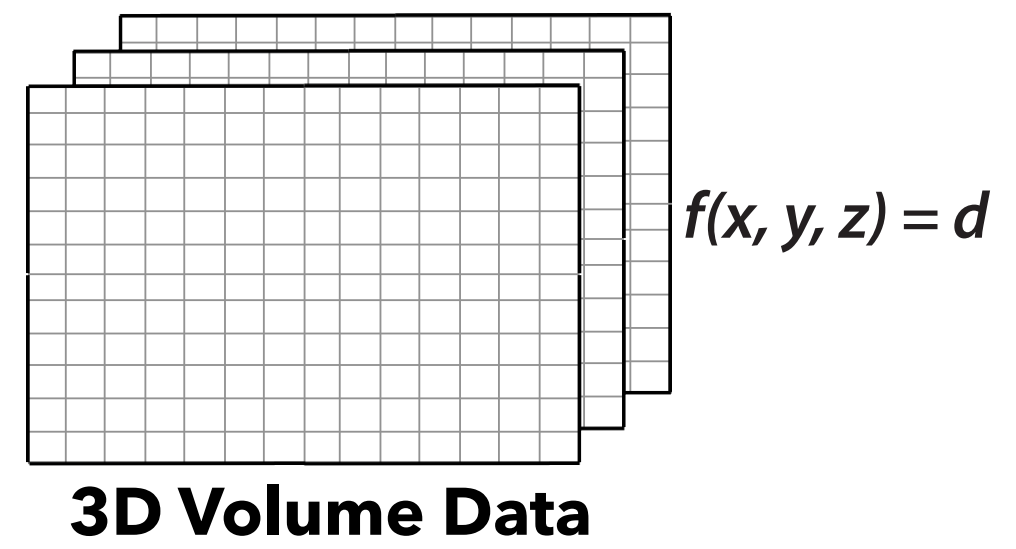
Interactive *In Situ* Visualization of Large Distributed Volume Data

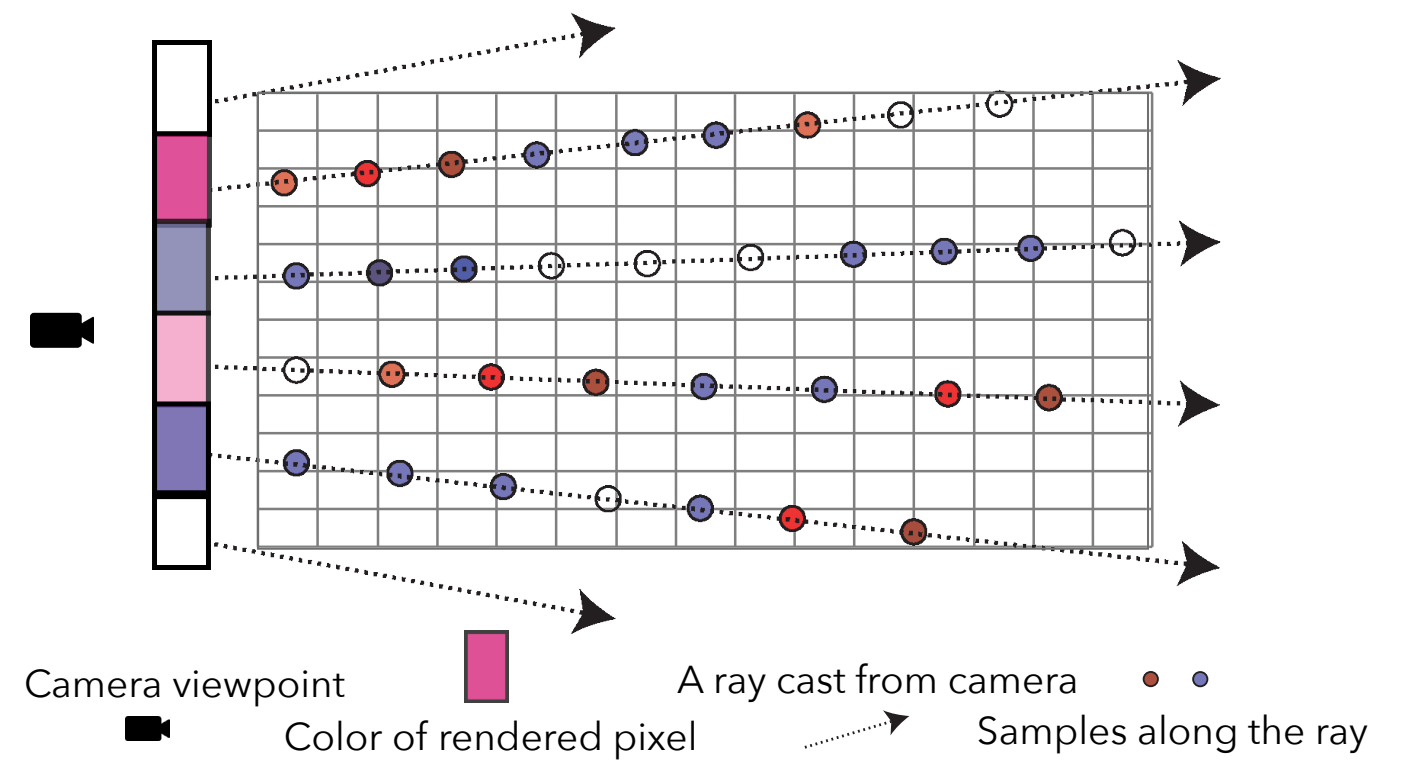
Aryaman Gupta, advised by Ivo F. Sbalzarini, Technische Universität Dresden, Center for Systems Biology Dresden, MPI-CBG, Dresden

Large distributed volume data are routinely generated in numerical simulations and experiments. Here, we propose a solution for visualizing them at smooth, interactive frame rates, outperforming existing tools and techniques.



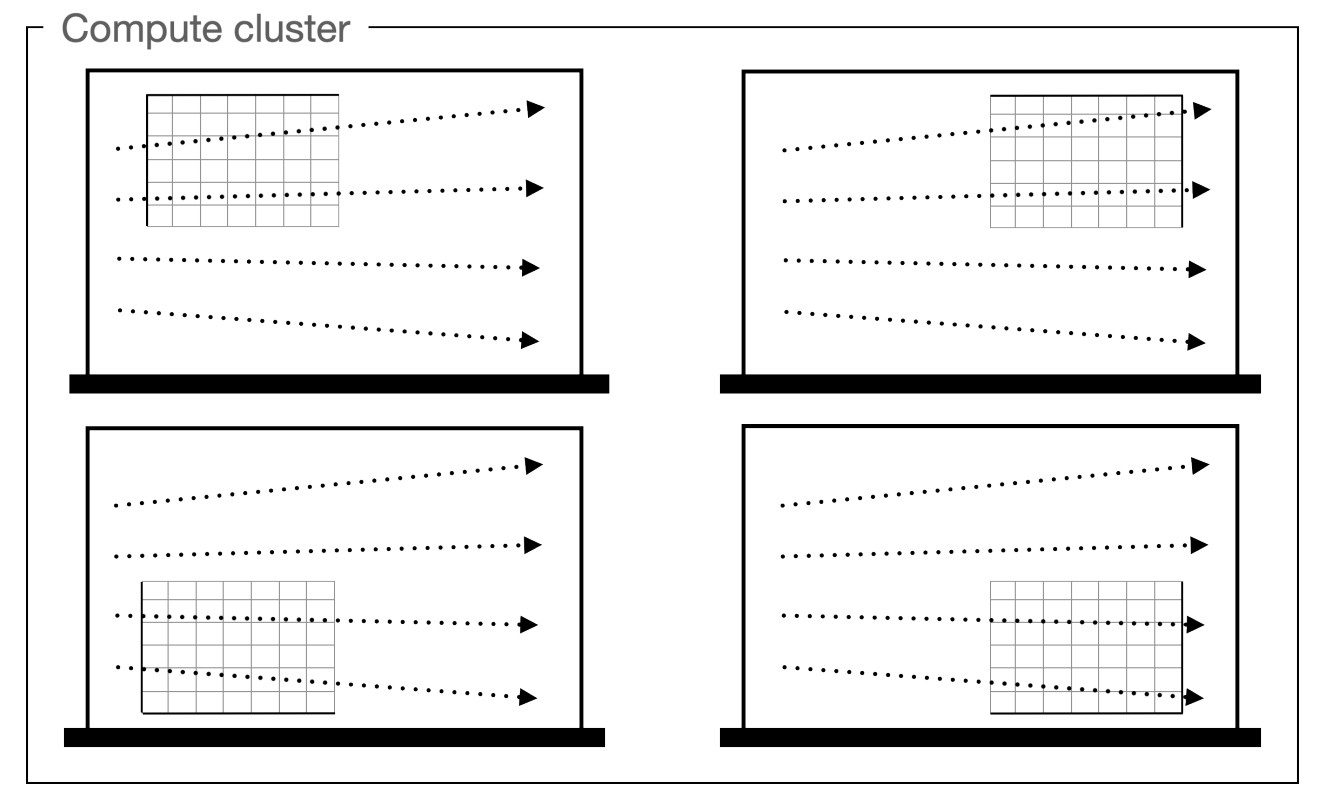
State of the art

Direct volume rendering (DVR)



