RAPID Programming of Pattern-Recognition Processors

Kevin Angstadt Westley Weimer Kevin Skadron
Department of Computer Science
University of Virginia
{**angstadt**, weimer, skadron}@cs.virginia.edu

19. August 2016

THE CENTER FOR AUTOMATA PROCESSING
UNIVERSITY of VIRGINIA

Finding Needles in a Haystack

- Researchers and companies are collecting increasing amounts of data
- 44x data production in 2020 than in 2009[†]
- Demand for real-time analysis of collected data[‡]

[†] Computer Sciences Corporation. Big data universe beginning to explode. 2012

[‡] Capgemini. Big & fast data: The rise of insight- driven business. 2015.

What is the common theme?

Locate the most
probab
DNA
hu

Find pr
mo
purc

Parse English text to
ident
reco
d

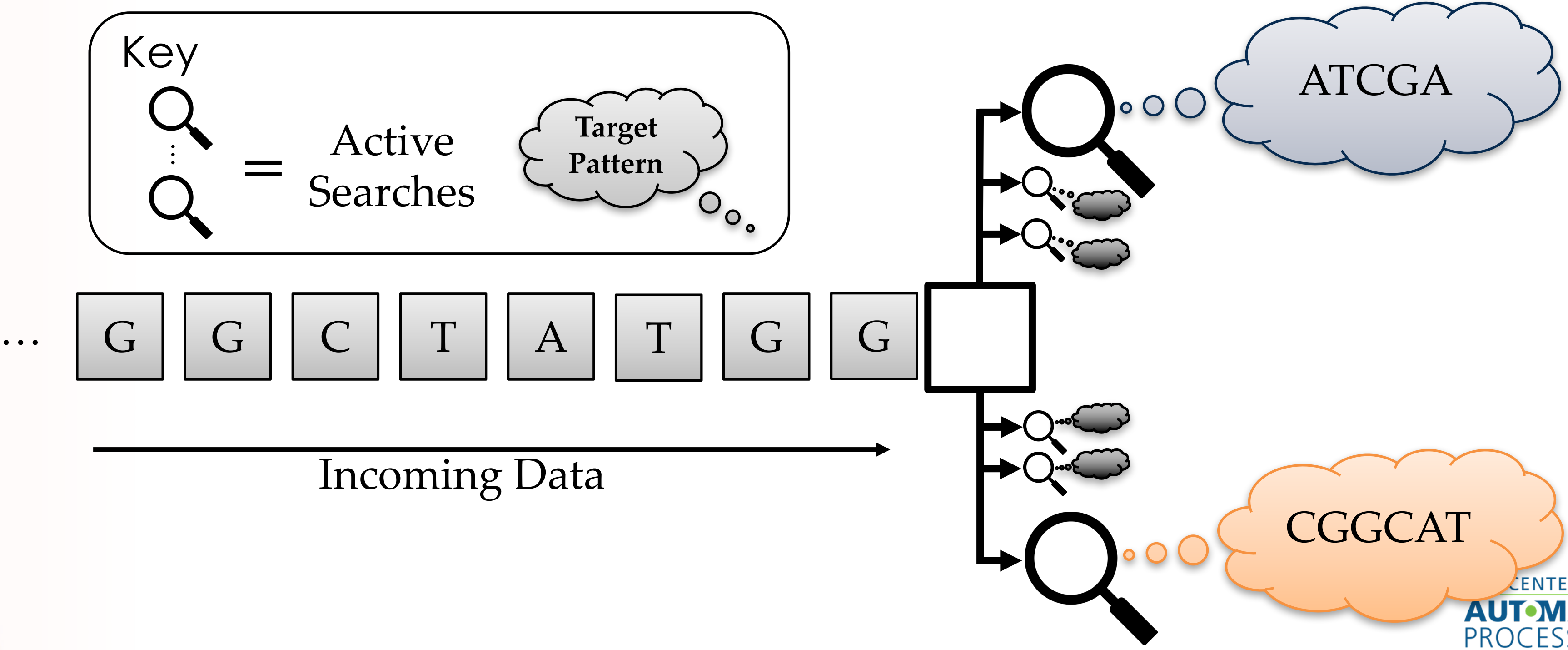
Identify consumer
sentime
social

Search for Higgs events based off on paths of subatomic particles

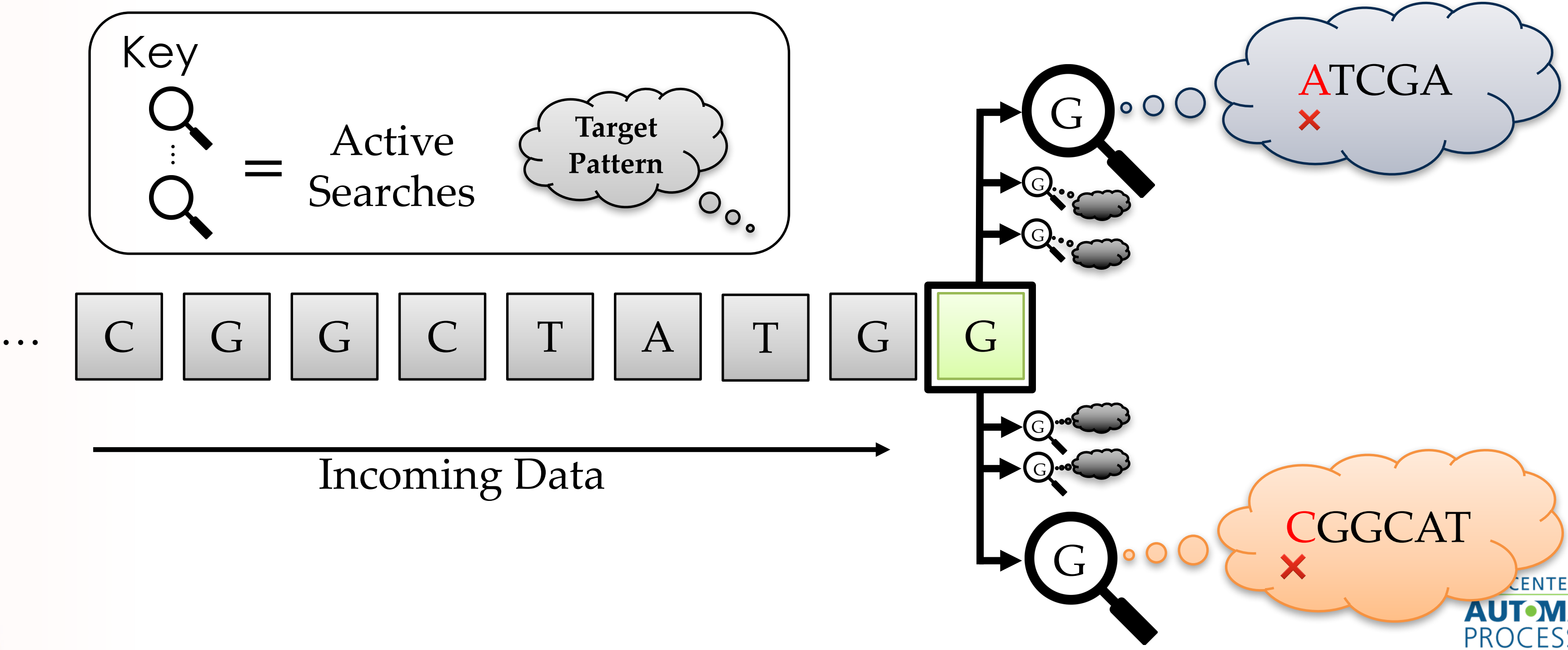
Pattern Search Problems

THE CENTER FOR AUTOMATA PROCESSING

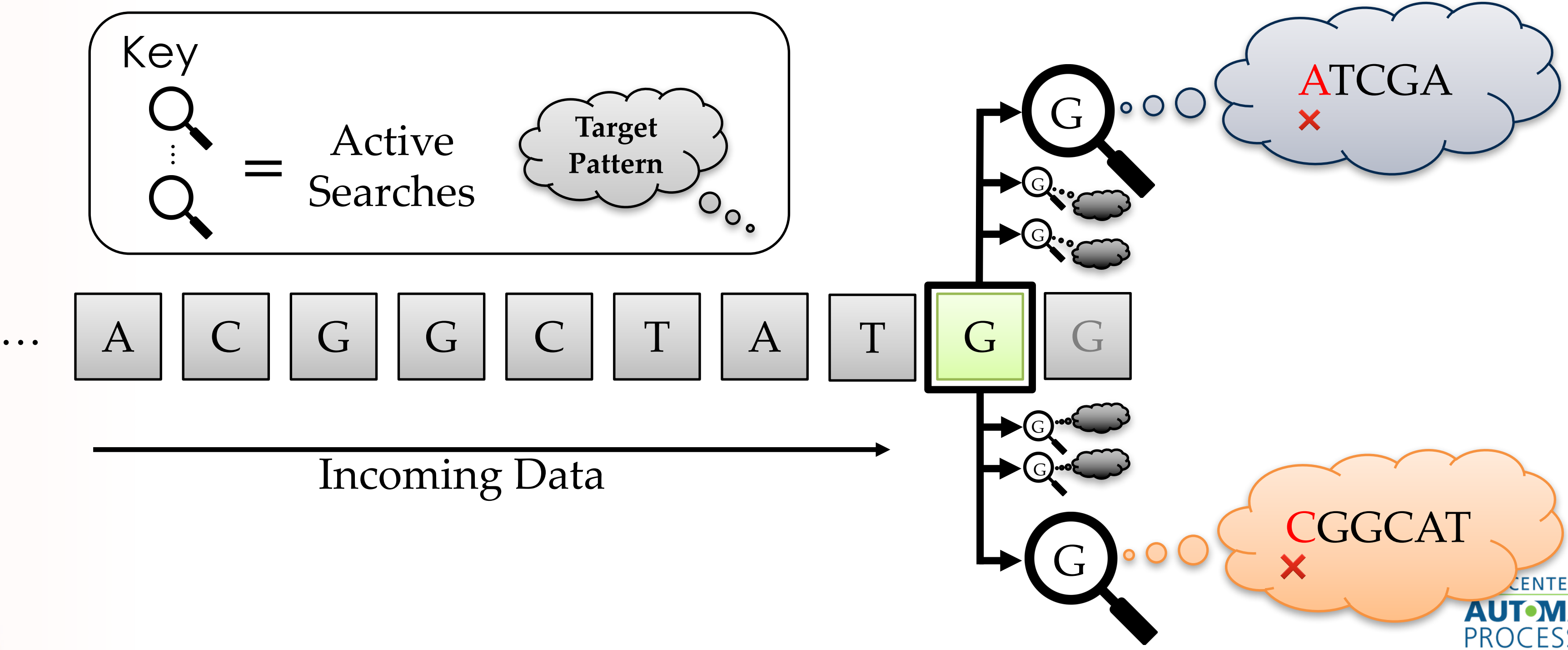
Parallel searches



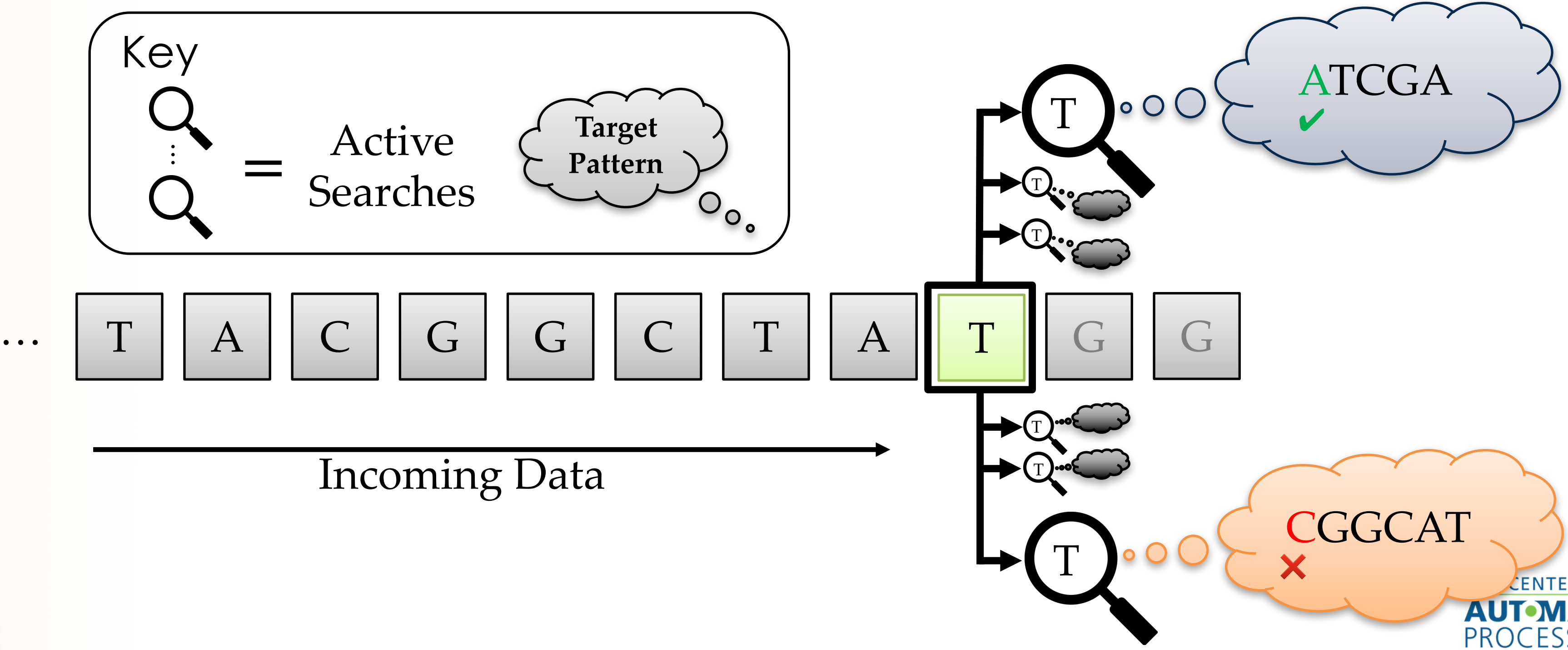
Parallel searches



