Neo: Real-Time On-Device 3D Gaussian Splatting with Reuse-and-Update Sorting Acceleration

Changhun Oh Seongryong Oh Jinwoo Hwang

Yoonsung Kim Hardik Sharma† Jongse Park

KAIST CASYS Lab

† Meta

KAIST Meta ASPLOS

Toward Various Immersive Experiences

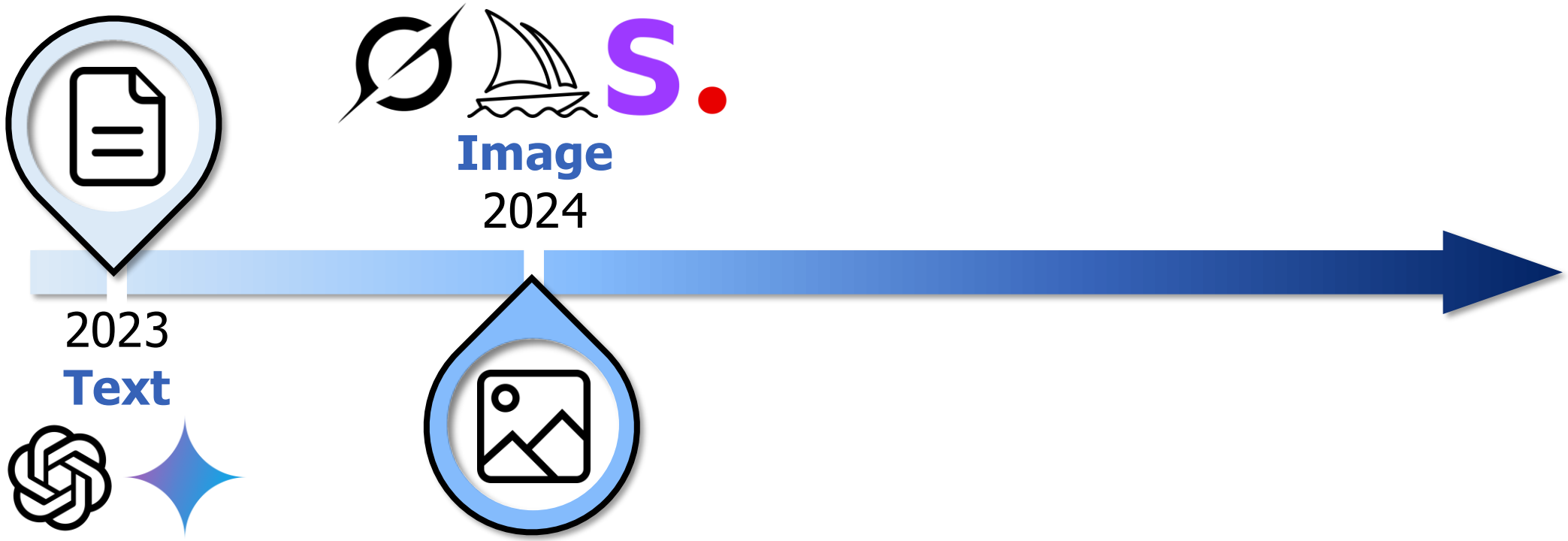
Generative AI has been transforming the way

we access and interact with a wide range of modalities

Toward Various Immersive Experiences

Generative AI has been transforming the way

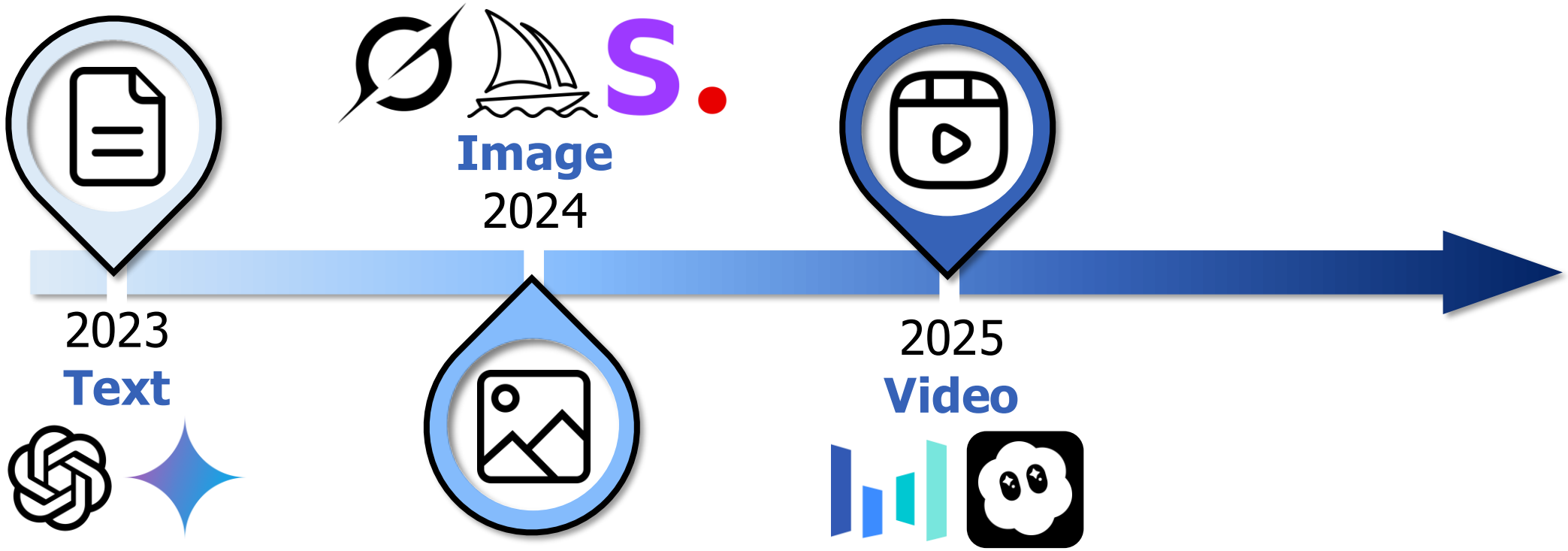
we access and interact with a wide range of modalities



Toward Various Immersive Experiences

Generative AI has been transforming the way

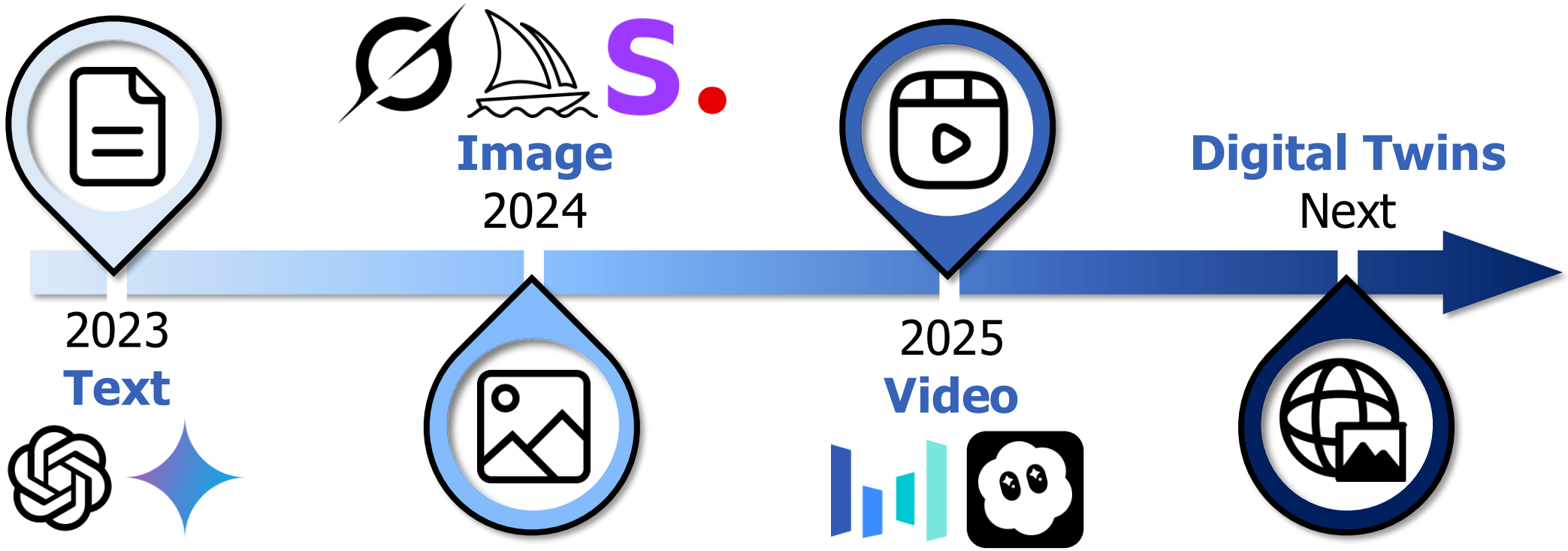
we access and interact with a wide range of modalities



Toward Various Immersive Experiences

Generative AI has been transforming the way

we access and interact with a wide range of modalities



Toward Various Immersive Experiences

Generative AI has been transforming the way

we access and interact with a wide range of modalities

Main Question:

How to reconstruct 3D environments?

Digital Twins

Next

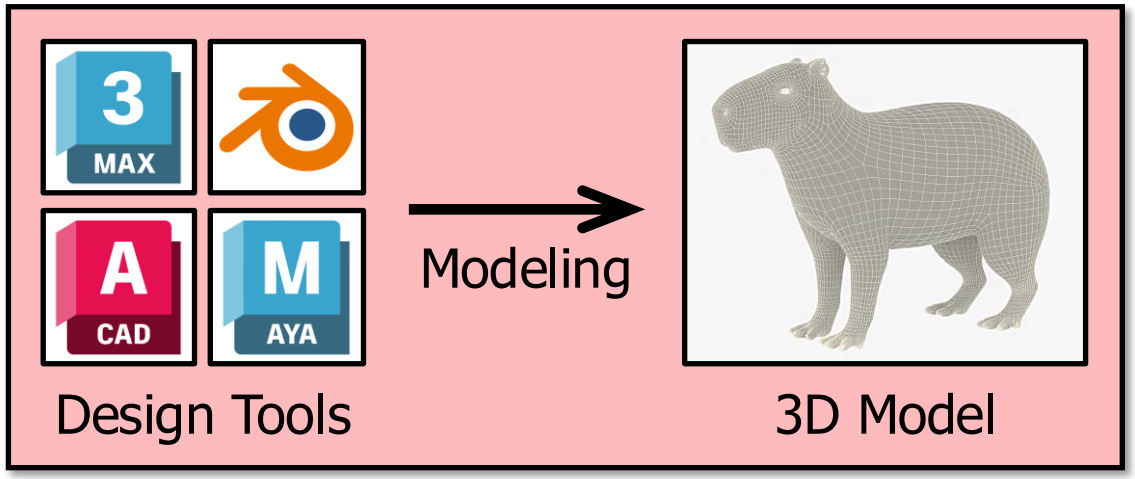
3D Gaussian Splatting for Digital Twins

- 3DGS as a key enabler for 3D reconstruction in digital twins

3D Gaussian Splatting for Digital Twins

- 3DGS as a key enabler for 3D reconstruction in digital twins

Classical Computer Graphics



Expert 3D designer

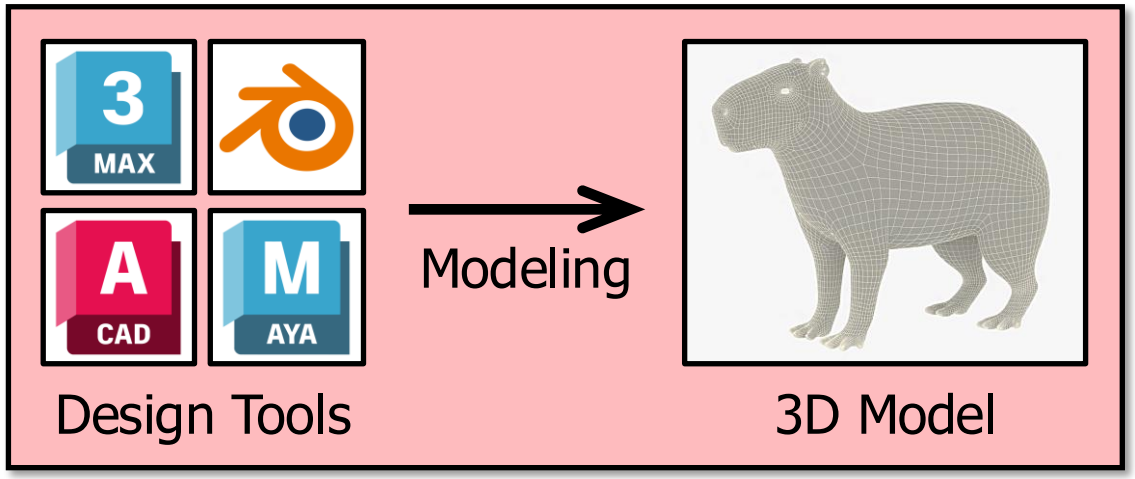
☹ **Time-consuming** asset creation (days)

☹ **High** content creation **cost**

3D Gaussian Splatting for Digital Twins

- 3DGS as a key enabler for 3D reconstruction in digital twins

Classical Computer Graphics

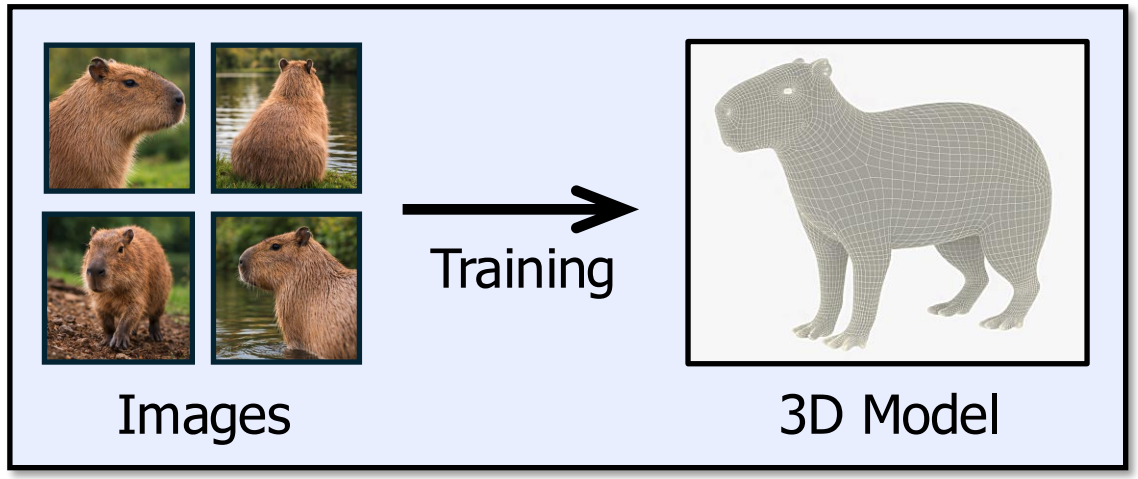


Expert 3D designer

☹ **Time-consuming** asset creation (days)

☹ **High** content creation **cost**

3D Gaussian Splatting (3DGS)



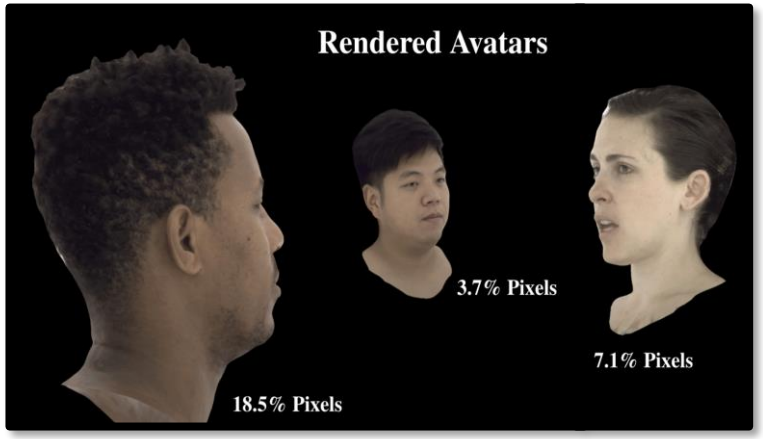
No manual 3D design

☺ **Rapid training** (hours)

