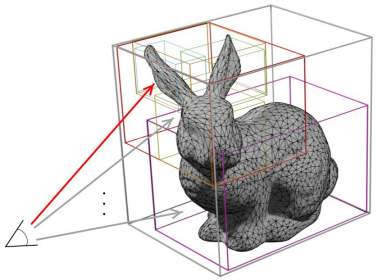
# Raster is Faster:
## Rethinking Ray Tracing in Database Indexing

Harish Doraiswamy
Microsoft Research India

Jayant R. Haritsa
Indian Institute of Science

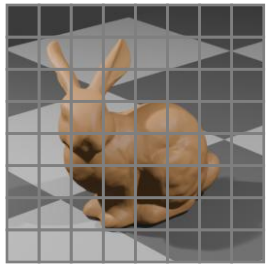
# What this paper is about



- Modern GPUs have native Ray Tracing (RT) support
- There is effort to utilize this for database operations
  - Indexing *[Henneberg and Schuhknecht, PVLDB 2023] [Lv et al., PVLDB 2024]*

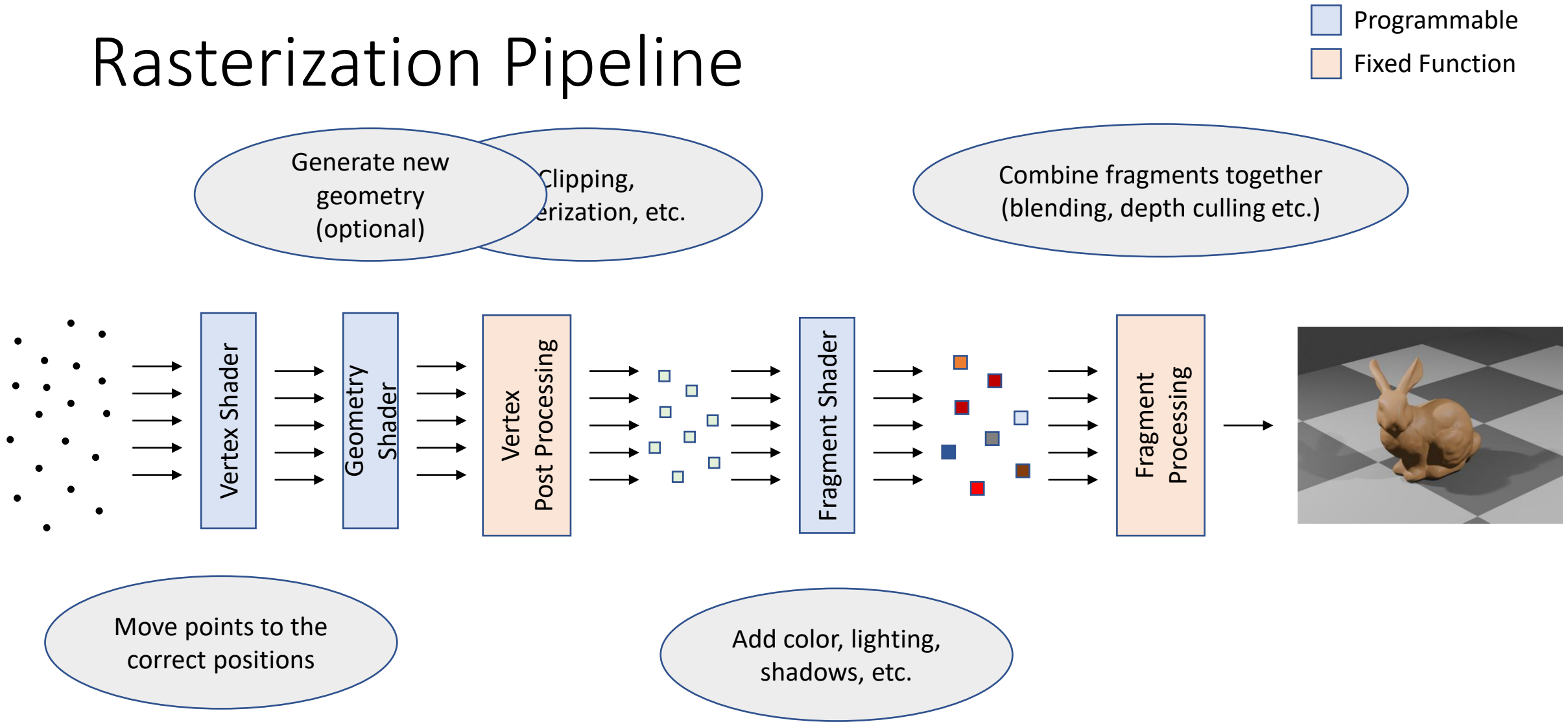
The way the problem is modelled to apply RT is inefficient

