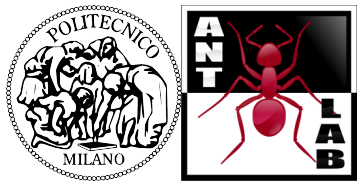
Beba
BEhavioural BAsed forwarding

Behavioral-based forwarding: Towards Wire-speed Platform-agnostic Control of OpenFlow Switches

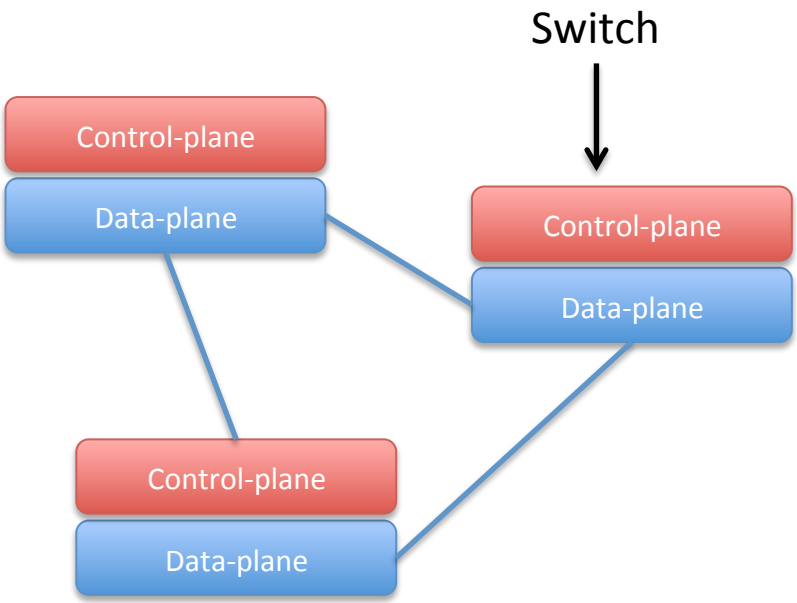
G. Bianchi, M. Bonola, S.Pontarelli
University of Rome "Tor Vergata"



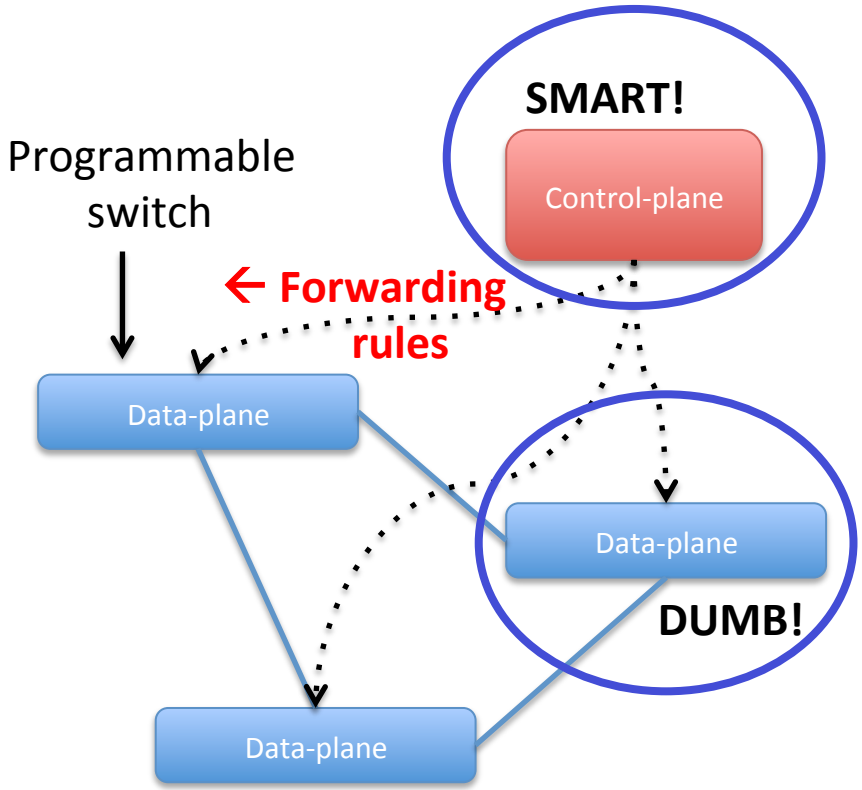
A. Capone, C. Cascone
Politecnico di Milano - ANTLab

