SAND2021-15267PE

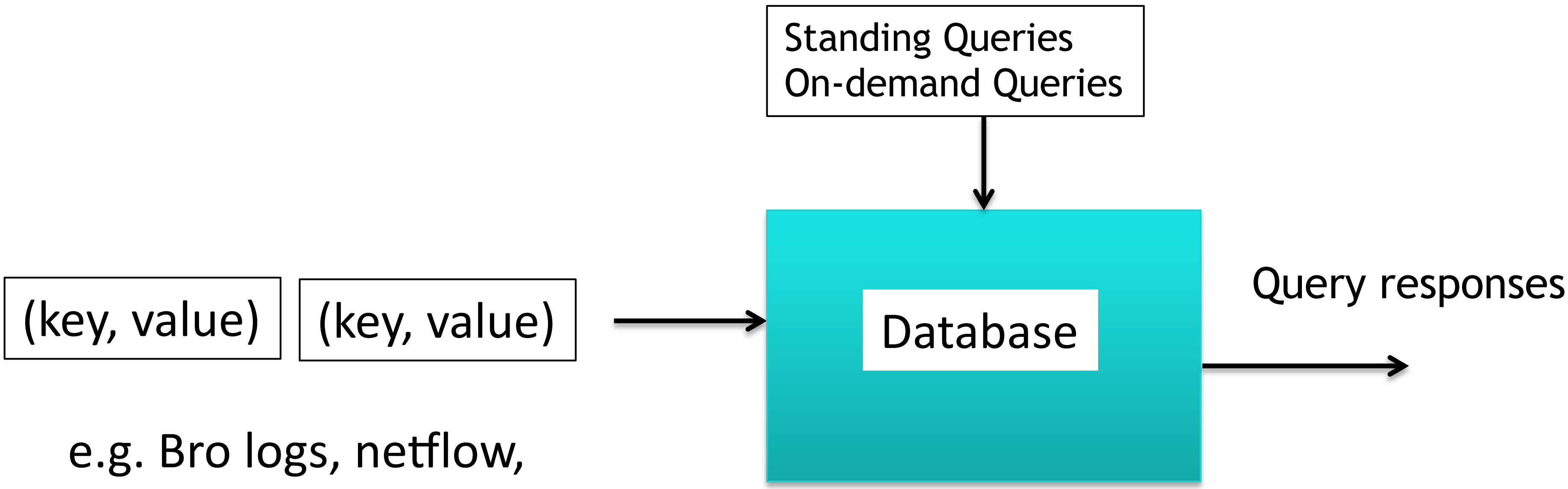
Advanced Data Structures for Monitoring Cyber Streams

Michael Bender (Stony Brook U)
Jon Berry (Sandia National Laboratories)
Martin Farach-Colton (Rutgers)
Rob Johnson (VMWare Research)
Tom Kroeger (Sandia National Laboratories)
Prashant Pandey (VMWare Research)
Cynthia Phillips, Sandia National Laboratories
Shikha Singh (Williams College)

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.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & 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.

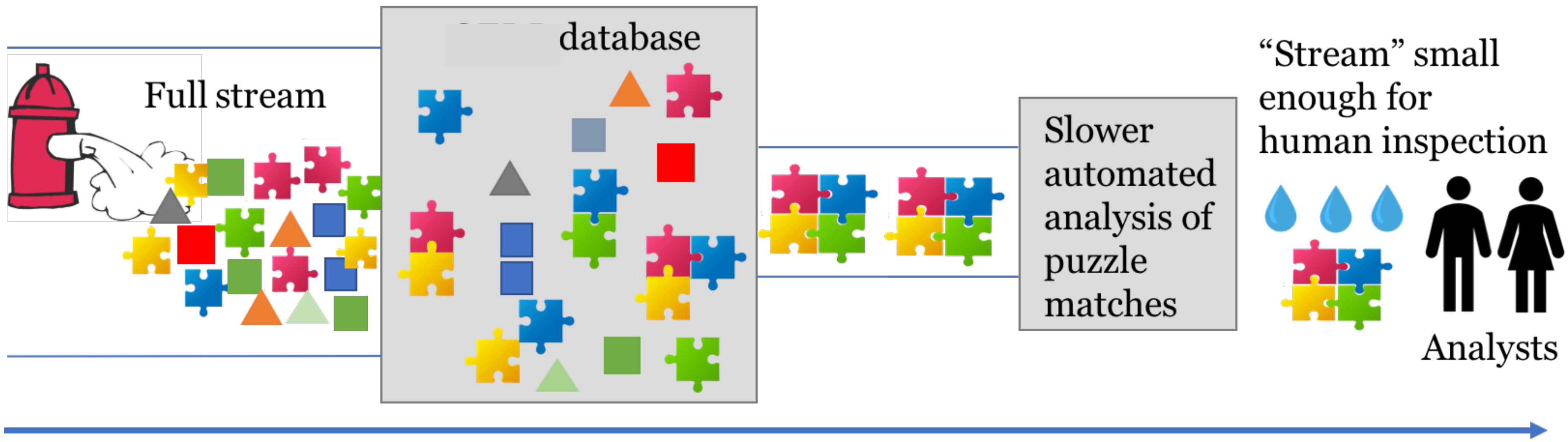
Cyber Streams and Analysis



e.g. Bro logs, netflow, relationships

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

Standing Queries



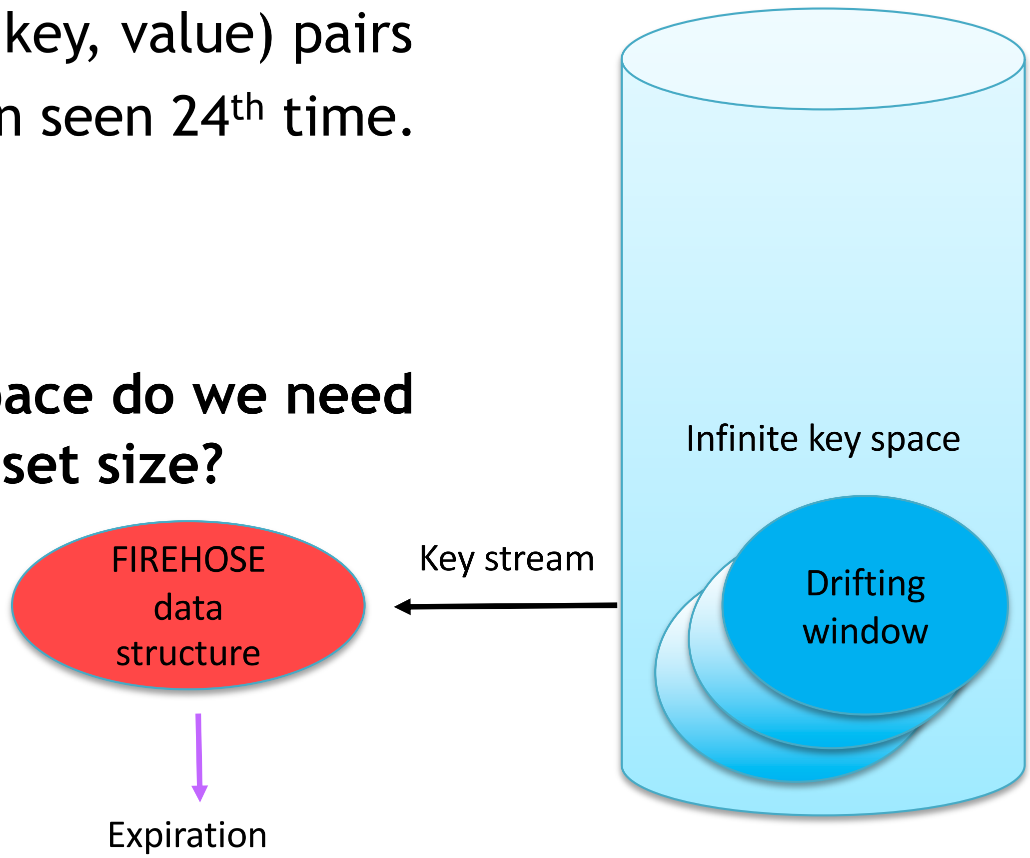
Database requirements:

- No false negatives
- Limited false positives
- Immediate response preferred
- Keep up with a fast stream (millions/sec or faster)
- 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.

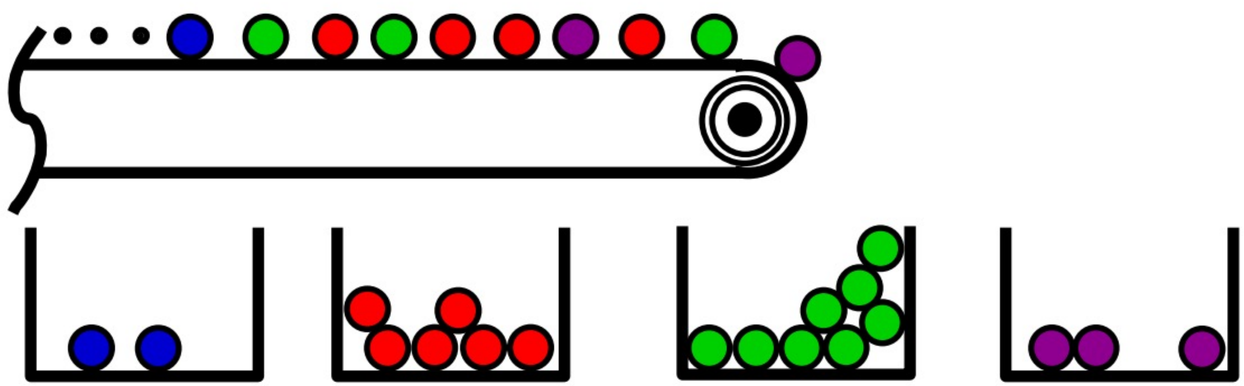
How much working space do we need relative to the active set size?



http://firehose.sandia.gov/

Heavy-Hitters Problem

- Also called **the frequent items problem**
- Given a finite stream of N items, find ones that appear most frequently, e.g., items that occur 10% of the time
- Formally, report all items that occur at least ϕN times



Sandia National Laboratories

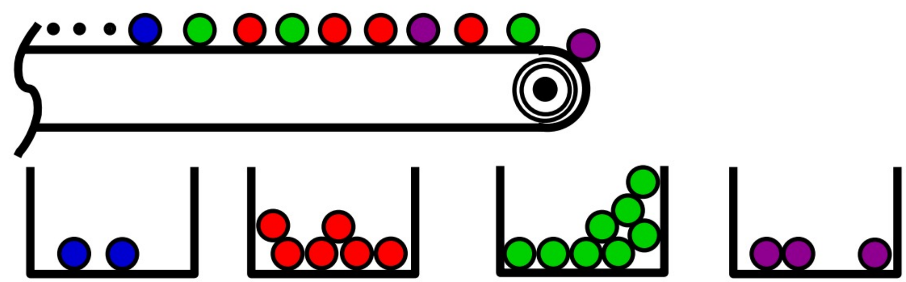
Misra Gries Observations

- Suppose we want to find any element in a stream of size N that has a constant fraction (say ϕ = 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

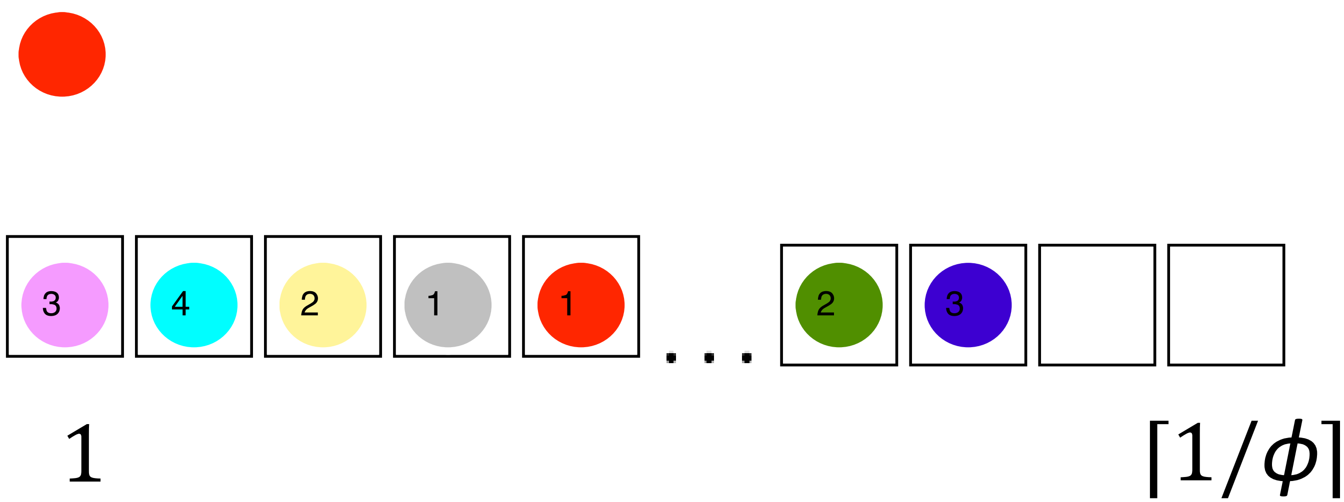
Misra Gries (MG) Algorithm

- Maintain $1/\phi$ 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
- In a second pass, get actual counts for items left from first pass

[Cormode 05]

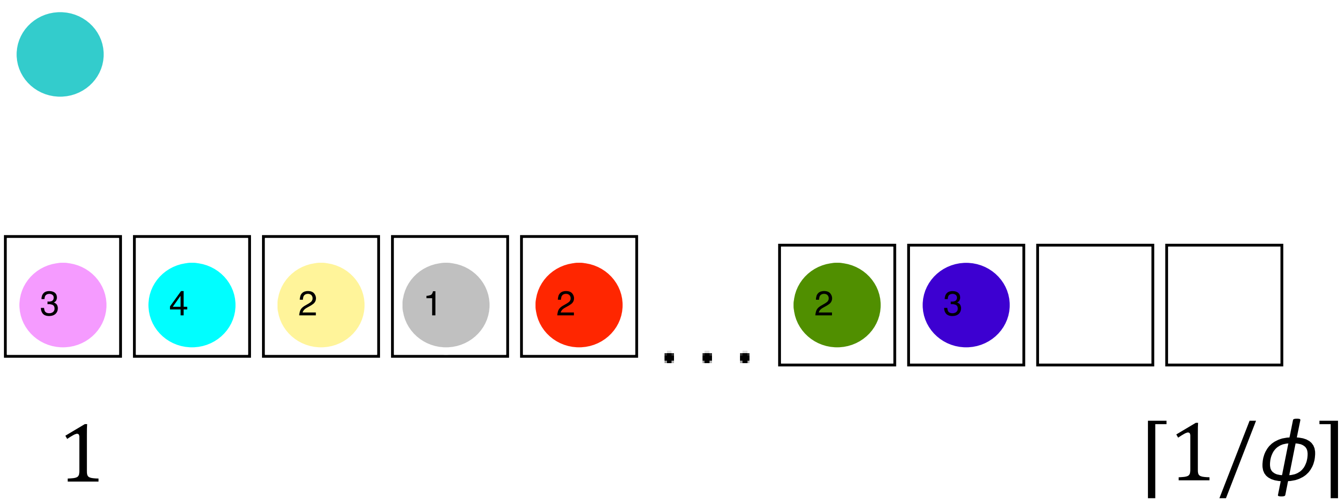


Misra Gries (MG) Algorithm



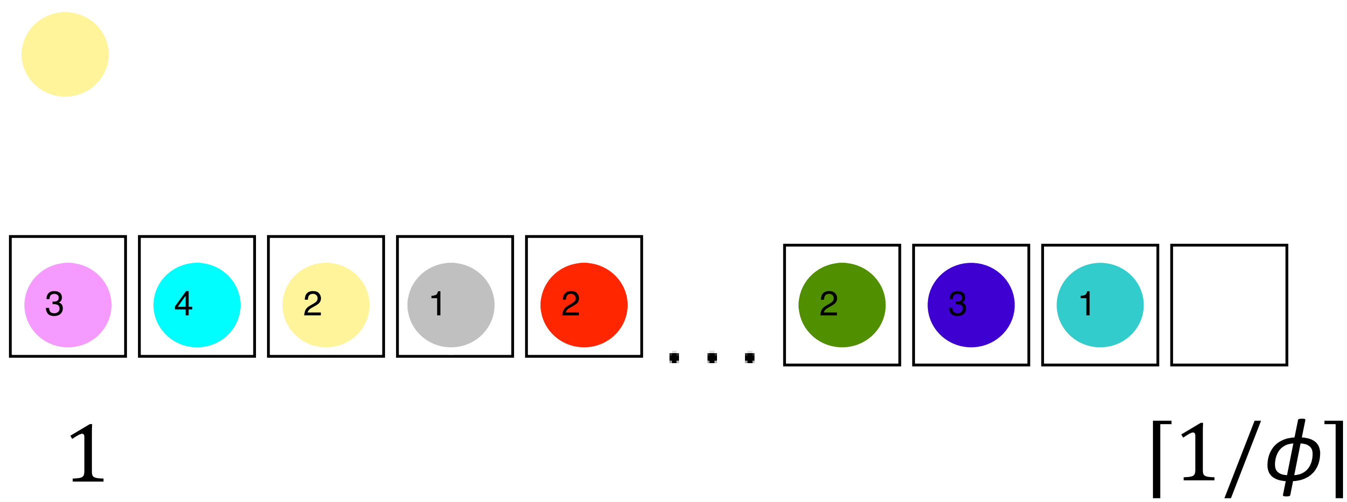
Items distinguished by color. Counts as shown

