SIMD-Accelerated Regular Expression Matching

Eva Sitaridi, Orestis Polychroniou, Ken Ross
Columbia University

DaMoN 2016, San Fransisco, California

CS@CU

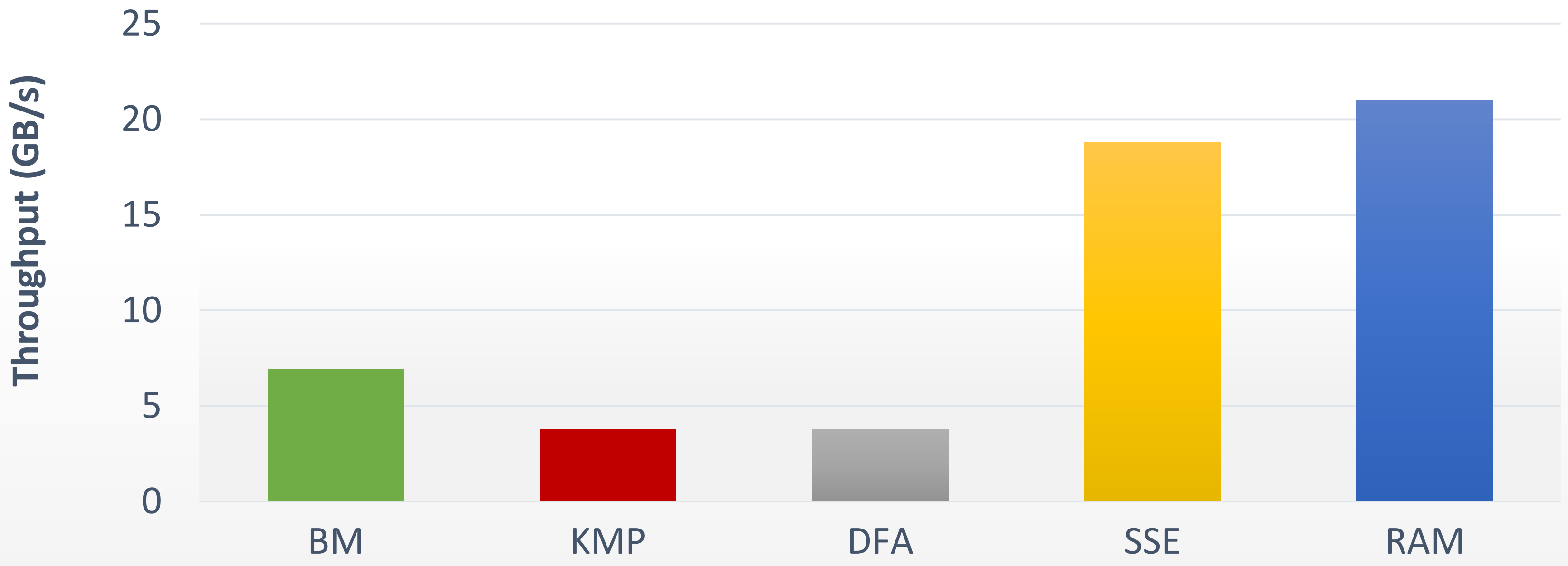
In-Memory Query Execution

- Bottleneck shifts from disk-access to RAM access
- Hardware-aware processing to reach RAM bandwidth
  - Mainstream multi-cores (e.g. Intel Haswell, Skylake)
    - Complex cores
    - Aggressively out-of-order
  - Many-core accelerators(e.g. Intel Xeon Phi)
    - Simpler cores
    - In-order execution
    - Wider SIMD width and more cores

Vectorized Database Operators

- Scan filters: e.g. age≥21
  - Saturate RAM bandwidth ✓
    - Use SIMD instructions & thread parallelism
- LIKE operators: e.g.: comment LIKE '%packages%'
  - Algorithms
    - Boyer-Moore
    - Knuth-Morris-Pratt
    - DFA
    - SSE instruction
  - Can we reach RAM bandwidth?

LIKE Performance



Saturate RAM bandwidth by using SSE cmpistri/cmpestri instructions

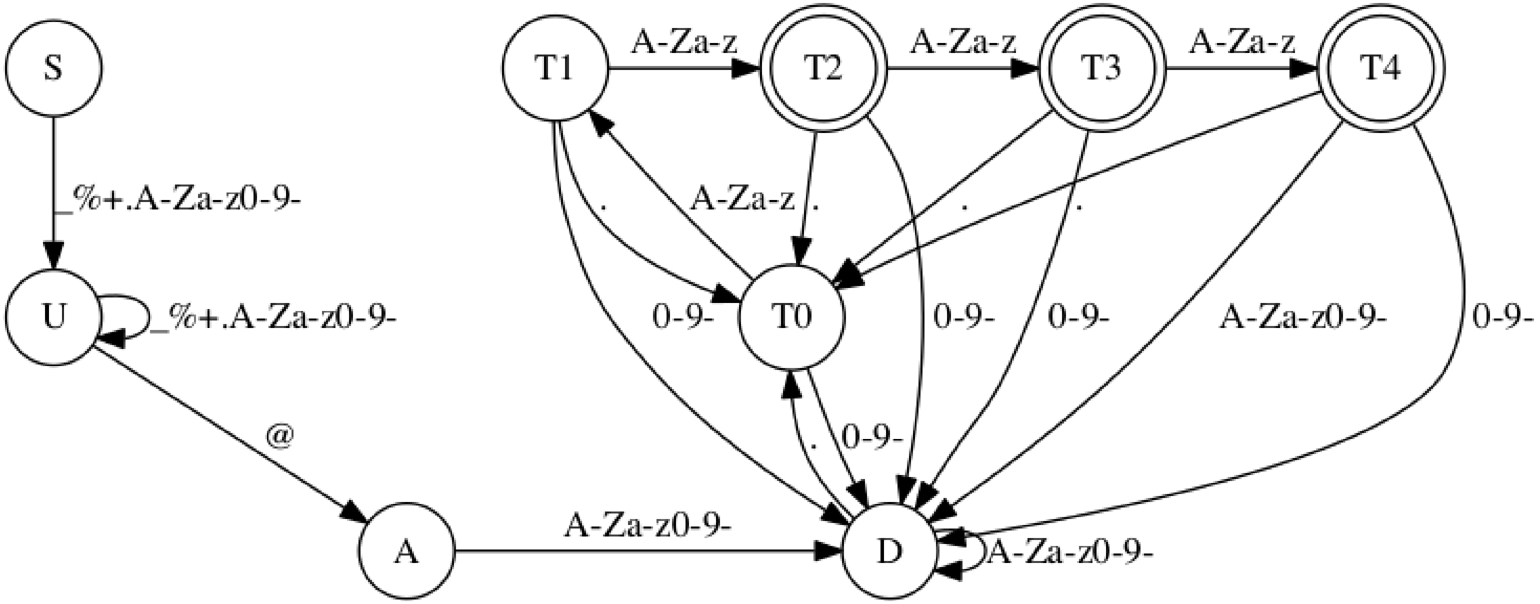
- For pattern ≤16 chars (common case for DB queries)

