consorzio nazionale
interuniversitario
per le telecomunicazioni

Università di Roma
Tor Vergata

Open Packet Processor:
Platform-agnostic Behavioral Forwarding and Stateful Flow Processing at wire speed

Valerio Bruschi, CNIT/University of Rome "Tor Vergata"

Joint work with: G. Bianchi, M. Bonola, S. Pontarelli, A. Capone, C. Cascone, D. Sanvito.

EU support:

SUPERFLUIDITY

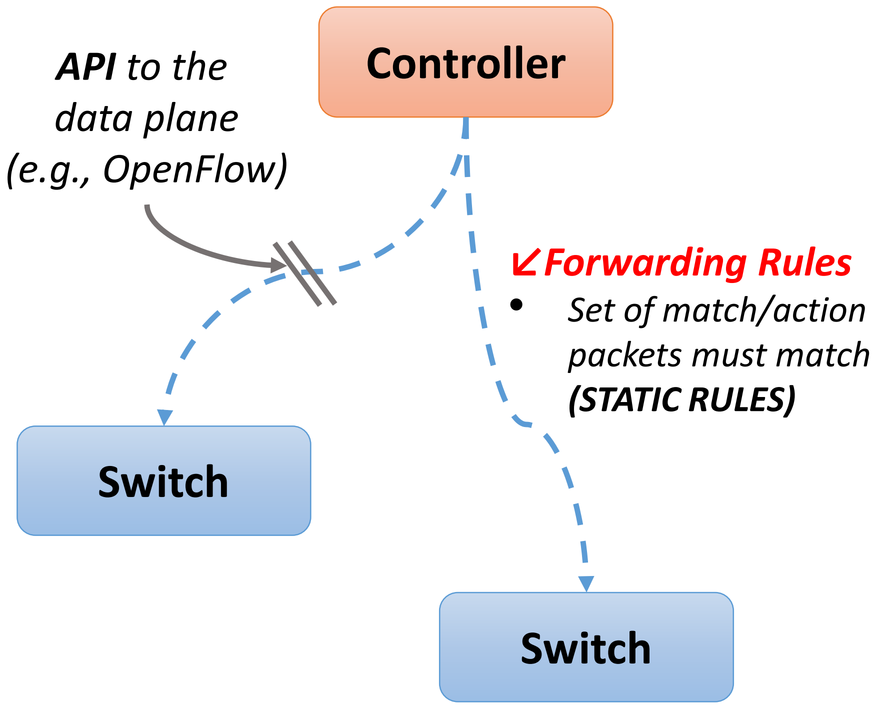
Beba
BEhavioural BAsed forwarding

Approach proposed

Stateful data plane

Background

OpenFlow/SDN (2009)

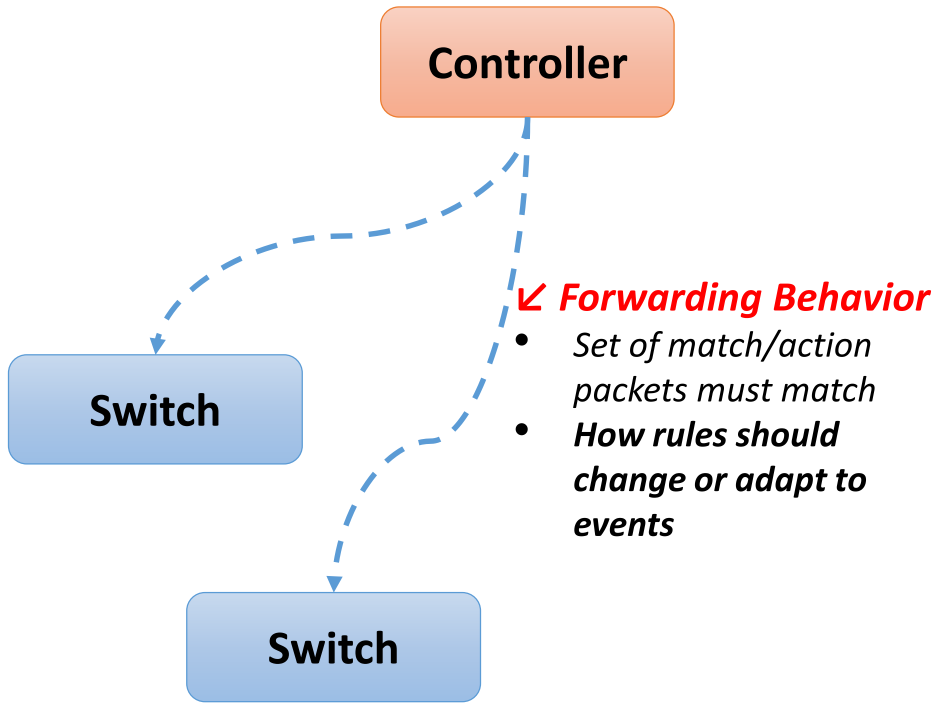


↙Forwarding Rules

- *Set of match/action packets must match* ***(STATIC RULES)***

Dumb switch: *need to ask controller if something changes*

OpenState/SDN (2014)



↙ Forwarding Behavior

- *Set of match/action packets must match*
- ***How rules should change or adapt to events***

Smart switch: *can dynamically update flow tables*

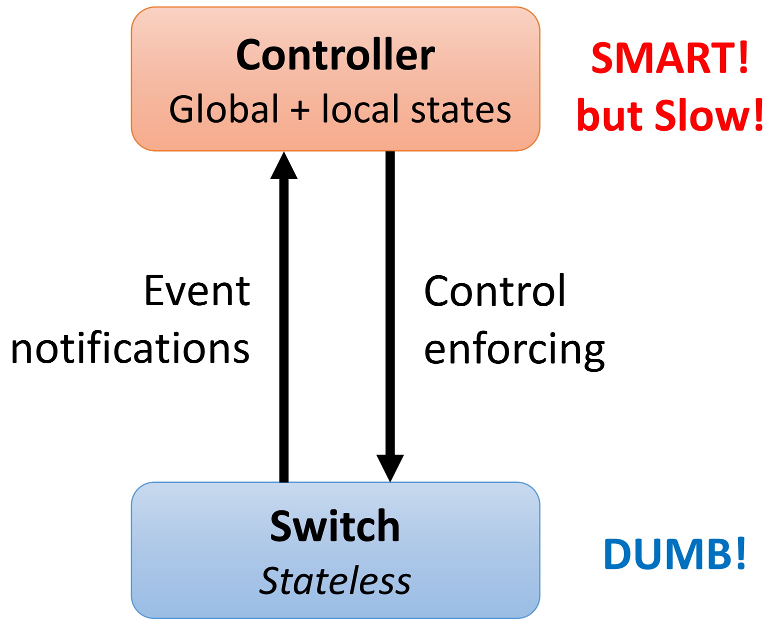
Motivations

- OpenFlow's platform-agnostic programmatic interface permits to dynamically update match/action forwarding rules **only via the explicit involvement of an external controller**
- OpenFlow **does not permit to deploy forwarding behaviors directly in the switches**, i.e. describe how rules should evolve in time as a consequence of packet-level events
- Such static nature of the OpenFlow forwarding abstraction raises serious concerns regarding:
  - **Scalability**
  - **Latency**
  - **Security/reliability**

Details in G. Bianchi, M. Bonola, A. Capone, C. Cascone,
*"**OpenState: programming platform-independent stateful OpenFlow applications inside the switch**",*
ACM SIGCOMM Computer Communication Review, vol. 44, no. 2, April 2014.

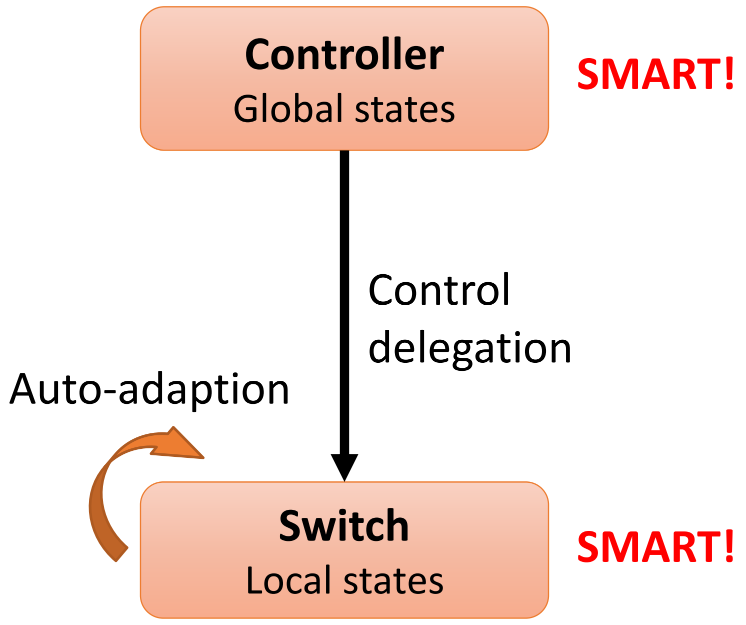
Stateless vs. Stateful in SDN

Stateless data plane model (e.g. OpenFlow)



Signalling & latency: *O(100* ***ms****)*
100ms = 30M packets lost @ 100 gbps

Stateful data plane model (e.g. OpenState)



Signalling & latency:
update forwarding rules in 1 packet time –
3 ***ns*** *@ 40B x 100 Gbps*

Beyond OpenState

Mealy Machine:
nice but insufficient!

