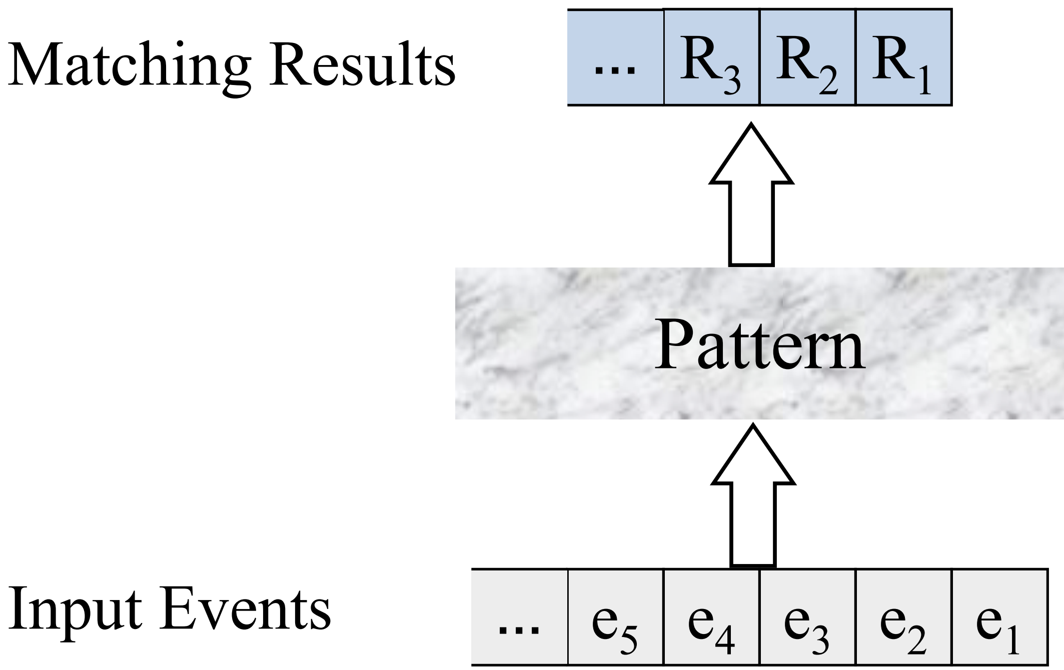
Efficient Pattern Matching over Event Streams

Jagrati Agrawal
Yanlei Diao
Daniel Gyllstrom
Neil Immerman

University of Massachusetts, Amherst

A New Stream Processing Paradigm

- Pattern matching over event streams



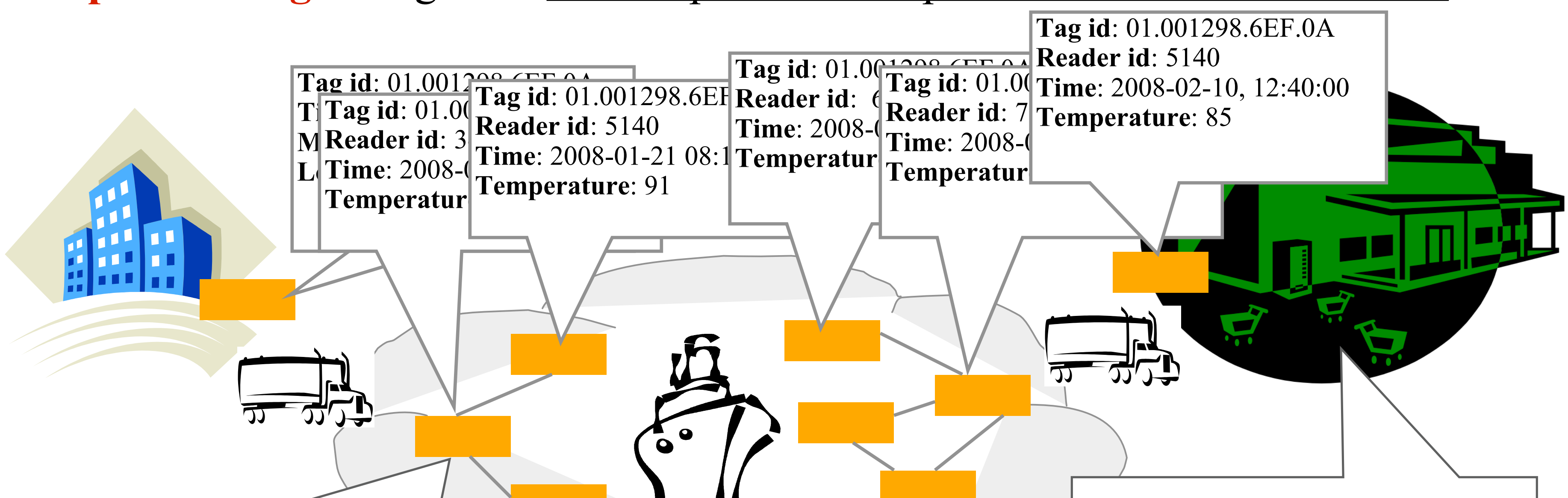
Applications

- Existing and emerging applications
  - Financial services
  - RFID-based supply chain management
  - Click stream analysis
  - Electronic health systems
  - Network monitoring
  - E-commerce purchase tracking
  - …

Supply Chain Management

Contaminated shipments: all shipments that were co-located with products originating from a source of contamination or subsequently infected products.

Spoiled drugs: drugs that were exposed to temperature > 100F for 24 hours.



Warehouse management: count pallets, detect missing pallets…

Retail management: shoplifting, misplaced inventory, fast shelf depletion…

Challenges

- Rich languages
  - *Sequencing*
  - *Kleene closure*
  - *Negation*
  - *Complex predicates*
  - *Event selection strategies…*

Significantly richer than regular languages!

- Efficient evaluation over streams
  - Relational stream systems: selection-join-aggregation
  - Recent event systems

Lacking support for key features. Not optimized.

Our Goal and Contributions

- A fundamental evaluation and optimization framework for the full set of event pattern queries
  - Formal evaluation model
    - Precise semantics of queries
    - Query evaluation plans
    - Formal results on expressive power
  - Runtime complexity analysis
  - Runtime algorithms and optimizations
  - Performance evaluation results

Event Pattern Languages

- Recent proposals: SQL-TS, Cayuga, SASE+, CEDR, StreamSQL, Coral8…
- Language structure of SASE+:

FROM <input stream>
PATTERN <pattern structure>
[**WHERE** <pattern matching condition>]
[**WITHIN** <time window>]
[**RETURN** <output specification>]

Q1: Stock Trend Monitoring

"In an hour, the volume of a stock sales record started high, but after a period of price increasing, the volume plummeted."

```
FROM InputStream
  PATTERN     SEQ(Stock+ a[], Stock b)
  WHERE
                [symbol] AND
                a[1].volume > 1000                    AND
                a[i].price > min(a[..i-1].price)     AND
                b.volume < 80% * a[a.LEN].volume

  WITHIN      1 hour
```

Q1 Using Partition Contiguity

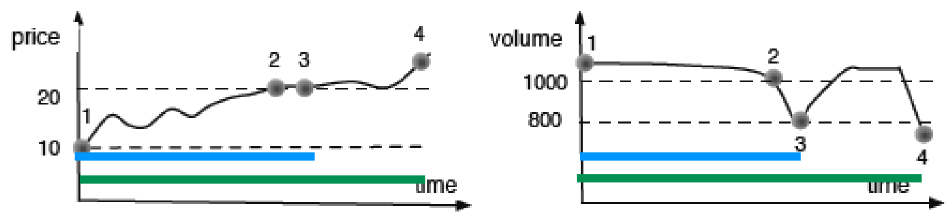
- Event Selection Strategy: <u>Partition Contiguity</u>
  - Captures a continuous trend in each partition

```
FROM InputStream
  PATTERN     SEQ(Stock+ a[], Stock b)
  WHERE       partition_contiguity(a[],b)
              { [symbol] AND
                a[1].volume > 1000            AND
                a[i].price > a[i-1].price     AND
                b.volume < 80% * a[a.LEN].volume
              }
  WITHIN       1 hour
```

