Automata Processing:
Massively-Parallel Acceleration for Approximate Pattern Matching and String Processing

Kevin Skadron
Department of Computer Science

Mircea Stan
Charles R. Brown Department of Electrical & Computer Engineering

UNIVERSITY of VIRGINIA

THE CENTER FOR AUTOMATA PROCESSING

Credits

- People:
  - Research Scientists: Tho Nguyen and Ke Wang
  - Visiting Faculty: Xiaoping Huang
  - Postdoctoral research associates: Mohamed Aly, Vinh Dang
  - Graduate Students: Kevin Angstadt, Chunkun Bo, Nathan Brunelle, Deyuan Guo, Mateja Putic, Elaheh Sadredini, Tom Tracy, Jack Wadden, Ted Xie
  - Undergraduate Student: Sanil Rao
  - Micron collaborators: Paul Dlugosch, Terry Leslie, Dan Skinner, Matt Tanner, Matt Grimm, Paul Glendenning, and many others
  - ARI liaison: Rob Jones
- Funding – this work was supported in part by:
  - Micron
  - C-FAR, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA
  - Virginia CIT
  - NSF
  - ARCS Fellowship (Wadden)

What is Automata Processing?

- Pattern Matching!
  - Most commonly (but not limited to!) regular expression processing
  - E.g.,
    - (1*01*01*)*
    - /<OBJECT\s+[^>]*classid\s*=\s*[\x22\x27]?\s*clsid\s*\x3a \s*\x7B?\s*A105BD70-BF56-4D10-BC91-41C88321F47C/si
- Many applications
  - Deep packet inspection, virus scanning, file carving, etc.
  - But also association rule mining, bioinformatics, etc.
  - Sometimes more easily expressed as automata, not regex

Ex: Brill Part-of-Speech (POS) Tagging

ICSC '15, IEEE BigData '15

- A task in Natural Language Processing (NLP)
- Grammatical tagging of words in text (corpus)
  - E.g.

Cats love dogs. -> [POS Tagger] -> Cats/noun love/verb dogs/noun ./.

- Complicated:
  - E.g. I book tickets. -> *book*: Noun? Verb?
- Baseline tagging:
  - Tag each word to its most frequent tag based on training corpus

Brill Tagging

- A two-stage tagging technique [3]
- Stage 1: Baseline tagging
- Stage 2: Update tags based on some rules (AP)
  - Example rule: NN VB PREVTAG TO

*… to/TO conflict/**NN** with/IN … ->* Apply the Rule *-> … to/TO conflict/**VB** with/IN …*

If current tag is NN, previous tag is TO, update current tag to VB

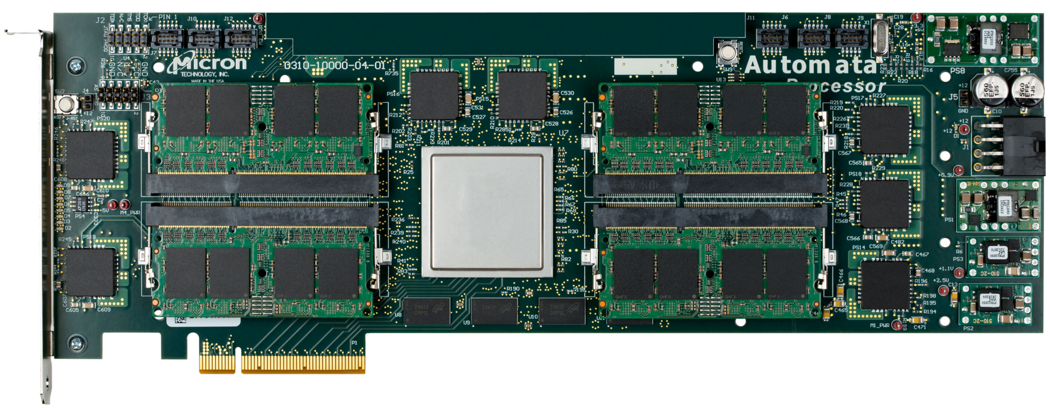
This is easily represented as regular expressions

➢ Can achieve high speedup, use many more (machine-learned) rules, achieve high accuracy

[3] Brill, Eric. "Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging." Computational Linguistics 21.4 (1995): 543-565.

The Automata Processor

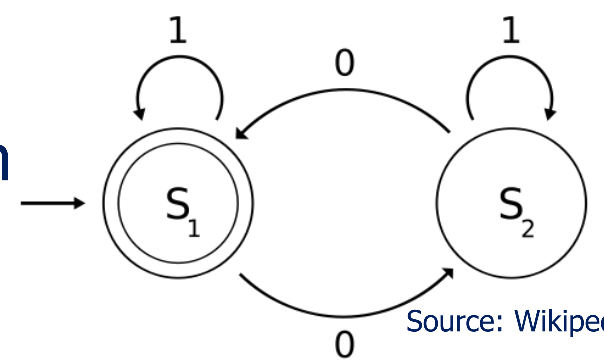
- Hardware accelerator specifically for symbolic pattern matching
- Hardware implementation of *non-deterministic finite automata* (*NFA*) (plus some extra features)
- A highly parallel, reconfigurable fabric comprised of ~50,000 pattern-matching elements per chip. First-generation boards have 32 chips, giving ~1.5M processing elements
- Exploits the very high and natural level of parallelism found in memory arrays
- High speedup potential motivates revisiting many algorithms to leverage automata processing
- On-board FPGA will allow sophisticated processing pipelines



AUTOMATA PROCESSING

What is an NFA?

- *A finite automaton* is a set of states and transition rules that respond to input
- Recognizes *regular languages,* e.g. (1*01*01*)*
- *Non-determinism* (NFA) allows multiple concurrent paths through the automaton
  - (Non-determinism != stochastic)
  - This is very powerful, handles combinatorial problems, checks many possibilities concurrently
  - Avoids exponential cost of DFA (deterministic finite automaton)
- AP adds counters, Boolean elements

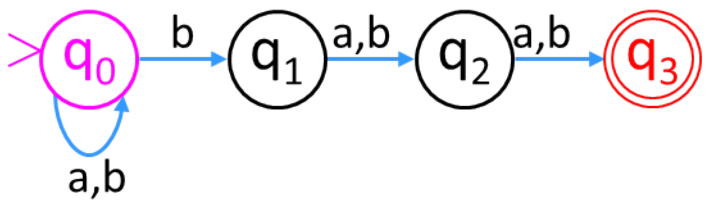


Source: Wikipedia

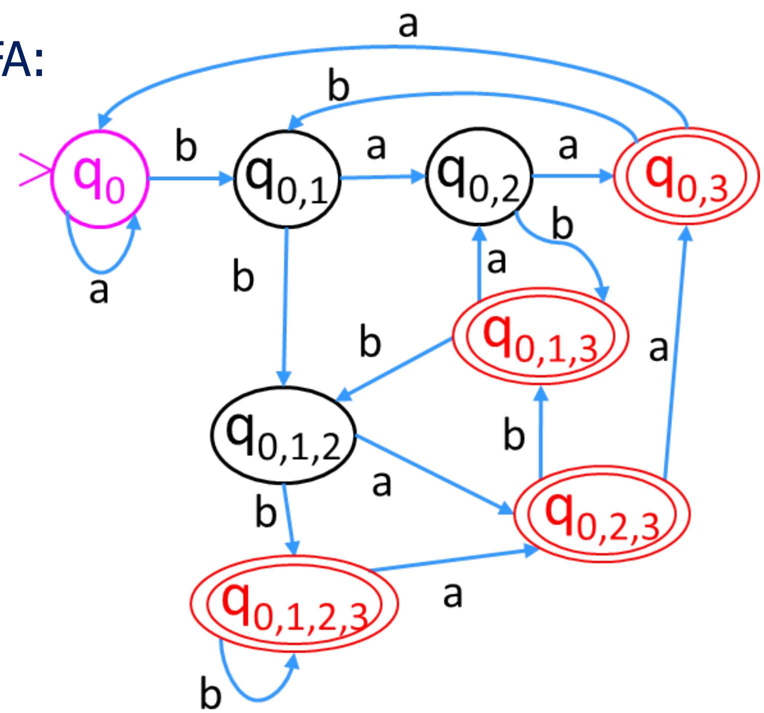
NFA vs. DFA

- Any nondeterministic machine can be modeled as deterministic – at the expense of exponential growth in states
- Ex: 3rd-to-last character in an a-b string is a "b"

NFA:



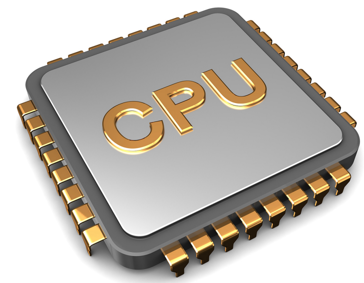
DFA:



- NFA allows multiple active states
  - NFA hardware is highly parallel
- NFA hardware's advantage increases when large number of states active

Architectures for Automata Processing

- Automata processing (regular expression processing) requires:
  - Lots of *irregular* parallelism
  - Massively high memory bandwidth
  - Low-latency access

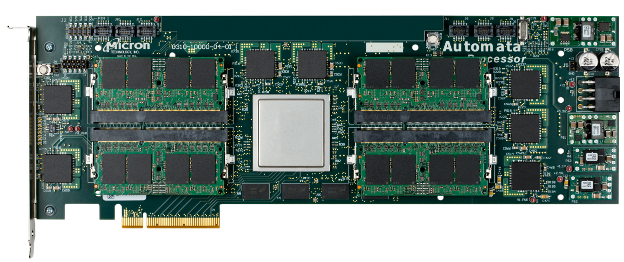


- We explore automata-based computation on a variety of parallel architectures:
  - Multi-core CPUs
  - Many-core Intel's XeonPhi accelerators
  - SIMD-based graphics processing units (GPUs)
  - Field programmable gate arrays (FPGAs)
  - Automata Processor (AP)

Multi-core CPUs, XeonPhi, GPUs: Von Neumann

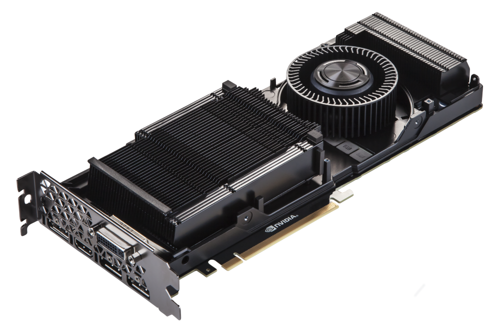
FPGAs, AP: Spatial

XeonPhi



AP

FPGA



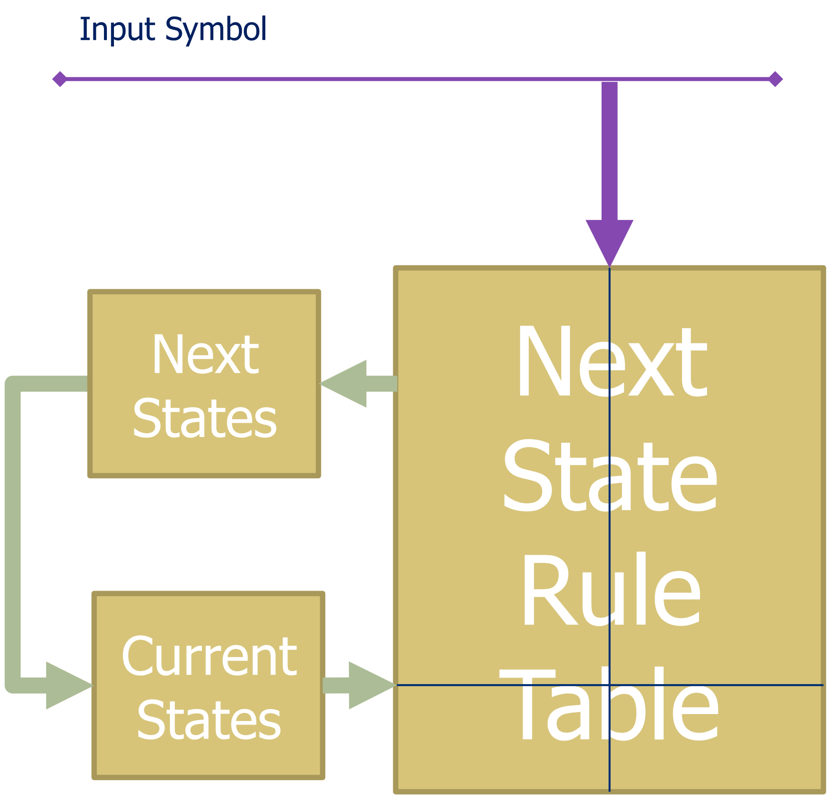
GPU

Outline

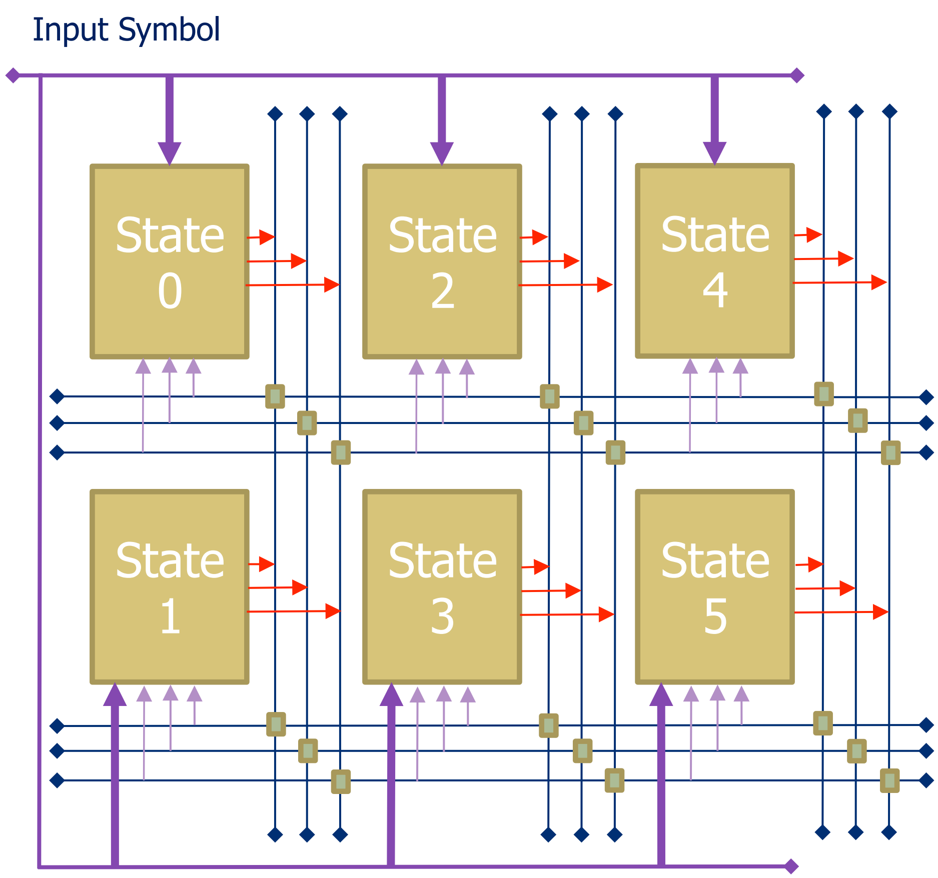
- Issues in automata/regex processing
  - Why von Neumann architectures struggle with large regex rulesets
- Overview of AP architecture
  - Why spatial architectures are a good fit
- Ongoing research and results
  - Why Automata Processing is about much more than regex processing
  - 10X-100sX speedup

Spatial architectures are a better fit for automata processing

Von Neumann (CPU/GPU)



Spatial, data-flow (FPGA/AP)

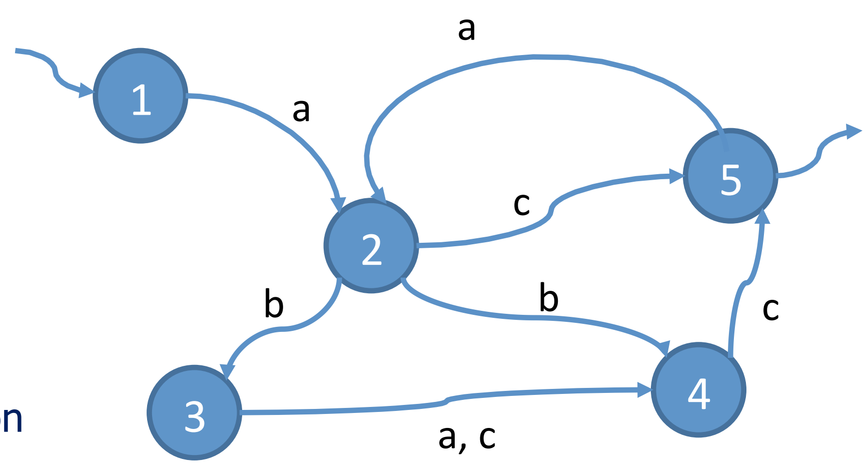


Spatial vs. Von Neumann Architectures

- Von Neumann (CPUs, GPUs)
  - Table lookup: for current state(s), identify correct transition(s) based on current input
    - NFA: potentially many lookups per cycle
      - Bad for most memory architectures
    - DFA: one lookup per cycle
      - But DFAs suffer exponential blowup, quickly blow out on-chip caches – especially with large # rules
      - Compression approaches help
    - Hybrid: recognizes that many RegEx's have low active count, so NFAs ok; bail out to DFA if active count exceeds # memory ports
    - Very difficult to build efficient DFAs with many rules
- Spatial (AP, FPGAs): Direct HW implementation of NFA!