- ❑ State alone is **insufficient**
- ❑ OpenFlow (forwarding) actions are **insufficient**
- ❑ **No** flow processing

Flow Processing

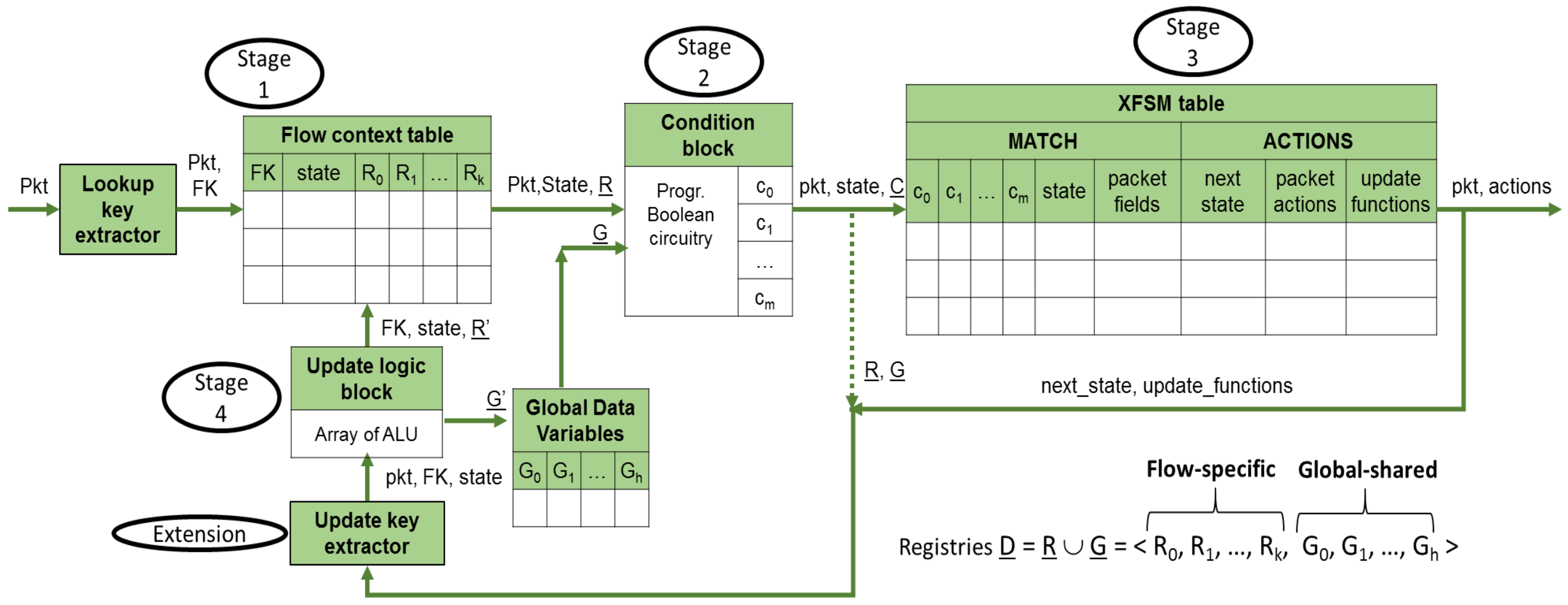
- ❑ Flow processing requires **memory, registries, counters**, etc
- ❑ Flow processing requires **operations** (compare, add, shift, etc)
- ❑ **Processing = CPU**! *cannot afford any ordinary CPUs at ns time scales wire speed!*

Open Packet Processor

- From mealy finite state machines(FSM) to **Extended finite state machines(XFSM)**
- An EFSM is a finite state machine in which:
  1) state transitions **depends** also on a set of triggering **conditions** depending on data variables;
  2) state transitions **trigger the update** of data variables
- It also allows **cross-flow** state modification.
- <u>Hard parts</u>: use platform agnostic abstractions and make it run at wire speed – ***no CPUs***!

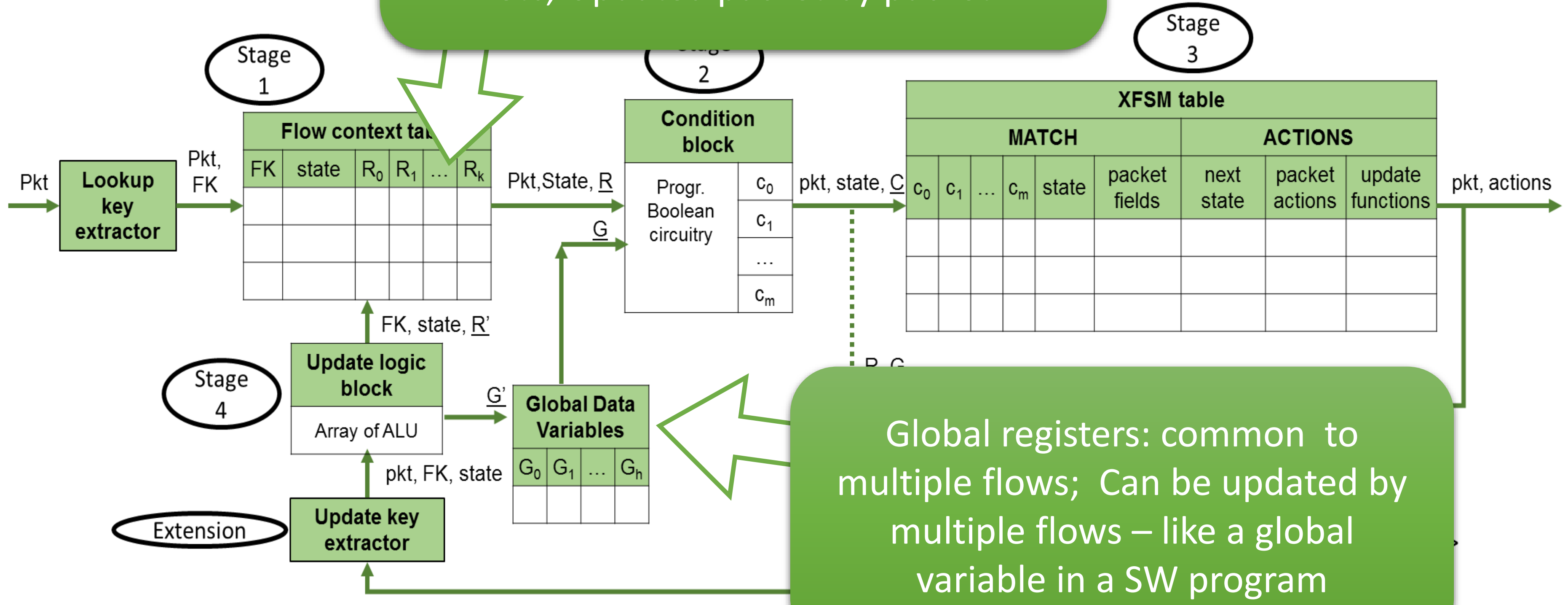
Details in G. Bianchi, M. Bonola, S. Pontarelli, D. Sanvito, A. Capone,
*“**Open Packet Processor: a programmable architecture for wire speed platform-independent stateful in-network processing**”, arXiv preprint arXiv:1605.01977, 2016.*

Open Packet Processor: workflow



$\text{Registries } \underline{D} = \underline{R} \cup \underline{G} = < \underbrace{R_0, R_1, \ldots, R_k}_{\text{Flow-specific}}, \underbrace{G_0, G_1, \ldots, G_h}_{\text{Global-shared}} >$

Open Packet Processor: workflow

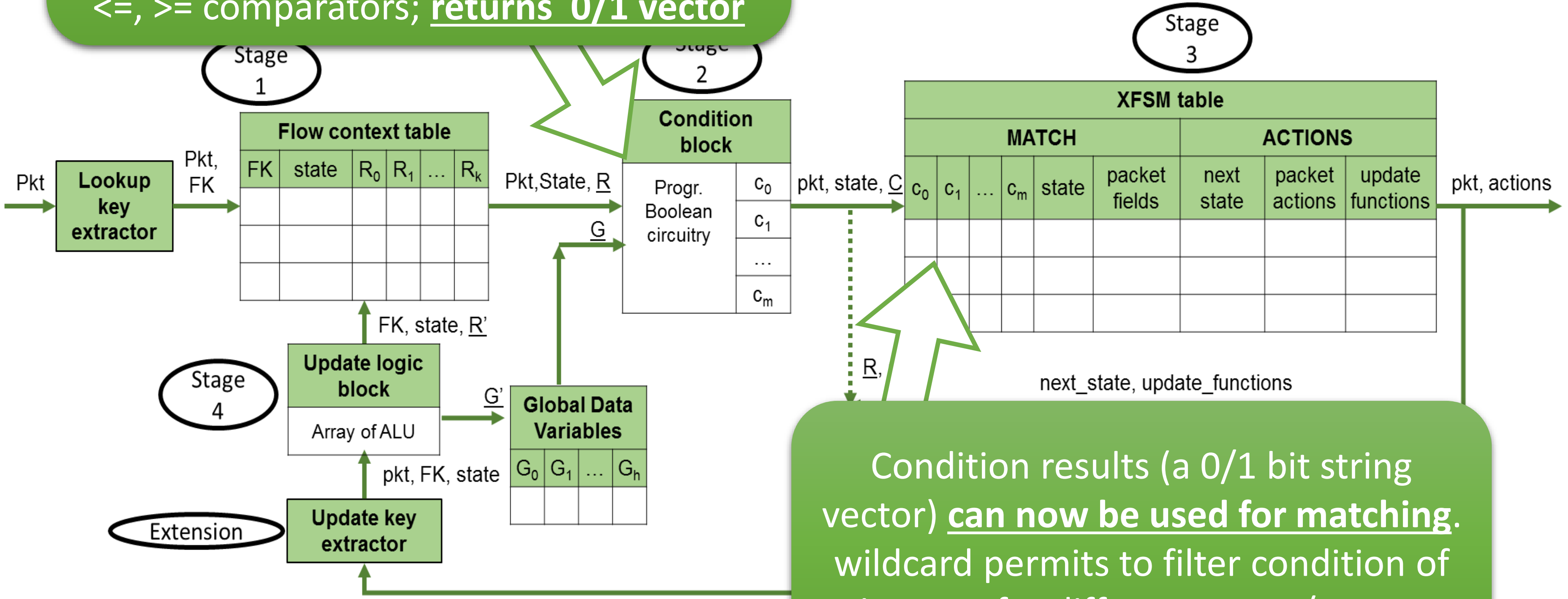


Per flow registers: programmer-defined (like variables in a program) e.g.: custom statistics, traffic features, etc; Updated packet by packet

Global registers: common to multiple flows; Can be updated by multiple flows – like a global variable in a SW program

