UNIVERSITY of VIRGINIA

COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF MICHIGAN

ASPEN: A Scalable In-SRAM Architecture for Pushdown Automata

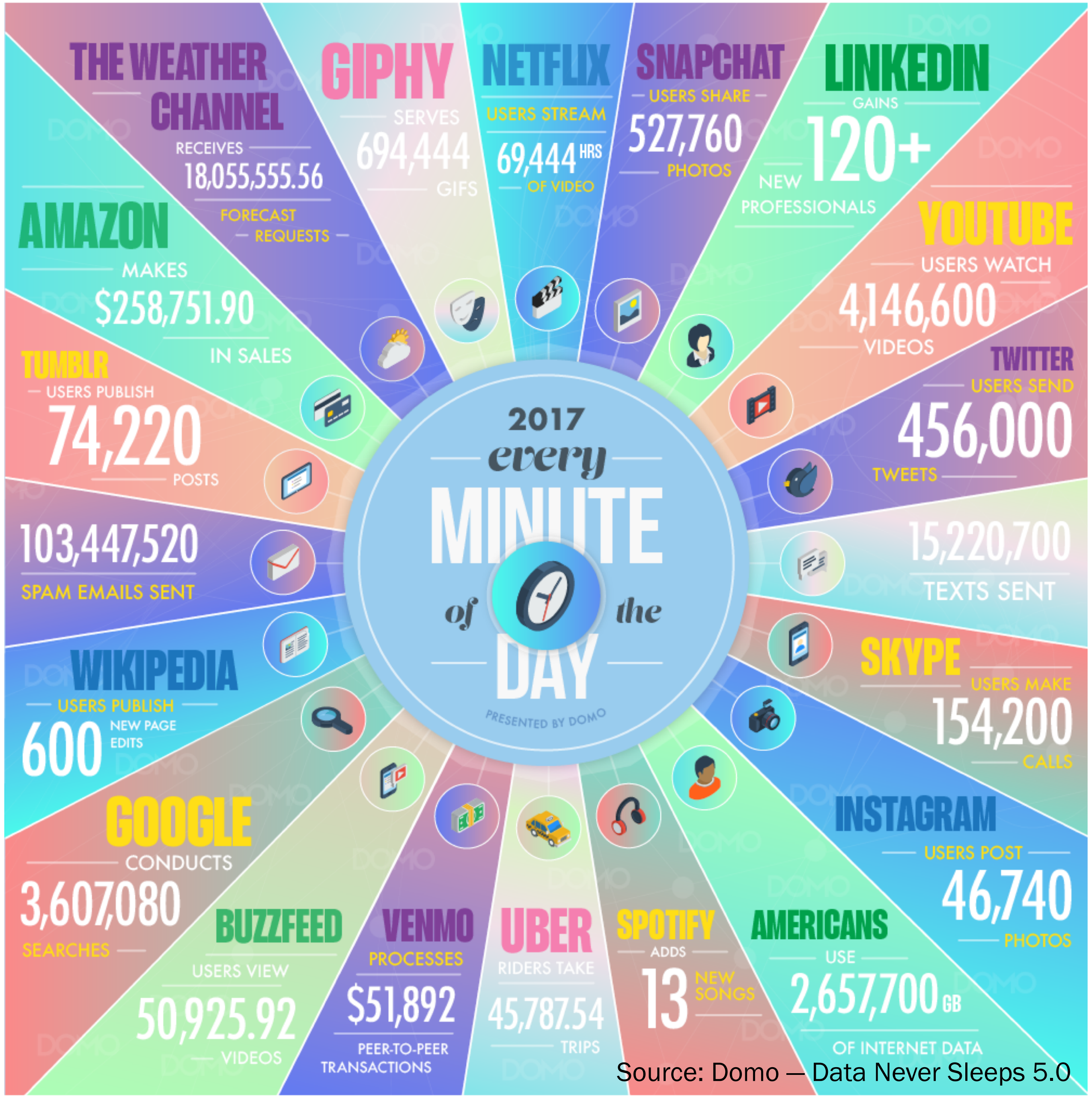
Kevin Angstadt[*], Arun Subramaniyan[*], Elaheh Sadredini[†], Reza Rahimi[†], Kevin Skadron[†], Westley Weimer[*], Reetuparna Das[*]

[*]Computer Science and Engineering, University of Michigan
[†]Department of Computer Science, University of Virginia

This work is funded, in part, by the NSF (1763674, 1619098, CAREER-1652294 and CCF-1629450); Air Force (FA8750-17-2-0079); and CRISP, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

Processing Growing Quantities of Data

- 2.5 quintillion bytes of data/day
- Analysis /manipulation requires deserialization
- Most data use recursively-nested grammars
  - XML
  - JSON
- Poor performance on CPU
  - High branching



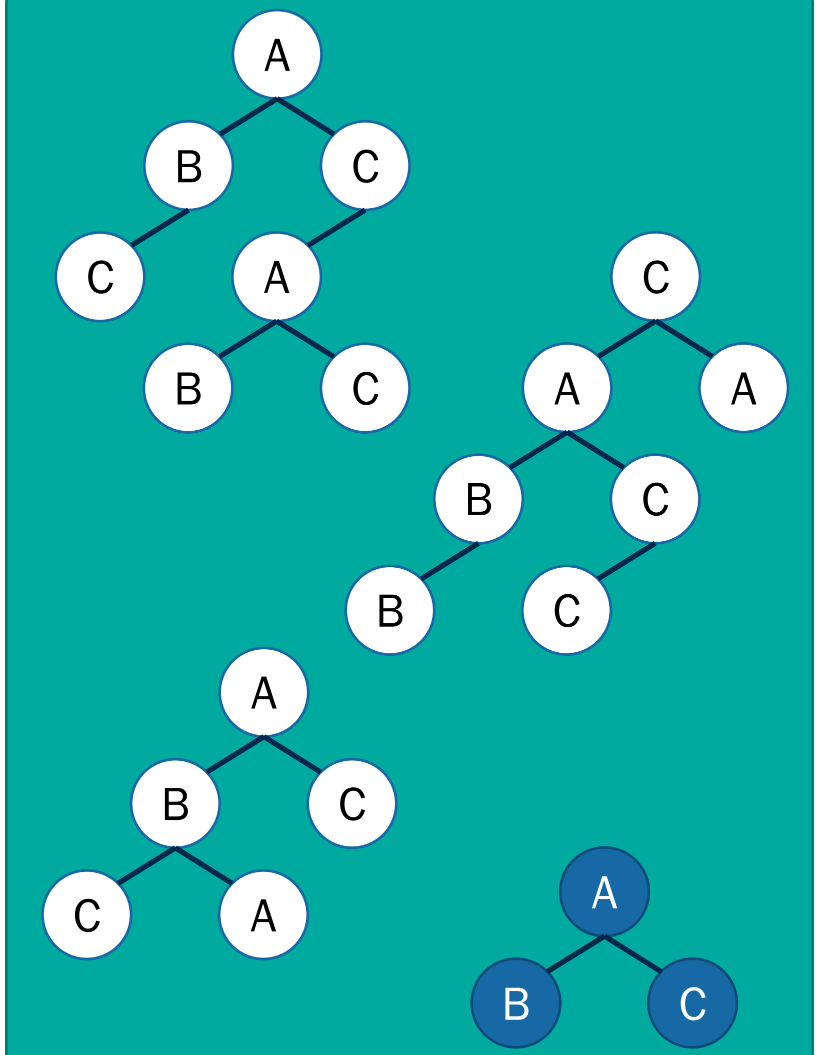
Source: Domo — Data Never Sleeps 5.0

```
<course>
  <footnote></footnote>
  <sln>10637</sln>
  <prefix>ACCTG</prefix>
  <crs>230</crs>
  <lab></lab>
  <sect>01</sect>
  <title>INT FIN ACCT</title>
  <credit>3.0</credit>
  <days>TU,TH</days>
  <times>
     <start>7:45</start>
     <end>9</end>
  </times>
  <place>
      <bldg>TODD</bldg>
      <room>230</room>
  </place>
  <instructor>
    B. MCELDOWNEY
  </instructor>
  <limit>0112</limit>
  <enrolled>0108</enrolled>
</course>
```

XML Nesting

```
<course>
  <footnote></footnote>
  <sln>10637</sln>
  <prefix>ACCTG</prefix>
  <crs>230</crs>
  <lab></lab>
  <sect>01</sect>
  <title>INT FIN ACCT</title>
  <credit>3.0</credit>
  <days>TU,TH</days>
  <times>
     <start>7:45</start>
     <end>9</end>
  </times>
  <place>
      <bldg>TODD</bldg>
      <room>230</room>
  </place>
  <instructor>
    B. MCELDOWNEY
  </instructor>
  <limit>0112</limit>
  <enrolled>0108</enrolled>
</course>
```

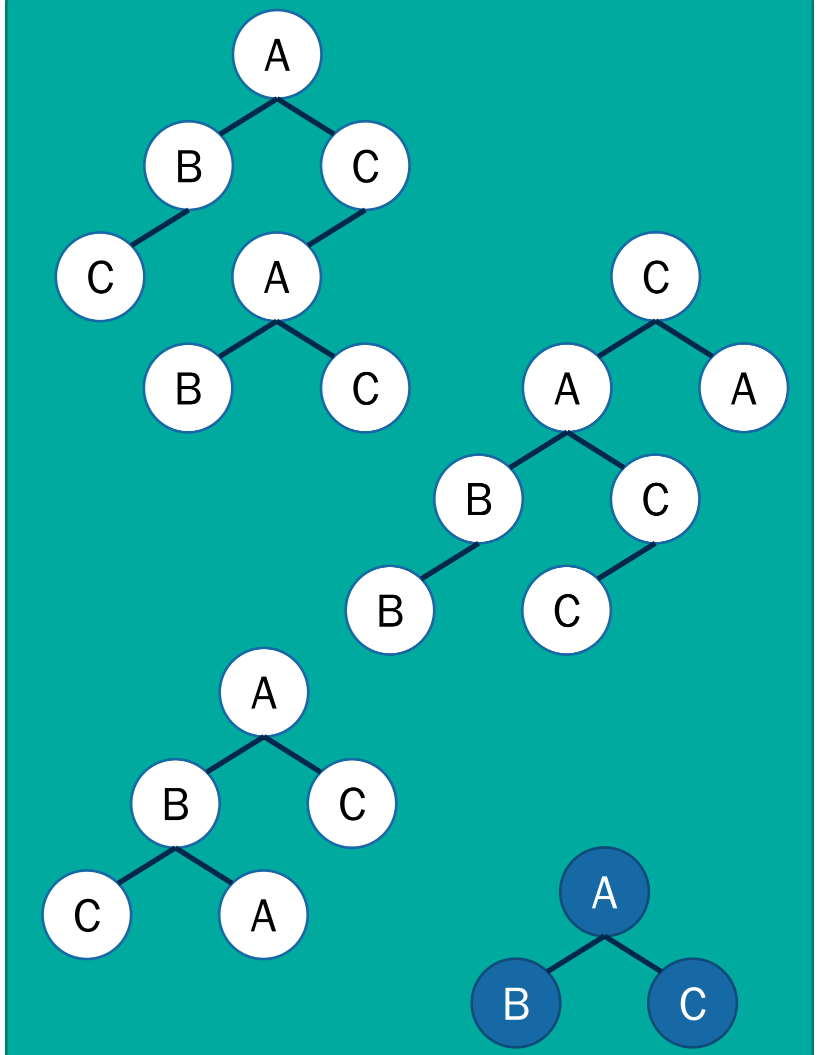
XML Nesting



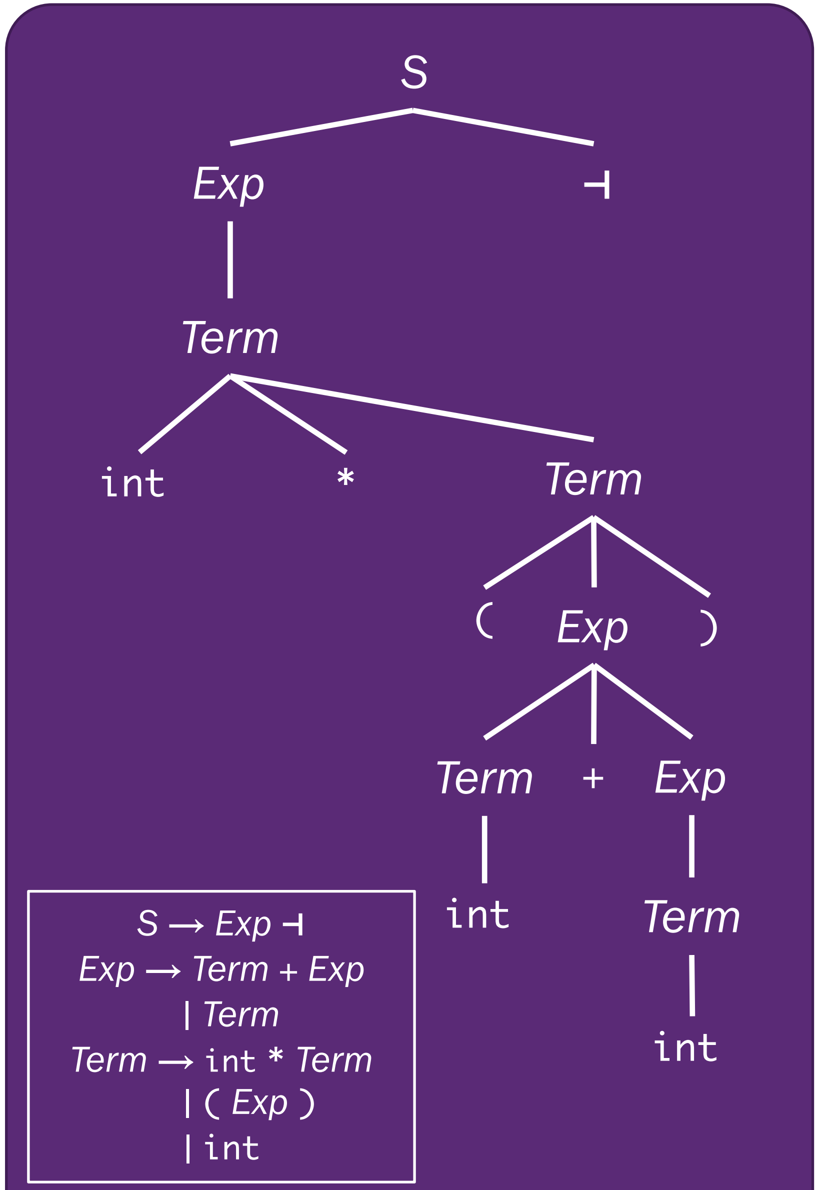
Subtree Mining

```
<course>
  <footnote></footnote>
  <sln>10637</sln>
  <prefix>ACCTG</prefix>
  <crs>230</crs>
  <lab></lab>
  <sect>01</sect>
  <title>INT FIN ACCT</title>
  <credit>3.0</credit>
  <days>TU,TH</days>
  <times>
     <start>7:45</start>
     <end>9</end>
  </times>
  <place>
      <bldg>TODD</bldg>
      <room>230</room>
  </place>
  <instructor>
    B. MCELDOWNEY
  </instructor>
  <limit>0112</limit>
  <enrolled>0108</enrolled>
</course>
```

XML Nesting



Subtree Mining

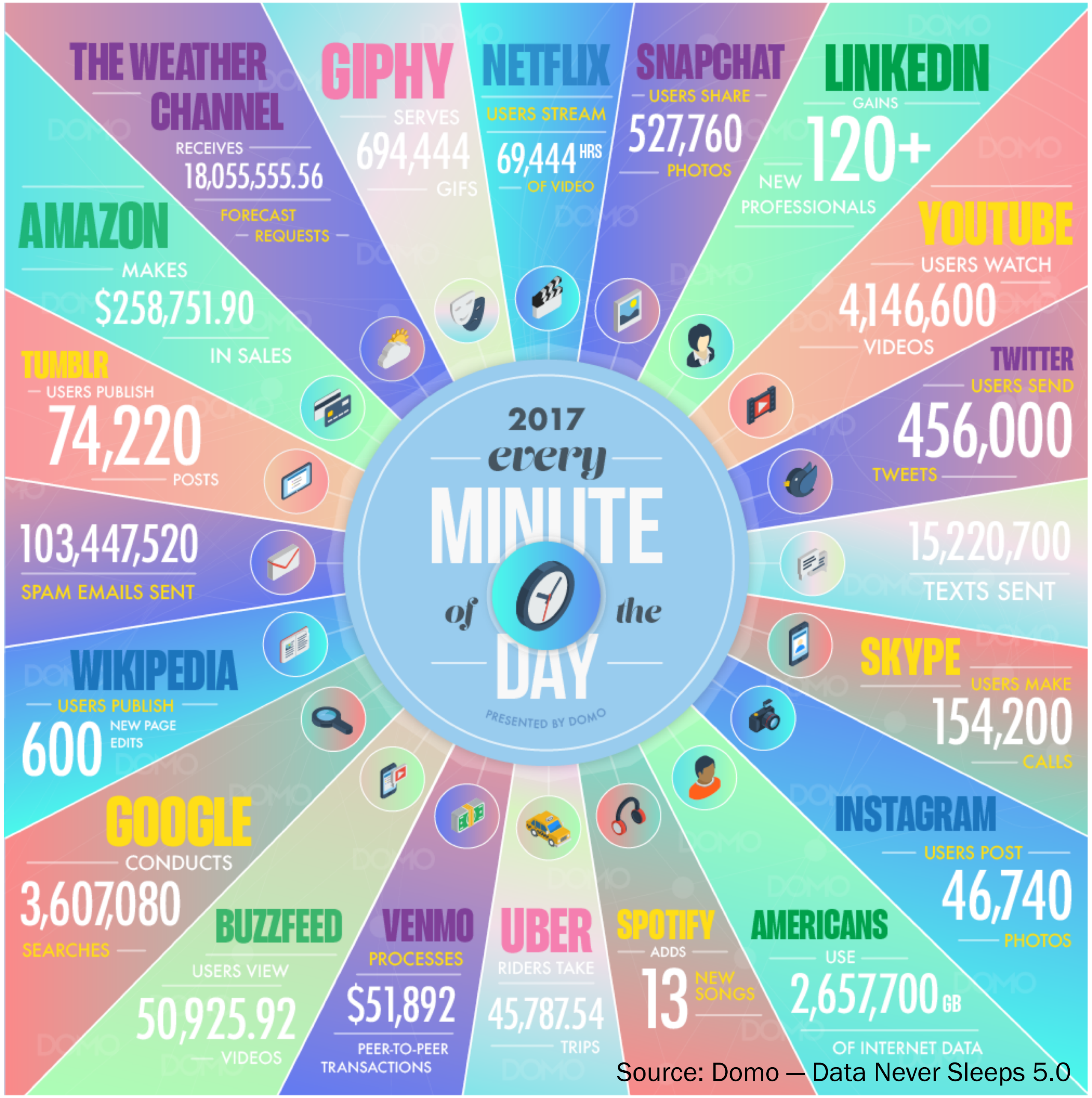


S → *Exp* ⊣
Exp → *Term* + *Exp*
| *Term*
Term → int * *Term*
| (*Exp*)
| int

Parsing

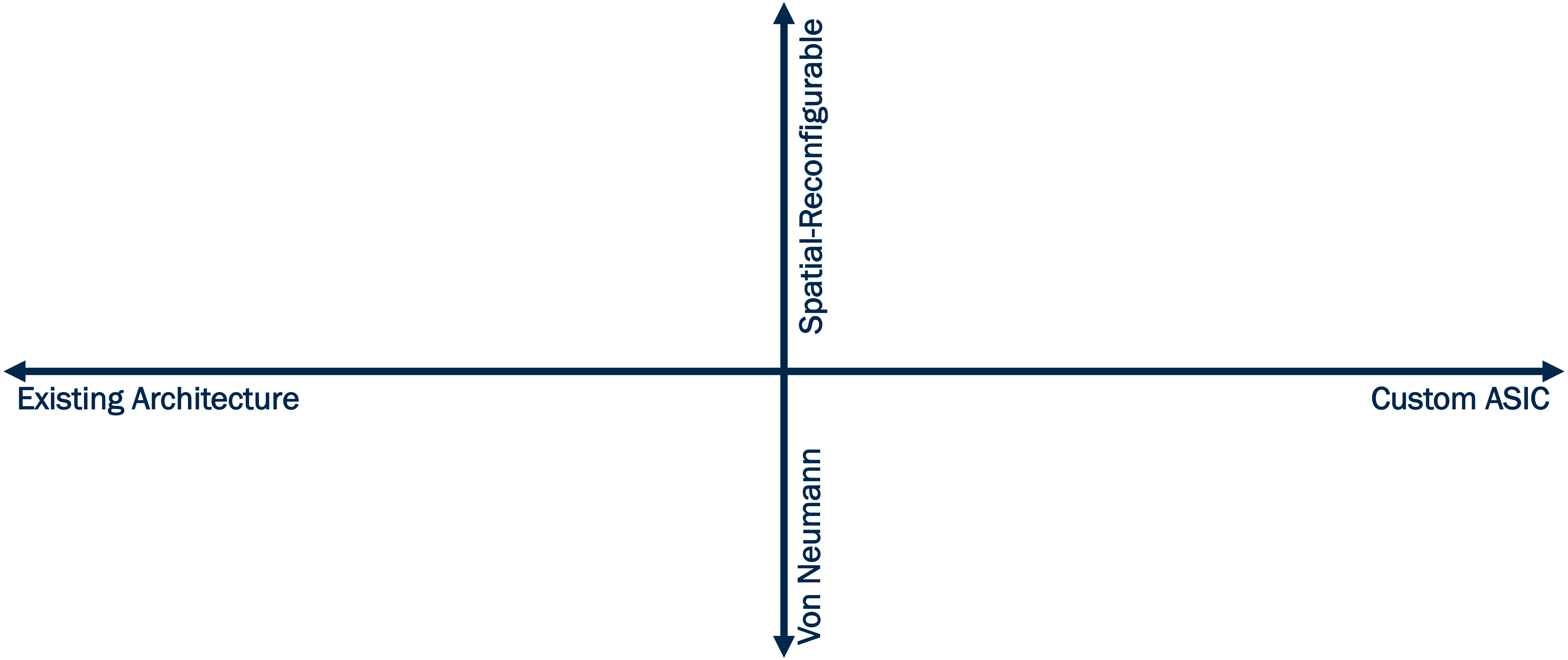
Processing Growing Quantities of Data

- Automata/RegEx help tame analysis of big data sets
  - Frequent Itemset/Pattern Mining
  - NLP Part-of-Speech Tagging
  - Data Deduplication
  - Ensemble-Based Classification
  - Particle Physics Analyses
- Growing number of architectural solutions

