Unsupervised Data Mining: From Batch to Stream Mining Algorithms

Prof. Dr. Stefan Kramer

Johannes Gutenberg-Universität
Mainz

Outline

- Pattern mining on streams

Acknowledgements

- Carlo Zaniolo

Pattern Mining on Data Streams

Finding Frequent Patterns for Association Rule Mining

- Given a set of transactions T and a support threshold s, find all patterns with support >= s
- Apriori [Agrawal' 94], FP-growth [Han' 00]
- Fast & light algorithms for data streams
  - Dozens of proposals
  - For mining windows over streams
  - Many data stream frameworks allow dividing streams into sliding windows

Moment

- *Moment: Maintaining closed frequent itemsets over a stream sliding window*
- *Naïve approaches:*
  1. "Regenerate frequent itemsets from the entire window whenever a new transaction comes into or an old transaction leaves the window."
  2. "Store every itemset, frequent or not, in a traditional data structure such as the prefix tree, and update its support whenever a new transaction comes into or an old transaction leaves the window."

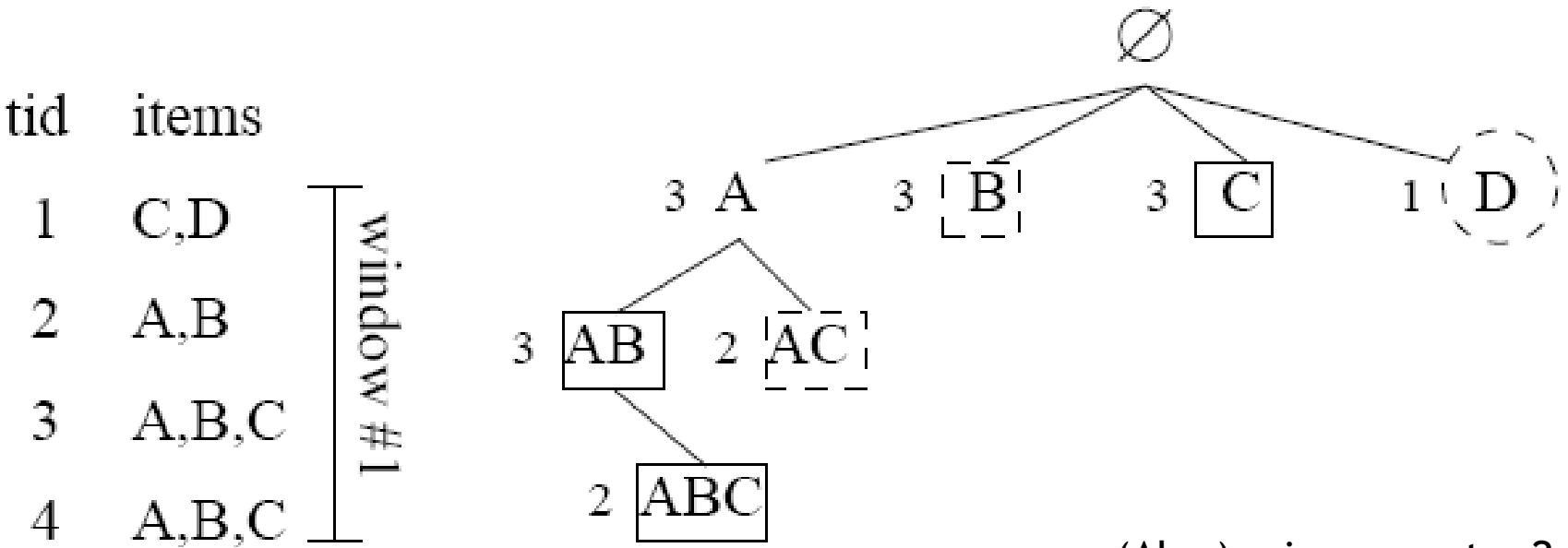
Moment Algorithm

- ***Hope:*** In the absence of concept drifts, not too many changes in status
- Maintains two types of boundary nodes:
  1. Frequent / non-frequent
  2. Closed / non-closed

Taking specific actions to maintain a shifting boundary whenever a concept shift occurs!

