SAND2019-13637PE

Advanced Data Structures for National Cyber Security

Cynthia Phillips, Sandia National Laboratories
(and many co-authors)

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Collaborators

- Michael Bender (Stony Brook U)
- Jon Berry (Sandia)
- Martin Farach-Colton (Rutgers)
- Rob Johnson (WMWare Research)
- Tom Kroeger (Sandia)
- Samuel McCauley (Williams)
- Prashant Pandey (CMU)
- Bertrand Simon (ENS Lyon)
- Shikha Singh (Williams)
- David Zage (Intel)

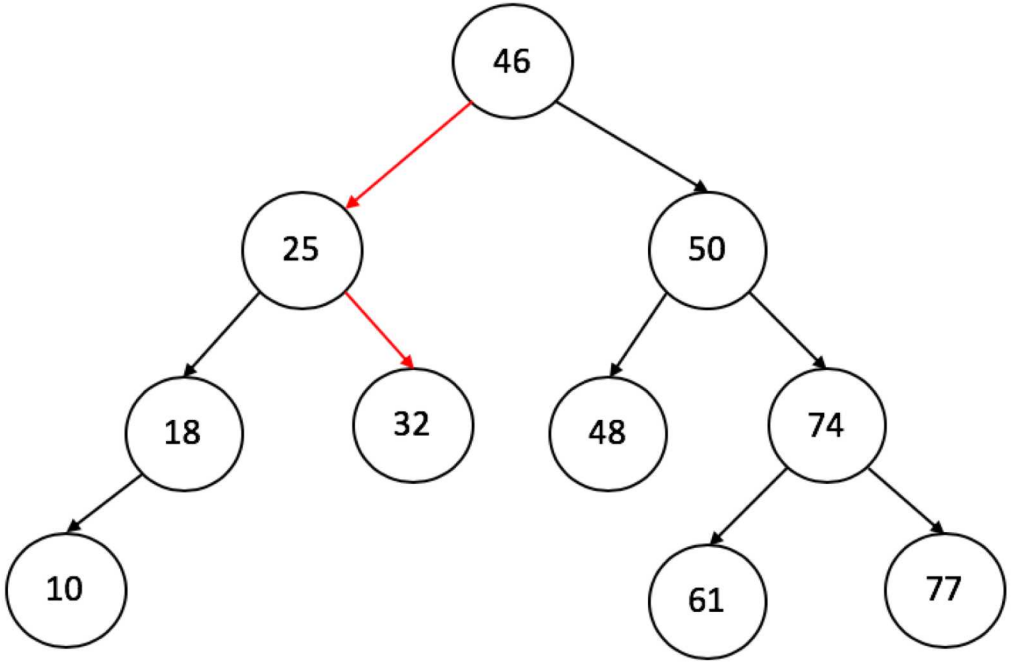
Dictionary Data Structures

Operations for a dictionary

- Insert, say (key, value)
- Delete
- Point query:
  - Is key k in the dictionary? (return pair)
- Range query:
  - Tell me all the keys in the dictionary between 100 and 1000.

In databases, these operations arrive one after the other (usually quickly) in a stream.

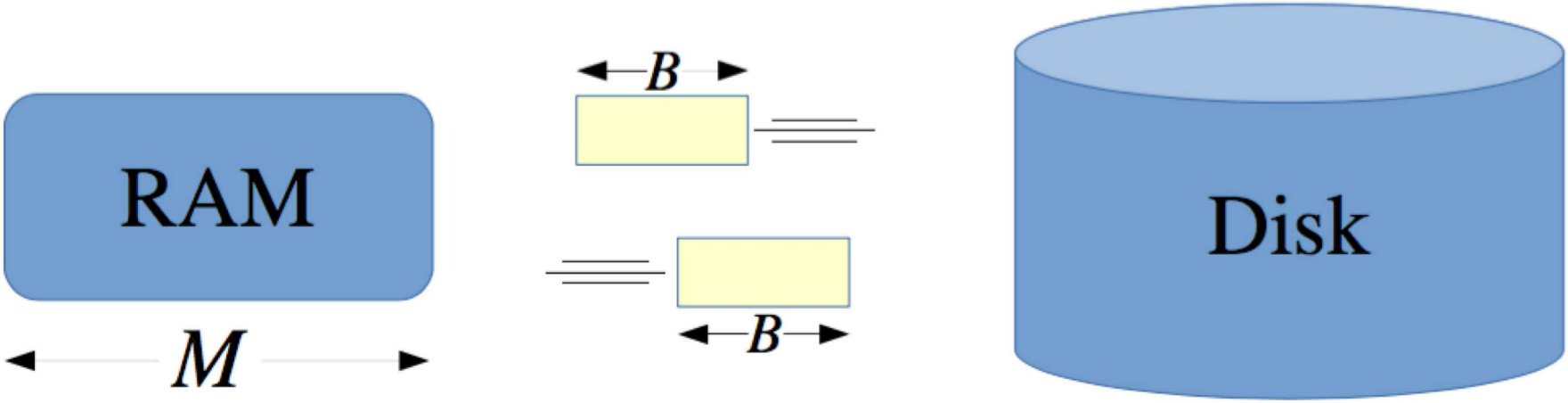
Binary Search Trees



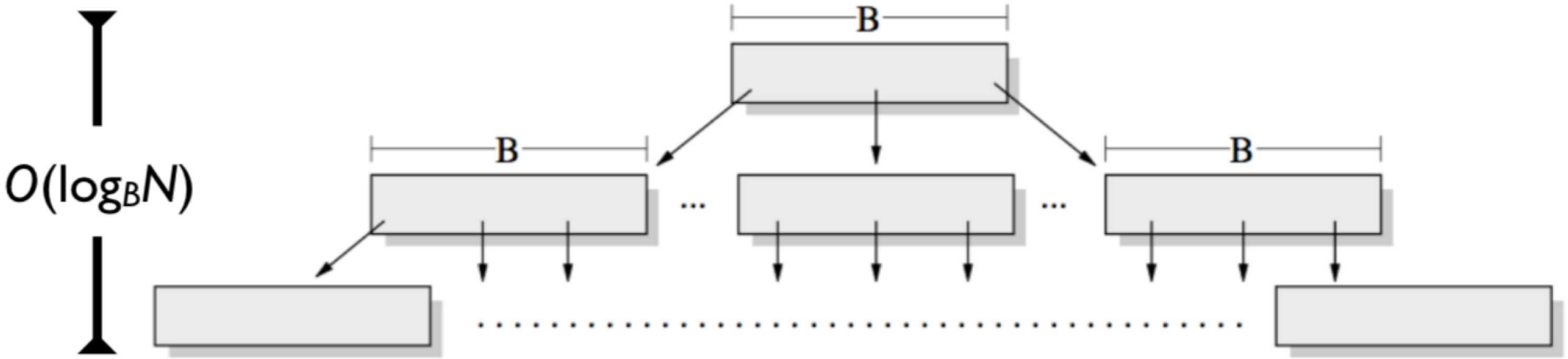
- A balanced binary tree with N elements has depth about $\log_2 N$
- Dictionary: insert, delete, search, range query

External Memory

- Disks, SSD (solid-state drives)
- Data transferred in blocks of size B
- Efficient algorithms ensure most of the block is used
- When possible, delay block transfers to fill blocks
- Theoretical analysis uses B, M, and data size N
  - Analysis counts only block transfers



B-trees

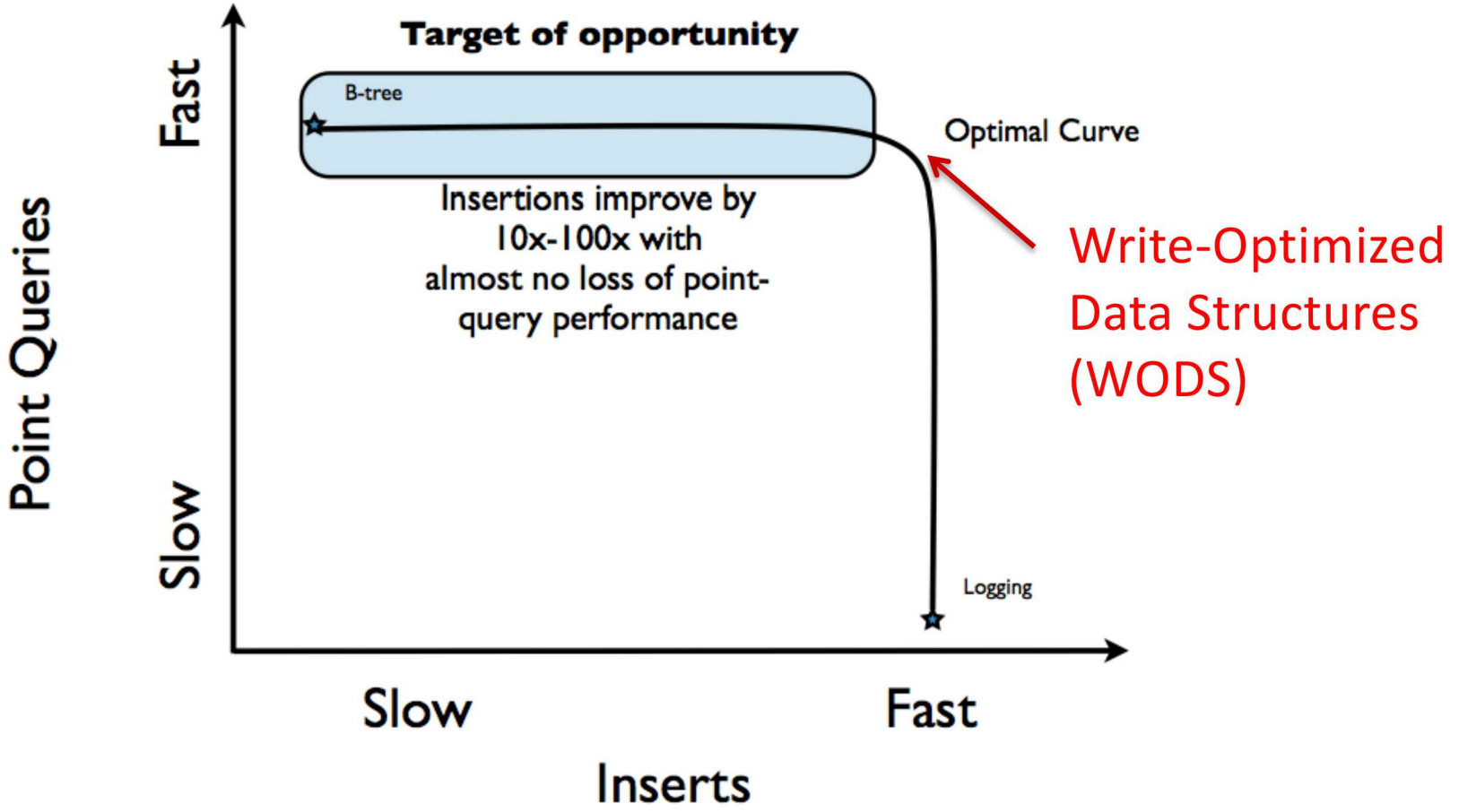


- Larger branching factor. B is block size

$$\log_B N = \frac{\log_2 N}{\log_2 B}$$

- If B is about 1024, this is $\log_2$ B = 9x fewer levels than binary trees
  - Fewer I/Os when lower levels are on disk/SSD

Write Optimization



Write-Optimized Data Structures (WODS)