Misra Gries (MG) Algorithm



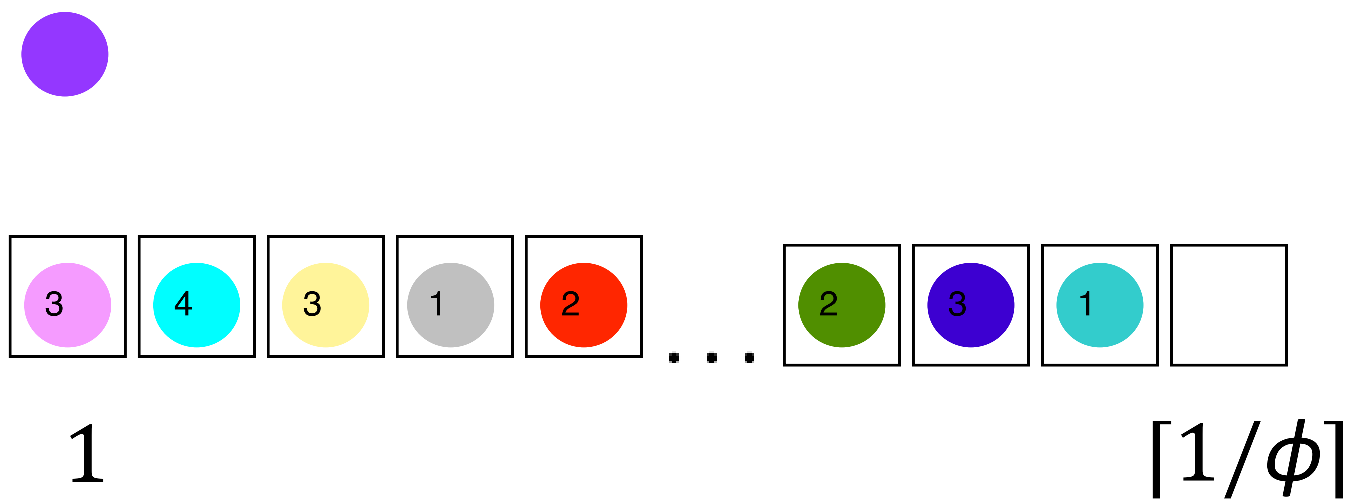
Sandia National Laboratories

Misra Gries (MG) Algorithm



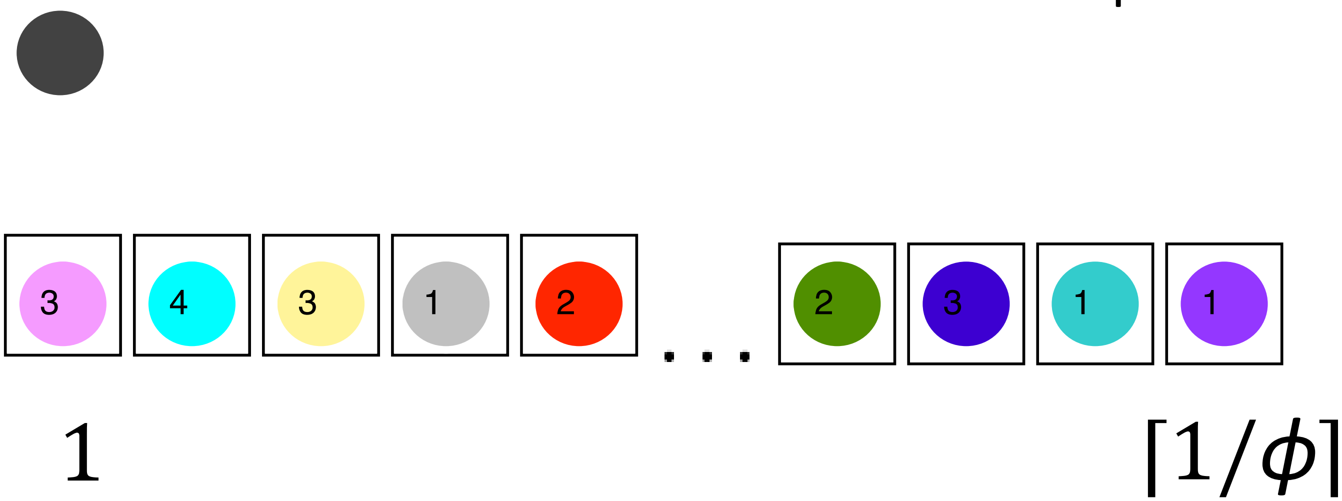
Sandia National Laboratories

Misra Gries (MG) Algorithm



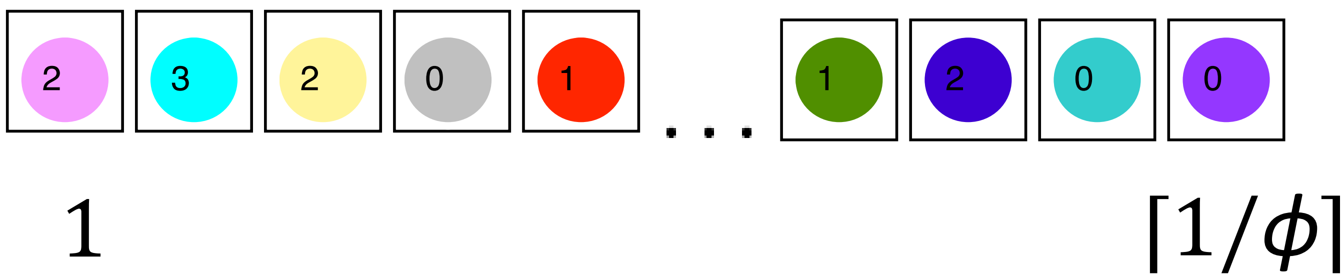
Misra Gries (MG) Algorithm

Item not in the list and there's no space



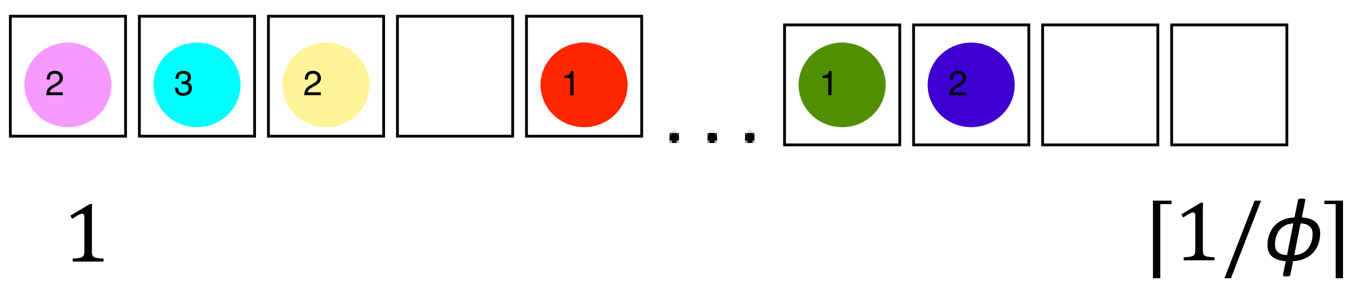
Misra Gries (MG) Algorithm

Decrement all counters



Misra Gries (MG) Algorithm

Remove if zero



Problems with Mishra-Gries for Us

- Requires 2 passes
  - Slow and real cyber streams are infinite
- Requires $\lceil 1/\phi \rceil$ space. For Firehose, ϕ = 24/N so this requires Ω(N) space.

Academic Streaming

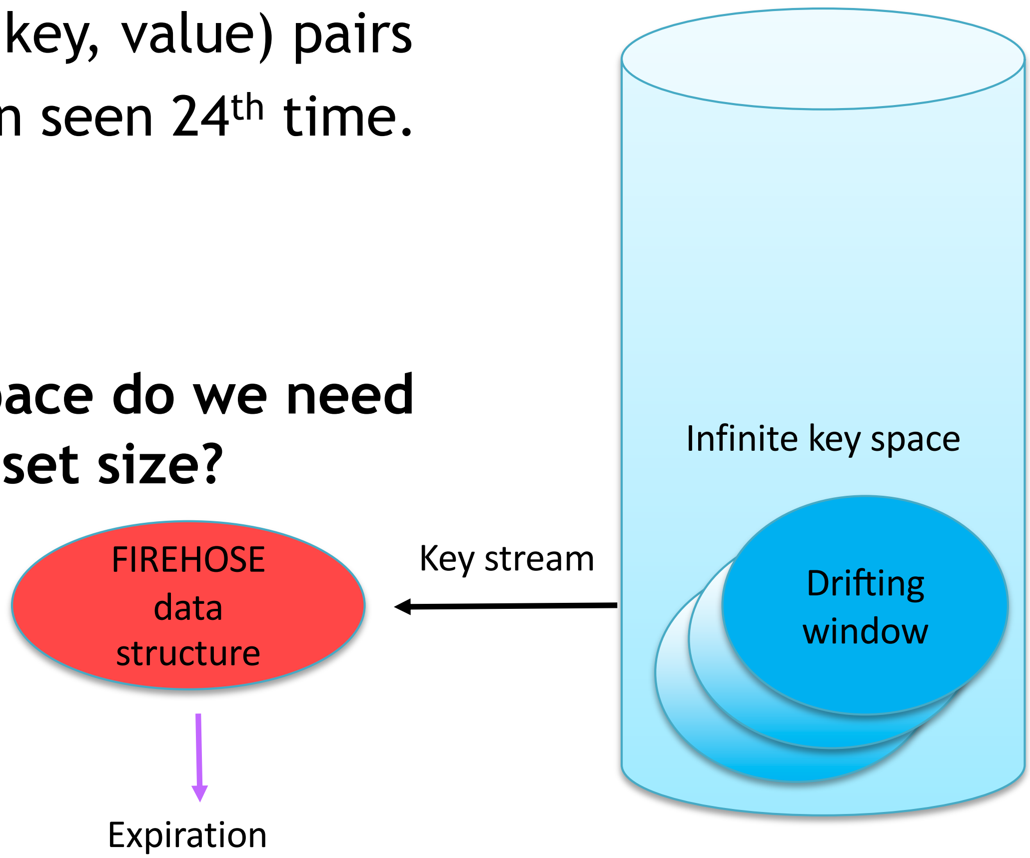
When there are large lower bounds (space required for an exact solution):

- Use more than fixed (constant) space, but as little as possible
- Use multiple passes
- Approximation (usually randomized)
  - Trade off space for accuracy [Alon et al. 96, Berinde et al. 10, Bhattacharyya et al. 16, Bose et al. 03, Braverman et al. 16, Charikar et al. 02, 05, Demaine et al. 02, Dimitropoulos et al. 08, Larsen et al. 16, Manku et al. 02., Misra and Gries. 82, etc.]
- But we require no false negatives (no approximation that drops)
- Need fast response, eventually on infinite streams (no 2-pass)
- Constant space (e.g. the size of RAM) will not be enough

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

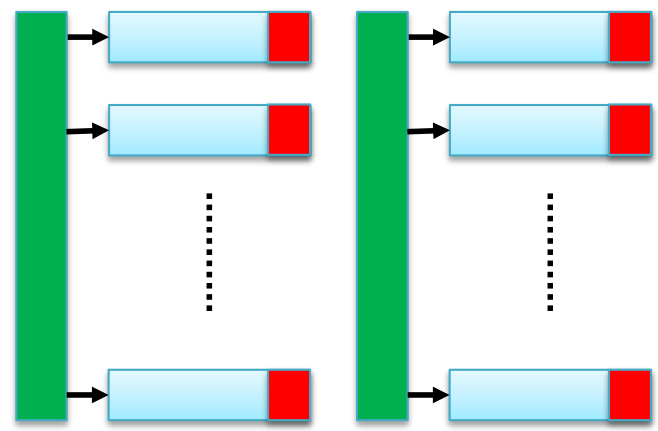
- Testing with benchmark reference implementation in Waterslide
  - 50M keys (varying counts)
  - Stable window
- Accuracy of cyber-analytics depends on keeping enough data
- Difficult to determine what to throw away
  - Most keys act the same at their start
- Keep as much data as we can!

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**

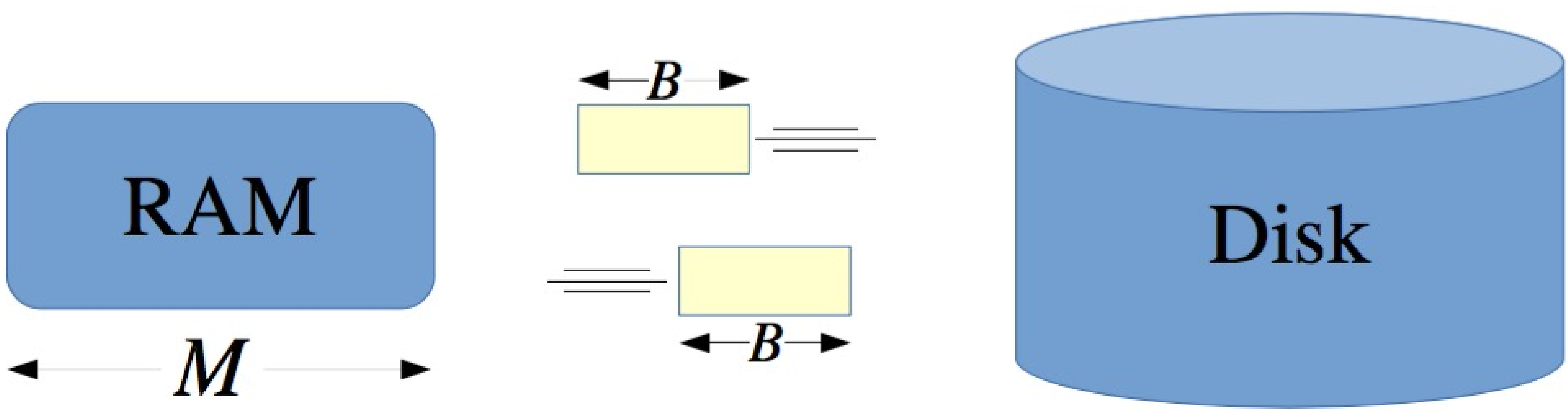
- **As active generator window grows, FIREHOSE accuracy quickly goes to zero**

