Data Structures in Adversarial Environments

Sam A. Markelon

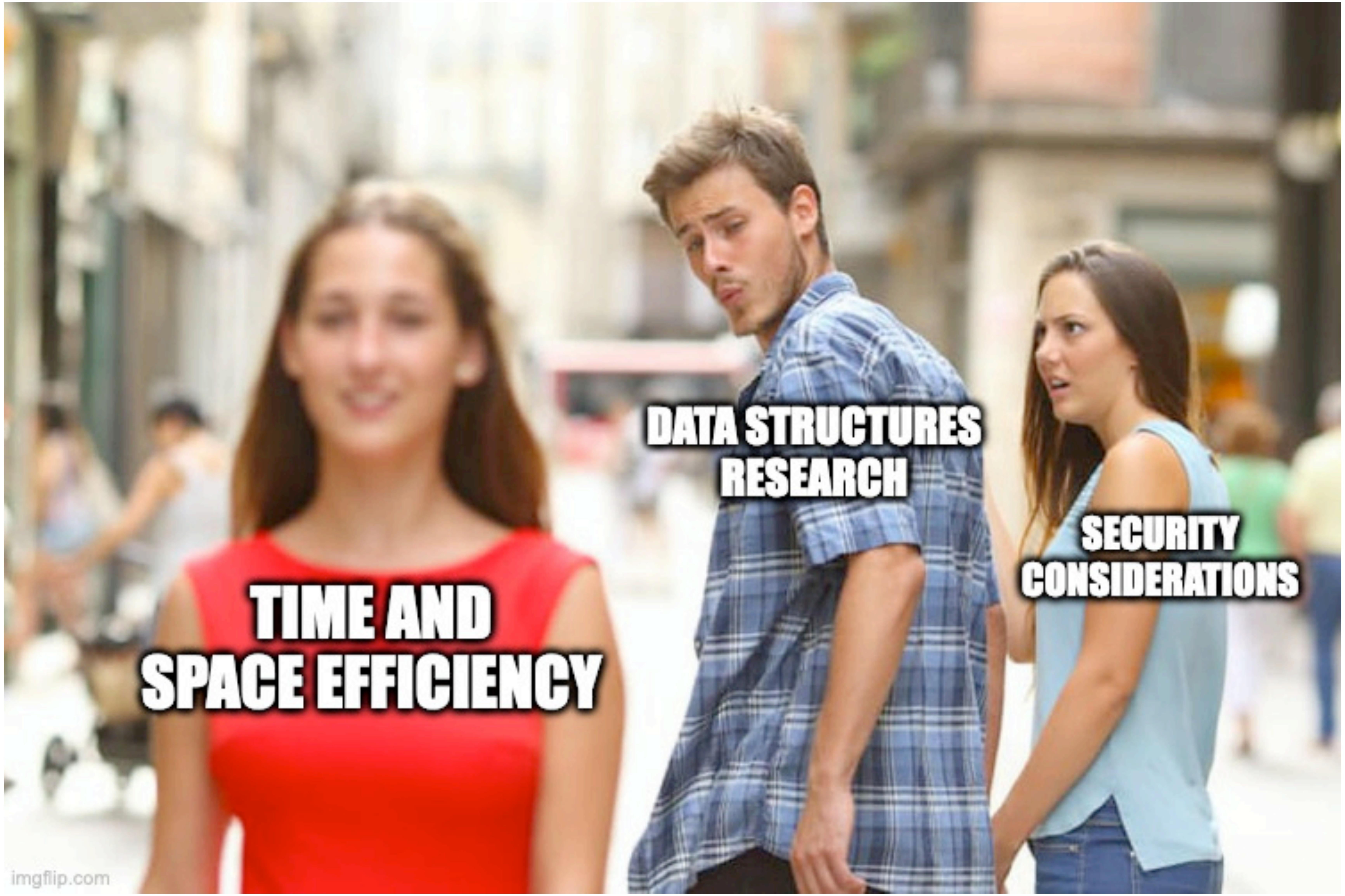
Dissertation Defense

June 19, 2025

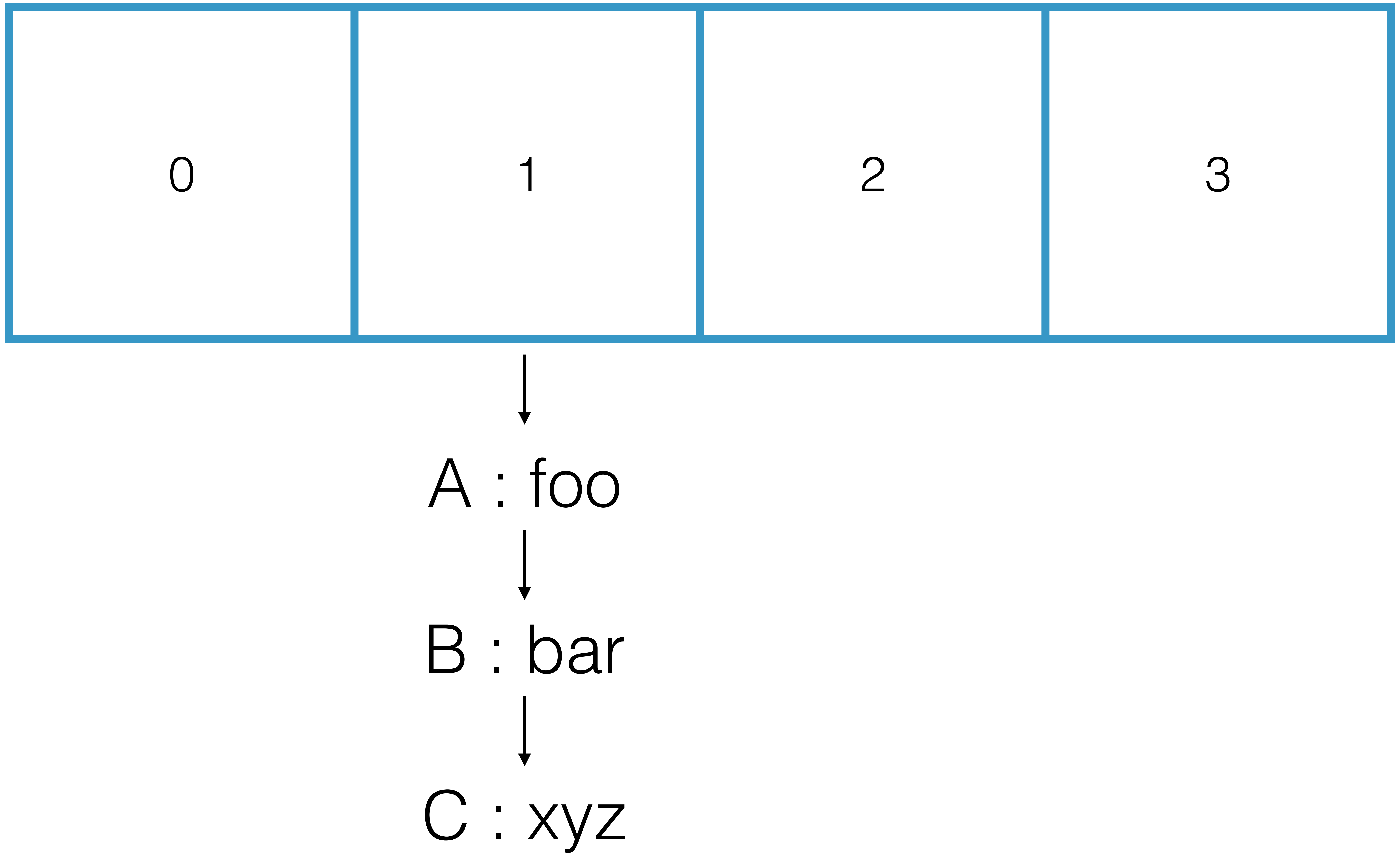
What are data structures?

Data structures define **representations** of possibly dynamic (multi)sets along with the **operations** that can be performed on the representation.

A Need for Speed (and Space)



Hash Flood DoS Attacks




```
hash(A) = 1
hash(B) = 1
hash(C) = 1
      .
      .
      .
```

Insertion of n elements ~ $O(n^2)$

Thesis Statement

For **compact frequency estimators** (a subclass of compressing probabilistic data structures) and **probabilistic skipping-based data structures** (including hash tables, skip lists, and treaps), **formal adversarial models** that capture the adaptive ability of adversaries can be defined under which these **structures are demonstrably insecure**. Specifically, these models capture scenarios in which an adversary, with knowledge of the structure's parameters, query responses, and, in certain cases, initialization choices and representations, can **degrade correctness or performance guarantees beyond acceptable thresholds**. It is further claimed that, for these same adversarial models, it is possible to **construct new variants of these data structures that are provably robust, with explicit, formal guarantees on their correctness, performance, and security under attack.**

Compressing Probabilistic Data Structures

Compactly represent (a stream of) data

and

provide **approximate answers** to queries about the data

- Frequency estimation
  How many times does x occur in the stream?
  Count-min sketch, Heavy-keeper
- Membership queries
  Is x in the set?
  Bloom filter, Cuckoo filter
- Cardinality estimation
  How many distinct elements in the set?
  HyperLogLog, KMV estimator

Compressing Probabilistic Data Structures

Compactly represent
(a stream of) data

and

provide **approximate answers** to
queries about the
data

- Bound on the **response error**
  - False positive rate for BF
  - Over-estimation bound for CMS
- Bound is strictly **non-adaptive**
  - Data does not depend on internal $$ of structure

Compressing Probabilistic Data Structures

Frequency estimation PDS

find the most sold stocks in the past minute

Membership query PDS

check certificate revocations in TLS/SSL

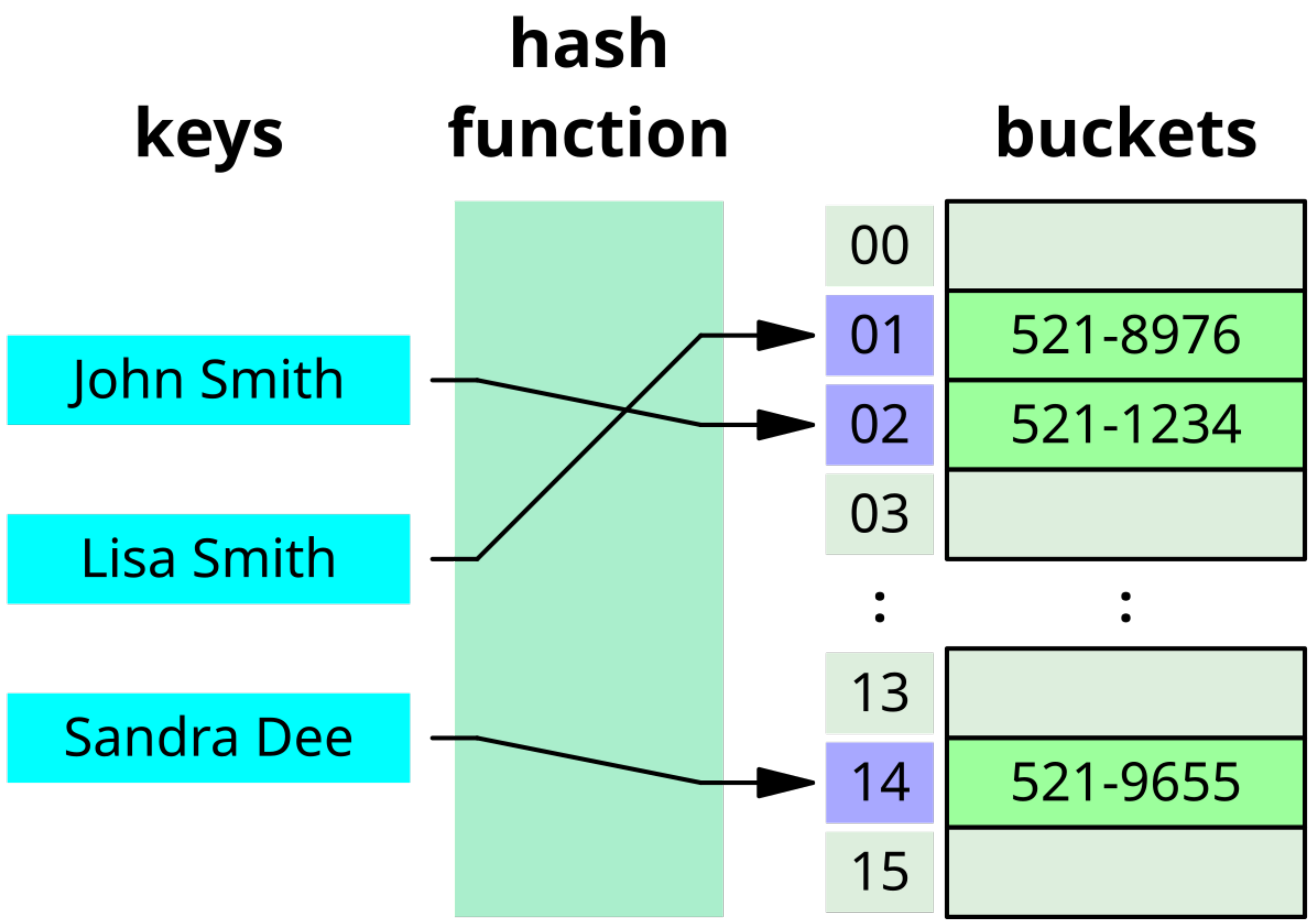
Frequency estimation PDS

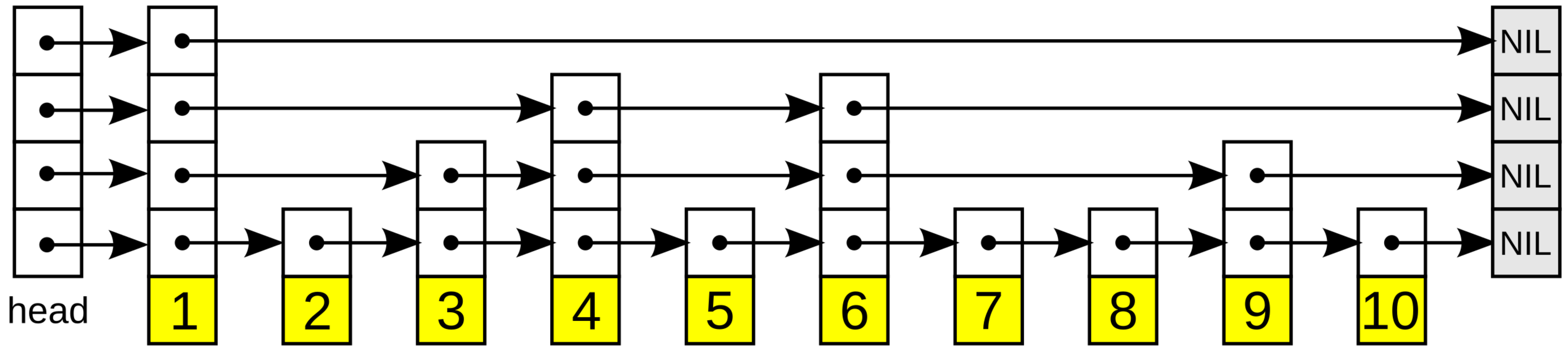
identify possible DoS threats (network-monitoring