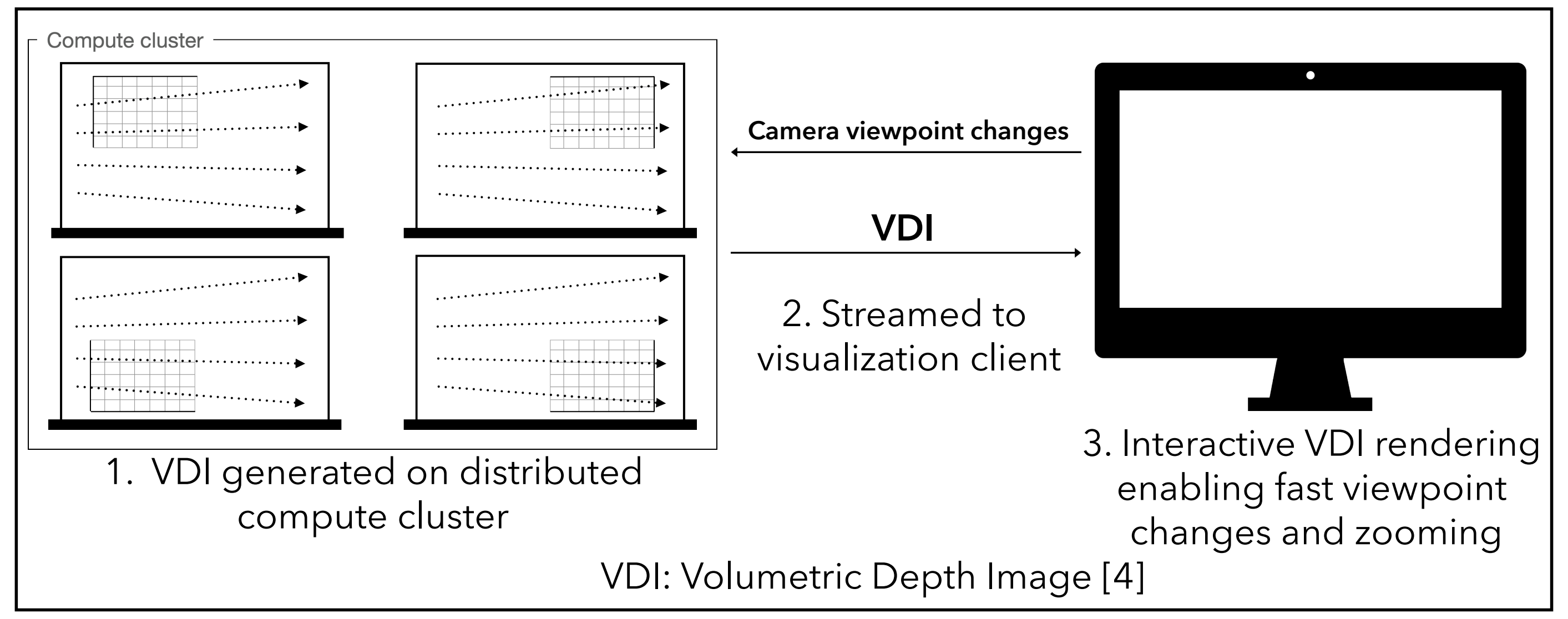
For each pixel to be rendered, a ray is cast from the user's viewpoint through the volume. The ray samples the volume at regular intervals to determine the final color and opacity for the pixel.

on distributed compute clusters



For distributed data, each processor performs direct volume rendering on its part of the domain to produce a "sub-image". These sub-images are then composited in parallel to produce the final image. This is called sort-last parallel rendering.

Solution Overview



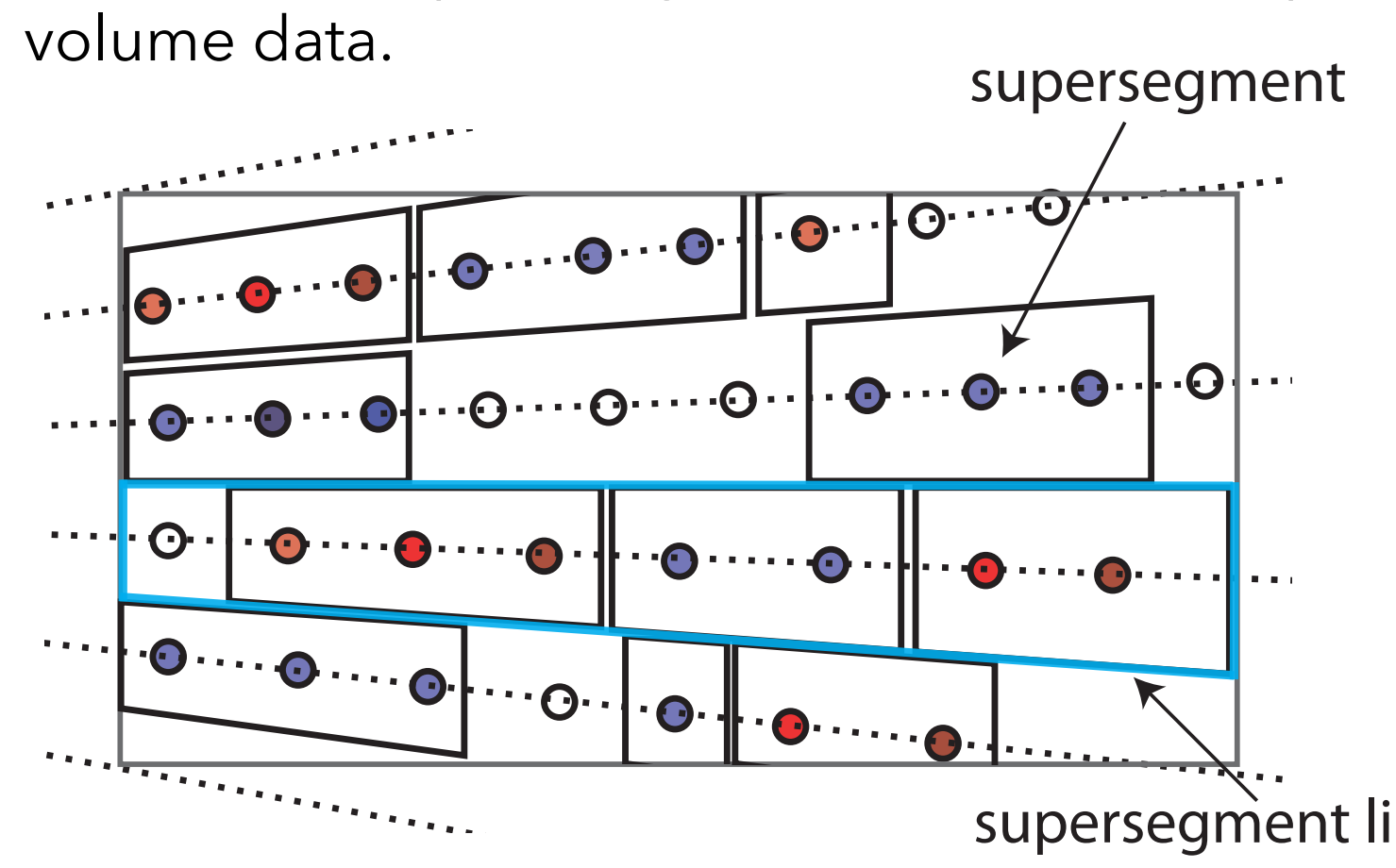
1. VDI generated on distributed compute cluster

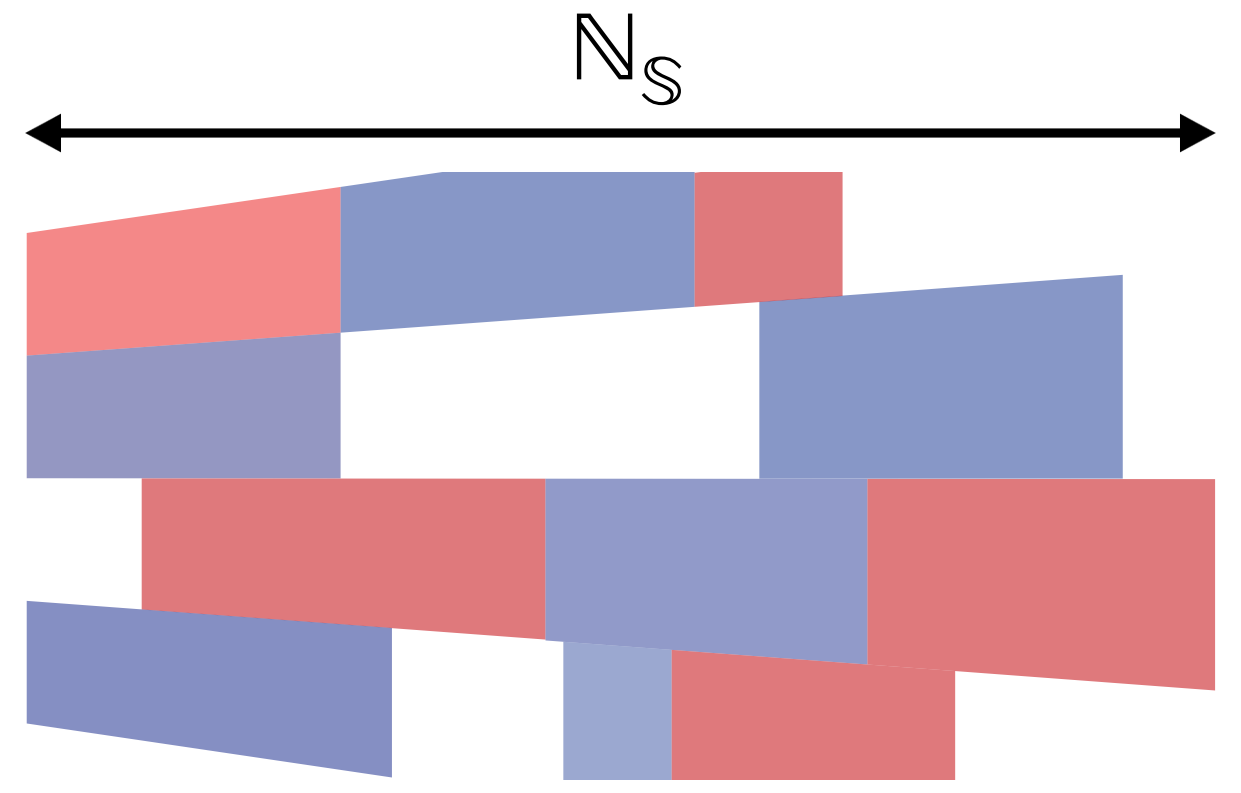
2. Streamed to visualization client

3. Interactive VDI rendering enabling fast viewpoint changes and zooming

VDI: Volumetric Depth Image [4]

Volumetric Depth Images (VDIs) are view-dependent compact representations of volume data.