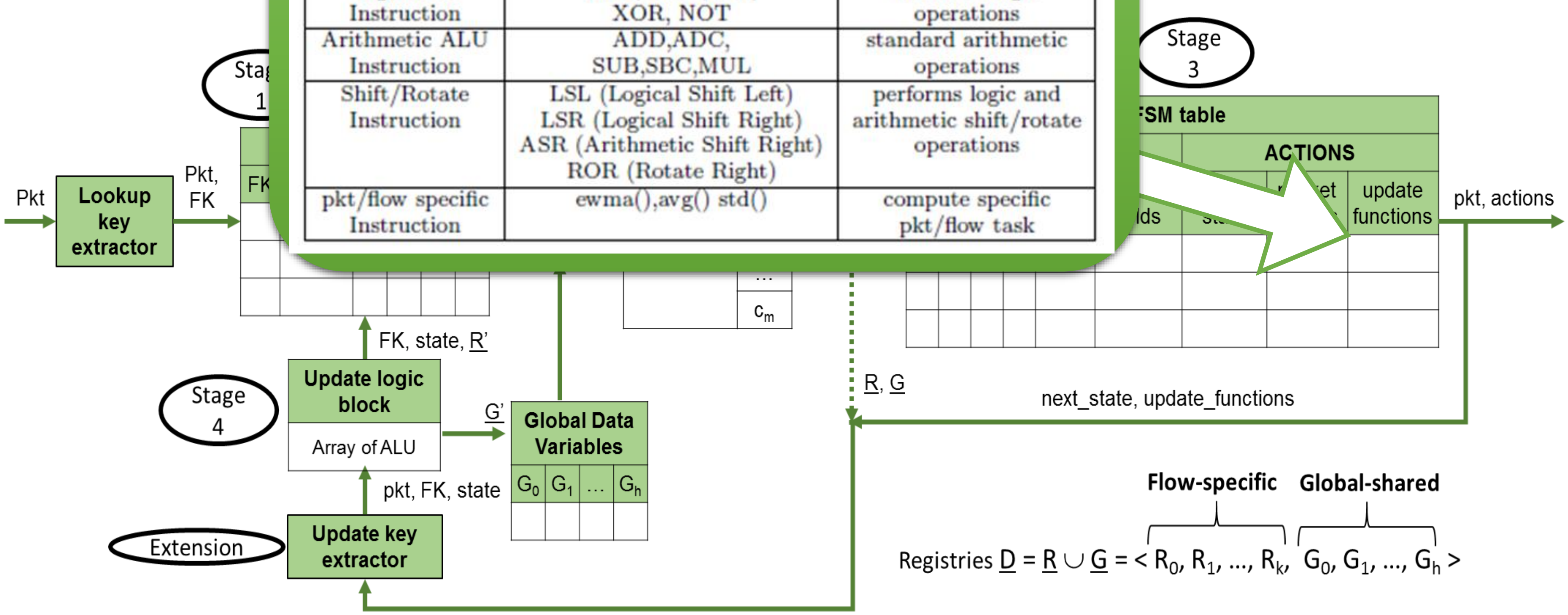
Open Packet Processor: workflow

User-programmed set of comparators. Compares pairs of quantities among registries, global variables, and packet header fields, using user-selected >, <, =, <=, >= comparators; **returns 0/1 vector**

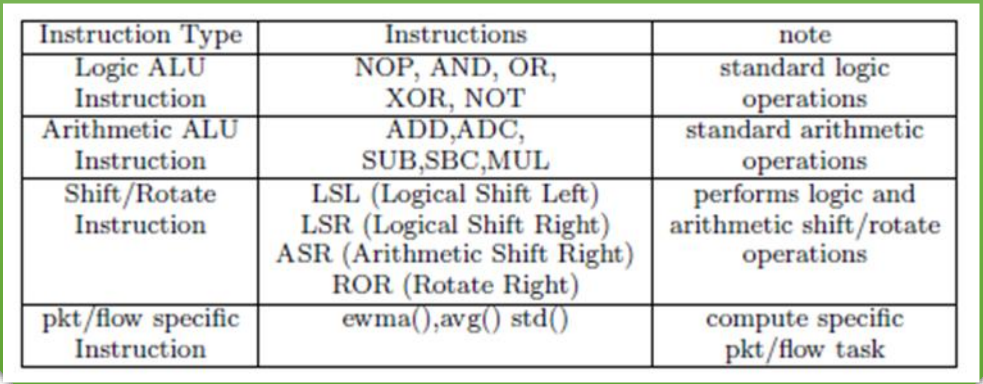


Condition results (a 0/1 bit string vector) **can now be used for matching**. wildcard permits to filter condition of interest for different states/events

Open Packet Processor: workflow

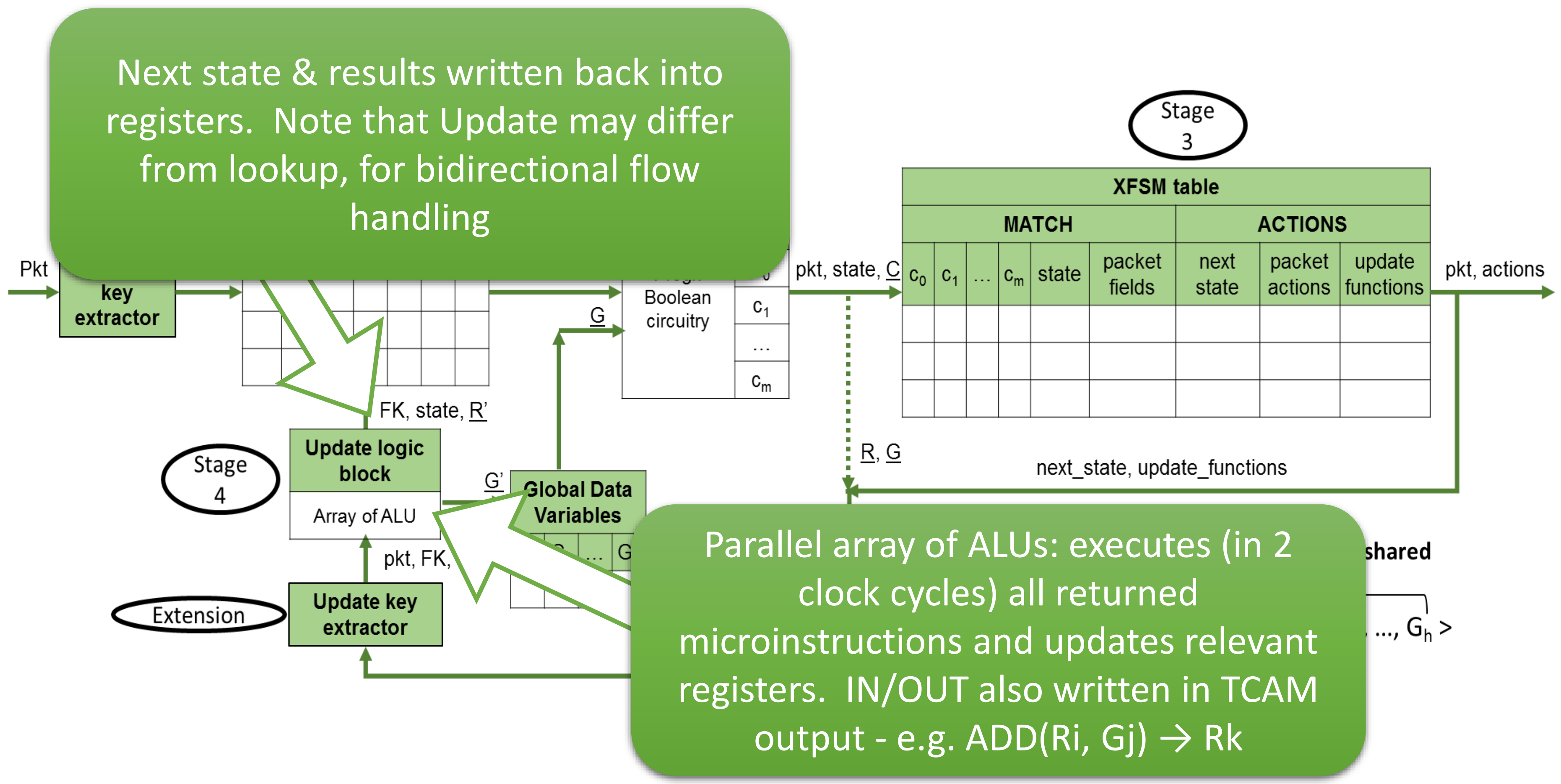


Returns microinstructions (of a domain-specific custom ALU instruction set) to be applied



Instruction Type	Instructions	note
Logic ALU Instruction	NOP, AND, OR, XOR, NOT	standard logic operations
Arithmetic ALU Instruction	ADD,ADC, SUB,SBC,MUL	standard arithmetic operations
Shift/Rotate Instruction	LSL (Logical Shift Left) LSR (Logical Shift Right) ASR (Arithmetic Shift Right) ROR (Rotate Right)	performs logic and arithmetic shift/rotate operations
pkt/flow specific Instruction	ewma(),avg() std()	compute specific pkt/flow task

Open Packet Processor: workflow

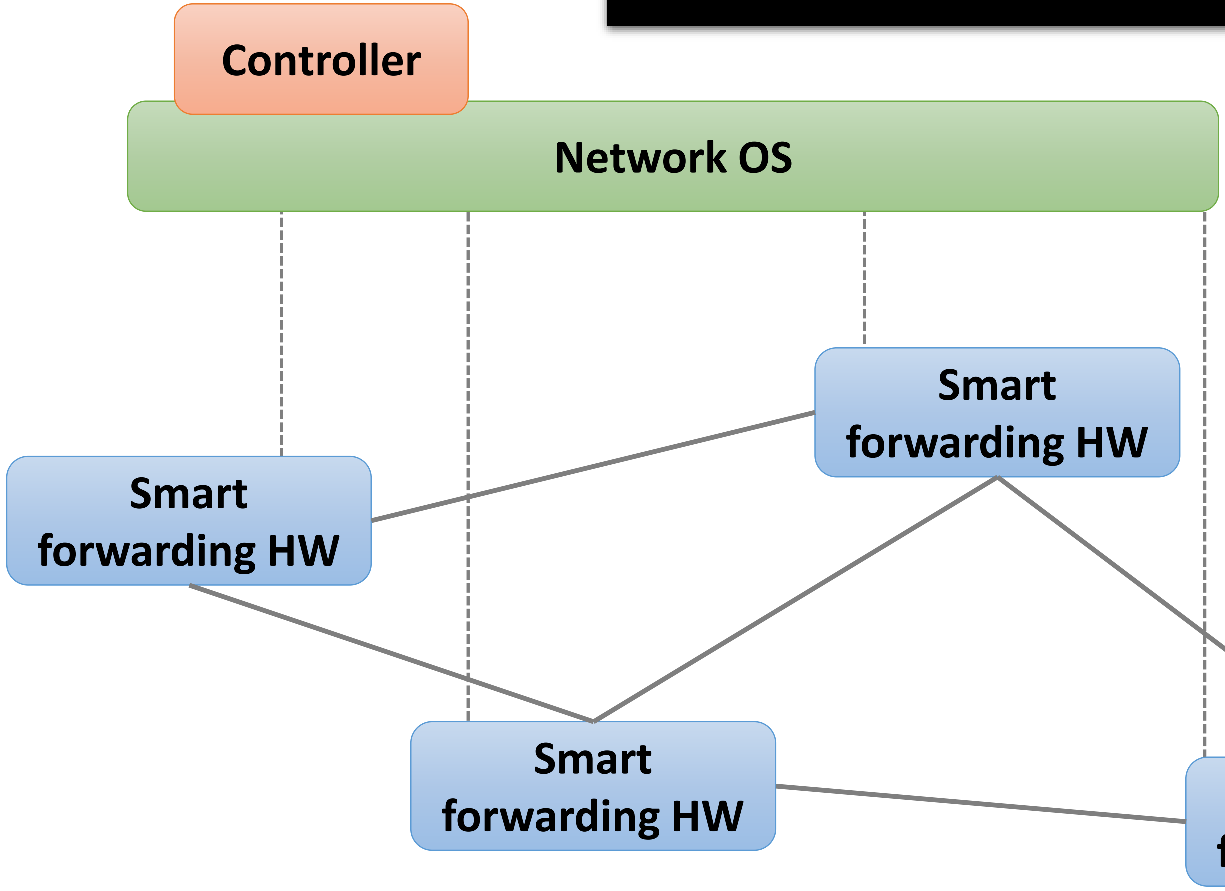


Next state & results written back into registers. Note that Update may differ from lookup, for bidirectional flow handling

