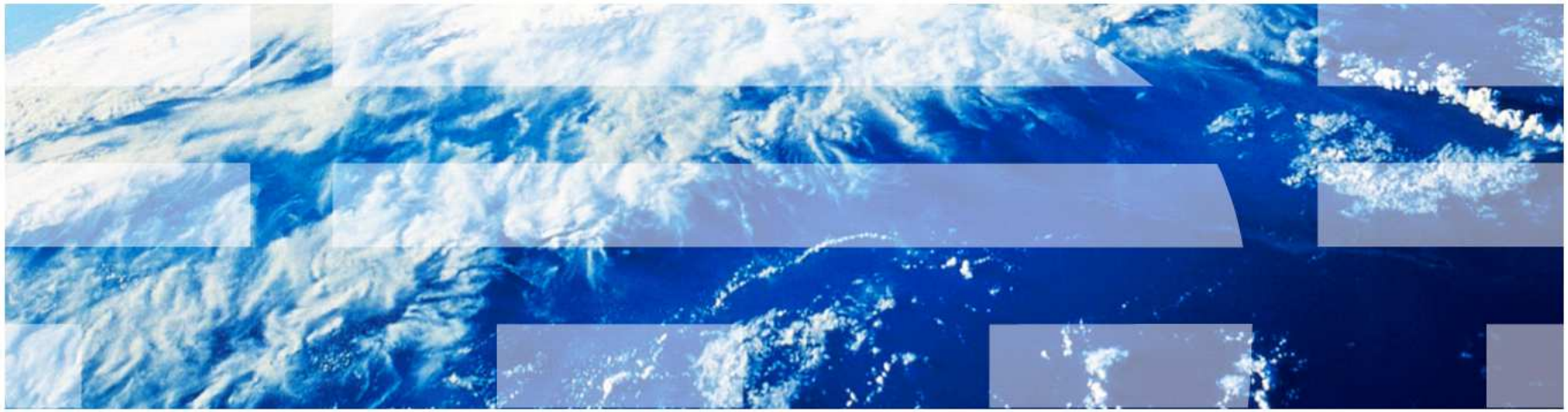
Designing a Programmable Wire-Speed Regular-Expression Matching Accelerator

Jan van Lunteren, Christoph Hagleitner, Timothy Heil, Giora Biran,
Uzi Shvadron, Kubilay Atasu



Agenda

1. Objectives and Challenges
2. RegX Architecture Overview
3. Memory Hierarchy
4. Local Results Processor
5. Compiler
6. IBM PowerEN™
7. Hardware-Measured Performance
8. Conclusions

Design Objectives

- a regular-expression *scanner* and pattern *compiler*, supporting
  1. large sets of string and regular expression patterns (~10K)
  2. multiple active pattern contexts
  3. high scan rates (~20 Gbit/s)
  4. parallel scans, multi-session support (millions of active sessions)
  5. incremental and dynamic updates

Design Challenges

- efficient use of available *memory capacity* and *bandwidth*
  - compact data structure
  - minimize number of memory accesses to process each input character
  - optimize cache hit rate
- exploit limited amount of parallelism to obtain small session state
- fast compilation times

RegX Architecture Overview

- Pattern-scanner designs (SW, HW) are typically based on deterministic (DFA) or non-deterministic finite automata (NFA)

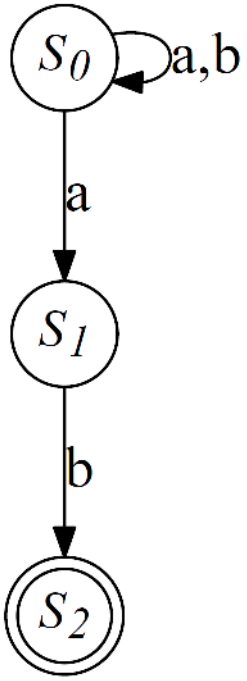
- Pros/cons:

	DFA	NFA
processing complexity	+	-
storage complexity	-	+

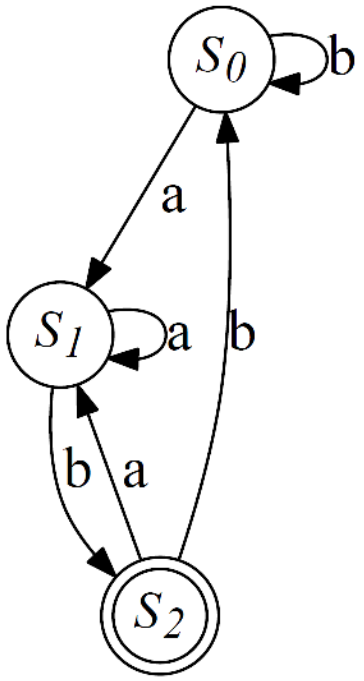
- Many published designs/solutions involve a DFA-based scan operation in combination with techniques to optimize storage-efficiency, targeting
  - compact executable representations of given DFAs
  - optimization of the DFAs themselves

- Differentiating factor from related work: techniques applied in RegX enable a deterministic scan throughput that is independent of the input characteristics, when executing out of dedicated memory (caches)
  - important for dealing with Denial of Service (DoS) attacks

- Sample regular expression: **(a|b)*ab**



Non-Deterministic Finite Automaton (NFA)



Deterministic Finite Automaton (DFA)

- NFA: several possible next states can exist for given state and input
- DFA: at most one possible next state exists for each state and input

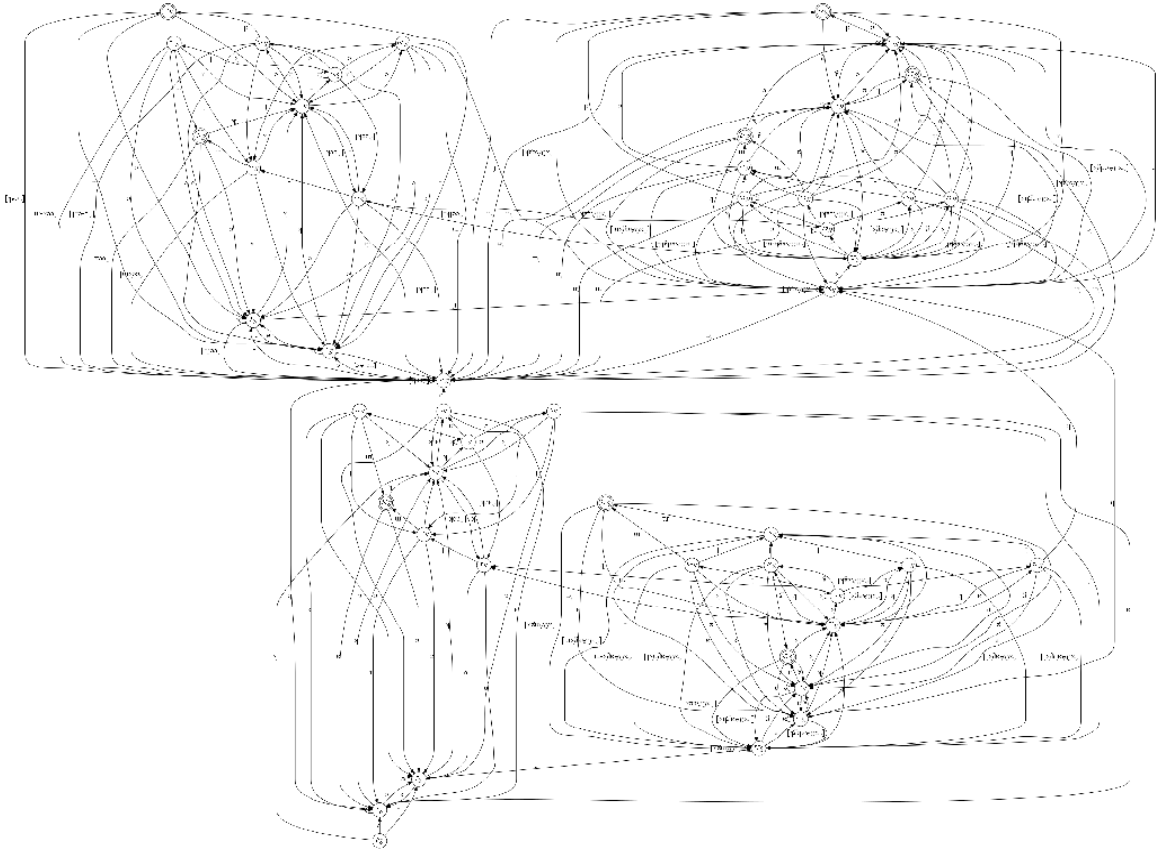
RegX Architecture Overview – *State Explosion*

- Certain combinations of regular-expression patterns can cause a "state-explosion" when mapped on a single DFA

- Example:

```
ab.*cd
ef[^\n]*gh
k.lm
```

➔ DFA with 48 states, 242 transition rules

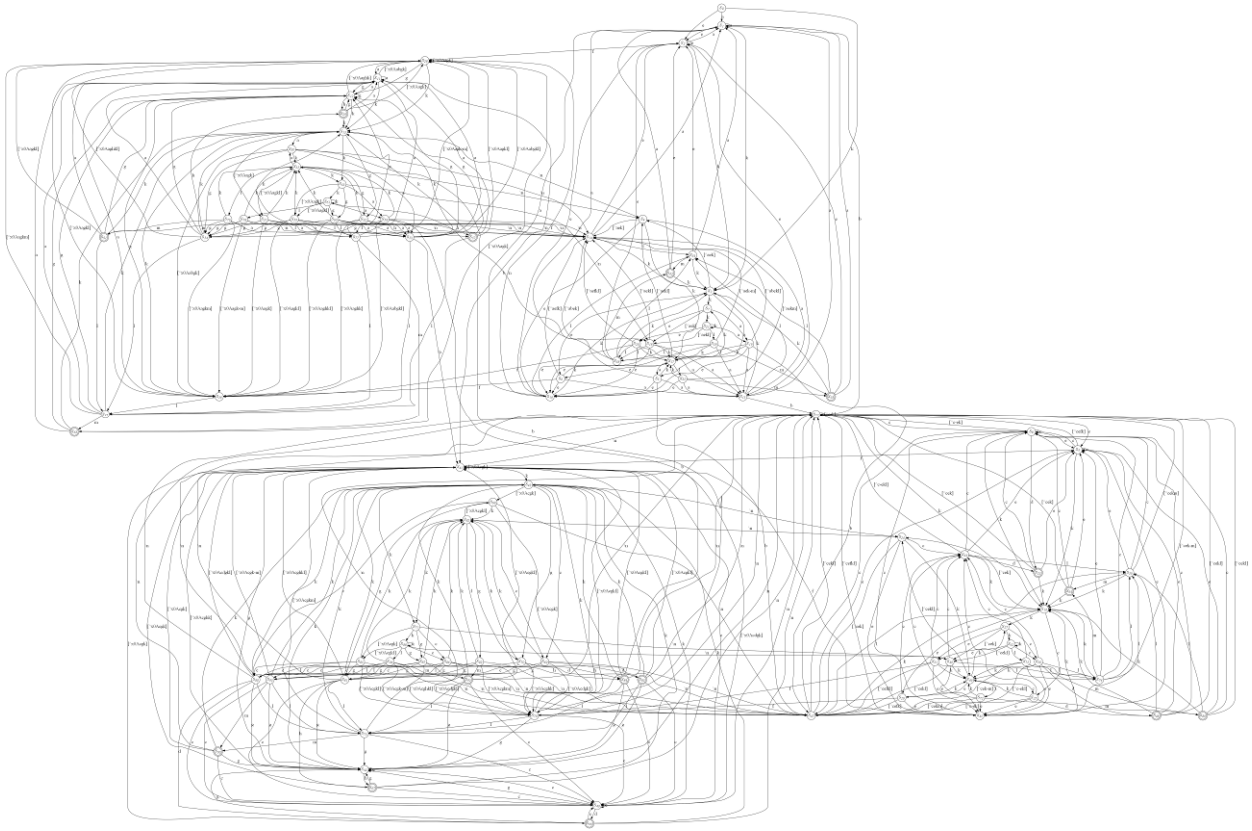


- Certain combinations of regular-expression patterns can cause a "state-explosion" when mapped on a single DFA

