Processing windowed Top-k Queries on Uncertain Streams

Cheqing Jin 金澈清 cqjin@ecust.edu.cn

East China Normal University, Shanghai, China

華東師范大學 上海 中國

Outline

- Introduction
  - Top-k queries
  - contribution
- Our solution
- Experiments
- Conclusion

Possible World Model

A small record set of reading logs

ID	Speed(*10)	Prob.
1	5	0.8
2	6	0.5
3	8	0.4
4	2	0.4

If we ignore the probability field, the semantic of top-k query is very clear.

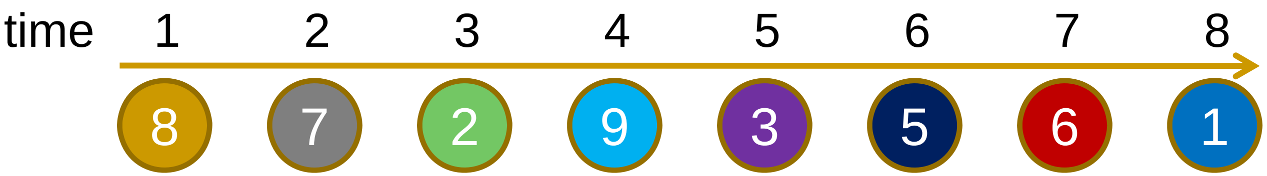
16 possible world instances

Tuples	Pr.	Tuples	Pr.	Tuples	Pr.
8, 6, 5, 2	.064	8, 6, 5	.096	8, 5, 2	.064
8, 5	.096	8, 6, 2	.016	8, 6	.024
8, 2	.016	8	.024	6, 5, 2	.096
6, 5	.144	6, 2	.024	6	.036
5, 2	.096	5	.144	2	.024
Empty	.036				

0.096 = 0.4 * 0.8 * (1-0.5) * (1-0.4)

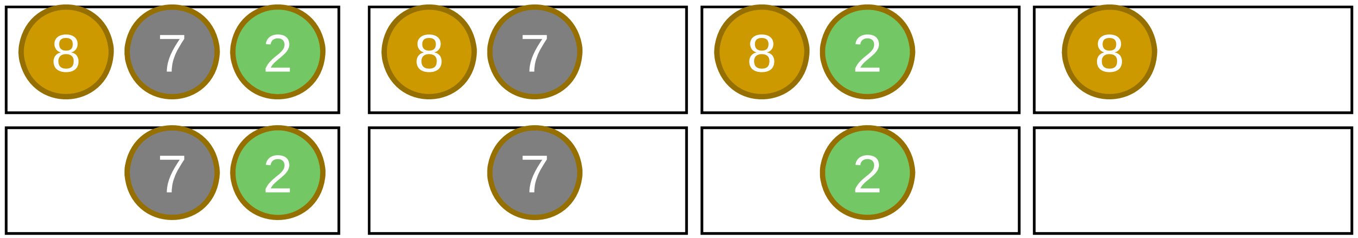
0.036=(1-0.8)*(1-0.5)*(1-0.4)*(1-0.4)

Windowed Model on Uncertain Streams

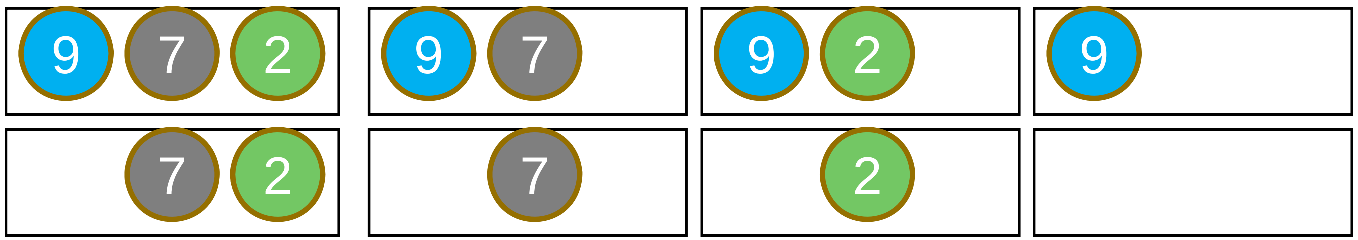


Assume W=3

At time 3, possible worlds are:



At time 4, possible worlds are:



Obviously, when a new tuple arrives, half of possible worlds will change.

Uncertain Top-*k* queries

- U-Top*k* [4]
  - returns the top-*k* tuples in all possible worlds with maximum probability.
- U-*k*Ranks [4]
  - returns the winner for the *i*-th rank for all $1 \leq i \leq k$.
- PT-*k* [2]
  - returns all the tuples with maximum aggregate probability greater than a user-given threshold *p*
    - aggregate probability: the prob. of being the top-*k* among all.
- P*k*-Top*k* [this paper]
  - returns the *k* most probable tuples of being the top-*k* among all.
  - A slight modification of PT-*k*, while without threshold *p*.

Example (k=2)

16 possible world instances

Tuples	Pr.	Tuples	Pr.	Tuples	Pr.
8, 6, 5, 2	.064	8, 6, 5	.096	8, 5, 2	.064
8, 5	.096	8, 6, 2	.016	8, 6	.024
8, 2	.016	8	.024	6, 5, 2	.096
6, 5	.144	6, 2	.024	6	.036
5, 2	.096	5	.144	2	.024
Empty	.036				