☺ **Low** content creation **cost**

On-Device VR with 3D Gaussian Splatting

Telepresence



Virtual Tour

VR Gaming

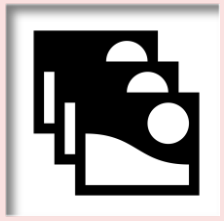
- These applications inherently require **real-time interactivity**
- To deliver seamless experience, **on-device processing becomes essential**

Challenges of On-Device 3DGS Rendering

On-Device VR Demands

High Resolution

QHD resolution per eye



High Frame Rate

At most 120 FPS required

Challenges of On-Device 3DGS Rendering

On-Device VR Demands

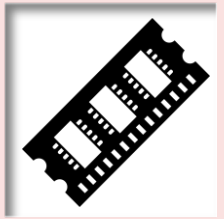
High Resolution

QHD resolution per eye

High Frame Rate

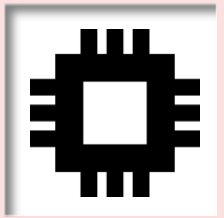
At most 120 FPS required

On-Device Hardware Reality



Limited Memory BW

51.2 – 68.2 GB/s



Limited Computation

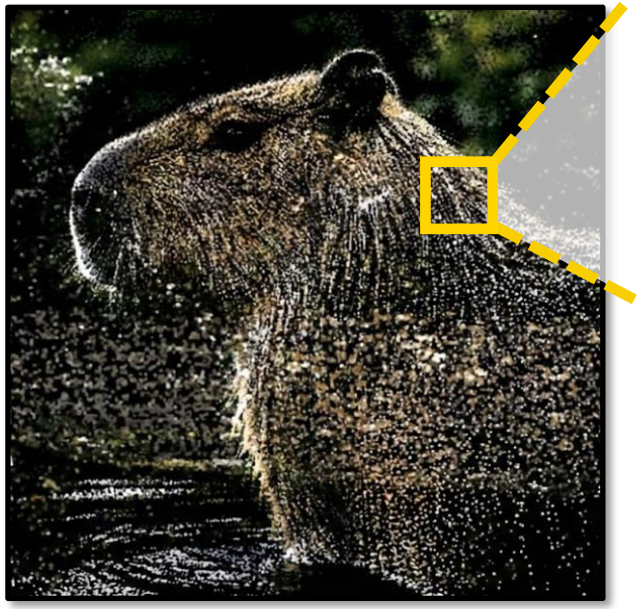
Mobile SoC constraints

Introduction to 3DGS

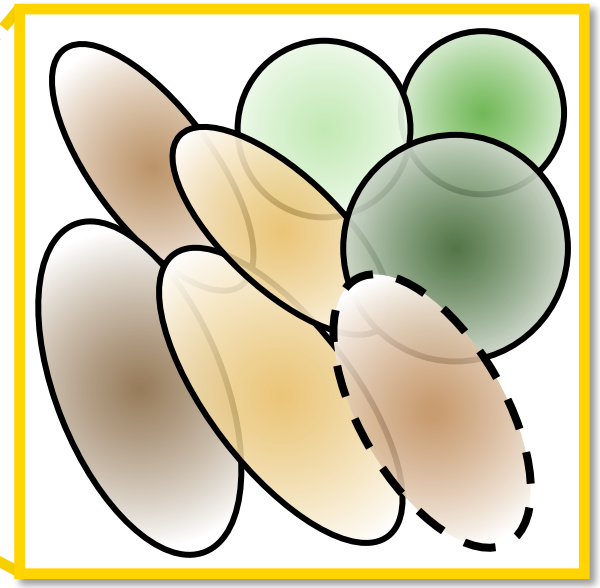
- Scene representation with millions of 3D Gaussians

Images

Train

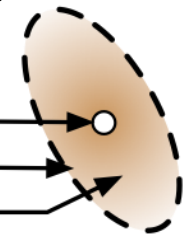


3D Model



3D Gaussians

Position	3D Mean (μ)
Shape	3D Covariance (Σ)
Opacity	Opacity (o)
Color	SH Coefficients (sh)

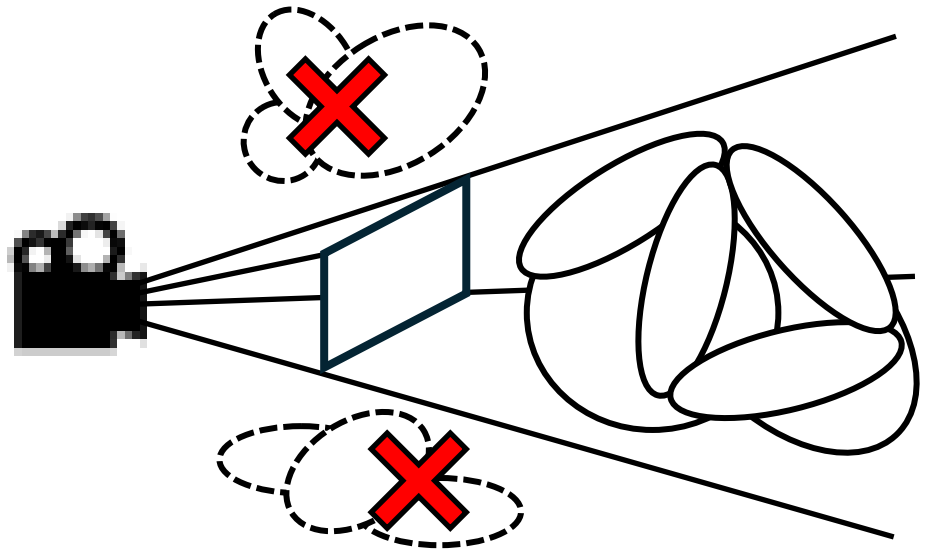


RGB: (203, 150, 92)

Pipeline of 3DGS Rendering

Preprocessing | **Sorting** | **Blending**

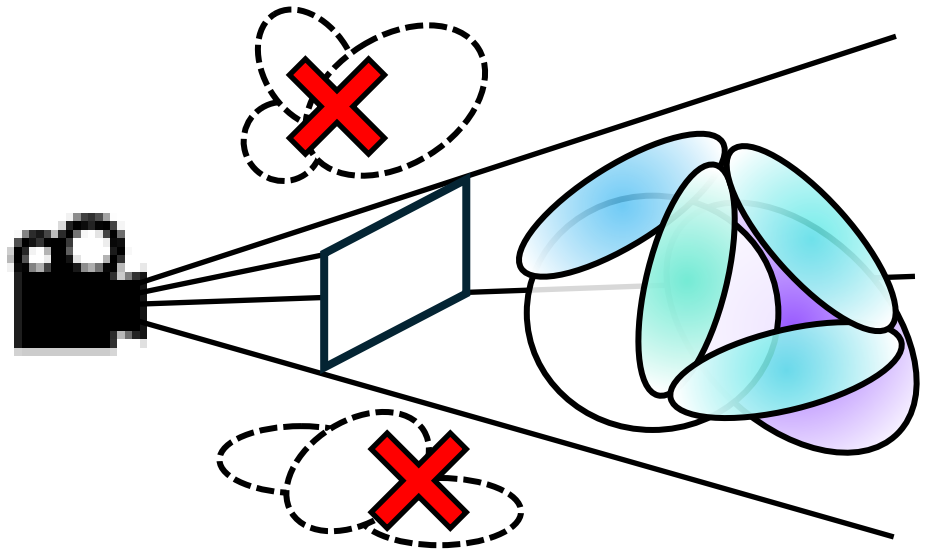
1. Removing Gaussians that are not visible from the current viewpoint



Pipeline of 3DGS Rendering

Preprocessing | **Sorting** | **Blending**

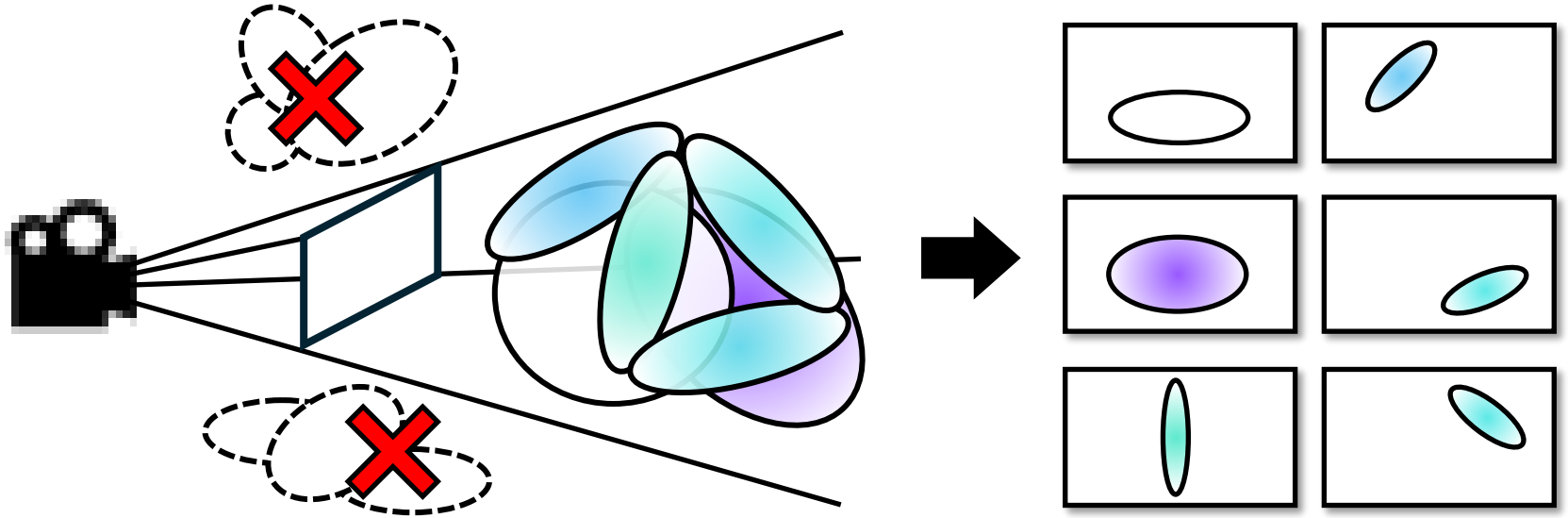
1. Removing Gaussians that are not visible from the current viewpoint
2. **Calculating view-dependent color**



Pipeline of 3DGS Rendering

Preprocessing | **Sorting** | **Blending**

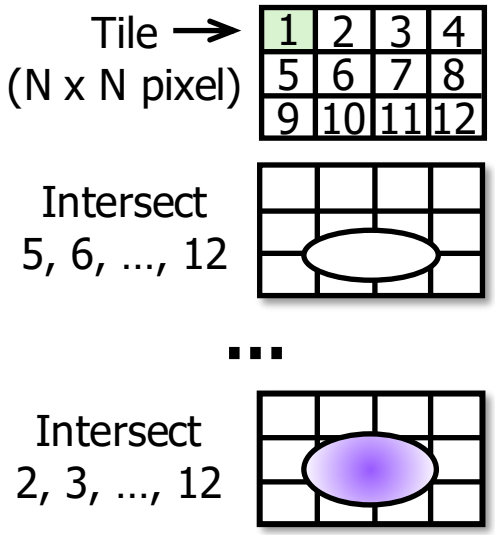
1. Removing Gaussians that are not visible from the current viewpoint
2. Calculating view-dependent color
3. **Projecting Gaussians onto the camera's image plane**



Pipeline of 3DGS Rendering

Preprocessing | Sorting | Blending

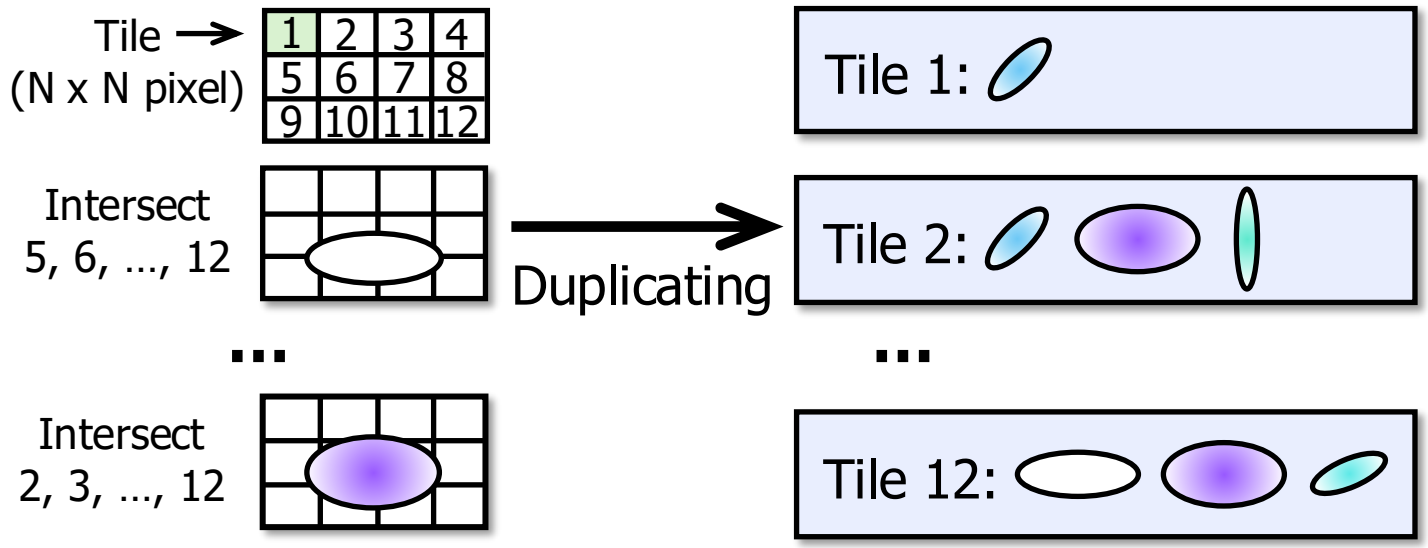
1. Checking Gaussian for every tile it overlaps



Pipeline of 3DGS Rendering

Preprocessing | **Sorting** | **Blending**

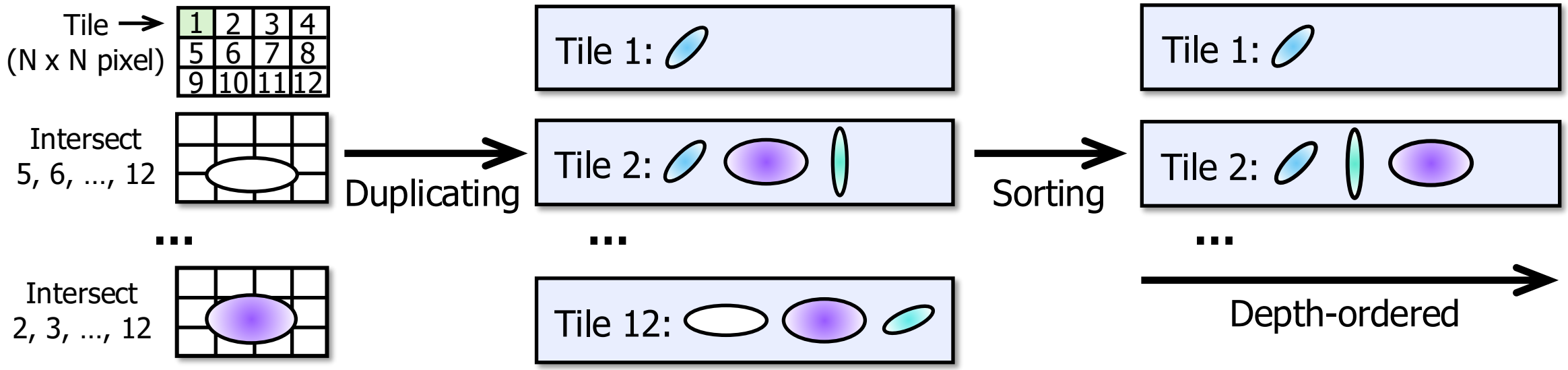
1. Checking Gaussian for every tile it overlaps
2. **Duplicating Gaussian entry for every tile**



