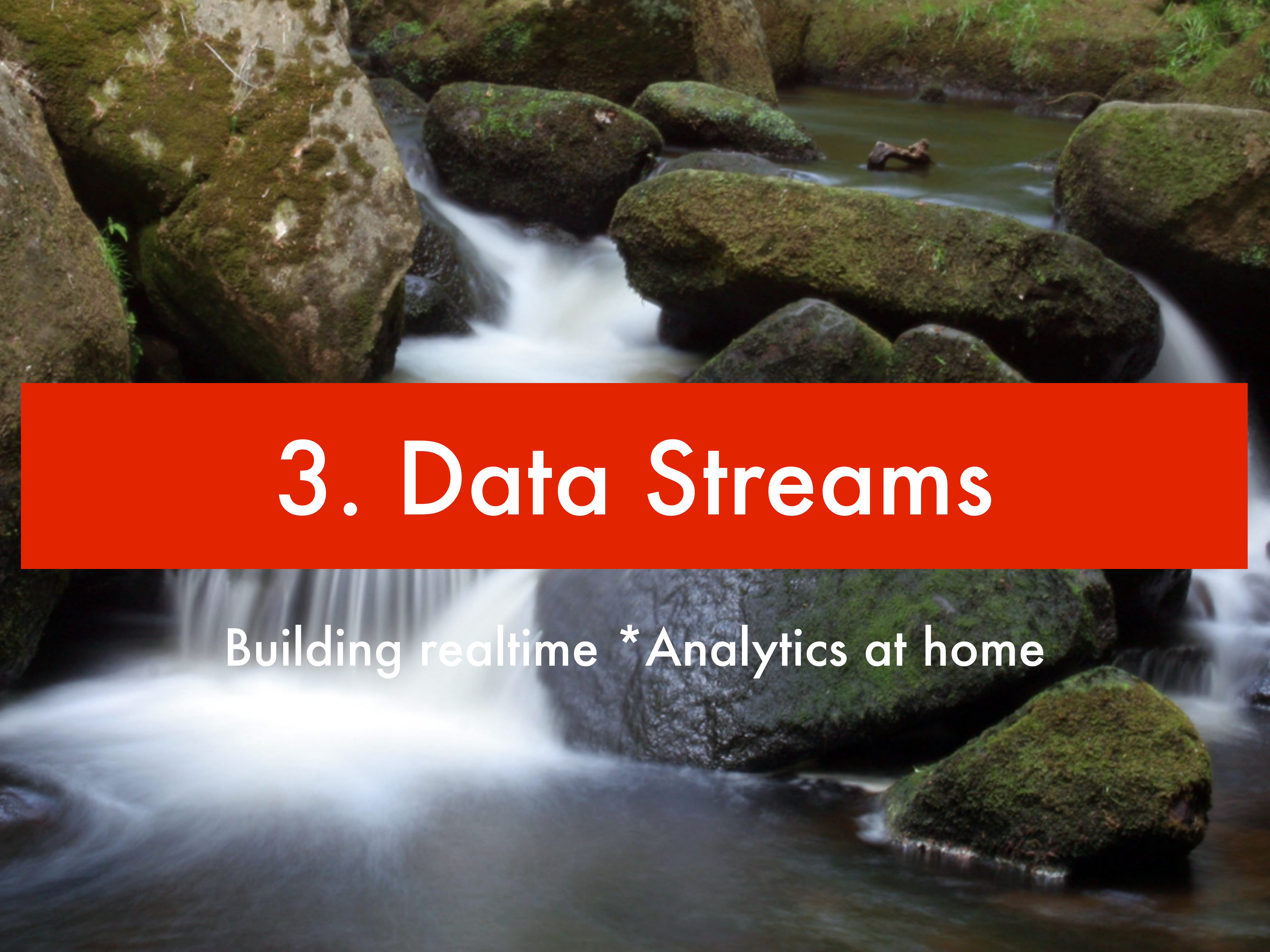
Scalable Machine Learning

3. Data Streams

Alex Smola
Yahoo! Research and ANU

http://alex.smola.org/teaching/berkeley2012
Stat 260 SP 12



3. Data Streams

Building realtime *Analytics at home

Data Streams

Data & Applications

- Moments
  - Flajolet-Martin counter
  - Alon-Matias-Szegedy sketch
- Heavy hitter detection
  - Lossy counting
  - Space saving
- Semiring statistics
  - Bloom filter
  - CountMin sketch
- Realtime analytics
  - Fault tolerance and scalability
  - Interpolating sketches

3.1 Streams

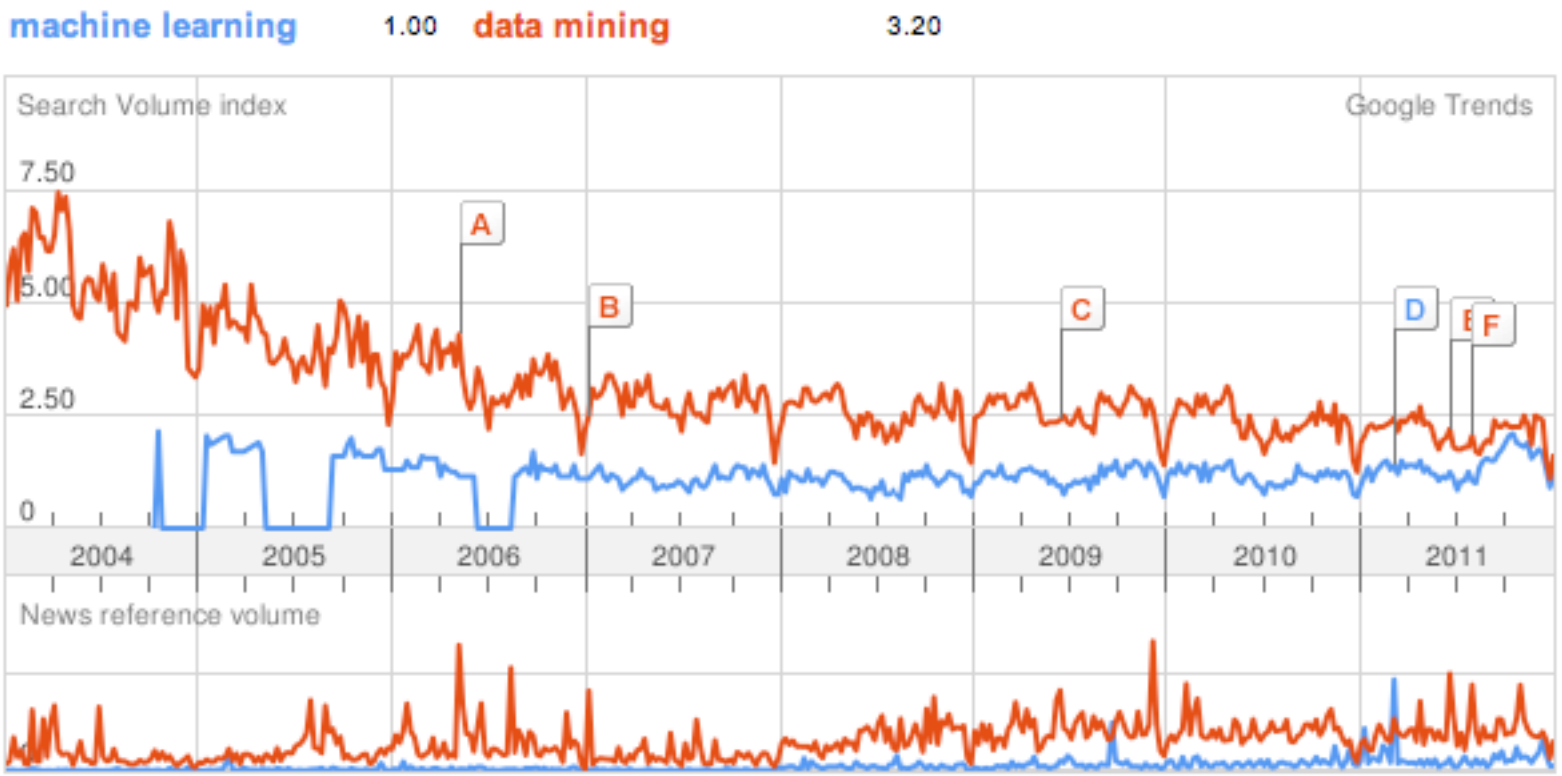
Data Streams

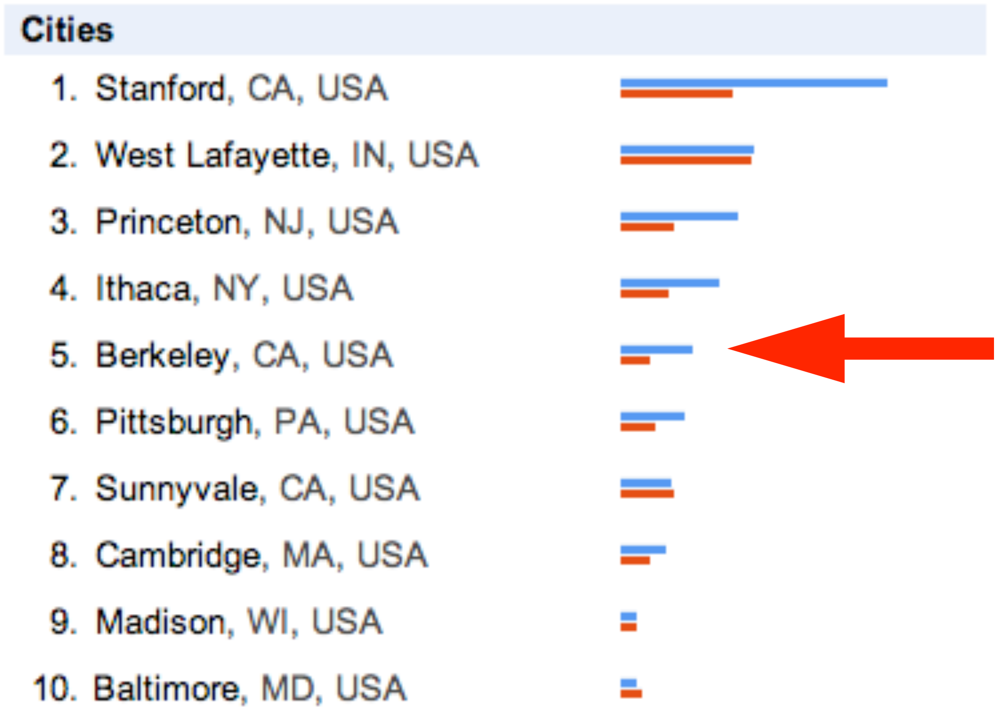
- Cannot replay data
- Limited memory / computation / realtime analytics
- Time series
  Observe instances (x_t, t)
  stock symbols, acceleration data, video, server logs, surveillance
- Cash register
  Observe instances x_i (weighted), always positive increments
  query stream, user activity, network traffic, revenue, clicks
- Turnstile
  Increments and decrements (possibly require nonnegativity)
  caching, windowed statistics

Website Analytics

- Continuous stream of users (tracked with cookie)
- Many sites signed up for analytics service
- Find hot links / frequent users / click probability / right now

Query Stream





- Item stream
- Find heavy hitters
- Detect trends early (e.g. Obsama bin Laden killed)
- Frequent combinations (cf. frequent items)
- Source distribution
- In real time

Network traffic analysis

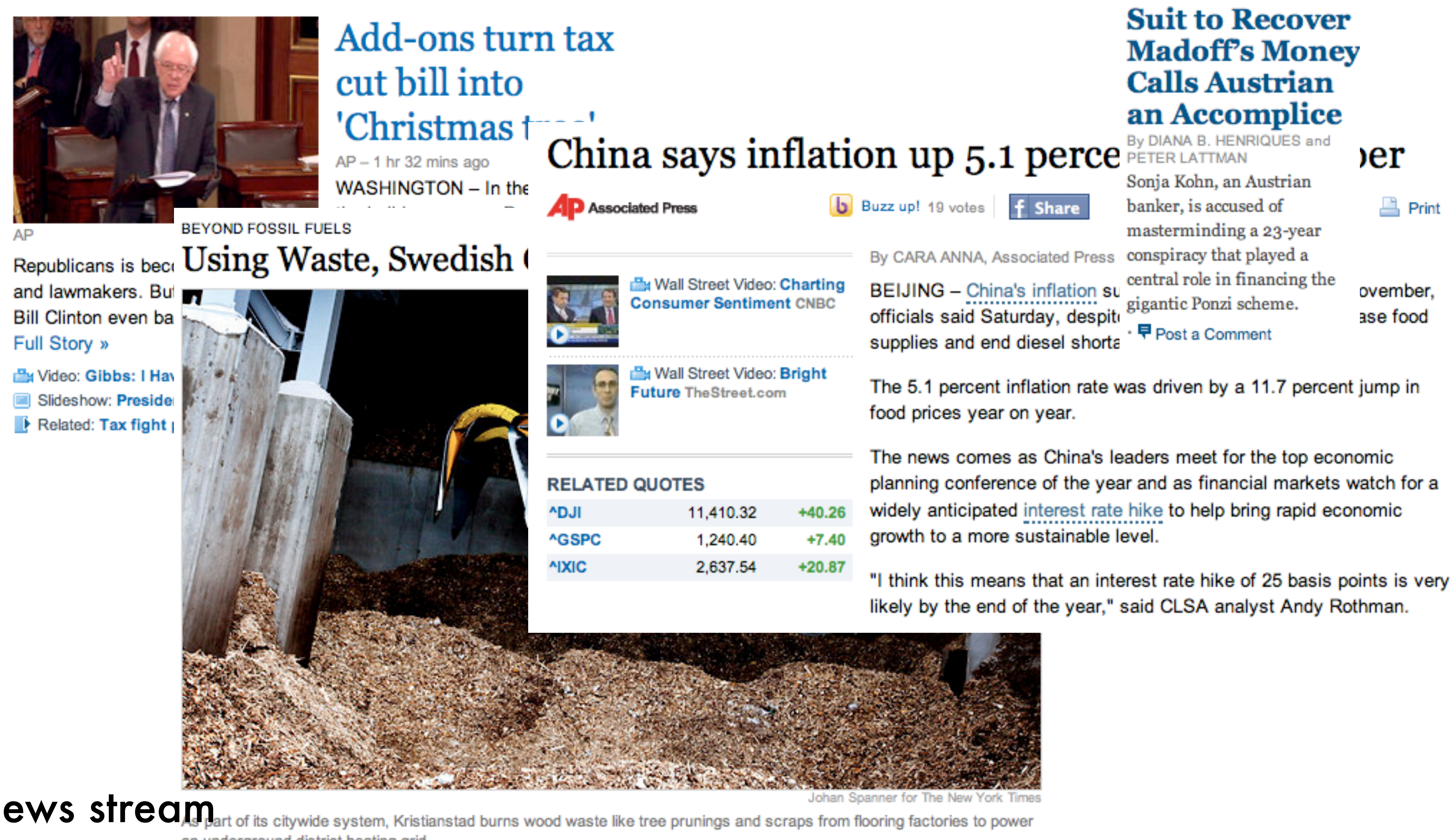
- TCP/IP packets
- On switch with limited memory footprint
- Realtime analytics
- Busiest connections
- Trends
- Protocol-level data
- Distributed information gathering

