



Zurich Research Laboratory

# A Novel Processor Architecture for High-Performance Stream Processing

Jan van Lunteren


# Agenda

1. Introduction
2. High-Level Concept
3. Programmable State Machine
4. Novel Processor
5. Instruction Cache and Prefetch
6. Experimental Results
7. Summary

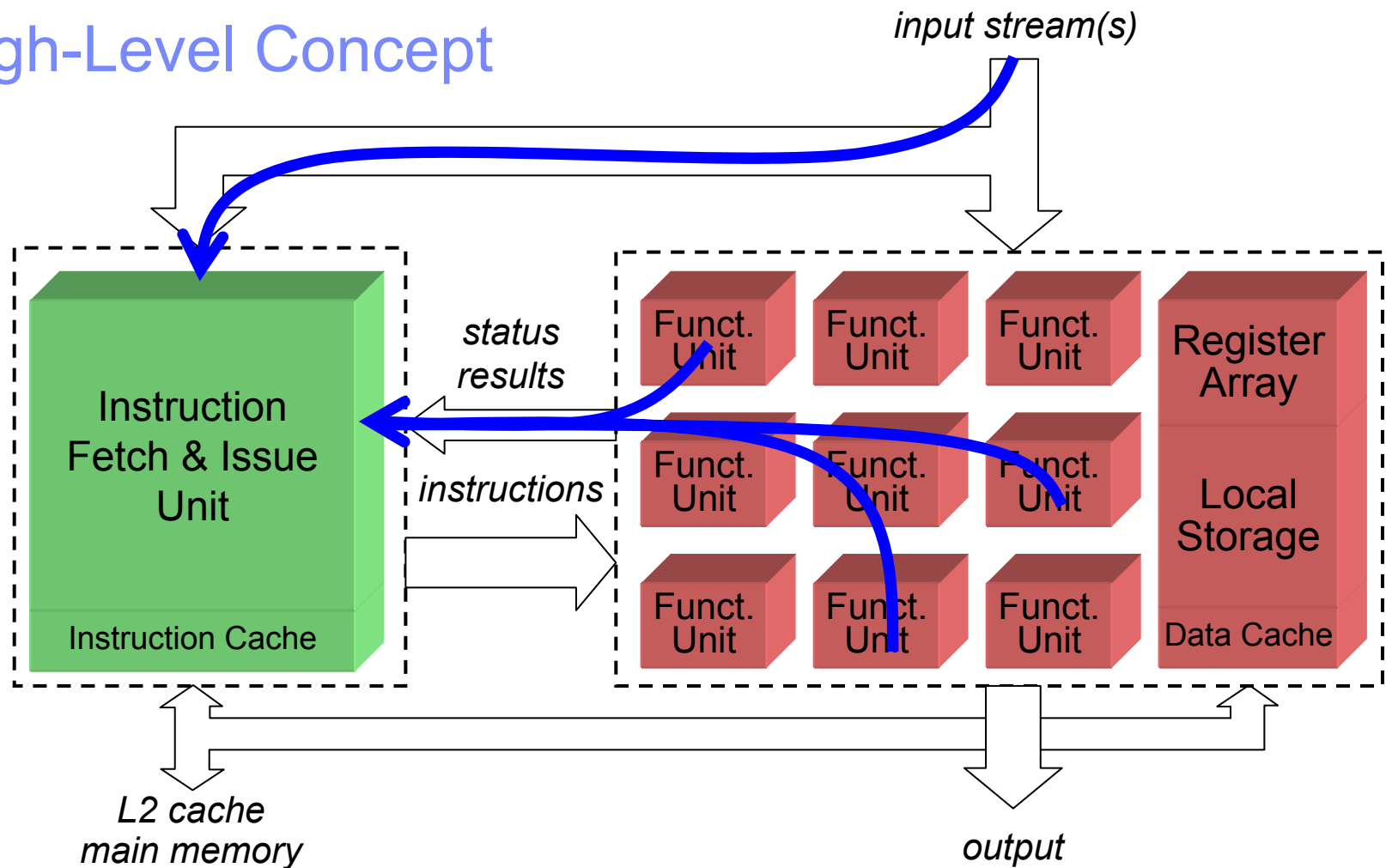
# Introduction

- At the IBM Zurich Research Laboratory, we have started a research project that examines opportunities for **novel processor concepts** which deviate from the traditional (“Von Neumann”) processor architecture
- The objective is to realize a new type of **programmable** “general-purpose” coprocessor that is optimized for applications that run into **performance** and **power** problems on conventional processors
- Initial focus is on accelerating applications that operate on **streams of data** such as XML processing, compression, pattern matching (IDS), encryption and networking
- This project was triggered by current trends in the processor industry, where physical limitations and power issues force a shift towards increasingly parallel processing on multi-core architectures