- Example:

```
ab.*cd
ef[^\n]*gh
k..lm
```

➔ DFA with 96 states, 508 transitions

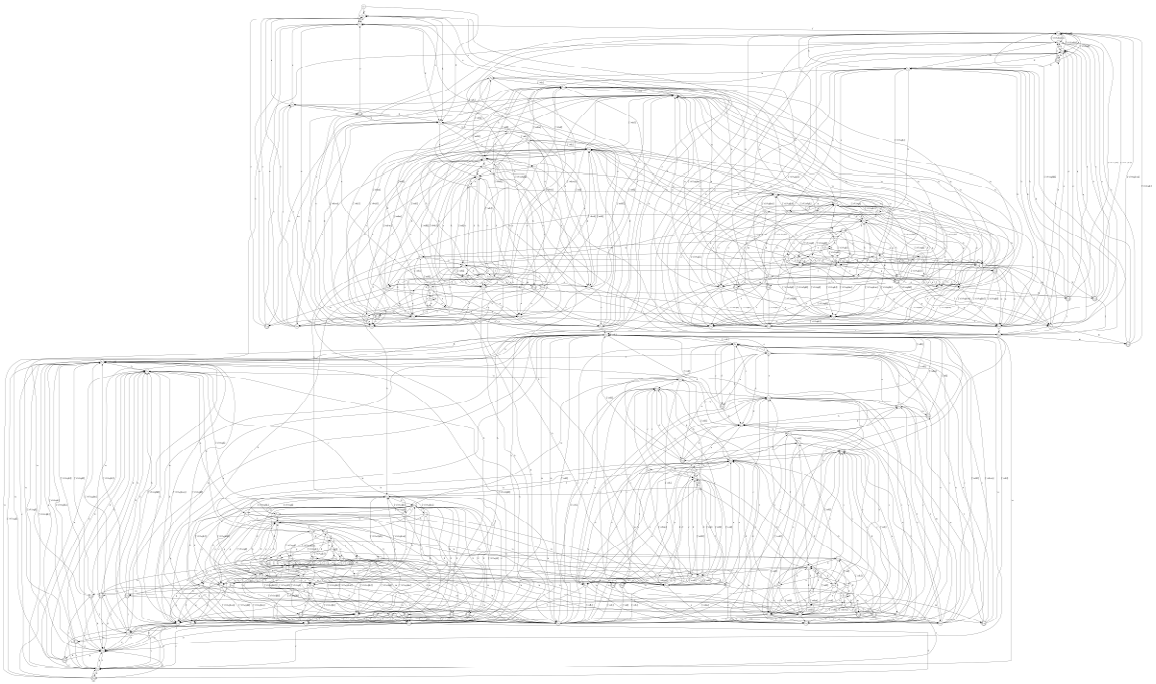


- Certain combinations of regular-expression patterns can cause a "state-explosion" when mapped on a single DFA

- Example:

```
ab.*cd
ef[^\n]*gh
k...lm
```

➔ DFA with 192 states, 1038 transitions

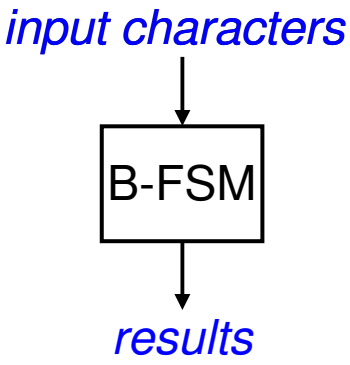


- Certain combinations of regular-expression patterns can cause a "state-explosion" when mapped on a single DFA

- Example:

```
ab.*cd
ef[^\n]*gh
k.{n}lm
```

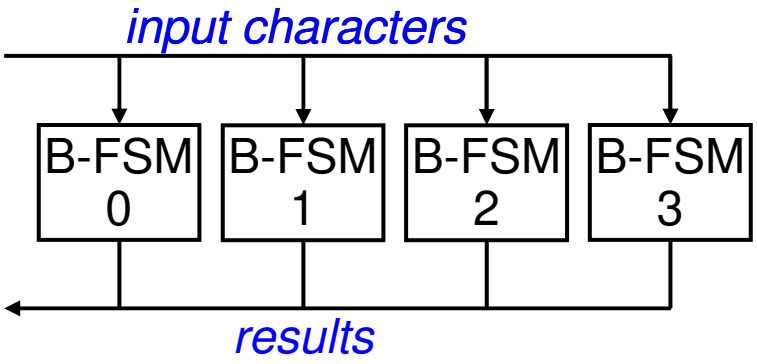
pattern	#states	#transitions
`k.lm`	48	242
`k..lm`	96	508
`k...lm`	192	1038
`k....lm`	384	2098
`k.....lm`	768	4218
`k......lm`	1536	8458
`k.......lm`	3072	16938



B-FSM

- Programmable state machine in HW
- Deterministic rate of one transition per clock cycle @ > 2 GHz
- Storage grows approximately linearly with #transitions

➔ *compact executable representation of given DFAs*



Parallel B-FSMs

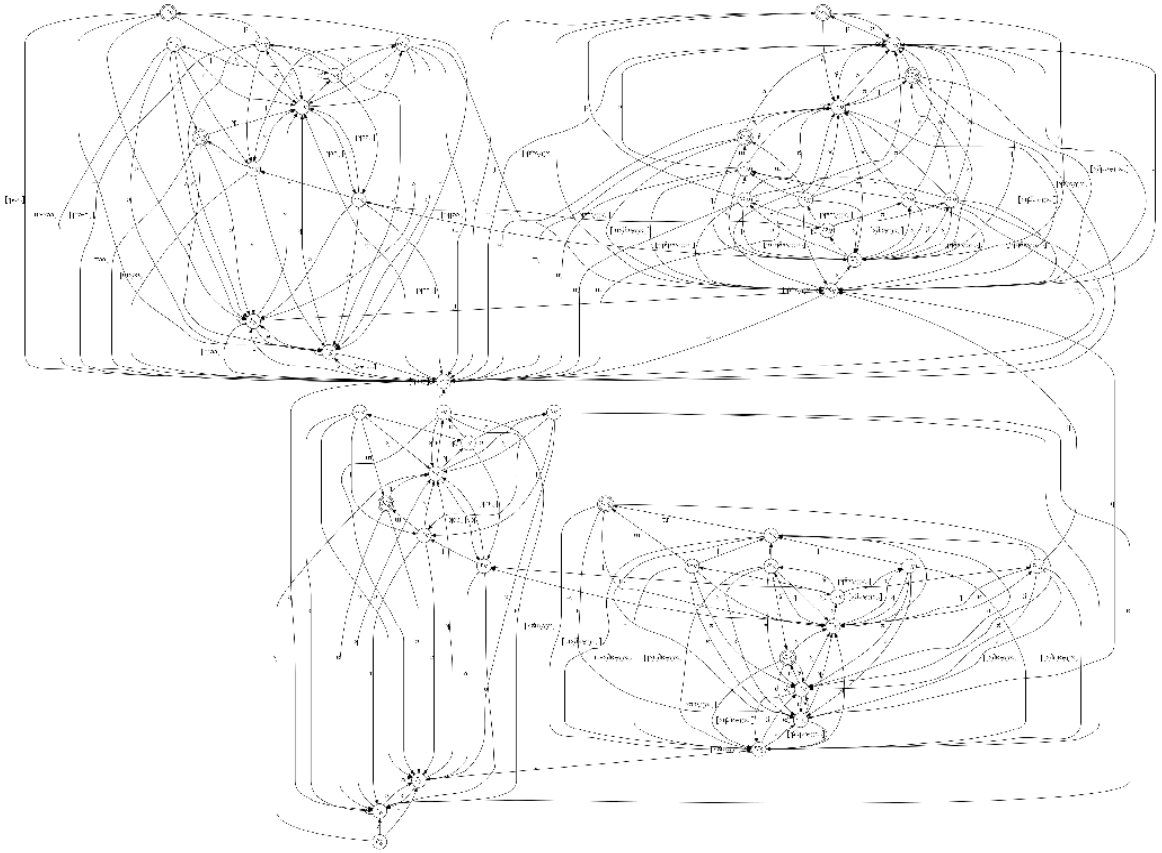
- Enables "NFA-like" storage optimization: multiple parallel states/transitions
- Intelligent distribution of patterns over B-FSMs by compiler
  - separate combinations of patterns that cause "state explosions"

- Trade-offs: additional computation costs and memory accesses, larger session state

- Example:

```
ab.*cd
ef[^\n]*gh
k.lm
```

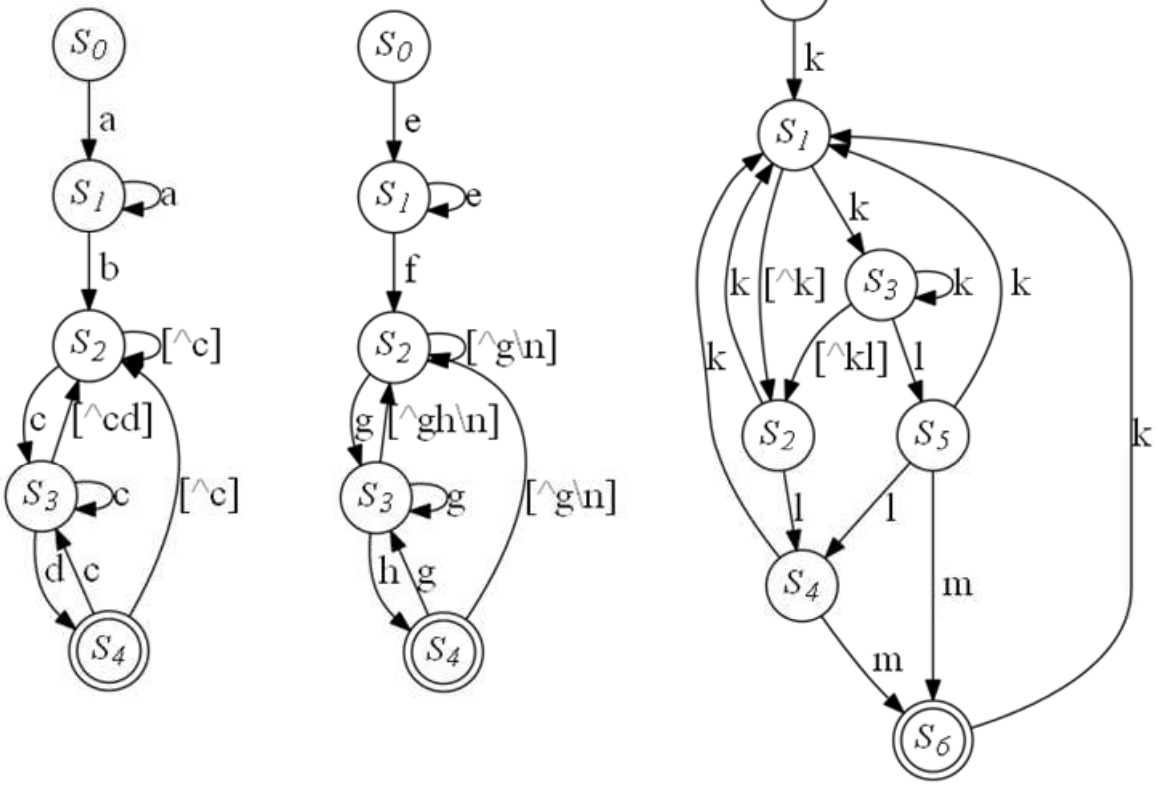
➔ DFA with 48 states, 242 transition rules



