Spatial Independent Range Sampling

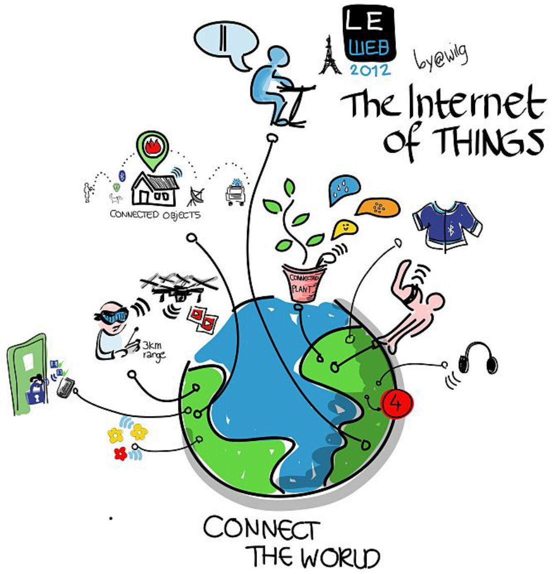
Dong Xie[2], Jeff M. Phillips[1], Michael Matheny[3], Feifei Li[4]

University of Utah[1], Penn State University[2], Amazon[3], Alibaba Inc[4]

Big Spatial Data

Location-based Services

IoT Projects & Sensor Networks



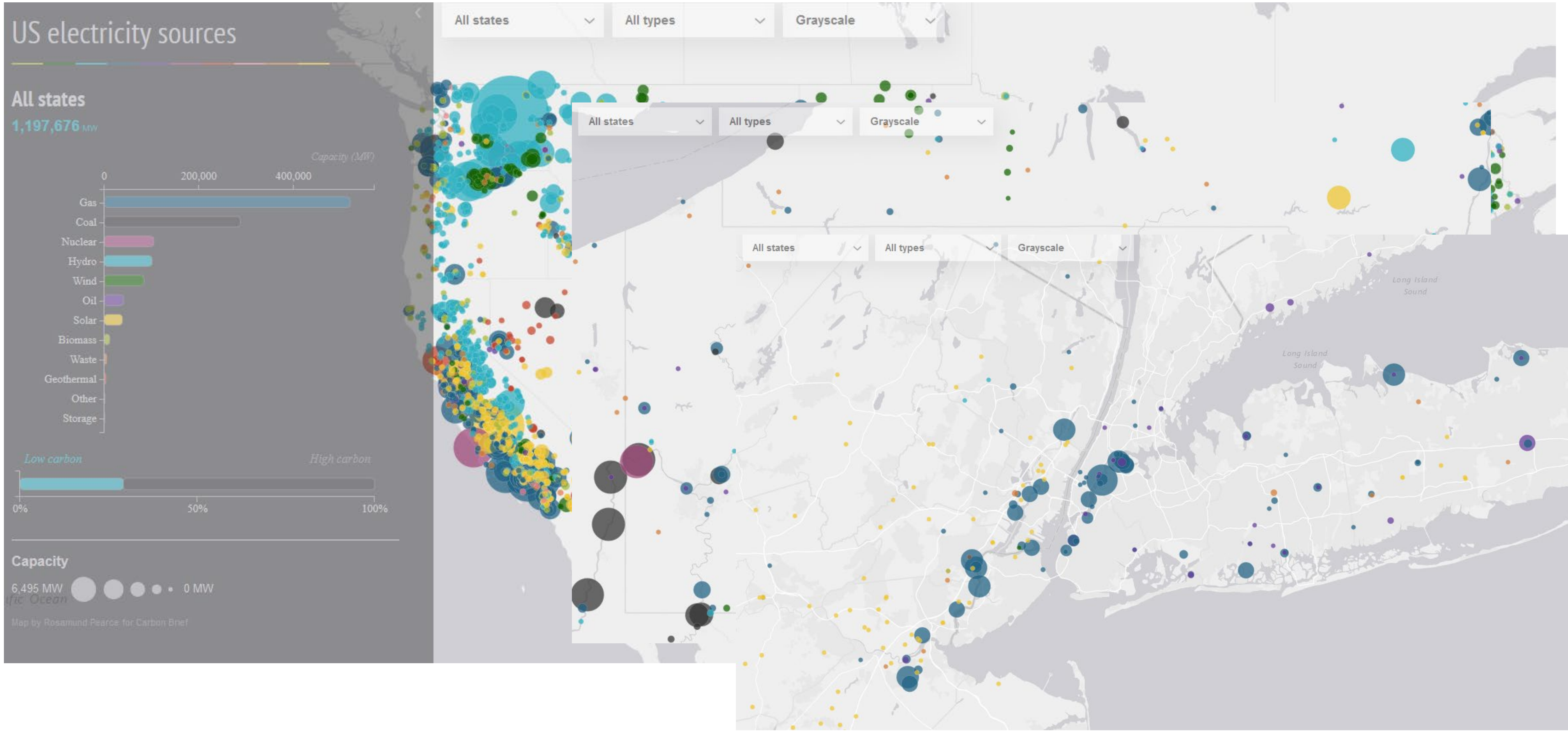
Social Media

Site Recommendations

Traffic Analysis

Transportation Optimization

Interactive Spatial Data Analysis



Source: https://www.carbonbrief.org/mapped-how-the-us-generates-electricity

How to Achieve “Interactive”?

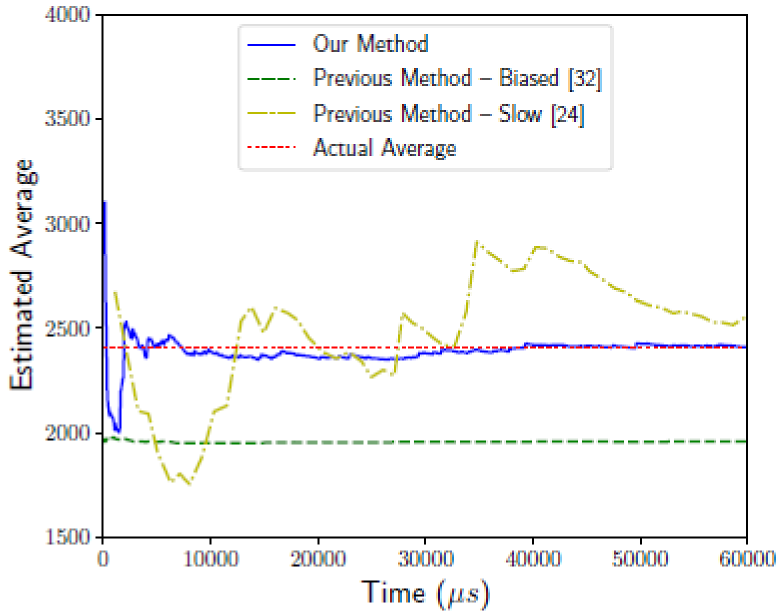
- What needs to be done?
  - Interactive exploration/analysis on a map app.
  - Large scale data visualization.
  - Randomized site recommendation.

- Low latency analysis w/ exact results → Slow/Resource intensive.

- Another Approach?
- Don’t need exact results -> approximation with guarantees
- Trade-off between accuracy and performance.
- **Approximate Query Processing**
- Need to **sampling on the fly**.

Spatial Independent Range Sampling (SIRS)

- Sample **Independence** is important!
  - Convenience for analysis.
  - Easy continuation.
- Numerous statistics tools requires sample independence.
- Other requirements:
  - Arbitrary range (MBR) to explore.
  - Fast sample retrieval for each query.
  - Low cost on preprocessing and storage.
- **Spatial Independent Range Sampling (SIRS).**



SIRS Problem Formalized

Uniform SIRS

Given a spatial data set $P \subset \mathbb{R}^d$, an MBR R, and an integer k,

a uniform SIRS query will return k independent random samples

from $R \cap P$ with each data point $p \in R \cap P$

having a probability of $\frac{1}{|R \cap P|}$ to be sampled.

Weighted SIRS

Given a spatial data set $P \subset \mathbb{R}^d$, weight function $w: P \rightarrow \mathbb{R}^+$, an MBR R, and an integer k,

a weighted SIRS query will return k independent random samples

from $R \cap P$ with each data point $p \in R \cap P$

having a probability of $\frac{w(p)}{\sum_{q \in R \cap P} w(q)}$ to be sampled.

Baseline Solutions

- [VLDB'89] Olken's Method
  - Key idea: **traverse tree randomly** with **rejection**.
  - Pros: straightforward, very easy to implement and generalized.
  - Requires a lot of RNG, cause a lot of rejections -> slow.

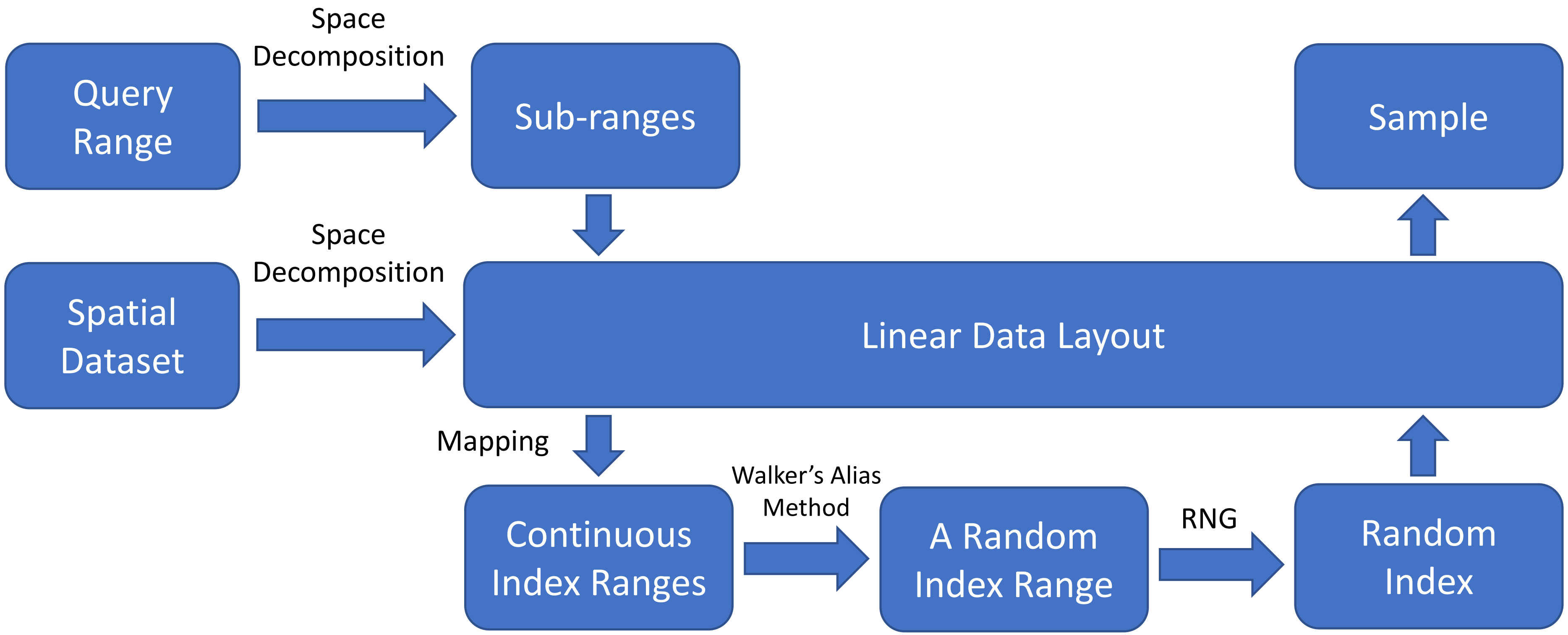
- [VLDB'15] Spatial Online Sampling.
  - Key idea: **sampling buffer** on each tree node to accelerate Olken's Method.
  - Pros: fast for low sample numbers.
  - Cons: **NO inter-query independence!!**

- Query then sample:
  - Get the full result and retrieve samples directly.
  - Need to issue a exact range query -> slow.

