Concurrent Big Data Processing

Data Structures & Semantics

Idit Keidar

Shout Out

1. Fast Concurrent Data Sketches, PPoPP 2020
   Arik Rinberg, Alexander Spiegelman, Edward Bortnikov,
   Eshcar Hillel, Idit Keidar, Lee Rhodes, Hadar Serviansky

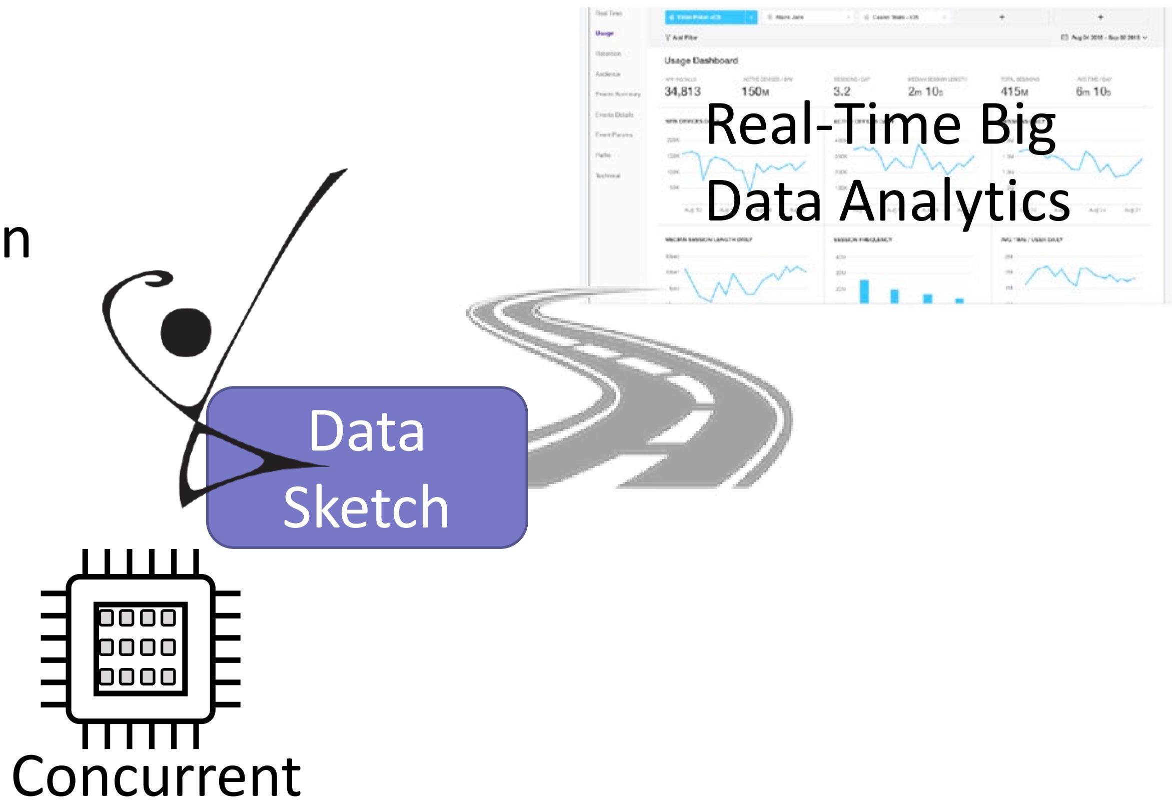
2. Intermediate Value Linearizability, DISC 2020
   Arik Rinberg and Idit Keidar
   (Best Student Paper)

Roadmap

Concurrent data sketches:

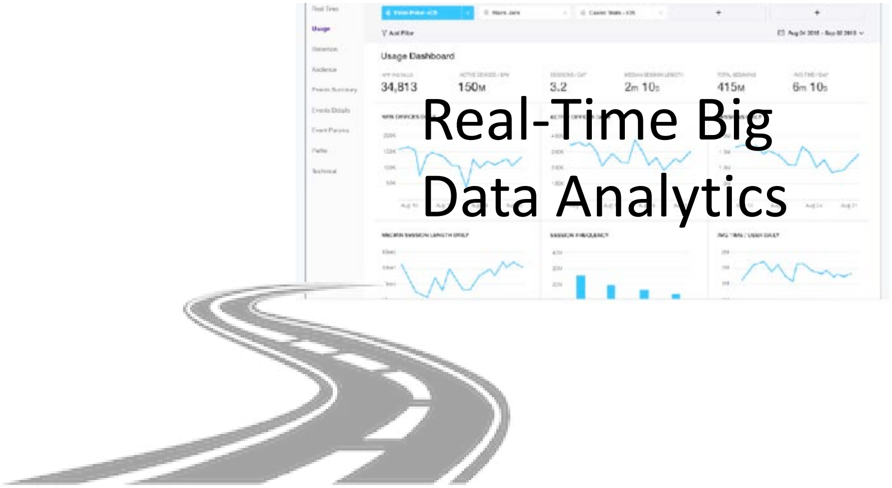
1. Fast implementation

2. Correctness semantics

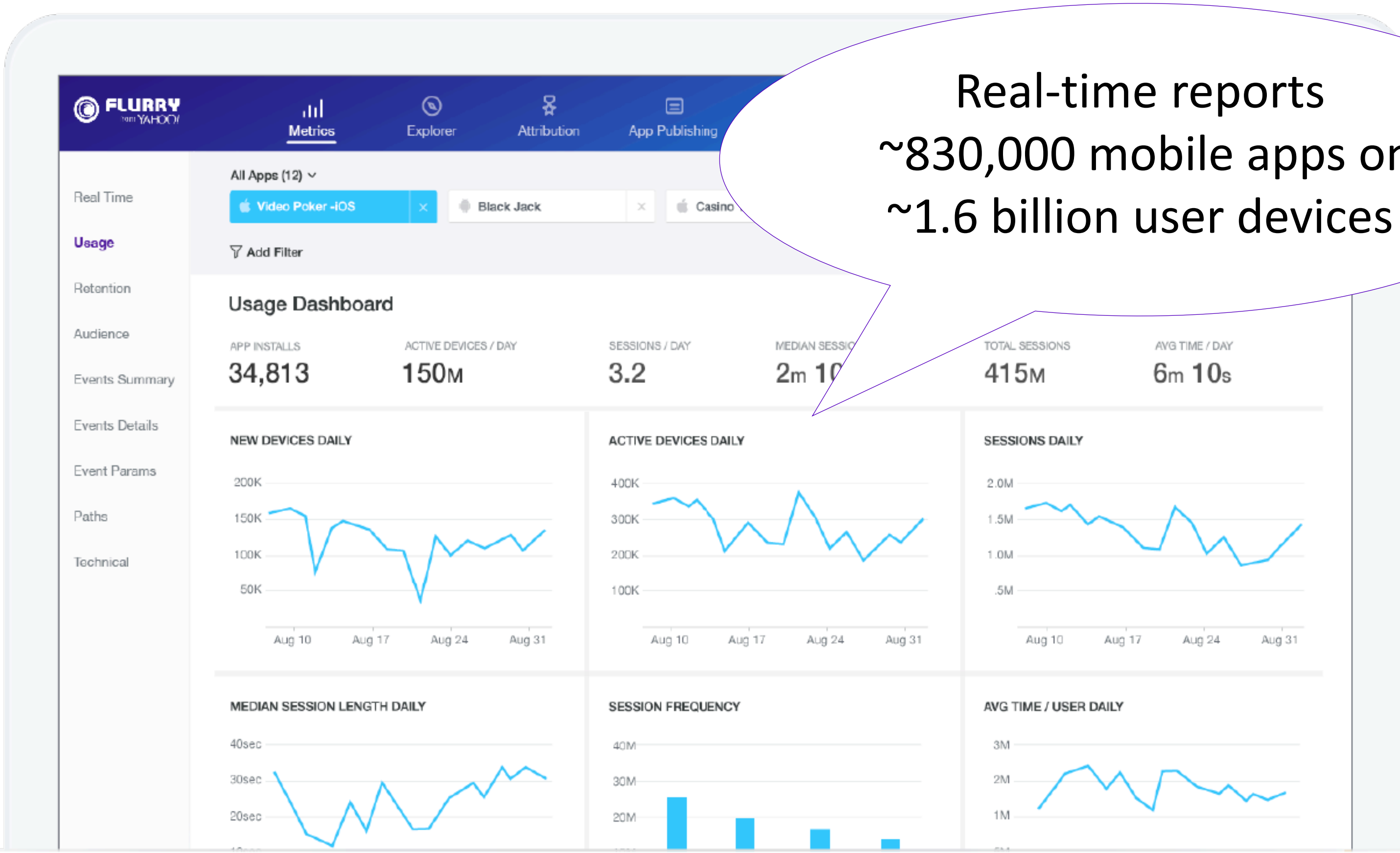


Why New Semantics?

- Amenable to efficient implementation
  - Linearizability is often too costly
- Meaningful
  - Bound sketches' estimation errors
- Leverage what we know about the sequential case
  - Error analysis



Motivation: Massive Real-Time Analytics



Real-time reports
~830,000 mobile apps on
~1.6 billion user devices

ed in Sign in Join

Fast First Analytics Will Simplify Your Life

Published on February 3, 2017

Tripp Smith | Follow
Chief Technology Officer at Clarity Solution Group

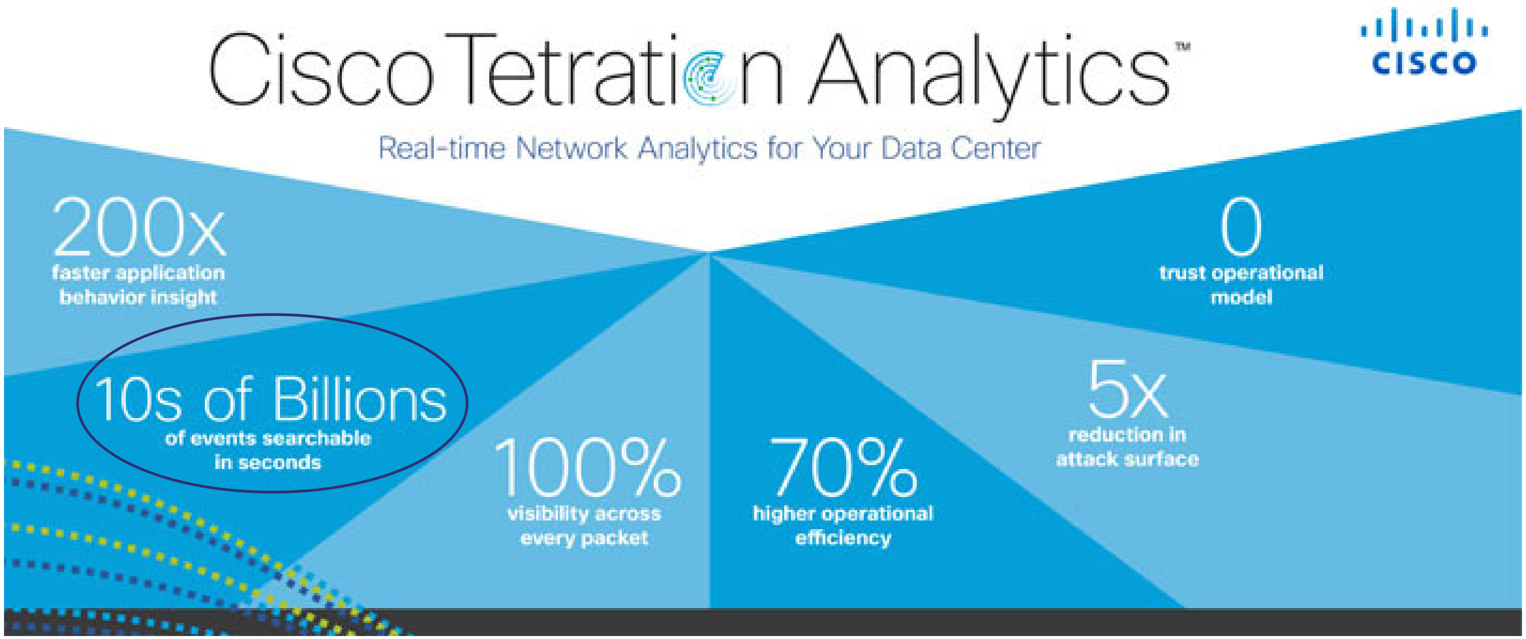
18 0 8

IDC estimates 82% of organizations are in some phase of adopting real-time analytics in the past year. [1] Low latency, "fast first" integration and analytics make managing big data easier ("low latency" and "fast first" here are used to avoid contention surrounding the semantic definition of commonly overused terms streaming or real-time). Capturing event data, generated in real time, in offline storage to process in batches at intervals, overnight, or at the end of the month was never easy. It was possibly a pattern born of scale.

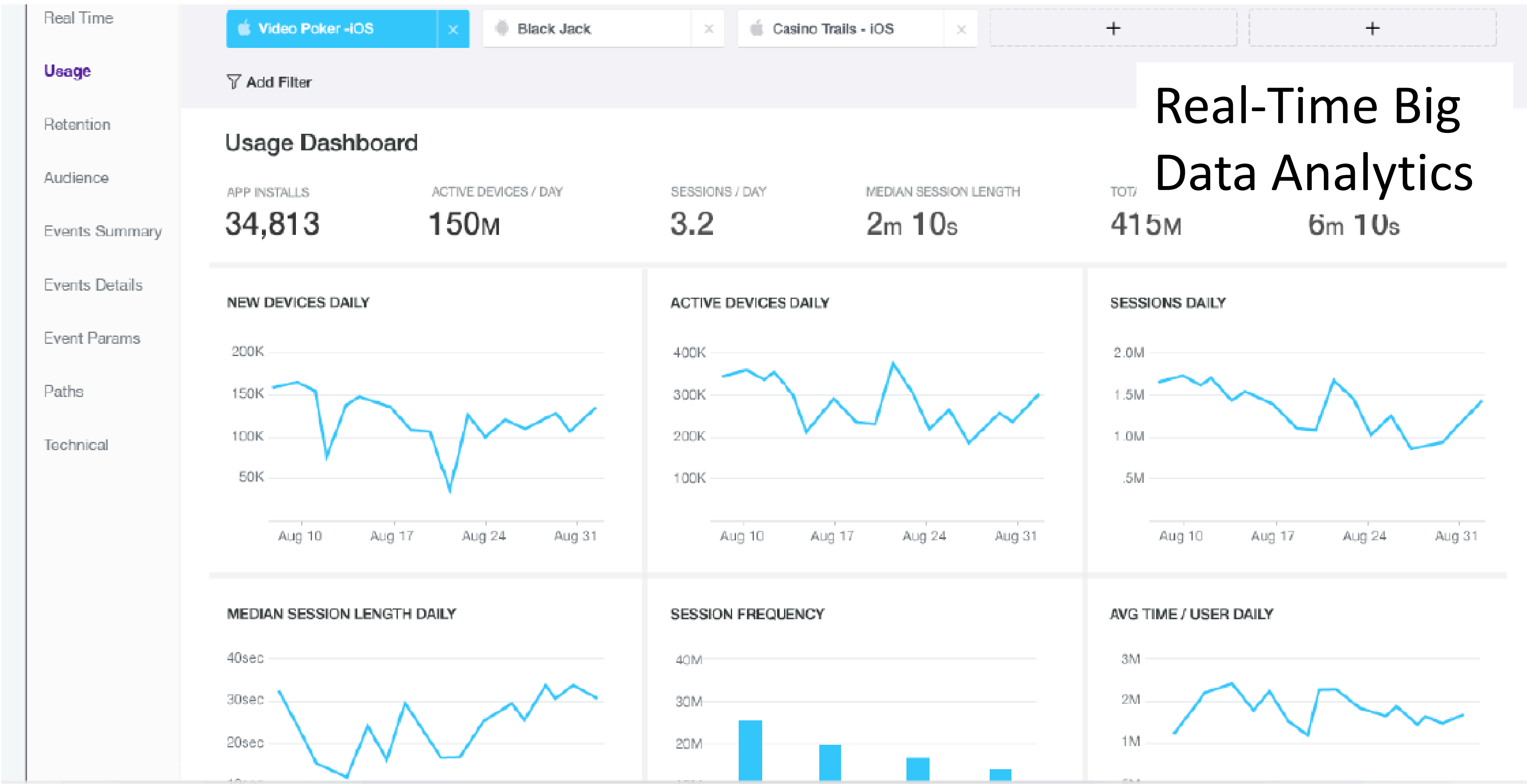
International Data Corporation
Market research company