Cardinality estimation PDS

count the number of distinct users on a given service

Probabilistic Skipping-Based Data Structures





Dissertation Work

Compact Frequency Estimators in Adversarial Environments

CCS '23

Probabilistic Data Structures in the Wild: A Security Analysis of Redis

CODASPY '25

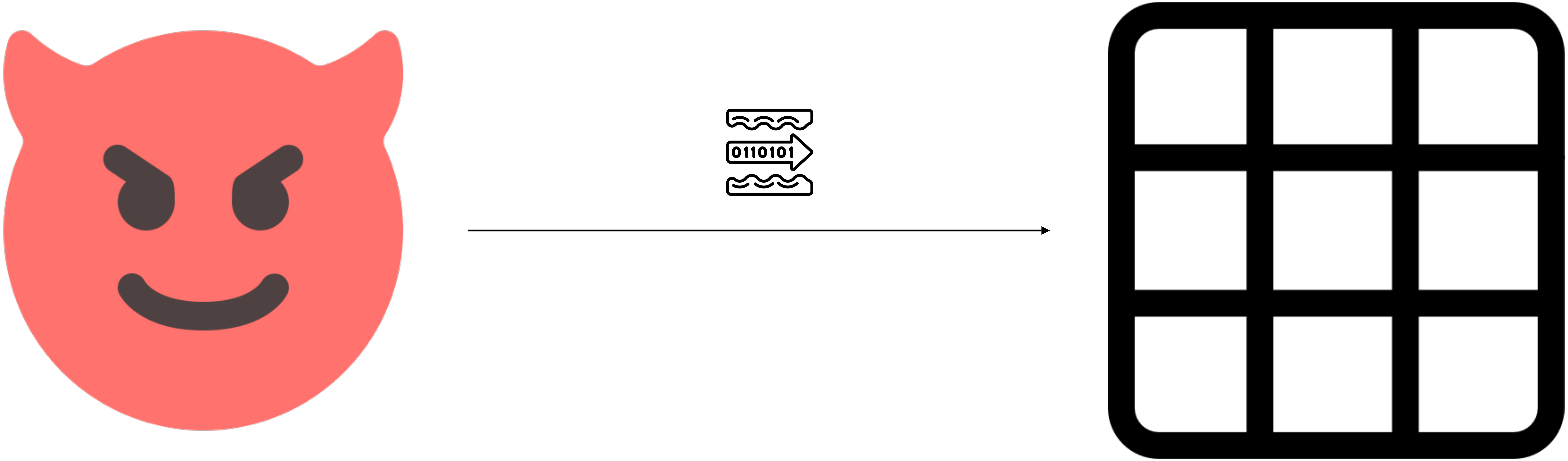
Probabilistic Skipping-Based Data Structures with Robust Efficiency Guarantees

Submitted to CCS '25

Compact Frequency Estimators in Adversarial Environments

Sam A. Markelon, Mia Filić, and Thomas Shrimpton
(CCS '23)

Adversarial Correctness of CFE



Adversarial Correctness of CPDS

[AuthorsYear]	Structures	Security Proof Style
[NY15]	Bloom filter	Game based
[CPS19]	Bloom Filter Counting Filter Count-min Sketch	Game based
[PR22]	HyperLogLog	Simulation
[FPUV22]	Bloom Filter Cuckoo Filter	Simulation (privacy notions!)
[MFS23]	Count-min Sketch HeavyKeeper	Game based*

Standard hash functions:
Large correctness errors

Swap to a keyed primitive:
Adversarial robust structures*

Count-min Sketch (CMS)

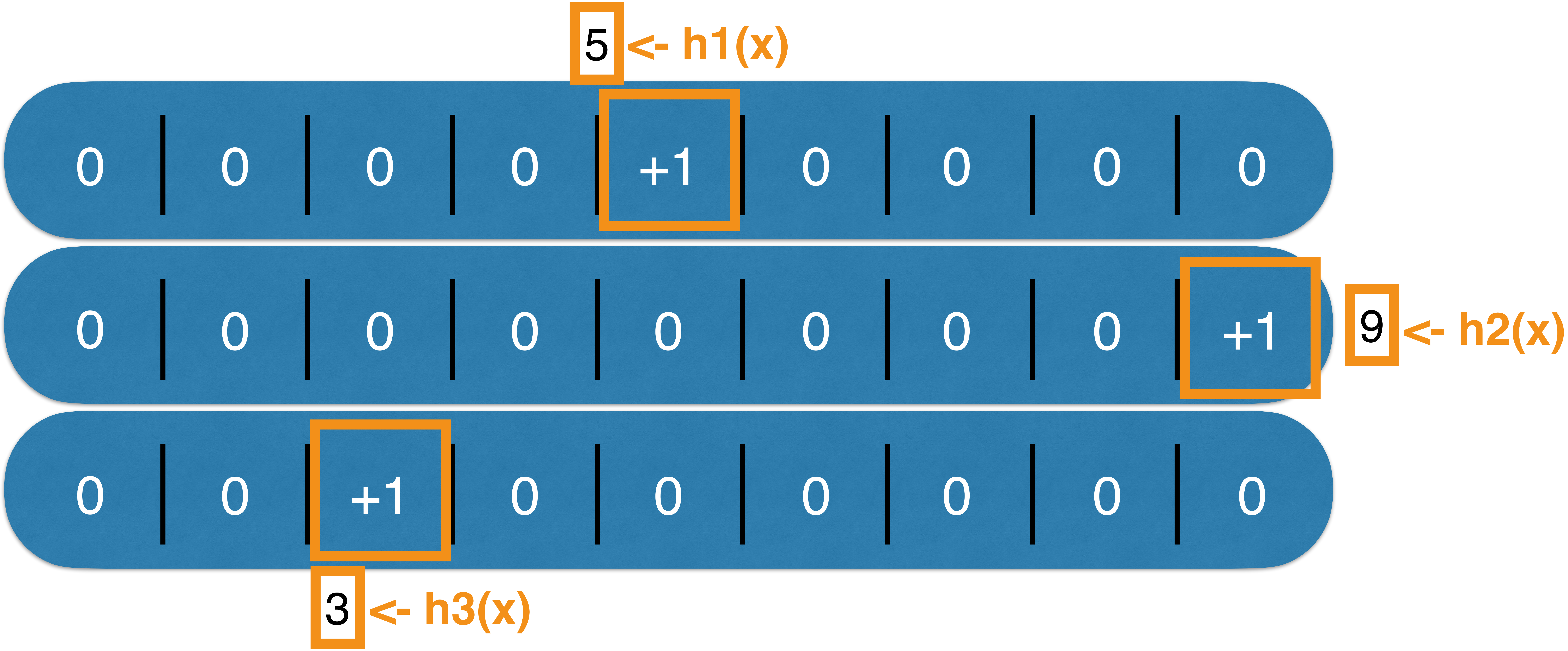
m columns

k rows

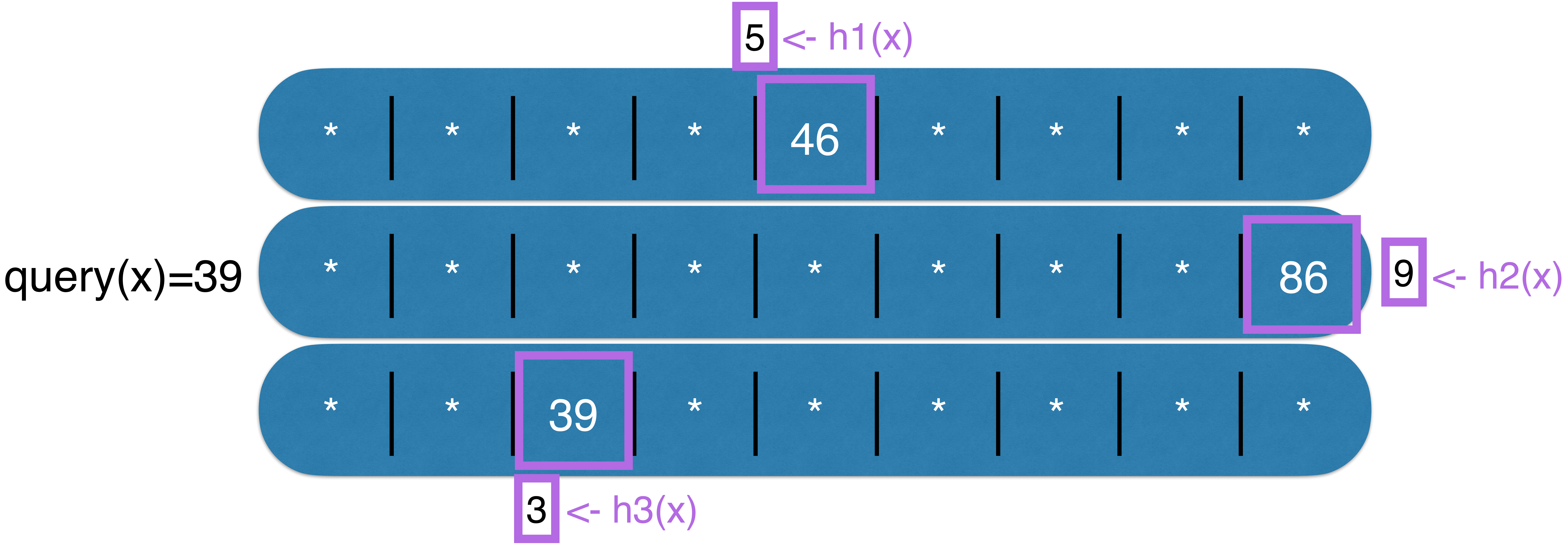
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

CMS Insert

insert(x)



CMS Query



CMS Properties

- Only overestimates
- “Honest Setting” guarantee
- Adversarial setting?