Q1 Using Skip Till Next Match

❖ Event Selection Strategy: Skip Till Next Match

- Captures a broad trend while ignoring local fluctuating values



(a) change of stock price (b) change of stock volume

```
FROM InputStream
  PATTERN SEQ(Stock+ a[], Stock b)
  WHERE   skip_till_next_match(a[],b)
          { [symbol] AND
            a[1].volume > 1000        AND
            a[i].price > a[i-1].price  AND
            b.volume < 80%*a[a.LEN].volume }
   WITHIN 1 hour
```

Two overlapping matches

Q2: Contaminated Shipments

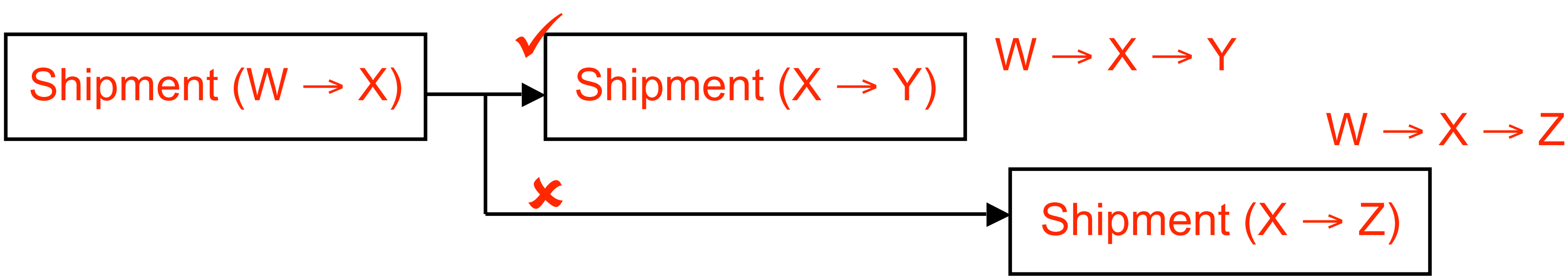
"In a food supply chain, detect contaminated shipments."

```
FROM InputStream
  PATTERN     SEQ(Alert a, Shipment+ b[])
  WHERE       skip_till_any_match(a, b[])
              { a.type = 'contaminated' AND
                b[1].from = a.site  AND
                b[i].from = b[i-1].to     }
  WITHIN      12 hours
  RETURN      a.type, a.site, b[].to
```

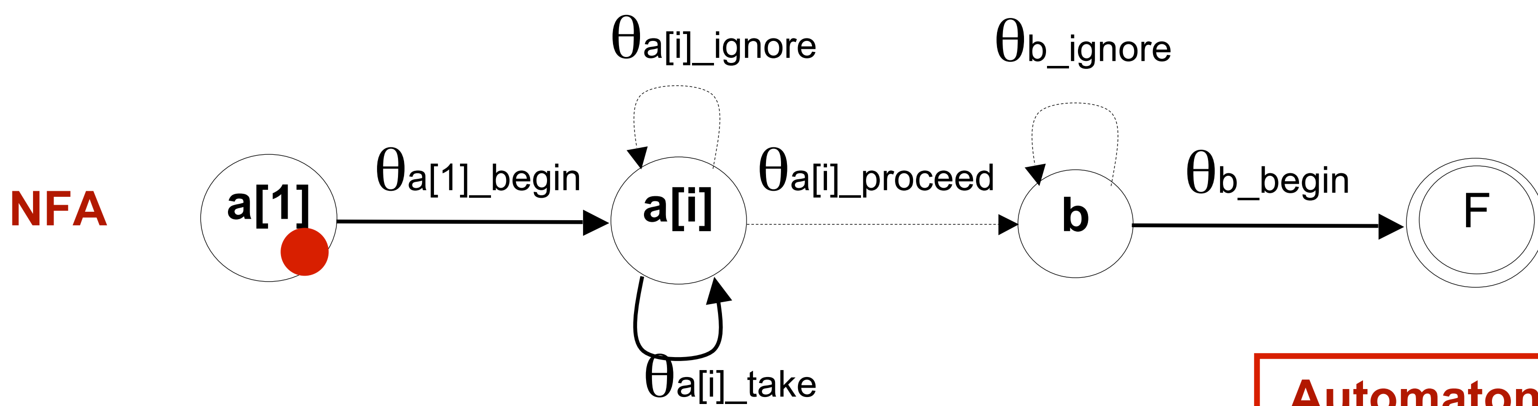
Q2 Using Skip Till Any Match

❖ Event Selection Strategy: <u>Skip Till Any Match</u>

- Can both select and skip relevant events, non-deterministic
- Computes transitive closure over an event stream



A Formal Evaluation Model: NFA^b



e1 **e2**

Match Buffer

a[1]	a[i]	b

Automaton State

a[1]