Sampling Framework

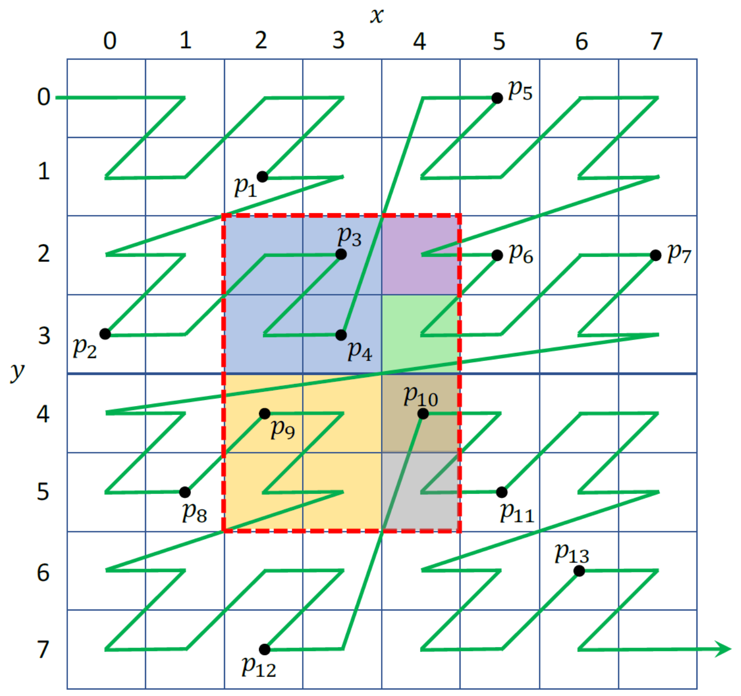
- Observation: *uniform IRS on 1D sequence over index range [s, t] is trivial*
  - Generate random numbers in [s, t] then report correspondent data.

- Reduction from SIRS to 1D sampling



Z-Value Sampling Method

- Natural data layout based on space-filling curves.
- Z-value decomposition -> linear quad tree
- Space Cost: $O(n)$; Query Cost: $O(c(R) + k)$;

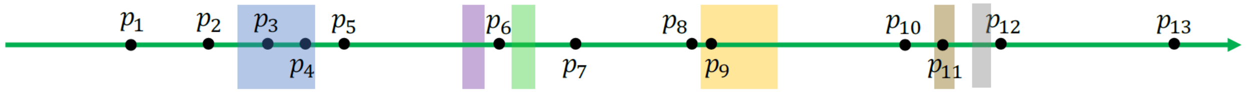


Query Region:

$(2,2) \rightarrow (4,5)$

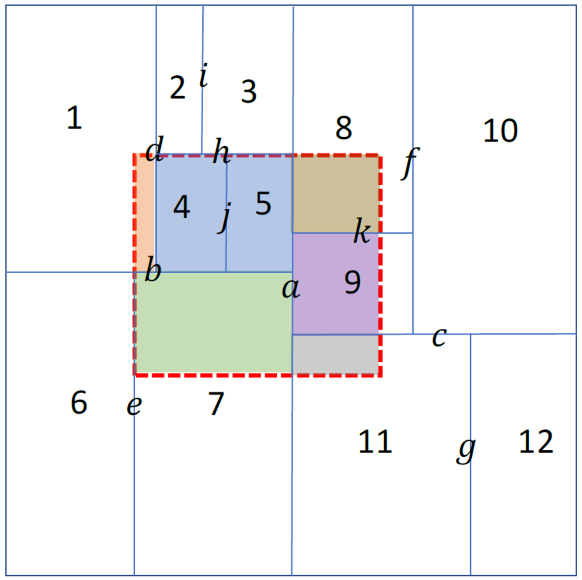
Decompose into Z-value ranges:

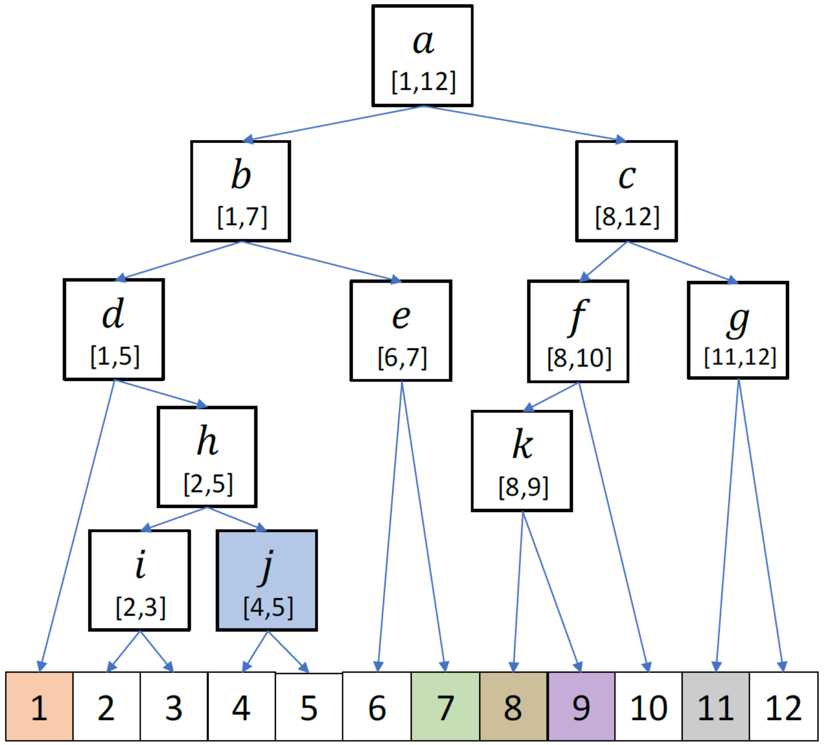
$(2,2) \rightarrow (3,3)$	$\Rightarrow$	[13,16]
$(4,2) \rightarrow (4,2)$	$\Rightarrow$	[25,25]
$(4,3) \rightarrow (4,3)$	$\Rightarrow$	[27,27]
$(2,4) \rightarrow (3,5)$	$\Rightarrow$	[37,40]
$(4,4) \rightarrow (4,4)$	$\Rightarrow$	[49,49]
$(4,5) \rightarrow (4,5)$	$\Rightarrow$	[51,51]



KD-Tree Sampling Method

- Another way decomposing the space with more precision and guarantees.
- Space Cost: $O(n)$; Query Cost: $O(\sqrt{n}+k)$, for higher dimension: $\mathrm{O}(n^{1-1/d}+k)$





Generalized for Other Spatial Indexes

- Accommodate data layout with spatial indexes.

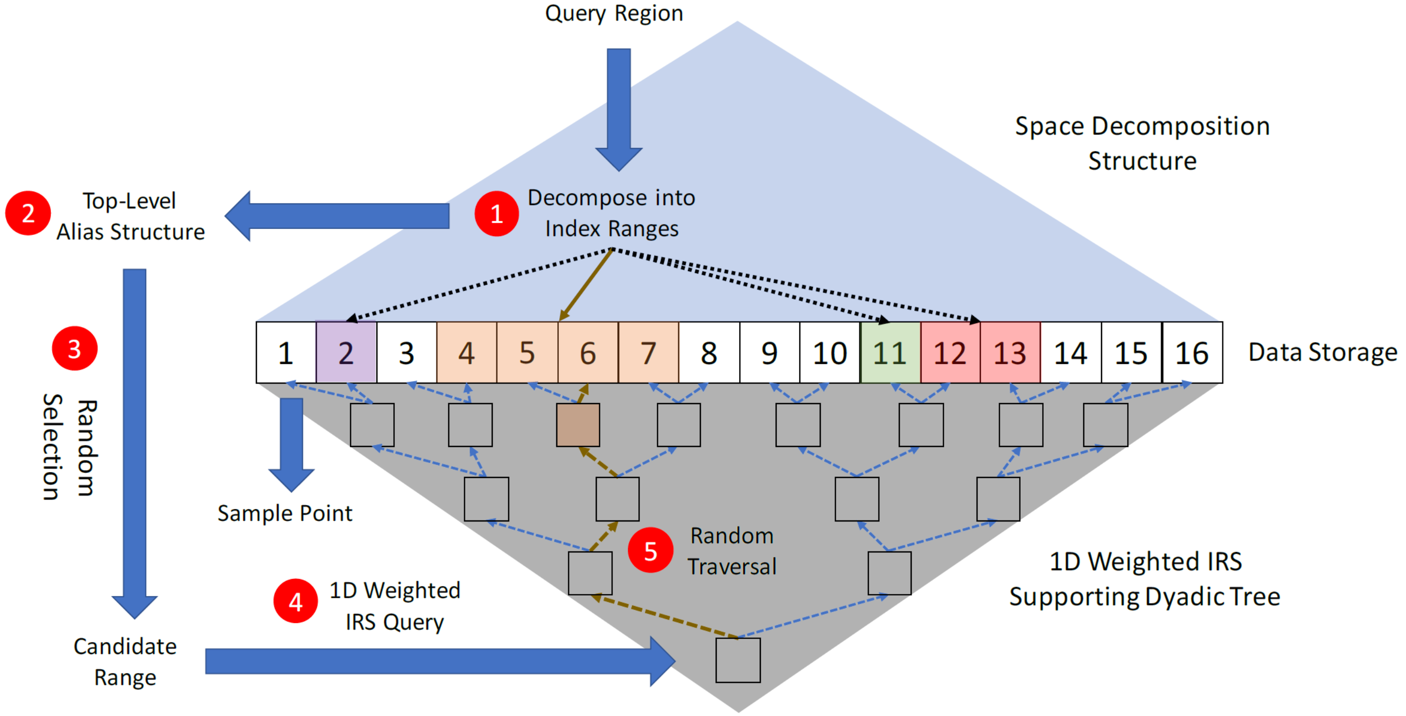
- Principles for the reduction:
  - Each tree node u is corresponded to a continuous interval $[s_u, t_u]$ on data storage.
  - If node u is descendant of node v, the interval of node u is covered by that of node v.