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

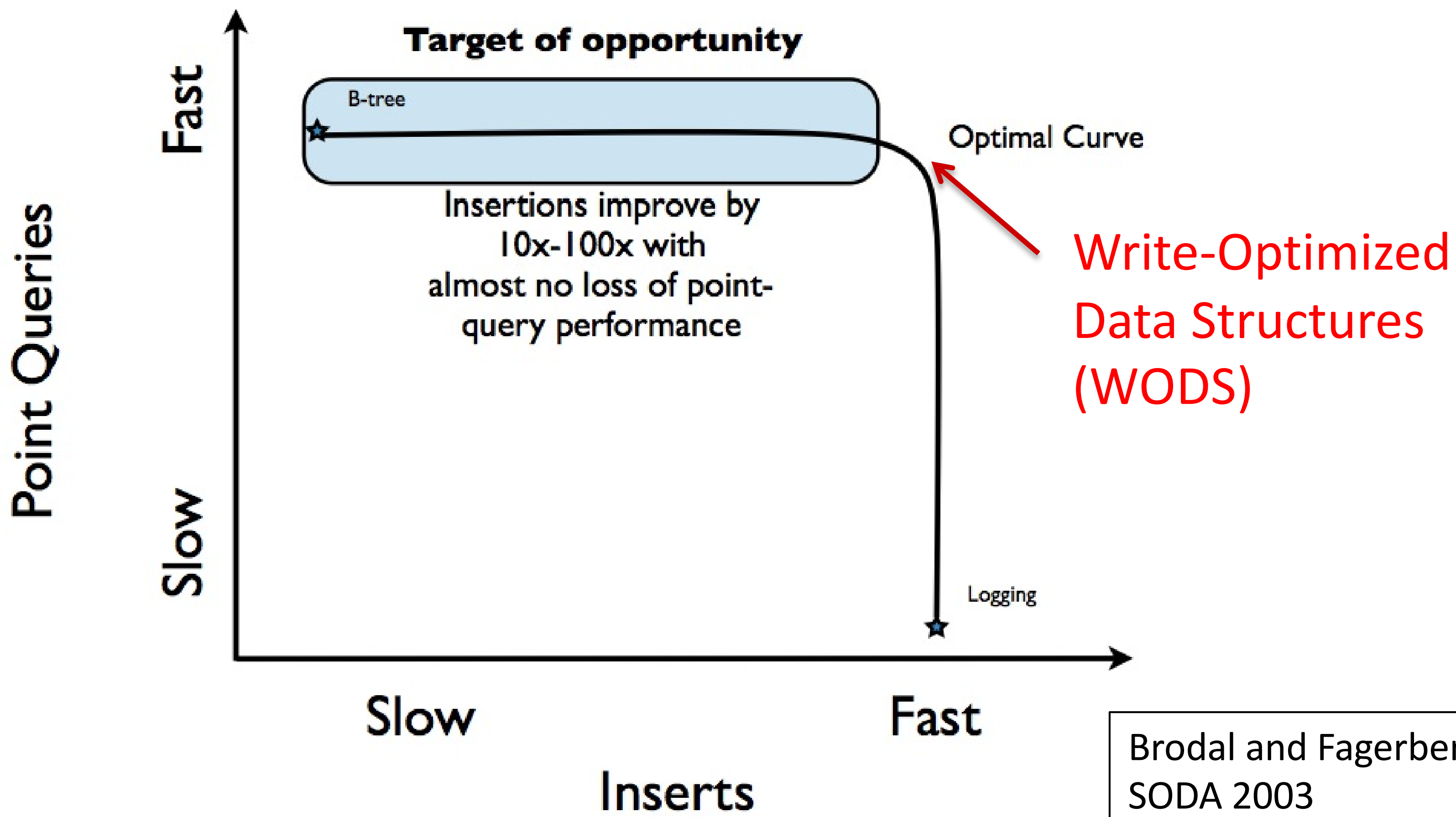
Sandia National Laboratories

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



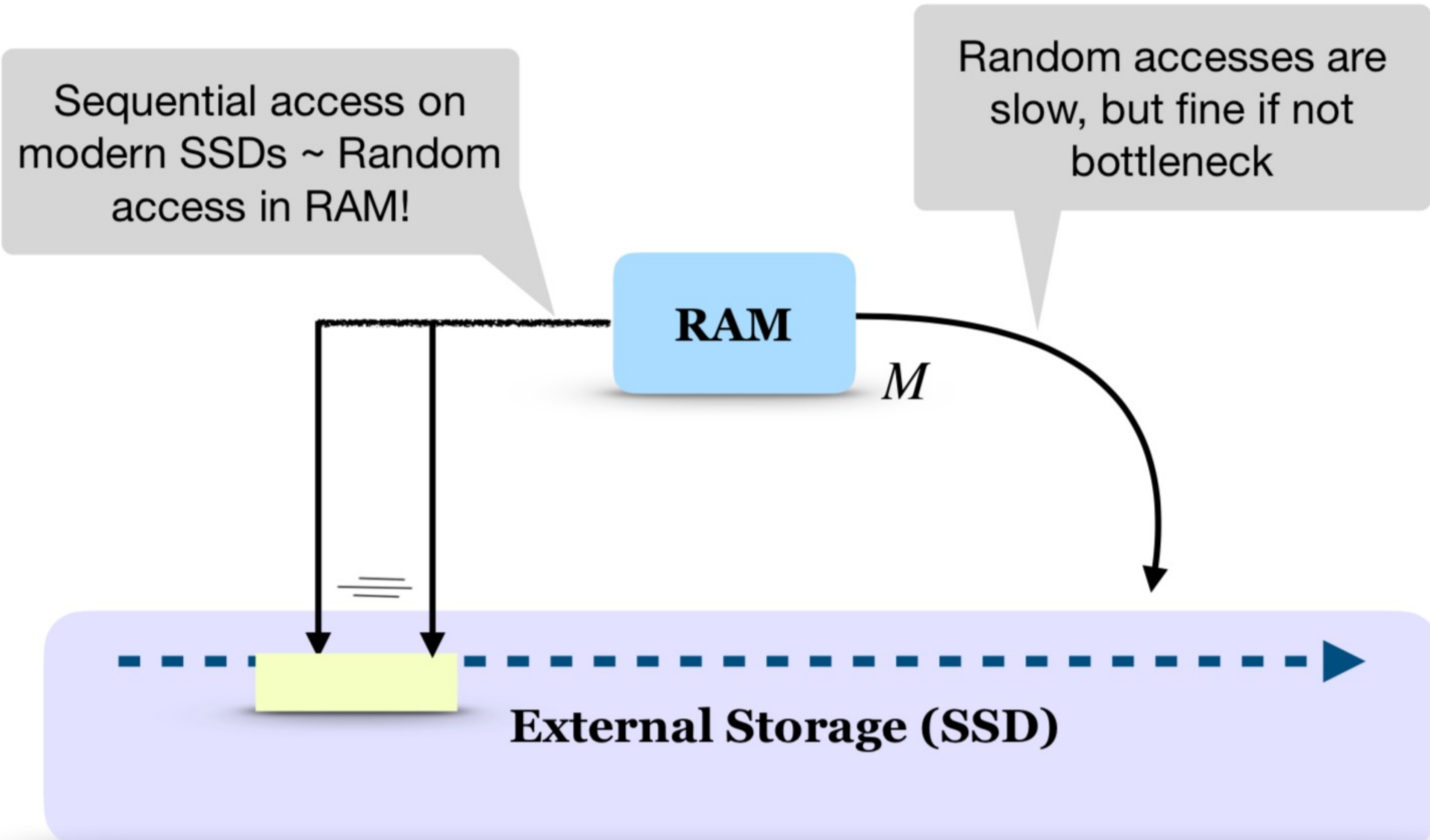
Write Optimization



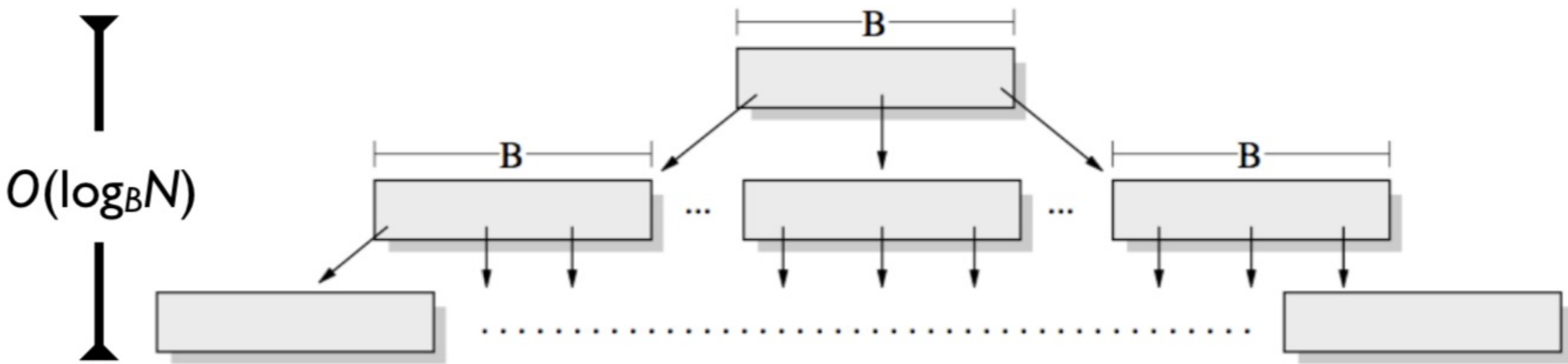
Brodal and Fagerberg
SODA 2003

- The basis for TokuDB

Modern External Memory: SSDs



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-Optimized Data Structures

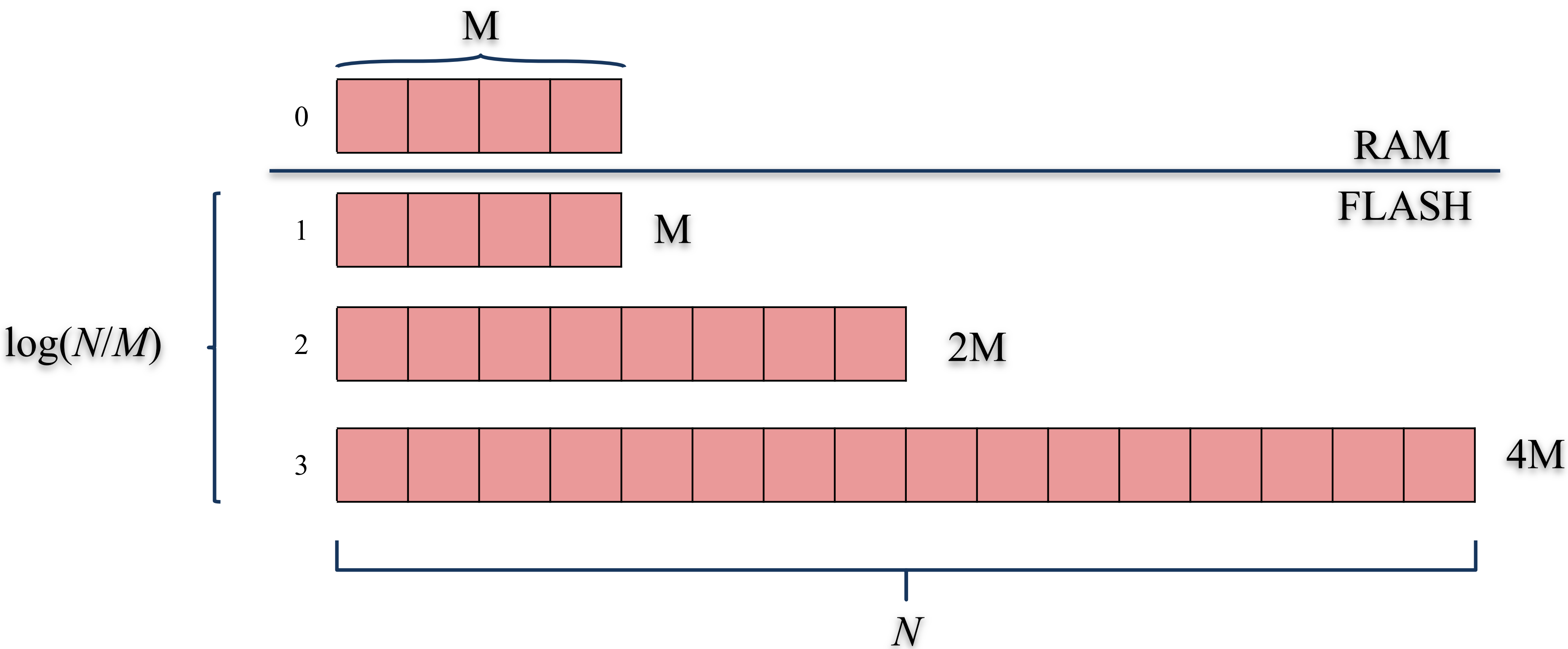
Write optimized data structures like COLA, cascade filters, etc. (WODs) let you do fast inserts and B-tree like queries

Amortized complexity: for a data structure with N elements

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

Write optimization: Cascade filter

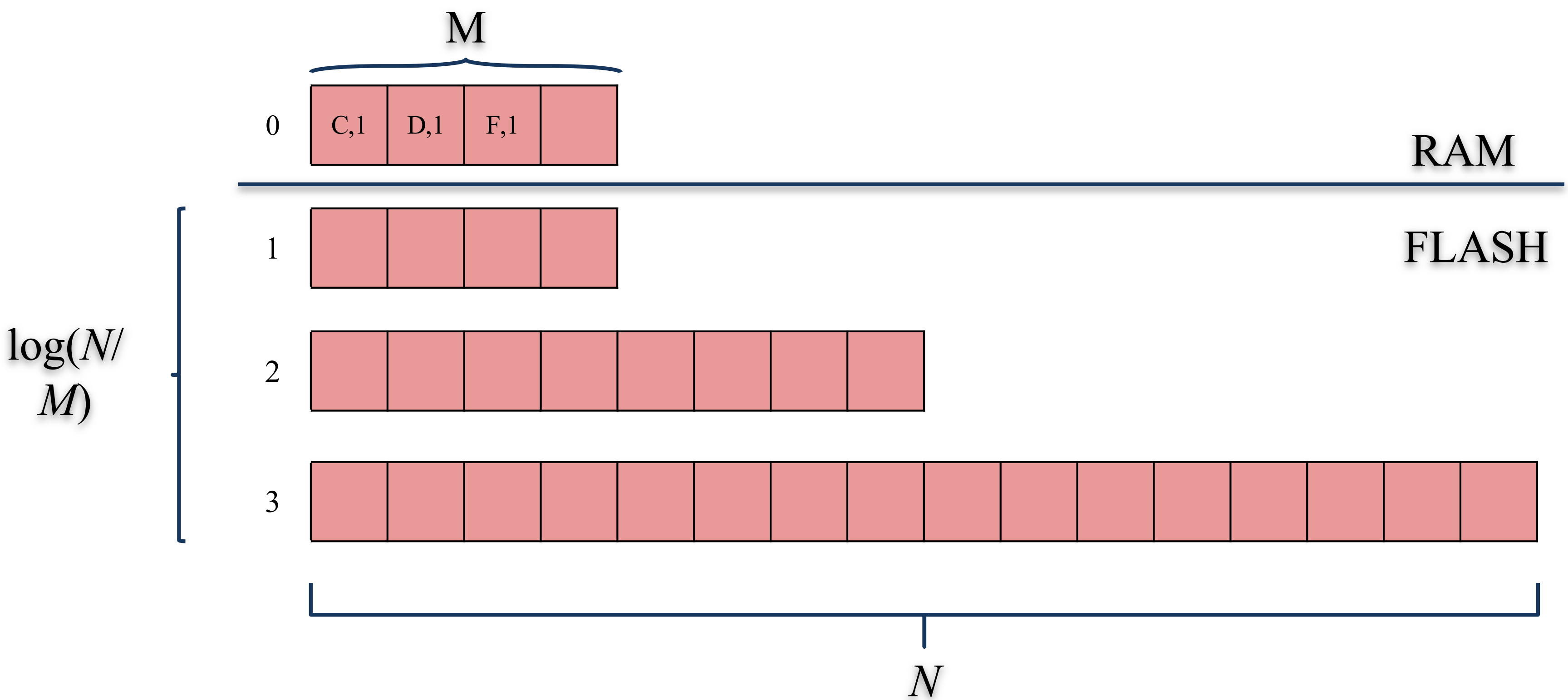
[Bender et al. 12, Pandey et al. 17]



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

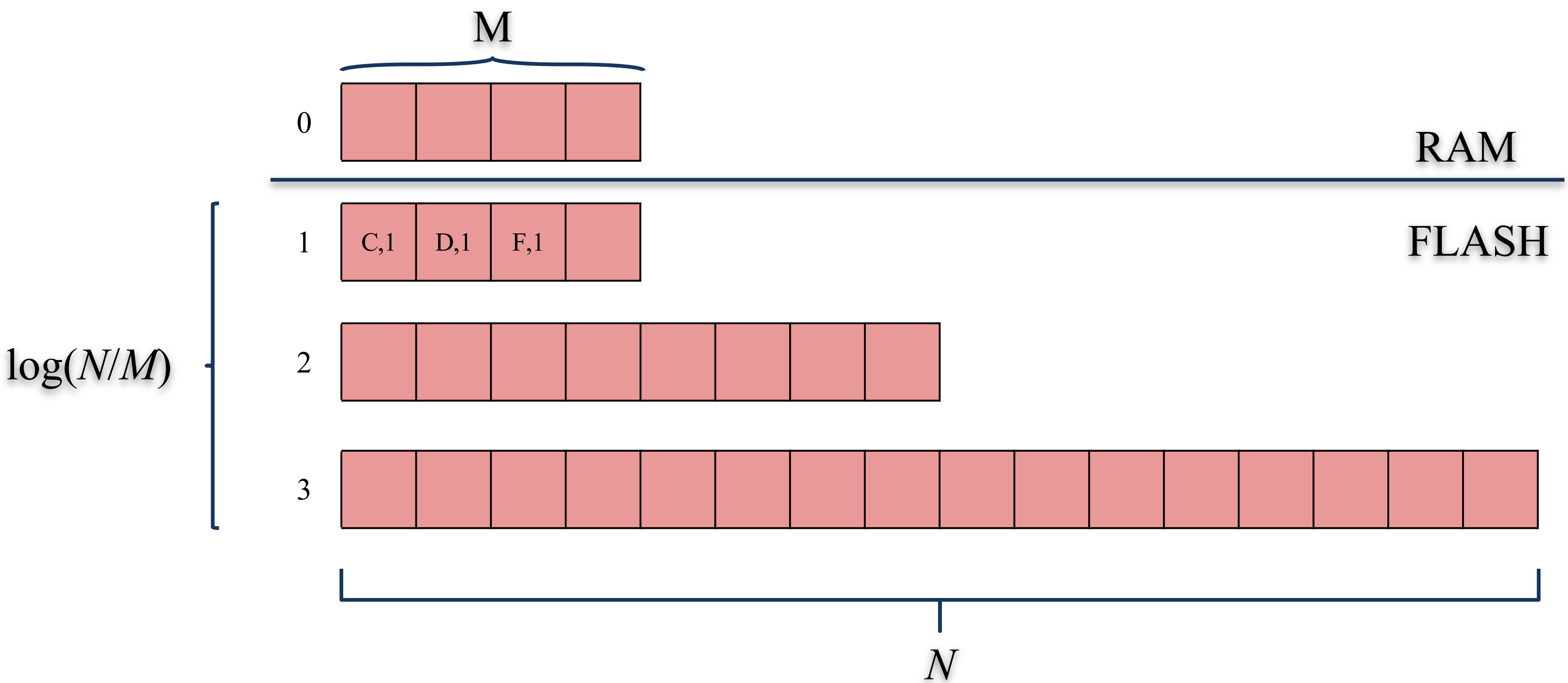
e.g. N = 1T
M = 8B
8 levels

Ingestion "cascades"