## High-Level Concept

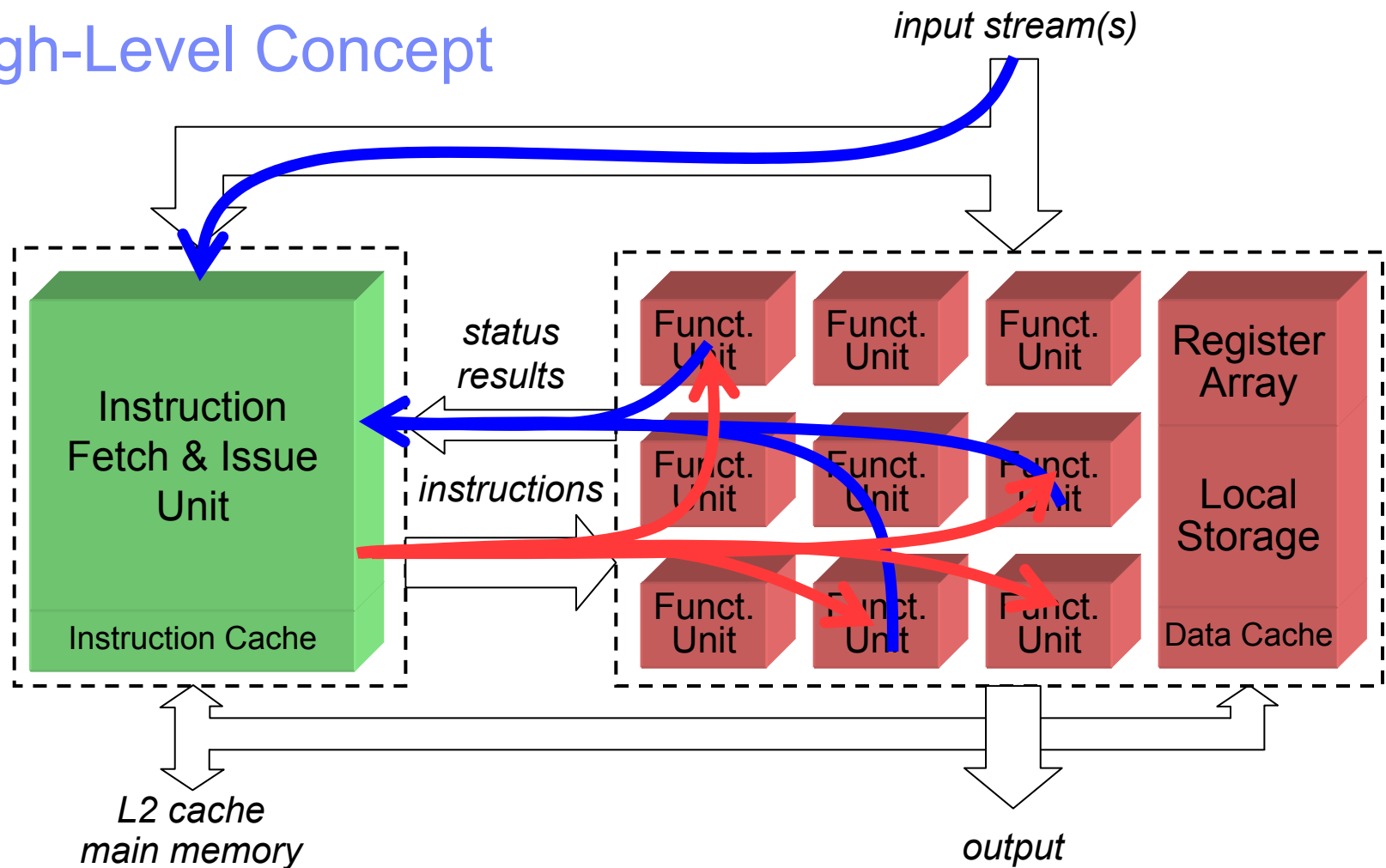
- VLIW type of processor architecture comprised of a large number of diverse functional units, ranging from ALUs to complex search engines
  - large variety in execution times and amount of data processed by functional units
- Targets: clock frequency 1-2 GHz, data processing rate >10 Gb/s
-  Challenge: flexible programmability combined with high performance
- Novel approach for instruction fetch and issue
  - enables dynamic scheduling of collections of functional units to operate in pipelined and parallel fashion on input data
  - powerful conditional branch capabilities
    - one multi-way branch involving up to 256 targets in each cycle
    - evaluation of multiple and complex conditions for each branch
    - direct testing of input data

# High-Level Concept



- In each clock cycle, the instruction fetch & issue unit monitors the current input value and the status/results of the functional units

# High-Level Concept



- In response, it will dispatch instructions to selected (or all) functional units within one clock cycle

# High-Level Concept

## Conventional processor (“Von Neumann”)

- Instructions are identified and selected for execution based on their **addresses**
- Basic instruction execution flow is **sequential** (program counter is incremented by default)
- Conditional branches allow the instruction execution flow to be changed by evaluating typically **one simple condition** (e.g., greater than, less than, equal to)

```
addr      instruction
addr+4    instruction
addr+8    instruction
addr+12   instruction
addr+16   branch if zero +20
...
addr+36   instruction
addr+40   instruction
addr+44   instruction
addr+48   instruction
addr+52   instruction
```

- 👉 Multiple and/or more complex conditions have to be translated into a sequence of several instructions and conditional branches

*Example:* “Is the input character a legal name character in a given programming or markup language?”

# High-Level Concept

## Novel processor – general concept

- Instructions are associated with one or multiple **conditions**
- In each cycle, all conditions for a current instruction group are evaluated simultaneously – the highest-priority instruction for which all conditions match is selected
- Special “branch” instructions allow a jump to a different group of instructions


conditions  
evaluated  
in parallel

conditions  
evaluated  
in parallel

```

group_1:
  conditions instruction
  conditions instruction
  conditions instruction
  conditions branch group_2
  conditions instruction
  conditions instruction
  default instruction

group_2:
  conditions instruction
  conditions instruction
  conditions instruction
  conditions instruction
  conditions instruction
  .....
  