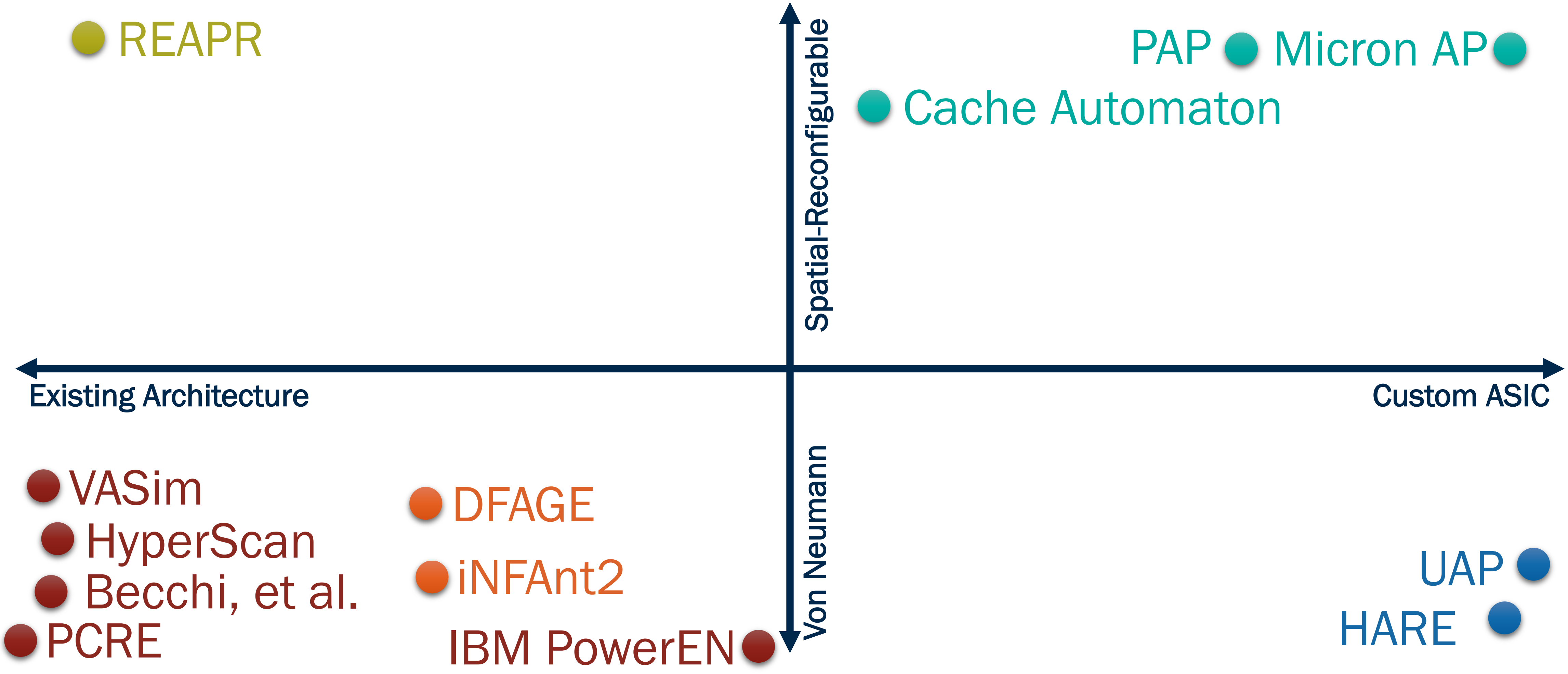


Source: Domo — Data Never Sleeps 5.0

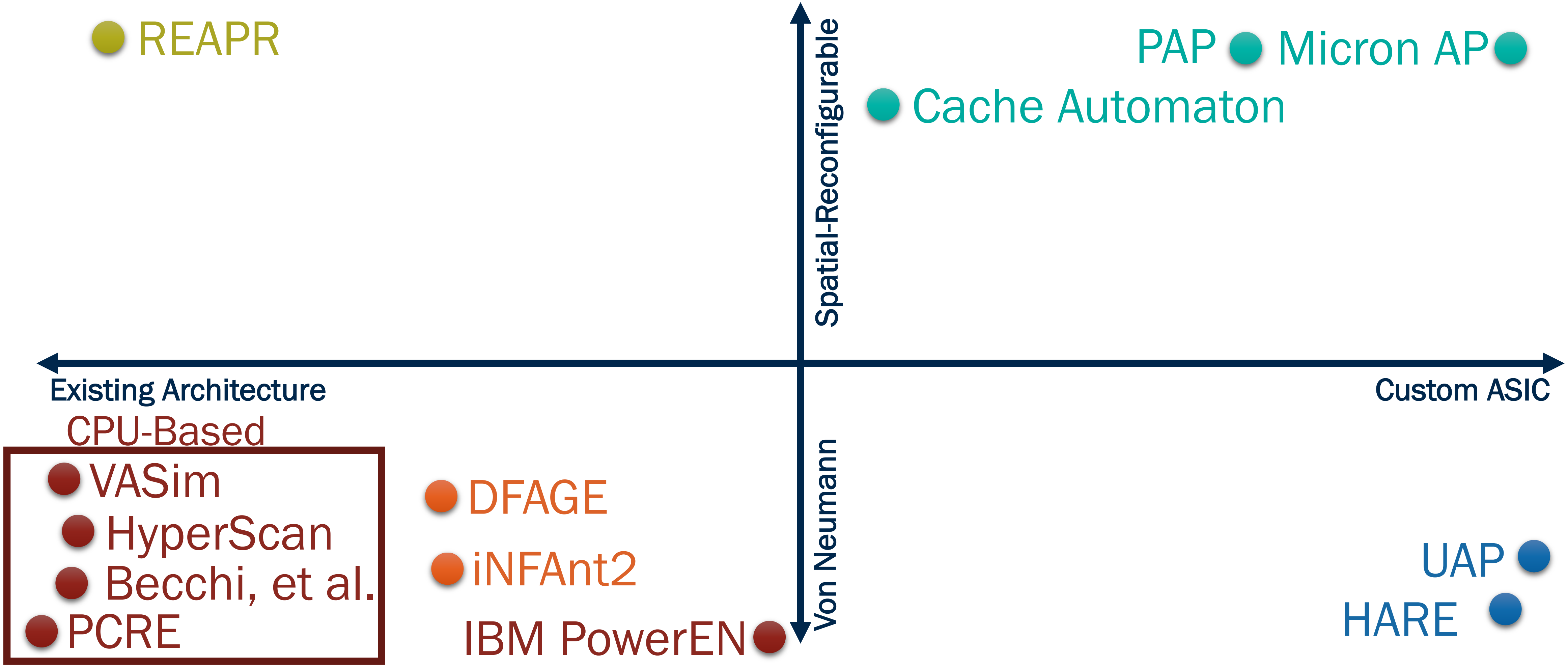
Automata/RegEx Processing Platforms



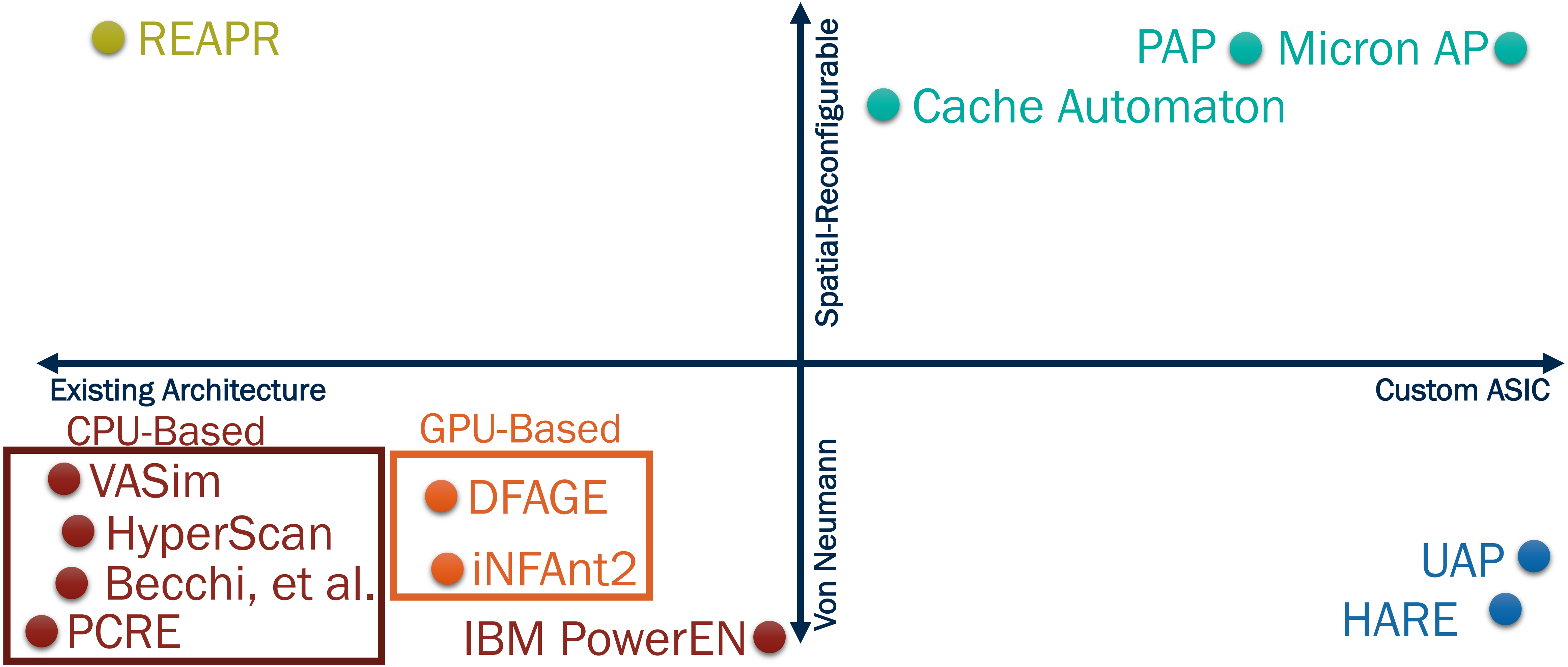
Automata/RegEx Processing Platforms



Automata/RegEx Processing Platforms



Automata/RegEx Processing Platforms



REAPR

Spatial-Reconfigurable

PAP

Micron AP

Cache Automaton

Existing Architecture

Custom ASIC

CPU-Based

VASim

HyperScan

Becchi, et al.

PCRE

GPU-Based

DFAGE

iNFAnt2

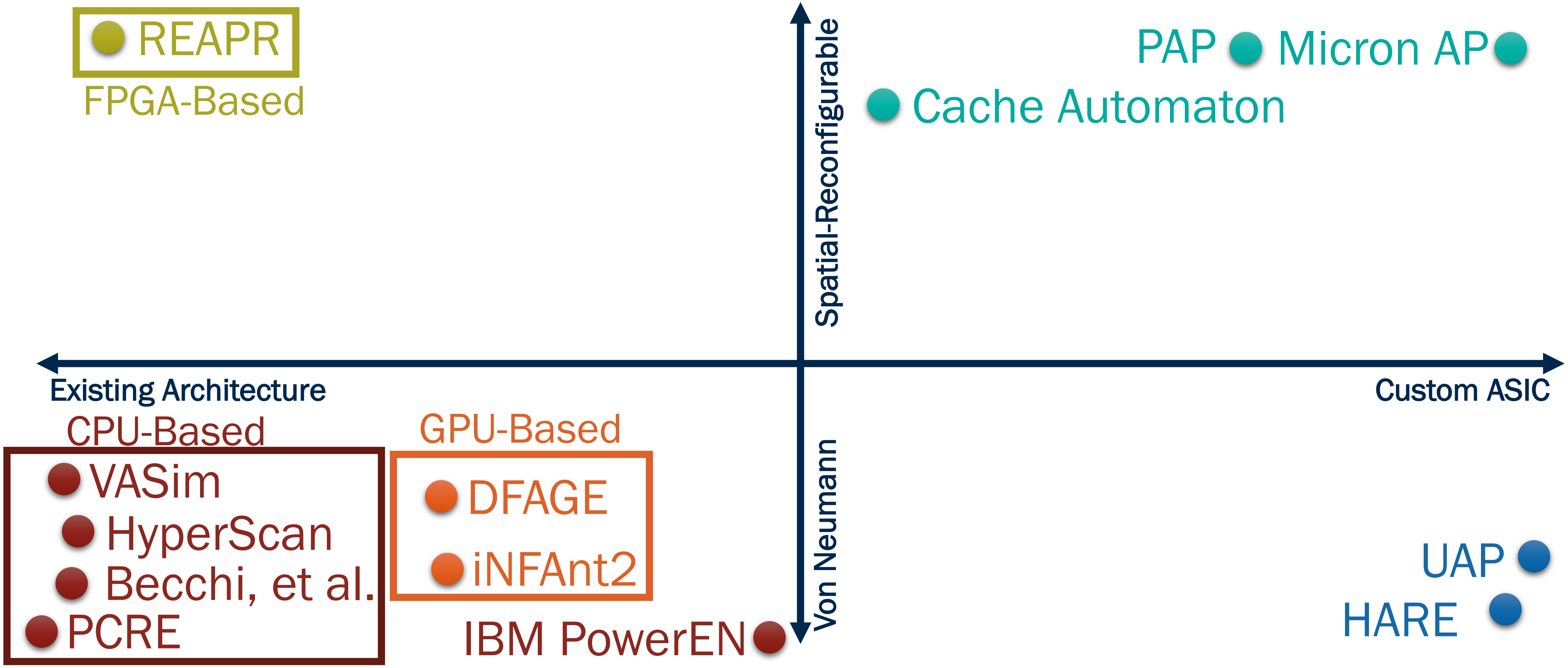
IBM PowerEN

Von Neumann

UAP

HARE

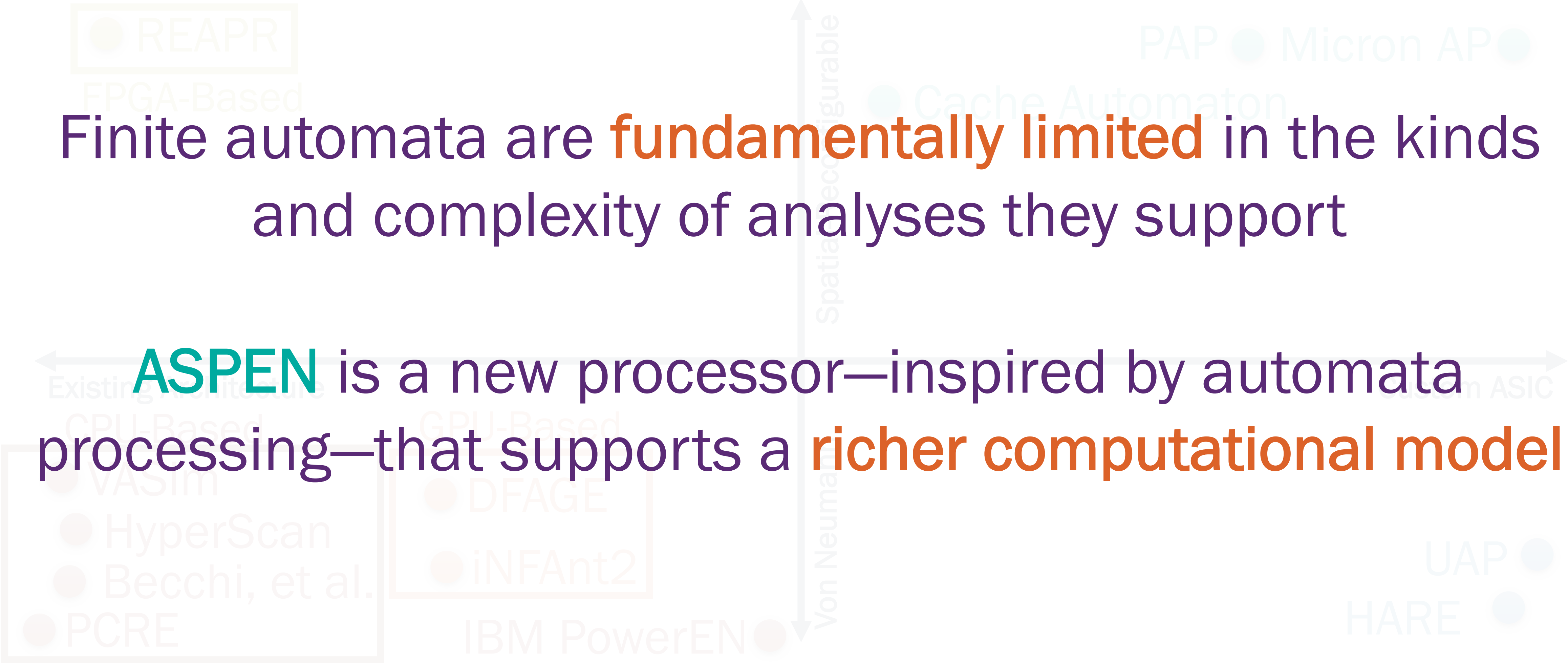
Automata/RegEx Processing Platforms



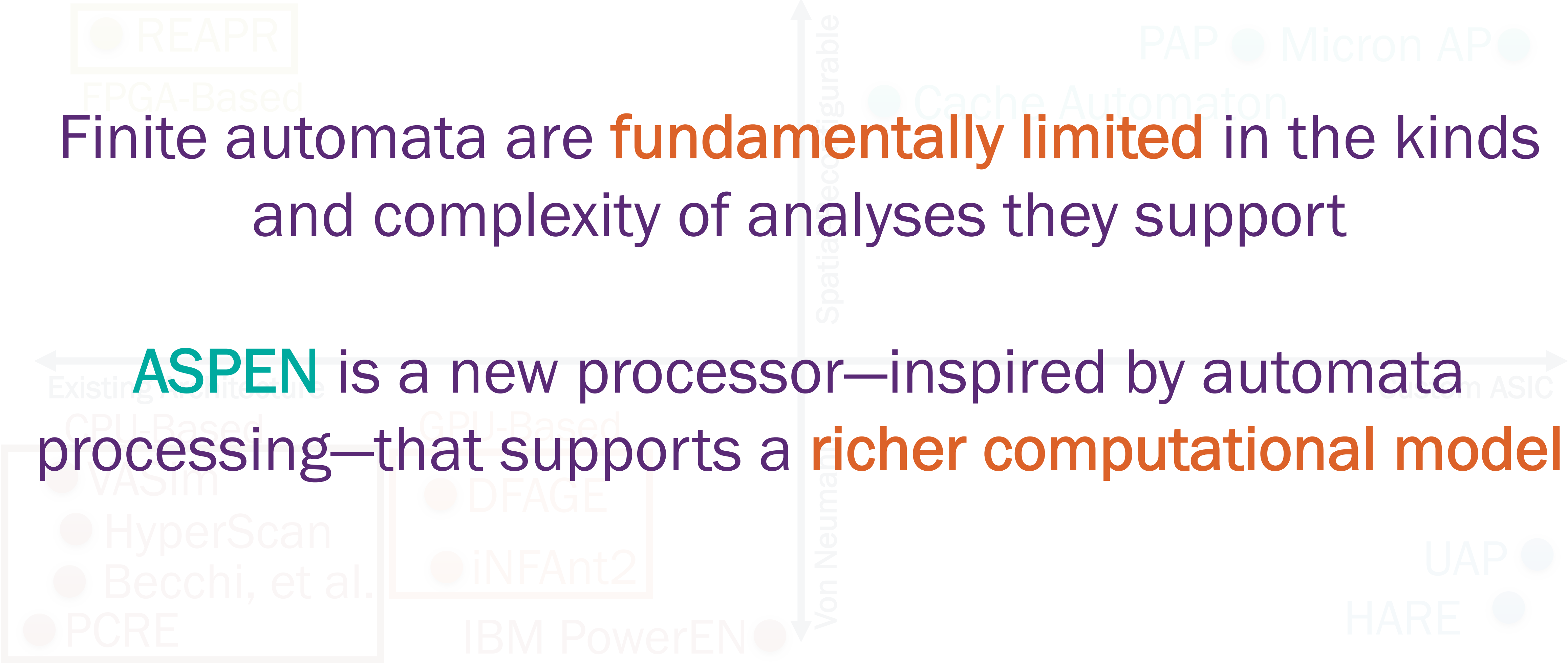
Automata/RegEx Processing Platforms

Finite automata are fundamentally limited in the kinds and complexity of analyses they support

ASPEN is a new processor—inspired by automata processing—that supports a richer computational model



Automata/RegEx Processing Platforms



Finite automata are fundamentally limited in the kinds and complexity of analyses they support

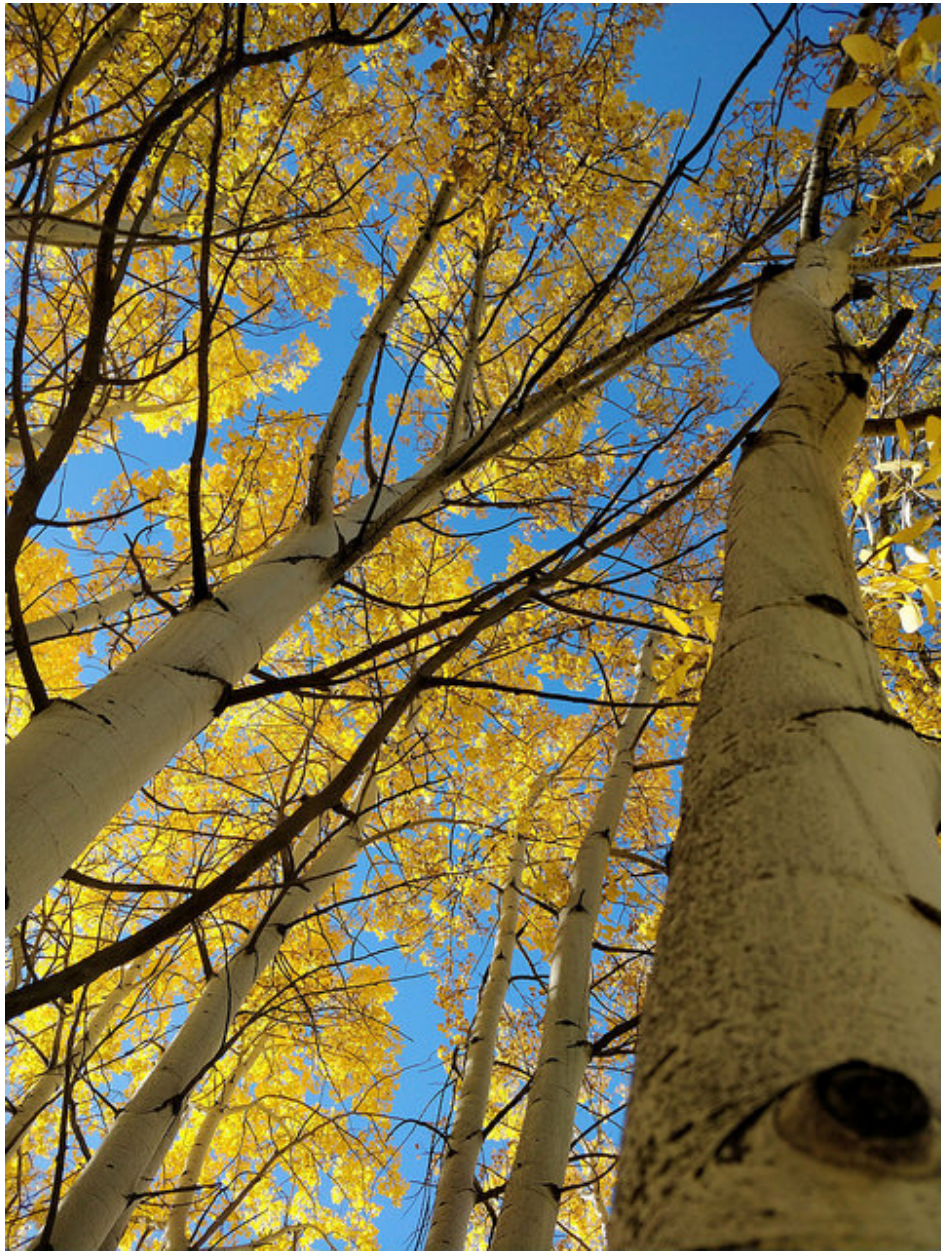
ASPEN is a new processor—inspired by automata processing—that supports a richer computational model

ASPEN Supports Richer Analyses

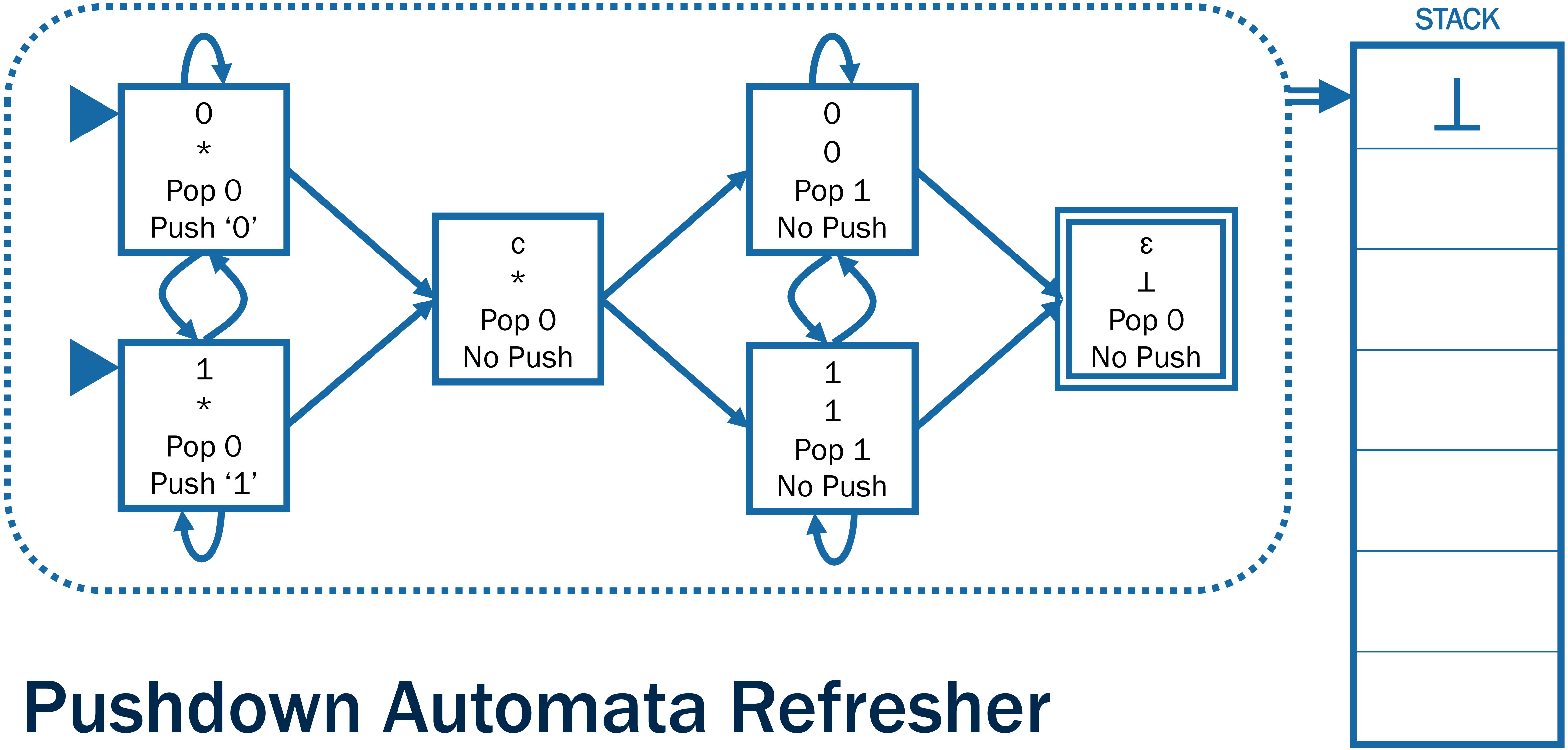
- Accelerated in-SRAM Pushdown ENgine
- Scalable processing engine that uses LLC slices to accelerate Pushdown Automata computation
- Custom five-stage datapath using SRAM lookups can process up to one byte per cycle
- Optimizing compiler supports existing grammars, packs states efficiently, and reduces the number processing stalls
- Provides additional cache when not in use

Overview of this Talk

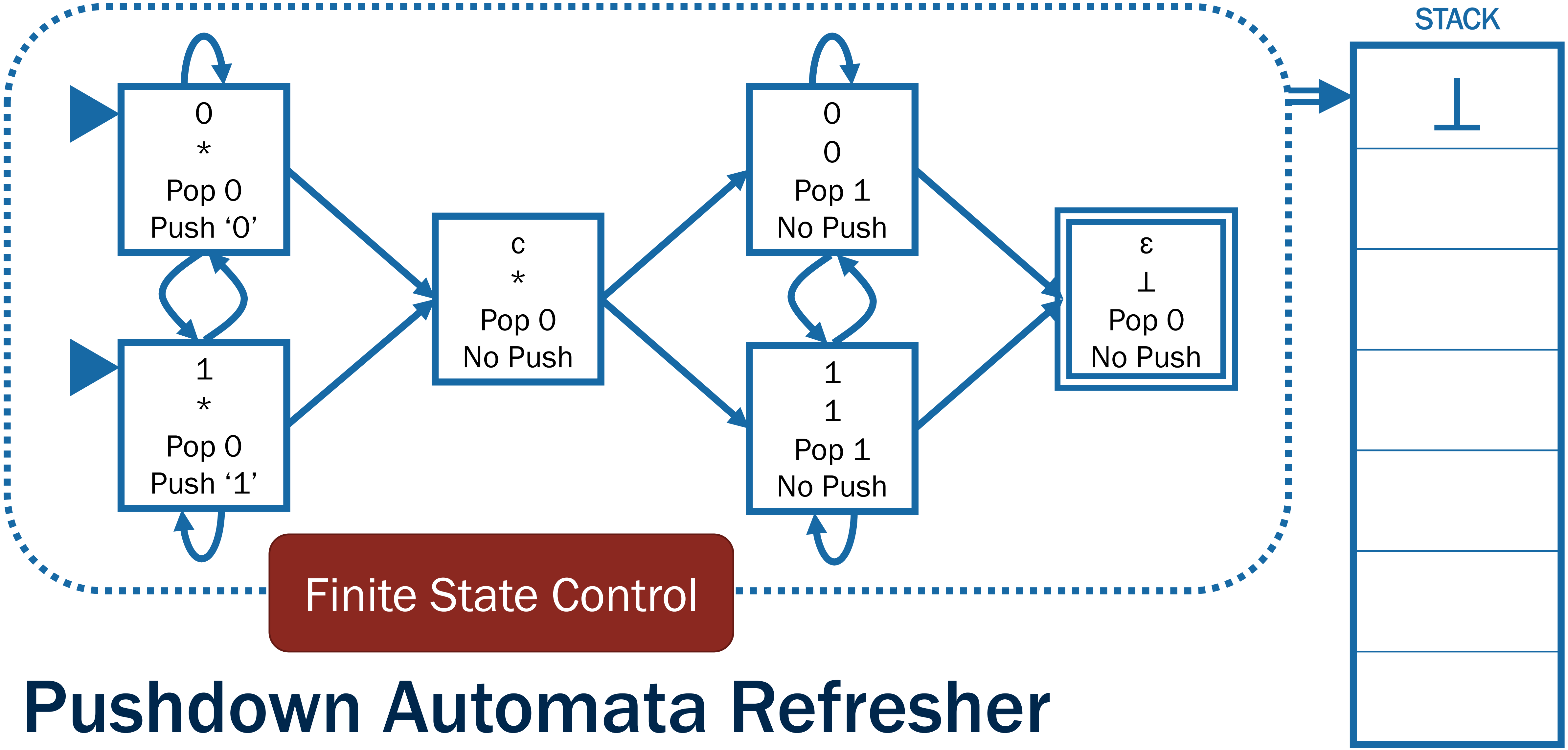
- Pushdown Automata Refresher
- Architectural Design of ASPEN
  - Why LLC?
  - Datapath innovation
- Optimizations
- Evaluation
  - XML Parsing
  - Subtree Mining



