Data Plane Programmability the next step in SDN

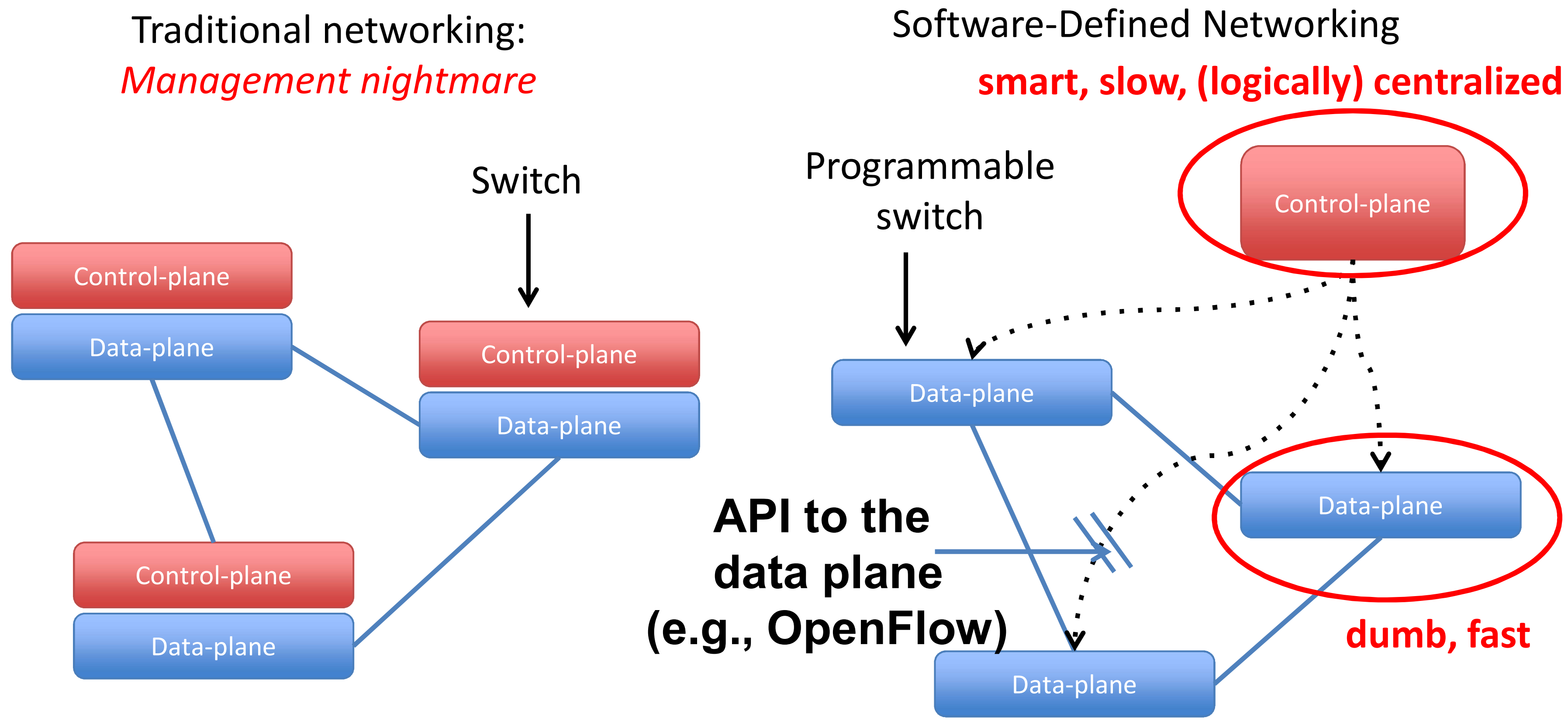
Giuseppe Bianchi
CNIT / University of Roma Tor Vergata

Credits to: *M. Bonola, A. Capone, C. Cascone, S. Pontarelli, D. Sanvito, M. Spaziani Brunella, V. Bruschi*

EU Support:

Once upon a time…

2008: SDN to the rescue



OpenFlow: a compromise
[original quotes: from OF 2008 paper]

- ➔ **Best approach:** "persuade commercial name-brand equipment vendors to provide an open, programmable, virtualized platform on their switches and routers"
  - ⇨ Plainly speaking: *open the box!! No way…*

- ➔ **Viable approach:** "compromise on generality and seek a degree of switch flexibility that is
  - ⇨ High performance and low cost
    - » We already had commodity TCAMs / hash tables!
  - ⇨ Capable of supporting a broad range of ~~research~~ innovation
    - » L2/L3 forwarding, Firewall, etc: at different layers, but all based on flow tables
  - ⇨ **Consistent with vendors' need for closed platforms**
    - » Who cares how the flow table is internally implemented?!

Very, VERY simple – e.g. compare to ForCES. But enough do to something non-trivial

OpenFlow's key insight: match/action abstraction

Programmable logic | **Vendor-implemented**

Matching Rule	Action

1. FORWARD TO PORT
2. ENCAPSULATE&FORWARD
3. DROP
4. …

Extensible

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport

Pre-implemented matching engine

The SDN/OpenFlow Model

- ➔ **Very elegant and performing**
  - ⇨ Switch as a «sort of» programmable device
  - ⇨ Line-rate/fast-path (HW) performance
  - ⇨ Can be «repurposed» as switch, router, firewall, etc
- ➔ **...but...**
  - ⇨ Static rules
  - ⇨ All intelligence in controller
  - ⇨ Lack of flexibility and espessivity: more of a config than a program!

OpenFlow's platform agnostic «program»: (abstract) Flow table

Match 1 → Act A
Match 2 → Act B

Controller

Run-time deployment

(flow-mod)

Match primitives

Match 1 → Act A
Match 2 → Act B

Pre-implemented actions

Networking-specific programmable device
OpenFlow (HW/SW) switch

The SDN/OpenFlow Model

OpenFlow's platform agnostic «program»: (abstract) Flow table

Match 1 → Act A	
Match 2 → Act B	

➔ **Very elegant and p...**

⇨ Switch as...

⇨ Lin...

Consequence

any «smart program» must be delegated to controller → latency!