Parallel array of ALUs: executes (in 2 clock cycles) all returned microinstructions and updates relevant registers. IN/OUT also written in TCAM output - e.g. ADD(Ri, Gj) → Rk

Overall vision: still "SDN"

Controller still in charge to 'program' the network
But **can 'push' <u>time-critical / localized</u>**
stateful control tasks down in the switches



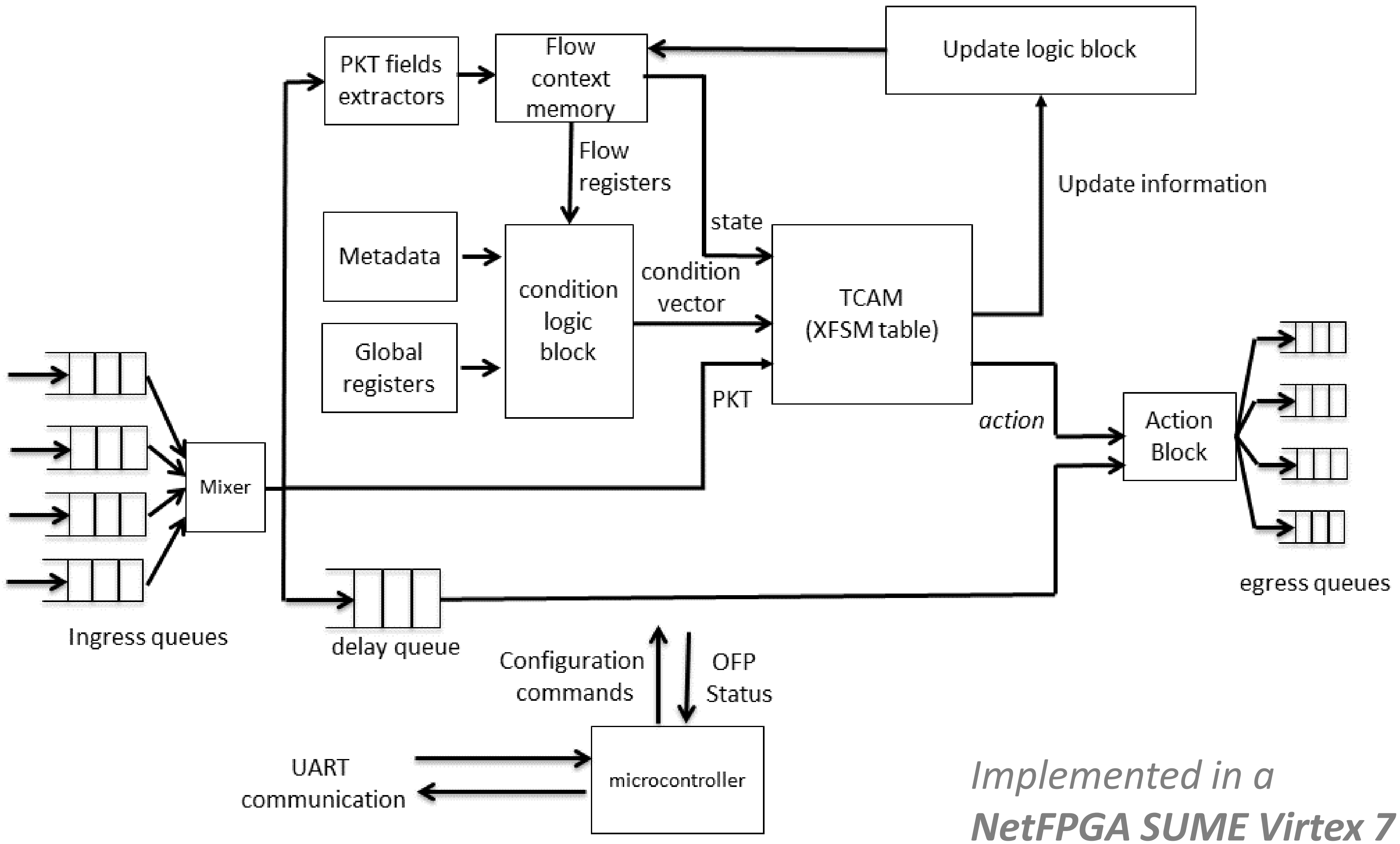
Several applications

- *Traffic policing*
- *Classifiers*
- *DoS mitigation*
- *Fault tolerance and fast failover*
- *Data driven routing*
- *Security/monitoring*
- *Stateful firewall*

NetFPGA prototype

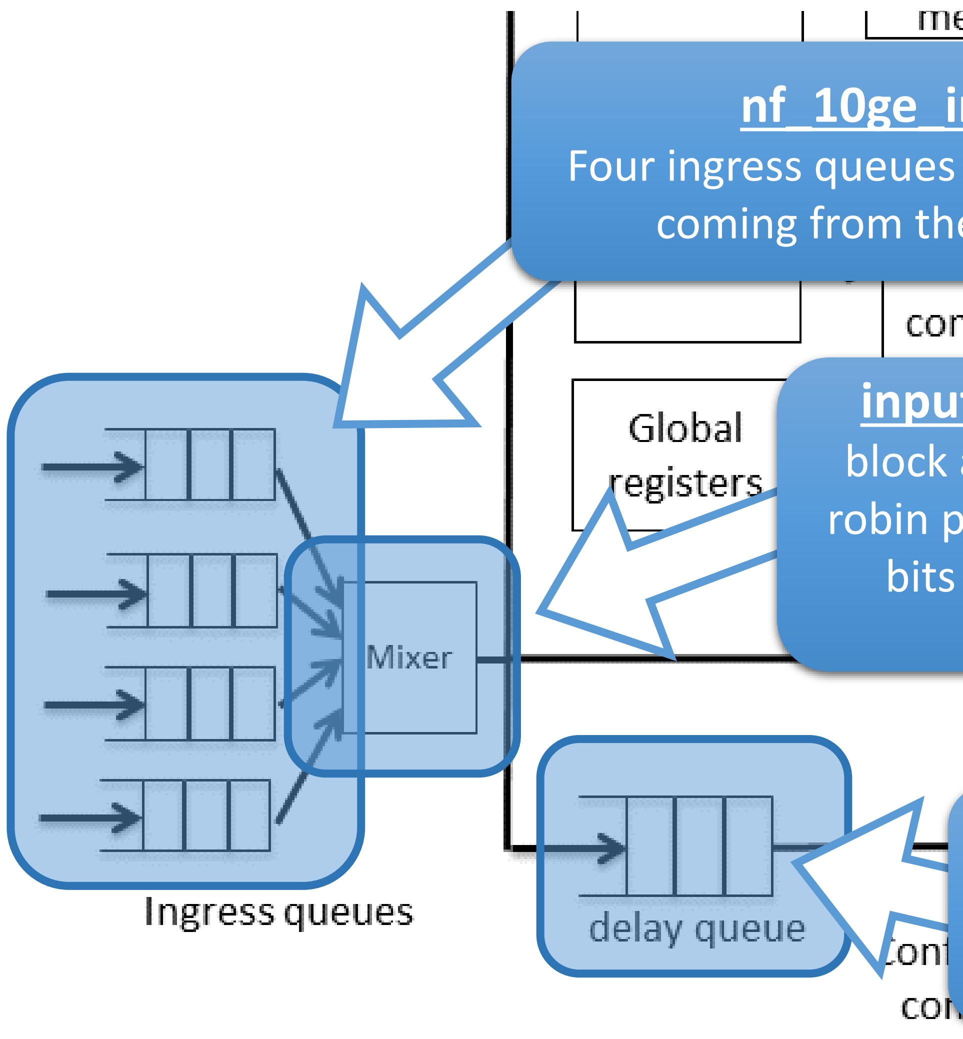
HW proof of concept implementation

Prototype architecture



Implemented in a ***NetFPGA SUME Virtex 7***

Prototype architecture

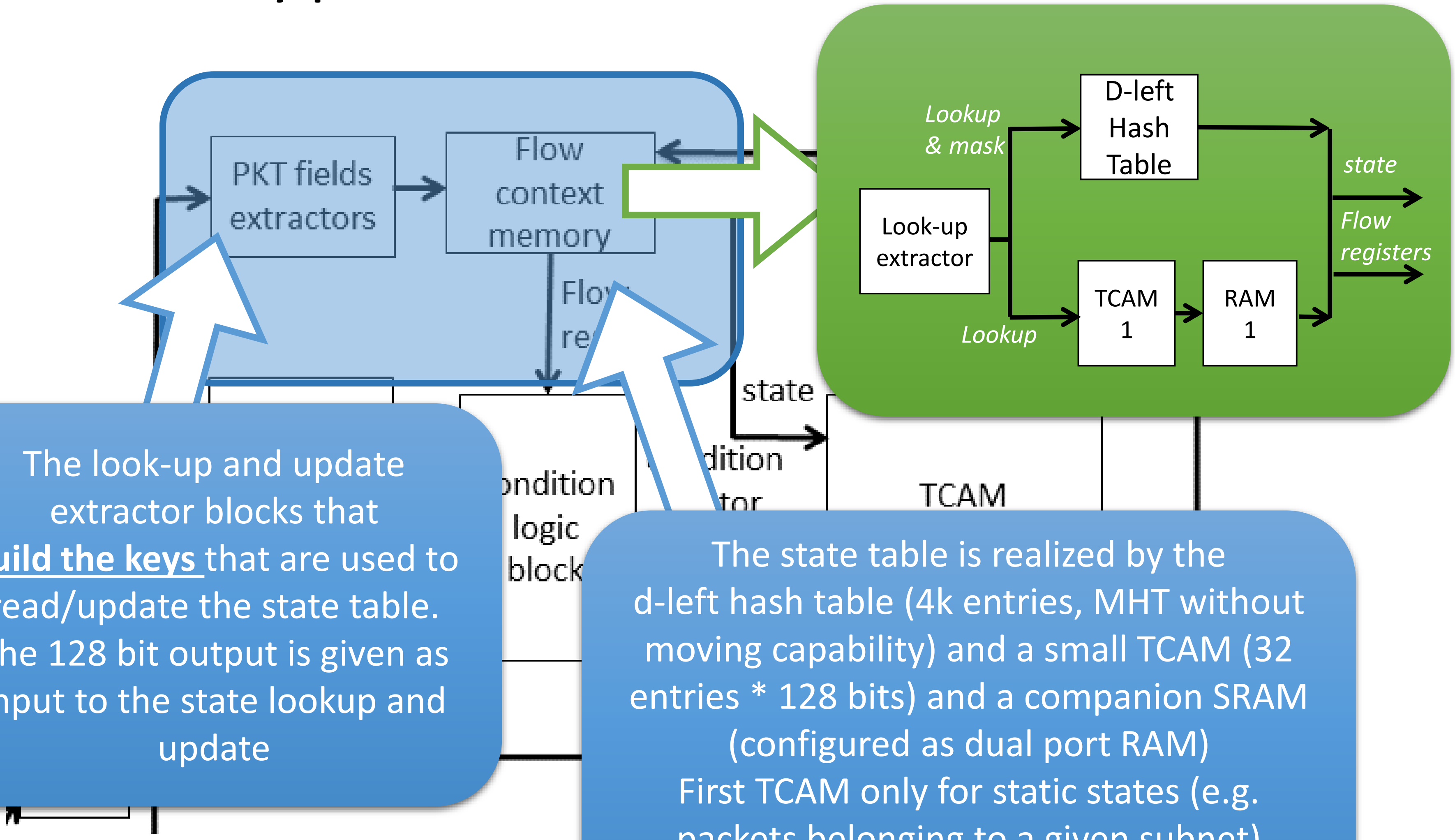


nf_10ge_interface:
Four ingress queues collect the packets coming from the ingress ports

input_arbiter: A 4-input 1-output mixer block aggregates the packets using a round robin policy. The output of the mixer is a 320 bits data bus able to provide an overall throughput of 50 Gbps