Databases & Regular Expressions

- Use more expressive predicates: RLIKE or REGEXP
  - Validate string columns: E-mail address
  - Pattern-matching
    - Her OR She
    - Her AND She
    - Her AND NOT She

Regular expressions mapped to Deterministic Finite Automata (DFA)

Example: E-mail DFA



Stopping conditions

- Invalid string value: e.g. user@@mail.com
- Locate search pattern: Valid addresses in a specific street
- Consume full input string

DFA-Based Expression Matching

- DFA Representation with **N** states and **c** alphabet size
  - Array of N x c characters
  - Special states
    - Reject sink
    - Accept sink
  - Reorder states to simplify termination condition
    - -2: Accept sink
    - -1: Reject sink
    - [0,N)

DFA Transition Array Example

Alphabet character

Current state

	…	‘a’	‘b’	‘c’	…	‘@’	‘#’	‘%’
0		2	5	**-1**	**…**	**-1**	**-2**	8
1		3	**-1**	4	…	**-2**	**-2**	8
2		1	5	**-1**	…	5	**-2**	**-1**
…								

DFA footprint: N states x 256 characters x Bytes/state

Scalar DFA matching

```
bool regexp(…){
    size_t i = 0, s = initial_state;
    do {
        s = dfa[(s << 8) | string[i]];
    } while (0 <= (ssize_t) s && ++i != str_len);
    return s + 1 > reject_states; }
```

Cache access

Optimize scalar performance

Scalar DFA matching

```
bool regexp(…){
    size_t i = 0, s = initial_state;
    do {                    Cache access
        s = dfa[(s << 8) | string[i]];
    } while (0 <= (ssize_t) s && ++i != str_len);
    return s + 1 > reject_states; }
```

Optimize scalar performance

- Use integer arithmetic to access the DFA transition table

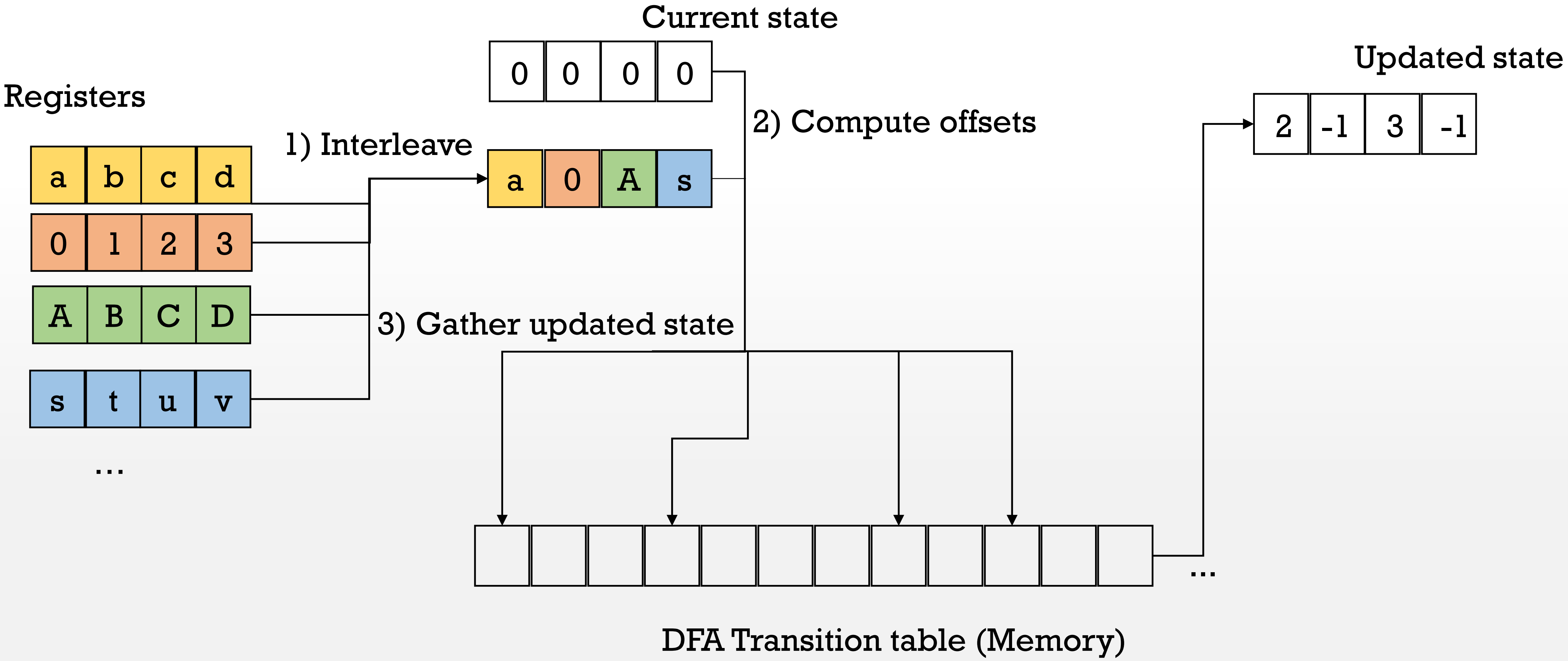
Scalar DFA matching

```
bool regexp(…){
	size_t i = 0, s = initial_state;
	do {                    Cache access
		s = dfa[(s << 8) | string[i]];
	} while (0 <= (ssize_t) s && ++i != str_len);
	return s + 1 > reject_states; }
```