*Rasterization* is better suited for this purpose



Not about introducing a new (comprehensive) index

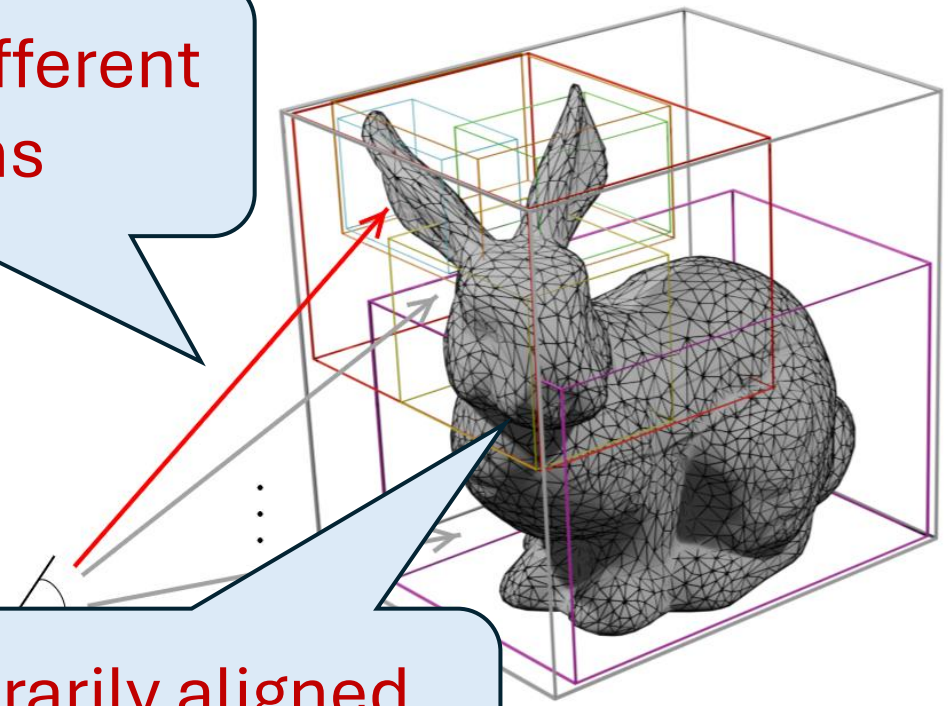
# Rasterization Pipeline



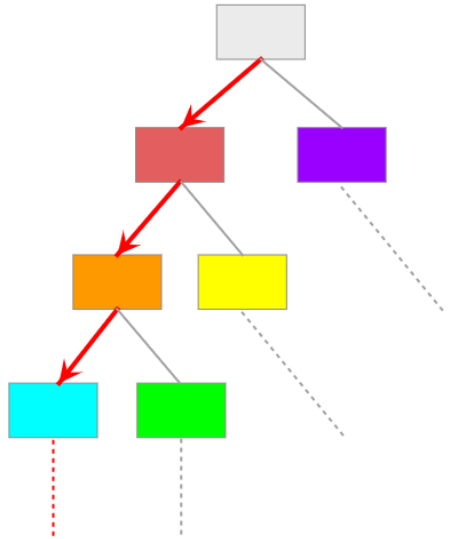
# Ray Tracing

- More accurate lighting and shading

Rays along different directions



Arbitrarily aligned triangles

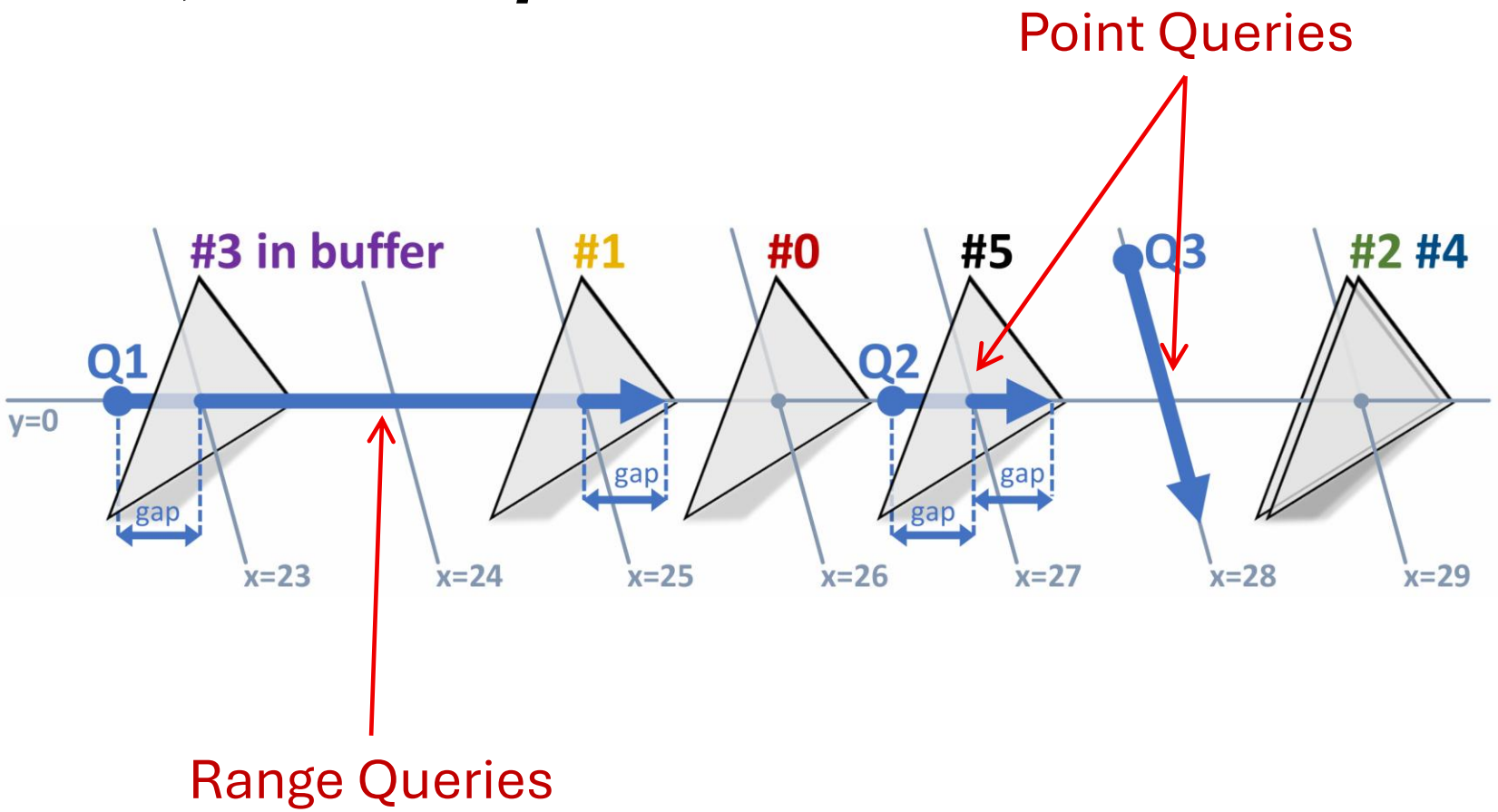



Native GPU support

# Database Indexing with RT: RTIndex

*[Henneberg and Schuhknecht, PVLDB 2023]*

| rowID | Article | Category |
|---|---|---|
| 0 | Juice | 26 ($k_0$) |
| 1 | Bread | 25 ($k_1$) |
| 2 | Cookies | 29 ($k_2$) |
| 3 | Coffee | 23 ($k_3$) |
| 4 | Donuts | 29 ($k_4$) |
| 5 | Wine | 27 ($k_5$) |