```

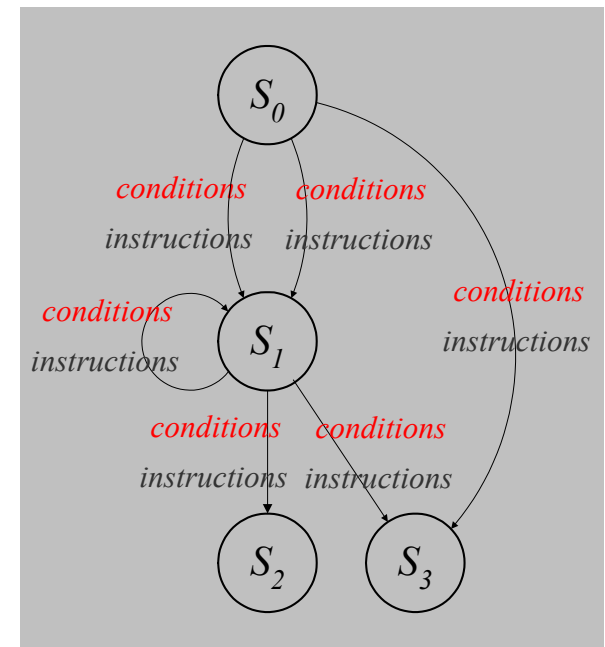
-  Because the execution of instructions will typically affect the evaluation of the conditions in subsequent cycles, this will determine the actual instruction execution flow



# High-Level Concept

## Novel processor – “embodiment”

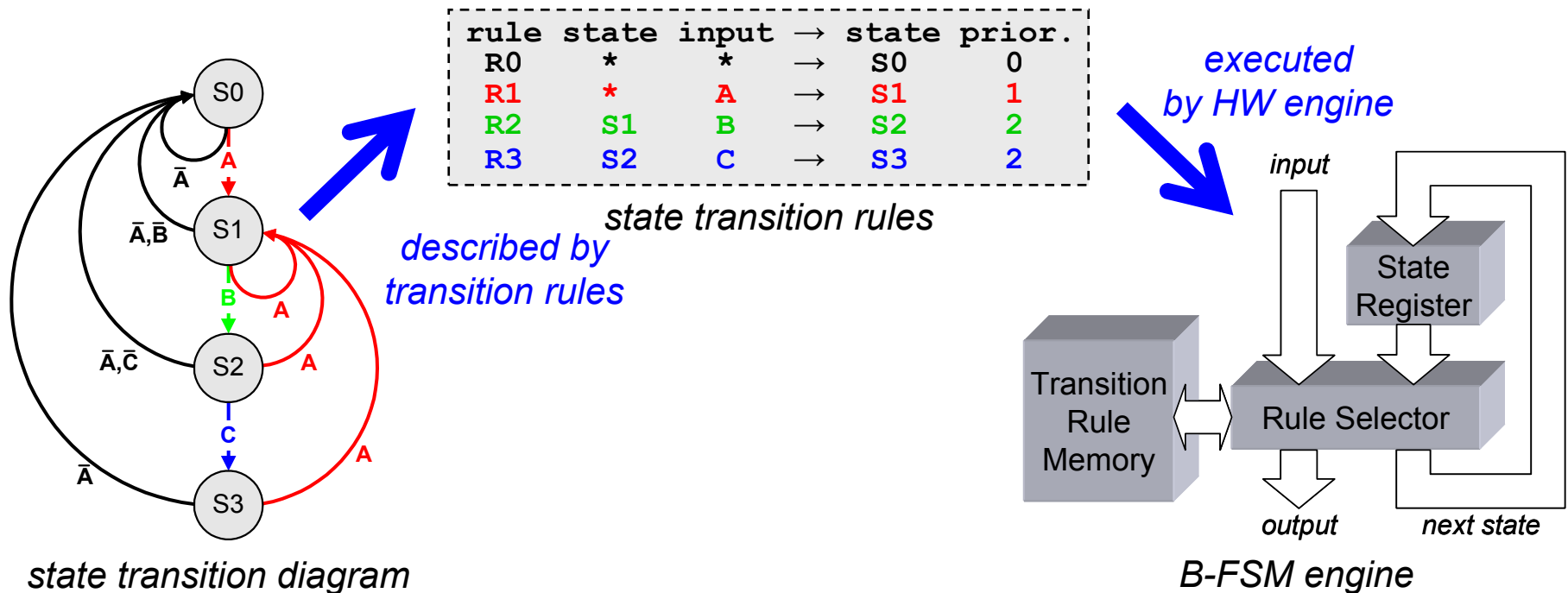
- Enhanced **state-transition diagram** defines numerous potential execution paths
- Actual path taken during program execution is determined by real-time evaluation of various sets of conditions associated with the state transitions
- Instructions associated with selected path are dispatched for execution