SDN, current view

Traditional networking

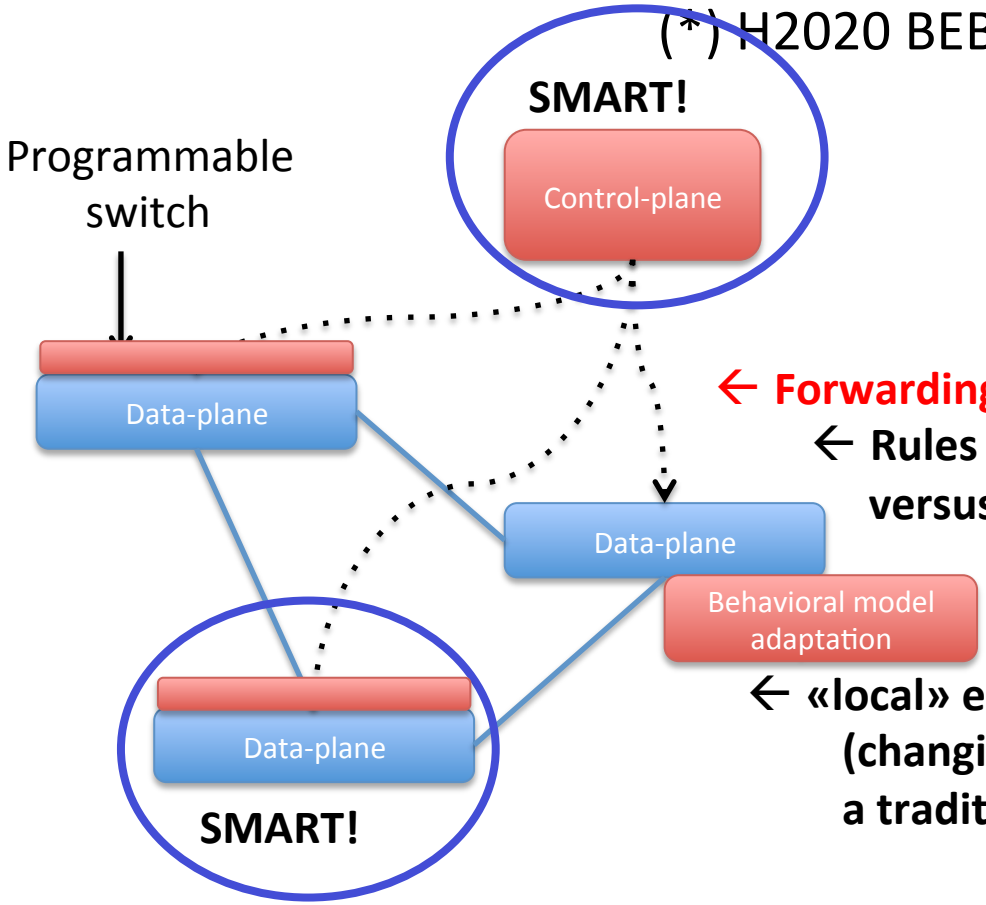


Software-Defined Networking



SDN, our(*) view

(*) H2020 BEBA project



← **Forwarding «behavioral models»** =

← **Rules + how to EVOLVE rules versus packet-level «events»**

← **«local» execution of control tasks (changing forwarding table «is» a traditional control task)**

Smart switches → can dynamically update flow tables

Central control → updates decided by «behavioral models»

