Data Structures and Algorithms
Review and Return to Cardinality

CS 225
Brad Solomon

December 2, 2024

Welcome back for exactly 1 week of lectures!

Then review :)

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Course Announcements

This week's lab is **optional.** Will be worth the equivalent value in EC

Part 2 of **External Research Survey** releases tomorrow! Worth 2 EC

Reminder: Exam 5 is this week!

Reminder: Final exam starts as early as Thursday December 12th

Please fill out ICES evaluations!

Learning Objectives

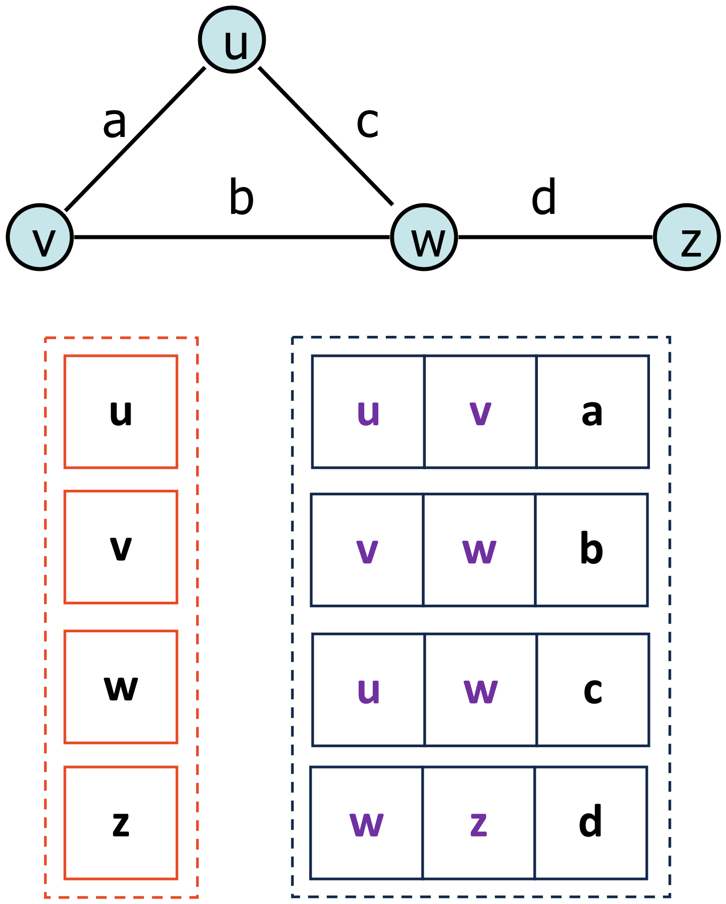
A brief review of exam 5 content

Review high level motivation behind sketching data structure

Introduce the concept of cardinality and cardinality estimation

Graph Implementation: Edge List $|V| = n, |E| = m$

The equivalent of an 'unordered' data structure



Vertex Storage:

An optional list of vertices

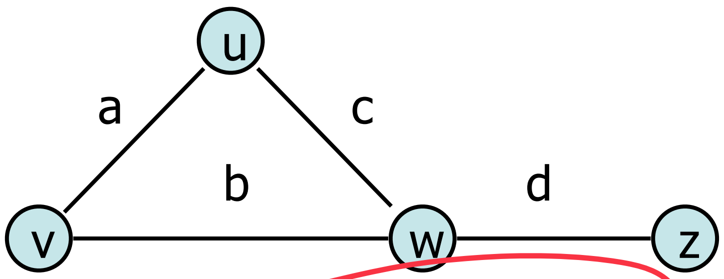
Edge Storage:

A list storing edges as (V1, V2, Weight)

Most graphs are stored as just an edge list!

Graph Implementation: Adjacency Matrix

|V|= n, |E|= m



u	0
v	1
w	2
z	3

	0	1	2	3
0	-	a	c	0
1		-	b	0
2			-	d
3				-

Vertex Storage:

A hash table of vertices

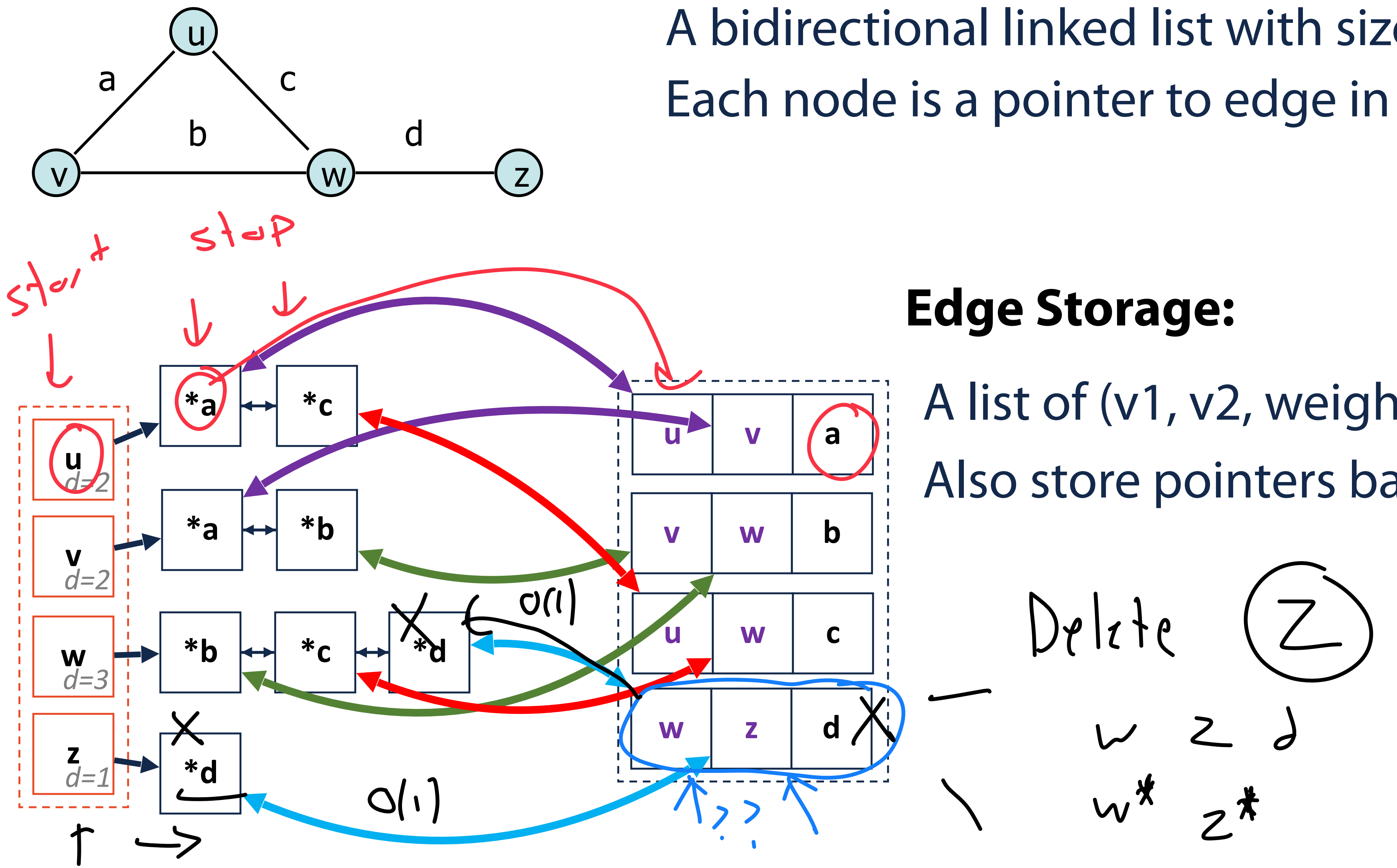
Implicitly or explicitly store index

Edge Storage:

A |V| x |V| matrix of edges

Weight is stored at position (u, v)

Adjacency List



Vertex Storage:

A bidirectional linked list with size variable

Each node is a pointer to edge in edge list

Edge Storage:

A list of (v1, v2, weight) edges

Also store pointers back to nodes

$|V| = n, |E| = m$

Expressed as O(f)	Edge List	Adjacency Matrix	Adjacency List
Space	n+m	n^2	n+m
insertVertex(v)	1*	n*	1*
removeVertex(v)	n+m	n	deg(v)
insertEdge(u, v)	1	1	1*
removeEdge(u, v)	m	1	min(deg(u), deg(v))
incidentEdges(v)	m	n	deg(v)
areAdjacent(u, v)	m	1	min(deg(u), deg(v))

Summary: DFS and BFS

$|V| = n, |E| = m$

Both are **O(n+m)** traversals! They label every edge and every node

BFS	**DFS**
Solves unweighted MST	Solves unweighted MST
Solves shortest path	
Solves cycle detection	Solves cycle detection
Memory bounded by width	Memory bounded by longest path

Kruskal's Algorithm

```
KruskalMST(G):
  DisjointSets forest
  foreach (Vertex v : G.vertices()):
    forest.makeSet(v)

  PriorityQueue Q    // min edge weight
  Q.buildFromGraph(G.edges())

  Graph T = (V, {})

  while |T.edges()| < n-1:
    Vertex (u, v) = Q.removeMin()
    if forest.find(u) != forest.find(v):
       T.addEdge(u, v)
       forest.union( forest.find(u),
                     forest.find(v) )

  return T
```

1) Build a **priority queue** on edges

2) Build a **disjoint set** on vertices

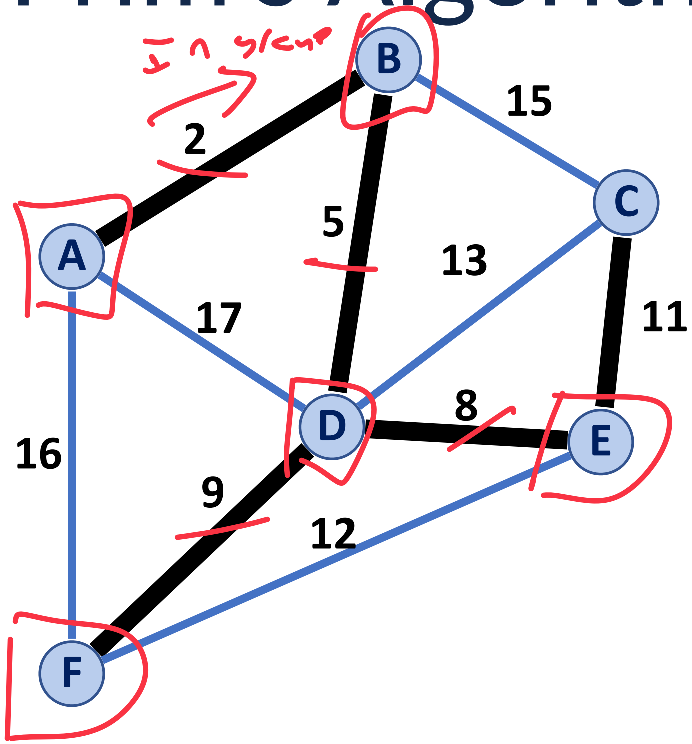
3) Repeatedly find min edge

If edge connects two sets

Union and record edge

4) Stop after n-1 edges recorded

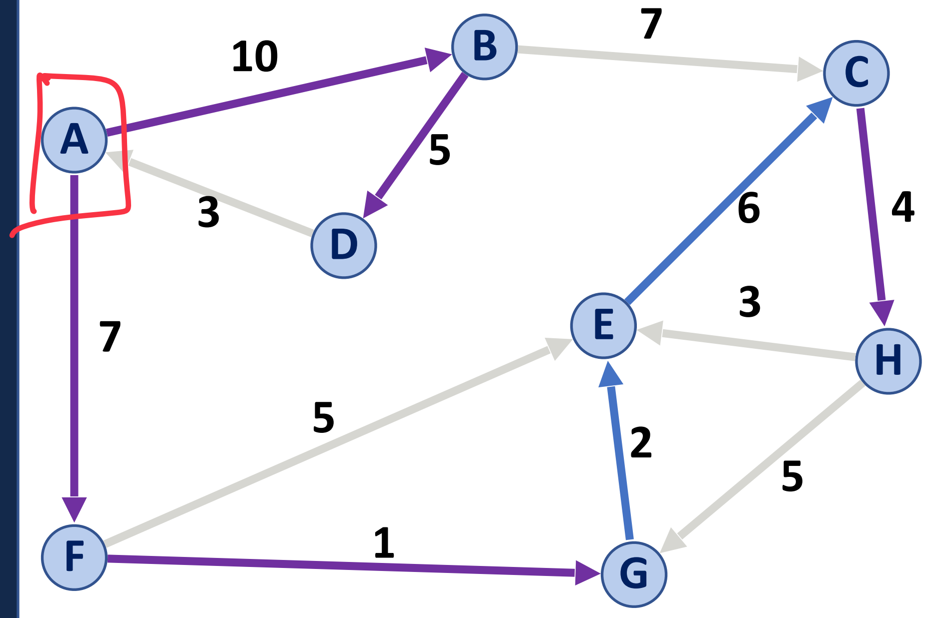
Prim's Algorithm



A	B	C	D	E	F
0, —	2, A	11, E	5, B	8, D	9, D

```
PrimMST(G, s):
  Input: G, Graph;
         s, vertex in G, starting vertex
  Output: T, a minimum spanning tree (MST) of G

  foreach (Vertex v : G.vertices()):
    d[v] = +inf
    p[v] = NULL
  d[s] = 0

  PriorityQueue Q   // min distance, defined by d[v]
  Q.buildHeap(G.vertices())
  Graph T           // "labeled set"

  repeat n times:
    Vertex m = Q.removeMin()
    T.add(m)
    foreach (Vertex v : neighbors of m not in T):
      if cost(v, m) < d[v]:
        d[v] = cost(v, m)
        p[v] = m

  return T
```

