NOT EXACTLY! APPROXIMATE ALGORITHMS FOR BIG DATA

FANGJIN YANG · DRUID COMMITTER · METAMARKETS

NELSON RAY · QUANTITATIVE ANALYST · GOOGLE

OVERVIEW

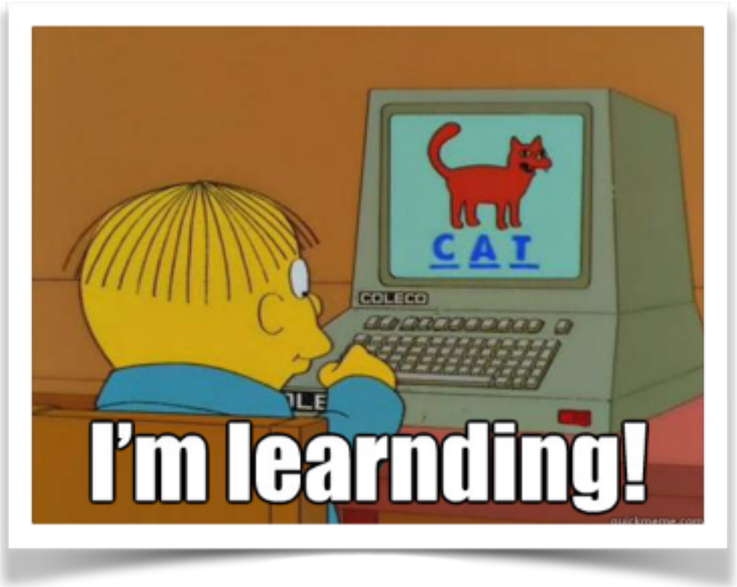
THE PROBLEM	MANAGE DATA COST EFFICIENTLY
THE DATA	DEALING WITH EVENT STREAMS
SIMPLIFYING STORAGE	DATA SUMMARIZATION
FINDING UNIQUES	HYPERLOGLOG
ESTIMATING DISTRIBUTION	APPROXIMATE HISTOGRAMS

THE PROBLEM

Real-time Bidding

PROBLEMS

- Storing/processing billions of rows is expensive
- Reduce storage, improve performance
- Reduce storage by throwing away information
- Throwing away information reduces accuracy



THE DATA

THE DATA

Timestamp	Bid Price
2013-10-28T02:13:43Z	1.19
2013-10-28T02:14:21Z	0.05
2013-10-28T02:55:32Z	1.04
2013-10-28T03:07:28Z	0.16
2013-10-28T03:13:43Z	1.03
2013-10-28T04:18:19Z	0.15
2013-10-28T05:36:34Z	0.01
2013-10-28T05:37:59Z	1.03

DATA SUMMARIZATION

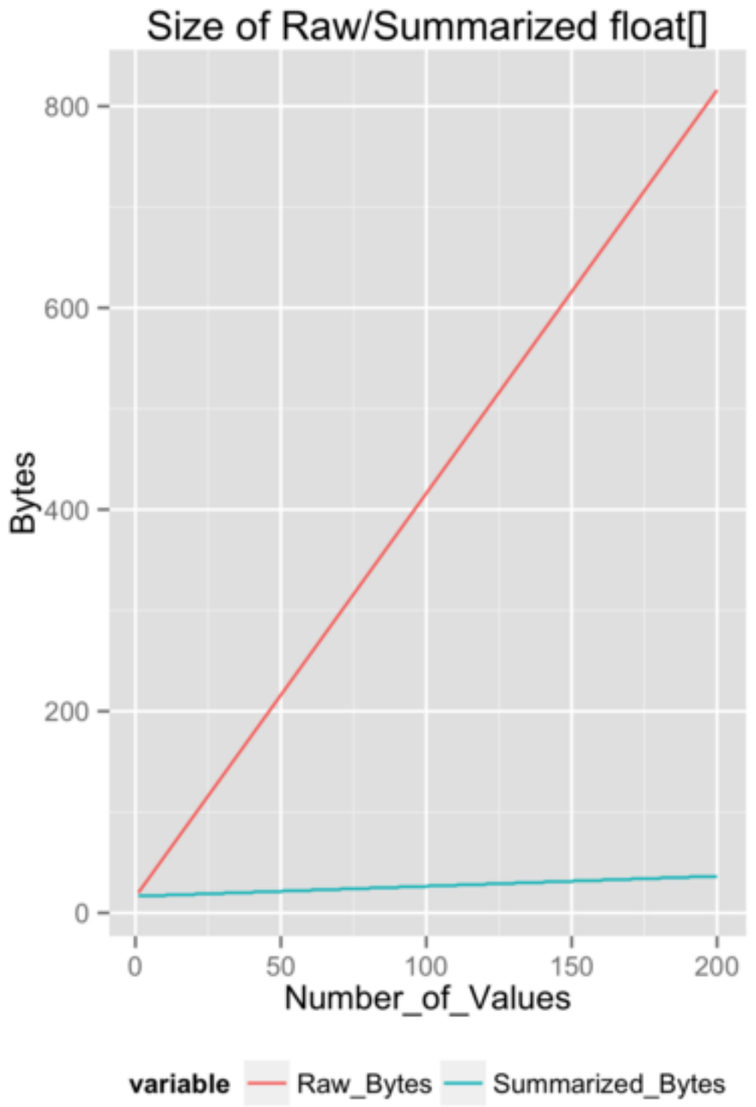
Timestamp	Bid Price
2013-10-28T02:13:43Z	1.19
2013-10-28T02:14:21Z	0.05
2013-10-28T02:55:32Z	1.04
2013-10-28T03:07:28Z	0.16
2013-10-28T03:13:43Z	1.03
2013-10-28T04:18:19Z	0.15
2013-10-28T05:36:34Z	0.01
2013-10-28T05:37:59Z	1.03

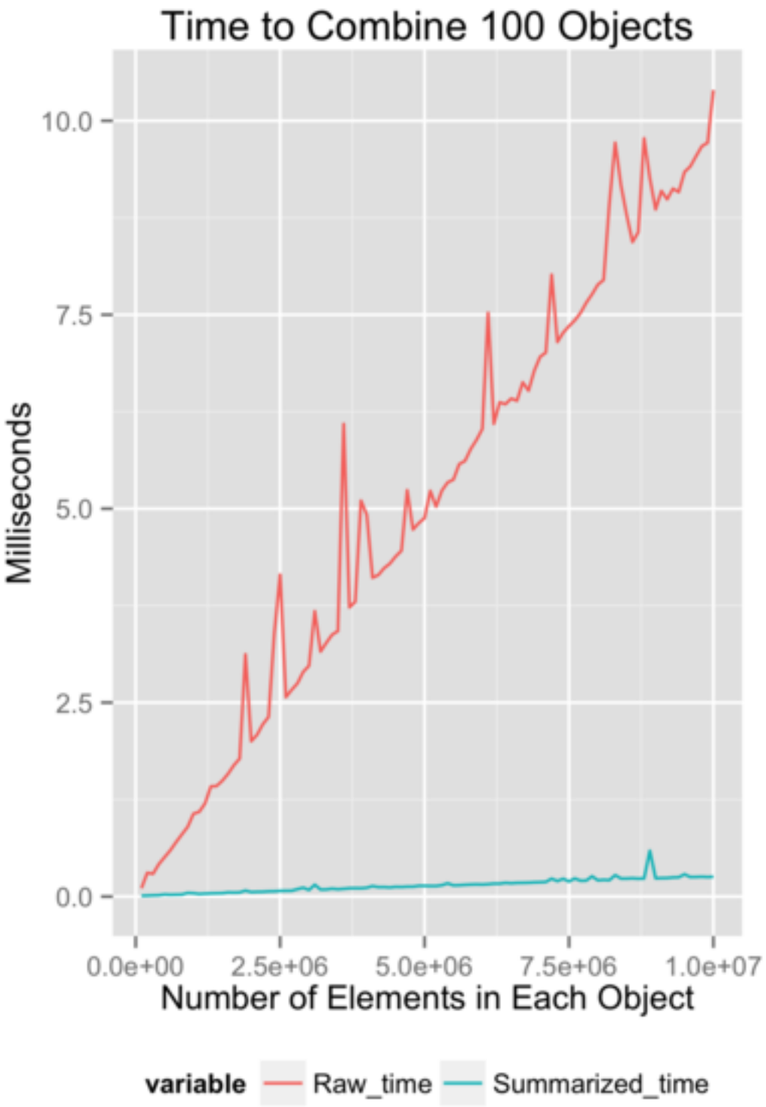
Timestamp	Revenue	Number of Prices
2013-10-28T02	2.28	3
2013-10-28T03	1.19	2
2013-10-28T04	0.15	1
2013-10-28T05	1.04	2

COMBINING SUMMARIZATIONS

Timestamp	Revenue	Number of Prices
2013-10-28T02	2.28	3
2013-10-28T03	1.19	2
2013-10-28T04	0.15	1
2013-10-28T05	1.04	2

Timestamp	Revenue	Number of Prices
2013-10-28	4.66	8





Fangjin Yang & Nelson Ray 2014

SUMMARIZATION SUMMARY

- Throw away information about individual events
- Drastically reduce storage and improve query speed
  - On average, 40x reduction in storage on with our own data
- We've lost info about individual prices
- Data summarization is not always trivial

Fangjin Yang & Nelson Ray 2014

CASE STUDY 1

CASE STUDY 1

- Problem: determine unique number of elements in a set
- Use case: measuring number of unique users

DATA

BIG DATA

EXACT SOLUTION

- Store every single username (in a Java HashSet)
- No loss of information, no accuracy tradeoff

HASHSET

Timestamp	Username
2013-10-28T02:13:43Z	user1
2013-10-28T02:14:21Z	user2
2013-10-28T02:55:32Z	user1
2013-10-28T03:07:28Z	user4
2013-10-28T03:13:43Z	user97
2013-10-28T04:18:19Z	user2
2013-10-28T05:36:34Z	user9834
2013-10-28T05:37:59Z	user97

Timestamp	Usernames
2013-10-28T02	{user1, user2}
2013-10-28T03	{user4, user97}
2013-10-28T04	{user2}
2013-10-28T05	{user9834, user97}