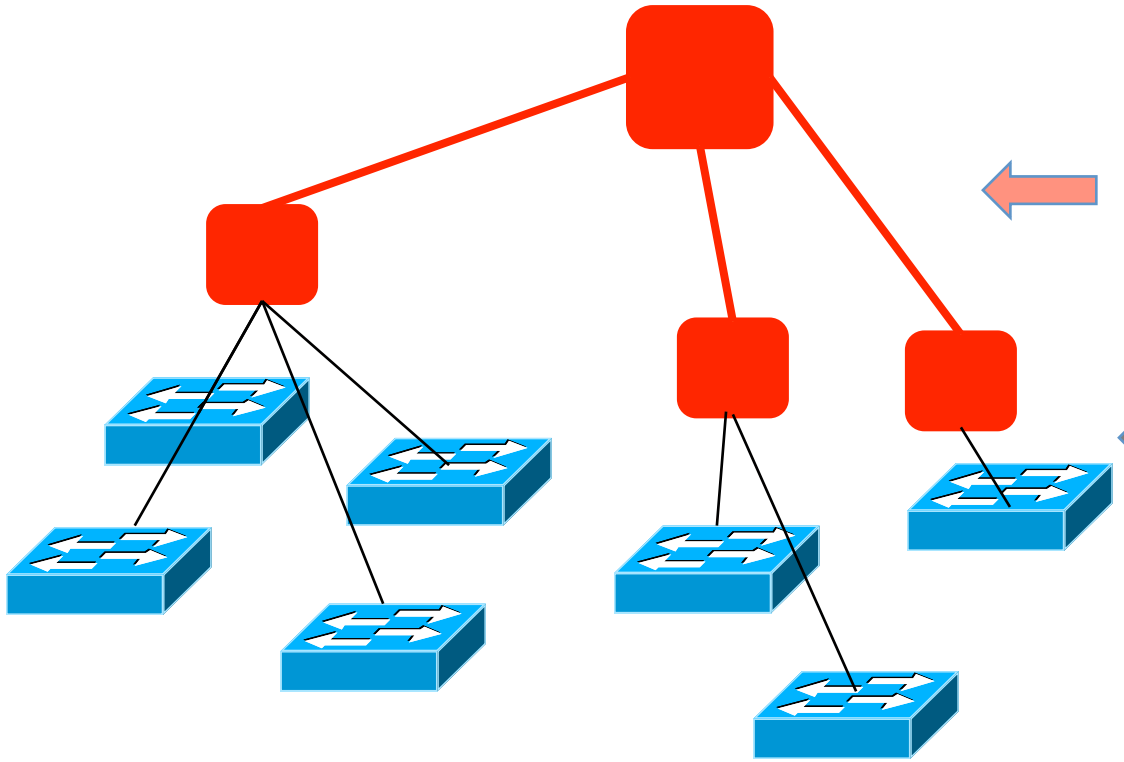
Many use cases

- **Consistent load balancing:**
  - If TCP SYN (new flow), set forwarding path; otherwise route via previously used port
- **Reverse Path forwarding, (MAC) Learning, …**
  - set return path based on input packet fields
  - set forwarding table based on input port / input label
- **DDoS mitigation, Security**
  - Traffic gets «heavy» → reroute it
  - Flow exhibits strange features → reroute to packet scrubber
- **Resiliance**
  - Reruote upon link failure/topology changes (see next pres)
- **Many more…**

Whh should I care? SDN controllers can do all this today!
What about latency? Signalling? Controllers' availability?

Distributed controllers

the «common» way to address Latency, Signalling and Controllers' availability cons



Proprietary controller extensions?
Back to Babel?

A non-solution!
still slow path latency!!

«true» fast path solution: update forwarding rules in 1 packet time – 3 ns @ 40B x 100 Gbps

