In-Memory Automata Processing

Reetuparna Das

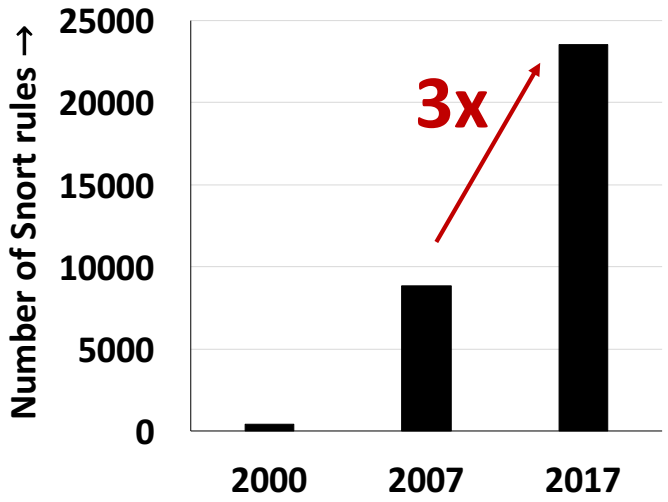
Assistant Professor

University of Michigan – Ann Arbor

Pattern matching in abundance …

```
# alert tcp $ExXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"PROTOCOL-
SCADA Cogent DataHub server-side information disclosure"; flow:to_server,established;
content:".asp."; nocase; http_uri; pcre:"/\x2easp\x2e($|\?)/iU"; metadata:service http;
reference:cve,2011-3502; classtype:web-application-attack; sid:20174; rev:4;)xx
```

Network Intrusion Detection and Protection



10-100 Gbps line rates

Pattern matching in abundance …

127.0.0.1 - - [07/Dec/2016:11:04:58 +0100] "GET /HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:49.0) Gecko/20100101 Firefox/49.0"

Log Processing

```
/ right/JJ to/[^\s]+ /
/ the/[^\s]+ back/RB /
/ [^/]+/DT longer/[^\s]+ /
/ ,/[^\s]+ have/VB /
/ [^/]+/VBD by/[^\s]+ /
```

Natural Language Processing

alert tcp $ExXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"PROTOCOL-SCADA Cogent D…aHub server-side information disclosure"; flow:to_server,established; con… case; … service http; re… 011-35… 174; …)xx

and many more …

Network Intrusion Detection

```
<book id="bk101">
<author>Gambardella, Matthew</author>
<title>XML Developer's Guide</title>
<genre>Computer</genre>
<price>44.95</price>
<publish_date>2000-10-01</publish_date> <description>An in-depth look at creating applications with XML.</description> </book>
```

XML Parsing

```
/([STX])(.{2}?)([DBEZX])/
/([RKX])(.{2,3}?)([DBEZX])(.{2,3}?)([YX])/
/([GX])([^EDRKHPFYW])(.{2}?)([STAGCNBX])([^P])/
```

Motif search

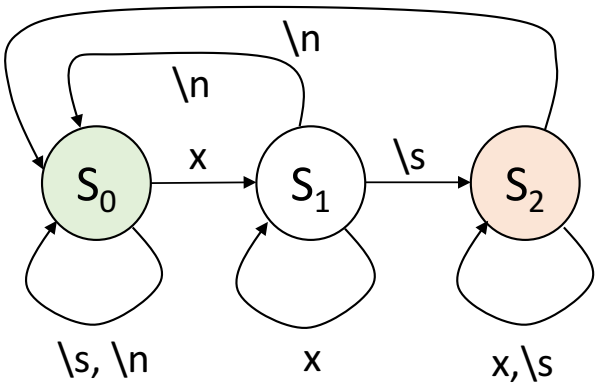
Video Decoding

Particle Path Tracking

```
Donald Duck
Donald Fauntleroy Duck
Donald F Duck
D F Duck
```

Data Analytics

Finite State Automata extract patterns



T	x	\s	\n
S_0	S_1	S_0	S_0
S_1	S_1	S_2	S_0
S_2	S_2	S_2	S_0

Σ = {x, \s, \n}

Compute-Centric Architectures

Switch-Case

```
1 while (c != EOF) {
2    if (c == '\n') { putchar('\n'); *state = S0; }
3    else
4       switch(*state) {
5          case S0:
6                c != '\s') {
7                   putchar(c); *state = S1;
8             } break;
9          case S1: if (c == '\s') { *state = S2; }
10                  else { putchar(c); } break;
11         case S2: break;
12    }
13 }
```

! **Branch Mispredictions**

! **Irregular memory accesses**

Compute-Centric Architectures

Table-Lookup

```
1 struct T_table [3][3] = {
2   { {S0, S1}, {S0, S0}, {S0, S0} },
3   { {S1, S1}, {S1, S2}, {S1, S0} },
6 while (c != EOF) {
7   int id1 = (c == '\s') ? 0 : (c == '\n') ? 1 : 2
8   *state = T_table [*state] [id] [1]
9   putchar(c);
10 }
```

! **Memory bandwidth bottlenecked** ➡ ! **Limited state transitions per cycle**

Memory-Centric Architectures

1 rank with 8 chips

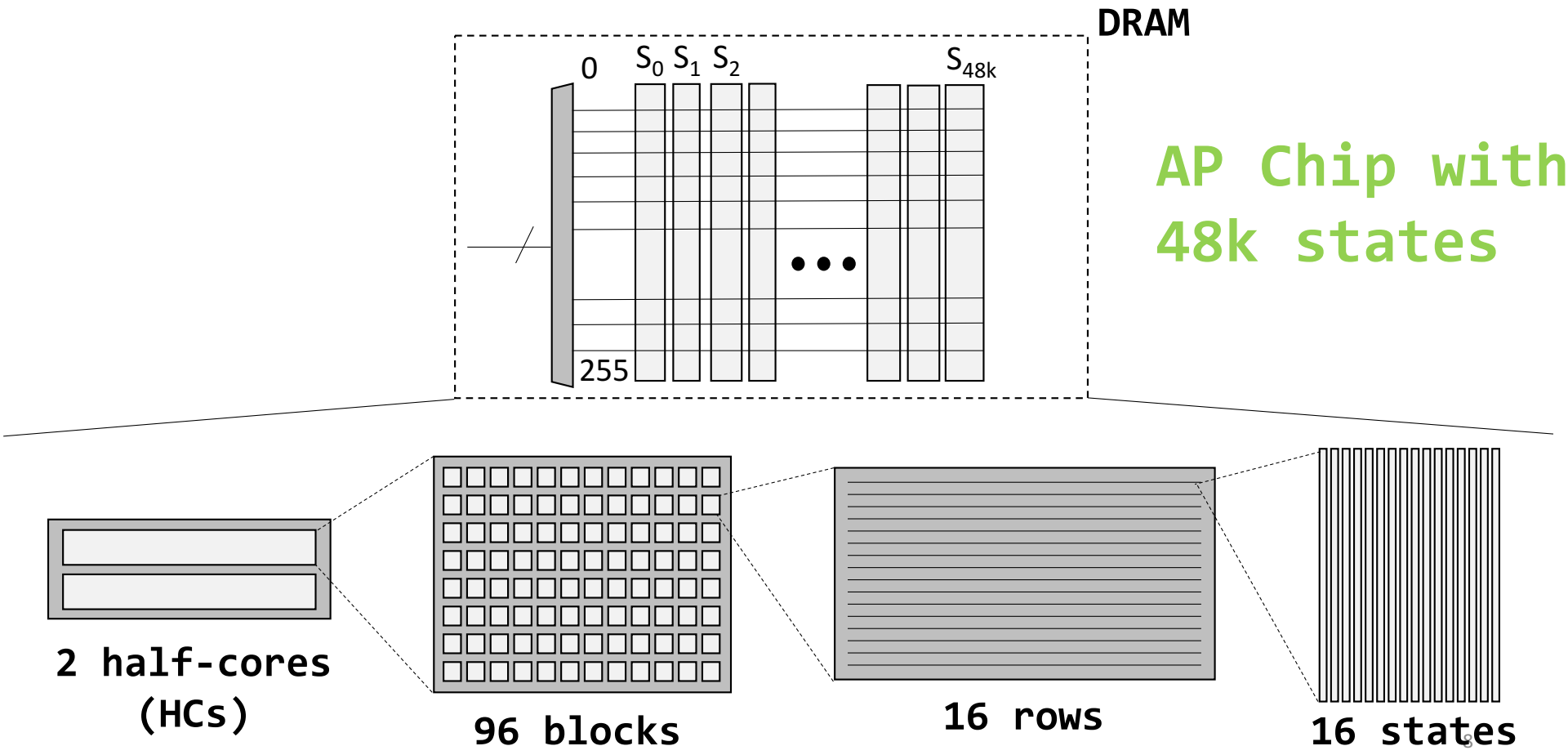
Micron's Automata Processor (AP)

48k state transitions per cycle per chip

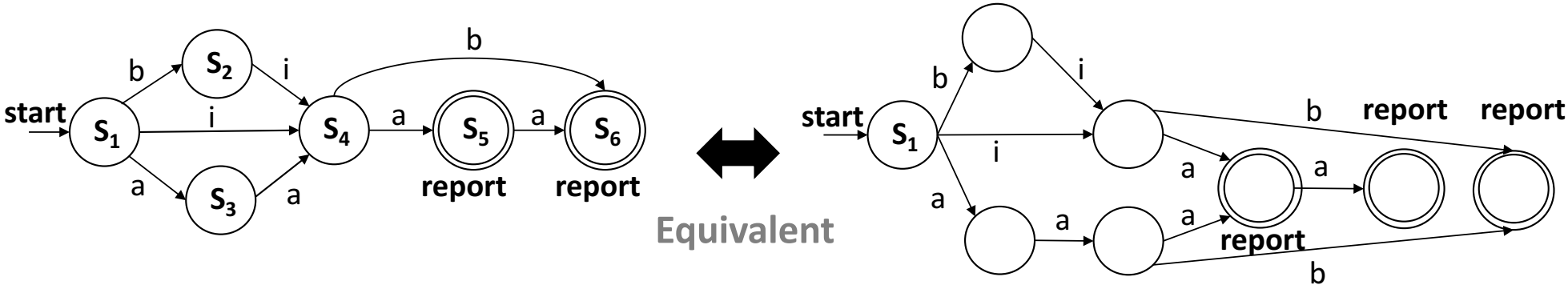
256x **faster than CPU** **170x** **faster than GPU (iNFAnt2)**

[ANMLZoo, Wadden et al. '16]

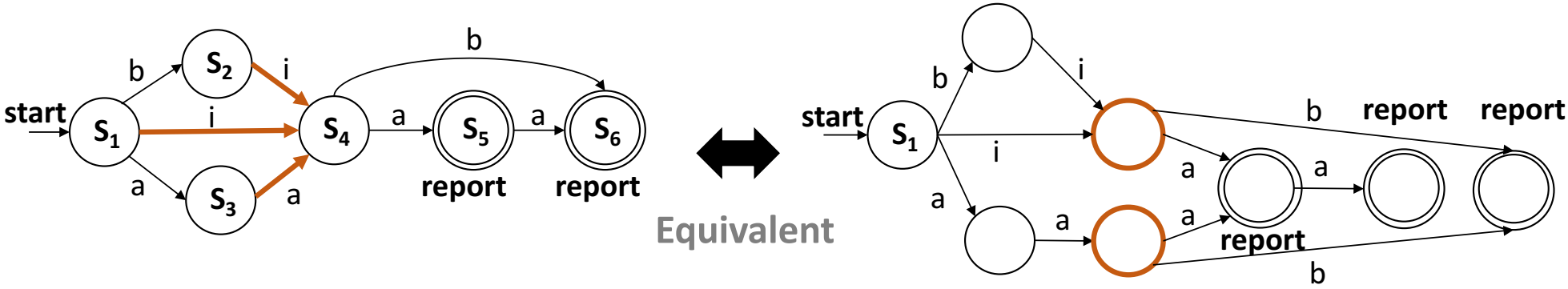
Hierarchical Organization



In-memory automata processing

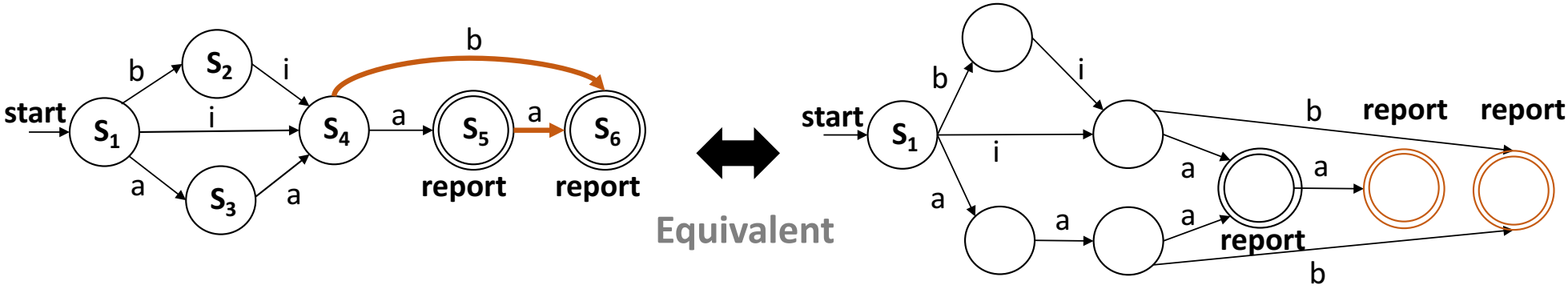


In-memory automata processing



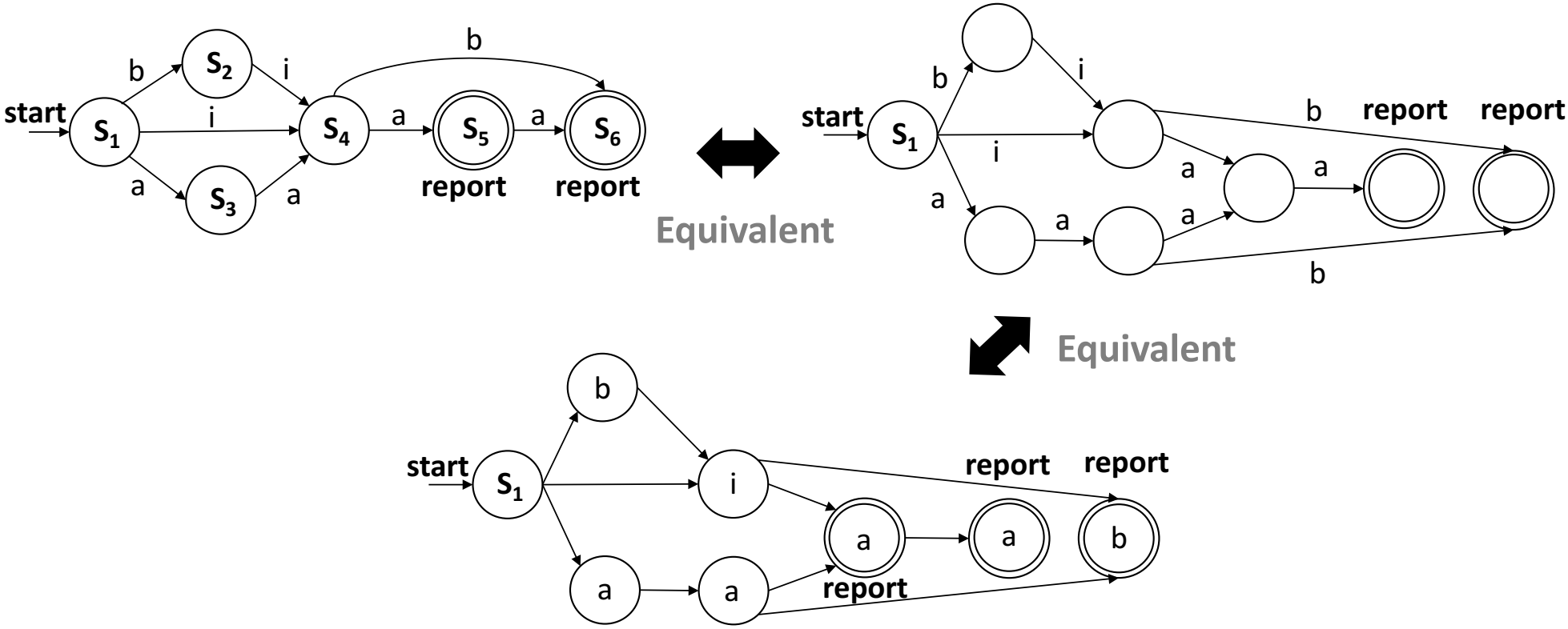
Each state has incoming transitions on one input symbol

In-memory automata processing

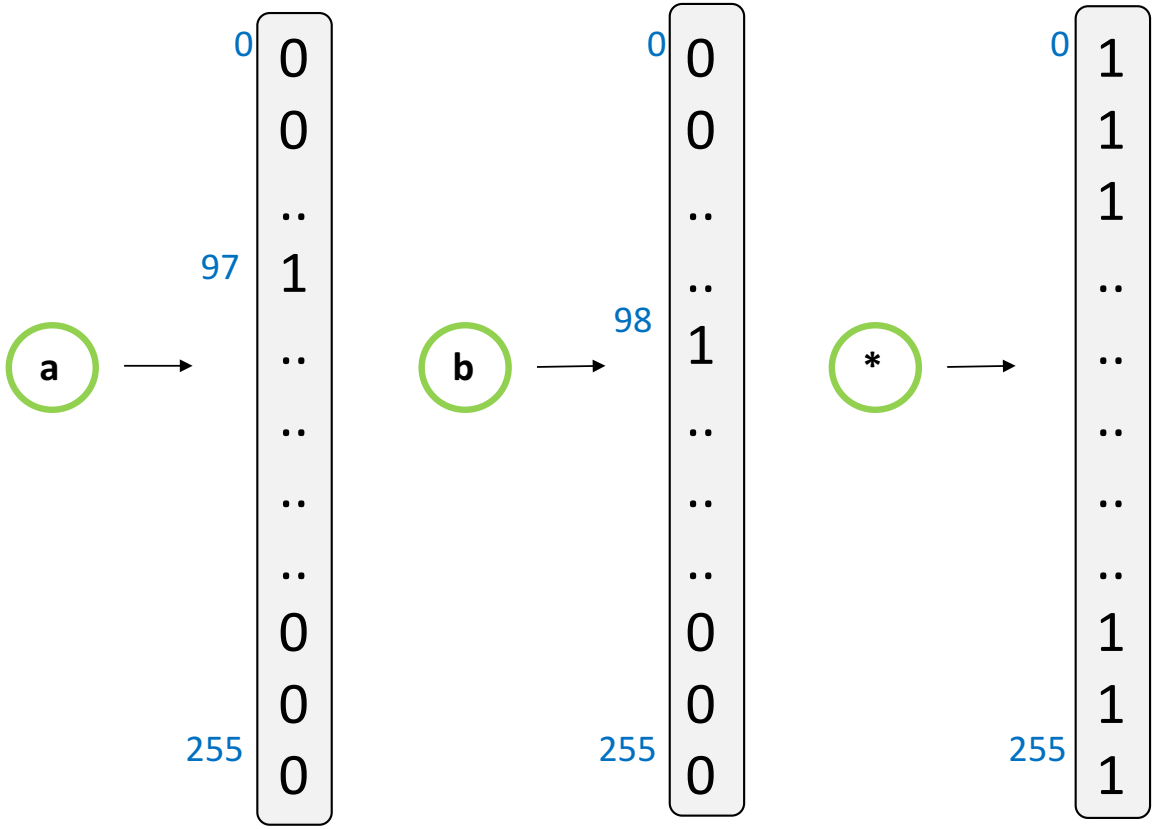


Each state has incoming transitions on one input symbol

In-memory automata processing



State representation

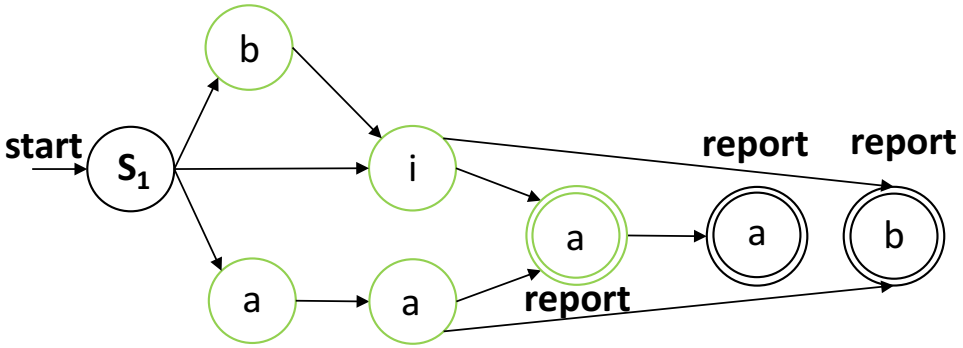


One-hot encoding

8-bit ASCII alphabet

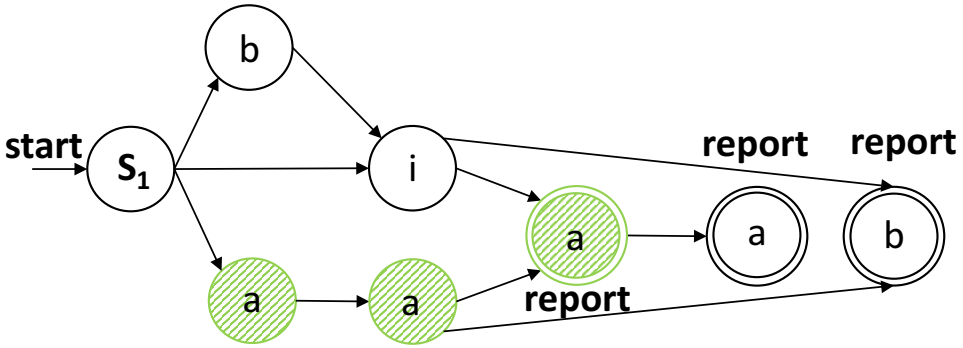
In-memory automata processing

Input symbols



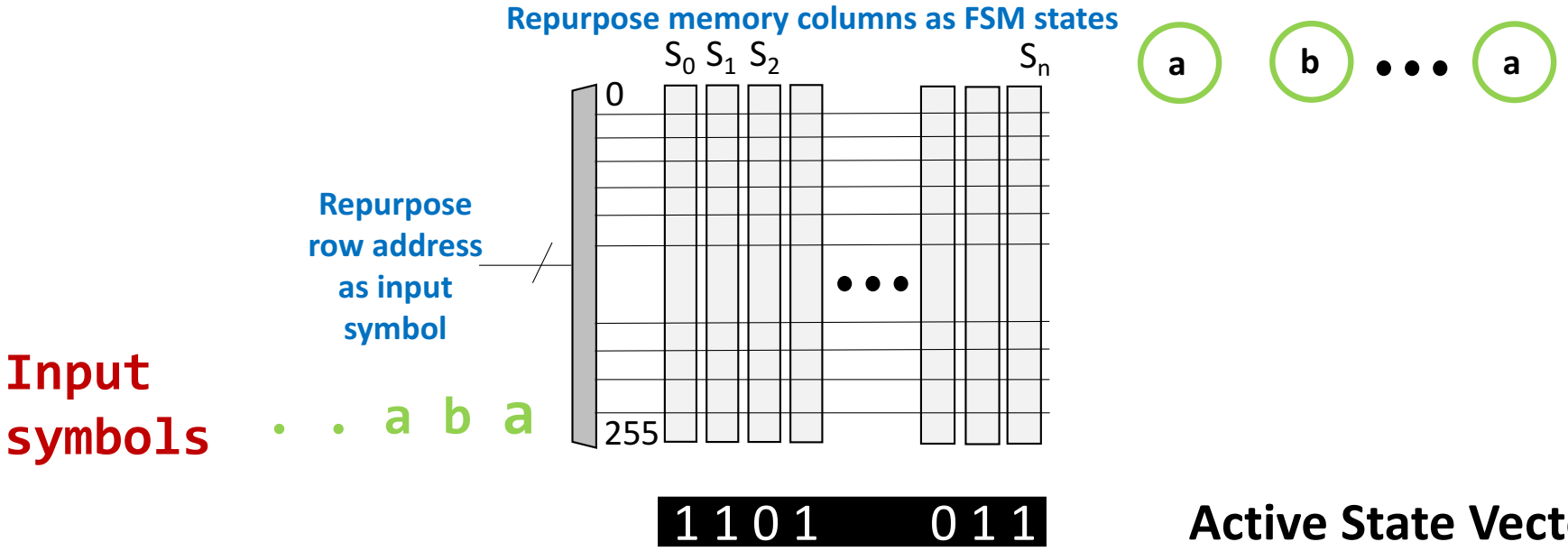
In-memory automata processing

Input symbols a b a. . .

