What Can ML Do For Algorithms?

Sergei Vassilvitskii

Google

Theme

Machine Learning is everywhere…

- Self driving cars
- Speech to speech translation
- Search ranking
- …

Theme

Machine Learning is everywhere…

- Self driving cars
- Speech to speech translation
- Search ranking
- …

…but it's not helping us get better theorems

Motivating Example

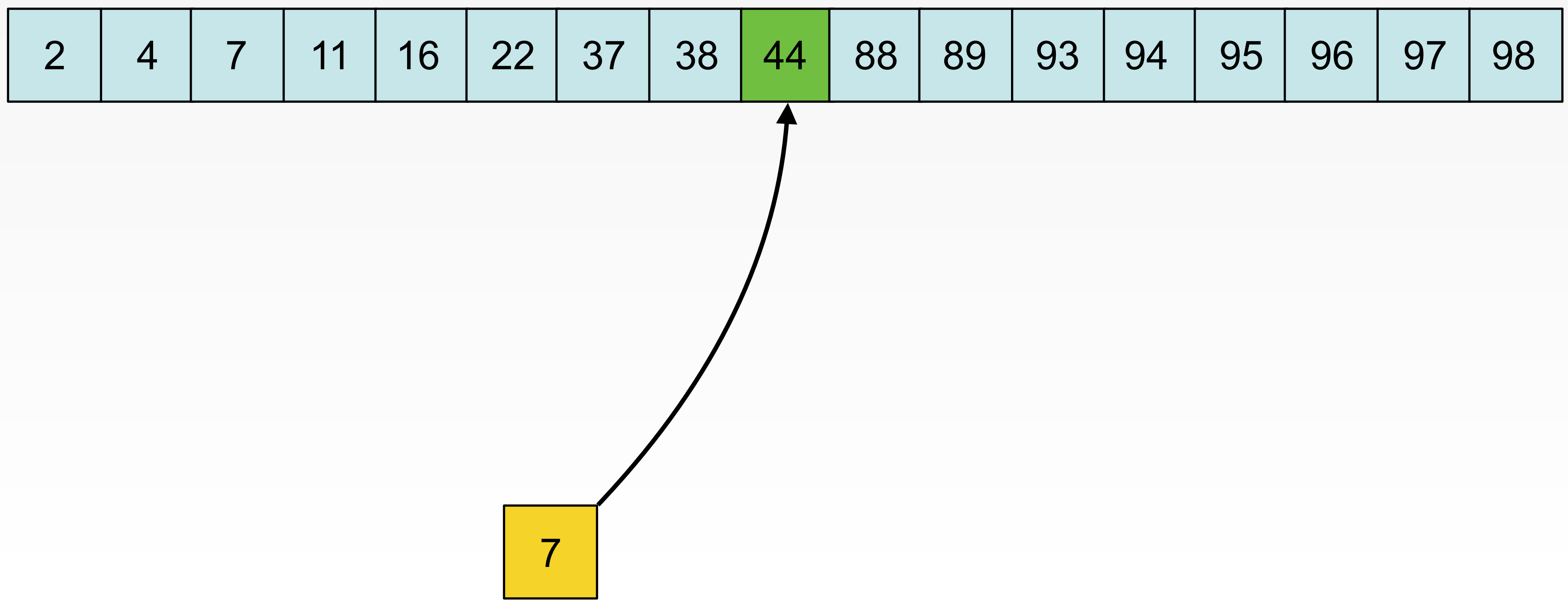
Given a sorted array of integers A[1…n], and a query q check if q is in the array.

2	4	7	11	16	22	37	38	44	88	89	93	94	95	96	97	98

7

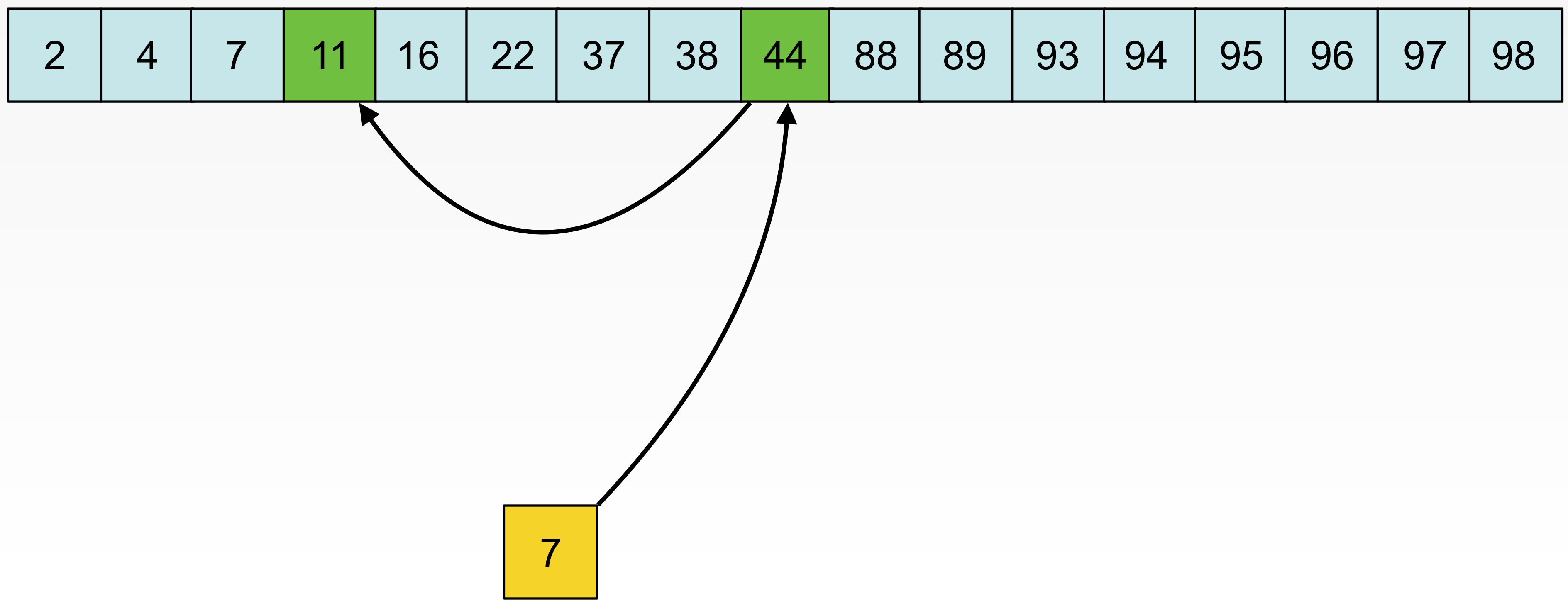
Motivating Example

Given a sorted array of integers A[1…n], and a query q check if q is in the array.



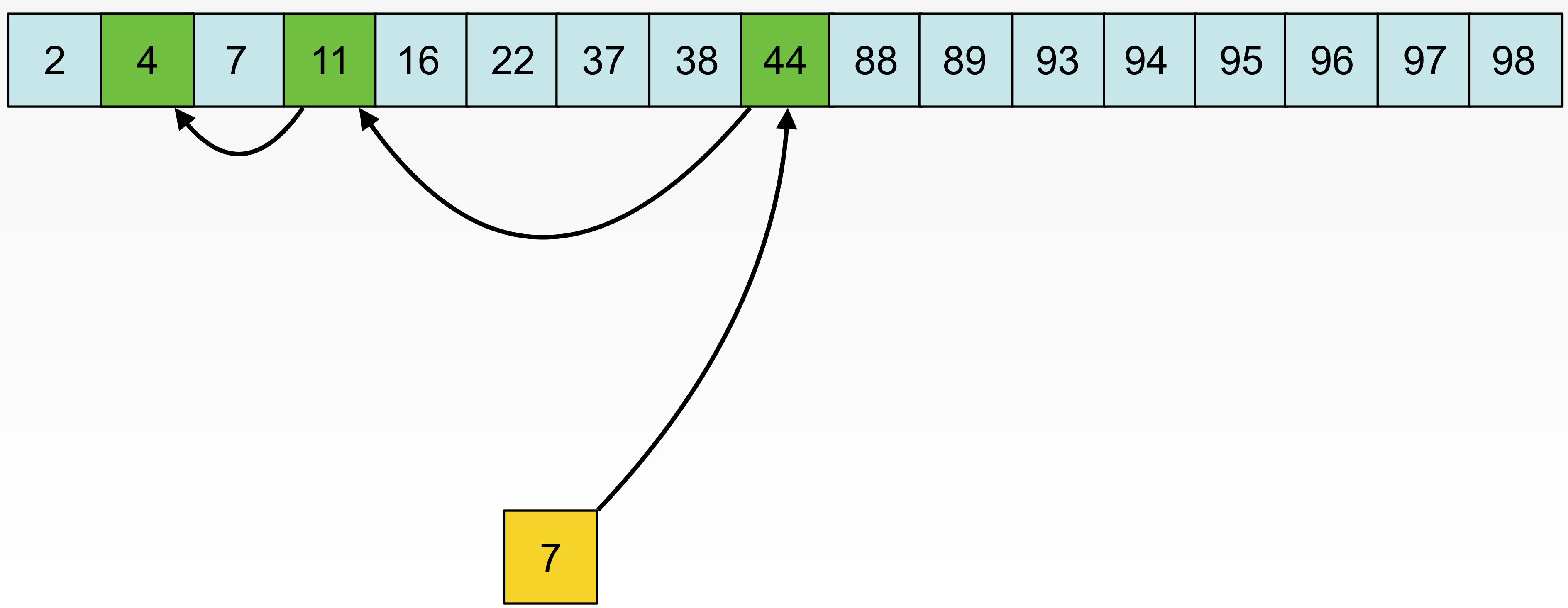
Motivating Example

Given a sorted array of integers A[1…n], and a query q check if q is in the array.