CPU-based Engine - VASim (CPU/XeonPhi)
IISWC'16

- VASim is a high-performance, open-source Virtual Automata SIMulator for automata processing research
- Optimized version of the classic NFA algorithm:
  - Looking up appropriate transition rules in memory for each symbol in the input stream based on active state(s) in the finite automaton and executing those transitions in the automaton
  - Considering *only* automata states that are *active*
  - *Optimized data structures* for low-overhead parallel execution
  - Parametrically *multithreaded in two dimensions*: separate automata and different sections of the input symbol stream
  - VASim is within ~2x of HyperScan
    - We think we can beat HyperScan – we still have lots of optimizations yet to include



Alphabet size

Number of states

	a	b	c
1	{2}		
2		{3,4}	{5}
3	{4}		{4}
4			{5}
5	{2}		

Automata are traditionally used to compute large regular expression rulesets

- **Snort** network intrusion detection ruleset
- **BRO** network intrusion detection ruleset
- **ClamAV** virus signature ruleset

- Many other applications
  - Eg, scanning text

Line-rate, streaming deep packet inspection requires fast automata processing of tens of thousands of rules

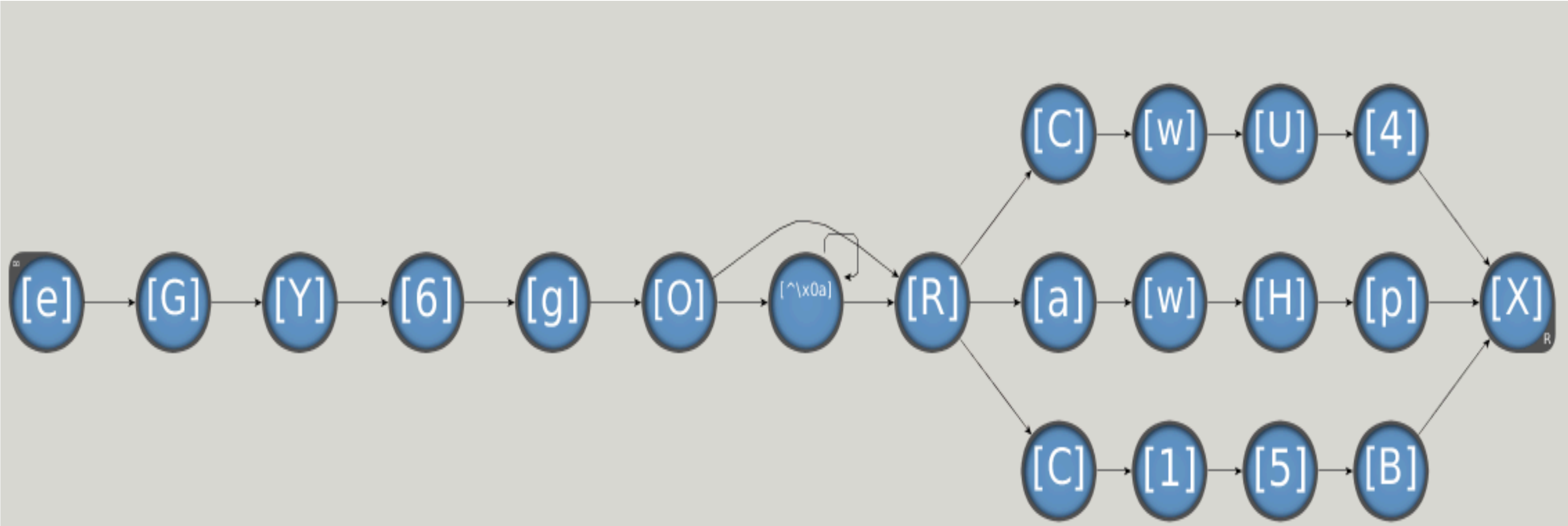
…but DFAs struggle with large # rules

Regular-expression-derived automata tend to have similar properties

- Long literals
- Low activity factors

Example synthetic regular expression pattern from PowerEN (IBM)

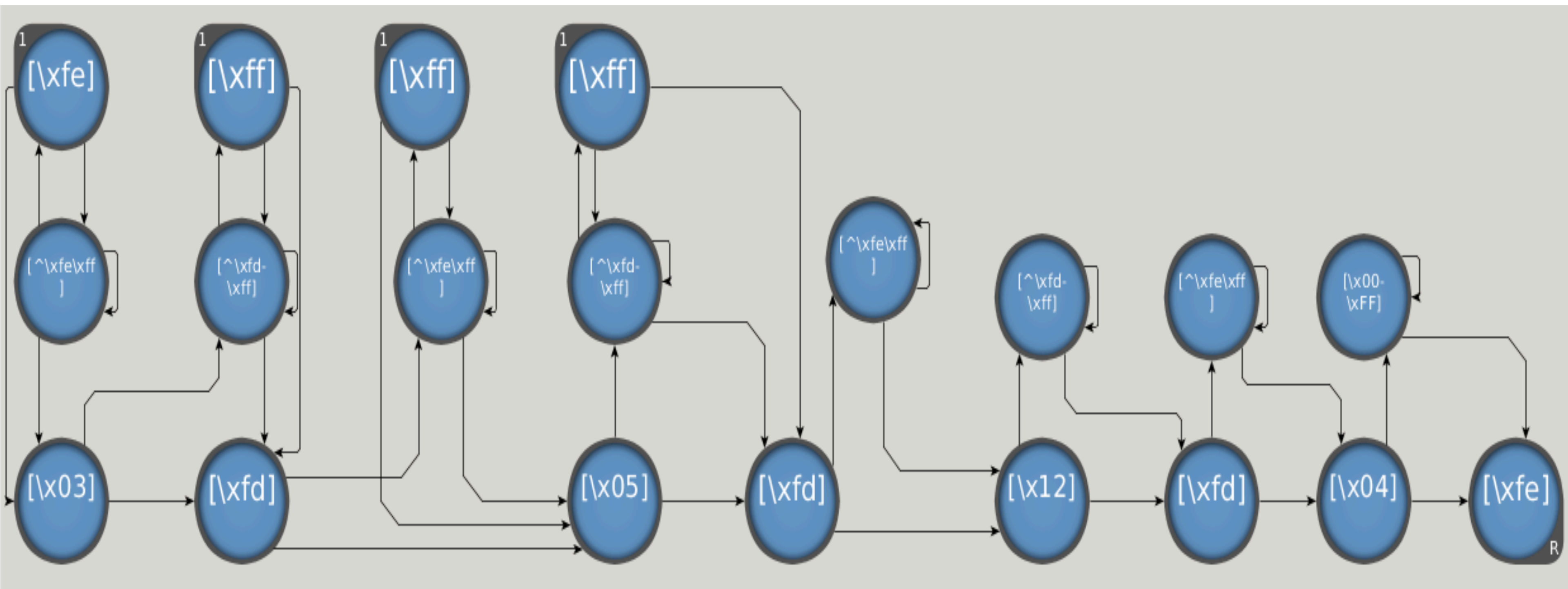
```
eGY6gO.*R(CwU4|C15B|awHp)X
```



Other automata-based applications (not regex-based) can have more diverse behavior

- Higher activity factors
- More complex topography, transition rule complexity
- More dynamic variation in behavior

Example: Sequential Pattern Mining Automata



Need Diverse Benchmarks for Automata Processing

- ANMLZoo is a collection of 14 diverse automata benchmarks and standard inputs that can be used to evaluate automata processing engines and architectures (IISWC'16)

Regular Expression Rulesets:
- Snort
- ClamAV
- Dotstar (Becchi et al.[1])
- PowerEN[2]
- Protomata
- Brill Tagging

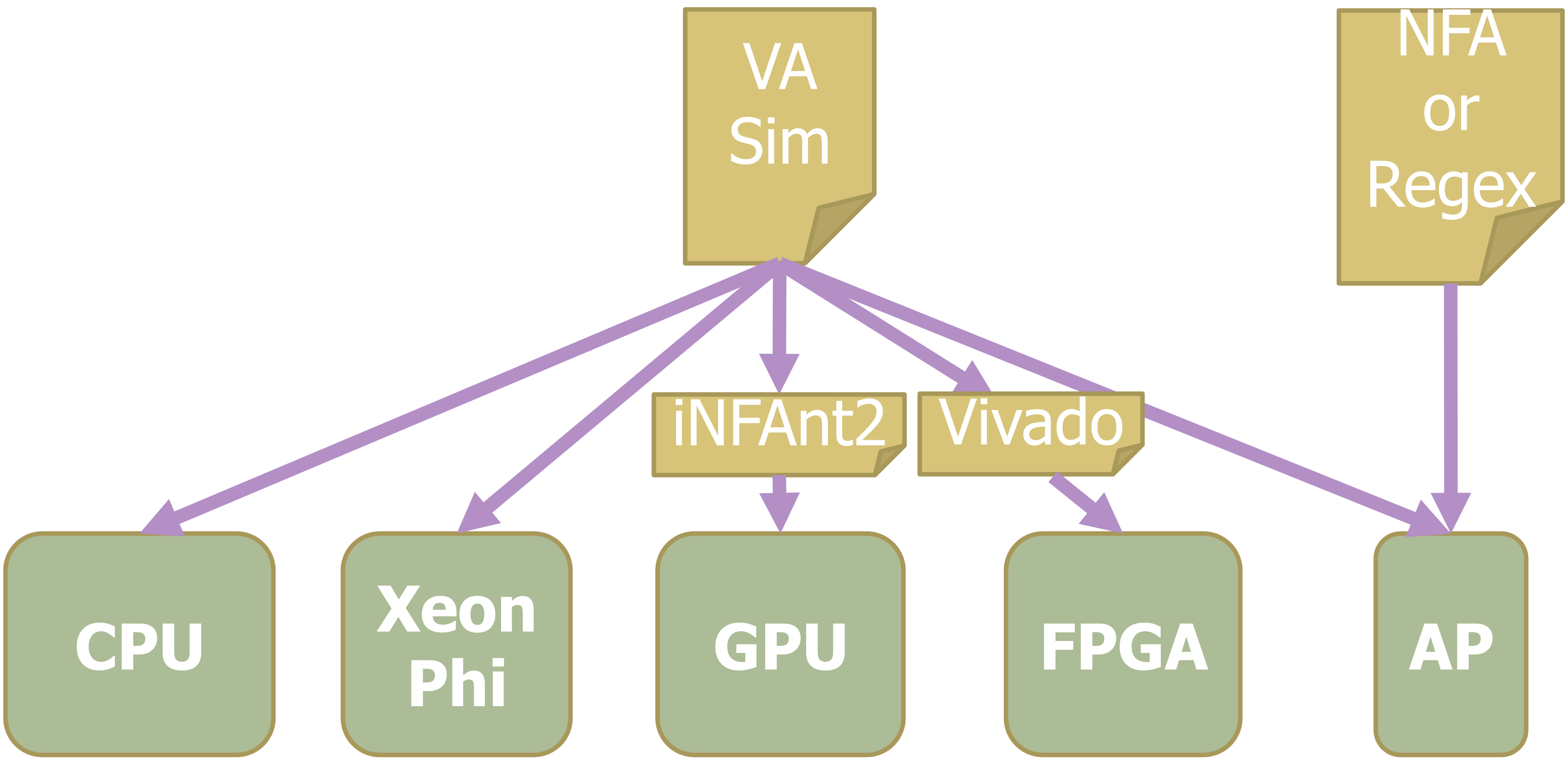
Mesh Automata:
- *Hamming
- *Levenshtein

Synthetic:
- *Block Rings
- *Core Rings

"Widgets":
- Sequential Pattern Mining
- Fermilab Particle Tracking
- Entity Resolution
- Random Forest

**Parametric code generation tools are included*

VASim is *also* a collection of software engines for varying architectures



VASim+ANMLZoo are being open sourced

VAsim:

- https://www.github.com/jackwadden/VASim
- Our optimized GPU engine ready but pending license issues
- FPGA back end will be released soon

Benchmarks:

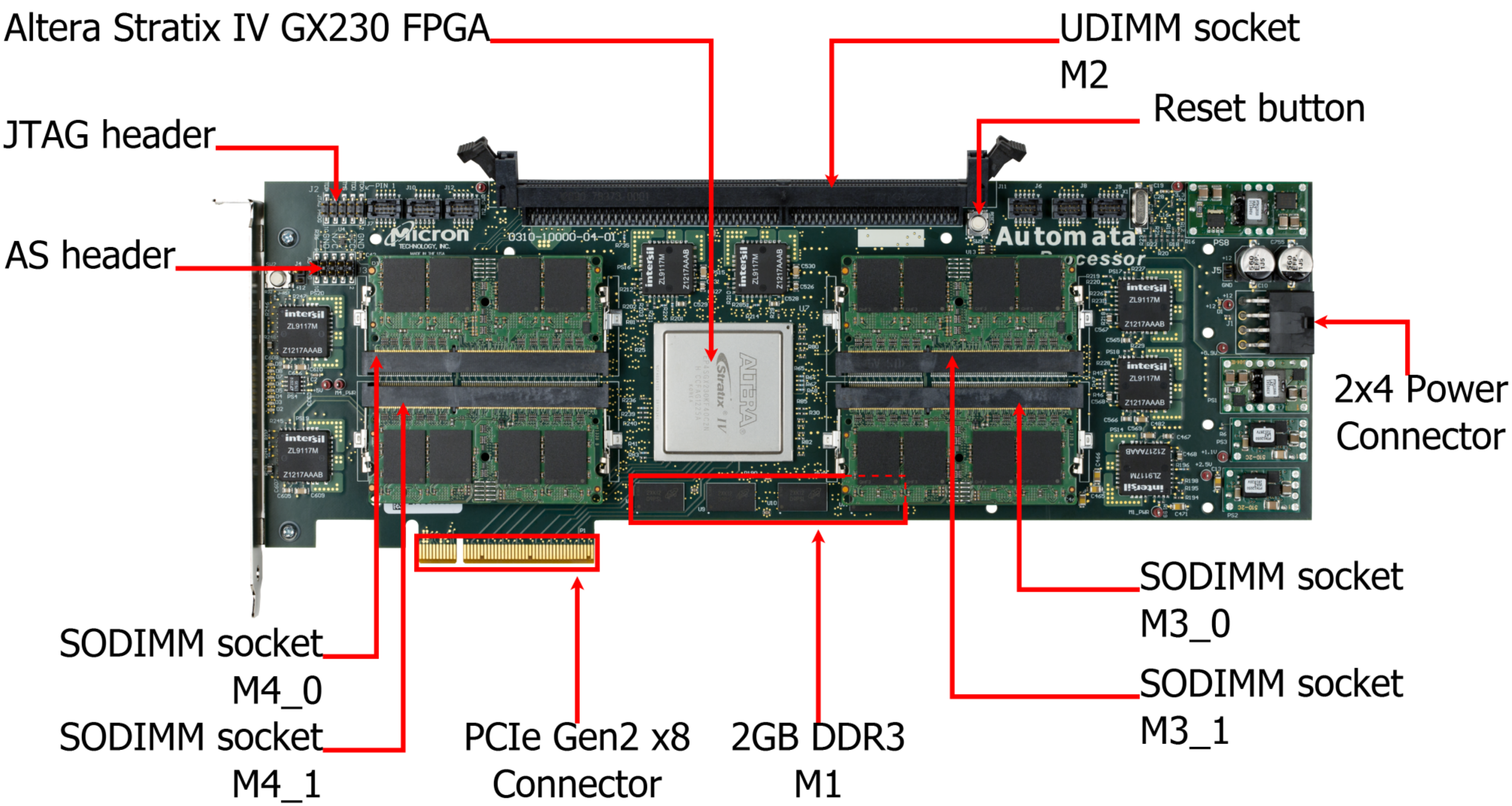
- ANMLZoo is a mixed license benchmark suite, with some applications awaiting permission (12/14 released so far, others pending license issues)
- https://github.com/jackwadden/ANMLZoo