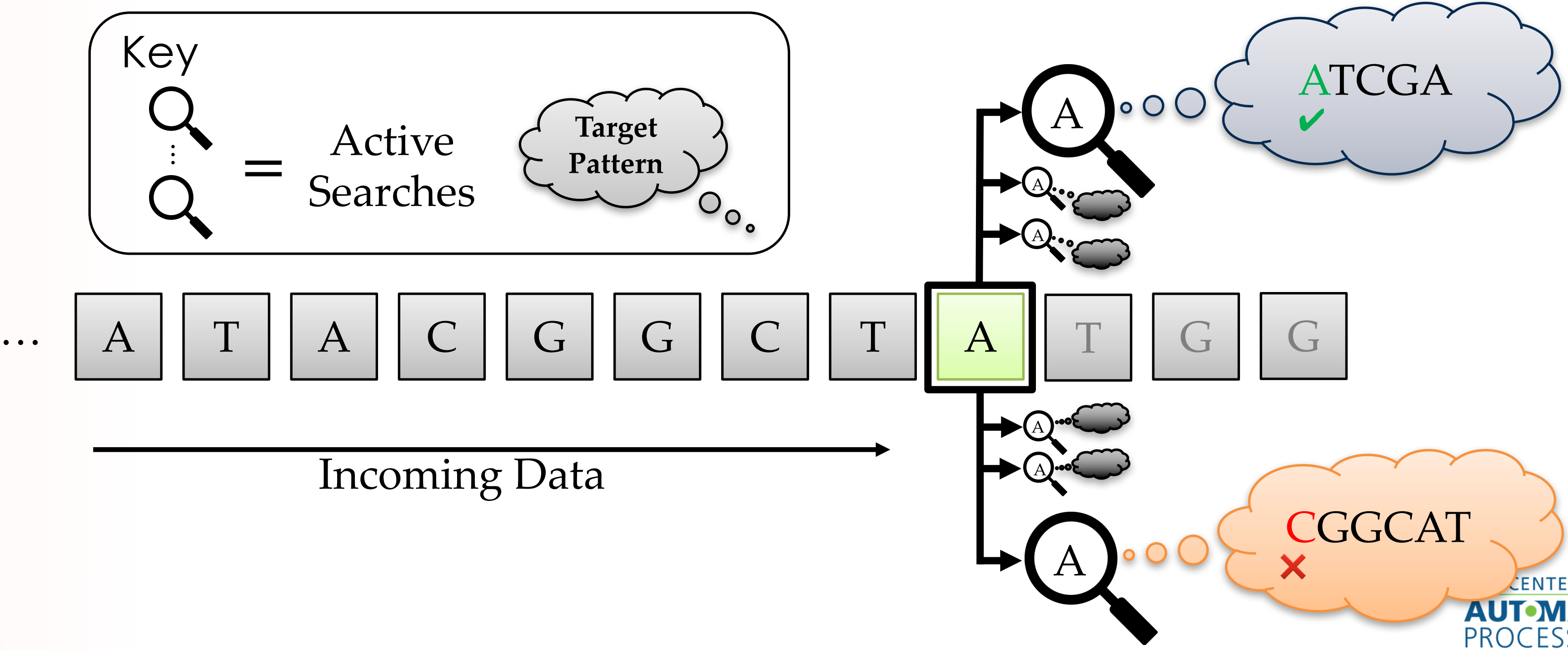
Parallel searches



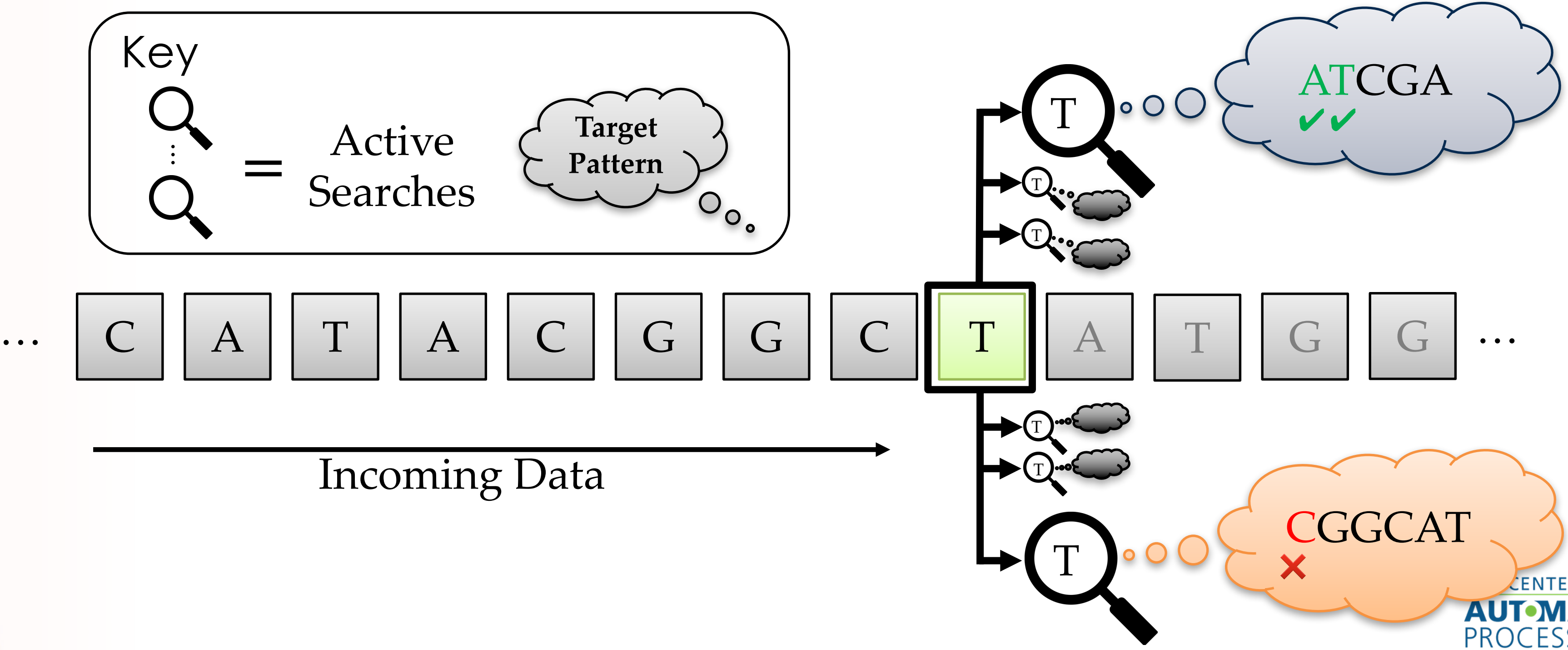
Parallel searches



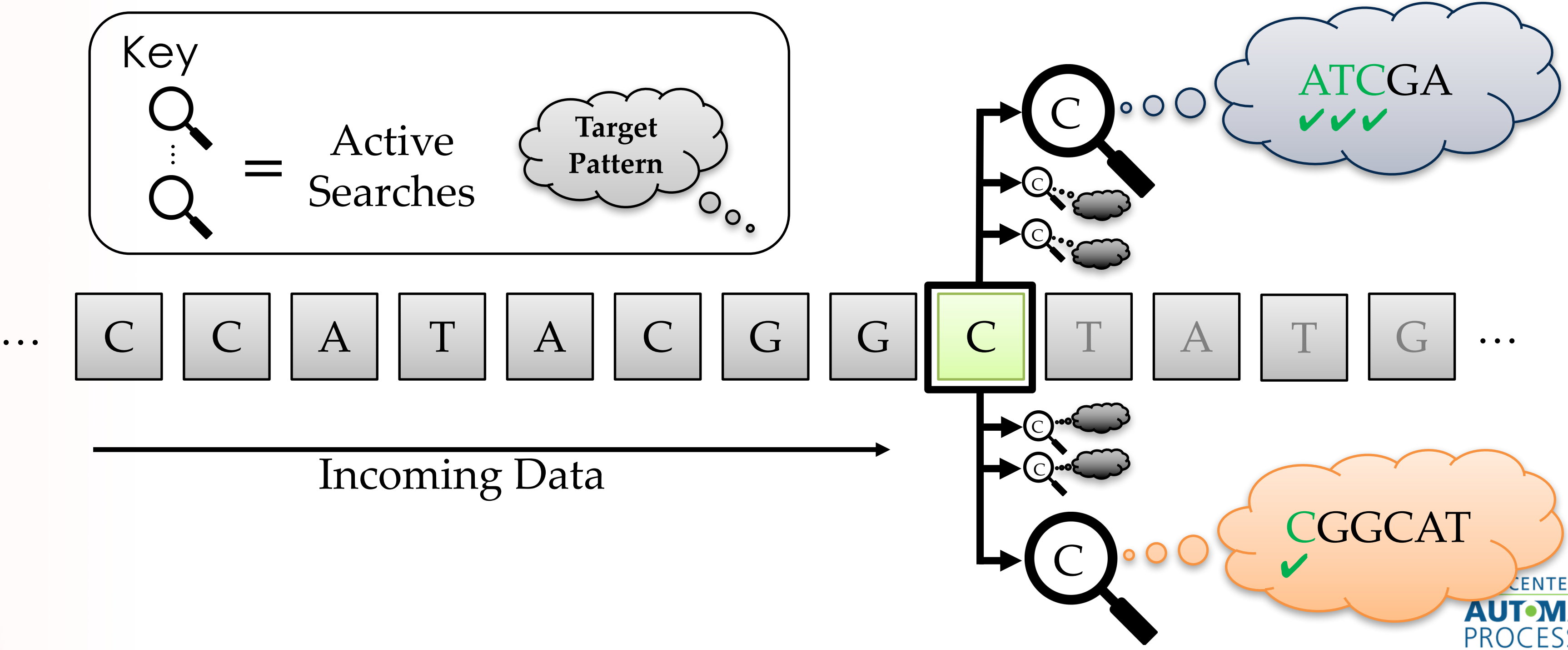
Parallel searches



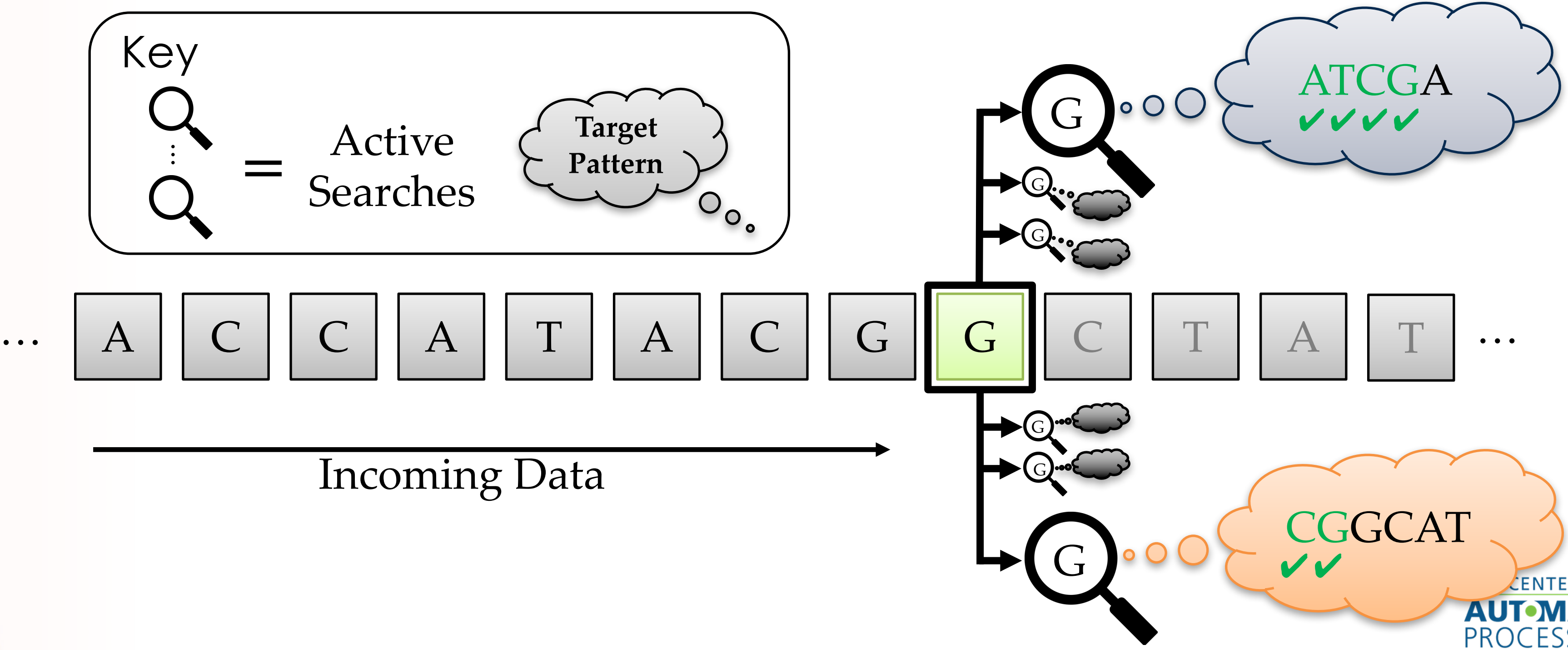
Parallel searches



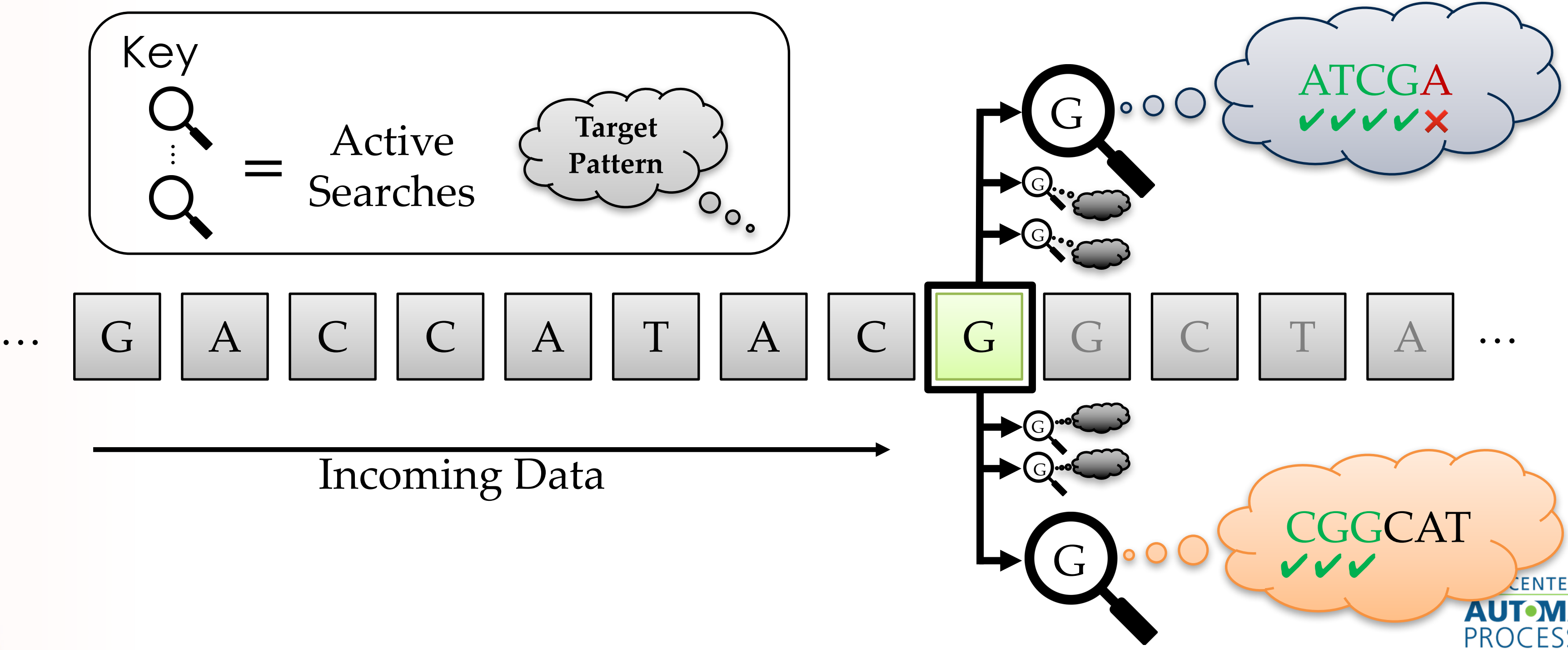
Parallel searches



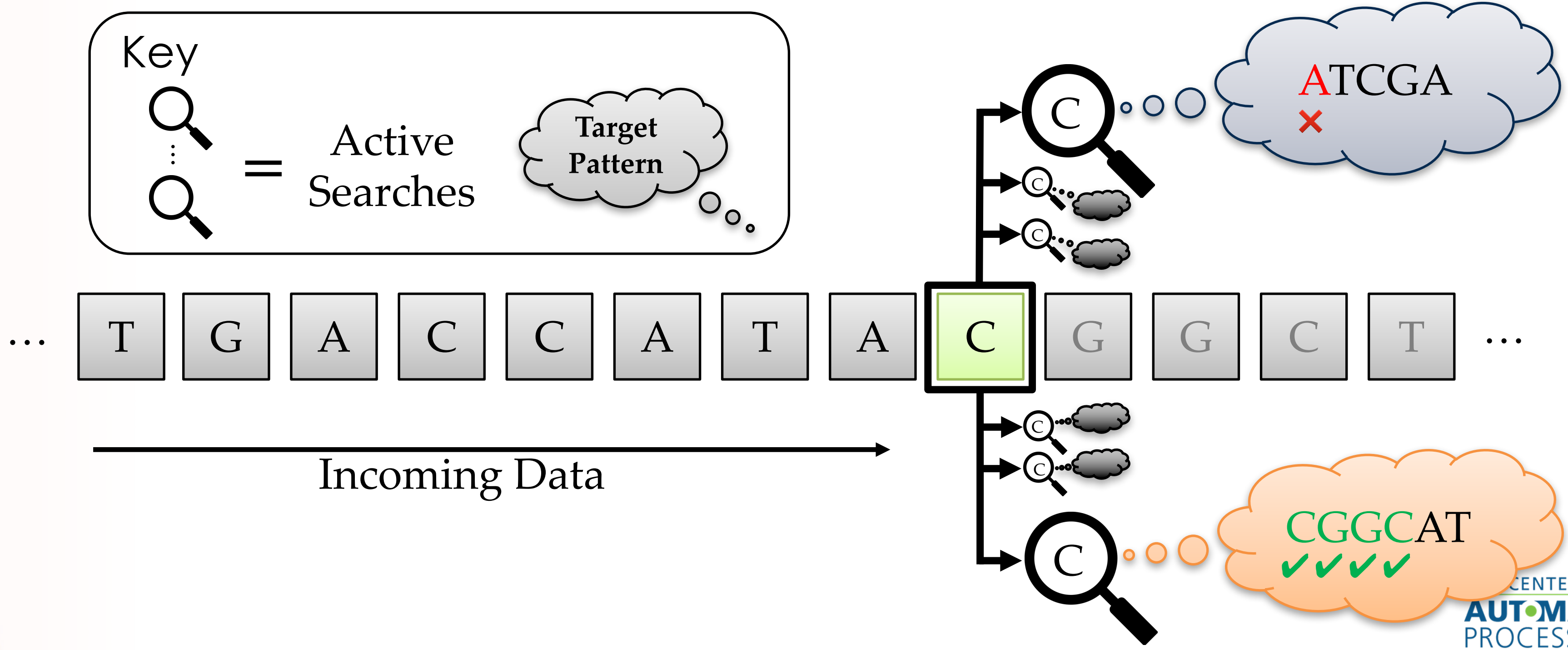
Parallel searches



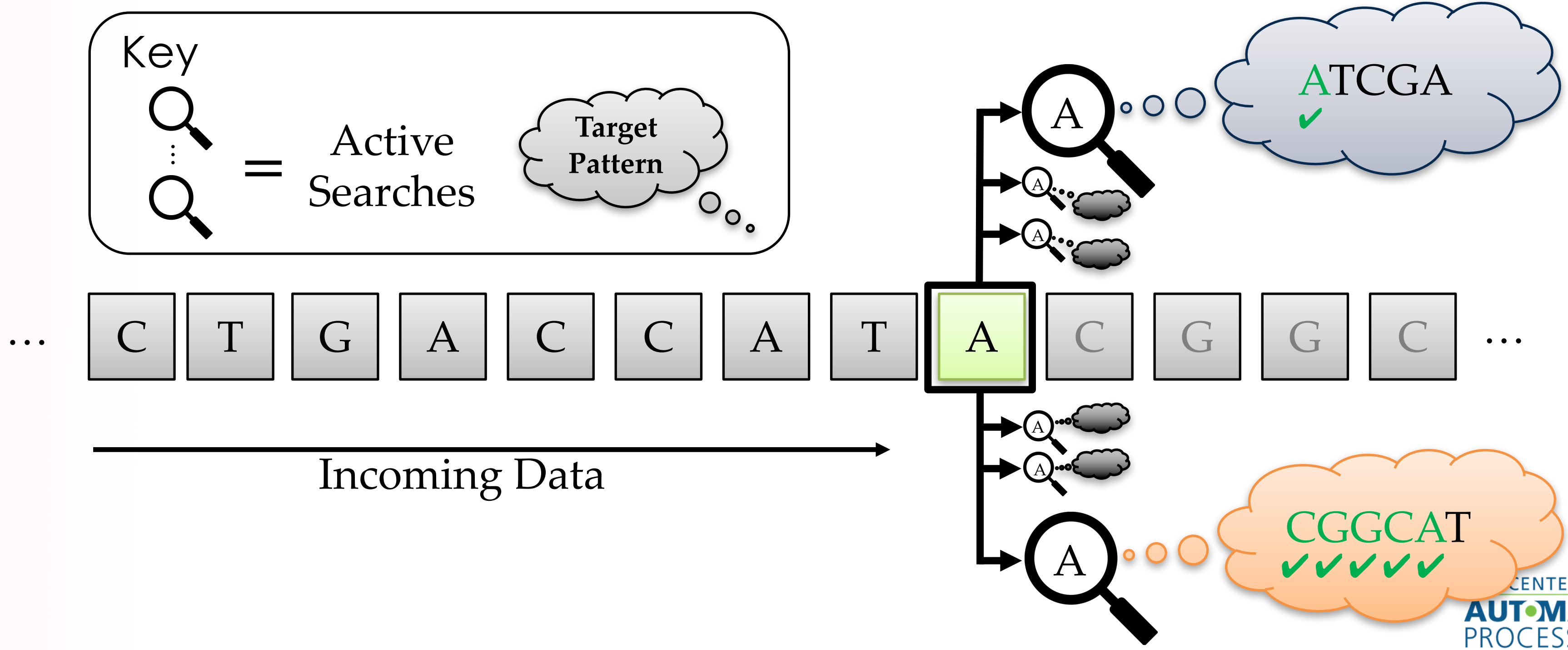
Parallel searches



Parallel searches

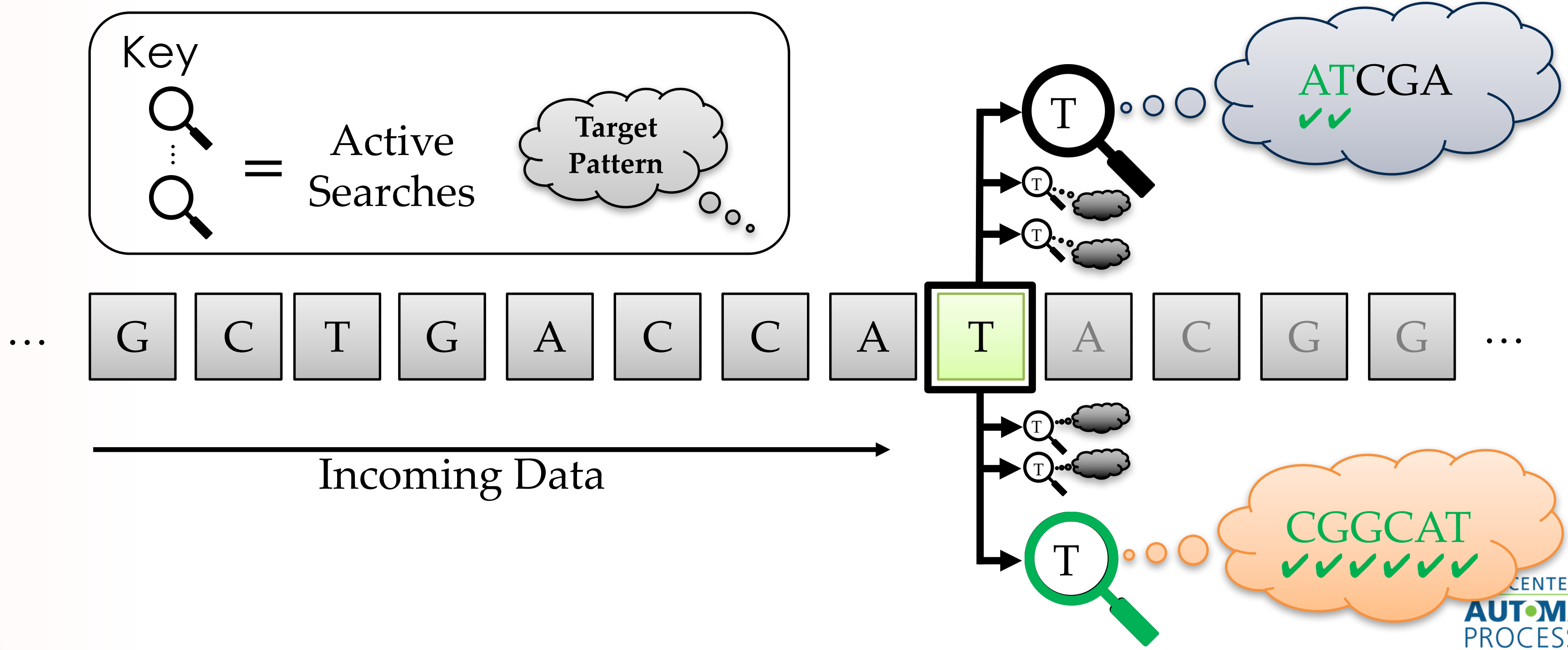


Parallel searches



CENTER FOR AUTOMATA PROCESSING

Parallel searches



Parallel Searches: Goals

- Fast processing
- Concise, maintainable representation
- Efficient compilation
  - High throughput
  - Low compilation time

Specialized Hardware

RAPID Programming Language

THE CENTER FOR AUTOMATA PROCESSING

A researcher should spend his or her time designing an algorithm to find the important data, not building a machine that will obey said algorithm.

The Remainder of this Talk

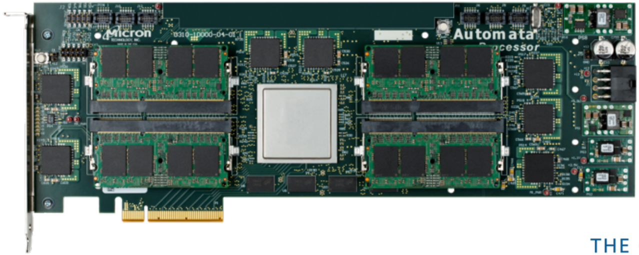
- Automata Processor
  - Architectural Overview
  - Current Programming Models
- RAPID Programming Language
