GSCore

Efficient Radiance Field Rendering via Architectural Support for 3D Gaussian Splatting

Junseo Lee Seokwon Lee Jungi Lee Junyong Park Jaewoong Sim

Seoul National University

ASPLOS'24 | April 2024

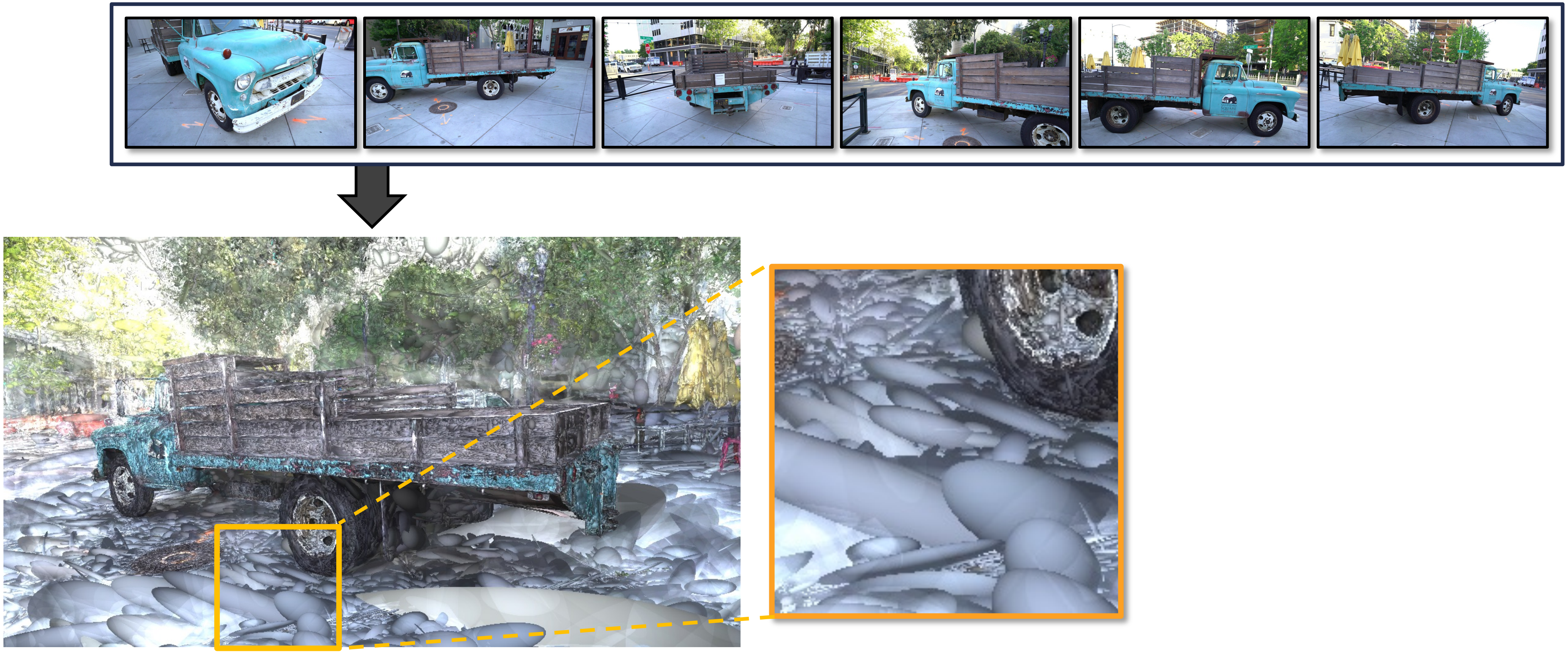
Department of ECE
Seoul National University

3D Gaussian Splatting

3D Gaussian Splatting

Captured Images

3D Gaussian Splatting

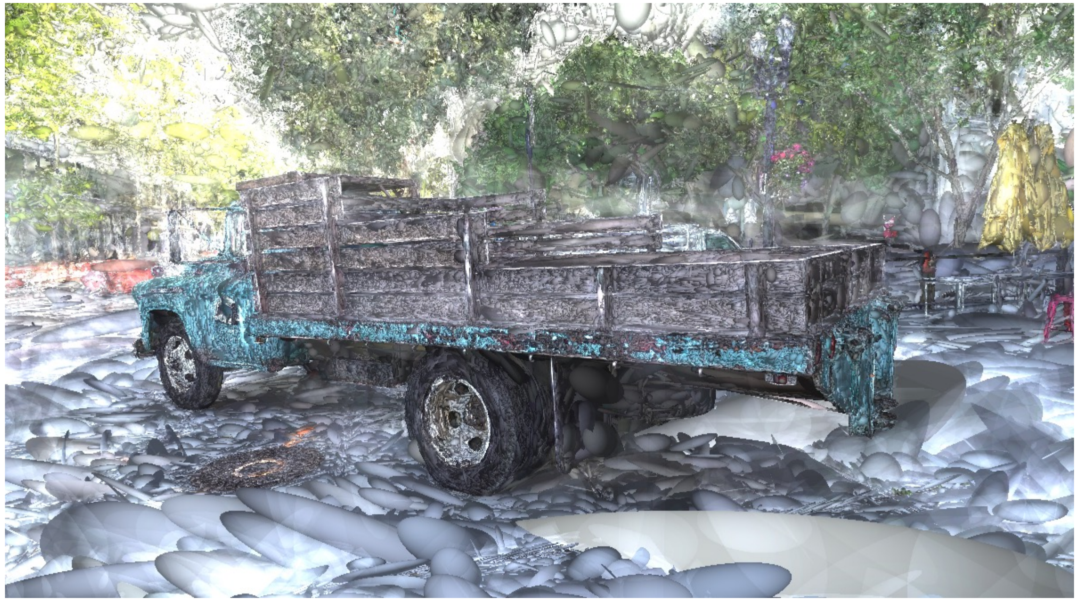


Captured Images

3D Gaussians

3D Gaussian Splatting

Captured Images



3D Gaussians

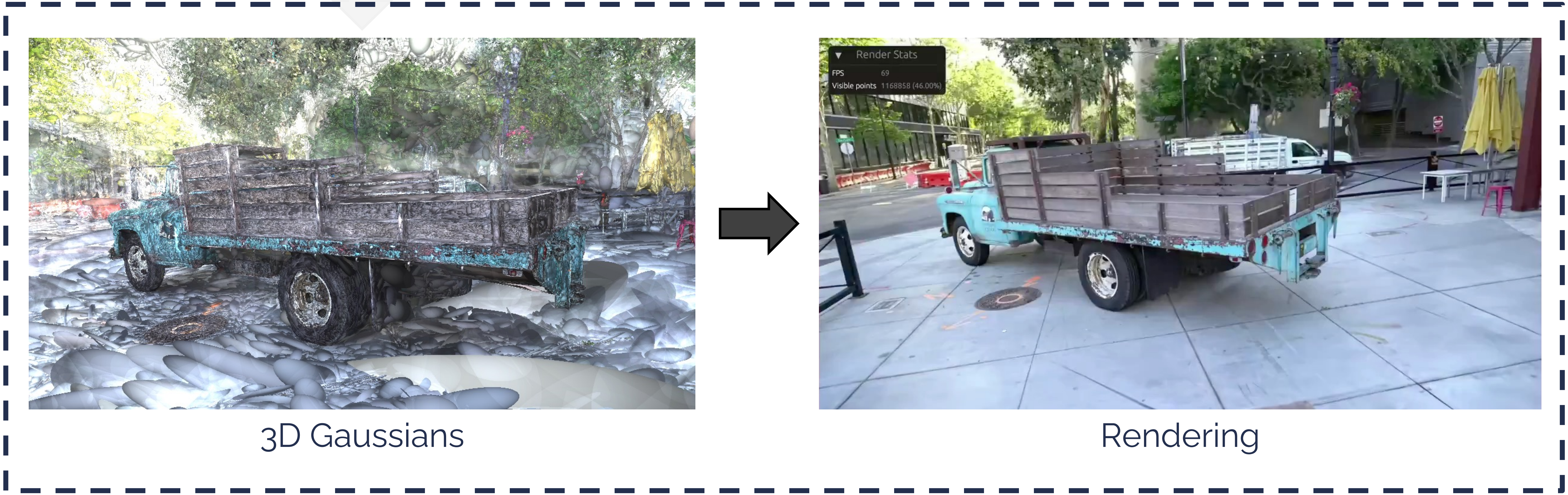
Render Stats
FPS 69
Visible points 1168858 (46.00%)

Rendering

3D Gaussian Splatting

Captured Images

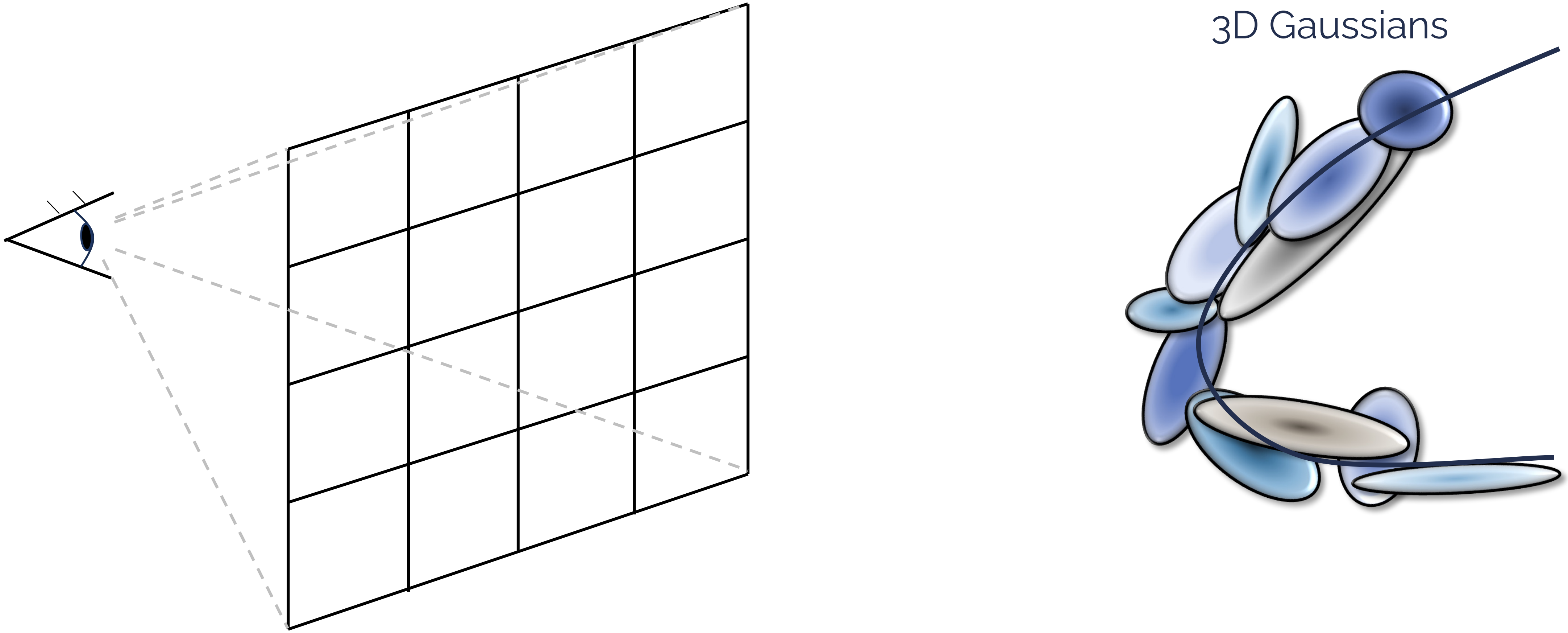
Rendering Process



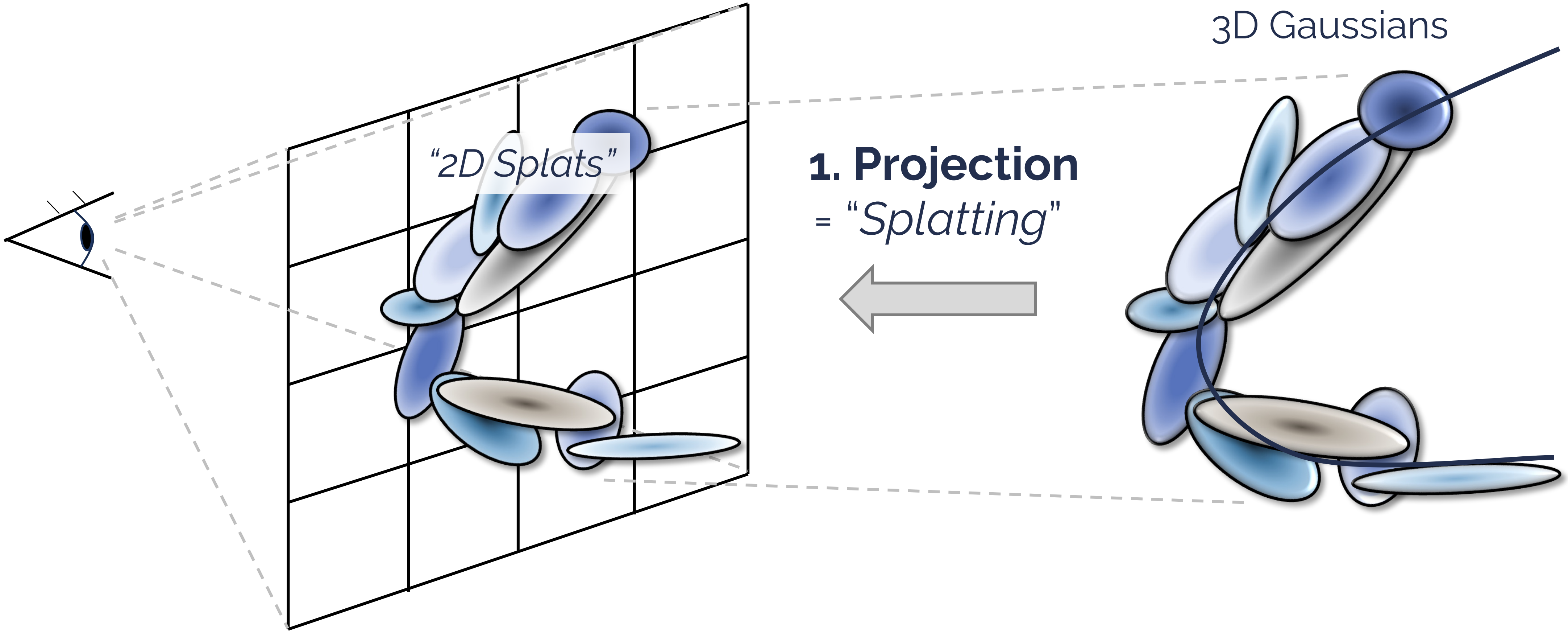
3D Gaussians

Rendering

Rendering Flow

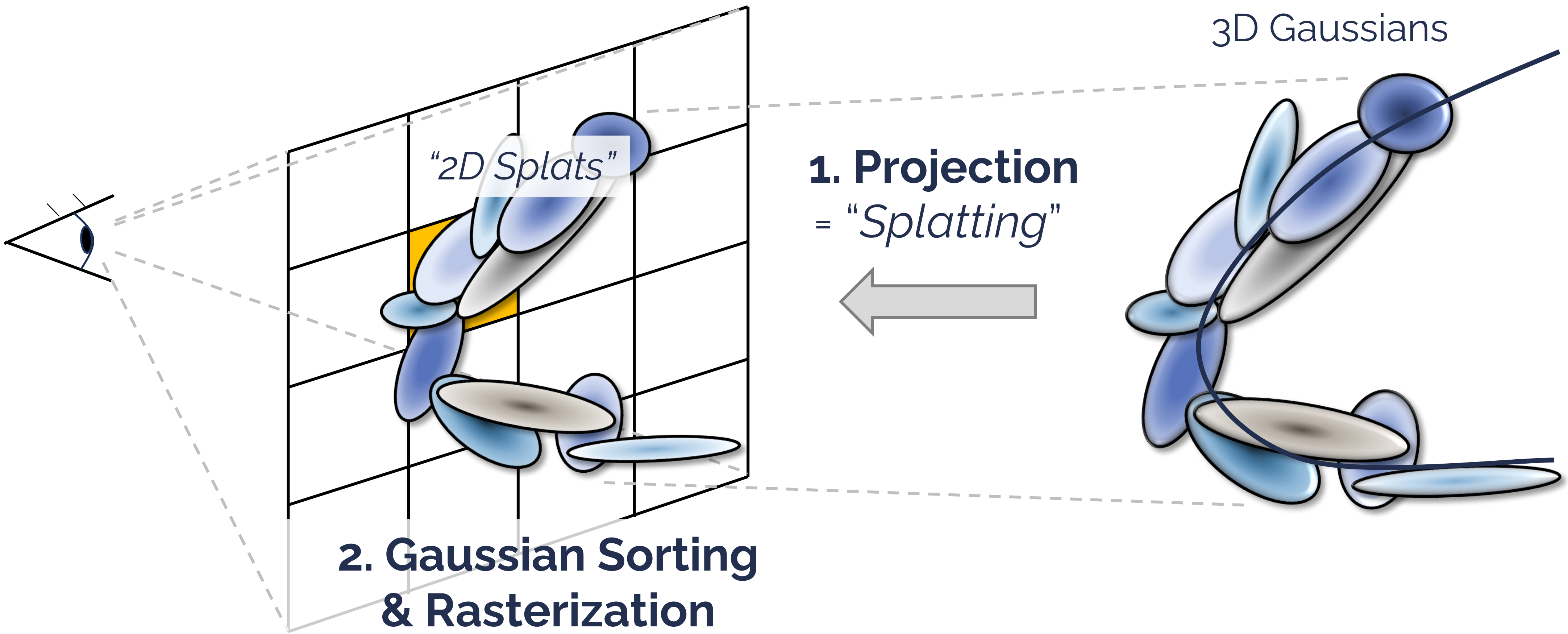


Rendering Flow



1. Projection
= *"Splatting"*

Rendering Flow



3D Gaussians

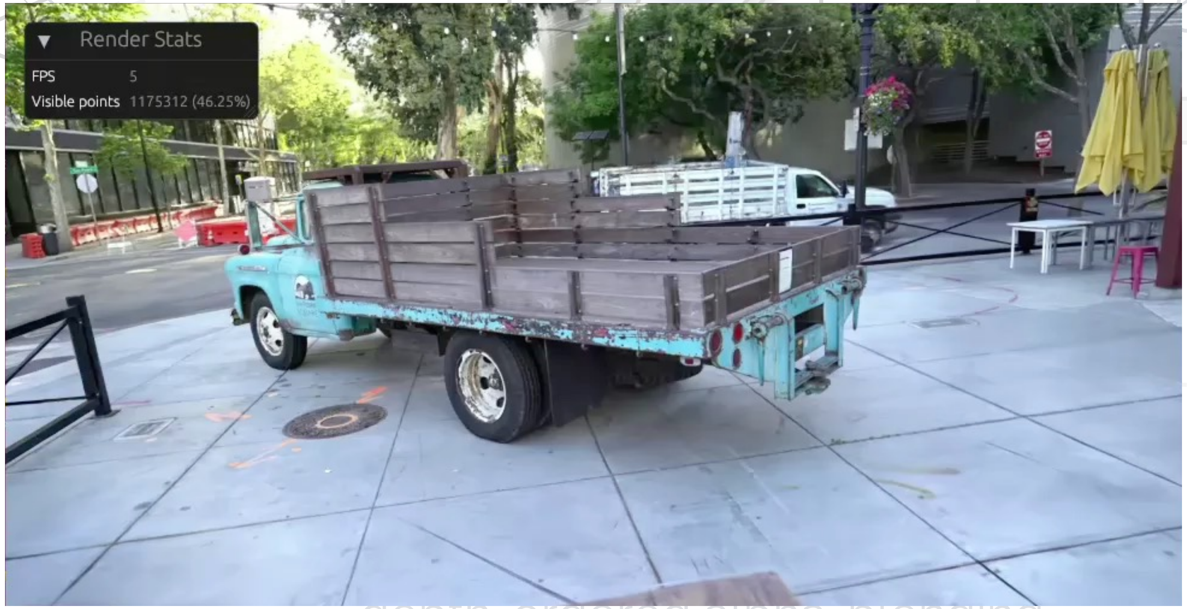
1. Projection
= *"Splatting"*

"2D Splats"

2. Gaussian Sorting & Rasterization
depth-ordered alpha-blending

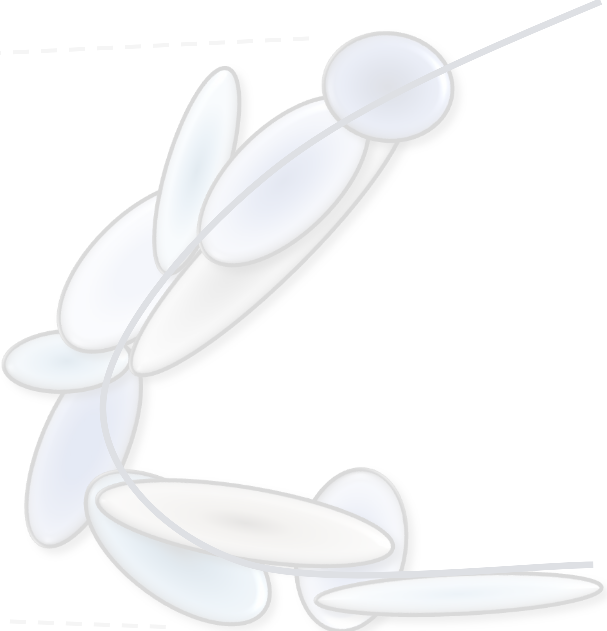
Rendering Flow

Still too slow on the edge GPU



Render Stats

```
FPS             5
Visible points  1175312 (46.25%)
```



3D Gaussians

1. Projection

“Splatting”

depth-ordered alpha-blending

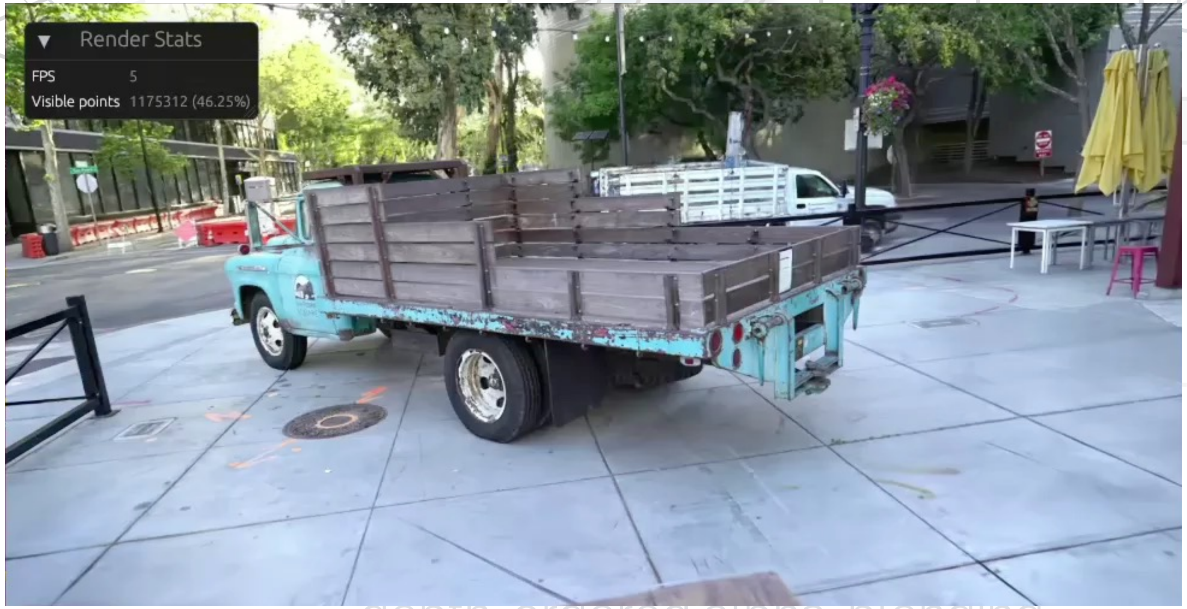
Rendering Flow

Still too slow on the edge GPU

Sorting & Rasterization are bottlenecks

1. Projection

“Splatting”

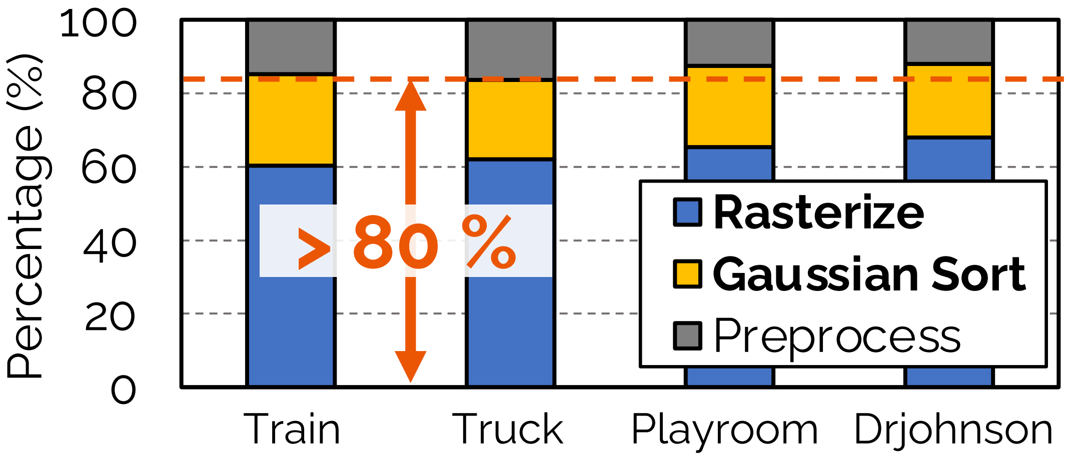


▼ Render Stats

FPS 5

Visible points 1175312 (46.25%)

depth-ordered alpha-blending



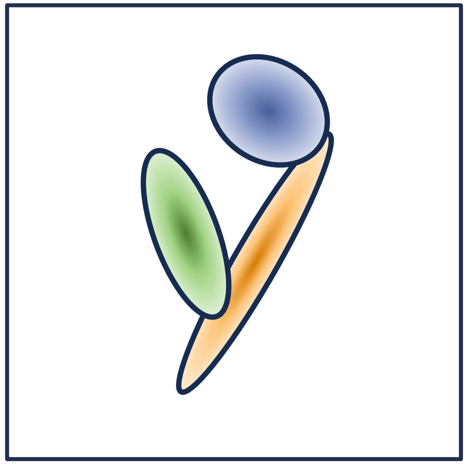
Outline

- Background
  - 3D Gaussian Splatting (3DGS)
- 3DGS Optimization & Inefficiencies
- GSCore: Efficient Radiance Field Rendering Accelerator
  - Algorithmic Optimizations
  - Hardware Architecture
- Evaluation
- Conclusion