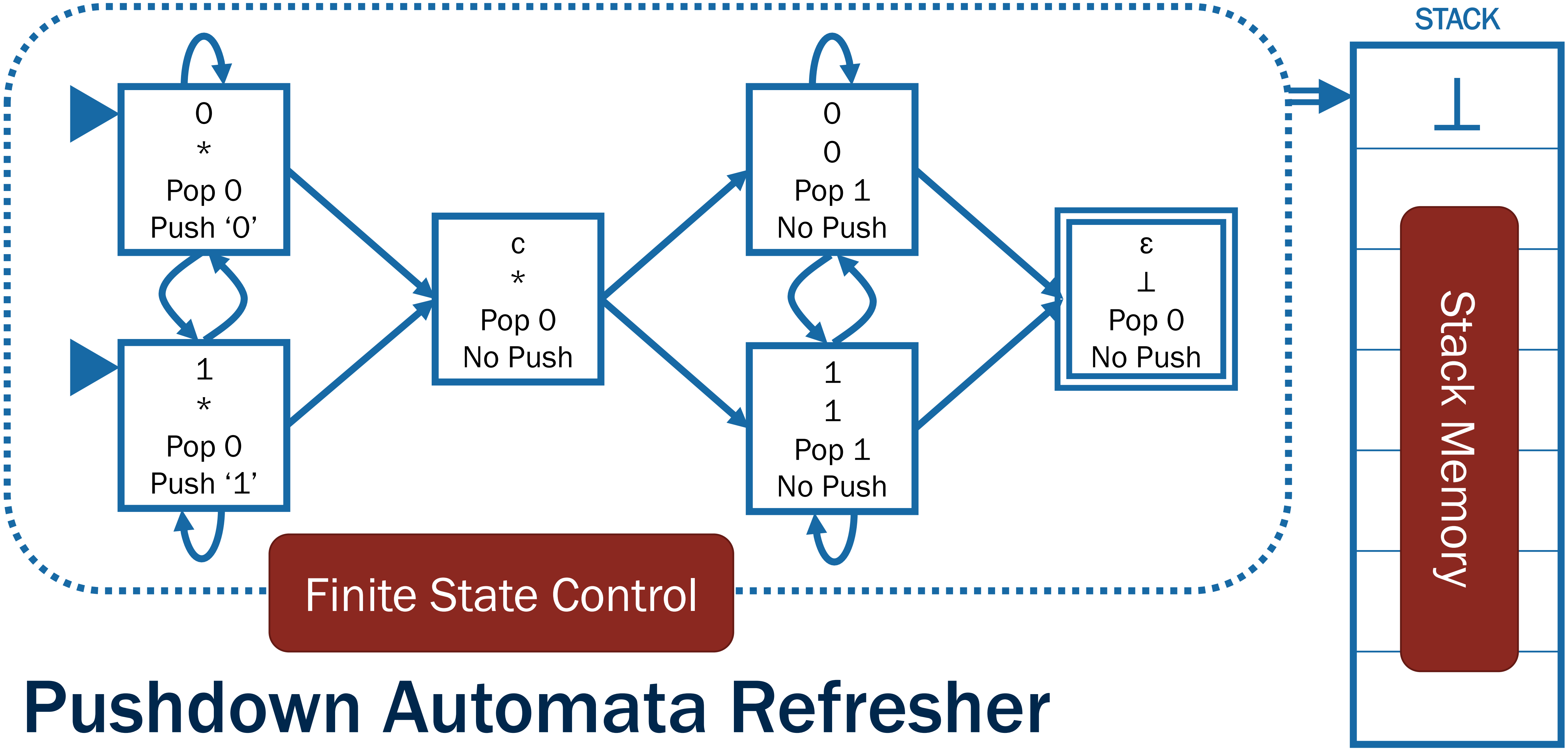
Source: https://www.flickr.com/photos/10623456@N02/45022262771 CC BY-NC 2.0



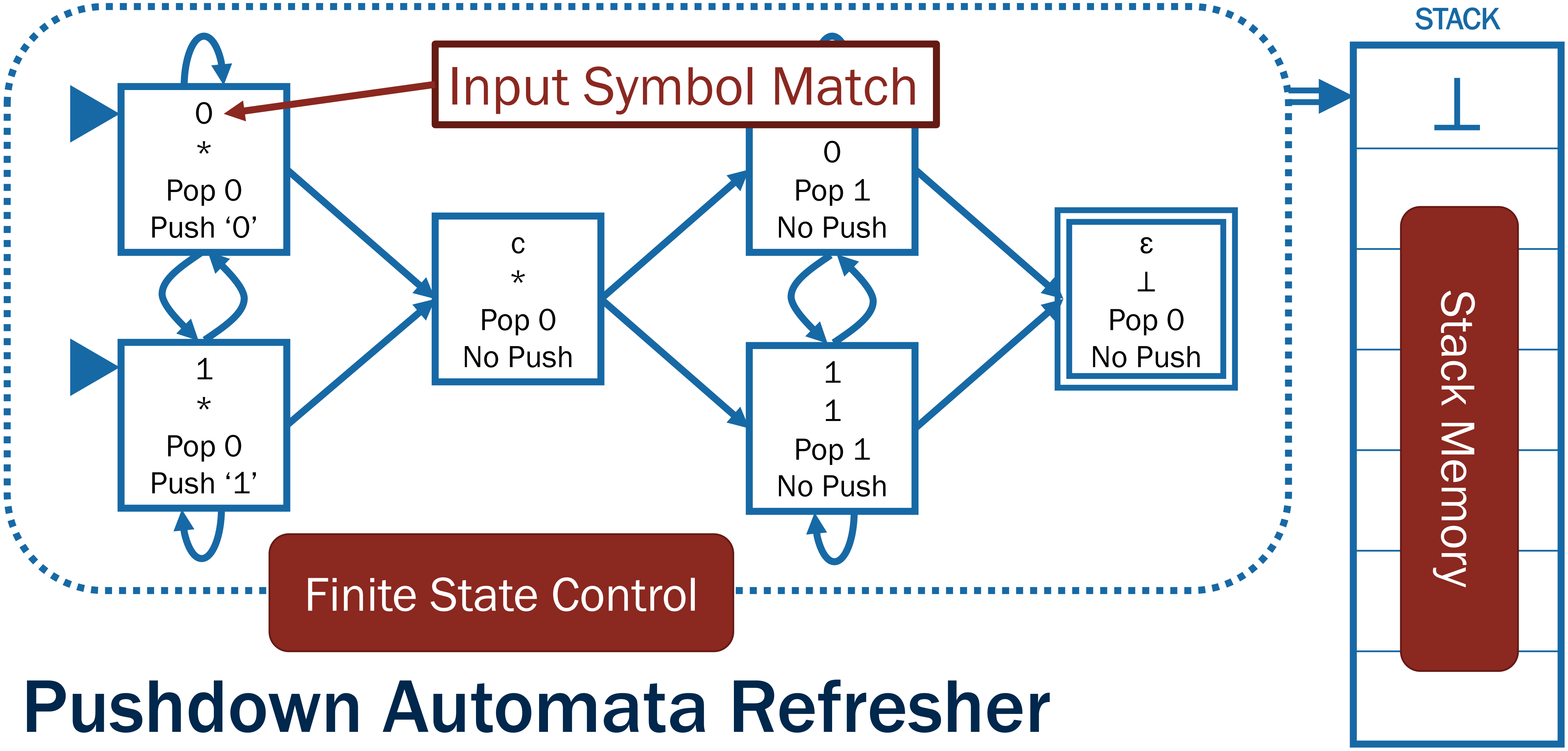
Pushdown Automata Refresher



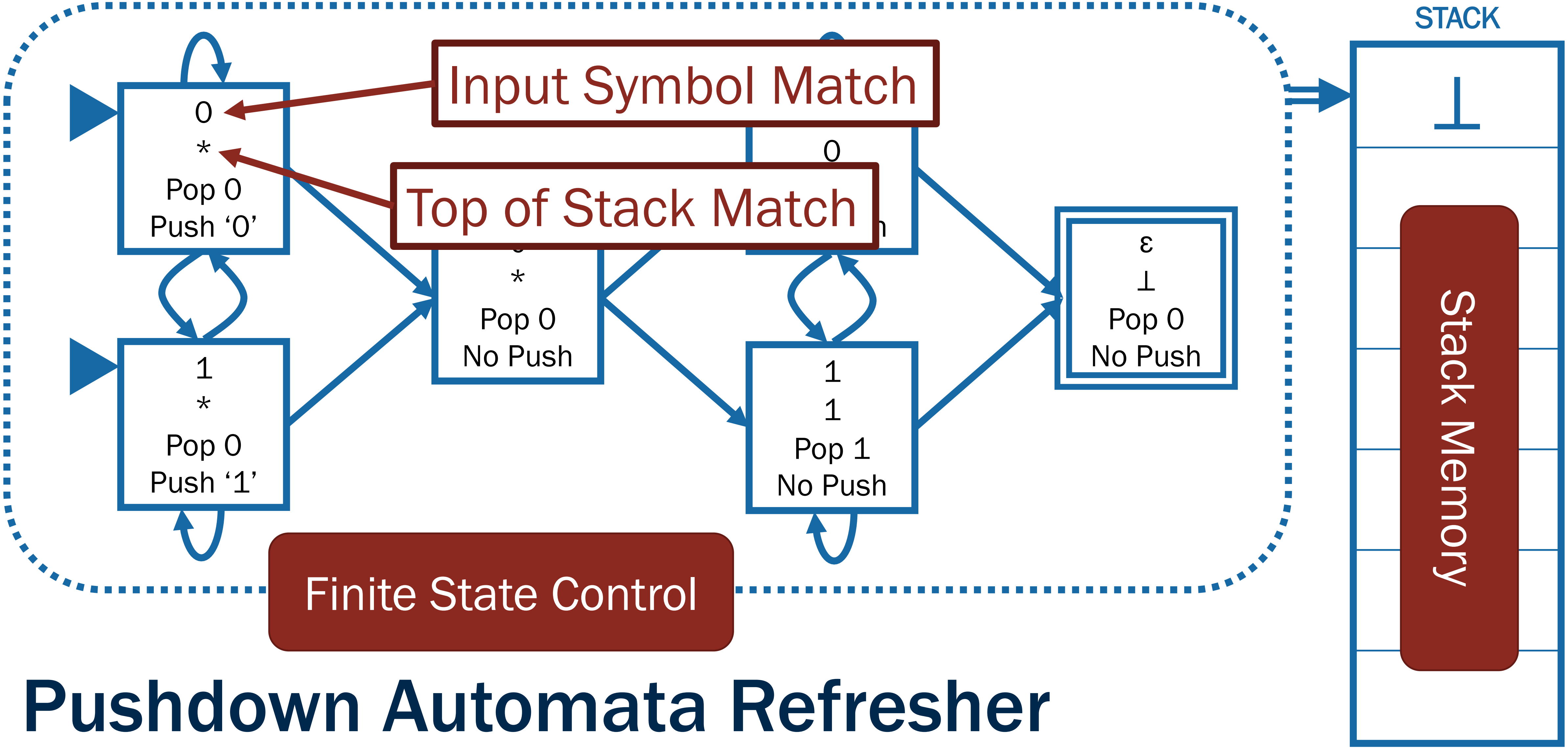
Pushdown Automata Refresher



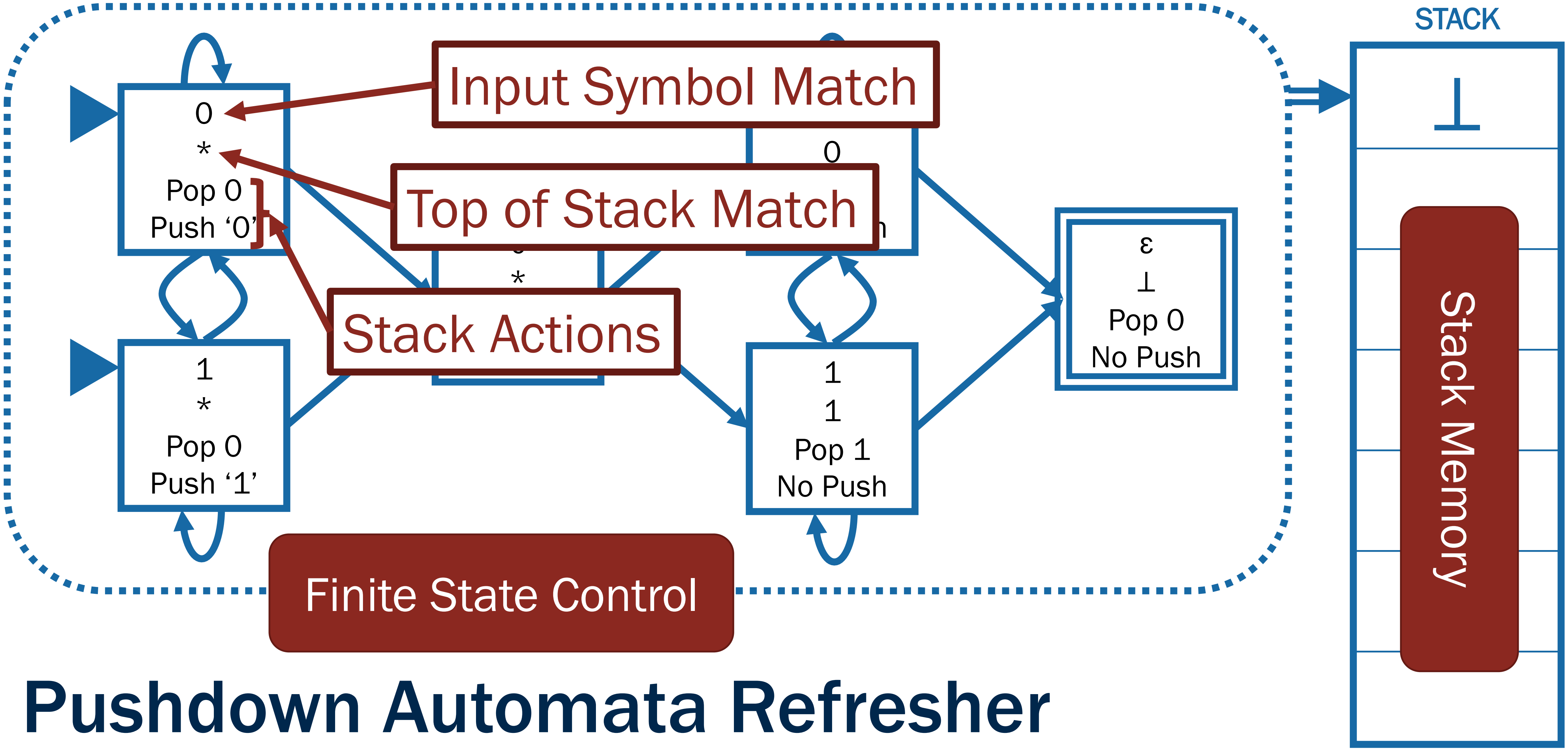
Pushdown Automata Refresher



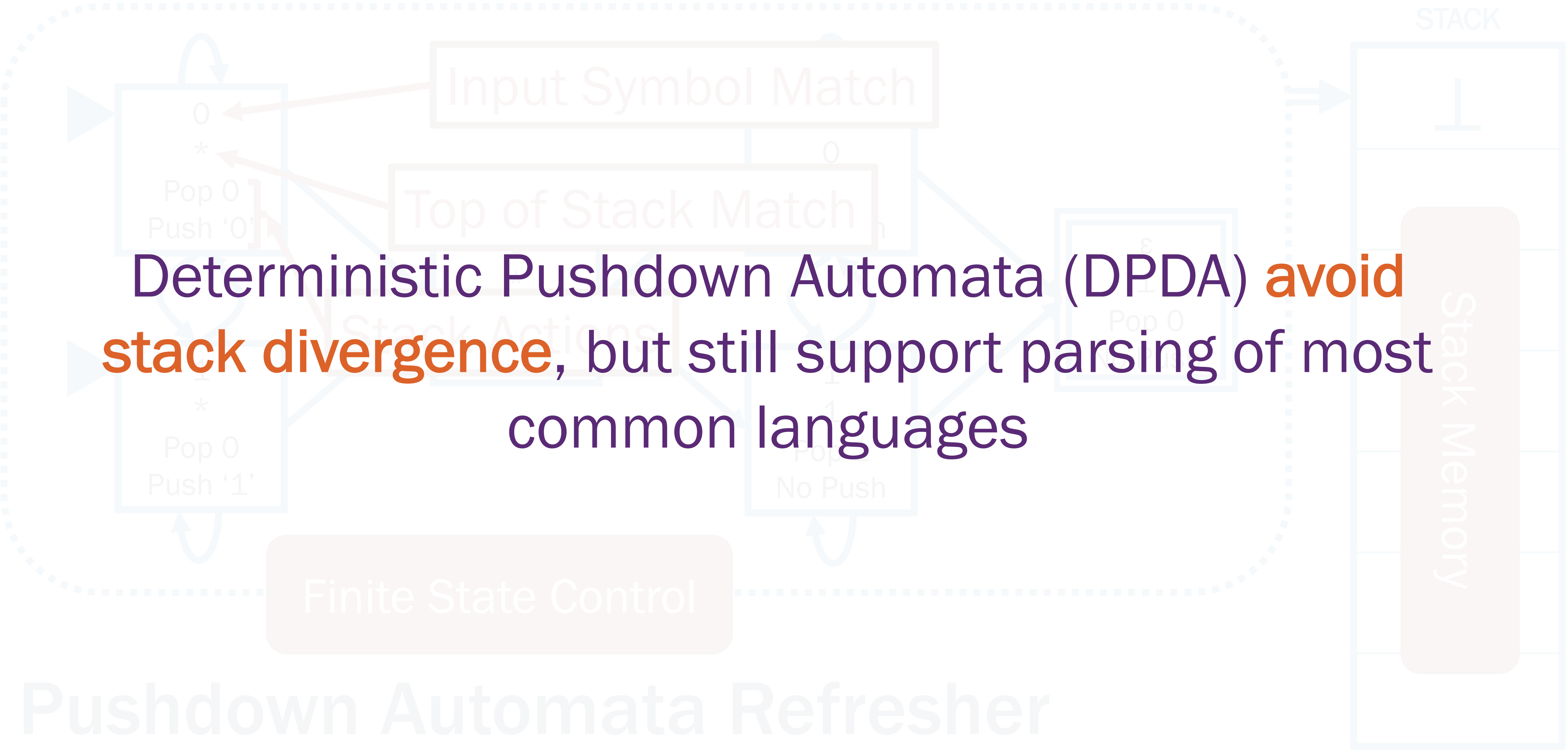
Pushdown Automata Refresher



Pushdown Automata Refresher

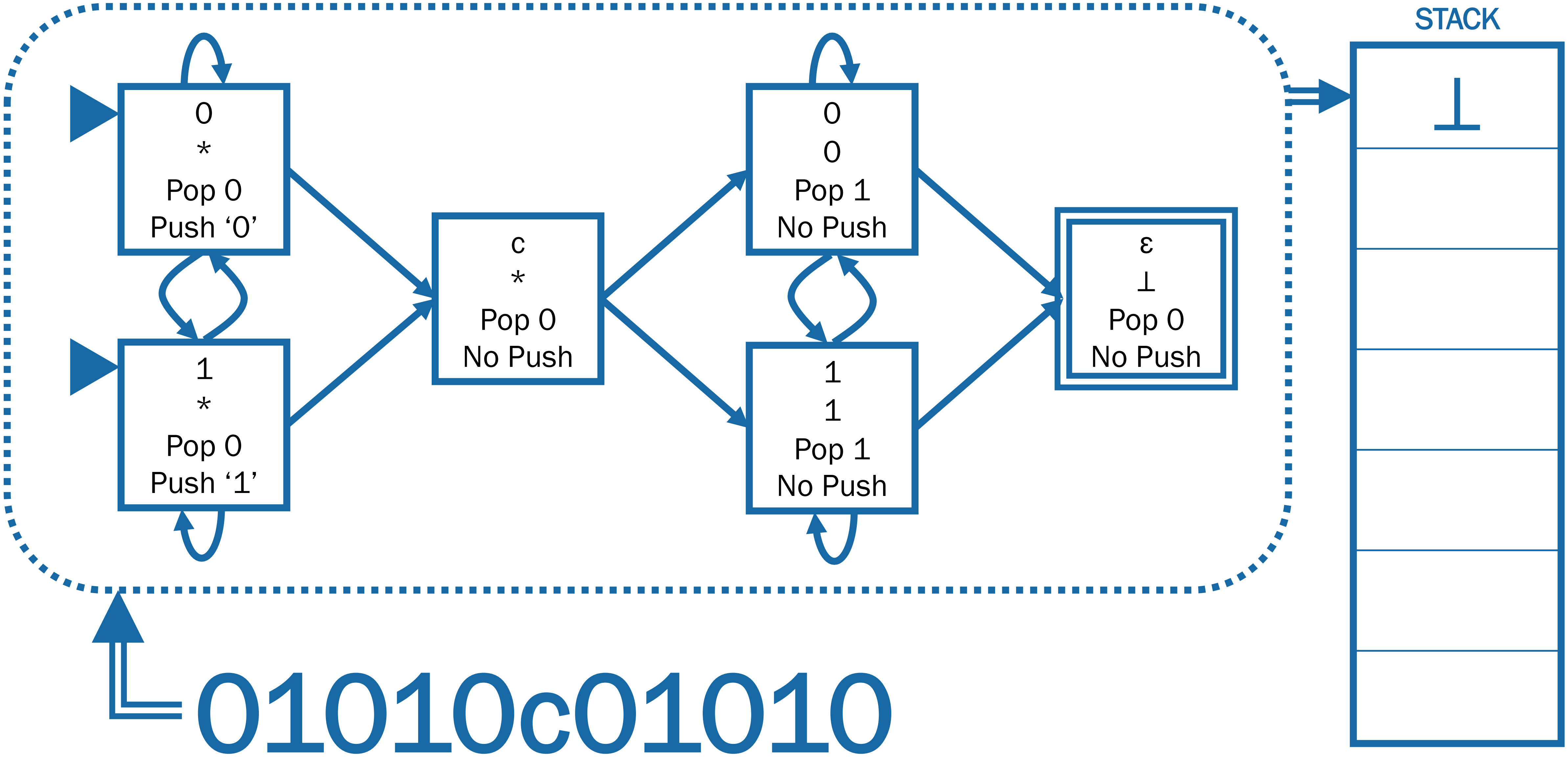


Pushdown Automata Refresher

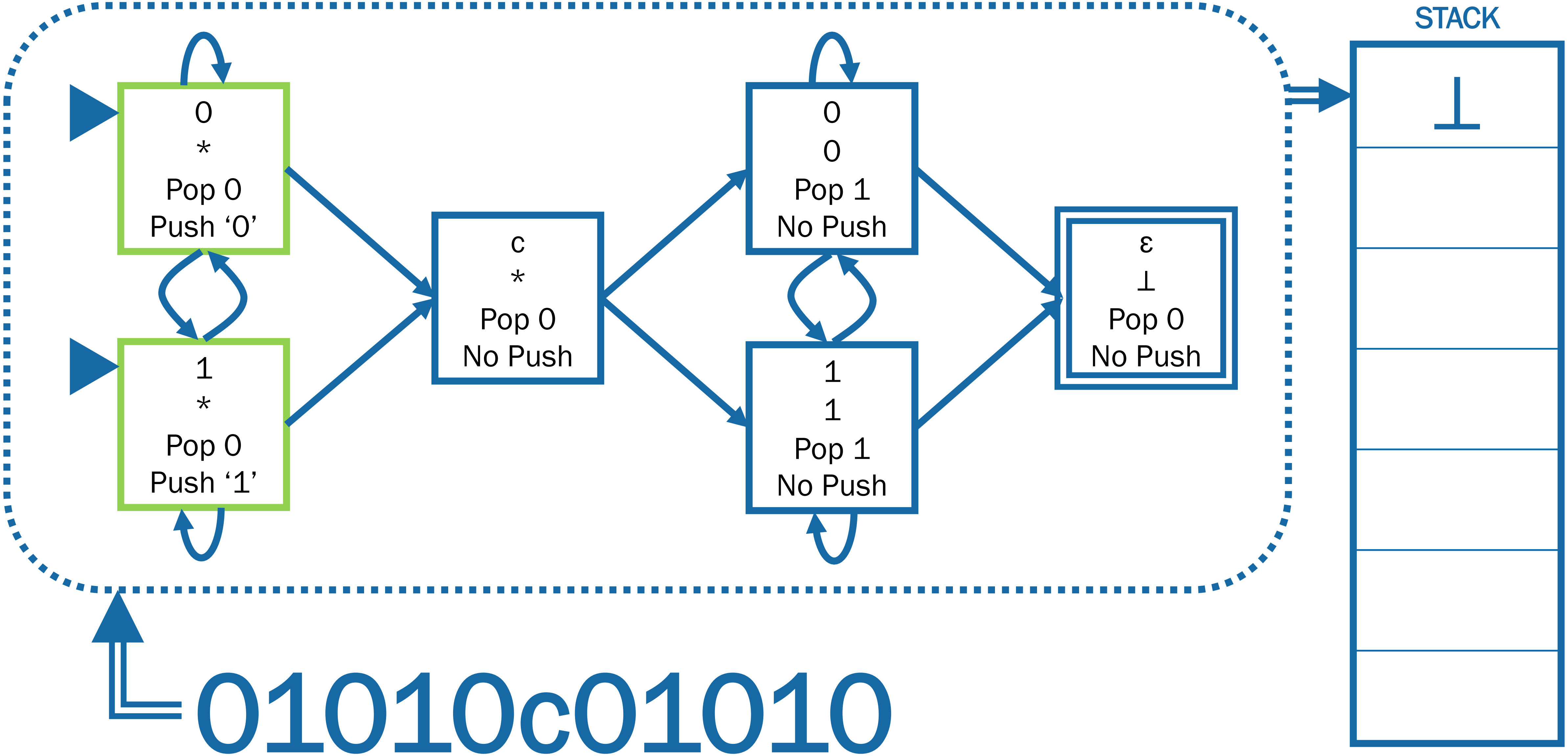


Deterministic Pushdown Automata (DPDA) avoid stack divergence, but still support parsing of most common languages

