Randomized Algorithms

Streaming frequency estimation in one pass

Alex Gittens
Department of Computer Science, Rensselaer Polytechnic Institute

Fall 2026

Learning Objectives

By the end of this lecture, you should be able to:

1. **Model** insertion-only frequency streams and state the heavy-hitter and point-query tasks precisely.
2. **Run** Boyer–Moore and MisraGries by hand, and prove the MisraGries additive-error bound by counting cancellations.
3. **Run** CountMin by hand, and prove its point-query bound: expected collision mass, then Markov, then independent rows.
4. **Choose** parameters (k; w and d) from a target accuracy and confidence, and say what each algorithm needs to know about the stream length N.

Lecture Roadmap

The streaming problem

Cancellation: Misra–Gries

Hashing: Count–Min

Comparison

PART 1

The streaming problem

1

A router that cannot remember everything

A router sees the source addresses of packets

$$(a_1, a_2, \ldots, a_N) \in U^N,$$

one packet at a time. Two questions an operator might ask:

- **Who dominates the traffic?** Which sources sent more than 1% of all packets?
- **How active was this source?** An intrusion detector flags address x; roughly how many packets came from x?

A router that cannot remember everything

A router sees the source addresses of packets

$$(a_1, a_2, \ldots, a_N) \in U^N,$$

one packet at a time. Two questions an operator might ask:

- **Who dominates the traffic?** Which sources sent more than 1% of all packets?
- **How active was this source?** An intrusion detector flags address x; roughly how many packets came from x?

Both questions are about the **frequency vector**: one count per possible key,

$$f \in \mathbb{N}^U, \qquad f_x = \#\{t : a_t = x\}, \qquad \sum_{x \in U} f_x = N.$$

The stream is a sequence of N keys; f forgets the order and keeps, for each key, how often it appeared.

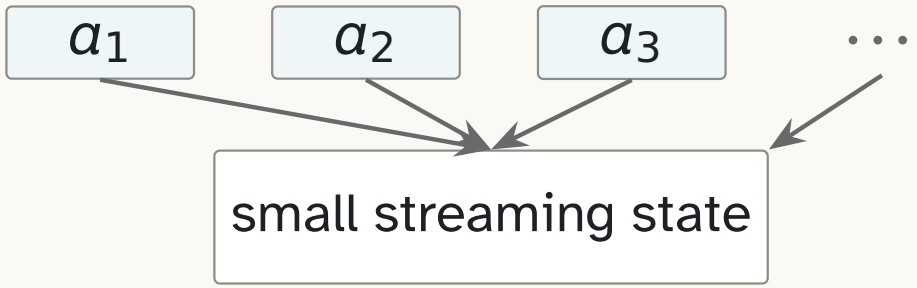
Can we keep the useful information in f without storing all of f?

Insertion-only streaming model

In an **insertion-only stream** every arriving item increases one coordinate of f by 1.

A streaming algorithm must:

- process a_t before seeing a_{t+1}, in one pass;
- use memory much smaller than the number of distinct keys;
- answer queries from its own state, not from the history.



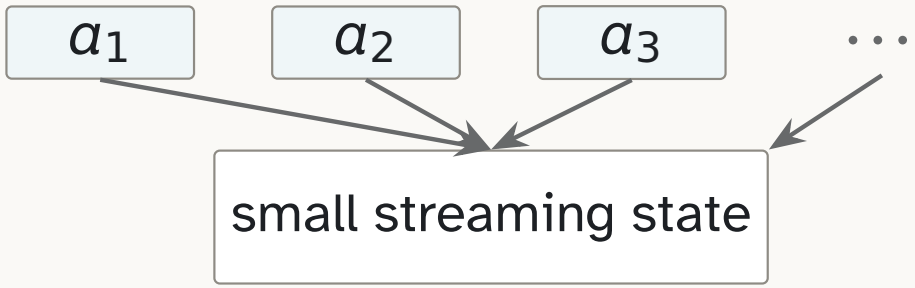
one pass
no rewind

Insertion-only streaming model

In an **insertion-only stream** every arriving item increases one coordinate of f by 1.

A streaming algorithm must:

- process a_t before seeing a_{t+1}, in one pass;
- use memory much smaller than the number of distinct keys;
- answer queries from its own state, not from the history.



one pass
no rewind

What about N? We do *not* assume the stream length is known in advance; the router cannot know how many packets will arrive. An algorithm may of course count arrivals as they come, at the cost of one extra counter. For each algorithm today we will ask whether it needs N at all.

Why not just keep an exact dictionary?

Let

$$D = \#\{x : f_x > 0\}$$

be the number of distinct keys that actually appear. An exact dictionary stores D key–counter pairs.

Nothing bounds D except N: in the worst case every packet comes from a new address, and $D = N$.

exact dictionary



D entries; grows with the stream

streaming summary



approximate counts only

So we give up exact counts. The question is which information we can throw away and still answer the two questions with a guarantee. We will see two answers: *cancel* occurrences of different keys against each other (deterministic), or *let keys share counters* chosen by hashing (randomized).

Point queries

Fix an accuracy $\epsilon > 0$.

Point query

Given a requested key x, output $\widehat{f}_x$ with

$$|\widehat{f}_x - f_x| \leq \epsilon N$$

(with probability at least $1 - \delta$, if the algorithm is randomized).

Point queries

Fix an accuracy $\epsilon > 0$.

Point query

Given a requested key x, output $\widehat{f}_x$ with

$$|\widehat{f}_x - f_x| \leq \epsilon N$$

(with probability at least $1 - \delta$, if the algorithm is randomized).

Why measure the error against N? Divide both sides by N:

$$\left| \frac{\widehat{f}_x}{N} - \frac{f_x}{N} \right| \leq \epsilon.$$

So a point query gives each key's *share of the traffic* to within $\pm\epsilon$. For the router: "x sent $3.2\% \pm 0.5\%$ of the packets."

Point queries

Fix an accuracy $\epsilon > 0$.

Point query

Given a requested key x, output $\widehat{f}_x$ with

$$|\widehat{f}_x - f_x| \leq \epsilon N$$

(with probability at least $1 - \delta$, if the algorithm is randomized).

Why measure the error against N? Divide both sides by N:

$$\left| \frac{\widehat{f}_x}{N} - \frac{f_x}{N} \right| \leq \epsilon.$$

So a point query gives each key's *share of the traffic* to within $\pm\epsilon$. For the router: "x sent $3.2\% \pm 0.5\%$ of the packets."

This does not depend on how long the stream is. Whether the router has seen 10^4 packets or 10^{12}, $\epsilon = 0.005$ means half a percentage point.

Later: Misra–Gries guarantees all keys at once, while Count–Min first guarantees one fixed key.

Heavy hitters and the gray zone

Fix a threshold $\phi > \epsilon$; for the router, $\phi = 1\%$.

Heavy hitters

Output a list of at most $O(1/\epsilon)$ keys such that

$f_x > \phi N \Rightarrow x$ is reported, $\qquad f_x \leq (\phi - \epsilon)N \Rightarrow x$ is not reported.

Keys with $(\phi - \epsilon)N < f_x \leq \phi N$ may go either way.

Heavy hitters and the gray zone

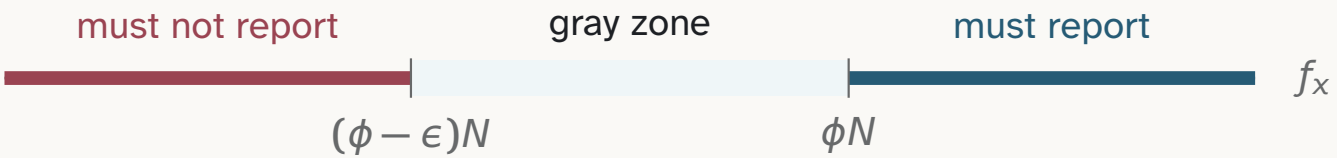
Fix a threshold $\phi > \epsilon$; for the router, $\phi = 1\%$.

Heavy hitters

Output a list of at most $O(1/\epsilon)$ keys such that

$f_x > \phi N \Rightarrow x$ is reported, $\quad f_x \leq (\phi - \epsilon)N \Rightarrow x$ is not reported.

Keys with $(\phi - \epsilon)N < f_x \leq \phi N$ may go either way.



Heavy hitters and the gray zone

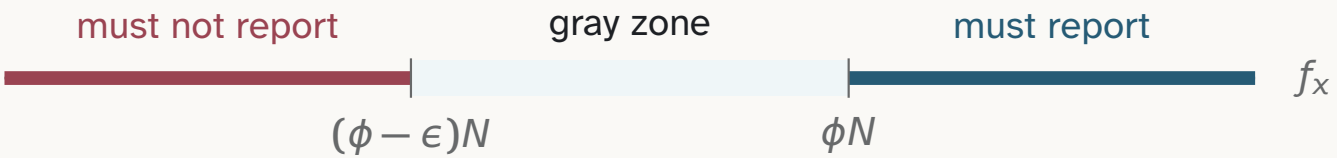
Fix a threshold $\phi > \epsilon$; for the router, $\phi = 1\%$.

Heavy hitters

Output a list of at most $O(1/\epsilon)$ keys such that

$f_x > \phi N \Rightarrow x$ is reported, $\quad f_x \leq (\phi - \epsilon)N \Rightarrow x$ is not reported.

Keys with $(\phi - \epsilon)N < f_x \leq \phi N$ may go either way.



Why allow a gray zone? Our counts will only be known approximately. Two keys whose counts differ by less than the estimation error can produce the same estimate, so no rule based on estimates can separate them. Misra–Gries will achieve a gray zone of width ϵN: with $\epsilon = 0.5\%$, report every source above 1% and none at or below 0.5%.

PART 2

Cancellation: Misra–Gries

2

A puzzle: finding a majority

A key m is a **strict majority** if $f_m > N/2$.

