This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2022-12476C

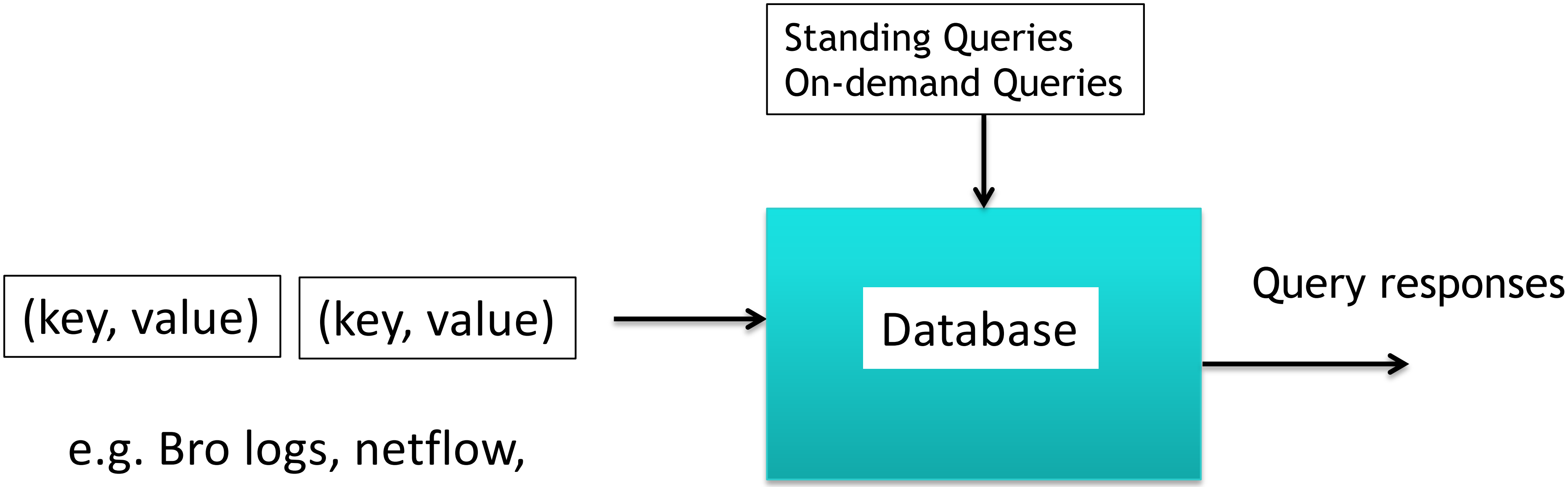
Write-Optimized Algorithms for Cybersecurity Stream Monitoring

Shikha Singh (Williams College)
Prashant Pandey (U. Utah)
Michael Bender (Stony Brook U)
Jonathan Berry (Sandia National Laboratories)
Daniel Delayo (Sandia National Laboratories)
Martin Farach-Colton (Rutgers U)
Rob Johnson (VMWare Research)
Thomas Kroeger (Sandia National Laboratories)
Cynthia Phillips, Sandia National Laboratories
David Tench (Rutgers U)
Eric Thomas (Sandia National Laboratories)

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



Standing Queries
On-demand Queries

(key, value) (key, value)

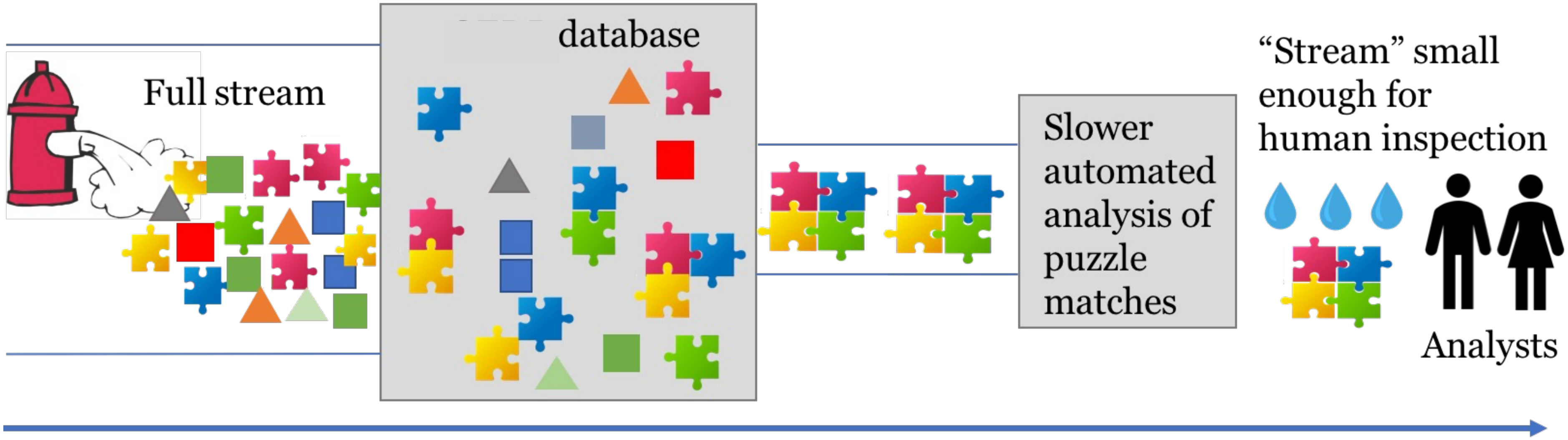
Database

Query responses

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



Full stream

database

Slower automated analysis of puzzle matches

"Stream" small enough for human inspection

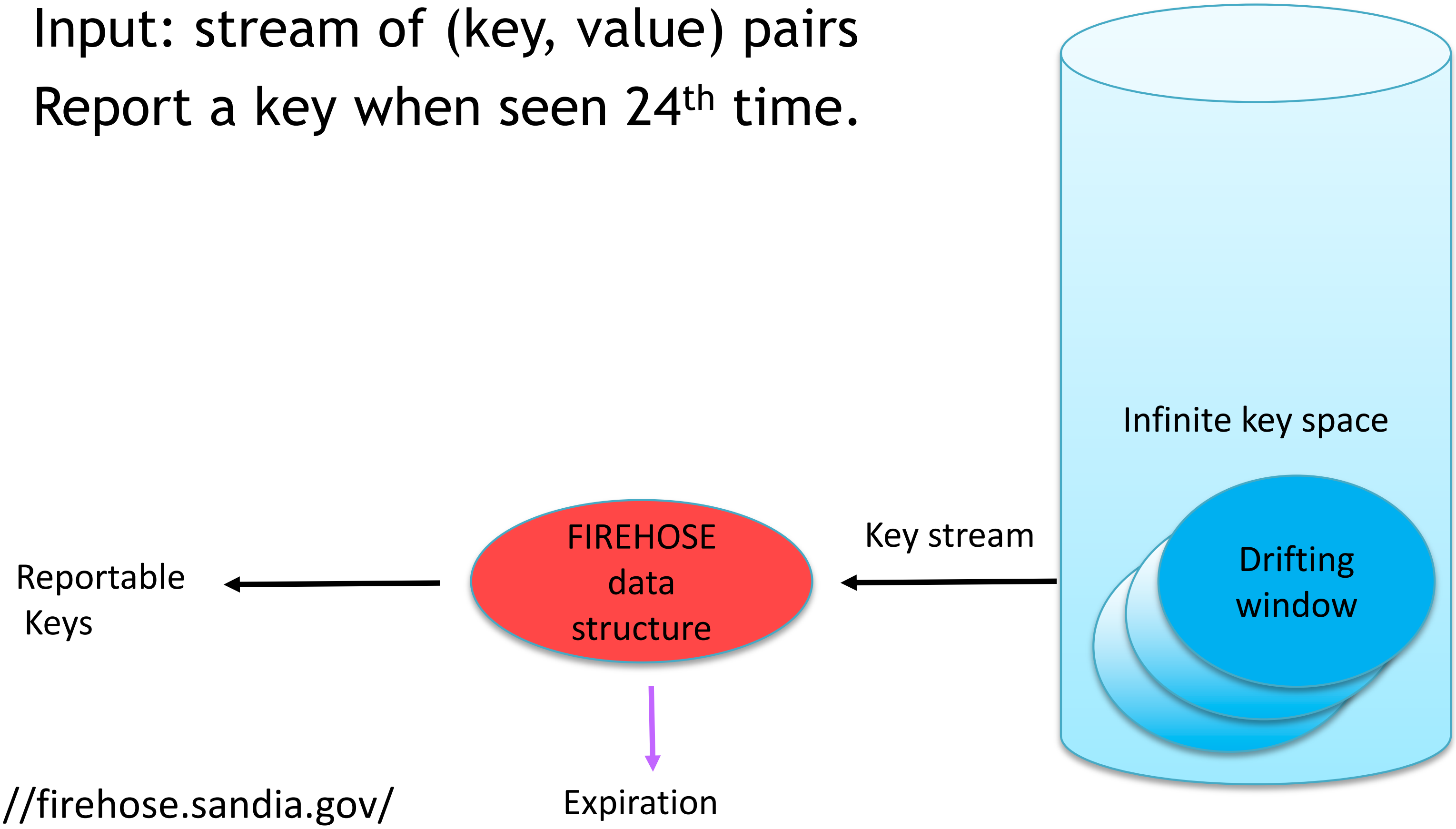
Analysts

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.



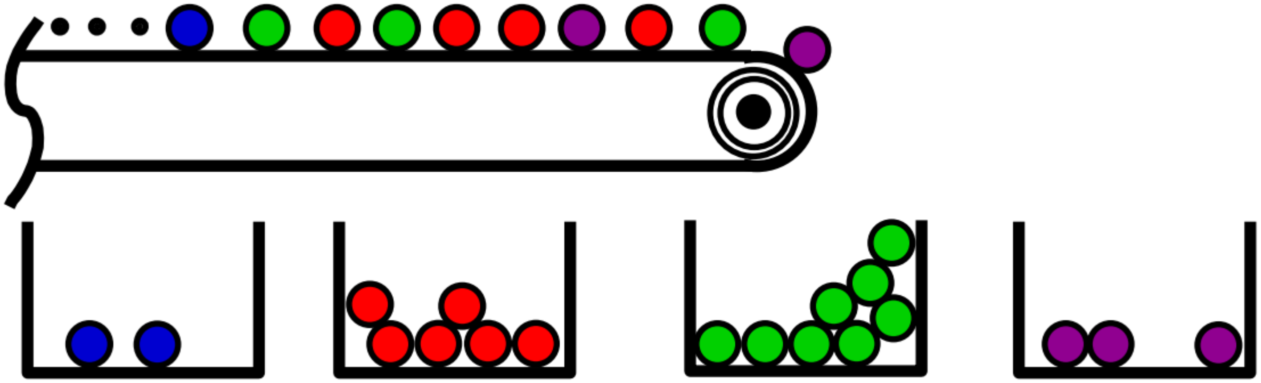
http://firehose.sandia.gov/

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
  – Requires $\Omega(1/\phi)$ space. For Firehose $\Omega(N)$.



Academic Streaming

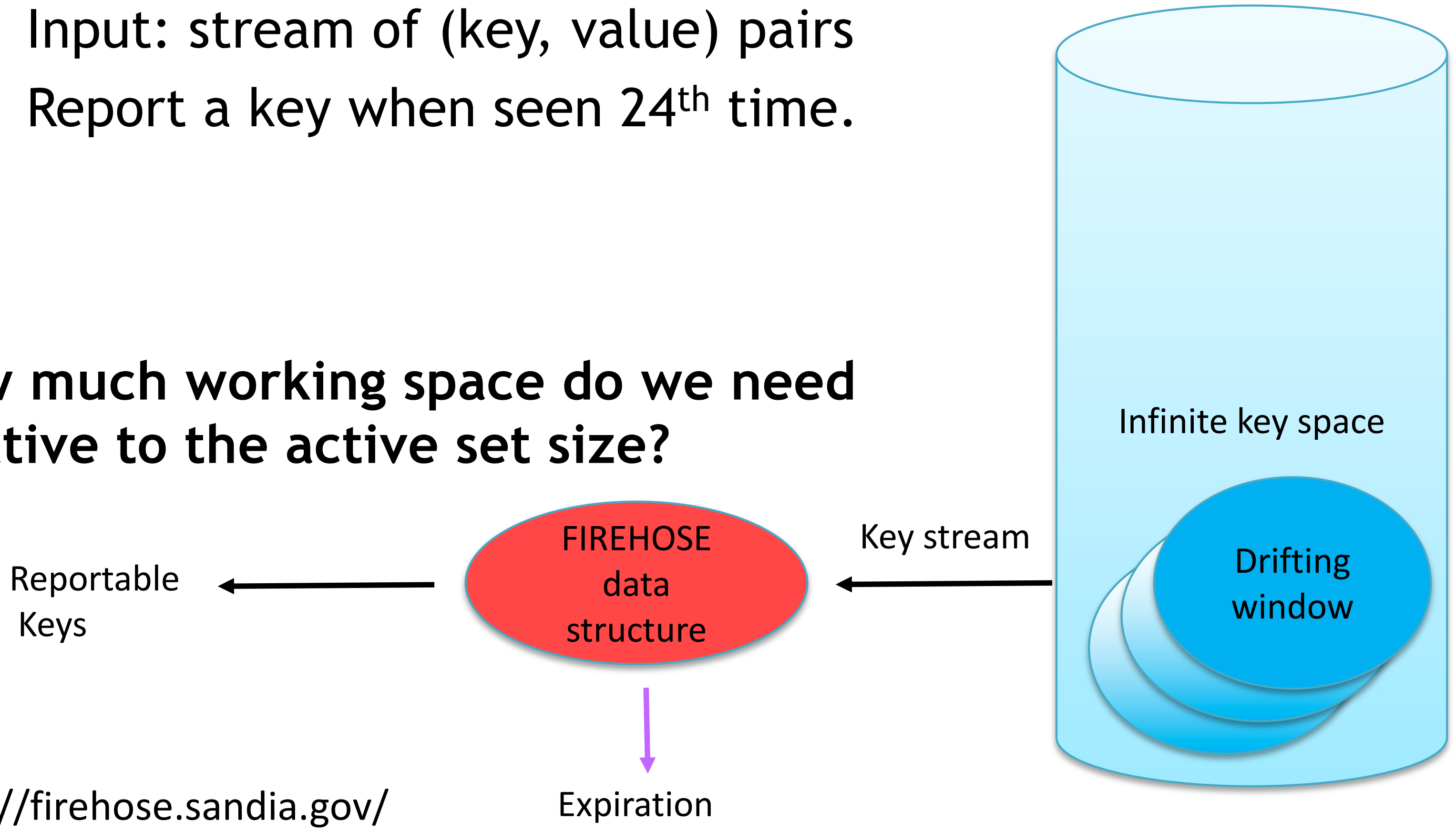
When there are large lower bounds (space required for an exact solution):

- Use more than fixed (constant) space, but as little as possible
- Use multiple passes
- Approximation (usually randomized)
  - Heavy-hitters, 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

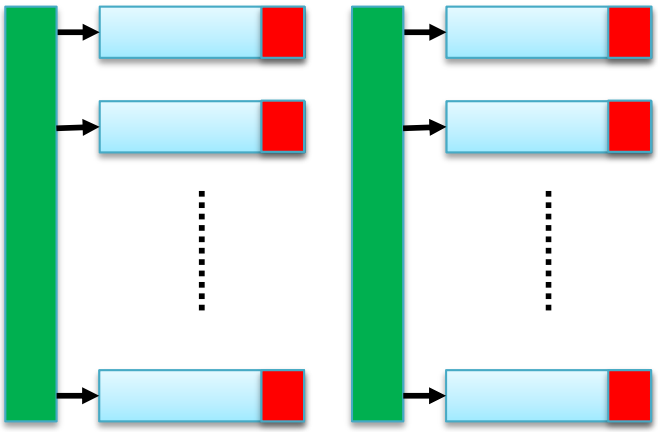
- Testing with benchmark reference implementation in Waterslide
  - 50M keys (varying counts)
  - Stable window
- Accuracy of cyber-analytics depends on keeping enough data
- Difficult to determine what to throw away
  - Most keys act the same at their start
- Keep as much data as we can!