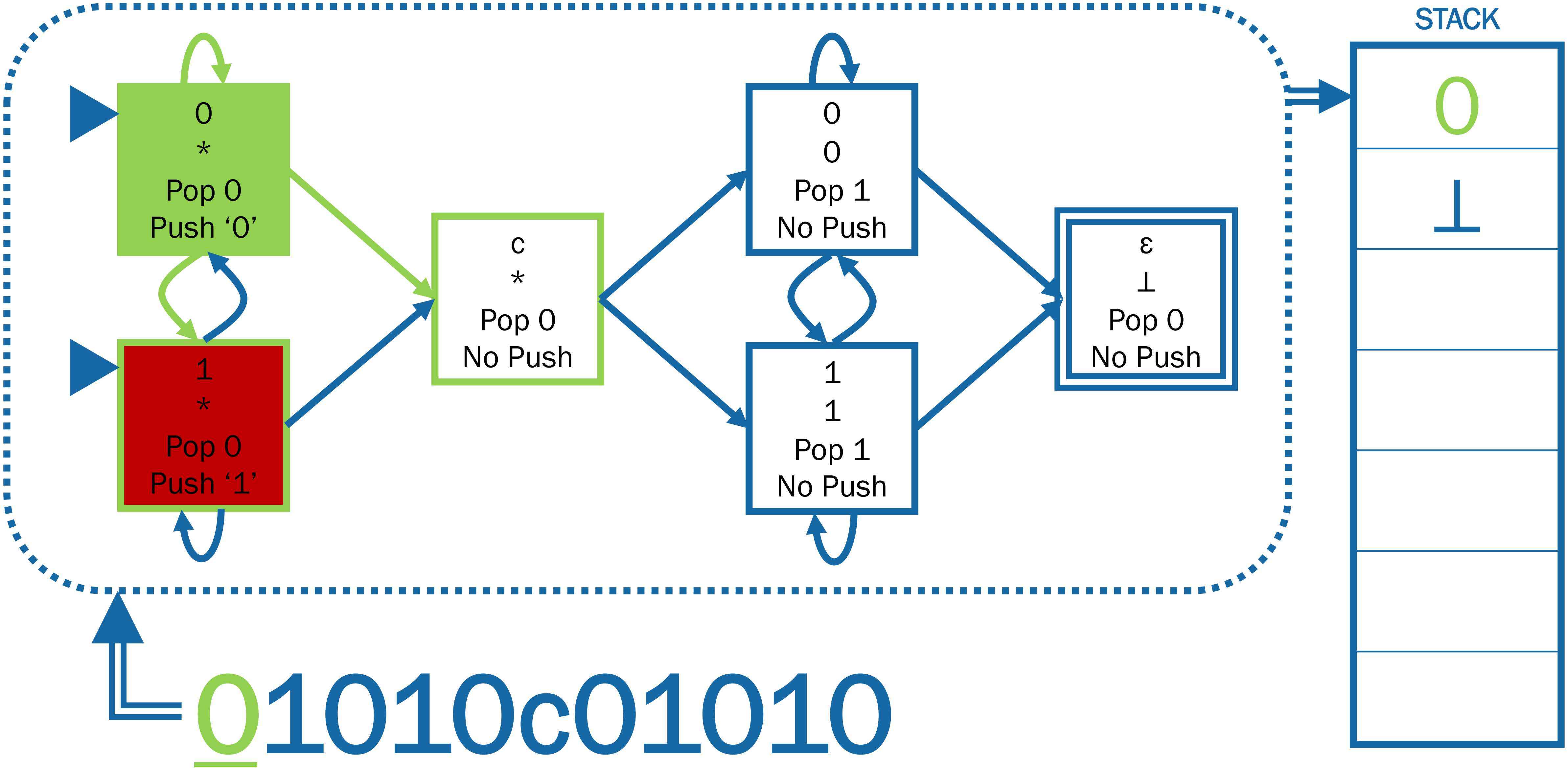
Recognizing Palindromes with a Middle Character



Recognizing Palindromes with a Middle Character



Recognizing Palindromes with a Middle Character



0
*
Pop 0
Push ‘0’

1
*
Pop 0
Push ‘1’

c
*
Pop 0
No Push

0
0
Pop 1
No Push

1
1
Pop 1
No Push

ε
⊥
Pop 0
No Push

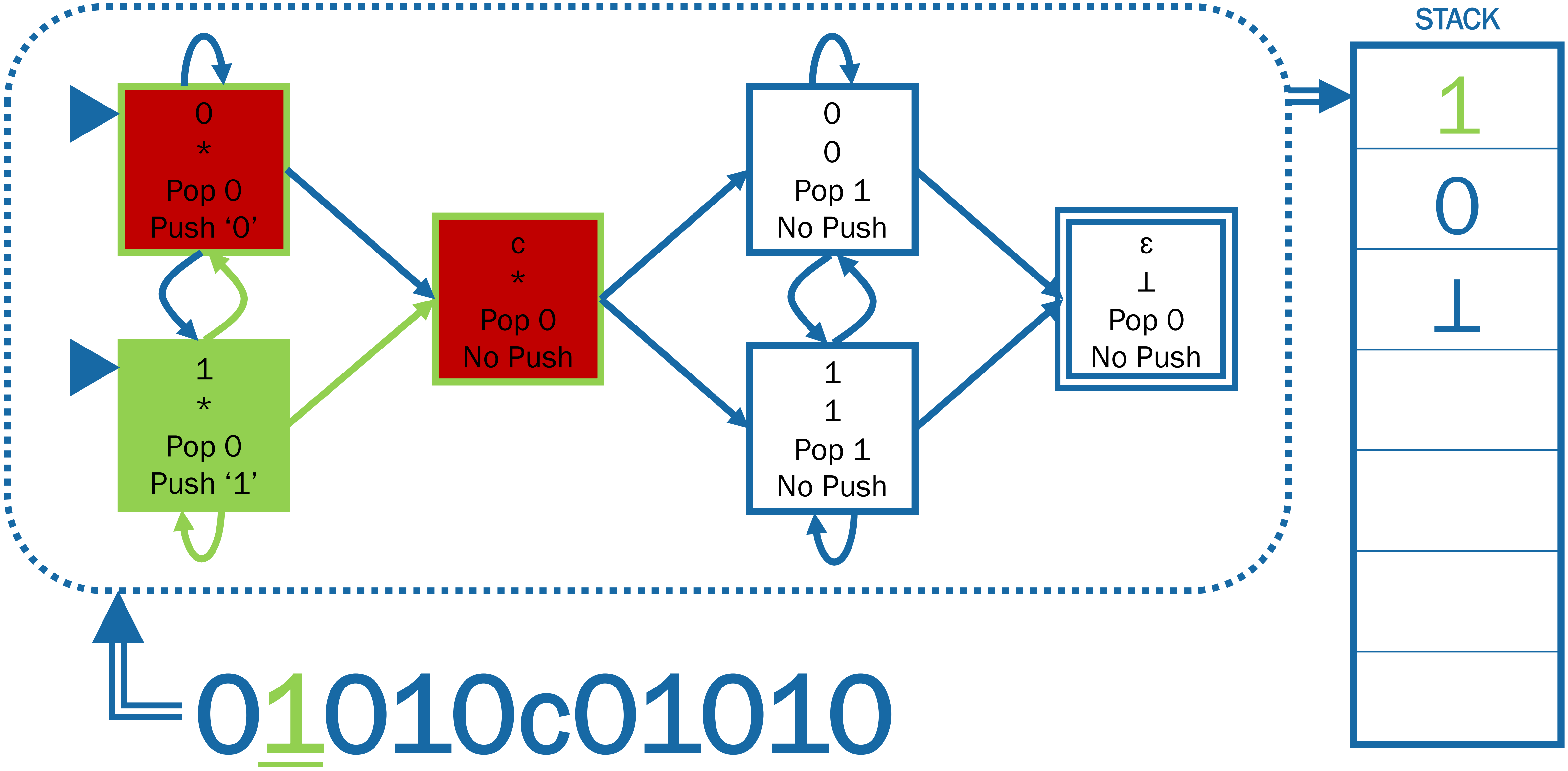
STACK

0

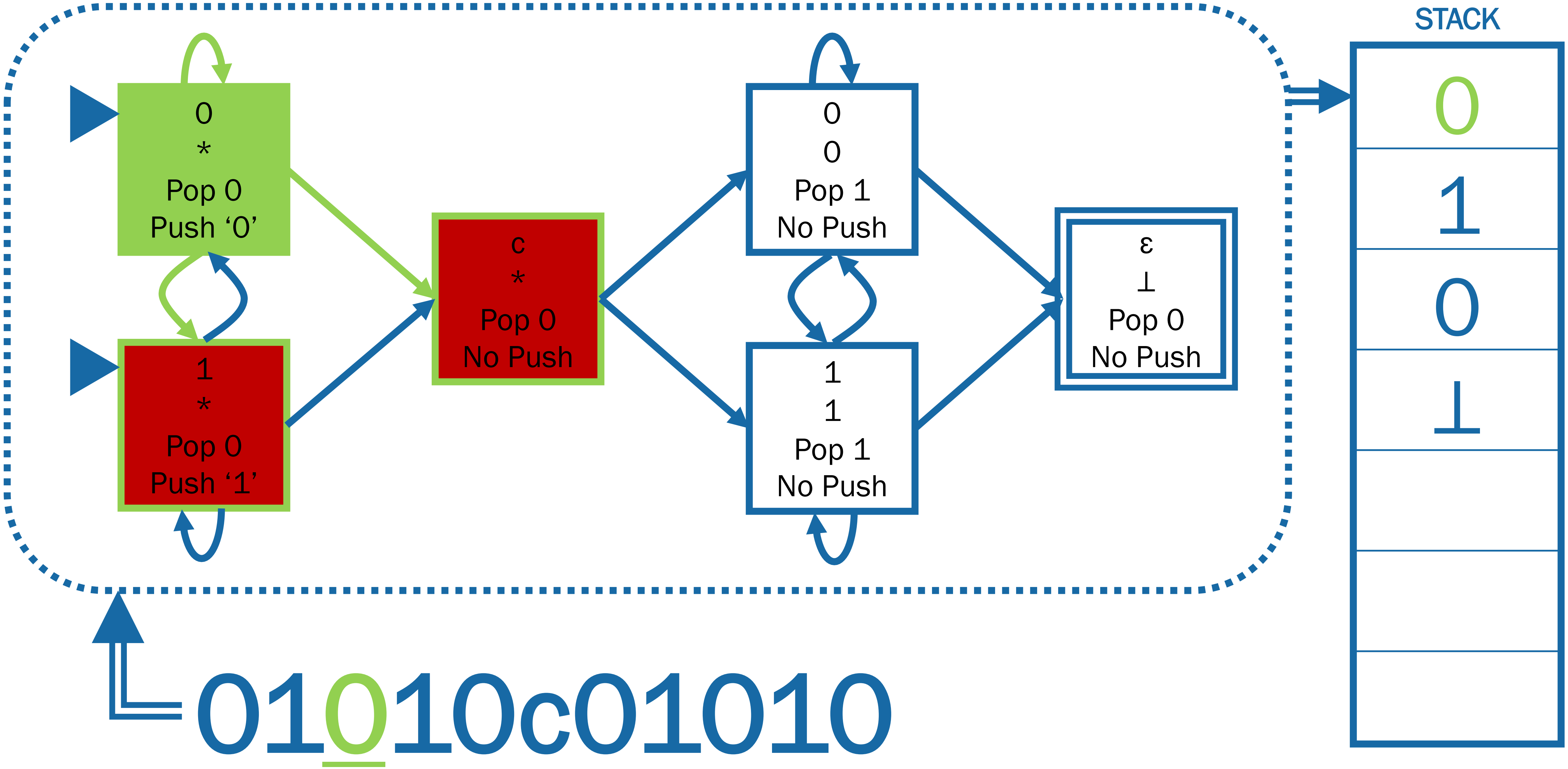
⊥

01010c01010

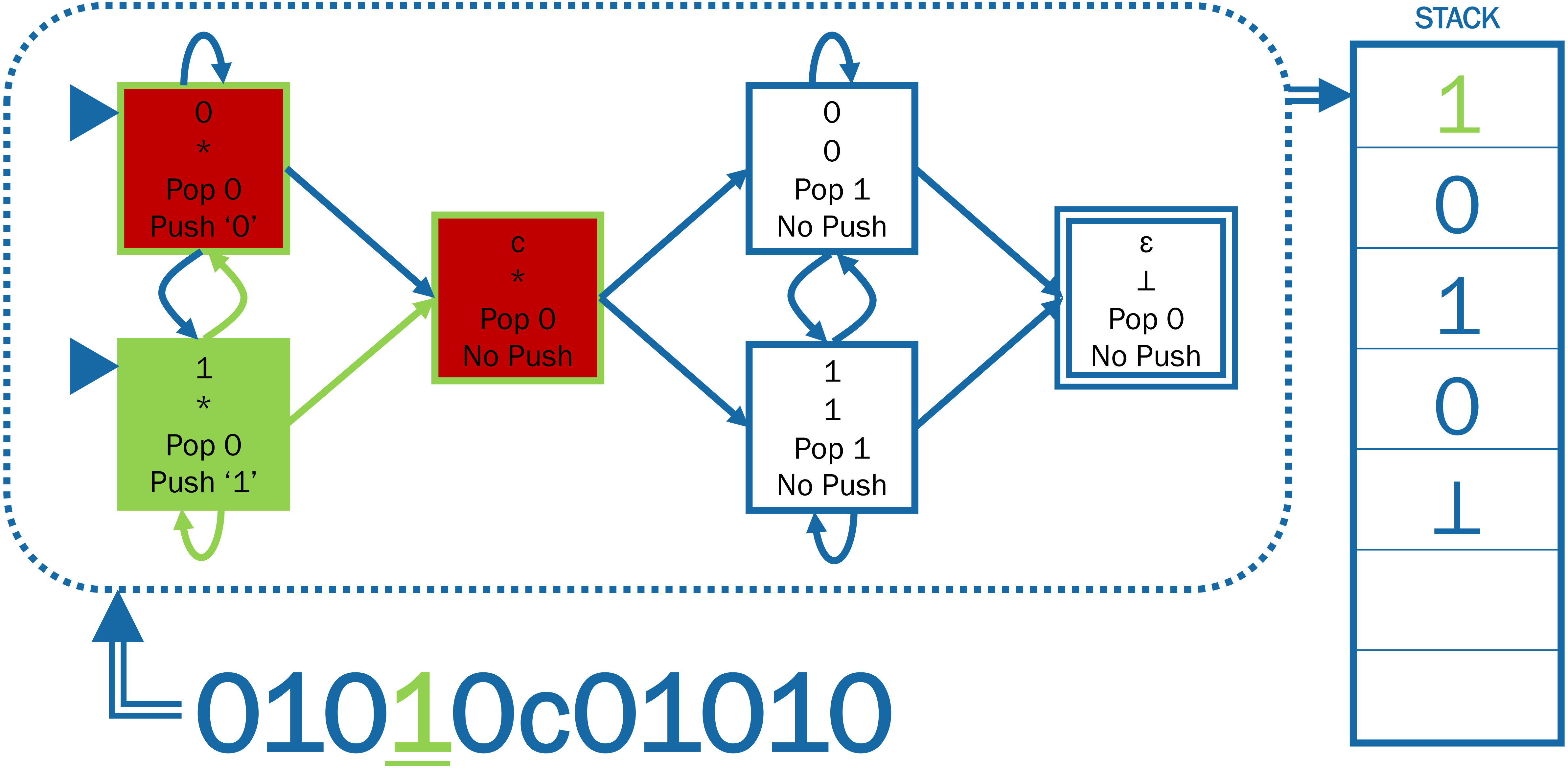
Recognizing Palindromes with a Middle Character



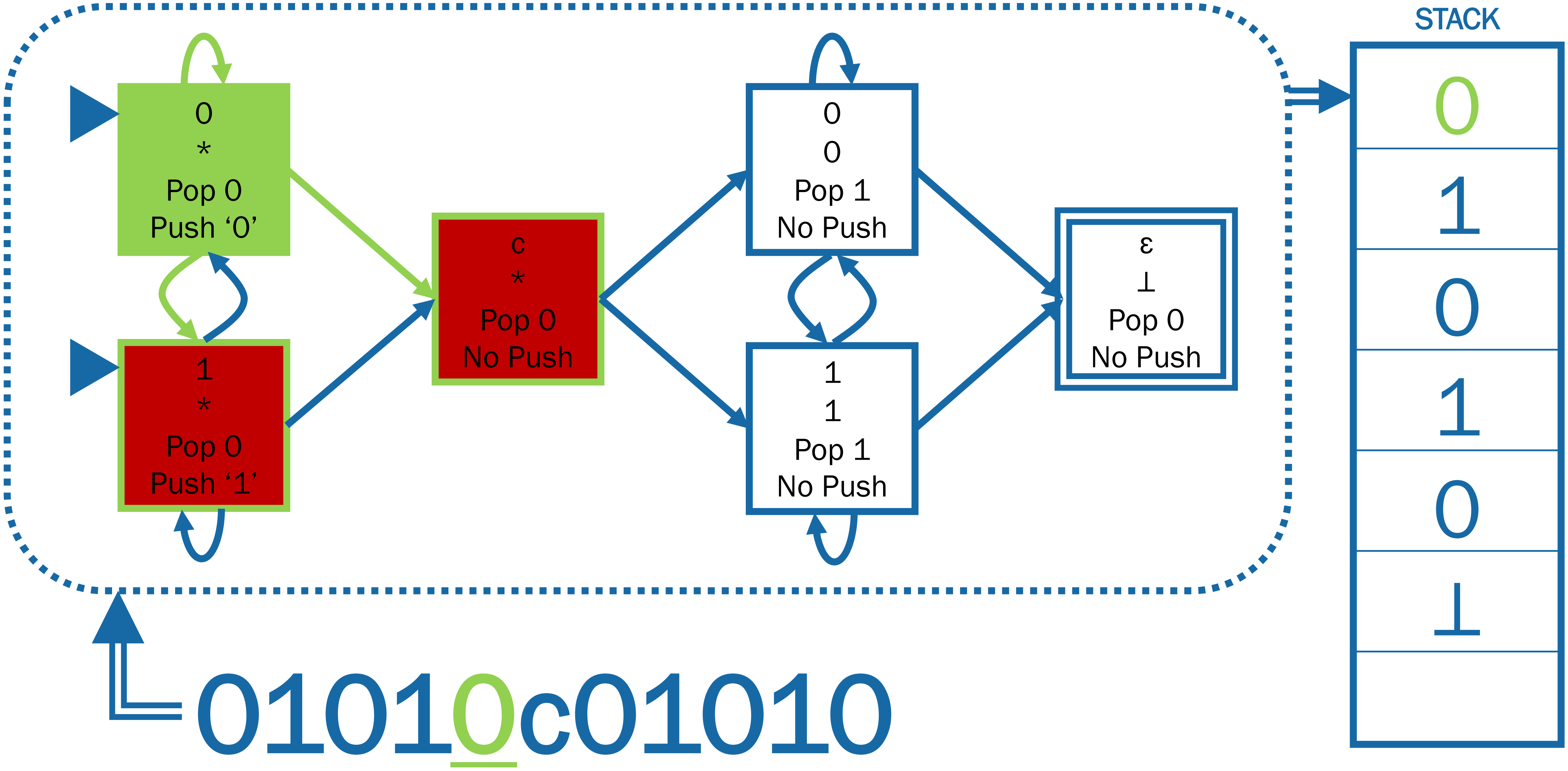
Recognizing Palindromes with a Middle Character



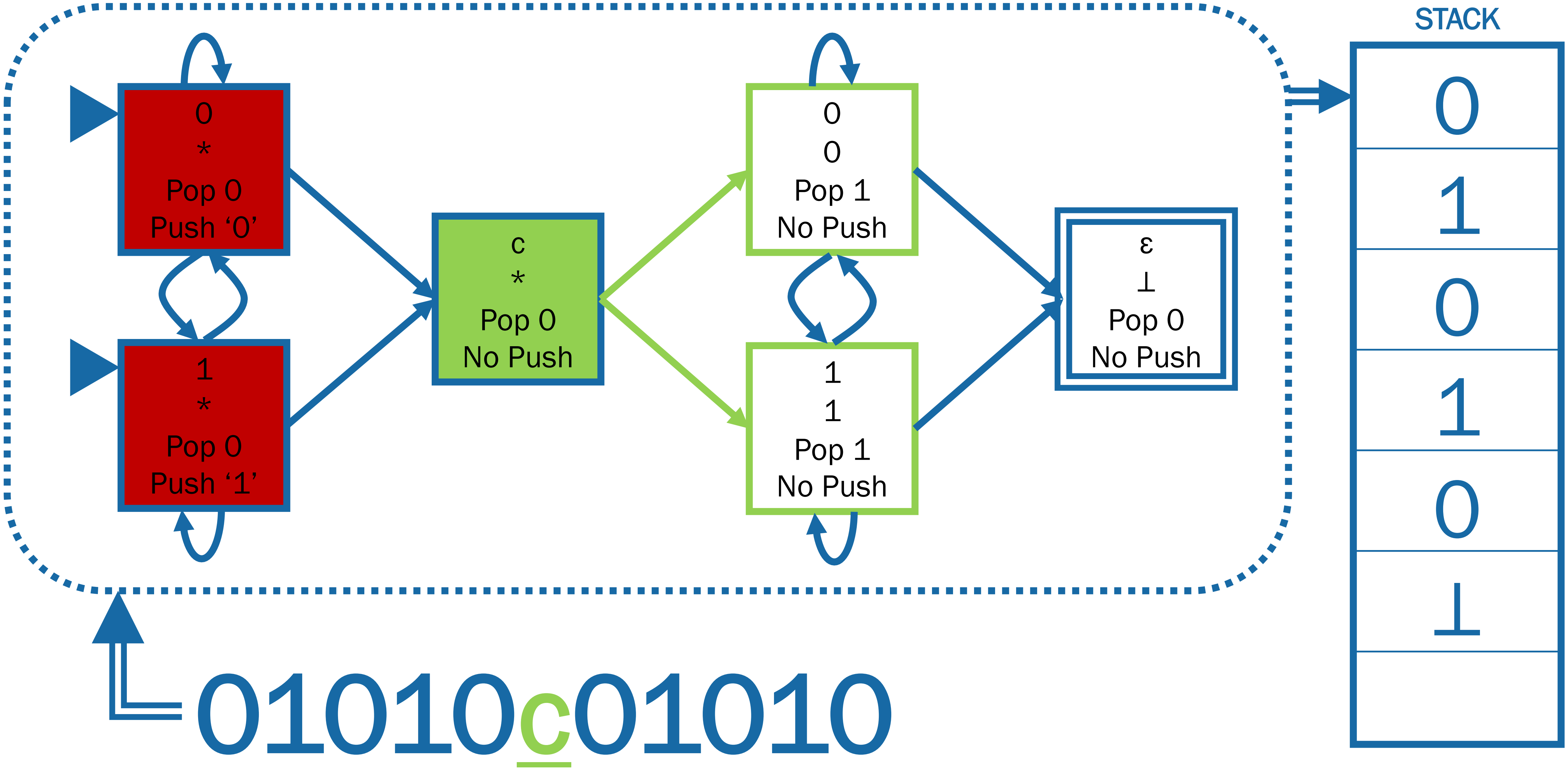
Recognizing Palindromes with a Middle Character



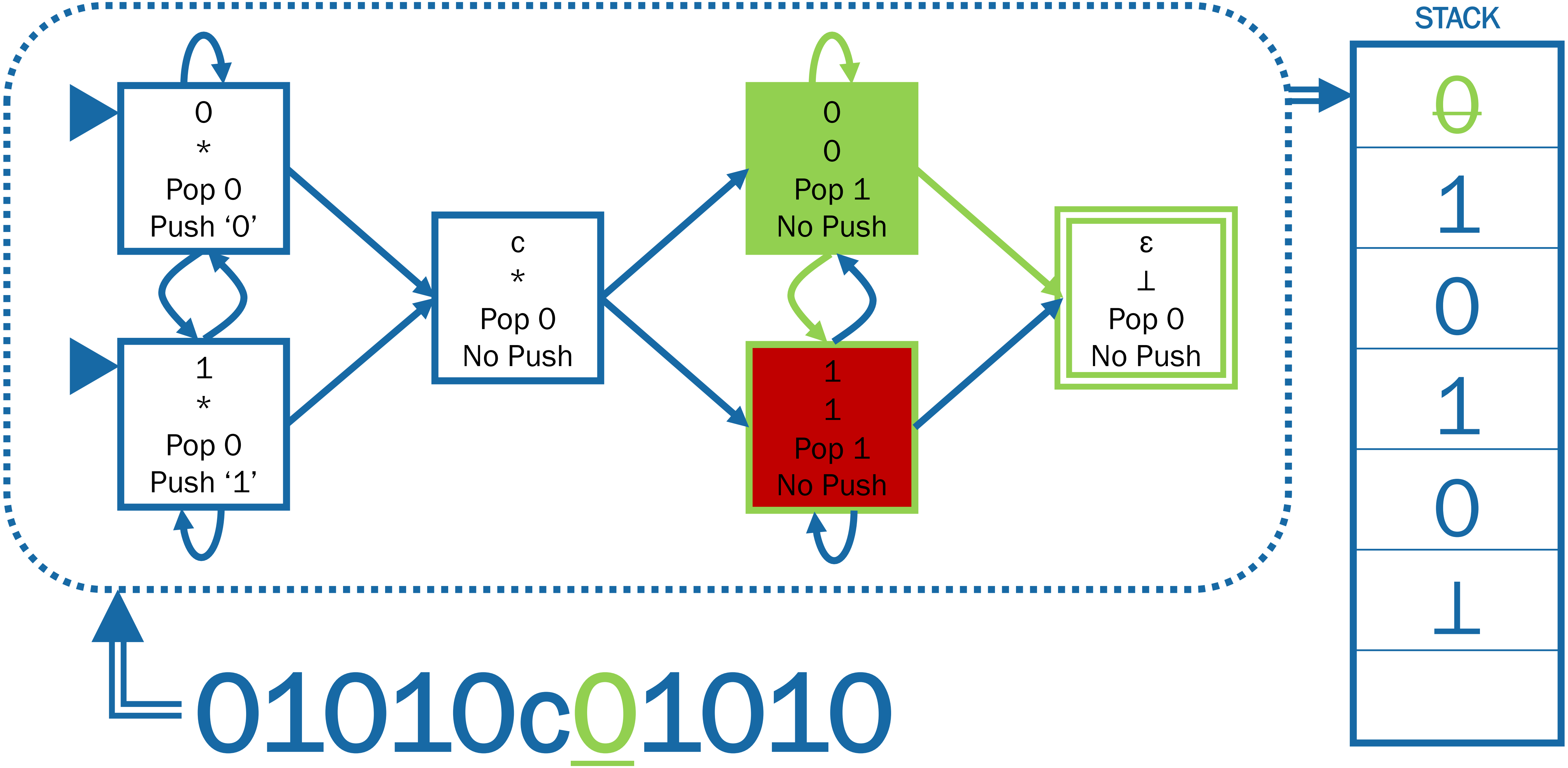
Recognizing Palindromes with a Middle Character