Financial Time Series

- real time prediction
- missing data
- metadata (news, quarterly reports, financial background)

- time-stamped data stream
- multiple sources
- different time resolution

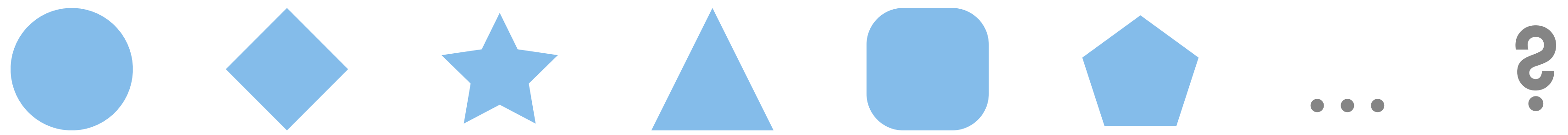
News



- Realtime news stream
  - Multiple sources (Reuters, AP, CNN, ...)
  - Same story from multiple sources
  - Stories are related

3.2 Moments

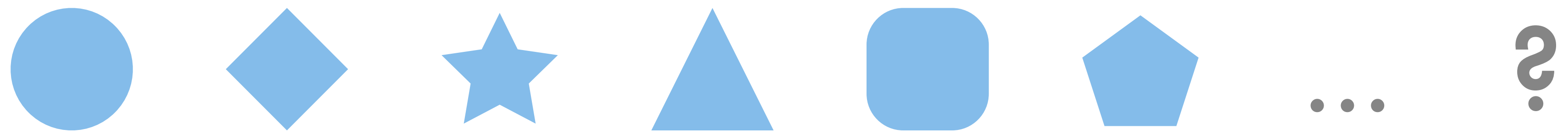
Warmup



- Stream of m items x_i
- Want to compute statistics of what we've seen

- Small cardinality n
  - Trivial to compute aggregate counts (dictionary lookup)
  - Memory is O(n)
  - Computation is O(log n) for storage & lookup

Warmup



- Stream of m items x_i
- Want to compute statistics of what we've seen

- Small cardinality n
  - Trivial to compute aggregate counts (dictionary lookup)
  - Memory is O(n)
  - Computation is O(log n) for storage & lookup
- Large cardinality n
  - Exact storage of counts impossible
  - Exact test for previous occurrence impossible
- Need approximate (dynamic) data structure

Finding the missing item

- Sequence of instances [1..N]
- One of them is missing
- Identify it

Finding the missing item

- Sequence of instances [1..N]
- One of them is missing
- Identify it

- Algorithm
  - Compute sum $s := \sum_{i=1}^{N} i$
  - For each item decrement s via $s \leftarrow s - x_i$
  - At the end identify missing item

Finding the missing item

- Sequence of instances [1..N]
- One of them is missing
- Identify it

- Algorithm
  - Compute sum $s := \sum_{i=1}^{N} i$
  - For each item decrement s via $s \leftarrow s - x_i$
  - At the end identify missing item

- We only need least significant log N bits

Finding the missing item

- Sequence of instances [1..N]
- Up to k of them are missing
- Identify them

Finding the missing item

- Sequence of instances [1..N]
- Up to k of them are missing
- Identify them

- Algorithm
  - Compute sum for p up to k $s_p := \sum_{i=1}^{N} i^p$
  - For each item decrement all s_p via $s_p \leftarrow s_p - x_i^p$
  - Identify missing item by solving polynomial system

- We only need least significant log N bits

Estimating F_k

Moments

- Characterize the skewness of distribution
  - Sequence of instances
  - Instantaneous estimates

$$F_p := \sum_{x \in \mathcal{X}} n_x^p$$

- Special cases
  - F_0 is number of distinct items
  - F_1 is number of items (trivial to estimate)
  - F_2 describes ‘variance’ (used e.g. for database query plans)

Flajolet-Martin counter

- Assume perfect hash functions (simplifies proof)
- Design hash with $\Pr(h(x) = j) = 2^{-j}$

log n bits

0	1	0	0	1	1	0	0	→ 0
1	0	0	1	1	0	1	1	→ 2
0	0	1	0	1	1	1	1	→ 4

- Position of the rightmost 0 (LSB is position 1)
- CDF for maximum over n items $F(j) = (1 - 2^{-j})^n$
  (CDF of maximum over n random variables is F^n)

Flajolet-Martin counter

◆	0	1	0	0	1	1	0	0	→ 0
★	1	0	0	1	1	0	1	1	→ 2
▲	0	0	1	0	1	1	1	1	→ 4

- Intuitively expect that $\max_{x \in \mathcal{X}} h(j) \approx \log |\mathcal{X}|$
- Repetitions of same element do not matter
- Need O(log log |X|) bits to store counter
- High probability bounding range

$$\Pr\left(\left|\max_{x \in \mathcal{X}} h(j) - \log |\mathcal{X}|\right| > \log c\right) \leq \frac{2}{c}$$

Proof (for a version with 2-way independent hash functions see Alon, Matias and Szegedy)

- Upper bound trivial

  $$|\mathcal{X}| \cdot 2^{-j} \leq \frac{1}{c} \Longrightarrow 2^j \geq c|\mathcal{X}|$$

  With probability at most 1/c the upper bound is exceeded (using union bound)

- Lower bound
  - Probability of not exceeding j is bounded by

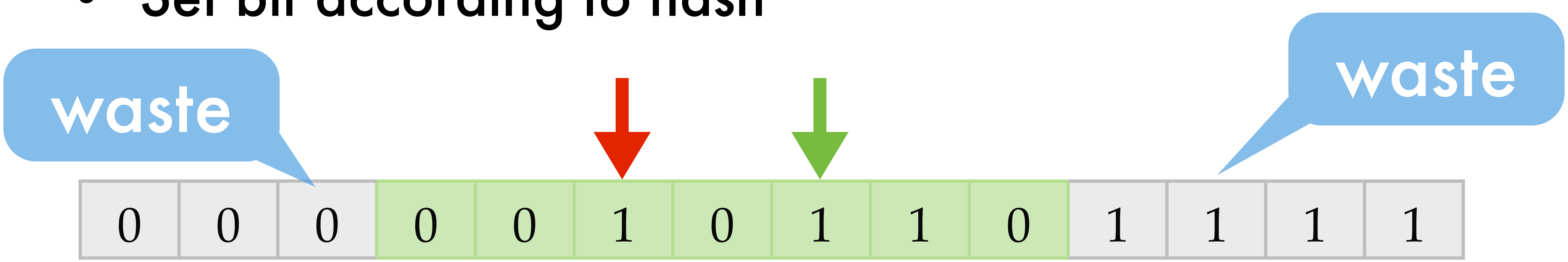
    $$(1 - 2^{-j})^{|\mathcal{X}|} \leq \exp\left(|\mathcal{X}| \cdot 2^{-j}\right) \leq \frac{1}{c} \leq e^{-c}$$

    Solve for j to obtain

    $$2^j \geq \frac{|\mathcal{X}|}{c}$$

Variations on FM counter

- Lossy counting
  - Increment counter j to c with probability p^{-c} for $p<0.5$
  - Yields estimate of log-count (normalization!)
- FM instead of bits inside Bloom filter ... more later
- log n rather than log log n array
  - Set bit according to hash



  - Count consecutive 1 instead of largest bit and fill gaps.
- The log log bounds are tight (see AMS lower bound)

Computing F_2

- Strategy
  - Design random variable with $$\mathbf{E}[X_{ij}] = F_2$$
  - Take average over subsets $$\bar{X}_i := \frac{1}{a}\sum_{j=1}^{a} X_{ij}$$
  - Estimate is median $$\bar{X} := \mathrm{med}\left[\bar{X}_1, \ldots, \bar{X}_b\right]$$
- Random variable $$X_{ij} := \left[\sum_{x \in \mathrm{stream}} \sigma(x, i, j)\right]^2$$
  - σ is Rademacher hash with equiprobable $\{\pm 1\}$
  - In expectation all cross terms cancel out yielding F_2

Average-Median Theorem

- Random variables X_{ij} with mean μ, variance σ
- Mean estimate $\bar{X}_i := \frac{1}{a}\sum_{j=1}^{a} X_{ij}$ and $\bar{X} := \mathrm{med}\left[\bar{X}_1, \ldots, \bar{X}_b\right]$

- The probability of deviation is bounded by

$$\Pr\left\{|\bar{X} - \mu| \geq \epsilon\right\} \leq \delta \text{ for } a = 8\sigma^2\epsilon^{-2} \text{ and } b = -\frac{8}{3}\log\delta$$

- Note - Alon, Matias & Szegedy claim $b = -2\log\delta$ but the Chernoff bounds don't work out AFAIK

Proof

- Bounding the mean
  Pick $a = 8\sigma^2\epsilon^{-2}$ and apply Chebyshev bound to see that $\Pr\left\{|\bar{X}_i - \mu| > \epsilon\right\} \leq \frac{1}{8}$
- Bounding the median
  - Ensure that for at least half $\bar{X}_i$ deviation is small
  - Failure probability is at most 1/8
  - Chernoff (Mitzenmacher & Upfahl Theorem 4.4)
    $$\Pr\left\{x \geq (1+\delta)\mu)\right\} \leq e^{-\frac{\mu\delta^2}{3}}$$
    Plug in
    $$\epsilon = 3; \mu = \frac{b}{8} \text{ hence } \delta \leq \exp\left(-\frac{3b}{8}\right) \text{ and } b \leq -\frac{8}{3}\log\delta$$

Computing F_2

- Mean

$$\mathbf{E}\left[X_{ij}\right] = \mathbf{E}\left[\sum_{x \in \text{stream}} \sigma(x,i,j)\right]^2 = \mathbf{E}\left[\sum_{x \in \mathcal{X}} \sigma(x,i,j) n_x\right]^2 = \sum_{x \in \mathcal{X}} n_x^2$$

- Variance

$$\mathbf{E}\left[X_{ij}^2\right] = \mathbf{E}\left[\sum_{x \in \text{stream}} \sigma(x,i,j)\right]^4 = 3 \sum_{x,x' \in \mathcal{X}} n_x^2 n_{x'}^2 - 2 \sum_{x \in \mathcal{X}} n_x^4$$

$$\mathbf{E}\left[X_{ij}^2\right] - \left[\mathbf{E}\left[X_{ij}\right]\right] 2 = 2 \sum_{x,x' \in \mathcal{X}} n_x^2 n_{x'}^2 - 2 \sum_{x \in \mathcal{X}} n_x^4 \leq 2F_2^2$$

- Plugging into the Average-Median theorem shows that algorithm uses $O(\epsilon^{-2} \log(1/\delta) \log |\mathcal{X}| n)$ bits

Computing F_k in general

- Random variable with expectation F_k
  - Pick uniformly random element in sequence
  - Start counting instances until end

	a	s	r	a	n	d	o	m	a	s	c	a	n	b	e
3				1					2			3			
1											1				

  - Use count r_{ij} for

$$X_{ij} = m\left(r_{ij}^k - (r_{ij} - 1)^k\right)$$

  - Apply the Average-Median theorem

More F_k

- Mean via telescoping sum

$$\mathbf{E}[X_{ij}] = \Big[1^k + (2^k - 1^k) + \ldots + (n_1^k - (n_1 - 1)^k) + \ldots + (n_{|\mathcal{X}|}^k - (n_{|\mathcal{X}|} - 1)^k)\Big] = \sum_{x \in \mathcal{X}} n_x^k = F_k$$

- Variance by brute force algebra

$$\mathrm{Var}\left[X_{ij}\right] \leq \mathbf{E}\left[X_{ij}\right] \leq k|\mathcal{X}|^{1-1/k} F_k^2$$

- We need at most $O(k|\mathcal{X}|^{1-1/k}\epsilon^{-2}\log 1/\delta(\log m + \log|\mathcal{X}|)$ bits to estimate F_k. The rate is tight.

More F_k

- Mean via telescoping sum

$$\mathbf{E}[X_{ij}] = \Big[1^k + (2^k - 1^k) + \ldots + (n_1^k - (n_1 - 1)^k) + \ldots + (n_{|\mathcal{X}|}^k - (n_{|\mathcal{X}|} - 1)^k)\Big] = \sum_{x \in \mathcal{X}} n_x^k = F_k$$

no better than brute force for large k

- Variance by brute force algebra

$$\mathrm{Var}\left[X_{ij}\right] \leq \mathbf{E}\left[X_{ij}\right] \leq k|\mathcal{X}|^{1-1/k} F_k^2$$

- We need at most $O(k|\mathcal{X}|^{1-1/k}\epsilon^{-2}\log 1/\delta(\log m + \log|\mathcal{X}|)$ bits to estimate F_k. The rate is tight.

