QUAKE

Adaptive Indexing for Vector Search

Jason Mohoney, Devesh Sarda, Mengze Tang, Anil Pacaci*, Shihab Chowdhury*, Ihab Ilyas*, Theodoros Rekatsinas*, Shivaram Venkataraman

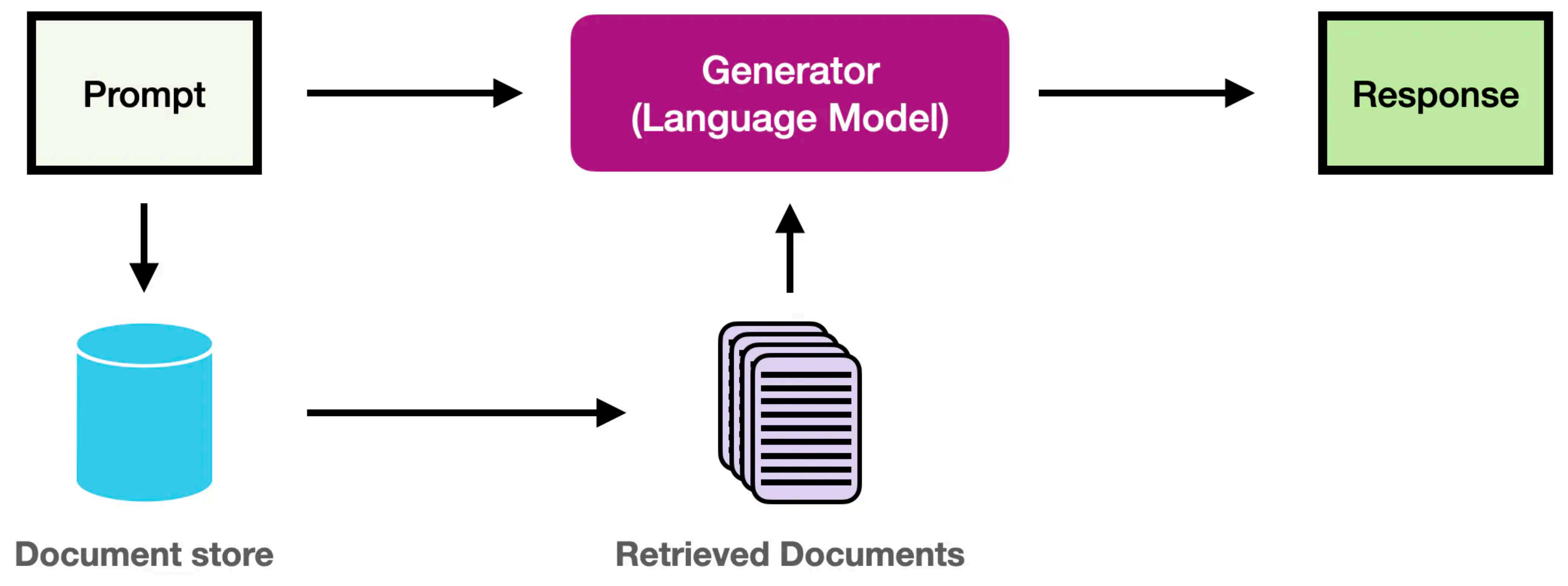
WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