A delay queue stores the packet during the time need by the Open Packet Processor tables to operate

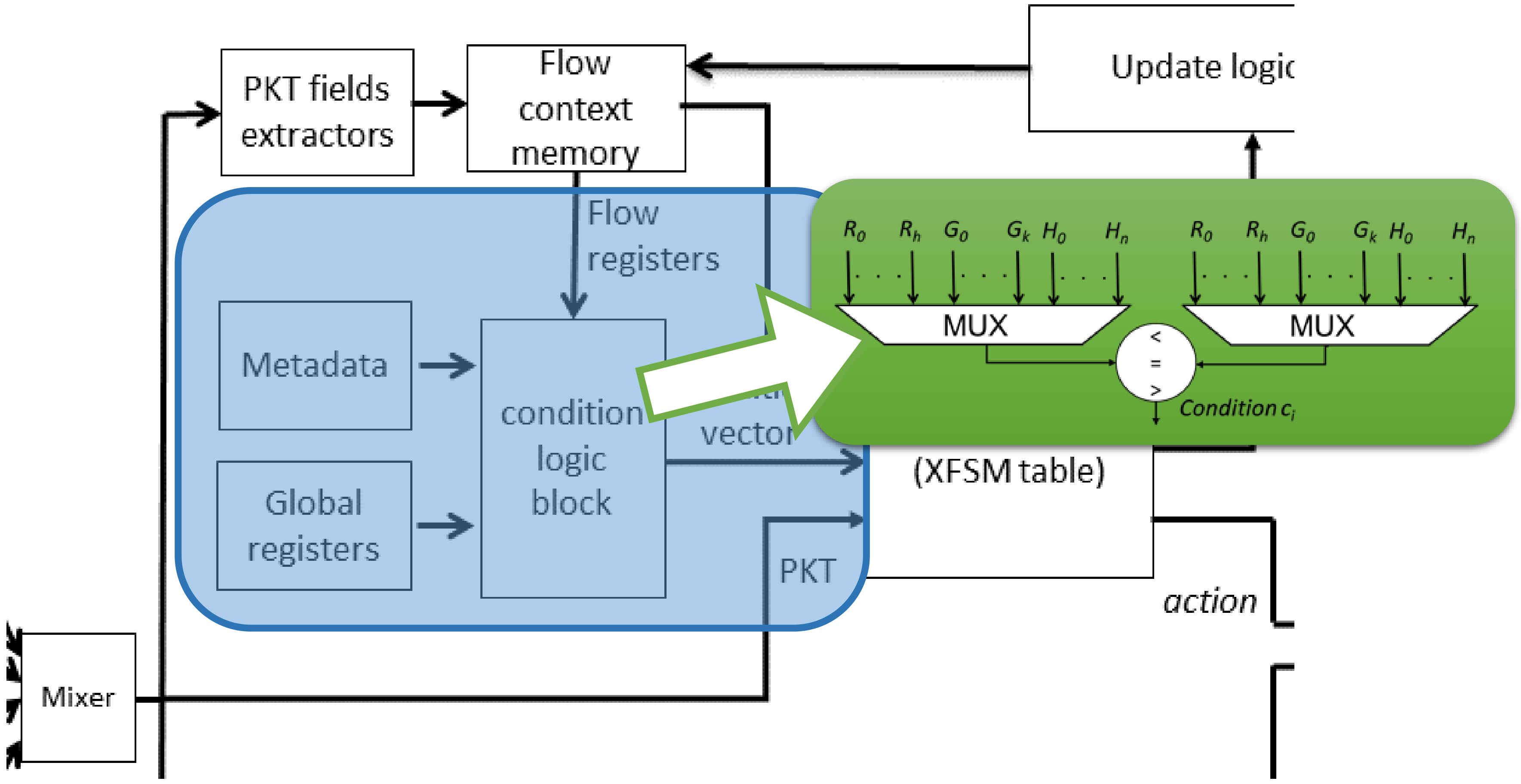
Prototype architecture



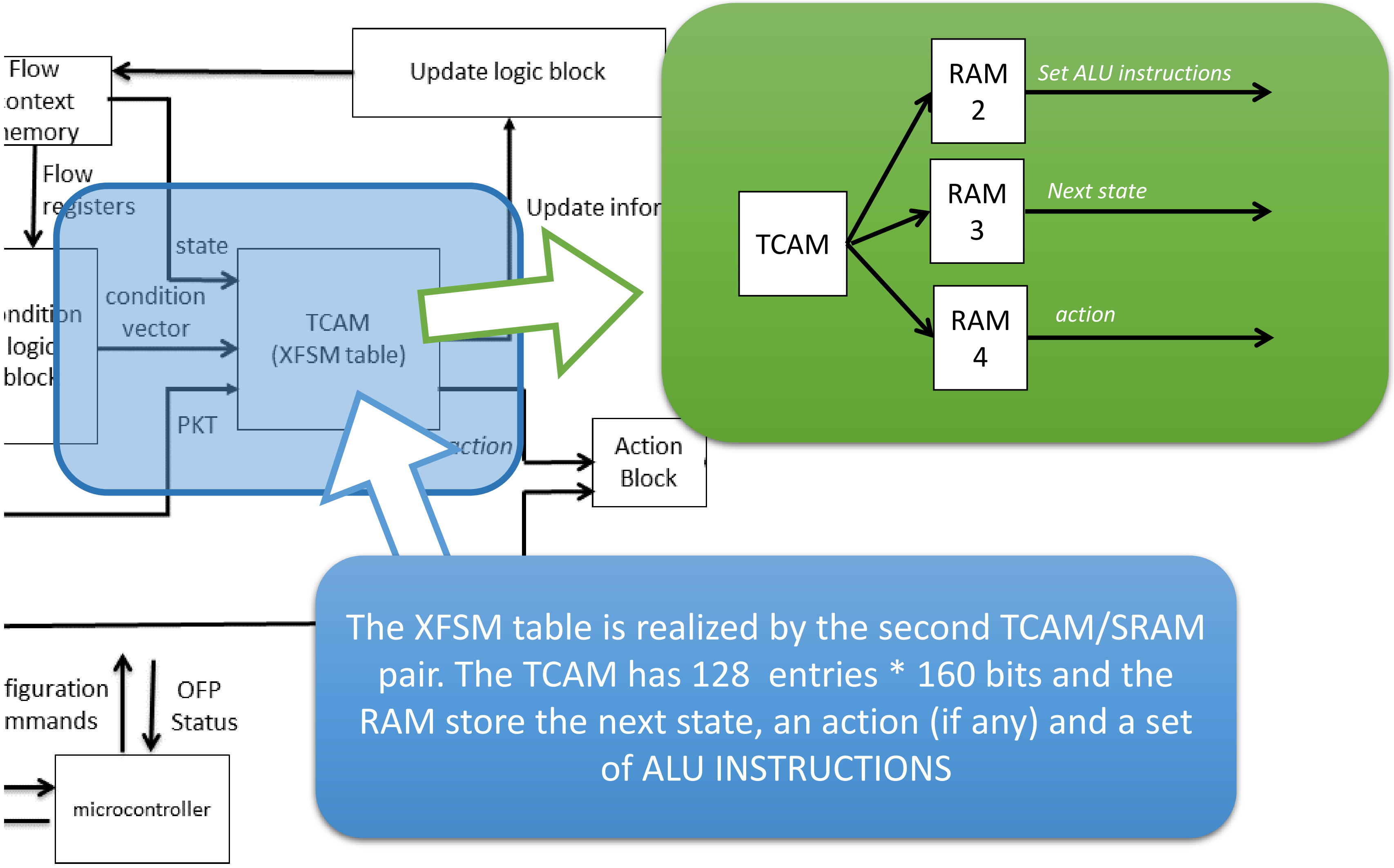
The look-up and update extractor blocks that **build the keys** that are used to read/update the state table. The 128 bit output is given as input to the state lookup and update

The state table is realized by the d-left hash table (4k entries, MHT without moving capability) and a small TCAM (32 entries * 128 bits) and a companion SRAM (configured as dual port RAM)
First TCAM only for static states (e.g. packets belonging to a given subnet)