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*

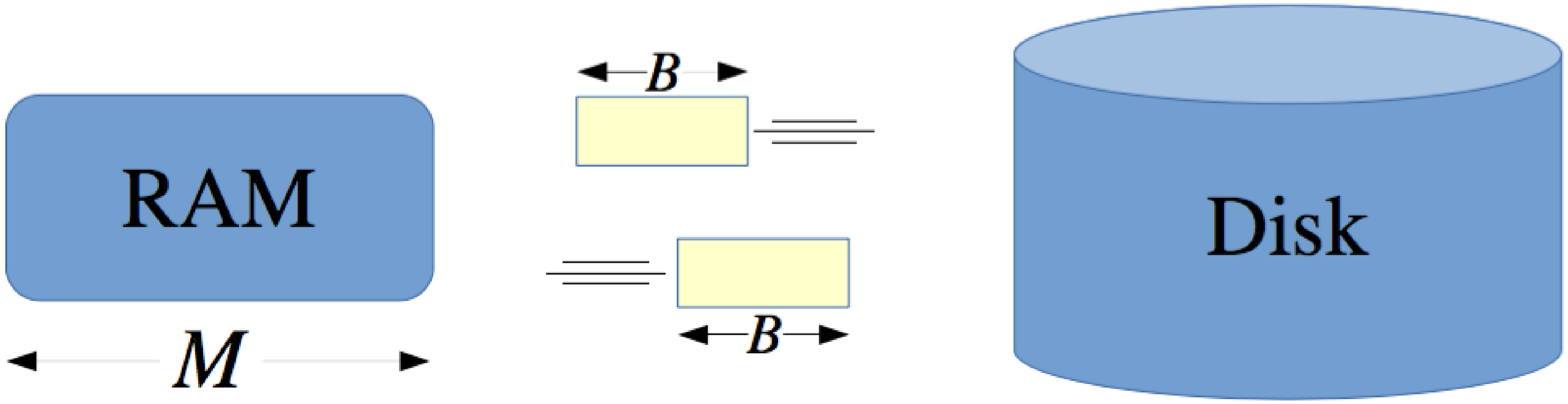
- **1/16 of all data is always subject to immediate expiration in steady state**

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

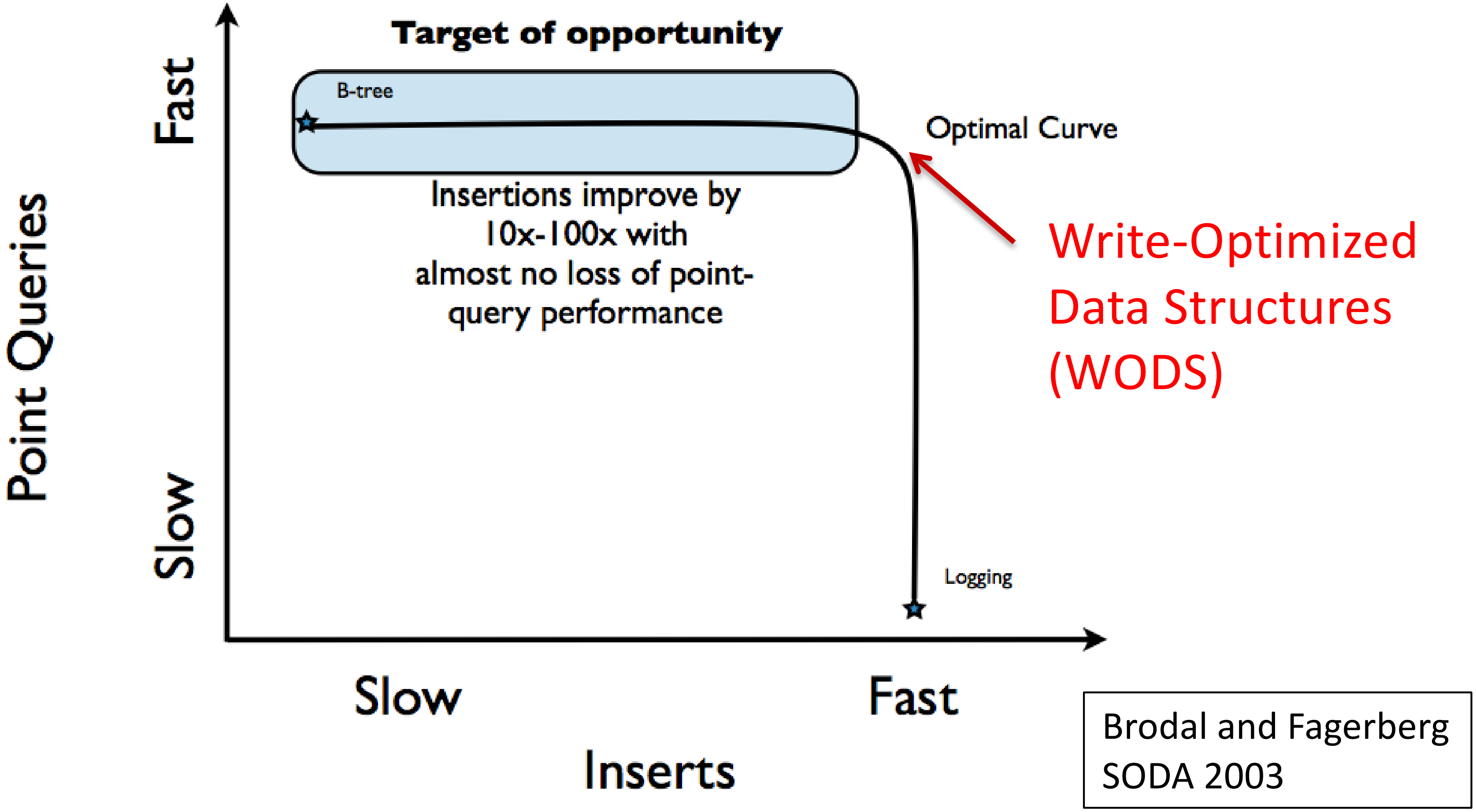
Even when window size is only 4x data structure size, most reportable data are lost before It is reported.

External Memory Model

- 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

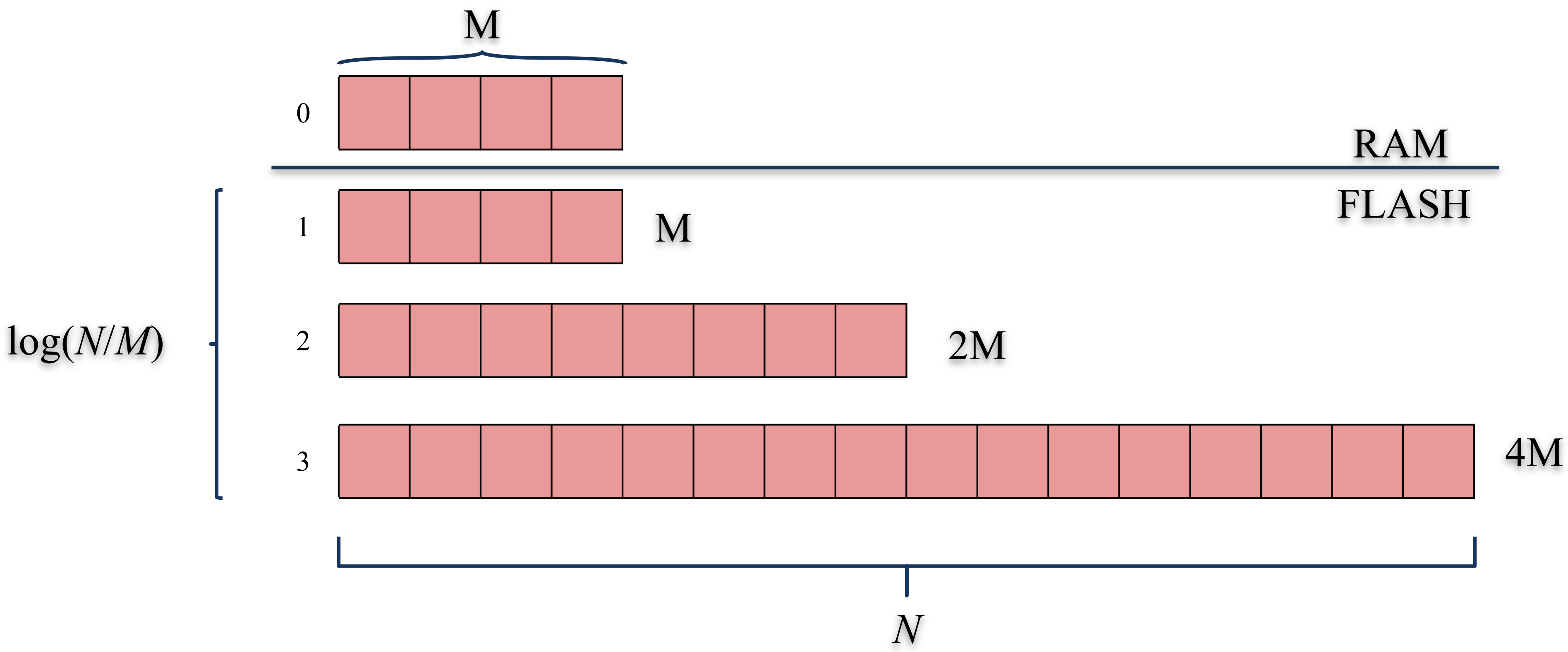


Brodal and Fagerberg
SODA 2003

- The basis for TokuDB

Write optimization: Cascade filter

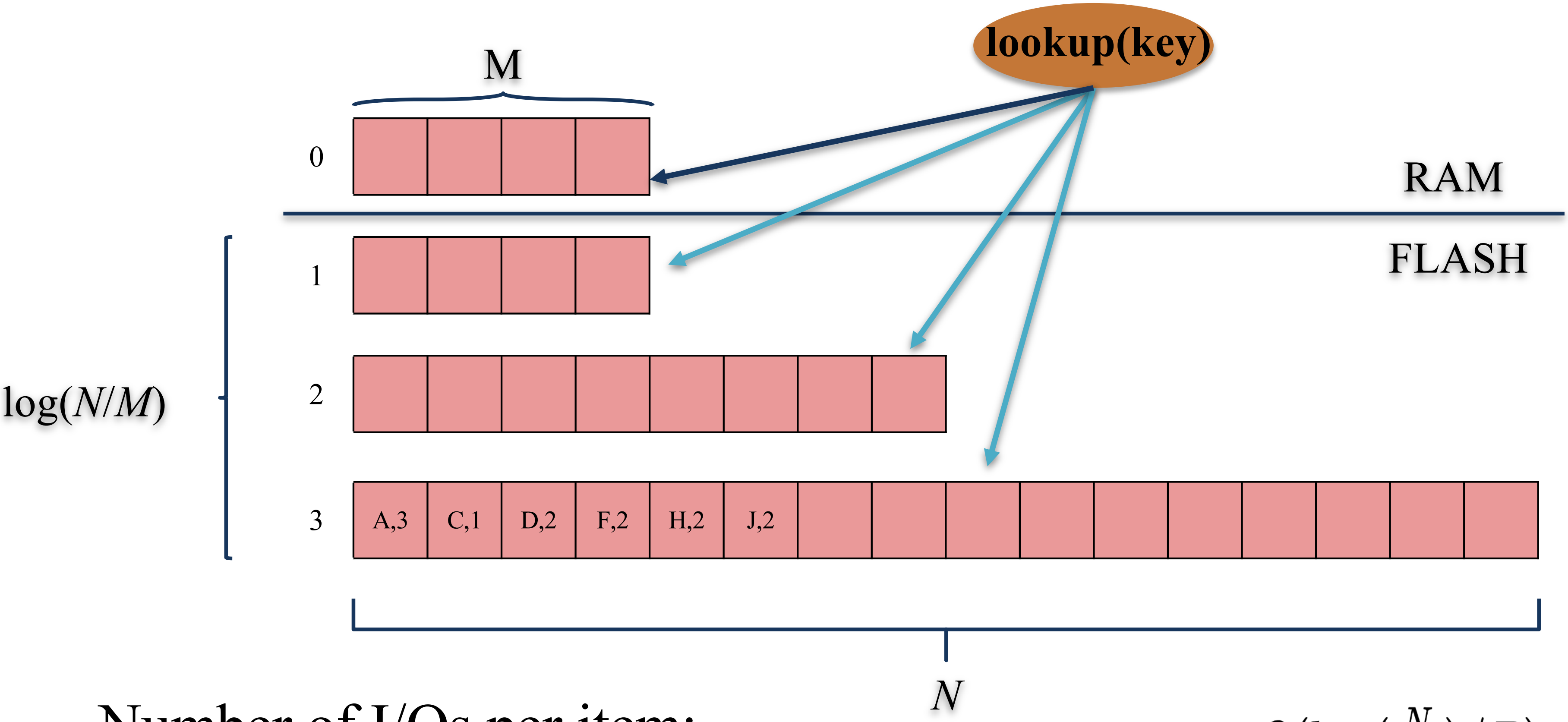
[Bender et al. 12, Pandey et al. 17]



