From dumb to smarter switches in software defined networks: towards a stateful data plane

Antonio Capone

Politecnico di Milano – ANTLab

joint work with:

G. Bianchi, M. Bonola, S. Pontarelli
– Univ. Rome Tor Vergata
C. Cascone, L. Pollini, D. Sanvito
– Politecnico di Milano

supported by EU project

Beba
BEhavioural BAsed forwarding

www.beba-project.eu

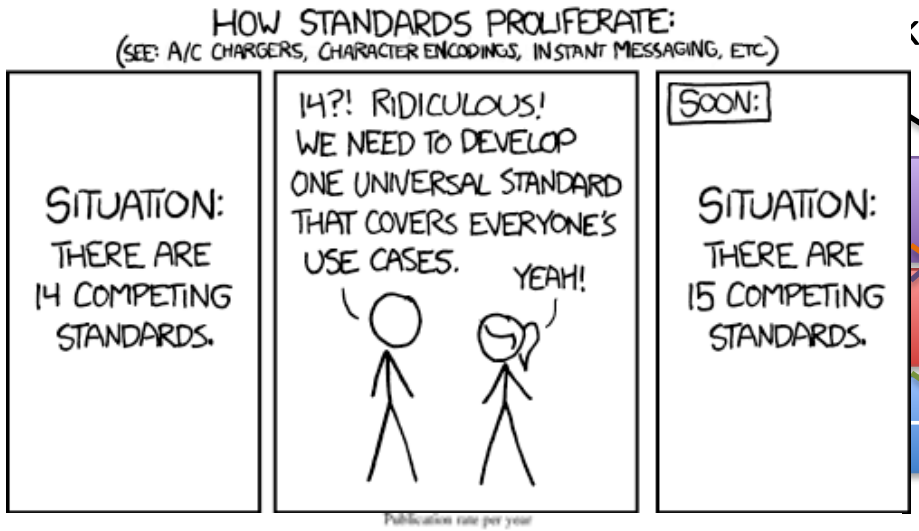
Outline

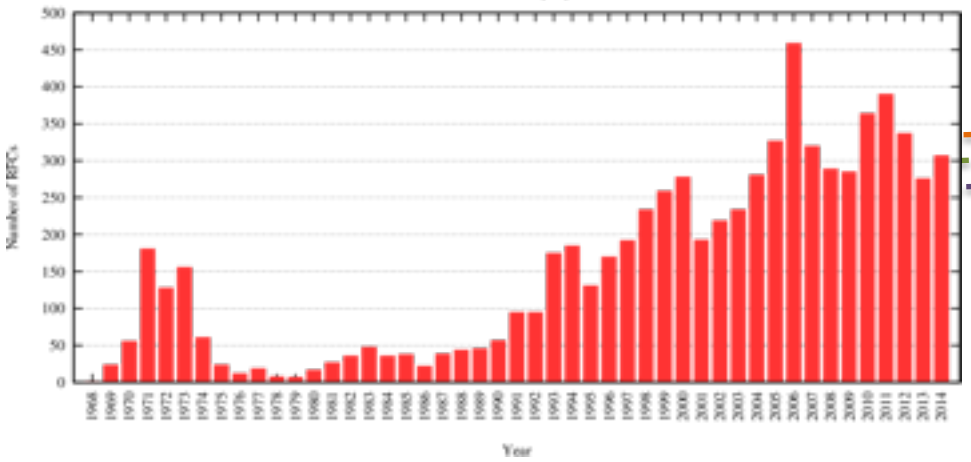
1) ***Switches cannot remain dumb:***
   - → **Data plane evolution.**
   - ⊙ **How far should we go?**
2) ***Not too much not too little:***
   - → ***OpenState and stateful data planes***
3) ***Applied smartness: stateful applications***
   - ⊙ **How to incorporate statefulness in languages?**

Setting the scene: OpenFlow and dataplane abstraction

The future has already arrived. It's just not evenly **distributed** yet. [William Gibson]

Classic network paradigm





Router/switch/appliance

functions

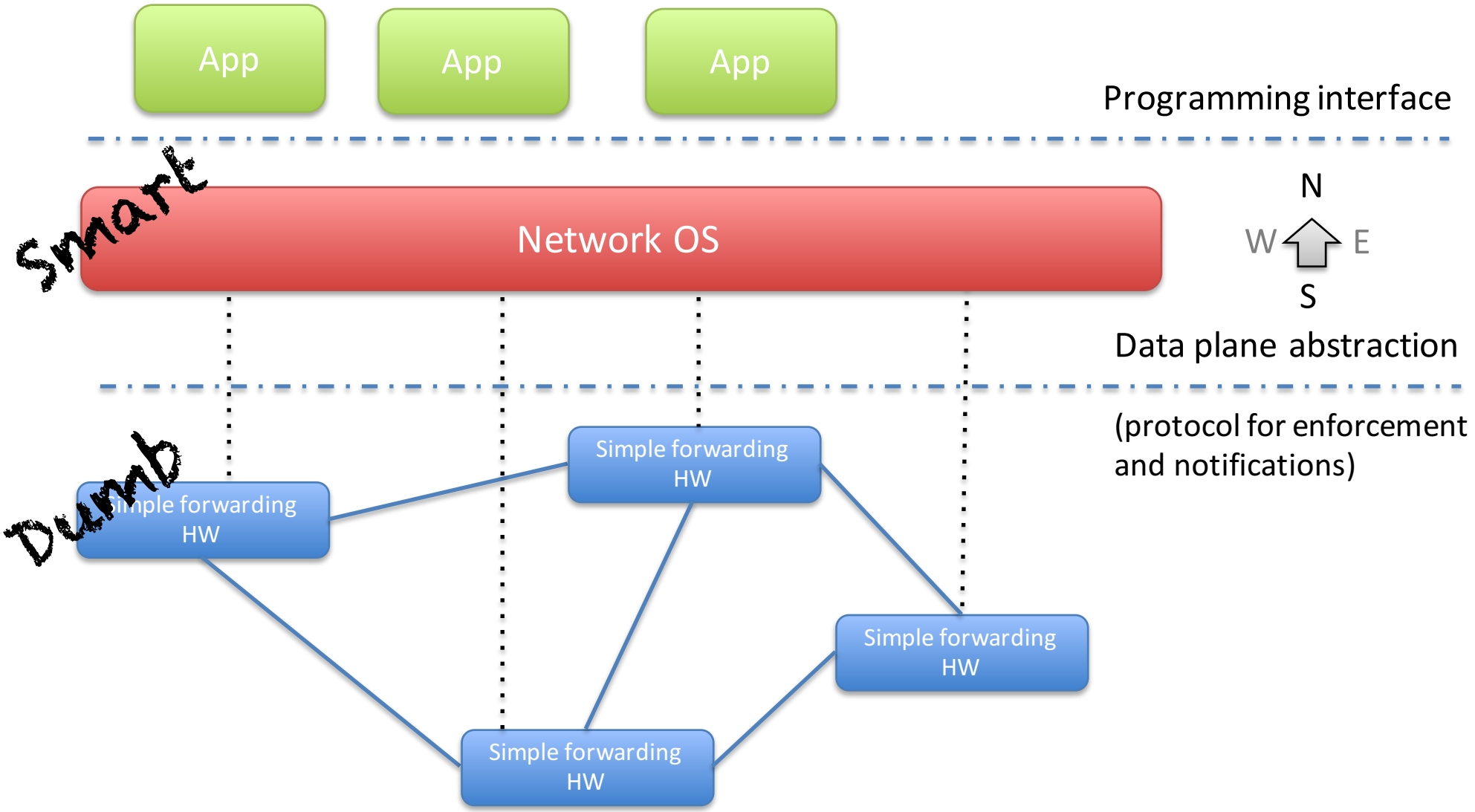
State distribution mechanism
(standard protocols)

OS

Forwarding HW

Closed platform!