Uniform sampling

Subsampling a stream

- Incoming data stream
- Draw item uniformly from support X
- But we don't know X

- Initialize c = 0 and g = ∞
- Observe x from stream
- If $h(x) = g$ increment $c = c + 1$
- If $h(x) < g$ set $c = 1$ and $g = h(x)$

Subsampling a stream

- Analysis
  - Hash function assigns random value
  - Probability that x has smallest hash is $\frac{1}{|\mathcal{X}|}$ (ignoring collisions)
  - Once we find it we count all occurrences
- Extension
  - Keep count of items with k smallest hashes
  - Reject duplicates
  - Use the hashID to get a handle on domain (see papers by Li, Hastie, Church; Broder's shingles)
  - Alternative estimate for F_k (but higher variance)

3.3 Heavy Hitters

Heavy Hitter Detection

- Data stream
- Find k heaviest items
  - For arbitrary sequence
  - Take advantage of power-law distribution if it exists (automatically)
  - Use O(k) space and O(1/k) accuracy
- Applications
  - Advertising (find frequent clickers, popular ads)
  - News (popular keywords, trending terms)
  - Web search (popular queries)
  - Network security (detect attacks, heavy resource users)

Space-Saving Algorithm

- Initialize k pairs $(\text{count}_i = 0, \text{label}_i = \emptyset)$ in list T
- observe x
  - if x is in label set of T
    increment counter $\text{count}_i = \text{count}_i + 1$
  - else locate label with lowest count
    update its $\text{count}_i = \text{count}_i + 1$ and set $\text{label}_i = x$

	(a,4)	(b,4)	(c,2)	(d,2)
e	(a,4)	(b,4)	(e,3)	(c,2)
b	(b,5)	(a,4)	(e,3)	(c,2)
f	(b,5)	(a,4)	(e,3)	(f,3)

Space-Saving Algorithm

- Initialize k pairs $(\text{count}_i = 0, \text{label}_i = \emptyset)$ in list T
- observe x
  - if x is in label set of T
    increment counter $\text{count}_i = \text{count}_i + 1$
  - else locate label with lowest count
    update its $\text{count}_i = \text{count}_i + 1$ and set $\text{label}_i = x$
- Trivial to implement e.g. with a Boost.Bimap
  http://www.boost.org/doc/libs/1_48_0/boost/bimap/bimap.hpp
  Provides list sorted by two indices (label&count)

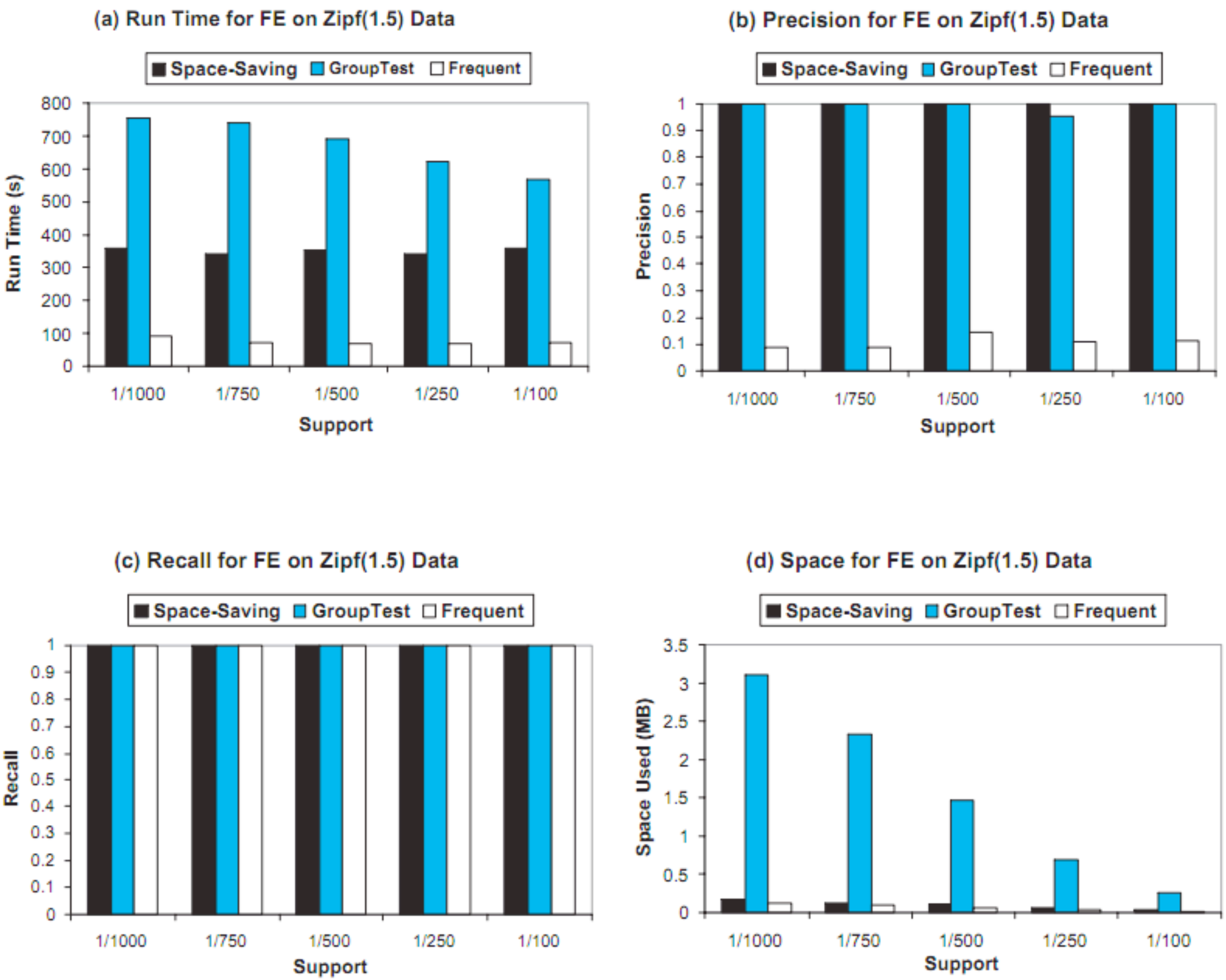
Guarantees

1. Error is bounded by $n_x \leq \mathrm{count}_x \leq n_x + \frac{n}{k}$
2. In fact, bound is even tighter - smallest counter
3. In fact, bound is even tighter

$$n_x \leq \mathrm{count}_x \leq n_x + \frac{F_1^{(k)}}{n-k} \text{ where } F_1^{(k)} = \sum_{i>k} n_i$$

4. In fact, the rate is optimal
5. Estimate at position i majorizes i^{th} true count
6. Inserting more 'head' items does not increase approximation error*

It works well, too



from Metwally, Agrawal, El Abbadi 2005

Proof

1. Error is bounded by $n_x \leq \mathrm{count}_x \leq n_x + \dfrac{n}{k}$

- At each step counter increments by 1
- k bins, so smallest bin smaller than n/k

2. Insert error bounded by smallest element in list
6. observing an element already in the list doesn't increase the error

- if we observe, drop, and then observe again, count only increases (so always upper bound)

Proof

4. Rate is optimal

- Deterministic algorithm tracking k counters
- Feed it two sequences S{a} and S{b}
  - Assume that {a} was never observed before
  - Assume that {b} is not being tracked. Can always make its frequency O(n/k)
- Since {b} isn't tracked, algorithm cannot distinguish it from {a}
- It must output same estimate for {a} and {b}.
- This forces an O(n/k) error
- Optimality proof for $F_1^{(k)}$ more tricky (see Berinde et al.)

Proof

7. Any item with count n_x larger than smallest count in T must be in array
- Assume it isn't
- At last occurrence it must have been inserted
- Counter in array is upper bound
- Hence it cannot have been removed

5. Count at position i majorizes i^{th} frequency
   a. Item is not in array. Hence smallest element in list must be larger.
   b. Item at position i. OK by upper bounding property.
   c. Item at position $j > i$. OK by fact that we have sorted list.
   d. Item at position $j < i$. Hence there must be counter k that has higher rank and is at or below position i. Monotonicity proves the claim.

Proof

3. Even tighter bound

$$n_x \le \mathrm{count}_x \le n_x + \frac{F_1^{(k)}}{n-k} \text{ where } F_1^{(k)} = \sum_{i>k} n_i$$

- Residual sum after first k terms must be upper bounded by $F_1^{(k)}$ due to property 5.
- Smallest element at most as large as average over residual bins.

More sketches

- Lossy counting (Manku & Motwani)
  - Keep list with confidence bounds
  - At each k observations eliminate items which are below accuracy threshold
  - New items are inserted with lose confidence
- Frequent (see e.g. Berinde et al.)
  - Keep k counters like Space Saving
  - When there’s space, insert new item with count 1
  - When counters full and new element occurs, decrement all counters by 1
- This yields a lower bound on item frequencies

Some (research) problems

- Distributed sketch generation
  - Each box receives fraction of realtime stream
  - Fault tolerant setup (what if a machine dies)
  - Improved accuracy with more machines
- Temporal attributes
  - Query for a given time interval
  - Compression over time
- Frequent item combinations

3.4 Semiring Statistics

Bloom filters

BLOOM.COM
SOCIAL BEAUTY STORE

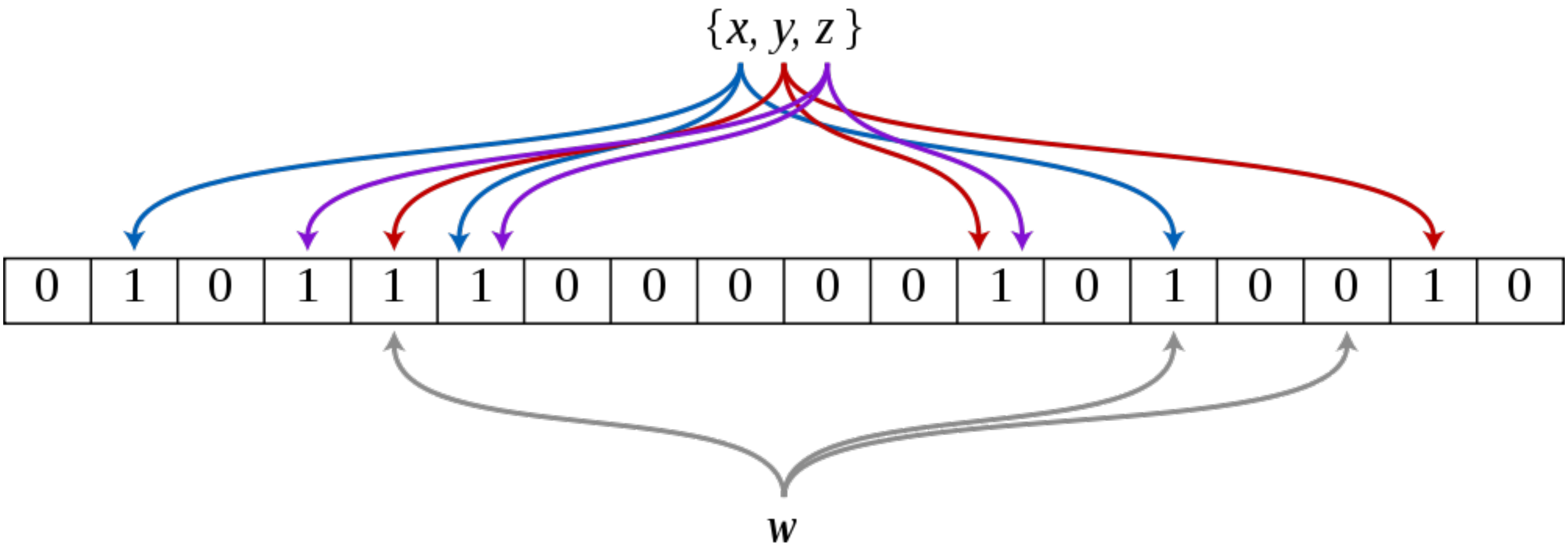
Beyond Heavy Hitters

- Check for previously seen items
  - but don’t need to have counts, just existence
- Check for frequency estimate
  - but don’t want to store labels
  - but want estimate for all items (not just HH)
  - but want to be able to aggregate
  - but want turnstile computation

Bloom filter, Count-Min sketch, Counter braids

Bloom Filter

- Bit array b of length n
  - insert(x): for all i set bit b[h(x,i)] = 1
  - query(x): return TRUE if for all i b[h(x,i)] = 1



Bloom Filter

- Bit array b of length n
  - insert(x): for all i set bit b[h(x,i)] = 1
  - query(x): return TRUE if for all i b[h(x,i)] = 1
- Only returns TRUE if all k bits are set
- No false negatives but false positives possible
  - Probability that an arbitrary bit is set

$$\Pr\left\{b[i] = 1\right\} = 1 - \left(1 - \frac{1}{n}\right)^{mk} \approx 1 - e^{-\frac{mk}{n}}$$

  - Probability of false positive (approx. indep.)

$$\Pr\left\{b[h(x,1)] = \ldots = b[h(x,k)] = 1\right\} \approx \left(1 - e^{-\frac{mk}{n}}\right)^{k}$$

Bloom Filter

- Minimizing k to minimize false positive rate

$$\partial_k \left[k \log \left(1 - e^{-mk/n} \right) \right] = \log \left(1 - e^{-mk/n} \right) + \frac{mk}{n} \frac{e^{-mk/n}}{1 - e^{-mk/n}}$$

  This vanishes for $\frac{mk}{n} = \log 2 \text{ and hence } k = \frac{n}{m} \log 2$ with a false positive rate of 2^{-k}
- More refined analysis & details, e.g. in the Mitzenmacher & Broder 2004 tutorial.
- Matching lower bound shows that Bloom filter is within 1.44 best efficiency.