Pipeline of 3DGS Rendering

Preprocessing | **Sorting** | **Blending**

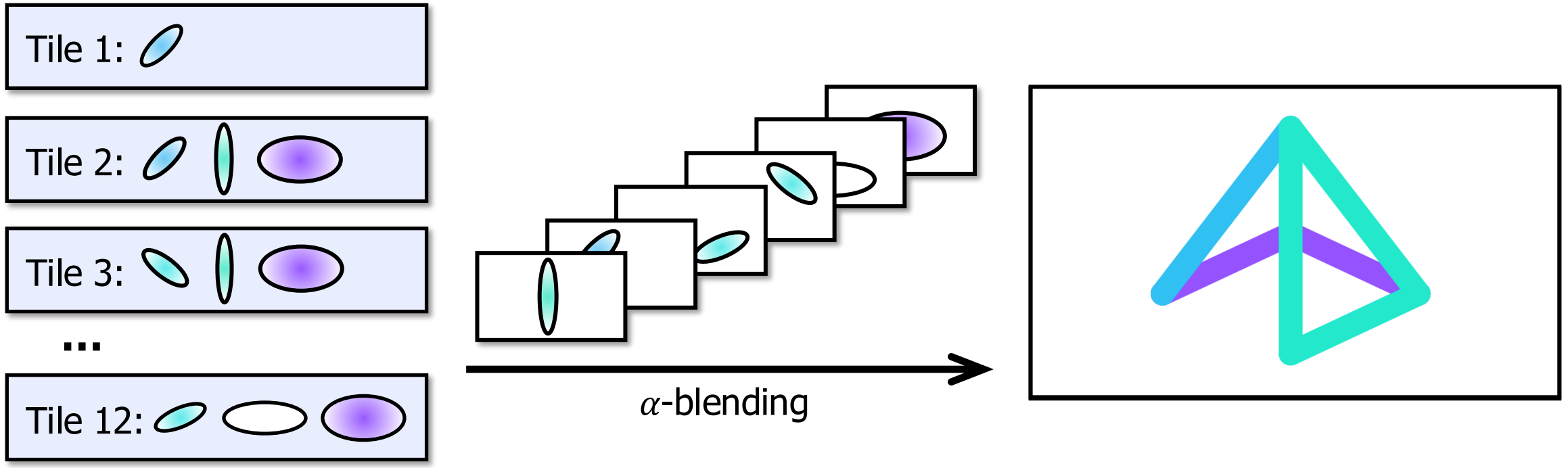
1. Checking Gaussian for every tile it overlaps
2. Duplicating Gaussian entry for every tile
3. **Sorting these entries in depth order for each tile independently**



Pipeline of 3DGS Rendering

Preprocessing | Sorting | Blending

1. α-blending–based pixel computation with depth-sorted Gaussians



Performance Characterization of 3DGS

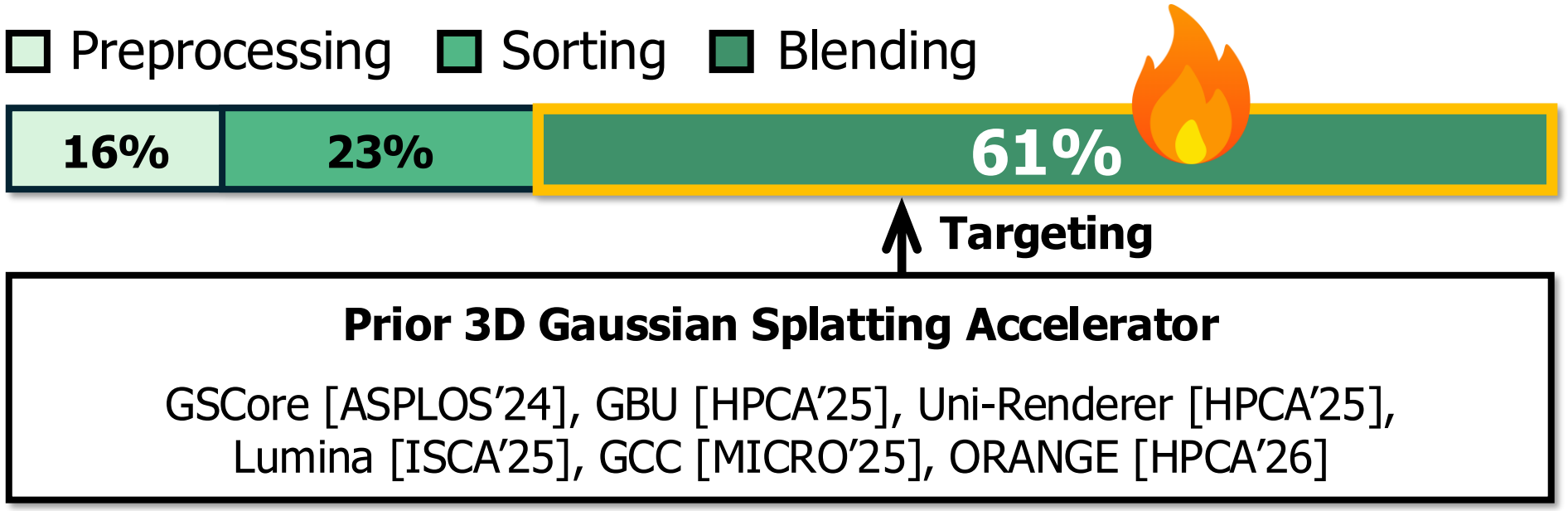
- Empirical analysis based on a representative on-device: Jetson Orin AGX

HD: 22 FPS

Performance Characterization of 3DGS

- Empirical analysis based on a representative on-device: Jetson Orin AGX

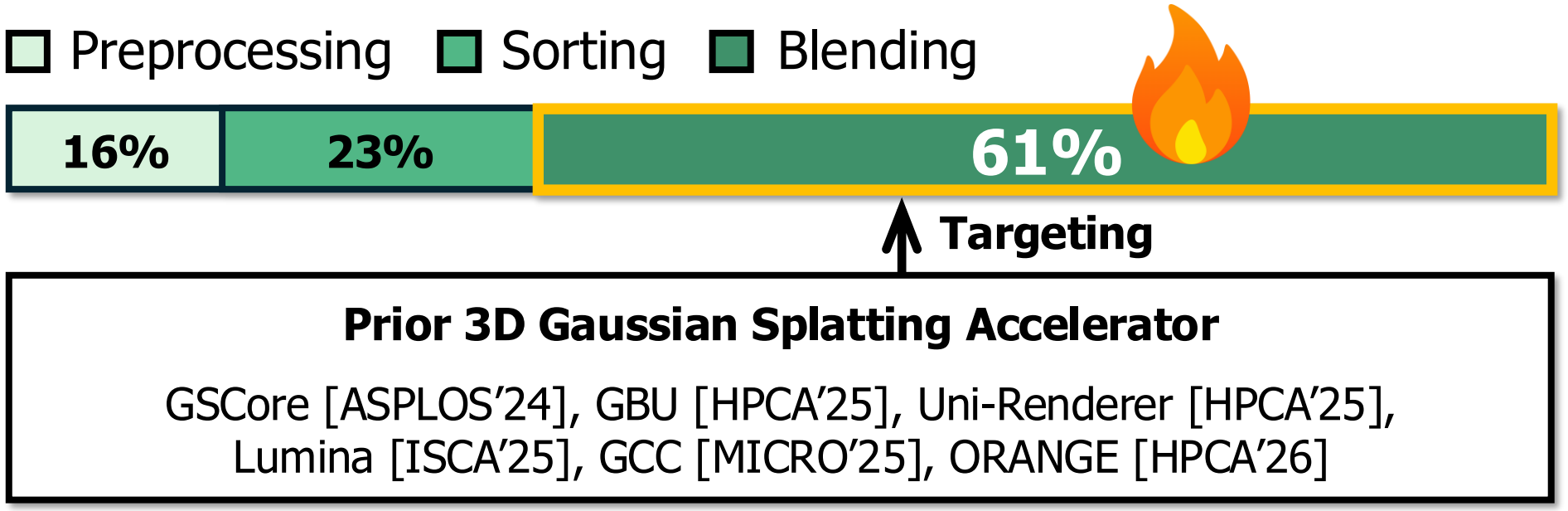
HD: 22 FPS



Performance Characterization of 3DGS

- Empirical analysis based on a representative on-device: Jetson Orin AGX

HD: 22 FPS

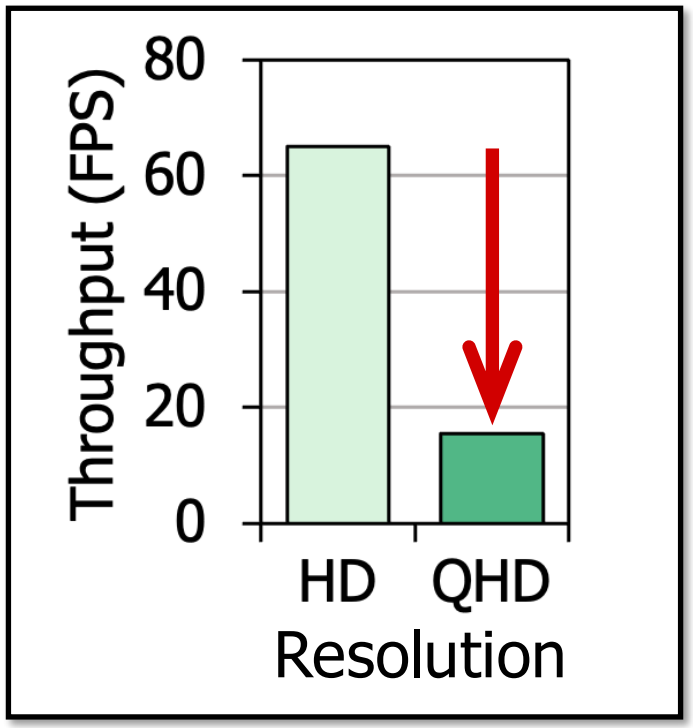


Are existing solutions sufficient under stricter on-device constraints?

(1) High Resolution (2) High Frame-Rate (3) Limited Memory Bandwidth

Performance Characterization of 3DGS

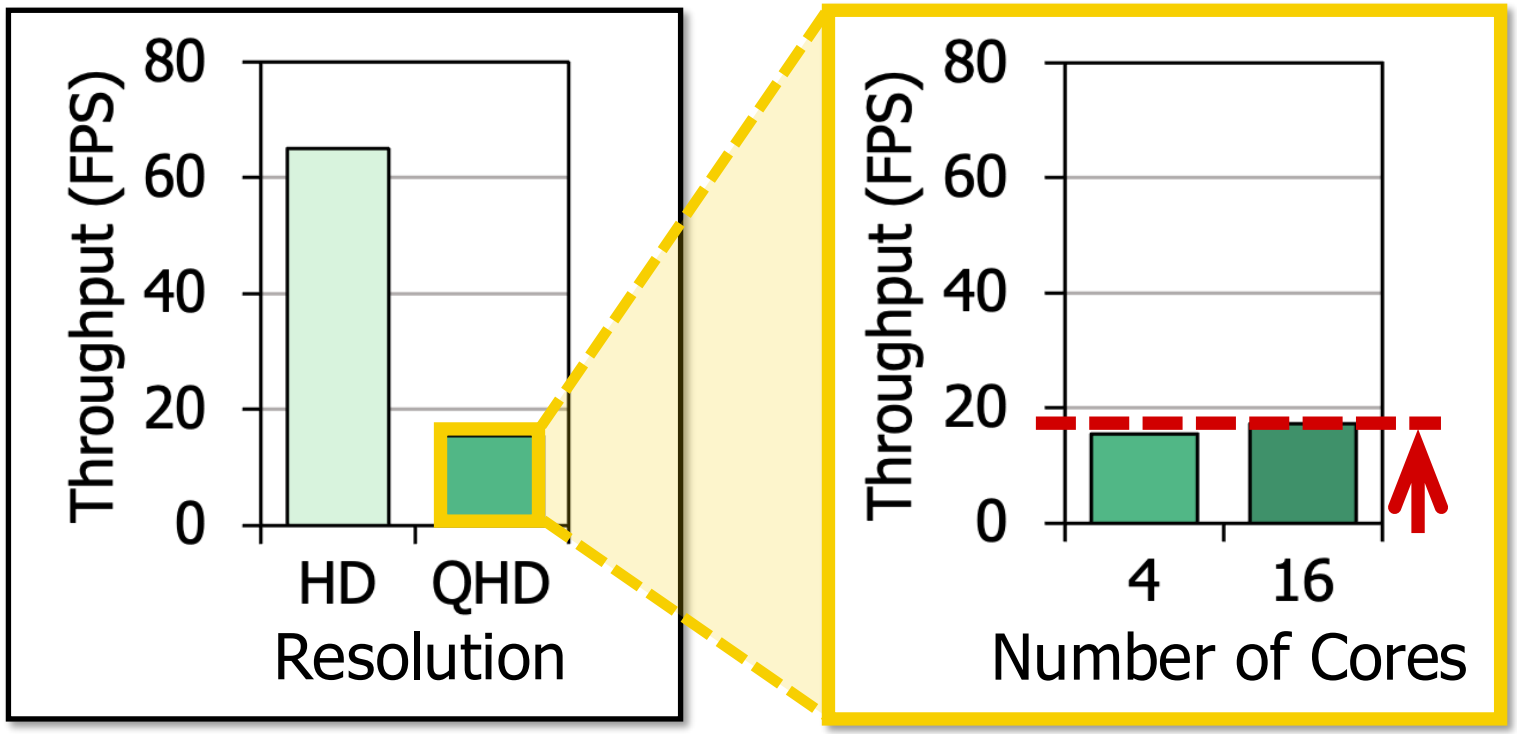
- Empirical analysis based on an existing 3DGS accelerator: GSCore[1]