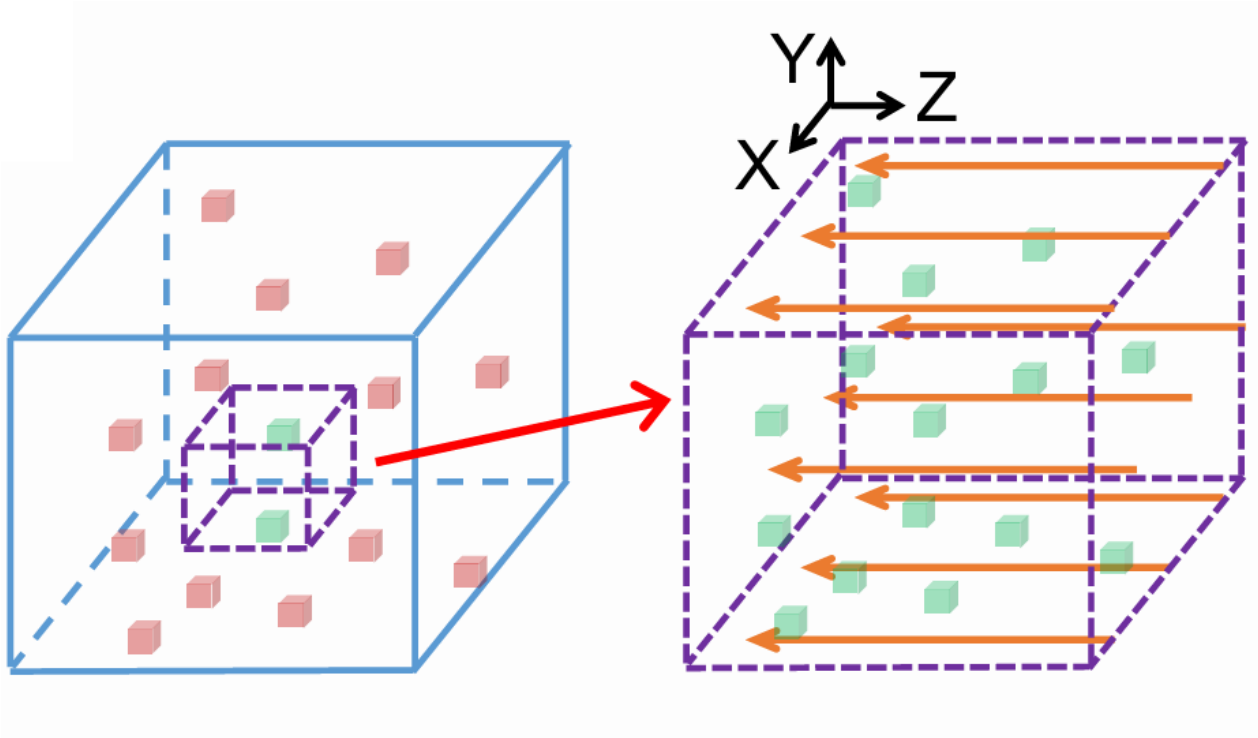


# Database Indexing with RT: RTScan

*[Lv et al., PVLDB 2024]*

Triangles/Cubes are places along grid points

Rays are axis aligned



# Database Indexing with RT: Overkill?

Triangles/Cubes are places along grid points

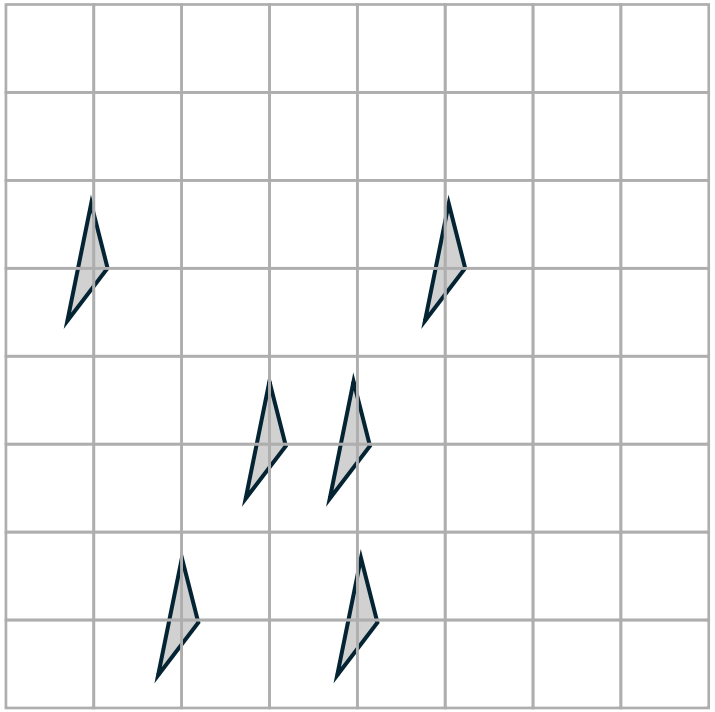
Rays are axis aligned

Still forced to do