Cool things to do with a Bloom Filter

- Bloom filter of union of two sets by OR

0	0	1	0	0	1	1	0	1	0	0	0	0	1	0	1	1	1
1	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	1
1	0	1	0	1	1	1	0	1	0	1	1	0	1	0	1	1	1

  - Parallel construction of Bloom filters
  - Time-dependent aggregation
  - Fast approximate set union
    (bitmap operation rather than set manipulation)
- Also use it to halve bit resolution of Bloom filter

Cool things to do with a Bloom Filter

- Set intersection via AND

0	0	1	0	0	1	1	0	1	0	0	0	0	1	0	1	1	1

1	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	1

0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1

  - No false negatives
  - More false positives than building from scratch
- Use bits to estimate size of set union/intersection

$$\Pr\{b=1\} = \Pr\{b=1|S_1\} + \Pr\{b=1|S_2\} - \Pr\{b=1|S_1 \cup S_2\}$$

$$\approx 1 - e^{-\frac{k|S_1|}{m}} - e^{-\frac{k|S_2|}{m}} + e^{-\frac{k|S_1 \cup S_2|}{m}}$$

Counting Bloom Filter

- Plain Bloom filter doesn't allow removal

0	0	1	0	0	1	1	0	1	0	0	0	0	1	0	1	1	1

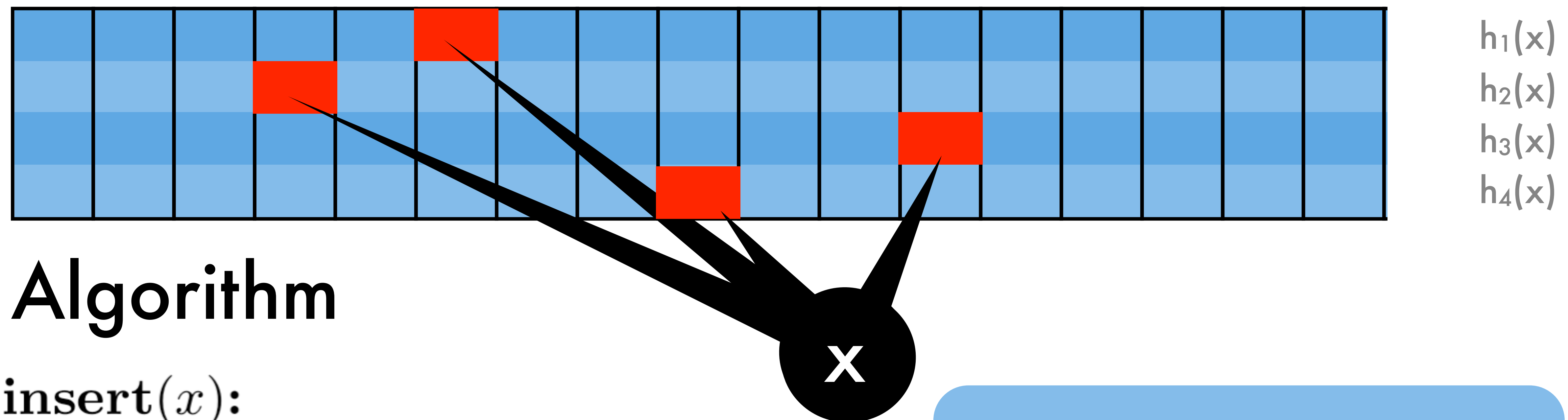
  - insert(x): for all i set bit b[h(x,i)] = 1
    we don't know whether this was set before
  - query(x): return TRUE if for all i b[h(x,i)] = 1
- Counting Bloom filter keeps track of inserts
  - query(x): return TRUE if for all i b[h(x,i)] > 0
  - insert(x): if query(x) = FALSE (don't insert twice)
    for all i increment b[h(x,i)] = b[h(x,i)] + 1
  - remove(x): if query(x) = TRUE (don't remove absents)
    for all i decrement b[h(x,i)] = b[h(x,i)] - 1

only needs log log m bits

Count min sketch

Count min sketch

- Datastructure



- Algorithm

insert(x)**:**
for $i = 1$ **to** d **do**
  $M[i, h_i(x)] \leftarrow M[i, h_i(x)] + 1$
end for

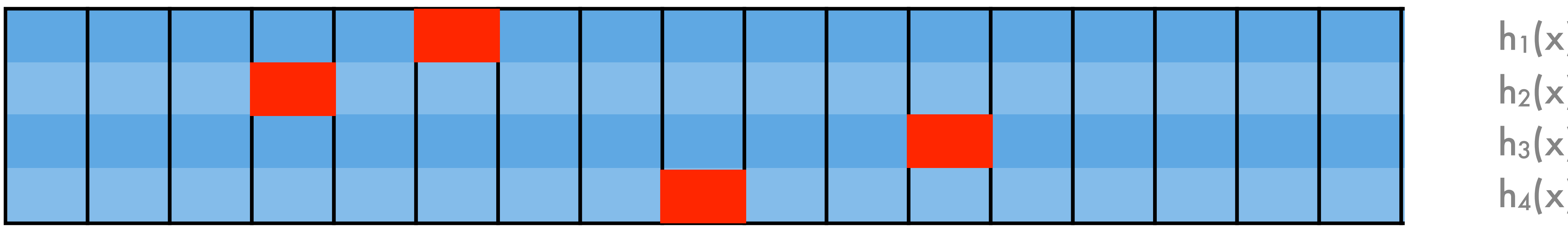
query(x)**:**
$c = \min\{h_i(x) \text{ for all } 1 \leq i \leq d\}$
return c

like Bloom filter but with counters

supports turnstile

Count min sketch

- Datastructure



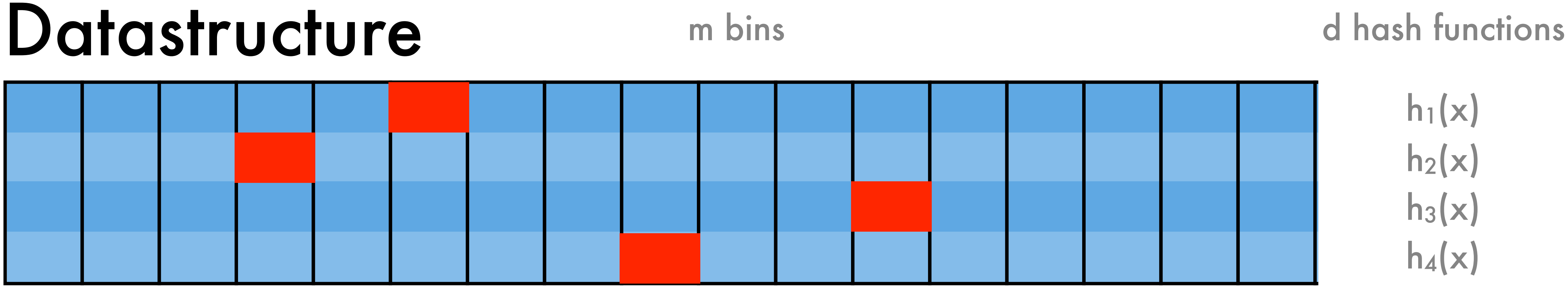
- Guarantees
  - Approximation quality is

    $$n_x \leq c_x \leq n_x + \epsilon \sum_{x'} n_{x'} \text{ for } m = \left\lceil \frac{e}{\epsilon} \right\rceil \text{ with probability } 1 - e^{-d}$$

  - For power law distributions with exponent z we need only $O(\epsilon^{-1/z})$ space (see Cormode & Muthukrishnan)

Proof

- Datastructure



- Lower bound
  - Each bin is updated whenever we see an item
  - So each bin is lower bound, hence min is OK
- Expectation
  - Probability of incrementing a bin at random is 1/m, hence expected overestimate is n/m.

Proof

- Gauss-Markov inequality on random variable

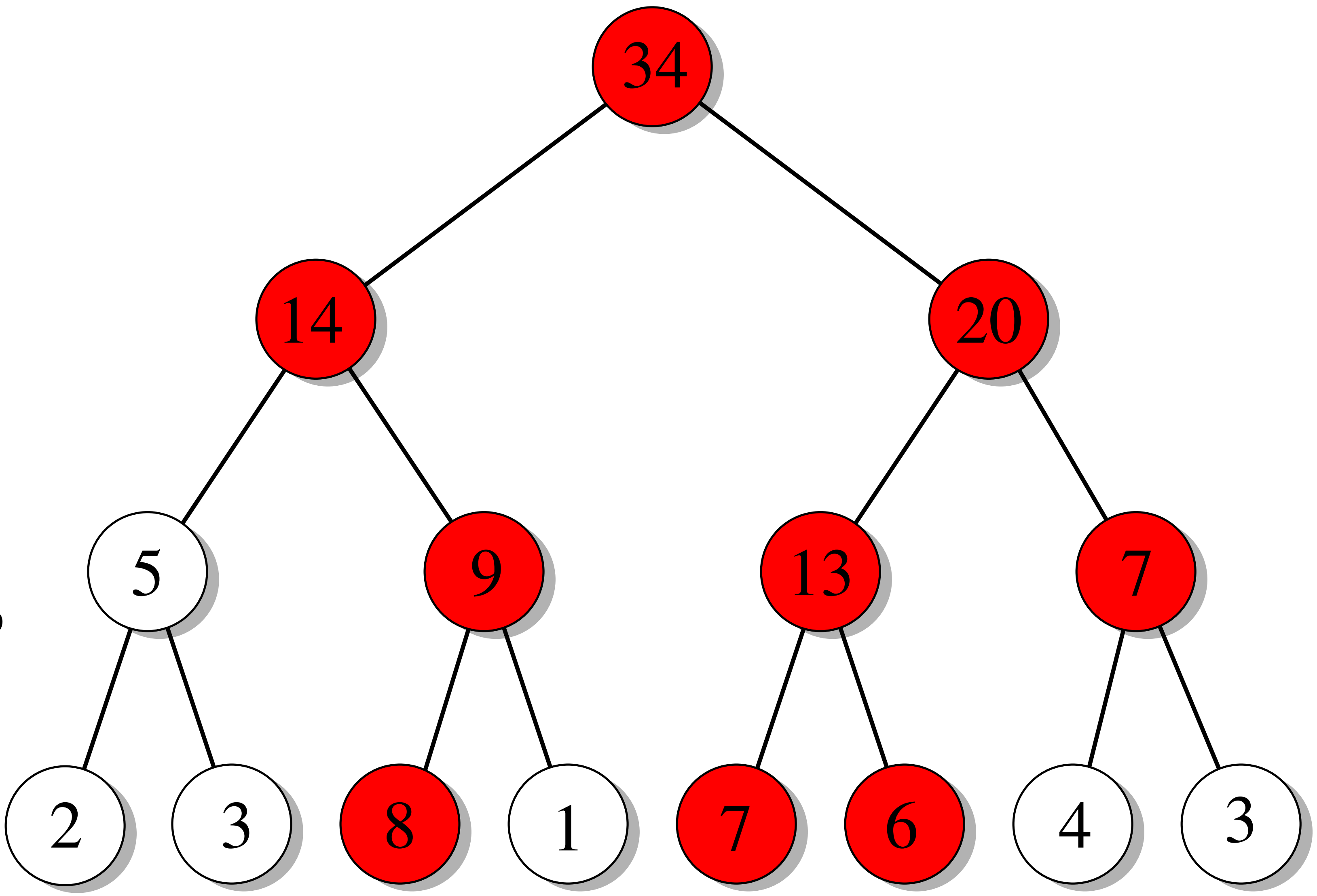
$$\mathbf{E}\left[w[i,h(i,x)] - n_x\right] = \frac{n}{m} \text{ hence } \Pr\left\{w[i,h(i,x)] - n_x > e\frac{n}{m}\right\} \leq e^{-1}$$

- Minimum boosts probability exponentially (only need to ensure that there's at least one random variable which satisfies the condition)

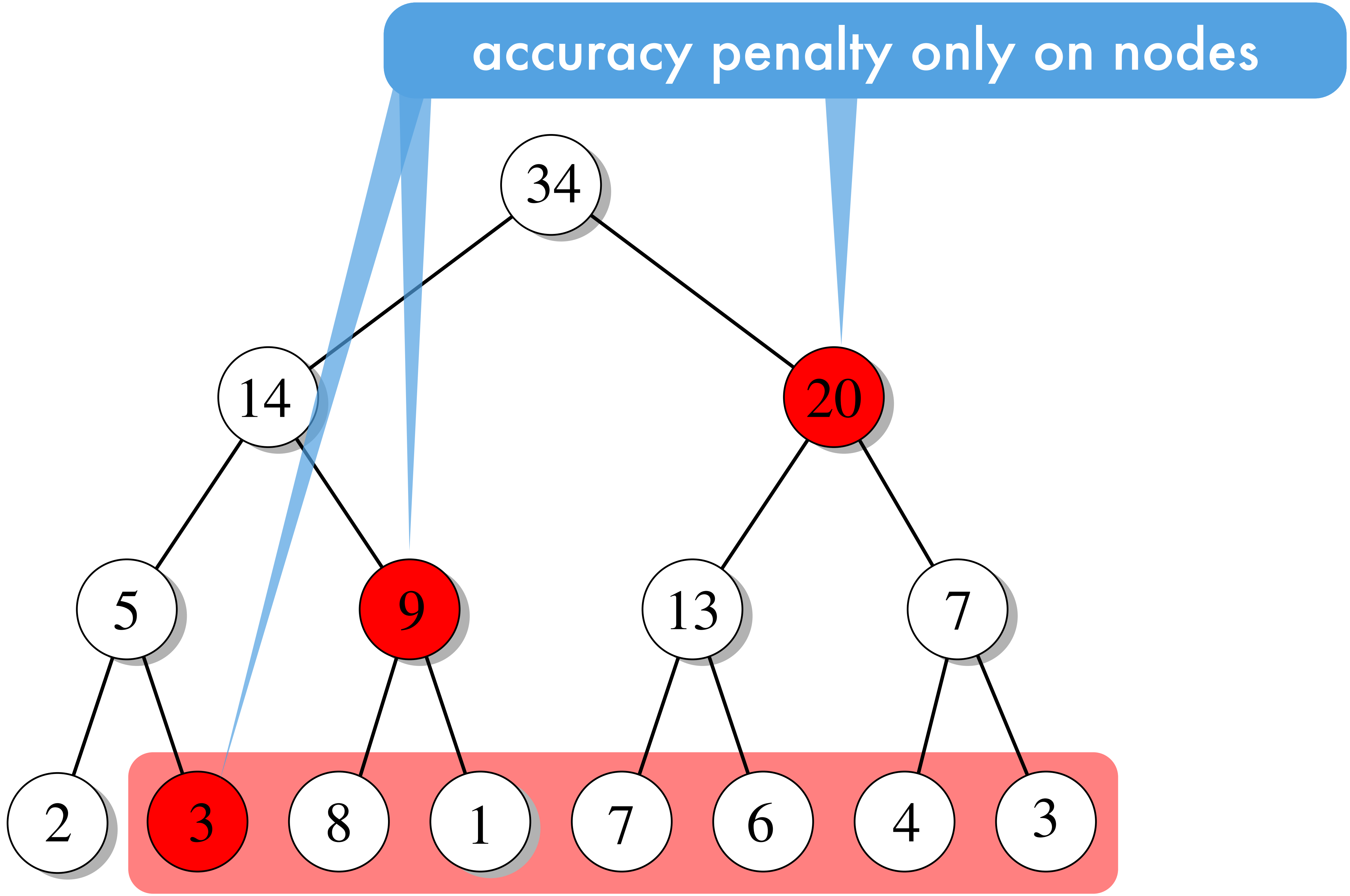
$$\Pr\left\{c_x - n_x > e\frac{n}{m}\right\} \leq e^{-d}$$