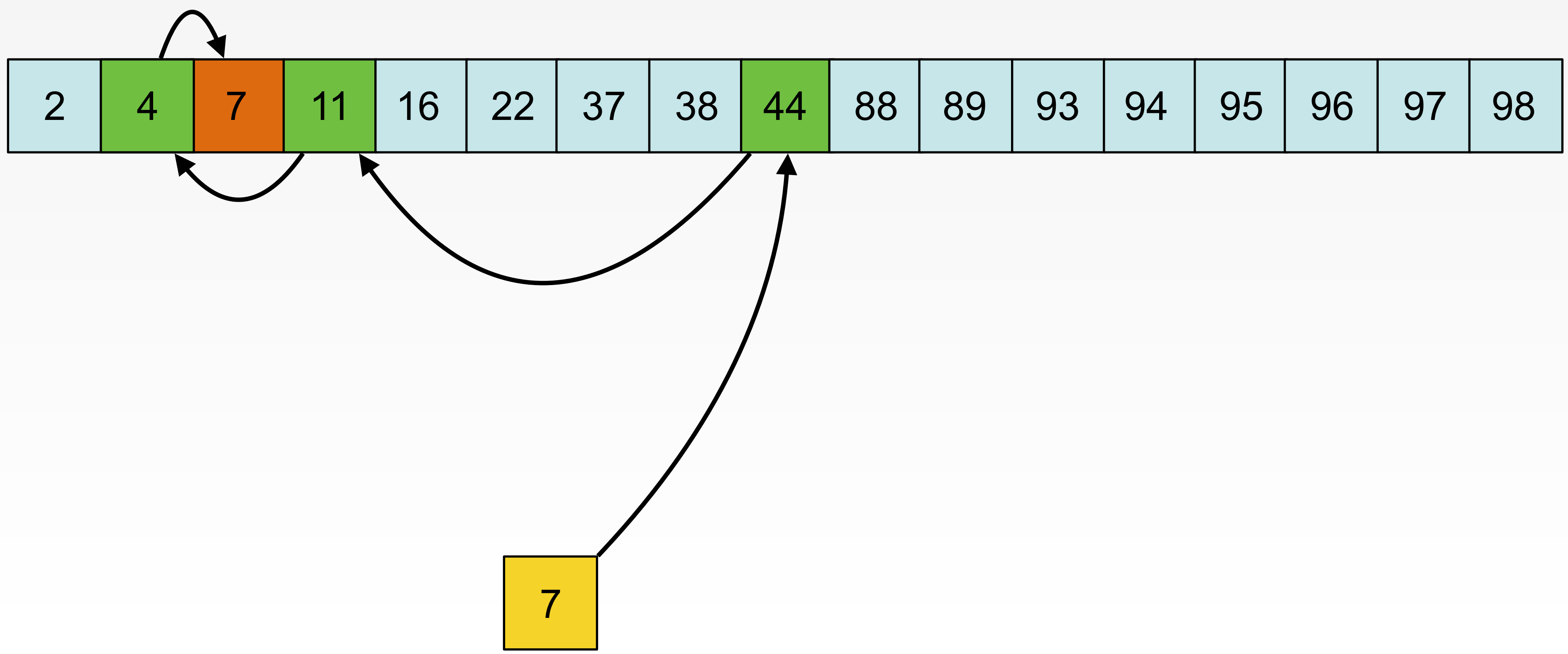
Motivating Example

Given a sorted array of integers A[1…n], and a query q check if q is in the array.



Motivating Example

Given a sorted array of integers A[1…n], and a query q check if q is in the array.



– Look up time: $O(\log n)$

Motivating Example

Given a sorted array of integers $A[1\ldots n]$, and a query q check if q is in the array.

2	4	7	11	16	22	37	38	44	88	89	93	94	95	96	97	98

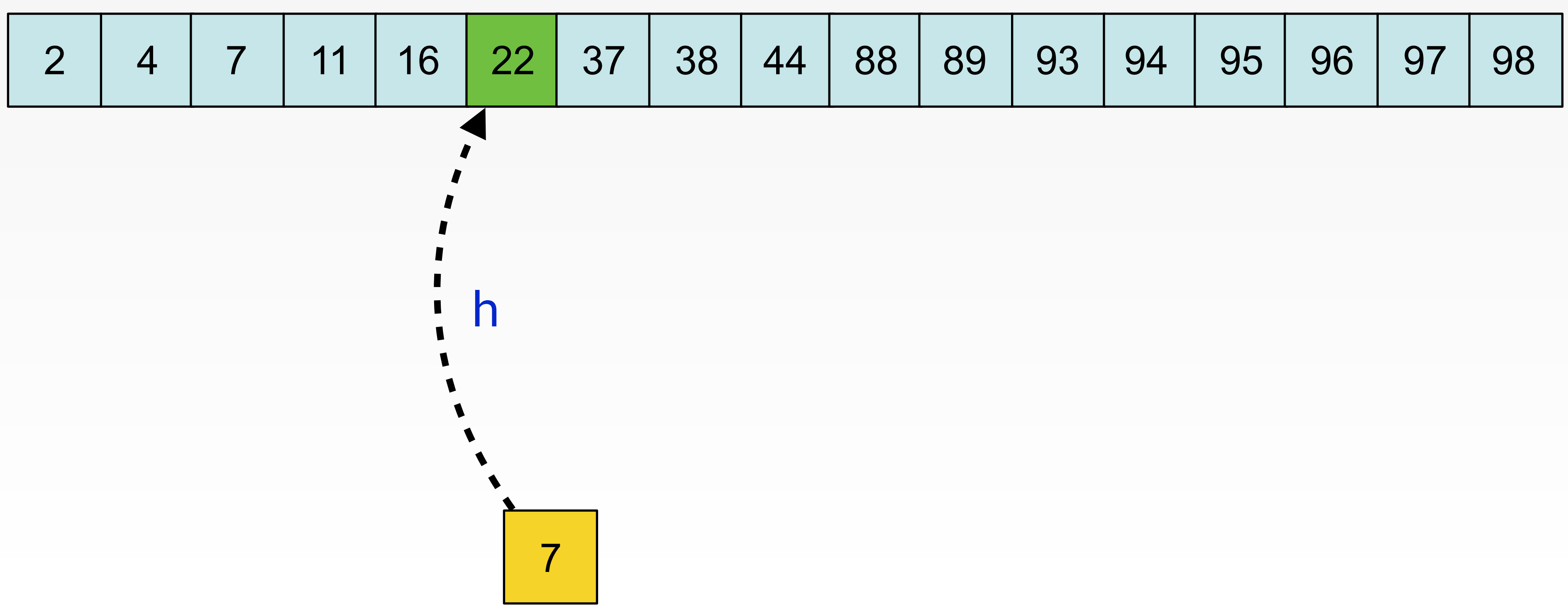
7

– Train a predictor h to learn where q should appear. [Kraska et al.'18]

– Then proceed via doubling binary search

Motivating Example

Given a sorted array of integers A[1…n], and a query q check if q is in the array.

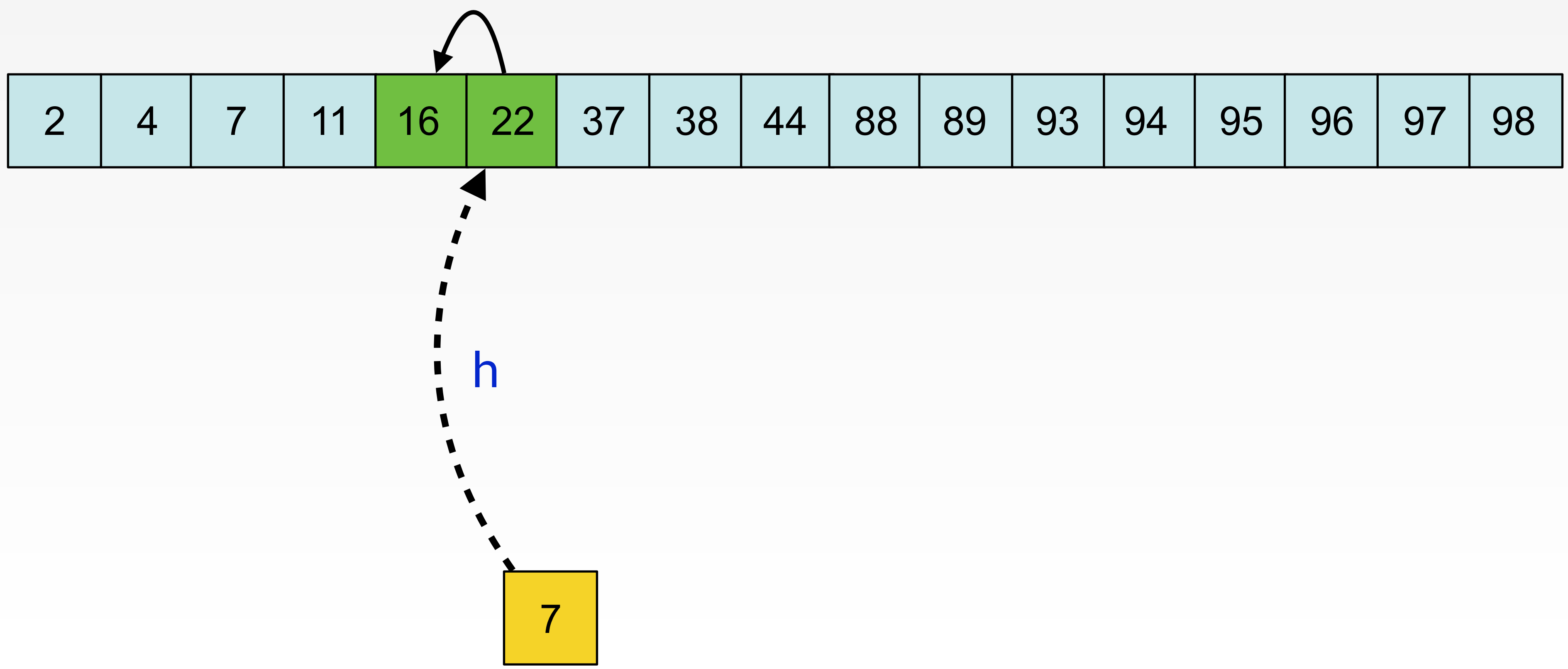


– Train a predictor h to learn where q should appear. [Kraska et al.'18]

– Then proceed via doubling binary search

Motivating Example

Given a sorted array of integers A[1…n], and a query q check if q is in the array.

