlinear.



For a linear problem the two schemes produce identical iterates — FAS buys generality (nonlinear operators, consistent AMR level boundaries), not speed. Enabling FAS in Athena costs no extra V-cycles, only the extra storage for the restricted solution.

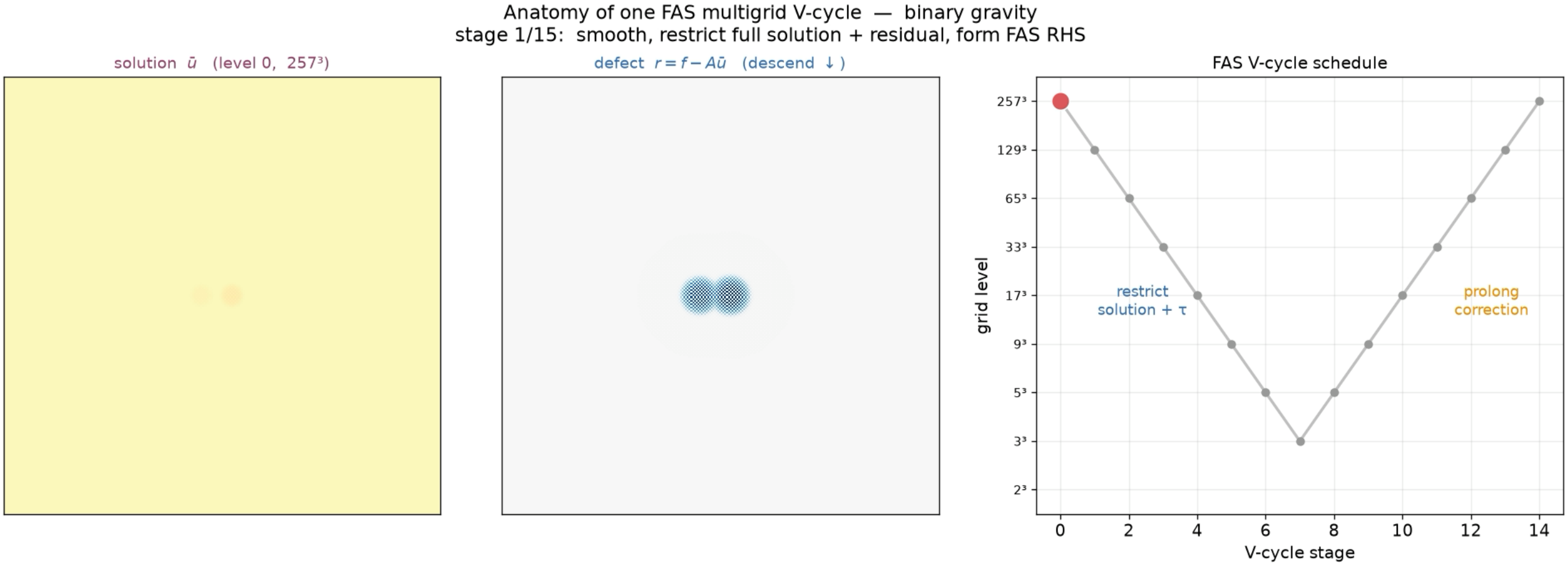
THE ALGORITHM

FAS, operator by operator



Same V, different bookkeeping: coarse levels carry the full solution uc (initialized as Ru) with the τ-correction hidden in fc = Rd + L(Ru); the correction sent up is δc = u'c − Ru. For linear L this reproduces the correction scheme exactly — with AMR it keeps level boundaries consistent.

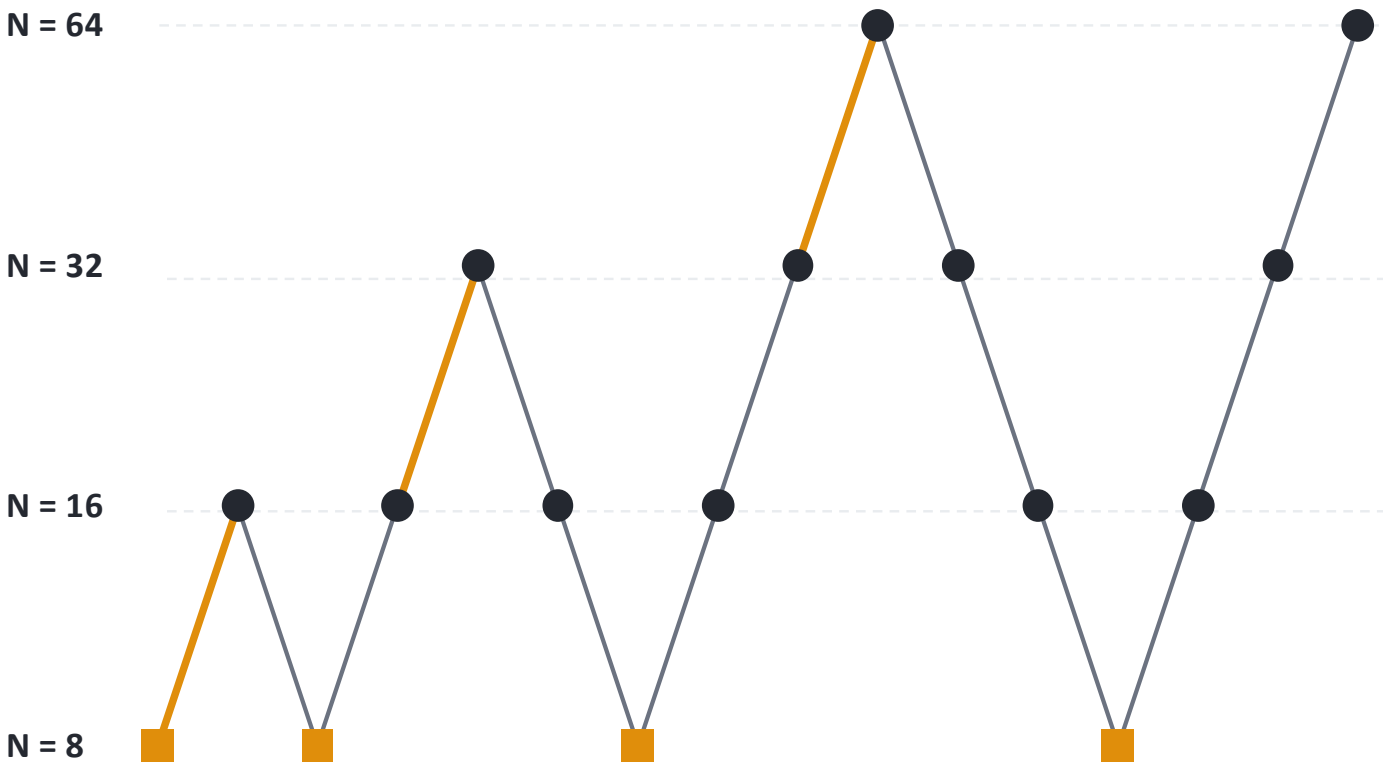
FAS, operator by operator



Same V, different bookkeeping: coarse levels carry the full solution uc (initialized as Ru) with the τ-correction hidden in fc = Rd + L(Ru); the correction sent up is δc = u'c − Ru. For linear L this reproduces the correction scheme exactly — with AMR it keeps level boundaries consistent.

THE ALGORITHM

Full Multigrid, operator by operator



1 coarsest: solve $Lu = f$ exactly

2 seed: $u \leftarrow \Pi\, uc$ (tricubic Π)

3 one V-cycle at this level

the inner V-cycles can use either the correction scheme or FAS — the schedule is identical