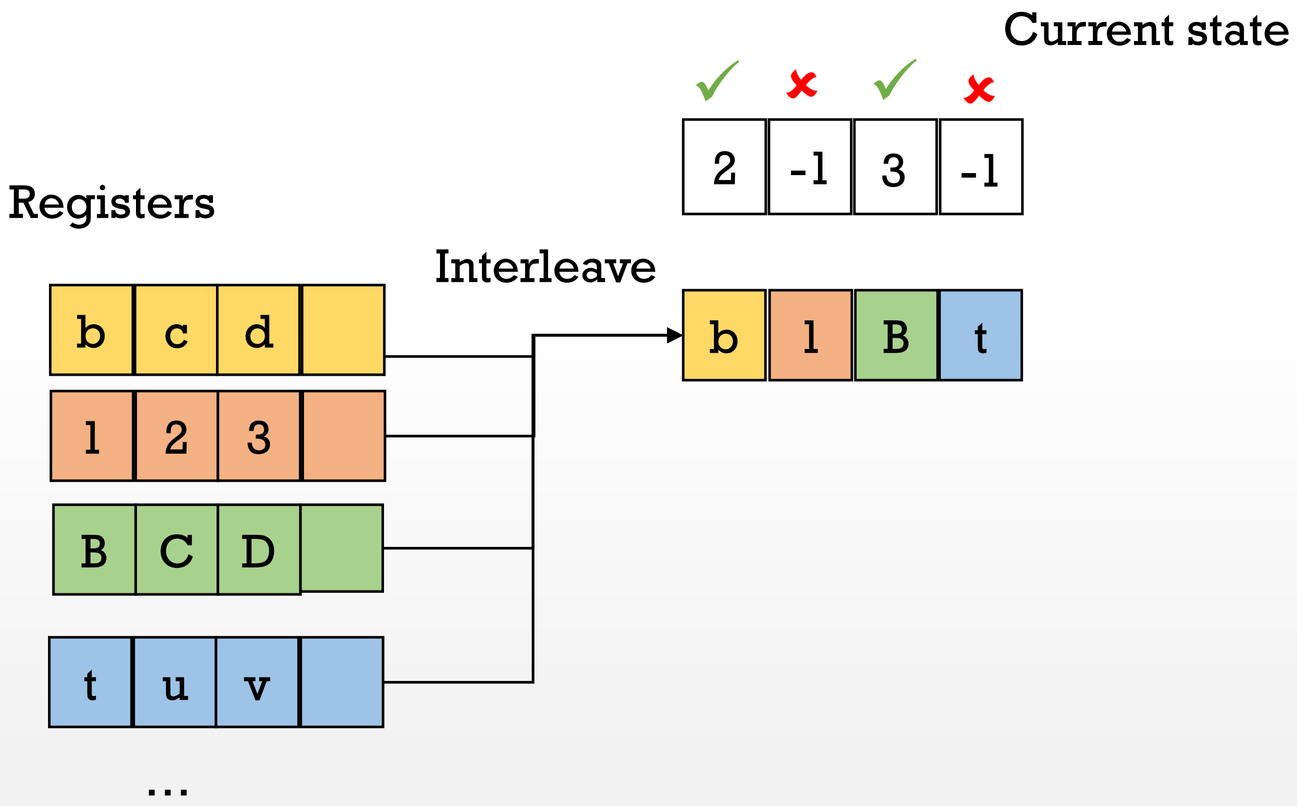
Optimize scalar performance

- Use integer arithmetic to access the DFA transition table
- Minimize branches

Previous Approach: Lock-step Processing

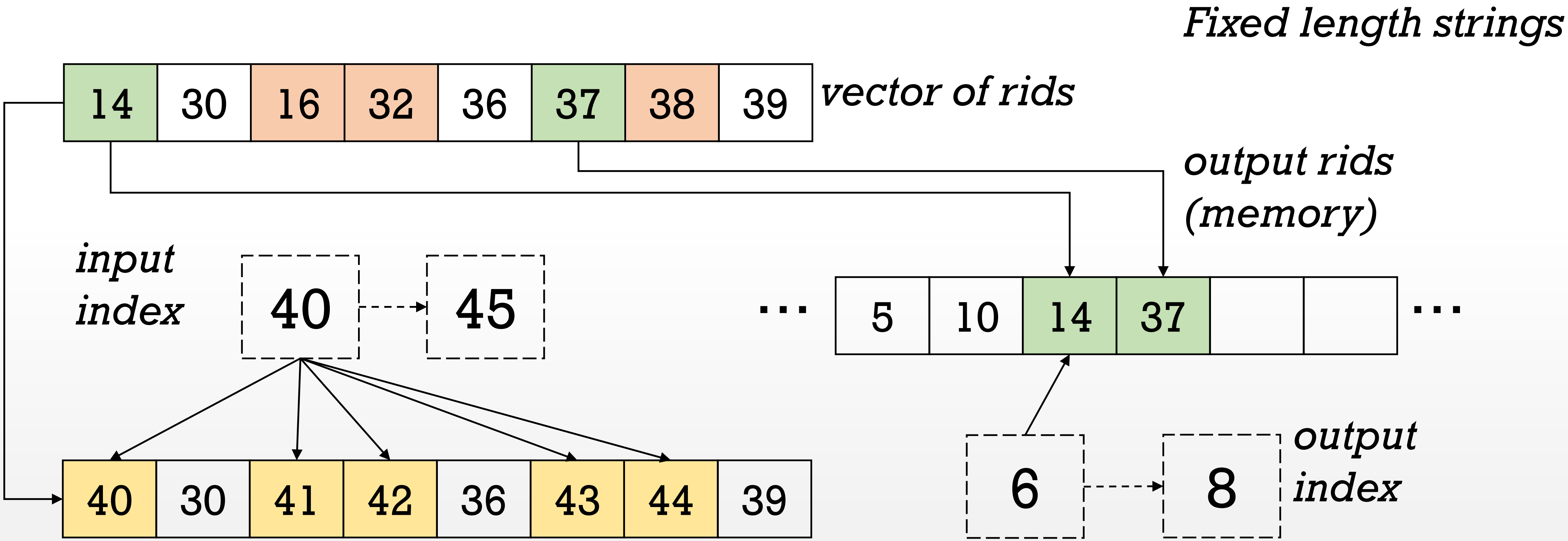