SDN paradigm



Several attempts …

Active Networks, IETF ForCES, …

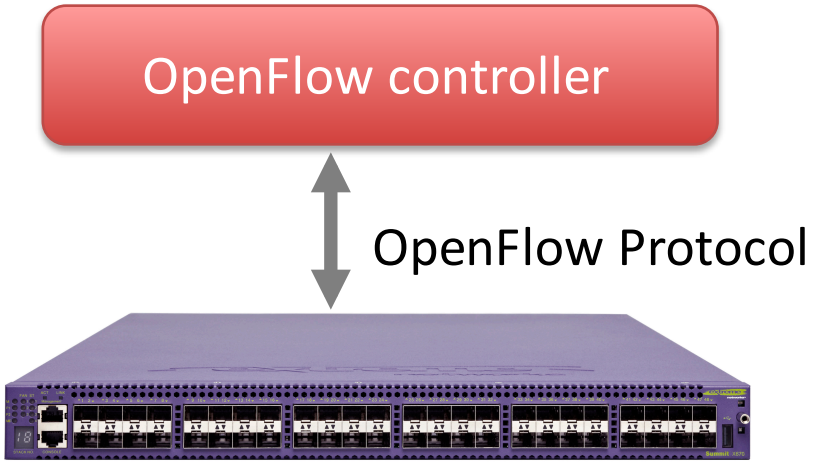
… but one winner

OpenFlow

OpenFlow

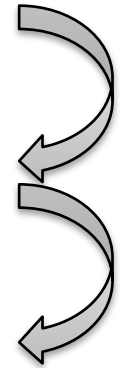
- OpenFlow has been proposed as a **clean slate approach** for new generation network

- … bit it is actually a **pragmatic approach** to SDN based on a simple HW abstraction that can be implemented with current **HW commercial platforms**



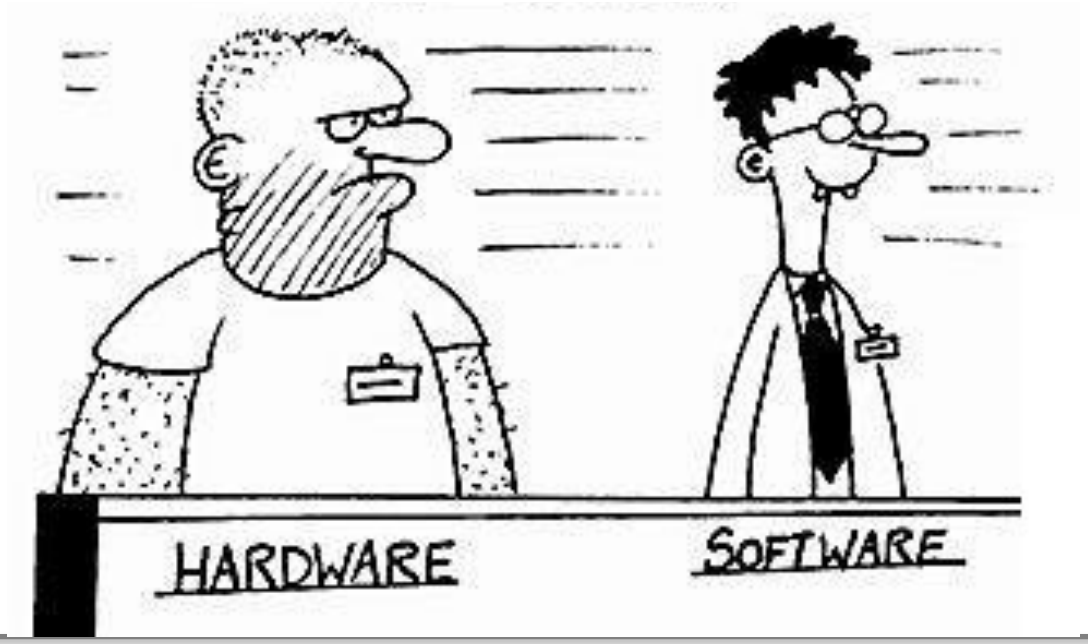
HW vs SW switches

- Soft switching based on general CPU
- Switching based on network processors
- Switching based on dedicated chips

