Learning Multi-dimensional Indexes

Vikram Nathan*, **Jialin Ding***, Mohammad Alizadeh, Tim Kraska

CSAIL

ML for Systems

ML for Systems

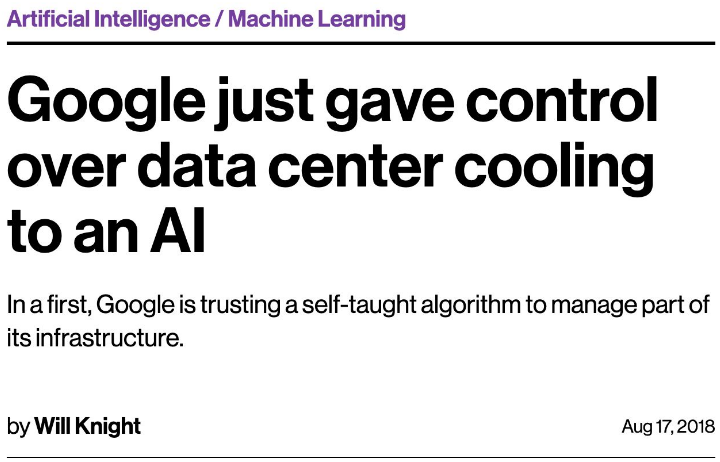
- Why ML?
  - Systems rely on heuristics and hand-tuning
  - Systems don’t adapt to specific data/workload

ML for Systems

- Why ML?
  - Systems rely on heuristics and hand-tuning
  - Systems don’t adapt to specific data/workload
- Examples
  - Query optimization
  - Job scheduling
  - Indexing
  - Sorting

ML for Systems

- Why ML?
  - Systems rely on heuristics and hand-tuning
  - Systems don’t adapt to specific data/workload
- Examples
  - Query optimization
  - Job scheduling
  - Indexing
  - Sorting


Artificial Intelligence / Machine Learning

Google just gave control over data center cooling to an AI

In a first, Google is trusting a self-taught algorithm to manage part of its infrastructure.

by **Will Knight** Aug 17, 2018

ML for Systems

- Why ML?
  - Systems rely on heuristics and hand-tuning
  - Systems don't adapt to specific data/workload
- Examples
  - Query optimization
  - Job scheduling
  - Indexing
  - Sorting
- Differences with "mainstream" ML:
  - Objectives beyond accuracy (e.g., latency, space usage, cost)
  - Want 10X, not 10%

Artificial Intelligence / Machine Learning

Google just gave control over data center cooling to an AI

In a first, Google is trusting a self-taught algorithm to manage part of its infrastructure.

by **Will Knight** Aug 17, 2018

ML for Systems

- Why ML?
  - Systems rely on heuristics and hand-tuning
  - Systems don’t adapt to specific data/workload
- Examples
  - Query optimization
  - Job scheduling
  - Indexing
  - Sorting
- Differences with “mainstream” ML:
  - Objectives beyond accuracy (e.g., latency, space usage, cost)
  - Want 10X, not 10%
  - Implication: favor creative uses of simple models

Artificial Intelligence / Machine Learning

Google just gave control over data center cooling to an AI

In a first, Google is trusting a self-taught algorithm to manage part of its infrastructure.

by **Will Knight** Aug 17, 2018

Outline

1. **Completed work: Flood (SIGMOD 2020)**
2. Future work: column correlations, query skew, categorical attributes

Motivation: scanning and filtering in analytics

Motivation: scanning and filtering in analytics

Motivation: scanning and filtering in analytics

<u>Order ID</u>	**Item ID**	**Ship Date**	**Receipt Date**	**Price**	**Quantity**	**Discount**	**Tax**
1	42	9/14/19	9/16/19	2	1	0	5%
2	137	9/18/19	9/25/19	5	1	0	6.5%
3	314	10/3/19	10/6/19	14	2	0	5.5%
4	272	9/30/19	10/15/19	52	1	0.1	8%
5	162	10/2/19	10/4/19	13	1	0	6.5%
6	602	10/5/19	10/10/19	7	5	0.5	8.5%
7	141	10/4/19	10/9/19	150	1	0.1	7%
8	173	10/8/19	10/12/19	20	2	0	8%

Motivation: scanning and filtering in analytics

Order ID	**Item ID**	**Ship Date**	**Receipt Date**	**Price**	**Quantity**	**Discount**	**Tax**
1	42	9/14/19	9/16/19	2	1	0	5%
2	137	9/18/19	9/25/19	5	1	0	6.5%
3	314	10/3/19	10/6/19	14	2	0	5.5%
4	272	9/30/19	10/15/19	52	1	0.1	8%
5	162	10/2/19	10/4/19	13	1	0	6.5%
6	602	10/5/19	10/10/19	7	5	0.5	8.5%
7	141	10/4/19	10/9/19	150	1	0.1	7%
8	173	10/8/19	10/12/19	20	2	0	8%

Query

```
SELECT COUNT(*)
FROM table
WHERE Price >= 10
      AND Price < 100
```

Motivation: scanning and filtering in analytics

Order ID	**Item ID**	**Ship Date**	**Receipt Date**	**Price**	**Quantity**	**Discount**	**Tax**
1	42	9/14/19	9/16/19	2	1	0	5%
2	137	9/18/19	9/25/19	5	1	0	6.5%
3	314	10/3/19	10/6/19	14	2	0	5.5%
4	272	9/30/19	10/15/19	52	1	0.1	8%
5	162	10/2/19	10/4/19	13	1	0	6.5%
6	602	10/5/19	10/10/19	7	5	0.5	8.5%
7	141	10/4/19	10/9/19	150	1	0.1	7%
8	173	10/8/19	10/12/19	20	2	0	8%

Query

```
SELECT COUNT(*)
FROM table
WHERE Price >= 10
      AND Price < 100
```

Motivation: scanning and filtering in analytics

Order ID	**Item ID**	**Ship Date**	**Receipt Date**	**Price**	**Quantity**	**Discount**	**Tax**
1	42	9/14/19	9/16/19	2	1	0	5%
2	137	9/18/19	9/25/19	5	1	0	6.5%
3	314	10/3/19	10/6/19	14	2	0	5.5%
4	272	9/30/19	10/15/19	52	1	0.1	8%
5	162	10/2/19	10/4/19	13	1	0	6.5%
6	602	10/5/19	10/10/19	7	5	0.5	8.5%
7	141	10/4/19	10/9/19	150	1	0.1	7%
8	173	10/8/19	10/12/19	20	2	0	8%

Query

```
SELECT COUNT(*)
FROM table
WHERE Price >= 10
      AND Price < 100
```

Motivation: scanning and filtering in analytics

Order ID	**Item ID**	**Ship Date**	**Receipt Date**	**Price**	**Quantity**	**Discount**	**Tax**
1	42	9/14/19	9/16/19	2	1	0	5%
2	137	9/18/19	9/25/19	5	1	0	6.5%
3	314	10/3/19	10/6/19	14	2	0	5.5%
4	272	9/30/19	10/15/19	52	1	0.1	8%
5	162	10/2/19	10/4/19	13	1	0	6.5%
6	602	10/5/19	10/10/19	7	5	0.5	8.5%
7	141	10/4/19	10/9/19	150	1	0.1	7%
8	173	10/8/19	10/12/19	20	2	0	8%

Query

```
SELECT COUNT(*)
FROM table
WHERE Price >= 10
      AND Price < 100
```

Scan overhead: ratio of records scanned to number of filtered results

Motivation: scanning and filtering in analytics

Order ID	**Item ID**	**Ship Date**	**Receipt Date**	**Price**	**Quantity**	**Discount**	**Tax**
1	42	9/14/19	9/16/19	2	1	0	5%
2	137	9/18/19	9/25/19	5	1	0	6.5%
3	314	10/3/19	10/6/19	14	2	0	5.5%
4	272	9/30/19	10/15/19	52	1	0.1	8%
5	162	10/2/19	10/4/19	13	1	0	6.5%
6	602	10/5/19	10/10/19	7	5	0.5	8.5%
7	141	10/4/19	10/9/19	150	1	0.1	7%
8	173	10/8/19	10/12/19	20	2	0	8%

Query	**Scan Overhead**
SELECT COUNT(*) FROM table WHERE Price >= 10 AND Price < 100	2

Scan overhead: ratio of records scanned to number of filtered results

Motivation: scanning and filtering in analytics

Order ID	**Item ID**	**Ship Date**	**Receipt Date**	**Price**	**Quantity**	**Discount**	**Tax**
1	42	9/14/19	9/16/19	2	1	0	5%
2	137	9/18/19	9/25/19	5	1	0	6.5%
3	314	10/3/19	10/6/19	14	2	0	5.5%
4	272	9/30/19	10/15/19	52	1	0.1	8%
5	162	10/2/19	10/4/19	13	1	0	6.5%
6	602	10/5/19	10/10/19	7	5	0.5	8.5%
7	141	10/4/19	10/9/19	150	1	0.1	7%
8	173	10/8/19	10/12/19	20	2	0	8%

Query	**Scan Overhead**
SELECT COUNT(*) FROM table WHERE Price >= 10 AND Price < 100	2

Scan overhead: ratio of records scanned to number of filtered results

Lower scan overhead generally leads to lower query time

Single-dimensional indexes

Order ID	**Item ID**	**Ship Date**	**Receipt Date**	***Price***	**Quantity**	**Discount**	**Tax**
1	42	9/14/19	9/16/19	2	1	0	5%
2	137	9/18/19	9/25/19	5	1	0	6.5%
6	602	10/5/19	10/10/19	7	5	0.5	8.5%
5	162	10/2/19	10/4/19	13	1	0	6.5%
3	314	10/3/19	10/6/19	14	2	0	5.5%
8	173	10/8/19	10/12/19	20	2	0	8%
4	272	9/30/19	10/15/19	52	1	0.1	8%
7	141	10/4/19	10/9/19	150	1	0.1	7%

Query

```
SELECT COUNT(*)
FROM table
WHERE Price >= 10
      AND Price < 100
```

Single-dimensional indexes

Order ID	Item ID	Ship Date	Receipt Date	***Price***	Quantity	Discount	Tax
1	42	9/14/19	9/16/19	2	1	0	5%
2	137	9/18/19	9/25/19	5	1	0	6.5%
6	602	10/5/19	10/10/19	7	5	0.5	8.5%
5	162	10/2/19	10/4/19	13	1	0	6.5%
3	314	10/3/19	10/6/19	14	2	0	5.5%
8	173	10/8/19	10/12/19	20	2	0	8%
4	272	9/30/19	10/15/19	52	1	0.1	8%
7	141	10/4/19	10/9/19	150	1	0.1	7%

Query	**Scan Overhead**
SELECT COUNT(*) FROM table WHERE Price >= 10 AND Price < 100	1

Single-dimensional indexes

Order ID	Item ID	Ship Date	Receipt Date	***Price***	Quantity	Discount	Tax
1	42	9/14/19	9/16/19	2	1	0	5%
2	137	9/18/19	9/25/19	5	1	0	6.5%
6	602	10/5/19	10/10/19	7	5	0.5	8.5%
5	162	10/2/19	10/4/19	13	1	0	6.5%
3	314	10/3/19	10/6/19	14	2	0	5.5%
8	173	10/8/19	10/12/19	20	2	0	8%
4	272	9/30/19	10/15/19	52	1	0.1	8%
7	141	10/4/19	10/9/19	150	1	0.1	7%

Query	**Scan Overhead**
SELECT COUNT(*) FROM table WHERE Price >= 10 AND Price < 100	1
SELECT COUNT(*) FROM table WHERE Price >= 10 AND Price < 100 AND Quantity = 1	

Single-dimensional indexes

Order ID	Item ID	Ship Date	Receipt Date	***Price***	Quantity	Discount	Tax
1	42	9/14/19	9/16/19	2	1	0	5%
2	137	9/18/19	9/25/19	5	1	0	6.5%
6	602	10/5/19	10/10/19	7	5	0.5	8.5%
5	162	10/2/19	10/4/19	13	1	0	6.5%
3	314	10/3/19	10/6/19	14	2	0	5.5%
8	173	10/8/19	10/12/19	20	2	0	8%
4	272	9/30/19	10/15/19	52	1	0.1	8%
7	141	10/4/19	10/9/19	150	1	0.1	7%

Query	**Scan Overhead**
SELECT COUNT(*) FROM table WHERE Price >= 10 AND Price < 100	1
SELECT COUNT(*) FROM table WHERE Price >= 10 AND Price < 100 AND Quantity = 1	2

Single-dimensional indexes

Order ID	Item ID	Ship Date	Receipt Date	***Price***	Quantity	Discount	Tax
1	42	9/14/19	9/16/19	2	1	0	5%
2	137	9/18/19	9/25/19	5	1	0	6.5%
6	602	10/5/19	10/10/19	7	5	0.5	8.5%
5	162	10/2/19	10/4/19	13	1	0	6.5%
3	314	10/3/19	10/6/19	14	2	0	5.5%
8	173	10/8/19	10/12/19	20	2	0	8%
4	272	9/30/19	10/15/19	52	1	0.1	8%
7	141	10/4/19	10/9/19	150	1	0.1	7%