3 ns = 60cm signal propagation…

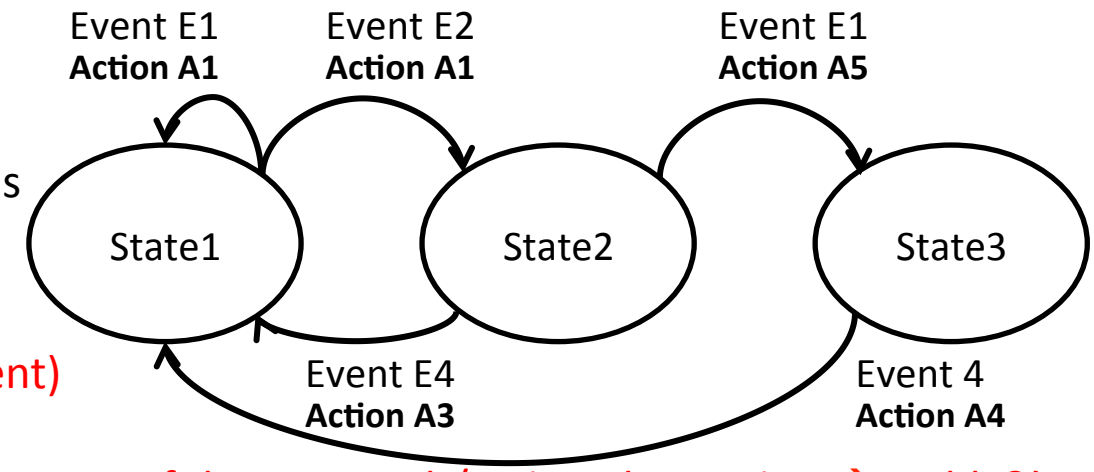
Make or break issue: which **pragmatic abstraction** for «behavioral forwarding»?

- **Openflow: the right balance between vendors' needs, viability, and flexibility**
  - openFlow Match/action abstraction:
    - High performance (@ wire speed)
    - Low cost, **relying on already available HW**
    - Capable of supporting a broad range of innovation
    - Consistent with **vendors' need for closed platforms**. (!!!)
- **Behavioral forwarding** = rules + how forwarding rules shall change based on packet level events
    - Which pragmatic balance? Which Platform independent model? How to leverage existing fast path HW?
    - Reliance on *«slow path» (CPU) processing NOT an option!!*

Our behavioral abstraction (1):

Mealy Machines + R/W state decoupling

Mealy Machine: 4-tuple
{States}
{Events}
{Actions}
{Tx} = States x Events → States x Actions



Match → Action
(state dependent) (state dependent)

Rings any bell? Looks like a «trivial» extension of the OF match/action abstraction → viable?!

Finding 1: Any **control program** that can be described by a Mealy (Finite State) Machine is already (!) basically compliant with OF1.3 HW