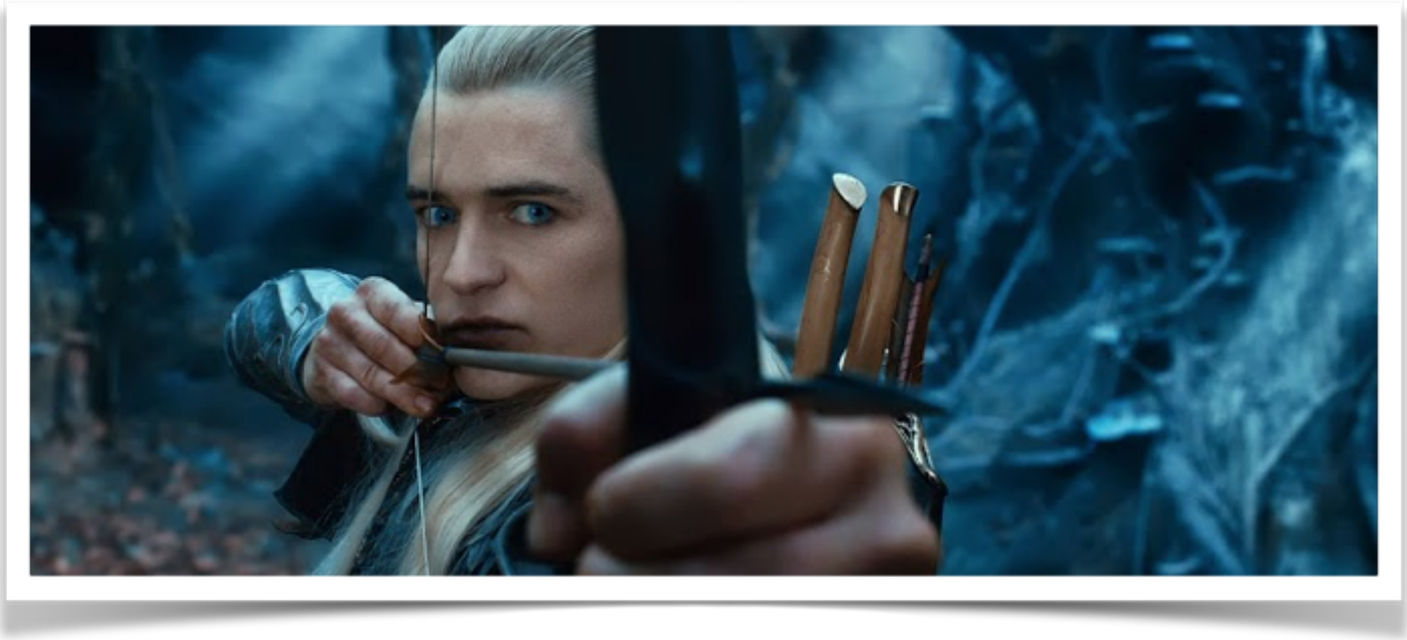
HASHSET

Timestamp	Usernames
2013-10-28T02	{user1, user2}
2013-10-28T03	{user4, user97}
2013-10-28T04	{user2}
2013-10-28T05	{user9834, user97}

Timestamp	Usernames
2013-10-28	{user1, user2, user4, user97, user9834}

EXACT SOLUTION

- Storage/Computation: O(# uniques)
- We're not throwing away any information about usernames
- Accuracy: 100%



INFEASIBLE STORAGE

- High cardinality user dimensions == infeasible storage
  - Storage cost for 10^9 unique elements == ~48GB of storage

CARDINALITY ESTIMATION

- Plenty of literature
  - Linear Counting
  - Count-Min Sketch
  - LogLog

HYPERLOGLOG

- Storage: 1.5 KB (for cardinalities 10^9 and above)
  - 99.999997% decrease in storage size
- Computation: O(1) (for cardinalities < ~10^10)
- Accuracy: 97%

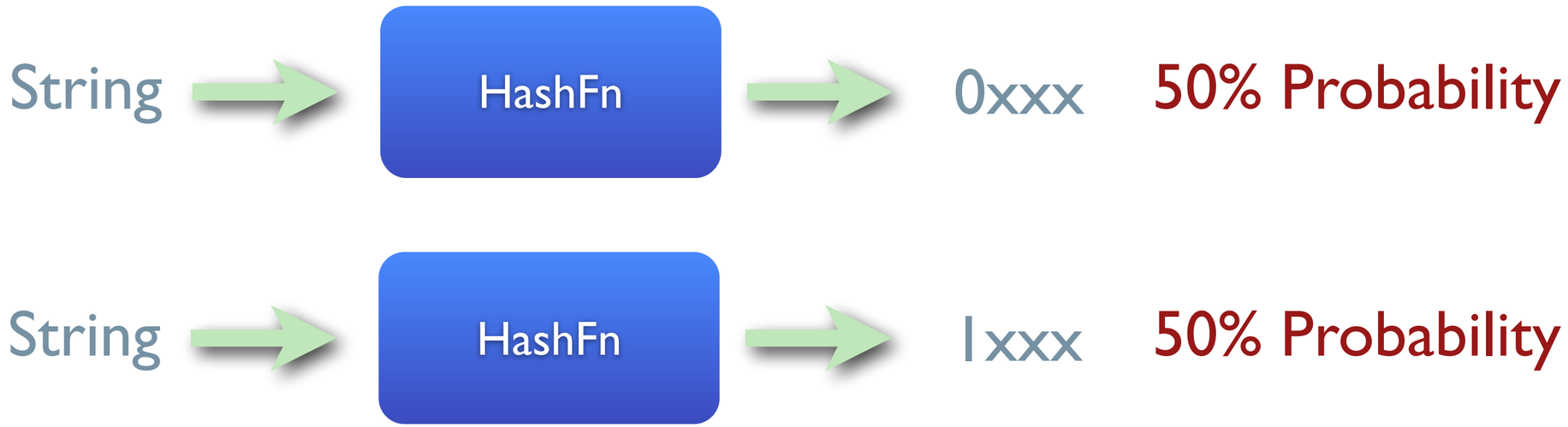
Fangjin Yang & Nelson Ray 2014

HASH FUNCTIONS

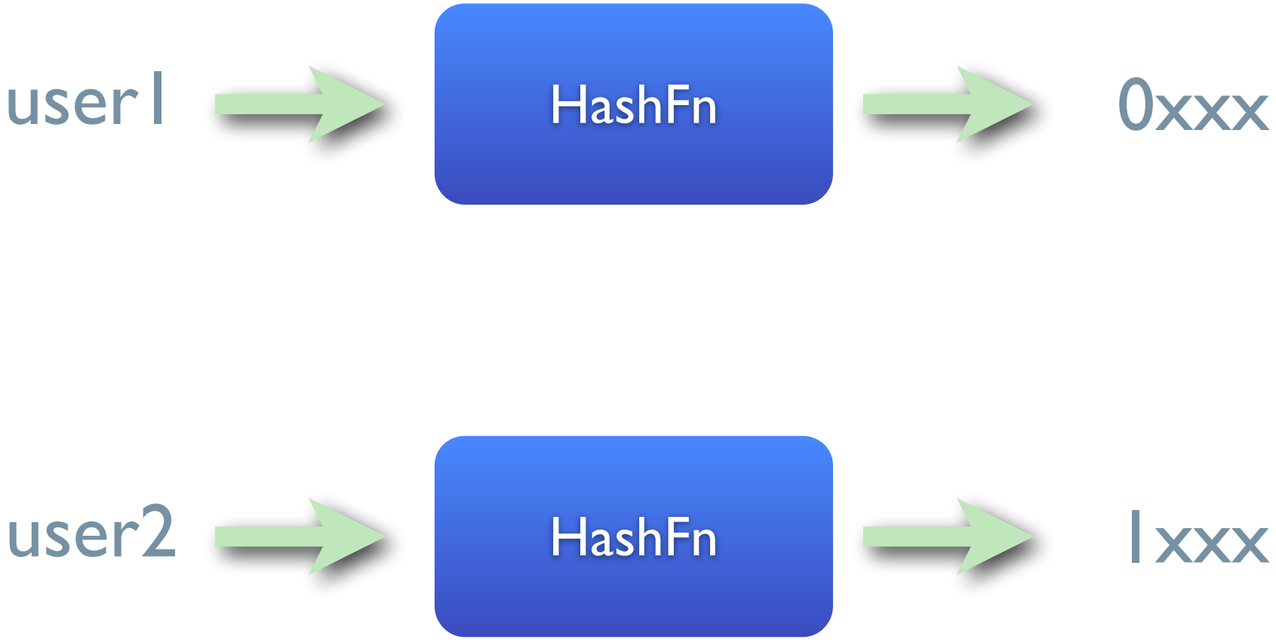
- Maps value in one space (generally larger) to another value in another space (generally smaller)

WHAT MAKES A GOOD HASH FUNCTION?

- Bits of output value are independent and have an equal probability of occurring (50%)



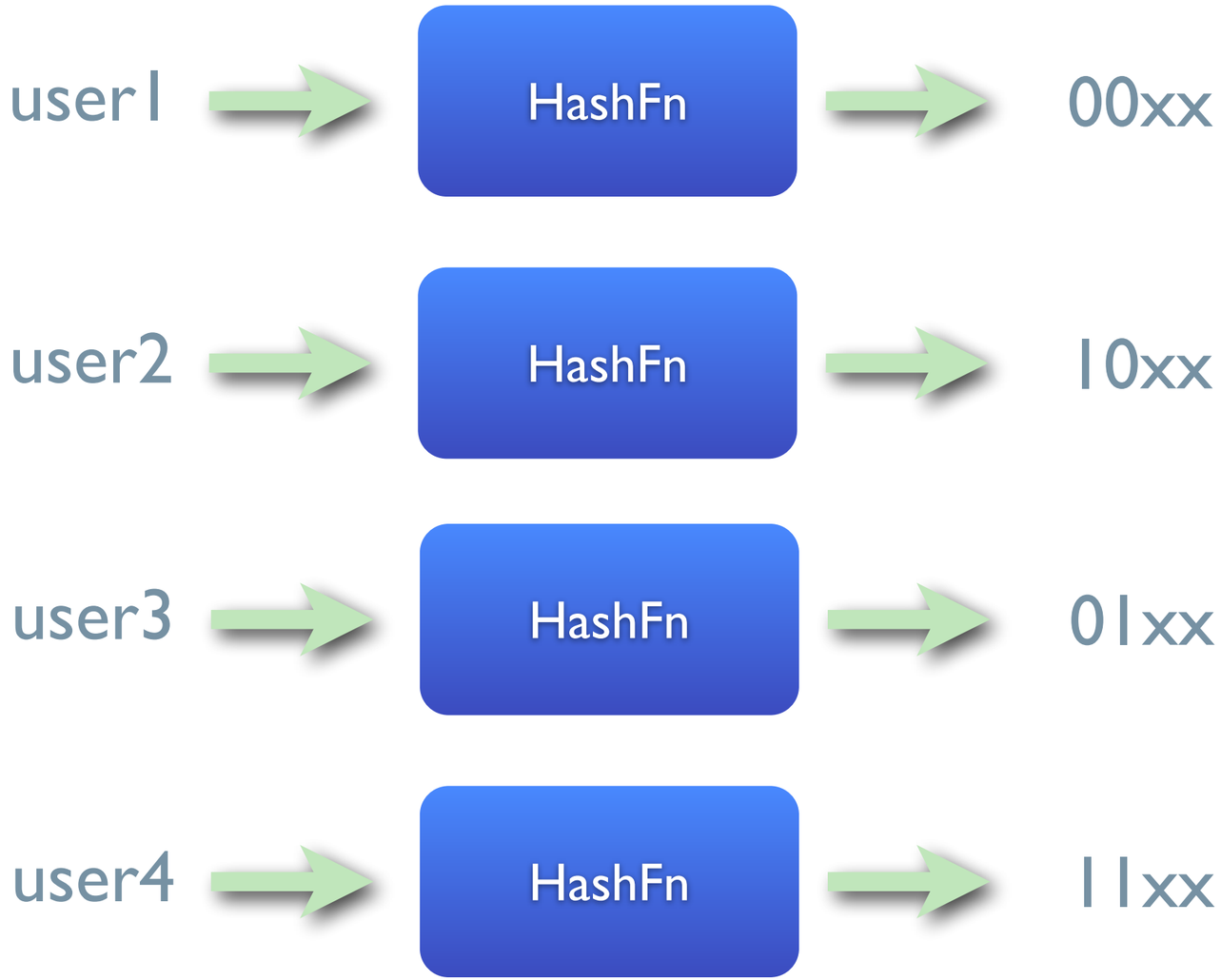
HASHING TWO STRINGS



THE NEXT BIT

METAMARKETS

HASHING 4 STRINGS



METAMARKETS

HYPERLOGLOG

- What about 001x?
  - If we hashed one string, 12.5% chance this could occur
  - If we hashed 8 strings, one of them should be this value

