

---

# Sliding-window Top-k Queries on Uncertain Streams

---

Jeffrey Xu Yu

Chinese University of Hong Kong

yu@se.cuhk.edu.hk

This work is collaborated among Cheqing Jin, Ke Yi, Lei Chen, Jeffrey Xu Yu, and Xuemin Lin.

---

# Outline

- Introduction
- Top-k queries on uncertain data
- Our solution on uncertain streams
- Experiments
- Conclusion

---

# Uncertainty in Data

- **Sensor networks**
    - Sensor readings are often imprecise due to sensors and periodic reporting mechanisms
  - **Mobile equipments**
    - A mobile object reports its position periodically, the exact location is often uncertain
  - **Social data collections**
    - Errors and estimations inherent in customer surveys and sampling
-

---

# Possible Worlds

- A possible world
    - a possible snapshot that may be observed
  - Uncertain object model
    - A possible world = a set of instances of uncertain objects
    - At most one instance per object in a possible world
  - Probabilistic database model
    - A possible world = a set of tuples
    - At most one tuple per generation rule in a possible world
  - A possible world carries an existence probability
-

# Example

ID	Reading Info	Speed ( $\times 10$ )	prob.
1	AM 10:33, Honda, X-123	5	0.8
2	AM 10:35, Toyota, Y-245	6	0.5
3	AM 10:37, Mazda, Z-341	8	0.4
4	AM 10:38, Benz, W-541	2	0.4

# Possible Worlds

**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

**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				

# Top- $k$ queries

- U-Top $k$ 
  - returns the top- $k$  tuples in all possible worlds with maximum probability.
- U- $k$ Ranks
  - returns the winner for the  $i$ -th rank for all  $1 \leq i \leq k$ .
- PT- $k$ 
  - 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.

# Top- $k$ queries

- U-Top $k$ 
  - returns the top- $k$  tuples in all possible worlds with maximum probability.
- U- $k$ Ranks
  - returns the winner for the  $i$ -th rank for all  $1 \leq i \leq k$ .
- PT- $k$ 
  - 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 $k$ 
  - returns the  $k$  most probable tuples of being the top- $k$  among all.
  - A slight modification of PT- $k$ , without threshold  $p$ .

# Example (k=2)

Query	U-Topk	U-kRanks	PT-k	Pk-Topk
Result	{6, 5}	{8, 5}	{5, 6, 8}	{5, 6}

## 16 possible world instances

### Explain:

- {6, 5} is top-k in two possible worlds.
- Prob.=0.096+ 0.144=0.24.

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	<b>6, 5, 2</b>	<b>.096</b>
<b>6, 5</b>	<b>.144</b>	6, 2	.024	6	.036
5, 2	.096	5	.144	2	.024
Empty	.036				

# Example (k=2)

Query	U-Topk	U-kRanks	PT-k	Pk-Topk
Result	{6, 5}	{8, 5}	{5, 6, 8}	{5, 6}

## 16 possible world instances

### Explain:

- Tuple 8: Rank-1 with prob.=0.4
- Tuple 5: Rank-2 with prob.=0.4

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				

# Example (k=2)

Query	U-Topk	U-kRanks	PT-k	Pk-Topk
Result	{6, 5}	{8, 5}	{5, 6, 8}	{5, 6}

## 16 possible world instances

### Explain:

- Assume threshold  $p=0.3$
- Tuple 5: 0.64
- Tuple 6: 0.5
- Tuple 8: 0.4

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				

# Example (k=2)

Query	U-Top $k$	U- $k$ Ranks	PT- $k$	<b>P<math>k</math>-Top<math>k</math></b>
Result	{6, 5}	{8, 5}	{5, 6, 8}	<b>{5, 6}</b>

## 16 possible world instances

### Explain:

- Tuple 5: 0.64
- Tuple 6: 0.5
- Tuple 8: 0.4
- Returns top-2

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				

---

# Related Work

- [Soliman et al. ICDE 2007]
  - Definitions and solutions for U-top $k$  and U- $k$ Ranks queries
- [Yi et al. ICDE 2008]
  - New solutions for U-top $k$  and U- $k$ Ranks
- [Hua et al. ICDE 2008]
  - Definition and solution for PT- $k$  query
- [Hua et al. SIGMOD 2008]
  - New solutions for PT- $k$  query

---

# Sliding-Window on Uncertain Streams

- Rank on uncertain streams
- Use a sliding window
- Process sliding-window queries on uncertain streams
- Consider both time/space factors

---

# An Overview

- Three steps
  - Design the compact set
    - A small subset of the original set
    - Self-maintenance
    - Capable of answering top-k query
  - Find all of compact sets useful in future
    - $W$  different windows. i.e.,  $[t-j, t]$ , for  $j=0..W-1$
    - One compact set for each window
  - Techniques to remove duplicates.

---

# Compact Set

- Let  $D$  be an uncertain database.
- A compact set  $C(D)$  is a small set of  $D$ , which is capable of answering the top- $k$  query for  $D$ .
- Compact Set is used to answer rank queries instead of the entire uncertain database.

# Notations

- $D_i$ : the subset of  $D$  containing the first  $i$  tuples in  $D$ .
- $r_{i,j}$ : the probability that a randomly generated world from  $D_i$  has exactly  $j$  tuples.
- $p(t_i)$ : the occurring probability of tuple  $t_i$ .
- $p(t_i)r_{i-1,j-1}$ : the probability that  $t_i$  ranks the  $j$ -th in a randomly generated world from  $D$ .
- $p(t_i)\sum_{i=1..k}r_{i-1,j-1}$ : the probability that  $t_i$  ranks top- $k$  in all possible worlds generated from  $D$ .
- $r_{i,j}$  can be computed using dynamic program.
  - (1) If  $i \geq j \geq 0$ :  $r_{i,j} = p(t_i)r_{i-1,j-1} + (1-p(t_i))r_{i-1,j}$
  - (2) If  $i=j=0$ :  $r_{i,j}=1$
  - (3) Else:  $0$

# Compact Set for Pk-Topk

- The compact set  $C(D)$  for Pk-Topk is the smallest subset of  $D$ .

$$p(t_\alpha) \sum_{1 \leq l \leq k} r_{\alpha-1, l-1} \geq \sum_{1 \leq l \leq k} r_{d, l-1}$$