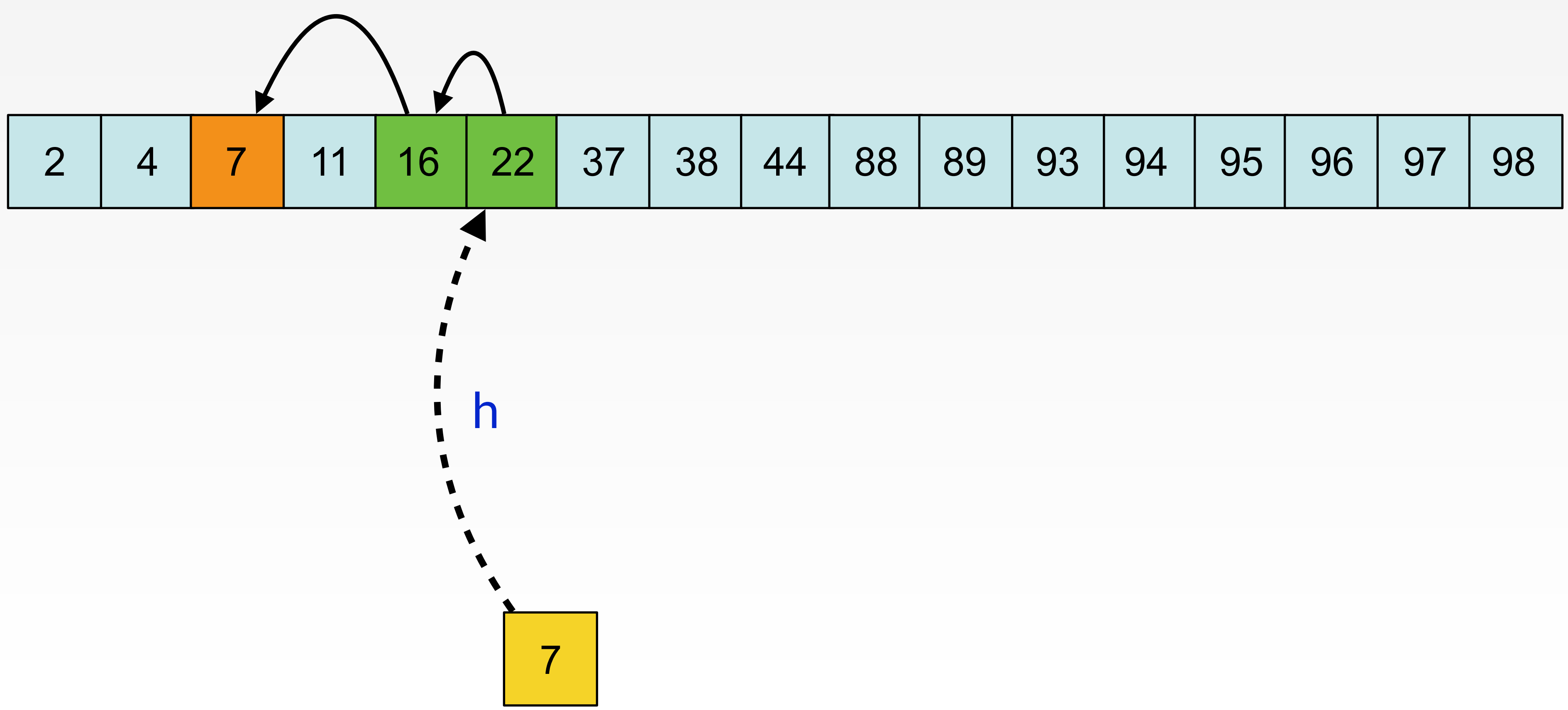


– Train a predictor h to learn where q should appear. [Kraska et al.'18]

– Then proceed via doubling binary search

Motivating Example

Given a sorted array of integers A[1…n], and a query q check if q is in the array.



– Train a predictor h to learn where q should appear. [Kraska et al.’18]

– Then proceed via doubling binary search

Empirical Slide [Kraska et al. 2018]

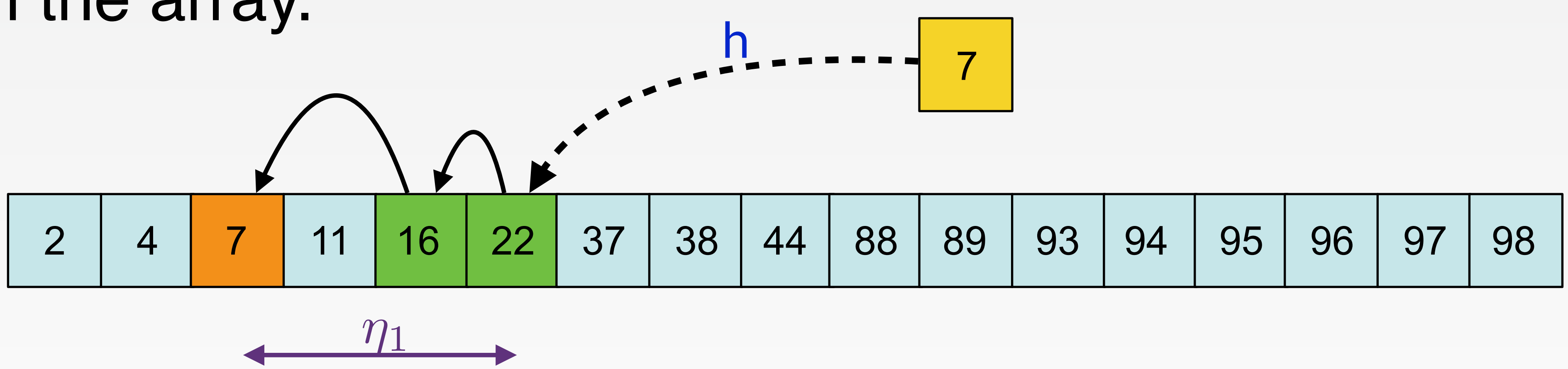
Type	Config	Map Data		
		Size (MB)	Lookup (ns)	Model (ns)
Btree	page size: 32	52.45 (4.00x)	274 (0.97x)	198 (72.3%)
	page size: 64	26.23 (2.00x)	277 (0.96x)	172 (62.0%)
	page size: 128	13.11 (1.00x)	265 (1.00x)	134 (50.8%)
	page size: 256	6.56 (0.50x)	267 (0.99x)	114 (42.7%)
	page size: 512	3.28 (0.25x)	286 (0.93x)	101 (35.3%)
Learned Index	2nd stage models: 10k	0.15 (0.01x)	98 (2.70x)	31 (31.6%)
	2nd stage models: 50k	0.76 (0.06x)	85 (3.11x)	39 (45.9%)
	2nd stage models: 100k	1.53 (0.12x)	82 (3.21x)	41 (50.2%)
	2nd stage models: 200k	3.05 (0.23x)	86 (3.08x)	50 (58.1%)

– Smaller Index

– Faster lookups when error is low, including ML cost

Motivating Example

Given a sorted array of integers A[1…n], and a query q check if q is in the array.



Analysis:

– Let $\eta_1 = |h(q) - \mathrm{OPT}(q)|$ be the error of the predicted position

– Running time: $O(\log \eta_1)$

- Can be made practical (must worry about speed & accuracy of predictions)

More on the analysis

Comparing

- Classical: $O(\log n)$
- Learning augmented: $O(\log \eta_1)$

Results:

- Consistent: perfect predictions recover optimal (constant) lookup times.
- Robust: even if predictions are bad, not (much) worse than classical

More on the analysis

Comparing

- Classical: $O(\log n)$
- Learning augmented: $O(\log \eta_1)$

Results:

- Consistent: perfect predictions recover optimal (constant) lookup times.
- Robust: even if predictions are bad, not (much) worse than classical

Punchline:

- Use Machine Learning together with Classical Algorithms to get better results.

Outline

Introduction

Motivating Example

Learning Augmented Algorithms

- Overview
- Online Algorithms
- Streaming Algorithms
- Data Structures

Conclusion

Learning Augmented Algorithms

Nascent Area with a number of recent results:

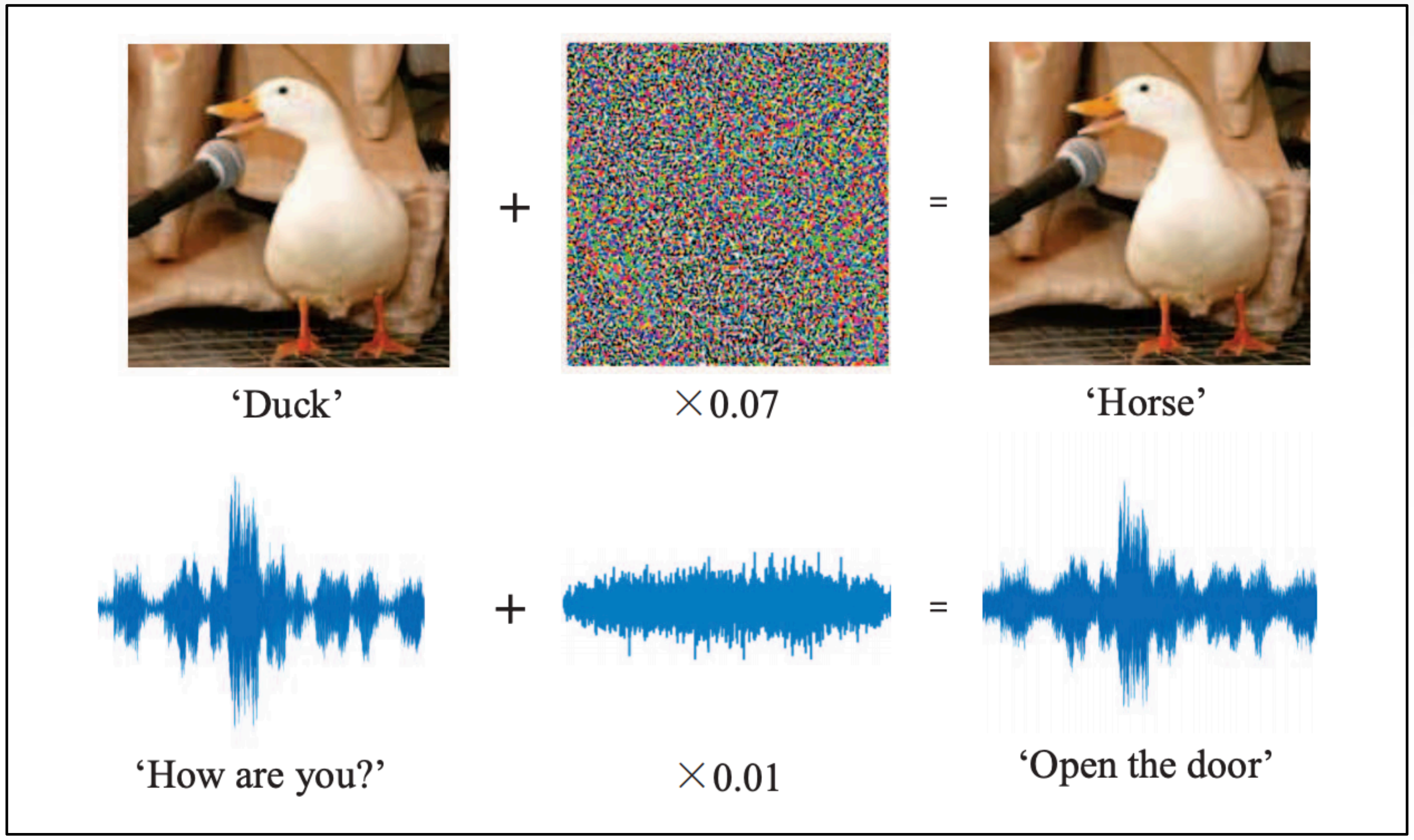
- Build better data structures
  - Indexing: Kraska et al. 2018
  - Bloom Filters: Mitzenmacher 2018
- Improve Competitive and Approximation Ratios
  - Pricing : MedinaV 2017,
  - Caching: LykourisV 2018
  - Scheduling: Kumar et al. 2018, Lattanzi et al. 2019, Mitzenmacher 2019
- Reduce running times
  - Branch and Bound: Balcan et al. 2018
- Reduce space complexity
  - Streaming Heavy Hitters: Hsu et al. 2019

Limitations of Machine Learning

Limitations of Machine Learning

Limit 1. Machine learning is imperfect.

- Algorithms must be robust to errors



Limitations of Machine Learning

Limit 1. Machine learning is imperfect.

- Algorithms must be robust to errors

Limit 2. ML is best at learning a few things

- Generalization is hard, especially with little data
- e.g. predicting the whole instance is unreasonable

Limitations of Machine Learning

Limit 1. Machine learning is imperfect.