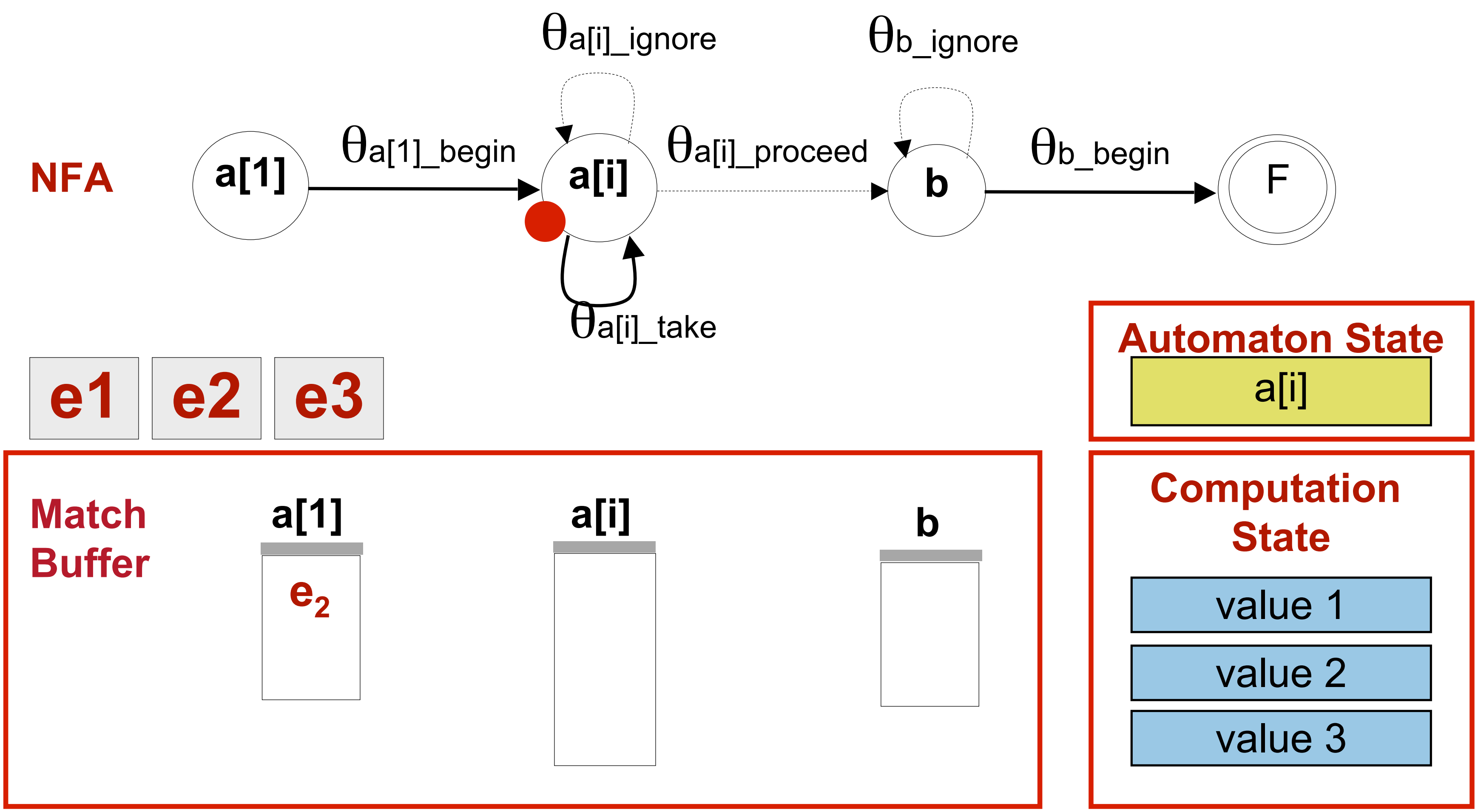
Computation State

value 1

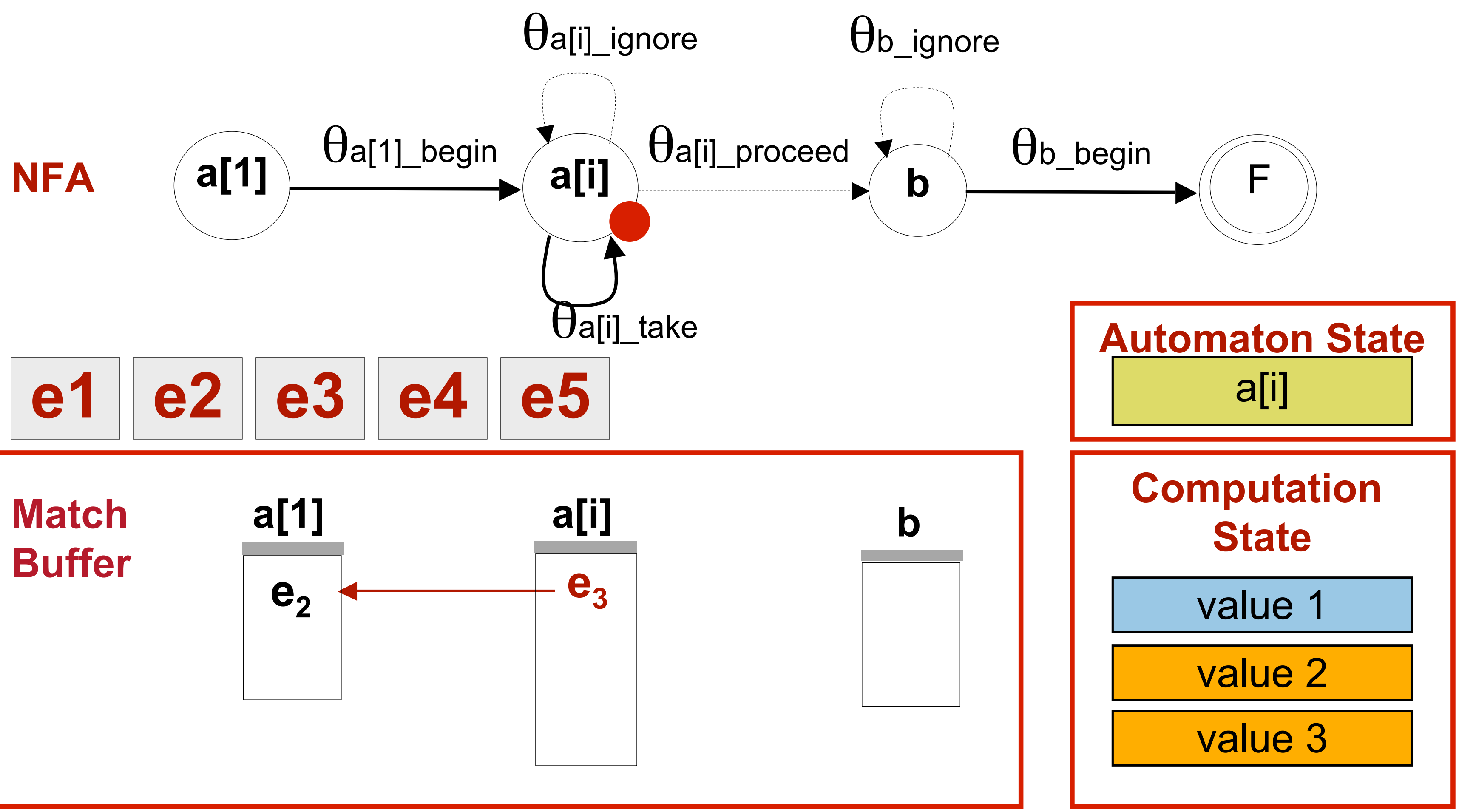
value 2

value 3

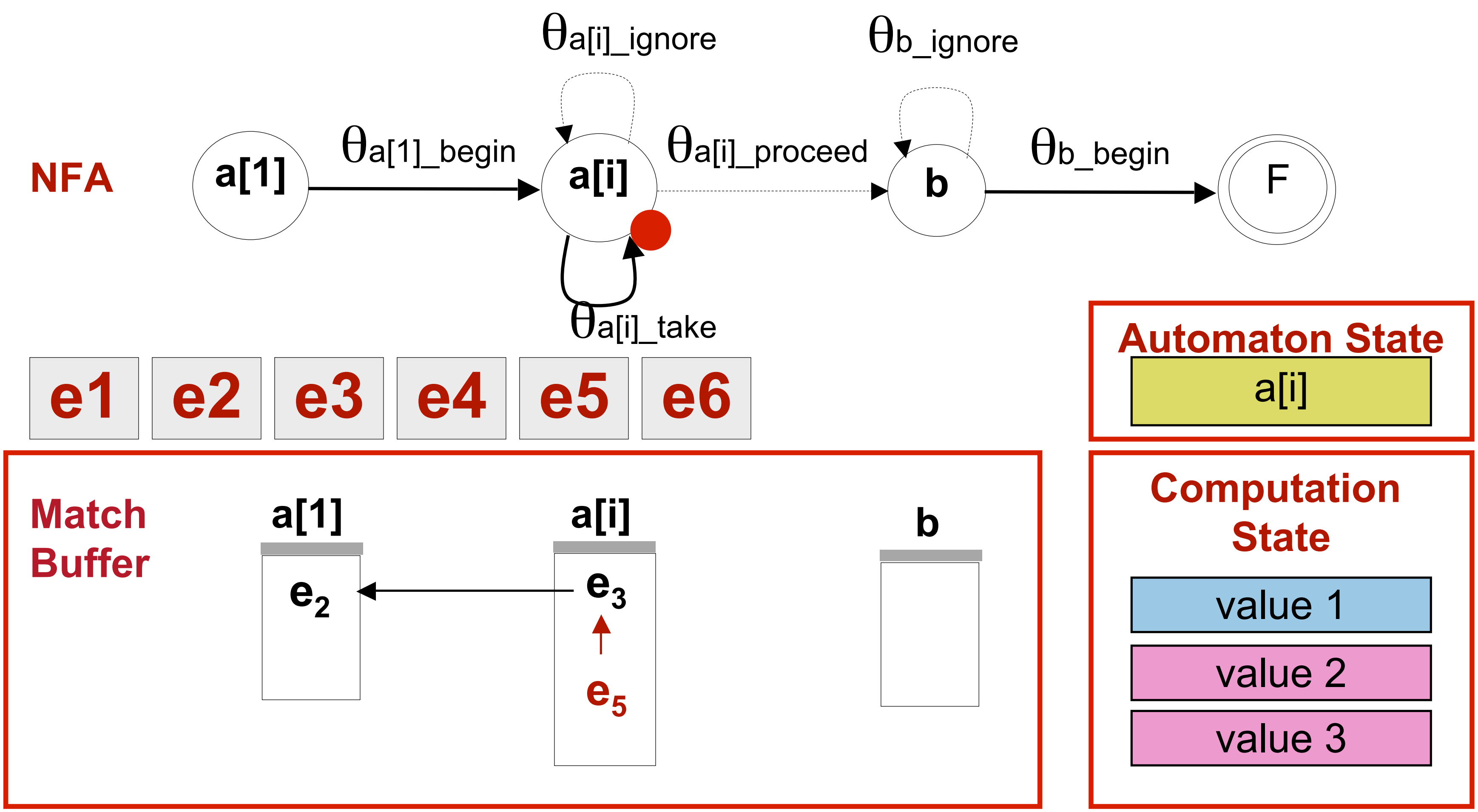
A Formal Evaluation Model: NFA^b



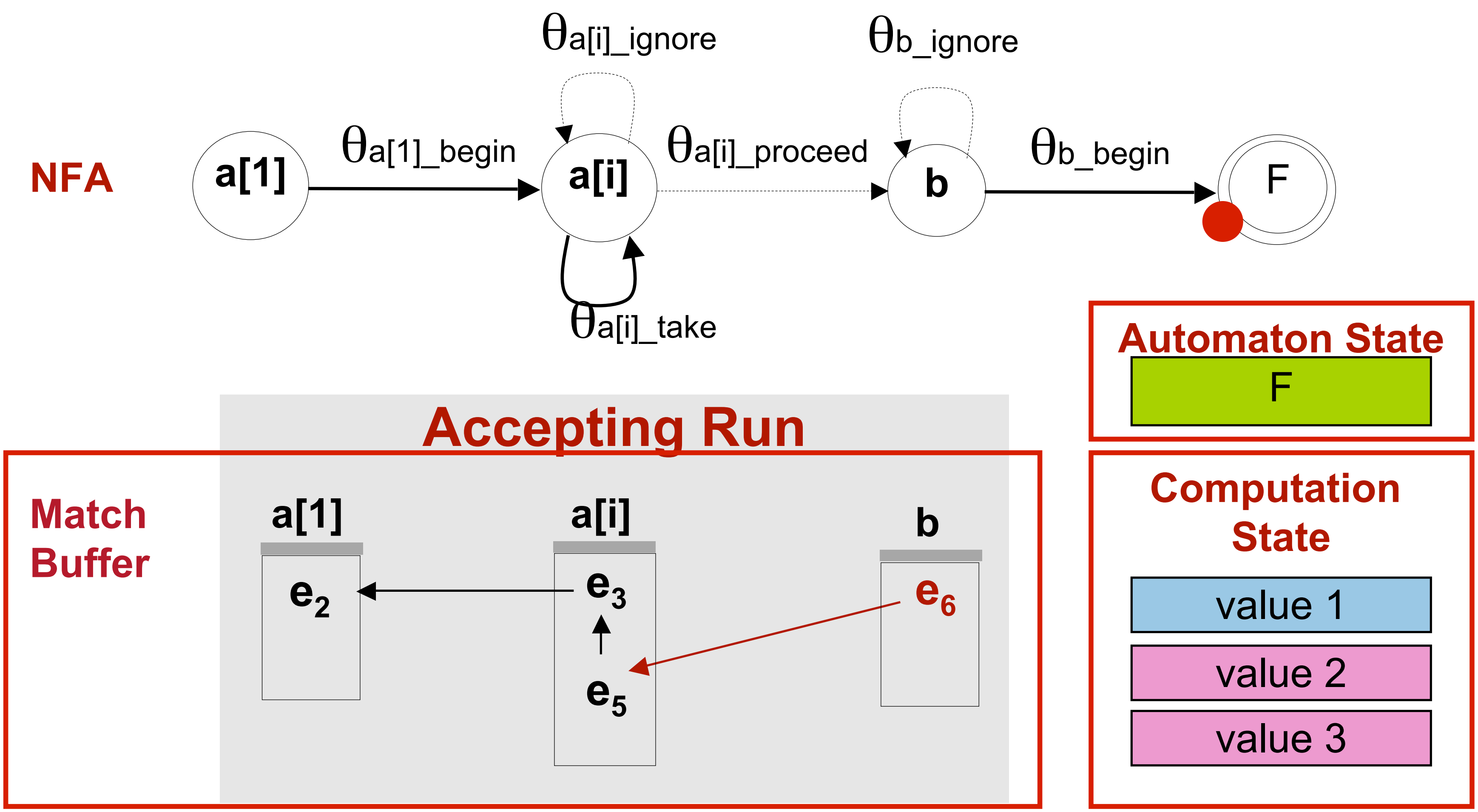
A Formal Evaluation Model: NFA^b



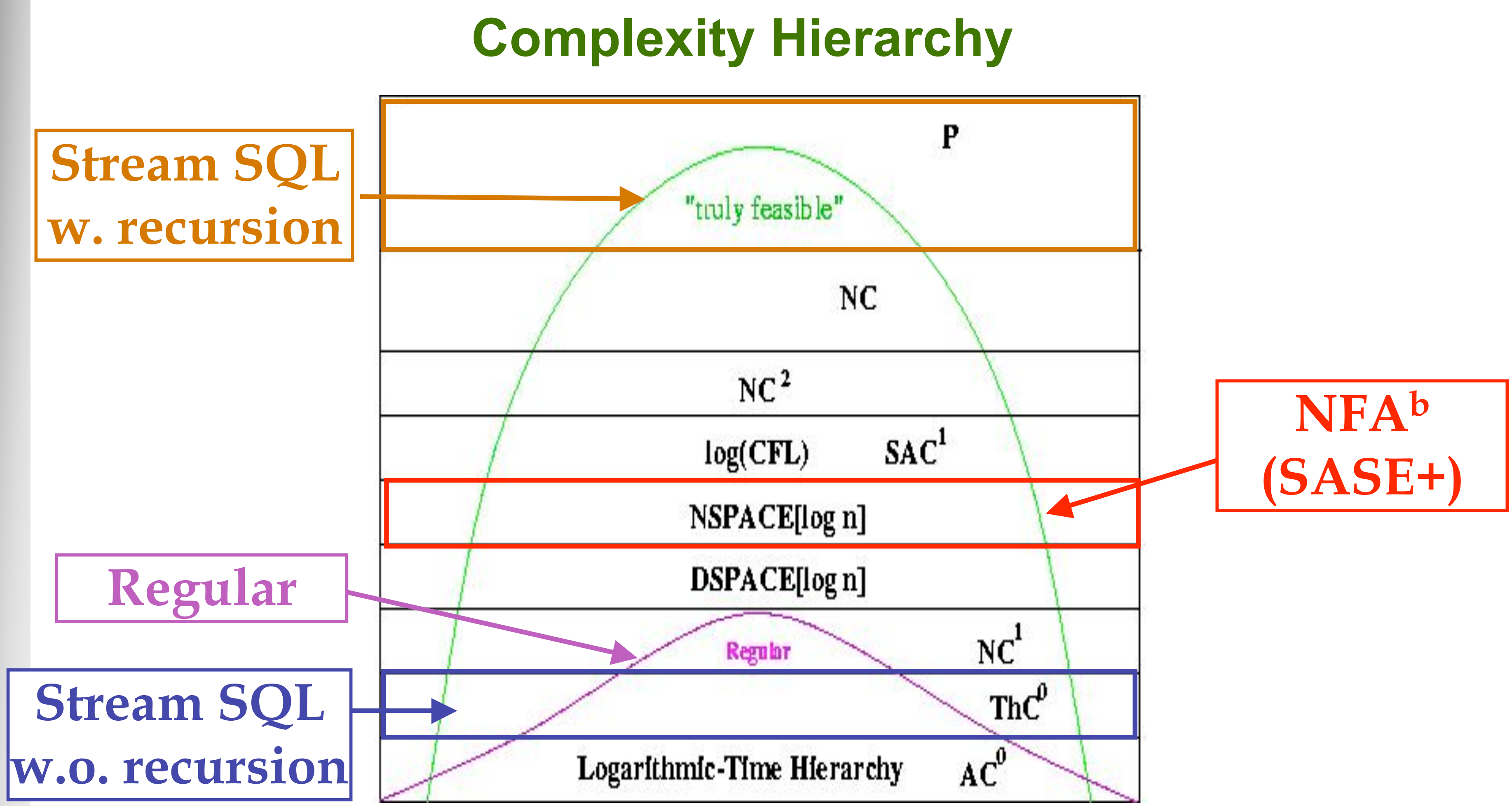
A Formal Evaluation Model: NFA^b



A Formal Evaluation Model: NFA^b

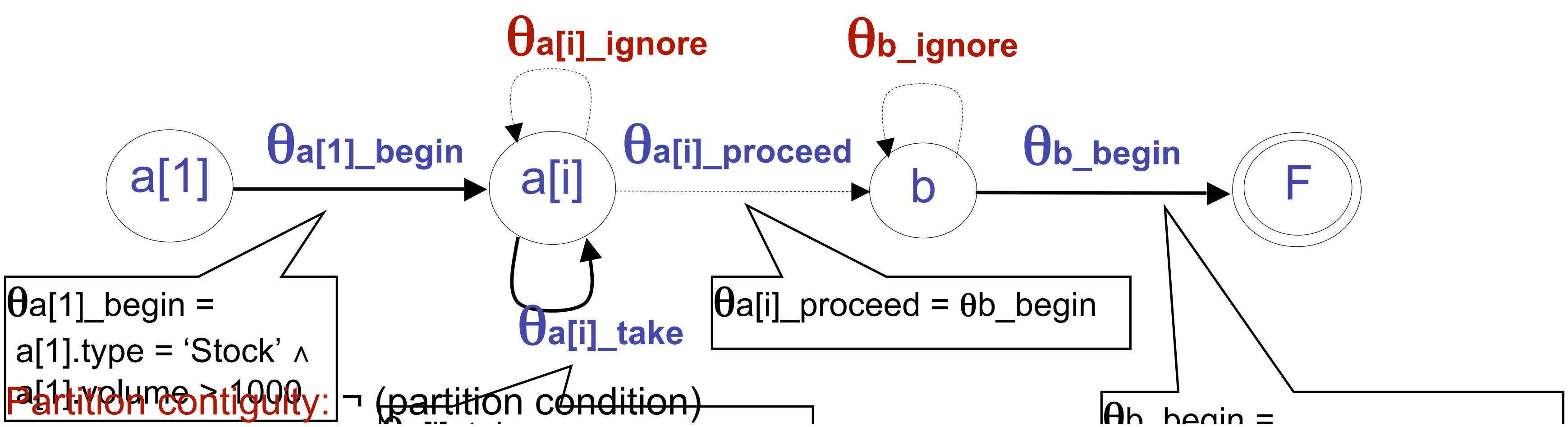