Reduced performance with increasing resolution

Performance Characterization of 3DGS

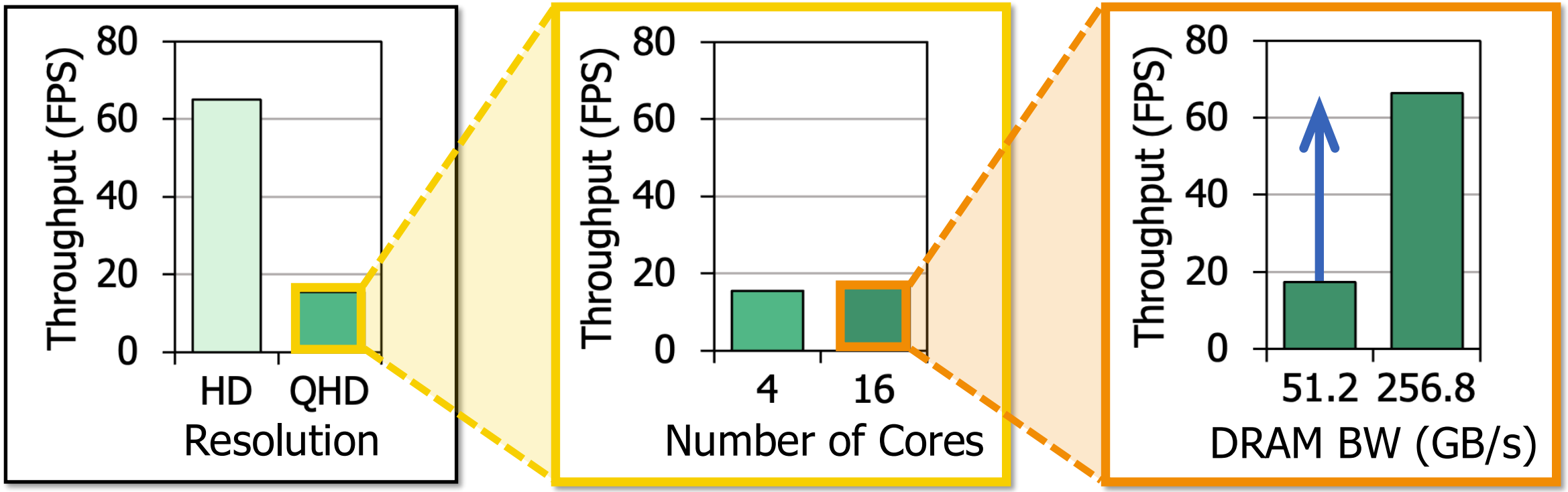
- Empirical analysis based on an existing 3DGS accelerator: GSCore[1]



Marginal gains from core scaling

Performance Characterization of 3DGS

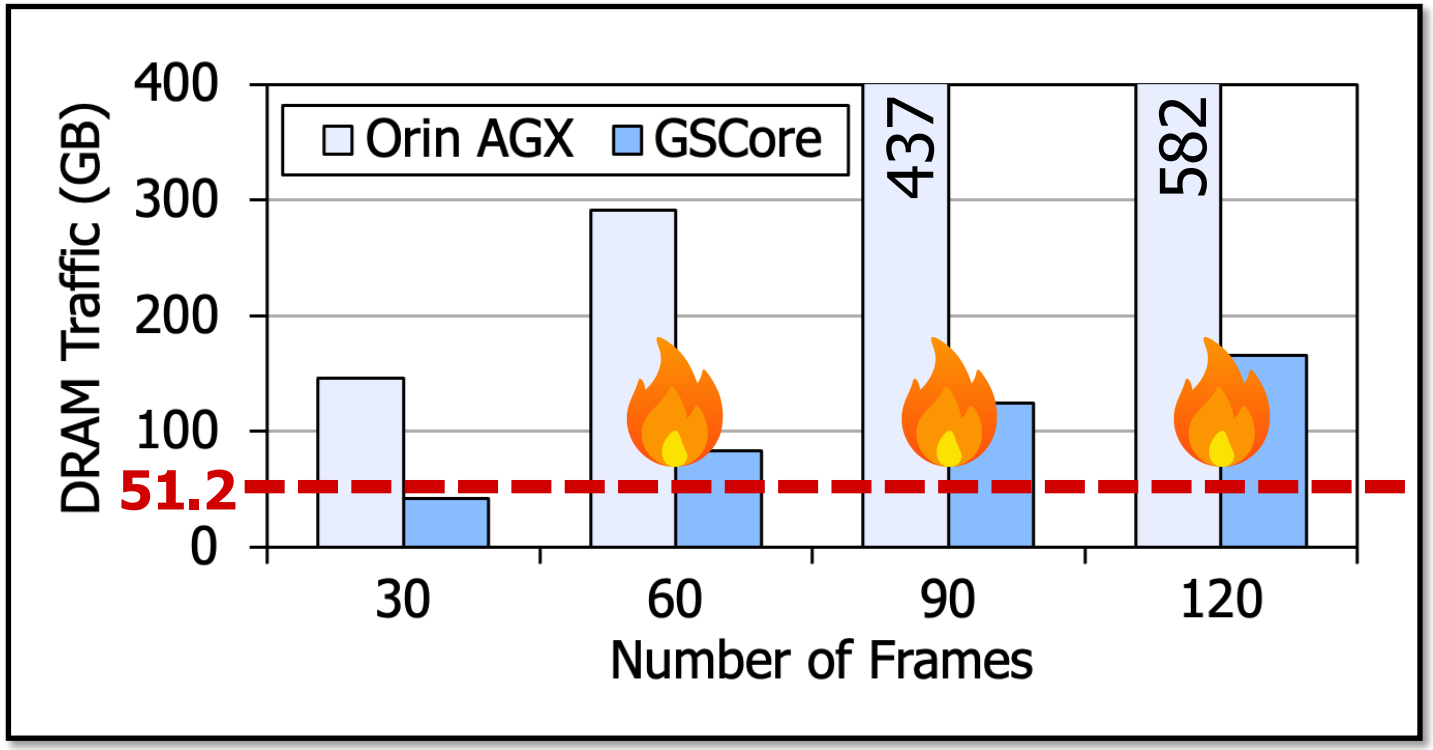
- Empirical analysis based on an existing 3DGS accelerator: GSCore[1]



Higher DRAM bandwidth as a prerequisite for core-scaling gains

Performance Characterization of 3DGS

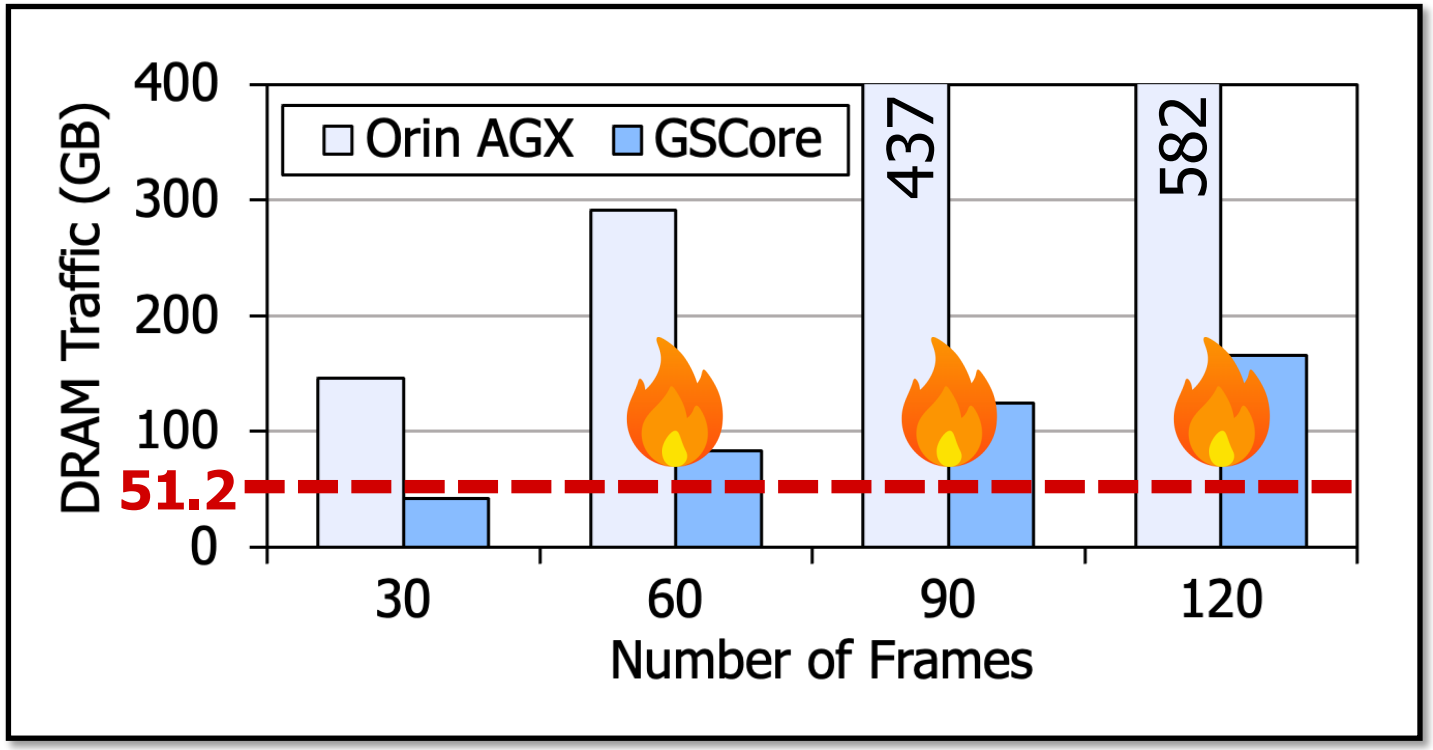
- **Excessive DRAM traffic** as the main cause of this behavior

