Virtual switching technologies and Linux bridge

Toshiaki Makita
NTT Open Source Software Center

Today's topics

- **Virtual switching technologies in Linux**
  - Software switches (bridges) in Linux
  - Switching technologies for KVM environment
  - Performance of switches
  - Userland APIs and commands for bridge

- **Introduction to Recent features of bridge (and others)**
  - FDB manipulation
  - VLAN filtering
  - Learning/flooding control

- **Features under development**
  - 802.1ad (Q-in-Q) support for bridge
  - Non-promiscuous bridge

Who is Toshiaki Makita?

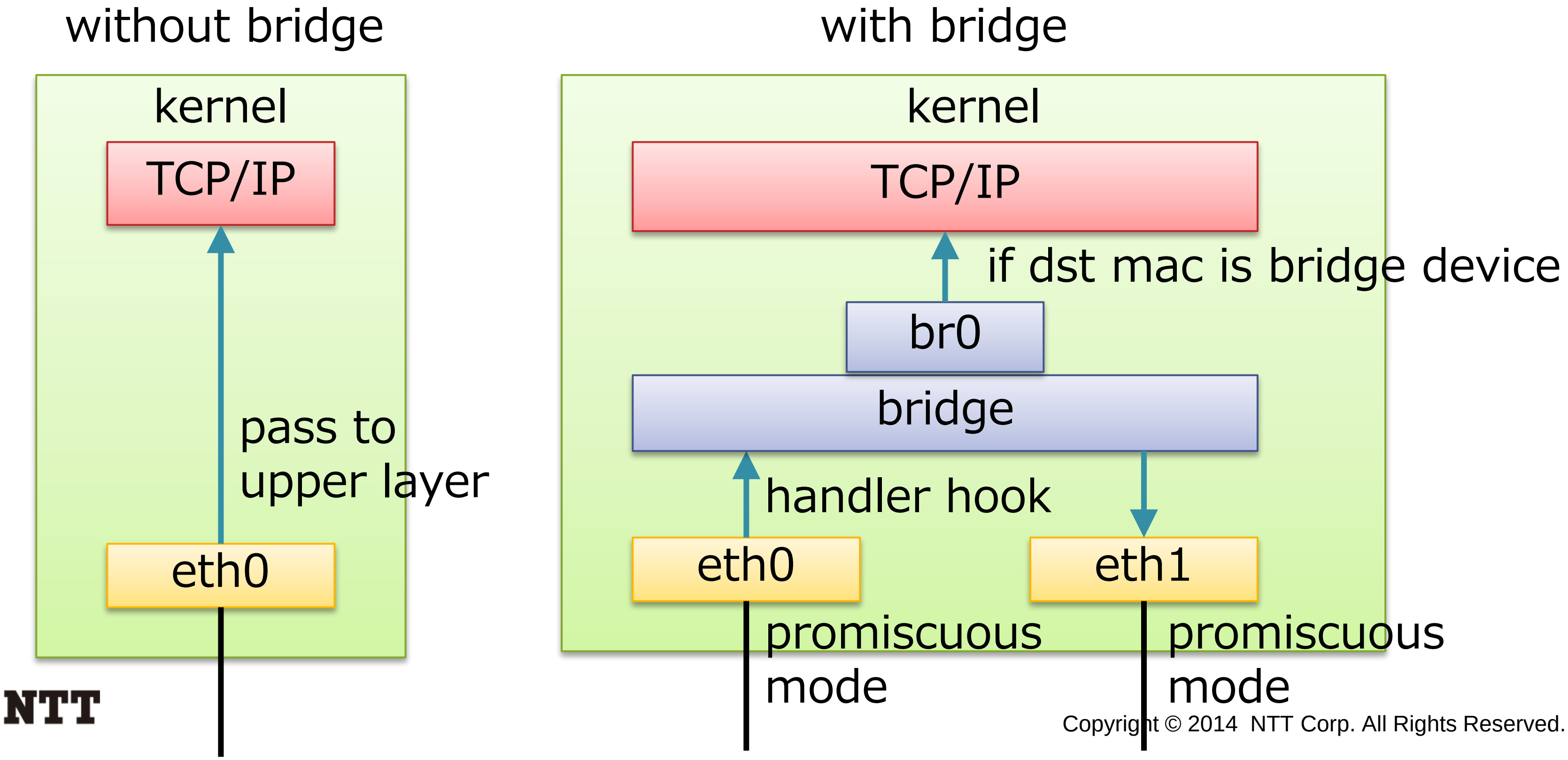
- **Linux kernel engineer at NTT Open Source Software Center**
- **Technical support for NTT group companies**
- **Active patch submitter on kernel networking subsystem**
  - bridge, etc.