- Such a compact set is sufficient to answer a Pk-Topk query
  - for any tuple  $t_i$  in  $D - C(D)$ , the probability of being top- $k$  member is smaller than:  $\sum_{l=1..k} r_{d, l-1}$

# Example (1)

## ■ Original dataset

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

## After Ranking

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

## ■ Calculate 2D array $r_{i,j}$

r	0	1	2
0	1	0	0
1	0.6	0.4	0
2	0.3	0.5	0.2
3	0.06	0.34	0.44
4	0.036	0.228	0.4

$r_{0,0}=1$ , according to (2)

$r_{0,1}=r_{0,2}=0$ , according to (3)

$r_{1,0}=(1-0.4)*1+0.4*0=0.6$ , according to (1)

$r_{1,1}=0.4*1+(1-0.4)*0=0.4$ , according to (1)

# Example (2)

- Let  $k=2$ .
  - First,  $C(D) = (8)$ 
    - Bound =  $0.6 + 0.4 = 1.0$
    - Prob. of 8:  $0.4*(1+0)=0.4$
    - Prob. of 6:  $0.5*(0.6+0.4)=0.5$
    - No such  $t_a$  exists.
  - Second,  $C(D) = (8, 6)$ 
    - Bound =  $0.3 + 0.5 = 0.8$
    - Prob. of 5:  $0.8*(0.3+0.5)=0.64$
    - No such  $t_a$  exists.
  - Third,  $C(D) = (8, 6, 5)$ 
    - Bound =  $0.06 + 0.34 = 0.4$
    - Stop! Because  $0.5 > 0.4$ , and  $0.4 = 0.4$

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

r	0	1	2
0	1	0	0
1	0.6	0.4	0
2	0.3	0.5	0.2
3	0.06	0.34	0.44
4	0.036	0.228	0.4

First bound

Second bound

Last bound

# Properties of 2D array $r_{i,j}$ (1)

- Let  $q_{i,j} = r_{i,j+1}/r_{i,j}$ . The value of  $q_{i,j}$  monotonically decreases for any tuple  $t_i$ , when  $j \geq 1$ .

- Example 1 ( $q_{2,*}$ )

- $q_{2,0} = 0.5/0.3 = 1.7$
- $q_{2,1} = 0.2/0.5 = 0.4$
- $q_{2,0} > q_{2,1}$

- Example 2 ( $q_{3,*}$ )

- $q_{3,0} = 0.34/0.06 = 5.7$
- $q_{3,1} = 0.44/0.34 = 1.3$
- $q_{3,0} > q_{3,1}$

r	0	1	2
0	1	0	0
1	0.6	0.4	0
2	0.3	0.5	0.2
3	0.06	0.34	0.44
4	0.036	0.228	0.4

# Properties of 2D array $r_{i,j}$ (2)

- If  $i < j$ , we have  $q_{i,l} \leq q_{j,l}$ , where  $l \geq 0$ .

Array  $r_{i,j}$

r	0	1	2
0	1	0	0
1	0.6	0.4	0
2	0.3	0.5	0.2
3	0.06	0.34	0.44
4	0.036	0.228	0.4



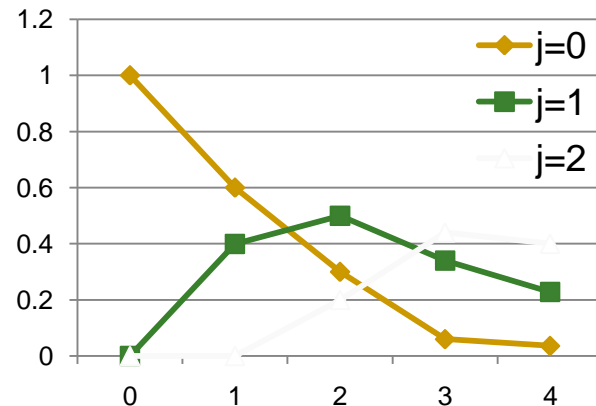
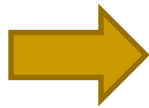
Array  $q_{i,j}$

r	0	1
0	0	0
1	0.7	0
2	1.7	0.4
3	5.8	1.3
4	6.3	1.8

# Properties of 2D array $r_{i,j}$ (3)

- For any tuple  $t_i$ , the series  $r_{i,j}$  is unimodal, i.e., there exists some  $m$  such that  $r_{i,j}$  is monotonically increasing when  $j < m$  while monotonically decreasing when  $j > m$ .

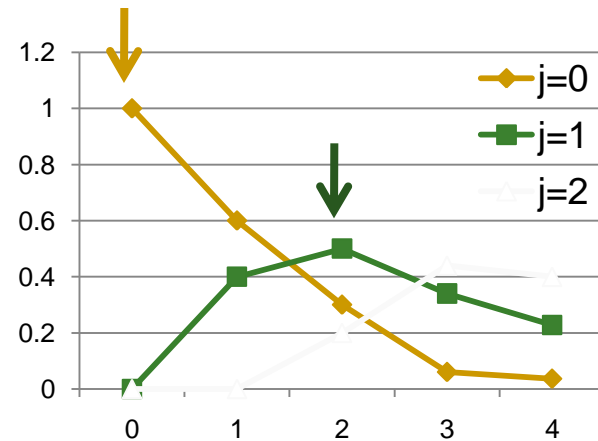
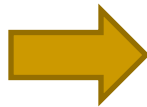
r	0	1	2
0	1	0	0
1	0.6	0.4	0
2	0.3	0.5	0.2
3	0.06	0.34	0.44
4	0.036	0.228	0.4



# Properties of 2D array $r_{i,j}$ (4)

- If  $i < j$ , the peak point of the corresponding series for  $t_i$  is no later than  $t_j$ .

r	0	1	2
0	1	0	0
1	0.6	0.4	0
2	0.3	0.5	0.2
3	0.06	0.34	0.44
4	0.036	0.228	0.4



---

# Remark

- The compact set is self-maintainable under insertion.
  - However tuples **expire**.
    - Such a  $C(D)$  for insertions is not enough.
  - In addition, how much memory is needed?
-

---

# Random Order Stream

- We consider a more general scenario: random order stream model.
  - The fields' values and the probability of each tuple is randomly picked up.