Heavy Hitters finding

- Hierarchical event structure
  - IP numbers
  - Prices
  - Activity logs
- Keep top nodes explicitly
- Traverse range via CM sketch



Range query



Tail guarantees

- Zipfian distributions

$$\Pr\{x\} = \frac{c}{(a+x)^z}$$

- Bounding heads/tails (for a = 0 and z > 1)

$$\frac{c_z k^{1-z}}{z-1} \leq \sum_{i=k}^{U} f_i \leq \frac{c_z (k-1)^{1-z}}{z-1}$$

  - only small number of heavy items exists
  - bound heavy hitters separately
  - probability of collision is small
  - tail is small enough for low offset

Tail guarantees

- Set head to m/3 of all bins
- Probability we don't hit head is 2/3 per hash

$$\mathbf{E}[c_x|\text{noheavy}] = n_x + \frac{3}{2m} \sum_{i=k+1, i \neq x}^{\infty} n_i \leq n_x + \frac{n^{-z}}{m} \frac{c_z}{3^z 2(z-1)} \text{ for } k = \frac{n}{3}$$

- Apply Gauss-Markov for 'noheavy' with p=1/2
- Boost residual probability by min operation
- The space needed for Zipfian distribution is $O\left(\epsilon^{-\min\{1,1/z\}} \log 1/\delta\right)$ **with** $\Pr\left\{c_x > n_x + \epsilon n\right\} \leq \delta$

Counter Braids

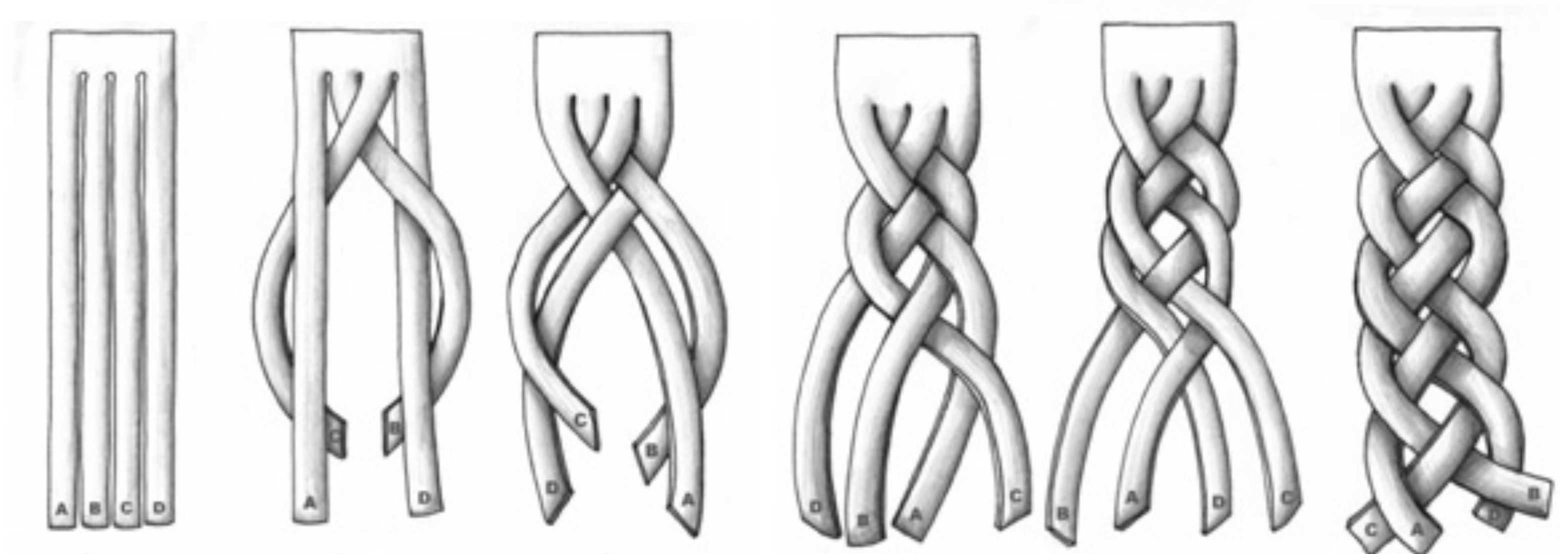
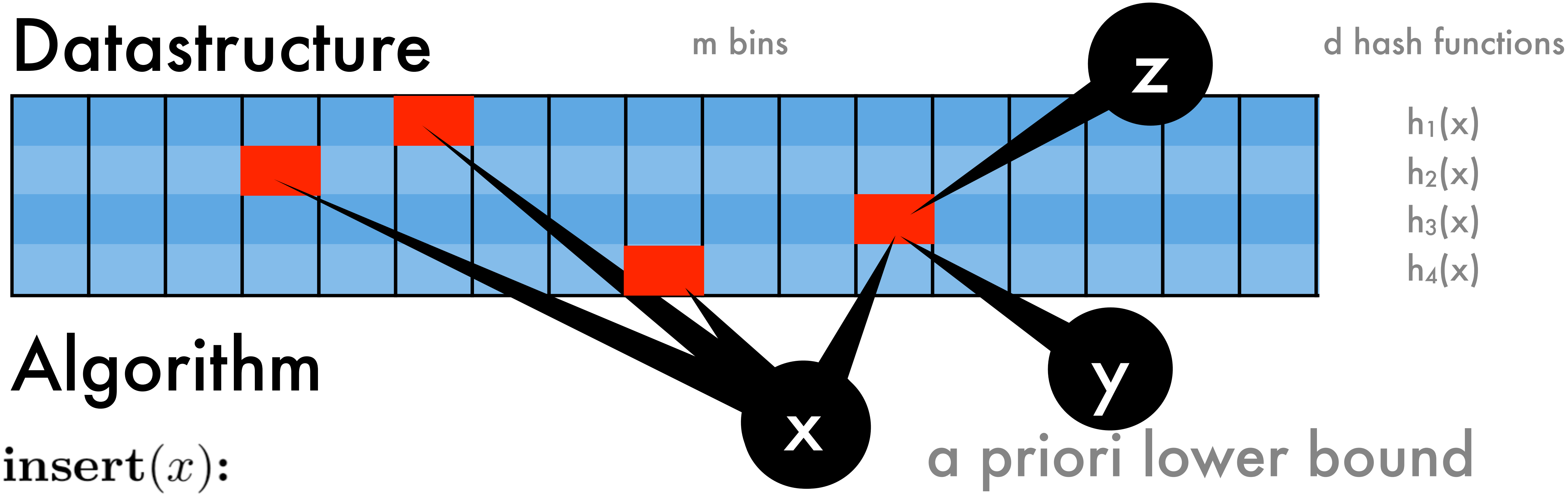


fig. 1 fig. 2 fig. 3 fig. 4 fig. 5 fig. 6

Part A - The Counter

- Datastructure



- Algorithm

insert(x)**:**
for $i = 1$ **to** d **do**
$\quad M[i, h_i(x)] \leftarrow M[i, h_i(x)] + 1$
end for

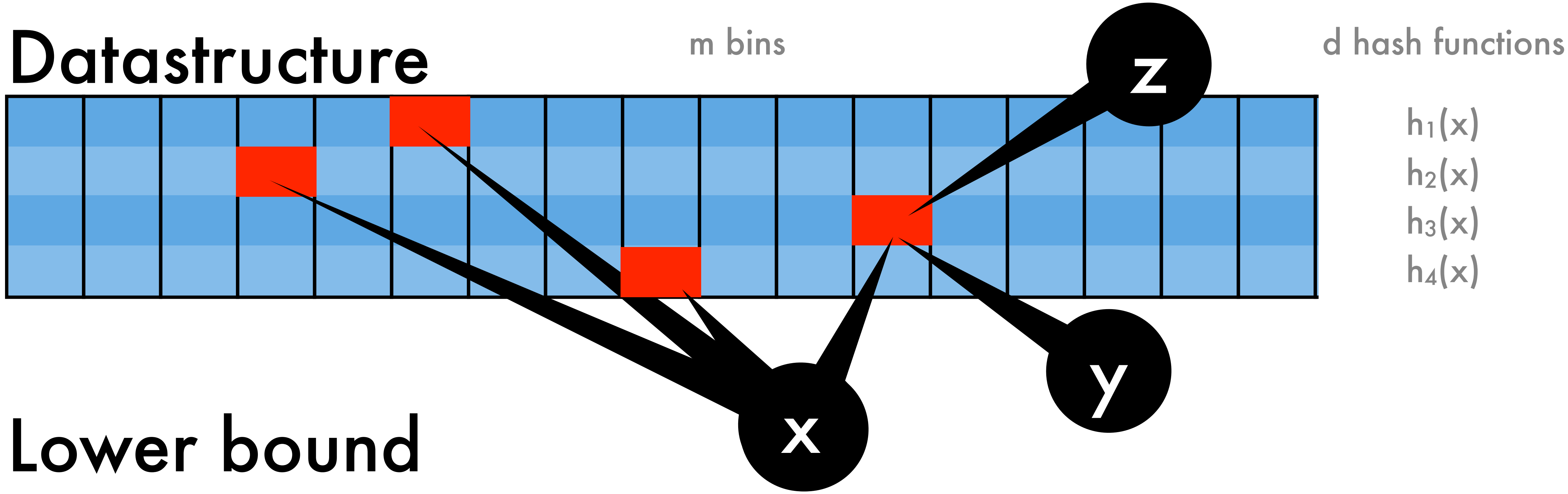
query(x)**:**
$c = \min\{h_i(x) \text{ for all } 1 \leq i \leq d\}$
return c

a priori lower bound on counter is 0

if we know all inserts we can get new lower bound

Part A - The Counter

- Datastructure



- Lower bound

$$w[i,j] = \sum_{h(i,x)=j} n_x \leq \sum_{h(i,x)=j} c_x \text{ hence } c_x \geq l_x := w[i,j] - \sum_{h(i,x)=j, x' \neq x} c_{x'}$$

- Upper bound

$$w[i,j] \geq \sum_{h(i,x)=j} l_x \text{ hence } c_x \leq u_x := w[i,j] - \sum_{h(i,x)=j, x' \neq x} l_{x'}$$

Part A - The Counter

- Iterate lower and upper bounds until converged
  - proof highly nontrivial
  - cheap construction but expensive decoding

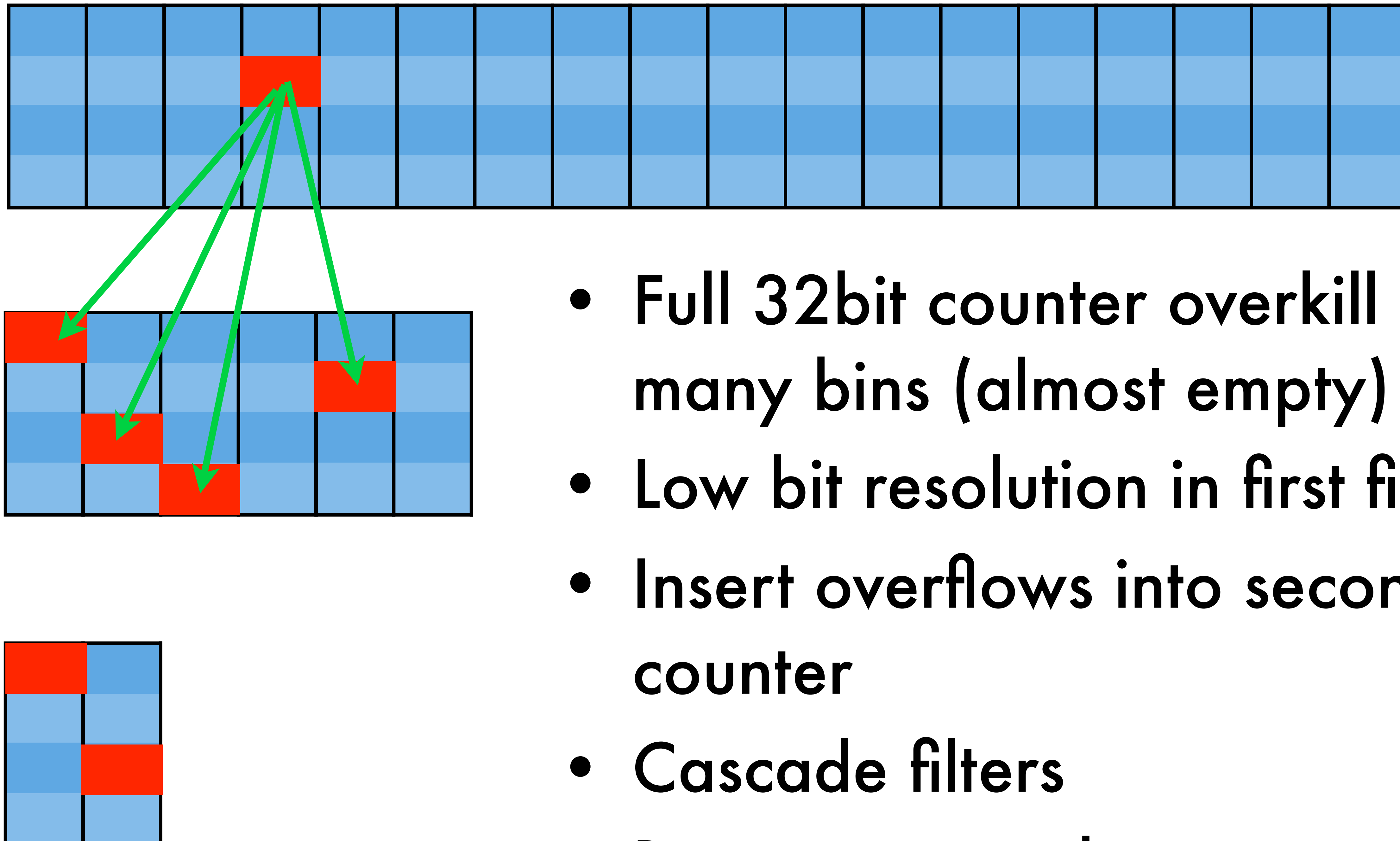
- Lower bound

$$w[i,j] = \sum_{h(i,x)=j} n_x \leq \sum_{h(i,x)=j} c_x \text{ hence } c_x \geq l_x := w[i,j] - \sum_{h(i,x)=j, x' \neq x} c_{x'}$$