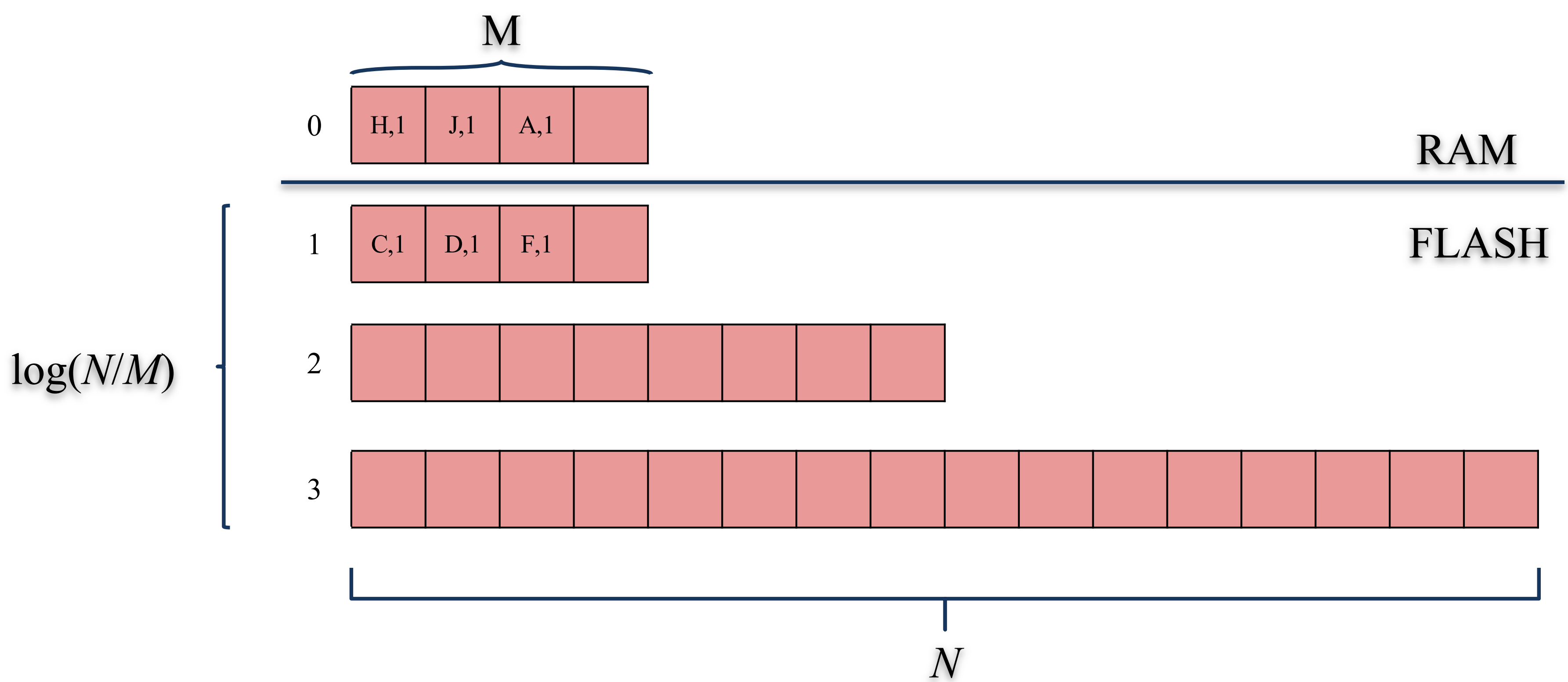
- Items are first inserted into the in-memory hash table.
- When the in-memory table reaches maximum load factor it flushes

Ingestion "cascades"



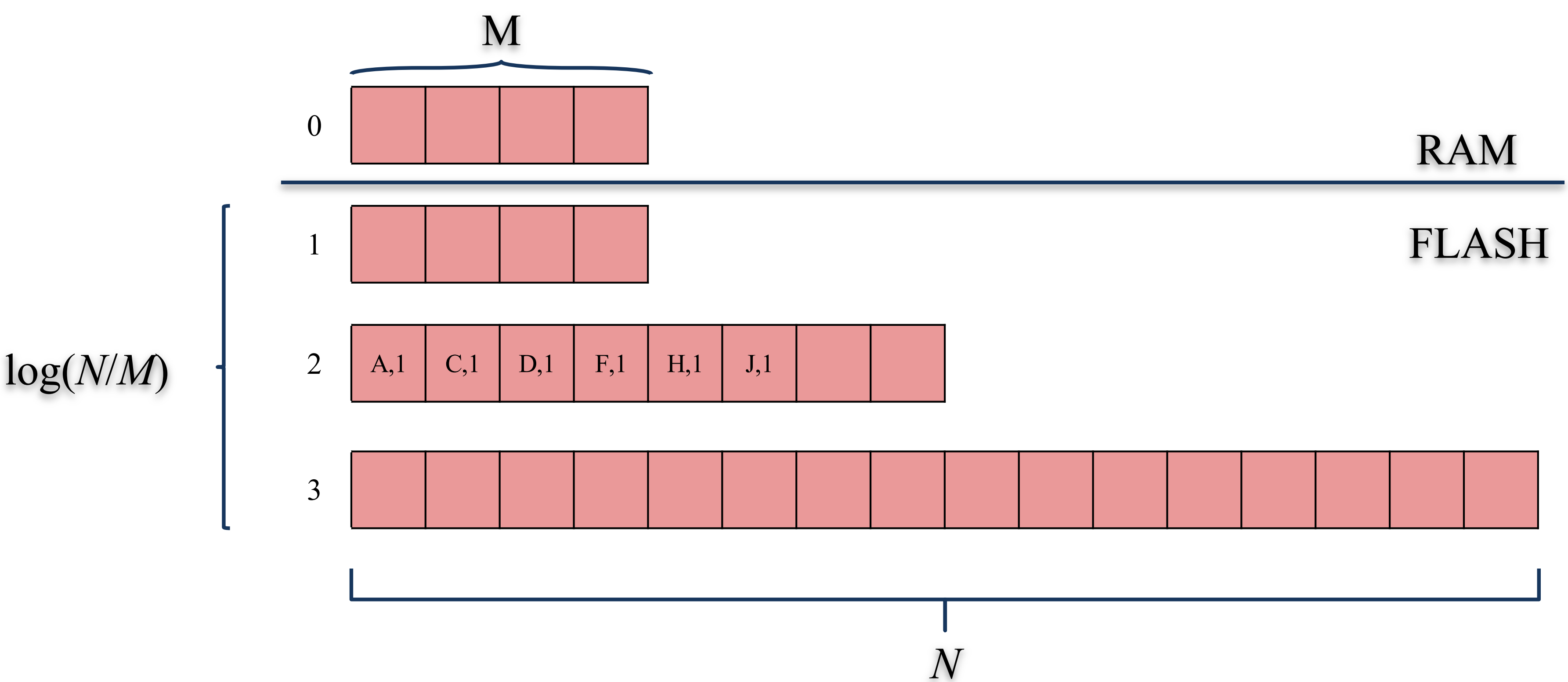
- During a flush, find the smallest i such that the items in $l_0, \ldots, l_i$ can be merged into level i.

Ingestion "cascades"

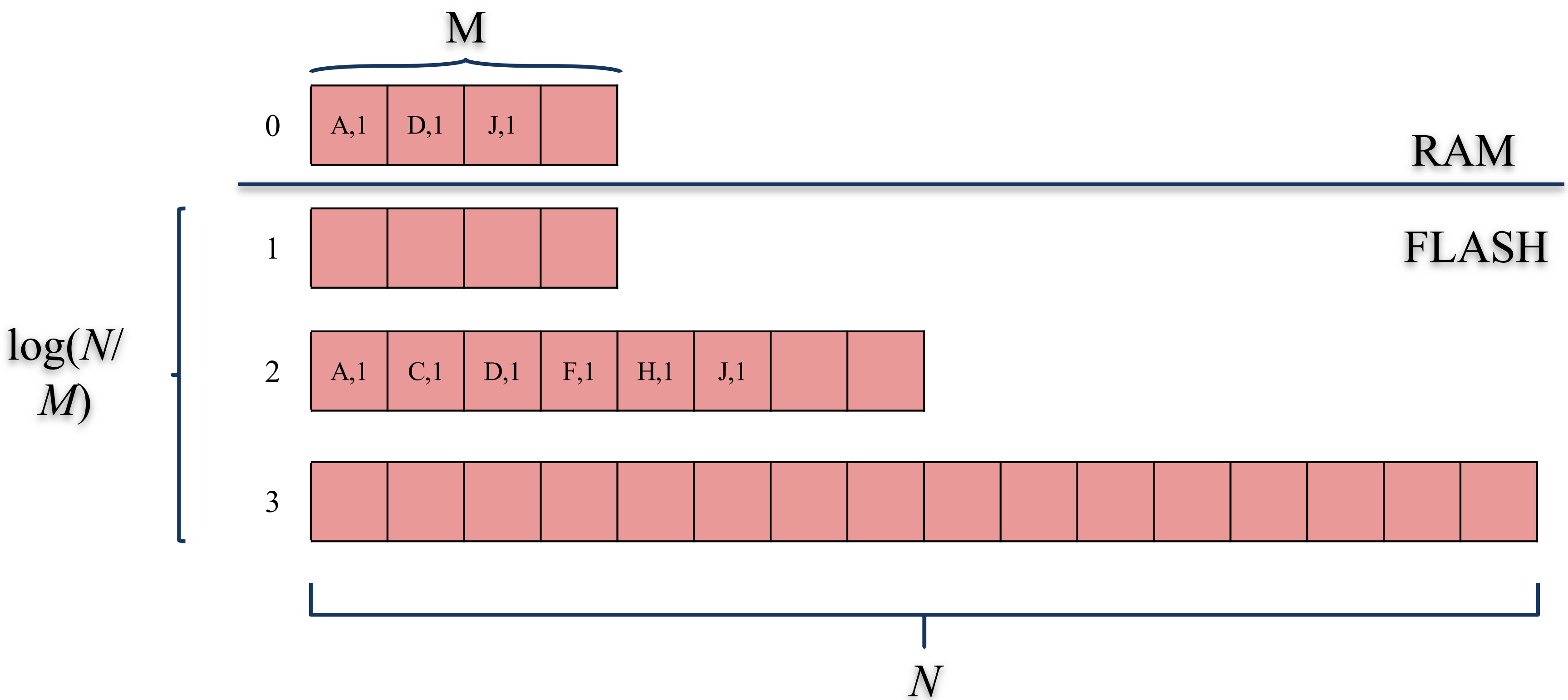


Sandia National Laboratories

Ingestion "cascades"

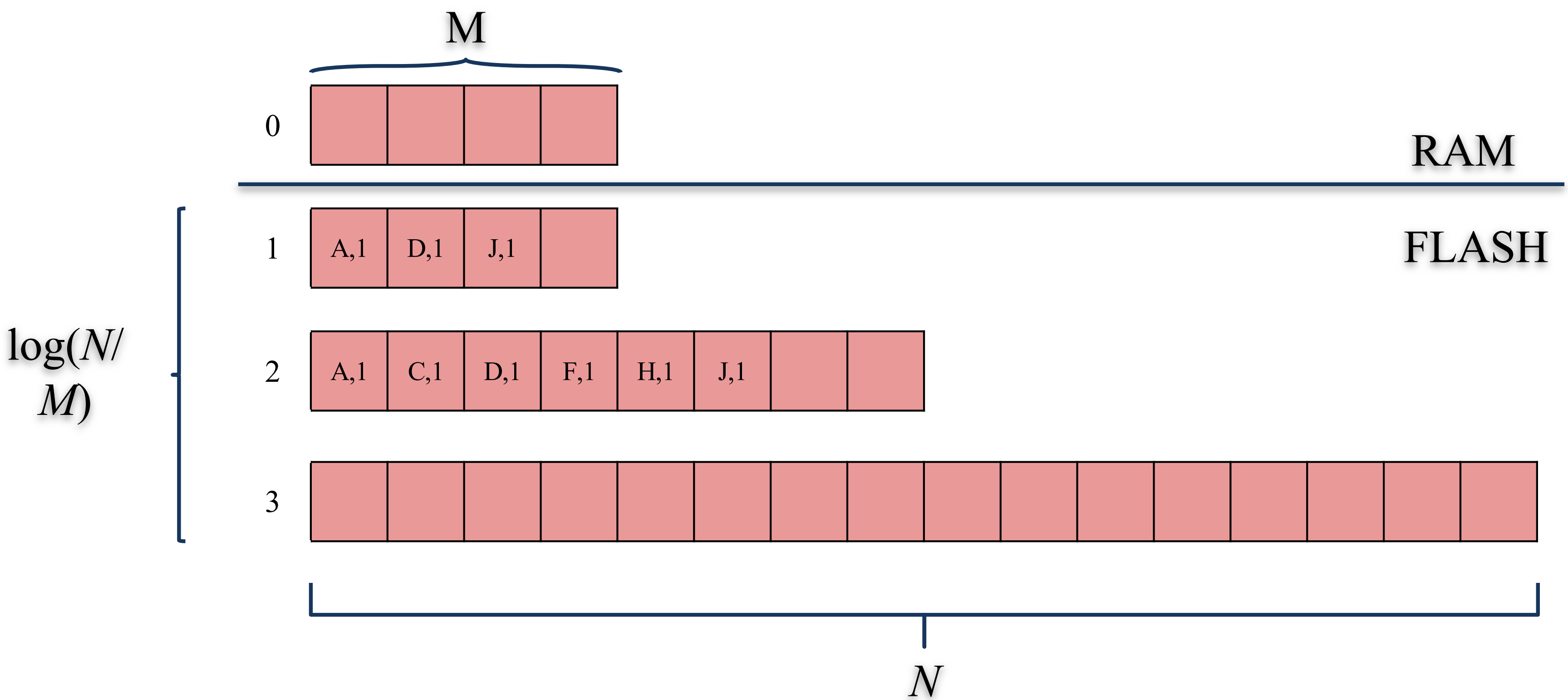


Ingestion "cascades"