  - Language Overview
  - AP Code Generation and Optimizations
- Experimental Evaluation
- Conclusions and Future Directions

The Remainder of this Talk

- **Automata Processor**
  - **Architectural Overview**
  - **Current Programming Models**
- RAPID Programming Language
  - Language Overview
  - AP Code Generation and Optimizations
- Experimental Evaluation
- Conclusions and Future Directions

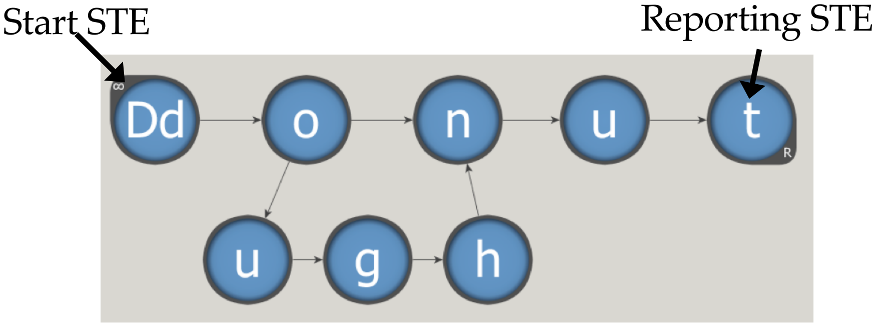
Micron's Automata Processor

- Accelerates identification of patterns in input data stream using massive parallelism
- Hardware implementation of non-deterministic finite automata
- 1 gbps data processing
- MISD architecture

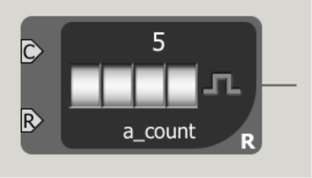


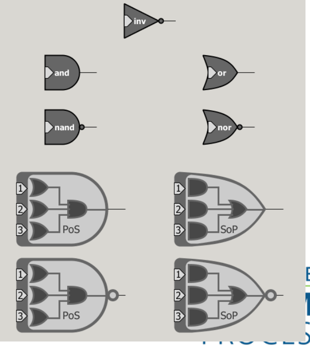
Micron's Automata Processor

- Implements homogeneous NFAs
  - All incoming edges to state have same symbol(s)
  - State Transition Element (STE)
- Memory-derived architecture
  - Memory as a computational medium
  - State consists of a column in DRAM array
  - Connections made with reconfigurable routing matrix partitioned into blocks
- 1.5 million states on development board