- What about 000001...x?
  - Extremely unlikely to occur if we only hashed one string

HYPERLOGLOG

- Looks at distribution of bits of hashed values
- Cares about the position of the left most ‘1’ bit
  - 1000 -> position == 1
  - 0100 -> position == 2
  - 0011 -> position == 3

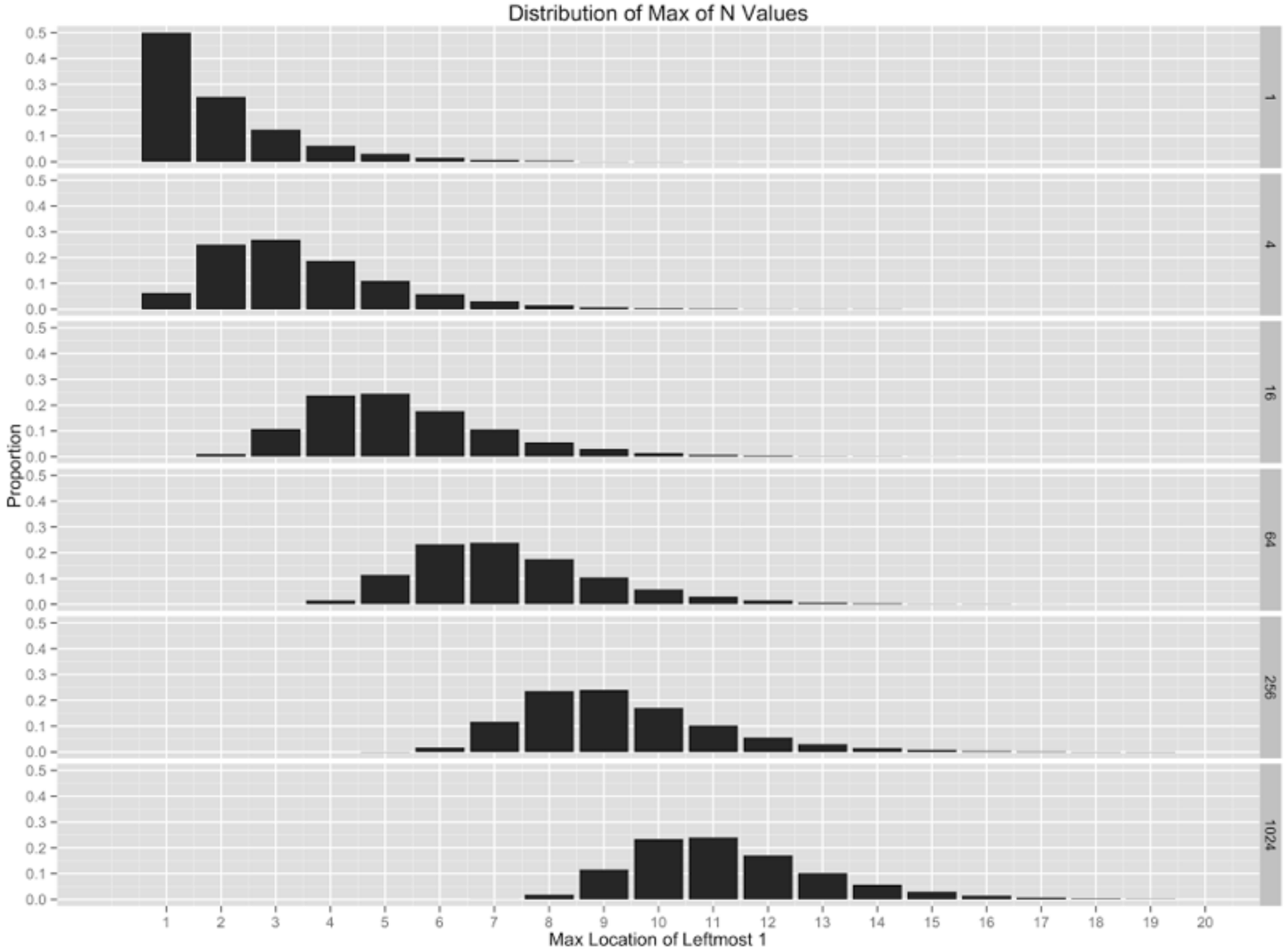
Fangjin Yang & Nelson Ray 2014

HYPERLOGLOG

- Stores the max position of the left-most ‘1’ bit of hashed values
  - User1 —> hash —> 1000 (position == 1)
  - User2 —> hash —> 0100 (position == 2)
  - User3 —> hash —> 0011 (position == 3)
- HLL will store position == 3

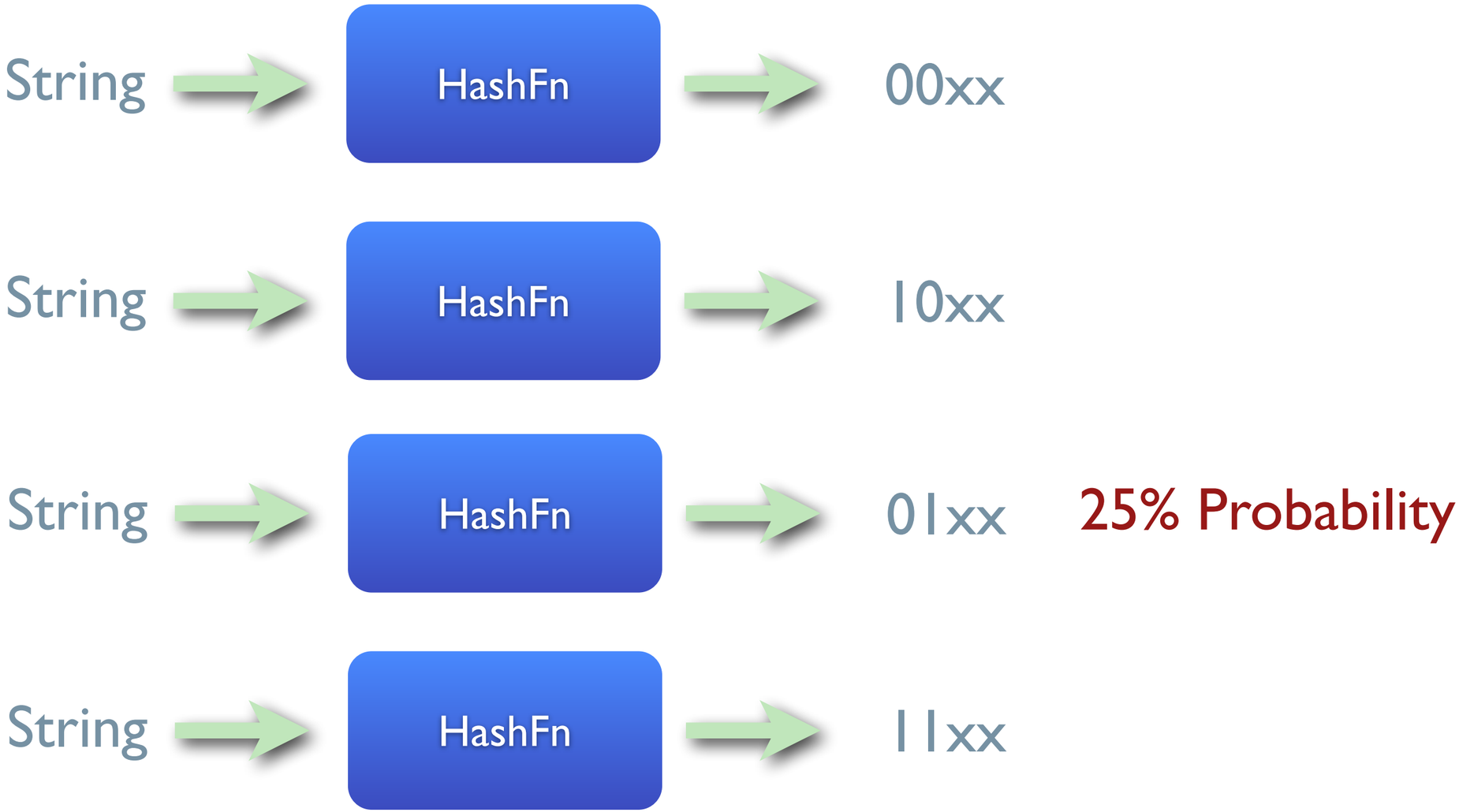
Fangjin Yang & Nelson Ray 2014

HYPERLOGLOG



Fangjin Yang & Nelson Ray 2014

HYPERLOGLOG ACCURACY



METAMARKETS

Fangjin Yang & Nelson Ray 2013

HYPERLOGLOG

- If we fed the stream through a second hash function, we'd have a second independent estimate
- Adding more hash functions gives us more independent estimates that we can combine together for a lower variance estimate
- This is expensive because we have to hash the same data n times

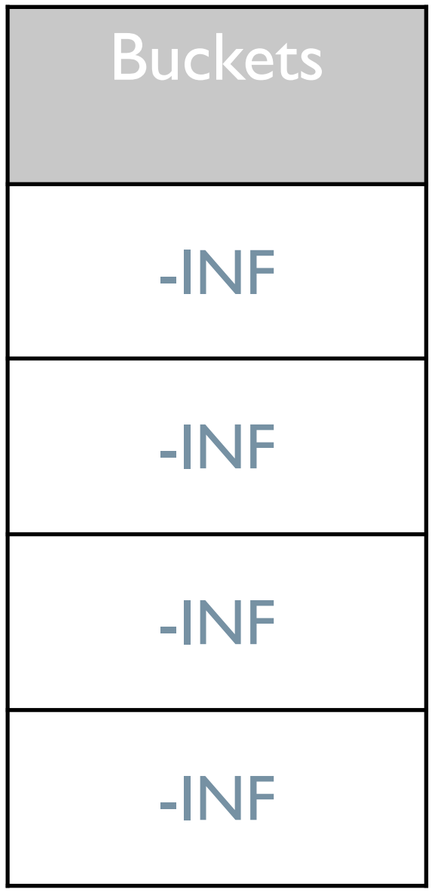
Fangjin Yang & Nelson Ray 2014

HYPERLOGLOG

- Instead we can split the stream
- Estimate the cardinality of each sub-stream
- For each sub-stream
  - Store the maximum over the positions of the leftmost '1' bit for hashed values of the sub-stream