IDC Analyze the Future

OLAP - Online Analytical Processing Examples

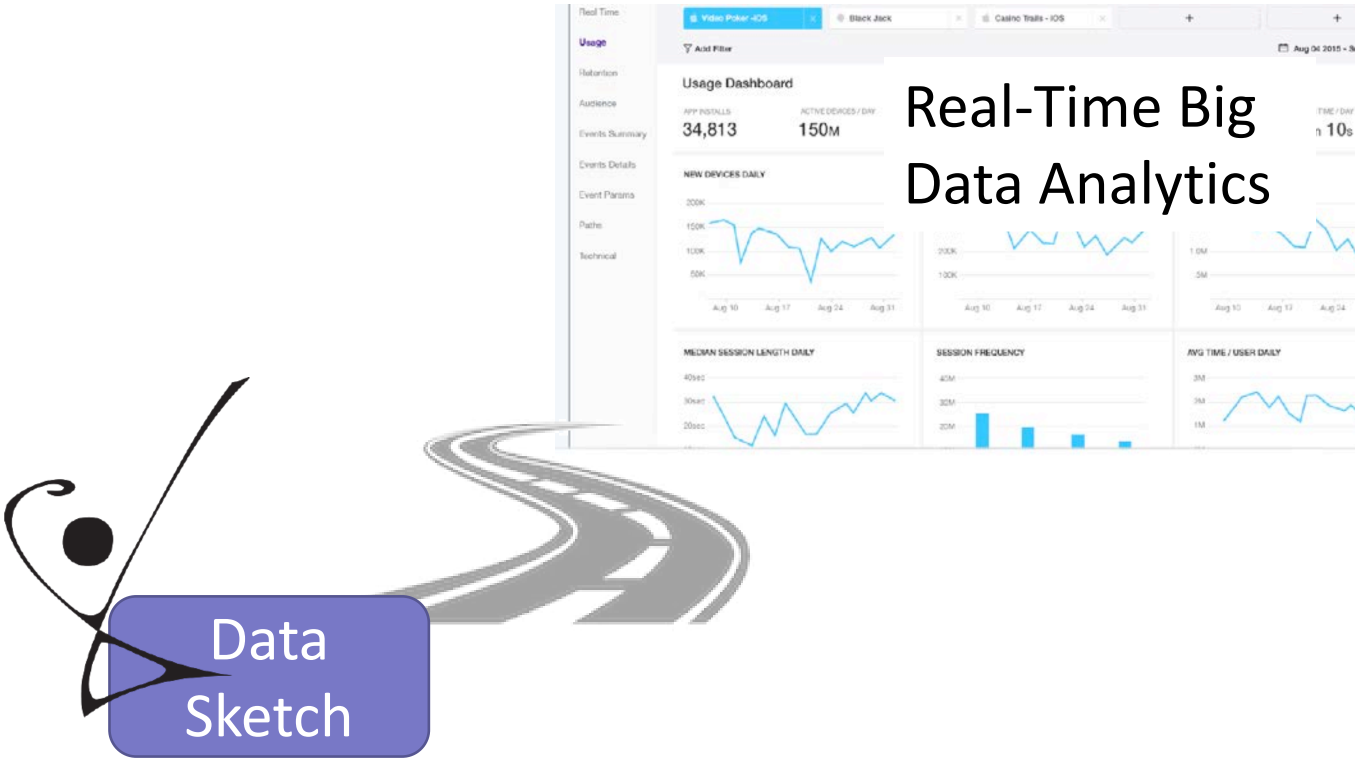


Motivation: Big Data Analytics & Monitoring



Real-Time Big Data Analytics

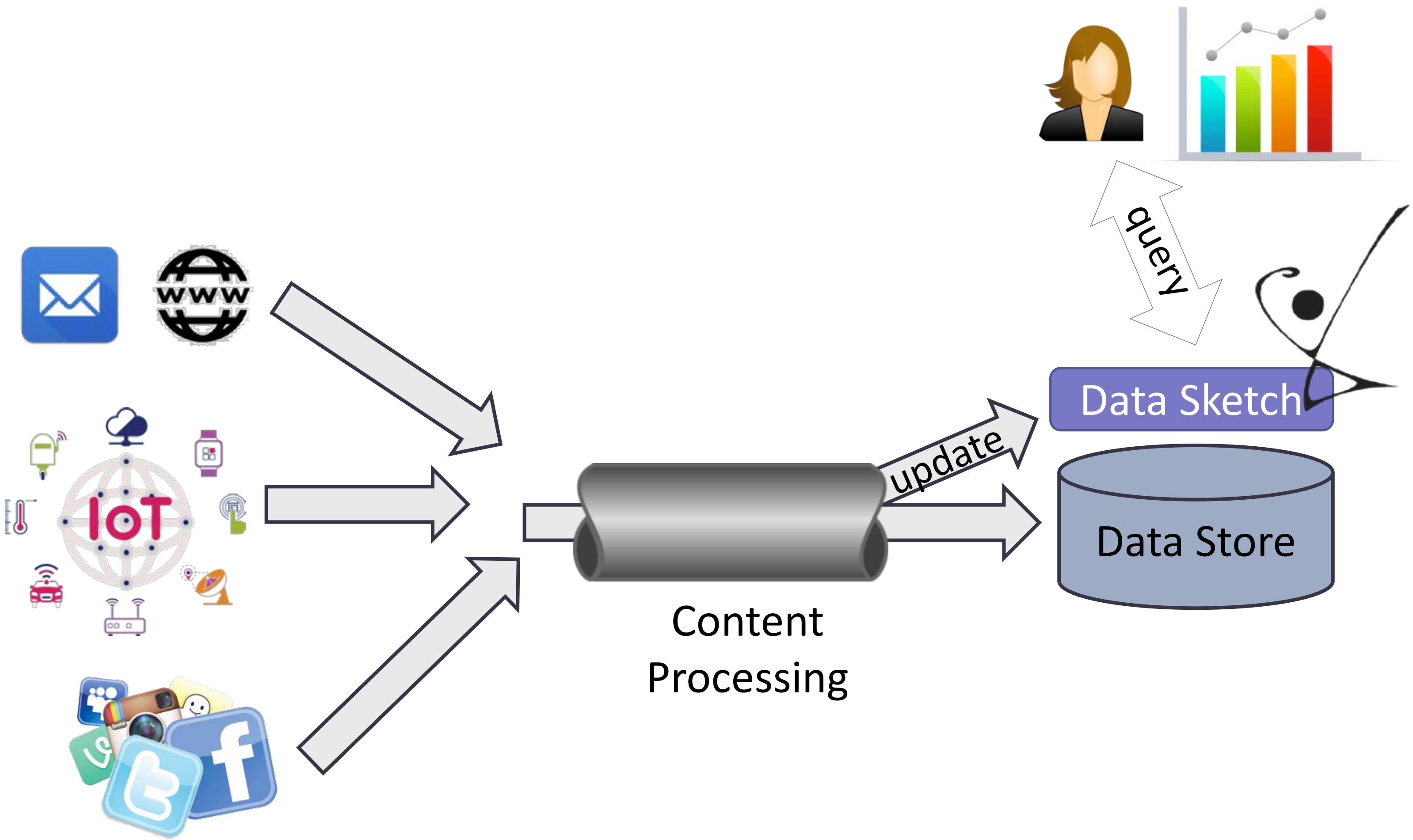
The Tool: Data Sketches



Data Sketches: Lean & Mean Aggregation

- Statistical summary of large stream
- Estimates some aggregate
  - #uniques
  - quantiles
  - heavy-hitters
  - item frequencies
- Fast
- Small memory footprint
- Widely-used

Real-Time Analytics – Where We Fit In

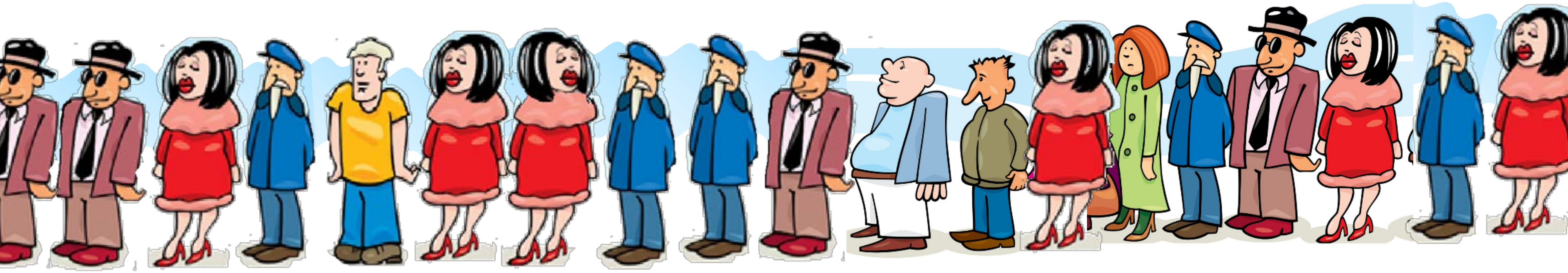


Example: Estimating the Number of Uniques

- E.g., unique visitors to a web page
- How many uniques?

Θ Sketch: Basic Idea

- Hash unique elements into [0,1] uniformly at random



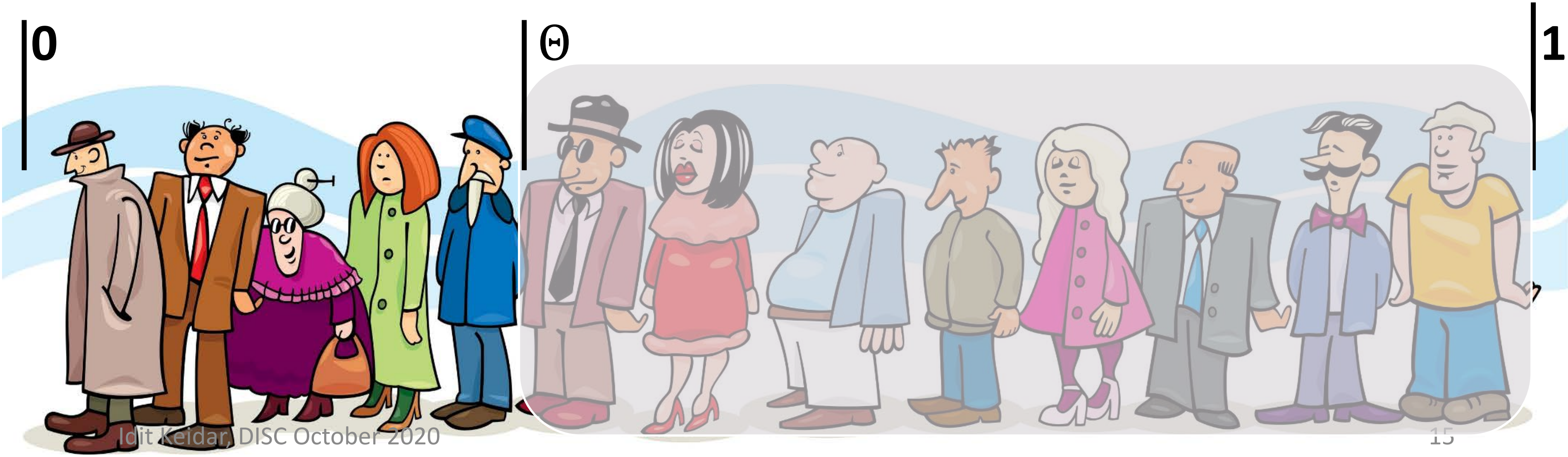
Θ Sketch: Basic Idea

- Hash unique elements into [0,1] uniformly at random

- How do we estimate how many there are?
- Without keeping all of them in memory?

Θ Sketch: Basic Idea

- Hash unique elements into [0,1] uniformly at random
- For a threshold Θ, $0 < \Theta \leq 1$
- Keep elements with hashes smaller than Θ
  - In expectation, a Θ portion of the uniques in the stream



KMV Θ Sketch
[Bar-Yossef et al. 2002]

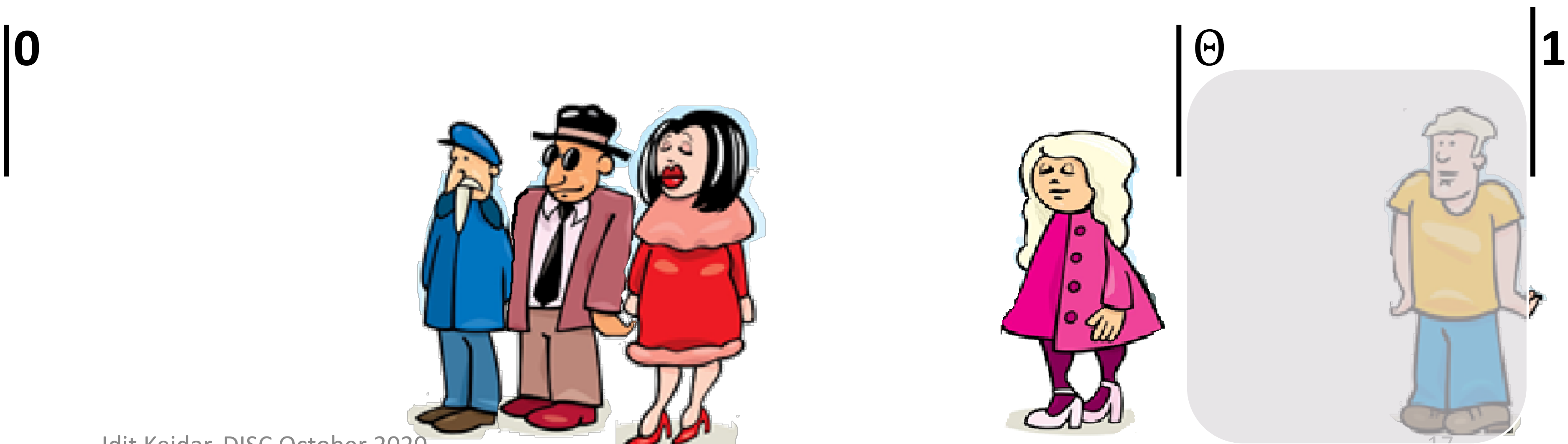
- $\Theta = k^{th}$ minimum hash value seen (initially Θ = 1)
- Estimate = k/Θ
- Example: k=4

0

$\Theta =$ **1**

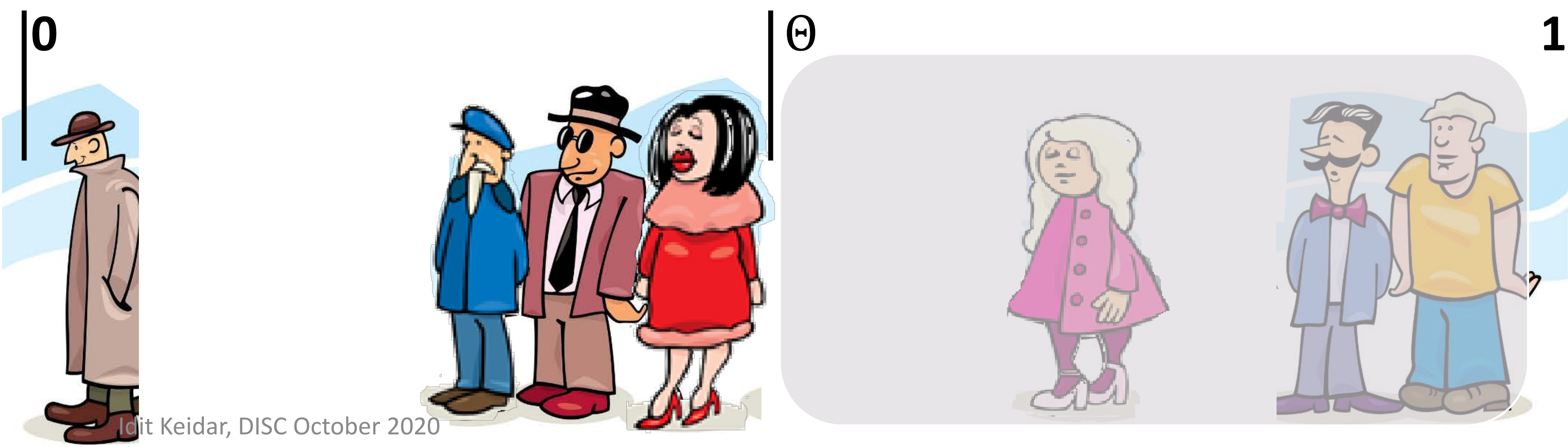
KMV Θ Sketch

- $\Theta = k^{th}$ minimum hash value seen (initially Θ = 1)
- Estimate = k/Θ
- Example: k=4



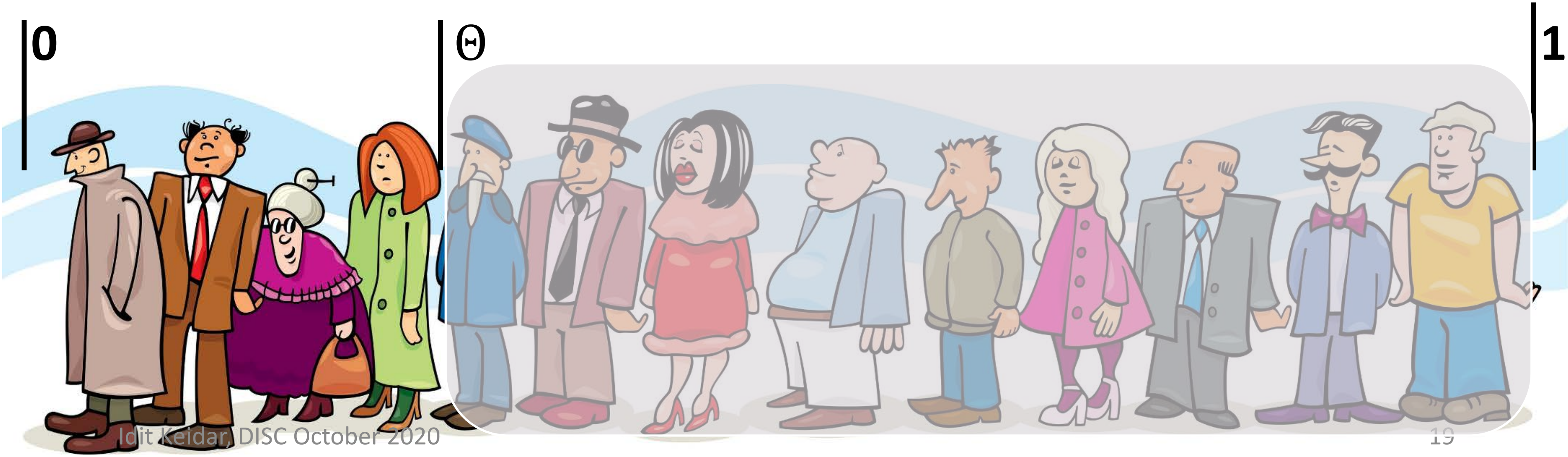
KMV Θ Sketch

- $\Theta = k^{th}$ minimum hash value seen (initially Θ = 1)
- Estimate = k/Θ
- Example: k=4