- The basis for TokuDB

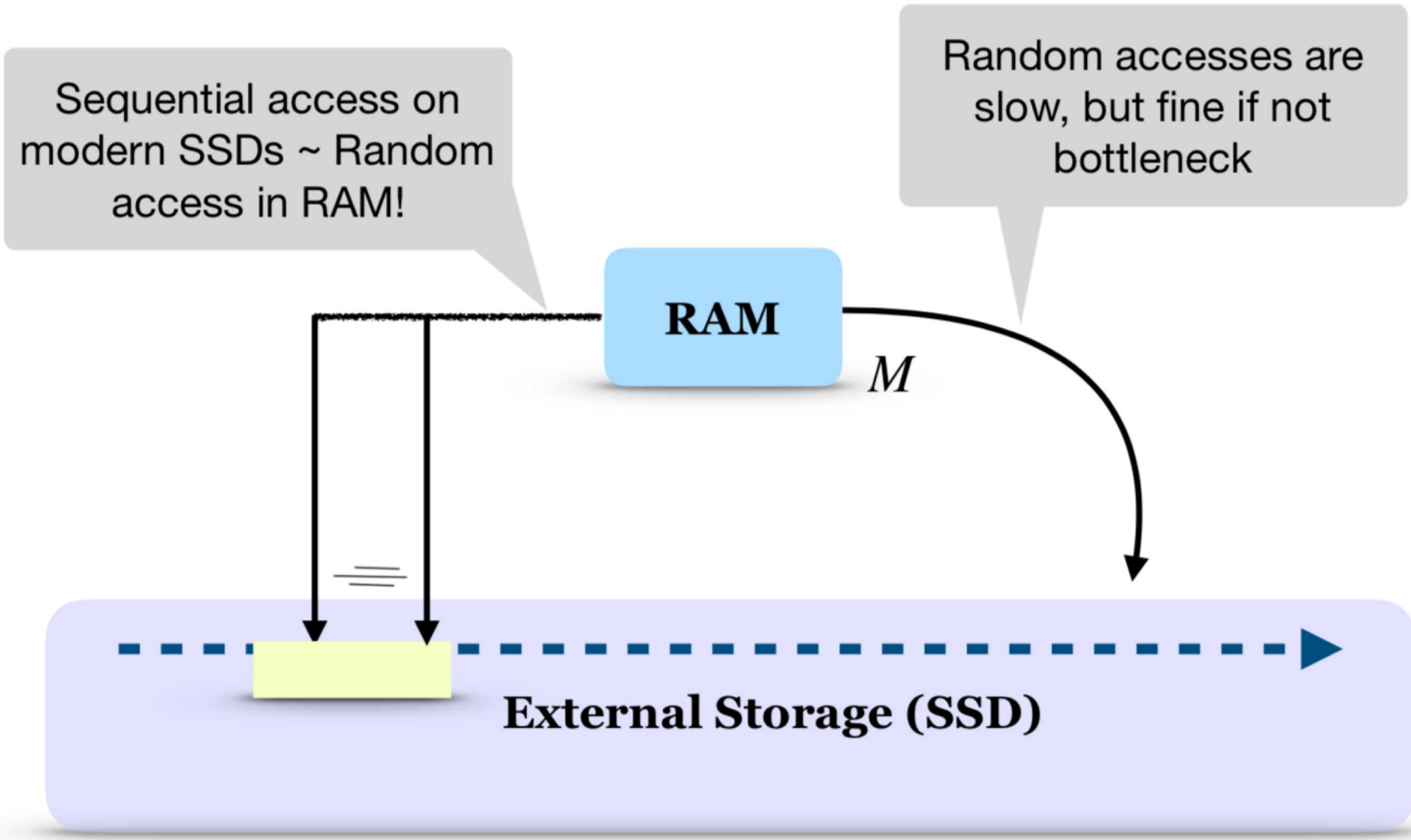
Write-Optimized Data Structures

Write optimized data structures like COLA, cascade filters, etc. (WODs) let you do fast inserts and B-tree like queries

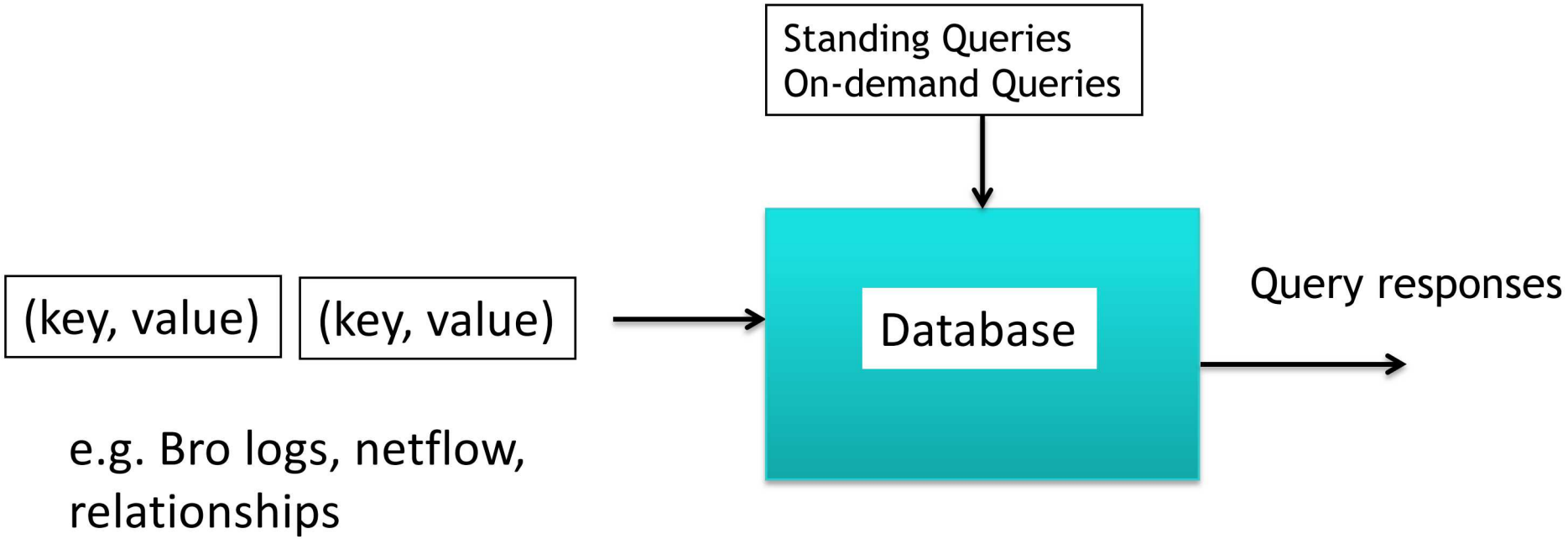
Amortized complexity: for a data structure with N elements

Optimal Insert	Optimal Query
$O\left(\frac{\log N/M}{B}\right)$	$\Omega(\log_B N)$

Modern External Memory: SSDs

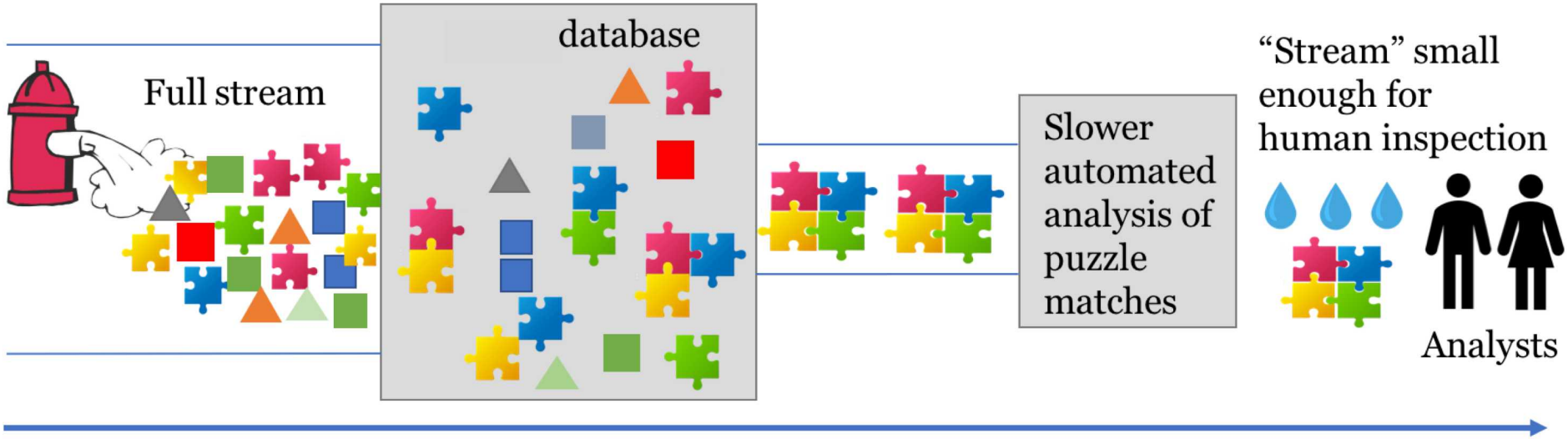


Cyber Streams and Analysis



- Stream is fast
- Interesting events can have multiple pieces that are spread in time and can hide among non-interesting pieces

Standing Queries



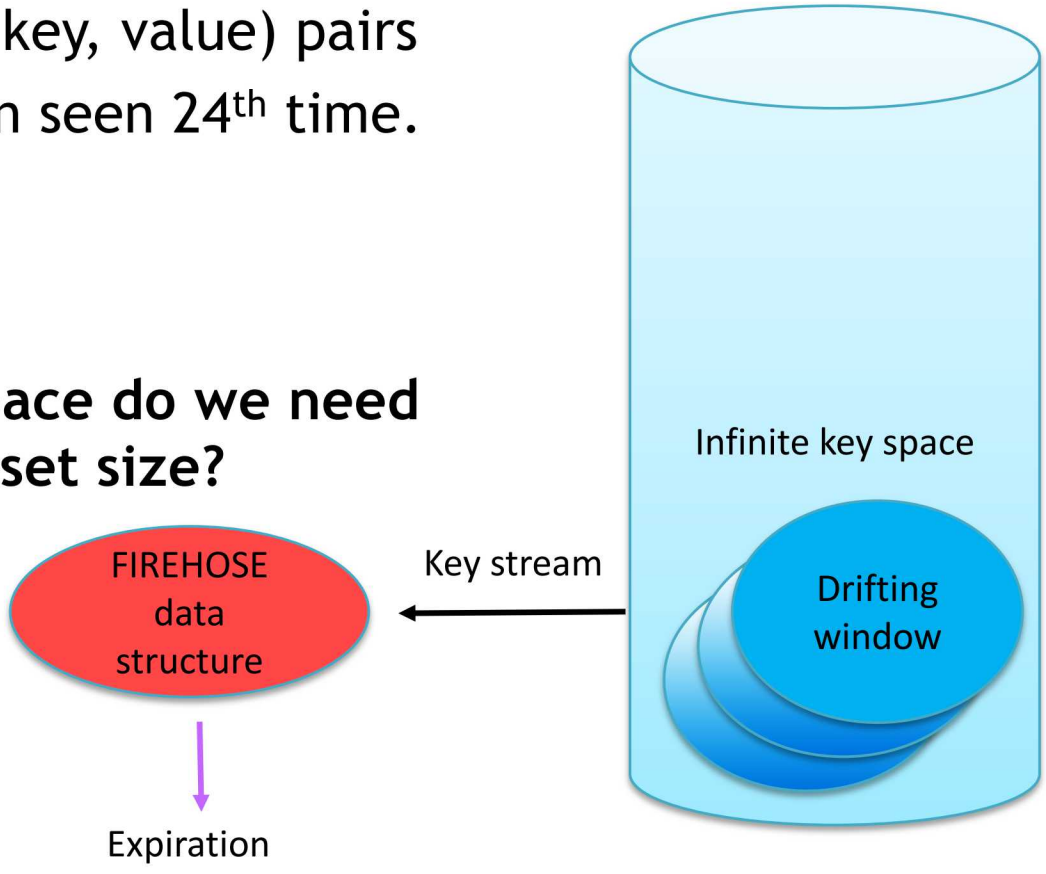
Database requirements:

- No false negatives
- Limited false positives
- Immediate response preferred
- Also relevant to other monitoring problems: power, water utilities

Firehose

- Benchmark that captures the essence of cyber standing queries
  - Sandia National Laboratories + DoD
- Input: stream of (key, value) pairs
- Report a key when seen 24th time.