*Apple

Vector Search is Everywhere

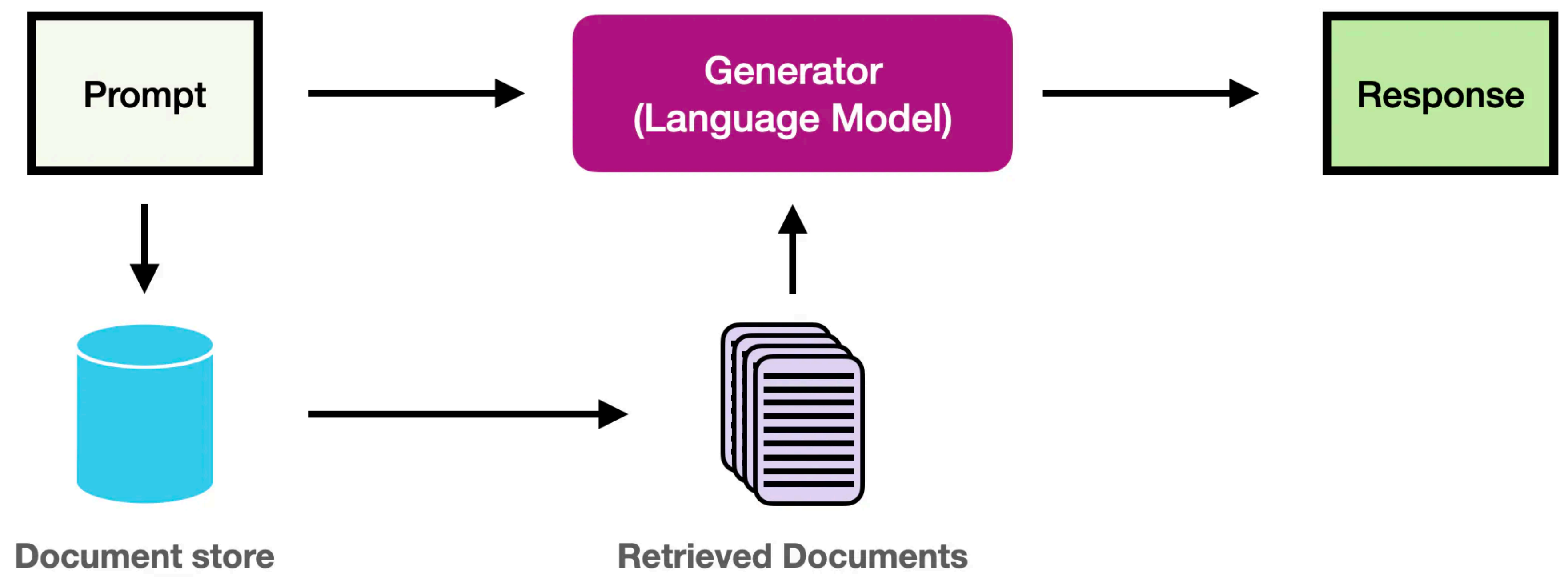
Vector Search is Everywhere

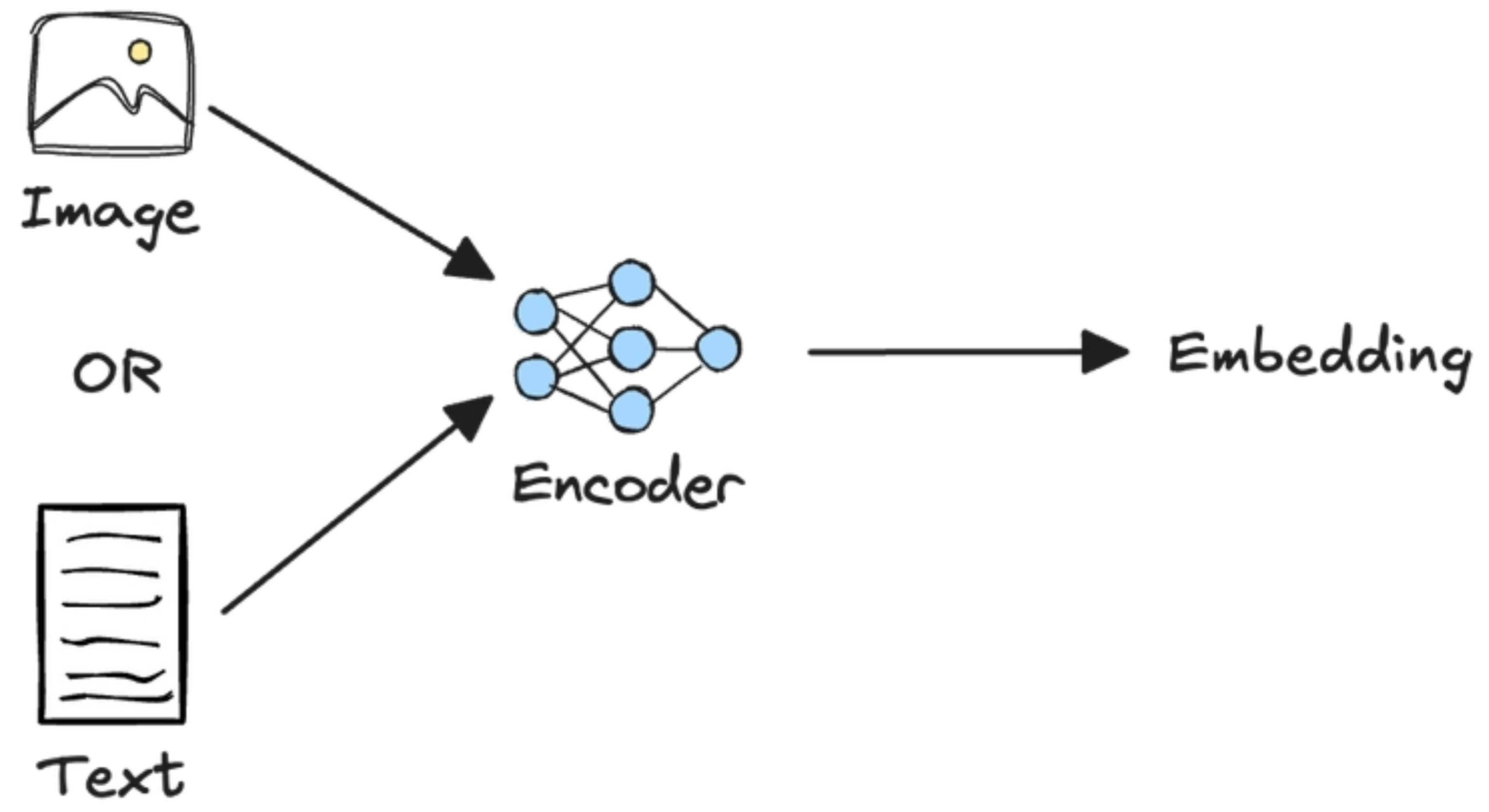
Retrieval Augmented Generation



Vector Search is Everywhere

Retrieval Augmented Generation

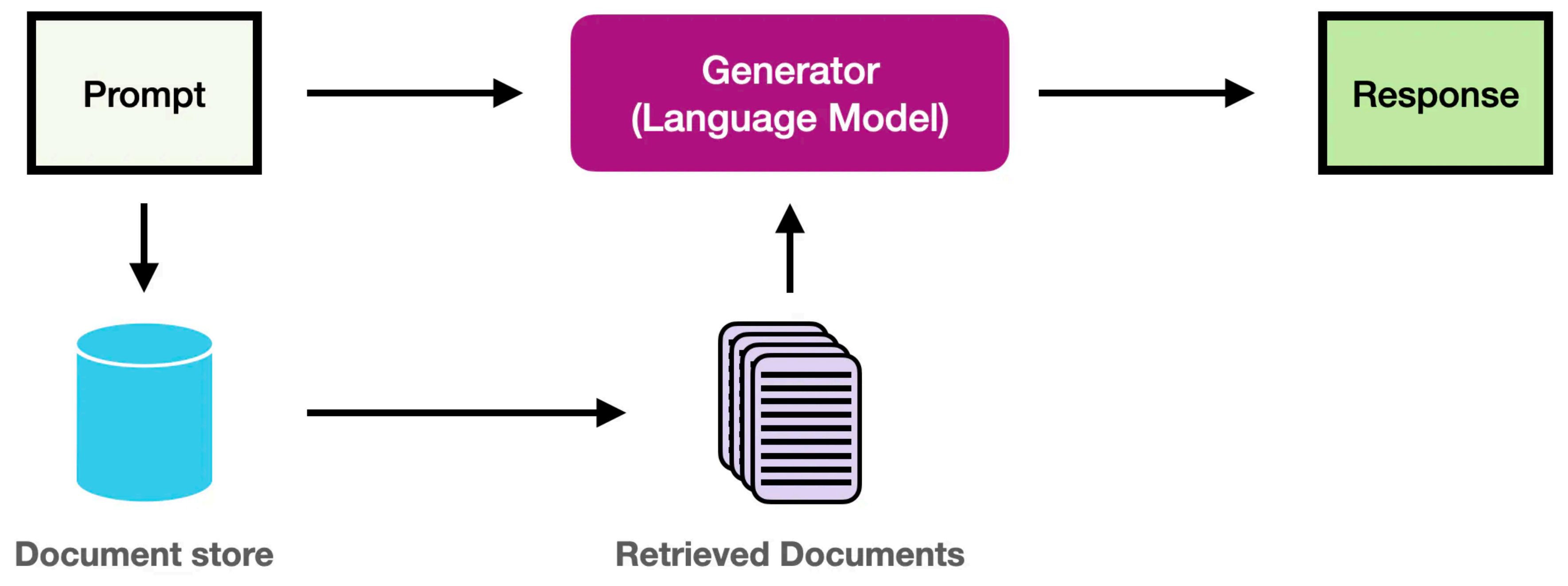


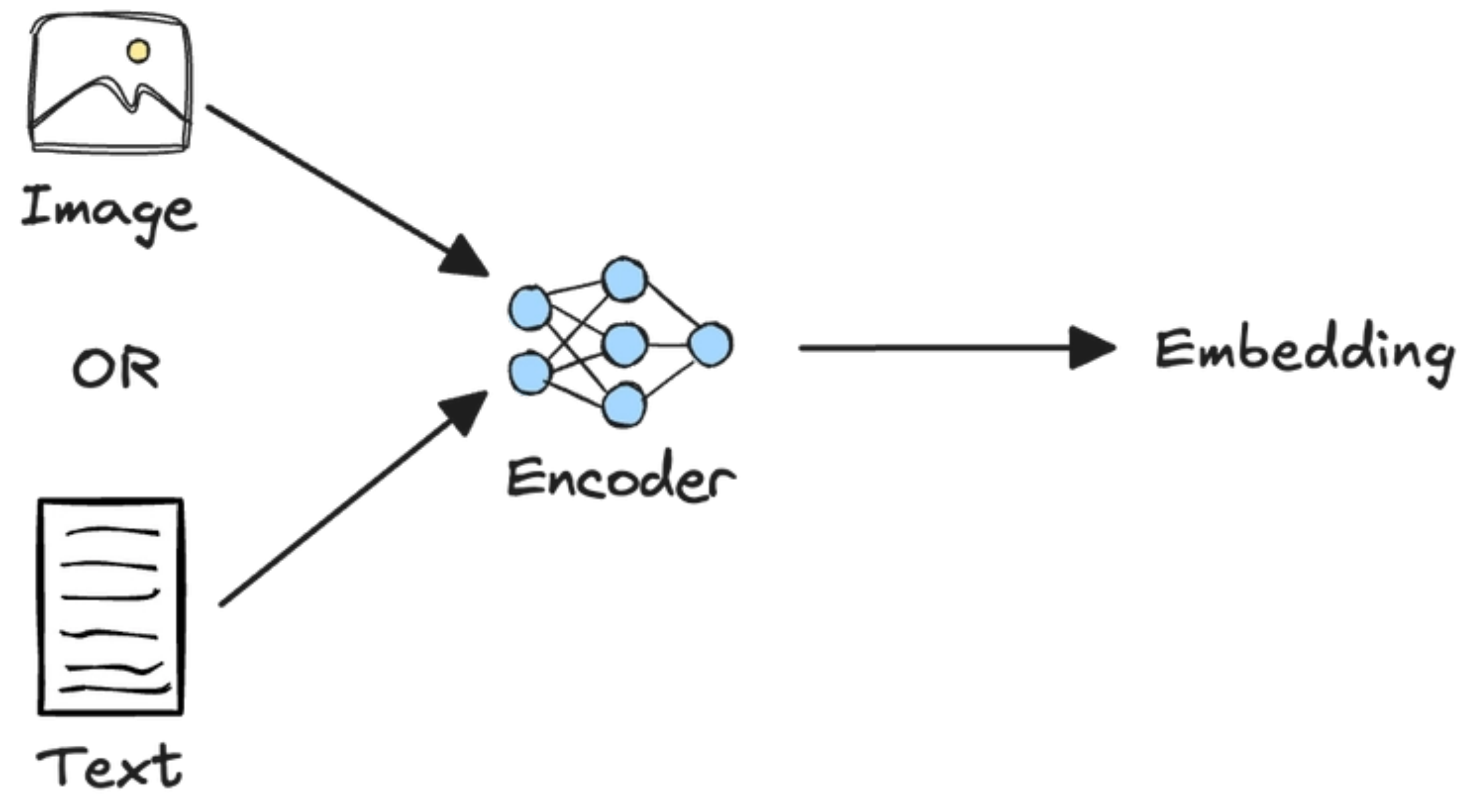


Multi-modal Search

Vector Search is Everywhere

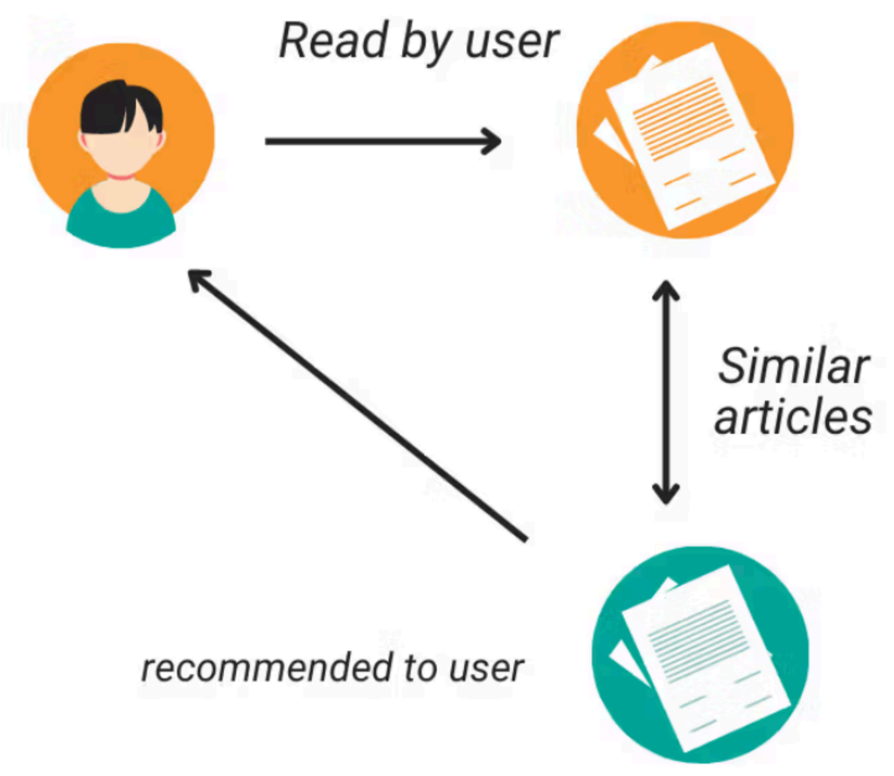
Retrieval Augmented Generation





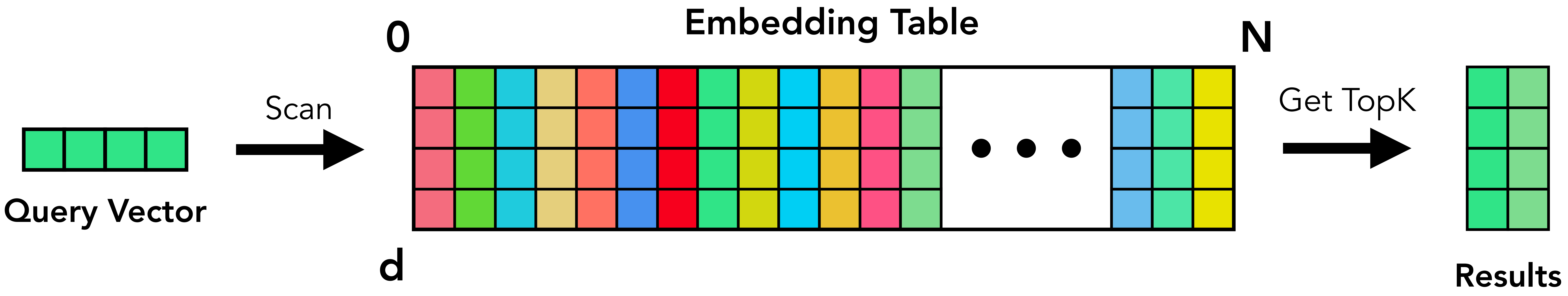
Multi-modal Search

Recommendation



Vector Search

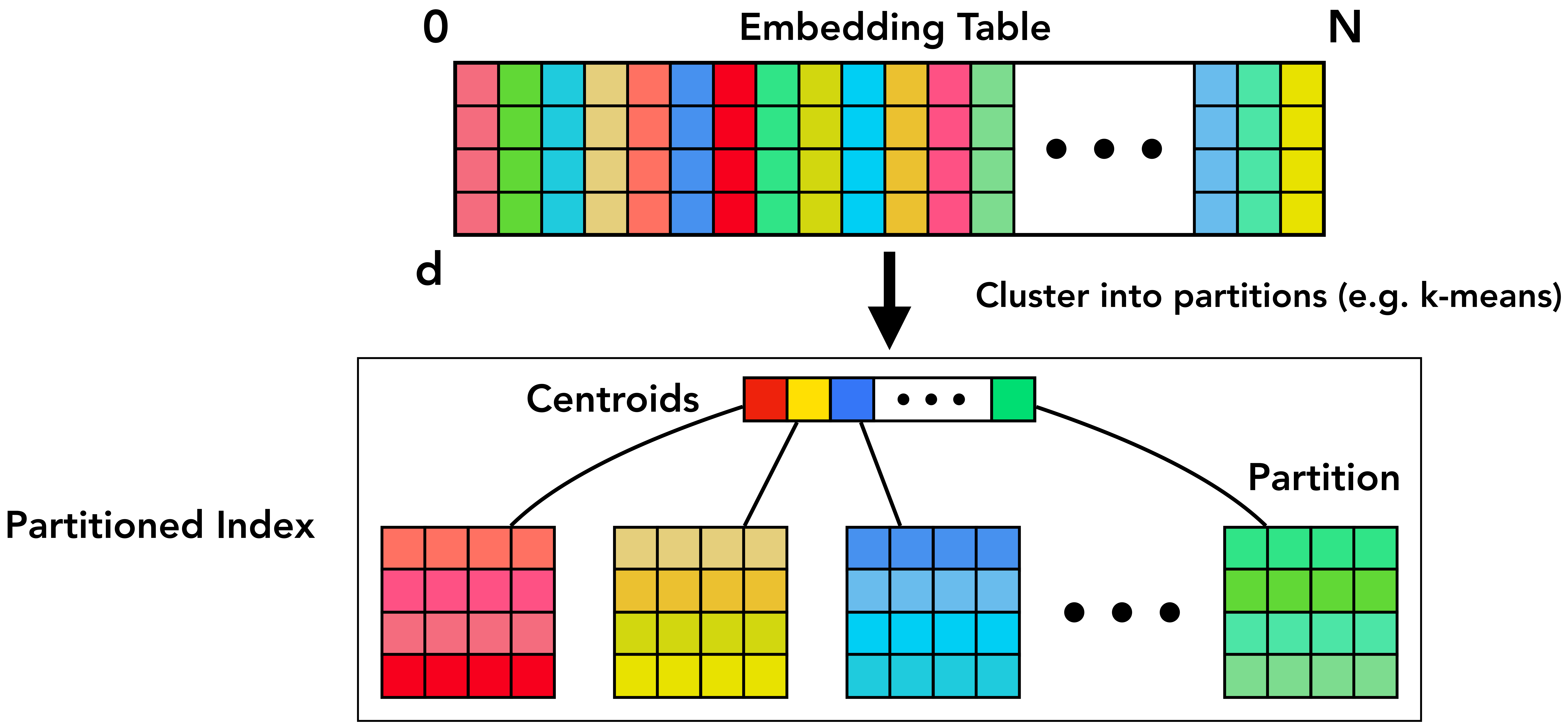
Get top-k nearest vectors to query vector by a similarity metric