Prototype architecture

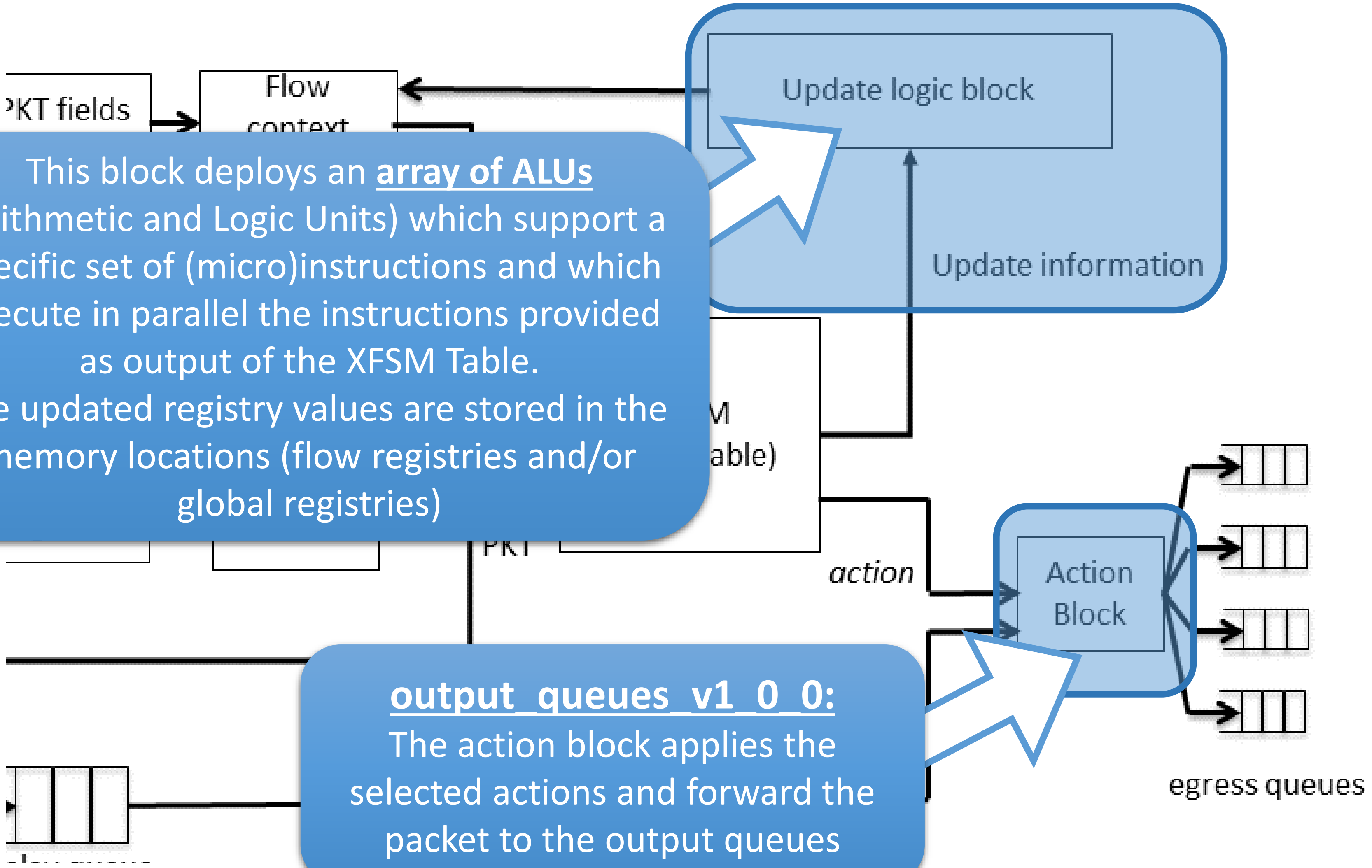


Prototype architecture



The XFSM table is realized by the second TCAM/SRAM pair. The TCAM has 128 entries * 160 bits and the RAM store the next state, an action (if any) and a set of ALU INSTRUCTIONS

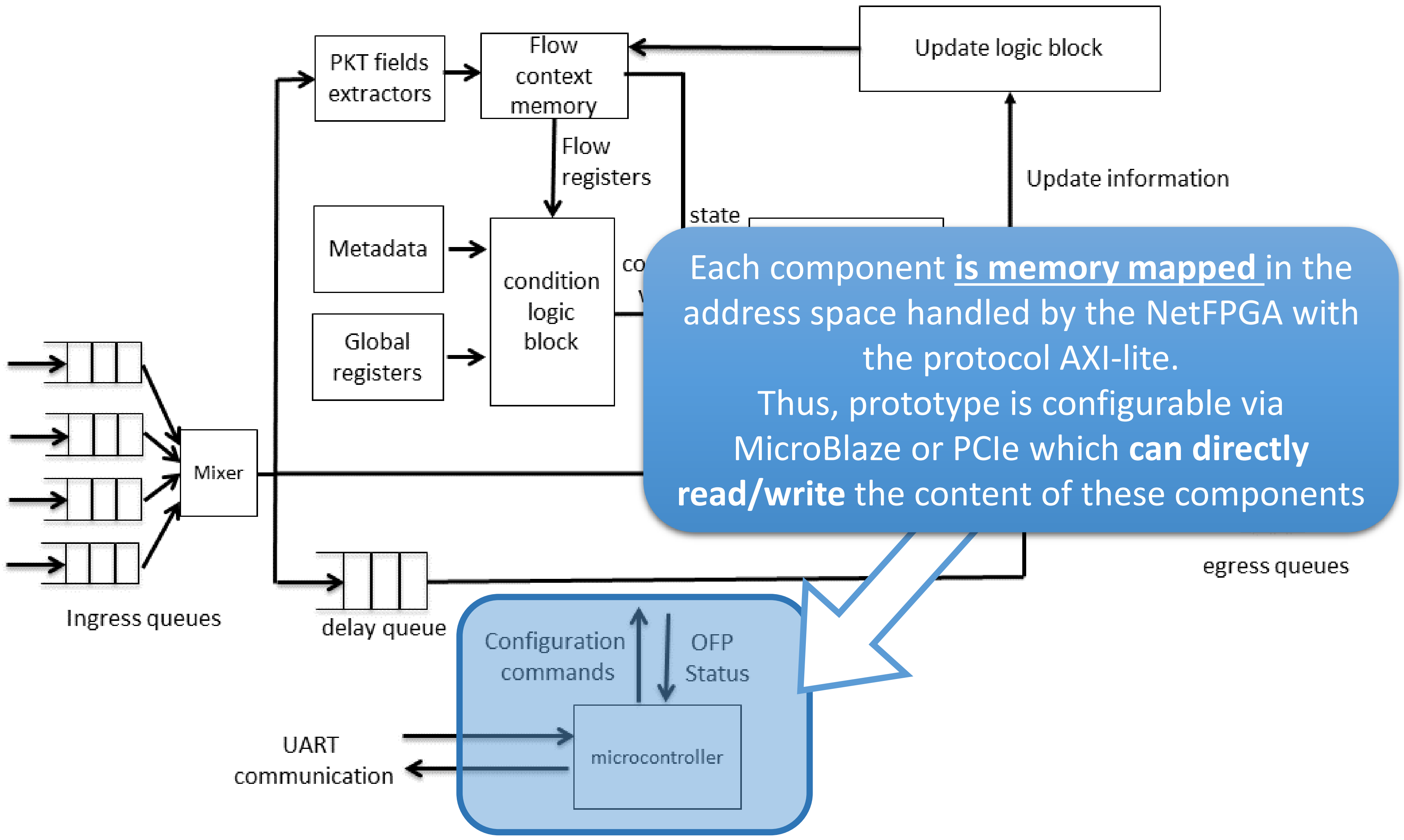
Prototype architecture



This block deploys an **array of ALUs** (Arithmetic and Logic Units) which support a specific set of (micro)instructions and which execute in parallel the instructions provided as output of the XFSM Table.
The updated registry values are stored in the memory locations (flow registries and/or global registries)

output_queues_v1_0_0:
The action block applies the selected actions and forward the packet to the output queues

Prototype architecture



Each component **is memory mapped** in the address space handled by the NetFPGA with the protocol AXI-lite.
Thus, prototype is configurable via MicroBlaze or PCIe which **can directly read/write** the content of these components

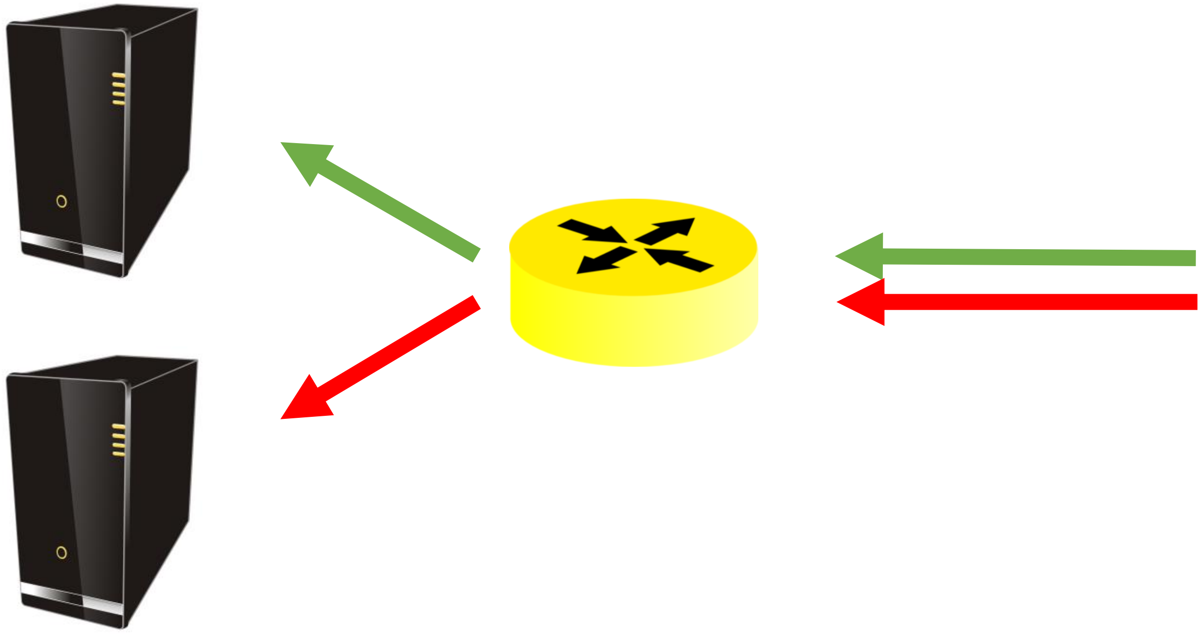
TCAM-based packet processing engine!

