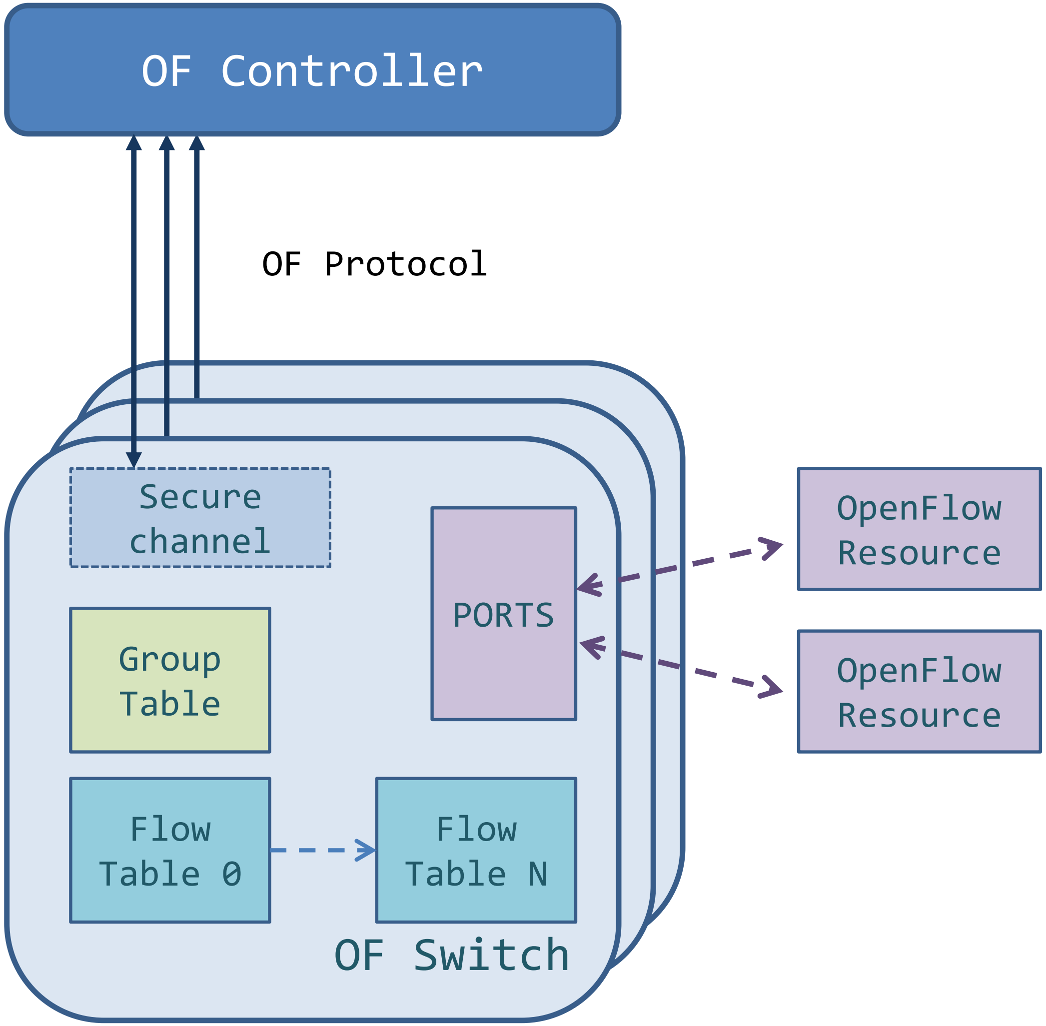
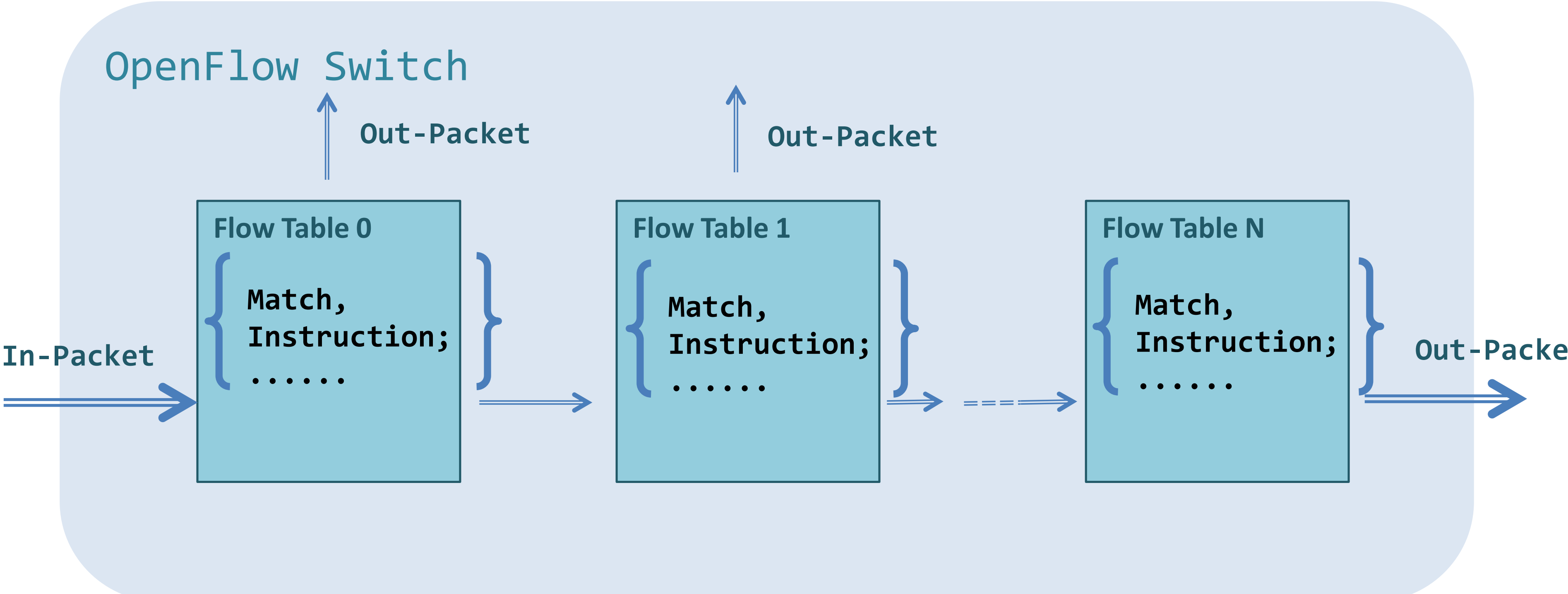
LINC – Open Source, Enterprise, Full-Functional OpenFlow Switch, written on Erlang