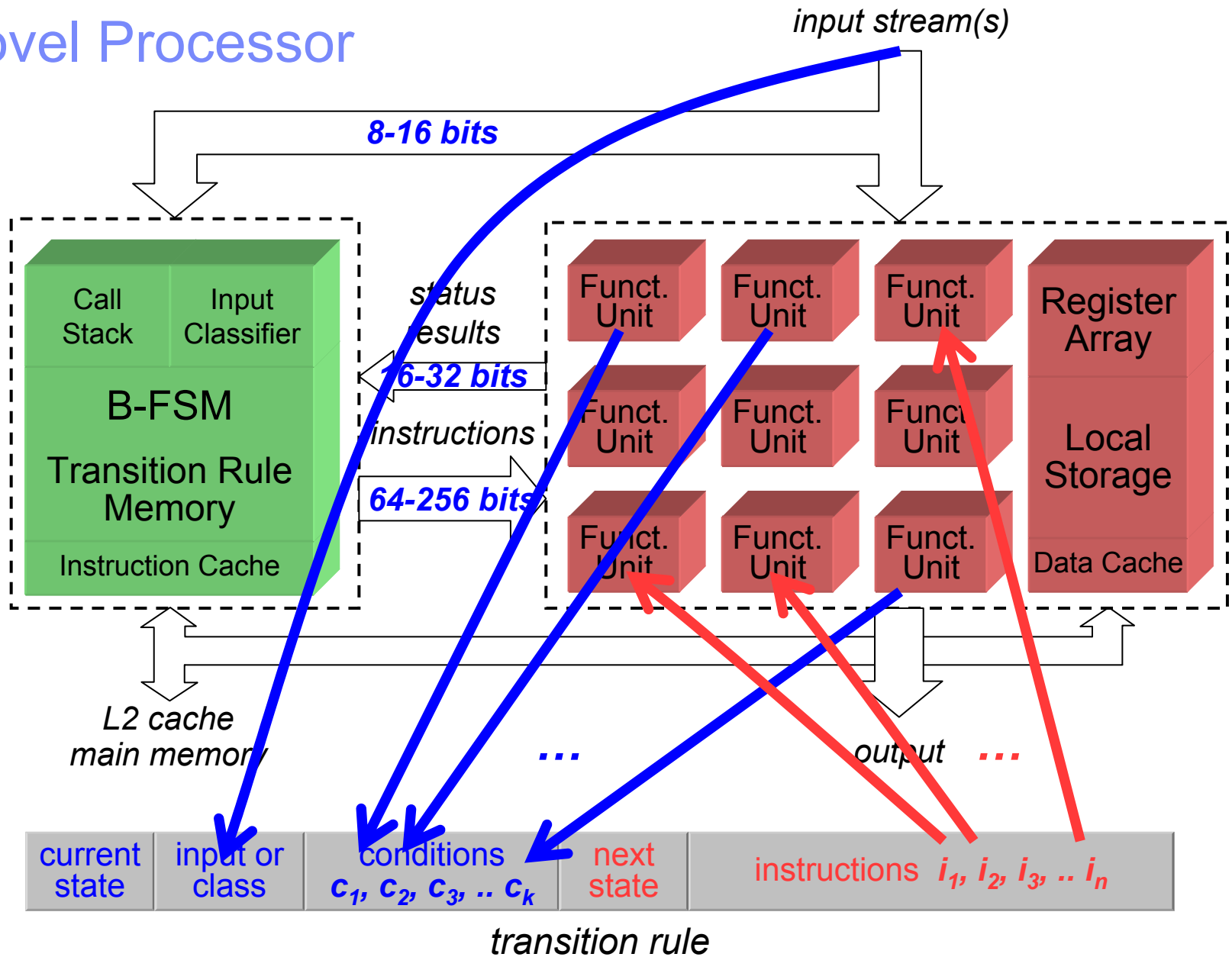
👉 Enabled by new programmable state machine technology: *B-FSM*

# Programmable State Machine

- State transition diagrams are described by *state transition rules* involving wildcards and priorities
- BaRT-based FSM (B-FSM):
  - rule selection based on BaRT search algorithm (hash function)
  - determines the highest-priority matching transition rule in a single cycle at clock frequencies into the GHz range