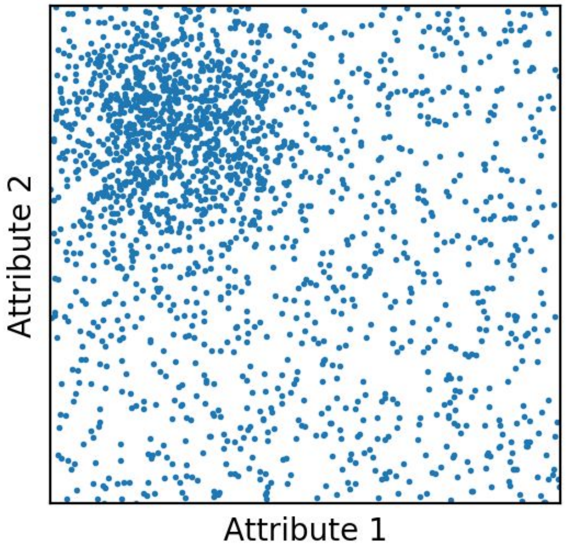
Query	**Scan Overhead**
SELECT COUNT(*) FROM table WHERE Price >= 10 AND Price < 100	1
SELECT COUNT(*) FROM table WHERE Price >= 10 AND Price < 100 AND Quantity = 1	2
SELECT COUNT(*) FROM table WHERE Quantity > 2	

Single-dimensional indexes

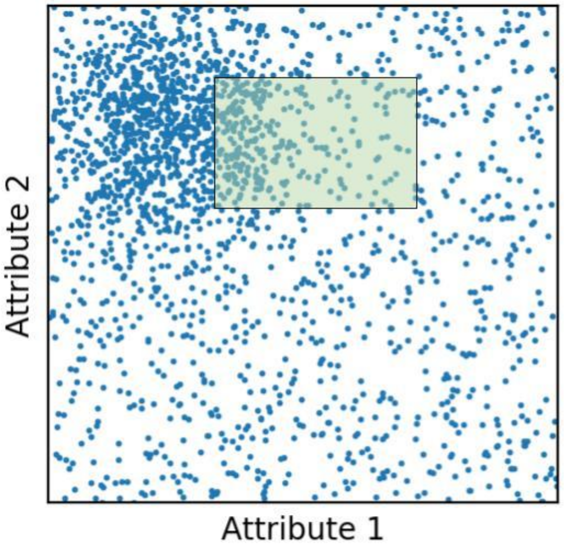
Order ID	Item ID	Ship Date	Receipt Date	***Price***	Quantity	Discount	Tax
1	42	9/14/19	9/16/19	2	1	0	5%
2	137	9/18/19	9/25/19	5	1	0	6.5%
6	602	10/5/19	10/10/19	7	5	0.5	8.5%
5	162	10/2/19	10/4/19	13	1	0	6.5%
3	314	10/3/19	10/6/19	14	2	0	5.5%
8	173	10/8/19	10/12/19	20	2	0	8%
4	272	9/30/19	10/15/19	52	1	0.1	8%
7	141	10/4/19	10/9/19	150	1	0.1	7%

Query	**Scan Overhead**
SELECT COUNT(*) FROM table WHERE Price >= 10 AND Price < 100	1
SELECT COUNT(*) FROM table WHERE Price >= 10 AND Price < 100 AND Quantity = 1	2
SELECT COUNT(*) FROM table WHERE Quantity > 2	8

Multi-dimensional indexes



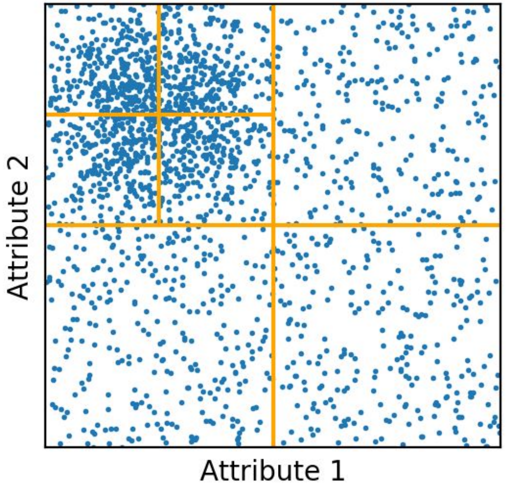
Multi-dimensional indexes



Query

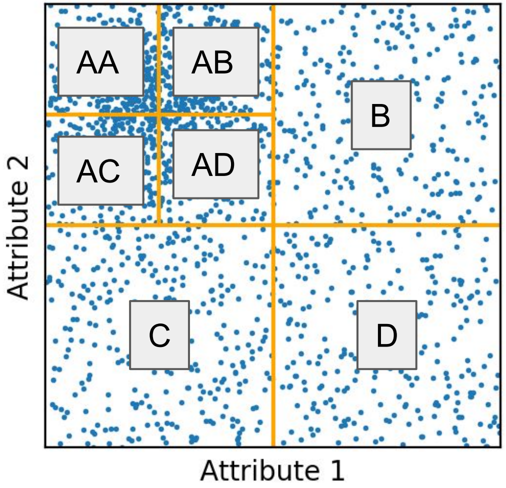
SELECT COUNT(*)
FROM table
WHERE Attribute 1 >= A
AND Attribute 1 <= B
AND Attribute 2 >= C
AND Attribute 2 <= D

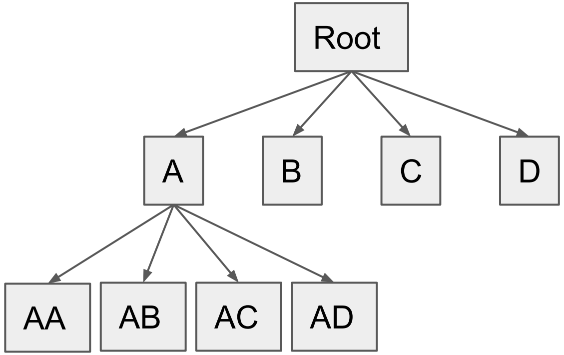
Existing multi-dimensional indexes



- Quadtree (hyper-octree in higher dimensions)

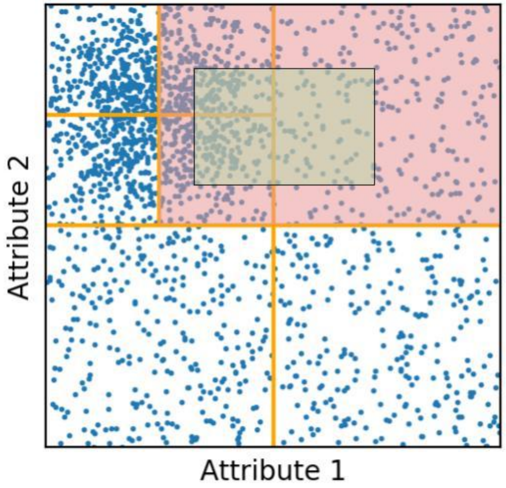
Existing multi-dimensional indexes

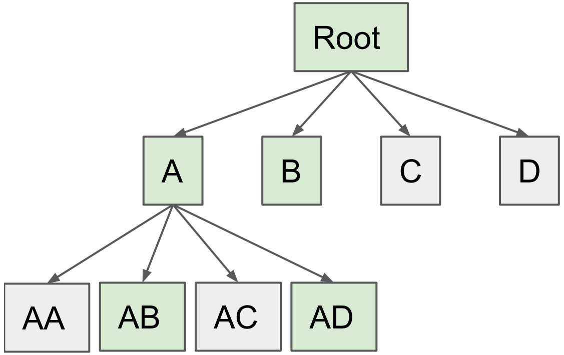




- Quadtree (hyper-octree in higher dimensions)

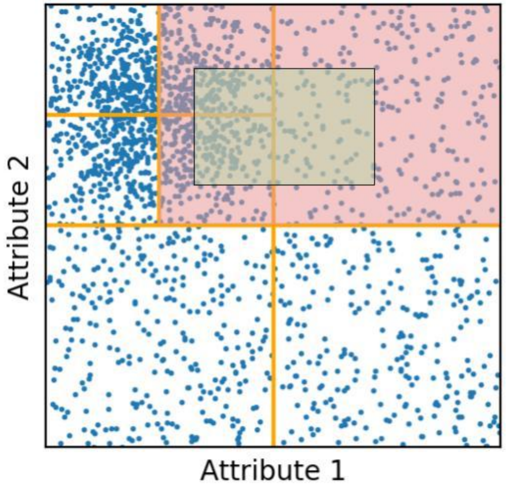
Existing multi-dimensional indexes

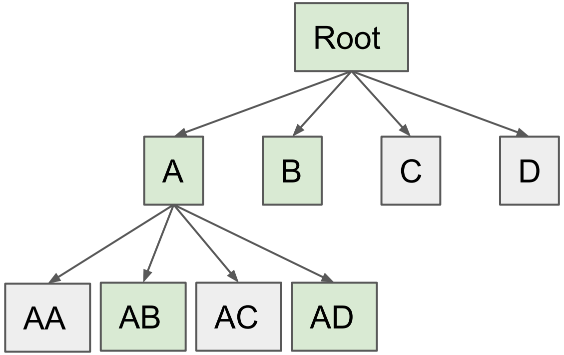




- Quadtree (hyper-octree in higher dimensions)

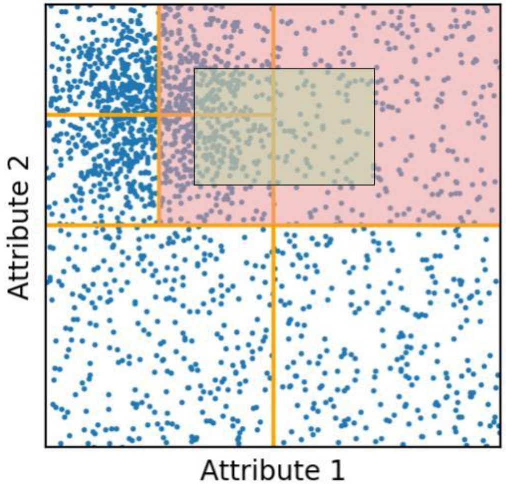
Existing multi-dimensional indexes

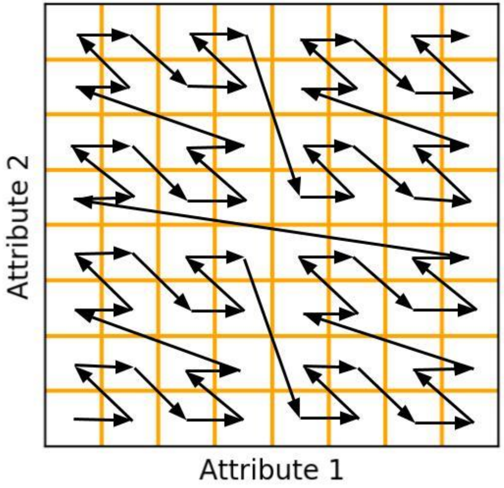




- Quadtree (hyper-octree in higher dimensions)
- Other tree-based indexes: R-tree, k-d tree
- Geospatial databases

Existing multi-dimensional indexes

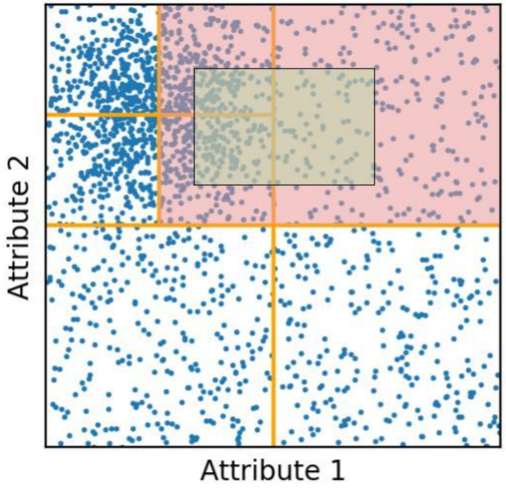


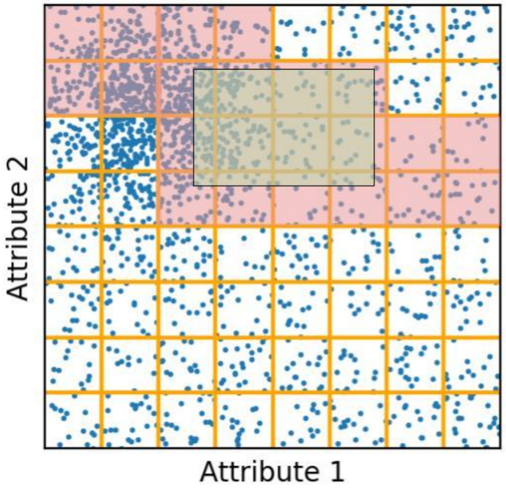


- Quadtree (hyper-octree in higher dimensions)
- Other tree-based indexes: R-tree, k-d tree
- Geospatial databases