Puzzle. Suppose we are promised that a strict majority exists. Can you find it in one pass, storing *one* key and *one* counter?

Try it on a, b, a, c, a before going on.

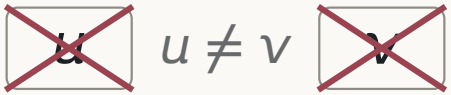
A puzzle: finding a majority

A key m is a **strict majority** if $f_m > N/2$.

Puzzle. Suppose we are promised that a strict majority exists. Can you find it in one pass, storing *one* key and *one* counter?

Try it on a, b, a, c, a before going on.

Observation. Delete one occurrence each of two *distinct* keys $u \neq v$.



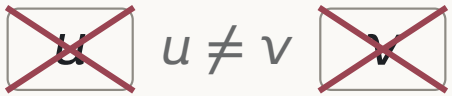
A puzzle: finding a majority

A key m is a **strict majority** if $f_m > N/2$.

Puzzle. Suppose we are promised that a strict majority exists. Can you find it in one pass, storing *one* key and *one* counter?

Try it on a, b, a, c, a before going on.

Observation. Delete one occurrence each of two *distinct* keys $u \neq v$.



At most one of the two deleted items is a copy of m, and the length drops by 2. If $f_m > N/2$, then

$$f_m - 1 > \frac{N}{2} - 1 = \frac{N-2}{2},$$

so m is still a strict majority of the shorter stream. (If neither deleted item is m, this is even easier.) So we may repeatedly delete pairs of distinct keys without losing the majority.

Boyer–Moore: cancelling pairs online

We never store the pairs. Keep a candidate key q (or $\varnothing$) and a counter $\mathbf{c}$, starting at $\mathbf{c} = 0$.

arriving key x	**update the state** $(q, \mathbf{c})$
$\mathbf{c} = 0$	$q \leftarrow x$, $\mathbf{c} \leftarrow 1$
$\mathbf{c} > 0$ and $x = q$	$\mathbf{c} \leftarrow \mathbf{c} + 1$
$\mathbf{c} > 0$ and $x \neq q$	$\mathbf{c} \leftarrow \mathbf{c} - 1$; if it becomes 0, set $q \leftarrow \varnothing$

Boyer–Moore: cancelling pairs online

We never store the pairs. Keep a candidate key q (or $\varnothing$) and a counter $\mathbf{c}$, starting at $\mathbf{c} = 0$.

arriving key x	**update the state** $(q, \mathbf{c})$
$\mathbf{c} = 0$	$q \leftarrow x$, $\mathbf{c} \leftarrow 1$
$\mathbf{c} > 0$ and $x = q$	$\mathbf{c} \leftarrow \mathbf{c} + 1$
$\mathbf{c} > 0$ and $x \neq q$	$\mathbf{c} \leftarrow \mathbf{c} - 1$; if it becomes 0, set $q \leftarrow \varnothing$

In the last case, pair the arriving x with one earlier, not-yet-paired copy of q and delete both. So $\mathbf{c}$ counts the copies of q that are still unpaired.

Throughout, keys (x, q, q_i) are plain symbols and counters ($\mathbf{c}$, $\mathbf{c_i}$, $\mathbf{C}$) are blue.

Boyer–Moore on a stream

Consider

$$a, b, a, c, a, a, b, a, \qquad f_a = 5 > 8/2.$$

The table shows the state *after* each item is processed.

t	arriving x	candidate q	counter $\mathbf{c}$	operation
1	a	a	**1**	start a candidate
2	b	$\varnothing$	**0**	pair a with b

Boyer–Moore on a stream

Consider

$$a, b, a, c, a, a, b, a, \qquad f_a = 5 > 8/2.$$

The table shows the state *after* each item is processed.

t	arriving x	candidate q	counter **c**	operation
1	a	a	**1**	start a candidate
2	b	$\varnothing$	**0**	pair a with b
3	a	a	**1**	start a candidate
4	c	$\varnothing$	**0**	pair a with c

Boyer–Moore on a stream

Consider

$$a, b, a, c, a, a, b, a, \qquad f_a = 5 > 8/2.$$

The table shows the state *after* each item is processed.

t	arriving x	candidate q	counter $\mathbf{c}$	operation
1	a	a	**1**	start a candidate
2	b	$\varnothing$	**0**	pair a with b
3	a	a	**1**	start a candidate
4	c	$\varnothing$	**0**	pair a with c
5	a	a	**1**	start a candidate
6	a	a	**2**	another unpaired a

Boyer–Moore on a stream

Consider

$$a, b, a, c, a, a, b, a, \qquad f_a = 5 > 8/2.$$

The table shows the state *after* each item is processed.

t	arriving x	candidate q	counter $\mathbf{c}$	operation
1	a	a	**1**	start a candidate
2	b	$\varnothing$	**0**	pair a with b
3	a	a	**1**	start a candidate
4	c	$\varnothing$	**0**	pair a with c
5	a	a	**1**	start a candidate
6	a	a	**2**	another unpaired a
7	b	a	**1**	pair one a with b
8	a	a	**2**	another unpaired a

At the end, two copies of a are unpaired; the other six items form the pairs $(a, b), (a, c), (a, b)$.

Why Boyer–Moore finds a majority

Cancellation invariant

After every prefix, the processed items split into

- disjoint deleted pairs, each made of two distinct keys; and
- exactly $\mathbf{c}$ unpaired copies of the current candidate q.

Each update preserves this: the three cases create the first unpaired copy, add another unpaired copy of q, or form one new pair of distinct keys.

Why Boyer–Moore finds a majority

Cancellation invariant

After every prefix, the processed items split into

- disjoint deleted pairs, each made of two distinct keys; and
- exactly $\mathbf{c}$ unpaired copies of the current candidate q.

Each update preserves this: the three cases create the first unpaired copy, add another unpaired copy of q, or form one new pair of distinct keys. Now suppose m is a strict majority of the whole stream. Deleting the pairs one at a time, the previous slide shows m is a strict majority of what remains: the $\mathbf{c}$ unpaired copies of q. A strict majority of a stream appears in it, so that stream is nonempty and $m = q$.

Why Boyer–Moore finds a majority

Cancellation invariant

After every prefix, the processed items split into

- disjoint deleted pairs, each made of two distinct keys; and
- exactly $\mathbf{c}$ unpaired copies of the current candidate q.

Each update preserves this: the three cases create the first unpaired copy, add another unpaired copy of q, or form one new pair of distinct keys. Now suppose m is a strict majority of the whole stream. Deleting the pairs one at a time, the previous slide shows m is a strict majority of what remains: the $\mathbf{c}$ unpaired copies of q. A strict majority of a stream appears in it, so that stream is nonempty and $m = q$.

Remarks.

- Boyer–Moore never uses N.
- Without the promise, the output may not be a majority (try a, b, c). A second pass that counts q checks whether $f_q > N/2$.

From pairs to groups of $k + 1$

Boyer–Moore protects one key with $f > N/2$. To protect every key above $N/(k + 1)$, delete groups of $k + 1$ distinct items instead of pairs.

From pairs to groups of $k+1$

Boyer–Moore protects one key with $f > N/2$. To protect every key above $N/(k+1)$, delete groups of $k+1$ distinct items instead of pairs. Suppose $f_x > N/(k+1)$ and we delete $k+1$ items with distinct keys. At most one of them is x, so

$$f_x - 1 > \frac{N}{k+1} - 1 = \frac{N-(k+1)}{k+1}.$$

The shorter stream has length $N-(k+1)$, so x is still more than a $1/(k+1)$ fraction of it.

From pairs to groups of $k+1$

Boyer–Moore protects one key with $f > N/2$. To protect every key above $N/(k+1)$, delete groups of $k+1$ distinct items instead of pairs.
Suppose $f_x > N/(k+1)$ and we delete $k+1$ items with distinct keys. At most one of them is x, so

$$f_x - 1 > \frac{N}{k+1} - 1 = \frac{N-(k+1)}{k+1}.$$

The shorter stream has length $N-(k+1)$, so x is still more than a $1/(k+1)$ fraction of it.
Repeating: as long as every deletion uses $k+1$ distinct keys, every key with $f_x > N/(k+1)$ survives to the end. With $k=1$ this is the majority argument.

From pairs to groups of $k+1$

Boyer–Moore protects one key with $f > N/2$. To protect every key above $N/(k+1)$, delete groups of $k+1$ distinct items instead of pairs.
Suppose $f_x > N/(k+1)$ and we delete $k+1$ items with distinct keys. At most one of them is x, so

$$f_x - 1 > \frac{N}{k+1} - 1 = \frac{N-(k+1)}{k+1}.$$

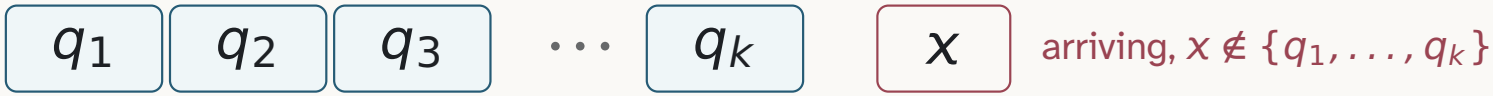
The shorter stream has length $N-(k+1)$, so x is still more than a $1/(k+1)$ fraction of it.
Repeating: as long as every deletion uses $k+1$ distinct keys, every key with $f_x > N/(k+1)$ survives to the end. With $k=1$ this is the majority argument.

We cannot store the remaining items and search them for $k+1$ distinct keys. How can we find such groups online, with small memory?

The Misra–Gries update rule

Keep k slots $(q_1, \mathbf{c_1}), \ldots, (q_k, \mathbf{c_k})$: q_i is a stored key and $\mathbf{c_i}$ counts its stored, not-yet-deleted copies. A slot is empty when $\mathbf{c_i} = 0$.