# The Worst Case Situation

- Consider an ultimate situation where any tuple must be reserved in memory!
- Example
  - $t_i$ : the  $i$ -th tuple in a stream, (**value:** $1/n$ , **prob.:** $1/n$ ).
  - Let window size be  $W$ .
  - For any given  $k$ , tuple  $t_{n-W+1}$  is at the query result of  $Pk$ -Top $k$  query.
- There does not exist any good synopses to reduce memory.

# Synopsis for sliding window

- Basic Synopsis (BS)
  - Reserve every tuple in memory
  - Maintain an 2D  $r_{i,j}$  array for the compact set  $C(D)$ 
    - When a new tuple comes, **drop** the out-of-date tuple from  $C(D)$  if possible; **insert** the new tuple into  $C(D)$  if its ranks is higher than the tail element in  $C(D)$ ;
  - Build a tree on all  $W$  tuples to hasten process.
- Analysis
  - Time-efficient, but space-inefficient.
  - The space complexity is  $O(W)$ .

---

# Synopsis for sliding window

- Big idea
  1. List all possible compact sets
  2. Eliminate duplicated compact sets
  3. Compress them
- The new novel synopsis
  - Compact Set Queue (CSQ)
  - Compressed Compact Set Queue (CCSQ)
  - Segmental Compact Set Queue (SCSQ)
  - SCSQ with buffer (SCSQ-buffer)

# Possible sub-windows

- Assume window size  $W=8$ ,  $k=3$



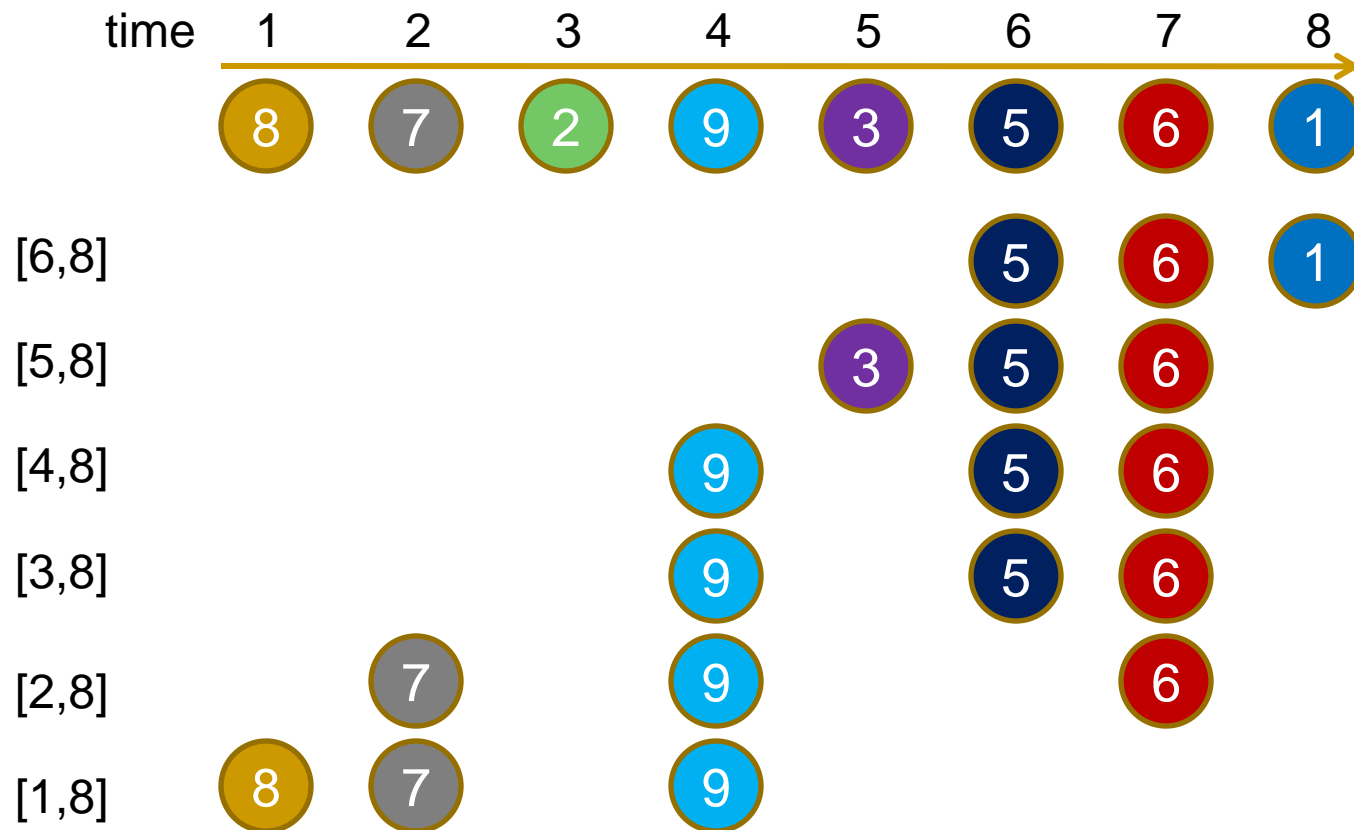
Possible sub-windows



Create a compact set for each sub-window!