3DGS Optimization

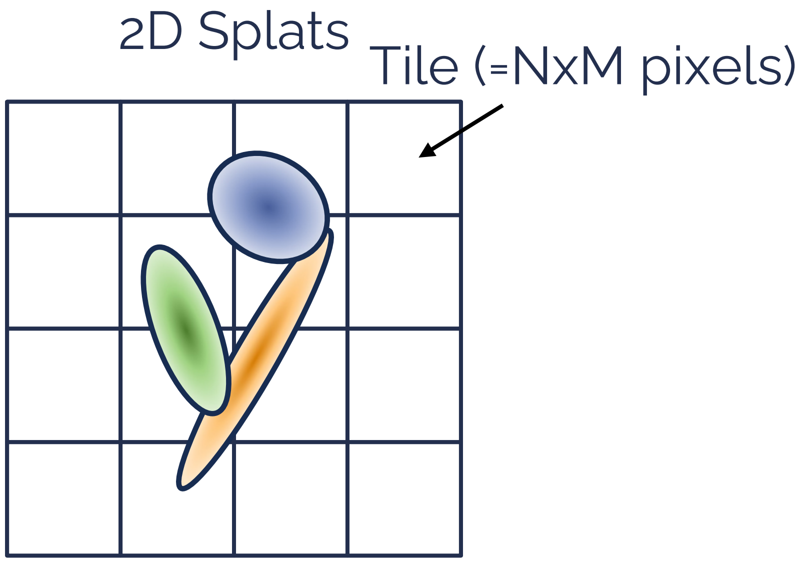
Tile-based Rasterization

2D Splats



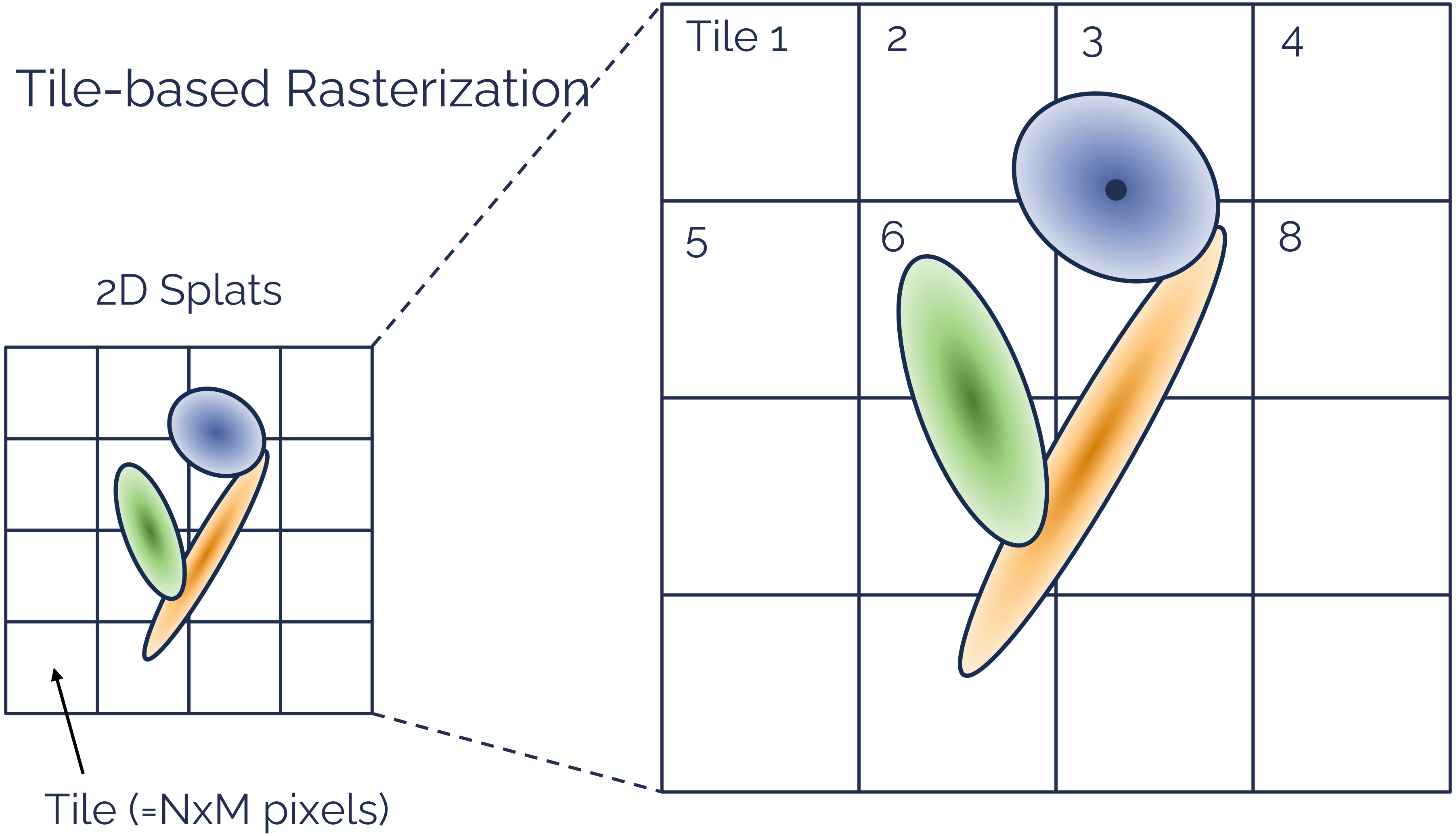
3DGS Optimization

Tile-based Rasterization



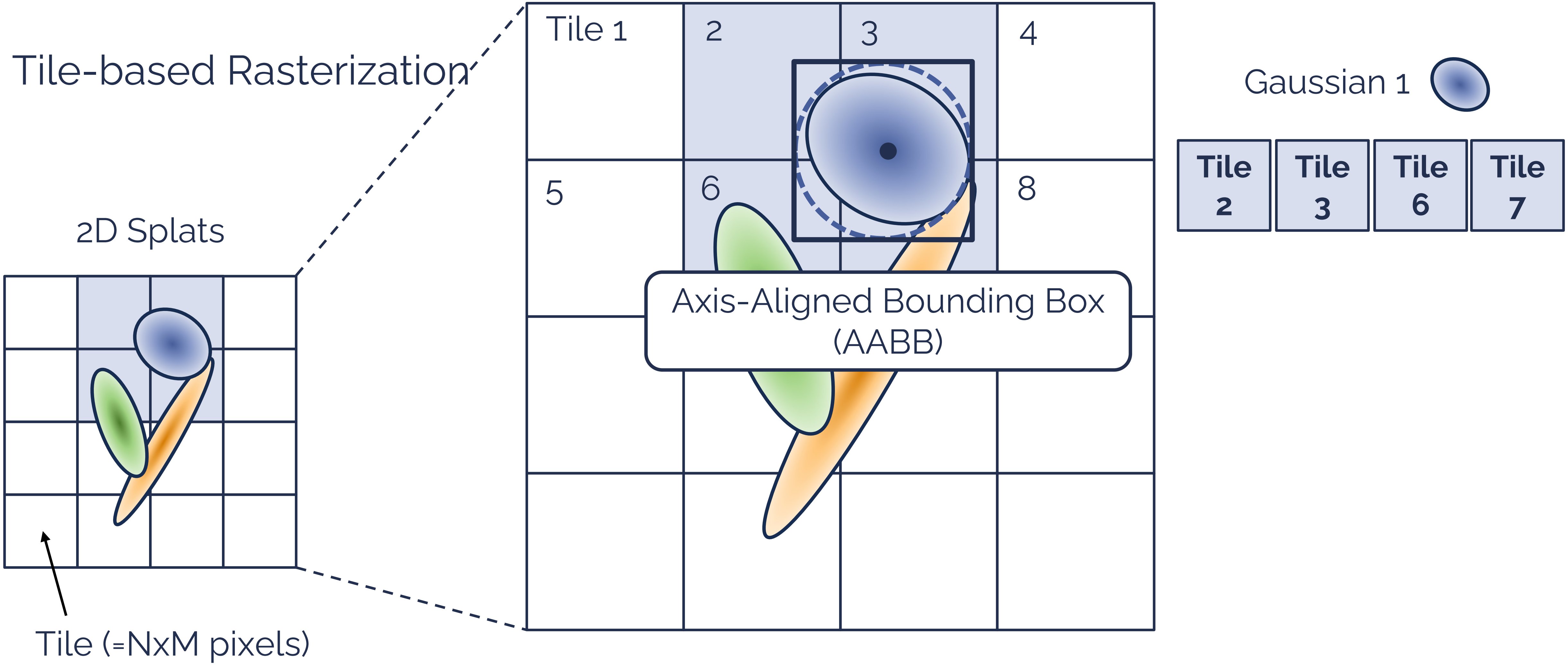
3DGS Optimization

Tile-based Rasterization

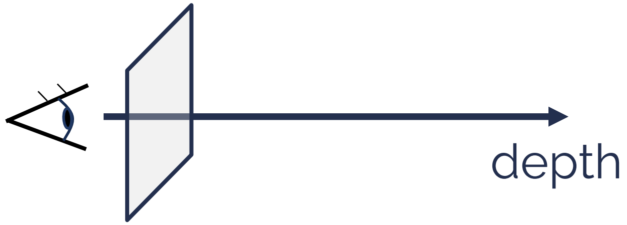


3DGS Optimization

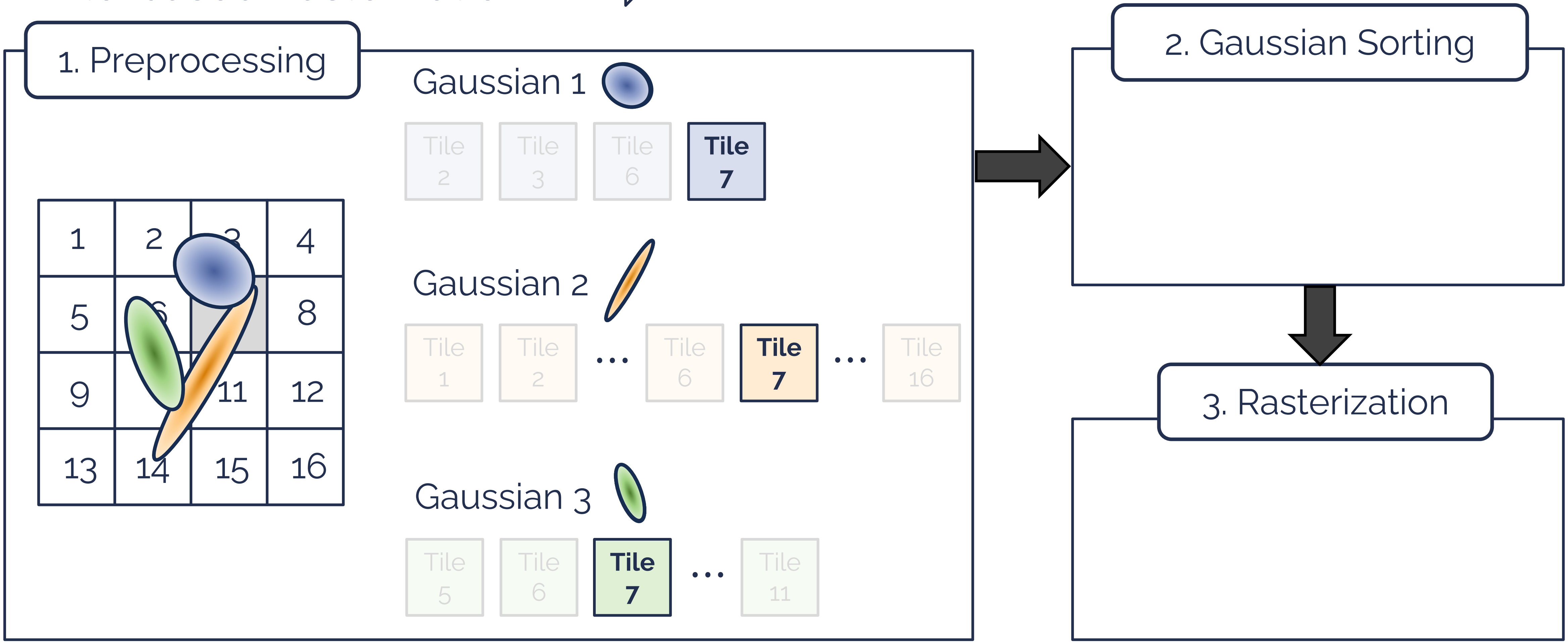
Tile-based Rasterization



3DGS Optimization



Tile-based Rasterization



1. Preprocessing

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Gaussian 1

Tile 2, Tile 3, Tile 6, **Tile 7**

Gaussian 2

Tile 1, Tile 2 … Tile 6, **Tile 7** … Tile 16

Gaussian 3

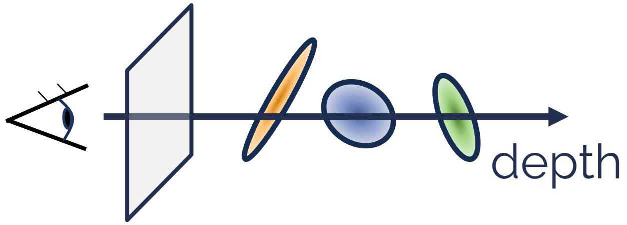
Tile 5, Tile 6, **Tile 7** … Tile 11

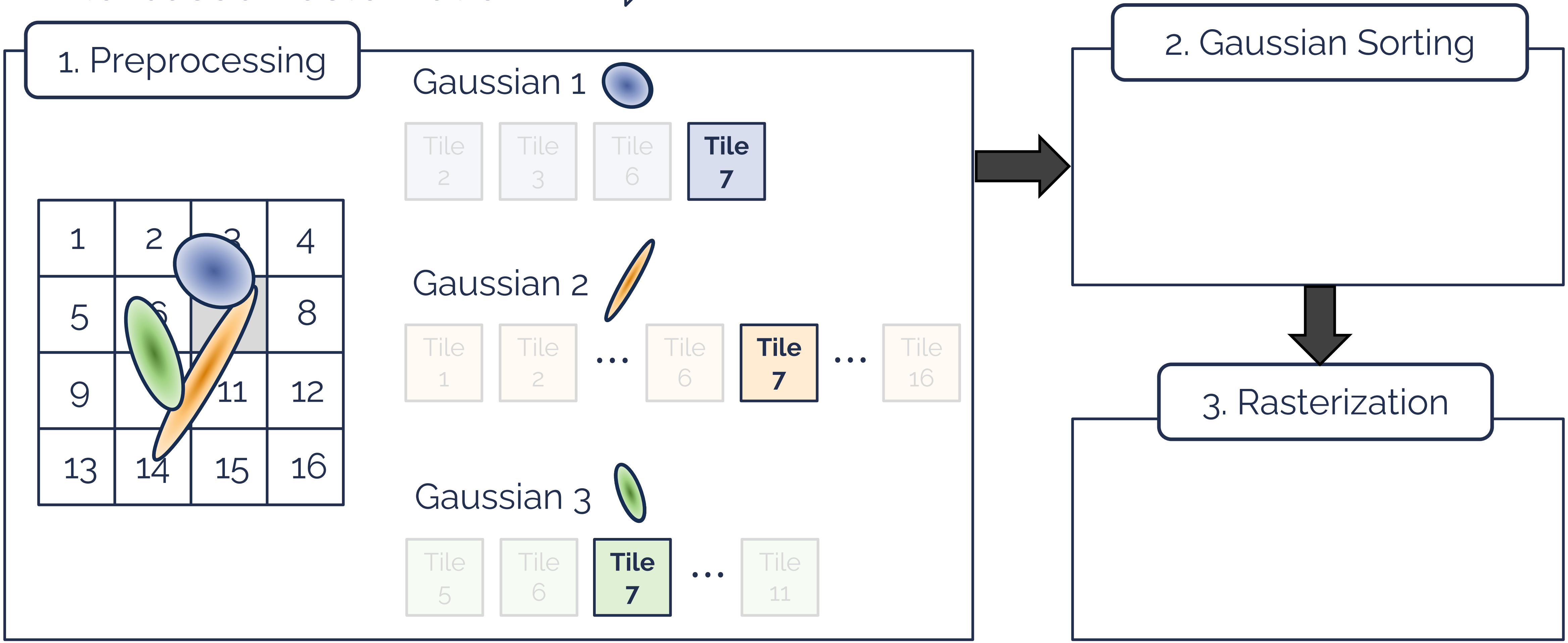
2. Gaussian Sorting

3. Rasterization

3DGS Optimization

Tile-based Rasterization





1. Preprocessing

Gaussian 1

Tile 2 | Tile 3 | Tile 6 | **Tile 7**

Gaussian 2

Tile 1 | Tile 2 | … | Tile 6 | **Tile 7** | … | Tile 16

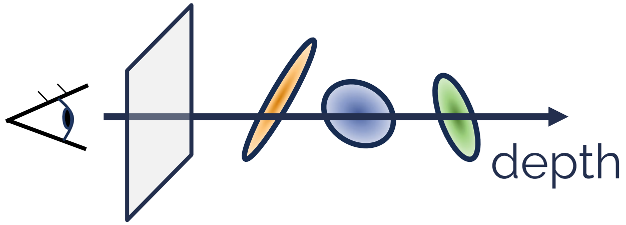
Gaussian 3

Tile 5 | Tile 6 | **Tile 7** | … | Tile 11

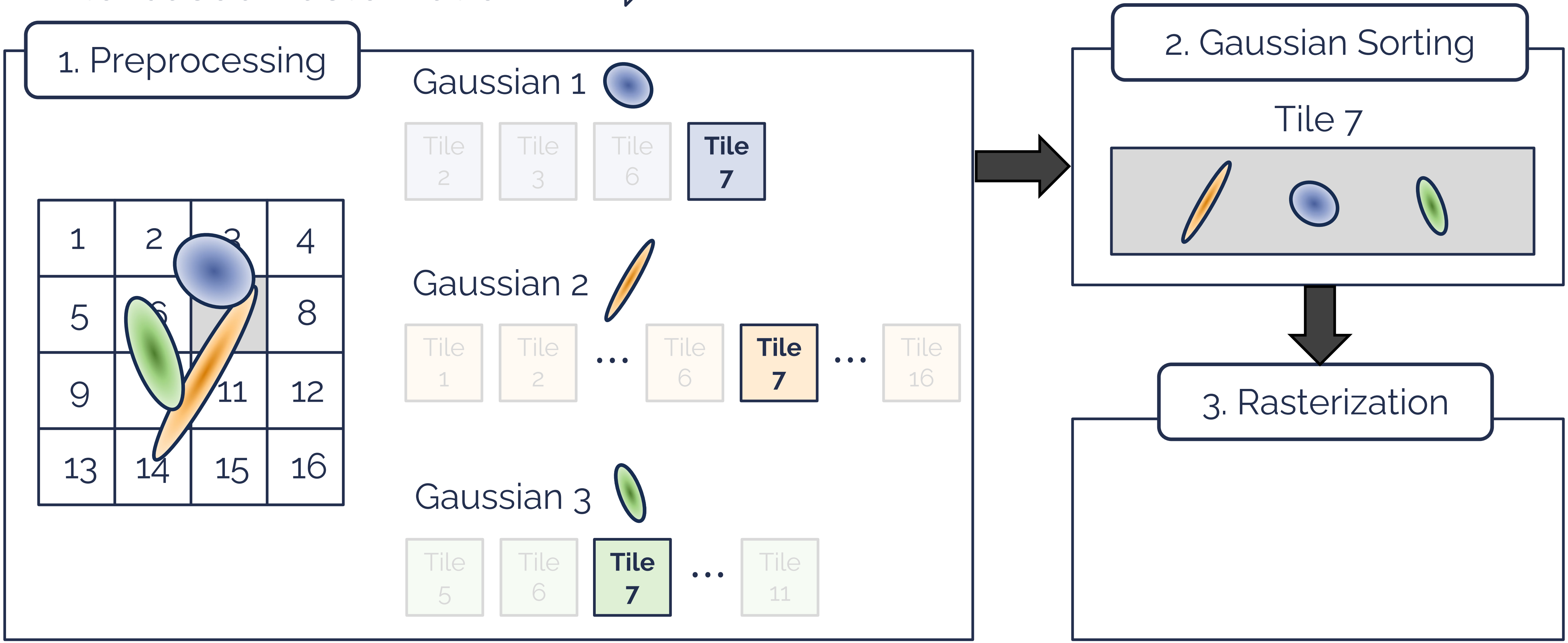
2. Gaussian Sorting

3. Rasterization

3DGS Optimization



Tile-based Rasterization

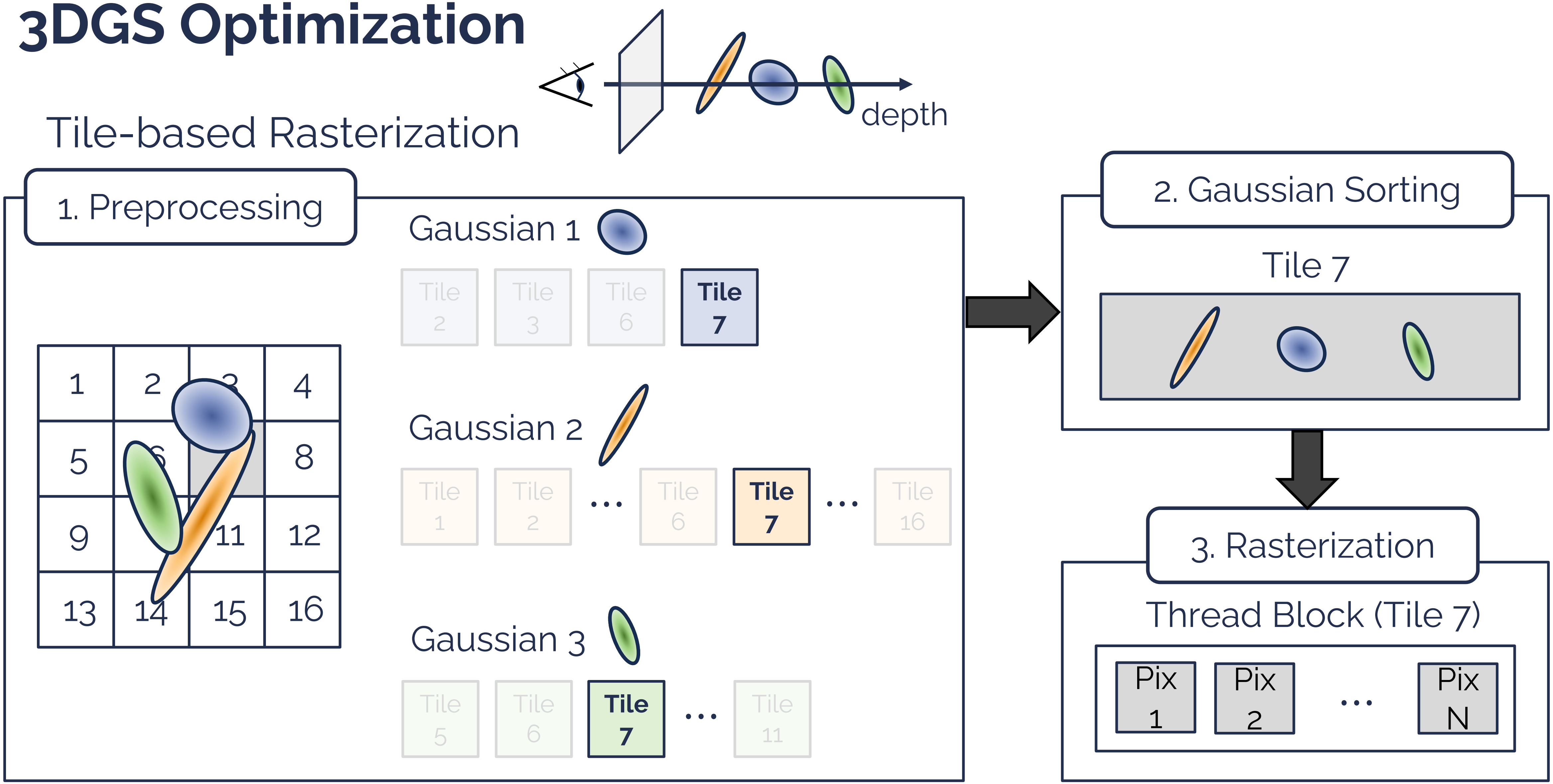

1. Preprocessing

2. Gaussian Sorting

3. Rasterization

3DGS Optimization

Tile-based Rasterization



1. Preprocessing

Gaussian 1

Gaussian 2

Gaussian 3

2. Gaussian Sorting

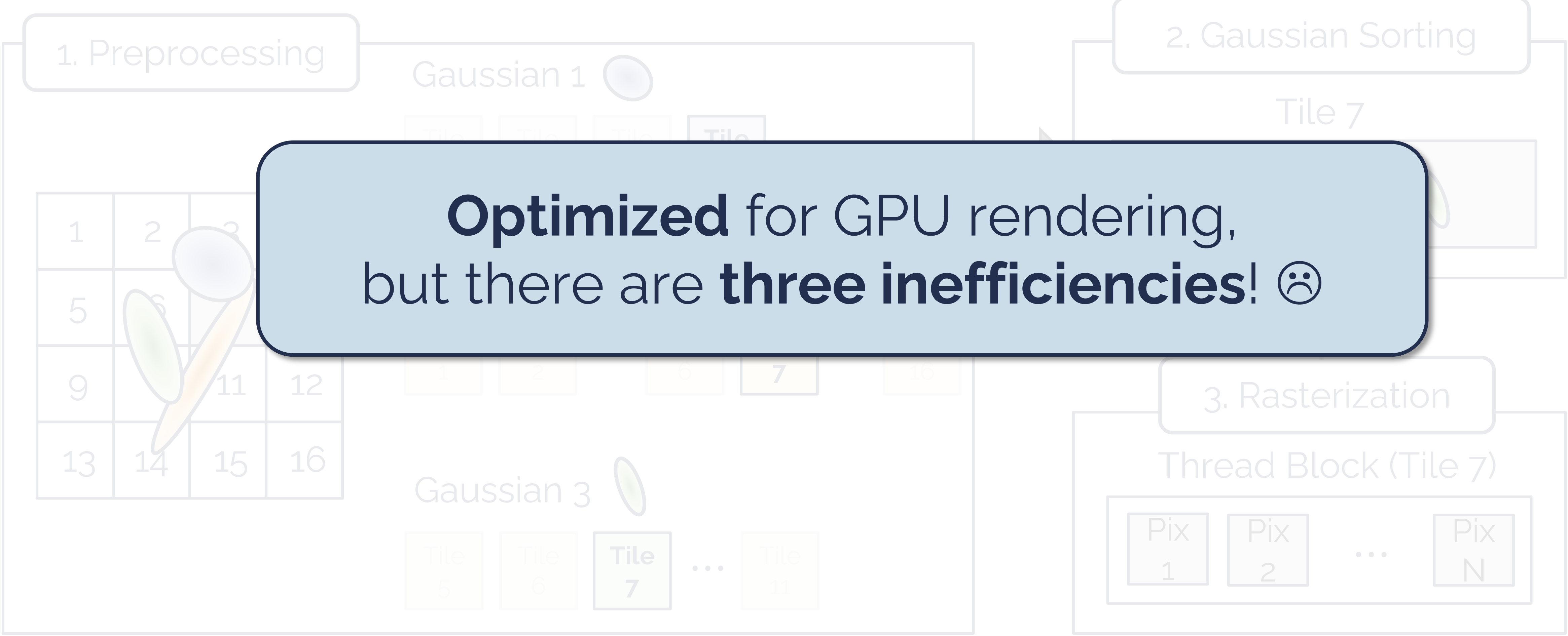
Tile 7

3. Rasterization

Thread Block (Tile 7)