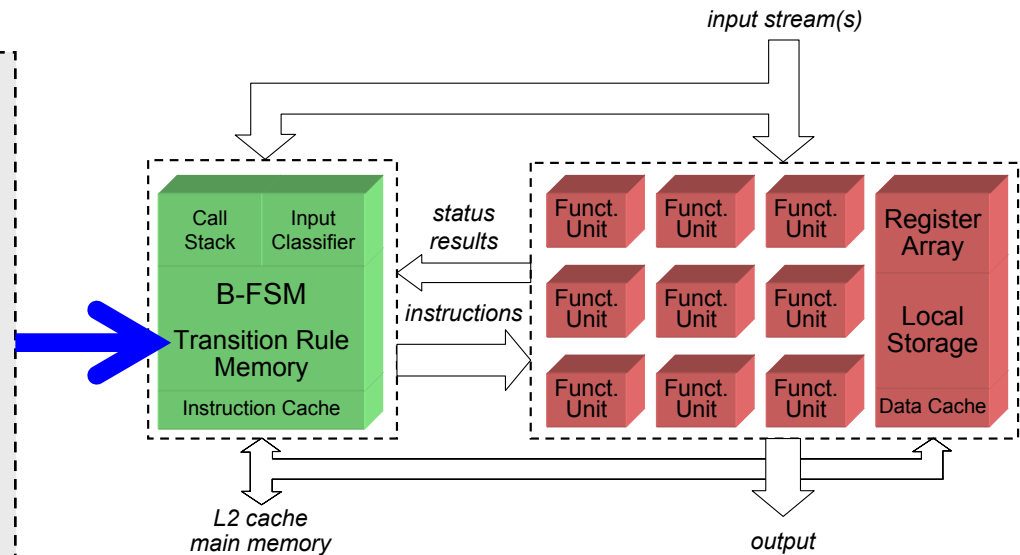
# Novel Processor



# Novel Processor

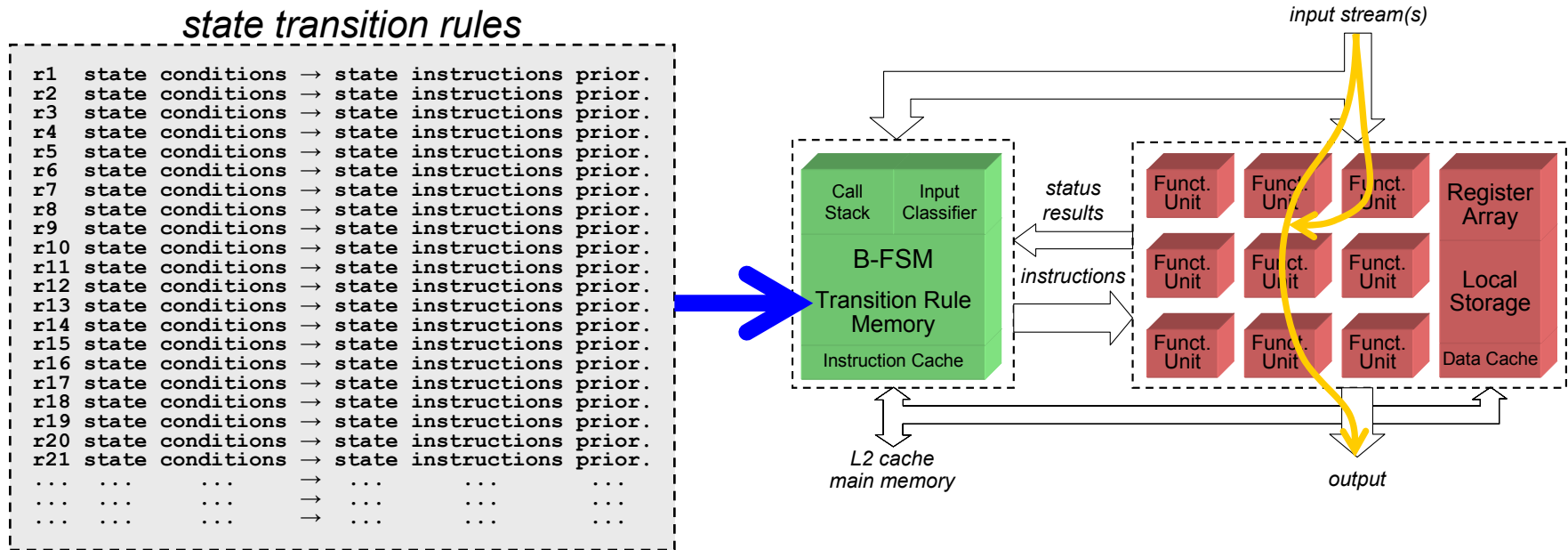
## state transition rules

r1	state	conditions	→	state	instructions	prior.
r2	state	conditions	→	state	instructions	prior.
r3	state	conditions	→	state	instructions	prior.
r4	state	conditions	→	state	instructions	prior.
r5	state	conditions	→	state	instructions	prior.
r6	state	conditions	→	state	instructions	prior.
r7	state	conditions	→	state	instructions	prior.
r8	state	conditions	→	state	instructions	prior.
r9	state	conditions	→	state	instructions	prior.
r10	state	conditions	→	state	instructions	prior.
r11	state	conditions	→	state	instructions	prior.
r12	state	conditions	→	state	instructions	prior.
r13	state	conditions	→	state	instructions	prior.
r14	state	conditions	→	state	instructions	prior.
r15	state	conditions	→	state	instructions	prior.
r16	state	conditions	→	state	instructions	prior.
r17	state	conditions	→	state	instructions	prior.
r18	state	conditions	→	state	instructions	prior.
r19	state	conditions	→	state	instructions	prior.
r20	state	conditions	→	state	instructions	prior.
r21	state	conditions	→	state	instructions	prior.
...	...	...	→	...	...	...
...	...	...	→	...	...	...
...	...	...	→	...	...	...



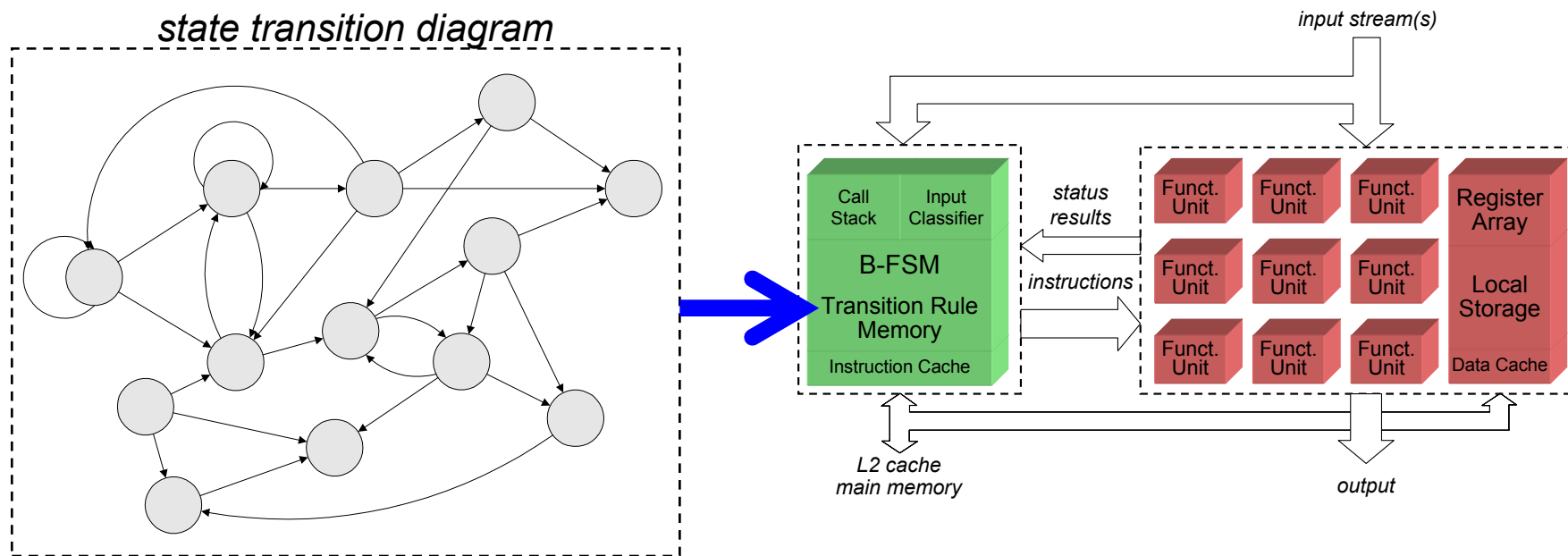
- Executable program (“binary”)
  - prioritized list of transition rules
  - in BaRT-compressed format
- In each clock cycle, hundreds of transition rules can be evaluated in parallel and the instructions associated with the highest-priority matching rule are dispatched to the functional units