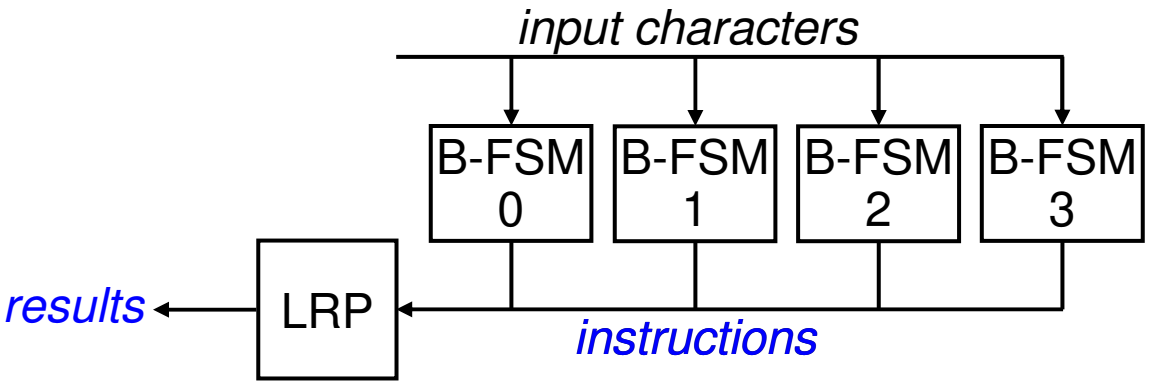
- Example:

```
ab.*cd
ef[^\n]*gh
k.lm
```

➔ 3 DFAs with 17 states, 34 transition rules



DFA 1 *DFA 2* *DFA 3*



Local Result Processor (LRP)

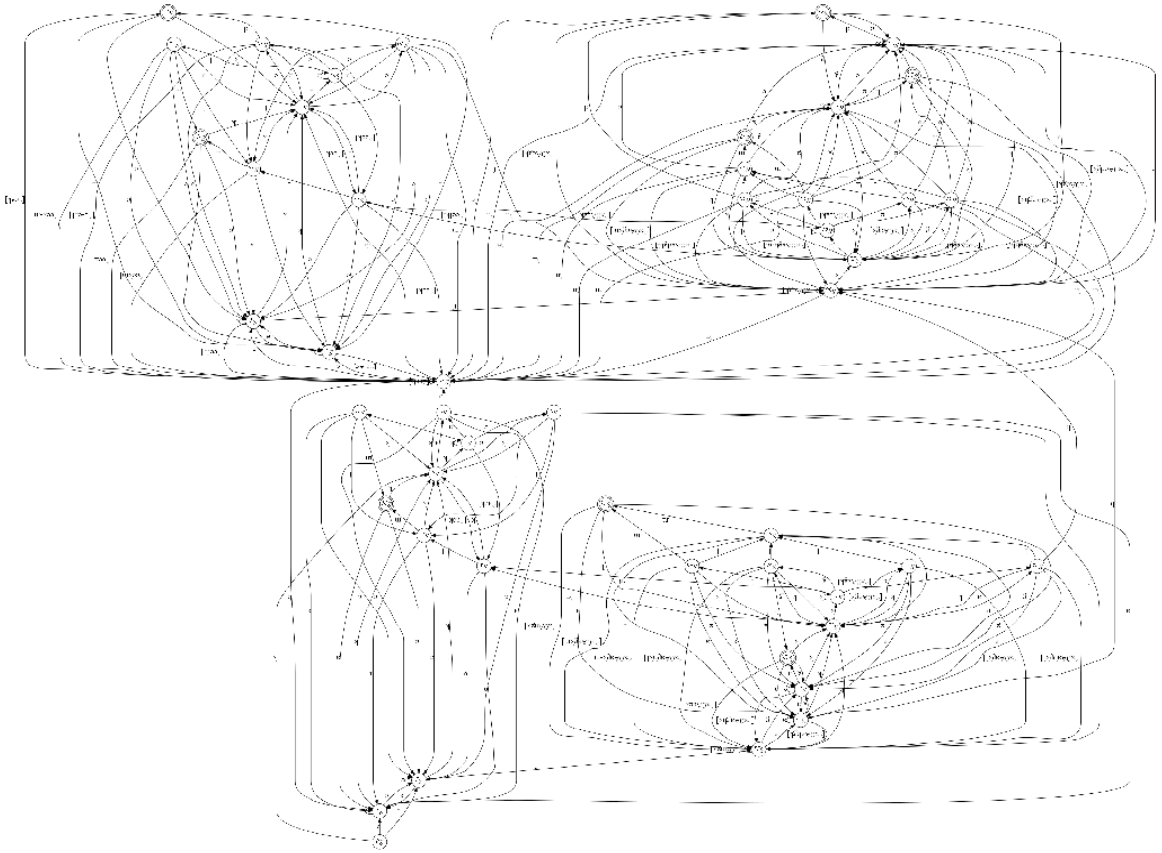
- Storage reduction by off-loading problematic "NFA-states" from B-FSMs
  - split individual problematic patterns into simpler parts
  - LRP checks if parts are detected in the right order, distance, etc.
- LRP provides additional features

- Trade-offs: instructions consume storage, larger session state

- Example:

```
ab.*cd
ef[^\n]*gh
k.lm
```

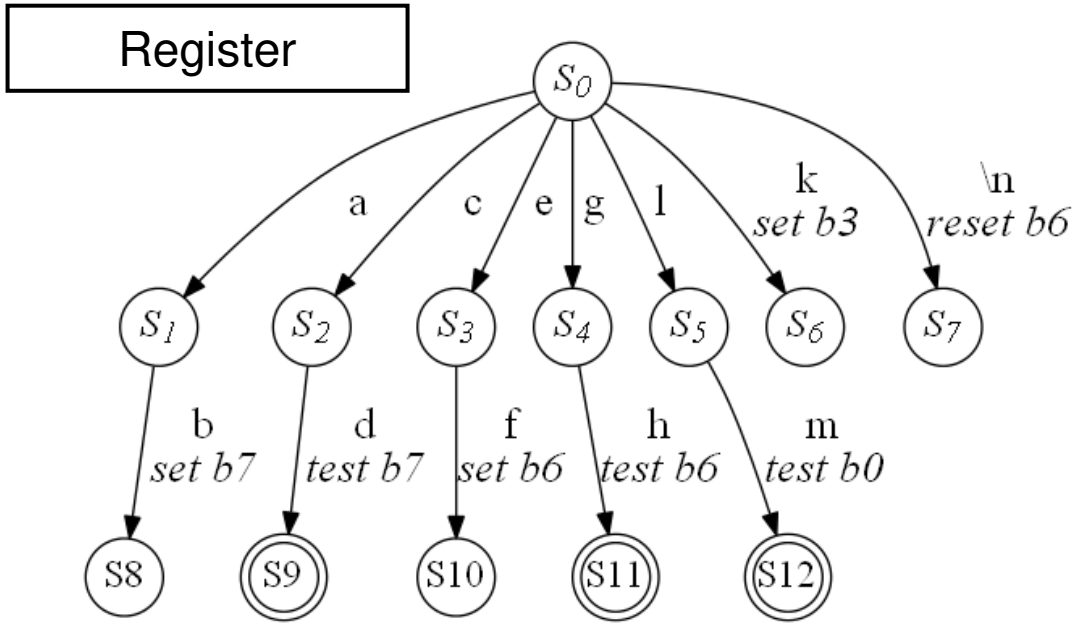
➔ DFA with 48 states, 242 transition rules



- Example:

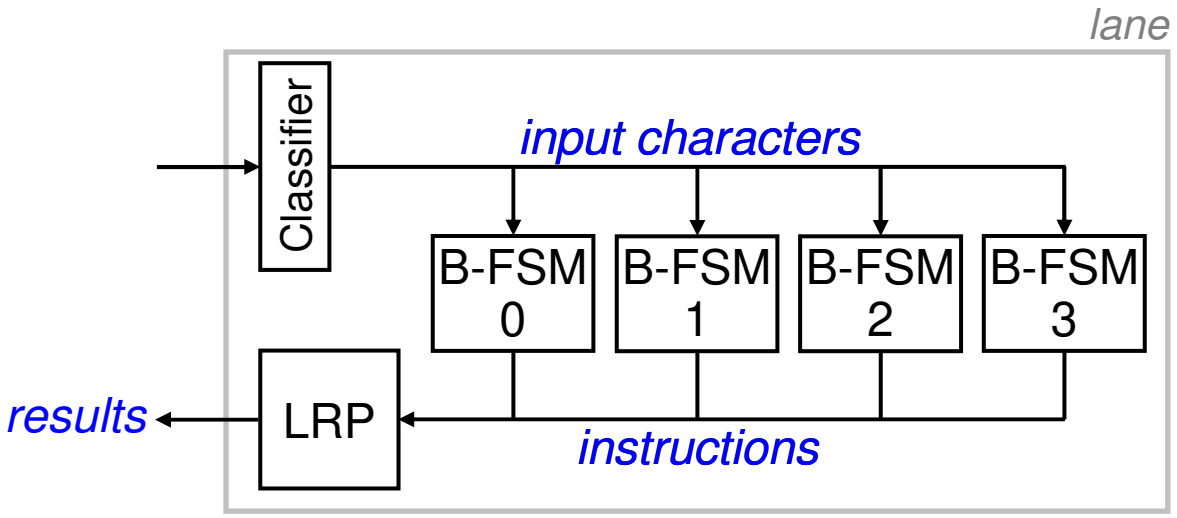
```
ab.*cd
ef[^\n]*gh
k.lm
```

➔ DFA with 13 states,
12 transition rules,
7 instructions



- DFA detects 7 simple patterns:
  ab, **cd**, **ef**, **\n**, **gh**, **k**, **lm**