- Z-order

Existing multi-dimensional indexes

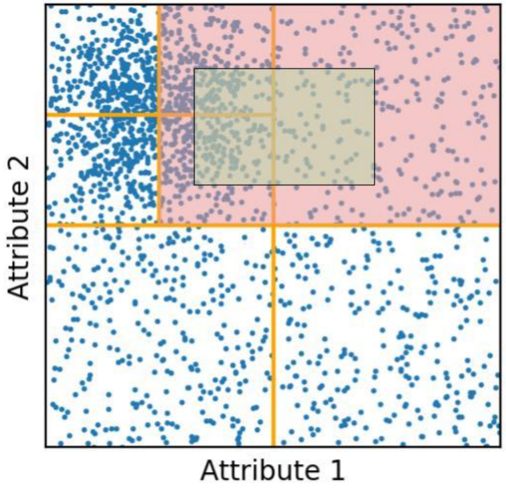


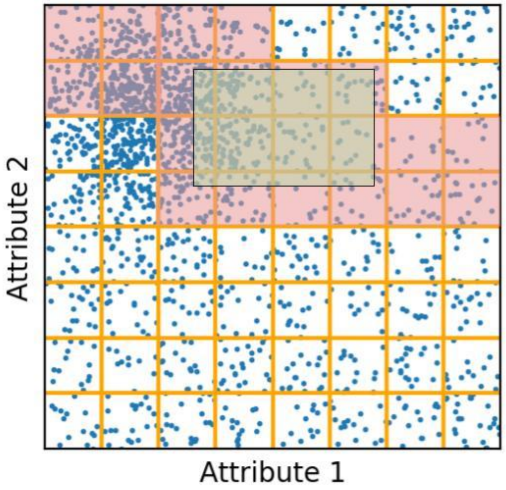


- Quadtree (hyper-octree in higher dimensions)
- Other tree-based indexes: R-tree, k-d tree
- Geospatial databases

- Z-order

Existing multi-dimensional indexes

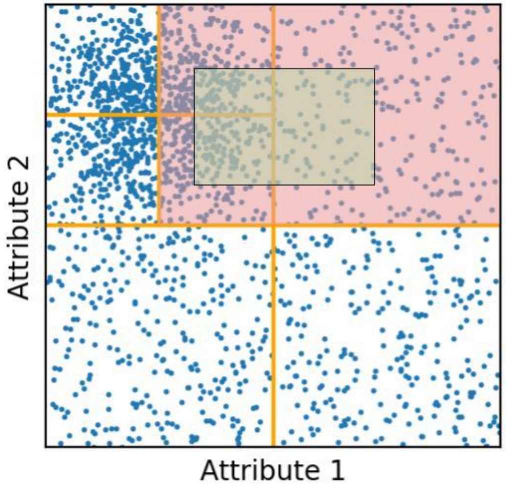


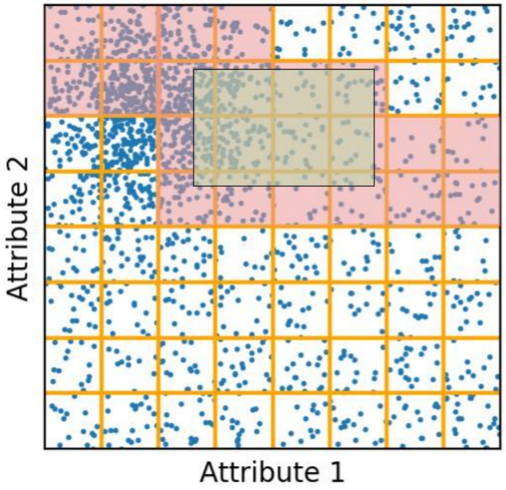


- Quadtree (hyper-octree in higher dimensions)
- Other tree-based indexes: R-tree, k-d tree
- Geospatial databases

- Z-order
- Other sort-order-based indexes: UB-tree
- Amazon Redshift

Existing multi-dimensional indexes





- Quadtree (hyper-octree in higher dimensions)
- Other tree-based indexes: R-tree, k-d tree
- Geospatial databases

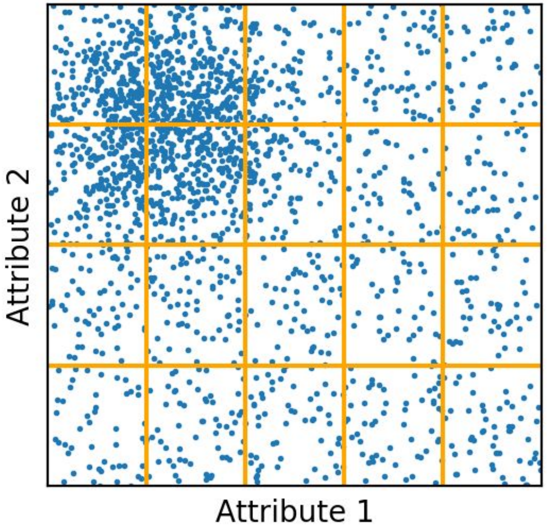
- Z-order
- Other sort-order-based indexes: UB-tree
- Amazon Redshift

Drawbacks

- Difficult to create and maintain, requires DBA
- No index dominates all others
- Does not allow fined-grained customization

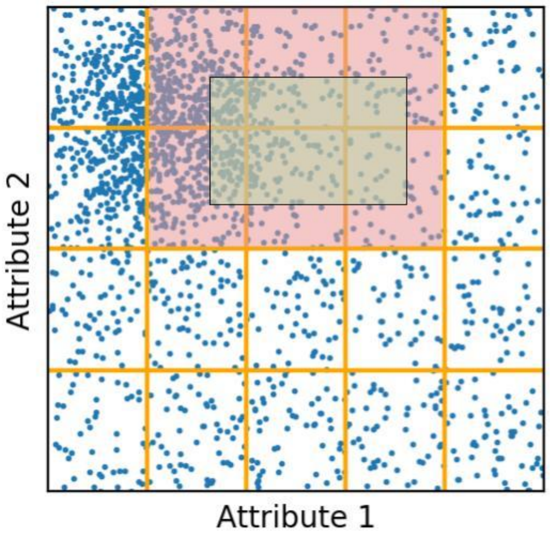
Our new index: *Flood*

- Multi-dimensional in-memory read-optimized index
- Grid-based layout



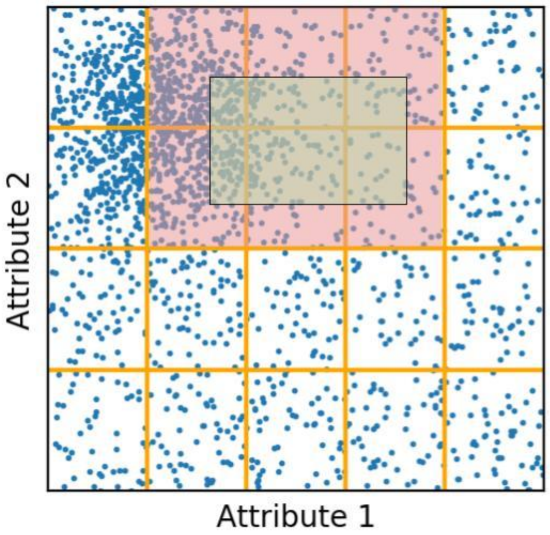
Our new index: *Flood*

- Multi-dimensional in-memory read-optimized index
- Grid-based layout



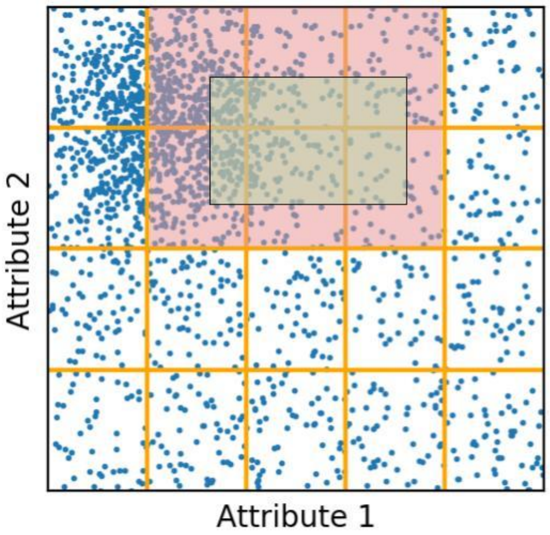
Our new index: *Flood*

- Multi-dimensional in-memory read-optimized index
- Grid-based layout
  - Low index time (vs. tree-based index)
  - Has good number of tunable parameters



Our new index: *Flood*

- Multi-dimensional in-memory read-optimized index
- Grid-based layout
  - Low index time (vs. tree-based index)
  - Has good number of tunable parameters
- Key idea: learning-based approach to jointly optimize the index structure and layout
  - Learn from data
  - Learn from queries

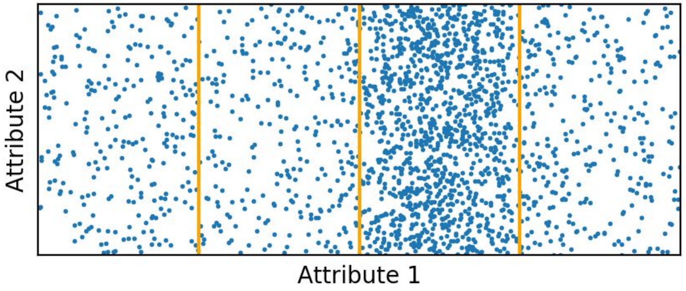


(1) Learning from the data: “flattening”

- Goal: uniform number of points per cell

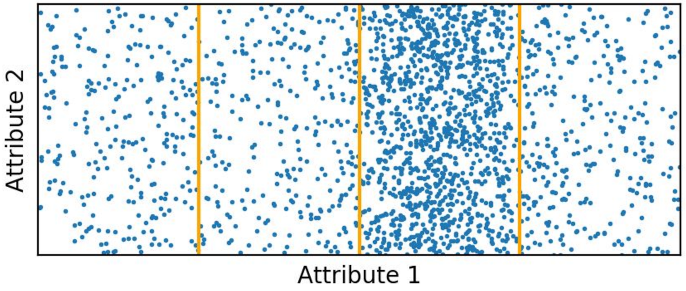
(1) Learning from the data: “flattening”

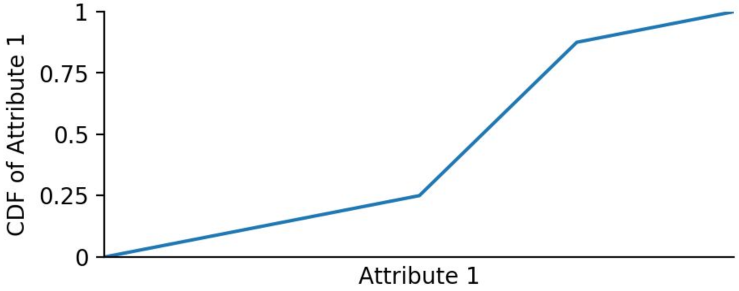
- Goal: uniform number of points per cell



(1) Learning from the data: “flattening”

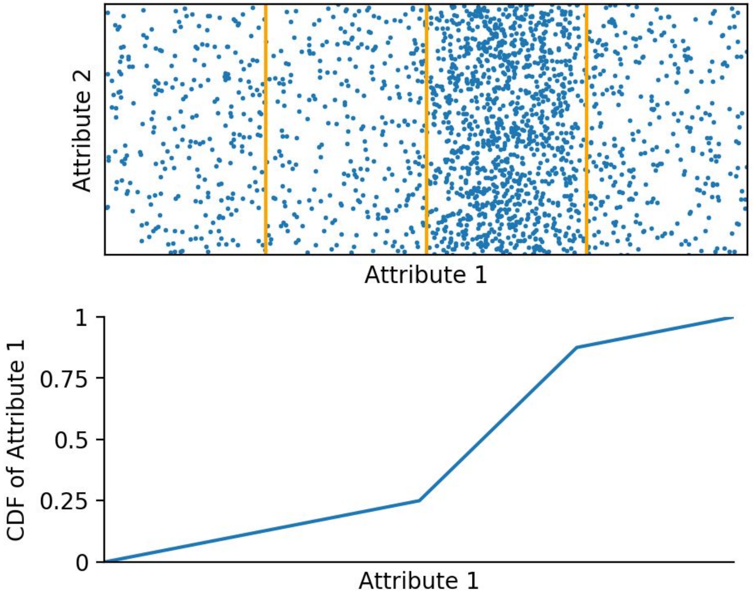
- Goal: uniform number of points per cell
- Key idea: model the CDF of each dimension

