Comparing Open vSwitch (OpenFlow) and P4 Dataplanes for Agilio™ SmartNICs

Johann Tönsing

May 24, 2016

Agenda

NETRONOME

- Contributions of OpenFlow, Open vSwitch and P4
- OpenFlow features missing in P4, and P4 features missing in OpenFlow
- Architecture of Open vSwitch accelerated by a SmartNIC
- Architecture of a P4 SmartNIC datapath
- Performance considerations
- Next steps: proposals towards convergence and alignment

Contributions of OpenFlow

NETRONOME

Architectural Principle:

- Logically Centralized, Decoupled Control

Model:

- Defined by specification: dataplane protocols, pipeline of match-action tables

(Soon: dataplane protocols no longer in main specification, dynamic dataplane protocol definition being considered)

Interface:

- Protocol (handcrafted, binary C struct style)

Contributions of OpenFlow

NETRONOME

Architectural Principle:

- Logically Centralized, Decoupled Control

Well known SDN (and older) principle -
not unique to OpenFlow

No plans to abandon —
perhaps add selective delegation, hierarchies

Model:

- Defined by specification: dataplane protocols, pipeline of match-action tables

(Soon: dataplane protocols no longer in main specification, dynamic dataplane protocol definition being considered)

Interface:

- Protocol (handcrafted, binary C struct style)

Contributions of OpenFlow

NETRONOME

Architectural Principle:

- Logically Centralized, Decoupled Control

Well known SDN (and older) principle -
not unique to OpenFlow

No plans to abandon —
perhaps add selective delegation, hierarchies

Model:

- Defined by specification: dataplane protocols, pipeline of match-action tables

(Soon: dataplane protocols no longer in main specification, dynamic dataplane protocol definition being considered)

Too general: Table Typing Patterns define what is required by application, supported by implementation

Too limited: annoying to have to create specification revisions to add protocols, behavior

Interface:

- Protocol (handcrafted, binary C struct style)

Contributions of OpenFlow

NETRONOME

Architectural Principle:

- Logically Centralized, Decoupled Control

Well known SDN (and older) principle -
not unique to OpenFlow

No plans to abandon —
perhaps add selective delegation, hierarchies

Model:

- Defined by specification: dataplane protocols, pipeline of match-action tables

(Soon: dataplane protocols no longer in main specification, dynamic dataplane protocol definition being considered)

Too general: Table Typing Patterns define what is required by application, supported by implementation

Too limited: annoying to have to create specification revisions to add protocols, behavior

Interface:

- Protocol (handcrafted, binary C struct style)

Software usually built using libraries with callable APIs

Protocol can be a language independent interface (but prefer automated marshalling, e.g. Thrift)

Need faster interface mechanism (e.g. callable API) for datapath extension code

Contributions of P4

NETRONOME

Architectural Principle:

- Language enabling datapath programming
- Deployable with central or distributed control, as switch or part of switch

Model:

- Datapath protocols and behavior are defined by a program (not enshrined in a standard)

Interface:

- Evolving - currently callable API is *generated*

Contributions of P4

NETRONOME

Architectural Principle:

- Language enabling datapath programming
- Deployable with central or distributed control, as switch or part of switch

Model:

- Datapath protocols and behavior are defined by a program (not enshrined in a standard)

Interface:

- Evolving - currently callable API is *generated*

Can be used with logically centralized decoupled control

Interoperability may be limited if embedded in proprietary datapaths

Contributions of P4

NETRONOME

Architectural Principle:

- Language enabling datapath programming
- Deployable with central or distributed control, as switch or part of switch

Can be used with logically centralized decoupled control

Interoperability may be limited if embedded in proprietary datapaths

Model:

- Datapath protocols and behavior are defined by a program (not enshrined in a standard)

Useful to enable protocols and behavior to rapidly evolve: killer app = flexibility

Useful as a unambiguous model of programmable and fixed datapath behavior

Interface:

- Evolving - currently callable API is *generated*

Contributions of P4

NETRONOME

Architectural Principle:

- Language enabling datapath programming
- Deployable with central or distributed control, as switch or part of switch

Can be used with logically centralized decoupled control

Interoperability may be limited if embedded in proprietary datapaths

Model:

- Datapath protocols and behavior are defined by a program (not enshrined in a standard)

Useful to enable protocols and behavior to rapidly evolve: killer app = flexibility

Useful as a unambiguous model of programmable and fixed datapath behavior

Interface:

- Evolving - currently callable API is *generated*

Convenient for initial experimentation

However complicates matters for controllers - need to link with different library for each version of each program

Consider generic interface instead

Contributions of Open vSwitch Evaluated

NETRONOME

Widely deployed virtual switch

- Integrated with many controllers, cloud management systems (OpenStack, ODL, ONOS…)

Accepted into Linux kernel

- Uses Linux kernel facilities where appropriate, e.g. for connection tracking or QoS

Developed by community as open source project

- Frequently leading OpenFlow standards

Contributions of Open vSwitch Evaluated

NETRONOME

Widely deployed virtual switch

- Integrated with many controllers, cloud management systems (OpenStack, ODL, ONOS…)

Great!

Accepted into Linux kernel

- Uses Linux kernel facilities where appropriate, e.g. for connection tracking or QoS

Developed by community as open source project

- Frequently leading OpenFlow standards

Contributions of Open vSwitch Evaluated

NETRONOME

Widely deployed virtual switch

- Integrated with many controllers, cloud management systems (OpenStack, ODL, ONOS…)

Great!

Accepted into Linux kernel

- Uses Linux kernel facilities where appropriate, e.g. for connection tracking or QoS

Using Linux specific features is problematic for interoperability - e.g. with hardware switches, or DPDK version of OVS

Developed by community as open source project

- Frequently leading OpenFlow standards

Contributions of Open vSwitch Evaluated

NETRONOME

Widely deployed virtual switch

- Integrated with many controllers, cloud management systems (OpenStack, ODL, ONOS…)

Great!

Accepted into Linux kernel

- Uses Linux kernel facilities where appropriate, e.g. for connection tracking or QoS

Using Linux specific features is problematic for interoperability - e.g. with hardware switches, or DPDK version of OVS

Developed by community as open source project

- Frequently leading OpenFlow standards

Great!

OpenFlow Features not in P4

NETRONOME

- At any time, the SDN controller can send a flow add or flow modify message:
  - … creating a new table or a new control flow path,
  - … matching an additional field or invoking an additional action, etc.

- In P4, actions, tables, and control flow options are pre-declared in the program

- Solutions to retain dynamic functionality with OpenFlow compatibility:
  - JIT compile program variant with a capability once it is first required [analogous to PyPy]
  - Cache these variants - remember last used (issue: when to “garbage collect”)

- Solutions to avoid need for dynamic functionality
  - Model (e.g. TTP) is used to declare required functionality (e.g. action list patterns) at start time
  - No changes needed to OpenFlow otehrwise (device returns error if functionality unsupported)