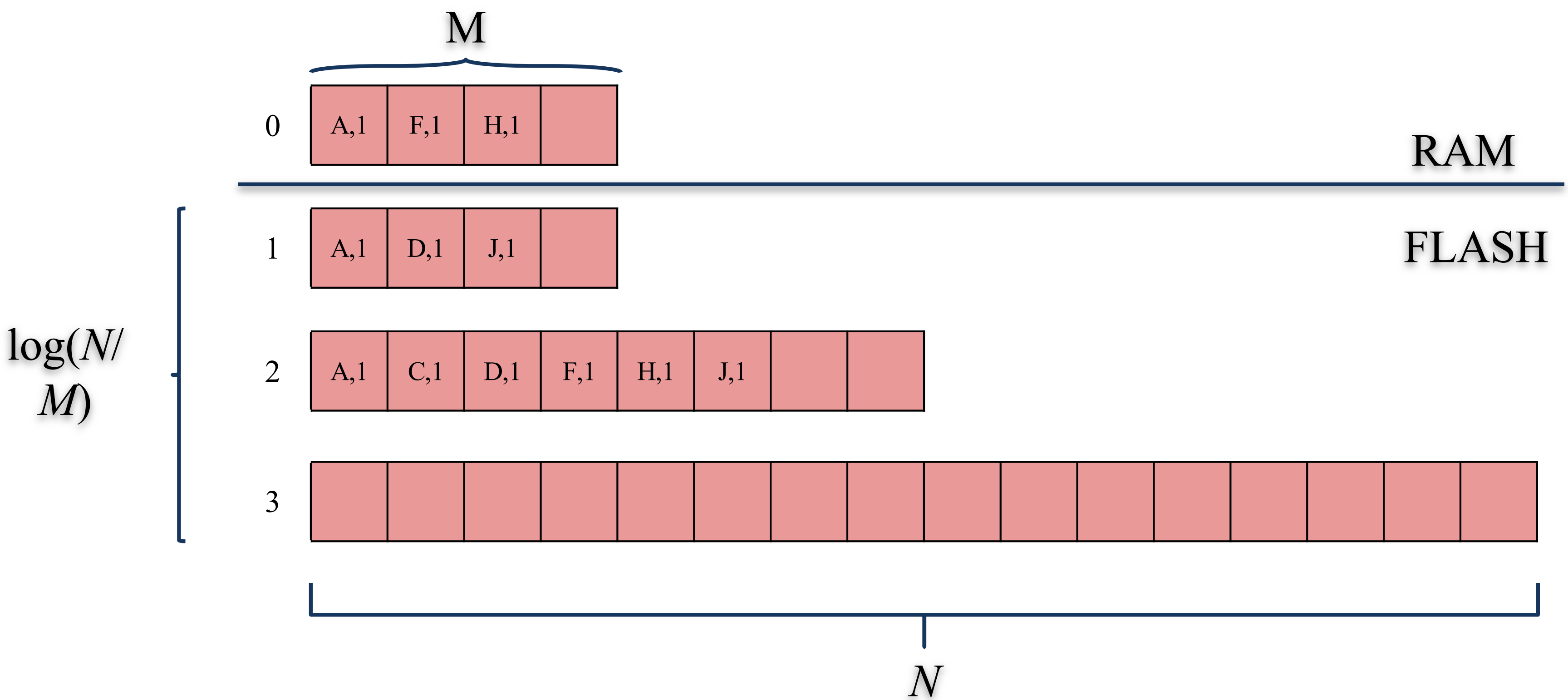


Sandia National Laboratories

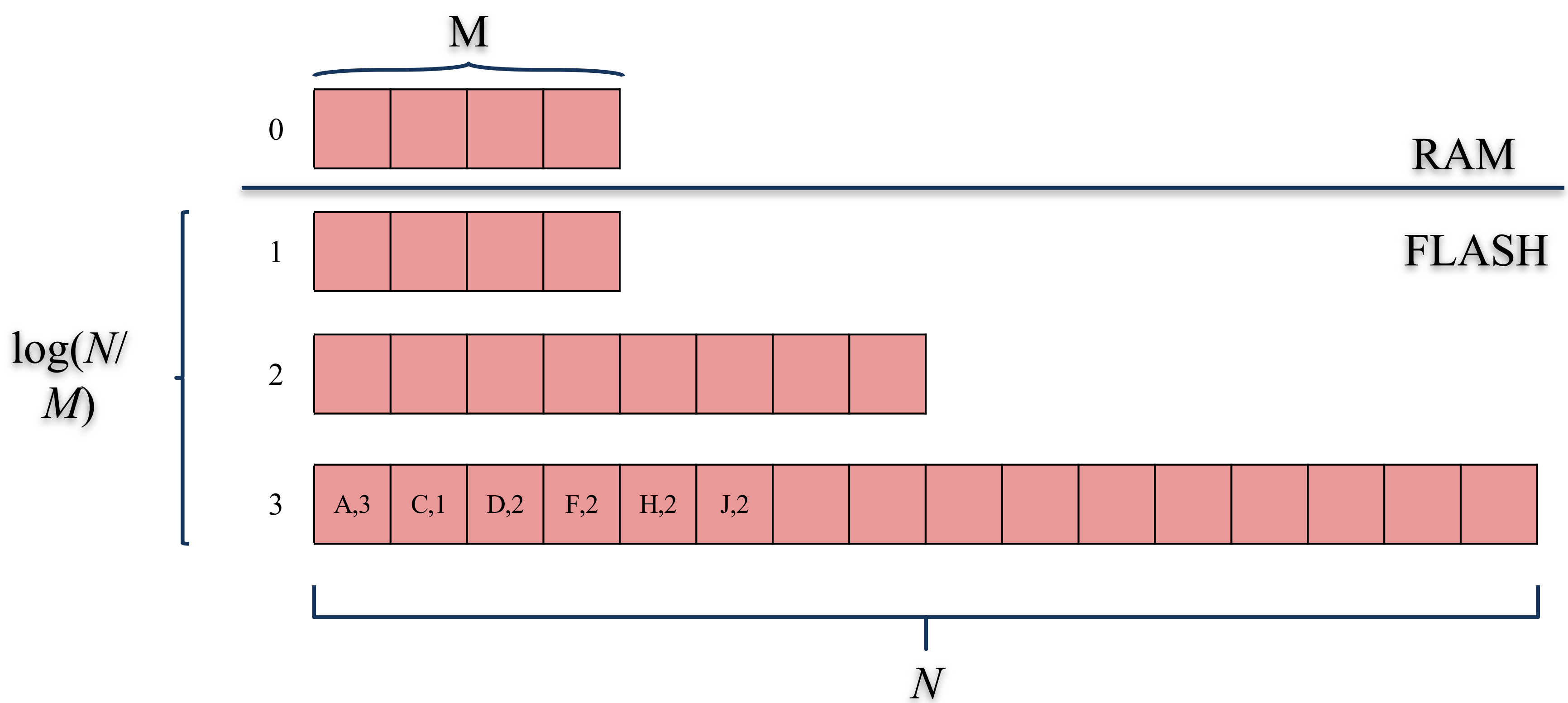
Ingestion "cascades"



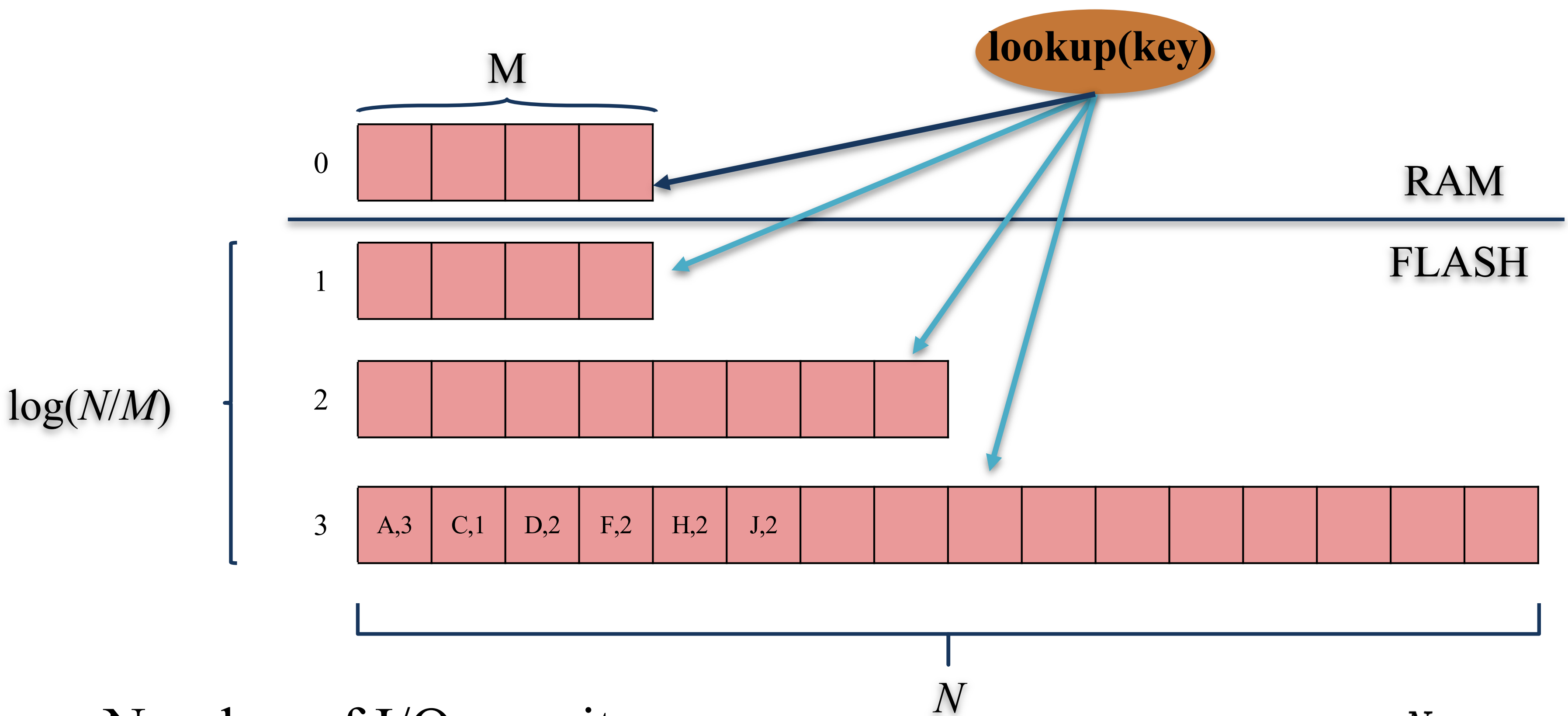
Ingestion "cascades"



Ingestion “cascades”



Cascade filter Performance

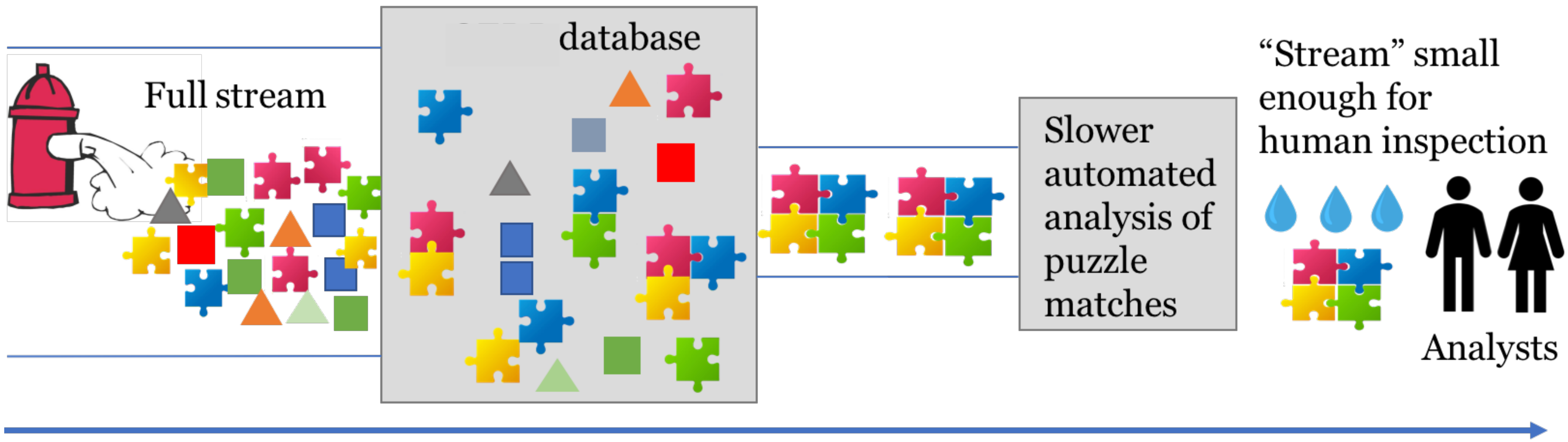


Number of I/Os per item:

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

Look up: $O(\log(\frac{N}{M}))$ Queries too slow for us

Reminder: Standing Queries

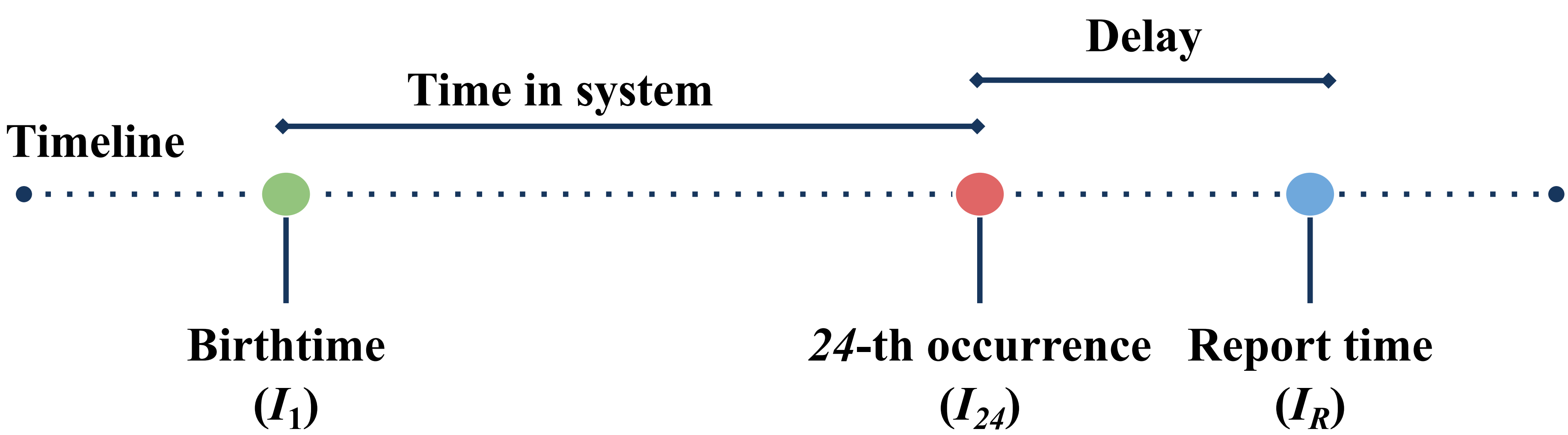


Database requirements:

- No false negatives -> Keep at much data as possible; use external memory
- Limited false positives
- Immediate response preferred
- Keep up with a fast stream (millions/sec or faster) -> write-optimization
  - Standing queries have a query per time step
  - Can delay reporting to keep up with stream

Time Stretch

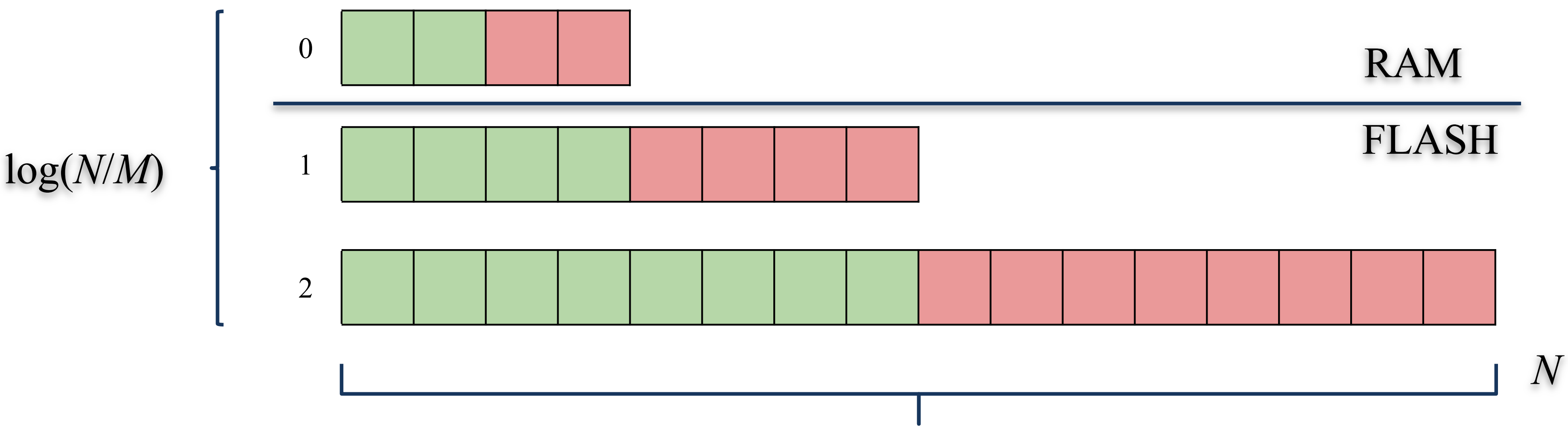
- Can’t afford multiple look ups per element
- Compromise: allow a little delay



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

Sandia National Laboratories

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 1 hour late and system runs 2x slower than when we gave no promises on delay

Time-Stretch Filter Analysis

Theorem. Given a stream of size *N*, the amortized cost of solving firehose with a time stretch $1 + \alpha$ is

$$O\left(\left(\frac{1+\alpha}{\alpha}\right)\frac{1}{B}\log\frac{N}{M}\right)$$

Optimal insert cost for EM & write-optimized dictionaries

Time-Stretch Filter Analysis

Theorem. Given a stream of size *N*, the amortized cost of solving firehose with a time stretch $1 + \alpha$ is

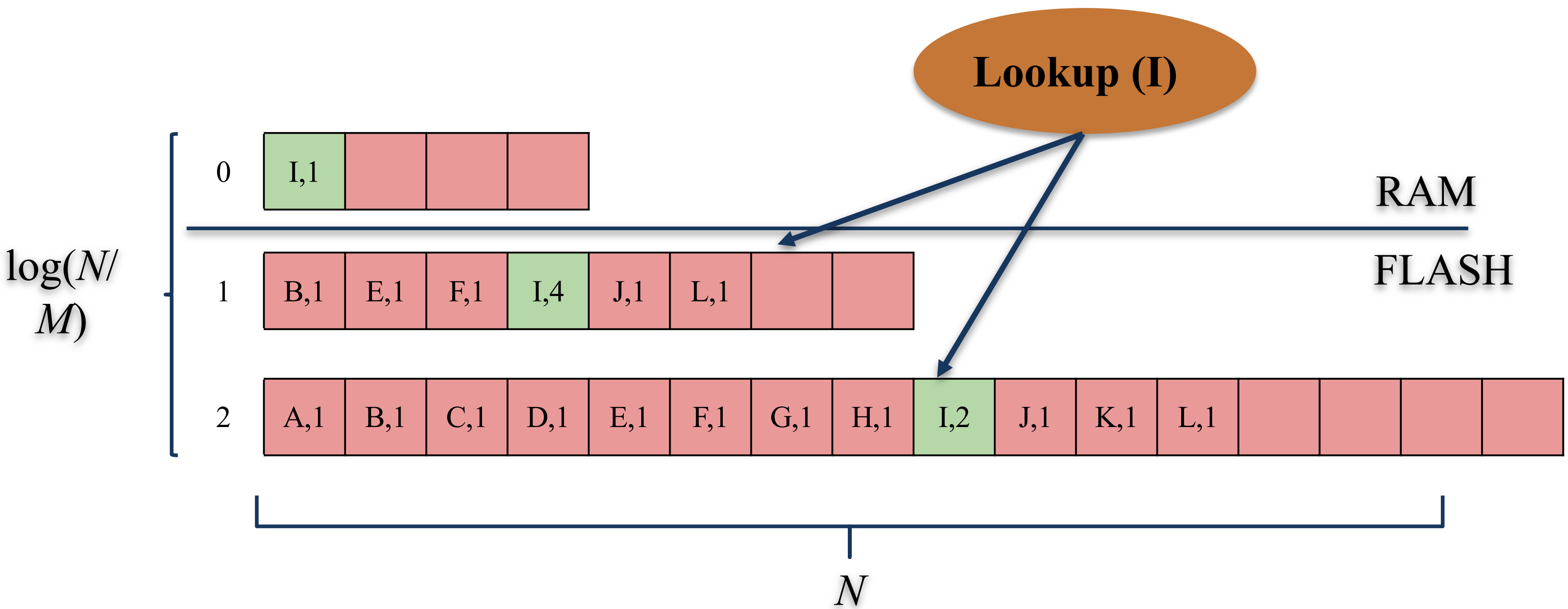
$$O\left(\left(\frac{1+\alpha}{\alpha}\right)\frac{1}{B}\log\frac{N}{M}\right)$$

Factor lost because we only flush **a fraction of each level**;
Constant loss for constant α

Almost-online reporting with no extra query cost!