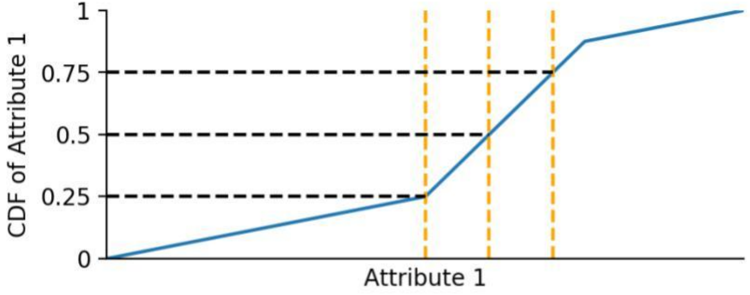




(1) Learning from the data: “flattening”

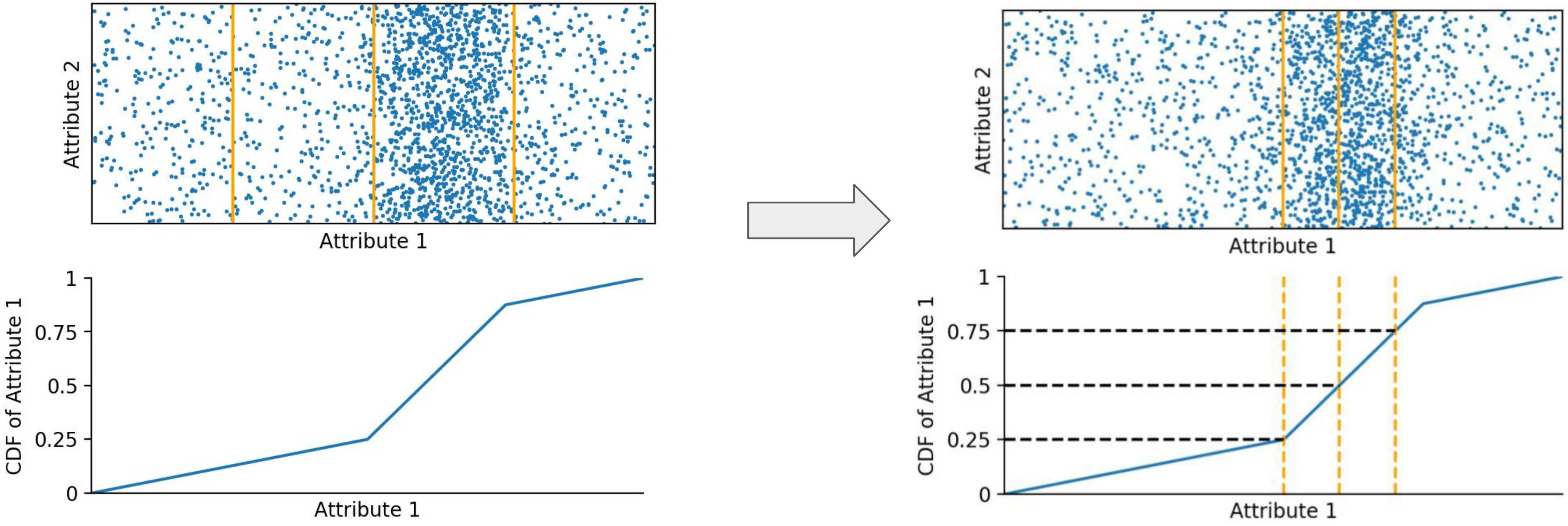
- Goal: uniform number of points per cell
- Key idea: model the CDF of each dimension





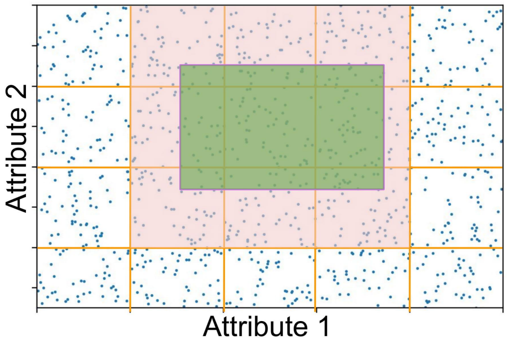
(1) Learning from the data: “flattening”

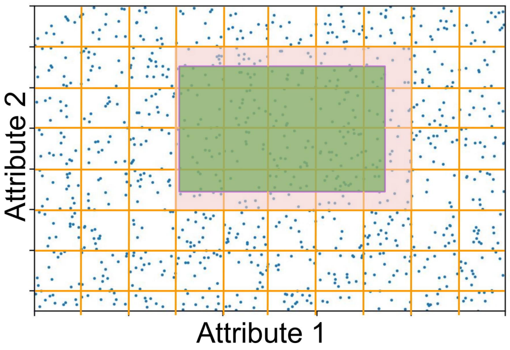
- Goal: uniform number of points per cell
- Key idea: model the CDF of each dimension



(2) Learning from the queries: optimizing layout

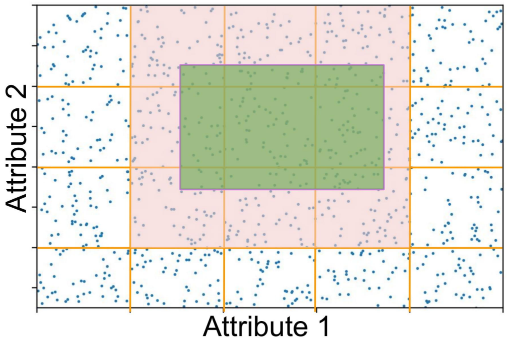
- Goal: find optimal number of partitions in each dimension

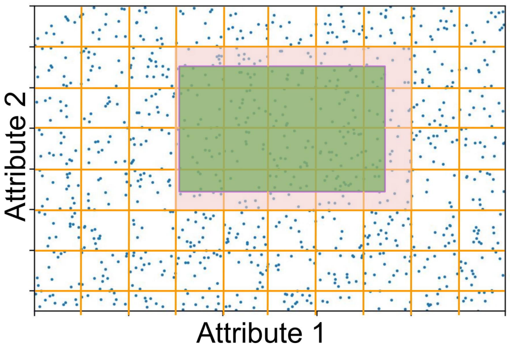




(2) Learning from the queries: optimizing layout

- Goal: find optimal number of partitions in each dimension

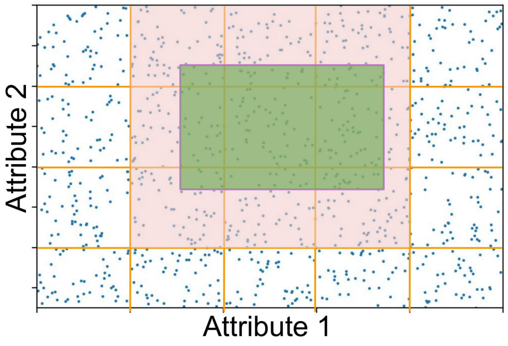




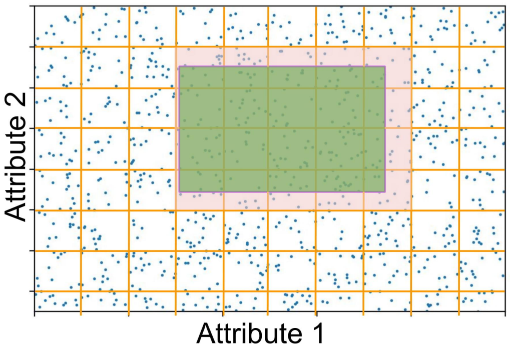
- Pro: Lower scan overhead
- Con: More cells

(2) Learning from the queries: optimizing layout

- Goal: find optimal number of partitions in each dimension



- Pro: Fewer cells
- Con: Higher scan overhead



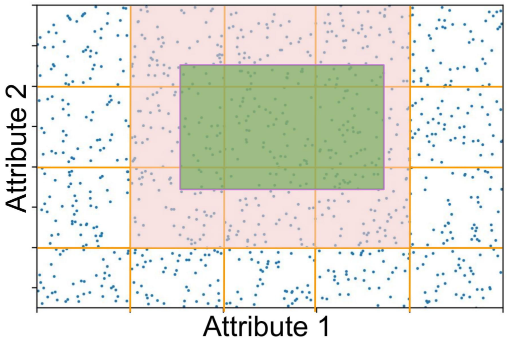
- Pro: Lower scan overhead
- Con: More cells

(2) Learning from the queries: optimizing layout

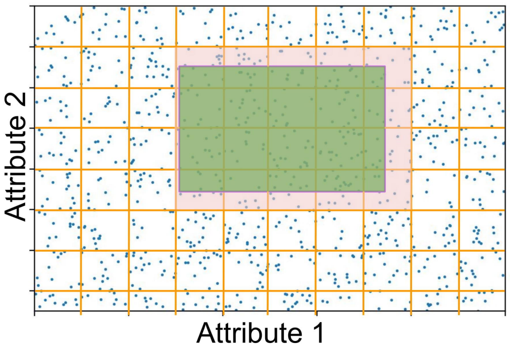
- Goal: find optimal number of partitions in each dimension
- Key idea: use cost model to predict query time

$$Time(D, q, L) = w_c N_c + w_s N_s$$

\# cells   # scanned points



- Pro: Fewer cells
- Con: Higher scan overhead



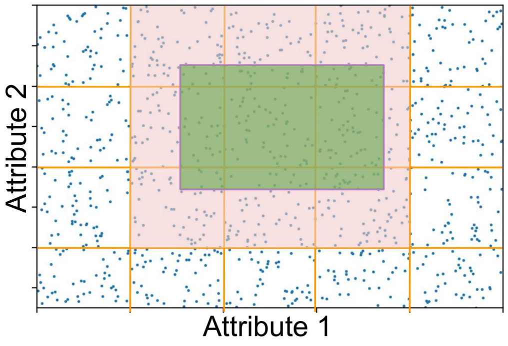
- Pro: Lower scan overhead
- Con: More cells

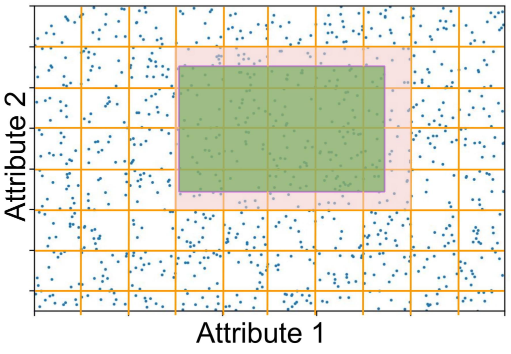
(2) Learning from the queries: optimizing layout

- Goal: find optimal number of partitions in each dimension
- Key idea: use cost model to predict query time

$$Time(D, q, L) = w_c N_c + w_s N_s$$

N_c: # cells N_s: # scanned points





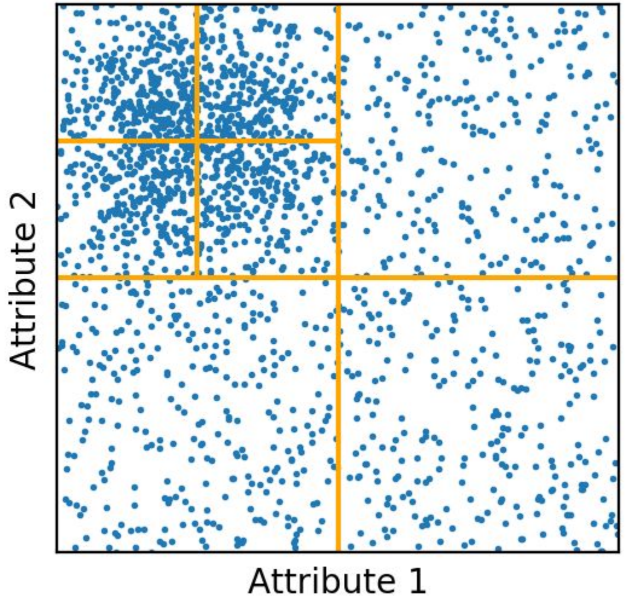
- Solve for layout with lowest average query time using gradient descent

- Pro: Fewer cells
- Con: Higher scan overhead

- Pro: Lower scan overhead
- Con: More cells

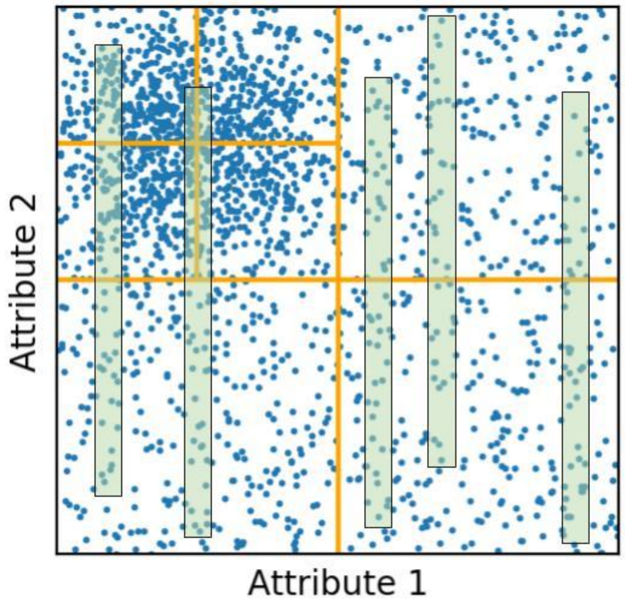
(2) Learning from the queries: optimizing layout