Outline

- Issues in automata/regex processing
  - Why von Neumann architectures struggle with large regex rulesets
- Overview of AP architecture
  - Why spatial architectures are a good fit
- Ongoing research and results
  - Why Automata Processing is about much more than regex processing
  - 10X-100sX speedup

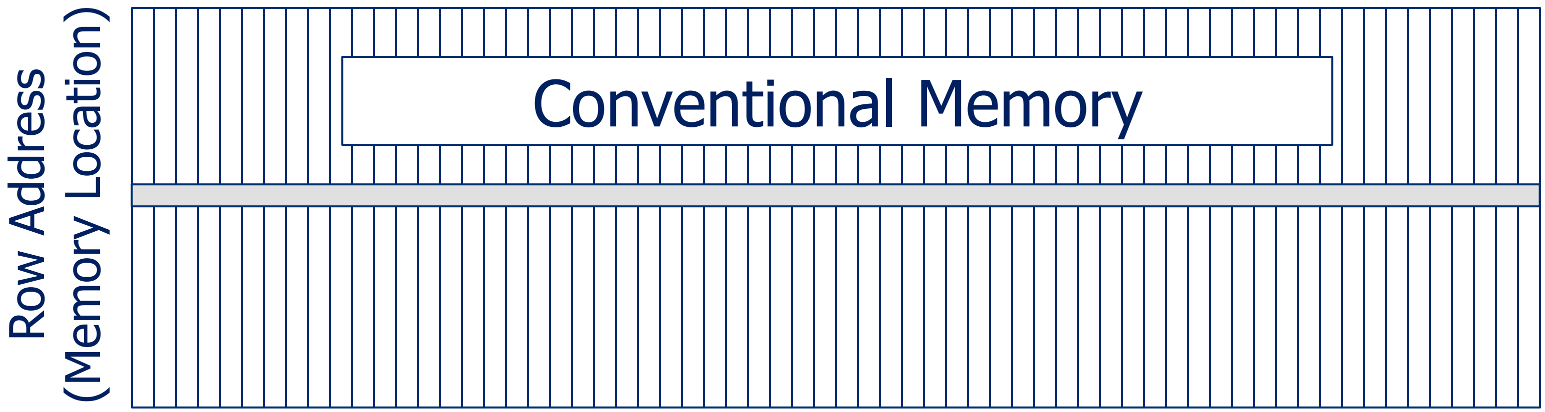
Automata Processor Development Board

PCIe, 4 Ranks, 32 chips, 1.5M STEs

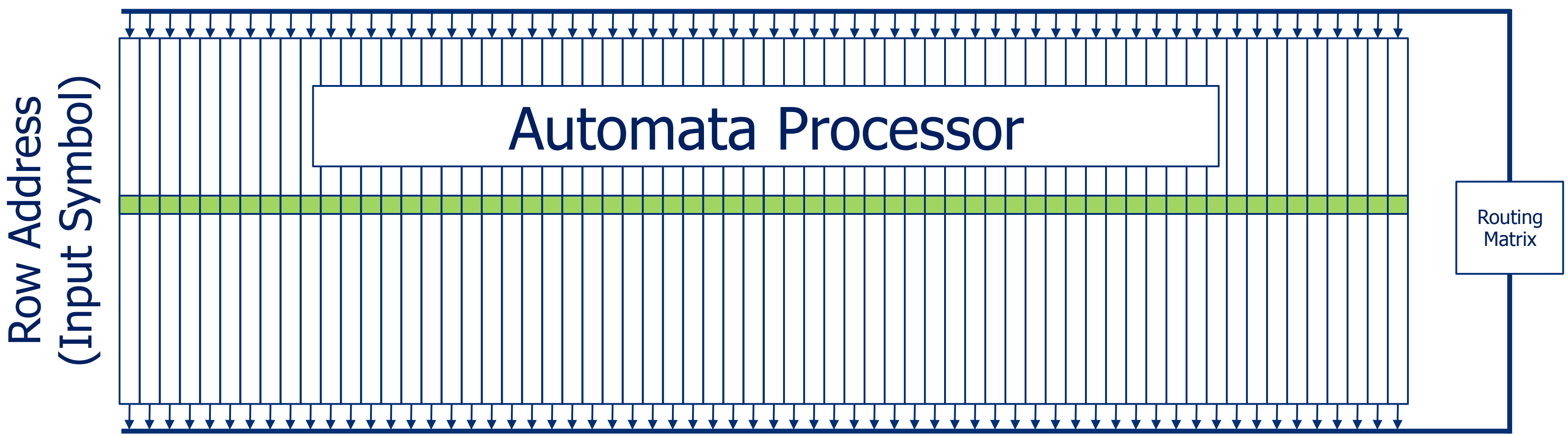


- The FPGA provides substantial flexibility to augment the NFAs with other types of computation

Automata Processor – Basic Operation



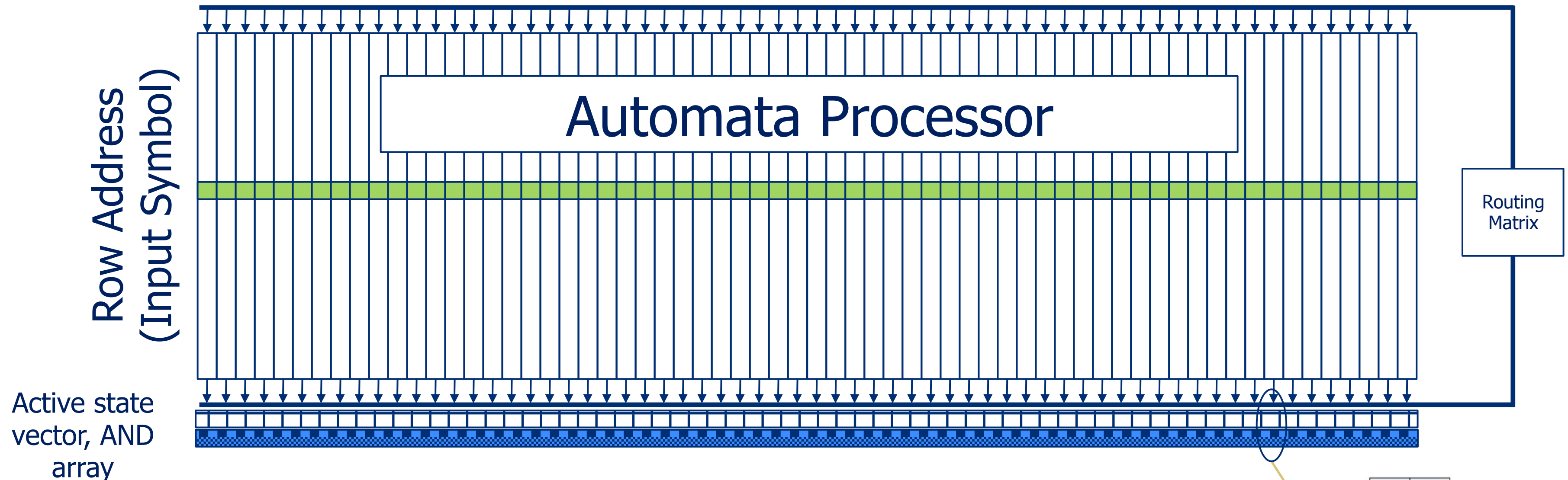
Row Access results in **one** word being retrieved from memory.



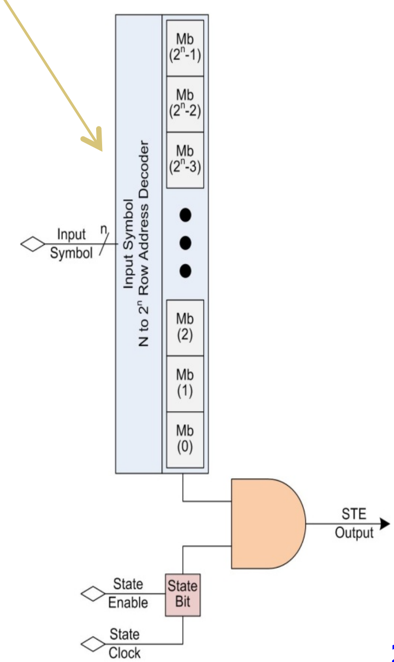
Row Access results in **49,152** match & route operations
(then Boolean AND with "active" bit-vector)

Figures courtesy of Micron

Automata Processor – Basic Operation



- One column = one State Transition Element
- STE "fires" when
  - Symbol match
  - AND the STE is active
- Row Access results in 49,152 match & route ops

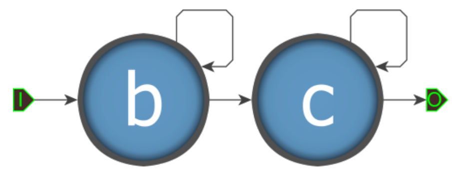


Figures courtesy of Micron

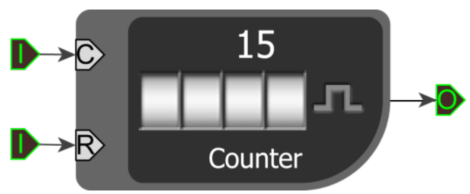
Automata Processor Hardware Building Blocks

per chip

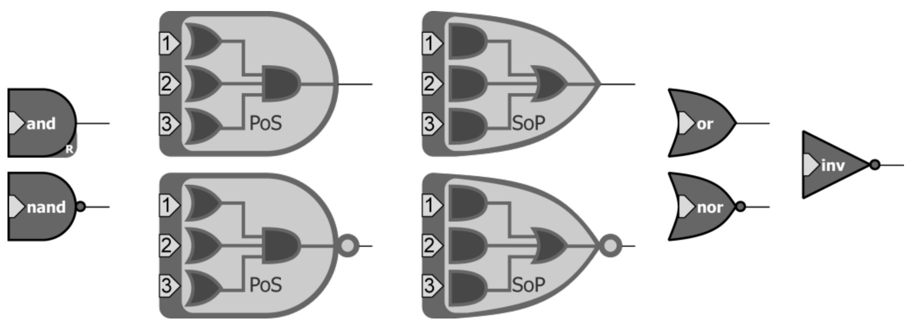
State Transition Element (STE) — 49,152
(note shift in notation)



Counter Element — 768



Boolean Logic Element — 2,304
Nine Programmable Functions



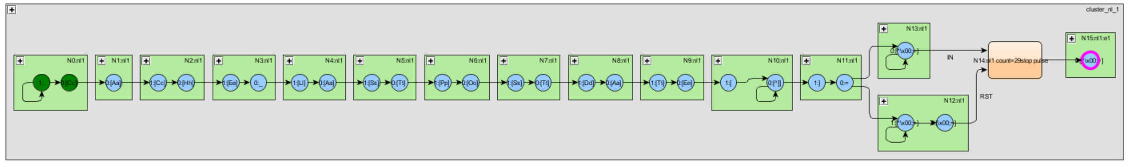
Report buffer — 6,144

Figures courtesy of Micron

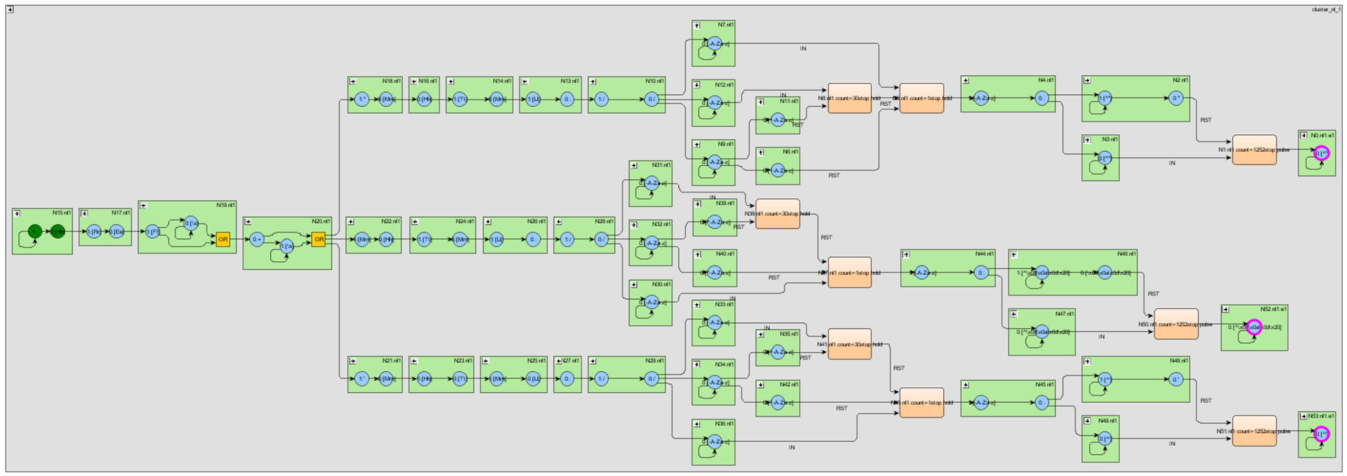
- Important: ALL elements on all chips see input symbol every cycle

Parallel Automata/Rules

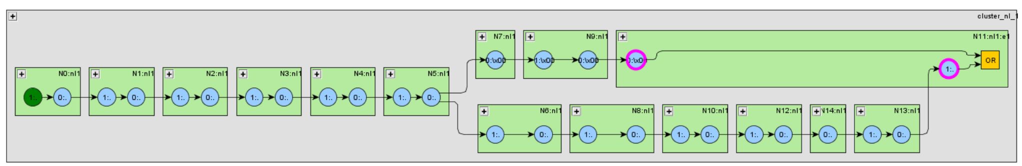
Pattern #1 →



Pattern #2 →



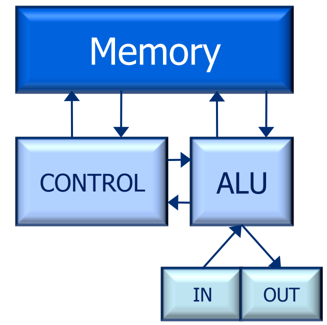
Pattern #3 →



- *Parallelization of automata requires no special consideration by the user. Each automaton operates independently upon the input data stream*
- *NFAs are extremely compact, allowing many parallel rules*