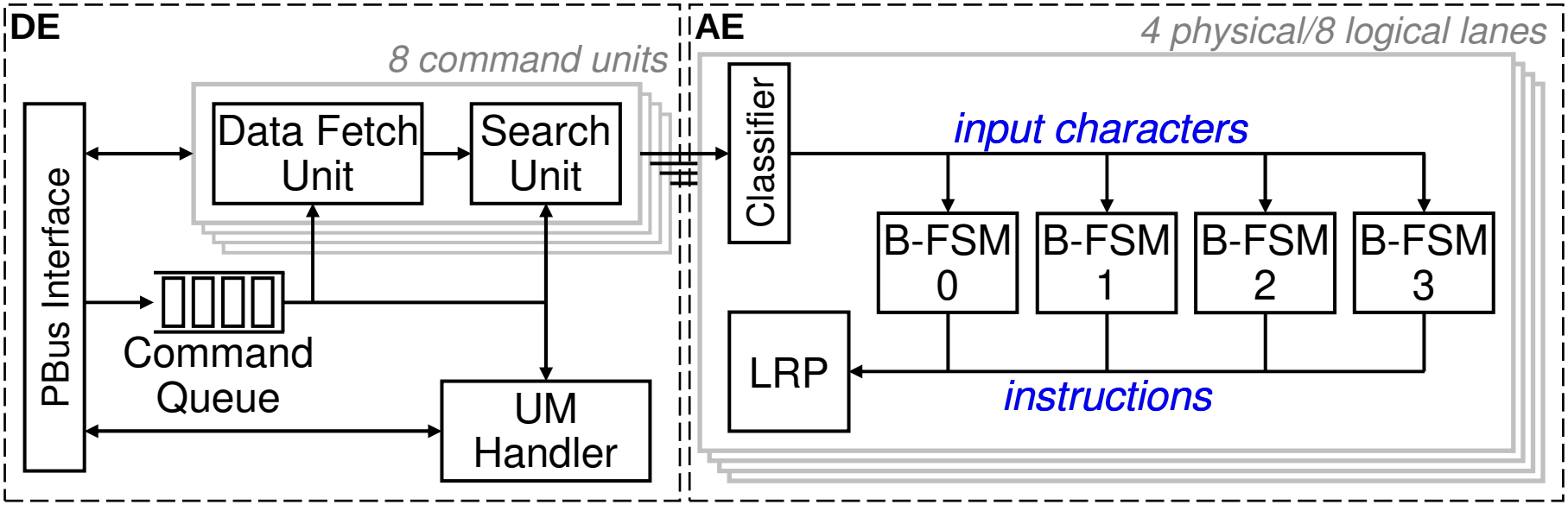
- LRP checks if patterns are found in the right order
  - **cd** after **ab** [bit b7]
  - **gh** after **ef** with no **\n** in between [bit b6]



Lane

- Minimum set of resources allocated for scanning an input stream
- Physical lane implements multiple logical lanes
  - time-interleaved processing of multiple input streams

➔ Can sustain maximum scan rate of one input character per cycle when the B-FSMs process out of transition-rule caches without requiring back-pressure and independent of input characteristics

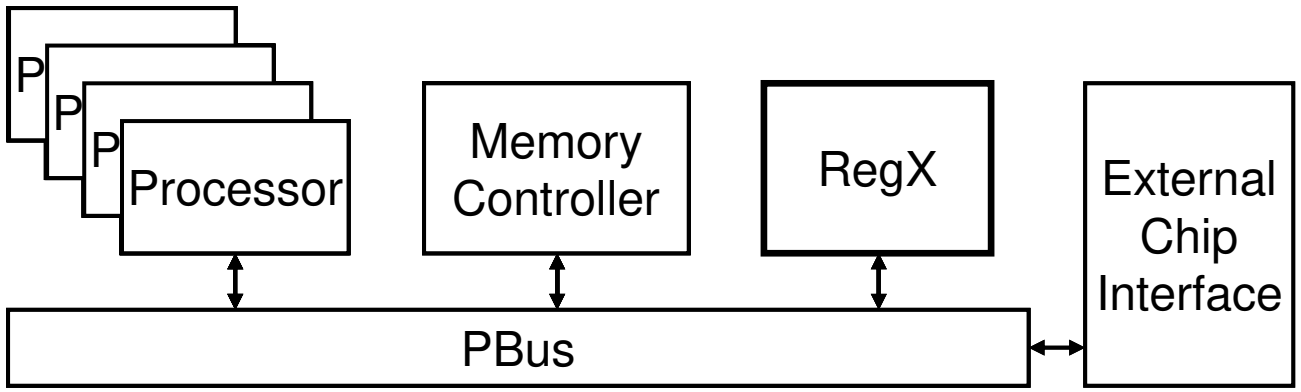


Data Engine

- Enqueues and schedules scan commands
- Fetches and transmits input data streams
- Controls storage and retrieval of scan state, writing of scan results

Algorithmic Engine

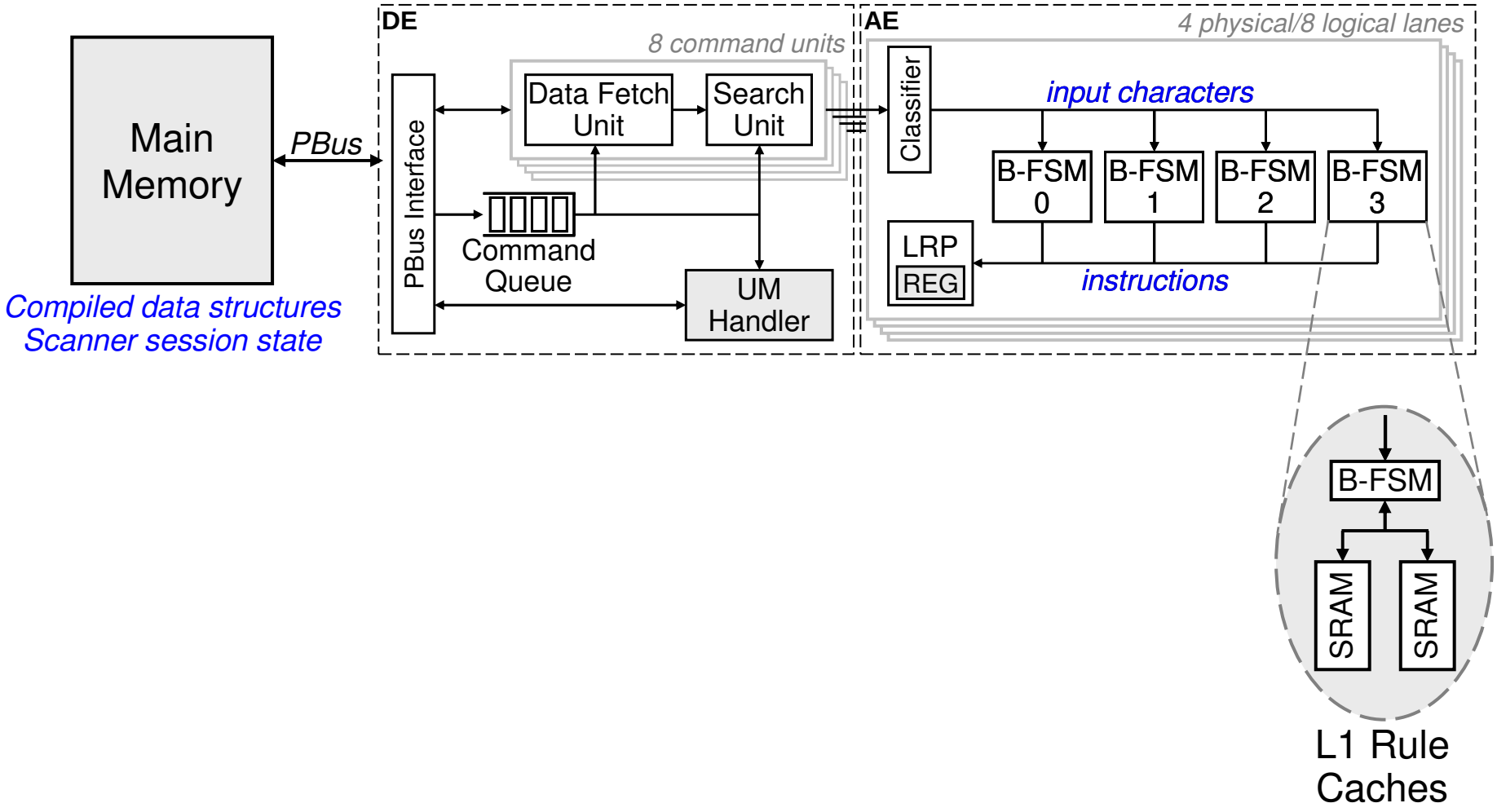
- Scan lanes

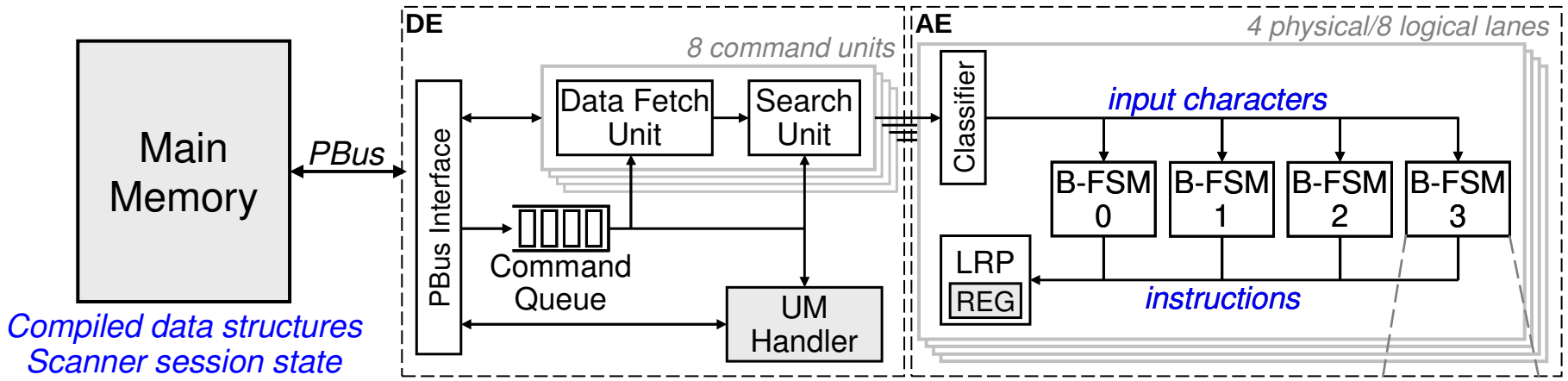


Processor Bus Interface

- Threads running on general-purpose core initiate scan by sending a scan command to RegX which includes pointers to
  - input data
  - compiled pattern set (context)
  - output buffer for storing scan results

Memory Hierarchy





Challenges

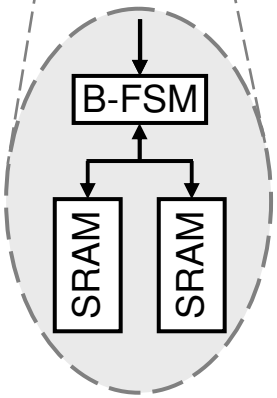
- High access latency to main memory (e.g., 400 cycles)

➔ performance targets require high L1 cache hit rate (>99%)

- Single-cycle access to L1 rule cache required to realize maximum lane scan rate of one input character per cycle

➔ limits number of tags and associativity that can be supported

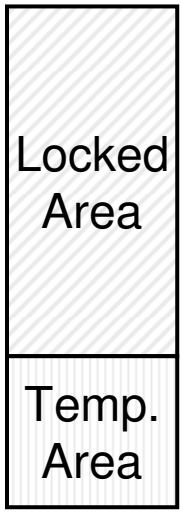
➔ required to achieve high single-stream scan rates



- HW-managed caches do not provide good performance for large target workloads: e.g., two-way set associativity results in a large amount of cache trashing for multiple active pattern contexts

L1 Transition-Rule Cache

1. Locked area
   - managed in SW by Upload Manager (UM)
   - tagless: exposed to B-FSMs as addressable memory area
   - almost fully associative
2. Temporary area
   - 2-way set-associative cache managed in HW
   - caches non-locked transitions upon miss in locked area