# Novel Processor



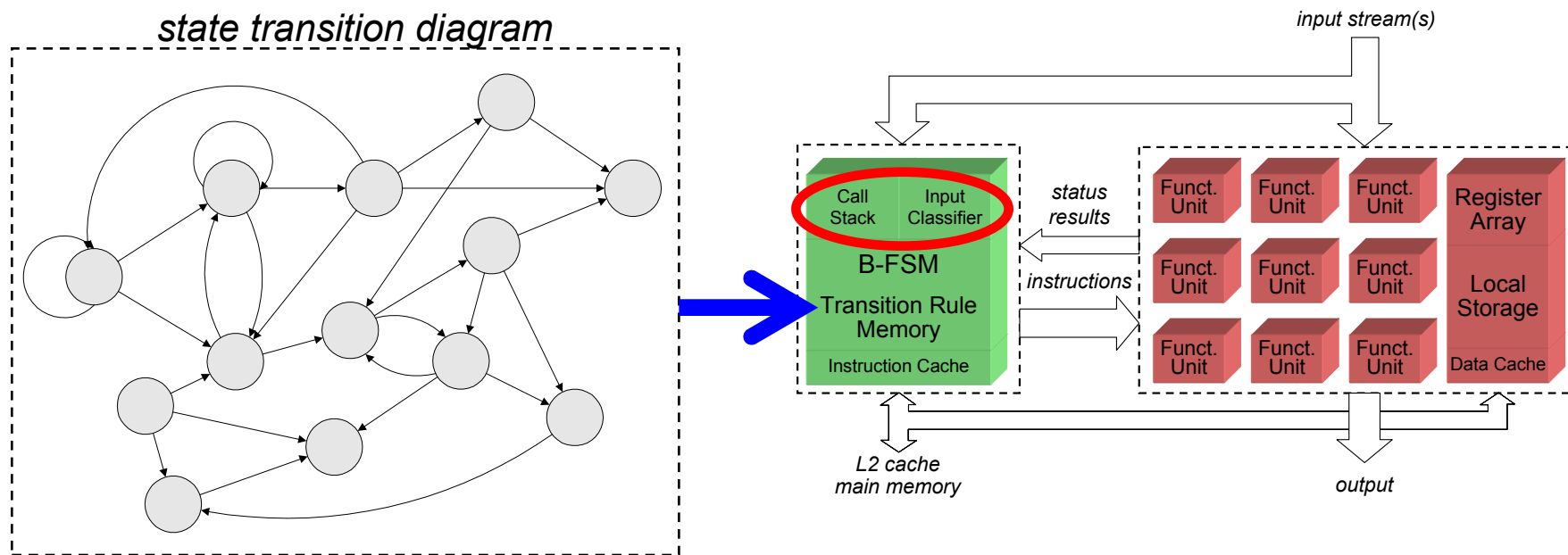
- Transition rules enable the programming of multiple functional units to operate in pipelined and/or parallel fashion on input data
  - rules can check the status of selected functional units and, in response, move data between these units, temporarily buffer data, put functional units on hold, synchronize functional units
- 👉 Allows pipelines comprised of multiple functional units to be set up dynamically


# Novel Processor



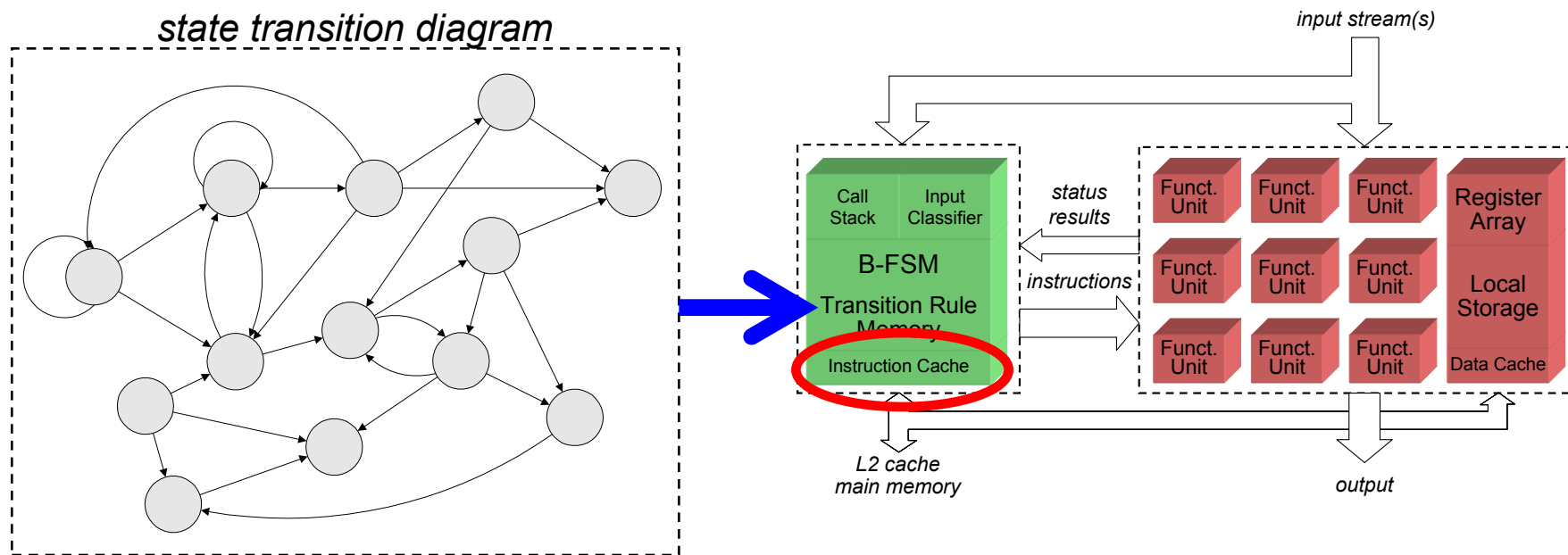
- Transition rules “visualized” as state transition diagram
  - state transition diagram defines all potential execution paths
  - actual path taken during program execution is determined by real-time evaluation of conditions associated with the state transitions
  - instructions “along” selected path are dispatched for execution
  