# Creating Compact Sets

## ■ Possible compact sets



---

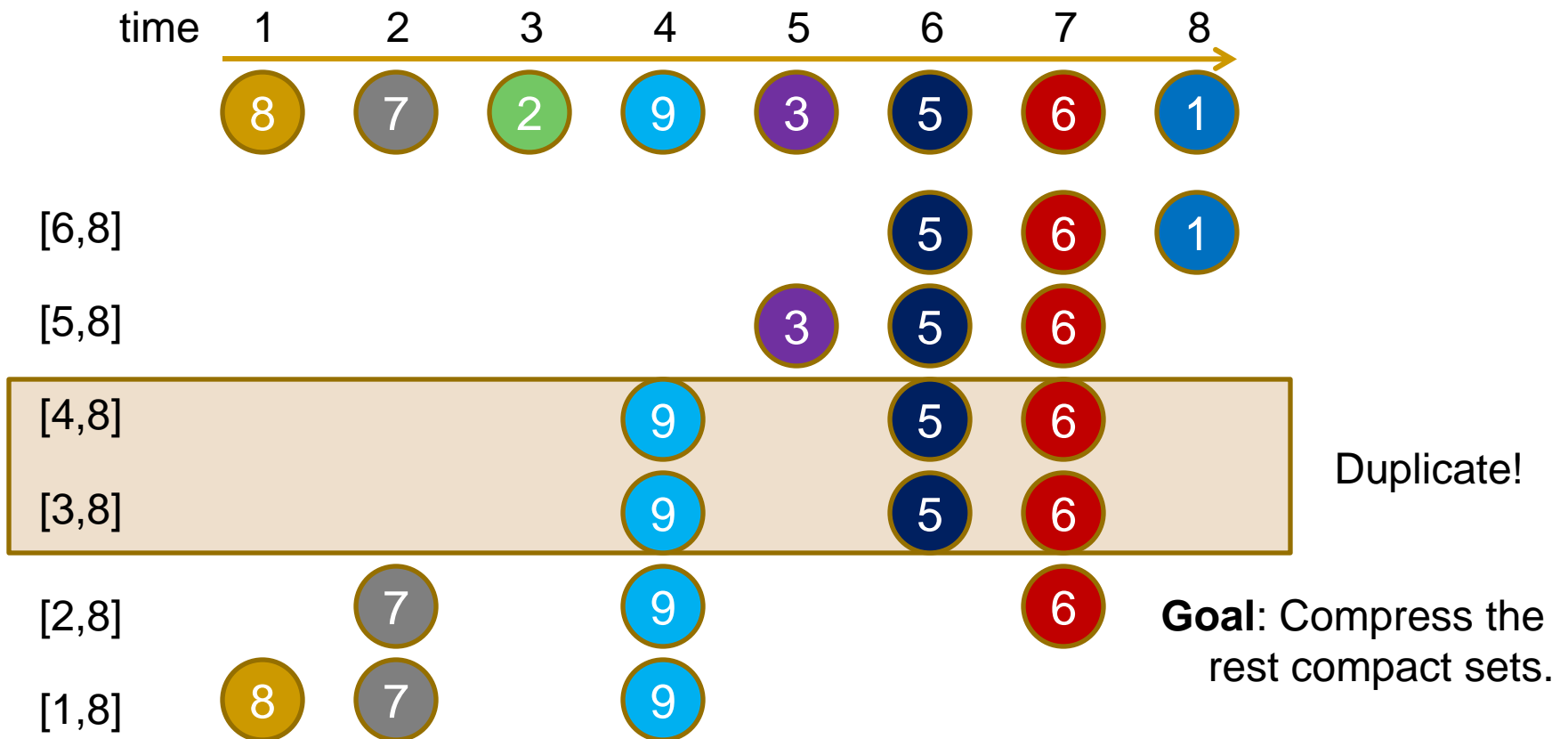
## Algorithm 1 MaintainCSQ

---

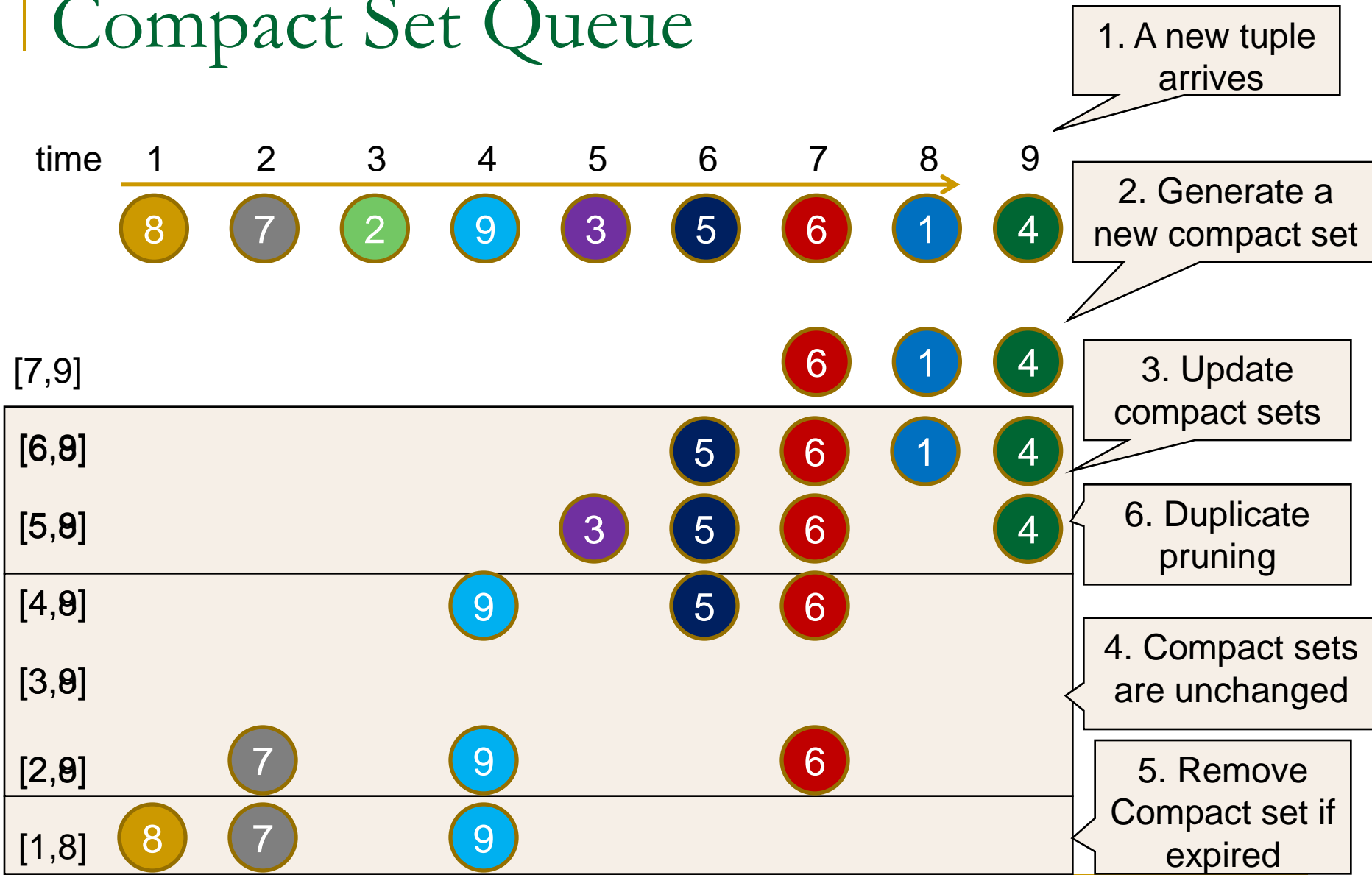
- 1: Tuple set  $D = \emptyset$ ; compact set queue  $\Psi = \emptyset$ ;
  - 2: **for** each arriving tuple  $t$
  - 3:     insert  $t$  into  $D$ ;
  - 4:     **if** (successfully create a compact set  $C(D)$  for  $D$ )
  - 5:         append  $C(D)$  to  $\Psi$ ;
  - 6:         remove tuples in  $D$  older than  $t''$  (including  $t''$ ), where  $t''$  is the oldest tuple in  $C(D)$ ;
  - 7:     **for** (each compact set  $C(S_i) \in \Psi$  from new to old)
  - 8:         **if** ( $t \prec_f$  lowest ranked tuple in  $C(S_i)$ )
  - 9:             update  $C(S_i) := C(C(S_i) \cup \{t\})$ ;
  - 10:             **if** ( $C(S_i) =$  the previous compact set in  $\Psi$ )
  - 11:                 remove  $C(S_i)$  from  $\Psi$ ;
  - 12:             **else**
  - 13:                 **break**;
  - 14:     **if** (the expiring tuple  $\in C(S_W)$ )
  - 15:         remove  $C(S_W)$  from  $\Psi$ ;
  - 16:          $C(S_W) :=$  first compact set in  $\Psi$ ;
  - 17:         compute the array  $r$  on the new  $C(S_W)$ ;
-