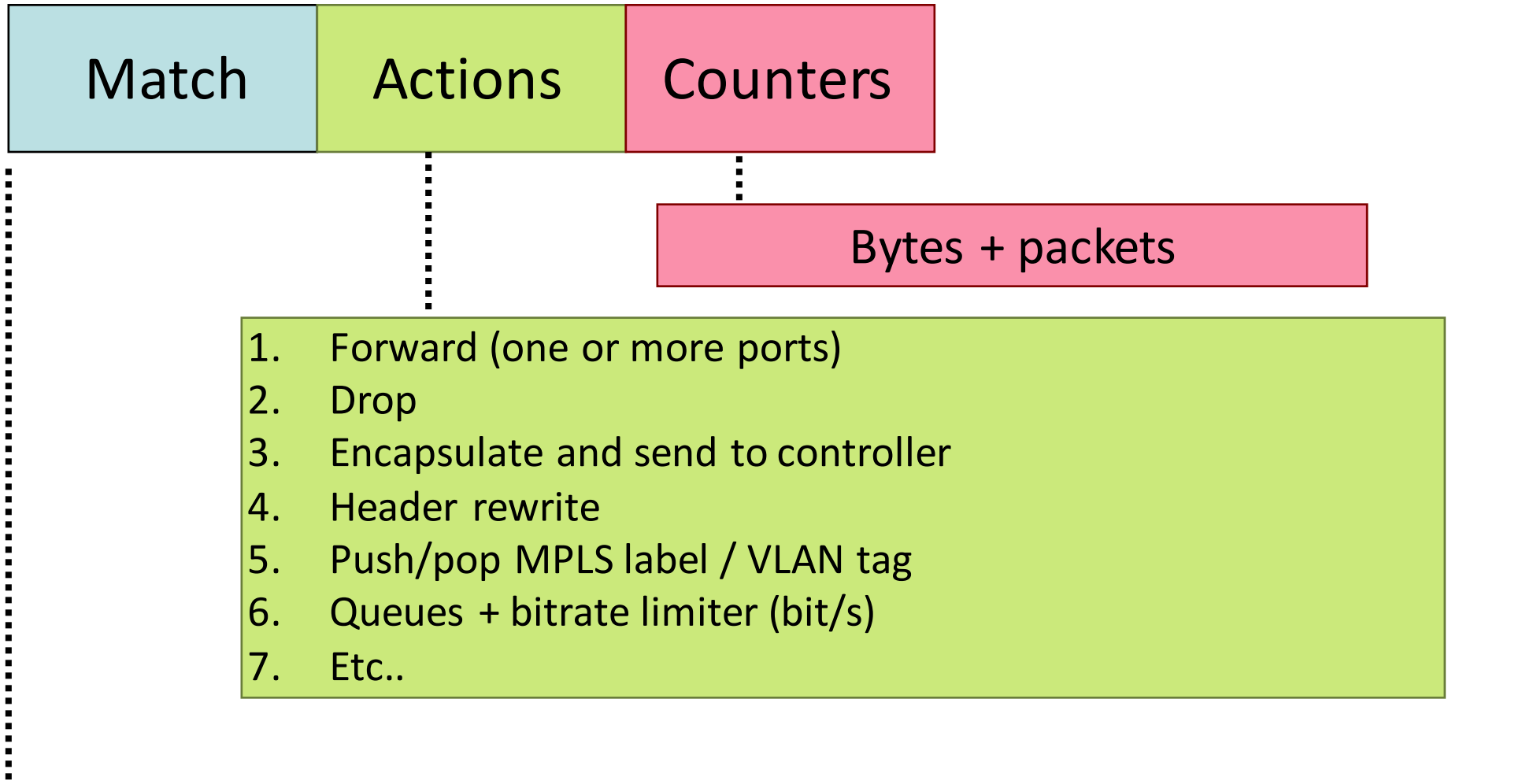


10x

10x



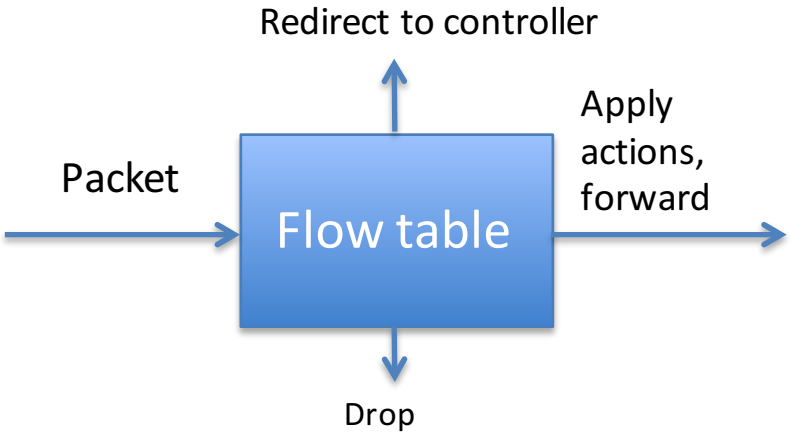
Data plane abstraction: Flow table

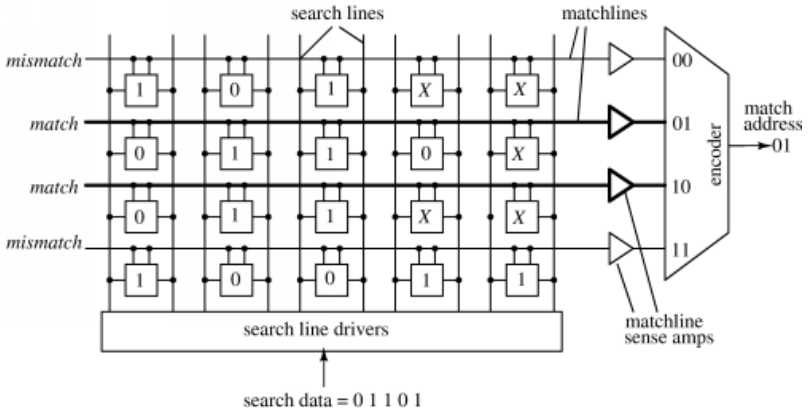


Slide courtesy: Rob Sherwood

Dataplane Abstraction

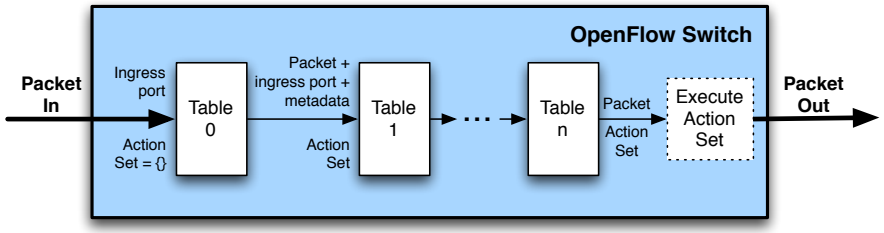
OF1.0: Single Match Table (SMT)

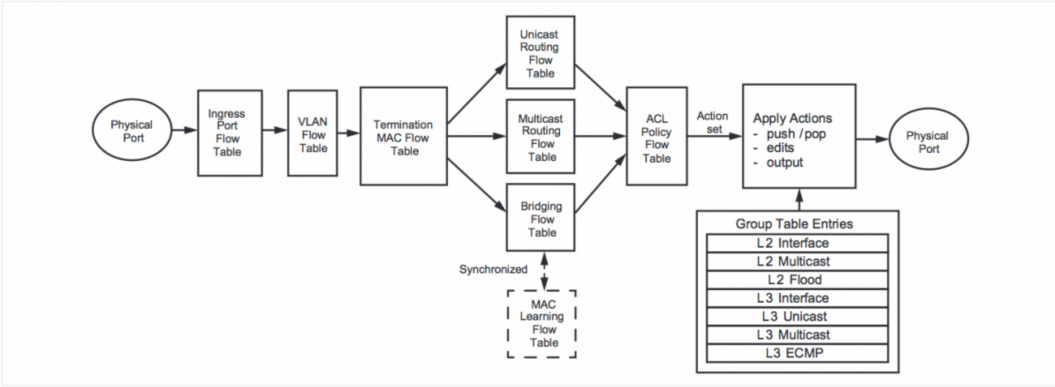




TCAM
Ternary Content Addressable Memory

OF1.1: Multiple Match Table (MMT)





Abstract Switch Pipeline for Bridging and Routing

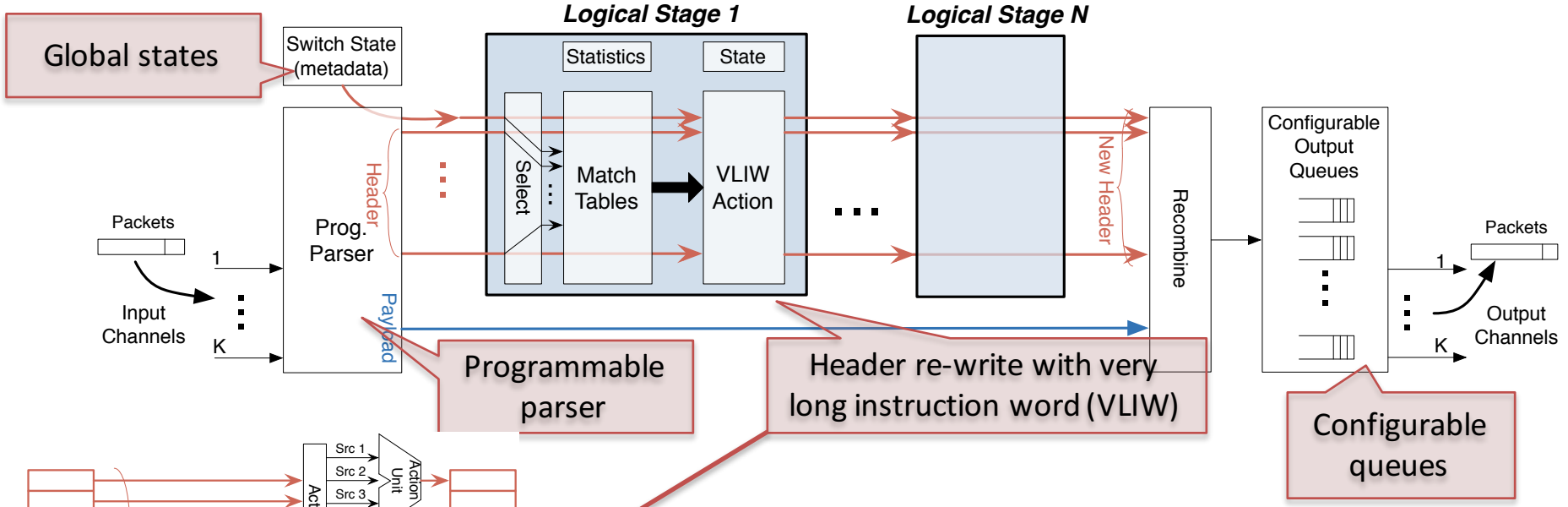
Switches cannot remain dumb: Data plane evolution

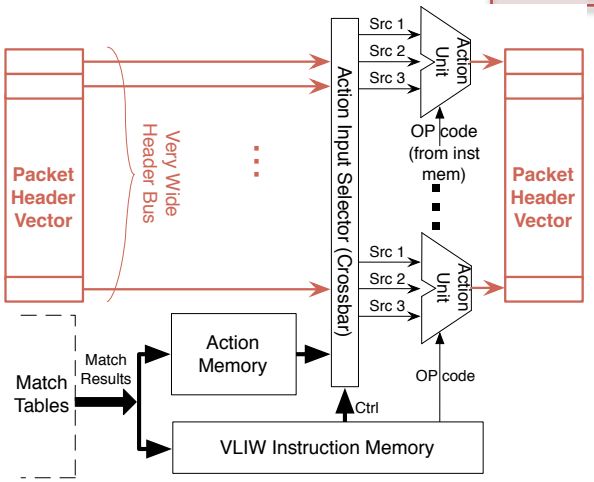
One man alone can be pretty **dumb** sometimes, but for real bona fide stupidity, there ain't nothin' can beat **teamwork**. [Edward Abbey]

Models can be perfect and clean, reality is dirty!