- Upper bound

$$w[i,j] \geq \sum_{h(i,x)=j} l_x \text{ hence } c_x \leq u_x := w[i,j] - \sum_{h(i,x)=j, x' \neq x} l_{x'}$$

Part B - The Braid



- Full 32bit counter overkill for many bins (almost empty)
- Low bit resolution in first filter
- Insert overflows into secondary counter
- Cascade filters
- Reconstruction by iteration

3.5 Realtime Analytics

Problems

- How to scale sketches beyond single machine?
  - Accuracy (limited memory)
  - Reliability (fault tolerance)
  - Scalability (more inserts)
- Time series data
  - Limited memory
  - Sequence compression

3 Tools

1. Count min sketch (as before)
   - Provides real-time sketching service (but no time intervals)
2. Consistent hashing
   - Provides load-balancing.
   - Extension to sets provides fault tolerance.
3. Interpolation
   - Marginals of joint distribution
   - Exponential backoff of count statistics

Consistent hashing

Increasing Insert Throughput

$$m(x) := \operatorname*{argmin}_{m \in M} h(m, x)$$

- Consistent hashing (Karger et al.)
  - Split the keys x between a pool of machines M
  - Reproducible
  - Small memory footprint & fast
  - Can be extended to proportional hashing (see Reed, USENIX 2011)

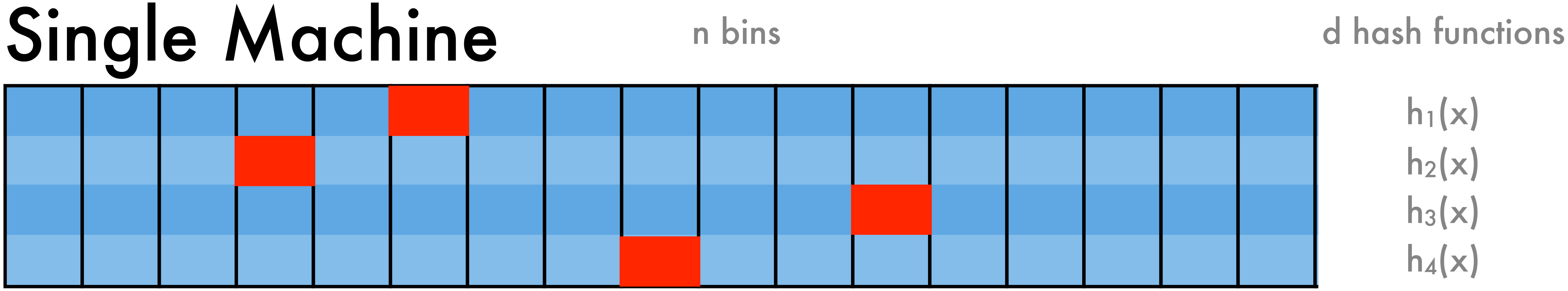
Increasing Insert Throughput

$$m(x) := \operatorname*{argmin}_{m \in M} h(m, x)$$

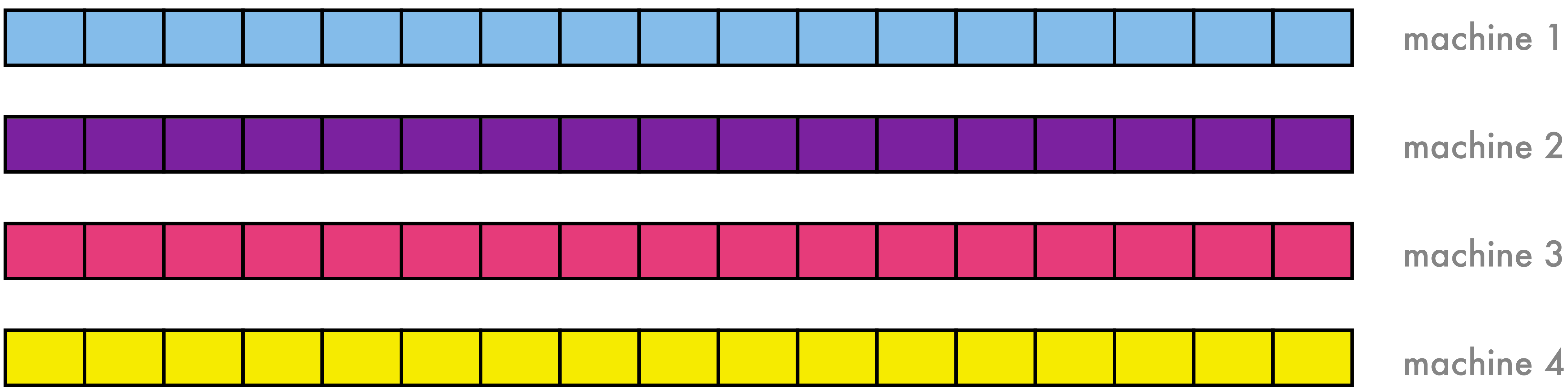
- Accuracy increases with O(1/k)
- Throughput increases with O(k)
- Reliability decreases

Increasing Reliability

- Single Machine



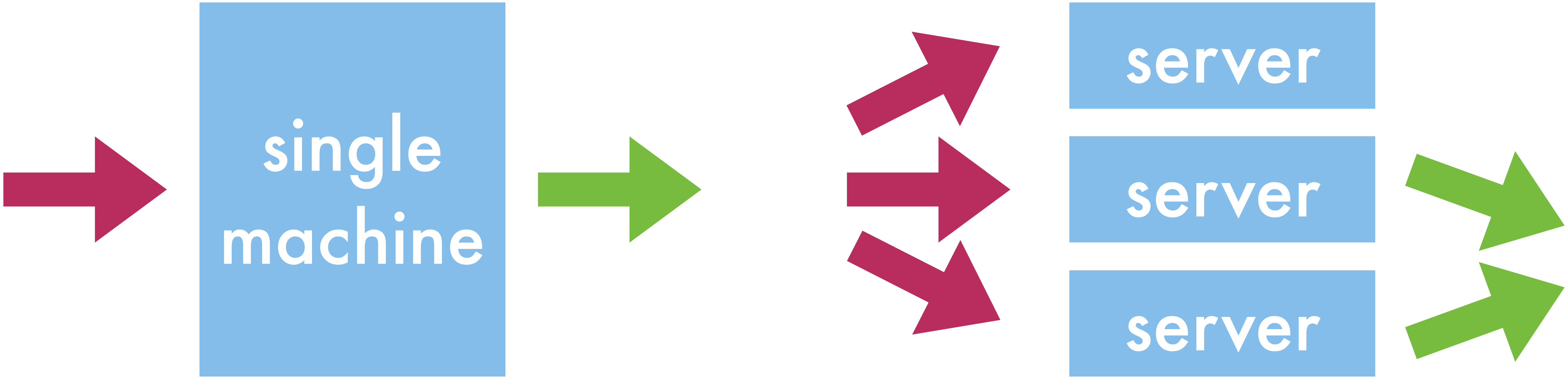
- Multiple Machines



Increased Reliability

- Failure probability decreases exponentially
- Throughput is constant
- Query latency increases
- No acceleration of insert parallelism

Increasing Query Throughput



- Failure probability decreases exponentially (if machine fails we can use others)
- Insert throughput is constant
- Query throughput is O(k)

Putting it all together

- Tricks
  - Assign keys only to a subset of machines
  - Overreplicate for reliability
  - Overreplicate for query parallelism
- Consistent set hashing

$$C(x) := \operatorname*{argmin}_{C \in M \text{ with } |C|=k} \sum_{m \in C} h(m, x)$$

  - Insert into a k machines at a time
  - Request from k' < k machines at a time (use set hashing on C(x) with client ID)

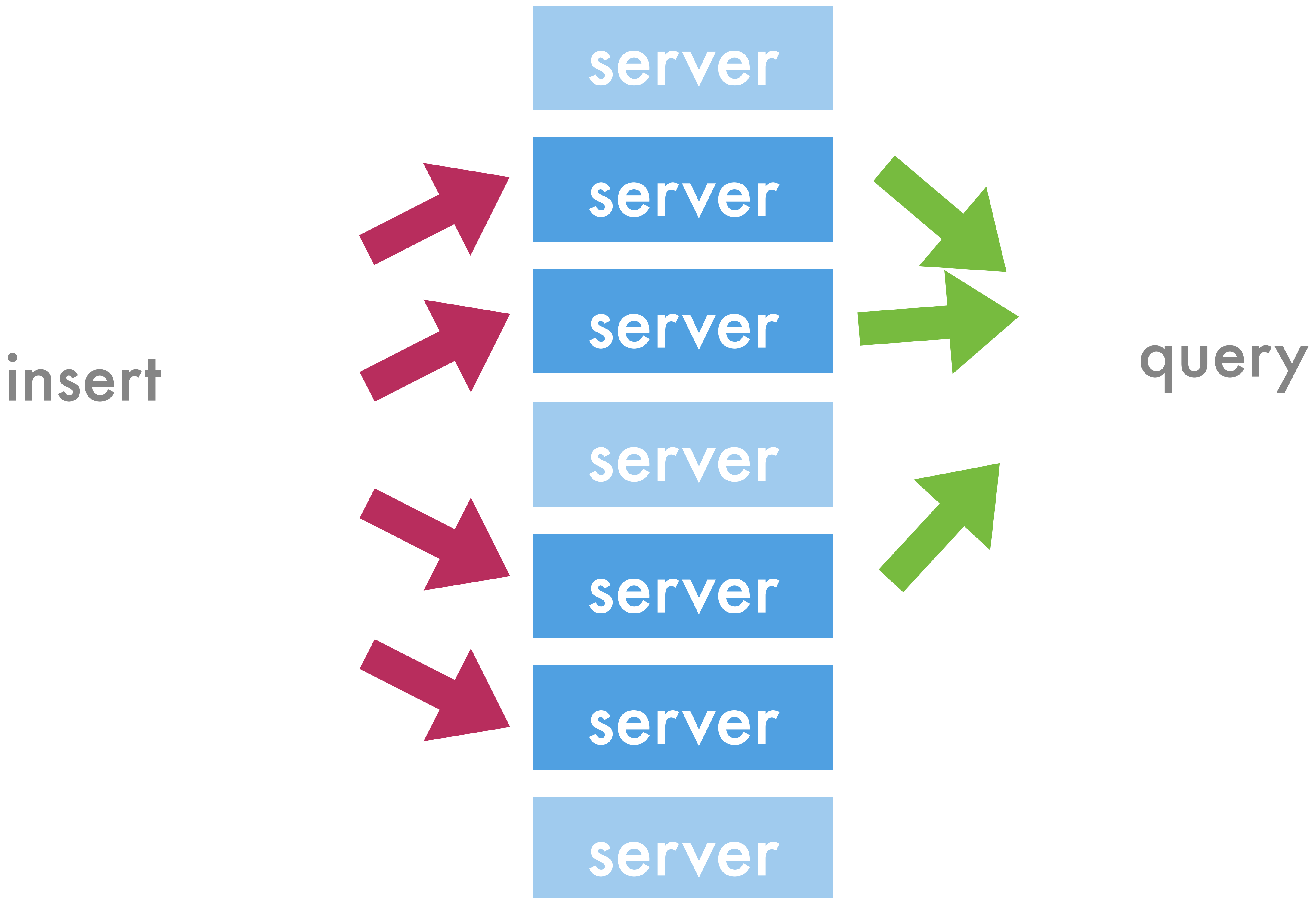
Putting it all together

- Theorem

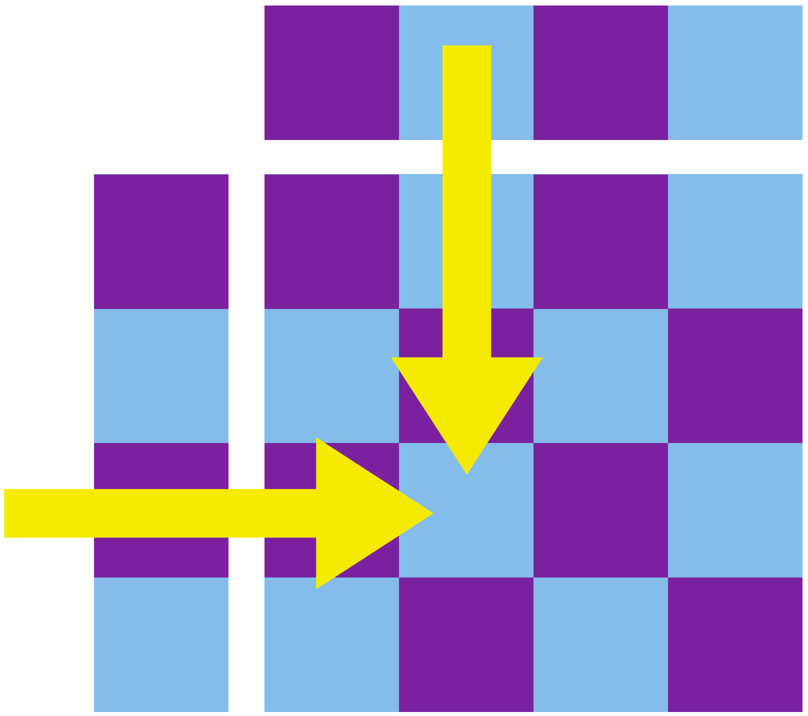
  Assume we have up to f failures among m machines and let $2d < m$. Then we need at most $1.72\, fd/m$ additional inserts over the single machine count min sketch for e^{-d} error.

