

# Improving Programming Support for Hardware Accelerators Through Automata Processing Abstractions

---

**Kevin Angstadt**

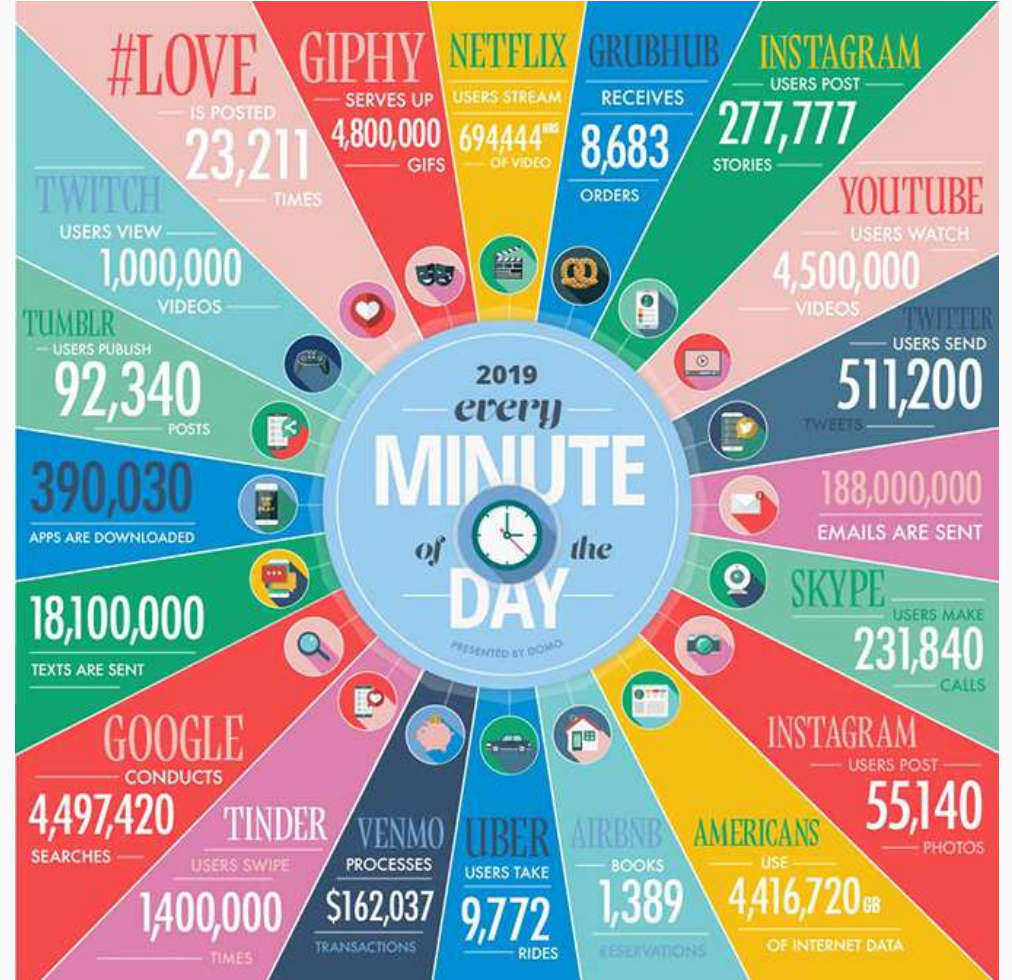
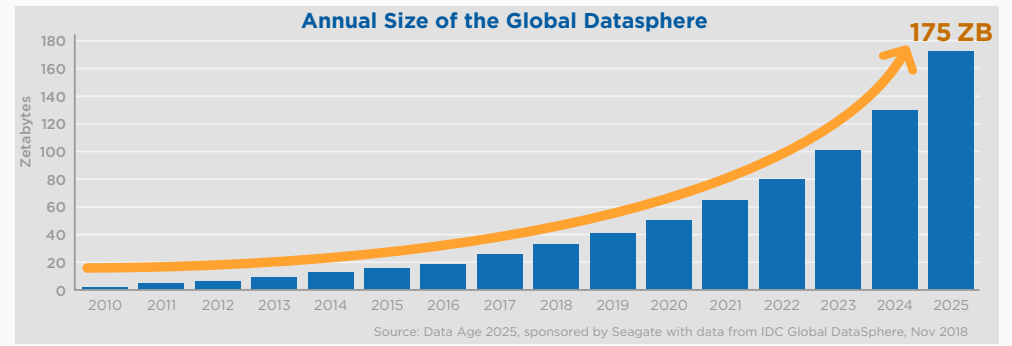
**[angstadt@umich.edu](mailto:angstadt@umich.edu)**

10. March 2020



**COMPUTER SCIENCE  
& ENGINEERING**

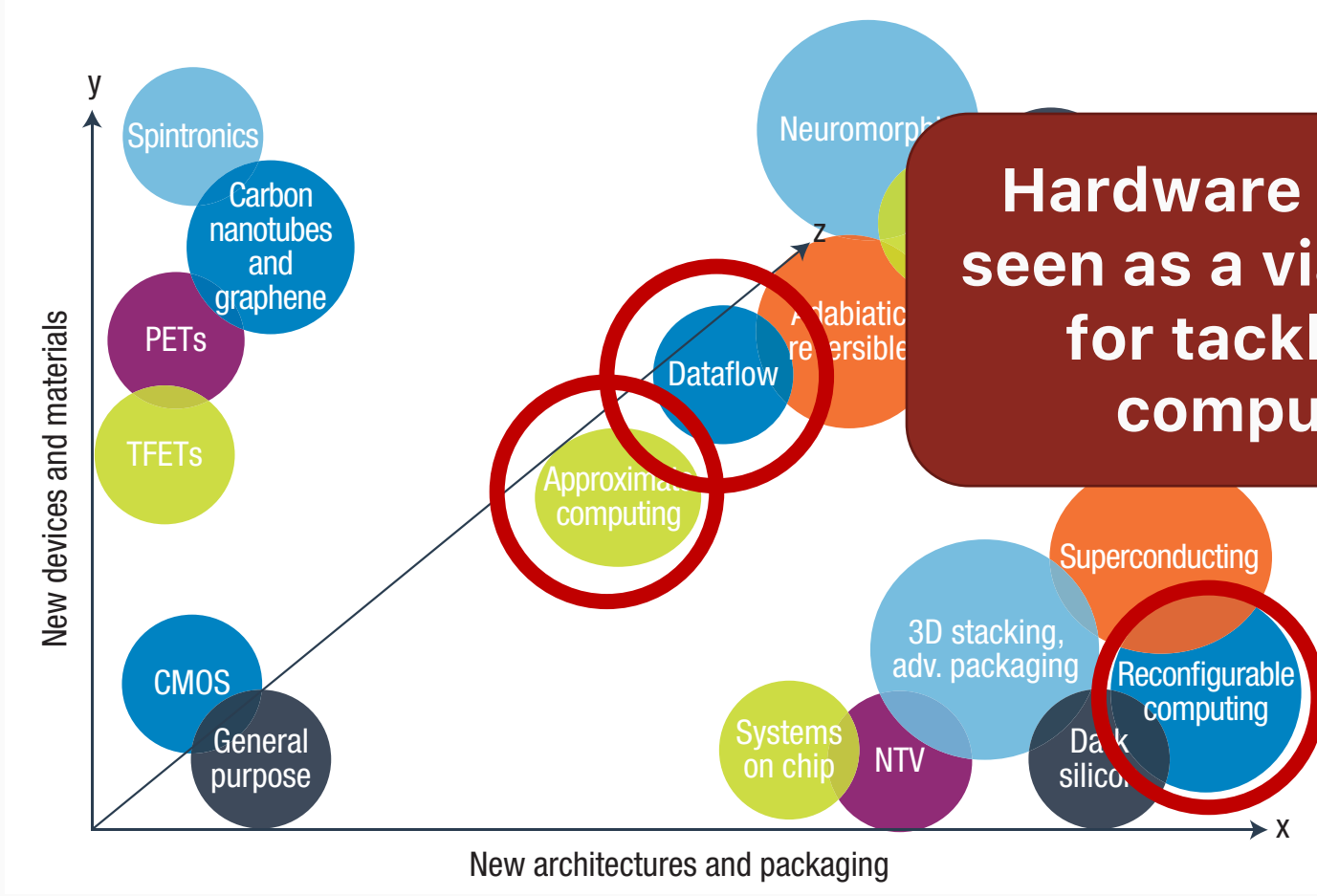
UNIVERSITY OF MICHIGAN



By 2020, there will be 40x more bytes of data than there are stars in the observable universe.

DOMO, "Data Never Sleeps 7.0". 2019

# Physical Limits Spark Creativity



**Hardware accelerators are seen as a viable path forward for tackling increasing compute demands.**

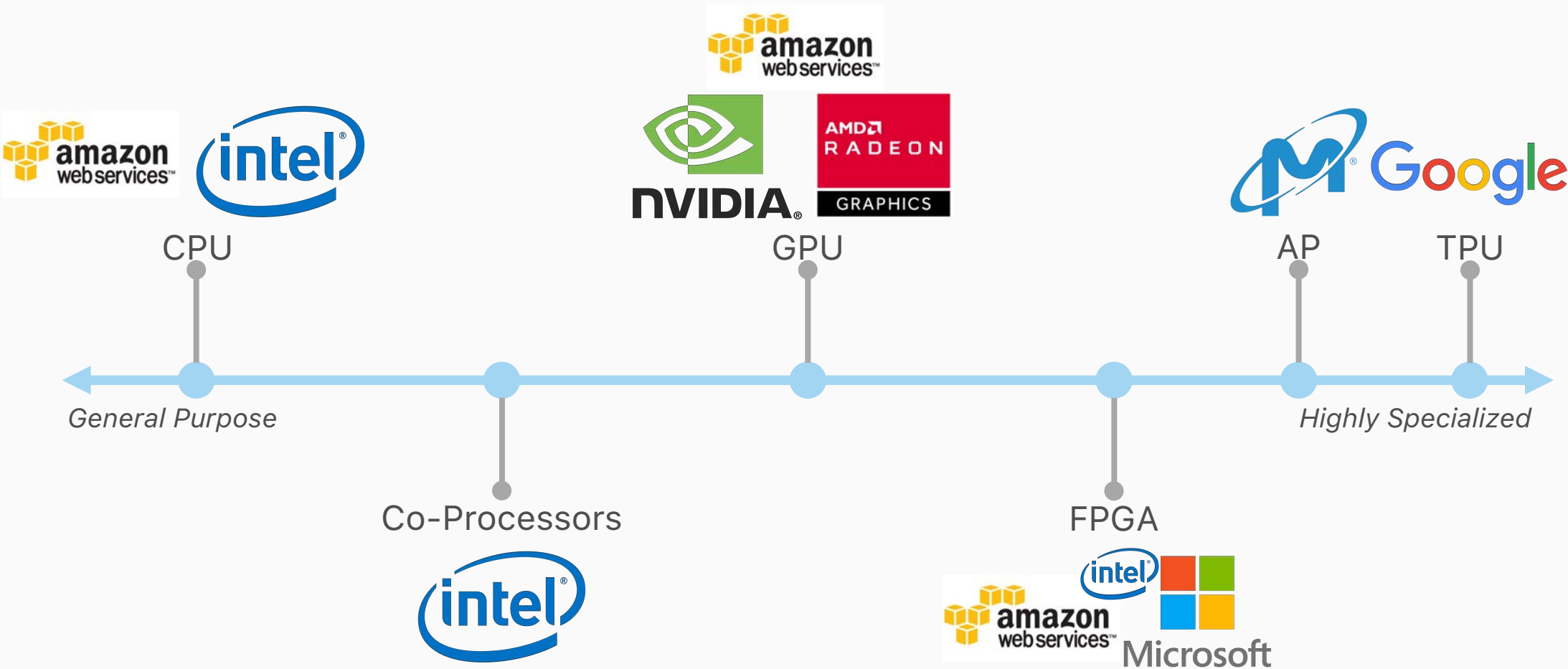


Source: Dazeinfo

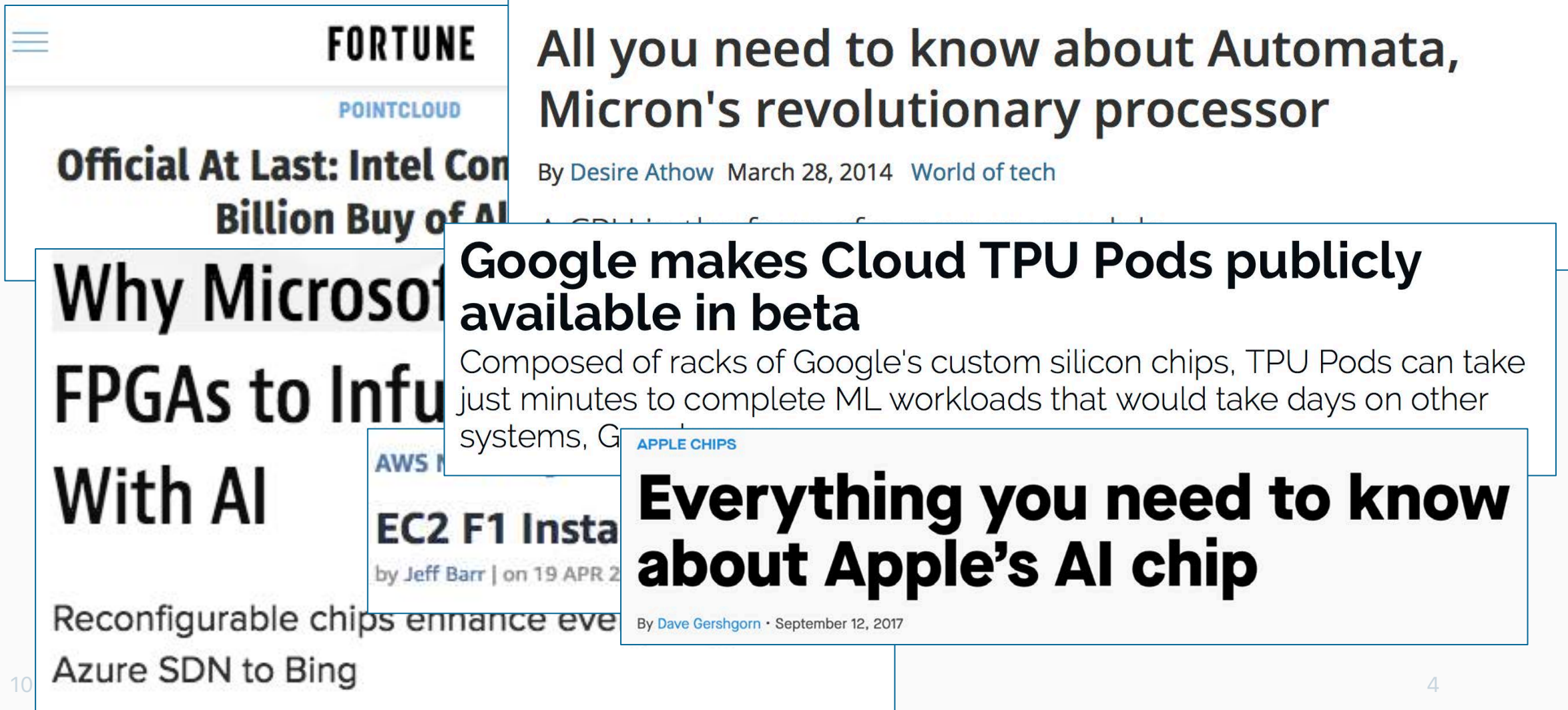


J. M. Shalf and R. Leland, "Computing Beyond Moore's Law". IEEE Computer, 2015.

# New Kinds of Processors



# New Kinds of Processors



FORTUNE

POINTCLOUD

Official At Last: Intel Con  
Billion Buy of AI

## All you need to know about Automata, Micron's revolutionary processor

By Desire Athow March 28, 2014 World of tech

Why Microsoft  
FPGAs to Infu  
With AI

## Google makes Cloud TPU Pods publicly available in beta

Composed of racks of Google's custom silicon chips, TPU Pods can take just minutes to complete ML workloads that would take days on other systems, Google says,

AWS

EC2 F1 Insta

by Jeff Barr | on 19 APR 2

APPLE CHIPS

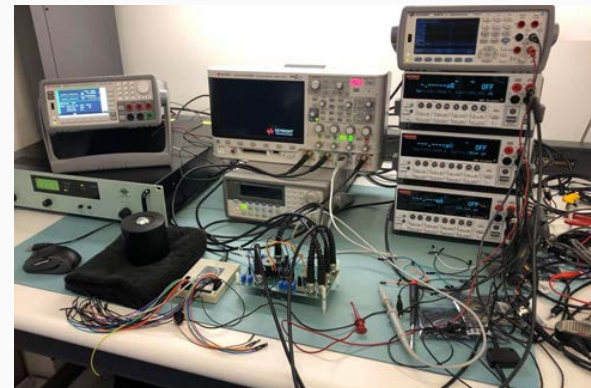
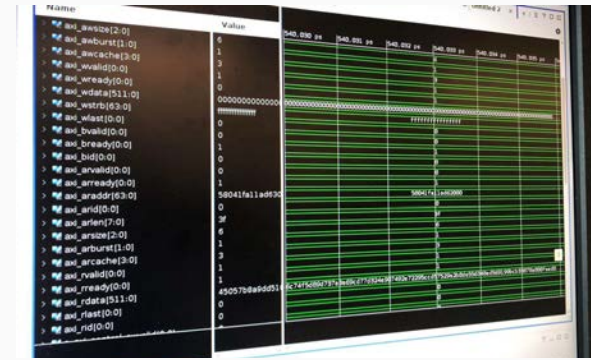
## Everything you need to know about Apple's AI chip

By Dave Gershgorn · September 12, 2017

Reconfigurable chips enhance eve  
Azure SDN to Bing

# Lack of [Good] Programming Models

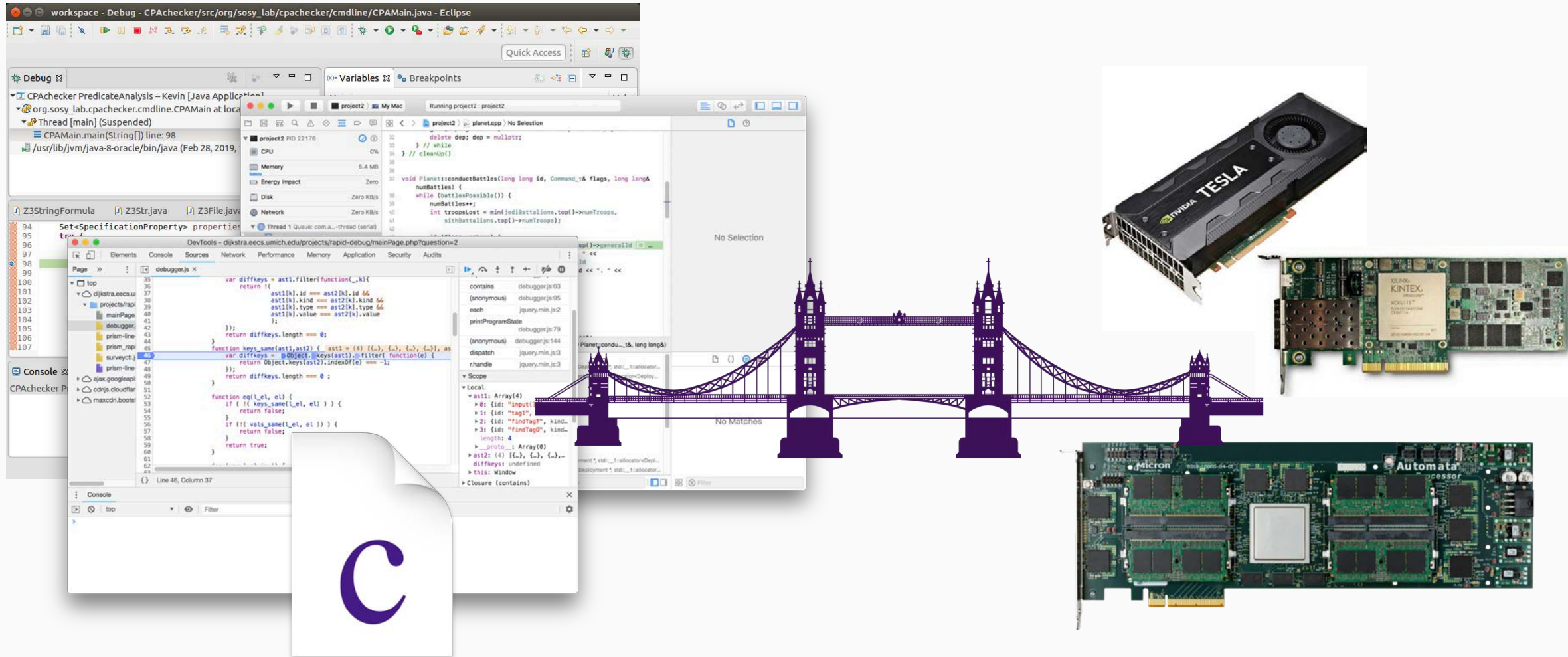
- Akin to “assembly-level” programming on CPU architectures
  - HDLs are *not* an emphasis of CS curricula
- Require **low-level knowledge** of architectural design to produce performant code
- Difficult to **debug** and **maintain**: oscilloscopes and logic analyzers
- Many efforts to improve
  - OpenCL, Xilinx SDAccel, etc.
  - High-level language + annotations + decent performance
  - Rarely compiles out of the box
  - **Non-intuitive impact** of high-level implementation on performance



# Successful Programming Models

- **Performance and Scalability:** minimize overhead introduced by high-level programming models and tools.
- **Ease of Use:** provide familiar abstractions and a shallow learning curve.
- **Expressive Power:** support the applications that developers wish to accelerate with dedicated hardware.
- **Legacy Support:** support the adaptation of existing software to execute efficiently on hardware accelerators while placing a minimal burden on developers.