- ❑ Extreme flexibility!
  - XFSM ‘programs’ almost flexible as ordinary programming language
    - can define variables, store and change values, compute features, etc
- ❑ Guaranteed wire speed!
  - Fixed time per-packet computational loop
    - 6 clock cycles in our ongoing HW design
- ❑ (currently two tech limitations)
  - Only 1 ALU operation per each packet → ***pipelined ALU arrays*** *possible, but* ***would increase processing time*** *and yield more complex configuration*
  - **ALUs only in update**, not in conditions → *does not permit conditions such as (R1+R2>100)*
    - Solution (not nice, but workaround): compute R1+R2 → R3 during previous packet, then use (R3>100)

DEMO

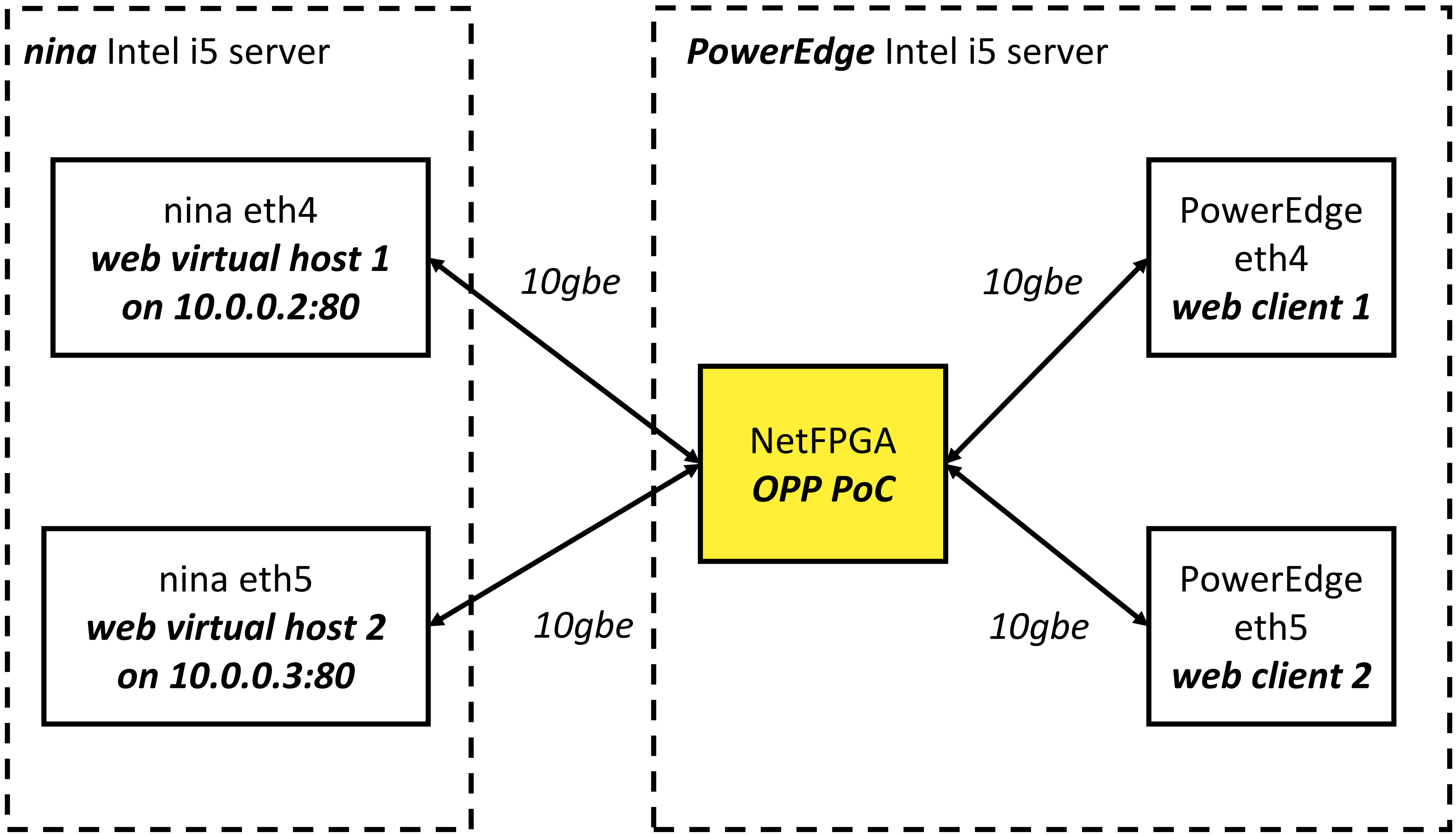
LOAD BALANCING, flow-consistent

Demo high level description

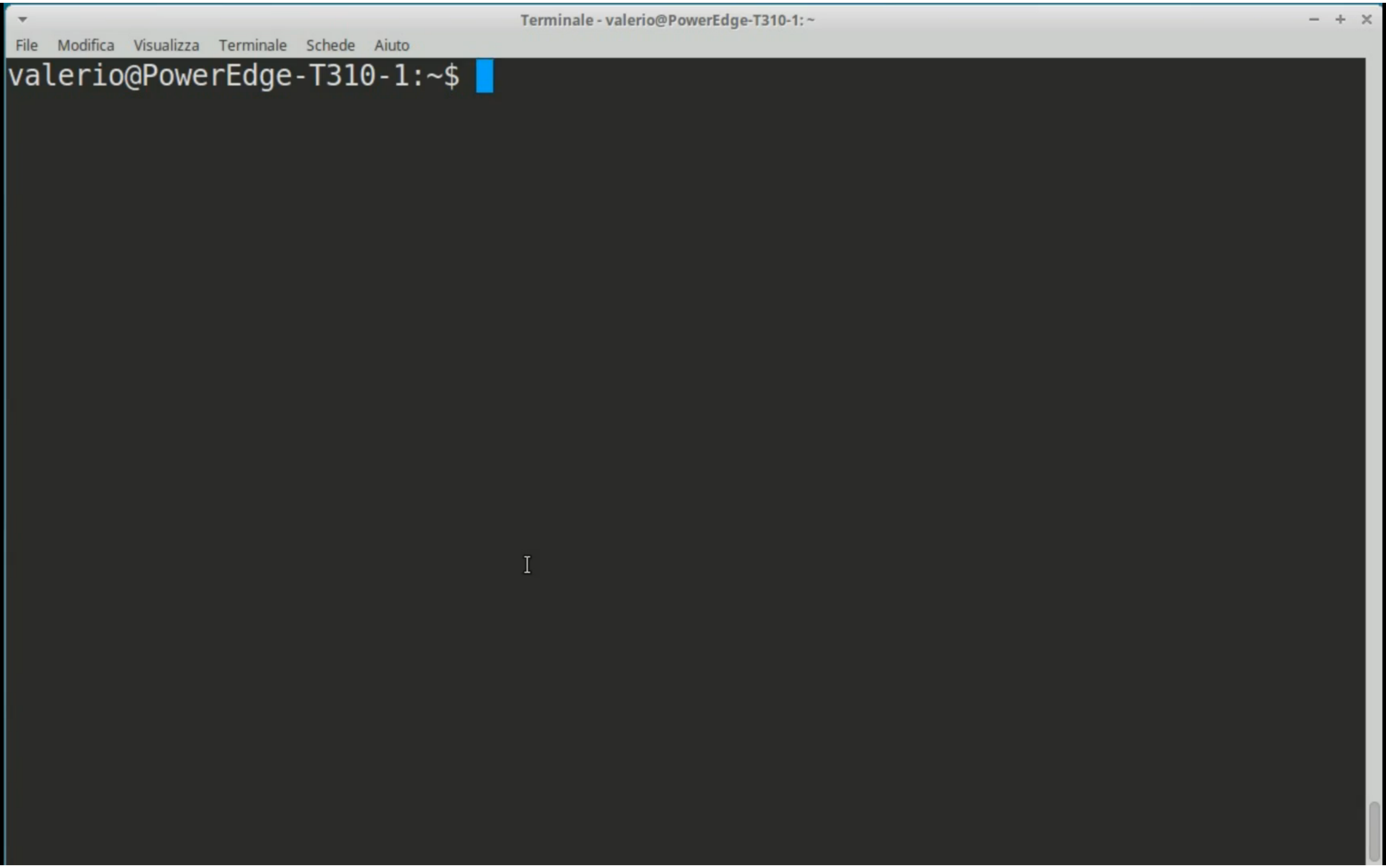


Counter: 2

Demo detailed deployment

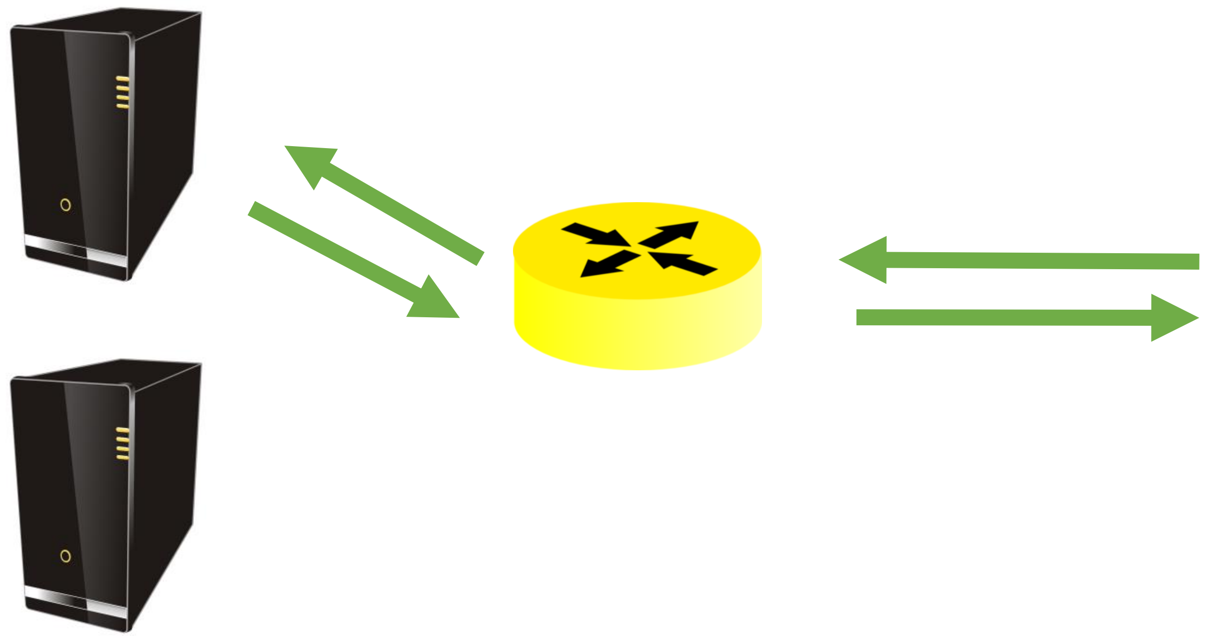


Configuring the NetFPGA

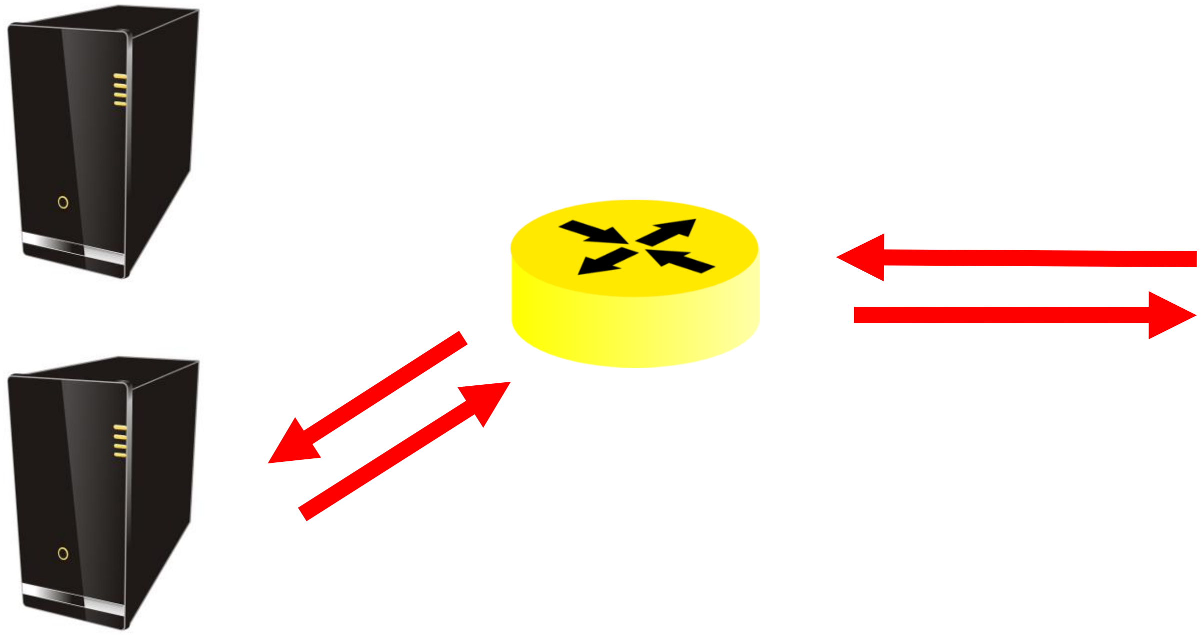

Terminale - valerio@PowerEdge-T310-1: ~
File Modifica Visualizza Terminale Schede Aiuto