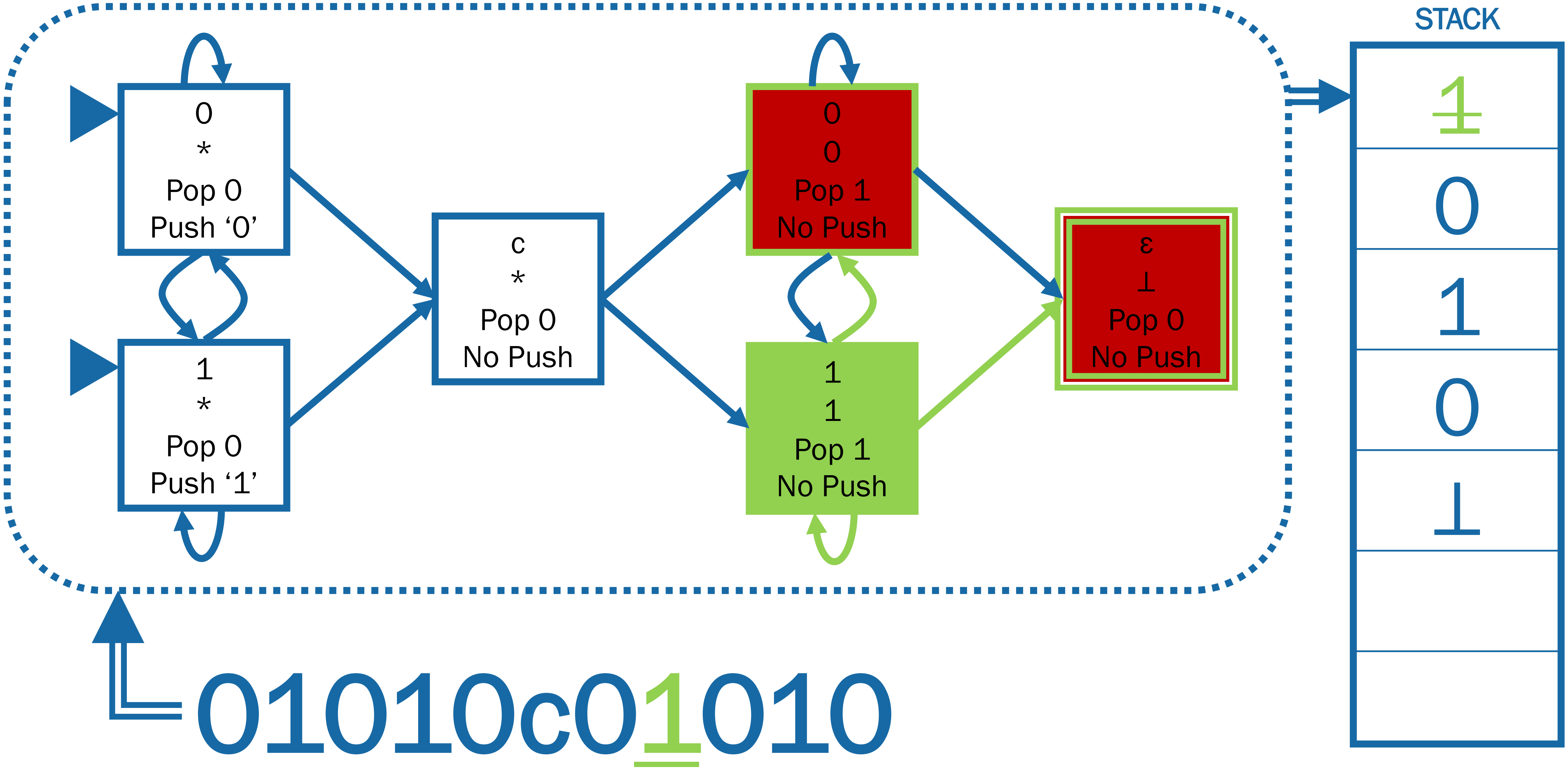
Recognizing Palindromes with a Middle Character



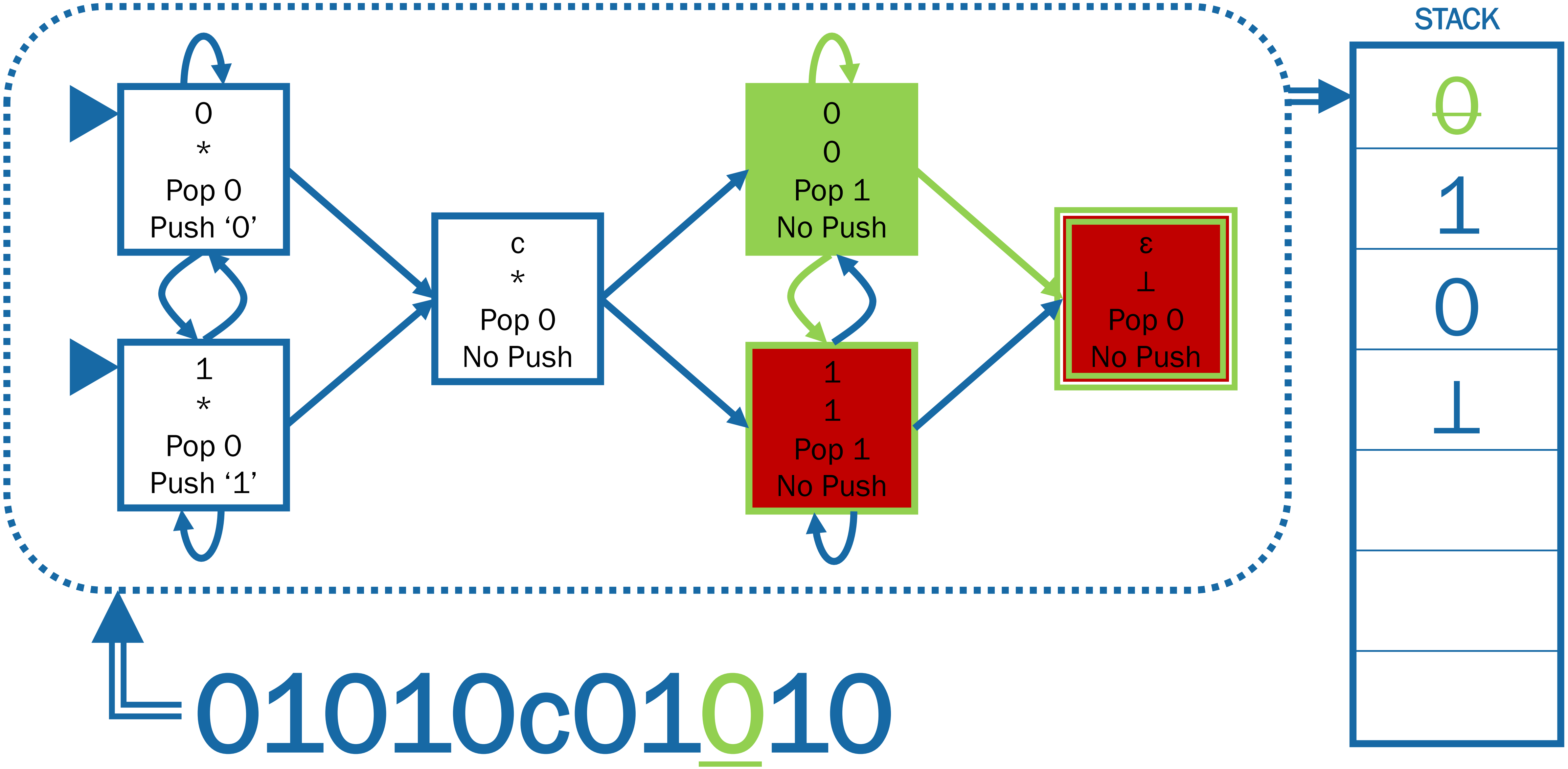
Recognizing Palindromes with a Middle Character



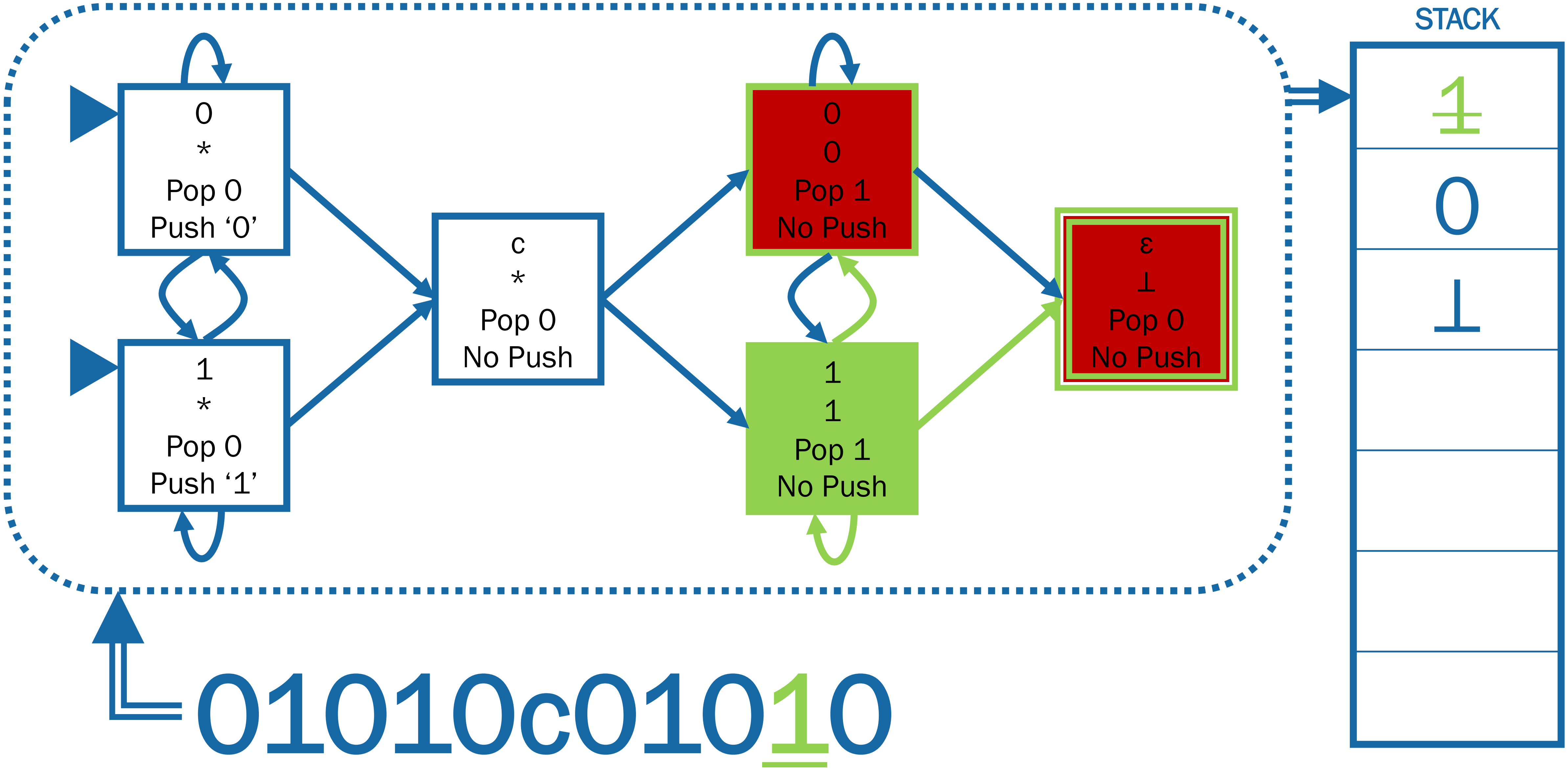
Recognizing Palindromes with a Middle Character



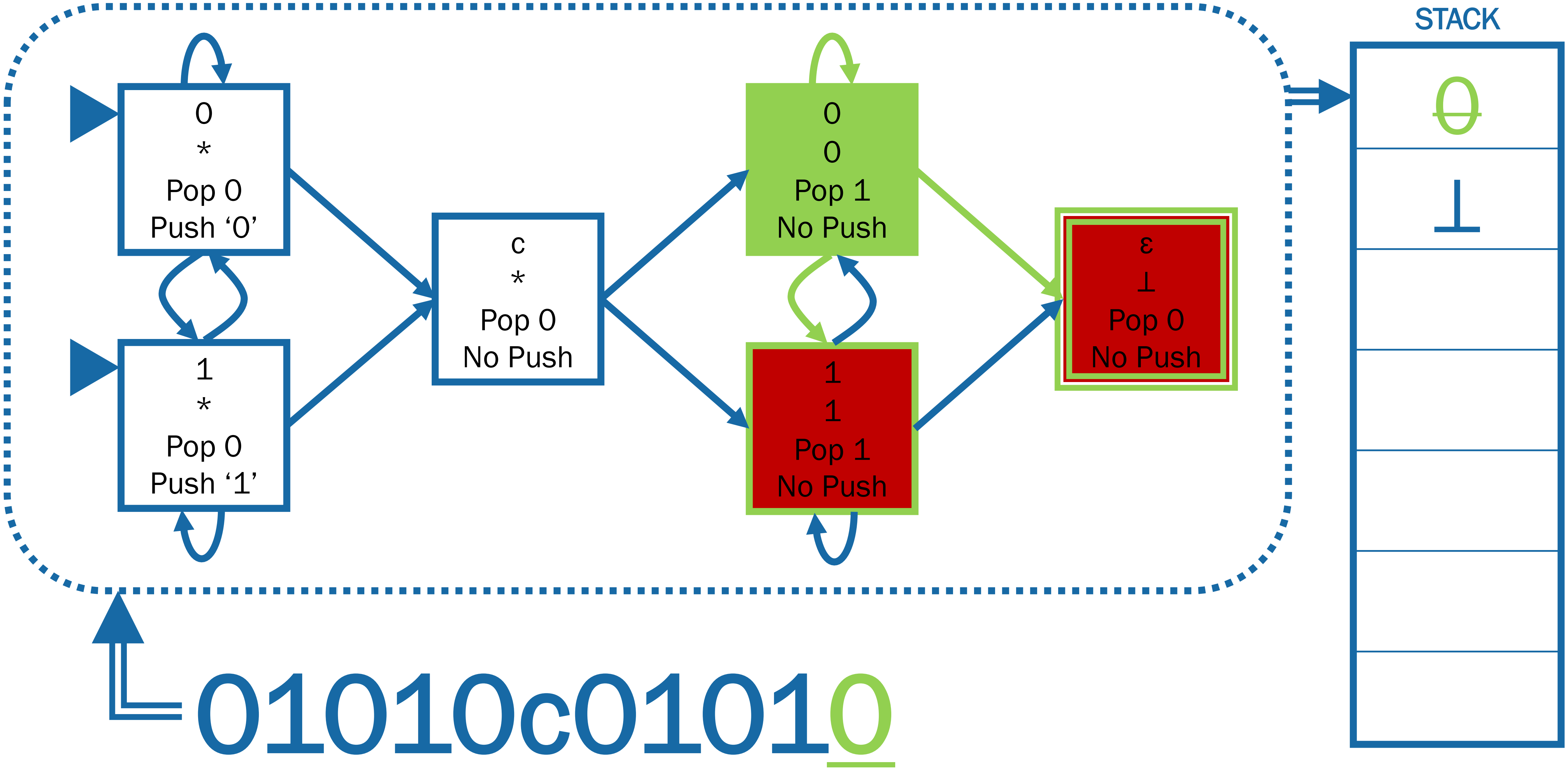
Recognizing Palindromes with a Middle Character



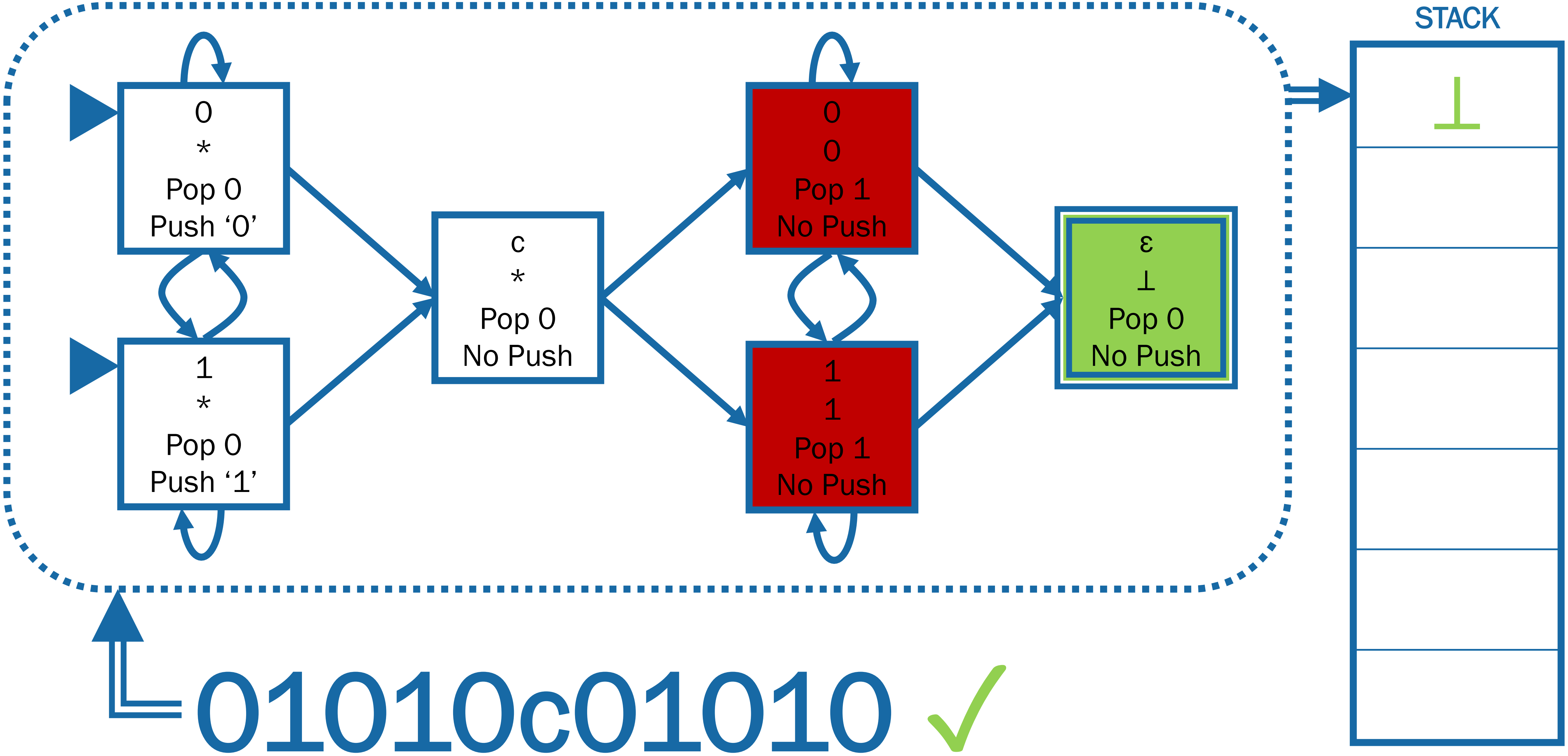
Recognizing Palindromes with a Middle Character



Recognizing Palindromes with a Middle Character



Recognizing Palindromes with a Middle Character

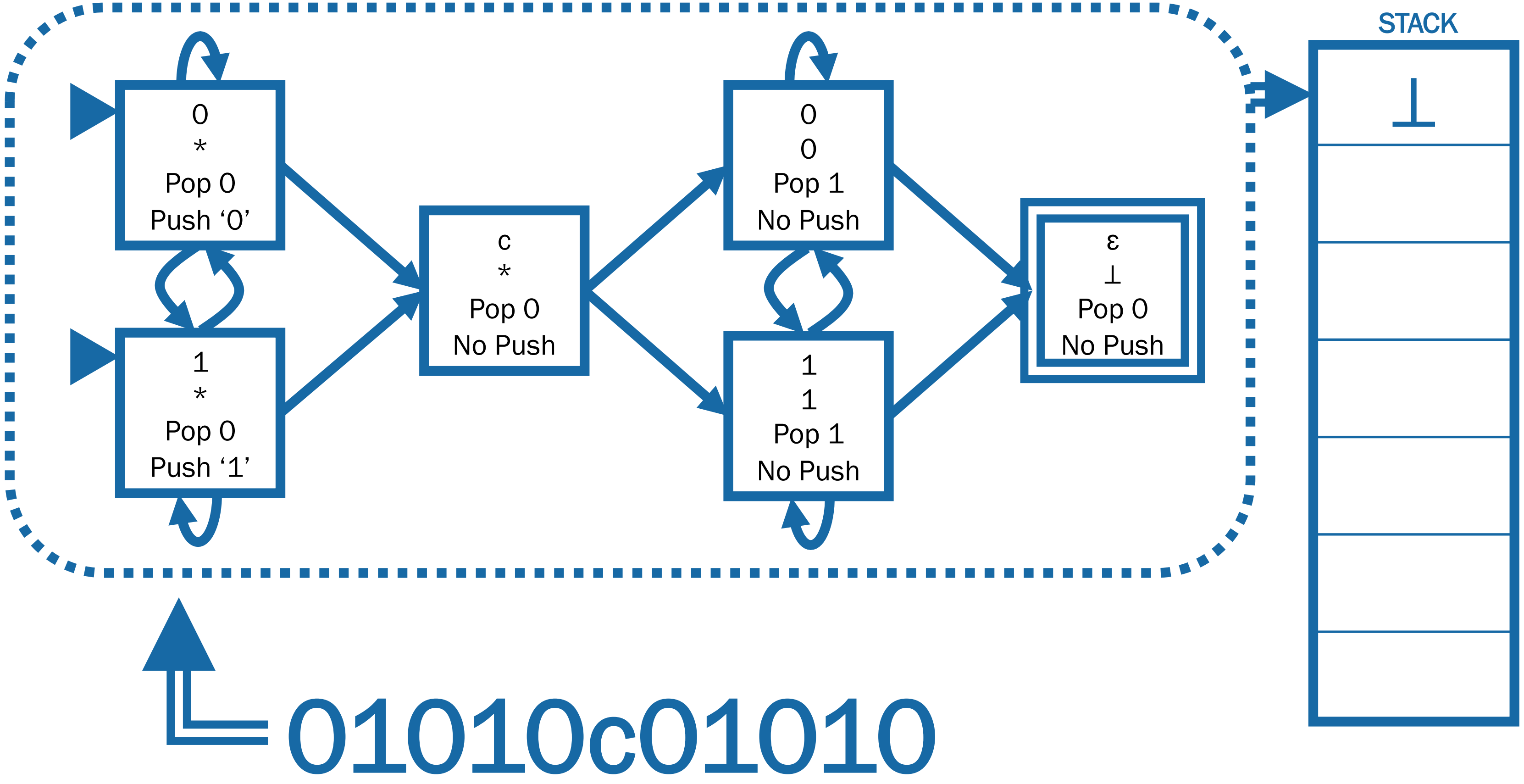


01010c01010 ✓

Mapping DPDA Efficiently to Hardware

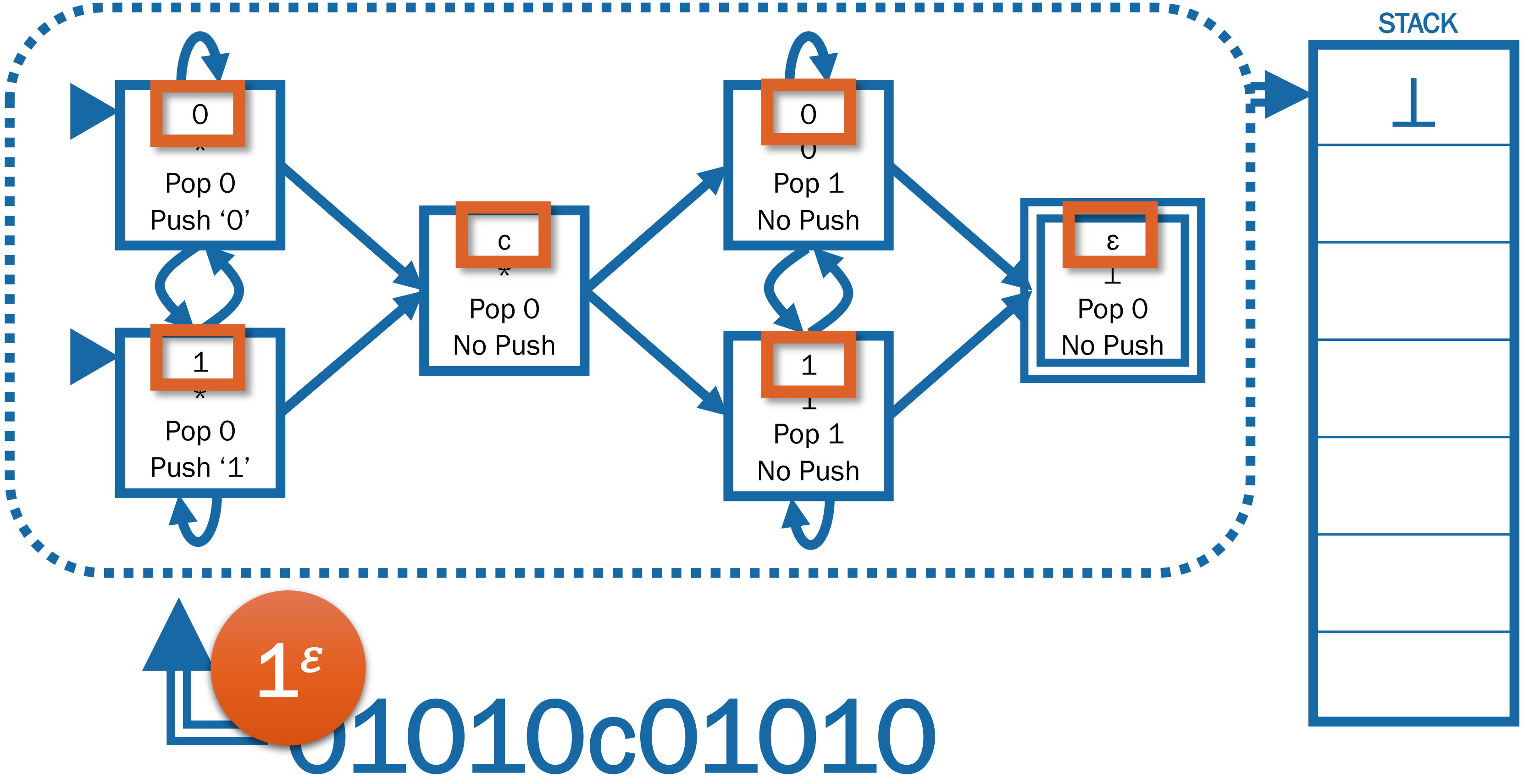
- ASPEN supports homogeneous DPDA
  - All transitions to a state occur on the same input character, stack comparison, and stack operation
  - Similar in nature to homogeneous NFAs
- Equal expressive power as standard DPDA
- State increase is quadratic in the worst case with a fixed alphabet
- Allows for efficient mapping to hardware resources
  - Transitions decoupled from input/stack matches