# Hardware/Software Co-Design





Finite automata provide a suitable abstraction for bridging the gap between high-level programming models and maintenance tools familiar to developers and the low-level representations that execute efficiently on hardware accelerators.

Dissertation Thesis

# Automata Processing in the Big Data World

Detecting Intrusion Attempts in Network Packets

Learning Association Rules with an *a priori* approach

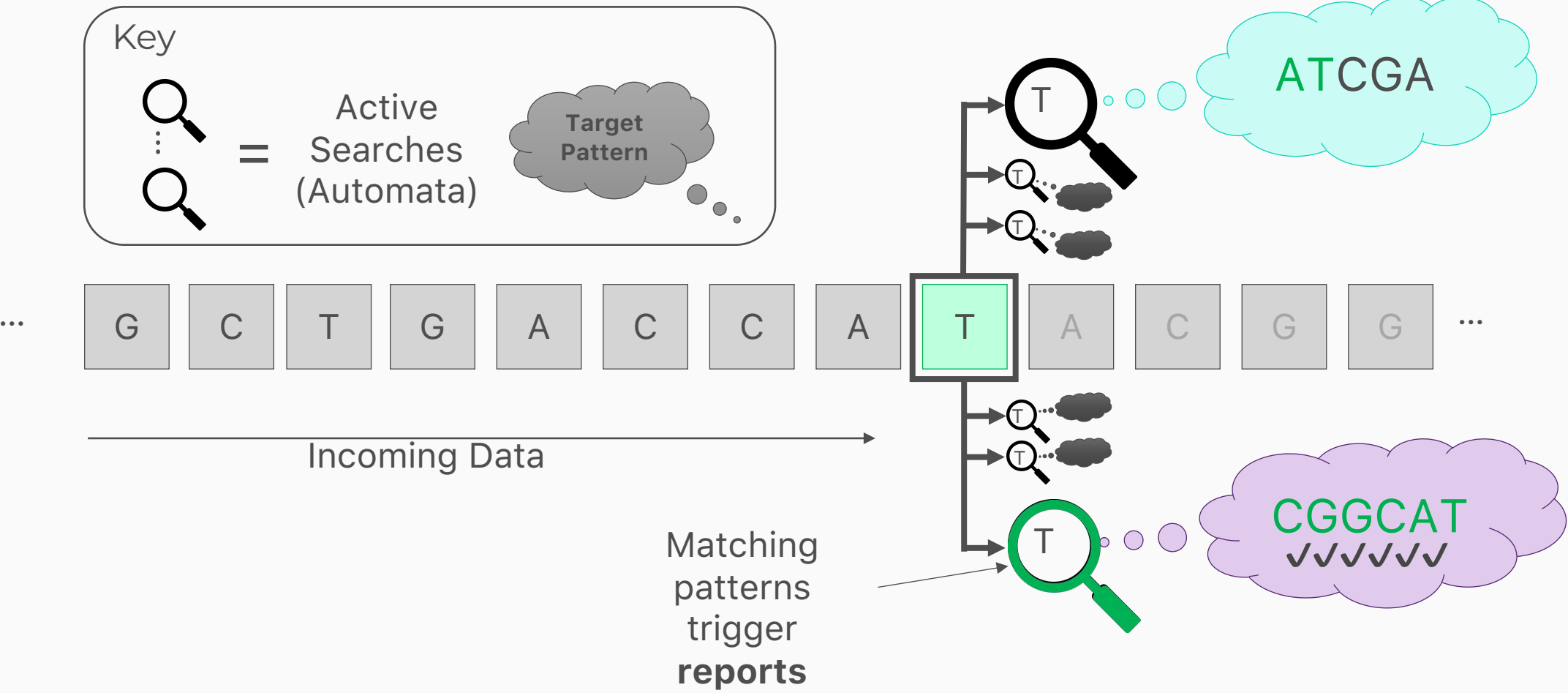
Detecting incorrect POS tags in NLP

Looking for Virus Signatures in Binary Data

Detecting Higgs Events in Particle Collider Data

Aligning DNA Fragments to the Human Genome

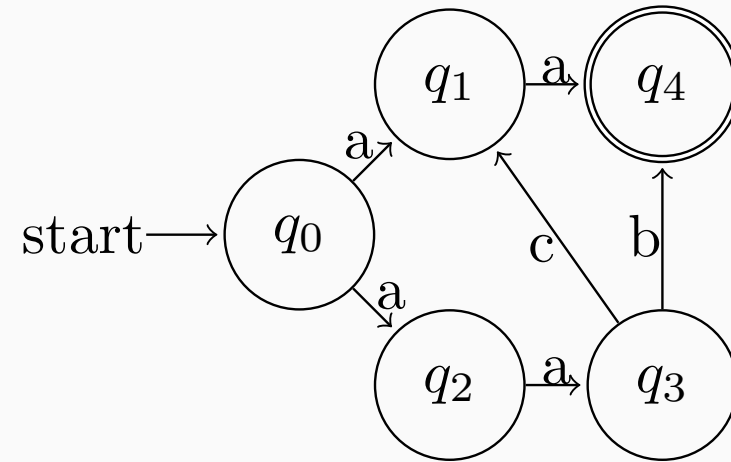
# Finite Automata: 10,000ft View



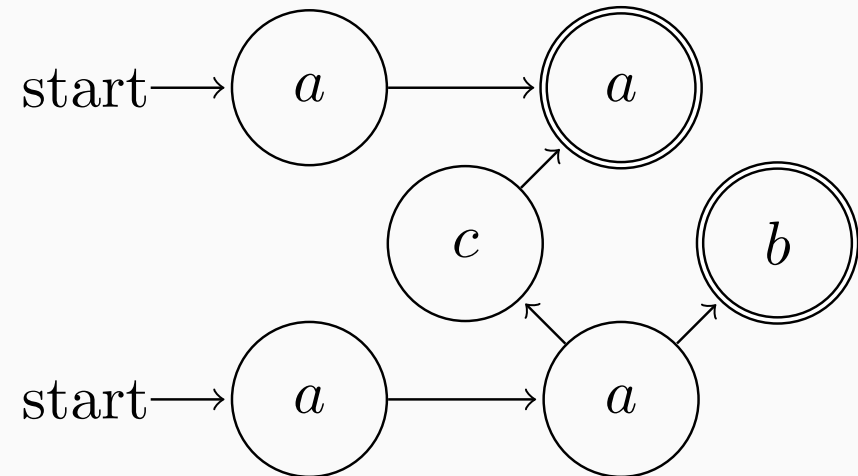
# Homogeneous Finite Automata

- Finite set of states with transitions operating over a finite alphabet
- Input data processed by repeatedly applying transition rules
- **Non-determinism**: multiple transitions on single input
- **Homogeneity**: all incoming transitions occur on the same input character

Traditional NFA



Homogeneous NFA

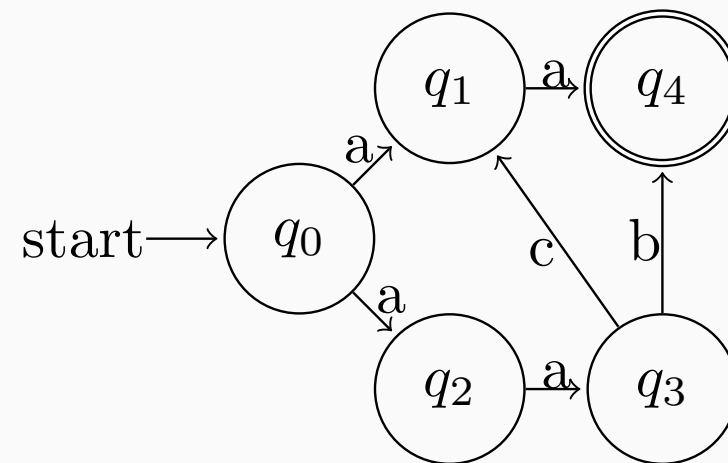


# Homogeneous Finite Automata

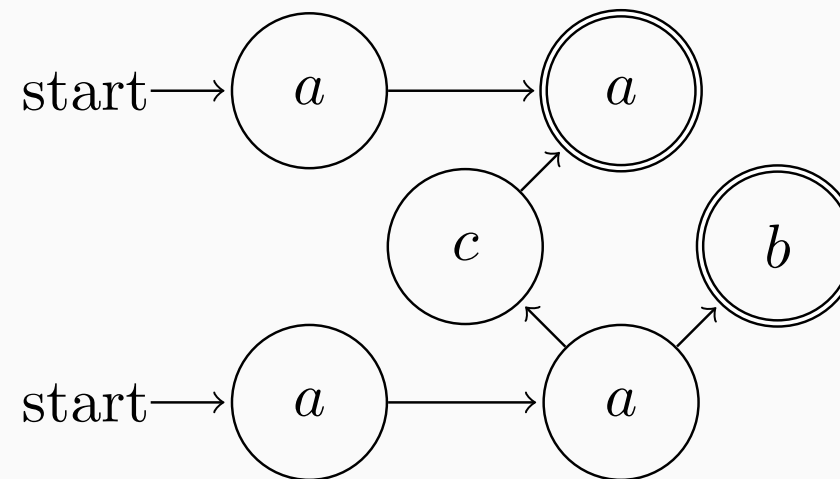
**State Transition Element (STE):** a state in a homogeneous NFA

- **Non-determinism:** multiple transitions on single input
- **Homogeneity:** all incoming transitions occur on the same input character

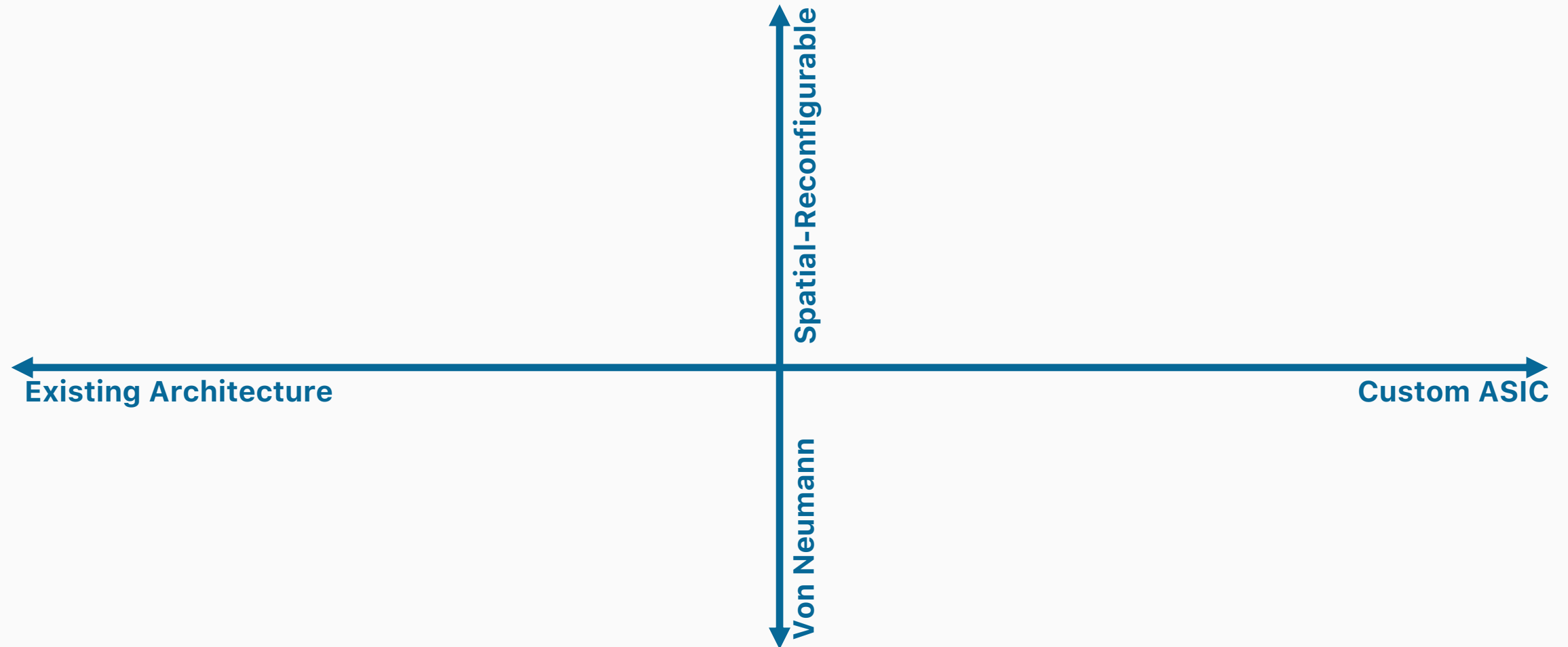
Traditional NFA



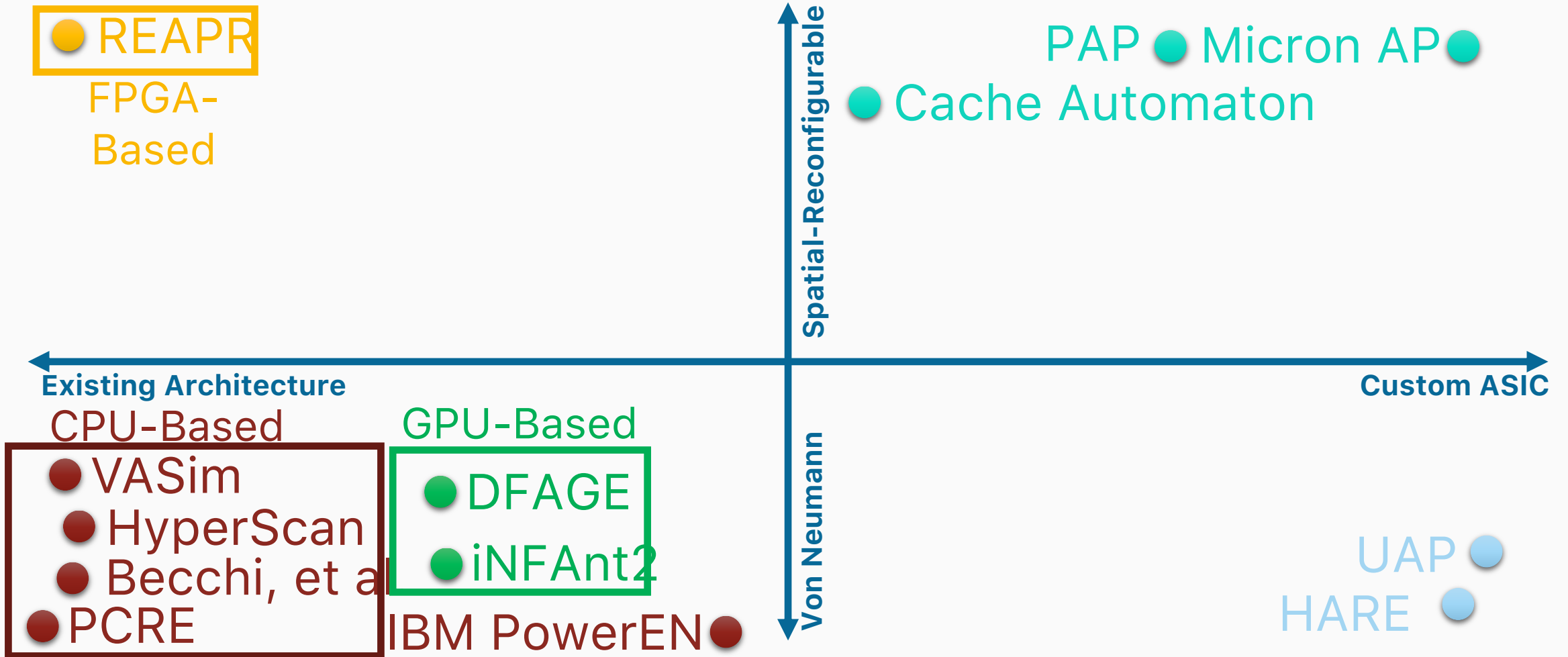
Homogeneous NFA



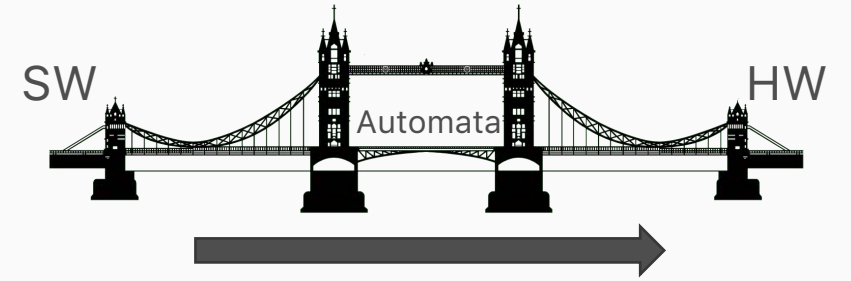
# Automata/RegEx Processing Platforms



# Automata/RegEx Processing Platforms



# Dissertation Overview



- Research Contributions
  - Acceleration of Legacy Code
  - High-Level Programming Language: RAPID
  - High-Speed, Interactive Debugger for Hardware Accelerators
  - Hardware Support for New Application Domains:
    - In-Cache Accelerator for Parsing
    - In-Cache Hardware Unit for Detecting Security Attacks
- Broader Impact and Mentorship
- Conclusions / Discussion



# Acceleration of Legacy Code (String Functions)

ASPLOS 2020

# Legacy Code in the Age of Hardware Accelerators

- Legacy code typically cannot be directly compiled for accelerators
- Learning a new programming model is costly and slows rate of adoption of new accelerators
- May want to “try out” new hardware with existing software
  - No training on new hardware
  - Limited time or resources to allocate

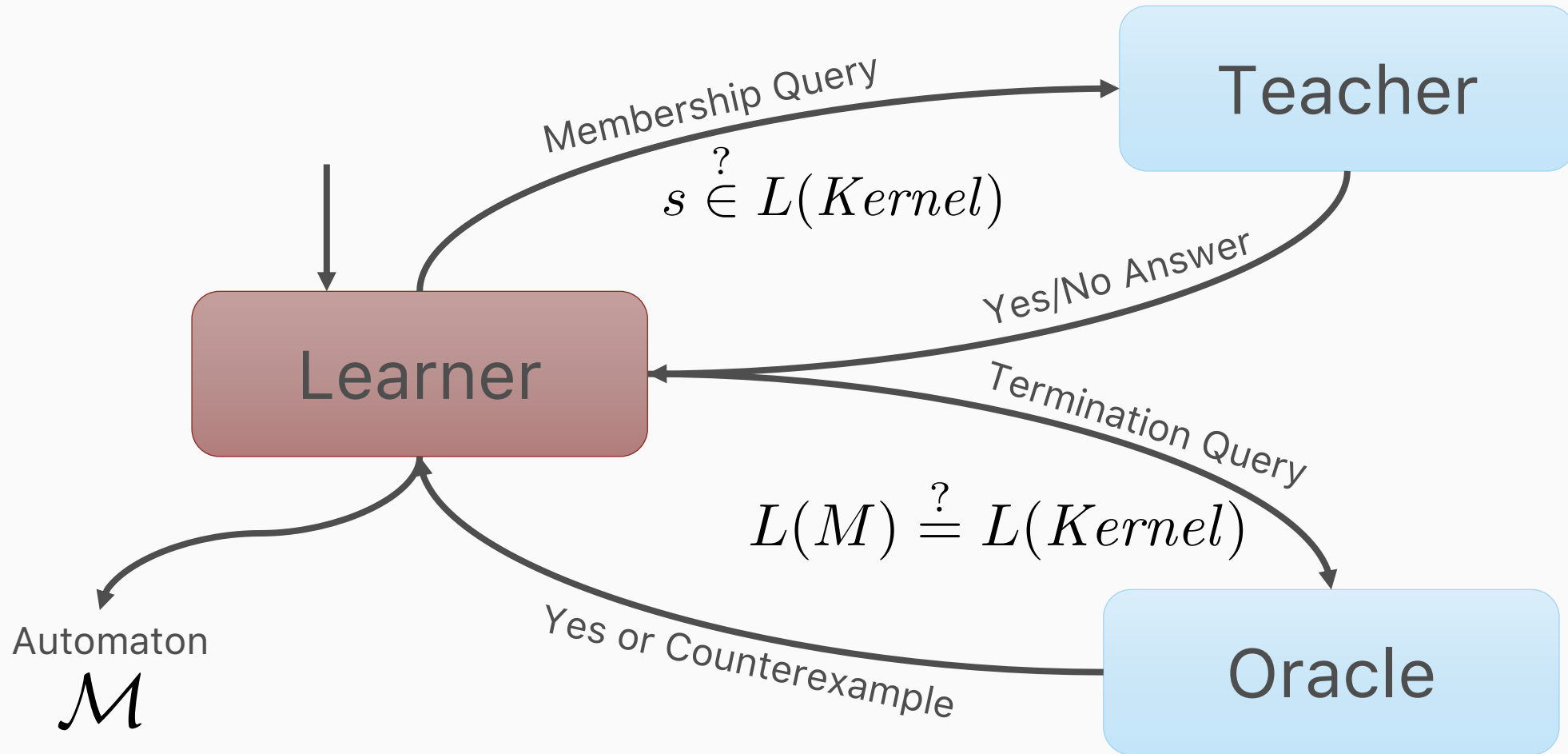
# AutomataSynth at a Glance

- Framework for executing code (legacy software) on FPGAs and other hardware accelerators
- Dynamically observe and statically analyze program behavior to synthesize a **functionally-equivalent hardware design**
- Novel combination of **model learning** (learning theory), **software model checking** (software engineering), **string decision procedures** (PL/theory), and **high-performance automata architectures** (hardware)