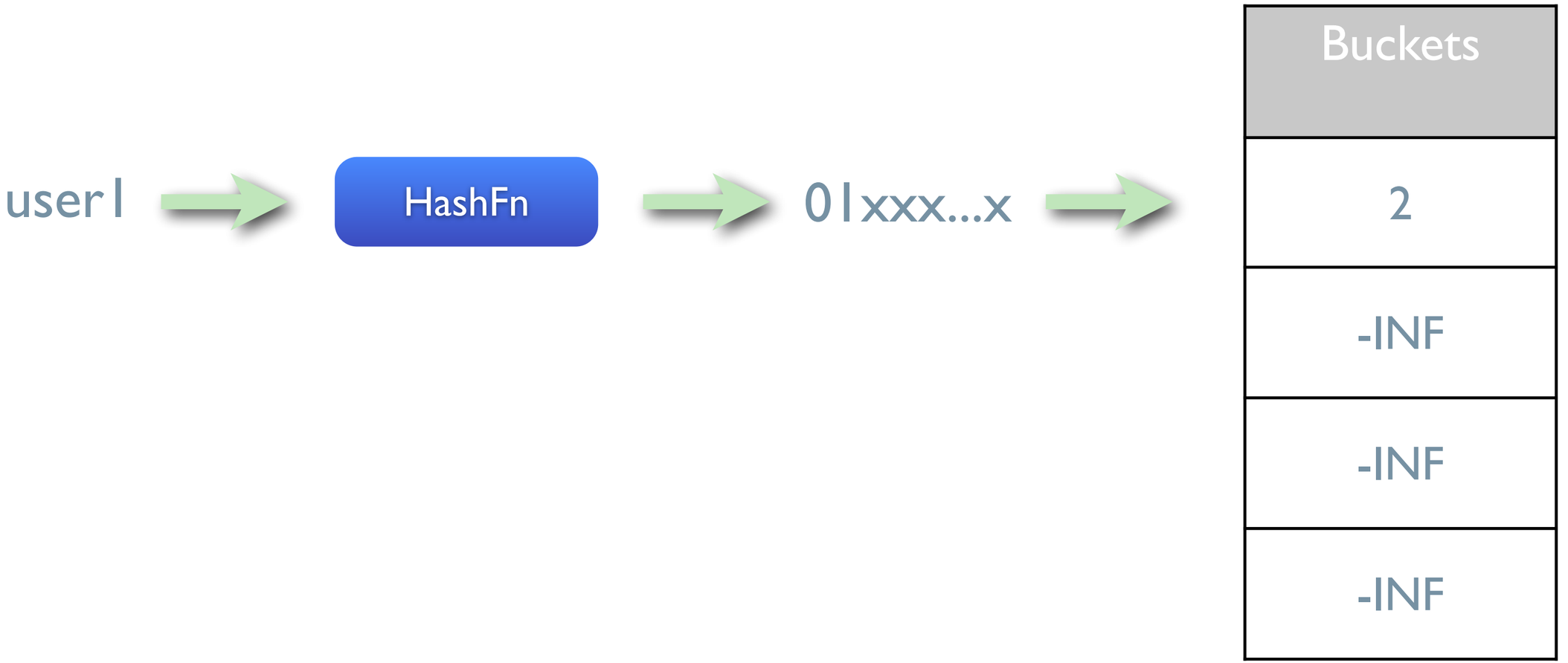
Fangjin Yang & Nelson Ray 2014

HYPERLOGLOG

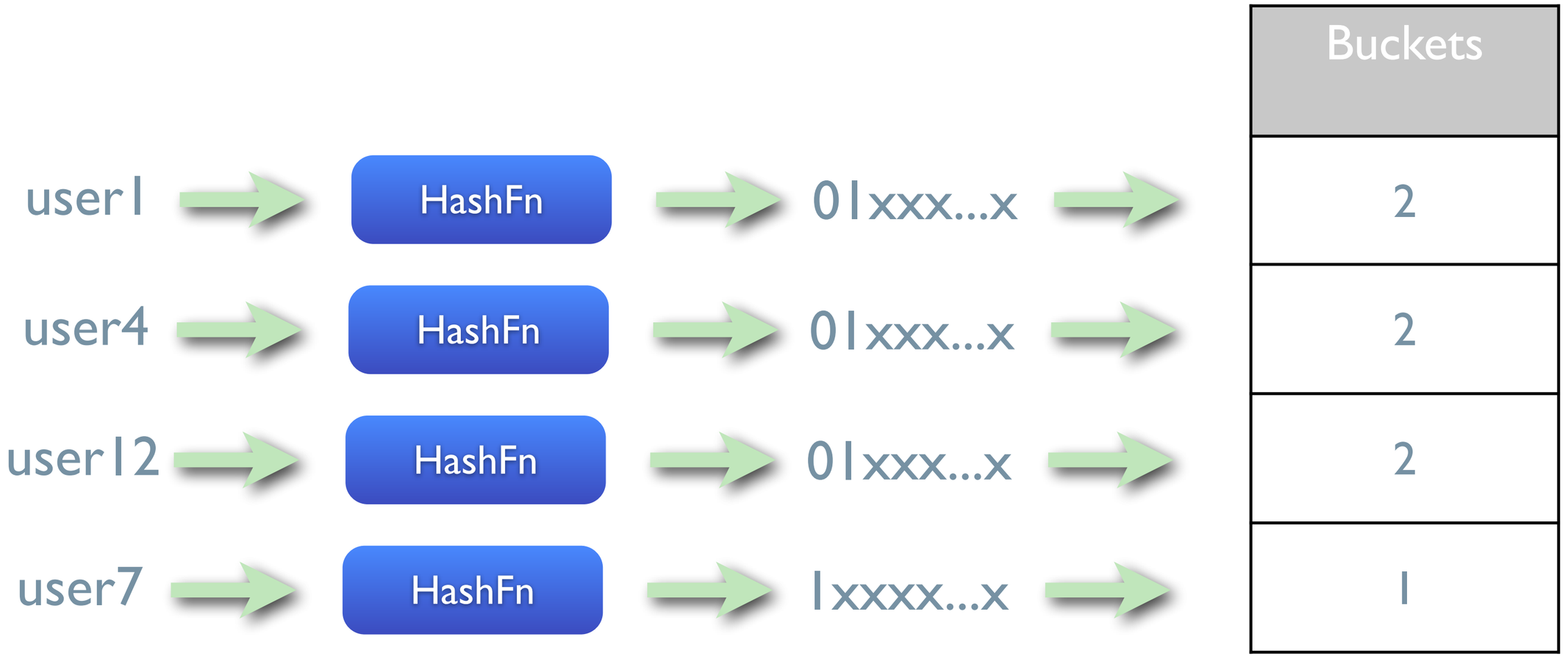


Fangjin Yang & Nelson Ray 2014

HYPERLOGLOG



HYPERLOGLOG



HYPERLOGLOG

Buckets
2 -> 3
2
2
1

DETERMINING FINAL CARDINALITY

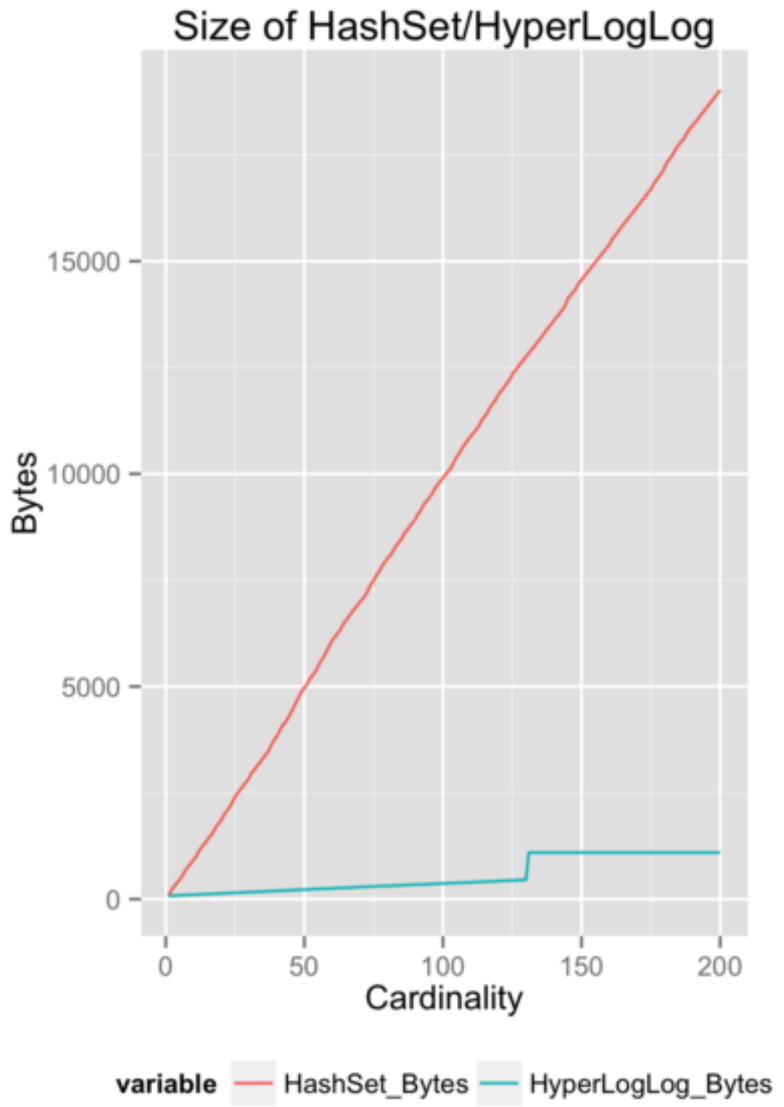
Buckets
3
2
2
1

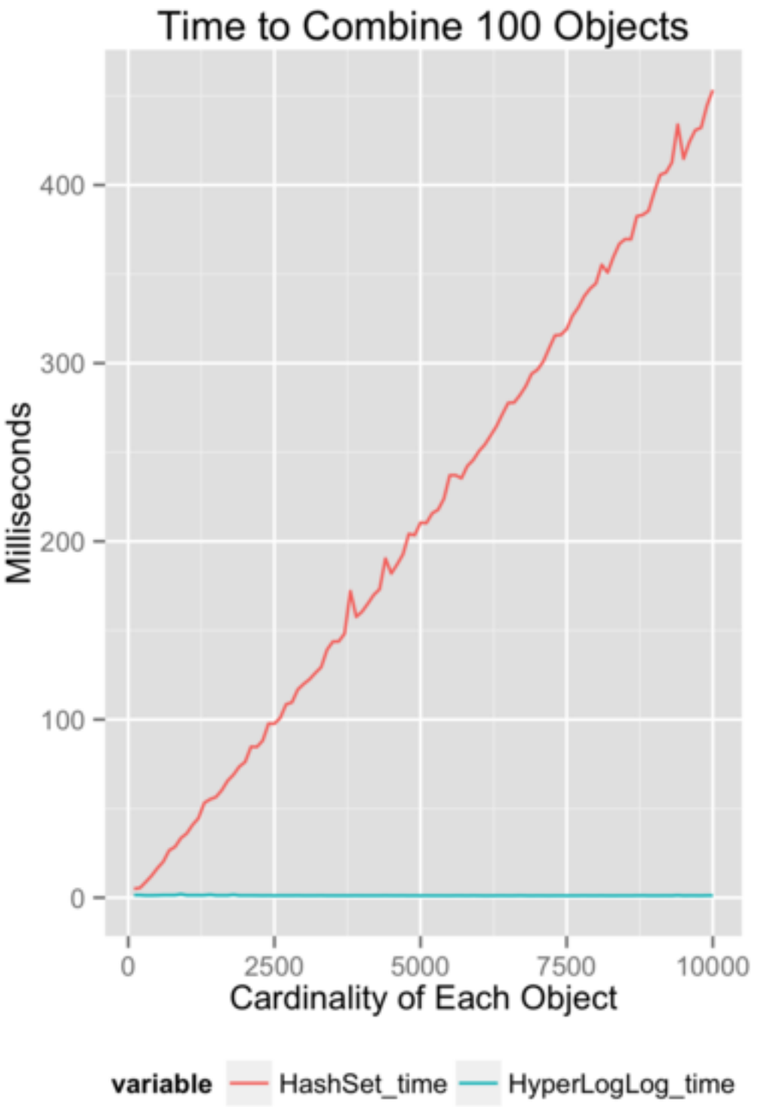
HYPERLOGLOG

Timestamp	Buckets
2013-10-28T02	[3, 2, 2, 1]
2013-10-28T03	[1, 2, 1, 2]
2013-10-28T04	[2, 1, 4, 1]
2013-10-28T05	[2, 2, 3, 1]

HYPERLOGLOG

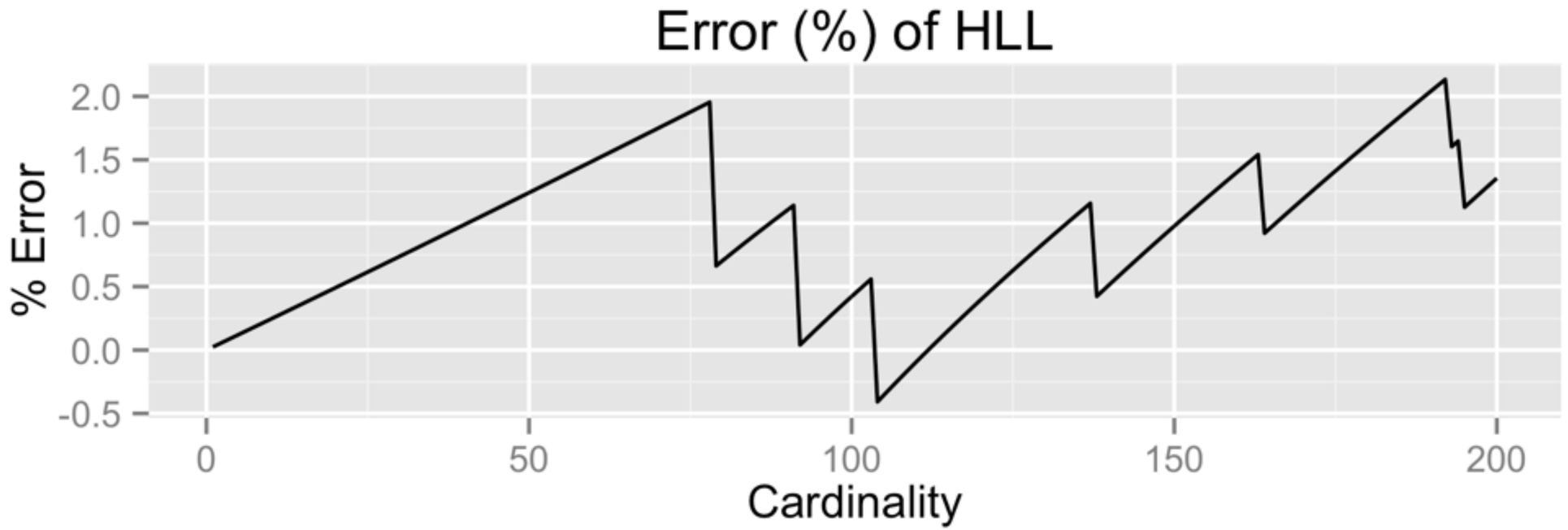
Timestamp	HLL Object
2013-10-28	[3, 2, 4, 2]





Fangjin Yang & Nelson Ray 2014

RESULTS

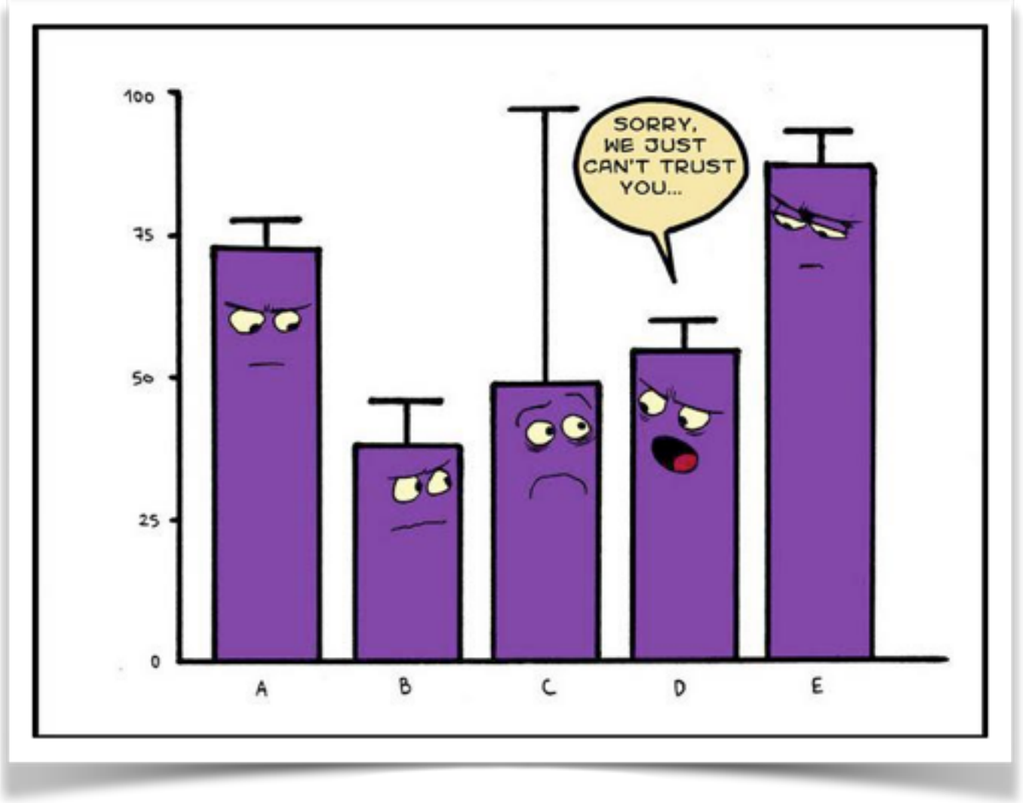


Fangjin Yang & Nelson Ray 2014

CASE STUDY 2

CASE STUDY 2

- Problem: determine distribution of values
- Use case: quantiles and histograms
- Hourly truncation



Fangjin Yang & Nelson Ray 2014

THE DATA

Timestamp	Bid Price
2013-10-28T02:13:43Z	1.19
2013-10-28T02:14:21Z	0.05
2013-10-28T02:55:32Z	1.04