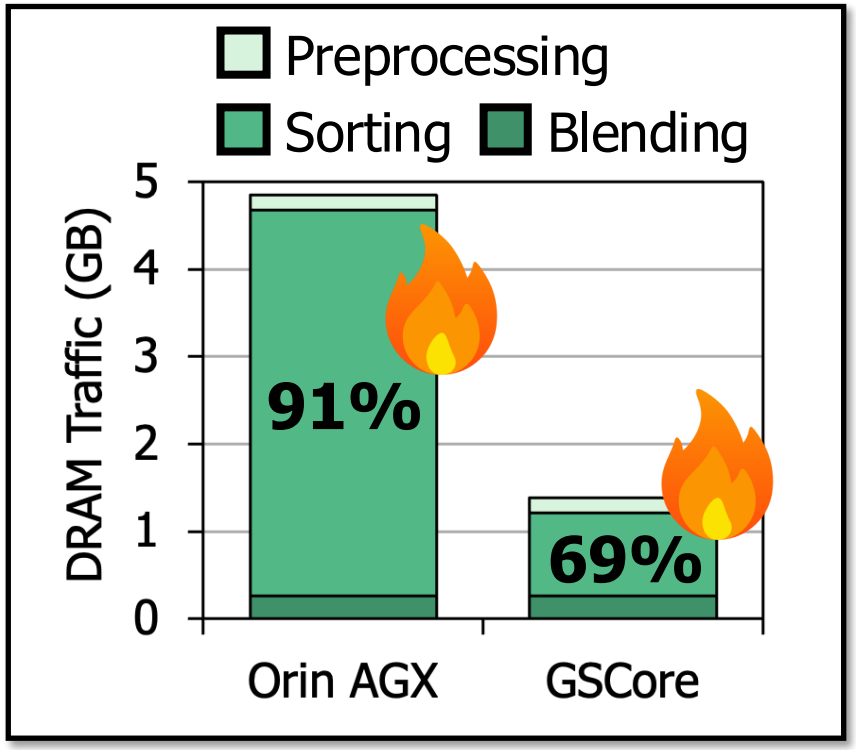


Frame-rate limitation by low DRAM bandwidth at high resolution

Performance Characterization of 3DGS

- **Excessive DRAM traffic** as the main cause of this behavior

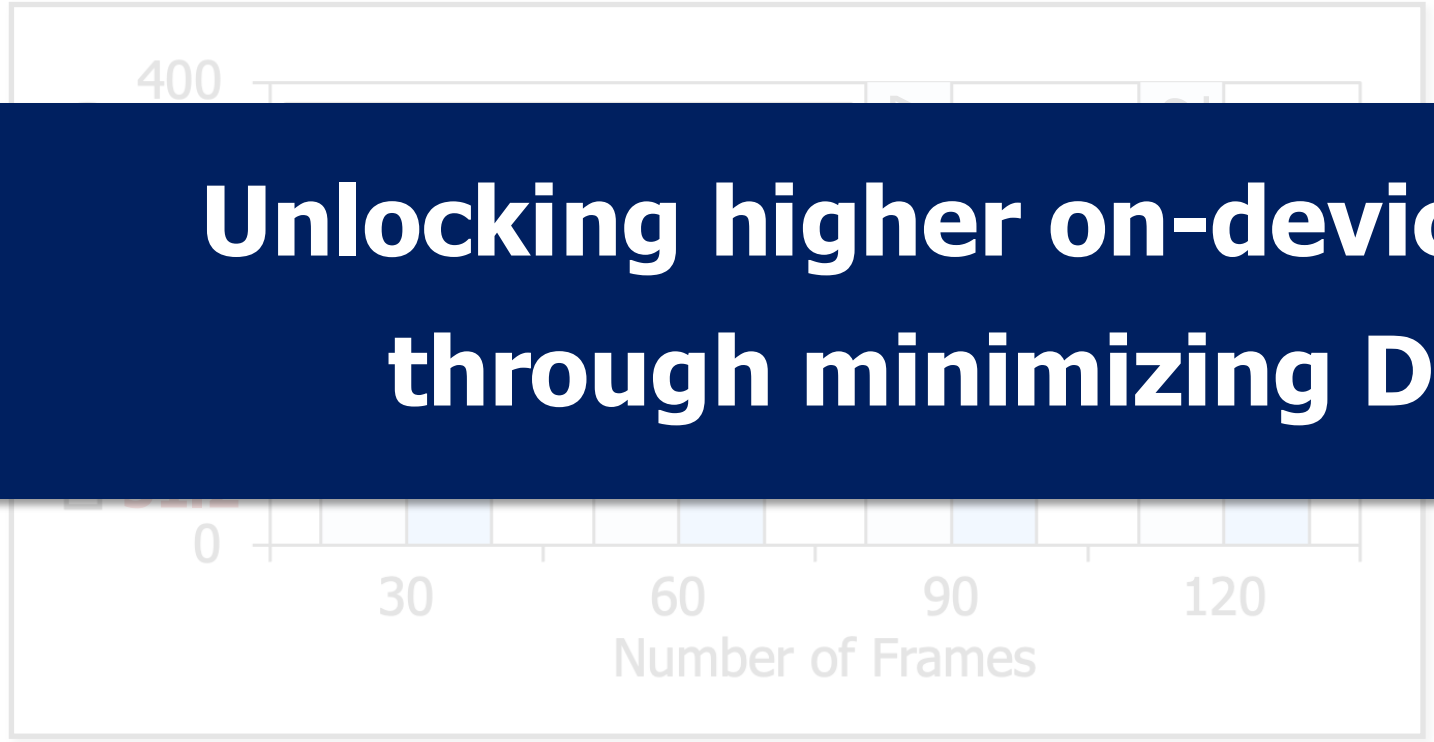


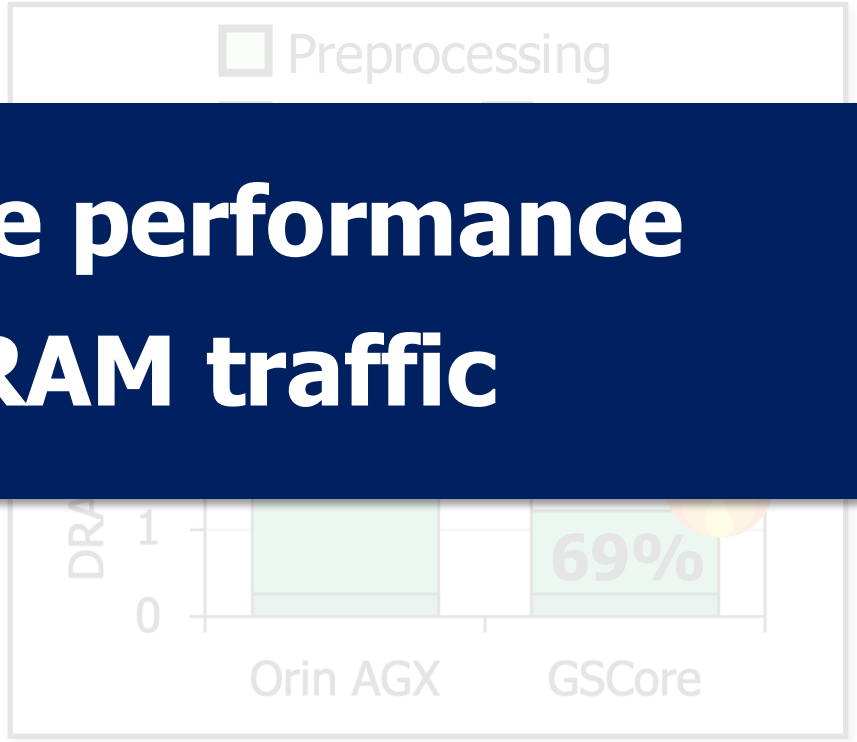


Sorting as the dominant source of DRAM traffic

Performance Characterization of 3DGS

- **Excessive DRAM traffic** as the main cause of this behavior



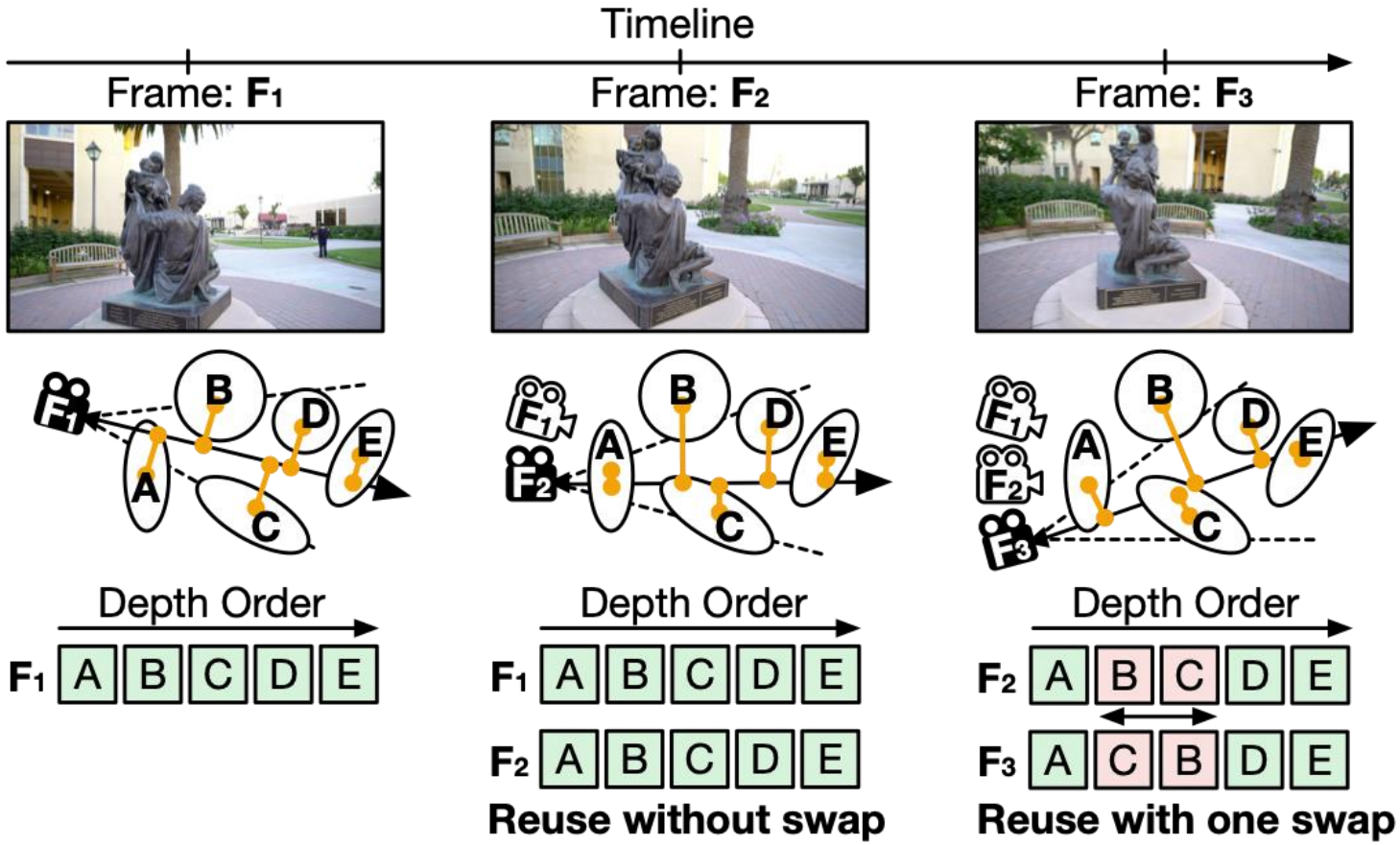


Unlocking higher on-device performance through minimizing DRAM traffic

Sorting as the dominant source of DRAM traffic

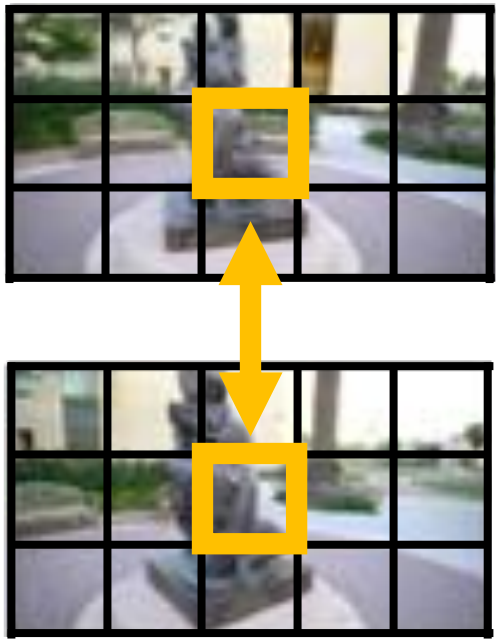
Opportunity to Reduce Memory Traffic

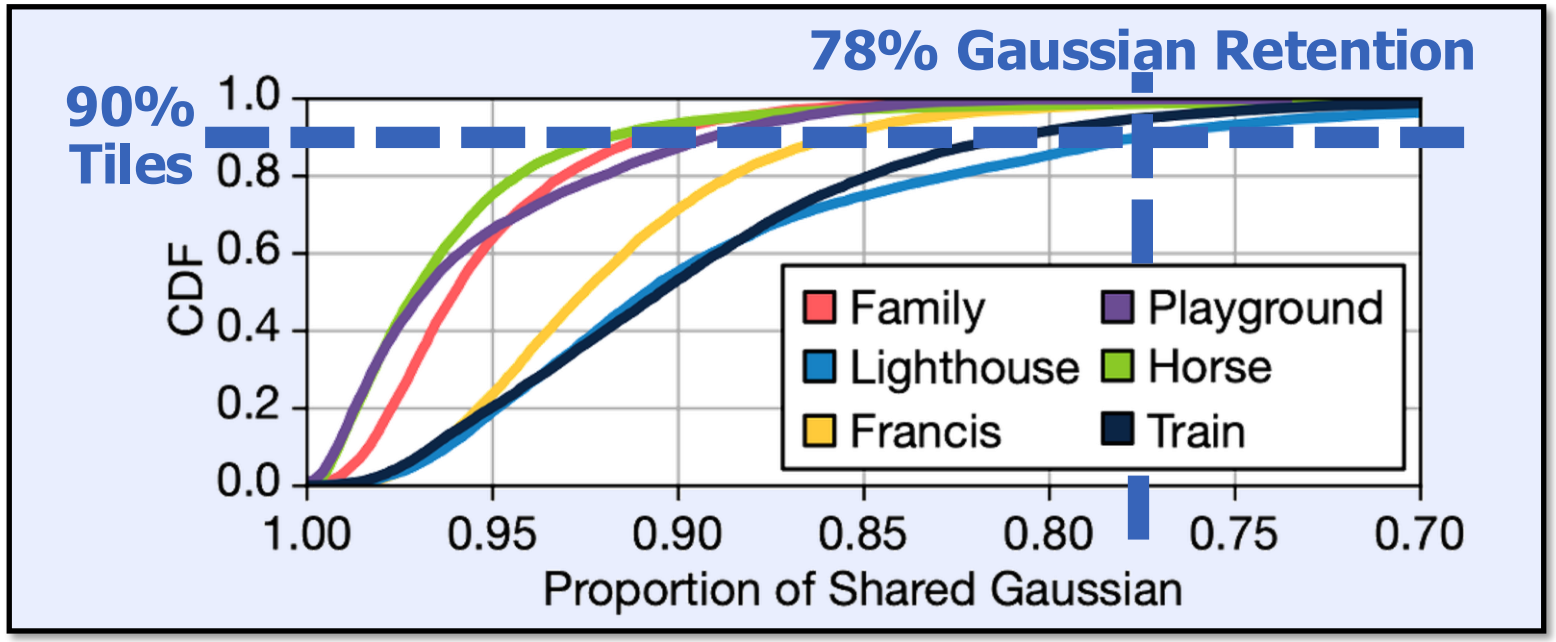
- The key insight: temporal similarity of sorting across frames



Opportunity to Reduce Memory Traffic

- Question 1: Per tile, how much can we reuse from the previous frame?

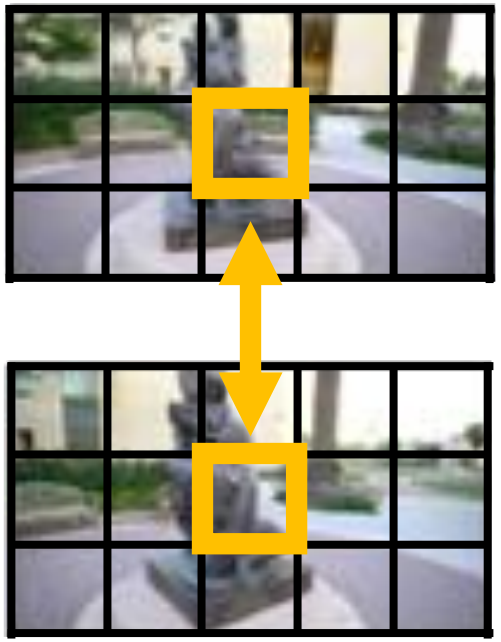


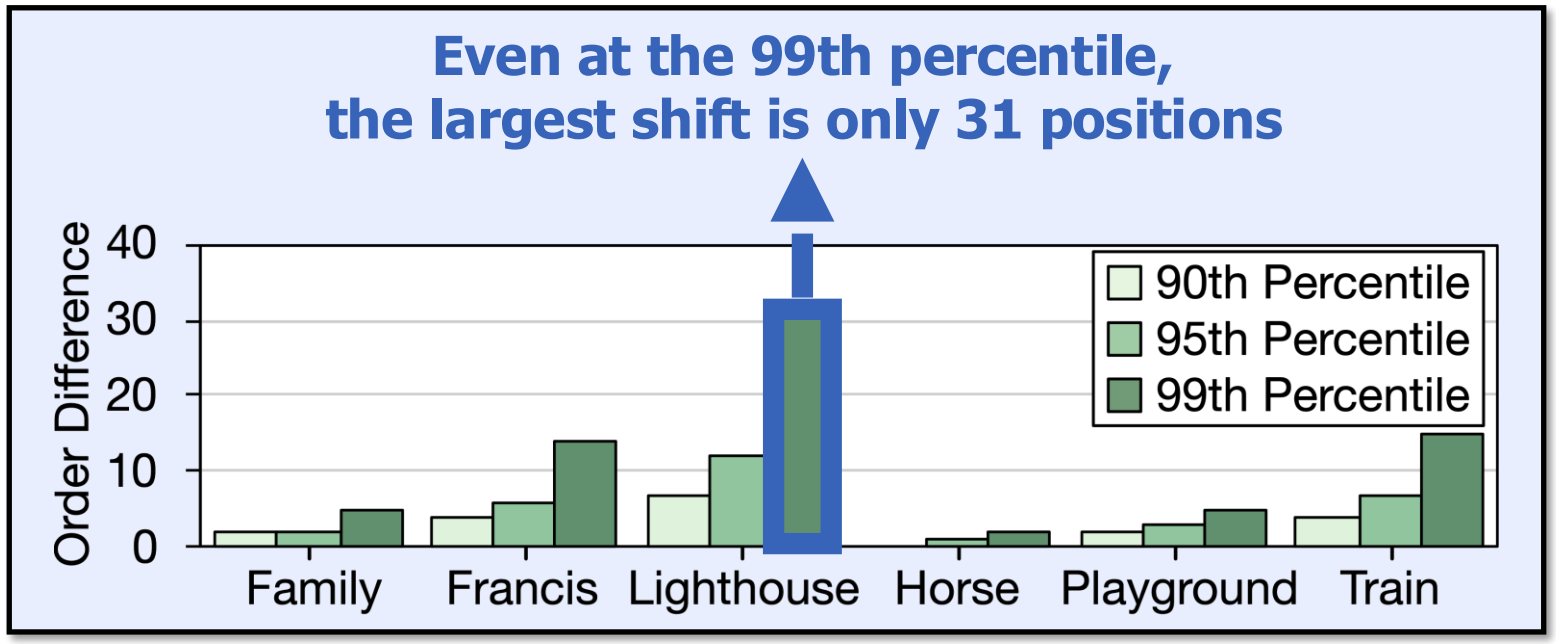


1. Over 78% Gaussian retention in 90% of tiles

Opportunity to Reduce Memory Traffic

- Question 2: Per tile, how much does depth order shift between frames?





1. **Over 78% Gaussian retention in 90% of tiles**
2. **High sorting-order consistency in 99% of cases**

Opportunity to Reduce Memory Traffic

- Question 2: Per tile, how much does depth order shift between frames?

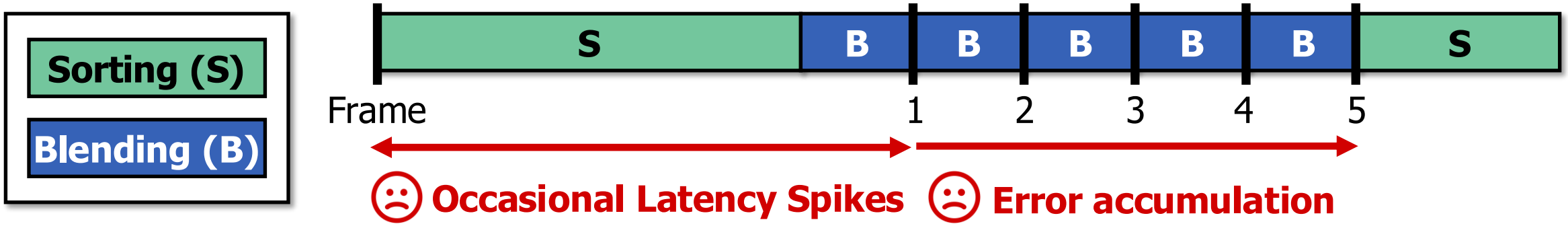
Strong opportunity to reuse sorting across frames

→ How to exploit this opportunity?