Expressibility of NFA^b



Query Compilation into NFA^b

PATTERN SEQ(Stock+ a[], Stock b)
WHERE Event_Selection_Strategy(a[], b) {
[symbol] AND
a[1].volume > 1000 AND
a[i].price > a[i-1].price AND
b.volume < 80% * a[a.LEN].volume }

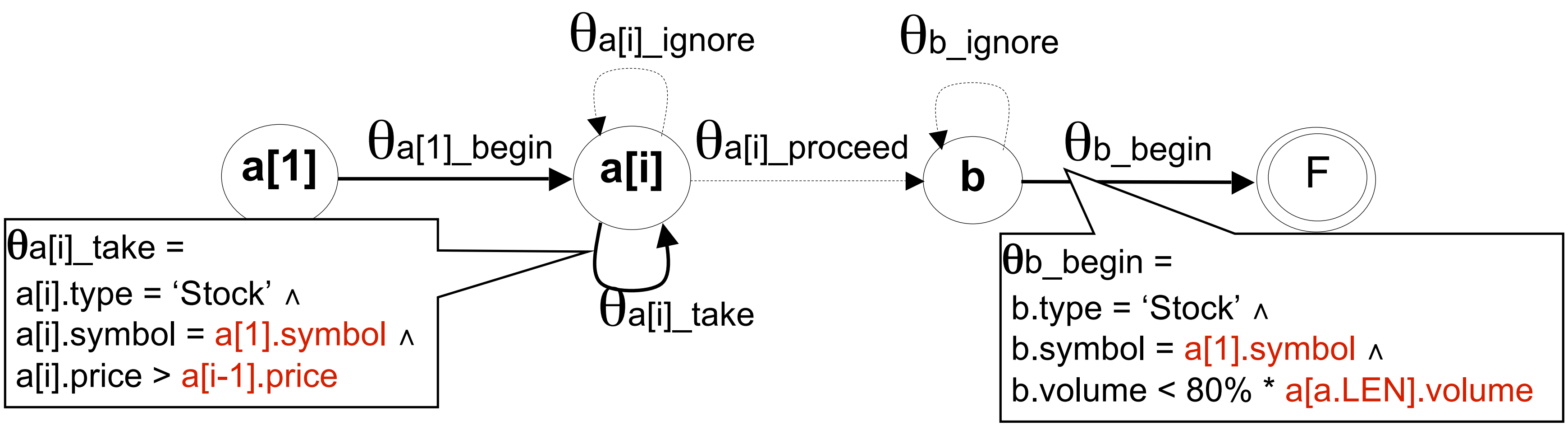


Partition contiguity: ¬ (partition condition)

Compile time optimizations: push filtering and stopping conditions early in appropriate places in the automaton