Quadtree



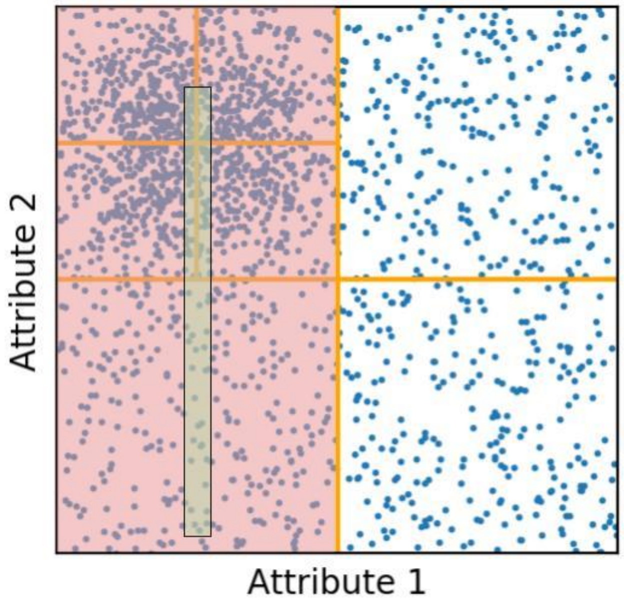
(2) Learning from the queries: optimizing layout

Quadtree



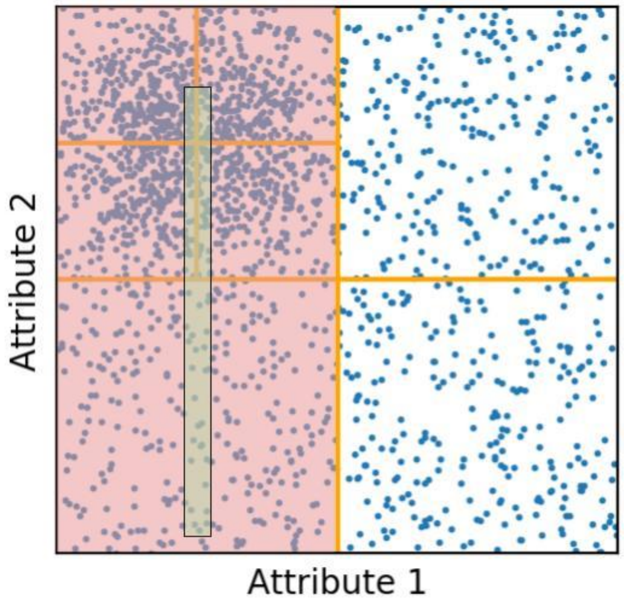
(2) Learning from the queries: optimizing layout

Quadtree

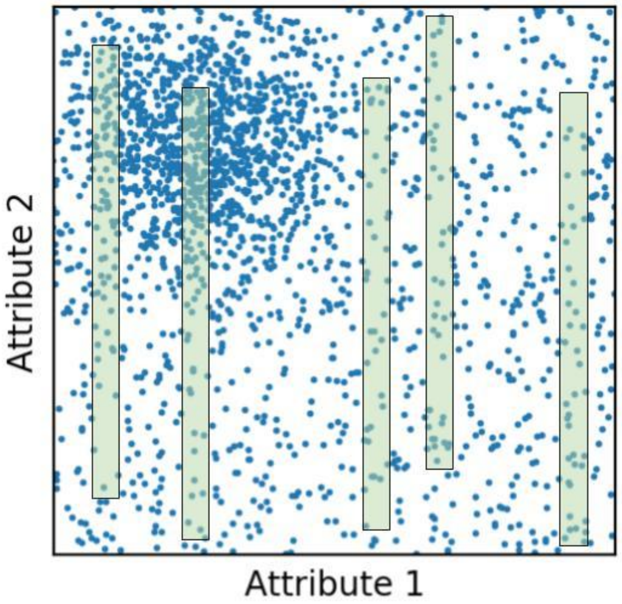


(2) Learning from the queries: optimizing layout

Quadtree

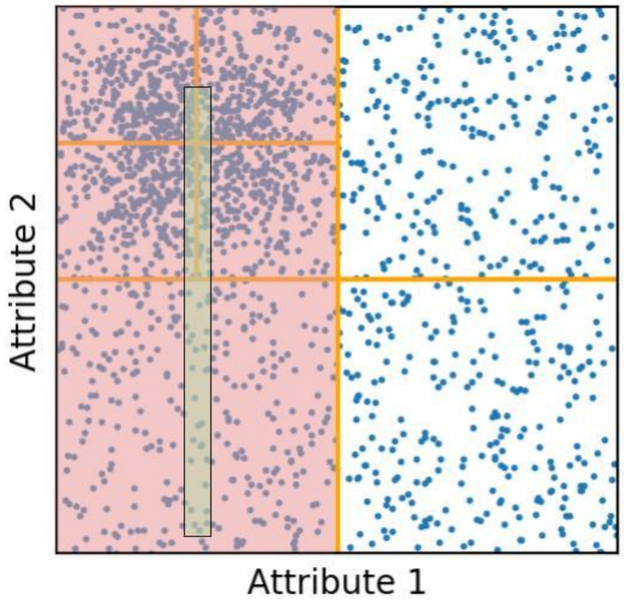


Flood

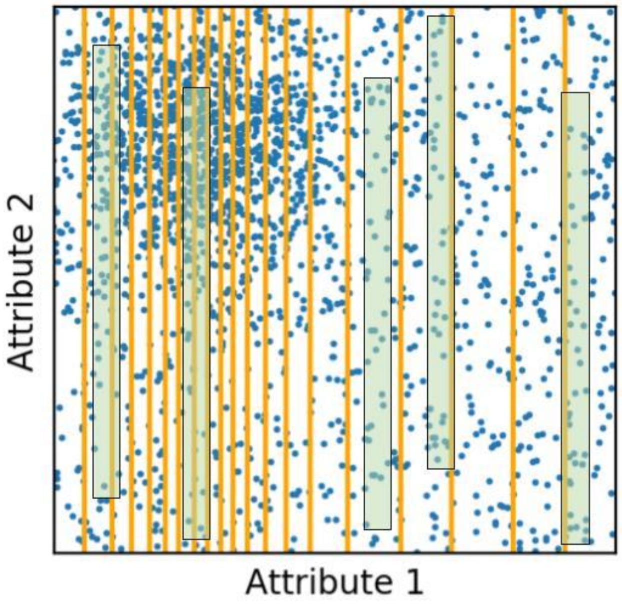


(2) Learning from the queries: optimizing layout

Quadtree

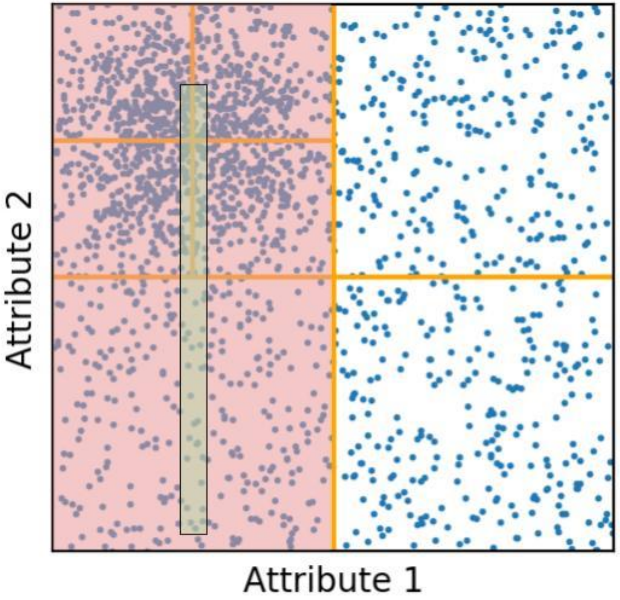


Flood

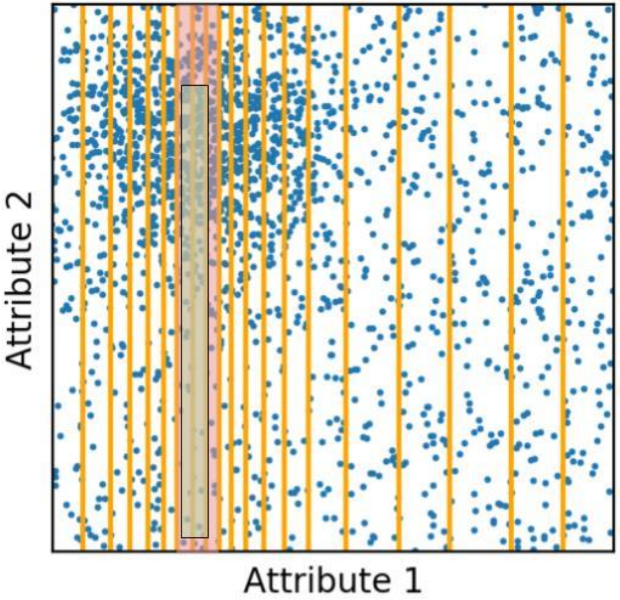


(2) Learning from the queries: optimizing layout

Quadtree



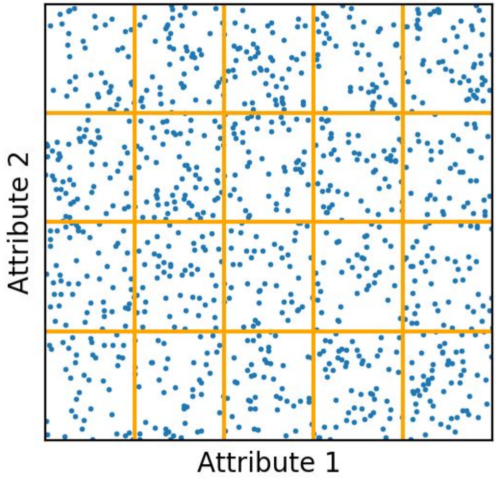
Flood



(3) Optimization: the “sort dimension”

<u>“Naive” grid</u>

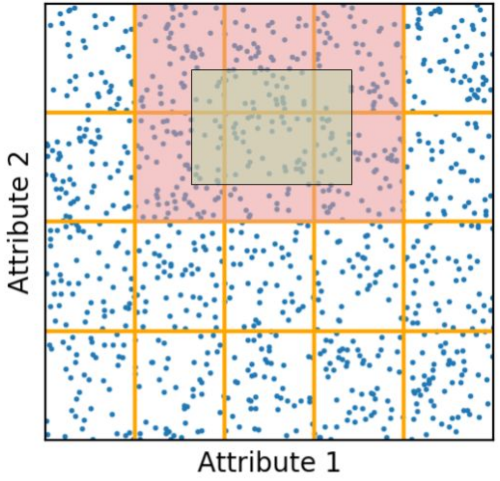
- Grid over d dimensions
- Cells are ordered
- Within each cell, points are unsorted



(3) Optimization: the “sort dimension”

<u>“Naive” grid</u>

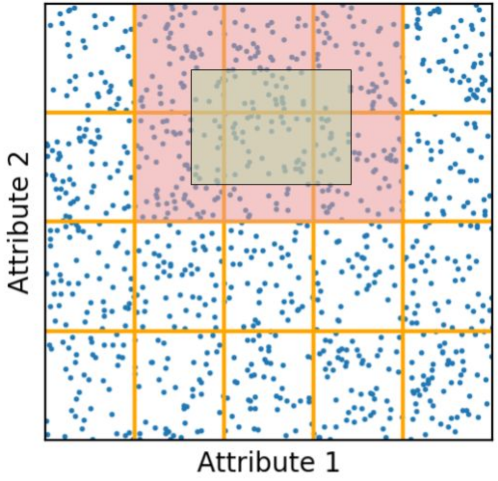
- Grid over d dimensions
- Cells are ordered
- Within each cell, points are unsorted



(3) Optimization: the “sort dimension”

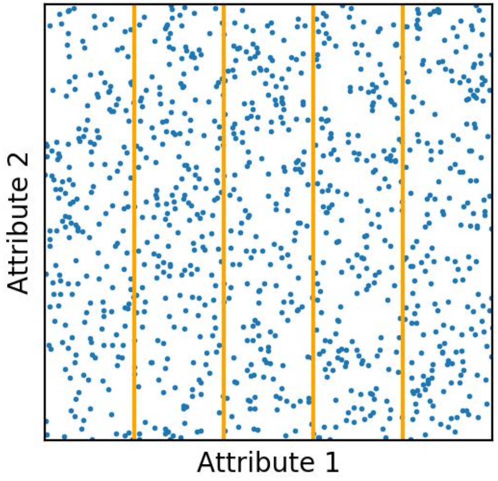
<u>“Naive” grid</u>

- Grid over d dimensions
- Cells are ordered
- Within each cell, points are unsorted