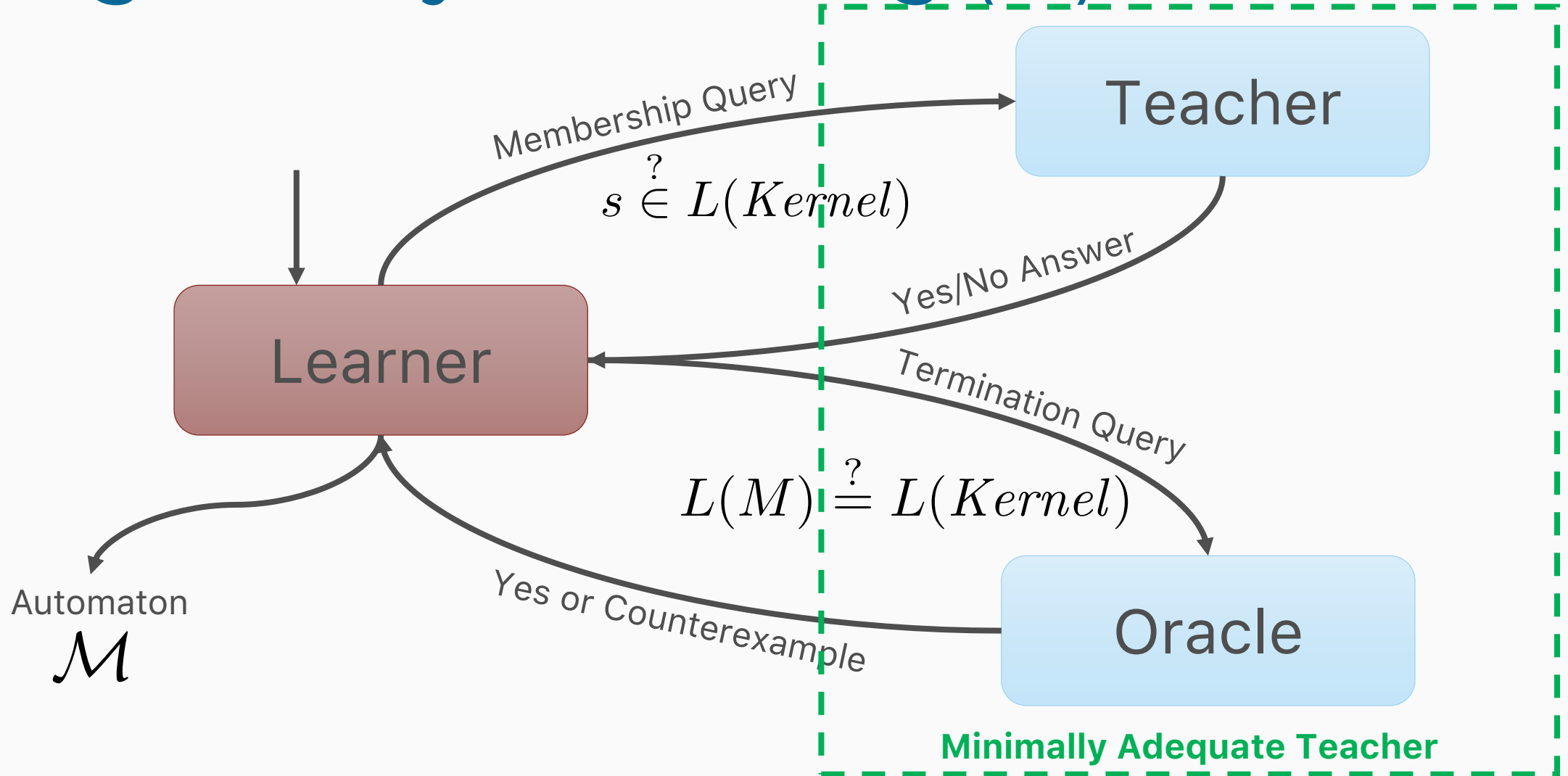
# Problem Statement

- Input: function `kernel : string -> bool`
- Assumptions:
  - Function decides a regular language
  - Source code for function is available
- Output: finite automaton with the same behavior on "all" inputs as kernel

# Angluin-Style Learning ( $L^*$ )



# Angluin-Style Learning ( $L^*$ )



# Membership Queries are Direct

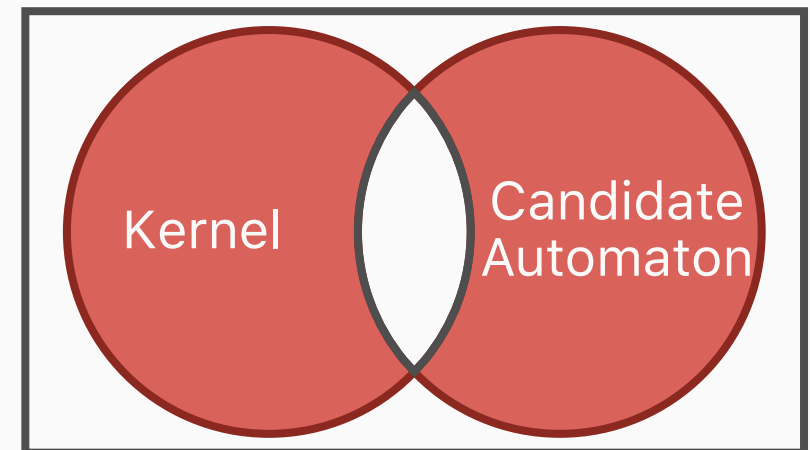
$$s \stackrel{?}{\in} L(\textit{Kernel})$$

- Check if kernel accepts input by **running the code**
- Return value of the kernel is the answer from the MAT
- Caution: take care with ASCII encoding and null terminators (not all functions assume C-style strings)

# Understanding Termination Queries

$$L(M) \stackrel{?}{=} L(\text{Kernel})$$

- Don't have held-out automaton for comparison
- Test inputs generally do not suffice
  - Coverage, generation, etc. difficult challenges
- Constraint over string inputs
  - No inputs that are accepted by the kernel are rejected by the candidate machine (and vice versa)
  - "The symmetric difference is empty"
  - Allows for formulation as a software verification query

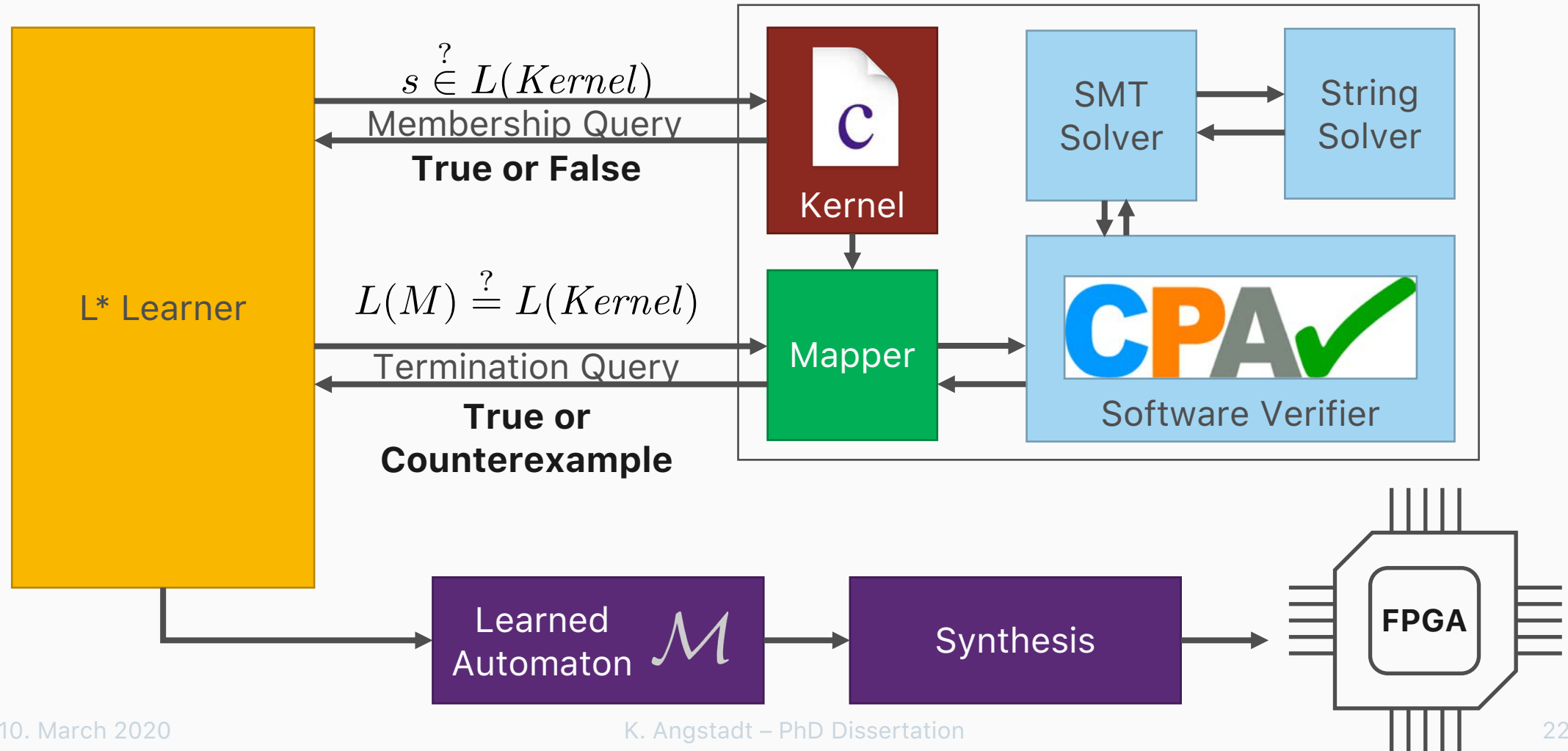




# Equality Checking as Software Verification

- Explores control flow graph looking for property violations
  - Success finding variety of bugs (e.g., double-free, locking violations, etc.)
  - Used in industry for driver verification
- Bounded Model Checking suitable for this domain
  - Verifies that property holds for all program execution up to length  $k$  (i.e., fixed number of loop unrollings)
  - Incremental unrolling to check longer and longer executions
  - Use theorem prover to identify executions that violate property
- Wrapper program to encode the "symmetric difference" property
- Add in string solver to generate counterexamples

# AutomataSynth System Architecture



# Caveats

**Theorem provers are relatively complete.**

- Software verification will occasionally return an **unknown** result
- No **counterexample** is produced, so  $L^*$  cannot continue
- **Implication:** resulting automaton is **approximate**, but correct for all inputs shorter than some fixed bound

**BMC with incremental unrolling is a semi-algorithm.**

- Unrolling of program with infinite loops could continue indefinitely
- Termination query might **never terminate**
- For regular languages **finite unrolling** suffices (See §3.2.4)
- **Implication:** BMC+string solver will terminate and satisfies requirements for Termination Queries

# Guiding Research Questions

- How many real-world string kernels can AutomataSynth correctly learn? With approximation?
- Does AutomataSynth learn automata that fit within the design constraints of modern, automata-derived, reconfigurable architectures?

# Experimental Methodology

- Mine GitHub for string functions in top C repositories
- Use Cil framework to iteratively parse each source file and extract all string functions
- Filter for duplicates and manual analysis to filter on Boolean return type
- Considered 26 repositories, 973 separate string functions, **18 meaningfully-distinct real-world benchmarks**
  - AutomataSynth did not support 3 due to functionality of underlying string solver (e.g., no math on characters)

Benchmark	Project	LOC	Member Queries	Term. Queries	States	Runtime (min)	Correct
git_offset_1st_component	Git: Revision control	6	4,090	2	2	0.12	✓
checkerrormsg	jq: Command-line JSON processor	4	32,664	2	15	1436.58	✓*
checkfail		14	189,013	3	35	1438.47	✓*
skipline		17	7,663	3	3	4.90	✓
end_line	Linux: OS kernel	11	510,623	4	44	491.88	✓
start_line		11	206,613	2	46	80.22	Approx.
is_mcounted_section_name		54	672,041	7	57	1439.98	Approx.
is_numeric_index	MASSCAN: IP port scanner	17	10,727	3	4	4.95	✓
is_comment		11	4,090	2	2	0.23	✓
AMF_DecodeBoolean	OBS Studio: Live streaming and recording software	2	2,557	2	2	0.07	✓
cf_is_comment		28	4,599	2	4	5.00	✓
cf_issplice		22	1,913	2	4	0.05	✓
is_reserved_name		39	240,705	8	42	1424.48	✓
has_start_code	Openpilot: Open-source driving agent	18	10,213	2	7	0.08	✓
stbtt__isfont		24	79,598	5	19	0.22	✓

Benchmark	Project	LOC	Member Queries	Term. Queries	States	Runtime (min)	Correct
git_offset_1st_component	Git: Revision control	6	4,090	2	2	0.1	✓
checkerrormsg	jq: Command-line JSON processor					1436.5	✓*
checkfail						438.4	✓*
skipline						4.9	✓
end_line	Linux: OS kernel					491.8	✓
start_line						80.2	Approx.
is_mcounted_section_name						1439.9	Approx.
is_numeric_index	MASSCAN: IP port scanner	17	10,727	3	4	4.9	✓
is_comment		11	4,090	2	2	0.2	✓
AMF_DecodeBoolean	OBS Studio: Live streaming and recording software	2	2,557	2	2	0.0	✓
cf_is_comment		28	4,599	2	4	5.0	✓
cf_issplice		22	1,913	2	4	0.0	✓
is_reserved_name		39	240,705	8	42	1424.4	✓
has_start_code		18	10,213	2	7	0.0	✓
stbtt__isfont	Openpilot: Open-source driving agent	24	79,598	5	19	0.2	✓

AutomataSynth learns  
13/18 kernels correctly  
and a further 2  
approximately

Benchmark	Project	LOC	Member Queries	Term. Queries	States	Runtime (min)	Correct
git_offset_1st_component	Git: Revision control	6	4,090	2	2	0.12	✓
checkerrormsg		4	32,664	2	15	1436.58	✓*
checkfail	jq: Command-line JSON processor	14	189,013	3	35	1438.47	✓*
skipline		17	7,663	3	3	4.90	✓
end_line		11	510,623	4	44	491.88	✓
start_line	Linux: OS kernel	11	206,613	2	46	80.22	Approx.
is_mcounted_section_name					57	1439.98	Approx.
is_numeric_index	MASSCAN				4	4.95	✓
is_comment	scan				2	0.23	✓
AMF_DecodeBoolean					2	0.07	✓
cf_is_comment					4	5.00	✓
cf_issplice	OBS Studio streaming recording software				4	0.05	✓
is_reserved_name		39	240,705	8	42	1424.48	✓
has_start_code		18	10,213	2	7	0.08	✓
stbtt__isfont	Openpilot: Open-source driving agent	24	79,598	5	19	0.22	✓

Learning took an average of 7 hours. More than half take fewer than 5 minutes



Benchmark	Project	LOC	Member Queries	Term. Queries	States	Runtime (min)	Correct
git_offset_1st_component	Git: Revision control	6	4,090	2	2	0.12	✓
checkerrormsg		4	32,664	2	15	1436.58	✓*
checkfail	jq: Command-line JSON processor	14	189,013	3	35	1438.47	✓*
skipline		17	7,663	3	3	4.90	✓
end_line		11	510,623	4	44	491.88	✓
start_line	Linux: OS kernel	11	206,613	2	46	80.22	Approx.
is_mcounted_section_name		54	672,041	7	57	1439.98	Approx.
is_numeric_index	MASSCAN: IP port scanner	17	10,727	3	4	4.95	✓
is_comment		11	4,090	2	2	0.23	✓
AMF_DecodeBoolean					2	0.07	✓
cf_is_comment					4	5.00	✓
cf_is_splice					4	0.05	✓
is_reserved_name					42	1424.48	✓
has_start_code					7	0.08	✓
stbtt__isfont	source driving agent			5	19	0.22	✓

Learned automata fall within resource constraints of FPGA-based architectures

# AutomataSynth Summary

- Framework for accelerating legacy Boolean string kernel functions using FPGAs
- Constructs a behaviorally-equivalent automaton that can be accelerated with an FPGA
- Novel combination of Angluin-style learning with software model checking and string solvers
- Successfully construct equivalent (or near equivalent) FPGA designs for more than 80% of real-world benchmarks mined from GitHub
- Provides **legacy support** and **performance**

# RAPID: A High-Level Language for Portable Automata Processing

ASPLOS 2016, TPDS 2019

# RAPID at a Glance

- Provides concise, maintainable, and efficient representations for pattern-identification algorithms
- Conventional, C- or Java-style language with domain-specific parallel control structures
- Excels in applications where patterns are best represented as a combination of text and computation
- Compilation to automata supports execution on AP, FPGAs, CPUs, and GPUs

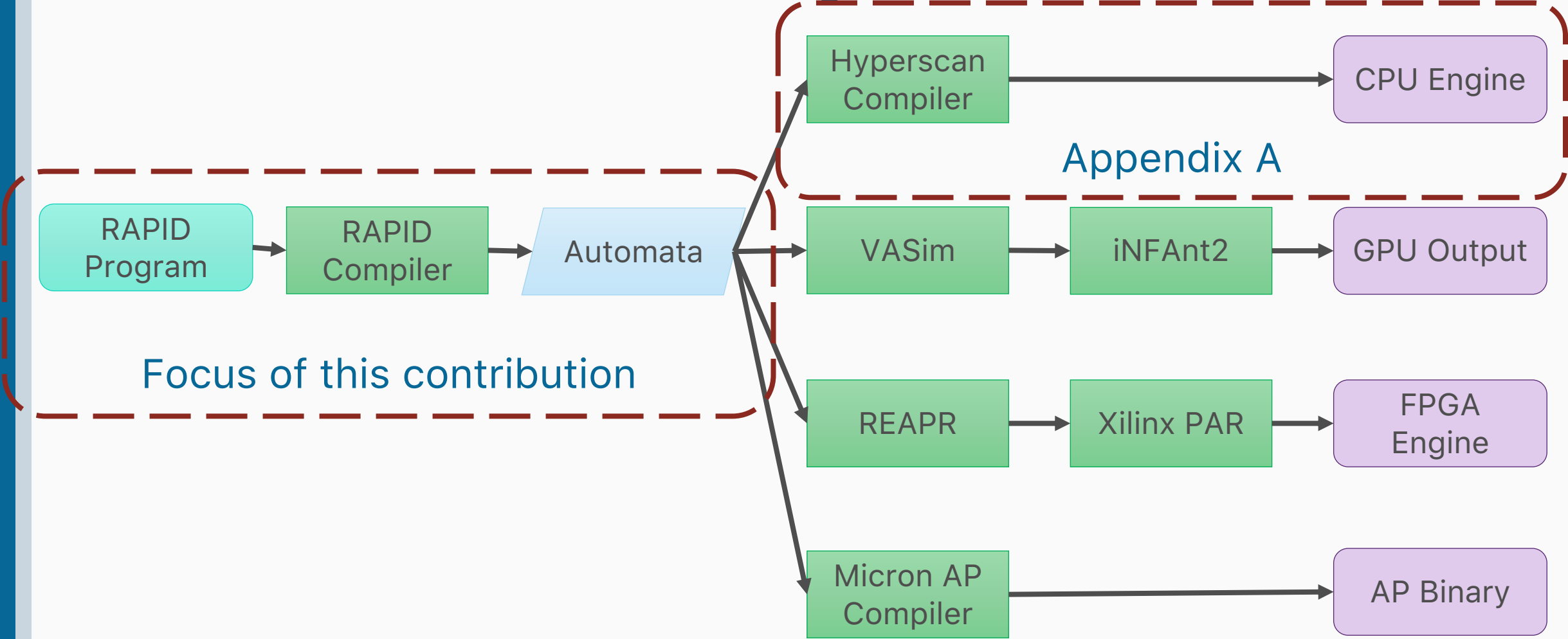
# Parallel Control Structures

- Concise specification of multiple, simultaneous comparisons against a single data stream
- Support common pattern search paradigms
- Static and dynamic thread spawning for massive parallelism support
- Explicit support for sliding window computations

@NITBDELGMVUDBQZZDWIEFHPTG@ZBGEXDGHXSVCMKADSKFJÖKLGJADSKGOWESIOHGADHYCBGOASDGBAEGKQEYKPREBN...