Each ray divides the data into regions of similar color and opacity, called *supersegments*. The set of supersegments generated by a ray is called a *supersegment list*, or simply *list*.

Each supersegment stores accumulated color and opacity between its start and end points, and the volume is represented by the set of supersegments.

The number of supersegments generated in a list is limited to a user-defined maximum $\mathbb{N}_{\mathbb{S}}$.

The VDI is highly compressed along the view direction for which it was constructed, and can be rendered fast, providing high quality approximations.

Contributions:

1. First method for generating VDIs on distributed computers.
2. Strategies for interactive *in situ* visualization of distributed numerical simulations.
3. Novel rendering technique for VDIs, enabling interactive visualization.

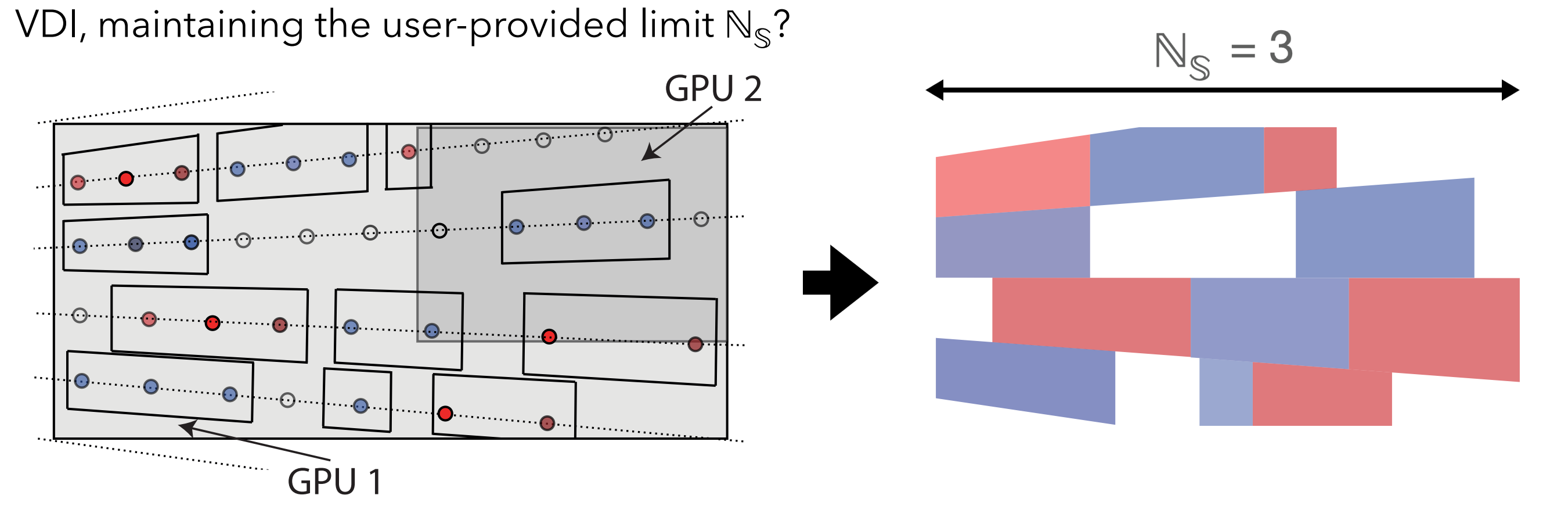
Open-source software: Contributions implemented as extension of visualization framework *scenery* [4] and simulation framework OpenFPM [6].

1 Generating VDIs on distributed compute clusters

A novel method for generating VDIs on distributed data maintaining scalability, quality, and handling non-convex domain decompositions (EGPGV 2023, Best Paper Award [1]).

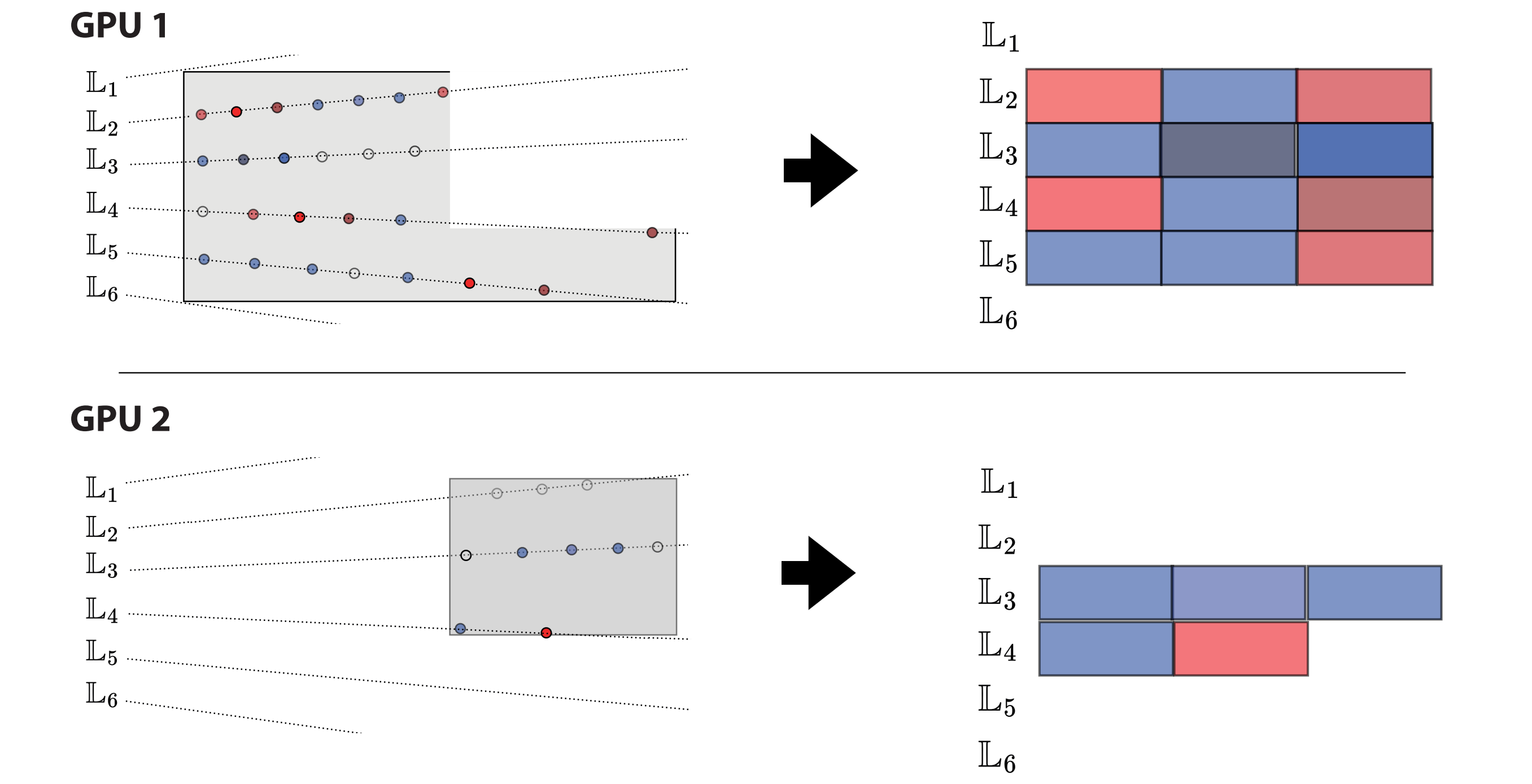
Problem Statement

Data distributed across multiple processors (e.g. GPUs). How do we generate the same VDI, maintaining the user-provided limit $\mathbb{N}_{\mathbb{S}}$?