<u>Flood’s grid</u>

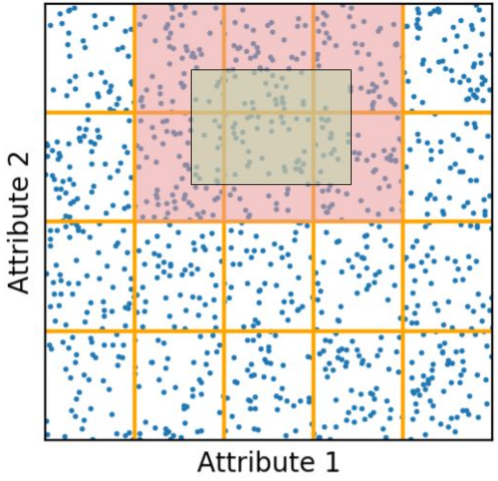
- Grid over d-1 dimensions
- Cells are ordered
- Within each cell, points are sorted by d-th dimension



(3) Optimization: the “sort dimension”

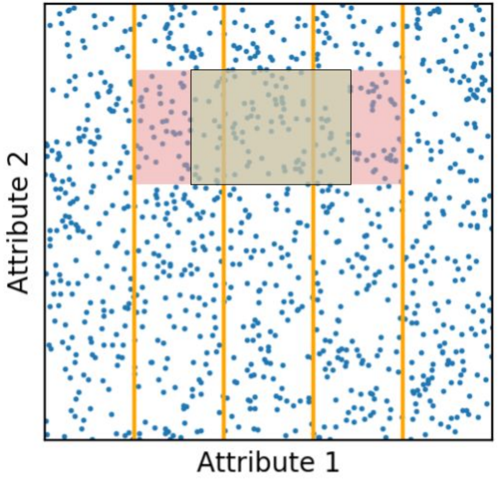
“Naive” grid

- Grid over d dimensions
- Cells are ordered
- Within each cell, points are unsorted



Flood’s grid

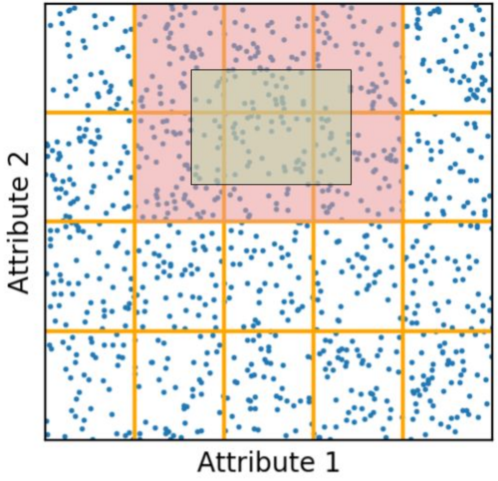
- Grid over d-1 dimensions
- Cells are ordered
- Within each cell, points are sorted by d-th dimension



(3) Optimization: the “sort dimension”

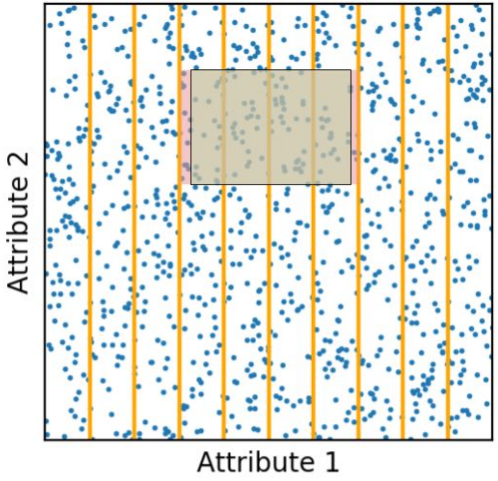
“Naive” grid

- Grid over d dimensions
- Cells are ordered
- Within each cell, points are unsorted



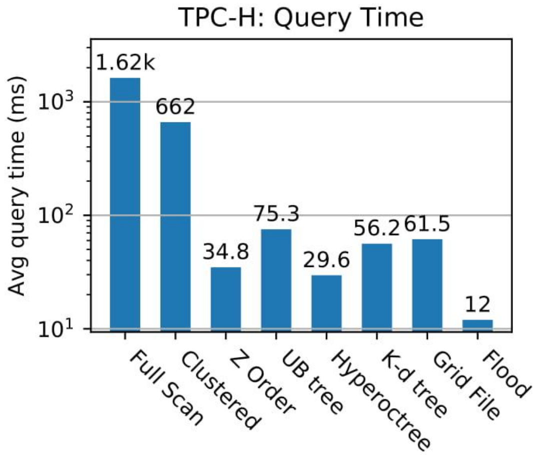
Flood’s grid

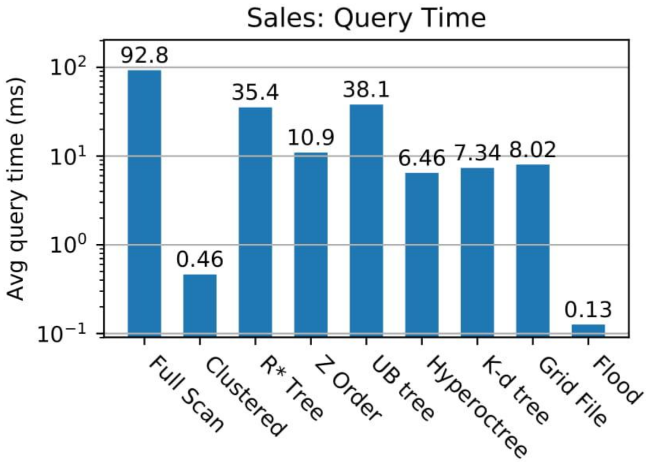
- Grid over d-1 dimensions
- Cells are ordered
- Within each cell, points are sorted by d-th dimension



Results

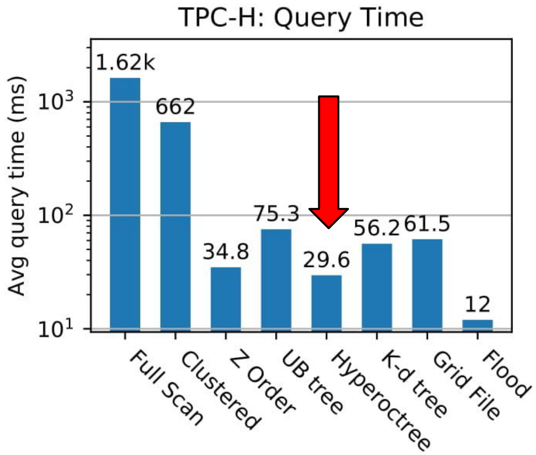
- How does Flood compare to other indexes?
  - Faster than every other index

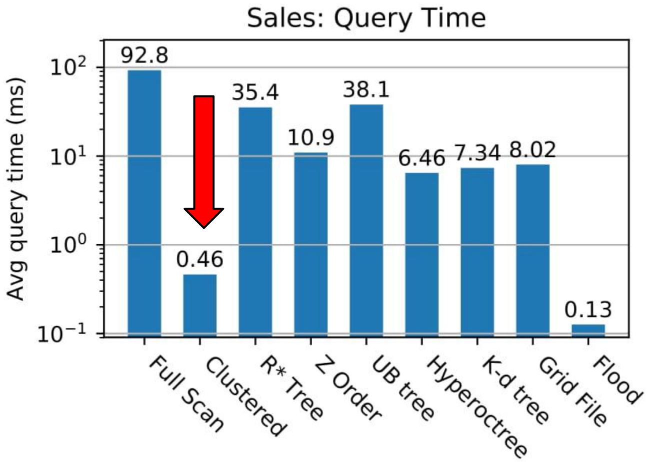




Results

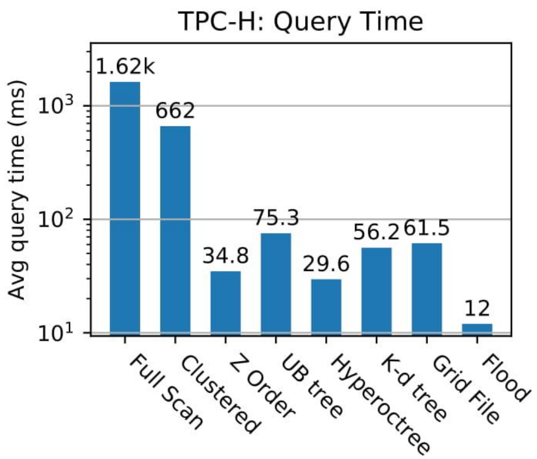
- How does Flood compare to other indexes?
  - Faster than every other index





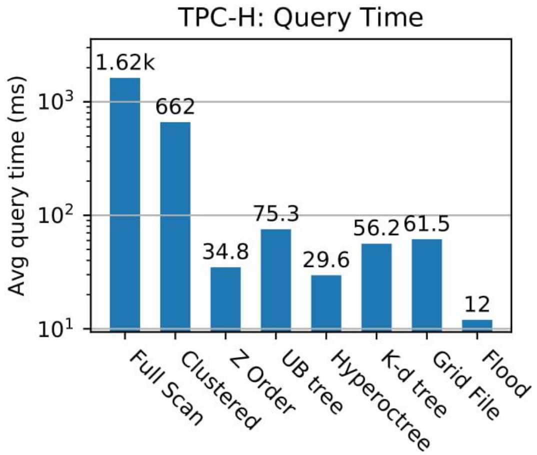
Results

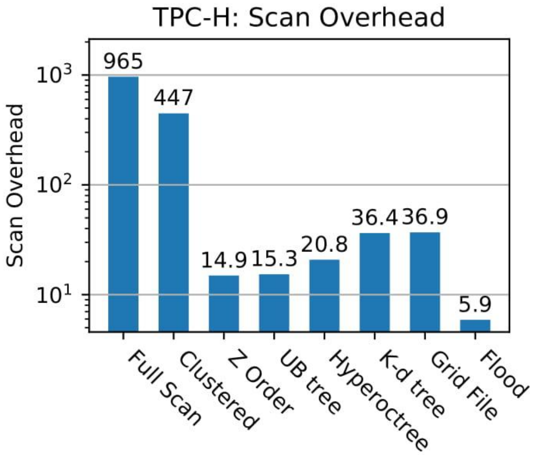
- How does Flood compare to other indexes?
  - Faster than every other index
- Where does Flood's advantage come from?
  - Low scan overhead



Results

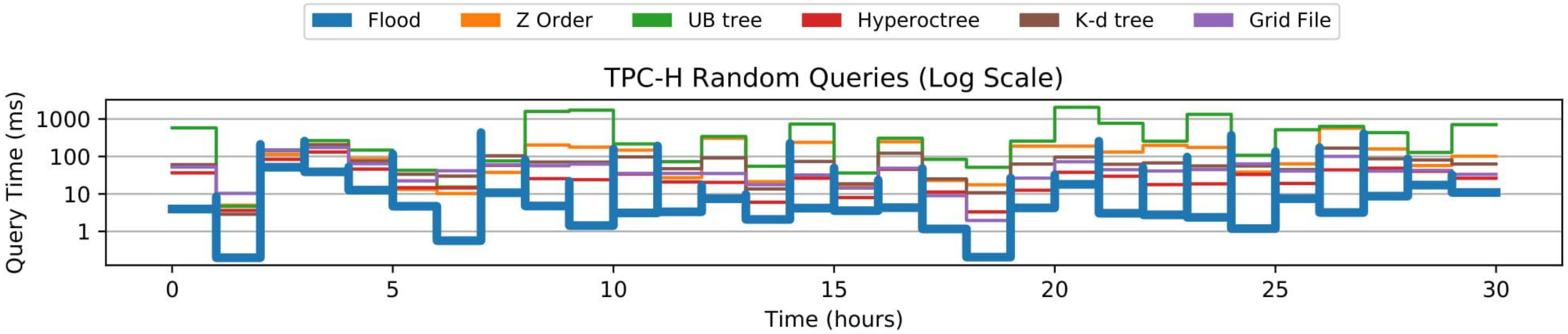
- How does Flood compare to other indexes?
  - Faster than every other index
- Where does Flood's advantage come from?
  - Low scan overhead