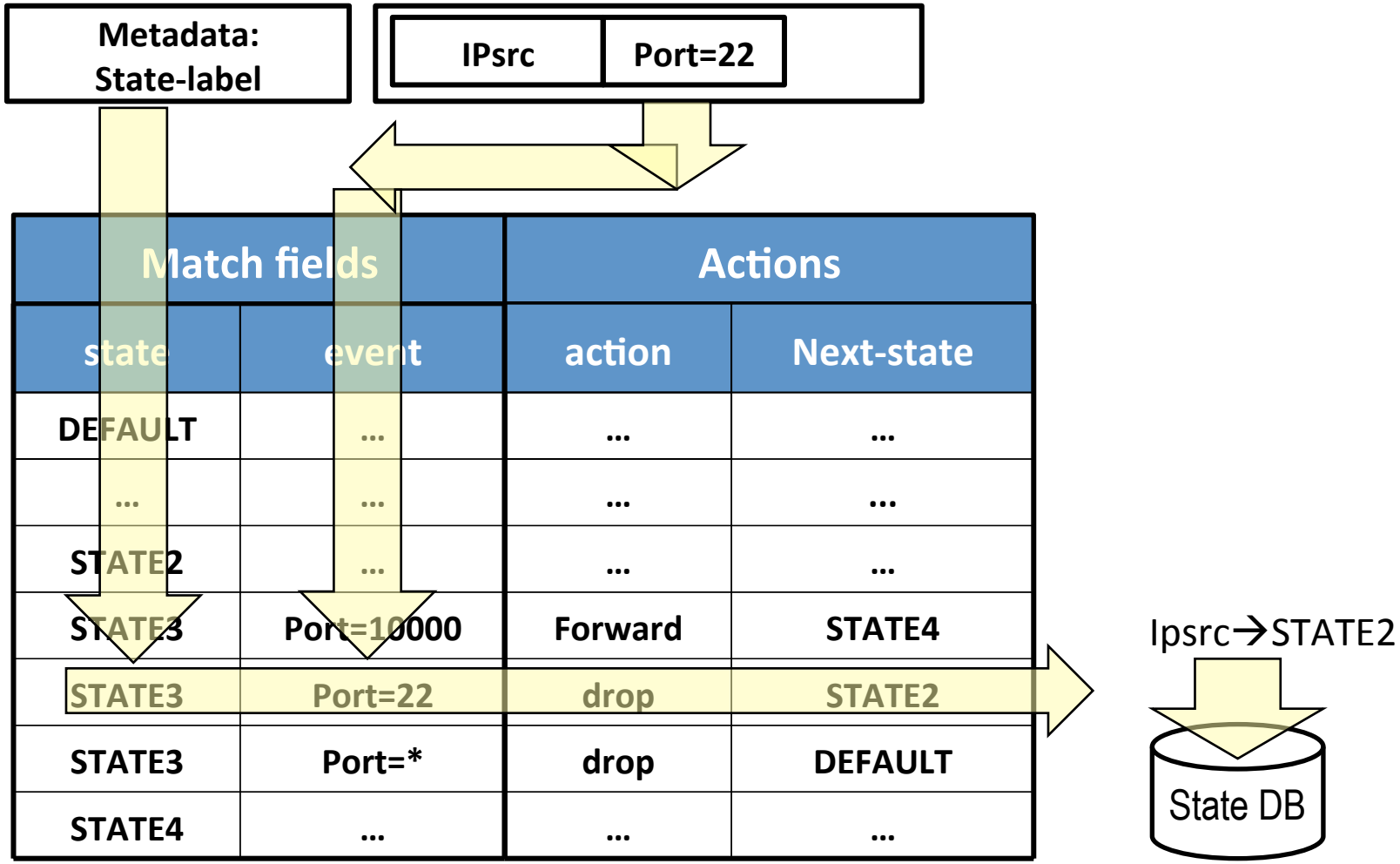
MM can be cast into ordinary (!) OF table

Ipsrc: ??