Query results

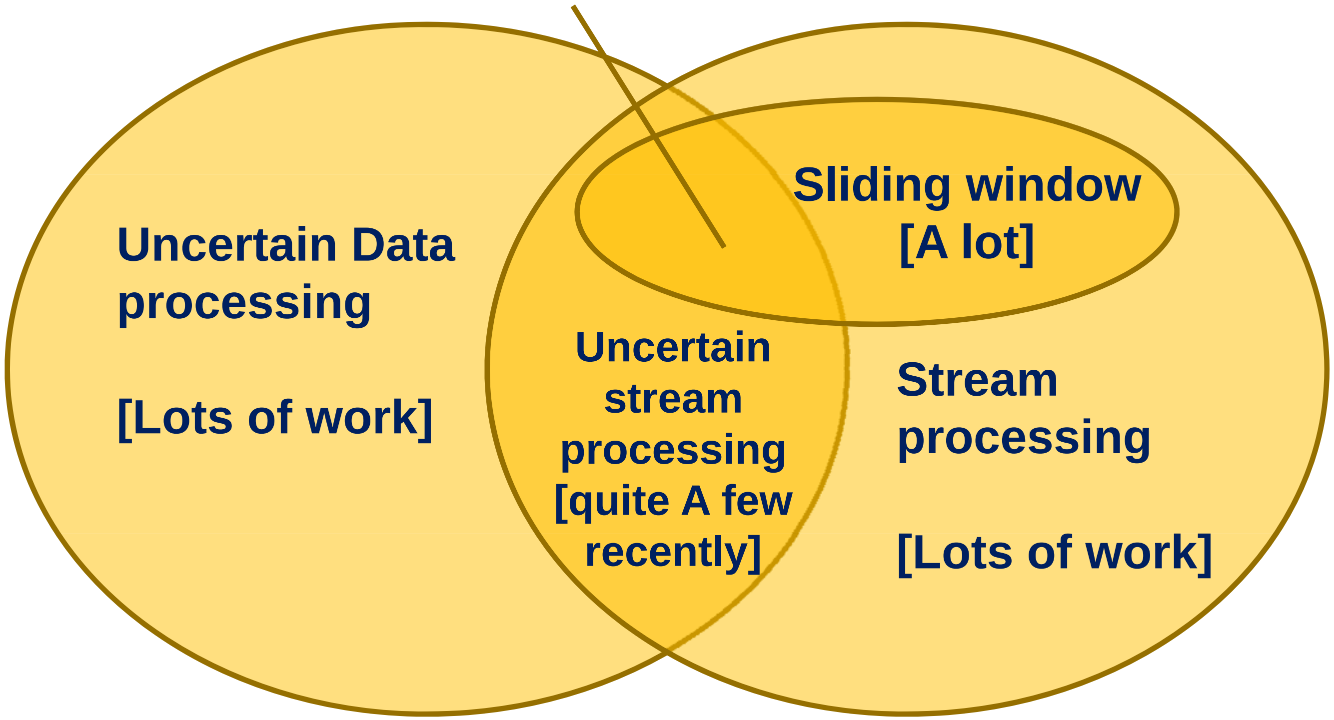
Query	U-Top*k*	U-*k*Ranks	PT-*k*	P*k*-Top*k* (*p=0.3*)
Result	{6, 5}	{8, 5}	{5, 6, 8}	{5, 6}

Claim: our goal is not to propose yet another new top-k query definition, but a new framework that all top-k queries can be processed in the sliding window model.

Contributions

However, Sliding-window solutions on uncertain streams
[Still None]

Contribution:
Ours is the first work In this area, especially for Top-k queries!



Working model

- In the ultimate situation, all tuples in the window must be saved in memory!
  - Example
    - t_i: the i-th tuple in stream, (value:$1/i$, probability:$1/i$).
    - The window size is W, at time n, for any k, tuple t_{n-W+1} is at the query result of Pk-Top$k$ query.
  - So, worst-case bounds are trivial and meaningless.

- So, we consider a more general scenario: ***random order stream model***.
  - The value and probability of a tuple are both randomly and independently drawn from some (arbitrary) distribution.

Naïve solution

- Basic Synopsis (BS)
  - Reserve all recent W tuples in memory
  - Use traditional method to answer top-k queries
- Analysis
  - Time-efficient, but space-inefficient.
  - The space complexity is $O(W)$.

Framework

GOAL: designing a general framework for all kinds of top-k queries, not only for a special kind of query.

Design Compact Set → **List all Useful Compact Sets** → **Compress Compact Sets**

Design Compact Set

1. A small subset of the original dataset
2. Self-maintenance $C(D \cup \{t\}) \subseteq C(D) \cup \{t\}$
3. Capable of answering a top-k query

List all Useful Compact Sets

1. *W* different windows. i.e., $[t-j, t]$, for $j=0..W-1$
2. One compact set for each window