Closed Enumeration Tree (CET)

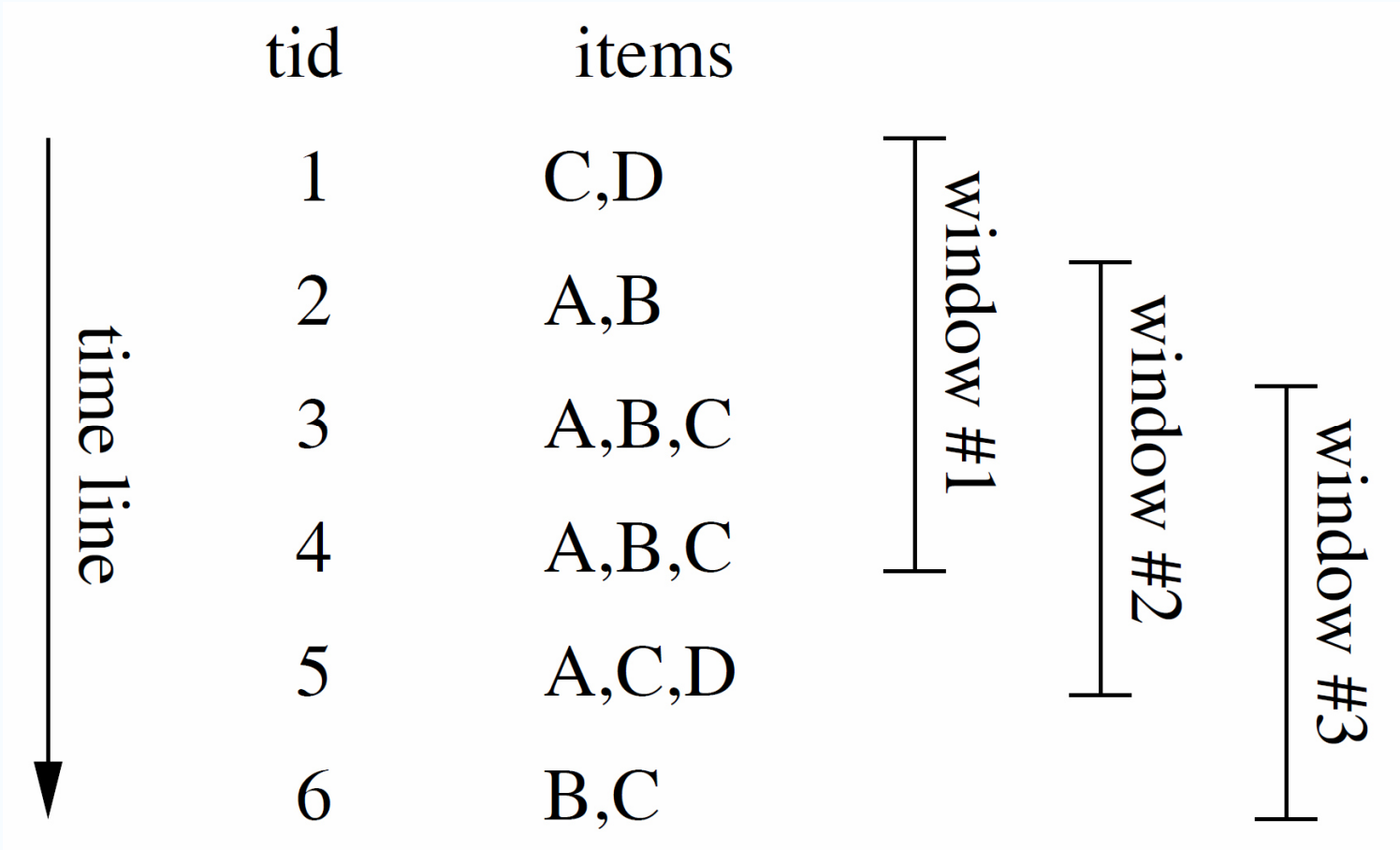
- Very similar to FP-tree, except that keeps a dynamic set of items:
  - closed frequent itemsets
  - boundary itemsets