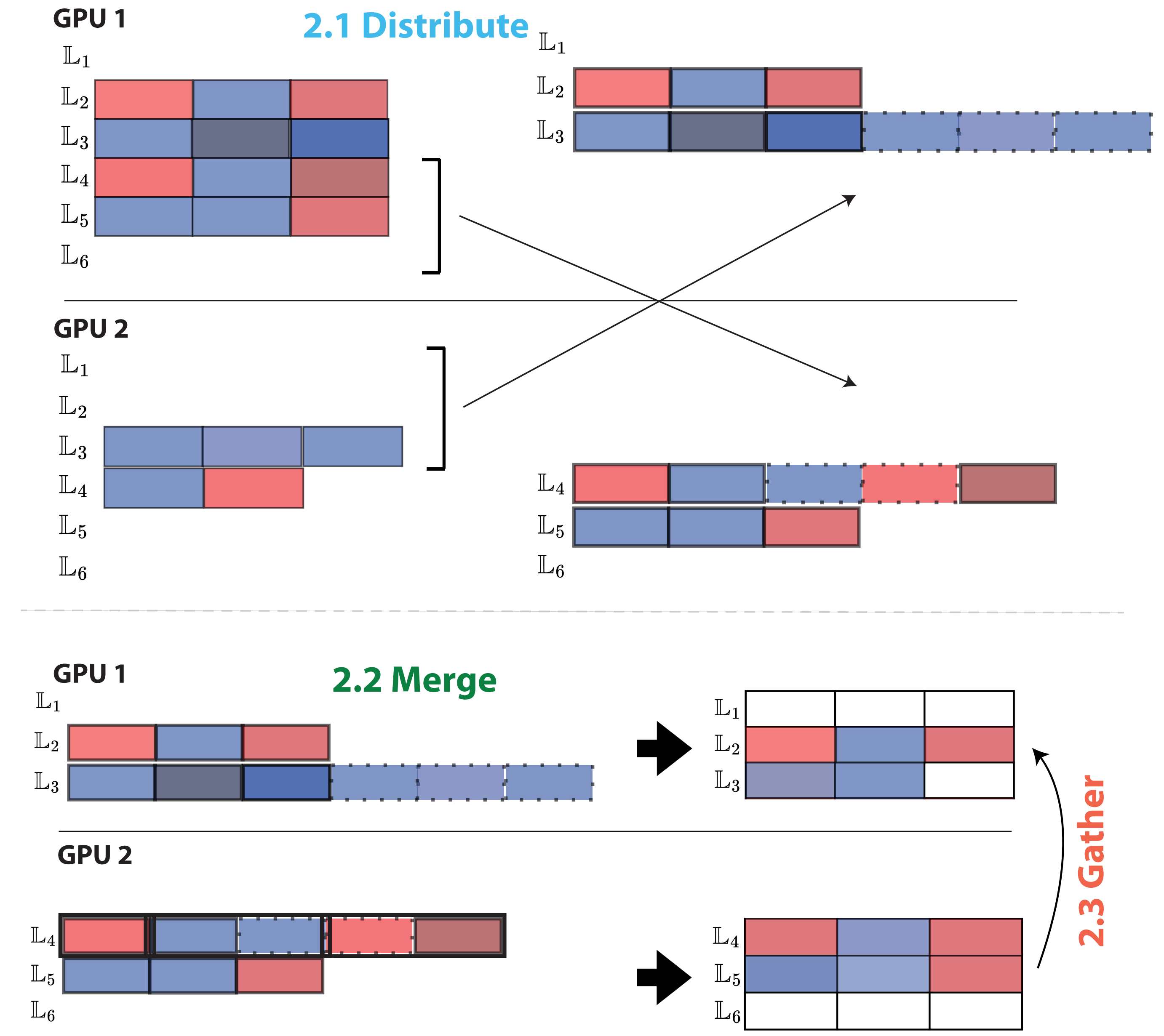


We follow a sort-last approach, as described below:

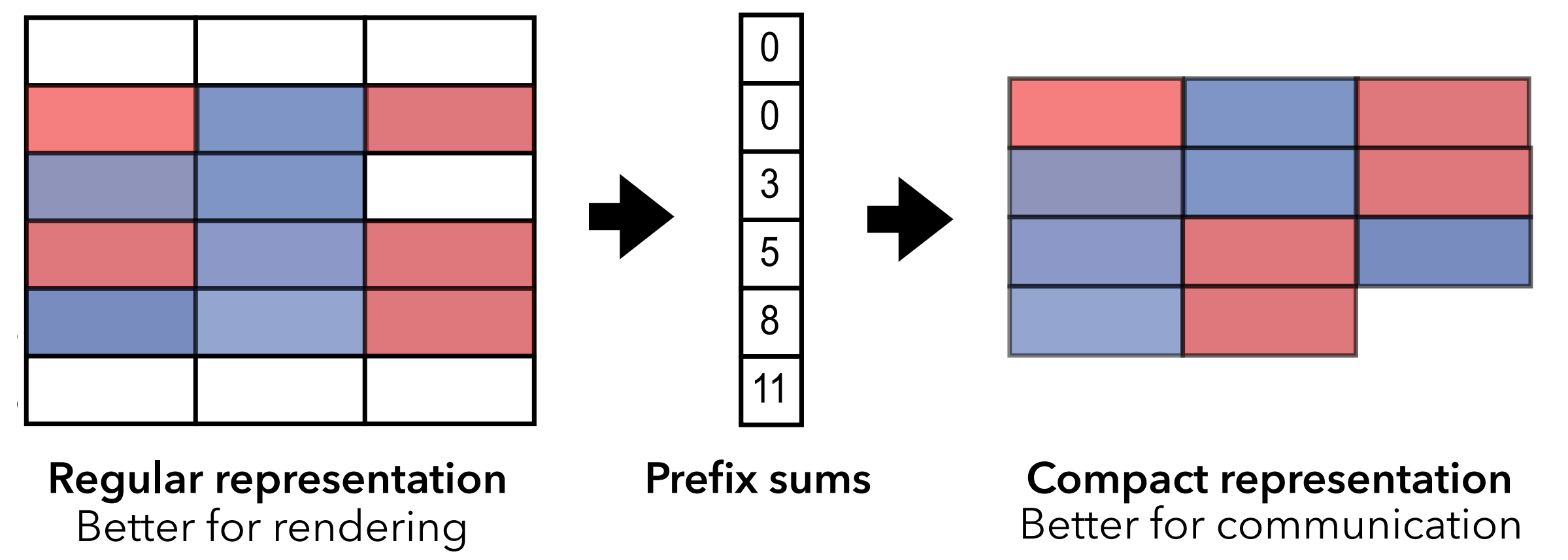
Phase 1: sub-VDI generation


GPU 1
$\mathbb{L}_1$ $\mathbb{L}_2$ $\mathbb{L}_3$ $\mathbb{L}_4$ $\mathbb{L}_5$ $\mathbb{L}_6$
GPU 2
$\mathbb{L}_1$ $\mathbb{L}_2$ $\mathbb{L}_3$ $\mathbb{L}_4$ $\mathbb{L}_5$ $\mathbb{L}_6$

Phase 2: Parallel compositing of sub-VDIs


GPU 1
2.1 Distribute
GPU 2
GPU 1
2.2 Merge
GPU 2
2.3 Gather
$\mathbb{L}_1$ $\mathbb{L}_2$ $\mathbb{L}_3$ $\mathbb{L}_4$ $\mathbb{L}_5$ $\mathbb{L}_6$

We propose a compact representation to optimize inter-process (MPI) communication.