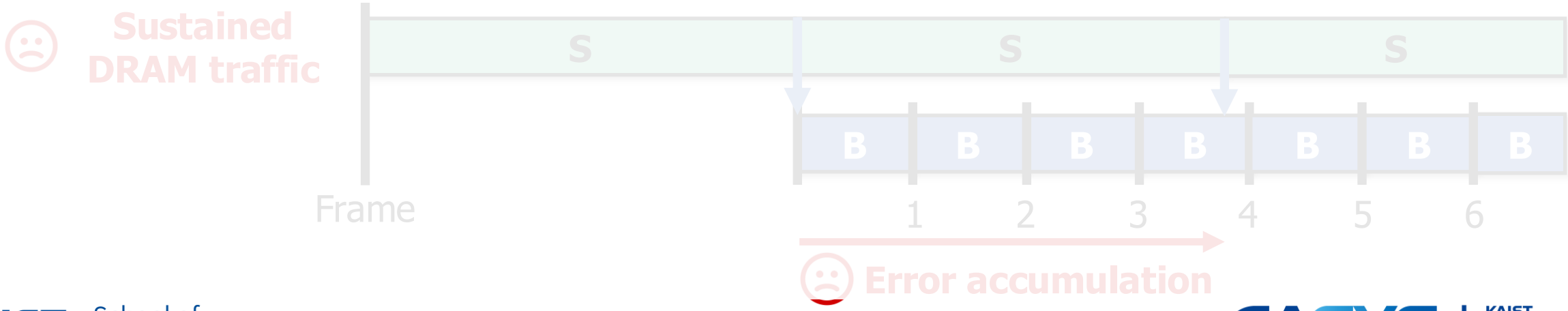
1. **Over 78% Gaussian retention in 90% of tiles**
2. **High sorting-order consistency in 99% of cases**

Exploring Sorting Reuse Strategies

- Periodic sorting: sorting at intervals, reusing intermediate frames



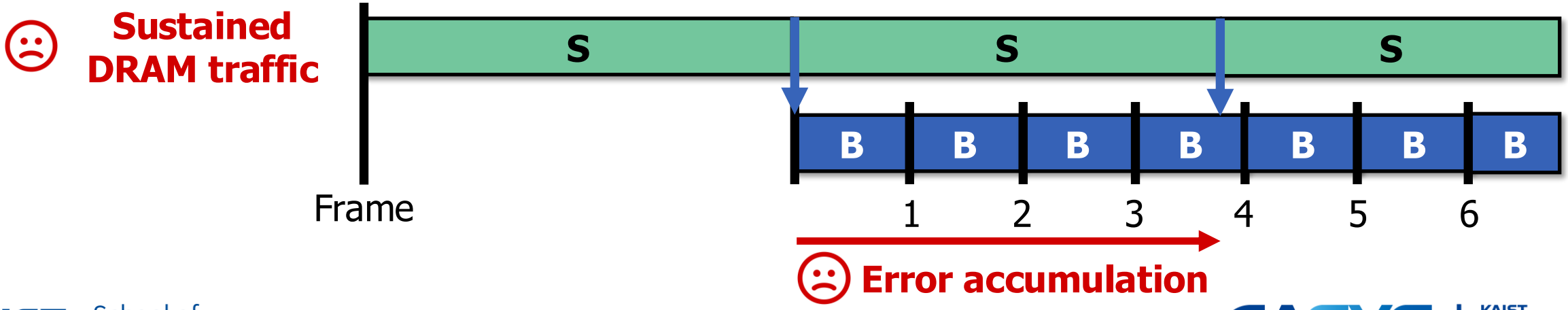
- Background sorting: sorting in parallel with rendering



Exploring Sorting Reuse Strategies

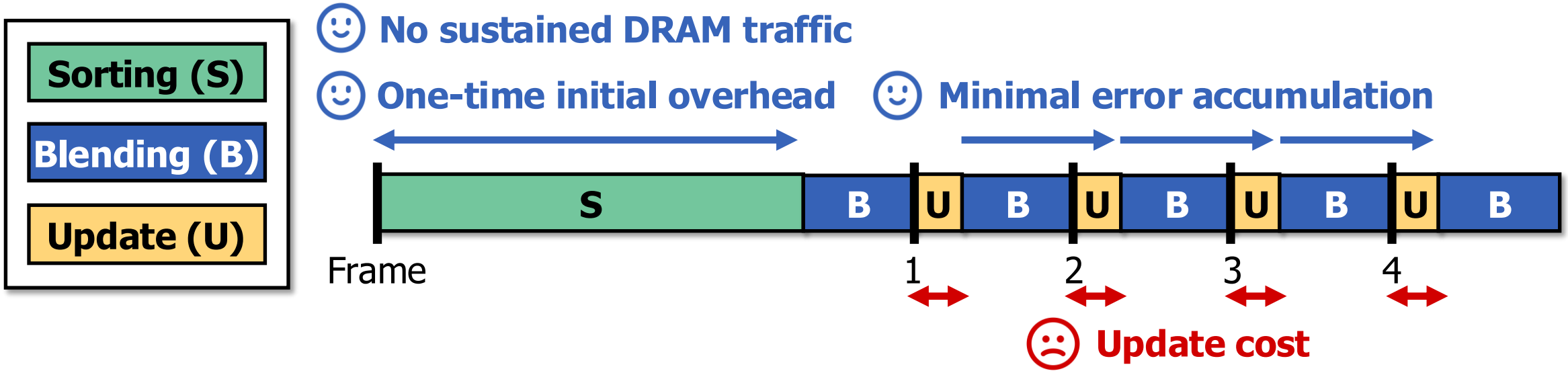
- Periodic sorting: sorting at intervals, reusing intermediate frames

- Background sorting: sorting in parallel with rendering



Exploring Sorting Reuse Strategies

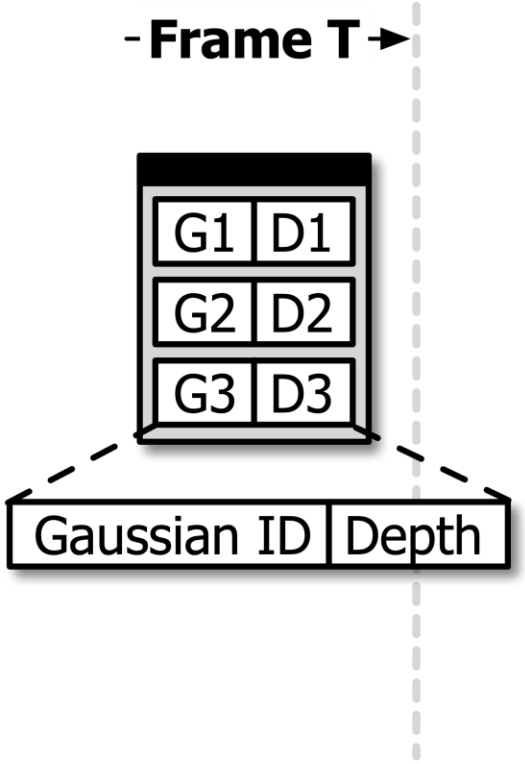
- Incremental Sorting: reusing prior sorting with fine-grained updates



Flow of Reuse-and-Update Sorting

Reordering → **Insertion** → **Deletion** → **Depth Update**

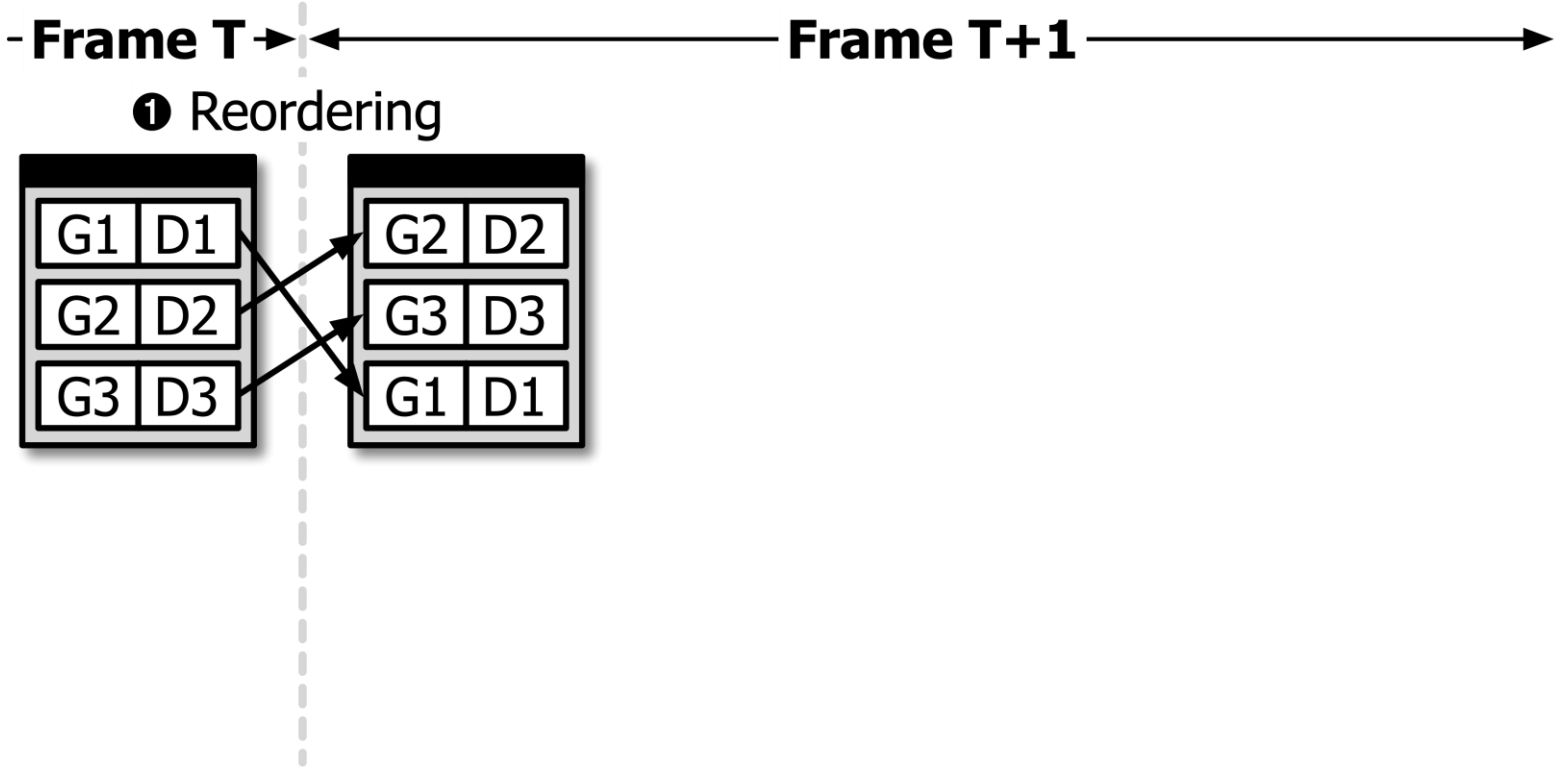
- Four steps to produce the sorted table for the current frame



Flow of Reuse-and-Update Sorting

Reordering | **Insertion** | **Deletion** | **Depth Update**

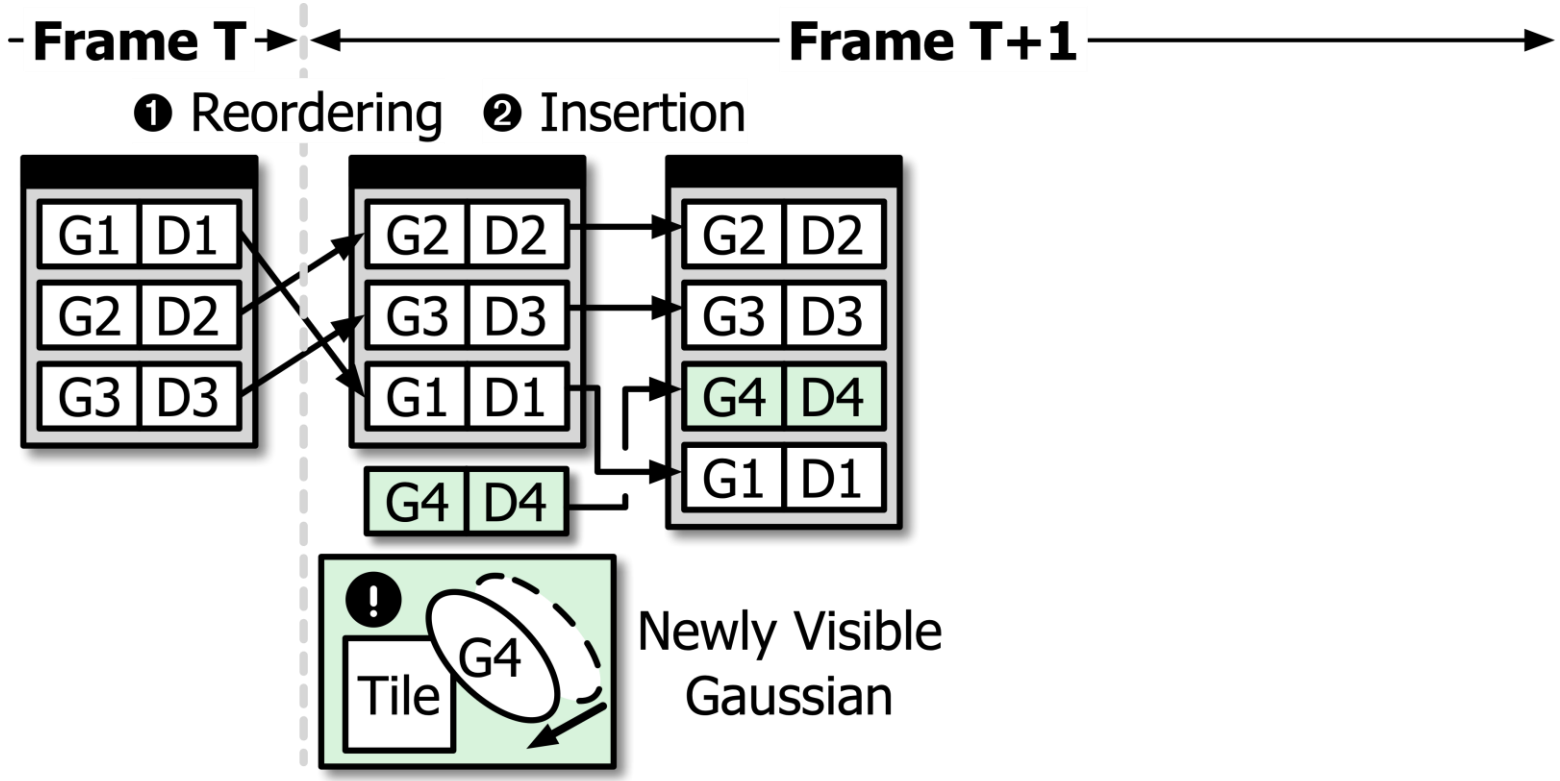
1. Rearrangement of existing entries to match the new depth order