- Saturating Up Counter, Boolean Logic

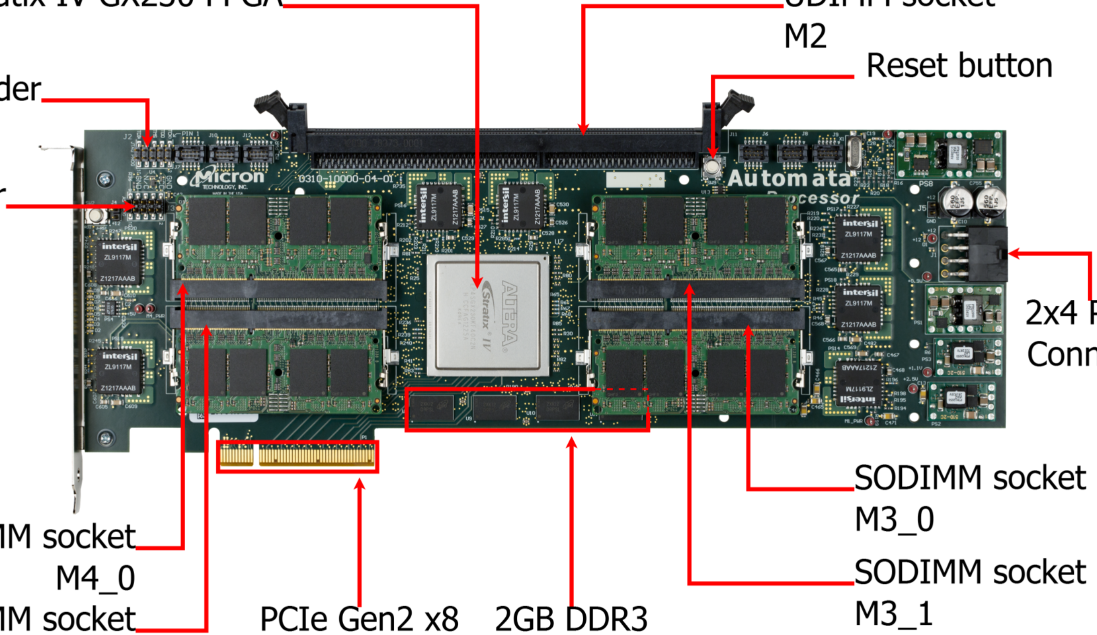


.*[Dd](o|ough)nut





Micron's Automata Processor



Altera Stratix IV GX230 FPGA

UDIMM socket
M2

Reset button

JTAG header

AS header

2x4 Power
Connector

SODIMM socket
M3_0

SODIMM socket
M3_1

SODIMM socket
M4_0

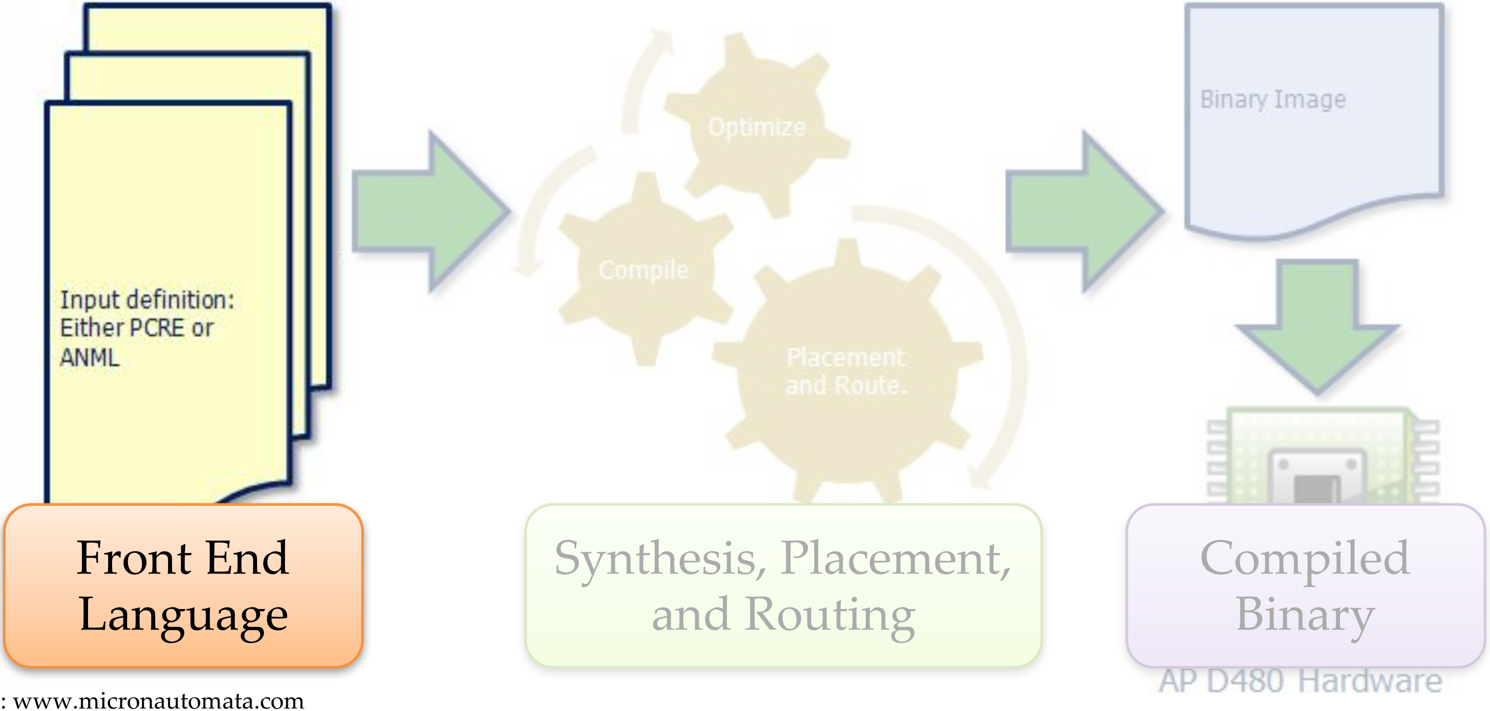
SODIMM socket
M4_1

PCIe Gen2 x8
Connector

2GB DDR3
M1

Figure courtesy of Micron

Programming Workflow



Front End Language

Synthesis, Placement, and Routing

Compiled Binary

Source: www.micronautomata.com

Current Programming Models

ANML

- Automata Network Markup Language
- Directly specify homogeneous NFA design
- High-level programming language bindings for generation

RegEx

- Support for a list of regular expressions
- Support for PCRE modifiers
- Compiled directly to binary

Programming Challenges

- ANML development akin to **assembly programming**
  - Requires knowledge of automata theory **and** hardware properties
  - Tedious and error-prone development process
- Regular expressions challenging to implement
  - Often exhaustive enumerations
  - Similarly error-prone