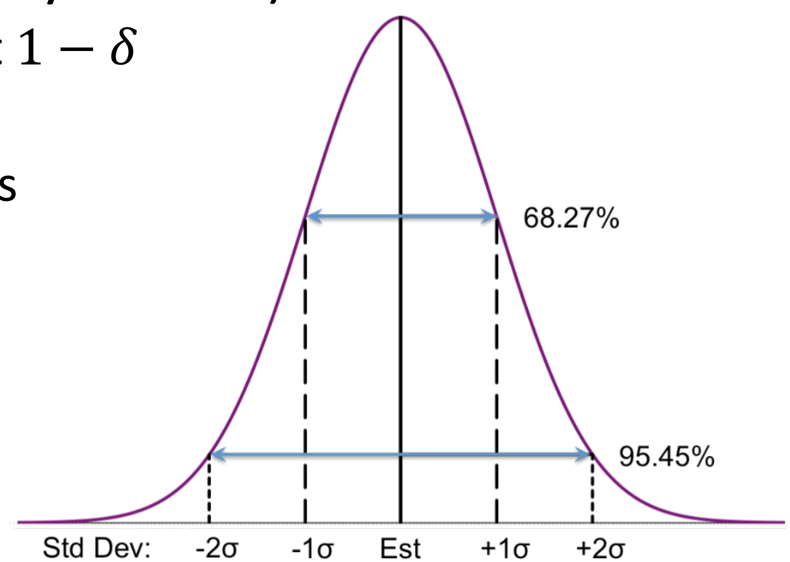
KMV Θ Sketch

- $\Theta = k^{th}$ minimum hash value seen (initially Θ = 1)
- Estimate = k/Θ
- Example: k=4



Sketches Are Approximate

- Typically PAC (probably approximately correct)
  - Error at most ϵ with probability at least $1-\delta$
  - With appropriately chosen parameters
  - Each sketch comes with its own analysis



- KMV provides an estimate $\hat{e}$
  - E[$\hat{e}$] = n, the number of uniques
  - RSE[$\hat{e}$] = $\frac{1}{\sqrt{k-2}}$
  - RSE is the relative standard error = $\frac{\sigma}{n}$

  [Bar-Yossef et al. 2002]

Θ Sketches Are Fast

- On incoming id

  h = hash(id)

  if h < Θ

  add h to sketch

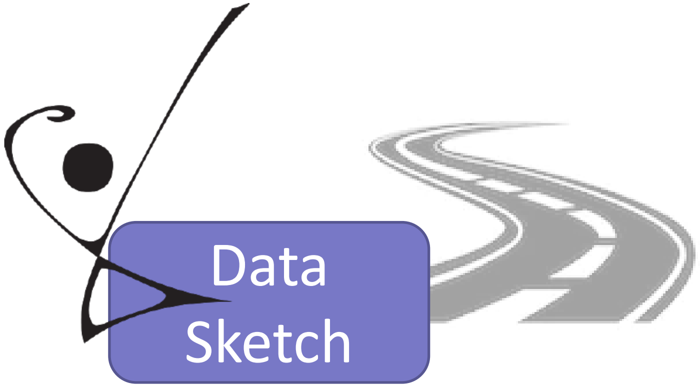
  if |sketch| > k, remove largest

  Θ = largest hash in sketch

No else!
Once Θ is small, usually does nothing more

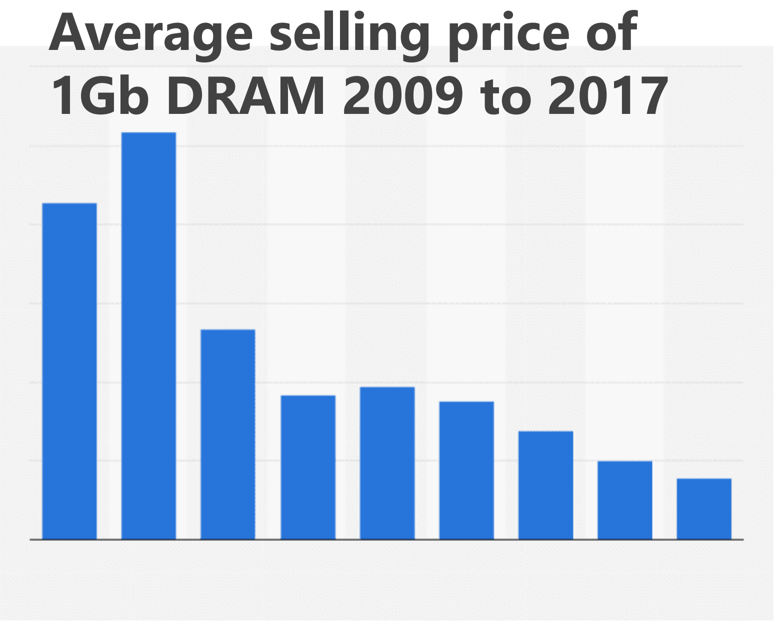
More Examples

- Event counters
- Quantiles – e.g., duration of 90th percentile of sessions
- Item frequency – CountMin
- Heavy hitters



Hardware Trends

- Multi-core servers
  - Performance via parallelism, not sequential speed
- Cheaper DRAM
  - In-memory processing of bigger data



What and Why - Recap

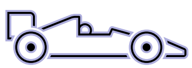
- What?
  - Concurrent data sketches, approximate counters
- Why?
  - Online monitoring & analytics of big data streams
- Why concurrent?
  - Today’s hardware: multi-core with larger RAM

- Challenges
  - Efficient implementation
  - Meaningful semantics – leveraging what we know about the sequential case

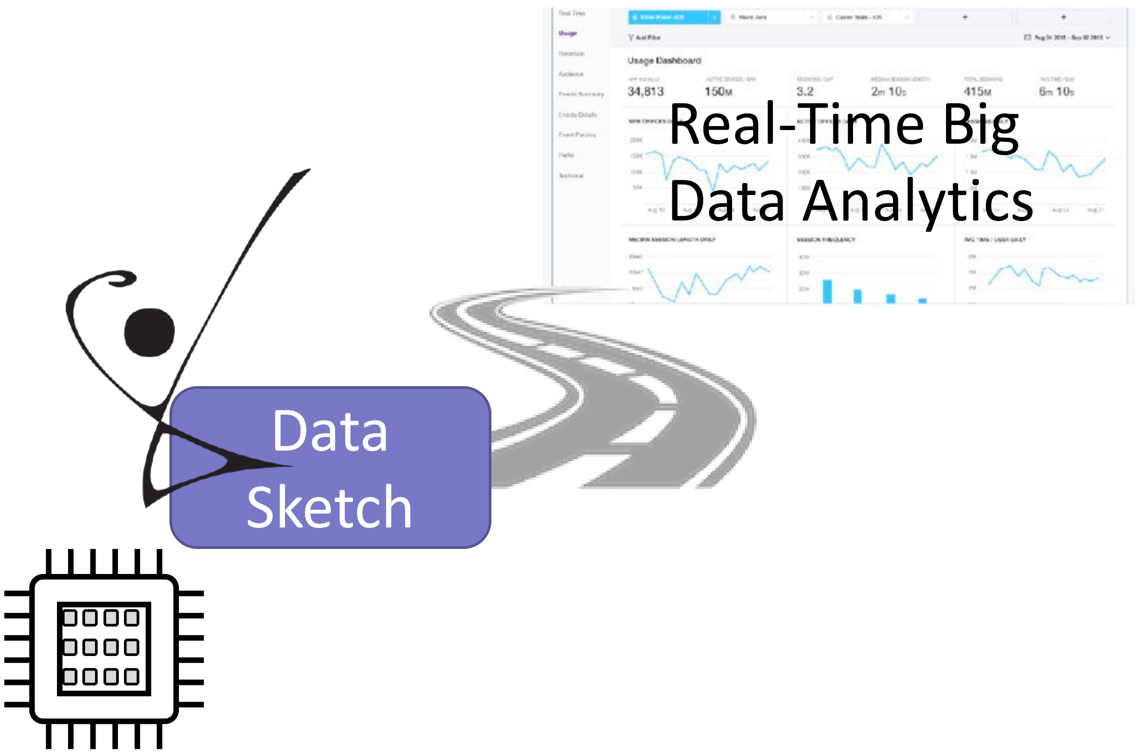
Roadmap Recap

Concurrent data sketches:

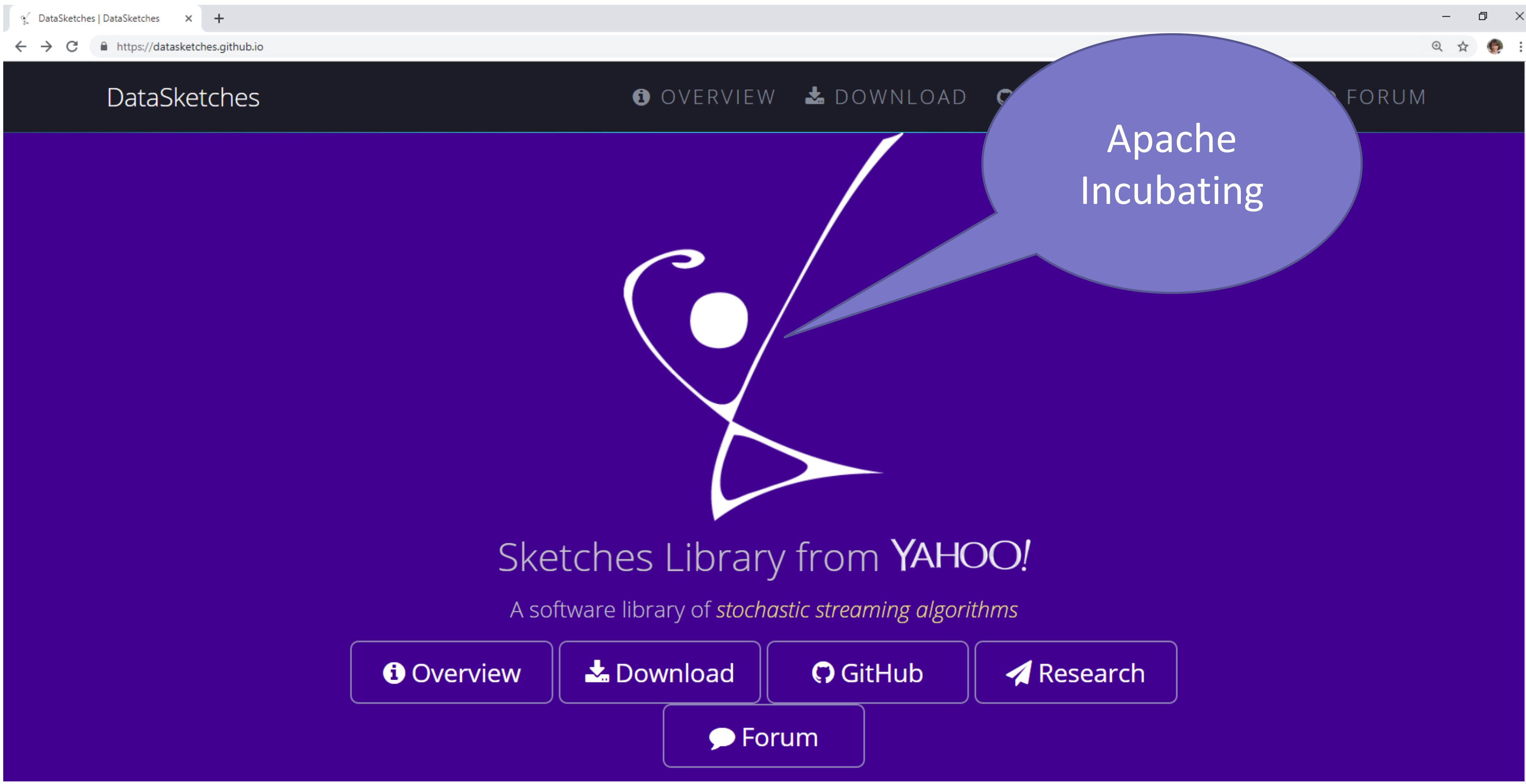
1. Fast implementation



2. Correctness semantics



Context: Open-Source DataSketches Library



Today's Sketches Aren't Thread-Safe

sketches-user ›

SketchesArgumentException: Key not found and no empty slot in table

6 posts by 2 authors

Hi guys,

I encounter this exception when update sketch. I have googled but found nothing.
Anyone encountered the same issue? Please help me!

leerho commented on Jan 18, 2018 — Contributor

None of the sketches in the library are multi-threaded. If you have concurrent threads reading and writing to the same sketch you must make your sketch wrapper synchronized.

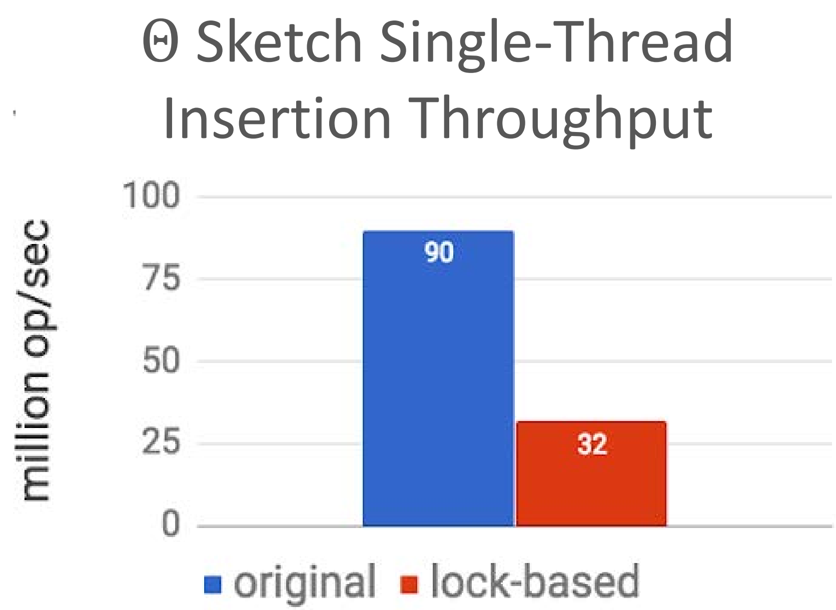
https://github.com/apache/incubator-datasketches-java/issues/178#issuecomment-365673204

Challenge 1: Sketches Aren’t Thread-Safe

Need protection: 

```
try {
        lock (sketch)
        sketch.update(...);
} finally {
        unlock (sketch)
}
```

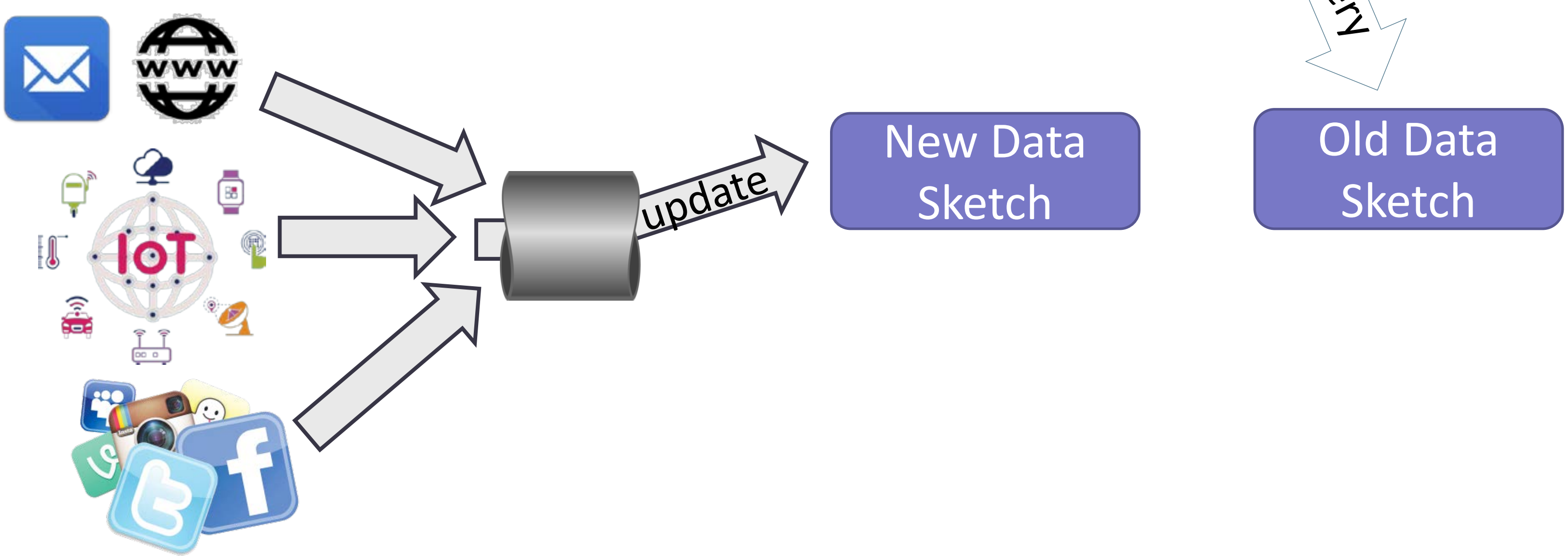
But locks are costly:



Challenge 2: Can’t Query While Updating

Current approach:

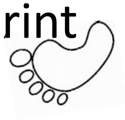
- Use locks 
  or
- Update in epochs, query previous epoch



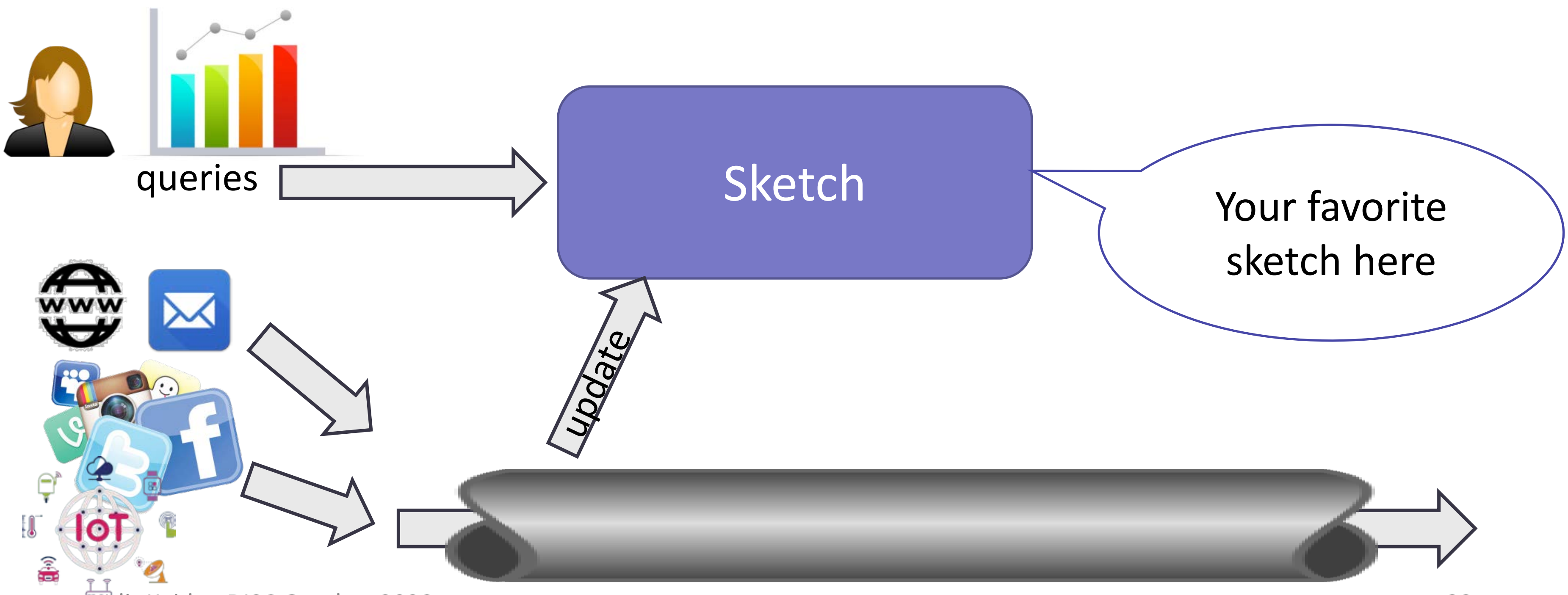
Concurrent DataSketches

Concurrent Sketches - Goals

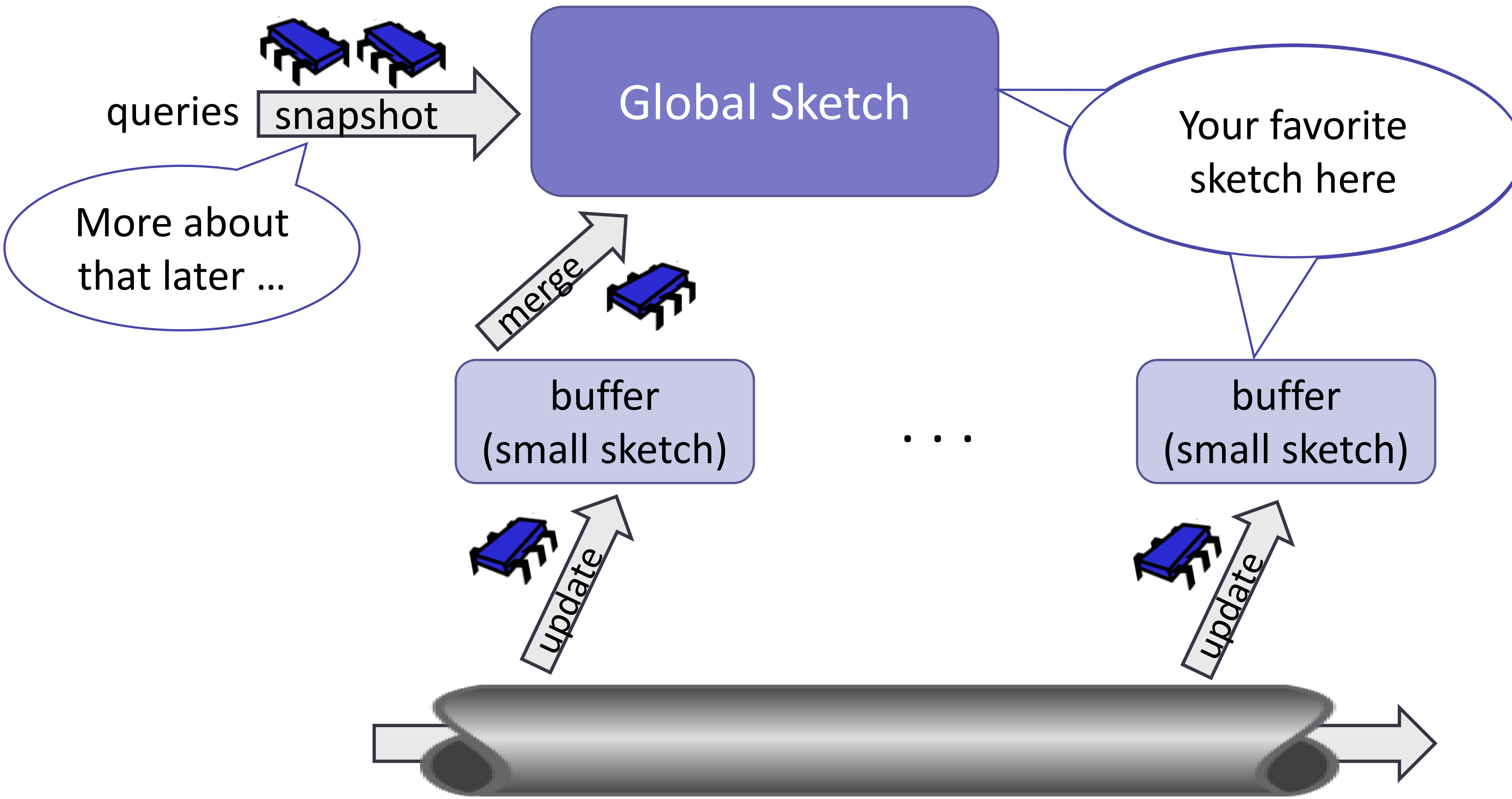
- High throughput
  - Concurrent updates
  - Harness multi-cores for multi-threaded stream processing
- Query freshness
  - Allow queries during updates
- Ease-of-use
  - Library, not application, responsible for synchronization
- Enjoy sketch's benefits
  - Fast
  - Bounded estimation error
  - Small memory footprint



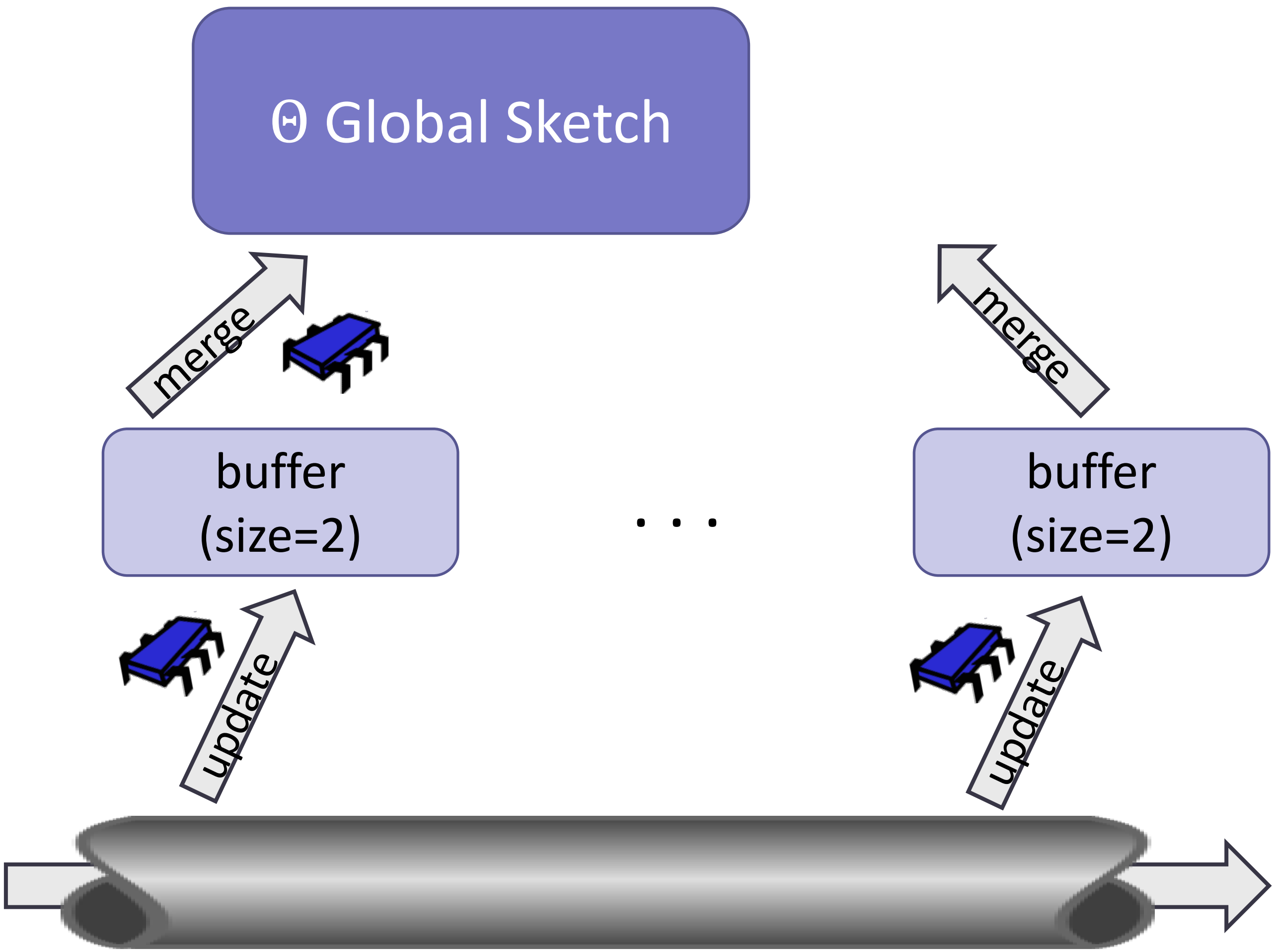
Concurrent Sketches: Generic Architecture



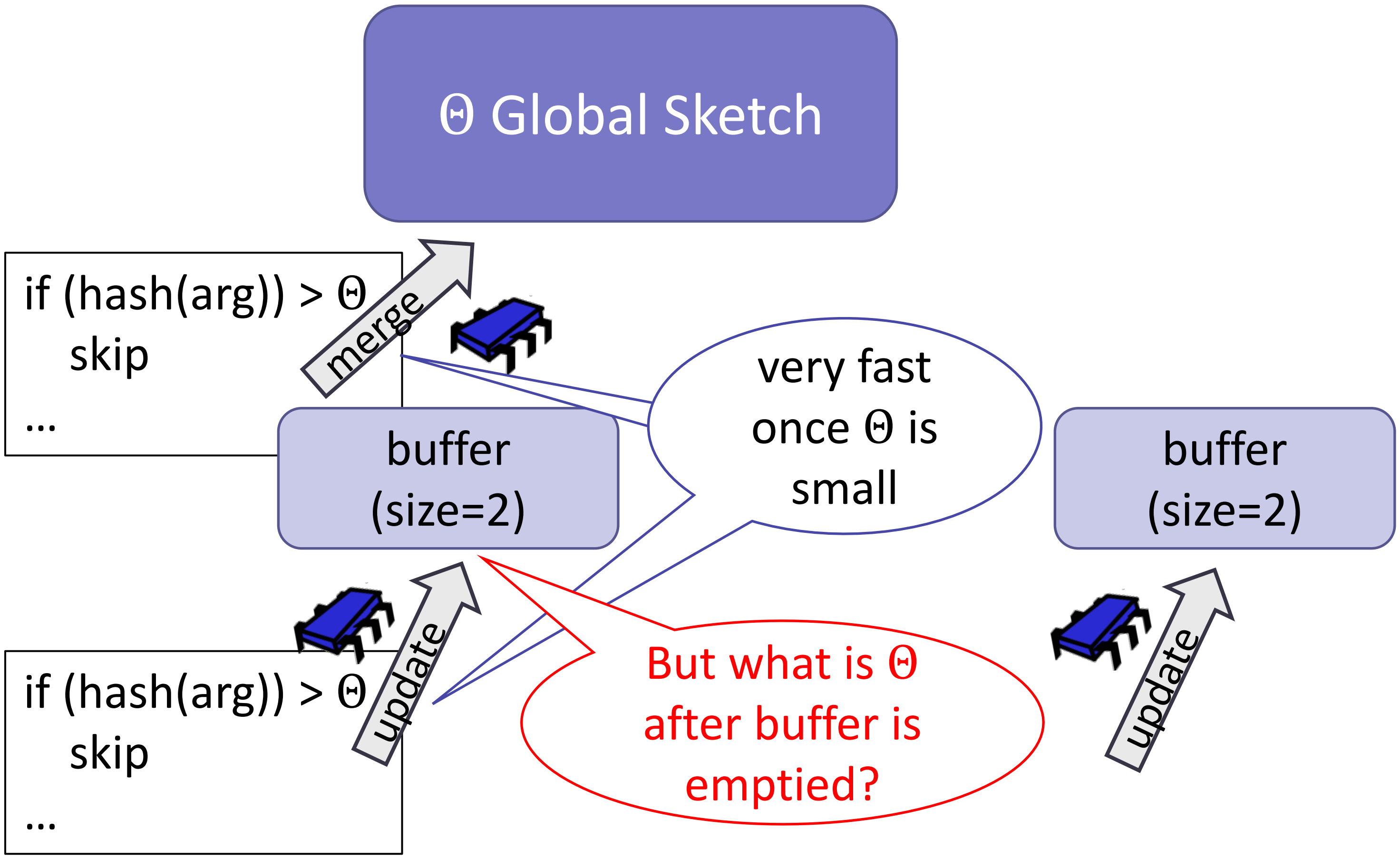
Concurrent Sketches: Generic Architecture



Example


Θ Global Sketch
merge
merge
buffer
(size=2)
. . .
buffer
(size=2)
update
update

What About Fastness?