3DGS Optimization

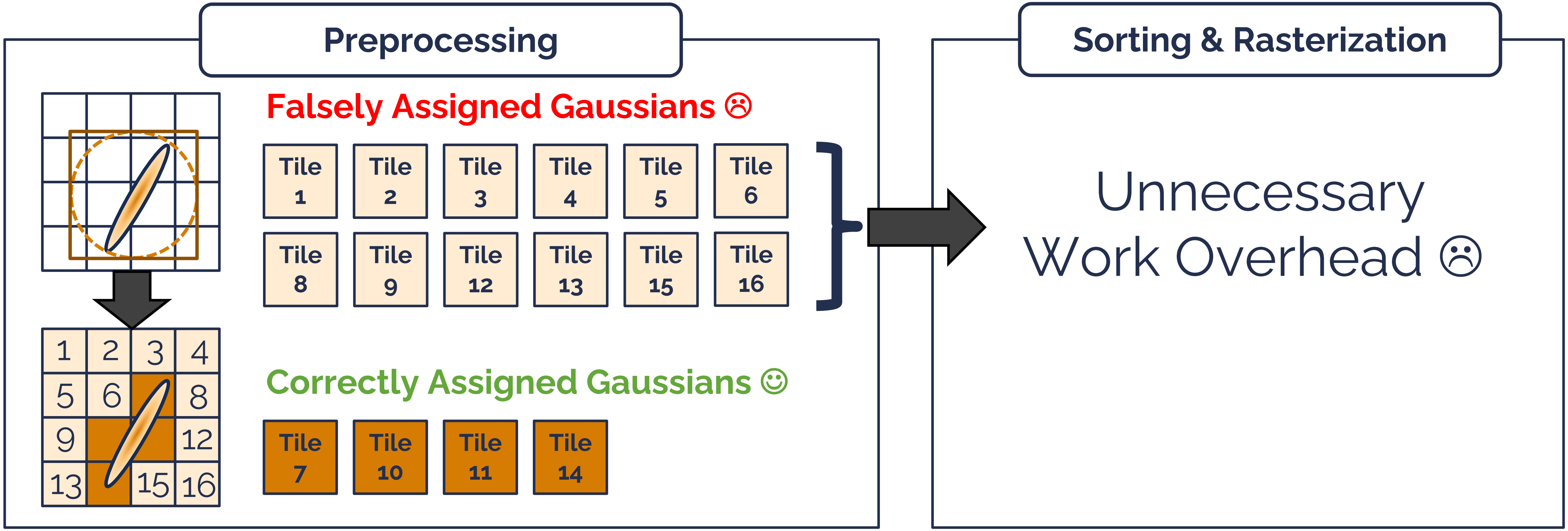
Tile-based Rasterization



Optimized for GPU rendering,
but there are **three inefficiencies**! ☹

Inefficiencies in 3DGS

Problem 1.
Unnecessary Sorting & Rasterization
due to **Falsely Assigned Gaussians**



Preprocessing

Falsely Assigned Gaussians ☹

Tile 1, Tile 2, Tile 3, Tile 4, Tile 5, Tile 6, Tile 8, Tile 9, Tile 12, Tile 13, Tile 15, Tile 16

Correctly Assigned Gaussians ☺

Tile 7, Tile 10, Tile 11, Tile 14

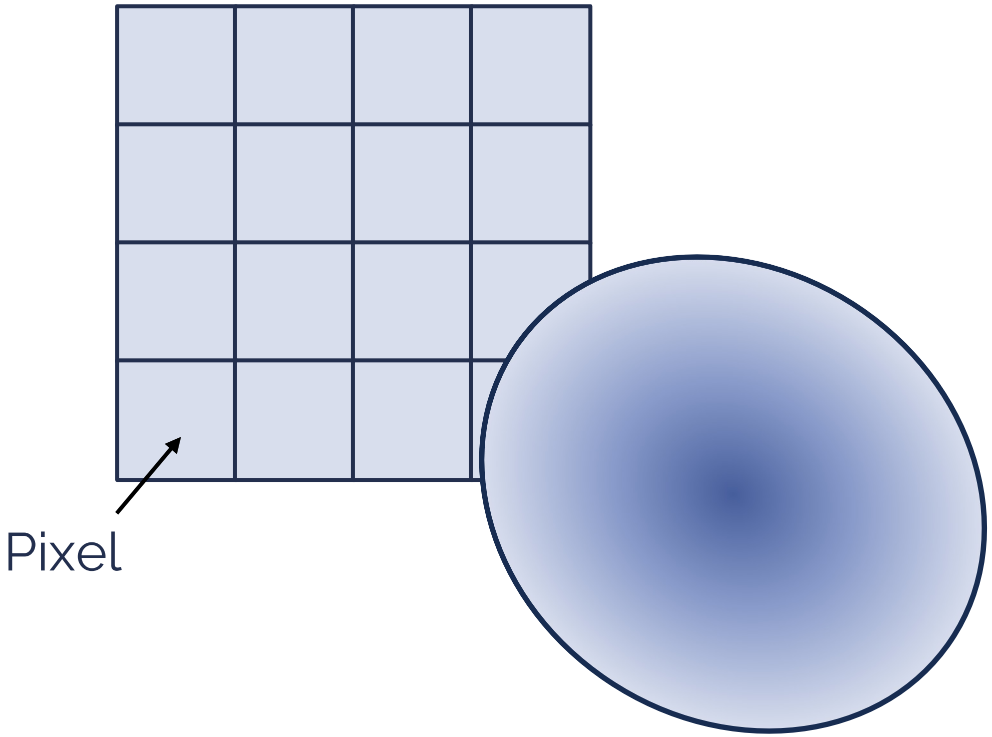
Sorting & Rasterization

Unnecessary
Work Overhead ☹

Inefficiencies in 3DGS

Problem 2.

Ineffective Alpha Computation



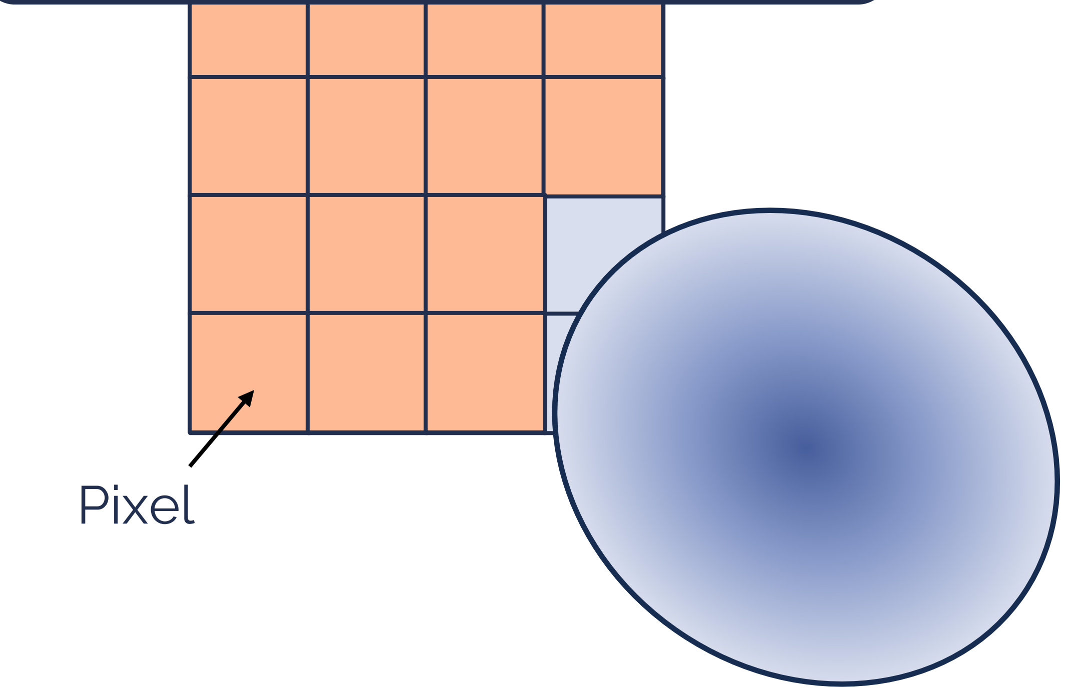
Pixel Rendering

```
for Gaus in { Gaus 1 , ... }
    $\alpha$-Computation
    if (pixel is out of boundary)
        continue
    $\alpha$-Blending
```

Inefficiencies in 3DGS

Problem 2.
Ineffective Alpha Computation

14 out of 16 (=87.5%) threads do **ineffective α computation** ☹



Pixel Rendering

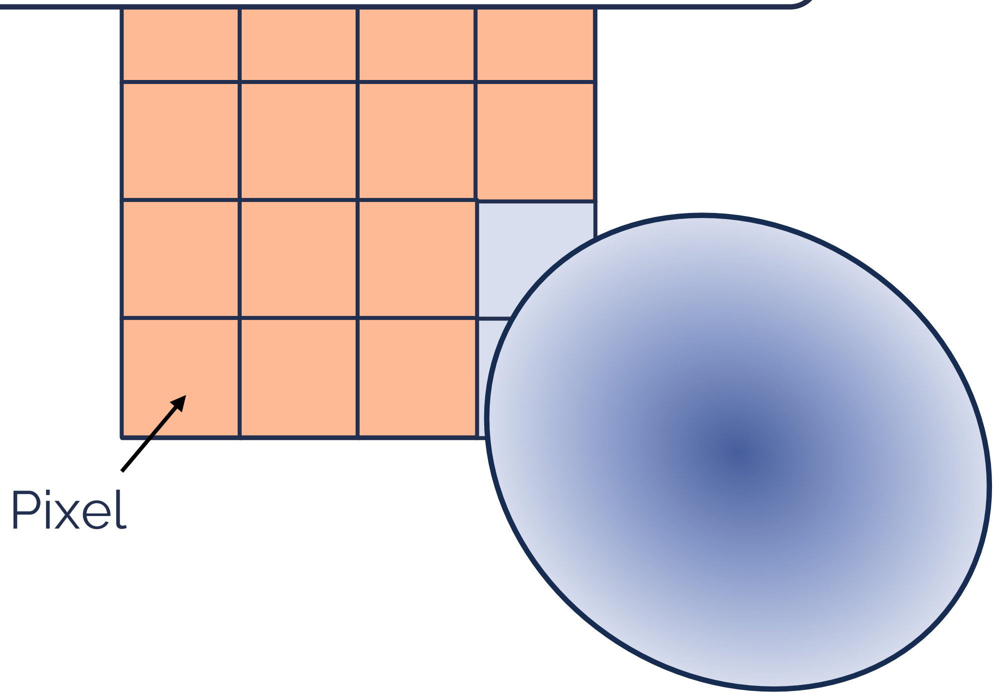
```
for **Gaus** in { [Gaus 1] , … }
    $\alpha$-Computation
    if (pixel is out of boundary)
        continue
    $\alpha$-Blending
```

Inefficiencies in 3DGS

Problem 2.

Ineffective Alpha Computation

14 out of 16 (=87.5%) threads do **ineffective α computation** ☹



Multiple Exp. & FP operations ☹

$$\alpha = o_i * e^{(p-\mu)^T \Sigma'^{-1}(p-\mu)}$$

Pix

```
for (
    $\alpha$-Computation
    if (pixel is out of boundary)
        continue
    $\alpha$-Blending
```

Inefficiencies in 3DGS

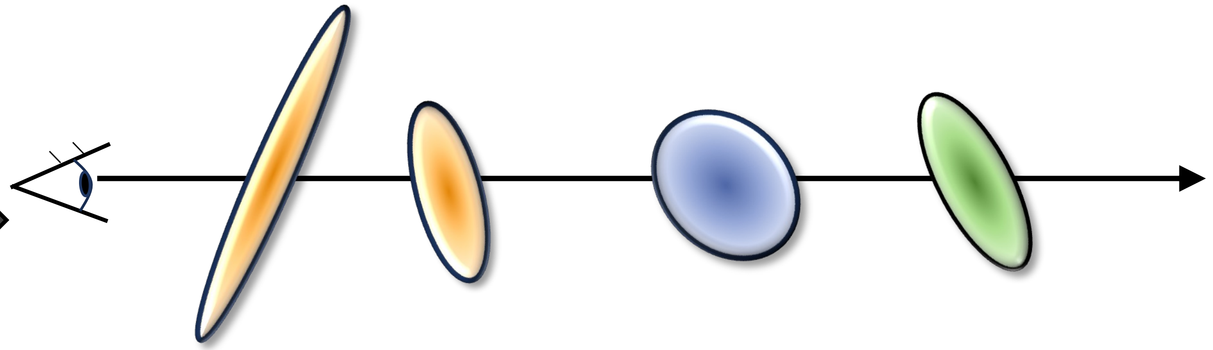
Problem 3.

Unnecessary Sorting Overhead

Gaussian Sorting

Sort the ***entire*** Gaussians ***before*** rasterization

Rasterization



Inefficiencies in 3DGS

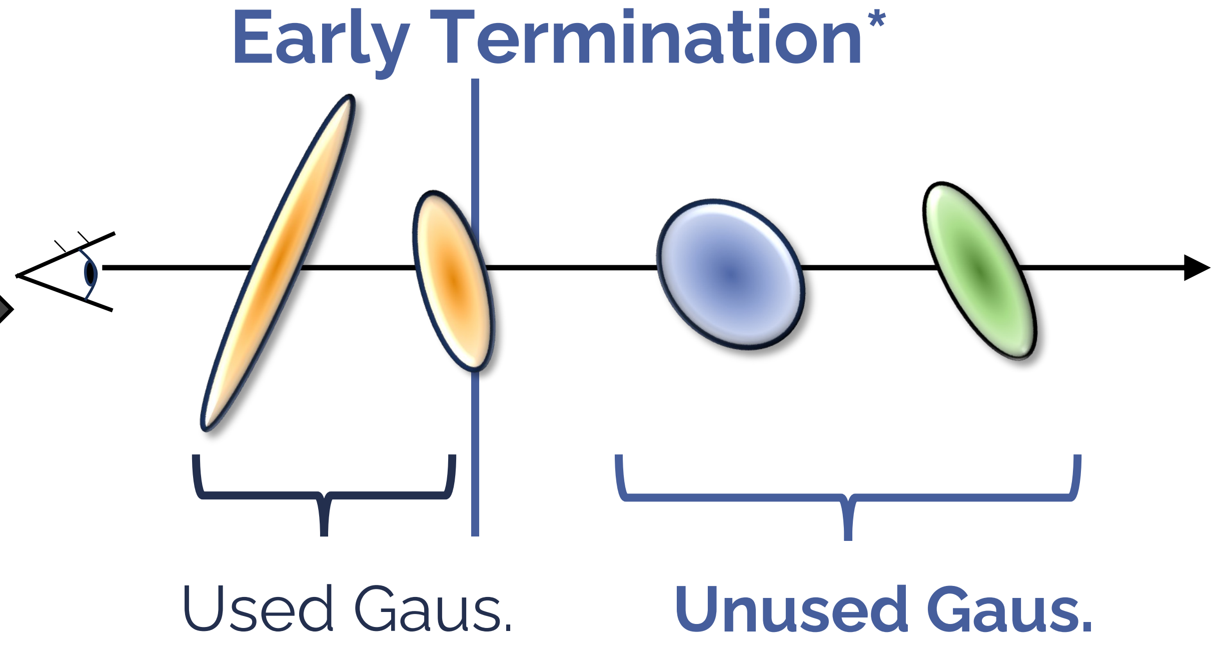
Problem 3.
Unnecessary Sorting Overhead

Early Termination
: **Stop** rasterization
if we meet the **surface**

Gaussian Sorting

Sort the ***entire*** Gaussians ***before*** rasterization

Rasterization



Inefficiencies in 3DGS

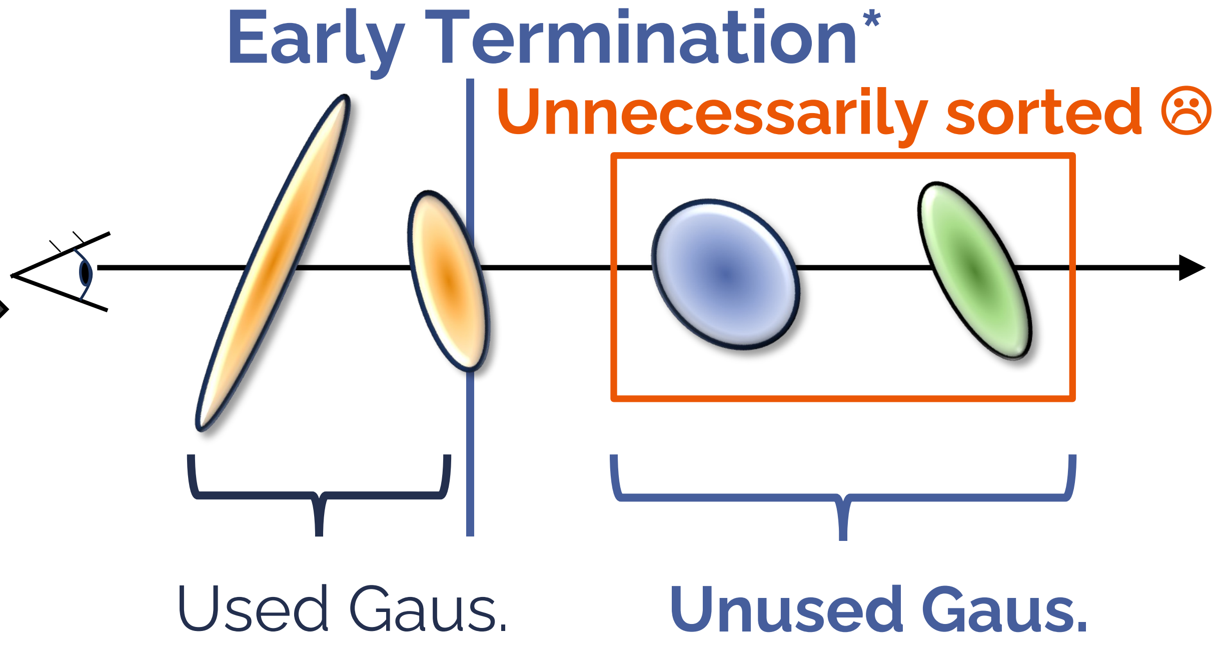
Problem 3.
Unnecessary Sorting Overhead

Early Termination
: **Stop** rasterization
if we meet the **surface**

Gaussian Sorting

Sort the ***entire*** Gaussians ***before*** rasterization

Rasterization



Outline

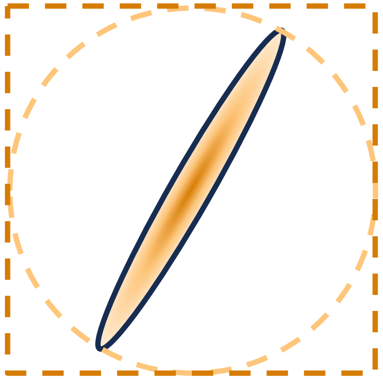
- Background
  - 3D Gaussian Splatting (3DGS)
- 3DGS Optimization & Inefficiencies
- **GSCore: Efficient Radiance Field Rendering Accelerator**
  - Algorithmic Optimizations
  - Hardware Architecture
- Evaluation
- Conclusion

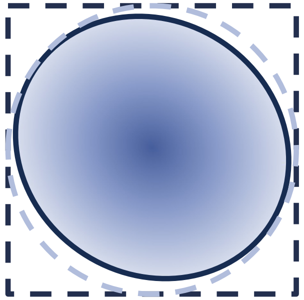
Gaussian Shape-Aware Intersection Test

Original

: **AABB-Based** Intersection Test

Axis-Aligned Bounding Box (AABB)





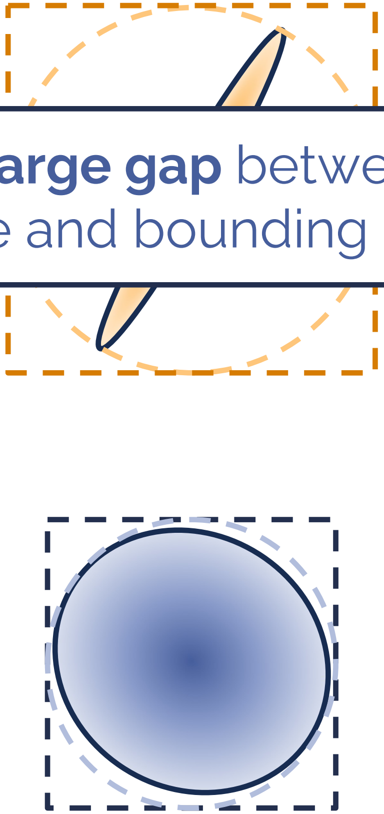
Gaussian Shape-Aware Intersection Test

Original

: **AABB-Based** Intersection Test

VS.

Axis-Aligned Bounding Box (AABB)



A **large gap** between ellipse and bounding box ☹

Gaussian Shape-Aware Intersection Test

Original

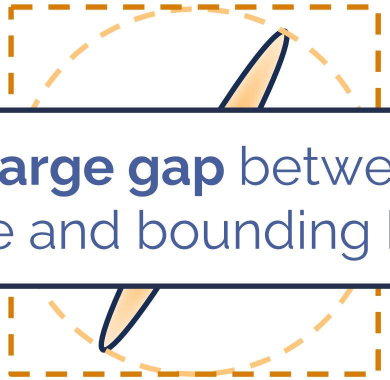
: **AABB-Based** Intersection Test

vs.

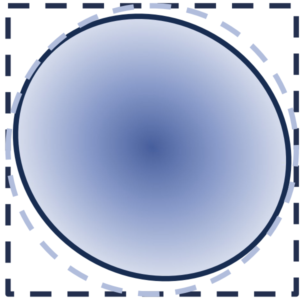
GSCore

: **Shape-Aware** Intersection Test

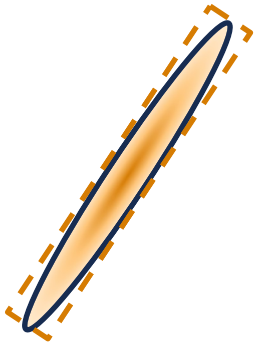
Axis-Aligned Bounding Box (AABB)



A **large gap** between ellipse and bounding box ☹



Tighter Bounding Box
(e.g., Oriented Bounding Box (OBB))



Gaussian Shape-Aware Intersection Test

Original

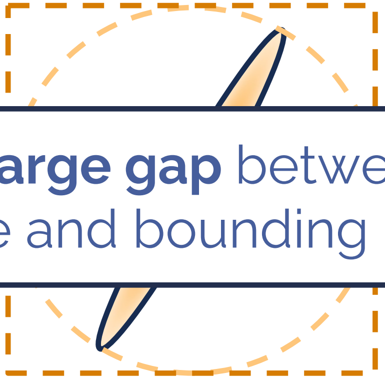
: **AABB-Based** Intersection Test

vs.

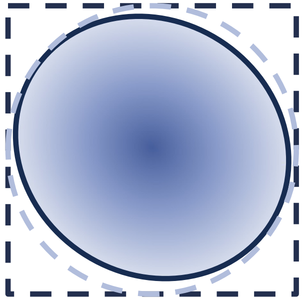
GSCore

: **Shape-Aware** Intersection Test

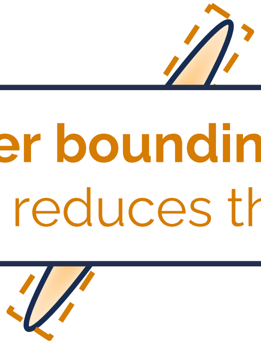
Axis-Aligned Bounding Box (AABB)



A **large gap** between ellipse and bounding box ☹



Tighter Bounding Box
(e.g., Oriented Bounding Box (OBB))



A **tighter bounding box** effectively reduces the gap! ☺

Gaussian Shape-Aware Intersection Test

Original

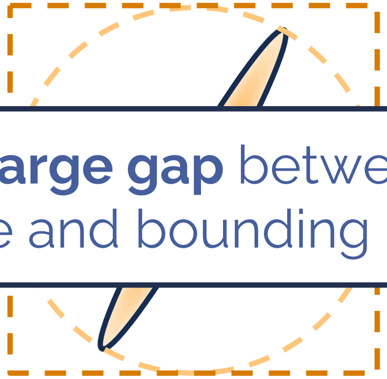
: **AABB-Based** Intersection Test

VS.

GSCore

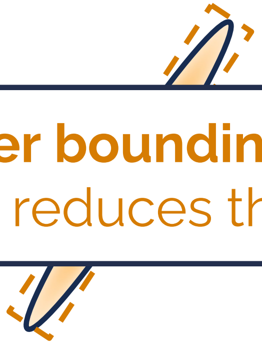
: **Shape-Aware** Intersection Test

Axis-Aligned Bounding Box (AABB)



A **large gap** between ellipse and bounding box ☹

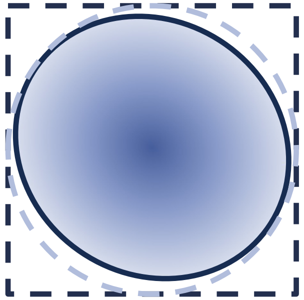
Tighter Bounding Box
(e.g., Oriented Bounding Box (OBB))

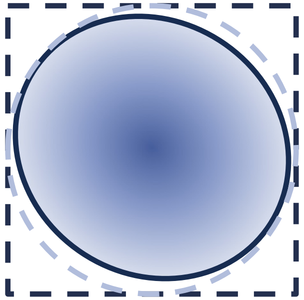


A **tighter bounding box** effectively reduces the gap! ☺

OR

Axis-Aligned Bounding Box (AABB)





Opportunistically apply
only for skewed Gaussians

Gaussian Shape-Aware Intersection Test

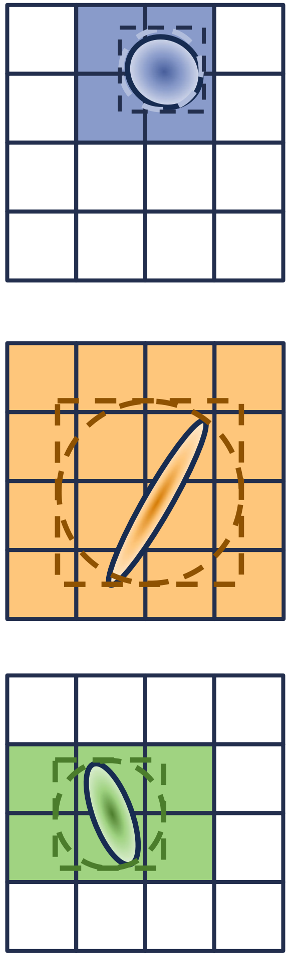
Original

: **AABB-Based** Intersection Test

vs.