O(50ms) switch-controller latency = 10 million packets @ 100 gbps

Aftermath

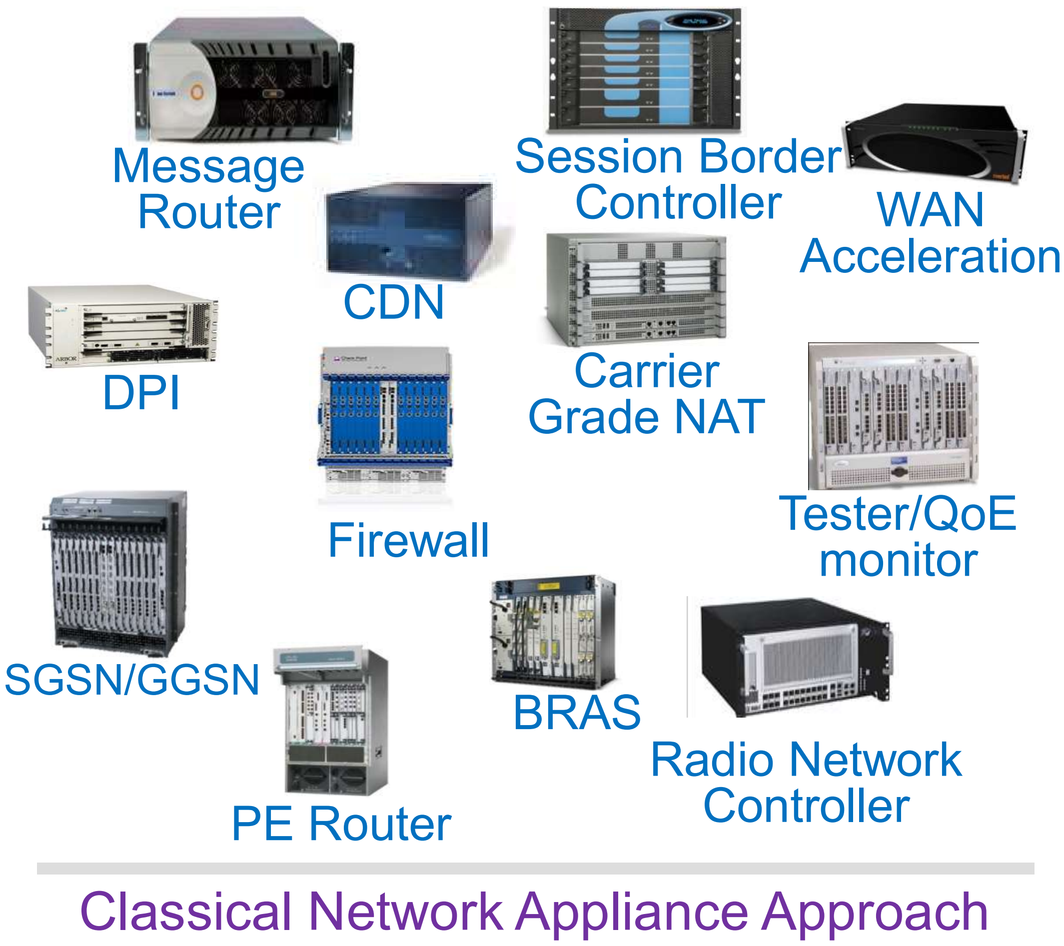
Openflow: was the original SDN enabler…

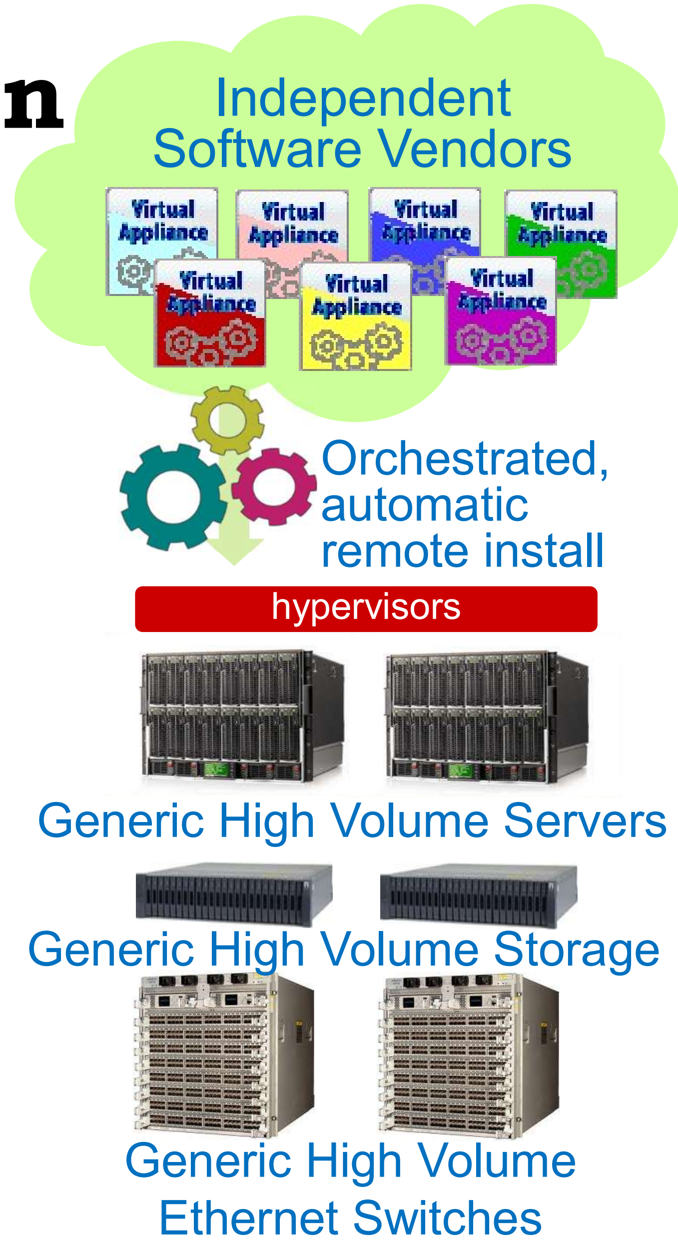
… but now is the SDN sore spot!

...evice

...on

2012: Network Functions Virtualization to the rescue!





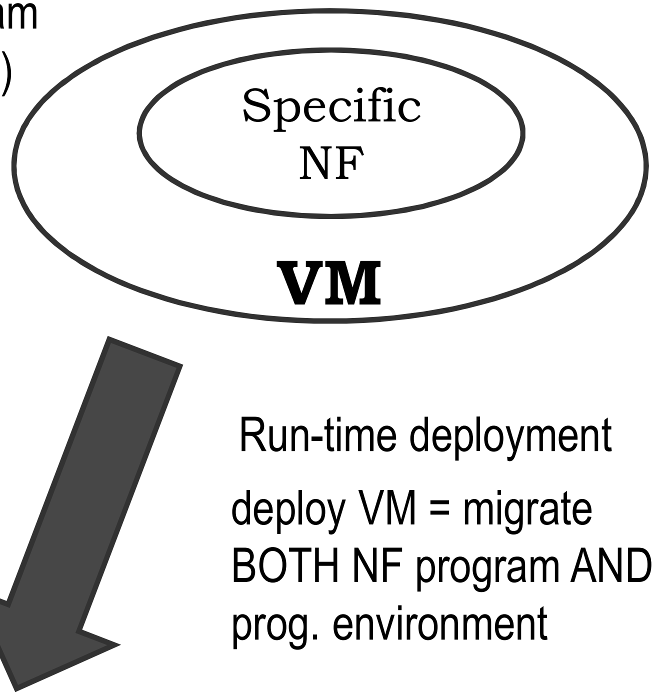
Adapted from Bob Briscoe, BT

The NVF model (opposite extreme than SDN/OF)

- ➔ **Ultra flexible**
  - ⇨ C/C++ coding
- ➔ **...but BIG price to pay...**
  - ⇨ Poor performance (slow path)

- ➔ **Point is: NFV is «just» a software implementation of an NF**
  - ⇨ Not nearly a programming abstraction!!
  - ⇨ As efficient as the implementor makes it efficient!
    - → *take an old crappy code, wrap it in an VM/container/unikernel, and here is your «new» VNF...*

Ordinary SW program
(possibly closed src)



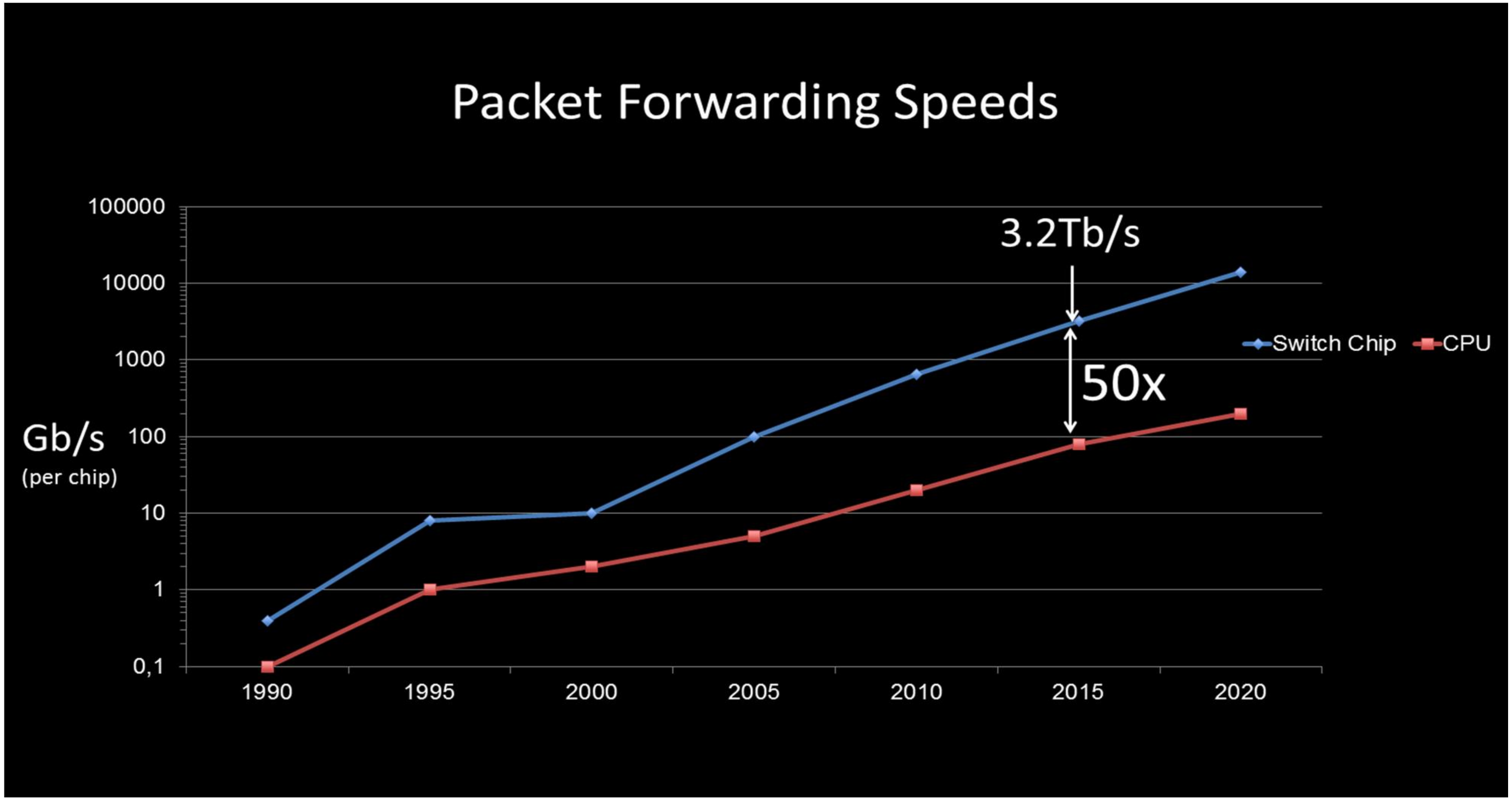
Run-time deployment

deploy VM = migrate
BOTH NF program AND
prog. environment

Virtualization (e.g. hypervisor)

General purpose computing platform
X86, ARM, etc

Fact: CPU-based SW is not a panacea
(especially when performance is key)



Moreover, worth to keep in mind: 1 64B packet @ 100 gbps = 5 ns = 100cm signal propagation (@ 2/3 c)

Source (plot only): Nick McKeown, 2015, Stanford

Towards a new model

- ➔ **Same SDN-like model**
  - ⇨ Based on abstractions
  - ⇨ Native line-rate
  - ⇨ Portable!! (platform independent)
- ➔ **But much closer to the NFV programming needs**
  - ⇨ MUCH more expressive and flexible than OpenFlow
- ➔ **Price to pay:**
  - ⇨ Need for network-specific HW/SW «netlanguage processor»
    - → But still general purpose processor!