Optimizations

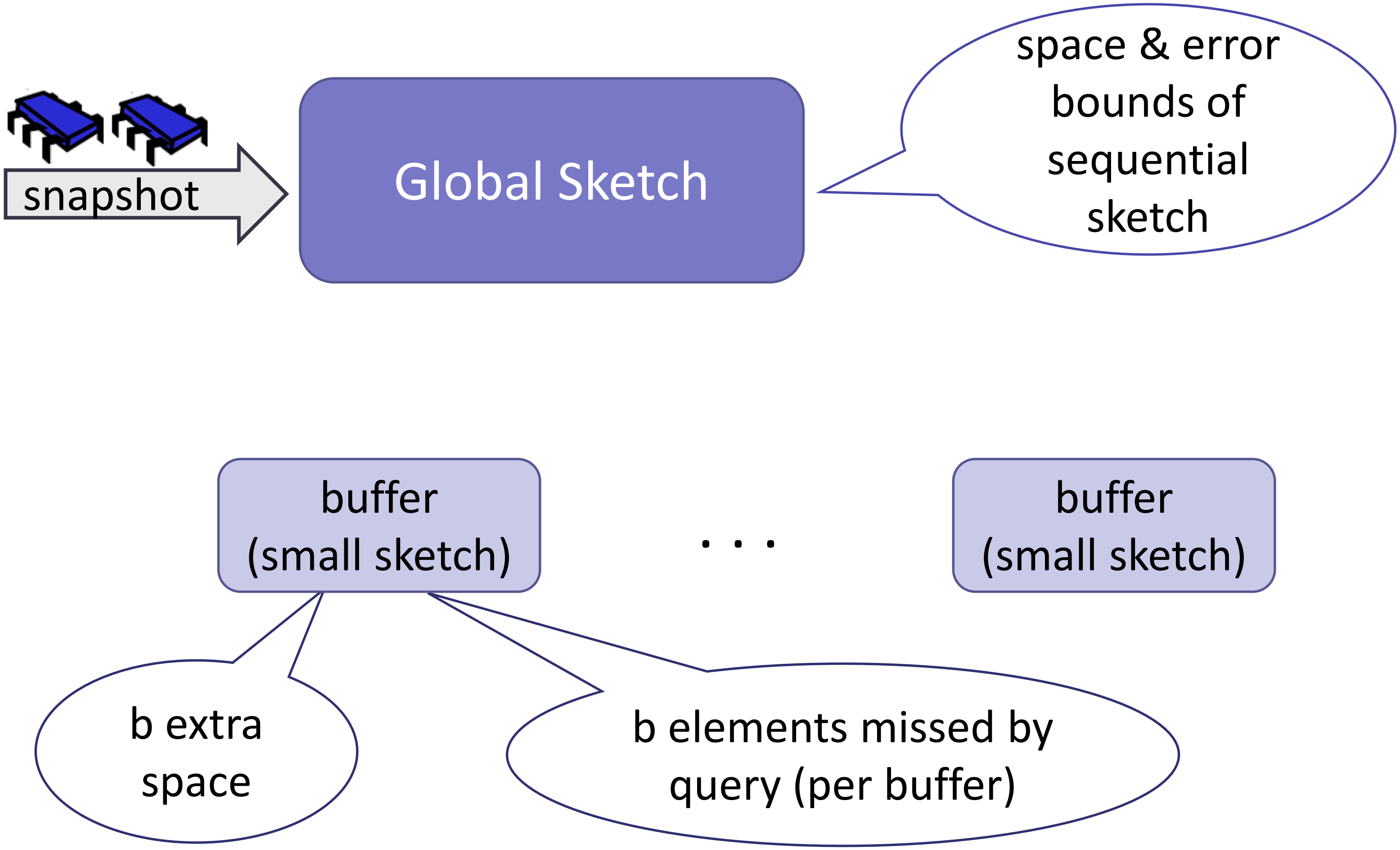
Problem: Missing critical information (e.g., Θ)

Solution: Piggyback sketch-specific information on existing generic synchronization

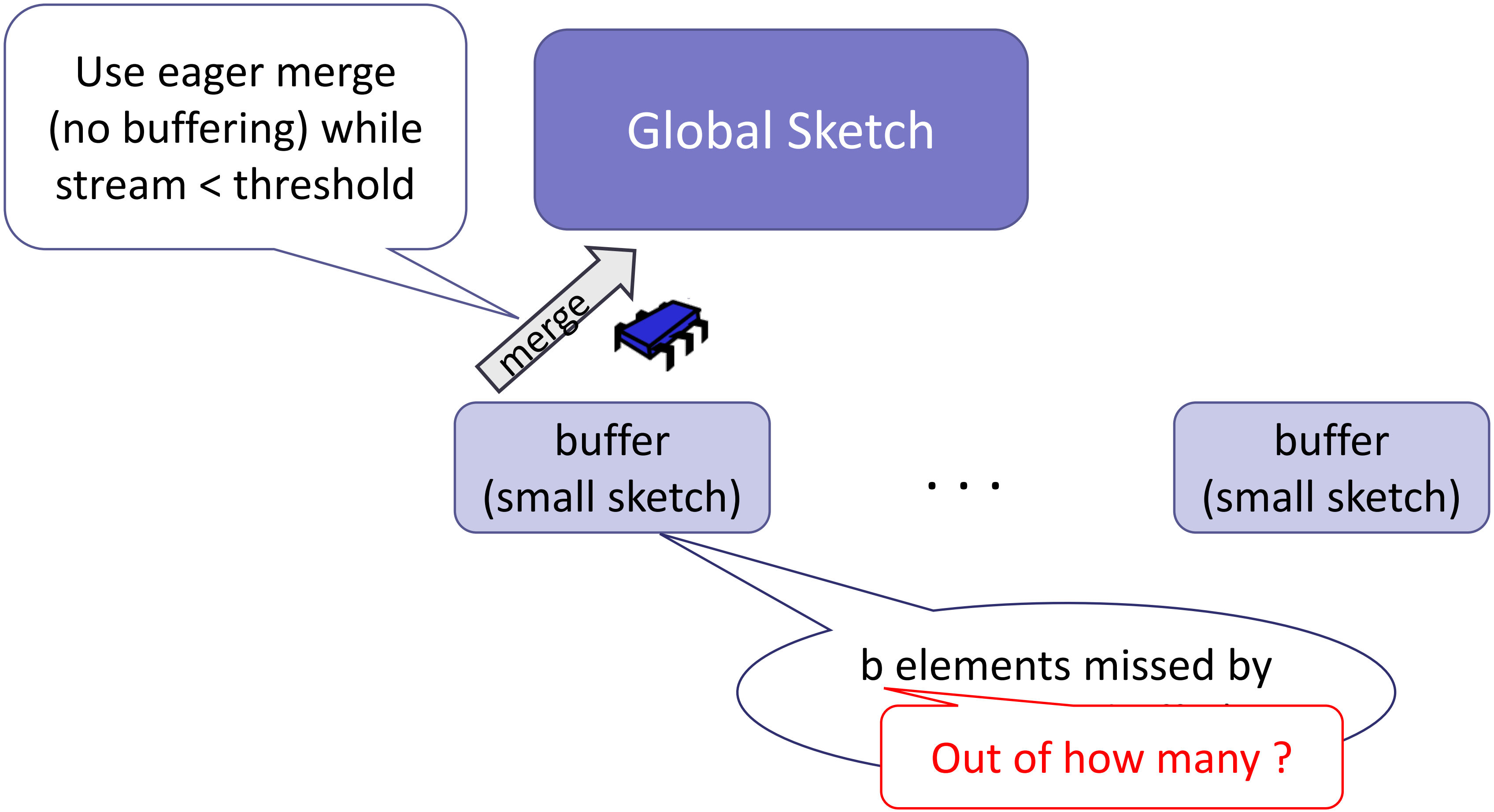
Problem: Thread is idle during propagation

Solution: Use double buffering

Space and Error



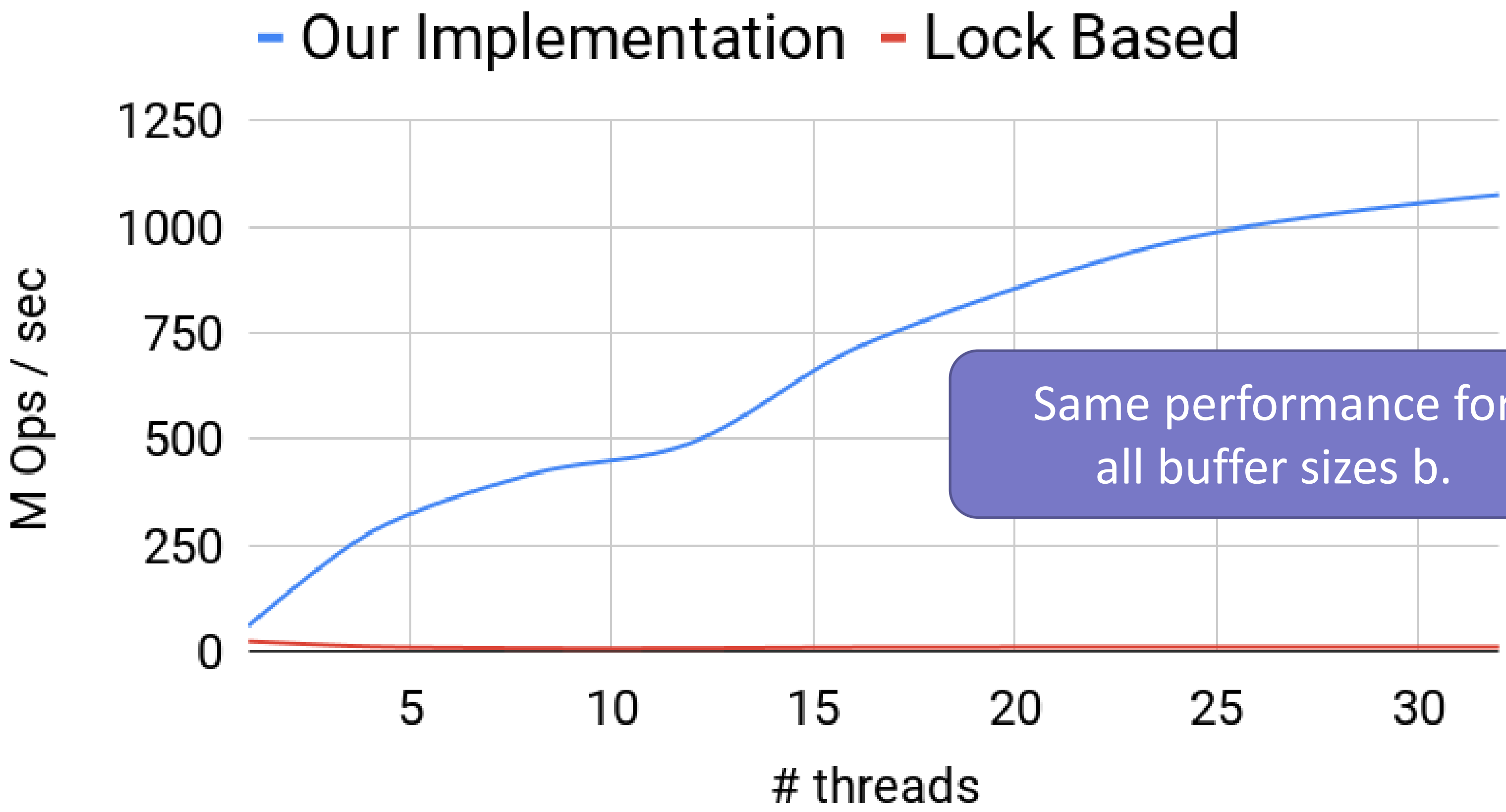
Bounding the Error in Small Streams



Keys to Performance

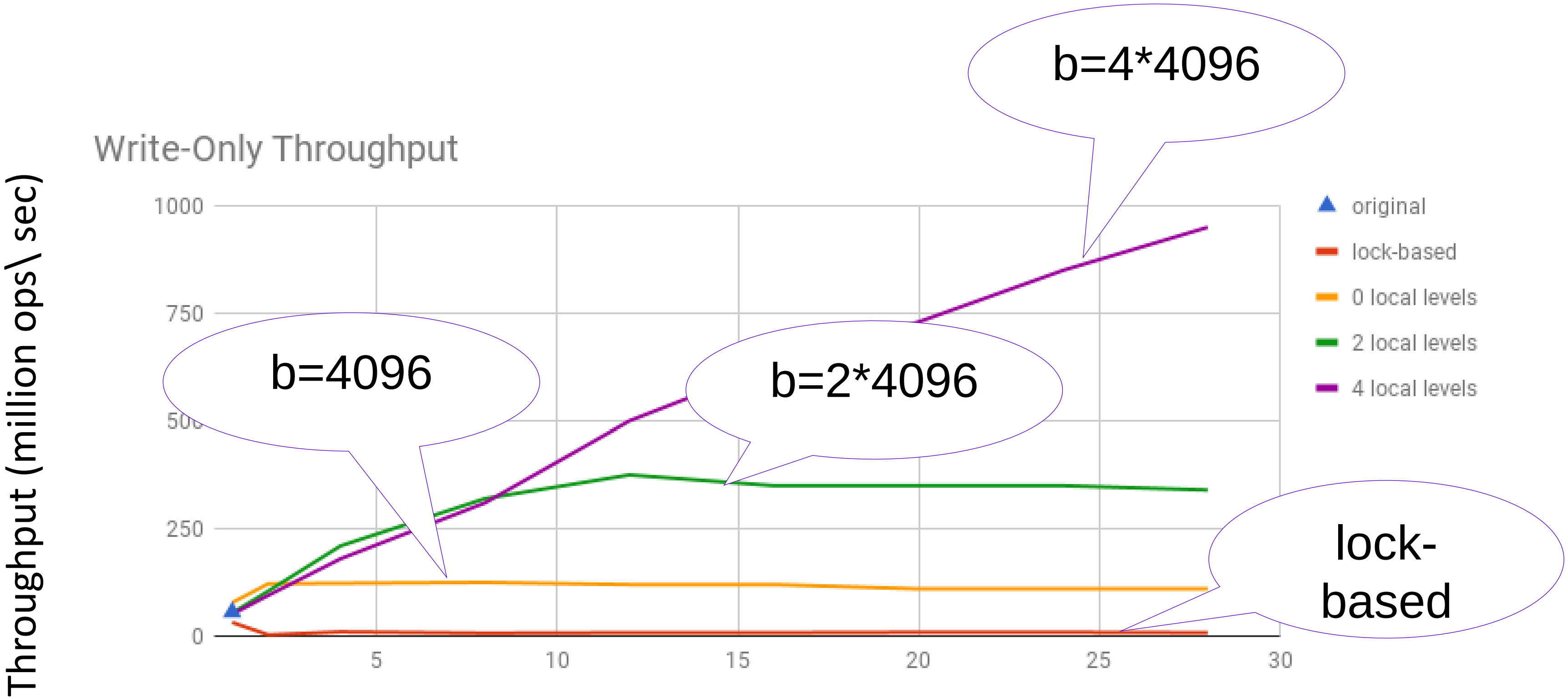
- Minimize synchronization
  - Few fences
  - Synchronize only when buffer is filled/empty
- Locality
  - Cache & NUMA friendly
  - Threads work in (mostly) unshared memory
- But … share pertinent information
  - E.g., up-to-date Θ for fast processing

Update Throughput



Same performance for all buffer sizes b.

Another Example: Quantiles Sketch



Here, the buffer size b matters.

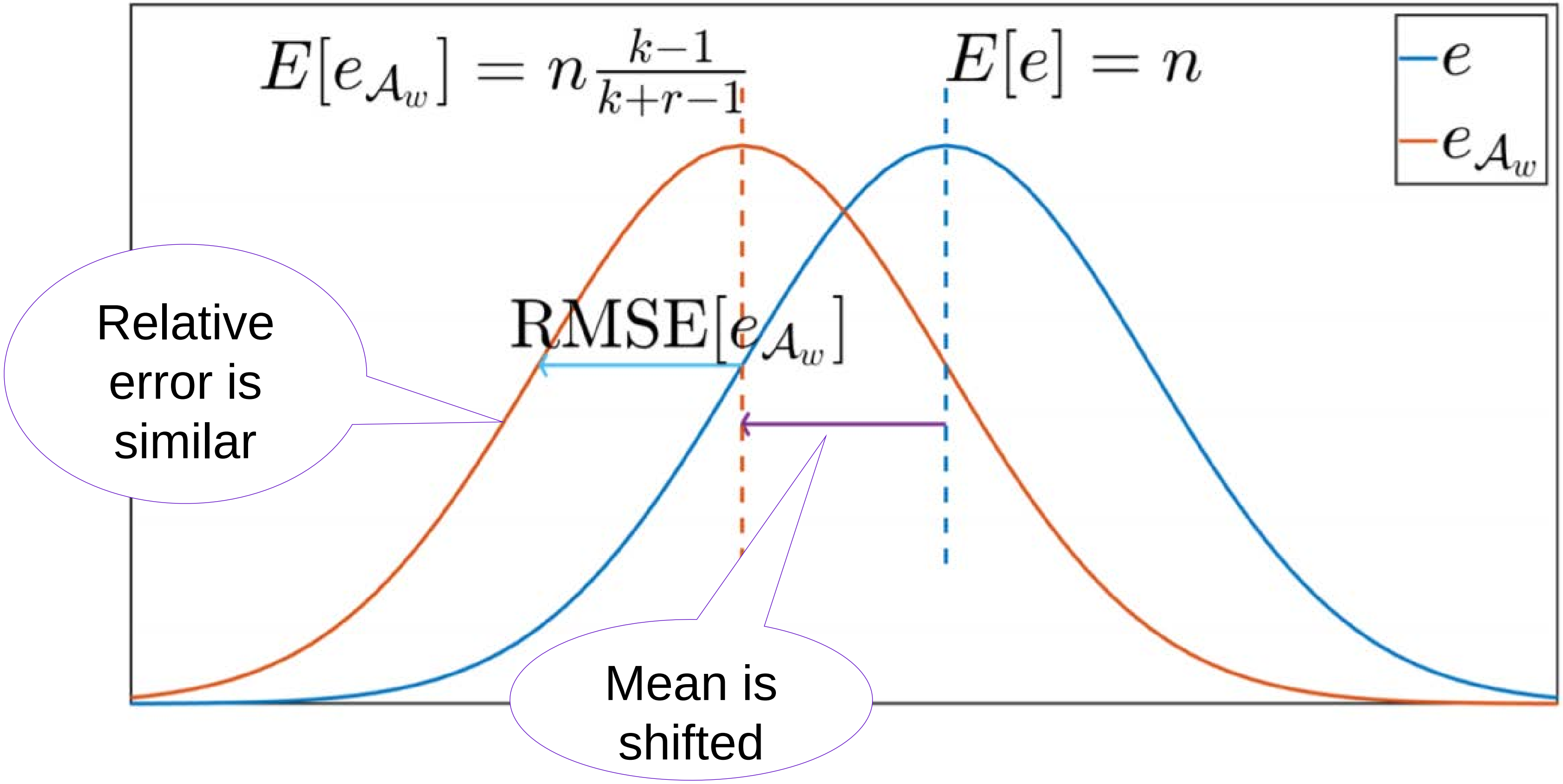
Proof Overview

- We show that
  - our generic algorithm
  - instantiated with a composable sketch
  - satisfies strong linearizability [Golab et al.]
  - wrt an r-relaxation [Henzinger et al.] of
  - the sequential specification derived from the sequential sketch
  - for r = 2Nb; N = #threads, b = buffer size