Software switches in Linux

- **Linux has 3 types of software switches**
  - bridge
  - macvlan
  - Open vSwitch

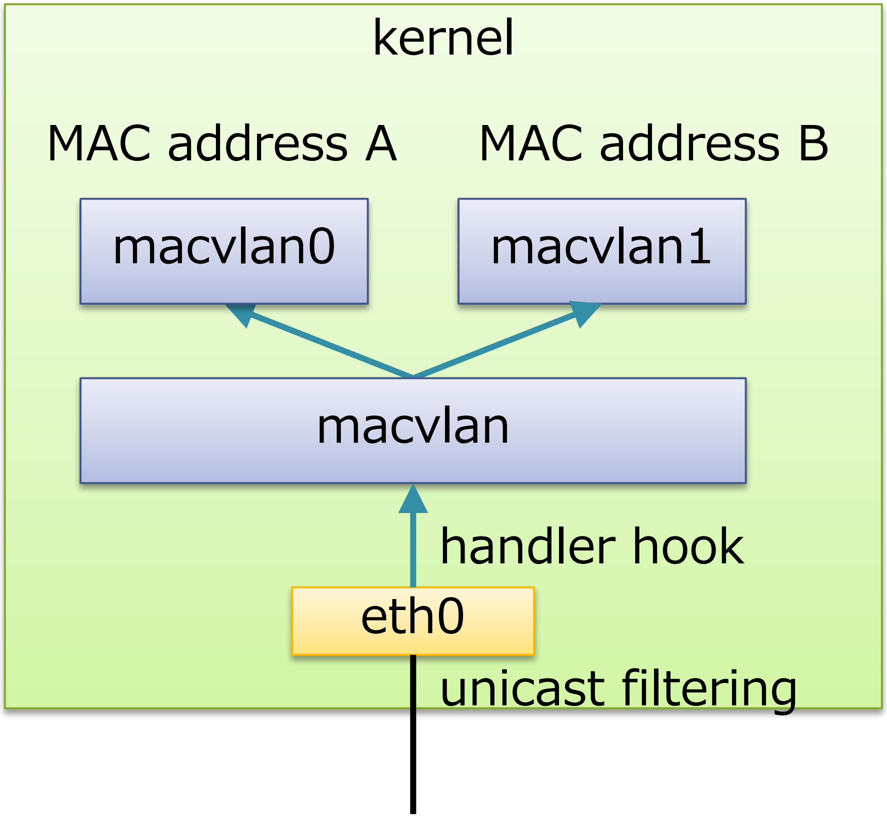
bridge

- **HW switch like device (IEEE 802.1D)**
  - Has FDB (Forwarding DB), STP (Spanning tree), etc.
  - Using promiscuous mode that allows to receive all packets
    - Common NIC filters unicast whose dst is not its mac address without promiscuous mode
    - Many NICs also filter multicast / vlan-tagged packets by default