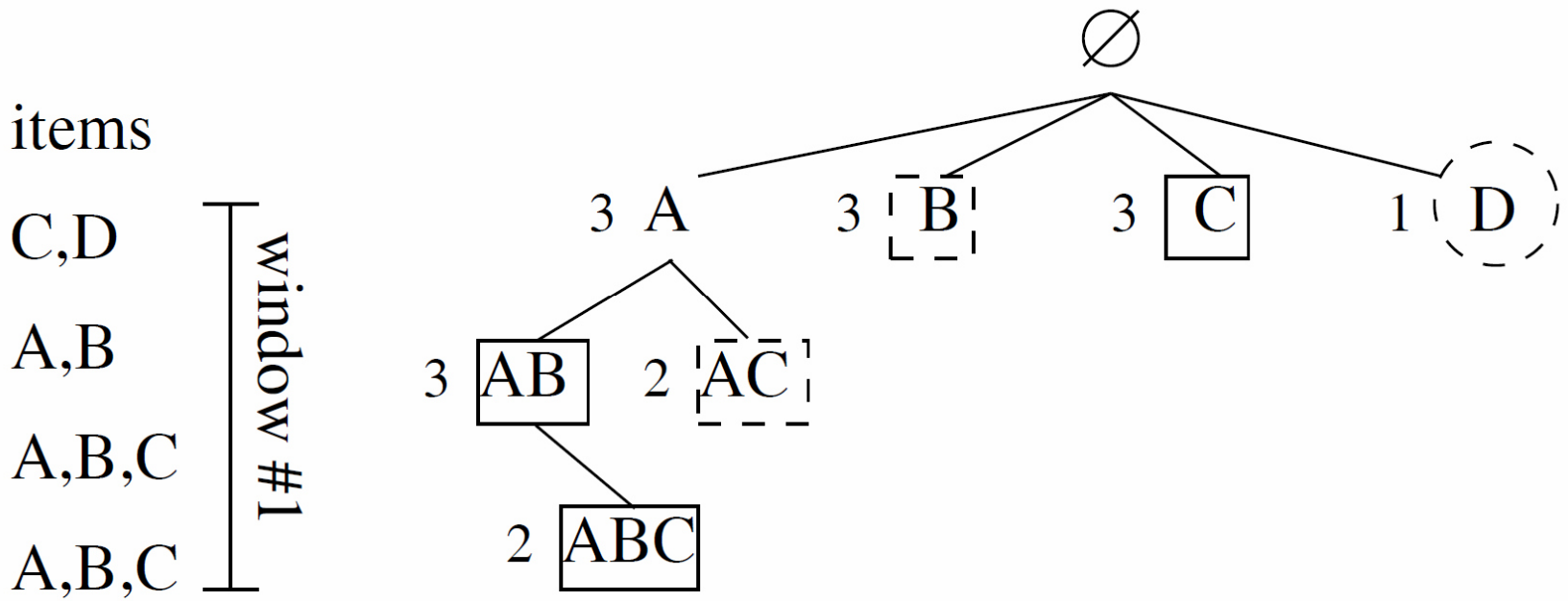
Moment Algorithm

1. **Infrequent gateway nodes**
   - Infrequent and its parent frequent or the siblings of its parent (D)
2. **Unpromising gateway nodes**
   - Frequent and it is a subset of a frequent closed itemset with same support that is also lexicographically smaller (B, AC)
3. **Intermediate nodes**
   - Frequent and has a child with same support and not unpromising (A)
4. **Closed nodes**
   - In fact: *frequent* closed (AB, ABC, C)

