High-Performance Complex Event Processing over Streams

Eugene Wu
UC Berkeley

Yanlei Diao
UMass Amherst

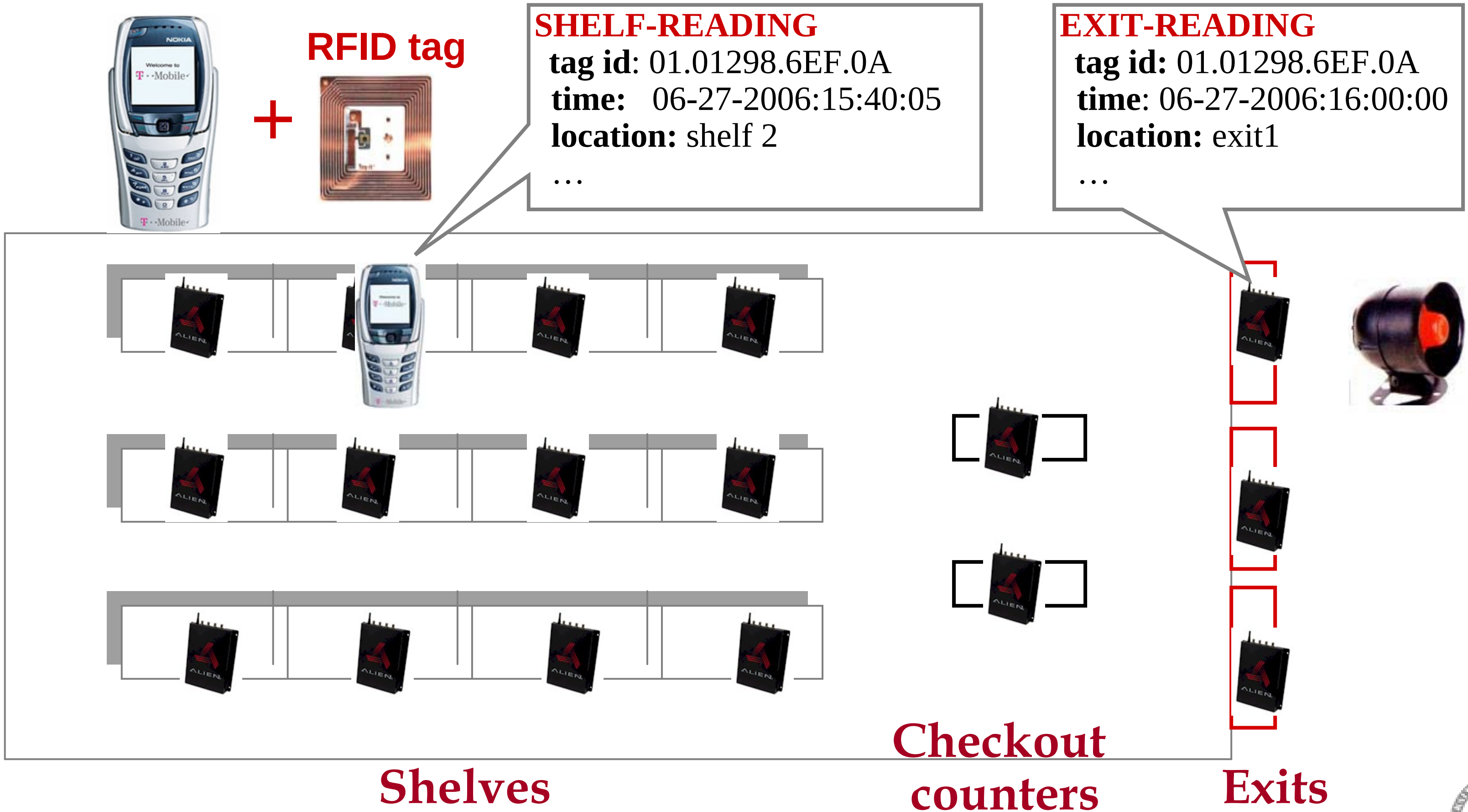
Shariq Rizvi
Google Inc.