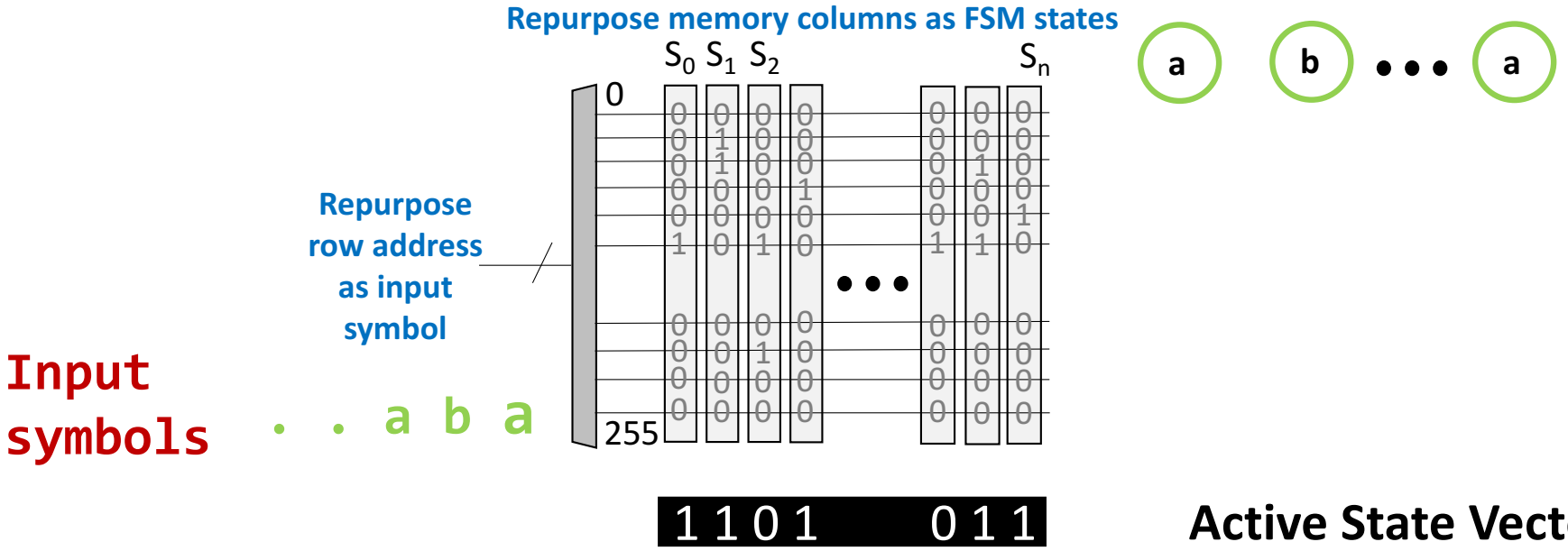


1 **State-Match**

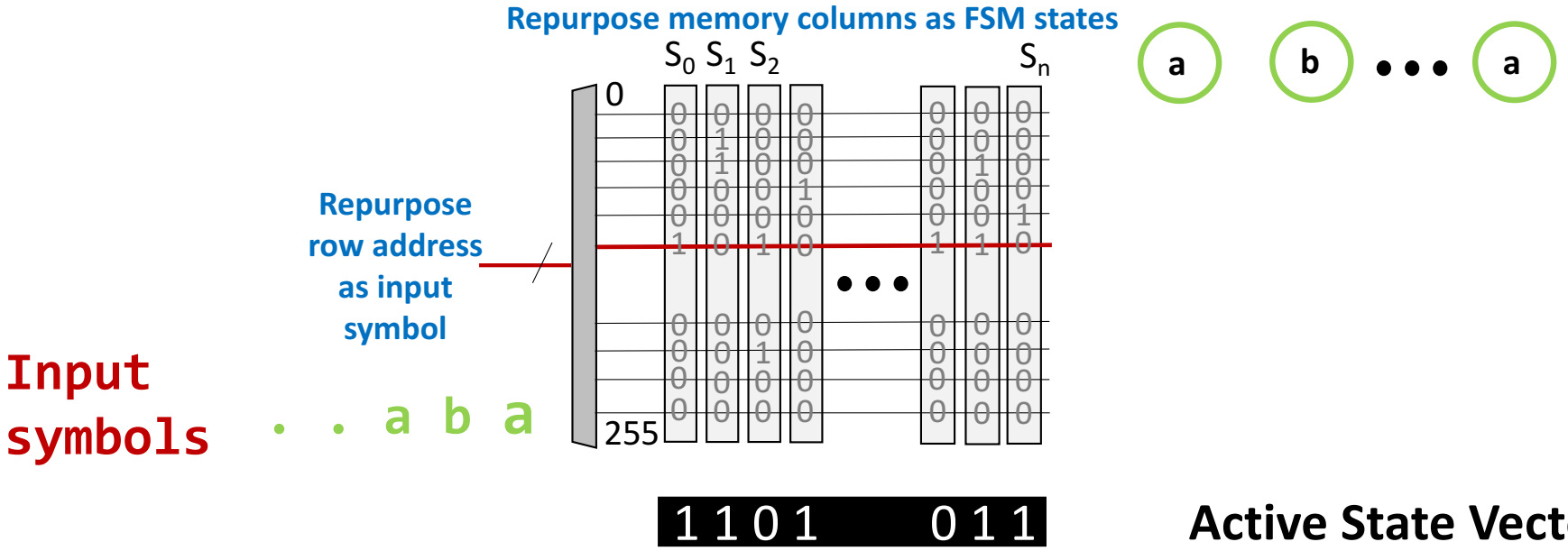
Massively Parallel State-Match



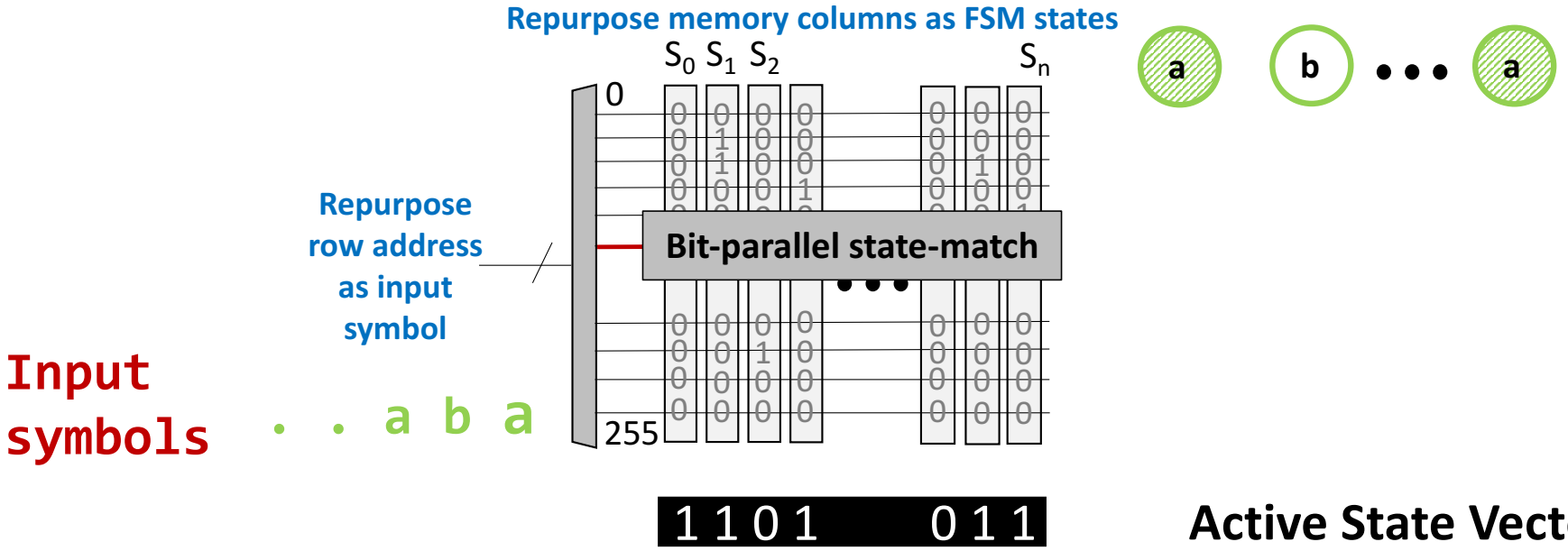
Massively Parallel State-Match



Massively Parallel State-Match

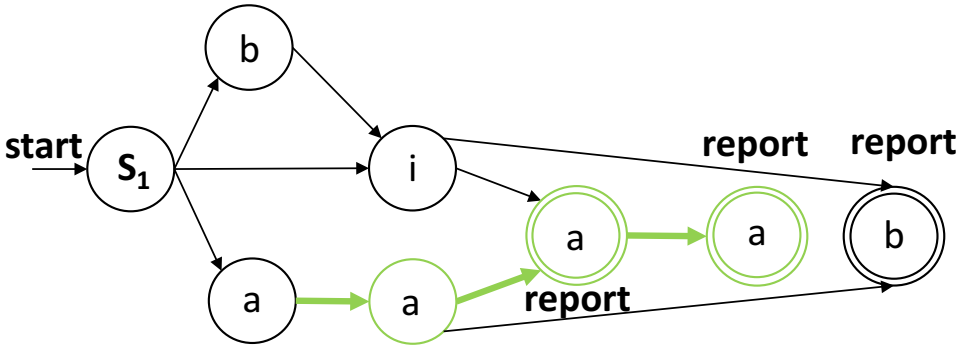


Massively Parallel State-Match



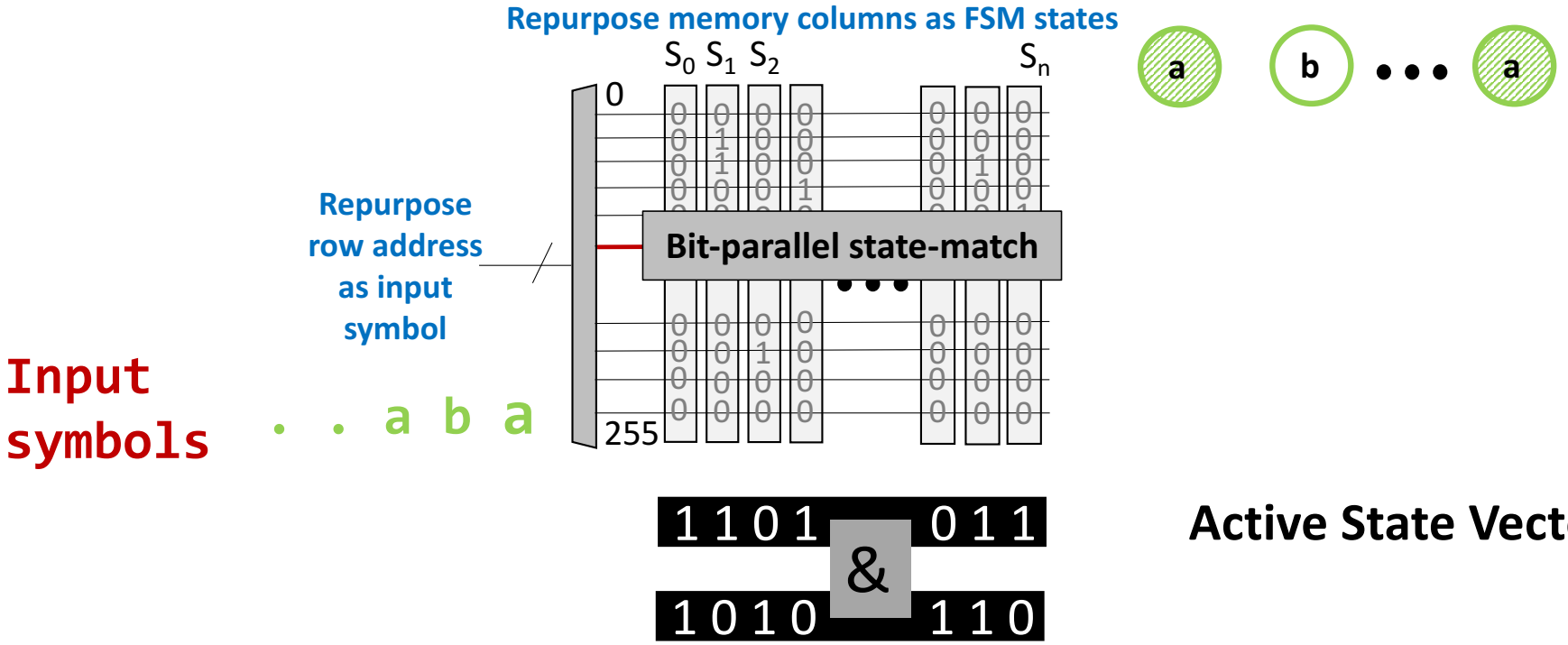
In-memory automata processing

Input symbols a b a. . .

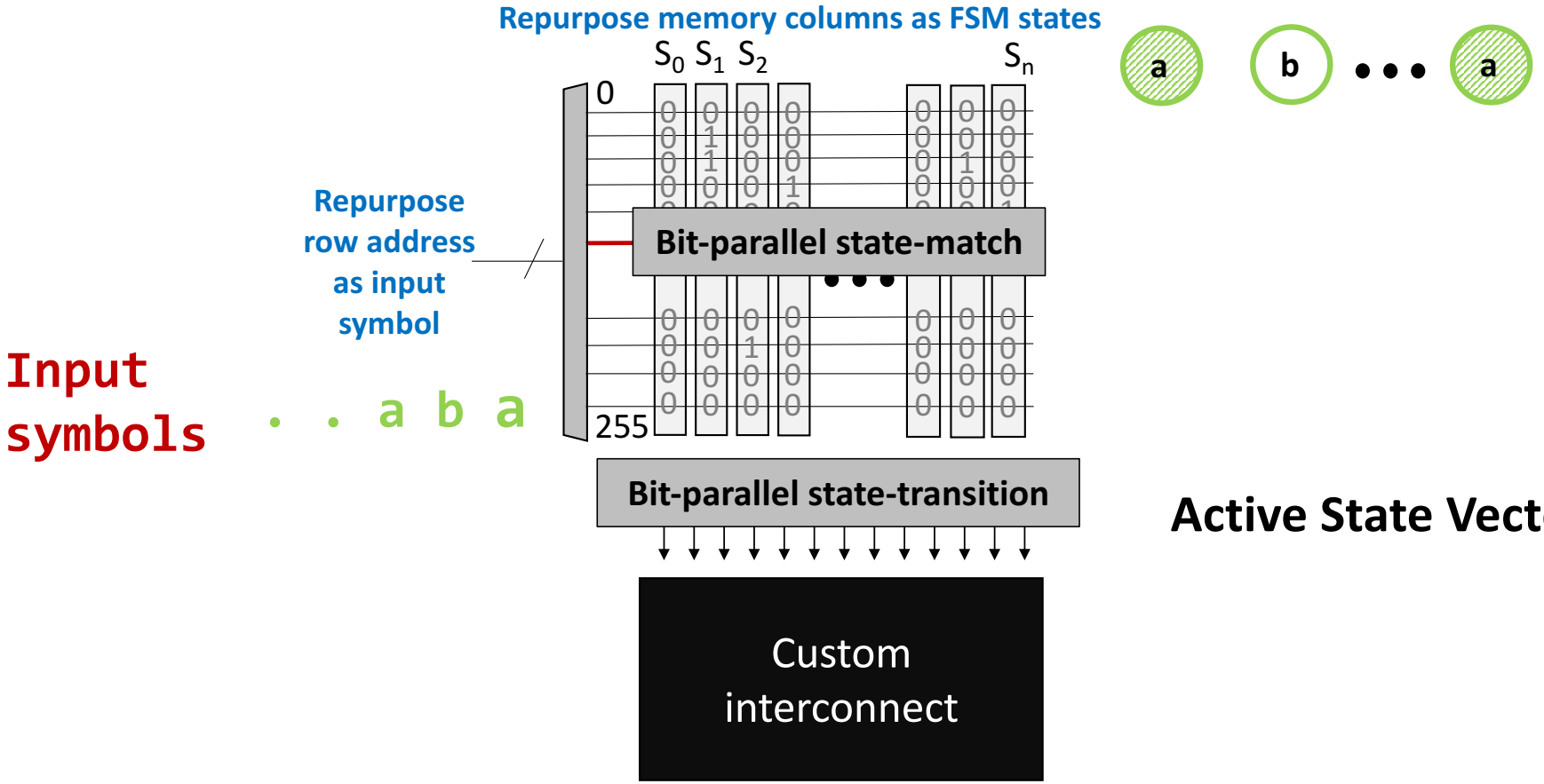


1. **State-Match**
2. **State-Transition**

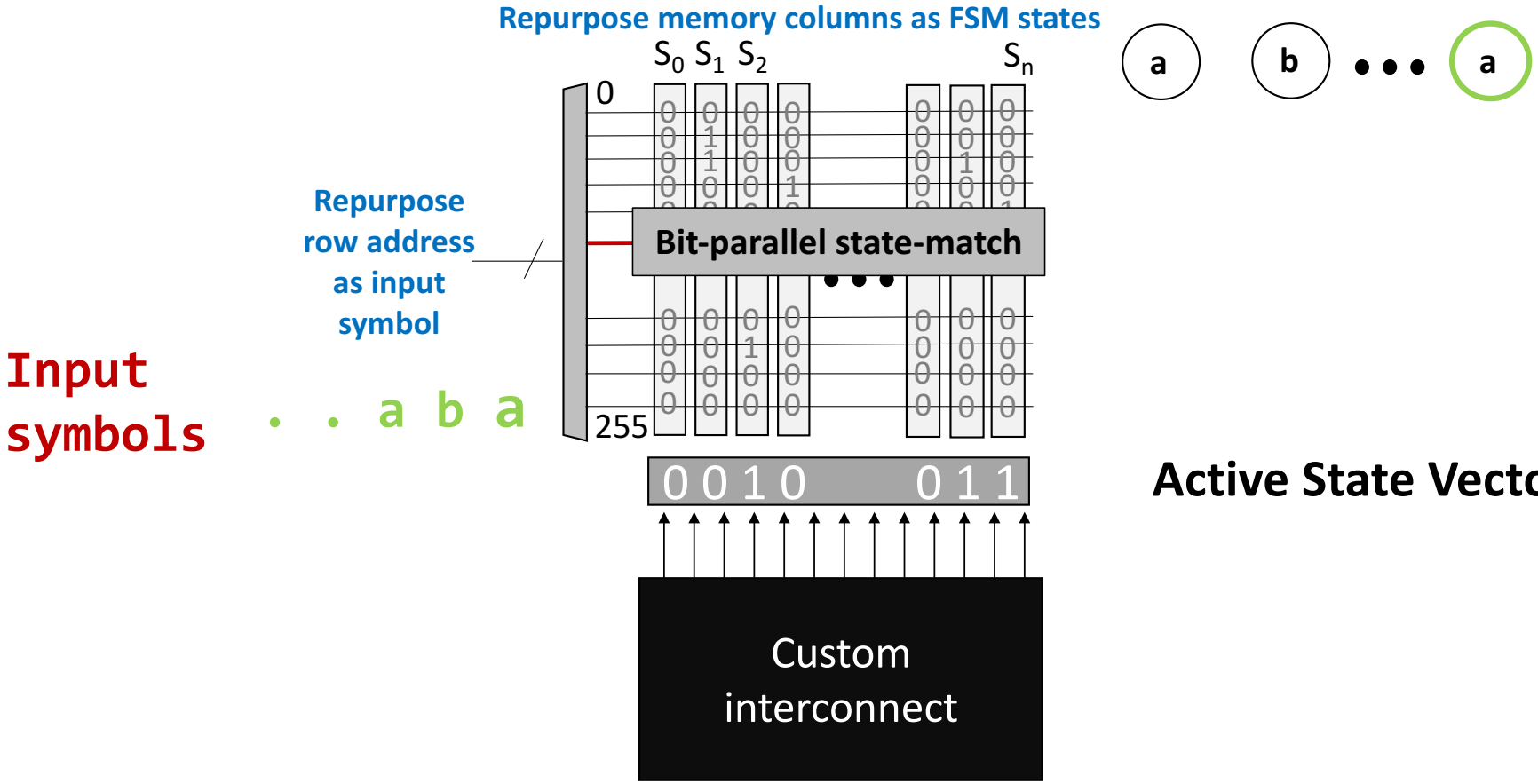
Massively Parallel State-Transition



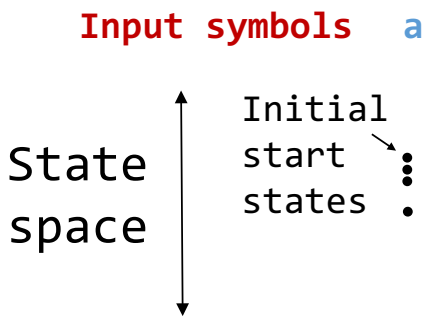
Massively Parallel State-Transition



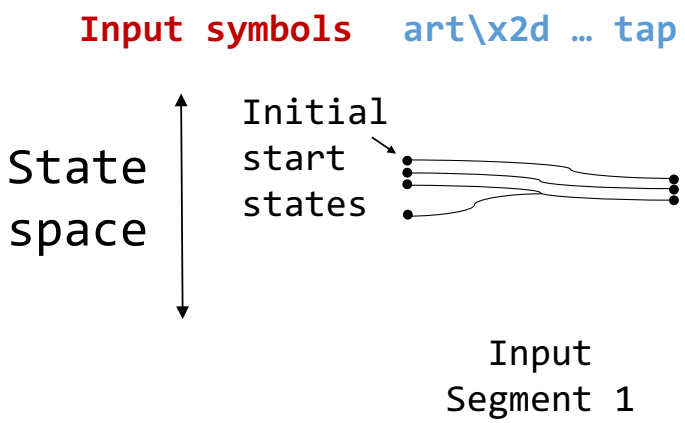
Massively Parallel State-Transition



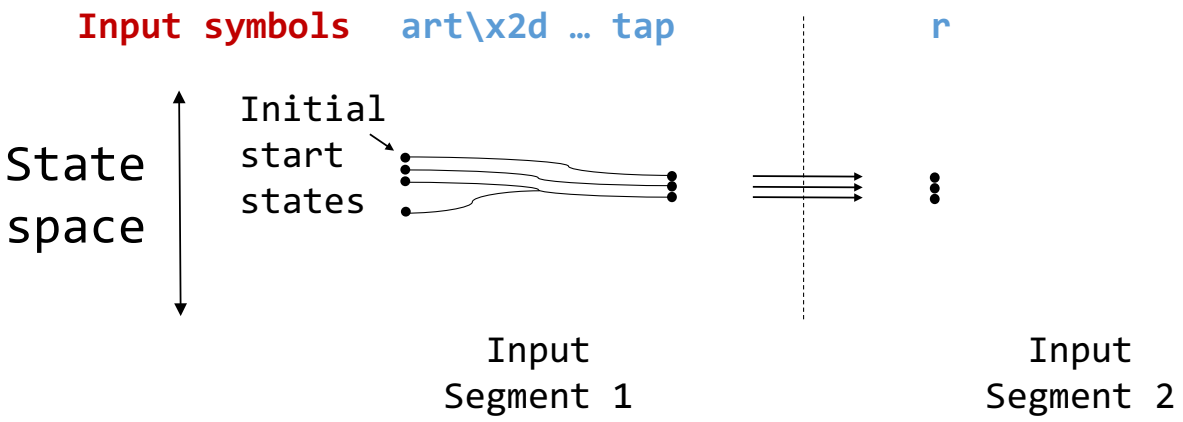
But sequential processing of input symbols …



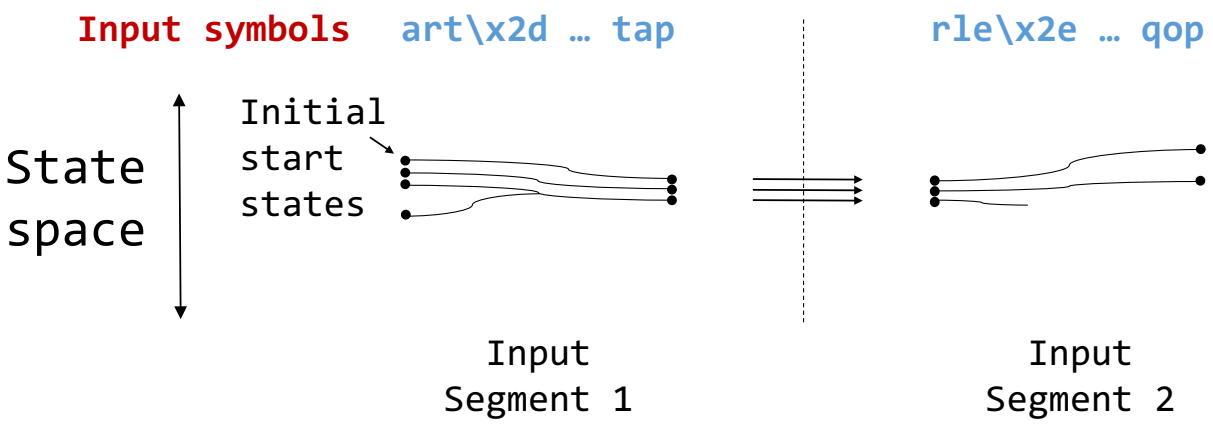
But sequential processing of input symbols …



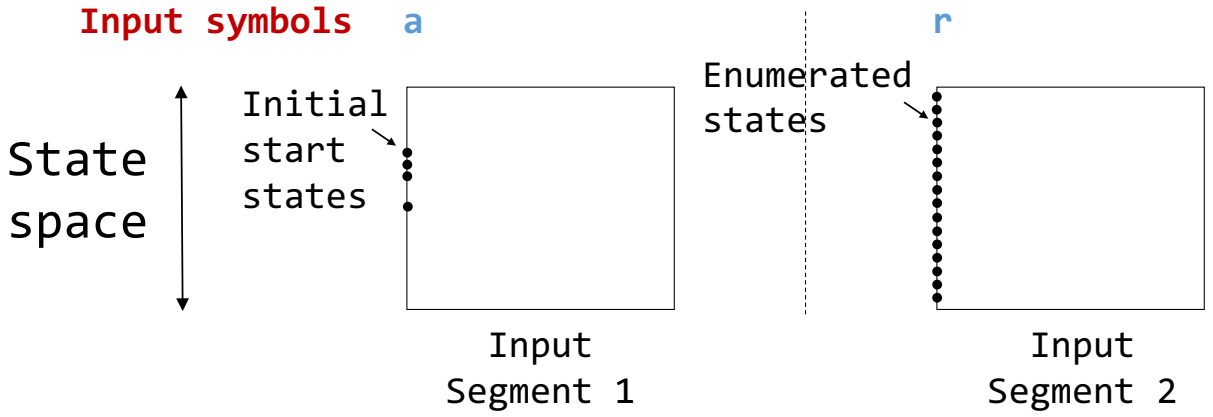
But sequential processing of input symbols …



But sequential processing of input symbols …



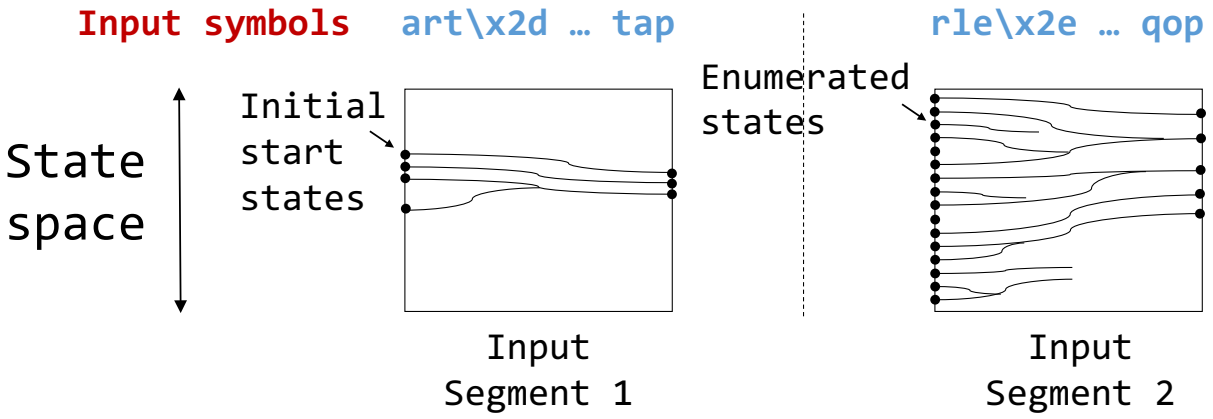
Parallel ~~Sequential~~ Automata Processing



Enumerative Parallelization

[Mytkowicz et al.@ASPLOS'14]

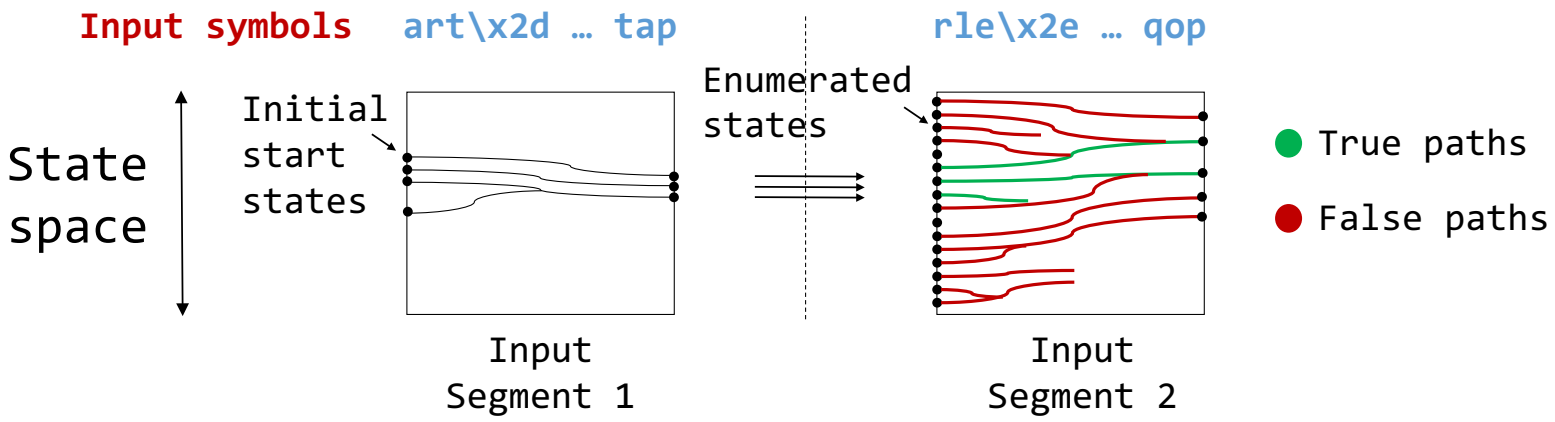
Parallel ~~Sequential~~ Automata Processing



Enumerative Parallelization

[Mytkowicz et al.@ASPLOS'14]

Parallel ~~Sequential~~ Automata Processing



Enumerative Parallelization

[Mytkowicz et al.@ASPLOS'14]

Outline

- Motivation
- In-Memory Automata Processing
- **Parallel Automata Processing**
- Cache Automaton

Parallel Automata Processor

leverages *algorithmic insights* and *repurposes features* in the AP architecture to realize low-cost enumerative parallel execution

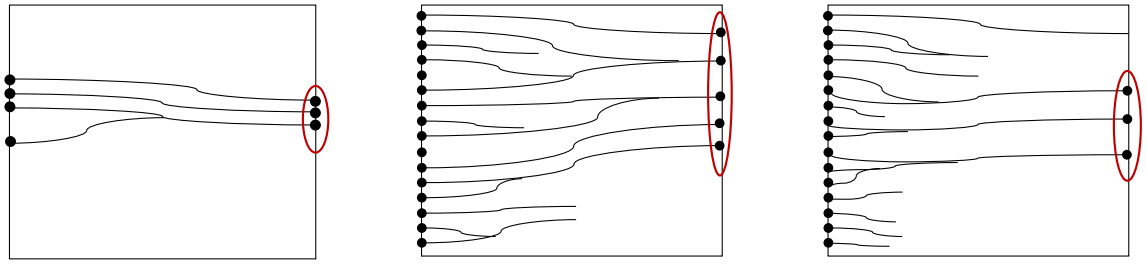
Challenges

Enumeration path tracking

Why do enumeration paths need to be tracked ?

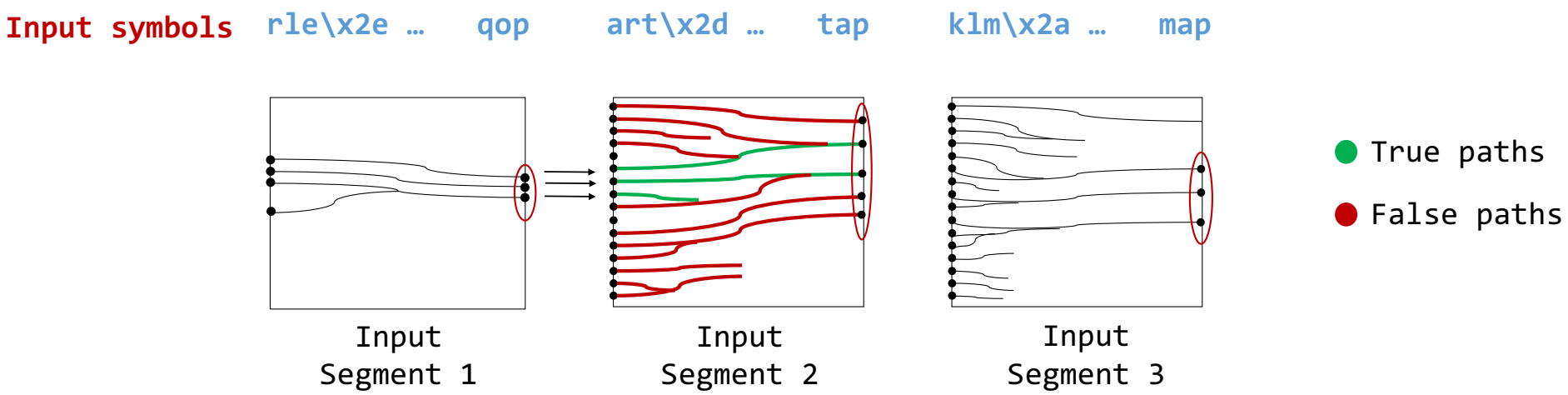
To compose results from input segments …

Input symbols `rle\x2e …` `qop` `art\x2d …` `tap` `klm\x2a …` `map`

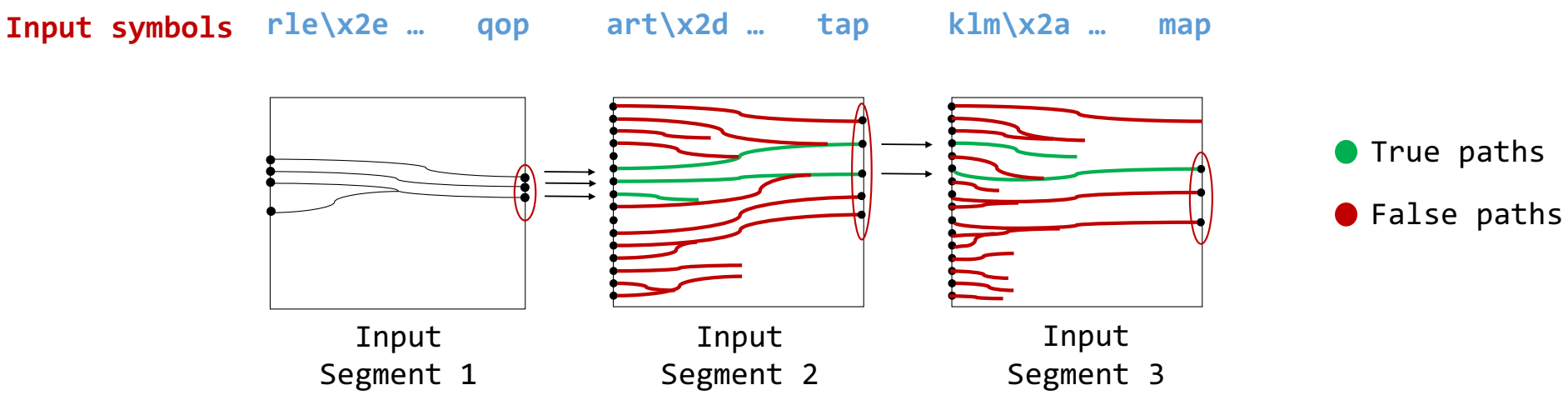


Input Segment 1 | Input Segment 2 | Input Segment 3

To compose results from input segments …



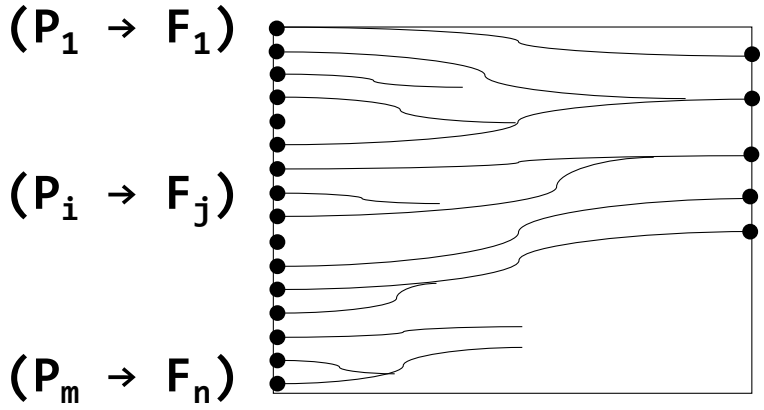
To compose results from input segments …