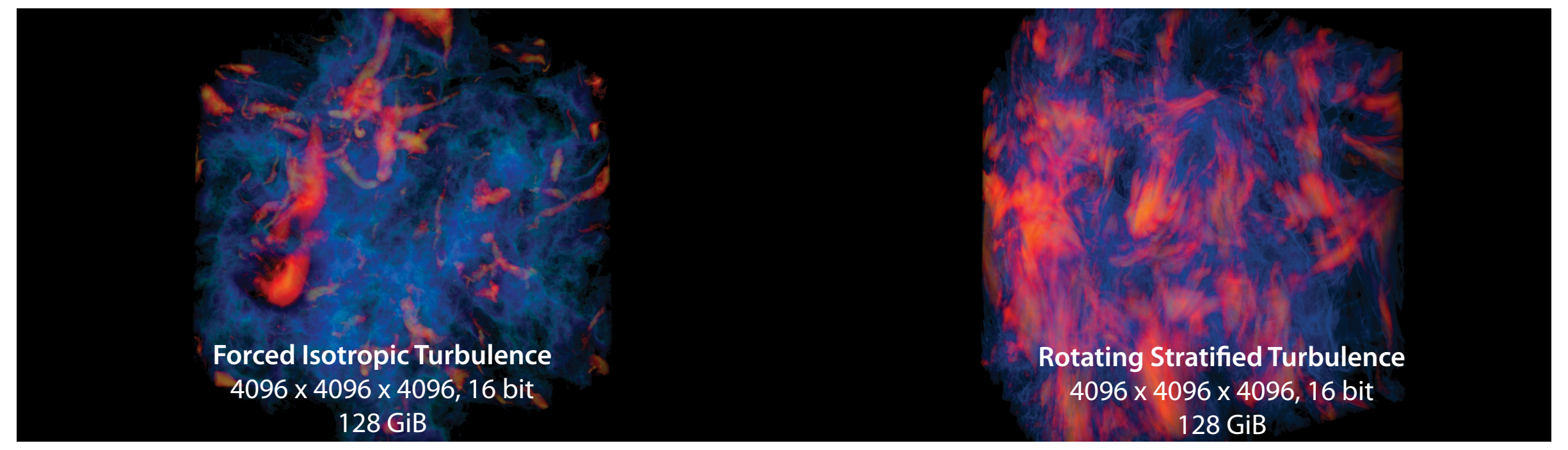


Regular representation Better for rendering

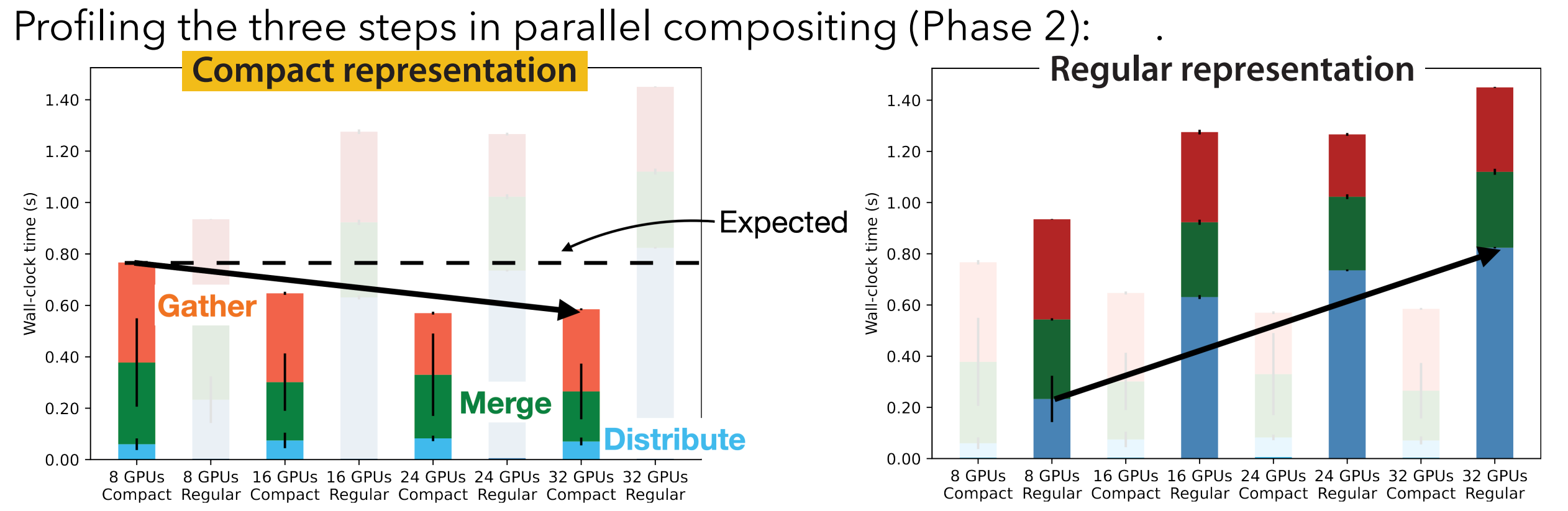
Prefix sums

Compact representation Better for communication

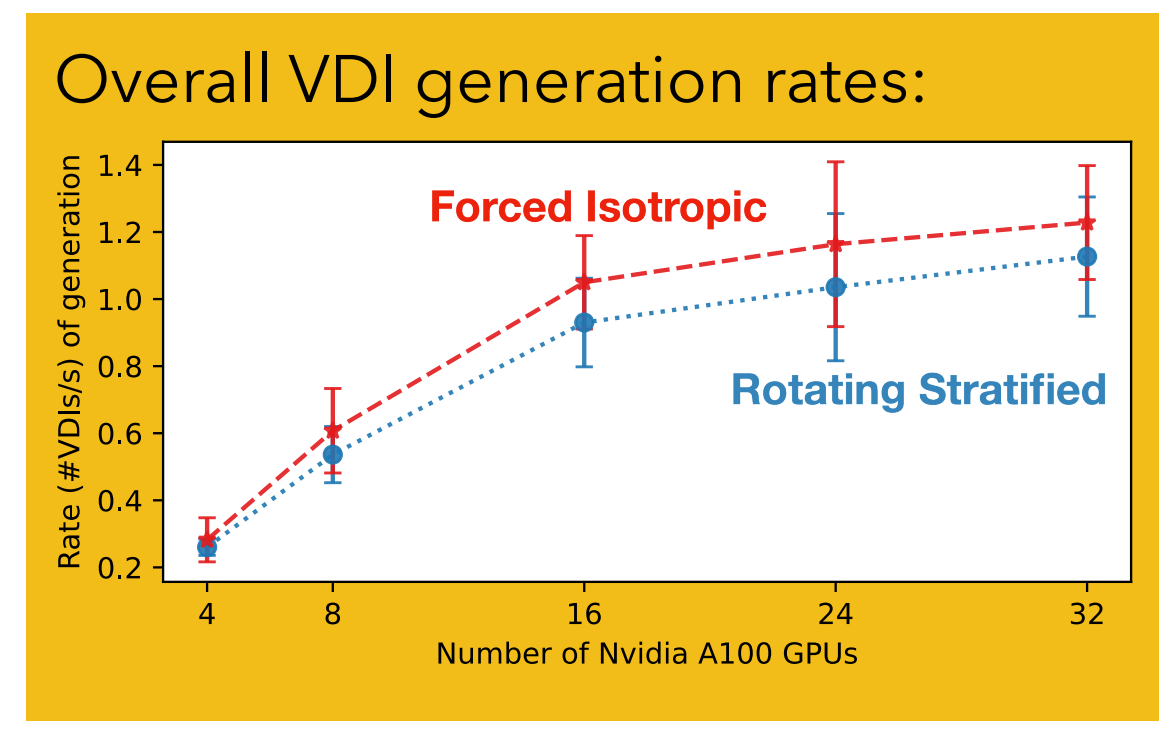
Performance Benchmarks


Forced Isotropic Turbulence
4096 x 4096 x 4096, 16 bit
128 GiB
Rotating Stratified Turbulence
4096 x 4096 x 4096, 16 bit
128 GiB

Profiling the three steps in parallel compositing (Phase 2):



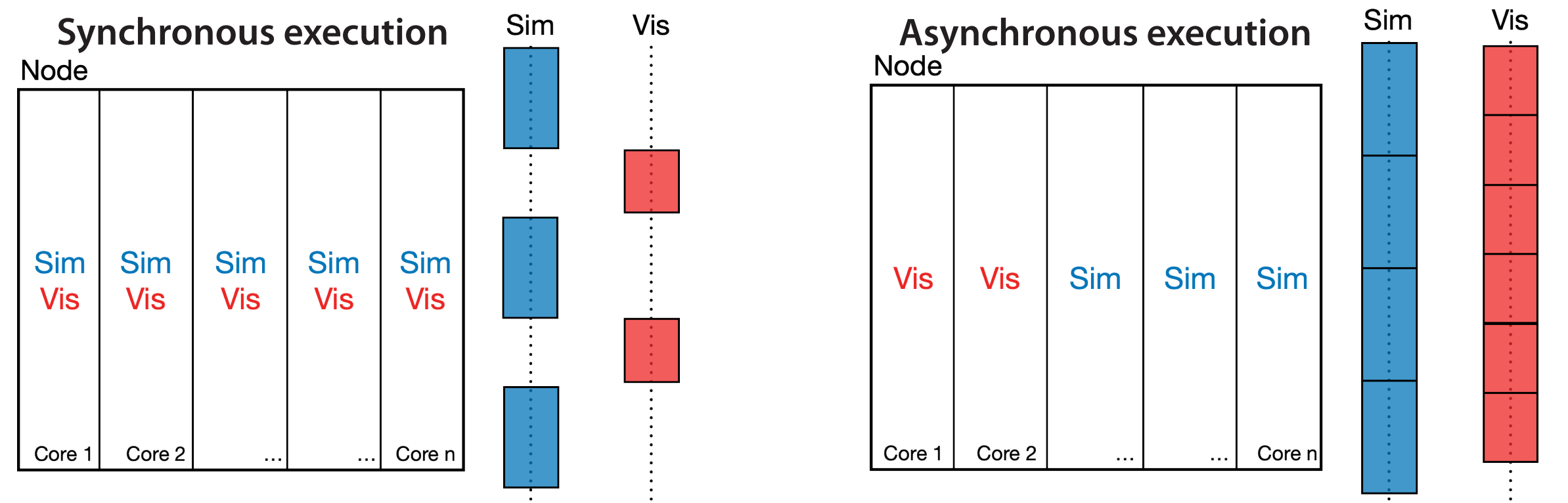
Overall VDI generation rates:



The proposed compact representation is crucial for scalability in parallel compositing. Overall, VDIs are generated in less than one second when using 32 GPUs for both 128 GiB datasets. This is smaller than typical simulation time steps, implying the VDI could be used for live *in situ* visualization at full temporal resolution.

2 Architecture for *in situ* visualization

We propose asynchronous execution to minimize latency, and provide an implementation as extension of the simulation framework OpenFPM (ISAV 2020 [2]).

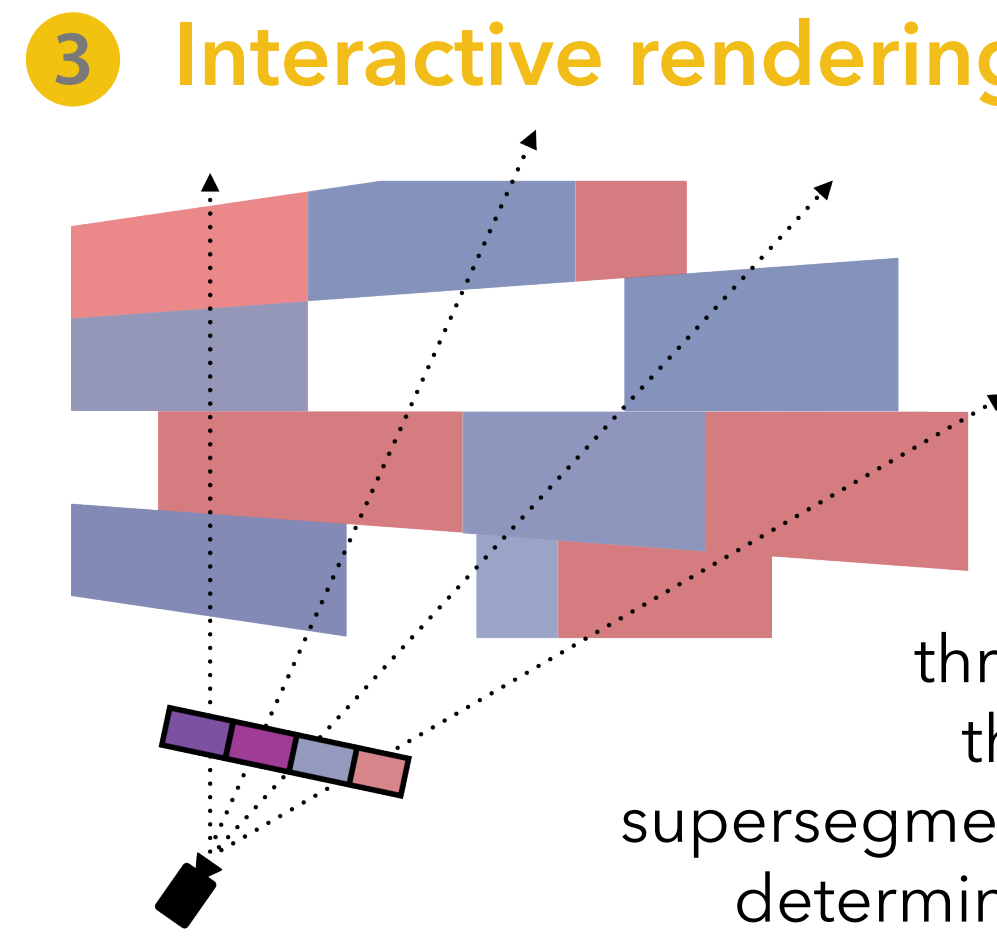


OpenFPM is an open-source framework for particle and mesh simulations. We extend it, transparently providing *in situ* visualization functionality requiring minimal changes in existing OpenFPM-based simulation code.

```
…
openfpm_init(&argc, &argv,
  init_options::in_situ_visualization);
…
vd.visualize<0>(); //in the time loop
…
```

Scan the QR code for the OpenFPM framework. The *in situ* visualization functionality is currently on the *insitu_visualization* branch, soon to be merged into the *main* branch.