How much working space do we need relative to the active set size?



http://firehose.sandia.gov/

Critical Data Structure Size

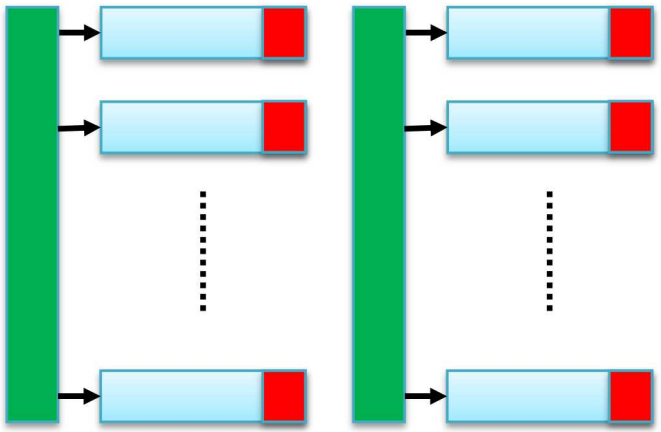
- Testing with benchmark reference implementation in Waterslide
  - 50M keys (varying counts)
  - Stable window
- Accuracy of cyber-analytics depends on keeping enough data
- Difficult to determine what to throw away
  - Most keys act the same at their start
- Keep as much data as we can!

Table Size	Generator Window Size	Reportable keys	Reported keys	**Packet drops**
2^20	2^20	94,368	62,317	0
2^20	2^21	63,673	15,168	0
2^20	2^22	17,063	9	0

https://github.com/waterslideLTS/waterslide

What is Happening?

- **Waterslide uses 'd-left hashing'**
  - Two rows of buckets
  - Constant-size
  - Fast
  - Waterslide adds LRU expiration *per bucket*



Broder, Andrei, and Michael Mitzenmacher. "Using multiple hash functions to improve IP lookups." *INFOCOM 2001*

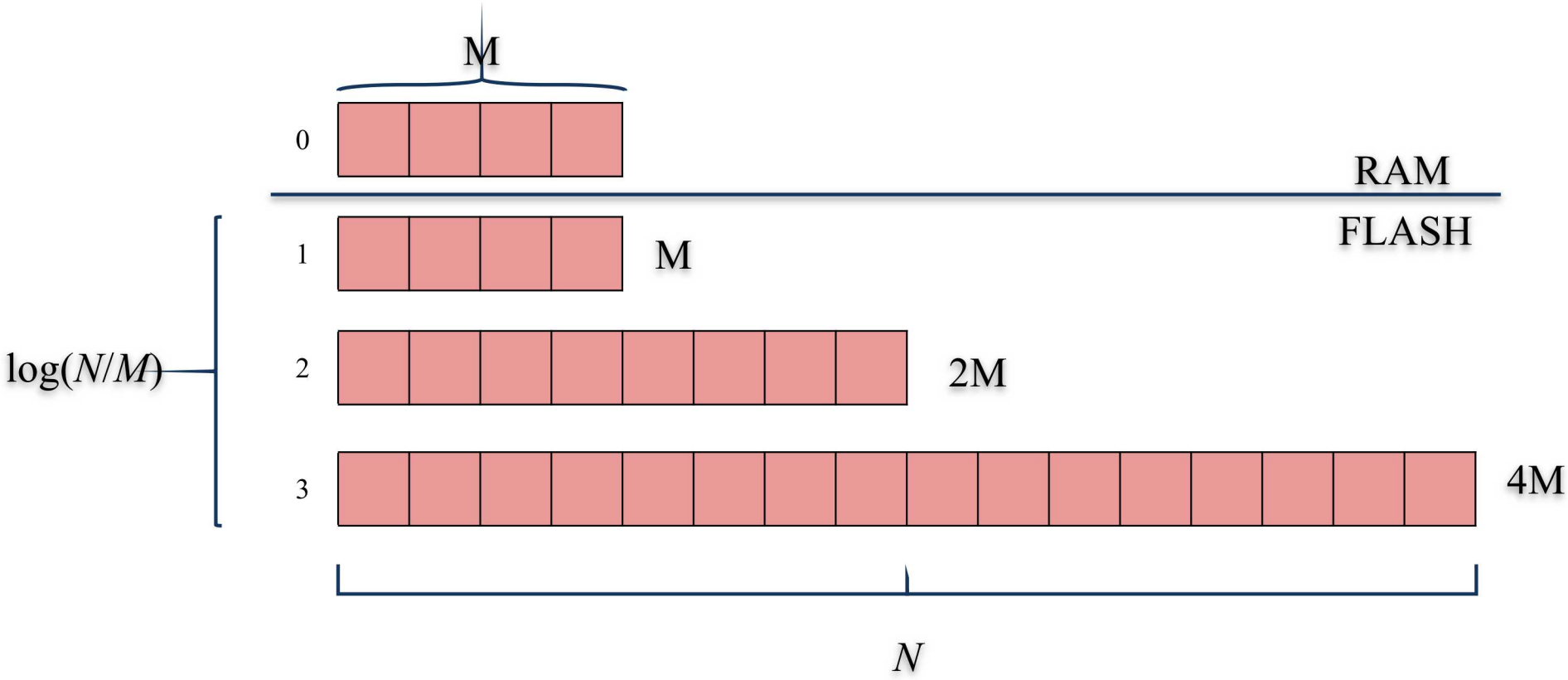
- **1/16 of all data is always subject to immediate expiration in steady state**

- **As active generator window grows, FIREHOSE accuracy quickly goes to zero**

Even when window size is only 4x data structure size, most reportable data are lost before It is reported.

Write optimization: Cascade filter

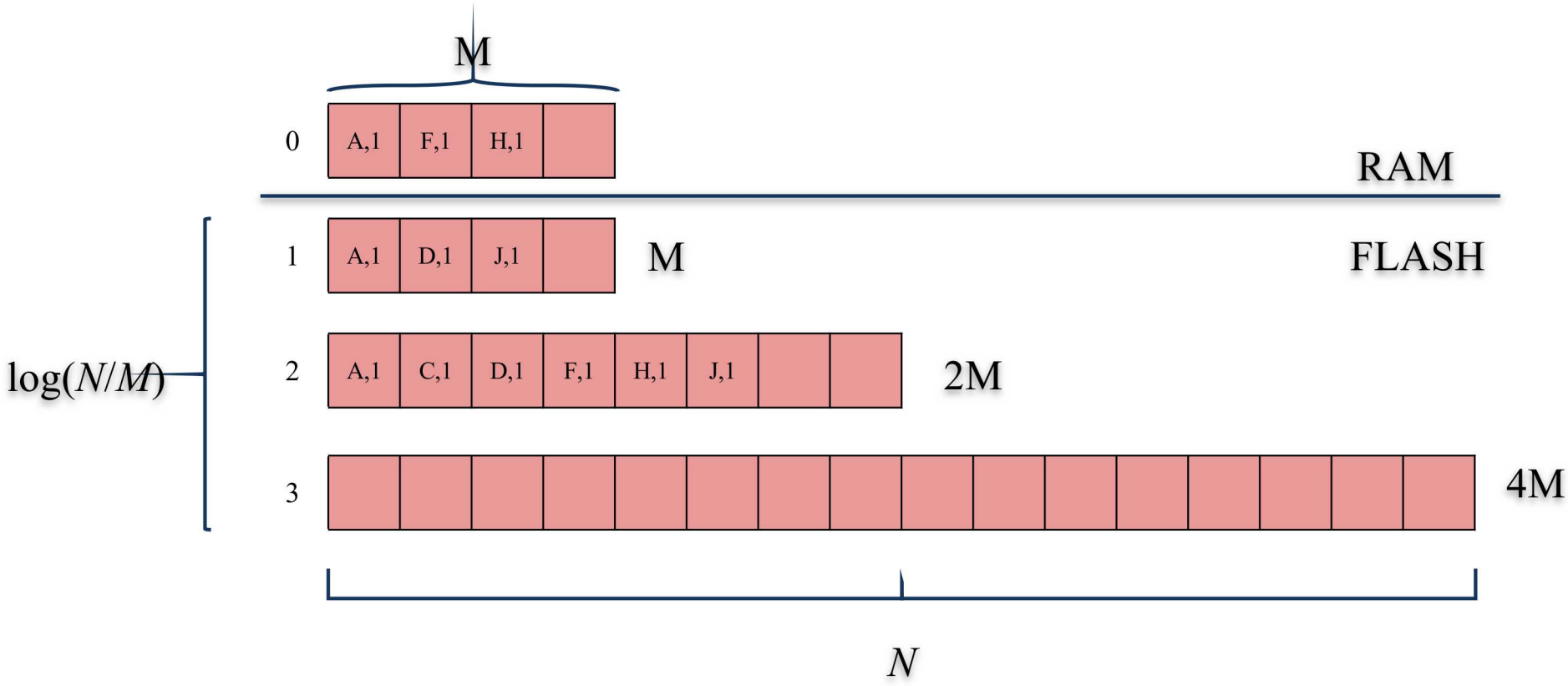
[Bender et al. 12, Pandey et al. 17]



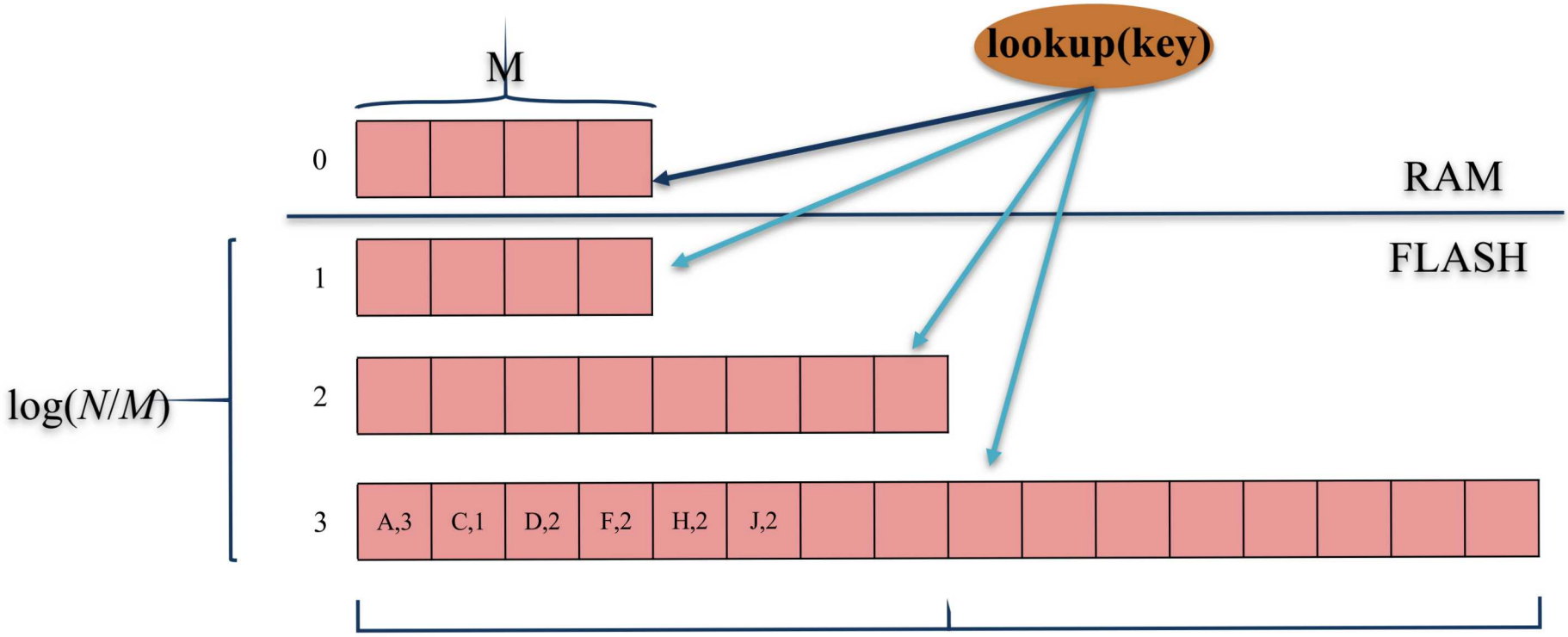
- Each level is an efficient hash table with counts
- It greatly accelerates insertions at some cost to queries.

e.g. N = 1T
M = 8B
8 levels

Ingestion “cascades”