Running Example

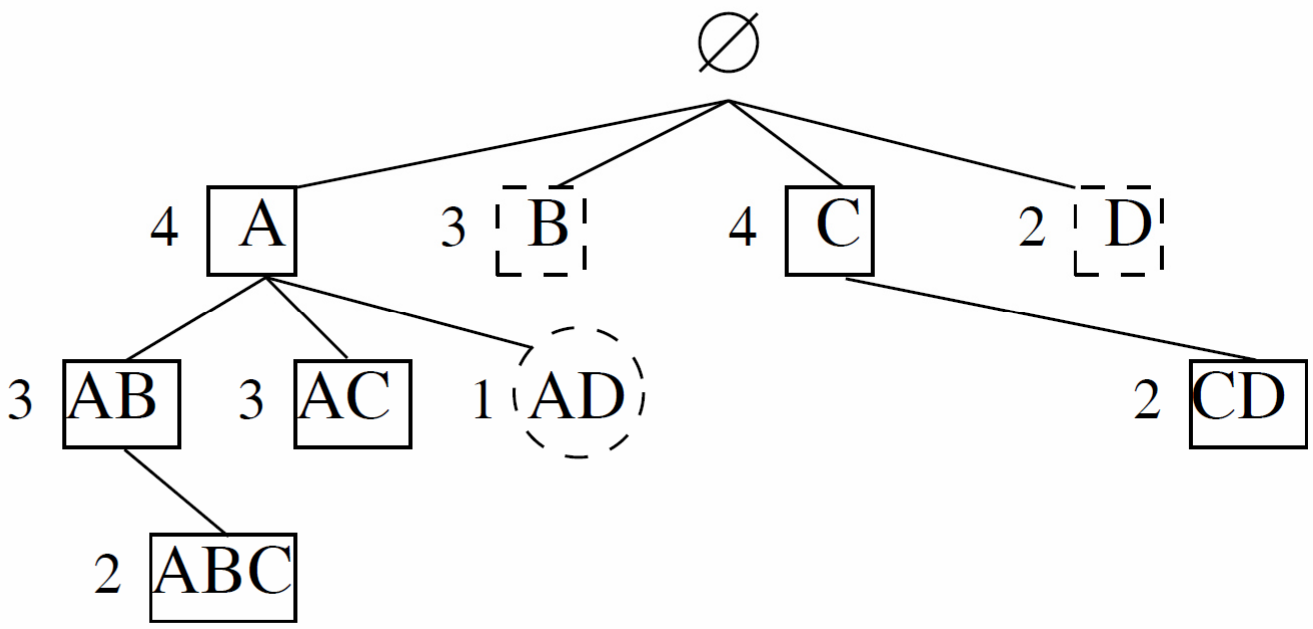


Initial State (As Before)