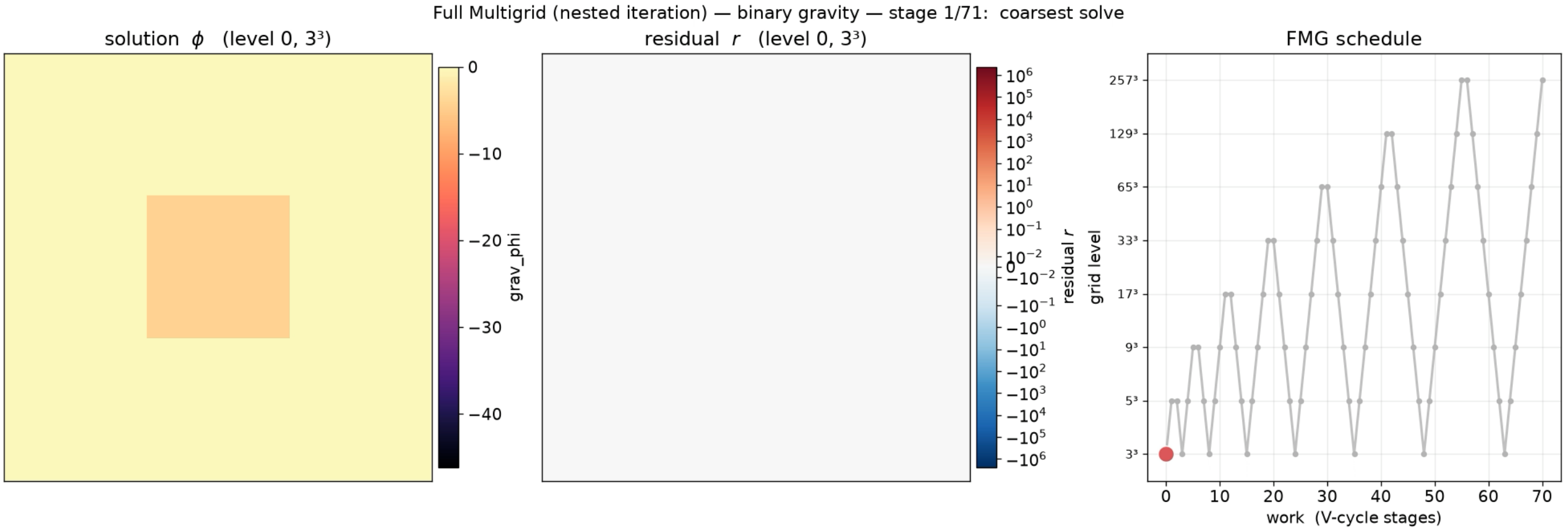
repeat 2–3 up to the finest level

one pass ≈ 1.5 V-cycles of work

■ *exact coarsest solve* ● *smoothing* — *amber: Π (FMG prolongation)*

FMG is a wrapper around the V-cycle: solve the coarsest level, Π-prolongate the solution up as the initial guess, run one V-cycle, repeat to the finest level. Π is higher-order (tricubic) than P because a good first guess needs accurate interpolation. Athena runs FMG when no previous potential exists.

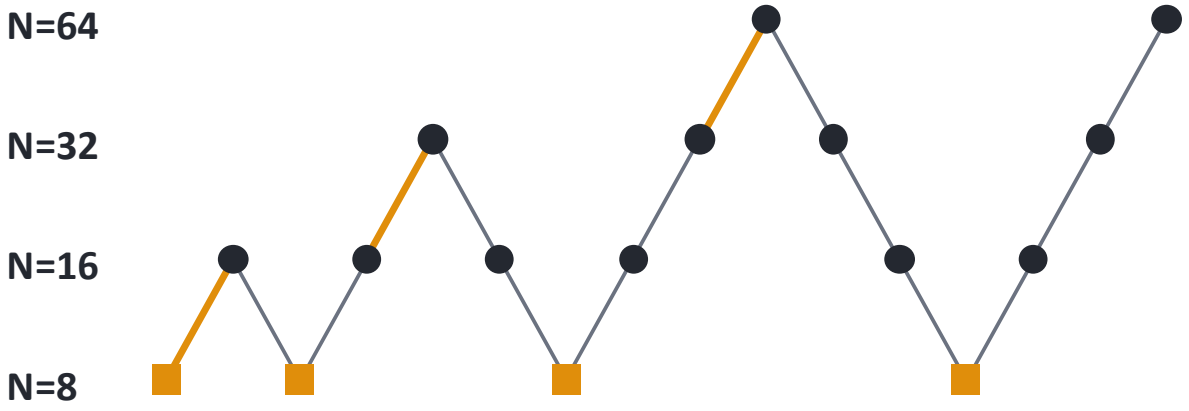
Full Multigrid, operator by operator



FMG is a wrapper around the V-cycle: solve the coarsest level, Π-prolongate the solution up as the initial guess, run one V-cycle, repeat to the finest level. Π is higher-order (tricubic) than P because a good first guess needs accurate interpolation. Athena runs FMG when no previous potential exists.

THE ALGORITHM

Full Multigrid: start coarse, arrive converged

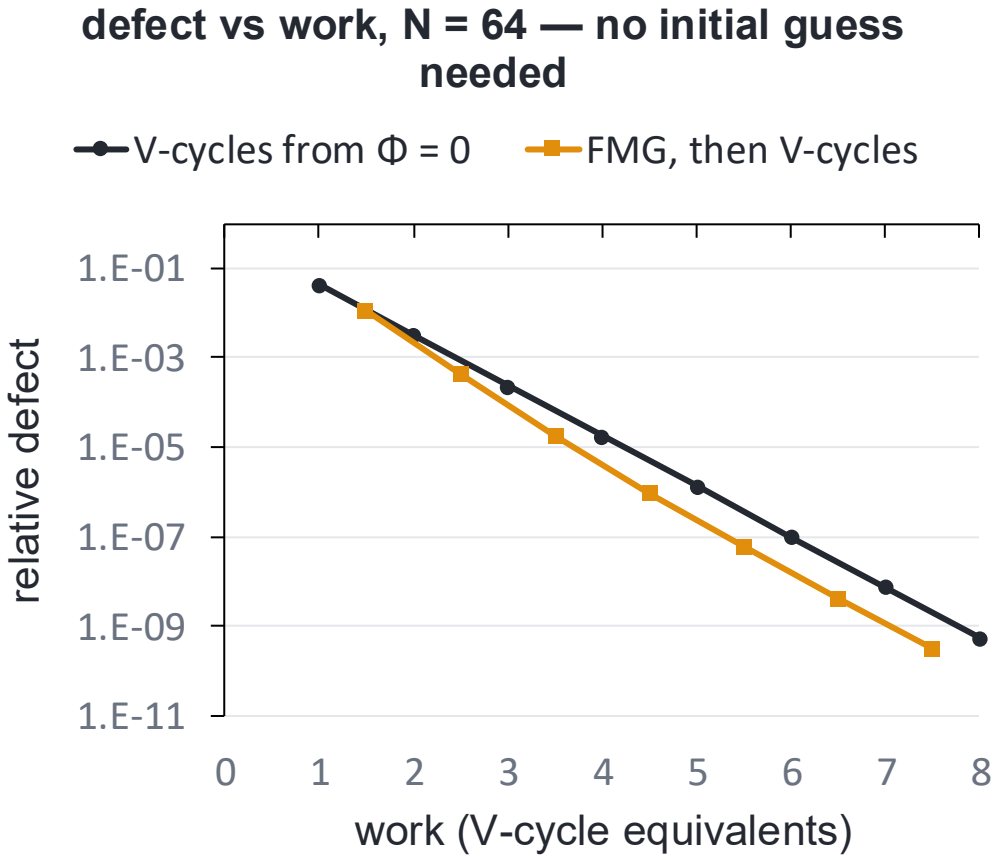


■ *exact coarsest solve* ● *smoothing* — *thick amber: FMG prolongation (tricubic)*

1 · solve the coarsest level — nearly free

2 · FMG-prolongate the solution one level up as the initial guess

3 · run one V-cycle there, then repeat — total cost ≈ 1.5 V-cycles



FMG builds the solution from the coarsest level up, seeding each finer level with a higher-order (tricubic) prolongation, plus one V-cycle. One FMG pass already reaches the truncation-error level; Athena uses FMG when no previous potential exists, then per-timestep V-cycles seeded by the last solution.