Computation State of NFA^b

Computation state: a minimum set of values for edge evaluation, extracted from *parameterized predicates*.

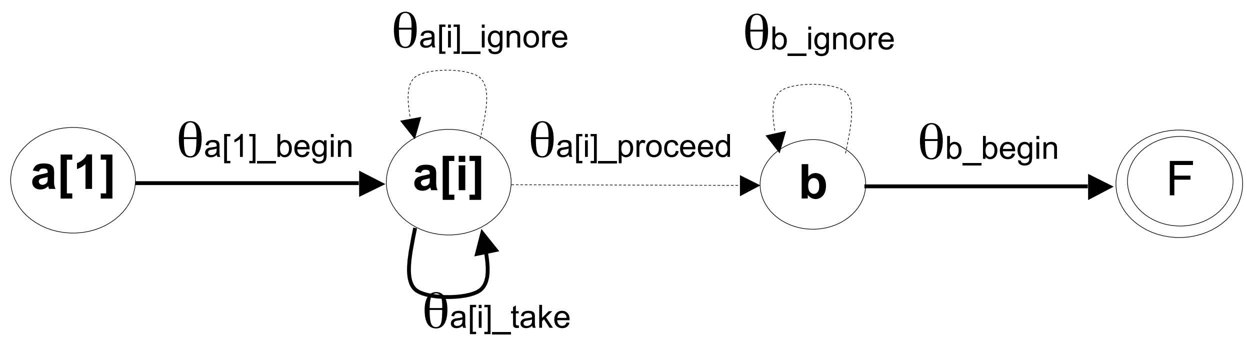


Computation State

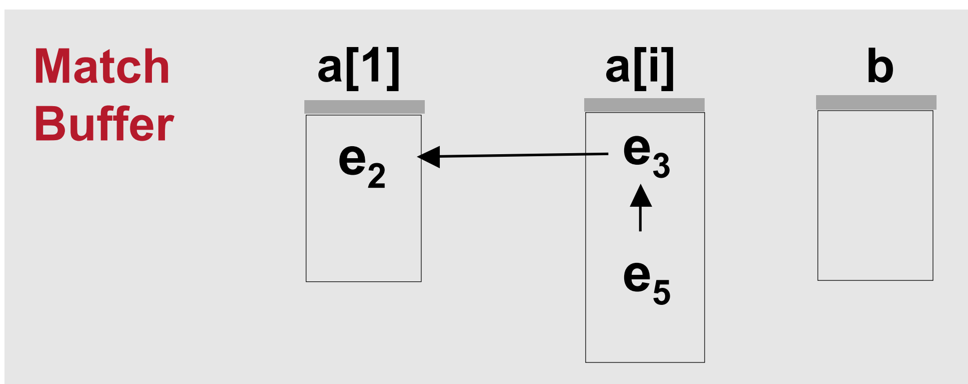
NFA^b state	Attribute	Operation
a[1]	symbol	set()
a[i]	price	setLast()
a[i]	volume	setLast()

Runtime Challenges

NFA



A Single Run of NFA^b



Automaton State

a[i]