Dijkstra's Algorithm (SSSP)




```
DijkstraSSSP(G, s):
  foreach (Vertex v : G.vertices()):
    d[v] = +inf
    p[v] = NULL
  d[s] = 0

  PriorityQueue Q // min distance, defined by d[v]
  Q.buildHeap(G.vertices())
  Graph T          // "labeled set"

  repeat n times:
    Vertex u = Q.removeMin()
    T.add(u)
    foreach (Vertex v : neighbors of u not in T):
      if cost(u, v) + d[u] < d[v]:
        d[v] = cost(u, v) + d[u]
        p[v] = u
```

A	B	C	D	E	F	G	H
--	A	E	B	G	A	F	C
0	10	16	15	10	7	8	20

Floyd-Warshall Algorithm

|V| = n

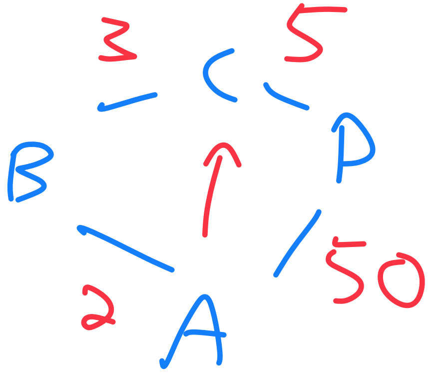
Floyd-Warshall's Algorithm is an alternative to Dijkstra in the presence of negative-weight edges (not negative weight cycles).

```
FloydWarshall(G):
  Let d be a adj. matrix initialized to +inf
  foreach (Vertex v : G):
    d[v][v] = 0
  foreach (Edge (u, v) : G):
    d[u][v] = cost(u, v)

  foreach (Vertex u : G):
    foreach (Vertex v : G):
      foreach (Vertex w : G):
        if (d[u, v] > d[u, w] + d[w, v])
          d[u, v] = d[u, w] + d[w, v]
```

n^3

u - w - v

start mid stop



A→B→C

A → C = 5

A → C → D

A → D

A Hash Table based Dictionary

User Code (is a map):

```
Dictionary<KeyType, ValueType> d;
d[k] = v;
```

A **Hash Table** consists of three things:

1. A hash function

2. A data storage structure

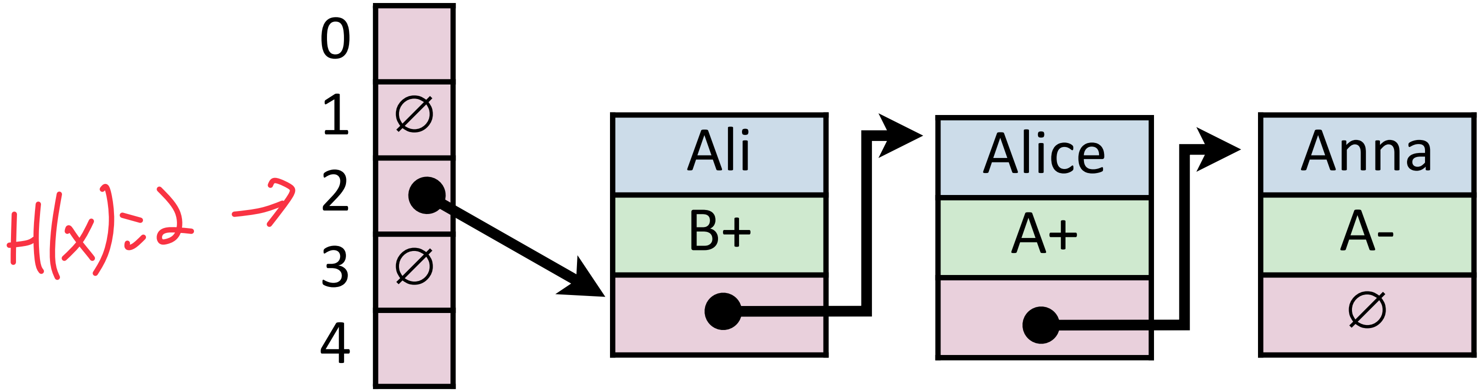
3. A method of addressing *hash collisions*

Open vs Closed Hashing

Addressing hash collisions depends on your storage structure.

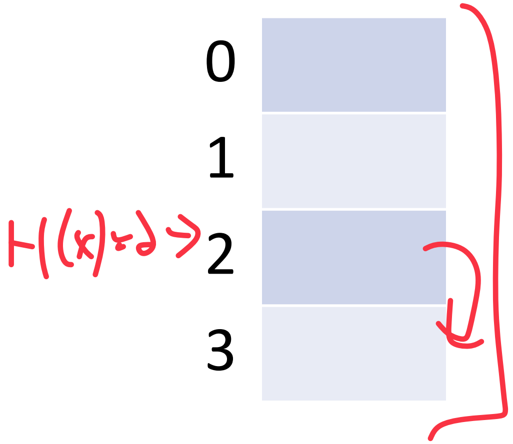
- **Open Hashing:** store *k,v* pairs externally

↳ closed addressing



- **Closed Hashing:** store *k,v* pairs in the hash table

↳ open addressing



Separate Chaining Under SUHA

Claim: Under SUHA, expected length of chain is $\frac{n}{m}$

Table Size: m

Num objects: n

$\alpha_j =$ expected # of items hashing to position j

$$\alpha_j = \sum_i H_{i,j}$$

$$H_{i,j} = \begin{cases} 1 \text{ if item i hashes to j} \\ 0 \text{ otherwise} \end{cases}$$

$$E[\alpha_j] = E[\sum_i H_{i,j}]$$

$$Pr[H_{i,j} = 1] = \frac{1}{m}$$

$$E[\alpha_j] = n * Pr(H_{i,j} = 1)$$

$$\mathbf{E[\alpha_j] = \frac{n}{m}}$$

*** Load Factor

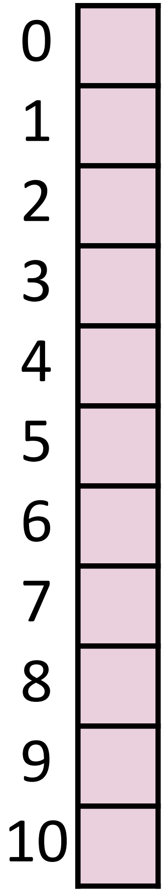
Separate Chaining Under SUHA

Under SUHA, a hash table of size *m* and *n* elements:

Find runs in: $O(1 + \alpha)$

Insert runs in: $O(1)$

Remove runs in: $O(1 + \alpha)$



Running Times *(Don't memorize these equations, no need.)*

The expected number of probes for find(key) under SUHA

$\alpha < 1$

Linear Probing:

- Successful: **$\frac{1}{2}(1 + 1/(1-\alpha))$**
- Unsuccessful: **$\frac{1}{2}(1 + 1/(1-\alpha))^2$**

Double Hashing:

- Successful: **$1/\alpha * \ln(1/(1-\alpha))$**
- Unsuccessful: **$1/(1-\alpha)$**

Separate Chaining:

- Successful: **$1 + \alpha/2$**
- Unsuccessful: **$1 + \alpha$**

$\alpha \to \infty$

Instead, observe:

- As α increases:

Runtime approaches infinity!

- If α is constant:

Runtime is a constant!

Resizing a hash table

When and how do you resize?

when $\alpha = 0.7 - 0.9$

$k \% m$

Hash has changed!

Any (review) questions?

Memory-Constrained Data Structures

What method would you use to build a search index on a collection of objects *in a memory-constrained environment*?

Constrained by Big Data (Large N)

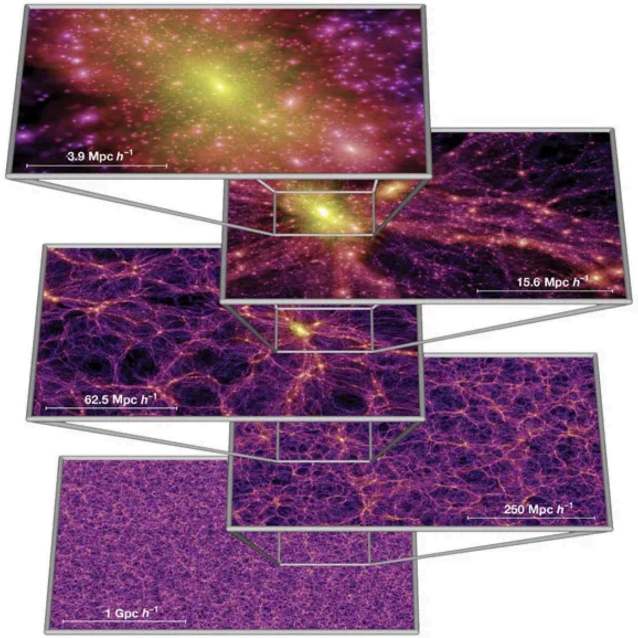


Image: https://doi.org/10.1038/nature03597

Sky Survey Projects	Data Volume
DPOSS (The Palomar Digital Sky Survey)	3 TB
2MASS (The Two Micron All-Sky Survey)	10 TB
GBT (Green Bank Telescope)	20 PB
GALEX (The Galaxy Evolution Explorer)	30 TB
SDSS (The Sloan Digital Sky Survey)	40 TB
SkyMapper Southern Sky Survey	500 TB
PanSTARRS (The Panoramic Survey Telescope and Rapid Response System)	~ 40 PB expected
LSST (The Large Synoptic Survey Telescope)	~ 200 PB expected
SKA (The Square Kilometer Array)	~ 4.6 EB expected

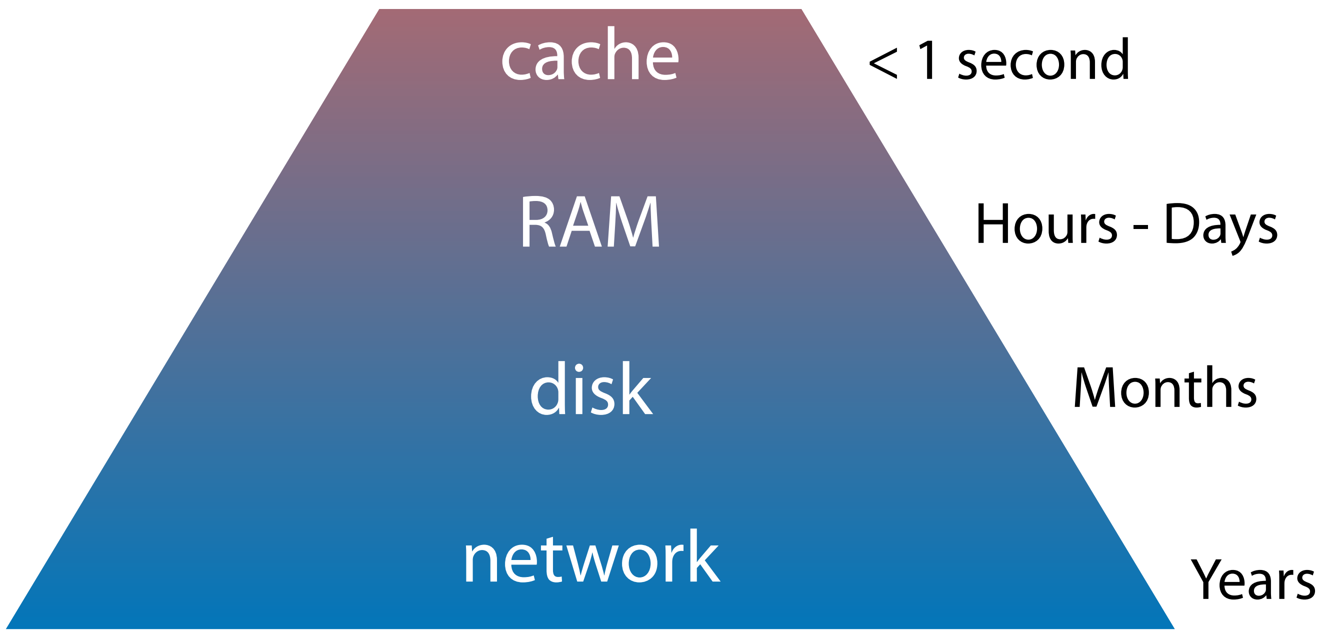
Table: http://doi.org/10.5334/dsj-2015-011

Estimated total volume of one array: 4.6 EB

Memory-Constrained Data Structures

What method would you use to build a search index on a collection of objects *in a memory-constrained environment*?