- 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

Cascade filter Performance

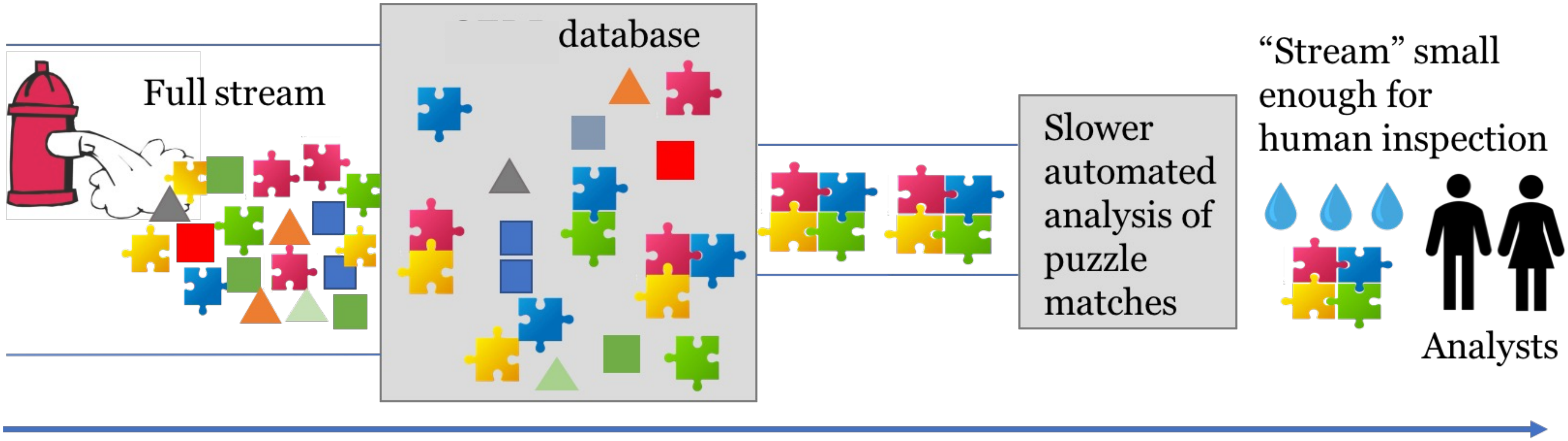


Number of I/Os per item:

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

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

Reminder: Standing Queries



Full stream

database

Slower automated analysis of puzzle matches

"Stream" small enough for human inspection

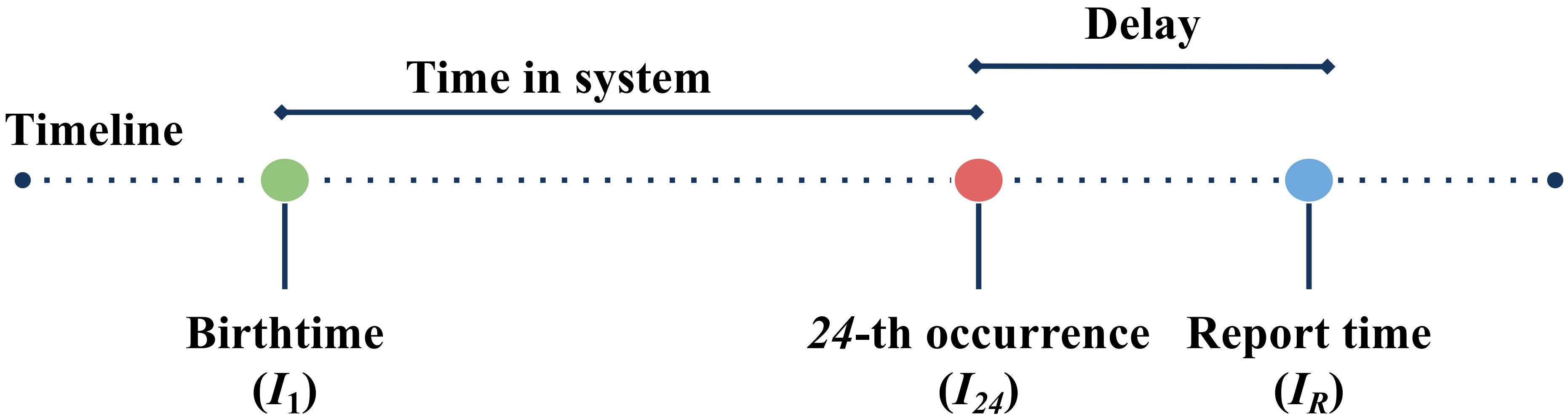
Analysts

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



Delay

Time in system

Timeline

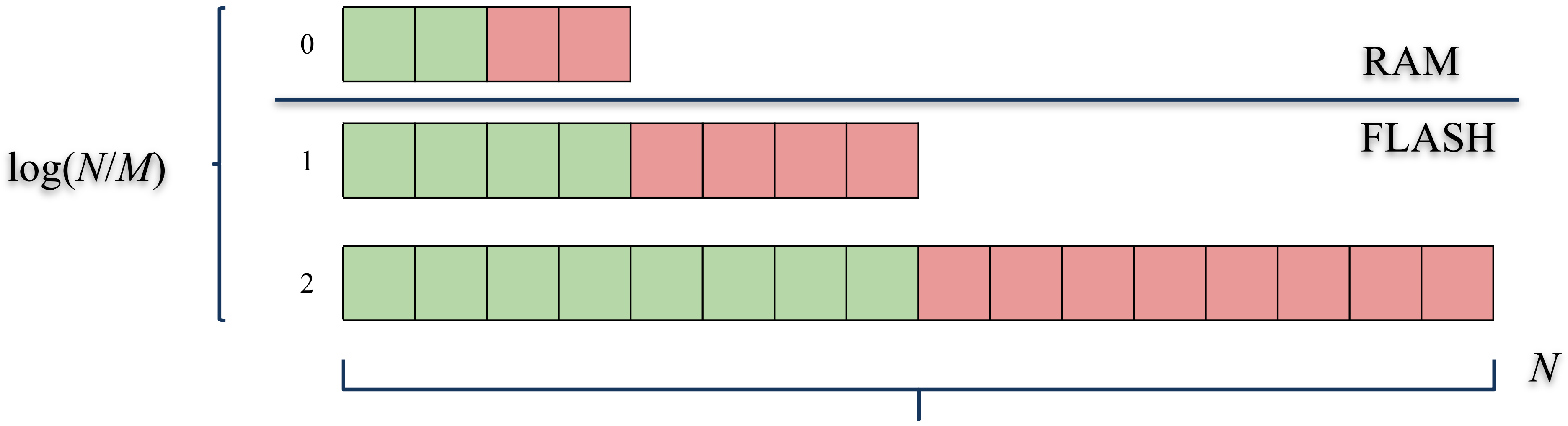
Birthtime (I_1)

24-th occurrence (I_{24})

Report time (I_R)

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

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 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 per-element 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

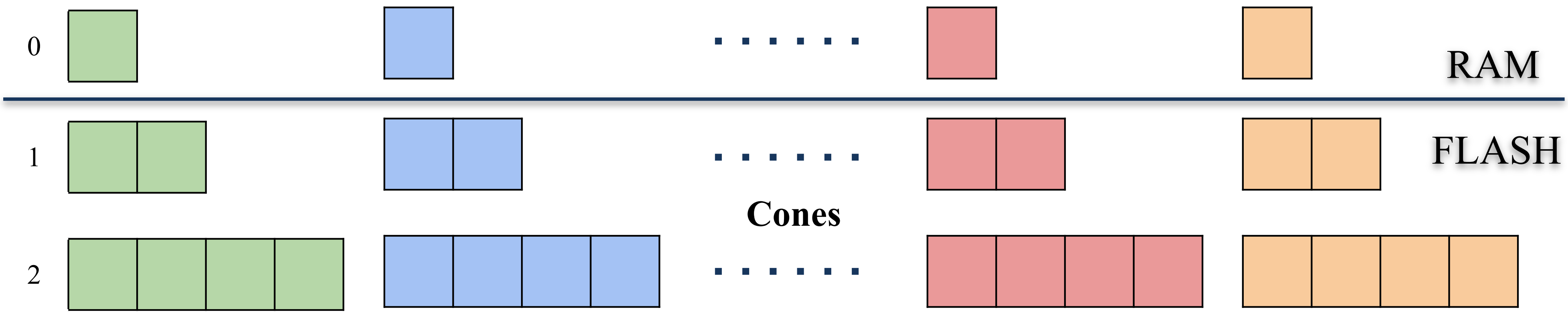
$$O\left(\left(\frac{1+\alpha}{\alpha}\right)\frac{1}{B}\log\frac{N}{M}\right)$$

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

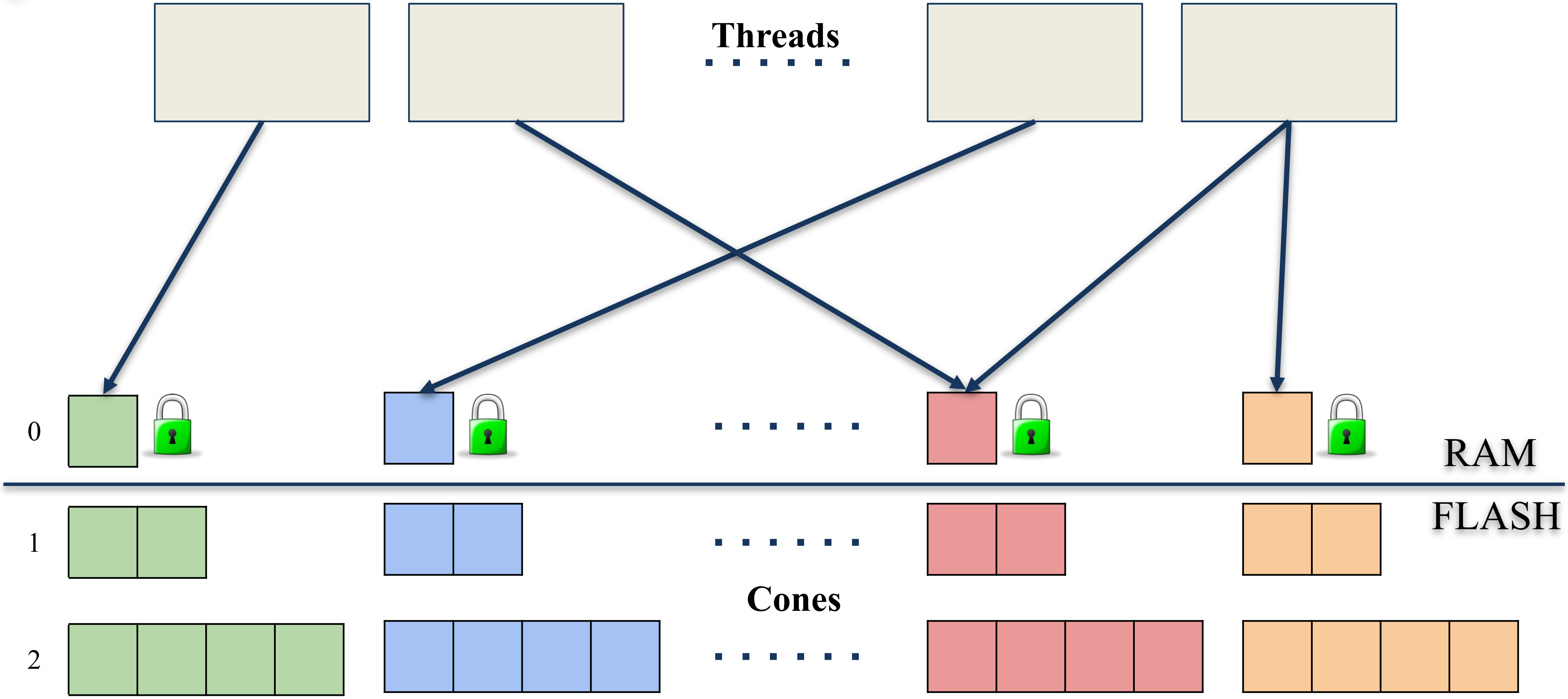
Almost-online reporting with no extra query cost!