Programming Challenges

- Implement **single instance** of a problem
  - Each instance of a problem requires a brand new design
  - Need for meta-programs to generate final design

- Current programming models place unnecessary burden on developer

Goals: Current Approaches Fail

- Fast processing ✔
- Concise, maintainable representation ❌
- Efficient compilation
  - High throughput ✔
  - Low compilation time ⚠️

The Remainder of this Talk

- Automata Processor
  - Architectural Overview
  - Current Programming Models
- **RAPID Programming Language**
  - **Language Overview**
  - **AP Code Generation and Optimizations**
- Experimental Evaluation
- Conclusions and Future Directions

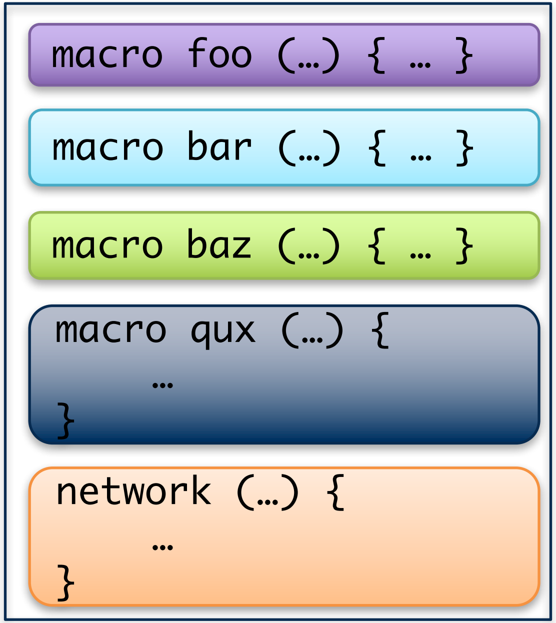
THE CENTER FOR AUTOMATA PROCESSING

RAPID at a Glance

- Provides concise, maintainable, and efficient representations for pattern-identification algorithms
- Conventional, C-style language with domain-specific parallel control structures
- Excels in applications where patterns are best represented as a combination of text and computation
- Compilation strategy balances synthesis time with device utilization

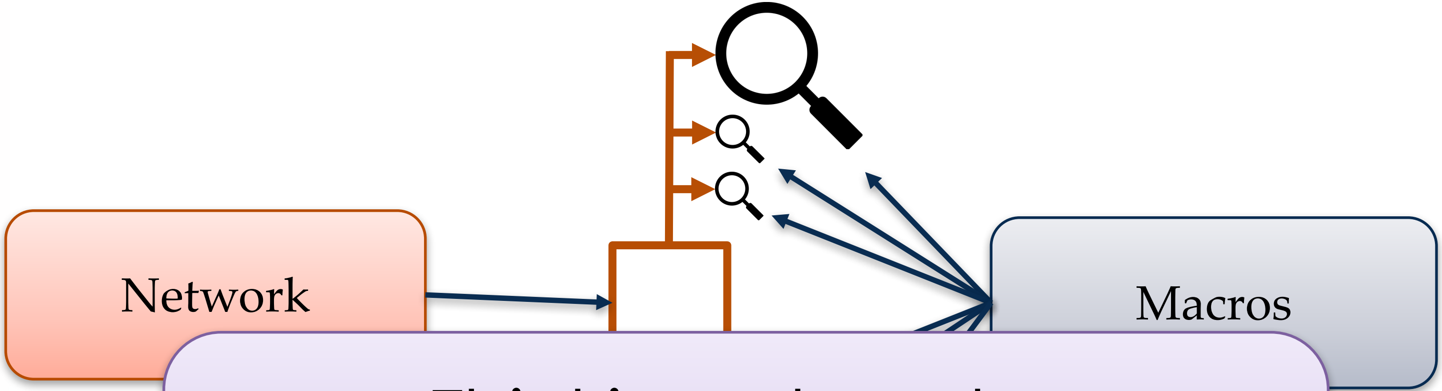
Program Structure

- **Macro**
  - Basic unit of computation
  - Sequential control flow
  - Boolean expressions as statements for terminating threads of computation
- **Network**
  - High-level pattern matching
  - Parallel control flow
  - Parameters to set run-time values



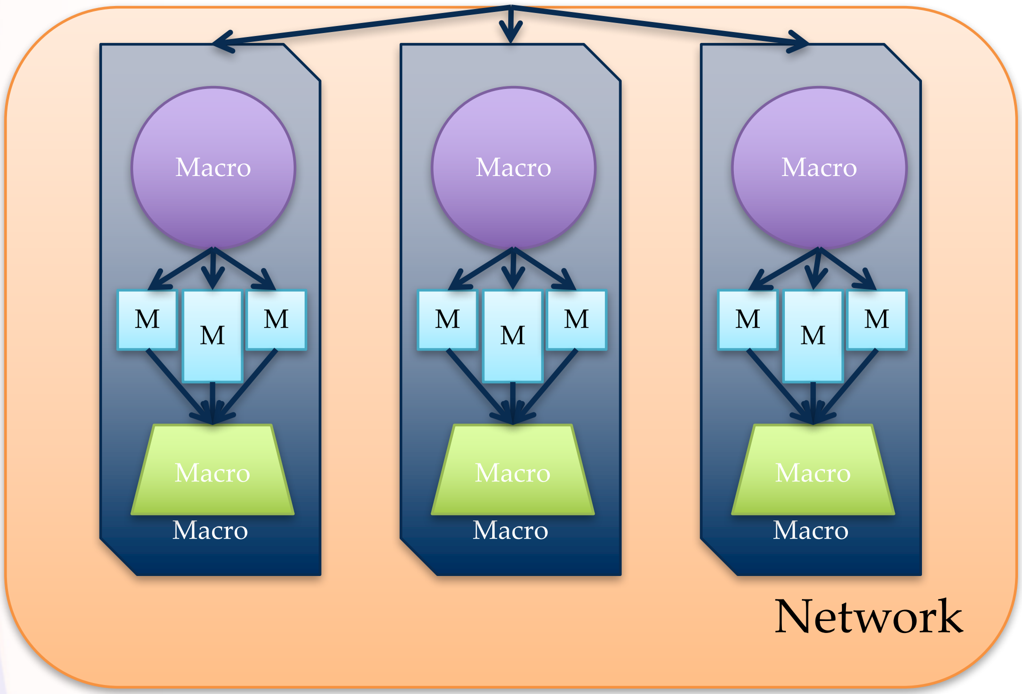
```
macro foo (…) { … }

macro bar (…) { … }

macro baz (…) { … }

macro qux (…) {
     …
}

network (…) {
     …
}
```

Program Structure



Thinking ahead…
This program structure also exposes optimizations

Program Structure



```
macro foo (…) { … }

macro bar (…) { … }

macro baz (…) { … }

macro qux (…) {
      …
}

network ( … ) {
      …
}
```

Data in RAPID

- Input data stream as special function
  - Stream of characters
  - `input()`
    - Calls to `input()` are synchronized across all active macros
    - All active macros receive the same input character

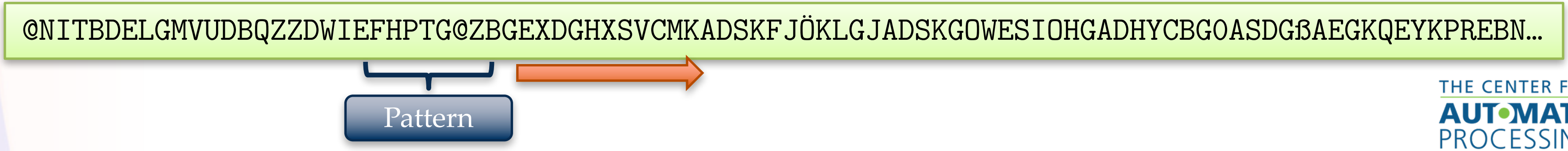
Counting and Reporting

- `Counter`: Abstract representation of saturating up counters
  - Count and Reset operations
  - Can compare against threshold
- RAPID programs can *report*
  - Triggers creation of report event
  - Captures offset of input stream and current macro

THE CENTER FOR AUTOMATA PROCESSING

Parallel Control Structures

- Concise specification of multiple, simultaneous comparisons against a single data stream
- Support MISD computational model
- Static and dynamic thread spawning for massive parallelism support
- Explicit support for sliding window computations


@NITBDELGMVUDBQZZDWIEFHPTG@ZBGEXDGHXSVCMKADSKFJÖKLGJADSKGOWESIOHGADHYCBGOASDGßAEGKQEYKPREBN…

THE CENTER FOR AUTOMATA PROCESSING

Parallel Control Structures

Sequential Structure	Parallel Structure
if…else	either…orelse
foreach	some
while	whenever

Either/Orelse Statements

```
either {
    hamming_distance(s,d); //hamming distance
    ’y’ == input();         //next input is ’y’
    report;                 //report candidate
} orelse {
    while(’y’ != input()); //consume until ’y’
}
```

- Perform parallel exploration of input data
- Static number of parallel operations

Some Statements

```
network (String[] comparisons) {
    some(String s : comparisons)
        hamming_distance(s,5);
}
```

- Parallel exploration may depend on candidate patterns
- Iterates over items, dynamically spawn computation

Whenever Statements

```
whenever( ALL_INPUT == input() ) {
    foreach(char c : "rapid")
        c == input();
    report;
}
```

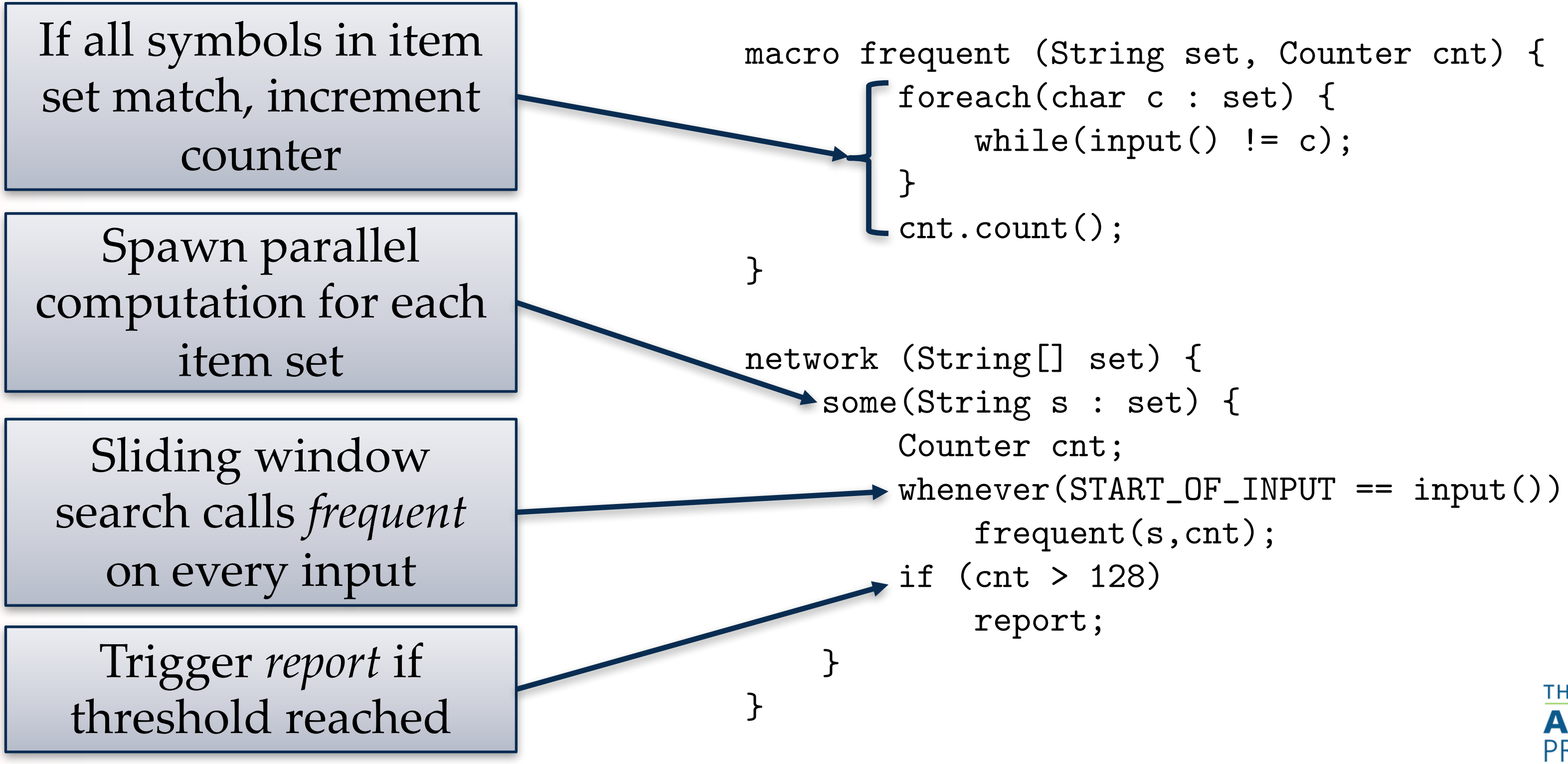
- Body triggered whenever guard becomes true
- `ALL_INPUT`: any symbol in the input stream