CFE Error Model (simplified)

x

CMS[🔑]:
insert
query

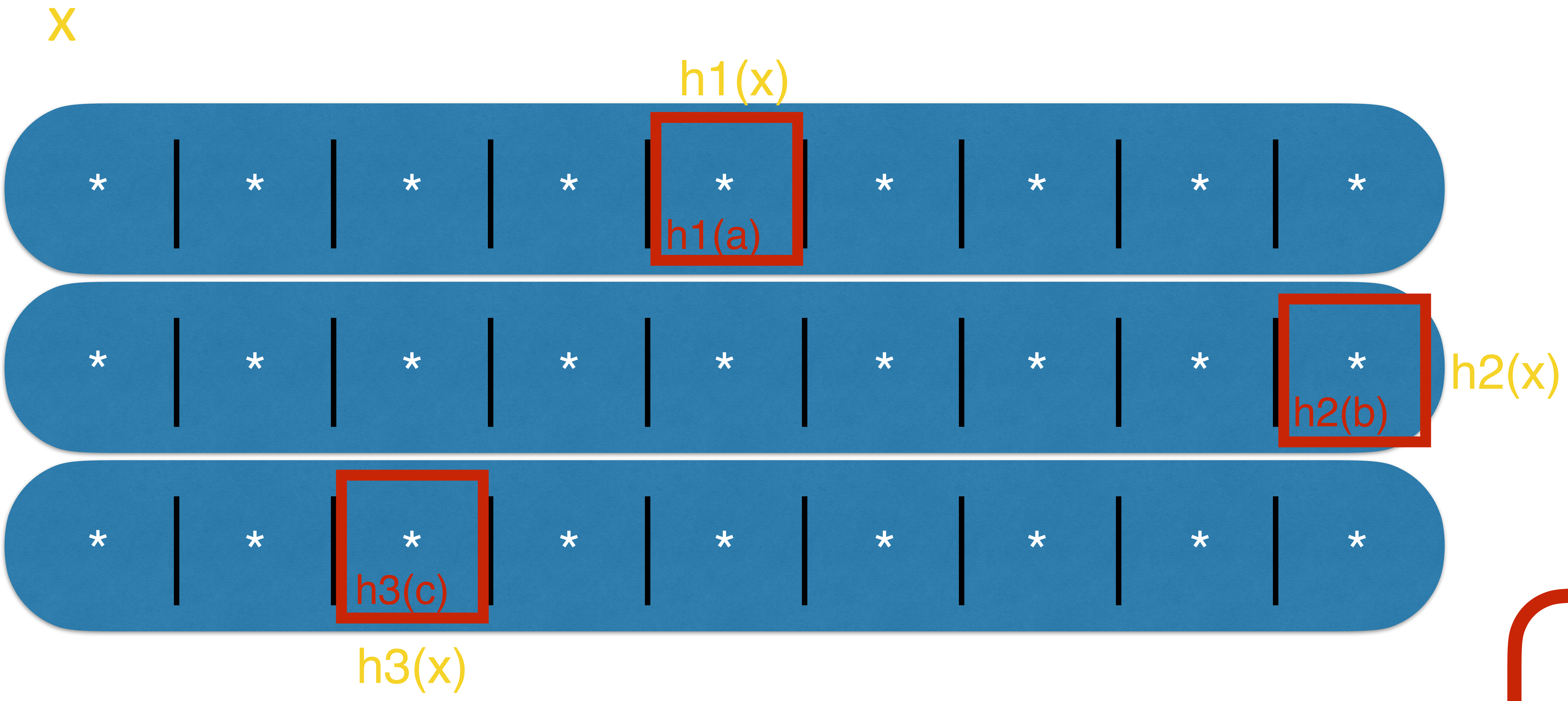
reveal

hash

Maximise
CMS error

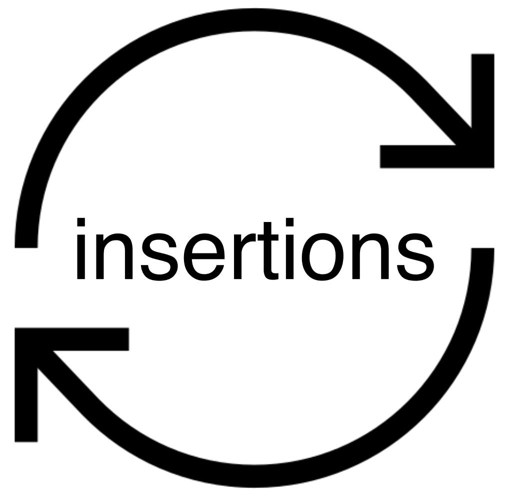
query(x) >> true_frequency(x)

CMS "Public Hash" Attack



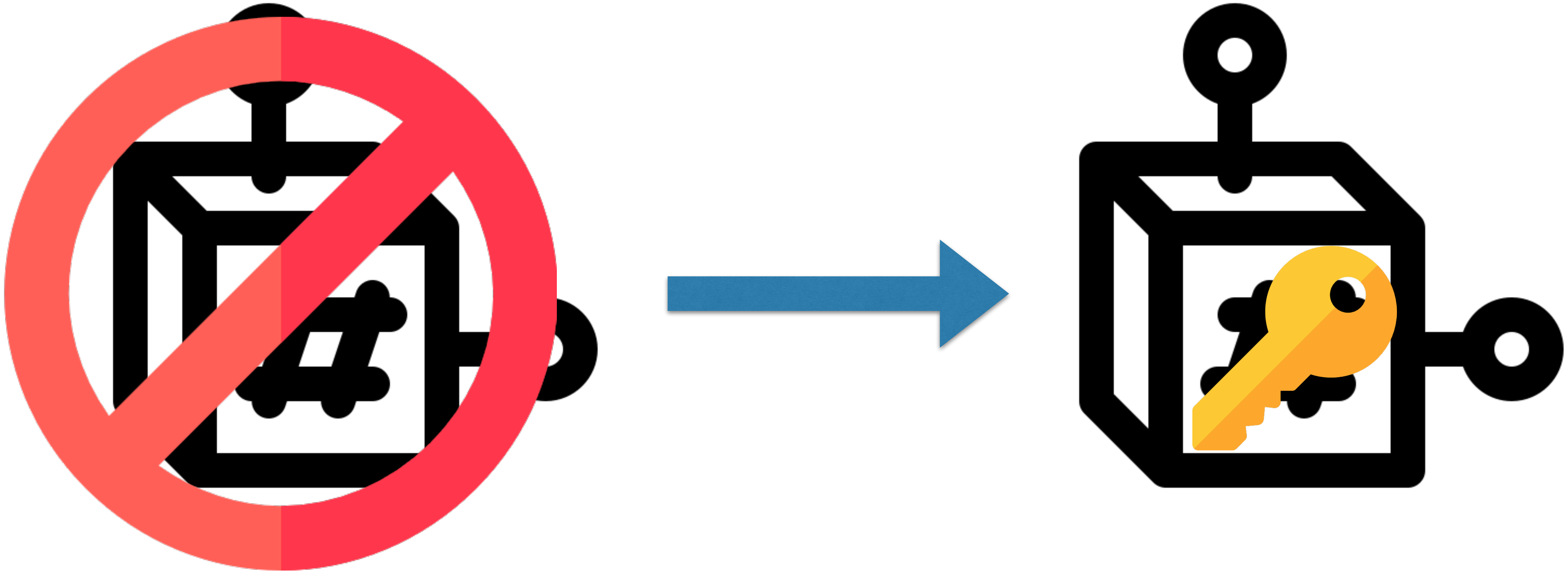
Cover set = {a, b, c}

Cover set

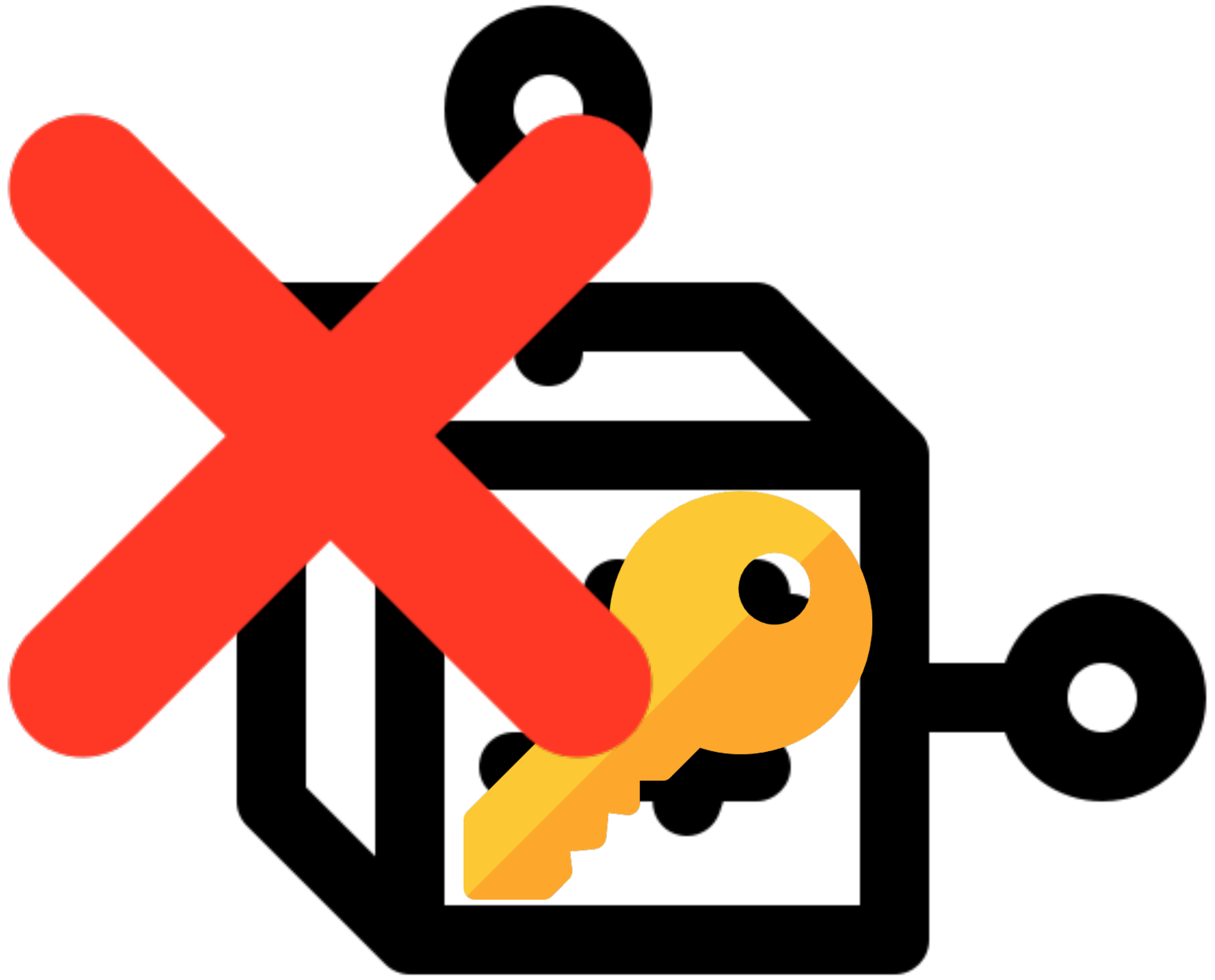


Err = insertions/k

CMS Attacks Mitigations



CMS Attacks Mitigations



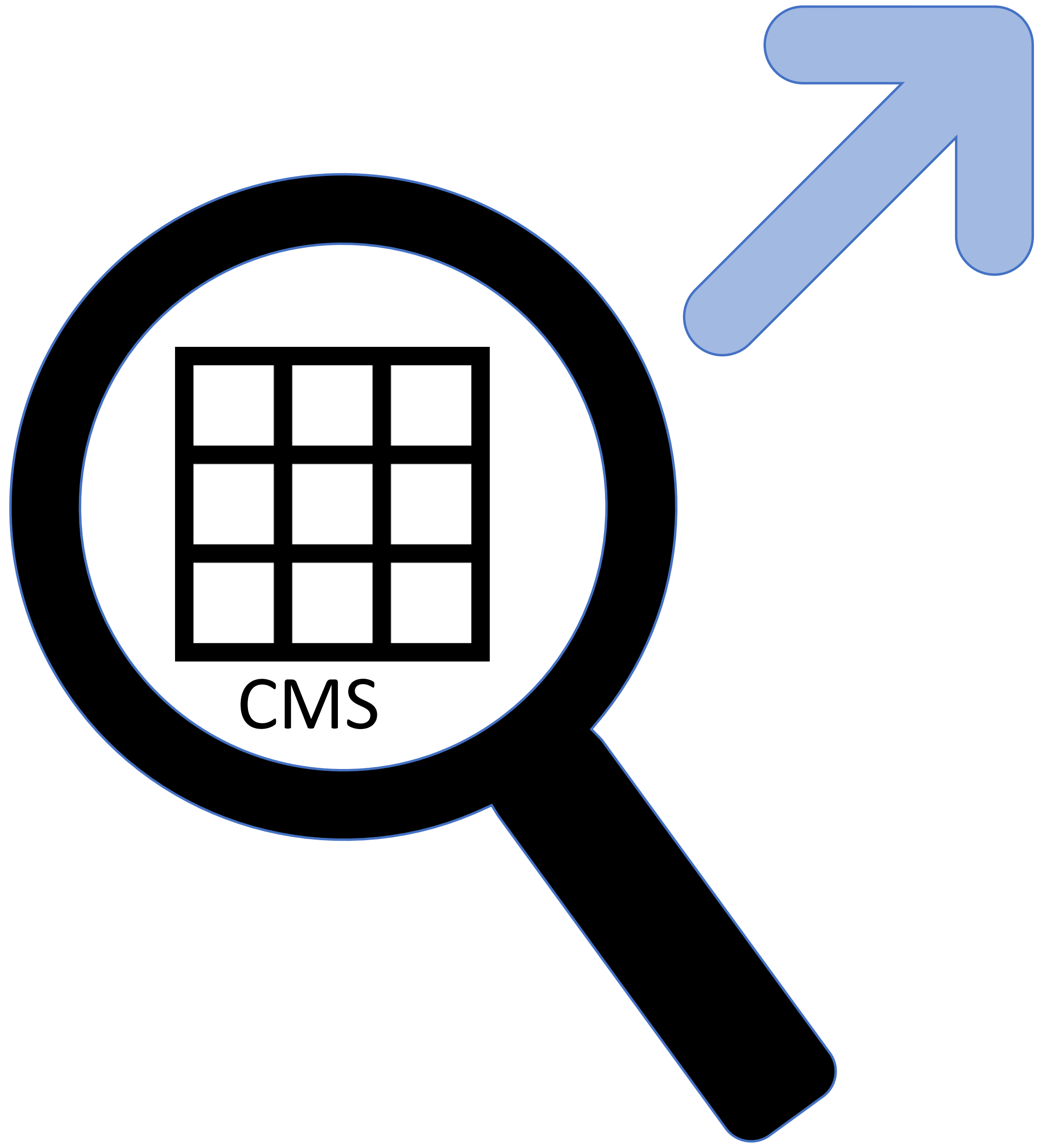
Still attacks when using a PRF and blackboxed structure!