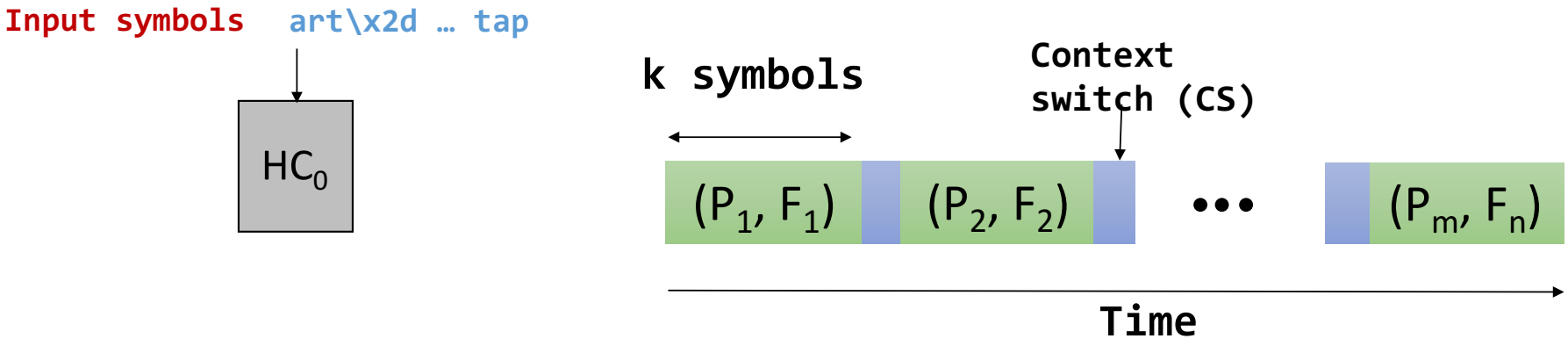
Enumeration path tracking

Enumeration path (P)

AP flow (F)



Execution timeline

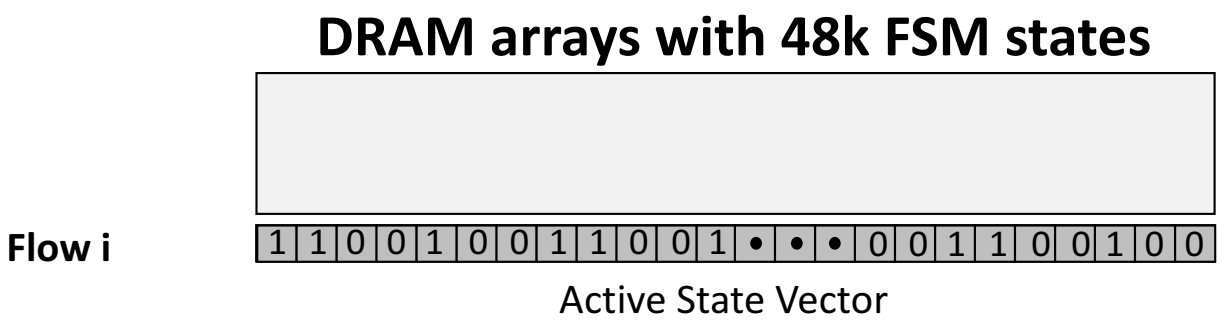


To process k symbols: kn cycles

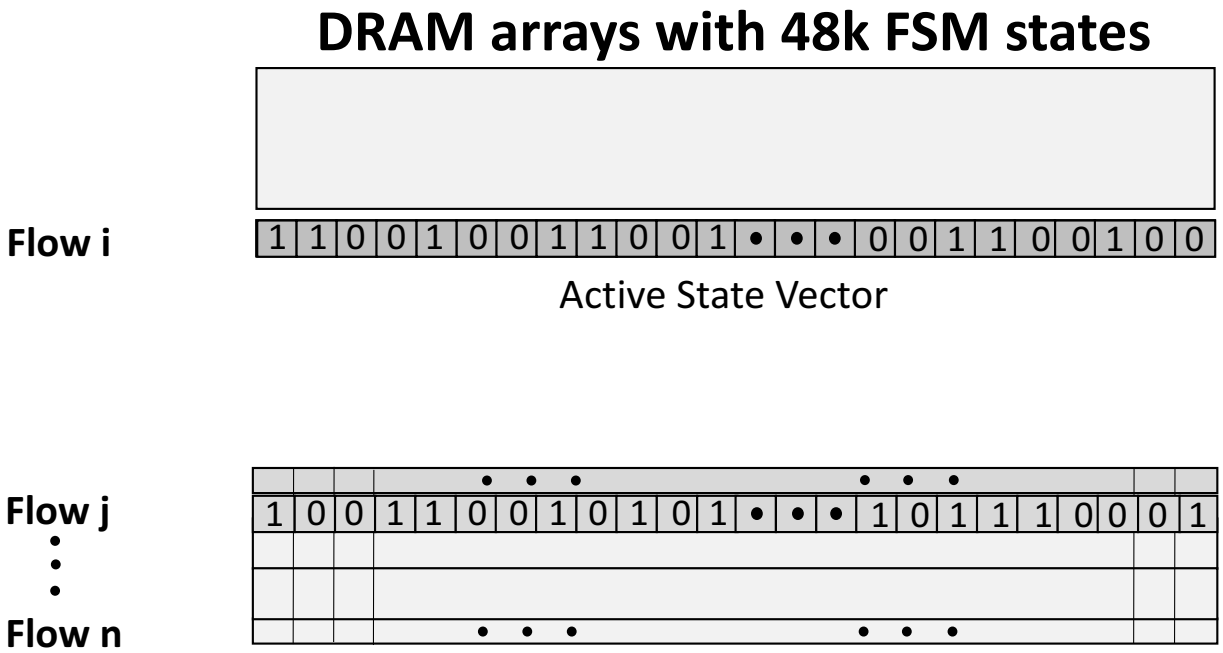
+

(n-1) * context switch cycles

Context switching

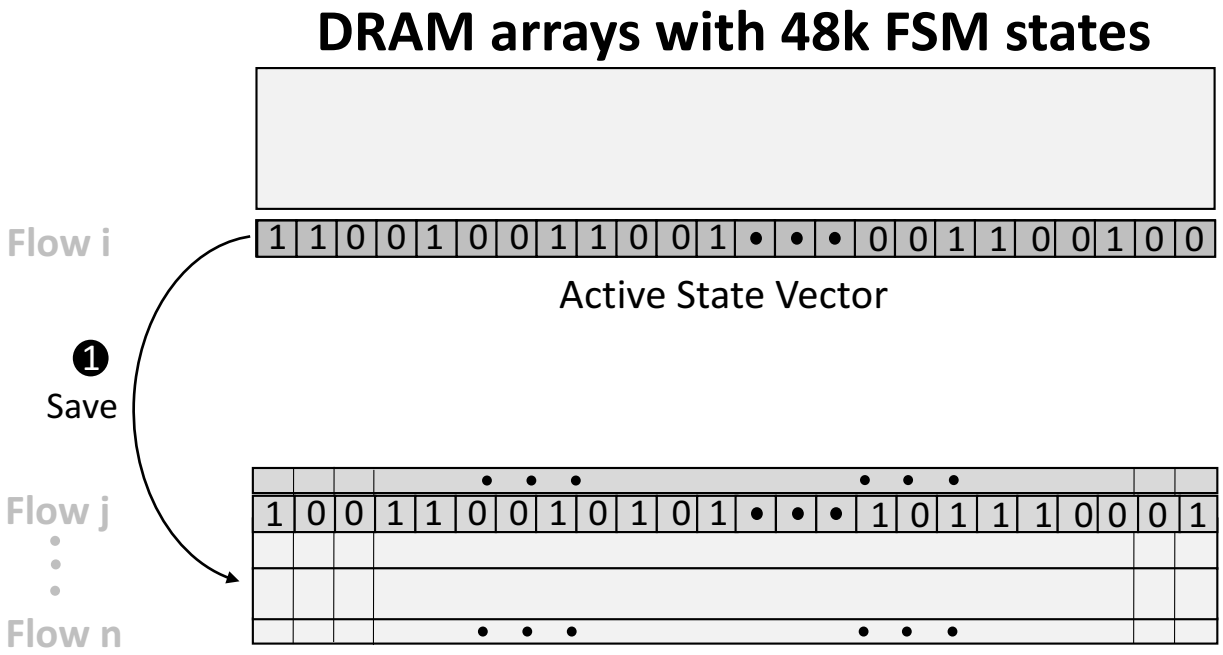


Context switching



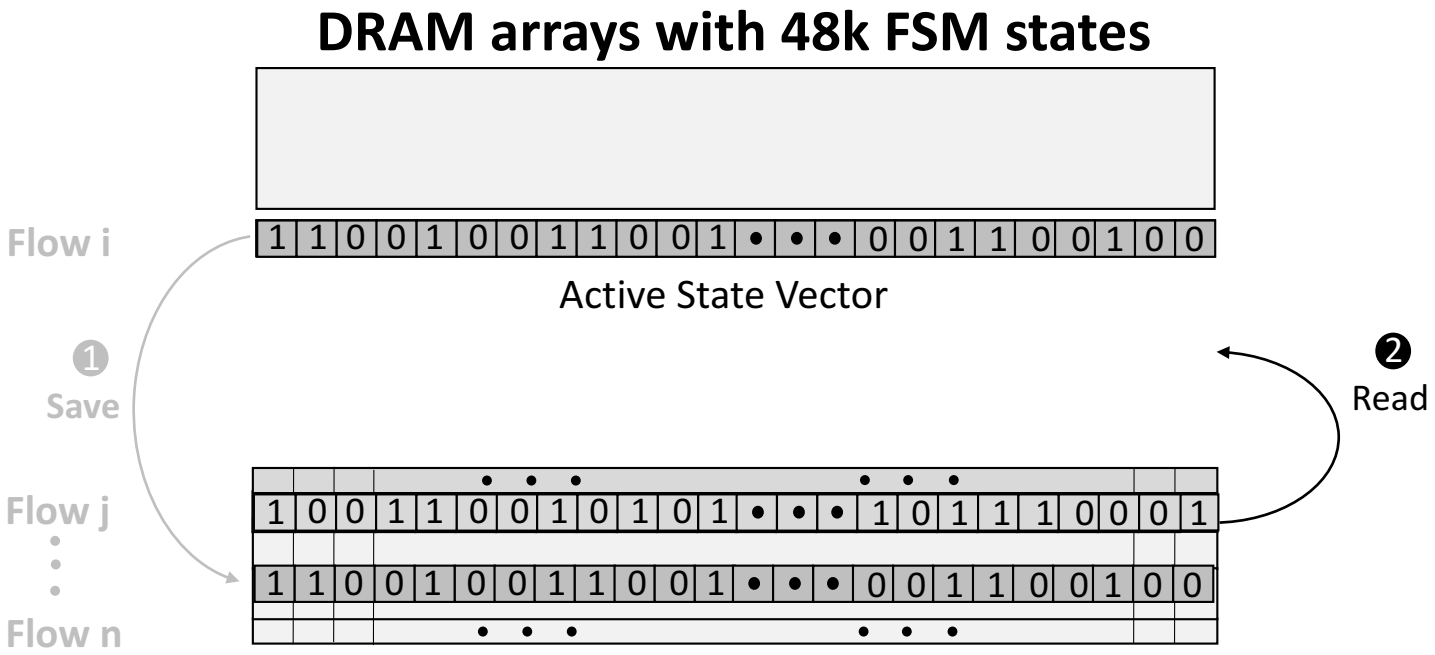
State Vector Cache (512 entries per chip)

Context switching



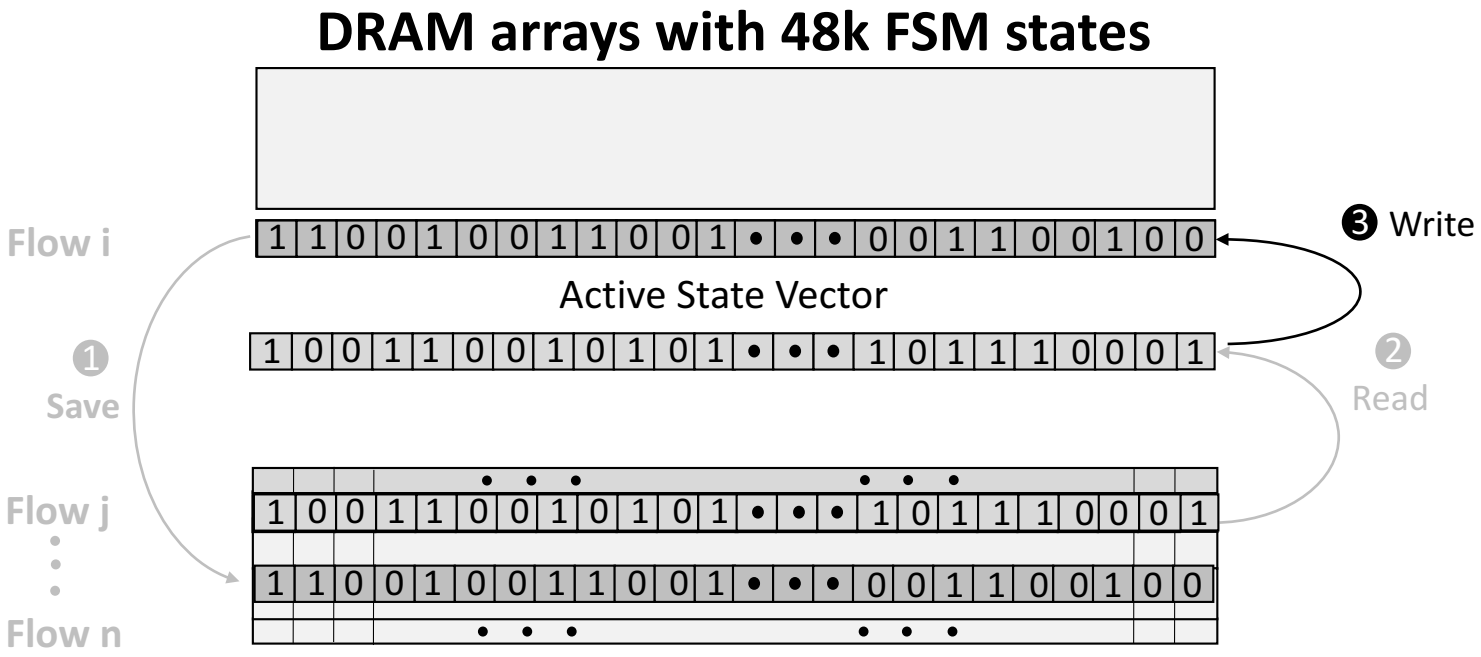
State Vector Cache (512 entries per chip)

Context switching



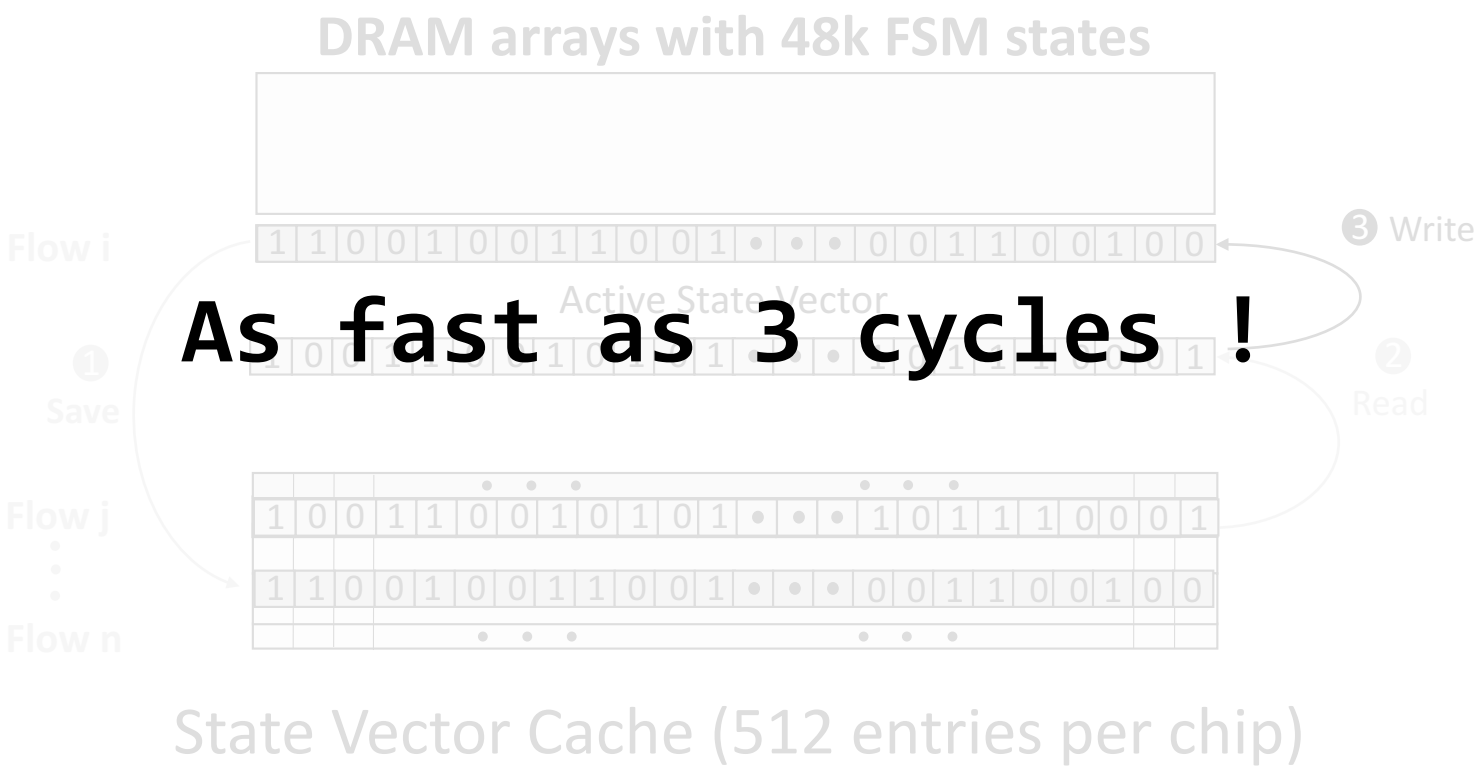
State Vector Cache (512 entries per chip)

Context switching



State Vector Cache (512 entries per chip)

Context switching



As fast as 3 cycles !

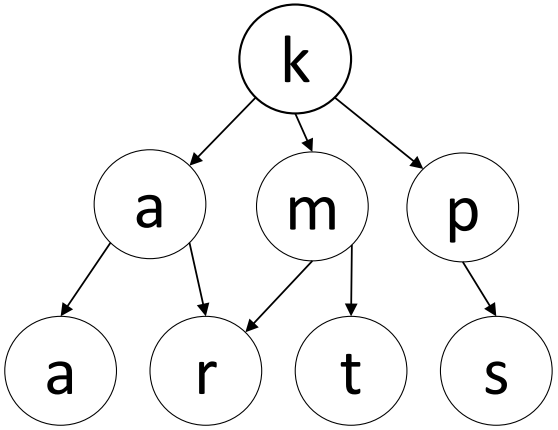
Challenges

Enumeration path tracking

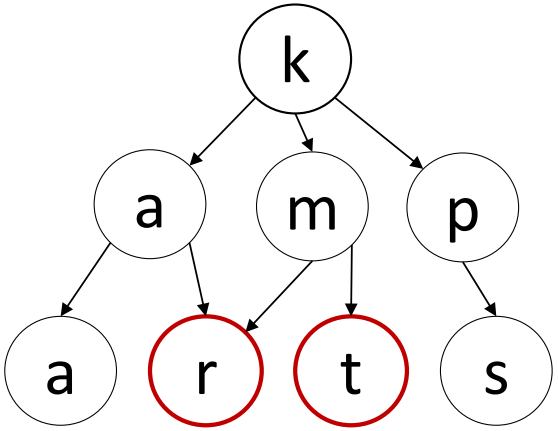
Reducing enumeration complexity

1. Reducing number of flows

Reducing number of flows



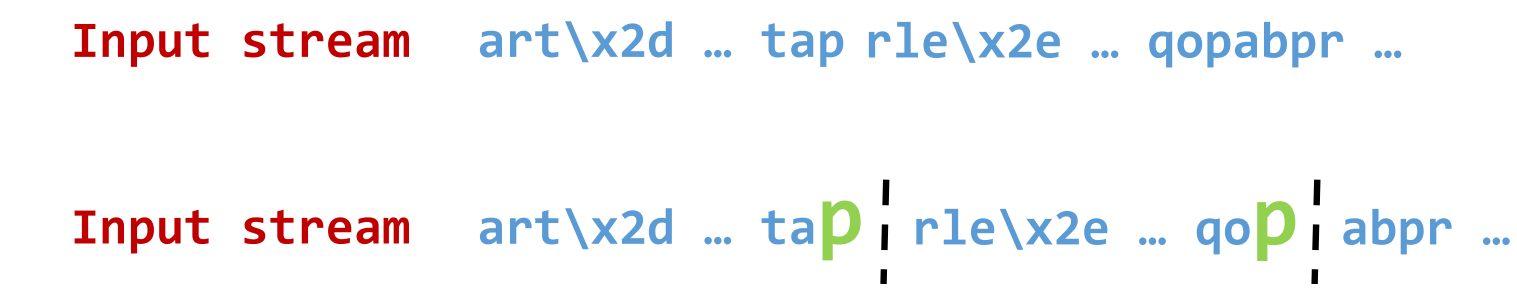
Reducing number of flows



Range { m } = 2

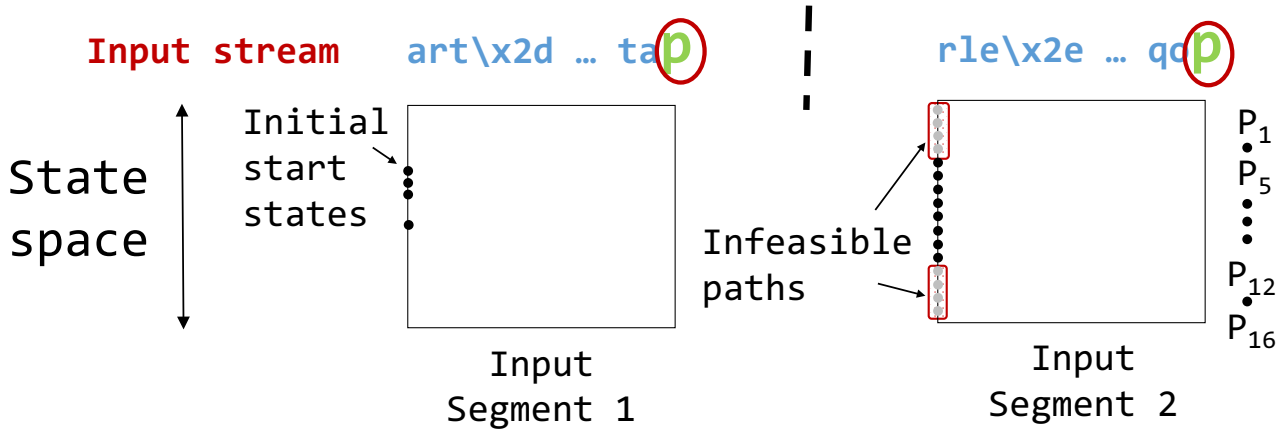
Range for many input symbols typically much smaller than states in FSM

Range-Guided Input Partitioning



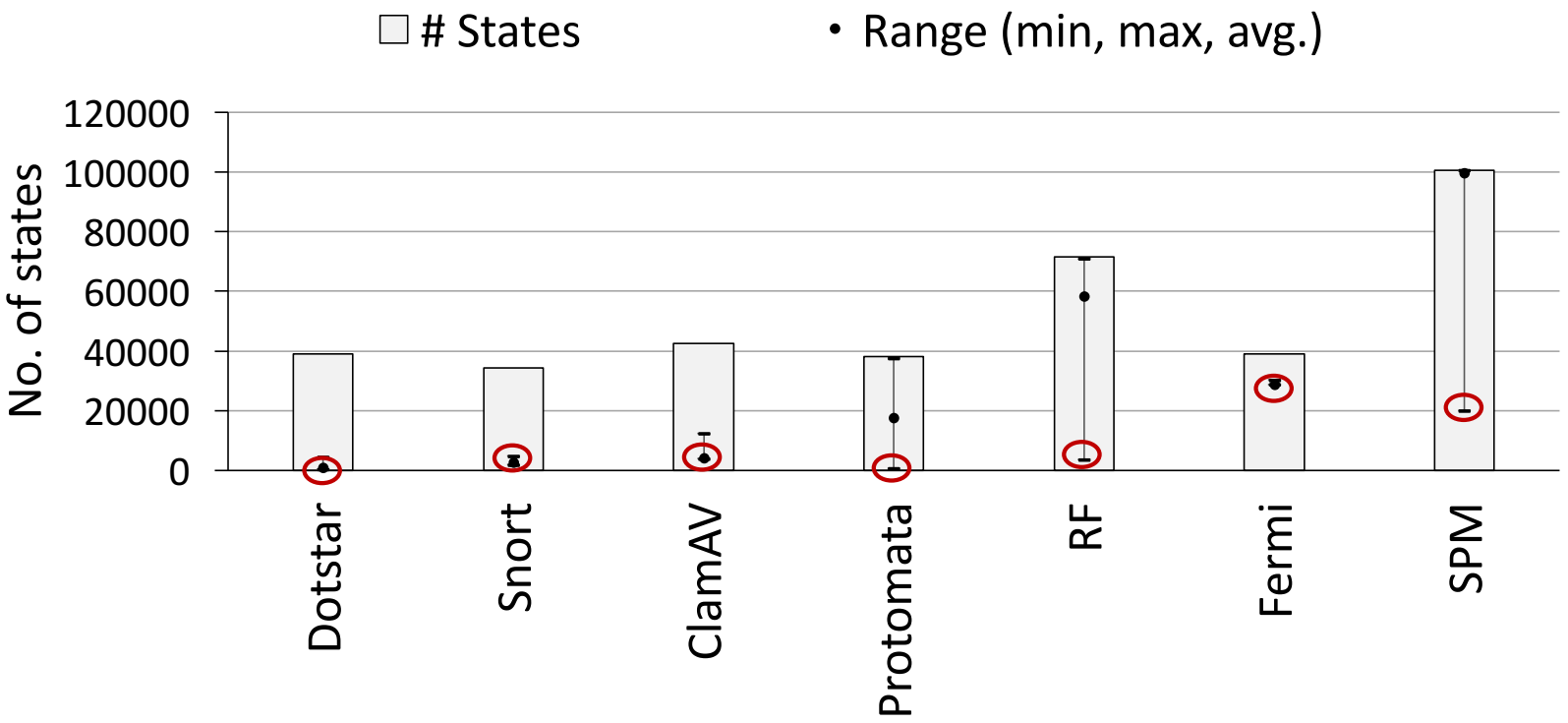
p – minimum range, frequently occurring symbol

Range-Guided Input Partitioning



Only P_5 – P_{12} are feasible enumeration paths

Minimum range much smaller than state space



Challenges

Enumeration path tracking

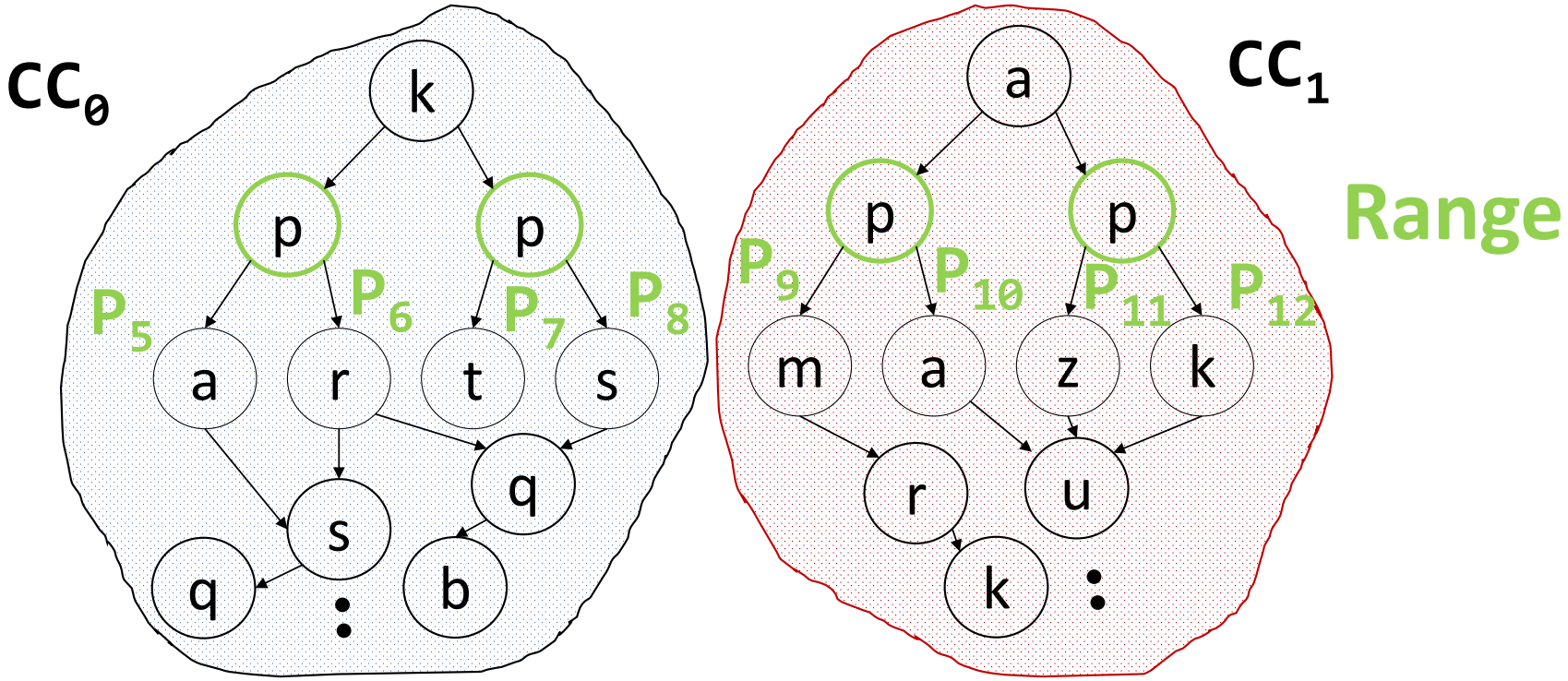
Reducing enumeration complexity

1. Reducing number of flows
2. Improving flow utilization

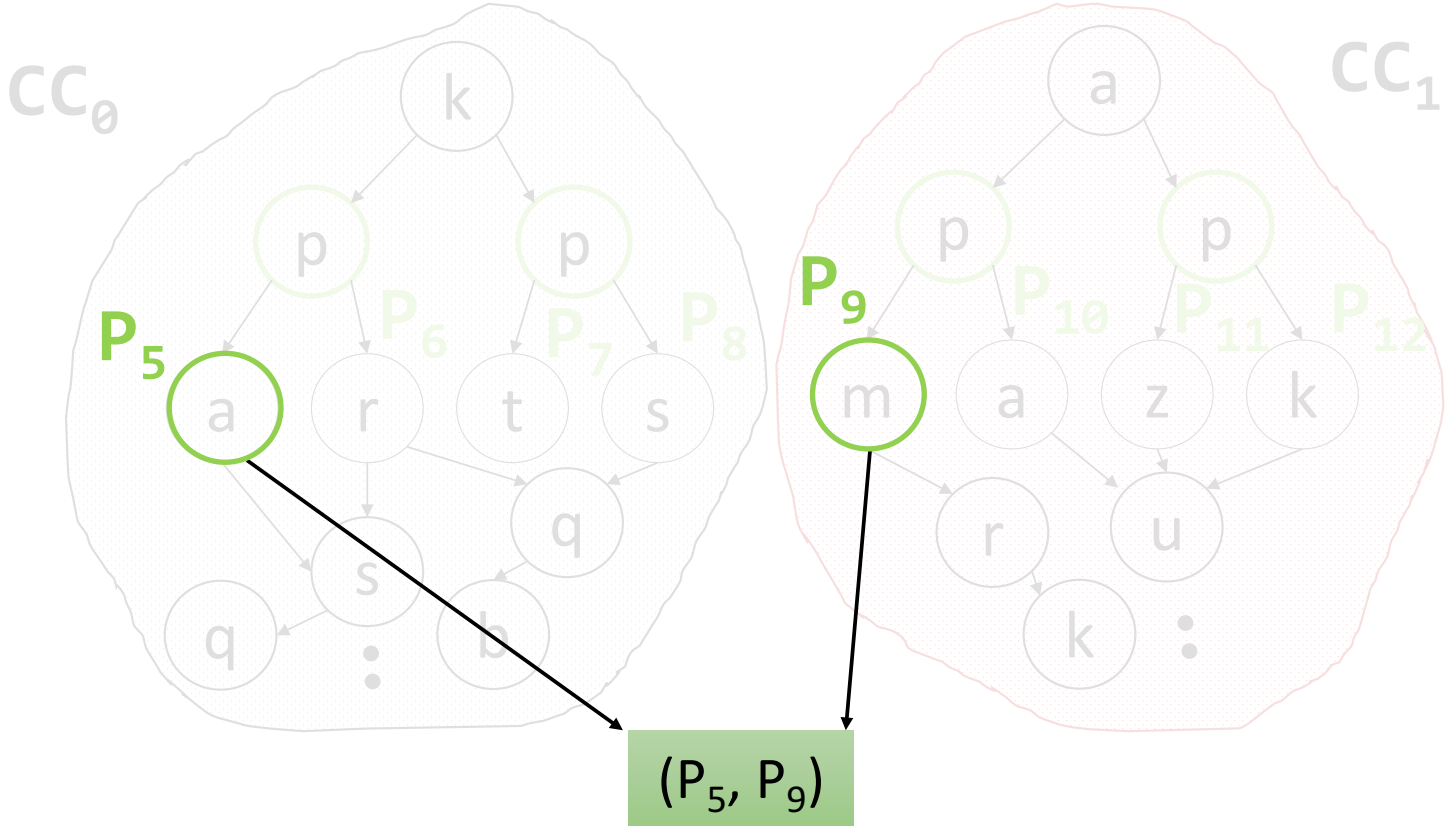
Improving flow utilization

Can we enumerate more states in the same flow ?