Platform agnostic «program»

(key: more expressive programming Abstraction than openflow!!)

NF as script in «netlanguage» (e.g. P4, XFSM, more later)

Controller

Run-time deployment

(inject netlanguage script)

Match primitives

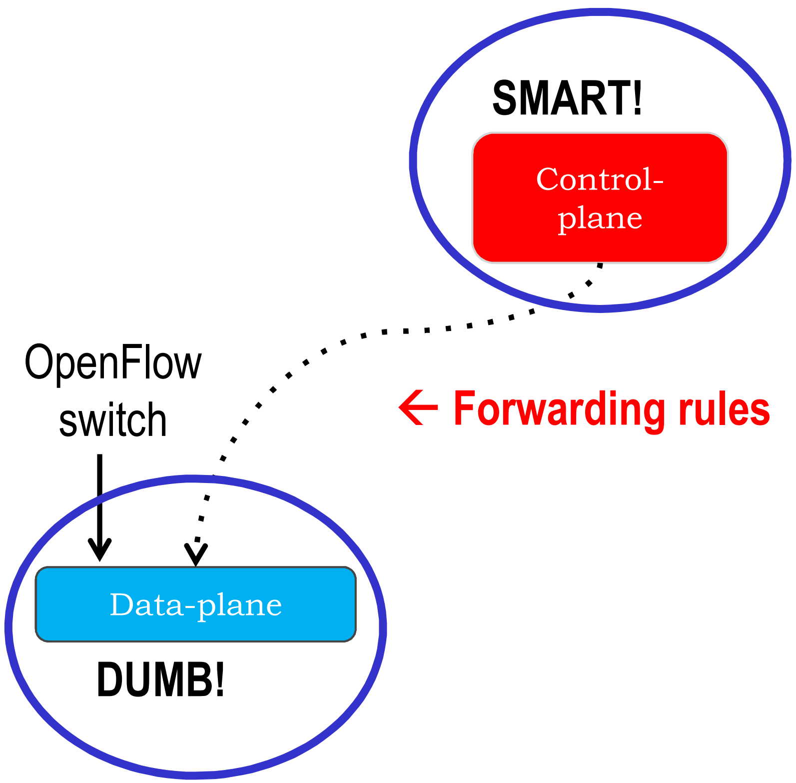
Pre-implemented «netlanguage» execution engine @ fast-path

Pre-implemented actions

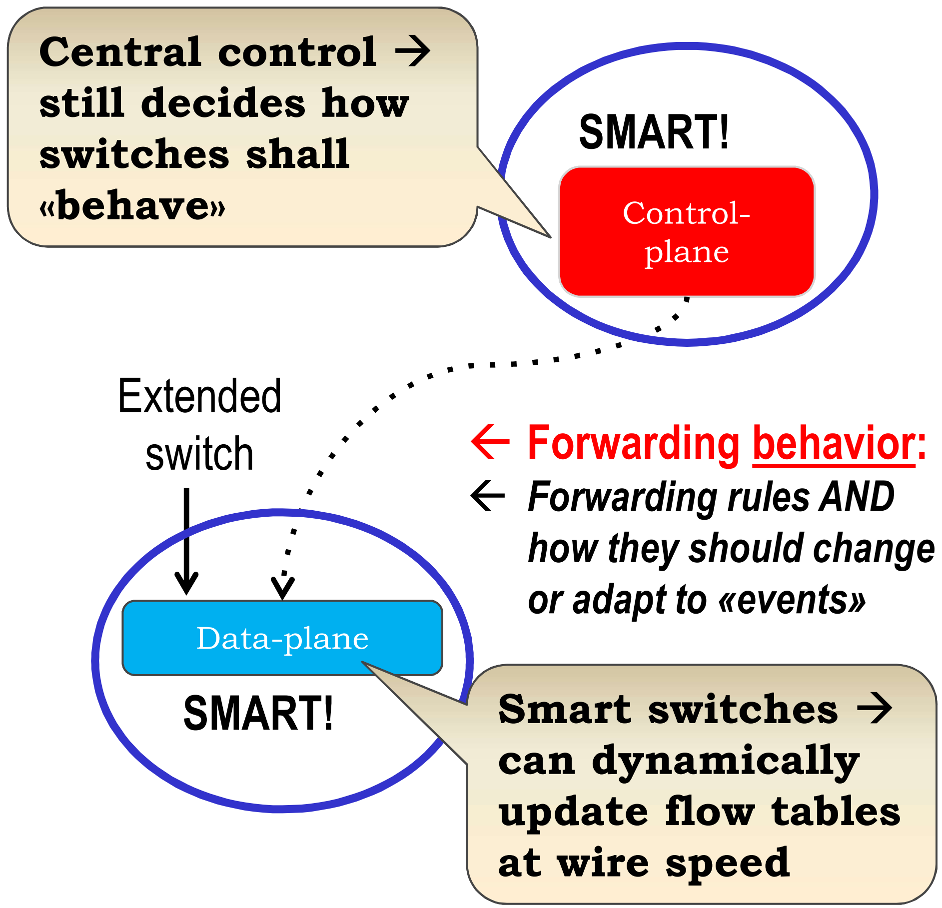
Networking-specific programmable device (HW/SW) switch: not x86/ARM but a general purpose netputing device!

Forwarding rules → forwarding behavior

OpenFlow / SDN



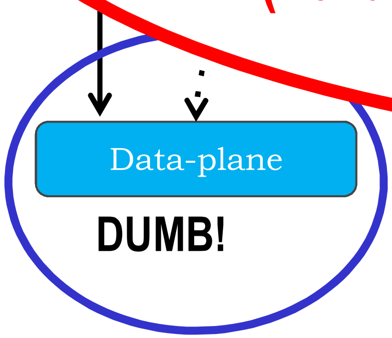
Our view / SDN



Describe forwarding behavior: requires stateful programming abstractions!

Forwarding rules → forwarding behavior

OpenFlow / SDN

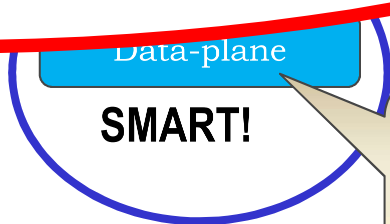


Our view / SDN

Behavioral Forwarding in a nutshell:
Dynamic forwarding rules/states →
some control tasks back (!) into the switch

(hard part: via platform-agnostic abstractions)

...AND
...should change
or adapt to «events»



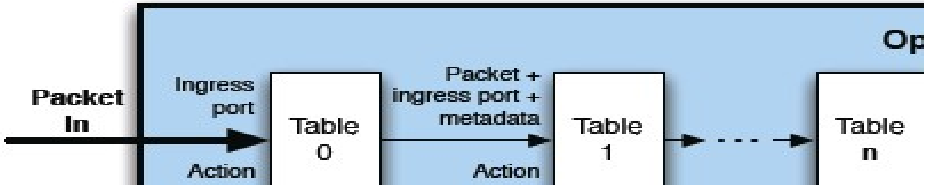
Smart switches → can dynamically update flow tables at wire speed

Describe forwarding behavior: requires stateful programming abstractions!

Towards data plane programmability

OpenFlow evolutions

- **Pipelined tables from v1.1**
  - Overcomes TCAM size limitation
  - Multiple matches natural
    - Ingress/egress, ACL, sequential L2/L3 match, etc.



- **Extension of matching capapilities**
  - More header fields
  - POF (Huawei, 2013): complete matching flexibility!

- **Openflow «patches» for (very!) specific processing needs and states**
  - Group tables, meters, synchronized tables, bundles, typed tables (sic!), etc
  - Not nearly clean, hardly a «first principle» design strategy
  - A sign of OpenFlow structural limitations?