Cascade filter Performance



Number of I/Os per item:

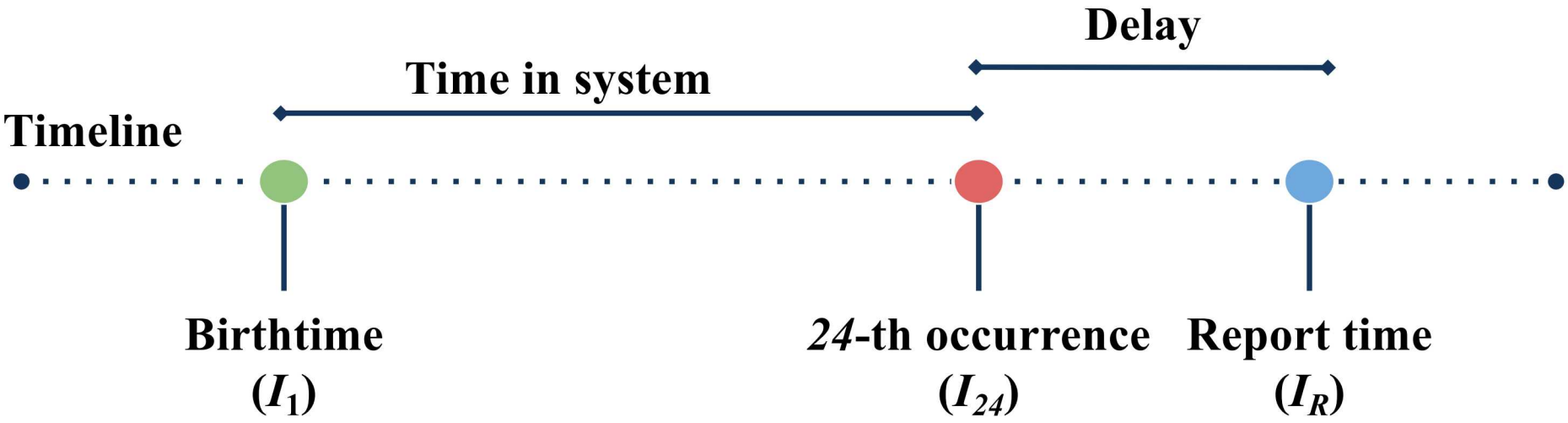
Look up: $O(\log(\frac{N}{M}))$

Insertion:

$O(\log(\frac{N}{M})/B)$

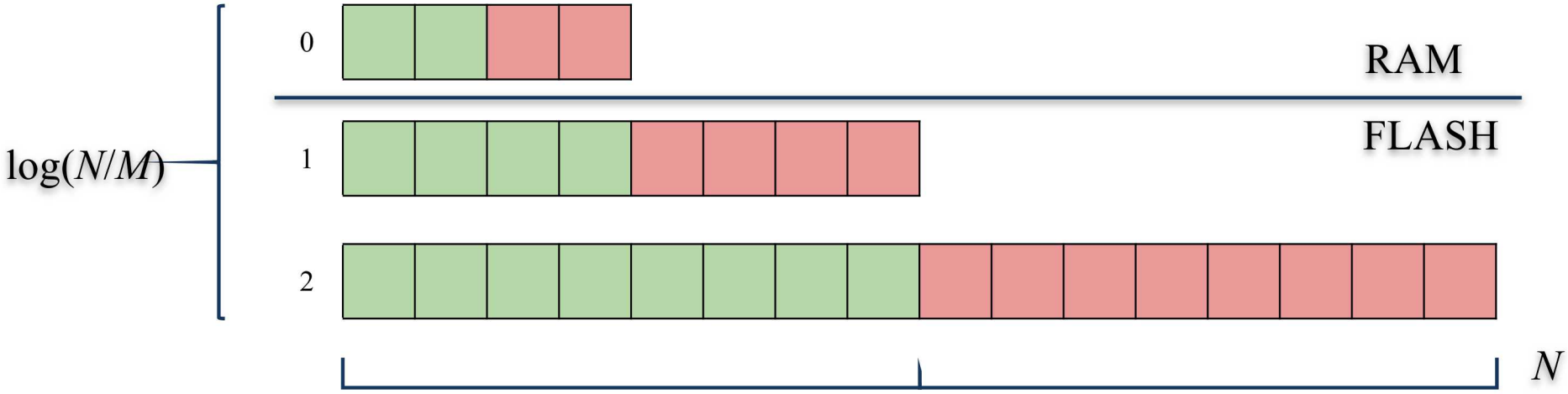
Time Stretch

- Can't afford multiple look ups per element
- Compromise: allow a little delay



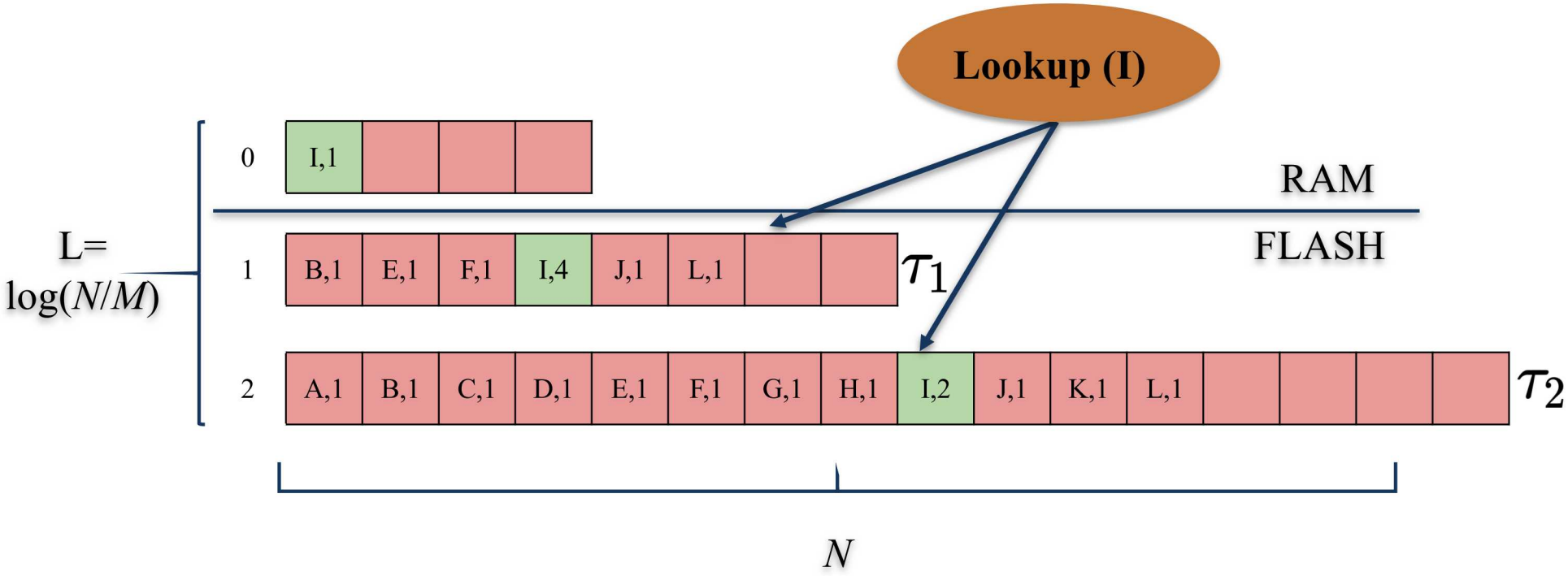
$$\text{delay} \leq \alpha * \text{time in system}$$

Time-stretch filter



- Arrays at each level split into $l = (\alpha+1)/\alpha$ equal-sized bins. Here $l = 2$ and $\alpha = 1$.
- Flushes at bin granularity on fixed round-robin schedule.
- Will always see the oldest element in time to report
- Bounded delay time, factor $(\alpha+1)/\alpha$ slower ingestion
- This example: 1 hour for 24 instances to arrive ➡ report up to 24 hours late and system runs 2x slower than when we gave no promises on delay

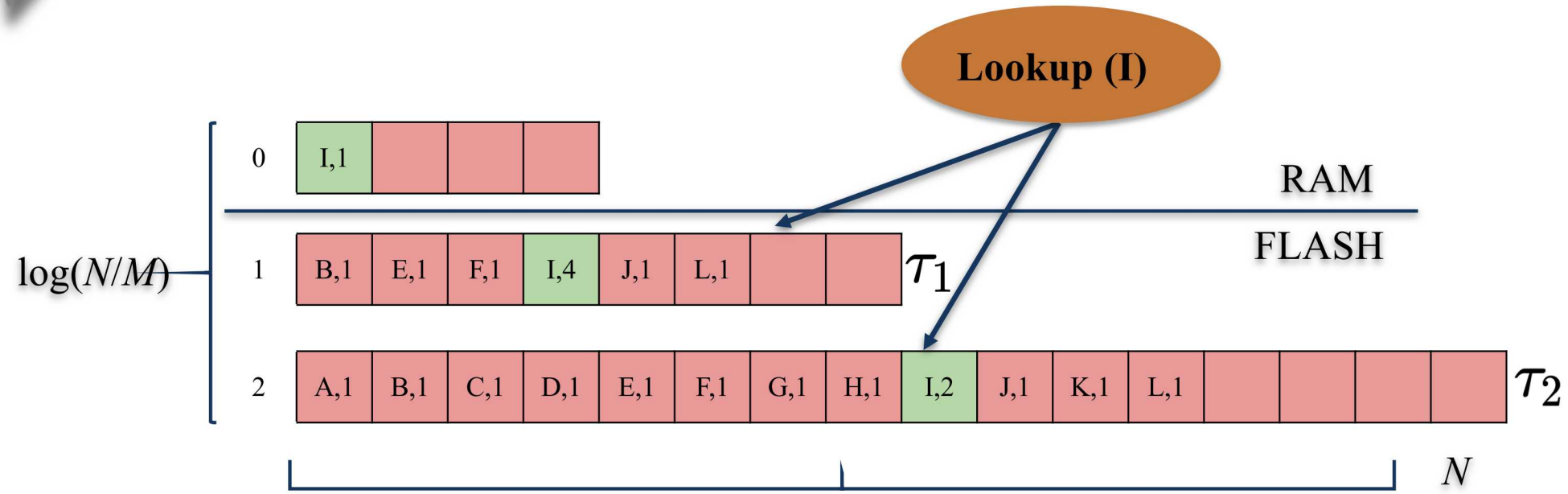
Popcorn filter: immediate reporting



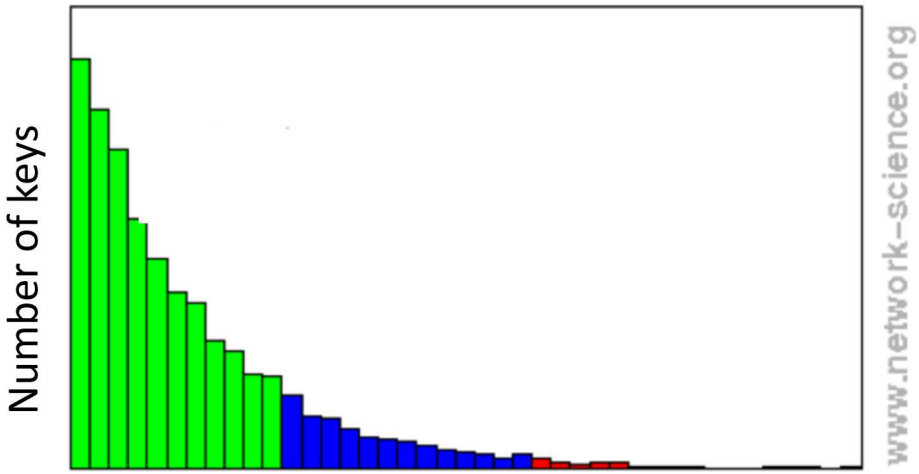
- Avoid unnecessary I/Os if we can **upper bound the total instances on disk**

$$\text{Lookup if } \text{RamCount} = 24 - \sum_{i=1}^{L} \tau_i$$

Popcorn filter



- Immediate reporting works if keys have power-law distribution
- Delay gives a count stretch: bounded extra counts