State DB

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

Plus a state lookup/update



Example: port knocking

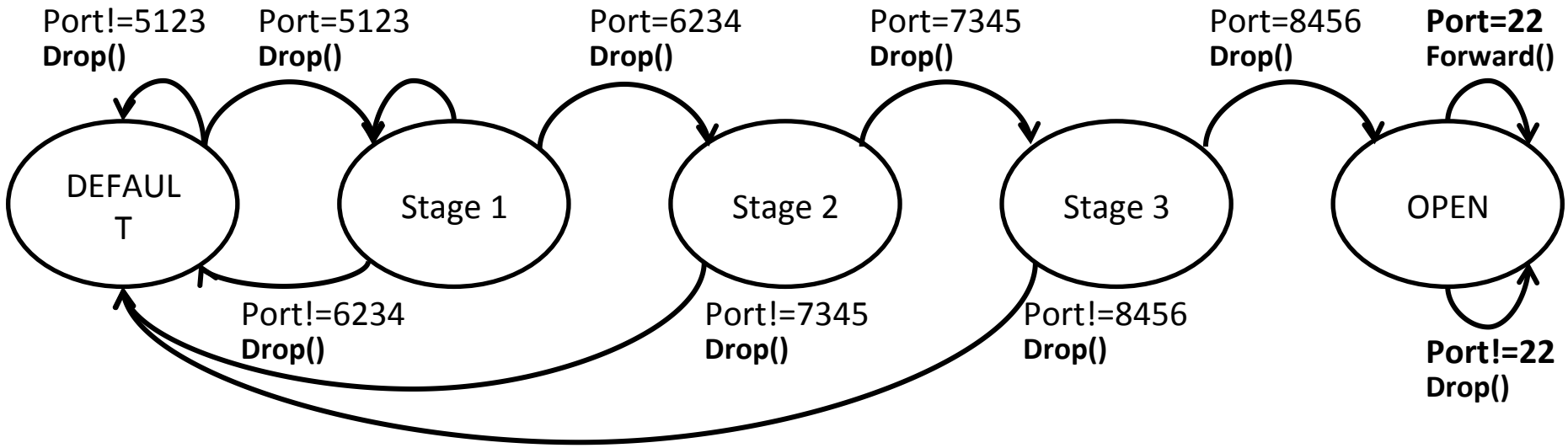
Mealy Machine

{States} = {DEF, S1,S2, S3, OPEN} ← **specified by programmer!!**

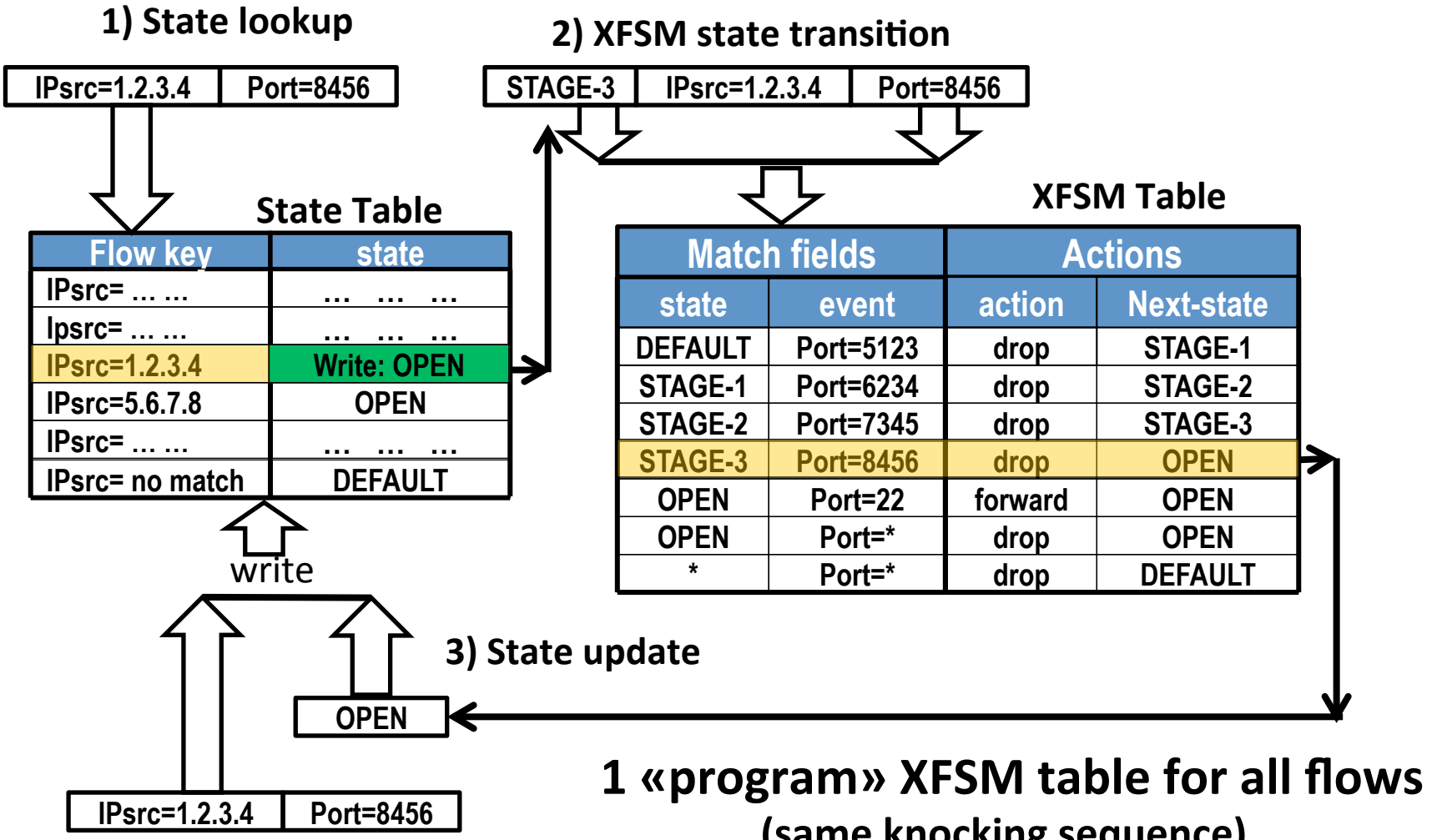
{Events} = match on port field ← **standard OF Match (subset)**