Compress Compact Sets

1. CSQ, CCSQ, SCSQ, SCSQ-buffer
2. Space-efficient
3. Time-efficient

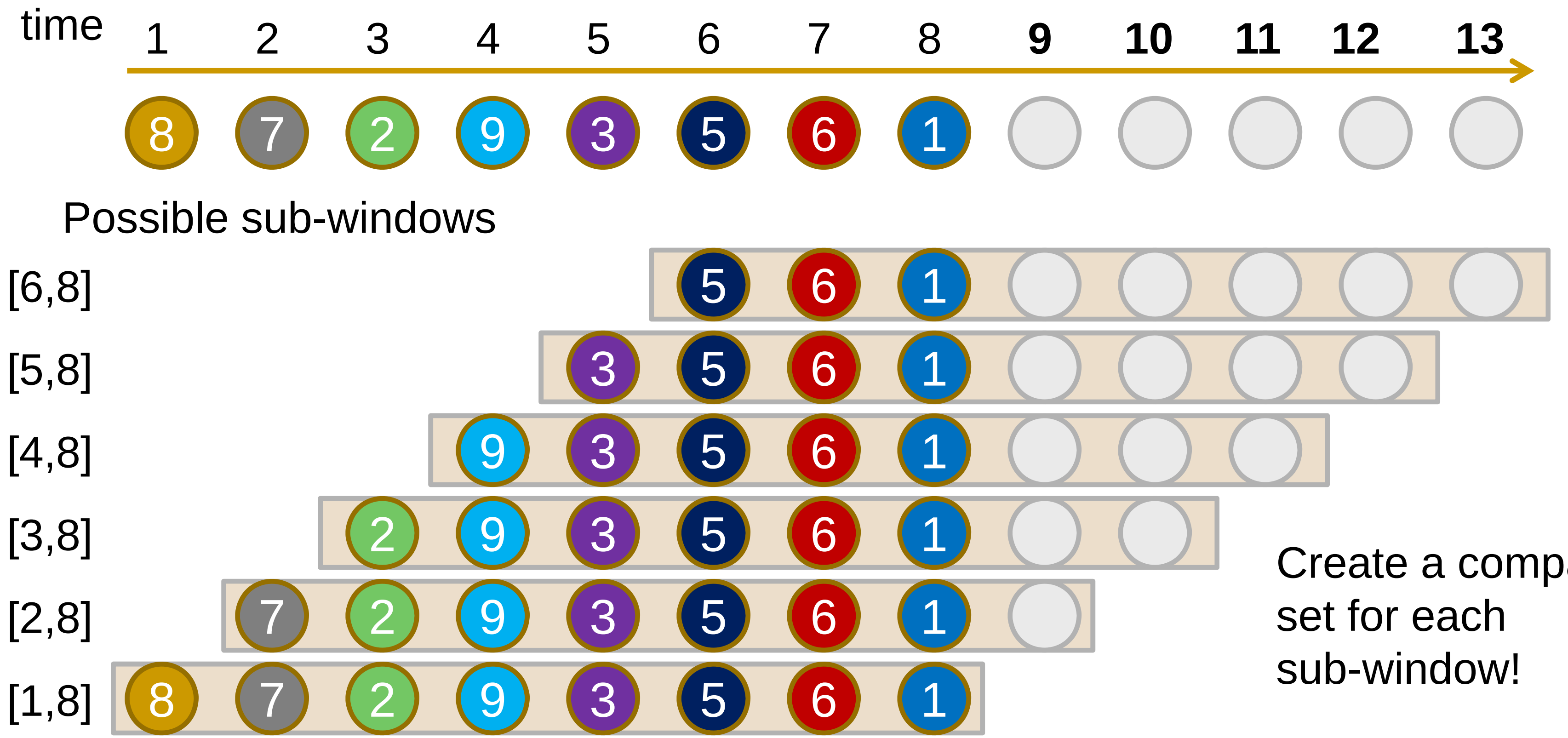
Example: Compact Set for P*k*-Top*k*

- Symbols
  1. D_i: the subset of D containing the first i tuples in D.
  2. $r_{i,j}$: the probability that a randomly generated world from D_i has exactly j tuples.
     1. $r_{i,j}$ can be maintained through dynamic program.
  3. $p(t_i)$: the probability of tuple t_i.
  4. $p(t_i)r_{i-1,j-1}$: the probability that t_i ranks the j-th in a randomly generated world from D.
  5. $p(t_i)\Sigma_{i=1..k}r_{i-1,j-1}$: the probability that t_i ranks top-k in all possible worlds generated from D.
  6. $\Sigma_{i=1..k}r_{i-1,d-1}$: The up-bound probability of any other tuple outside of D_d that ranks top-k in all possible worlds generated from D.

- Compact set for P*k*-Top*k*
  - *Smallest* D_d with k tuples (t_a) satisfying: $p(t_a)\Sigma_{i=1..k}r_{i-1,a-1} > \Sigma_{i=1..k}r_{i-1,d-1}$