- Proof
  - Bound probability that failures intersect with storage significantly
  - Majorize drawing without replacement by drawing with replacement

Putting it all together

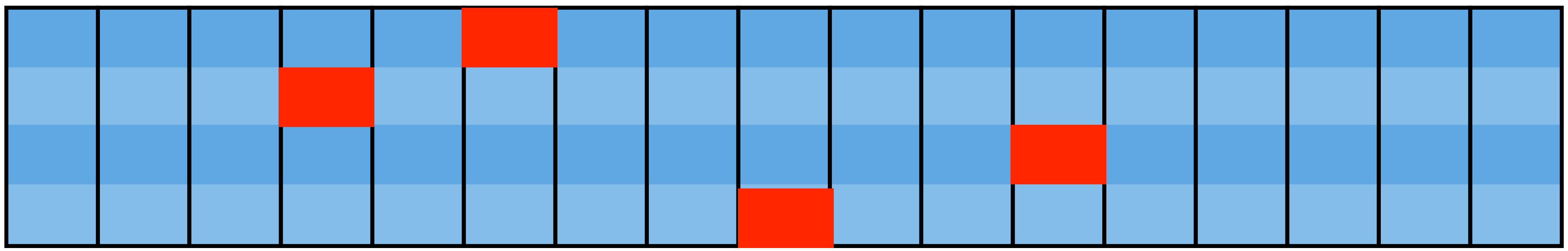


Interpolation



Properties of the count min sketch

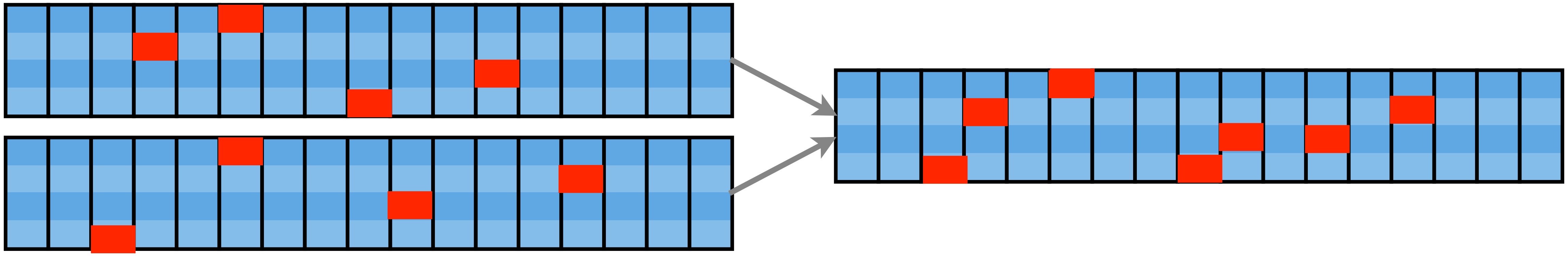
- Linear statistic



- Sketch of two sets is sum of sketches
  - We can aggregate time intervals
- Sketch of lower resolution is linear function
  - We can compress further at a later stage

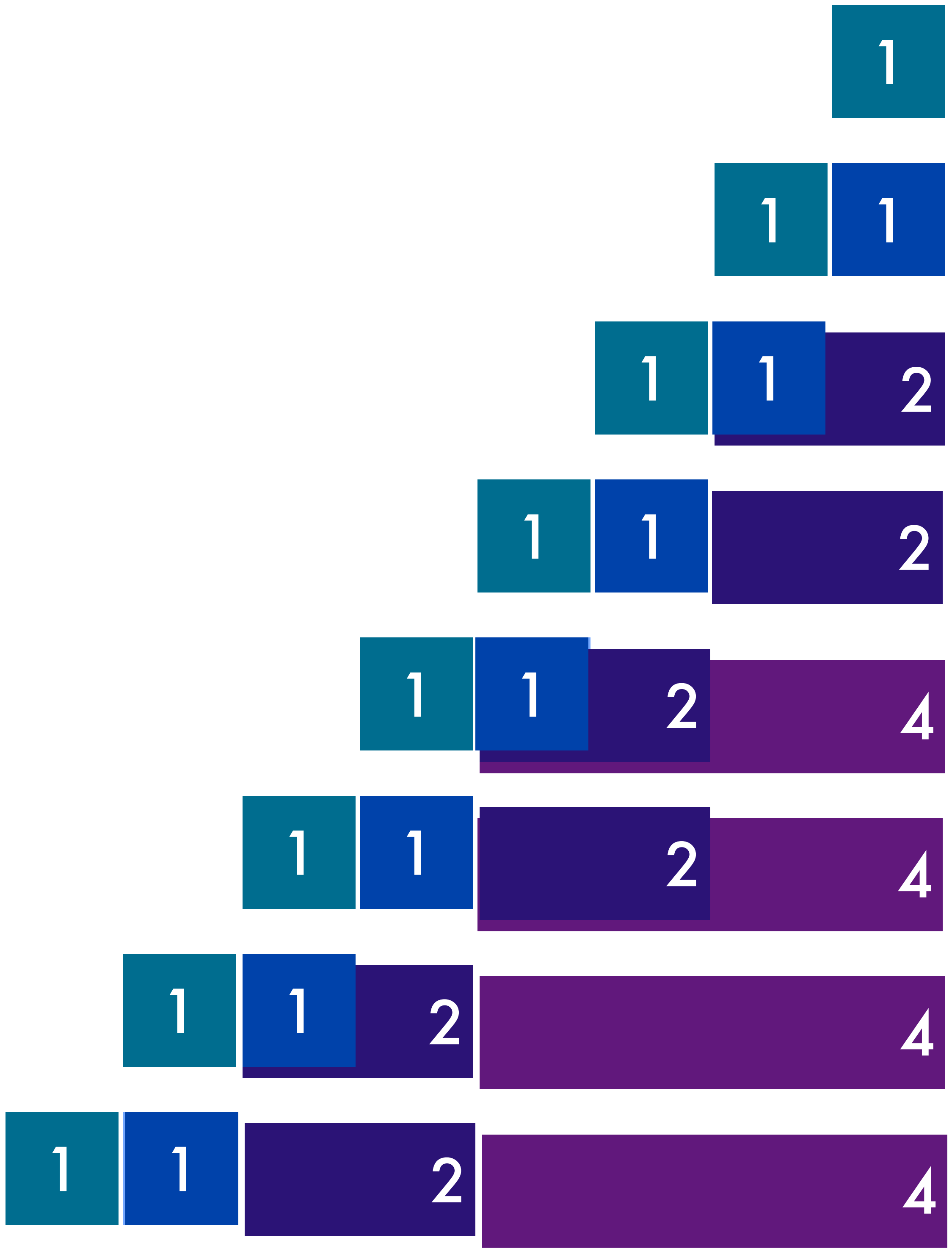
Time aggregation

- Time intervals of exponentially increasing length 1,1,2,4,8,16,32,64 ...

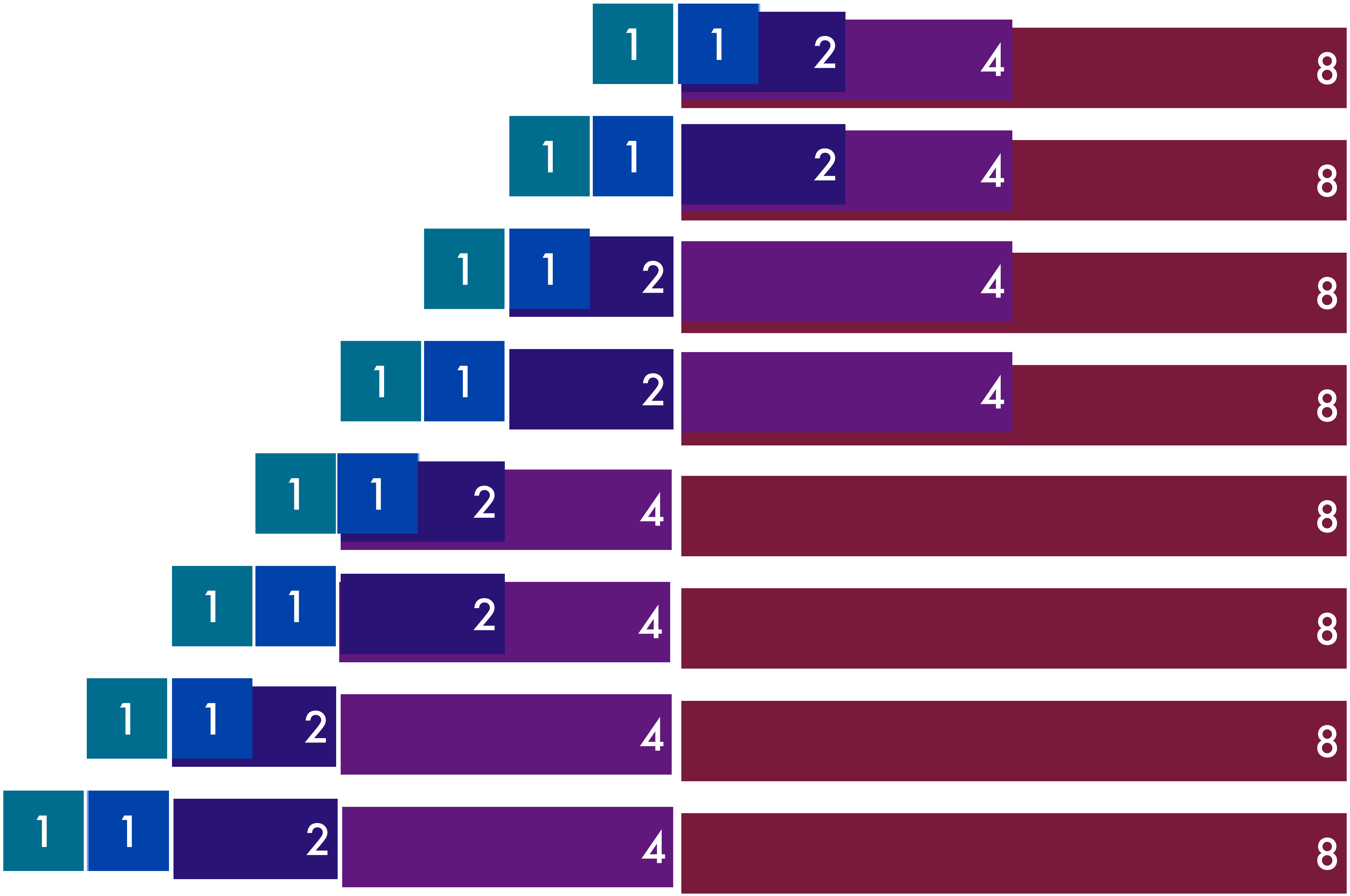


- Every 2n time steps recompute all bins up to 2n
  - 1+1=2; 1+1+2=4; 1+1+2+4=8; 1+1+2+4+8=16
  - Always fill first bin.
  - Aggregation is O(log log t) amortized.
  - Storage is O(log t)