BVH Traversal

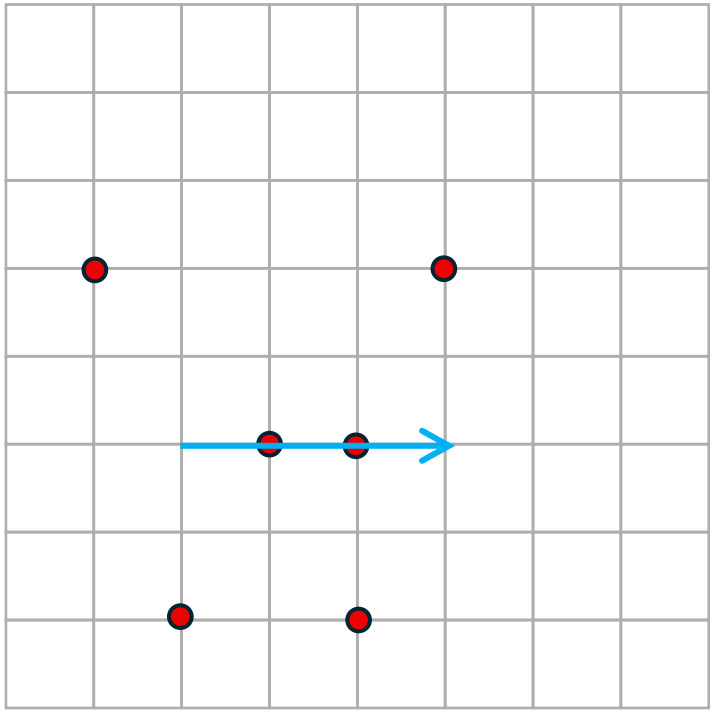
Ray-Triangle Intersections

# What about Rasterization?



# What about Rasterization?

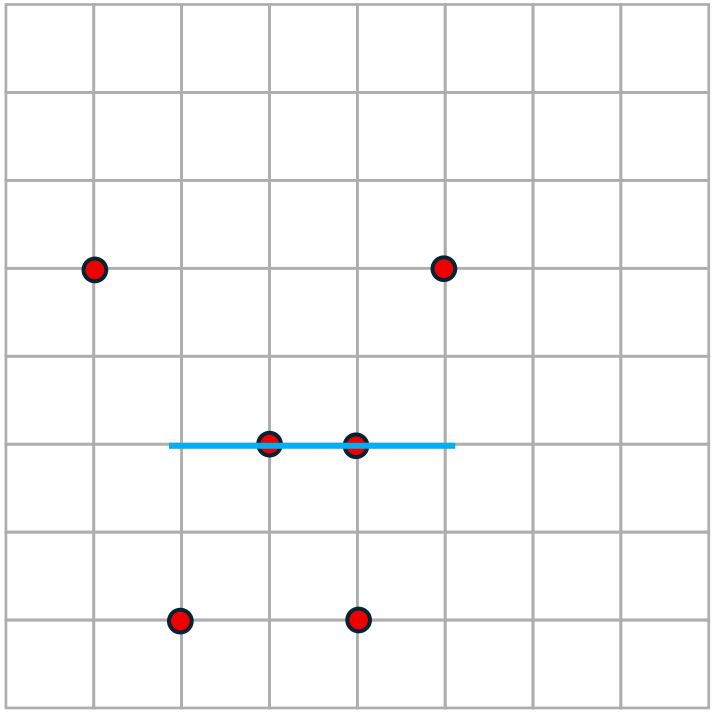
Triangles ⟶ Points



# What about Rasterization?

Triangles ⟶ Points

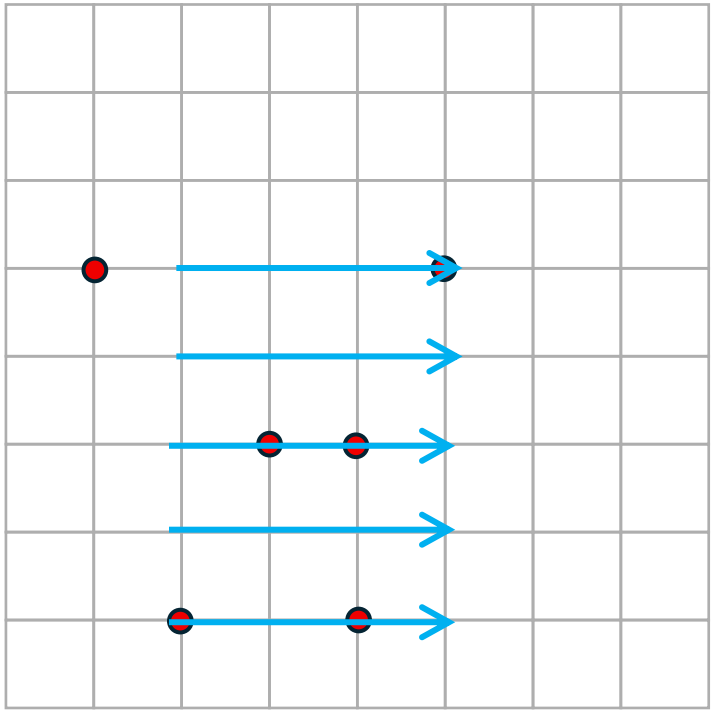
Rays ⟶ Lines