Five Steps of DPDA Execution Per Cycle



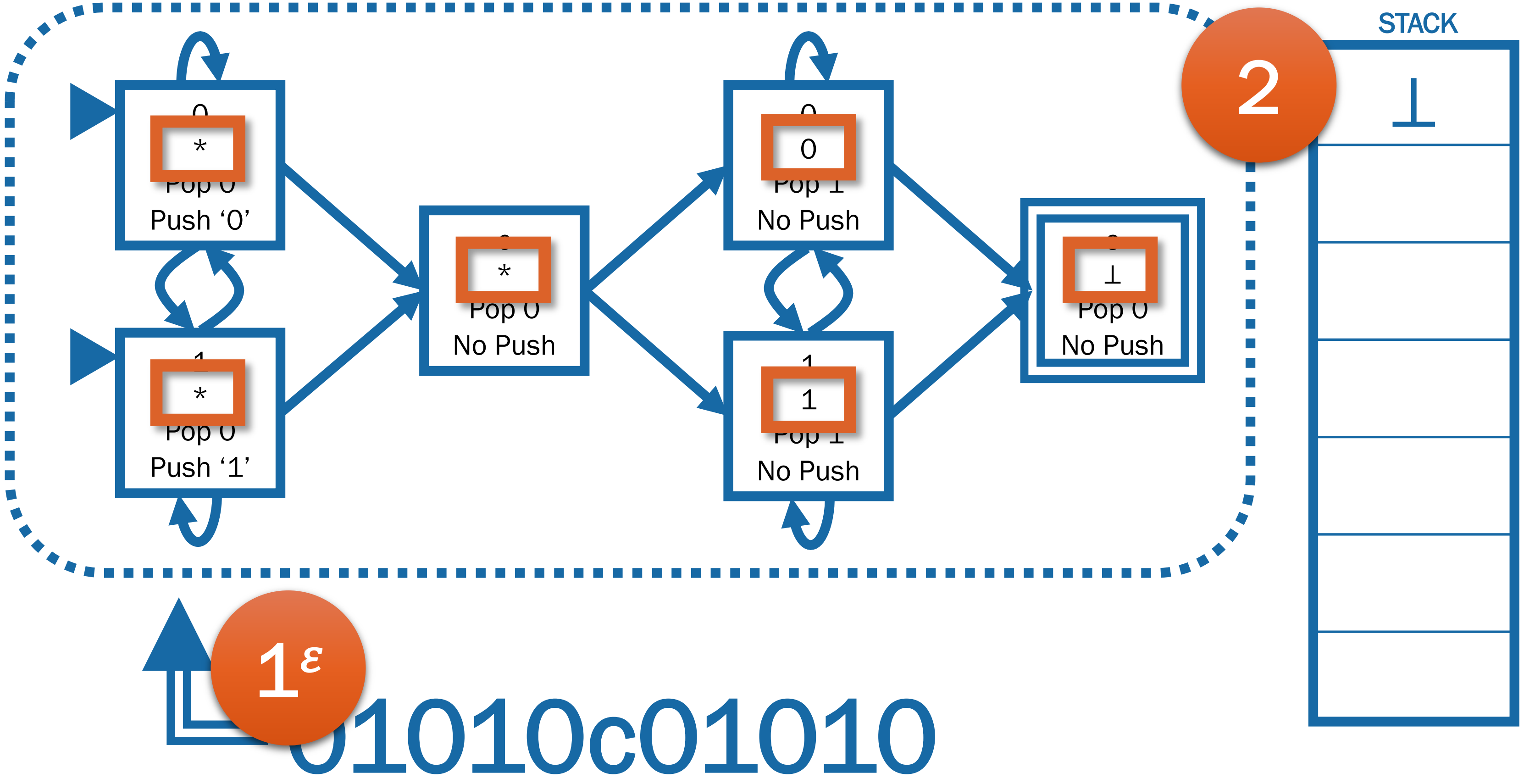
Five Steps of DPDA Execution Per Cycle

1. Input Match$^{\varepsilon}$
2. Stack Match
3. Action Lookup
4. Stack Update
5. State Transition



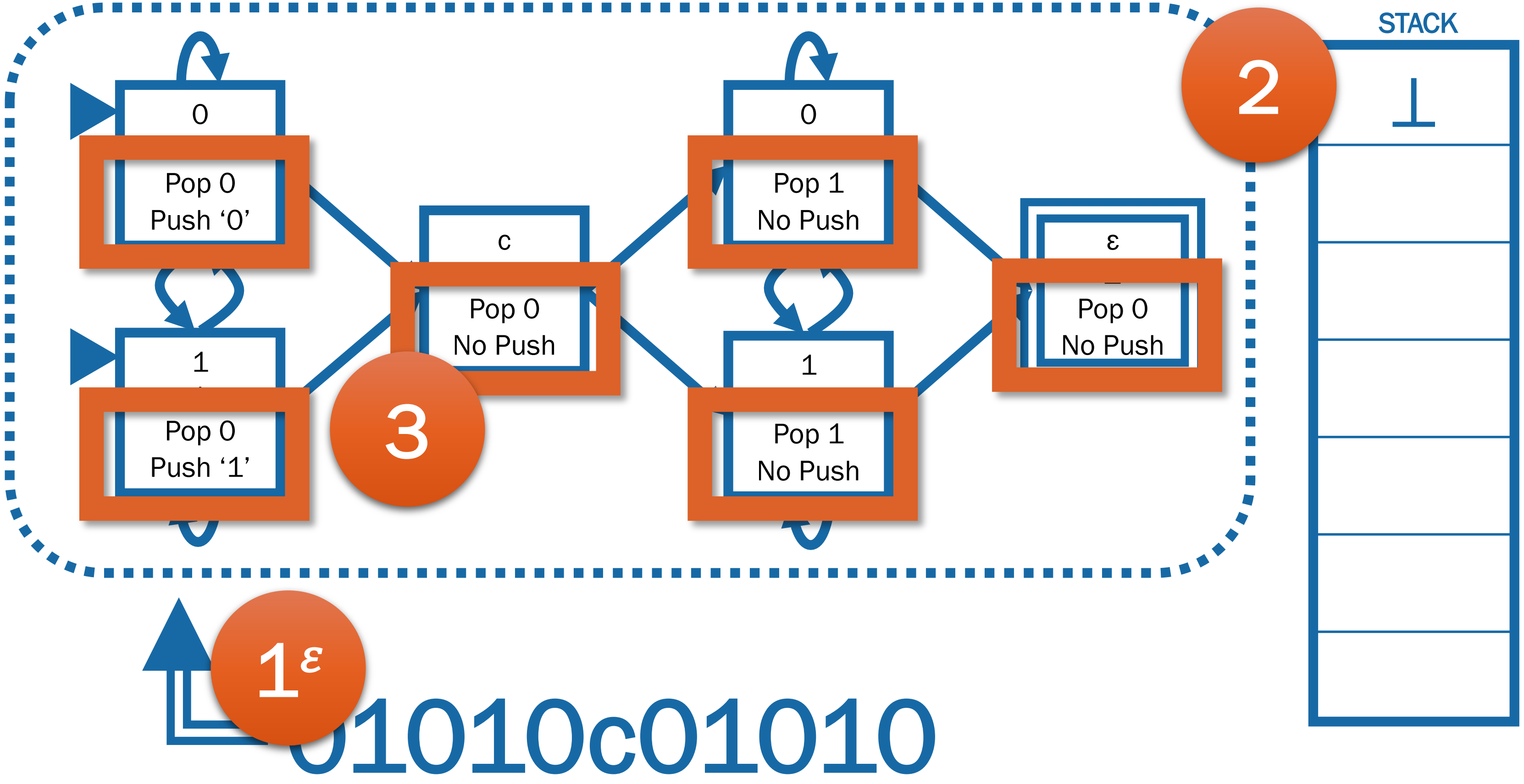
Five Steps of DPDA Execution Per Cycle

1. Input Match$^{\varepsilon}$
2. Stack Match
3. Action Lookup
4. Stack Update
5. State Transition



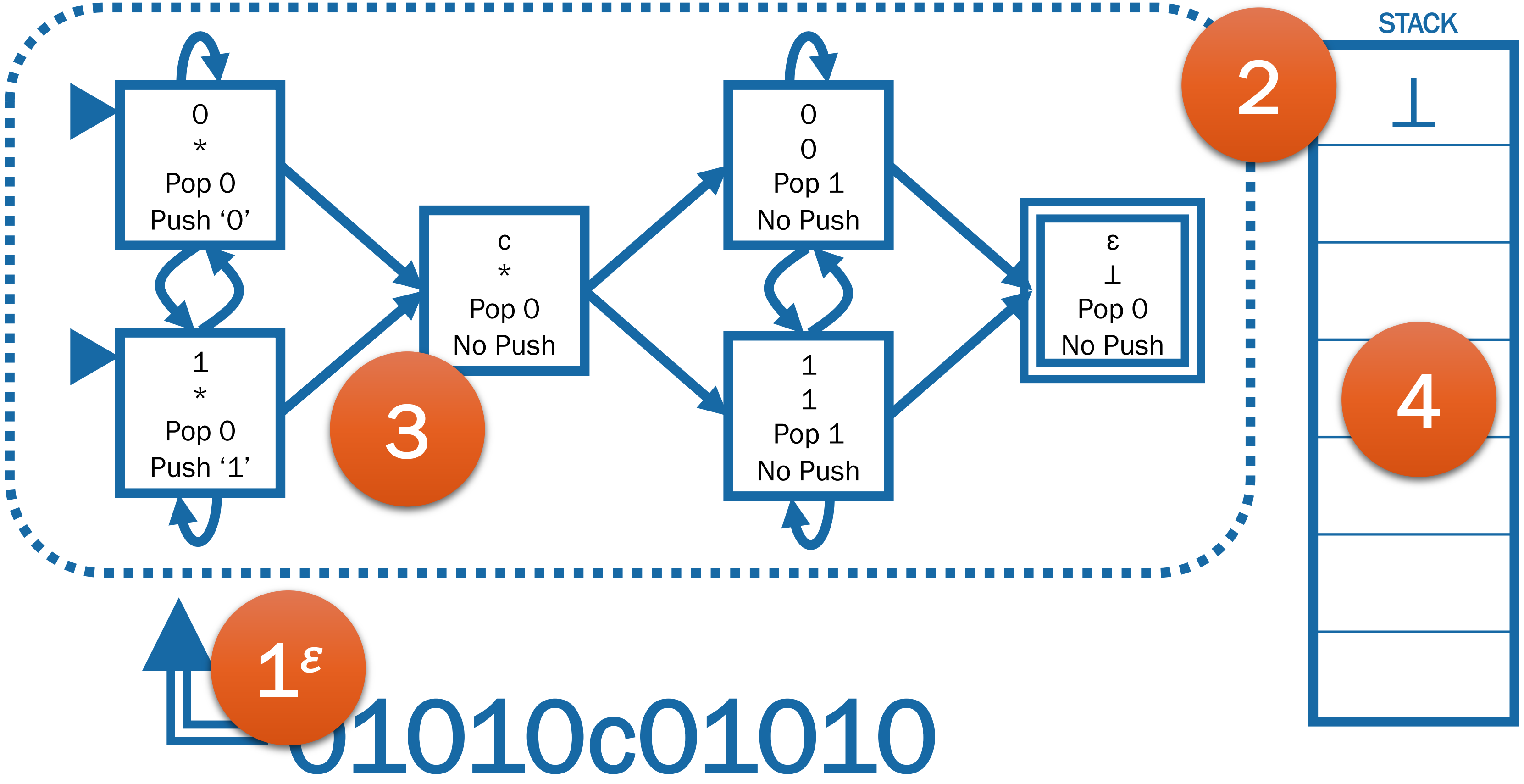
Five Steps of DPDA Execution Per Cycle

1. Input Match$^{\varepsilon}$
2. Stack Match
3. Action Lookup
4. Stack Update
5. State Transition



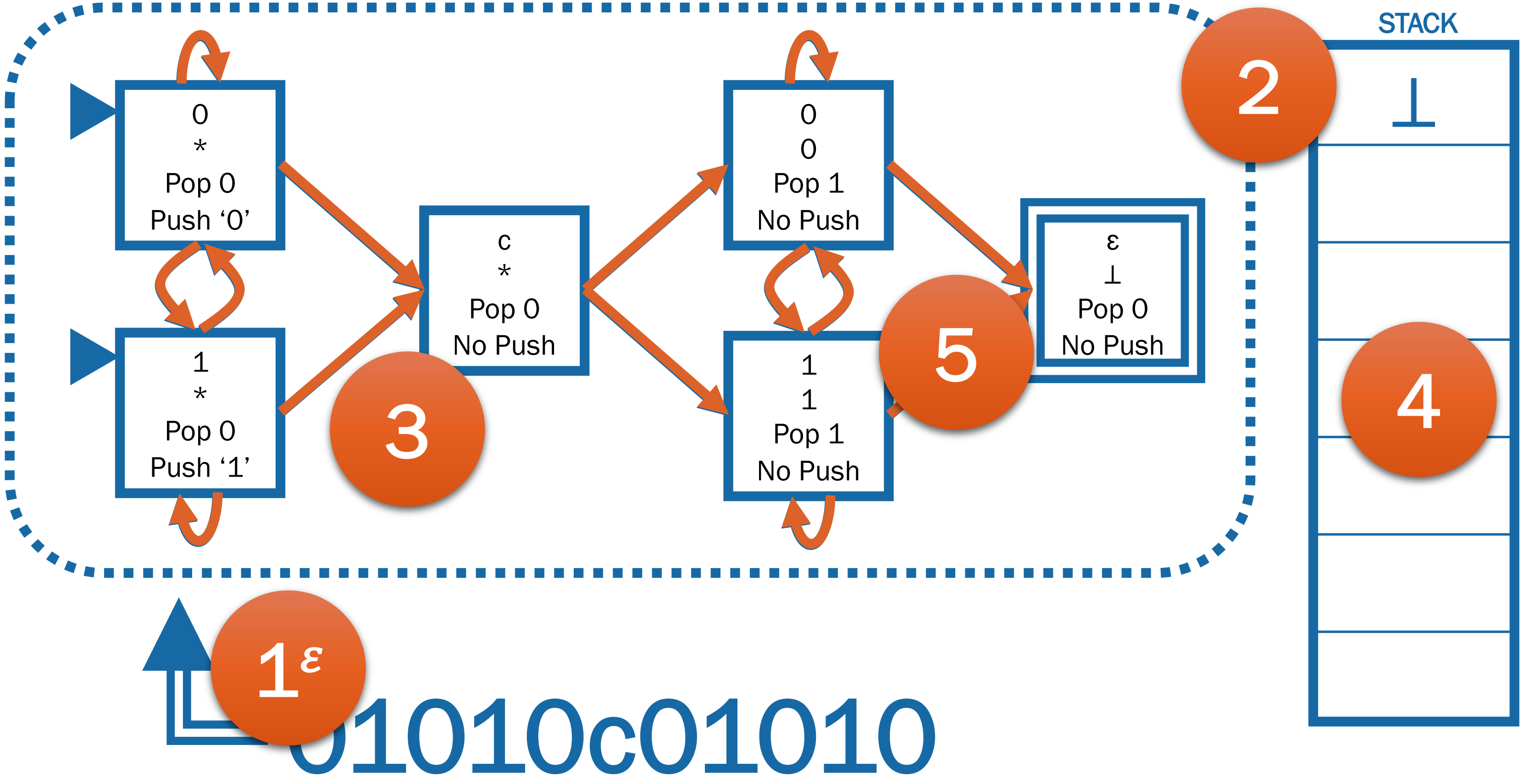
Five Steps of DPDA Execution Per Cycle

1. Input Match$^{\varepsilon}$
2. Stack Match
3. Action Lookup
4. Stack Update
5. State Transition



Five Steps of DPDA Execution Per Cycle

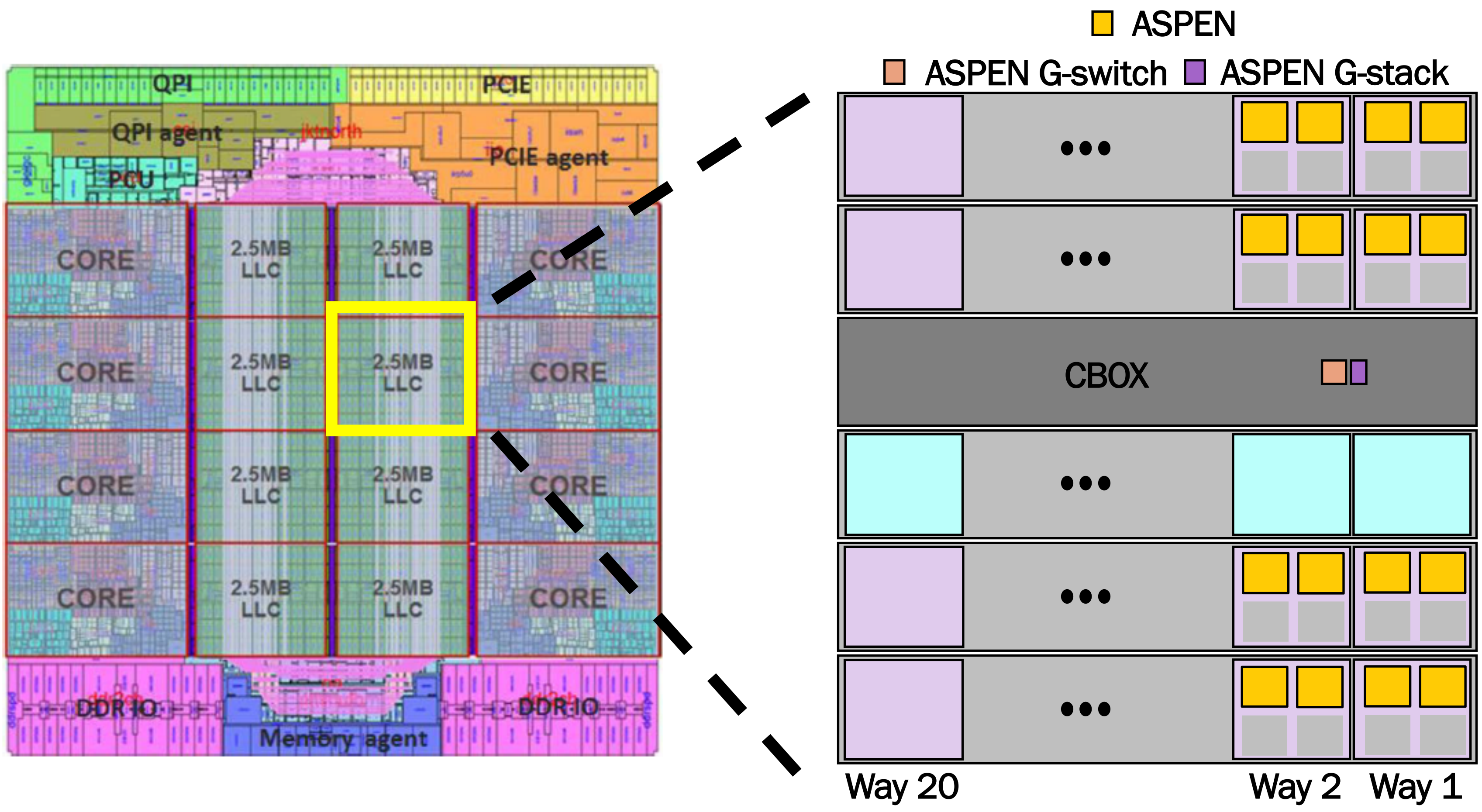
1. Input Match$^{\varepsilon}$
2. Stack Match
3. Action Lookup
4. Stack Update
5. State Transition



Implementing ASPEN in LLC

- ASPEN repurposes LLC slices for pushdown automata computation
- Location in LLC supports tighter coupling with CPU operations than dedicated accelerator
  - PDA often part of a larger workflow
  - ASPEN similar to auxiliary functional unit in CPU (similar to FPU or vector unit)
- SRAM arrays in LLC already support necessary operations for DPDA execution

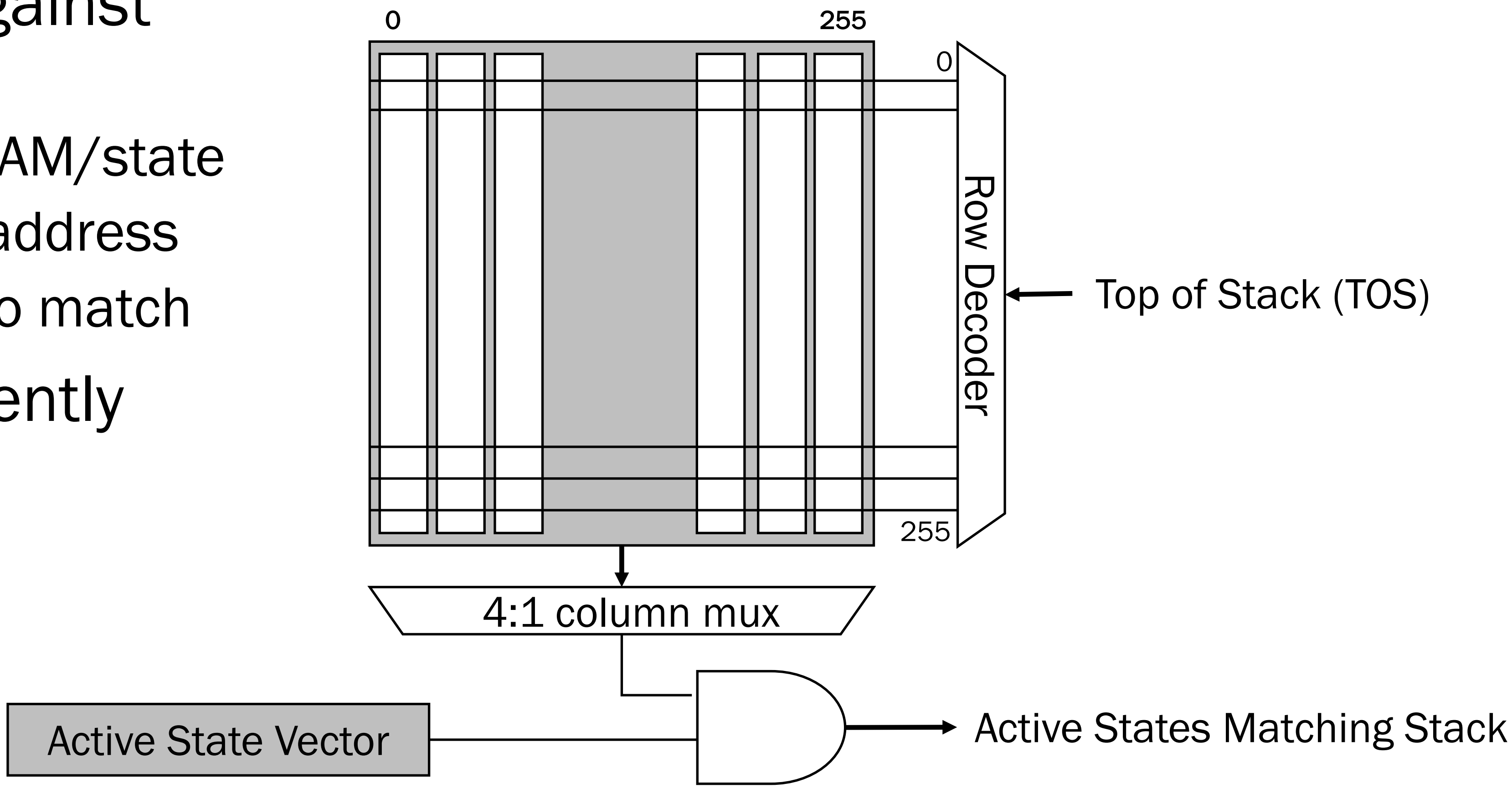
Where is ASPEN?



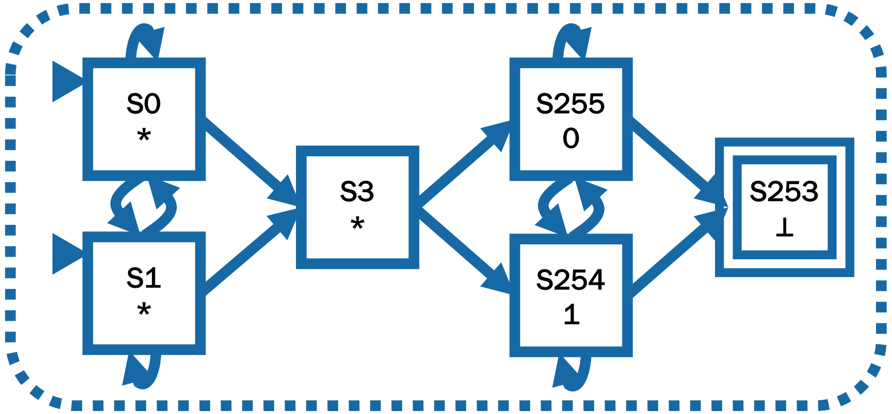
- ASPEN uses 2 arrays per bank
- 240 states per bank
- Full connectivity within bank
- Global switch and stack in CBOX for large DPDA

Stack Match in SRAM

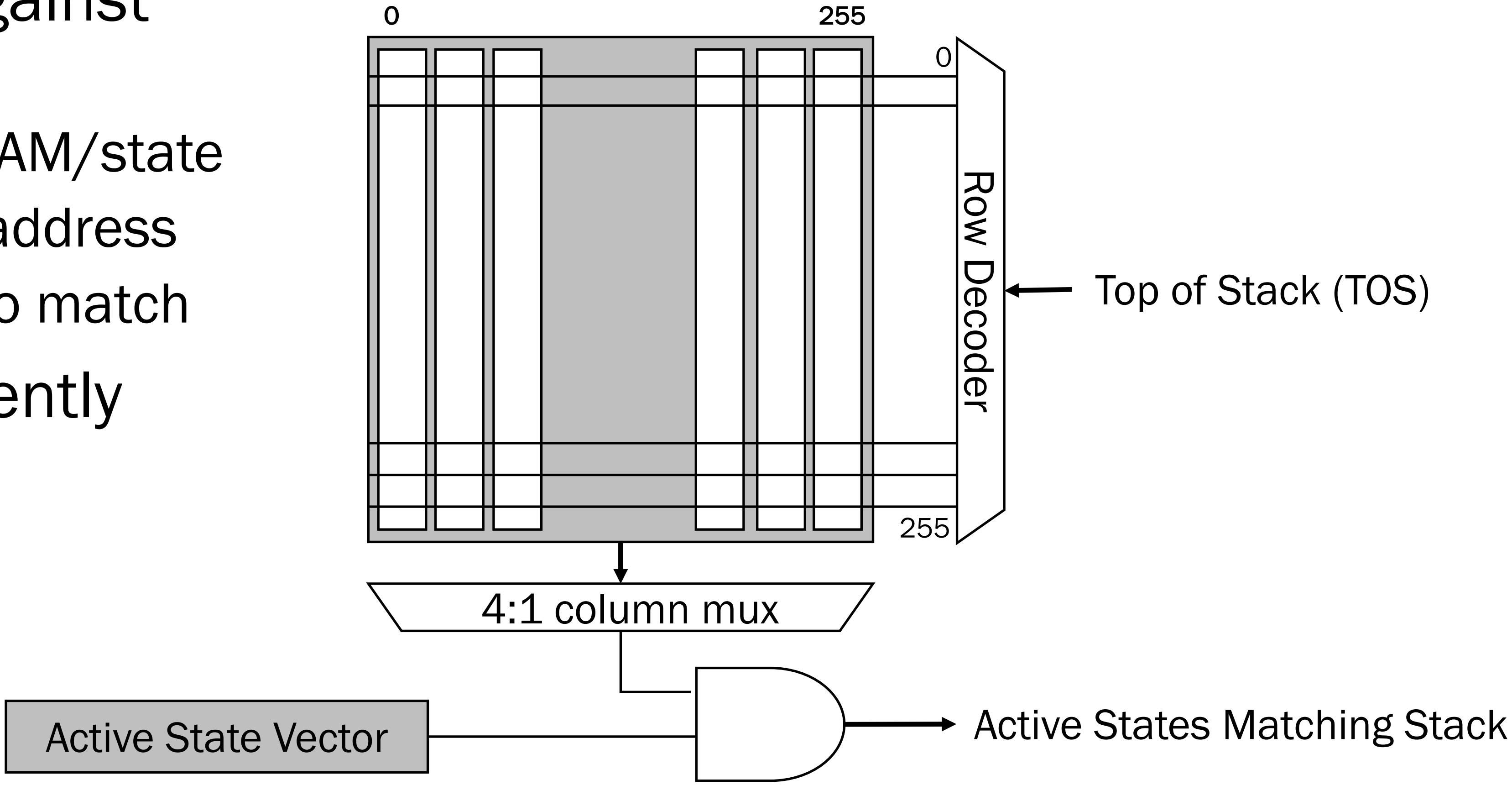
- Check all states against top of stack
  - One column of SRAM/state
  - Input TOS as row address
  - “1”: match; ”0”: no match
- Intersect with currently active states