- The match/action model can ideally be used to program any network behavior and to get rid of protocol limitations at any level
- But unfortunately, with OF:
  - **Matches** can be done only on a set of predefined header fields (Ethernet, IPv4, MPLS, VLAN tag, etc.)
  - **Actions** are limited to a rather small set
  - **Header manipulation** (like adding label/tags, rewriting of fields, etc.) is limited to standard schemes
- As a result, **OF is not really protocol independent** and standards (including OF standards) are still necessary

Breaking the limitation of specialized pipelines





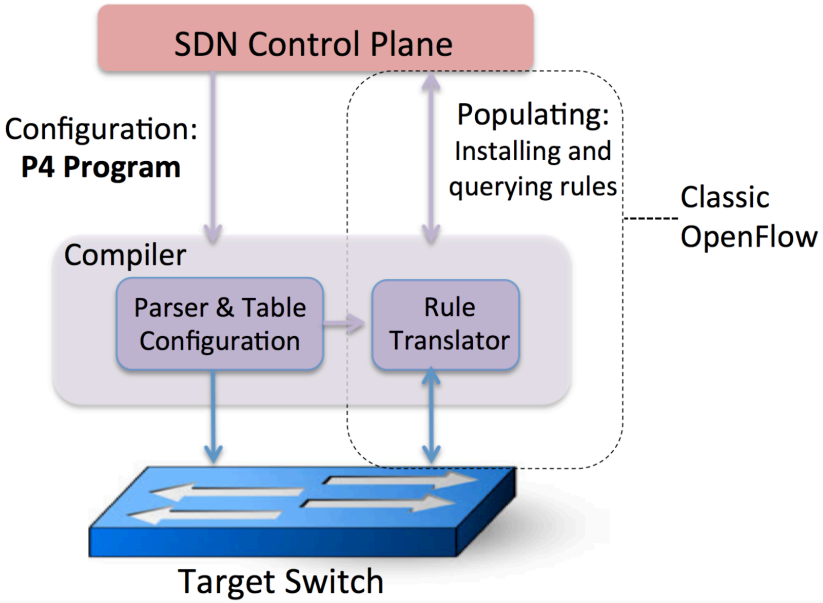
RMT - Reconfigurable Match Table

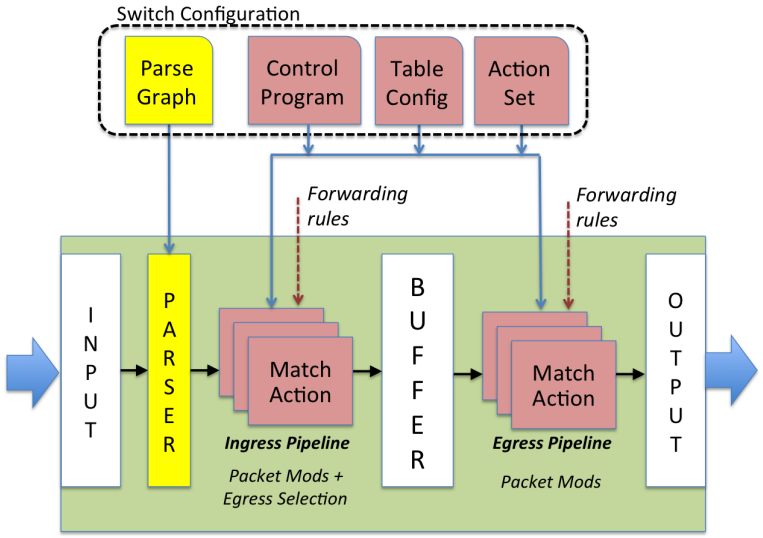
other examples: FlexPipe™ (Intel), …

[BOS13] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, ***"Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN",*** *in ACM SIGCOMM 2013.*

Target programmability: P4

- OpenFlow 2.0 proposal by McKeown, Rexford et al. [BOS14]
- 3 goals: **Protocol independence**, **Target independence**, **Reconfigurability**





other example: POF (Huawei)

*[BOS14] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker "**P4: Programming protocol-independent packet processors**," SIGCOMM CCR, July 2014*

Not too much not too little:
OpenState and stateful data planes

Too **clever** is **dumb**.
[Ogden Nash]

Looking for the “right” abstraction

- Programmability and real world viability
  - High levels of (deep) programmability in the data and control planes since ages
    - Active Networks
    - IETF ForCES
- Keywords for success:
  - Pragmatism
  - Compromise
  - “Right” mix of programmability: right **level of abstraction**
    - Many wonderful programmable platforms buried in the “lab world”

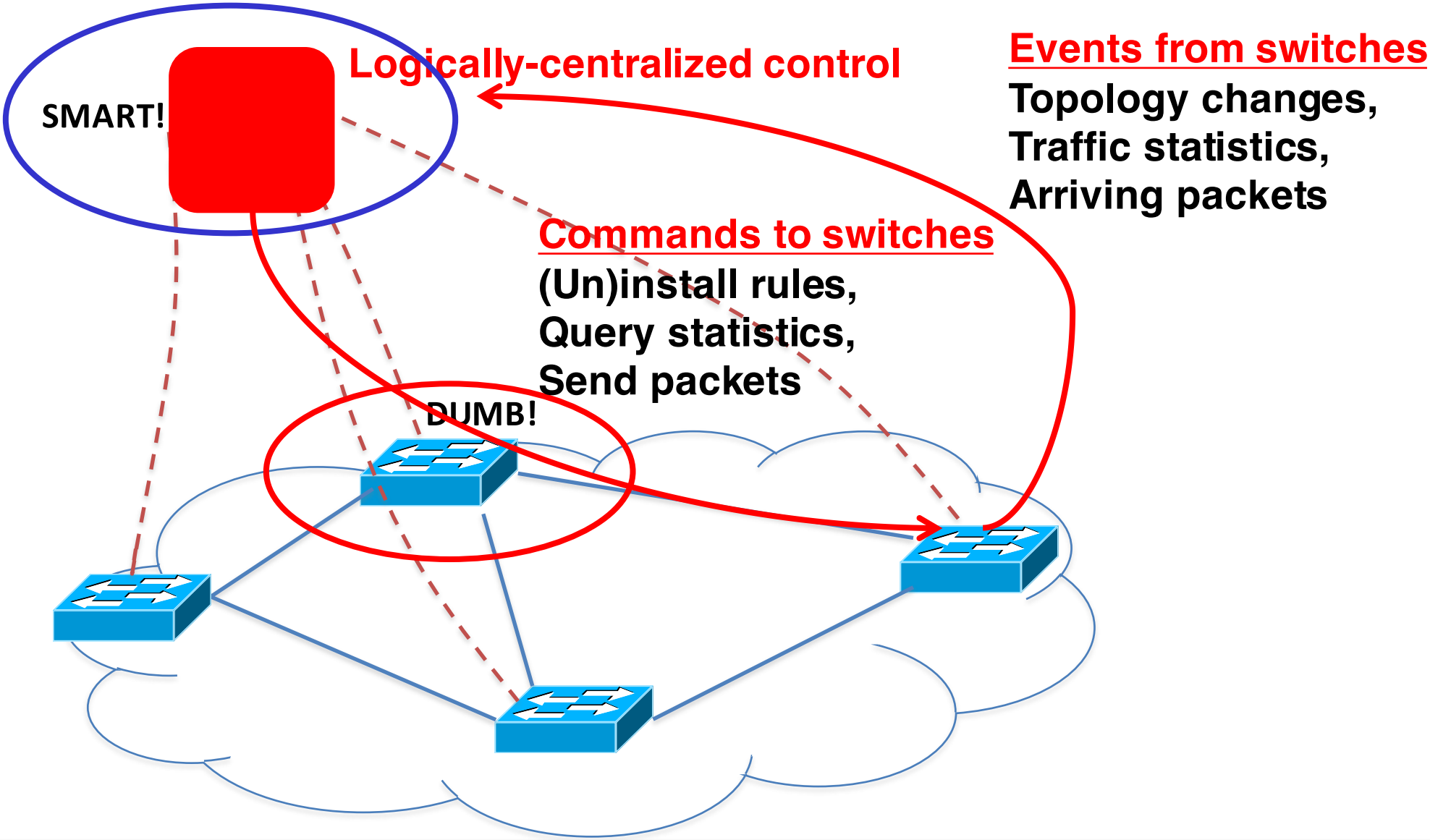
Remember: OF **meant to be** a compromise