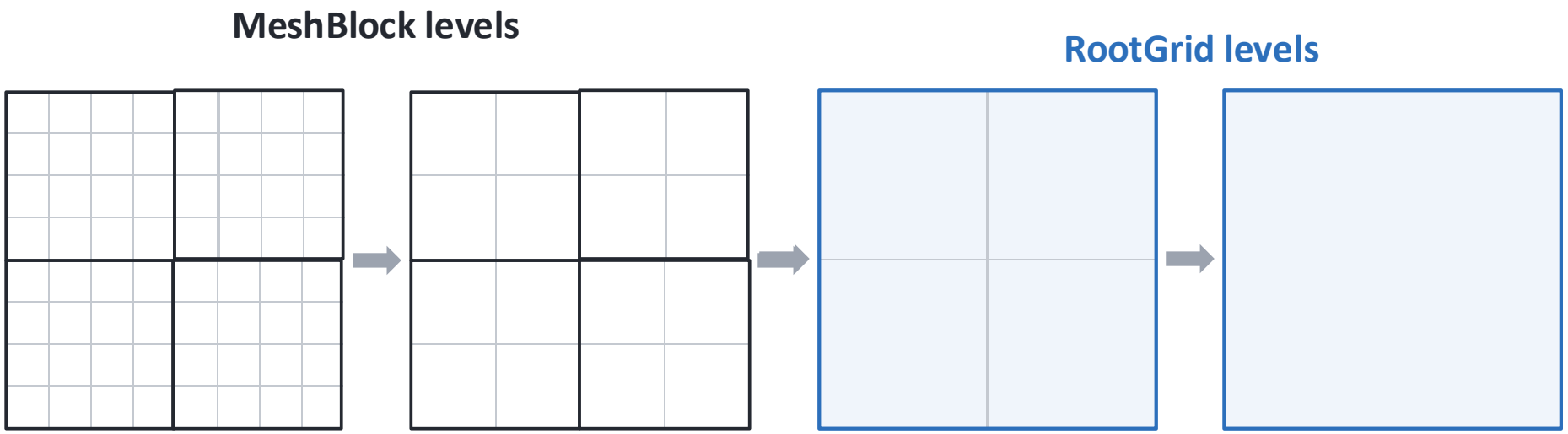
MOTIVATION

What AthenaK needs — and why multigrid fits

Block-structured octree AMR (MeshBlocks)	➡	multigrid is built on the very same blocks: MeshBlock, Octet and RootGrid levels → ***GRIDS & LEVELS***
GPUs via Kokkos — wide, local parallelism	➡	RBGS half-sweeps are one kernel launch per colour, with no data races → ***the smoother***
Thousands of MPI ranks	➡	nearest-neighbour halos, one small gather at the root, rank-packed messages → ***PERFORMANCE***
Isolated systems and mixed boundaries	➡	periodic, fixed, zero-gradient and multipole-expansion boundary conditions
A Poisson solve every single timestep	➡	grid-independent convergence; the last Φ seeds the next V-cycles, FMG for cold starts → ***FMG***

A small spoiler of the rest of the talk: every requirement on the left maps onto a multigrid property on the right — O(N) cost, block-local operations, and consistency across refinement levels.

The hierarchy, in 2D: blocks → root grid



→ restriction carries defects toward level 0 · ← prolongation carries corrections back to the finest level

GRIDS & LEVELS

Two kinds of levels

MeshBlock levels

levels m+1 … m+n — the finest

Restriction and prolongation run inside each block — hence 2^n cells per side. Blocks are smoothed and coarsened independently, in parallel, down to a single cell.

RootGrid levels

levels 0 … m — the coarsest

Once every block is one cell, all ranks gather (MPI_Allgather) into one global RootGrid covering the whole domain, coarsened down to level 0 — where the solve is exact.

Uniform grid: $L = m + n$ levels, $N_l = 2^l N_0$ cells per side

GRIDS & LEVELS

Three kinds of levels

MeshBlock levels

levels m+1 … m+n — the finest

Restriction and prolongation run inside each block — hence 2^n cells per side. Blocks are smoothed and coarsened independently, in parallel, down to a single cell.

Octet levels

inserted by AMR refinement

A refined block's 2×2×2 children collapse onto the parent's footprint as an Octet of 2×2×2 cells. Smoothing runs per Octet; coarse-facing sides use values prolongated from below.

RootGrid levels

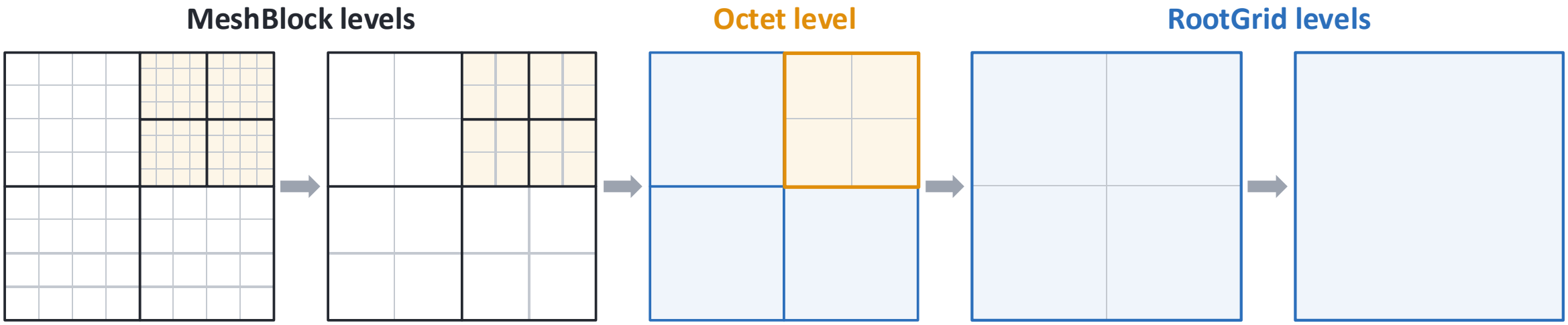
levels 0 … m — the coarsest

Once every block is one cell, all ranks gather (MPI_Allgather) into one global RootGrid covering the whole domain, coarsened down to level 0 — where the solve is exact.

Uniform grid: $L = m + n$ levels, $N_l = 2^l N_0$ cells per side · Octet levels appear only where the mesh is refined

Terminology of Tomida & Stone (2023): one Multigrid object per MeshBlock, Octets created by refinement, and a single global RootGrid — AthenaK's multigrid follows the same structure.

The full hierarchy, in 2D: blocks → Octet → root grid

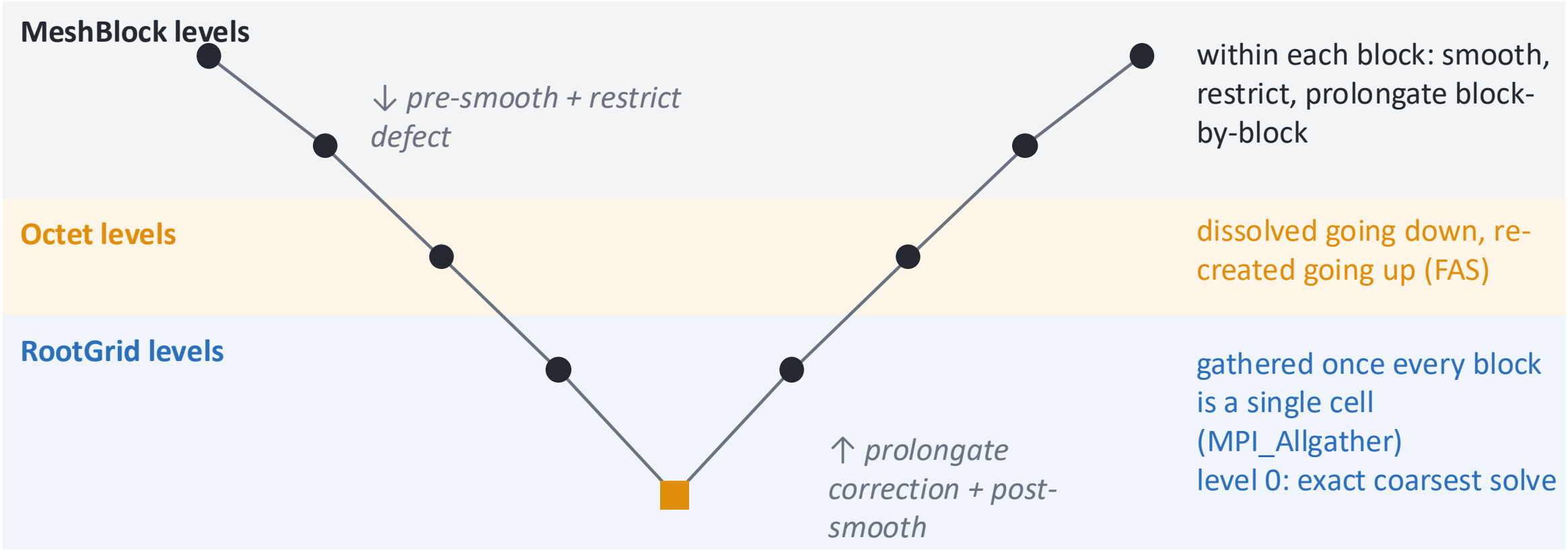


→ restriction carries defects toward level 0 · ← prolongation carries corrections back to the finest level

The paper's Figure 3 in miniature: a 2×2-block domain, top-right block refined once. Thick outlines are MeshBlocks / Octets / RootGrid tiles; thin lines are the cells inside each object.