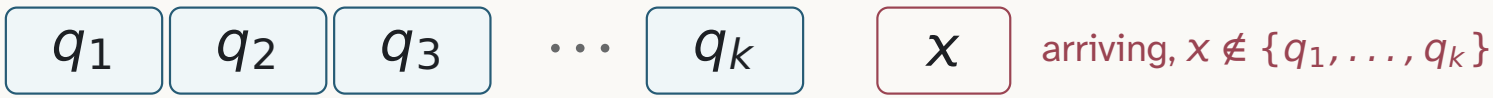
If all slots are full and an arriving key matches none of them, we have found $k + 1$ distinct keys:



The Misra–Gries update rule

Keep k slots $(q_1, \mathbf{c_1}), \ldots, (q_k, \mathbf{c_k})$: q_i is a stored key and $\mathbf{c_i}$ counts its stored, not-yet-deleted copies. A slot is empty when $\mathbf{c_i} = 0$.

If all slots are full and an arriving key matches none of them, we have found $k + 1$ distinct keys:



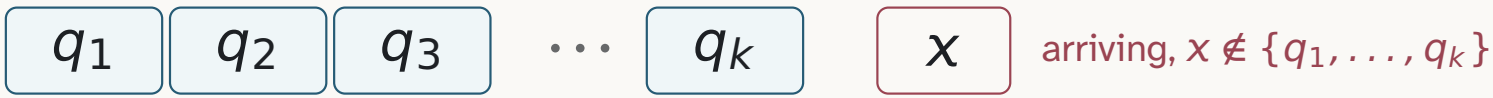
For each arriving key x:

1. If $x = q_i$ for a nonempty slot, increment $\mathbf{c_i}$.

The Misra–Gries update rule

Keep k slots $(q_1, \mathbf{c_1}), \ldots, (q_k, \mathbf{c_k})$: q_i is a stored key and $\mathbf{c_i}$ counts its stored, not-yet-deleted copies. A slot is empty when $\mathbf{c_i} = 0$.

If all slots are full and an arriving key matches none of them, we have found $k + 1$ distinct keys:



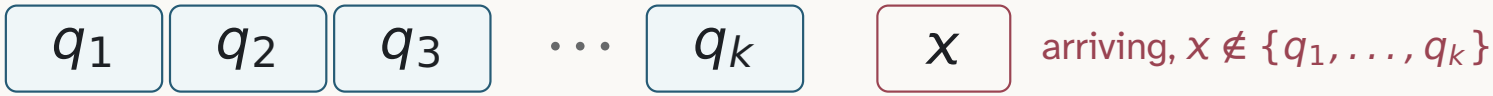
For each arriving key x:

1. If $x = q_i$ for a nonempty slot, increment $\mathbf{c_i}$.
2. Else, if some slot is empty, set $q_i \leftarrow x$ and $\mathbf{c_i} \leftarrow 1$.

The Misra–Gries update rule

Keep k slots $(q_1, \mathbf{c_1}), \ldots, (q_k, \mathbf{c_k})$: q_i is a stored key and $\mathbf{c_i}$ counts its stored, not-yet-deleted copies. A slot is empty when $\mathbf{c_i} = 0$.

If all slots are full and an arriving key matches none of them, we have found $k + 1$ distinct keys:



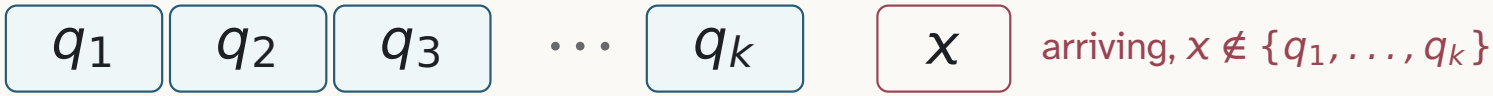
For each arriving key x:

1. If $x = q_i$ for a nonempty slot, increment $\mathbf{c_i}$.
2. Else, if some slot is empty, set $q_i \leftarrow x$ and $\mathbf{c_i} \leftarrow 1$.
3. Else delete one copy each of $x, q_1, \ldots, q_k$: discard x and decrement every $\mathbf{c_i}$; slots reaching 0 become empty.

The Misra–Gries update rule

Keep k slots $(q_1, \mathbf{c_1}), \ldots, (q_k, \mathbf{c_k})$: q_i is a stored key and $\mathbf{c_i}$ counts its stored, not-yet-deleted copies. A slot is empty when $\mathbf{c_i} = 0$.

If all slots are full and an arriving key matches none of them, we have found $k + 1$ distinct keys:



For each arriving key x:

1. If $x = q_i$ for a nonempty slot, increment $\mathbf{c_i}$.
2. Else, if some slot is empty, set $q_i \leftarrow x$ and $\mathbf{c_i} \leftarrow 1$.
3. Else delete one copy each of $x, q_1, \ldots, q_k$: discard x and decrement every $\mathbf{c_i}$; slots reaching 0 become empty.

Cases 1–2 keep the arriving occurrence explicitly. Case 3 discards $k + 1$ distinct occurrences, so the analysis will bound how many times case 3 can happen.

Misra–Gries on a stream

Take $k = 3$ and the stream

$$a, b, c, d, a, c, b, a, d, c, a, b, \qquad N = 12.$$

Keys above $N/(k+1) = 3$ must survive. Here $f_a = 4, f_b = 3, f_c = 3$, $f_d = 2$.

processed	slot 1	slot 2	slot 3
a, b, c	$(a, \mathbf{1})$	$(b, \mathbf{1})$	$(c, \mathbf{1})$

The first three distinct keys fill the three slots.

Misra–Gries on a stream

Take $k = 3$ and the stream

$$a, b, c, d, a, c, b, a, d, c, a, b, \qquad N = 12.$$

Keys above $N/(k+1) = 3$ must survive. Here $f_a = 4, f_b = 3, f_c = 3$, $f_d = 2$.

arrival	slot 1	slot 2	slot 3
d	$(a, \mathbf{1})$	$(b, \mathbf{1})$	$(c, \mathbf{1})$

All slots are full and d is new: delete one each of a, b, c, d. All counters become 0.

Misra–Gries on a stream

Take $k = 3$ and the stream

$$a, b, c, d, a, c, b, a, d, c, a, b, \qquad N = 12.$$

Keys above $N/(k+1) = 3$ must survive. Here $f_a = 4, f_b = 3, f_c = 3$, $f_d = 2$.

processed next	slot 1	slot 2	slot 3
a, c, b, a	$(a, \mathbf{2})$	$(c, \mathbf{1})$	$(b, \mathbf{1})$

The second a matches slot 1, so its counter rises to 2.

Misra–Gries on a stream

Take $k = 3$ and the stream

$$a, b, c, d, a, c, b, a, d, c, a, b, \qquad N = 12.$$

Keys above $N/(k+1) = 3$ must survive. Here $f_a = 4, f_b = 3, f_c = 3$, $f_d = 2$.

arrival	slot 1	slot 2	slot 3
d	$(a, \mathbf{2})$	$(c, \mathbf{1})$	$(b, \mathbf{1})$

Again d is new and the slots are full: delete one each of a, c, b, d. Now only $(a, \mathbf{1})$ remains.

Misra–Gries on a stream

Take $k = 3$ and the stream

$$a, b, c, d, a, c, b, a, d, c, a, b, \qquad N = 12.$$

Keys above $N/(k+1) = 3$ must survive. Here $f_a = 4, f_b = 3, f_c = 3, f_d = 2$.

final arrivals	slot 1	slot 2	slot 3
c, a, b	$(a, \mathbf{2})$	$(c, \mathbf{1})$	$(b, \mathbf{1})$

key	a	b	c	d
true f_x	4	3	3	2
stored $\widetilde{f}_x$	2	1	1	0

Misra–Gries on a stream

Take $k = 3$ and the stream

$$a, b, c, d, a, c, b, a, d, c, a, b, \qquad N = 12.$$

Keys above $N/(k+1) = 3$ must survive. Here $f_a = 4, f_b = 3, f_c = 3, f_d = 2$.

final arrivals	slot 1	slot 2	slot 3
c, a, b	$(a, \mathbf{2})$	$(c, \mathbf{1})$	$(b, \mathbf{1})$

key	a	b	c	d
true f_x	4	3	3	2
stored $\widetilde{f}_x$	2	1	1	0

In this particular example, every key appears in both deleted groups, so every key lost exactly 2 copies. In general a key can lose at most one copy per deleted group. The heavy key a survived; b and c also survived, though they are not above 3.

Counting what Misra–Gries forgets

Misra–Gries invariant (compare Boyer–Moore)

After every prefix, the processed items split into

- disjoint deleted groups, each made of $k + 1$ items with distinct keys; and
- for each slot i, exactly $\mathbf{c_i}$ undeleted copies of q_i.

Cases 1–2 add an undeleted copy; case 3 forms a new group from x and one undeleted copy of each q_i.

Counting what Misra–Gries forgets

Misra–Gries invariant (compare Boyer–Moore)

After every prefix, the processed items split into

- disjoint deleted groups, each made of $k + 1$ items with distinct keys; and
- for each slot i, exactly $\mathbf{c_i}$ undeleted copies of q_i.

Cases 1–2 add an undeleted copy; case 3 forms a new group from x and one undeleted copy of each q_i.
Let L be the number of times case 3 happens. The L groups are disjoint and each has $k + 1$ items, so $(k + 1)L \leq N$, that is, $L \leq N/(k + 1)$.

Counting what Misra–Gries forgets

Misra–Gries invariant (compare Boyer–Moore)

After every prefix, the processed items split into

- disjoint deleted groups, each made of $k+1$ items with distinct keys; and
- for each slot i, exactly $\mathbf{c_i}$ undeleted copies of q_i.

Cases 1–2 add an undeleted copy; case 3 forms a new group from x and one undeleted copy of each q_i.

Let L be the number of times case 3 happens. The L groups are disjoint and each has $k+1$ items, so $(k+1)L \le N$, that is, $L \le N/(k+1)$.