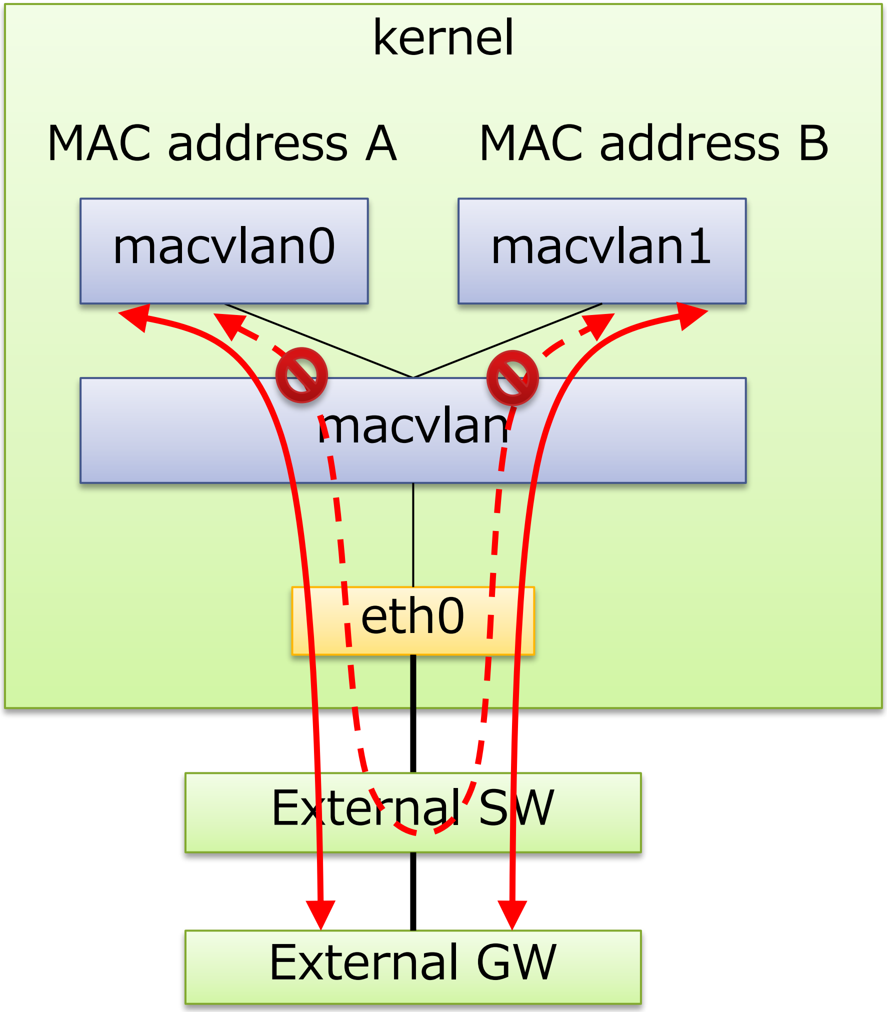
macvlan

- **VLAN using not 802.1Q tag but mac address**
- **4 types of mode**
  - private
  - vepa
  - bridge
  - passthru
- **Using unicast filtering if supported, instead of promiscuous mode (except for passthru)**
  - Unicast filtering allows NIC to receive multiple mac addresses