- DFS on the tree
- Concatenate leaf node data to the layout once it is reached.

- Generalized into R-Trees, Dyadic Trees, etc.
- KD-Tree has the best bounds for MBRs.

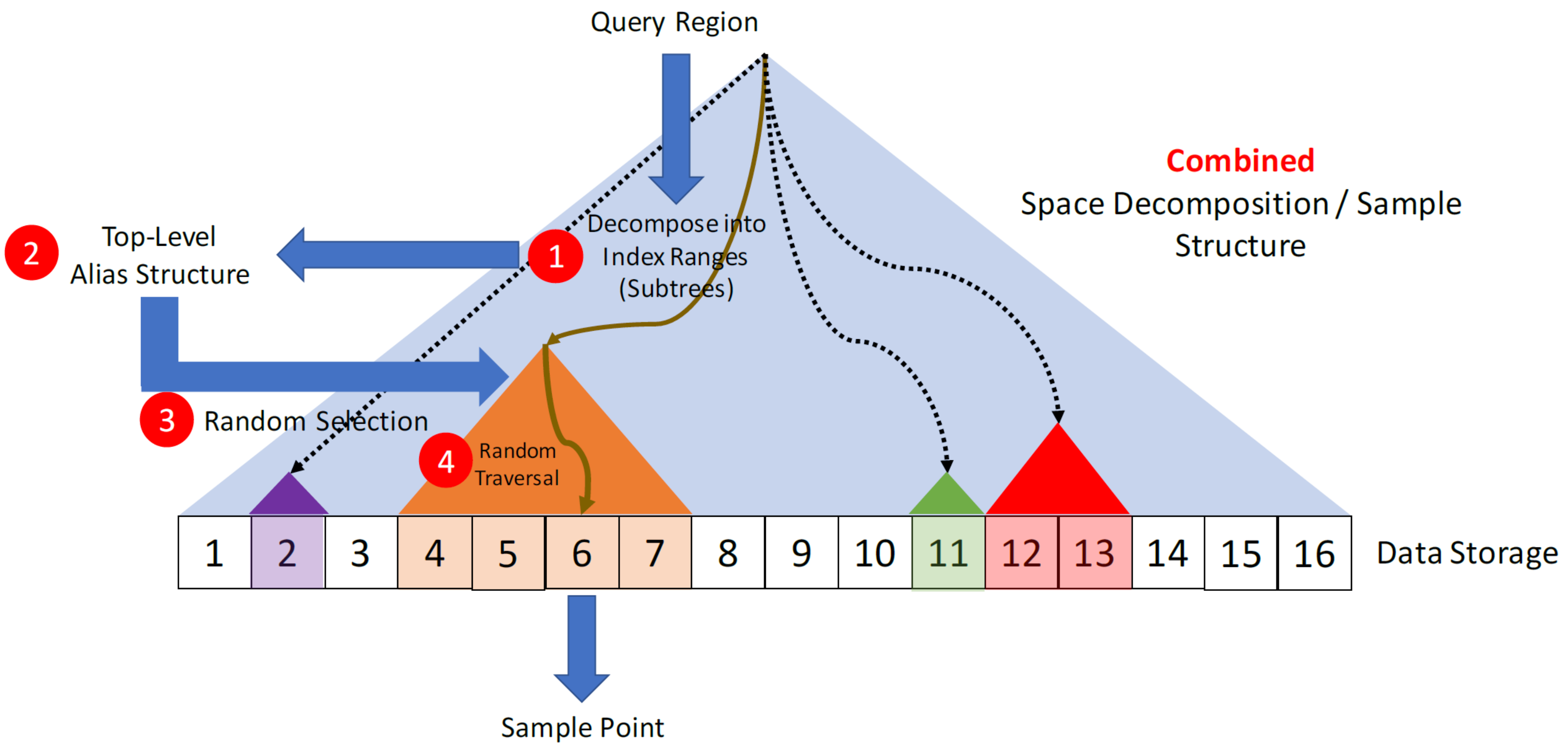
Weighted SIRS – Dual Tree Solution

- Reduction: Space Decomposition + Weighted 1D IRS.
- Theoretical best result: O(n) space cost, O(1) sample cost. ***NOT practical.***
- Practical weighted 1D IRS solution: avoiding rejections
  - Build a dyadic tree: query range -> a set of intervals
  - Pick a random interval -> traverse corresponding subtree.



Weighted SIRS – Combined Tree Solution

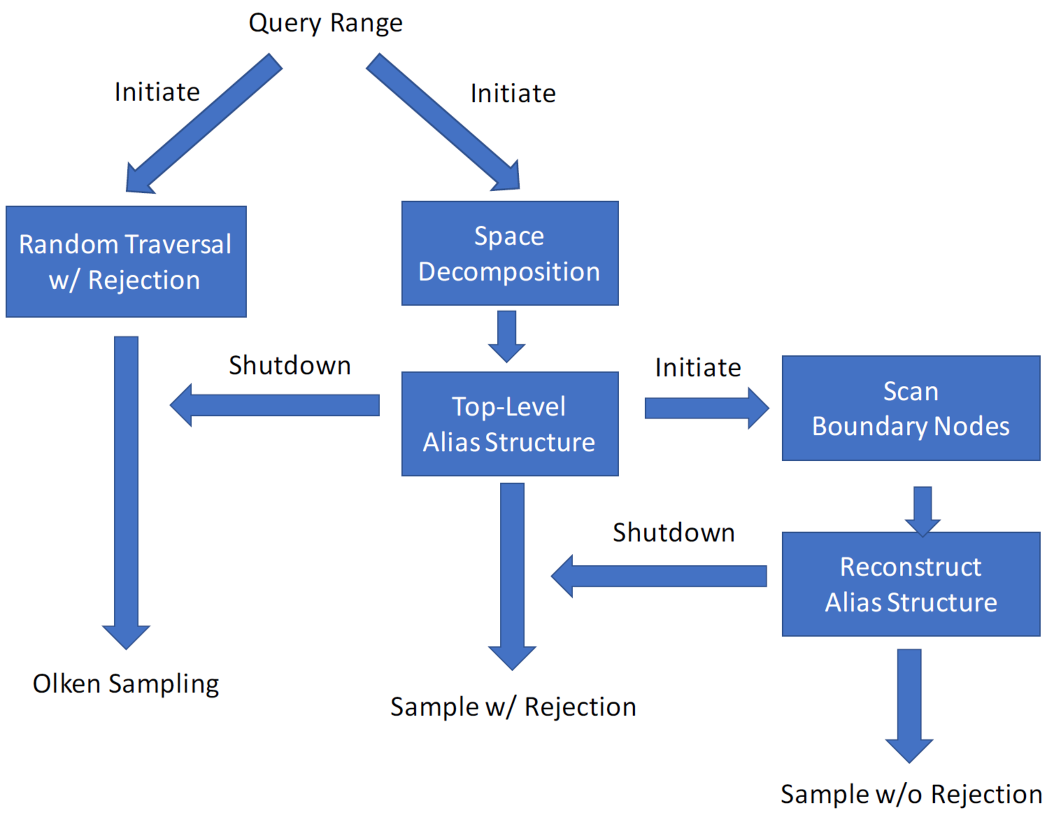
- Each **index range** generated by space decomposition map to **a subtree**.
- Direct traverse the subtree randomly.



Trade-off between Methods

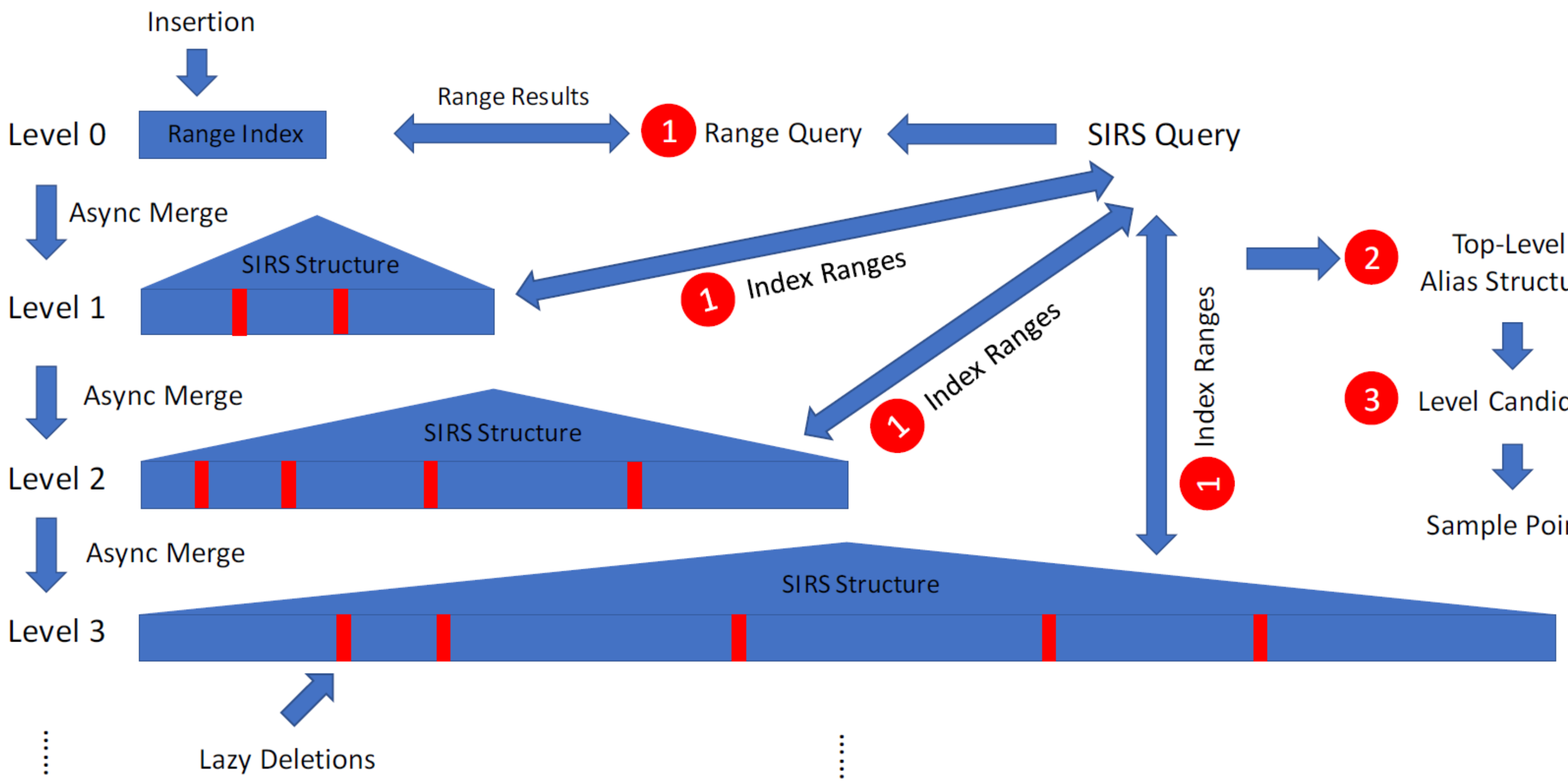
- Olken's Method: non-selective queries (> 10%), few number of samples (<100)
- Our solution: work for most cases, need a boost time.
- Can eliminate rejections to achieve higher throughput by scanning boundary leaf nodes.