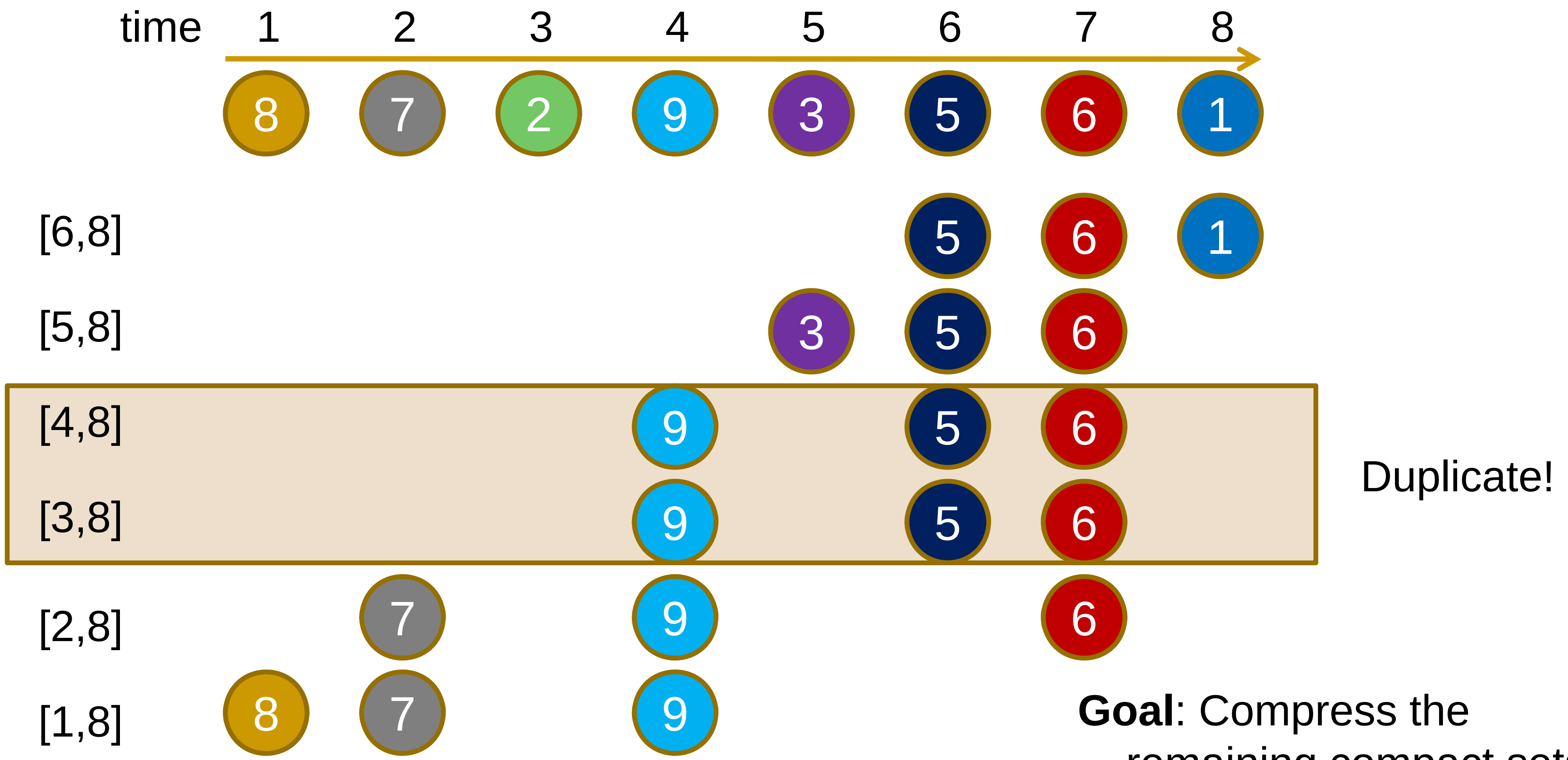
Possible sub-windows

- Assume window size W=8, k=3



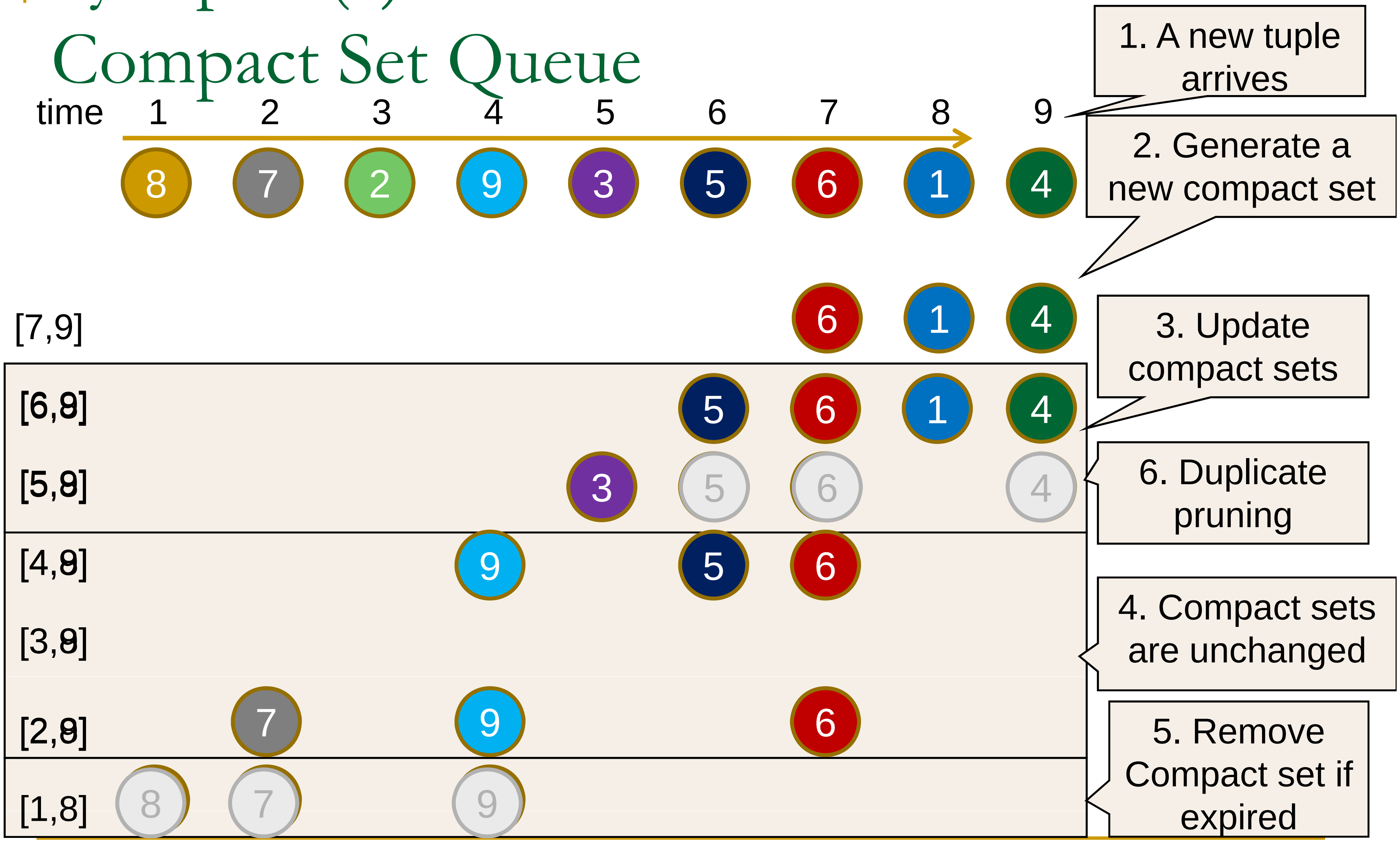
Create a compact set for each sub-window!

Create compact sets



Goal: Compress the remaining compact sets.