# Multi-Architecture System Overview



# Experimental Results (Summary)

- Successfully ported benchmarks from real-world applications (**expressiveness**)
  - Demonstrated RAPID can implement regular expressions
- 2-8x more compact than scripts generating automata and 13-71x more compact than defining automata (**scalability**)
  - RAPID size remains constant or grows sublinearly with application instance size
- Overheads of compiled RAPID on FPGA and AP are within 15% of hand-optimized automata (**performance**)
- Automata optimizations supports more stable porting of applications across architectures than OpenCL (**performance**)

# RAPID Summary

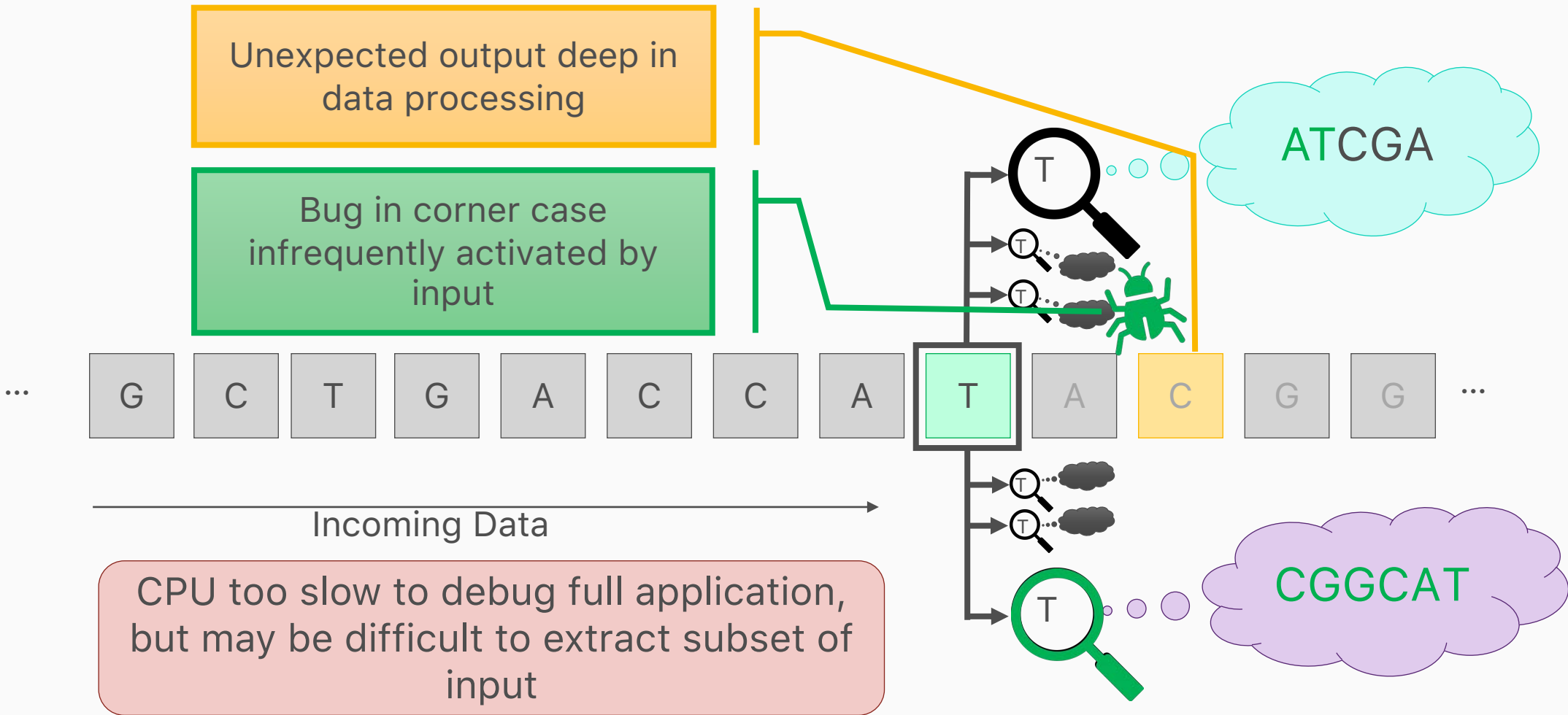
- RAPID allows developers to write new pattern-searching programs for hardware accelerators
  - Programs compile to a set of finite automata
  - Domain-specific parallel control structures support common tasks
  - Programs are significantly more concise than hand-crafted automata
- RAPID programs can execute on a wide variety of hardware platforms (FPGA, AP, CPU, GPU, etc.)
  - Portability of automata provides more stable performance across architectures
- Provides **scalability** and **performance**



# Interactive Debugging for High-Level Languages and Accelerators

ASPLOS 2019

# Houston, we have a problem!

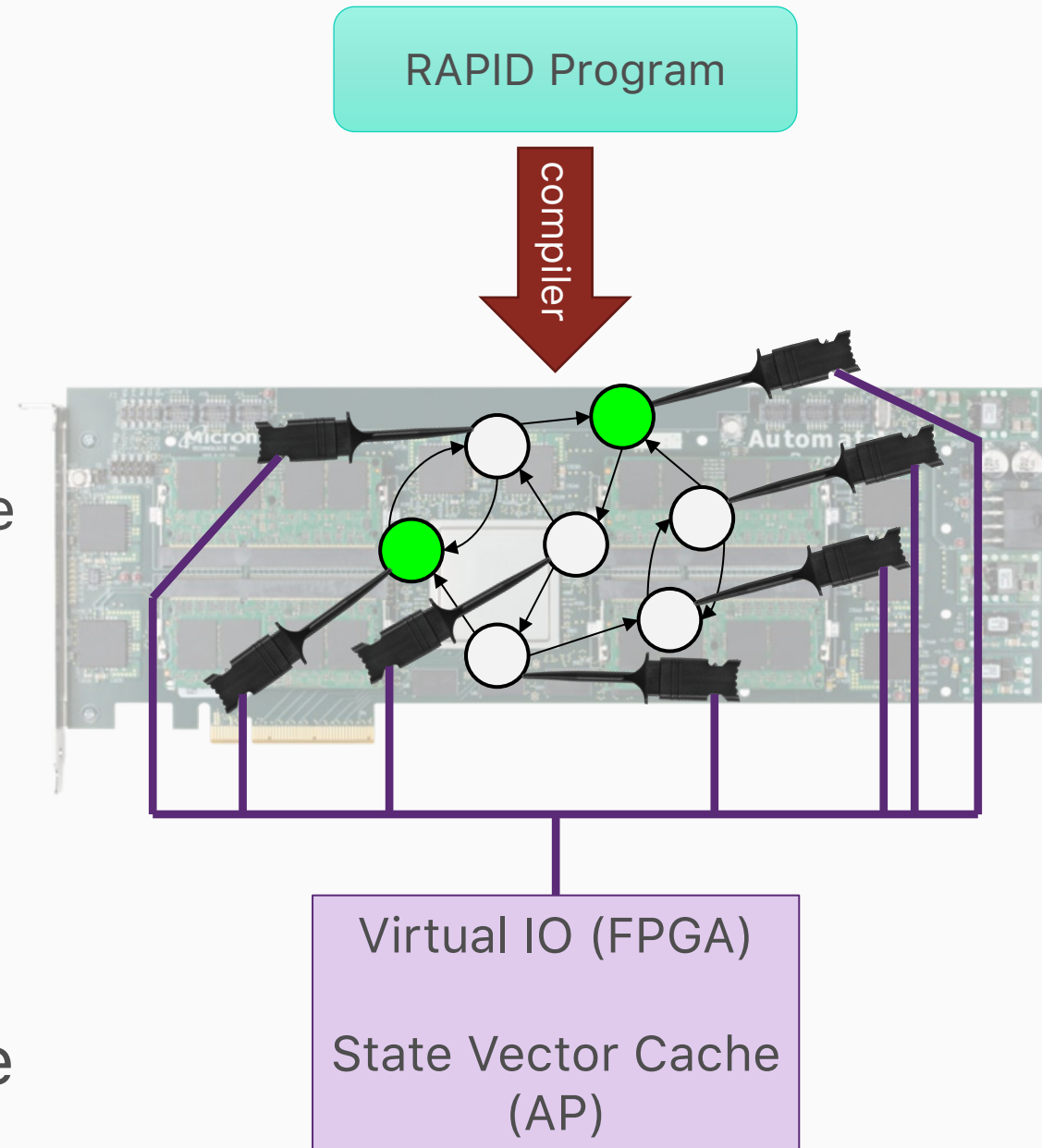


# Where do we stop?

- **Breakpoints** annotate expressions/statements to specify locations to pause execution for inspection
  - Traditional notion relies on instruction streams
  - Mechanism does not apply directly to architectures with no instructions (e.g., FPGAs, AP)
- **Key Insight:** Automata computation driven by input
  - Set breakpoints on input data, not instructions
  - Supports use case of stopping computation at abnormal behavior
  - Can also provide abstraction of traditional breakpoints

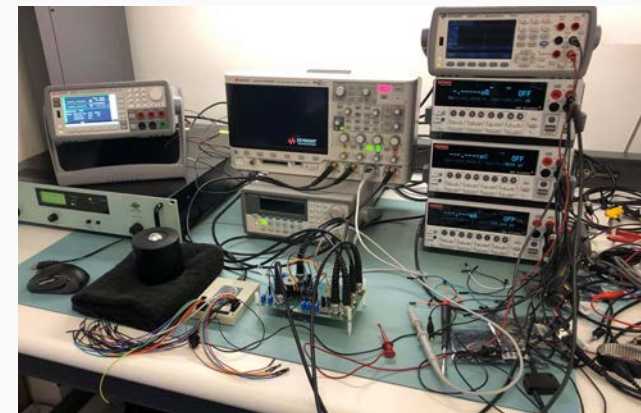
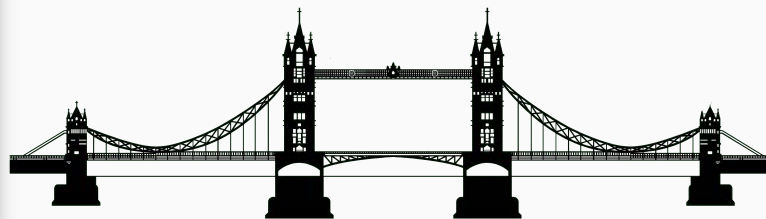
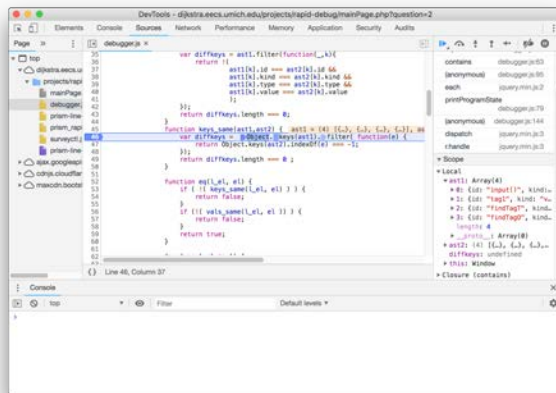
# Capturing State

- Process input data up to breakpoint
- State of automata is **compact**
  - $O(n)$  in the number of states of the NFA
- Repurpose existing hardware to capture
  - AP: State vector cache
  - FPGA: Integrated Logic Analyzers (ILAs) and Virtual I/O pins (VIOs) allow for probing of activation bits
- Cache state vectors to decrease latency



# Bridging the Gap

- Developer can set breakpoint in program or data
- Automatically translate this location to automaton states
- Use hardware test equipment to monitor and read state information in the circuit
- Automatically translate HW information back to program source code



# Experimental Results Summary

- Using server-class FPGA and standard ANMLZoo benchmarks, we found debugging required 3x logic elements and 6x register elements of baseline designs (**scalability**)
- Debugging of ANMLZoo benchmarks can occur at ~80% of baseline clock frequencies (**performance**)
- Human study of 61 undergraduate and graduate programmers found statistically significant increase in fault location accuracy (**ease of use**)
  - Debugger aided novices and relative experts alike

# Debugging Summary

- Developers are now able to debug on FPGAs using high-level languages (e.g., RAPID)
- Bridge the semantic gap by storing mappings between program expressions and low-level hardware resources
- Leverage Virtual I/O on FPGAs to capture state
  - Automata abstraction produces minimal state to capture
- Provides **performance**, **scalability**, and **ease of use**

# Architectural Support for New Application Domains

MICRO 2018, (Current Manuscript)



# Hardware/Software Co-Design

The image is a collage illustrating hardware/software co-design. It features several key elements:

- Software Development:** Multiple screenshots of development environments. One shows Eclipse IDE with a Java project 'workspace - Debug - CPAChecker/src/org/sosy\_lab/cpachecker/cmdline/CPAMain.java'. Another shows DevTools with a JavaScript debugger for 'dijkstra.eecs.umich.edu/projects/rapid-debug/mainPage.php?question=2'. A third shows a code editor with JavaScript code for a graph algorithm.
- Hardware Components:** Three hardware boards are shown: a black NVIDIA Tesla GPU, a green Xilinx Kintex FPGA board, and a green Automata board with a central processor and various components.
- Visual Elements:** A purple silhouette of the Tower Bridge is overlaid across the IDE screenshots. A white sticky note with a purple letter 'C' is positioned in the bottom left corner.

# Two New Application Domains

## **XML Parsing**

- Parsing of data central to most (all?) data processing pipelines
- XML is one of the most common formats
- Parsing notoriously difficult to accelerate

## **Detecting Security Attacks**

- Continual cat and mouse game
- Recent discoveries of hardware bugs leave billions of devices vulnerable
- Current fixes are costly and/or ineffectual

```

<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. MCELLOWNEY
  </instructor>
  <limit>0112</limit>
  <enrolled>0108</enrolled>
</course>

```

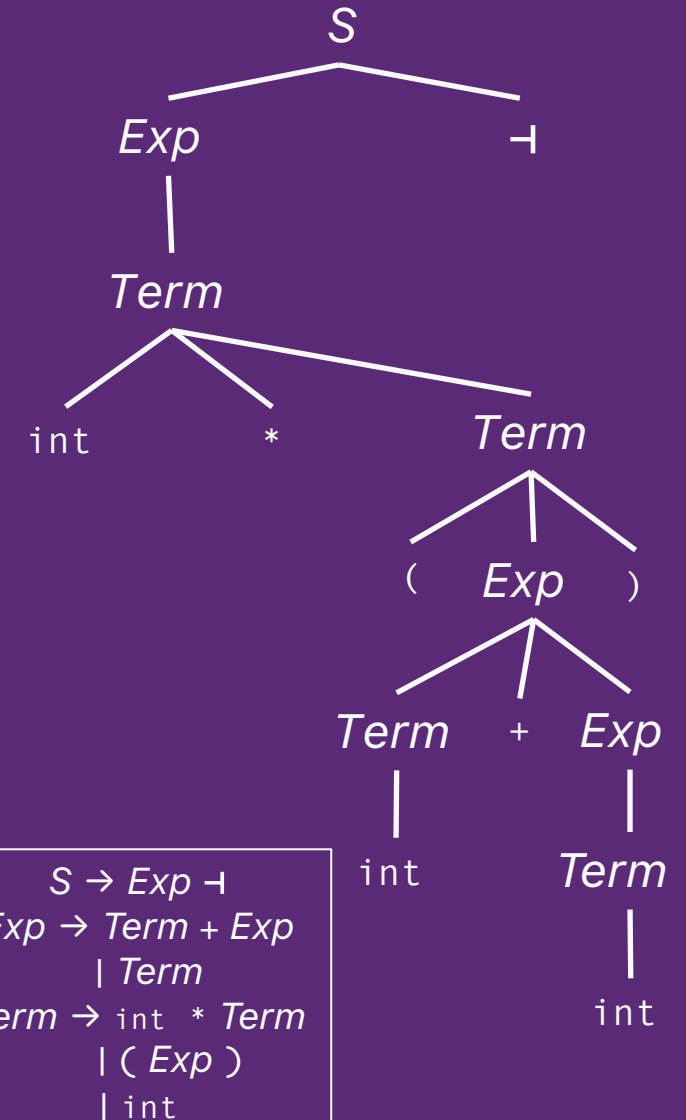
## XML Nesting

```

{
  "id": "0t_15l_5r",
  "type": "hState",
  "enable": "onActivateIn",
  "report": true,
  "inputDefs": [
    {
      "width": 1,
      "portId": "i"
    }
  ],
  "outputDefs": [
    {
      "width": 1,
      "activate": [],
      "portId": "o"
    }
  ],
  "attributes": {
    "reportId": 5,
    "latched": false,
    "symbolSet": "[\\xFF]"
  }
}

```

## JSON Encoding

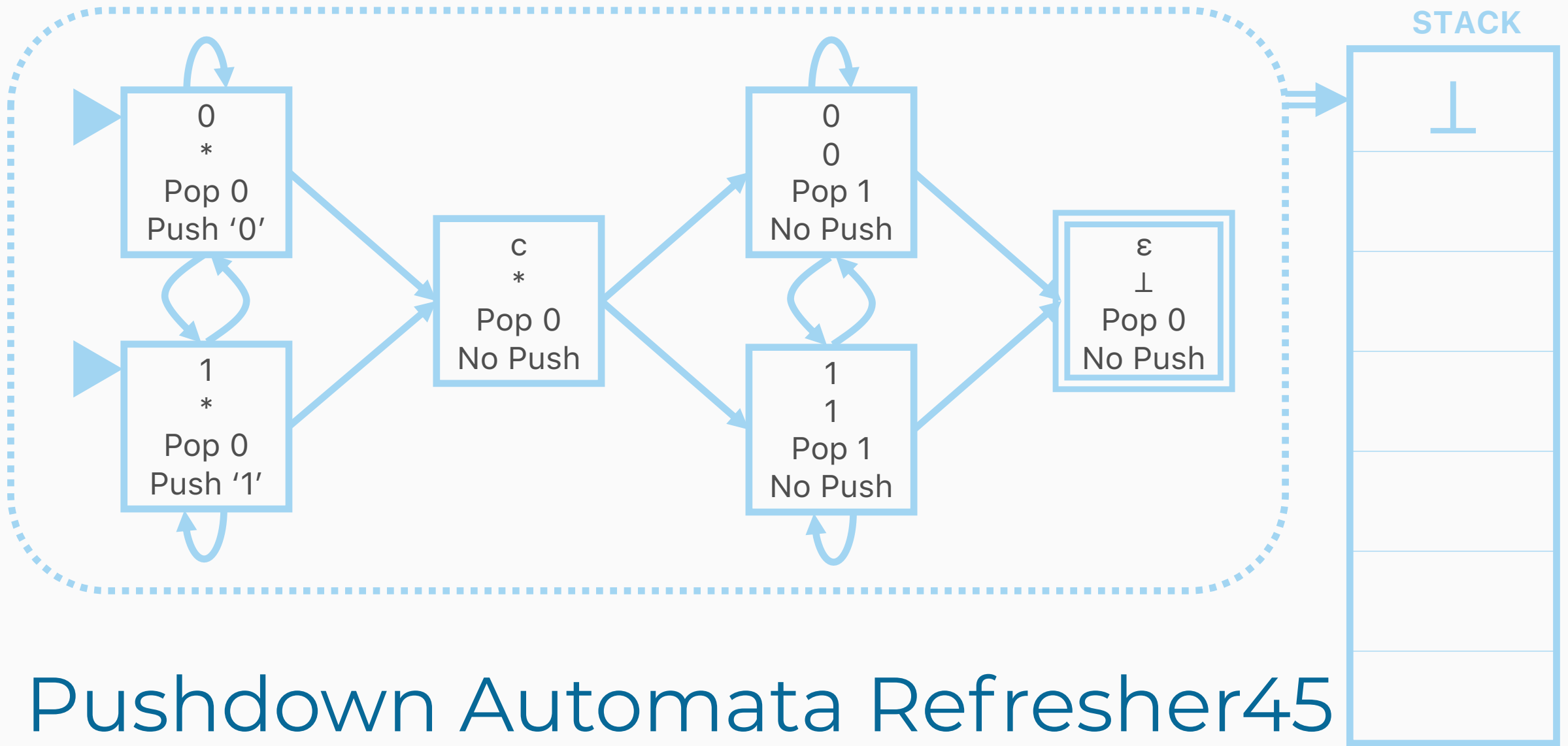


```

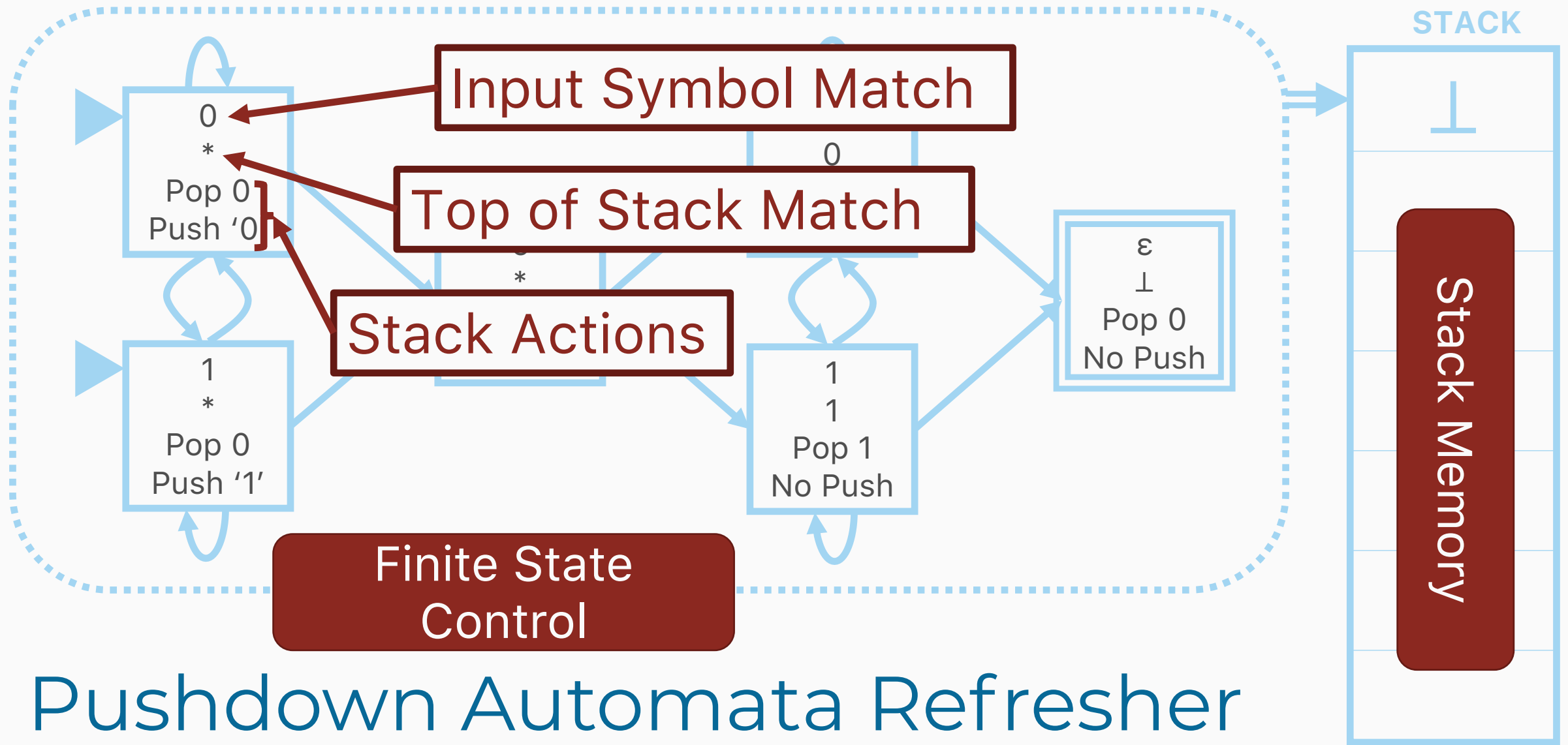
S → Exp +
Exp → Term + Exp
    | Term
Term → int * Term
    | ( Exp )
    | int