Hybrid Method



Supporting Updates

- Incorporate the idea of LSM tree.
- Huge design space to explore.



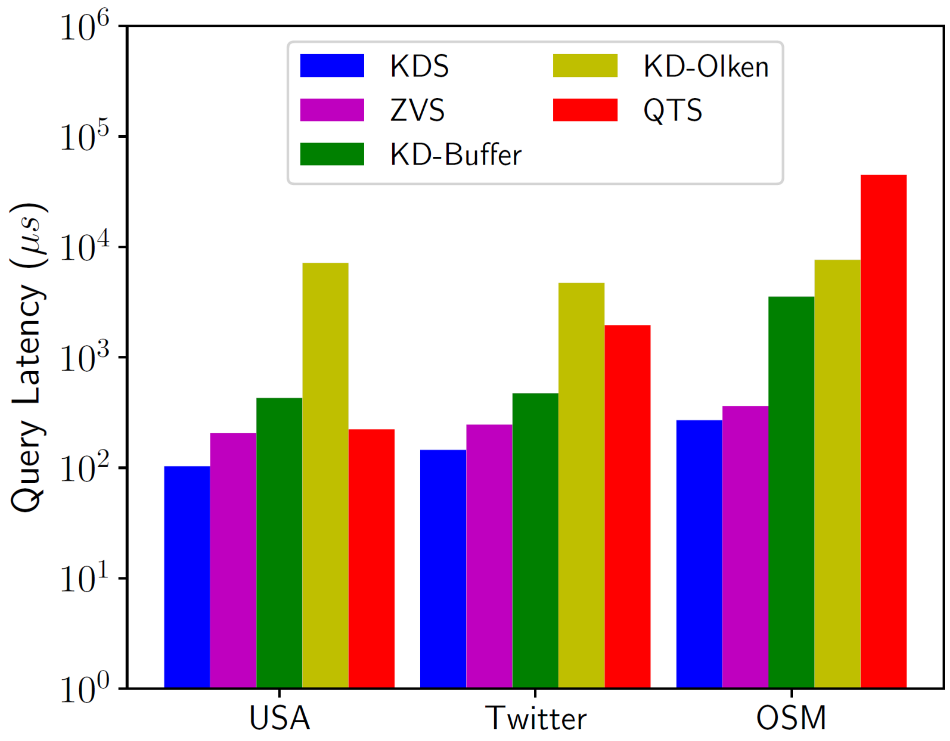
Evaluation

- Intel Xeon E5-2609 2.4GHz
- 256GB RAM, Rust 1.39.0, Pcg64Mcg RNG.

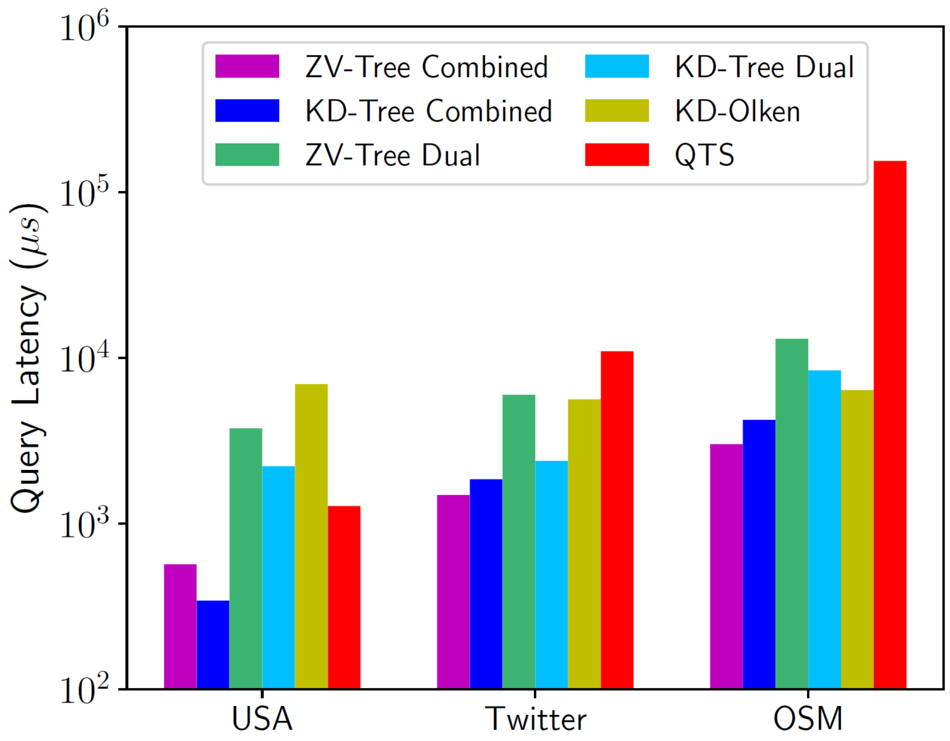
- USA: road network nodes, 24 million pts.
- Twitter: three-month tweets with geotag, 240 million pts.
- OSM: OpenStreetMap POIs, 2.68 billion pts.

- Sample size = 1000
- 0.1% selectivity square region
- 1000 query average

Query Performance



Uniform



Weighted

KDS = KD-Tree Sampling Method
ZVS = Z-Value Sampling Method
KD-Buffer = Buffer Sampling on KD-Tree
KD-Olken = Olken Method on KD-Tree
QTS = Query Then Sampling

Query CPU Breakdown

KDS = Kd-Tree Sampling Method
ZVS = Z-Value Sampling Method
KD-Buffer = Buffer Sampling on KD-Tree
KD-Olken = Olken Method on KD-Tree
QTS = Query Then Sampling

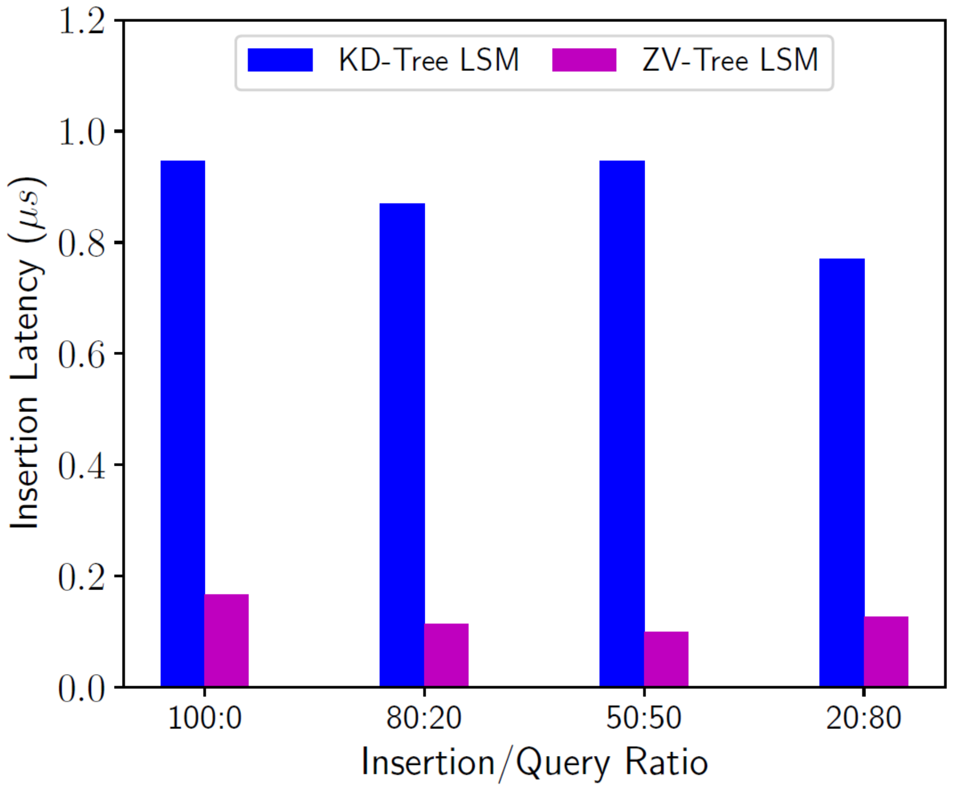
Uniform

Method	Tot Latency (μs)	CPU Breakdown (μs / %)		
		Effective RNGs	Wasted RNGs	Other Major Components
QTS	1892.64	11.20 (0.60%)	0.00 (0.00%)	Query Time: 1881.44 (99.41%)
KD-Olken w/o LCA	62078.03	642.03 (1.03%)	61435.55 (98.97%)	-
KD-Olken w/ LCA	4981.30	477.31 (9.58%)	4411.35 (88.56%)	LCA Optimization: 2.64 (0.05%);
KD-Buffer	798.56	8.69 (1.09%)	2.97 (0.37%)	Buffer Replenish: 270.53 (57.95%);
KDS w/ Rejection	140.26	99.45 (70.90%)	6.73 (4.80%)	Alias Construction: 23.80 (16.96%);
KDS w/o Rejection	396.30	98.24 (24.79%)	0.00 (0.00%)	Alias Construction: 289.79 (73.12%);