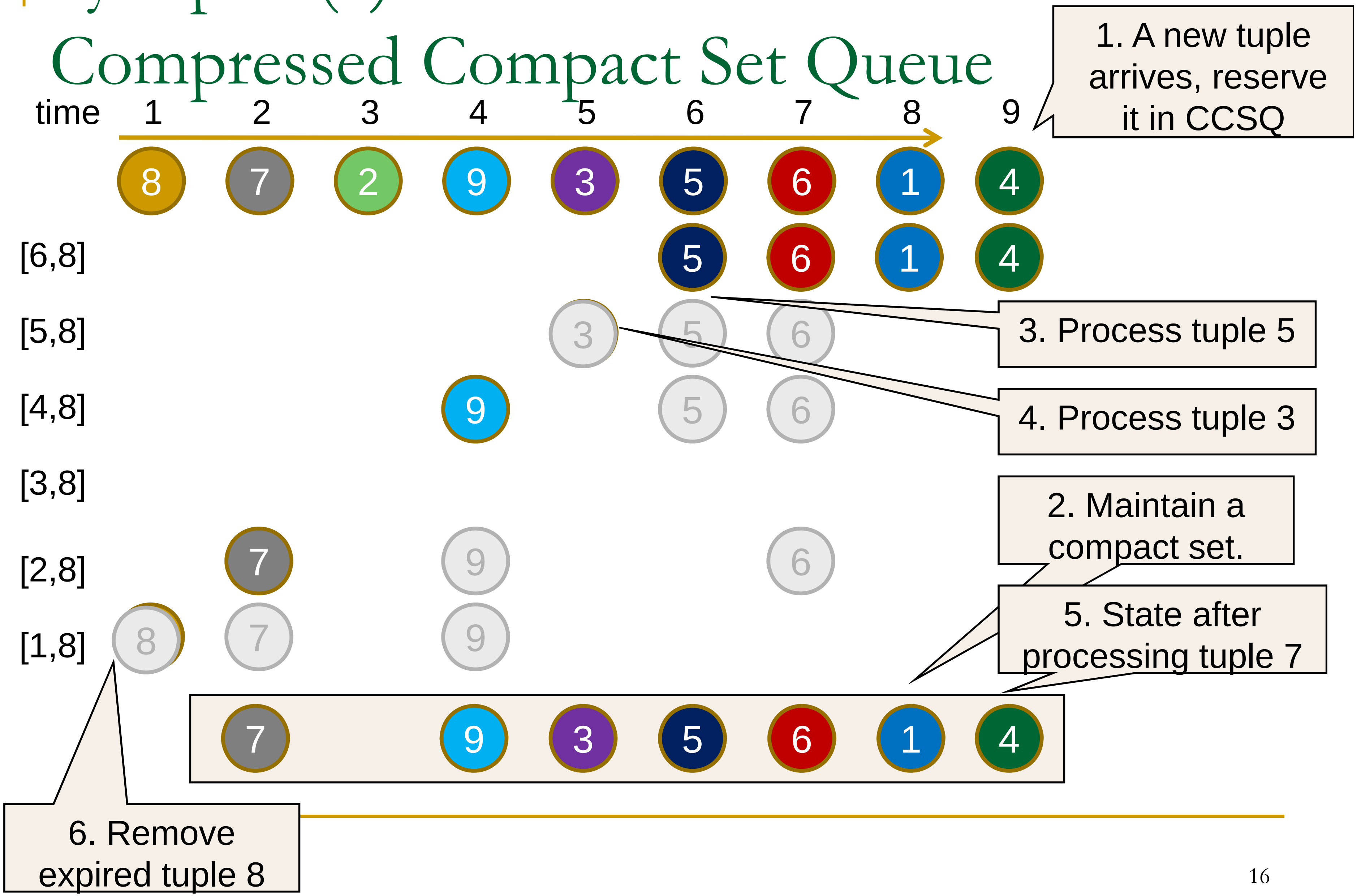
Synopsis (1): Compact Set Queue



Compact Set Queue: Analysis

- Advantages
  - Easy to understand
  - The space consumption and the per-tuple processing cost are small.
- Disadvantages
  - Redundancy exists between neighbor compact sets.
- Solution
  - compress neighbor compact sets!

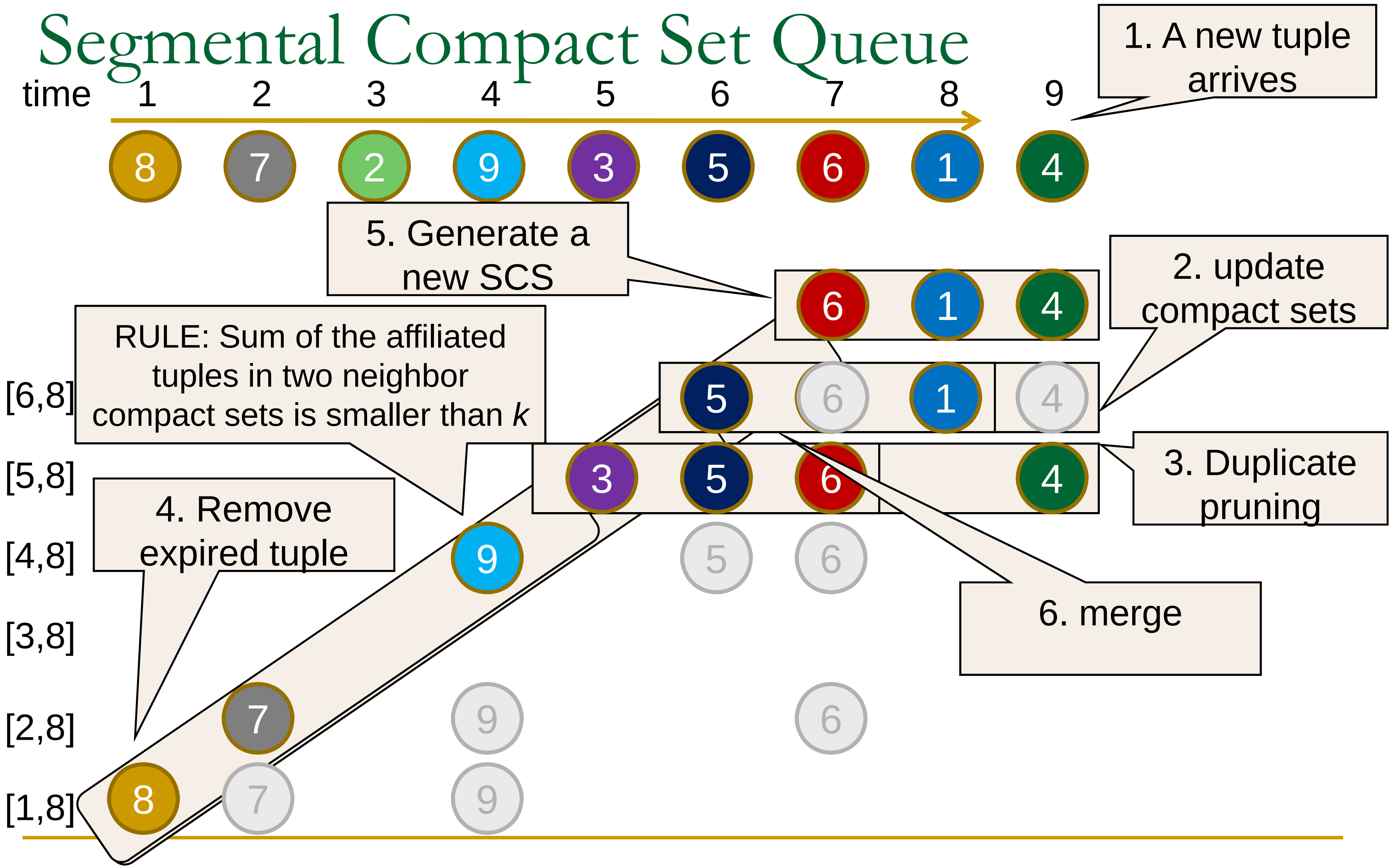
Synopsis (2): Compressed Compact Set Queue



Compressed Compact Set Queue: Analysis

- Advantages
  - The space consumption is reduced.
- Disadvantages
  - Lots of compact sets must be generated and checked for each incoming tuple, which results in high per-tuple processing cost.
- Solution
  - Group neighbor compact sets!
  - In fact, it's the combination of CSQ and CCSQ.