# Example

## ■ Eliminate duplicate compact sets



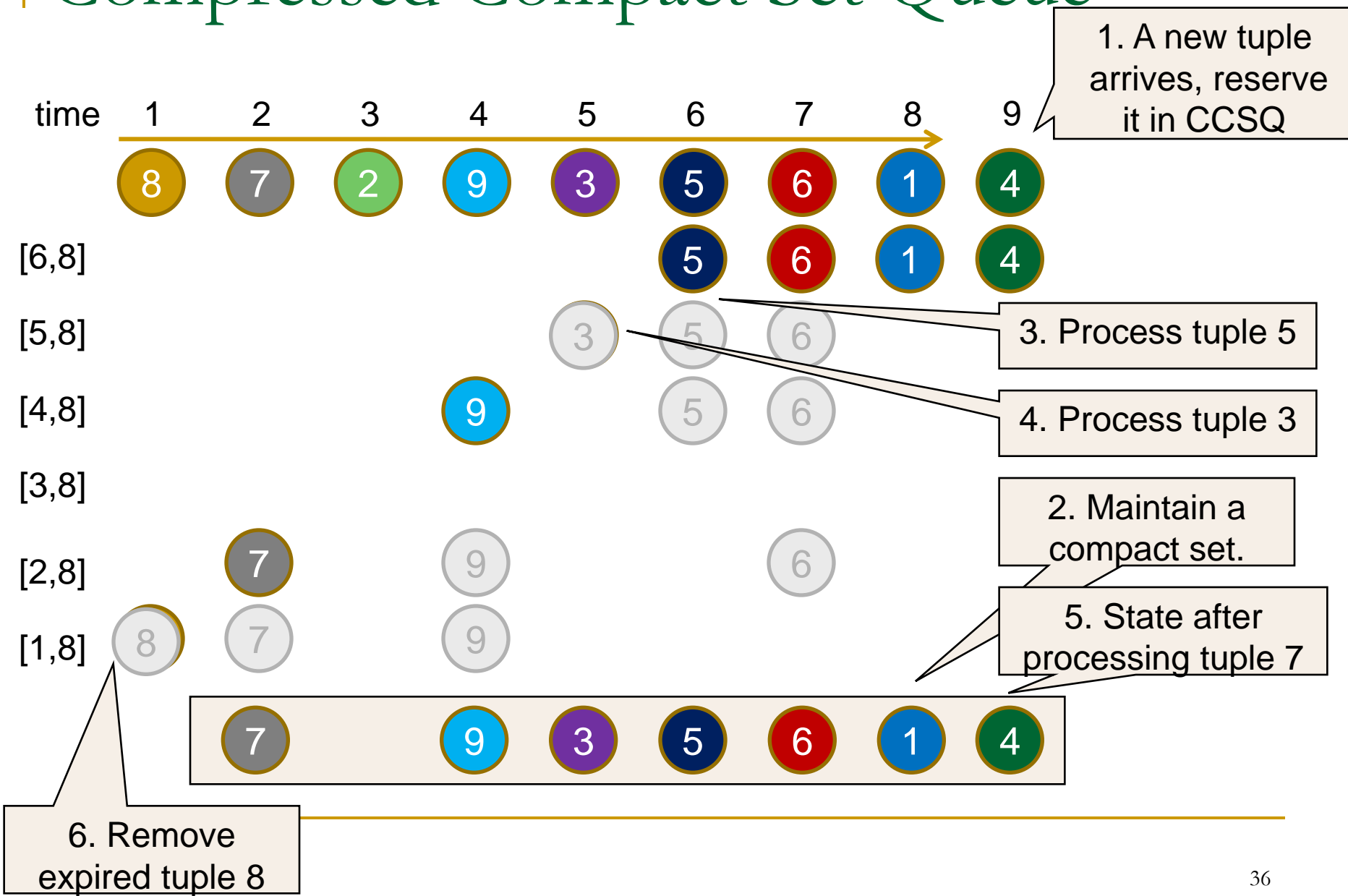
# Compact Set Queue



# Compressed Compact Set Queue

- Problem (Compact Set Queue): There is redundancy between neighbor compact sets.
- Solution: Compress neighbor compact sets!
  - Eliminate redundancy by storing only the difference between two adjacent compact sets  $C(S_i)$  and  $C(S_{i-1})$  if they are different.
  - Delta+ :  $C(S_i) - C(S_{i-1})$
  - Delta- :  $C(S_{i-1}) - C(S_i)$