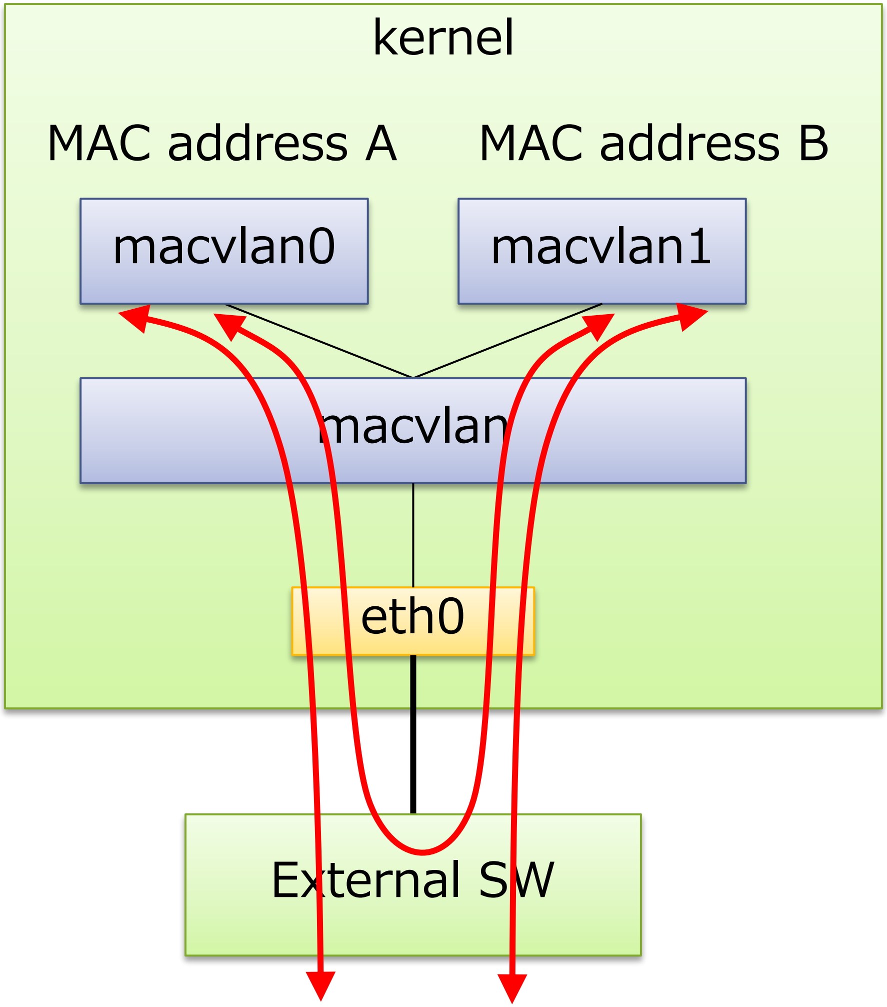
macvlan (private mode)

- vlan device like behavior
- Not a bridge
- Prohibit inter-macvlan traffic (except for those via external GW)



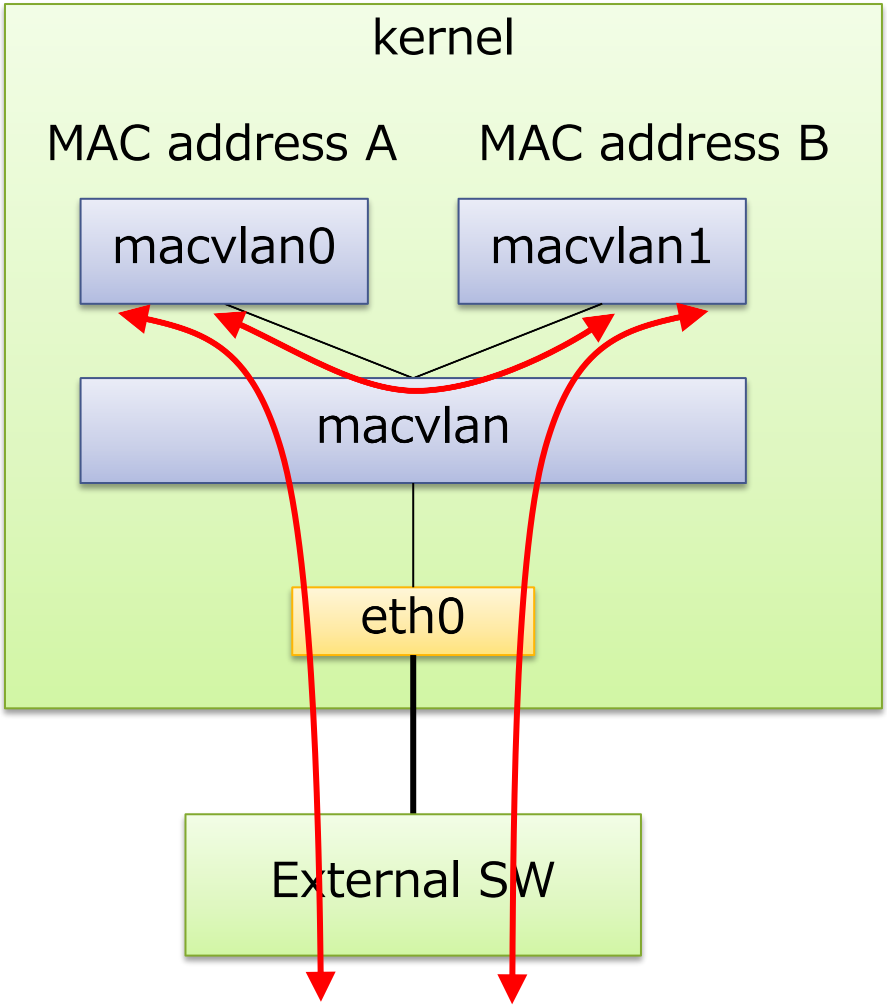
macvlan (vepa mode)