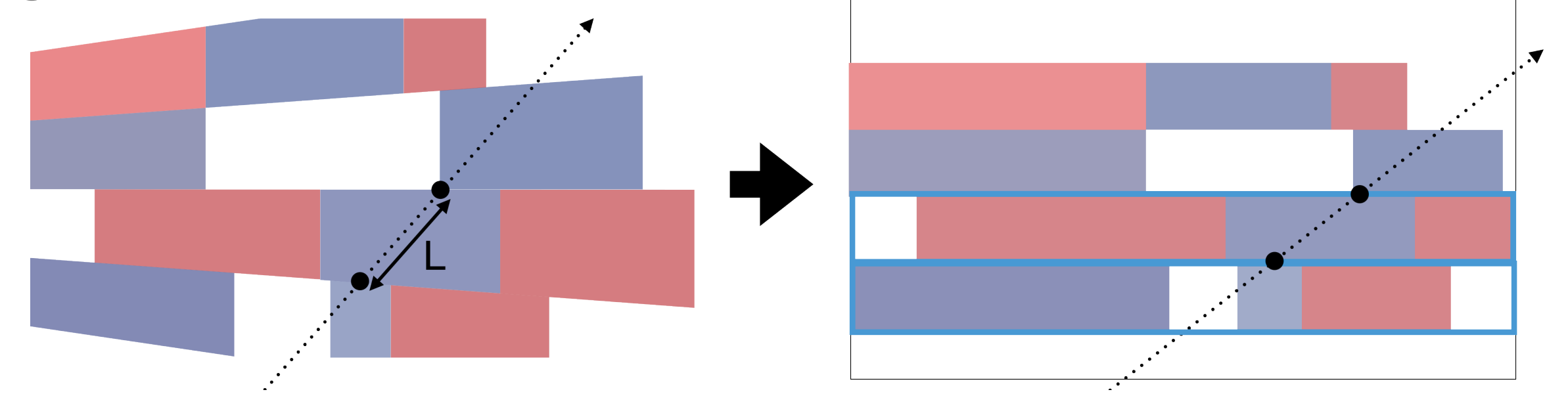
3 Interactive rendering of VDIs



A raycasting-based technique for rendering VDIs, outperforming existing solution by more than an order of magnitude (IEEE PacificVis, 2023 [3]).

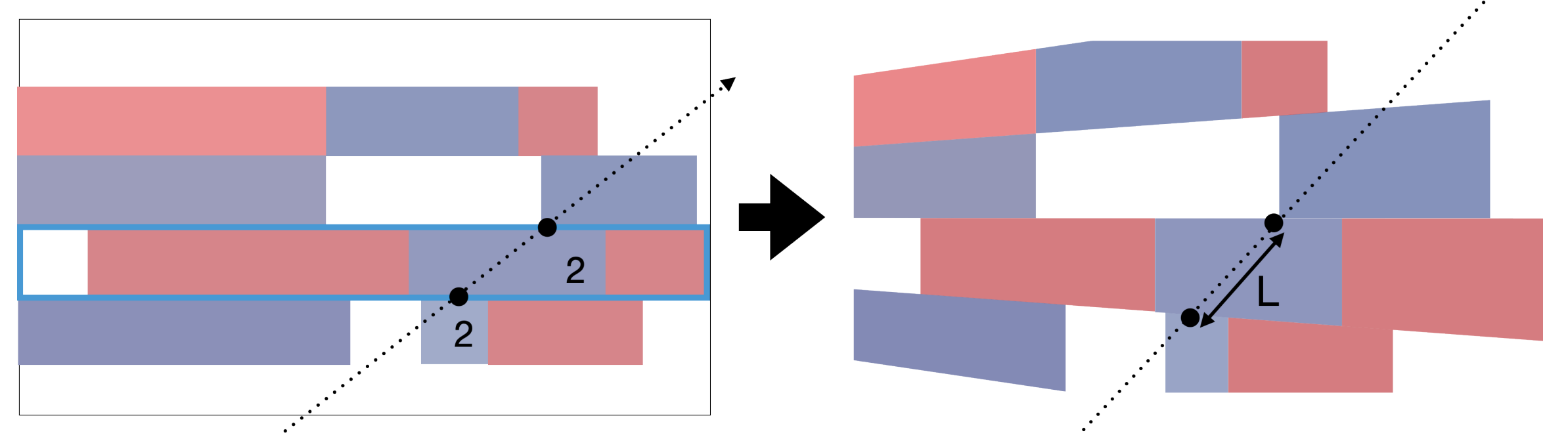
For each pixel to be rendered, a ray is cast through the VDI. Each ray is mapped to a single GPU thread. Rays intersect supersegment lists, search for supersegments within them, accumulating color and opacity to determine the final color and opacity of the rendered pixel.

Algorithm overview



Calculating intersection lengths is expensive as supersegments are irregular pyramidal frustums in 3D.

We optimize by projecting the VDI to Normalized Device Coordinate (NDC) space, where supersegments are transformed into cuboids.



To minimize memory accesses, index of the supersegment intersected in the previous list is used as an initial guess. To minimize memory latency, supersegment depths are pre-fetched before comparison. This increases rendering frame-rates by up to 40%.

Finally, the intersection points are projected back to view space to determine intersection length. The process continues until the ray exits the VDI or opacity saturates.

Performance is further optimized by empty-space skipping and preview rendering. VDI generation and rendering code is implemented as an extension of the *scenery* [4] framework.

scenery framework
(VDI code currently on *remote_volume_visualization* branch)

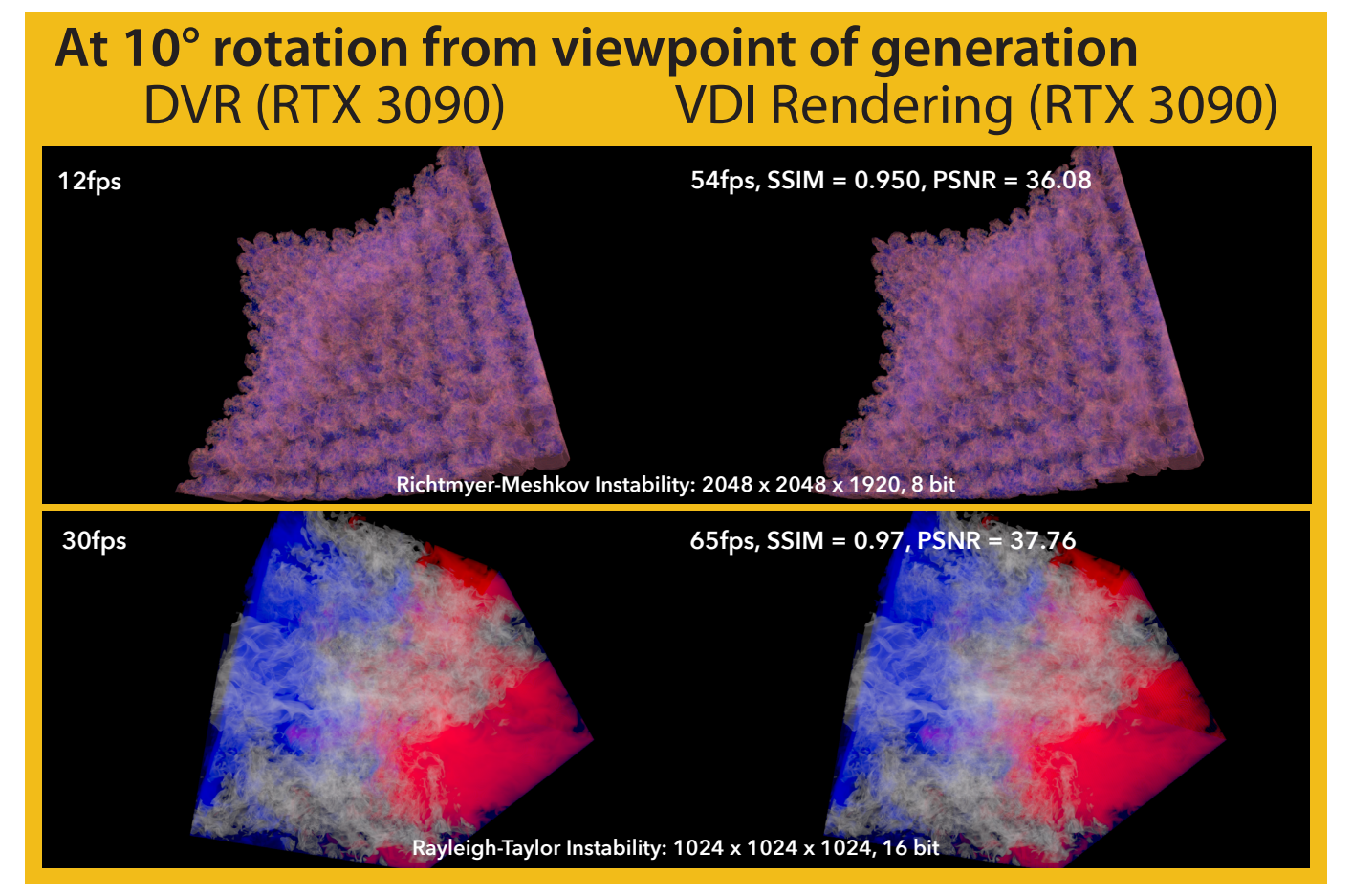
Performance Benchmarks

Our method vastly outperforms the existing technique for rendering VDIs by Frey et al. [4]. Both methods are tested on identical VDIs and hardware (an Nvidia RTX 3090), at different viewpoint deviations about the viewpoint from which the VDI was generated. Performance is measured in frames per second (fps).