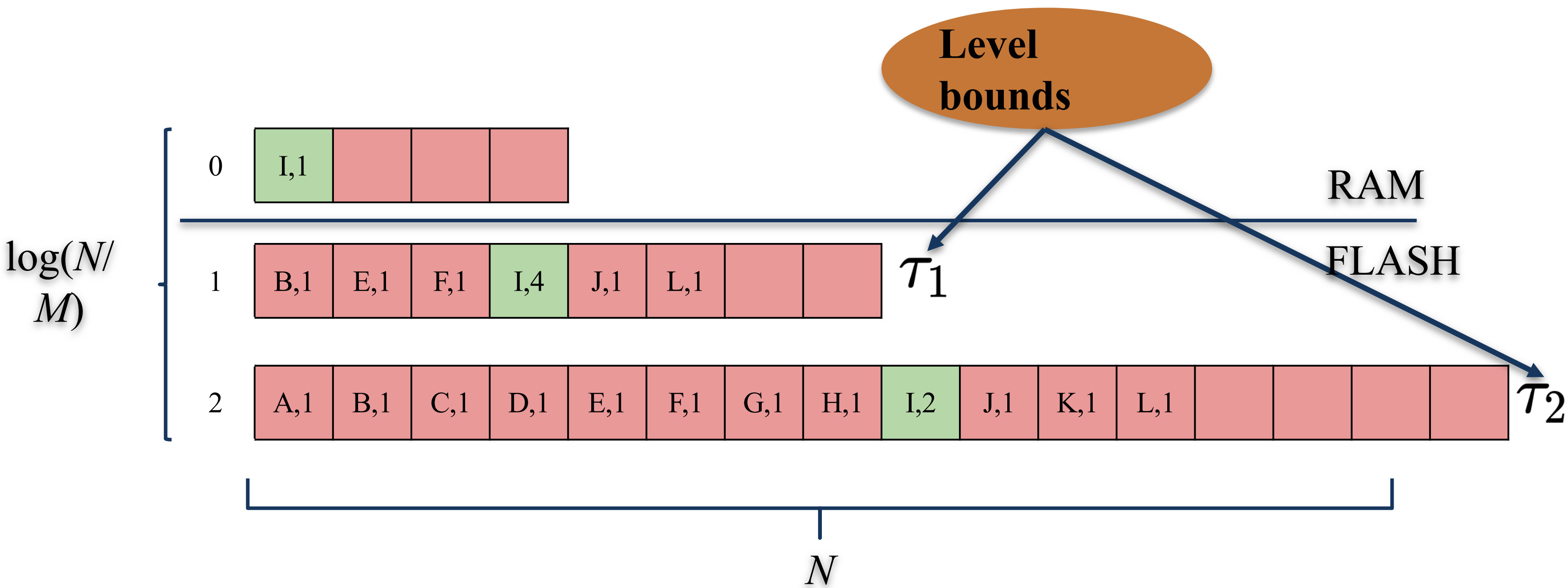
Sandia National Laboratories

How to do immediate reporting



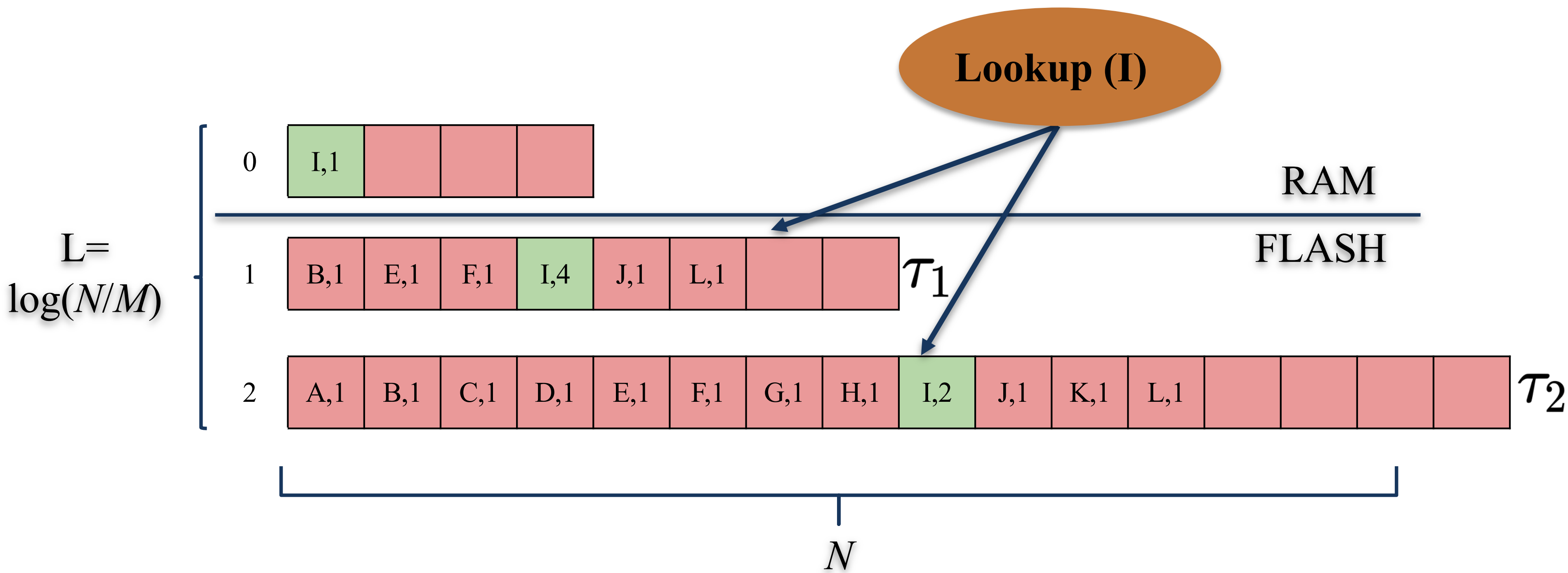
- In a cascade filter, we would need to perform multiple I/Os for every new item arriving in the RAM QF.

Level Thresholds



- At most τ_i counts of a key can be stored at level i. Higher closer to RAM.
- Shuffle merge: combine total count for a key on all visible levels, report if appropriate, otherwise push as low as possible respecting level thresholds.

Popcorn filter: immediate reporting

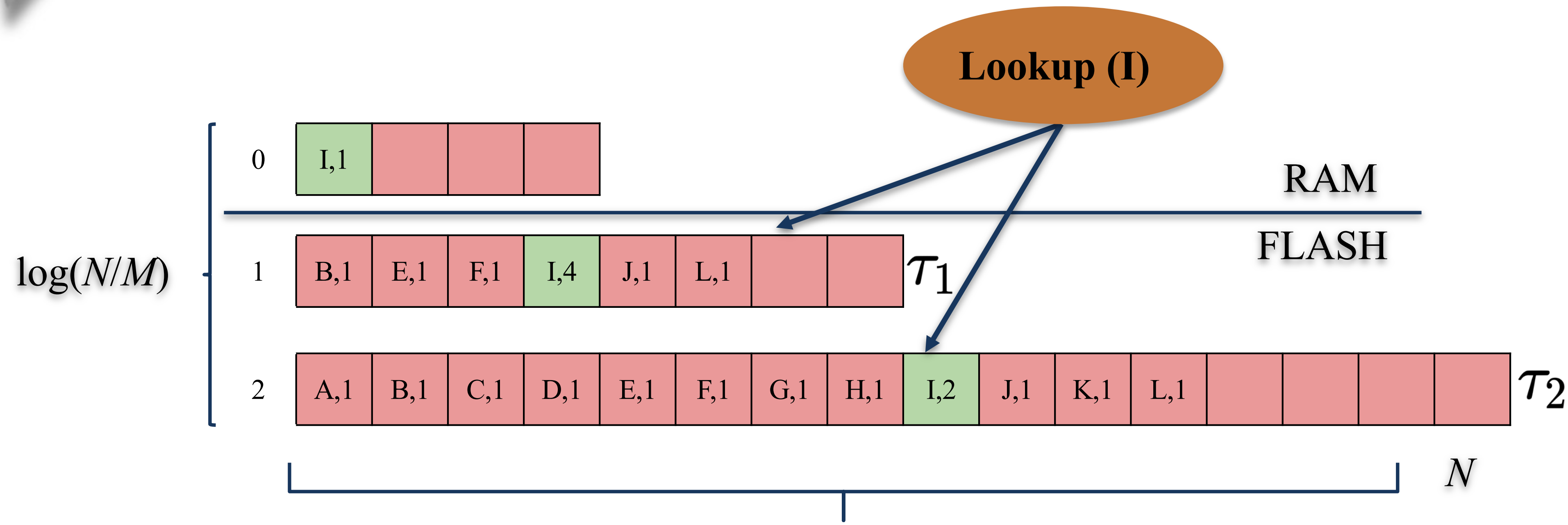


- Avoid unnecessary I/Os if we can **upper bound the total instances on disk**

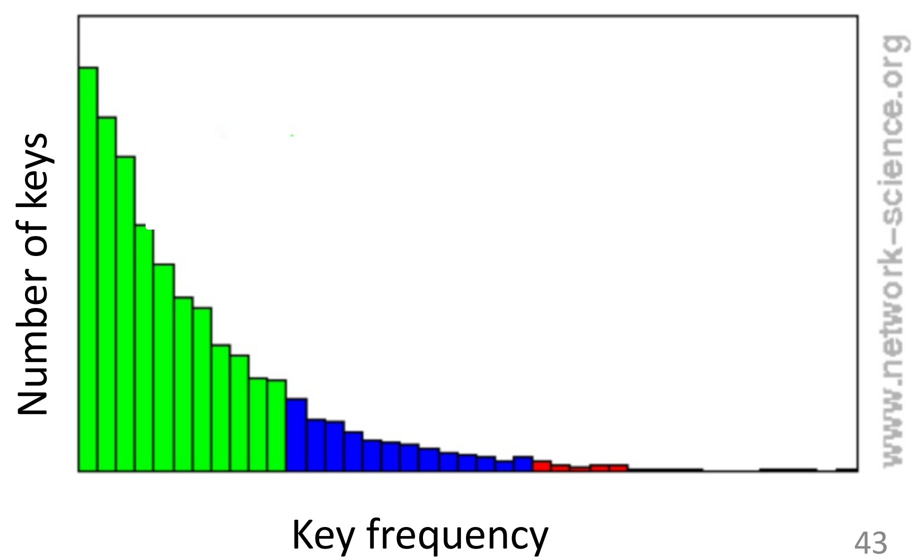
Lookup if $\text{RamCount} = 24 - \sum_{i=1}^{L} \tau_i$

Sandia National Laboratories

Popcorn filter



- Immediate reporting works if keys have power-law distribution: probability key count is c is $Zc^{-\theta}$, where Z is a normalization constant



Popcorn filter: immediate reporting

The number of I/Os per stream element is

$$O\left(\left(\frac{1}{B} + \frac{1}{(\phi N - \gamma)^{\theta-1}}\right)\log\left(\frac{N}{M}\right)\right)$$

About 1/1000

< 1/100 for Firehose for θ=2.96 and N/M=25

When

$$\Theta > 2$$

$$\phi N > \gamma$$

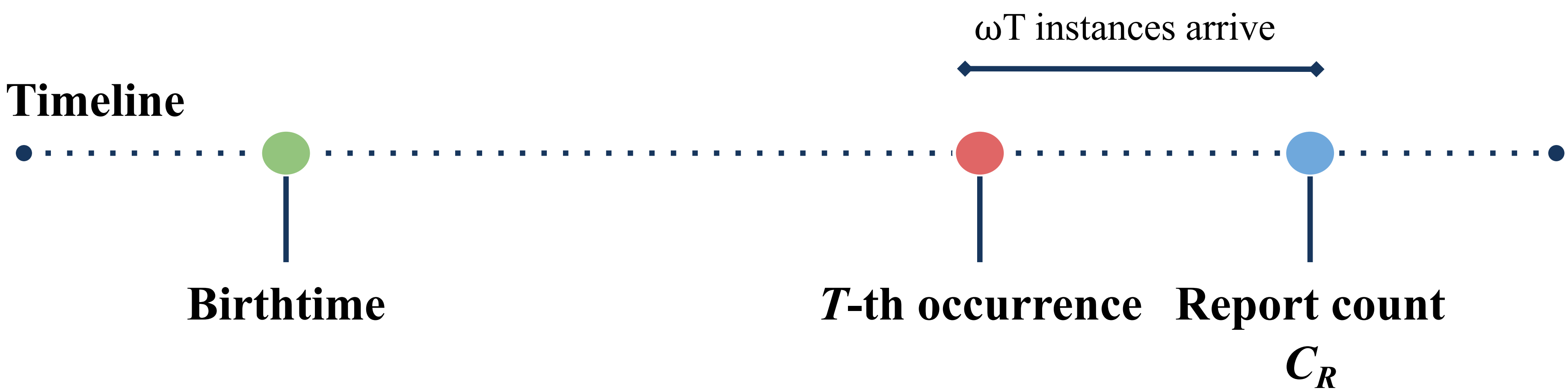
$$\gamma = \frac{e^{1/(\Theta-1)}}{e^{1/(\Theta-1)} - 1} \cdot \left(\frac{N}{M}\right)^{1/(\Theta-1)}$$

Note: for $\theta < 2.96$

$$\frac{e^{1/(\Theta-1)}}{e^{1/(\Theta-1)} - 1} < 2.5$$

Count stretch

A **count-stretch** of ω, we must report an element no later than when its count hits $(1+\omega)T$. In **immediate reporting** $\omega = 0$.