Leveraging connected components in FSMs



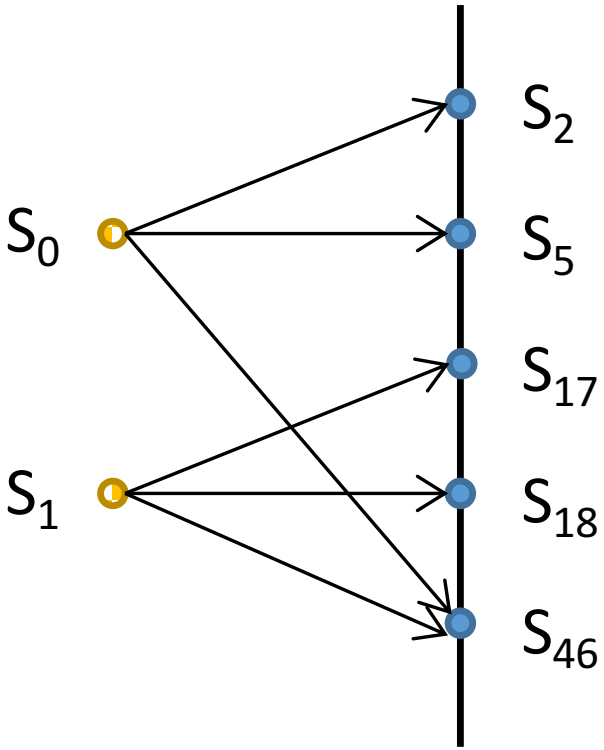
Leveraging connected components in FSMs



Reducing number of flows

What about enumerated states with common parents ?

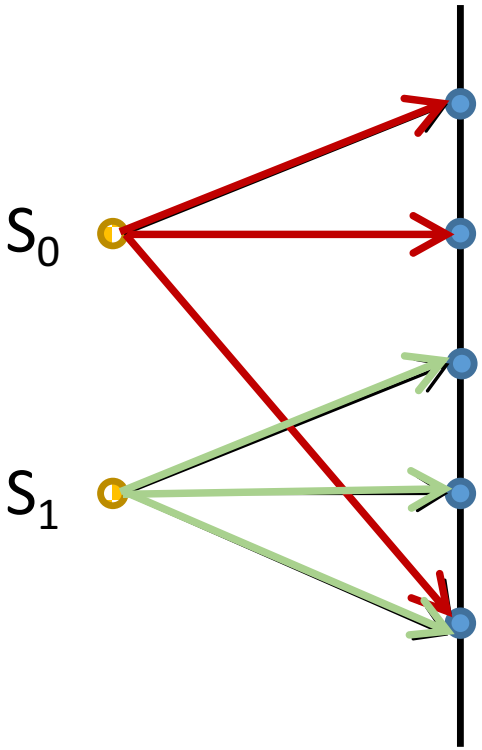
Common Parent Merging



5 flows

Enumerated start states

Common Parent Merging



2 flows

$F_1 = \{S_2 , S_5 , S_{46}\}$

$F_2 = \{S_{17}, S_{18} , S_{46}\}$

Enumerated start states

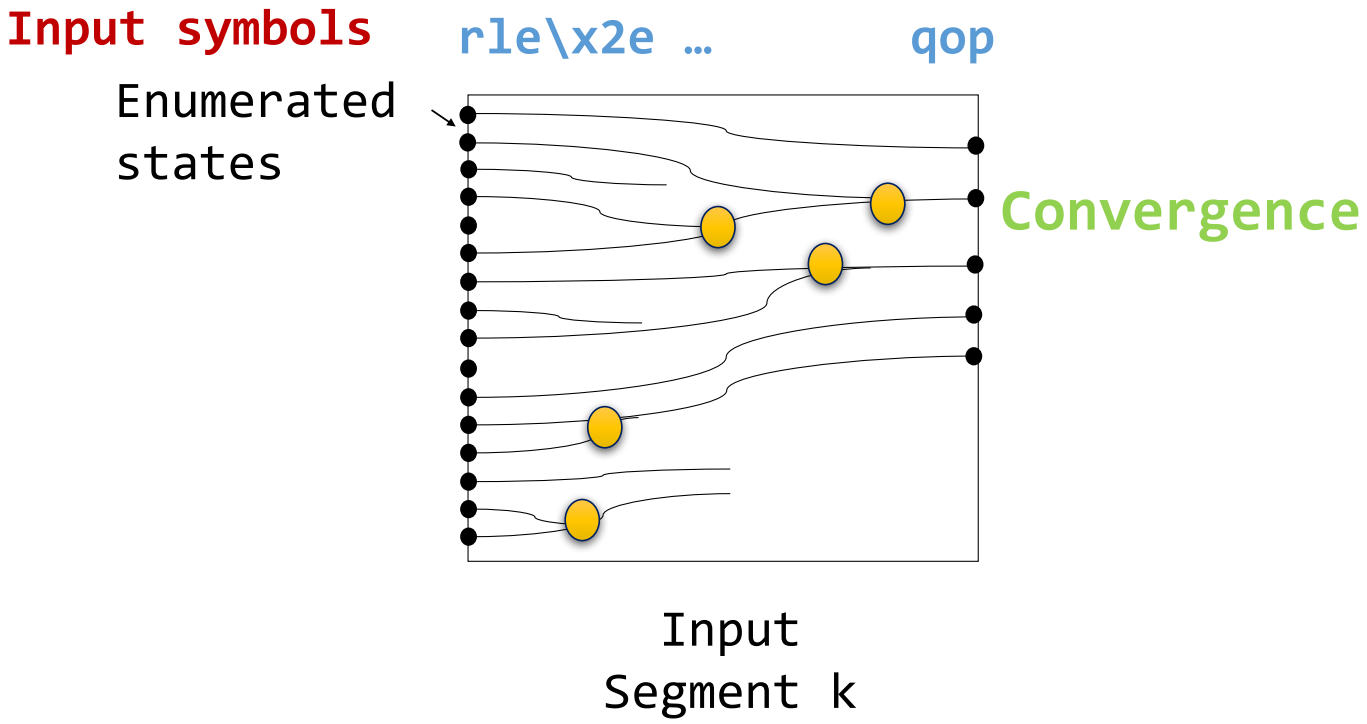
Challenges

Enumeration path tracking

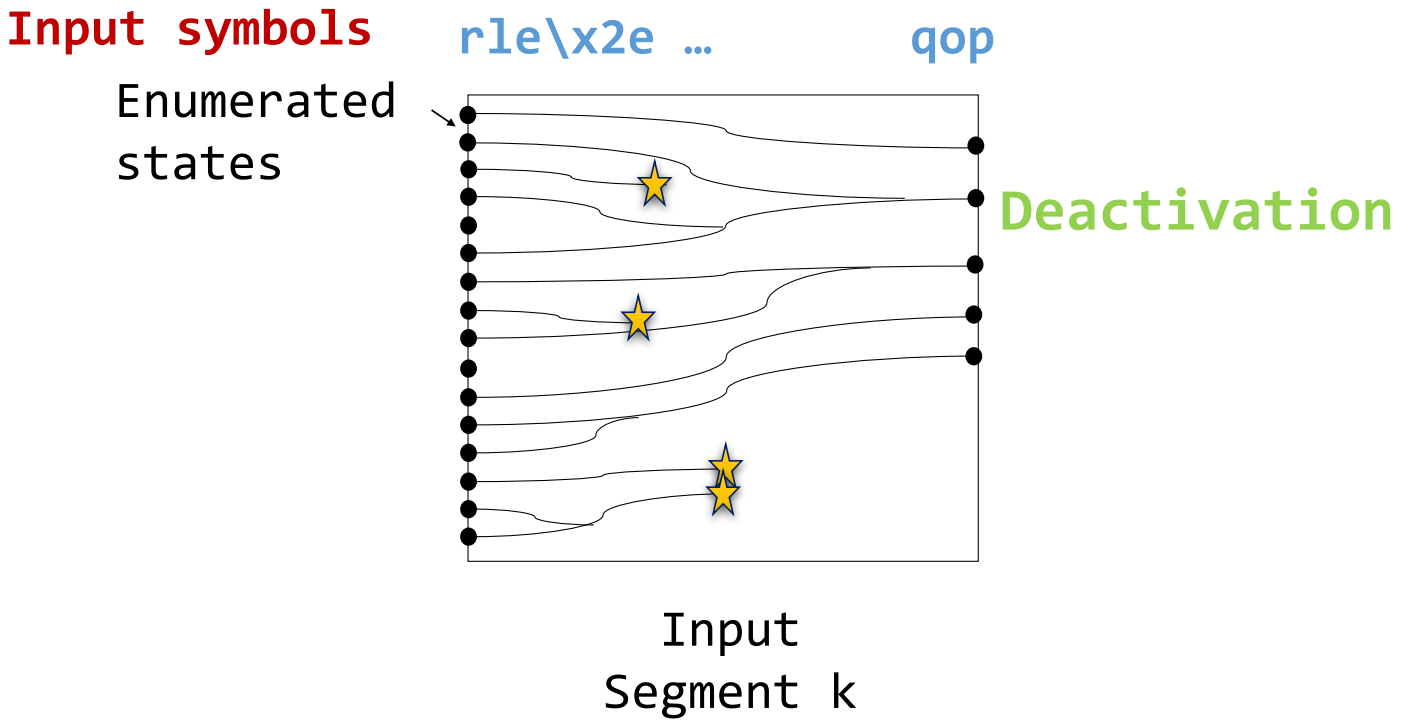
Reducing enumeration complexity

1. Reducing number of flows
2. Improving flow utilization
3. Avoiding redundant work at runtime

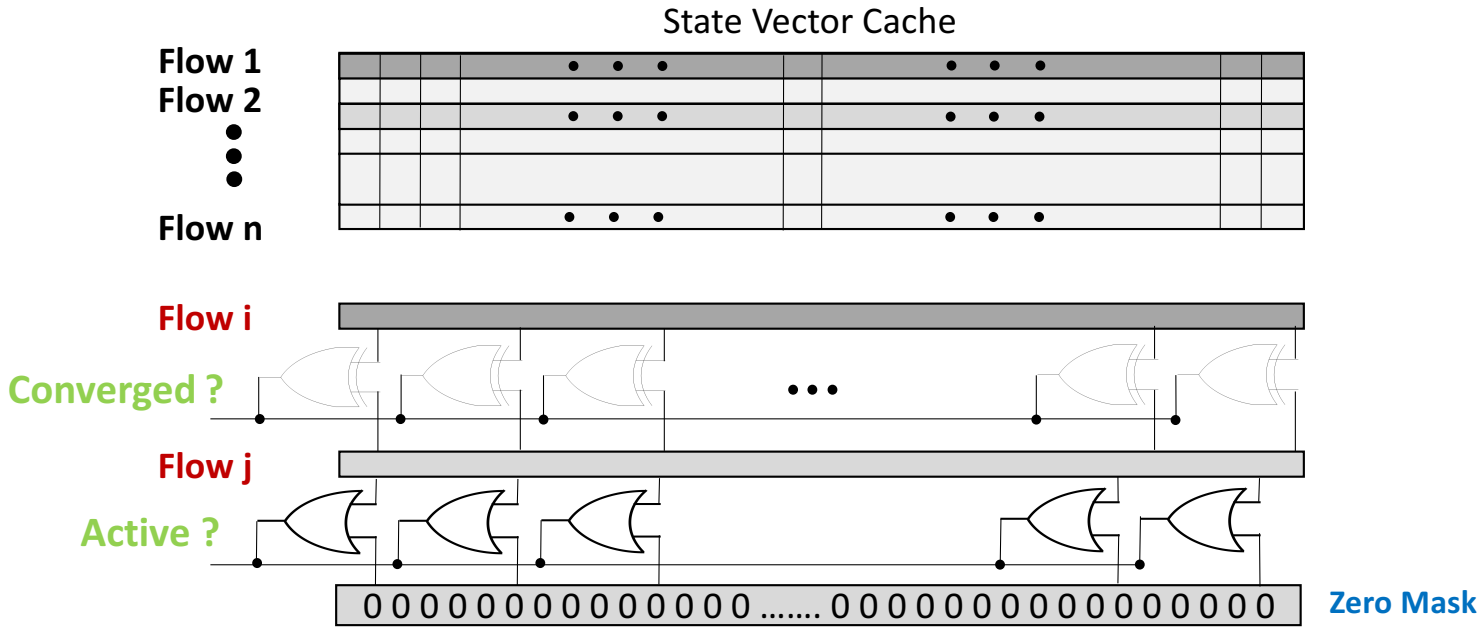
Avoiding redundant work



Avoiding redundant work



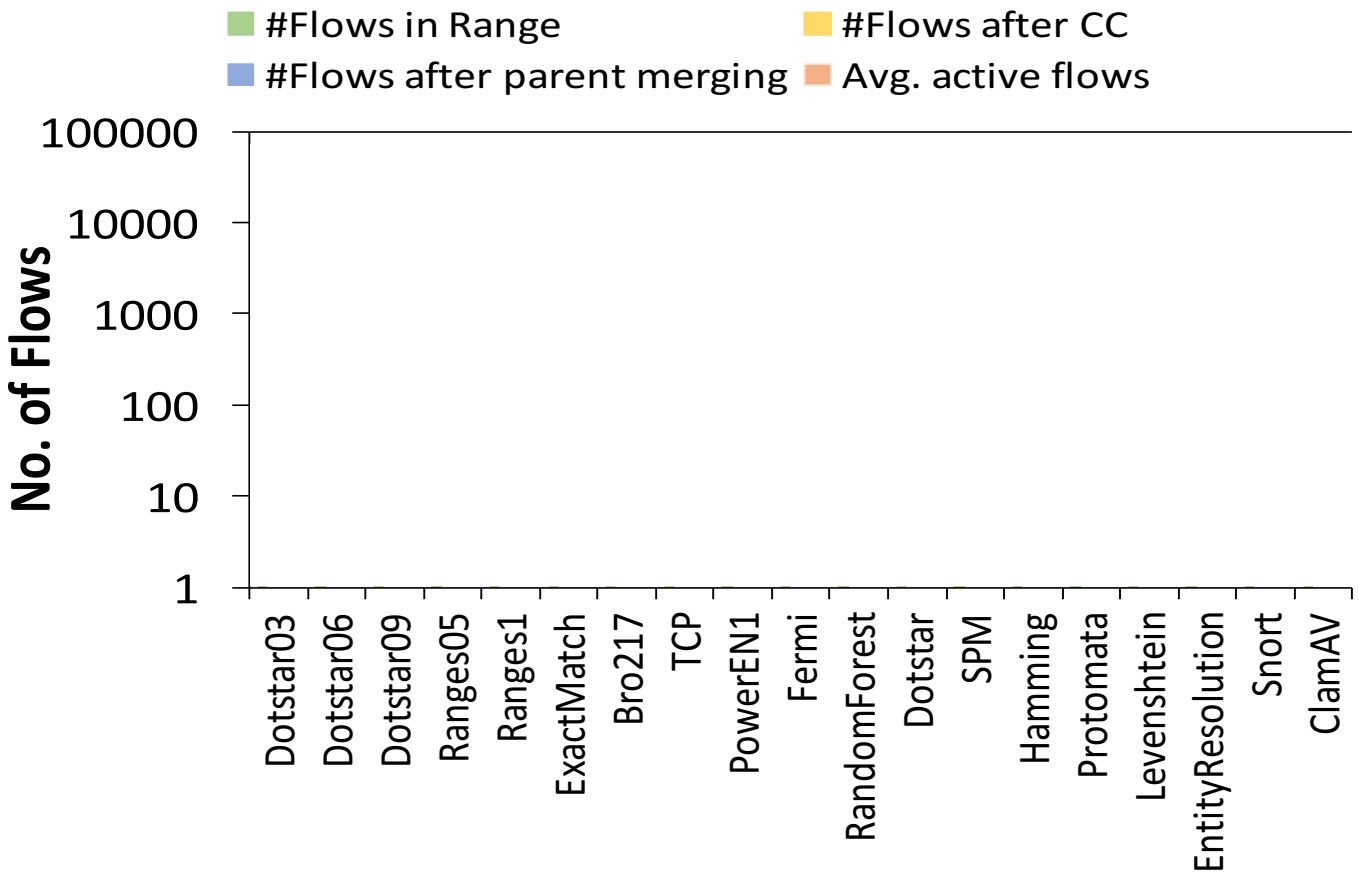
Low-cost runtime checks



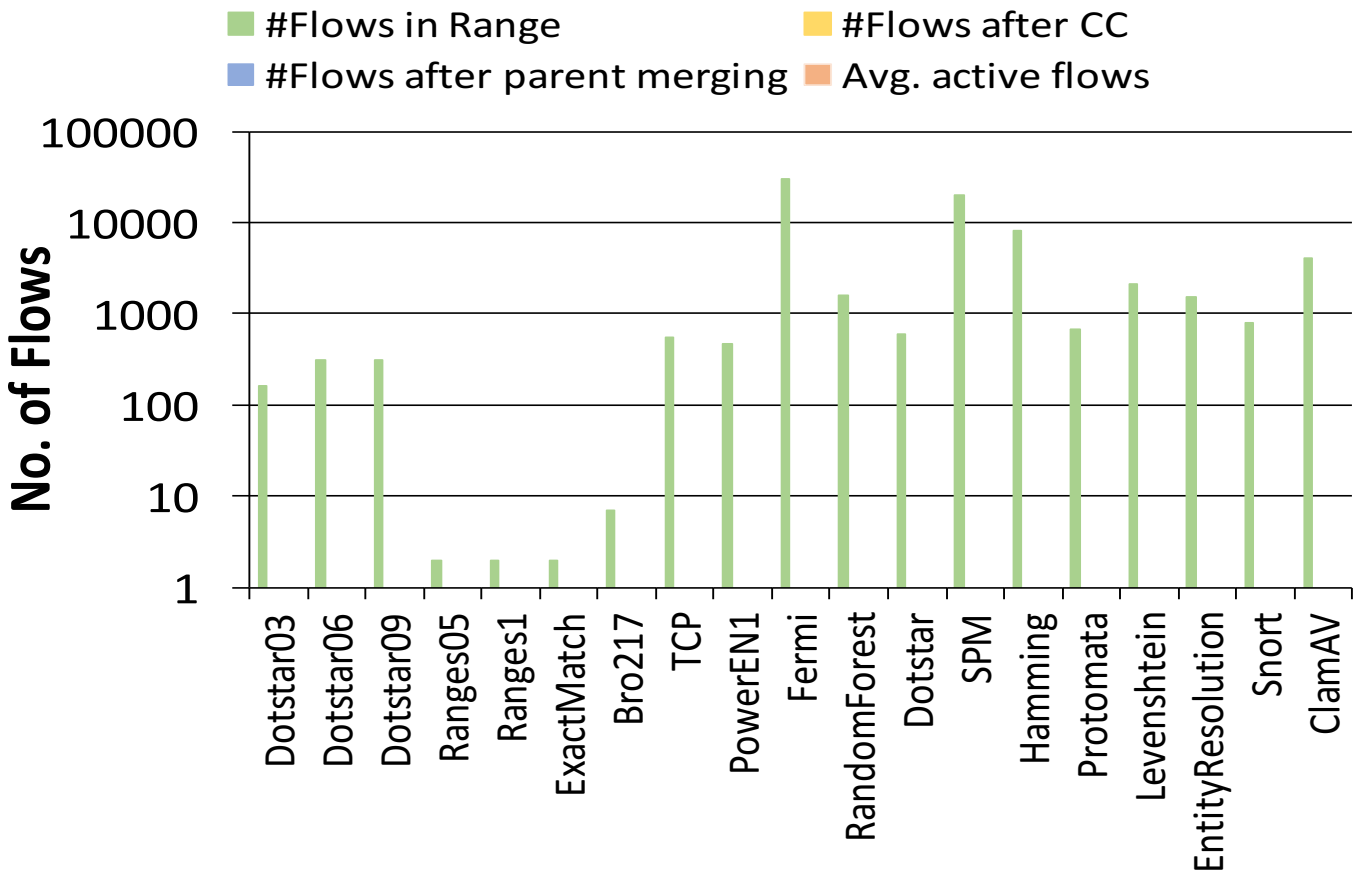
Experimental Methodology

- ✓ Multi-Threaded Virtual Automata Simulator (VASim)
- ✓ Deterministic cycle time for Automata Processor (7.5 ns)
- ✓ Context switch – 3 cycles, Convergence check – overlapped with processing
- ✓ ANMLZoo and Regex benchmark suites, 1/10 MB input data