Dmitry Orekhov, Epam Systems

OpenFlow switch and Controller



Packet forwarding inside OpenFlow switch



- Packet may transferred to other table
- Packet header may be modified
- Packet may be forwarded to given port or just dropped
- Packet may be applied to given QoS

Flow table entry: key elements

Match Fields	Priority	Counters	Timeout	Cookies	Instruction set

Match criteria:

- Ingress-port
- Ethernet MAC
- ARP
- IPv4 and IPv6
- TCP ports
- VLAN, MPLS etc.

Instruction:

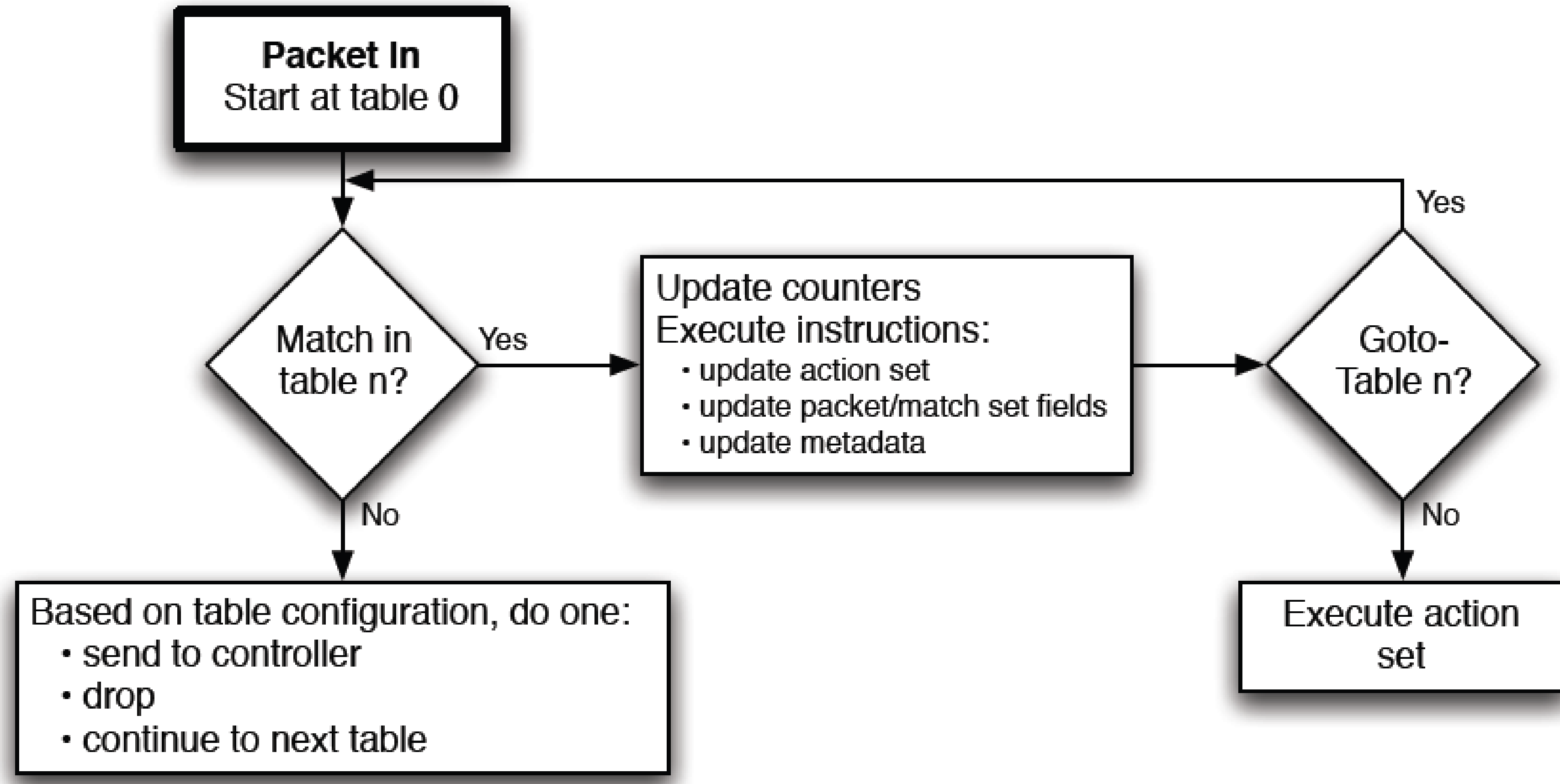
- Go-To Table
- Modify Metadata
- Action Set {forward, apply QoS, drop, Apply to Group}

OpenFlow examples

	Switch port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Prot	TCP sport	TCP dport	Action
Switching	*	*	00:1f :..	*	*	*	*	*	*	Port6
Flow switching	Port3	00:2 0..	00:1f ..	0800	Vlan1	1.2.3.4	5.6.7.8	4	17264	Port6
Firewall	*	*	*	*	*	*	*	*	22	Drop
Routing	*	*	*	*	*	*	5.6.7.8	*	*	Port6
VLAN switching	*	*	00:1f ..	*	Vlan1	*	*	*	*	Port6, port7, port8

OpenFlow can be compared to the instruction set of a CPU. It specifies basic primitives that can be used by an external software application to program the forwarding plane of network devices, just like the instruction set of a CPU would program a computer system.

Matching



Packet In
Start at table 0
Yes
Match in table n?
Yes
Update counters
Execute instructions:
• update action set
• update packet/match set fields
• update metadata
Goto-Table n?
No
No
Based on table configuration, do one:
• send to controller
• drop
• continue to next table
Execute action set