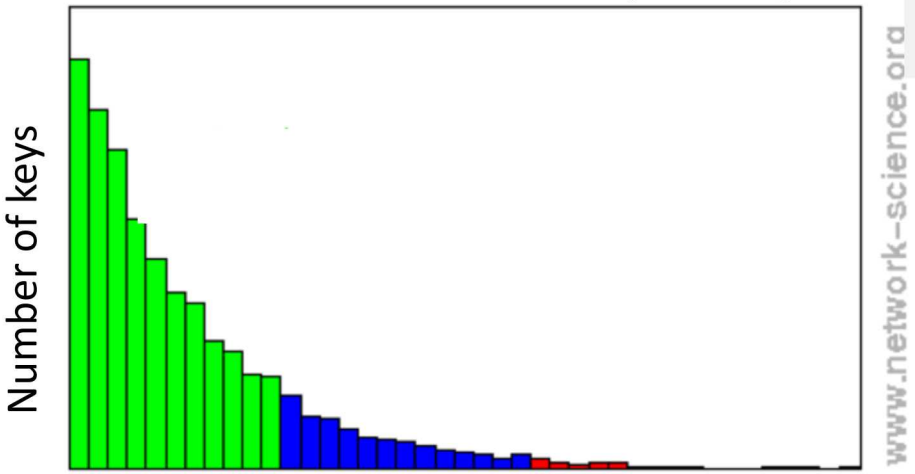
Sandia National Laboratories

A little math

- Power-law distribution: Counts of keys follow a power-law distribution with exponent θ if the probability an item has count c is proportional to $c^{-(\theta-1)}$
- Theorem: If a stream has N elements with counts following a power-law distribution with $2 < \theta < 2.96$, T is the reporting threshold, and

$$T\omega > 2.5\left(\frac{N}{M}\right)^{\frac{1}{1-\theta}}$$

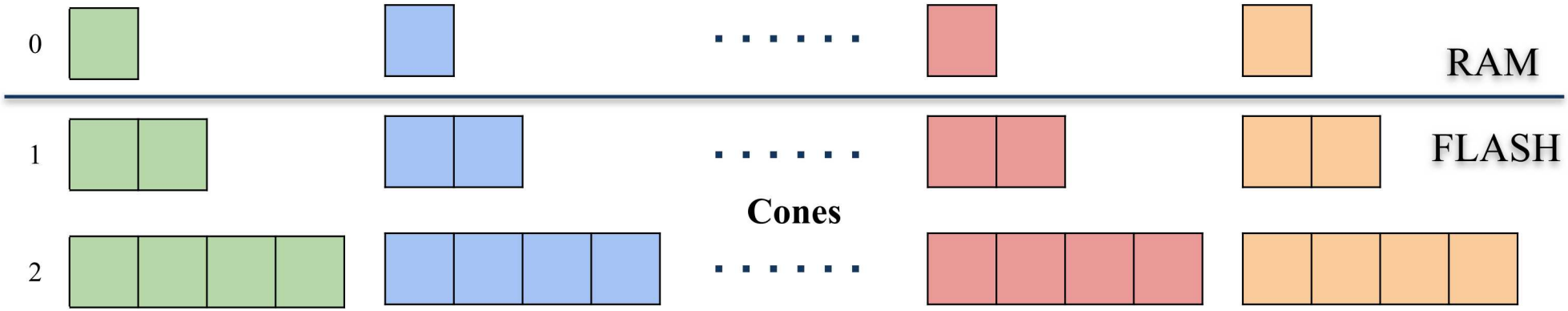
Then our algorithm gives a count stretch of $(1 + \omega)$ with amortized I/O complexity of $O\left(\frac{1}{B}\log\frac{N}{M}\right)$ per item w.h.p



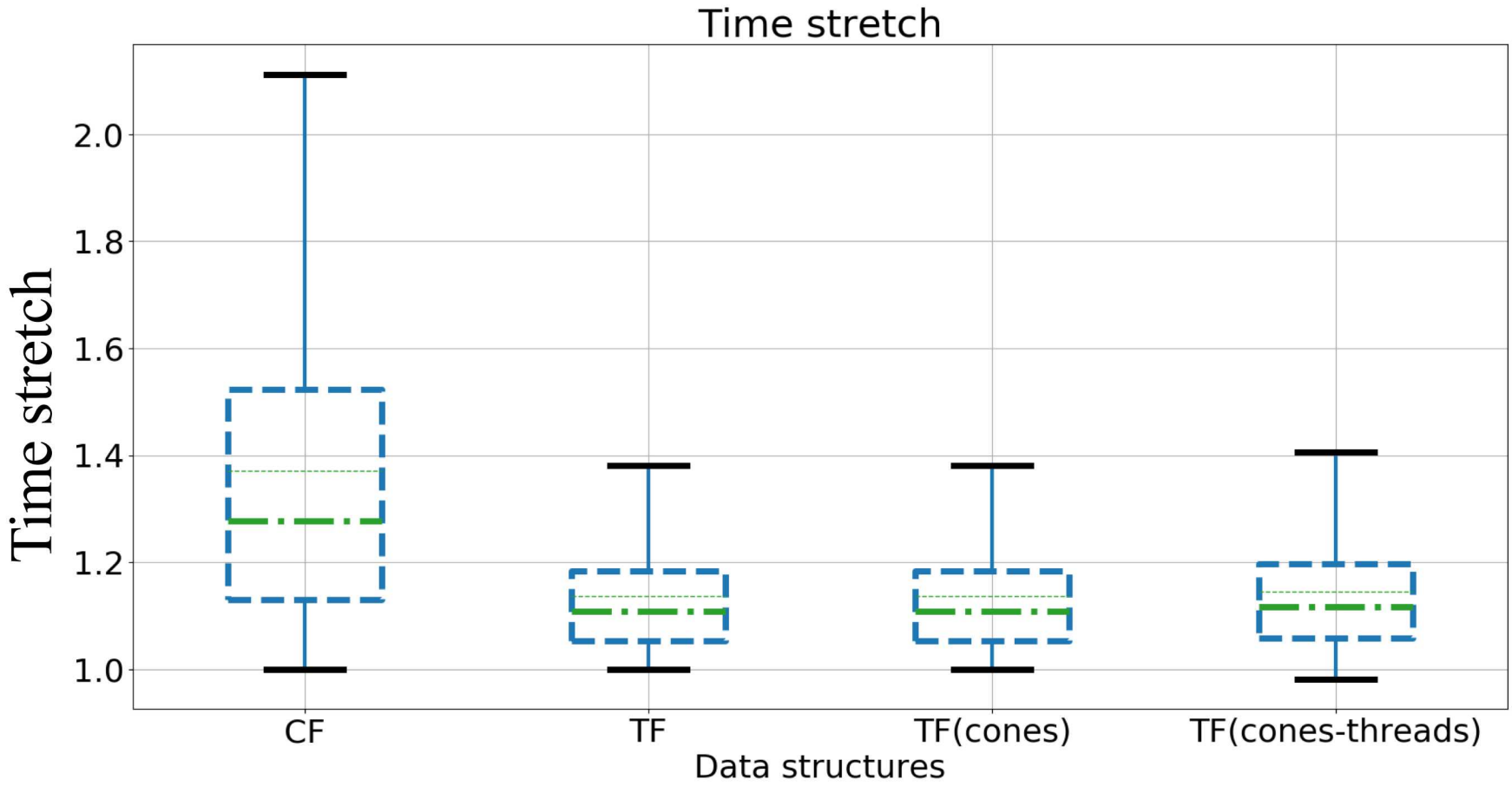
Key frequency

Multithreading and Deamortization

- Data structures run well on average, but some operations take a long time
- Do a little work for each arriving element