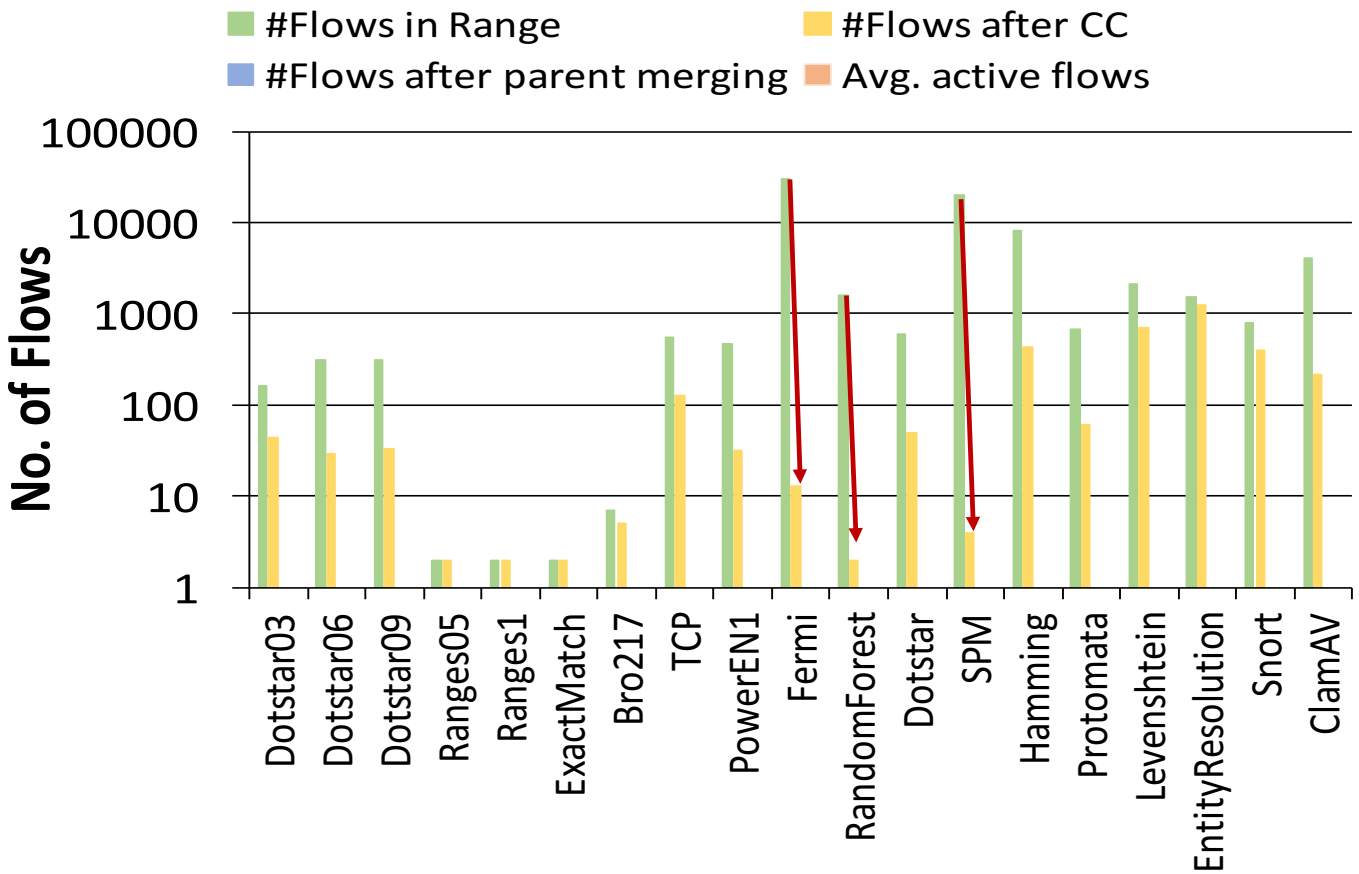
Active flows



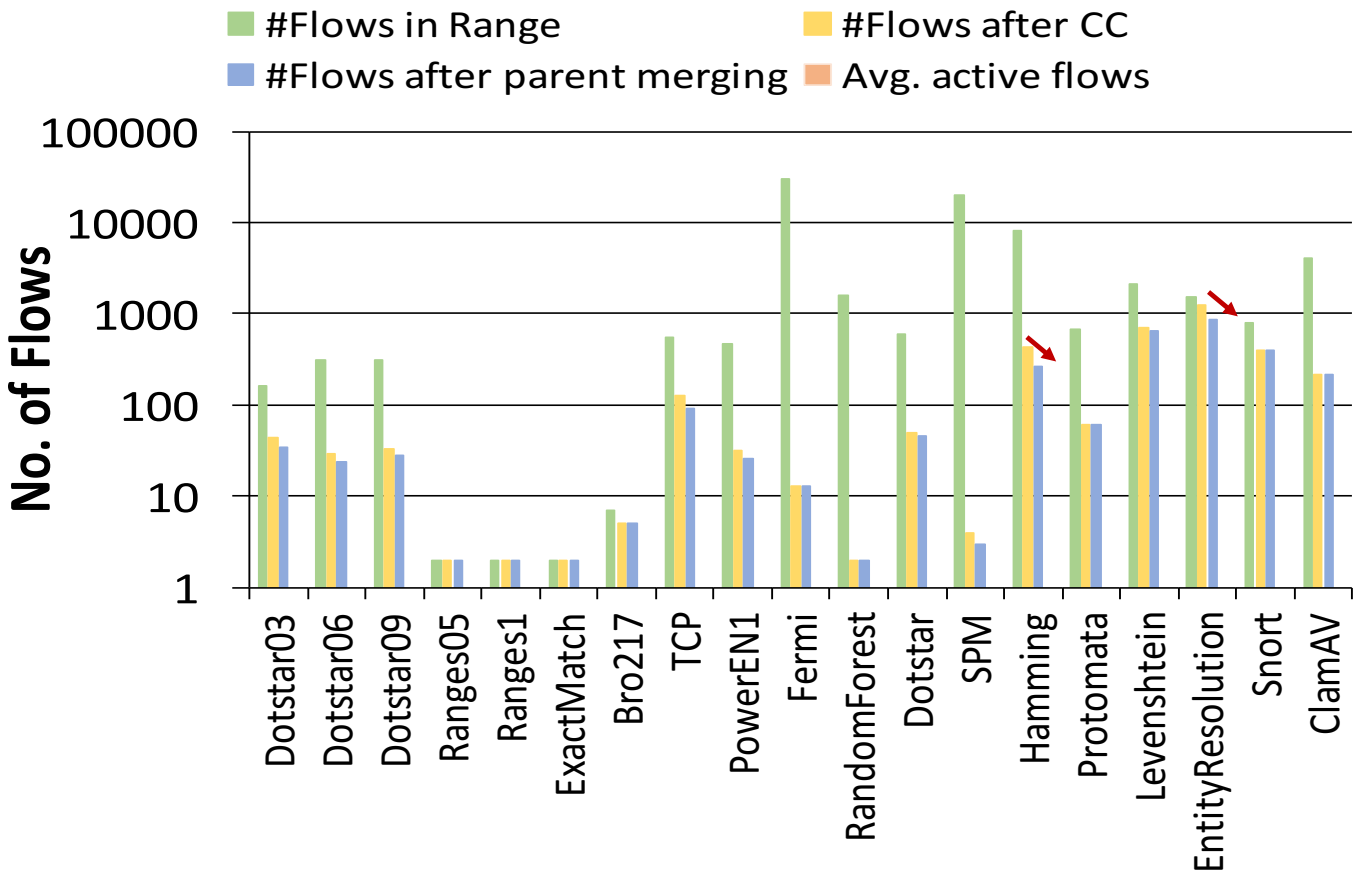
Active flows



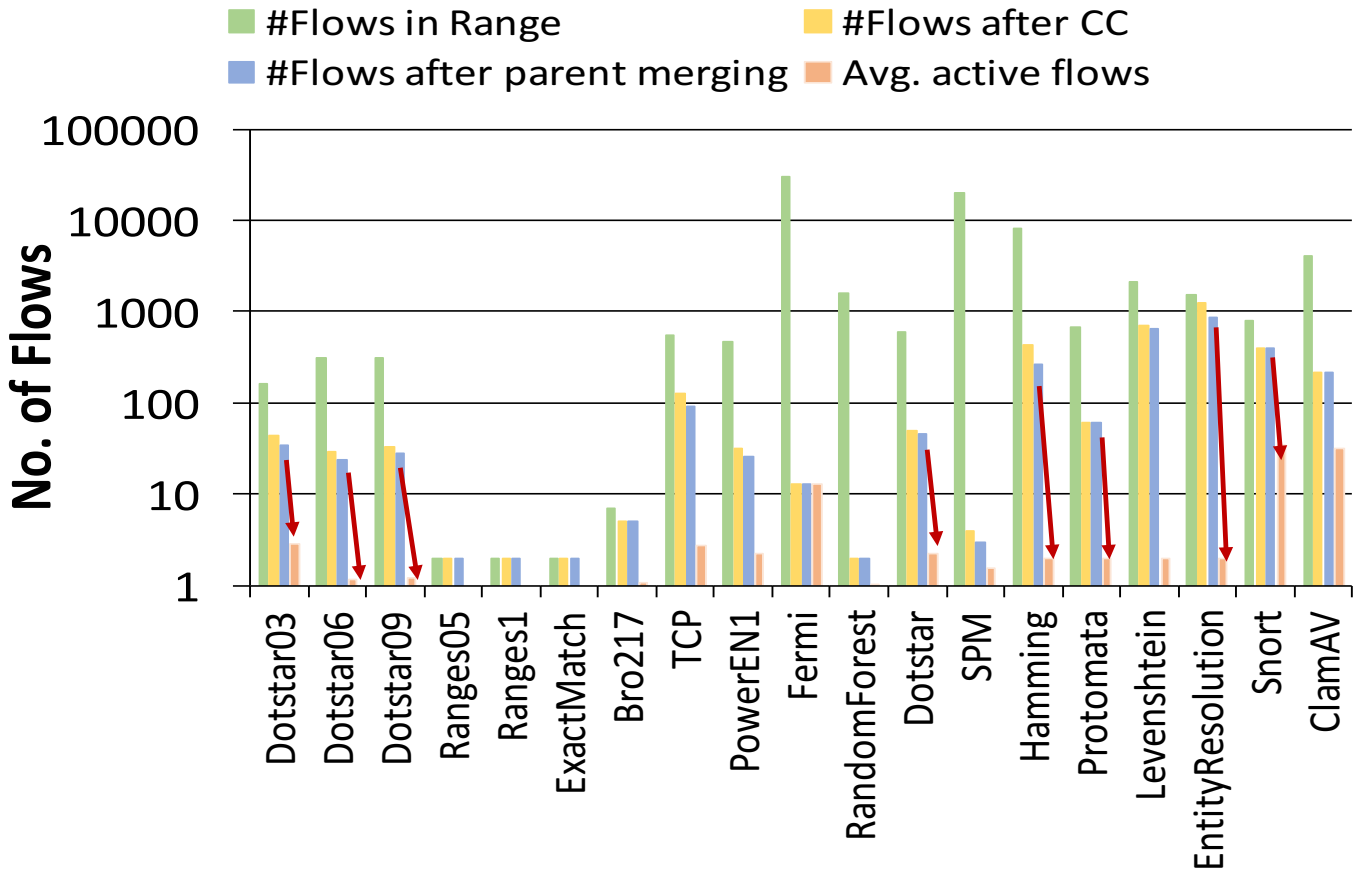
Active flows



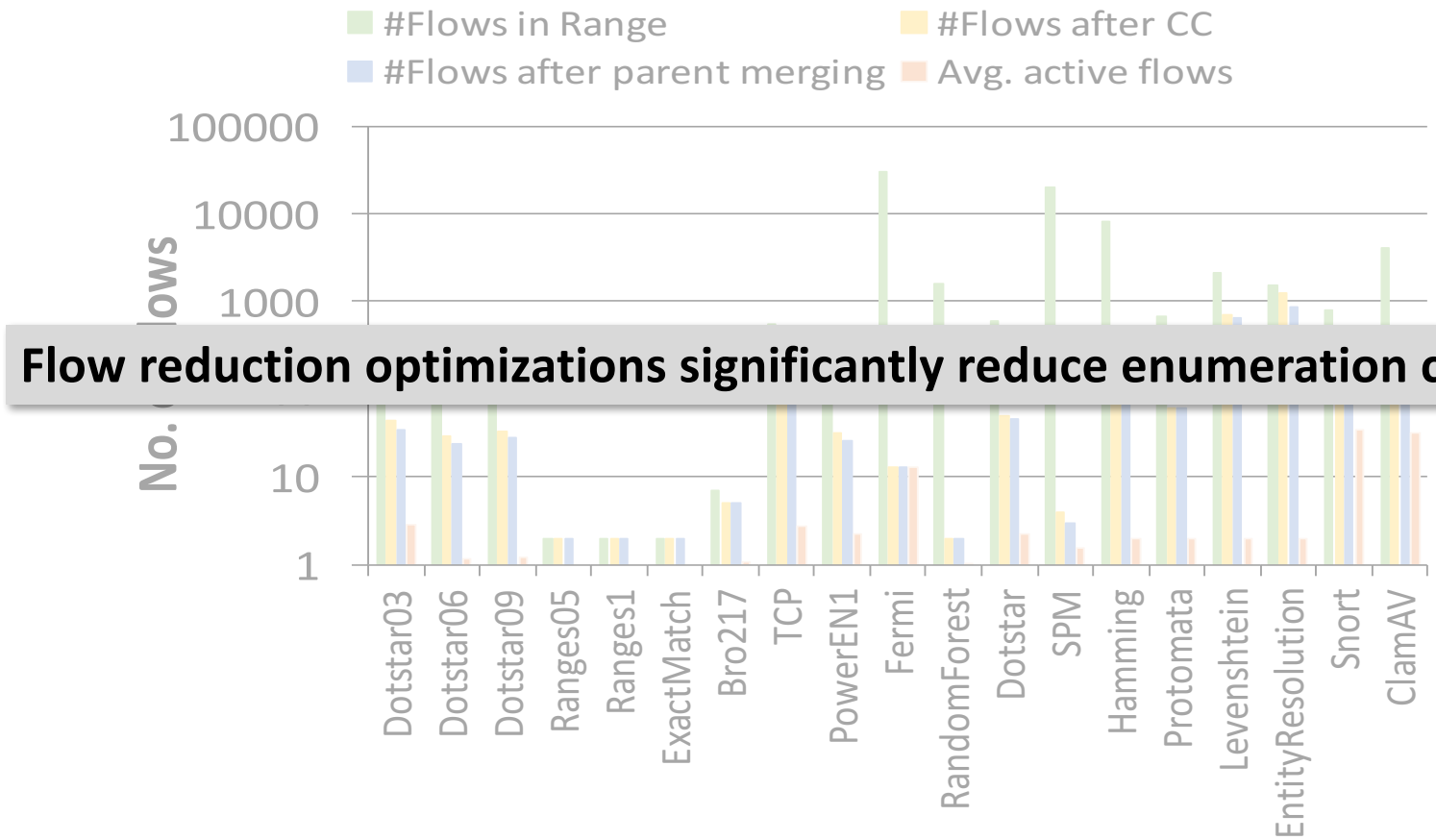
Active flows



Active flows

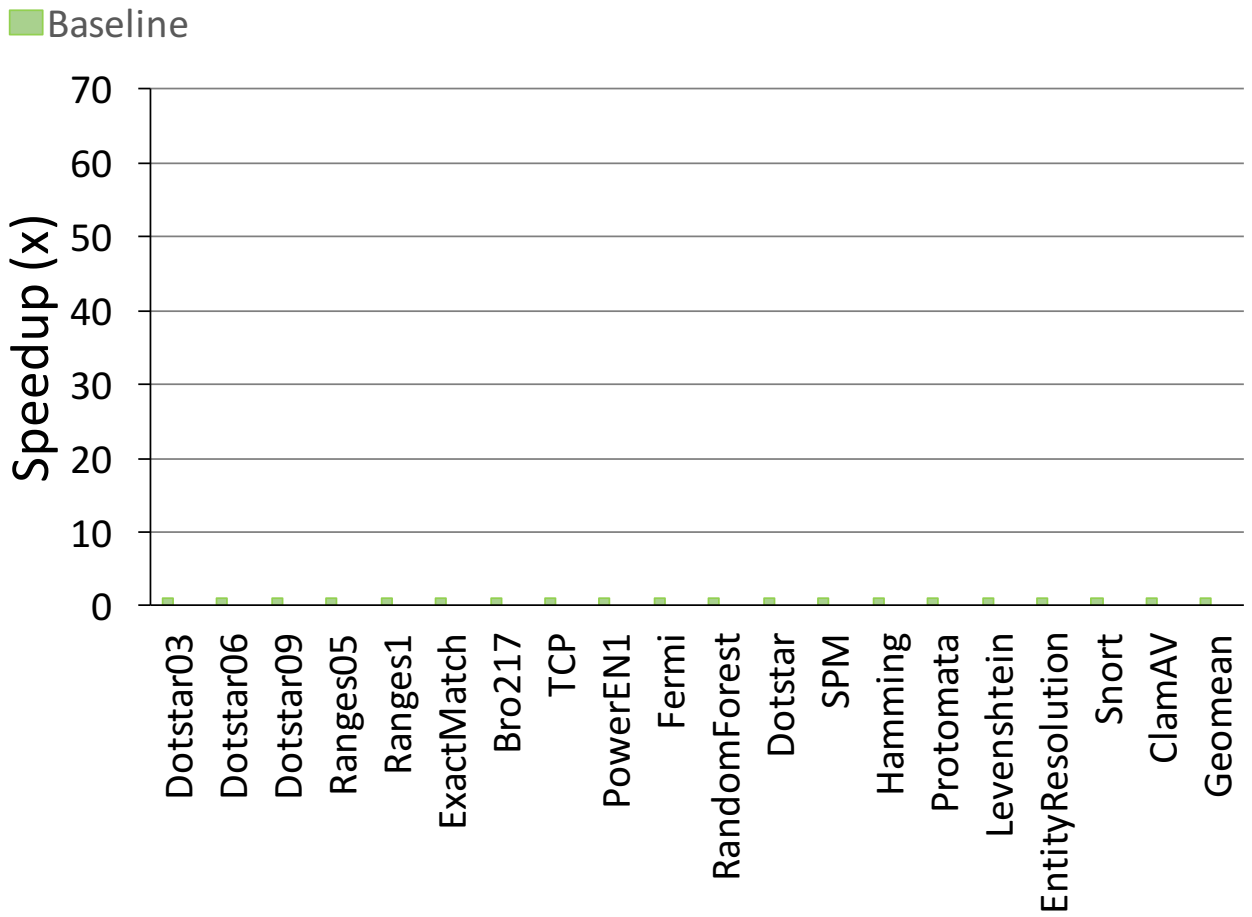


Active flows

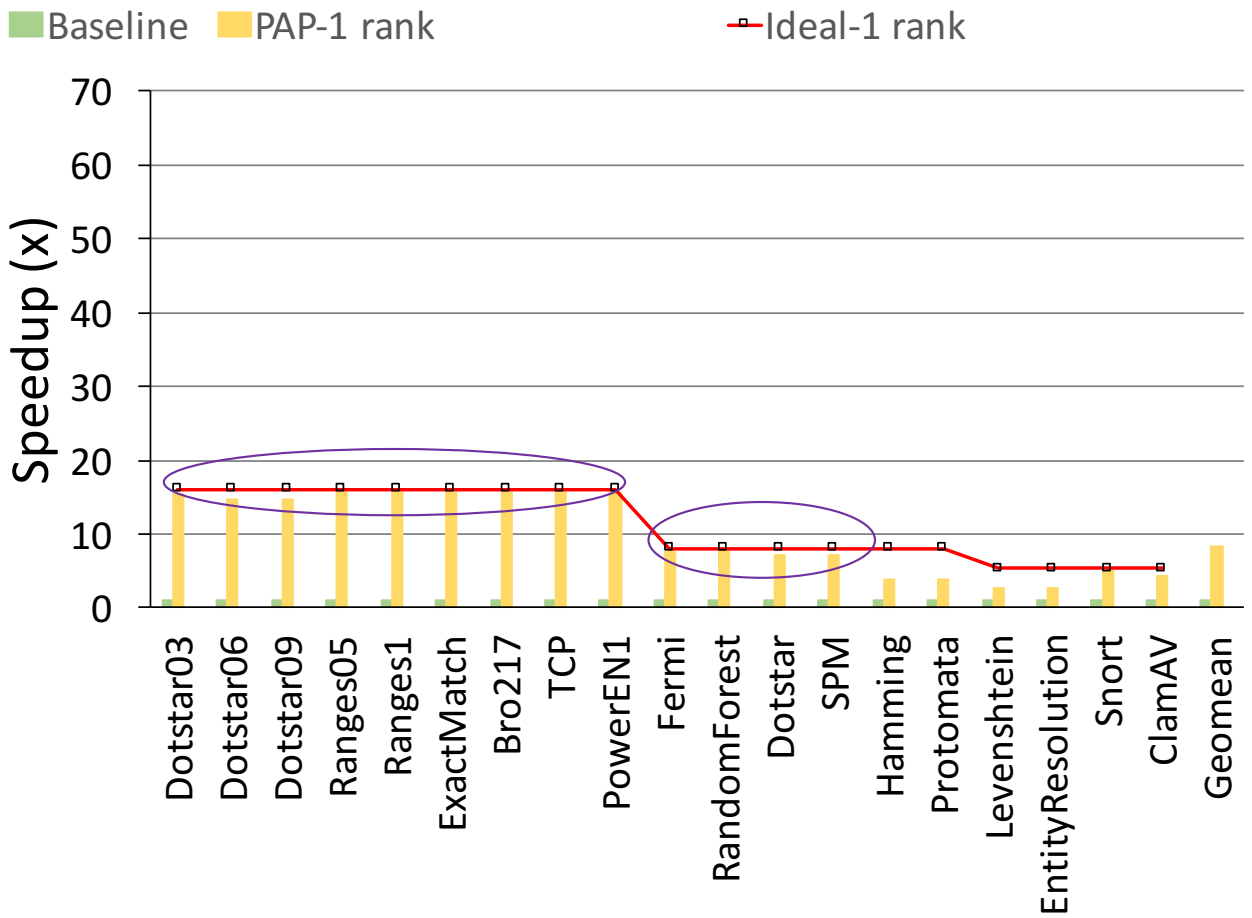


Flow reduction optimizations significantly reduce enumeration complexity

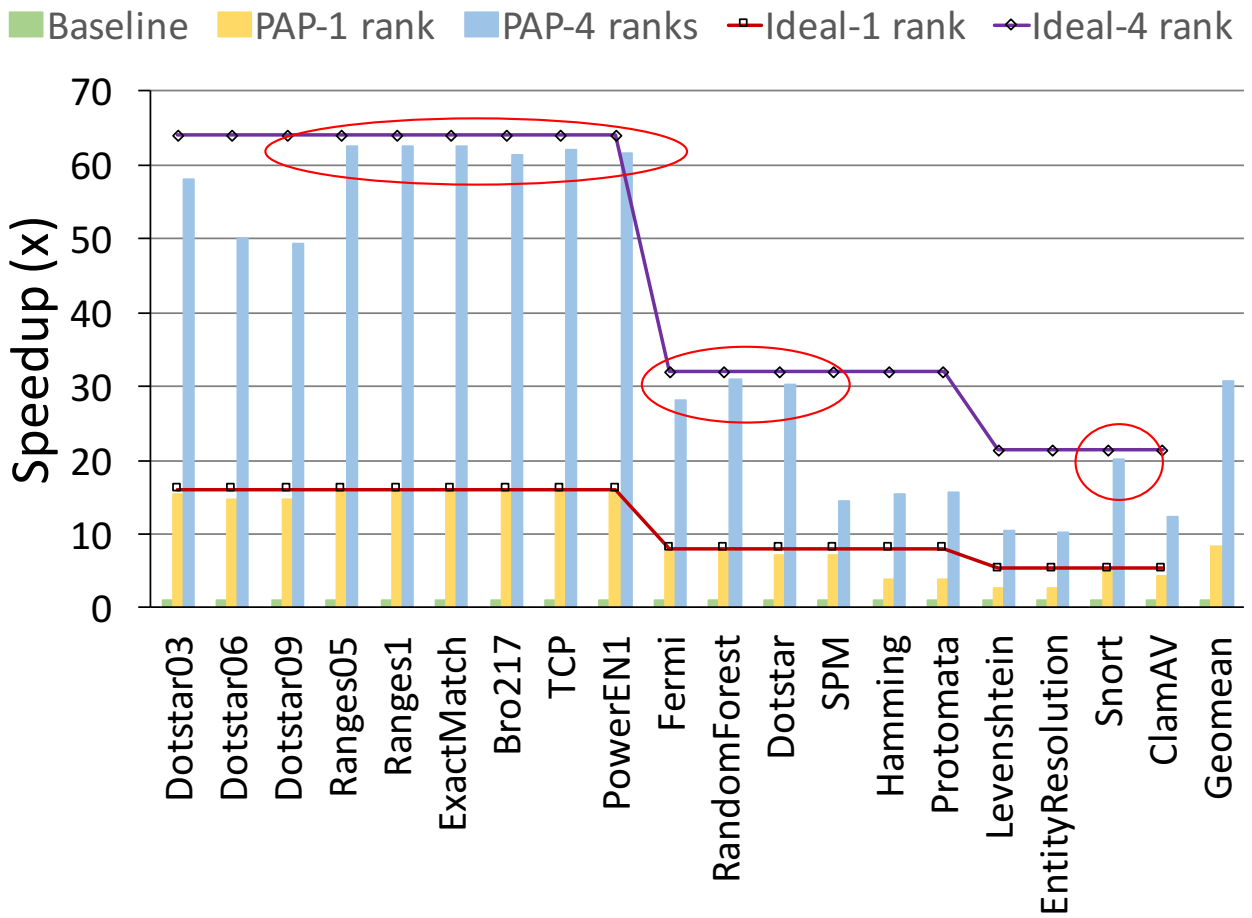
Speedup


Baseline
Speedup (x)
70
60
50
40
30
20
10
0
Dotstar03
Dotstar06
Dotstar09
Ranges05
Ranges1
ExactMatch
Bro217
TCP
PowerEN1
Fermi
RandomForest
Dotstar
SPM
Hamming
Protomata
Levenshtein
EntityResolution
Snort
ClamAV
Geomean

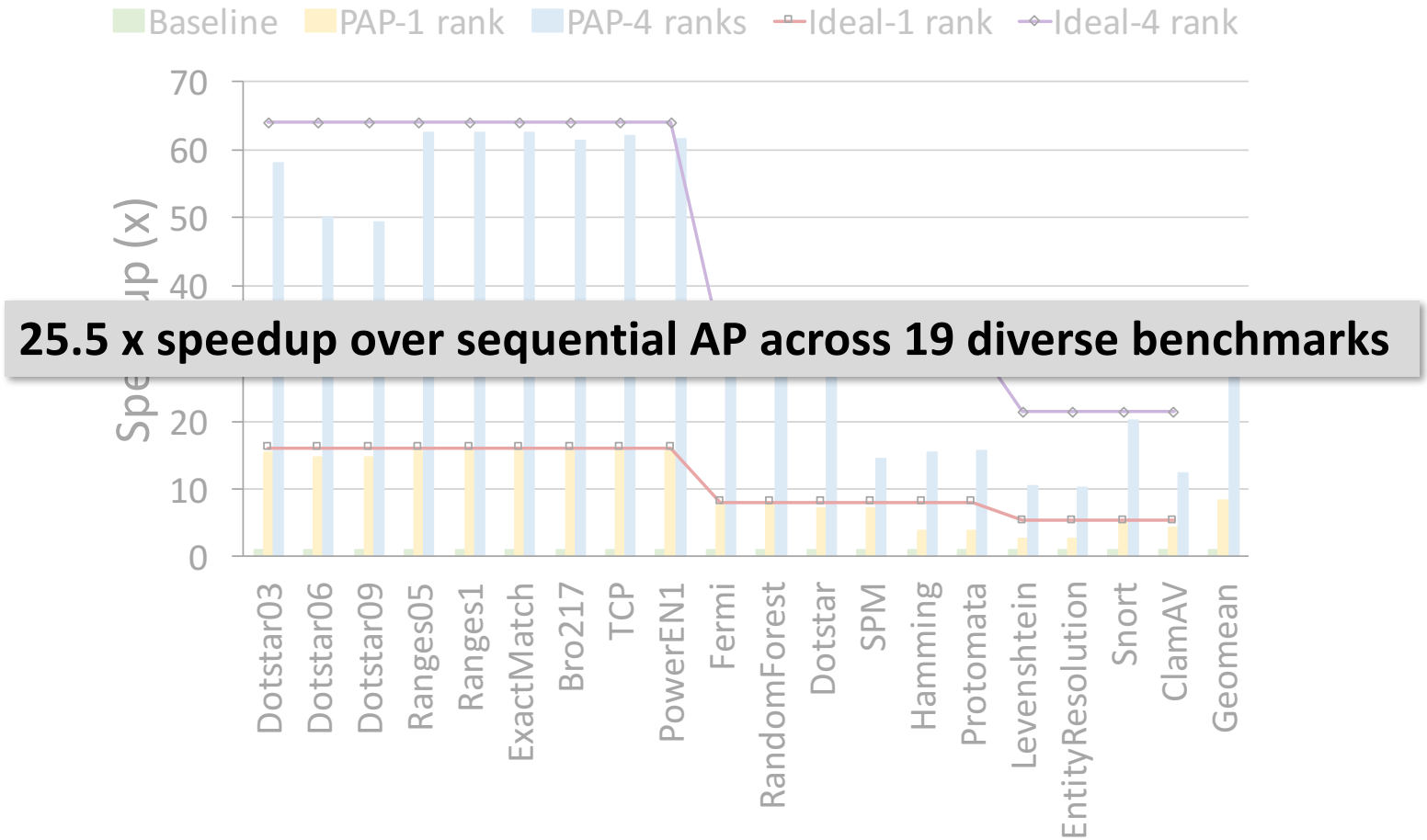
Speedup


Baseline
PAP-1 rank
Ideal-1 rank
Speedup (x)
70
60
50
40
30
20
10
0
Dotstar03
Dotstar06
Dotstar09
Ranges05
Ranges1
ExactMatch
Bro217
TCP
PowerEN1
Fermi
RandomForest
Dotstar
SPM
Hamming
Protomata
Levenshtein
EntityResolution
Snort
ClamAV
Geomean

Speedup


Baseline
PAP-1 rank
PAP-4 ranks
Ideal-1 rank
Ideal-4 rank
Speedup (x)
70
60
50
40
30
20
10
0
Dotstar03
Dotstar06
Dotstar09
Ranges05
Ranges1
ExactMatch
Bro217
TCP
PowerEN1
Fermi
RandomForest
Dotstar
SPM
Hamming
Protomata
Levenshtein
EntityResolution
Snort
ClamAV
Geomean

Speedup



25.5 x speedup over sequential AP across 19 diverse benchmarks

PAP Summary

✓ Enumerative Parallelization on Micron's AP profitable

`Parallel Automata Processor`

Enumeration path tracking

repurposed AP flows

Enumeration complexity

range-guided input partitioning

connected components

common parent merging

dynamic convergence, deactivation

Outline

- Motivation
- In-Memory Automata Processing
- Parallel Automata Processing
- **Cache Automaton**

Cache Automaton ?

- ! DRAM technology energy inefficient and AP is slow (~133 MHz)
- ! Low packing density (12 MB AP = 200 MB regular DRAM)
- ! Off-chip host to accelerator communication overhead

Are SRAM-based caches suitable for automata processing ?

Opportunity

✓ **SRAM-based caches are faster, more energy efficient than DRAM**

✓ **Integrated on processor dies with performance-optimized logic**

! **Is cache capacity an issue for large NFA ?**

Opportunity

	Form Factor	Capacity	# States
DRAM	1 rank (8 devices)	200 MB	-
DRAM-based AP	1 rank (8 devices)	12 MB	384,000,000
Last-Level Cache	1 socket	25 MB	400,000,000

Challenges

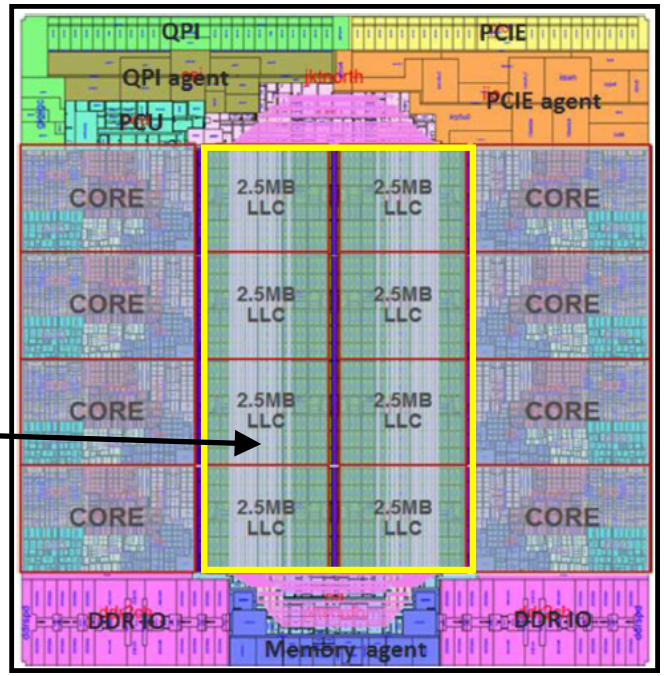
1. **Using cache hierarchy is slow**

Using the cache hierarchy is slow !

L1 cache
~ 2 cycles

L2 cache
~ 12 cycles

L3 cache
~ 36 cycles



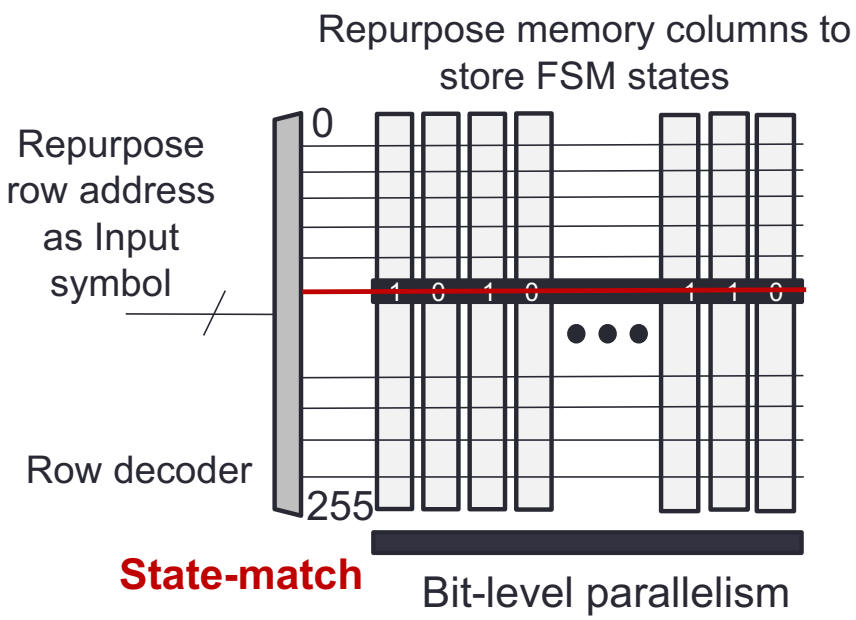
Credits: Intel

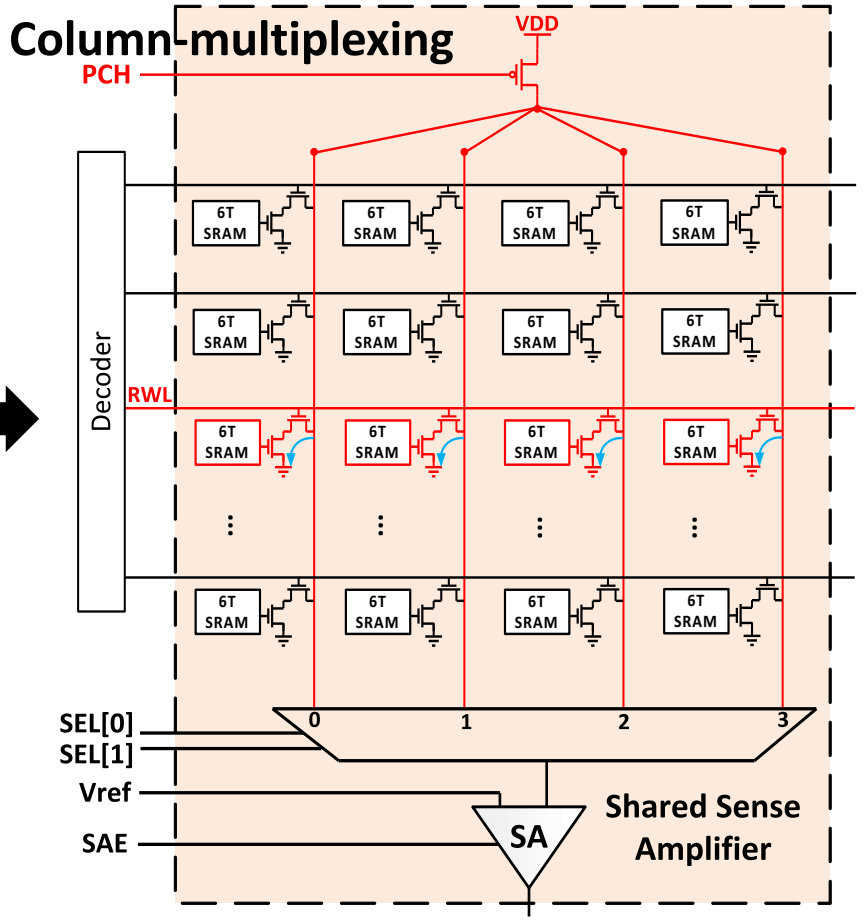
Low operating frequency ~ 111 MHz

Challenges

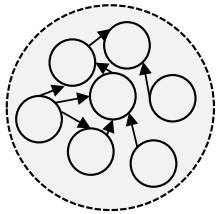
2 **Efficient state-match and state-transition in cache**

Column multiplexing reduces bit-parallelism

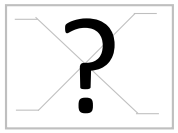




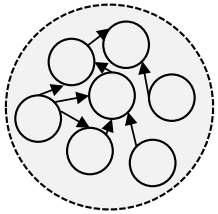
Scalable interconnect for state-transition in cache



Crossbar



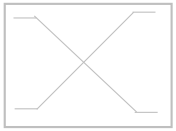
! Overprovisioned w/ dynamic arbitration, multi-bit ports