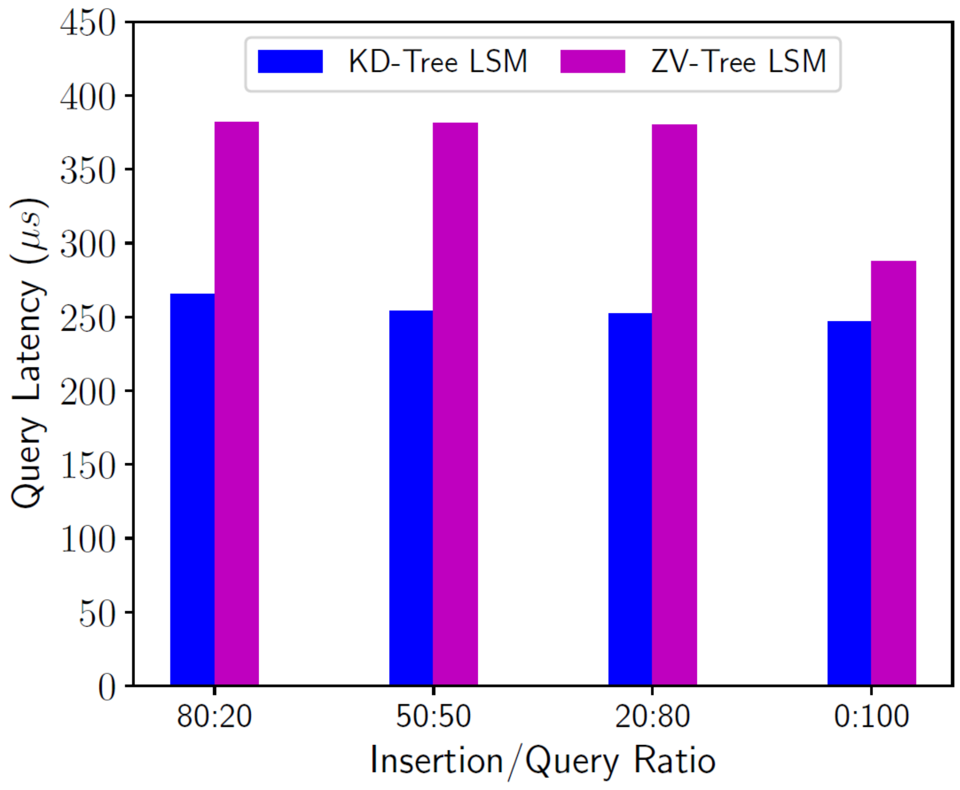
Weighted

Method	Tot Latency (μs)	CPU Breakdown (μs / %)		
		Effective RNGs	Wasted RNGs	Other Major Components
QTS	11128.86	112.41 (1.10%)	0.00 (0.00%)	Query Time: 11006.45 (98.90%)
KD-Olken w/o LCA	70328.76	483.38 (0.69%)	69844.77 (99.32%)	-
KD-Olken w/ LCA	5770.88	355.40 (6.16%)	5412.44 (93.79%)	LCA Optimization: 3.04 (0.05%)
KD-Tree Dual w/ Rej	2491.19	2293.56 (92.07%)	115.31 (4.62%)	Alias Construction: 79.80 (3.20%)
KD-Tree Dual w/o Rej	3143.37	2242.30 (71.33%)	0.00 (0.00%)	Alias Construction: 896.03 (28.51%)
KD-Tree Combined w/ Rej	1245.58	1137.30 (91.31%)	36.29 (2.91%)	Alias Construction: 70.56 (5.66%)
KD-Tree Combined w/o Rej	1356.54	491.69 (36.24%)	0.00 (0.00%)	Alias Construction: 863.08 (63.62%)

Update Support with LSM



Insertion Latency



Query Latency

Summary

- **Approximation approach** to achieve interactive spatial data analysis
- Independent sampling is **foundation operation**.
- Sampling framework: multi-dimension problem to 1D reduction.
- Different **space decomposition**: Z-Value, KD-Tree, general spatial index
- Extension to weighted SIRS: dual-tree / combined-tree solution.
- Key principles: ***minimize RNG calls***, ***avoid rejection***.
- Trade-offs -> hybrid method.
- LSM-tree based update support.

- **1-3 orders of magnitude** performance improvement!

Backup

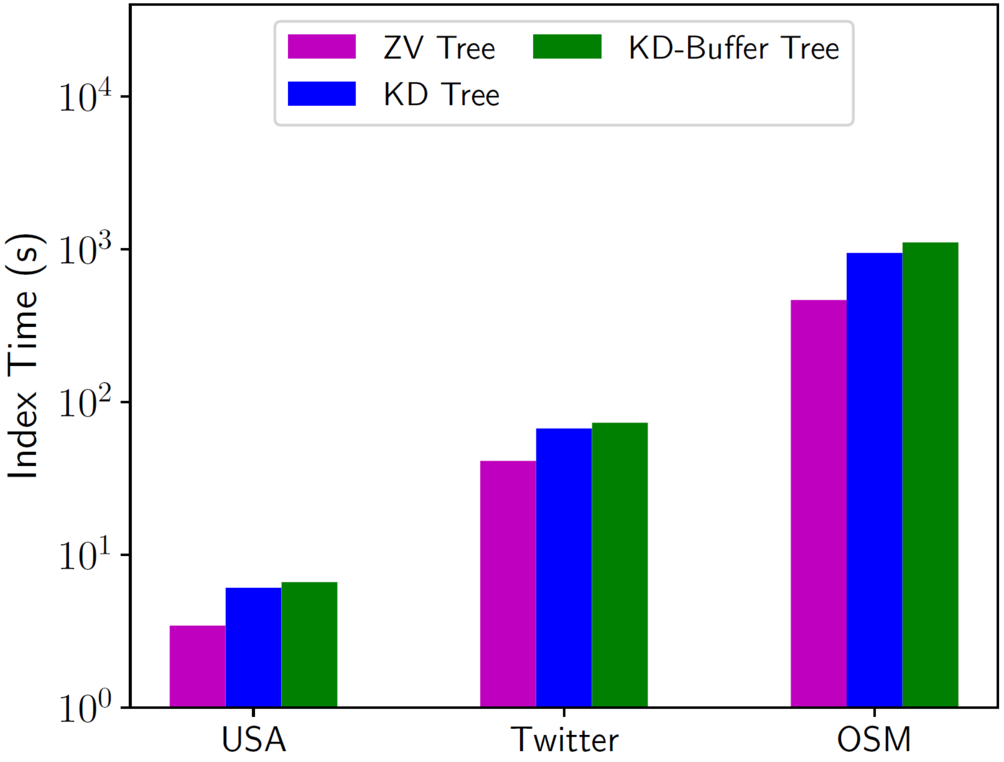
Cost of Rejection Sampling

- In Olken, ~90% of CPU time is wasted due to rejection sampling
- In Uniform KDS and ZVS, <7% CPU time is wasted in rejection

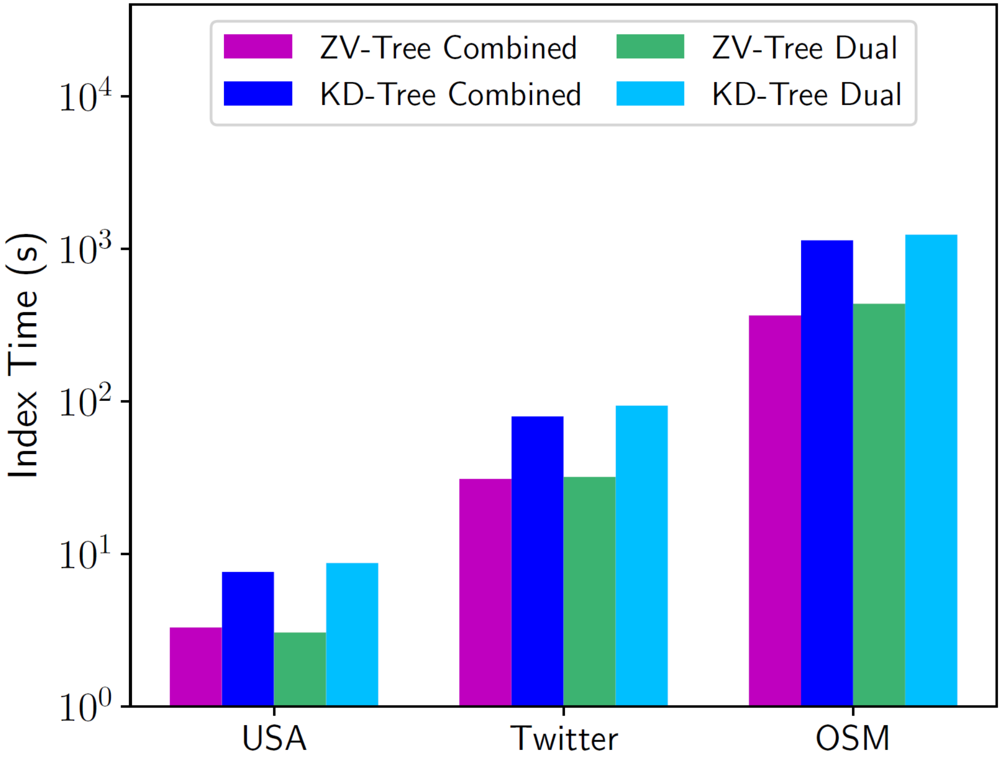
- Fast pseudo RNG Pcg64Mcg: ~13 billion RNG calls/s
- Crypto-safe RNG: ~61 million RNG calls/s (213x slower!!!)