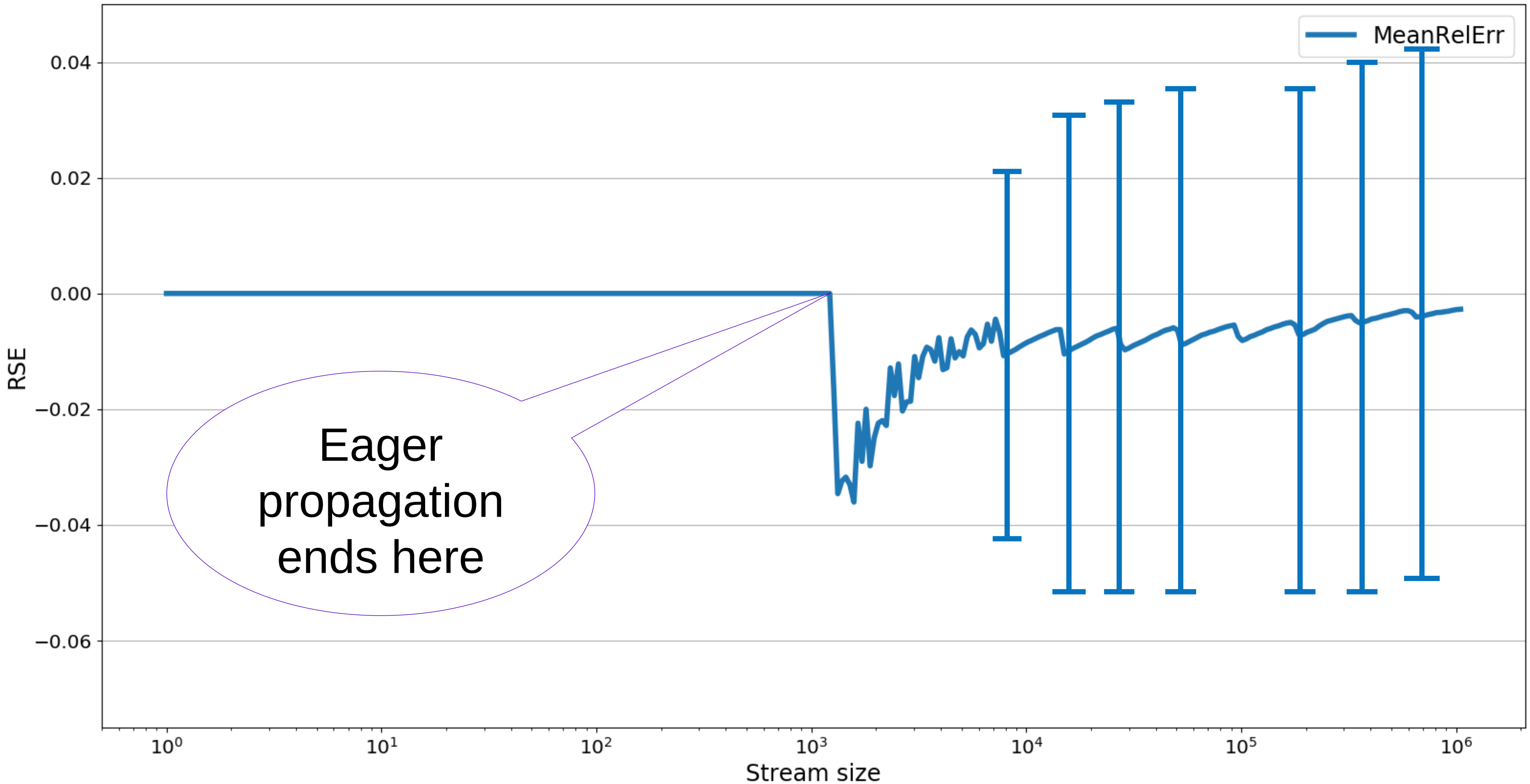
We then analyze the error of the relaxed specification

By strong linearizability, this is the error of our sketch!

Analyzed Error of Concurrent Θ sketch

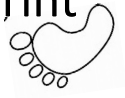


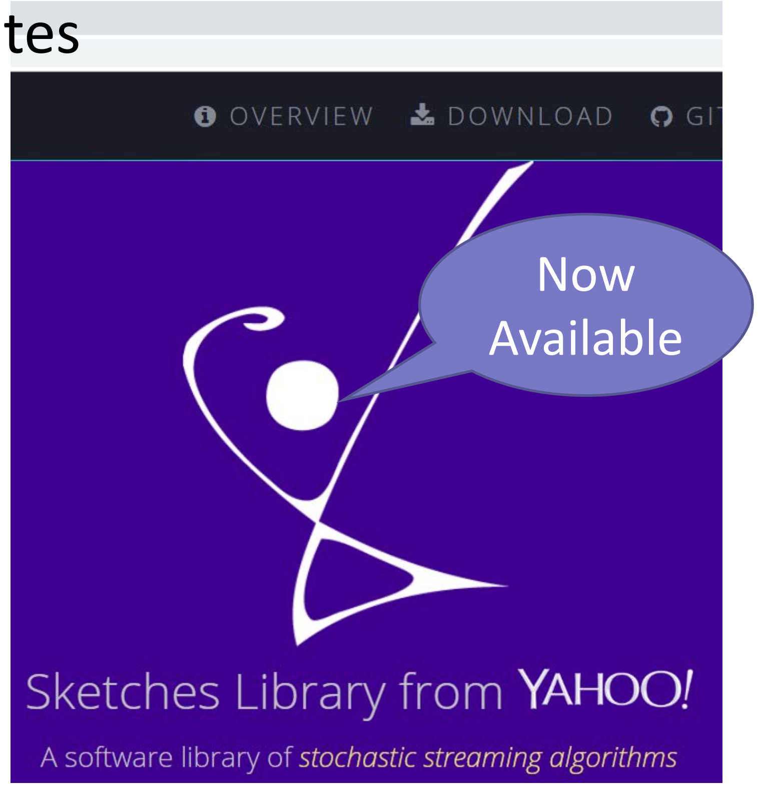
Empirical Evaluation of Relative Error



Interim Summary: Fast Concurrent Sketches

- Generic solution based on composable sketches
  - Rigorous correctness proof using relaxed consistency
- High throughput via concurrent updates
- Query freshness
  - Allow queries during updates
- Ease-of-use
  - Library responsible for synchronization
- Enjoy sketches' benefits
  - Fast
  - Bounded estimation error
  - Small memory footprint



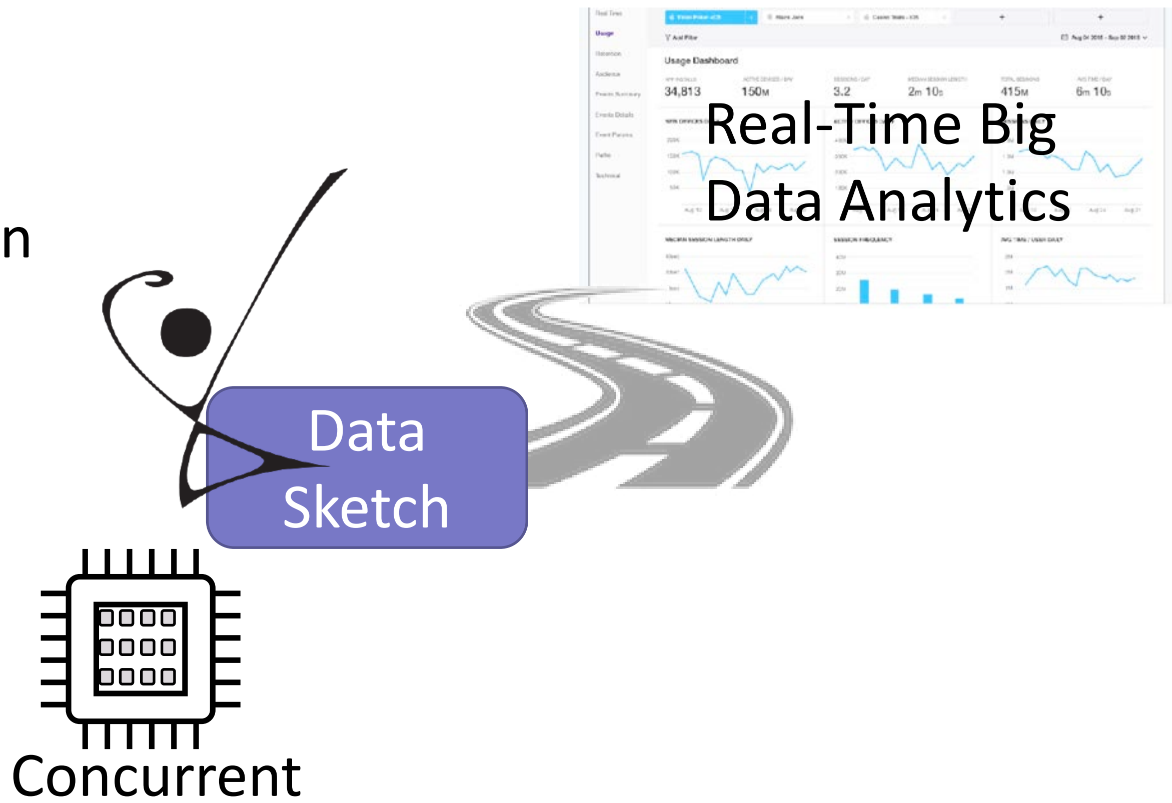


Roadmap Recap

Concurrent data sketches:

1. Fast implementation

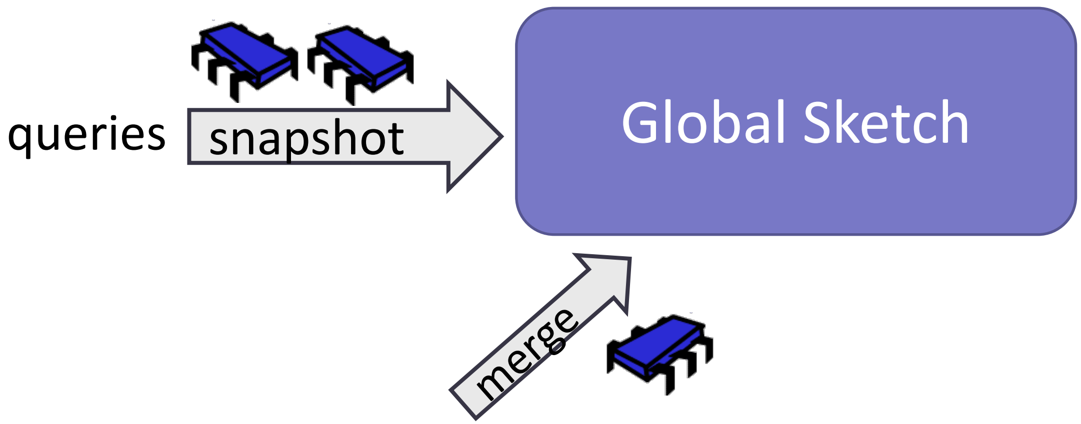
2. Correctness semantics



Wait, didn't you just say you proved correctness?

Something about r-relaxed strong linearizability?

Concurrency on the Global Sketch Revisited



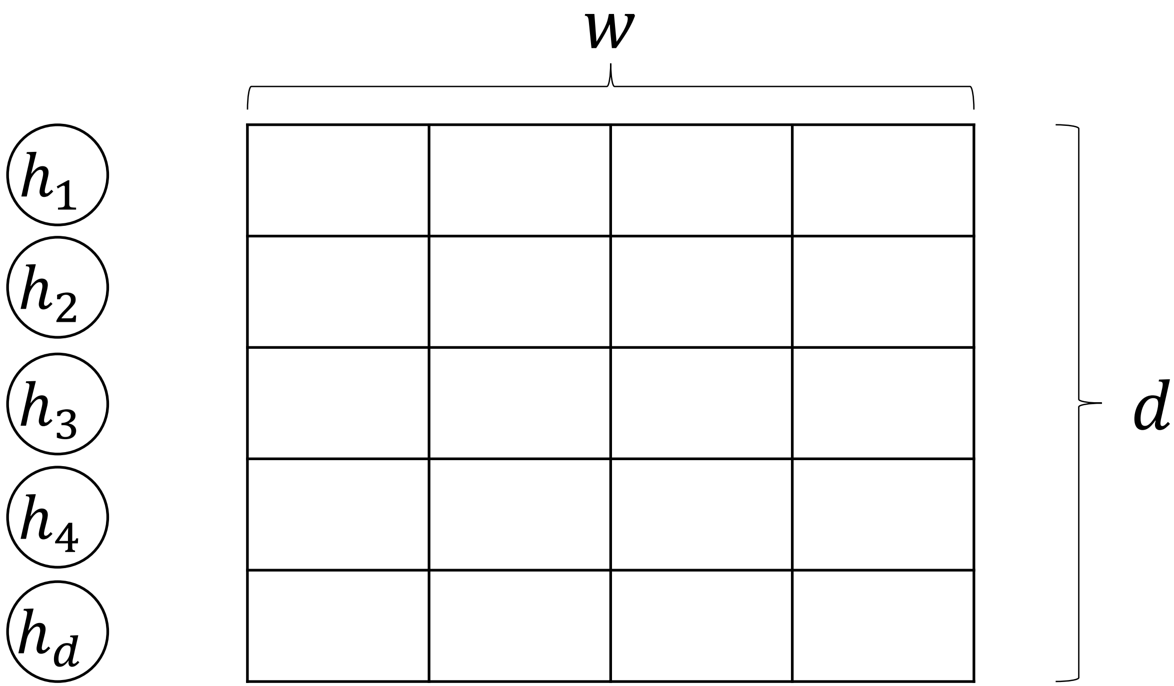
- The global sketch is strongly linearizable
  - The r-relaxation only arises due to buffering (local sketches)
- In general, this requires atomic snapshots
  - In the Θ sketch, snapshots are cheap
  - Alas, this is not always the case

Told ya I'd say more about that.