2013-10-28T03:07:28Z	0.16
2013-10-28T03:13:43Z	1.03
2013-10-28T04:18:19Z	0.15
2013-10-28T05:36:34Z	0.01
2013-10-28T05:37:59Z	1.03

EXACT SOLUTION

Timestamp	Bid Price
2013-10-28T02:13:43Z	1.19
2013-10-28T02:14:21Z	0.05
2013-10-28T02:55:32Z	1.04
2013-10-28T03:07:28Z	0.16
2013-10-28T03:13:43Z	1.03
2013-10-28T04:18:19Z	0.15
2013-10-28T05:36:34Z	0.01
2013-10-28T05:37:59Z	1.03

Timestamp	Bid Prices
2013-10-28T02	[1.19, 0.05, 1.04]
2013-10-28T03	[0.16, 1.03]
2013-10-28T04	[0.15]
2013-10-28T05	[0.01, 1.03]

EXACT SOLUTION

Timestamp	Bid Prices
2013-10-28T02	[1.19, 0.05, 1.04]
2013-10-28T03	[0.16, 1.03]
2013-10-28T04	[0.15]
2013-10-28T05	[0.01, 1.03]

Timestamp	Bid Prices
2013-10-28	[1.19, 0.05, 1.04, 0.16, 1.03, 0.15, 0.01, 1.03]

EXACT SOLUTION

- Arrays of values
- Storage: Linear
- Computation: Linear
- Accuracy: 100%

- Problem: Storing raw values can often be more expensive than storing the rest of the row.
- Solution: Store an approximate representation!