Example RAPID Program

Association Rule Mining

Identify items from a database that frequently occur together

Example RAPID Program



If all symbols in item set match, increment counter

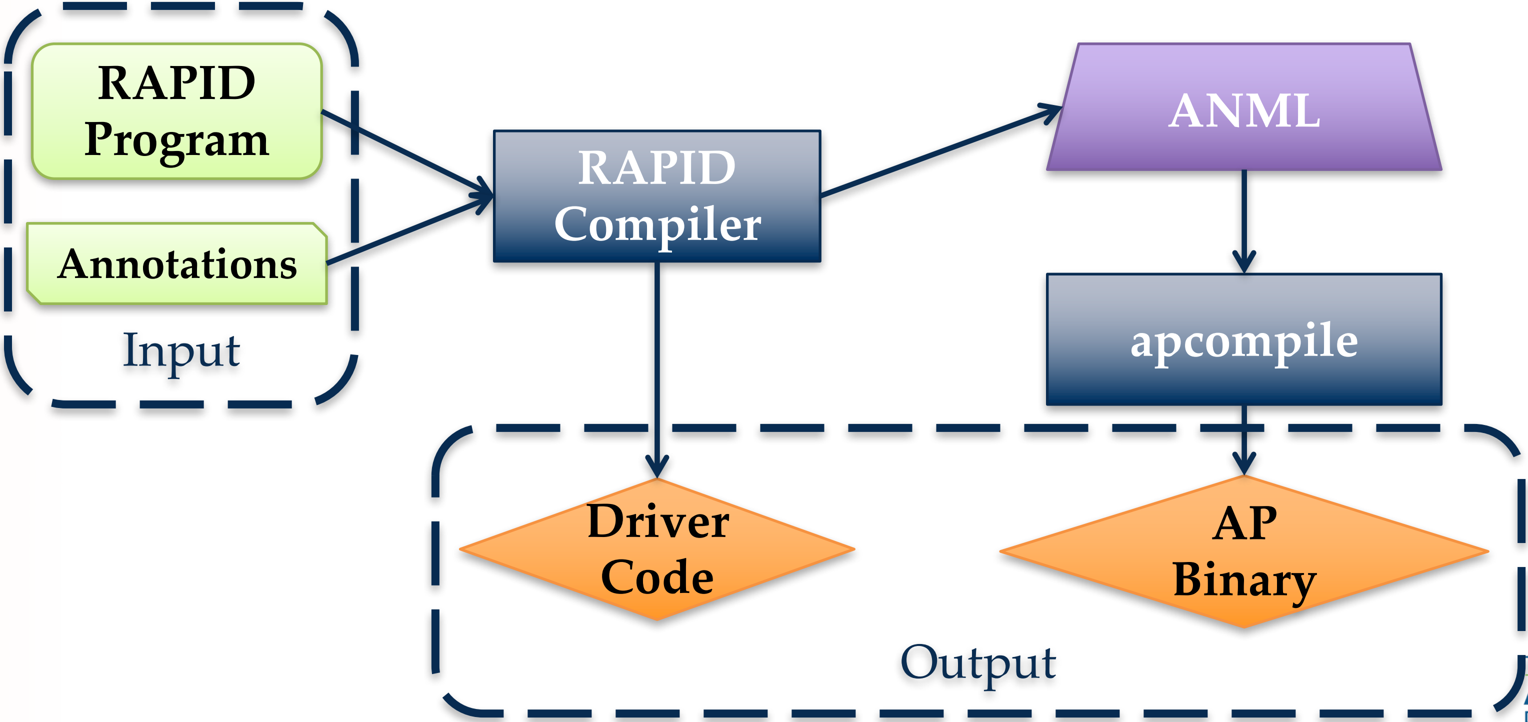
Spawn parallel computation for each item set

Sliding window search calls *frequent* on every input

Trigger *report* if threshold reached

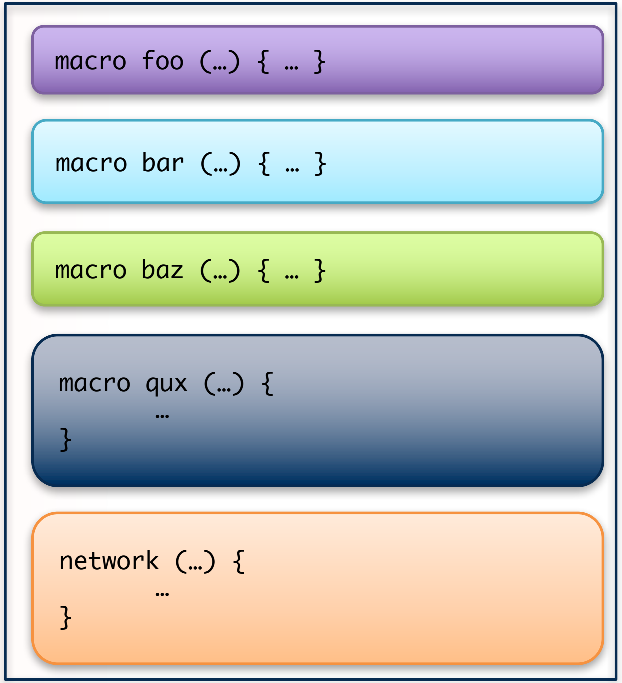
```
macro frequent (String set, Counter cnt) {
    foreach(char c : set) {
      while(input() != c);
    }
    cnt.count();
}

network (String[] set) {
  some(String s : set) {
    Counter cnt;
    whenever(START_OF_INPUT == input())
      frequent(s,cnt);
    if (cnt > 128)
      report;
  }
}
```

System Overview



Code Generation

RAPID Program



```
macro foo (…) { … }

macro bar (…) { … }

macro baz (…) { … }

macro qux (…) {
        …
}

network (…) {
        …
}
```

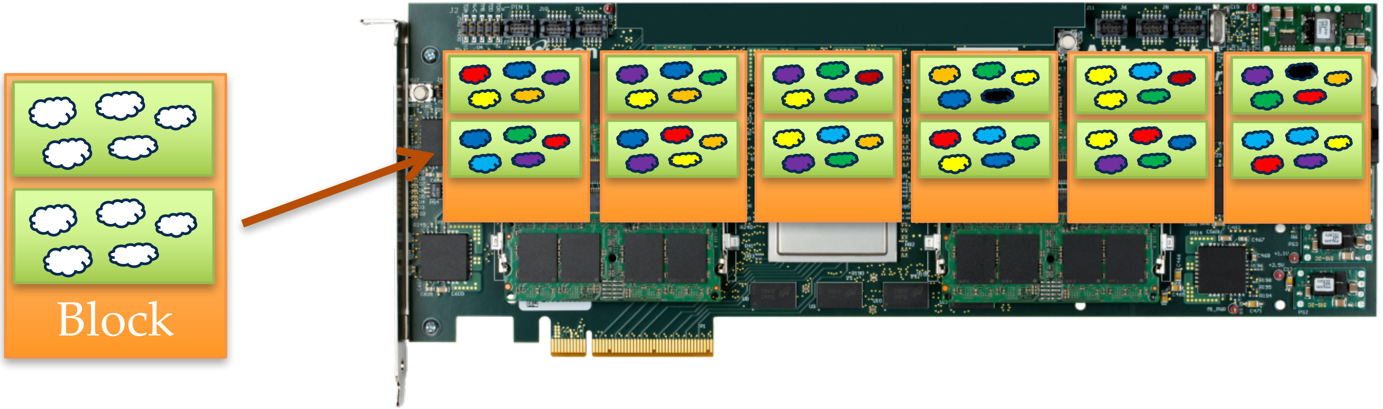
- Recursive transformation of RAPID program
  - Input Stream → STEs
  - Counters → 1 or more physical counter(s)
- Similar to RegEx → NFA transformation

Challenge: Synthesis

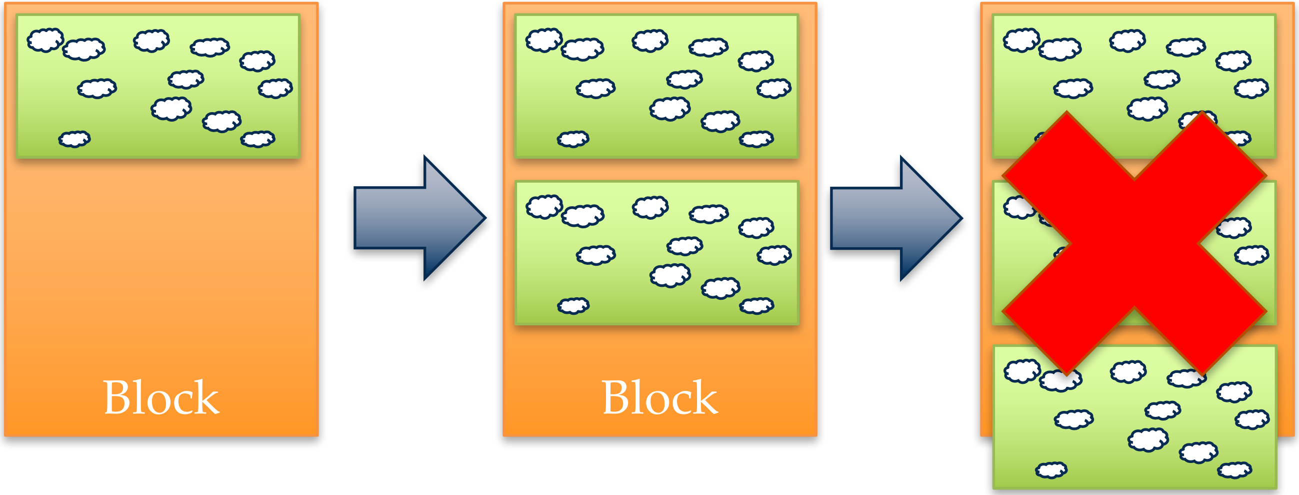
- Placement and routing are resource-intensive
- Large AP designs often fail outright
- **Goal:** technique to reduce AP design such that synthesis tools succeed

Tessellation Optimization

- Automata Processor designs are often **repetitive**
- Programmatically **extract** repetition, and compile once
- Load **dynamically** at runtime



Auto-Tuning Optimization



Tessellation Advantages

- Reduces overall compilation time
  - Smaller design requires less time to place and route
  - Shorter debug cycle increases productivity
- Improved device utilization
  - Reduced search space size for place and route
  - Able to find "better" solution

The Remainder of this Talk

- Automata Processor
  - Architectural Overview
  - Current Programming Models
- RAPID Programming Language
  - Language Overview
  - AP Code Generation and Optimizations
- **Experimental Evaluation**
- Conclusions and Future Directions

Reminder: Goals

- Fast processing ✓
- Concise, maintainable representation
- Efficient compilation
  - High throughput
  - Low compilation time

Research Questions

1. Do RAPID constructs generalize to pattern search problems across multiple problem domains?
2. (**Conciseness**) Do RAPID programs require fewer lines of code than a functionally equivalent ANML program to represent a given pattern search problem?
3. (**Maintainability**) Does a RAPID program require fewer modifications than an equivalent ANML program to alter functionality?
4. (**Efficiency**) Are RAPID programs no less efficient at runtime and during synthesis than hand-optimized ANML programs?

Research Questions

1. **Do RAPID constructs generalize to pattern search problems across multiple problem domains?**
2. (**Conciseness**) Do RAPID programs require fewer lines of code than a functionally equivalent ANML program to represent a given pattern search problem?
3. (**Maintainability**) Does a RAPID program require fewer modifications than an equivalent ANML program to alter functionality?
4. (**Efficiency**) Are RAPID programs no less efficient at runtime and during synthesis than hand-optimized ANML programs?