Previous Approach: Lock-step Processing



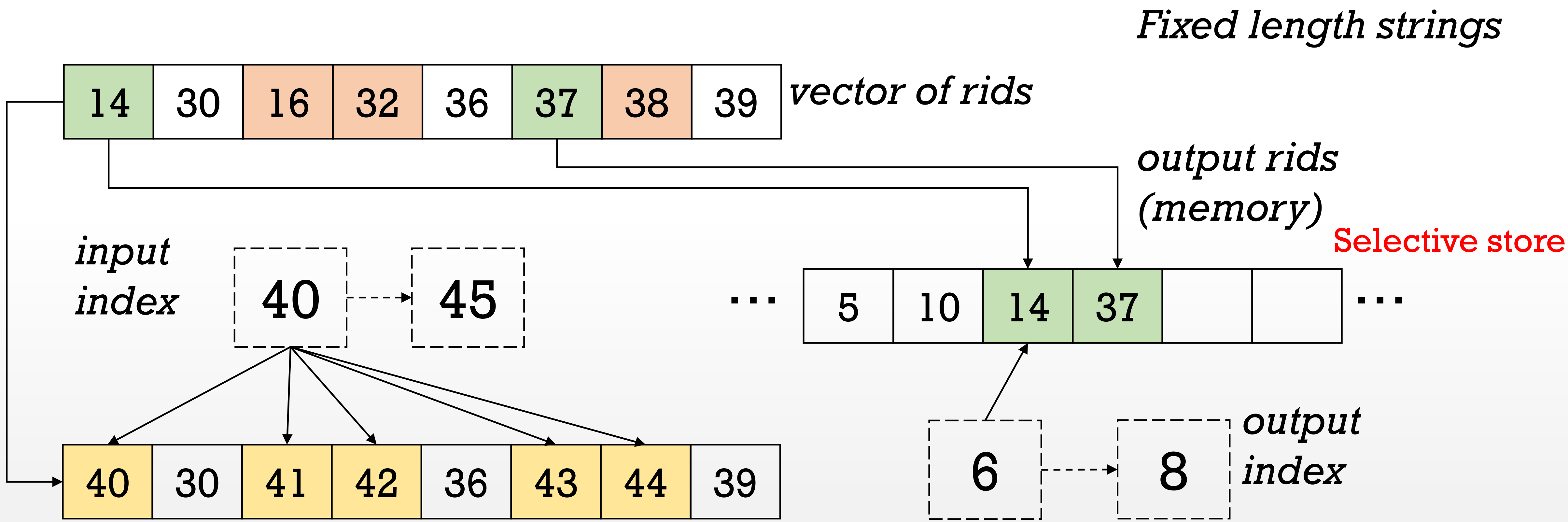
Weakness: Lanes completed underutilized!

Our Approach: Selective Loads & Stores



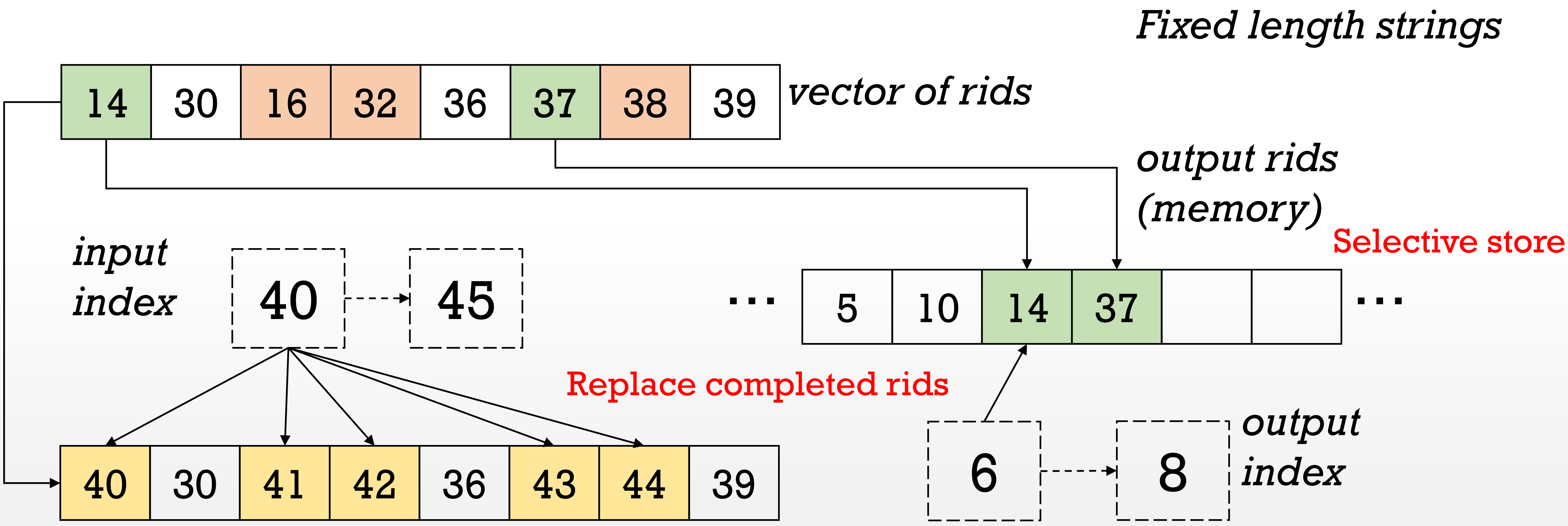
Short Circuiting: Replace completed strings