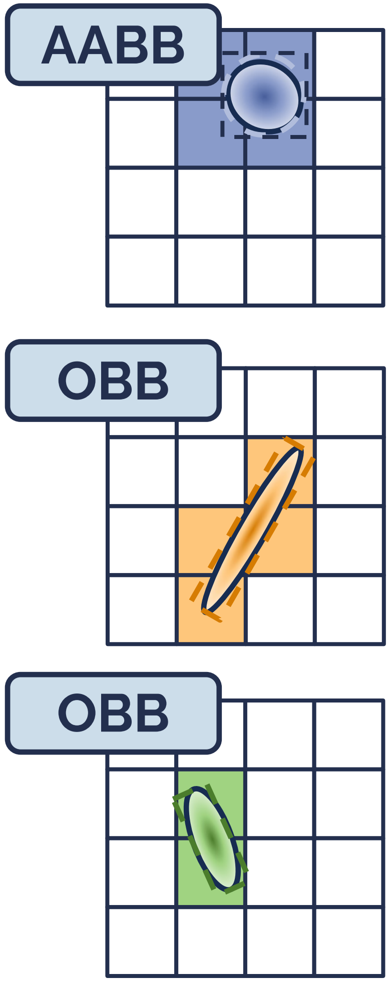
GSCore

: **Shape-Aware** Intersection Test



Total = **26**
tiles intersected

Total = **10**
tiles intersected

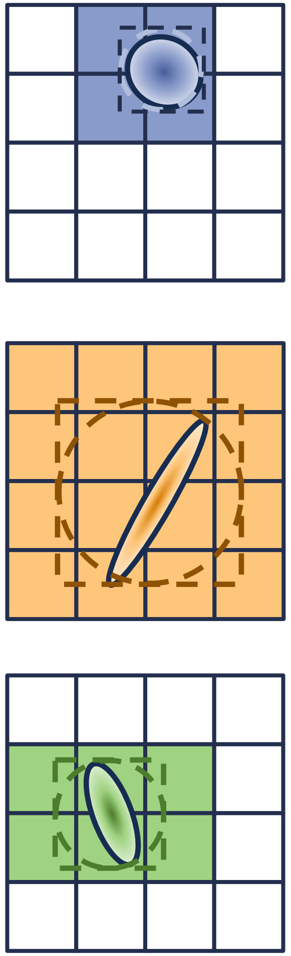


Gaussian Shape-Aware Intersection Test

Original
: **AAABB-Based** Intersection Test

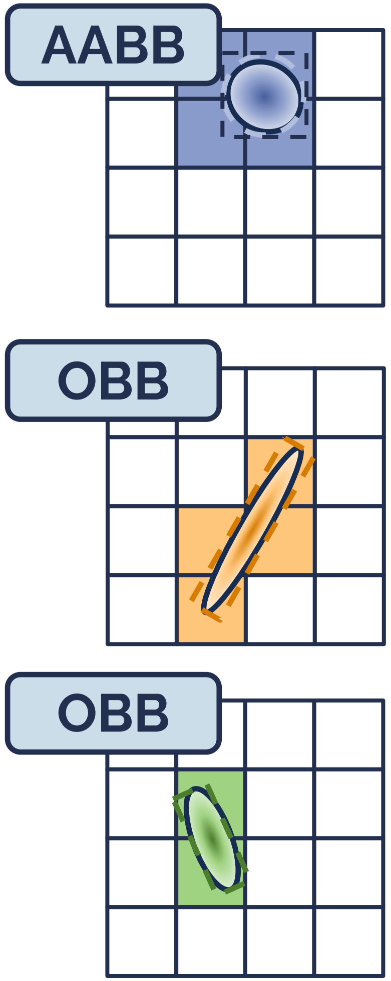
vs.

GSCore
: **Shape-Aware** Intersection Test



Total = **26**
tiles intersected

Total = **10**
tiles intersected

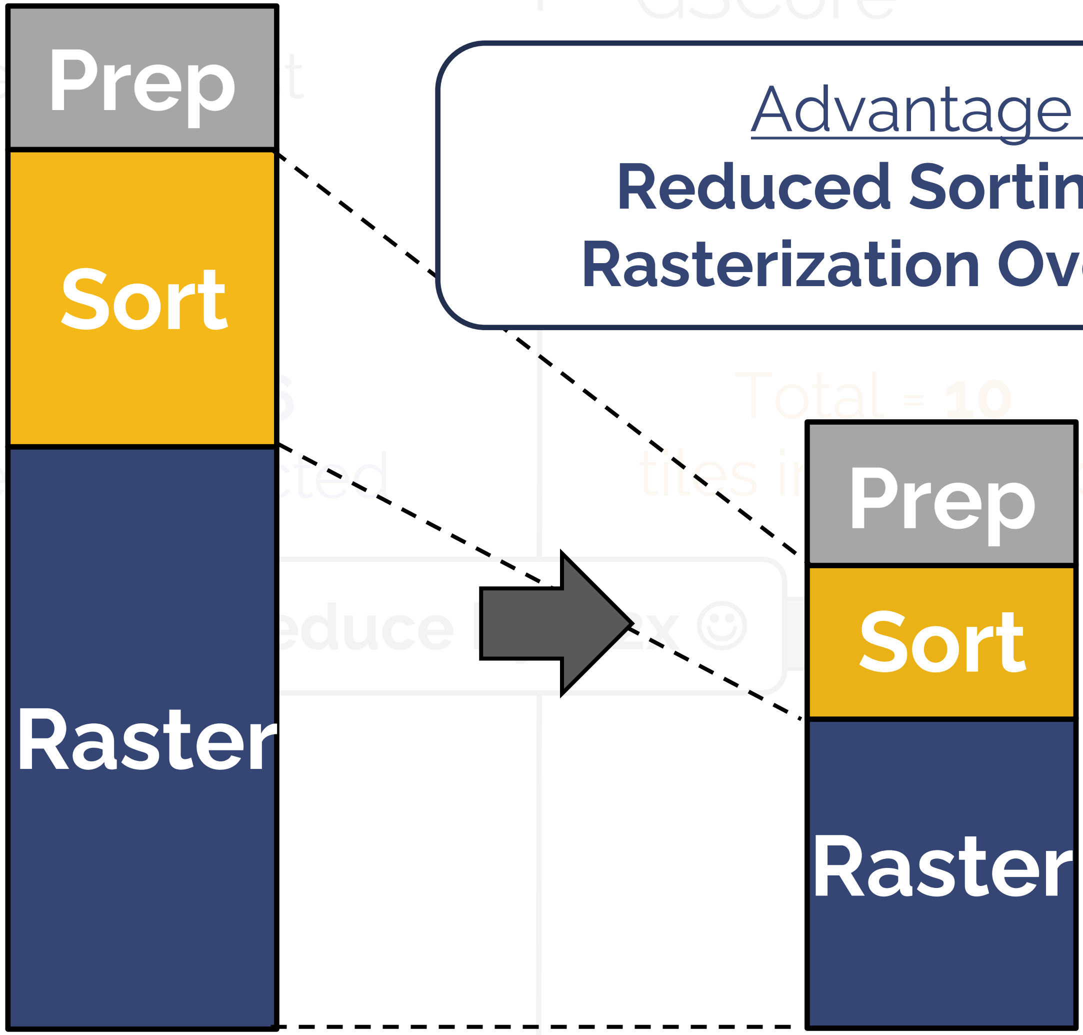


Reduce by > 2x ☺

Gaussian Shape-Aware Intersection Test

Original

GSCore

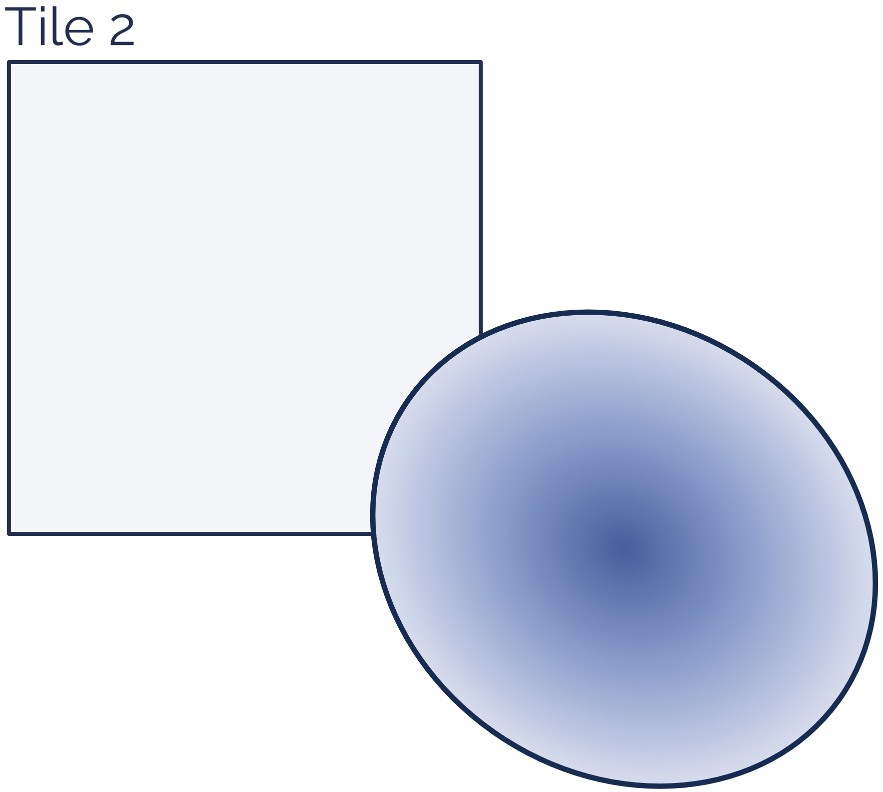


Advantage 1.

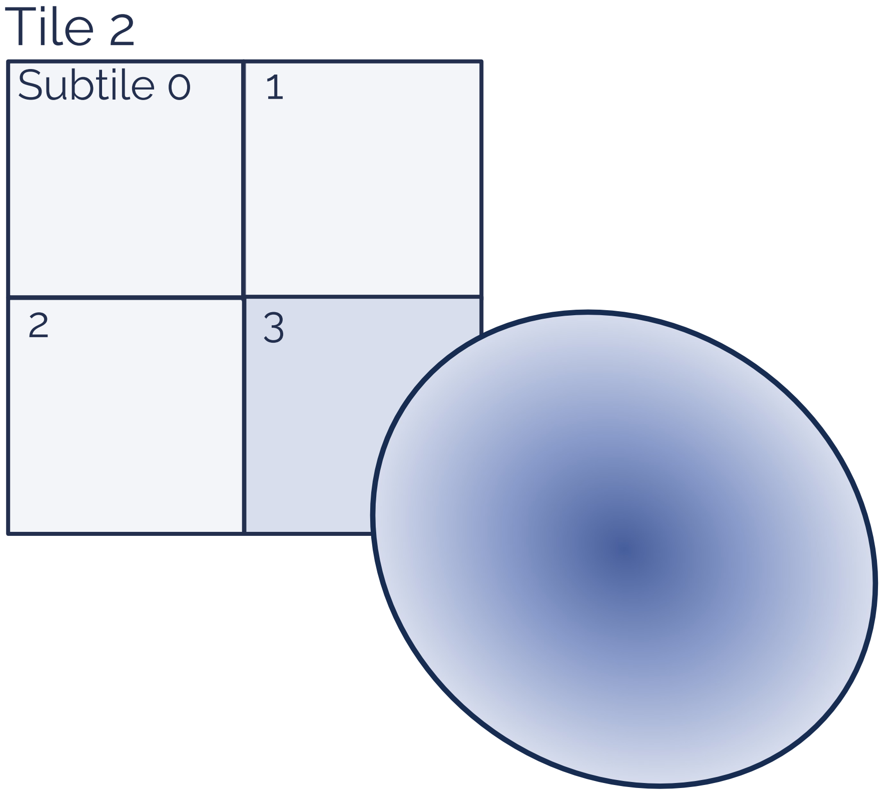
Reduced Sorting and Rasterization Overhead

Total = 10

Subtile Skipping



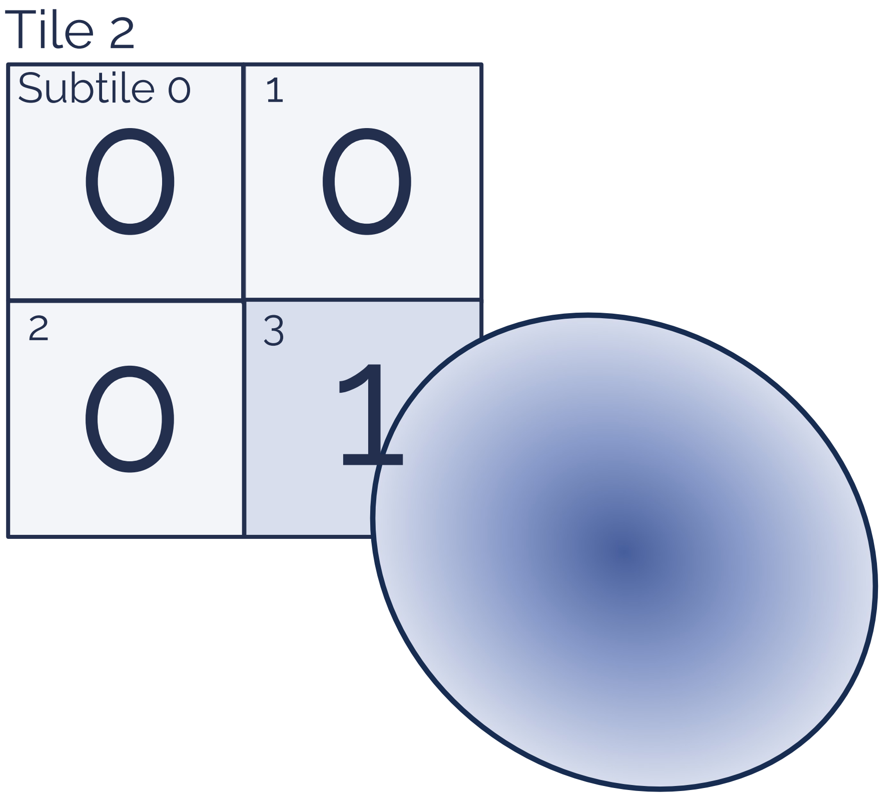
Subtile Skipping



Subtile Skipping

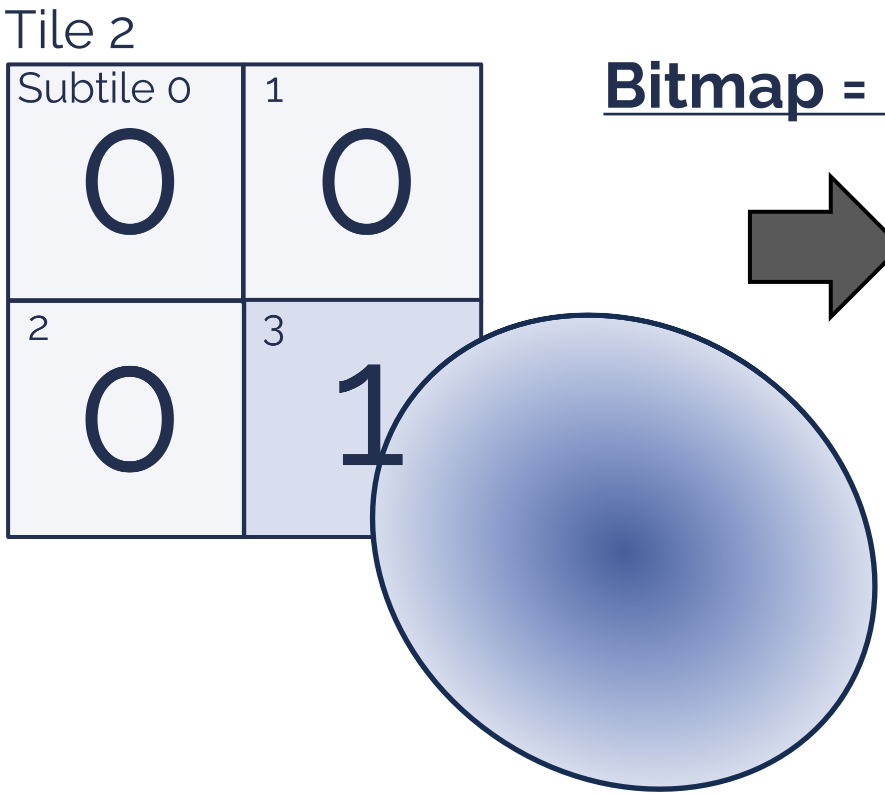
@ **Preprocessing**

: Generate a **subtile bitmap**



Subtile Skipping

@ **Preprocessing**
: Generate a **subtile bitmap**



Bitmap = 0001

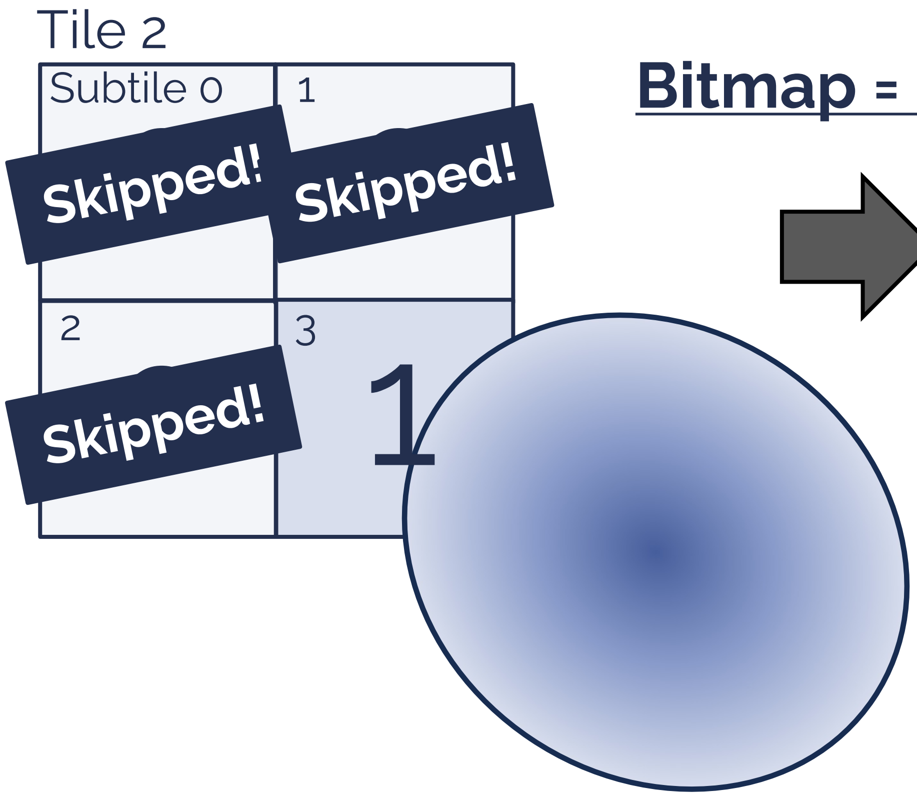
Pixel Rendering

```
for Gaus in { | Gaus 1 | Bitmap 0001 | , ... }
    2) $\alpha$-Computation
    3) Boundary Checking & Early Term.
    4) $\alpha$-Blending
```

Subtile Skipping

@ **Preprocessing**
: Generate a **subtile bitmap**

@ **Rasterization**
: **Skip the subtile** using the subtile bitmap



Bitmap = 0001

Pixel Rendering

```
for Gaus in { | Gaus 1 | Bitmap 0001 | , ... }
    1) Subtile Skipping
    2) α-Computa
    3) B                                   Term.
    4) α-
```

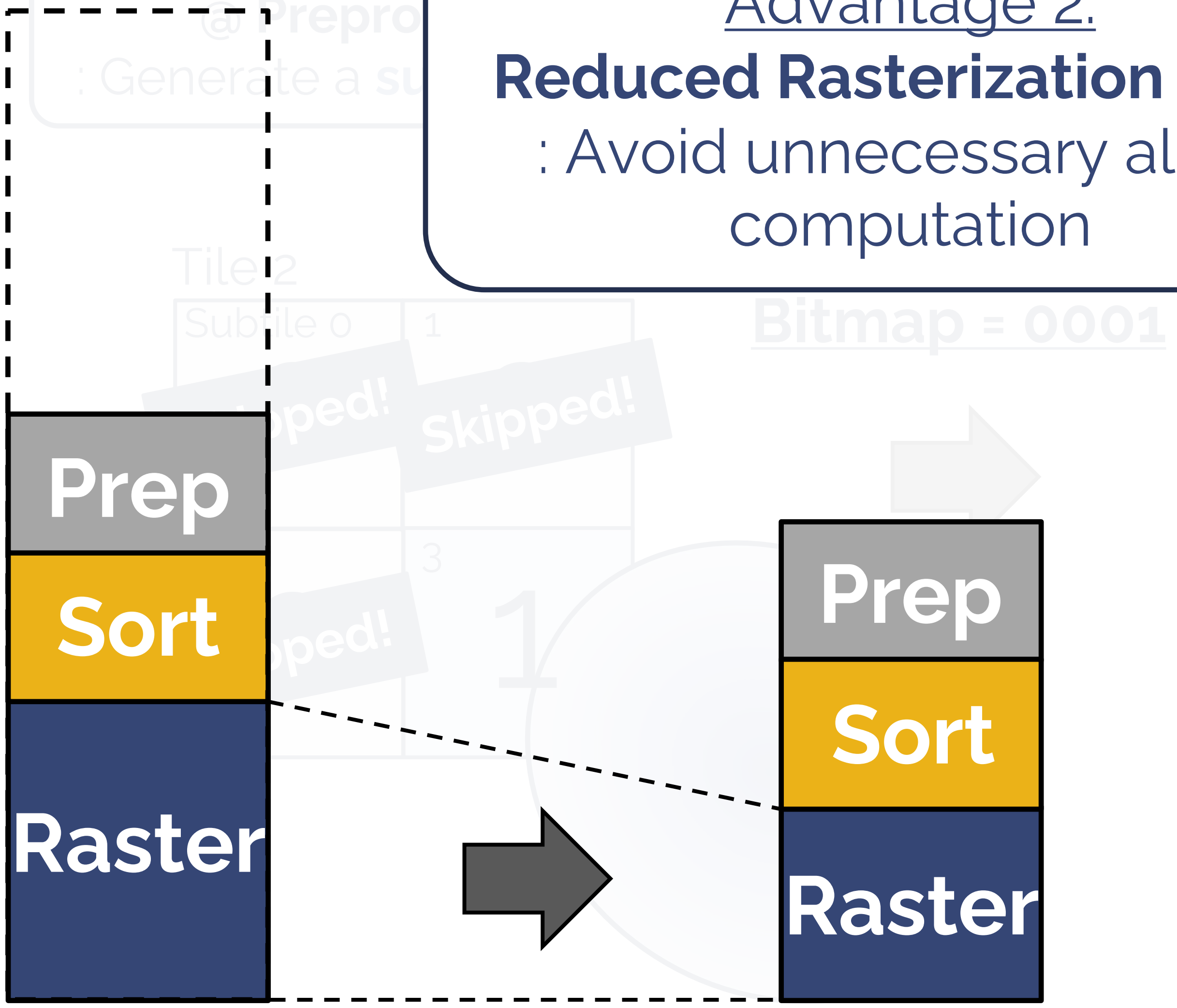
Skipped!

Subtile Skipping

Advantage 2.

Reduced Rasterization Time

: Avoid unnecessary alpha computation



Pixel Rendering

```
for Gaus in { | Gaus 1 | Bitmap 0001 | , ... }
    1) Subtile Skipping
    2) α-Computati
    3) B                                  Term.
    4) α-
```

Skipped!