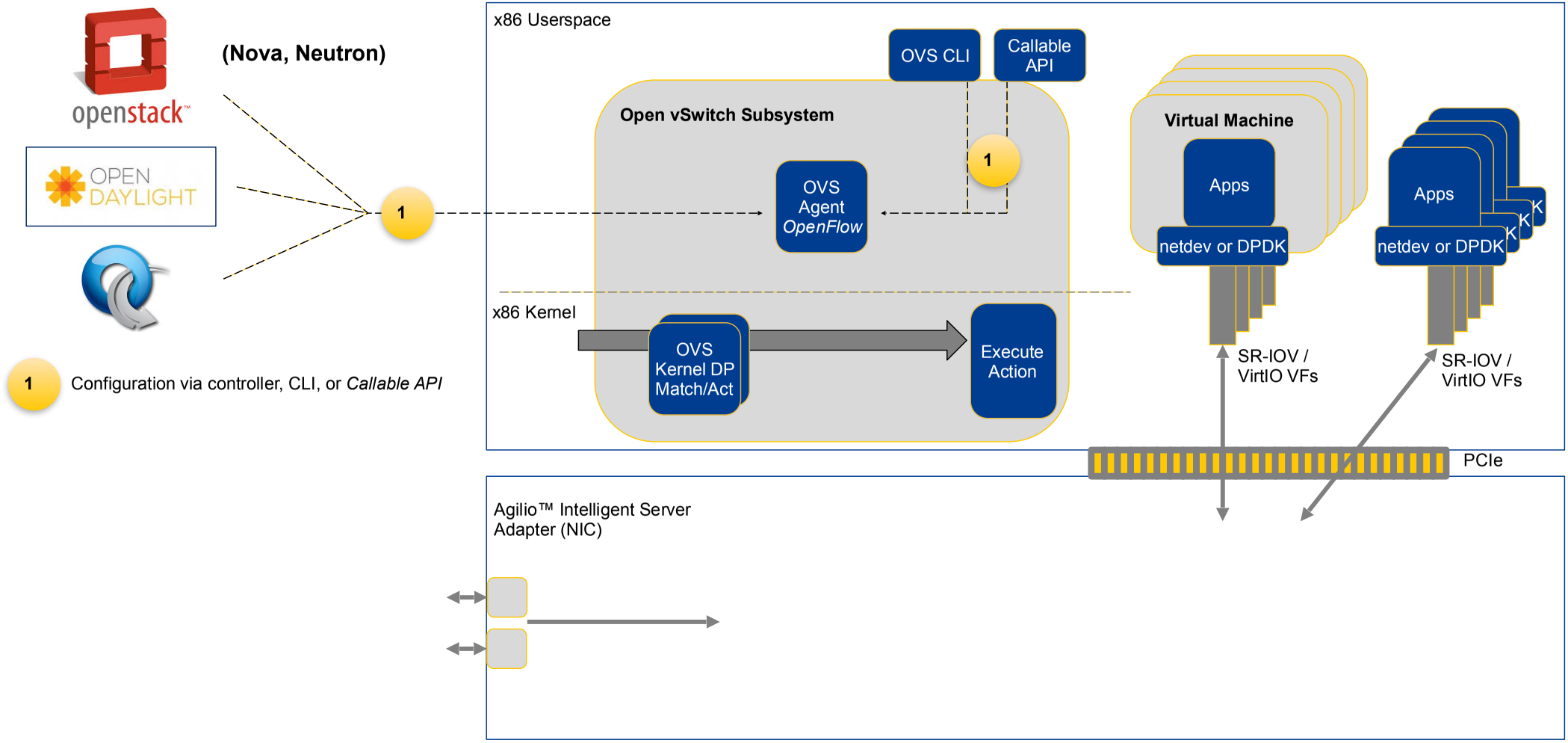
P4 Features not in OpenFlow

NETRONOME

- Too many to list here! Nice!
- Flexibility can actually be a problem for controllers, cloud management systems: difficult to generically support all datapath programs / program versions
  - Solution: “plugin” into controller to go along with datapath program
    - Run time interface becomes distributed system infrastructure (RPC)
  - Solution: implement functionality with a well defined (less rapidly evolving) interface
- Run-time interface generated from program is problematic - too many libraries / modules for controller writers to consume
  - Consider generic interface also: APIs to discover and access objects
  - Generated interface remains useful for fast datapath extension software (… see demo …)