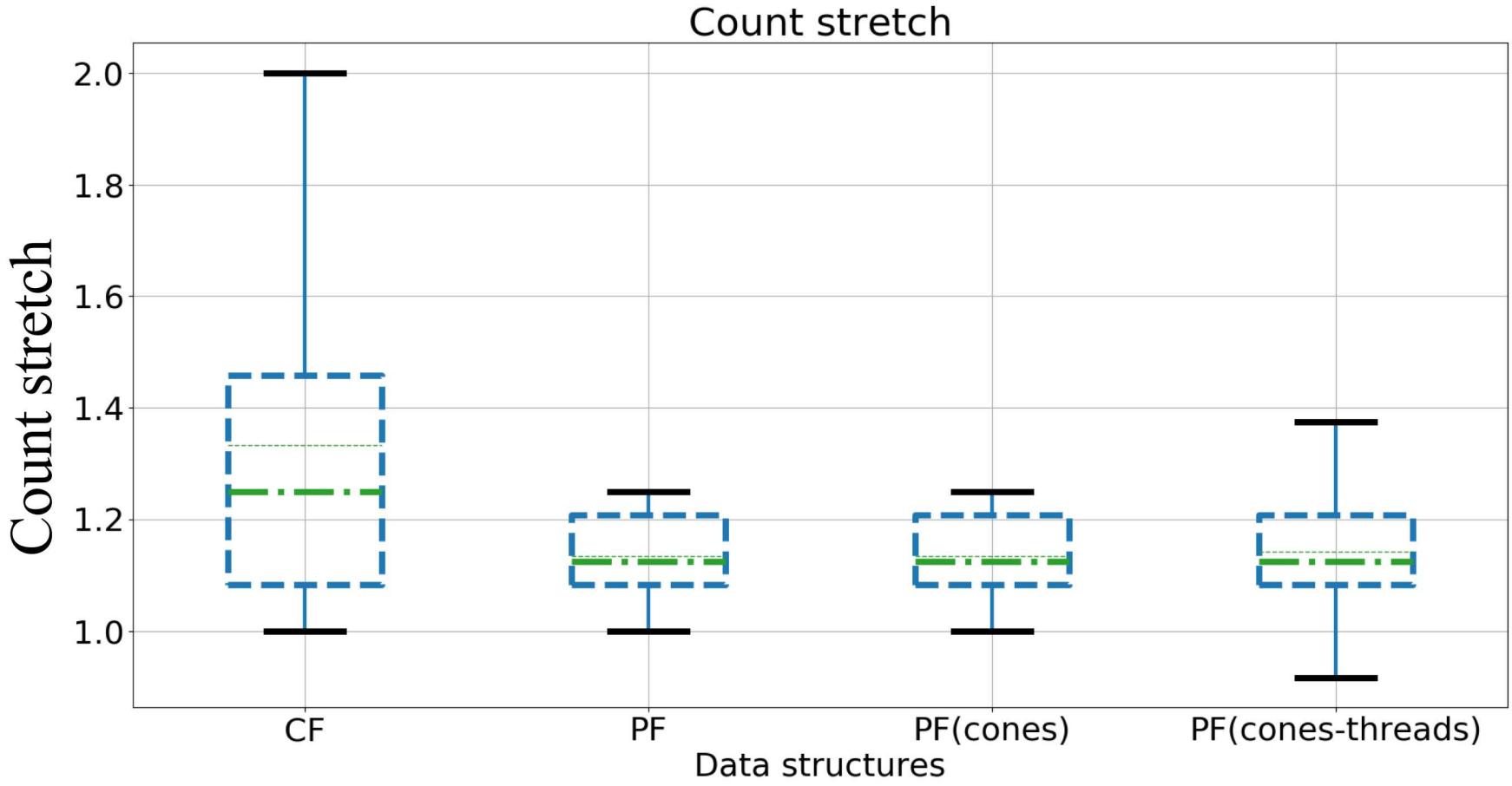


Time stretch



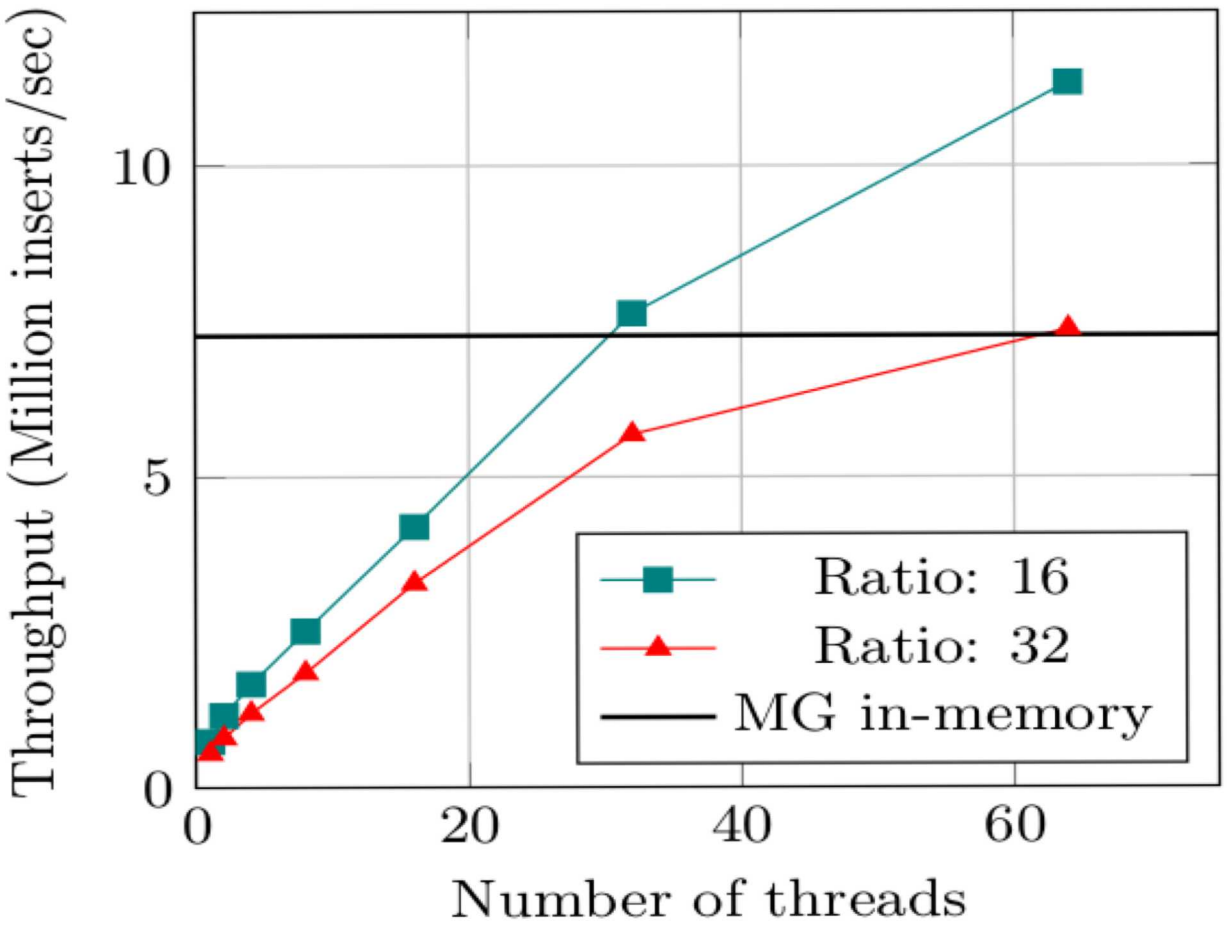
- Time-stretch filter has the smallest empirical time stretch.
- Deamortization and multithreading had negligible effect on empirical time stretch.

Count stretch



- Count-stretch filter has the smallest empirical count stretch.
- Deamortization and multithreading had negligible effect on empirical count stretch.

Scalability – count stretch

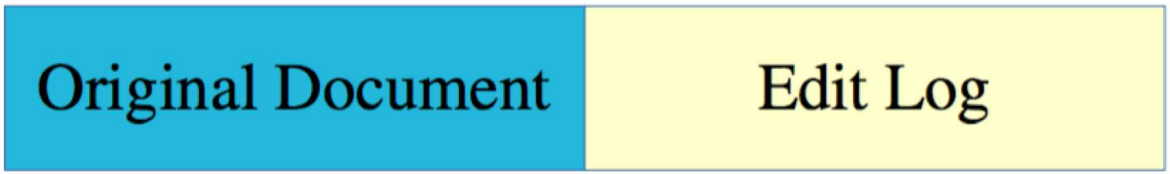


16 threads

- This runs faster than Waterslide (10M insertions/sec) and reports all reportable keys

Additional Challenge

- Systems sacrifice security for I/O efficiency
  - Example: Microsoft Word "fast save" appends edit log
  - Adversaries can recover old versions of documents



- Hide the history of a data structure on disk
  - Order of arrival
  - No idea if there has ever been a deletion

History-Independent Data Structures

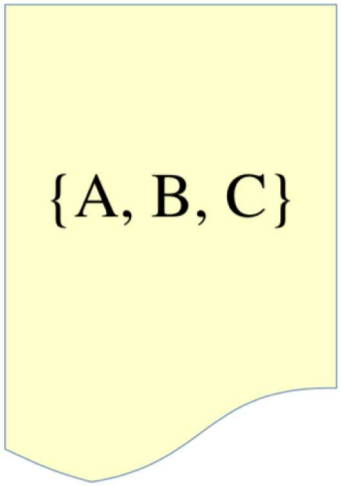
- An added level of protection for data on disk
- An adversary who acquires the disk and examines memory cannot determine anything more than API would give
- If the adversary can examine the disk cannot determine:
  - Order elements arrived
  - If any data has been deleted
- Order information can reveal sources, policy, etc.
- One potential motivation: drones

From: Wikipedia

History-Independent Data Structures [Naor & Teague '01]

[Blelloch & Golovin '07] [Buchbinder & Petrank '03] [Bajaj, Chakrabati, Sion '15] [Bajaj & Sion '13] [Molnar, Kohno, Sastry, Wagner '06] [Moran, Naor, Segev '07] [Naor, Segev, Wieder '08] [Roche, Aviv, Choi '15] [Tzouramanis '12] [Golovin '08, '09, '10]

- Bit representation reveals no additional info about past states of the data structure
- Example:



Observer cannot infer sequence of operations leading to current state

1.Insert A	1.Insert C
2.Insert B	2.Insert B
3.Insert C	3.Insert A
4.Insert D	
5.Delete D	

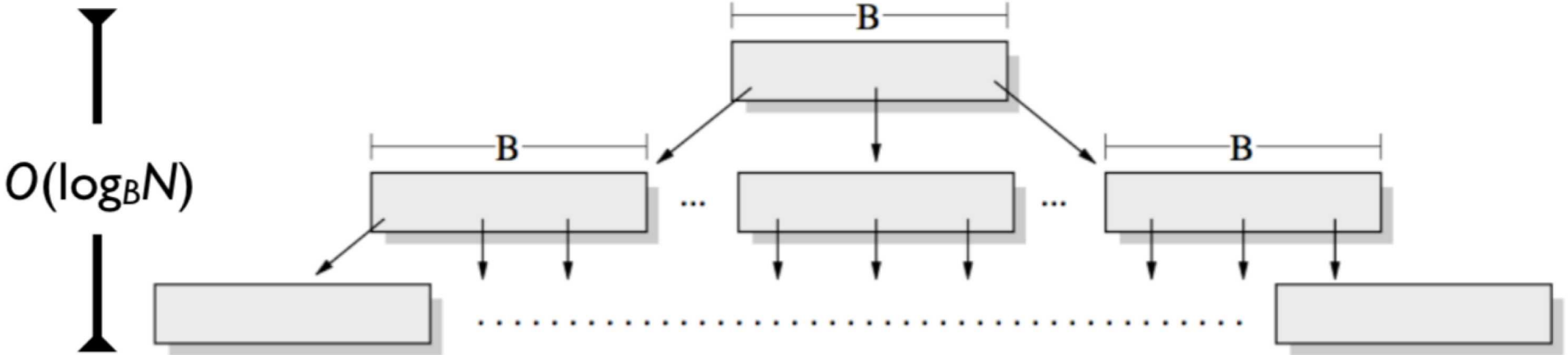
History Independence (HI)

- Strong history independence gives guarantees if the adversary sees the data representation multiple times
  - Requires a canonical representation
  - Expensive
    - Provably cannot achieve amortized o(N) operation cost whp
- Weak history independence protects against a one-time theft
  - Representation is drawn uniformly at random from a given large structured set
  - Can be much more efficient
  - The right model if a disk can only be stolen once

History-Independent Dictionaries

- Skip lists. External memory block size B, n items
  - Insert, delete, search: $O(\log_B n)$
  - Range search with k items in range: $O(\log_B n + k/B)$ block
  - Amortized, with high probability: $1 - O\left(\frac{1}{n^c}\right)$
  - Optimal
- Previous work for HI skip lists: insert $\Theta(\log n)$
- Cache-oblivious B-trees
  - Same bounds except inserts are (optimal) $O(\frac{log^2 n}{B} + \log_B n)$
  - O(n) space
  - Experiments show small slowdown
- Oblivious adversary for analysis: sets order of operations, but does not know the random tosses of the data structure

B-trees



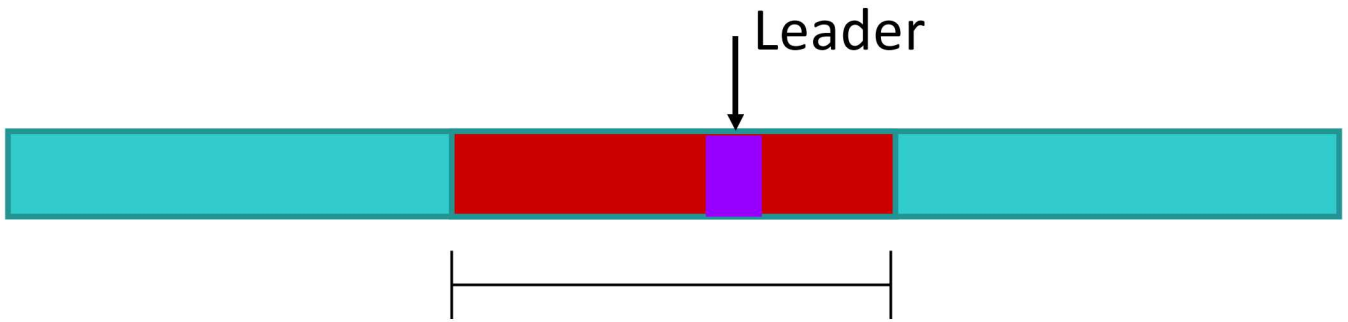
- All the elements are in the leaves (on disk)
- Randomization involves how many elements in each leaf

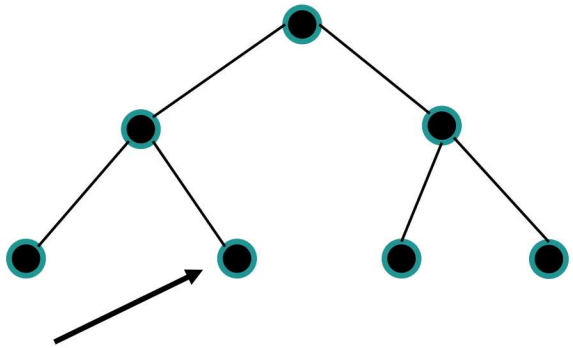
To start, though size/storage allocation

- For N elements, allocate array size |A| from N to 2N-1 uniformly
- For any insert/delete reallocate with probability $\Theta\left(\frac{1}{|A|}\right)$