{Actions} = {Drop(), Forward()} ← **standard OF actions (subset)**

{Tx} = States x Events → States x Actions ← specified by programmer, see diagram for knock sequence 5123, 6234, 7345, 8456



Example: port knocking



1 «program» XFSM table for all flows

(same knocking sequence)

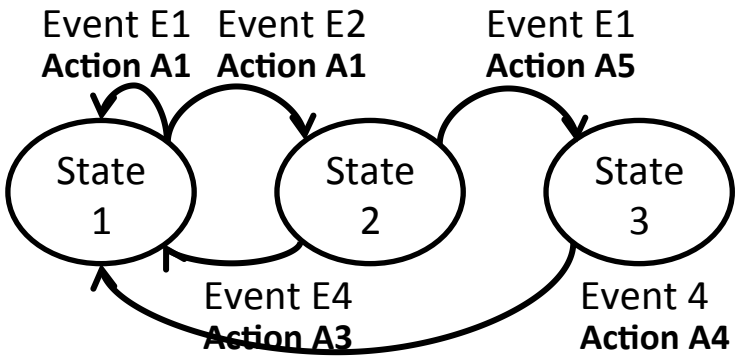
N states, one per (active) flow

Our behavioral abstraction (2):

Mealy Machines + R/W state decoupling

State Table

Flow key	state
IPsrc= … …	… … …
Ipsrc= … …	… … …
IPsrc=1.2.3.4	STATE1
IPsrc=5.6.7.8	STATE2
IPsrc= … …	… … …
IPsrc= no match	DEFAULT = S1



«state» table semantically different from match table (though both can be OF tables)