GRIDS & LEVELS

One V-cycle walks every level

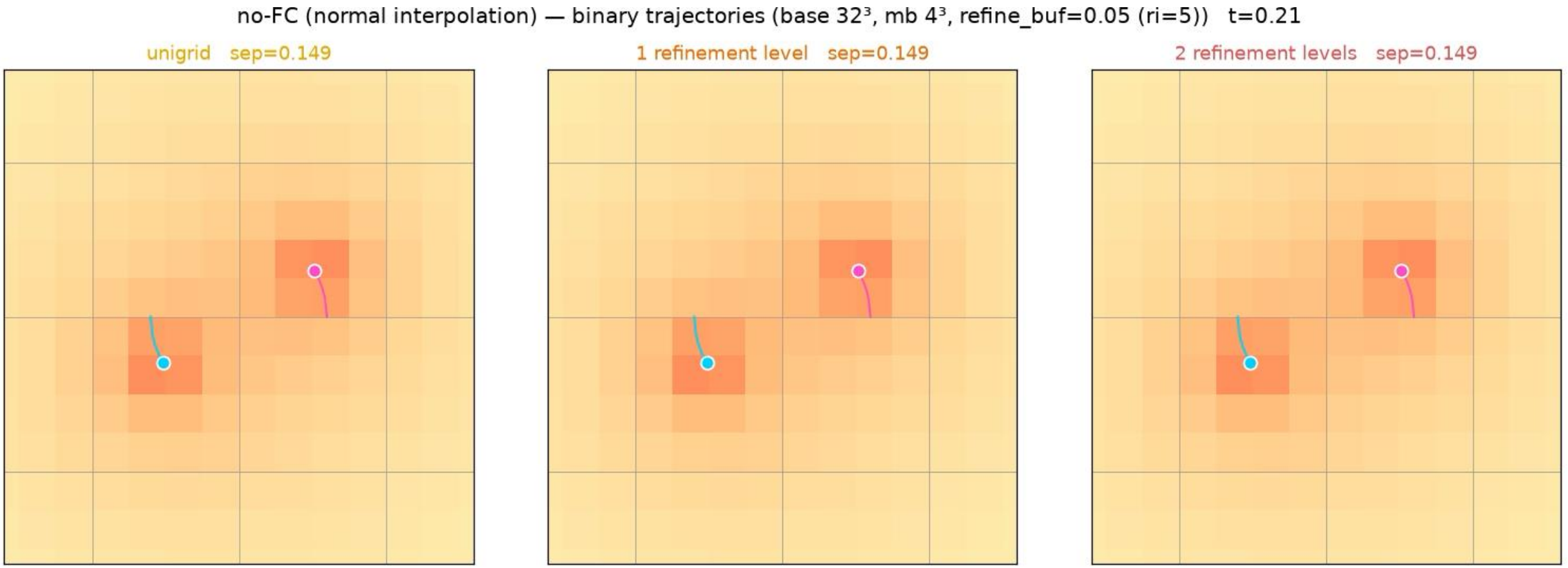


● smoothing (RBGS, ω = 1.15) ■ exact solve on the coarsest level

Going down the V, defects are smoothed and restricted; at level 0 the problem is solved exactly; going up, corrections are prolongated and post-smoothed. FMG stacks these V-cycles from coarse to fine.

APPLICATIONS

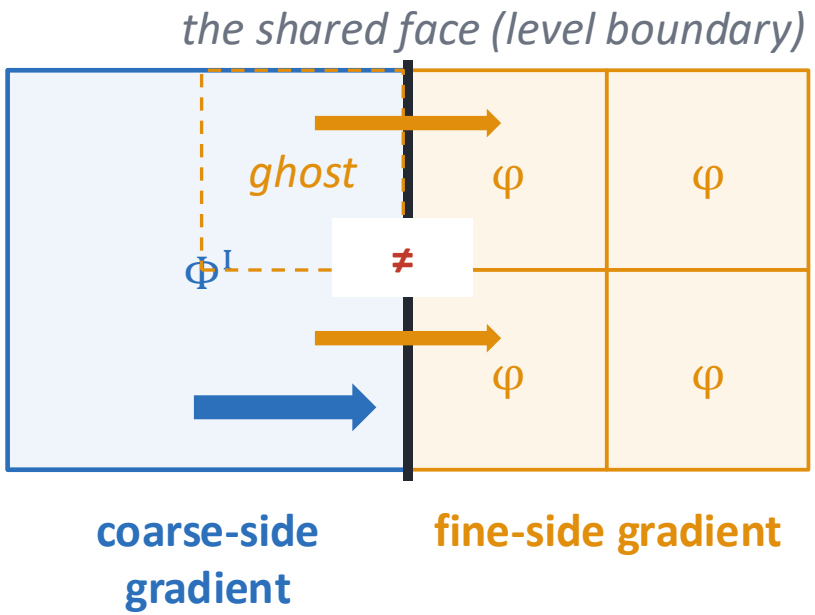
A binary orbit, evolved self-consistently



Gravitational potential over a full binary orbit; the self-gravity solver is called every step and feeds back on the gas dynamics.

Level boundaries, the simple way — and its flaw

$$\text{interpolated ghosts} \longrightarrow \partial\Phi/\partial x \text{ (coarse face)} \neq \tfrac{1}{4}\,\Sigma\,\partial\varphi/\partial x \text{ (fine faces)} \longrightarrow \text{spurious surface mass}$$

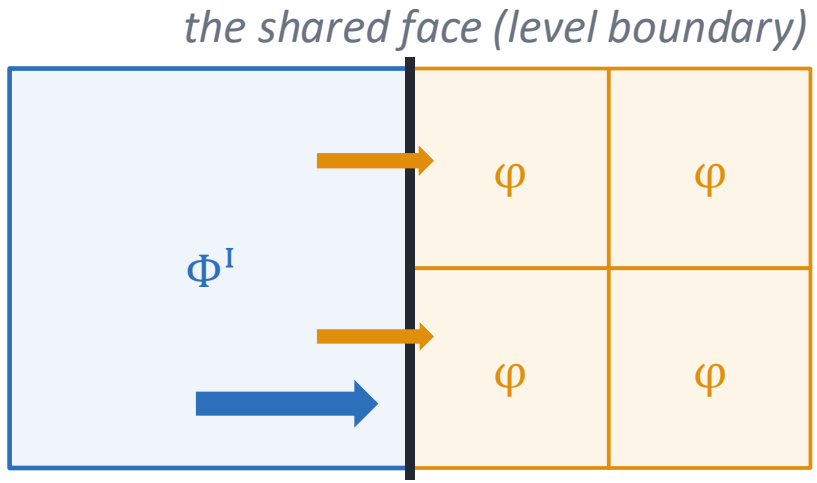


- ghost cells on each side are filled by simple (tri)linear interpolation from the neighbouring level
- the gradient at the shared face then differs between the fine side and the coarse side
- the mismatch behaves like fake mass sitting on the boundary → artificial accelerations on gas crossing it
- cheap — and fine for prolongation-stage communication, but not for smoothing or force calculation

The "normal" boundary: interpolate ghost values and difference as usual. Athena still uses it before prolongation operations, where errors get smoothed away — but forces computed this way kick the gas at every refinement boundary. The fix is on the next slide.

Level boundaries: flux-conserving prolongation

$\partial\Phi/\partial x$ (coarse face) $=$ ¼ Σ $\partial\varphi/\partial x$ (its four fine faces) — the gradient is single-valued across the boundary