L1 Rule Cache

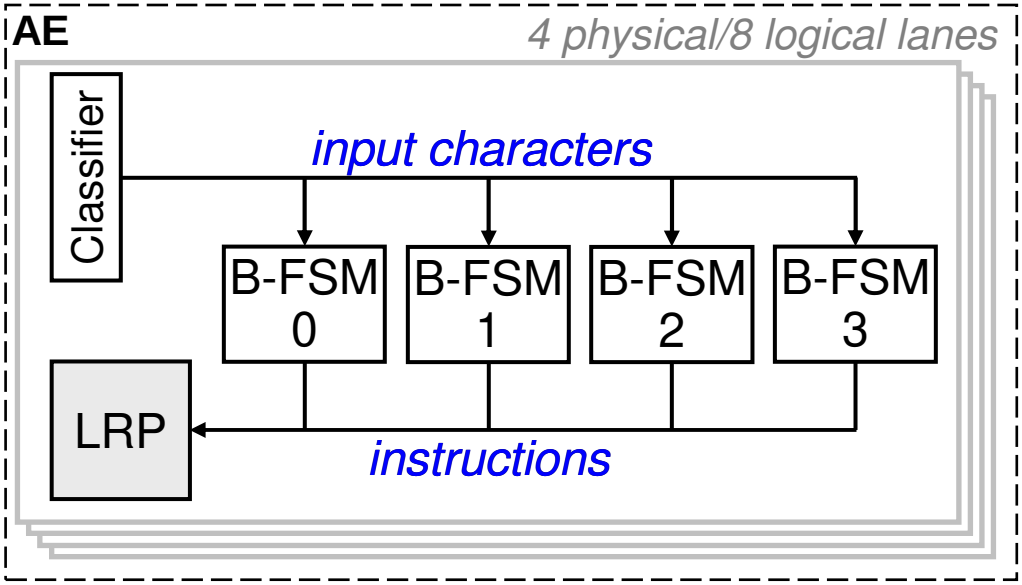
Address translation

- Transitions to locked states are translated to refer directly to physical location in locked area

Upload Manager

- Selection and placement of the most frequently used memory lines (includes B-FSM hash function “adaptation”)
- Driven by a statistical profile of transition access patterns, produced by dedicated HW counters embedded within RegX

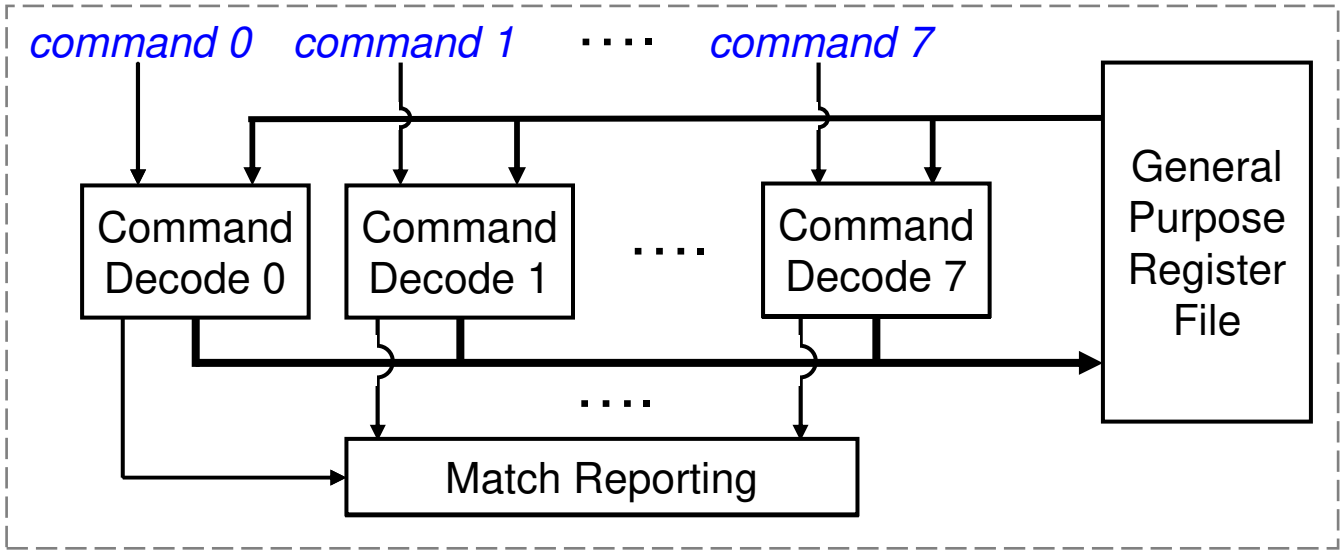
Local Result Processor



Features

- LRP handles eight instructions in parallel in each cycle
- B-FSM can dispatch two instructions per cycle
  - default instructions are triggered by selected character values
  - regular instructions are triggered by the detection of (sub)patterns in the input stream – these are attached to transitions in the DFA
- LRP can sustain peak scan rate when all B-FSMs execute out of the rule caches without requiring a back-pressure mechanism

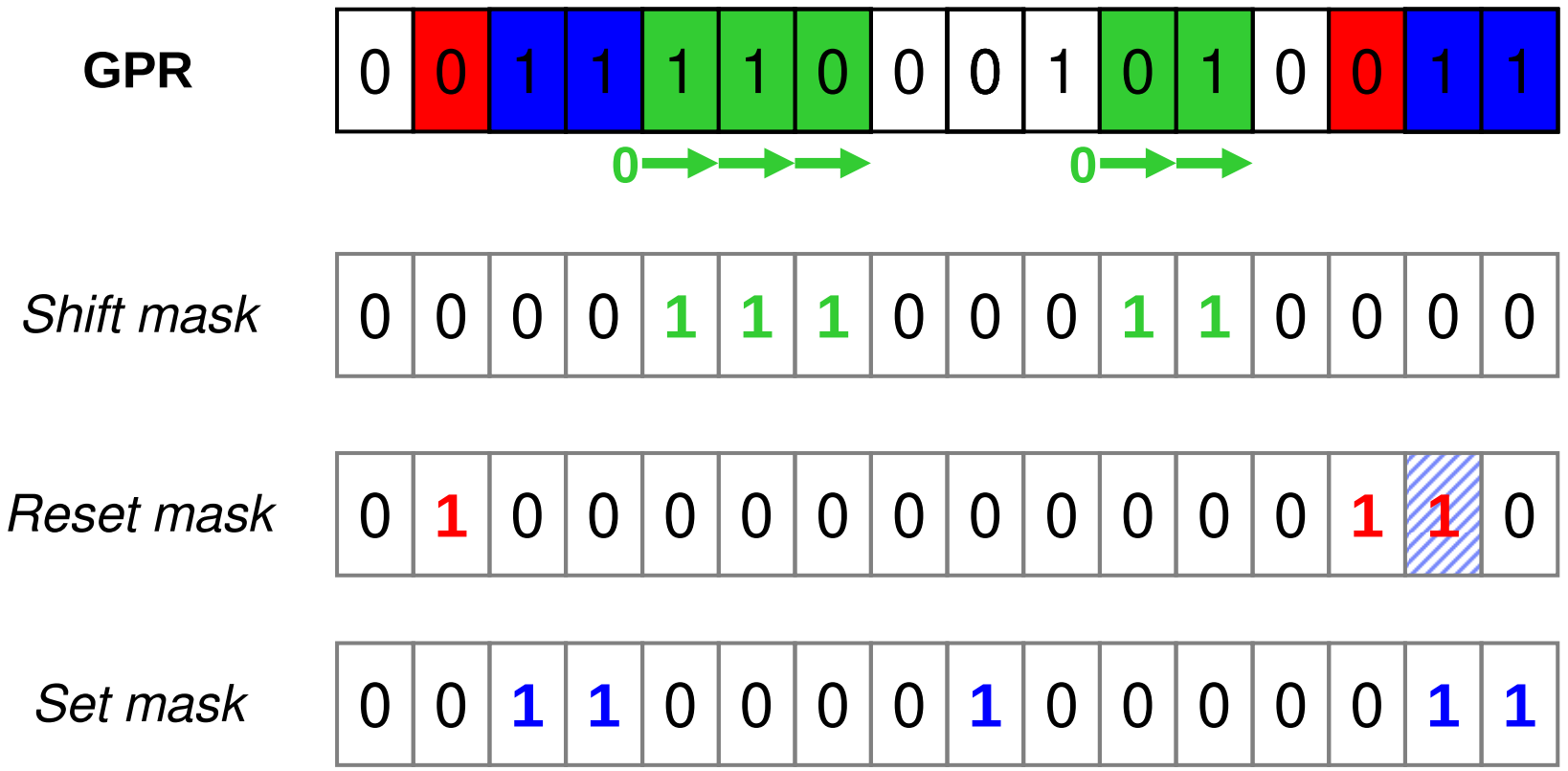
Local Result Processor



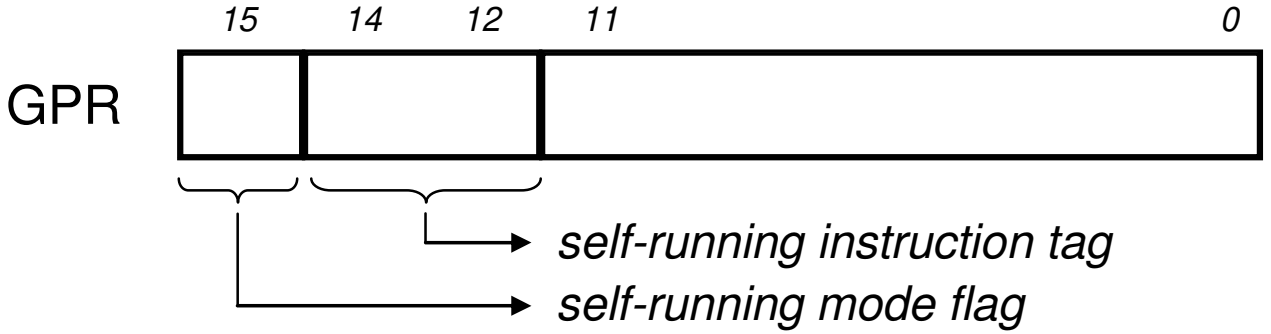
Instructions

- Instructions operate on general-purpose and offset registers
  - examples: set, reset, load immediate, count, shift
  - conditions: selected byte is equal, has all/at least one set bit in common
- Match instructions report detected pattern information to application
  - efficient support for transfer of GPR content for post-processing in SW (SW Result Processor – SRP)

Local Result Processor



- Multiple instructions can operate in parallel on the same register
  - enables efficient allocation of bits to individual patterns
- Priority order when instructions operate on the same bits
  - set, reset, shift, increment (decreasing priority)

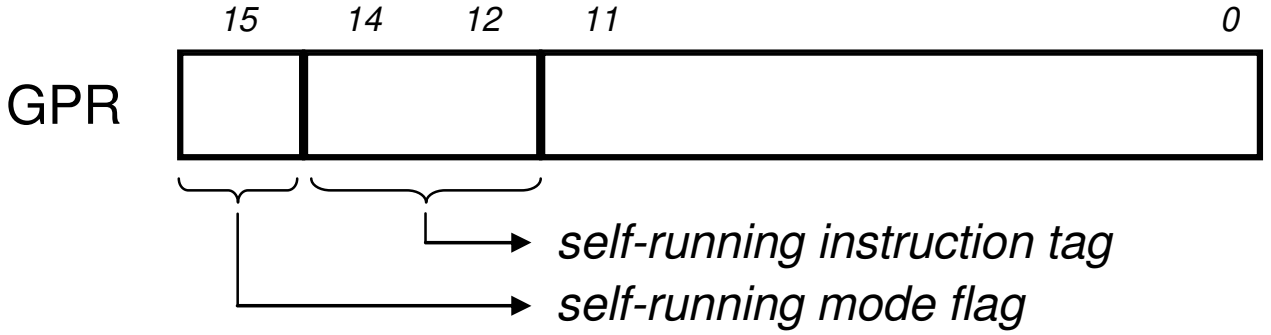


Autonomously Self-Running Instructions

- Self-running instructions enable the execution of selected operations (shift, count) for every input character in certain states
  - allows efficient measuring/testing of distance and length conditions
- Activated by setting self-running mode flag in GPR
  - instruction tag defines operation (normal GPR bits in regular mode)
- Single load operation can (de)activate and configure self-running instruction, and store initial (shift register/counter) contents
- Regular (conditional) instructions can manipulate and test GPR contents