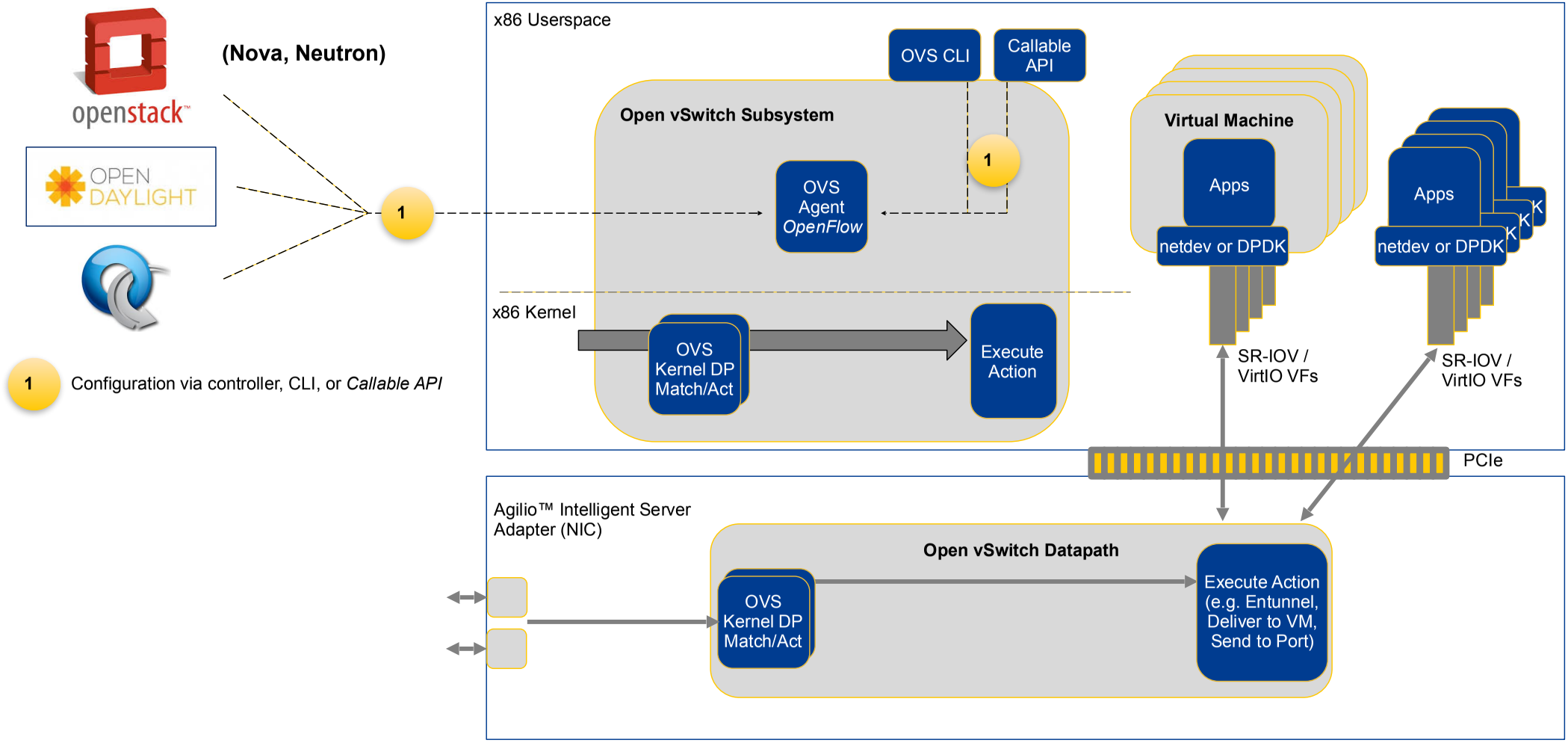
OVS Acceleration using Agilio™ SmartNIC

NETRONOME



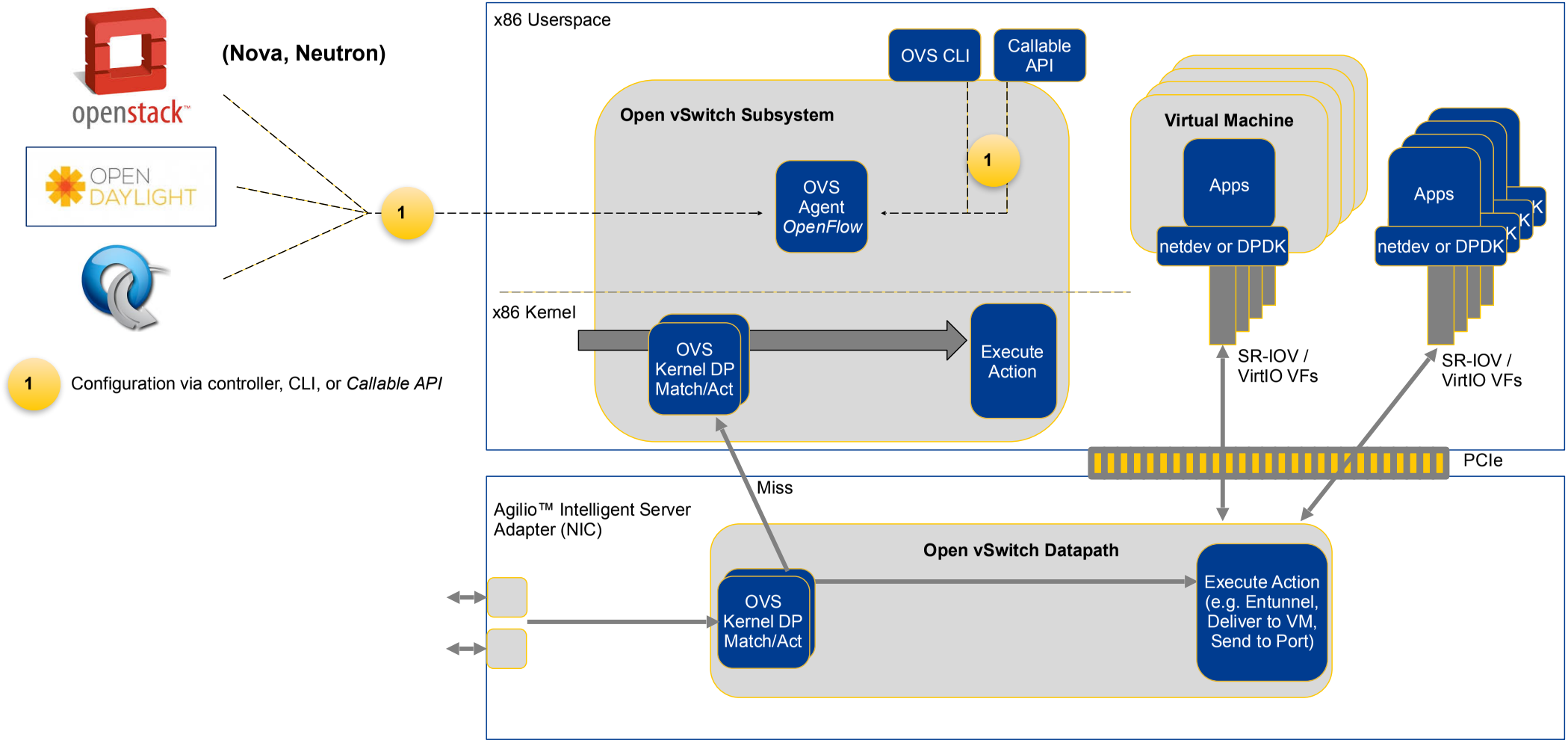
OVS Acceleration using Agilio™ SmartNIC

NETRONOME



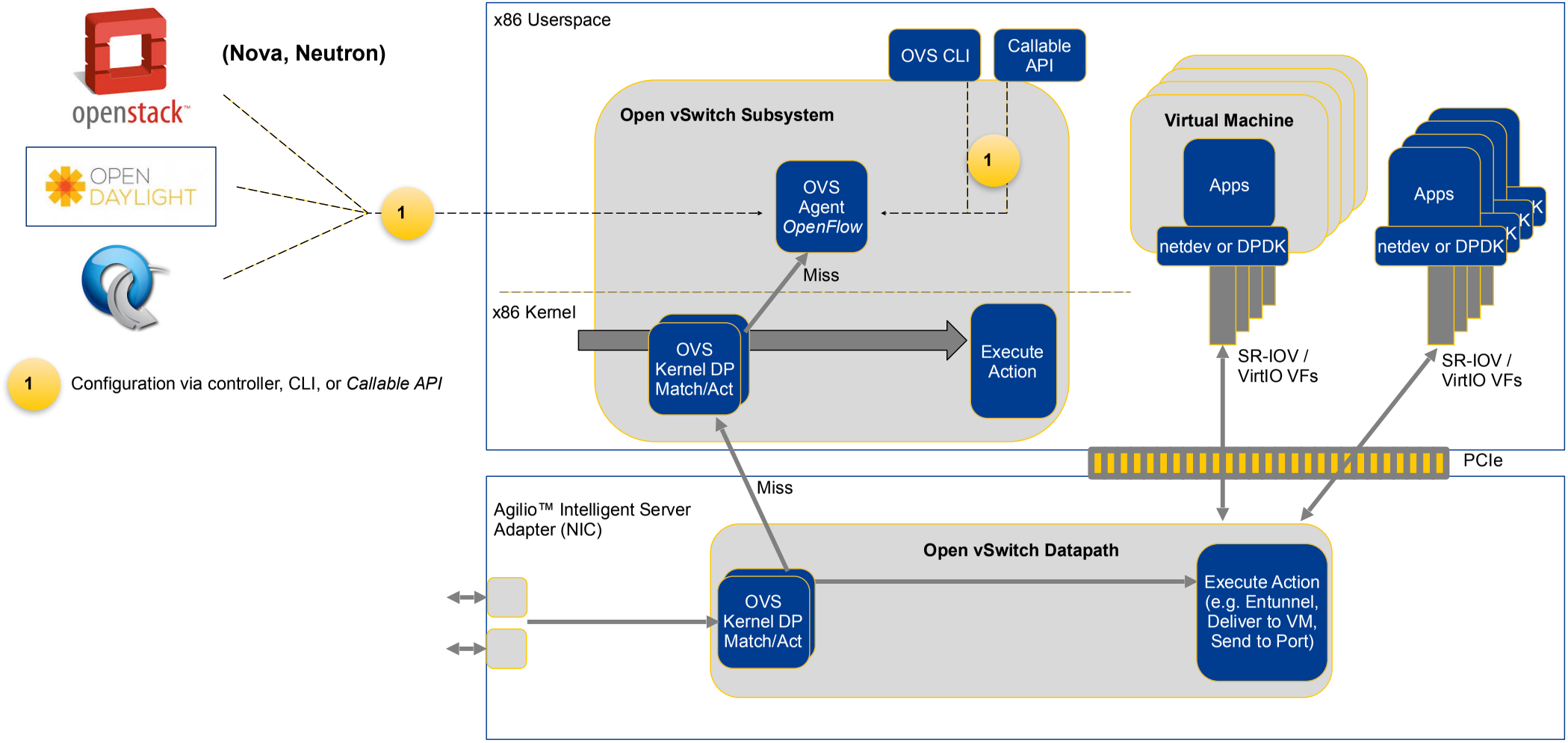
OVS Acceleration using Agilio™ SmartNIC

NETRONOME



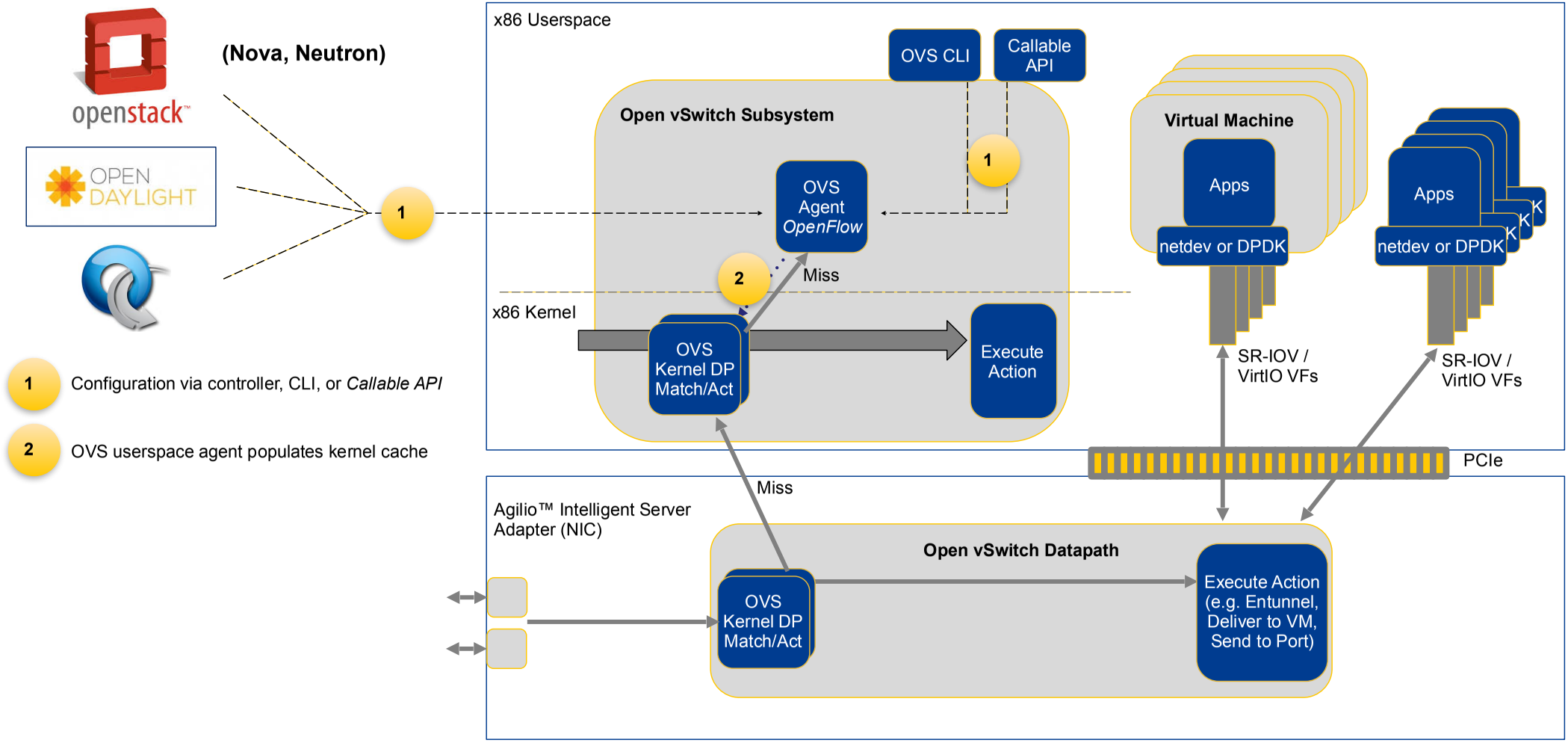
OVS Acceleration using Agilio™ SmartNIC