Computation State

value 1

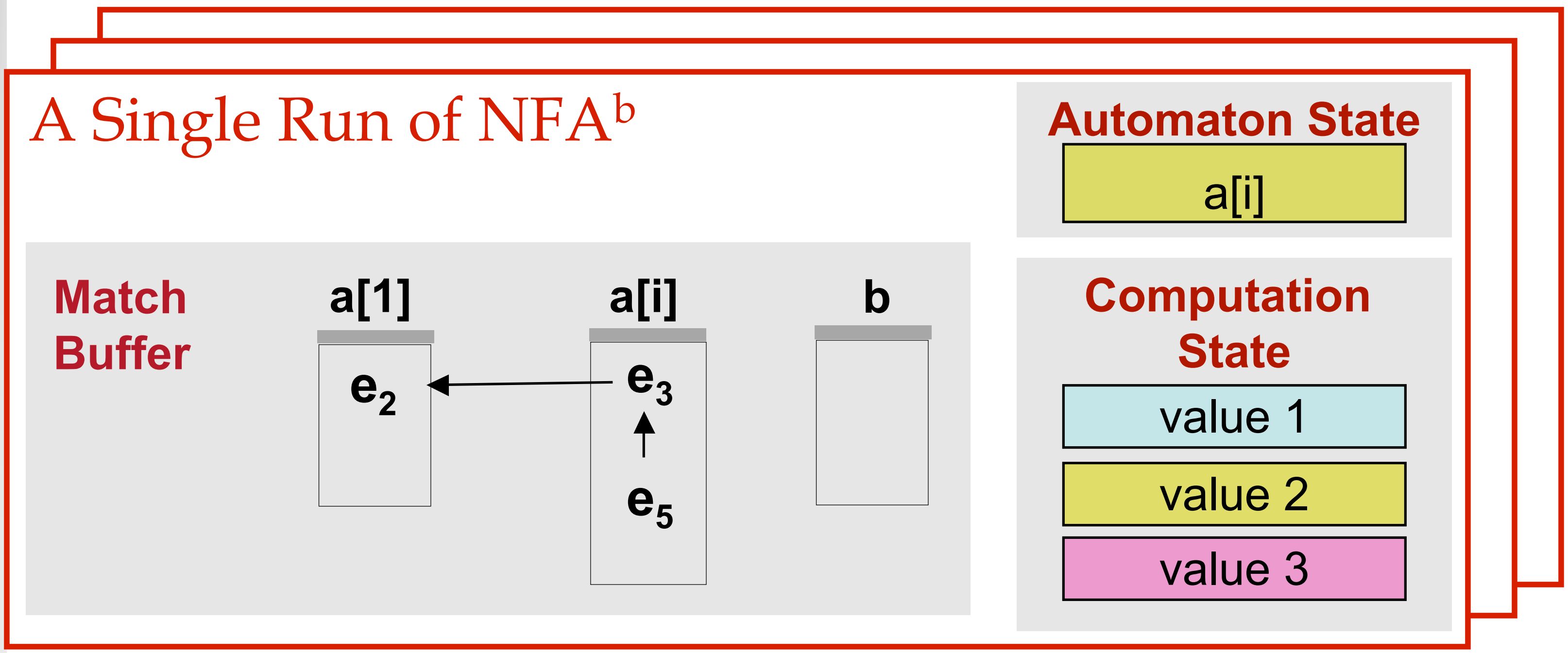
value 2

value 3

Runtime Challenges

Simultaneous runs of NFA^b:

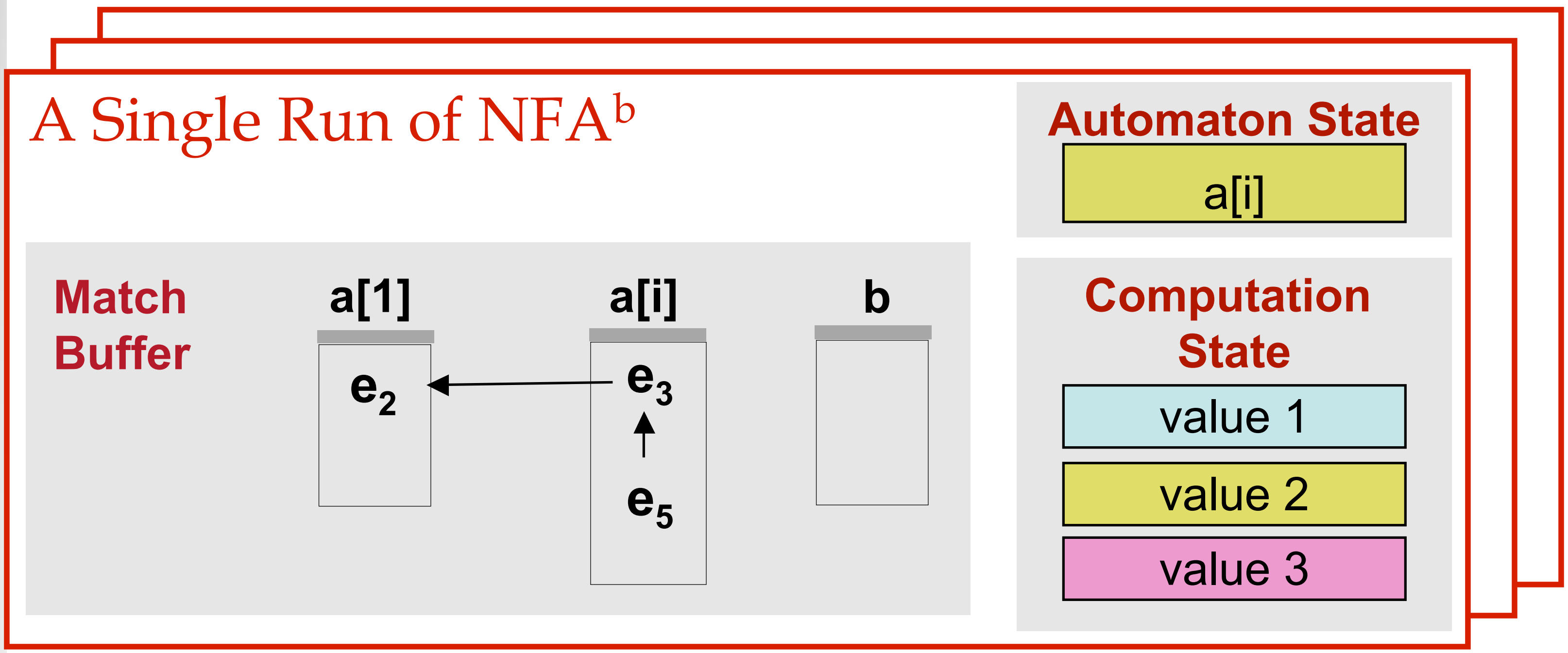
- A new run can start before an old run completes.
- A run can branch at an NFA^b state due to *nondeterminism*.



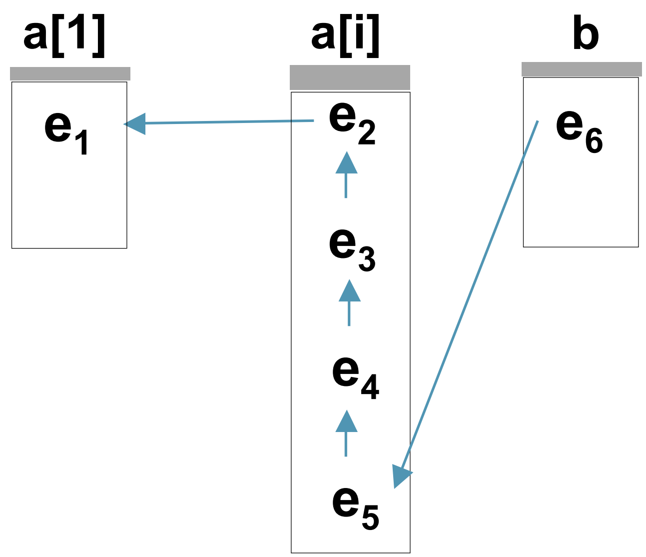
Runtime Complexity

How many runs can we have?

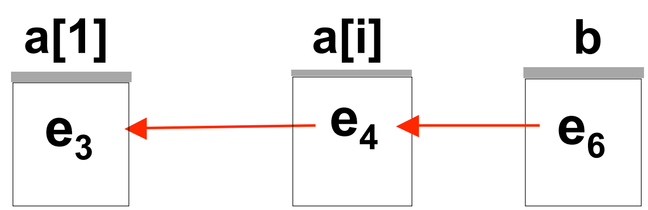
- Depends on *event selection strategy*, partition window size…
- Polynomial to exponential in the worse case.