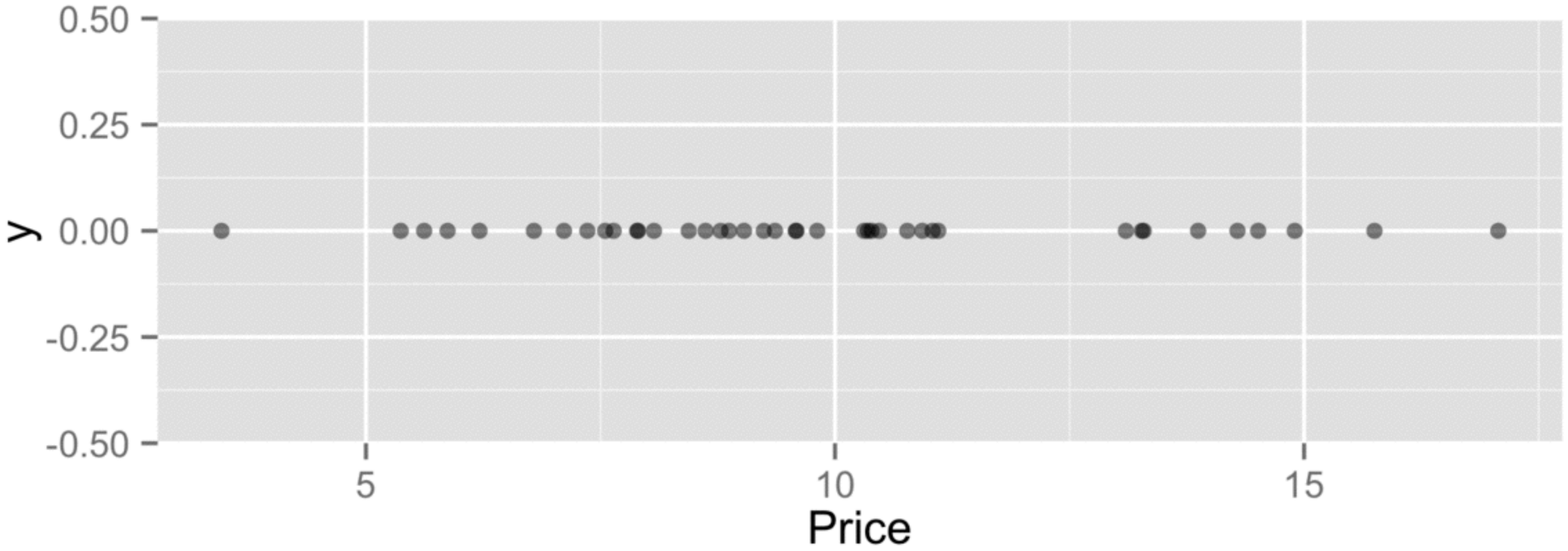
APPROXIMATE HISTOGRAMS

- “A Streaming Parallel Decision Tree Algorithm”
- Yael Ben-Haim & Elad Tom-Tov
- Storage: Sublinear/Linear
- Computation: Sublinear/Linear
- Accuracy: pretty good

RAW DATA

- 40 Prices: 3.46, 5.37, 5.62, 5.87, 6.21, 6.79, 7.11, 7.36, 7.55, 7.64, 7.89, 7.9, 8.07, 8.44, 8.62, 8.78, 8.87, 9.03, 9.24, 9.36, 9.58, 9.59, 9.81, 10.31, 10.35, 10.39, 10.47, 10.77, 10.93, 11.04, 11.1, 13.1, 13.27, 13.29, 13.87, 14.29, 14.51, 14.9, 15.75, 17.07

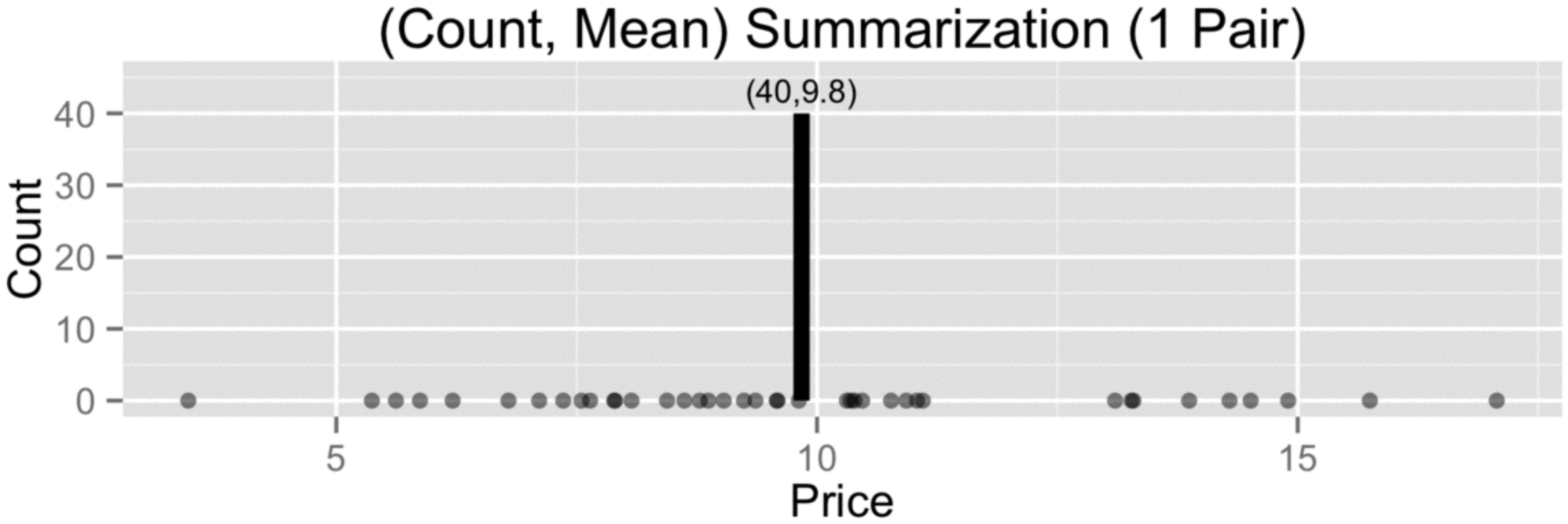
RAW DATA



METAMARKETS

Fangjin Yang & Nelson Ray 2013

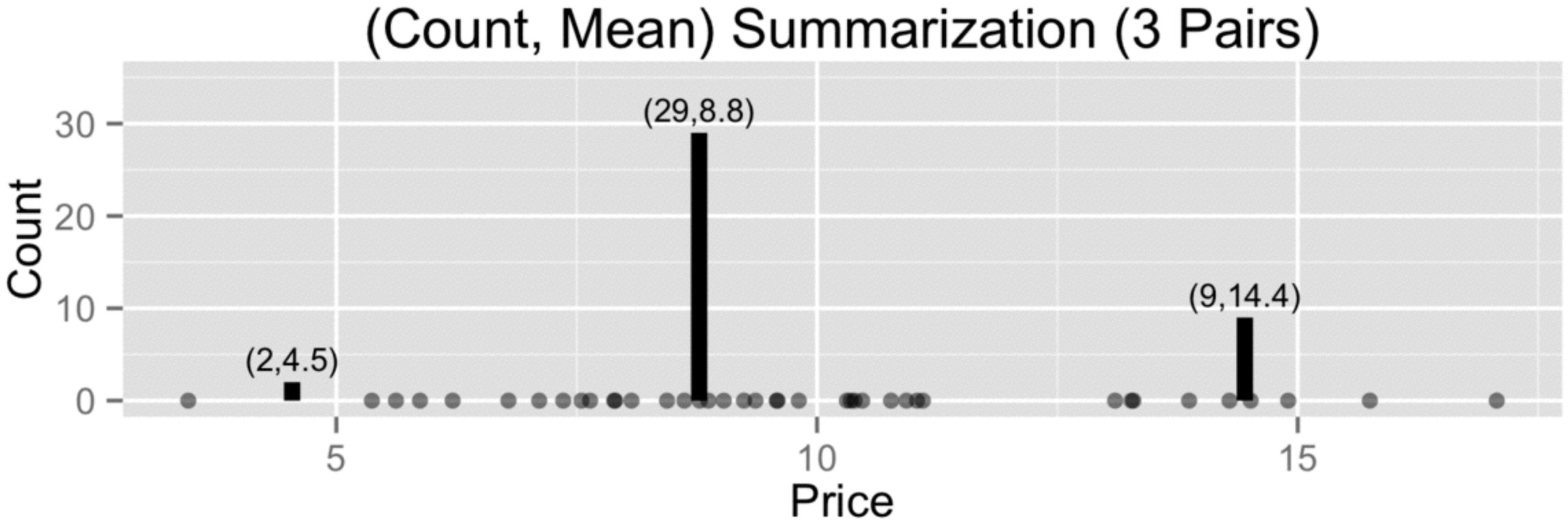
SUMMARIZE WITH (COUNT, MEAN)



METAMARKETS

Fangjin Yang & Nelson Ray 2013

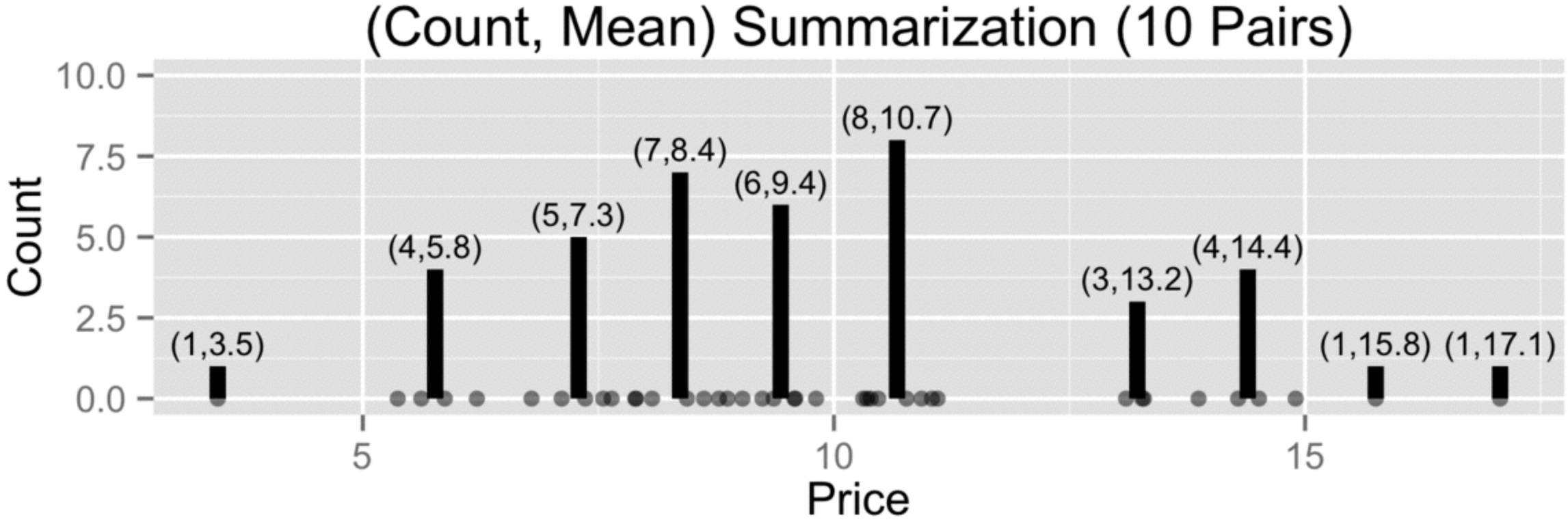
SUMMARIZE WITH (COUNT, MEAN)