- Algorithms must be robust to errors

Limit 2. ML is best at learning a few things

- Generalization is hard, especially with little data
- e.g. predicting the whole instance is unreasonable

Limit 3. Most ML minimizes a few different functions

- Squared loss is most popular
- Esoteric loss functions are hard to optimize (e.g. pricing)

But.. the power of ML

Machine learning reduces uncertainty

- Image recognition : uncertainty of what is in the image
- Click prediction: uncertainty about which ad will be clicked
- …

Online Algorithms with ML Advice

Augment online algorithms with some information about the future.

Goals:

- If the ML prediction is good : algorithm should perform well
  - Ideally: perfect predictions lead to competitive ratio of 1
- If the ML prediction is bad : revert back to the non augmented optimum
  - Then trusting the prediction is "free"

- Isolate the role of the prediction as a plug and play mechanism.
  - Allow to plug in richer ML models.
  - Ensure that better predictions lead to better algorithm performance.

Online Algorithms with ML Advice

Augment online algorithms with some information about the future.

Not a new idea:

- Advice Model : minimize the number of bits of perfect advice to recover OPT
- Noisy Advice: minimize the number of bits of imperfect advice to recover OPT

What is new:

- Look at quality of natural prediction tasks rather than measuring # of bits.

Outline

Introduction

Motivating Example

Learning Augmented Algorithms

- Overview
- *Online Algorithms: Paging*
- Streaming Algorithms: Heavy Hitters
- Data Structures: Bloom Filters

Conclusion

Caching (aka Paging)

Caching problem:

Have a cache of size k.

Elements arrive one a time.

- If arriving element is in the cache: cache hit, cost 0.
- If arriving element is not in the cache. Cache miss. Pay cost of 1.
  - Evict one element from the cache, and place the arriving element in its slot

State of the Art (in theory)

Bad News:

- Any deterministic algorithm is k-competitive
- There exist randomized algorithms that are $\log k$ competitive
- But no better competitive ratio is possible

A bit unsatisfying:

- Would like a constant competitive algorithm
- Would like to use theory to guide us in selection of a good algorithm

ML Advice

What kind of ML predictions would be helpful?

ML Advice

What kind of ML predictions would be helpful?

Generally:

- The richer the prediction space, the harder it is to learn
- Lots of learning theory results quantifying this exactly

- Intuition: need enough examples for every possible outcome.

ML Advice

What kind of ML predictions would be helpful?

Generally:

- The richer the prediction space, the harder it is to learn
- Lots of learning theory results quantifying this exactly

- Intuition: need enough examples for every possible outcome

What to predict for caching?

Offline Optimum

What is the offline optimum solution?

Offline Optimum

What is the offline optimum solution?

Simple greedy scheme (Belady's rule)

- Evict element that reappears <u>furthest</u> in the future

- Intuition: greedy stays ahead (makes fewest evictions) as compared to any other strategy.

What to Predict?

What do we need to implement Belady's rule?

Predict: the next appearance time of each element upon arrival.

Notes:

- One prediction at every time step
- No need to worry about consistency of predictions from one time step to the next

Measuring Error

Tempting:

- Use the performance of the predictor, h, in the caching algorithm

Better:

- Use a standard error function
- For example squared loss, absolute loss, etc.

Why Better?

- Most ML methods are used to optimize squared loss
- Want the training to be independent of how the predictor is used
- Decomposes the problem into (i) find a good prediction and (ii) use this prediction effectively

A bit more formal

Optimum Algorithm:

– Always evict element that appears furthest in the future.

Prediction:

– Every time an element arrives, predict when it will appear next

– Today consider absolute loss:

$$\eta = \sum_i |h(i) - t(i)|$$

Actual Arrival Time (integral)

Predicted Arrival Time

Using the predictions

Now have a prediction. What’s next?

Blindly Following the Oracle

Algorithm:

– Evict element that is predicted to appear furthest in the future

Blindly Following the Oracle

Elements

- x in position 2r
- y in position 2r+1
- c at position 1,T

Predictions of next arrival

- For x : always correct
- For y : always correct
- For c : 1

c x y x y x y x y x y x y x y x y … c

Evict Element Predicted Furthest in the Future

Blindly Following the Oracle

Elements

- x in position 2r
- y in position 2r+1
- c at position 1,T

Predictions of next arrival

- For x : always correct
- For y : always correct
- For c : 1

`c x y x y x y x y x y x y x y x y … c`

↑

Algorithm :

- [t = 2] Initial Cache: `[c,x]`

Evict Element Predicted Furthest in the Future

Blindly Following the Oracle

Elements

- x in position 2r
- y in position 2r+1
- c at position 1,T

Predictions of next arrival

- For x : always correct
- For y : always correct
- For c : 1

c x y x y x y x y x y x y x y x y … c

Algorithm :

- [t = 2] Initial Cache: `[c,x]`
- [t = 3] Evict x, place y: `[c,y]`

Evict Element Predicted Furthest in the Future

Blindly Following the Oracle

Elements

- x in position 2r
- y in position 2r+1
- c at position 1,T

Predictions of next arrival

- For x : always correct
- For y : always correct
- For c : 1

c x y x y x y x y x y x y x y x y … c

Algorithm :

- [t = 2] Initial Cache: [c,x]
- [t = 3] Evict x, place y: [c,y]
- [t = 4] Evict y, place x: [c,y]
- …

Error :

- Constant on average

Evict Element Predicted Furthest in the Future

Using the Prediction

Blindly following the oracle:

- Not a good idea
- Constant average error can lead to super-constant competitive ratio

Algorithms to the rescue!

Using the Prediction

Marker Algorithm:

- In beginning of a phase all elements unmarked
- When an element arrives, mark it.
- When need to evict, pick a random unmarked element
- When all elements are marked, start a new phase, and unmark all elements
- Theorem: $2\log k$ - competitive [Fiat+'91].

Predictive Marker [LykourisV'18]

Marker Algorithm:

- In beginning of a phase all elements unmarked
- When an element arrives, mark it.
- When need to evict, pick ~~random unmarked element~~ unmarked element predicted to appear furthest in the future
- When all elements are marked, start a new phase, and unmark all elements

Predictive Marker [LykourisV'18]

Marker Algorithm:

- In beginning of a phase all elements unmarked
- When an element arrives, mark it.
- When need to evict, pick ~~random unmarked element~~ unmarked element predicted to appear furthest in the future
- When all elements are marked, start a new phase, and unmark all elements

Notes:

- If predictions are perfect, almost follows Belady's rule. Recover a 2-competitive algorithm.
- When predictions are terrible, algorithm is k-competitive, small tweaks can ensure log k competitive in the worst case.

Proof Intuition

What causes cache misses?

- Elements appearing that have not been seen for a long time
  - OPT has to pay for these as well
- Recent elements being evicted
  - Tried to minimize this (subject to predictions)
  - Charge these to error of the predictor
  - Phases defined by marker cap the maximum impact of errors

Analysis

Main claim:

- Suppose the absolute error of predictor during the phase is η. Then number of misses due to mispredictions is at most $O(\sqrt{\eta})$.
- Intuition: loss on two length t sequences: a,b,c,…,t and t,…,c,b,a is $\Omega(t^2)$.

Altogether:

- Given a predictor with total error η, predictive marker has competitive ratio of $O(1 + \sqrt{1 + 4\eta/OPT})$
- Can tune to recover worst case bounds: $\min(O(\frac{\sqrt{\eta/OPT}}{\epsilon}), (2+\epsilon)\log k)$

Empirical Slide

Algorithm	Britekite Competitive ratio	Citi Bike Competitive ratio
BlindOracle	2.049	2.023
LRU	1.280	1.859
Marker	1.310	1.869
Predictive Marker	1.266	1.810

Discussion:

– Blind Oracle is too sensitive to errors in the data

– LRU tends to outperform Marker (latter is too pessimistic)

– Predictive marker consistently outperforms LRU.

Online Algorithms

Other algorithms analyzed in this setting:

- Ski Rental
- Non clairvoyant job scheduling
- Online scheduling with restricted assignment
- Online matching
- Online pricing

Many open problems:

- Clustering
- Submodular Maximization
- k-server
- …

Outline

Introduction

Motivating Example

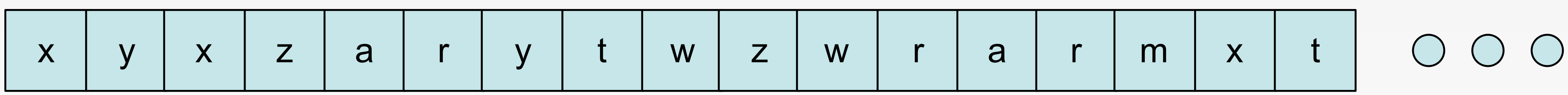
Learning Augmented Algorithms

- Overview
- Online Algorithms
- Streaming Algorithms
- Data Structures

Conclusion

Streaming Algorithms

See a never ending stream of elements, only allowed to use small (typically logarithmic) amount of memory.



Canonical question:

- Frequency estimation: compute the frequency of every element in the stream
- If elements are drawn from U trivial to do in O(|U|) space
- How to use less space?

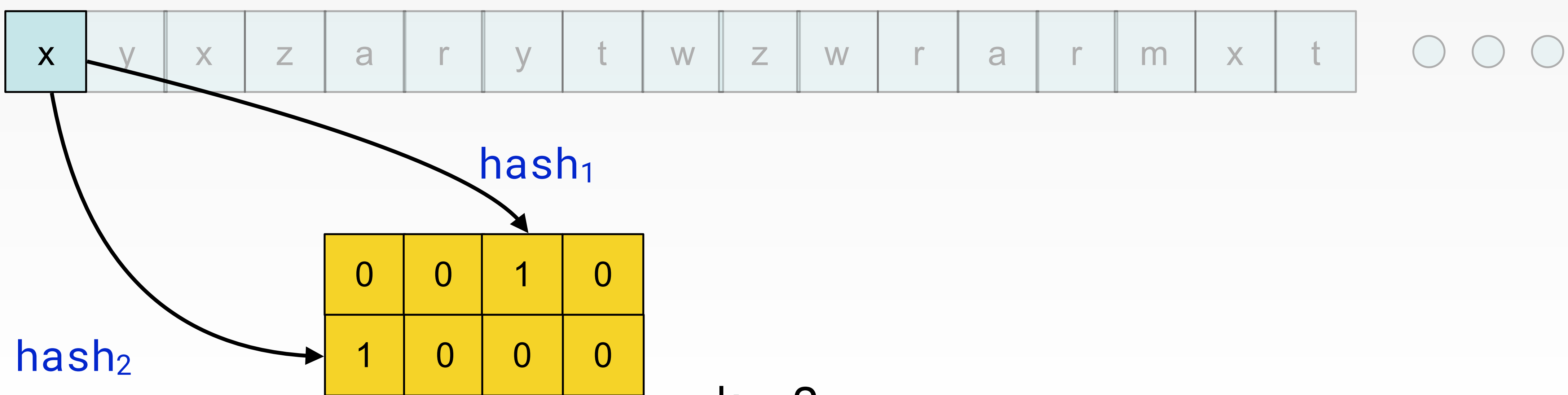
Frequency Estimation: Count Min Sketch

CountMin:

– Prepare k hash functions to use B/k buckets each.