Our Approach: Selective Loads & Stores



Short Circuiting: Replace completed strings

Our Approach: Selective Loads & Stores

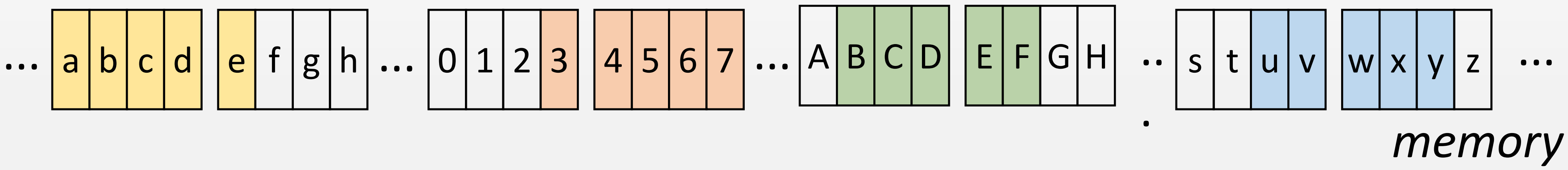


Short Circuiting: Replace completed strings

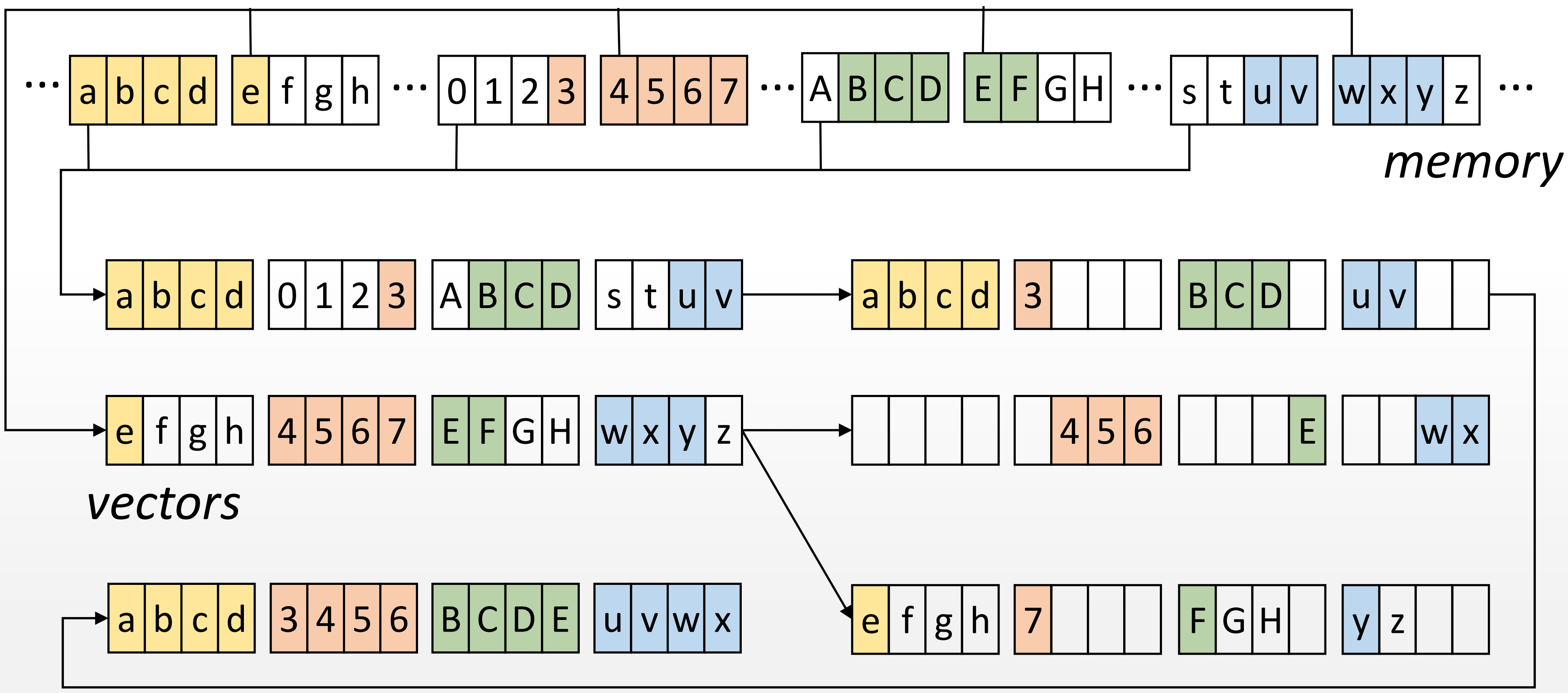
Our Approach: Buffer Multiple Bytes

- Executing a gather of 1 byte same cost with gather of a 4-byte word
- A non-trivial portion per string processed to determine if matches regular expression