Complex Event Processing

- Sensor and RFID (*Radio Frequency Identification*) technologies are gaining mainstream adoption
- Emerging applications: *retail management, food & drug distribution, healthcare, library, postal services…*
- High volume of events with complex processing
  - filtered
  - correlated for complex pattern detection
  - transformed to reach an appropriate semantic level
- A new class of queries
  - translate data of a physical world to useful information

A Retail Management Scenario

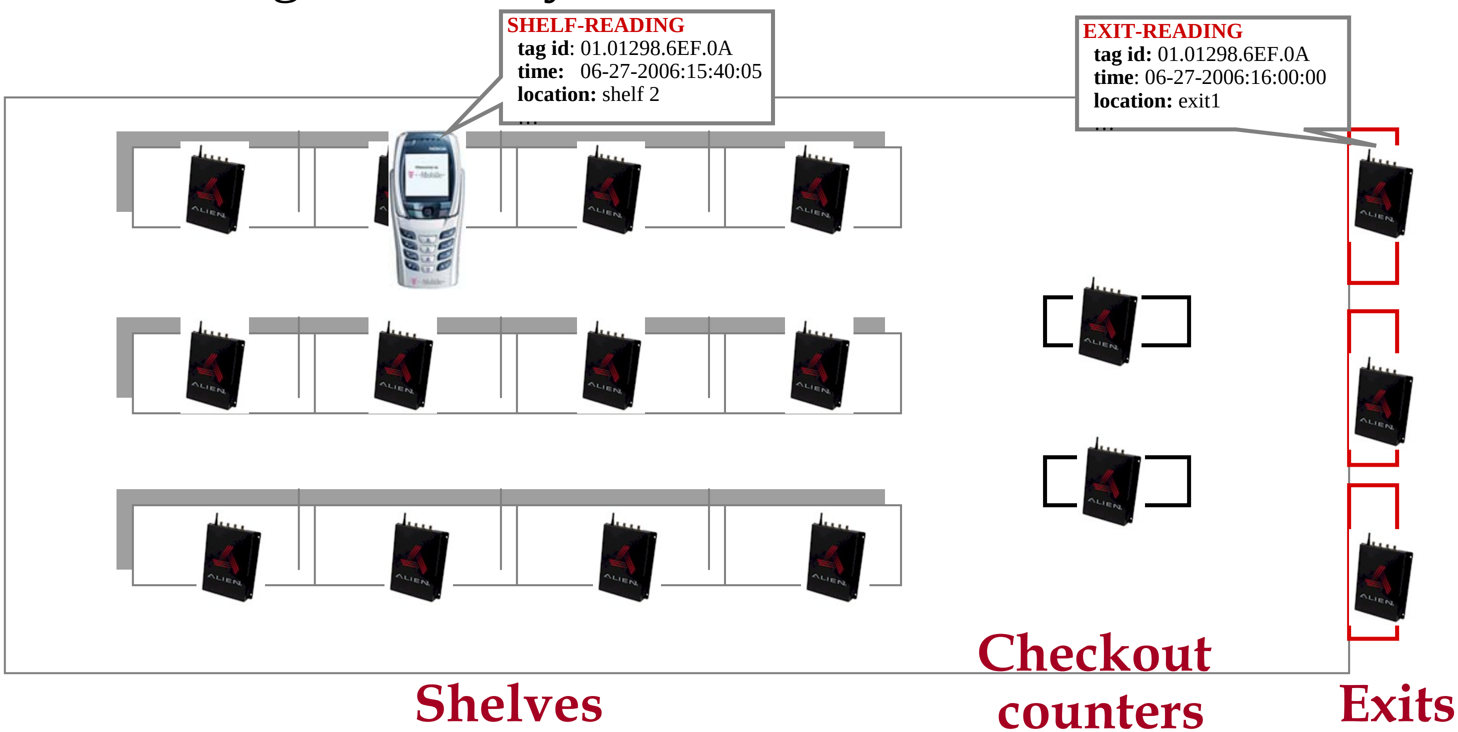
Shoplifting: an item was first read at a shelf and then at an exit but not at any checkout counter in between.