Programming the data plane: The P4 initiative (July 2014)

- **SIGCOMM CCR 2014. Bosshart, McKeown, et al. P4: Programming protocol-independent packet processors**
  - Dramatic flexibility improvements in packet processing pipeline
    - **Configurable packet parser** → parse graph
    - **Target platform independence** → compiler maps onto switch details
    - **Reconfigurability** → change match/process fields during pipeline
- **Feasible with HW advances**
  - Reconfigurable Match Tables, SIGCOMM 2013
  - Intel's FlexPipe™ architectures
- **P4.org: Languages and compilers**
  - Further support for «registry arrays» and counters meant to persist across multiple packets
    - Though no HW details, yet

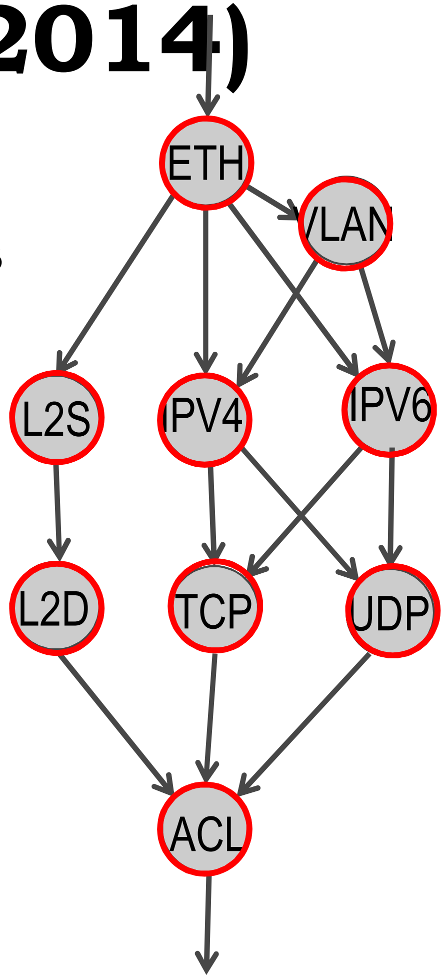


Table Graph

Programming the data plane: The P4 initiative (July 2014)

- ➔ **SIGCOMM CC**
  McKeo
  prot
  - ⇨ Int
- ➔ **P4.org: L**
  - ⇨ Further support fo ... meant to persist across multiple packets
    - → Though no HW details, yet

OpenFlow 2.0 proposal?

Stateful processing, but only «inside» a packet processing pipeline!

Not yet (clear) support for stateful processing «across» subsequent packets in the flow

"[…] extend P4 to express stateful processing",

Nick McKeown talking about P4 @ OVSconf Nov 7, 2016

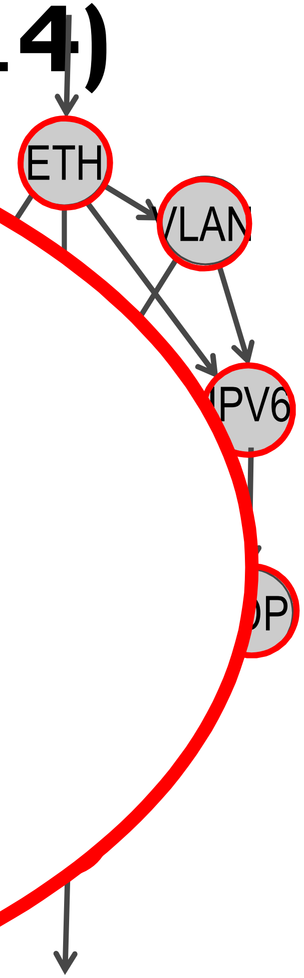
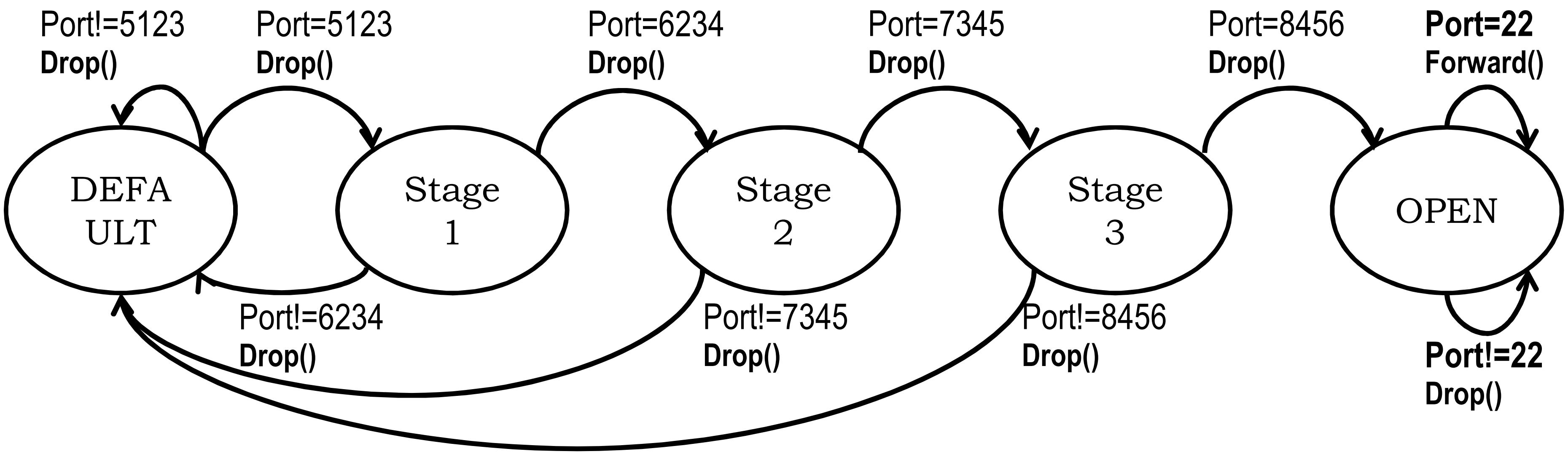


Table Graph

OpenState, April 2014

- **Our group, SIGCOMM CCR April 2014, "OpenState: programming platform-independent stateful OpenFlow applications inside the switch"**
  - surprising result: an OpenFlow switch can «already» support stateful evolution of the forwarding rules
  - With almost marginal (!) architecture modification
- **Our findings at a glance:**
  - Any control program that can be described by a Mealy (Finite State) Machine is already (!) compliant with OF1.3
  - MM + Bidirectional flow state handling requires minimal hardware extensions to OF1.1+
- **Candidate for inclusion in as early as OpenFlow 1.6**
  - Ongoing discussion on fine grained details

Our finding: if application can be «abstracted» as a mealy Machine…



Example: Port Knocking firewall

knock «code»: 5123, 6234, 7345, 8456 → then open Port 22

… it can be transformed in a Flow Table!

Ipsrc: ??

State DB

MATCH: <state, port> → ACTION: <drop/forward, state_transition>

Plus a state lookup/update

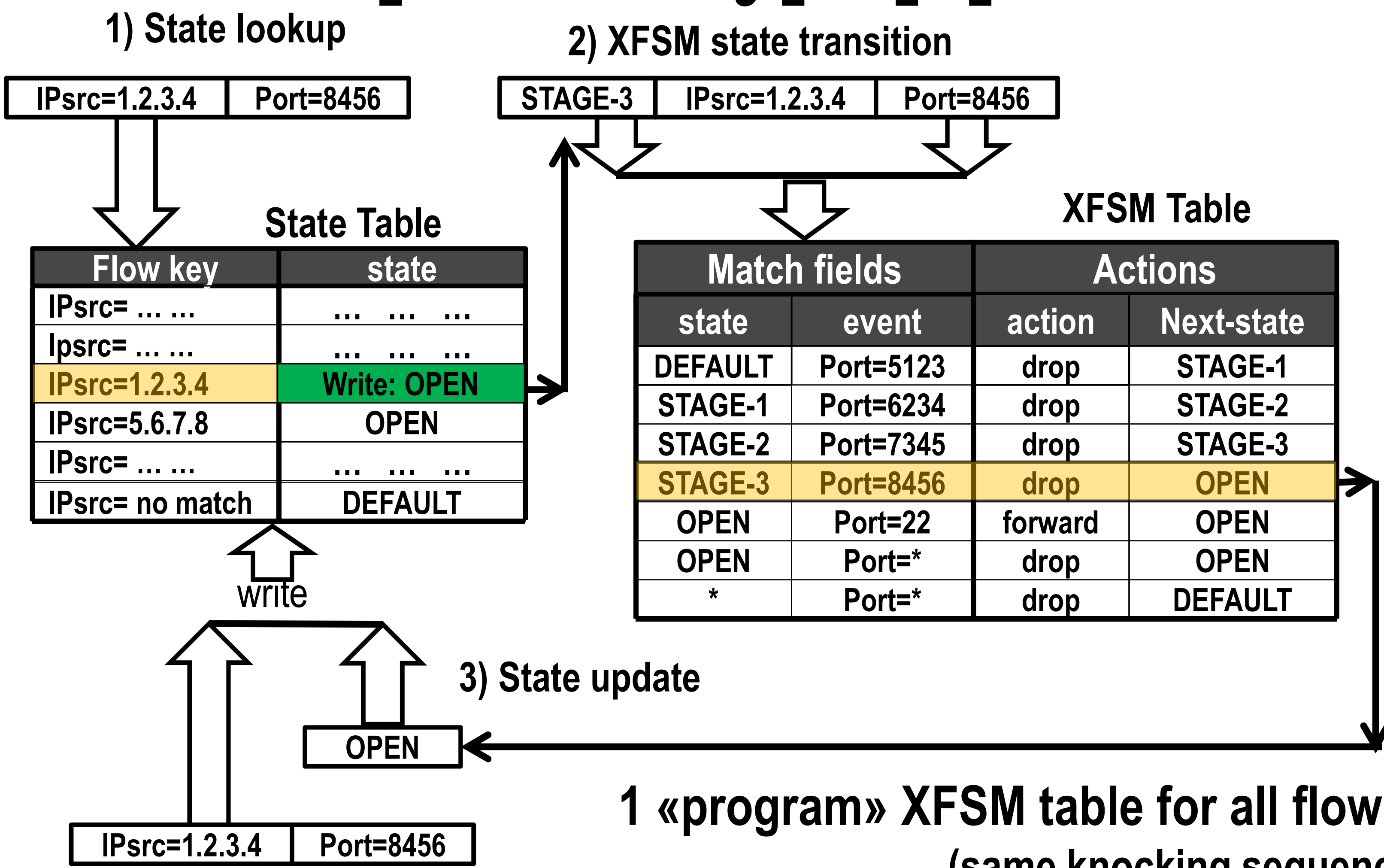
Metadata: State-label	**IPsrc**	**Port**