# What about Rasterization?

Triangles ⟶ Points

Rays ⟶ Lines

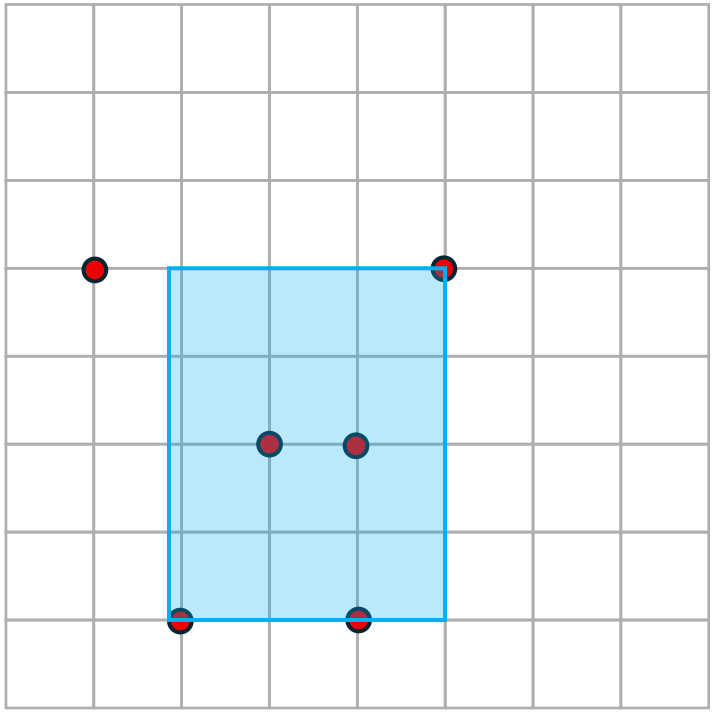


# What about Rasterization?

Triangles ⟶ Points

Rays ⟶ Lines

Multiple Rays ⟶ Rectangle

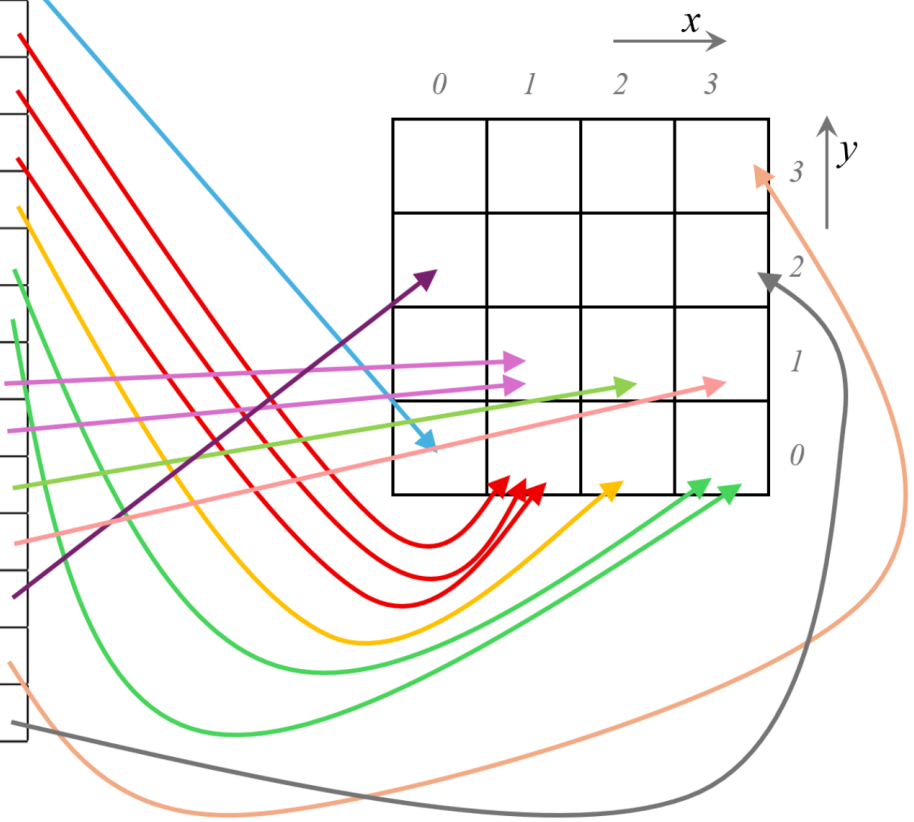


# Our Proposal: RasterScan Index

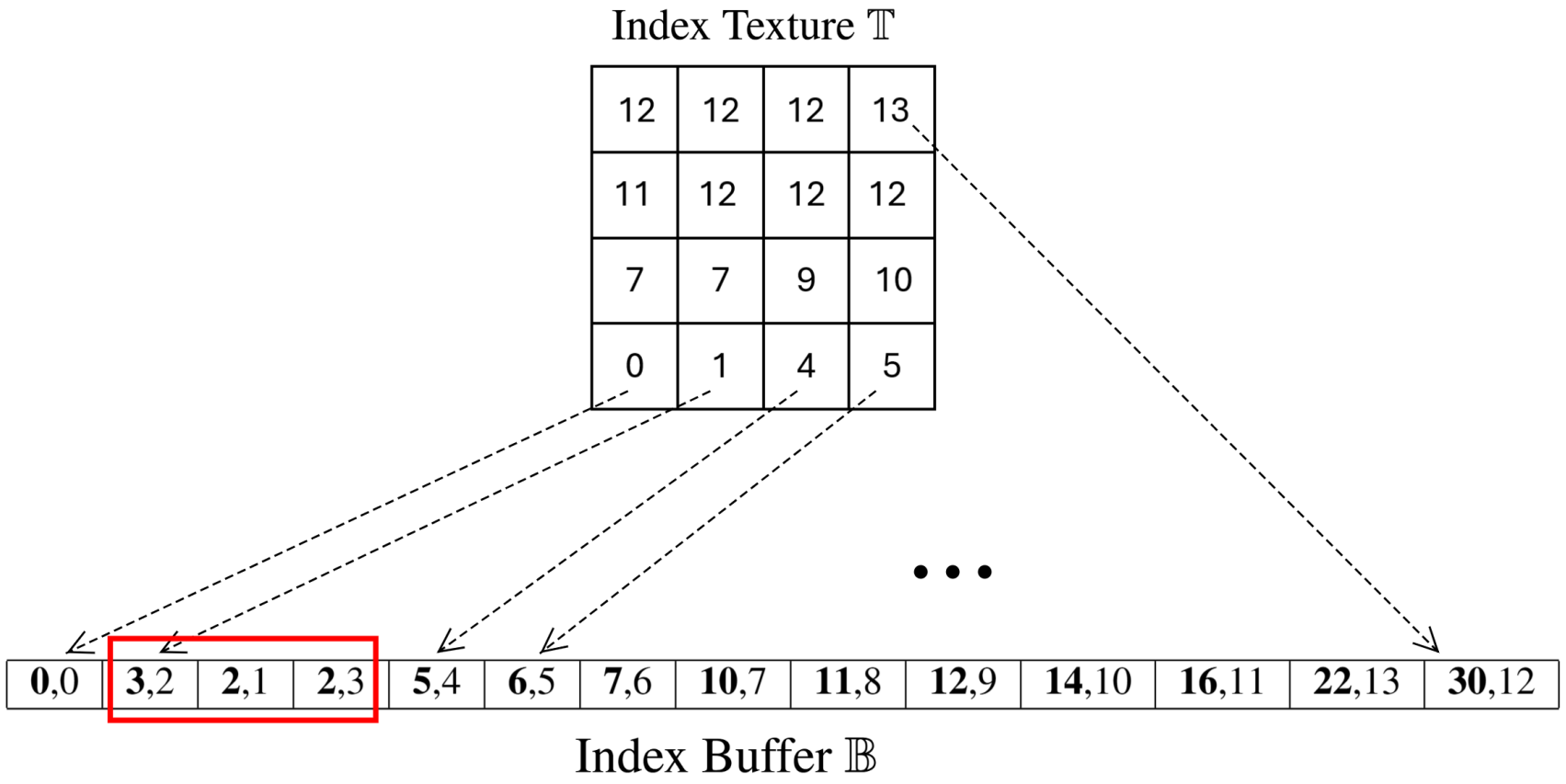
| RID | $\mathbb{C}$ |
|---|---|
| 0 | 0 |
| 1 | 2 |
| 2 | 3 |
| 3 | 2 |
| 4 | 5 |
| 5 | 6 |
| 6 | 7 |
| 7 | 10 |
| 8 | 11 |
| 9 | 12 |
| 10 | 14 |
| 11 | 16 |
| 12 | 30 |
| 13 | 22 |

# Our Proposal: RasterScan Index

| RID | ℂ |
|---|---|
| 0 | 0 |
| 1 | 2 |
| 2 | 3 |
| 3 | 2 |
| 4 | 5 |
| 5 | 6 |
| 6 | 7 |
| 7 | 10 |
| 8 | 11 |
| 9 | 12 |
| 10 | 14 |
| 11 | 16 |
| 12 | 30 |
| 13 | 22 |