```

## Parsing



# Pushdown Automata Refresher45

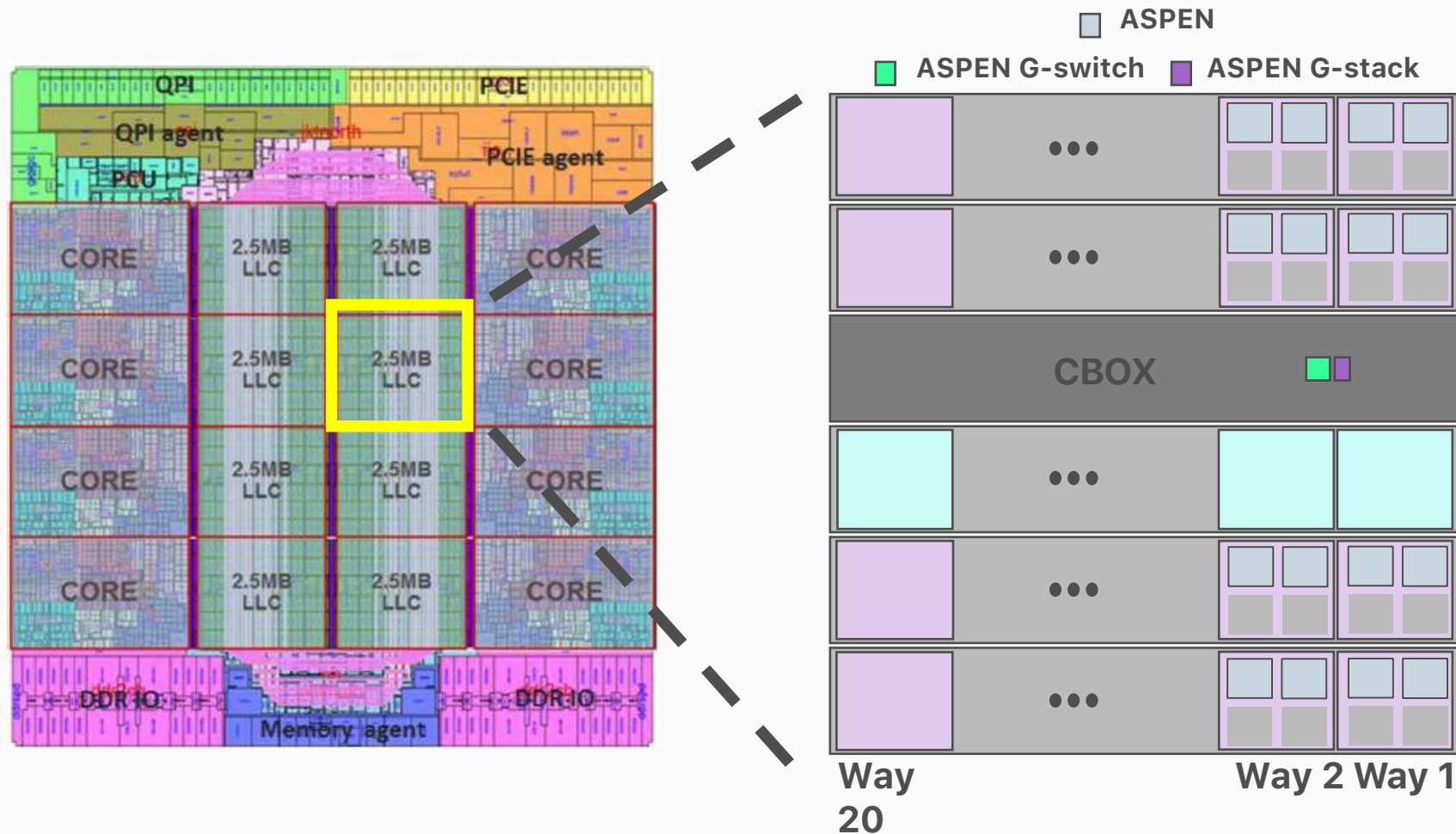


# Pushdown Automata Refresher

# ASPEN Supports Richer Analyses

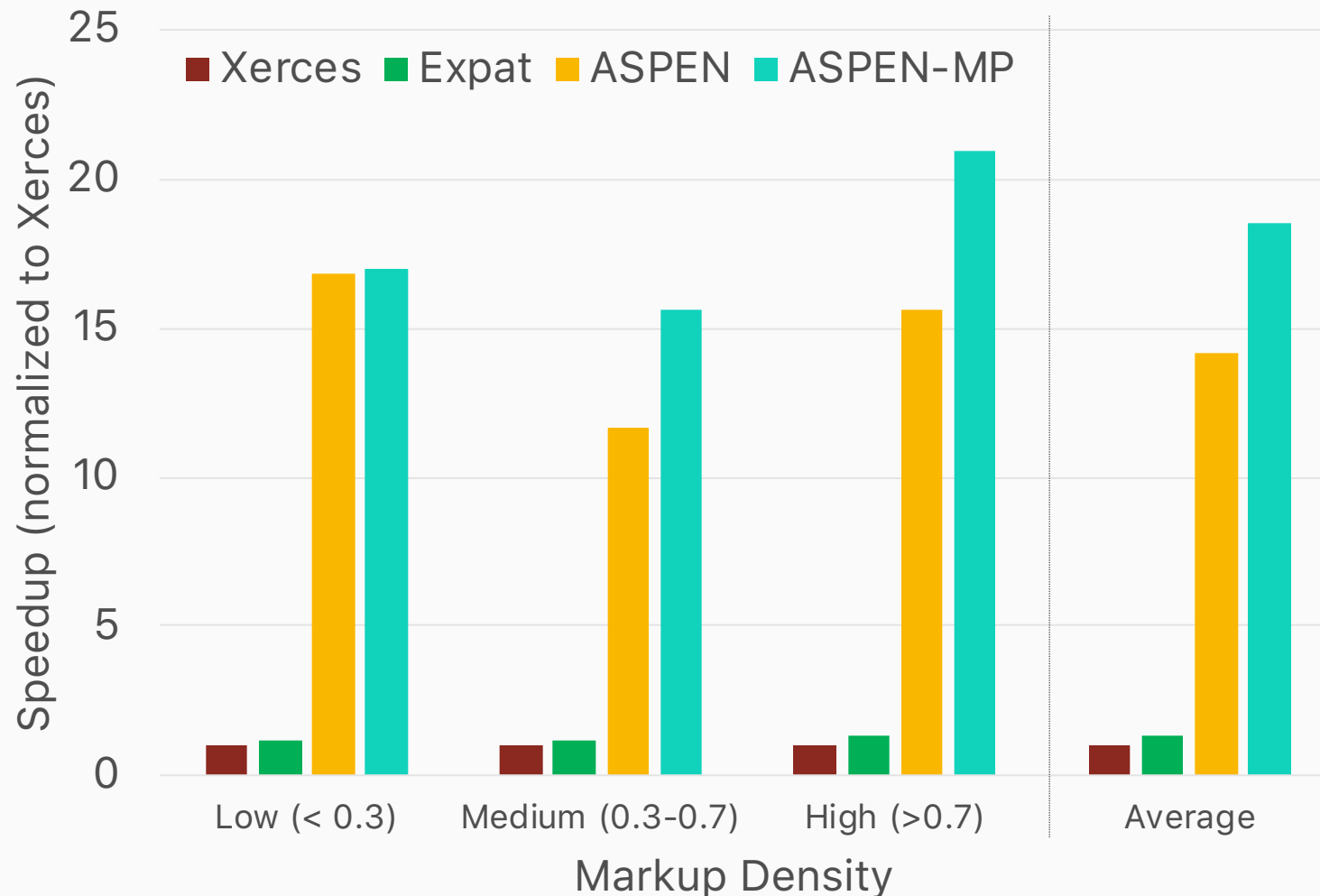
- Accelerated in-**SRAM Pushdown EN**gine
- Scalable processing engine that **uses LLC slices** to accelerate Pushdown Automata computation
- Custom five-stage datapath using SRAM lookups can process up to **one byte per cycle**
- Optimizing compiler **supports existing grammars**, packs states efficiently,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

# 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

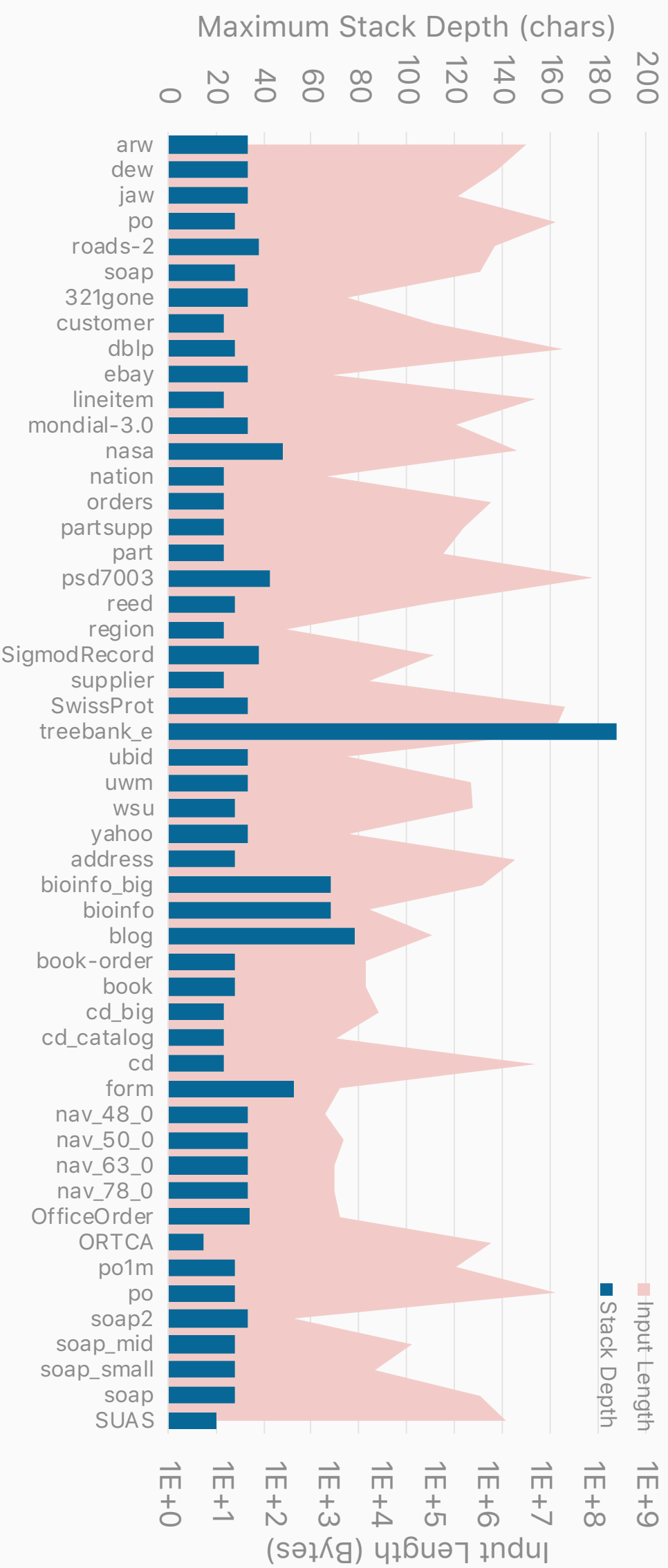
# XML Parsing Experimental Results



- Benchmarks: 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  $\epsilon$ -stalls



# Maximum Stack Depth



# Processor Designs are Flawed

TECHNOLOGY

## The New Way Your Computer Can Be Attacked

Un  
cyb  
BRU

# Spectre and Meltdown explained: What they are, how they work, what's at risk

Spectre and Meltdown are the names given to a trio of variations on a vulnerability that affects nearly every computer chip manufactured in the last 20 years. The flaws are so fundamental and widespread that security researchers are calling them catastrophic.



By **Josh Fruhlinger**  
CSO | JAN 15, 2019

## Intel Performance Hit 5x Harder Than AMD After Spectre, Meltdown Patches

By Joel Hruska on May 20, 2019 at 1:46 pm | [255 Comments](#)

# Processor Designs are Flawed

TECHNOLOGY

The New

Un Spectre  
cyb are,

BRU

Spectre  
nearly e  
and wide



**Davey Winder** Senior Contributor ⓘ

Cybersecurity

*I report and analyse breaking cybersecurity and privacy stories*



By **Josh Fruhlinger**

CSO | JAN 15, 20

## Intel Performance Hit 5x Harder Than AMD After Spectre, Meltdown Patches

By Joel Hruska on May 20, 2019 at 1:46 pm | **255 Comments**

EDITOR'S PICK | 4,927 views | Nov 13, 2019, 09:18am

## Intel Confirms 'ZombieLoad 2' Security Threat

ability that affects  
to fundamental

# Understanding Spectre and Meltdown

- **Conditional branches** (if statements) are slow
  - Time needed to determine next instruction to execute
- Modern processors use **speculation** to guess the next instruction to execute
  - When the processor guesses wrong, it can "undo" running the speculated instruction
- Modern computers use caches to speed up access to data
  - Processors do not "undo" changes made to cache data as a result of speculation
  - Careful programming **combines misprediction and caching to steal information**

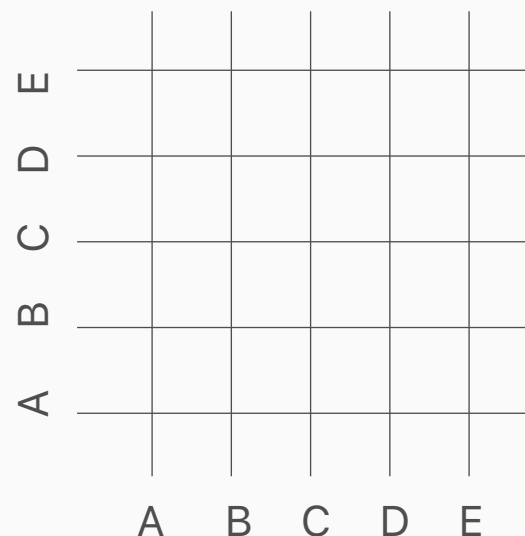
# Anomaly Intrusion Detection

- Fixing hardware takes time and has significant cost
- In the meantime, can we detect attacks with minimal modifications?
- **Key Idea:** Run known good programs many times to **learn correct behavior** of the software
- Use this model to categorize other running programs
- What do we monitor?
  - Previous work monitored sequences of **system (OS) calls**
  - Does not capture the speculative behavior we want to monitor
  - Instead, monitor **sequence of memory accesses**

# Basic Approach: Sliding Windows

- Record **sequences of memory addresses** accessed by a program during execution
- **Sliding windows** captures order in the sequences
  - Windows from training stored in a **dictionary**
  - Can be encoded using **finite automata**

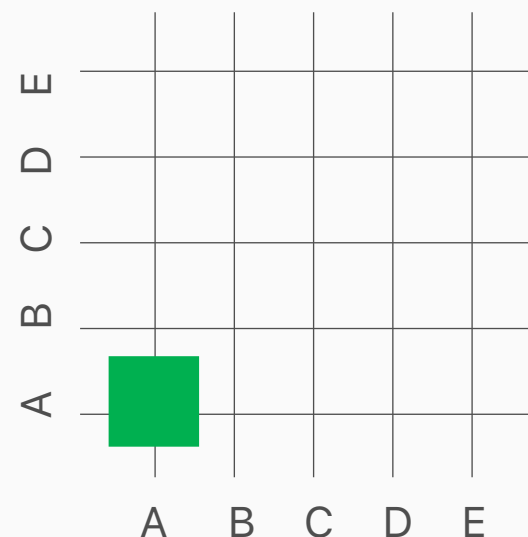
Training  
Memory **AABACCBBC**



# Basic Approach: Sliding Windows

- Record **sequences of memory addresses** accessed by a program during execution
- **Sliding windows** captures order in the sequences
  - Windows from training stored in a **dictionary**
  - Can be encoded using **finite automata**

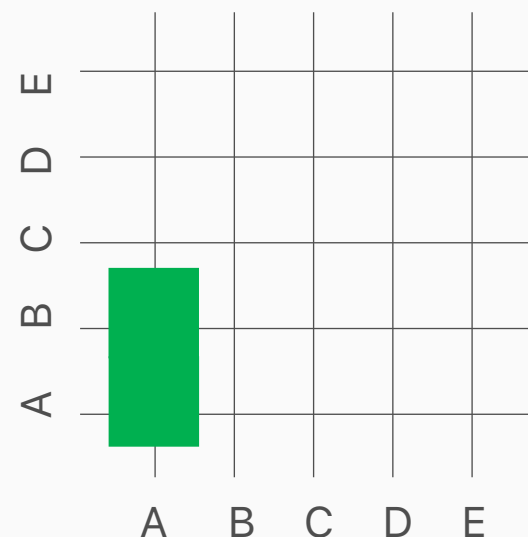
Training  
Memory **AABACCBBC**



# Basic Approach: Sliding Windows

- Record **sequences of memory addresses** accessed by a program during execution
- **Sliding windows** captures order in the sequences
  - Windows from training stored in a **dictionary**
  - Can be encoded using **finite automata**

Training  
Memory **AABACCBBC**

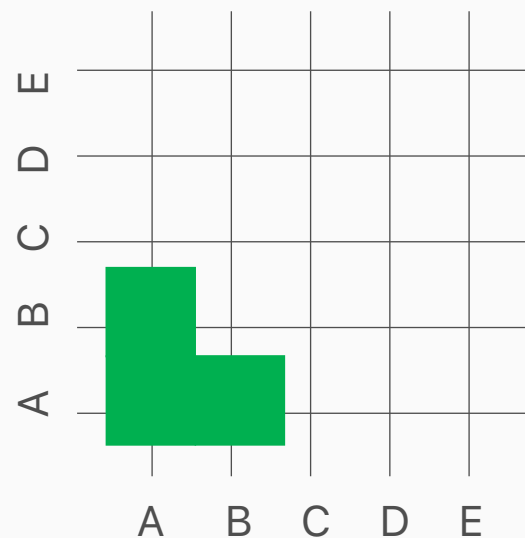




# Basic Approach: Sliding Windows

- Record **sequences of memory addresses** accessed by a program during execution
- **Sliding windows** captures order in the sequences
  - Windows from training stored in a **dictionary**
  - Can be encoded using **finite automata**

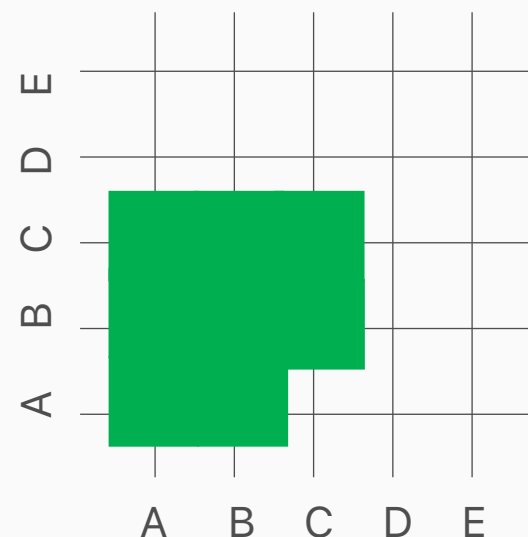

Training  
Memory **AABACCBBC**



# Basic Approach: Sliding Windows

- Record **sequences of memory addresses** accessed by a program during execution
- **Sliding windows** captures order in the sequences
  - Windows from training stored in a **dictionary**
  - Can be encoded using **finite automata**


Training  
Memory **AABACCBBC**

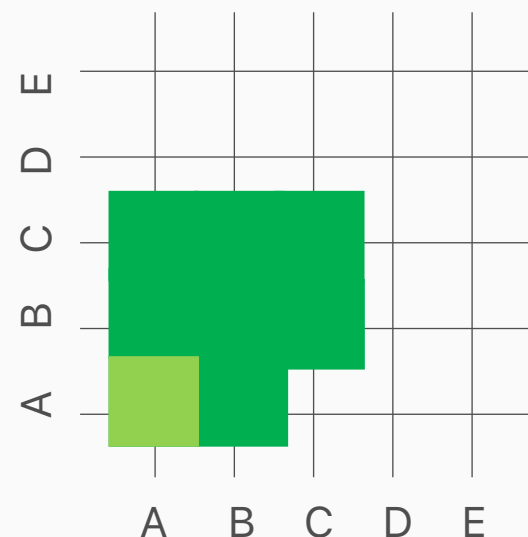


# Basic Approach: Sliding Windows

- Record **sequences of memory addresses** accessed by a program during execution
- **Sliding windows** captures order in the sequences
  - Windows from training stored in a **dictionary**
  - Can be encoded using **finite automata**