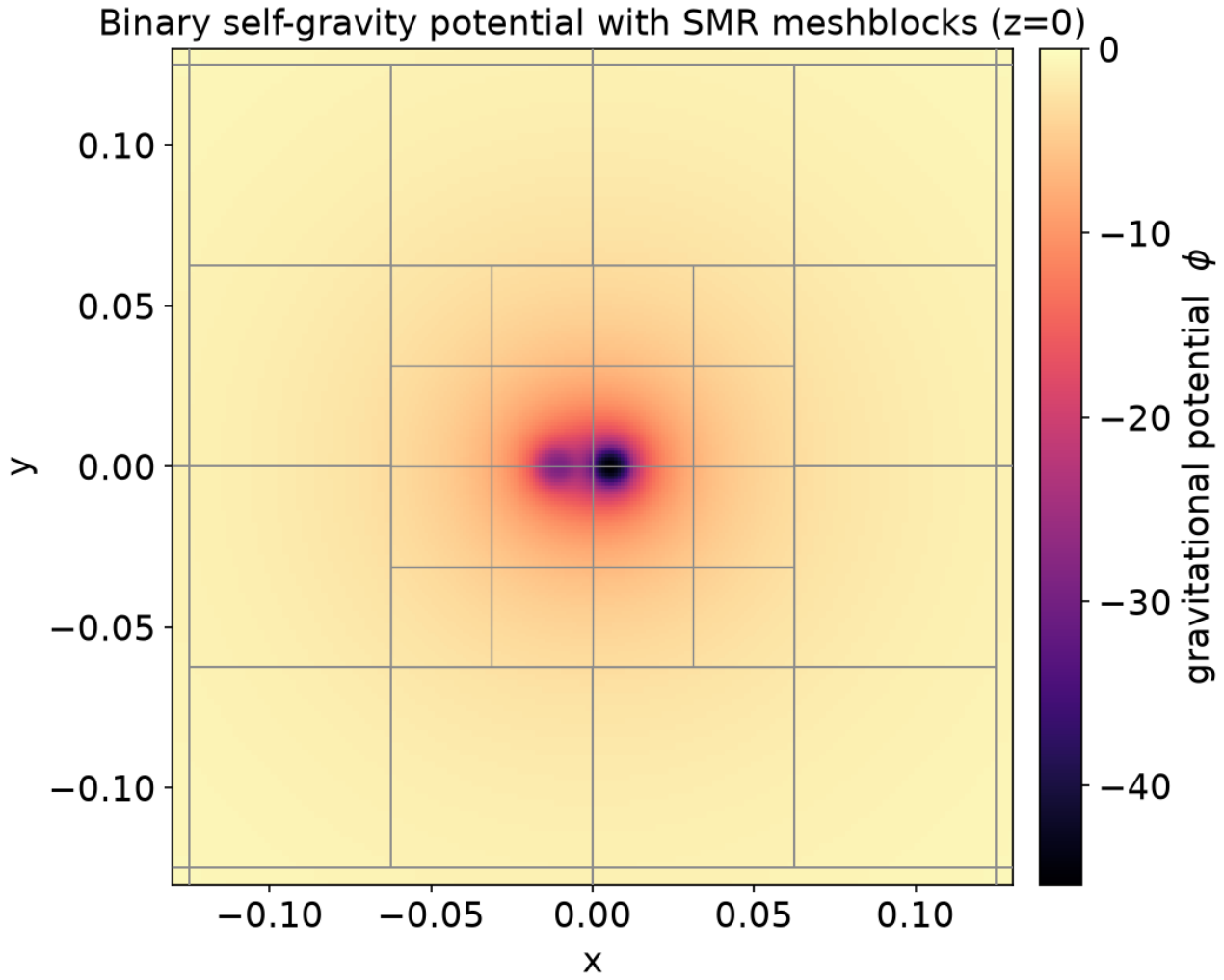


- gravity g = −∇Φ must be single-valued at every fine/coarse face
- ghost cells at the boundary are built so the shared-face gradient matches exactly
- the truncation error stays divergence-free → no “fake mass”, no artificial acceleration at level boundaries
- conservative by construction — no extra flux-correction step, unlike the MHD integrator's AMR fluxes

Following Feng et al. (2018), as in Athena++ (Tomida & Stone 2023, Fig. 4) and AthenaK's flux-conserving prolongation. One ghost-cell layer suffices for the 7-point Laplacian even with AMR; a simpler boundary is used before prolongation operations.

APPLICATIONS

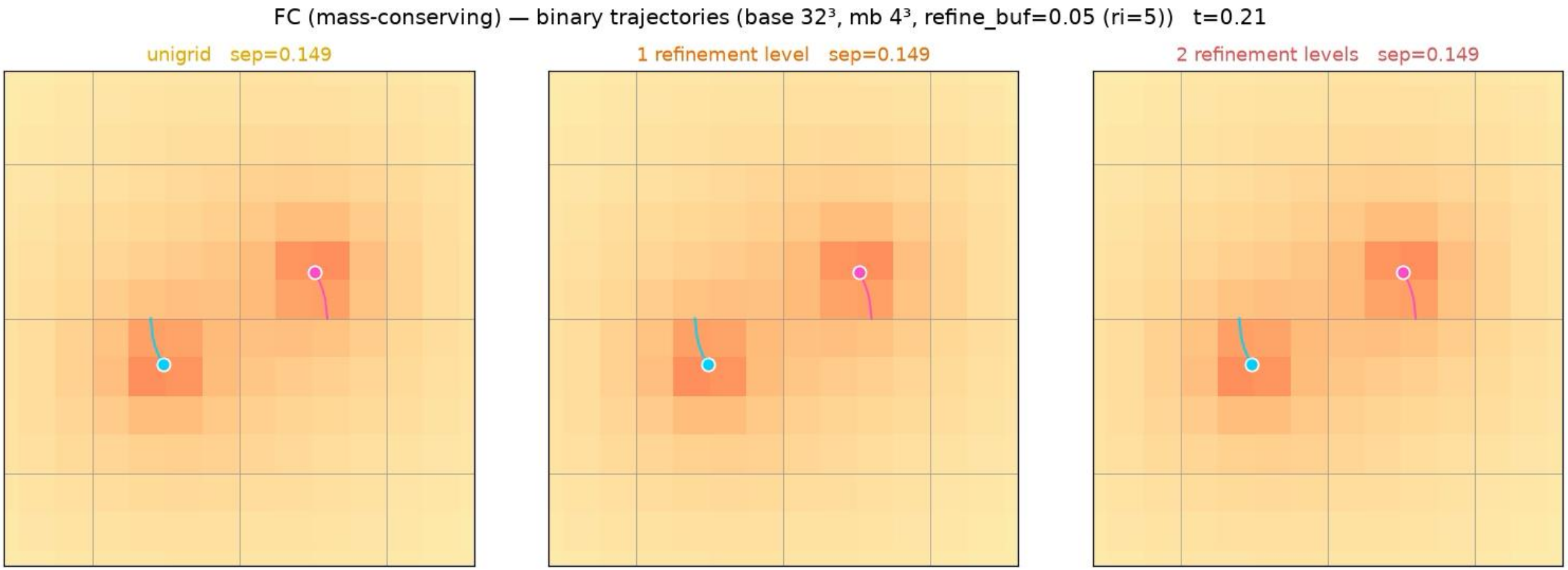
Binary self-gravity with static mesh refinement



The potential of two point masses resolved with an SMR patch — the fine grid captures the steep central wells while the coarse grid covers the far field.

APPLICATIONS

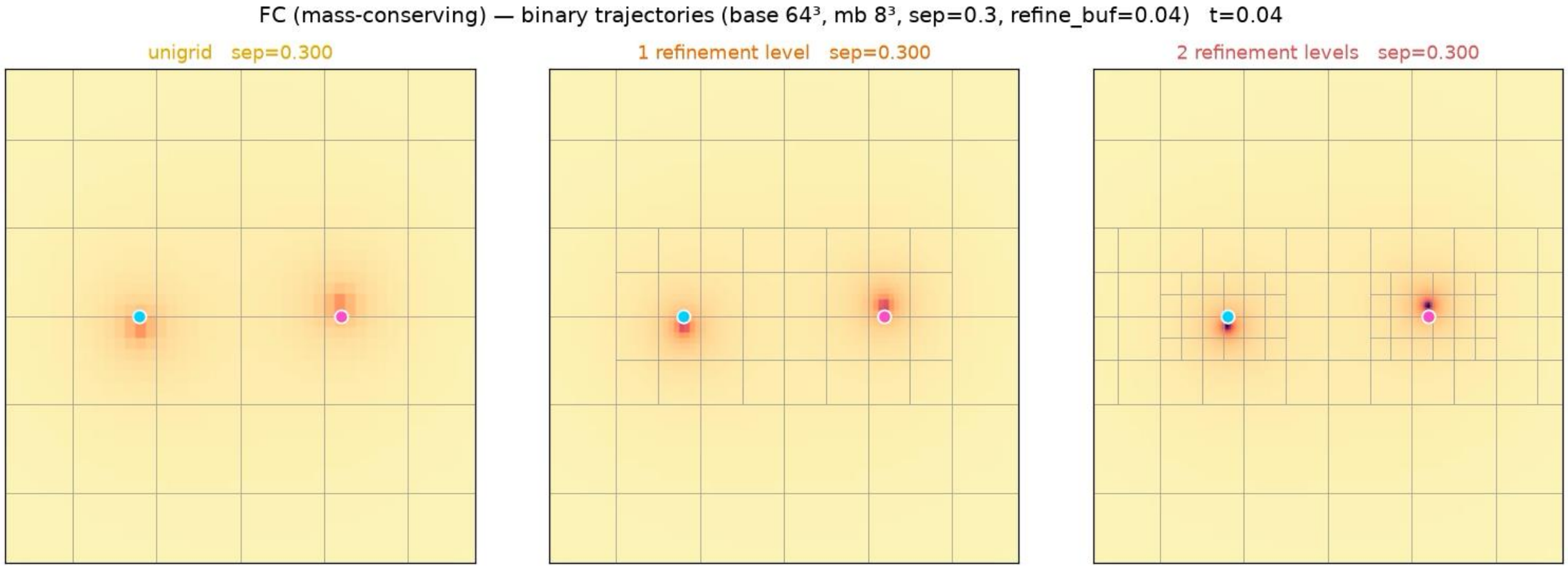
Refinement must not perturb the orbit



Mass-conserving (FC) scheme, close binary (sep ≈ 0.15 on a 32³ base grid): the same orbit run on a uniform grid, with one, and with two refinement levels — added refinement should leave the sink trajectories unchanged. The animation plays in slideshow mode.