- Similar to private mode
- Allow traffic between macvlans (via external SW)



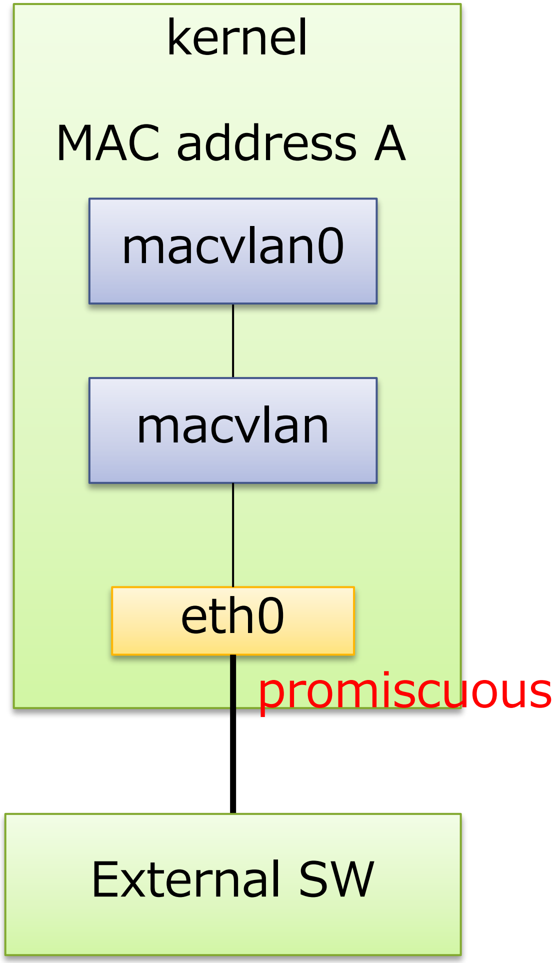
macvlan (bridge mode)

- **Light weight bridge**
  - No source learning
  - No STP
  - Only one uplink
- **Allow traffic between macvlans (via macvlan stack)**