Brute-force computationally infesible for large sets of vectors

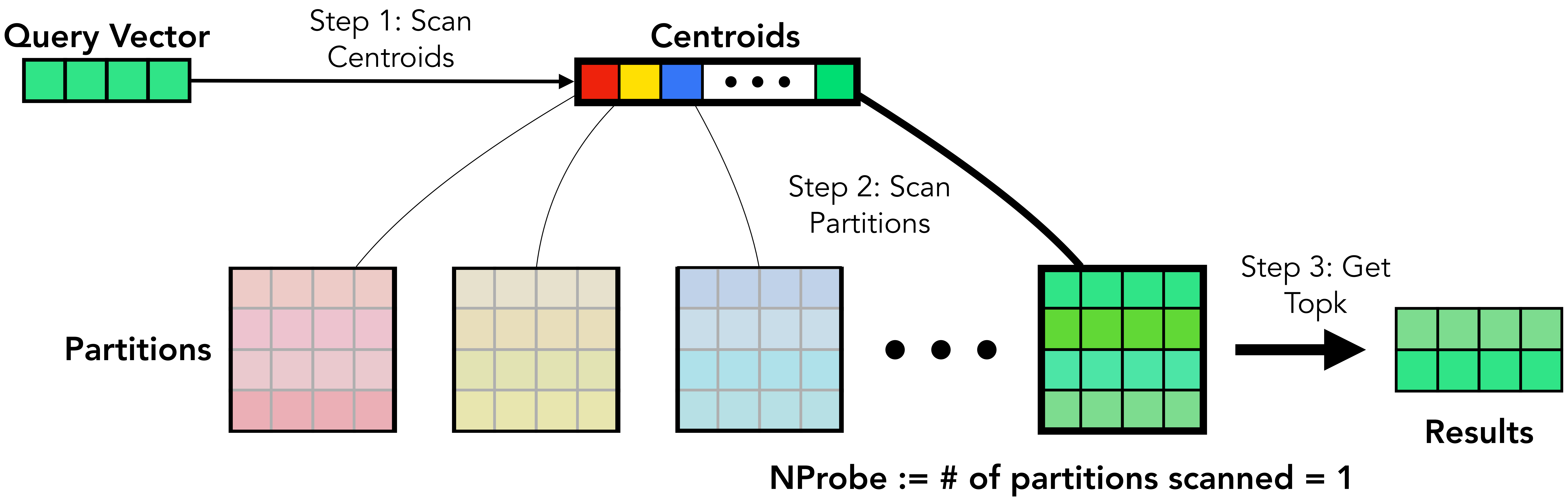
Partitioned Vector Search Index (e.g. Faiss-IVF)

Approximate indexes allow for scalable vector search



Partitioned Index: Search

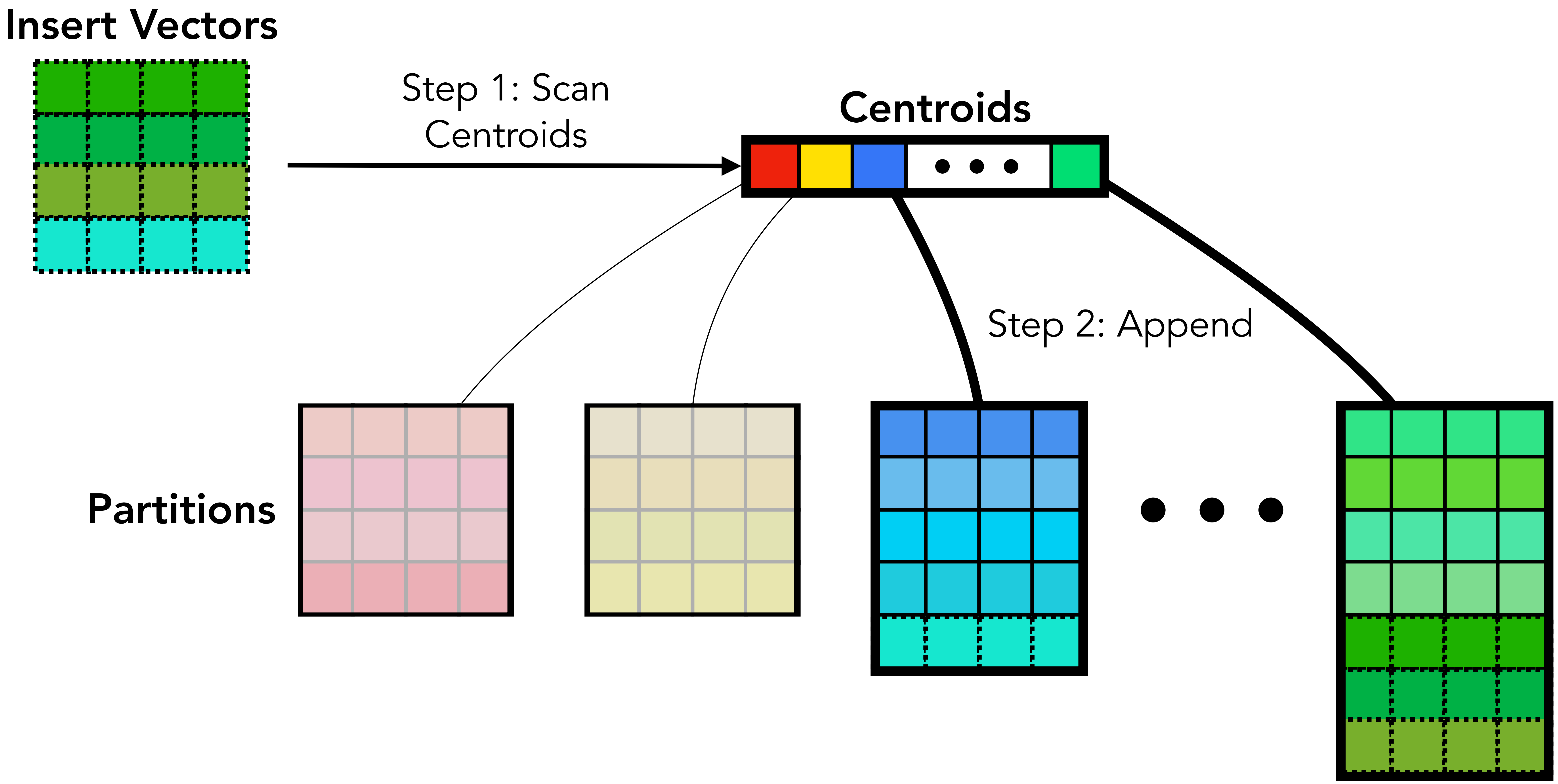
Queries scan a subset of partitions based on the nearest centroids



NProbe controls recall vs. query latency trade-off

Partitioned Index: Insert

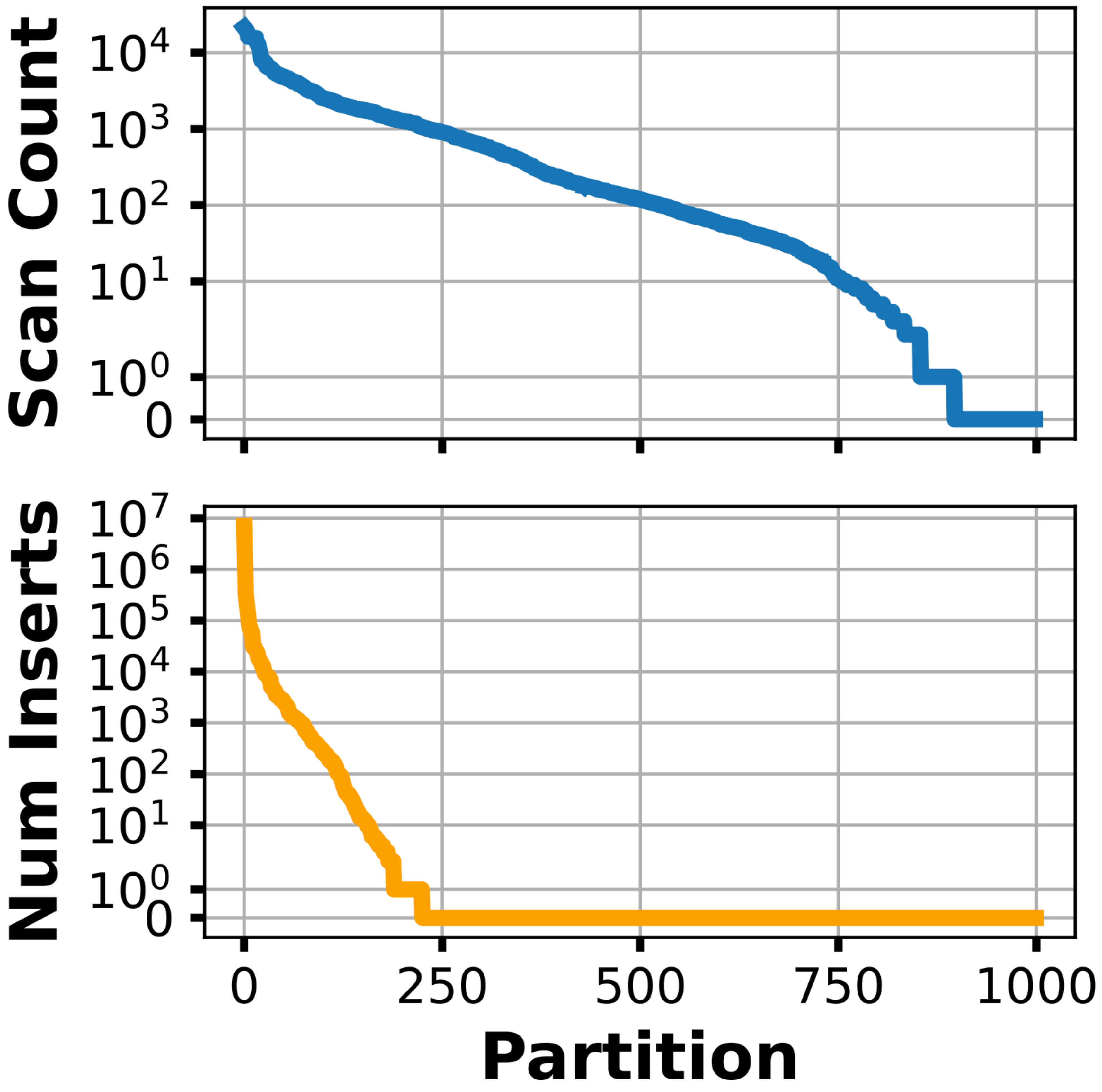
Inserted vectors are appended to nearest partitions