Constrained by resource limitations

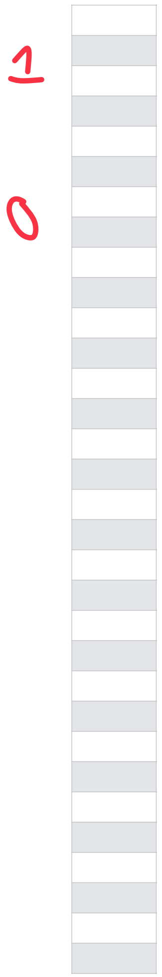


(Estimates are Time x 1 billion courtesy of https://gist.github.com/hellerbarde/2843375)

Bloom Filters

A probabilistic data structure storing a set of values

$h_{\{1,2,3,\ldots,k\}}$



Has three key properties:

- k, number of hash functions
- n, expected number of insertions
- m, filter size in bits

Expected false positive rate: $$\left(1-\left(1-\frac{1}{m}\right)^{nk}\right)^{k} \approx \left(1-e^{\frac{-nk}{m}}\right)^{k}$$

Optimal accuracy when: $$k^* = \ln 2 \cdot \frac{m}{n}$$

Bloom Filter Use Cases

Which of the following problems can be solved with a bloom filter?

A) Find the closest matching item to a query of interest ← X

✓ AVL tree

No hash is exact match only

B) Check if a query exists in a dataset ✓

the main use case of BF

C) Compare the similarity between two datasets Meh

↳ Yes but we will see better

↳ minHash

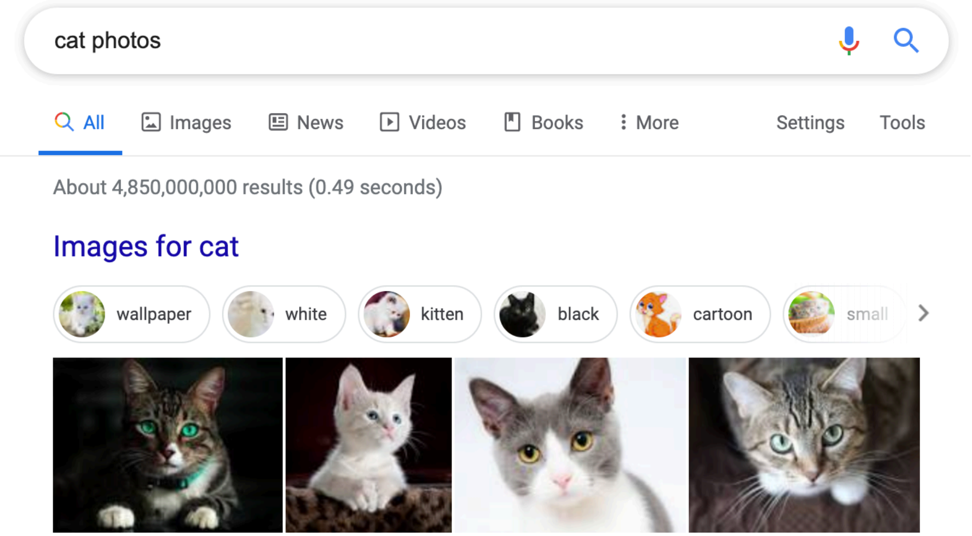
D) Count the number of unique items in a dataset Meh? Not very good

↳ Yes but with low accuracy

↳ Cardinality estimation

Cardinality

Sometimes its not possible or realistic to count all objects!



Estimate: 60 billion — 130 trillion

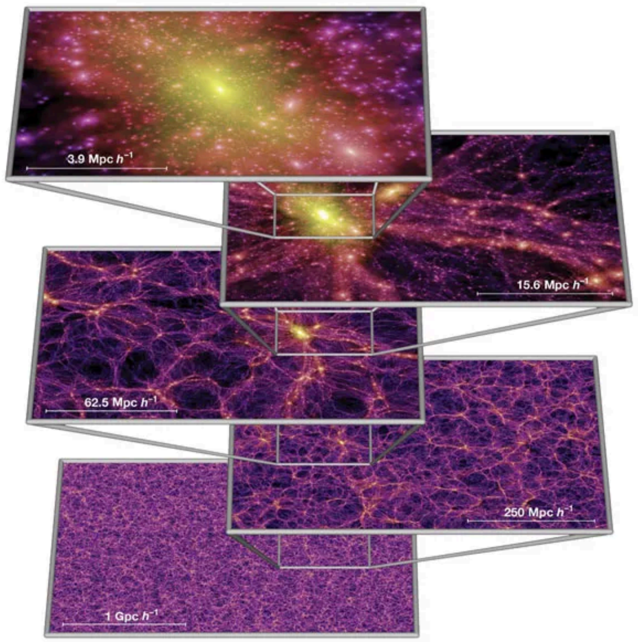


Image: https://doi.org/10.1038/nature03597

5581
8945
6145
8126
3887
8925
1246
8324
4549
9100
5598
8499
8970
3921
8575
4859
4960
42
6901
4336
9228
3317
399
6925
2660
2314

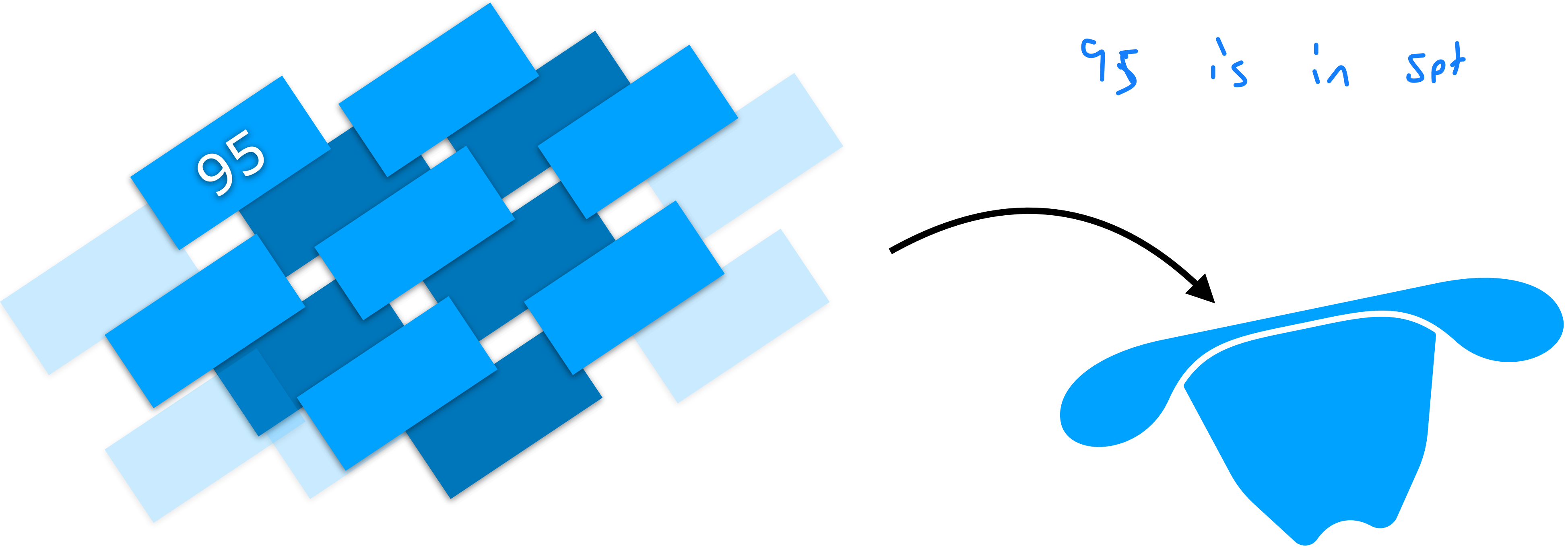
Cardinality Estimation

Imagine I fill a hat with numbered cards and draw one card out at random.

If I told you the value of the card was 95, what have we learned?

↳ very little

95 is in set



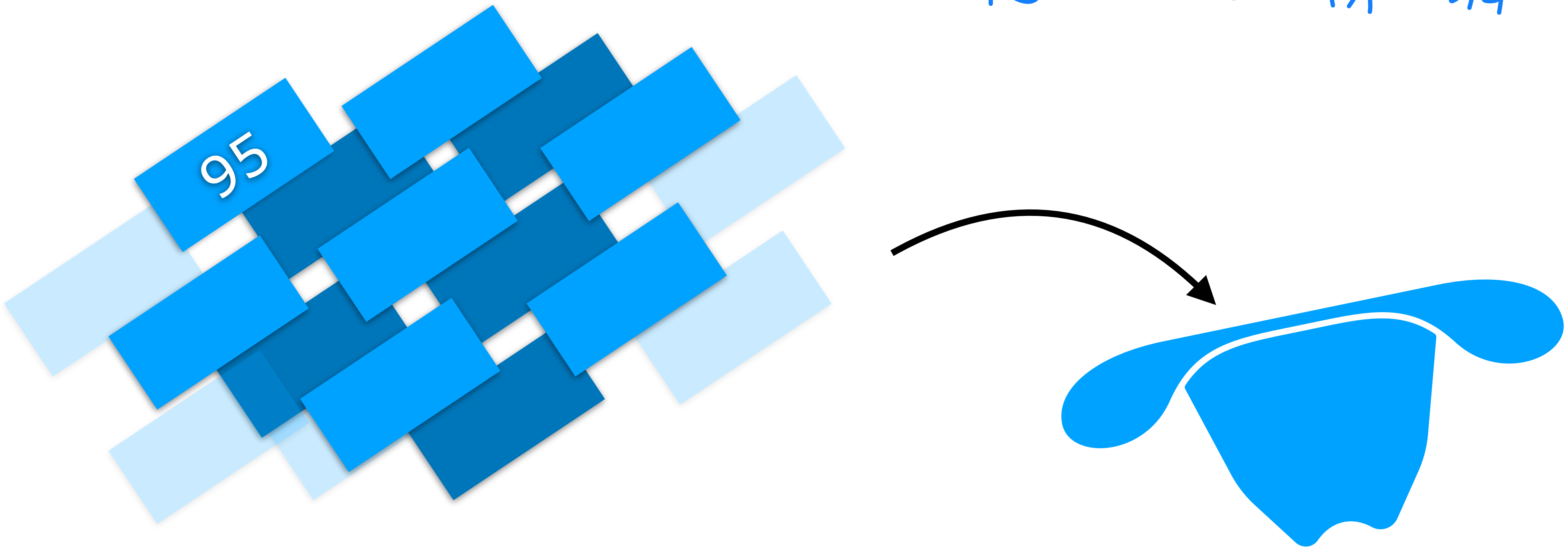
Analogy from Ben Langmead

Cardinality Estimation

Imagine I fill a hat with **a random subset** of numbered cards **from 0 to 999**