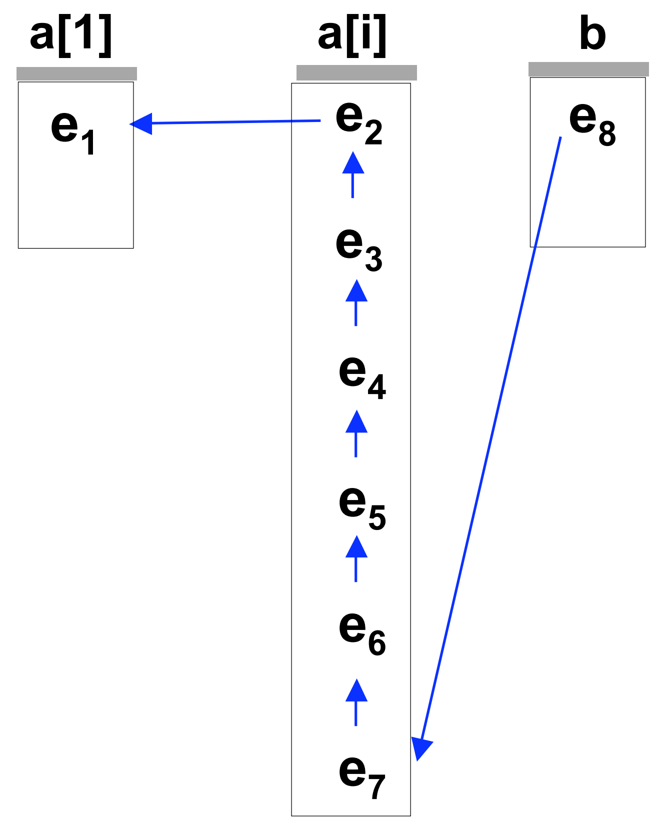
Merging Match Buffers



Match Buffer for Run 1

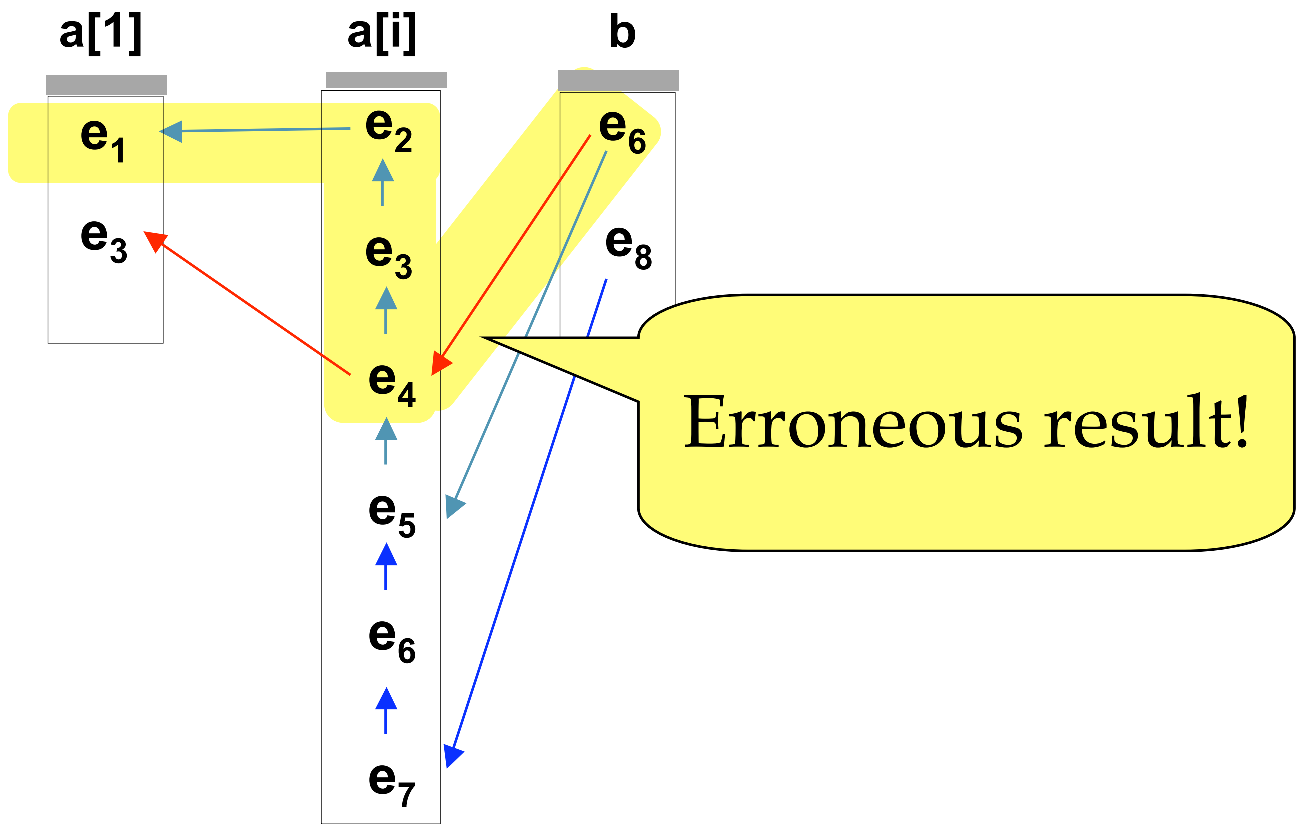


Match Buffer for Run 2



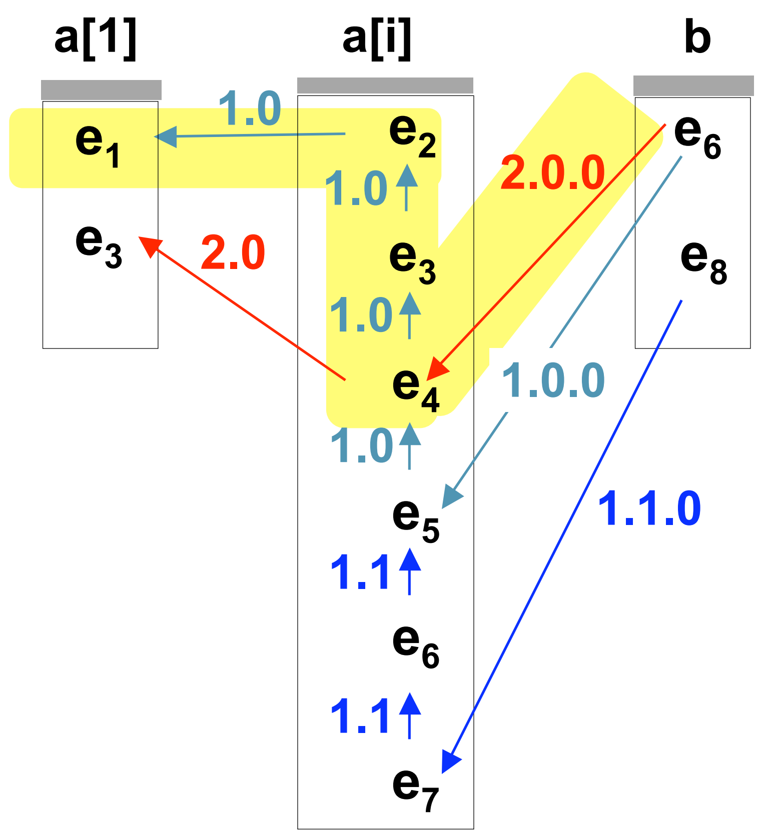
Match Buffer for Run 3

Merging Match Buffers



Shared Buffer for Runs 1, 2, 3

A Shared, Versioned Match Buffer



Shared, Versioned Buffer for Runs 1, 2, 3

Merging Equivalent Runs

- Equivalent runs
  - Despite distinct history, two runs have the same *computation state* at present.
  - They will select the same events till completion, hence can be merged.

Merging Equivalent Runs

PATTERN SEQ(Stock+ a[], Stock b)
WHERE skip_till_next_match(a[],b) {
[symbol] AND
a[1].volume > 1000 AND
a[i].price > a[i-1].price AND
b.volume < 80% * a[a.LEN].volume }

(symbol, price and volume of the recent selected event)

Computation State of Run 1

NFA^b state	attribute	value
a[1]	symbol	**XYZ**
a[i]	price	**last:121**
a[i]	volume	**last:1000**

Computation State of Run 2

NFA^b state	attribute	operation
a[1]	symbol	**XYZ**
a[i]	price	**last:121**
a[i]	volume	**last:1000**

Merging Equivalent Runs

```
PATTERN    SEQ(Stock+ a[], Stock b)
WHERE      skip_till_next_match(a[],b) {
              [symbol] AND
              a[1].volume > 1000                   AND
              a[i].price > min(a[..i-1].price)  AND
              b.volume < 80% * a[a.LEN].volume      }
```

(symbol, min price of all selected events, volume of last event)

Computation State of Run 1

NFA^b state	attribute	value
a[1]	symbol	**XYZ**
a[i]	price	**min:101**
a[i]	volume	**last:1000**