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

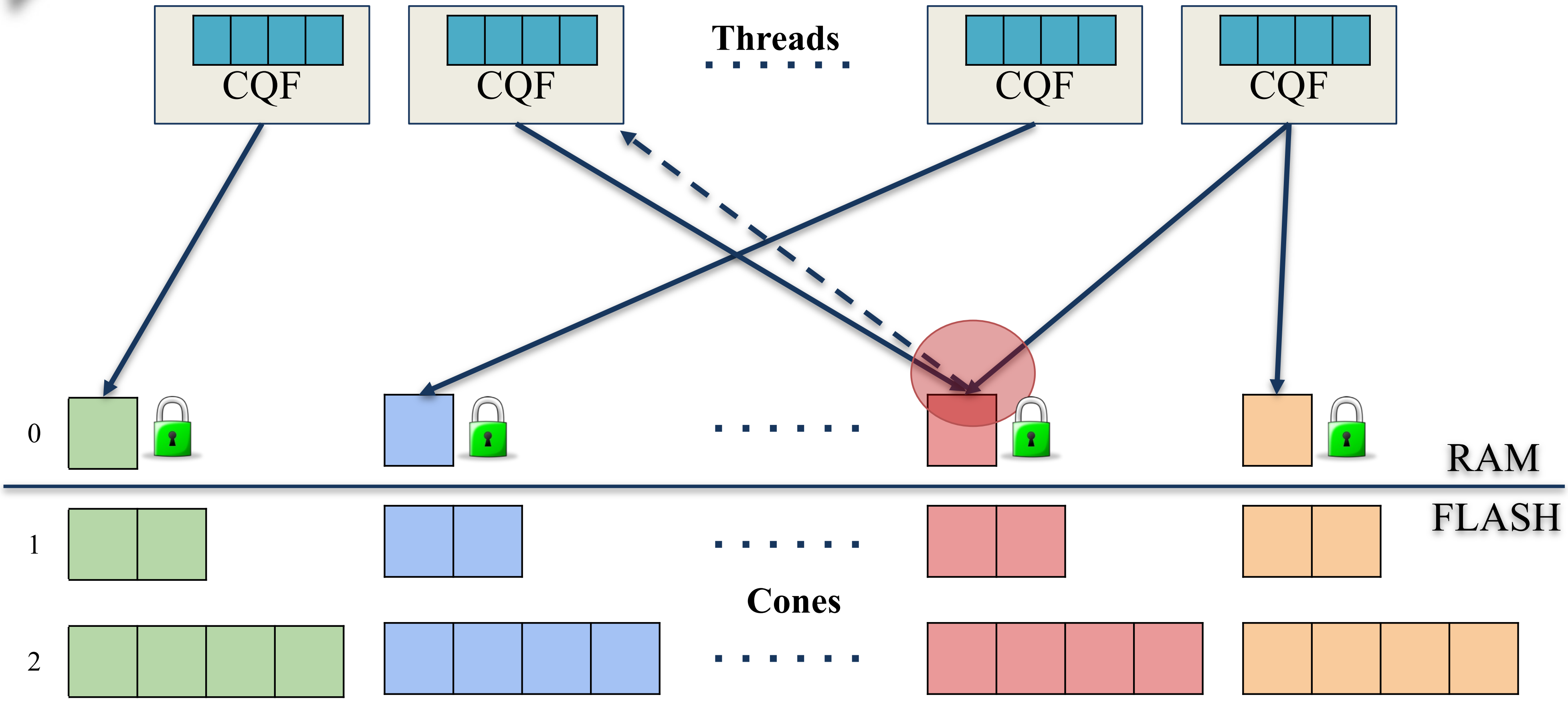


Multithreading/Deamortization



Each thread has a small chunk of stream elements. Takes a lock at the cone and then inserts.

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.

Experiments

Machines:

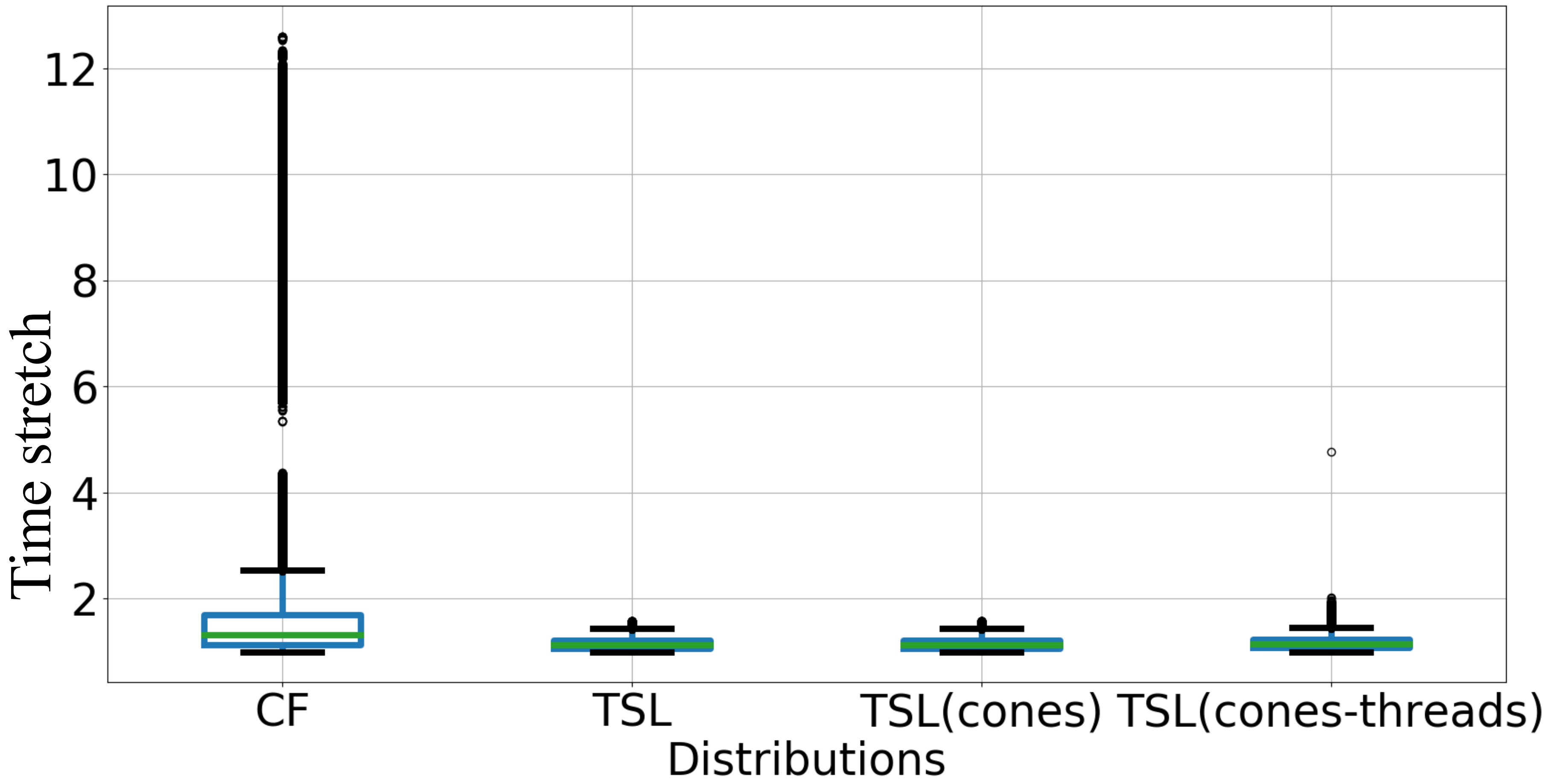
- Most experiments: Skylake CPU, 4 cores, 2.6 GHz, 32GB RAM, 1TB SSD
- Scalability experiments: Intel Xeon(R) CPU, 64 cores, 512 GB RAM, 1TB SSD

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

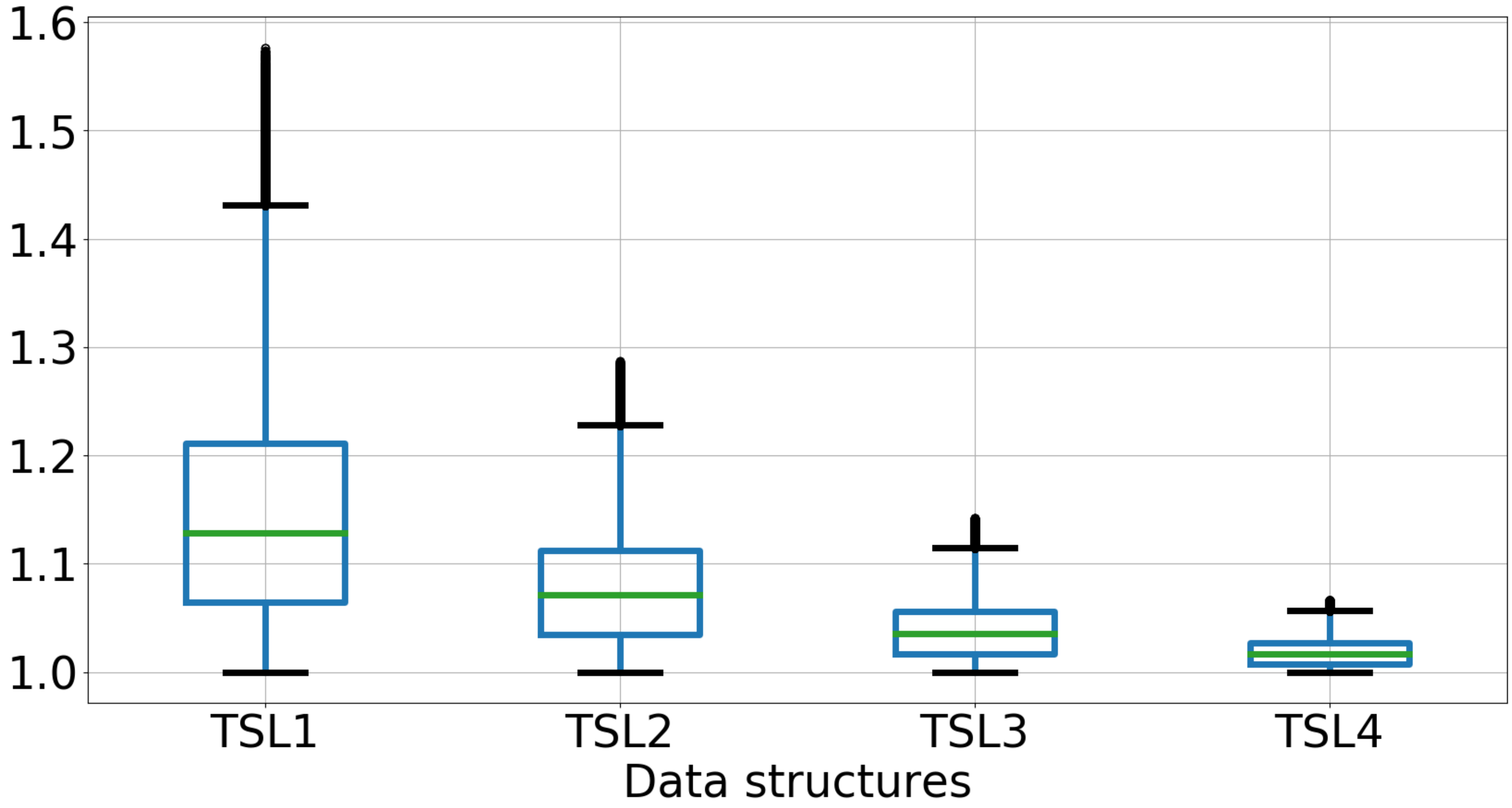
Time stretch



Deamortization and multithreading had negligible effect on empirical time stretch

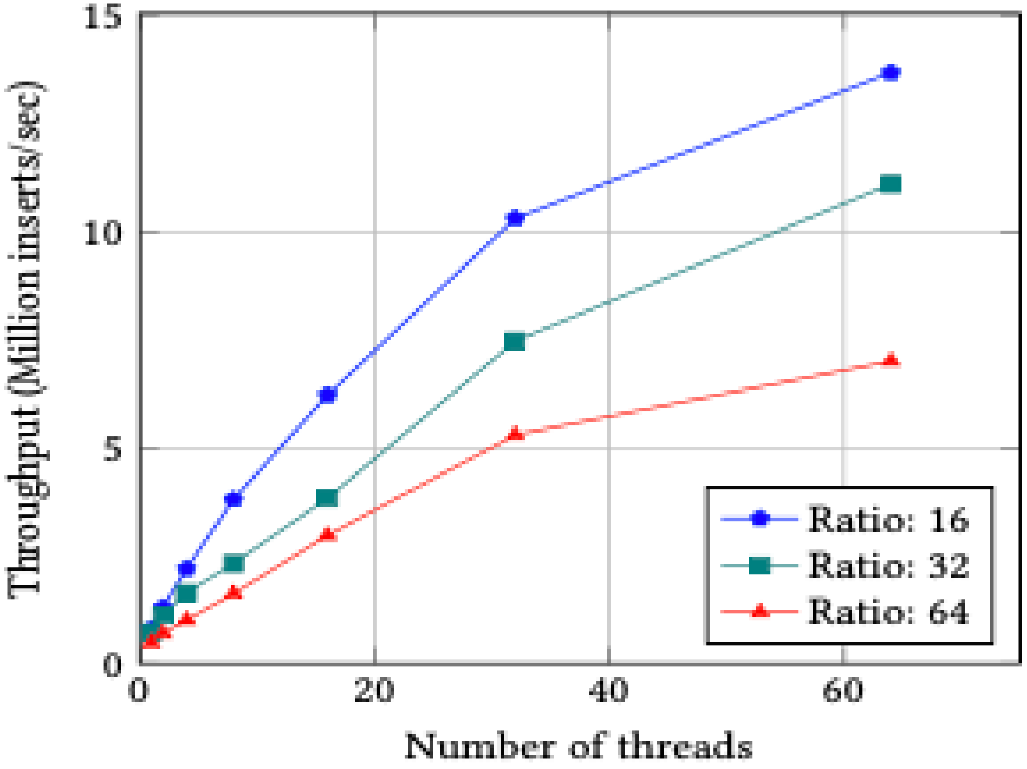
RAM level: 8388608 slots, levels: 4, growth factor: 4, cones: 8, threads: 8, number of observations: 512M. (I think α = 1)