Match fields		Actions	
state	**event**	**action**	**Next-state**
DEFAULT	Port=5123	drop	STAGE-1
STAGE-1	Port=6234	drop	STAGE-2
STAGE-2	Port=7345	drop	STAGE-3
STAGE-3	Port=8456	drop	OPEN
OPEN	Port=22	forward	OPEN
OPEN	Port=*	drop	OPEN
*	Port=*	drop	DEFAULT

Ipsrc→OPEN

State DB

And “executed” inside a two stage openflow-type pipeline



Flow key	state
IPsrc= … …	… … …
Ipsrc= … …	… … …
IPsrc=1.2.3.4	Write: OPEN
IPsrc=5.6.7.8	OPEN
IPsrc= … …	… … …
IPsrc= no match	DEFAULT

Match fields		Actions	
state	event	action	Next-state
DEFAULT	Port=5123	drop	STAGE-1
STAGE-1	Port=6234	drop	STAGE-2
STAGE-2	Port=7345	drop	STAGE-3
STAGE-3	Port=8456	drop	OPEN
OPEN	Port=22	forward	OPEN
OPEN	Port=*	drop	OPEN
*	Port=*	drop	DEFAULT

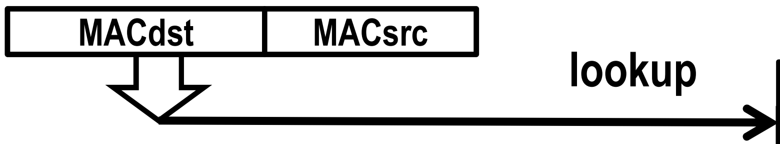
1 «program» XFSM table for all flows

(same knocking sequence)

N states, one per (active) flow

Cross-flow state handling

➔ **Yes but what about MAC learning, multi-port protocols (e.g., FTP), bidirectional flow handling, etc?**

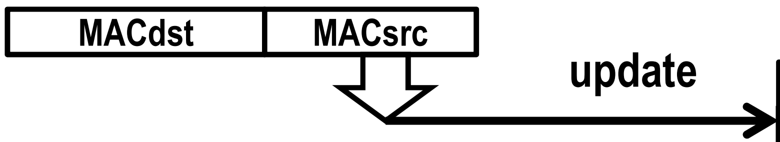


State Table

Flow key	state
48 bit MAC addr	Port #

XFSM Table

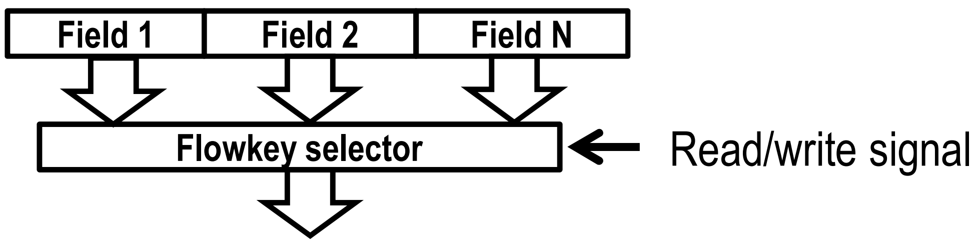
state	event	action	Next-state
Port#	*	forward	In-port



State Table

Flow key	state
48 bit MAC addr	Port #

DIFFERENT lookup/update scope

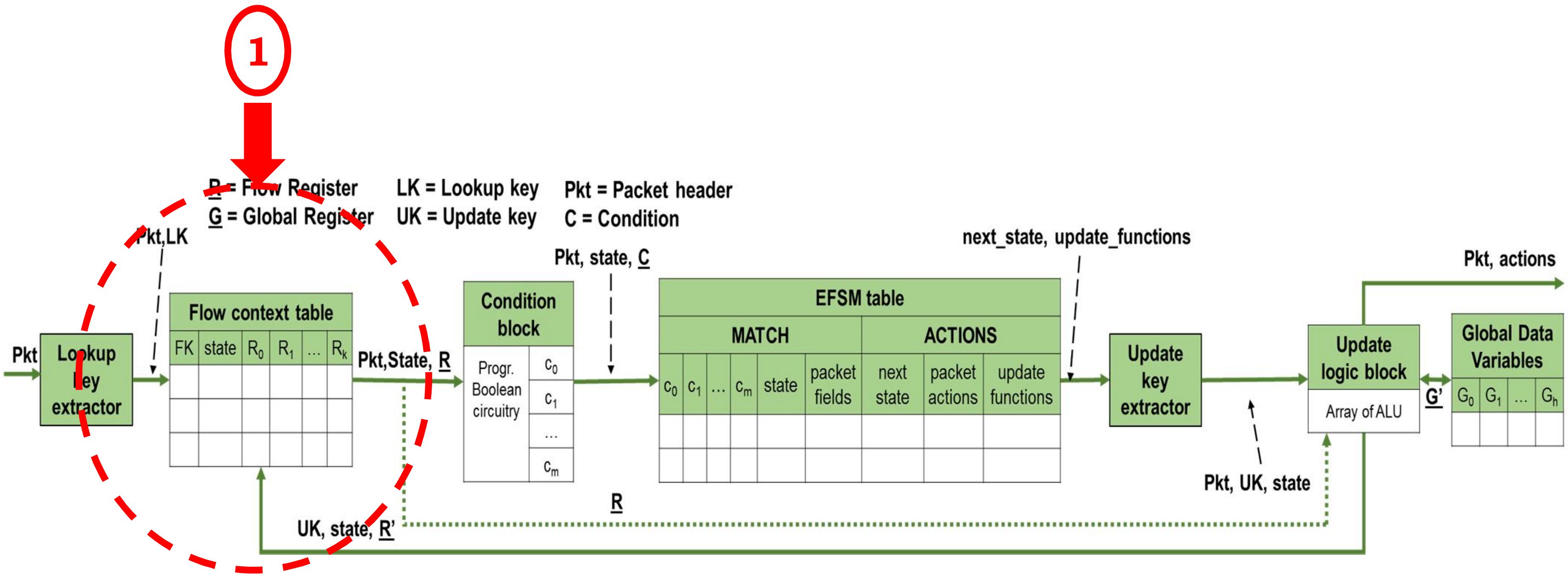


Beyond Mealy machines?

- **Mealy machines: a huge step forward from (static) OpenFlow, but still far from «true» programmability**
  - No «user-defined» variables
  - No arithmetic operations
  - No conditional execution

- **Better abstraction: extended finite state machines (Open Packet Processor, 2016, arxiv)**
  - Turing-complete
  - Abstraction still based on matches (events) and actions
    - A la OpenFlow, but with much more behavioral logic
  - Can STILL be executed on the fast path!
    - Proved with concrete architecture and HW implementation
  - What you write (XFSM) is guaranteed to execute in bounded # of clocks
    - No compiler on target… which may not compile…
  - Multiple stateful processing stages can be pipelined
    - As per OpenFlow Multiple Match/action pipelines
  - Require new HW beyond OpenFlow