– Keep a histogram on frequency of each hash function

– Return the minimum hashed value for any element



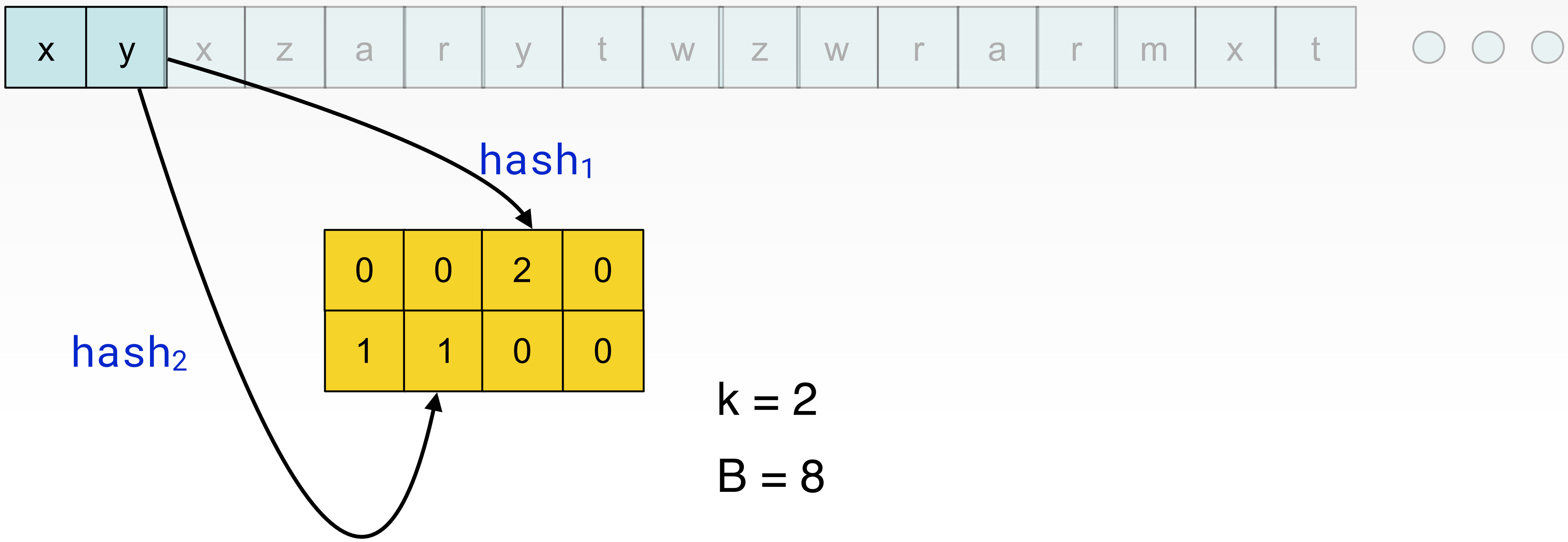
k = 2

B = 8

Frequency Estimation: Count Min Sketch

CountMin:

- Prepare k hash functions to use B/k buckets each.
- Keep a histogram on frequency of each hash function
- Return the minimum hashed value for any element



k = 2

B = 8

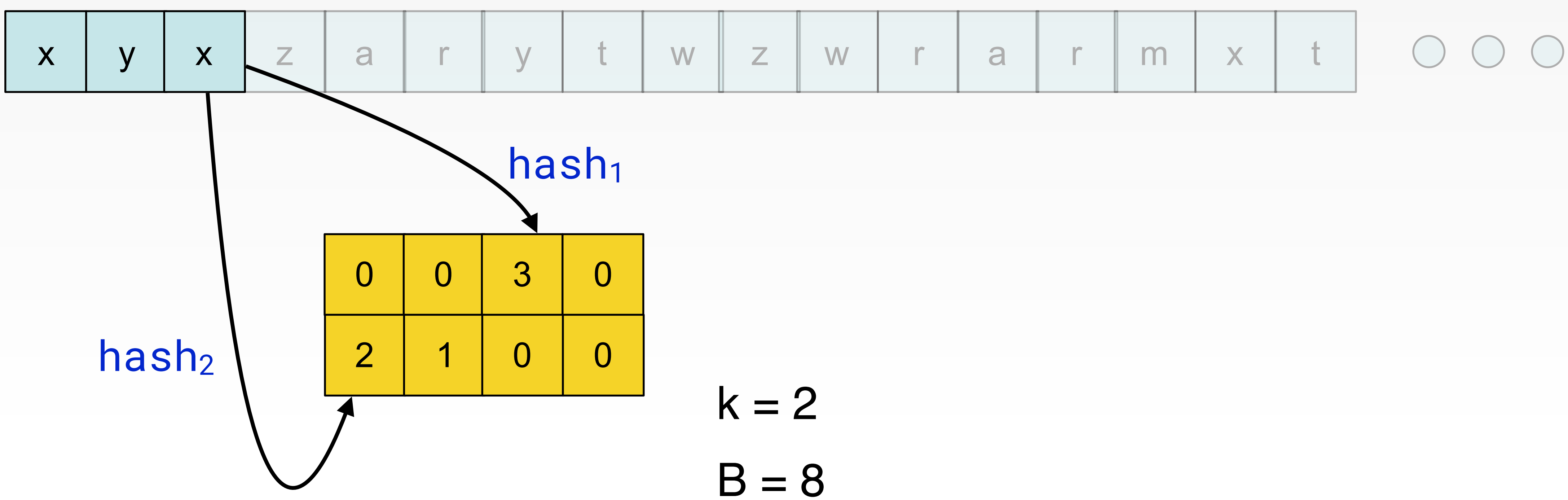
Frequency Estimation: Count Min Sketch

CountMin:

– Prepare k hash functions to use B/k buckets each.

– Keep a histogram on frequency of each hash function

– Return the minimum hashed value for any element



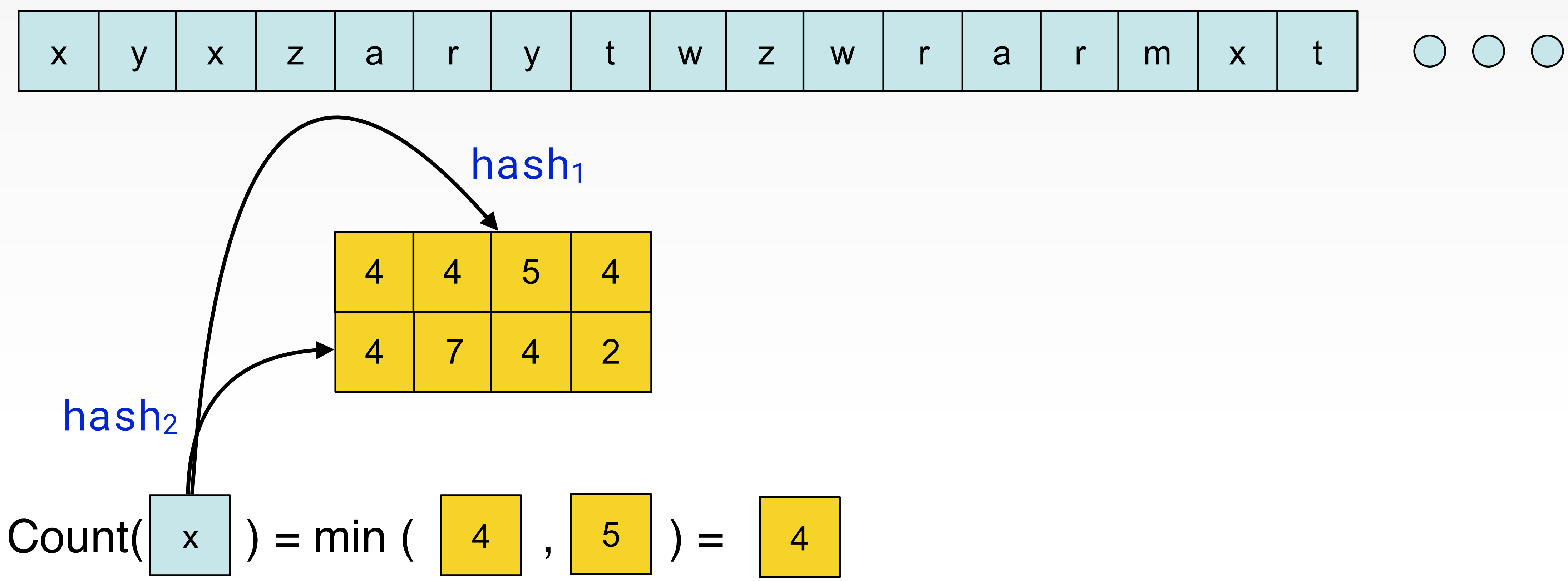
k = 2

B = 8

Frequency Estimation: Count Min Sketch

CountMin:

- Prepare k hash functions to use B/k buckets each.
- Keep a histogram on frequency of each hash function
- Return the minimum hashed value for any element



Learned CountMin [Hsu+'19]

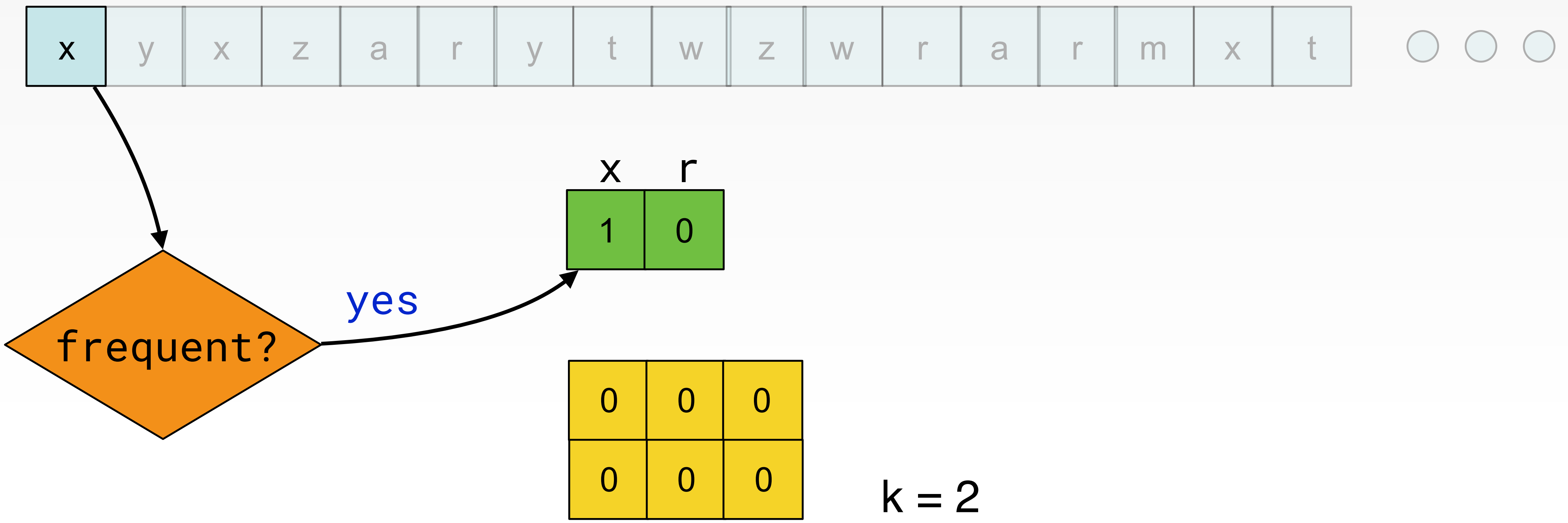
Idea:

- Train a classifier to predict whether an item is a heavy hitter
- For those predicted to be frequent elements, keep their counts exactly
- For the rest, use a CountMin sketch

Frequency Estimation: Count Min Sketch

Learned CountMin:

- Predict whether an element is frequent
- If so, keep its count exactly
- Otherwise, use CountMin



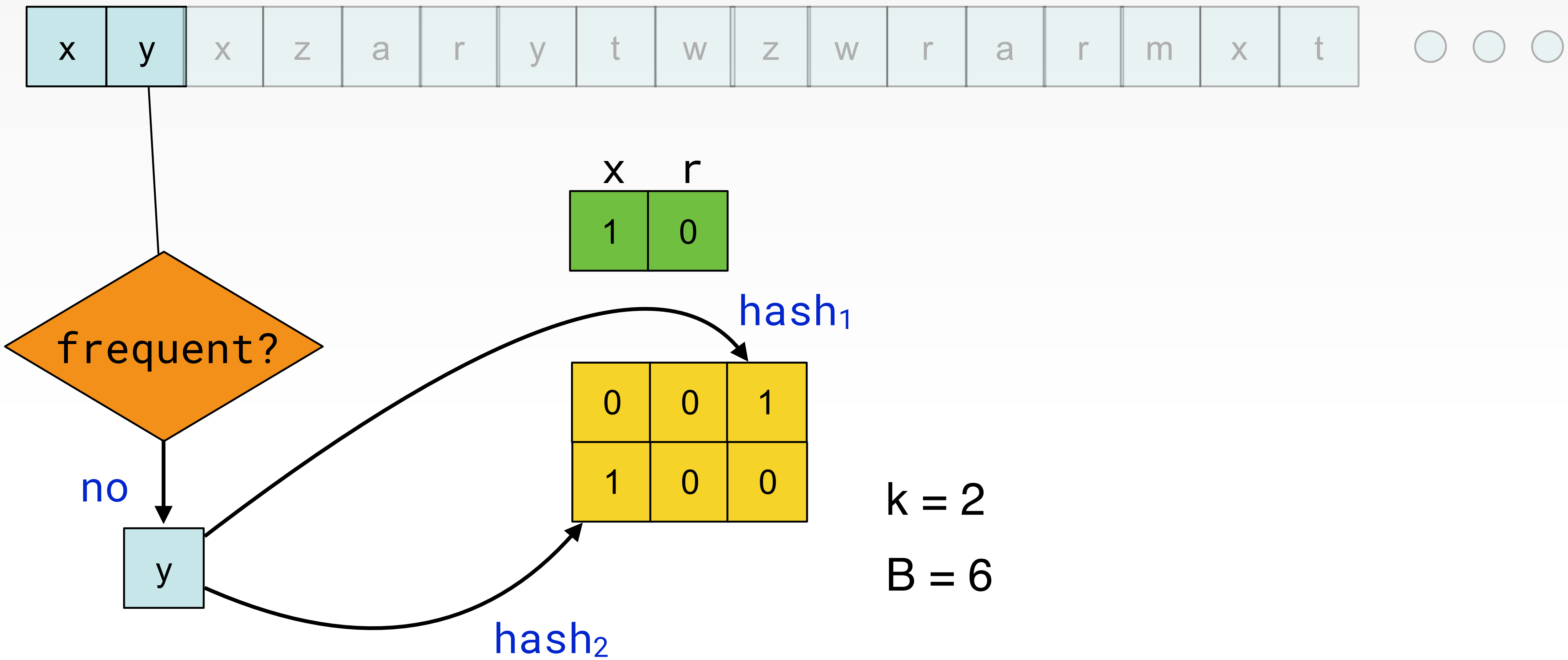
k = 2

B = 6

Frequency Estimation: Count Min Sketch

Learned CountMin:

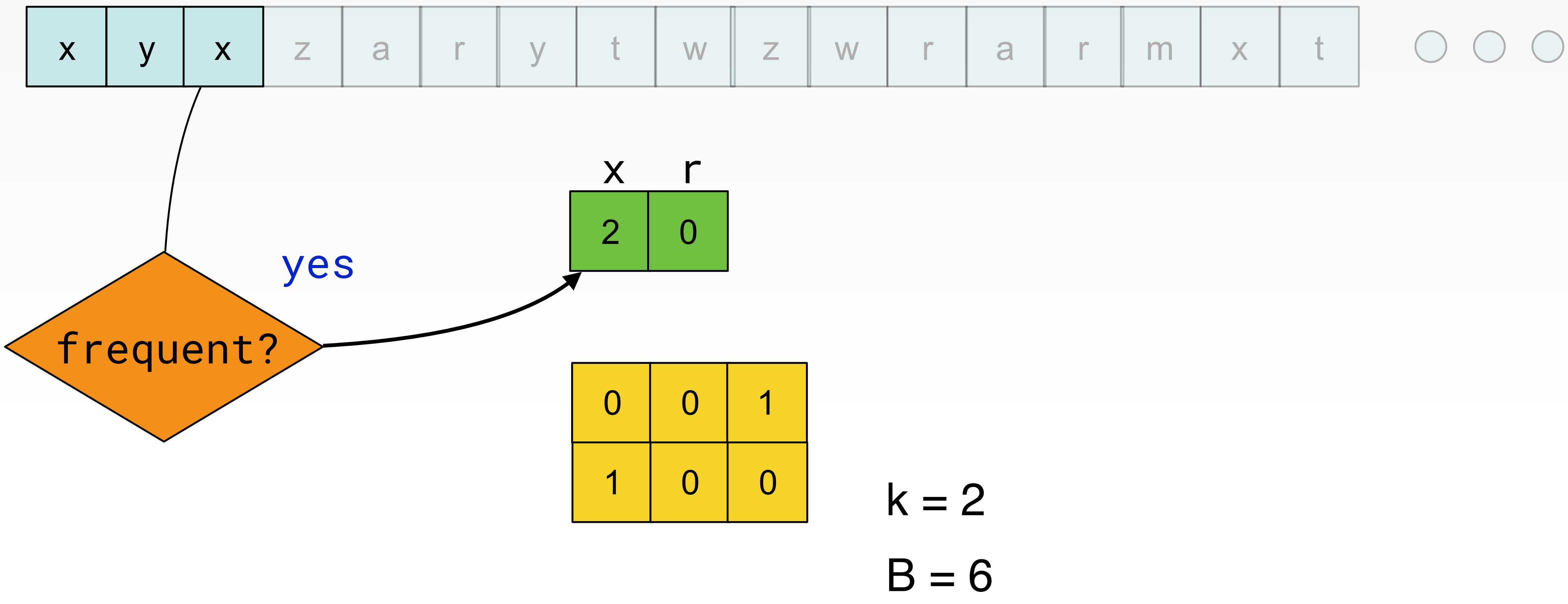
- Predict whether an element is frequent
- If so, keep its count exactly
- Otherwise, use CountMin



Frequency Estimation: Count Min Sketch

Learned CountMin:

- Predict whether an element is frequent
- If so, keep its count exactly
- Otherwise, use CountMin



Analysis

Main question:

- Space vs. Accuracy trade-off.
- Fix space of B buckets. Measure accuracy

Error Function:

- "Expected" error
- Given true counts f_i and estimated counts $\hat{f}_i$.

$$\textsc{Err}(f, \hat{f}) = \frac{1}{N} \sum_i |f_i - \hat{f}_i| \cdot f_i$$

Analysis of Learned CountMin

For Zipf Distributions:

– Vanilla Count Min: $$O\left(\frac{k \ln n \ln(\frac{kn}{B})}{B}\right)$$

– Perfect Predictions: $$O\left(\frac{\ln^2 \frac{n}{B}}{B}\right)$$

– Noisy Predictions: $$O\left(\frac{\delta^2 \ln^2 B + \ln^2 \frac{n}{B}}{B}\right)$$

Analysis of Learned CountMin

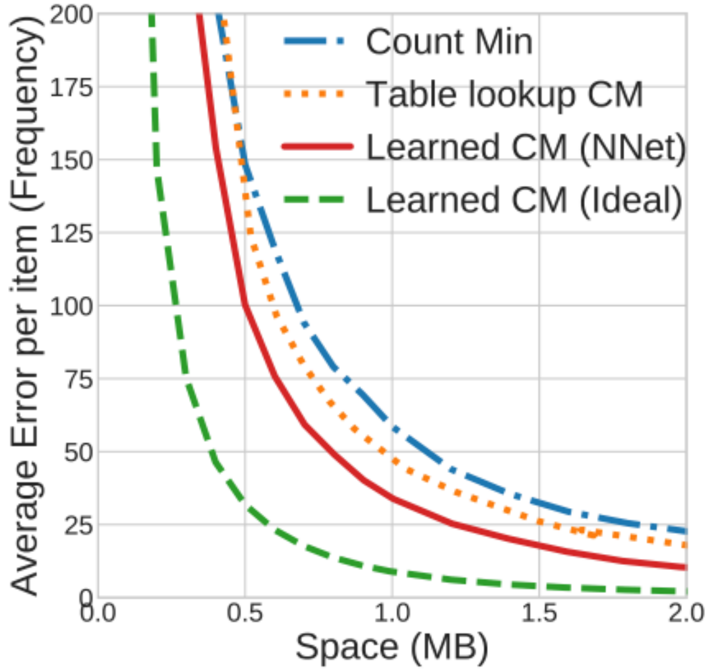
For Zipf Distributions: When $B = \Theta(n)$

– Vanilla Count Min: $O\left(\frac{k \ln n \ln(\frac{kn}{B})}{B}\right)$ $O\left(\frac{\ln n}{n}\right)$