Time stretch



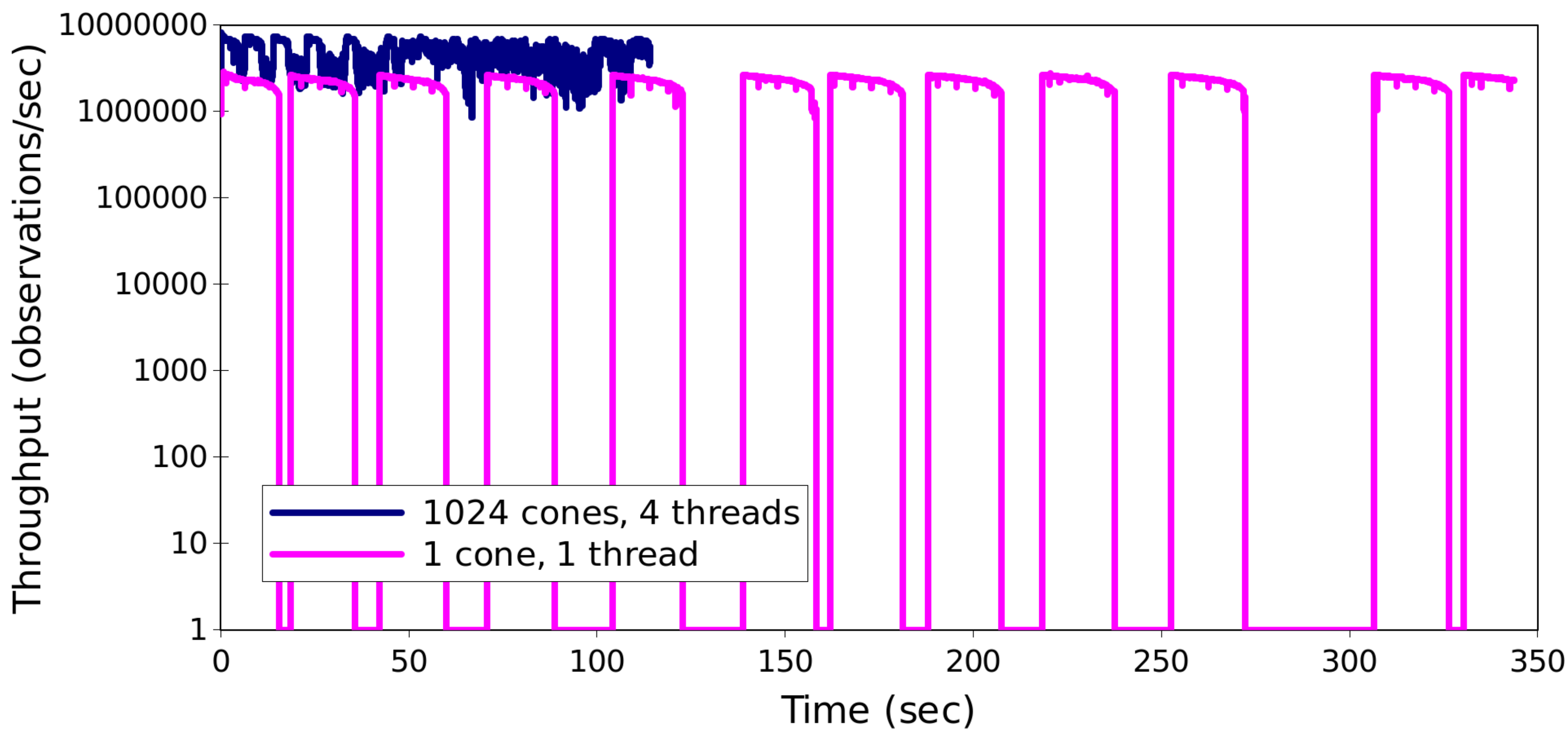
Values of α left to right: $1, 0.33, 0.14, 0.06$.

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.

Moving to Infinite Streams

Missing detail: Separate data structure in RAM of reported keys

- Reporting a key twice is an error
- Now also need to forget reports. Seeing months/years later is likely a new incident

- Must remove data before fills
  - Based on age
  - Based on importance

- Must do expiration without ever stopping the stream

Moving to Harder Standing Queries

- Host-based cyber data
  - Higher volume than network data
  - Consider examples based on MITRE ATT&CK Database

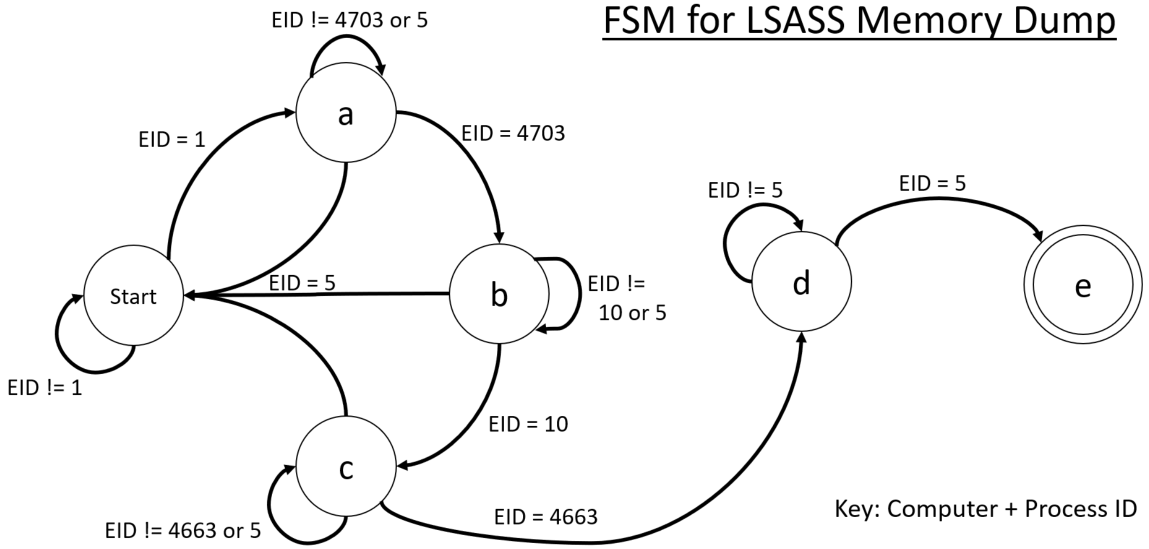
Example Cyber Standing Query

Stamp-collector problem

- Given fixed set of cyber events/use of tools
  - Learn about system services, software, network configuration…
  - See, e.g., the MITRE ATT&CK matrix, https://attack.mitre.org/
- Given a threshold T
- For each user, track the subset of these tools he/she uses
- Report any user who uses more than T of these tools

Event Sequence

Intuition: Report when a sequence of events matches the fingerprinting a known attack



- Finite-State Machine in this case
  - Database stores a contiguous set of events. Accept when run from start?
    - Not clear this is the best definition
  - Generally cannot test until flush to the bottom

Final Thoughts

- This work bridges the gap between streaming and external memory
- Finite streams: compromise between fast ingestion and queries, but can approximately have both

ACDA example:

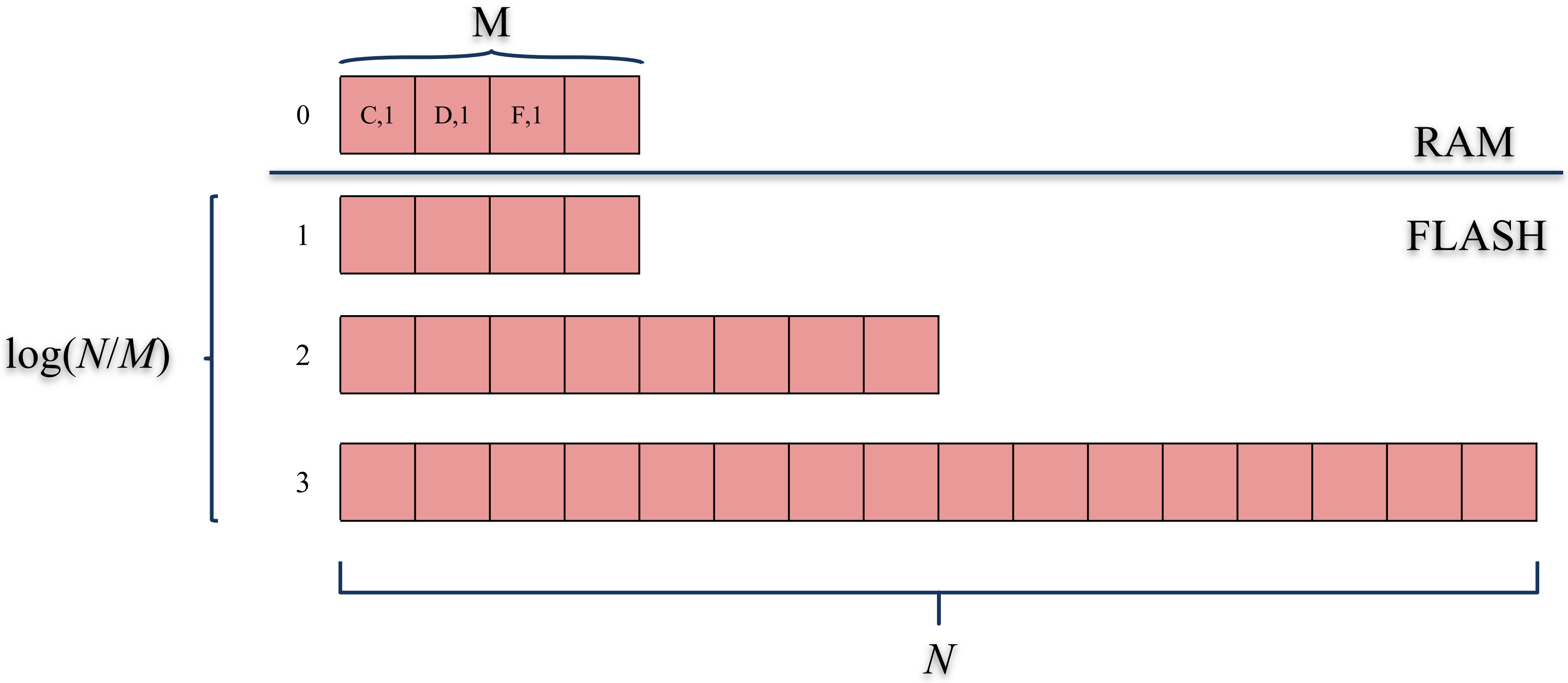
- Motivated by a real problem
- Finite-stream version has some serious theory
  - SIGMOD paper, Transactions on Databases paper
- This work has had practical impact (sorry, no details)

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.

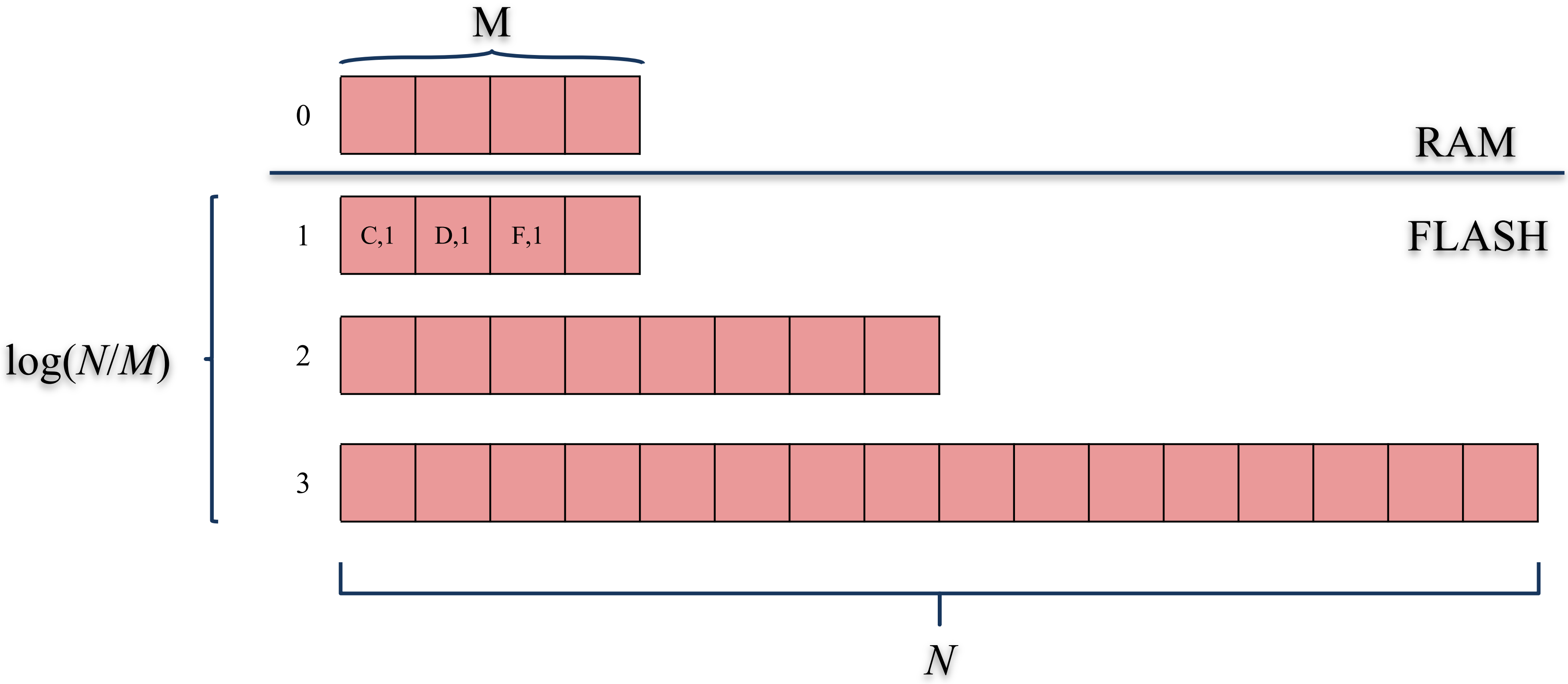
Back Up/Extra Slides

Ingestion “cascades”