Existing CFEs are not adversarially robust!

Motivating a more robust CFE

$$\text{cnt} = n_x + \sum_{y \in V_x^i} n_y$$

CMS minimizes the "collision noise"

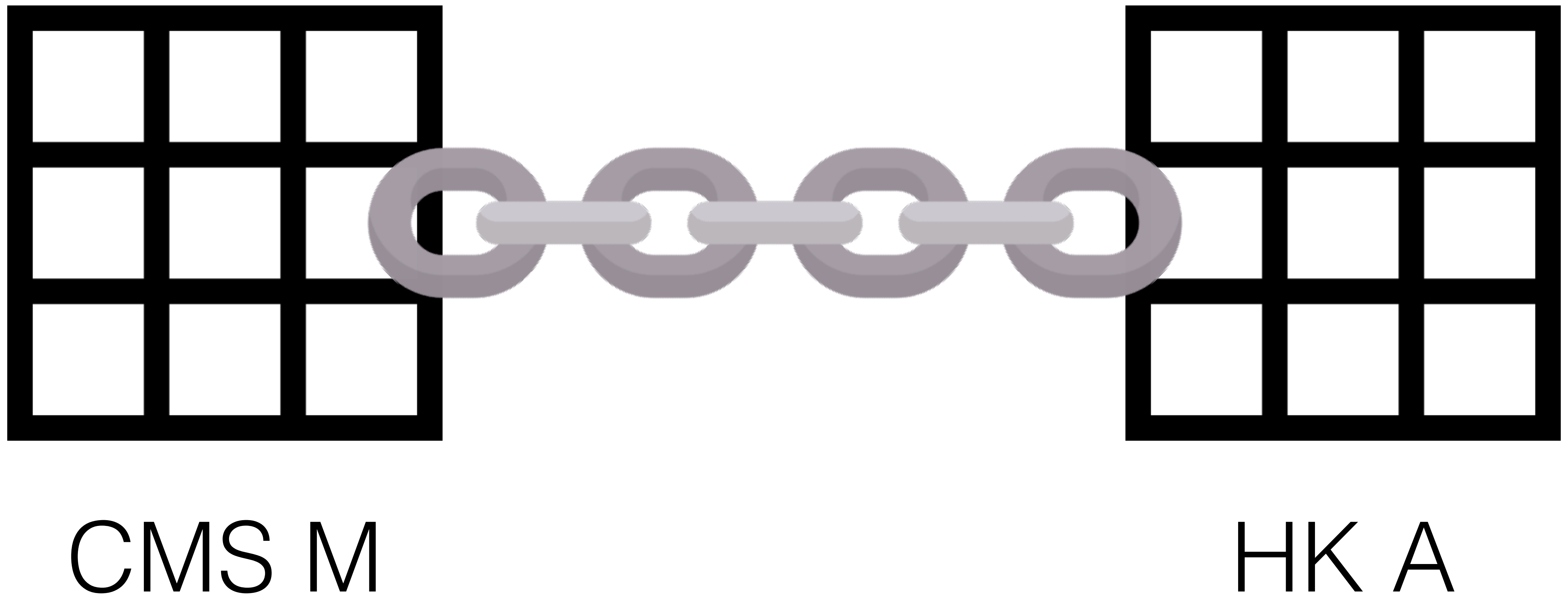


Can we do better? Yes!

Idea: Use information from an auxiliary sketch!

Count-Keeper

- Hybrid between a CMS and HK
- Detects attacks
  - Flagging mechanism
  - Attacks are less damaging
- Works well in practice
  - Honest setting performance



Future Work

- CFE that **prevent** attacks?
- Other compositions of CPDS?

Probabilistic Data Structures in the Wild: A Security Analysis of Redis

Mia Filić, Jonas Hofmann, **Sam A. Markelon**, Kenneth G. Paterson, Anupama Unnikrishnan*

(CODASPY '25 — Best Paper Award)

* Alphabetical Ordering Used

Redis and RedisBloom

RedisBloom Details

- Open Source
- Widely Used
- Six PDS (We examine four)
  - Bloom Filters, Cuckoo filters, **Count-min Sketch, Top-K (HeavyKeeper)**
  - HyperLogLog, T-Digest
- Use MurmurHash2 with fixed seeds

Redis Security Model

“…it’s totally insecure to let **untrusted clients access the system**, please protect it from the outside world yourself”

“an **attacker might insert data** into Redis that **triggers pathological (worst case) algorithm complexity on data structures** implemented inside Redis internals”

Our Attacks

- Ten different attacks against the four CPDS we consider
  - **One against CMS and three against HK**
- MurmurHash2 family has fast inversion algorithms!
  - Target hash h and seed s, can generate arbitrarily many x s.t. h = hash(s,x).
  - Due to ASCII formatting constraints need to try ~16 inversions to find a collision
  - **Upshot for CFE: Find cover sets very fast!**

Use Hash Inversions!

- We have invertible MMH2 in Redis
- Find cover set using inversions!
- Say that we have target x
  - h_1(x)=25,h_2(x) = 278…
  - Simply compute y_1=mmh2_inverse(25,1), y_2=mmh2_inverse(278,2)…
- Eliminates our exhaustive search

```python
def mmh64A_inverse(h: int, seed: int) -> int:
    """Calculate a one-block inverse of an element using MurmurHash64A

    Args:
        h (int): Hash value to invert
        seed (int): Seed value for the hash

    Returns:
        int: Pre-image for h
    """
    # hashing constants
    m = 0xc6a4a7935bd1e995
    # Multiplicative inverse of m under % 2^64
    minv = 0x5f7a0ea7e59b19bd
    r = 47

    h = uint64(h ^ (h >> r))
    h = uint64(h * minv)
    h = uint64(h ^ (h >> r))
    h = uint64(h * minv)

    hforward = uint64(seed ^ (8 * m))
    k = uint64(h ^ hforward)

    k = uint64(k * minv)
    k = uint64(k ^ (k >> r))
    k = uint64(k * minv)

    return k
```

CMS Overestimation Attack

ϵ, δ (m, k)	Ours	[24]
$2.7 \times 10^{-3}, 1.8 \times 10^{-2}$ $(1024, 4)$	66.85	8533.32
$6.6 \times 10^{-4}, 1.8 \times 10^{-2}$ $(4096, 4)$	61.11	34133.36
$2.7 \times 10^{-3}, 3.4 \times 10^{-4}$ $(1024, 8)$	124.22	22264.72
$6.6 \times 10^{-4}, 3.4 \times 10^{-4}$ $(4096, 8)$	128.8	89058.72

Table 1: Experimental number (average over 100 trials) of equivalent *MurmurHash2* calls needed to find a cover for a random target x. We compare the average to the expected number of *MurmurHash2* calls needed in the attack of [24], namely kmH_k.

Implement attack from CCS '23 paper far more efficiently!

HK Attacks

- Very efficiently cause frequent elements to "disappear" (CCS '23)
- Overestimation attacks due to being able to efficiently find fingerprint collisions
- DoS the entire structure
  - Pre-compute elements that map to every counter in the structure
  - Insert them ~100 times each in succession
  - Any subsequent insertions are never recorded

Countermeasures for RedisBloom

- PRF switch for Bloom filter and Cuckoo Filter
- Recall — no provably secure CFE that prevents attacks
  - Suggestion: use Count-Keeper with a PRF

Future Work

- Other PDS Suites
  - Google Big Query and Apache Spark
- Extend Provable Security Work to Deployed Variants
- Educate Developers about PDS in Adversarial Environments
- Safe-by-default PDS Libraries

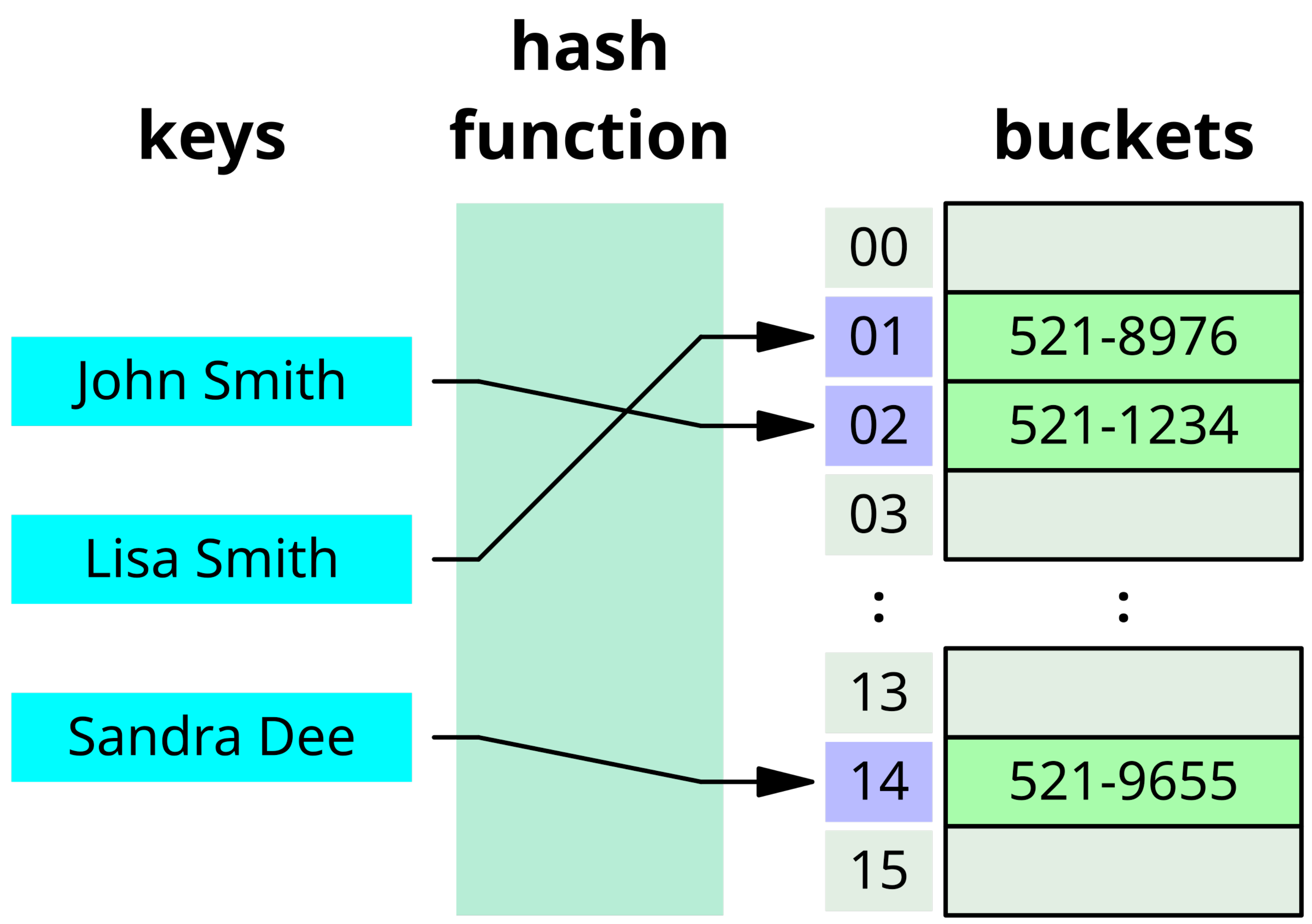
Probabilistic Skipping-Based Data Structures with Robust Efficiency Guarantees

Moritz Huppert, **Sam A. Markelon**, Marc Fischlin*

(In Submission: CCS '25)

* Alphabetical Ordering Used

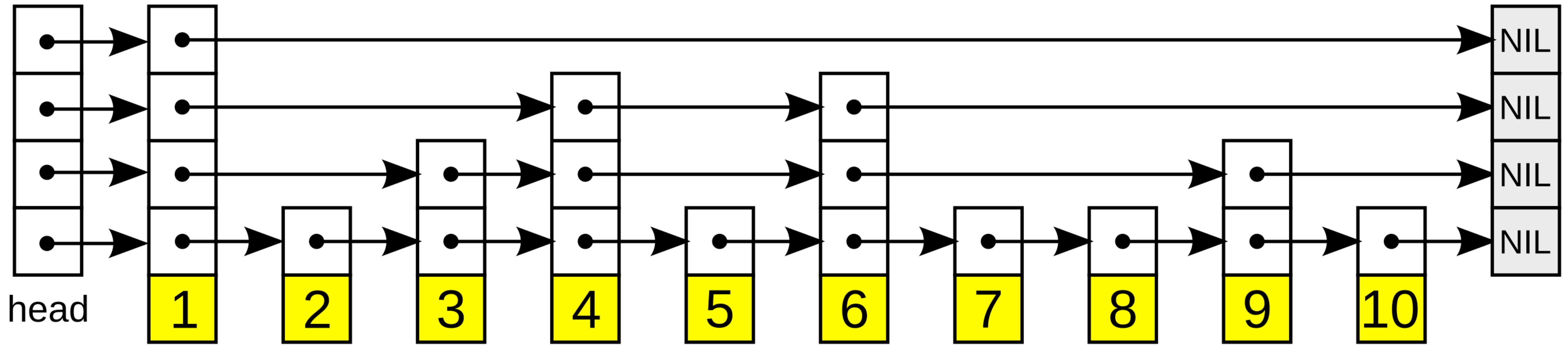
Probabilistic Skipping-Based Data Structures (PSDS)



https://en.wikipedia.org/wiki/Hash_table#/media/File:Hash_table_3_1_1_0_1_0_0_SP.svg