Example: CountMin Sketch

[Cormode and Muthukrishna, 2005]

- Estimates item frequency



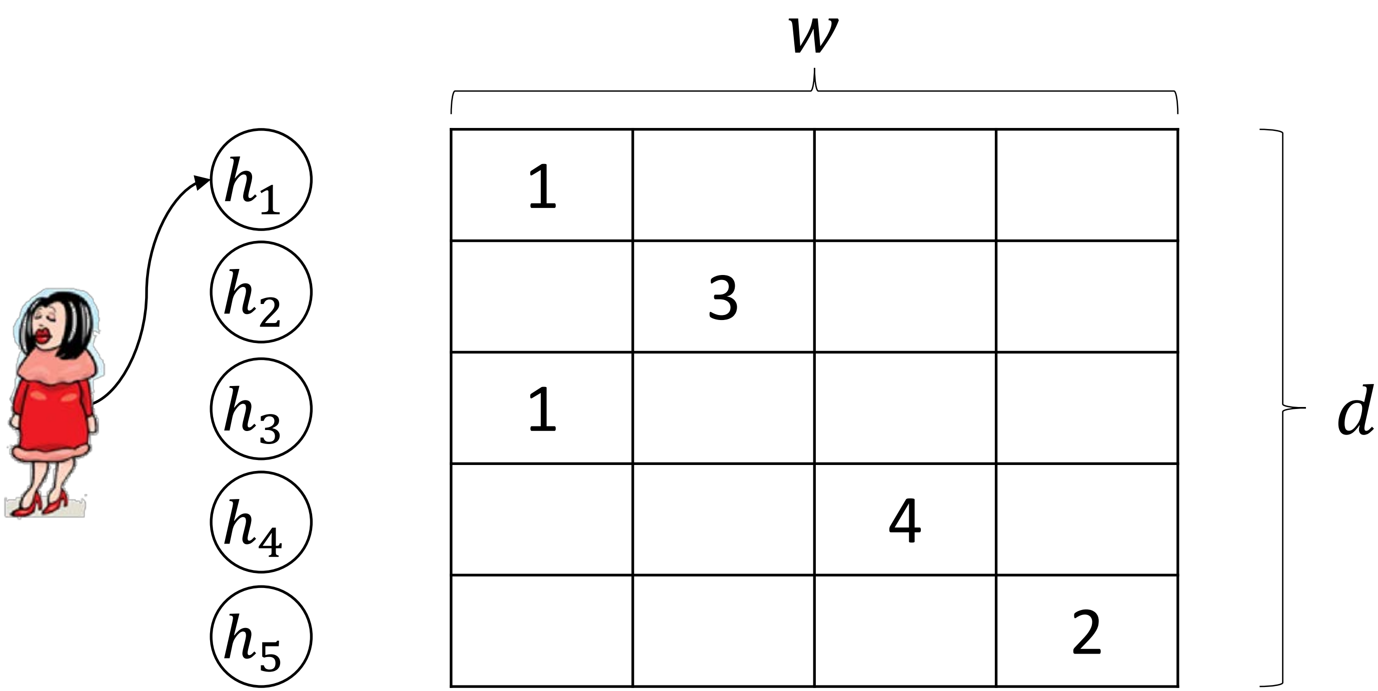
$$h_1, \dots, h_d : \Sigma \mapsto [w]$$

Example: CountMin Sketch

$$
\begin{array}{c}
w \\
\left.
\begin{array}{c|c|c|c|c|}
\cline{2-5}
h_1 & 1 & & & \\
\cline{2-5}
h_2 & & 3 & & \\
\cline{2-5}
h_3 & 1 & & & \\
\cline{2-5}
h_4 & & & 4 & \\
\cline{2-5}
h_5 & & & & 2 \\
\cline{2-5}
\end{array}
\right\} d
\end{array}
$$

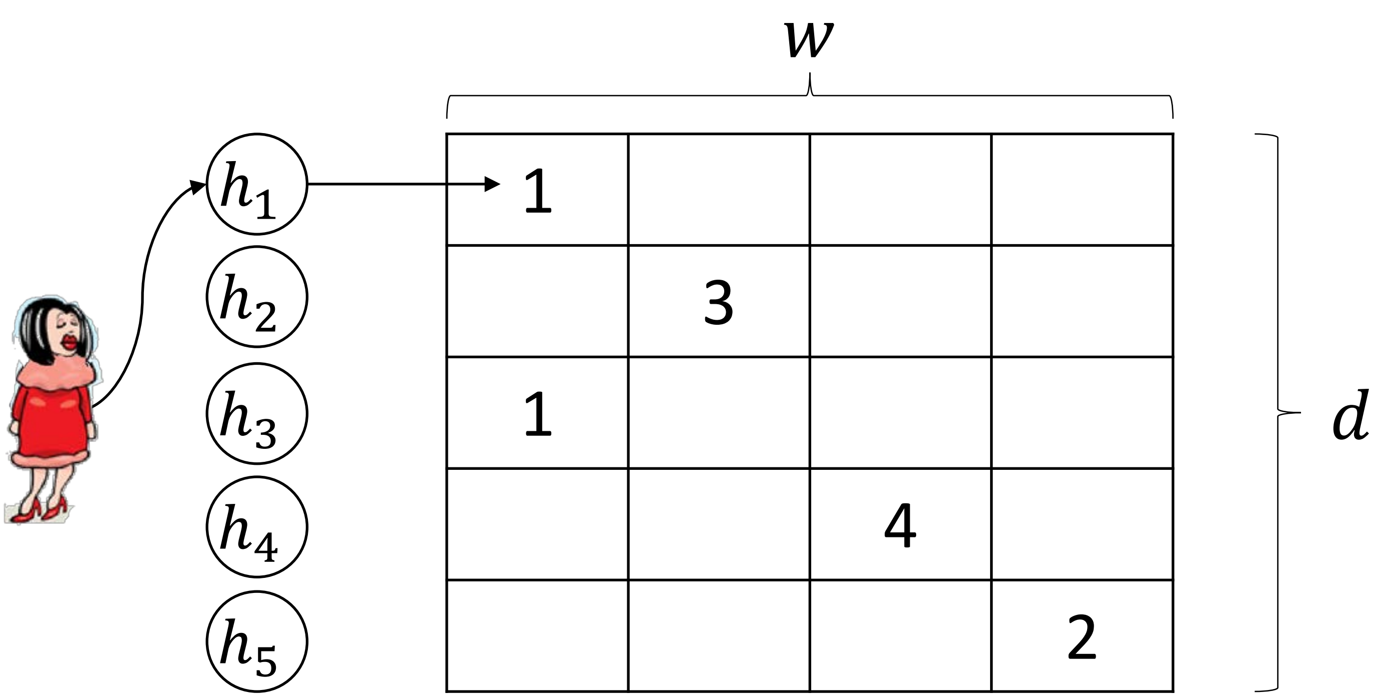
$$h_1, \dots, h_d : \Sigma \mapsto [w]$$

Example: CountMin Sketch



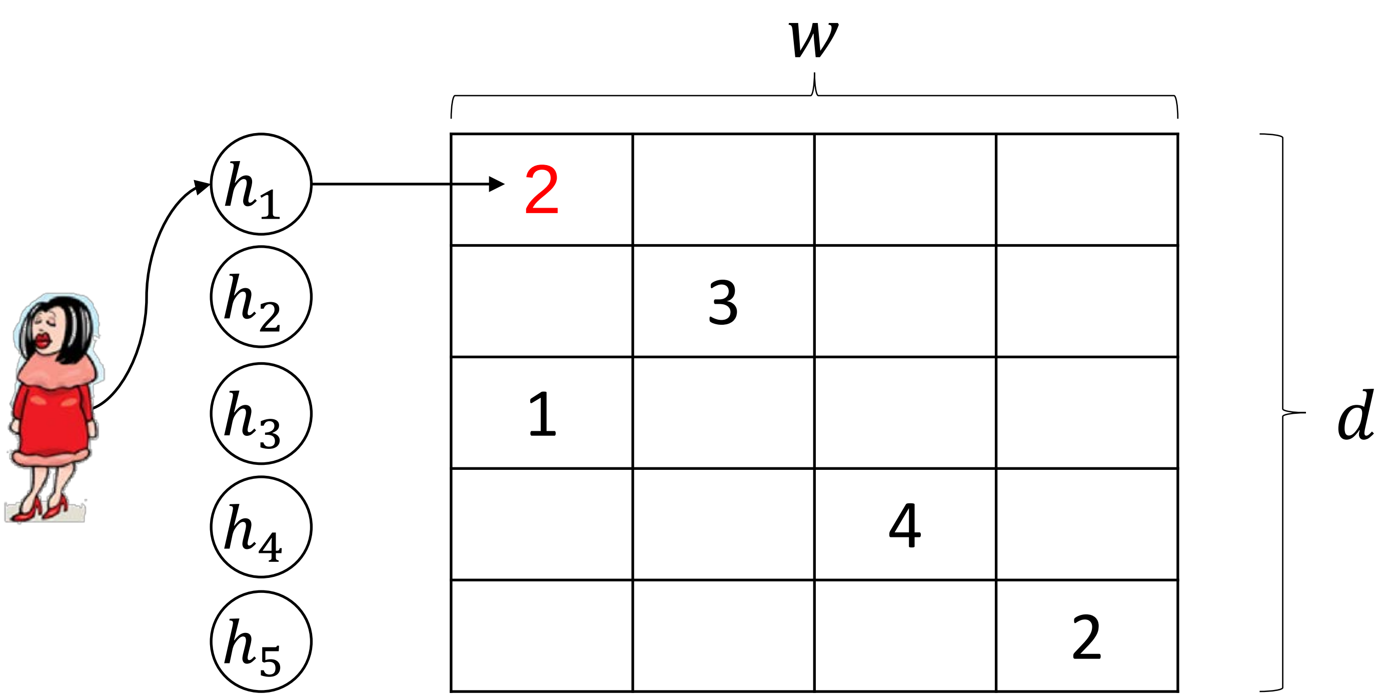
$$h_1, \dots, h_d : \Sigma \mapsto [w]$$

Example: CountMin Sketch



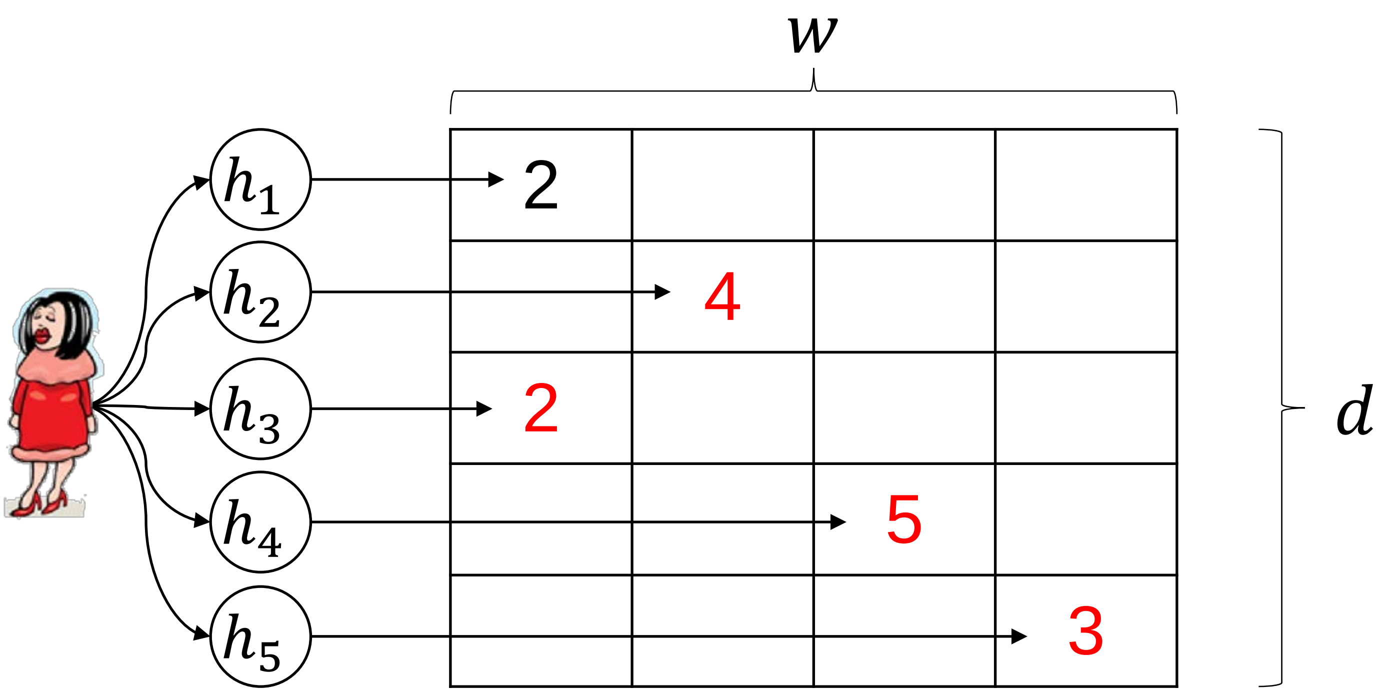
$$h_1, \ldots, h_d : \Sigma \mapsto [w]$$

Example: CountMin Sketch



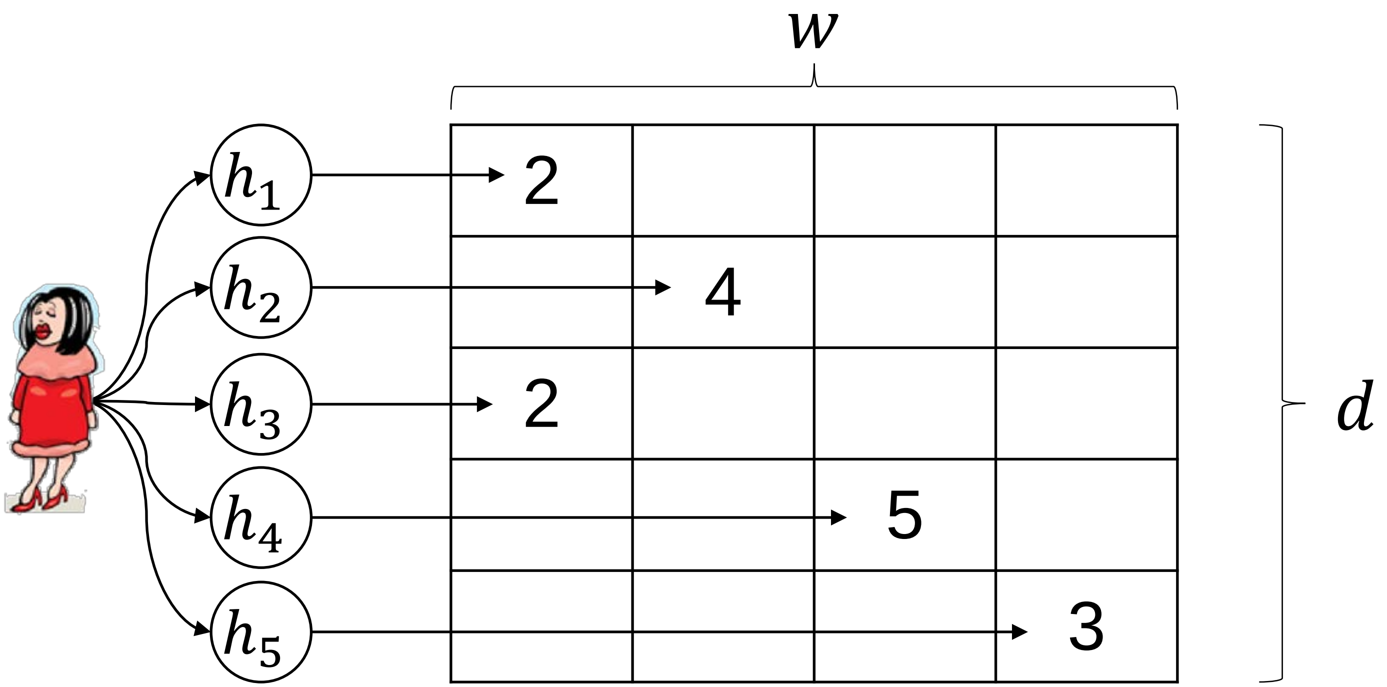
$$h_1, \dots, h_d : \Sigma \mapsto [w]$$

Example: CountMin Sketch



$$h_1, \dots, h_d \colon \Sigma \mapsto [w]$$

Example: CountMin Sketch



$h_1, \dots, h_d : \Sigma \mapsto [w]$

Example: CountMin Sketch

query() returns:

$\min\{h_i($$)\}_{i=1}^{d}$

Returns 2

w

h_1	2			
h_2		4		
h_3	2			
h_4			5	
h_5				3

d

$h_1, \ldots, h_d : \Sigma \mapsto [w]$

CountMin Sequential Error Bounds

- Consider a query invoked after N updates
- Let $f(a)$ denote the frequency of a in these updates
- $\text{query}(a)$ returns an estimate $\hat{f}(a)$ of $f(a)$
- For desired parameters ϵ, δ,

  CountMin's parameters w and d can be chosen so that

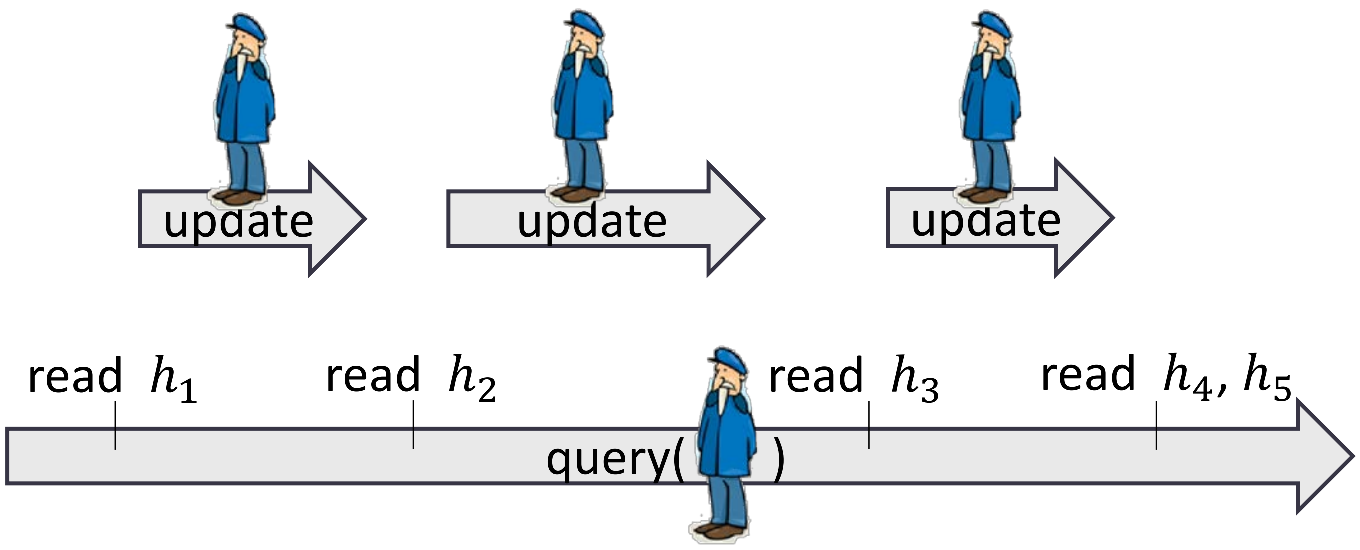
  $f(a) \leq \hat{f}(a)$, and with probability at least $1 - \delta$:

  $$\hat{f}(a) \leq f(a) + \epsilon N$$

[Cormode and Muthukrishna, 2005]