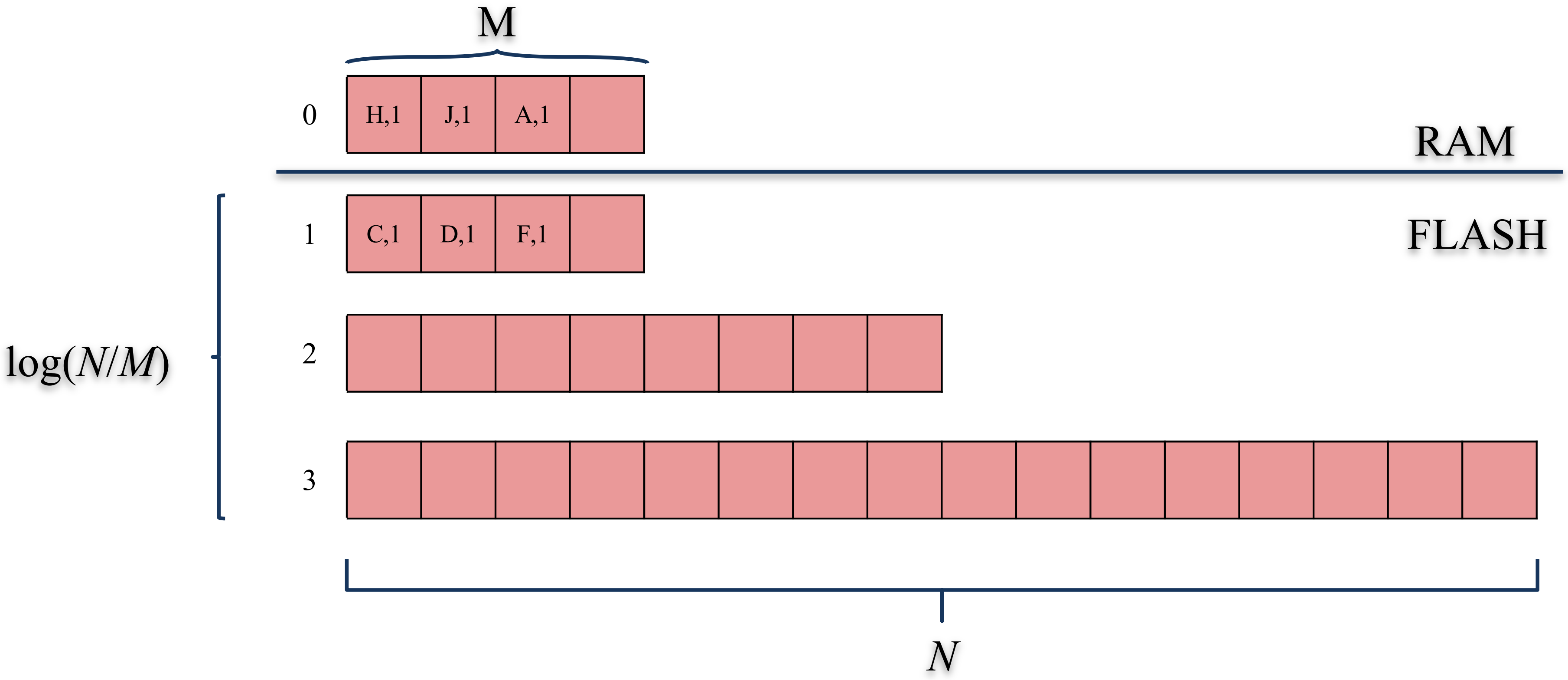
- Items are first inserted into the in-memory hash table.
- When the in-memory table reaches maximum load factor it flushes

Ingestion “cascades”

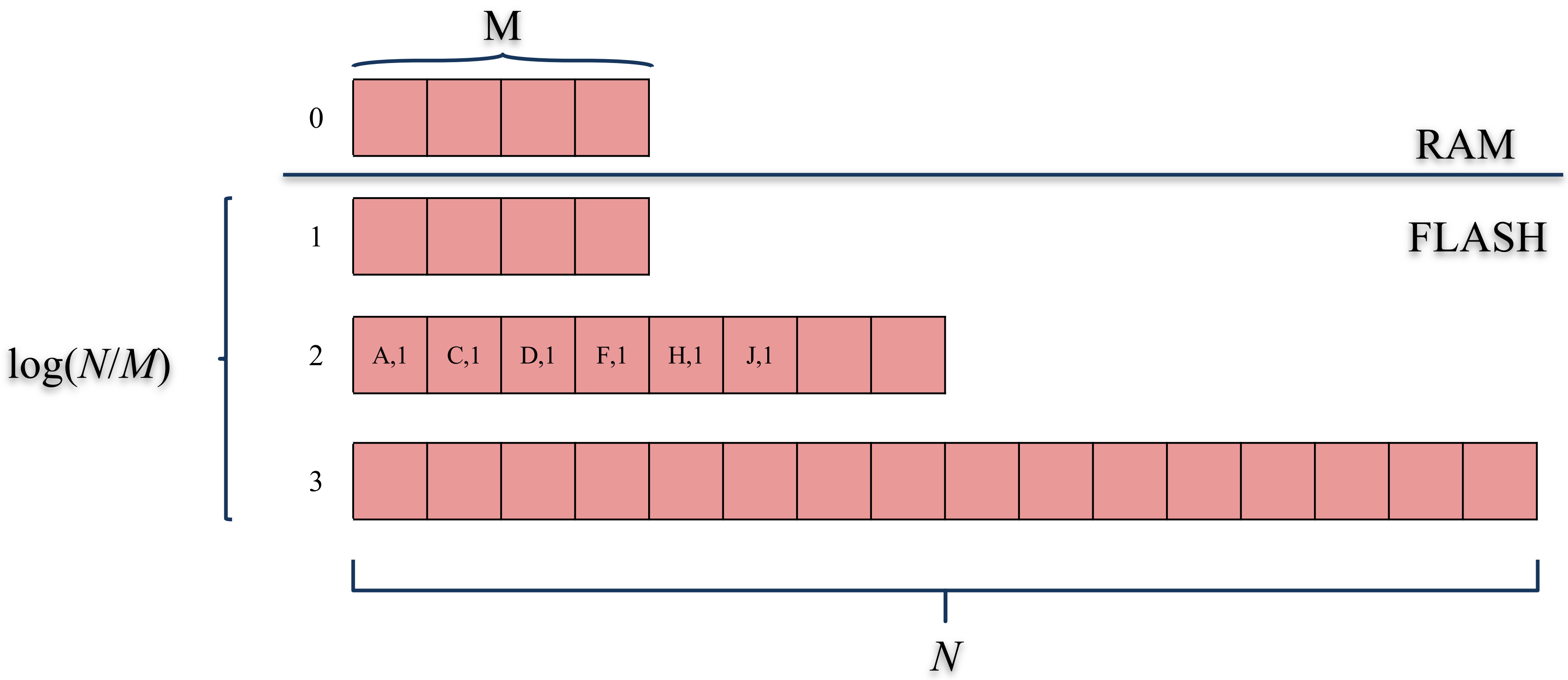


- During a flush, find the smallest i such that the items in $l_0, \ldots, l_i$ can be merged into level i.

Ingestion “cascades”



Ingestion “cascades”