Index updates cause imbalance

Skew in Partitioned Indexes

Workload: Wikipedia-12M



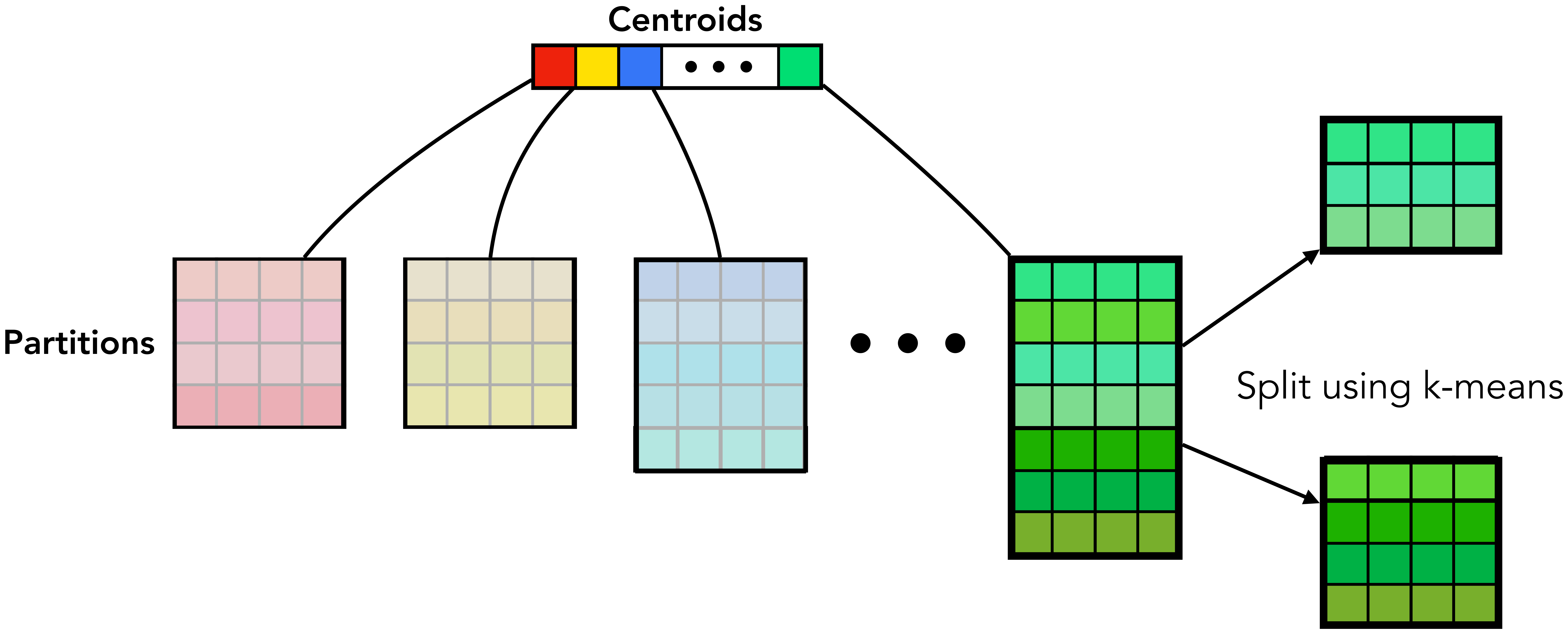
(a) Read (top) and write skew.

Certain partitions are scanned more frequently by searches

Certain partitions are updated more frequently, causing imbalance

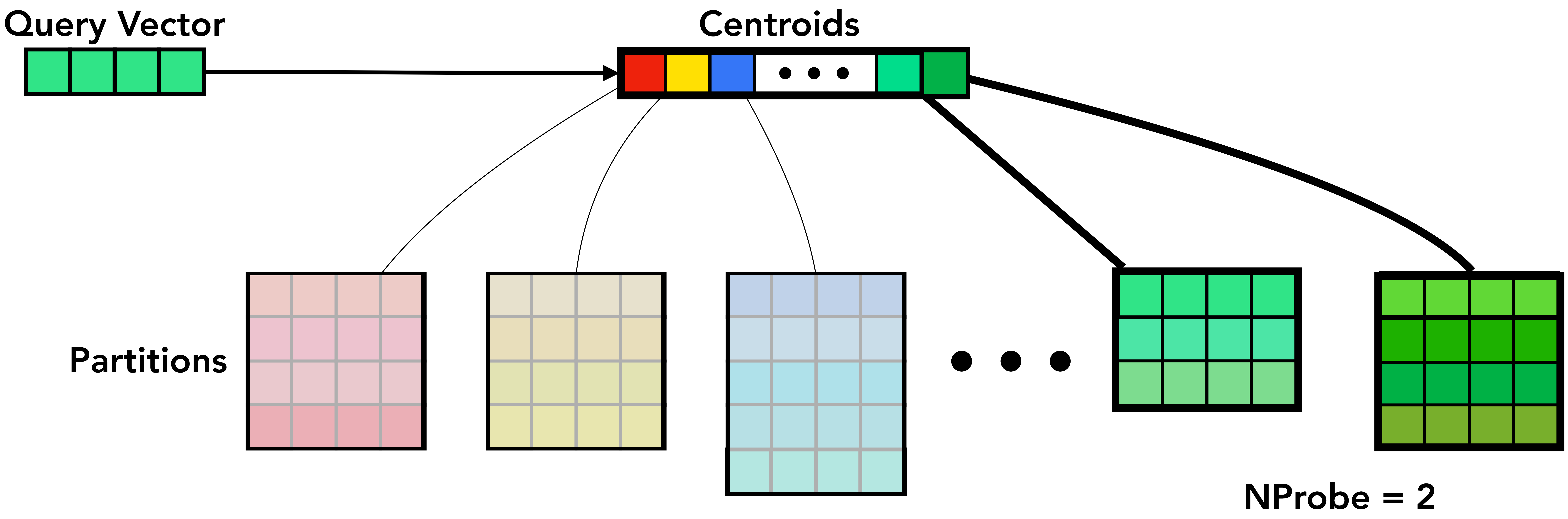
Partitioned Index: Maintenance

Splitting and merging resolves imbalance



Partitioned Index: Maintenance

Splitting and merging resolves imbalance



NProbe needs to be tuned as the number of partitions changes

Challenge: Handling Updates

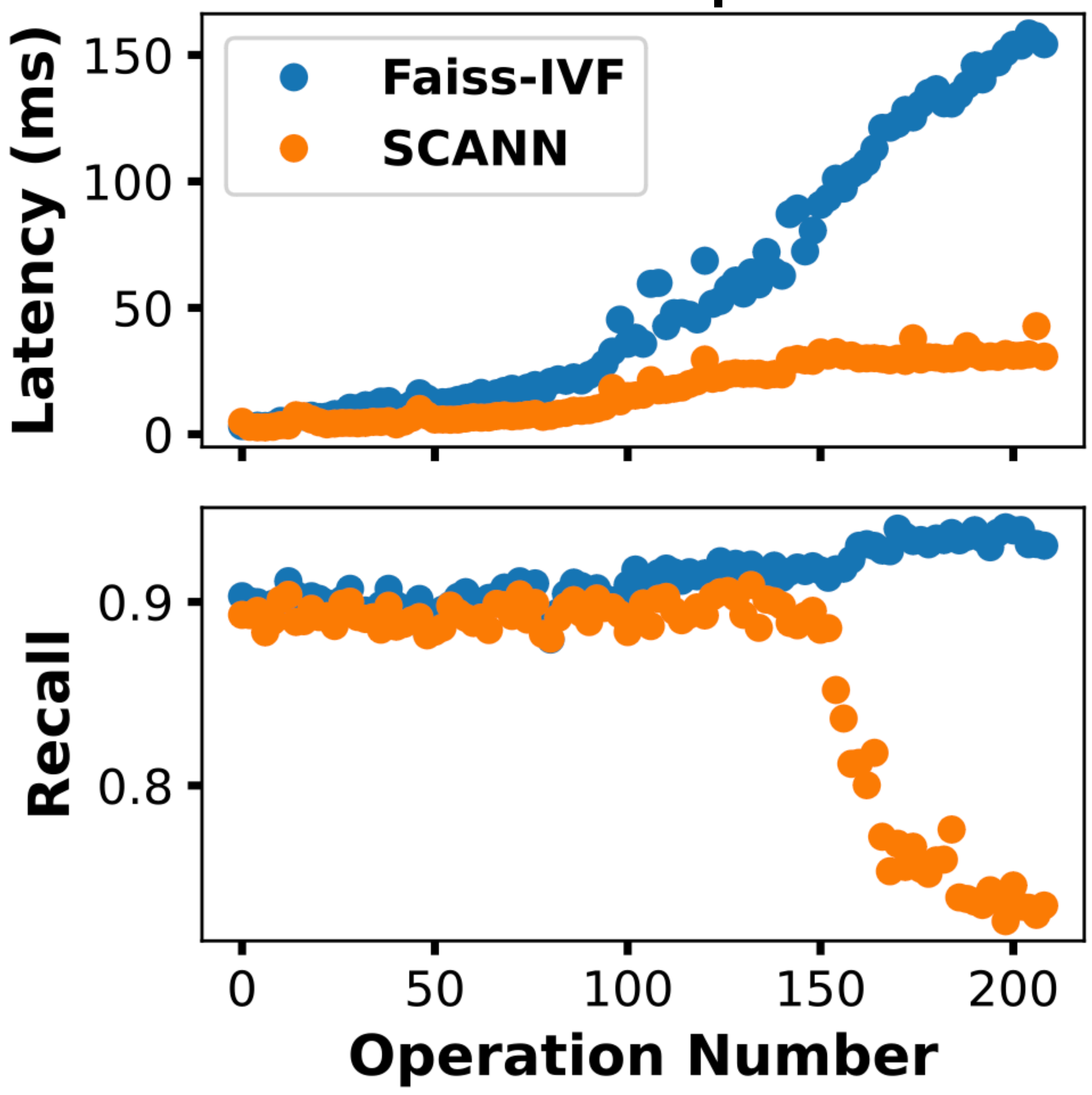
Updates **degrade the latency and recall** of partitioned vector search indexes

Why?

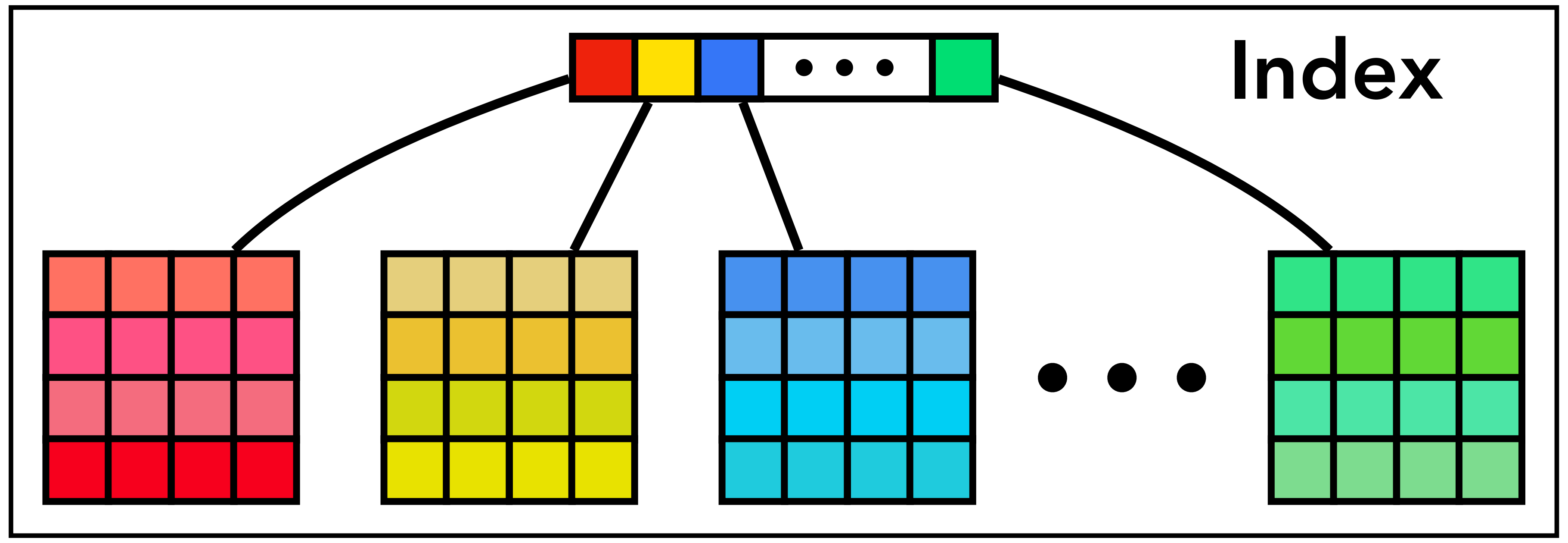
1. Partitions grow and become imbalanced. E.g. Faiss-IVF