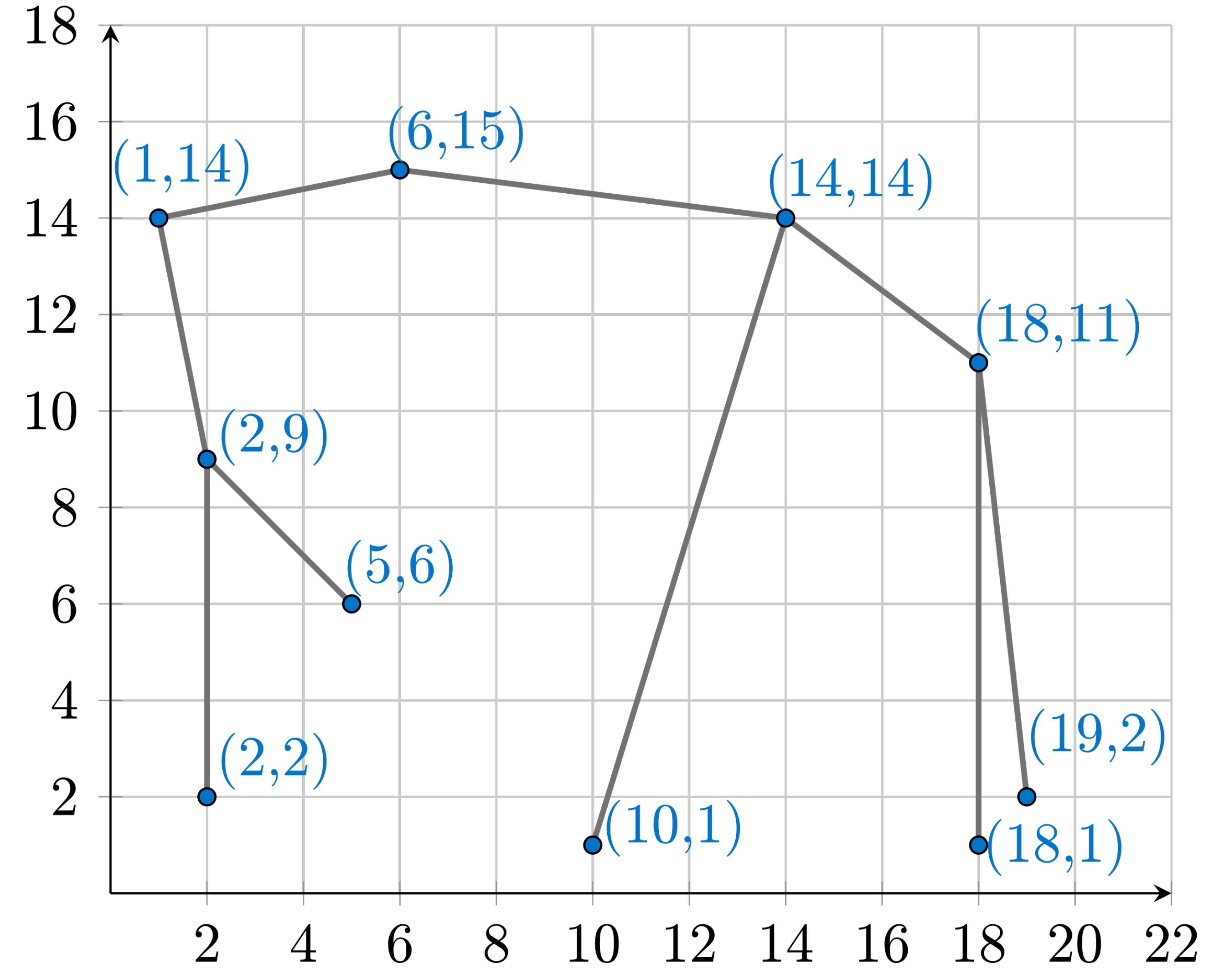
Hash table

Skip list



https://en.wikipedia.org/wiki/Skip_list#/media/File:Skip_list.svg

Treap



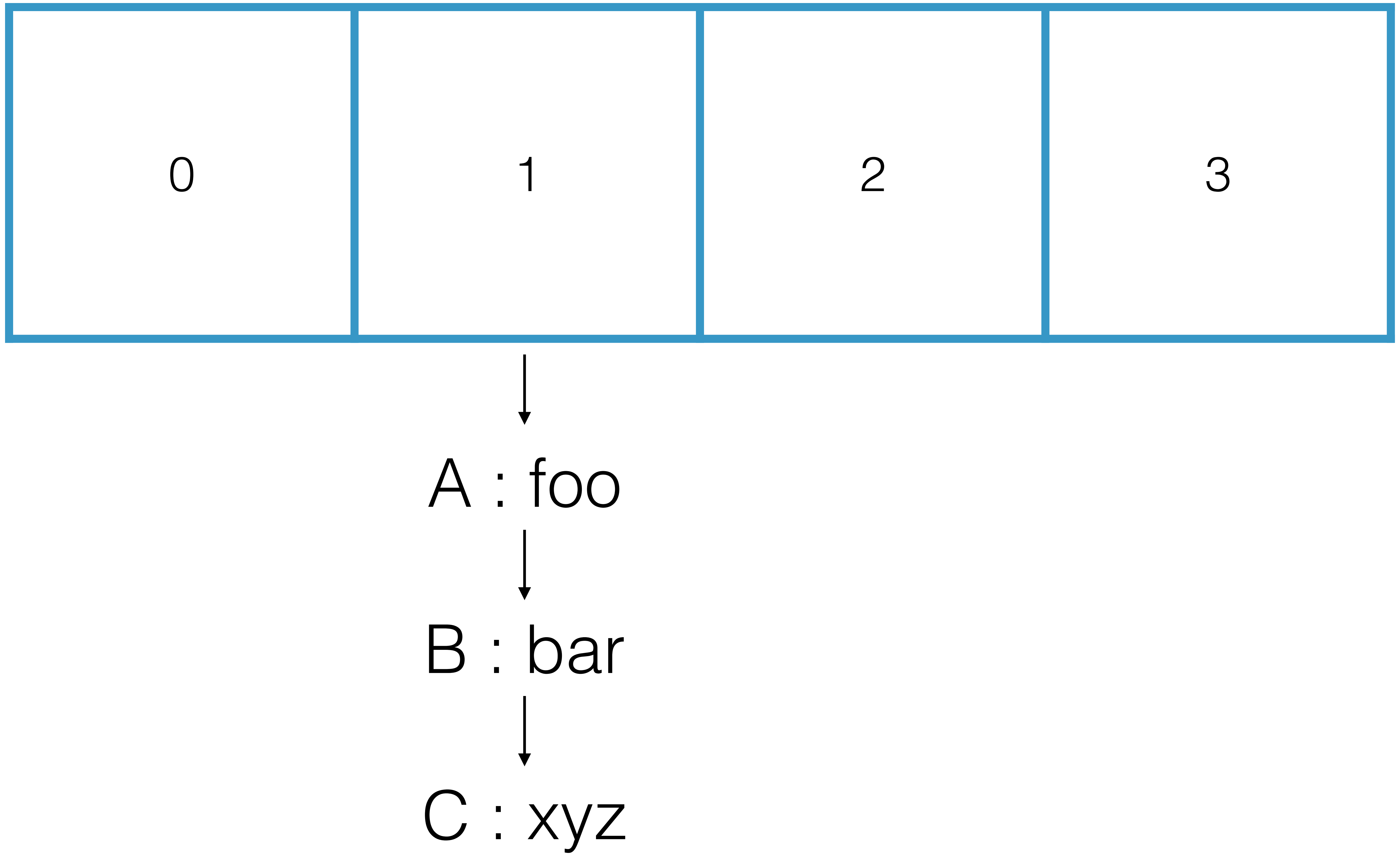
https://en.wikipedia.org/wiki/Treap#/media/File:Treap.svg

Why care about PSDS?

- Fast average-case search
  - Dominates update and deletion operation
- What about worst-case runtime?
- We are in the average case with high probability!
  - $\Pr[\textbf{search cost} \geq \epsilon(\textbf{average-case search cost})] \leq \delta$
  - *Under non-adversarial assumptions**

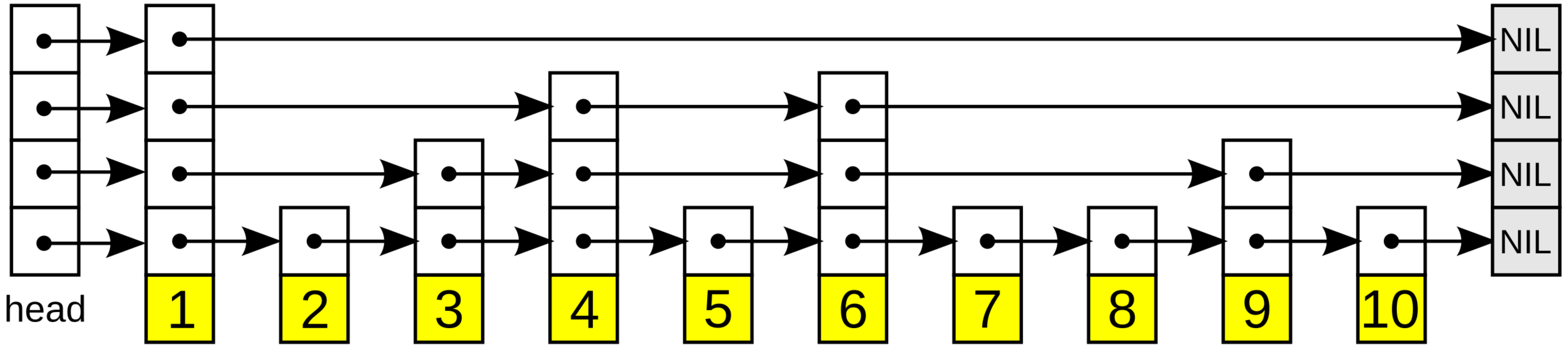
Recall: Hash Flood DoS Attacks

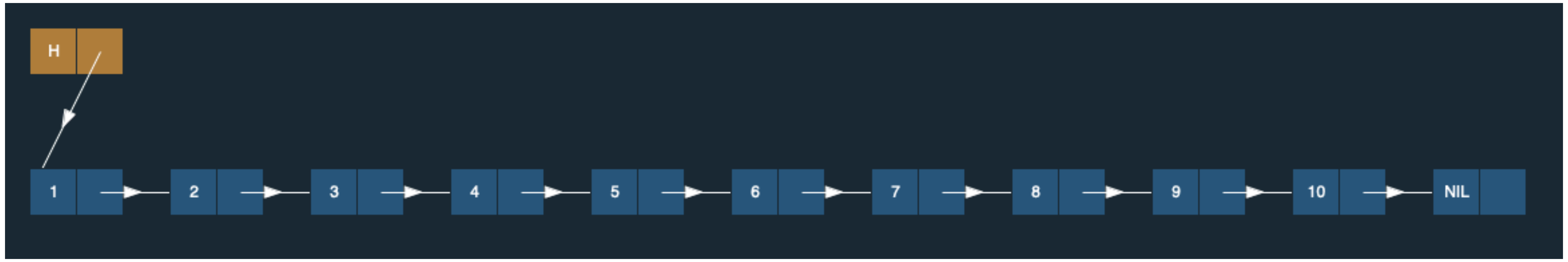
```
hash(A) = 1
hash(B) = 1
hash(C) = 1
      .
      .
      .
```



Insertion of n elements ~ $O(n^2)$

Similar Attacks Against Skip Lists





Motivating a Security Model

- A plethora of attack papers against hash tables
  - Some of these exploit timing side channels
  - Limited attacks for skip list and treaps
  - Some countermeasures explored
  - No real attempt to formalize a security model
- We consider the strongest adversary
  - Can perform any sequence of operations (wrt to some budget)
  - Has access to the internals of the structure at all times

Conserve Target Properties of the DS

- Want to conserve fast search operation
  - Entirely determined by the representation
- Known "non-adaptive" bounds
  - We care about **longest** search time
  - Maximum bucket population for HT
  - Maximum search path length for skip list and treap
- Adversary wins in our game if the measured property after their execution **exceeds** the non-adaptive bound by more than some limit

HT Maximum Search Path: $\phi(D, \text{repr})$

```
e ← 0
for i ← 1 to m
    ℓ ← length(T[i])
    if ℓ > e
        e ← ℓ
return e
```

(a) The HT Maximum Search Path function $\phi : \mathcal{D} \times \{0,1\}^* \rightarrow \mathbb{R}$. The function iterates through all m buckets, returning the bucket with the greatest population, which is equivalent to the longest search path in the table.

TR Maximum Search Path: $\phi(D, \text{repr})$

```
return φ^rec(T.root, 0)
```

$\phi^{\text{rec}}(n, e)$

```
if n = null then
    return
e1 ← φ^rec(n[2], e + 1)
e2 ← φ^rec(n[3], e + 1)
return max(e1, e2)
```

(b) The TR Maximum Search Path function $\phi : \mathcal{D} \times \{0,1\}^* \rightarrow \mathbb{R}$. The function performs an in-order traversal for all elements $d \in D$, returning the longest search path cost among them.