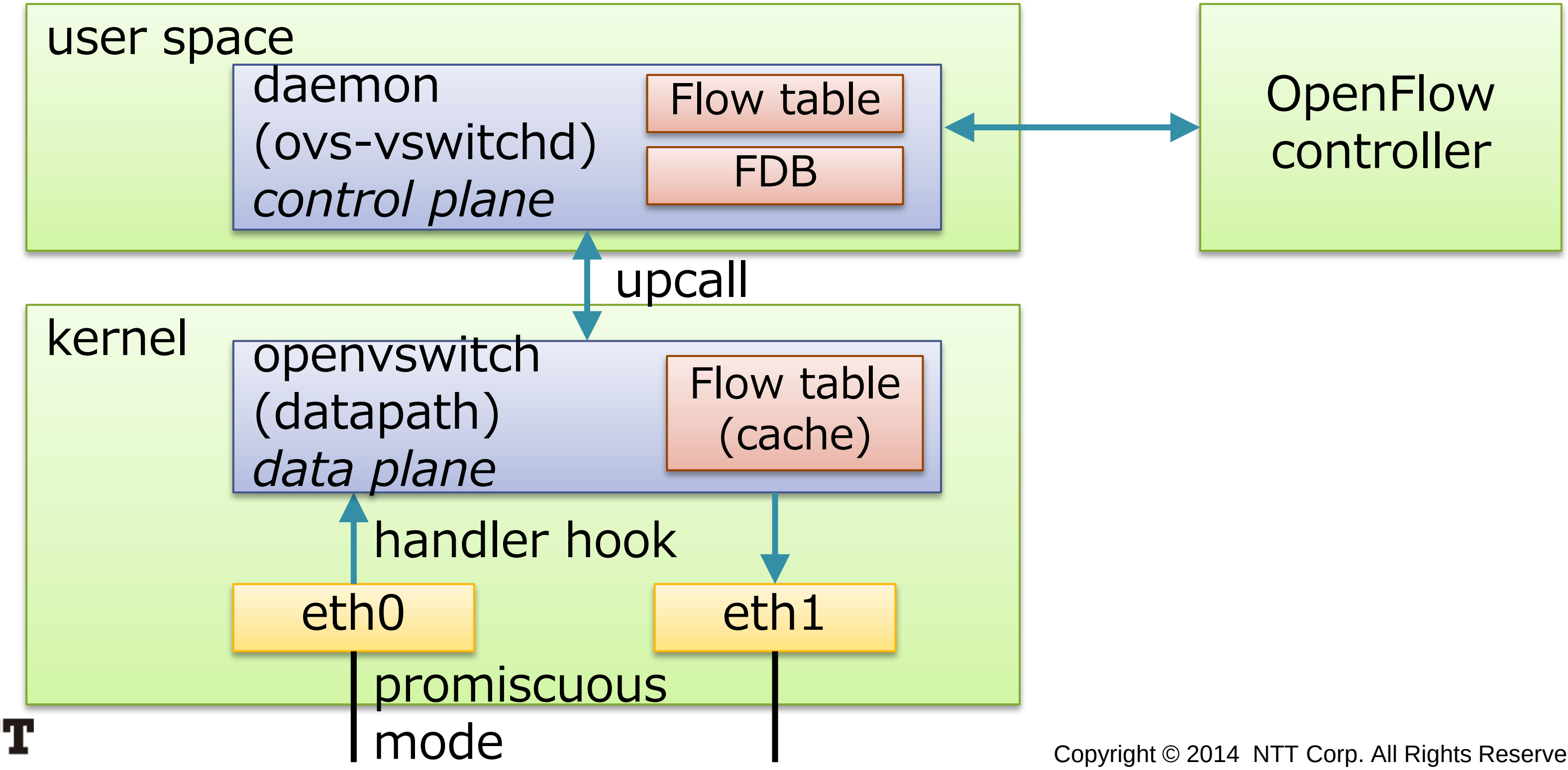
macvlan (passthru mode)

- **Allow only one macvlan device**
- **Used for VM (as macvtap)**
- **Promiscuous**
  - allow VM to use any mac address / vlan device



Open vSwitch

- **Supports OpenFlow**
- **Can be used as a normal switch as well**
  - Has many features (VLAN tagging, VXLAN, GRE, bonding, etc.)
- **Flow based forwarding**
- **Control plane in user space**
  - flow miss-hit causes upcall to userspace daemon