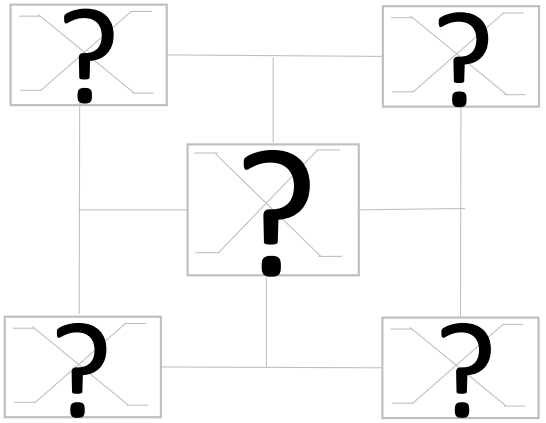
100,000 states

Infeasible

Crossbar



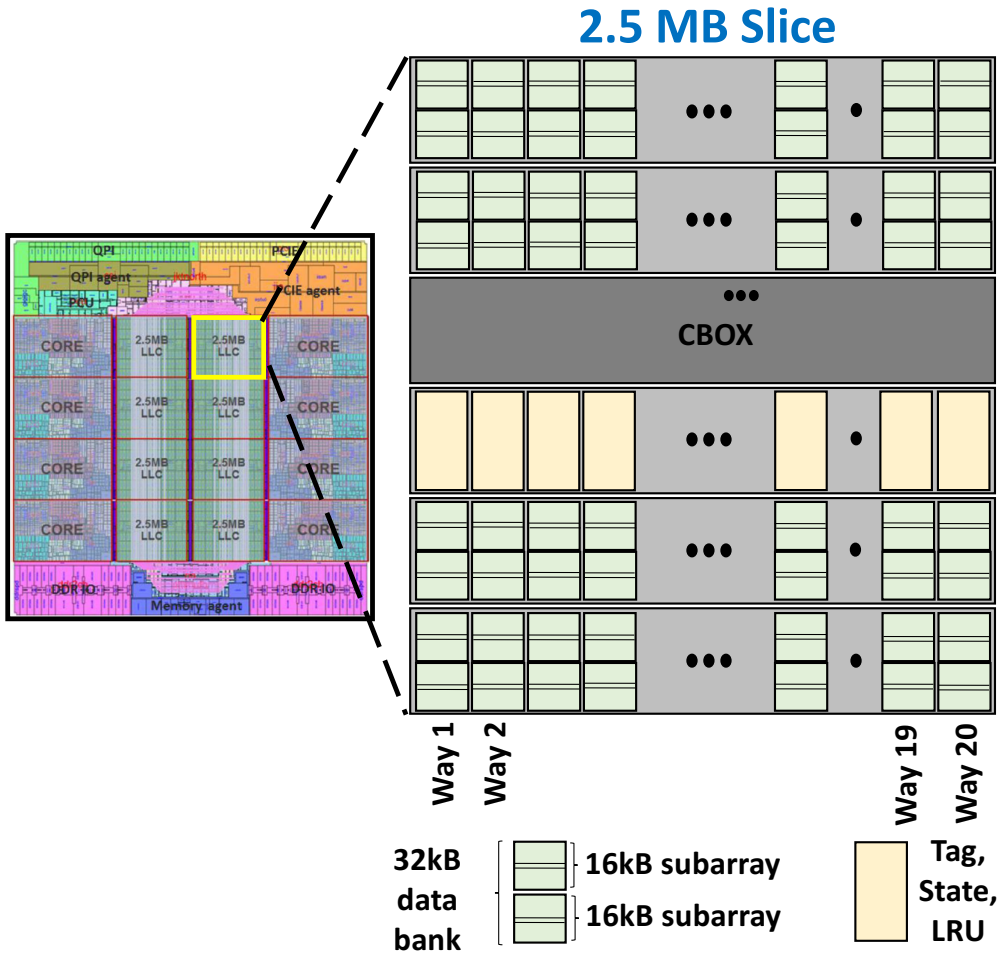
100,000 x 100,000



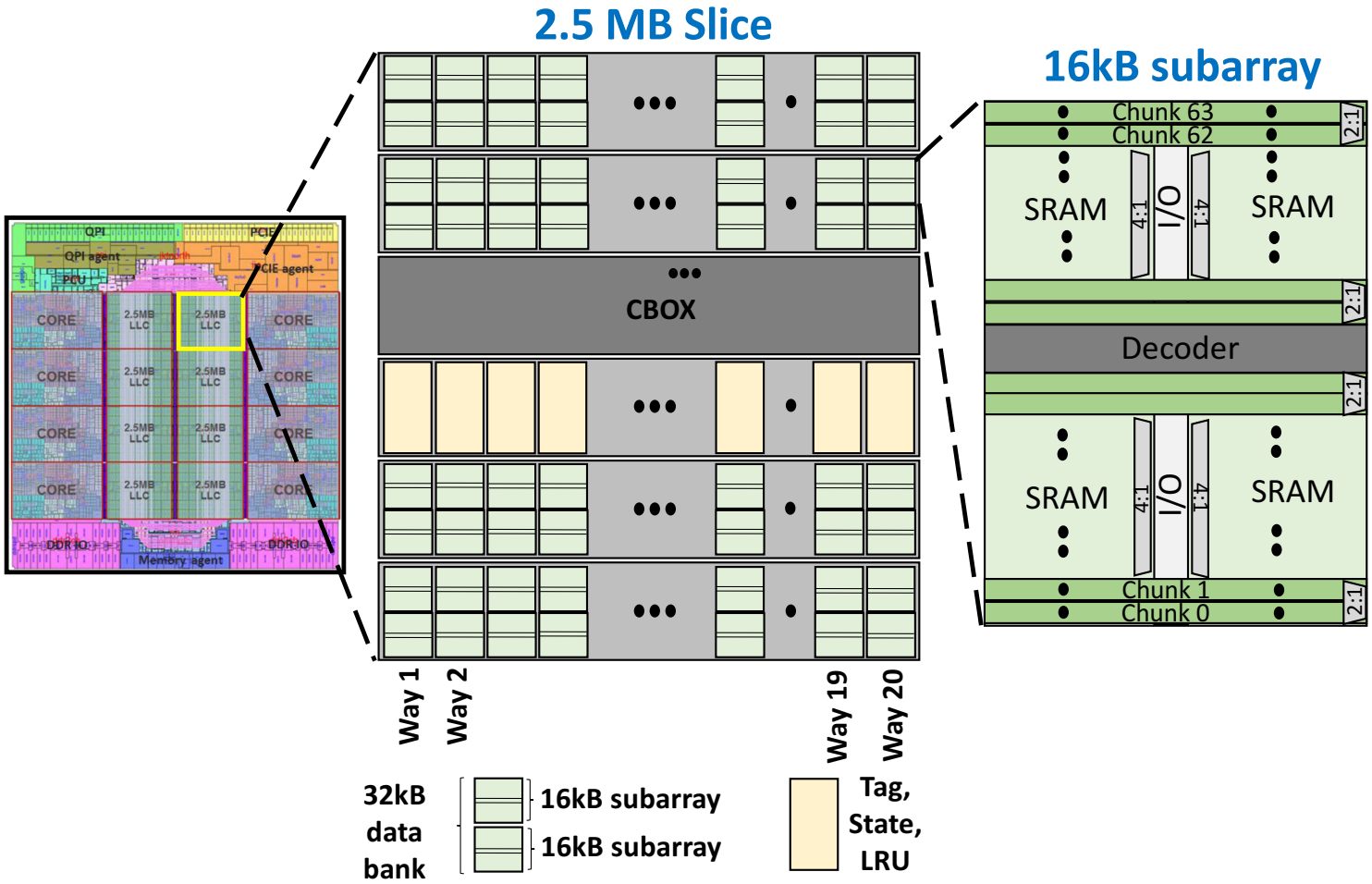
Network Architecture

1 **In-situ computation cognizant of cache geometry can provide benefits**

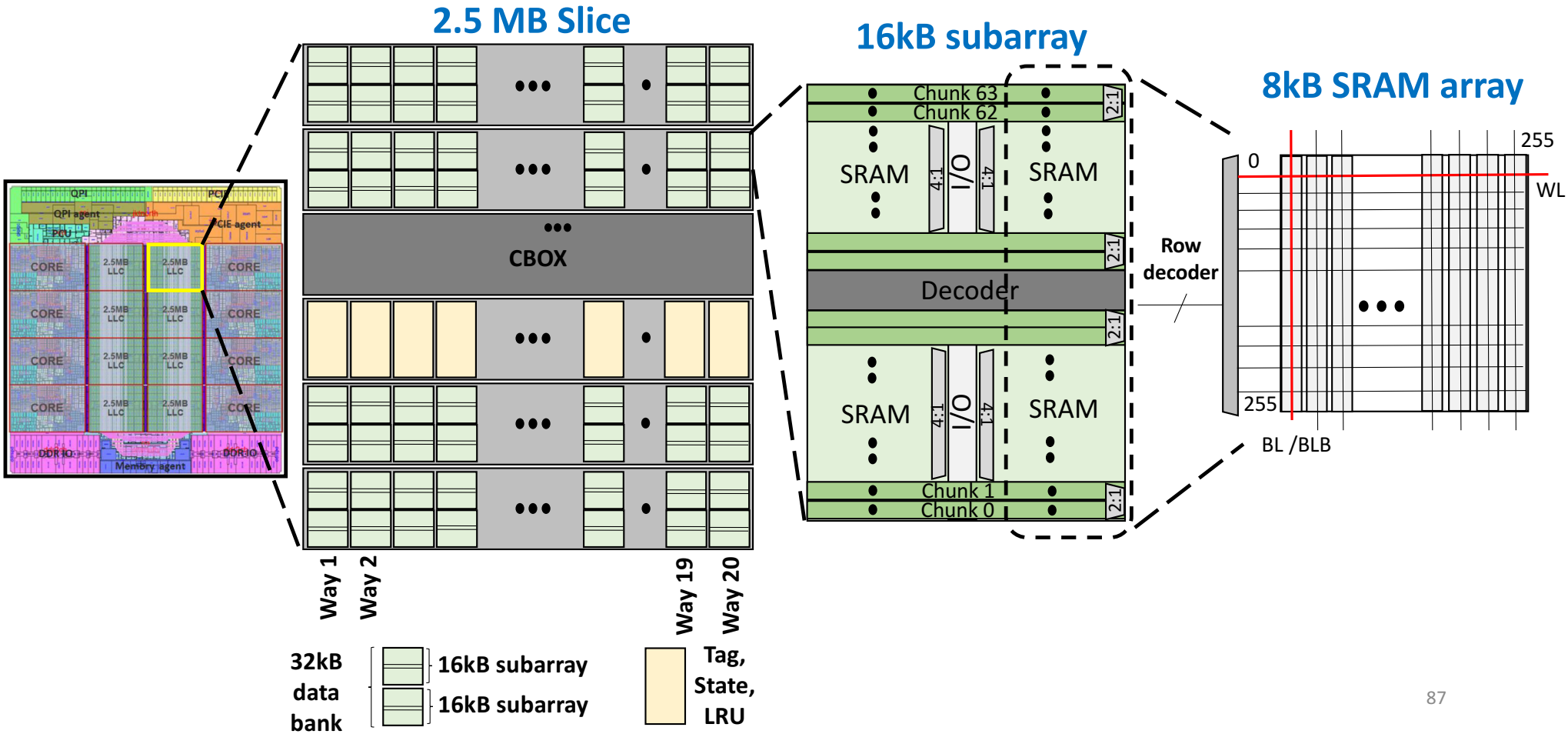
Intel Xeon Last-Level Cache



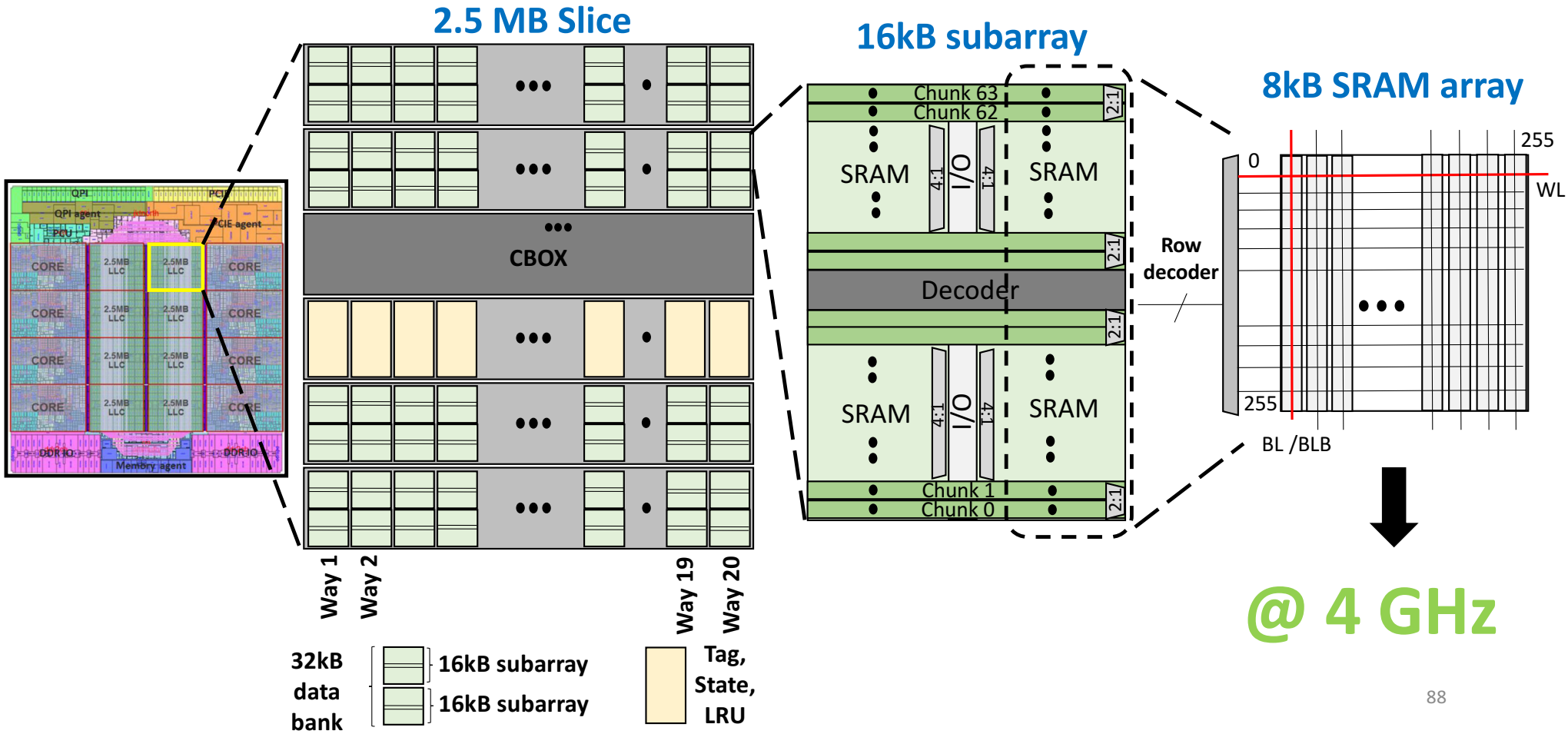
Intel Xeon Last-Level Cache



Intel Xeon Last-Level Cache

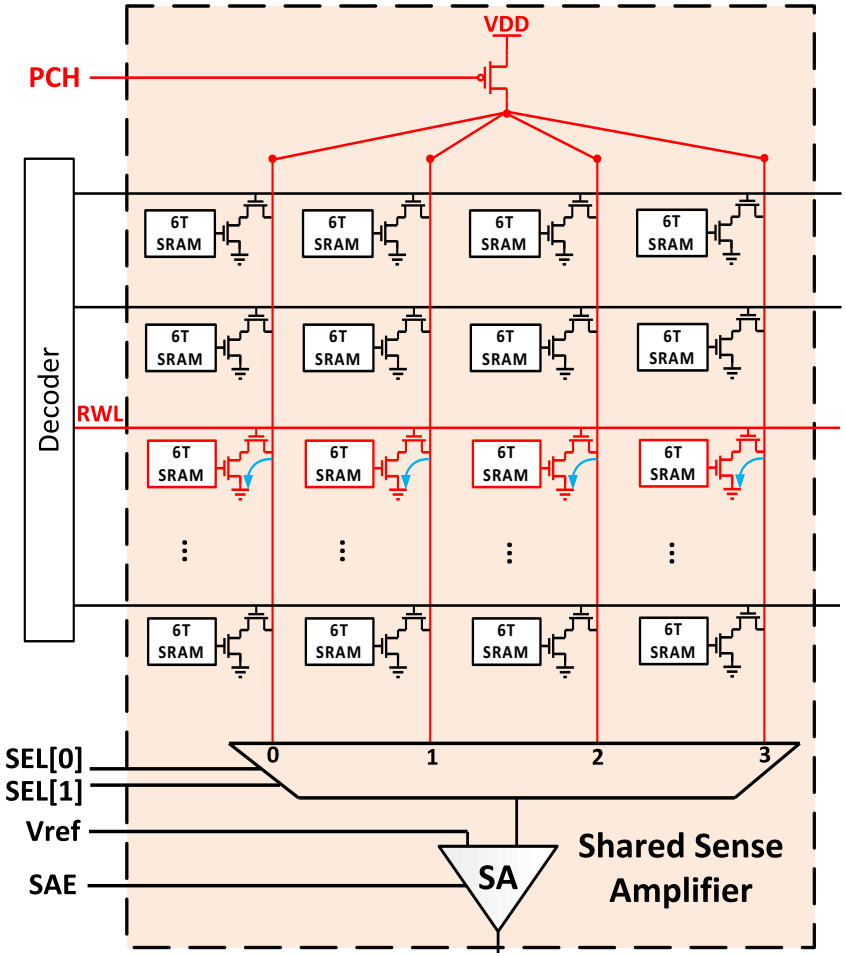


Intel Xeon Last-Level Cache



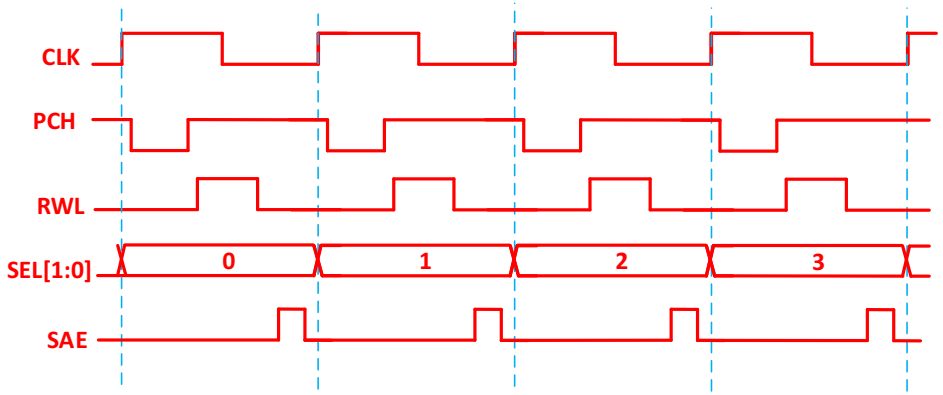
2 Sense-amplifier cycling enables highly parallel, low latency state-match

Accelerating state-match is challenging …



Column-multiplexing

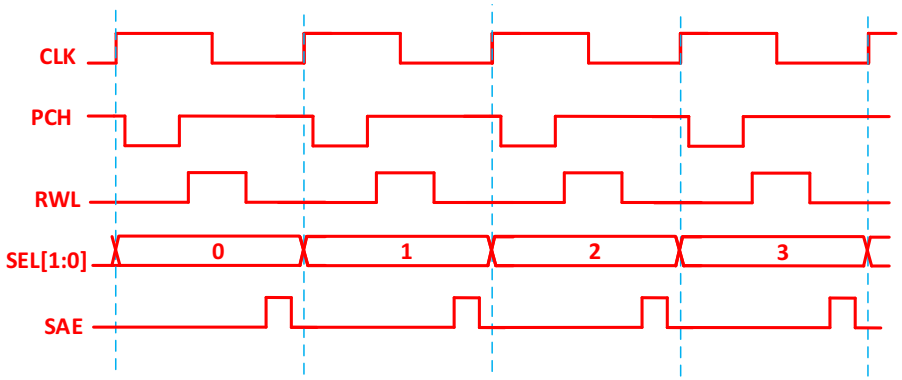
Read sequence



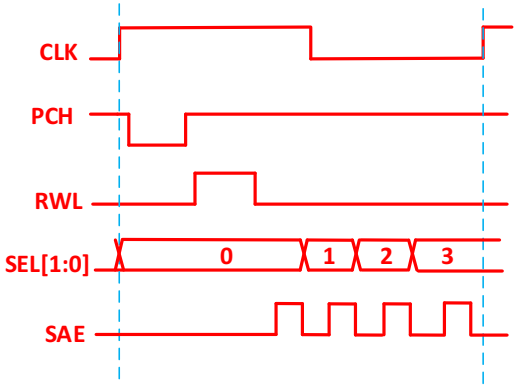
! **4 cycles to read 4 adjacent bits (state-match results)**

Sense-amplifier cycling

Read sequence



Read sequence (optimized)



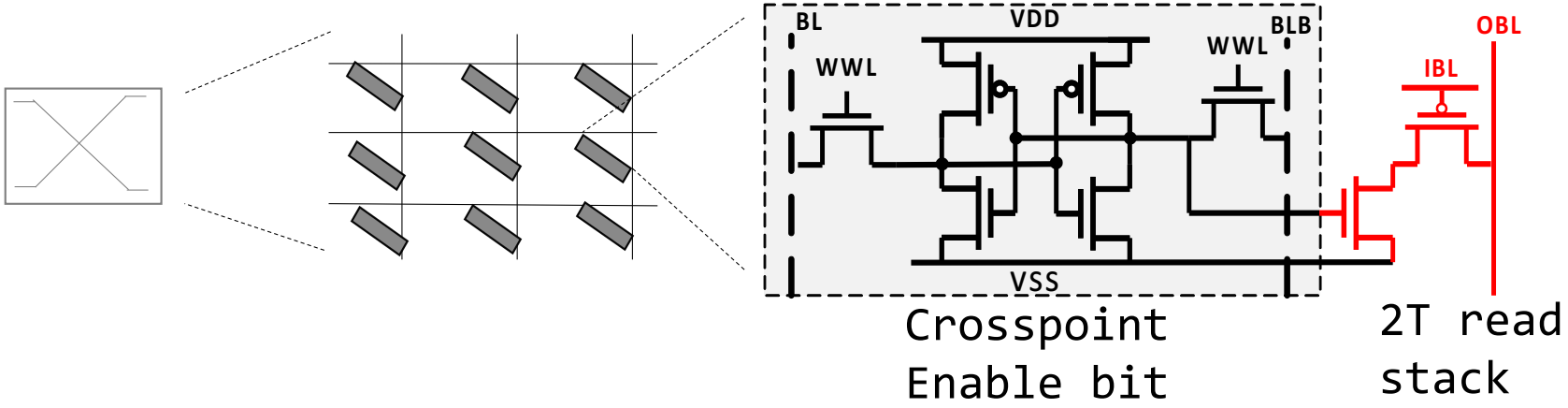
! **4 cycles to read 4 adjacent bits (state-match results)**

✓ **< 2 cycles to read 4 adjacent bits (state-match results)**

3

8T-based SRAM arrays can be repurposed to compactly encode state-transitions

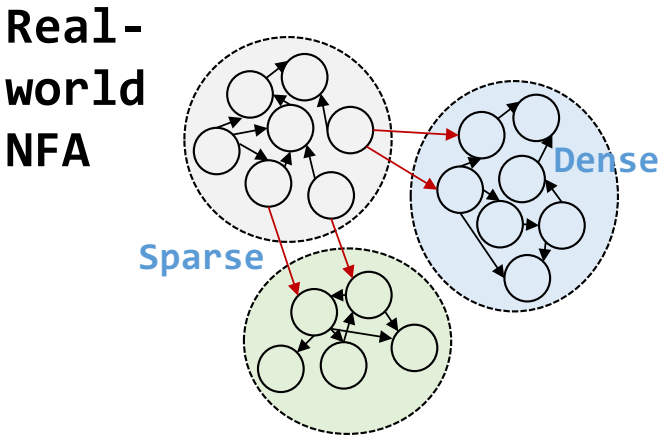
Accelerating state-transition

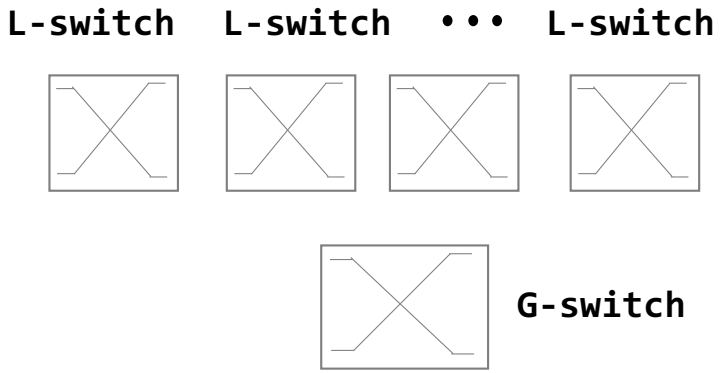


Compact 8T SRAM-based switches as building block of programmable interconnect

4 **Real-world NFA can be grouped into dense regions with sparse connectivity**

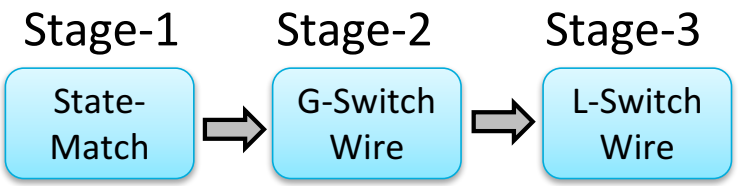
Hierarchical network architecture





Hierarchical switch topology scales to large NFA

Improving throughput by pipelining

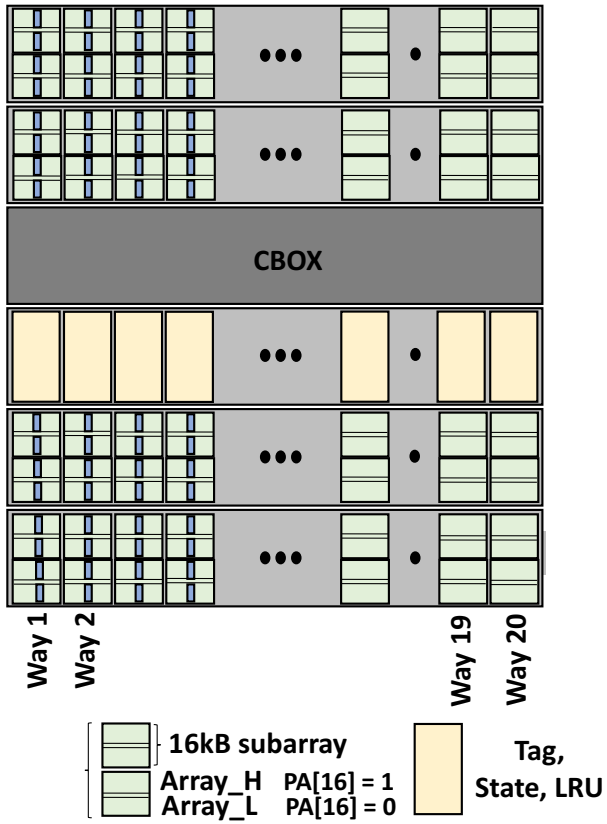


State match = Array access
State transition = Switch and link traversals

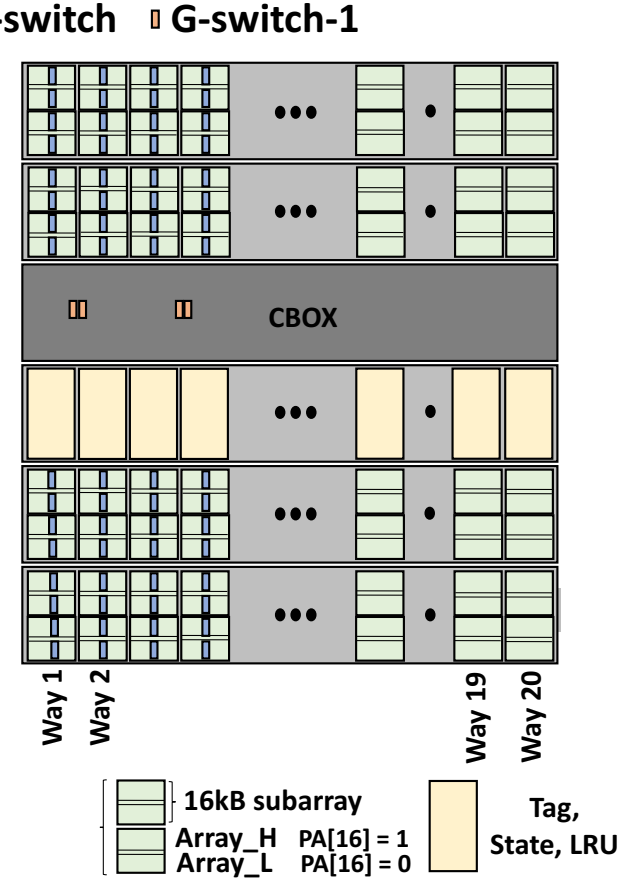
Decouple and overlap state-match and state-transition to maximize symbol processing throughput