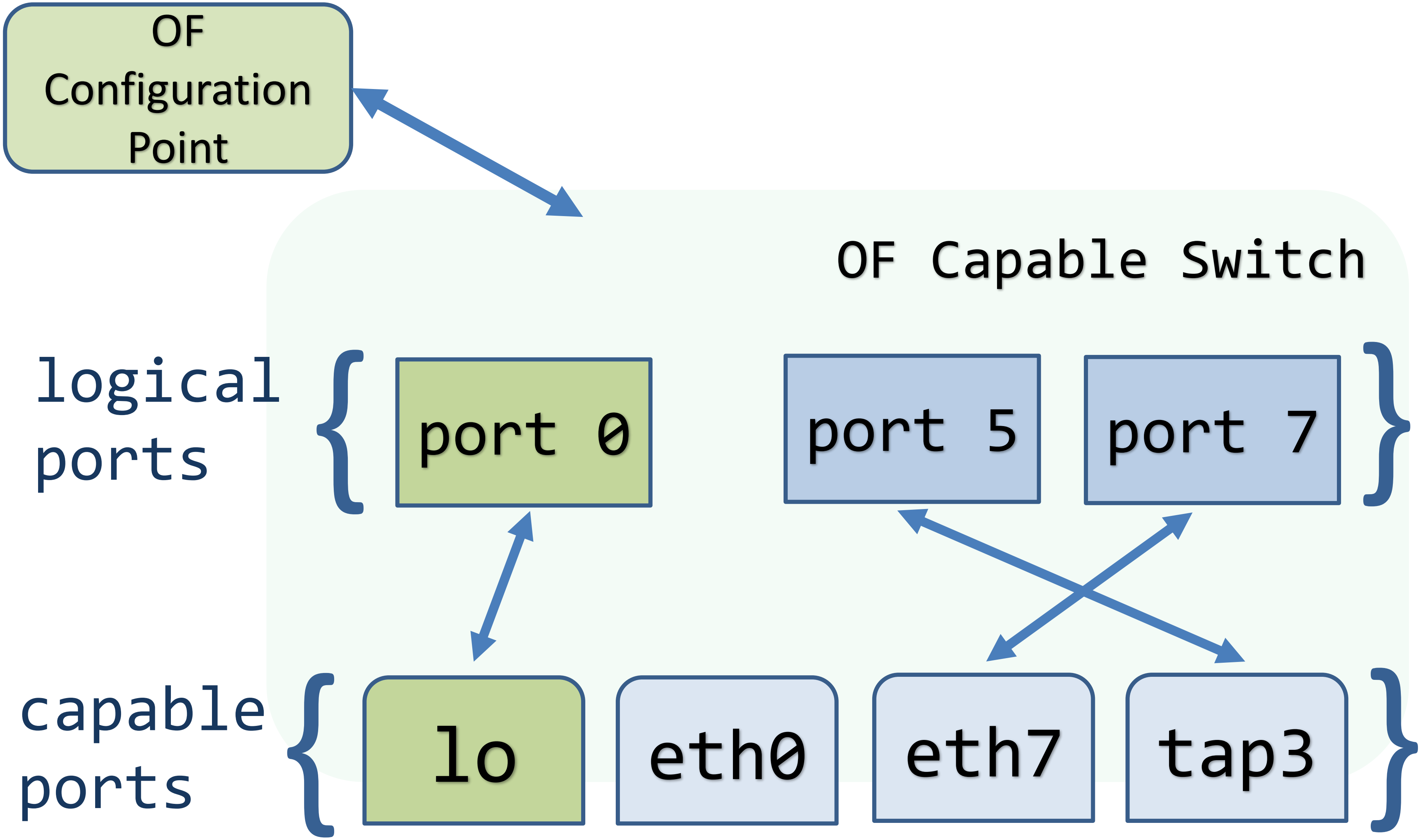
Group Table: “Aspects” of OpenFlow

Group Identifier	Group Type	Counters	Action bucket

All
Select
Indirect
Fast Failover

Groups represent sets of actions for flooding, as well as more complex forwarding semantics (e.g. multipath, fast reroute, and link aggregation). As a general layer of indirection, groups also enable multiple flows to forward to a single identifier (e.g. IP forwarding to a common next hop). This abstraction allows common output actions across flows to be changed efficiently.

OF Config – the new concept of OpenFlow Capable Switch



NETCONF

Layer	Example
Content	<capable-switch>…</capable-switch>
Operations	<get-config>,<set-config>,<notification>
RPC	<rpc>,<rpc-reply>
Transport Protocol	SSH, TLS, BEEP, SOAP

Example

```
<capable-switch>
   <id>CapableSwitch0</id>

   <configuration-points>
   ...
   </configuration-points>

   <resources>
   ...
   </resources>

   <logical-switches>
   ...
   </logical-switches>
</capable-switch>
```

So what do we really have?

- OpenFlow capable switch looks like a container of many (probably thousands and tens of thousands) processes which are totally independent.
- Processes can be created/terminated in runtime, always, always!
- Connections – probably millions of them!
- One process must not crash another – no way!
- What about support new incoming OpenFlow versions? Do we need stop our switches?
- What about scalability? Who does take care of this?