Hierarchical Sorting

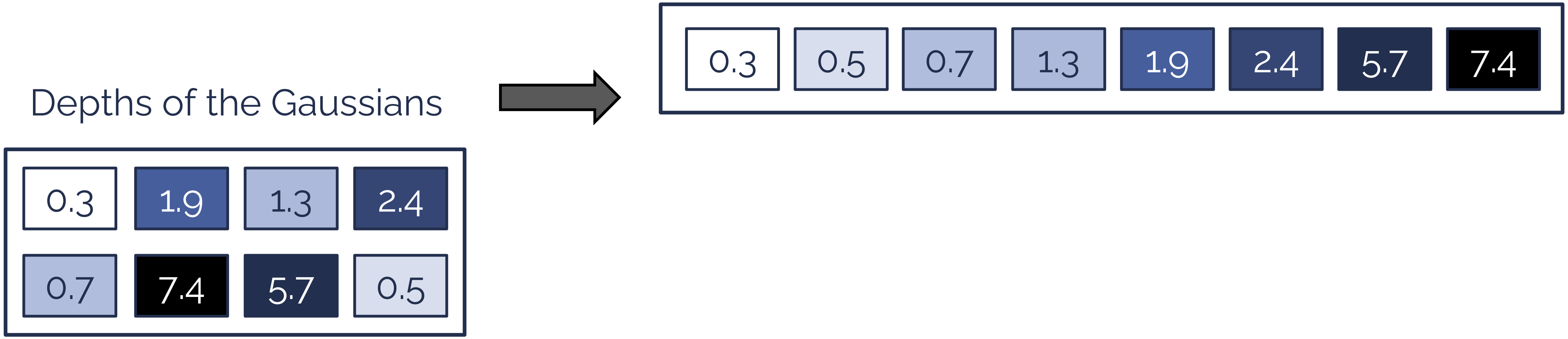
Hierarchical Sorting

Depths of the Gaussians

0.3	1.9	1.3	2.4
0.7	7.4	5.7	0.5

Hierarchical Sorting

Original: **Global Sorting**



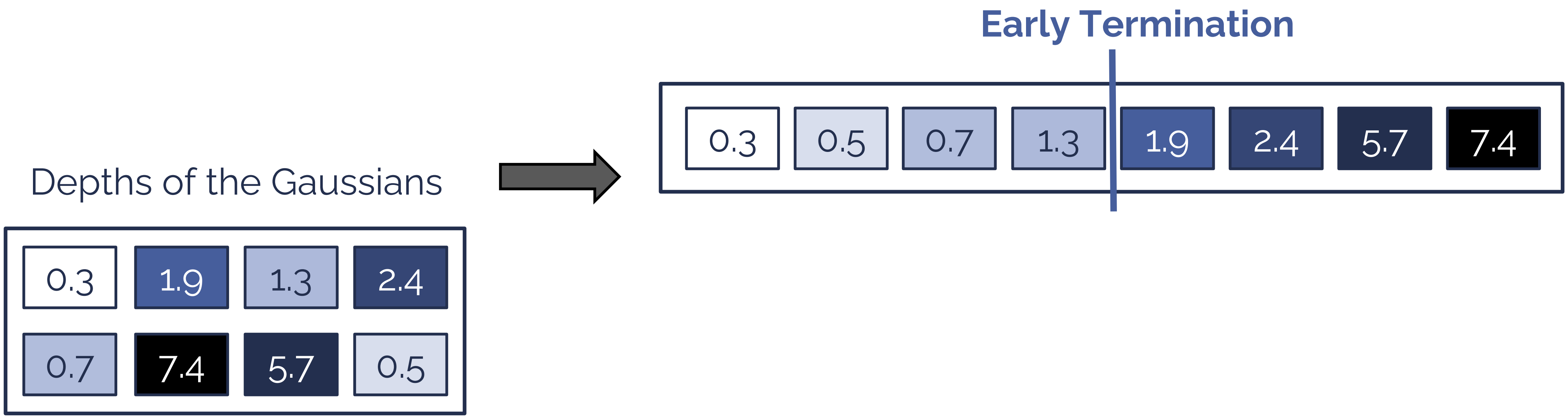
Depths of the Gaussians

0.3	1.9	1.3	2.4
0.7	7.4	5.7	0.5

0.3	0.5	0.7	1.3	1.9	2.4	5.7	7.4

Hierarchical Sorting

Original: **Global Sorting**



Early Termination

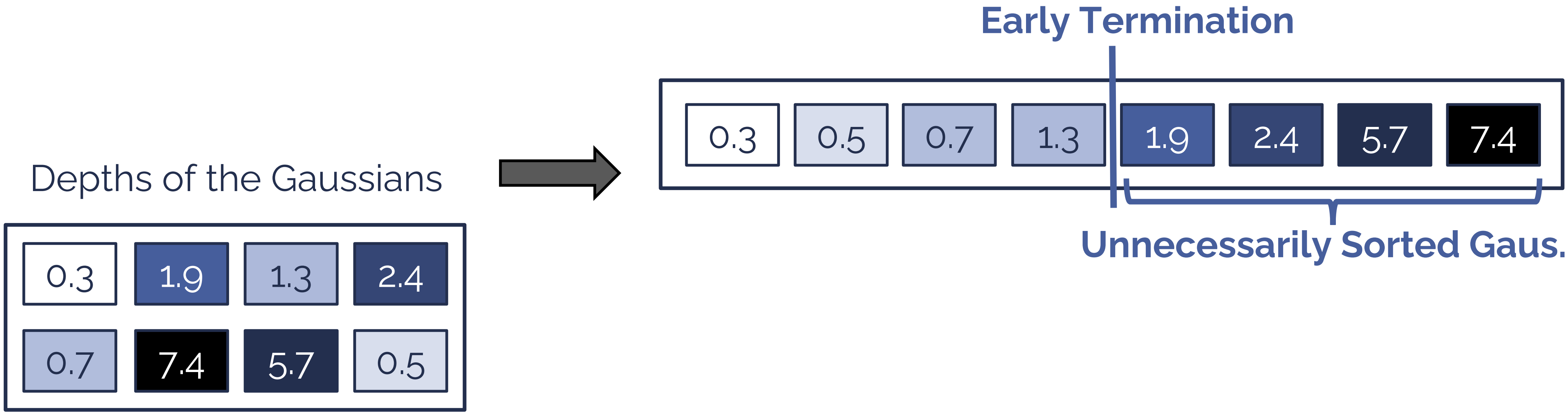
Depths of the Gaussians

0.3	1.9	1.3	2.4
0.7	7.4	5.7	0.5

0.3	0.5	0.7	1.3	1.9	2.4	5.7	7.4

Hierarchical Sorting

Original: **Global Sorting**



Depths of the Gaussians

0.3	1.9	1.3	2.4
0.7	7.4	5.7	0.5

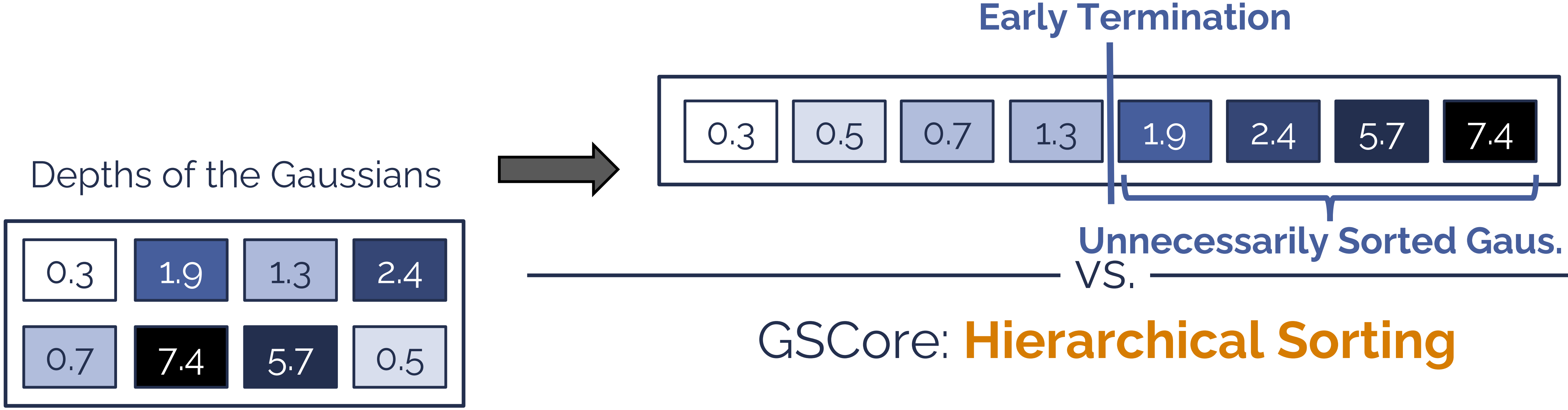
Early Termination

0.3	0.5	0.7	1.3	1.9	2.4	5.7	7.4

Unnecessarily Sorted Gaus.

Hierarchical Sorting

Original: **Global Sorting**



Depths of the Gaussians

0.3	1.9	1.3	2.4
0.7	7.4	5.7	0.5

Early Termination

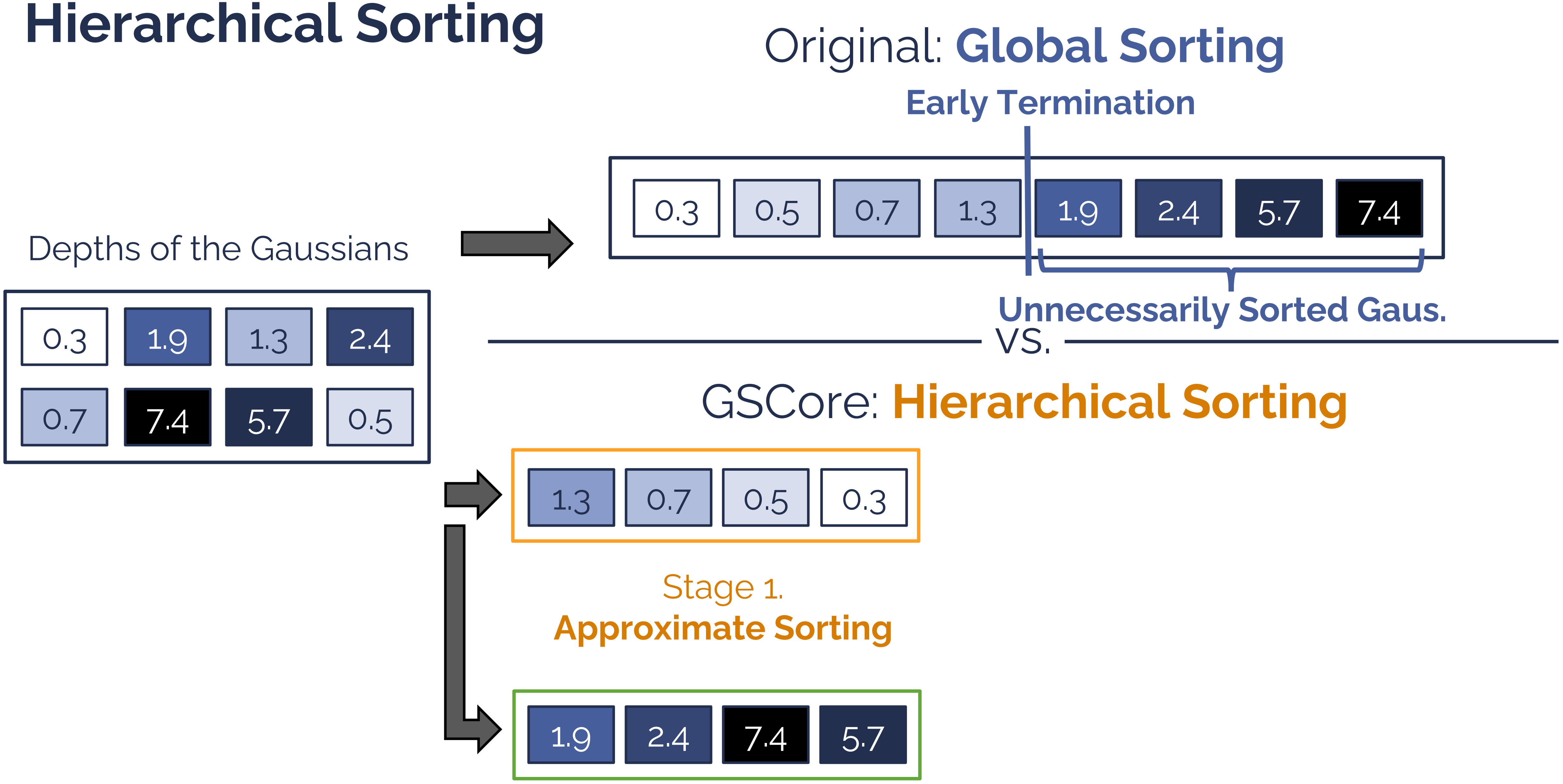
0.3	0.5	0.7	1.3	1.9	2.4	5.7	7.4

Unnecessarily Sorted Gaus.

VS.

GSCore: **Hierarchical Sorting**

Hierarchical Sorting



Depths of the Gaussians

0.3	1.9	1.3	2.4
0.7	7.4	5.7	0.5

Original: **Global Sorting**

Early Termination

0.3	0.5	0.7	1.3	1.9	2.4	5.7	7.4

Unnecessarily Sorted Gaus.

VS.

GSCore: **Hierarchical Sorting**

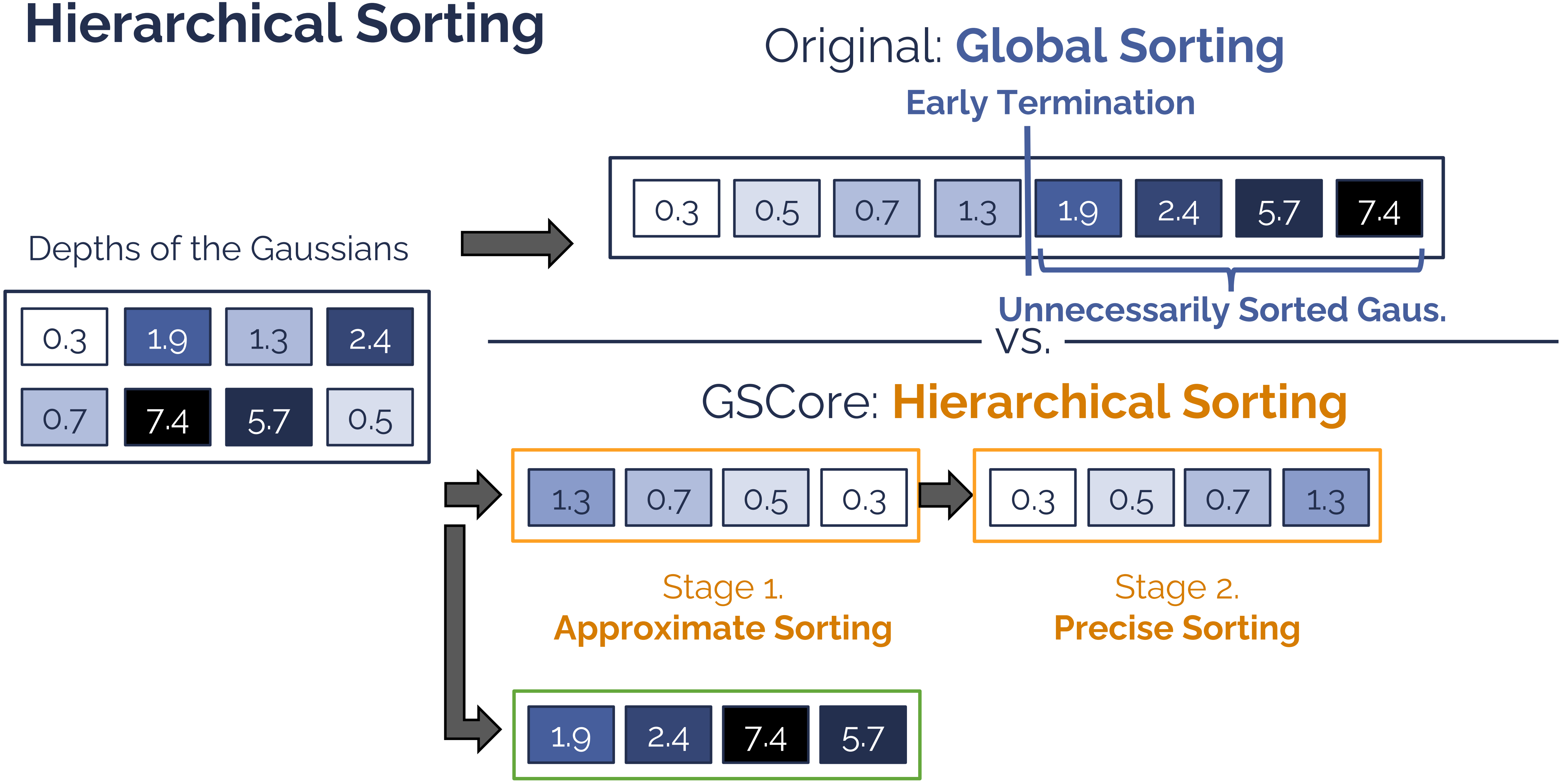
1.3	0.7	0.5	0.3

Stage 1.

Approximate Sorting

1.9	2.4	7.4	5.7

Hierarchical Sorting



Original: **Global Sorting**

Depths of the Gaussians

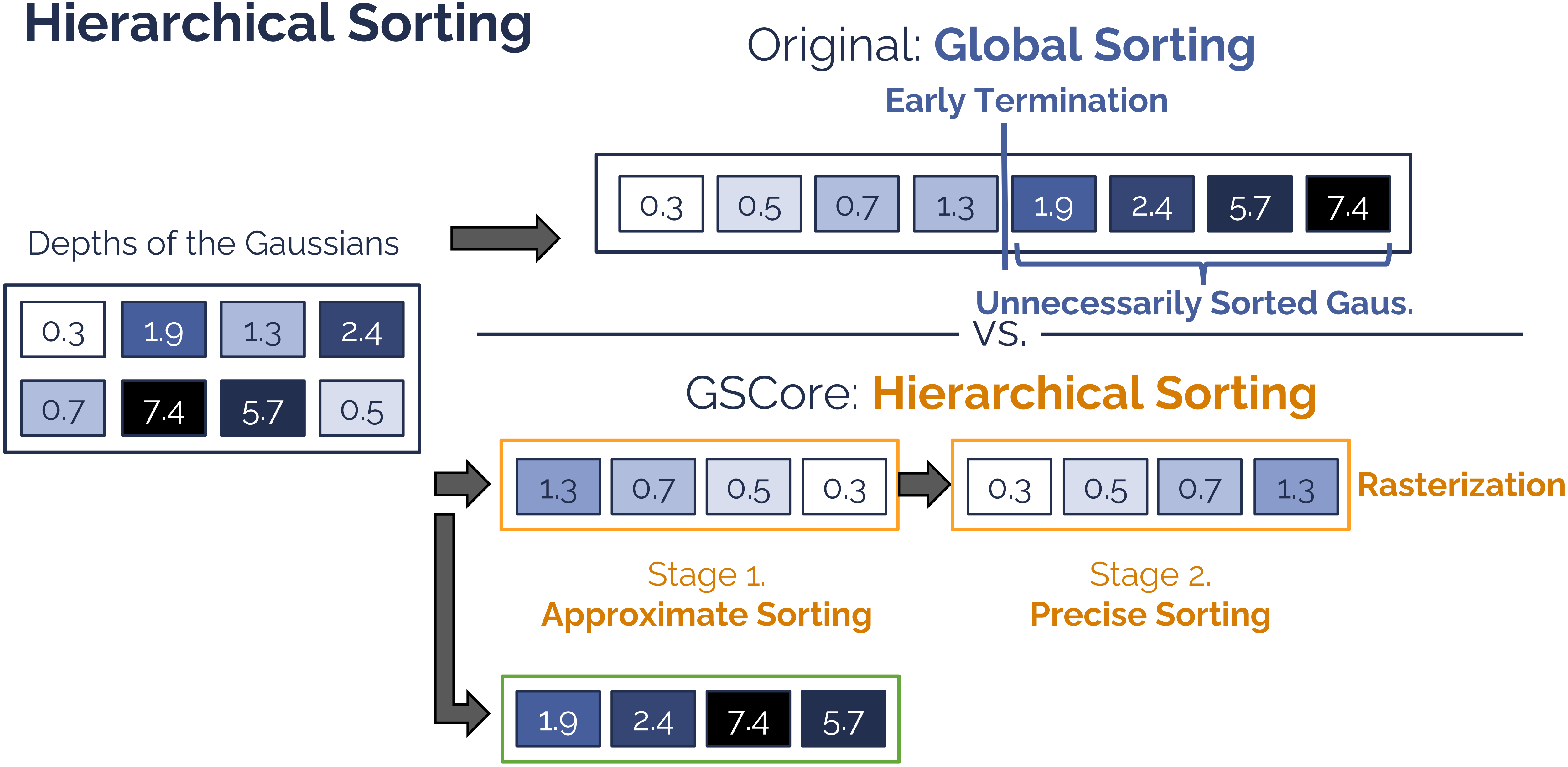
VS.

GSCore: **Hierarchical Sorting**

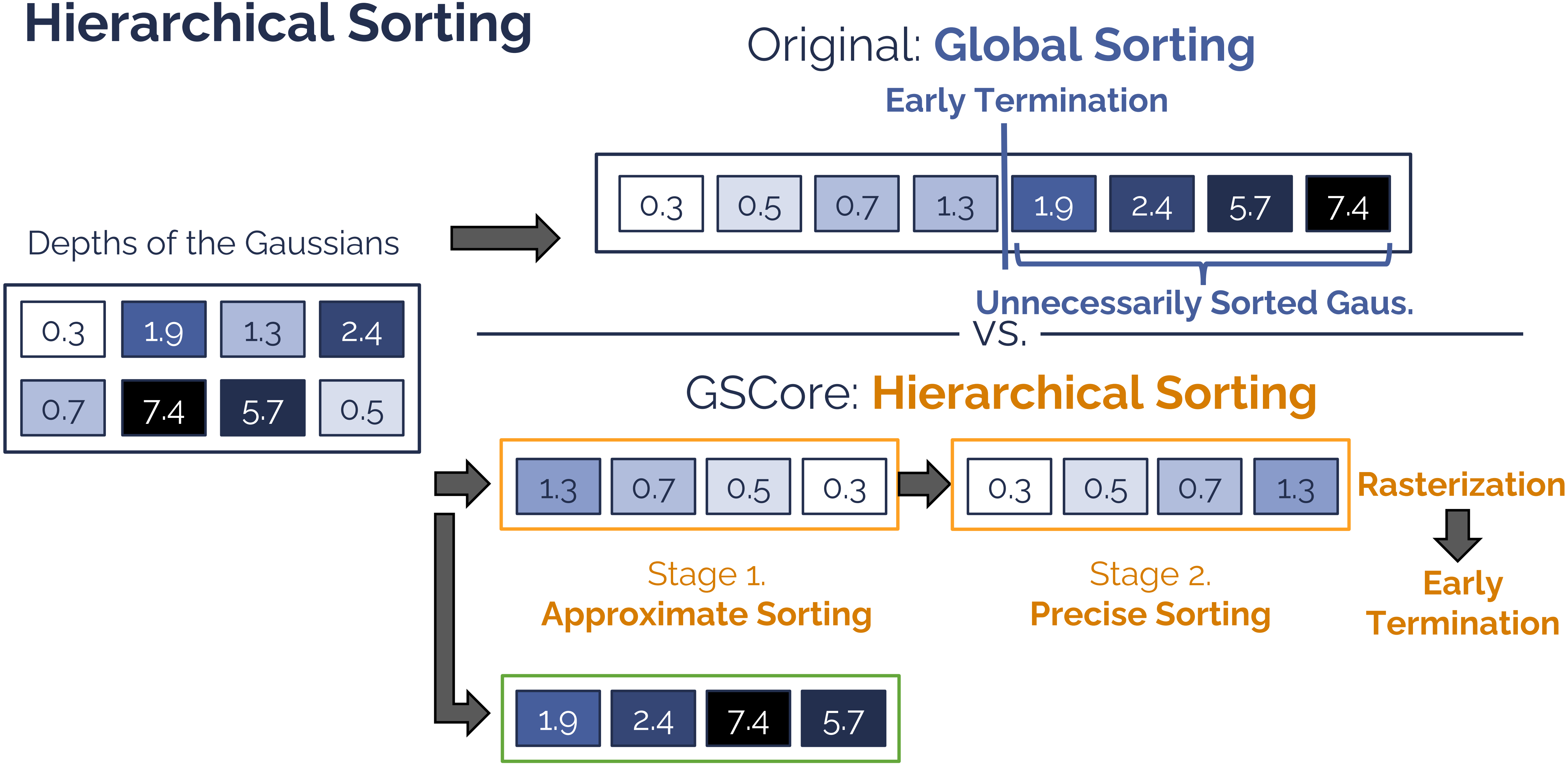
Stage 1.
Approximate Sorting

Stage 2.
Precise Sorting

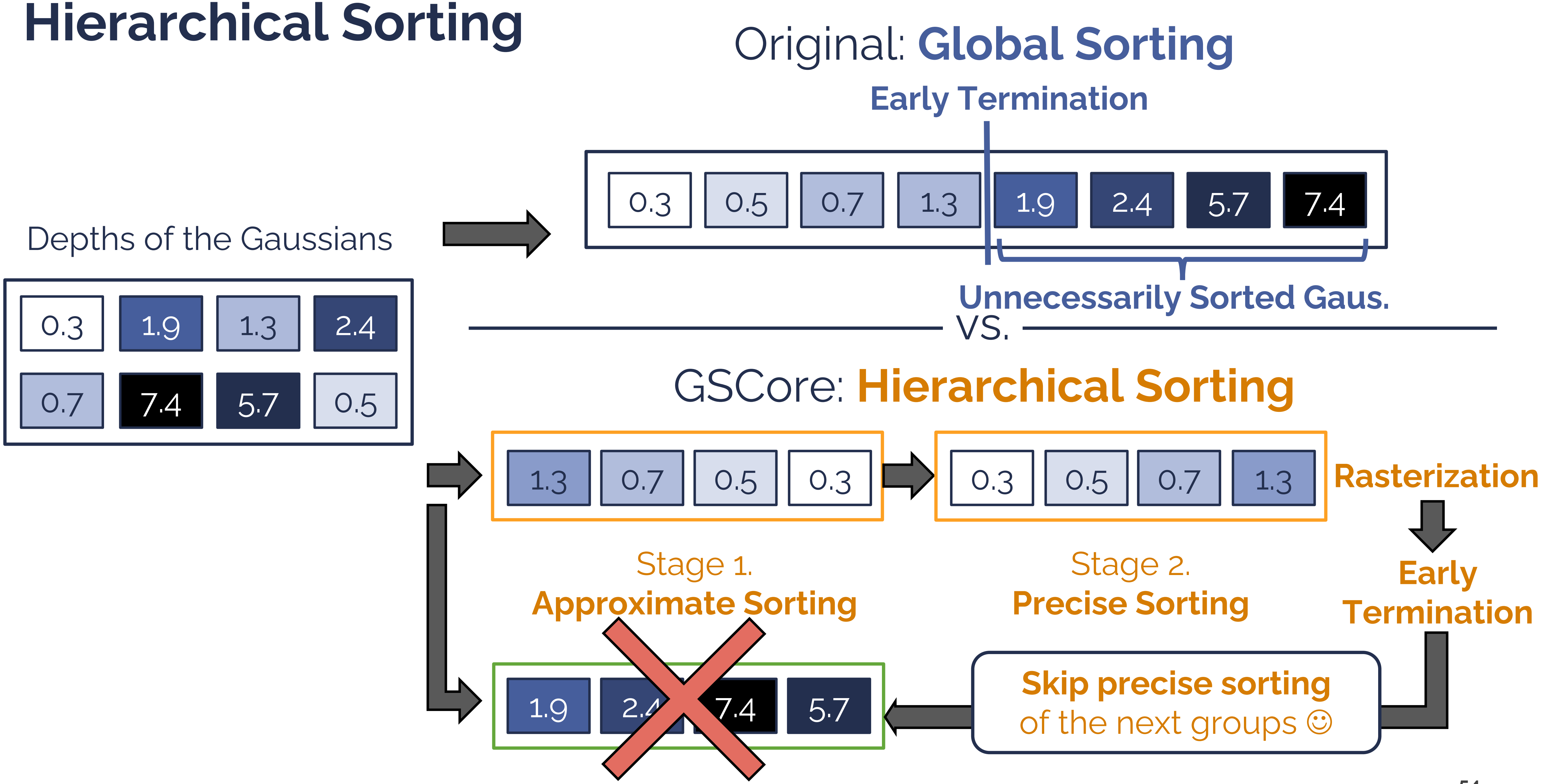
Hierarchical Sorting



Hierarchical Sorting



Hierarchical Sorting



Original: **Global Sorting**

Early Termination

0.3 0.5 0.7 1.3 | 1.9 2.4 5.7 7.4

Unnecessarily Sorted Gaus.