APPLICATIONS

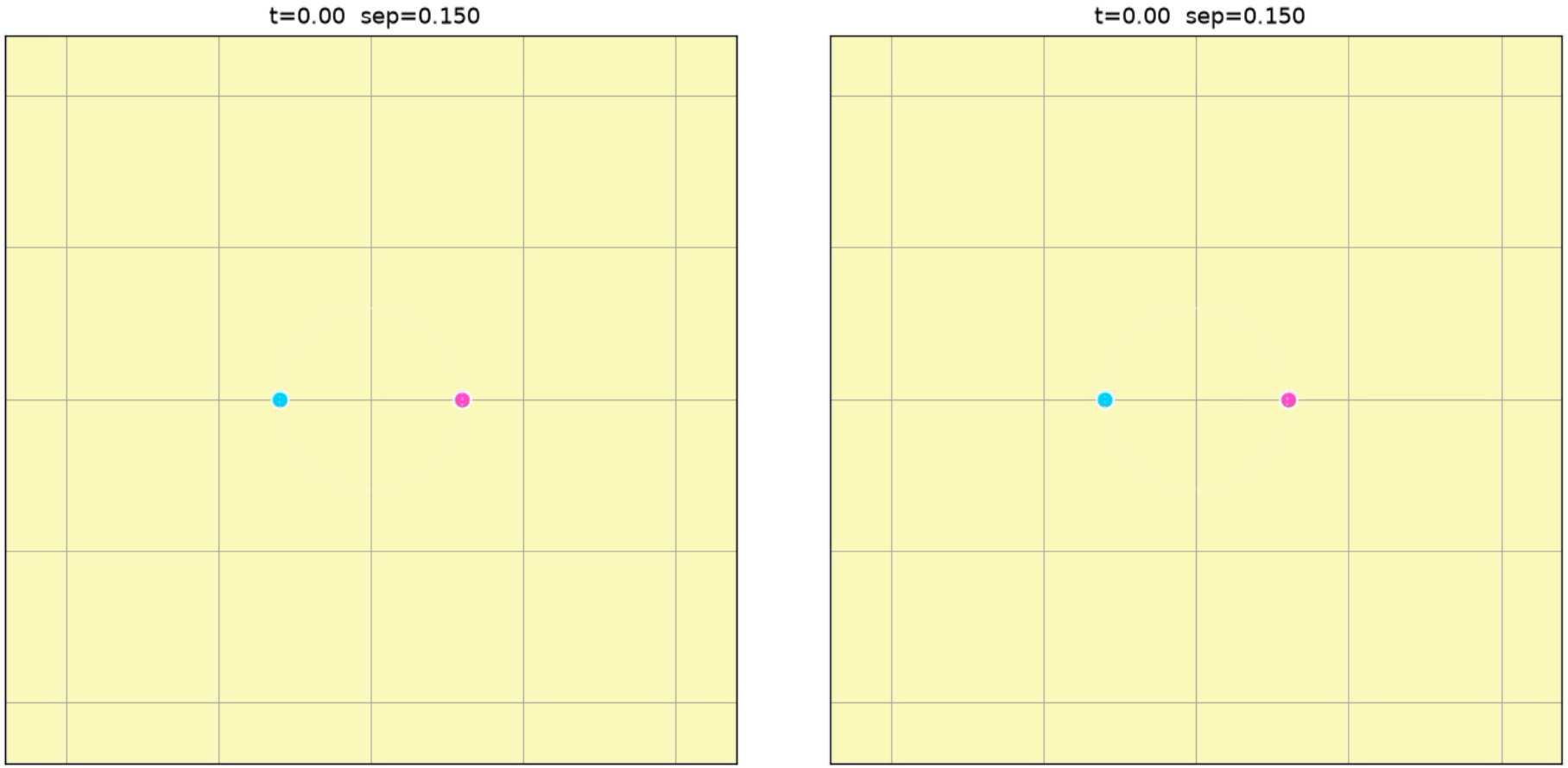
Refined blocks lock onto the sinks



The wider binary (sep = 0.3 on a 64³ base grid): refined meshblocks appear around each sink and travel with it along the orbit — again comparing unigrid against one and two refinement levels. The animation plays in slideshow mode.

APPLICATIONS

Four-level AMR vs the analytic orbit



RAMSES Fig 5 binary — adaptive multigrid self-gravity (base 32^3, mb 4^3, 4 levels); gray = live meshblocks, cyan = analytic orbit

Binary evolved with four refinement levels (base 32^3): grey shows the live meshblocks following the sinks, cyan the analytic orbit the trajectory should hold. The animation plays in slideshow mode.

PERFORMANCE

Why multigrid and not FFT?

Multigrid

· O(N): iteration count independent of resolution

· AMR & SMR native — solves all levels consistently

· periodic, fixed, zero-gradient, multipole boundaries

· mostly nearest-neighbour communication

· weak scaling stays nearly flat

FFT (spectral)

· O(N log N): cost per cell grows with resolution

· uniform Cartesian grids only — no AMR

· efficient only with periodic boundaries

· global all-to-all transposes every solve

· throughput degrades at large N — intrinsic

≈ 15%

cost of one FMG sweep vs the MHD solver (64^3 blocks)

≈ 70%

even FMG + 10 V-cycles — production needs only a few

O(N)

vs O(N log N): the gap widens exactly at the scales you care about

FFT is excellent on a uniform periodic box — but self-gravity in AthenaK needs refinement, mixed boundaries, and scale. Athena++'s own FFT solver degrades at large N by construction, while multigrid stays a small fraction of the MHD cost (Tomida & Stone 2023, Fig. 6).

PERFORMANCE

Why multigrid and not FFT?

256^3 · single GH200 · 1 FMG + N V-cycles = 1 MHD step

MHD baseline = **upstream main, ideal EOS** (PLM/HLLD, γ=5/3)

meshblock decomposition increases → (cooler = multigrid more favorable)

	128^3 8 blocks MHD 66.9ms	64^3 64 blocks MHD 78.2ms	32^3 512 blocks MHD 147.4ms	16^3 4096 blocks MHD 310.8ms
FMG cycle × MHD step	0.77×	0.56×	0.28×	0.27×
V-cycle × MHD step	0.26×	0.24×	0.13×	0.12×
+ V-cycles fit after 1 FMG	0.9	1.8	5.4	6.0

row 1–2: solve wall ÷ one upstream ideal-MHD step. row 3: (1 – FMG/MHD) ÷ (V-cycle/MHD) = V-cycles that fit in the leftover of one MHD step after a single FMG. 1-block case dropped (aborts under ideal MHD). On 1 GPU upstream MHD = user's MHD (opts are comm-path).

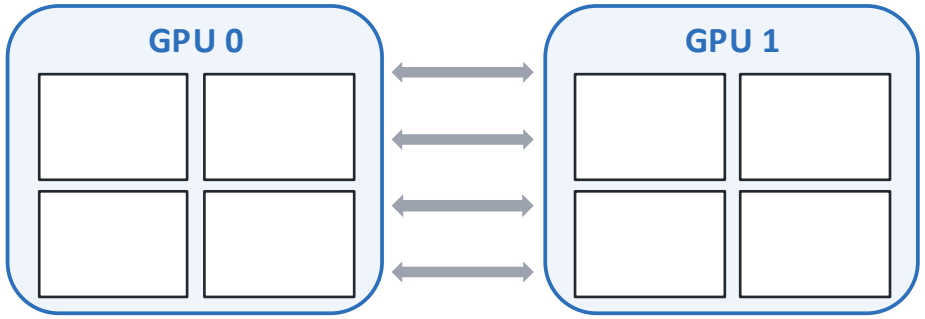
FFT is excellent on a uniform periodic box — but self-gravity in AthenaK needs refinement, mixed boundaries, and scale. Athena++'s own FFT solver degrades at large N by construction, while multigrid stays a small fraction of the MHD cost (Tomida & Stone 2023, Fig. 6).