Description of Benchmarks

Benchmark	Description	Domain	Baseline Generation Method
ARM	Association Rule Mining	ML	Meta Program
Brill	Brill Part of Speech Tagging	NLP	Meta Program
Exact	Exact DNA Alignment	Bioinformatics	ANML
Gappy	DNA Alignment with Gaps	Bioinformatics	ANML
MOTOMATA	Planted Motif Search	Bioinformatics	ANML

Research Questions

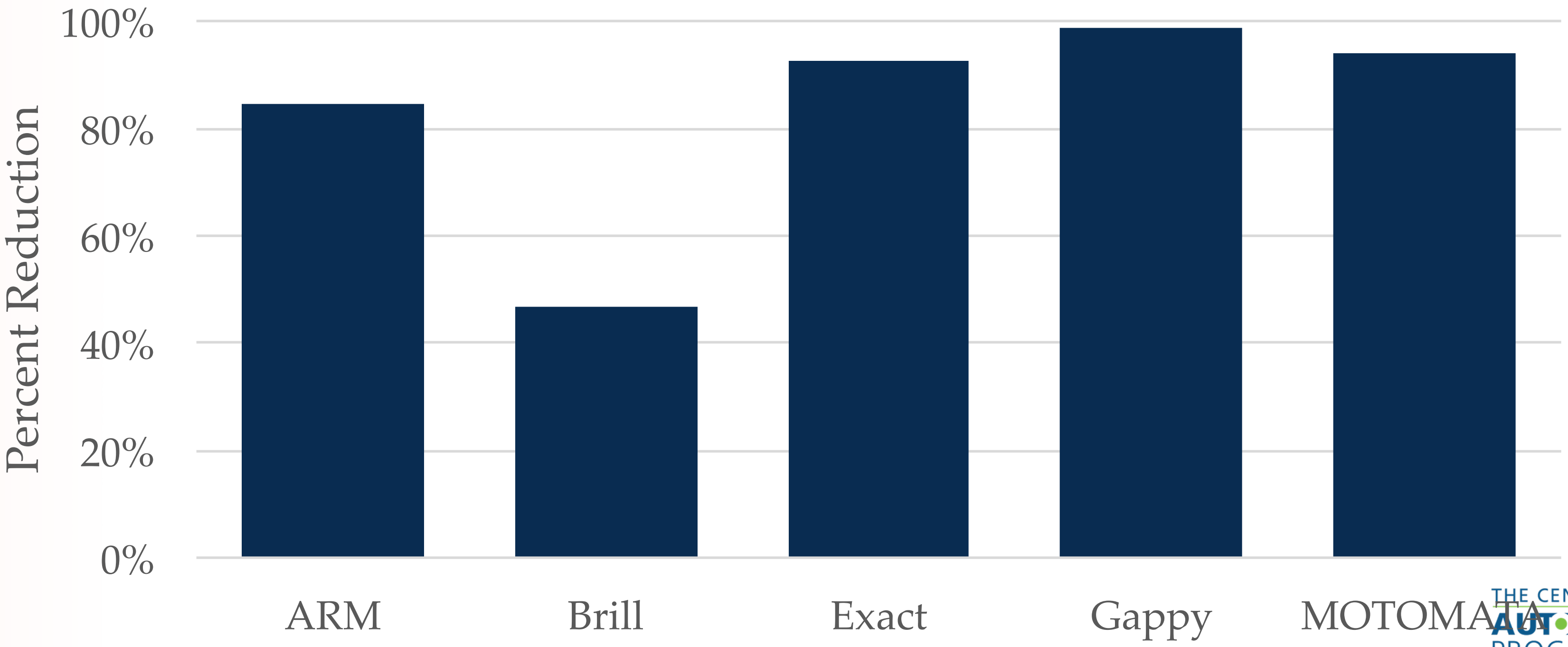
1. Do RAPID constructs generalize to pattern search problems across multiple problem domains?

2. **(Conciseness) Do RAPID programs require fewer lines of code than a functionally equivalent ANML program to represent a given pattern search problem?**

3. (**Maintainability**) Does a RAPID program require fewer modifications than an equivalent ANML program to alter functionality?

4. (**Efficiency**) Are RAPID programs no less efficient at runtime and during synthesis than hand-optimized ANML programs?

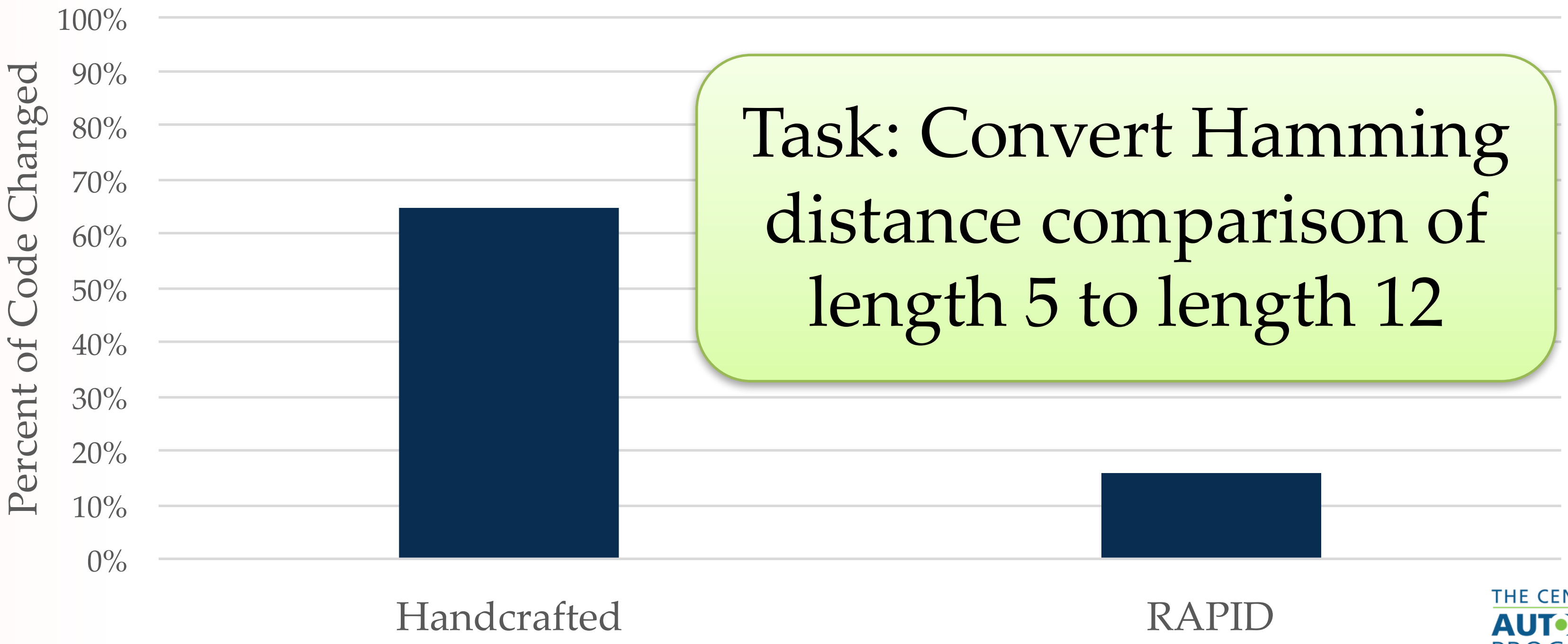
RAPID Lines of Code



Research Questions

1. Do RAPID constructs generalize to pattern search problems across multiple problem domains?
2. (**Conciseness**) Do RAPID programs require fewer lines of code than a functionally equivalent ANML program to represent a given pattern search problem?
3. **(Maintainability) Does a RAPID program require fewer modifications than an equivalent ANML program to alter functionality?**
4. (**Efficiency**) Are RAPID programs no less efficient at runtime and during synthesis than hand-optimized ANML programs?

RAPID is Maintainable



Task: Convert Hamming distance comparison of length 5 to length 12

Research Questions

1. Do RAPID constructs generalize to pattern search problems across multiple problem domains?

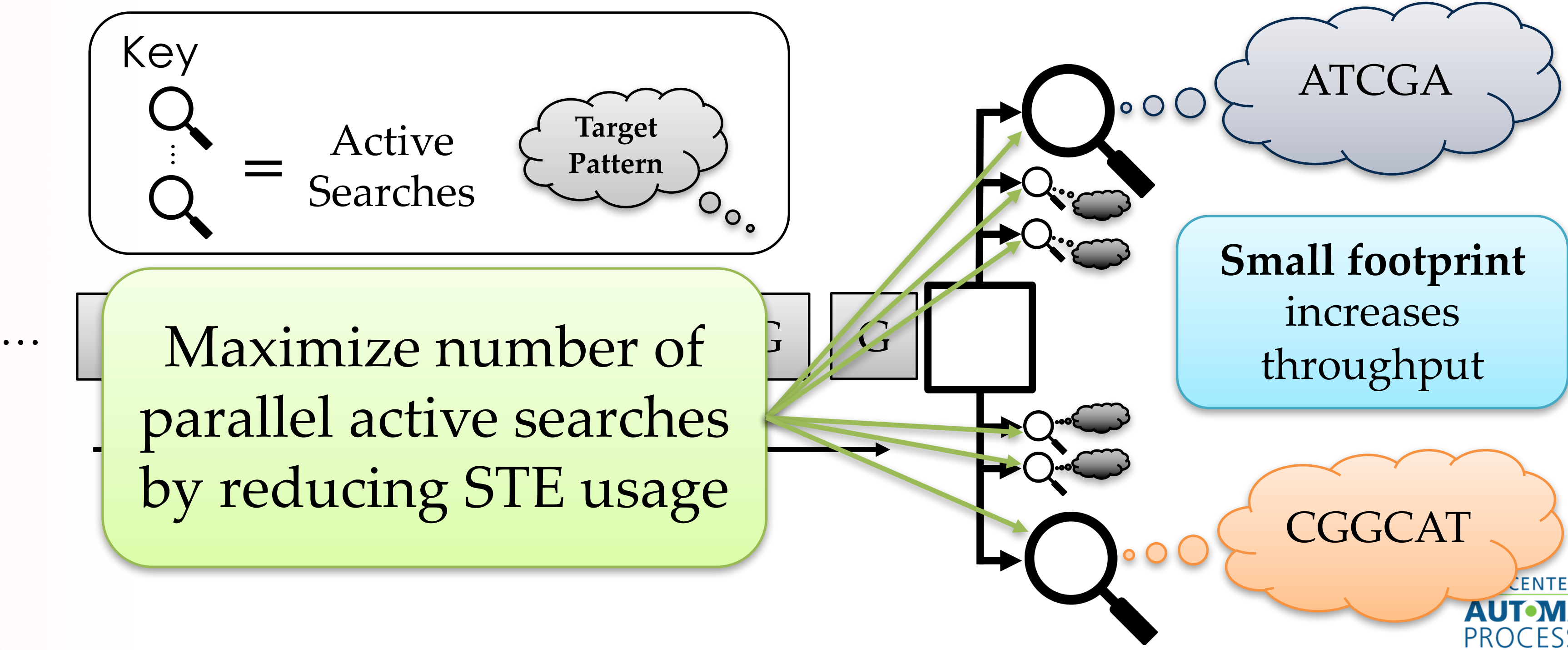
2. (**Conciseness**) Do RAPID programs require fewer lines of code than a functionally equivalent ANML program to represent a given pattern search problem?

3. (**Maintainability**) Does a RAPID program require fewer modifications than an equivalent ANML program to alter functionality?