# Our Proposal: RasterScan Index



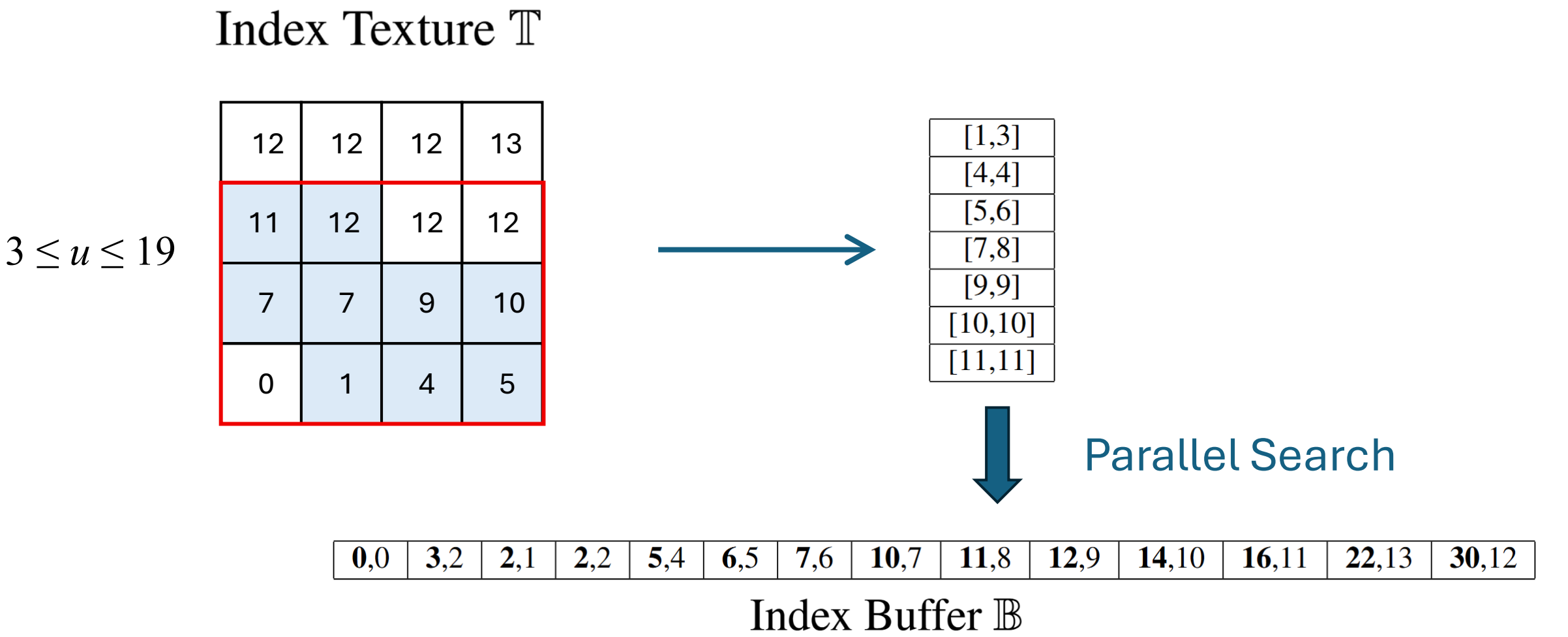
# RasterScan: Search

$3 \leq u \leq 19$

Index Texture $\mathbb{T}$

| | | | |
|---|---|---|---|
| 12 | 12 | 12 | 13 |
| 11 | 12 | 12 | 12 |
| 7 | 7 | 9 | 10 |
| 0 | 1 | 4 | 5 |

# RasterScan: Search

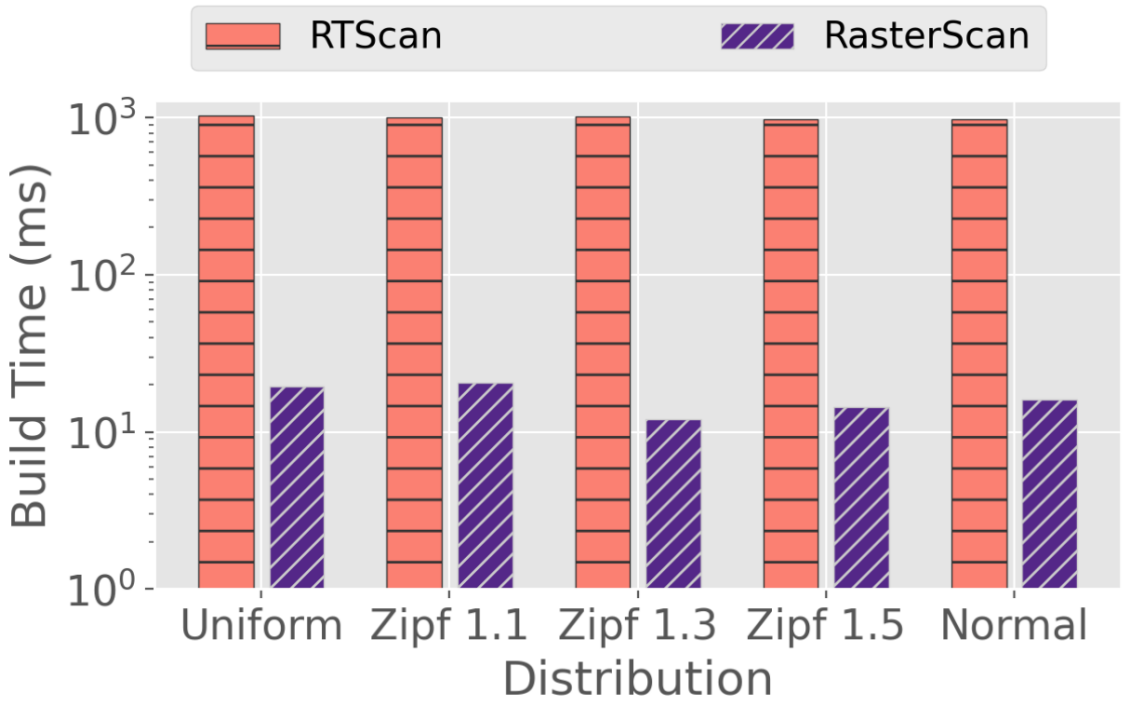


# Evaluation

- Implementation
  - C++, Vulkan (Graphics API)
- Baseline
  - RTScan *[Lv et al., PVLDB 2024]*
- GPU
  - RTX 4090
- Datasets
  - Synthetic with Uniform and Skewed distributions
  - 100M rows
  - 3 Columns, Integers