Fix a key x and let $\widetilde{f}_x$ be its counter at the end (0 if x is not stored). The copies of x are either undeleted, and there are $\widetilde{f}_x$ of those, or in a group. Each group holds at most one copy of x, so

$$0 \le f_x - \widetilde{f}_x \le L \le \frac{N}{k+1}.$$

The Misra–Gries guarantee

Misra–Gries

With k slots, at the end of the stream, simultaneously for every key x,

$$f_x - \frac{N}{k+1} \leq \widetilde{f}_x \leq f_x.$$

In particular, every x with $f_x > N/(k+1)$ has $\widetilde{f}_x > 0$, so it is stored in a slot.

The Misra–Gries guarantee

Misra–Gries

With k slots, at the end of the stream, simultaneously for every key x,

$$f_x - \frac{N}{k+1} \le \widetilde{f}_x \le f_x.$$

In particular, every x with $f_x > N/(k+1)$ has $\widetilde{f}_x > 0$, so it is stored in a slot.

In the example, $L = 2 \le 12/4$, and every key lost exactly L because every key was in both groups.

The Misra–Gries guarantee

Misra–Gries

With k slots, at the end of the stream, simultaneously for every key x,

$$f_x - \frac{N}{k+1} \leq \widetilde{f}_x \leq f_x.$$

In particular, every x with $f_x > N/(k+1)$ has $\widetilde{f}_x > 0$, so it is stored in a slot.

In the example, $L = 2 \leq 12/4$, and every key lost exactly L because every key was in both groups.

No randomness. The bound holds for every stream, including one chosen by an adversary.

No knowledge of N. The update rule never uses N, and the proof applies to every prefix. If we stop after t items, the bound holds with N replaced by t.

Back to the router

Point queries. If x occupies a slot, return its counter; otherwise return 0. Take $k = \lceil 1/\epsilon \rceil$. Then $N/(k+1) < \epsilon N$, so returning $\widetilde{f}_x$ gives

$$f_x - \epsilon N < \widetilde{f}_x \leq f_x \qquad \text{for every key } x.$$

Back to the router

Point queries. If x occupies a slot, return its counter; otherwise return 0. Take $k = \lceil 1/\epsilon \rceil$. Then $N/(k+1) < \epsilon N$, so returning $\widetilde{f}_x$ gives

$$f_x - \epsilon N < \widetilde{f}_x \leq f_x \qquad \text{for every key } x.$$

Heavy hitters. Report the stored keys with $\widetilde{f}_x > (\phi - \epsilon)N$.

- If $f_x > \phi N$, then $\widetilde{f}_x > f_x - \epsilon N > (\phi - \epsilon)N$, so x is reported.
- If x is reported, then $f_x \geq \widetilde{f}_x > (\phi - \epsilon)N$.
- At most k keys are stored, so at most k are reported.

The threshold uses N, so the reporting step needs the stream length so far: one extra counter.

Back to the router

Point queries. If x occupies a slot, return its counter; otherwise return 0. Take $k = \lceil 1/\epsilon \rceil$. Then $N/(k+1) < \epsilon N$, so returning $\widetilde{f}_x$ gives

$$f_x - \epsilon N < \widetilde{f}_x \leq f_x \qquad \text{for every key } x.$$

Heavy hitters. Report the stored keys with $\widetilde{f}_x > (\phi - \epsilon)N$.

- If $f_x > \phi N$, then $\widetilde{f}_x > f_x - \epsilon N > (\phi - \epsilon)N$, so x is reported.
- If x is reported, then $f_x \geq \widetilde{f}_x > (\phi - \epsilon)N$.
- At most k keys are stored, so at most k are reported.

The threshold uses N, so the reporting step needs the stream length so far: one extra counter.

Numbers. Sources above 1%: $\phi = 0.01$, $\epsilon = 0.005$, so $k = 200$. With 200 key–counter pairs we report every source above 1% and none below 0.5%, whether the router sees 10^4 packets or 10^{12}.

PART 3

Randomization: Count–Min

3

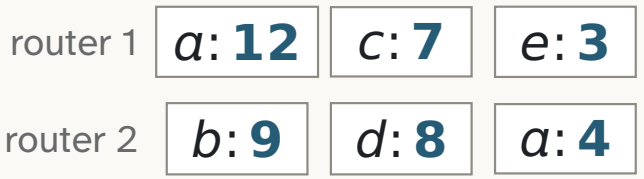
Why look beyond Misra–Gries?

For one router and an insertion-only stream, Misra–Gries is hard to beat: deterministic, every key at once, heavy hitters included. Two situations push us toward a different kind of summary.

Why look beyond Misra–Gries?

For one router and an insertion-only stream, Misra–Gries is hard to beat: deterministic, every key at once, heavy hitters included. Two situations push us toward a different kind of summary.

Merging. A Misra–Gries state is a weighted list of key–counter pairs, not an array whose coordinates have fixed meanings.

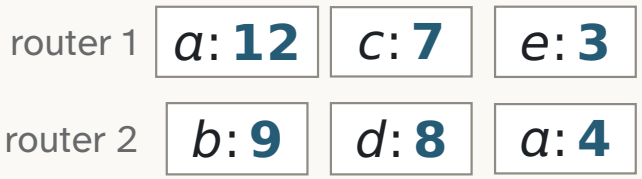


combine equal keys, then
compress up to $2k$ candidates back to k

Why look beyond Misra–Gries?

For one router and an insertion-only stream, Misra–Gries is hard to beat: deterministic, every key at once, heavy hitters included. Two situations push us toward a different kind of summary.

Merging. A Misra–Gries state is a weighted list of key–counter pairs, not an array whose coordinates have fixed meanings.



combine equal keys, then
compress up to $2k$ candidates back to k

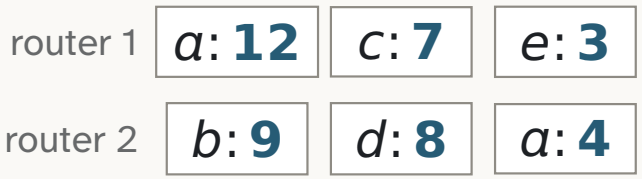
We do know which key each counter belongs to. The issue is *compression*: after concatenation, a merge procedure must reduce the weighted candidate list back to k slots by another cancellation argument. So Misra–Gries can be merged, but not by immediate coordinatewise addition.

Deletions. In the strict-turnstile model, counts go up and down but stay nonnegative. Misra–Gries cannot simply undo an earlier case-3 deletion.

Why look beyond Misra–Gries?

For one router and an insertion-only stream, Misra–Gries is hard to beat: deterministic, every key at once, heavy hitters included. Two situations push us toward a different kind of summary.

Merging. A Misra–Gries state is a weighted list of key–counter pairs, not an array whose coordinates have fixed meanings.



combine equal keys, then
compress up to $2k$ candidates back to k

We do know which key each counter belongs to. The issue is *compression*: after concatenation, a merge procedure must reduce the weighted candidate list back to k slots by another cancellation argument. So Misra–Gries can be merged, but not by immediate coordinatewise addition.

Deletions. In the strict-turnstile model, counts go up and down but stay nonnegative. Misra–Gries cannot simply undo an earlier case-3 deletion.

A sketch with fixed counter coordinates makes merging and deletions algebraic. The price is that different keys share counters.

One hash row

Can we let keys collide, and still control the error? Start with a single row of counters. Choose a hash function $h : U \to [w]$ and keep w counters $\mathbf{C}[1], \ldots, \mathbf{C}[w]$, initially zero.

Update. When a_t arrives, $\mathbf{C}[h(a_t)] \leftarrow \mathbf{C}[h(a_t)] + 1$.

Query. To estimate f_x, return $\widehat{f}_x^{(1)} = \mathbf{C}[h(x)]$.

One hash row

Can we let keys collide, and still control the error? Start with a single row of counters. Choose a hash function $h : U \to [w]$ and keep w counters $\mathbf{C}[1], \ldots, \mathbf{C}[w]$, initially zero.

Update. When a_t arrives, $\mathbf{C}[h(a_t)] \leftarrow \mathbf{C}[h(a_t)] + 1$.

Query. To estimate f_x, return $\widehat{f}_x^{(1)} = \mathbf{C}[h(x)]$.

Small example: $w = 4$, $h(x) = 2$, $h(a) = 2$, $h(b) = 4$, stream x, a, x, b.

	arriving key	$\mathbf{C}[1]$	$\mathbf{C}[2]$	$\mathbf{C}[3]$	$\mathbf{C}[4]$
initial	–	**0**	**0**	**0**	**0**

One hash row

Can we let keys collide, and still control the error? Start with a single row of counters. Choose a hash function $h : U \to [w]$ and keep w counters $\mathbf{C}[1], \ldots, \mathbf{C}[w]$, initially zero.

Update. When a_t arrives, $\mathbf{C}[h(a_t)] \leftarrow \mathbf{C}[h(a_t)] + 1$.

Query. To estimate f_x, return $\widehat{f}_x^{(1)} = \mathbf{C}[h(x)]$.

Small example: $w = 4$, $h(x) = 2$, $h(a) = 2$, $h(b) = 4$, stream x, a, x, b.

	arriving key	$\mathbf{C}[1]$	$\mathbf{C}[2]$	$\mathbf{C}[3]$	$\mathbf{C}[4]$
initial	–	**0**	**0**	**0**	**0**
1	x	**0**	**1**	**0**	**0**
2	a	**0**	**2**	**0**	**0**

One hash row

Can we let keys collide, and still control the error? Start with a single row of counters. Choose a hash function $h : U \to [w]$ and keep w counters $\mathbf{C}[1], \ldots, \mathbf{C}[w]$, initially zero.

Update. When a_t arrives, $\mathbf{C}[h(a_t)] \leftarrow \mathbf{C}[h(a_t)] + 1$.

Query. To estimate f_x, return $\widehat{f}_x^{(1)} = \mathbf{C}[h(x)]$.