Open Packet Processor at a glance

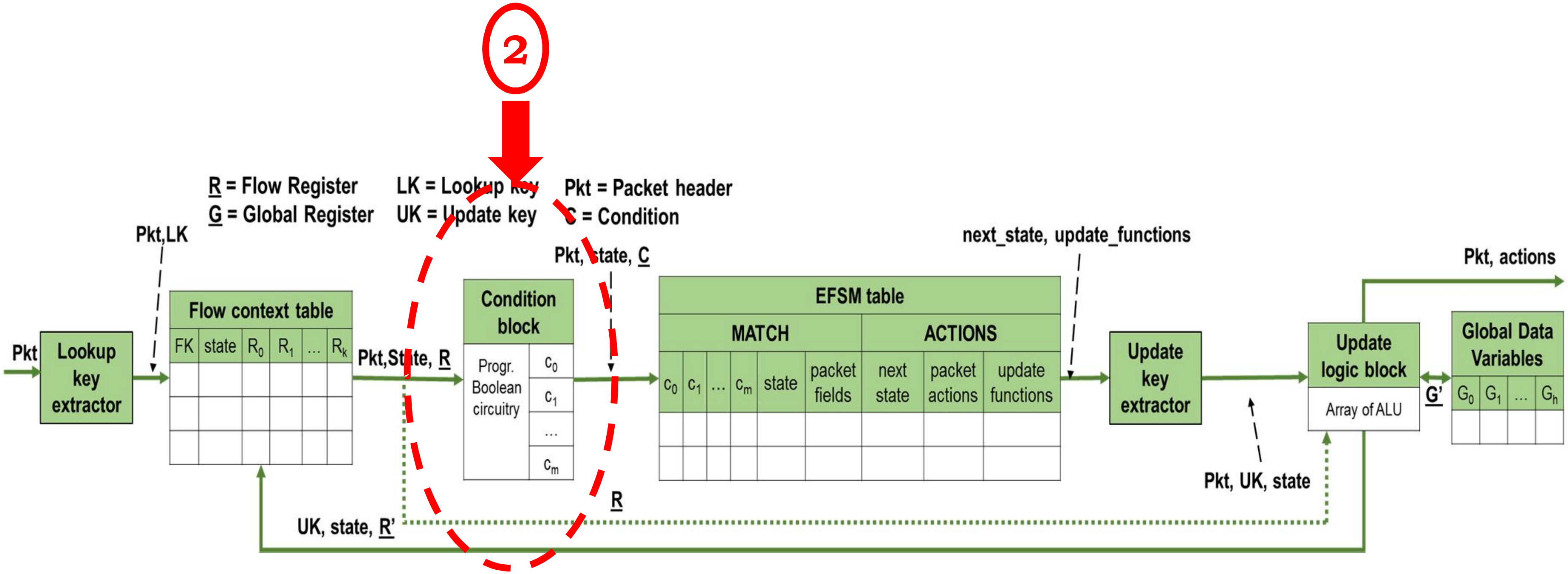


Flow context retrieval

Tell me what flow the packet belongs to

and what is its state (and associated registries)

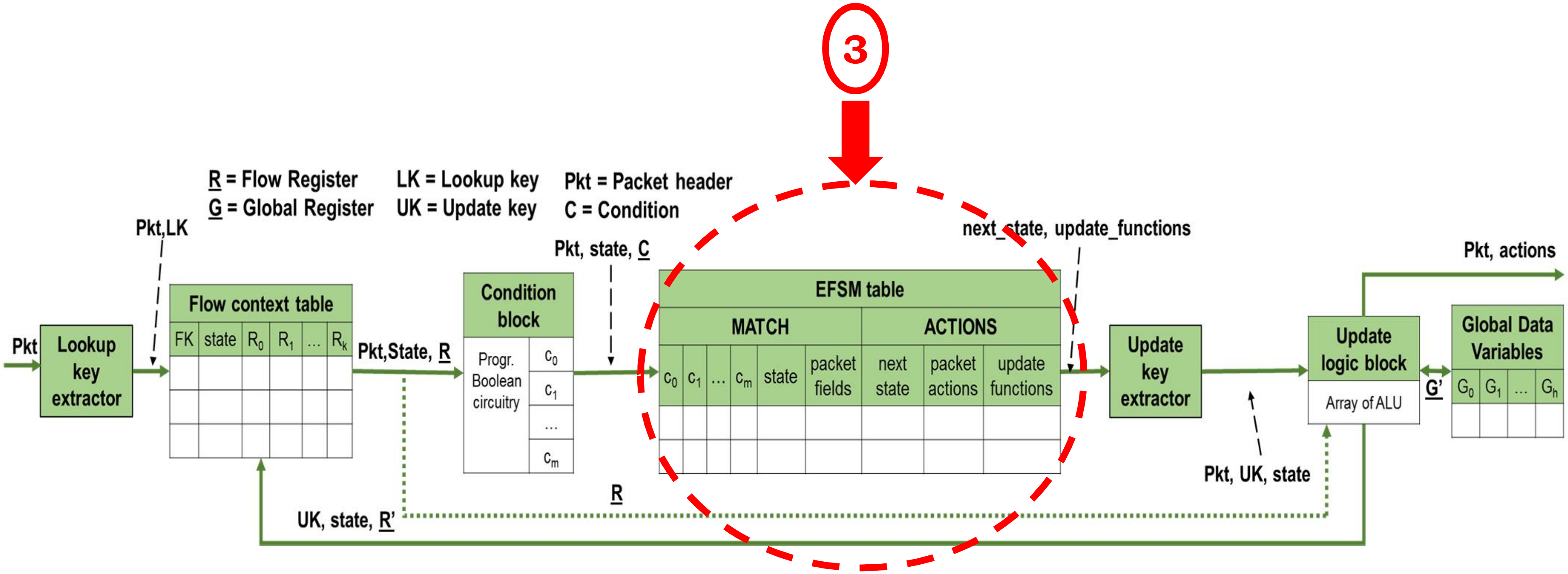
Open Packet Processor at a glance



Condition verification

Does the flow context respect some (user defined) conditions?

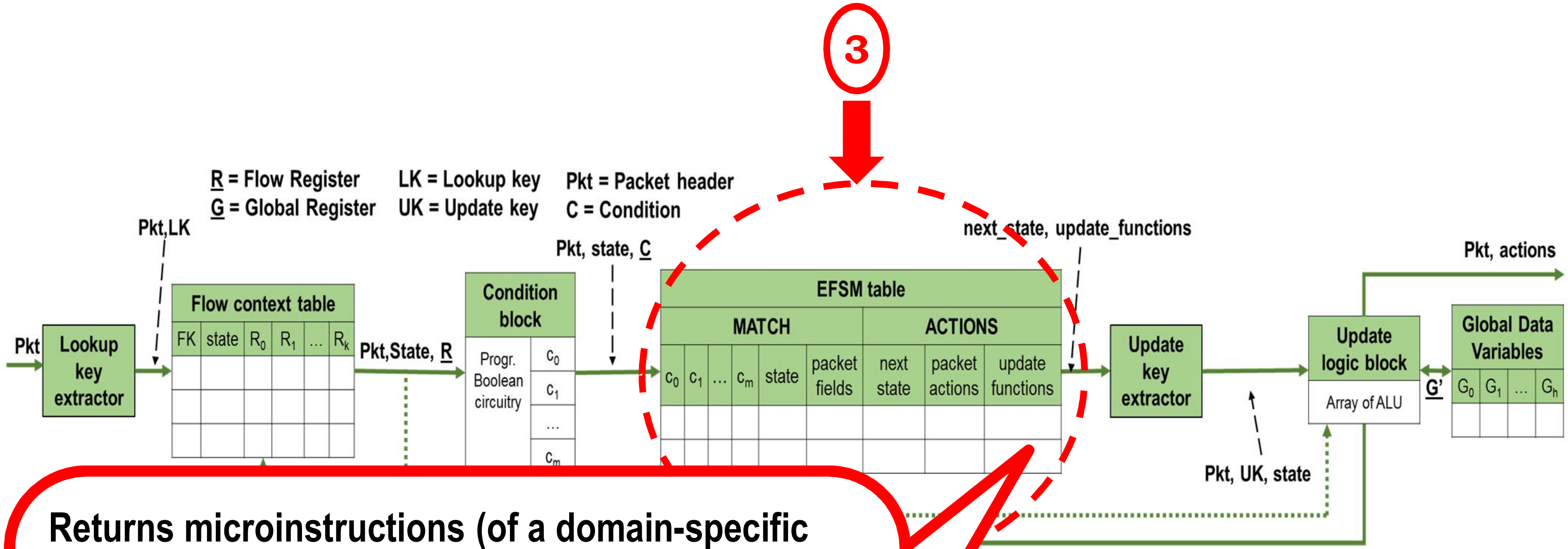
Open Packet Processor at a glance



XFSM execution

Match current status and conditions and retrieve next state and update functions (fetch packet actions)