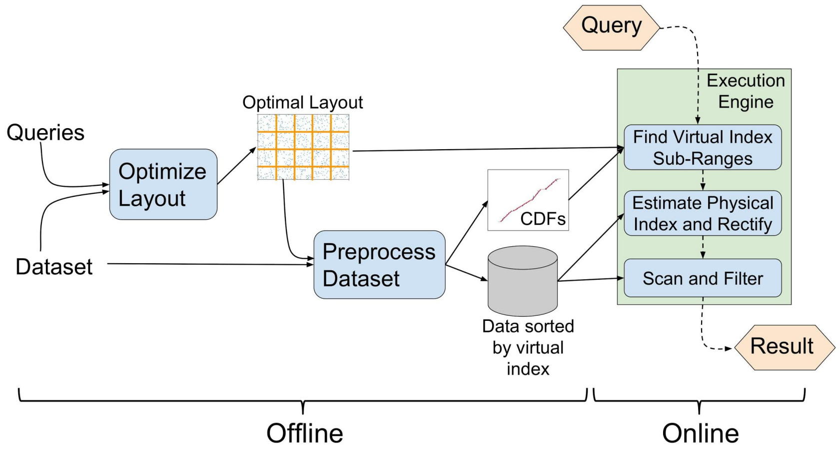
Results

- How does Flood compare to other indexes?
  - Faster than every other index
- Where does Flood's advantage come from?
  - Low scan overhead
- What if the query workload changes?
  - Flood can adapt



Summary of Flood

- Multi-dimensional in-memory read-optimized index
- Automatically learned based on data distribution and query workload
- Outperforms all other indexes by achieving lower scan overhead



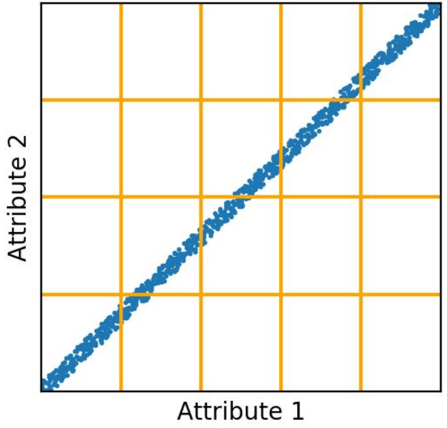
Outline

1. Completed work: Flood (SIGMOD 2020)
2. **Future work: column correlations, query skew, categorical attributes**

(1) Column correlations

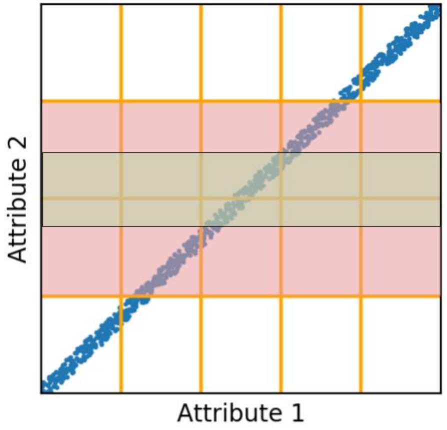
(1) Column correlations

Monotonic correlations



(1) Column correlations

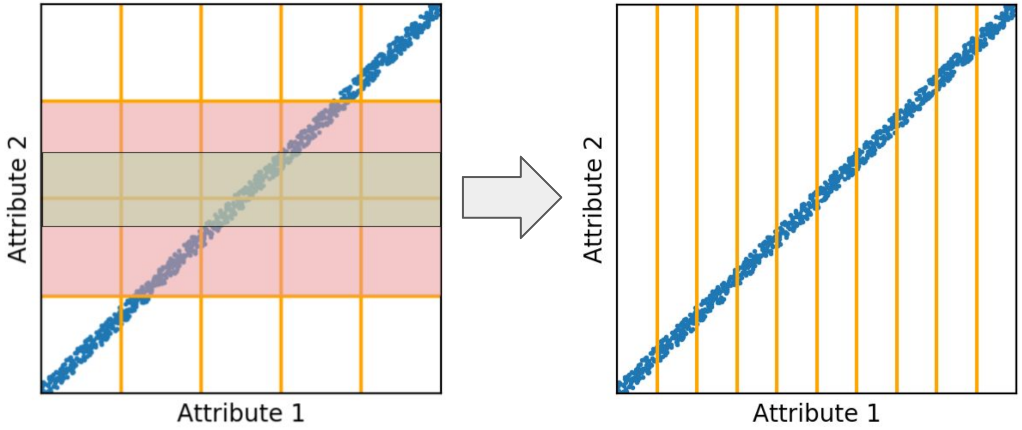
Monotonic correlations



(1) Column correlations

Monotonic correlations

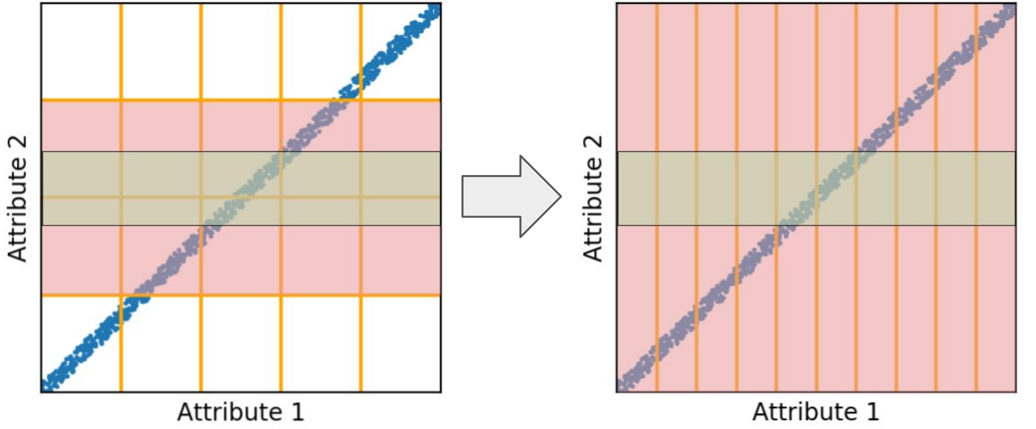
- Possible solution: function-based mapping



(1) Column correlations

Monotonic correlations

- Possible solution: function-based mapping

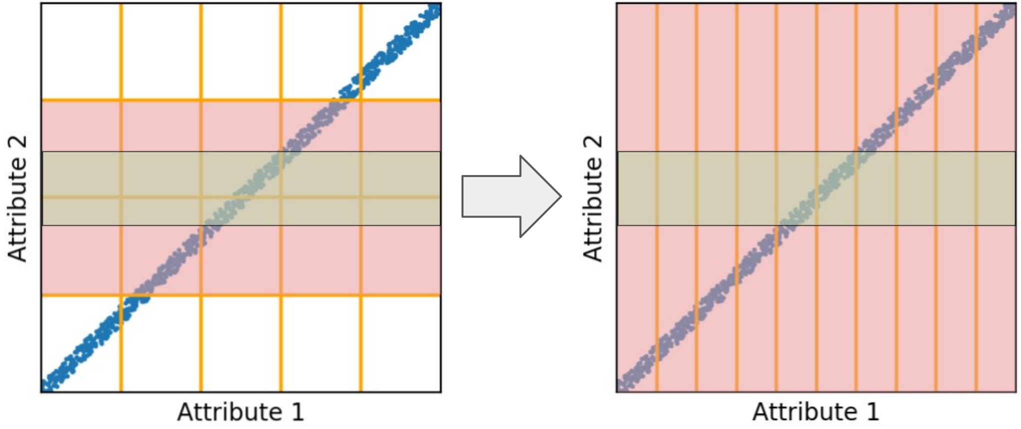


(1) Column correlations

Monotonic correlations

- Possible solution: function-based mapping

[A_min, A_max] = Fn([B_min, B_max]) + error

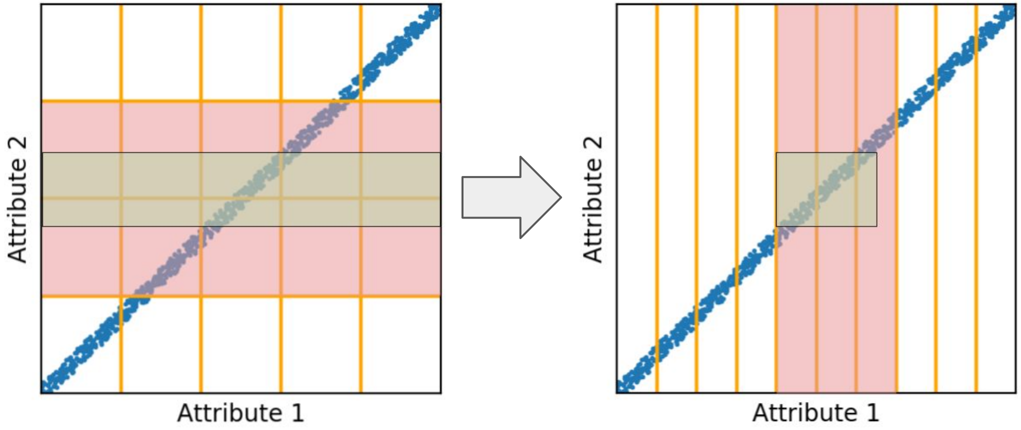


(1) Column correlations

Monotonic correlations

- Possible solution: function-based mapping

[A_min, A_max] = Fn([B_min, B_max]) + error

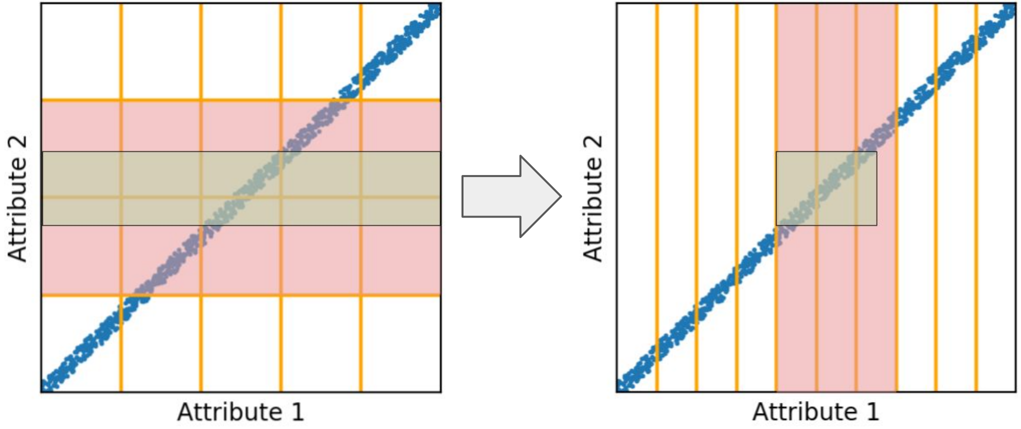


(1) Column correlations

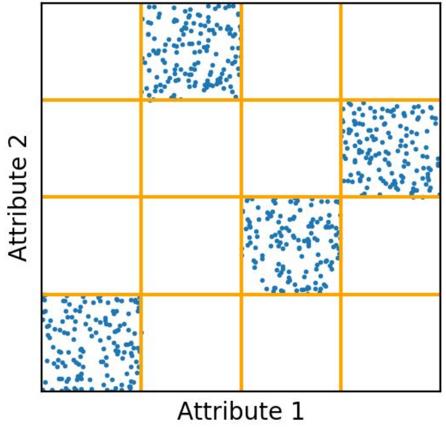
Monotonic correlations

- Possible solution: function-based mapping

[A_min, A_max] = Fn([B_min, B_max]) + error



Hotspots

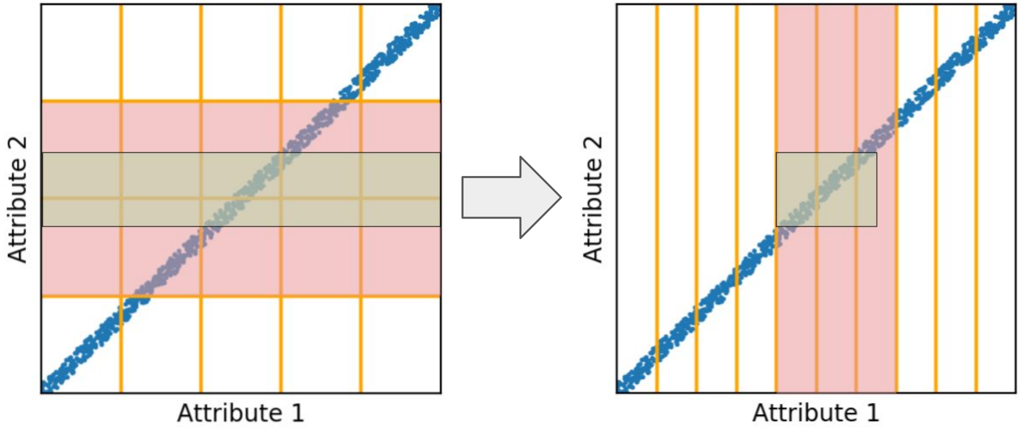


(1) Column correlations

Monotonic correlations

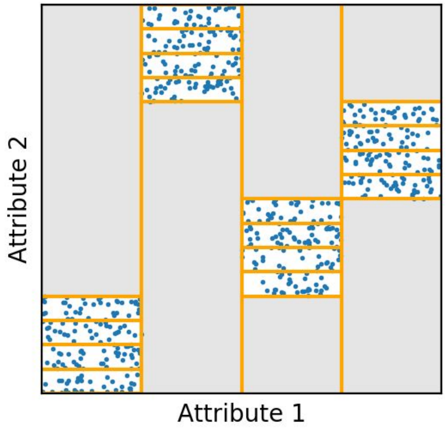
- Possible solution: function-based mapping