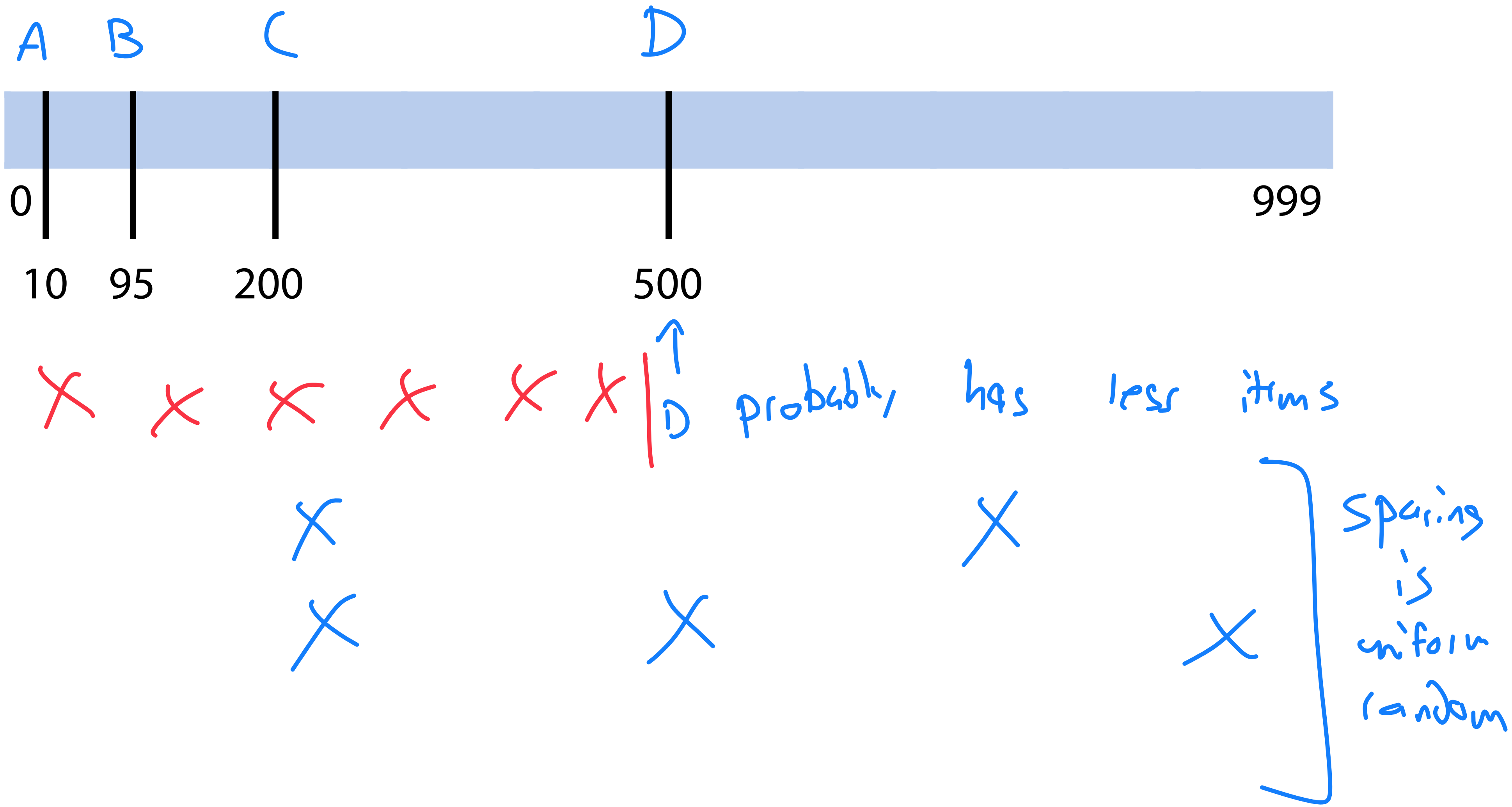
If I told you that the **minimum** value was 95, what have we learned?

~10 items in hat



Cardinality Estimation

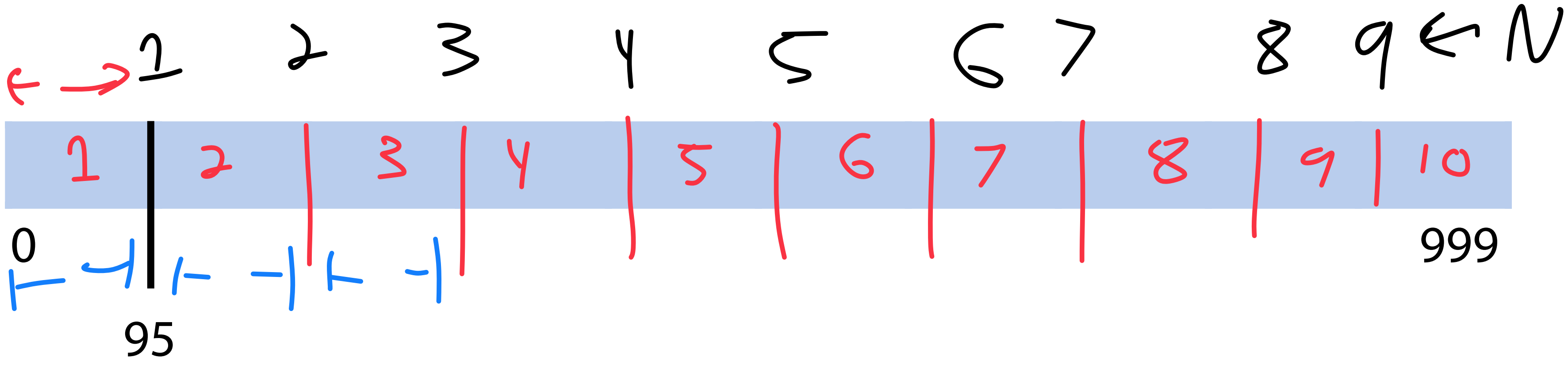
Imagine we have multiple uniform random sets with different minima.



Cardinality Estimation

Assume uniform random dist

Let min = 95. Can we estimate N, the cardinality of the set?



$$95 \approx \frac{1000}{N+1} \rightarrow N \approx 9.5$$

Cardinality Estimation

Let min = 95. Can we estimate N, the cardinality of the set?

Claim: $95 \approx \dfrac{1000}{(N+1)}$

Cardinality Estimation

Let min = 95. Can we estimate N, the cardinality of the set?

Conceptually: If we scatter N points randomly across the interval, we end up with $N + 1$ partitions, each about $1000/(N+1)$ long

Assuming our first 'partition' is about average:

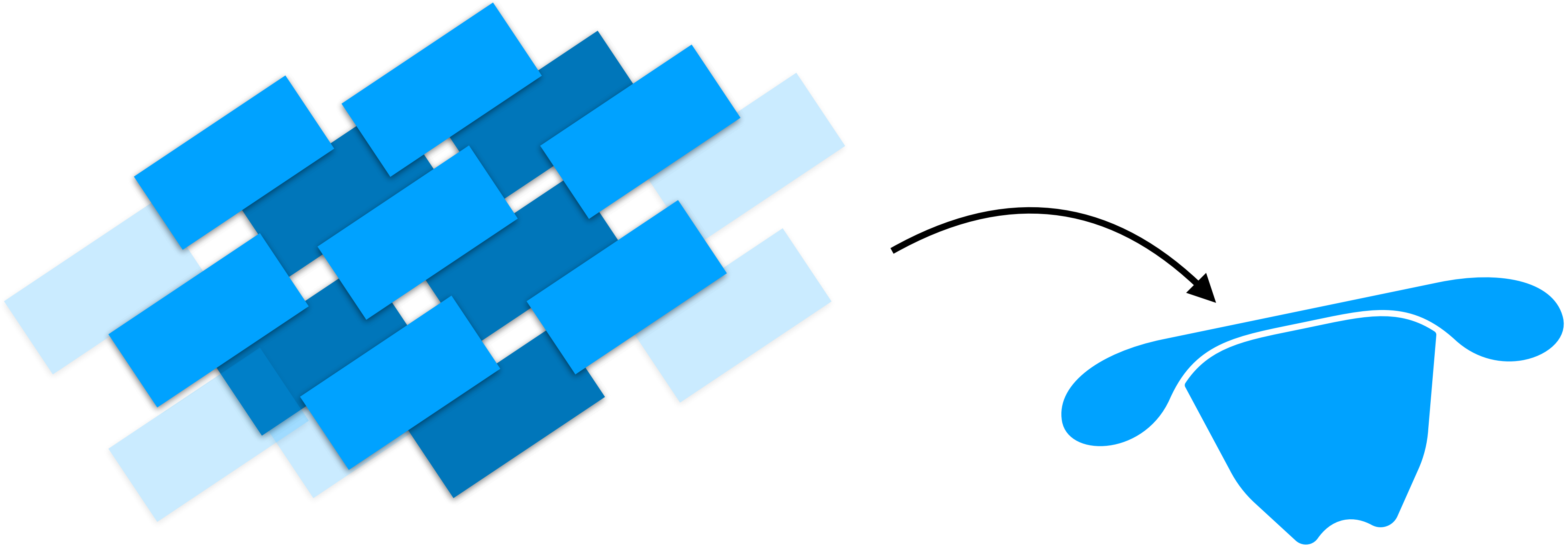
$$95 \approx 1000/(N+1)$$

$$N + 1 \approx 10.5$$

$$N \approx 9.5$$

Cardinality Estimation

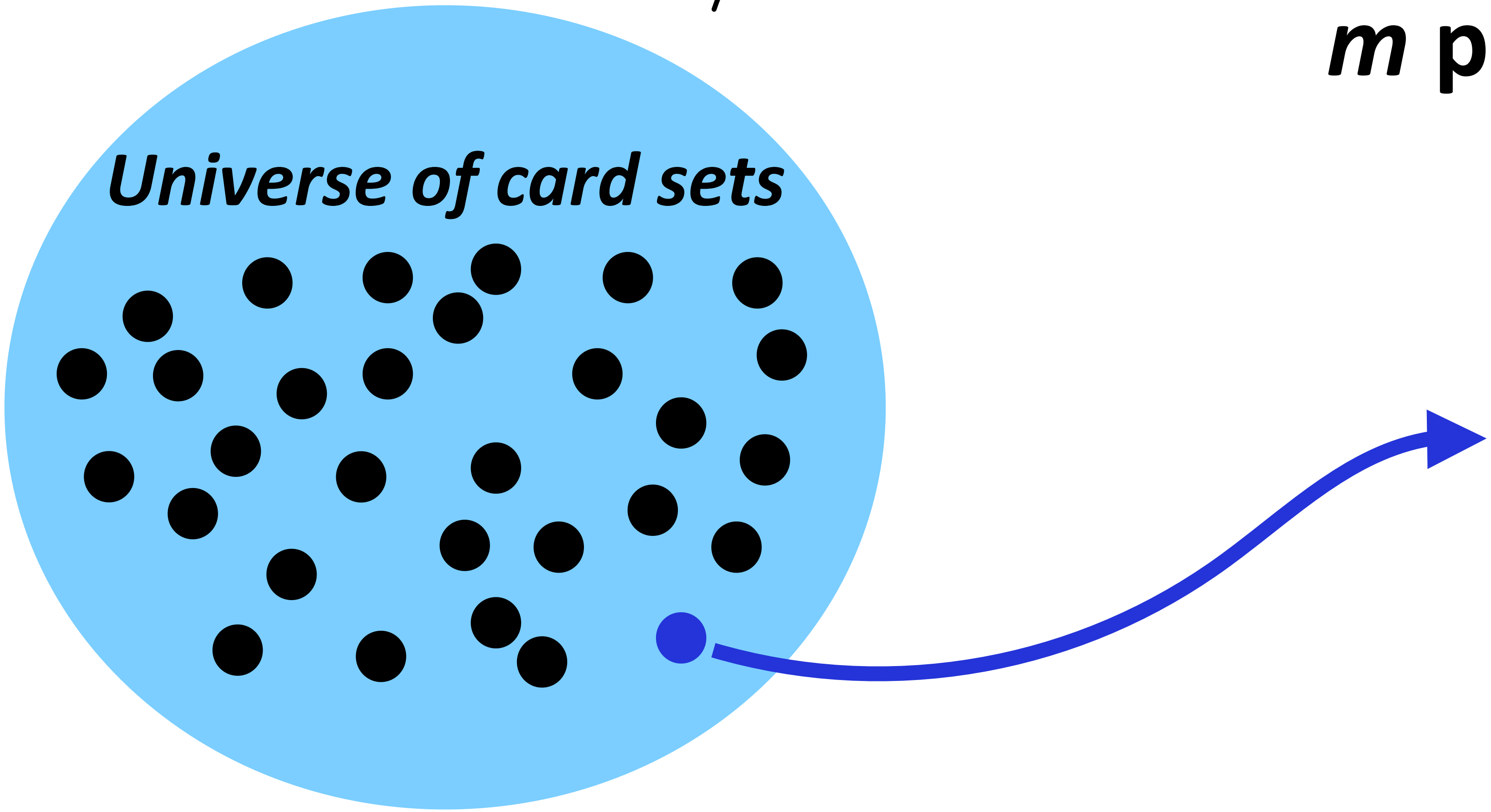
Why do we care about “the hat problem”?