AAPC Security Model

Adaptive Adversary Property Conservation

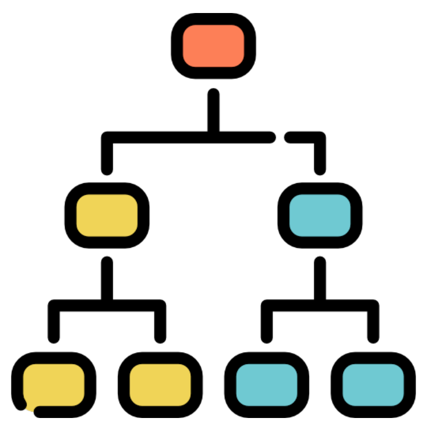
PSDS[🔑]:

insert

delete

query

hash



Maximise Search Path Cost

search time() >> expected search time()

Preserve the Expected Search Path Cost for the "Worst" Search

Probability that the adversary can make a search path cost deviate far from the expected search path cost is small.

Definition 5.1 ($(\phi, \beta, \epsilon, \delta, t)$-Conserved). We say a skipping-based probabilistic data structure Π is $(\phi, \beta, \epsilon, \delta, t)$-conserved if the advantage of an AAPC-adversary $\mathcal{A}$ running in time t is less-than-or-equal to δ for some property function ϕ, some target bound β, some $\epsilon \in \mathbb{R}, \epsilon > 0$, and some $\delta \in [0, 1)$. More precisely, we say the structure is $(\phi, \epsilon, \beta, \delta, t)$-conserved iff,

$$\mathbf{Adv}^{\mathrm{aapc}}_{\Pi,\phi,\beta,\epsilon}(\mathcal{A}) = \Pr[\mathbf{Exp}^{\mathrm{aapc}}_{\Pi,\phi,\beta,\epsilon}(\mathcal{A}) = 1] \leq \delta$$

Towards Robust Structures

- **No deletions**
  - Replicate functionality by marking elements deleted — “lazy” deletion
  - Prevents trivial attacks, aligns with usual operational parameters, can be overwritten by fresh insertions
- **No choosing how or where elements are inserted**
  - Hash table: PRF instead of hash function
  - Skip list: localized “swapping” mechanism
  - Treap: inherent robustness!

Robust Hash Table

- Lazy deletions + PRF
  - Lazy deletions prevent trivial attacks
  - PRFs prevent hash flood attack
- Ball-in-bins average case target
  - n = b
- Tradeoff between space and robustness

$\epsilon = 1 \times s = 3\frac{\log b}{\log \log b}$

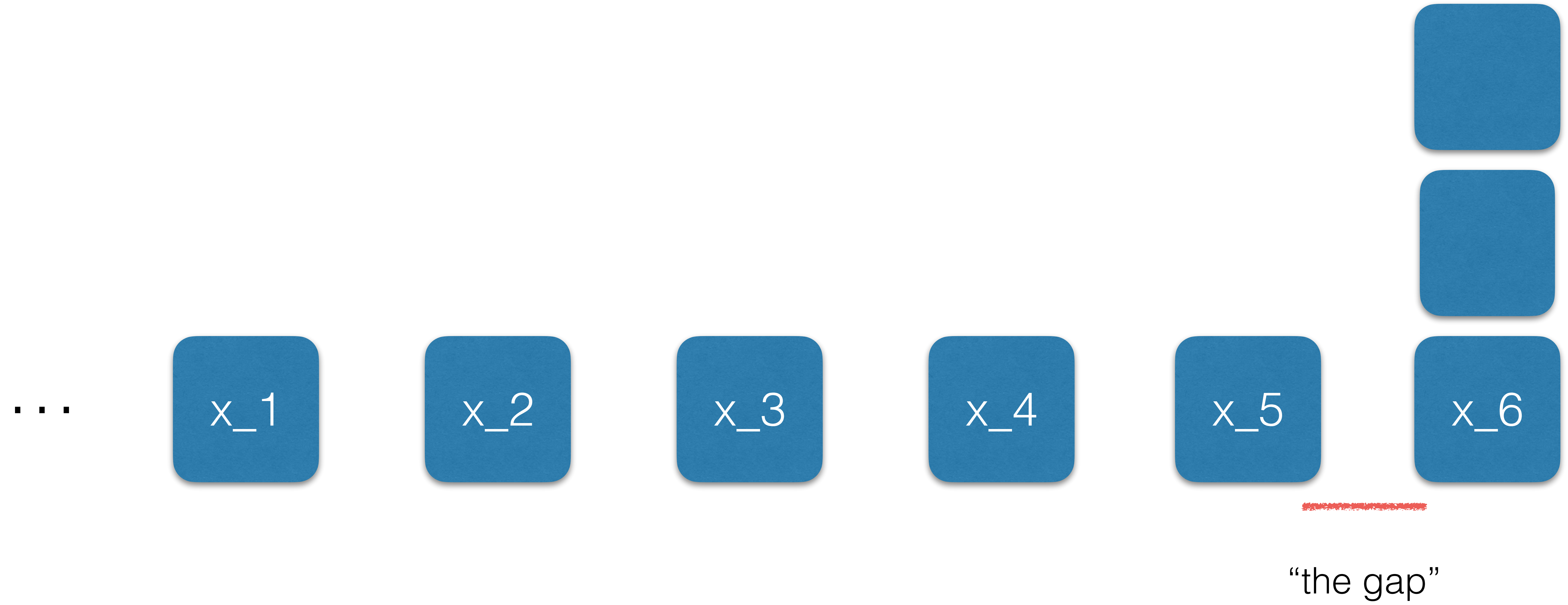
$\leq \frac{1}{b} +$ "PRF advantage"

$b = n = 2^{32}$

$\epsilon = 1 \times \approx 21.47$

$\leq \frac{1}{2^{32}} +$ "PRF advantage"

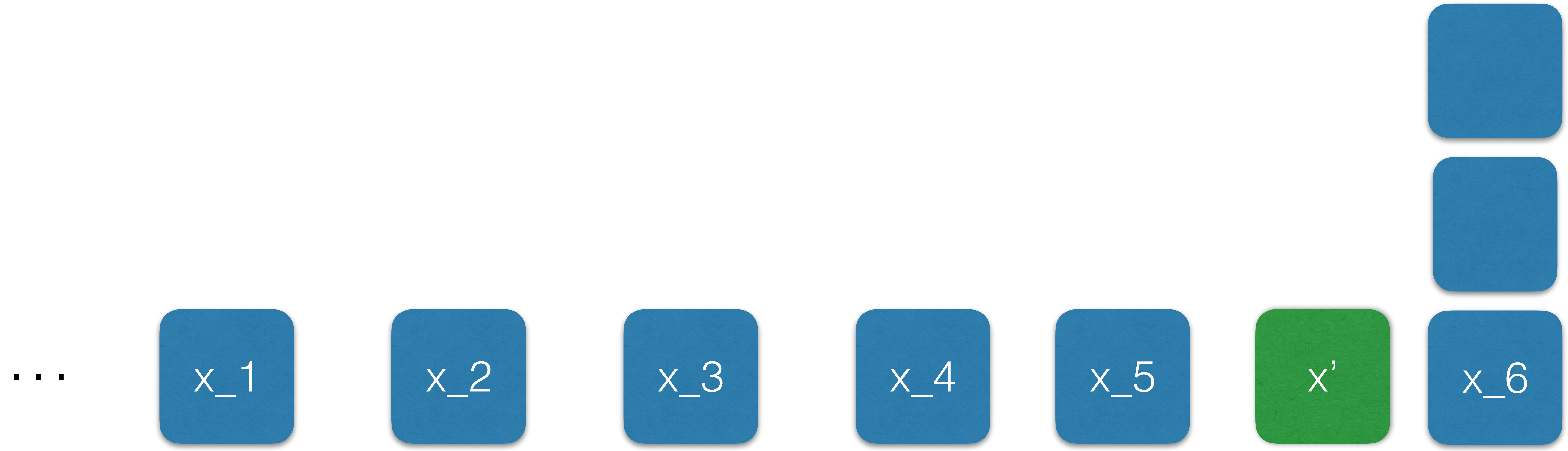
Skip List Gap Attack



Goal: Extend this “flat” run

insert x’ $x_5 < x' < x_6$

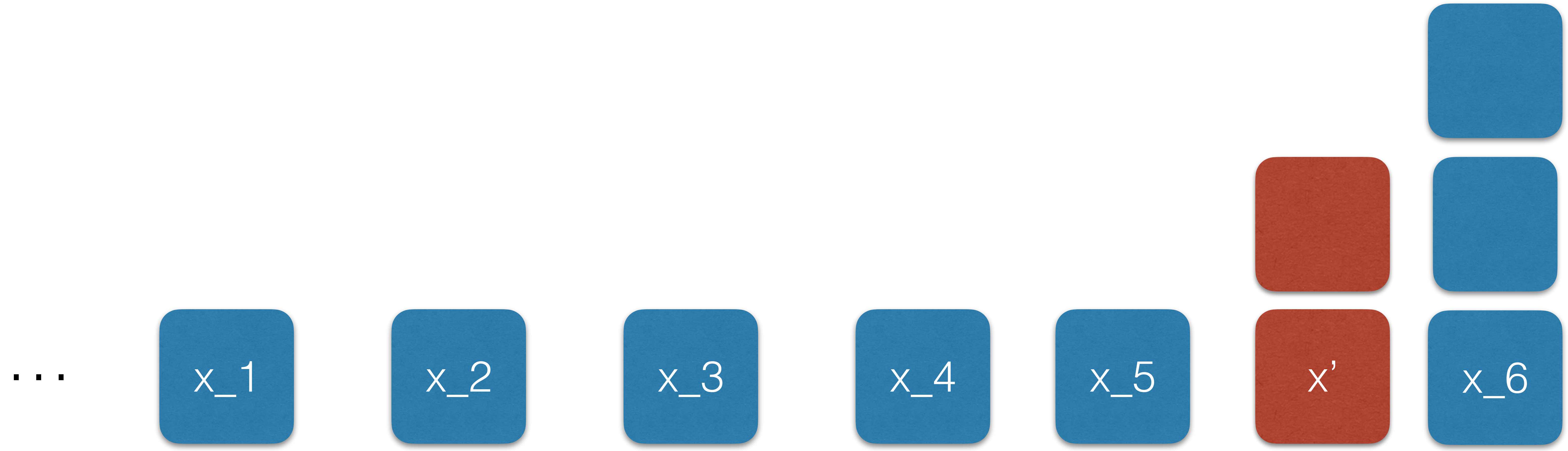
Skip List Gap Attack



Goal: Extend this “flat” run

insert x’ $x_5 < x' < x_6$

Skip List Gap Attack

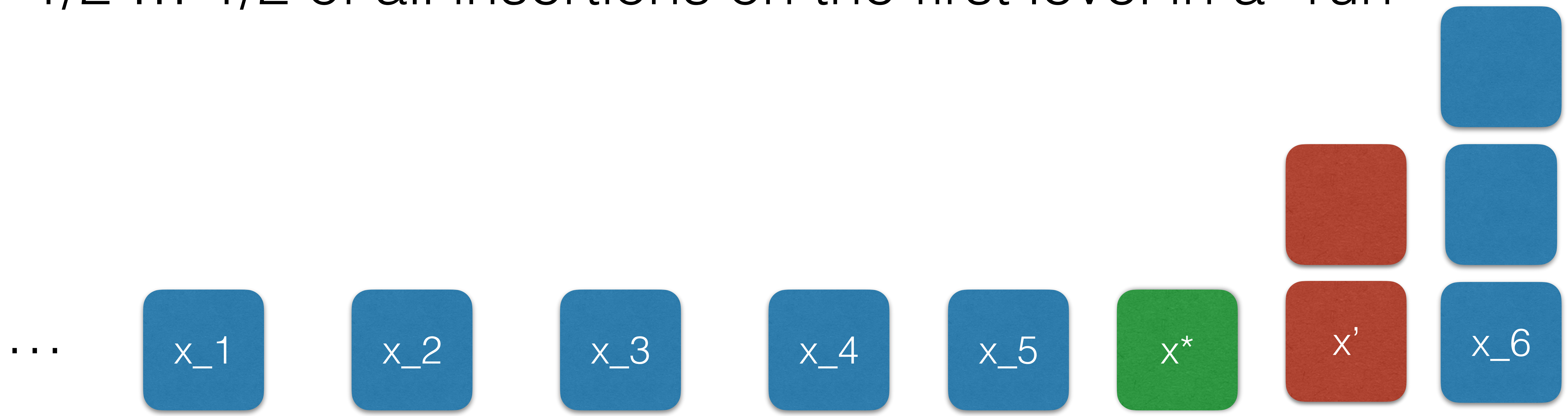


Goal: Extend this "flat" run

insert x' $x_5 < x' < x_6$

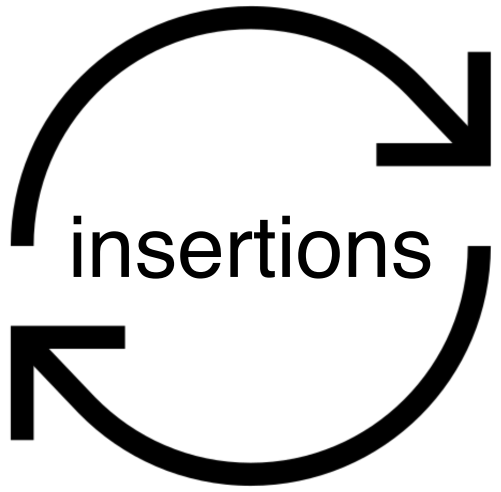
Skip List Gap Attack

p=1/2 … 1/2 of all insertions on the first level in a “run”