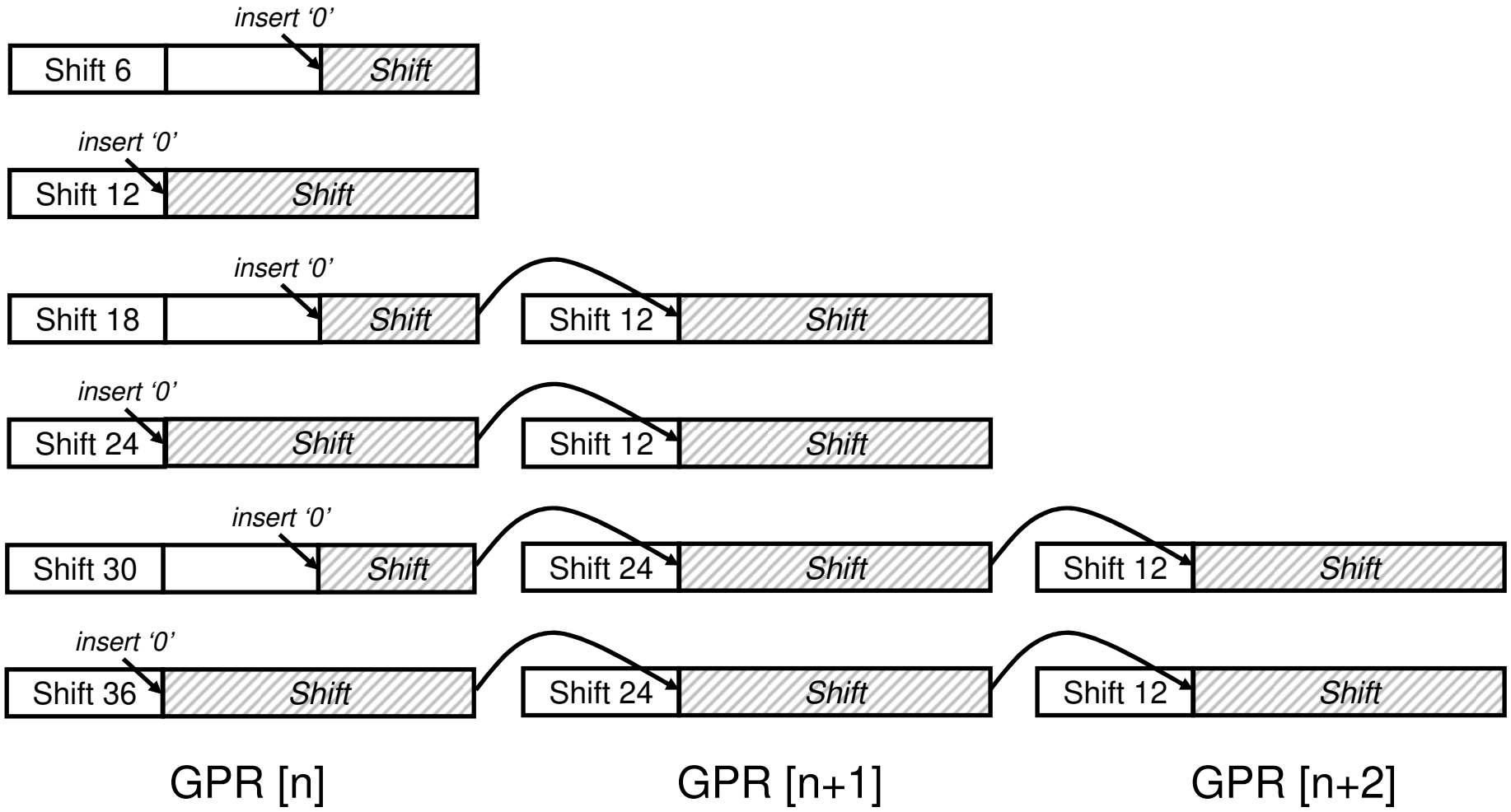
➔ Concept allows flexible and dynamic allocation of register resources as variable-width counters and shift registers

Local Result Processor – *Self-Running Instructions*



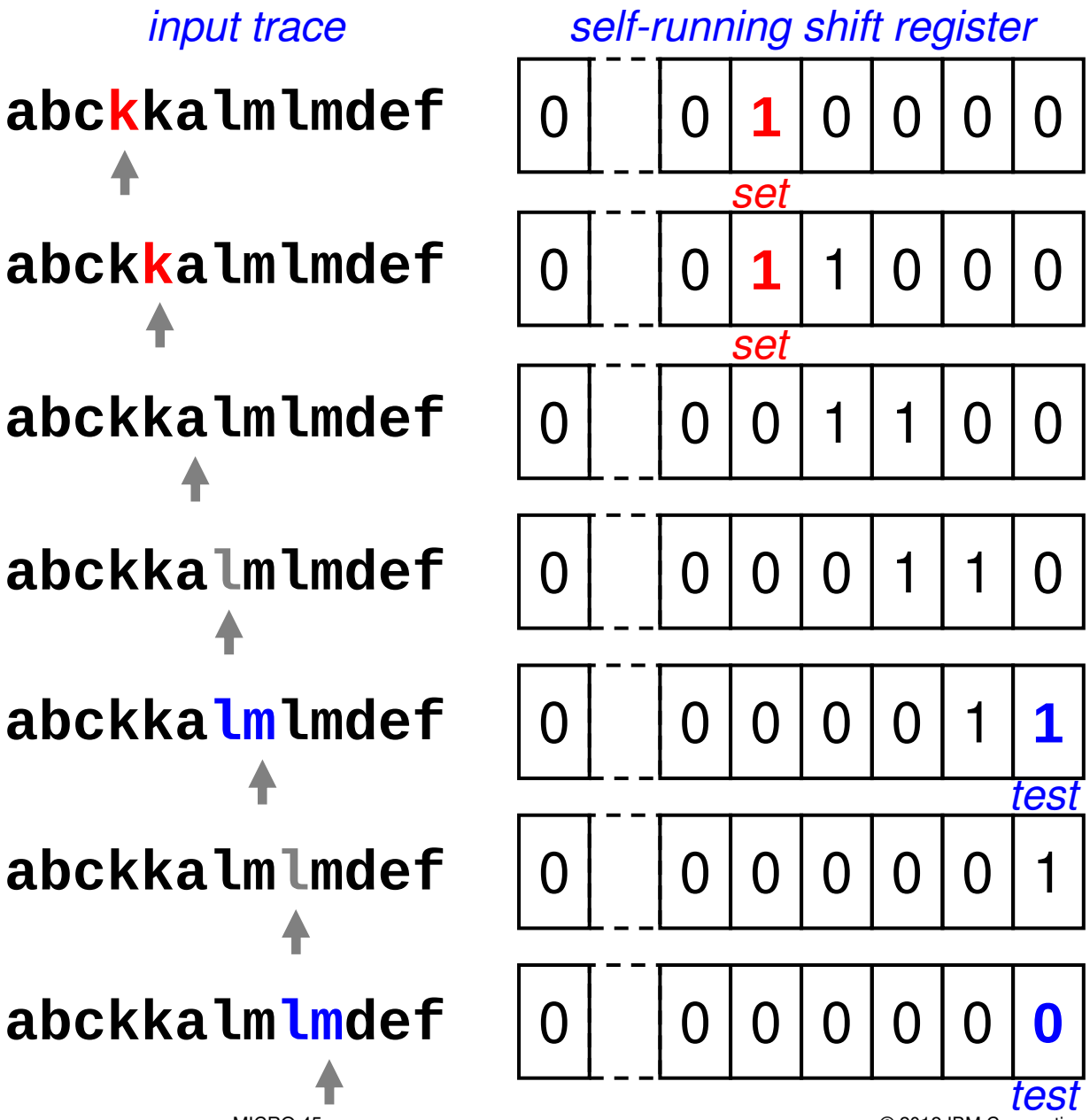
bits 15-12	instruction	bits 15-12	instruction
0xxxb	nop		
1000b	6-bit shift	1101b	30-bit shift
1001b	12-bit shift	1101b	36-bit shift
1010b	18-bit shift	1110b	8-bit counter
1011b	24-bit shift	1111b	12-bit counter

Local Result Processor – *Self-Running Instructions*



Example

- pattern **k..lm**
  - **k** triggers set
  - **lm** triggers test



Pattern Compiler

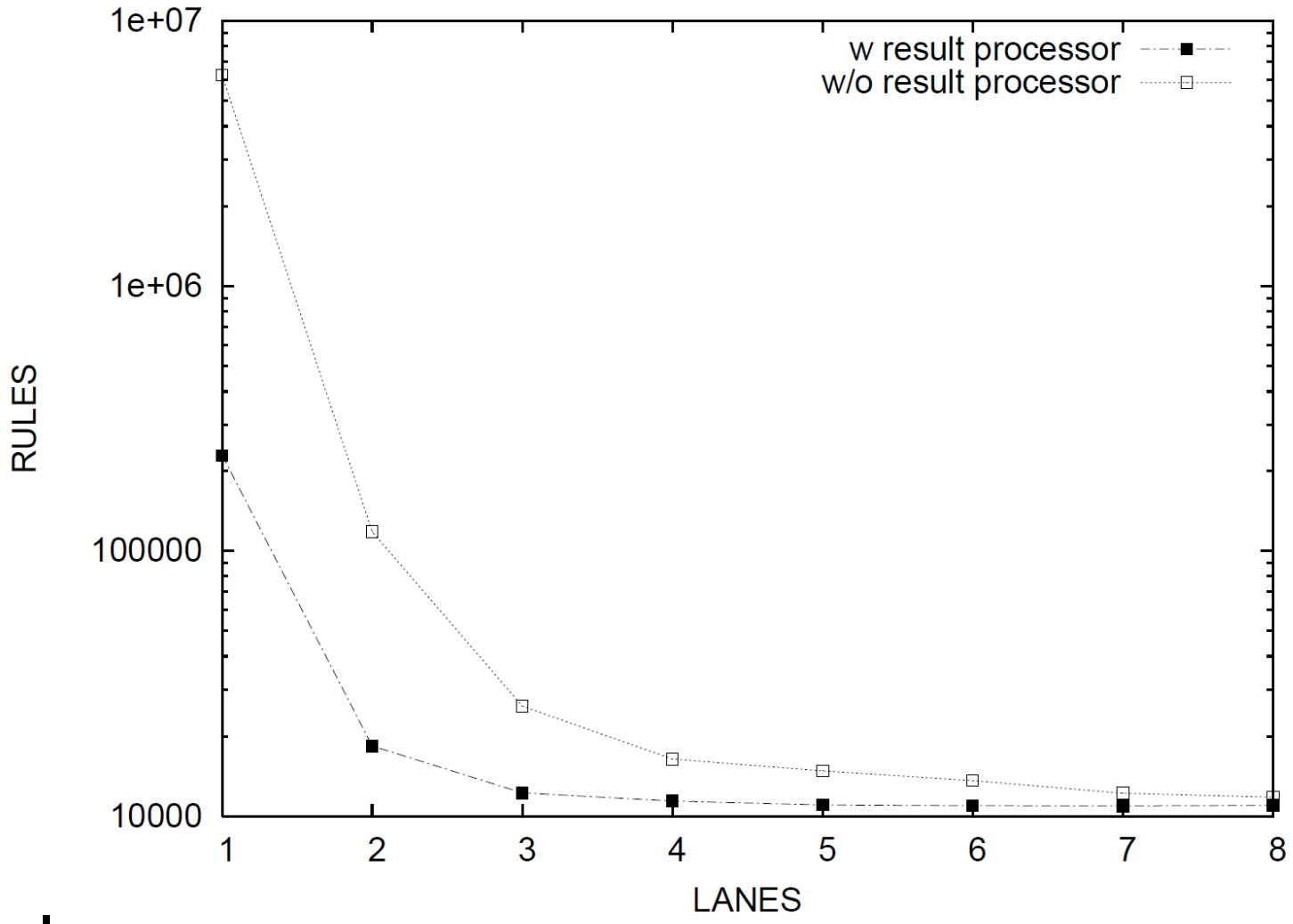
- Converts pattern sets into DFAs
  - pattern splitting, LRP instruction generation and register allocation
- lane count selection
  - distribution of patterns over lanes and B-FSM engines
  - mapping of patterns on DFAs, attachment of LRP instructions