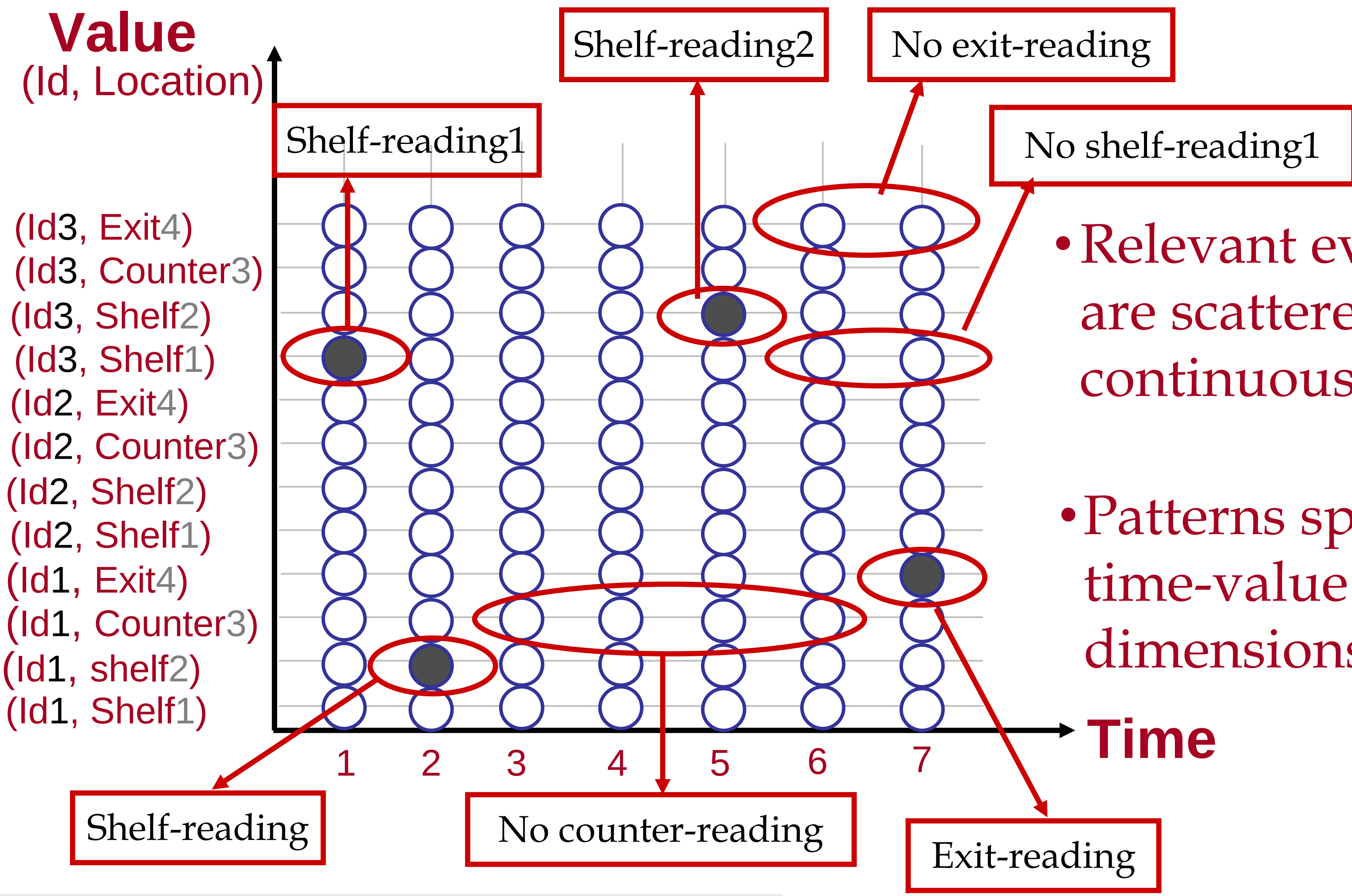
A Retail Management Scenario

Shoplifting: an item was first read at a shelf and then at an exit but not at any checkout counter in between.

Misplaced inventory: an item was first read at shelf 1, then at shelf 2, without being read at any checkout counter or back at shelf 1 afterwards.