- Our method can get rid of rejection totally
- Scanning boundary leaf nodes -> put data points inside query range
- Separate candidate pool

Index Building Time



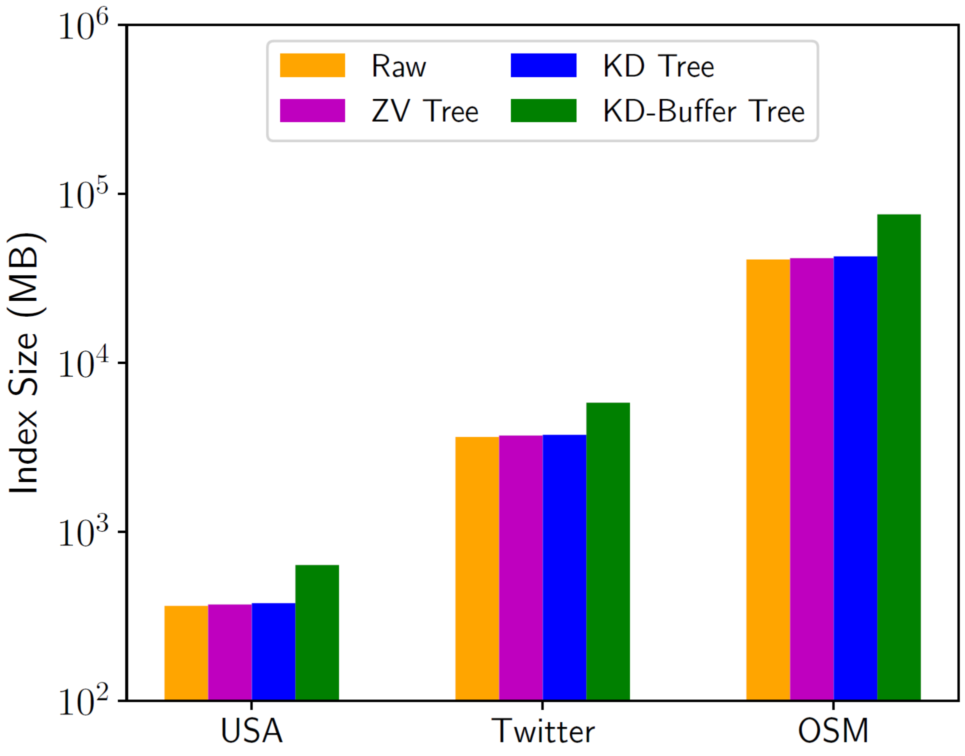
Uniform



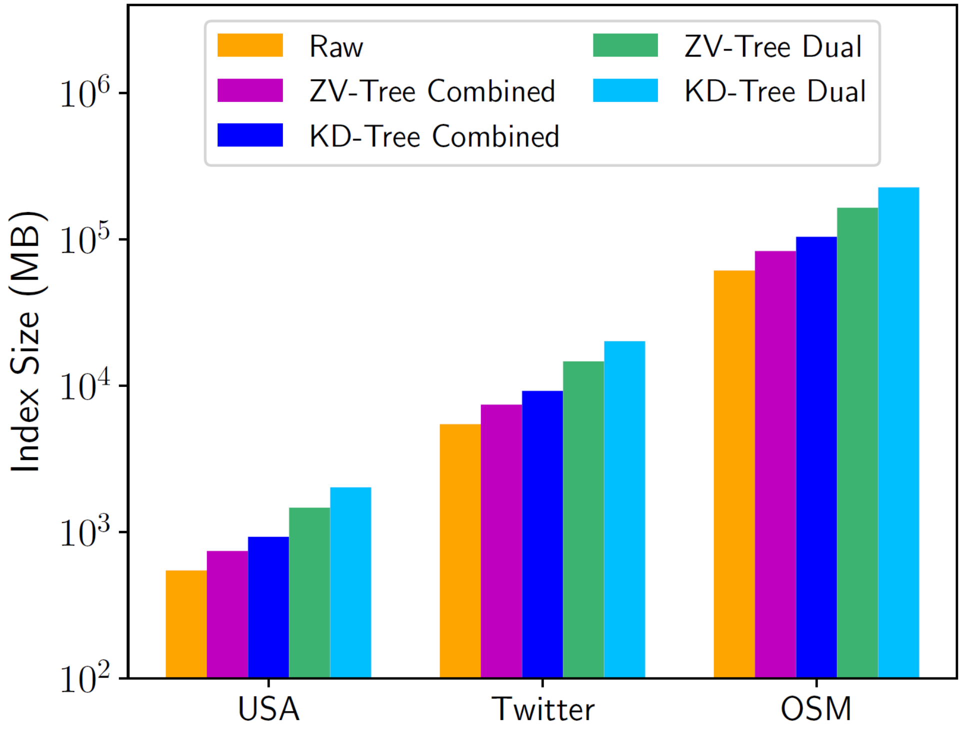
Weighted

KDS = KD-Tree Sampling Method
ZVS = Z-Value Sampling Method
KD-Buffer = Buffer Sampling on KD-Tree
KD-Olken = Olken Method on KD-Tree
QTS = Query Then Sampling

Index Size



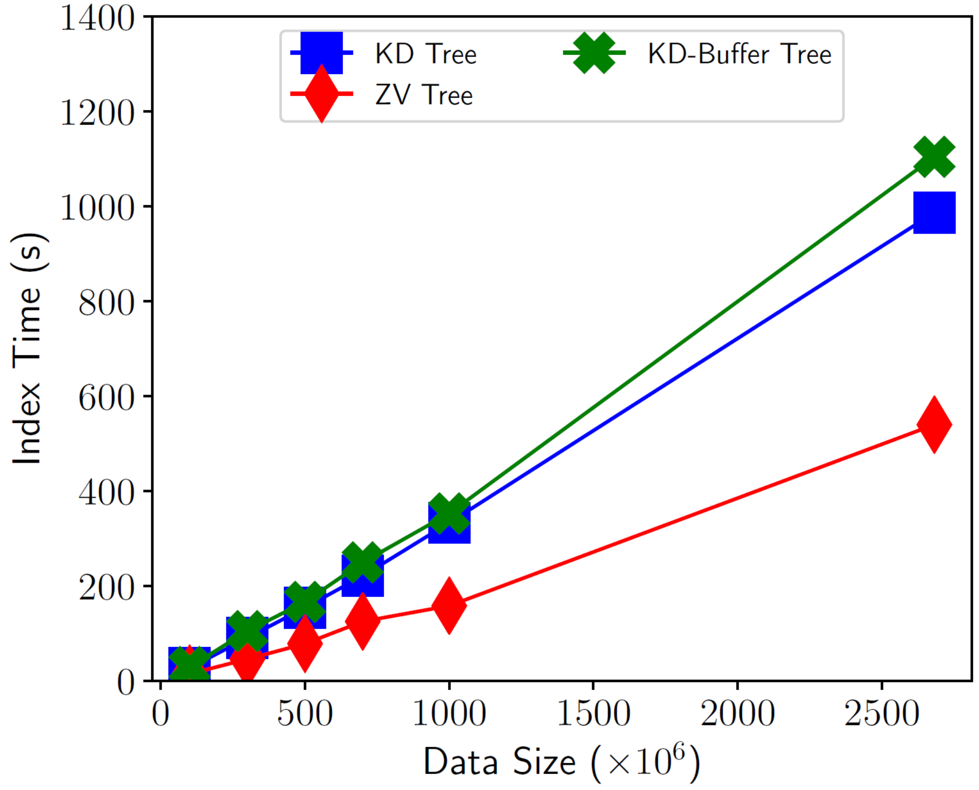
Uniform



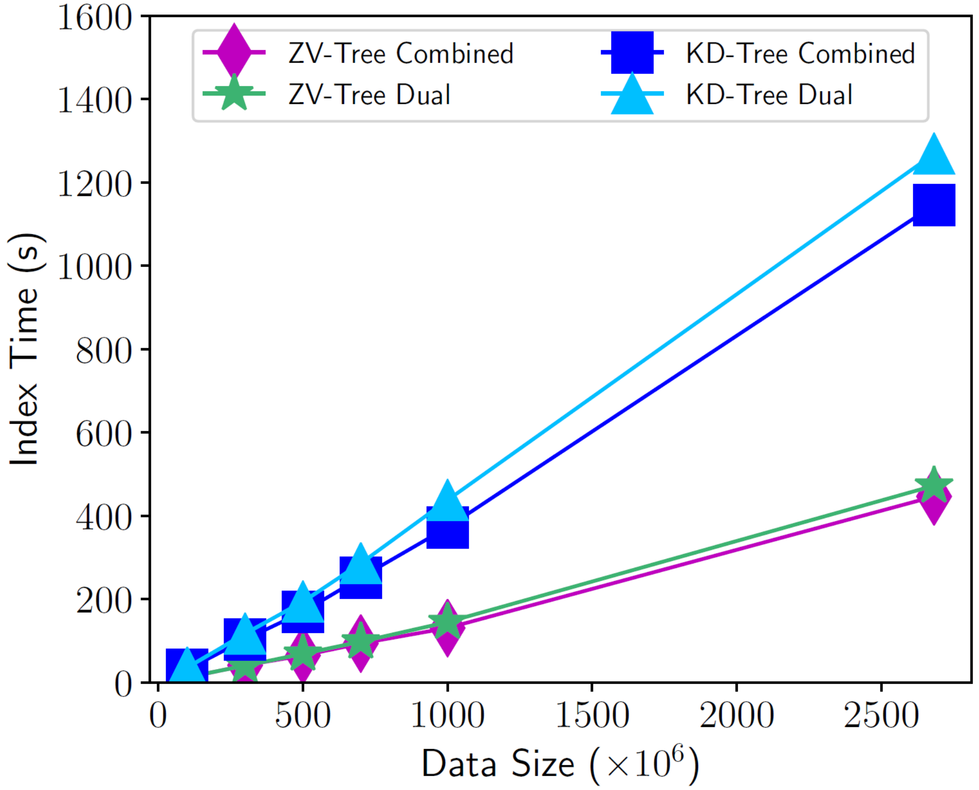
Weighted

KDS = KD-Tree Sampling Method
KD-Buffer = Buffer Sampling on KD-Tree
QTS = Query Then Sampling
ZVS = Z-Value Sampling Method
KD-Olken = Olken Method on KD-Tree