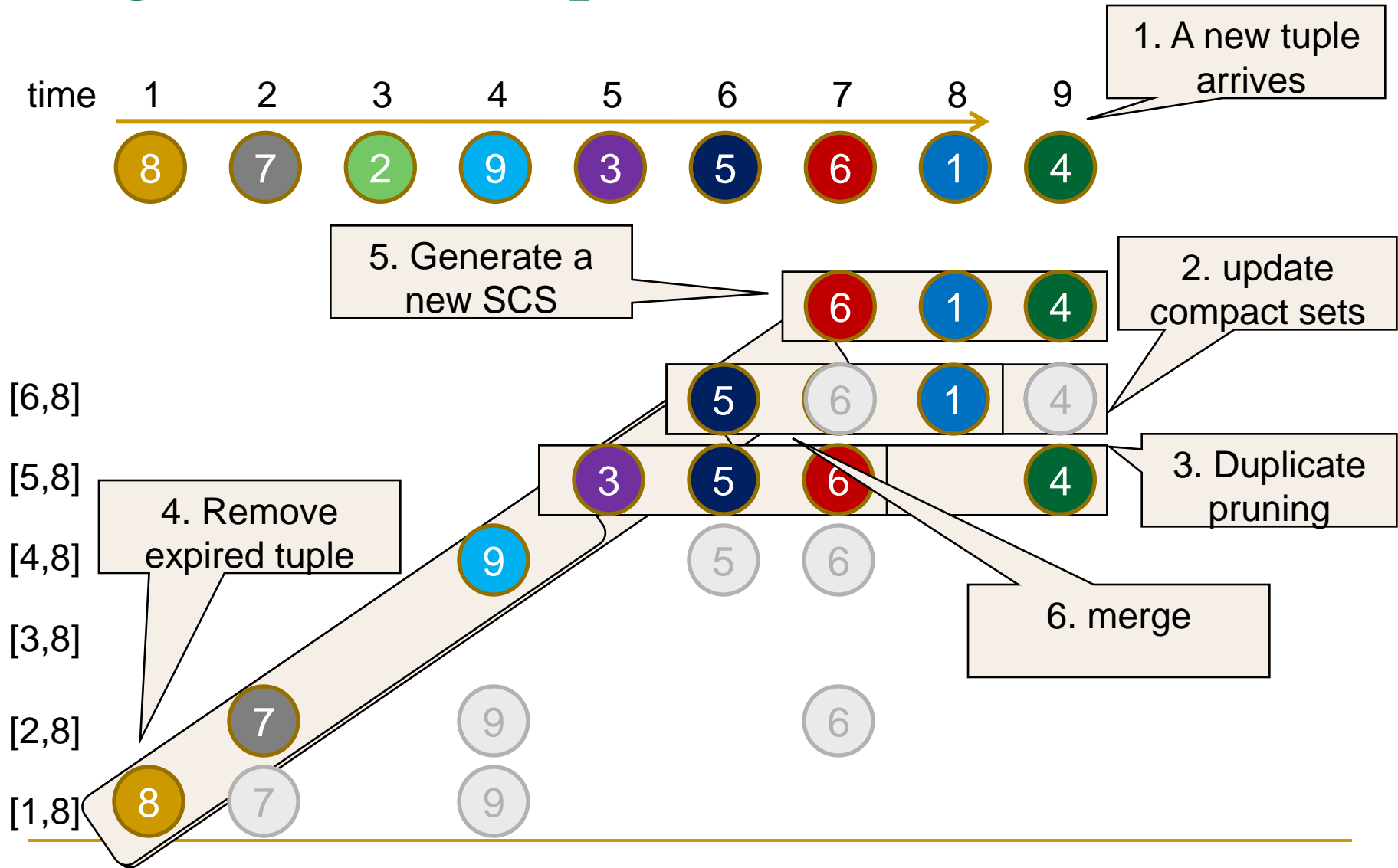
# Compressed Compact Set Queue



# Segment Compact Set Queue

- ❑ Problem (Compressed Compact Set Queue): Lots of compact sets must be generated and checked for every incoming tuple when it arrives.
- ❑ Solution: Group neighbor compact sets!
  - Only the oldest compact set  $C(S_W)$  is needed to extract the top-k query results.
  - All the other compact sets simply act as continuous “supply” for  $C(S_W)$  when it expires.