[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…***

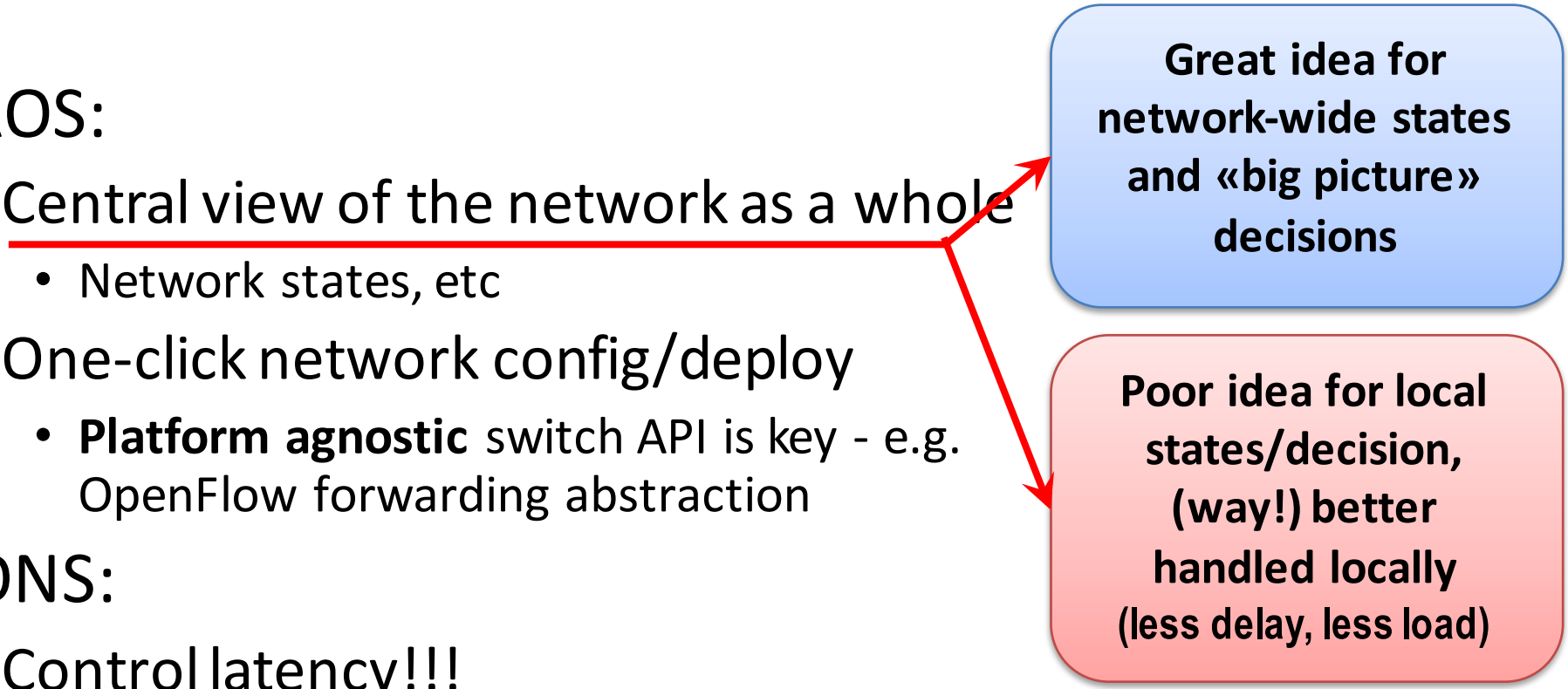
- **Viable approach:** "compromise on generality and seek a degree of switch flexibility that is
  - High performance and low cost
  - Capable of supporting a broad range of research
  - **Consistent with vendors' need for closed platforms.**

OF **forces** separation of data and control

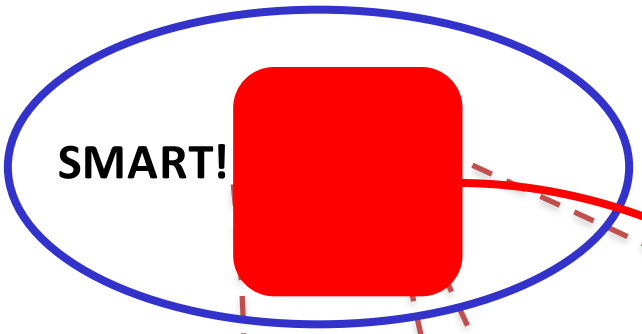


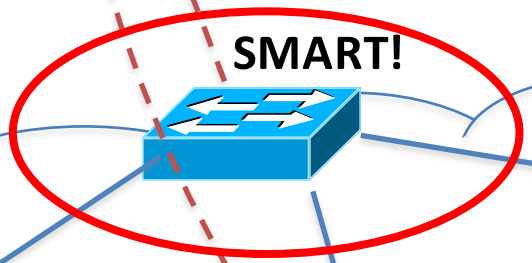
Centralized Control: pros and cons

- PROS:
  - Central view of the network as a whole
    - Network states, etc
  - One-click network config/deploy
    - **Platform agnostic** switch API is key - e.g. OpenFlow forwarding abstraction
- CONS:
  - Control latency!!!
    - O(second)
      1s = 300M packets lost @ 100 gbps
  - Signalling overhead



Our vision





Events from switches & central rule updates

Restricted to those of interest for GLOBAL decisions

Decision on how network should operate remains at the controller (SDN vision)

But "execution" of forwarding plane updates can be locally delegated

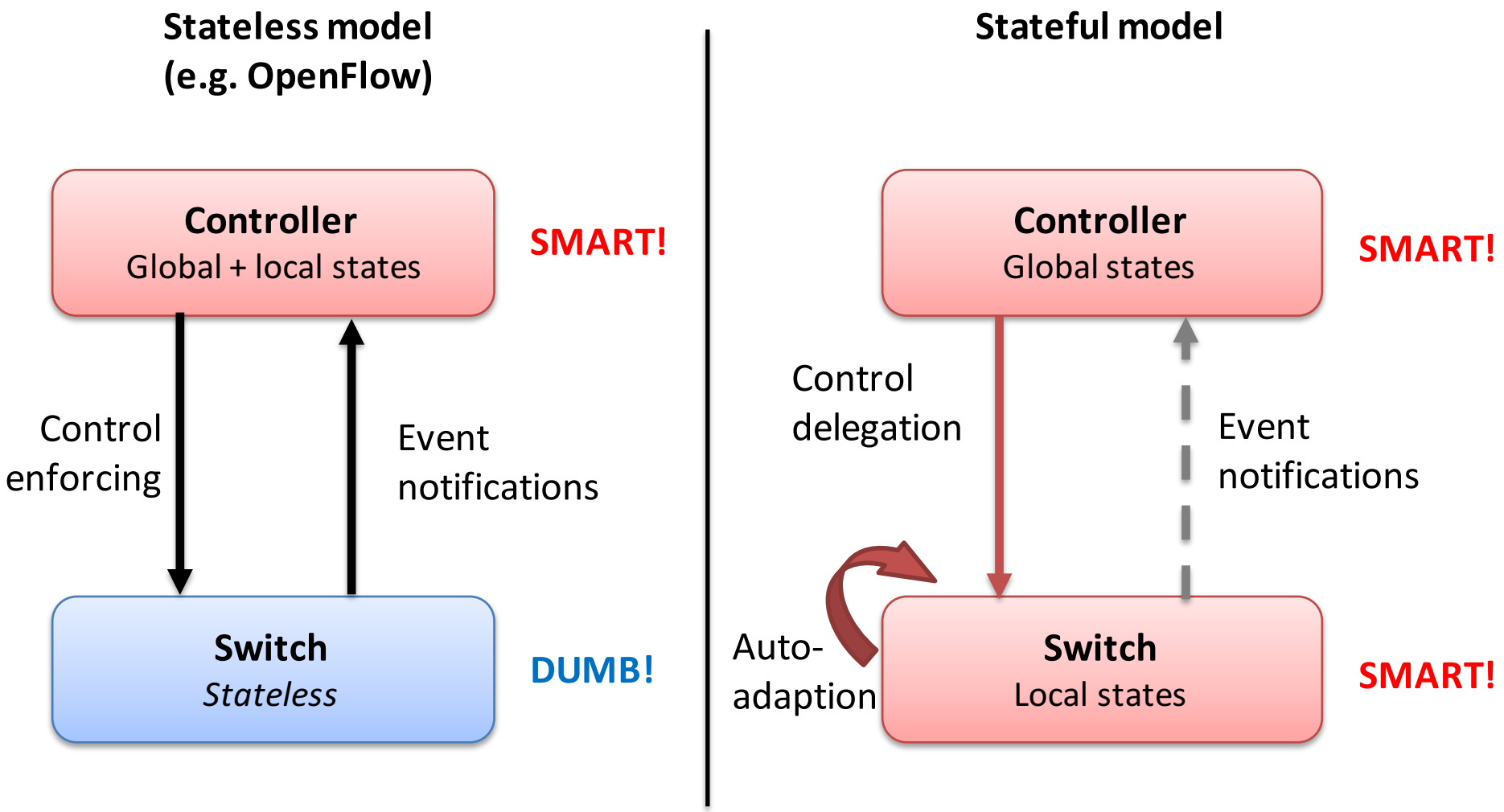
Inject "switch control programs"