Cardinality Estimation

Why do we care about "the hat problem"?

key



int (hash value)

***m* possible minima**

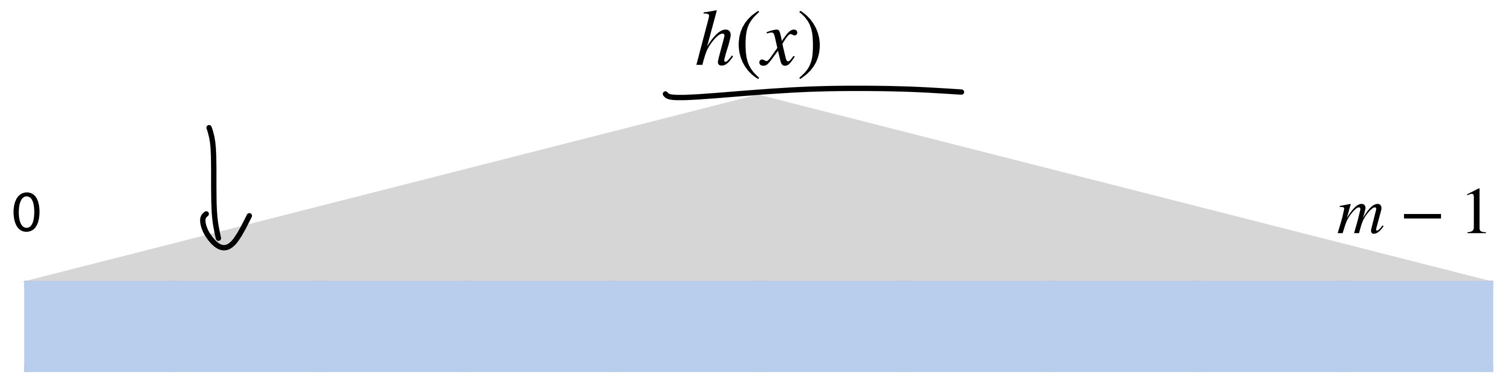
Key	Value

Cardinality Estimation

Imagine we have a SUHA hash h over a range m.

Inserting a new key is equivalent to adding a card to our hat!

Tracking only the minimum value is a **sketch** that estimates the cardinality!



Cardinality Estimation

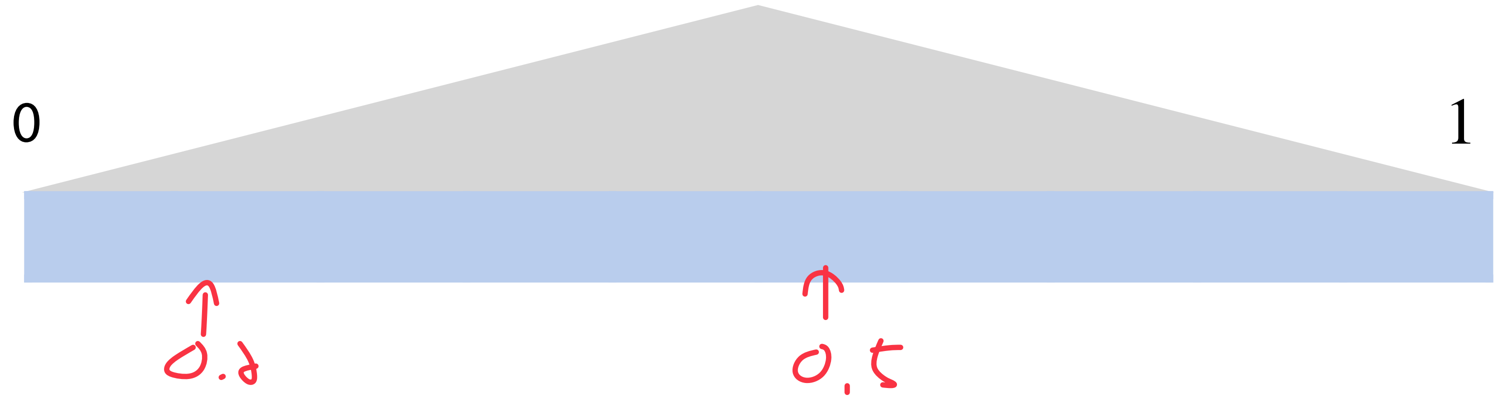
Imagine we have a SUHA hash h over a range m.

Inserting a new key is equivalent to adding a card to our hat!

Tracking only the minimum value is a **sketch** that estimates the cardinality!

To make the math work out, lets normalize our hash…

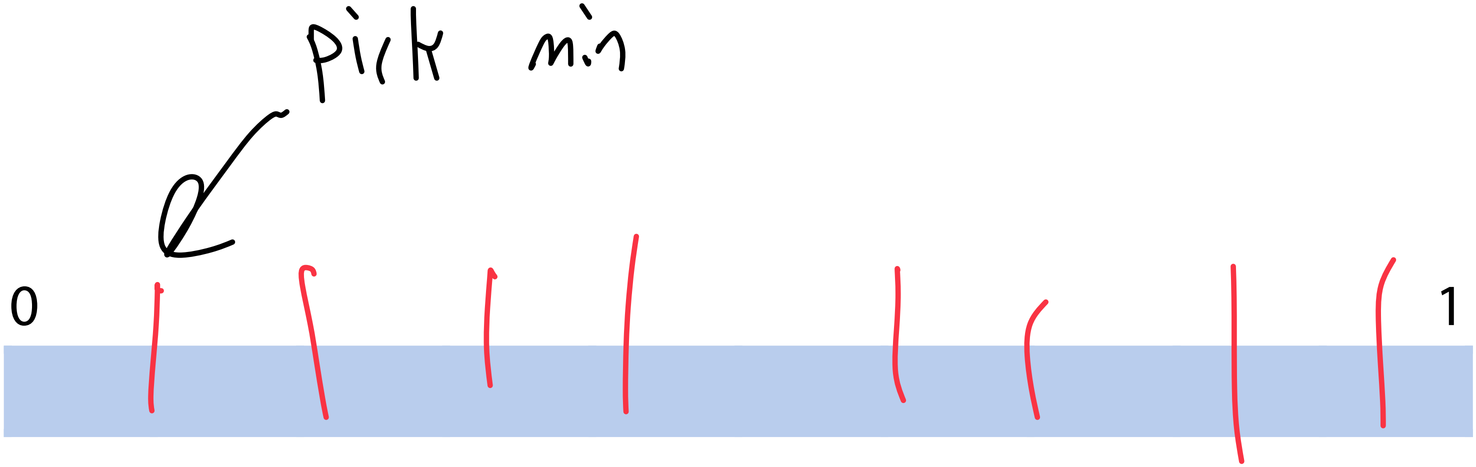
$$h'(x) = h(x) \ / \ (m-1)$$



Cardinality Sketch

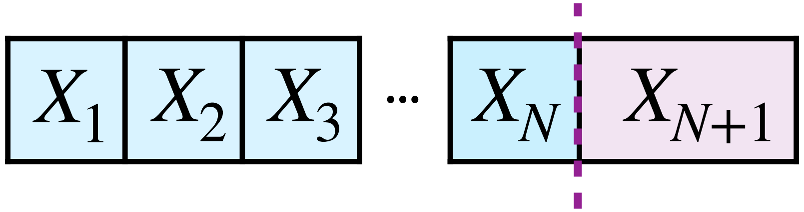
Let $M = min(X_1, X_2, \ldots, X_N)$ where each $X_i \in [0, 1]$ is an uniform independent random variable

Claim: $\mathbf{E}[M] = \dfrac{1}{N+1}$ (can get estimate for N using M)



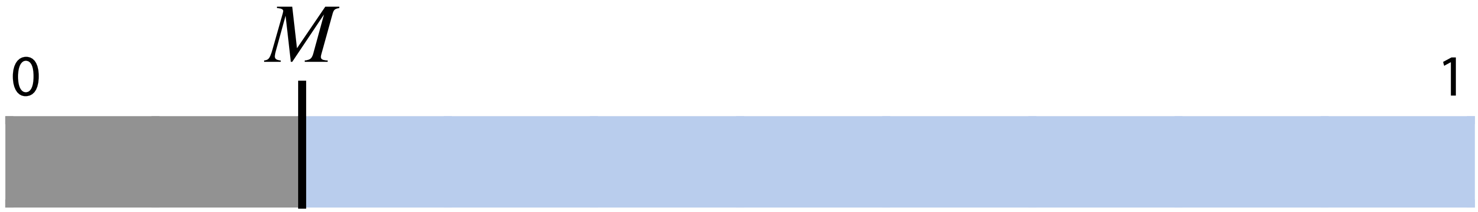
Cardinality Sketch

Consider an $N + 1$ draw:



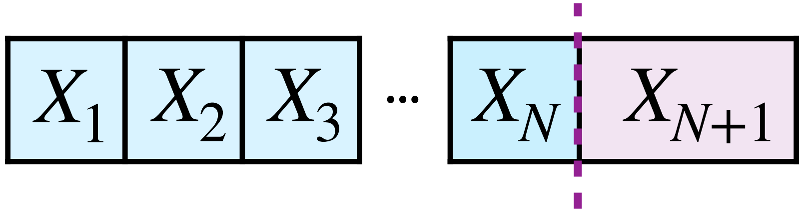
$$M = \min_{1 \leq i \leq N} X_i$$

X_{N+1} can end up in one of two ranges:



Cardinality Sketch

Consider an $N+1$ draw:

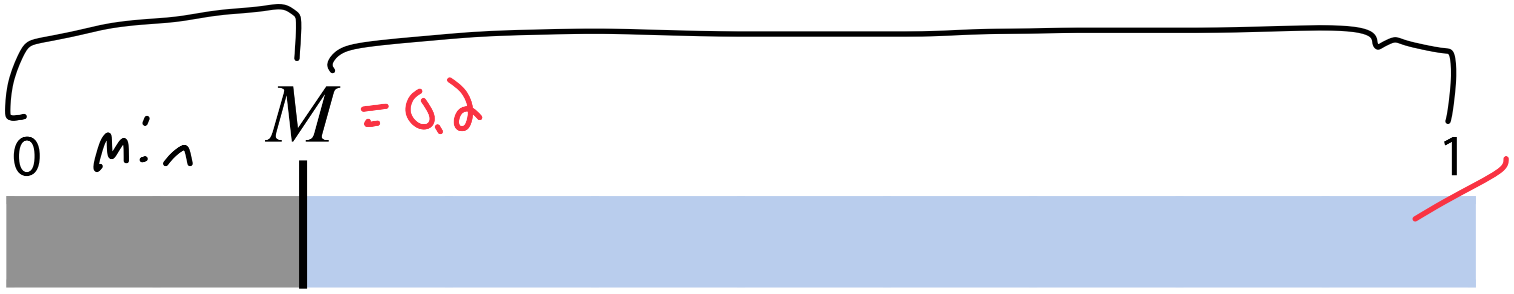


$$M = \min_{1 \leq i \leq N} X_i$$

X_{N+1} can end up in one of two ranges:

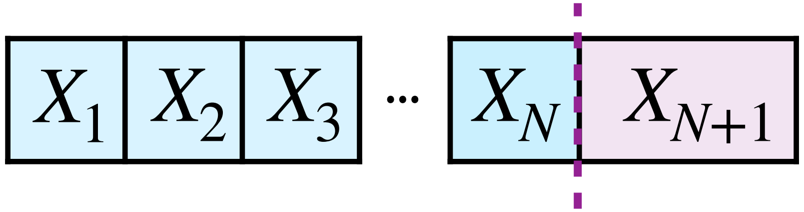
X_{N+1} will be the new minimum with probability M

↳ prob is size of range



Cardinality Sketch

Consider an $N + 1$ draw:

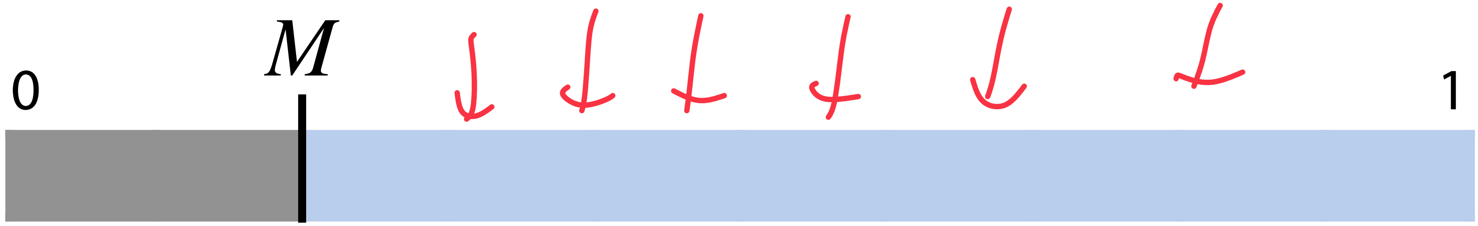


$$M = \min_{1 \leq i \leq N} X_i$$

X_{N+1} can end up in one of two ranges:

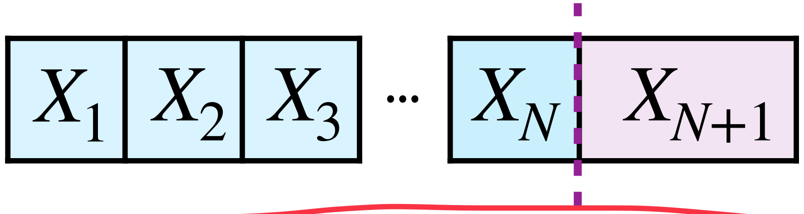
X_{N+1} will be the new minimum with probability M

X_{N+1} will not change minimum with probability $1 - M$



Cardinality Sketch

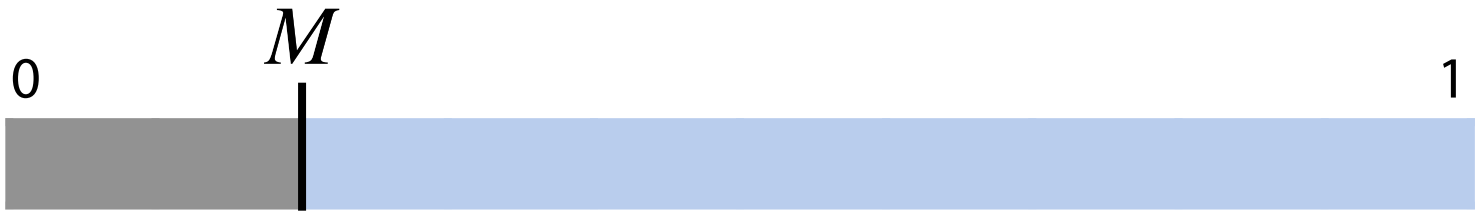
Consider an $N + 1$ draw:



$$M = \min_{1 \leq i \leq N} X_i$$

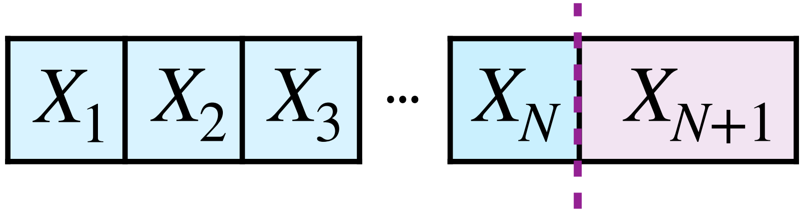
X_{N+1} **will be the new minimum with probability** M

By definition of SUHA, X_{N+1} has a $\dfrac{1}{N+1}$ chance of being smallest item



Cardinality Sketch

Consider an $N+1$ draw:

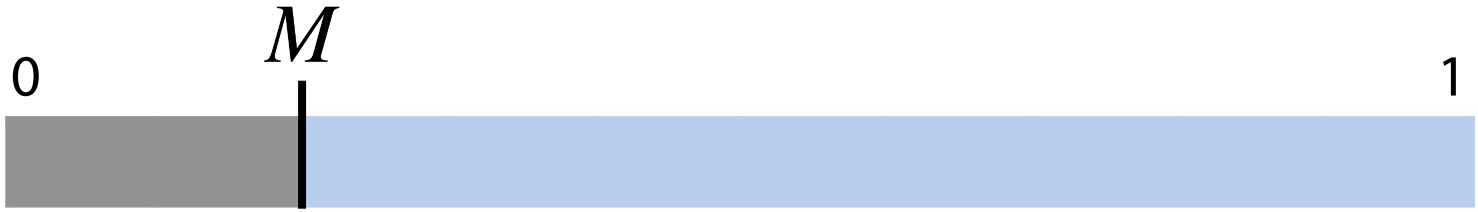


$$M = \min_{1 \le i \le N} X_i$$

X_{N+1} **will be the new minimum with probability** M

By definition of SUHA, X_{N+1} has a $\dfrac{1}{N+1}$ chance of being smallest item

Thus, $\mathbf{E}[M] = \dfrac{1}{N+1}$



Cardinality Sketch

Claim: $\mathbf{E}[M] = \frac{1}{N+1}$ $N \approx \frac{1}{M} - 1$

True N: 5

Attempt 1	0.962	0.328	0.771	0.952	0.923	N = 2.05
Attempt 2	0.253	0.839	0.327	0.655	0.491	N = 2.953
Attempt 3	0.134	0.580	0.364	0.743	0.931	N = 6.5

Cardinality Sketch

The minimum hash is a valid sketch of a dataset but can we do better?

↳ Random values are random!

$|A| = 1$ $X = 0.1$

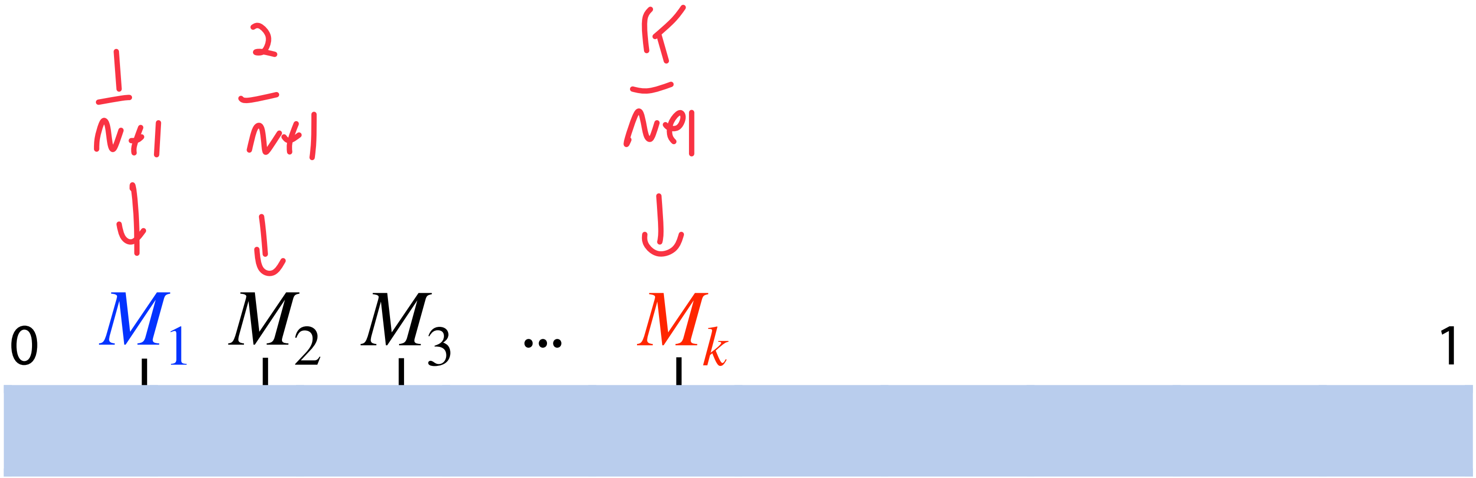
$|B| = 10$

0.4

Cardinality Sketch

Claim: Taking the k^{th}-smallest hash value is a better sketch!

Claim: $\mathbf{E}[M_k] = \dfrac{k}{N+1}$



Cardinality Sketch

Claim: Taking the k^{th}-smallest hash value is a better sketch!

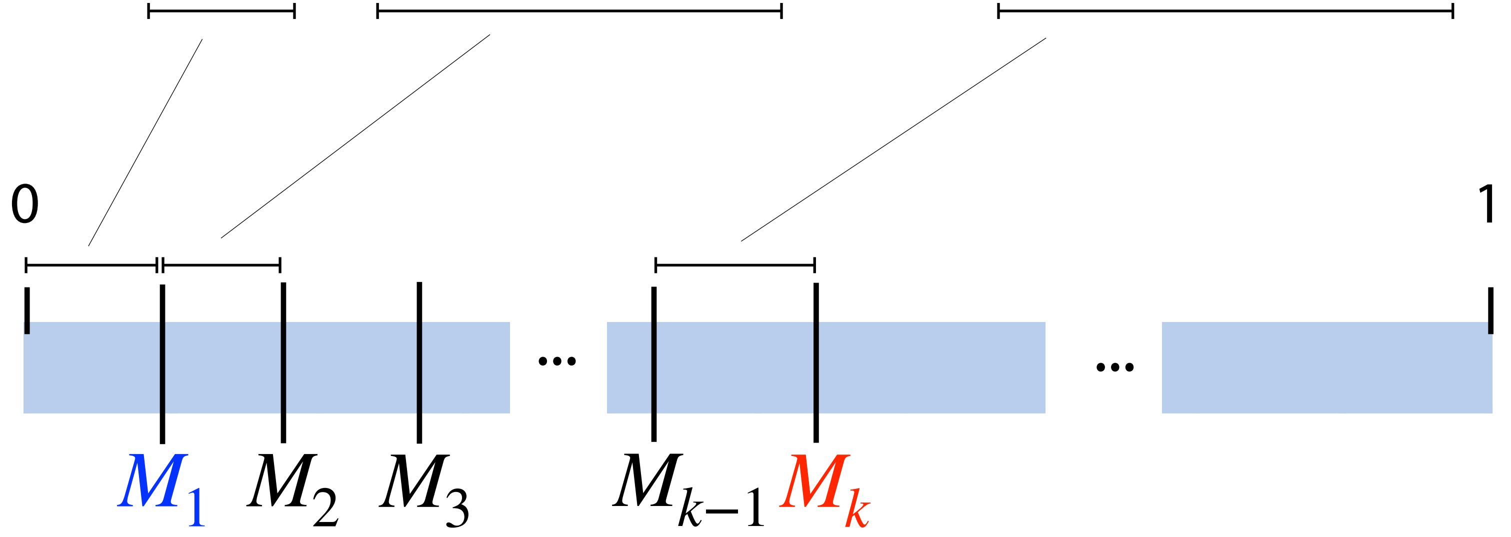
Claim: $\dfrac{\mathbf{E}[M_k]}{k} = \dfrac{1}{N+1}$

$$= \left[\mathbf{E}[M_1] + (\mathbf{E}[M_2] - \mathbf{E}[M_1]) + \ldots + (\mathbf{E}[M_k] - \mathbf{E}[M_{k-1}])\right] \cdot \frac{1}{k}$$