# Segmental Compact Set Queue



# SCSQ-buffer

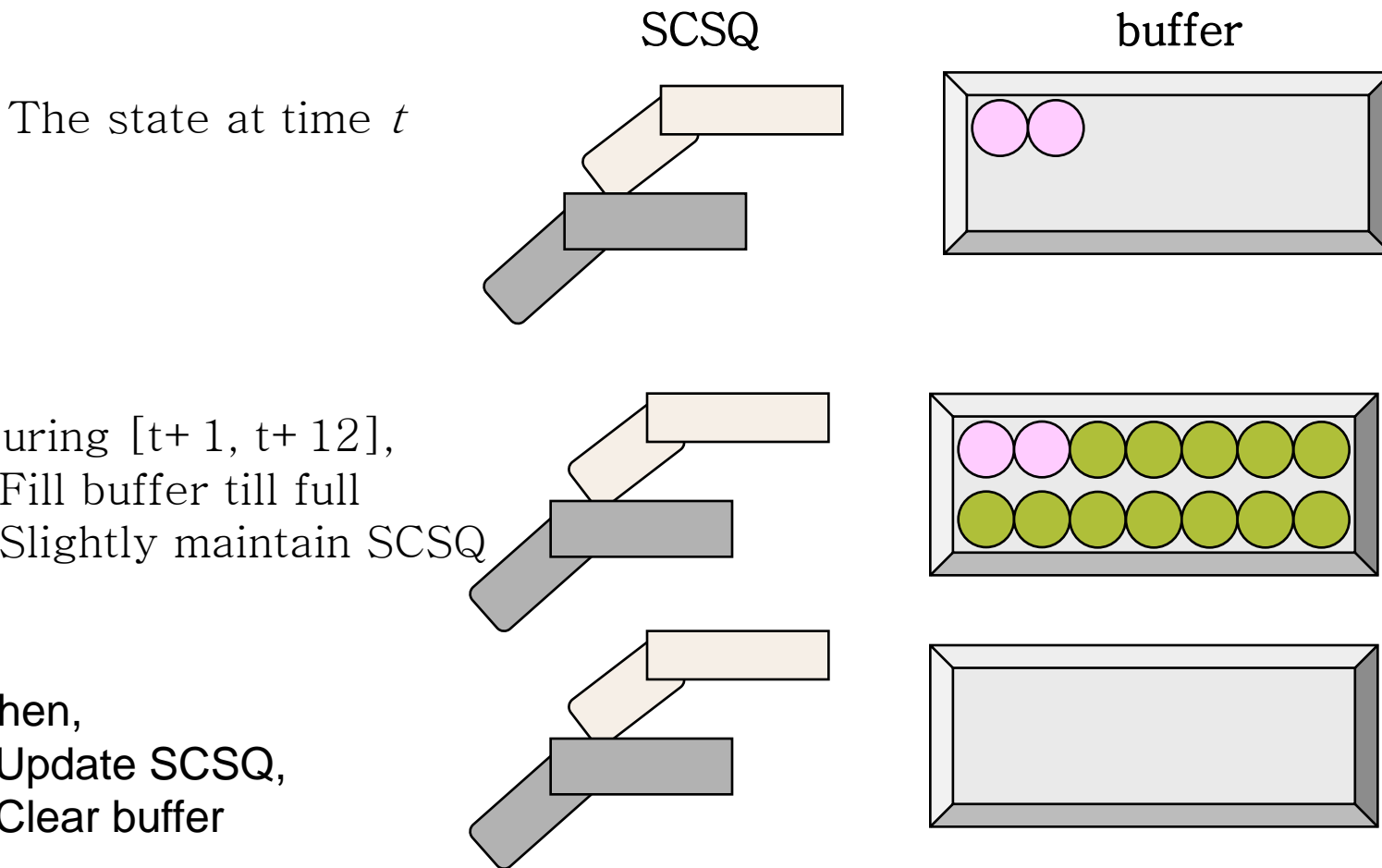
---

## Algorithm 2 BatchUpdate

---

- 1: let  $B$  be a buffer with size  $kH$ ;
  - 2: for (each arriving tuple  $t$ )
  - 3:   insert  $t$  into  $B$ ;
  - 4:   if ( $B$  is full)
  - 5:     find the smallest  $i$  such that  $B_i$  admits a compact set;
  - 6:     starting from  $i$ , build SCSQ on  $B$ ;
  - 7:     update the existing SCSQ;
  - 8:      $B = \emptyset$ ;
  - 9:   if ( $C(S_W)$  is affected)
  - 10:     update  $C(S_W)$ ;
  - 11:   remove expired compact sets in SCSQ;
-

# SCSQ-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	$O(H(k + \log W))$	$O(kH^2/W + \log W)$

# Use Compact Set to Support PT-k

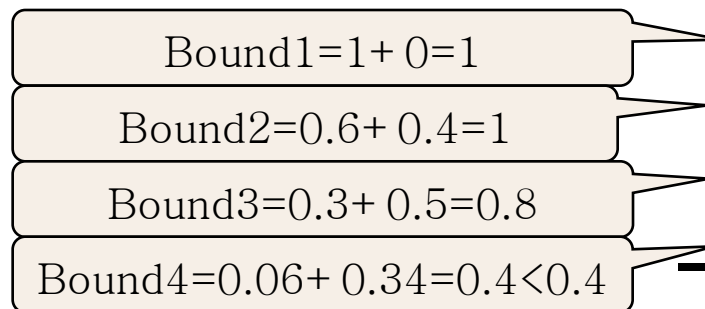
Assume  $k=2$ , threshold  $p=0.45$

**Rule:** Calculate 2D array  $r_{i,j}$  until the bound is smaller than 0.45

**Result:** compact set= $\{8, 6, 5\}$

**Analysis:**

1. If new tuple  $t$  is greater than 5, the new CS  $< \{8, 6, 5, t\}$ , because the bound for 5 is reduced.
2. If the new tuple  $t$  is smaller than 5, CS remains unchanged.



5

**Data set**

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

**2D array  $r_{i,j}$**

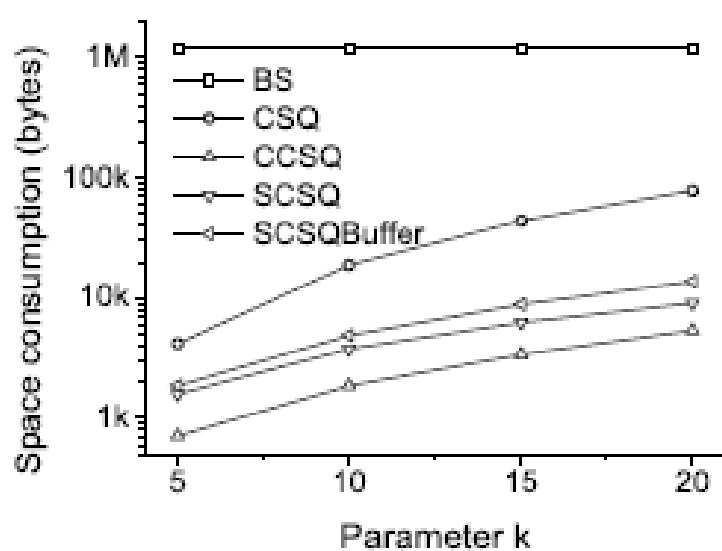
r	0	1	2
0	1	0	0
1	0.6	0.4	0
2	0.3	0.5	0.2
3	0.06	0.34	0.44
4	0.036	0.228	0.4