[A_min, A_max] = Fn([B_min, B_max]) + error



Hotspots

- Possible solution: “non-uniform grid”

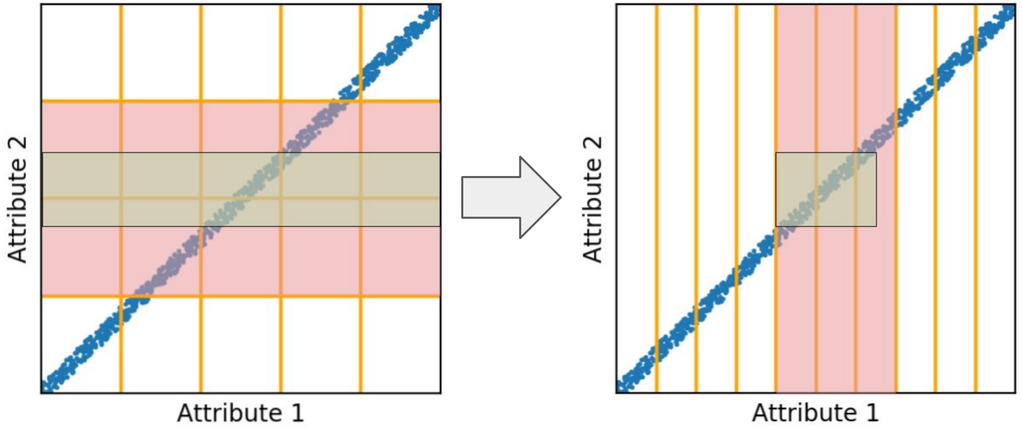


(1) Column correlations

Monotonic correlations

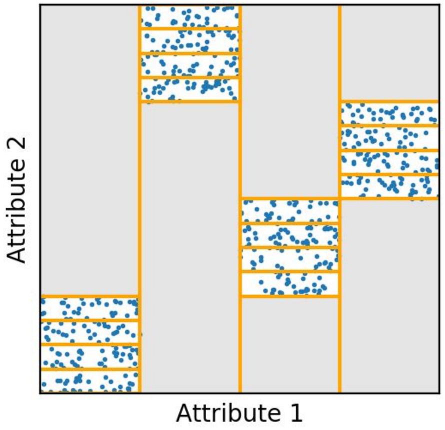
- Possible solution: function-based mapping

[A_min, A_max] = Fn([B_min, B_max]) + error



Hotspots

- Possible solution: “non-uniform grid”



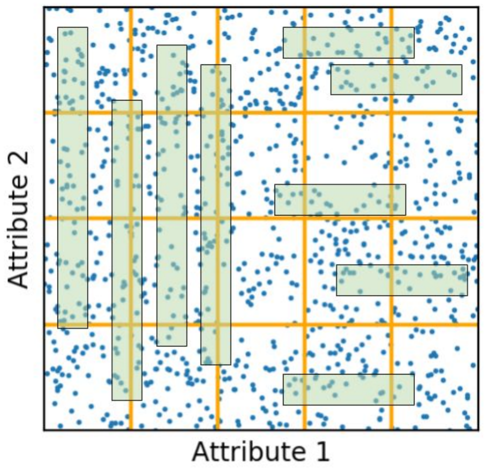
***Challenge**: combinatorial explosion of possible layouts*

(2) Query skew

- Queries “look different” in different regions
  - Selectivity
  - Frequency

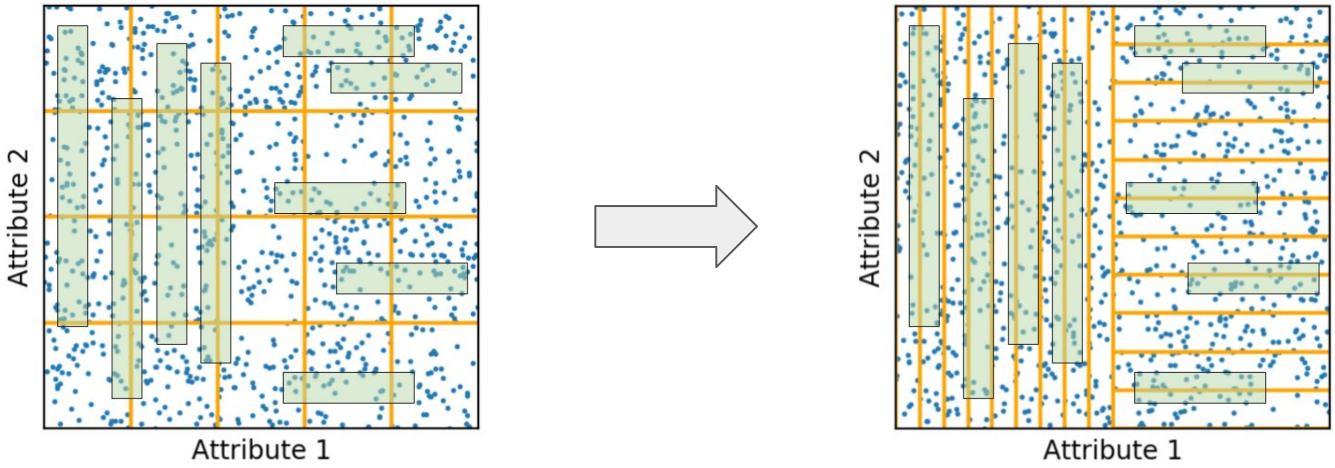
(2) Query skew

- Queries “look different” in different regions
  - Selectivity
  - Frequency



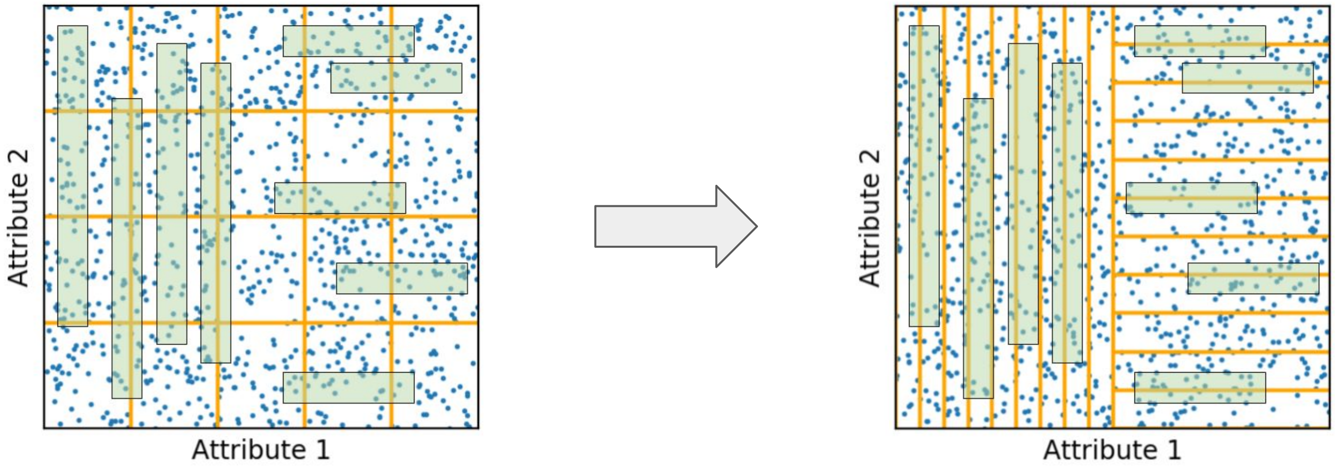
(2) Query skew

- Queries “look different” in different regions
  - Selectivity
  - Frequency



(2) Query skew

- Queries “look different” in different regions
  - Selectivity
  - Frequency
- Possible solution: “Flood tree”
  - Lightweight decision tree
  - Each leaf node is an instance of Flood



(3) Categorical attributes

- Opportunity 1: No semantic sort order
  - Order based on co-access frequency

(3) Categorical attributes

- Opportunity 1: No semantic sort order
  - Order based on co-access frequency
- Opportunity 2: Column correlations
  - Direct mapping

rid	***State***	**City**	**...**
0	IL	Springfield	...
1	MA	Salem	...
2	MA	Springfield	...
3	MA	Salem	...
4	MO	Springfield	...
5	OR	Salem	...

(3) Categorical attributes

- Opportunity 1: No semantic sort order
  - Order based on co-access frequency
- Opportunity 2: Column correlations
  - Direct mapping

rid	***<u>State</u>***	City	...
0	IL	Springfield	...
1	MA	Salem	...
2	MA	Springfield	...
3	MA	Salem	...
4	MO	Springfield	...
5	OR	Salem	...

Secondary index

City	rid
Salem	1
Salem	3
Salem	5
Springfield	0
Springfield	2
Springfield	4

(3) Categorical attributes

- Opportunity 1: No semantic sort order
    - Order based on co-access frequency
- Opportunity 2: Column correlations
    - Direct mapping

rid	***State***	City	...
0	IL	Springfield	...
1	MA	Salem	...
2	MA	Springfield	...
3	MA	Salem	...
4	MO	Springfield	...
5	OR	Salem	...

Secondary index

City	rid
Salem	1
Salem	3
Salem	5
Springfield	0
Springfield	2
Springfield	4

Direct Mapping

City	State
Salem	{MA, OR}
Springfield	{IL, MA, MO}

(3) Categorical attributes

- Opportunity 1: No semantic sort order
    - Order based on co-access frequency
- Opportunity 2: Column correlations
    - Direct mapping

rid	***State***	City	...
0	IL	Springfield	...
1	MA	Salem	...
2	MA	Springfield	...
3	MA	Salem	...
4	MO	Springfield	...
5	OR	Salem	...

Secondary index

City	rid
Salem	1
Salem	3
Salem	5
Springfield	0
Springfield	2
Springfield	4

Direct Mapping

City	State
Salem	{MA, OR}
Springfield	{IL, MA, MO}

(3) Categorical attributes

- Opportunity 1: No semantic sort order
  - Order based on co-access frequency
- Opportunity 2: Column correlations
  - Direct mapping

rid	***State***	City	...
0	IL	Springfield	...
1	MA	Salem	...
2	MA	Springfield	...
3	MA	Salem	...
4	MO	Springfield	...
5	OR	Salem	...

Secondary index

City	rid
Salem	1
Salem	3
Salem	5
Springfield	0
Springfield	2
Springfield	4

Direct Mapping

City	State
Salem	{MA, OR}
Springfield	{IL, MA, MO}

Pro: smaller space

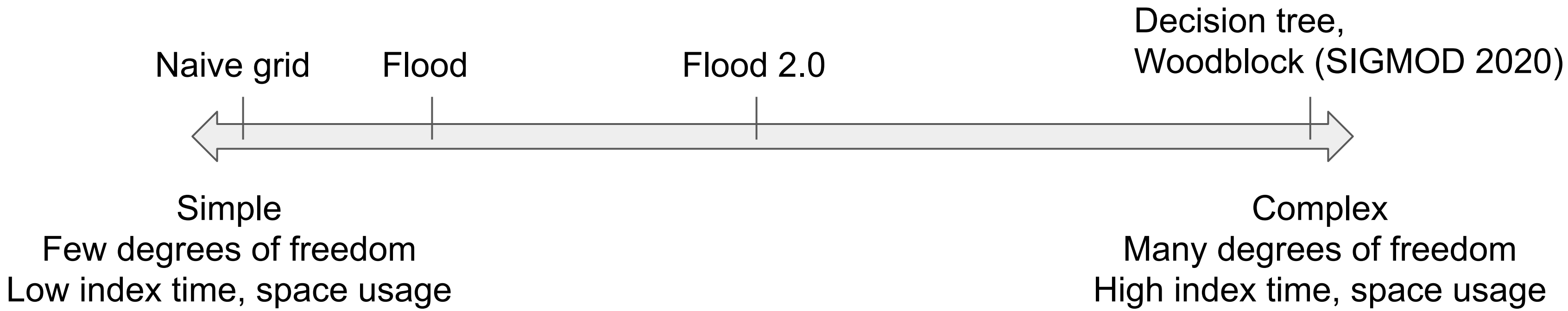
Con: higher scan overhead?

Summary of Future Work

- Column correlations
  - Solution: function-based mapping, non-uniform grid
- Query skew
  - Solution: Flood tree
- Categorical attributes
  - Solution: ordering based on co-access frequency, direct mapping

Summary of Future Work

- Column correlations
  - Solution: function-based mapping, non-uniform grid
- Query skew
  - Solution: Flood tree
- Categorical attributes
  - Solution: ordering based on co-access frequency, direct mapping



http://dsg.csail.mit.edu/mlforsystems/