Cardinality Sketch

$$\frac{1}{N+1} = \frac{\mathbf{E}[M_k]}{k}$$

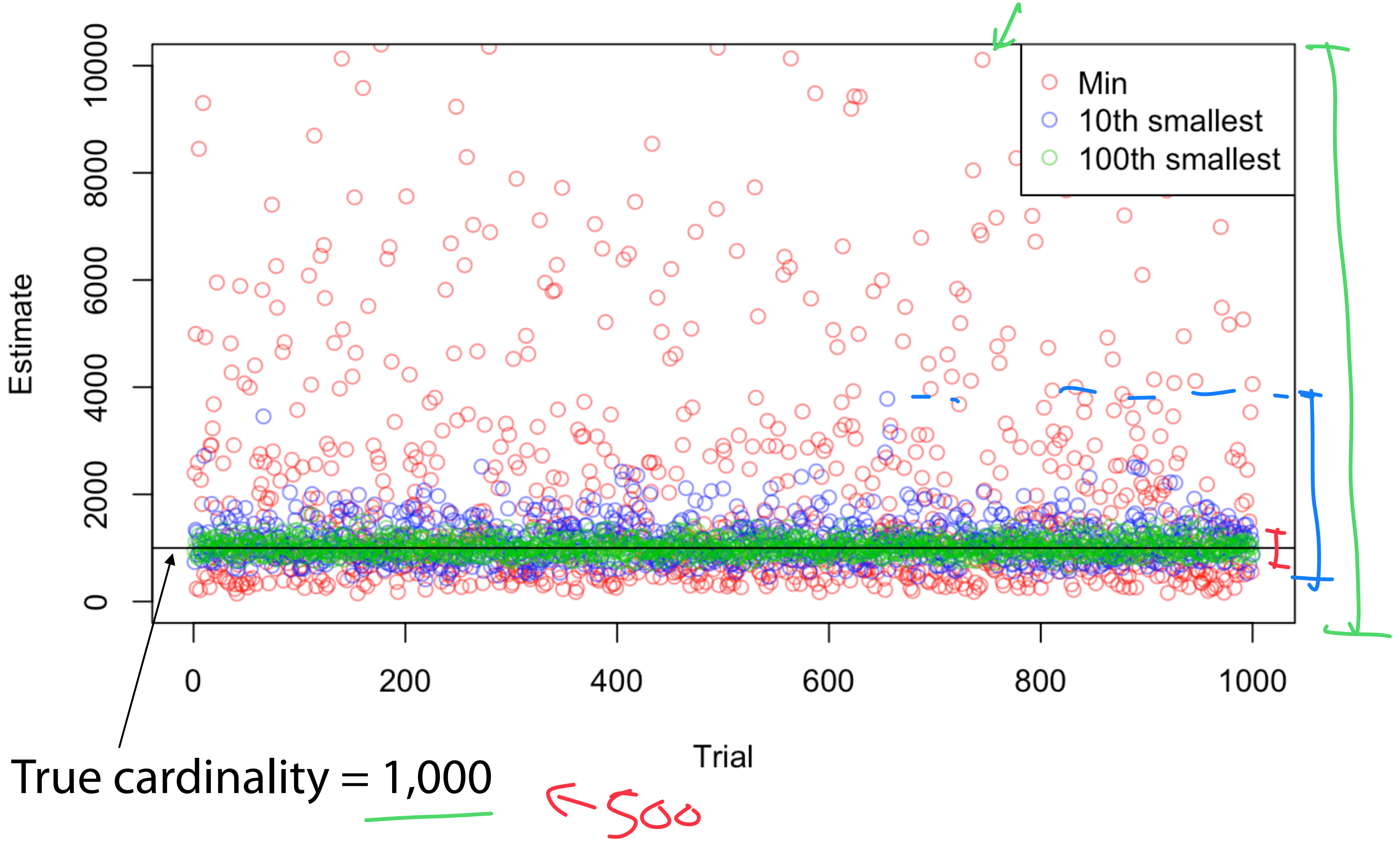
$$= \left[\mathbf{E}[M_1] + (\mathbf{E}[M_2] - \mathbf{E}[M_1]) + \ldots + (\mathbf{E}[M_k] - \mathbf{E}[M_{k-1}])\right] \cdot \frac{1}{k}$$



k^{th} minimum value (KMV)

Averages k estimates for $\frac{1}{N+1}$

Cardinality Sketch



True cardinality = 1,000

Cardinality Sketch

Given any dataset and a SUHA hash function, we can **estimate the number of unique items** by tracking the **k-th minimum hash value**.

Hash →

0.253	0.839	0.327	0.655	0.491

Tradeoff!

To use the k-th min, we have to track k minima. **Can we use ALL minima?**

All min is just storing all values!

Applied Cardinalities

Cardinalities → **Set similarities** → **Real-world Meaning**

$|A|$

$|B|$

$|A \cup B|$

$|A \cap B|$

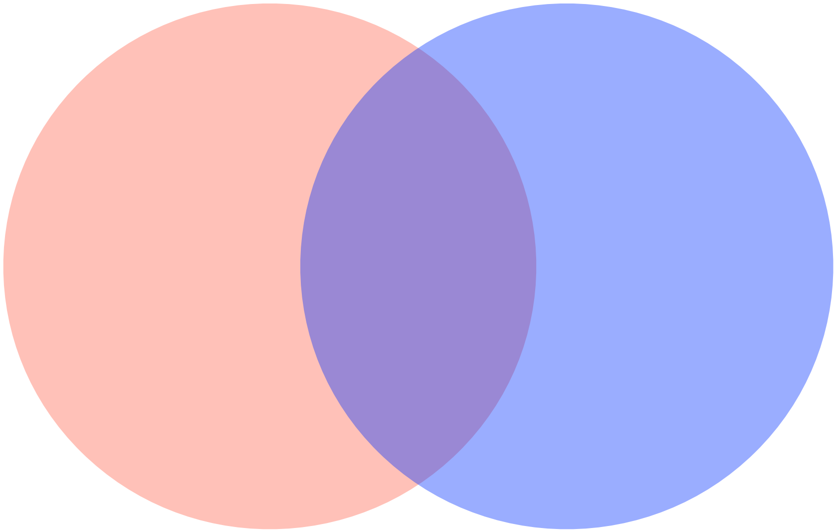
$$O = \frac{|A \cap B|}{min(|A|, |B|)}$$

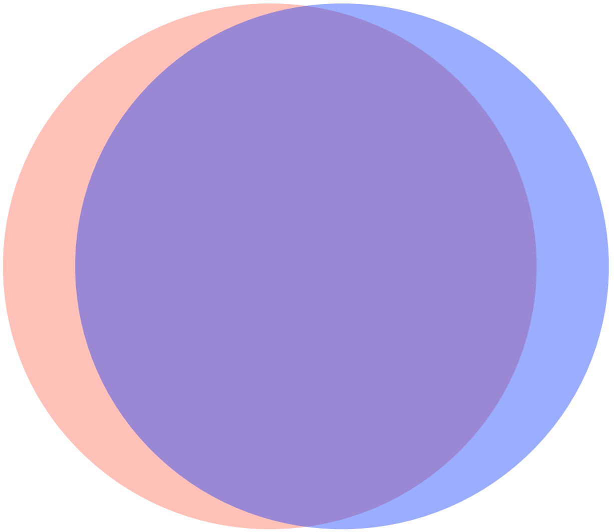
$$J = \frac{|A \cap B|}{|A \cup B|}$$

```
AGGCCACAGTGTATTATGACTG
|||||||||||  |||||||||
AGGCCACAGTGAGTTATGACTG

AAAAAAAAAAAGATGT-AAGTA
|||||||||||||||| |||||
AAAAAAAAAAAGATGTAAAGTA

GAGG--TCAGATTCACAGCCAC
||||  ||||||||||||||||
GAGGGGTCAGATTCACAGCCAC
```

Set Similarity Review

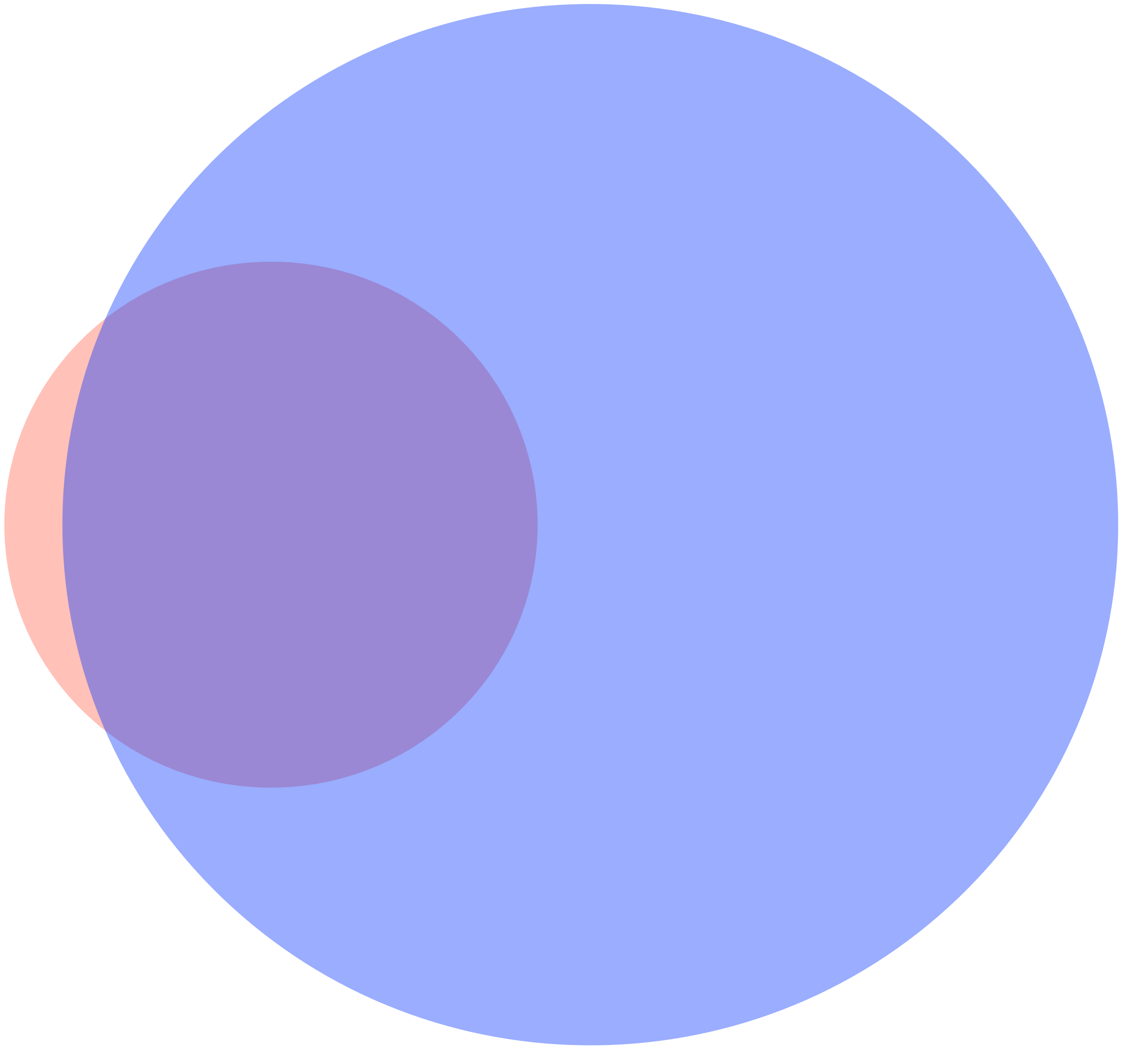
How can we describe how ***similar*** two sets are?

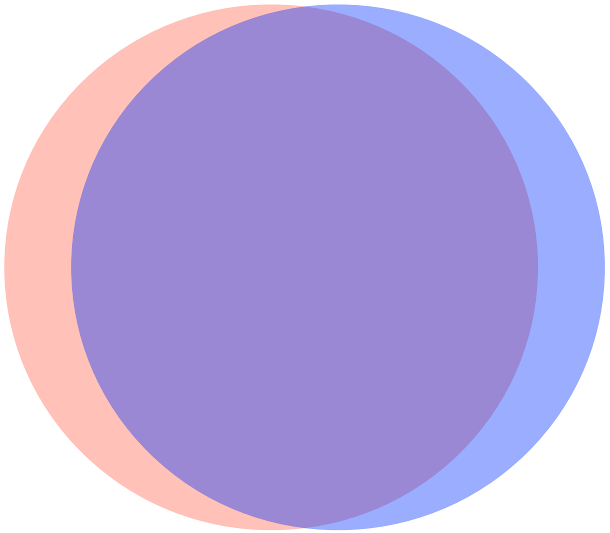




Set Similarity Review

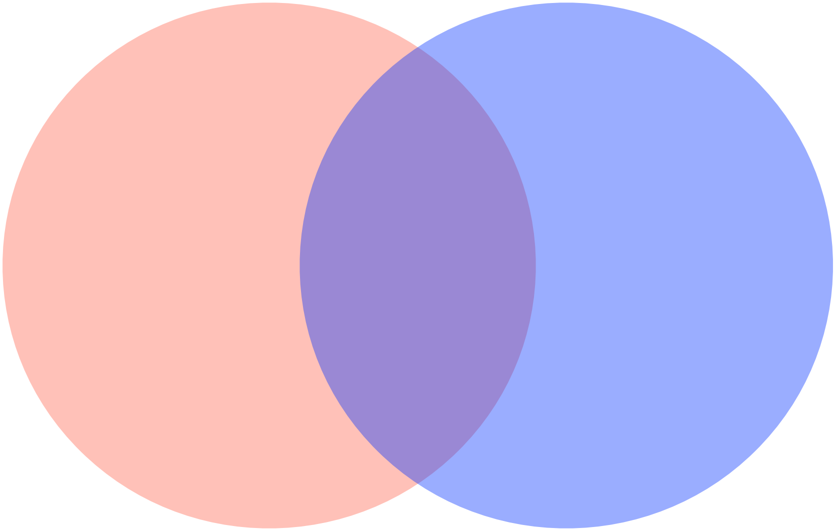
How can we describe how ***similar*** two sets are?