Buffer multiple characters per string



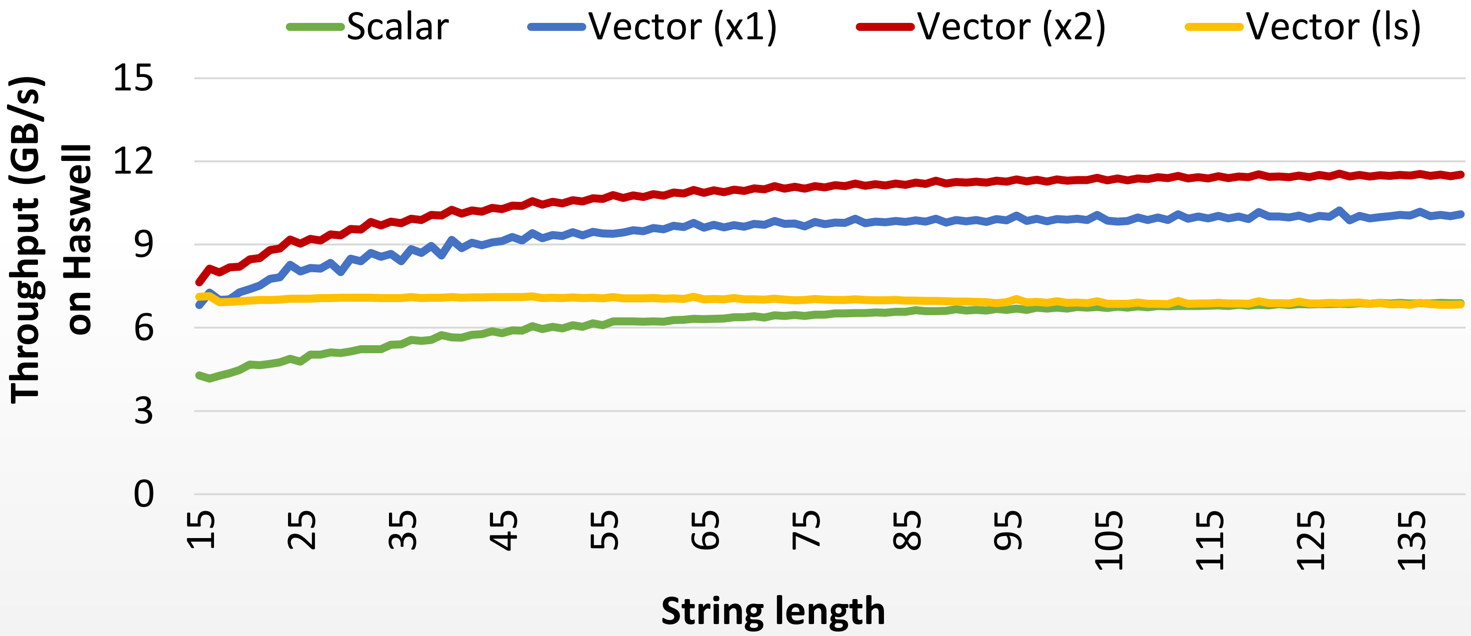
Our approach: Gather Unaligned Input



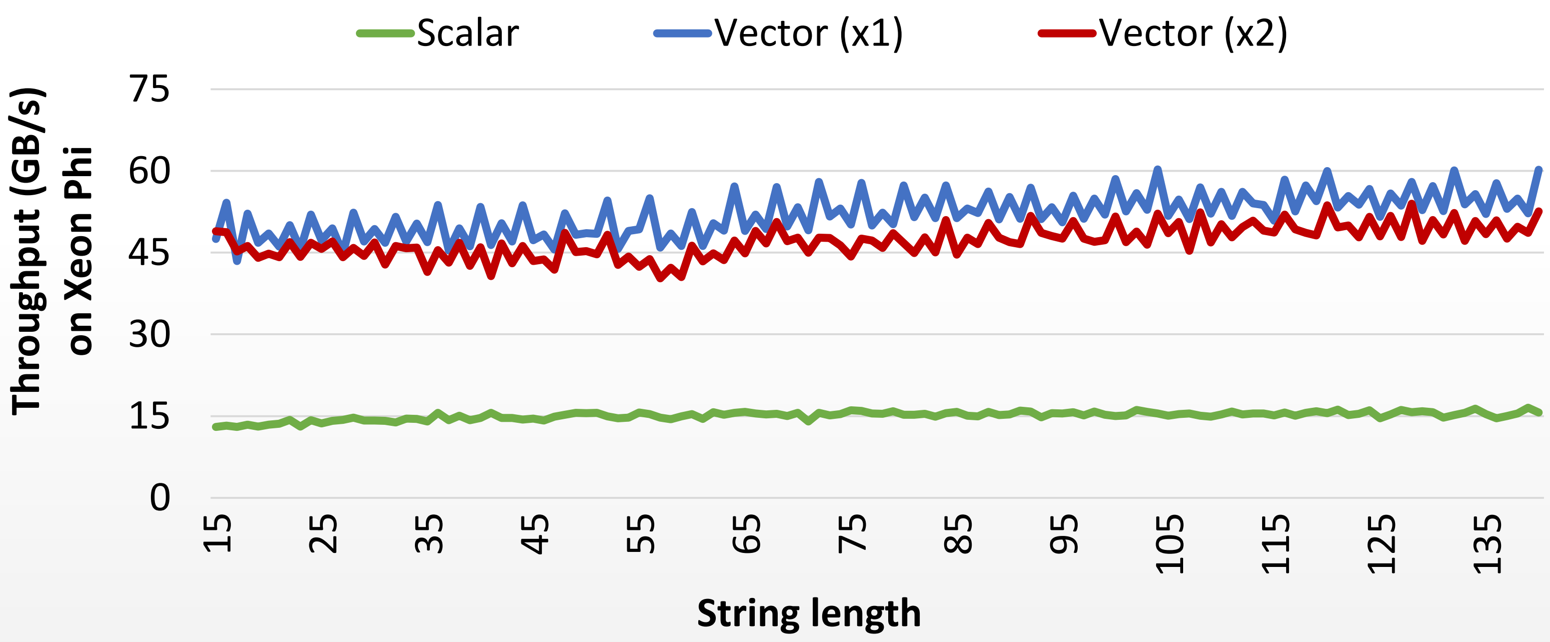
Experimental Results Overview

- DFA Matching Implementations
  - Scalar
  - Lock-step (AVX2)
  - Short circuit AVX2 (Unrolling/No unrolling)
  - Short circuit Intel Xeon Phi (Unrolling/No unrolling)
- Experiments for URL validation (90 total states)
  - Vary string length
  - Vary matching failure point
  - Vary matching selectivity