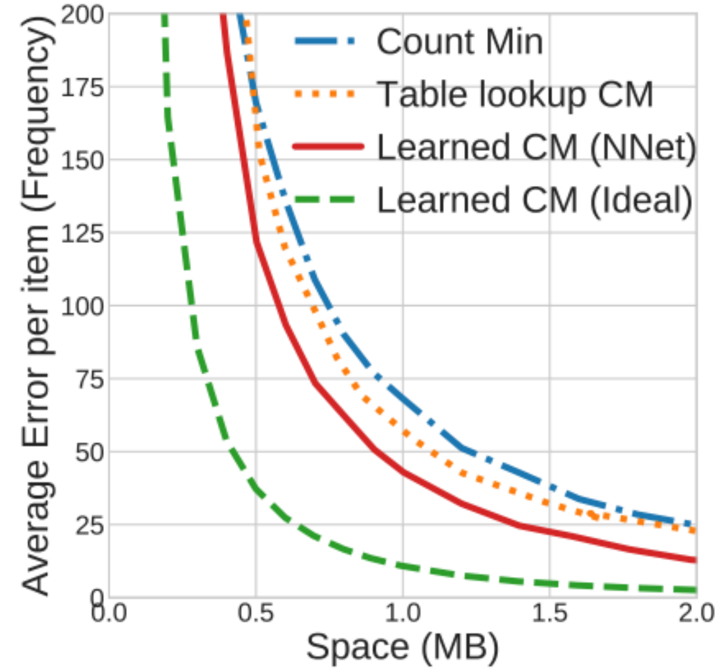
– Perfect Predictions: $O\left(\frac{\ln^2 \frac{n}{B}}{B}\right)$ $O\left(\frac{1}{n}\right)$

– Noisy Predictions: $O\left(\frac{\delta^2 \ln^2 B + \ln^2 \frac{n}{B}}{B}\right)$ $O\left(\frac{\delta^2 \ln^2 n}{n}\right)$

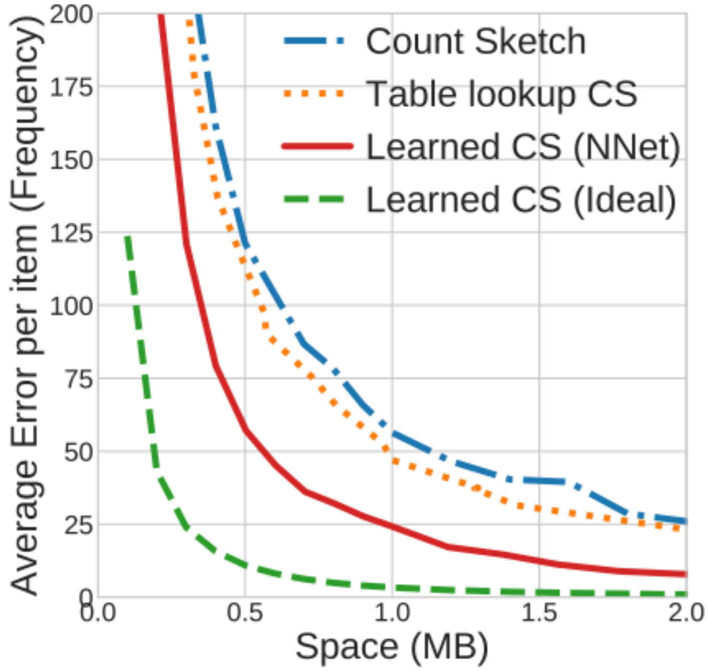
Empirical Slide



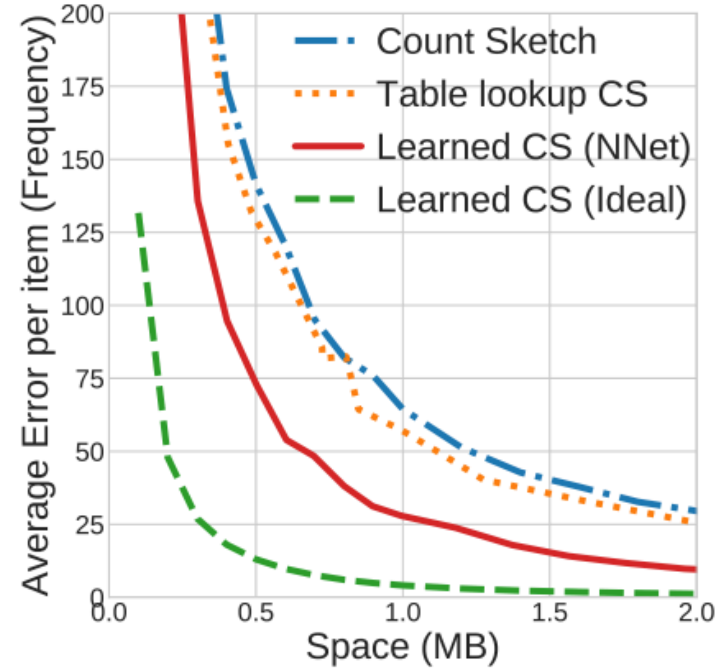
(a) Learned Count-Min - 20th test minute



(b) Learned Count-Min - 50th test minute



(c) Learned Count-Sketch - 20th test minute



(d) Learned Count-Sketch - 50th test minute

Outline

Introduction

Motivating Example

Online Algorithms

Streaming Algorithms

Data Structures

Conclusion

Outline

Already saw “learned indexes” [Kraska+’18, LykourisV’18]

- Predict offset rather than doing binary search

New idea:

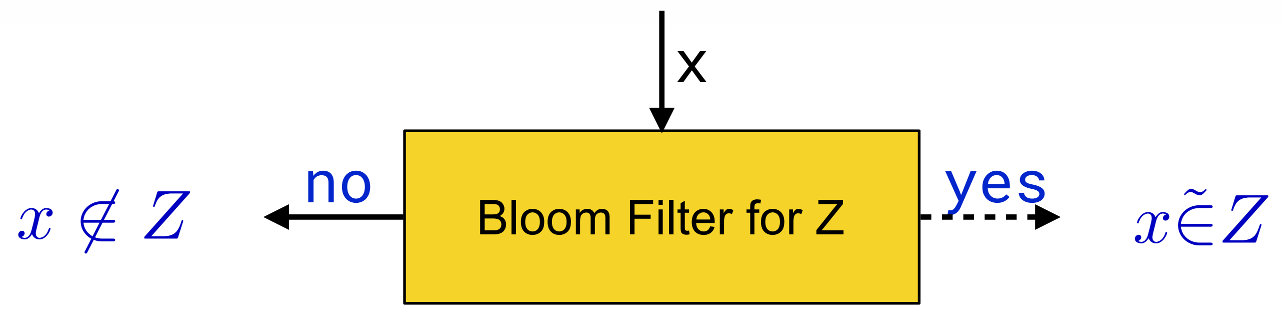
- Learned Bloom Filters.

Bloom Filters Review

Bloom Filter

- Data Structure to test set membership
- Never returns a false negative (elements in the set always returned as in the set)
- Sometimes returns a false positive (elements not in the set are claimed to be in the set)

Trade-off between space & false positive probability.



Learned Bloom Filters [Mitzenmacher ’18]

Train a predictor on whether an element is in the set.

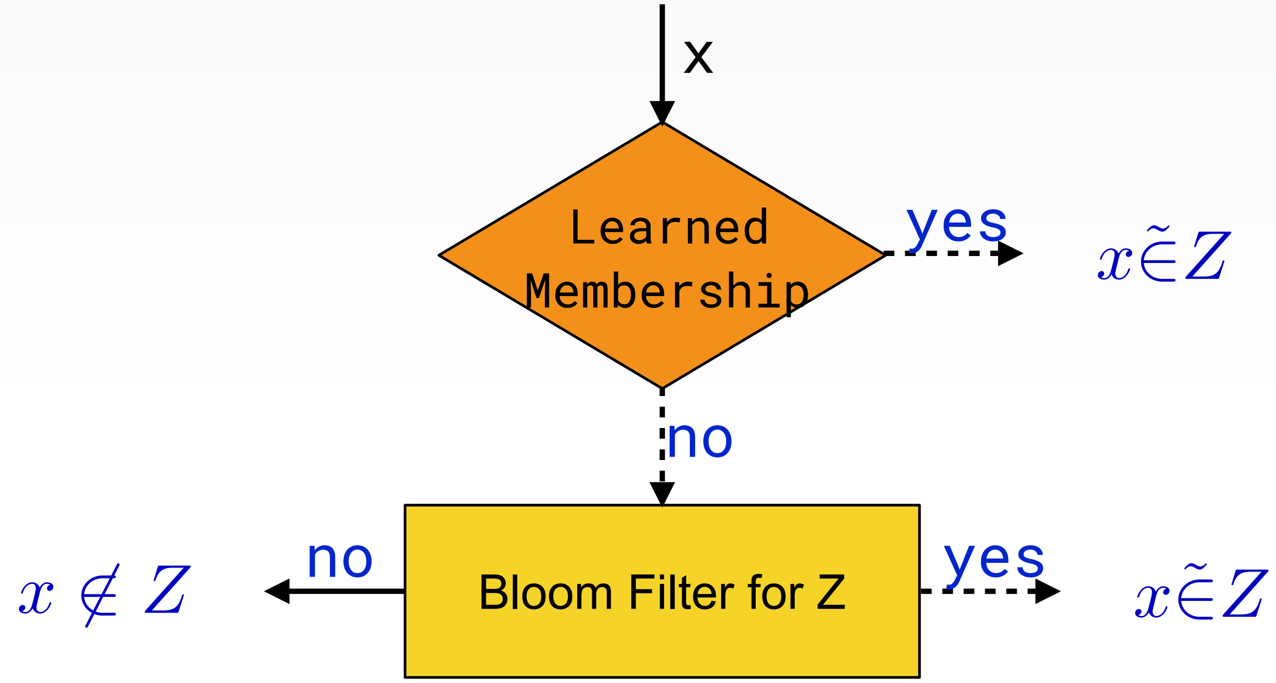
– Prediction has both false positive & false negative rates

Learned Bloom Filters

Train a predictor on whether an element is in the set.