Putting it together …

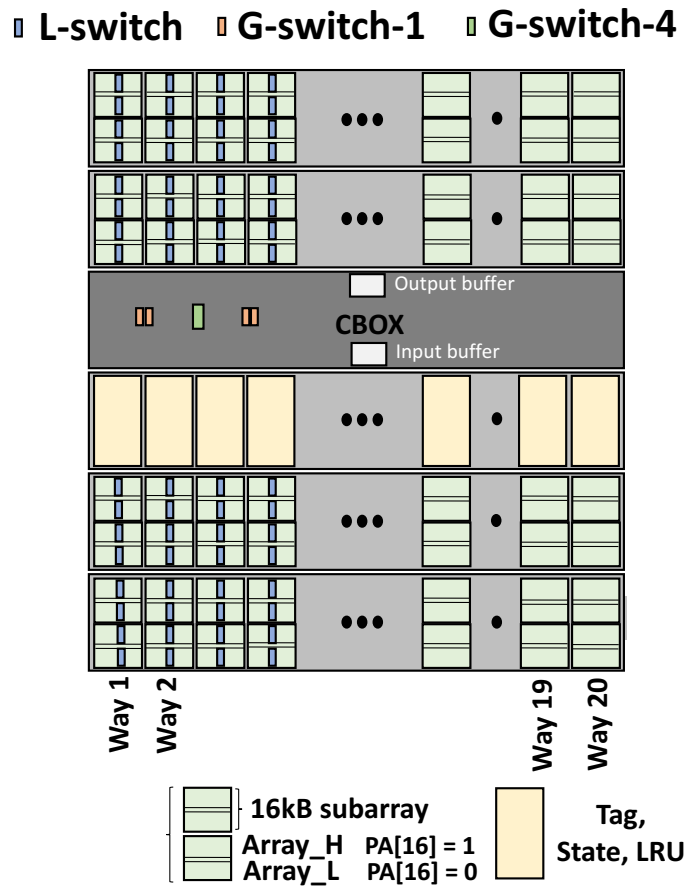
- L-switch

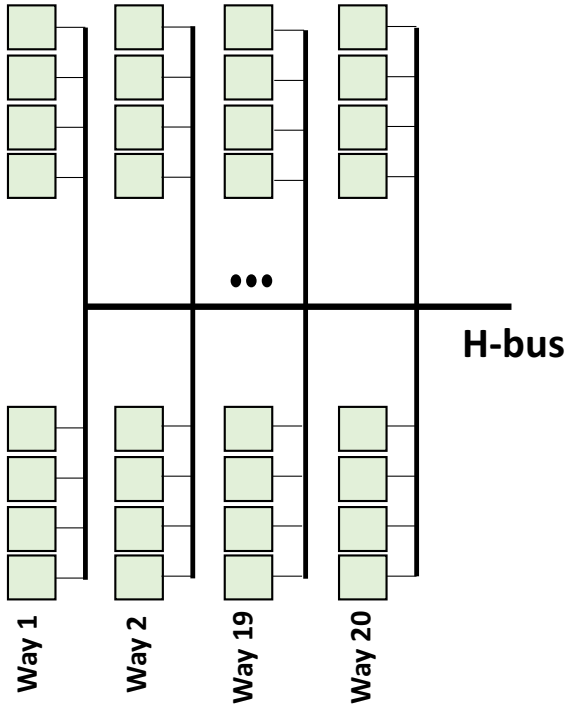


Cache Automaton Architecture

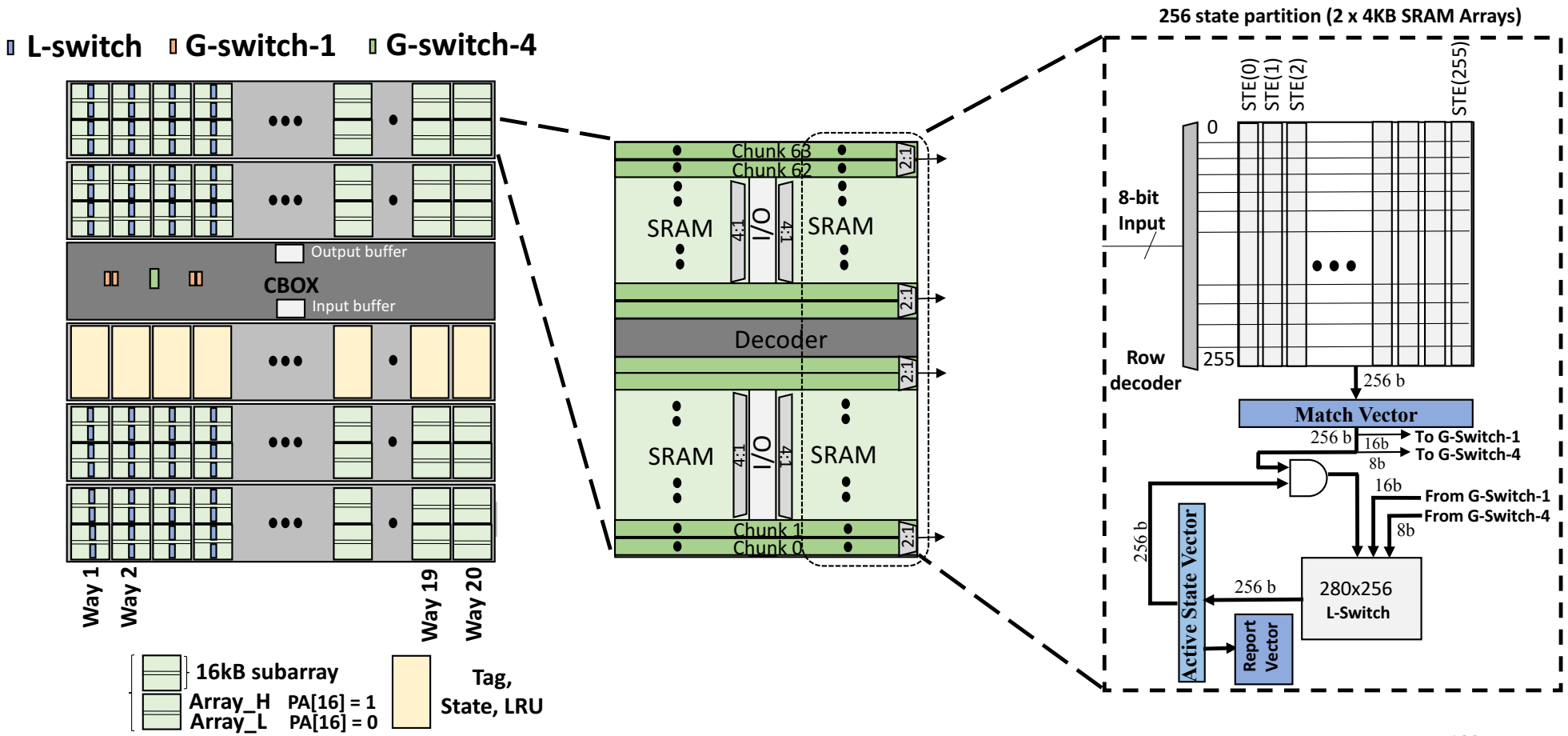


Cache Automaton Architecture

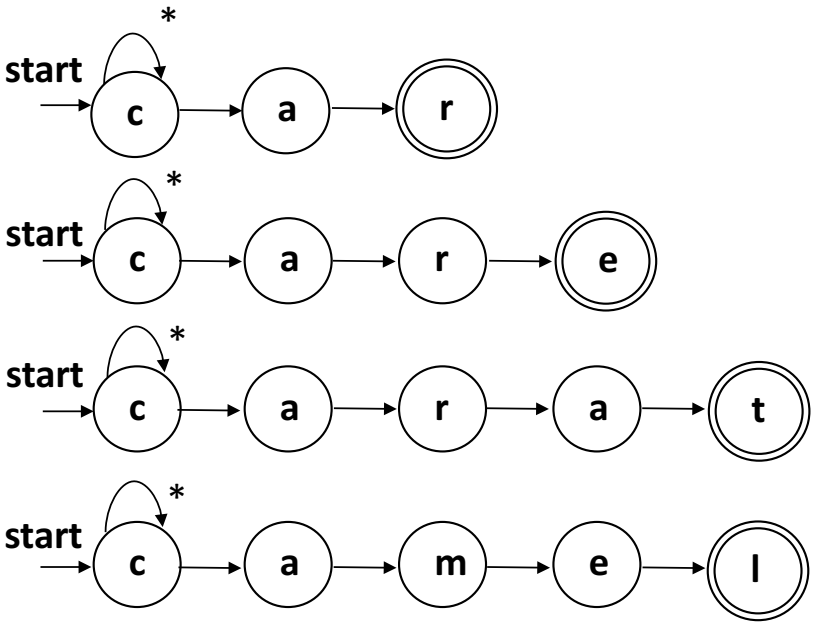


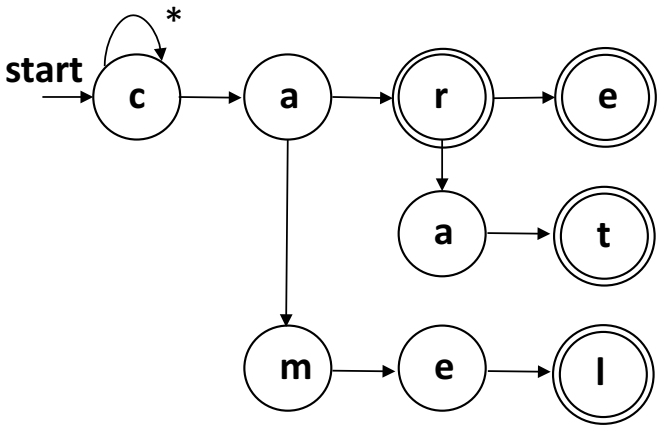


Cache Automaton Architecture



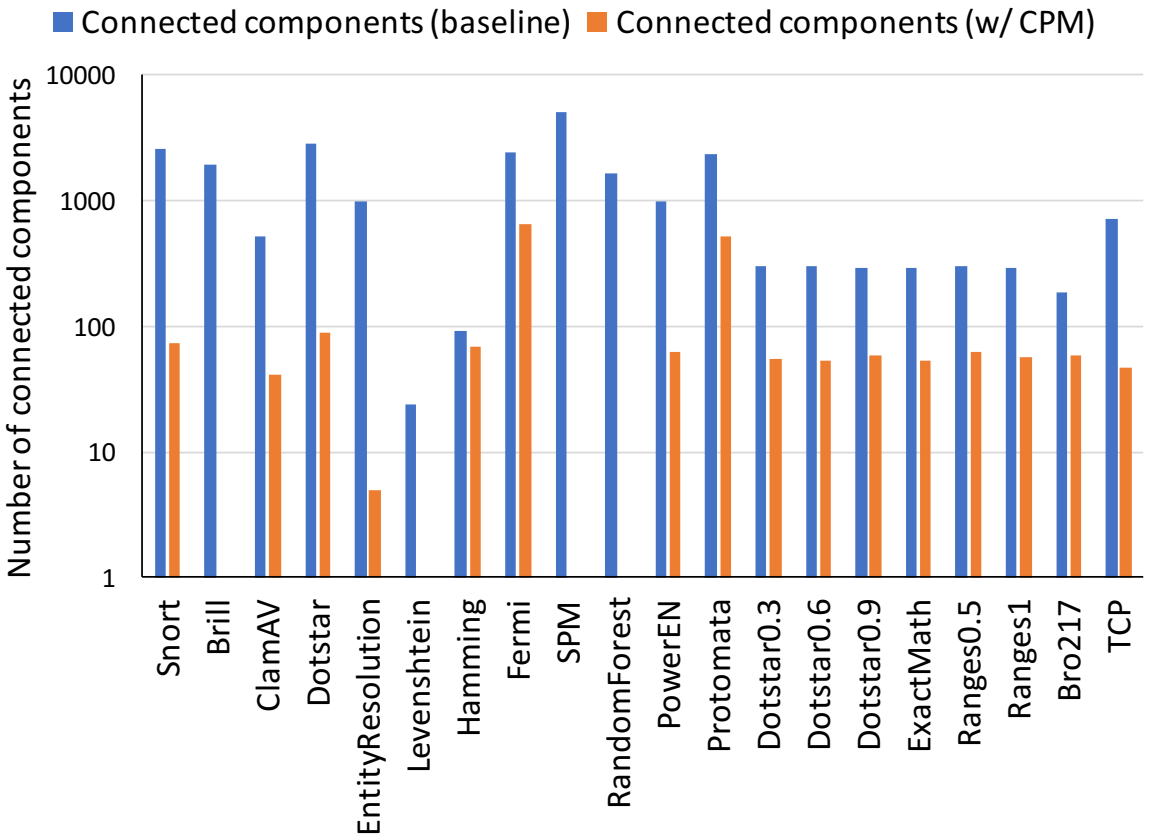
Merging connected components



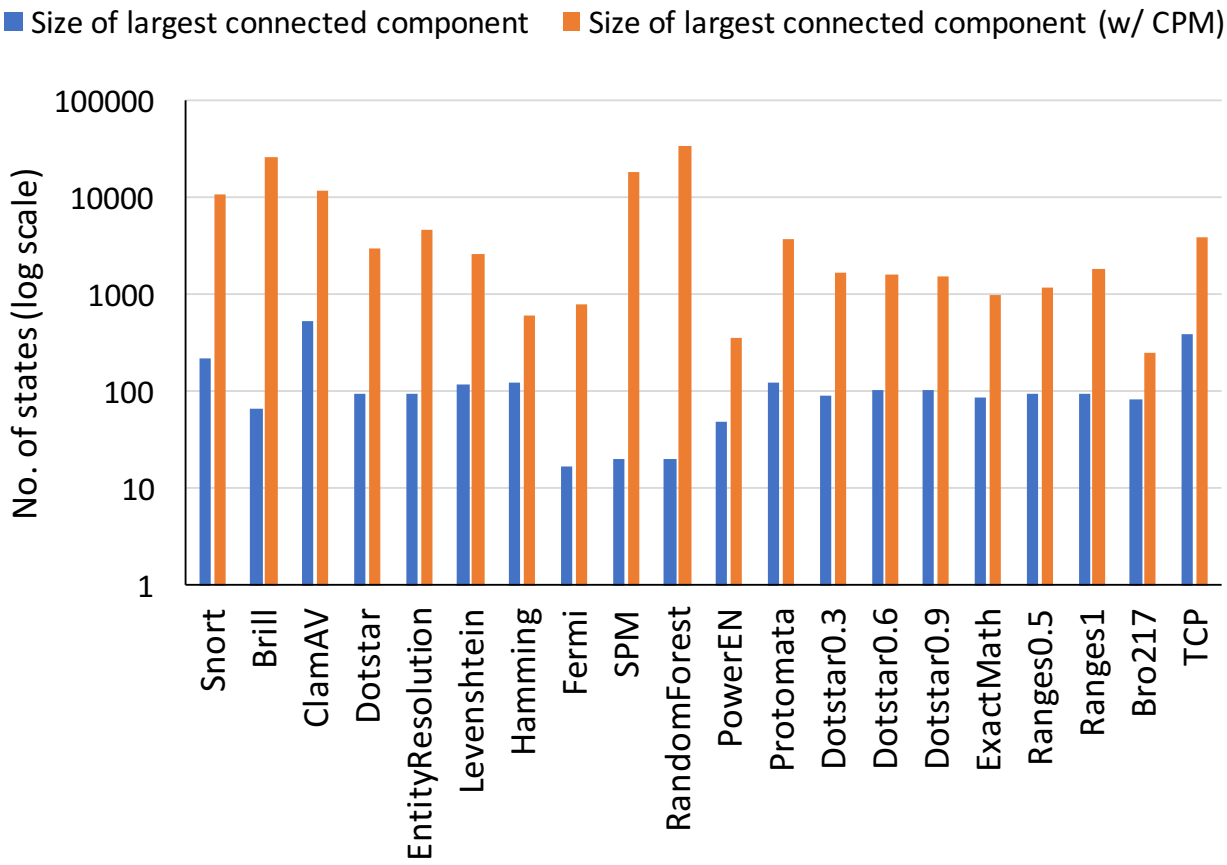


Merging connected components

Merging connected components – fewer CCs



Merging connected components - larger CCs



Cache Automaton designs

Performance-optimized (CA_P)

Small connected components -> low radix switches

Redundant state activity

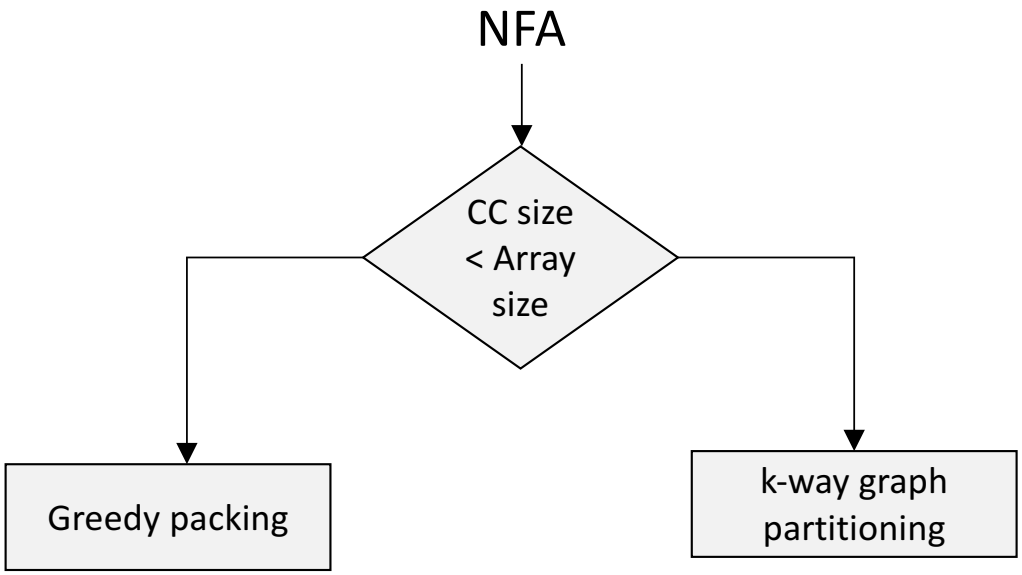
Space-optimized (CA_S)

Low footprint, no redundant state activity

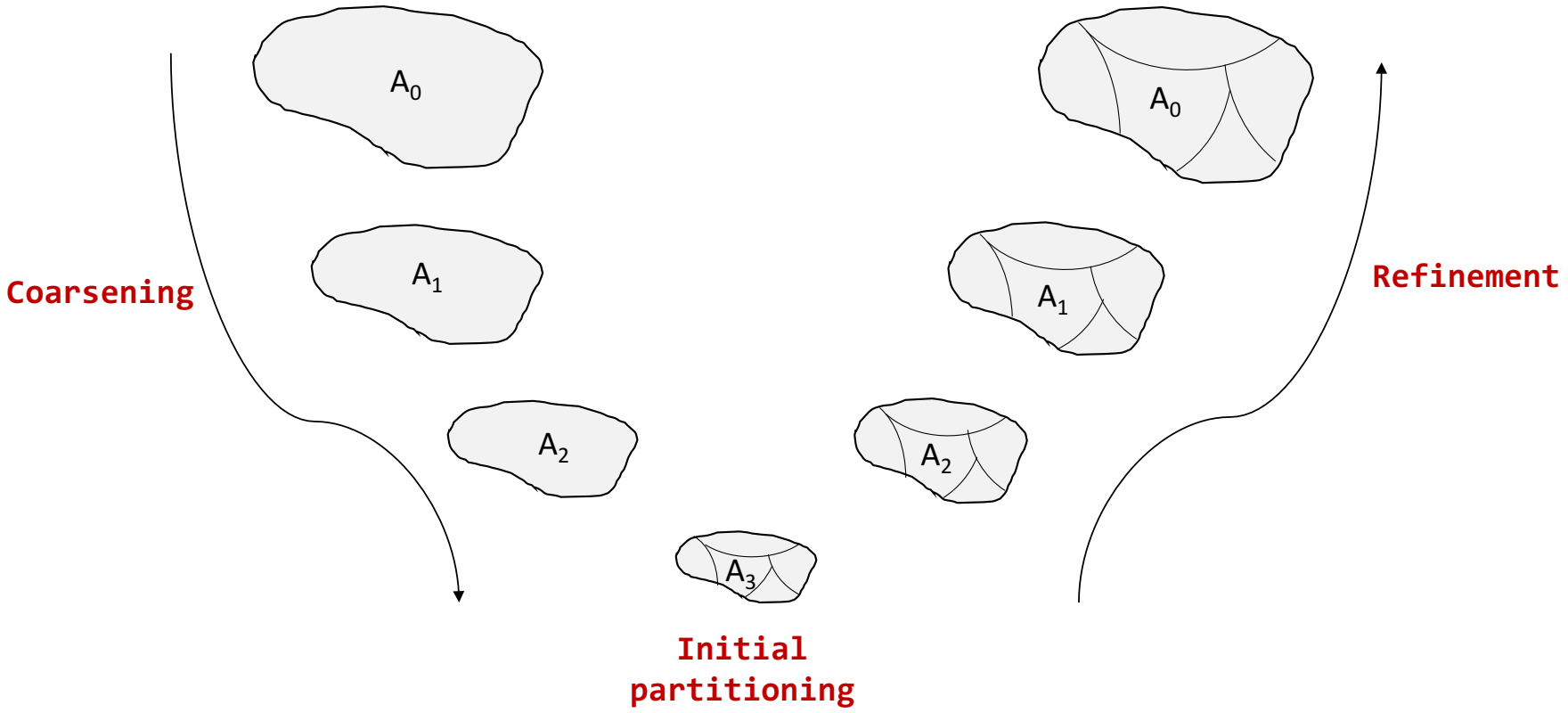
Large connected components ->
high radix switches and careful mapping

Compiler

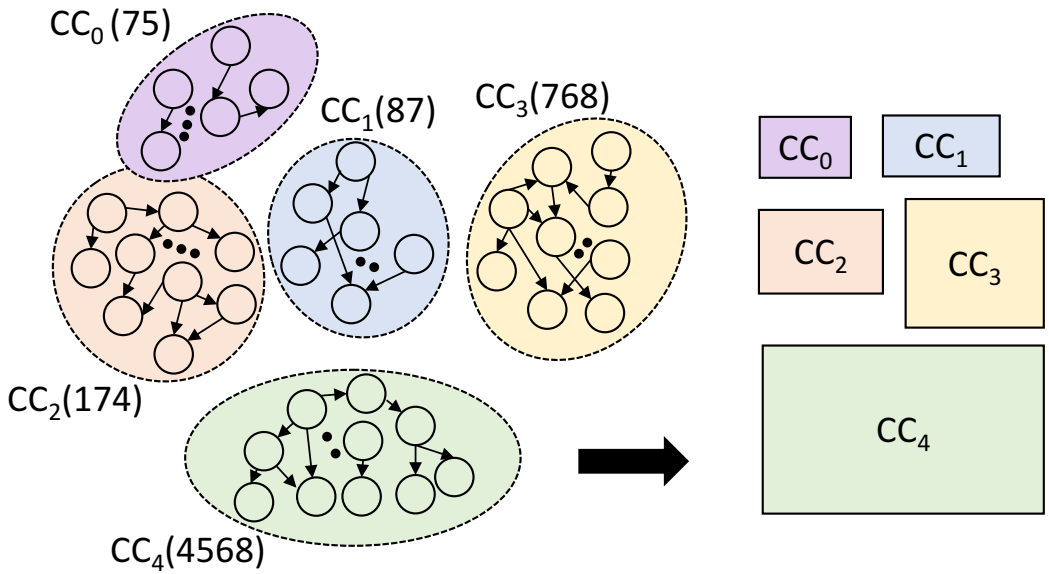
Maps NFA states to cache arrays while obeying the connectivity constraints of local and global switches



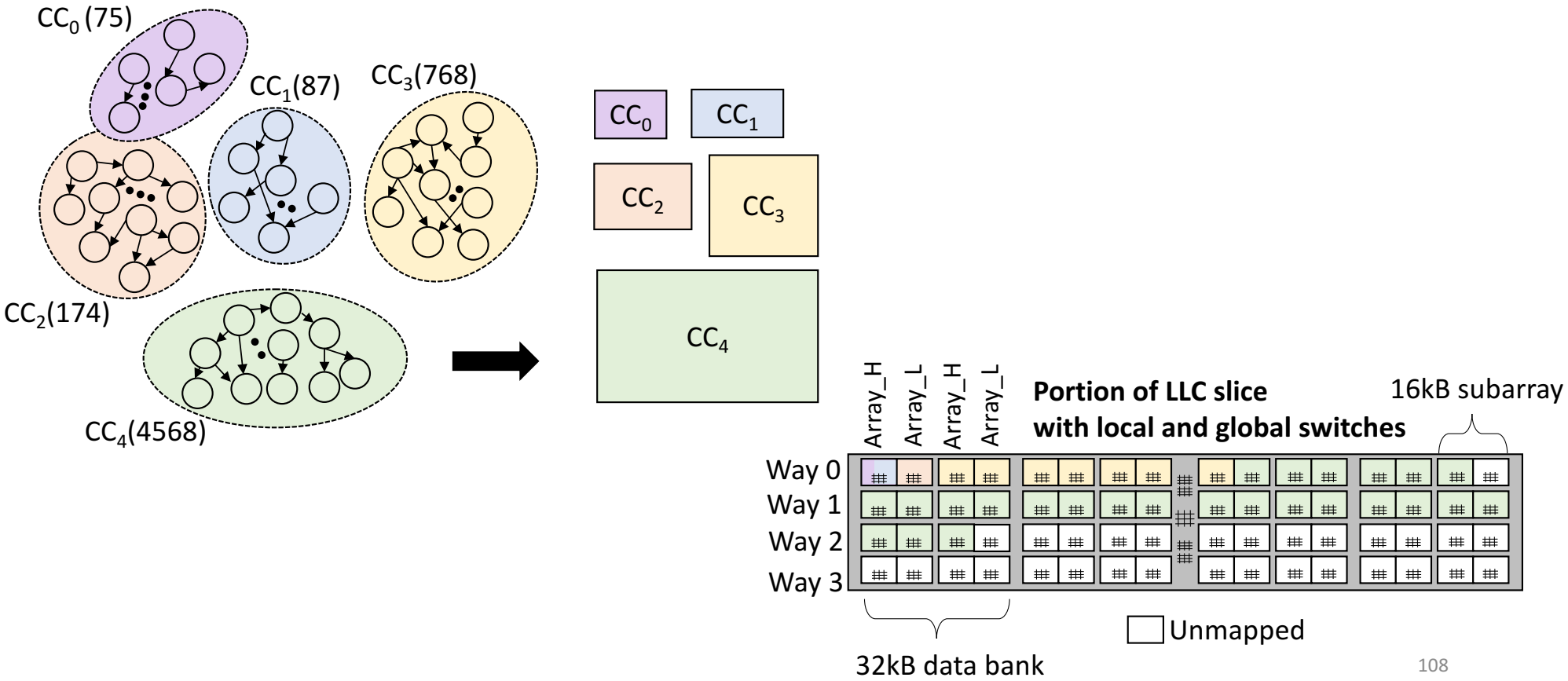
k-way graph partitioning (METIS)



Case Study: Entity Resolution



Case Study: Entity Resolution

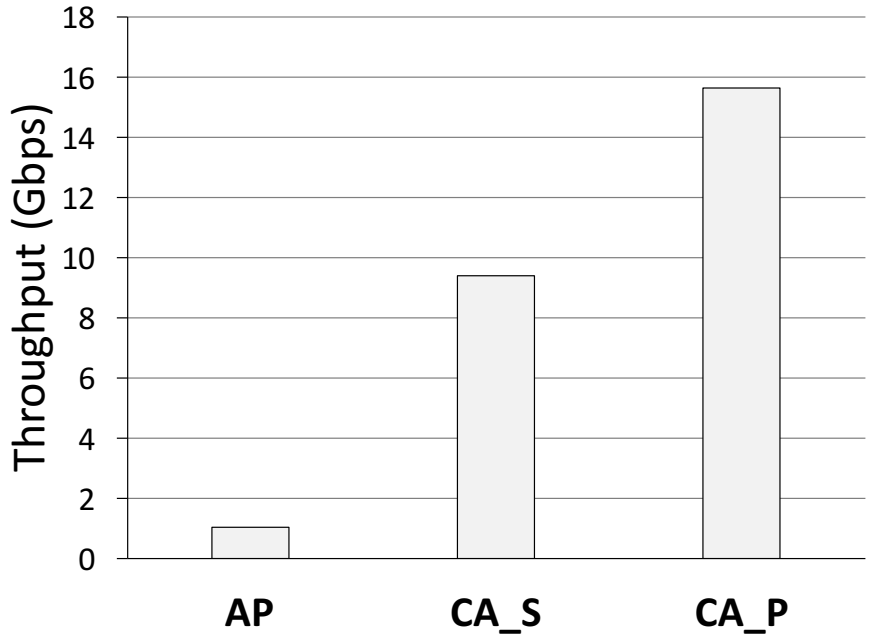


Experimental Setup

- ✓ Multi-Threaded Virtual Automata Simulator (VASim)
- ✓ METIS – k-way graph partitioning
- ✓ ANMLZoo and Regex benchmark suites, 1/10 MB input data
- ✓ Memory compiler 28nm – Area, power, delay of 6T/8T SRAM
- ✓ Wire Delay (Global wires – 66 ps/mm, H-bus – 300 ps/mm[1])

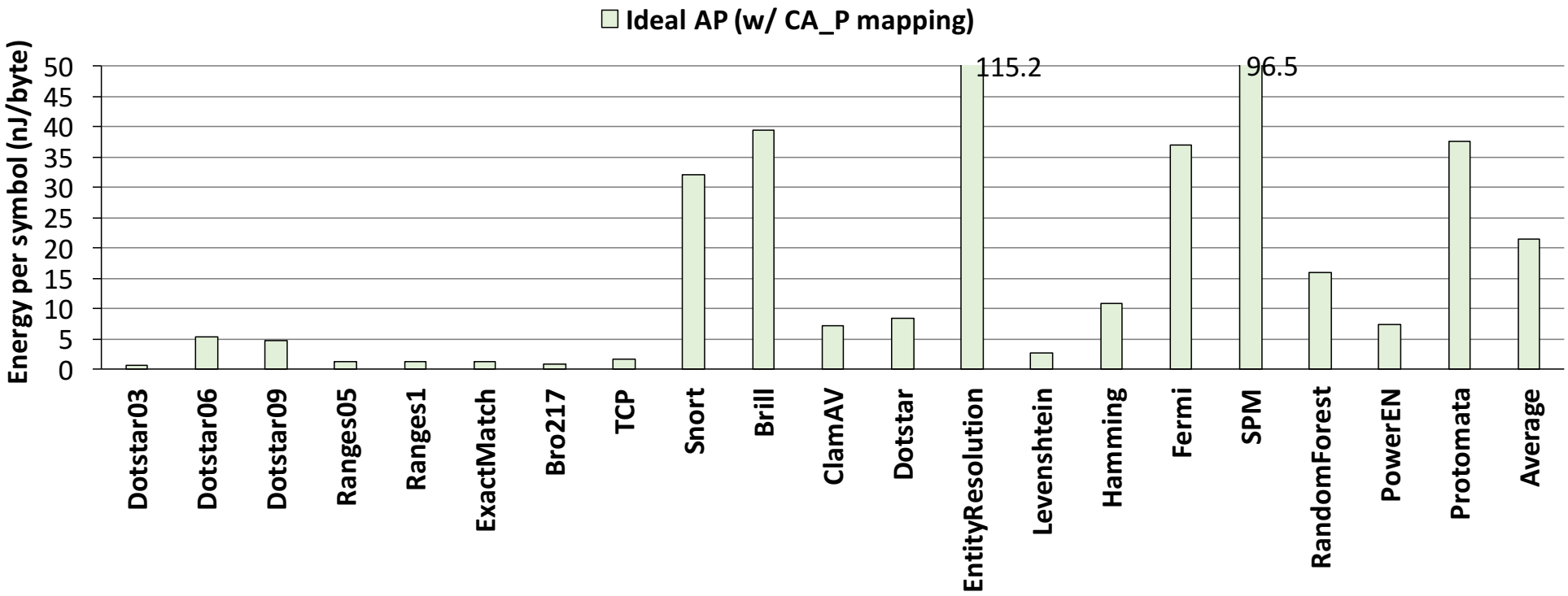
Das et al. “SLIP: reducing wire energy in the memory hierarchy”, ISCA’15