M

0

RAM

FLASH

$\log(N/M)$

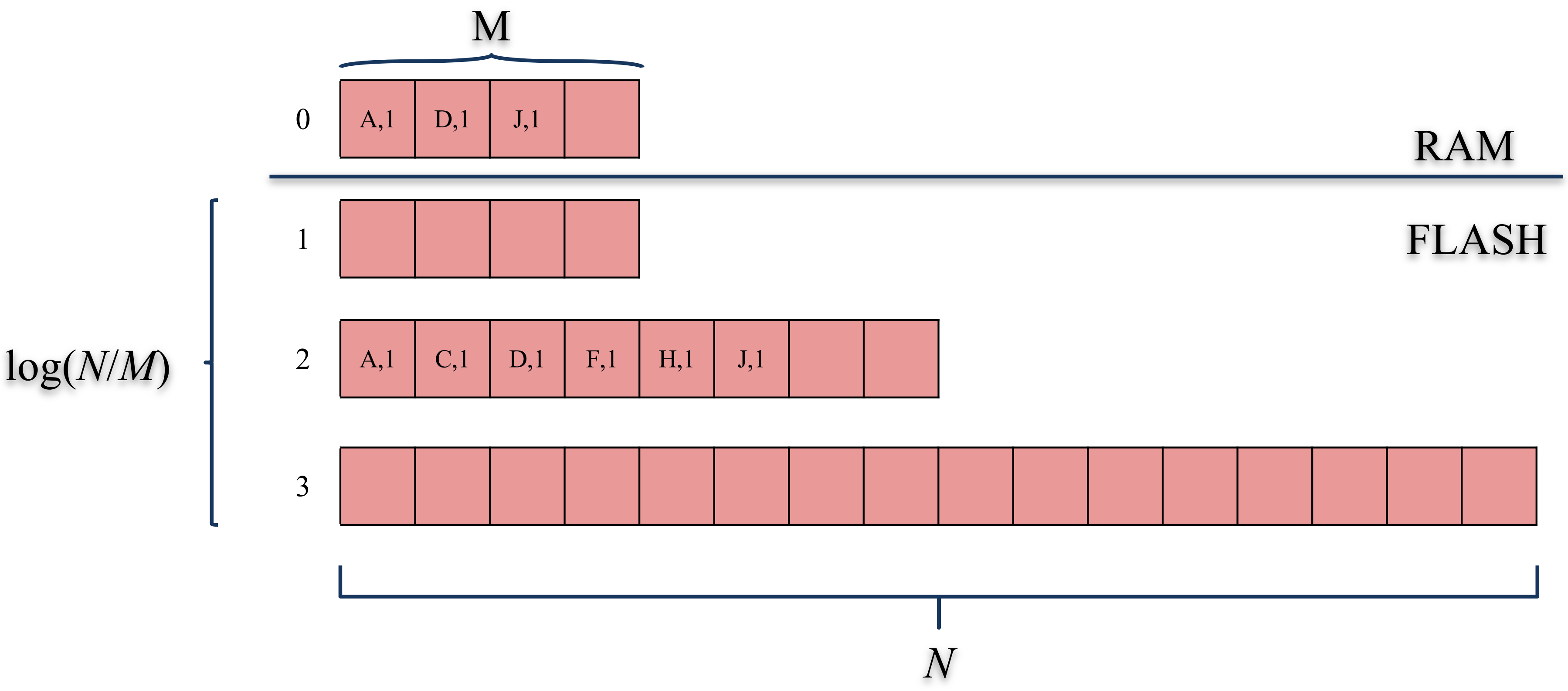
1

2 A,1 C,1 D,1 F,1 H,1 J,1

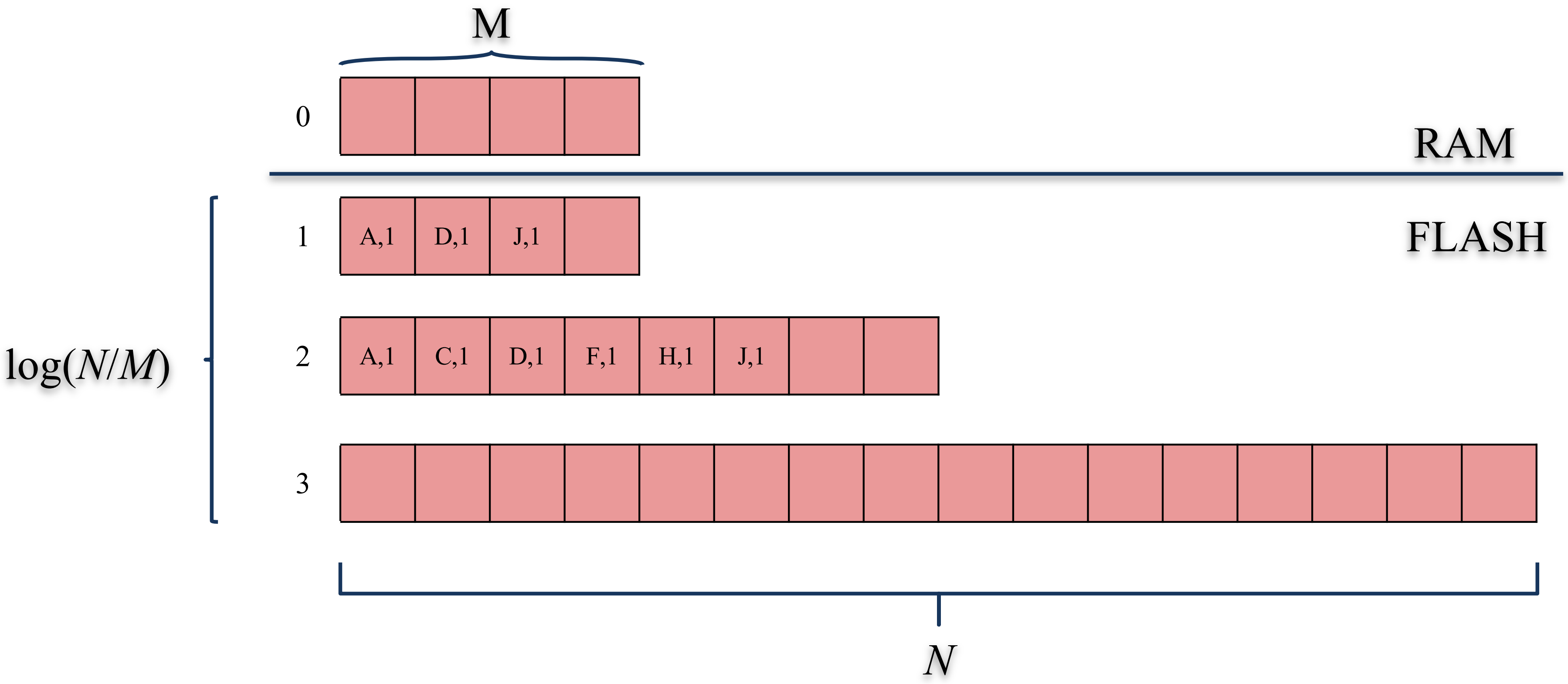
3

N

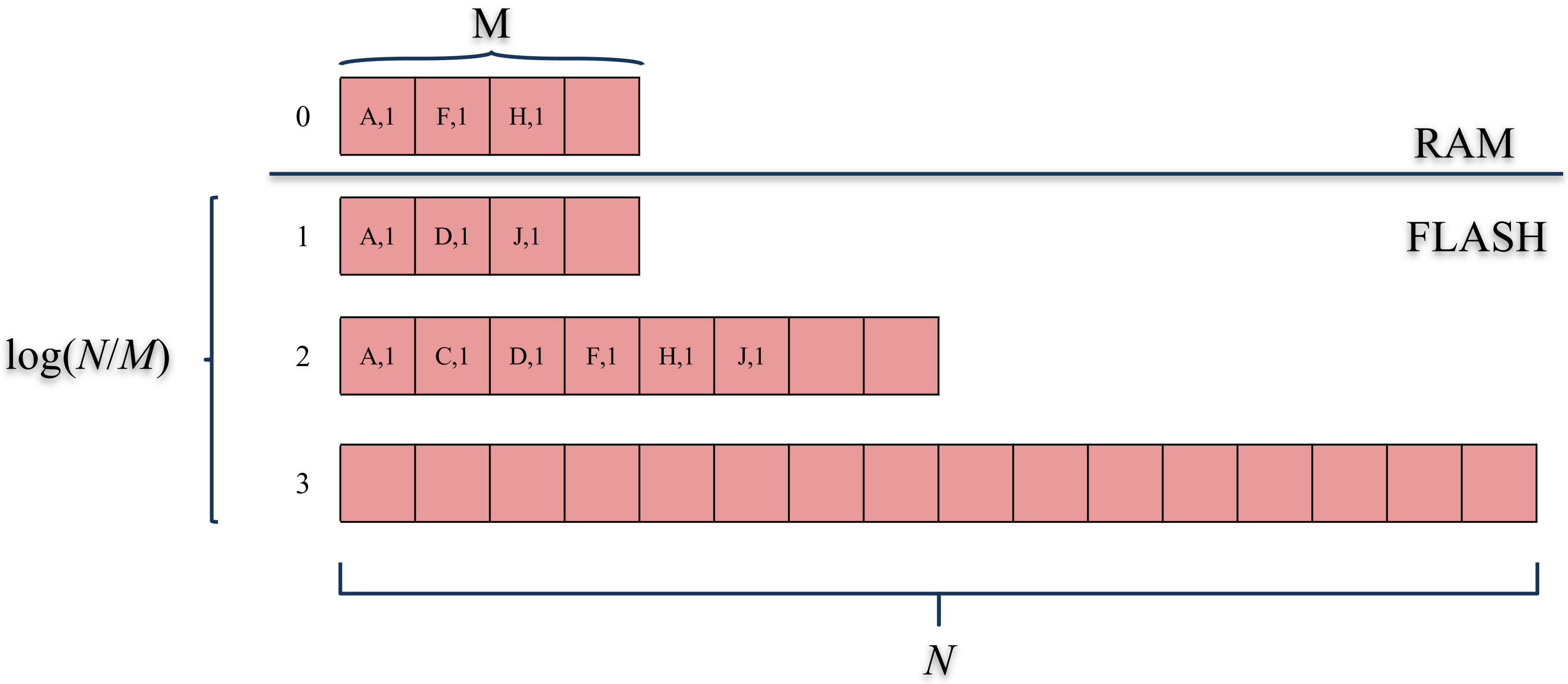
Ingestion "cascades"



Ingestion “cascades”



Ingestion "cascades"



M

0 A,1 F,1 H,1

RAM

FLASH

1 A,1 D,1 J,1

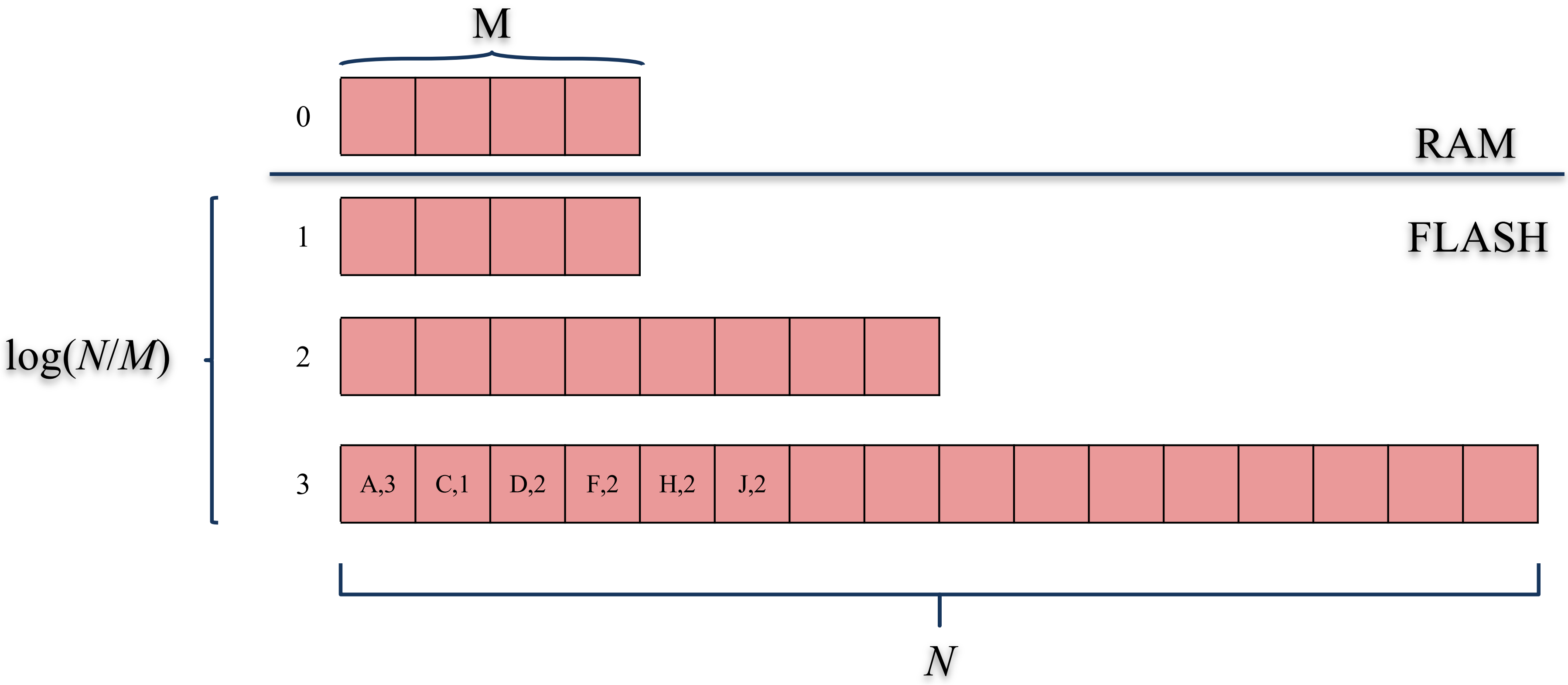
$\log(N/M)$

2 A,1 C,1 D,1 F,1 H,1 J,1

3

N

Ingestion "cascades"



M

0

RAM

FLASH

1

log(*N*/*M*)

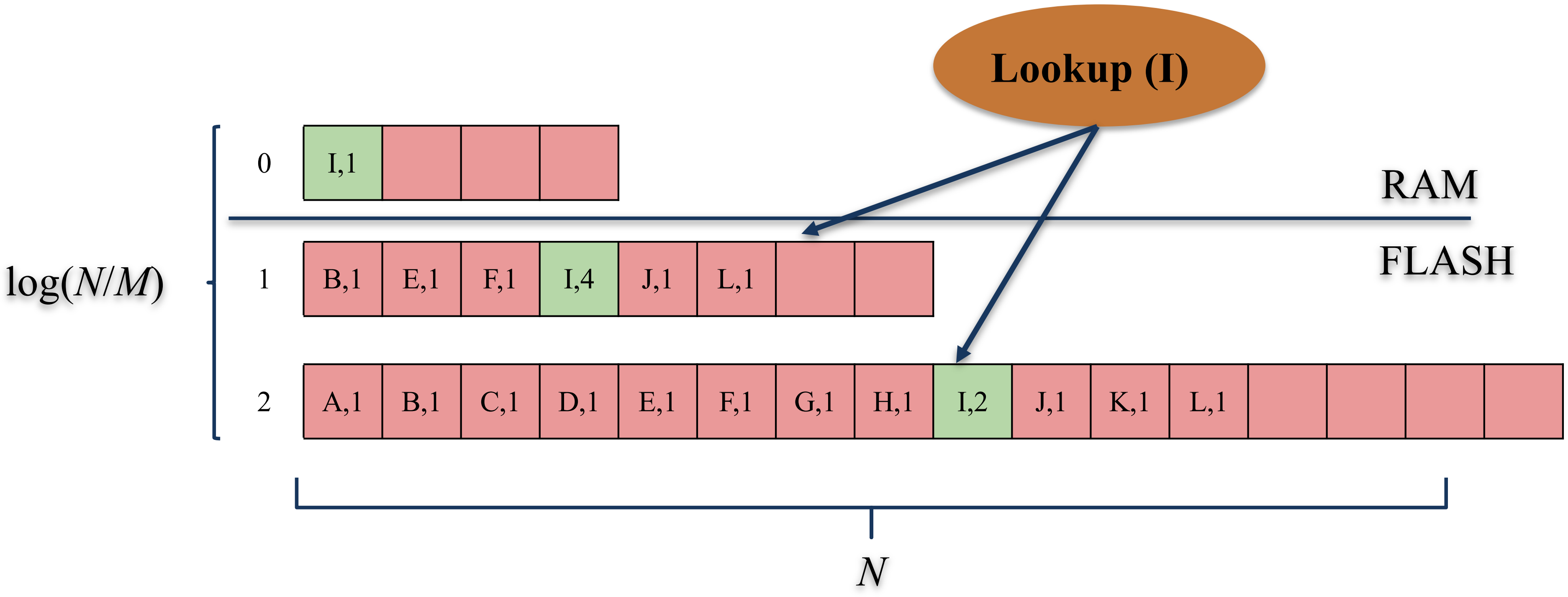
2

3

A,3 C,1 D,2 F,2 H,2 J,2

N

How to do immediate reporting



Lookup (I)

RAM

FLASH

log(*N*/*M*)

0 | I,1 | | |

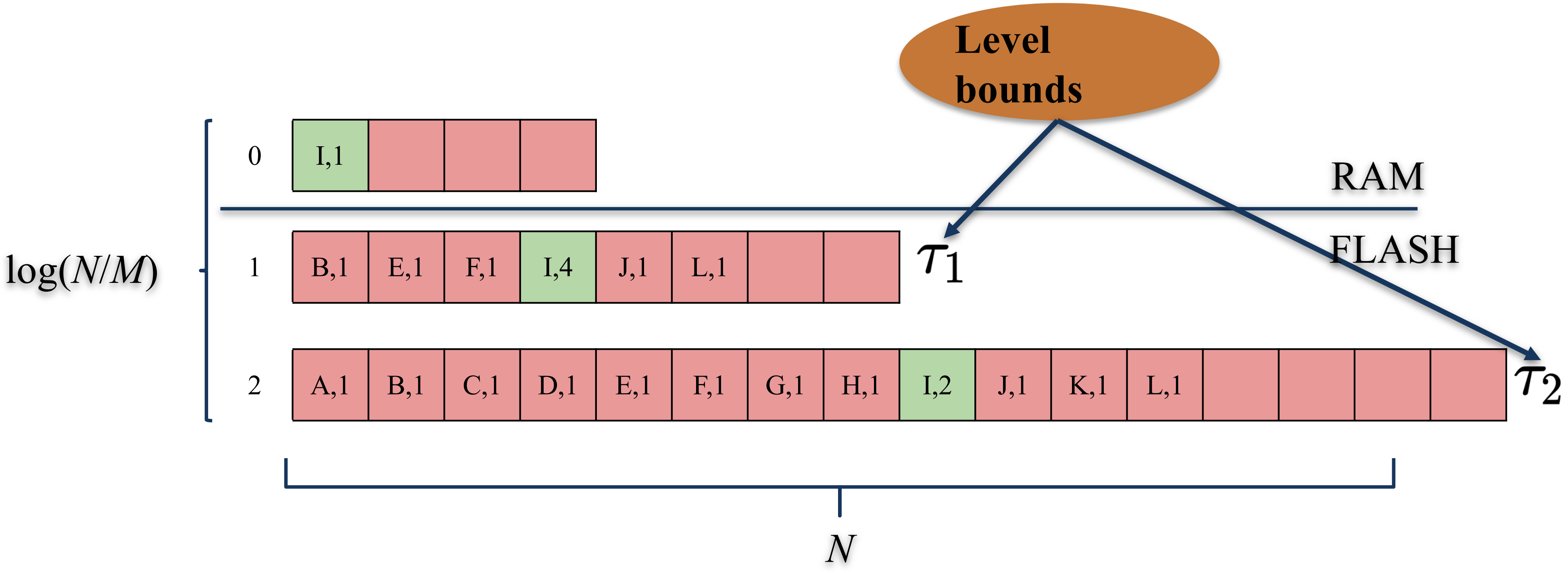
1 | B,1 | E,1 | F,1 | I,4 | J,1 | L,1 | |

2 | A,1 | B,1 | C,1 | D,1 | E,1 | F,1 | G,1 | H,1 | I,2 | J,1 | K,1 | L,1 | | | |

N

- In a cascade filter, we would need to perform multiple I/Os for every new item

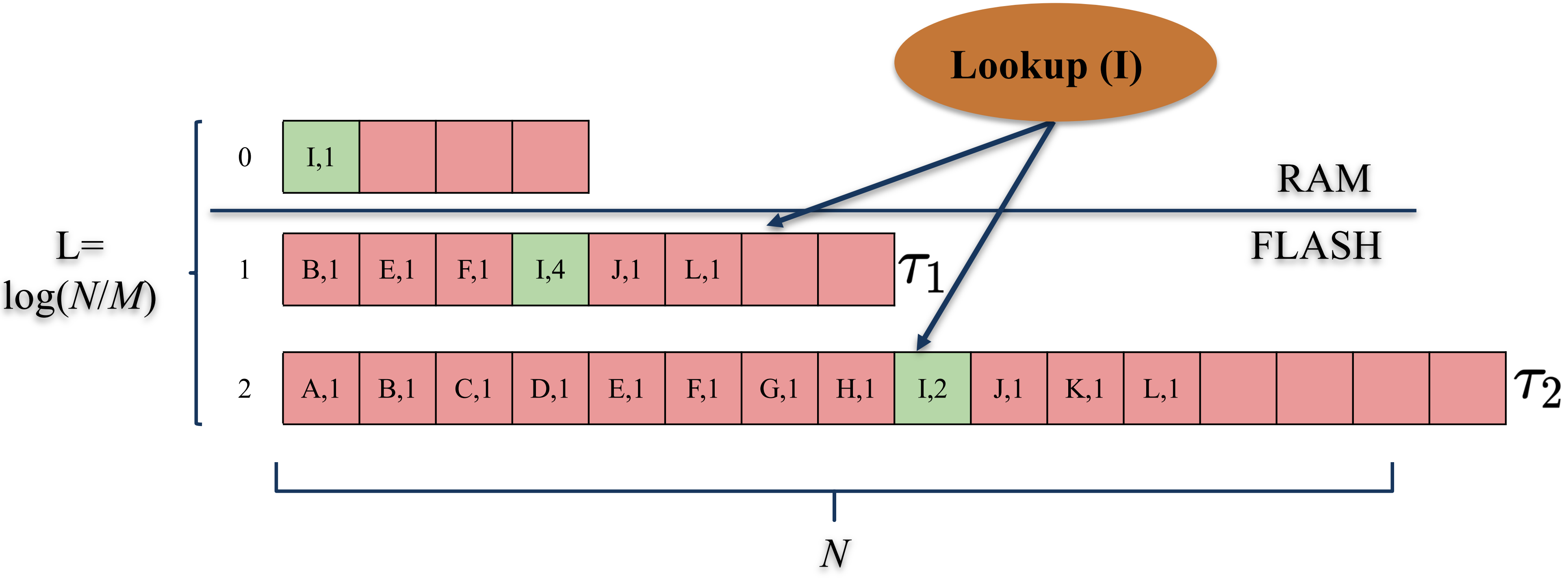
Level Thresholds



0	I,1															
1	B,1	E,1	F,1	I,4	J,1	L,1										
2	A,1	B,1	C,1	D,1	E,1	F,1	G,1	H,1	I,2	J,1	K,1	L,1				

- 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



Lookup (I)