Flow of Reuse-and-Update Sorting

Reordering | **Insertion** | Deletion | Depth Update

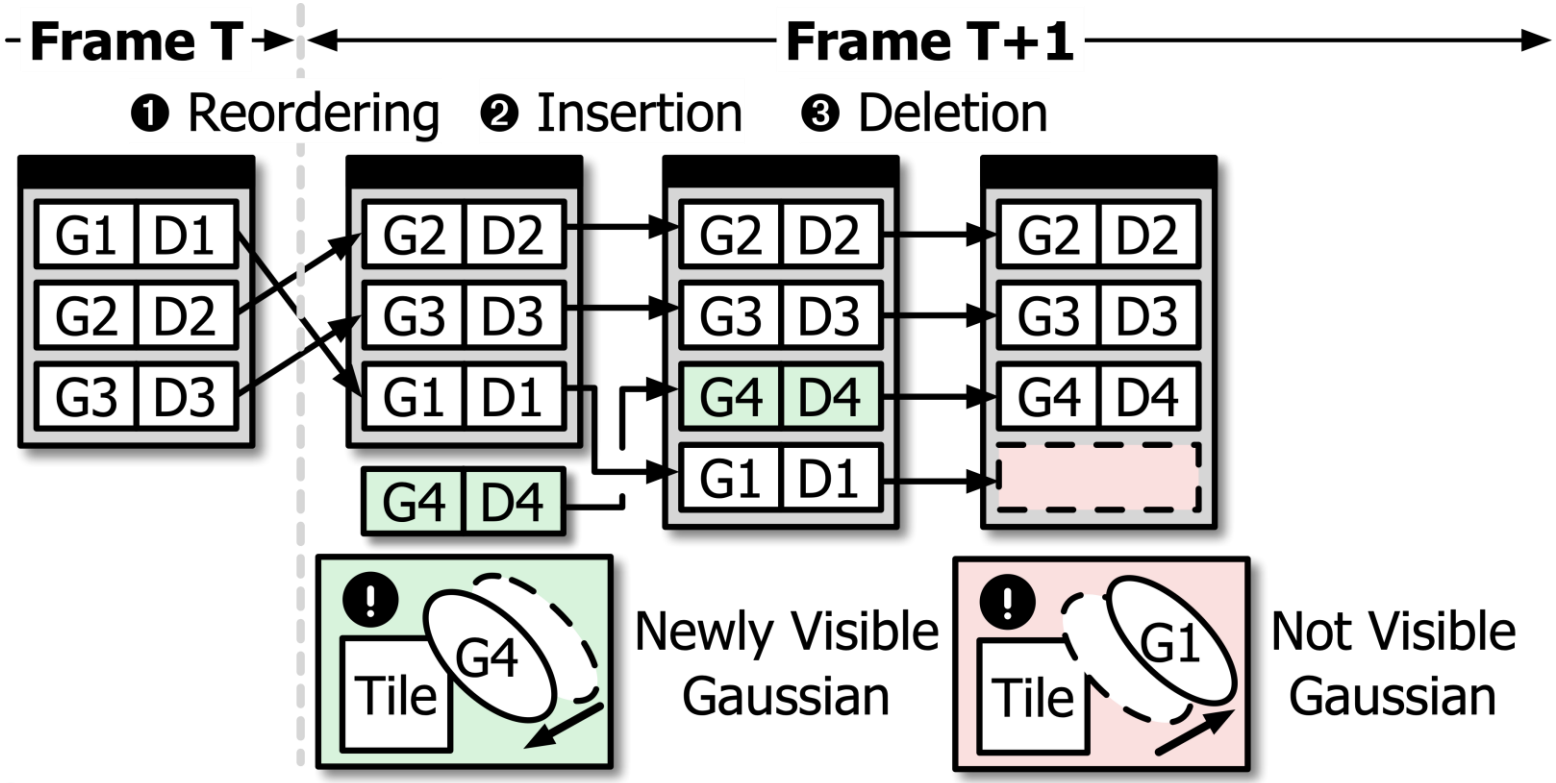
2. Insertion of newly visible Gaussians into the table



Flow of Reuse-and-Update Sorting

Reordering | Insertion | **Deletion** | Depth Update

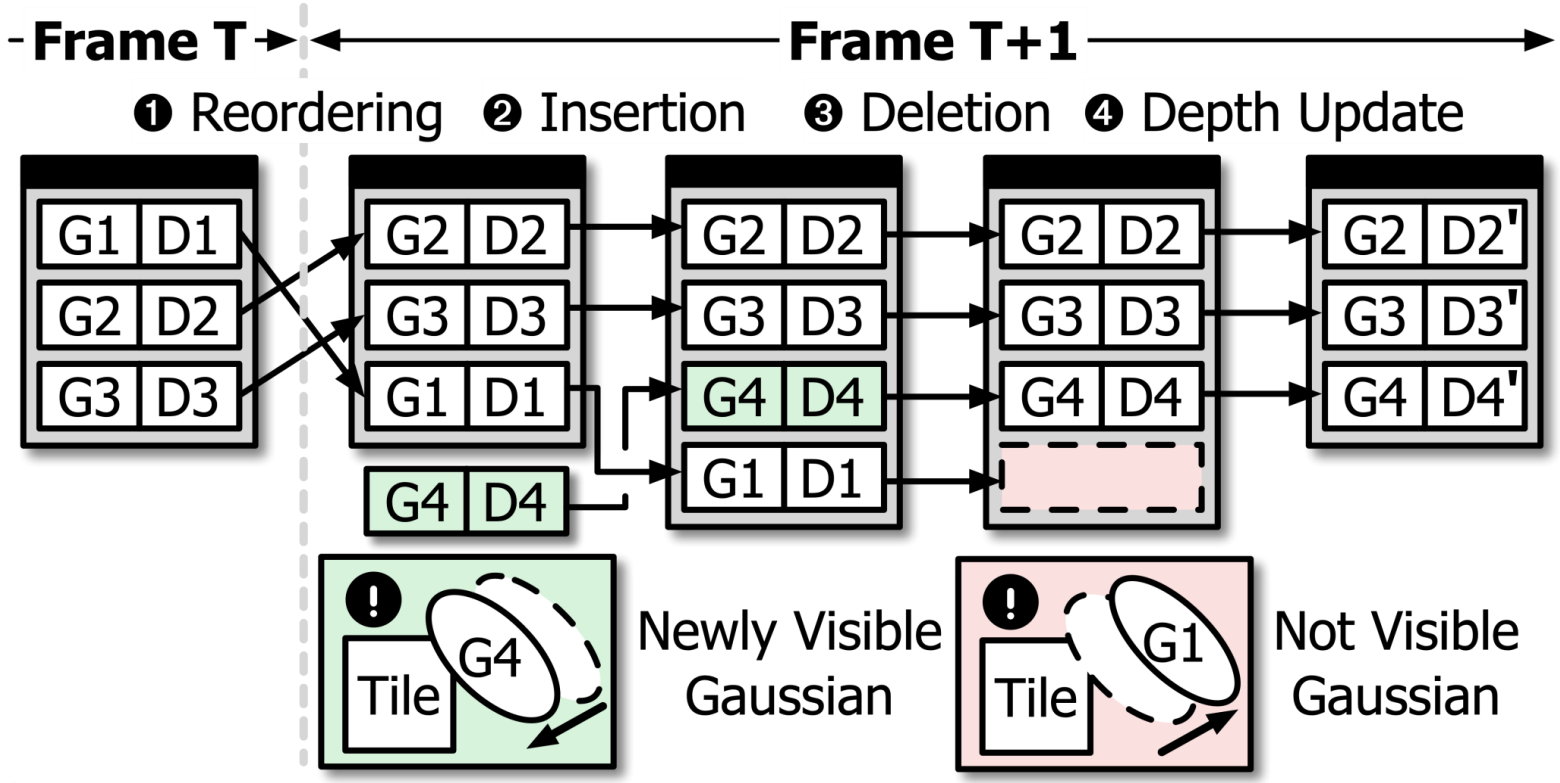
3. Deletion of Gaussians that are no longer visible



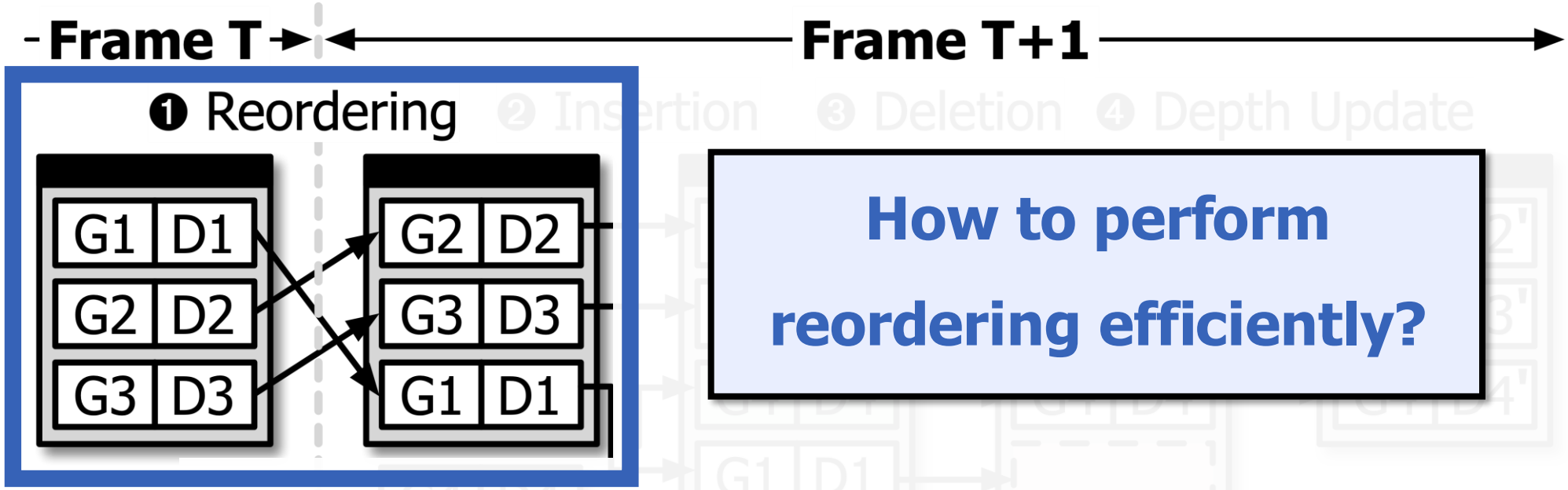
Flow of Reuse-and-Update Sorting

Reordering | Insertion | Deletion | **Depth Update**

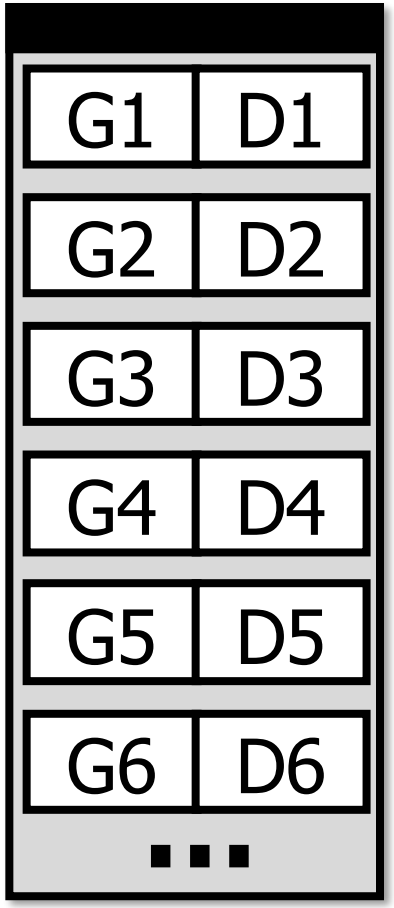
4. Refresh of the depth values of all remaining entries



Flow of Reuse-and-Update Sorting



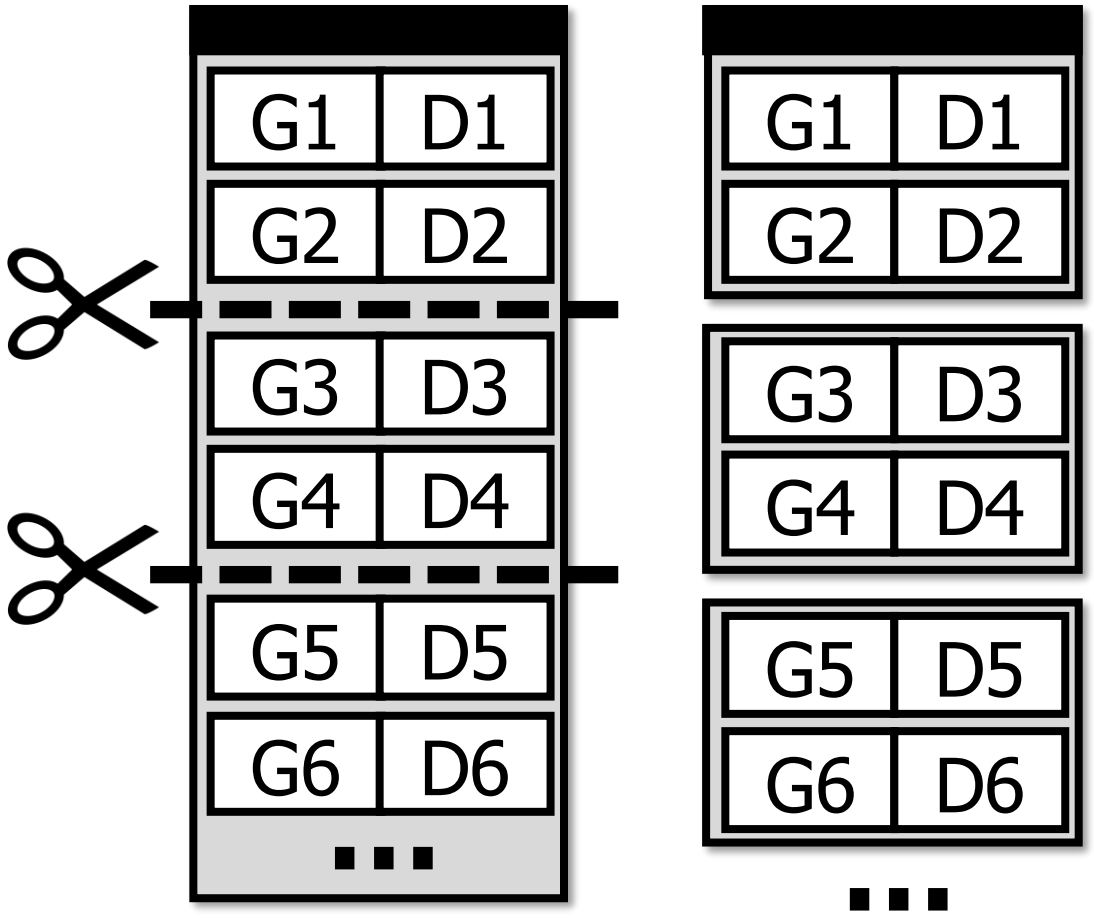
Details of Reordering



Insight

Relative order of Gaussians changes only slightly

Details of Reordering



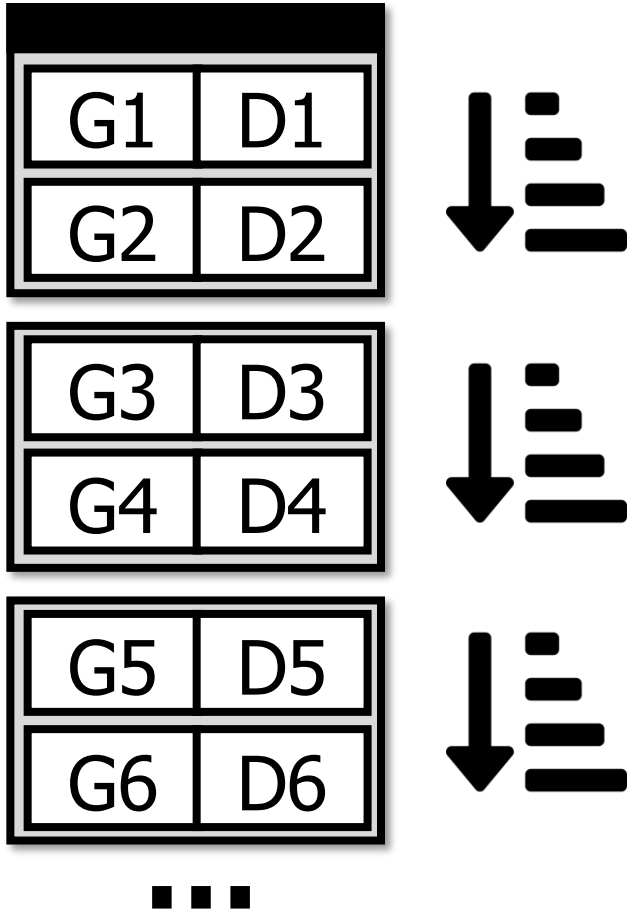
Insight

Relative order of Gaussians changes only slightly

Algorithm

① Splitting the table into chunks

Details of Reordering



Insight

Relative order of Gaussians changes only slightly

Algorithm

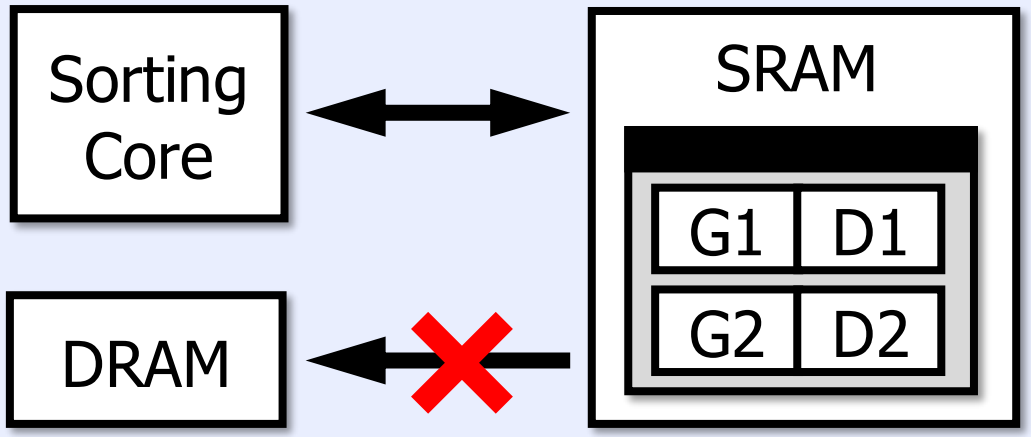
① **Splitting the table into chunks**

② **Sorting chunks independently**

Details of Reordering

Advantage

On-chip sorting
without additional DRAM access



Insight

Relative order of Gaussians changes only slightly

Algorithm

① **Splitting the table into chunks**

② **Sorting chunks independently**

Details of Reordering

Validation

Negligible quality degradation

difference of PSNR < 1.0 dB

Scene	Original 3DGS		Neo	
	PSNR↑	LPIPS↓	PSNR↑	LPIPS↓
Family	30.2	0.037	30.0 (▼0.2)	0.041 (▲0.004)
Francis	29.2	0.161	29.2 (●)	0.161 (●)
Horse	28.0	0.085	27.9 (▼0.1)	0.086 (▲0.001)
Lighthouse	26.1	0.071	26.1 (●)	0.071 (●)
Playground	25.2	0.179	25.2 (●)	0.179 (●)
Train	25.4	0.087	25.3 (▼0.1)	0.088 (▲0.001)