Key ideas

Recursive stick breaking





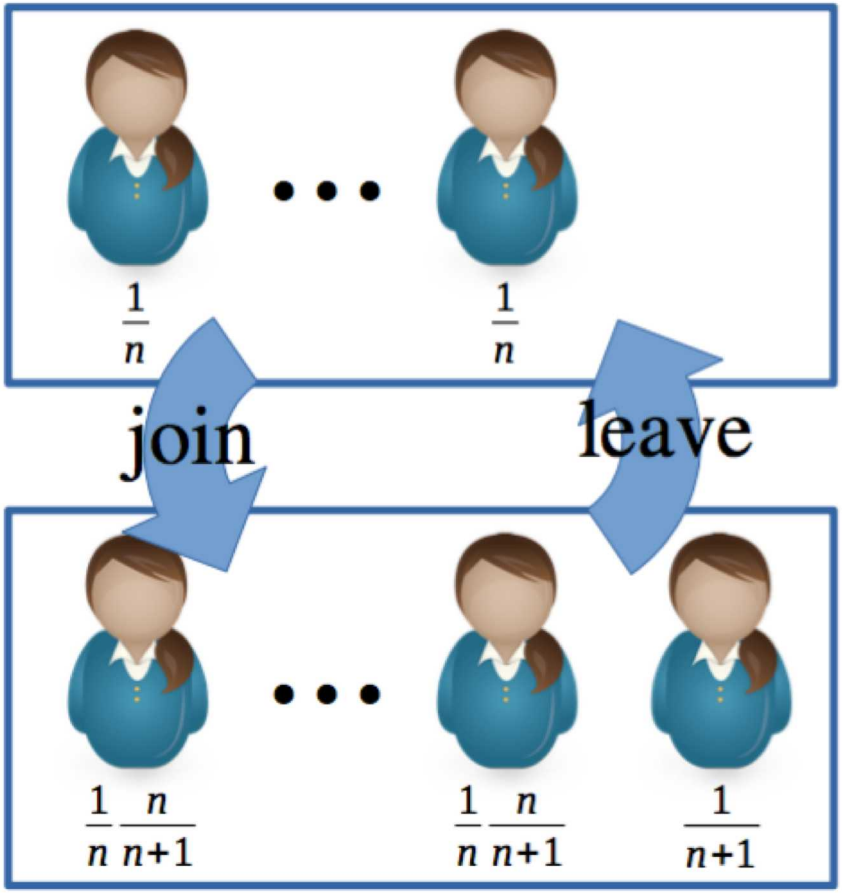
Leaves have log N elements
Always packed left

Reservoir Sampling with Joins and Leaves [Vitter '85]

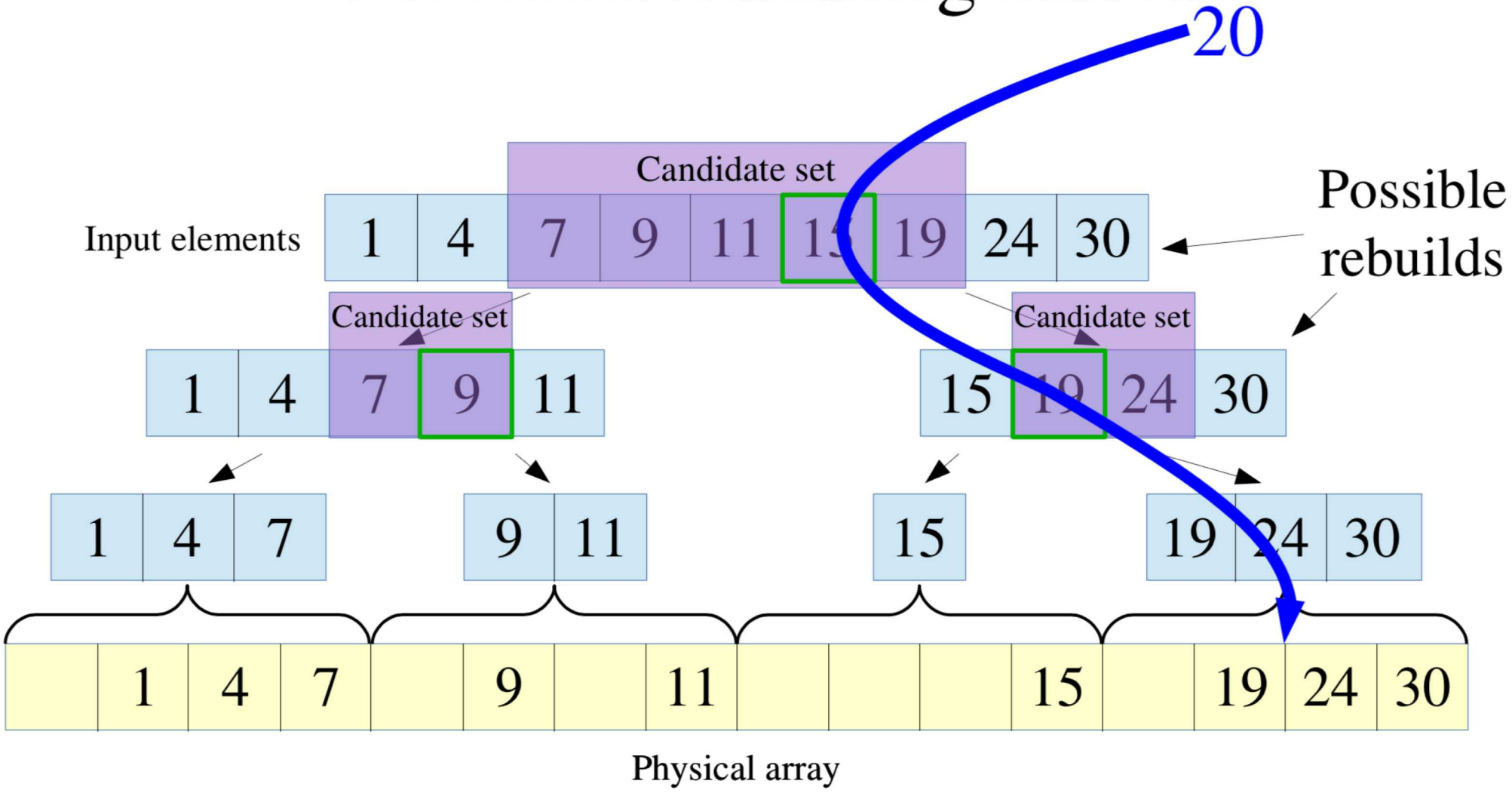
- Two goals:
  - Maintain a club leader uniformly randomly from all current club members
  - Make leader changes rare as members join and leave

1. Elect new member w/ prob $1/(n+1)$
2. Elect new leader when leader leaves

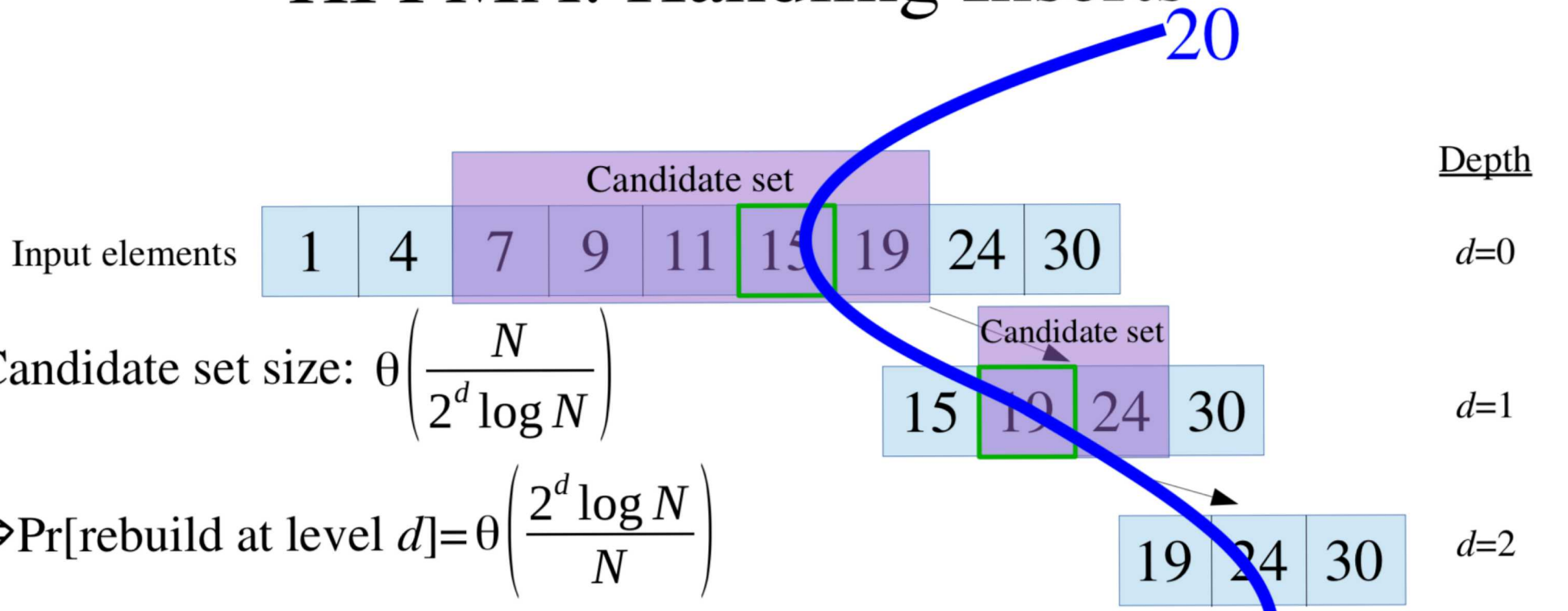
Prob[leader changes] $\approx 1/n$



HI PMA: Handling Inserts



HI PMA: Handling Inserts



Candidate set size: $\theta\left(\frac{N}{2^d \log N}\right)$

$\Rightarrow$Pr[rebuild at level d]$=\theta\left(\frac{2^d \log N}{N}\right)$

$\Rightarrow$PMA size at level d: $\theta\left(N/2^d\right)$

$\Rightarrow$E[work for rebuild at level d]$=\theta\left(\log N\right)$

E[insert cost]$=\theta\left(\log^2 N\right)$

Validation

- HI data structures cost something (but only a constant factor in theory and about 7x on initial experiments
- If there is any error in the implementation, could lose HI property
  - History independence is delicate
- How to validate an implementation

Validation

1) Pick an arbitrary element $y \in S$,
2) Build an HIPMA from scratch on $S - \{y\}$,
3) Insert y into the PMA.

Let Y be the distribution of the PMA layout after this procedure. Finally consider the following procedure: 1) Pick an arbitrary element $z \notin S$, 2) build an HI PMA from scratch on $S \cup \{z\}$, and 3) delete z from the HI PMA. Let Z be the distribution of the HI PMA layout after this procedure. If insertion and deletion are implemented correctly, then all three distributions X , Y , and Z should be identical.

Kullback-Leibler Divergence (like testing for a fair die)

- Smallest size non-trivial data structure
- Trials in parallel

Final Thoughts

Online event detection:

- Algorithms and data structures allow rapid stream monitoring using "normal" architecture such as SSDs
- Compromise between fast ingestion and queries, but can approximately have both
- Store as much as you can to get the best information

Open research questions (firehose):

- Intentional data expiration in dictionaries for infinite streams
  - Theory and practice (larger tests)

History-independence:

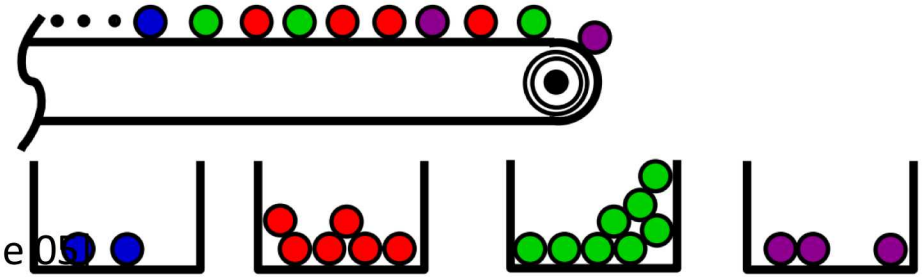
- Can have weak HI at no asymptotic cost
- Applications?