Training Memory **AABACCBBC**

Testing Memory **AAECD**  


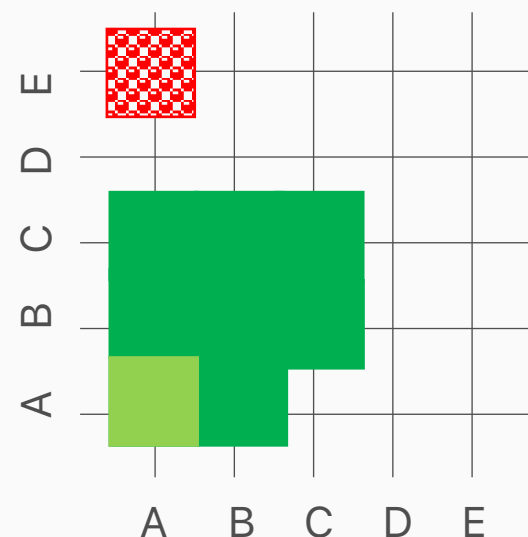


# Basic Approach: Sliding Windows

- Record **sequences of memory addresses** accessed by a program during execution
- **Sliding windows** captures order in the sequences
  - Windows from training stored in a **dictionary**
  - Can be encoded using **finite automata**

Training Memory **AABACCBBC**

Testing Memory **AAECD**  

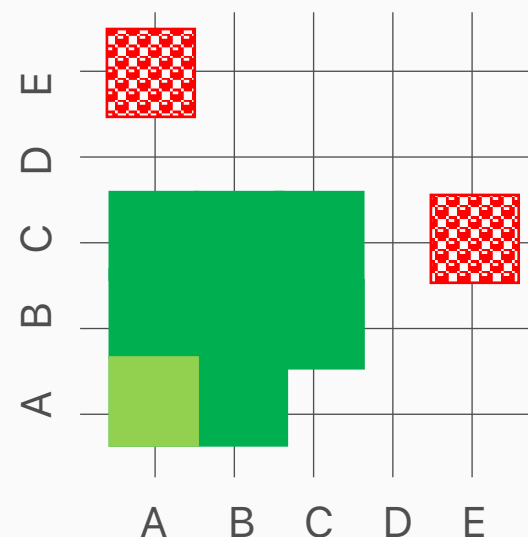



# Basic Approach: Sliding Windows

- Record **sequences of memory addresses** accessed by a program during execution
- **Sliding windows** captures order in the sequences
  - Windows from training stored in a **dictionary**
  - Can be encoded using **finite automata**

Training Memory **AABACCBBC**

Testing Memory **AAECD**  
}

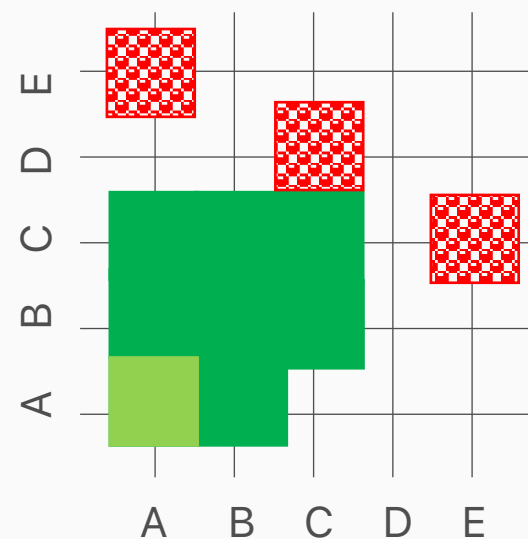


# Basic Approach: Sliding Windows

- Record **sequences of memory addresses** accessed by a program during execution
- **Sliding windows** captures order in the sequences
  - Windows from training stored in a **dictionary**
  - Can be encoded using **finite automata**

Training Memory **AABACCBBC**

Testing Memory **AAECD**  
}



# MARTINI: Detecting Attacks

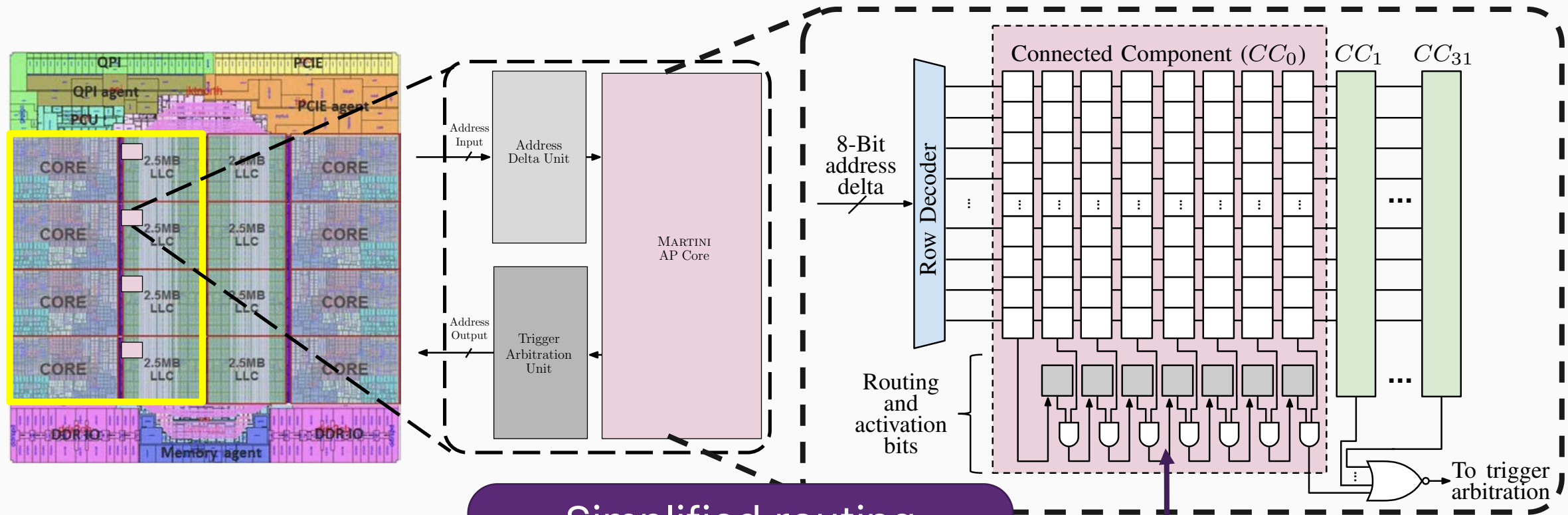
- For real-time monitoring, use a small hardware accelerator that executes automata
  - Embedded in the cache of the processor
- Dictionaries need simplification to embed in processor
  - Improve interaction with other security and system abstractions
  - Fit within hardware resource constraints
- Three primary techniques

# MARTINI: Detecting Attacks

- **$\Delta$ -windows:** to mitigate overfitting due to ASLR, we store the **difference between subsequent memory accesses**
- **Truncation:** to mitigate difference between physical and virtual addresses, we **truncate all deltas to 8 bits**
  - Mask selectable
  - Lower 7 + sign bit worked well in our experiments
- **Compression:** to reduce state space (hardware utilization), we **represent windows with unordered sets**
  - More permissive but smaller



# Where is MARTINI?



Simplified routing because automata are chains of 8 STEs

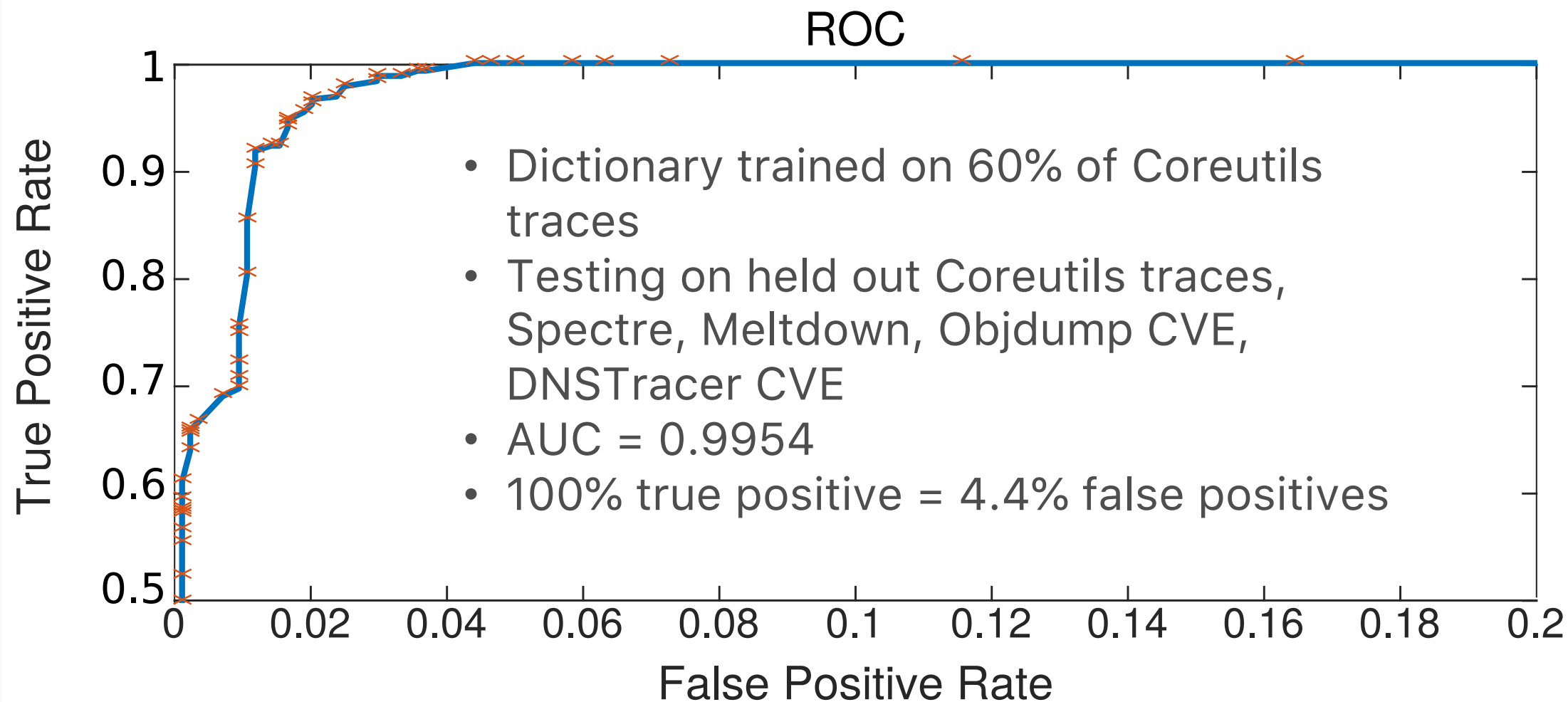
# Experimental Methodology

- Benchmark suite of real-world applications and exploits
  - GNU Coreutils programs
  - PARSEC benchmarks
  - Exploits: Spectre, Meltdown, Objdump CVE, DNSTracer CVE
- Use instrumented QEMU and Pin tools to collect traces
- 2,400 program traces and over 13 billion memory accesses
- Simulate MARTINI with custom version of VASim
  - Simulate address delta, automata core, and trigger arbitration units

# Guiding Research Questions

- Do sequences of memory accesses differentiate programs?
- Is MARTINI able to detect malicious inputs to trained programs?
- Can MARTINI detect anomalous programs, including recent hardware attacks?

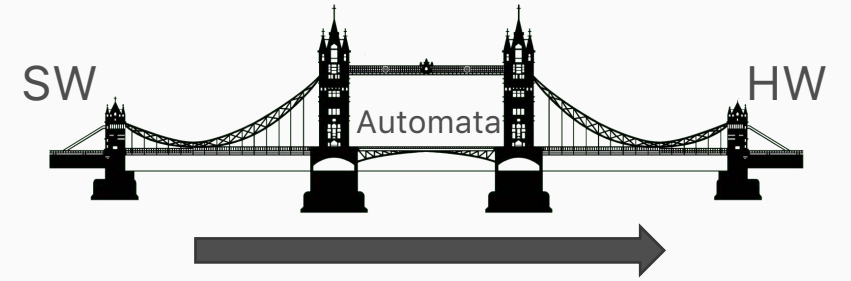
# MARTINI can Detect Attacks



# Architectural Support Summary

- Extend **expressive power** of current automata-derived architectures to support DPDA (and parsing)
- Custom 5-stage datapath for executing DPDA
  - 18.5x faster for parsing XML than state-of-the-art parsers
- Custom datapath for detecting security attacks
  - Based on sliding windows of abstracted memory accesses
  - Can detect recent hardware exploits (e.g., Spectre, and Meltdown)
- Architectures provide **scalability**, and **performance** for new application domains

# Dissertation Overview



- Research Contributions
  - Acceleration of Legacy Code
  - High-Level Programming Language: RAPID
  - High-Speed, Interactive Debugger for Hardware Accelerators
  - Hardware Support for New Application Domains:
    - In-Cache Accelerator for Parsing
    - In-Cache Hardware Unit for Detecting Security Attacks
- Broader Impact and Mentorship
- Conclusions / Discussion

# Broader Impact

- **Tools to Promote Adoption:** prototype tools used in this dissertation are (being) released as part of MNCaRT, an end-to-end automata processing ecosystem
  - AutomataSynth, RAPID Compiler, DPDA compiler, MNRL state machine language, VASim (+DPDA)
  - Appendix A
- **Undergraduate Mentorship and Teaching:** training the next generation of researchers and introducing students to the beauty of automata

# Teaching and Mentorship

- **Automata-based Instruction:** lecture on string algorithms (regex) in DS+A; lecture on language design in grad/undergrad PL; guest lecture on automata processing in grad architecture; guest lecture on accelerator debugging in grad PL
- **Undergraduate Research Projects:**
  - ‡MARTINI (Yujun Qin, Samuel Gonzalez, Linh Le)
  - †Debugging (Matthew Casias, UVA)
  - ‡Automata-based file carving and disk damage modeling (Ian Bertram, Michael Flanagan, Aniruddh Agarwal)
  - Automata-based surface detection (Emma Fass, Luke Merrick, Joe Tidwell, UVA)
  - ‡Diversity in undergrad CS (Fee Christoph)
  - †Quadcopter Security (Kate Highnam, UVA)

†Peer-Reviewed Publication

‡In-Flight Publication



# Proposed Research (2018)

- Four components to improve programming support for hardware accelerators using automata abstractions
  - High-level programming language (RAPID)
  - High-speed, interactive debugging for RAPID on AP and FPGA
  - In-cache accelerators
    - For pushdown automata (ASPEN)
    - For detection of security attacks
  - Adapt legacy kernels for execution on hardware accelerators
- Evaluation w.r.t. **Performance & scalability, ease of use, expressive power, and legacy support**

# Publications Supporting Contributions

1. **Kevin Angstadt**, Jean-Baptiste Jeannin, and Westley Weimer. Accelerating Legacy String Kernels via Bounded Automata Learning. In *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems*, Lausanne, Switzerland, 2020. ACM, to appear.
2. Matthew Casias, **Kevin Angstadt**, Tommy Tracy II, Kevin Skadron, and Westley Weimer. Debugging Support for Pattern-Matching Languages and Accelerators. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*, Providence, Rhode Island, 2019. ACM. (21% acceptance rate)
3. **Kevin Angstadt**, Jack Wadden, Westley Weimer, and Kevin Skadron. Portable Programming with RAPID. In *Transactions on Parallel and Distributed Systems*, to appear. IEEE. (4.181 journal impact factor)
4. **Kevin Angstadt**, Arun Subramaniyan, Elaheh Sadredini, Reza Rahimi, Kevin Skadron, Westley Weimer, Reetuparna Das. ASPEN: A Scalable In-SRAM Architecture for Pushdown Automata. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, Fukuoka, Japan. 2018. IEEE. (21% acceptance rate)
5. **Kevin Angstadt**, Jack Wadden, Vinh Dang, Ted Xie, Dan Kramp, Westley Weimer, Mircea Stan, and Kevin Skadron. MNCaRT: An Open-Source, Multi-Architecture Automata-Processing Research and Execution Ecosystem. In *Computer Architecture Letters*, vol. 17, no. 1, pp. 84-87, Jan.-June 1 2018. IEEE. (~24% acceptance rate)
6. **Kevin Angstadt**, Westley Weimer, and Kevin Skadron. RAPID Programming of Pattern-Recognition Processors. In *Proceedings of the 21st International Conference on Architectural Support for programming Languages and Operating Systems*, Atlanta, Georgia, 2016. ACM. (22% acceptance rate)
7. (*Under review*) Yujun Qin, Samuel Gonzalez, **Kevin Angstadt**, Xiaowei Wang, Stephanie Forrest, Reetuparna Das, Kevin Leach, and Westley Weimer. MARTINI: Memory Access Traces to Detect Attacks.

Undergraduate collaborators are underlined

# Additional Publications

8. Jack Wadden, **Kevin Angstadt**, and Kevin Skadron. Characterizing and Mitigating Output Reporting Bottlenecks in Spatial Automata Processing Architectures. In *Proceedings of the 24th IEEE International Symposium on High-Performance Computer Architecture*, Vienna, Austria, 2018. IEEE. (21% acceptance rate)
9. **Kevin Angstadt** and Ed Harcourt. A Virtual Machine Model for Accelerating Relational Database Joins using a General Purpose GPU. In *Proceedings of the High Performance Computing Symposium*, Alexandria, VA, 2015. Society for Computer Simulation International.
10. Yu Huang, **Kevin Angstadt**, Kevin Leach, and Westley Weimer. Selective Symbolic Type-Guided Checkpointing and Restoration for Autonomous Vehicle Repair. In *Proceedings of the 1st International Workshop on Automated Program Repair*, Seoul, South Korea, 2020. To appear.
11. Sihang Liu, **Kevin Angstadt**, Mike Ferdman, Samira Khan. ARMOR: Towards Restricted Approximation with a Worst-Case Guarantee. In: *Proceedings of the 2018 Workshop on Approximate Computing Across the Stack*, Williamsburg, VA, 2018.
12. Kate Highnam, **Kevin Angstadt**, Kevin Leach, Westley Weimer, Aaron Paulos, and Patrick Hurley. An Uncrewed Aerial Vehicle Attack Scenario and Trustworthy Repair Architecture. In *Proceedings of the 46th International Conference on Dependable Systems and Networks*, Industrial Track, Toulouse, France, 2016. IEEE.
13. (*In Preparation*) Kevin Leach, **Kevin Angstadt**, Anh Nguyen-Tuong, Christopher S. Timperley, Aaron Paulos, Zech Bertilson, Partha Pal, Christopher Hall, Jacob Wende, Padraic Cashin, Stephanie Forrest, Claire Le Goues, Jack W. Davidson, Westley Weimer, Patrick Hurley, and Carl Thomas. A Framework for Trusted and Resilient Autonomous Vehicles.

## Invited Papers and Tech Reports

14. Ke Wang, **Kevin Angstadt**, Chunkun Bo, Nathan Brunelle, Elaheh Sadredini, Tommy Tracy, II, Jack Wadden, Mircea Stan, and Kevin Skadron. An overview of Micron's Automata Processor. In *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, Pittsburgh, PA, 2016. ACM.
15. **Kevin Angstadt**, Jack Wadden, Westley Weimer, and Kevin Skadron. MNRL and MNCaRT: An Open-Source, Multi-Architecture State Machine Research and Execution Ecosystem. Technical Report CS-2017-01, Department of Computer Science, University of Virginia, May 2017.