Figures courtesy of Micron

Non von Neumann Parallel Architecture

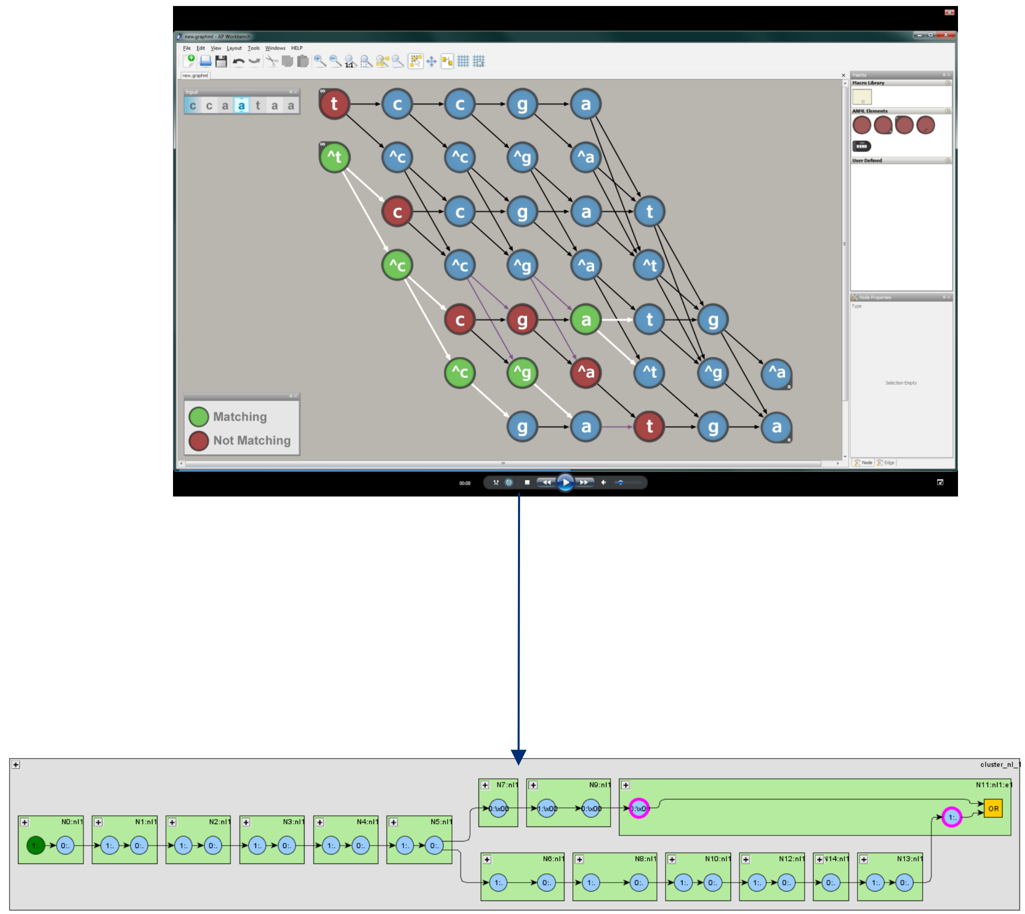


- Spatial architecture avoids the von Neumann bottleneck of instruction fetch and data fetch
  - Instead: hardware reconfiguration and higher density of NFAs vs. DFAs
- Spatial architecture allows massive parallelism
  - Every automaton node can inspect every input symbol
    - Leverages full-row memory access—fundamental insight
  - Can process a new input symbol every clock cycle
  - Approaches efficiency of an *Alternating Finite Automaton*
- Fills the unusual "MISD" role in Flynn's taxonomy

SISD	**SIMD**
MISD	**MIMD**

Programming Options

- Currently, like other PCIe-attached accelerators
  - Offload model, mediated by device driver
- Input
  - RegEx
  - GUI – Workbench
  - C/Python APIs
  - RAPID – C-like language
  - ANML
- Compiling
  - Input → ANML
  - ANML→ Netlist
  - Netlist → Place & route



Figures courtesy of Micron

I/O

- Bandwidth in 1st-gen boards
  - Input side: 1 Gbit per second throughput from input side
    - But >1 Gbit/s possible: board can be partitioned to support multiple, concurrent dataflows, each to a different subset of AP chips
    - Then the limit is the PCIe bandwidth
  - Output side: depends on number of report events generated by the design and the input stream
  - 1 Gb/s per node for highly complex analysis = substantial speedup!

- Note: input limitation is due to DRAM process in 1st-gen
  - Also lower density due to 50nm node
  - These should change in 2nd-gen
  - Logic – enables much higher clock rates and higher density
  - New system architectures allow higher input/output rates

Streaming Analytics

- PCIe offload model puts driver in the critical path
- However, other system architectures are possible
  - E.g., direct data ingress
  - Load “program” (configuration), stream data directly, allow concurrent output
  - Many other possibilities…

Problems Aligned with the Automata Processor

Applications requiring **deep analysis** of **data streams** containing **spatial** and **temporal** information are often impacted by the **memory wall** and will benefit from the **processing efficiency** and **parallelism** of the Automata Processor

Network Security:
- Millions of patterns
- Real-time results
- Unstructured data

Bioinformatics:
- Large operands
- Complex patterns
- Many combinatorial problems
- Unstructured data

Video Analytics:
- Highly parallel operation
- Real-time operation
- Unstructured data

Data Analytics:
- Highly parallel operation
- Real-time operation
- Complex patterns
- Many combinatorial problems
- Unstructured data

So far: 10-100sX speedups possible!

Problems Aligned with the Automata Processor

- AP strengths
  - Complex/fuzzy pattern matching, e.g. regex, edit distance
  - Combinatorial search space (but only with pruning)
  - Highly parallel set of symbolic analysis steps for each input item
  - Unstructured data, unstructured communication
    - Esp. with high fan-out/fan-in
  - These challenges are common in “big data” analytics!
  - Also Markov chains, some neural models

- AP limitations
  - No arithmetic, only counting (but on-board FPGA can help)
  - Changing the “program” requires a reconfiguration step

Outline

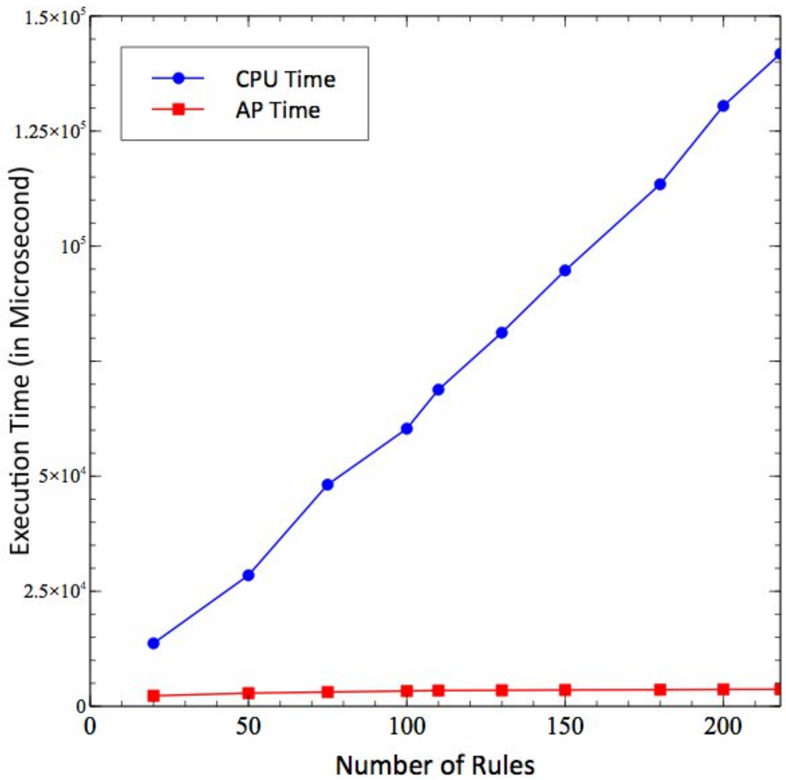
- Issues in automata/regex processing
- Overview of AP architecture
- Ongoing research and results

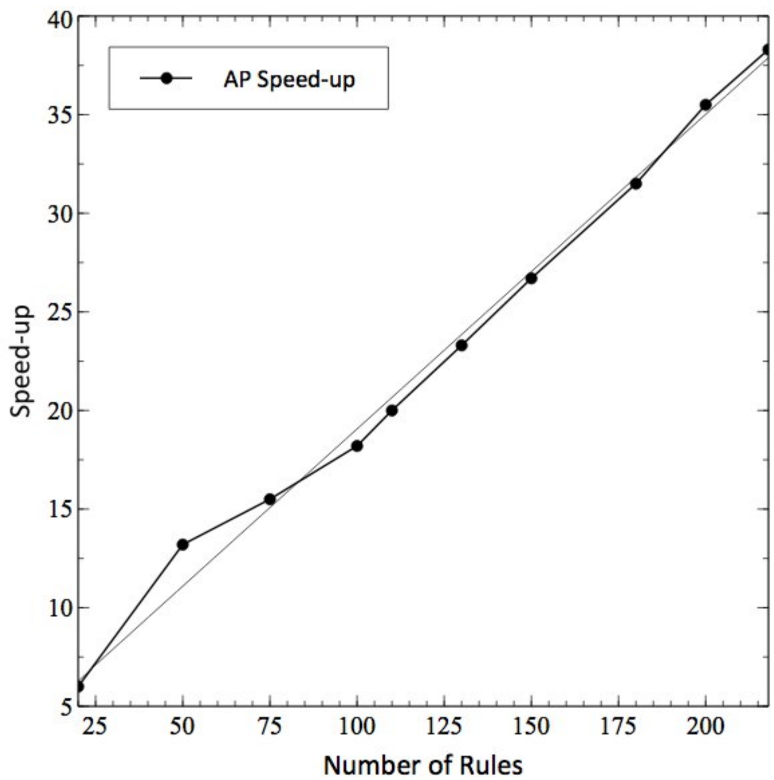
A few examples of ongoing CAP research

- Regular expressions (e.g., Brill tagging)
- Entity resolution
- Association rule mining
- Bioinformatics – CRISPR
- Random Forest
- Markov processes
- Hierarchical temporal memory
- Automata benchmarking

Results – Brill POS Tagging
(ICSC'15, BigData'15)

- Performance of the AP as a function of the number of rules



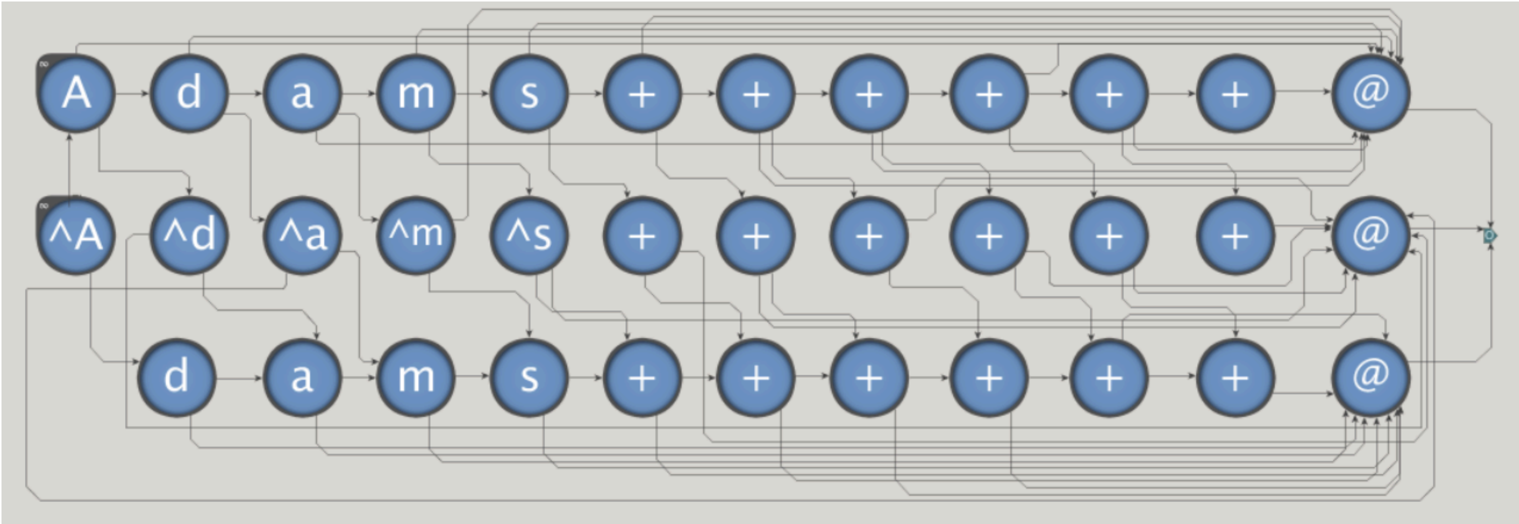


- Our largest dataset: 218 rules
- Maximum number of rules in the literature: 1729 [5]
  - Estimated Speed-up: **276X**

[5] Brill, Eric. "Unsupervised learning of disambiguation rules for part of speech tagging." *Proceedings of the third workshop on very large corpora. Vol. 30. Association for Computational Linguistics*, 1995

Entity Resolution (ER)
IEEE BigData '16

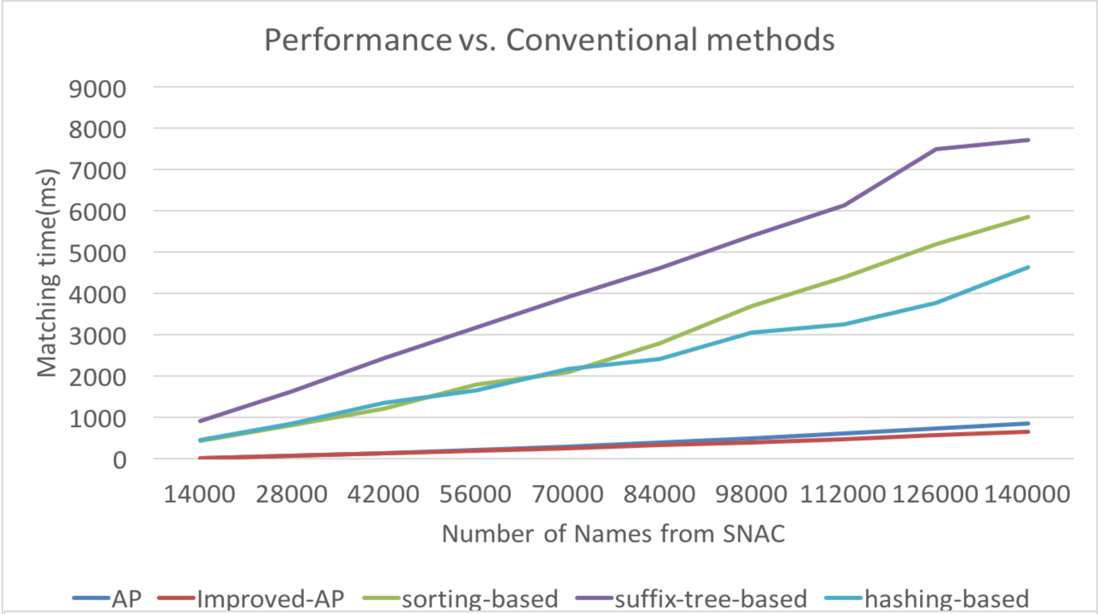
- Identify matching records despite mismatches in key(s)
- E.g., names – typos, transliteration, different formats
  - Qaddaffi, Gaddaffi, etc.
  - FDR; Franklin Delano Roosevelt; Roosevelt, Franklin D., Pres. Roosevelt, etc.
- Handle with variations of Hamming distance macro