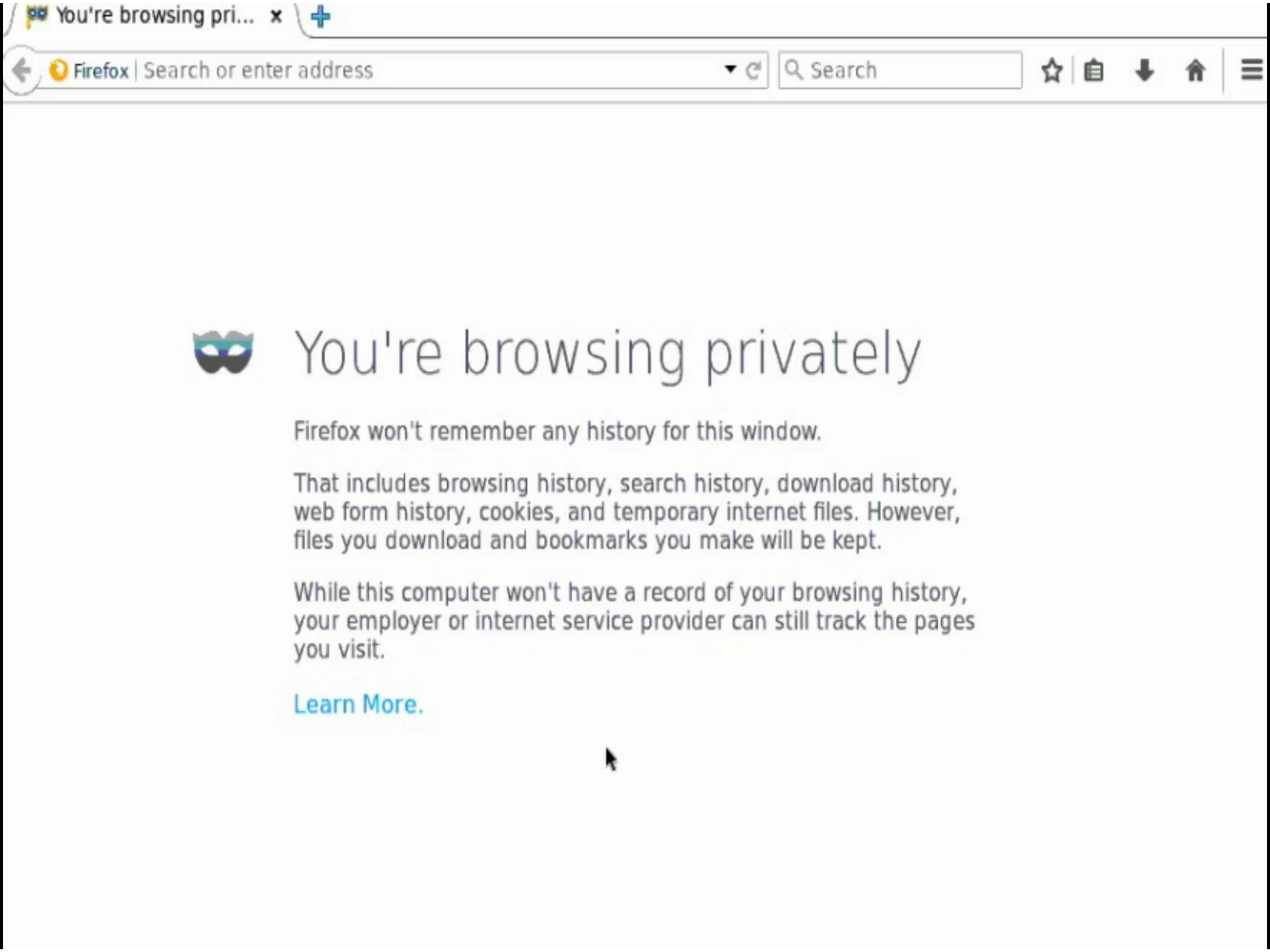
```
valerio@PowerEdge-T310-1:~$
```

WEB client 1 get http://www.sosr-demo.eu



WEB client 2 get http://www.sosr-demo.eu





You're browsing privately

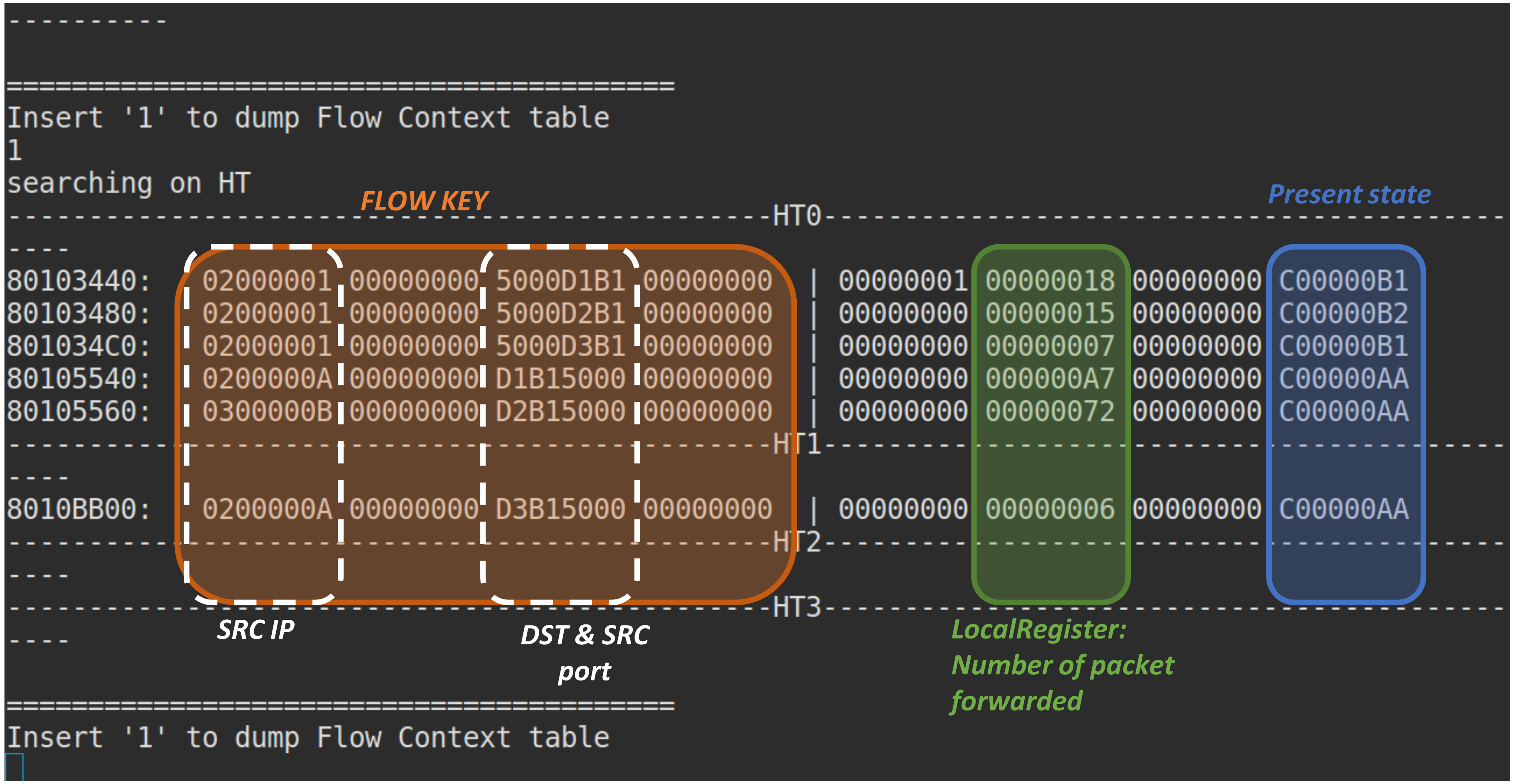
Firefox won't remember any history for this window.

That includes browsing history, search history, download history, web form history, cookies, and temporary internet files. However, files you download and bookmarks you make will be kept.

While this computer won't have a record of your browsing history, your employer or internet service provider can still track the pages you visit.

Learn More.

Dumping Flow Context table



```
----------


========================================
Insert '1' to dump Flow Context table
1
searching on HT
-------------------------------------------------HT0-------------------------------------------
----
80103440:  02000001 00000000 5000D1B1 00000000  | 00000001 00000018 00000000 C00000B1
80103480:  02000001 00000000 5000D2B1 00000000  | 00000000 00000015 00000000 C00000B2
801034C0:  02000001 00000000 5000D3B1 00000000  | 00000000 00000007 00000000 C00000B1
80105540:  0200000A 00000000 D1B15000 00000000  | 00000000 000000A7 00000000 C00000AA
80105560:  0300000B 00000000 D2B15000 00000000  | 00000000 00000072 00000000 C00000AA
-------------------------------------------------HT1-------------------------------------------
----
8010BB00:  0200000A 00000000 D3B15000 00000000  | 00000000 00000006 00000000 C00000AA
-------------------------------------------------HT2-------------------------------------------
----
-------------------------------------------------HT3-------------------------------------------
----

========================================
Insert '1' to dump Flow Context table
```

Thank you!

Contact:

- **Valerio.Bruschi@students.uniroma2.eu**
- **Valerio.Bruschi@cnit.it**