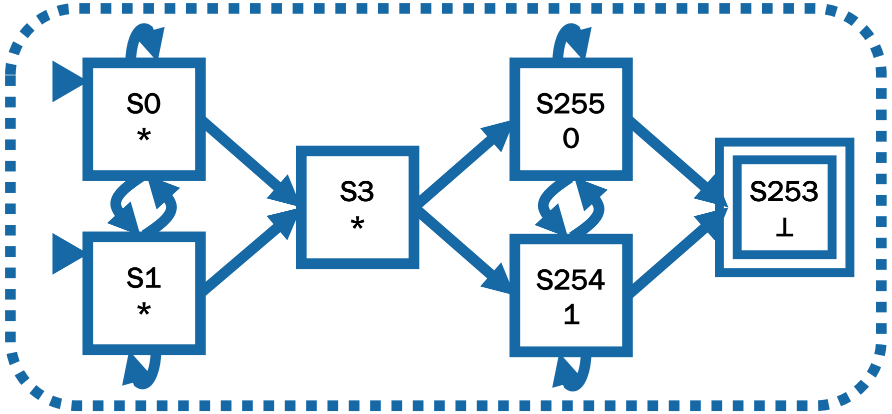
Stack Match in SRAM



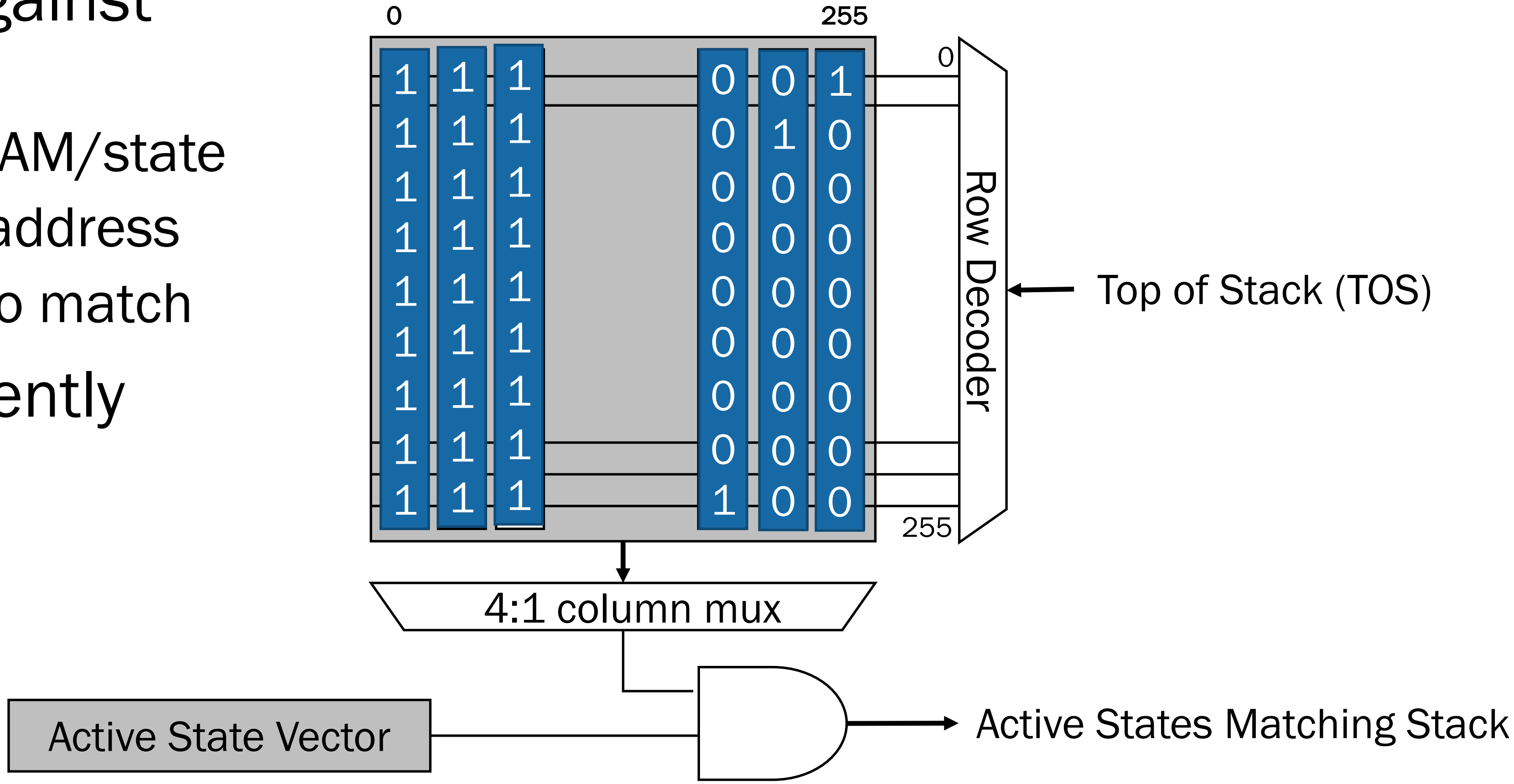
- Check **all states** against top of stack
  - One column of SRAM/state
  - Input TOS as row address
  - “1”: match; ”0”: no match
- **Intersect** with currently active states



Stack Match in SRAM



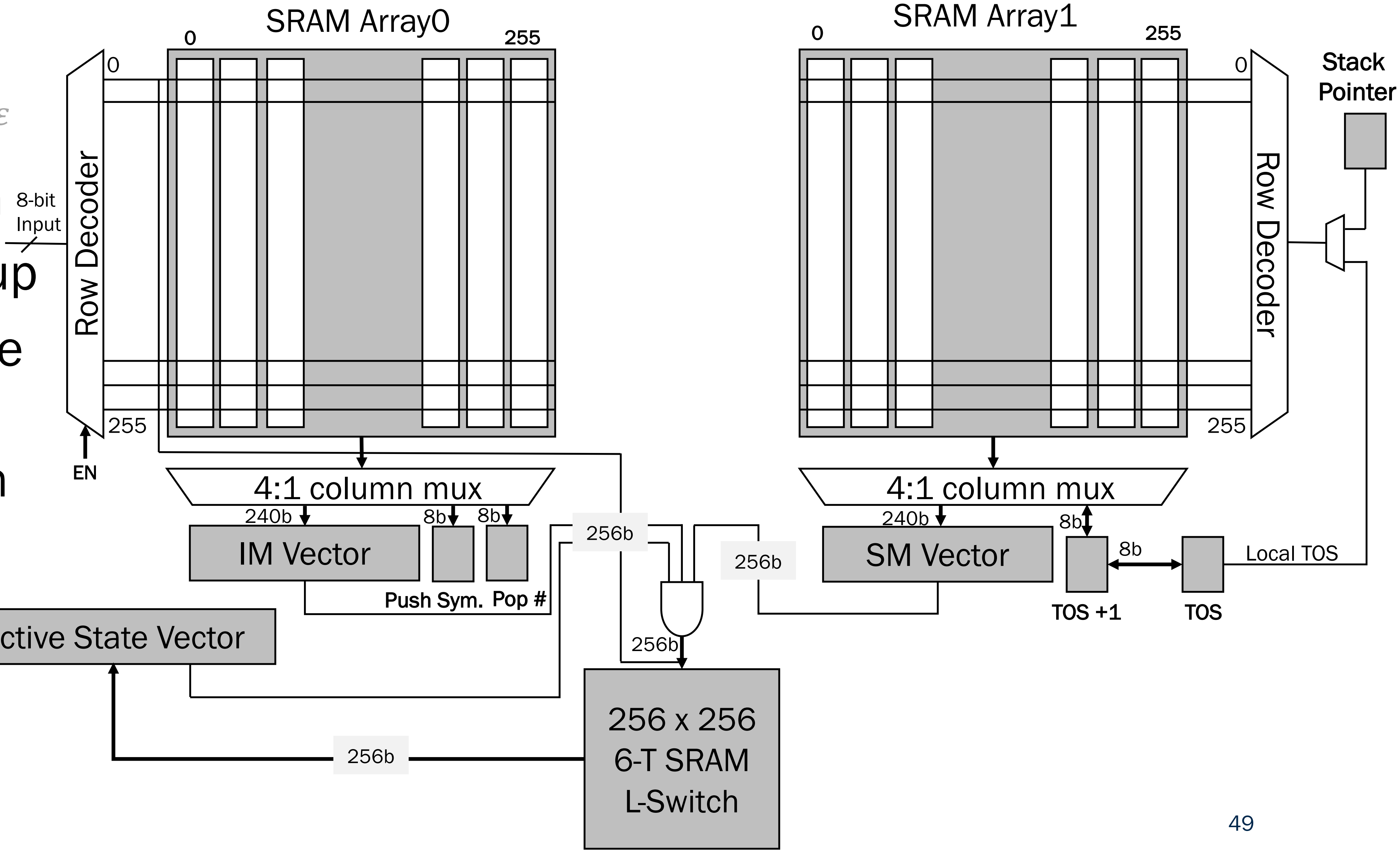
- Check **all states** against top of stack
  - One column of SRAM/state
  - Input TOS as row address
  - “1”: match; ”0”: no match
- **Intersect** with currently active states



1	1	1	0	0	1
1	1	1	0	1	0
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	1	0	0

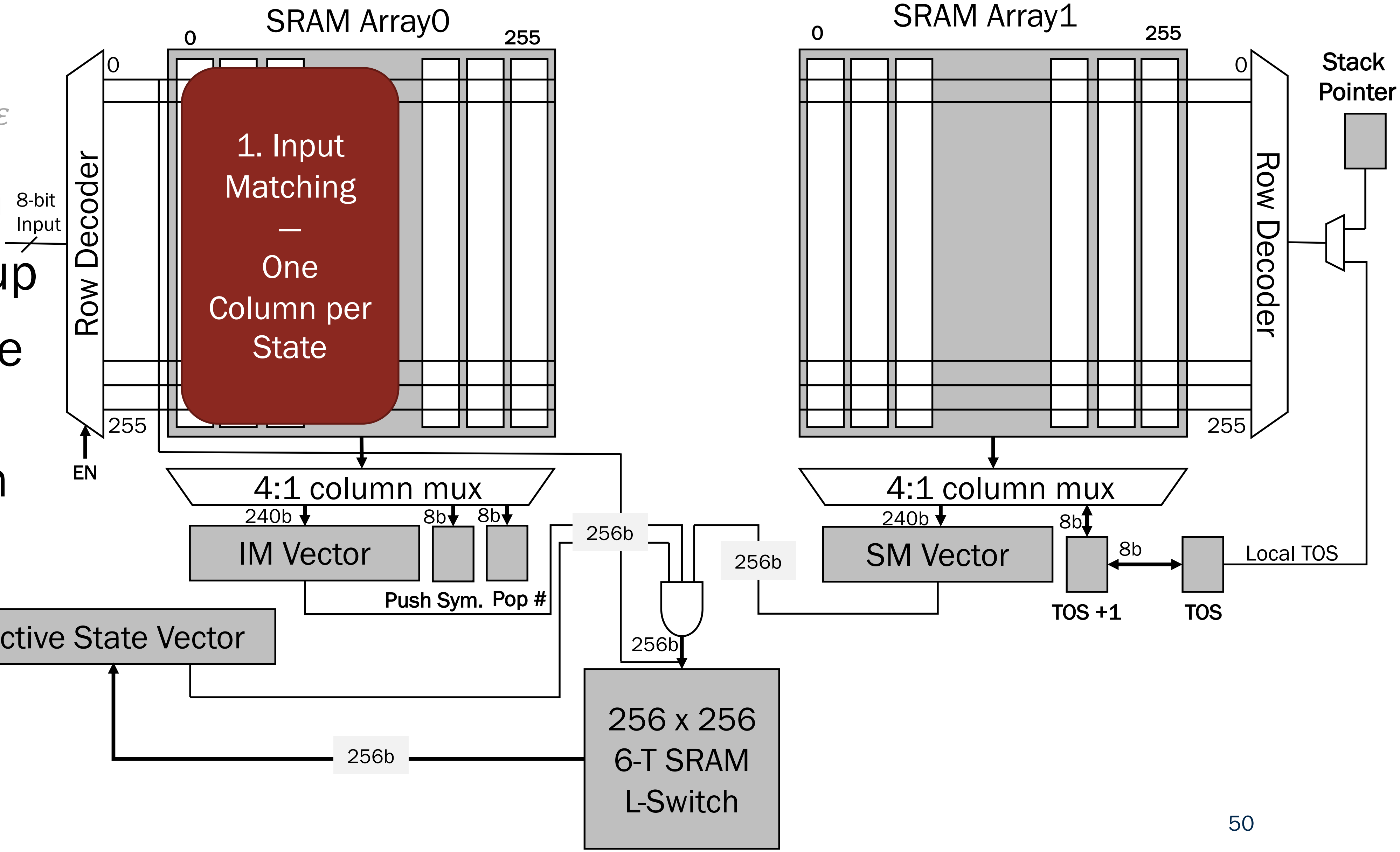
ASPEN Datapath — 240 States per Two SRAM Arrays

1. Input Match$^{\varepsilon}$
2. Stack Match
3. Action Lookup
4. Stack Update
5. State Transition



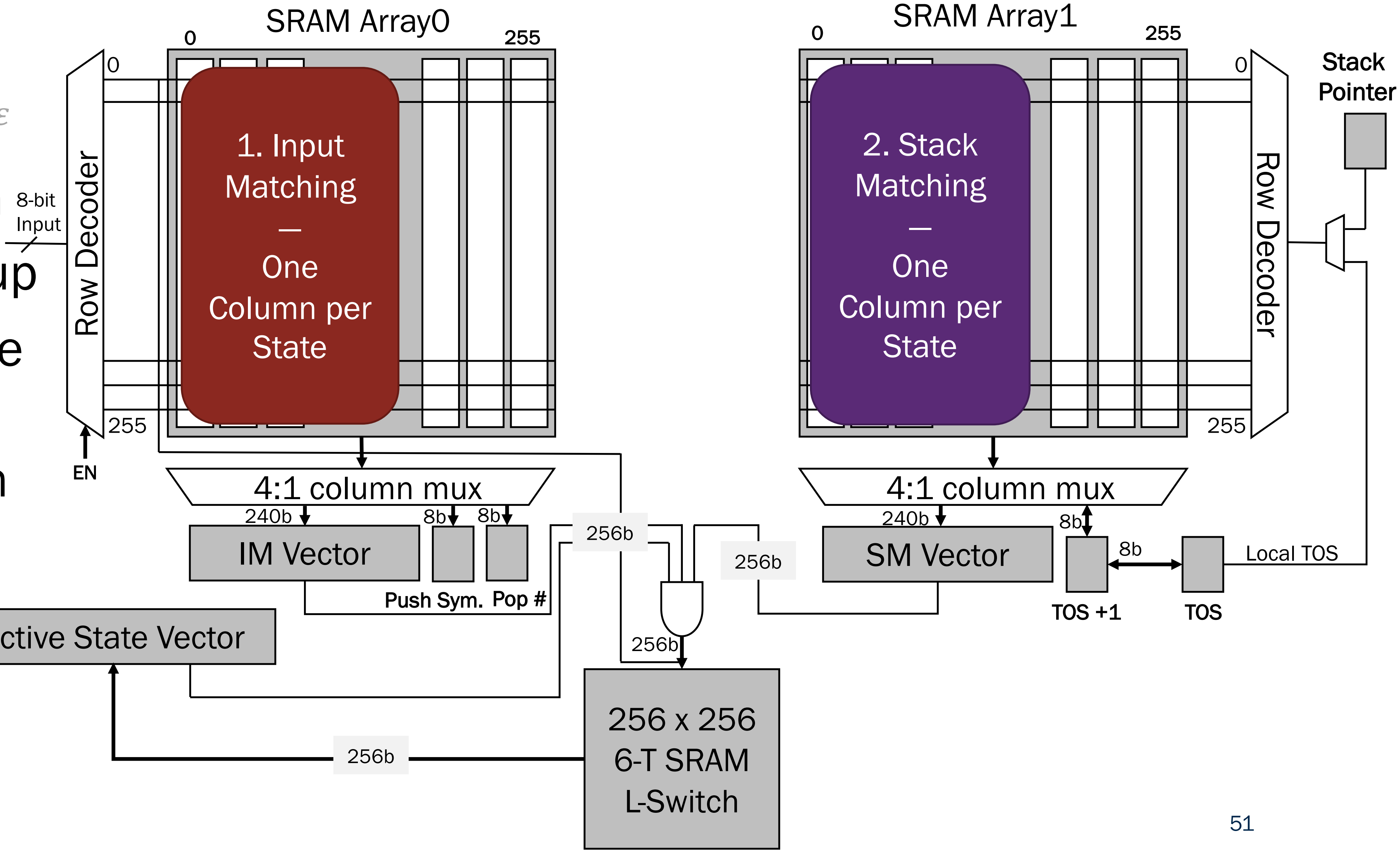
ASPEN Datapath — 240 States per Two SRAM Arrays

1. Input Match$^{\varepsilon}$
2. Stack Match
3. Action Lookup
4. Stack Update
5. State Transition



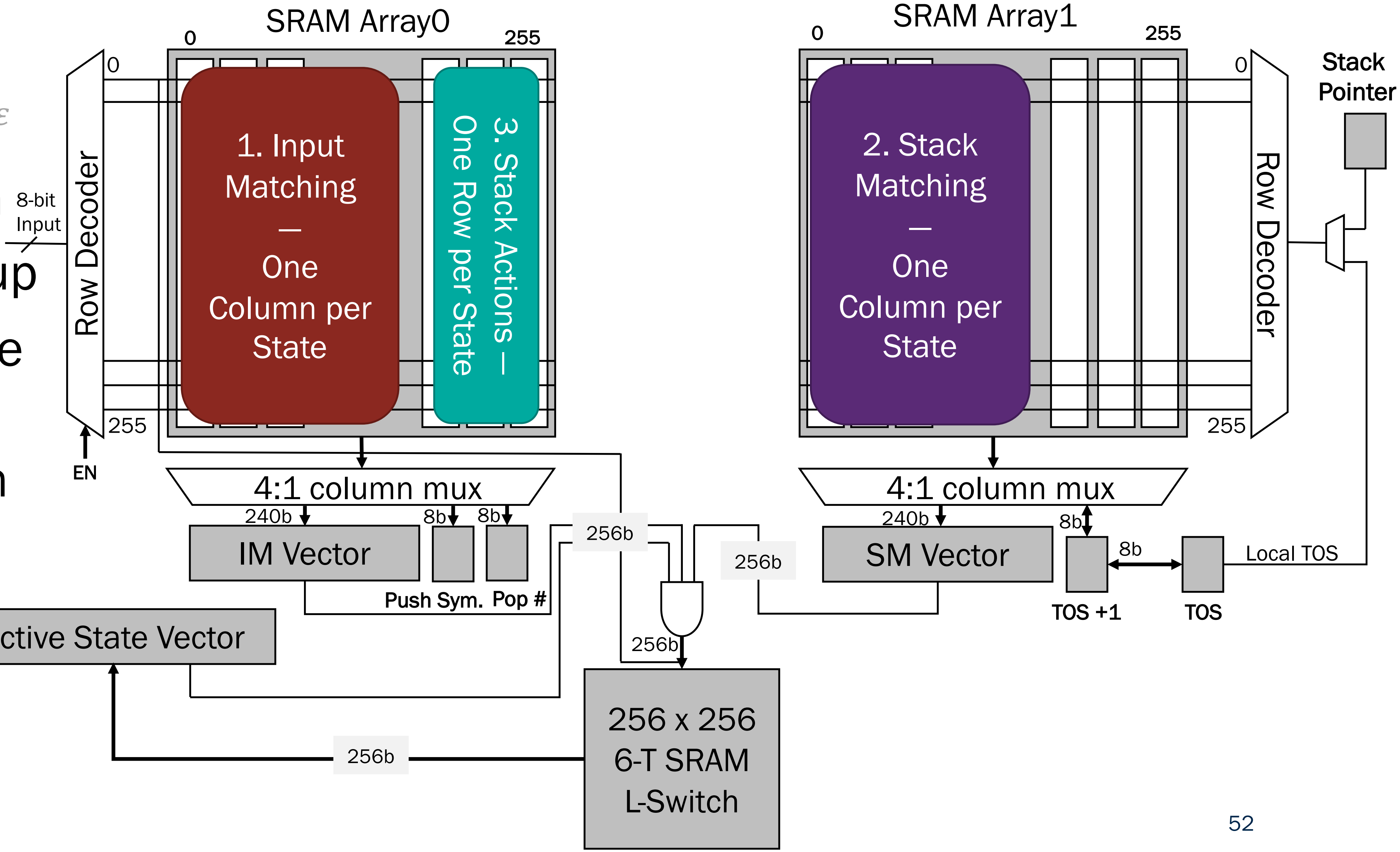
ASPEN Datapath — 240 States per Two SRAM Arrays

1. Input Match$^{\varepsilon}$
2. Stack Match
3. Action Lookup
4. Stack Update
5. State Transition



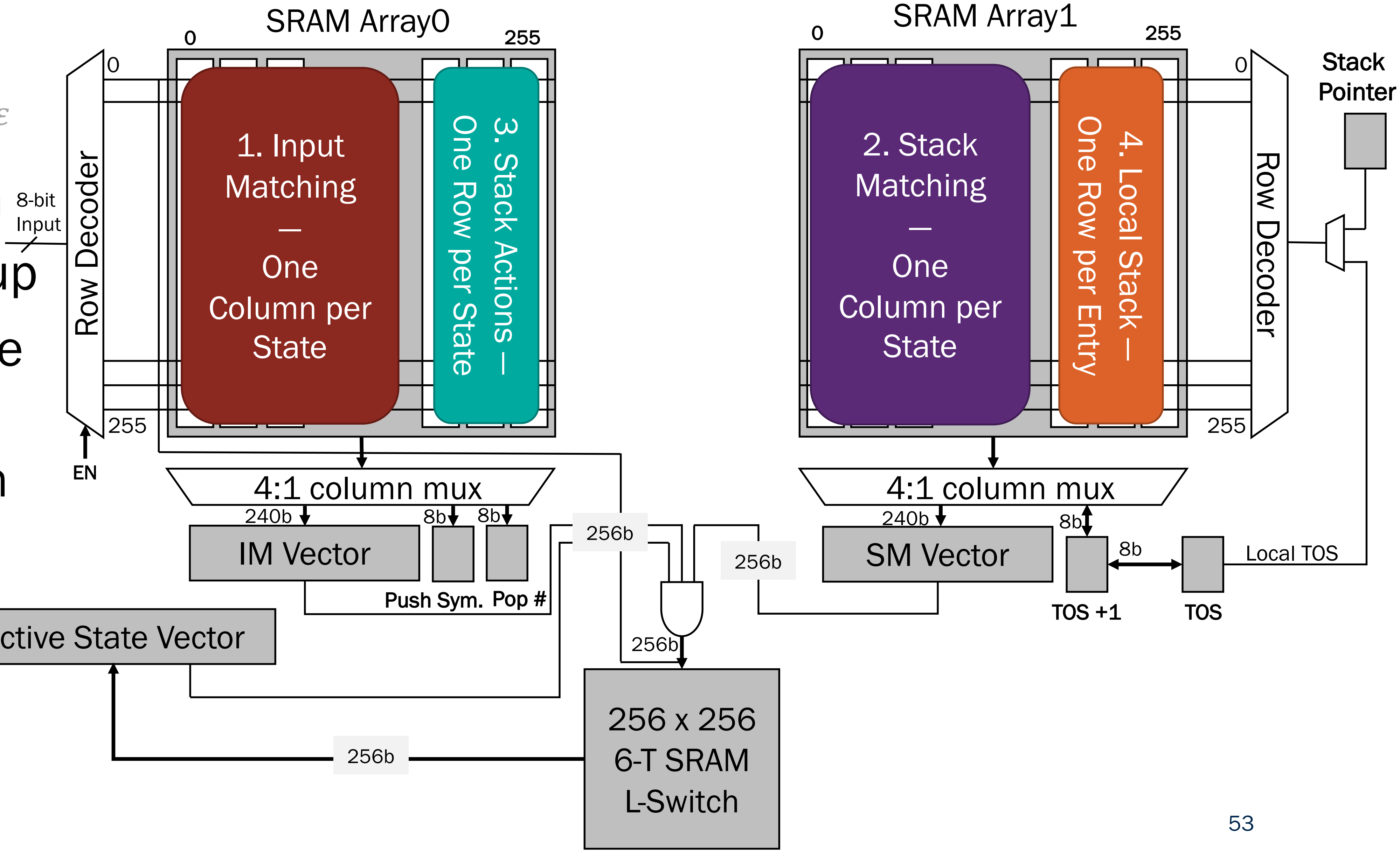
ASPEN Datapath — 240 States per Two SRAM Arrays