RAM

FLASH

L= log(N/M)

0 | I,1 | | |

1 | B,1 | E,1 | F,1 | I,4 | J,1 | L,1 | | | τ_1

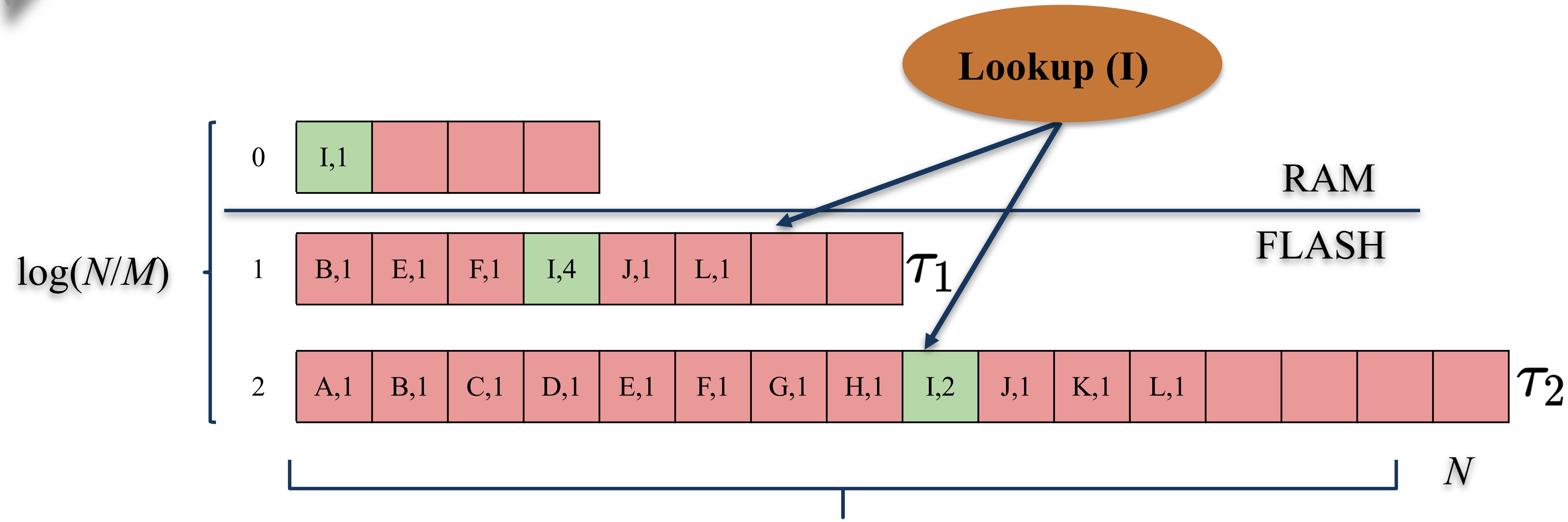
2 | A,1 | B,1 | C,1 | D,1 | E,1 | F,1 | G,1 | H,1 | I,2 | J,1 | K,1 | L,1 | | | | | τ_2

N

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

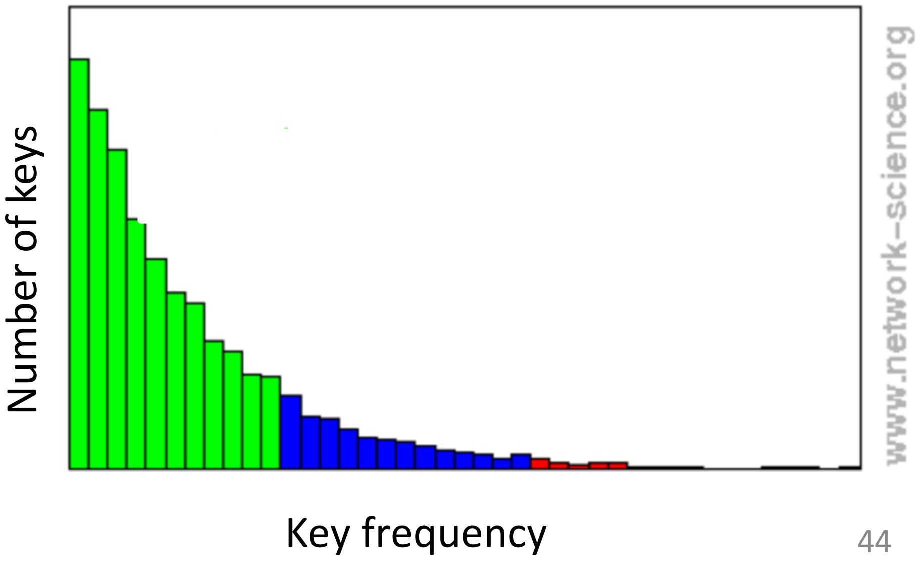
Lookup if $\text{RamCount} = 24 - \sum_{i=1}^{L} \tau_i$

Popcorn filter



0	I,1															
1	B,1	E,1	F,1	I,4	J,1	L,1										
2	A,1	B,1	C,1	D,1	E,1	F,1	G,1	H,1	I,2	J,1	K,1	L,1				

- 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

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

About 1/1000

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

< 1/16 for Firehose for θ=2.5 and N/M=25

When

$$\Theta > 2$$

$$\phi N > \gamma$$

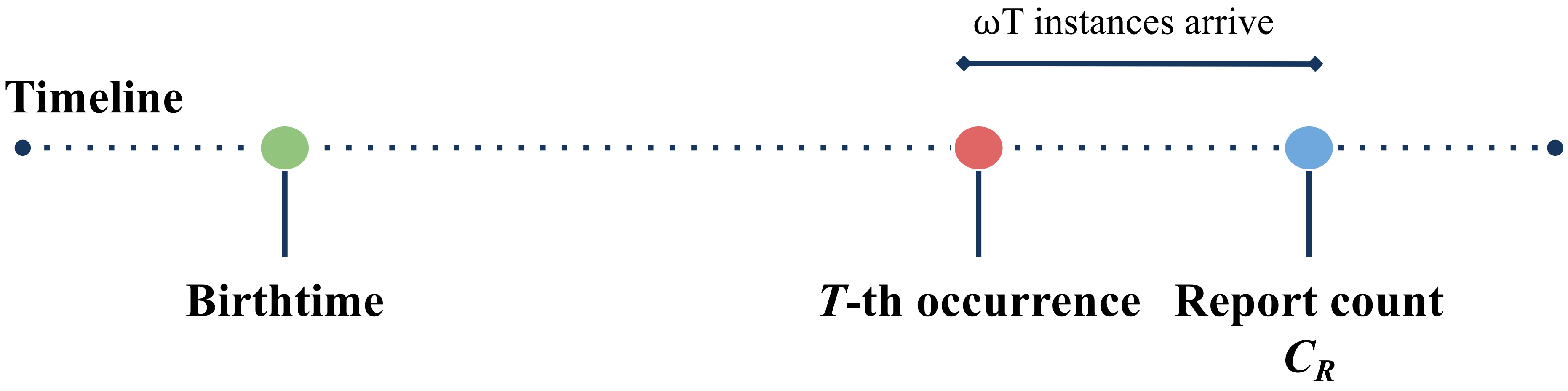
$$\gamma = \frac{e^{1/(\Theta-1)}}{e^{1/(\Theta-1)} - 1} \cdot \left(\frac{N}{M}\right)^{1/(\Theta-1)}$$

Note: for θ < 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

$$\Theta > 2$$

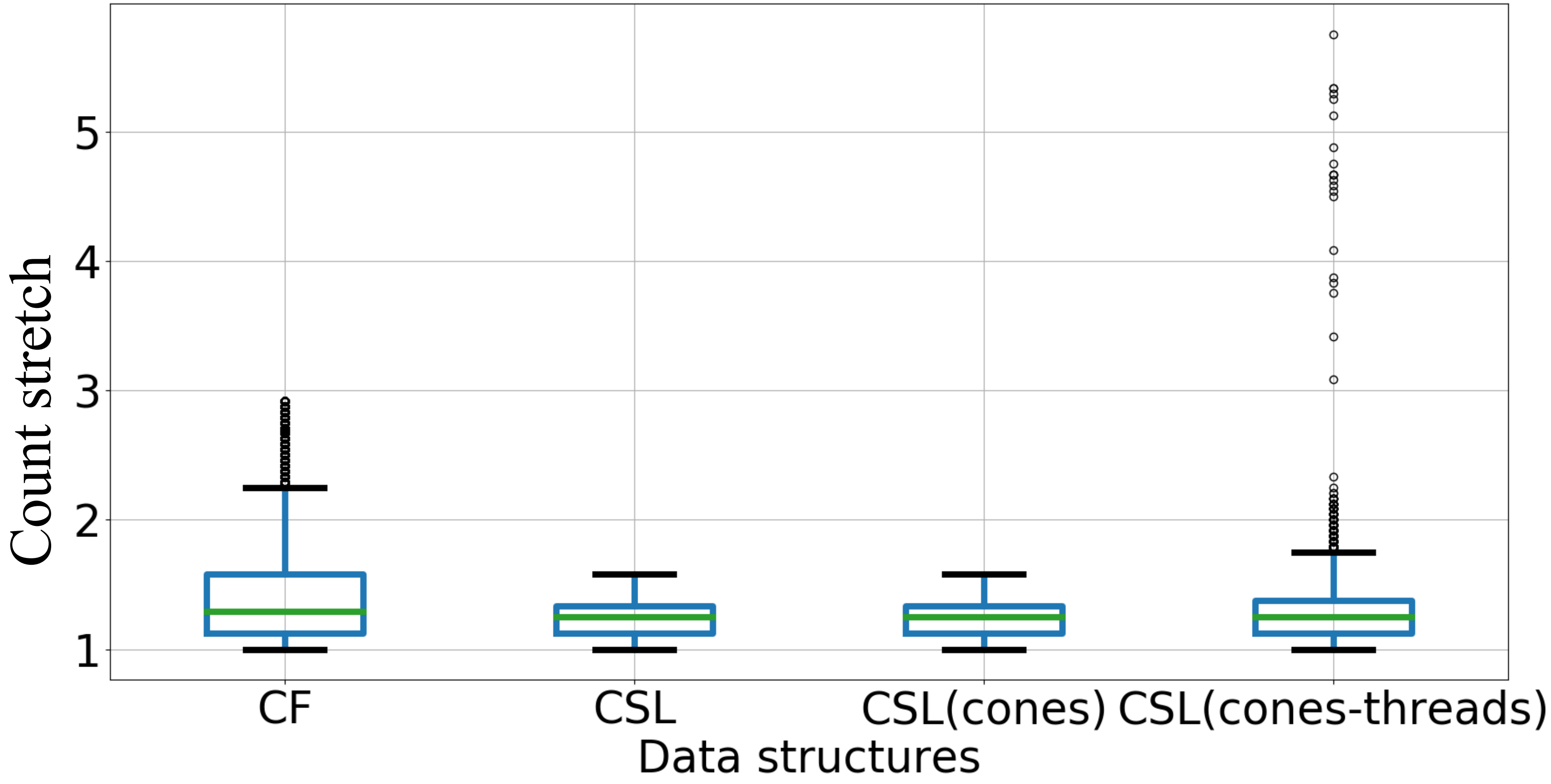
$$\phi N \cdot \omega > \frac{e^{1/(\Theta-1)}}{e^{1/(\Theta-1)} - 1}$$

Note: for θ < 2.96

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

So report by count 27

Count stretch



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

Multithreaded Count Stretch

- P = # of threads, T is reporting threshold
- 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.

Counting (Heavy Hitters)

- Intuition: Report a key having more “interesting” events than some threshold
- Examples:
- Scan detector – A computer attempting to scan a network to identify open ports and service vulnerabilities
  - Reconnaissance – Active Scanning (T1595.001, T1595.002)
- Brute force password guessing detector – A computer or account attempting to guess many passwords
  - Credential Access – Brute Force (T1110.001, T1110.002)
- Denial of Service Flood detector – A computer attempting to degrade a network resource by saturating the resource with requests
  - Impact – Endpoint Denial of Service (T1499.002, T1499.003)

Novelty Problem

- Intuition: Report an observation that has not occurred in the (known) past, ignoring the time component.
- Examples:
- Abnormal Access Pattern detector – Accounts accessing resources at abnormal times
  - Lateral Movement – Remote Service Session Hijacking (T1563)
  - Lateral Movement – Remote Services (T1021)
  - Lateral Movement – Software Deployment Tools (T1072)
  - Lateral Movement – Use Alternate Authentication Material (T1550)
  - … many many more
- Unknown USB Stick detector – Users inserting dangerous removeable media into their systems
  - Initial Access - Replication Through Removable Media (T1091)

Last-Bin Expiration

- For aging, delete last bin, when flush to bottom
  - Time-stretch guarantee for fixed alpha
- For importance, remove # = size of last bin when flush to bottom