Depths of the Gaussians

0.3 1.9 1.3 2.4

0.7 7.4 5.7 0.5

VS.

GSCore: **Hierarchical Sorting**

1.3 0.7 0.5 0.3

0.3 0.5 0.7 1.3

Rasterization

Stage 1.
Approximate Sorting

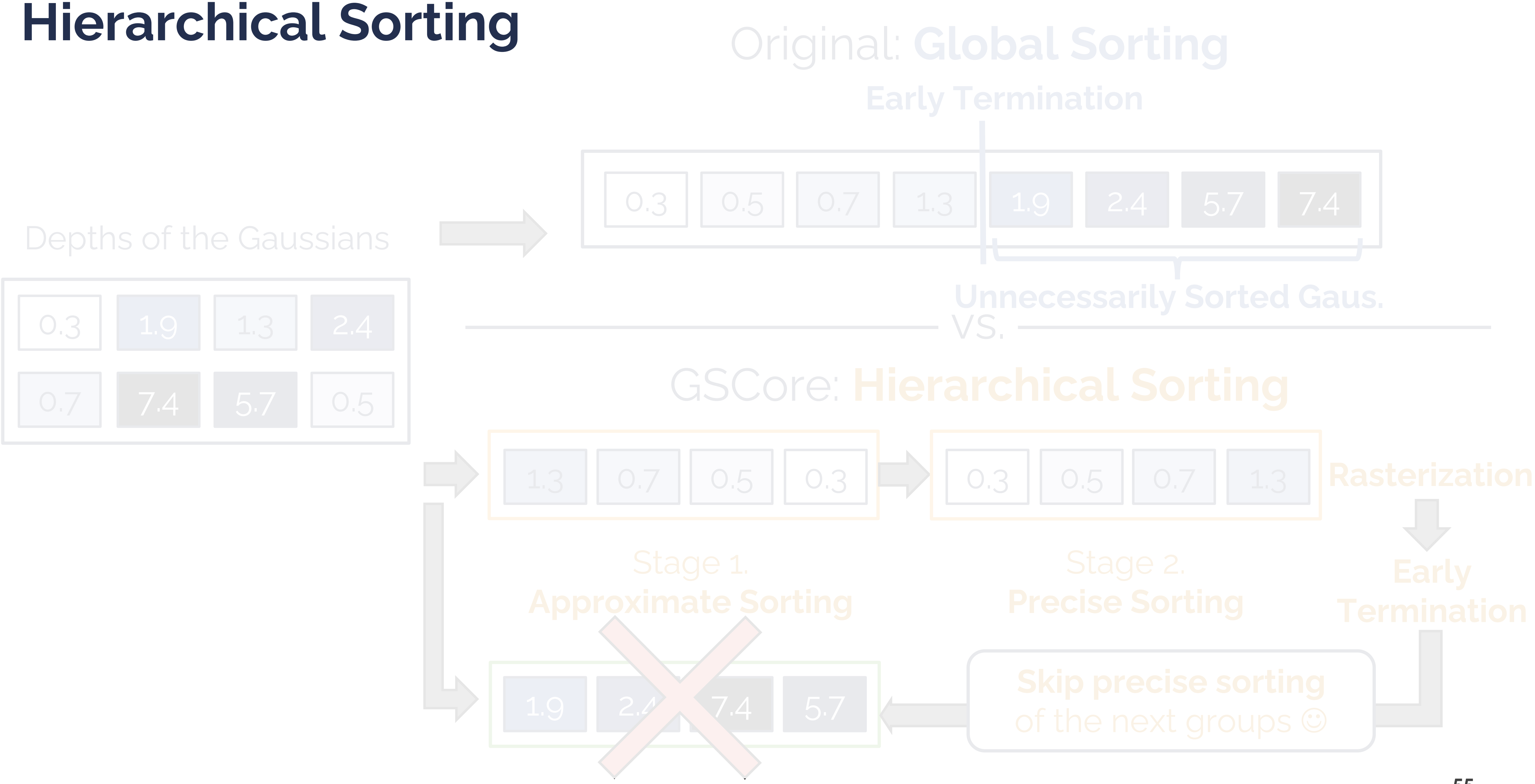
Stage 2.
Precise Sorting

Early Termination

1.9 2.4 7.4 5.7

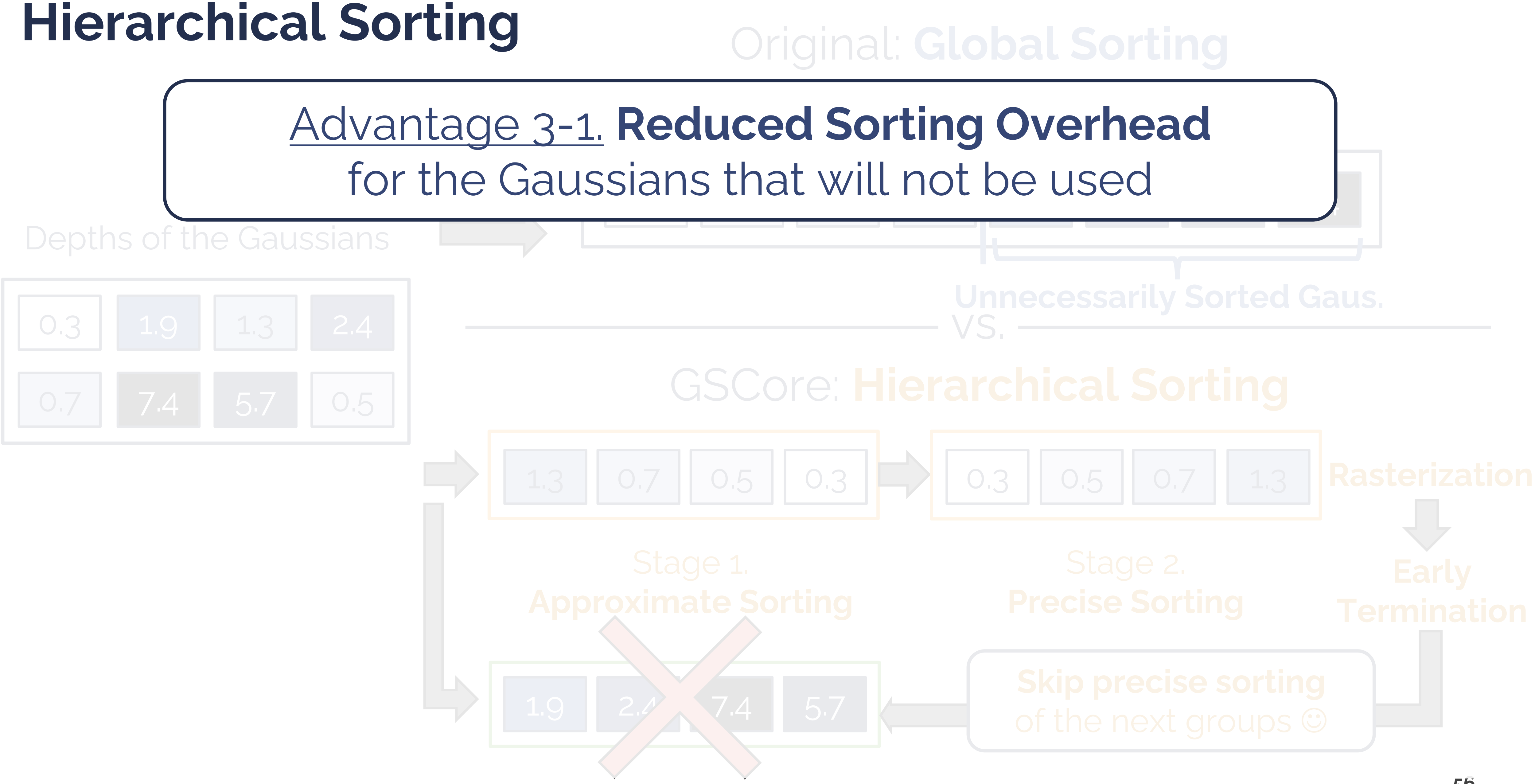
Skip precise sorting
of the next groups ☺

Hierarchical Sorting



Hierarchical Sorting

Advantage 3-1. **Reduced Sorting Overhead**
for the Gaussians that will not be used



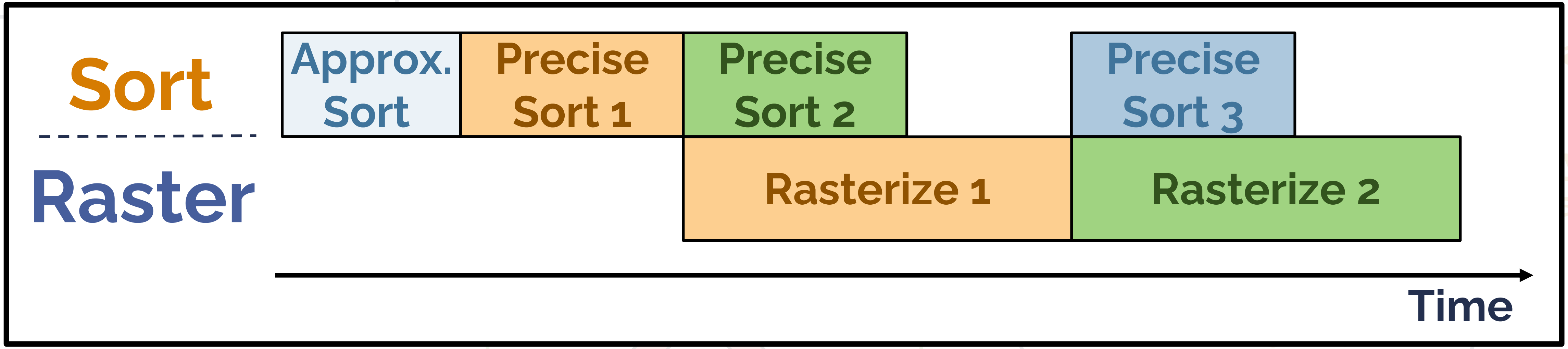
Hierarchical Sorting

Original: **Global Sorting**

Depths of the Gaussians

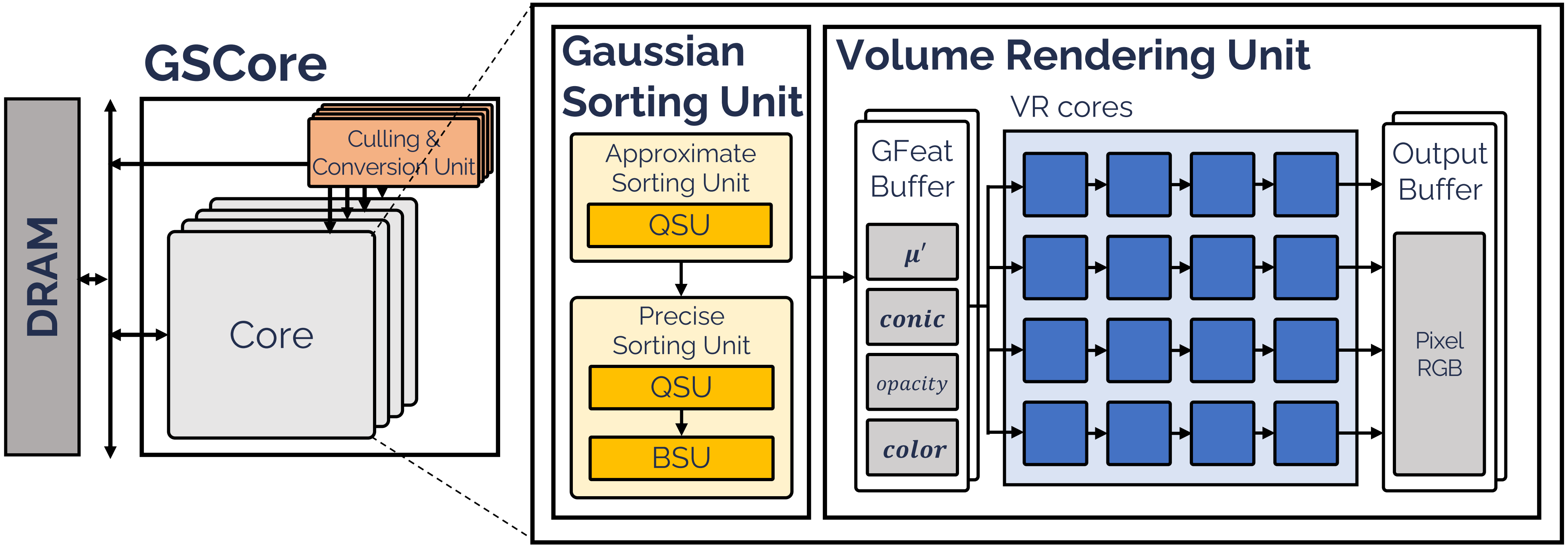
Advantage 3-1. **Reduced Sorting Overhead**
for the Gaussians that will not be used

Advantage 3-2. **Hide Precise Sorting Time**
by execution overlap of sorting and rasterization



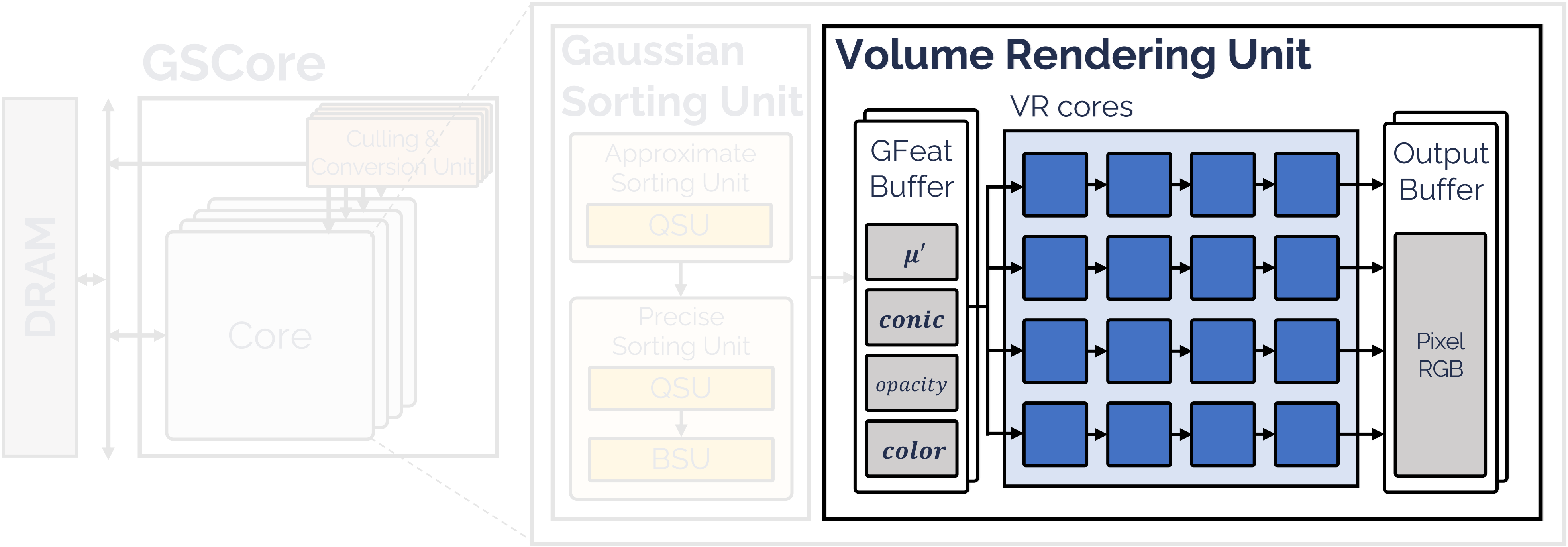
GSCore: Rendering Acceleration Unit

1. Preprocessing in **Culling & Conversion Unit**
2. Gaussian Sorting in **Gaussian Sorting Unit**
3. Rasterization in **Volume Rendering Unit**

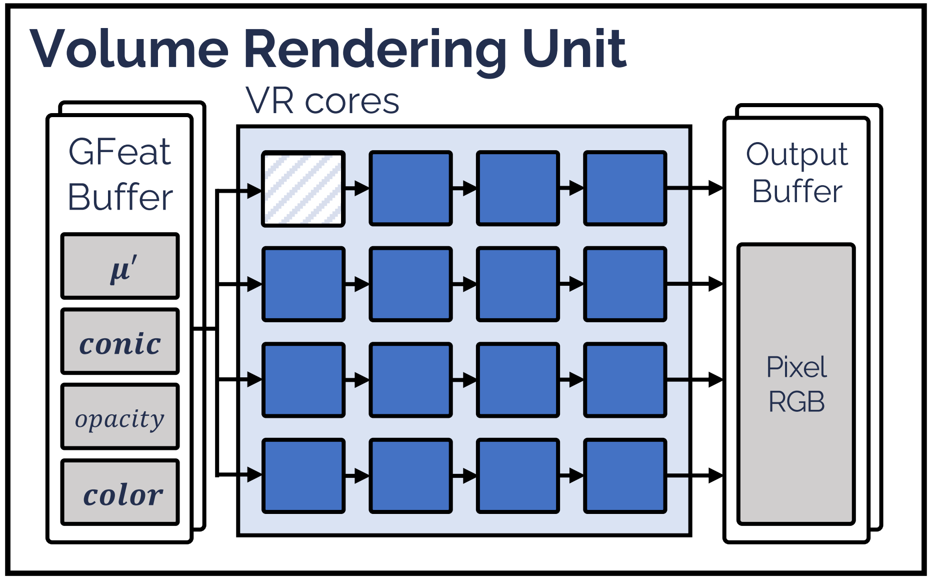


GSCore: Rendering Acceleration Unit

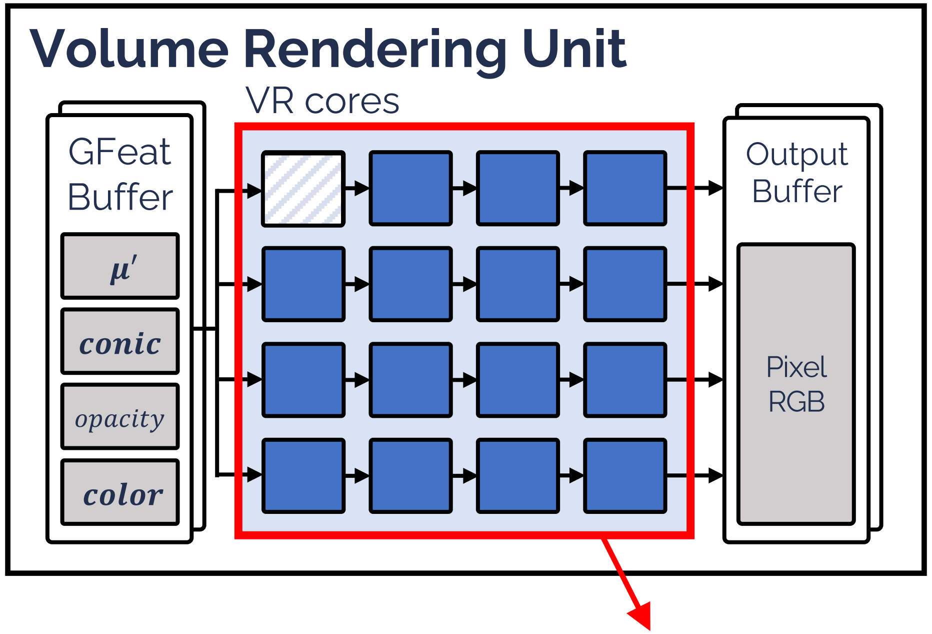
1. Preprocessing in **Culling & Conversion Unit**
2. Gaussian Sorting in **Gaussian Sorting Unit**
3. Rasterization in **Volume Rendering Unit**



Volume Rendering Unit

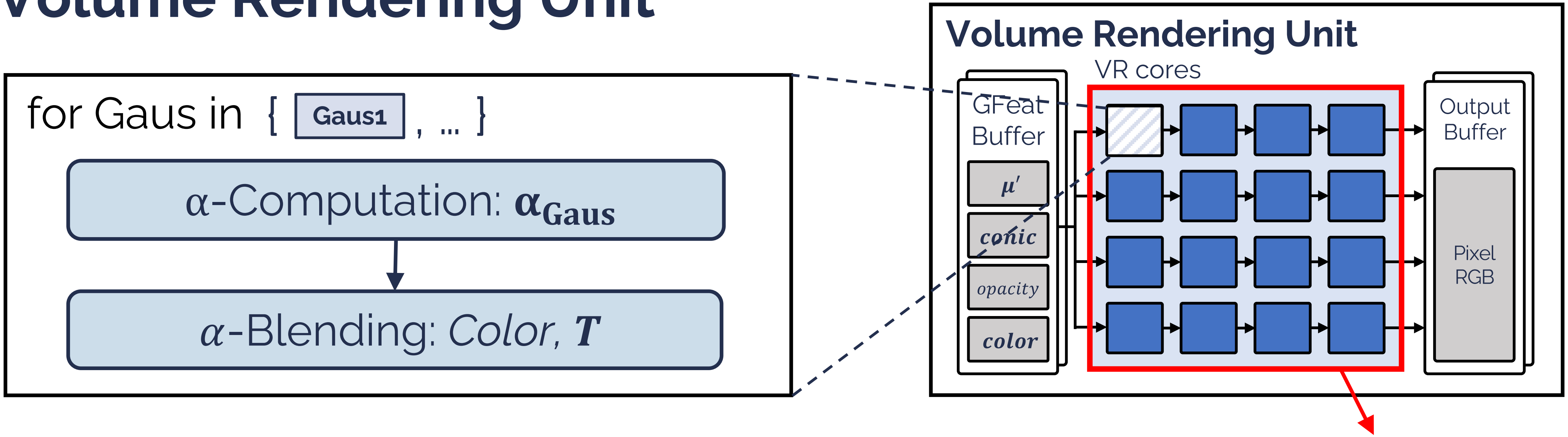


Volume Rendering Unit



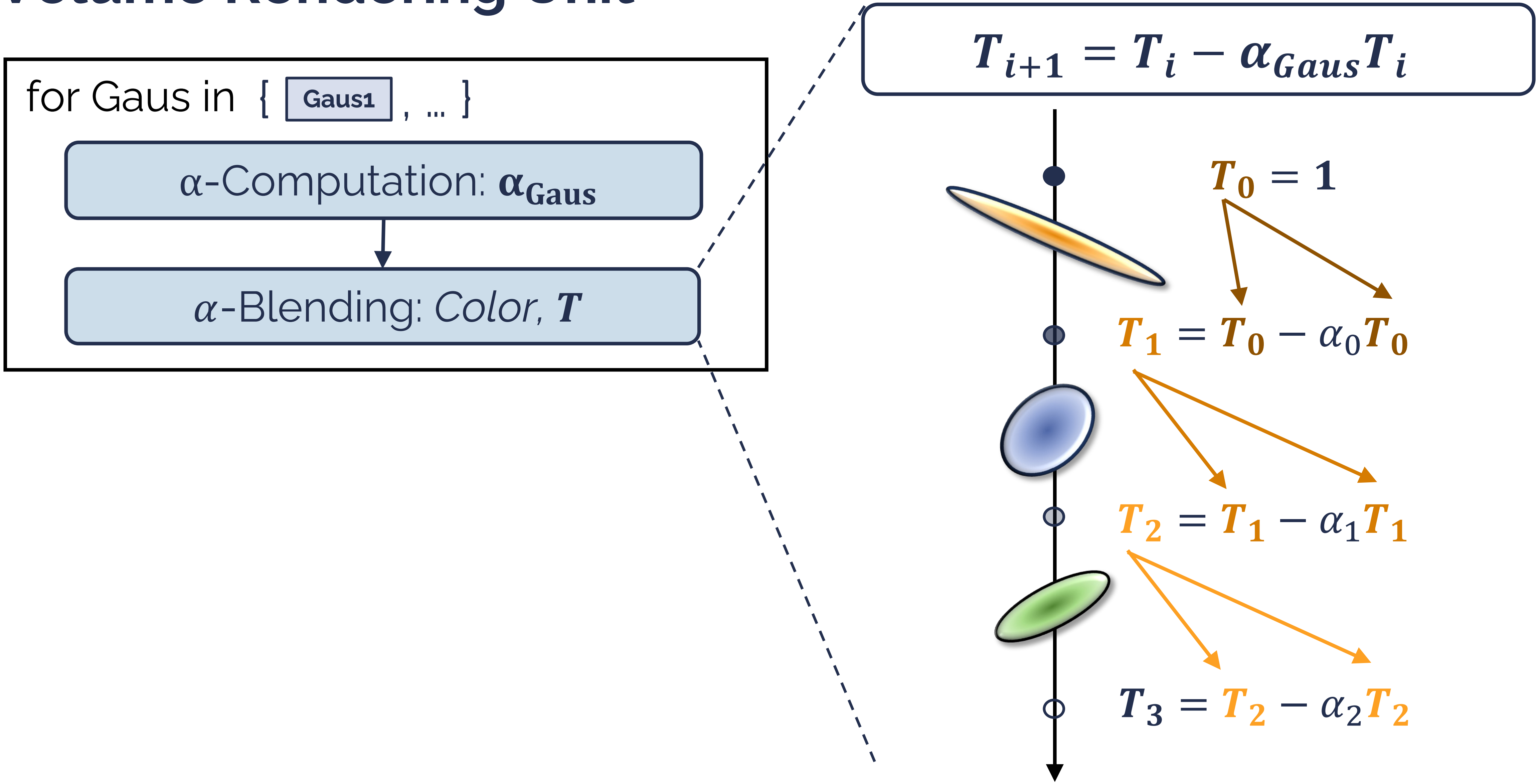
Subtile Rendering

Volume Rendering Unit


for Gaus in { Gaus1 , … }
α-Computation: $\boldsymbol{\alpha}_{\mathbf{Gaus}}$
α-Blending: *Color*, $\boldsymbol{T}$
Volume Rendering Unit
VR cores
GFeat Buffer
μ'
conic
opacity
color
Output Buffer
Pixel RGB