NETRONOME



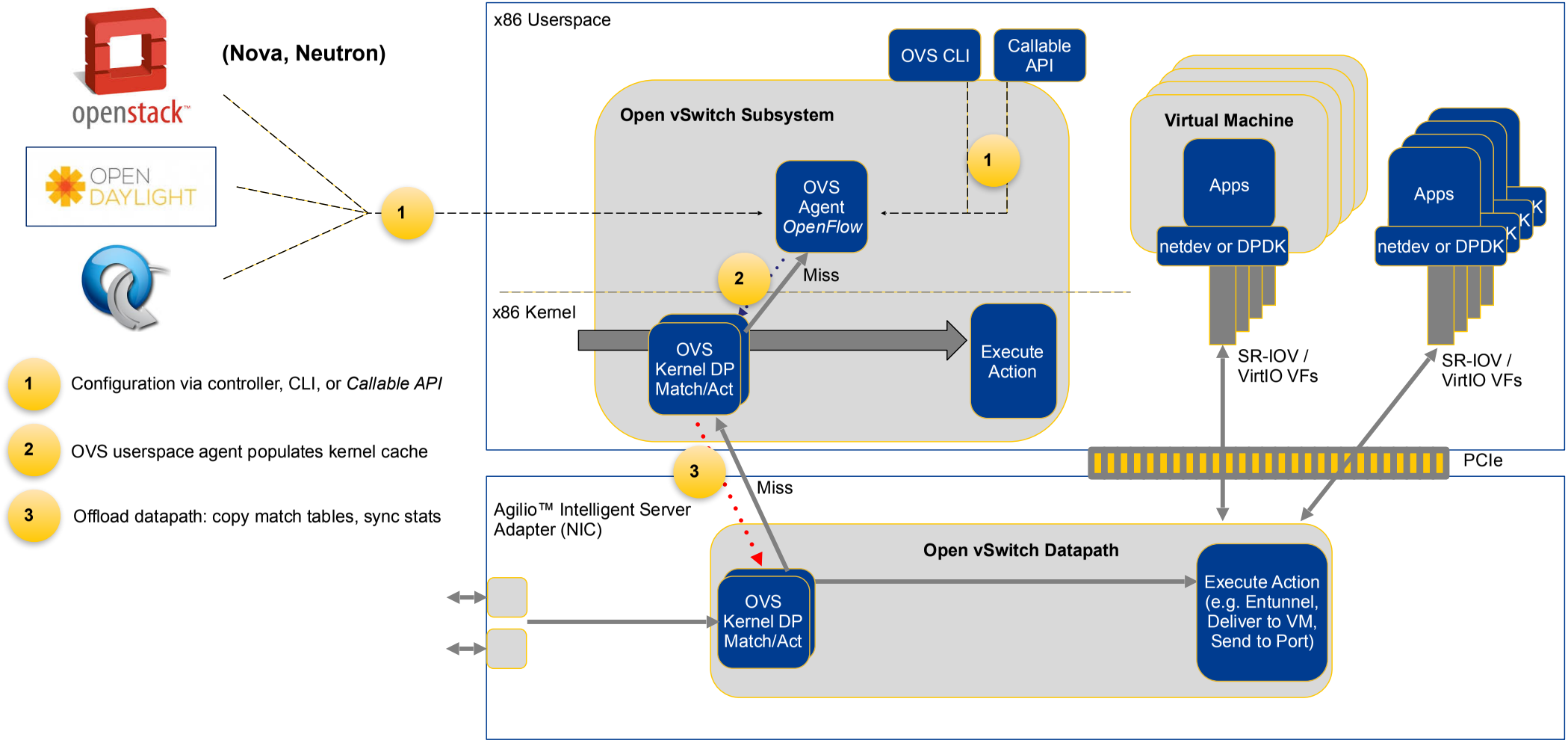
OVS Acceleration using Agilio™ SmartNIC

NETRONOME



OVS Acceleration using Agilio™ SmartNIC

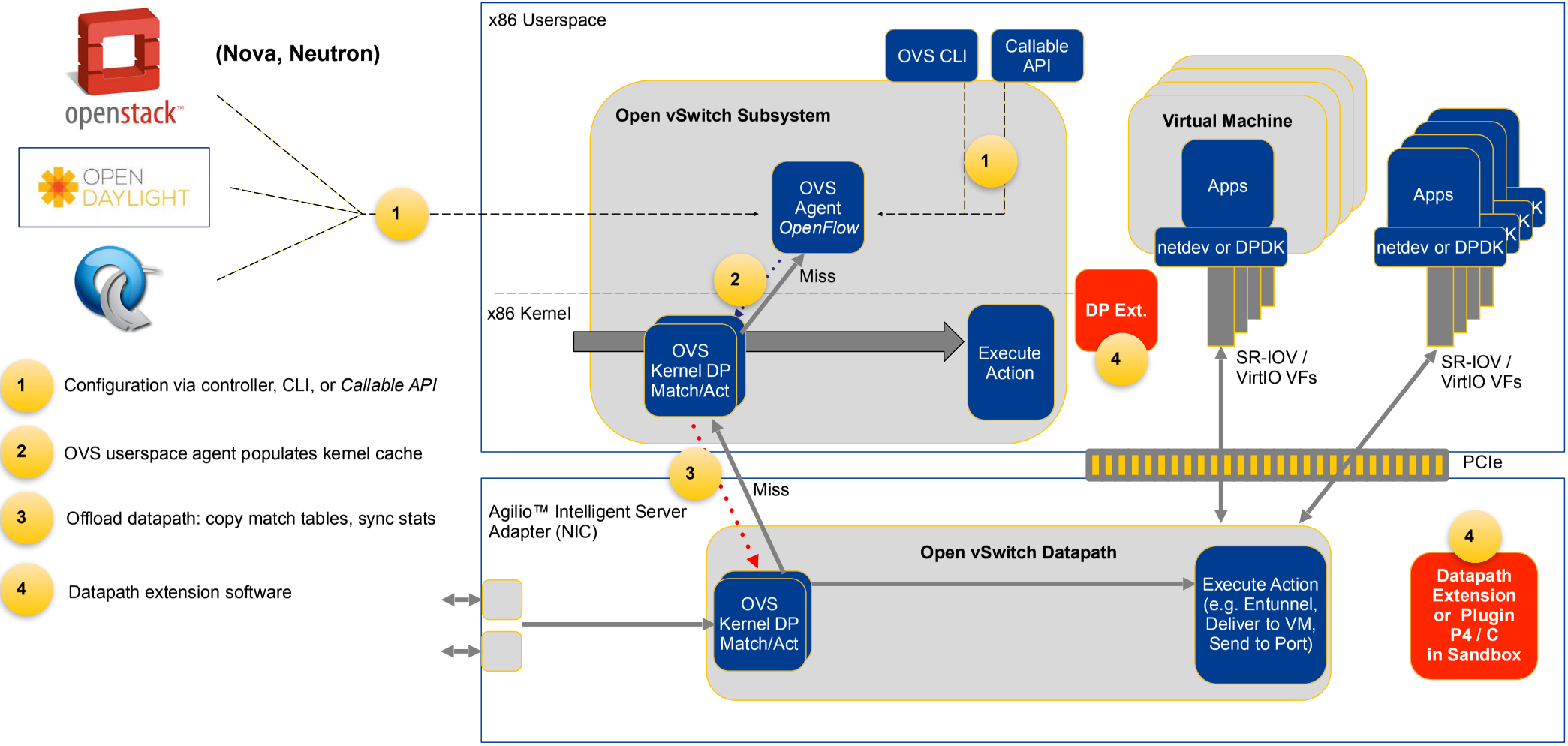
NETRONOME



1. Configuration via controller, CLI, or *Callable API*
2. OVS userspace agent populates kernel cache
3. Offload datapath: copy match tables, sync stats