Small example: $w = 4$, $h(x) = 2$, $h(a) = 2$, $h(b) = 4$, stream x, a, x, b.

	arriving key	$\mathbf{C}[1]$	$\mathbf{C}[2]$	$\mathbf{C}[3]$	$\mathbf{C}[4]$
initial	–	**0**	**0**	**0**	**0**
1	x	**0**	**1**	**0**	**0**
2	a	**0**	**2**	**0**	**0**
3	x	**0**	**3**	**0**	**0**
4	b	**0**	**3**	**0**	**1**

One hash row

Can we let keys collide, and still control the error? Start with a single row of counters. Choose a hash function $h : U \to [w]$ and keep w counters $\mathbf{C}[1], \ldots, \mathbf{C}[w]$, initially zero.

Update. When a_t arrives, $\mathbf{C}[h(a_t)] \leftarrow \mathbf{C}[h(a_t)] + 1$.

Query. To estimate f_x, return $\widehat{f}_x^{(1)} = \mathbf{C}[h(x)]$.

Small example: $w = 4$, $h(x) = 2$, $h(a) = 2$, $h(b) = 4$, stream x, a, x, b.

	arriving key	$\mathbf{C}[1]$	$\mathbf{C}[2]$	$\mathbf{C}[3]$	$\mathbf{C}[4]$
initial	–	**0**	**0**	**0**	**0**
1	x	**0**	**1**	**0**	**0**
2	a	**0**	**2**	**0**	**0**
3	x	**0**	**3**	**0**	**0**
4	b	**0**	**3**	**0**	**1**

Here $f_x = 2$ but $\mathbf{C}[h(x)] = \mathbf{C}[2] = 3$. The extra 1 is the copy of a, which landed in x's bucket.

What is the one-row error?

Fix a query key x. Its bucket holds the copies of x plus every copy of every other key that hashes to the same bucket:

$$\widehat{f}_x^{(1)} = \mathbf{C}[h(x)] = \underbrace{f_x}_{\text{copies of } x} + \underbrace{\sum_{y \neq x} f_y \, \mathbf{1}\{h(y) = h(x)\}}_{X\text{: collision mass}} .$$

What is the one-row error?

Fix a query key x. Its bucket holds the copies of x plus every copy of every other key that hashes to the same bucket:

$$\widehat{f}_x^{(1)} = \mathbf{C}[h(x)] = \underbrace{f_x}_{\text{copies of } x} + \underbrace{\sum_{y \neq x} f_y \, \mathbf{1}\{h(y) = h(x)\}}_{X\text{: collision mass}}.$$

Two consequences:

1. $X \geq 0$, so $\widehat{f}_x^{(1)} \geq f_x$ always. One row never undercounts.
2. A colliding key y contributes all f_y of its copies. One collision with a very frequent key is much worse than many collisions with rare keys.

What is the one-row error?

Fix a query key x. Its bucket holds the copies of x plus every copy of every other key that hashes to the same bucket:

$$\widehat{f}_x^{(1)} = \mathbf{C}[h(x)] = \underbrace{f_x}_{\text{copies of } x} + \underbrace{\sum_{y \neq x} f_y \, \mathbf{1}\{h(y) = h(x)\}}_{X\text{: collision mass}}.$$

Two consequences:

1. $X \geq 0$, so $\widehat{f}_x^{(1)} \geq f_x$ always. One row never undercounts.
2. A colliding key y contributes all f_y of its copies. One collision with a very frequent key is much worse than many collisions with rare keys.

So the question is how much *frequency* shares x's bucket, not how many keys do.

The randomness model

The hash function is chosen at random before the stream is processed, and we assume the stream was not chosen after seeing that random choice. Equivalently, we fix the stream and analyze only the randomness in h.



the stream is fixed independently of h

The randomness model

The hash function is chosen at random before the stream is processed, and we assume the stream was not chosen after seeing that random choice. Equivalently, we fix the stream and analyze only the randomness in h.



the stream is fixed independently of h

This is often called the **oblivious-stream model**. Today it lets us treat the stream as fixed and randomize only over the hash functions.

The randomness model

The hash function is chosen at random before the stream is processed, and we assume the stream was not chosen after seeing that random choice. Equivalently, we fix the stream and analyze only the randomness in h.



the stream is fixed independently of h

This is often called the **oblivious-stream model**. Today it lets us treat the stream as fixed and randomize only over the hash functions. About h we assume only that for distinct keys $x \neq y$,

$$\Pr_h\{h(x) = h(y)\} \leq \frac{1}{w},$$

which a universal hash family guarantees.

How large is the collision error on average?

Recall the one-row error for a fixed query key x:

$$X = \sum_{y \neq x} f_y \, \mathbf{1}\{h(y) = h(x)\}.$$

How large is the collision error on average?

Recall the one-row error for a fixed query key x:

$$X = \sum_{y \neq x} f_y \, \mathbf{1}\{h(y) = h(x)\}.$$

By linearity of expectation,

$$\mathbb{E}[X] = \sum_{y \neq x} f_y \Pr_h\{h(y) = h(x)\} \leq \frac{1}{w} \sum_{y \neq x} f_y = \frac{N - f_x}{w} \leq \frac{N}{w}.$$

How large is the collision error on average?

Recall the one-row error for a fixed query key x:

$$X = \sum_{y \neq x} f_y \, \mathbf{1}\{h(y) = h(x)\}.$$

By linearity of expectation,

$$\mathbb{E}[X] = \sum_{y \neq x} f_y \Pr_h\{h(y) = h(x)\} \leq \frac{1}{w} \sum_{y \neq x} f_y = \frac{N - f_x}{w} \leq \frac{N}{w}.$$

We did not need the collision events for different y to be independent of each other. Linearity uses only each collision probability on its own, which is why a universal family is enough here.

The average error is at most N/w: the total stream, spread over w buckets. This is why ϵN is the natural accuracy scale for a hashing summary.

One row gives constant confidence

Since $X \geq 0$, Markov's inequality applies:

$$\Pr\{X > \epsilon N\} \leq \frac{\mathbb{E}[X]}{\epsilon N} \leq \frac{N/w}{\epsilon N} = \frac{1}{\epsilon w}.$$

One row gives constant confidence

Since $X \geq 0$, Markov's inequality applies:

$$\Pr\{X > \epsilon N\} \leq \frac{\mathbb{E}[X]}{\epsilon N} \leq \frac{N/w}{\epsilon N} = \frac{1}{\epsilon w}.$$

Choose $w = \lceil 3/\epsilon \rceil$. Then $\epsilon w \geq 3$, and

$$\Pr\left\{f_x \leq \widehat{f}_x^{(1)} \leq f_x + \epsilon N\right\} \geq \frac{2}{3}.$$

One row gives constant confidence

Since $X \geq 0$, Markov's inequality applies:

$$\Pr\{X > \epsilon N\} \leq \frac{\mathbb{E}[X]}{\epsilon N} \leq \frac{N/w}{\epsilon N} = \frac{1}{\epsilon w}.$$

Choose $w = \lceil 3/\epsilon \rceil$. Then $\epsilon w \geq 3$, and

$$\Pr\left\{f_x \leq \widehat{f}_x^{(1)} \leq f_x + \epsilon N\right\} \geq \frac{2}{3}.$$

Success probability $2/3$ is not good enough if we want failure probability $\delta = 10^{-6}$.

Why not just make the row wider?

Markov handles any δ: with $w = \lceil 1/(\epsilon\delta) \rceil$,

$$\Pr\{X > \epsilon N\} \leq \frac{1}{\epsilon w} \leq \delta.$$

Why not just make the row wider?

Markov handles any δ: with $w = \lceil 1/(\epsilon\delta) \rceil$,

$$\Pr\{X > \epsilon N\} \leq \frac{1}{\epsilon w} \leq \delta.$$

The cost is a factor $1/\delta$ in memory. Compare at $\delta = 10^{-6}$:

one row, width $\lceil 1/(\epsilon\delta) \rceil$	about $10^6/\epsilon$ counters
d rows, width $\lceil 3/\epsilon \rceil$	$d = \lceil \log_3 10^6 \rceil = 13$, about $39/\epsilon$ counters

Why not just make the row wider?

Markov handles any δ: with $w = \lceil 1/(\epsilon\delta) \rceil$,

$$\Pr\{X > \epsilon N\} \leq \frac{1}{\epsilon w} \leq \delta.$$

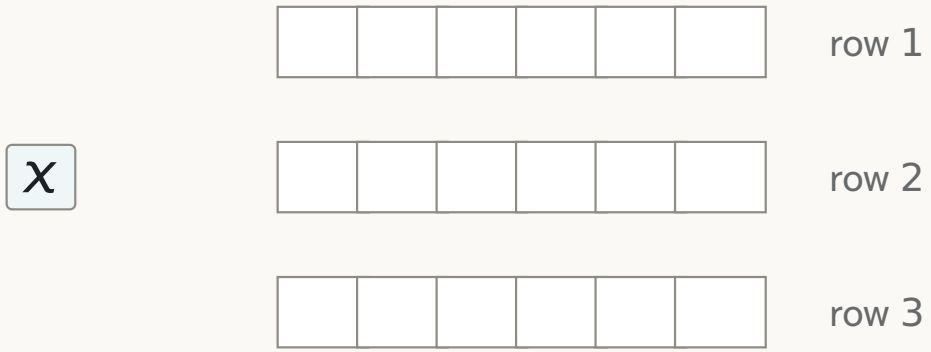
The cost is a factor $1/\delta$ in memory. Compare at $\delta = 10^{-6}$:

one row, width $\lceil 1/(\epsilon\delta) \rceil$	about $10^6/\epsilon$ counters
d rows, width $\lceil 3/\epsilon \rceil$	$d = \lceil \log_3 10^6 \rceil = 13$, about $39/\epsilon$ counters

The second line comes from several independent rows, which we build next. In fact, the $1/\delta$ cost for a single row is real; the lower-bound example is in the appendix.