PERFORMANCE

Multigrid is communication-bound on coarse grids

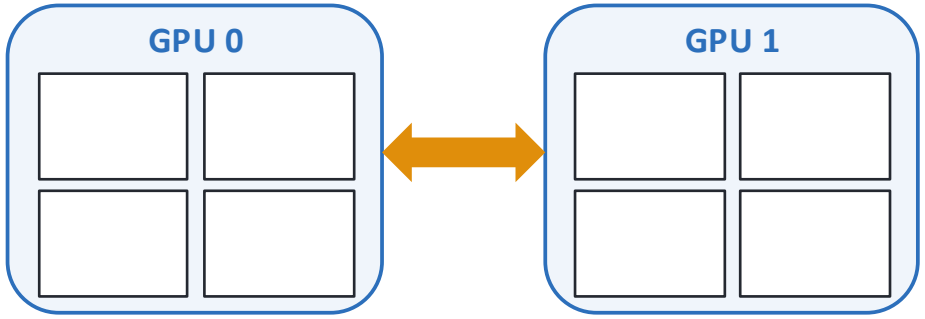
- **Many tiny messages.** Every V-cycle visits every level; coarse levels are heavily decomposed, so each rank exchanges many small halos.
- **Latency, not bandwidth.** On coarse grids the messages are tiny — cost is dominated by per-message latency.
- **Baseline collapse.** As meshblocks-per-GPU and node count grow, naive per-message comms throttle the whole solve.
- **The fix.** Rank-packed communication aggregates all halos for a neighbor rank into one message, per level.

baseline: one message per neighbouring block pair



× every block · × every level · × every V-cycle

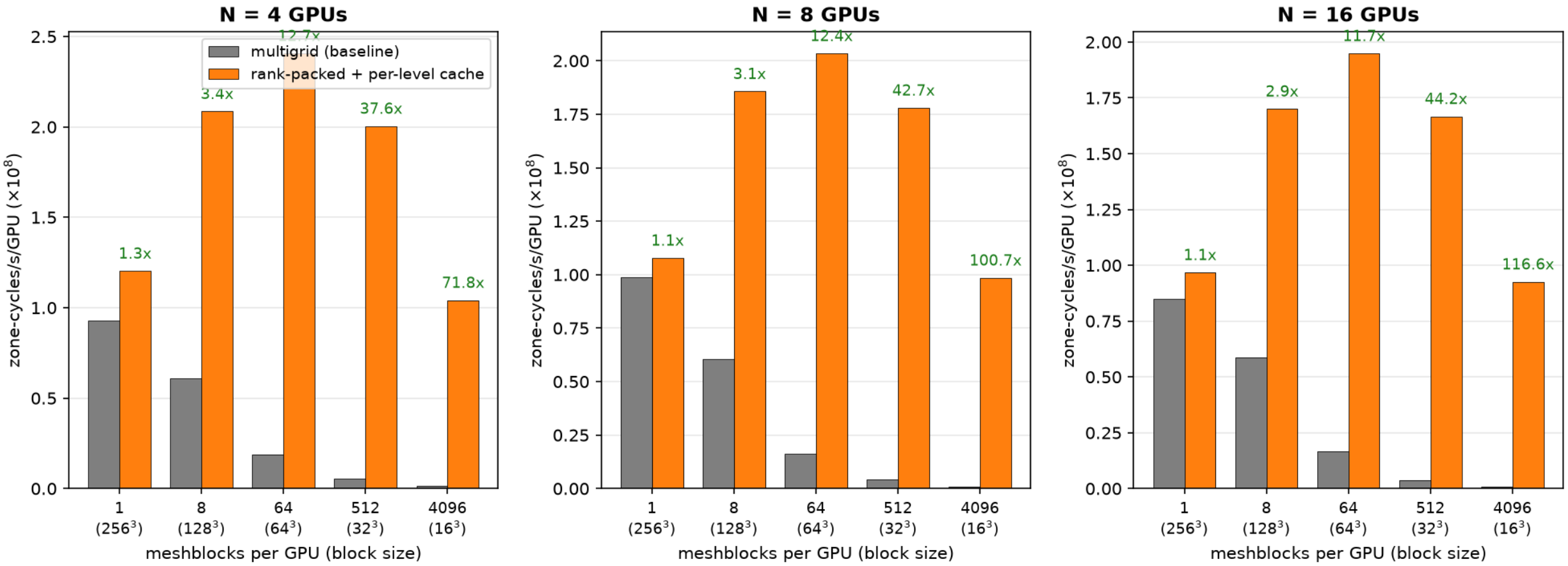
rank-packed: one buffer per neighbouring rank



messages per level: O(blocks) → O(ranks) — the latency is paid once per rank, not once per block

PERFORMANCE

Rank-packed communication: up to ~100× at scale



Weak scaling on Vista B200 GPUs, 256³ per GPU. Speedup grows with both GPU count and decomposition, reaching ~100–117× in the most-decomposed regime.

PERFORMANCE

Multi-GPU weak scaling

Multi-GPU weak scaling · 256³/GPU · GH200 (isothermal MHD)

rows = meshblocks per GPU · cols = GPU count · cooler = multigrid cheaper · flat rows ⇒ weak-scaling holds

FMG cycle / MHD step

blk/GPU	1 GPU	2 GPUs	4 GPUs	8 GPUs
8 (128³)	0.94	1.08	1.08	1.13
512 (32³)	0.37	0.43	0.43	0.45
4096 (16³)	0.35	0.36	0.35	0.35

V-cycle / MHD step

blk/GPU	1 GPU	2 GPUs	4 GPUs	8 GPUs
8 (128³)	0.33	0.38	0.38	0.40
512 (32³)	0.17	0.19	0.19	0.20
4096 (16³)	0.16	0.16	0.16	0.15

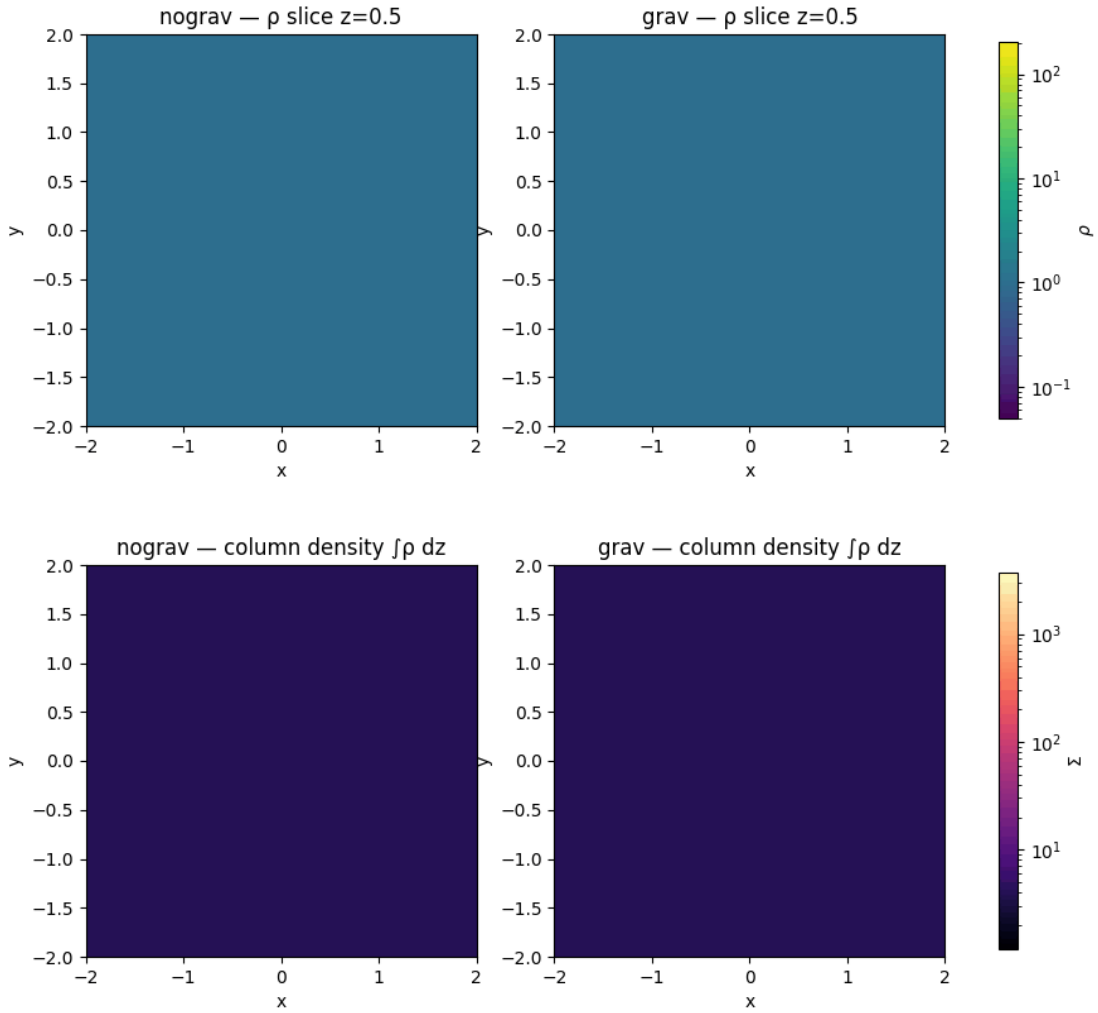
cost = solve wall ÷ one MHD step, per timestep. Both MG and MHD grow ~20% in absolute time from 1→8 GPUs (comm), but together — so the ratio is GPU-count-independent (rank-packed comms).