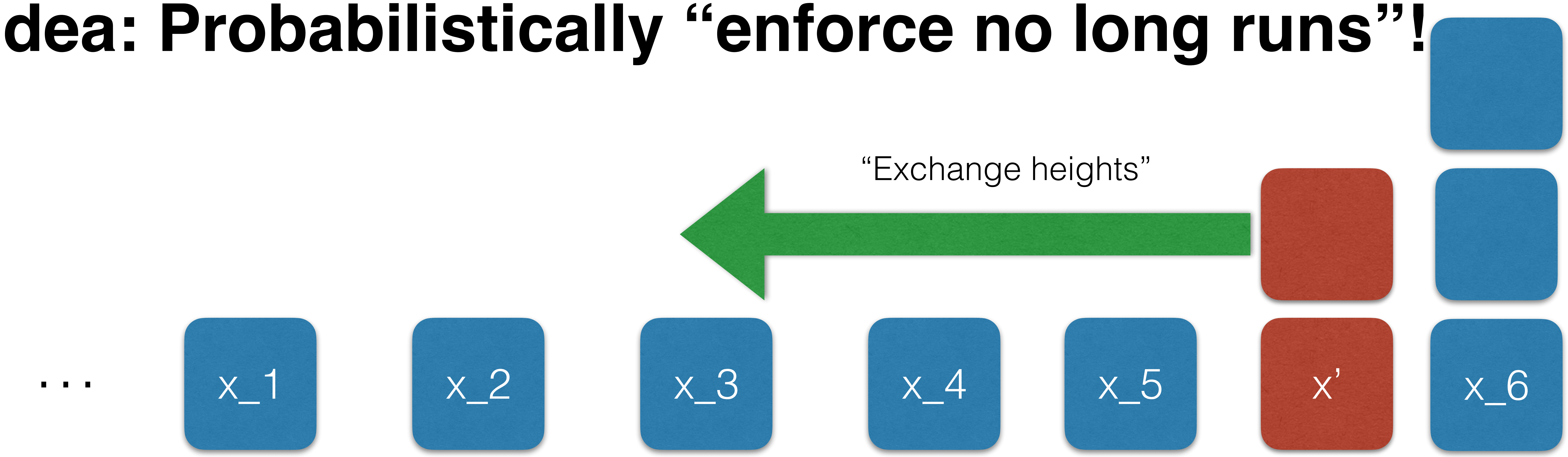
Goal: Extend this “flat” run

insert x* $x_5 < x^* < x'$



Solution: Localized "Swapping"

Idea: Probabilistically "enforce no long runs"!

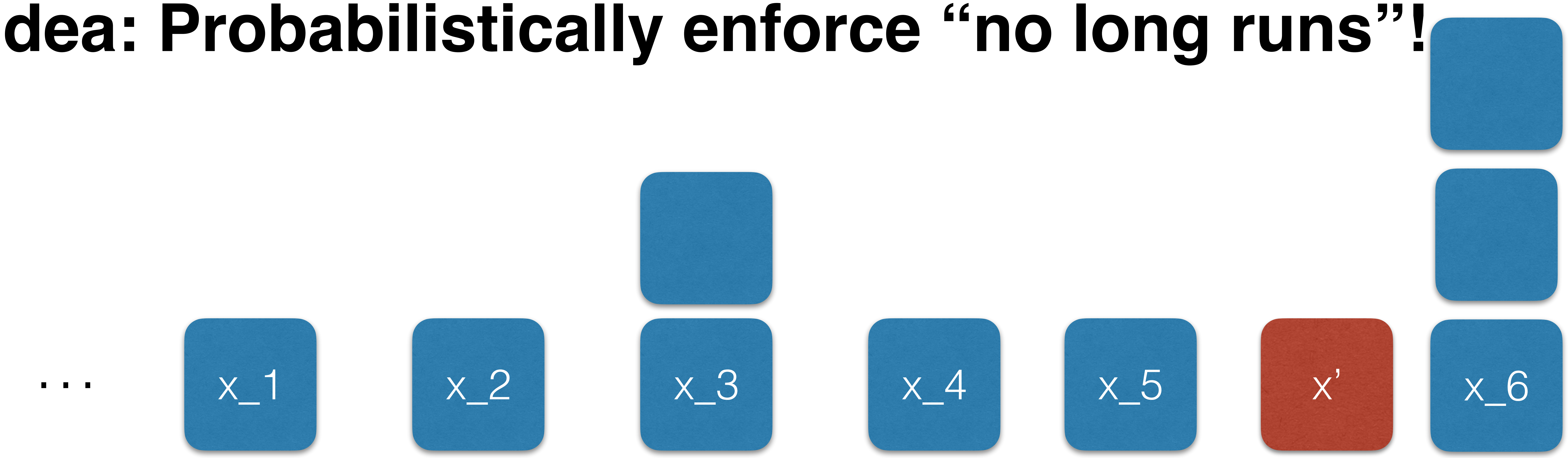


Goal: Extend this "flat" run

insert x' $x_5 < x' < x_6$

Solution: Localized “Swapping”

Idea: Probabilistically enforce “no long runs”!



Solution: Adversary can’t create such a long run before its length halves with high probability

insert x’ $x_5 < x' < x_6$

Robust Skip List

- Lazy deletions + localized "swap"
  - Lazy deletions prevent trivial attacks
  - Swaps prevent long runs
- c log(n) average case target
- Epsilon is "artificially" large

$\epsilon > 0 \times$ $s = a(1+\epsilon)\log(n)$

$\leq e^{(\lambda^* a) - (\epsilon \lambda^* a)} + e^{-\frac{((1-p)a\log_{1/p}(n)-1)^2}{(1-p)(2+(1-p)a\log_{1/p}(n)-1)}}$

$a = \dfrac{2(1+p)}{p}$ and λ^* is the maximal solution $\lambda > 0$ to $(1-p)e^{\lambda} + p(1-p)e^{-\lambda\left(\frac{1}{p}+\frac{a}{2}\right)} + p^2 \leq 1$

$n = 2^{32}, p = \dfrac{1}{2} : a = 6, \lambda^* \approx 0.34$

$\epsilon = 108 \times$ 32

$\leq \approx 6.27 \times 10^{-7}$

$n \to \infty, \epsilon$ is constant

Robust Treap

- Lazy deletions only
  - Lazy deletions prevent trivial attacks
  - Per-insertion randomized priorities prevent creating long branches inherently
- Adaptive adversary still can "attack"
- 2 ln(n) + 1 average case target

$\epsilon > 0 \times s = 2\ln(n) + 1$

$\leq n e^{\frac{-\epsilon^2 H_n}{2(1+\epsilon)}}$, H_n is the nth harmonic number

---

$n = 2^{32}, \epsilon = 5$

$\epsilon = 5 \times \approx 45.36$

$\leq 6.65 \times 10^{-12}$

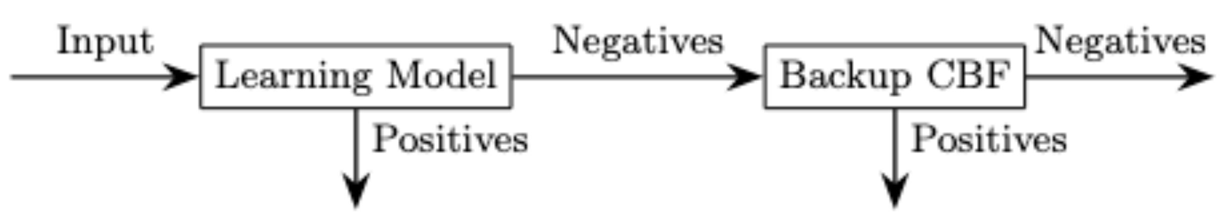
Future Work

- Tighter bounds
- Analyze other PSDS
    - Zip trees, zip-zip trees, skip graphs, randomized meldable heaps, etc.
- Explore other options for handling deletions
    - Localized reinitializations
- Explore other structural properties using AAPC

Future Directions

Vast Ocean of Data Structures

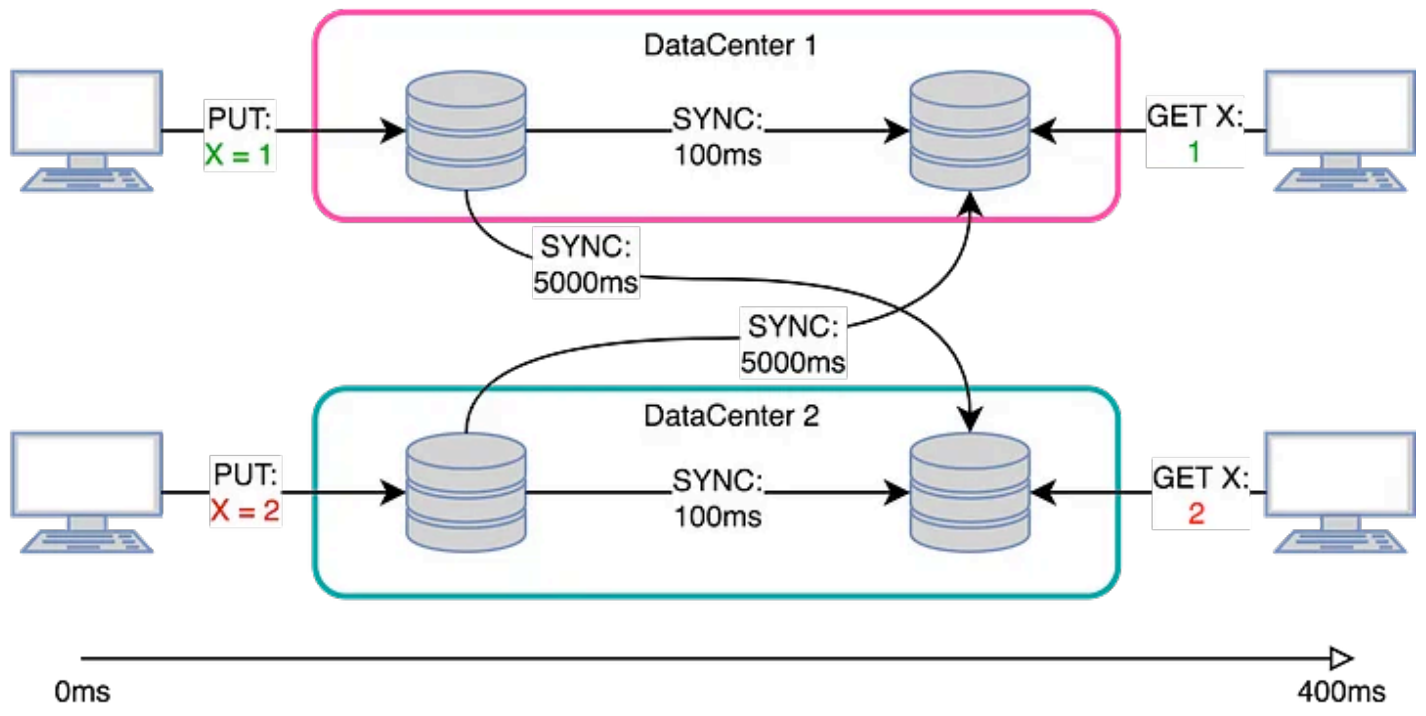
- Compositions of Data Structures
- "Learned" Data Structures
- Conflict Free Replicated Data Types
- Real-world deployments have often not been analyzed for security



Adversary Resilient Learned Bloom Filters

Ghada Almashaqbeh[1], Allison Bishop[2,3] and Hayder Tirmazi[3]

[1] University of Connecticut, ghada@uconn.edu
[2] Proof Trading, abishop@ccny.cuny.edu
[3] City College of New York, hayder.research@gmail.com



https://medium.com/@amberovsky/crdt-conflict-free-replicated-data-types-b4bfc8459d26

On the Insecurity of Bloom Filter-Based Private Set Intersections

Jelle Vos* J.V.Vos@tudelft.nl
Jorrit van Assen* J.S.VanAssen@tudelft.nl
Tjitske Koster T.O.Koster@tudelft.nl
Evangelia Anna Markatou E.A.Markatou@tudelft.nl
Zekeriya Erkin Z.Erkin@tudelft.nl