Dataset	fps at 10° deviation		fps at 40° deviation	
	Existing	Ours	Existing	Ours
Kingsnake	11	**441**	10	**44**
Rayleigh-Taylor	5	**226**	5	**124**

VDI rendering is significantly faster than direct volume rendering, while providing close approximations. The adjacent figure reports image similarity using the SSIM [5] and PSNR (Peak Signal-to-Noise Ratio) metrics.

At 10° rotation from viewpoint of generation
DVR (RTX 3090) VDI Rendering (RTX 3090)


12fps
54fps, SSIM = 0.950, PSNR = 36.08
Richtmyer-Meshkov Instability: 2048 x 2048 x 1920, 8 bit
30fps
65fps, SSIM = 0.97, PSNR = 37.76
Rayleigh-Taylor Instability: 1024 x 1024 x 1024, 16 bit

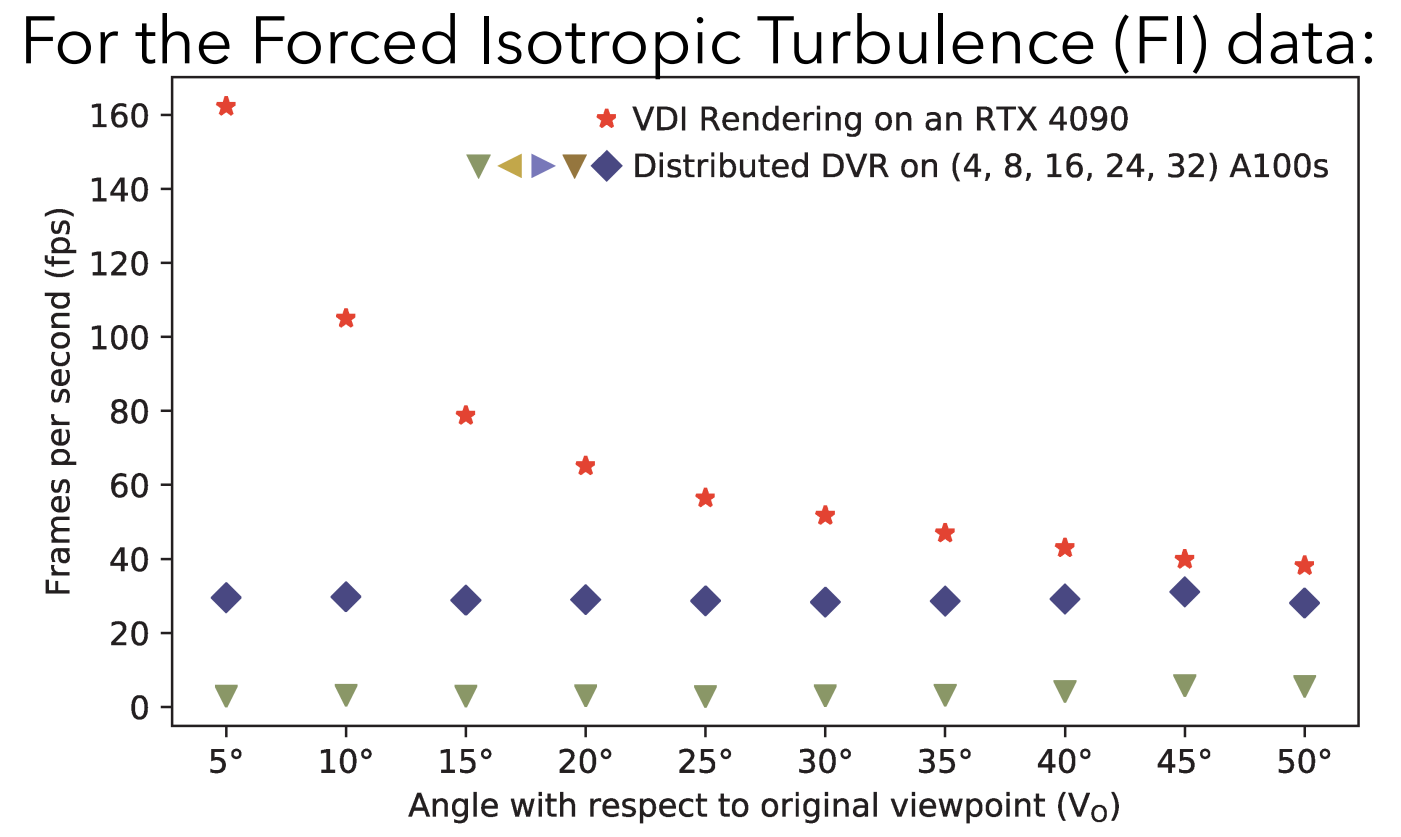
Interactive rendering videos

Single-GPU datasets

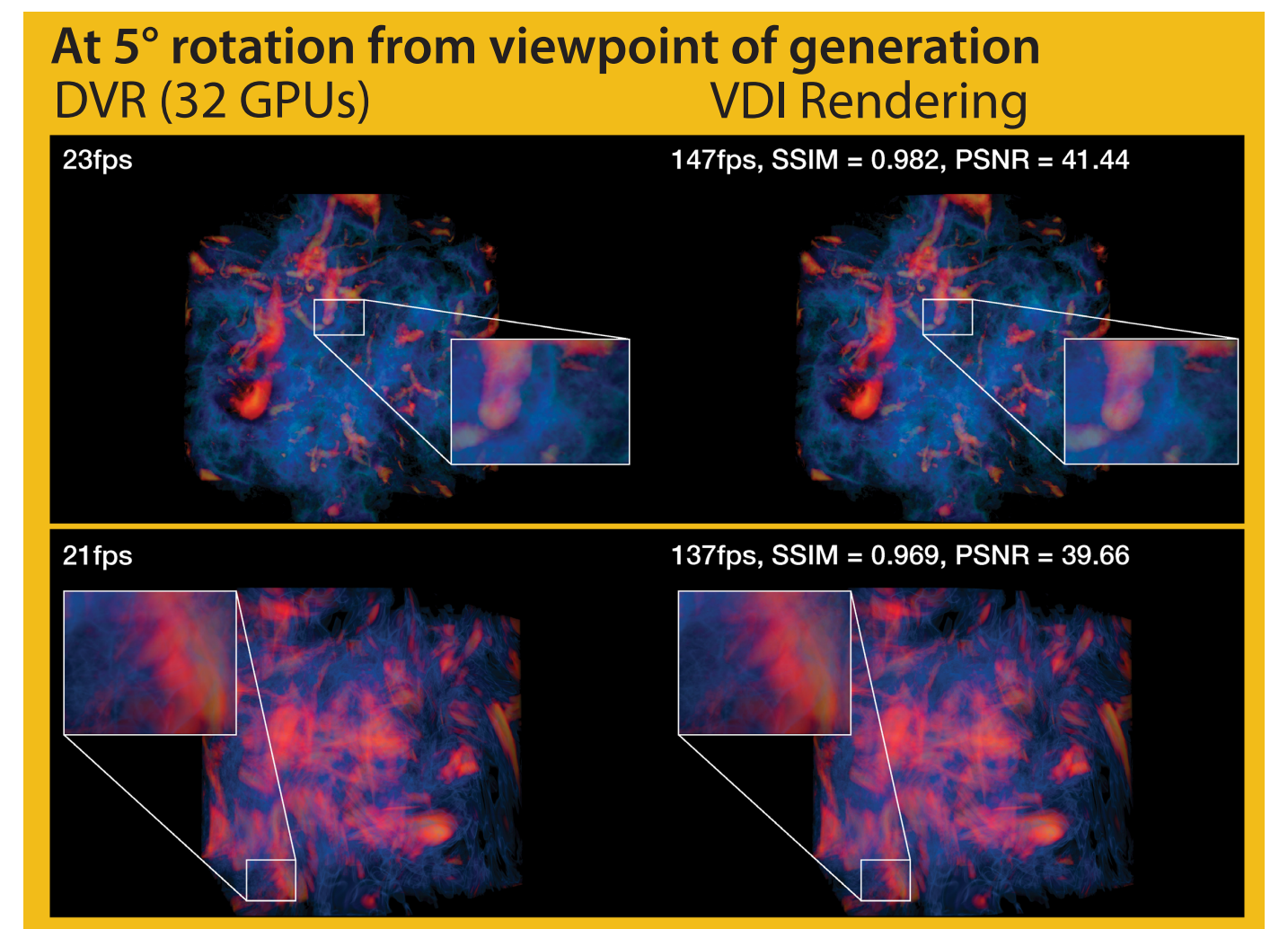
Larger datasets

VDI rendering provides higher frame rates than distributed direct volume rendering (DVR) implemented using IceT library for parallel compositing, which is used by popular visualization tools like ParaView and VisIt.