Insight

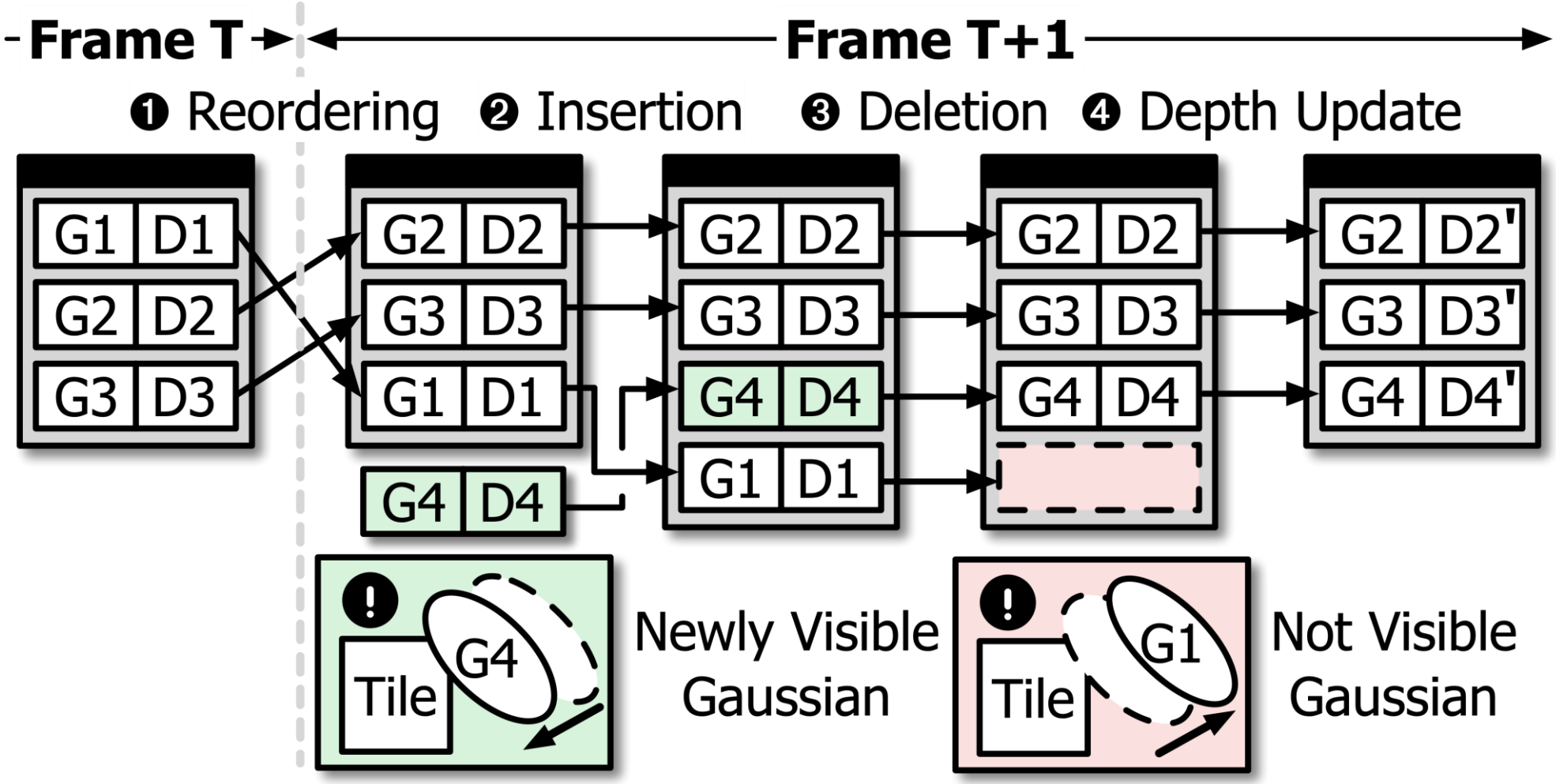
Relative order of Gaussians changes only slightly

Algorithm

① **Splitting the table into chunks**

② **Sorting chunks independently**

Overview of Neo-SW



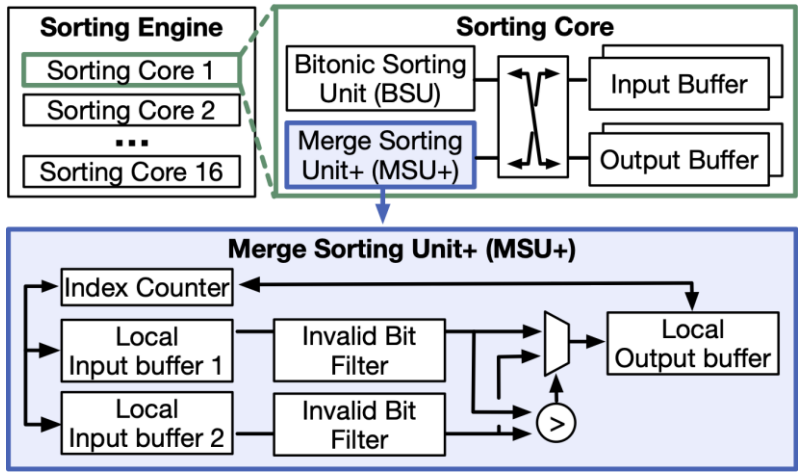
Neo Accelerator Integration

- Neo-HW supporting reuse-and-update sorting on-device

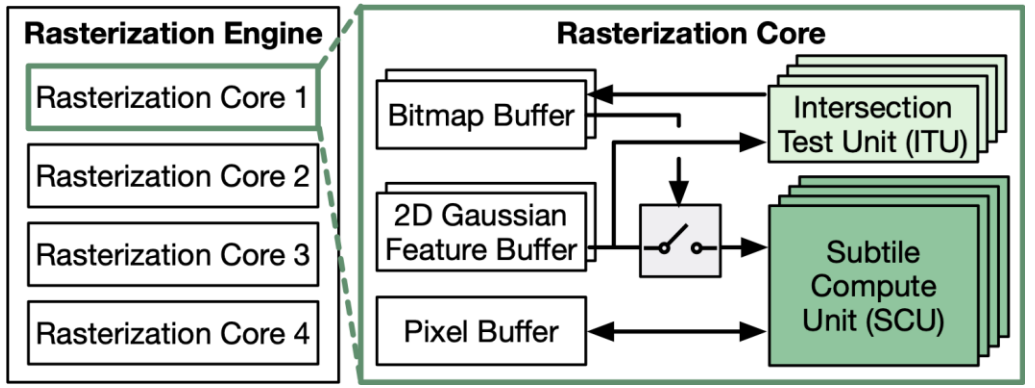
1 **Irregular insertion, deletion, and reordering limit SIMD efficiency**

2 **Requiring additional support to run Neo-SW during blending**

Dedicated sorting core



Microarchitecture with rendering core

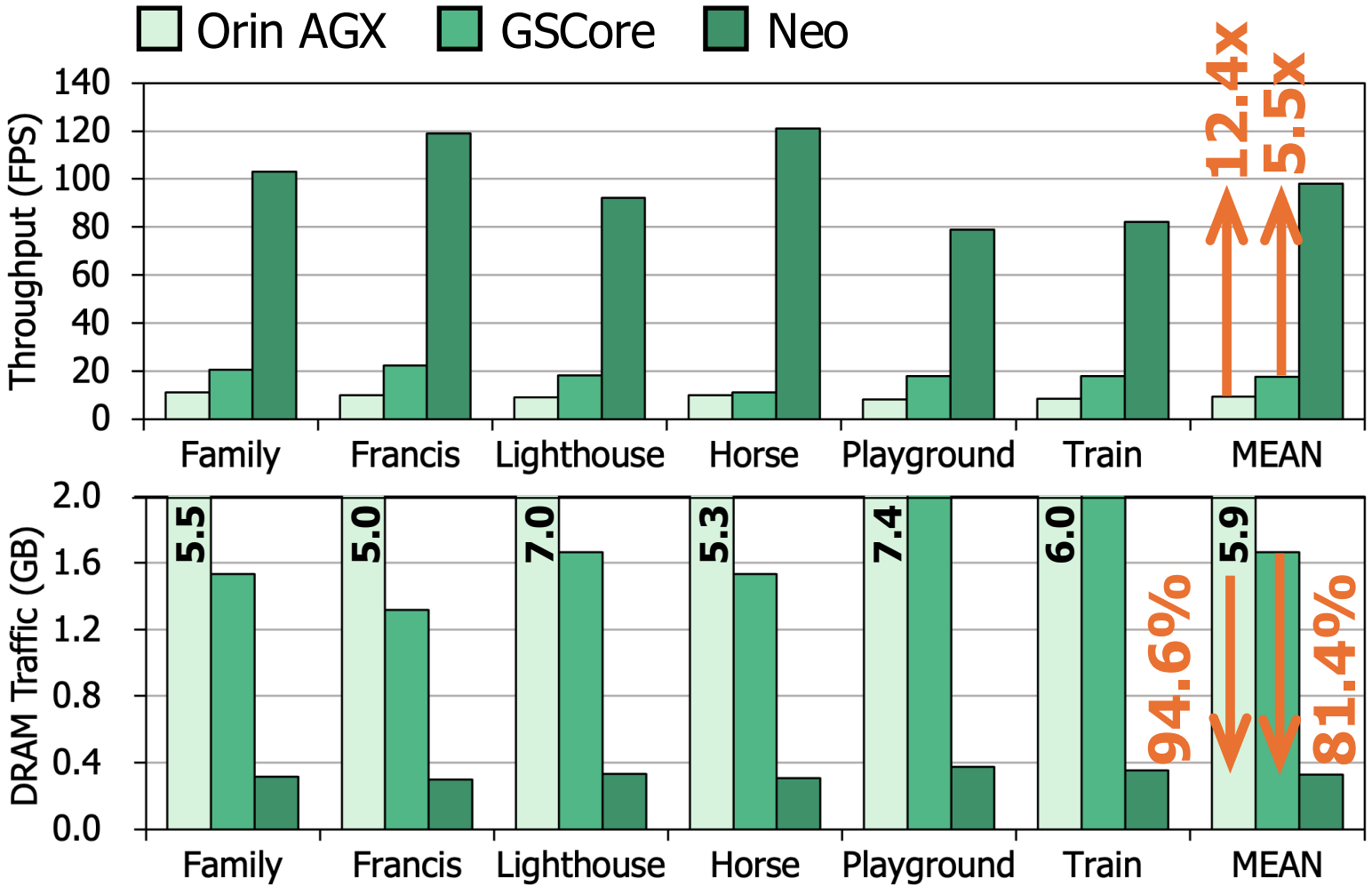


*** More details in paper**

Evaluation Methodology

- **Workloads**
  - Tanks and Temples: Family, Francis, Horse, Lighthouse, Playground, Train
  - Resolution: HD, FHD, QHD
- **Baselines**
  - GPU: NVIDIA Jetson AGX Orin 64GB
  - ASIC: GSCore, 16 core configuration
- **Hardware Modeling**
  - Synopsys Design Compiler (RTL synthesis), CACTI (SRAM)
- **System Modeling**
  - Modified GSCore simulator
  - LPDDR4 in Ramulator

Performance Evaluation



- **Throughput (QHD)**
  - vs Orin: **12.4×** FPS
  - vs GSCore: **5.5×** FPS
- **Memory Traffic (QHD)**
  - vs Orin: **94.6%↓**
  - vs GSCore: **81.4%↓**

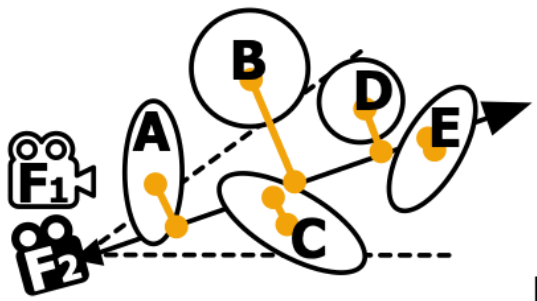
More Results in Paper

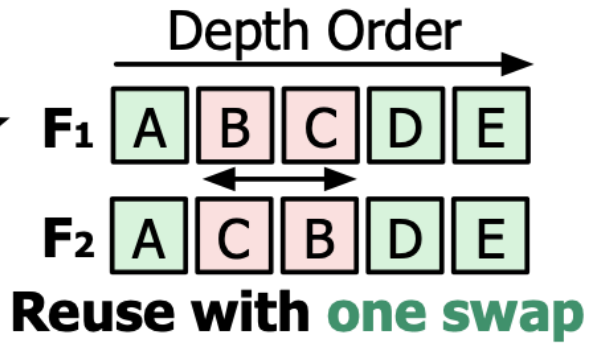
- **Area and power breakdown of hardware components in Neo**
- **Comparison with other sorting reuse strategies**
  (1) Periodic sorting
  (2) Background sorting
- **Performance results of extreme scenarios**
  (1) Large scale scene
  (2) Rapid camera movement
- **Performance breakdown of Neo hardware**
  (1) Speedup
  (2) Relative DRAM traffic

Conclusion

- **Neo**
  - Real-time on-device 3DGS acceleration via reuse-and-update sorting
- **Contributions**
  - **Finding**: stricter on-device constraints expose sorting as a critical bottleneck
  - **Idea**: temporal similarity across consecutive frames enables sorting reuse
  - **System**: HW/SW co-designed pipeline realizes reuse-and-update sorting on-device

Reuse-and-Update Sorting





Performance

5.5x
Faster Rendering

81.4%
Reducing DRAM Traffic

Code