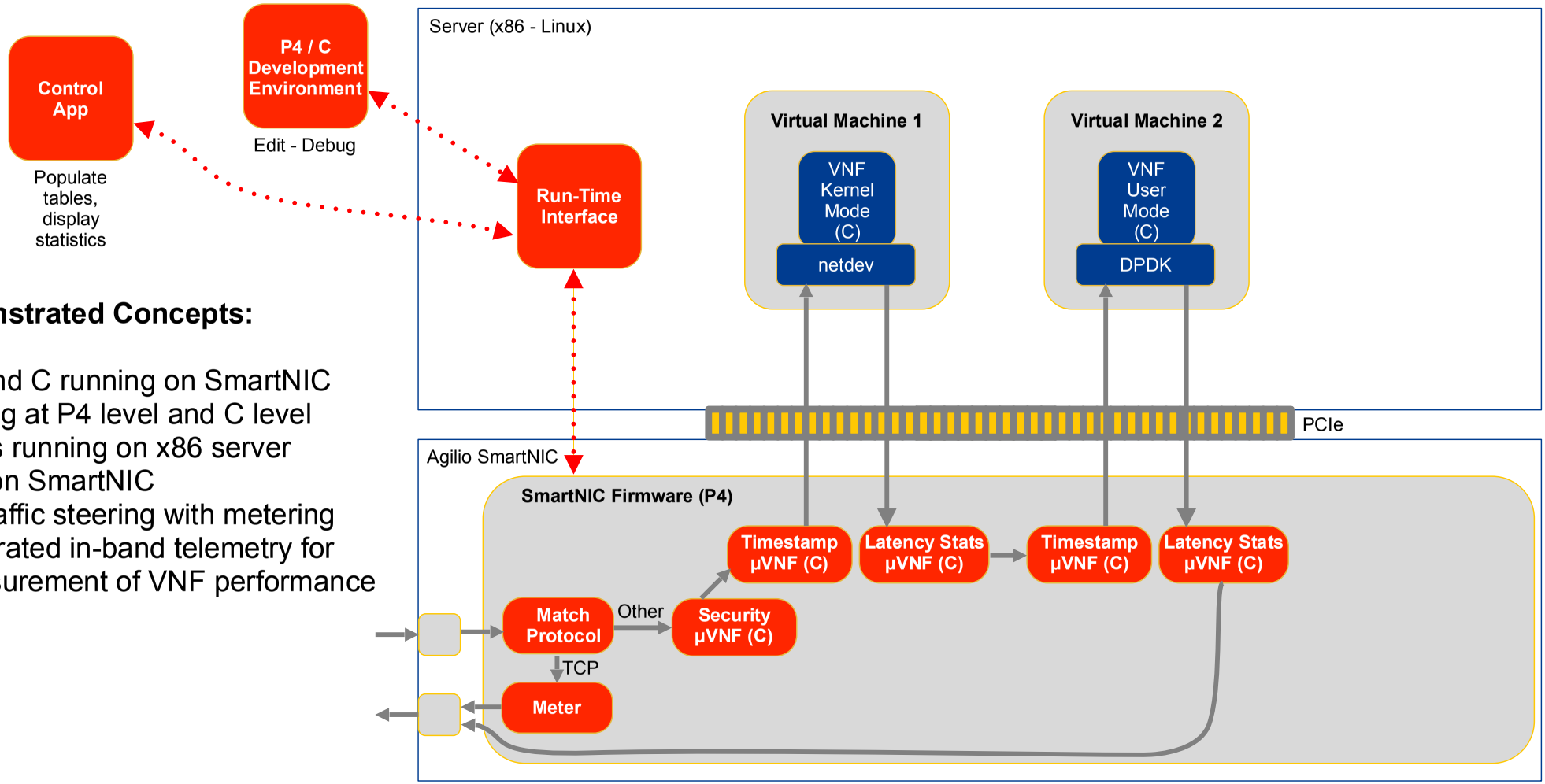
OVS Acceleration using Agilio™ SmartNIC

NETRONOME



P4 for SmartNICs with VNF and µVNF chaining

NETRONOME

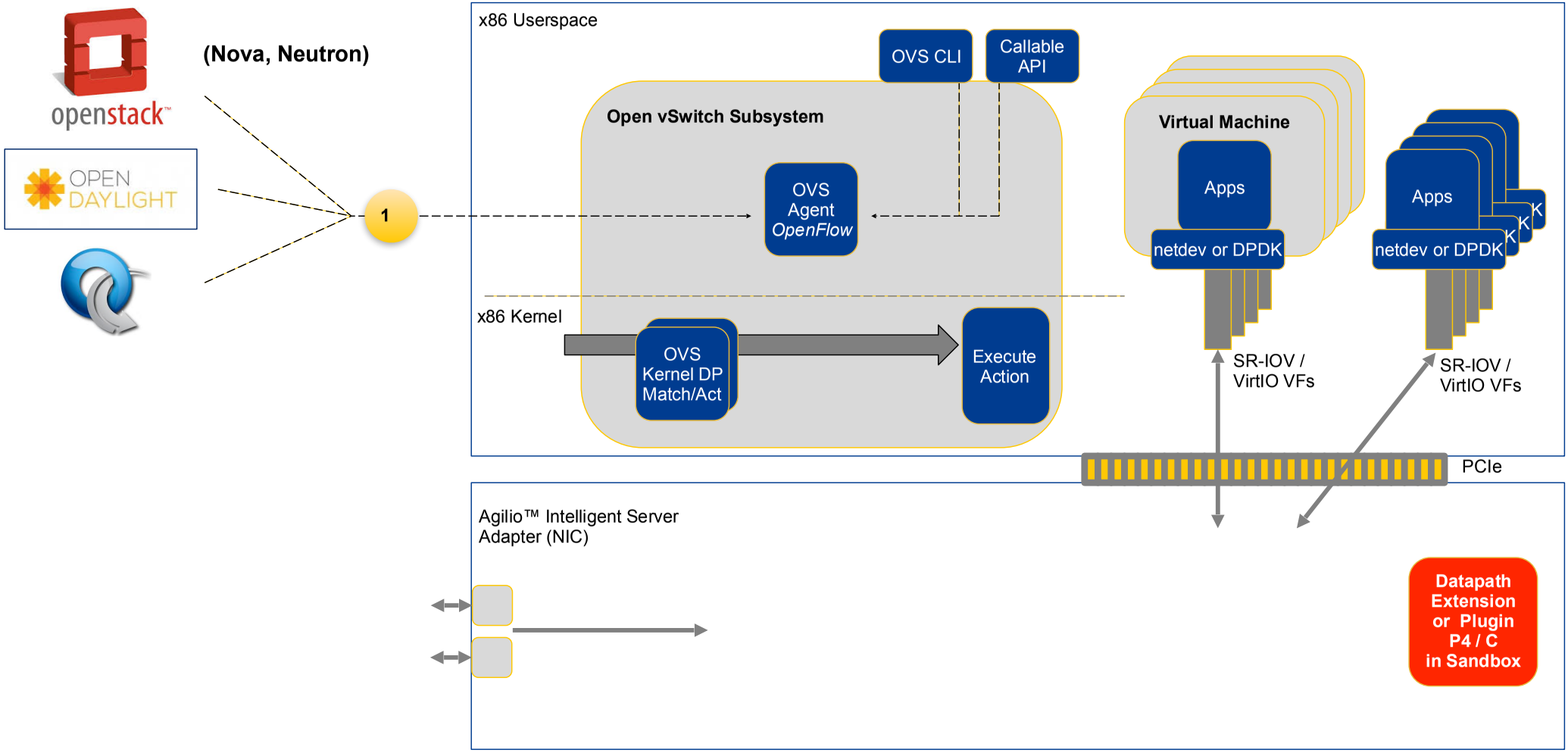


Demonstrated Concepts:

- P4 and C running on SmartNIC
- Debug at P4 level and C level
- VNFs running on x86 server and on SmartNIC
- P4 traffic steering with metering
- Integrated in-band telemetry for measurement of VNF performance