Subtile Rendering

Volume Rendering Unit



for Gaus in { Gaus1 , ... }

α-Computation: $\boldsymbol{\alpha}_{\mathbf{Gaus}}$

α-Blending: *Color*, $\boldsymbol{T}$

$$T_{i+1} = T_i - \alpha_{Gaus} T_i$$

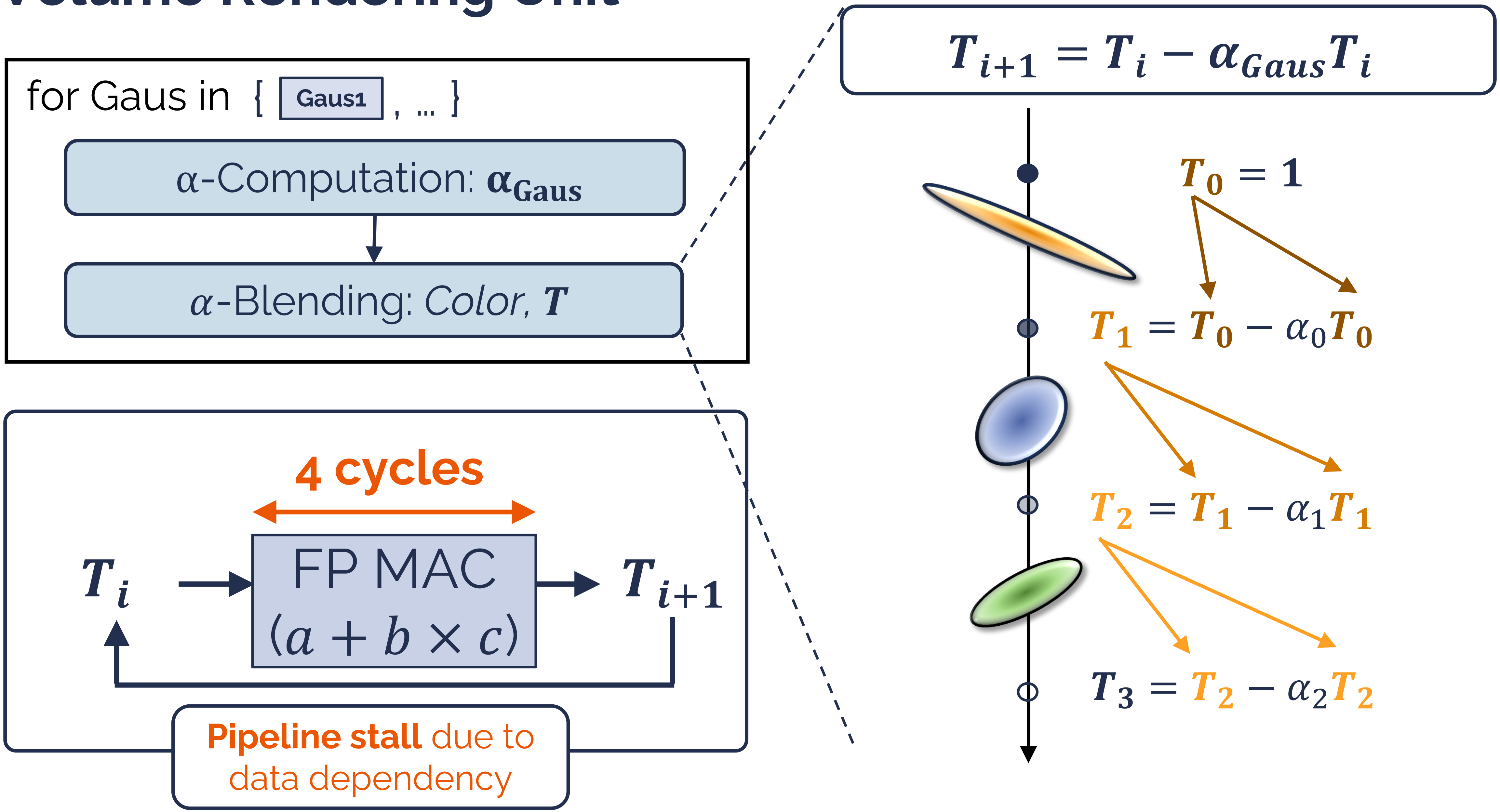
$$T_0 = 1$$

$$T_1 = T_0 - \alpha_0 T_0$$

$$T_2 = T_1 - \alpha_1 T_1$$

$$T_3 = T_2 - \alpha_2 T_2$$

Volume Rendering Unit



for Gaus in { Gaus1 , ... }

α-Computation: $\boldsymbol{\alpha}_{\mathbf{Gaus}}$

α-Blending: *Color*, $\boldsymbol{T}$

4 cycles

$\boldsymbol{T_i}$

FP MAC

$(a + b \times c)$

$\boldsymbol{T_{i+1}}$

Pipeline stall due to data dependency

$$\boldsymbol{T_{i+1} = T_i - \alpha_{Gaus} T_i}$$

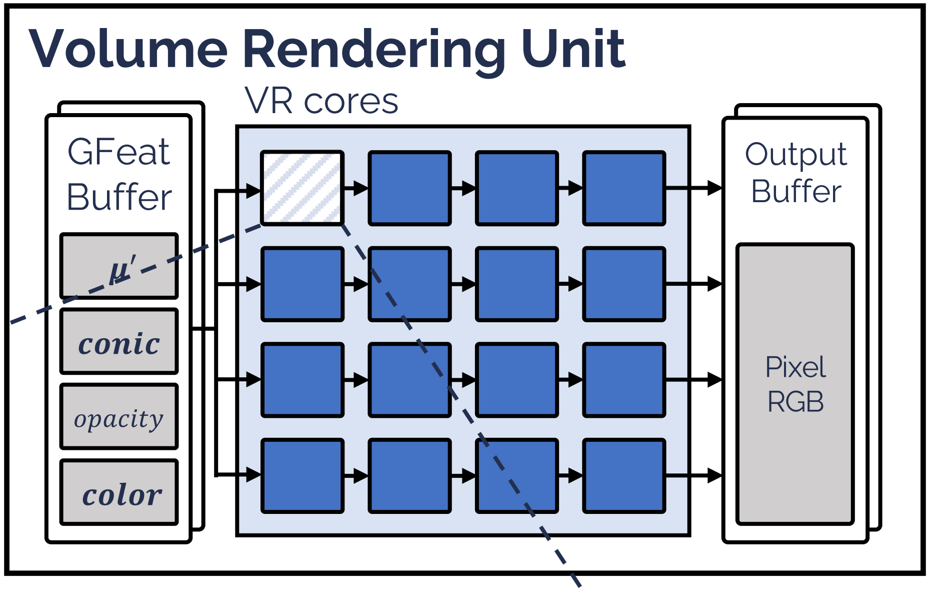
$$\boldsymbol{T_0 = 1}$$

$$\boldsymbol{T_1 = T_0 - \alpha_0 T_0}$$

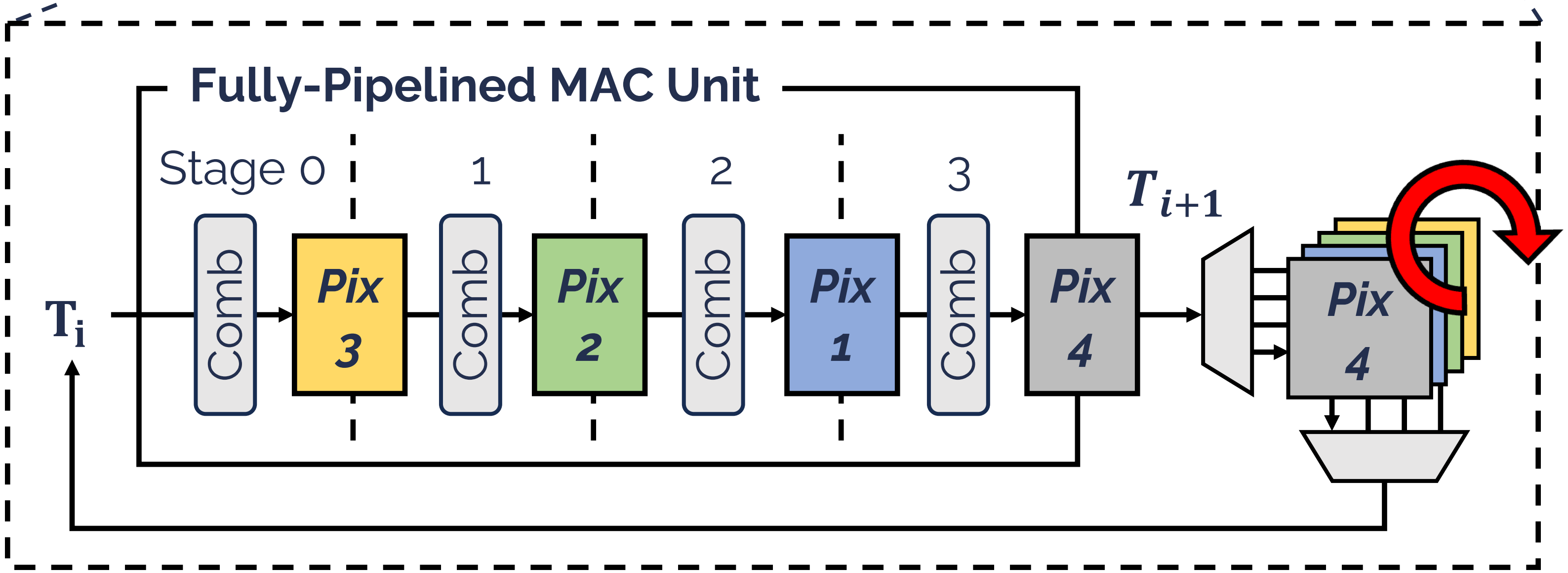
$$\boldsymbol{T_2 = T_1 - \alpha_1 T_1}$$

$$\boldsymbol{T_3 = T_2 - \alpha_2 T_2}$$

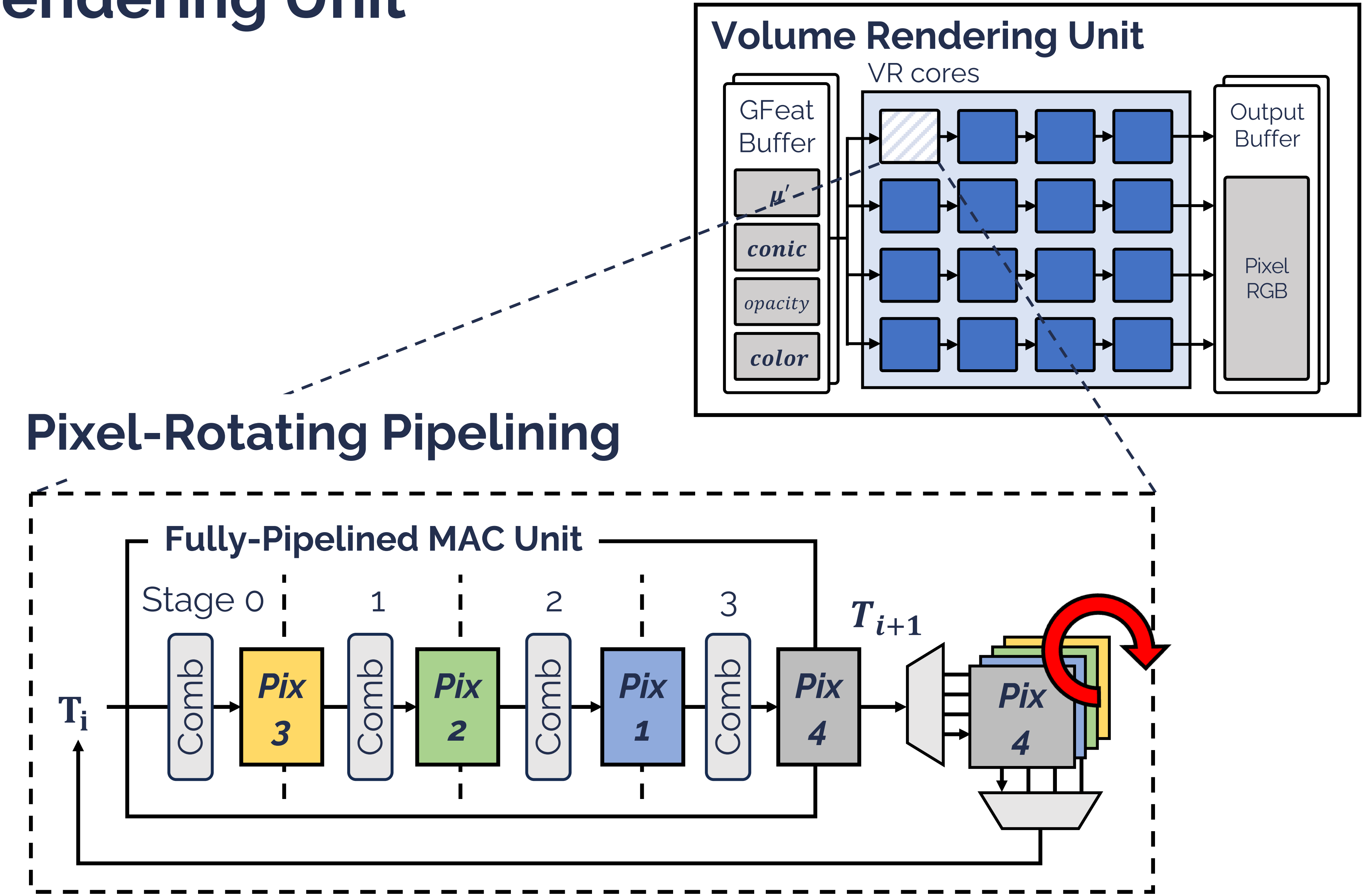
Volume Rendering Unit



Pixel-Rotating Pipelining



Volume Rendering Unit



Pixel-Rotating Pipelining

Outline

- Background
  - 3D Gaussian Splatting (3DGS)
- 3DGS Optimization & Inefficiencies
- GSCore: Efficient Radiance Field Rendering Accelerator
  - Algorithmic Optimizations
  - Hardware Architecture
- **Evaluation**
- Conclusion

Methodology

RTL Implementation

- Process node: 28nm technology

Baselines

- SW: Author-released 3DGS impl.
- HW: Jetson Xavier NX
  - GSCore: **3.95 mm^2** ⇔Xavier NX: **350 mm^2**

Performance Evaluation

- Cycle-level simulator
  - **with functional simulation**

Evaluated Workloads

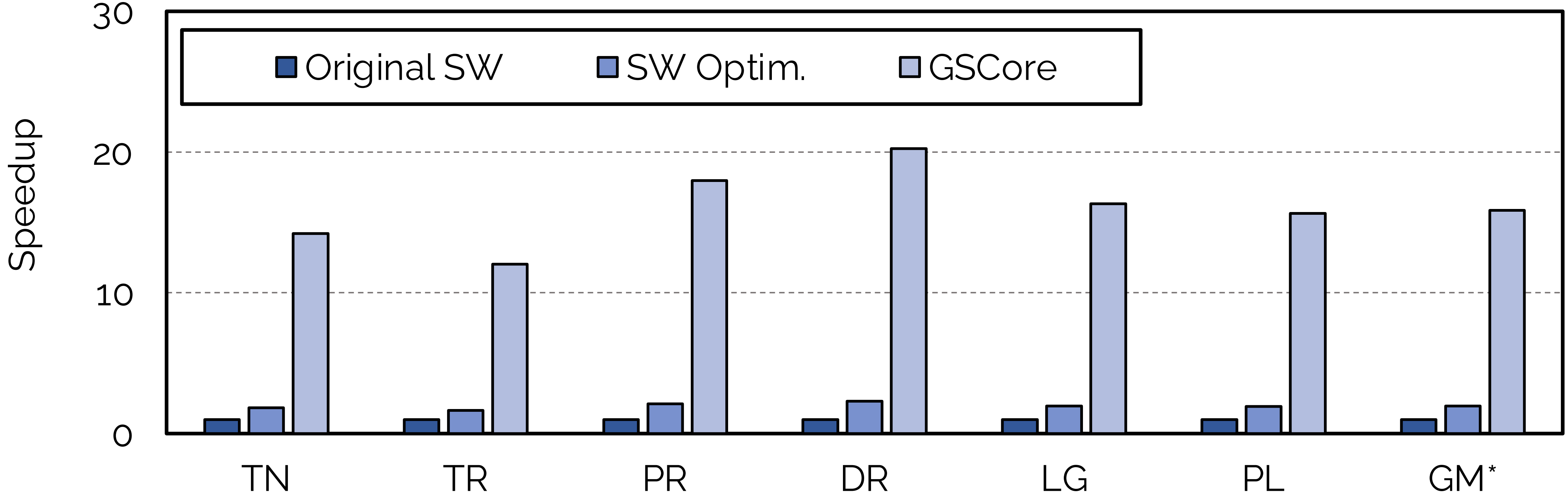
Dataset	Scene (Abbr.) (Resolution)	Type
Tanks& Temples	Train (TN) (980x545)	Real World & Outdoor
	Truck (TR) (979x546)	
Deep Blending	Playroom (PR) (1264x832)	Real World & Indoor
	Dr. Johnson (DR) (1332x876)	
Syn-NeRF	Lego (LG) (800x800)	Synthetic
Syn-NSVF	Palace (PL) (800x800)	

Performance

End-to-End Speedup

Performance

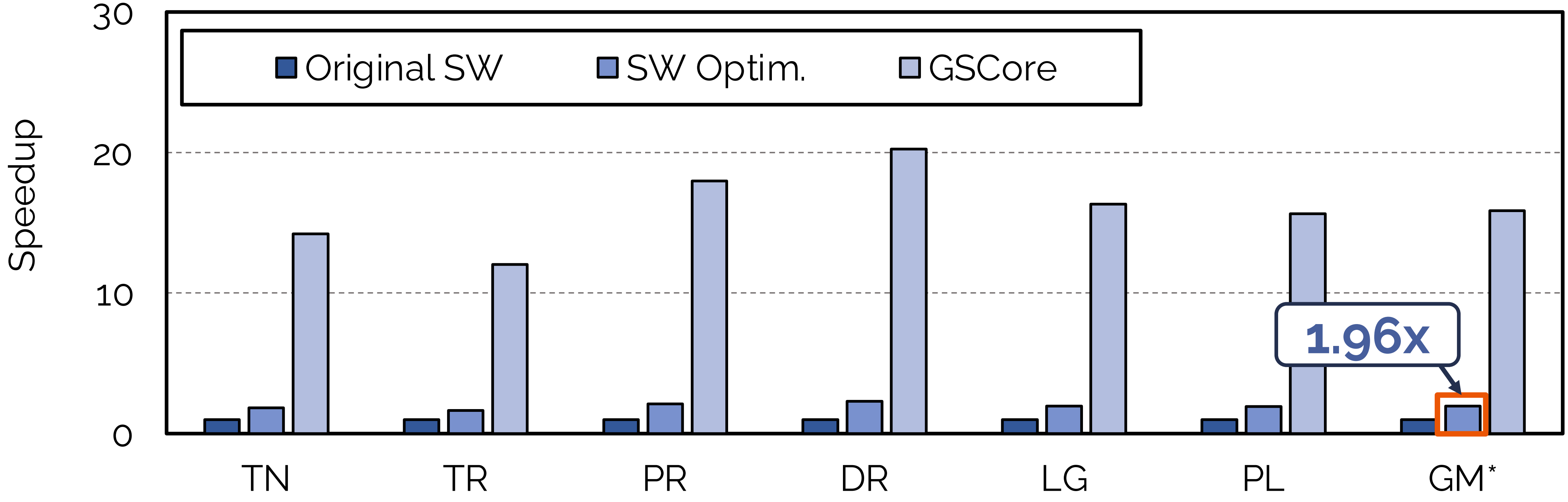
End-to-End Speedup



*GM: Geo Mean

Performance

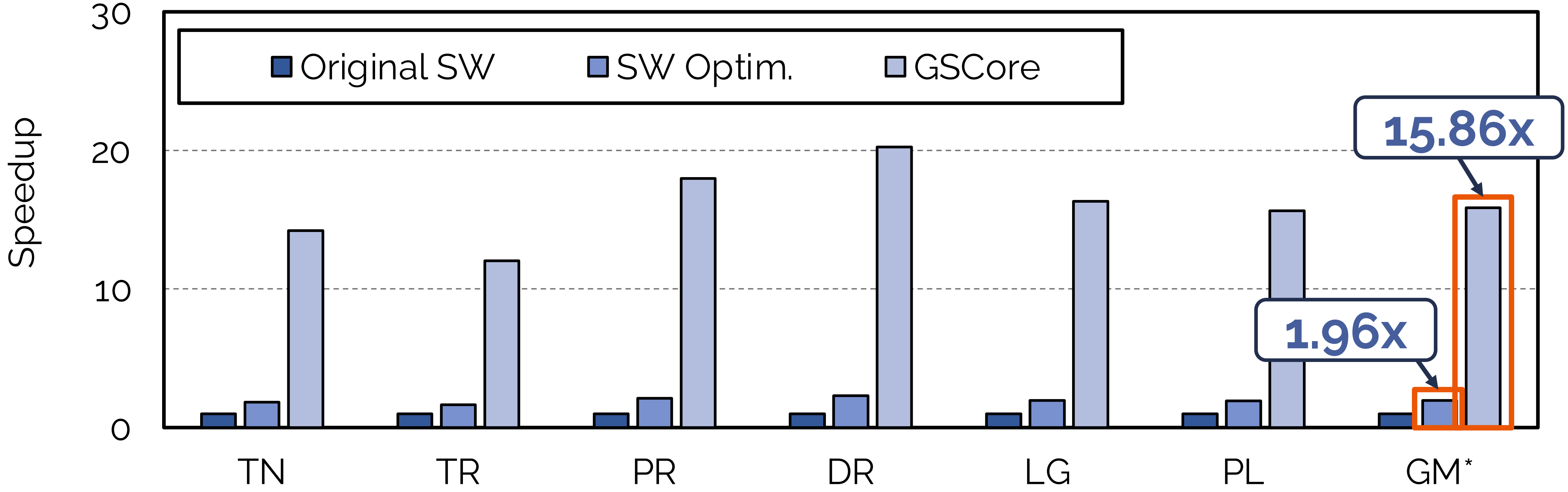
End-to-End Speedup



*GM: Geo Mean

Performance

End-to-End Speedup



*GM: Geo Mean

Performance

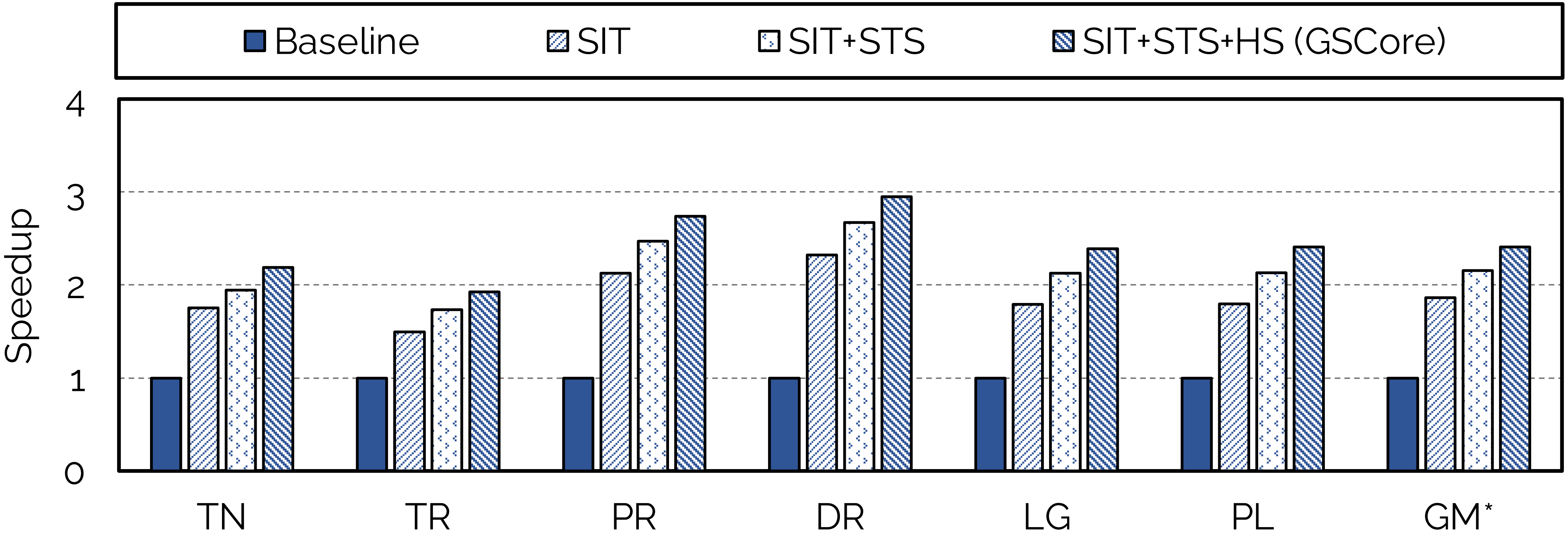
Source of Performance Gain

SIT: **S**hape-Aware **I**ntersection **T**est
STS: **S**ub**T**ile **S**kipping
HS: **H**ierarchical **S**orting

Performance

Source of Performance Gain

SIT: **S**hape-Aware **I**ntersection **T**est
STS: **S**ub**T**ile **S**kipping
HS: **H**ierarchical **S**orting

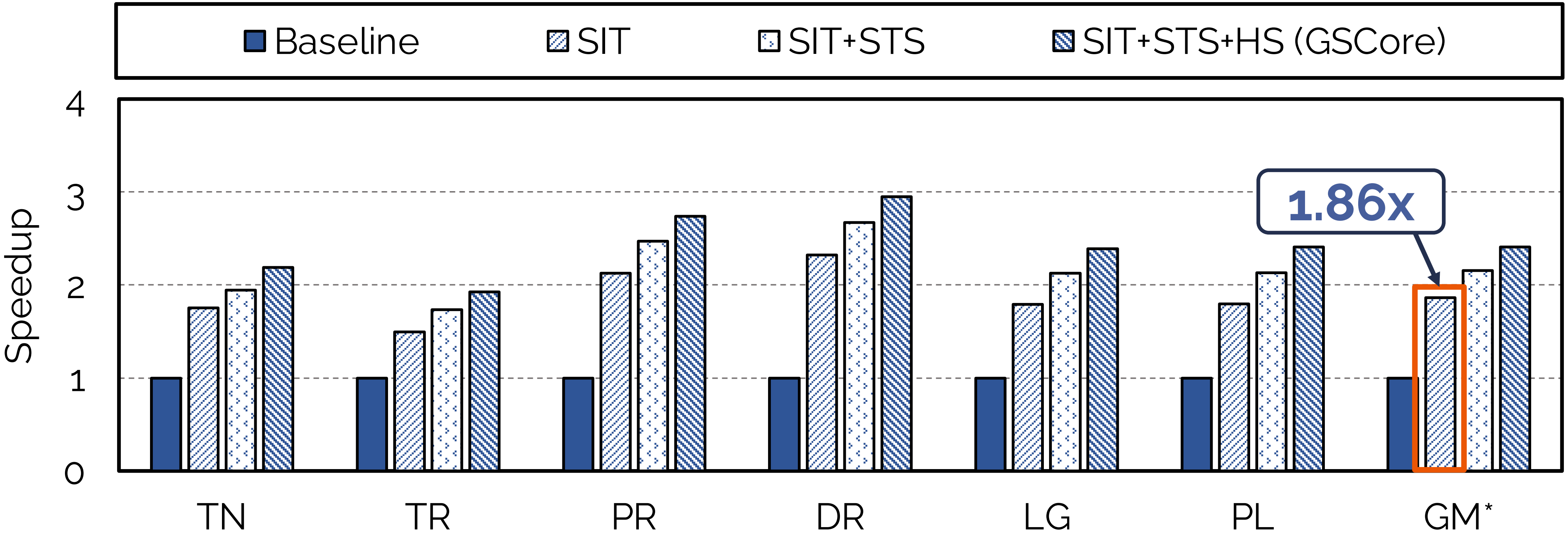


*GM: Geo Mean

Performance

Source of Performance Gain

SIT: **S**hape-Aware **I**ntersection **T**est
STS: **S**ub**T**ile **S**kipping
HS: **H**ierarchical **S**orting