November 22, 2024

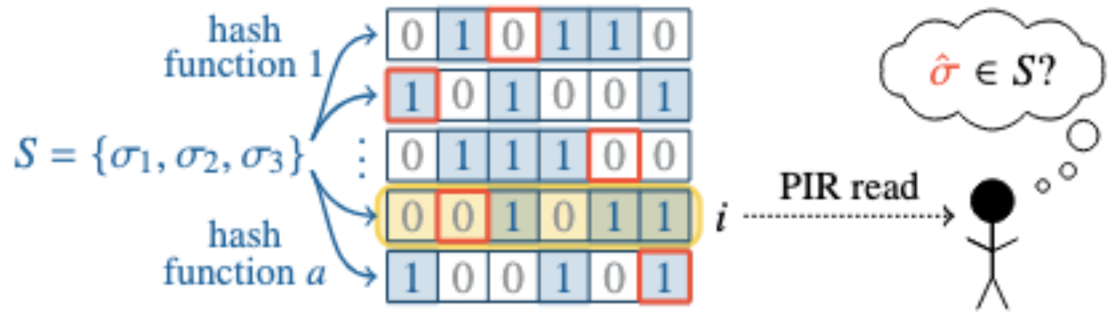
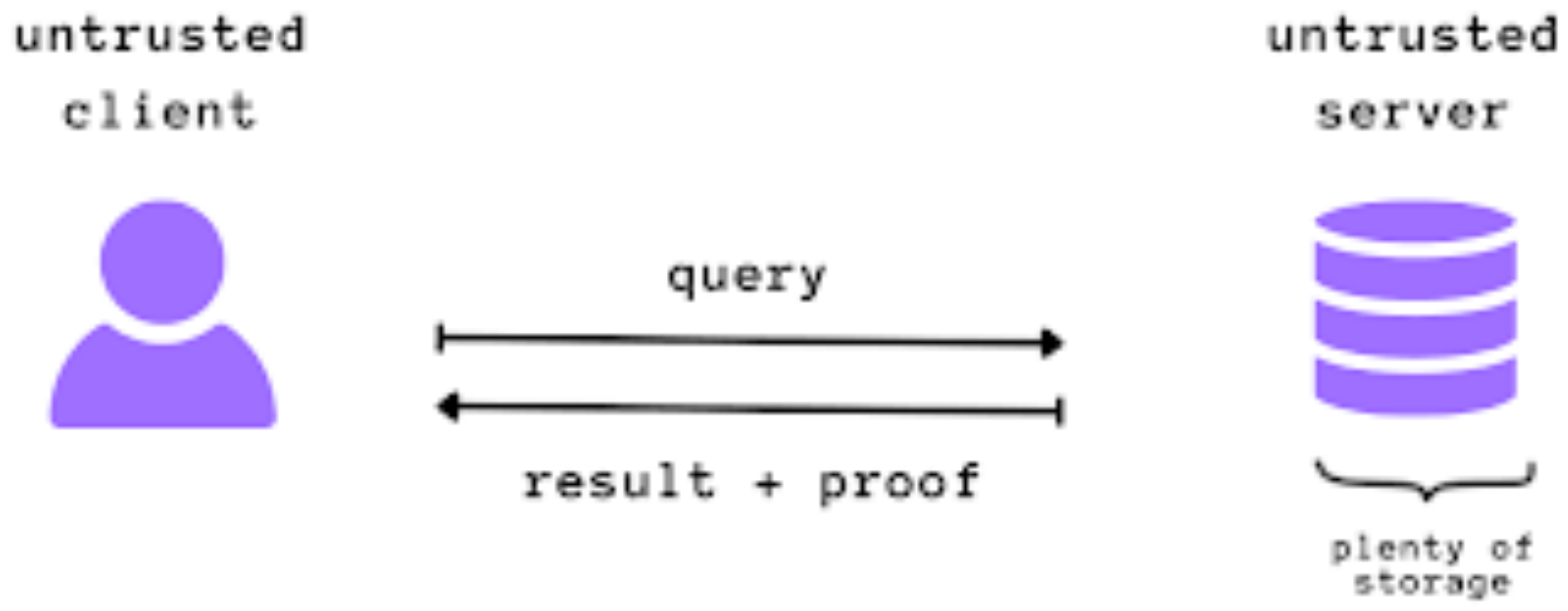
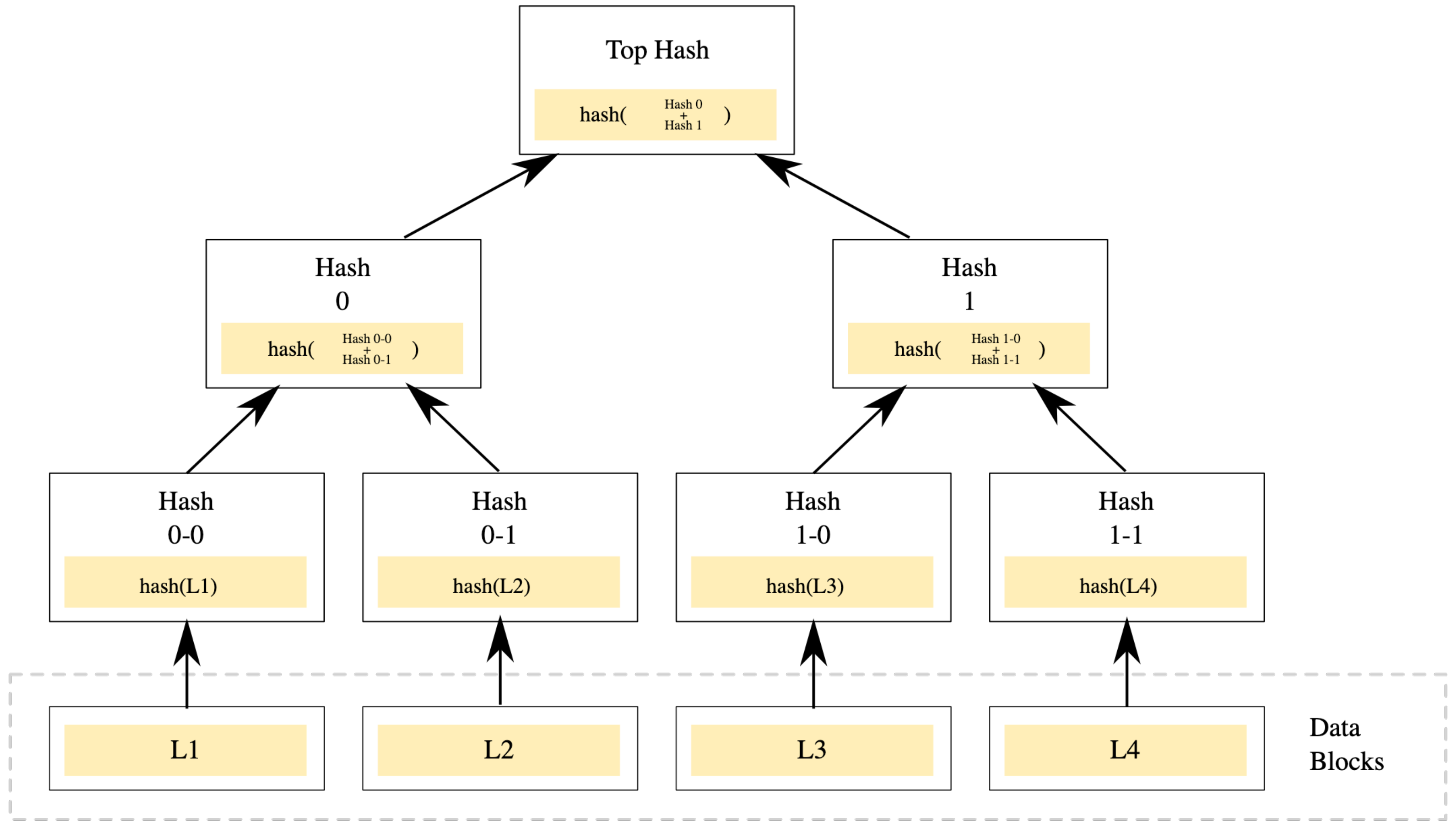


Figure 4: Our data structure for private, approximate set membership with adversarial soundness, when instantiated with a set S consisting of three strings and with $a = 5$ hash functions. We highlight in blue the bits of the data structure that are set, in red the bits that the query string $\hat{\sigma}$ maps to, and in yellow the area covered by the client's PIR read, when the client probes the i-th one-hash-function Bloom filter.

Authenticated Data Structures

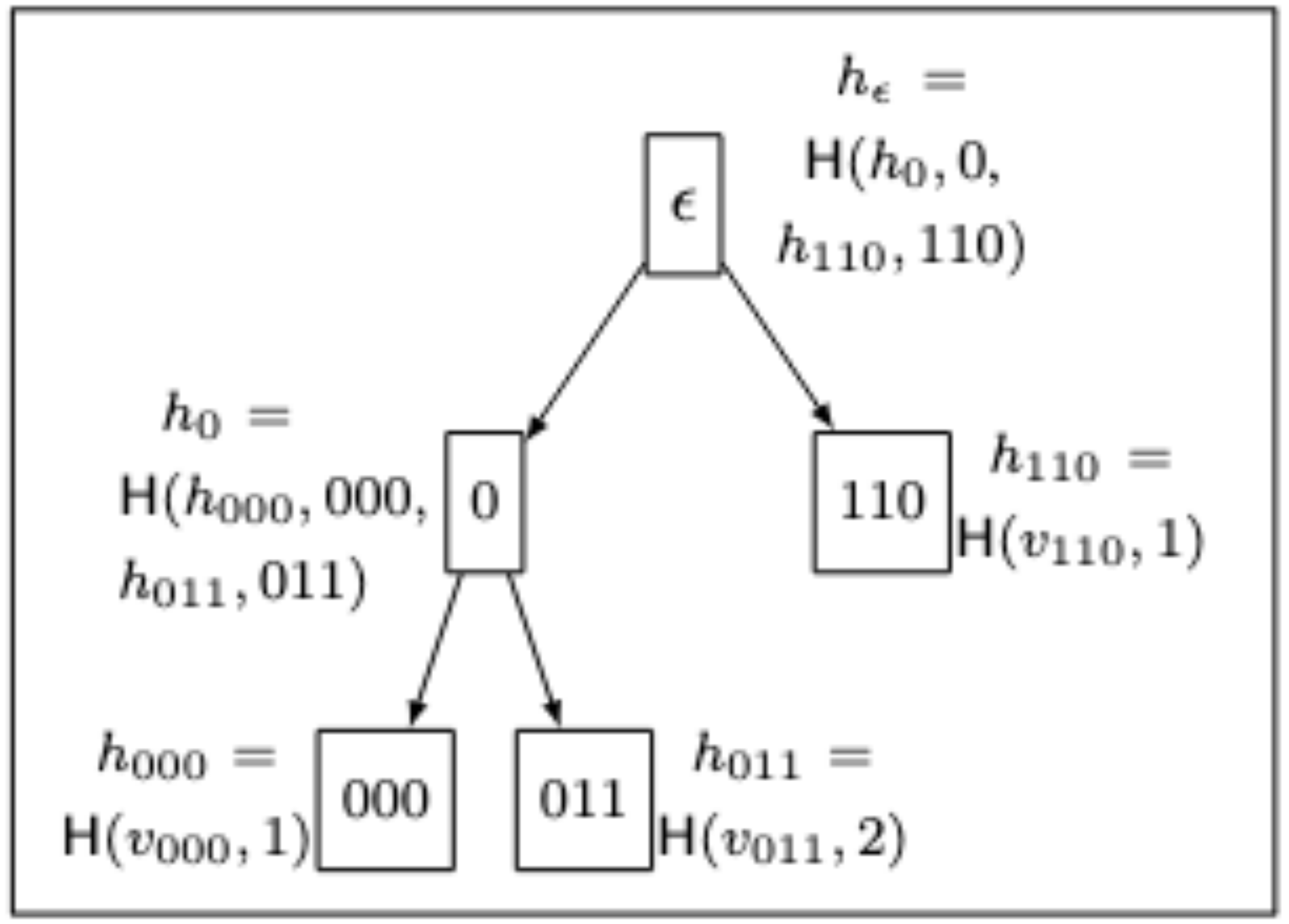


https://mirror.xyz/go-outside.eth/zX1BaGZLHAcQOKdhFnSSM0VW67_-OFCi5ZegGFPryvg



https://en.wikipedia.org/wiki/Merkle_tree#/media/File:Hash_Tree.svg

aoZKS



https://eprint.iacr.org/2023/081.pdf

Homomorphic Merkle Trees

Definition 10 (Generalized hash tree). *Given functions* $h : \mathcal{D} \times \mathcal{D} \to \mathcal{R}$ *and* $f : \mathcal{D} \to \mathcal{R}$, *a* generalized hash tree (T, λ, f, h) *is a* labeled binary tree (T, λ) *such that* **(a)** *for all* $w \in T$, $\lambda(w) \in \mathcal{D}$; **(b)** *for all internal nodes* $w \in T$, $f(\lambda(w)) = h(\lambda(w0), \lambda(w1))$, *where* $w0$ *and* $w1$ *are the left and right children of* w *respectively.*

https://link.springer.com/content/pdf/10.1007/978-3-642-38348-9_22.pdf

Pushing the Boundaries

- Randomized Algorithms
- Machine Learning
- Databases and data processing systems
    - Encrypted databases

Lessons Learned

What is *security?*

Formalism is important.

The real world is messy.

Tradeoffs are unavoidable.

Adversaries adapt — we must too.

Finis

(the end)

Publications and Other Work

Compact Frequency Estimators in Adversarial Environments

CCS '23

Probabilistic Data Structures in the Wild: A Security Analysis of Redis*

Submitted: CODASPY '25

On the Fuzzy Guarantees of Fuzzy Hashing

Soon!

Probabilistic Skipping-Based Data Structures with Robust Efficiency Guarantees

In progress: CCS '25

SoK: On the Security Goals of Key Transparency Systems

On ePrint

Verifiable Summaries to Scale Key Transparency Deployments

Soon!

The DecCert PKI: A Solution to Decentralized Identity Attestation and Zooko's Triangle*

IEEE DAPPS '22

Leveraging Generative Models for Covert Messaging: Challenges and Tradeoffs for "Dead-Drop" Deployments

CODASPY '24

*Best paper award

Thank You! Questions?

Sam A. Markelon
smarkelon@ufl.edu
https://smarky7cd.github.io/