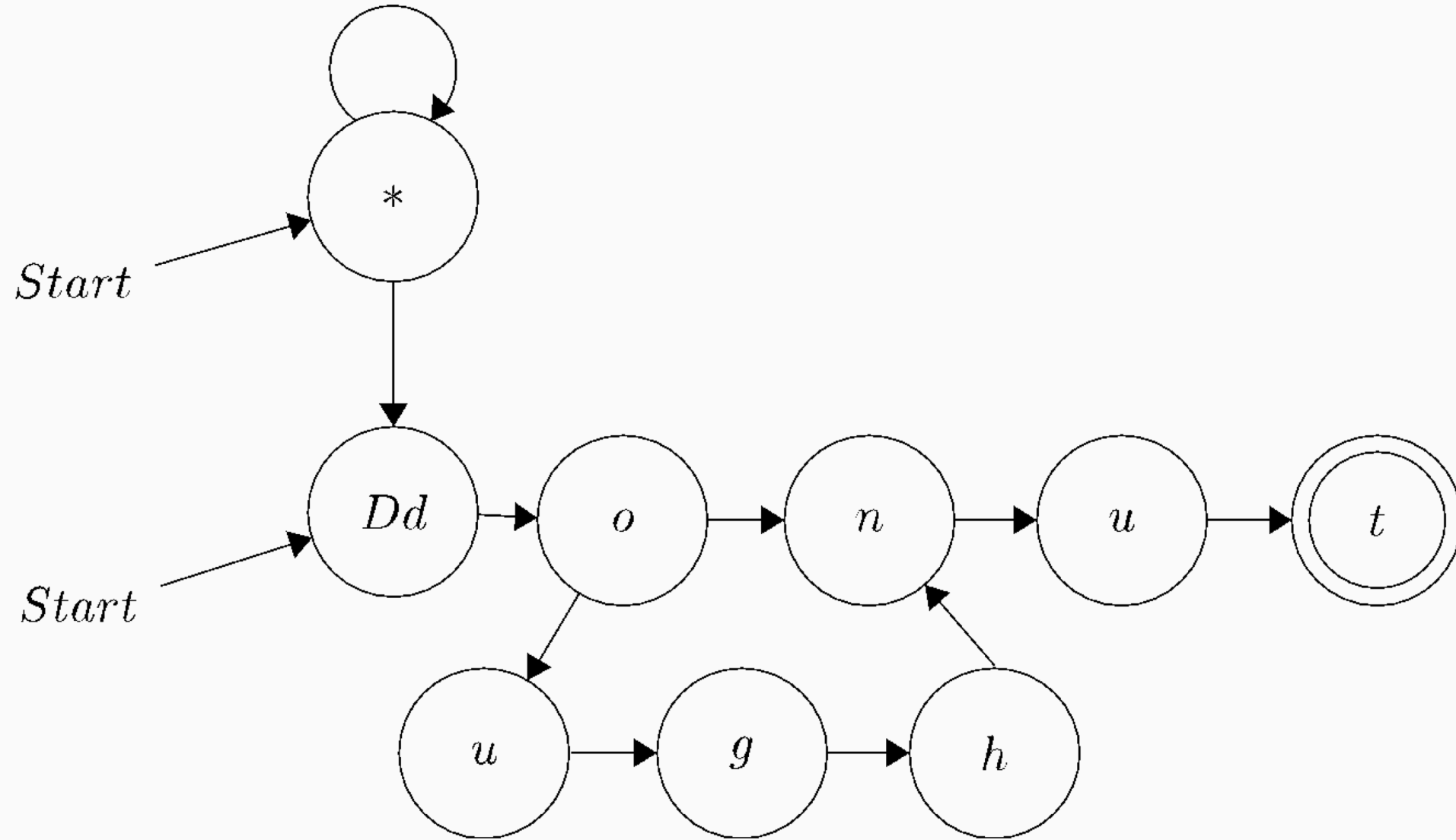
Undergraduate collaborators are underlined

# Dissertation Summary

- Hardware accelerators more commonplace—need for programming models and maintenance tools
- Using finite automata as an abstraction, we developed a programming model that provides **performance & scalability, ease of use, expressive power, and legacy support**
  - AutomataSynth: porting legacy code to FPGAs
  - RAPID: writing new pattern-searching programs for hardware accelerators
  - High-speed, FPGA-based debugger for RAPID programs
  - Two in-cache accelerators for new applications
    - Parsing of XML and detecting security attacks

# Bonus Slides

(Excerpts)



# What is a Program?

**Task:** Blink a lightbulb



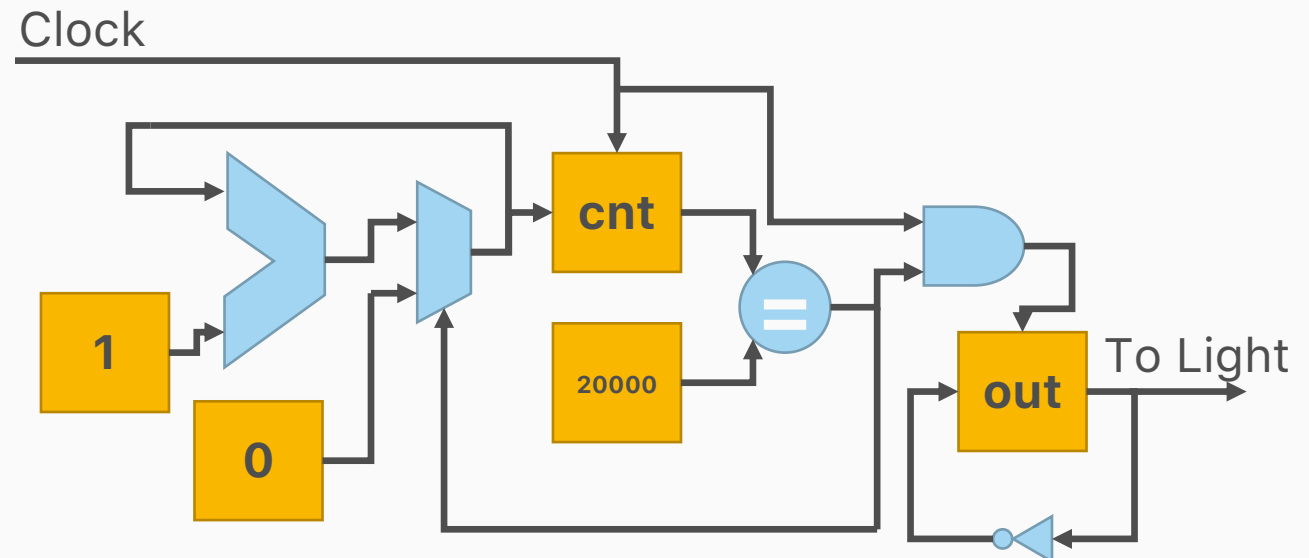
**CPU (Python)**

```
import gpiozero
import time

led = gpiozero.LED(14)

while True:
    led.on()
    time.sleep(1)
    led.off()
    time.sleep(1)
```

**FPGA (Circuit)**



Generally written in Verilog or VHDL language

# Why Automata(Synth)?

- FPGA designs are often described in terms of state machines
- Automata a versatile and broadly-applicable
- Can build on significant research effort accelerating state machine execution
- Other high-level approaches (cf. HLS) generally fail to abstract low-level architectural details
- Our approach **decouples** high-level program and low-level implementation



# Reasoning About Strings

String solver should support the following:

- Unbounded string length
- Regular expression-based constraints over strings
- Access to individual characters of strings
- Comparison of individual characters and strings
- Reasoning about the length of strings
- Comparison between strings and bitvectors (treat characters as numbers)\*\*\*
- Ability to generate strings that satisfy a set of constraints

Mirrors low-level string manipulation in C

\*\*\*Not currently supported

# Example RAPID Program

If all symbols in item set match, increment counter

Spawn parallel computation for each item set

Sliding window search calls *frequent* on every input

Trigger *report* if threshold reached

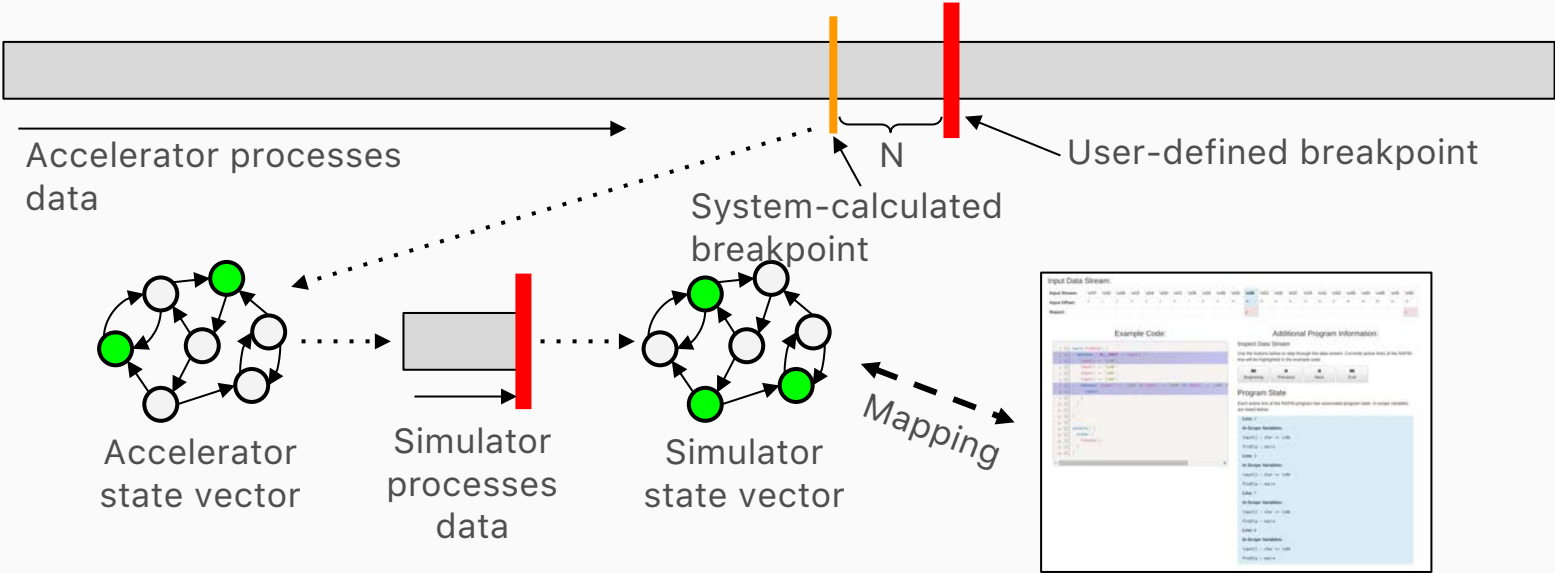
```
macro frequent (String set, Counter cnt) {  
  foreach(char c : set) {  
    while(input() != c);  
  }  
  cnt.count();  
}  
  
network (String[] set) {  
  some(String s : set) {  
    Counter cnt;  
    whenever(START_OF_INPUT == input())  
      frequent(s, cnt);  
    if (cnt > 128)  
      report;  
  }  
}
```

# Putting it all together

## Standard Program Execution



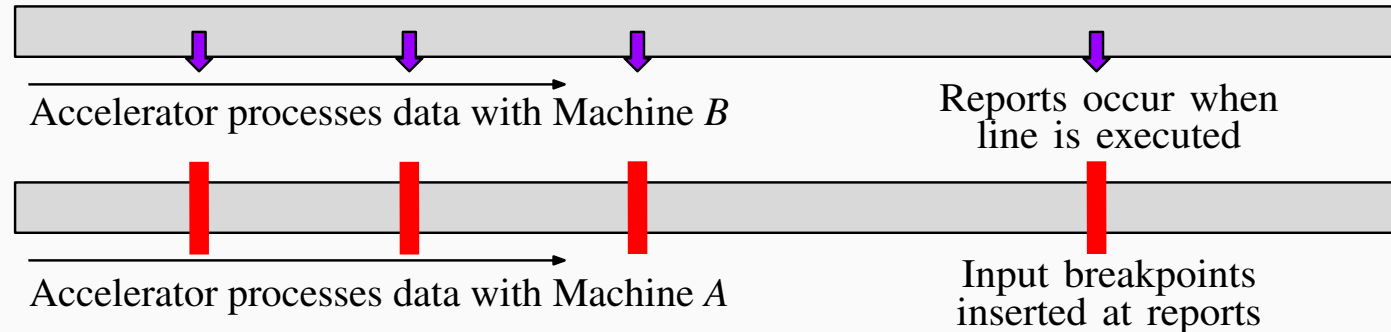
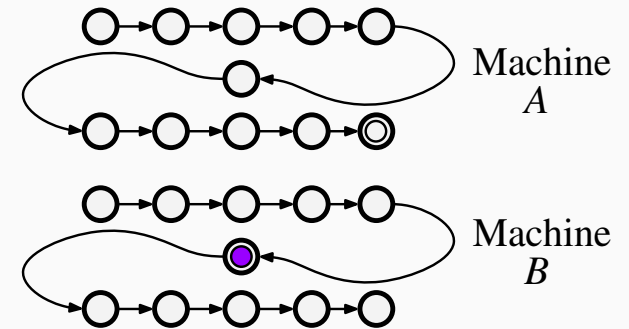
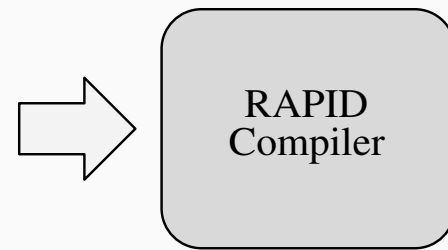
## Debugging Execution



# Traditional Breakpoints

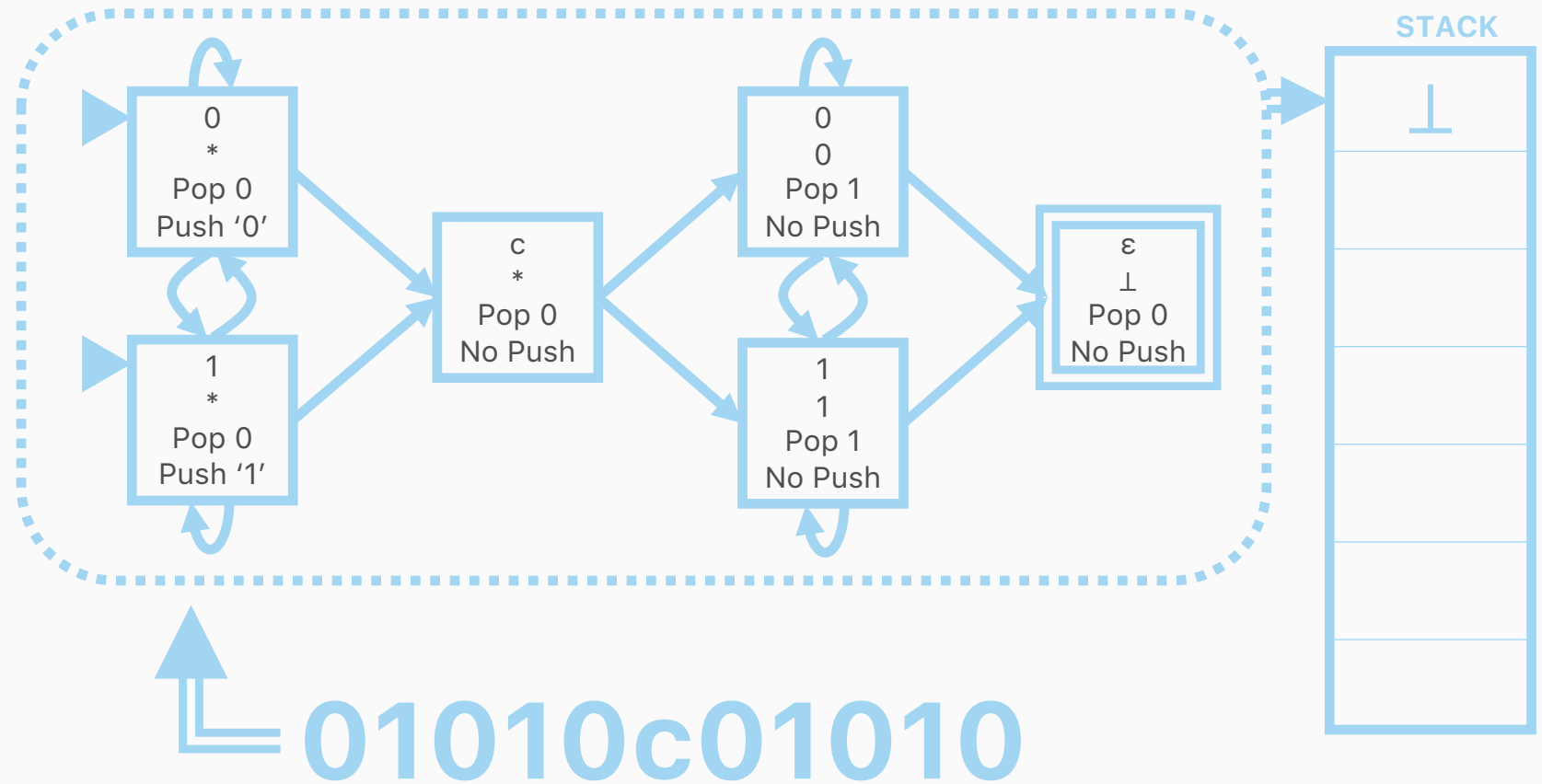
## RAPID Program

```
macro helloWorld() {  
  whenever( ALL_INPUT == input() ) {  
    foreach(char c : "Hello") {  
      c == input();  
    }  
    * input() == ' ';  
    foreach(char c : "world") {  
      c == input();  
    }  
    report;  
  }  
}  
  
network() {  
  helloWorld();  
}
```



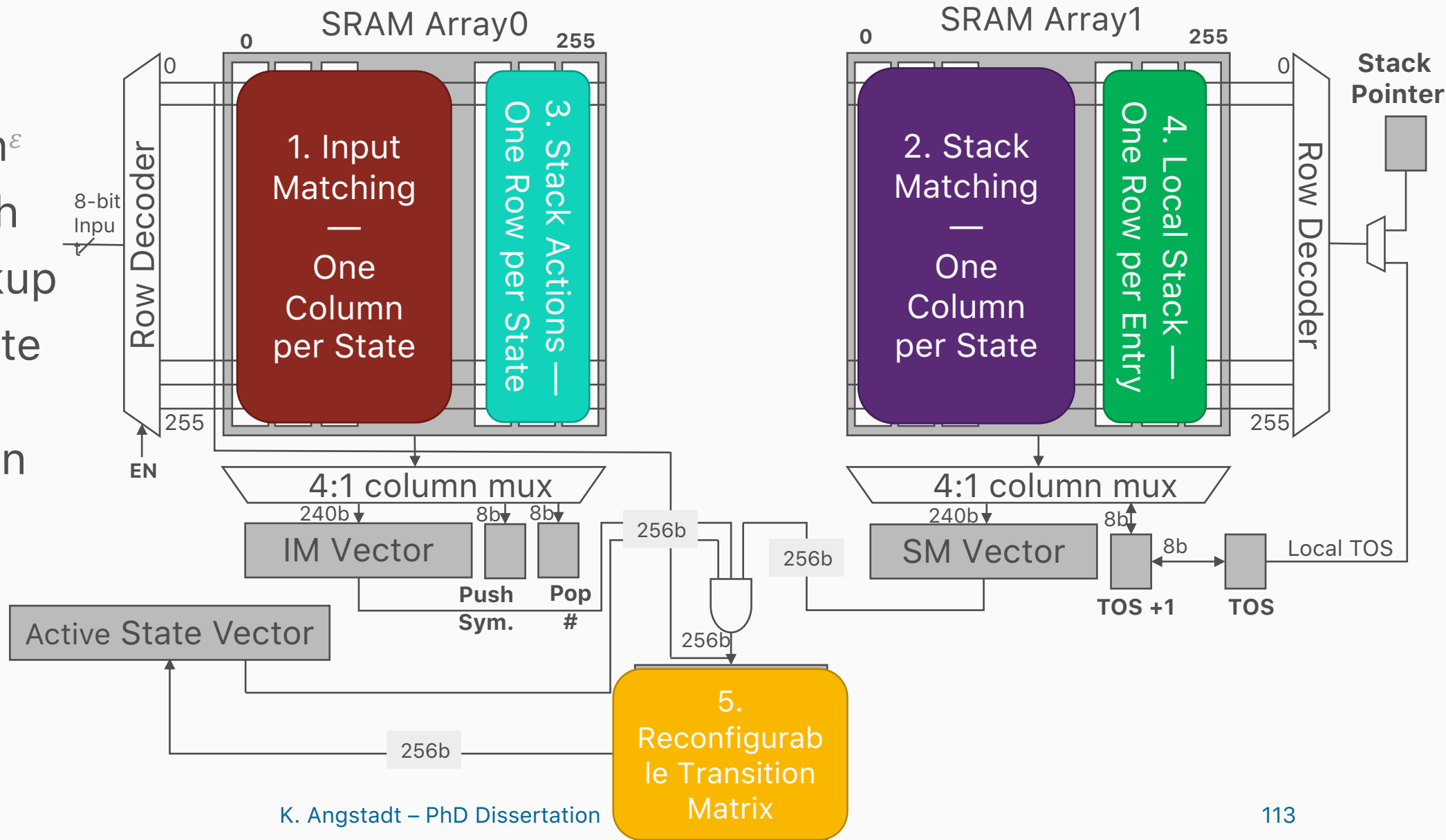
# Five Steps of DPDA Execution Per Cycle

1. Input Match $\epsilon$
2. Stack Match
3. Action Lookup
4. Stack Update
5. State Transition



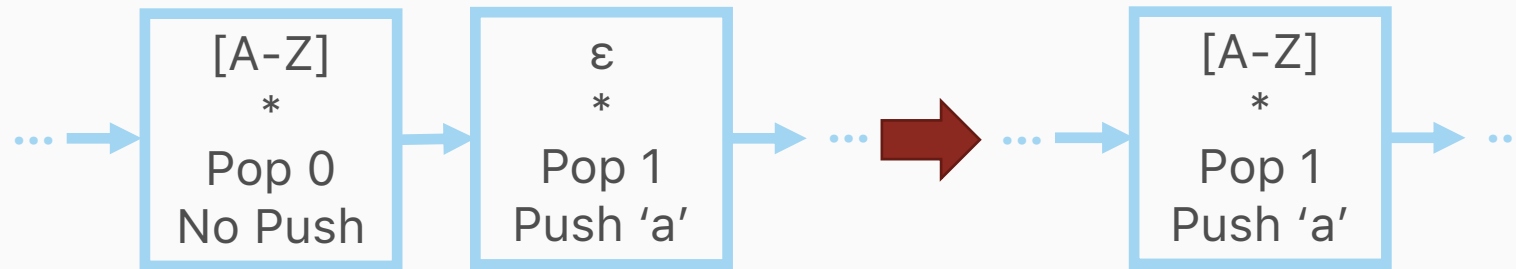
# ASPEN Datapath — 240 States in 2 SRAM Arrays