+ V-cycles fit after 1 FMG

blk/GPU	1 GPU	2 GPUs	4 GPUs	8 GPUs
8 (128³)	0.2	-0.2	-0.2	-0.3
512 (32³)	3.7	3.0	3.0	2.8
4096 (16³)	4.1	4.0	4.2	4.2

APPLICATIONS

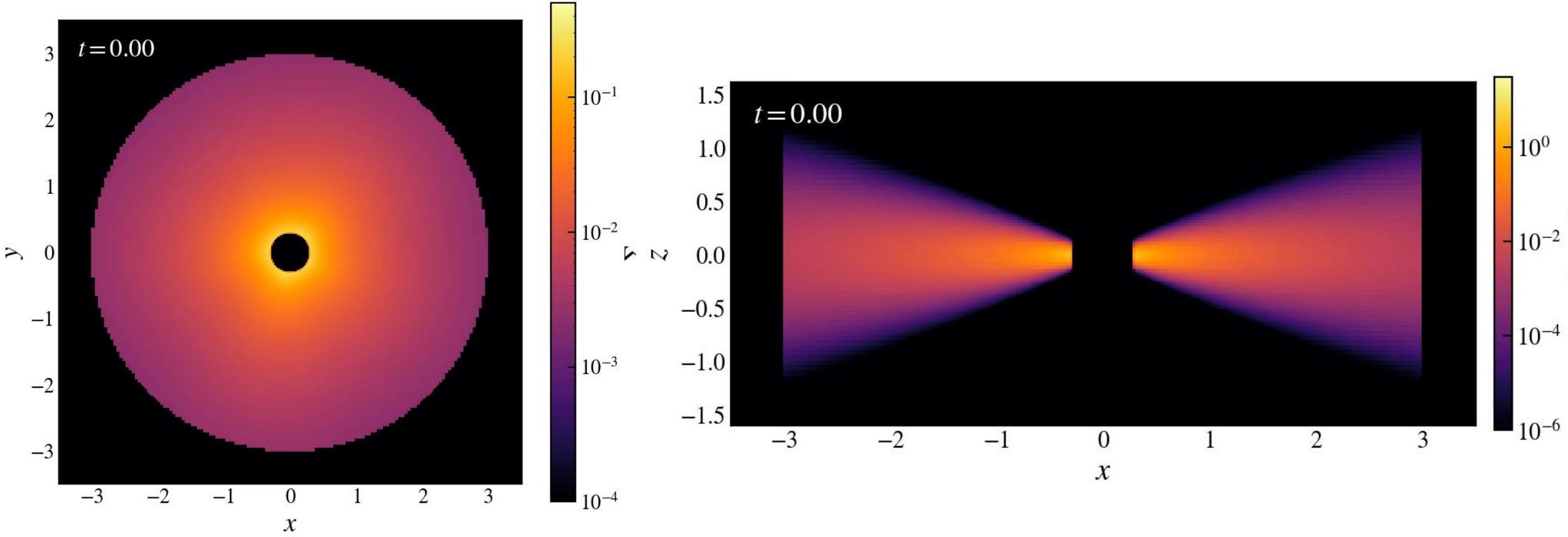
Turbulent fragmentation under self-gravity



Driven turbulence with self-gravity in AthenaK: overdense filaments exceed their local Jeans mass and collapse into fragments — the multigrid solver runs every timestep as the collapse deepens. The animation plays in slideshow mode.

APPLICATIONS

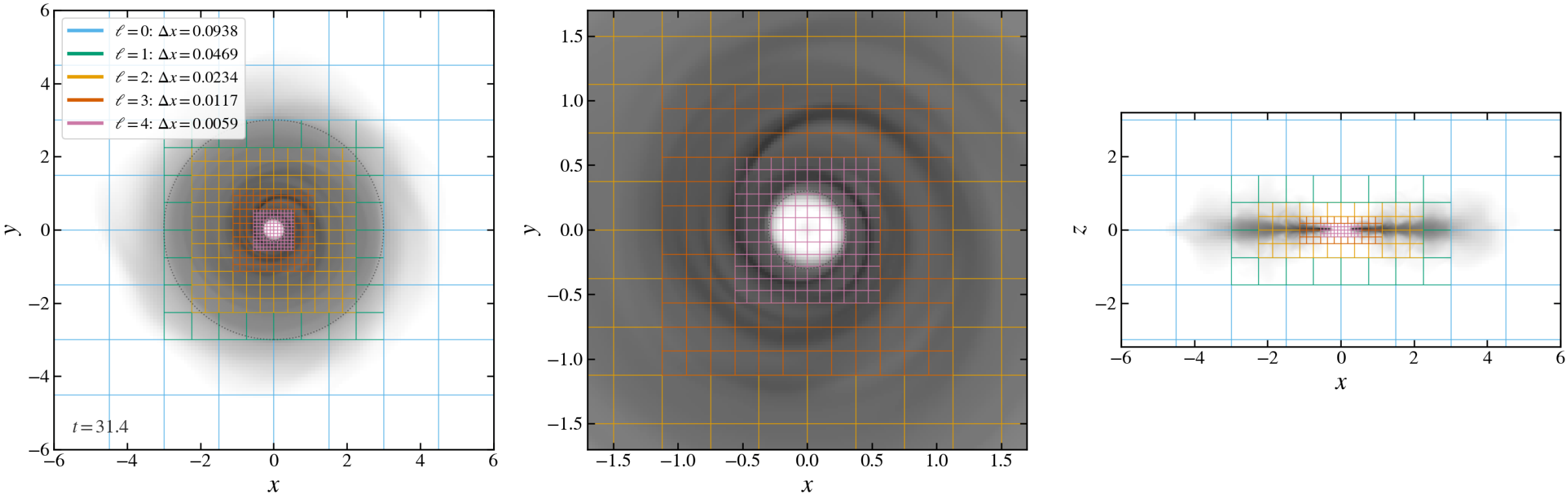
Gravitationally unstable disk



A self-gravitating disk in AthenaK, face-on (surface density, left) and edge-on (right): spiral perturbations grow until overdense arms exceed their local Jeans mass and collapse — the multigrid solver supplies the potential every timestep. The animations play in slideshow mode.

APPLICATIONS

Refinement follows the collapse



Snapshot at t ≈ 31: five levels of adaptive refinement (Δx = 0.094 → 0.006) concentrate resolution on the spiral arms and the collapsing central fragment — multigrid solves the Poisson equation across the full level hierarchy. Panels: face-on view, zoom on the fragment, edge-on view.

SUMMARY

A fast, portable, validated self-gravity solver

- **Intuition.** Relaxation smooths; coarsening turns smooth error into rough error — so relaxation across a grid hierarchy attacks every scale.
- **Method.** V-cycle / FMG with full-weighting restriction and interpolation gives grid-independent convergence (~0.06 per cycle to machine precision).
- **Validated.** Defect → 1e-8 in ~10 cycles, decomposition-independent, and reproduces the Jeans dispersion relation for hydro and MHD.
- **Portable & scalable.** One Kokkos source, CPU/GPU, SMR/AMR; rank-packed comms give up to ~100× at scale on B200 GPUs.
- **Applications.** Binary self-gravity with SMR and AMR, evolved self-consistently.