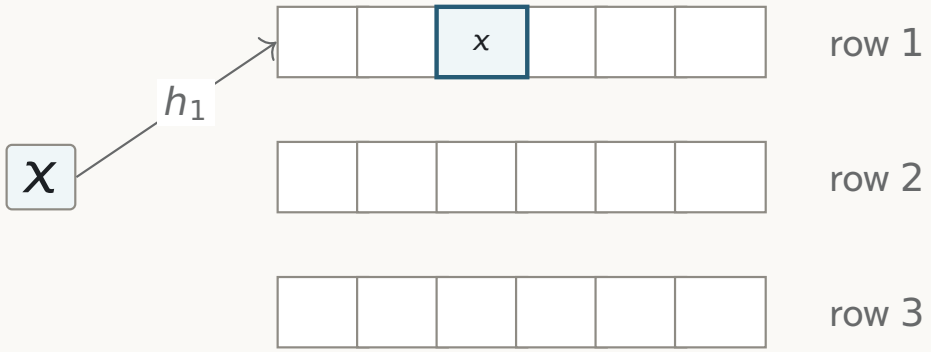
Several independent rows

Use d hash functions $h_1, \ldots, h_d$, drawn independently. Each row is a separate one-row estimator for x, with its own chance of putting x in a lightly loaded bucket.



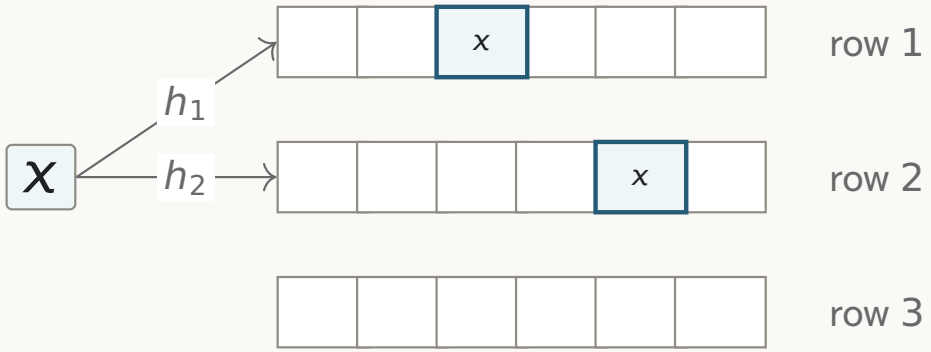
Several independent rows

Use d hash functions $h_1, \ldots, h_d$, drawn independently. Each row is a separate one-row estimator for x, with its own chance of putting x in a lightly loaded bucket.



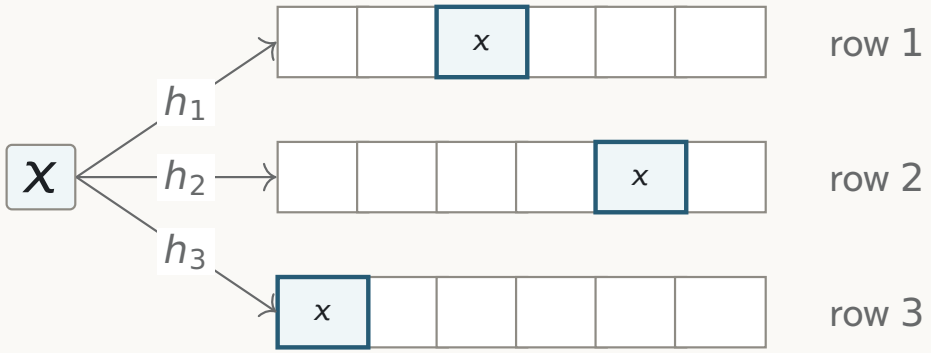
Several independent rows

Use d hash functions $h_1, \ldots, h_d$, drawn independently. Each row is a separate one-row estimator for x, with its own chance of putting x in a lightly loaded bucket.



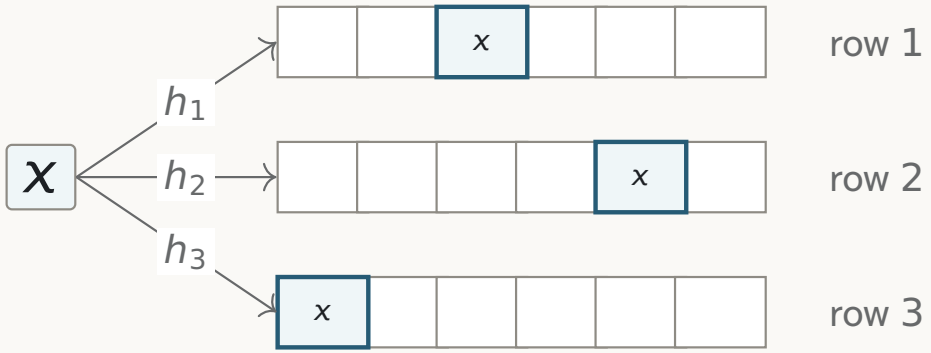
Several independent rows

Use d hash functions $h_1, \ldots, h_d$, drawn independently. Each row is a separate one-row estimator for x, with its own chance of putting x in a lightly loaded bucket.



Several independent rows

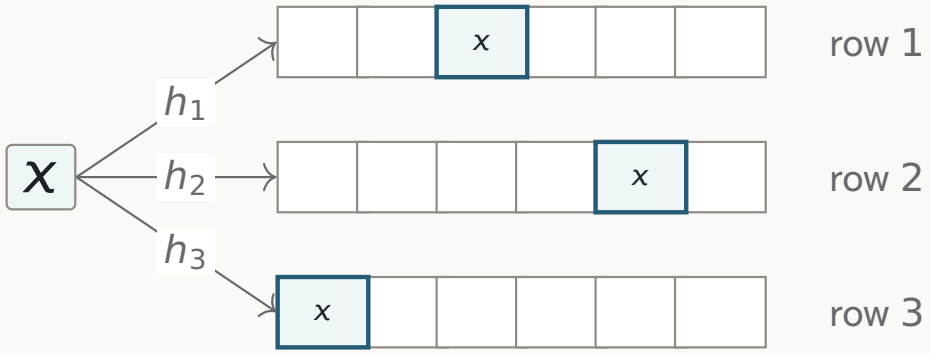
Use d hash functions $h_1, \ldots, h_d$, drawn independently. Each row is a separate one-row estimator for x, with its own chance of putting x in a lightly loaded bucket.



Row j returns $f_x + X_j$. How should we combine the d answers: average? median? something else?

Several independent rows

Use d hash functions $h_1, \ldots, h_d$, drawn independently. Each row is a separate one-row estimator for x, with its own chance of putting x in a lightly loaded bucket.



Row j returns $f_x + X_j$. How should we combine the d answers: average? median? something else?
Every $X_j \geq 0$, so every row's answer is an upper bound on f_x, and the smallest is the best upper bound. Take the *minimum*.

Count–Min

Choose w, d and independent hash functions $h_1, \ldots, h_d : U \to [w]$ from a universal family. Keep a $d \times w$ array of counters $\mathbf{C}$, initially zero.
Update. When a_t arrives, increment one counter in every row:

$$\mathbf{C}[j, h_j(a_t)] \leftarrow \mathbf{C}[j, h_j(a_t)] + 1, \qquad j = 1, \ldots, d.$$

Query. For a key x, return

$$\widehat{f}_x = \min_{1 \le j \le d} \mathbf{C}[j, h_j(x)].$$

The hats differ on purpose: $\widetilde{f}$ (Misra–Gries) underestimates, $\widehat{f}$ (Count–Min) overestimates.

Count–Min

Choose w, d and independent hash functions $h_1, \ldots, h_d : U \to [w]$ from a universal family. Keep a $d \times w$ array of counters $\mathbf{C}$, initially zero.
Update. When a_t arrives, increment one counter in every row:

$$\mathbf{C}[j, h_j(a_t)] \leftarrow \mathbf{C}[j, h_j(a_t)] + 1, \qquad j = 1, \ldots, d.$$

Query. For a key x, return

$$\widehat{f}_x = \min_{1 \le j \le d} \mathbf{C}[j, h_j(x)].$$

The hats differ on purpose: $\widetilde{f}$ (Misra–Gries) underestimates, $\widehat{f}$ (Count–Min) overestimates.

Suppose $f_x = 4, f_a = 1, f_b = 3$, and in three rows x shares its bucket as follows:

	sharing x's bucket	counter	as $f_x + X_j$
row 1	a	**5**	$4 + 1$
row 2	b	**7**	$4 + 3$
row 3	nothing	**4**	$4 + 0$

Then $\widehat{f}_x = \min\{5, 7, 4\} = 4$: the row that happened to avoid heavy collisions wins.

More rows, more confidence

Fix x and let X_j be the collision error in row j. With $w = \lceil 3/\epsilon \rceil$, the one-row bound gives

$$\Pr\{X_j > \epsilon N\} \le \frac{1}{3} \qquad \text{for each } j.$$

More rows, more confidence

Fix x and let X_j be the collision error in row j. With $w = \lceil 3/\epsilon \rceil$, the one-row bound gives

$$\Pr\{X_j > \epsilon N\} \le \frac{1}{3} \qquad \text{for each } j.$$

Since $\widehat{f}_x = f_x + \min_j X_j$, the estimate is bad only if *every* row is bad. Each X_j depends only on h_j, and the h_j are independent, so

$$\begin{aligned}\Pr\{\widehat{f}_x > f_x + \epsilon N\} &= \Pr\{X_1 > \epsilon N, \ldots, X_d > \epsilon N\} \\ &= \prod_{j=1}^{d} \Pr\{X_j > \epsilon N\} \le \left(\frac{1}{3}\right)^d.\end{aligned}$$

This is the step that needs independence. Within one row we used only collision probabilities; multiplying across rows needs the rows to be independent of each other.