1. Input Match<sup>ε</sup>
2. Stack Match
3. Action Lookup
4. Stack Update
5. State Transition



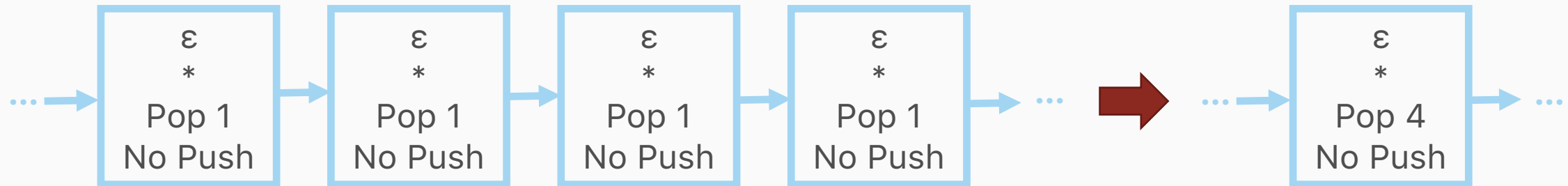
# Optimizations

## Epsilon Merging



**Goal:** Reduce the number of stalls while processing input

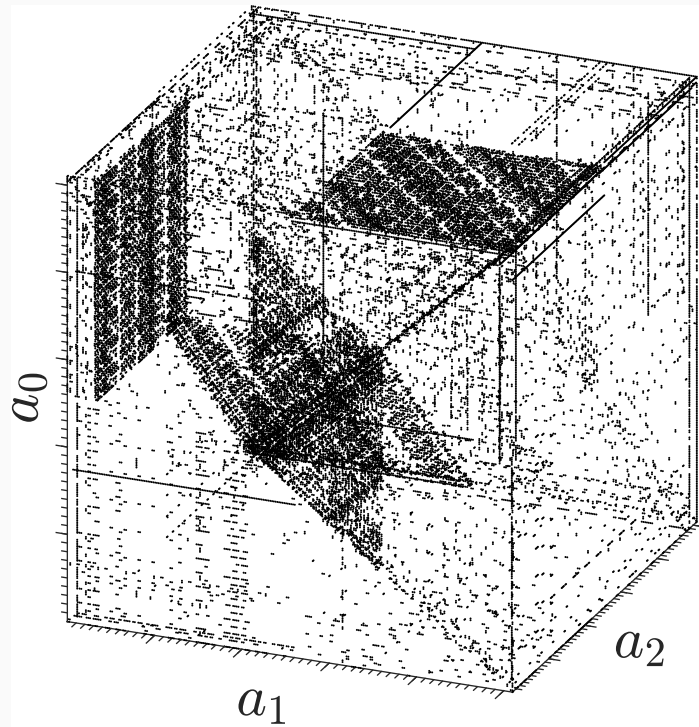
## Multipop



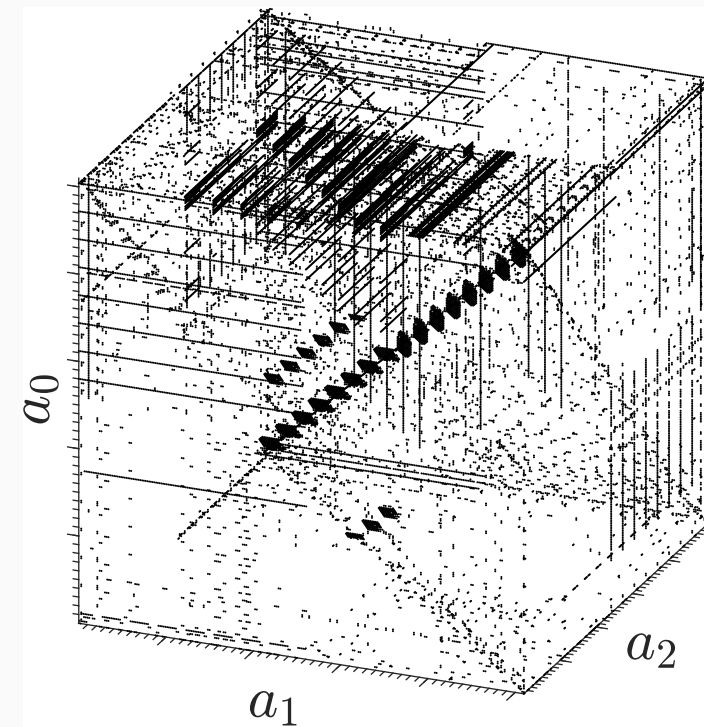
- **Average of 65% reduction** in epsilon states

# Differentiating Programs

Each point represents a sequence of three addresses accessed by the program



cal — displays a calendar

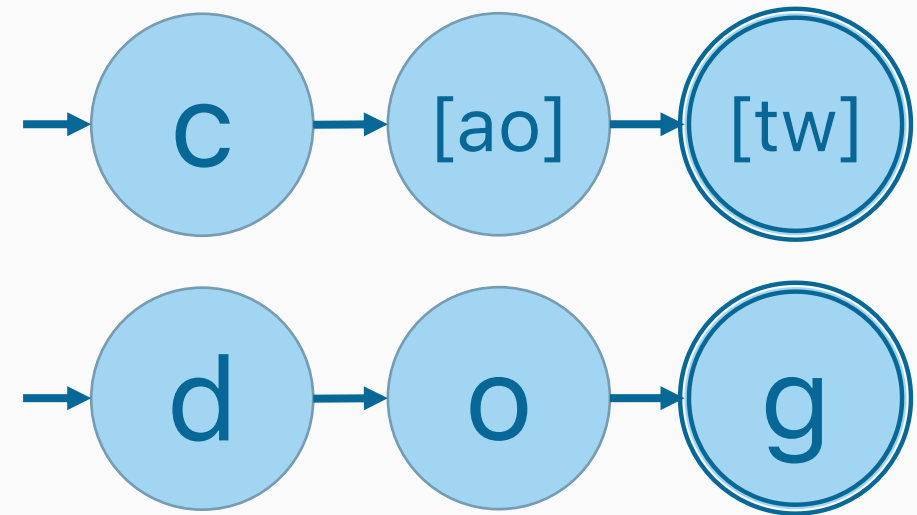
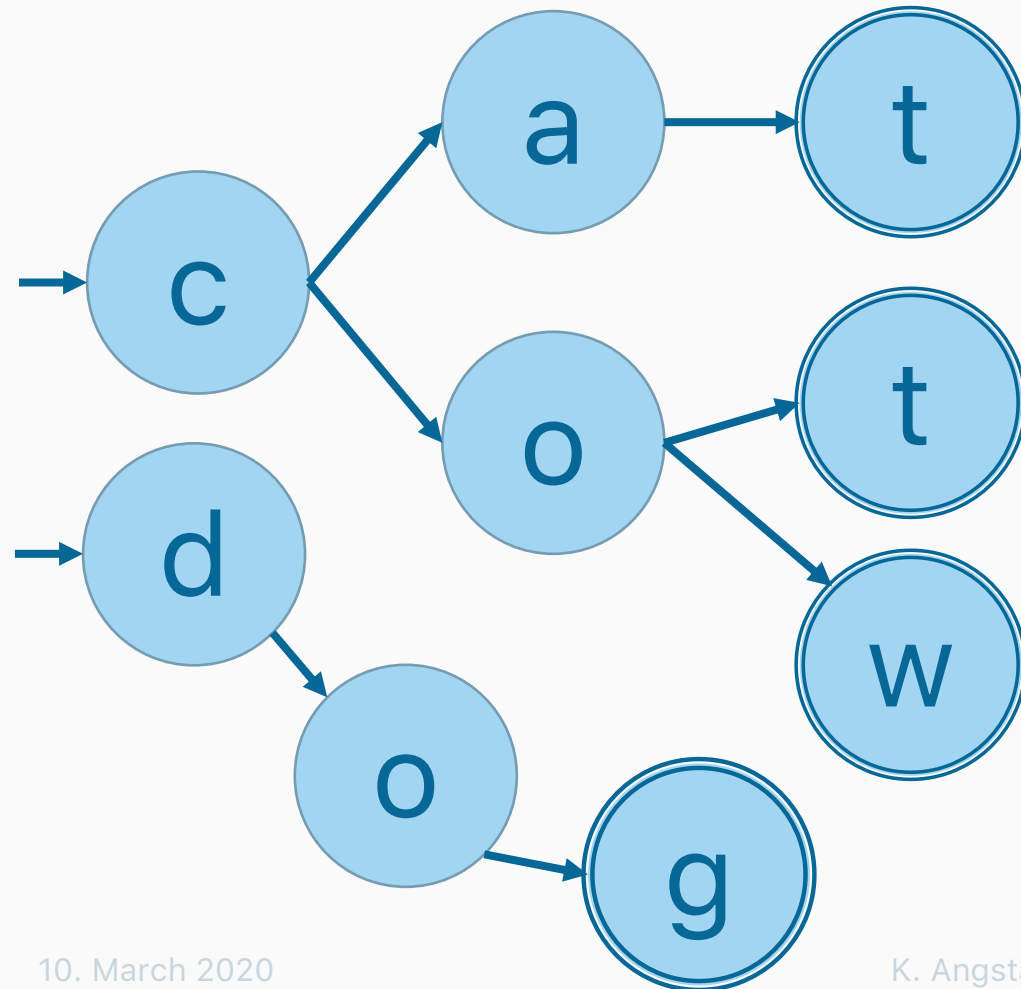


dmesg — displays system message buffer



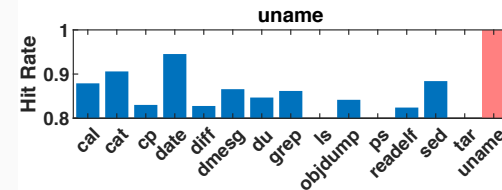
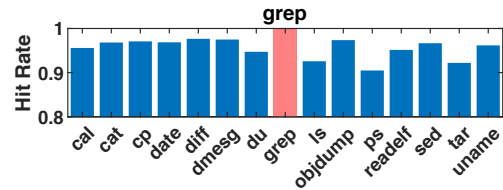
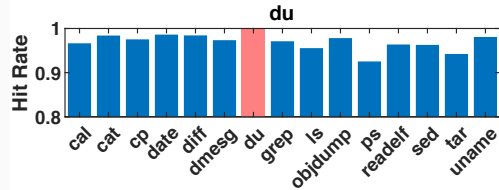
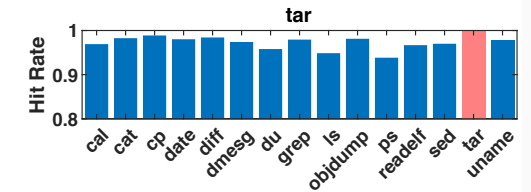
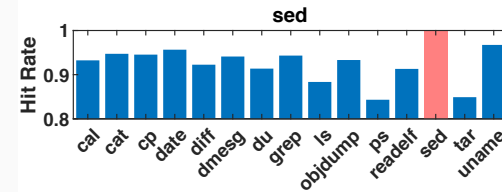
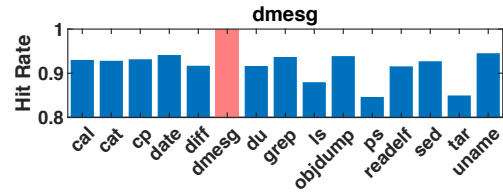
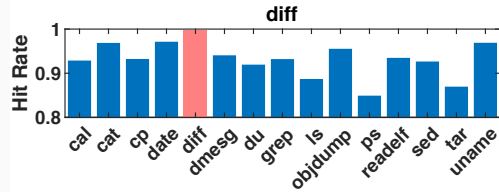
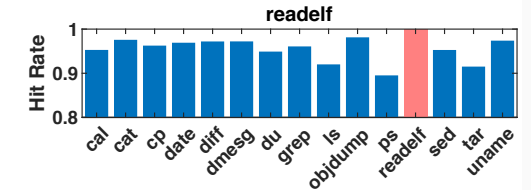
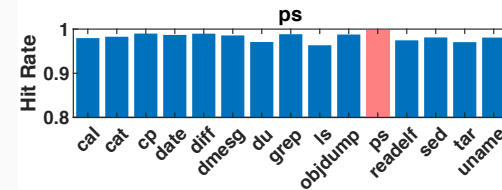
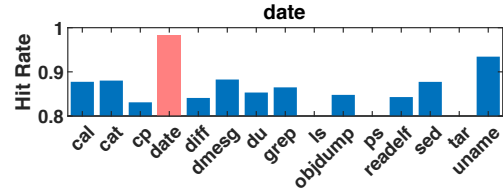
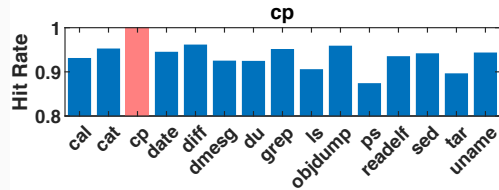
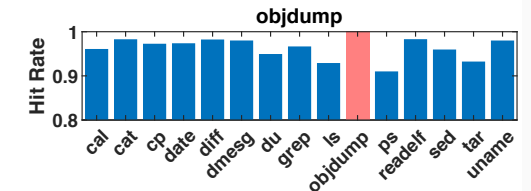
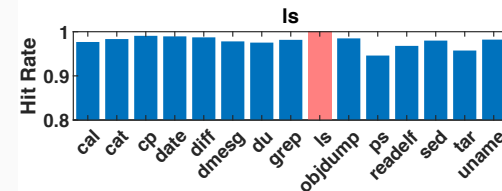
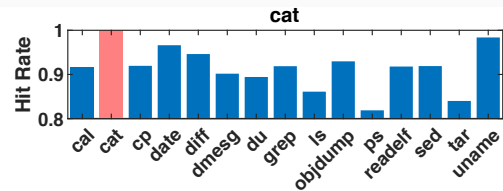
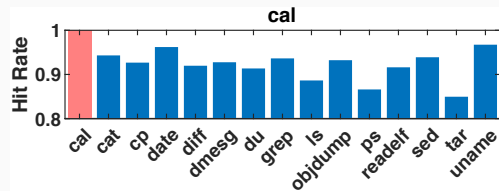
# Compression Example

Dictionary contains:  $\langle c,a,t \rangle$ ,  $\langle c,o,t \rangle$ ,  $\langle c,o,w \rangle$ , and  $\langle d,o,g \rangle$

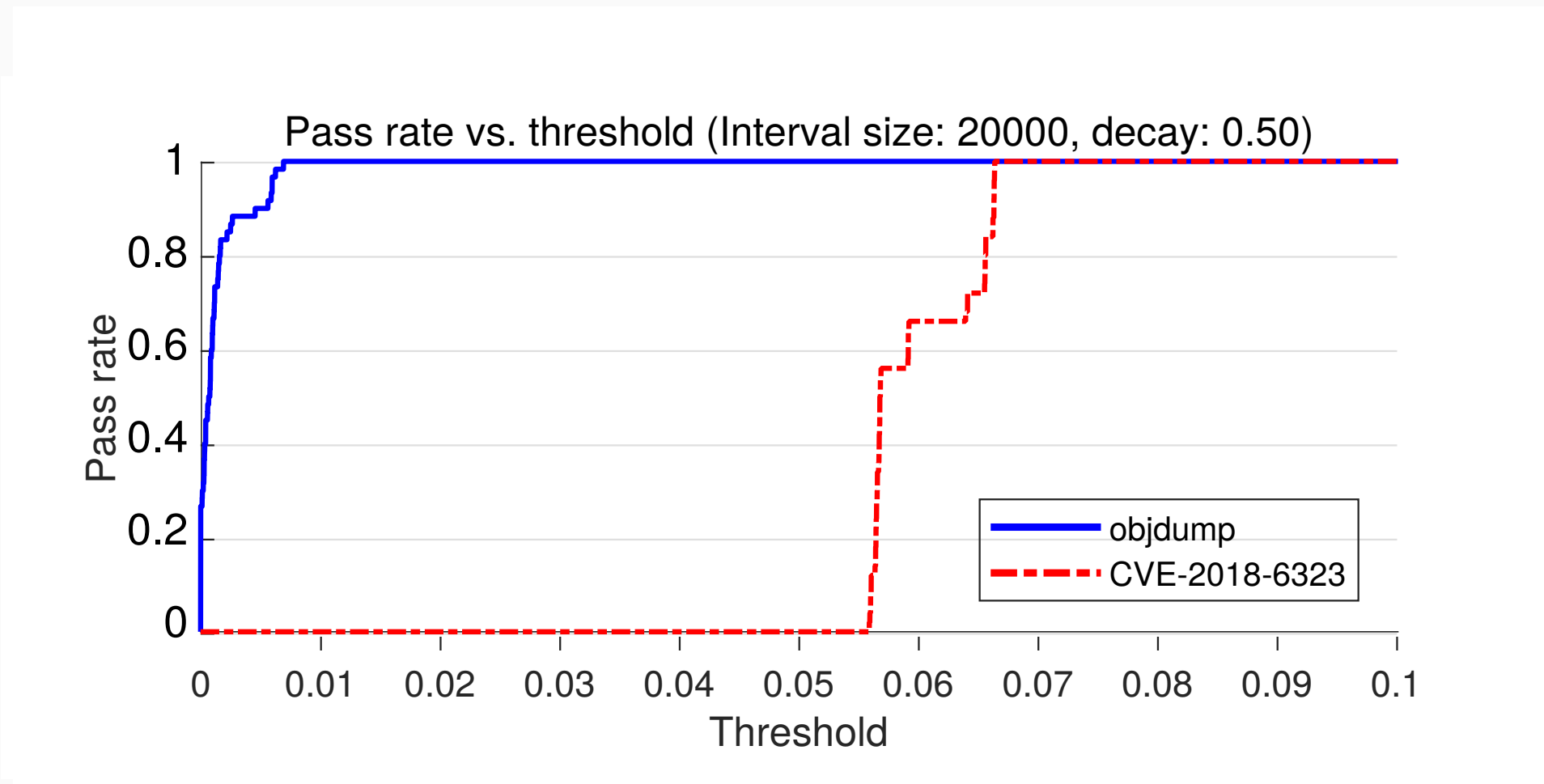


- Sacrifices accuracy for space
  - Now accepts  $\langle c,a,w \rangle$
  - 9 to 6 states
- In limit, reduces  $2^{64}$  states to 2,048 states

# MARTINI can Differentiate Programs



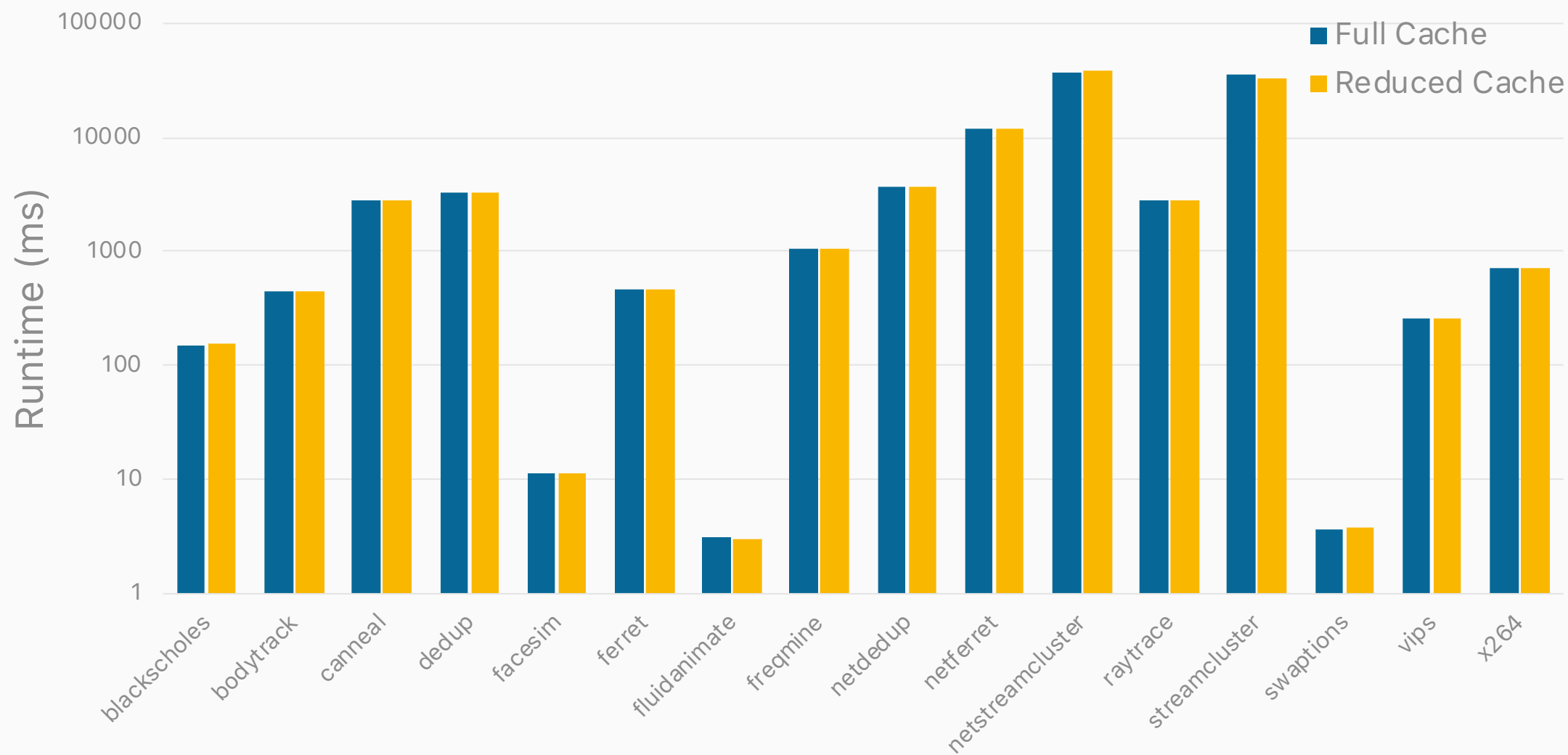
# MARTINI can Detect Anomalous Inputs



# Meta-Evaluation: System Performance

- Both ASPEN and MARTINI consume a portion of LLC
- What does this do to total system performance?
- **Platform:** Ubuntu 16.04, 192 GB RAM, 2 Intel Xeon Platinum 8275CL CPUs (36 cores, each) @ 3 GHz, 36 MB LLC per processor, subdivided into 11 ways
- **Experiment:** execute PARSEC benchmarks 40 times and measure wall clock time (20 with full cache, 20 with reduced cache)
  - Reduce by 1 way to simulate MARTINI footprint

# Runtime Comparison with Reduced Cache



# Runtime Comparison with Reduced Cache