Semantic Complexity



- Relevant events are scattered, not continuous
- Patterns span time-value dimensions

Performance Requirements

- Low-latency
  - Up-to-the-second information
  - Time-critical actions
- Scalability
  - High-volume event streams
  - Large monitoring windows

SASE: Complex Event Language

Language structure

EVENT <event pattern> :structure of an event pattern

[**WHERE** <qualification>] :value-based predicates over the pattern

[**WITHIN** <sliding window>] :sliding window over the pattern

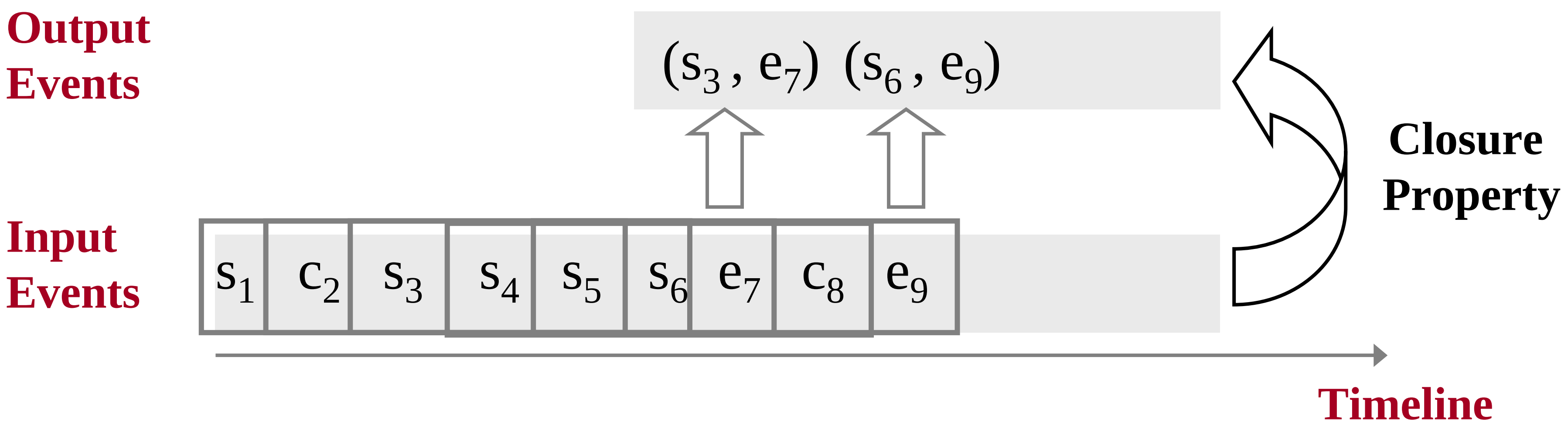
SASE: Complex Event Language

Shoplifting Query

EVENT **SEQ**(*SHELF-READING s*, **!** (*COUNTER-READING c*), *EXIT-READING e*)

WHERE *s.tag_id=c.tag_id* ∧ *s.tag_id=e.tag_id* /* equivalence test [*tag_id*] */