- State with one transition having no conditions: sequential execution

# Novel Processor



- Support for procedure calls (call stack provides return state)
- Programmable input classifier enables direct evaluation of complex conditions based on user-definable input classes
  - examples: digit, white space, legal name character (e.g., XML)
-  Improved storage efficiency by reducing the number of transition rules

# Instruction Cache and Prefetch

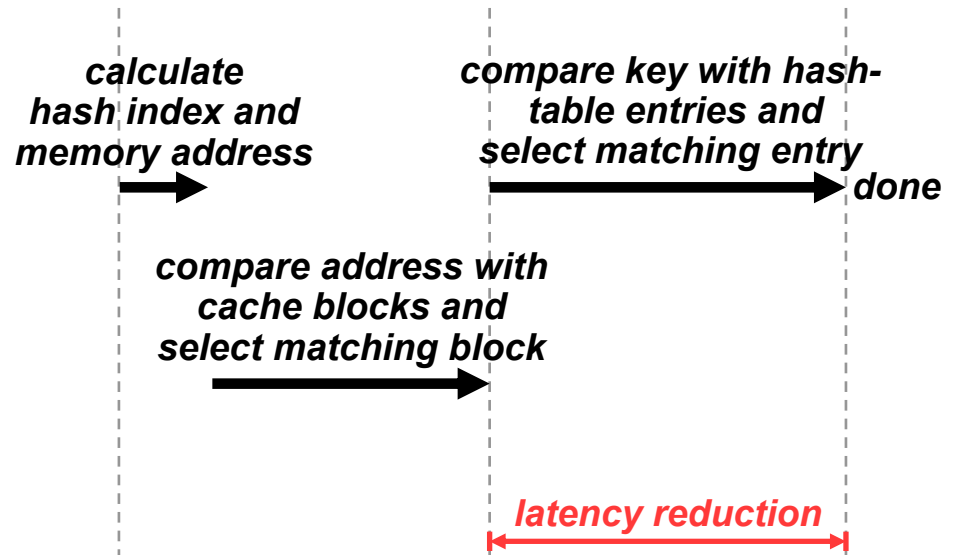
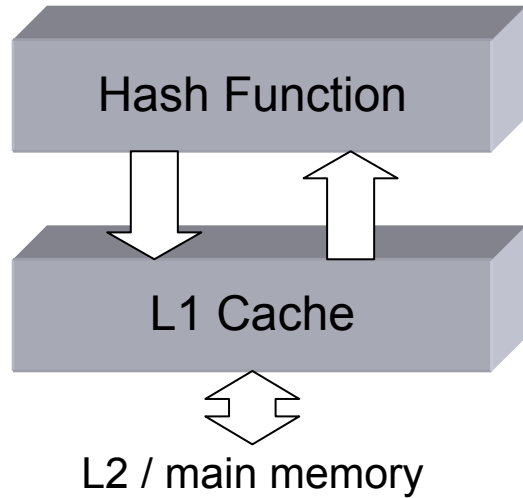


- Instruction cache and prefetching optimized for B-FSM operation
  - integration of B-FSM functionality into cache
  - hardware prefetching of instructions (transition rules)
  - selective mapping of transition rules on cache lines by compiler (enabled by B-FSM technology)

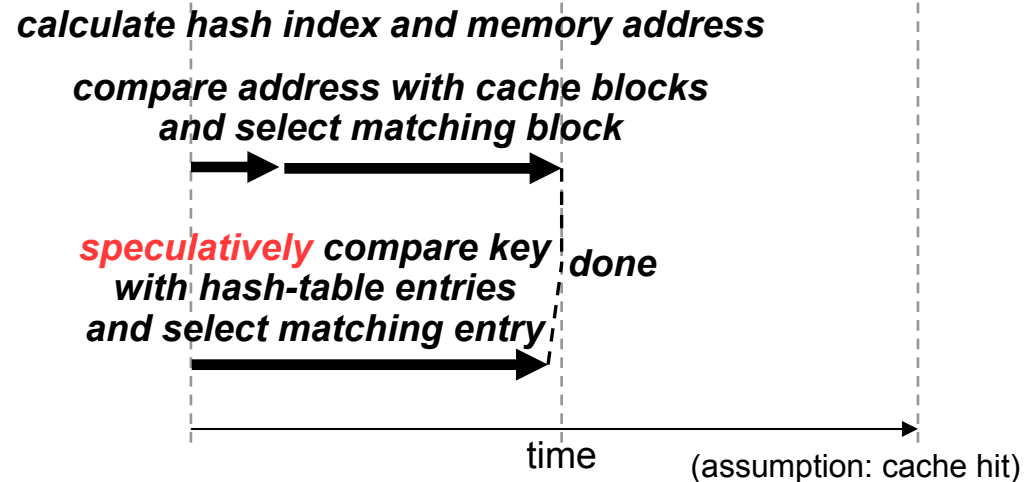
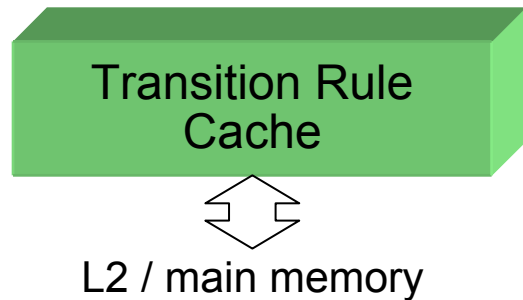


# Instruction Cache and Prefetch

- Hash function “on top of” cache



- Integrated hash function






## Experimental Results

- (Co)Processor concepts have been verified and validated by simulations and (FPGA) prototypes for a range of applications
- Synthesis experiments have shown the feasibility of running the B-FSM at a clock frequency of up to 2 GHz (65 nanometer CMOS)
  - max. rate of one transition per cycle out of the transition-rule cache
  - actual rate depends on cache performance, program, compiler, etc.
- Experience:
  - funct. units do not need to run for extended times in a stand-alone fashion, but receive new instructions from B-FSM within one or a few cycles in response to events related to input data or other units
  - this allows the functional units to be simplified and (re)used for multiple tasks, resulting in a faster and smaller implementation