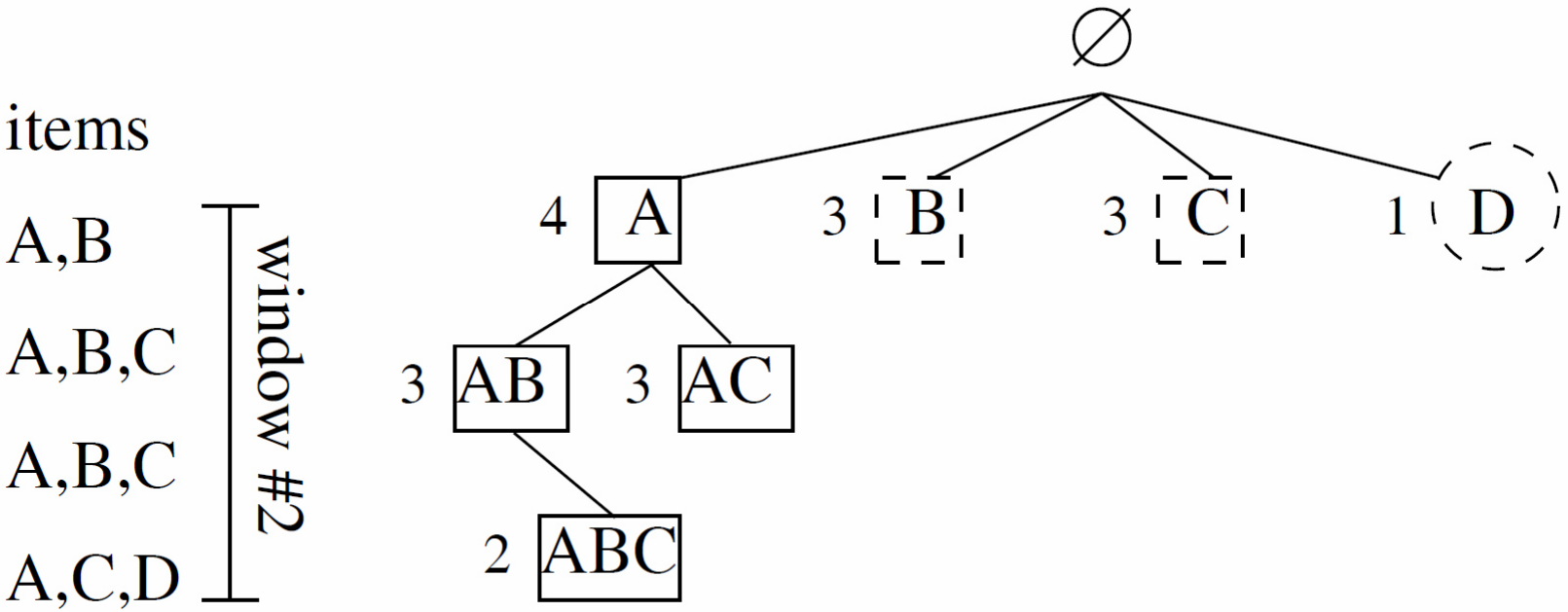


Adding a Transaction

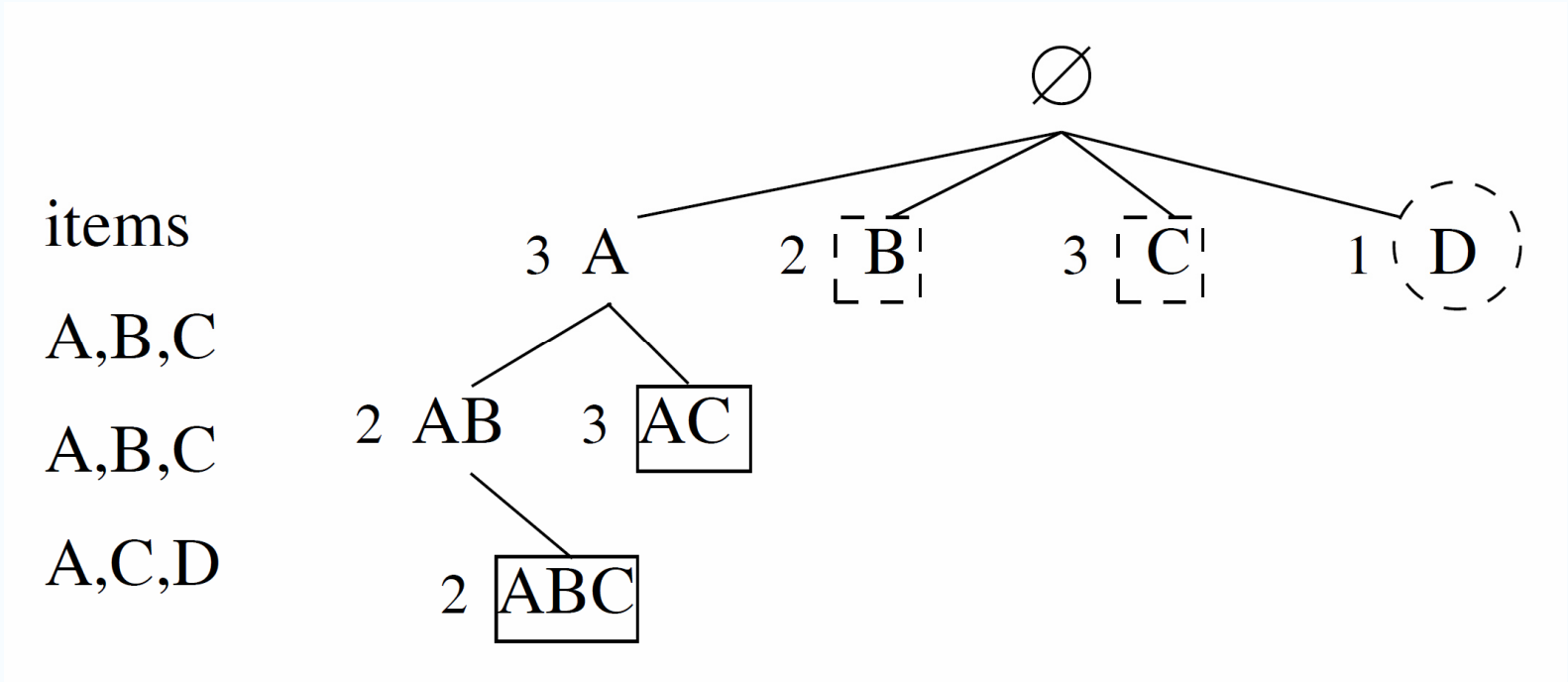
tid	items
1	C,D
2	A,B
3	A,B,C
4	A,B,C
5	A,C,D