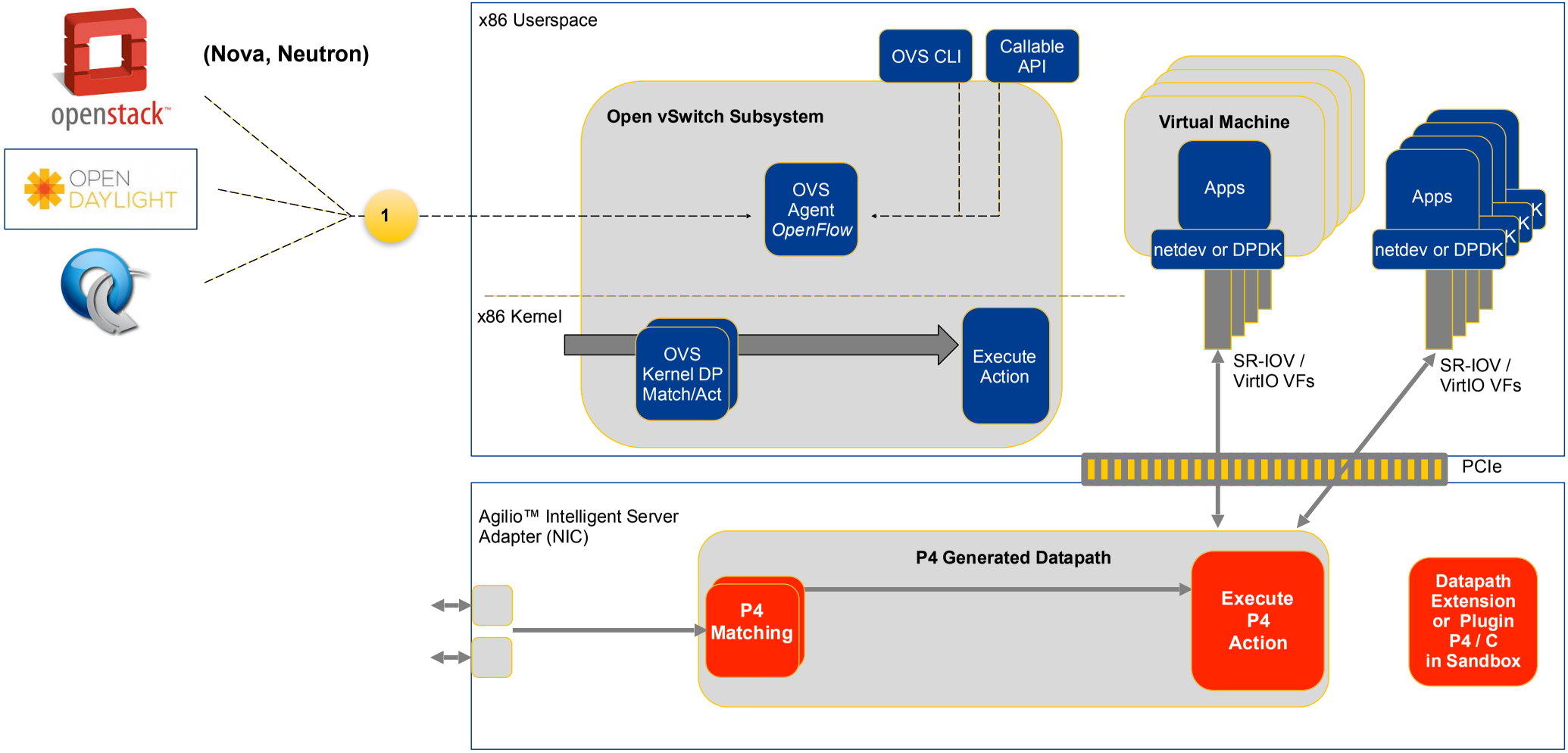
OVS “on” P4 Datapath

NETRONOME



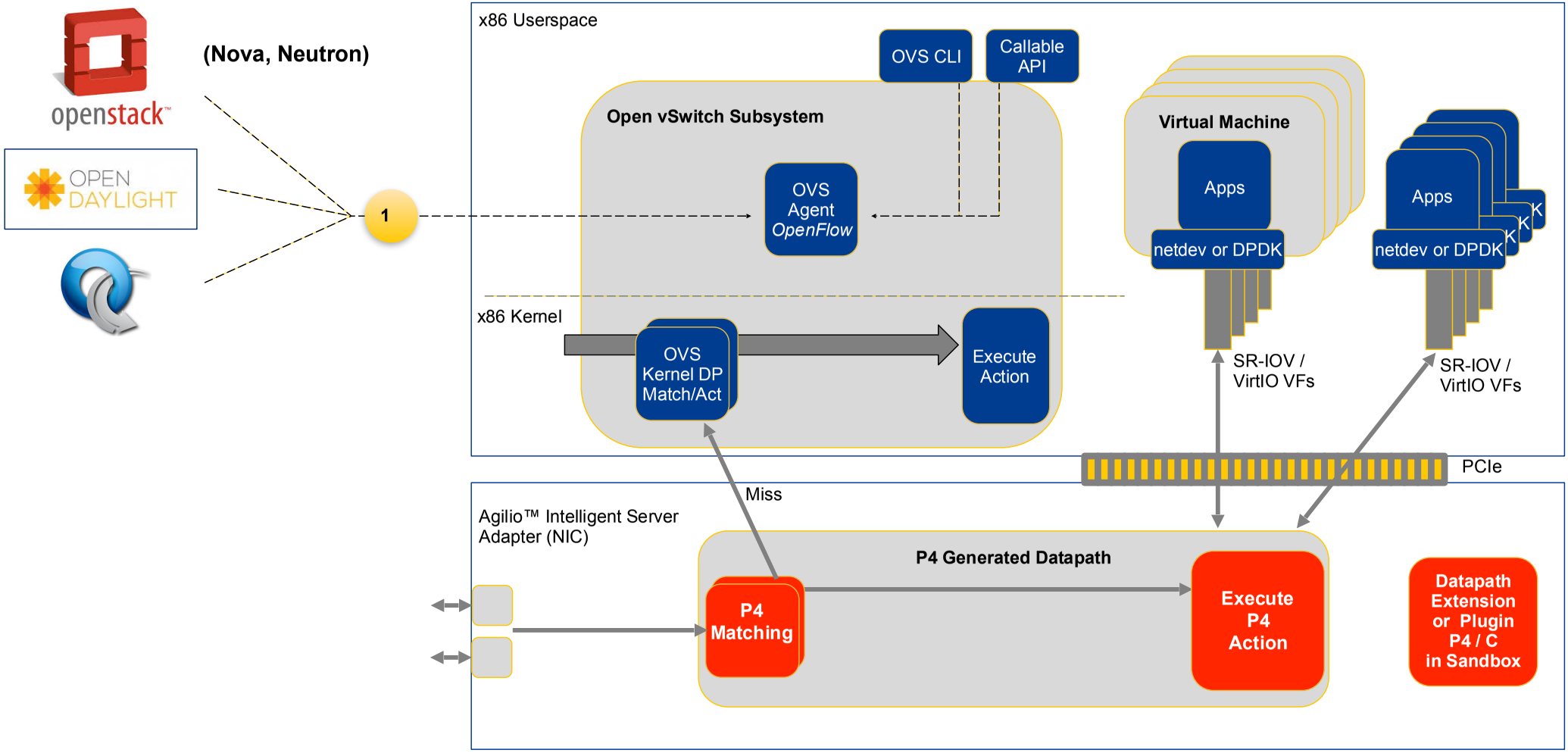
OVS “on” P4 Datapath

NETRONOME



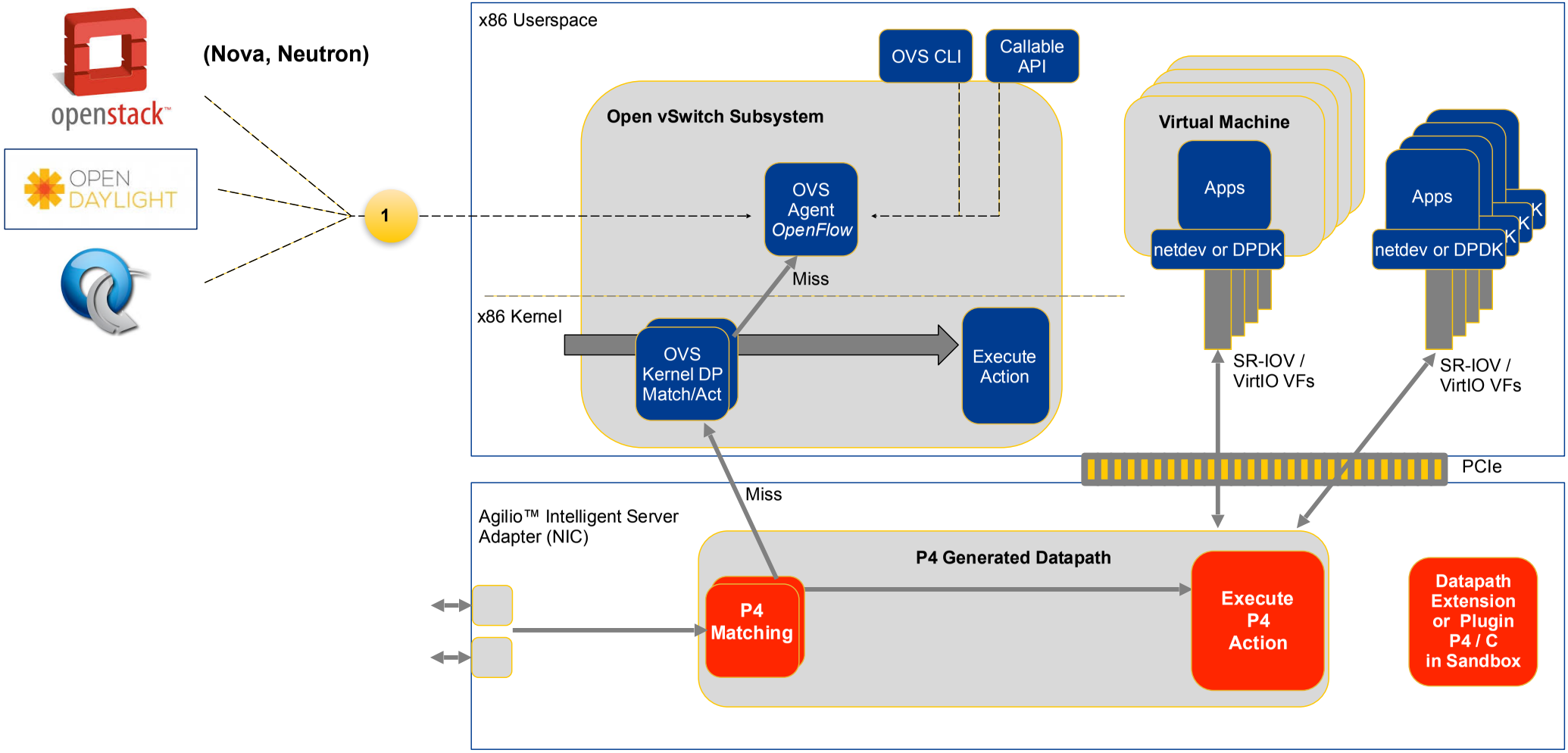
OVS “on” P4 Datapath

NETRONOME



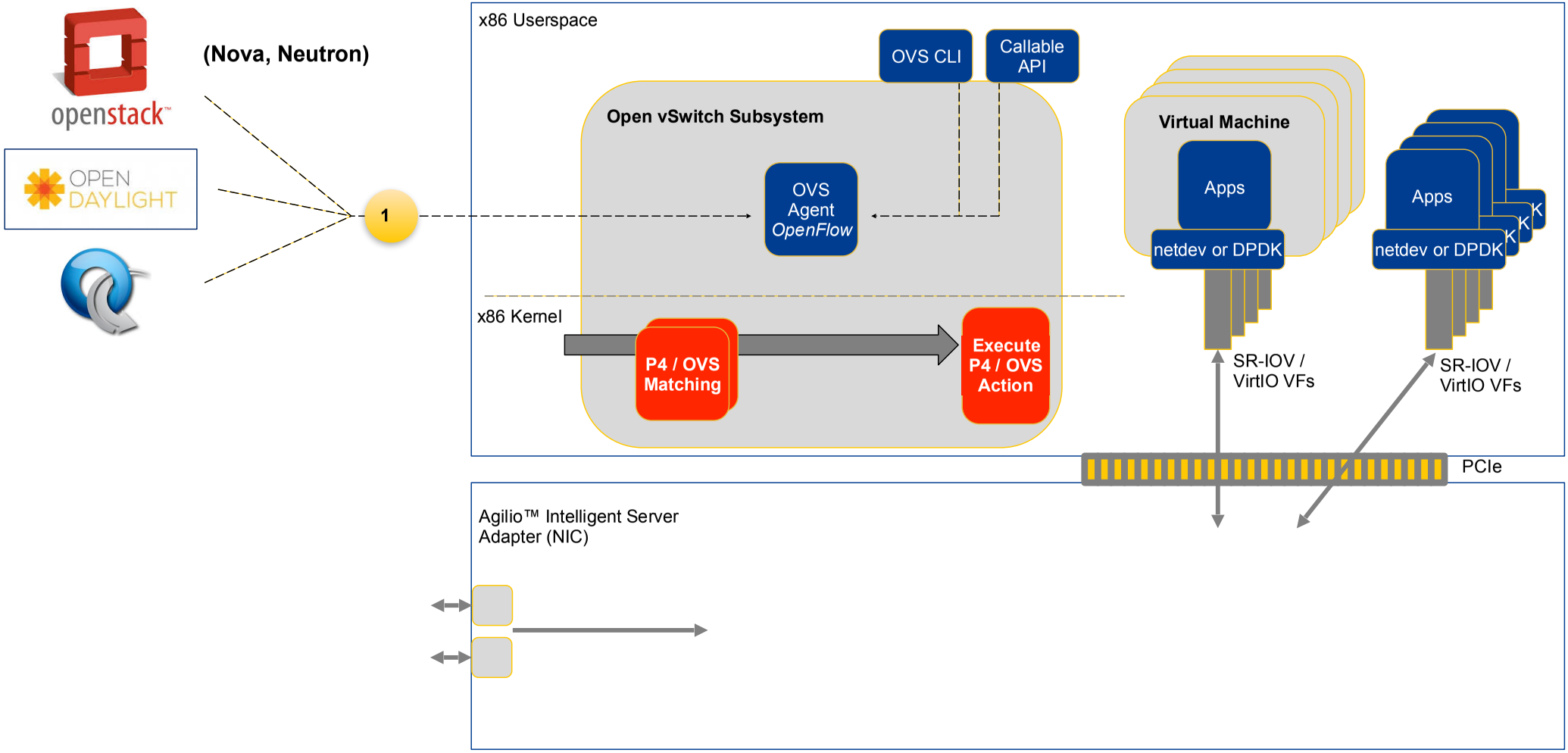
OVS “on” P4 Datapath

NETRONOME



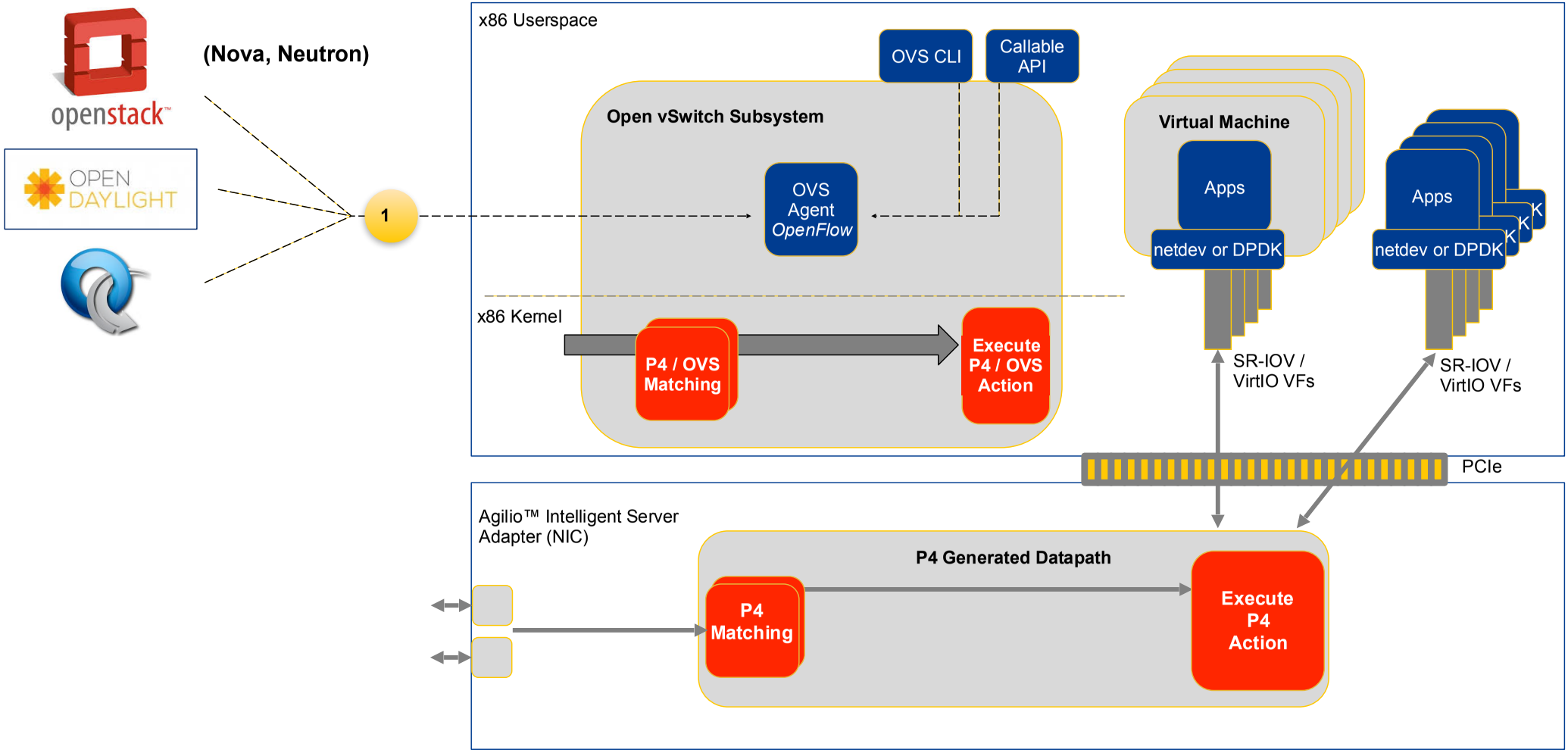
P4 “into” OVS Datapath

NETRONOME



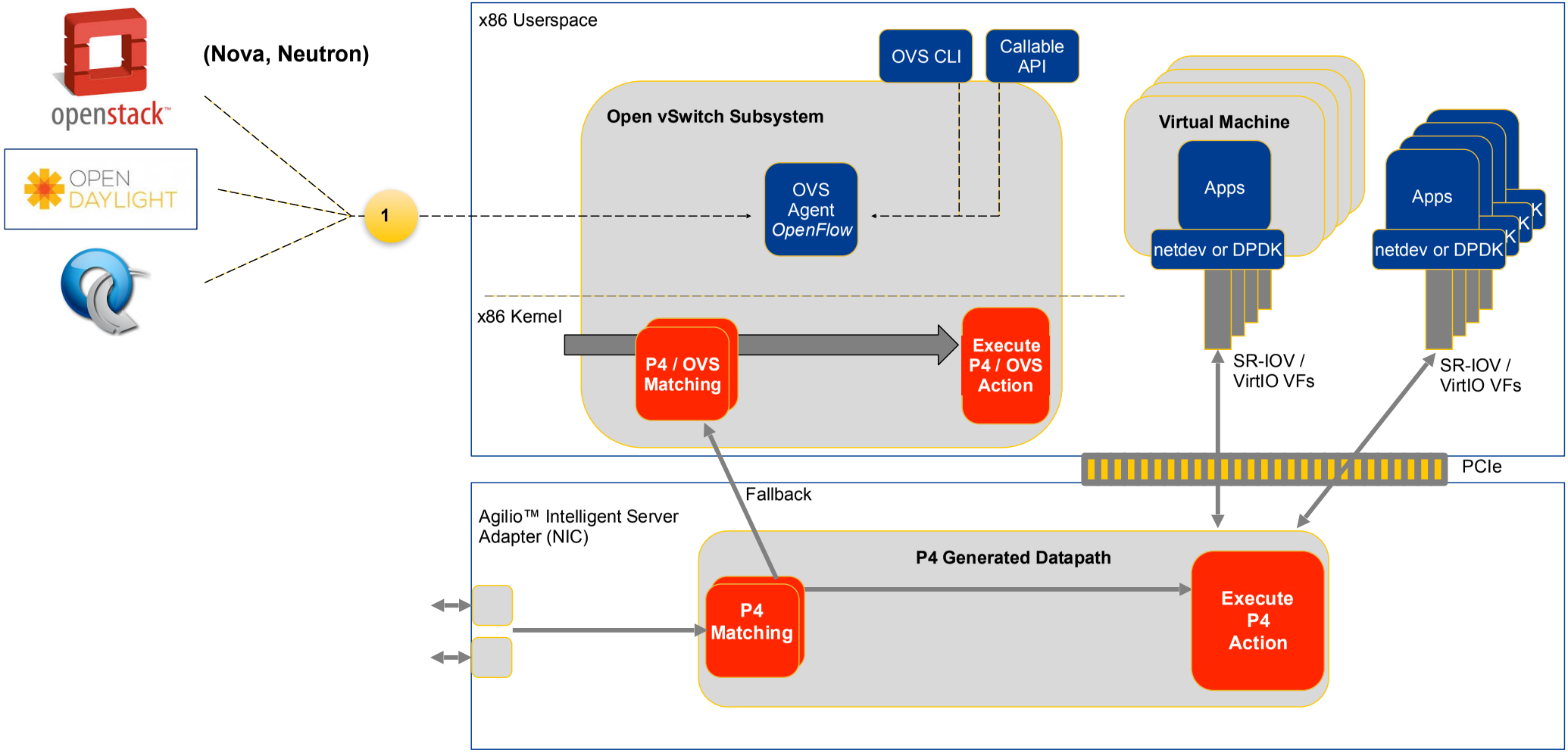
P4 “into” OVS Datapath

NETRONOME



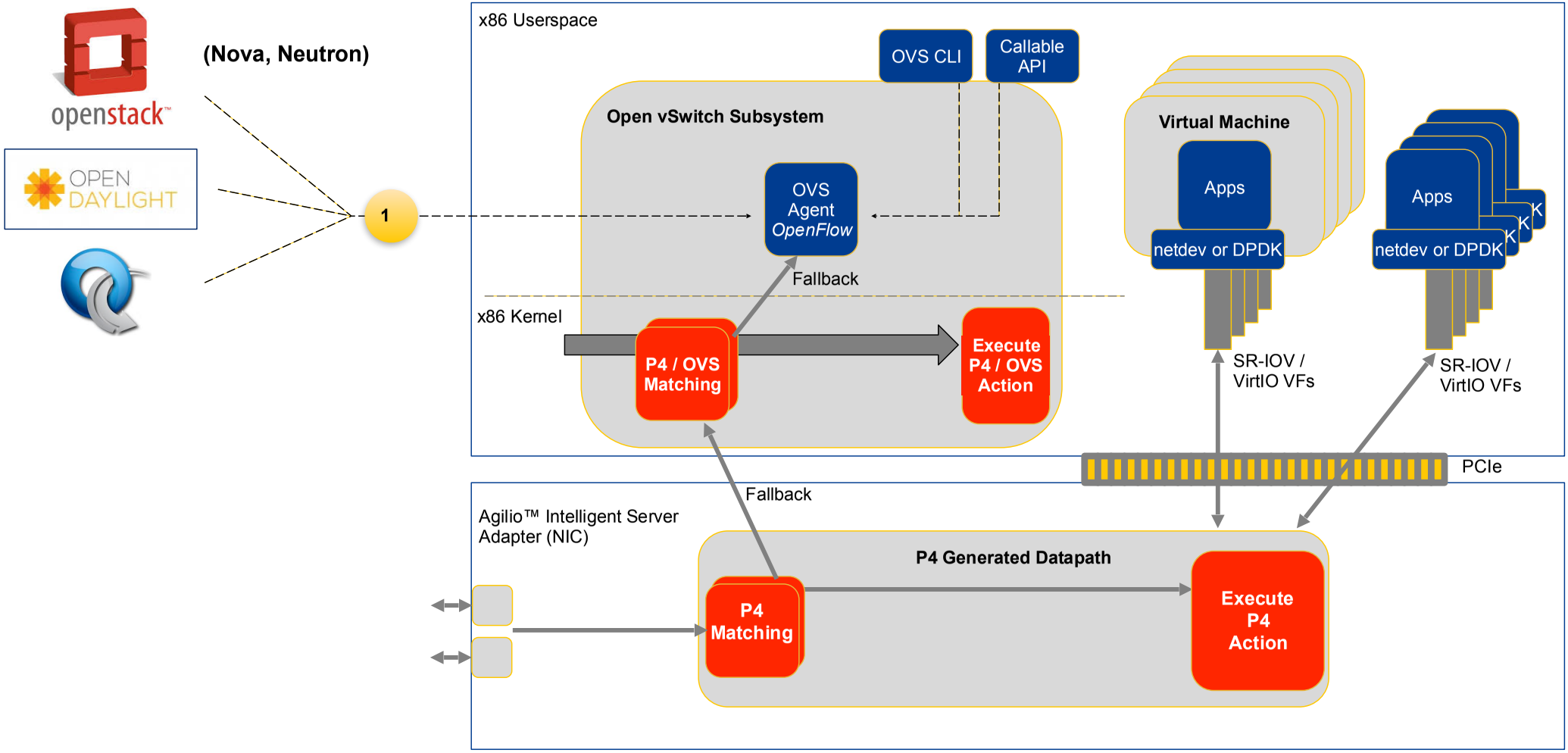
P4 “into” OVS Datapath

NETRONOME



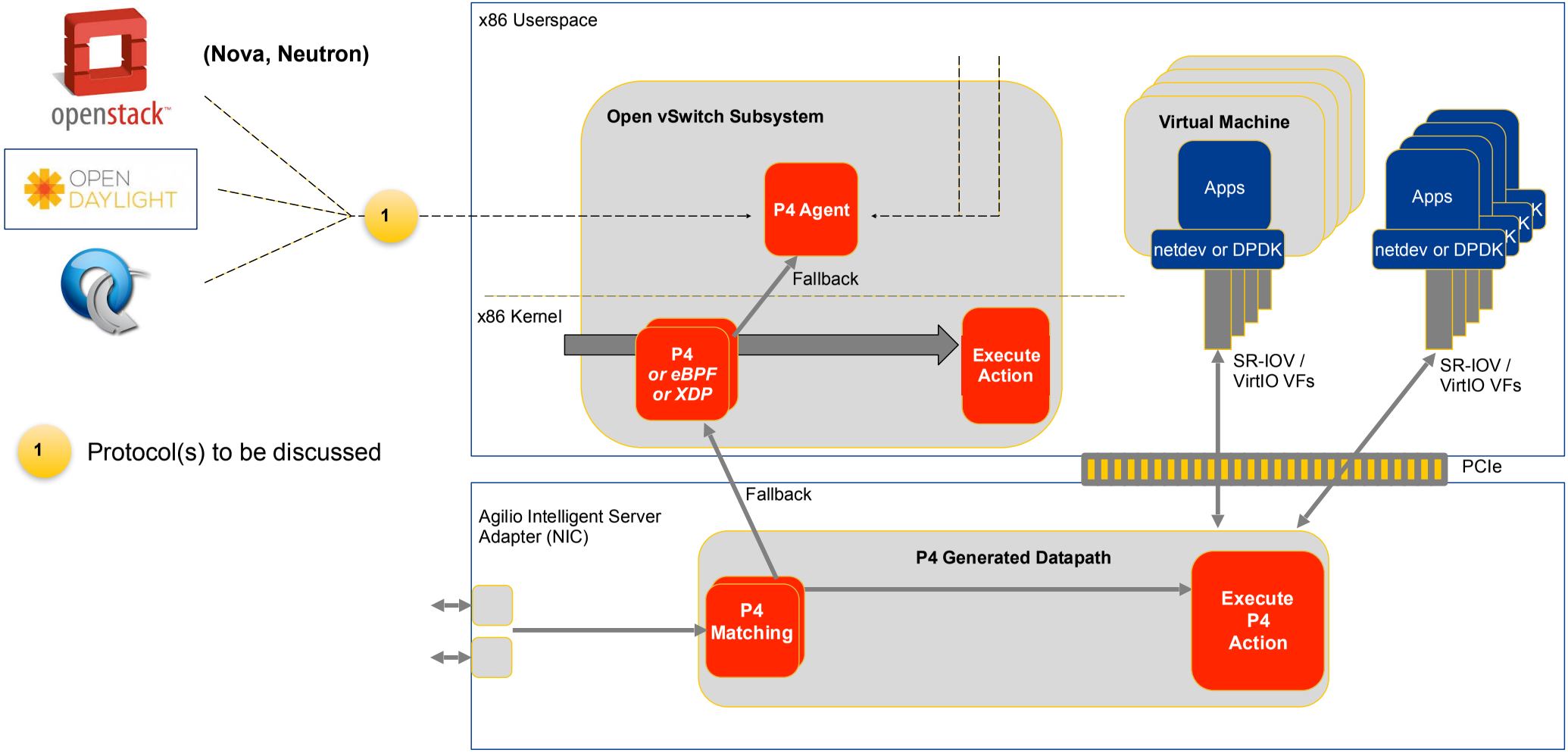
P4 “into” OVS Datapath

NETRONOME



P4 “instead of” OVS (Datapath and Control)

NETRONOME

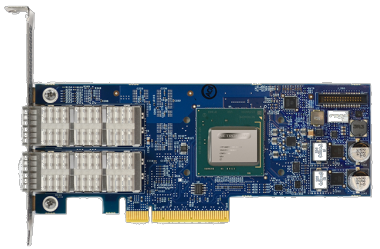


Performance Observations (Agilio™-CX SmartNIC)

NETRONOME

- Simple scenario