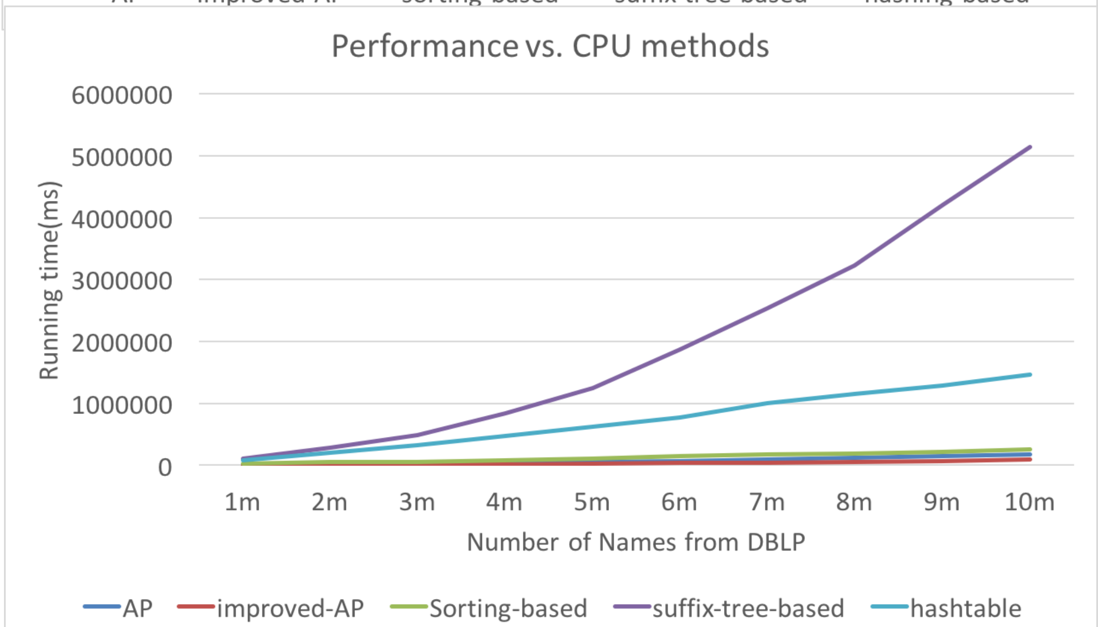


Name "Adams"

Running Time

- Running time of the AP approach increases almost linearly as databases increase
- The AP approach works the best for both SNAC and DBLP databases
- At least 17x speedup is achieved
- These speedups increase with higher edit distance





Results Quality

- **Compression rate:** record number after matching / original record number
- **Correct Pair number:** every two records inside the group is counted as one pair
- **Generalized merge distance:** numbers of merge and split operations to convert results to "correct" results

Method	Comp Rate	Correct Pairs #	Percentage	GMD
Lucene	65.3%	262	80.6%	54
Sorting	71.4%	233	71.7%	63
Hashing	73.2%	213	65.6%	72
Suffix-tree	73.2%	213	65.6%	72
AP	57.2%	292	89.8%	31
Manual	47.4%	325	100%	0

Accuracy for SNAC

Method	Correct Pairs #	Percentage	GMD
Sorting	502	74.4%	183
Hashing	484	71.7%	212
Suffix-tree	484	71.7%	212
AP	615	91.4%	62
Manual	675	100%	0

Accuracy for DBLP

Large Parallelism – String capacity

- The AP can process a large number of strings simultaneously

Mismatches or Gaps allowed \ Pattern length	10	20	30	40	50	60	70	80	90	100
0	150000	75000	50000	37500	30000	25000	21428	18750	16666	15000
1	53571	25862	17045	12711	10135	8426	7211	6302	5597	5033
2	35714	16304	10563	7812	6198	5136	4835	3826	3393	3048
3	28846	12295	7815	5725	4518	3731	3177	2767	2450	2199
4	25862	10135	6302	4573	3588	2952	2508	2180	1928	1728
5	25000	8823	5357	3846	3000	2459	2083	1807	1595	1428

Number of strings that can be processed on one 1st generation AP board

Association Rule Mining, Frequent Itemsets

IPDPS '15

- Widely used building block in data mining to identify *associations*, e.g. frequent itemsets
  - Example: {pen, ink, paper}
- *Support*: # occurrences to qualify
- Applications: market basket analysis, social network analysis, categorization, text mining, anomaly detection, cybersecurity, etc.
  - Ex: Traffic accident analysis: which events are strongly correlated with accidents?
  - Ex: Words, phrases, or other patterns associated with specific concepts
  - Ex: Intrusion detection
- AP can be used for *learning* as well as inference

Apriori Algorithm

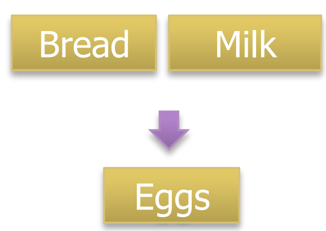
- Classic "a priori" algorithm a good fit for AP
  - Relies on *downward closure:* k-itemset with support N must include a k-1 itemset with support N
  - Identify large itemsets and prune search space by identifying 2-itemsets, then 3-itemsets, etc.
  - AP's large capacity can test many candidate itemsets in parallel
  - Current gen is counter limited
- Compare to Eclat algorithm on CPU
  - Better on CPU than simple a priori

Sequential Pattern Mining (SPM)

ACM CF'16

- Now order among transactions matters (instead of looking at each transaction in isolation)

Trans.	Items
1	<{Bread, Milk}, {Coke}>
2	<{Bread, Milk, Chips}{Beer, Eggs}{Chips}>
3	<{Milk} {Chips} {Beer, Coke}>
4	<{Bread, Milk, Chips}{Beer, Chips}{Beer, Coke, Eggs}>
5	<{Bread, Milk}{Coke}{Chips}{Eggs}>



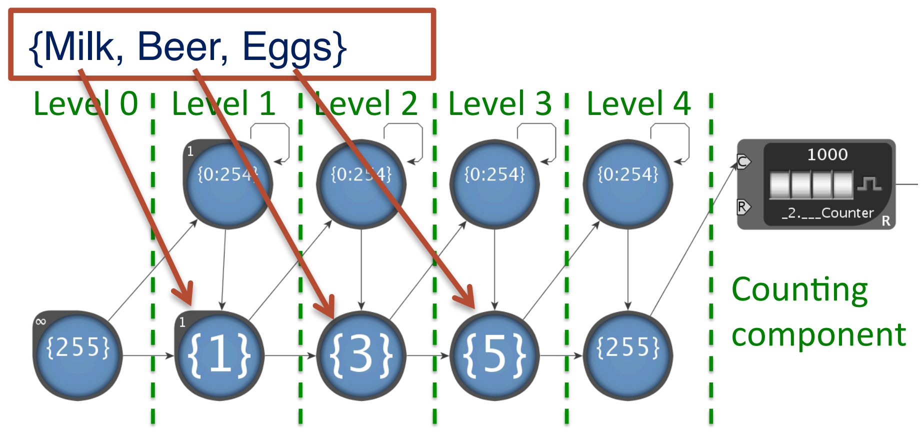
Mapping FIS to the AP

ARM		AP implementation
Item	⟺	Symbol 8-bit or 16-bit
Itemset	⟺	NFA by STEs
Transactions	⟺	Input Stream
Frequency counting	⟺	Counter Element

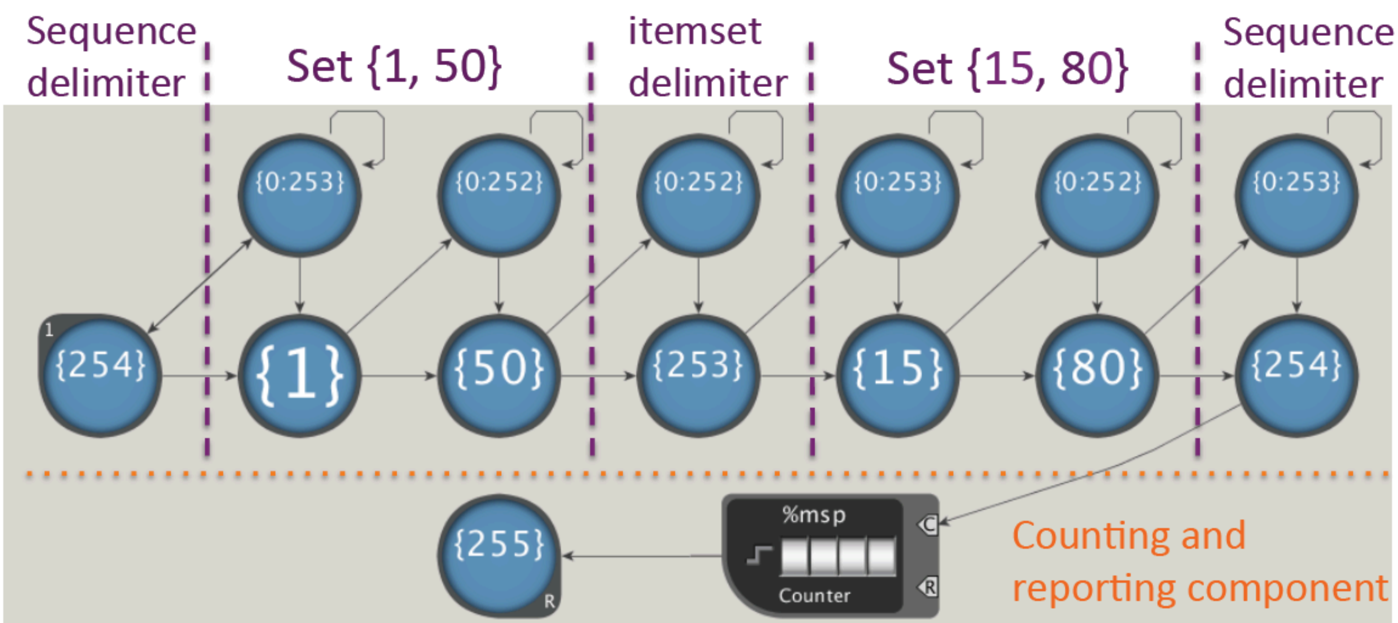
Item	Code
Bread	0
Milk	1
Chips	2
Beer	3
Coke	4
Eggs	5
Separator	255(\xFF)

Transaction stream:

01\xFF0235\xFF12345\xFF01234\xFF0124



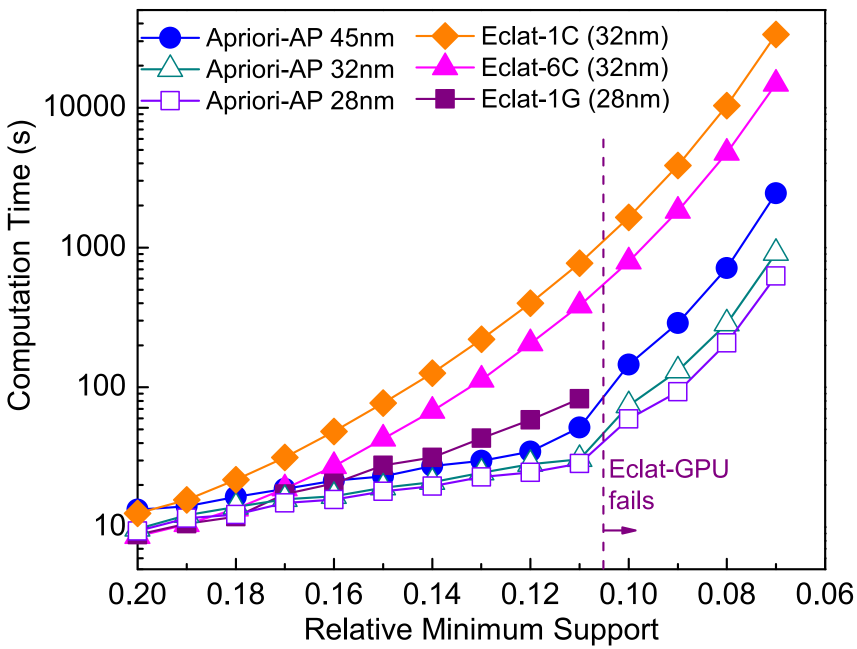
Automata Design for SPM: Flattened



(a) Automaton for matching sequence $< \{1, 50\}, \{15, 80\} >$

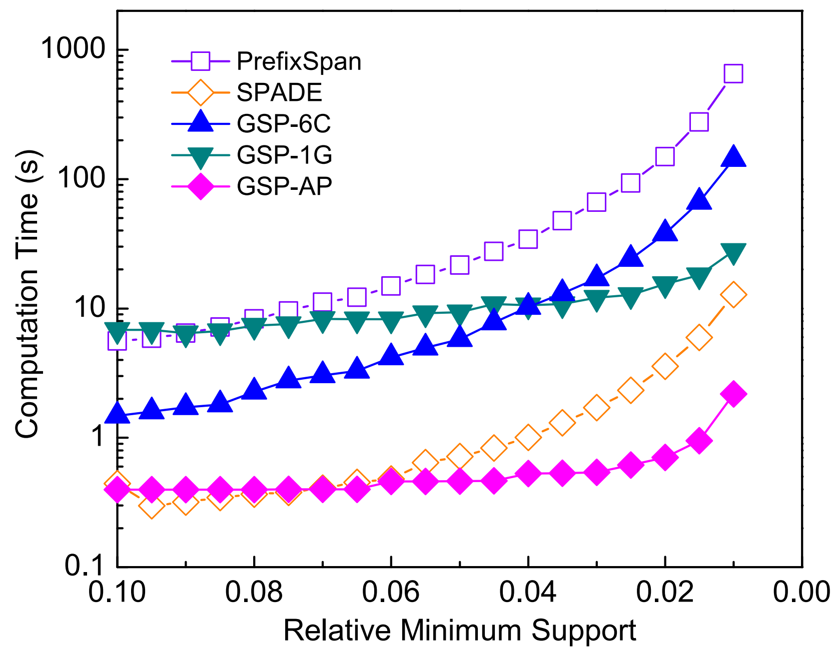
Performance Evaluation

FIS



Webdocs5X (7.1GB)

SPM



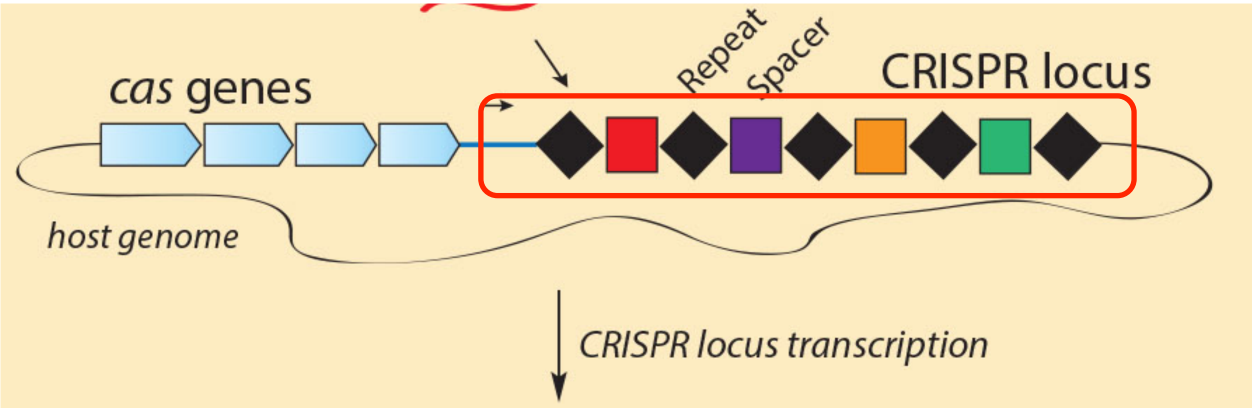
Bible

Performance summary

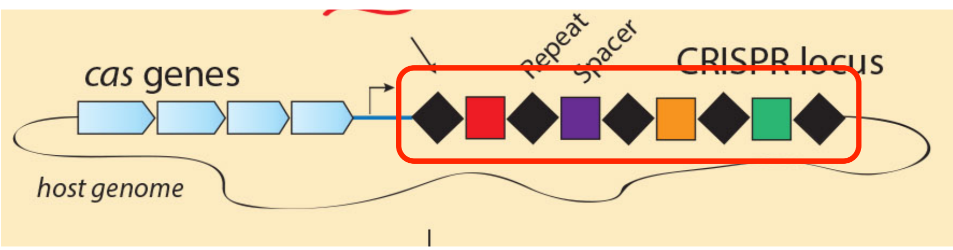
- **FIS:** Up to 129X speedup over single-core CPU implementation of Apriori and up to 49X speedup over multicore-based and GPU-based implementations of Eclat ARM
- **SPM:** Up to 430X, 90X, and 29X speedups are achieved by the AP-accelerated GSP, when compared with the single-threaded CPU, multicore CPU, and GPU GSP implementations, respectively