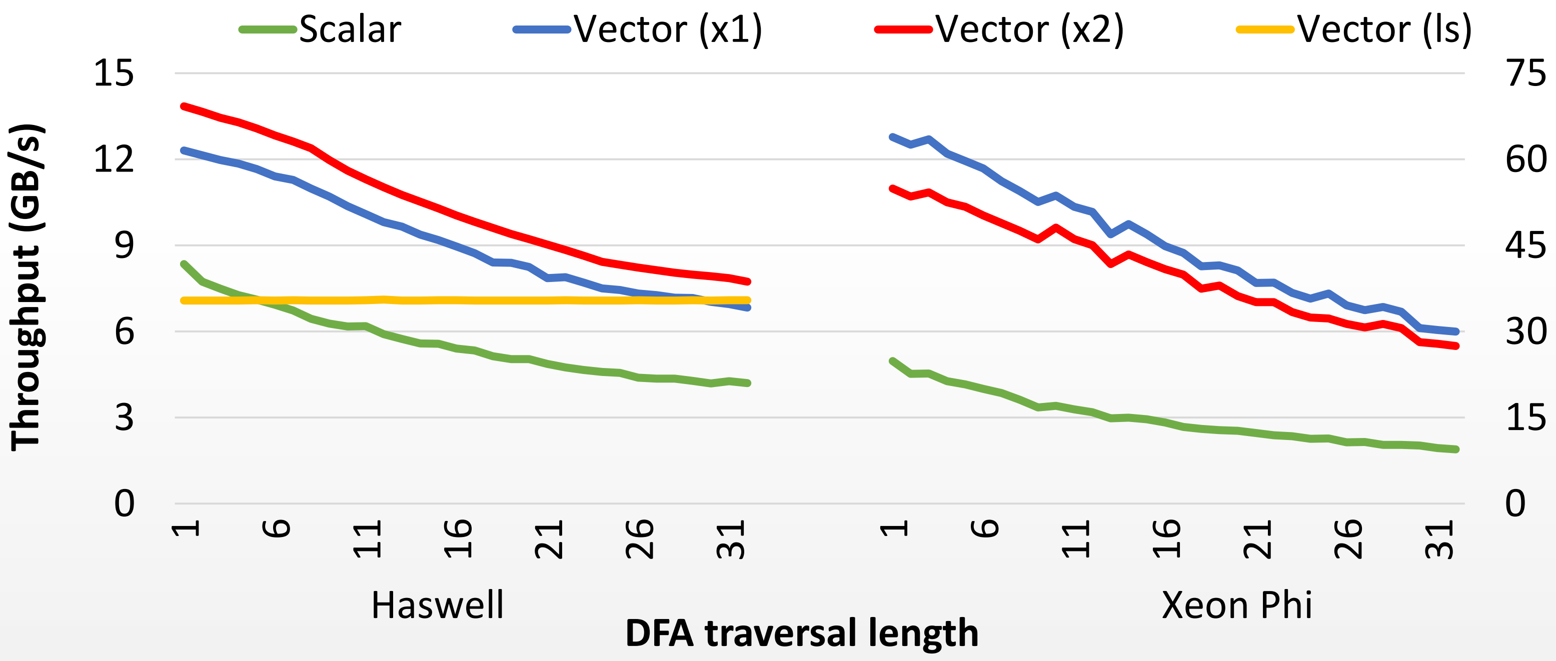
Vary String Length – Haswell



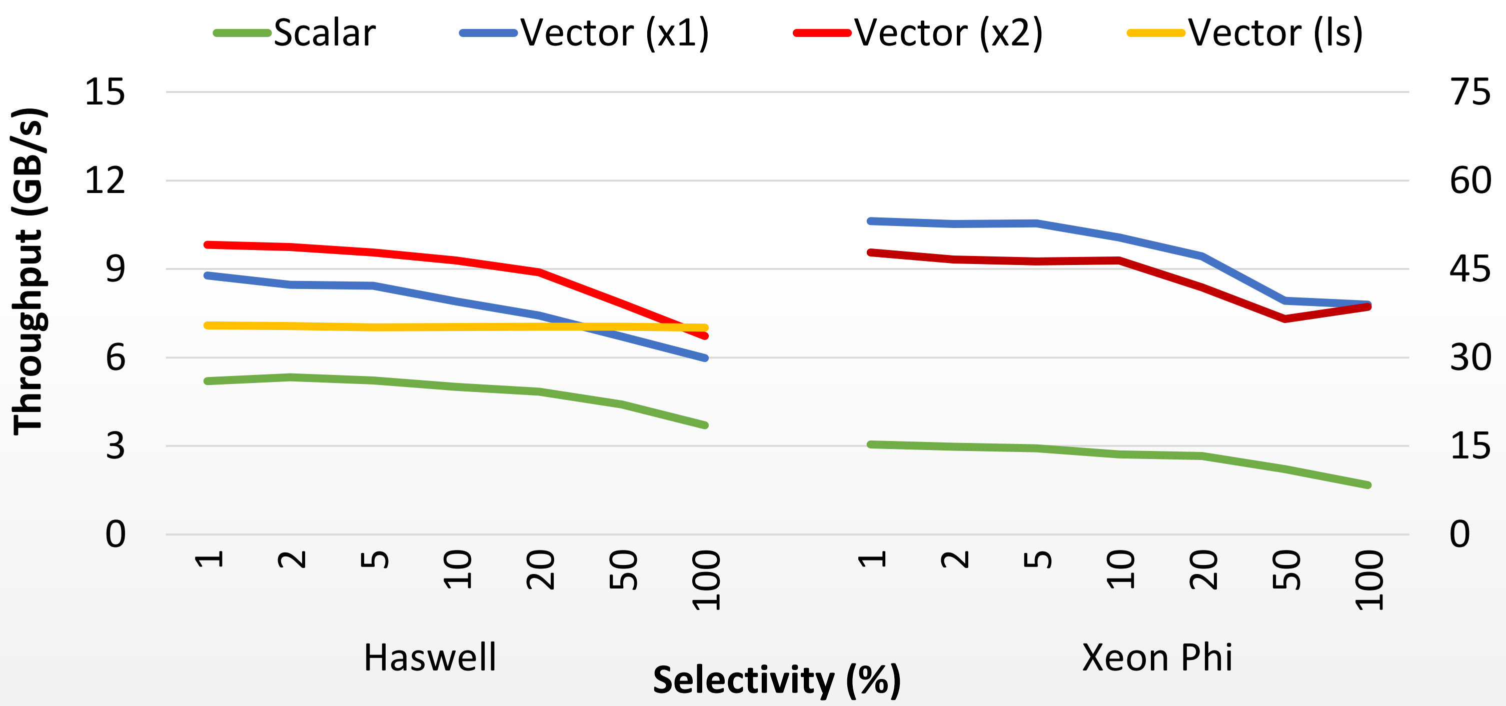
Vary String Length – Xeon Phi



Varying Matching Failure Point



Vary Matching Selectivity



Conclusions

- Vectorize regular expression matching
  - Prior approach: Lock-step input processing
  - Our approach: Dynamically replace completed input
    - 2X speed-ups on mainstream CPU & 5X speed-up on the Intel Xeon Phi
    - Maximum speed-ups for early failures or matches
- Show impact of vectorization on code bound by cache access

Questions?