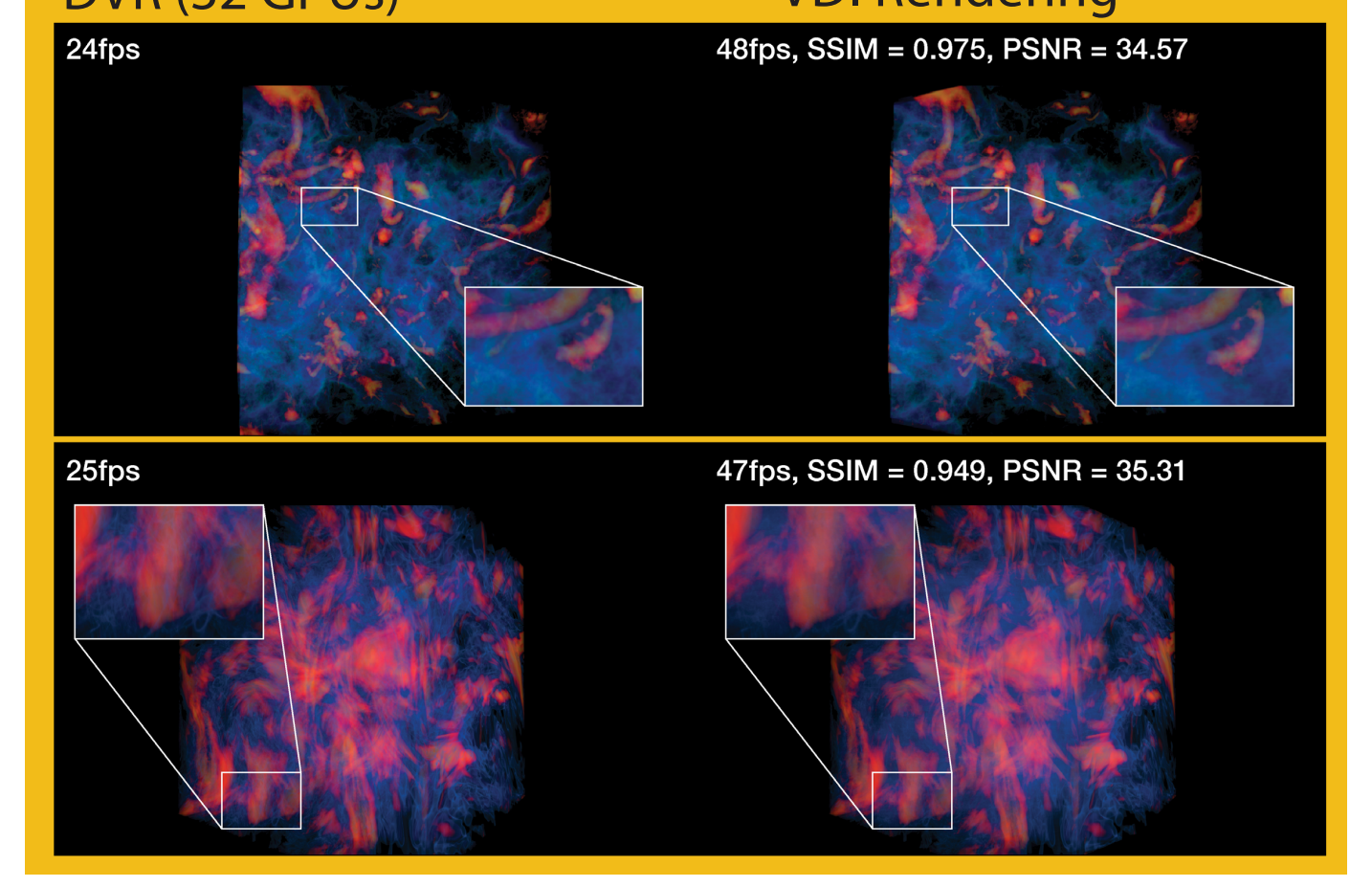
For the Forced Isotropic Turbulence (FI) data:



At 5° rotation from viewpoint of generation
DVR (32 GPUs) VDI Rendering

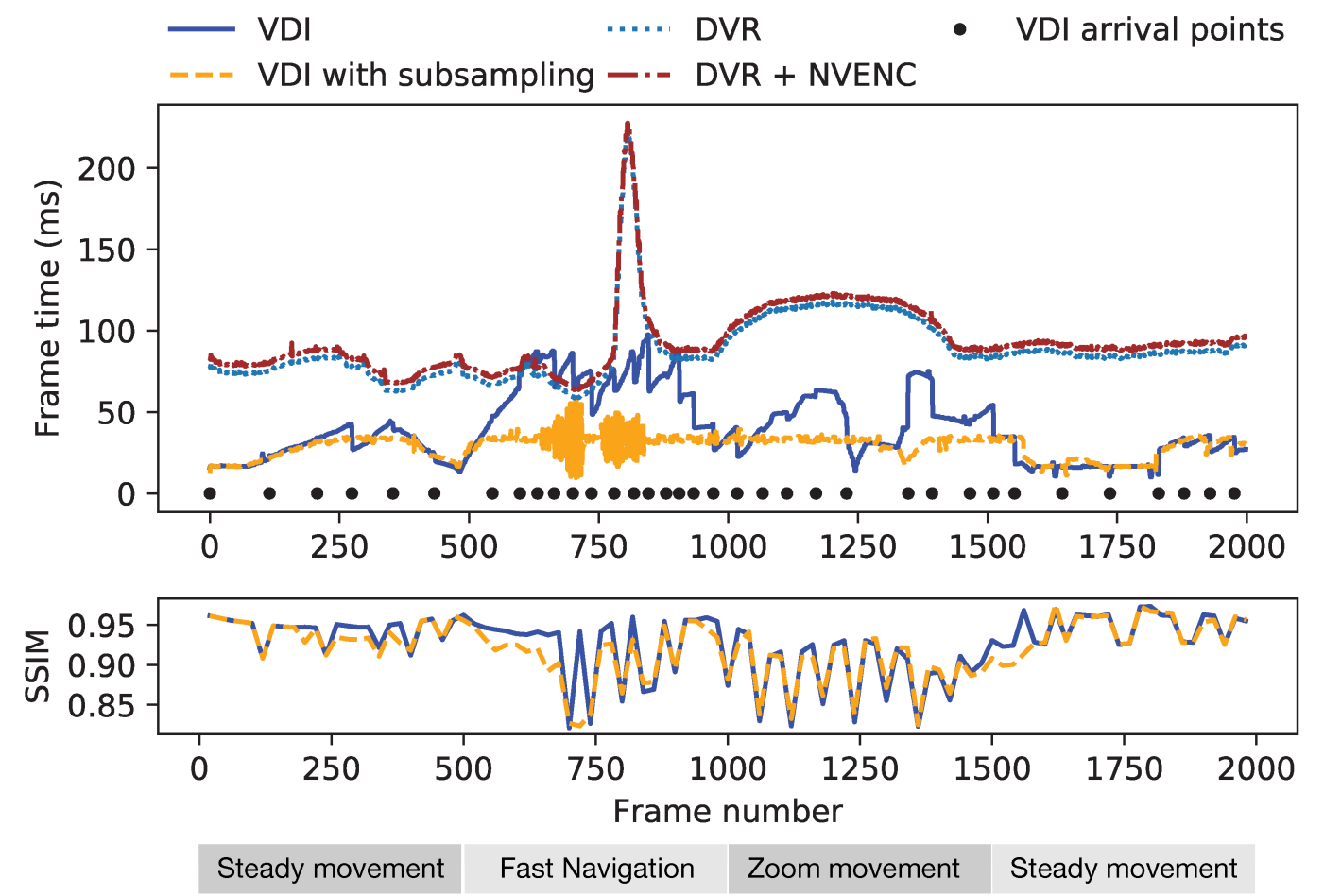

23fps
147fps, SSIM = 0.982, PSNR = 41.44
21fps
137fps, SSIM = 0.969, PSNR = 39.66

At 30° rotation from viewpoint of generation
DVR (32 GPUs) VDI Rendering


24fps
48fps, SSIM = 0.975, PSNR = 34.57
25fps
47fps, SSIM = 0.949, PSNR = 35.31

The end-to-end setup

VDI generation, streaming, and rendering provides lower latency, i.e. frame time, compared to remote volume rendering with hardware-accelerated video encoding.



References

[1] A. Gupta, P. Incardona, A. Brock, G. Reina, S. Frey, S. Gumhold, U. Günther, I. F. Sbalzarini. **Parallel compositing of Volumetric Depth Images for interactive visualization of distributed volumes at high frame rates.** *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV) 2023.*

[2] A. Gupta, P. Incardona, S. Gumhold, U. Günther, and I. F. Sbalzarini. **An architecture for interactive in situ visualization and its transparent implementation in OpenFPM.** *ISAV 2020 (co-located with SC).*

[3] A. Gupta, U. Günther, P. Incardona, G. Reina, S. Frey, S. Gumhold, I. F. Sbalzarini. **Efficient raycasting of Volumetric Depth Images for interactive visualization of remote volumes at high frame rates.** *IEEE PacificVis 2023.*

[4] U. Günther, T. Pietzsch, A. Gupta, K. I. S. Harrington, P. Tomancak, S. Gumhold, I. F. Sbalzarini. **scenery: Flexible virtual reality visualization on the Java VM.** *IEEE VIS 2019.*

[4] S. Frey, F. Sadlo, T. Ertl. **Explorable Volumetric Depth Images from Raycasting.** *IEEE Conference on Graphics, Patterns and Images 2013.*

[5] Zhou Wang, A. C. Bovik, H. R. Sheikh, E. P. Simoncelli. **Image quality assessment: from error visibility to structural similarity.** *IEEE Transactions on Image Processing 2004.*

[6] P. Incardona, A. Leo, Y. Zaluzhnyi, R. Ramaswamy, I. F. Sbalzarini. **OpenFPM: A Scalable Open Framework for Particle and Particle-Mesh Codes on Parallel Computers.** *Computer Physics Communications 2019.*