Scalability – Index Building Time



Uniform



Weighted

KDS = KD-Tree Sampling Method

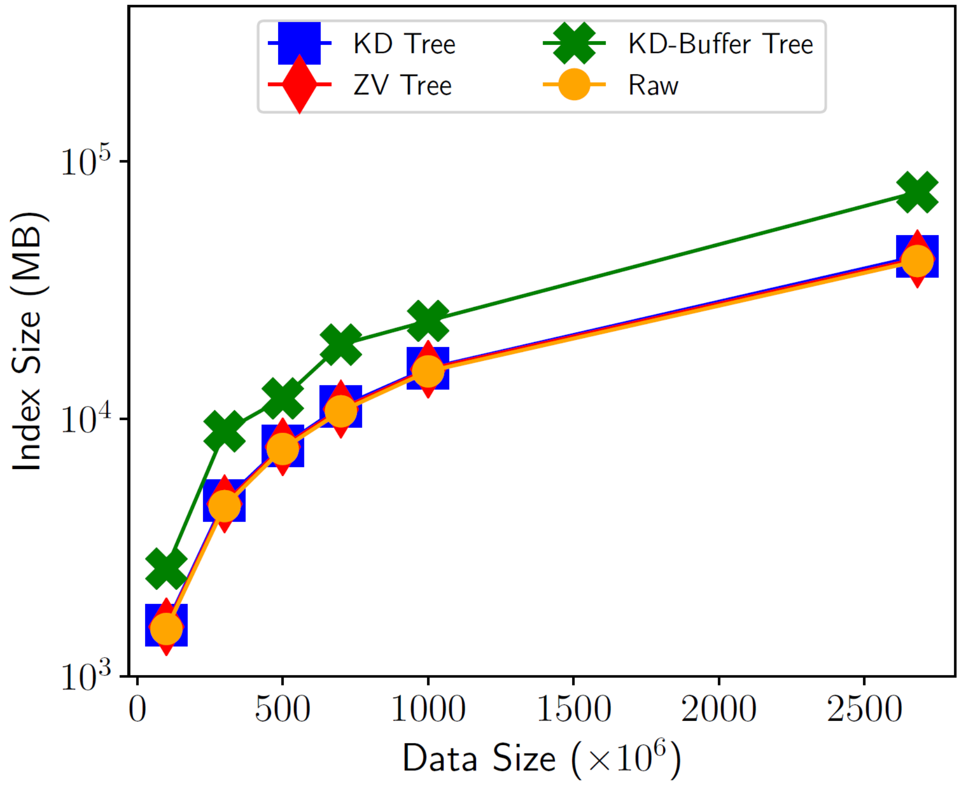
KD-Buffer = Buffer Sampling on KD-Tree

QTS = Query Then Sampling

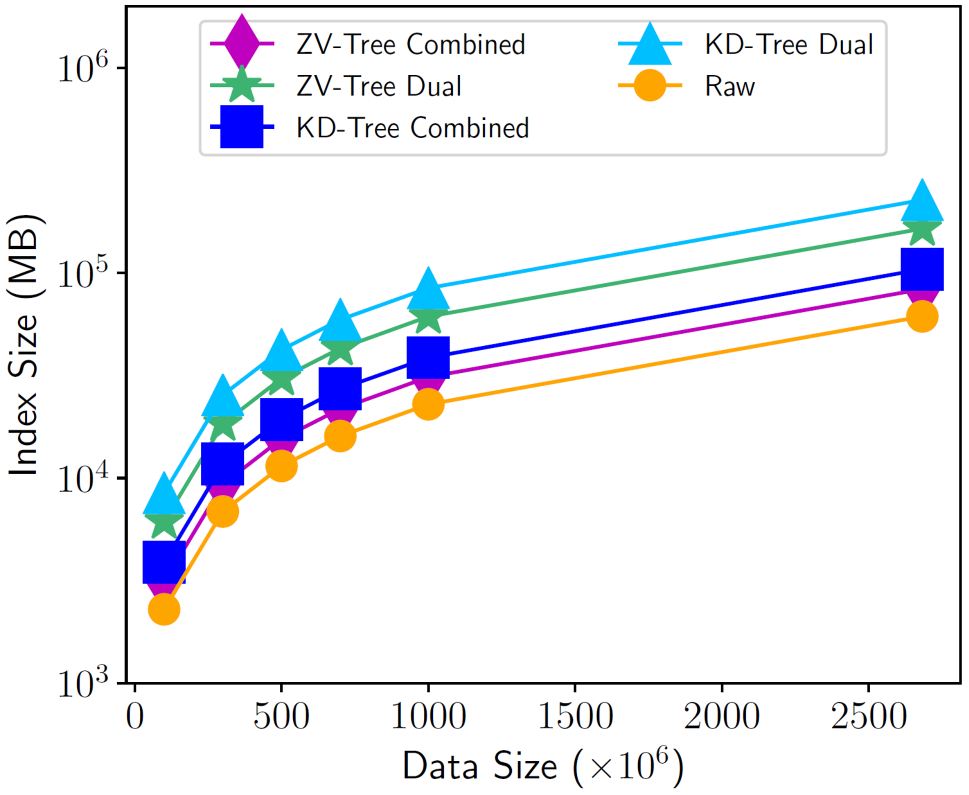
ZVS = Z-Value Sampling Method

KD-Olken = Olken Method on KD-Tree

Scalability – Index Size



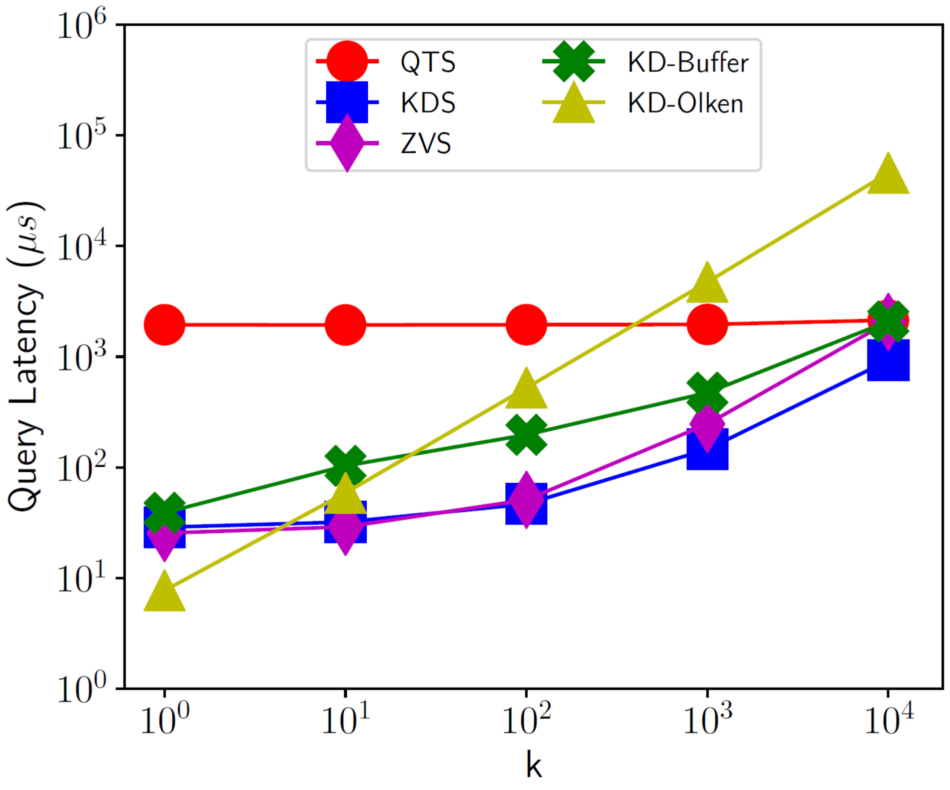
Uniform



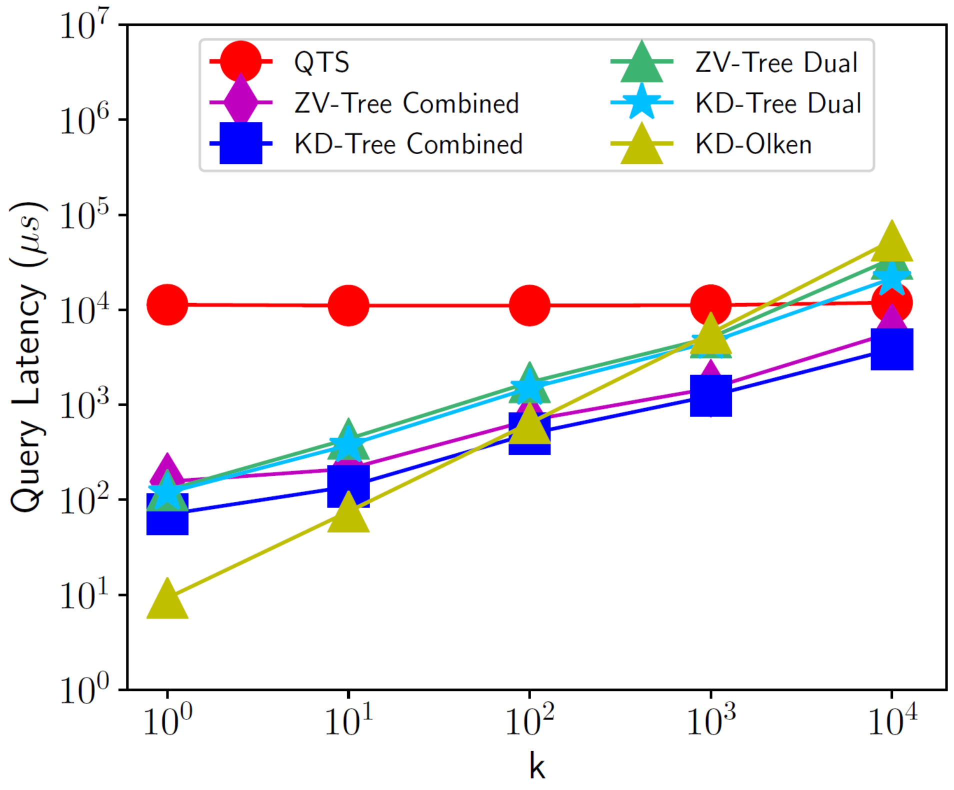
Weighted

KDS = KD-Tree Sampling Method
ZVS = Z-Value Sampling Method
KD-Buffer = Buffer Sampling on KD-Tree
KD-Olken = Olken Method on KD-Tree
QTS = Query Then Sampling