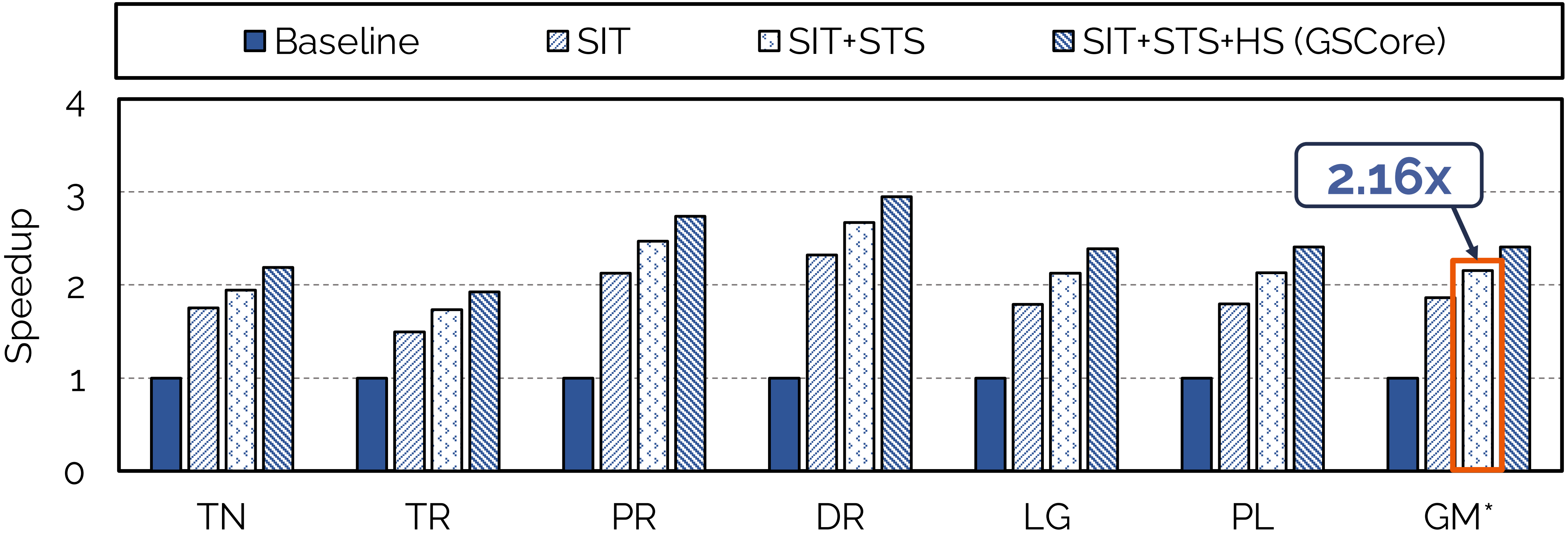


*GM: Geo Mean

Performance

Source of Performance Gain

SIT: **S**hape-Aware **I**ntersection **T**est
STS: **S**ub**T**ile **S**kipping
HS: **H**ierarchical **S**orting

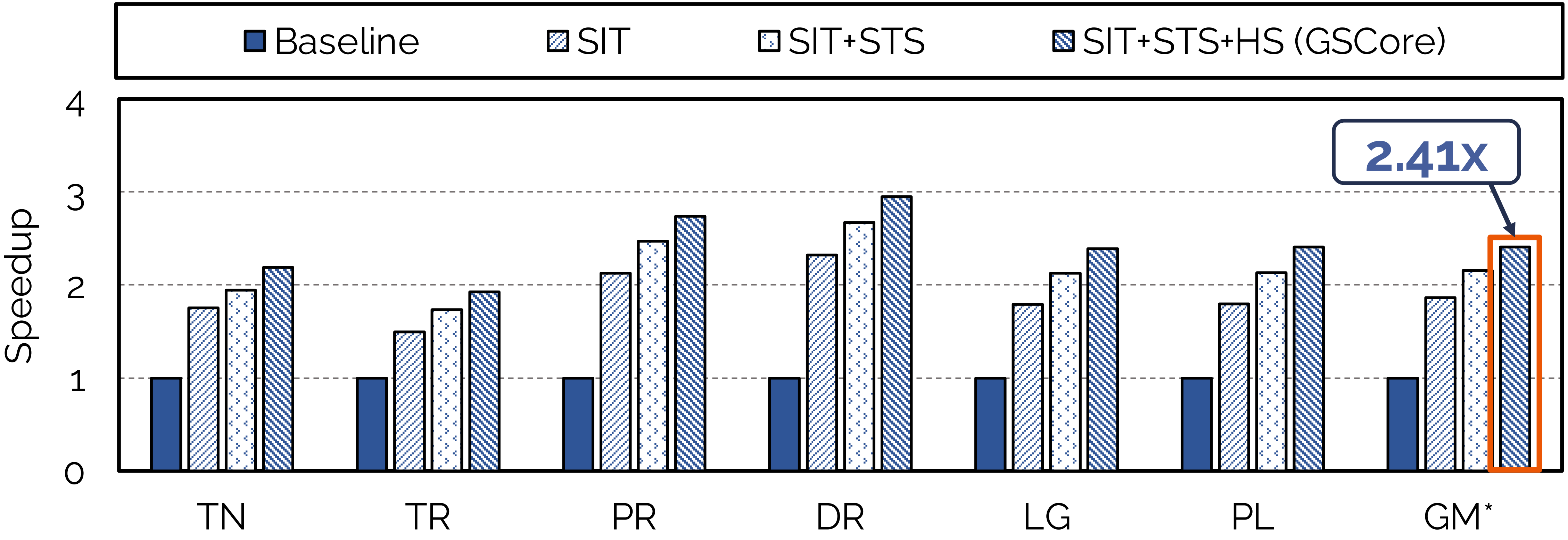


*GM: Geo Mean

Performance

Source of Performance Gain

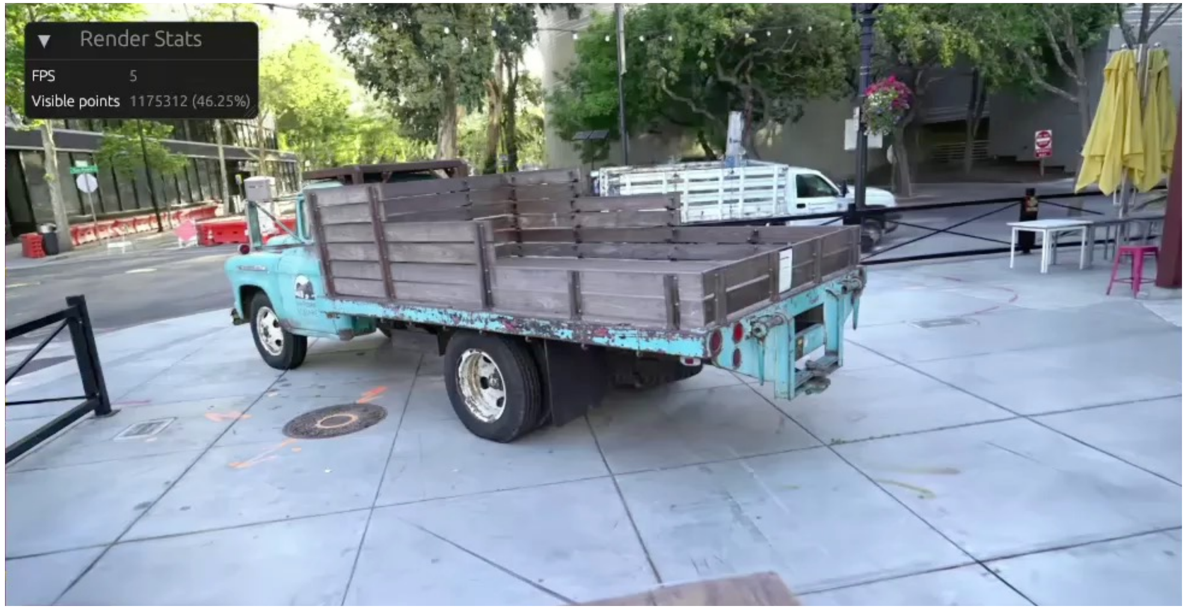
SIT: **S**hape-Aware **I**ntersection **T**est
STS: **S**ub**T**ile **S**kipping
HS: **H**ierarchical **S**orting



*GM: Geo Mean

Performance

Jetson Xavier NX

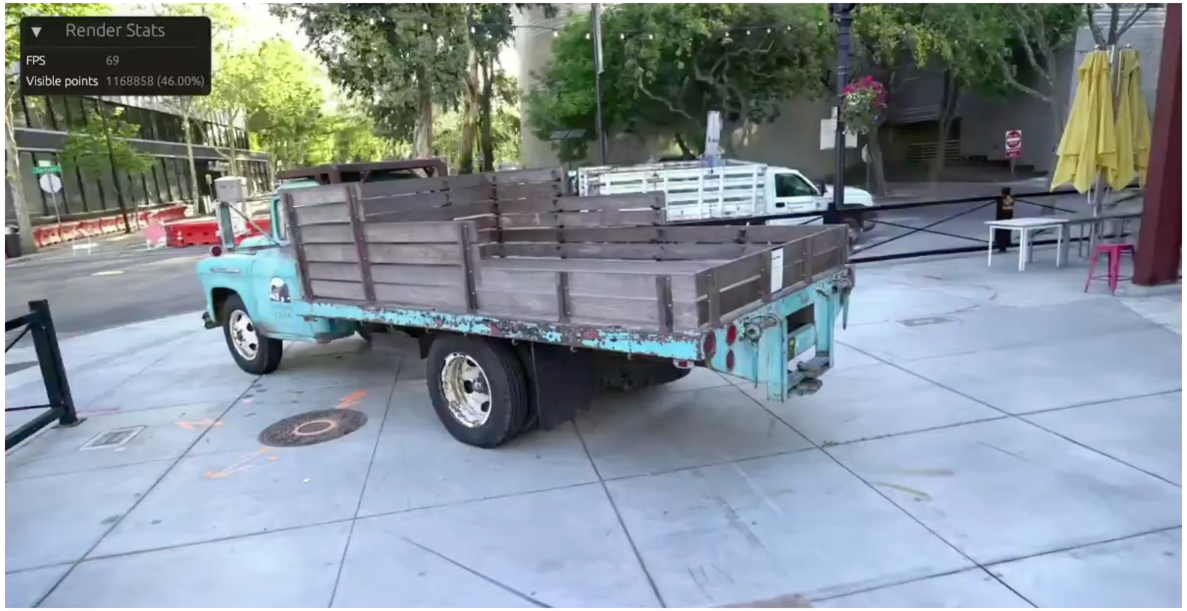


▼ Render Stats

FPS 5

Visible points 1175312 (46.25%)

GSCore



▼ Render Stats

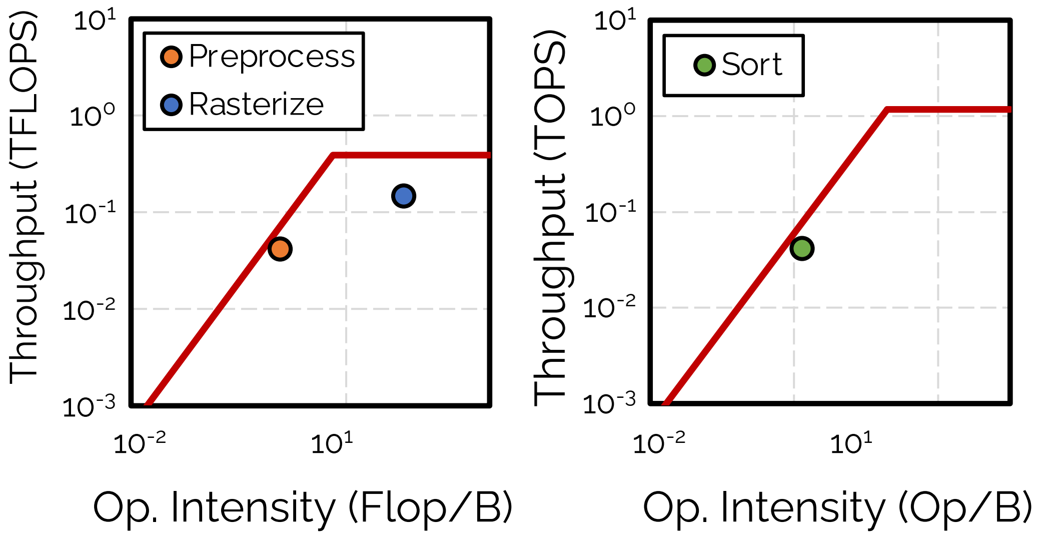
FPS 69

Visible points 1168858 (46.00%)

GSCore enables **real-time rendering**
with a **substantially small area overhead**! ☺

More Details in Our Paper

- Roofline Analysis
- Using RT cores for intersection test



- Sensitivity Study
- Area & Energy Efficiency
- Analysis & Discussion
  - Fixed-Function Rasterizer in GPU
  - Others…

Conclusion

Problem

- Gaussian Sorting & Rasterization are two main bottlenecks of 3DGS
- There are many inefficiencies in both steps

Solution: **GSCore**, an efficient radiance field rendering unit

- Algorithmic optimizations reduce ineffective computations
- Hardware design synergistic with algorithm optimizations

Result

- **GSCore** achieves an average of **15.86x end-to-end speedup** over the GPU with a **substantially small area** overhead! ☺

Thank You!

GSCore

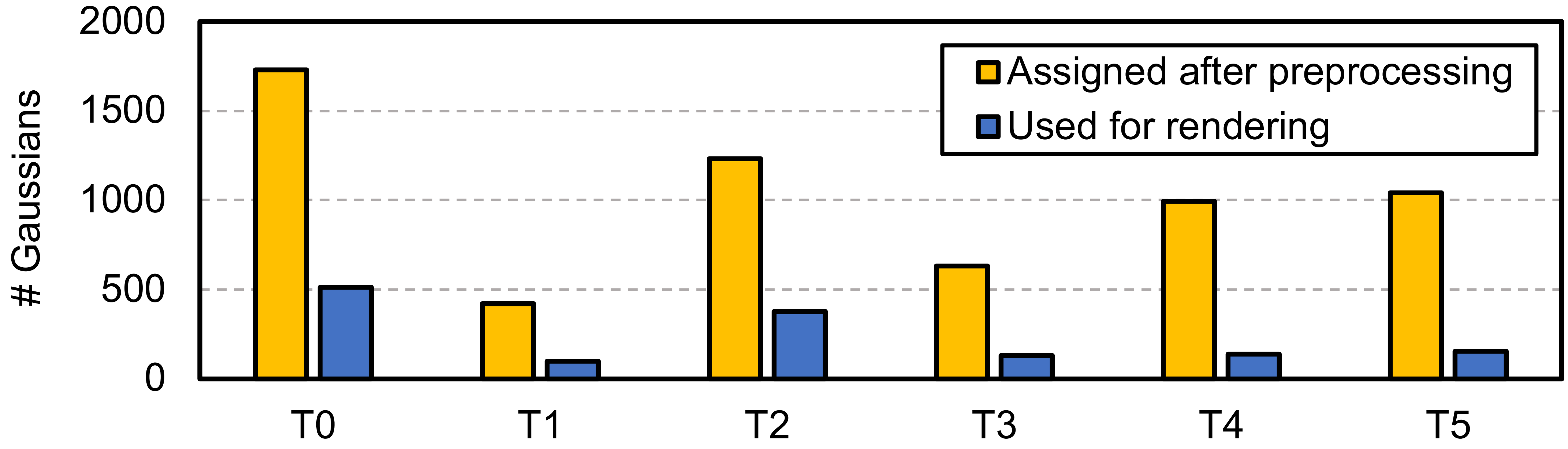
Efficient Radiance Field Rendering
via Architectural Support
for 3D Gaussian Splatting

Junseo Lee (junseo.lee@snu.ac.kr)

ASPLOS'24 | April 2024

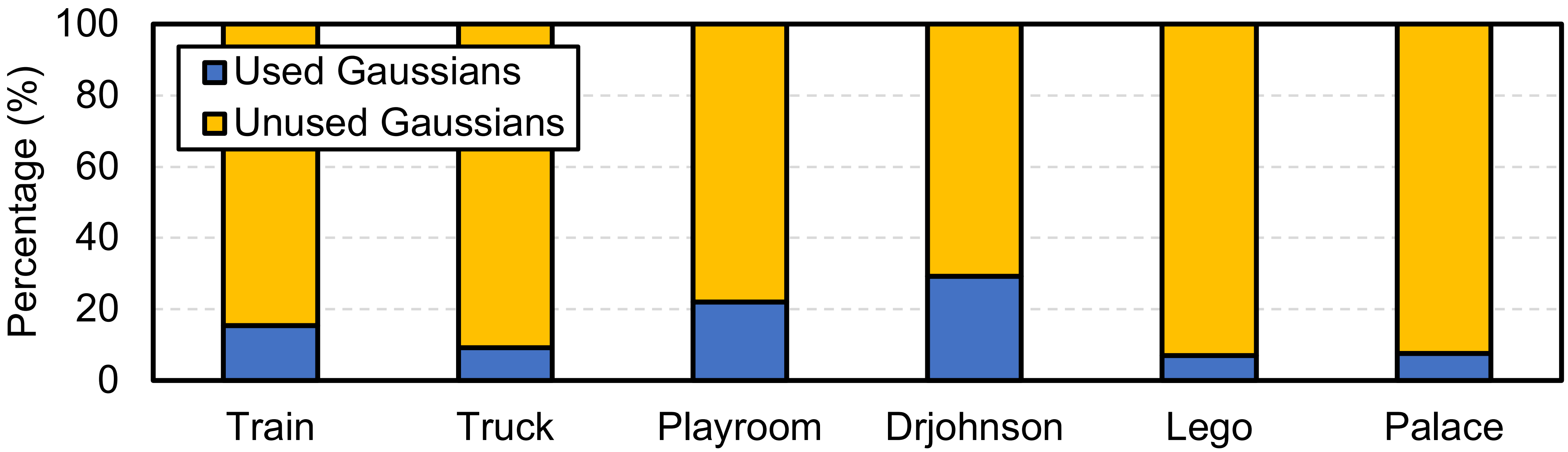
Appendix – Falsely Assigned Gaussians

AABB of the original algorithm leads to many tiles with falsely assigned gaussians!



Appendix – Ineffective Alpha Computation

Tile-based execution of the original algorithm leads to ineffective alpha computation in majority of threads!



Appendix – Equations in Rasterization Stage

Rasterization = ***α*-blending** of Gaussians

***α*-computation**

$$\alpha_{Gaus} = G(pixel)$$

$$o * e^{-\frac{1}{2}(p-\mu)^T {\Sigma'}^{-1}(p-\mu)}$$

***α*-blending**

Transmittance T

$$\boldsymbol{C}_{\boldsymbol{pixel}} += \boxed{\left(1-\alpha_{pixel}\right)} * \alpha_{Gaus} * \boldsymbol{C}_{\boldsymbol{Gaus}}$$

$$\alpha_{pixel} += \left(1-\alpha_{pixel}\right) * \alpha_{Gaus}$$

Appendix – Rendering Quality

* Higher is better

Playroom

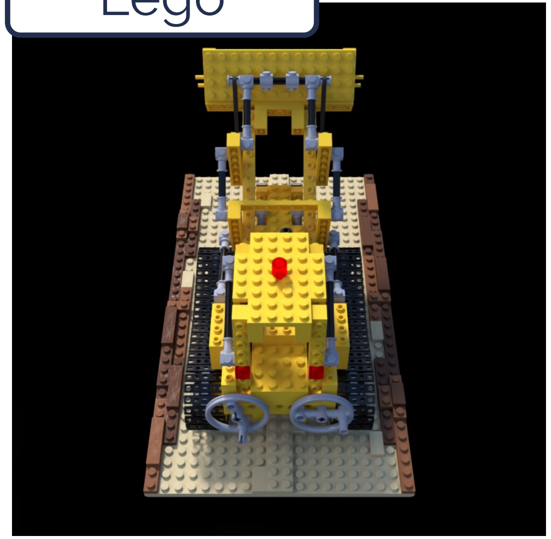
FP32

Org: 29.89 dB

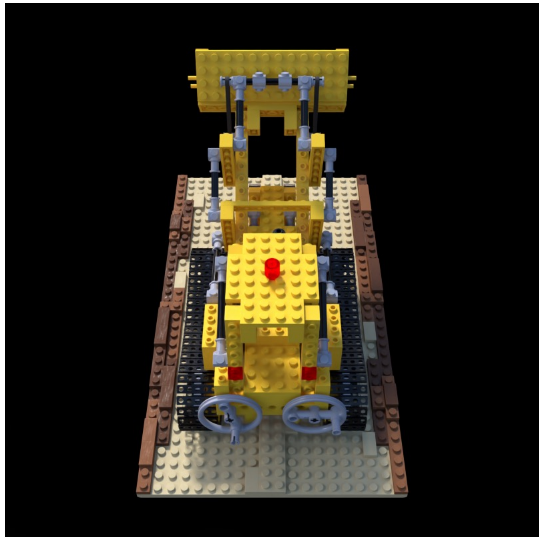
FP16

Ours: 29.83 dB

Lego



Org: 34.47 dB



Ours: 34.40 dB

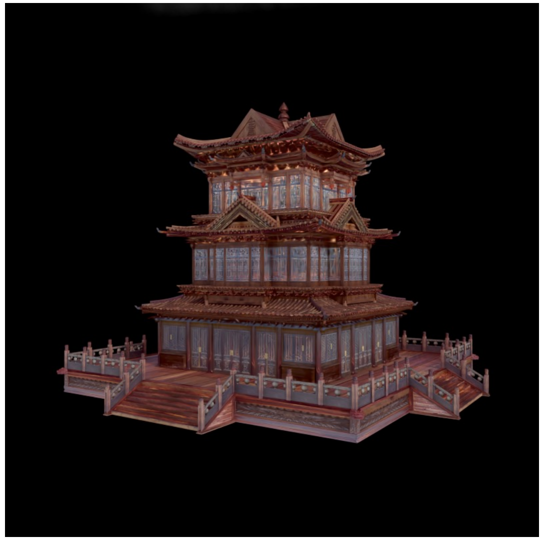
Drjohnson

Org: 35.19 dB

Ours: 35.00 dB

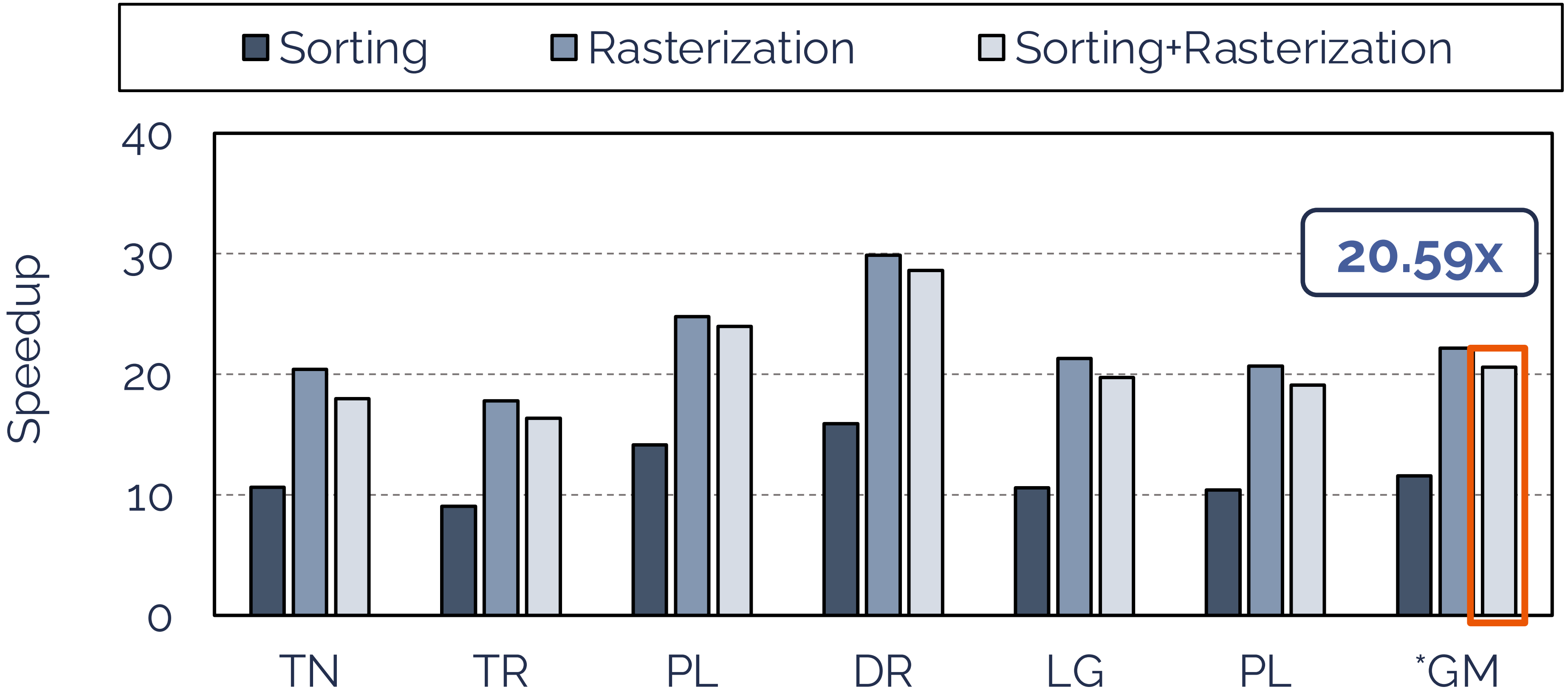
Palace

Org: 33.75 dB



Ours: 33.76 dB

Appendix - Sorting & Rasterization Speedup



*GM: GeoMean