# Summary

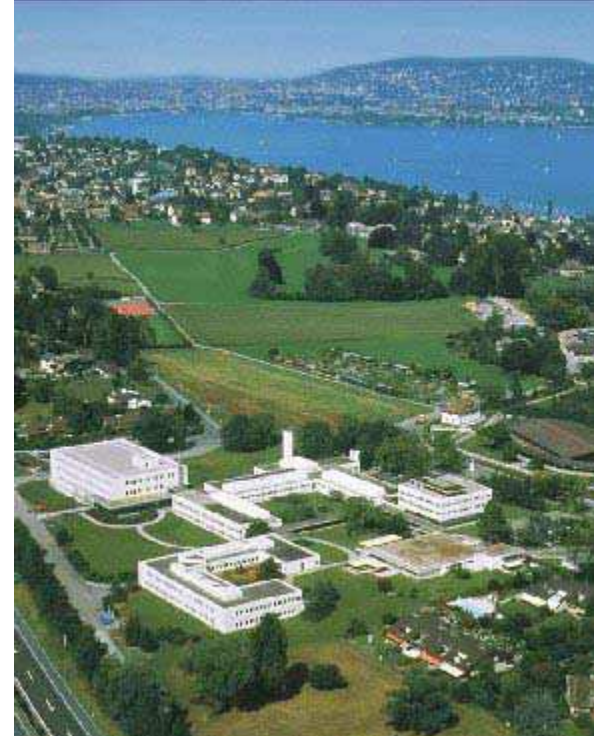
## Novel Processor Architecture for Stream Processing

- A single “general-purpose” (co)processor targeting multiple applications (e.g., XML processing, pattern-matching, compression, searching)
- Key features:
  - VLIW type of processor with instruction fetch and issue unit based on a high-performance programmable state machine
  - efficient programming of large variety of functional units to achieve high data-processing rates ( $> 10$  Gb/s)
  - “BISC”: B-FSM-based Instruction Scheduled Computer
- Although a considerable part of the project still is in a research phase, several key concepts and enabling technologies have already been realized and partially made available to customers
  - pattern-matching engine presented at Hot Chips 17, 2005
-  Making a coprocessor more general-purpose *programmable* does not always require large compromises on performance and hardware costs

For more information, contact:

Jan van Lunteren  
IBM Research GmbH  
Zurich Research Laboratory  
Säumerstrasse 4  
CH-8803 Rüschlikon  
Switzerland

E-mail: [jvl@zurich.ibm.com](mailto:jvl@zurich.ibm.com)  
Phone: +41 44 724 8111

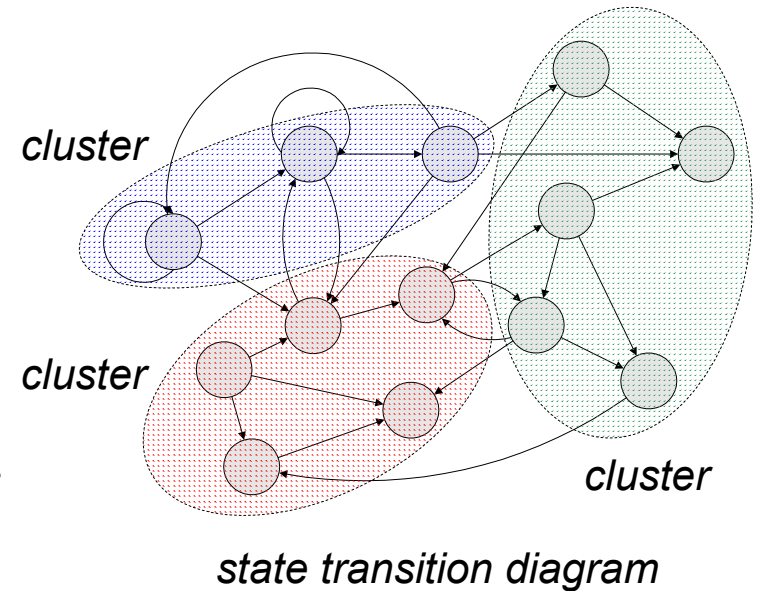


# Backup

# Programmable State Machine

## BaRT-based FSM (B-FSM) technology

- Rule selection by *BaRT* routing-table search algorithm (hash function)
- State transition diagram is divided into state clusters, which are independently mapped on BaRT-compressed transition-rule tables
- Optimized clustering and state encoding within each cluster
- 👉 Near-optimal filling of hash tables, enabling high storage efficiency
- 👉 Simplified hash function, enabling fast implementation of B-FSM logic
- One transition/clock cycle at 1-2 GHz
- Scalable to hundreds of thousands of states and transitions
- Supports wide input and output vectors
- Fast dynamic updates



# Publications

## B-FSM-related work

- J. van Lunteren, “High-performance pattern matching for intrusion detection,” Proc. IEEE Infocom, Barcelona, Spain, April 2006.
- J. van Lunteren et al., “High-performance pattern-matching engine for intrusion detection,” Hot Chips 17, Stanford University, Palo Alto, CA, August 2005.
- J. van Lunteren et al., “XML accelerator engine,” First Int. Workshop on High Performance XML Processing, in conjunction with WWW2004, May 2004.

## BaRT-related work

- J. van Lunteren et al., “Fast and scalable packet classification,” IEEE Journal of Selected Areas in Communications, vol. 21, no. 4, pp. 560-571, May 2003.
- J. van Lunteren, “Searching very large routing tables in wide embedded memory,” Proc. IEEE Globecom, vol. 3, pp. 1615-1619, November 2001.

## Other

- Scientific American, “Recognition engines,” January 2006.
- EE Times, “CPUs take parallel turn at Hot Chips,” August 2005.