- Last but not the least: Binary encoding/decoding process is too boring!

Erlang: Processes

- ✓ **Process creation – `spawn(fun task/0).`**
  It takes microseconds and down to hundreds bytes. Tens of thousands processes can be created per seconds.

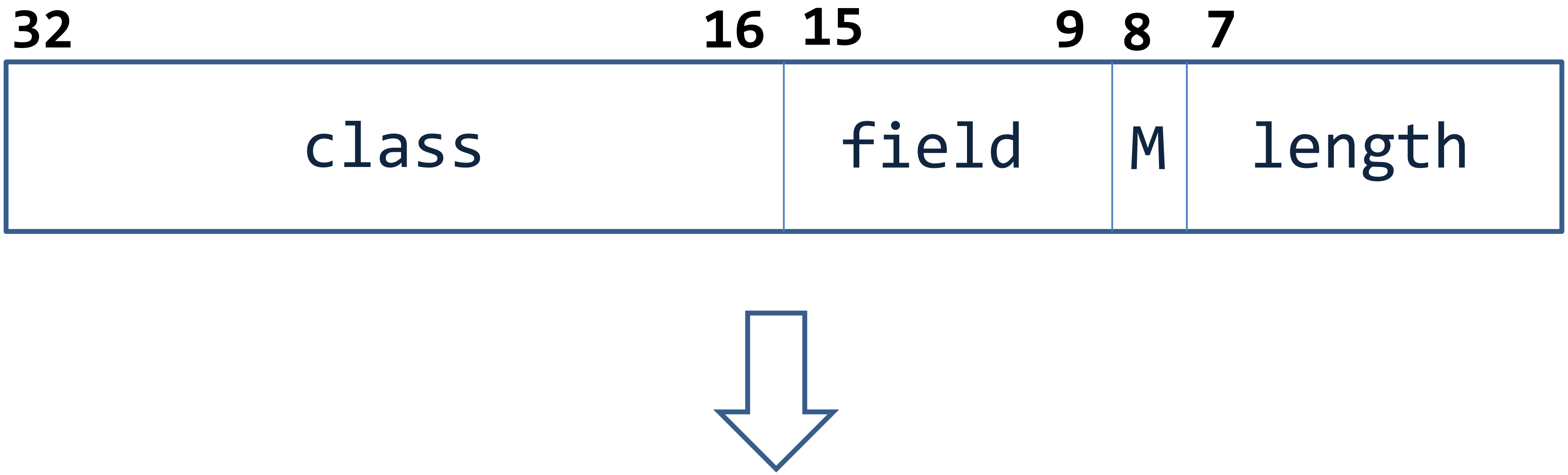
- ✓ **Process isolation.**
  Every process is isolated inside Erlang VM.
  Processes communicate using queues of messages.
  You're able to use global variables, but you won't!

- ✓ **Processes as an object.**
  Local variables as an internal state
  Messages as methods

Erlang: Crash handling

- ✓ The main principle: “Let it fall”
- ✓ Don’t use try-catch
- ✓ Supervisors – special processes controlling another processes.