Synopsis (3): Segmental Compact Set Queue



Segmental Compact Set Queue: Analysis

- Advantages
  - Low space consumption.
  - Low per-tuple processing cost.
- Disadvantages
  - Some medial compact sets are unnecessary to be maintained per tuple.
- Solution
  - Use a buffer!

Synopsis (4): SCSQ-buffer

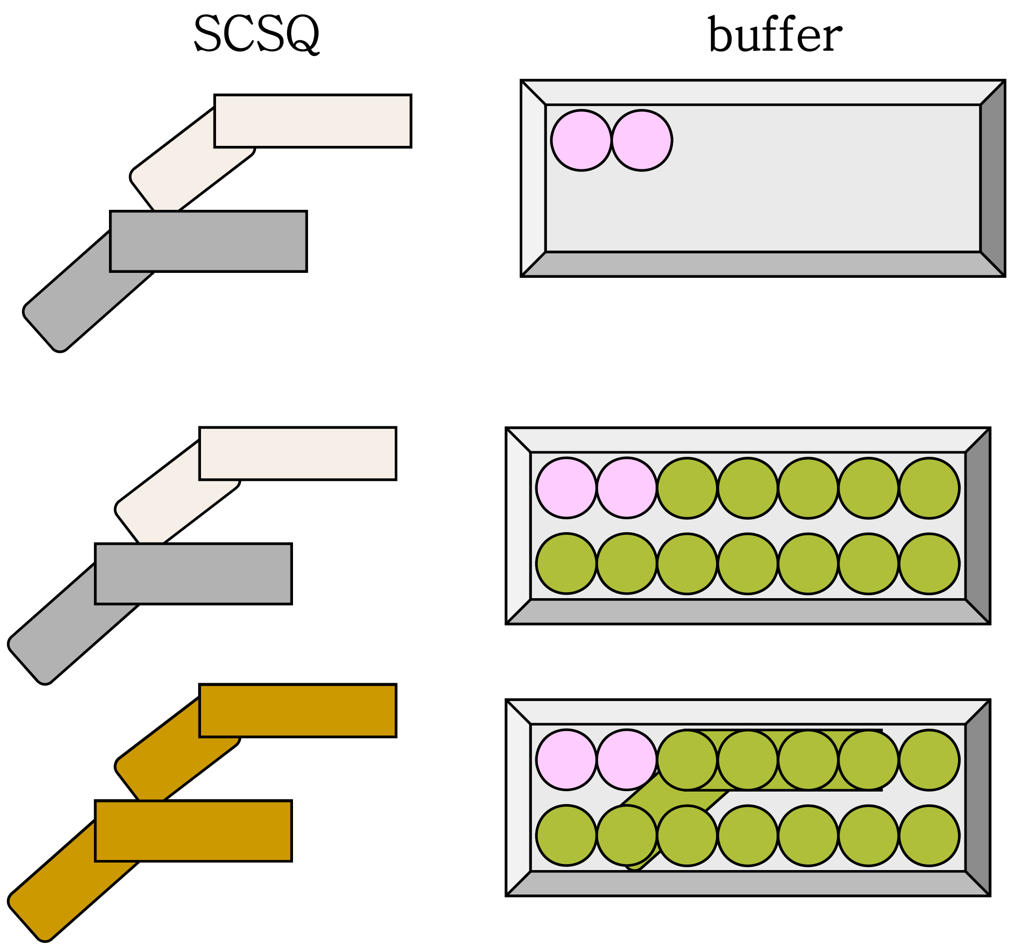
- Basic structure contains:
  - A buffer with size kH to reserve new tuples
  - A SCSQ for all tuples except the buffer
  - A compact set $C(S_W)$ for query result
- When a tuple t arrives
  - Insert t into B
  - remove out-of-date tuples in SCSQ if possible
  - If B is full, update SCSQ with B;
  - Else, update $C(S_W)$;

SCSQ-buffer

The state at time *t*

During [t+ 1, t+ 12],
1. Fill buffer till full,
2. Remover expired tuples in SCSQ

Then,
Update SCSQ,
Clear buffer



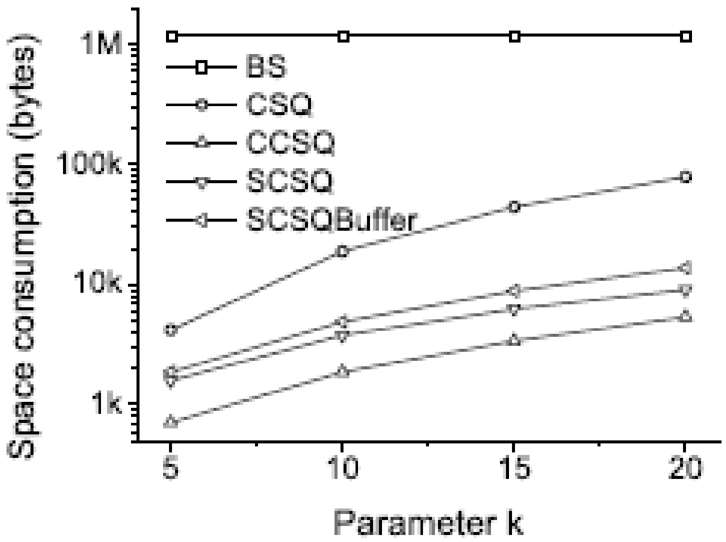
Performance summary

	Space consumption	Processing time
Basic Synopsis	$O(W + kH)$	$O(kH^2/W + \log W)$
Compact Set Queue	$O(H^2 \log W)$	$O(kH^2)$
Compressed Compact Set Queue	$O(H(k + \log W))$	$O(kH^2)$
Segmental Compact Set Queue	$O(H(k + \log W))$	$O(kH \log W)$
SCSQ-buffer	$O(H(k + \log W))$	$O(kH^2/W + \log W)$