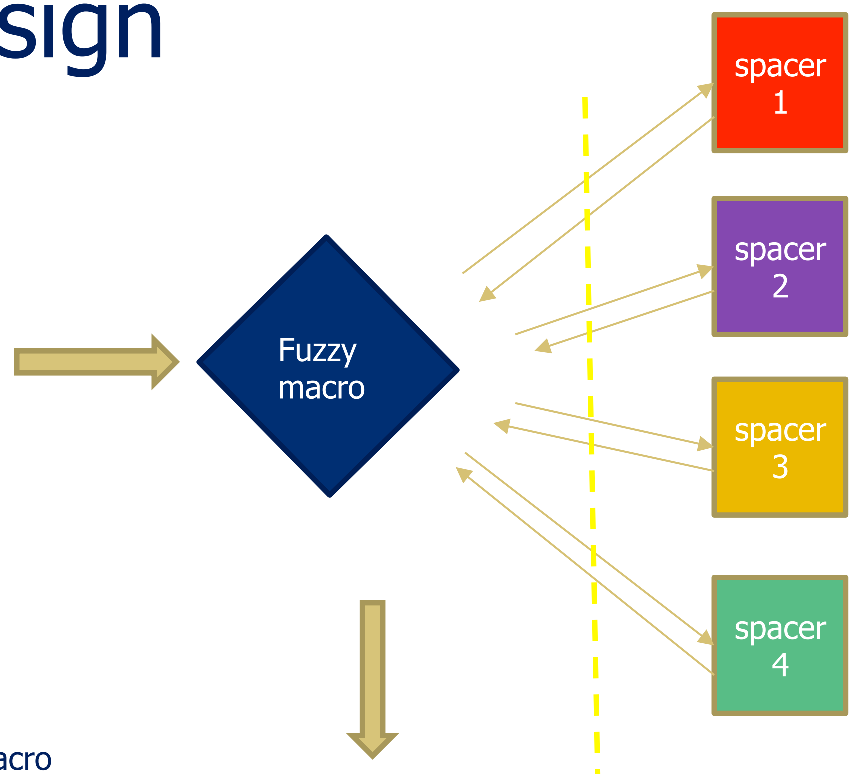
Bioinformatics: CRISPR Sites Discovery

- CRISPR: Clustered Regularly Interspaced Short Palindromic Repeats
- Each repeat is followed by a spacer DNA and the spacer could be either the same or different
- Mismatches/gaps may be allowed in repeats
- Potential applications: genome engineering, RNA editing, Biomedicine, etc.

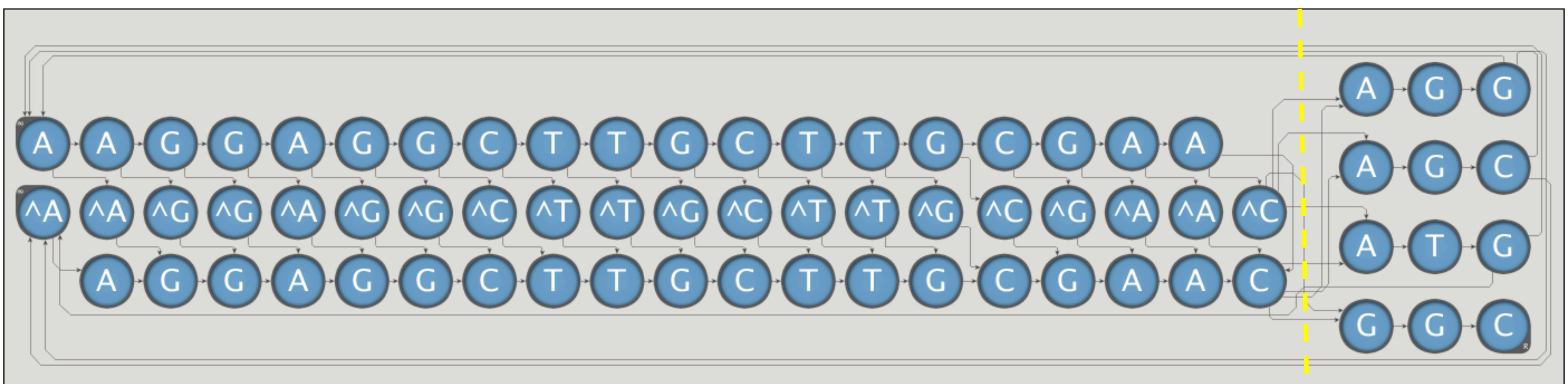


AP Design



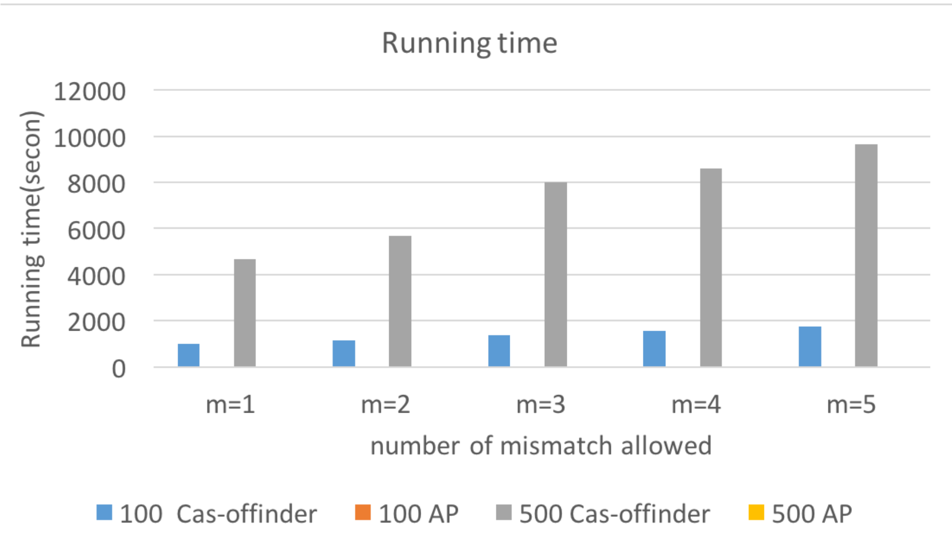


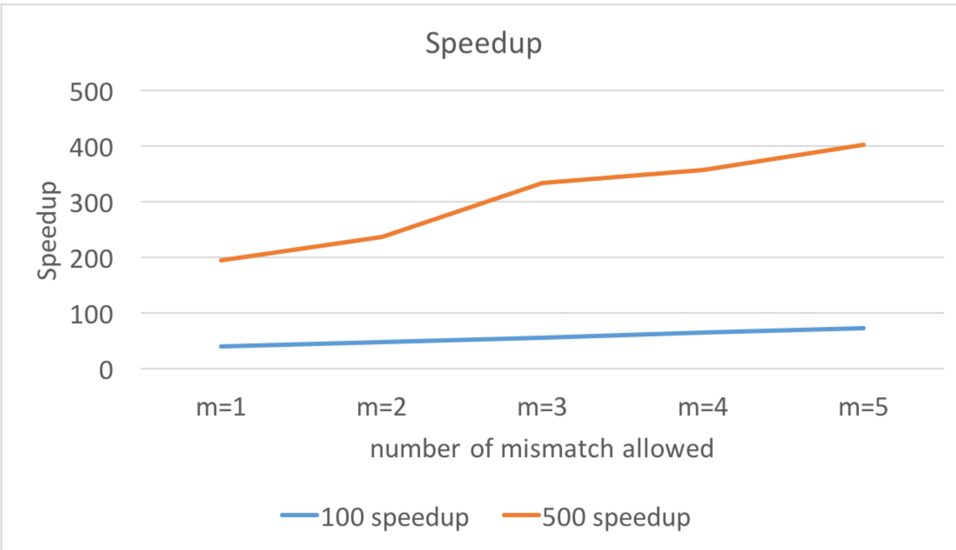
Mismatch fuzzy macro



Preliminary Results

- Find 100 and 500 CRISPRs
- Allow different number of mismatches (1~5)
- Promising speedup achieved, from 40.7x to 402x
- Speedup is better for larger database



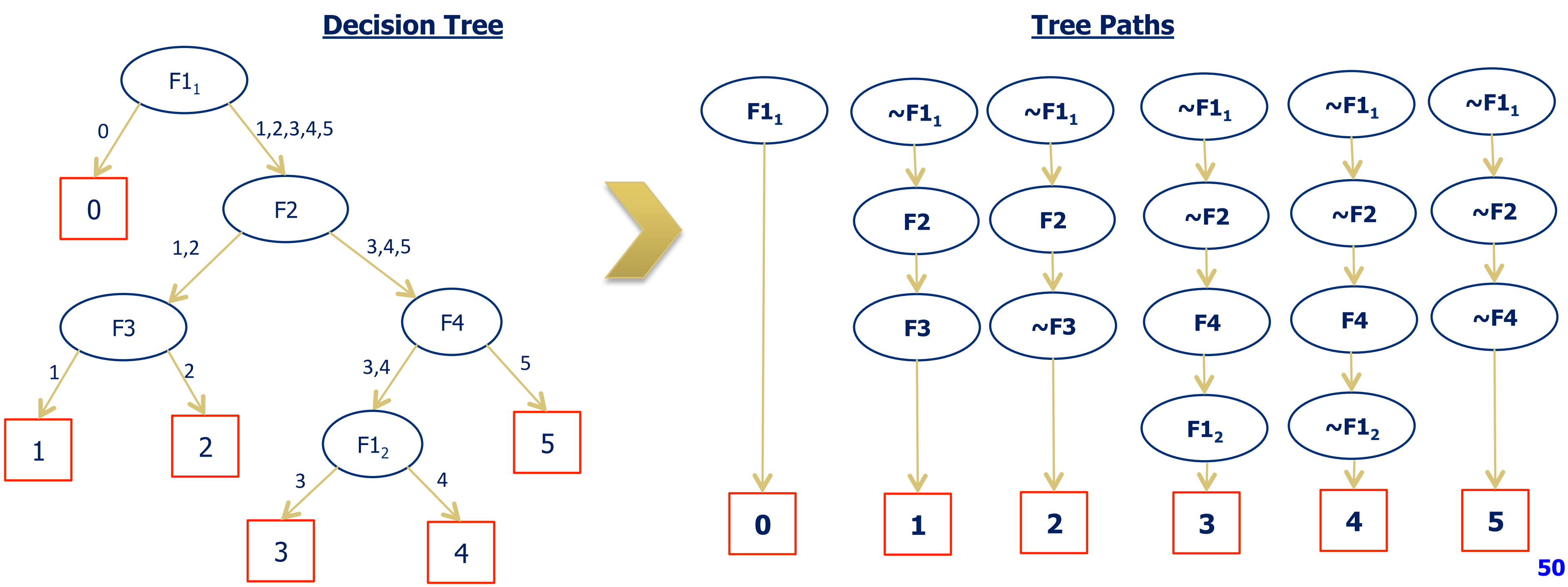


Random Forest on the AP ISC'16

- Ensemble learning method for classification, etc
- Construct a multitude of decision trees, test all
- Randomly restricted to be sensitive to only selected feature dimensions
- Reduces overfitting, better scalability
- Use AP for inference stage

Tree-Traversal to Pattern Matching?

- Restructure each Decision Tree into chains
  - Each chain represents a path through each tree in the forest.
  - Do this for ALL trees in the forest.



Experimental Results

- **Twitter**: The AP achieved a max 93x speedup over CPU
- **MNIST**: The AP achieved a max 63x speedup over CPU

Table 1: Key data points of Twitter Results

Trees	Leaves	Accuracy	AP Throughput (k Pred/Sec)	CPU Throughput (k Pred/Sec)	AP Speed Up
5	40	66.9%	14400	154	93
10	40	67.5%	8130	129	63
20	40	67.7%	5360	93.4	57
40	40	68.0%	3750	58.5	64
5	600	70.4%	2010	118	17
10	600	71.4%	1530	86.4	18
20	700	71.7%	385	51.5	7
40	700	71.9%	194	32.4	6

Table 2: Key data points of MNIST Results

Trees	Leaves	Accuracy	AP Throughput (k Pred/Sec)	CPU Throughput (k Pred/Sec)	AP Speed Up
5	50	82.2%	13200	337	39
10	50	86.1%	5980	242	25
20	50	87.8%	4170	150	28
40	50	88.7%	3350	86.5	39
80	50	89.2%	2940	46.4	63
160	50	89.6%	1350	25.0	54
10	500	93.3%	2480	205	12
20	500	94.3%	1160	125	9
40	750	95.2%	420	68.0	6
80	1250	96.0%	111	34.3	3
20	4000	96.1%	129	98.9	1.3
40	4750	96.7%	55.0	51.5	1.1
80	5000	96.9%	25.0	26.6	0.9
160	5000	97.1%	12.2	13.5	0.9

- AP exhibits tradeoff in capacity: larger trees/strings = fewer trees/strings per pass

Randomized Input
(ICCD'16)

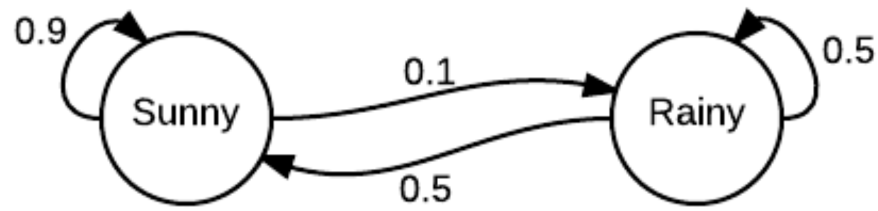
IDEA: *randomize the input symbol stream*

- Not using finite automata anymore
- What power does this give us?
  - AP allows conditional transitions based on input symbols
  - With randomized input, transition conditions are random!
    - Each character class now has a ***probability*** of being recognized based on the ***distribution*** of random input symbols
  - This means we can naturally build probabilistic automata (PA) on the AP
    - Generalization of a ***Markov Chain***

Markov Chain Examples

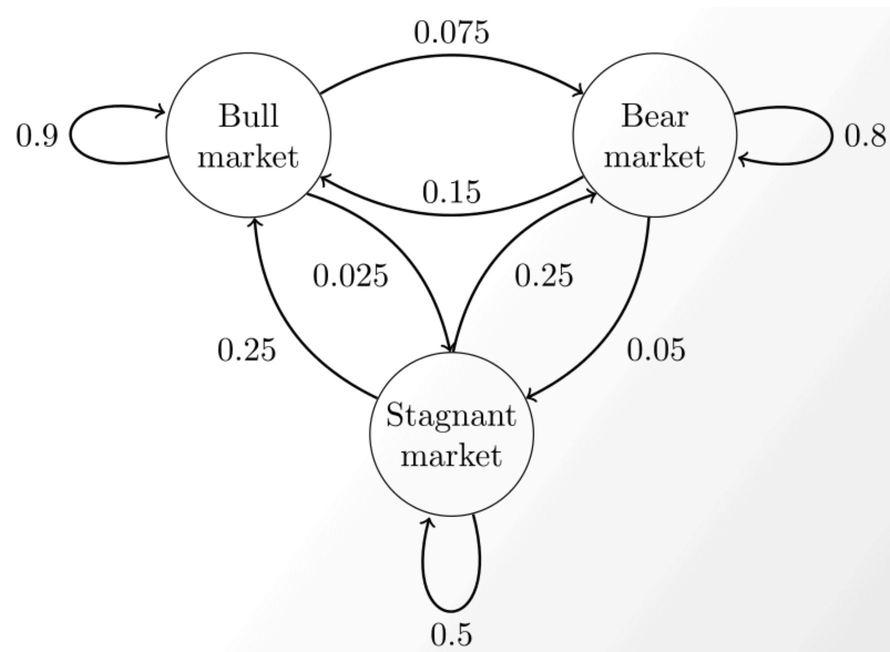
Stochastic Transition Matrix (rows sum to 1)