  - SmartNIC accelerated OVS: 35 Mpps @ < 1 core
  - P4 on SmartNIC: 36 Mpps @ < 1 core

  => Despite high level language overhead, only deploying the packet processing operations required by the application yields high performance



- More complex scenario
  - Unaccelerated OVS: 11 Mpps @ 22 server cores
  - SmartNIC accelerated OVS: 33 Mpps @ < 1 server core (>60x efficiency)
  - P4 on SmartNIC: 21 Mpps @ < 1 server core (>40x efficiency)

  => Hand-tuned implementation beats P4 (currently beta compiler: room for improvement)

  => Both offer significantly better efficiency (throughput per server core) than unaccelerated OVS

Also available: Agilio™-LX SmartNIC @ ~80 Mpps

Next Steps

NETRONOME

- Enable migration from existing OpenFlow network / cloud control interface to a P4 capable control interface (generated / generic)
  … collaborating with e.g. ONF, Open vSwitch, OpenStack, ODL, ONOS, NFV bodies…
- Standardize a persistent IR with a simple to parse format (YAML / JSON) and a stable schema
  … collaborating with e.g. OpenSourceSDN.org, ONF…
  - Use subset (extract) of IR as model, also as a description of the run-time interface
    - Equivalent: ".h" for ".c"
    - In effect a "TTP"
  - Supports multiple P4 language versions (or, in future, other languages)
    - Behavioral model v2 developers discovered the usefulness of a JSON based "IR"
  - Enables ecosystem of tools (optimizers, visualizers, front / back end compilers, etc.)
- Use Agilio™ SmartNICs
  - … to accelerate existing datapaths - OVS (OpenFlow), Contrail vRouter, Linux…
  - … to implement new datapaths - P4, eBPF/XDP…