Input and output format match

# Index Build Performance




~50X

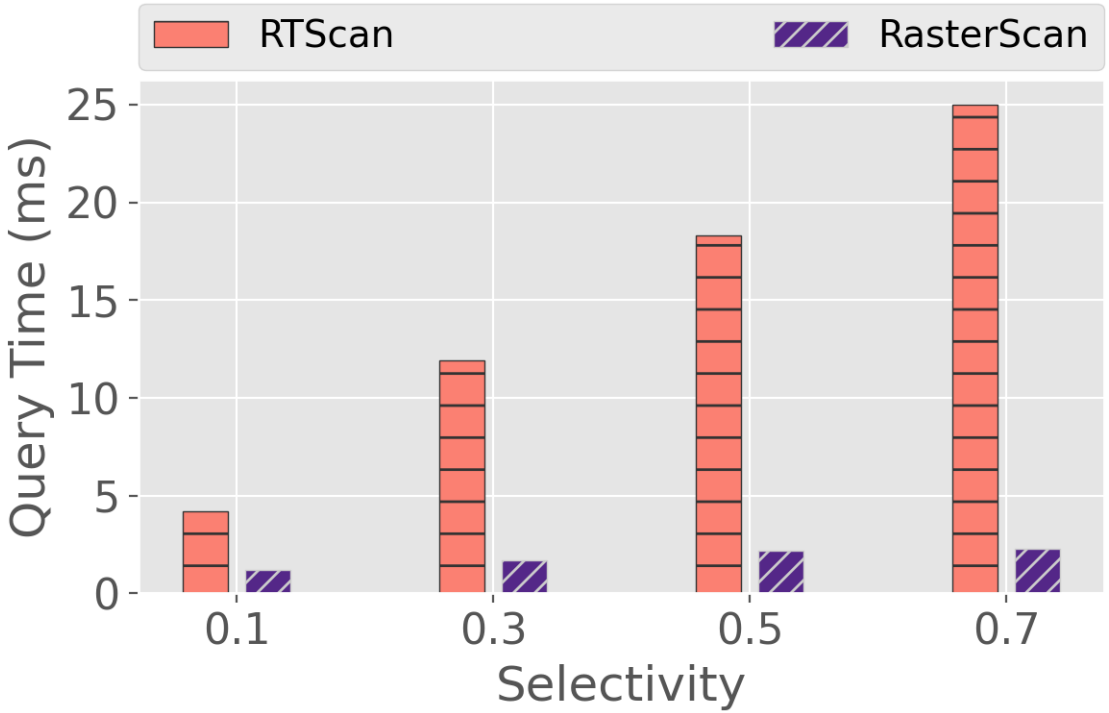
RTScan Index build (BVH tree computation) is handled by the Nvidia API

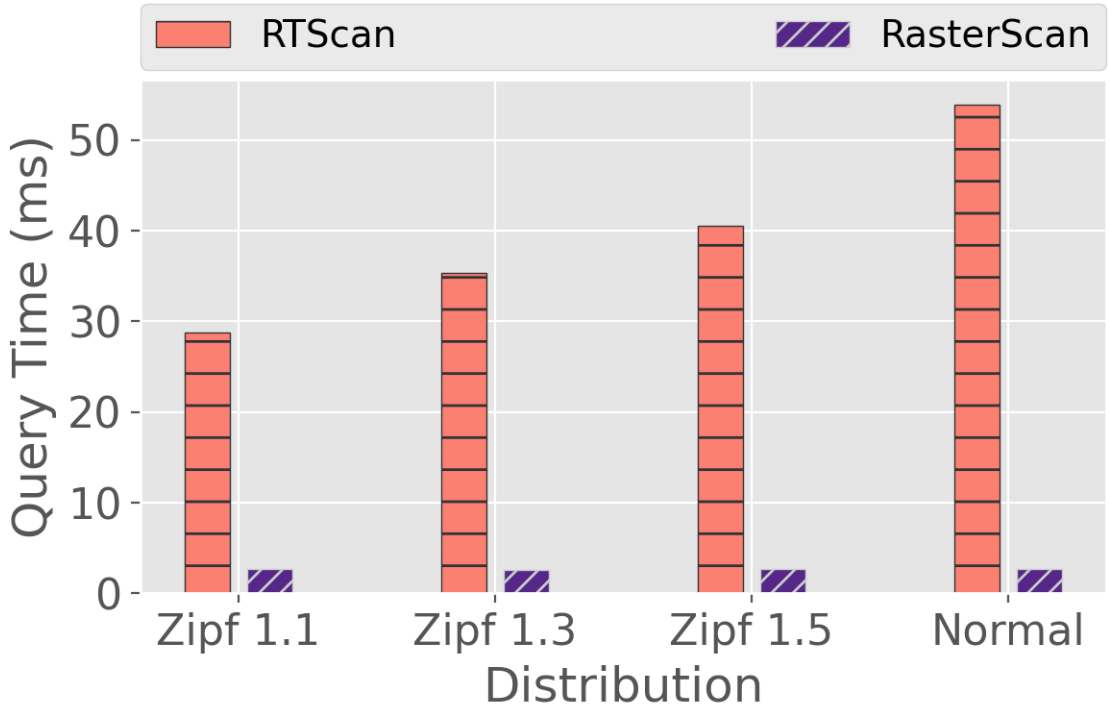
- Irregular memory accesses
- Synchronization

RasterScan Build

- Simple arithmetic
- Efficient atomic increments

# Search Performance: Range Queries




- At least 2X and >10X for high selectivity

Costly Ray Intersections vs Simple Arithmetic + Comparisons

# Take Away

- If data model is well behaved
  - Points along grid
  - Operations are axis aligned
- You want to use graphics primitives

Use Rasterization instead of Ray Tracing

Where should Ray Tracing be used?

Better left for graphics rendering!

https://github.com/Microsoft/raster-scan