METAMARKETS

Fangjin Yang & Nelson Ray 2013

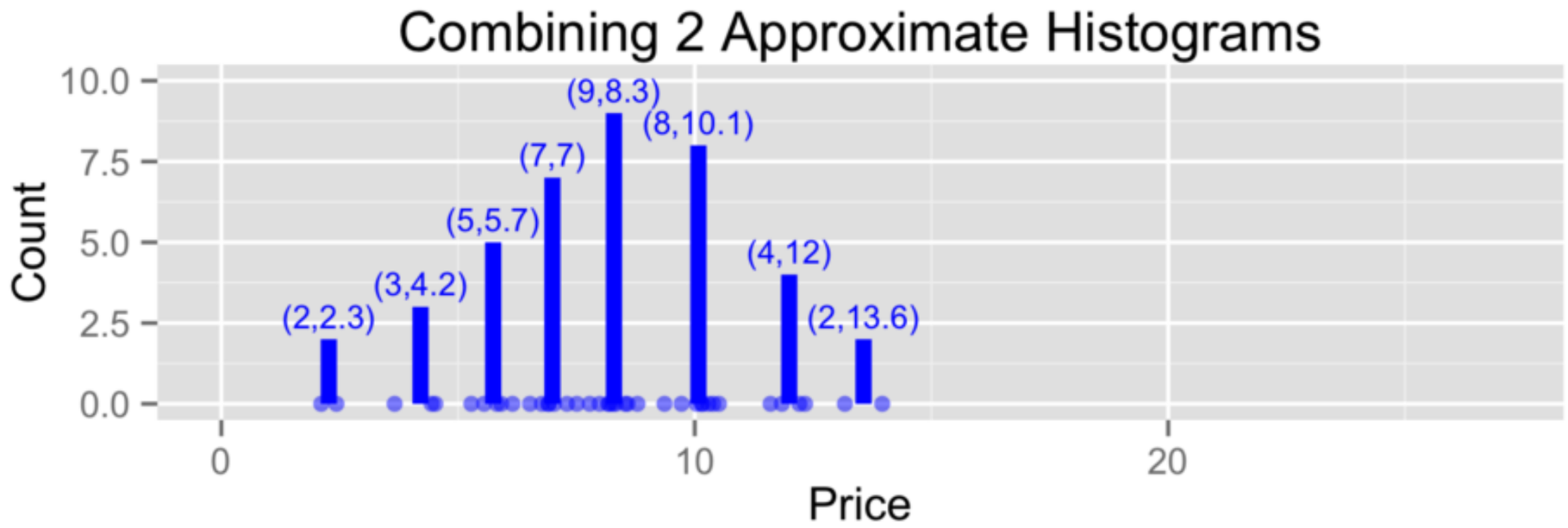
SUMMARIZE WITH (COUNT, MEAN)