4. **(Efficiency) Are RAPID programs no less efficient at runtime and during synthesis than hand-optimized ANML programs?**

THE CENTER FOR AUTOMATA PROCESSING

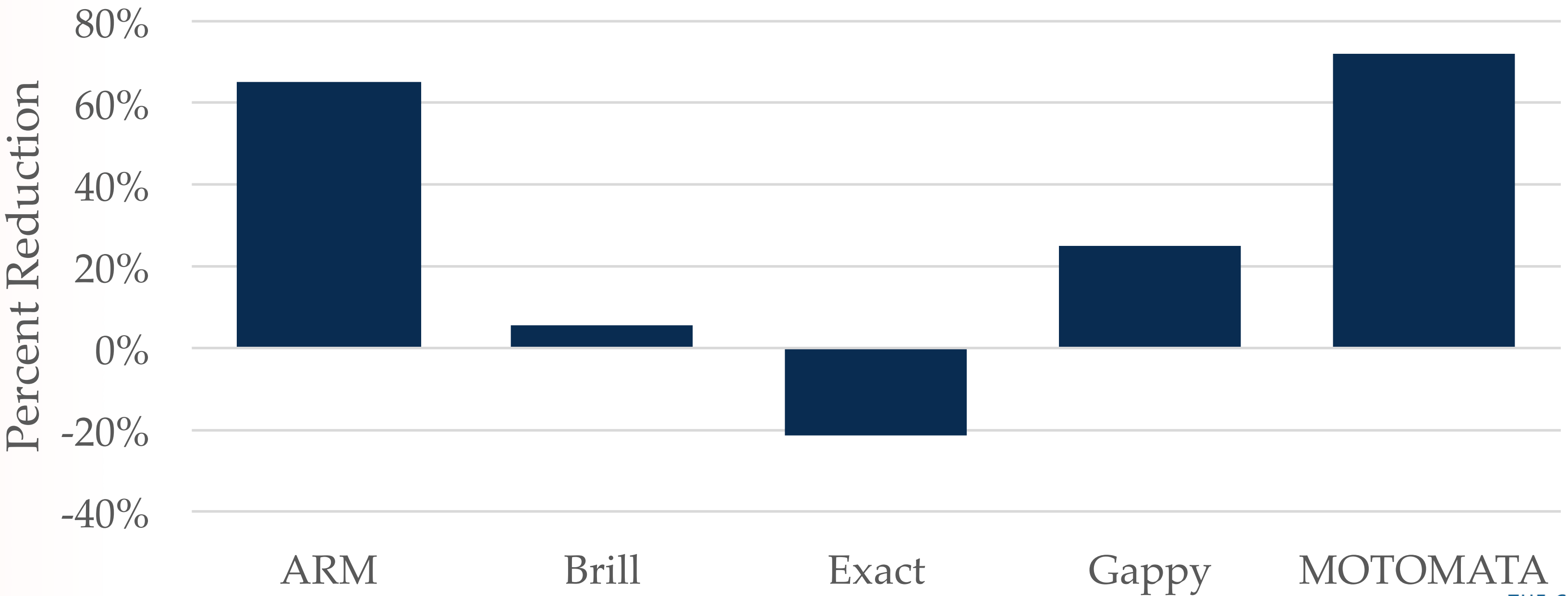
Parallel searches



Maximize number of parallel active searches by reducing STE usage

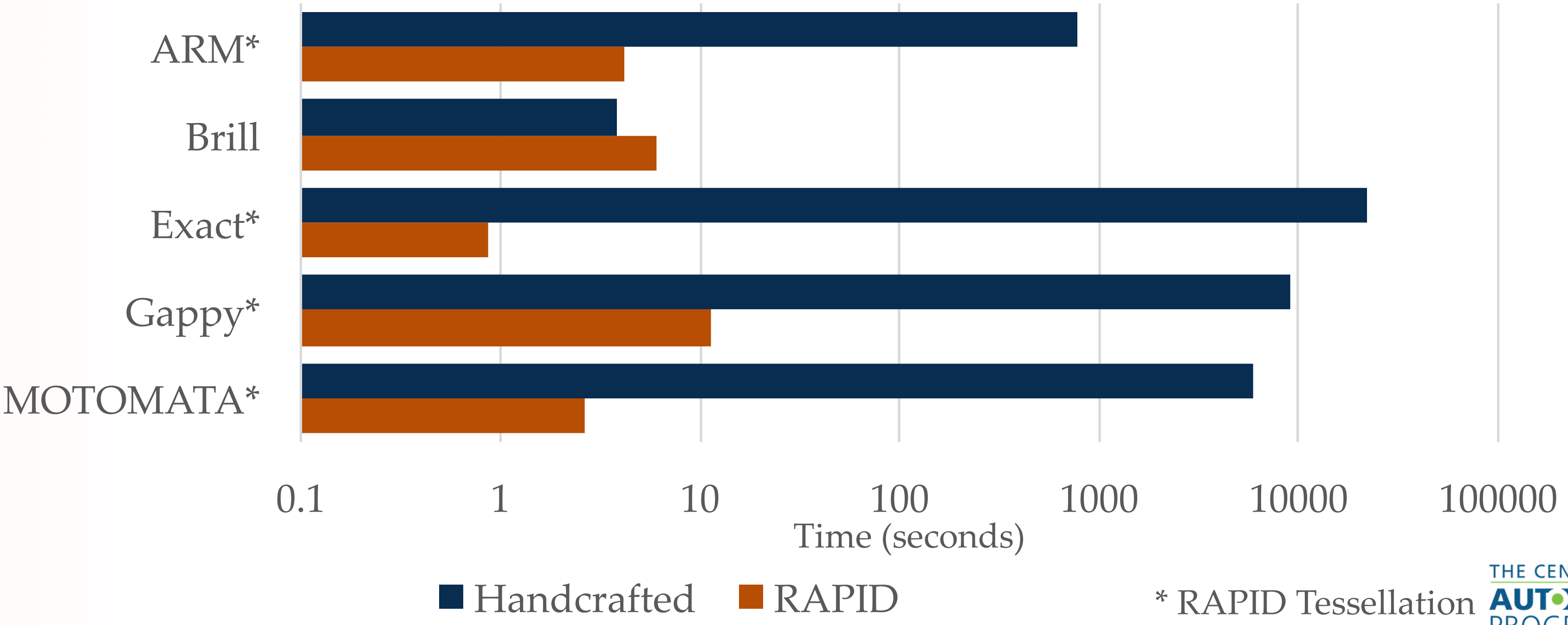
Small footprint increases throughput

Generated STEs



THE CENTER FOR AUTOMATA PROCESSING

Compilation Time



* RAPID Tessellation

Research Questions

1. Do RAPID constructs generalize to pattern search problems across multiple problem domains? **YES**

2. (**Conciseness**) Do RAPID programs require fewer lines of code than a functionally equivalent ANML program to represent a given pattern search problem? **YES**

3. (**Maintainability**) Does a RAPID program require fewer modifications than an equivalent ANML program to alter functionality? **YES**

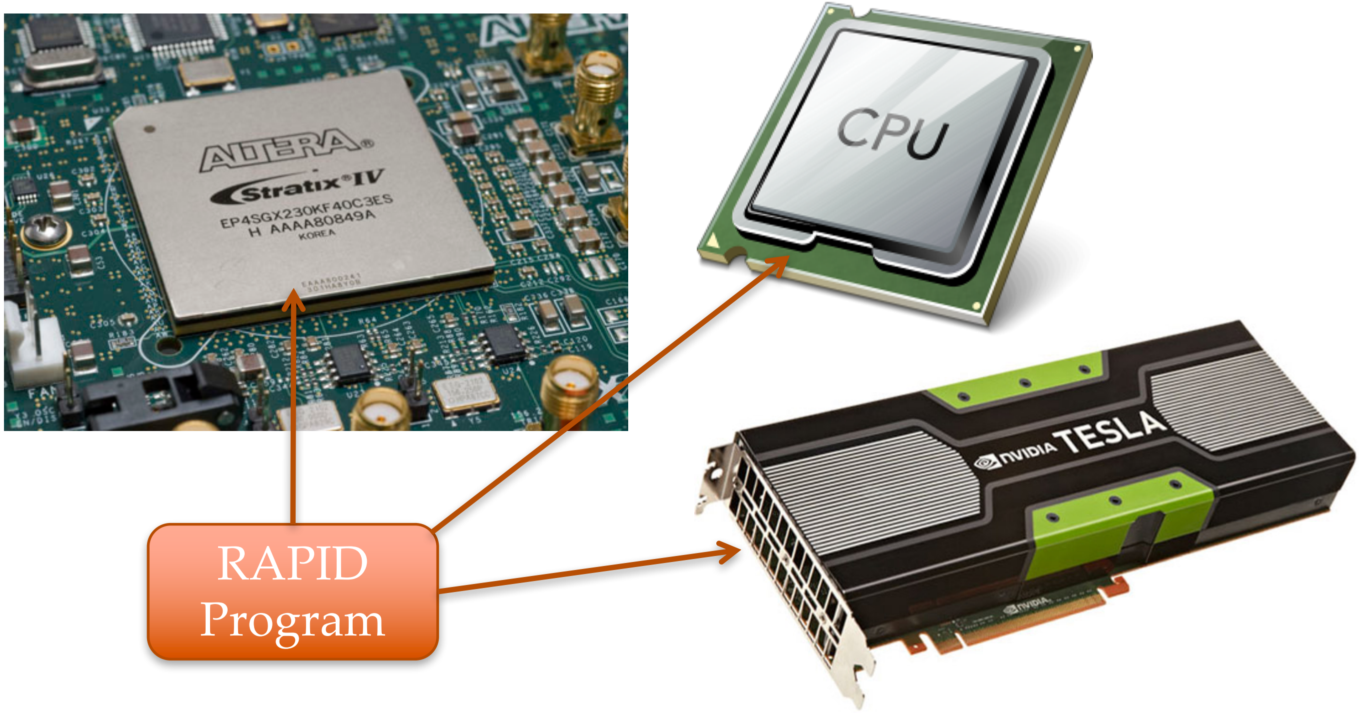
4. (**Efficiency**) Are RAPID programs no less efficient at runtime and during synthesis than hand-optimized ANML programs? **OFTEN** **(YES)**

The Remainder of this Talk

- Automata Processor
  - Architectural Overview
  - Current Programming Models
- RAPID Programming Language
  - Language Overview
  - AP Code Generation and Optimizations
- Experimental Evaluation
- **Conclusions and Future Directions**

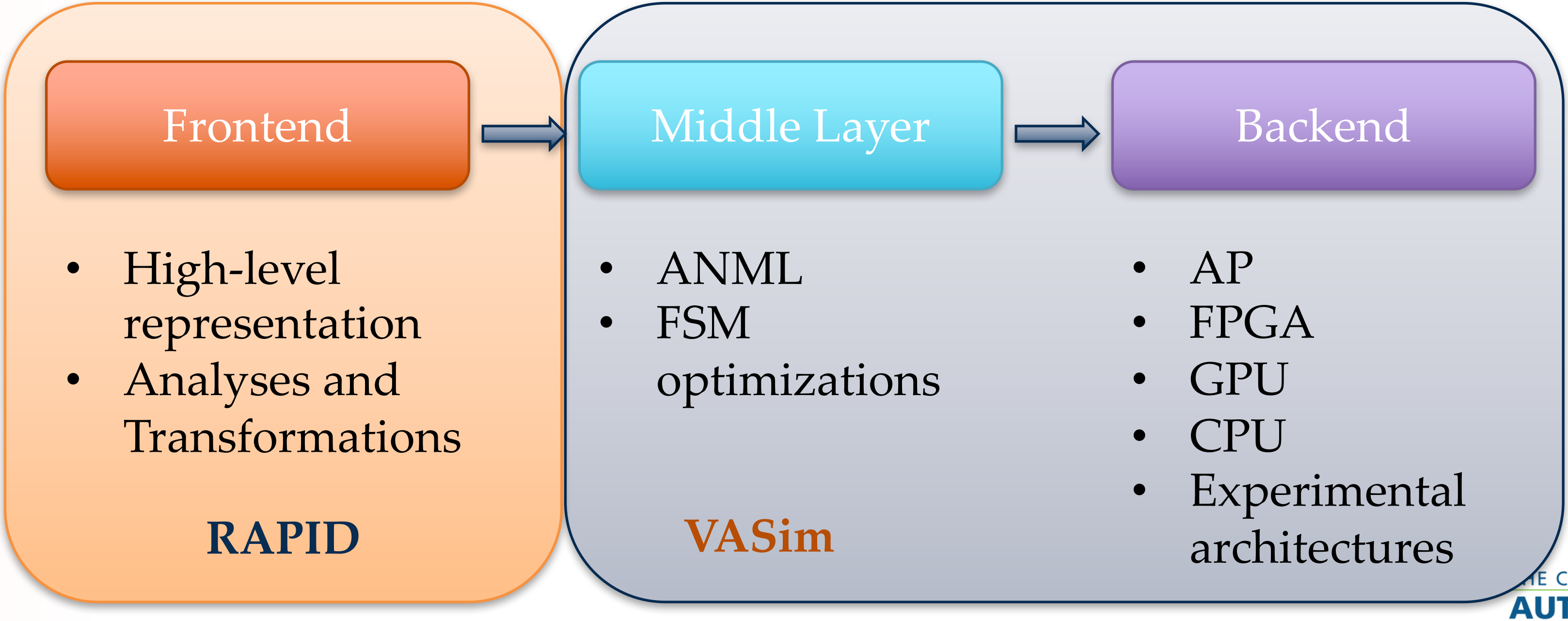
THE CENTER FOR AUTOMATA PROCESSING

Architectural Targets



RAPID Program

Multi-Layer Automata Research Framework



Frontend → Middle Layer → Backend

Frontend

- High-level representation
- Analyses and Transformations

RAPID

Middle Layer

- ANML
- FSM optimizations

VASim

Backend

- AP
- FPGA
- GPU
- CPU
- Experimental architectures

Debugging Support

- Spurious reports in large data stream
- Can we quickly "sweep" to problematic region and inspect?
- Replay debugging

Open Source Release

- Prototype compiler will be available on GitHub
- BSD-style license
- Available in the coming weeks

Conclusions

- RAPID is a **concise**, **maintainable**, and **efficient** high-level language for pattern-search algorithms
- Achieved with domain-specific **parallel control structures**, and **suitable data representations**
- Prototype compiler allows for execution using the Automata Processor, FPGA, CPU

QUESTIONS

This work was supported in part by the Center for Future Architectures Research (C-FAR), one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

THE CENTER FOR AUTOMATA PROCESSING