More rows, more confidence

Fix x and let X_j be the collision error in row j. With $w = \lceil 3/\epsilon \rceil$, the one-row bound gives

$$\Pr\{X_j > \epsilon N\} \le \frac{1}{3} \qquad \text{for each } j.$$

Since $\widehat{f}_x = f_x + \min_j X_j$, the estimate is bad only if *every* row is bad. Each X_j depends only on h_j, and the h_j are independent, so

$$\begin{aligned}\Pr\{\widehat{f}_x > f_x + \epsilon N\} &= \Pr\{X_1 > \epsilon N, \ldots, X_d > \epsilon N\} \\ &= \prod_{j=1}^{d} \Pr\{X_j > \epsilon N\} \le \left(\frac{1}{3}\right)^d.\end{aligned}$$

This is the step that needs independence. Within one row we used only collision probabilities; multiplying across rows needs the rows to be independent of each other.

To make this at most δ, we need $3^{-d} \le \delta$, i.e. $d \ge \log_3(1/\delta)$. Take $d = \lceil \log_3(1/\delta) \rceil$. Width sets the accuracy ϵ; depth sets the confidence δ.

The Count–Min guarantee

Count–Min

With $w = \lceil 3/\epsilon \rceil$ and $d = \lceil \log_3(1/\delta) \rceil$, for each **fixed** key x,

$$f_x \le \widehat{f}_x \le f_x + \epsilon N$$

with probability at least $1 - \delta$ over the choice of hash functions. The sketch has $dw = O(\epsilon^{-1} \log(1/\delta))$ counters.

"Fixed" means x is chosen without looking at the sketch. Querying, say, the key whose counters look largest is not covered.

The Count–Min guarantee

Count–Min

With $w = \lceil 3/\epsilon \rceil$ and $d = \lceil \log_3(1/\delta) \rceil$, for each **fixed** key x,

$$f_x \le \widehat{f}_x \le f_x + \epsilon N$$

with probability at least $1 - \delta$ over the choice of hash functions. The sketch has $dw = O(\epsilon^{-1} \log(1/\delta))$ counters.

"Fixed" means x is chosen without looking at the sketch. Querying, say, the key whose counters look largest is not covered.

Many queries. For Q keys named in advance, give each failure probability δ/Q by taking $d = \lceil \log_3(Q/\delta) \rceil$. A union bound gives

$$\Pr\{\text{some query fails}\} \le Q \cdot \delta/Q = \delta.$$

So depth grows only like $\log Q$.

The Count–Min guarantee

Count–Min

With $w = \lceil 3/\epsilon \rceil$ and $d = \lceil \log_3(1/\delta) \rceil$, for each **fixed** key x,

$$f_x \le \widehat{f}_x \le f_x + \epsilon N$$

with probability at least $1 - \delta$ over the choice of hash functions. The sketch has $dw = O(\epsilon^{-1} \log(1/\delta))$ counters.

"Fixed" means x is chosen without looking at the sketch. Querying, say, the key whose counters look largest is not covered.

Many queries. For Q keys named in advance, give each failure probability δ/Q by taking $d = \lceil \log_3(Q/\delta) \rceil$. A union bound gives

$$\Pr\{\text{some query fails}\} \le Q \cdot \delta/Q = \delta.$$

So depth grows only like $\log Q$.

No knowledge of N. w and d depend only on ϵ and δ, and the updates never use N. If we want N, it is free: every row sums to N.

Count–Min is a random linear map

For a tiny example, take $U = \{a, b, x\}$, width $w = 3$, depth $d = 2$, and

$$h_1(a) = 1,\ h_1(b) = 2,\ h_1(x) = 1, \qquad h_2(a) = 3,\ h_2(b) = 1,\ h_2(x) = 2.$$

Order the counters as $(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3)$ and the keys as (a, b, x).

Count–Min is a random linear map

For a tiny example, take $U = \{a, b, x\}$, width $w = 3$, depth $d = 2$, and

$$h_1(a) = 1,\ h_1(b) = 2,\ h_1(x) = 1, \qquad h_2(a) = 3,\ h_2(b) = 1,\ h_2(x) = 2.$$

Order the counters as $(1,1), (1,2), (1,3), (2,1), (2,2), (2,3)$ and the keys as (a, b, x).

$$\overset{\mathbf{C}}{\begin{bmatrix} \mathbf{C}_{1,1} \\ \mathbf{C}_{1,2} \\ \mathbf{C}_{1,3} \\ \mathbf{C}_{2,1} \\ \mathbf{C}_{2,2} \\ \mathbf{C}_{2,3} \end{bmatrix}} = \overset{A}{\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}} \overset{f}{\begin{bmatrix} f_a \\ f_b \\ f_x \end{bmatrix}}$$

Count–Min is a random linear map

For a tiny example, take $U = \{a, b, x\}$, width $w = 3$, depth $d = 2$, and

$$h_1(a) = 1,\ h_1(b) = 2,\ h_1(x) = 1, \qquad h_2(a) = 3,\ h_2(b) = 1,\ h_2(x) = 2.$$

Order the counters as $(1,1), (1,2), (1,3), (2,1), (2,2), (2,3)$ and the keys as (a, b, x).

$$\begin{matrix} \mathbf{C} \\ \begin{bmatrix} \mathbf{C}_{1,1} \\ \mathbf{C}_{1,2} \\ \mathbf{C}_{1,3} \\ \mathbf{C}_{2,1} \\ \mathbf{C}_{2,2} \\ \mathbf{C}_{2,3} \end{bmatrix} \end{matrix} = \begin{matrix} \\ \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \end{matrix} \begin{matrix} A \\ \\ \end{matrix} \begin{matrix} f \\ \begin{bmatrix} f_a \\ f_b \\ f_x \end{bmatrix} \end{matrix}$$

If $f = (2, 1, 3)^\top$, then

$$\mathbf{C} = Af = \begin{bmatrix} 5 & 1 & 0 & 1 & 3 & 2 \end{bmatrix}^\top.$$

For example, $C_{1,1} = 5$ because row 1 bucket 1 receives the two copies of a and the three copies of x.

Why linearity matters

In general, Count–Min has

$$A_{(j,b),y} = \mathbf{1}\{h_j(y) = b\}, \qquad \mathbf{C} = Af.$$

The random matrix A is fixed before the stream starts. Updating a key adds the corresponding column of A.

Why linearity matters

In general, Count–Min has

$$A_{(j,b),y} = \mathbf{1}\{h_j(y) = b\}, \qquad \mathbf{C} = Af.$$

The random matrix A is fixed before the stream starts. Updating a key adds the corresponding column of A.
Because the sketch is linear, it respects addition:

$$A(f + g) = Af + Ag.$$

Why linearity matters

In general, Count–Min has

$$A_{(j,b),y} = \mathbf{1}\{h_j(y) = b\}, \qquad \mathbf{C} = Af.$$

The random matrix A is fixed before the stream starts. Updating a key adds the corresponding column of A.
Because the sketch is linear, it respects addition:

$$A(f + g) = Af + Ag.$$

- **Merging.** If two routers use the same hash functions, they add their arrays entrywise to obtain the sketch of the combined traffic.
- **Deletions.** In the strict-turnstile model, a departing key subtracts the same column. Under the same hashing model, the proof still goes through with $N = \sum_y f_y$.

Why linearity matters

In general, Count–Min has

$$A_{(j,b),y} = \mathbf{1}\{h_j(y) = b\}, \qquad \mathbf{C} = Af.$$

The random matrix A is fixed before the stream starts. Updating a key adds the corresponding column of A.
Because the sketch is linear, it respects addition:

$$A(f + g) = Af + Ag.$$

- **Merging.** If two routers use the same hash functions, they add their arrays entrywise to obtain the sketch of the combined traffic.
- **Deletions.** In the strict-turnstile model, a departing key subtracts the same column. Under the same hashing model, the proof still goes through with $N = \sum_y f_y$.

If entries of f may become negative, collision error can have either sign and the minimum estimator is no longer justified. Count–Sketch handles that case.

PART 4

Comparison

4

Misra–Gries and Count–Min side by side

	Misra–Gries	**Count–Min**
state	k pairs $(q_i, \mathbf{c_i})$	$d \times w$ counters + d hash functions
size	$O(1/\epsilon)$ keys and counters	$O(\epsilon^{-1} \log(1/\delta))$ counters
randomness	none	over hash functions
error	underestimates by at most ϵN	overestimates by at most ϵN
queries	all keys simultaneously	one fixed key with prob. $1 - \delta$
heavy hitters	stored keys are the candidates	does not list keys
needs N?	only to report heavy hitters	no
merge	concatenate, combine, reduce to k	add the arrays
deletions	not supported	decrement, if counts stay ≥ 0

What to remember

- **Misra–Gries = cancel distinct keys.** With $k = \lceil 1/\epsilon \rceil$ slots, MisraGries underestimates every key by less than ϵN, deterministically, and every key above $N/(k+1)$ survives in a slot.

What to remember

- **Misra–Gries = cancel distinct keys.** With $k = \lceil 1/\epsilon \rceil$ slots, MisraGries underestimates every key by less than ϵN, deterministically, and every key above $N/(k+1)$ survives in a slot.
- **Count–Min = allow collisions, then repeat independently.** One row has collision error with mean at most N/w. Taking $w = \lceil 3/\epsilon \rceil$ gives constant success; taking $d = \lceil \log_3(1/\delta) \rceil$ independent rows boosts this to confidence $1 - \delta$.