Bonus Time

- Suppose we want to find any element in a stream of size N that has a constant fraction (say 1/5) of the elements
- There can be at most 5 such elements
- If we find a count from 5 different elements, we can throw them away
  - Can do that fewer than N/5 times if don't throw all out
  - So any element with count at least N/5 still has a representative

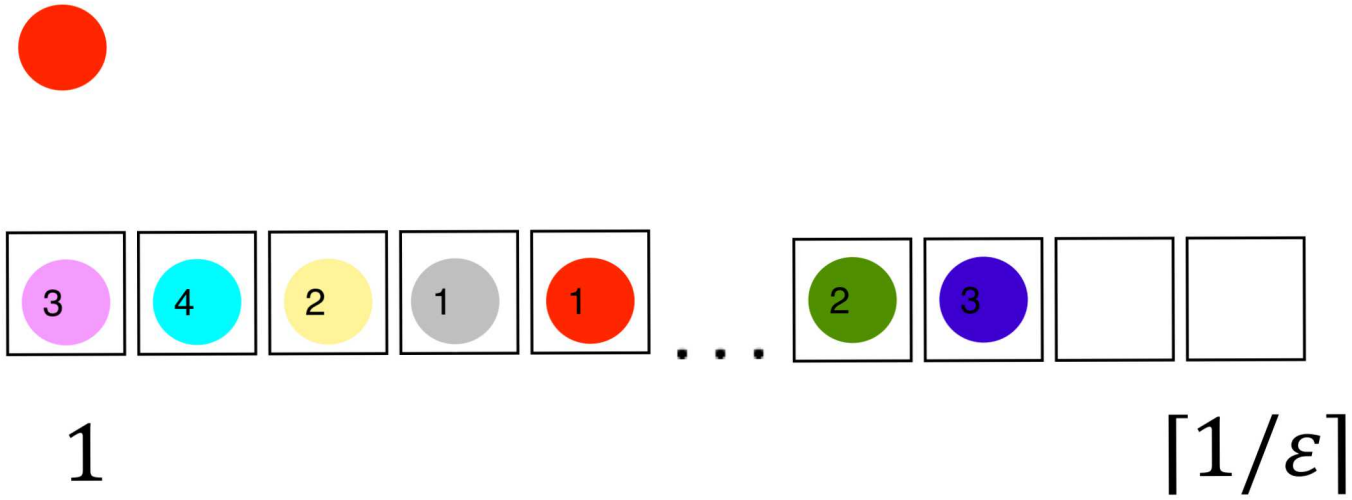
Misra Gries (MG) Algorithm

- Maintain $1/\varepsilon$ counters in memory
- When an item arrives
- if there is a counter for it, increment the counter
- if there is no counter for it
  - and there is space, add a counter and set to 1
  - otherwise, decrement all counters



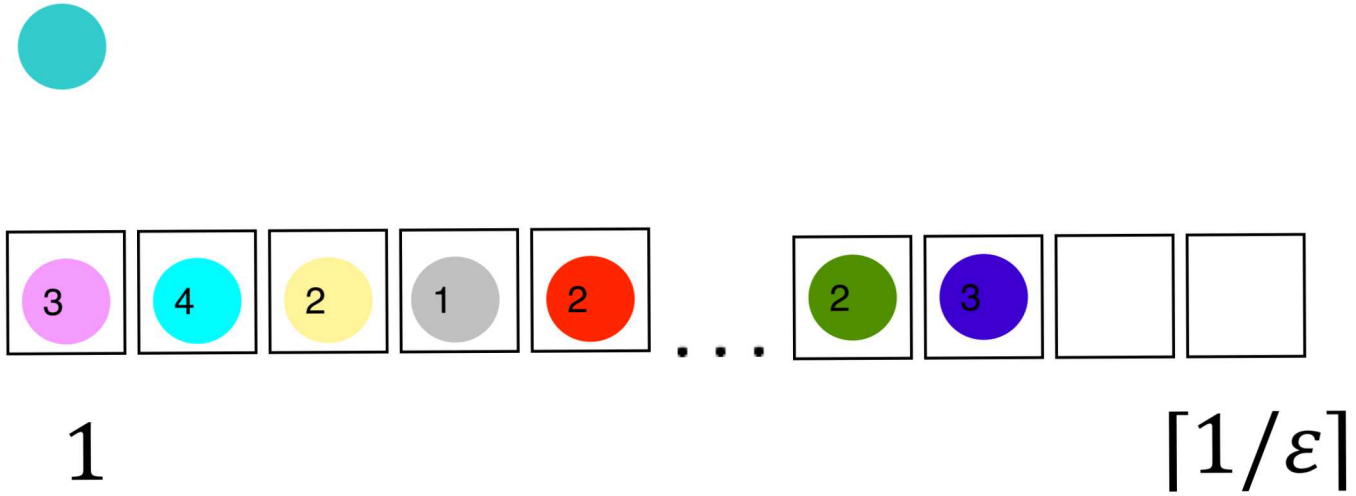
[Cormode 05]

Misra Gries (MG) Algorithm

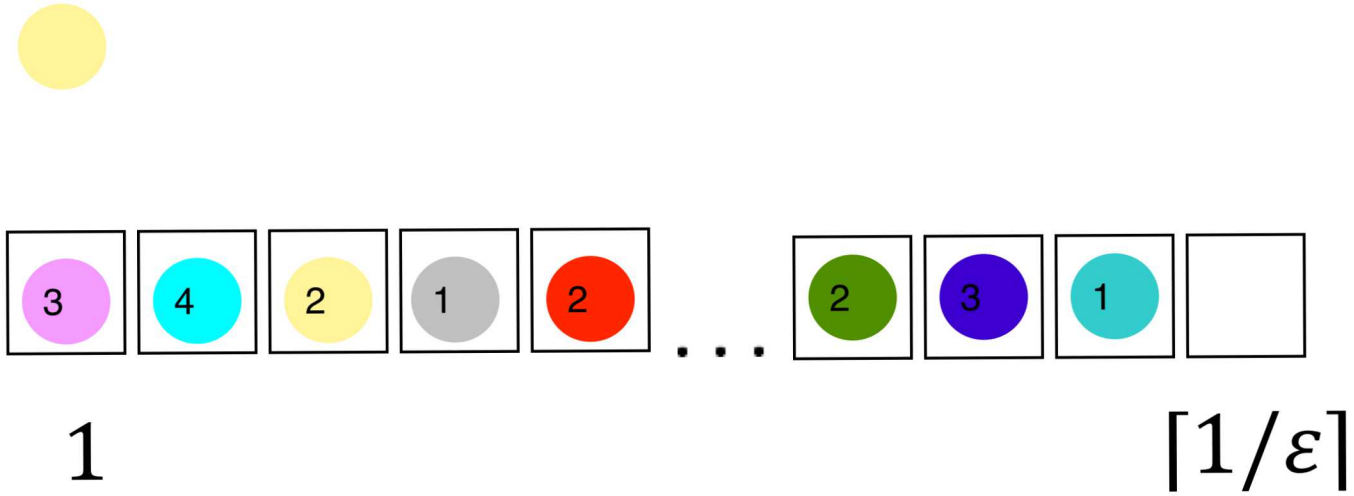


Items distinguished by color. Counts as shown

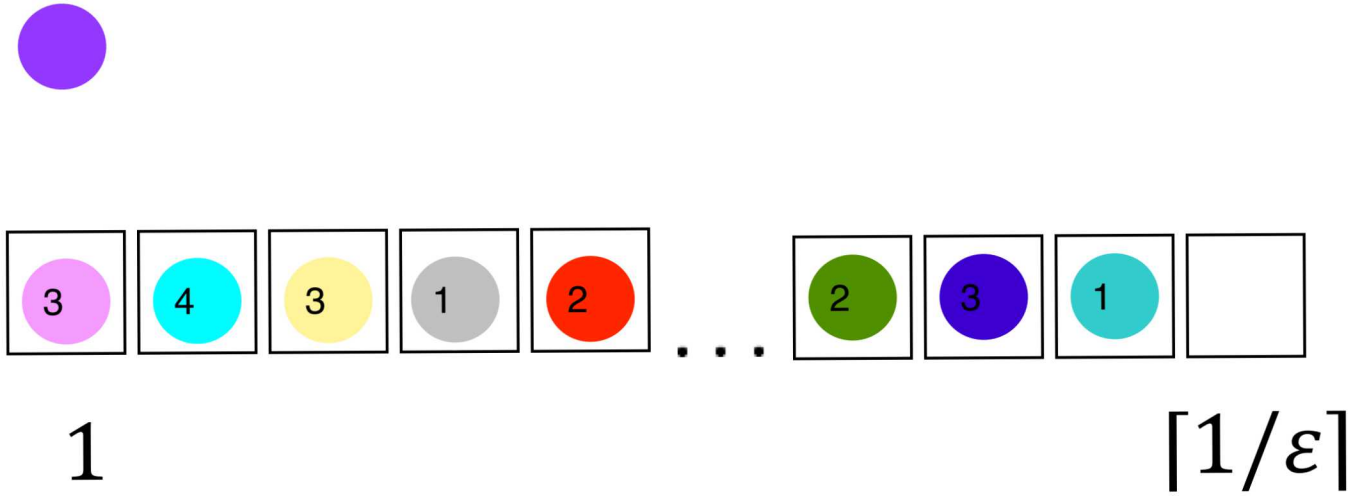
Misra Gries (MG) Algorithm



Misra Gries (MG) Algorithm

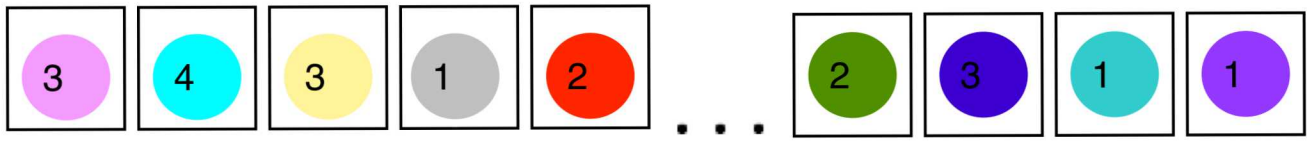


Misra Gries (MG) Algorithm



Misra Gries (MG) Algorithm

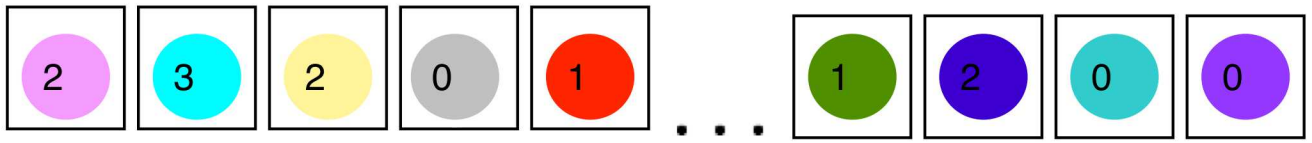
Item not in the list and there's no space



1 $\lceil 1/\varepsilon \rceil$

Misra Gries (MG) Algorithm

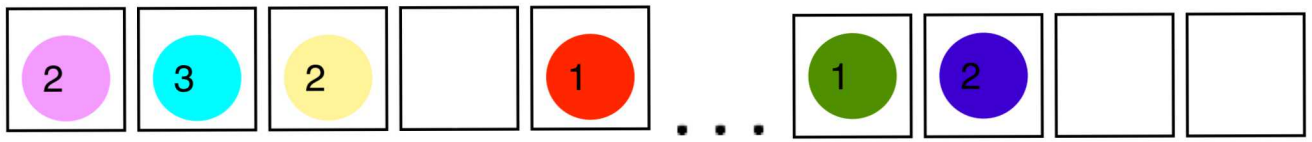
Decrement all counters



1 $\lceil 1/\varepsilon \rceil$

Misra Gries (MG) Algorithm

Remove if zero



1 $\lceil 1/\varepsilon \rceil$

External-Memory Misra Gries

Structure

- A sequence of geometrically increasing Misra-Gries tables
- The smallest table is in memory and is of size M, the last table is of size $\lceil 1/\varepsilon \rceil$
- Total levels = $O(\log 1/\varepsilon M)$

Algorithm

- The top level receives its input from the stream
- Decrements from one level are inputs to the level below
- Decrements from the last level leave the structure

Final Thoughts

Online event detection:

- Algorithms and data structures allow rapid stream monitoring using "normal" architecture such as SSDs
- Compromise between fast ingestion and queries, but can approximately have both
- Store as much as you can to get the best information

Open research questions (firehose):

- Intentional data expiration in dictionaries for infinite streams
  - Theory and practice (larger tests)

History-independence:

- Can have weak HI at no asymptotic cost
- Applications?