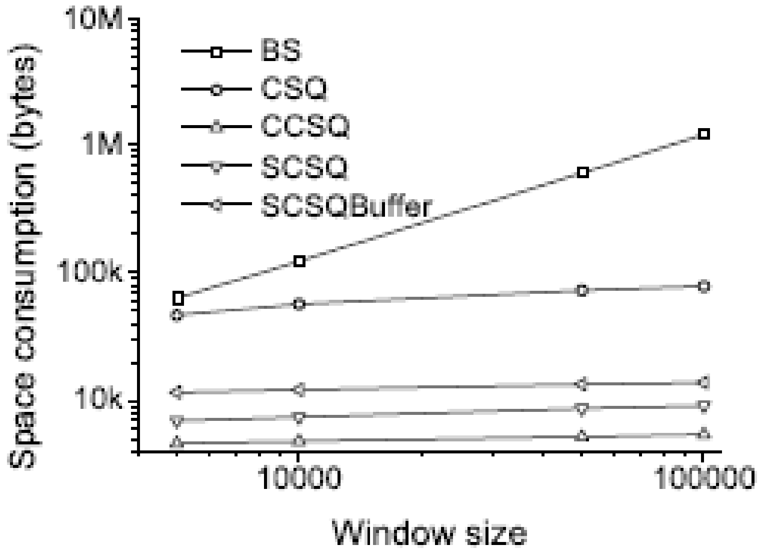
Experiments

- Dataset: International Ice Patrol (IIP) Iceberg Sightings Database
  - information on iceberg activity in North Atlantic to monitor iceberg danger near the Grand Banks
  - Sighting signals
    - R/V (radar and visual) – 0.8
    - VIS (visual only) – 0.7
    - RAD(radar only) – 0.6
    - SAT-LOW(low earth orbit satellite) – 0.5
    - SAT-MED (medium earth orbit satellite) – 0.4
    - SAT-HIGH (high earth orbit satellite) – 0.3
    - EST (estimated, used before 2005) – 0.4
  - we created a 1,000,000-record data stream by repeatedly selecting records randomly.

Space consumption

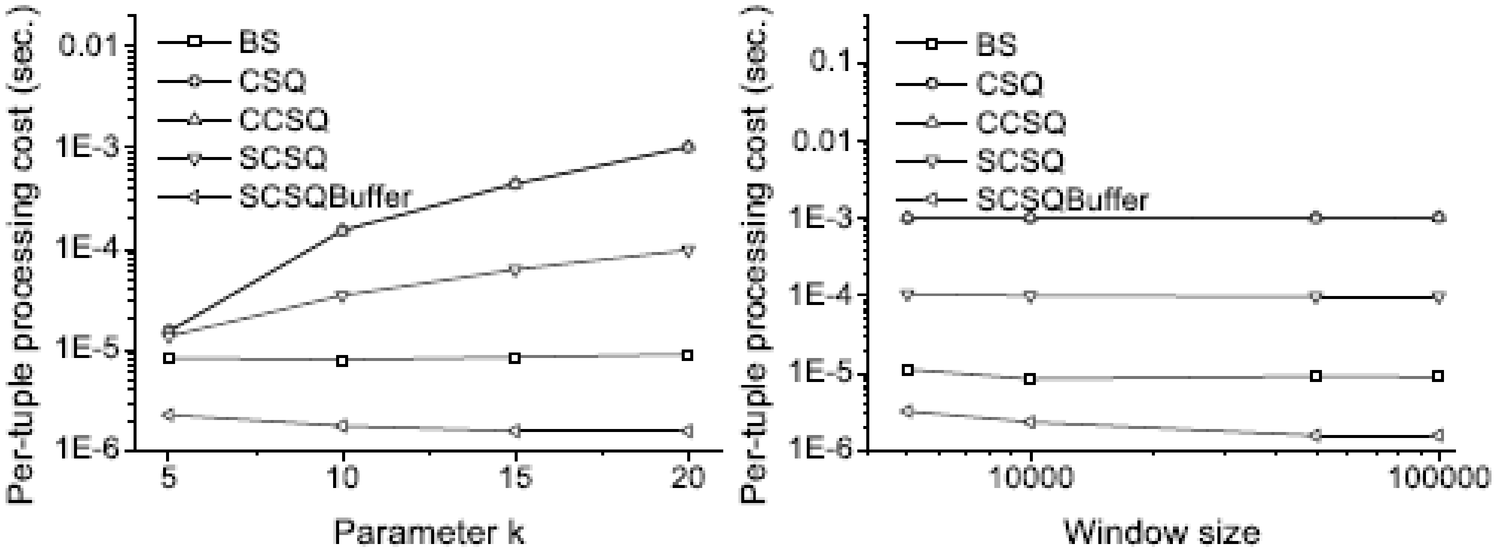


(a) Varying k ($W = 100,000$)

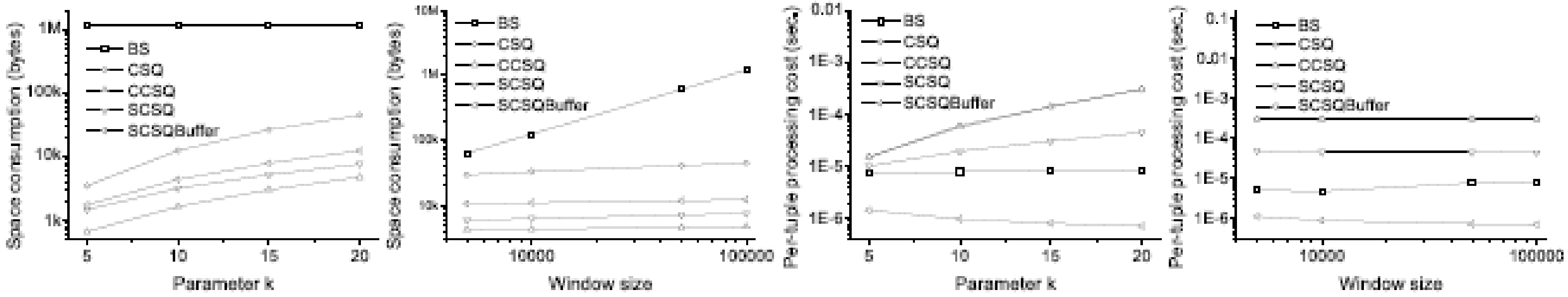


(b) varying W (k=20)