Delecting a Transaction



Another Deletion



Moment Algorithm

Increments:

- Add/Delete to/from CET upon arrival/expiration of each transaction.

Downside:

- Batch operations not applicable, suffers from big slide sizes

Advantage:

- Efficient for small slides

CanTree [Leung' 05]

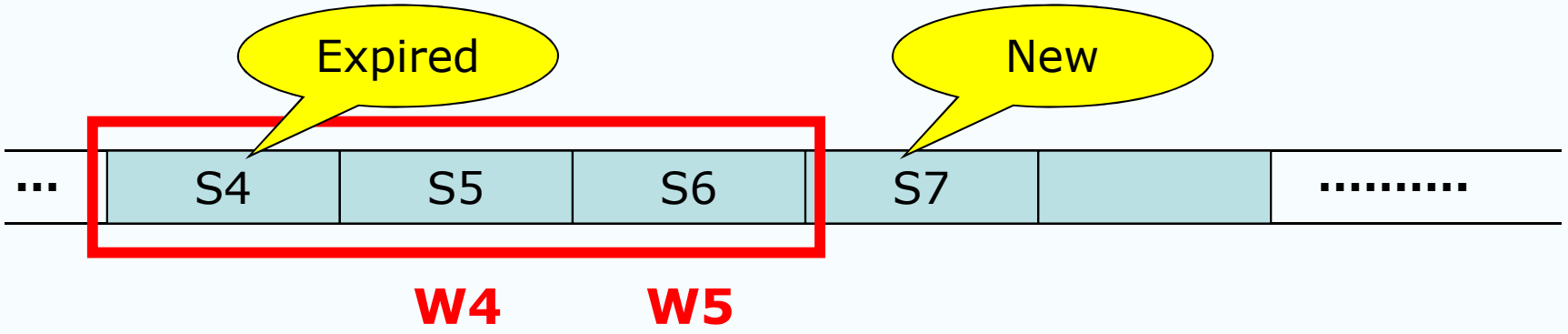
- Use a fixed canonical order according to decreasing single freq.
- Use a single-round version of FP-growth

Algorithm:

Upon each window move:

- Add/remove new/expired transaction to/from FP-tree (using the same item order)
- Run FP-growth! (Without any pruning)