B-FSM Compiler

- Converts DFAs into executable B-FSM data structures
  - construction of linked hash table structures
  - state encoding and instruction integration
  - optimizations to obtain high compression

Incremental and dynamic pattern updates

- the architecture supports incremental updates of the pattern set
- internal scanner data structures can be updated dynamically without interrupting the ongoing scan operations

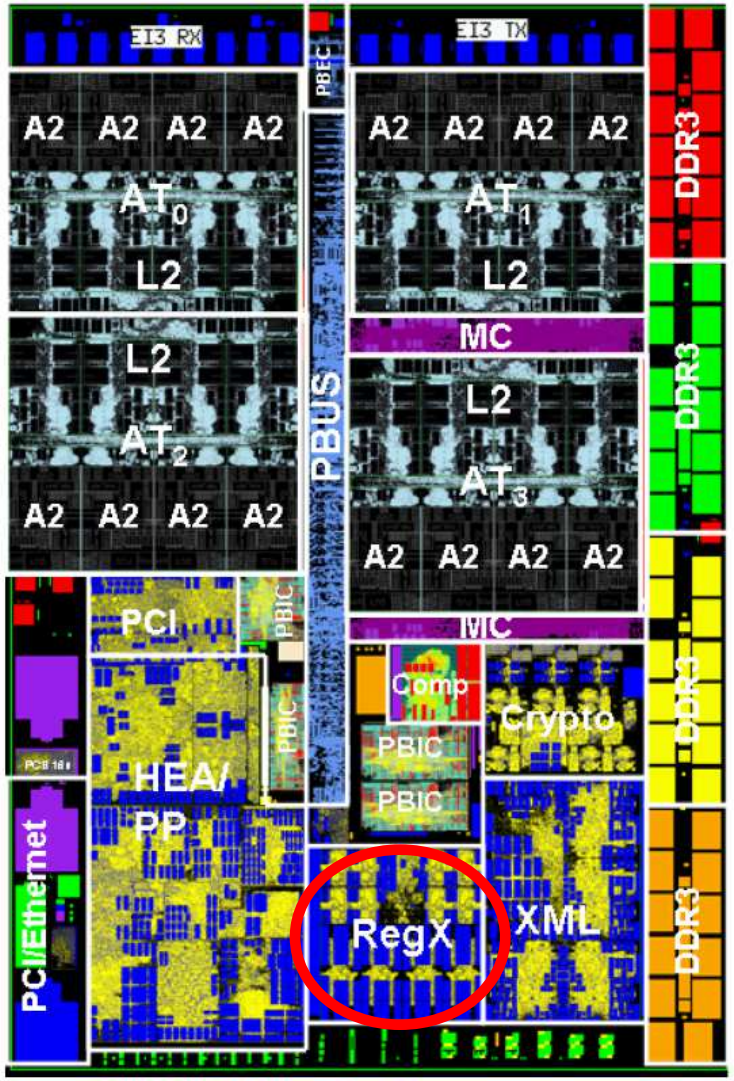


Example

- Storage reduction by exploiting lanes and LRP for publicly available "Application Layer Packet Classifier for Linux" (http://l7-filter.sourceforge.net/)

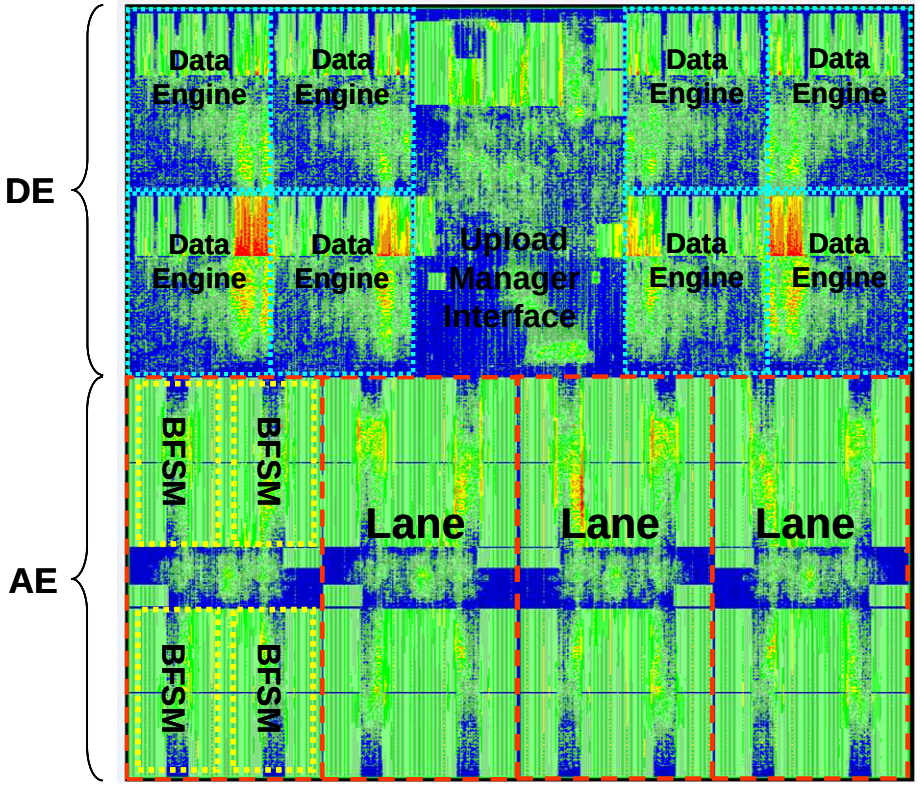
IBM PowerEN™

Technology	IBM 45nm SOI
Core Frequency	2.3GHz @ 0.97V (Worst Case Process)
Chip size	428 mm2 (including kerf)
Chip Power (4-AT node) Chip Power (1-AT node)	65W @ 2.0GHz, 0.85V Max Single Chip 20W @ 1.4GHz, 0.77V Min Single Chip
Main Voltage (VDD)	0.7V to 1.1V
Metal Layers	11 Cu (3-1x, 2-1.3x, 3-2x, 1-4x, 2-10x)
Latch Count	3.2M
Transistor Count	1.43B
A2 Cores / Threads	16 / 64
L1 I & D Cache	16 x (16KB + 16KB) SRAM
L2 Cache	4 x 2MB eDRAM
Hardware Accelerators	Crypto, Compression, RegX, XML
Intelligent Network Interfaces	Host Ethernet Adapter/Packet Processor 2 Modes: Endpoint & Network
Memory Bandwidth	2x DDR3 controllers 4 Channels @ 800-1600MHz
System I/O Bandwidth	4x 10G Ethernet, 2x PCI Gen2
Chip-to-Chip Bandwidth	3 Links, 20GB/s per link
Chip Scaling	4 Chip SMP
Package	50mm FCPBGA (4 or 6 layers)



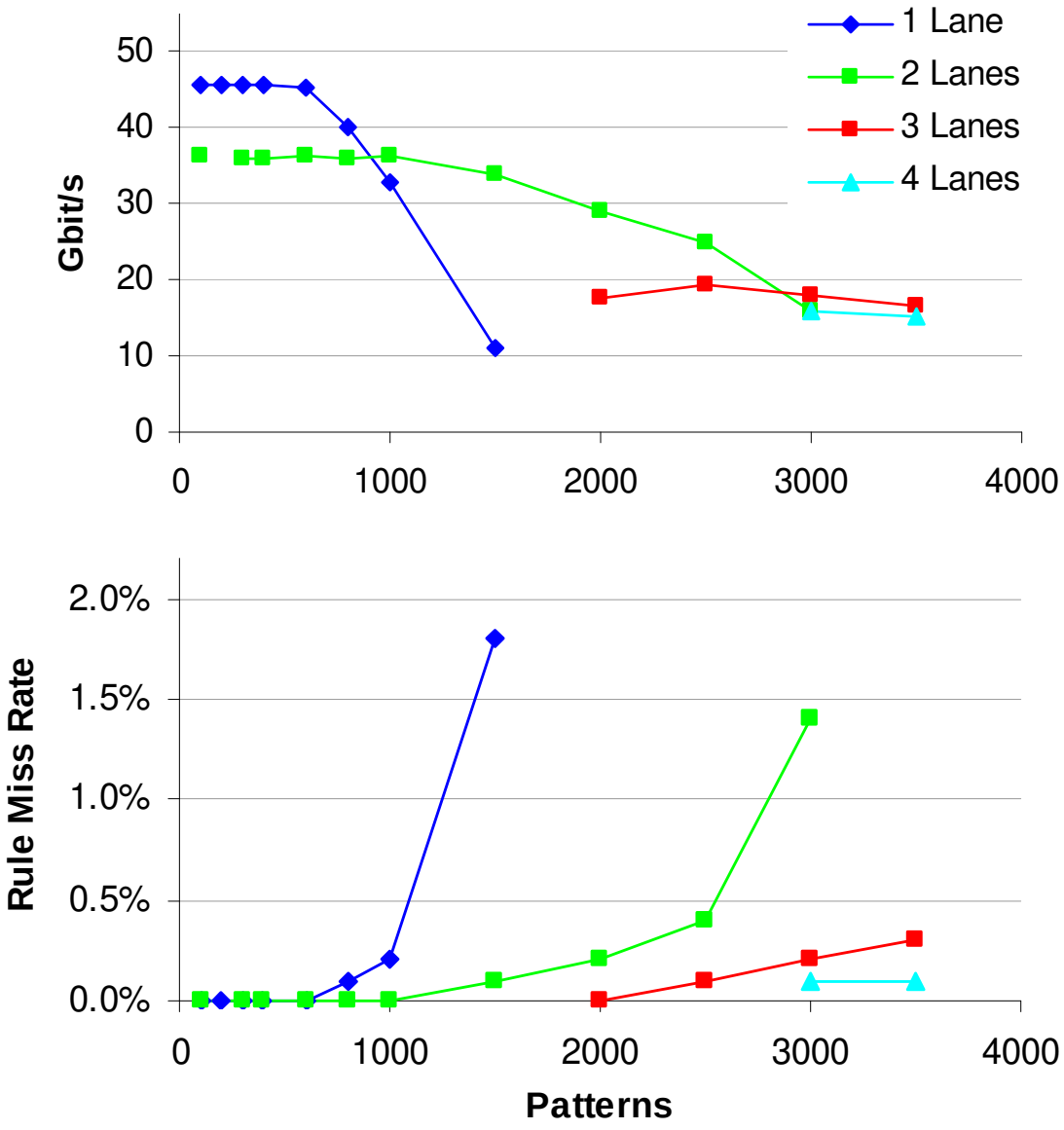
Source: Johnson et al., "A wire-speed power™ processor: 2.3GHz 45nm SOI with 16 cores and 64 threads," ISSCC 2010.