1. Input Match$^{\varepsilon}$
2. Stack Match
3. Action Lookup
4. Stack Update
5. State Transition



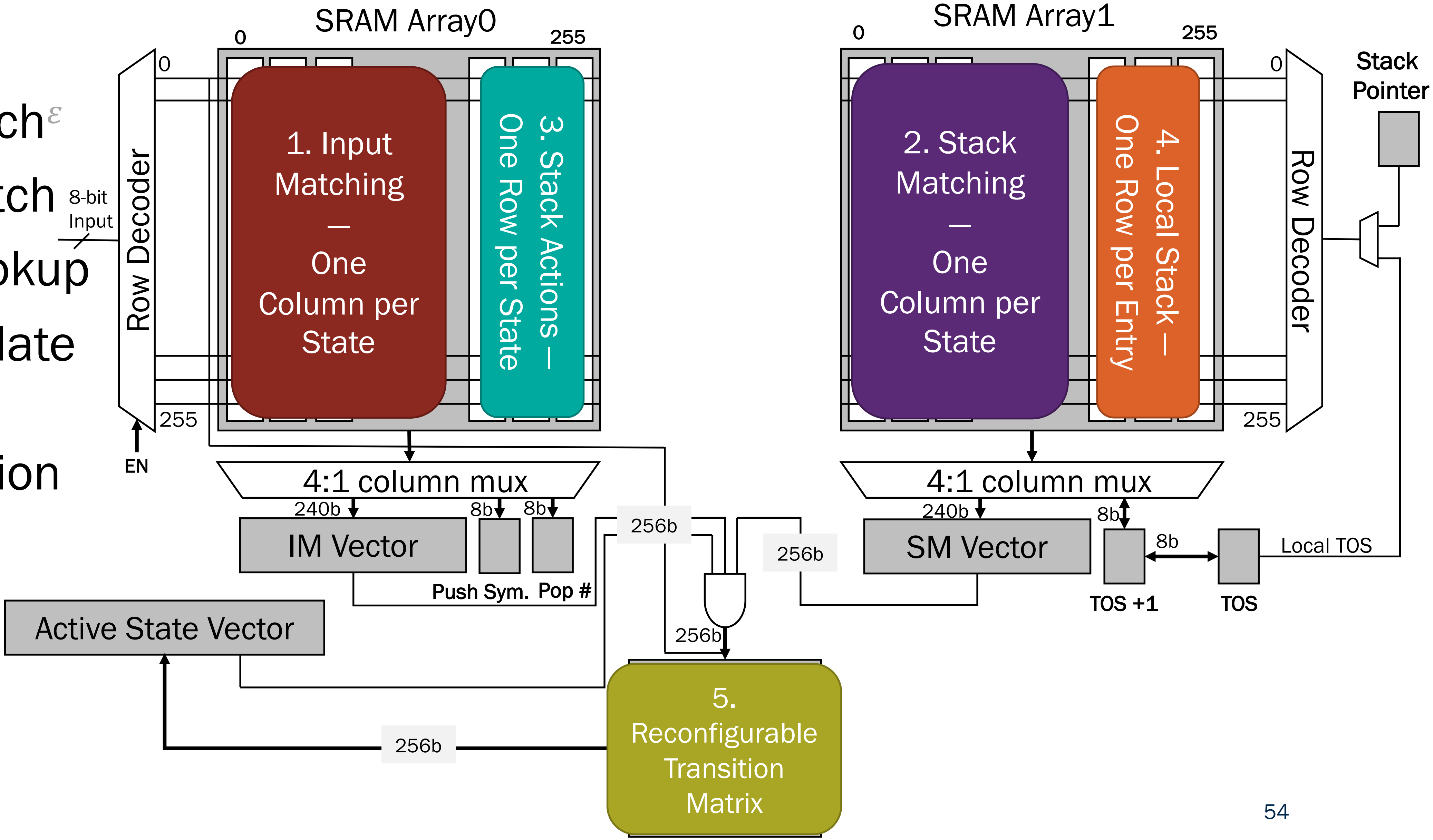
ASPEN Datapath — 240 States per Two SRAM Arrays

1. Input Match$^{\varepsilon}$
2. Stack Match
3. Action Lookup
4. Stack Update
5. State Transition



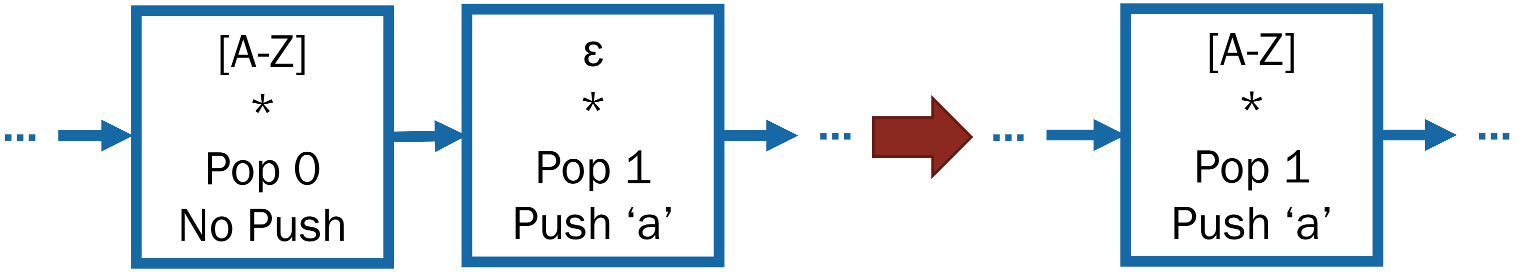
ASPEN Datapath — 240 States per Two SRAM Arrays

1. Input Match$^{\varepsilon}$
2. Stack Match
3. Action Lookup
4. Stack Update
5. State Transition



Optimizations

Epsilon Merging

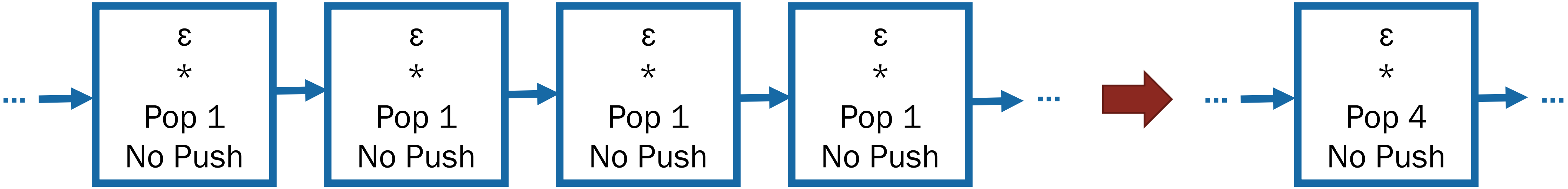


... [A-Z] * Pop 0 No Push → ε * Pop 1 Push ‘a’ → ...

... [A-Z] * Pop 1 Push ‘a’ → ...

Goal: Reduce the number of stalls while processing input

Multipop



... ε * Pop 1 No Push → ε * Pop 1 No Push → ε * Pop 1 No Push → ε * Pop 1 No Push → ...

... ε * Pop 4 No Push → ...

- Average of 65% reduction in epsilon states

Evaluation: Two Real-World Applications

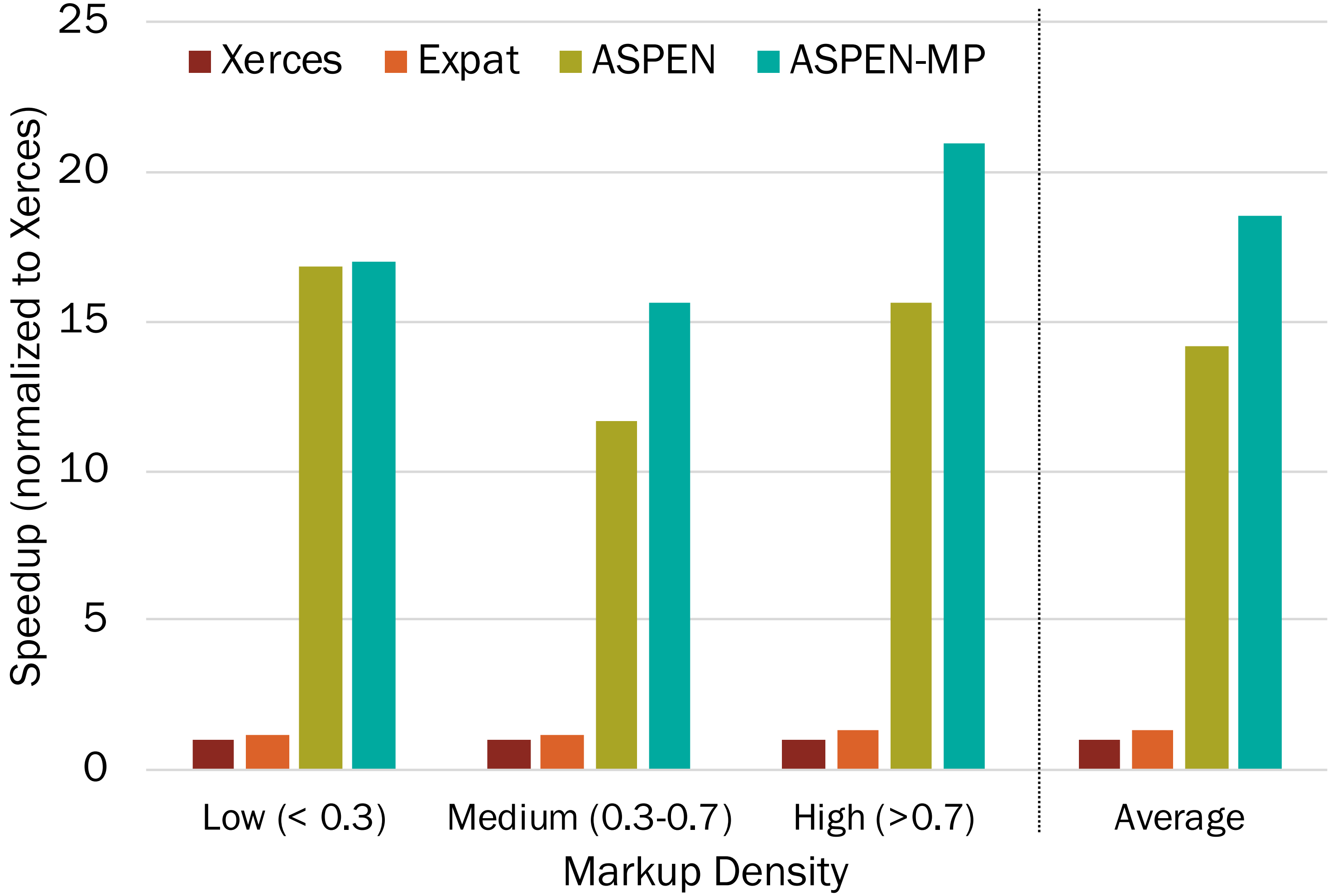
- XML Parsing
  - Common to many data analyses (needed to read input data)
  - Step in a larger pipeline: Tokenization, Parsing, Validation, DOM construction
  - Pipelined with Cache Automaton for tokenization
  - Single Large DPDA with Global Stack
- Frequent Subtree Mining
  - Task of identifying subtrees occurring above a threshold frequency in a corpus of trees
  - Common in recommendation systems, packet routing, NLP, etc.
  - Many Small DPDA with Local Stacks

Experimental Methodology

Component	Max Frequency	Operating Frequency
ASPEN	880 MHz	850 MHz
Cache Automaton	4 GHz	3.4 GHz

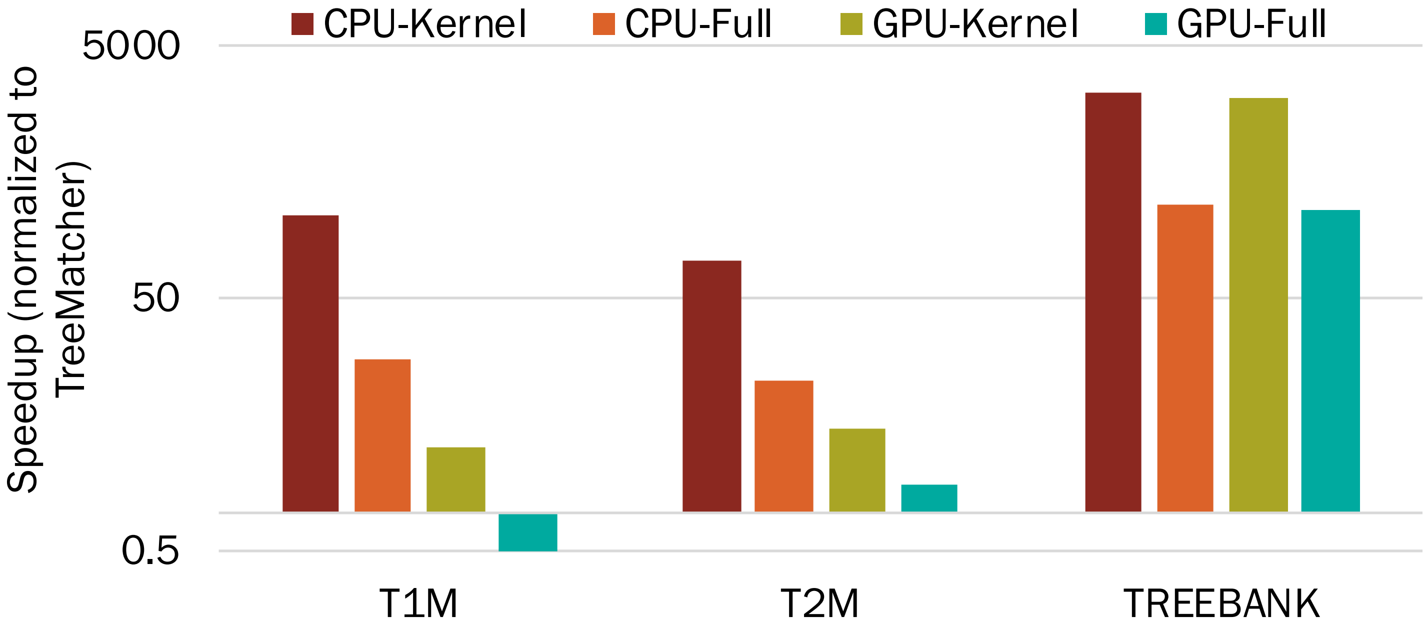
- **Baseline Evaluation**
  - **CPU:** 2.6 GHz dual-socket Intel Xeon E5-2697-v3 (28 cores total)
  - **GPU:** NVIDIA TITAN Xp
- **Performance and Power:** PAPI, Intel RAPL, NVIDIA *nvprof*
- **ASPEN Simulation:**
  - METIS graph partitioning framework
  - VASim modified for cycle-accurate DPDA simulation

XML Parsing: Parabix, Ximpleware, UW XML



- ASPEN is **13-18x faster** (on average) than popular **CPU** Parsers
- Performance did not vary significantly with complexity of XML
- Optimizations and tokenization hide *ε*-stalls

Frequent Subtree Mining

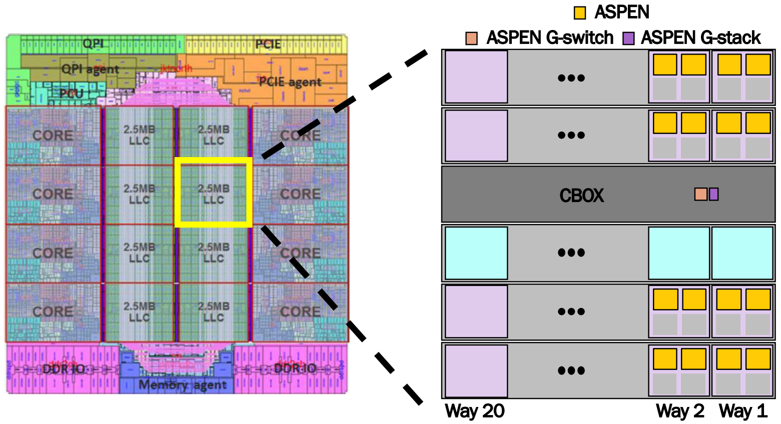


Dataset	Automata Alphabets	Stack Alphabets	Stack Size
T1M	16	17	29
T2M	38	39	49
TREEBANK	100	101	110

- ASPEN is (on average) 67x faster than CPUs 6x faster than GPUs for end-to-end application
- Performance on ASPEN is independent from tree size and complexity
- No ε-transitions

Conclusion

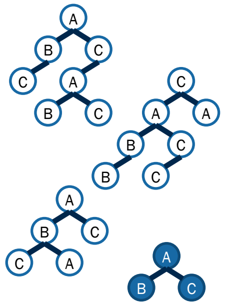
Conclusion



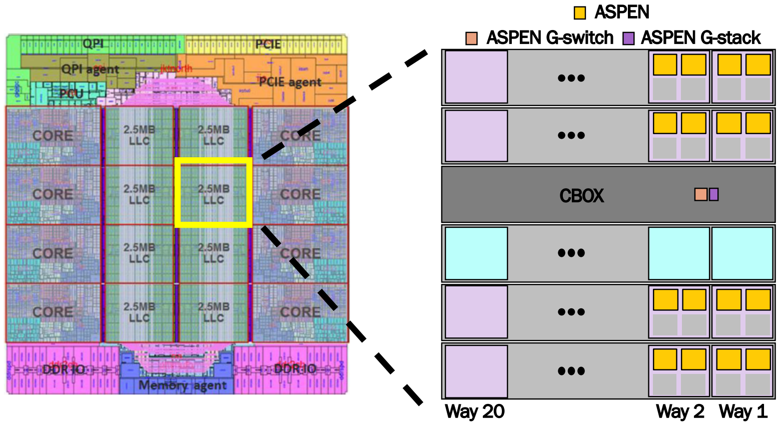
ASPEN: Processor for DPDA Acceleration

Conclusion

S → *Exp* ⊣
Exp → *Term* + *Exp*
| *Term*
Term → `int` * *Term*
| (*Exp*)
| `int`



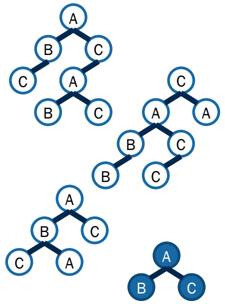
Supports Processing of
Recursively-Nested and Tree-Structured Data



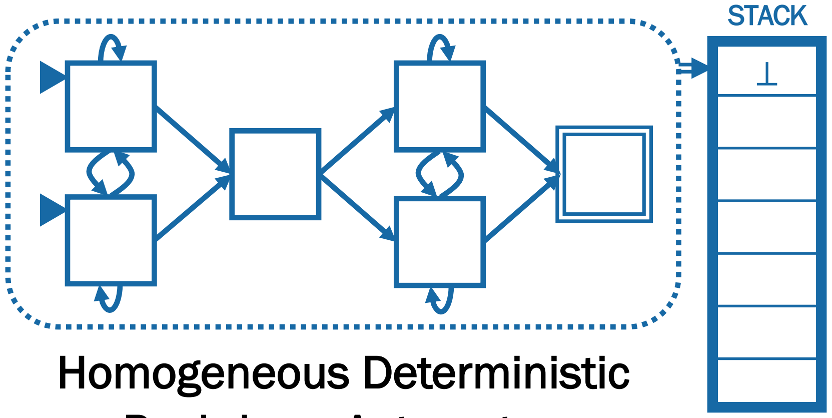
ASPEN: Processor for DPDA Acceleration

Conclusion

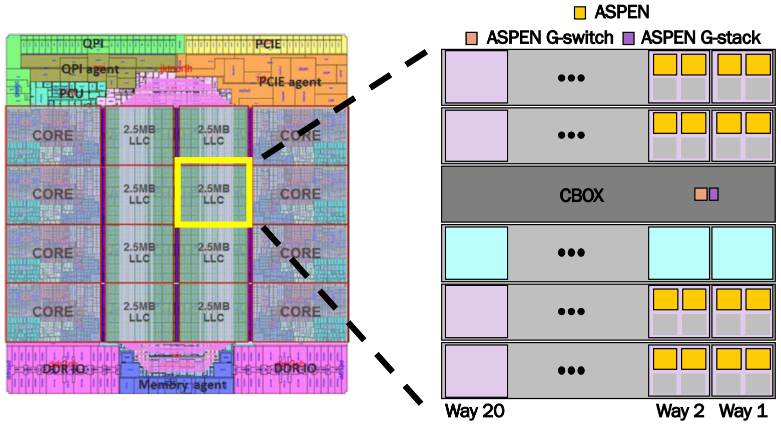
S ⟶ *Exp* ⊣
Exp ⟶ *Term* + *Exp*
| *Term*
Term ⟶ `int` * *Term*
| (*Exp*)
| `int`



Supports Processing of
Recursively-Nested and Tree-Structured Data



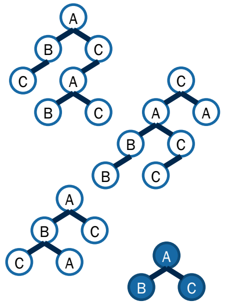
Homogeneous Deterministic
Pushdown Automaton



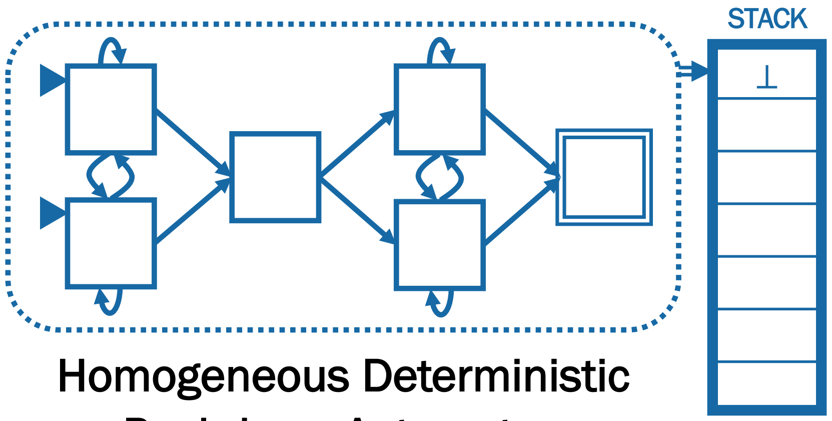
ASPEN: Processor for DPDA Acceleration

Conclusion

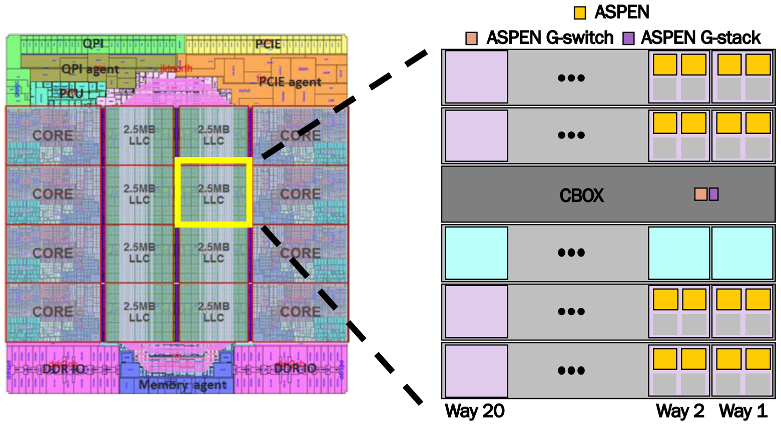
S ⟶ *Exp* ⊣
Exp ⟶ *Term* + *Exp*
| *Term*
Term ⟶ `int` * *Term*
| (*Exp*)
| `int`



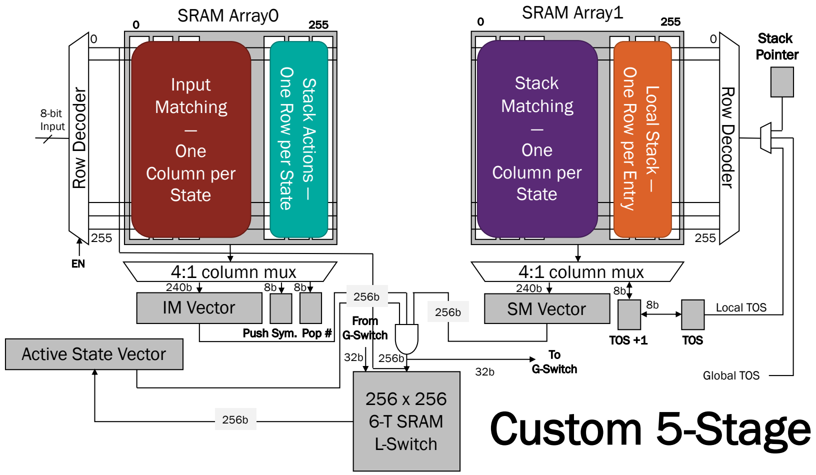
Supports Processing of
Recursively-Nested and Tree-Structured Data



Homogeneous Deterministic
Pushdown Automaton



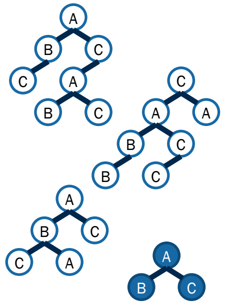
ASPEN: Processor for DPDA Acceleration



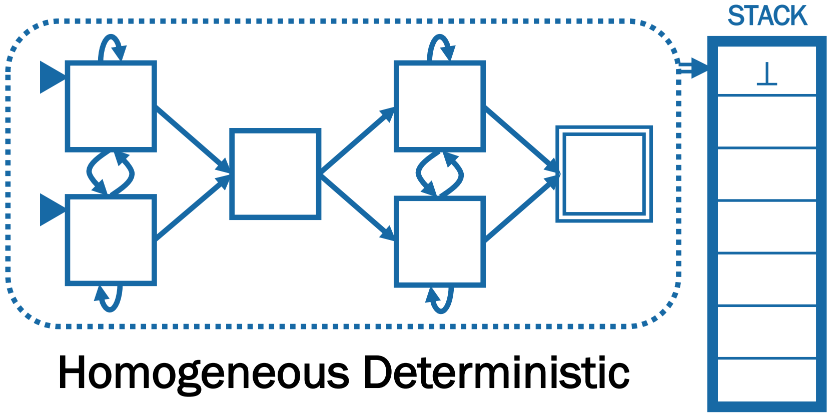
Custom 5-Stage Datapath

Conclusion

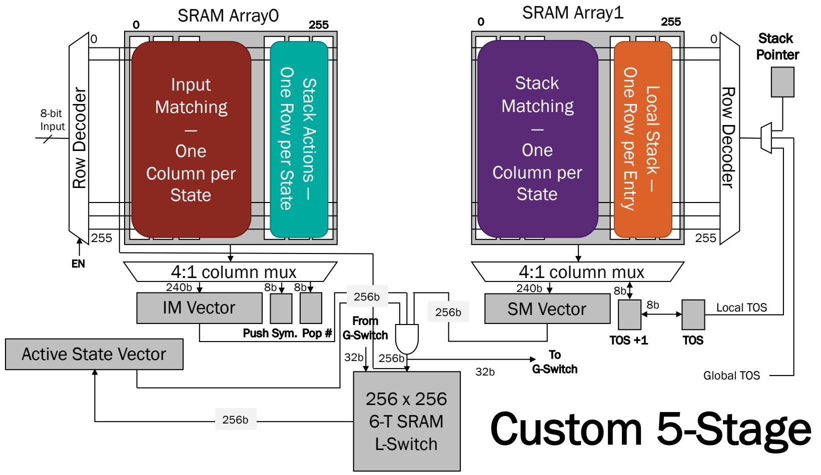
S → *Exp* ⊣
Exp → *Term* + *Exp*
| *Term*
Term → `int` * *Term*
| (*Exp*)
| `int`



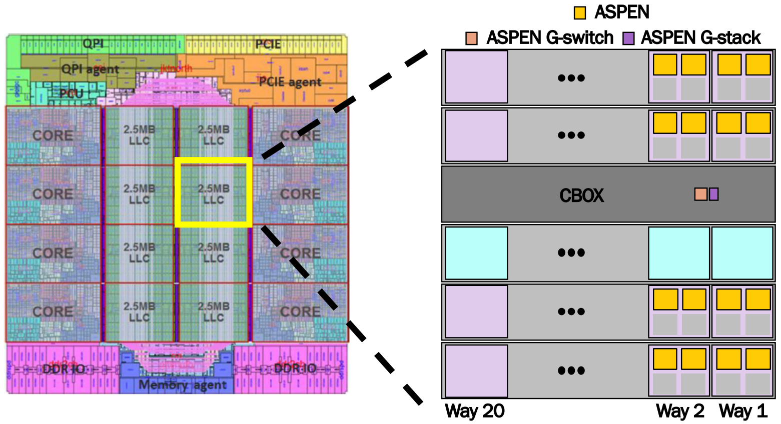
Supports Processing of
Recursively-Nested and Tree-Structured Data



Homogeneous Deterministic
Pushdown Automaton



Custom 5-Stage Datapath



ASPEN: Processor for DPDA Acceleration