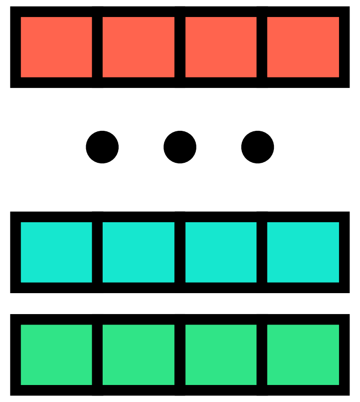
2. Applying maintenance (split/merge) requires retuning the system to maintain recall after maintenance. E.g. SCANN

Workload: Wikipedia-12M

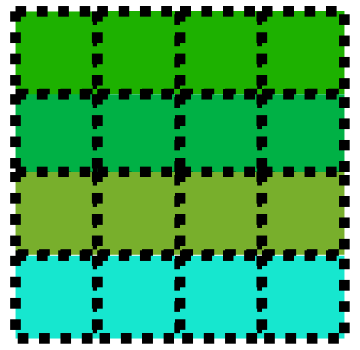


Quake: An Adaptive Approach





Searches

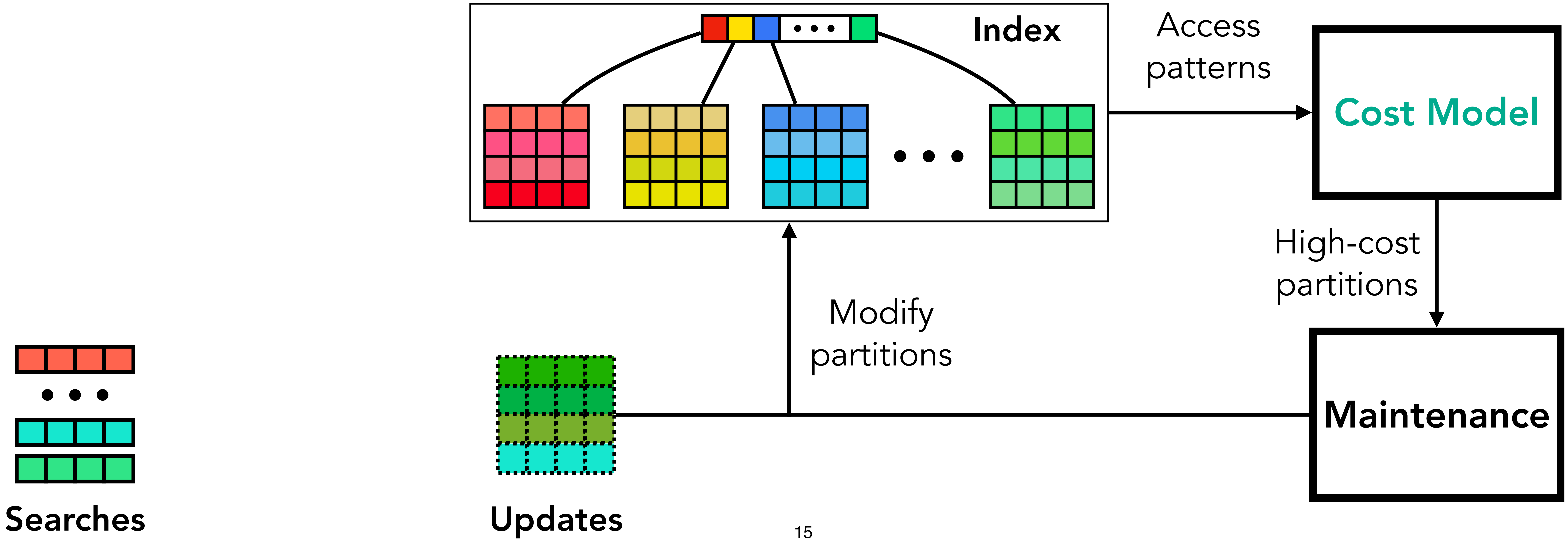


Updates

Quake: An Adaptive Approach

Challenge: Index imbalance and workload skew

Contribution: Skew-aware cost-driven maintenance



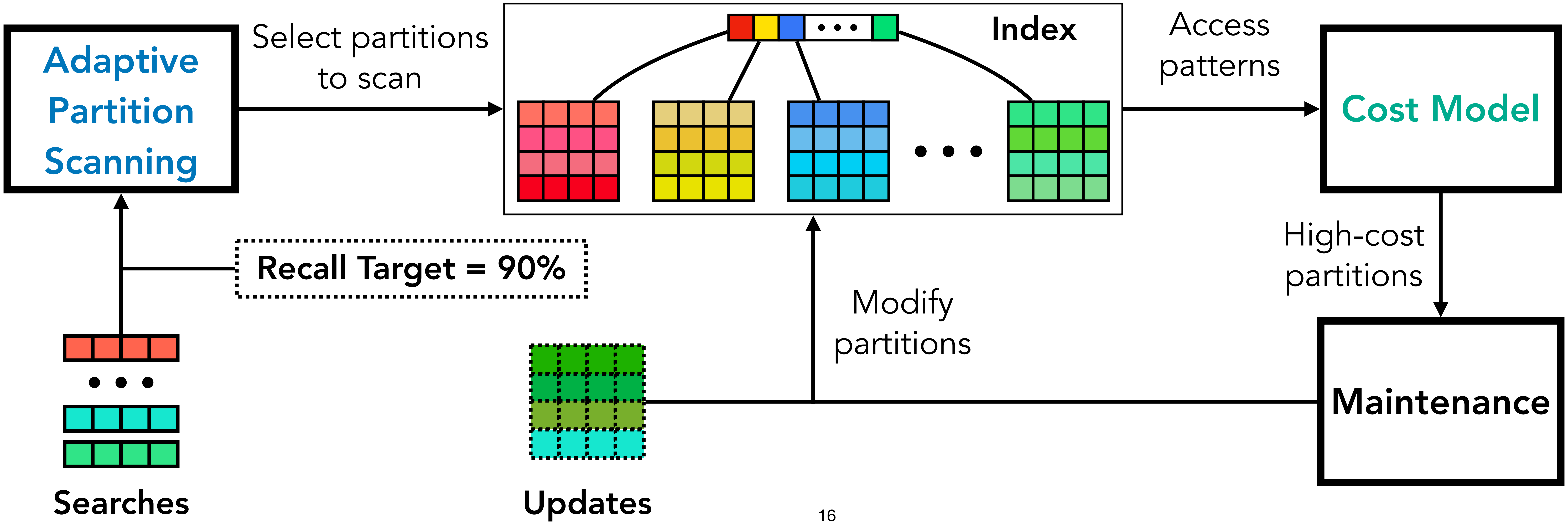
Quake: An Adaptive Approach

Challenge: Recall degradation due to maintenance

Contribution: Adaptive setting of NProbe

Challenge: Index imbalance and workload skew

Contribution: Skew-aware cost-driven maintenance



Quake: An Adaptive Approach

Challenge: Recall degradation due to maintenance

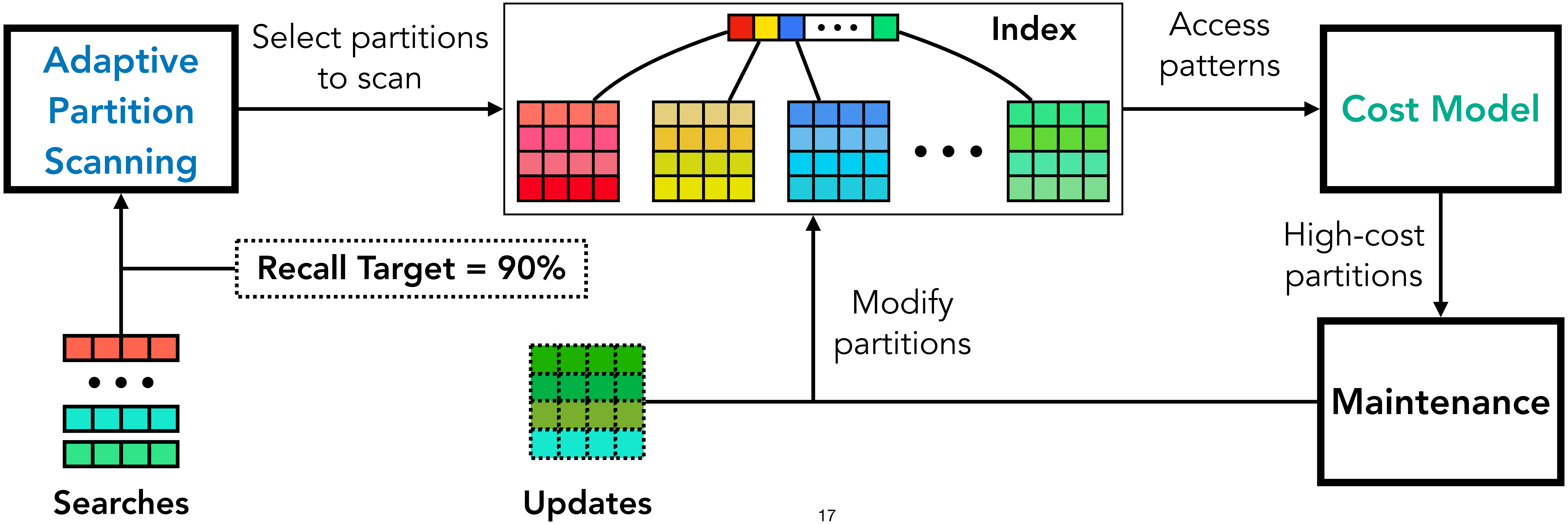
Contribution: Adaptive setting of NProbe

Challenge: Index imbalance and workload skew

Contribution: Skew-aware cost-driven maintenance

Result: 1.25x-28x lower query latency than DiskANN, SCANN without costly tuning of NProbe

Open-source: github.com/marius-team/quake



Cost-Driven Maintenance

Use a cost-model for query latency to determine which partitions to split/merge

$$C = \sum_i C_i$$ Total cost (query latency)

$$C_i = O_c + A_i \lambda(s_i)$$ Per-partition cost

A_i Access frequency of partition i (collected online)

$\lambda(s_i)$ Latency to scan partition with size s_i (obtain offline via profiling)

O_c Latency to scan centroid

Cost-Driven Maintenance

Use a cost-model for query latency to determine which partitions to split/merge

$C = \sum_i C_i$ Total cost (query latency)

$C_i = O_c + A_i \lambda(s_i)$ Per-partition cost

A_i Access frequency of partition i (collected online)

$\lambda(s_i)$ Latency to scan partition with size s_i (obtain offline via profiling)