Computation State of Run 2

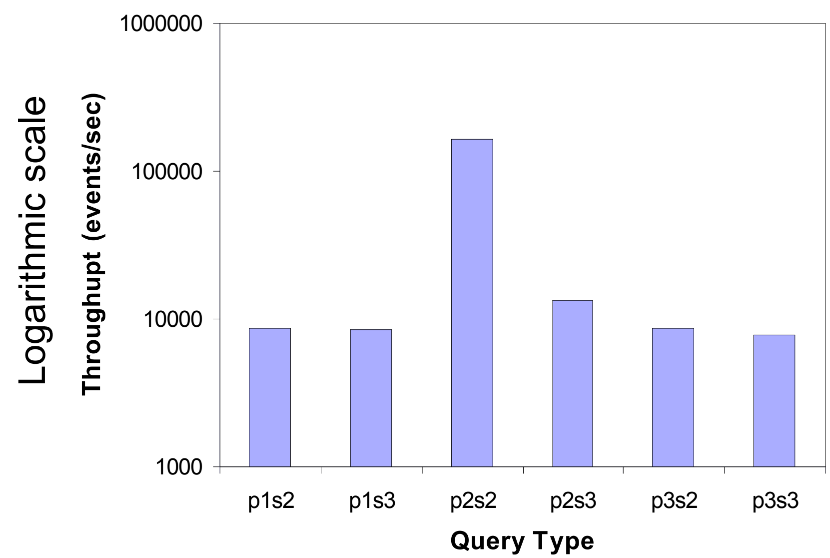
NFA^b state	attribute	operation
a[1]	symbol	**XYZ**
a[i]	price	**min:101**
a[i]	volume	**last:1000**

Performance of Kleene Closure

PATTERN SEQ(Stock+ a[ ], Stock b)
WHERE ***S***(a[ ], b) {
[*symbol*] AND
a[1].*price* % 500 = 0 AND
P(a[i]) AND
b.volume < 150 }
WITHIN ***W***

Parameters:
P = (p1) true
(p2) a[i].*price* > a[i-1].*price*
(p3) a[i].*price* > aggr(a[..i-1].*price*)
aggr = min | max | avg
S = (s2) partition contiguity
(s3) skip till next match
W = 500

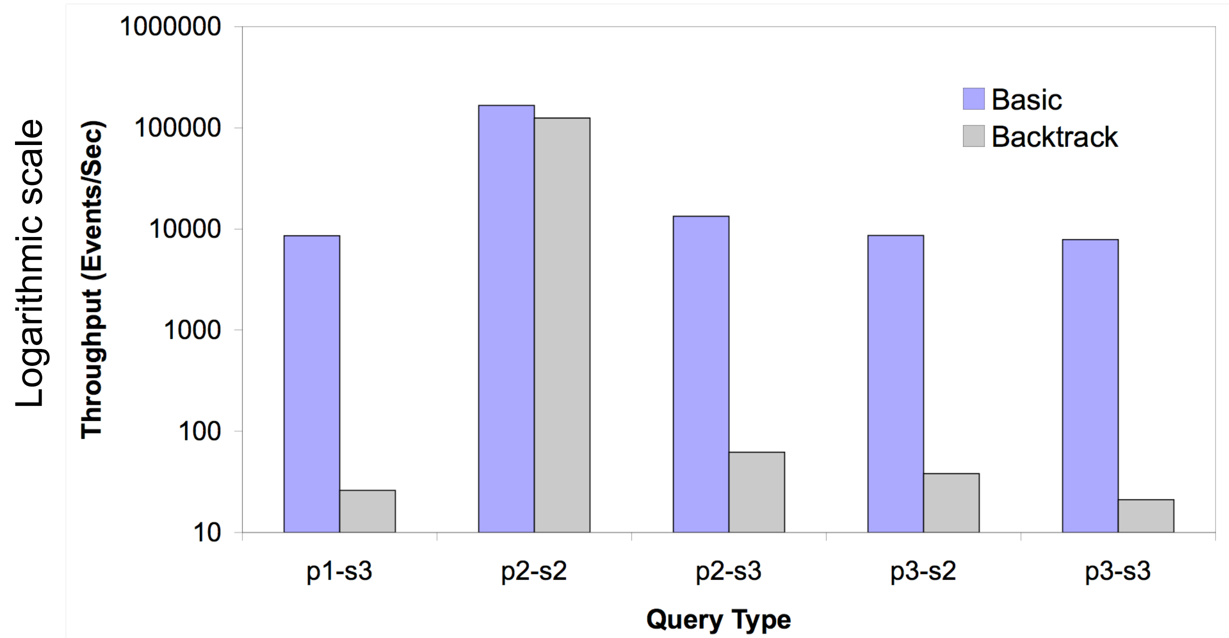
Basic Algorithm:
shared buffer + *separate run execution*



Predicate selectivity: strong effect on *match length* and *num. of runs*, hence overall performance.

Event selection strategy: s3 can be more expensive than s2.

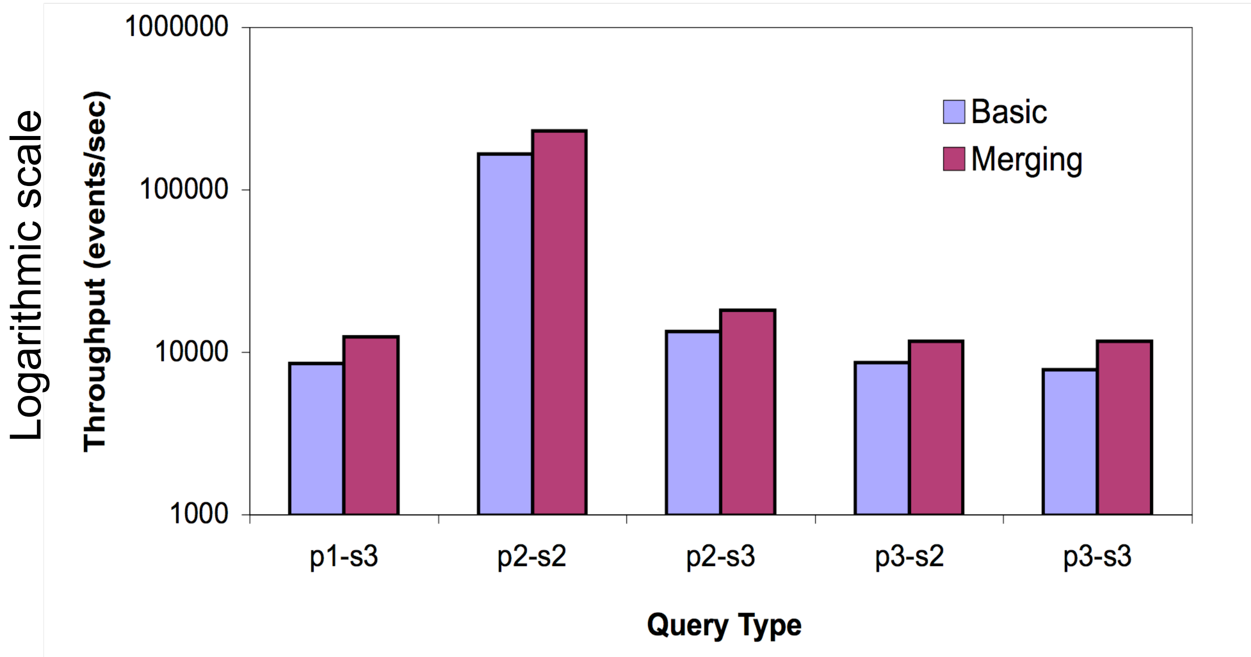
Comparing to a Backtrack Algorithm



Basic vs. Backtrack:

- Basic evaluates all runs of the automaton simultaneously; it processes each event only once.
- Backtrack handles one run at a time, backtracks upon failure or to find another match; it reprocesses events multiple times.

Benefit of Shared Processing



Benefits of merging runs of automata:

- Performance gains 40% to 110% across all queries.
- Throughput over 10,000 events/sec even for expensive queries.
- Higher performance gains when partition window size increases.

Summary

- NFA^b automaton, a formal evaluation model for event pattern queries
  - Expressibility of NFA^b
  - Compilation techniques
- Runtime complexity and sharing techniques
- Performance results
  - Tens of thousands of events/sec for fairly expensive queries
  - Even higher throughput for cheaper queries
  - Sharing among runs offers 40%-110% performance gains
- Potential impact: a pattern matching operator to be integrated into relational stream systems

Future Work

- Query complexity analysis
  - What pattern queries can be evaluated using constant time per event?
- Optimizations
  - Negation
  - Composed queries
- Robust event processing
  - Uncertain events
  - Out of order events
- Benchmark for event pattern matching

Questions

SASE+