Change forwarding behavior as specified by "this program" IF (local) events occur

Local processing: Ultra low Latency: o(nanosec) versus o(sec)

Local states: lower signalling

Stateless vs. Stateful data plane



Easier said than done

- We need a switch architecture and API which is…
  - **High performance**: state management tasks executed at wire-speed (packet-based events)
  - **Platform-independent**: consistent with vendors' needs for closed platforms
  - **Low cost and immediately viable**: based on commodity HW

Apparently, far beyond OpenFlow switches…

Our (perhaps surprising?) finding:
much closer to OF abstraction than expected!!

Our approach: **OpenState**

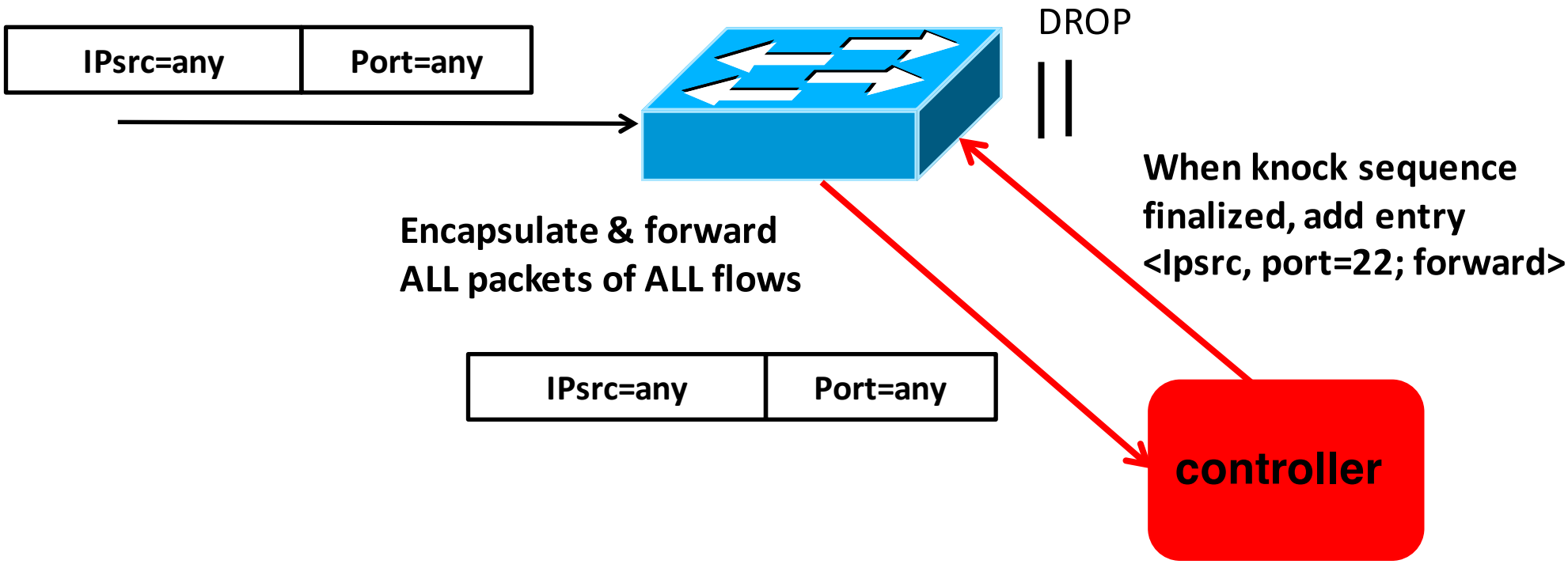
[CCR14] G. Bianchi, M. Bonola, A. Capone, C. Cascone, "**OpenState: programming platform-independent stateful OpenFlow applications inside the switch**", ACM Computer Communication Review, vol. 44, no. 2, April 2014.
[HPSR15] S. Pontarelli, M. Bonola, G. Bianchi, A. Capone, C. Cascone, "**Stateful Openflow: Hardware Proof of Concept**", IEEE HPSR 2015, Budapest, July 1-4, 2015.
[DRCN15] A. Capone, C. Cascone, A.Q.T. Nguyen, B. Sansò, "**Detour Planning for Fast and Reliable Failure Recovery in SDN with OpenState**", IEEE DRCN 2015, Kansas City, USA, March 24-27, 2015 – BEST PAPER RUNNER UP.
[ARX14] G. Bianchi, M. Bonola, A. Capone, C. Cascone, S. Pontarelli, "**Towards Wire-speed Platform-agnostic Control of OpenFlow Switches**", available on ArXiv, 2014.

Example: Port Knocking firewall

knock «code»: 5123, 6234, 7345, 8456

IPsrc=1.2.3.4	Port=5123	→ Drop(); 1.2.3.4 → 1° knock OK
IPsrc=1.2.3.4	Port=6234	→ Drop(); 1.2.3.4 → 2° knock OK
IPsrc=1.2.3.4	Port=7345	→ Drop(); 1.2.3.4 → 3° knock OK
IPsrc=1.2.3.4	Port=8456	→ Drop(); 1.2.3.4 → OPEN port SSH
IPsrc=1.2.3.4	Port=22	→ Forward()

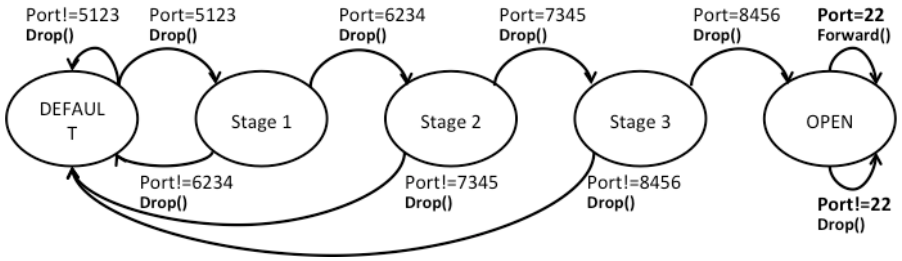
Port Knocking @ controller



Lots of overhead!!

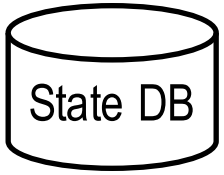
Can we move the state machine in the switch?

Maintain Knock state per flow



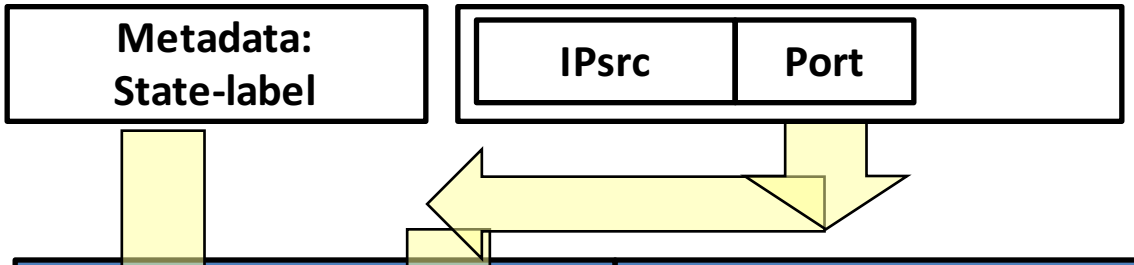
Can transform in a flow table? Yes:

Ipsrc: ??