WITHIN 12 hours



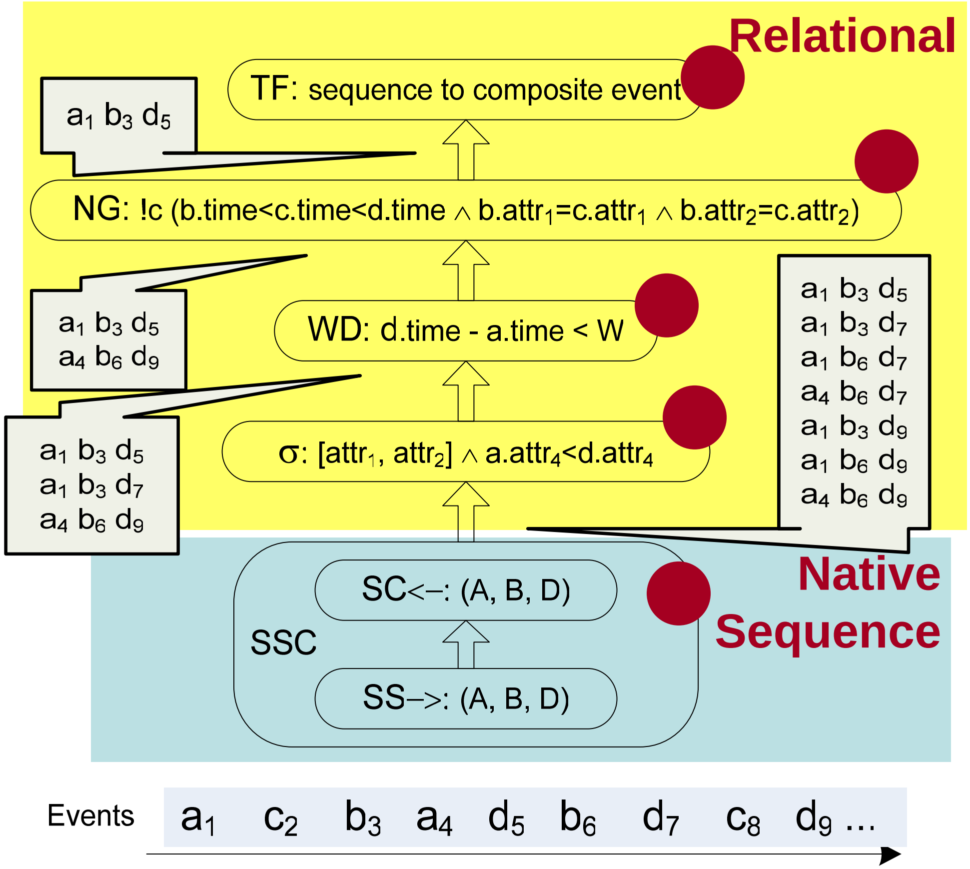
Abstraction of Complex Event Processing

- How can the language be efficiently implemented?
- Query plan-based approach
  - Dataflow paradigm with pipelined operators: flexible, optimizable, extensible
  - Existing event systems use fixed data structures
- New abstraction for complex event processing
  - Native sequence operators, pipelining query-defined sequences to subsequent relational style operators
  - Existing stream systems use relational joins

A Basic Query Plan

EVENT SEQ(*A a*, *B b*, **!**(*C c*), *D d*)
WHERE [*attr*1, *attr*2] ∧
*a.attr*4 < *d.attr*4
WITHIN W

- **Transformation** (TF)
- **Negation** (NG)
- **Window** (WD)
- **Selection** (σ)
- **Sequence scan & construction** (SSC)



Sequence Scan & Construction

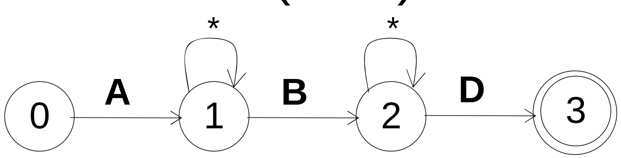
- Finite Automata are a natural formalism for sequences
- Two phases of processing
  - Sequence Scan (SS→): scans input stream to detect matches
  - Sequence Construction (SC←): searches backward (in a summary of the stream) to create event sequences.
  - Some techniques adapted from YFilter [Diao et al. 2003]

Illustration of SSC

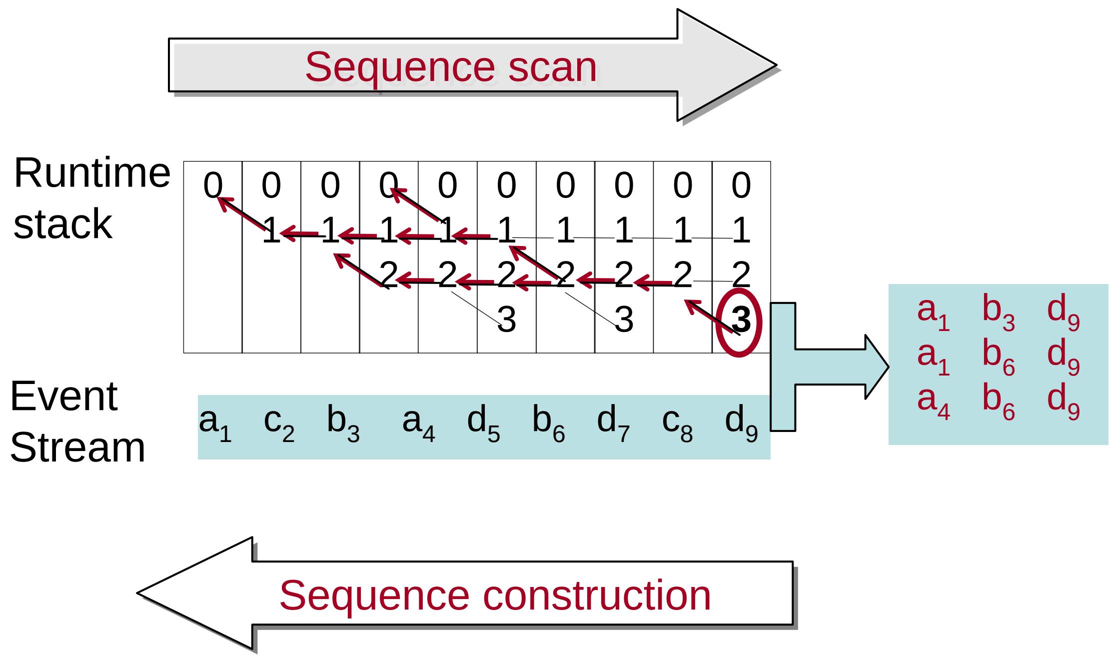
Sub-sequence type

(A, B, D)