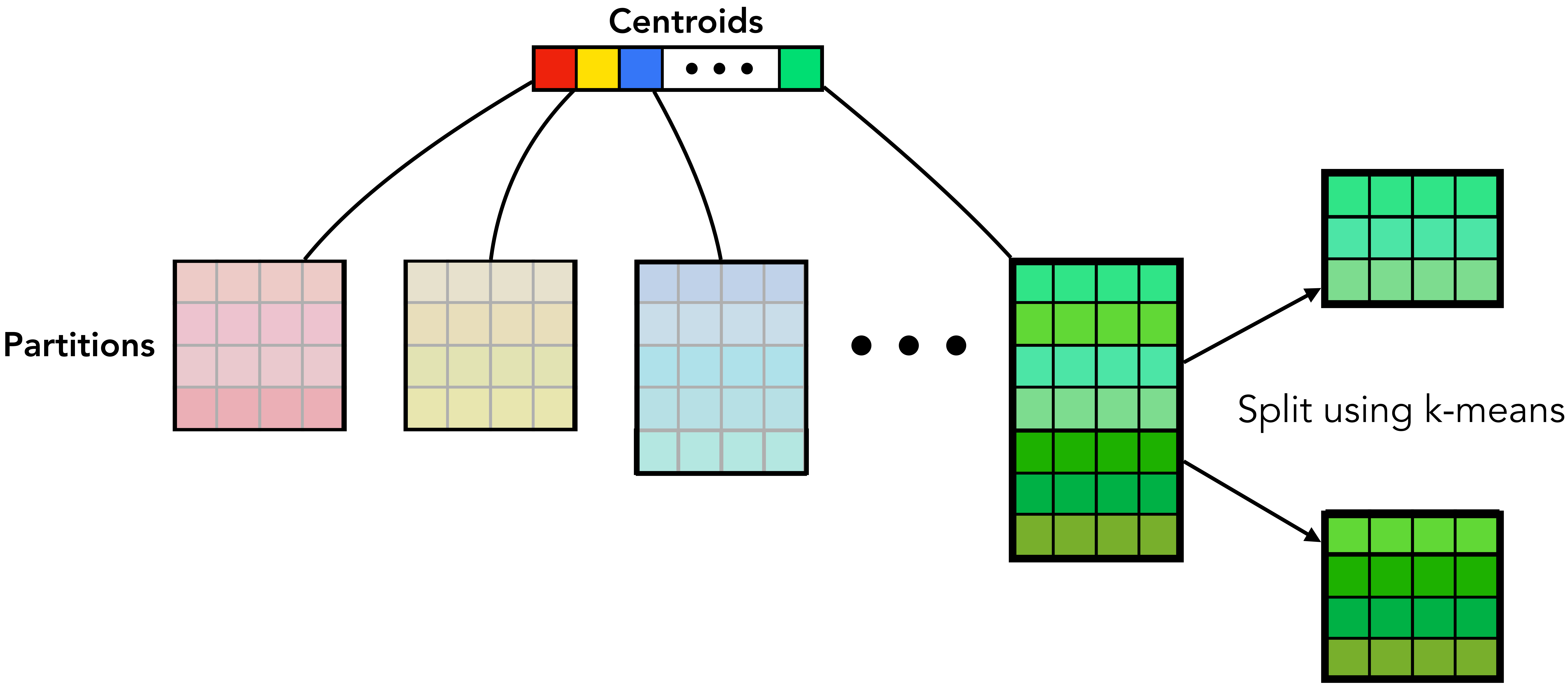
O_c Latency to scan centroid

Estimate the change in cost of a split/merge, take action if expected to reduce cost

$\Delta C_i = C_i + \Delta_{split_i}$ If $\Delta C_i < \tau$ then split partition i

Minimize the cost through maintenance

Cost-Driven Maintenance: Example



Cost-Driven Maintenance: Example

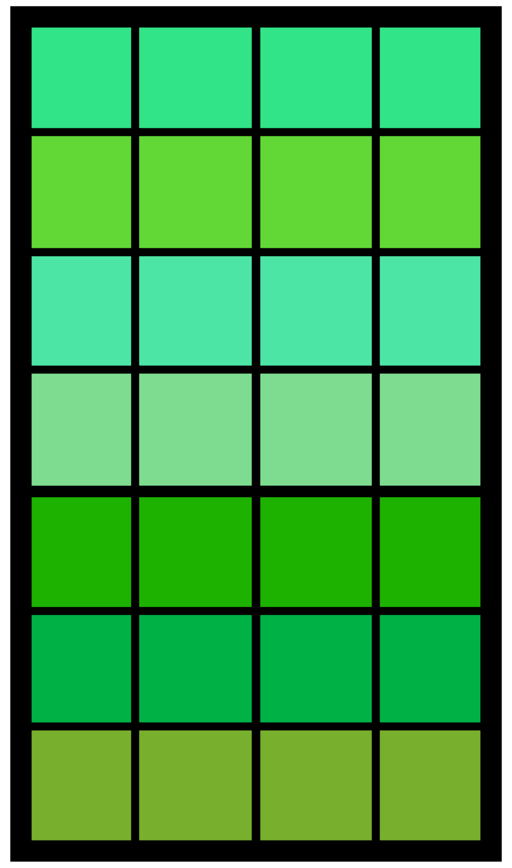
Cost is determined by partition access frequency, size and centroid overhead

Before Split

$O_c = .1\mu s$

Latency to scan centroid

Centroid



$A_i = .1$

$\lambda(s_i) = 20\mu s$

Cost-Driven Maintenance: Example

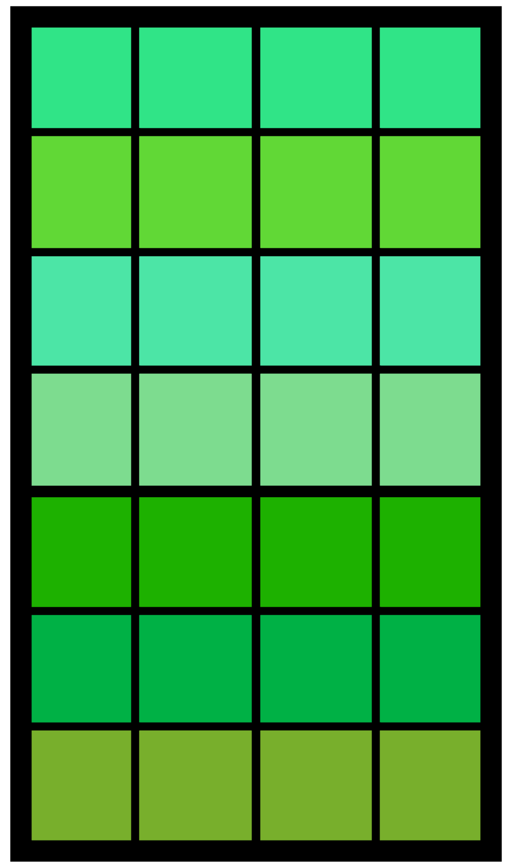
Cost is determined by partition access frequency, size and centroid overhead