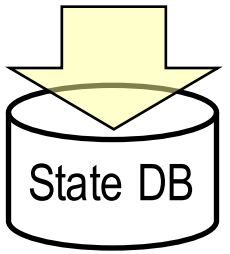
MATCH: <state, port> → ACTION: <drop/forward, state_transition>

Plus a state lookup/update

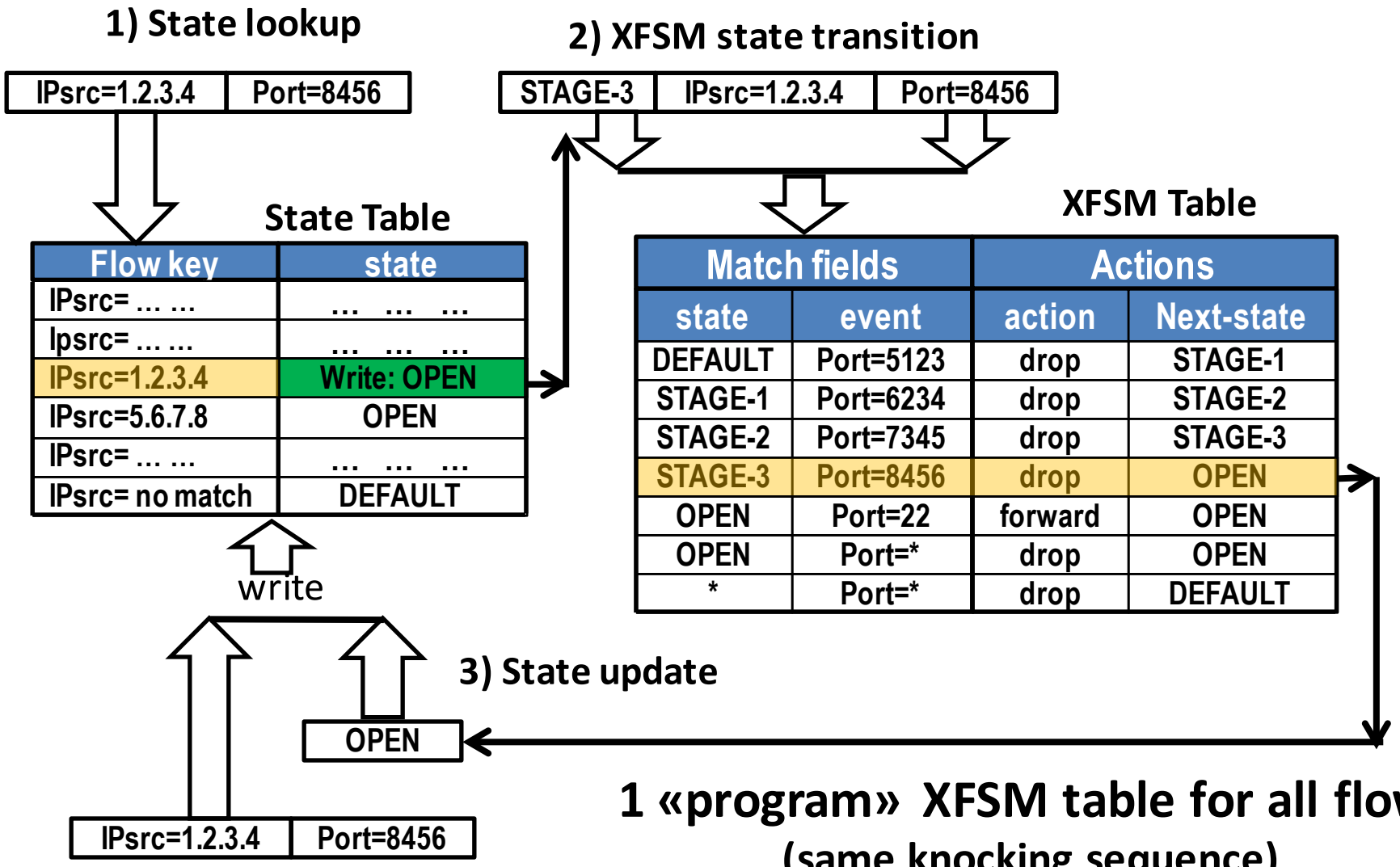


Match fields		Actions	
state	event	action	Next-state
DEFAULT	Port=5123	drop	STAGE-1
STAGE-1	Port=6234	drop	STAGE-2
STAGE-2	Port=7345	drop	STAGE-3
STAGE-3	Port=8456	drop	OPEN
OPEN	Port=22	forward	OPEN
OPEN	Port=*	drop	OPEN
*	Port=*	drop	DEFAULT

Ipsrc→OPEN



Putting all together



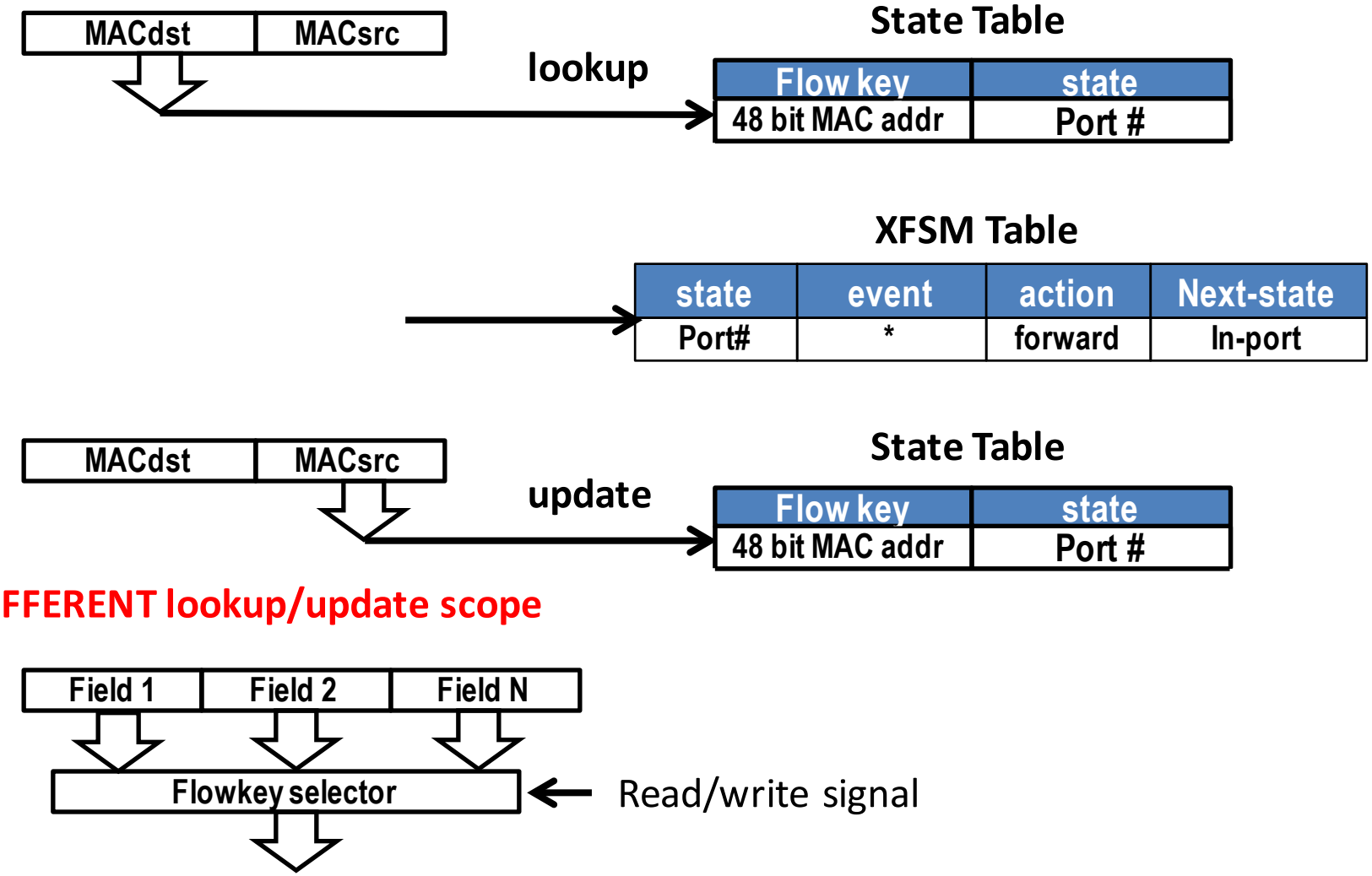
1 «program» XFSM table for all flows

(same knocking sequence)