Popcorn filter: Count Stretch

- Do as with the popcorn filter, but report when count in RAM is ϕN
- Set level thresholds such that maximum on disk is ωϕN
- Amortized I/Os per stream element is:

$$O\left(\frac{1}{B}\log\left(\frac{N}{M}\right)\right)$$

When

$$\Theta > 2$$

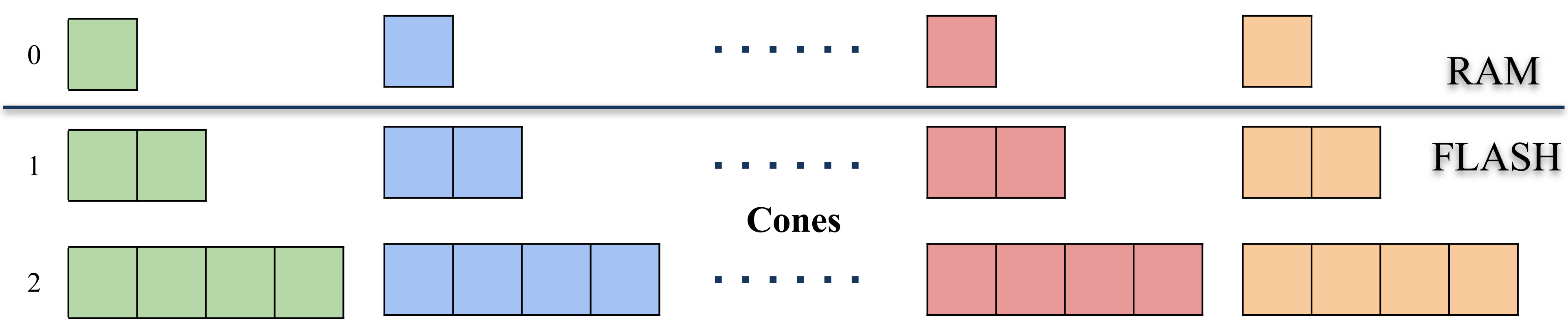
$$\phi N \cdot \omega > \frac{e^{1/(\Theta-1)}}{e^{1/(\Theta-1)}-1}$$

Note: for θ < 2.96

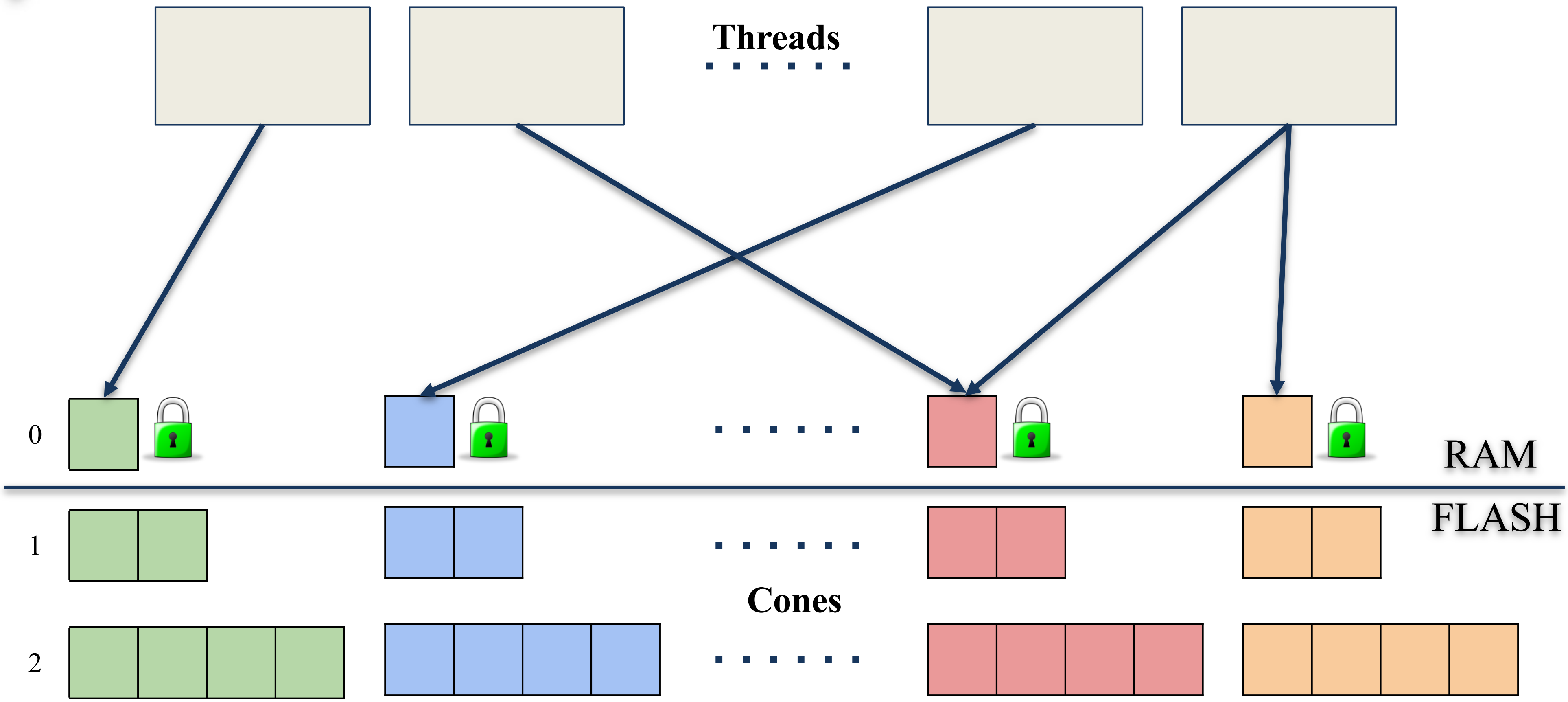
$$\frac{e^{1/(\Theta-1)}}{e^{1/(\Theta-1)}-1} < 2.5$$

Multithreading and Deamortization

- Data structures run well on average, but some operations take a long time
- Do a little work for each arriving element
  - Serial count-stretch guarantees still hold.
  - Time-stretch does not in general, does if input stream randomized

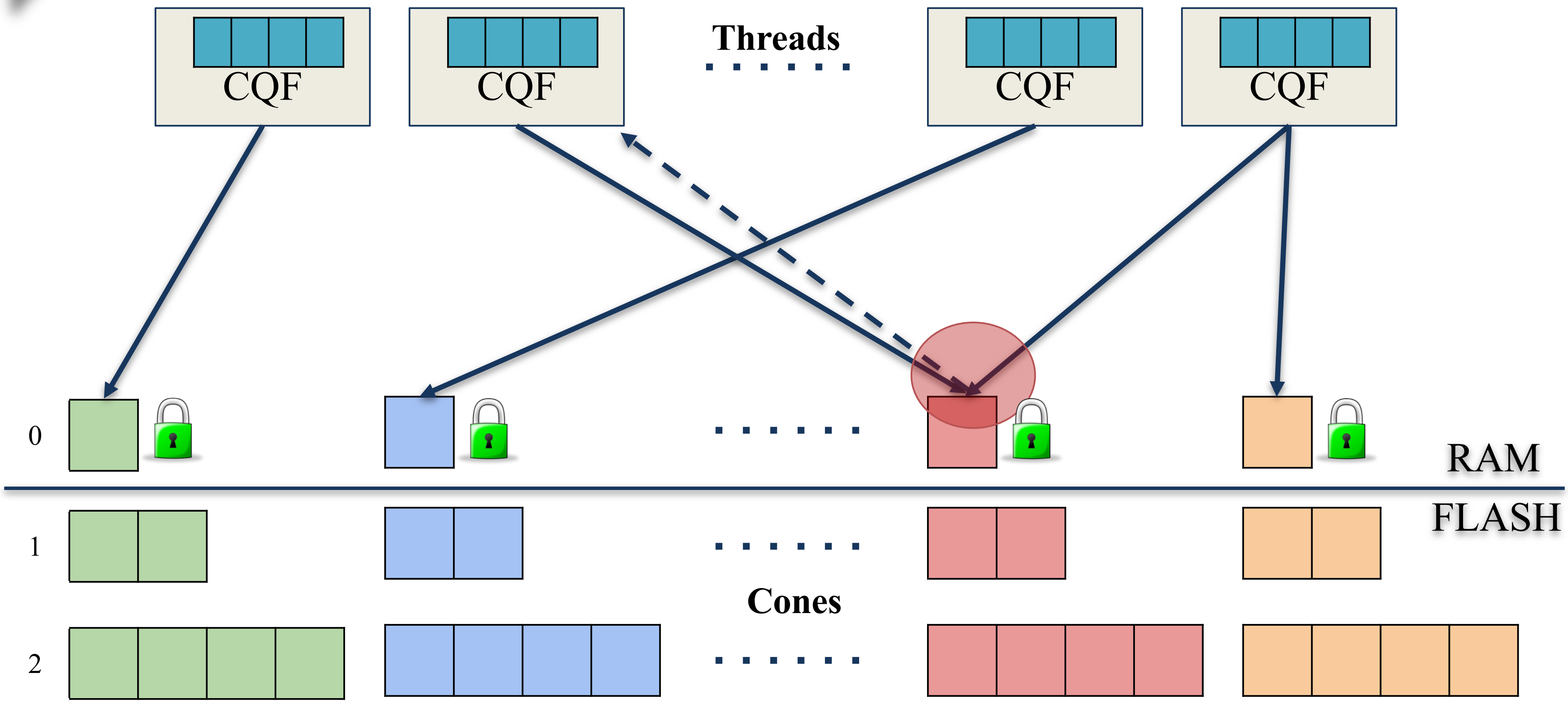


Multithreading/Deamortization



Each thread operates by first taking a lock at the cone and then performing the insert operation.

Multithreading/Deamortization



If there is contention, thread inserts the item in its local buffer (consolidating counts) and continues. When buffer full, waits for locks to clear buffer.

Multithreaded Count Stretch

- P = # of threads
- If I thread acquires local count for an element > T/P, waits to store that one element
- For multithreading, given ω and T > P, guarantees a count stretch of 2 + ω.

50

Experiments

Machines:

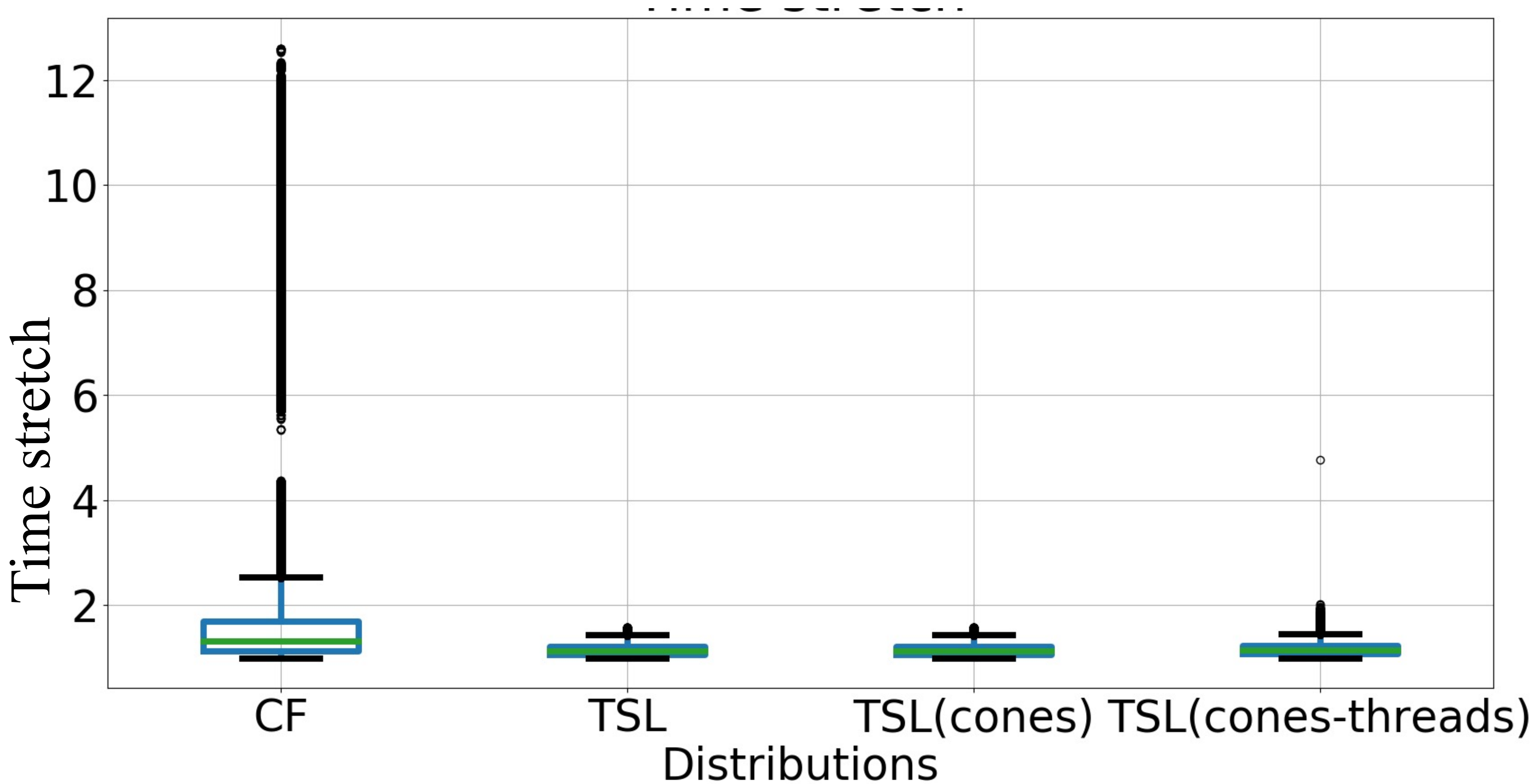
- Most experiments: Skylake CPU, 4 cores, 2.6 GHz, 32GB RAM, 1TB SSD
- Scalability experiments: Intel Xeon(R) CPU, 64 cores, 512 GB RAM, 1TB SSD

Input stream: mostly Firehose, power-law generator, active set of 1M key, drifting in larger key space. Read from file.

Stream size: 64M-512M for validation experiments (needs offline analysis; artificially reduce RAM); 4B for scalability experiments

Baseline comparison: Cascade filter

Time stretch

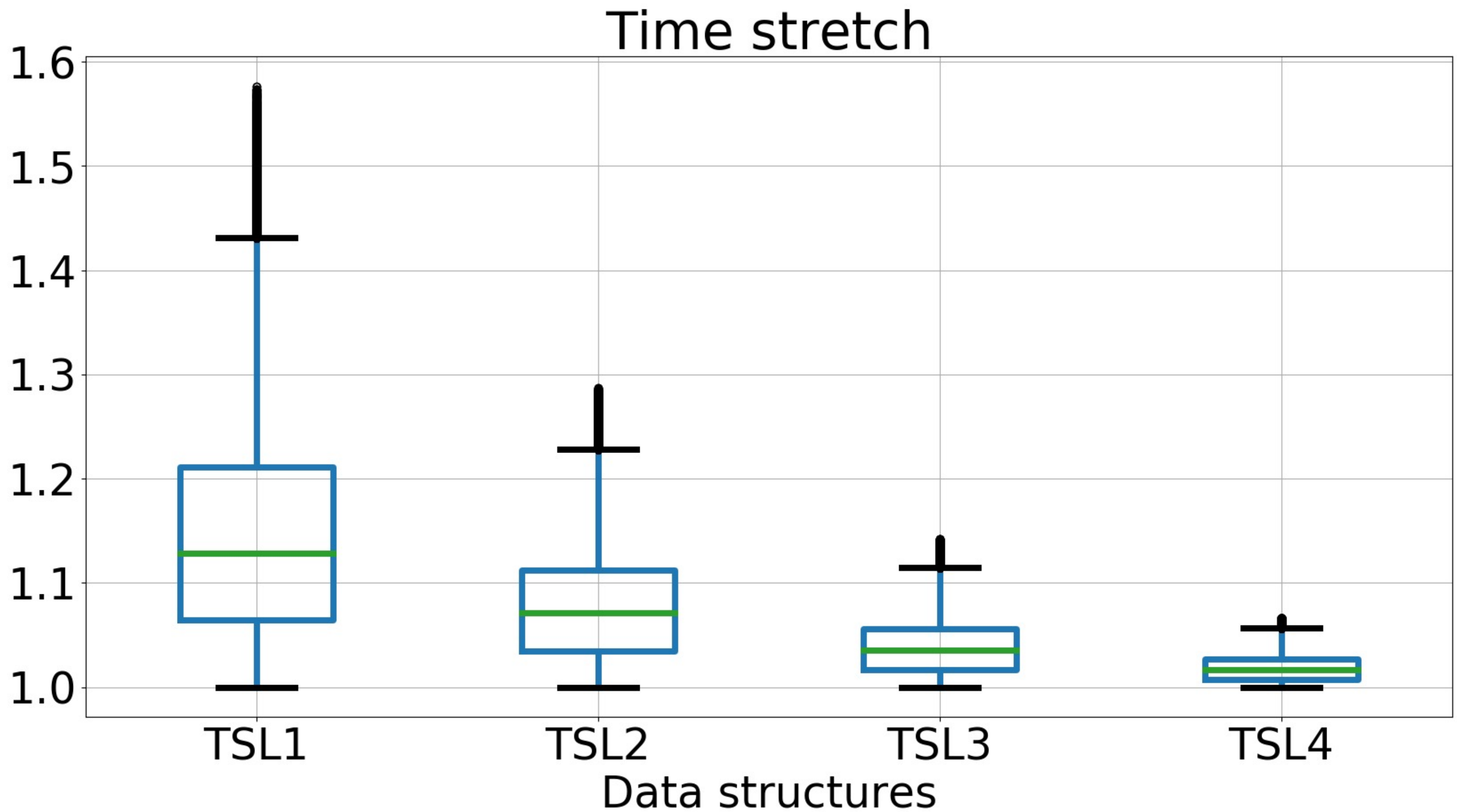


Deamortization and multithreading had negligible effect on empirical time stretch

RAM level: 8388608 slots, levels: 4, growth factor: 4, cones: 8, threads: 8, number of observations: 512M. (I think $\alpha = 1$)

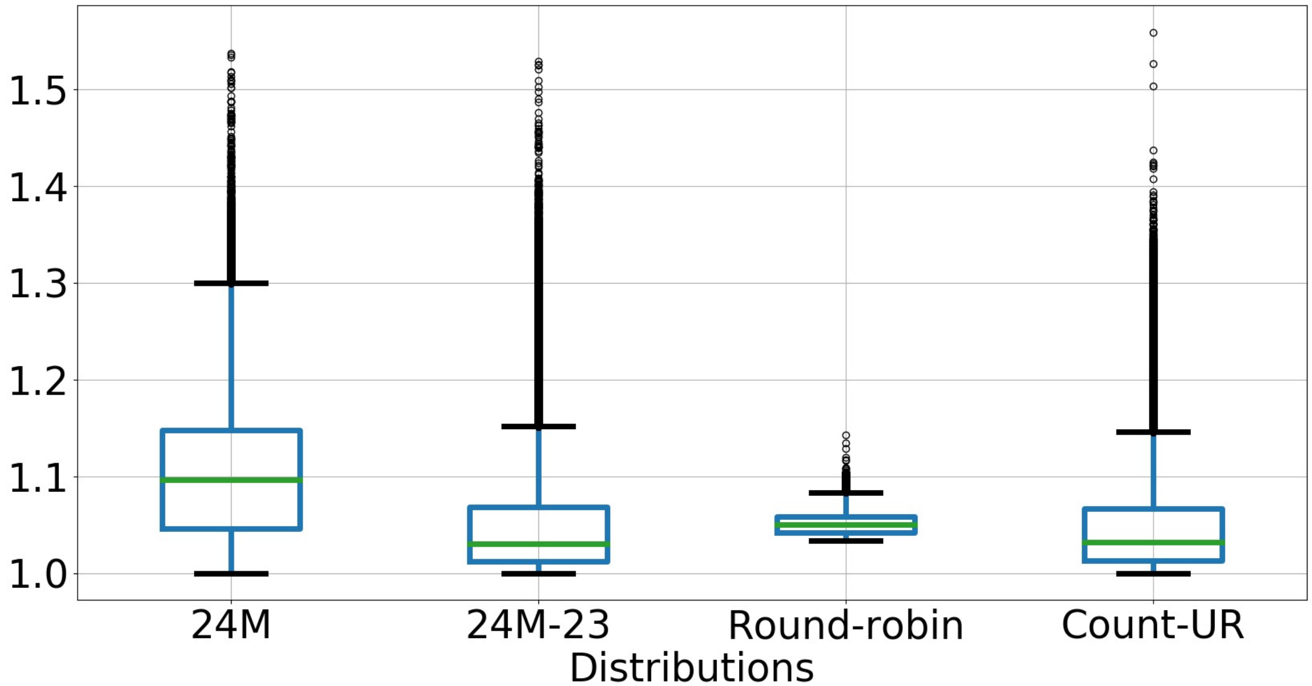
Sandia National Laboratories

Time stretch



Values of α left to right: $1, 0.33, 0.14, 0.06$.