Nondeterministic Finite Automaton (NFA)



O(SeqLen * Window)

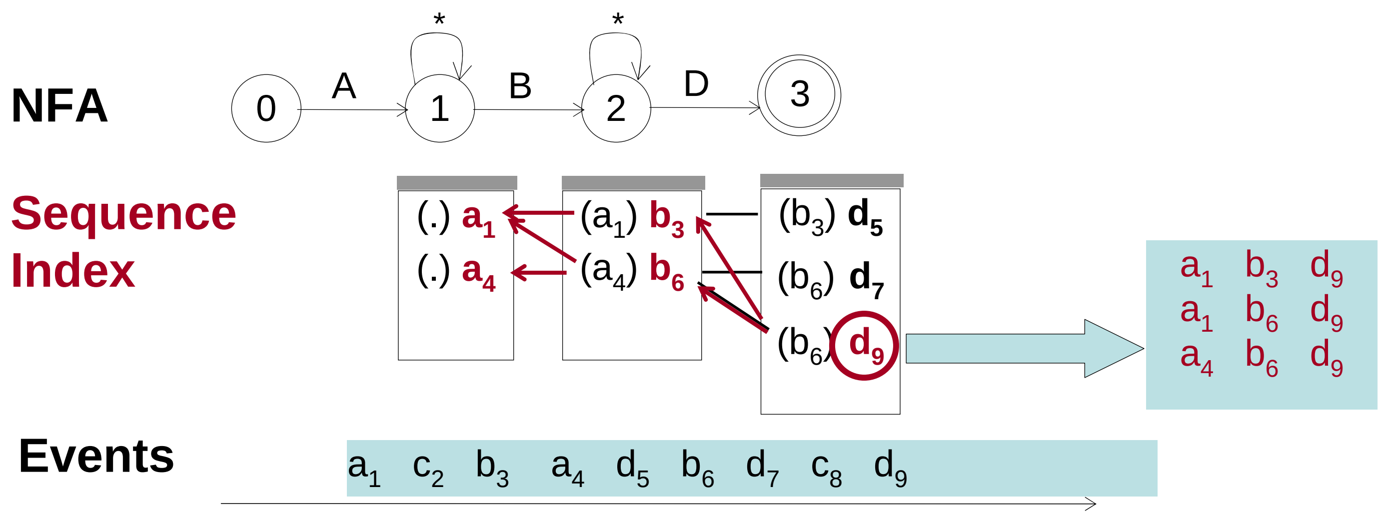


Optimization Issues

- What are the key issues for optimization?
  - Large sliding windows: e.g., "within past 12 hours"
  - Large intermediate result sizes: may cause wasteful work
- Intra-operator optimization to expedite SSC
  - Cost of sequence construction depends on the window size.
- Inter-operator optimizations to reduce intermediate results
  - How to evaluate predicates early in SSC?
  - How to evaluate windows early in SSC?
- Indexing relevant events in SSC both in temporal order and across value-based partitions