N states, one per (active) flow

Cross-flow state handling

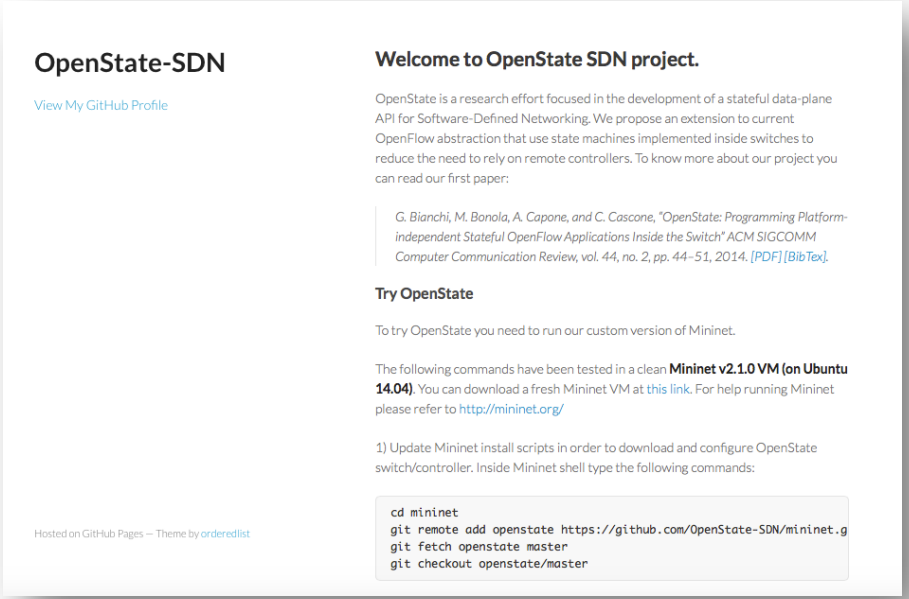
- Yes but what about MAC learning, multi-port protocols (e.g., FTP), bidirectional flow handling, etc?



Wrapping up

Proof of concept

- SW implementation:
  - Public domain: openstate-sdn.org
- HW implementation:
  - S. Pontarelli, M. Bonola, G. Bianchi, A. Capone, C. Cascone, "**Stateful Openflow: Hardware Proof of Concept**", IEEE HPSR 2015, Budapest, July 1-4, 2015.



Switch: **ofsoftswitch13**; Controller: **Ryu**

State Machines

- Events: matches, times, all-flow states, meters, …
- Machine instantiations: One table/machine per set of flows, parallel machines for different set of flows, multiple stateful stages

Applied smartness:

How to incorporate statefulness in languages?

There are science and the **applications** of science, bound together as the fruit of the tree which bears it. [Louis Pasteur]

Stateful applications

DONE:

- Port Knocking
- MAC learning
- Label/address advertisement learning
- Reverse Path Forwarding
- Flow-consistent Load Balancing
- Failure recovery
- DDoS multi-stage flow marking
- …

All (!) otherwise not possible without explicit controller's involvement

CHALLENGE:

- **IDS/DPI**
- **TCP flow processing**
- **Monitoring**
- **…**

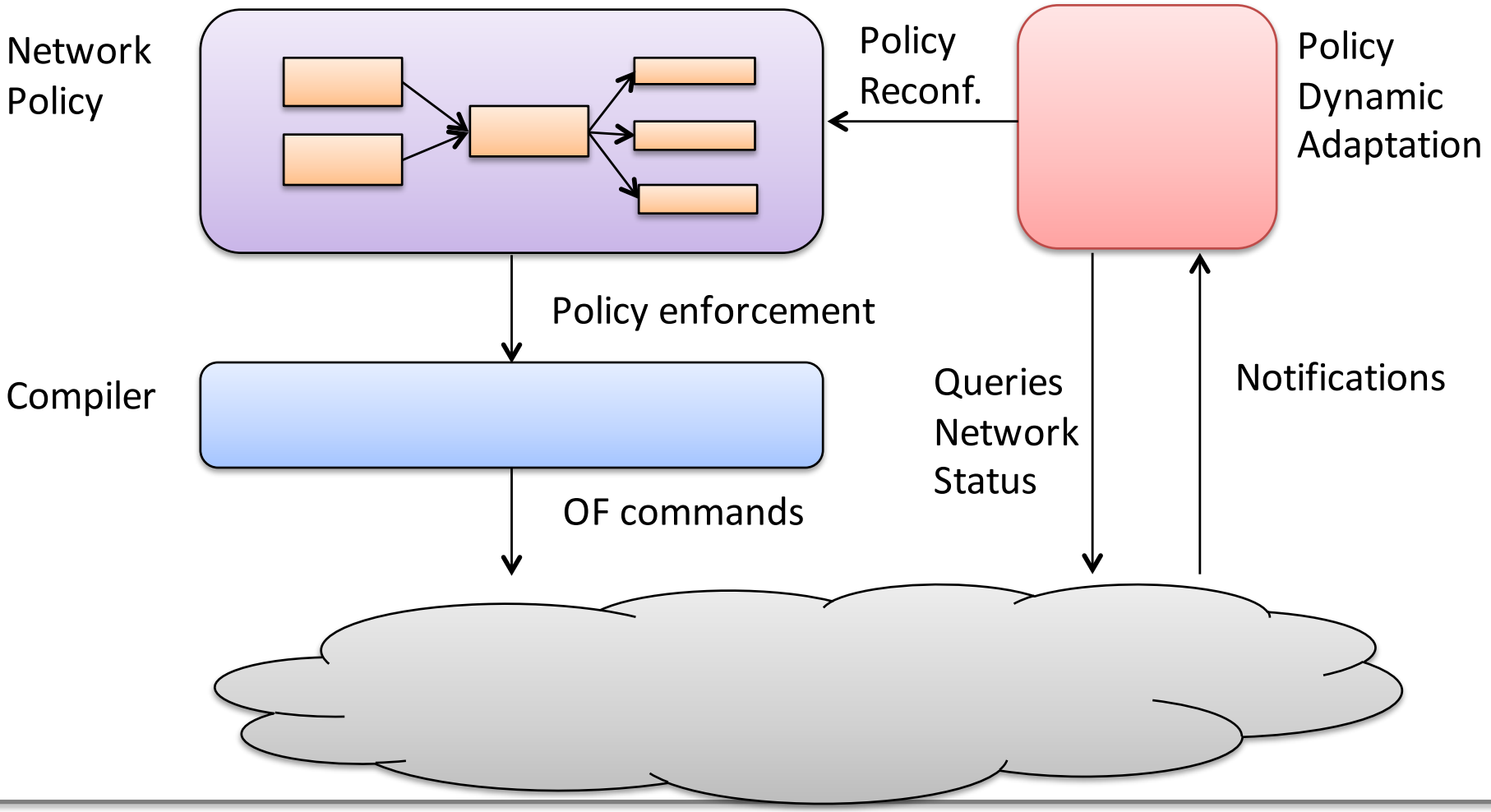
Need new «actions»

Need extra logic (full XFSM)

Custom implementation in the controller using the OpenState extension of OF1.3

SDN languages for applications

- Example: Frenetic/Pyretic



Open question

- **How can we use a stateful dataplane in programming languages for SDN?**

- **Easy solutions:** create a library of switch state machines to be used as functions in the language
- Can we automatically discover local state machines in a network policy and instantiate them (compiler)?
- Is it a parallelization problem of a global state machine with local and global states and events?
- How much of this parallelization problem should be exposed to the programmer?

Thanks

Antonio Capone

Email: *antonio.capone@polimi.it*

OpenState: openstate-sdn.org

EU project BEBA – http://www.beba-project.eu

Extended slide-set is available on OpenState web site!