---

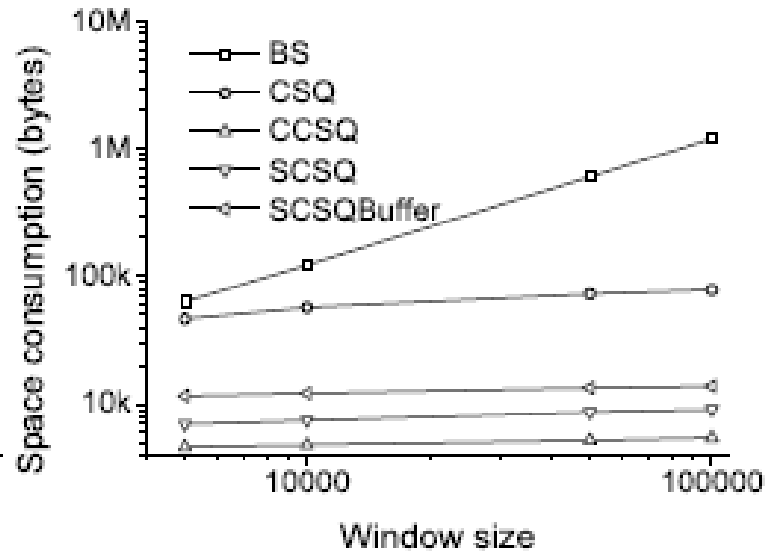
# Experiments

- Dataset: International Ice Patrol (IIP) Iceberg Sightings Database
  - Information on iceberg activity in North Atlantic to monitor iceberg danger near the Grand Banks
  - We created a 1,000,000-record data stream by repeatedly selecting records randomly.

# Space consumption

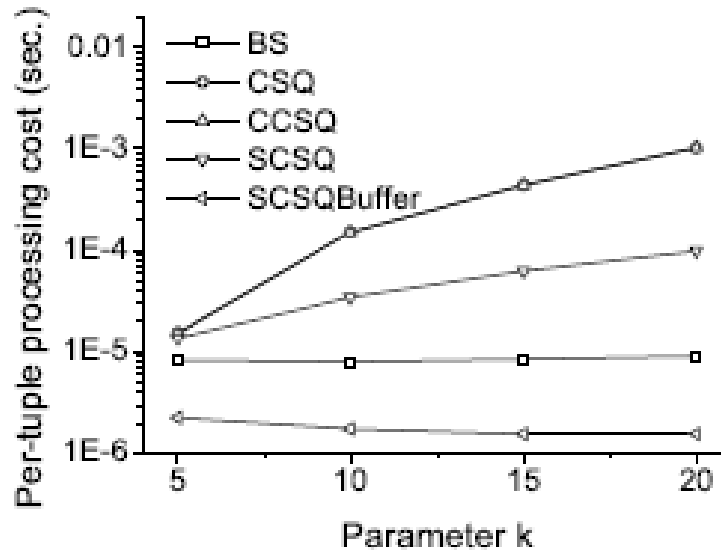


(a) Varying  $k$  ( $W = 100,000$ )

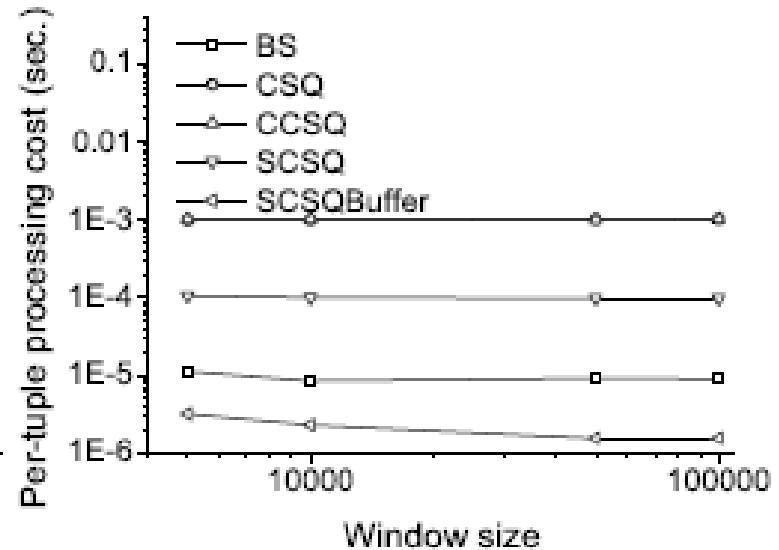


(b) varying  $W$  ( $k=20$ )

# Per-tuple processing cost



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



(b) varying  $W$  ( $k=20$ )

---

# Conclusion

- We propose a general framework to process Sliding-window Top-k queries on uncertain streams.
- Support U-kRanks, U-kRanks, PT-k, Pk-Topk
- We provide time/space complexity and conducted extensive performance studies to confirm the effectiveness of the proposed approaches.

# Reference

1. [Aggarwal et al. ICDE 2008] C. C. Aggarwal and P. S. Yu. A framework for clustering uncertain data streams. In *Proc. of ICDE*, 2008.
2. [Chen et al. ISAAC 2007] J. Chen and K. Yi. Dynamic structures for top-k queries on uncertain data. In *Proc. of ISAAC*, 2007.
3. [Cormode et al. SIGMOD 2007] G. Cormode and M. Garofalakis. Sketching probabilistic data streams. In *Proc. of ACM SIGMOD*, 2007.
4. [Hua et al. ICDE 2008] M. Hua, J. Pei, W. Zhang, and X. Lin. Efficiently answering probabilistic threshold top-k queries on uncertain data. In *Proc. of ICDE*, 2008.
5. [Hua et al. SIGMOD 2008] M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking queries on uncertain data: A probabilistic threshold approach. In *Proc. of SIGMOD*, 2008.

6. **[Jayram et al. SODA 2007]** T. Jayram, S. Kale, and E. Vee. Efficient aggregation algorithms for probabilistic data. In *Proc. of SODA*, 2007.
7. **[Jayram et al. PODS 2007]** T. Jayram, A. McGregor, S. Muthukrishnan, and E. Vee. Estimating statistical aggregates on probabilistic data streams. In *Proc. of PODS*, 2007.
8. **[Soliman et al. ICDE 2007]** M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang. Top-k query processing in uncertain databases. In *Proc. of ICDE*, 2007.
9. **[Yi et al. ICDE 2008]** K. Yi, F. Li, G. Kollios, and D. Srivastava. Efficient processing of top-k queries in uncertain databases. In *Proc. of ICDE*, 2008.
10. **[Zhang et al. SIGMOD 2008]** Q. Zhang, F. Li, and K. Yi. Finding frequent items in probabilistic data. In *Proc. of SIGMOD*, 2008.