	Sunny	Rainy
Sunny	0.9	0.1
Rainy	0.5	0.5



BULL BEAR STAG

$$P = \begin{bmatrix} 0.9 & 0.075 & 0.025 \\ 0.15 & 0.8 & 0.05 \\ 0.25 & 0.25 & 0.5 \end{bmatrix}$$



Markov Chain Example | "Fair Coin"

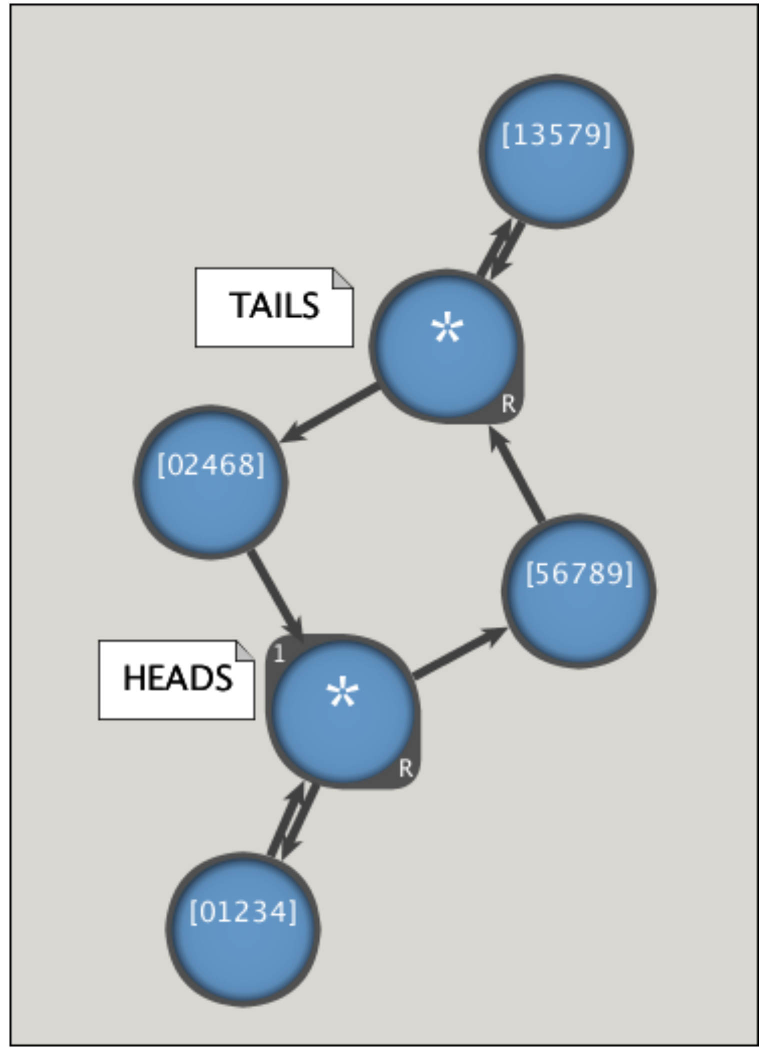
Stochastic transition matrix

	Heads	Tails
Heads	0.5	0.5
Tails	0.5	0.5

Stochastic symbol "buckets"

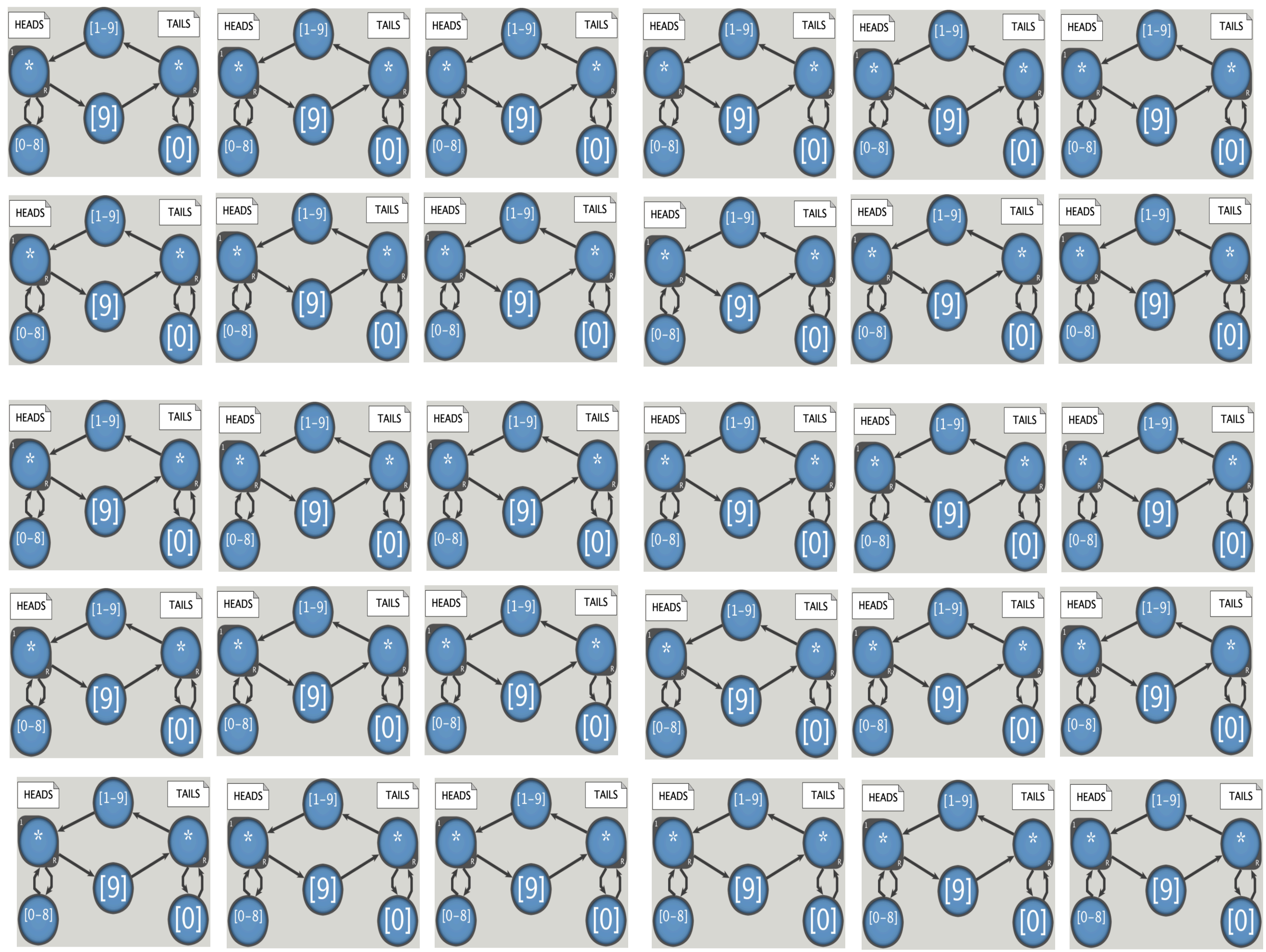
	Heads	Tails
Heads	[01234]	[56789]
Tails	[02468]	[13579]

For this example, we assume randomized input symbol [0-9]



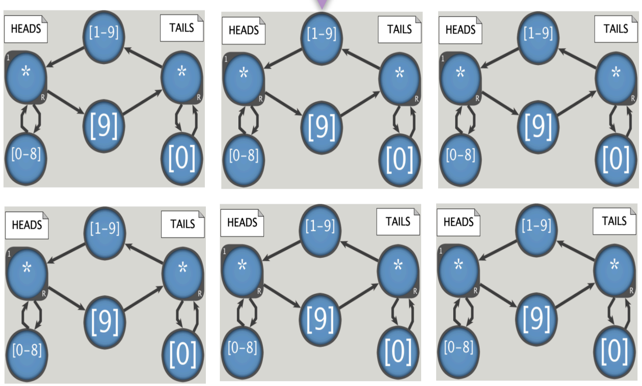
Hypothesis: Many parallel chains can create a massive amount of parallel probabilistic behavior

One 8-bit symbol stream



Applications? PRNG or Agent-based Simulation

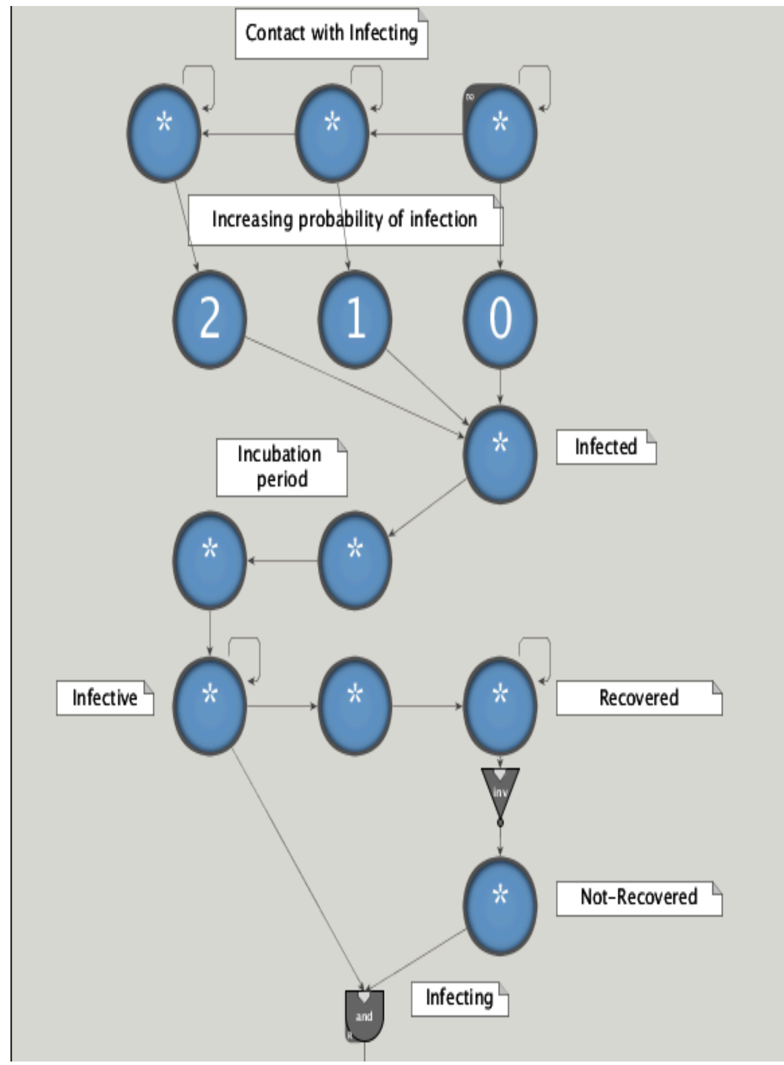
PRNG input stimulus



PRNG output

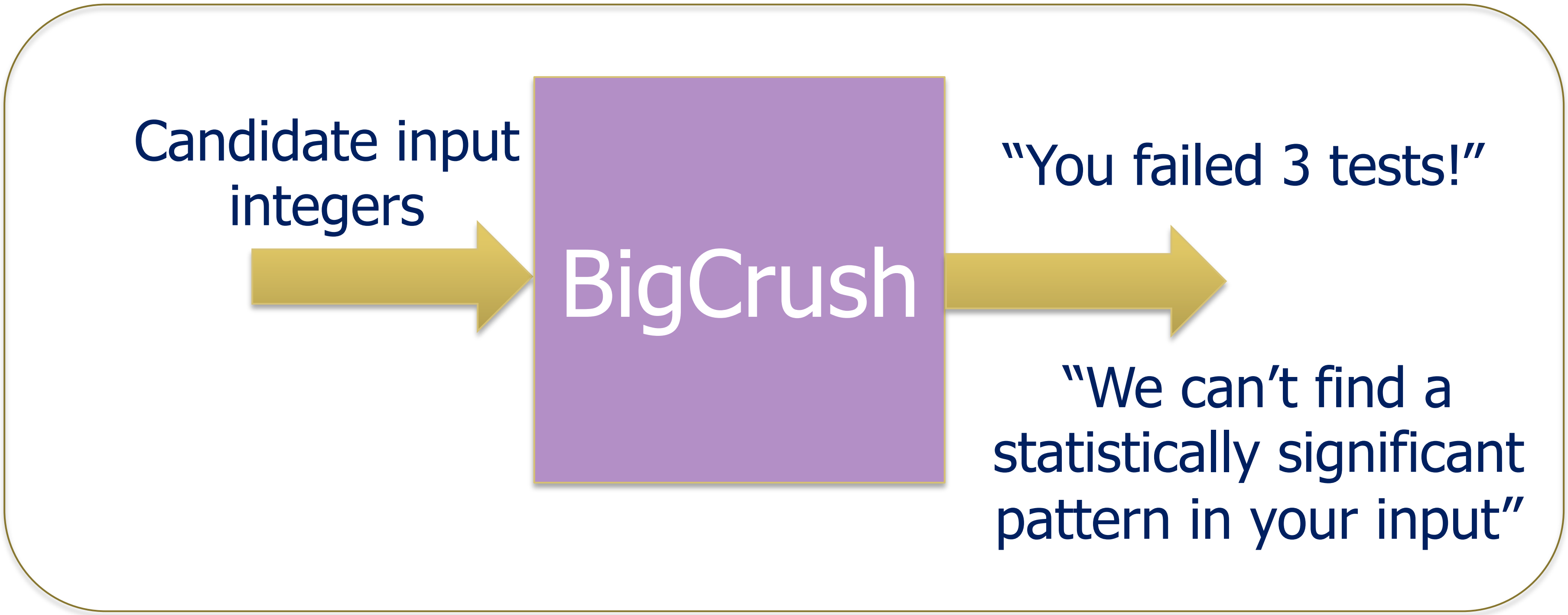
PRNG Consumer (e.g., Monte Carlo Simulation)

SIR Epidemiological Model



Statistical tests are used to measure quality of random output

- TestU01 Statistical Test Suite



If you pass BigCrush, you are indistinguishable from random

Results

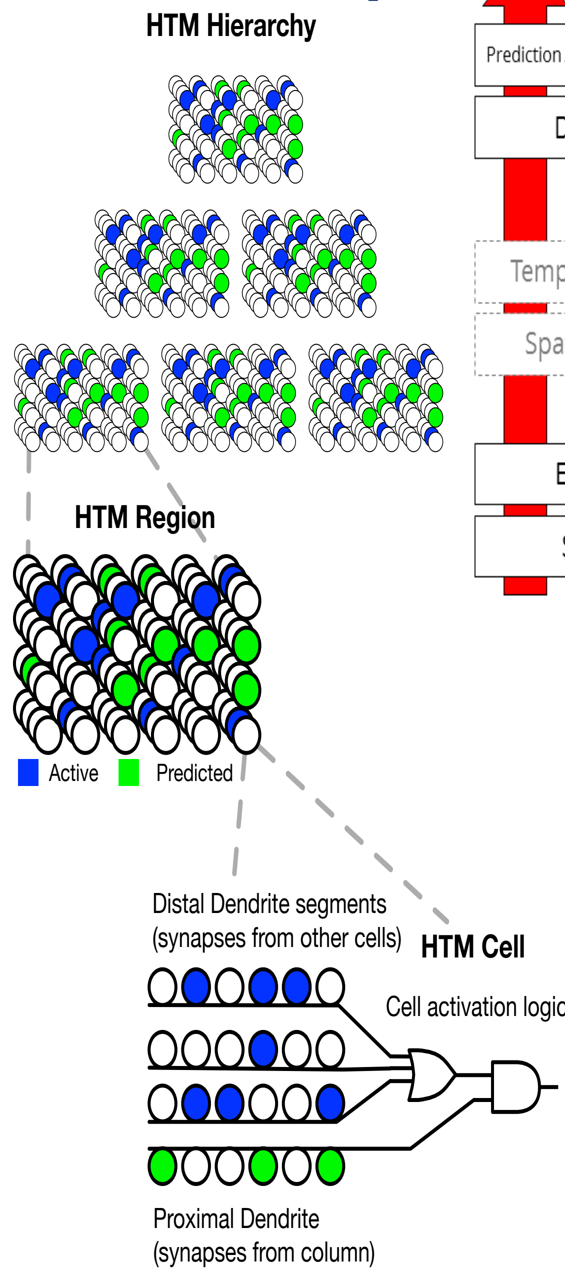
- 8-state chains sufficient
- Chain transitions need to be randomly generated
  - Need to reconfigure periodically
- Very high throughput possible