Effect of k



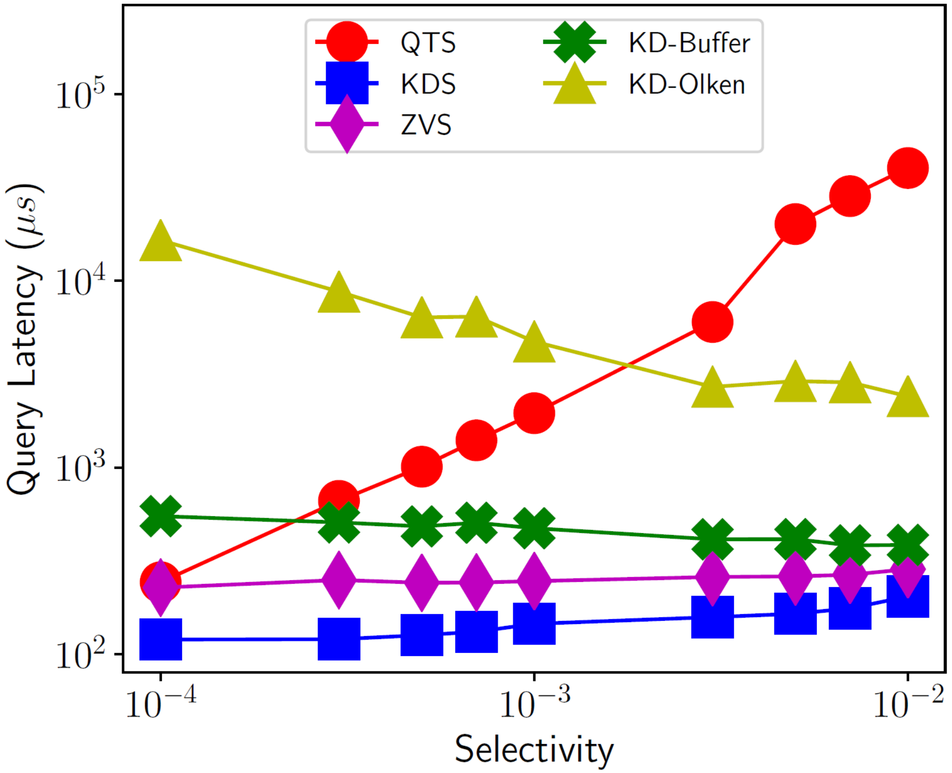
Uniform



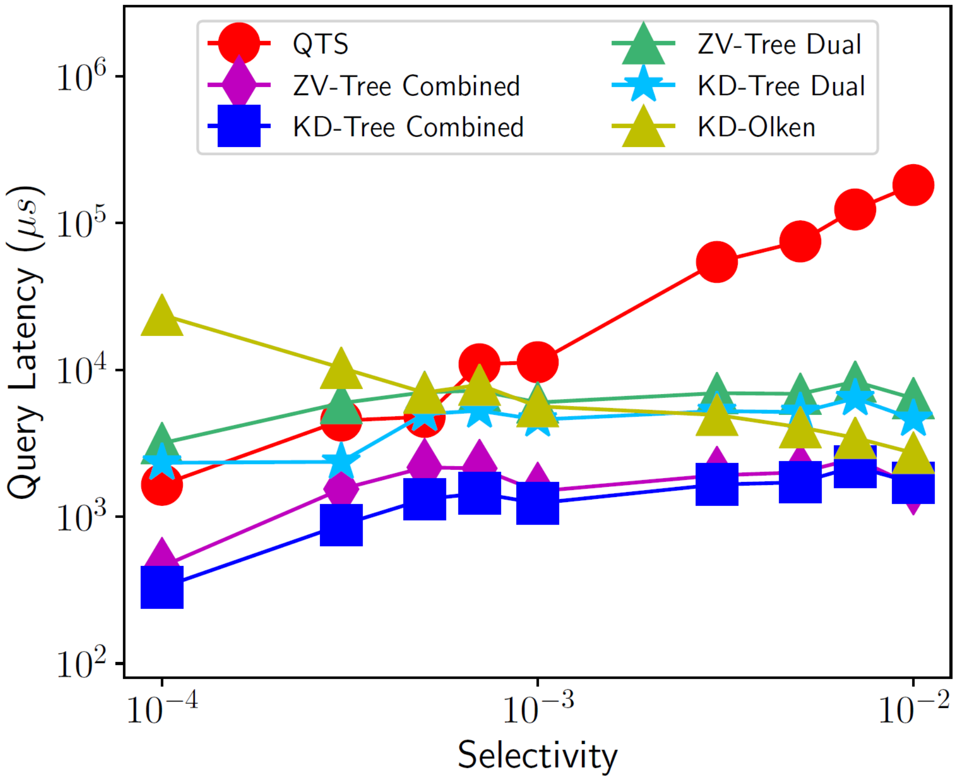
Weighted

KDS = <u>KD-Tree Sampling Method</u>
KD-Buffer = <u>Buffer Sampling on KD-Tree</u>
QTS = <u>Query Then Sampling</u>
ZVS = <u>Z-Value Sampling Method</u>
KD-Olken = <u>Olken Method on KD-Tree</u>

Effect of selectivity



Uniform



Weighted

KDS = KD-Tree Sampling Method

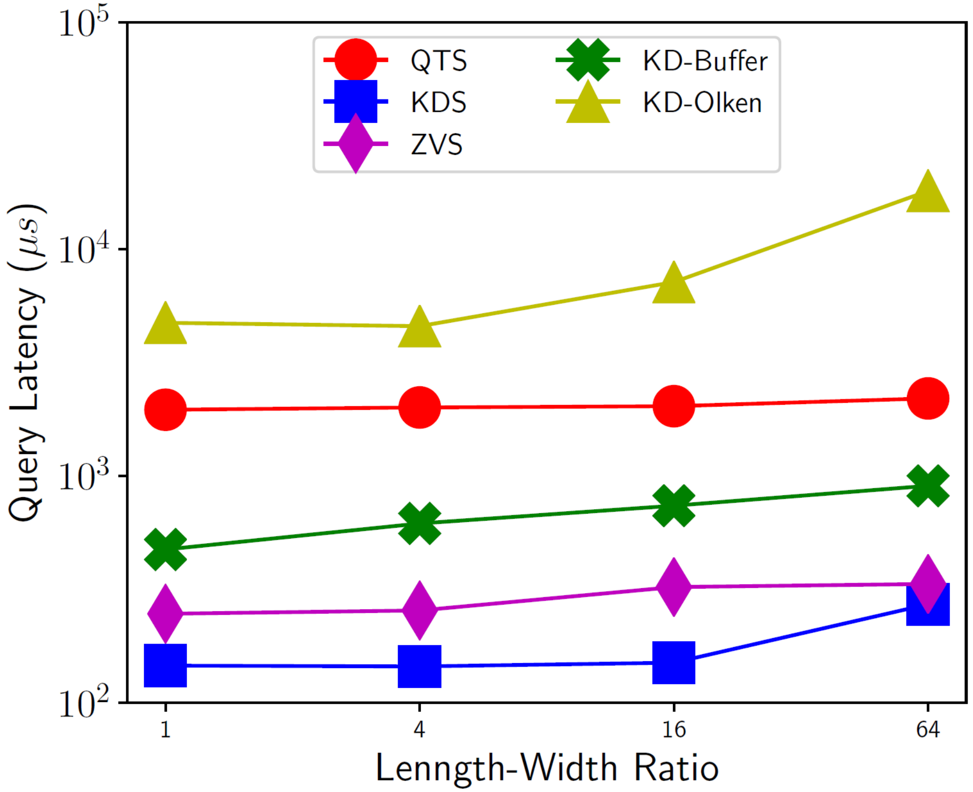
KD-Buffer = Buffer Sampling on KD-Tree

QTS = Query Then Sampling

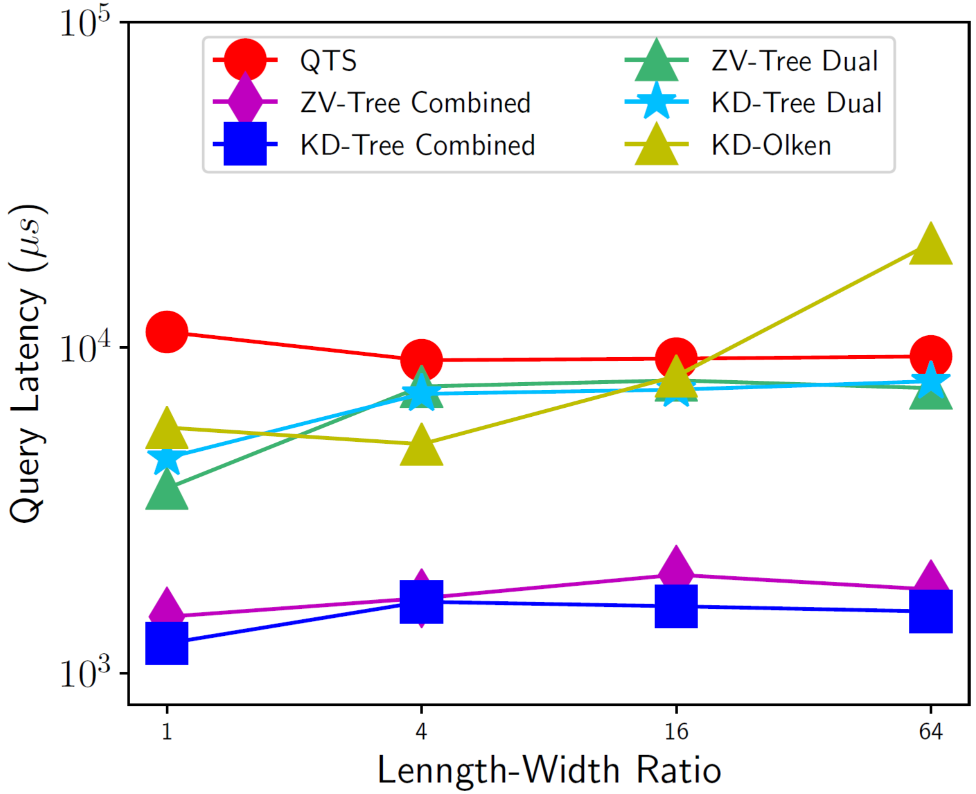
ZVS = Z-Value Sampling Method

KD-Olken = Olken Method on KD-Tree

Effect of range fatness



Uniform



Weighted

KDS = KD-Tree Sampling Method
KD-Buffer = Buffer Sampling on KD-Tree
QTS = Query Then Sampling
ZVS = Z-Value Sampling Method
KD-Olken = Olken Method on KD-Tree

Hybrid Method

KDS = KD-Tree Sampling Method
Comp = Combined Tree Method on KD-Tree
Olken = Olken Method on KD-Tree

Uniform

Method	# Samples Retrieved by Timeline (μs)						
	74	91	443	461	1000	3000	5000
Olken	86	106	517	538	1166	3498	5830
KDS w/ Rej	0	287	6237	6541	15651	49454	83257
KDS w/o Rej	0	0	0	364	11263	51706	92149
Hybrid	82	101	882	922	11877	52524	93172

Weighted

Method	# Samples Retrieved by Timeline (μs)						
	109	127	1570	1974	3000	5000	10000
Olken	222	243	2542	3045	4477	7962	14923
Comp w/ Rej	0	72	5855	7474	11585	19600	39636
Comp w/o Rej	0	0	0	1774	6279	15060	37014
Hybrid	216	252	3622	4566	9078	17875	39866