Frequent Patterns Mining over Data Streams

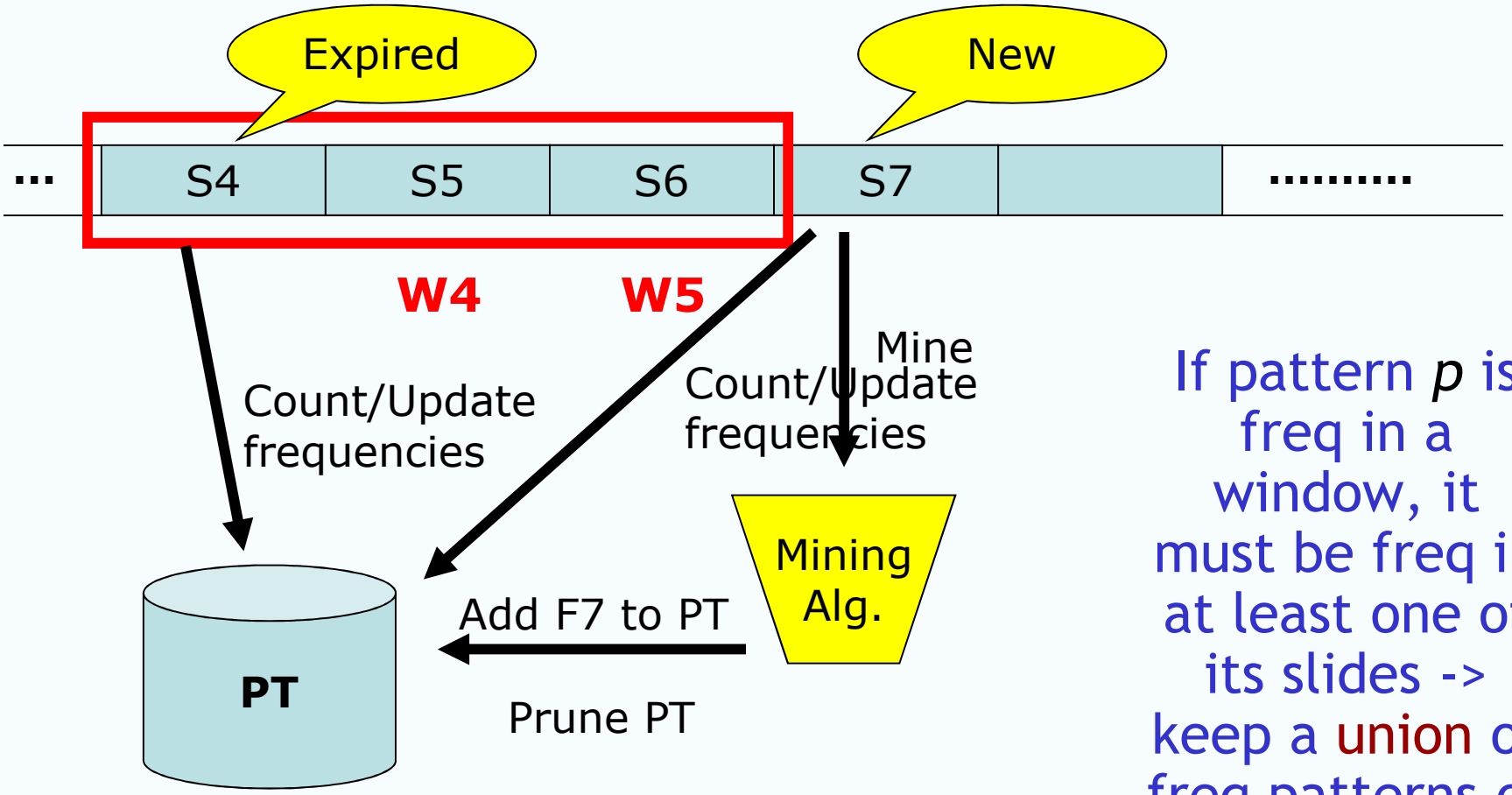


- Challenges
  - Computation
  - Storage
  - Real-time response
  - Integration with the Data Stream Management System (DSMS)

Frequent Patterns Mining over Data Streams

- Difficult problem: [Chi' 04, Leung' 05, …]
- Mining each window from scratch is too expensive
  - subsequent windows share many frequent patterns
- Updating frequent patterns every new tuple, also too expensive
- SWIM's middle-road approach: incrementally maintain frequent patterns over sliding windows
  - desiderata: scalability with *slide size* and *window size*

SWIM: Sliding Window Incremental Miner



If pattern *p* is freq in a window, it must be freq in at least one of its slides -> keep a union of freq patterns of all slides (PT)

SWIM

- For each new slide S_i
  - Find all frequent patterns in S_i (using FP-growth)
- Verify frequency of these new patterns in each window slide
  - Immediately or
  - With delay (< N slides)
  - Trade-off: max delay vs. computation.
- No false negatives or false positives!

SWIM – Design Choices

- Data Structure for S_i's: FP-tree [Han' 00]
- Data Structure for PT: FP-tree
- Mining Algorithm: FP-growth
- Count/update frequencies: naïve? Hash-tree?
  - Counting is the bottleneck
  - Improved counting method called Conditional Counting