Erlang: Binary Encoding/Decoding

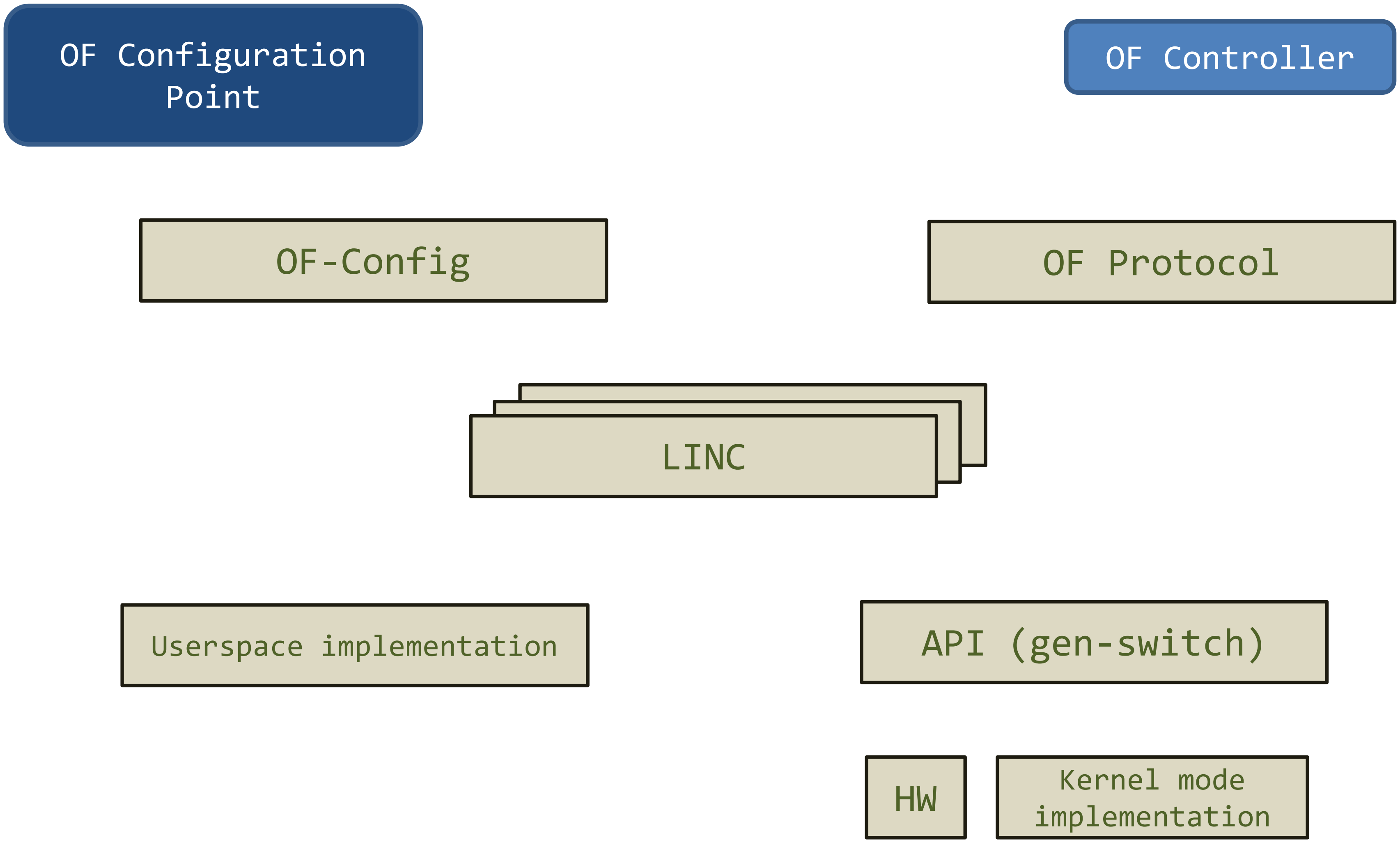



```
decode_match_field(<<Header:4/bytes, Binary/bytes>>) ->
   <<ClassInt:16, FieldInt:7, HasMaskInt:1,
      Length:8>> = Header,
```

Erlang: Development and Deployment

- ✓ OTP – A really reach library. Supervisors, evens handler, final-state machine – this is OTP.
- ✓ Erlang designed considering the fact that developers put bugs in the code – and try to stop developers to do it! Did I tell about immutability?
- ✓ Modules can be fixed and replaced in runtime – but don’t ask me how!

LINC switch



OF Configuration Point
OF Controller
OF-Config
OF Protocol
LINC
Userspace implementation
API (gen-switch)
HW
Kernel mode implementation

Is LINC REALLY able?

- 10,000 connections benchmark – Erlang looks great.
- All OpenFlow 1.3 features are implemented.
- ONF PlugFest – LINC was tested in topologies, together with enterprise switches and controllers.

- But we relly didn’t test it as a switch, under high load
- But we really out of System Integration Testing for LINC

How you can try it?

- ✓ Linux box with Erlang, scons and pcap library
- ✓ git clone https://github.com/FlowForwarding/LINC-Switch
- ✓ cd LINC-Switch
- ✓ make rel

- ✓ You can refer to README on GitHub. Also, wiki contains document with simple examples and topologies

Reference

- OpenNetworking Foundation (OpenFlow documents)

  https://www.opennetworking.org/about/onf-documents
- FlowForwarding

  http://www.flowforwarding.org/
- GitHub repository:

  https://github.com/FlowForwarding/LINC-Switch
- Testing framework for OpenFlow:

  http://onlab.us/testing.html
- And me, Dmitry Orekhov (Dmitry_Orekhov@epam.com)