Set Similarity Review

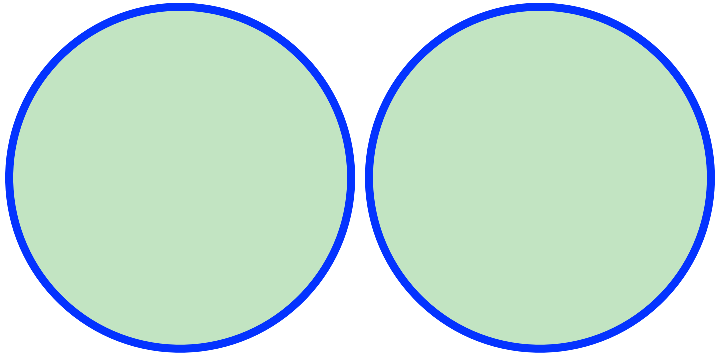
To measure **similarity** of A & B, we need both a measure of how similar the sets are but also the total size of both sets.



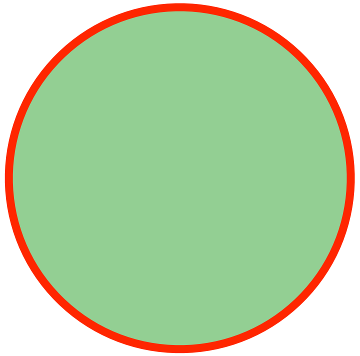
$$J = \frac{|A \cap B|}{|A \cup B|}$$

J is the ***Jaccard coefficient***

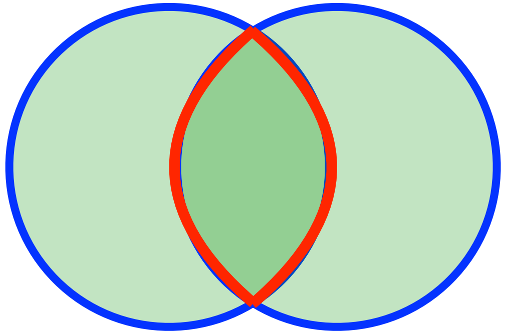
Set Similarity Review



$$\frac{|A \cap B|}{|A \cup B|} = 0$$



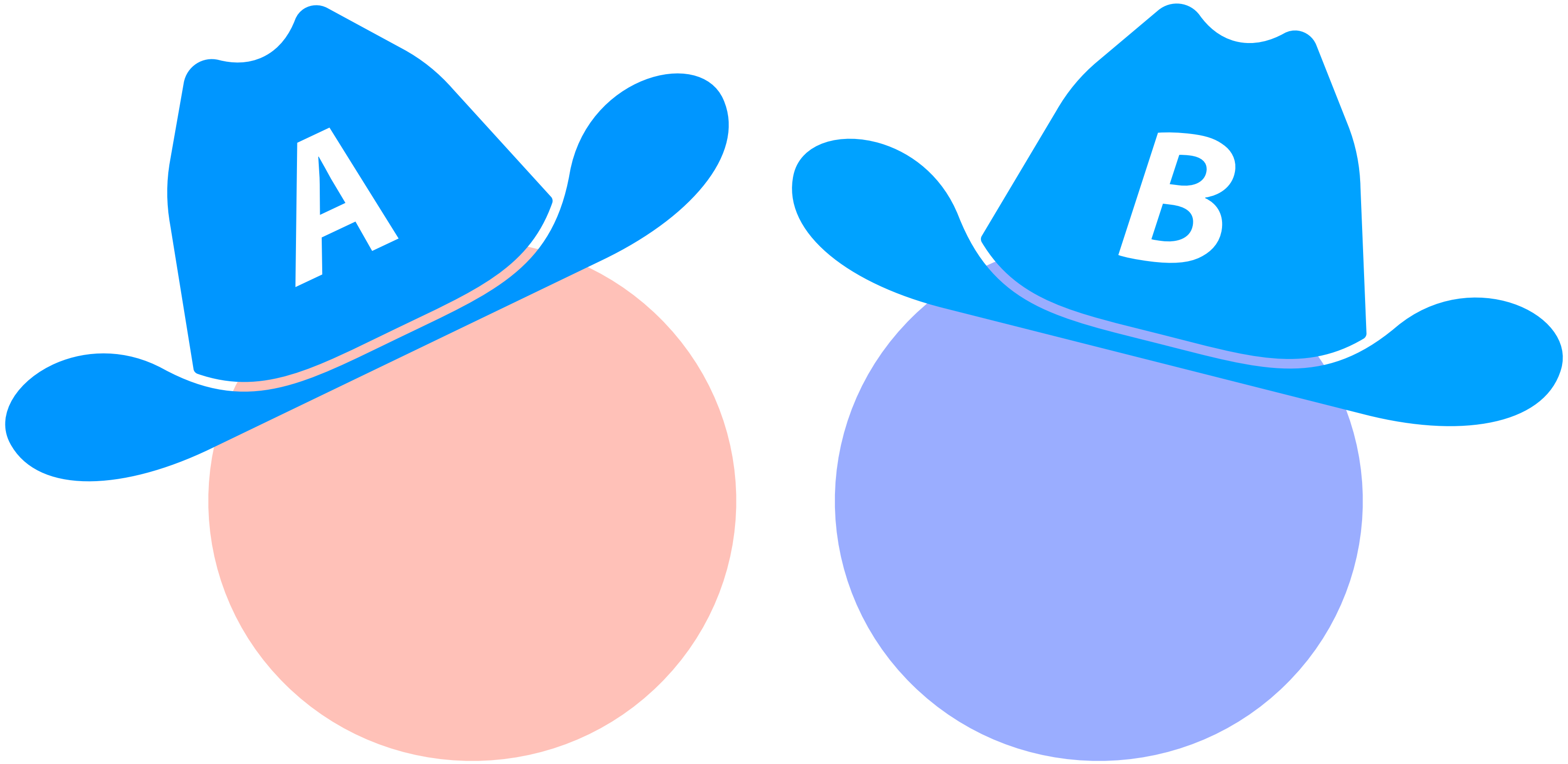
$$\frac{|A \cap B|}{|A \cup B|} = 1$$



$$0 < \frac{|A \cap B|}{|A \cup B|} < 1$$

Similarity Sketches

But what do we do when we only have a sketch?



Similarity Sketches

Imagine we have two datasets represented by their kth minimum values

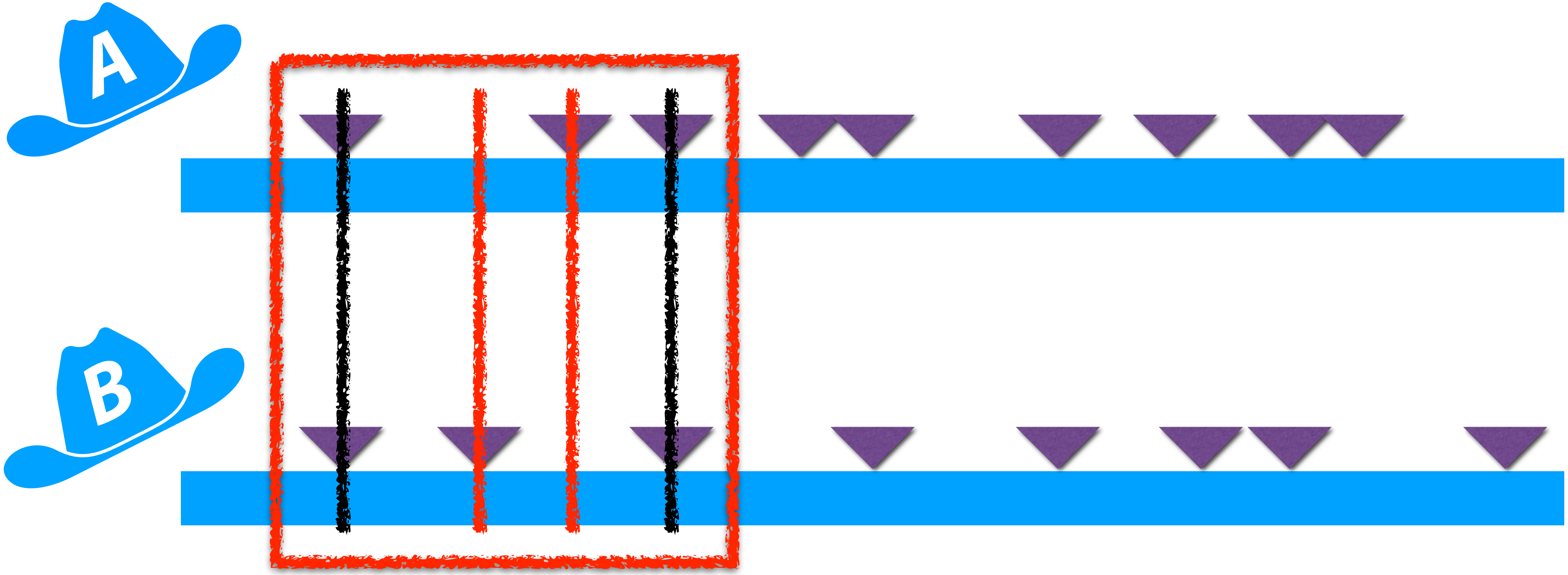


Image inspired by: Ondov B, Starrett G, Sappington A, Kostic A, Koren S, Buck CB, Phillippy AM. **Mash Screen: high-throughput sequence containment estimation for genome discovery**. *Genome Biol* 20, 232 (2019)

Similarity Sketches

Claim: Under SUHA, set similarity can be estimated by sketch similarity!

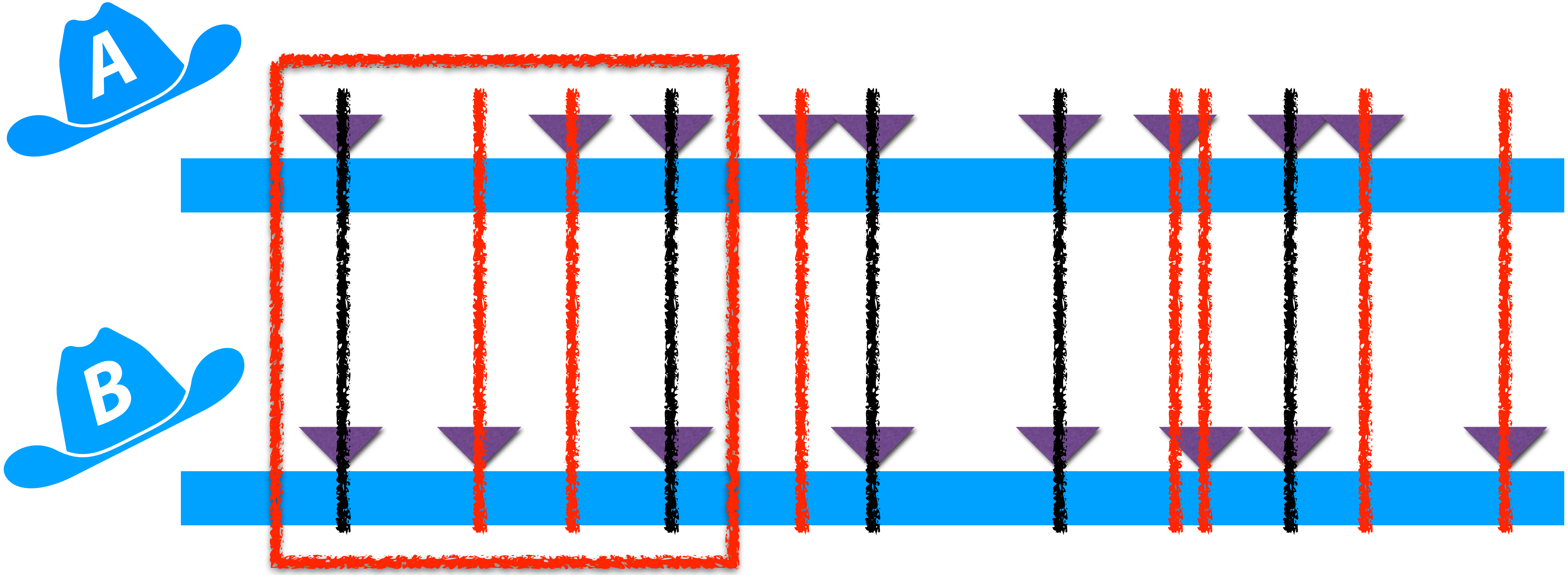


Image inspired by: Ondov B, Starrett G, Sappington A, Kostic A, Koren S, Buck CB, Phillippy AM. **Mash Screen: high-throughput sequence containment estimation for genome discovery**. *Genome Biol* 20, 232 (2019)