Optimizing SSC

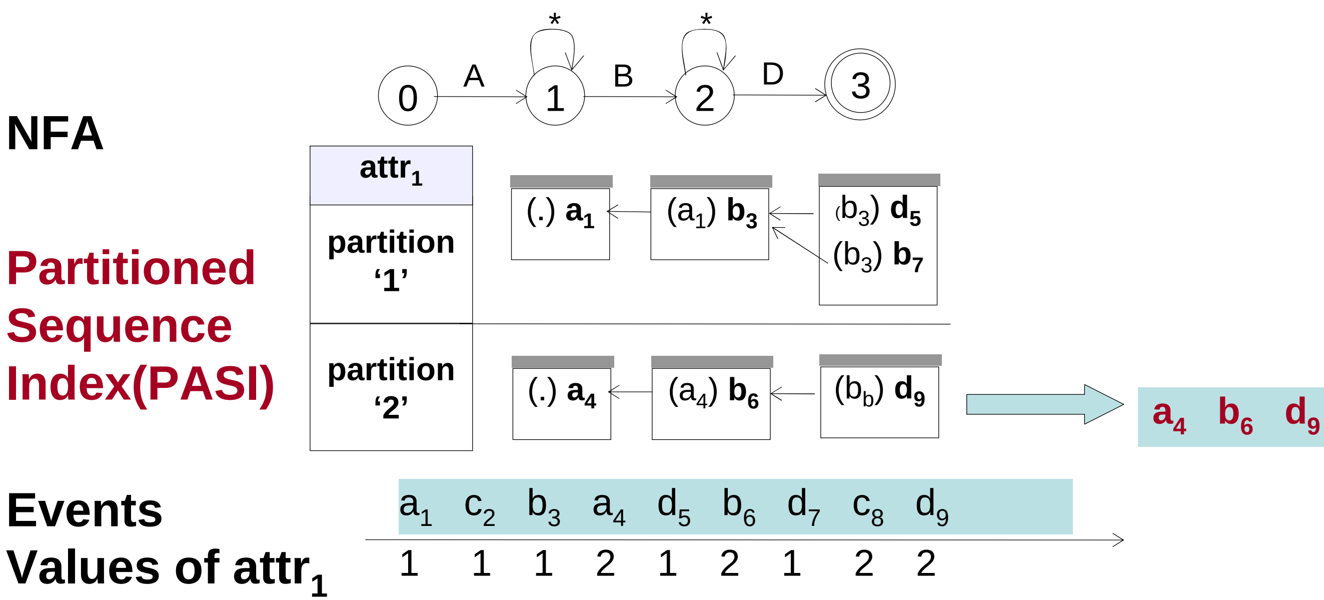
❖ "Sequence index" integrated with the NFA model



- SS→ builds the index in NFA execution
- SC← searches the sequence index for event sequences

Pushing An Equivalence Test To SSC

- Equivalence test: equality across all events in a sequence
- "Partitioned sequence index": sequence + value



- SS→ is extended with *transition filtering* & *stack maintenance*
- SC← searches only in a partitioned sequence index

Other Inter-Operator Optimizations

- Evaluating additional equivalence tests in SSC
  - Multi-attribute partitions: high memory overhead
  - Single-attribute partitions & cross filtering in SS$\rightarrow$
  - Dynamic filtering in SC$\leftarrow$
- Evaluating windows in SSC…
  - Windows in SS$\rightarrow$: coarse grained filtering, pruning
  - Windows in SC$\leftarrow$: precise checking

Performance Evaluation (1)

- Effectiveness of query processing in SASE
  - Sequence index offers an order-of-magnitude improvement with large windows & query result sizes.
  - Partitioned sequence index is highly effective. Pushing one equivalence test to SSC is a must!
  - Dynamic Filtering in SC← is memory economical and best performing for additional equivalence tests.
  - Pushing windows down…
  - Cost of negation…

Performance Evaluation (2)

❖ Comparison to a stream system using joins

SASE:

EVENT SEQ(E_1, E_2, …, E_L)
WHERE [$attr_1$ (, $attr_2$)?]
WITHIN W

Parameters:

L – Sequence length

W – Window size in # events

V_1 – domain size of $attr_1$

V_2 – domain size of $attr_2$

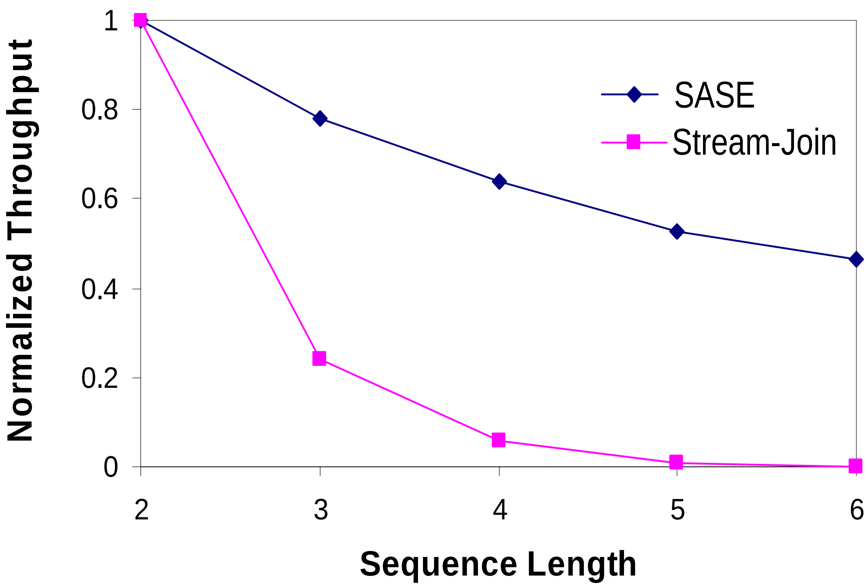
Join-based Stream Processor: L=3, W=10000, [$attr_1$]

```
With
R As   (Select * From ES e Where e.type = 'E1')
S As   (Select * From ES e Where e.type = 'E2')
T As   (Select * From ES e Where e.type = 'E3')
( Select   *
  From     R r        [range by 10000]
           S s        [range by 10000]
           T t        [range by 10000]
  Where   r.attr1 = s.attr1 and r.attr1 = t.attr1 and
          s.time > r.time and t.time > s.time )
```

- Offered hint on the most selective predicate to the stream optimizer
- Performance metric is throughput

Varying Sequence Length

EVENT SEQ($E_1, E_2, \ldots, E_L$)
WHERE [$attr_1$]
WITHIN W
Parameters:
L = 2-6
W = 10,000
[$Attr_1$] V_1 = 100

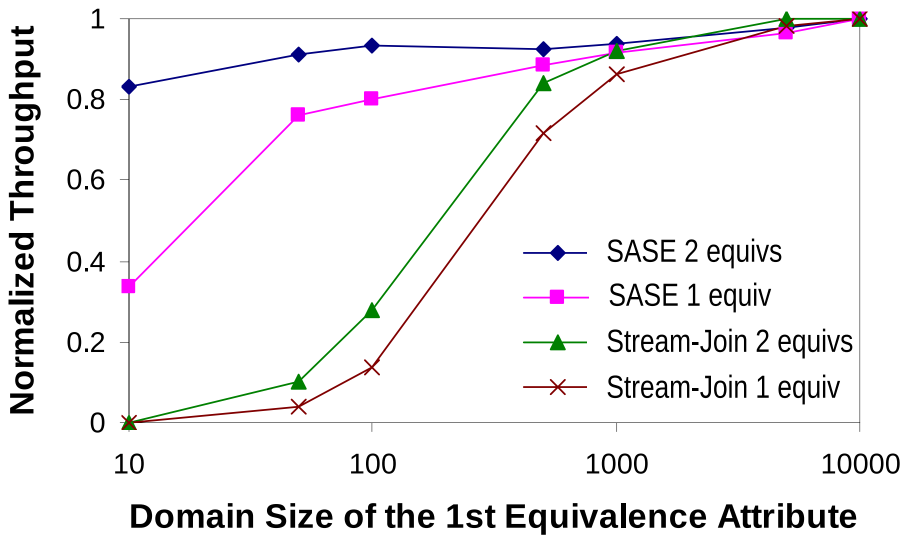


SASE scales better than Stream-Join for longer sequences.

- Stream Join: N-way joins, postponed temporal predicates
- SASE: NFA for sequences, value index for predicates, both in SSC

Varying Selectivity of Predicates

EVENT SEQ(E_1, E_2, …, E_L)
WHERE [$attr_1$ (, $attr_2$)?]
WITHIN W
Parameters:
L = 3
W = 10,000
[$Attr_1$] V_1 = 10 – 10,000
[$Attr_2$] V_2 = 20



SASE produces fewer intermediate results than Stream-Join.

- Stream-Join: cascading joins, postponed temporal predicates
- SASE: both sequencing and predicates in SSC, before producing any intermediate results

Conclusions

- Compact, expressive complex event language
  - Sequence, negation, predicates, sliding windows
- Query processing approach with a new abstraction
  - Native sequence operators + subsequent relational-style operators
- Optimization Techniques
  - Handling large slide windows
  - Reducing intermediate result sizes
- Summary of results
  - Relational stream systems not suited for complex event processing
  - Native sequence operators + optimized plans efficient and scalable
  - Our event processing technology can be integrated into stream systems