Before Split

$O_c = .1\mu s$

Latency to scan centroid

Centroid



$A_i = .1$

$\lambda(s_i) = 20\mu s$

$$\boxed{C_i = O_c + A_i\lambda(s_i) = 2.1\mu s}$$

Cost-Driven Maintenance: Example

Splitting reduces cost since not all queries will scan each partition

Before Split

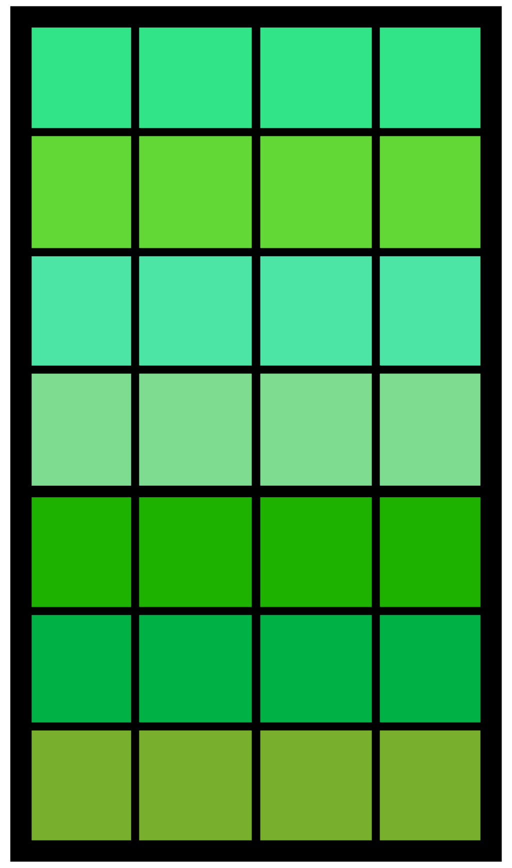
$O_c = .1\mu s$

Latency to scan centroid

Centroid

$A_i = .1$

$\lambda(s_i) = 20\mu s$



$\boxed{C_i = O_c + A_i \lambda(s_i) = 2.1\mu s}$

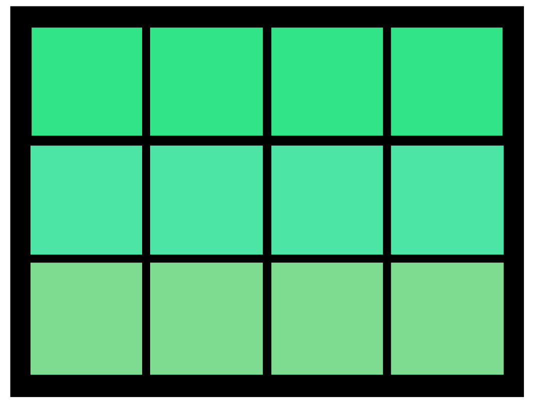
After Split

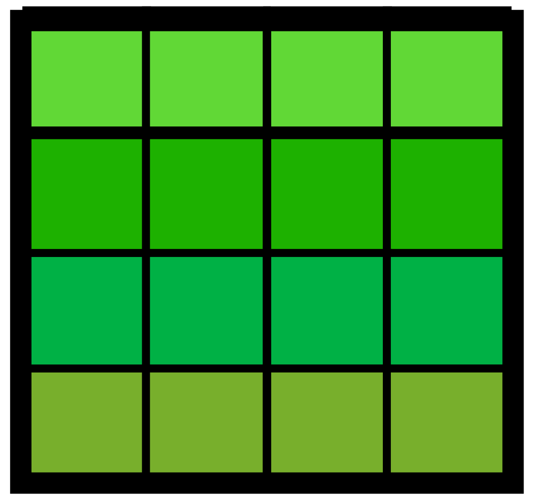
Centroid

Centroid

$A_L = .05$

$\lambda(s_L) = 8\mu s$





$A_R = .08$

$\lambda(s_R) = 12\mu s$

Cost-Driven Maintenance: Example

Splitting reduces cost since not all queries will scan each partition

Before Split

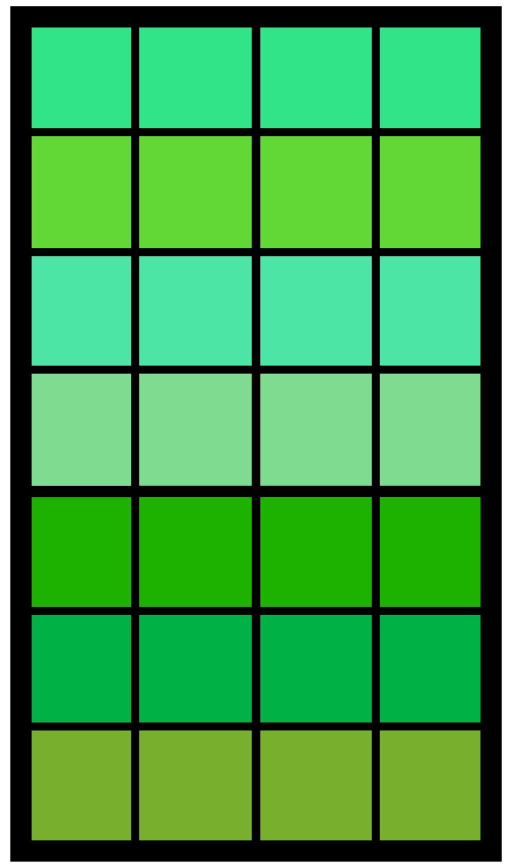
$O_c = .1\mu s$

Centroid

Latency to scan centroid

$A_i = .1$

$\lambda(s_i) = 20\mu s$



$$C_i = O_c + A_i \lambda(s_i) = 2.1\mu s$$

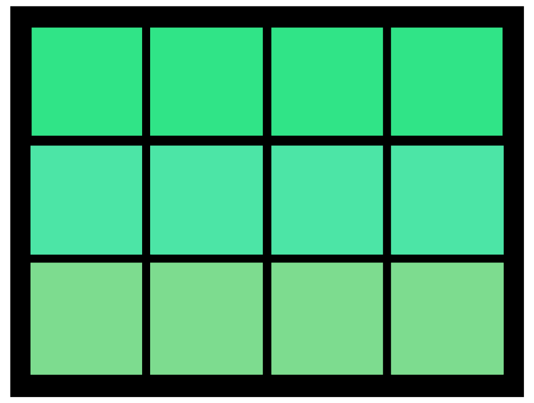
After Split

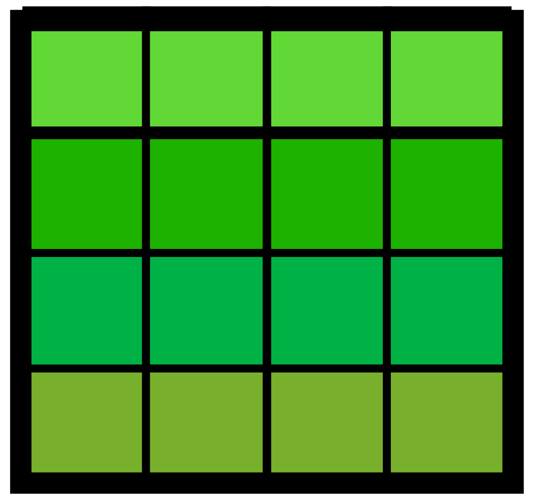
Centroid

Centroid

$A_L = .05$

$\lambda(s_L) = 8\mu s$





$A_R = .08$

$\lambda(s_R) = 12\mu s$

$C_L = .5\mu s$

$C_R = 1.06\mu s$

$$C_{split} = 1.56\mu s$$

Cost-Driven Maintenance: Example

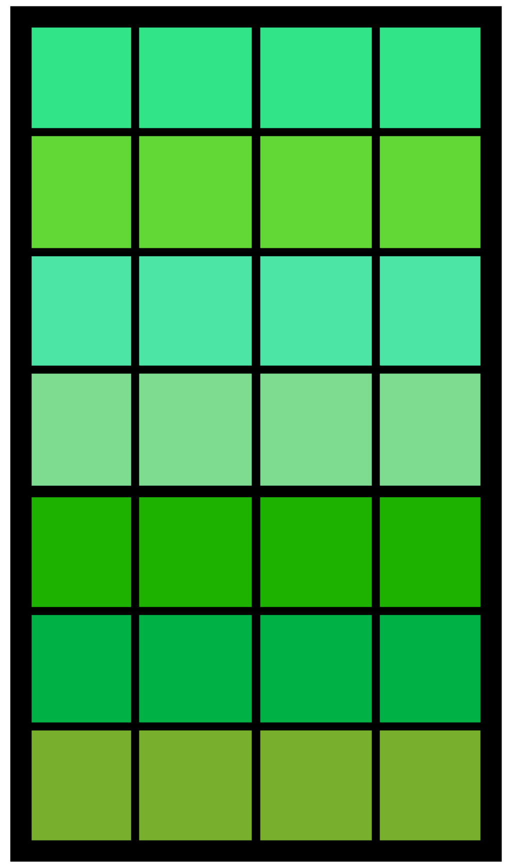
The catch: The access patterns after splitting are unknown so we must estimate them

Before Split

$O_c = .1\mu s$

Centroid

Latency to scan centroid



$A_i = .1$

$\lambda(s_i) = 20\mu s$

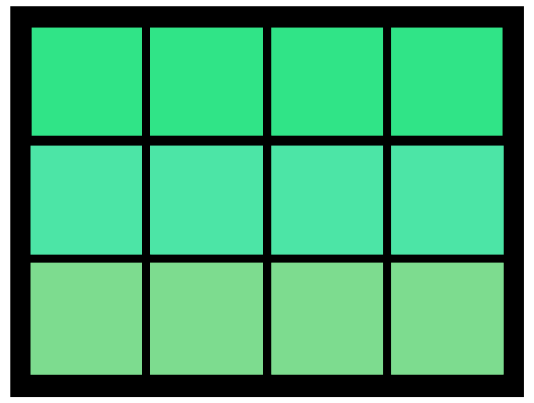
$$\boxed{C_i = O_c + A_i \lambda(s_i) = 2.1\mu s}$$

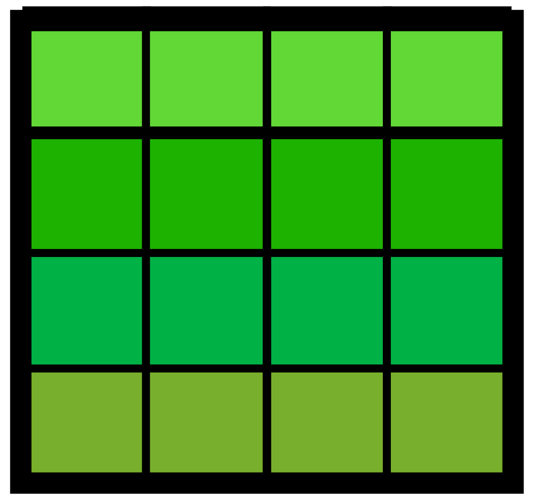
After Split

Centroid

Centroid

$\lambda(s_L) = 8\mu s$





$\lambda(s_R) = 12\mu s$

$$\boxed{A_L = ???, A_R = ???}$$

Cost-Driven Maintenance: Example

The catch: The access patterns after splitting are unknown so we must estimate them

Before Split

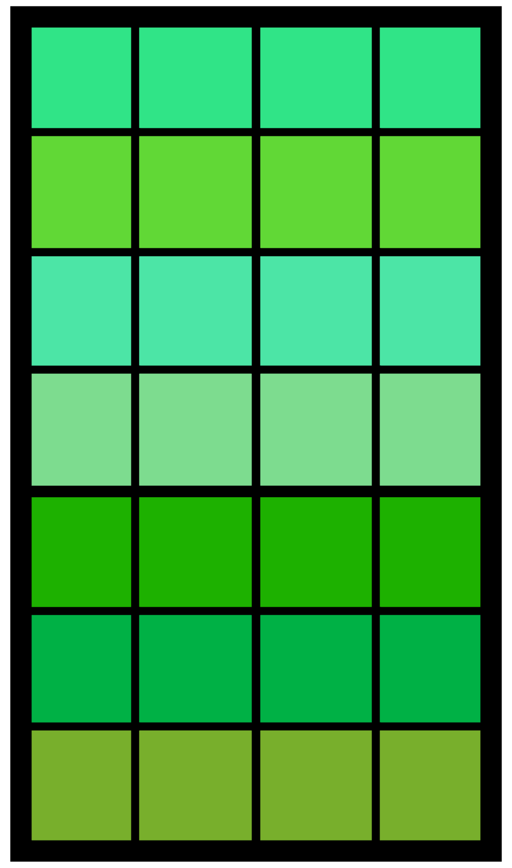
$O_c = .1\mu s$

Centroid

Latency to scan centroid

$A_i = .1$

$\lambda(s_i) = 20\mu s$



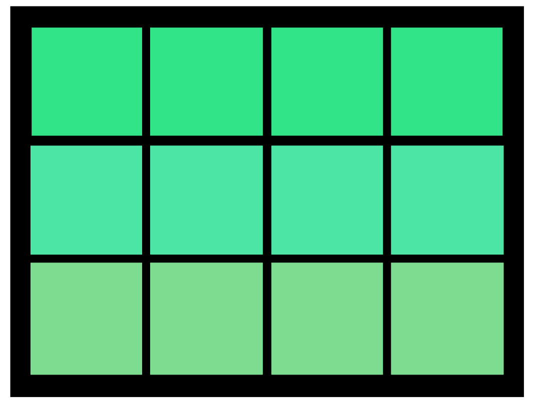
$$\boxed{C_i = O_c + A_i\lambda(s_i) = 2.1\mu s}$$

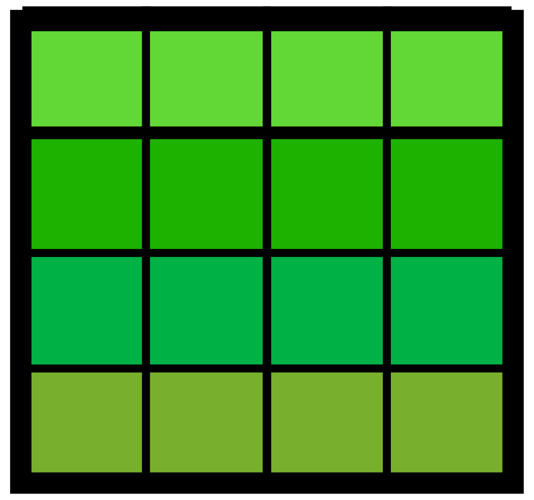
After Split

Centroid

Centroid

$\lambda(s_L) = 8\mu s$





$\lambda(s_R) = 12\mu s$

$$\boxed{A_L = A_R = \alpha A_i}$$

$\alpha :=$ Expected decrease in access frequency

Cost-Driven Maintenance: Example

Conduct split if estimated cost is lower than prior cost

Before Split

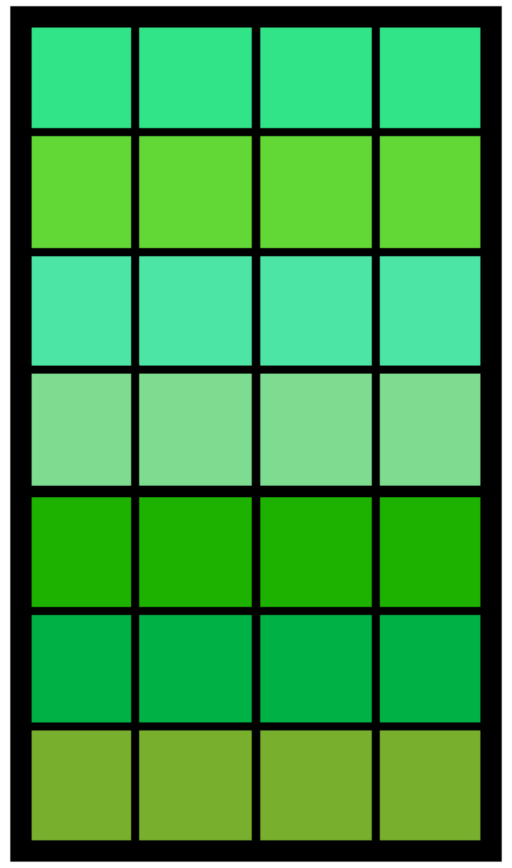
$O_c = .1\mu s$

Centroid

Latency to scan centroid

$A_i = .1$

$\lambda(s_i) = 20\mu s$



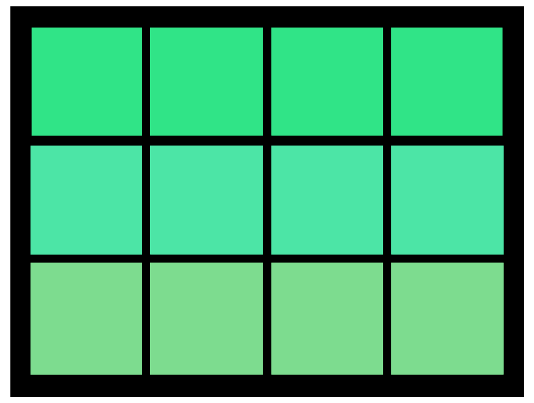
$$\boxed{C_i = O_c + A_i\lambda(s_i) = 2.1\mu s}$$

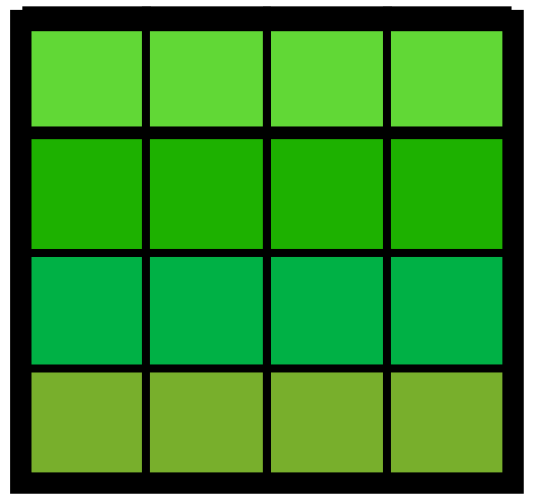
After Split

Centroid

Centroid

$\lambda(s_L) = 8\mu s$





$\lambda(s_R) = 12\mu s$

$A_L = A_R = \alpha A_i$

$\alpha :=$ Expected decrease in access frequency

$$C'_{split} = 2O_c + \alpha A_i(\lambda(s_L) + \lambda(s_R))$$

$$\alpha = .8 \qquad C'_{split} = 1.8\mu s$$

Adaptive Partition Scanning (APS)