Open Packet Processor at a glance



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

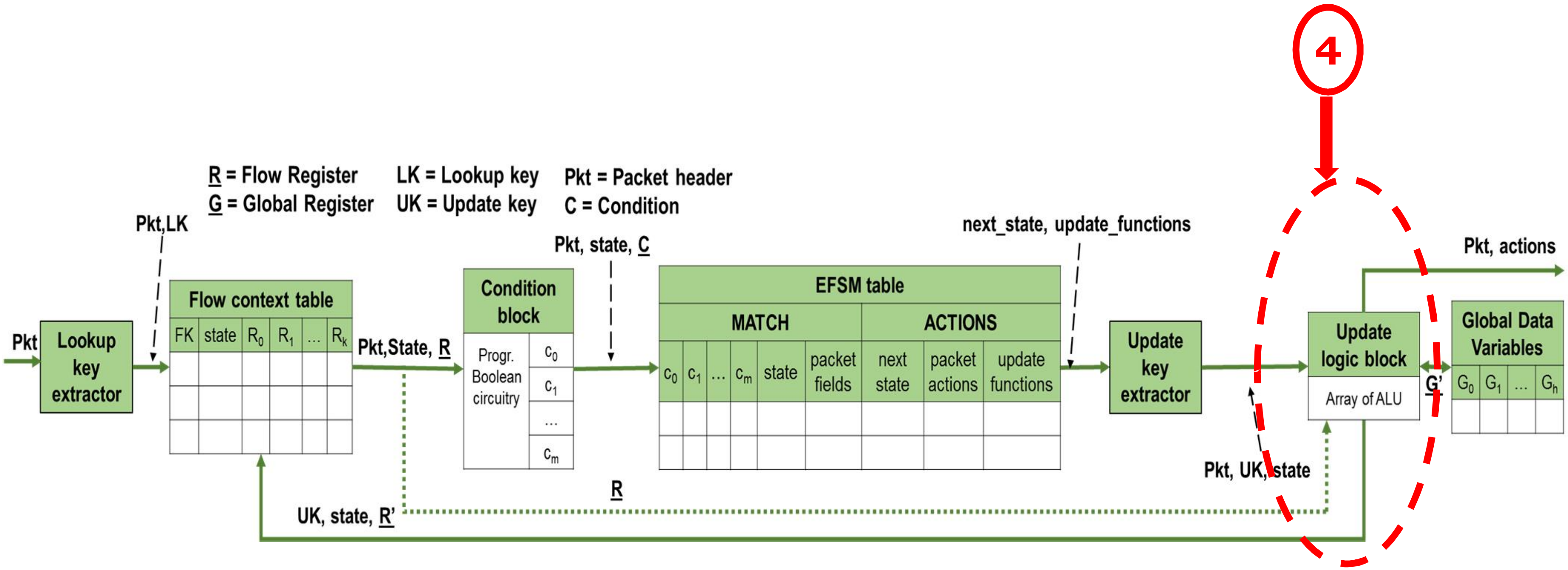
Instruction Type	Instructions	note
Logic ALU Instruction	NOP, AND, OR, XOR, NOT	standard logic operations
Arithmetic ALU Instruction	ADD,ADC, SUB,SBC,MUL	standard arithmetic operations
Shift/Rotate Instruction	LSL (Logical Shift Left) LSR (Logical Shift Right) ASR (Arithmetic Shift Right) ROR (Rotate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

tion

nditions and retrieve

s (fetch packet actions)

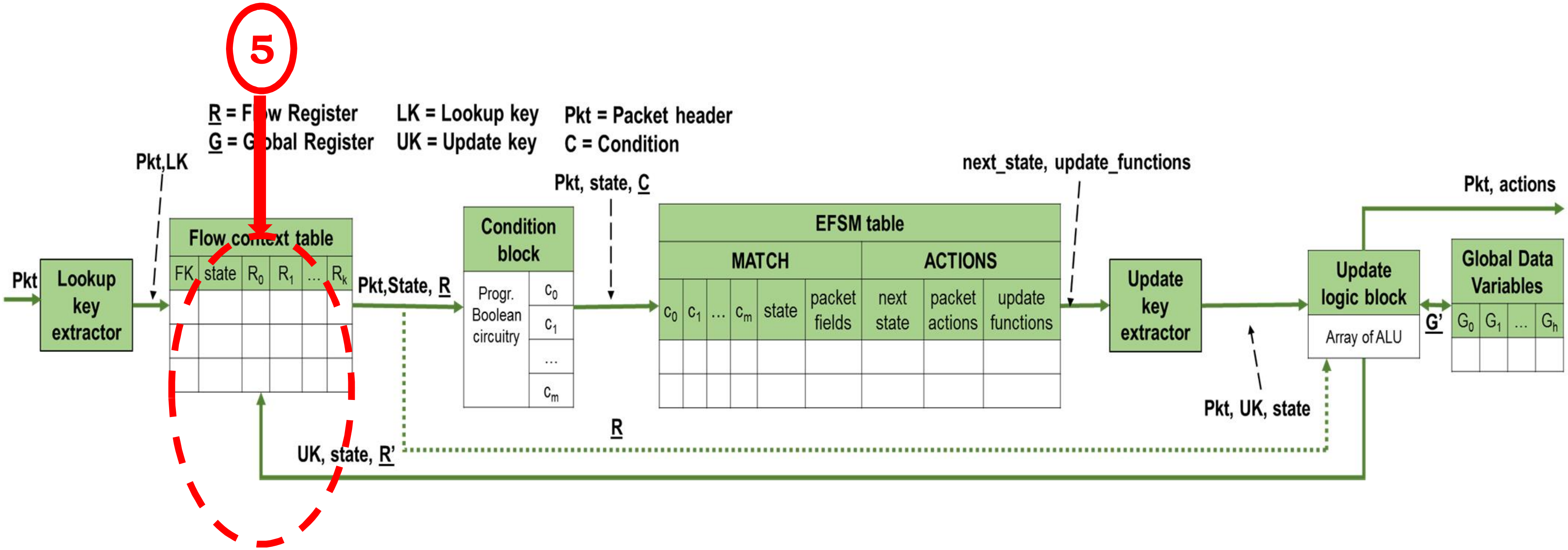
Open Packet Processor at a glance



Execute μ-instructions

Permits to embed user-defined computation in the pipeline

Open Packet Processor at a glance



And update state and registers for the next packet

Close the "computational loop" – no CPU involved

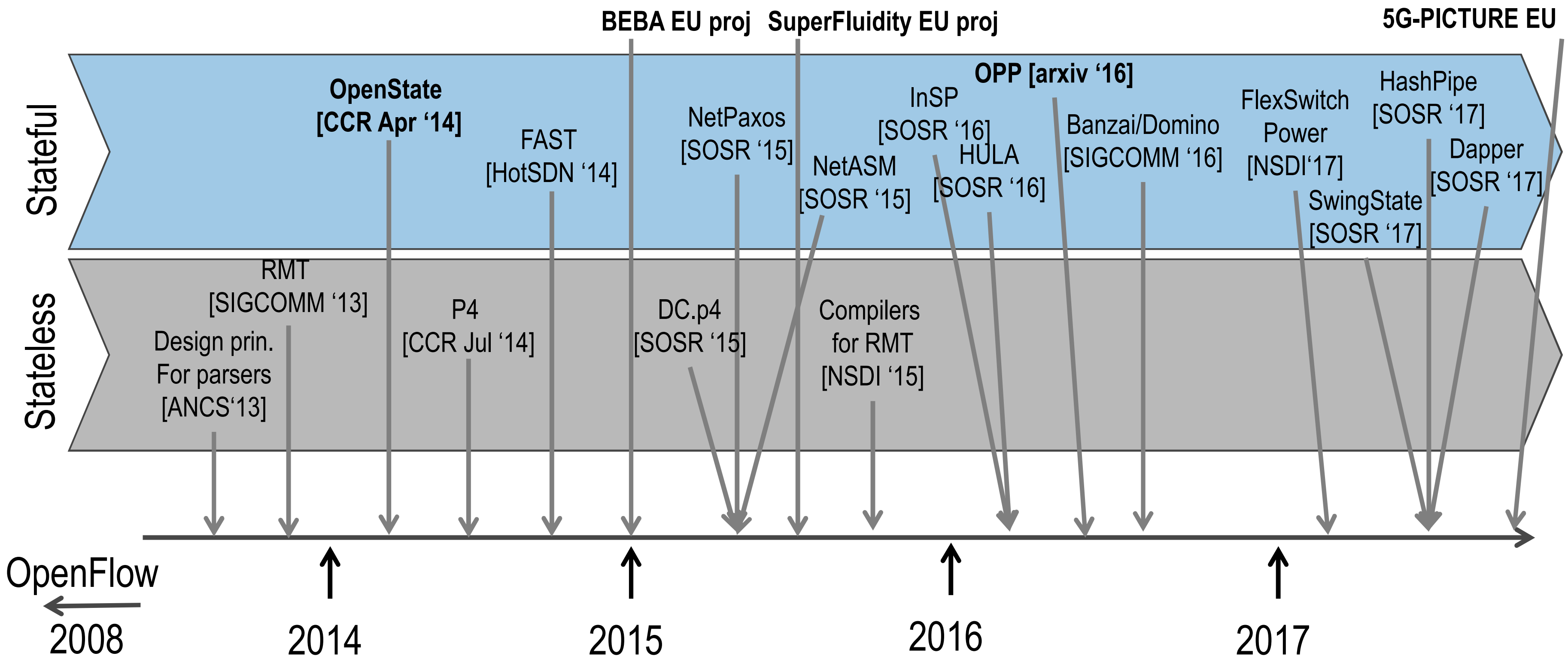
TCAM as state transition engine and ALUs as processing functions