METAMARKETS

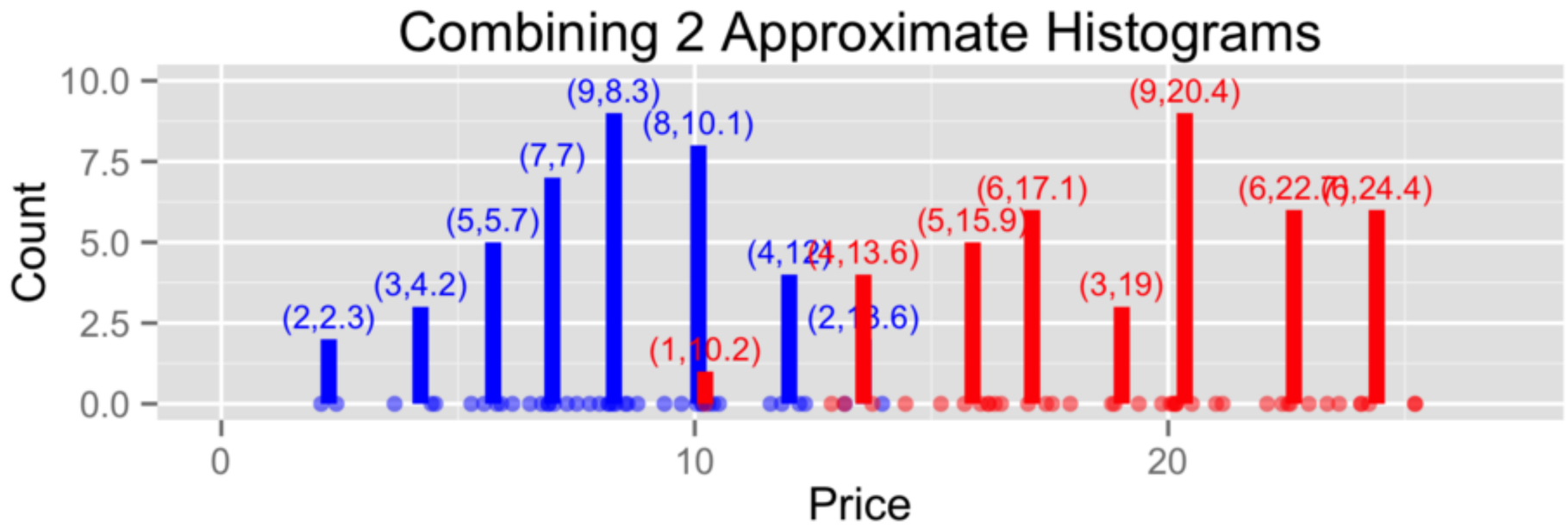
Fangjin Yang & Nelson Ray 2013

COMBINING HISTOGRAMS

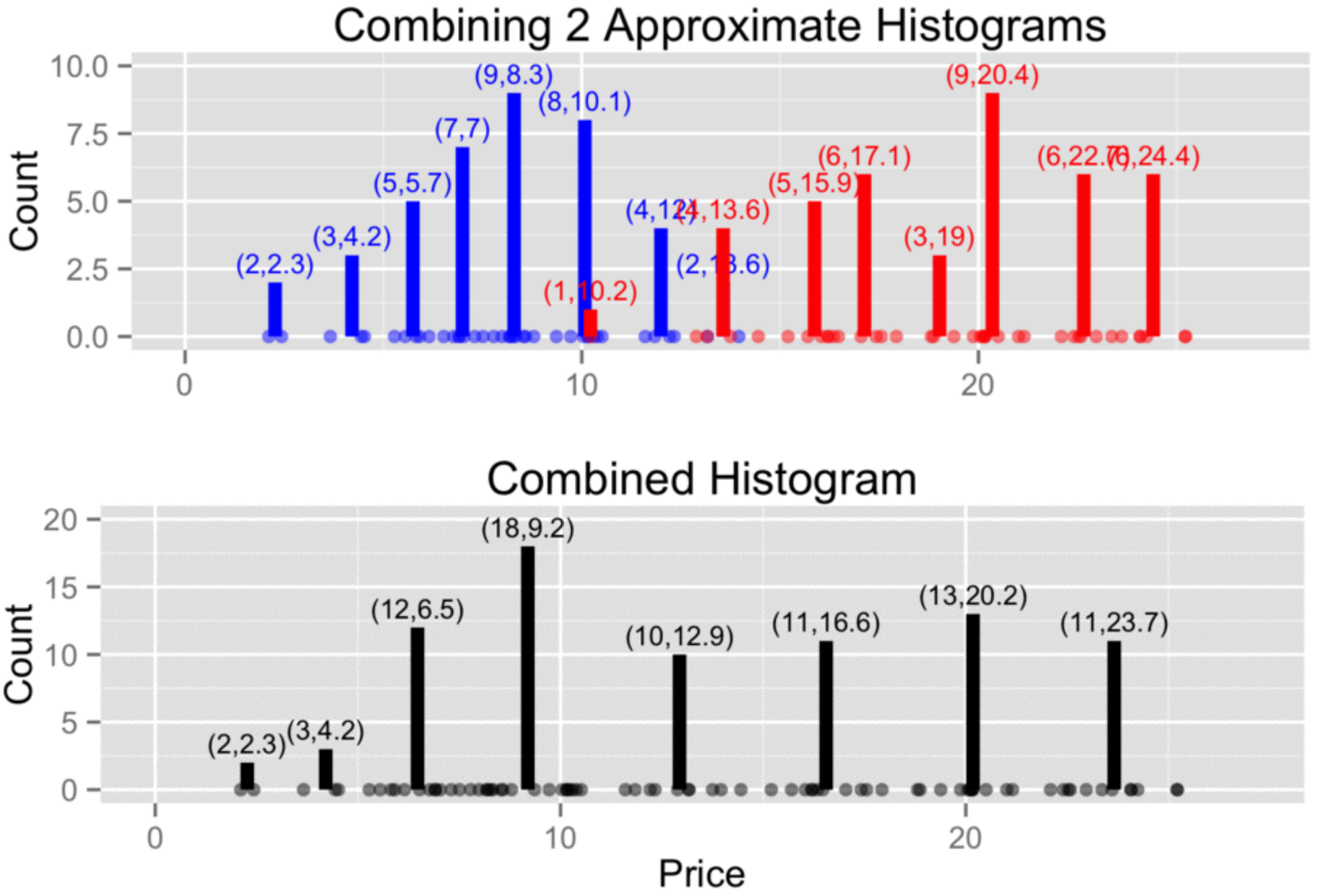


Fangjin Yang & Nelson Ray 2014

COMBINING HISTOGRAMS

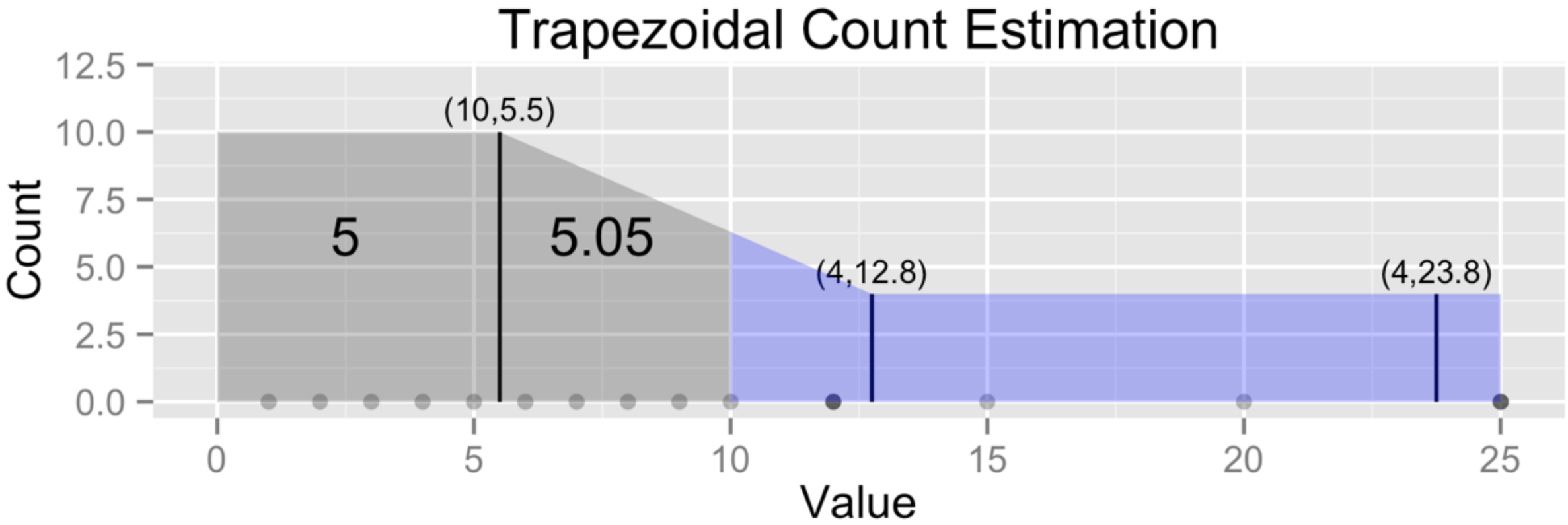


Fangjin Yang & Nelson Ray 2014

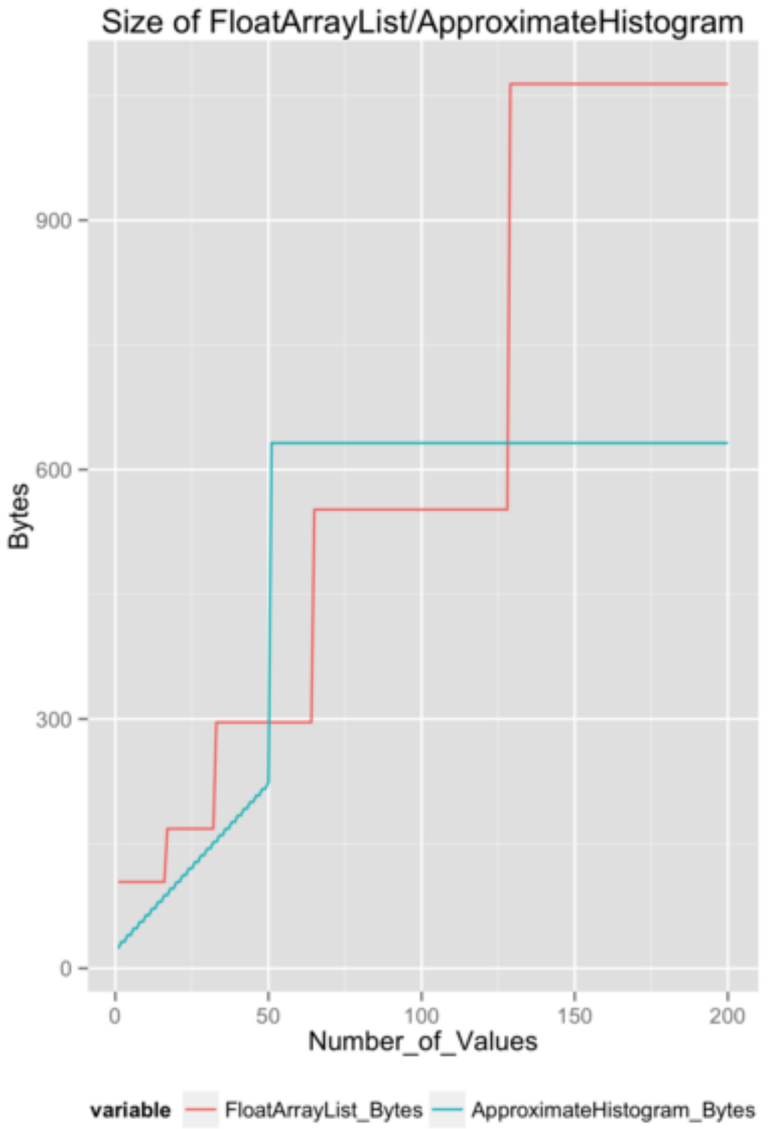


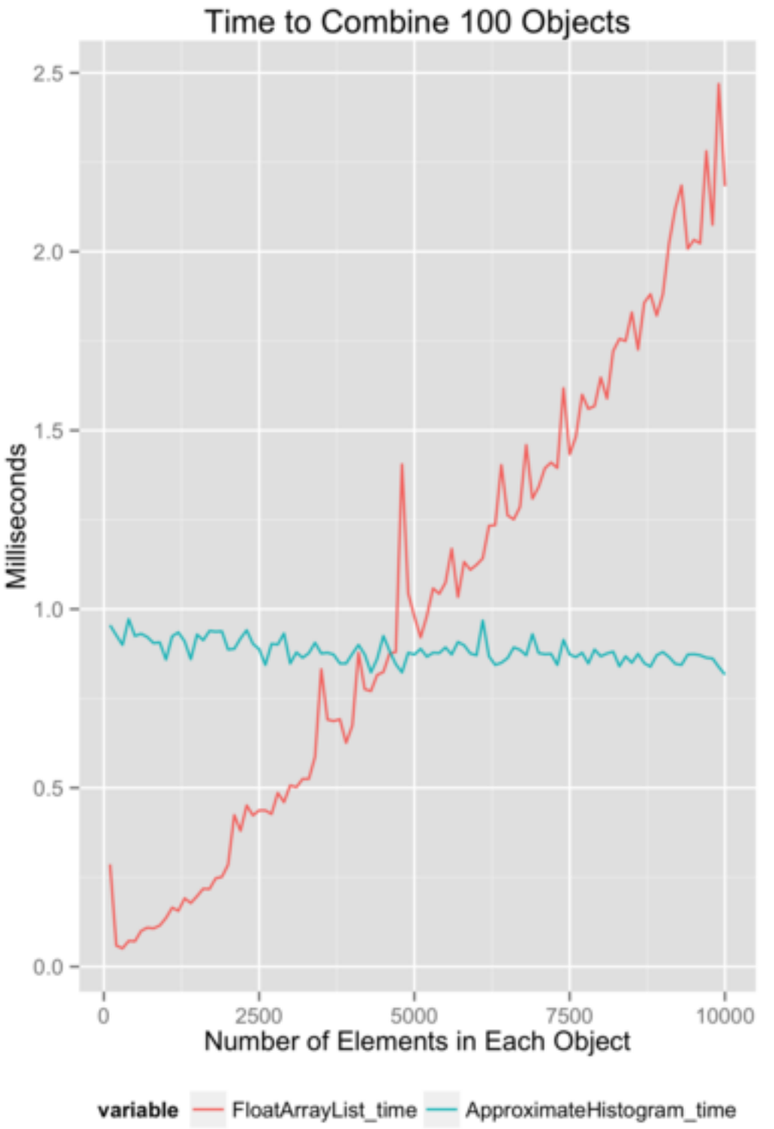
Fangjin Yang & Nelson Ray 2014

COUNT # <= X



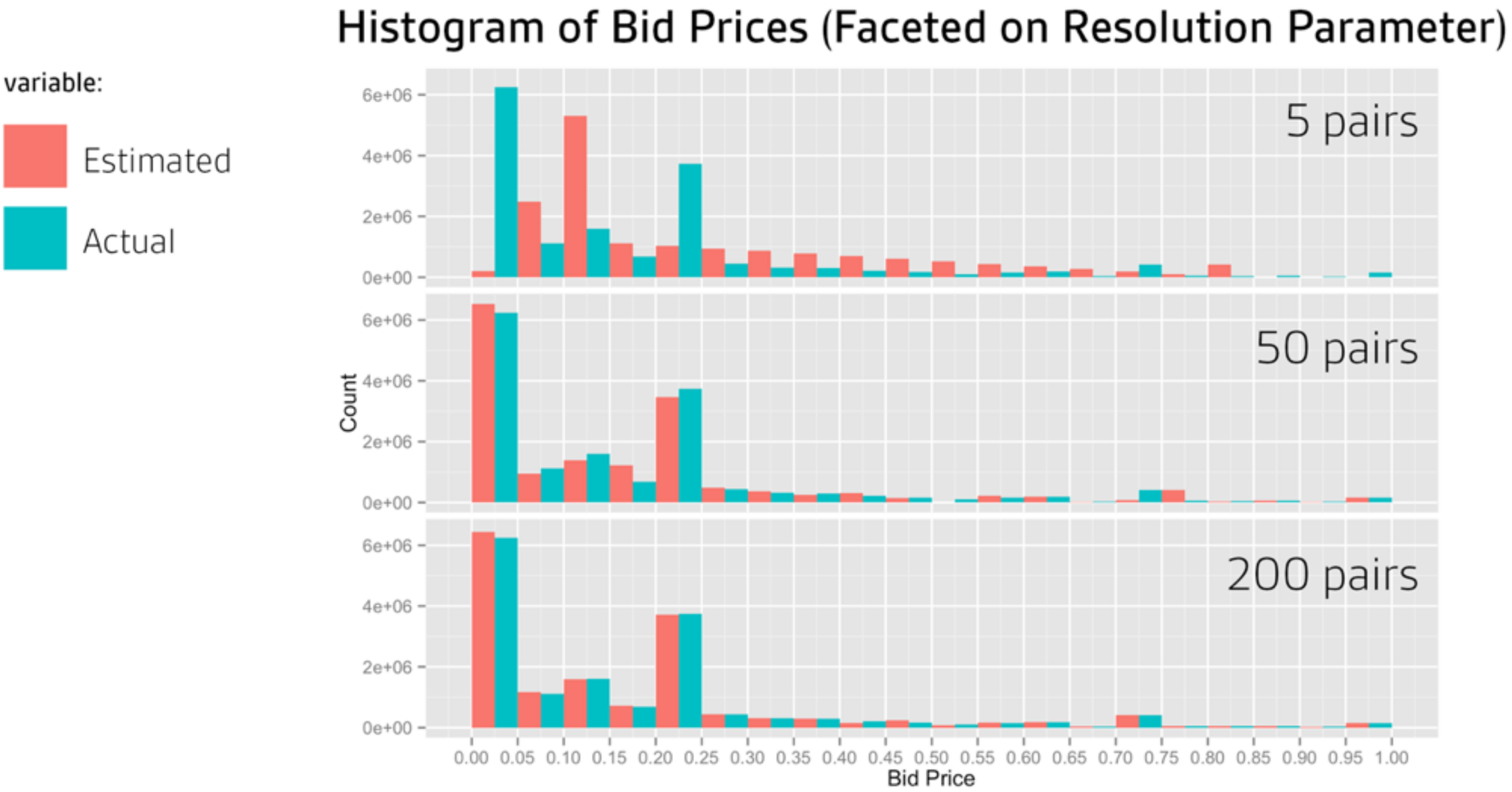
Fangjin Yang & Nelson Ray 2014





Fangjin Yang & Nelson Ray 2014

ACCURACY



Fangjin Yang & Nelson Ray 2014

DRUID

- Open source
- Designed to power interactive applications at scale
- Optimized for business intelligence (OLAP) queries
- Arbitrary slice-n-dice and drill into data
- Supports streaming and batch data ingestion
- Exact and approximate calculations (Hyperloglog, approximate histograms)

BENCHMARKS

- 100 cc2.8xlarge (1600 cores, 6TB RAM) Druid cluster
- 27B summarized rows/s scan rate
- Add 16B summarized (~640B raw) rows/s
- Combine 4B HyperLogLog objects/s
- Combine 1.5B ApproximateHistogram objects/s

CONCLUSIONS

- Summarization for sums: substantially (e.g. ~40x for us) faster/less storage
- 100% accuracy
- Sketches for cardinality/distribution: 1-2 orders of magnitude faster/less storage than raw
- 97% accuracy
- 40x lower costs is make or break
- interactive queries that are accurate enough

DRUID IS OPEN SOURCE

WWW.DRUID.IO

twitter **@druidio**

irc.freenode.net **#druid-dev**

THANK YOU