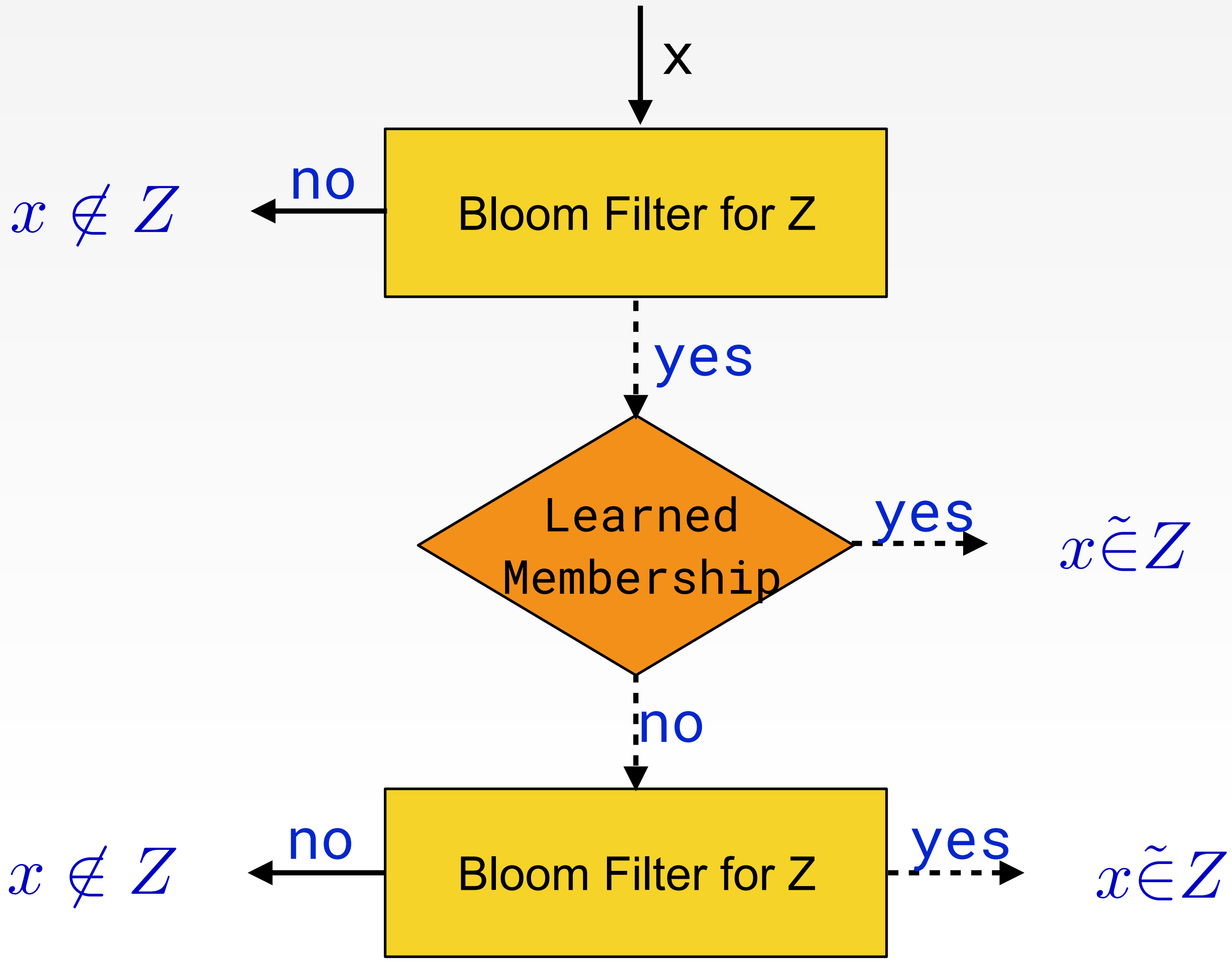
- Prediction has both false positive & false negative rates

- Combine the two:



Learned Bloom Filters

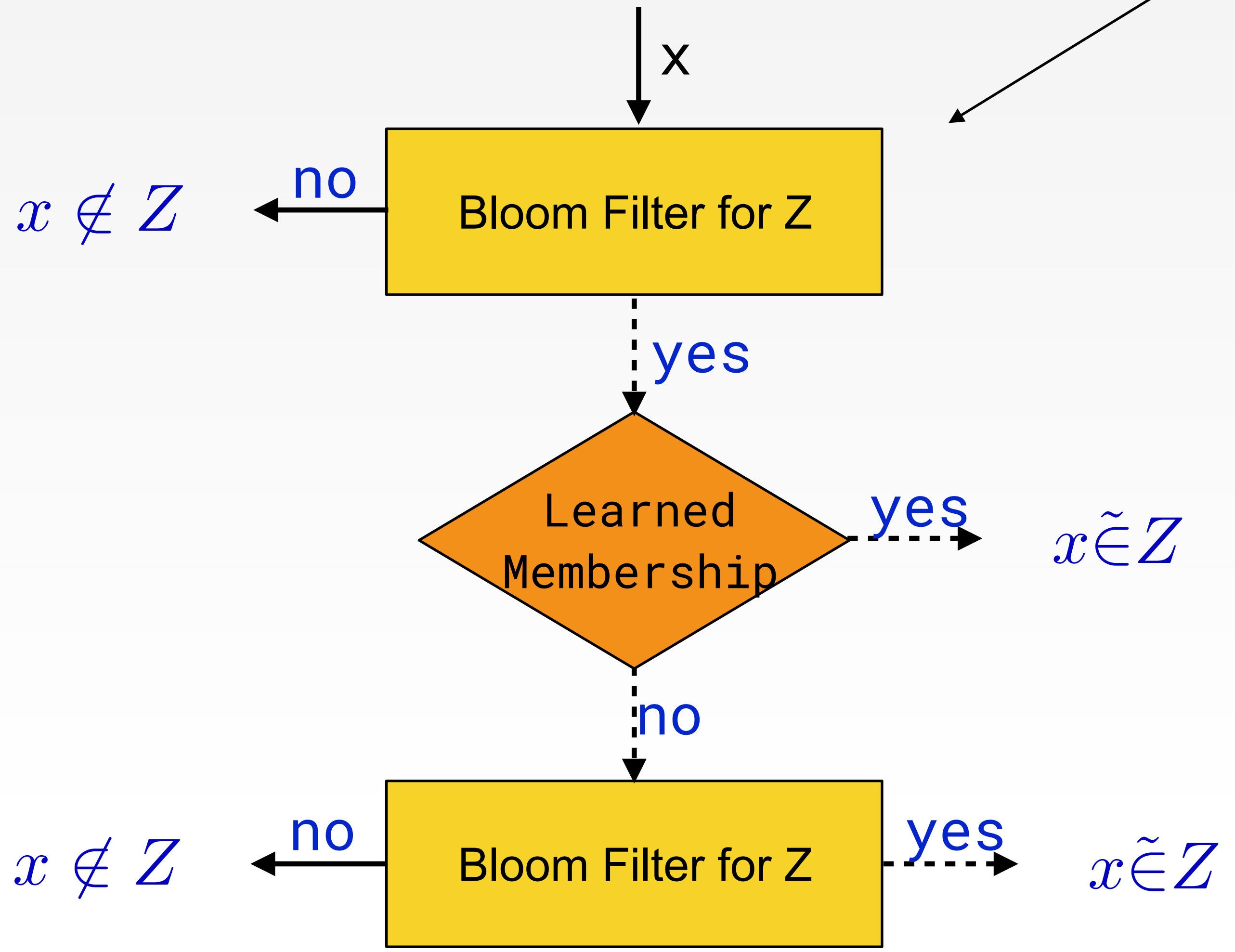
Do a step better:



Learned Bloom Filters

Do a step better:

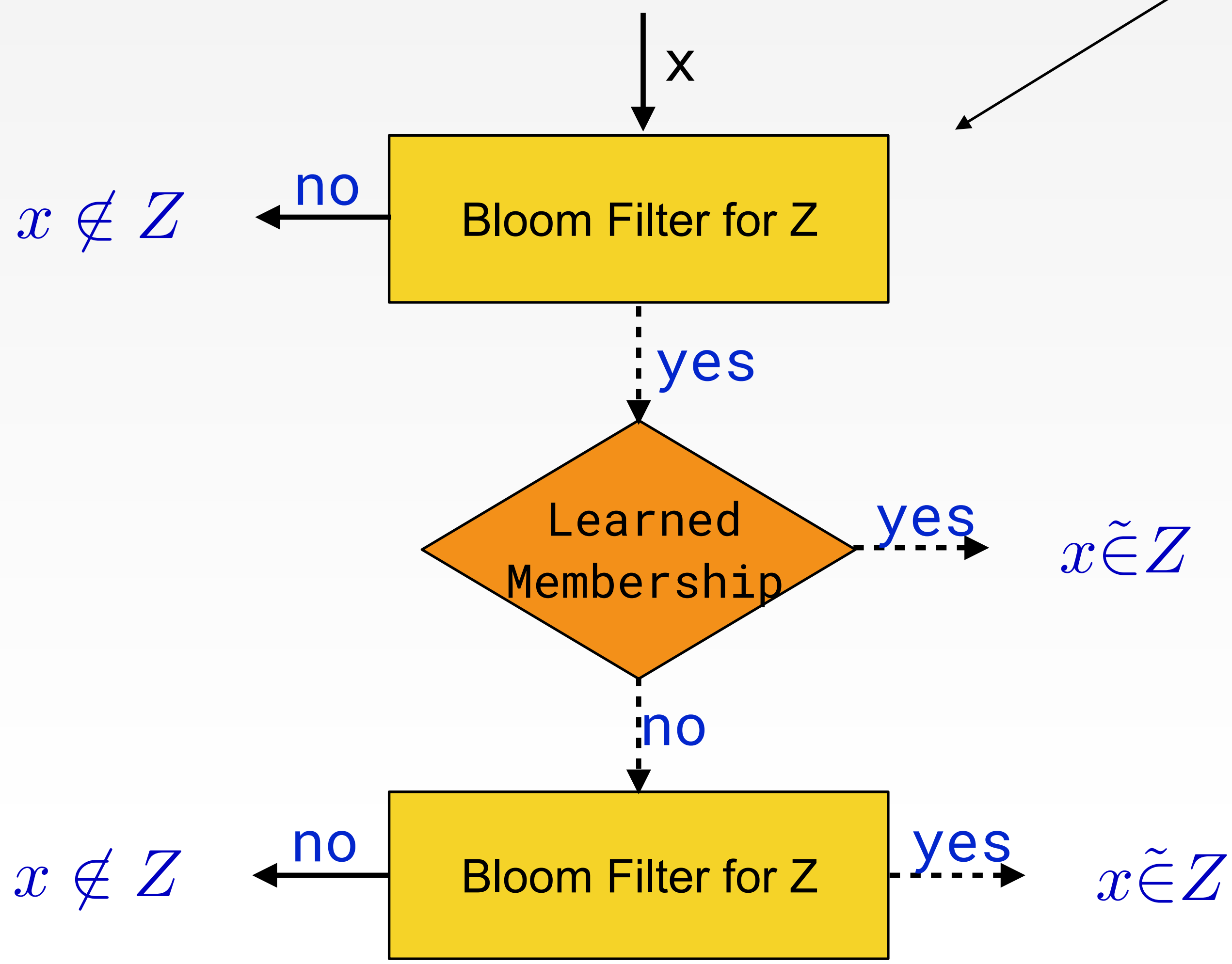
Filter out easy negatives to make learning easier



Learned Bloom Filters

Do a step better:

Filter out easy negatives to make learning easier



Back up filter to deal with prediction errors

Learned Bloom Filter Analysis

Trade-off between error rates and false positive / negative rates.

Main takeaways:

- The forward bloom filter makes the learning robust (if, for instance, examples are from a different distribution)
- The backup bloom filter does not grow with input size (it depends more on the quality of the learner)

Conclusion

Overall Question

How to incorporate (noisy, non-uniform) ML predictions to improve performance (time, space, approximation/competitive ratios) of classical algorithms.

Two Subproblems

Decide on what to predict.

- Predictions should be concise & compact
- Should use traditional loss functions

Incorporate predictions into algorithms.

- Full power of algorithm design and analysis
- Typically need a “trust but verify” approach

Final Thought

Another way to go beyond worst case analysis.

- Parametrize difficulty of the problem by the quality of the prediction
- Formally cast heuristics (e.g. LRU) as learning problems and evaluate their quality

Thank You