Data plane programmability on the rise…

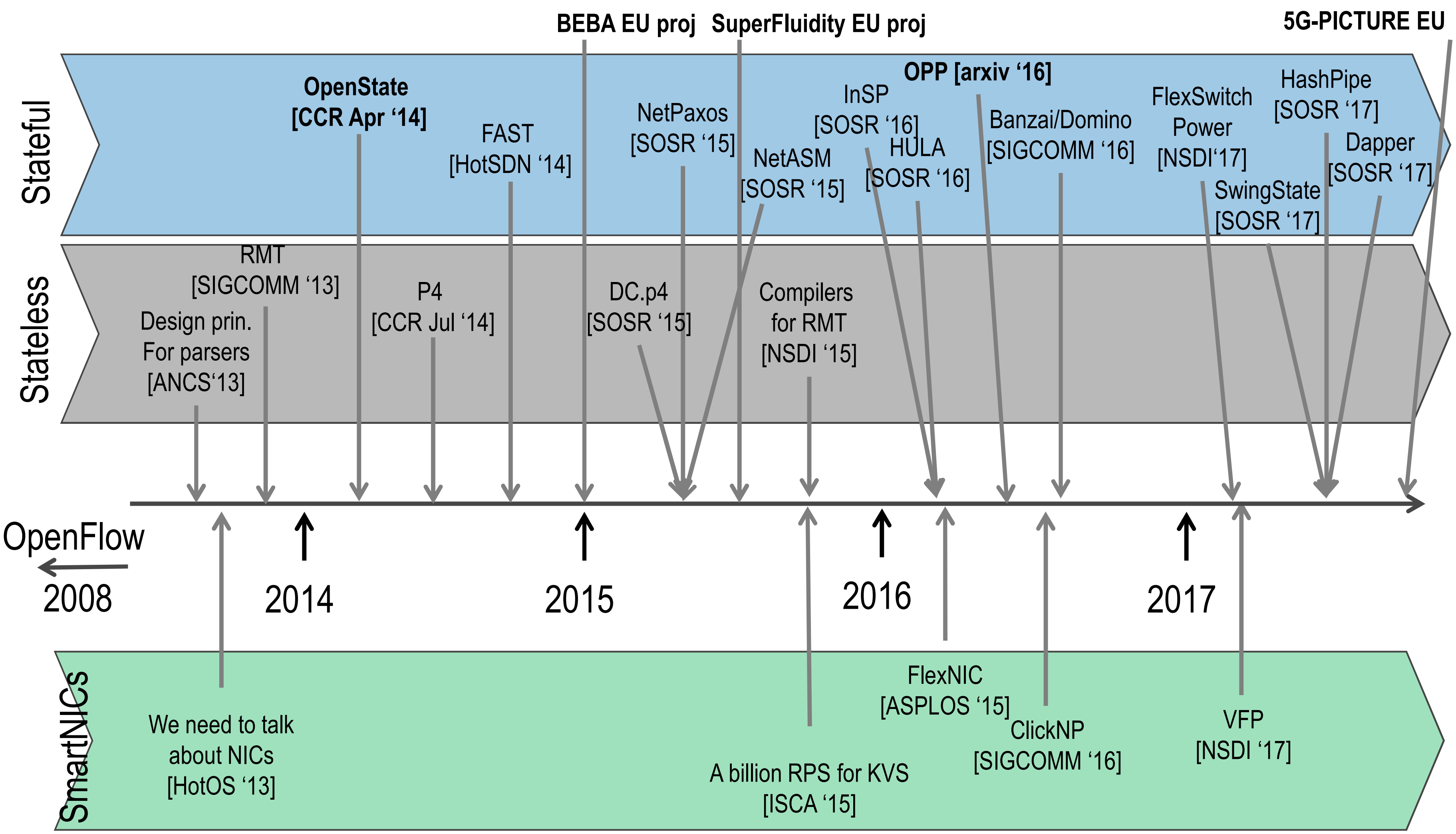
BEBA EU: design platform agnostic programmable stateful data plane
→ leveraging and extending OpenState, standardization target towards ONF

SUPERFLUIDITY EU: further steps in functional decomposition and actions' programmability
→ (see «smashing» paper @ 11 AM)

5G-PICTURE EU: just started; whole WP on data plane programmability and exploitation in 5G

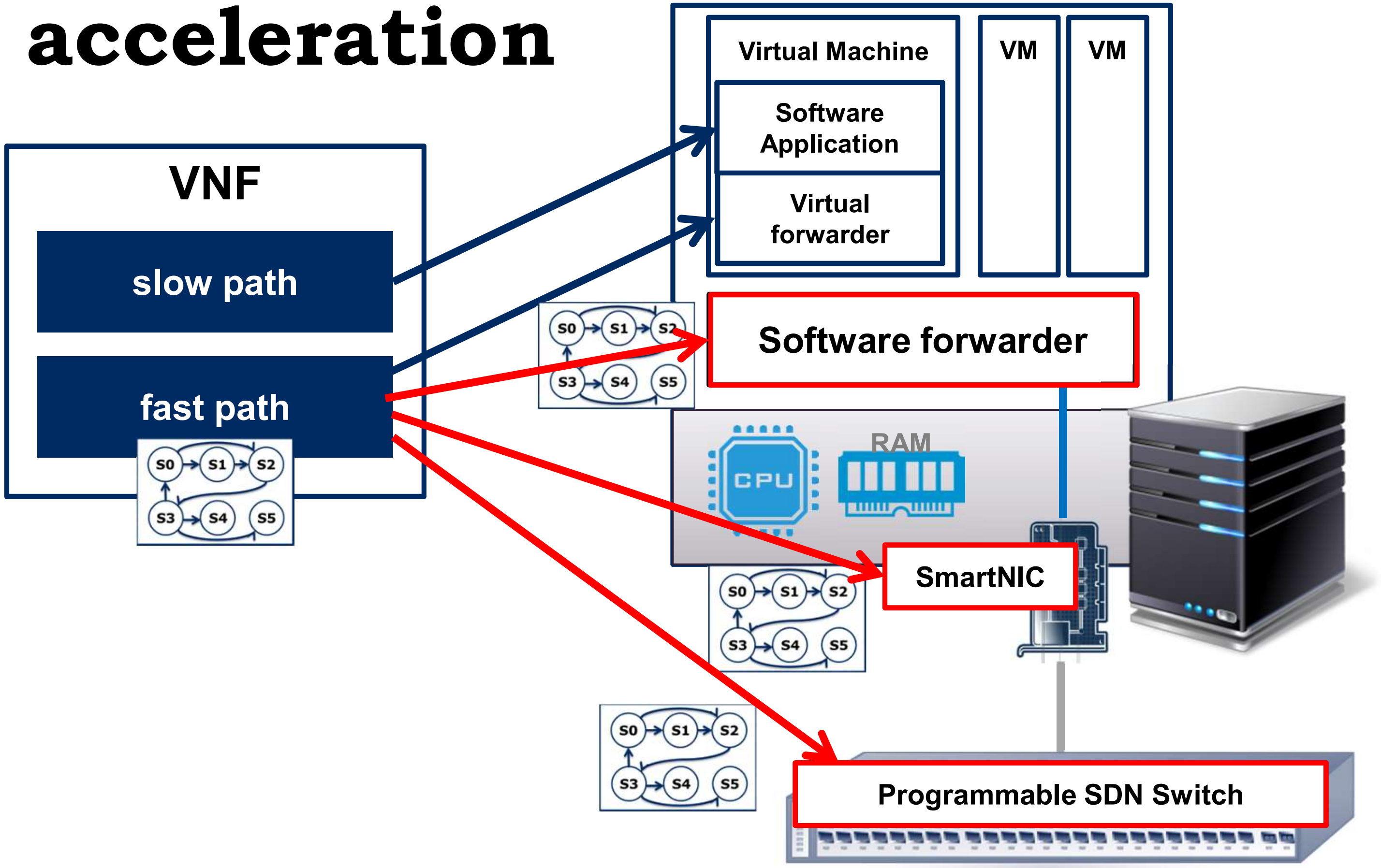


And smart NICs as well…



Not only (classical) SDN, then!

→An interface for <u>virtual network function</u> (VNF) fast path acceleration



Conclusions

- ➔ **Platform-agnostic programming of control intelligence <u>inside devices' fast path</u> seems viable**
  - ⇨ «small» OpenFlow extension – OpenState in (most likely) OpenFlow 1.6?
  - ⇨ TCAM as «State Machine processor»
    - →OpenState → Mealy Machines; OPP → full XFSM
    - →**without any slow path CPU**
  - ⇨ What about programmable actions?
    - →Not only P4; our proposal:tailored MIPS/VLIW (see talk @ 11 AM)

- ➔ **Rethinking control-data plane SDN separation**
  - ⇨ Control = Decide! Not decide+enforce!
  - ⇨ Data Plane programmability: delegate smart execution down in the switches!
  - ⇨ Back to active networking 2.0? (but now with a clearcut abstraction in mind)

- ➔ **VNF offloading**
  - ⇨ Program VNF fast path using data plane abstractions, and make it executable «everywhere» → e.g. in smart HW NIC implementing OPP…