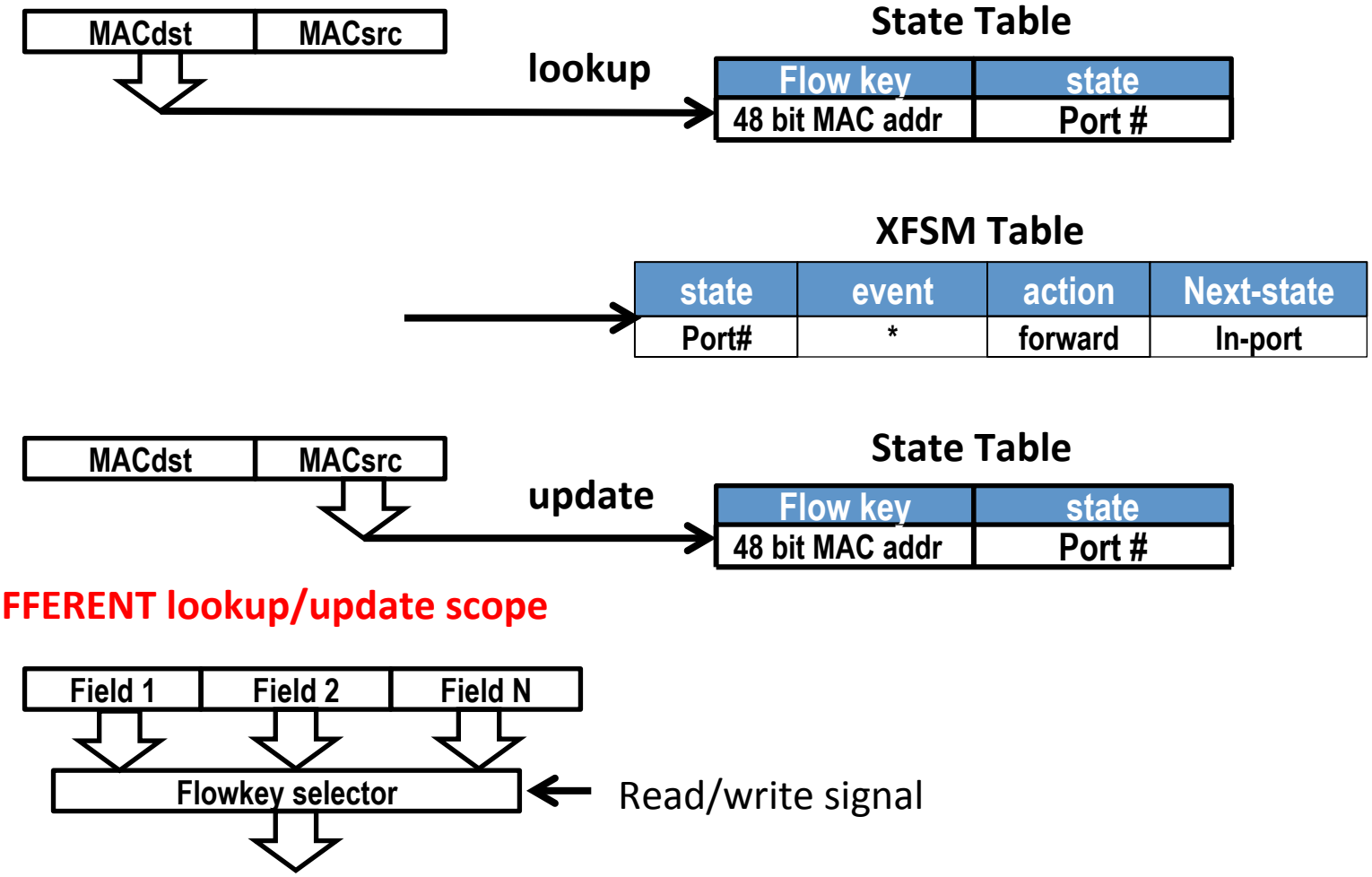
- Retrieves state associated to given incoming flow
- Stores (rewrites) state after eventual transition

Key idea: decouple flowkey used to «read» from flowkey used to «write»

e.g. MAC learning: read MAC destination (for forwarding), write MAC source (for learning)

Finding 2: decoupled R/W requires minimal HW extensions to OF1.1+ (trivial combinatorial circuit)

Example: MAC learning



Proof of concept

- SW implementation:
  - Trivial modifications to SoftSwitch
  - Ryu controller
  - Public domain
- HW implementation:
  - 5 clock (2 SRAM read + 2 TCAM + 1 SRAM write)
  - 10 Gbps just requires 156 MHz clock TCAM, trivial
  - Optimization in progress (pipelining) for 100 Gbps.

Aftermath: almost straightforward OF extension!

Taking stocks

- In-switch control **programs**
  - State machines
  - programmed outside and injected by controller
    - (we centrally decide behavior, switch locally executes desired behavior)

- Which CPU «executes» such «programs»?
  - The TCAM match…. **the TCAM is the processor**!!
    - Natively fast path

- So far, Mealy machines (actions + events + states + tx)
  - Valuable as fully compatible with OpenFlow
    - Practical, viable, immediately exploitable?
    - may impact ONF standard?

Looking further

- Mealy machines limited to actions + events + states +transitions

- What if we develop a computational environment involving full XFSM
  - Custom registries
    - E.g. RAM added to state table entries
    - Already in OF, but very specific and hardwired (e.g. flow stats)
  - Global state variables and event extensions
    - switch resource status (port, queues, etc)
    - Timer expiration handlers
  - «register update functions»
    - simple arithmetic operations' support
  - Triggering «conditions»
    - Already in OF, but very specific and hardwired (e.g. flow stats)

- From behavioral forwarding to **«full» flow computing model**??
  - Challenging, but we believe feasible
    - In part already hardcoded in OF extensions, just make it more general
  - let's see next year's presentation ☺