RegX accelerator

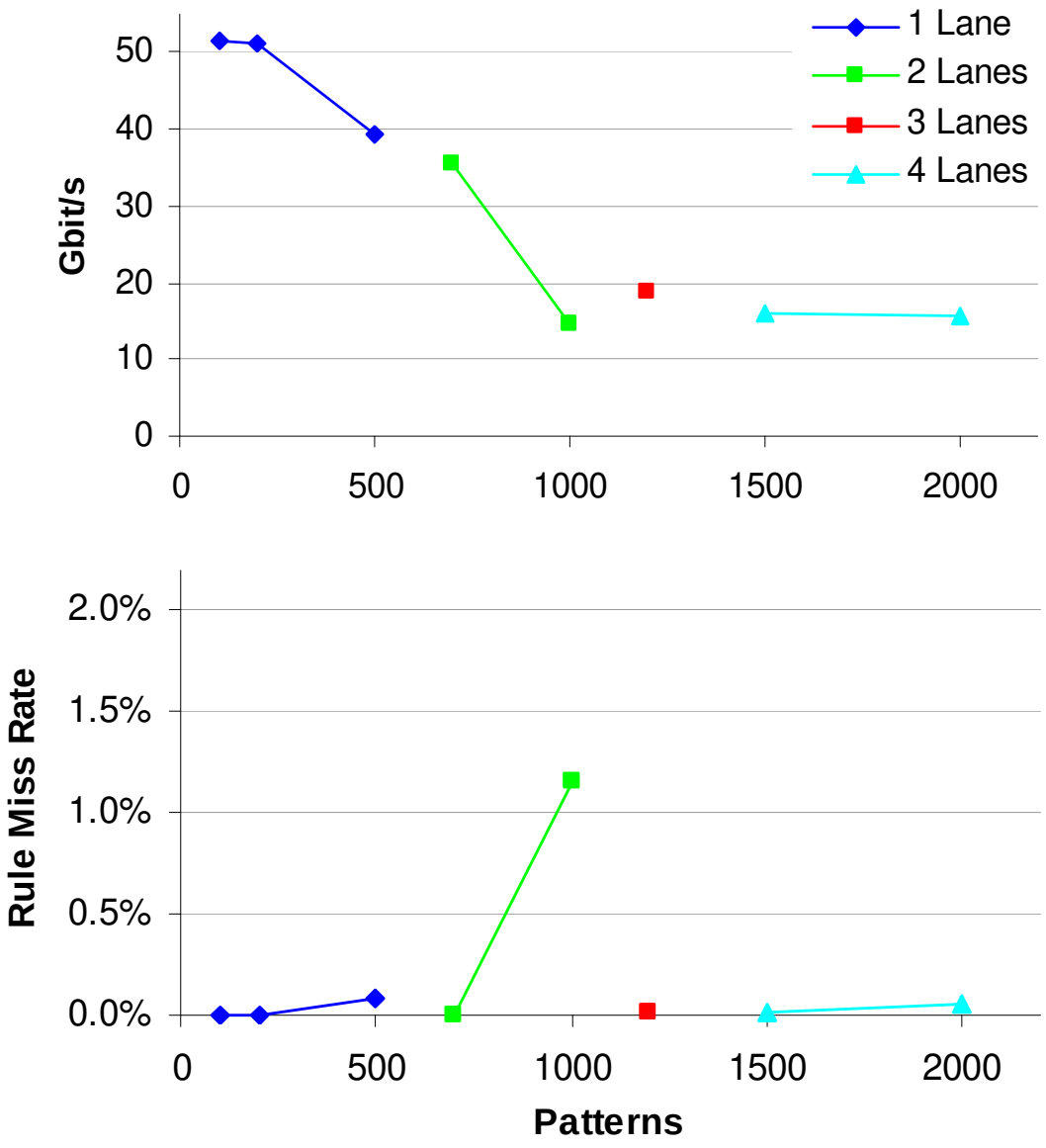


- Implementation
  - area: 15.4 mm^2
  - clocked at 2.3 GHz (1.15 GHz)
- Lanes
  - 4 physical (16 B-FSMs)
  - 8 logical (32 logical B-FSMs)
- Memory
  - 32 KB rule cache per B-FSM
  - 512 KB rule cache in total
  - LRP register file: 8 x 16 bit, 128 bits total
- Peak scan rates (data structure fits entirely in rule caches)
  - lane: one byte/cycle = 18.4 Gbit/s
  - stream: one byte/2 cycles = 9.2 Gbit/s
  - theoretical peak rate for 4 lanes: 73.6 Gbit/s

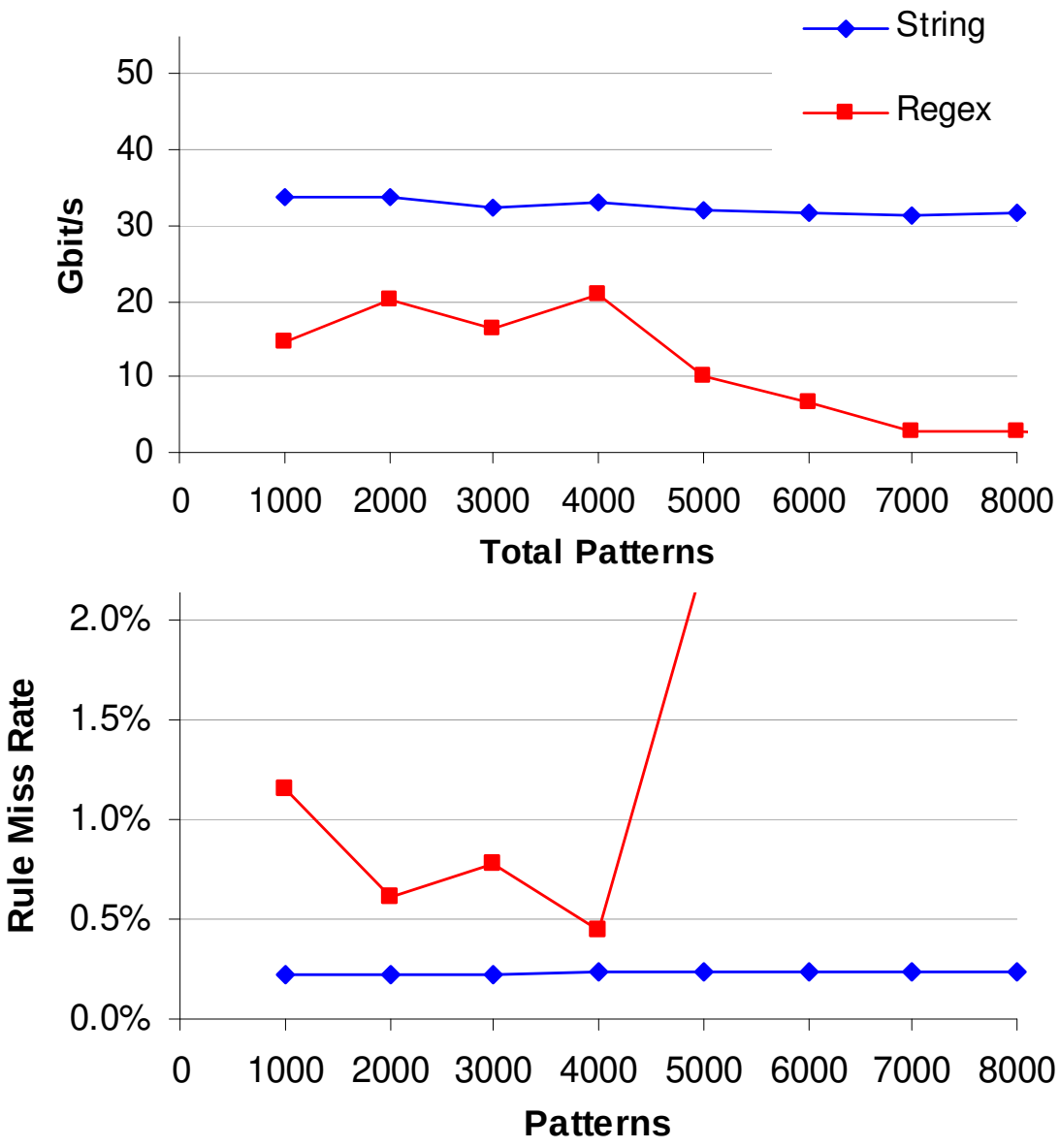
HW Measured Performance – *Single Context / String Patterns*



HW Measured Performance – *Single Context / Regex Patterns*



HW Measured Performance – *Multiple Contexts*



Conclusions

RegX accelerator

- An architecture, implementation, compiler and upload manager were designed to realize scan rates in the range of 15-40 Gbit/s for typical intrusion detection workloads
- Novel (micro-)architectural features
  - Local Result Processor design supporting eight instructions fully in parallel and the concept of self-running instructions
  - transition-rule caches, including address translation and cache line placement by a SW application exploiting hardware based profiling
  - physical/logical lane concept involving a time-interleaved processing of multiple data streams by B-FSMs and LRPs, sustaining the maximum scan rate when processing out of the rule caches, without requiring a back-pressure mechanism and independently of input characteristics

Future work

- RegX is part of our research towards more general-purpose accelerators

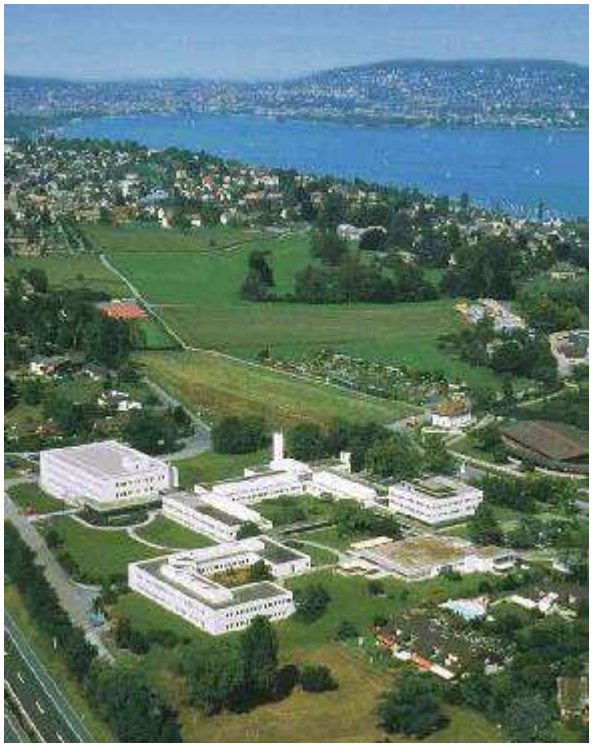
➔ this project has shown us that basic building blocks of such an accelerator are feasible at clock frequencies beyond 2 GHz

For more information, please contact:

Jan van Lunteren (jvl@zurich.ibm.com)

IBM Research - Zurich
Säumerstrasse 4
CH-8803 Rüschlikon
Switzerland

Phone: +41 44 724 8111
Fax: +41 44 724 8911



Backup

Backup