Conclusions

- Vectorize regular expression matching
  - Prior approach: Lock-step input processing
  - Our approach: Dynamically replace completed input
    - 2X speed-ups on mainstream CPU & 5X speed-up on the Intel Xeon Phi
    - Maximum speed-ups for early failures or matches
- Show impact of vectorization on code bound by cache access

Back-up slides

Early Termination Examples

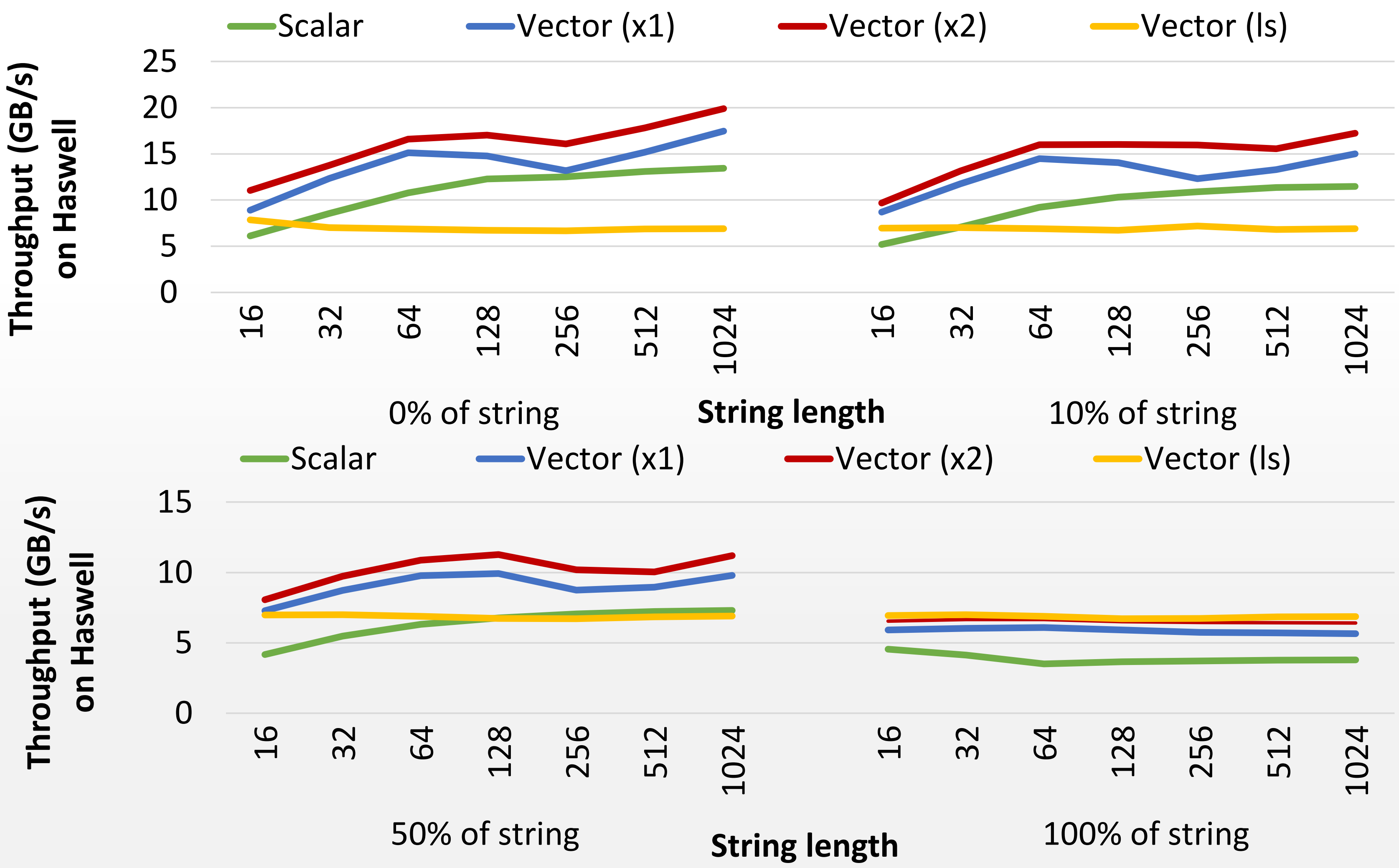
Regex	Scenario with possible early failures
e-mail	invalid character, e.g., me @mail.com double @ symbol, e.g., me@@mail.com specific domain: mail.com, e.g. me@meil.com specific username: john, e.g. jim@mail.com
URL	invalid character, e.g., http://site .com invalid scheme, e.g., htp://site.com specif8c site: site.com, e.g., http://no.com/a/b/c specific IP: 125.1.*.*, e.g., http://125.2.0.0/a/b/c specific path depth: 2, e.g., http://site.com/a/b/c
address	missing street number before street name non-numeric symbol in postal code specific street number in valid address
name	invalid symbol, e.g., J0hn Smith lowercase first letter, e.g., bob Smith specic surname: Stark, e.g., Peter Parker

URL Regular Expression

scheme, username, hostname (or IP), port, path, query, fragment

```
^(ht|f)tp(s)?://([!$&'()*+,;=A-Za-z0-9:-]+@)?
((([_~!$&'()*+,;=A-Za-z0-9-]|(%[0-9A-F]{2}))+.)+[a-zA-Z]{2,4})
|((([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5]).){3}
([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5]))(:[0-9]*)?
(/([._~!$&'()*+,;=A-Za-z0-9:@-]|(%[0-9A-F]{2}))+)*
(?([?._~!$&'()*+,;=A-Za-z0-9:@/-]|(%[0-9A-F]{2}))*)?
(#([?._~!$&'()*+,;=A-Za-z0-9:@/-]|(%[0-9A-F]{2}))*)?$
```

Varying String Length



Varying DFA Size