Per-tuple processing cost

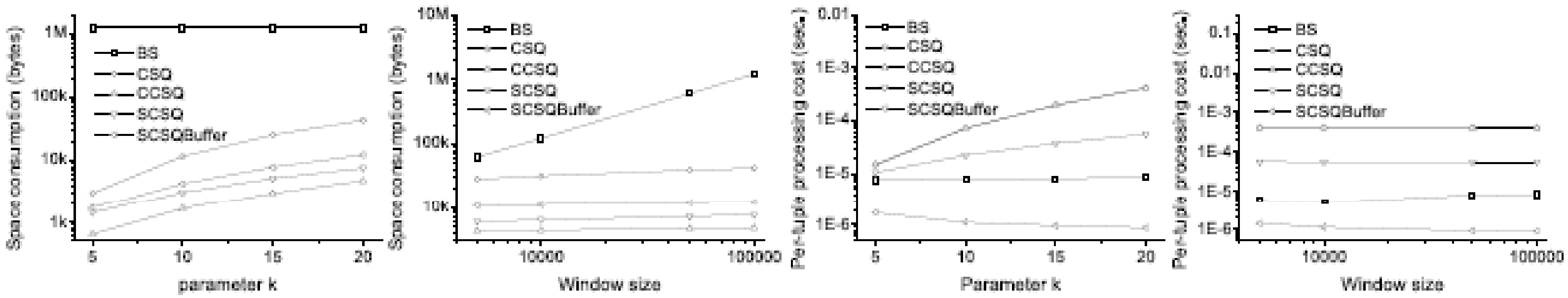


(a) Varying k ($W = 100,000$) (b) varying W (k=20)



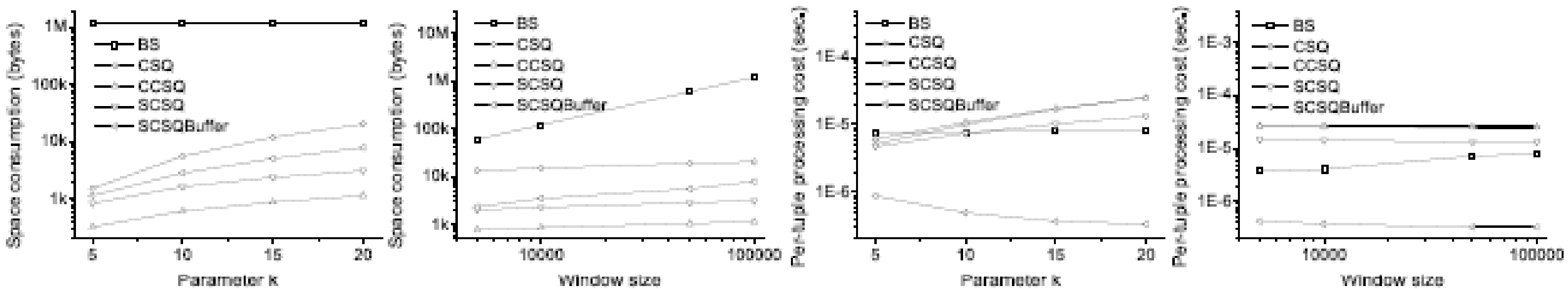
(a) Space used ($W = 10^5$) (b) Space used ($k = 20$) (c) Per-tuple cost ($W = 10^5$) (d) Per-tuple cost ($k = 20$)

Figure 8: PT-k query on real dataset



(a) Space used ($W = 10^5$) (b) Space used ($k = 20$) (c) Per-tuple cost ($W = 10^5$) (d) Per-tuple cost ($k = 20$)

Figure 9: U-kRanks query on real dataset



(a) Space used ($W = 10^5$) (b) Space used ($k = 20$) (c) Per-tuple cost ($W = 10^5$) (d) Per-tuple cost ($k = 20$)

Figure 10: U-top$k$ query on real dataset

My idea to produce a good paper: ——Doing as much as possible

Find a valuable problem → Application arguable? → Hard to sell

Keep working ↓

Get a basic idea → Write it down with simple experiments → At most a small paper

Keep working ↓

Doing more extended work and experiments → Submit it now → A potential good paper

Keep working ↓

Improve writing skill,
Reorganize contents,
Add examples → A qualified paper

Conclusion

- Conclusion
  - We propose a general framework to process Sliding-window Top-k queries on uncertain streams.
  - Support U-Topk, U-kRanks, PT-k, Pk-Topk.
  - Our work is the first work in processing sliding-window queries on uncertain streams.

Future work

- Handle other kinds of queries with this framework
- Handle more complex uncertain models, such as with constraints?
- Attribute level uncertainty?
- Using probability density function?
- Distributed uncertain stream?
- From top-k to NN query, range query, or skyline query?

Reference (uncertain top-k queries)

- [1] J. Chen and K. Yi. Dynamic structures for top-k queries on uncertain data. In *Proc. of ISAAC*, 2007.
- [2] M. Hua, J. Pei, W. Zhang, and X. Lin. Efficiently answering probabilistic threshold top-k queries on uncertain data. In *Proc. of ICDE*, 2008.
- [3] M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking queries on uncertain data: A probabilistic threshold approach. In *Proc. of SIGMOD*, 2008.
- [4] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang. Top-k query processing in uncertain databases. In *Proc. of ICDE*, 2007.
- [5] K. Yi, F. Li, G. Kollios, and D. Srivastava. Efficient processing of top-k queries in uncertain databases. In *Proc. of ICDE*, 2008.

Reference (uncertain stream processing)

- [6] C. C. Aggarwal and P. S. Yu. A framework for clustering uncertain data streams. In *Proc. of ICDE*, 2008.
- [7] G. Cormode and M. Garofalakis. Sketching probabilistic data streams. In *Proc. of ACM SIGMOD*, 2007.
- [8] T. Jayram, S. Kale, and E. Vee. Efficient aggregation algorithms for probabilistic data. In Proc. of SODA, 2007.
- [9] T. Jayram, A. McGregor, S. Muthukrishnan, and E. Vee. Estimating statistical aggregates on probabilistic data streams. In *Proc. of PODS*, 2007.
- [10] C. Jin, K. Yi, L. Chen, Jeffrey X. Yu and X. Lin. Sliding-window top-k queries on uncertain streams. In *Proc. Of VLDB*, 2008
- [11] Q. Zhang, F. Li, and K. Yi. Finding frequent items in probabilistic data. In *Proc. of SIGMOD*, 2008.

Questions?

Thanks.