What to remember

- **Misra–Gries = cancel distinct keys.** With $k = \lceil 1/\epsilon \rceil$ slots, MisraGries underestimates every key by less than ϵN, deterministically, and every key above $N/(k+1)$ survives in a slot.
- **Count–Min = allow collisions, then repeat independently.** One row has collision error with mean at most N/w. Taking $w = \lceil 3/\epsilon \rceil$ gives constant success; taking $d = \lceil \log_3(1/\delta) \rceil$ independent rows boosts this to confidence $1 - \delta$.
- **Reusable randomized-algorithms pattern.**

$$\begin{aligned} \mathbb{E}[\text{error}] &\Longrightarrow \text{constant success by Markov,} \\ &\Longrightarrow 1 - \delta \text{ by independent repetition.} \end{aligned}$$

What to remember

- **Misra–Gries = cancel distinct keys.** With $k = \lceil 1/\epsilon \rceil$ slots, MisraGries underestimates every key by less than ϵN, deterministically, and every key above $N/(k+1)$ survives in a slot.
- **Count–Min = allow collisions, then repeat independently.** One row has collision error with mean at most N/w. Taking $w = \lceil 3/\epsilon \rceil$ gives constant success; taking $d = \lceil \log_3(1/\delta) \rceil$ independent rows boosts this to confidence $1 - \delta$.
- **Reusable randomized-algorithms pattern.**

$$\begin{aligned} \mathbb{E}[\text{error}] &\Longrightarrow \text{constant success by Markov,} \\ &\Longrightarrow 1 - \delta \text{ by independent repetition.} \end{aligned}$$

- **Different strengths.** MisraGries gives simultaneous guarantees and heavy-hitter candidates; merging it requires merge-and-reduce. CountMin is linear, so sketches merge by addition and support strict-turnstile deletions.

What to remember

- **Misra–Gries = cancel distinct keys.** With $k = \lceil 1/\epsilon \rceil$ slots, MisraGries underestimates every key by less than ϵN, deterministically, and every key above $N/(k+1)$ survives in a slot.
- **Count–Min = allow collisions, then repeat independently.** One row has collision error with mean at most N/w. Taking $w = \lceil 3/\epsilon \rceil$ gives constant success; taking $d = \lceil \log_3(1/\delta) \rceil$ independent rows boosts this to confidence $1 - \delta$.
- **Reusable randomized-algorithms pattern.**

$$\begin{aligned} \mathbb{E}[\text{error}] &\Longrightarrow \text{constant success by Markov,} \\ &\Longrightarrow 1 - \delta \text{ by independent repetition.} \end{aligned}$$

- **Different strengths.** MisraGries gives simultaneous guarantees and heavy-hitter candidates; merging it requires merge-and-reduce. CountMin is linear, so sketches merge by addition and support strict-turnstile deletions.

parameter	MisraGries	CountMin
accuracy	$k = \Theta(1/\epsilon)$	$w = \Theta(1/\epsilon)$
confidence	deterministic	$d = \Theta(\log(1/\delta))$

Questions?

PART 5

Appendix

5

One row really does need width about $1/(\epsilon\delta)$

Assume $0 < \epsilon \leq 1/4$ and take a fully random h. Let

$$m = \left\lfloor \frac{1}{2\epsilon} \right\rfloor$$

and build a stream with keys $y_1, \ldots, y_m$, each having frequency $2\epsilon N$.

One row really does need width about $1/(\epsilon\delta)$

Assume $0 < \epsilon \le 1/4$ and take a fully random h. Let

$$m = \left\lfloor \frac{1}{2\epsilon} \right\rfloor$$

and build a stream with keys $y_1, \ldots, y_m$, each having frequency $2\epsilon N$.
If x shares a bucket with any y_i, then $X \ge 2\epsilon N > \epsilon N$. Each y_i lands in x's bucket independently with probability $1/w$, so

$$\Pr\{X > \epsilon N\} \ge 1 - \left(1 - \frac{1}{w}\right)^m \ge 1 - e^{-m/w} \ge \frac{m}{2w} \qquad \text{(when } m \le w\text{)},$$

using $1 - e^{-t} \ge t/2$ for $0 \le t \le 1$. If $m > w$, the failure probability is already above a constant.

One row really does need width about $1/(\epsilon\delta)$

Assume $0 < \epsilon \le 1/4$ and take a fully random h. Let

$$m = \left\lfloor \frac{1}{2\epsilon} \right\rfloor$$

and build a stream with keys $y_1, \ldots, y_m$, each having frequency $2\epsilon N$.
If x shares a bucket with any y_i, then $X \ge 2\epsilon N > \epsilon N$. Each y_i lands in x's bucket independently with probability $1/w$, so

$$\Pr\{X > \epsilon N\} \ge 1 - \left(1 - \frac{1}{w}\right)^m \ge 1 - e^{-m/w} \ge \frac{m}{2w} \qquad \text{(when } m \le w\text{)},$$

using $1 - e^{-t} \ge t/2$ for $0 \le t \le 1$. If $m > w$, the failure probability is already above a constant.
So to make the failure probability at most δ, we need

$$\frac{m}{2w} \lesssim \delta, \qquad \text{hence} \qquad w = \Omega\left(\frac{1}{\epsilon\delta}\right).$$

So the $1/\delta$ cost comes from using a *single row*, not from a weak application of Markov.

Why not aim for error proportional to D?

Recall $D = \#\{x : f_x > 0\}$, the number of distinct keys. One might ask for error ϵD instead of ϵN. Since $D \leq N$, that is a stronger demand.

Why not aim for error proportional to D?

Recall $D = \#\{x : f_x > 0\}$, the number of distinct keys. One might ask for error ϵD instead of ϵN. Since $D \leq N$, that is a stronger demand. For a fully random h, the collision probability is exactly $1/w$, so the computation on the main slides is an equality: $\mathbb{E}[X] = (N - f_x)/w$. For Markov to give constant success at scale ϵD we need $\mathbb{E}[X] \lesssim \epsilon D$, so

$$w \gtrsim \frac{N - f_x}{\epsilon D}.$$

When no key dominates, $N - f_x$ is comparable to N, and the required width is about $N/(\epsilon D)$: it grows with the average number of copies per key, N/D.

Why not aim for error proportional to D?

Recall $D = \#\{x : f_x > 0\}$, the number of distinct keys. One might ask for error ϵD instead of ϵN. Since $D \le N$, that is a stronger demand. For a fully random h, the collision probability is exactly $1/w$, so the computation on the main slides is an equality: $\mathbb{E}[X] = (N - f_x)/w$. For Markov to give constant success at scale ϵD we need $\mathbb{E}[X] \lesssim \epsilon D$, so

$$w \gtrsim \frac{N - f_x}{\epsilon D}.$$

When no key dominates, $N - f_x$ is comparable to N, and the required width is about $N/(\epsilon D)$: it grows with the average number of copies per key, N/D.

Example. $D = 10^6$ sources, $N = 10^{12}$ packets, $\epsilon = 0.01$. Then $\epsilon D = 10^4$ (the average source sends 10^6), and $w \gtrsim 10^{12}/(10^{-2} \cdot 10^6) = 10^8 > D$. A single row would be larger than an exact dictionary. In general $N/(\epsilon D) \ge D$ exactly when $N \ge \epsilon D^2$.

Why not aim for error proportional to D?

Recall $D = \#\{x : f_x > 0\}$, the number of distinct keys. One might ask for error ϵD instead of ϵN. Since $D \le N$, that is a stronger demand. For a fully random h, the collision probability is exactly $1/w$, so the computation on the main slides is an equality: $\mathbb{E}[X] = (N - f_x)/w$. For Markov to give constant success at scale ϵD we need $\mathbb{E}[X] \lesssim \epsilon D$, so

$$w \gtrsim \frac{N - f_x}{\epsilon D}.$$

When no key dominates, $N - f_x$ is comparable to N, and the required width is about $N/(\epsilon D)$: it grows with the average number of copies per key, N/D.

Example. $D = 10^6$ sources, $N = 10^{12}$ packets, $\epsilon = 0.01$. Then $\epsilon D = 10^4$ (the average source sends 10^6), and $w \gtrsim 10^{12}/(10^{-2} \cdot 10^6) = 10^8 > D$. A single row would be larger than an exact dictionary. In general $N/(\epsilon D) \ge D$ exactly when $N \ge \epsilon D^2$.

This shows only that this *analysis* cannot reach ϵD with small memory. It is not a lower bound for all algorithms.

Space in bits

The main slides count keys and counters. In bits, with keys from U and counts at most N:

- a key takes $\lceil \log_2 |U| \rceil$ bits; a counter takes $\lceil \log_2 (N+1) \rceil$ bits;
- MisraGries with k slots: $O\big(k(\log |U| + \log N)\big)$ bits;
- CountMin: $O(dw \log N)$ bits for the counters, plus $O(d \log |U|)$ bits to store seeds for standard universal hash families.

Space in bits

The main slides count keys and counters. In bits, with keys from U and counts at most N:

- a key takes $\lceil \log_2 |U| \rceil$ bits; a counter takes $\lceil \log_2 (N+1) \rceil$ bits;
- MisraGries with k slots: $O\big(k(\log |U| + \log N)\big)$ bits;
- CountMin: $O(dw \log N)$ bits for the counters, plus $O(d \log |U|)$ bits to store seeds for standard universal hash families.

Counter width is the one place the size of N enters the memory. If N is unknown, counters can be allocated with 64 bits, or grown when they overflow; neither changes the analysis.

Space in bits

The main slides count keys and counters. In bits, with keys from U and counts at most N:

- a key takes $\lceil \log_2 |U| \rceil$ bits; a counter takes $\lceil \log_2 (N+1) \rceil$ bits;
- MisraGries with k slots: $O\big(k(\log |U| + \log N)\big)$ bits;
- CountMin: $O(dw \log N)$ bits for the counters, plus $O(d \log |U|)$ bits to store seeds for standard universal hash families.

Counter width is the one place the size of N enters the memory. If N is unknown, counters can be allocated with 64 bits, or grown when they overflow; neither changes the analysis.
When keys are long (IPv6 addresses are 128 bits) and $\log(1/\delta)$ is small, storing no keys is a real saving for CountMin. When δ must be tiny, the $\log(1/\delta)$ factor favors MisraGries.