Performance

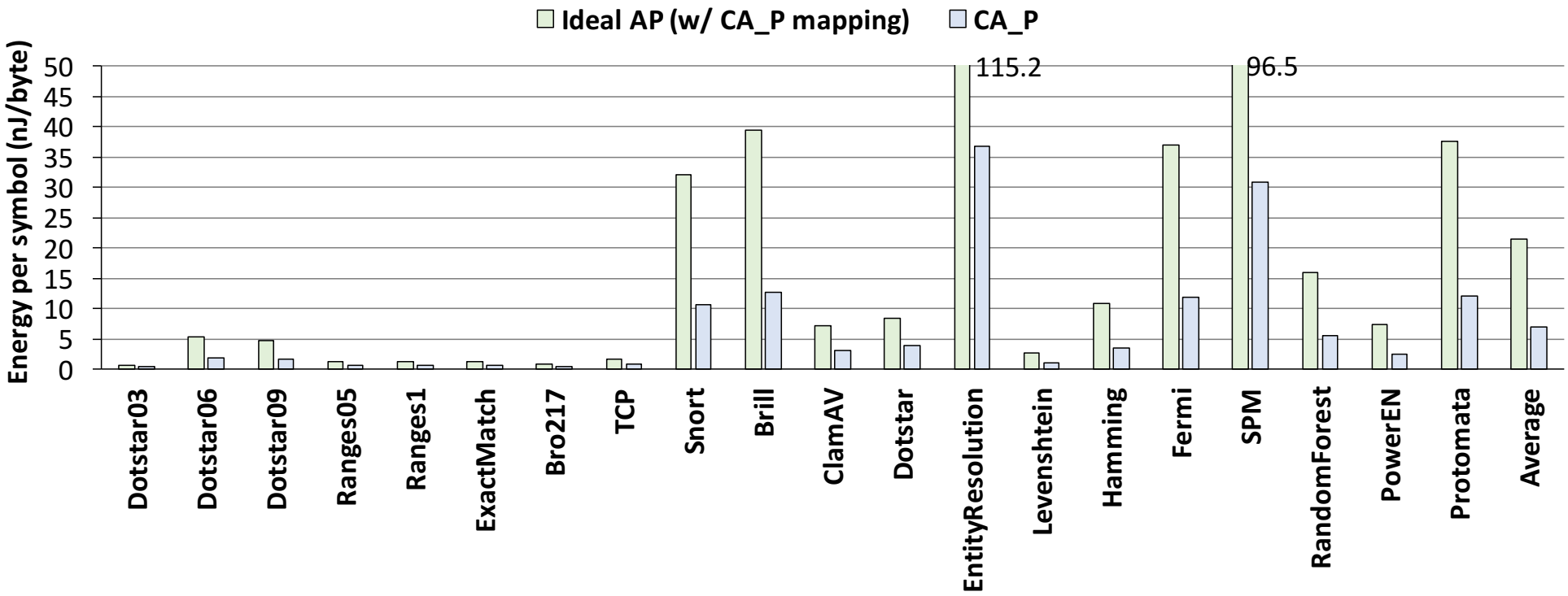


CA_P and CA_S achieve 9x and 15x speedup over AP respectively

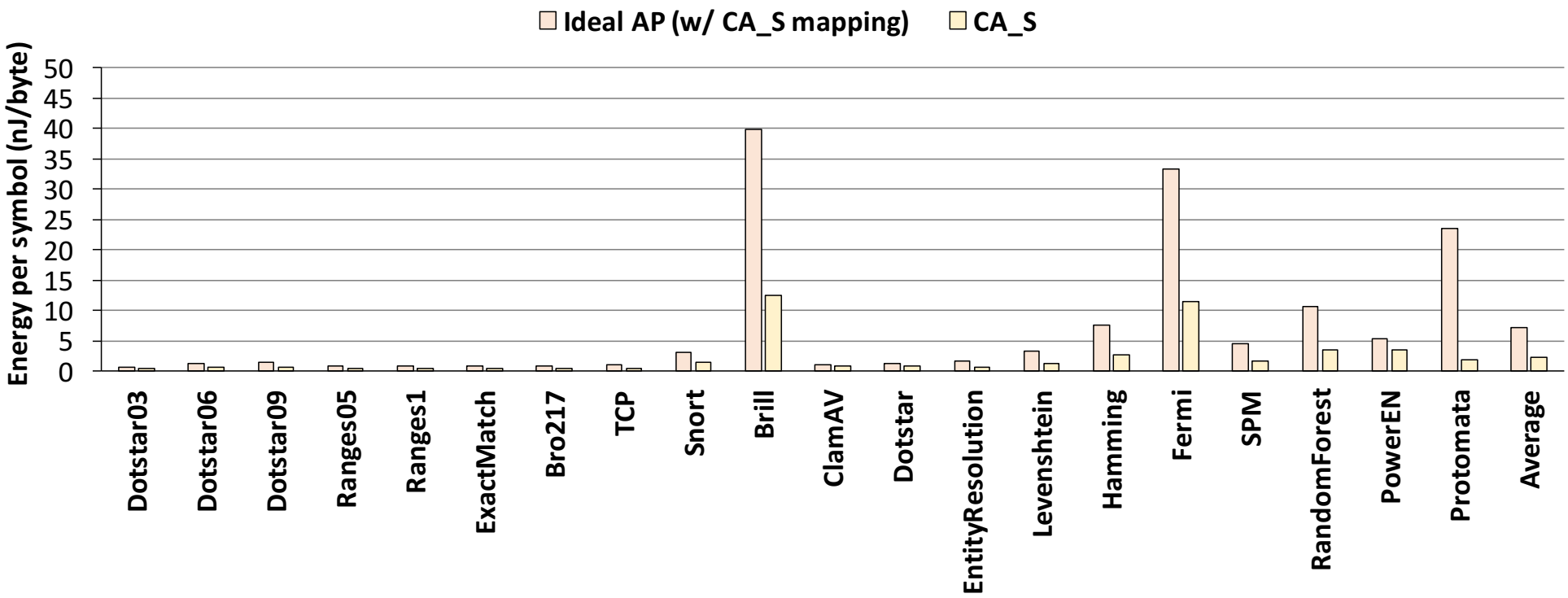
Energy


Ideal AP (w/ CA_P mapping)
Energy per symbol (nJ/byte)
50
45
40
35
30
25
20
15
10
5
0
115.2
96.5
Dotstar03
Dotstar06
Dotstar09
Ranges05
Ranges1
ExactMatch
Bro217
TCP
Snort
Brill
ClamAV
Dotstar
EntityResolution
Levenshtein
Hamming
Fermi
SPM
RandomForest
PowerEN
Protomata
Average

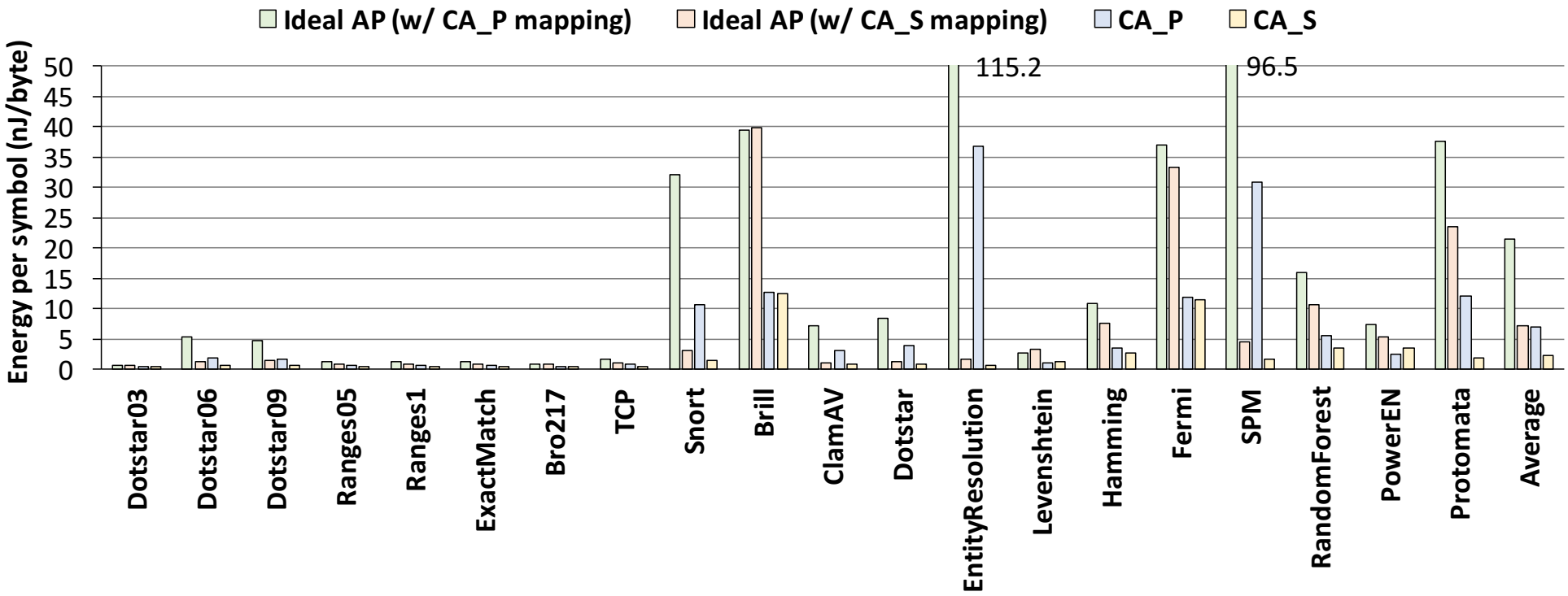
Energy


Ideal AP (w/ CA_P mapping)
CA_P
Energy per symbol (nJ/byte)
50
45
40
35
30
25
20
15
10
5
0
115.2
96.5
Dotstar03
Dotstar06
Dotstar09
Ranges05
Ranges1
ExactMatch
Bro217
TCP
Snort
Brill
ClamAV
Dotstar
EntityResolution
Levenshtein
Hamming
Fermi
SPM
RandomForest
PowerEN
Protomata
Average

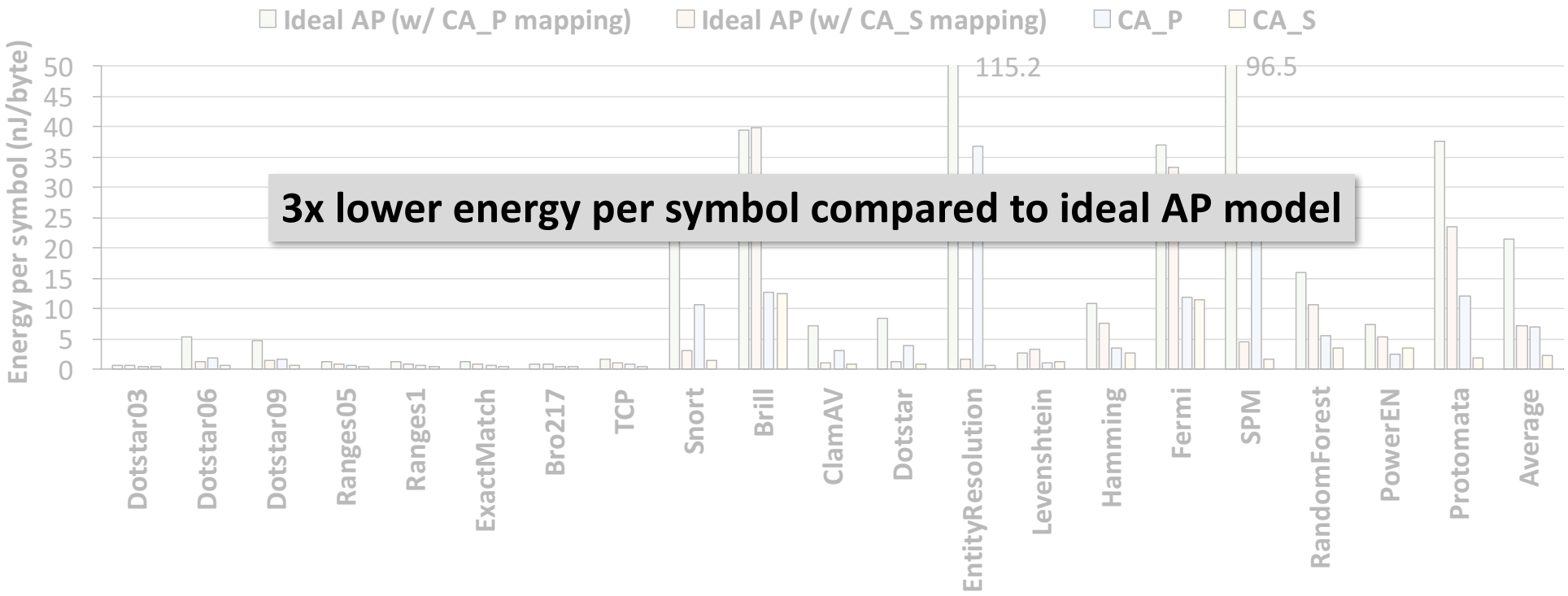
Energy


Ideal AP (w/ CA_S mapping)
CA_S
Energy per symbol (nJ/byte)
50
45
40
35
30
25
20
15
10
5
0
Dotstar03
Dotstar06
Dotstar09
Ranges05
Ranges1
ExactMatch
Bro217
TCP
Snort
Brill
ClamAV
Dotstar
EntityResolution
Levenshtein
Hamming
Fermi
SPM
RandomForest
PowerEN
Protomata
Average

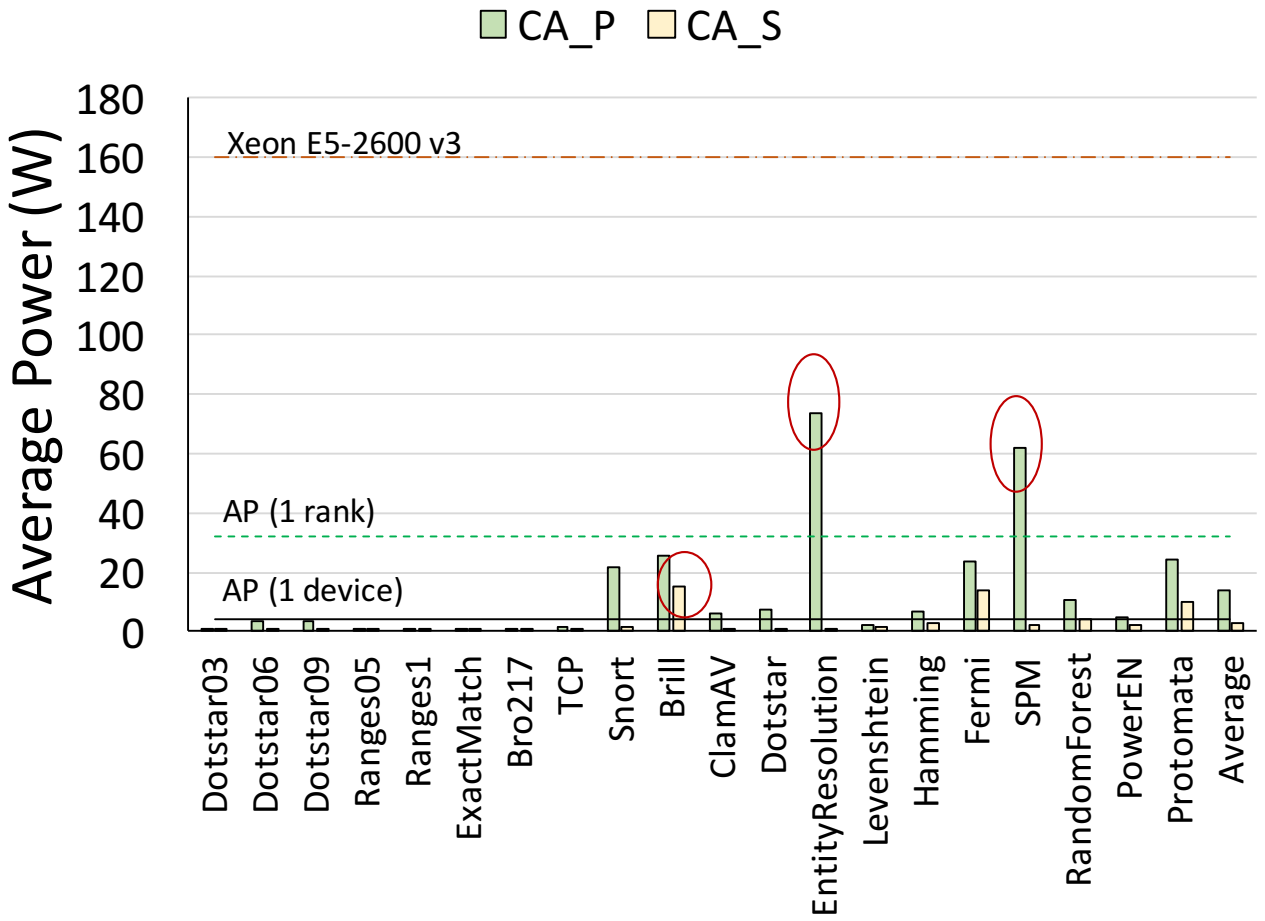
Energy


Ideal AP (w/ CA_P mapping)
Ideal AP (w/ CA_S mapping)
CA_P
CA_S
Energy per symbol (nJ/byte)
50
45
40
35
30
25
20
15
10
5
0
115.2
96.5
Dotstar03
Dotstar06
Dotstar09
Ranges05
Ranges1
ExactMatch
Bro217
TCP
Snort
Brill
ClamAV
Dotstar
EntityResolution
Levenshtein
Hamming
Fermi
SPM
RandomForest
PowerEN
Protomata
Average

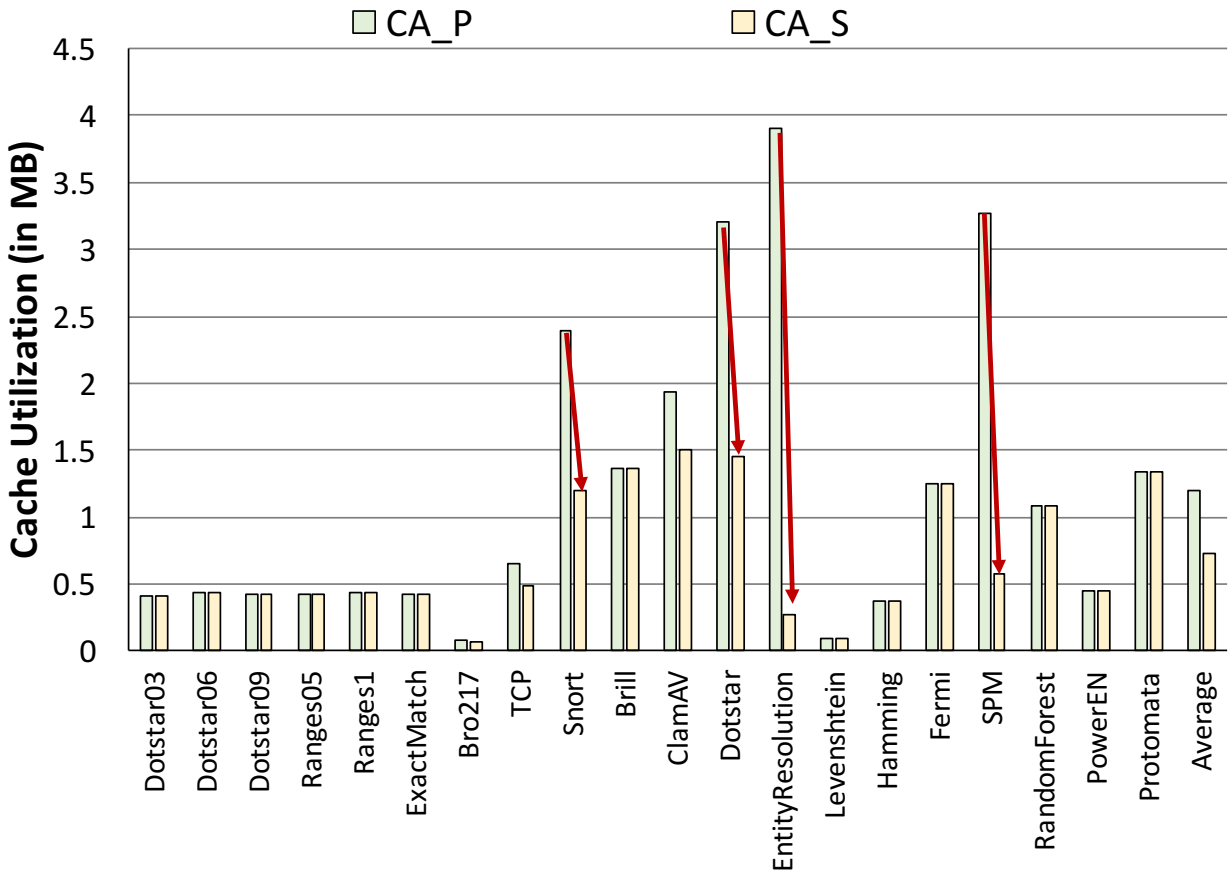
Energy


Ideal AP (w/ CA_P mapping)
Ideal AP (w/ CA_S mapping)
CA_P
CA_S
Energy per symbol (nJ/byte)
50
45
40
35
30
25
20
15
10
5
0
115.2
96.5
3x lower energy per symbol compared to ideal AP model
Dotstar03
Dotstar06
Dotstar09
Ranges05
Ranges1
ExactMatch
Bro217
TCP
Snort
Brill
ClamAV
Dotstar
EntityResolution
Levenshtein
Hamming
Fermi
SPM
RandomForest
PowerEN
Protomata
Average

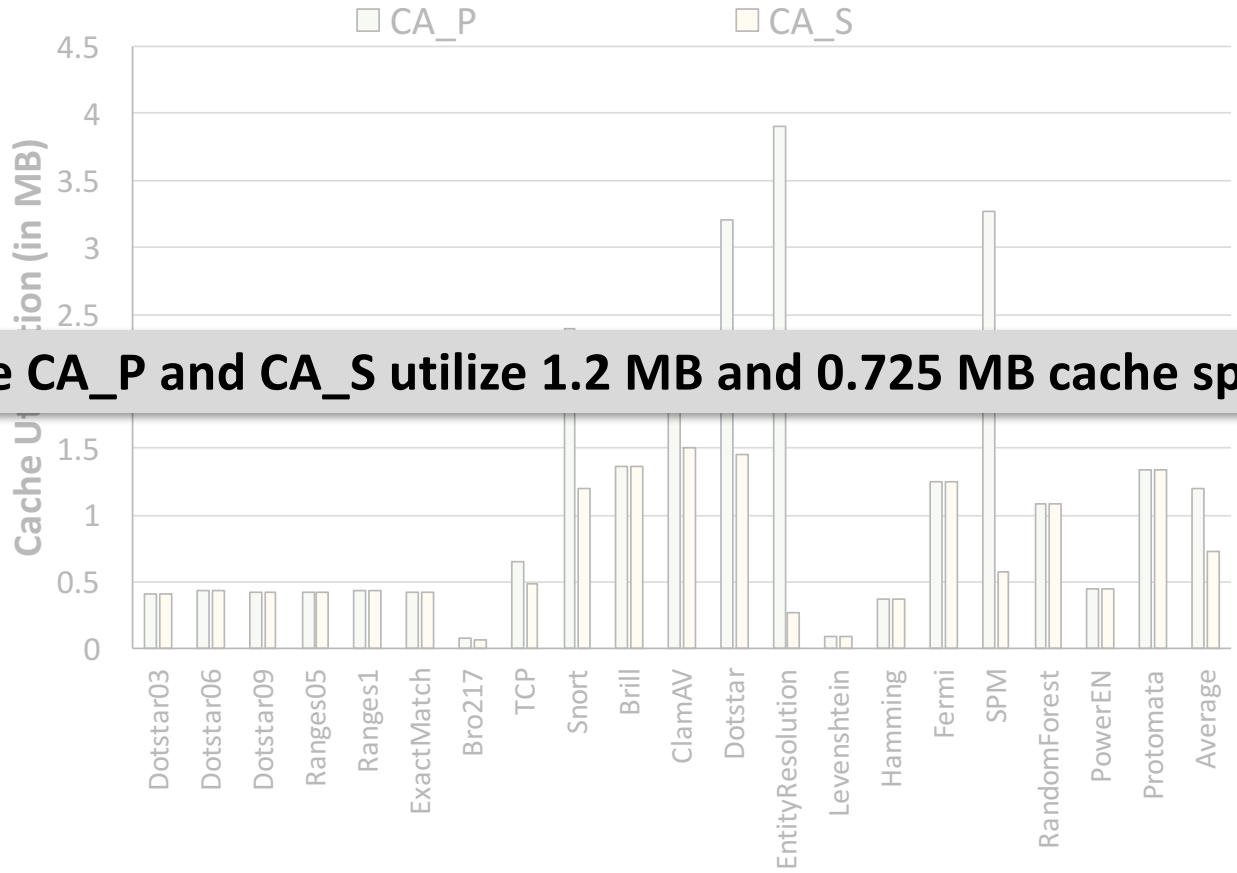
Power


CA_P
CA_S
Average Power (W)
180
160
140
120
100
80
60
40
20
0
Xeon E5-2600 v3
AP (1 rank)
AP (1 device)
Dotstar03
Dotstar06
Dotstar09
Ranges05
Ranges1
ExactMatch
Bro217
TCP
Snort
Brill
ClamAV
Dotstar
EntityResolution
Levenshtein
Hamming
Fermi
SPM
RandomForest
PowerEN
Protomata
Average

Cache Utilization



Cache Utilization



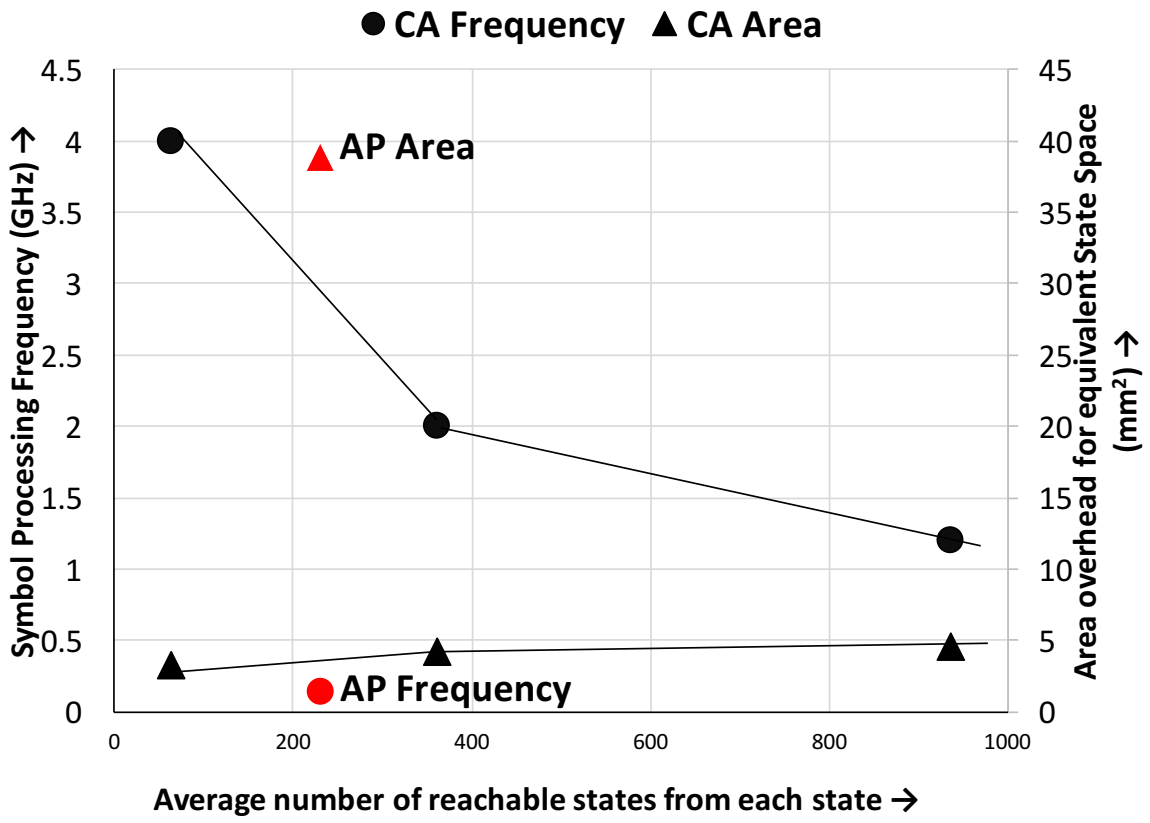
On average CA_P and CA_S utilize 1.2 MB and 0.725 MB cache space respectively

Impact of optimizations

Design	Frequency	w/o SA cycling	w/ H-bus
CA_P	2 GHz	1 GHz	1.5 GHz
CA_S	1.2 GHz	500 MHz	1 GHz

Operating frequency 7.5-11x better than AP even with H-bus

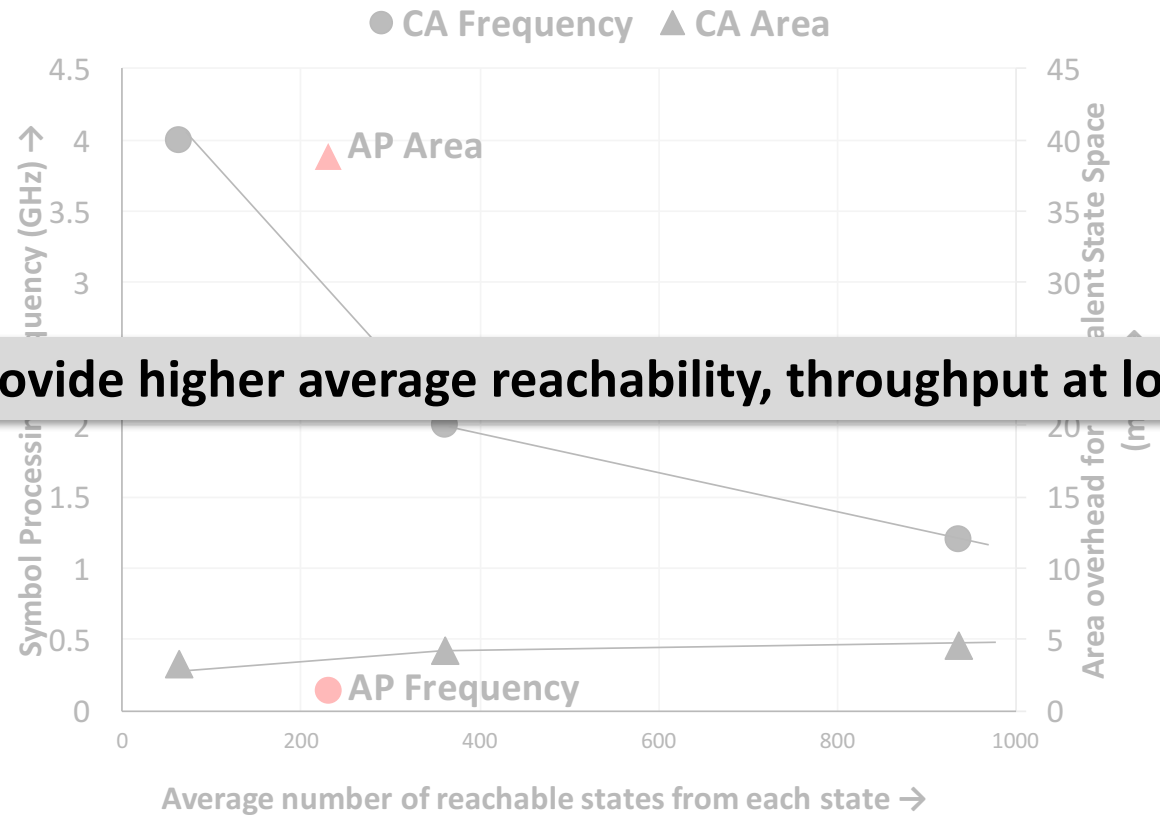
Design Space



AP - 48k states

CA – 32k states

Design Space



CA designs provide higher average reachability, throughput at low area costs

Cache Automaton Summary

✓ SRAM based last-level caches can accelerate automata processing by 9-15 x over Micron AP

Cache Automaton

Accelerating state match — Sense-amplifier cycling

Accelerating state transition — Programmable 8-T SRAM switch based interconnect

Compiler to automate mapping NFAs to cache arrays

In-Memory Automata Processing

Thank you!

Arun Subramaniyan and Reetuparna Das

Assistant Professor

University of Michigan – Ann Arbor