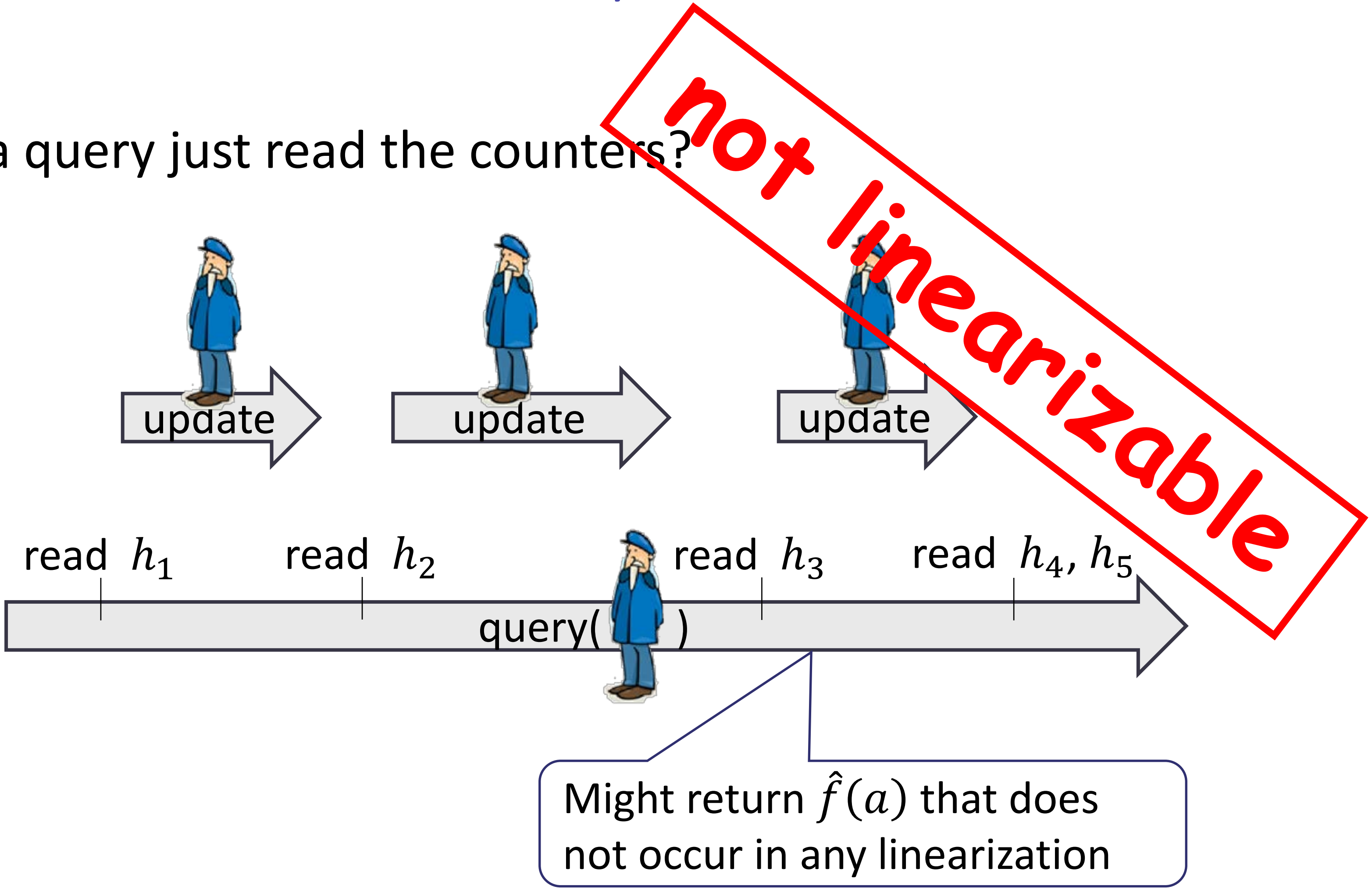
What About Concurrency?

- Can a query just read the counters?



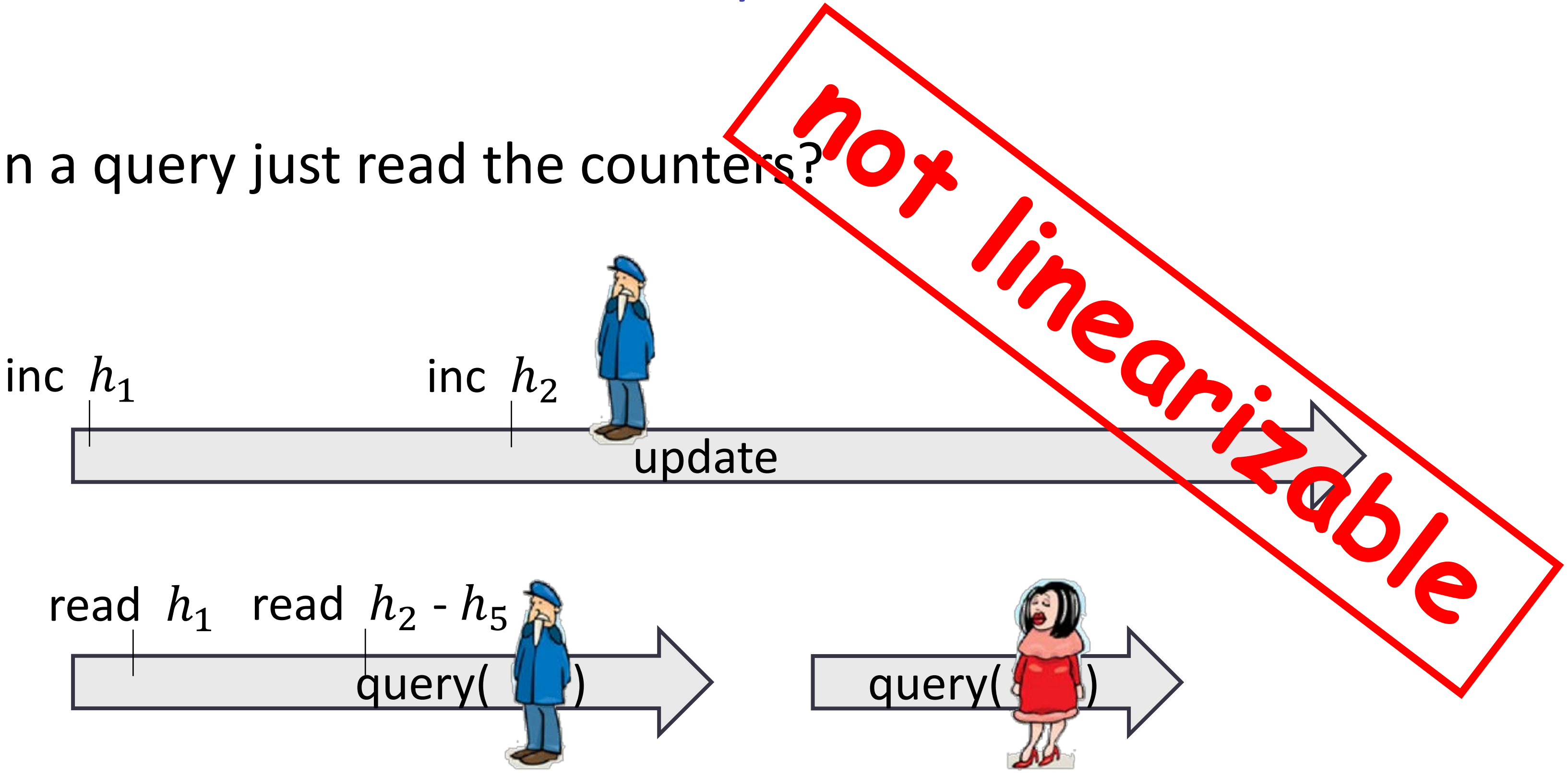
What About Concurrency?

- Can a query just read the counters?



What About Concurrency?

- Can a query just read the counters?



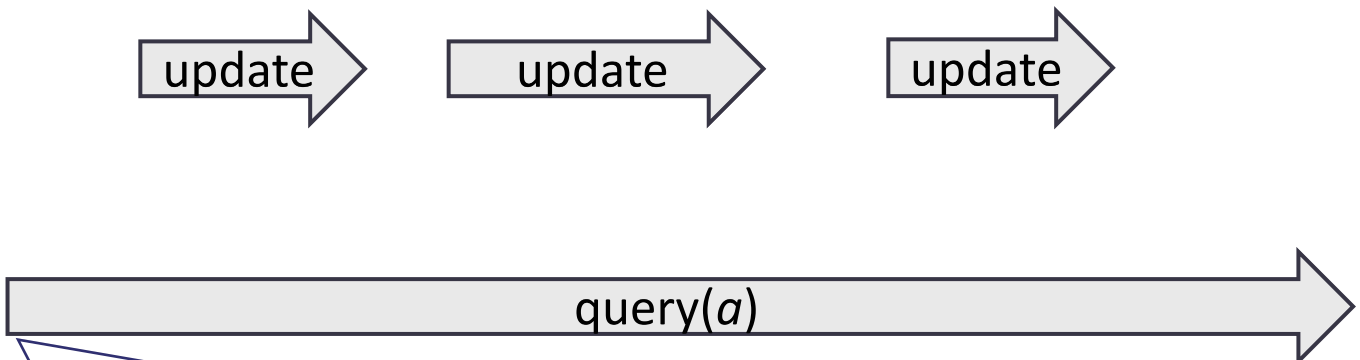
Problem?

- We required the shared global sketch to be strongly linearizable
- This makes it indistinguishable from an atomic variable
- And so preserves the error bounds of the sequential sketch
- Note: this holds for *any* sequential sketch

- But … requires an atomic snapshot (costly)

But …

- What if a query just reads the counters?

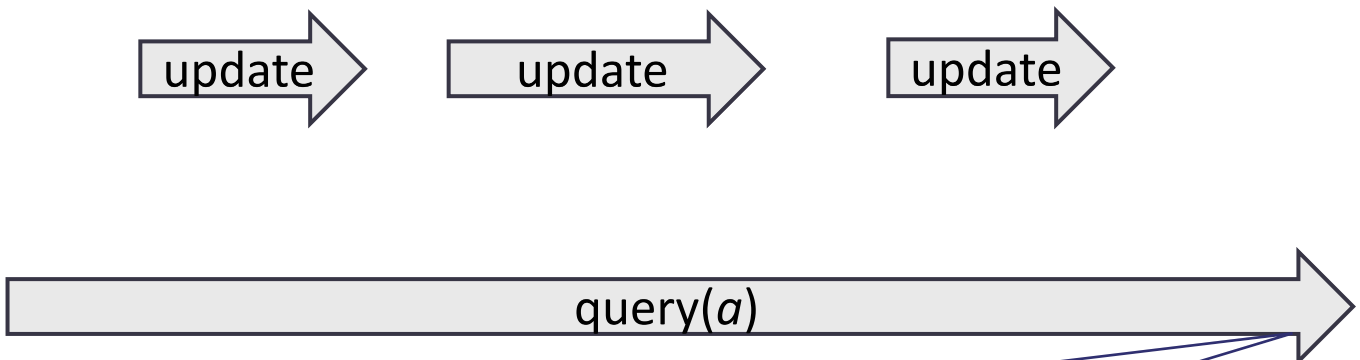


If the query atomically happens here, it returns $\hat{f}^s(a)$ so that

$$f^s(a) \leq \hat{f}^s(a)$$

But …

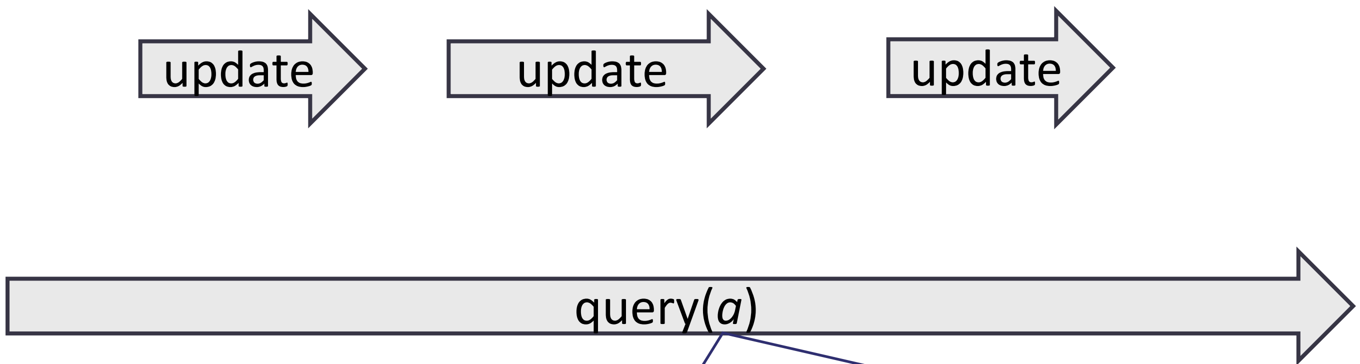
- What if a query just reads the counters?



If the query atomically happens here, it returns $\hat{f}^e(a)$ so that
$\hat{f}^e(a) \leq f^e(a) + \epsilon N^e$ with probability at least $1 - \delta$

But …

- What if a query just reads the counters?



All counters are monotonic, so the query returns $\hat{f}(a)$

$$\hat{f}^{s}(a) \leq \hat{f}(a) \leq \hat{f}^{e}(a)$$

So …

- $f^s(a) \leq \hat{f}^s(a)$
- $\hat{f}^e(a) \leq f^e(a) + \epsilon N^e$ with prob $\geq 1 - \delta$
- $\hat{f}^s(a) \leq \hat{f}(a) \leq \hat{f}^e(a)$

The error remains bounded without a snapshot

- We get: $\boxed{f^s(a) \leq \hat{f}(a) \leq f^e(a) + \epsilon N^e \text{ with prob } \geq 1 - \delta}$

The item's frequency at the time when the query begins

The item's frequency at the time when the query ends

The stream size at the time when the query ends

OK, so a concurrent CountMin sketch does not need to be linearizable, but can you specify what it does need to ensure?

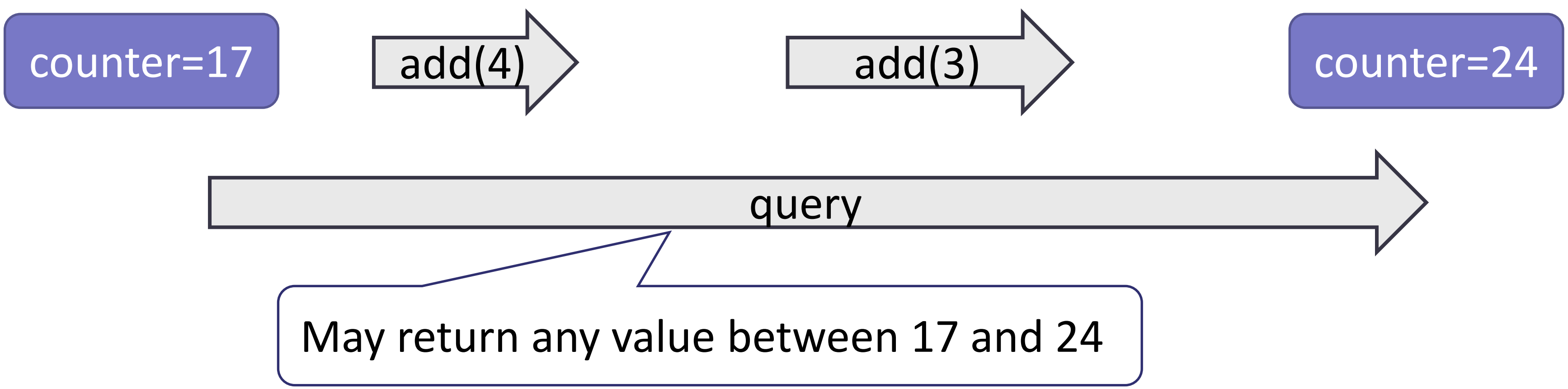
Better yet, can you specify a generic property that applies to many sketches?

Intermediate Value Linearizability (IVL)

- A correctness criterion for concurrent quantitative objects
  - A query returns a value from a totally ordered domain
  - E.g., sketches, counters
- Cheaper than linearizability
  - Inherently in some cases (see Arik's talk)
- Preserves the error bounds of the sequential object
- Enforces (non-relaxed) linearizability in sequential executions, allows more freedom in concurrent ones
- A local property (composable)

IVL – Simple Example

- Every query's return value is bounded between two legal values that can be returned in linearizations



(ϵ, δ)-Bounded Objects

- For an ideal value v, a query returns a value $\hat{v}$ such that

  with probability at least $1 - \delta/2$: $\hat{v} \geq v - \epsilon$

  and

  with probability at least $1 - \delta/2$: $\hat{v} \leq v + \epsilon$

- Many examples, including Θ, Quantiles, CountMin, …

Our Main Theorem

An IVL implementation of a sequential (ϵ, δ)-bounded object is a concurrent (ϵ, δ)-bounded object.

To Conclude

- Big data analytics has big demands
  - Memory is getting bigger – more data can be analyzed in memory
  - CPUs are not getting faster – need to harness multi-cores
- Concurrent processing challenges:
  - Efficiency – minimize synchronization, share pertinent information
  - Correctness – analyze impact of concurrency on error
- Our contributions:
  - Framework for fast concurrent sketches
  - Correctness semantics with guaranteed error bounds

Thank you!