Time stretch - robustness



Robustness to key-count distribution:

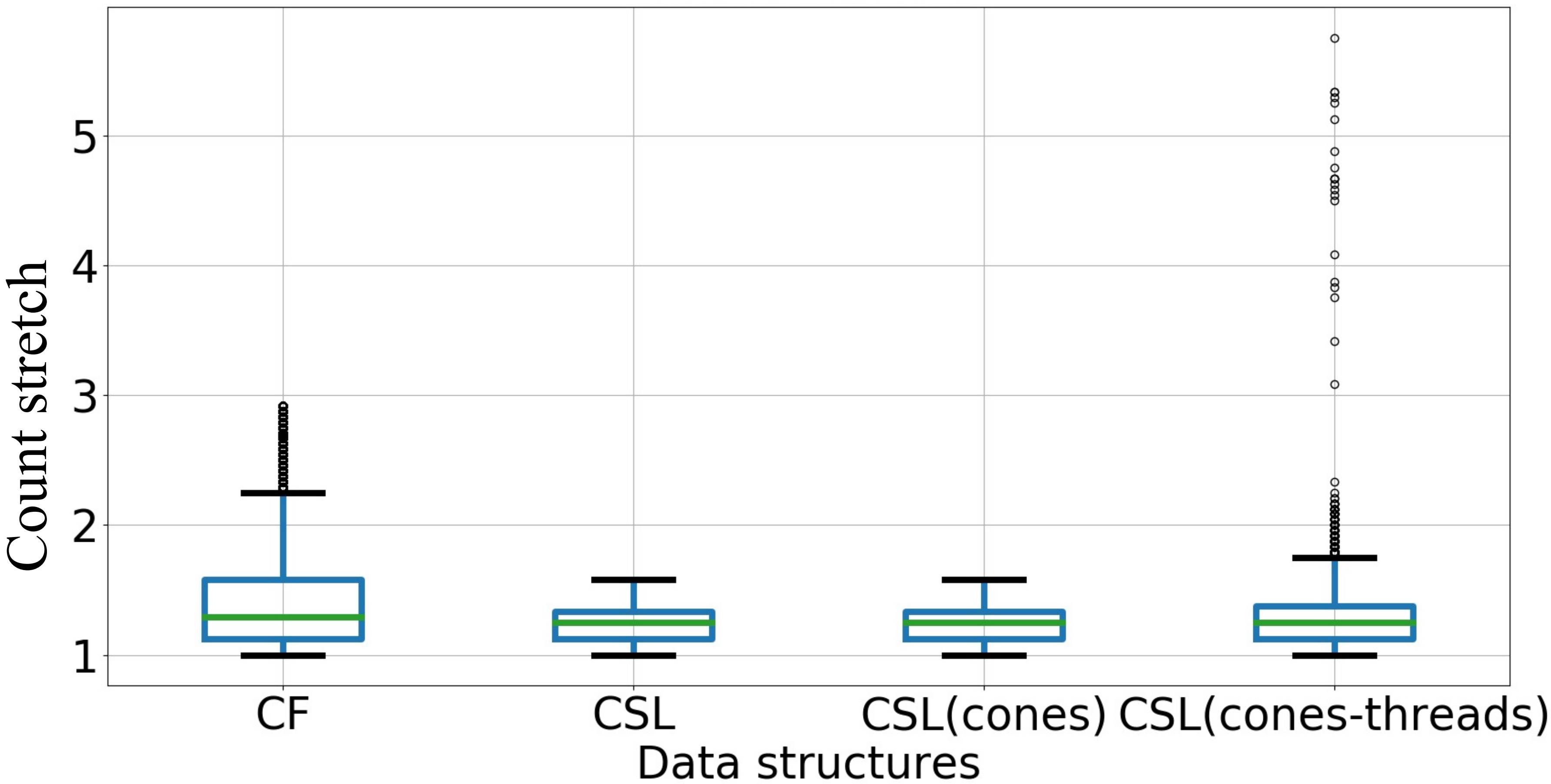
24M: RAM Size (M) keys 24-50 times, rest u.a.r from large universe

24M-23: M keys appear 24 times, rest appear 23

Round-robin: M keys in a round robin, all > 24

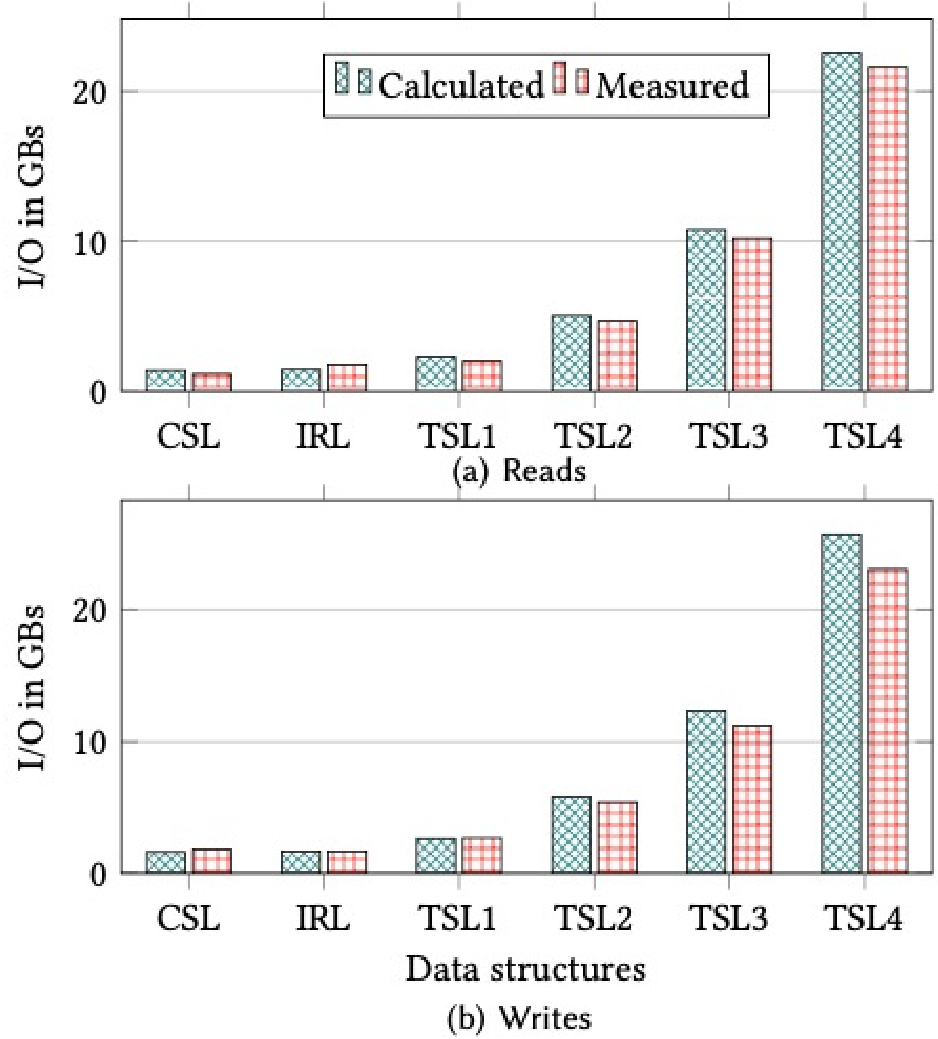
Count-UR: key counts all u.a.r 1 to 25

Count stretch



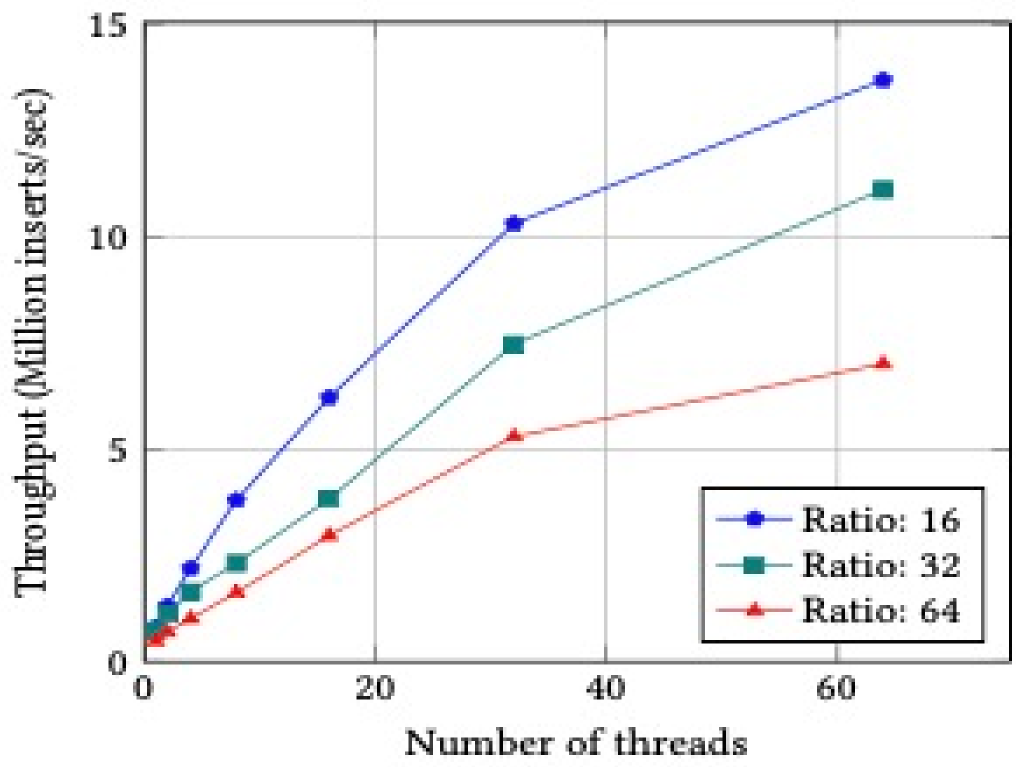
- Deamortization and multithreading had negligible effect on average count stretch. Multithreading had more variance.
- level thresholds: (2, 4, 8)

Total I/O



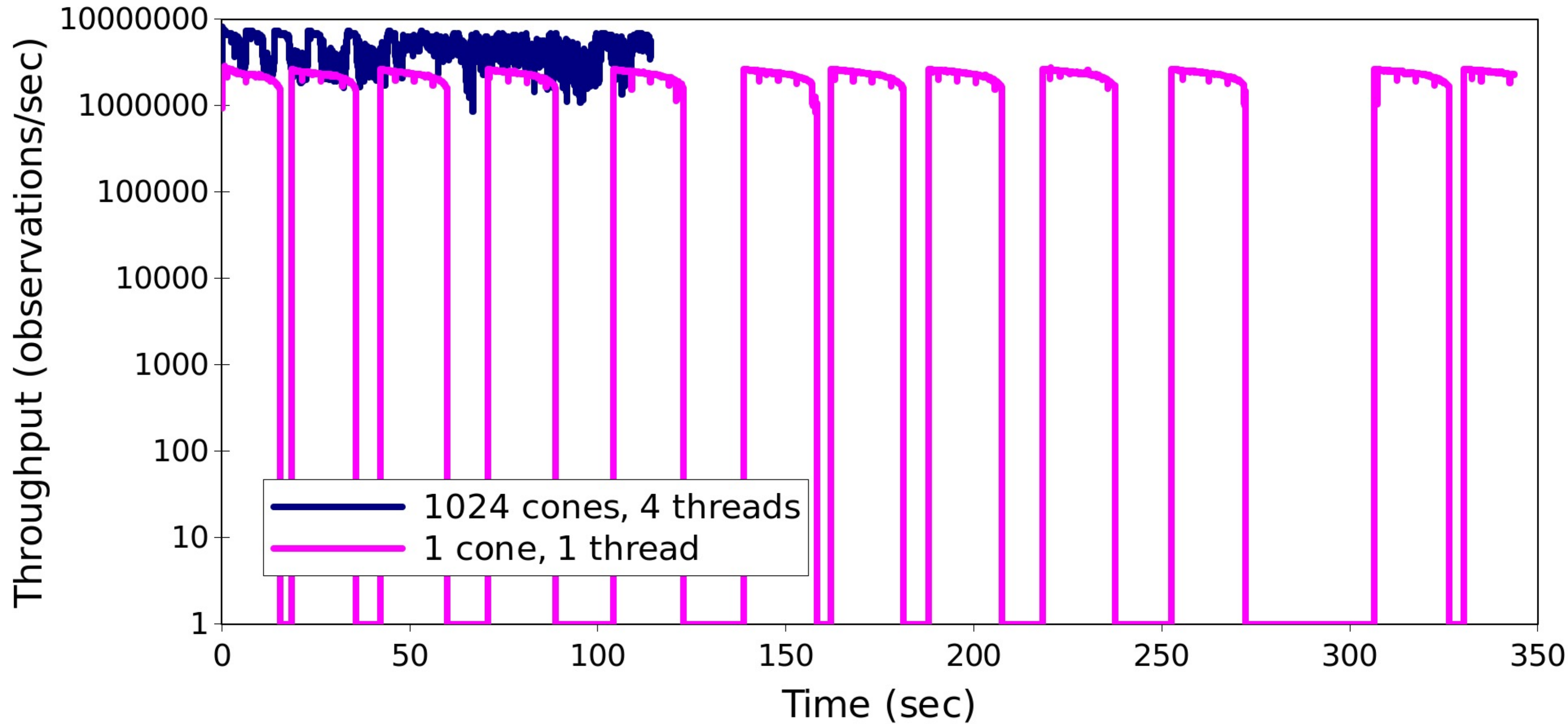
- Immediate reporting has about the same I/O as time-stretch with $\alpha = 1$
- RAM level: 4194304 slots in the CQF, levels: 3, growth factor: 4, number of observations: 64M

Scalability – count stretch



Reports all reportable keys. Stream size 4B.

Instantaneous Throughput



About 3x improvement of throughput with 4 threads, more steady

RAM level: 8388608 slots, levels: 4, growth factor: 4, cones: 8, threads: 8, number of observations: 512M. (I think $\alpha = 1$) – same as before

Final Thoughts

Missing detail: Separate data structure in RAM of reported keys

- Reporting a key twice is an error

Summary:

- 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, while keeping up with the stream, to get the best information
- This work bridges the gap between streaming and external memory

Next Steps:

- Intentional data expiration for infinite streams (preliminary results)

Prashant Pandey, Shikha Singh, Michael A Bender, Jonathan W Berry, Martín Farach-Colton, Rob Johnson, Thomas M Kroeger, and Cynthia A Phillips. 2020. Timely Reporting of Heavy Hitters using External Memory. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 1431–1446.

And journal version. Same authors (first two authors swapped), same title, ACM Transactions on Database Systems (TODS) 46.4 (2021): 1-35.

Extra Slides

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