Memory Technology	**DDR3**	**DDR4**	**HMC 2.0**
Peak Throughput (GB/s)	12.8	17.0	320
T_r (μs)	91.4	68.8	7.3
AP Chip Output (GB/s)	28.2	28.3	28.5
Throughput Limited Output (GB/s)	12.8	17.0	28.5

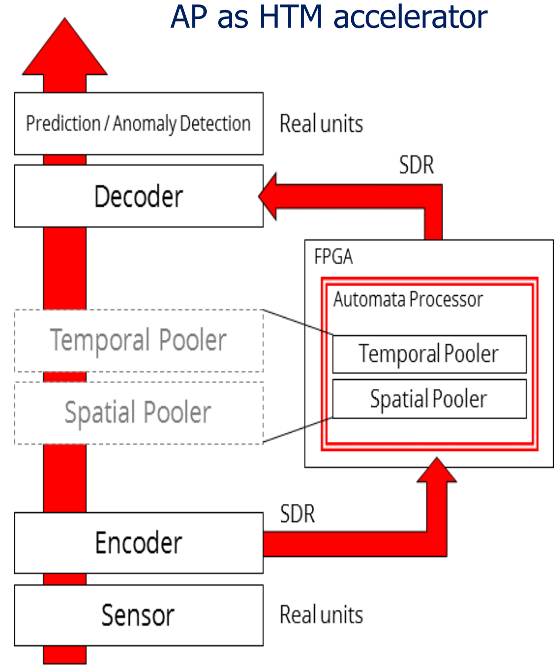
- Predict 6.8X better energy efficiency than GPU

Hierarchical Temporal Memory

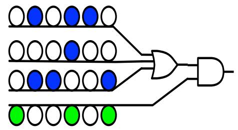
- Recurrent Neural Network (RNN) based on binary synapses
- Performs, learning, inference, and prediction on a continuous stream of inputs
- Has been used for prediction, anomaly detection, classification tasks
- ***Key idea***: Use AP as an accelerator for HTM



AP as HTM accelerator

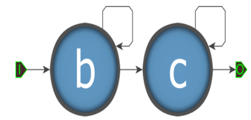


HTM-AP Correspondences



HTM

- Lateral connections make cell eligible to activate
- External inputs activate cell, propagating activation
- Predictions are determined based on past activations



AP

- Lateral connections from matching STEs make STE eligible to match
- STE matches if symbol on global input matches stored
- Next-state activations are computed based on current state and input symbol

Key idea: Exploit many natural correspondences to gain parallelism with AP

Benchmark Simulation Results

Benchmark	Base error (%)	AP error (%)	Base runtime (s)	AP runtime (s)	Speedup
Sine	13.6	14	2.05	4.59e-3	446
Hotgym	27	26.4	0.736	4.62e-3	159
NYCTaxi	11.6	8.8	8.76	63.9.e-3	137
	Columns	**Cells**	**STEs**	**Counters**	**Booleans**
Sine	1,170	18,395	1,478,742	30,367	2,340
Hotgym	329	6,320	593,670	11,609	658
NYCTaxi	1,804	44,161	8,540,630	160,997	3,608

Key result: HTM model in AP offers 137-446X speedup while preserving accuracy

ANMLZoo is a collection of 14 diverse automata benchmarks and standard inputs that can be used to evaluate automata processing engines and architectures (IISWC'16)

Regular Expression Rulesets:

- Snort
- ClamAV
- Dotstar (Becchi et al.[1])
- PowerEN[2]
- Protomata
- Brill Tagging

Mesh Automata:

- *Hamming
- *Levenshtein

Synthetic:

- *Block Rings
- *Core Rings

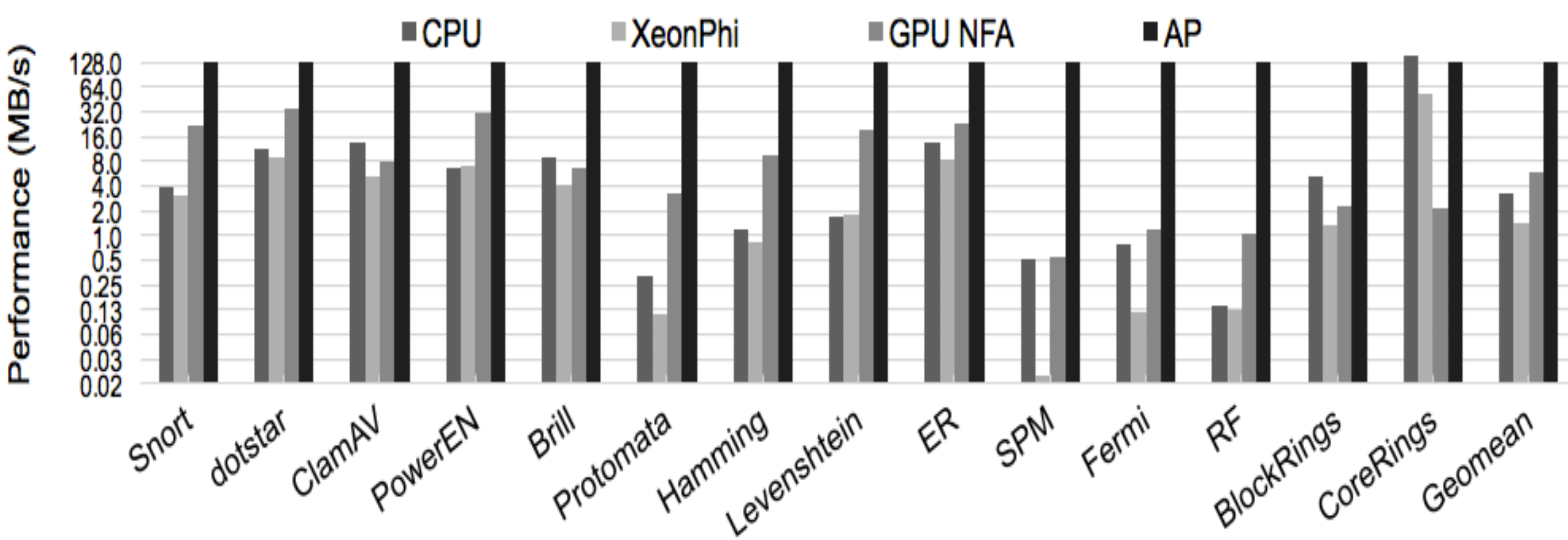
"Widgets":

- Sequential Pattern Mining[3]
- Fermilab Particle Tracking
- Entity Resolution
- Random ForestS

**Parametric code generation tools are included*

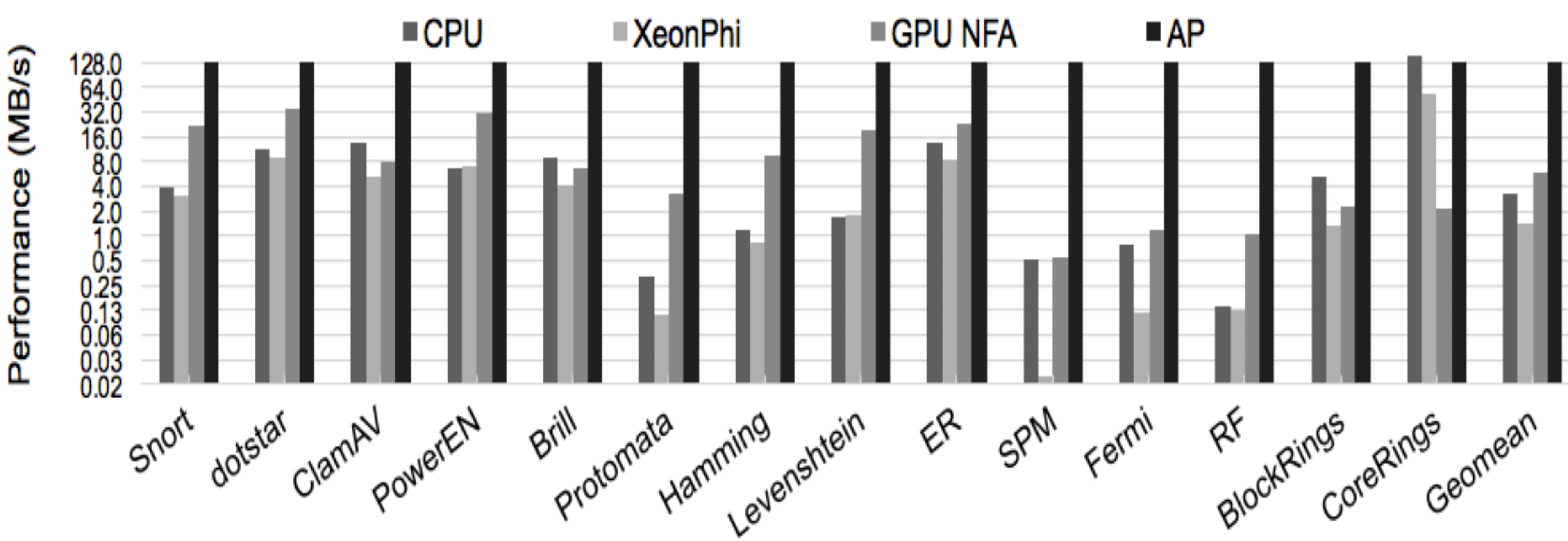
ANMLZoo Cross-Architecture Evaluation

- XeonPhi performance is lower than CPU performance because of reduced frequency and per-thread cache
- GPUs can outperform CPUs because of their superior latency hiding, *not* because of SIMD computation
- Reconfigurable fabrics can perform much better than von Neumann architectures if the automata can be placed-and-routed into the reconfigurable fabric



ANMLZoo Cross-Architecture Evaluation, cont.

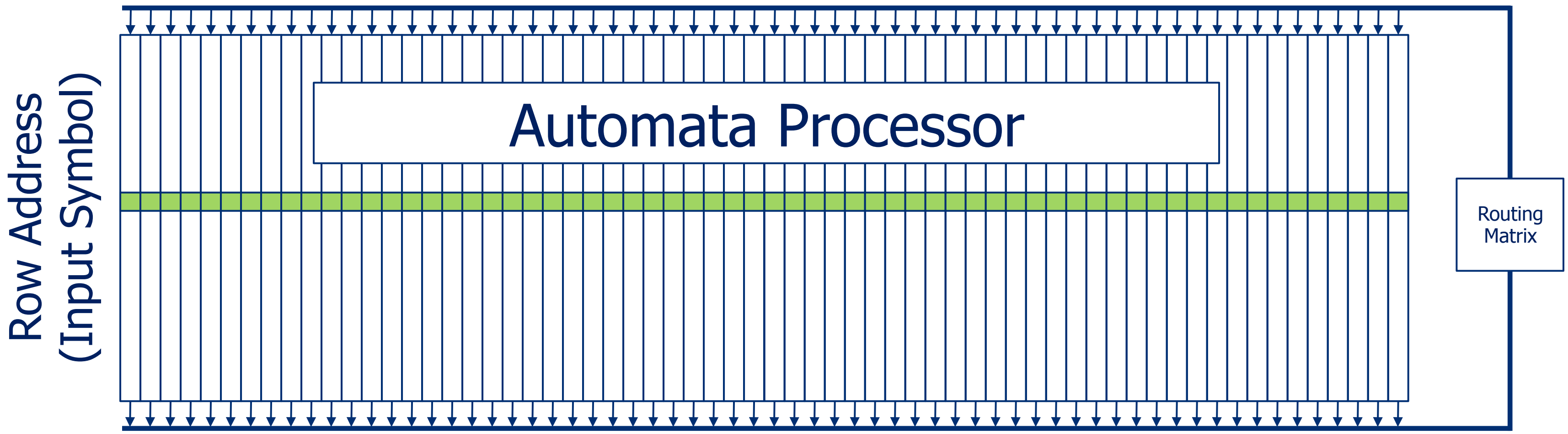
- Note substantial speedup even for "conventional" regex rulesets
- But much higher speedup for applications with more complex automata structures
  - Esp. high activity factors
- Very promising early results for FPGAs as well
  - AP still better, but benefits of spatial architecture are clear



Automata Processing and Spatial Architectures are very powerful for many applications – not just regex!

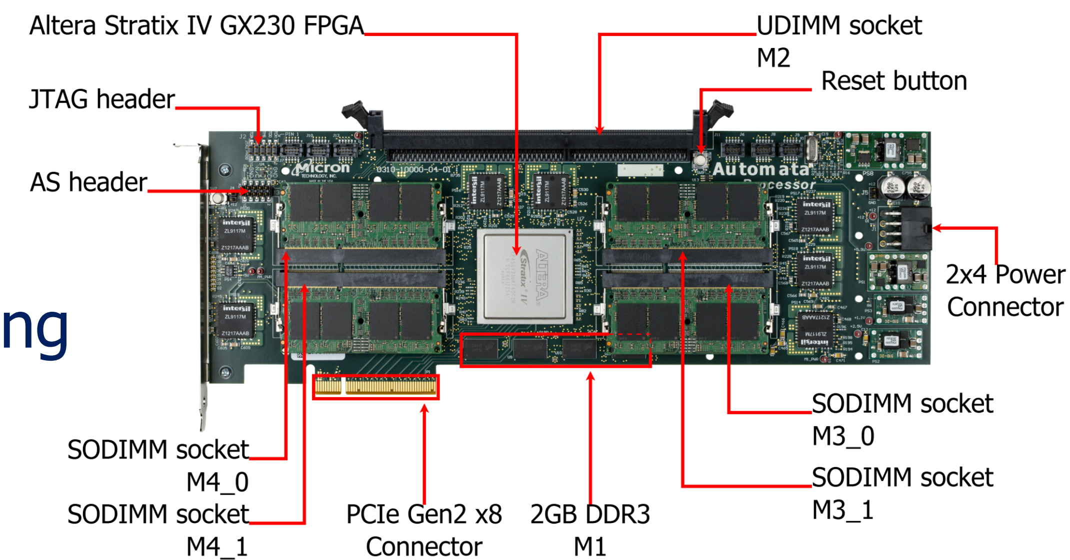
- Brill tagging
- Entity resolution
- Association rule mining
- Bioinformatics – CRISPR
- Random Forest
- Markov processes
- Hierarchical temporal memory
- Automata benchmarking

AP Architecture



Row Access results in **<u>49,152</u>** match & route operations/chip
(then Boolean AND with "active" bit-vector)

- Implements NFAs natively in hardware
- Non-determinism very powerful for fuzzy matching
- Massive parallelism



Figures courtesy of Micron

Many Exciting Research Questions

- Leveraging on-board FPGA
- Line-speed processing
- Cluster, datacenter-scale processing
- Processing pipelines
  - Including spanning multiple heterogeneous processing units
- New form factors
  - Make AP fully autonomous: CPU, memory, etc.
  - 3D stacking
  - New interfaces directly to high-bandwidth data streams
- New architectures, more flexible than just automata
  - E.g., numerical range checking
  - Extensions for graph processing, more neural models
- New algorithms, libraries, etc.
- And many more…

Questions?

THE CENTER FOR AUTOMATA PROCESSING

www.cap.virginia.edu

www.micronautomata.com

skadron@virginia.edu