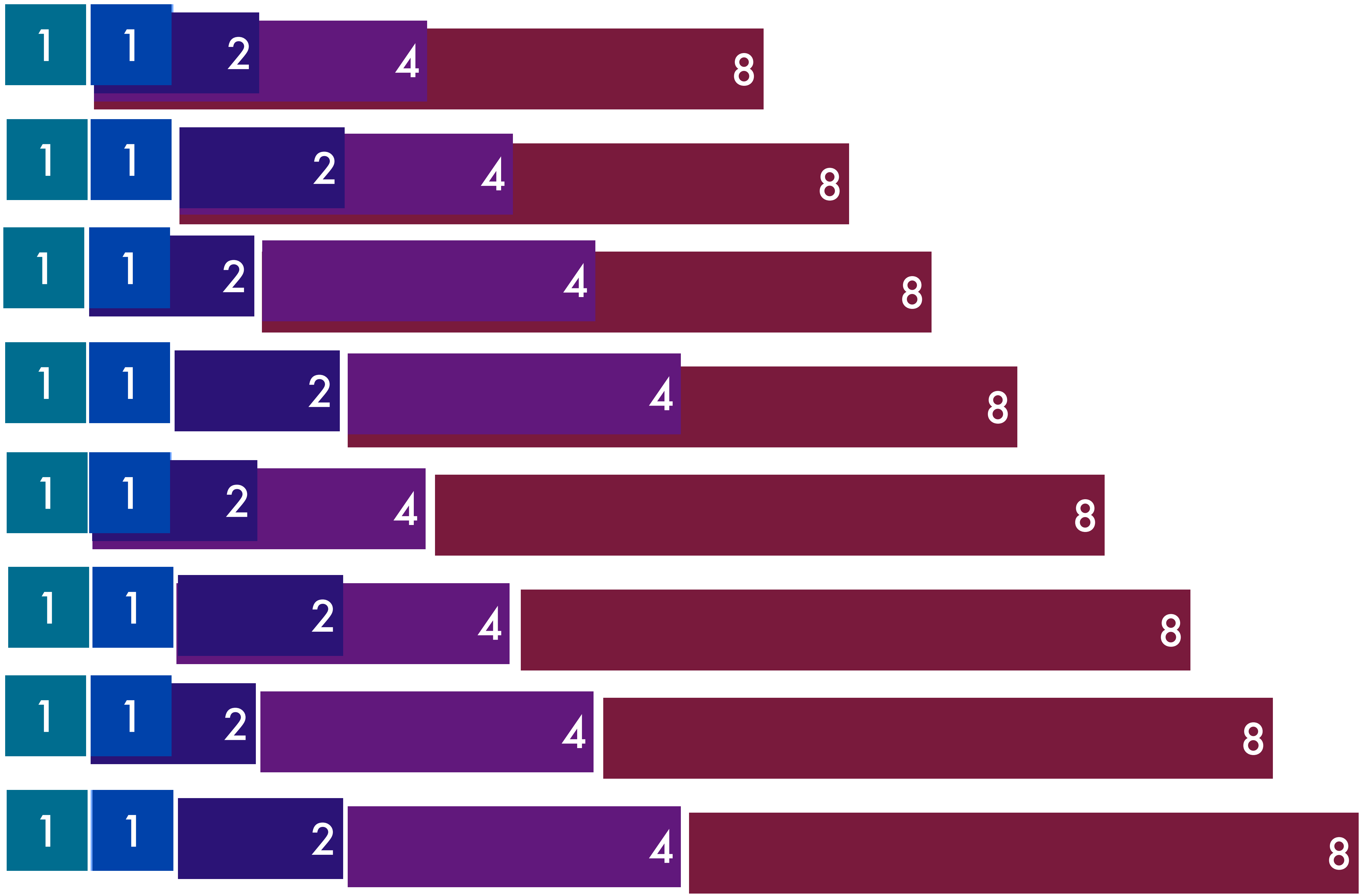
Time aggregation



Time aggregation

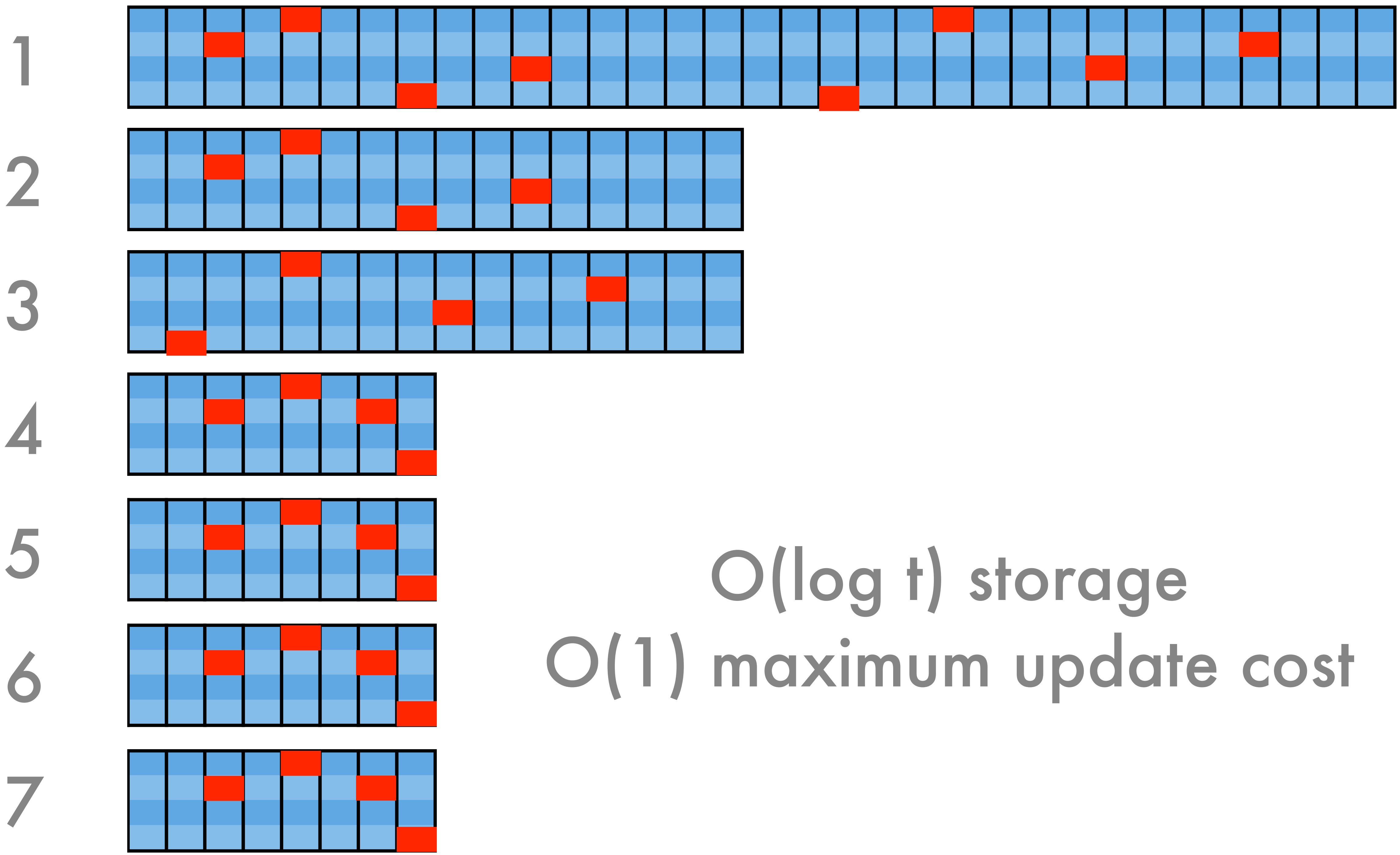


Time aggregation



Key aggregation

- Reduce bit resolution for sketch every 2^t steps

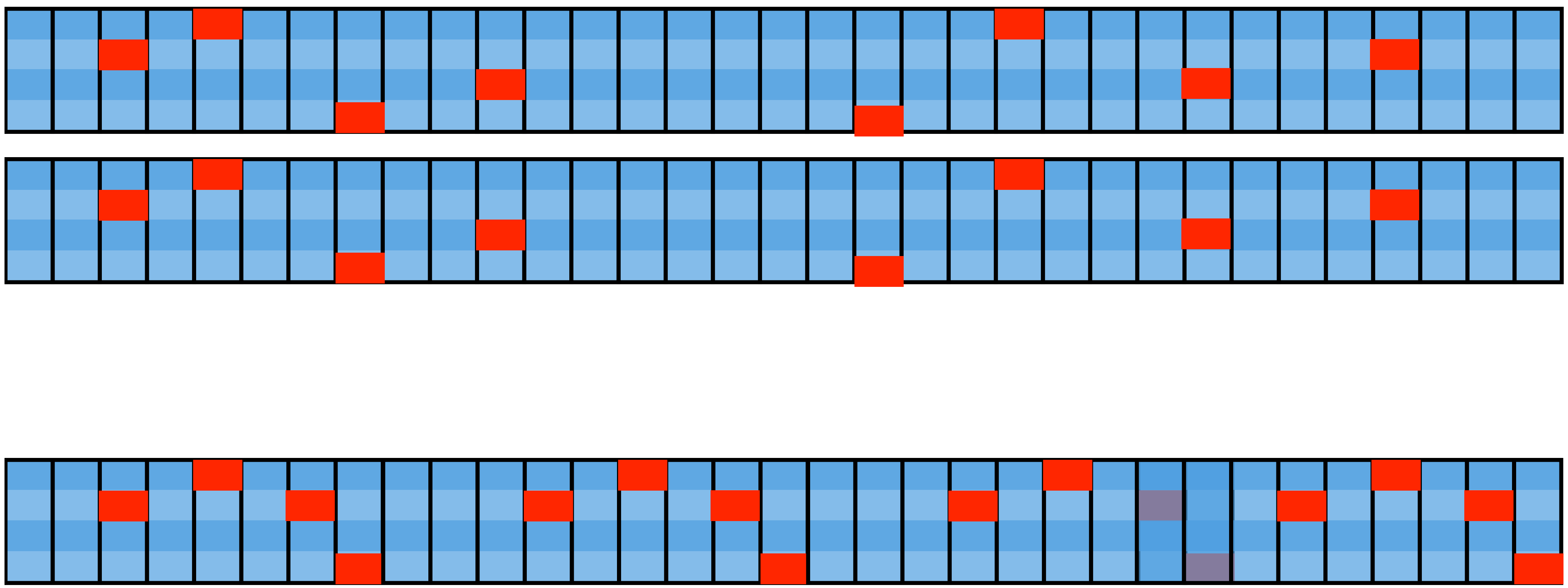


O(log t) storage

O(1) maximum update cost

Key aggregation

- Reduce bit resolution for sketch every 2^t steps

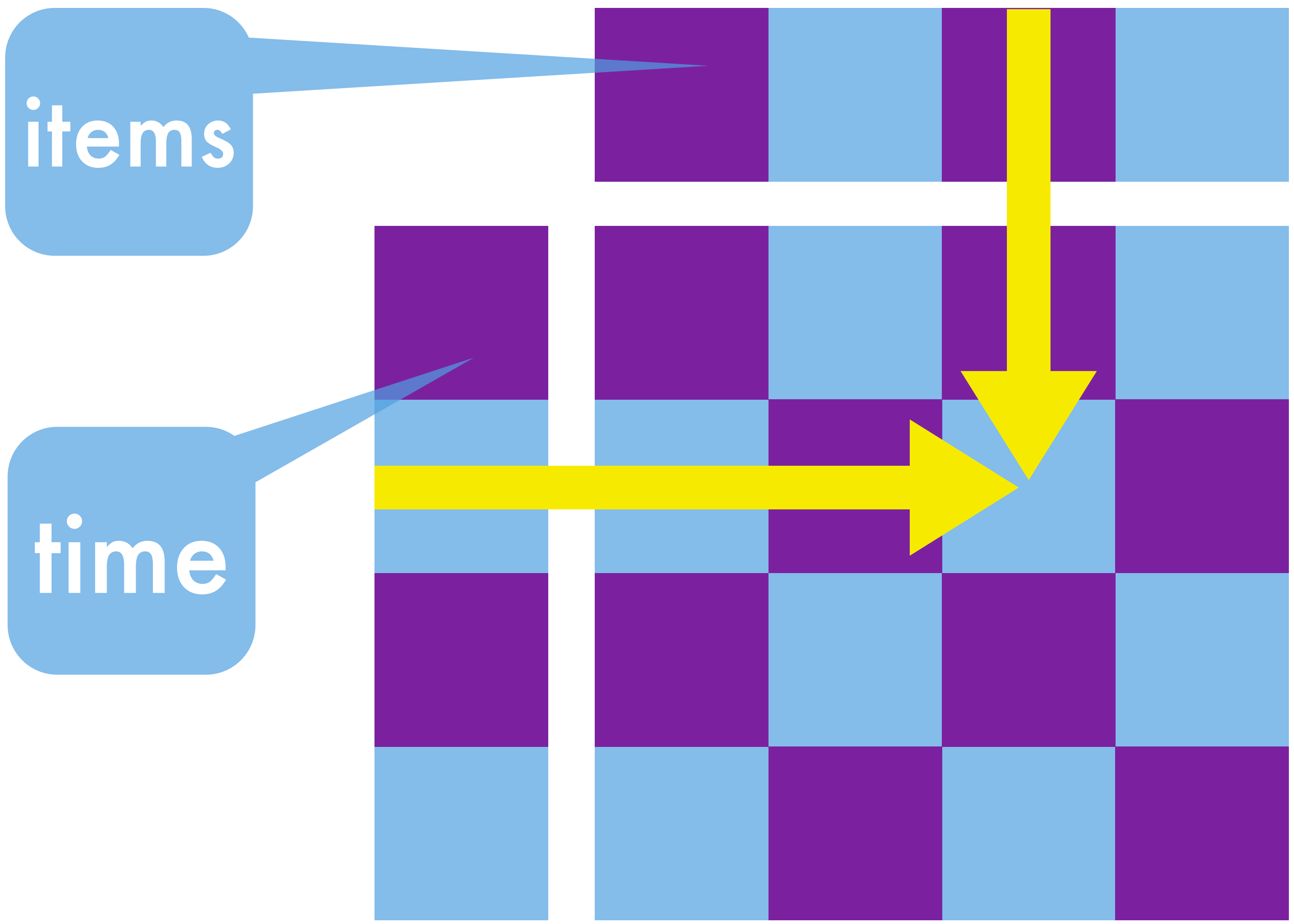


1
2
3
4
5
6
7

O(log t) storage
O(1) maximum update cost

Interpolation

- Time aggregation
  Decreasing temporal resolution - n(x,last year)
- Item aggregation
  Decreasing accuracy at fine time resolution



$$p(i,t) \approx p(i)p(t)$$

$$\Rightarrow n(i,t) \approx \frac{n(i)n(t)}{n}$$

maintain sketch aggregating both time and items

Data Streams

Data & Applications

- Moments
  - Flajolet counter
  - Alon-Matias-Szegedy sketch
- Heavy hitter detection
  - Lossy counting
  - Space saving
- Randomized statistics
  - Bloom filter
  - CountMin sketch
- Realtime analytics
  - Fault tolerance and scalability
  - Interpolating sketches

Further reading

- Muthu Muthukrishnan's tutorial
http://www.cs.rutgers.edu/~muthu/stream-1-1.ps
- Alon Matias Szegedy
http://www.sciencedirect.com/science/article/pii/S0022000097915452
- Count-Min sketch
https://sites.google.com/site/countminsketch/
- Bloom Filter survey by Broder & Mitzenmacher
http://www.eecs.harvard.edu/~michaelm/postscripts/im2005b.pdf
- Metwally, Agrawal, El Abbadi (space saving sketch)
http://www.cs.ucsb.edu/research/tech_reports/reports/2005-23.pdf
- Berinde, Indyk, Cormode, Strauss (space optimal bounds for space saving)
http://www.research.att.com/people/Cormode_Graham/library/publications/BerindeCormodeIndykStrauss10.pdf
- Graham Cormode's tutorial
http://dimacs.rutgers.edu/~graham/pubs/papers/sk.pdf
- Flajolet-Martin 1985
http://algo.inria.fr/flajolet/Publications/FlMa85.pdf