Determines the number of partitions to scan (NProbe) per query **online**

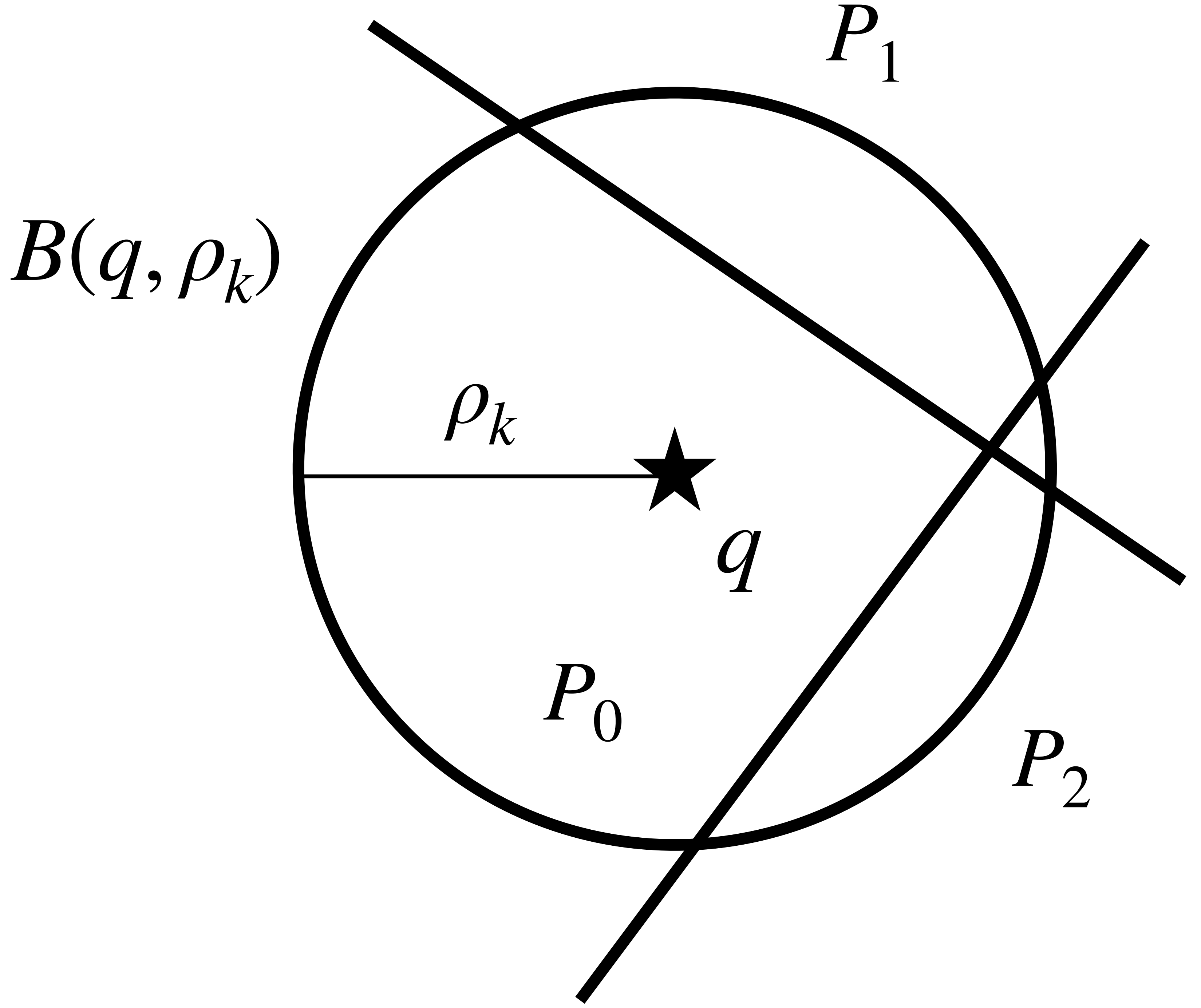
Geometric Model

Estimate the volume of overlap between the *query hypersphere* $B(q, \rho_k)$ and neighboring partitions/clusters.

ρ_k = Distance to k-th nearest neighbor

The amount of overlap is proportional to the amount of recall obtained from scanning the cluster

$$\boxed{Recall_i = \frac{Vol(B(q, \rho_k) \cap P_i)}{Vol(B(q, \rho_k))}}$$



Query hypersphere overlapping with partitions: P_0, P_1, P_2

Evaluation

First: Comparison of indexes on Wikipedia-12M workload

Second: Comparison of Adaptive Partition Scanning with tuning-based methods

More in paper!

- NUMA-aware query processing and parallelism

- Multi-level indexing

- Batch processing

- Ablation studies

- Three additional workloads

End-to-End Comparison: Wikipedia-12M

Workload: Wikipedia-12M:
- Index grows from 1 million to 12 million vectors
- 128 dimensional vectors, k=100, recall = 90%
- Queries sampled from wikipedia page access counts
- Queries executed one-by-one

Total Search, Update, and Maintenance Time (hours)

Method	Type	Search	Update	Maint.

End-to-End Comparison: Wikipedia-12M

Workload: Wikipedia-12M:

- Index grows from 1 million to 12 million vectors
- 128 dimensional vectors, k=100, recall = 90%
- Queries sampled from wikipedia page access counts
- Queries executed one-by-one

Total Search, Update, and Maintenance Time (hours)

Method	Type	Search	Update	Maint.
Quake (parallel)	Partitioned	**1.53**	0.01	0.44
Quake (single thread)	Partitioned	9.48	0.01	0.44
Faiss-IVF	Partitioned	165.8	0.005	0
SCANN	Partitioned	50.2	1.75	
DiskANN	Graph	12.1	0.32	0

Baselines: Do not support intra-query parallelism for queries

Hardware: 4 NUMA nodes with 20 CPUs each.

End-to-End Comparison: Wikipedia-12M

Workload: Wikipedia-12M:

- Index grows from 1 million to 12 million vectors
- 128 dimensional vectors, k=100, recall = 90%
- Queries sampled from wikipedia page access counts
- Queries executed one-by-one

Total Search, Update, and Maintenance Time (hours)

Method	Type	Search	Update	Maint.
Quake (parallel)	Partitioned	**1.53**	0.01	0.44
Quake (single thread)	Partitioned	9.48	0.01	0.44
Faiss-IVF	Partitioned	165.8	0.005	0
SCANN	Partitioned	50.2	1.75	
DiskANN	Graph	12.1	0.32	0

Takeaways:

1. Quake: **28x** (parallel) and **5x** (single-thread) lower search latency than partitioned indexes

Baselines: Do not support intra-query parallelism for queries

Hardware: 4 NUMA nodes with 20 CPUs each.

End-to-End Comparison: Wikipedia-12M

Workload: Wikipedia-12M:

- Index grows from 1 million to 12 million vectors
- 128 dimensional vectors, k=100, recall = 90%
- Queries sampled from wikipedia page access counts
- Queries executed one-by-one

Takeaways:

1. Quake: **28x** (parallel) and **5x** (single-thread) lower search latency than partitioned indexes
2. Quake achieves **8x** (parallel) and **1.25x** (single-thread) lower search latency than DiskANN

Baselines: Do not support intra-query parallelism for queries

Total Search, Update, and Maintenance Time (hours)

Method	Type	Search	Update	Maint.
Quake (parallel)	Partitioned	**1.53**	0.01	0.44
Quake (single thread)	Partitioned	9.48	0.01	0.44
Faiss-IVF	Partitioned	165.8	0.005	0
SCANN	Partitioned	50.2	1.75	
DiskANN	Graph	12.1	0.32	0

Hardware: 4 NUMA nodes with 20 CPUs each.

Comparison with tuning-based methods

Dataset: Sift1M, 1M 128-d vectors, 10,000 queries

Index: 1,000 partitions

LAET ***(****Sigmod '20****)*** is a machine-learning method for setting NProbe and is trained offline

Oracle sets the optimal NProbe per query

Target = 80%

Method	Recall	Latency	Tuning Time
APS			
LAET			
Oracle			

Target = 90%

APS			
LAET			
Oracle			

Target = 99%

APS			
LAET			
Oracle			

Comparison with tuning-based methods

Dataset: Sift1M, 1M 128-d vectors, 10,000 queries

Index: 1,000 partitions

***LAET** (Sigmod '20)* is a machine-learning method for setting NProbe and is trained offline

Oracle sets the optimal NProbe per query

1. ***Accurate: APS*** achieves the recall targets

Target = 80%

Method	Recall	Latency	Tuning Time
APS	82.1%		
LAET	81.3%		
Oracle	83.3%		

Target = 90%

Method	Recall	Latency	Tuning Time
APS	91.2%		
LAET	90.5%		
Oracle	92.4%		

Target = 99%

Method	Recall	Latency	Tuning Time
APS	98.9%		
LAET	99.0%		
Oracle	99.2%		

Comparison with tuning-based methods

Dataset: Sift1M, 1M 128-d vectors, 10,000 queries

Index: 1,000 partitions

LAET *(Sigmod '20)* is a machine-learning method for setting NProbe and is trained offline

Oracle sets the optimal NProbe per query

1. ***Accurate:*** ***APS*** achieves the recall targets
2. **Low overhead:** 10-30% increase in latency over **Oracle**

Target = 80%

Method	Recall	Latency	Tuning Time
APS	82.1%	.34ms	
LAET	81.3%	.29ms	
Oracle	83.3%	.29ms	

Target = 90%

Method	Recall	Latency	Tuning Time
APS	91.2%	.48ms	
LAET	90.5%	.42ms	
Oracle	92.4%	.41ms	

Target = 99%

Method	Recall	Latency	Tuning Time
APS	98.9%	.96ms	
LAET	99.0%	1.04ms	
Oracle	99.2%	.74ms	

Comparison with tuning-based methods

Dataset: Sift1M, 1M 128-d vectors, 10,000 queries

Index: 1,000 partitions

***LAET** (Sigmod '20)* is a machine-learning method for setting NProbe and is trained offline

Oracle sets the optimal NProbe per query

1. ***Accurate: APS*** achieves the recall targets
2. **Low overhead:** 10-30% increase in latency over **Oracle**
3. **No offline tuning**

Target = 80%

Method	Recall	Latency	Tuning Time
APS	82.1%	.34ms	0
LAET	81.3%	.29ms	81s
Oracle	83.3%	.29ms	320s

Target = 90%

Method	Recall	Latency	Tuning Time
APS	91.2%	.48ms	0
LAET	90.5%	.42ms	104s
Oracle	92.4%	.41ms	331s

Target = 99%

Method	Recall	Latency	Tuning Time
APS	98.9%	.96ms	0
LAET	99.0%	1.04ms	232s
Oracle	99.2%	.74ms	368s

QUAKE

Adaptive Indexing for Vector Search

Jason Mohoney, Devesh Sarda, Mengze Tang, Anil Pacaci, Shihab Chowdhury, Ihab Ilyas, Theodoros Rekatsinas, Shivaram Venkataraman

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

Challenge: Recall degradation due to maintenance

Contribution: Adaptive setting of NProbe

Challenge: Index imbalance and workload skew

Contribution: Skew-aware cost-driven maintenance

Result: 1.25x-28x lower query latency than DiskANN, SCANN without costly tuning of NProbe

Open-source: github.com/marius-team/quake

Contact: jasonmohoney@gmail.com