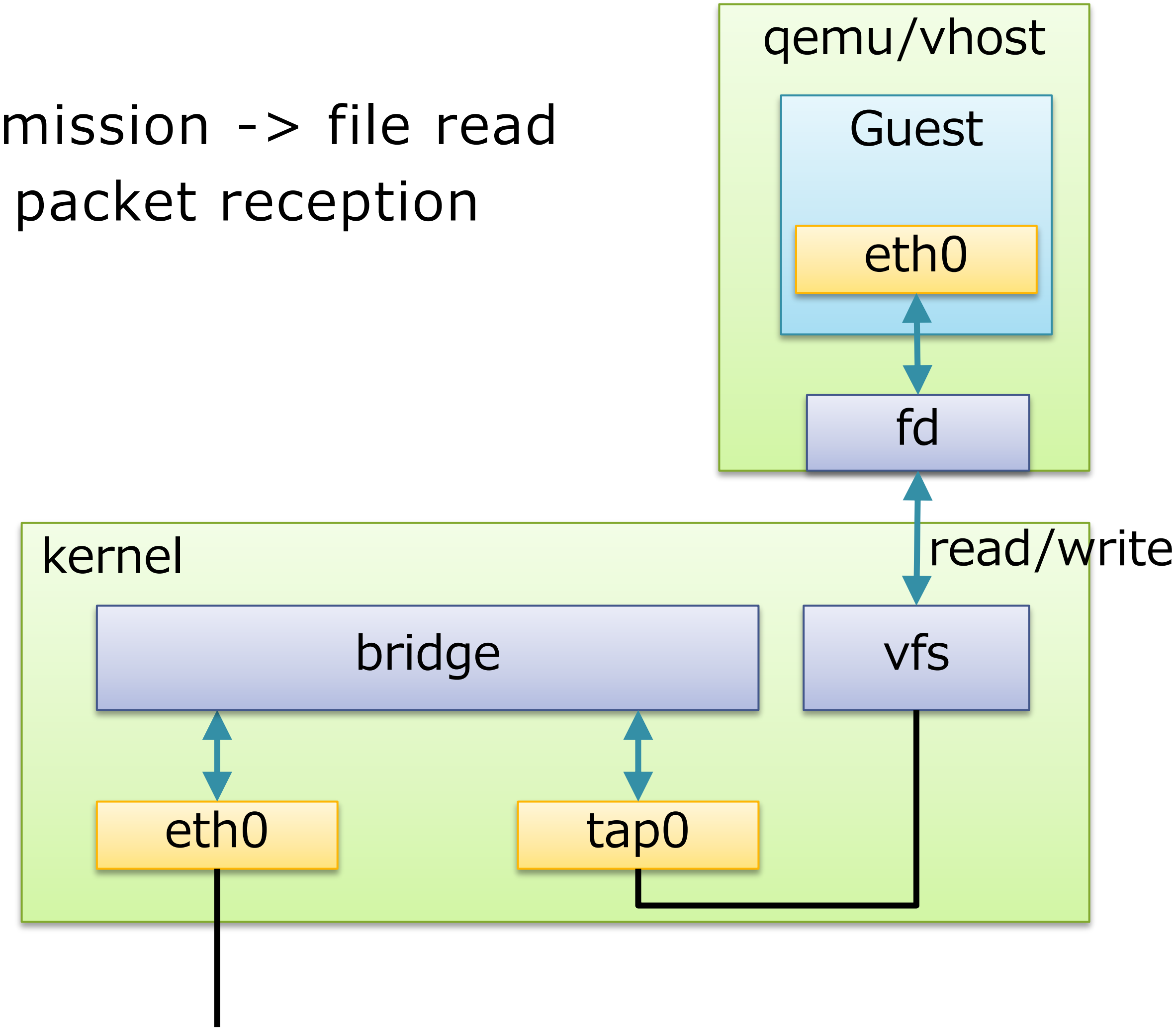
Switching technologies for KVM

- **Software switches**
  - bridge
  - macvlan
  - Open vSwitch

- **Hardware switch**
  - NIC embedded switch (in SR-IOV device)

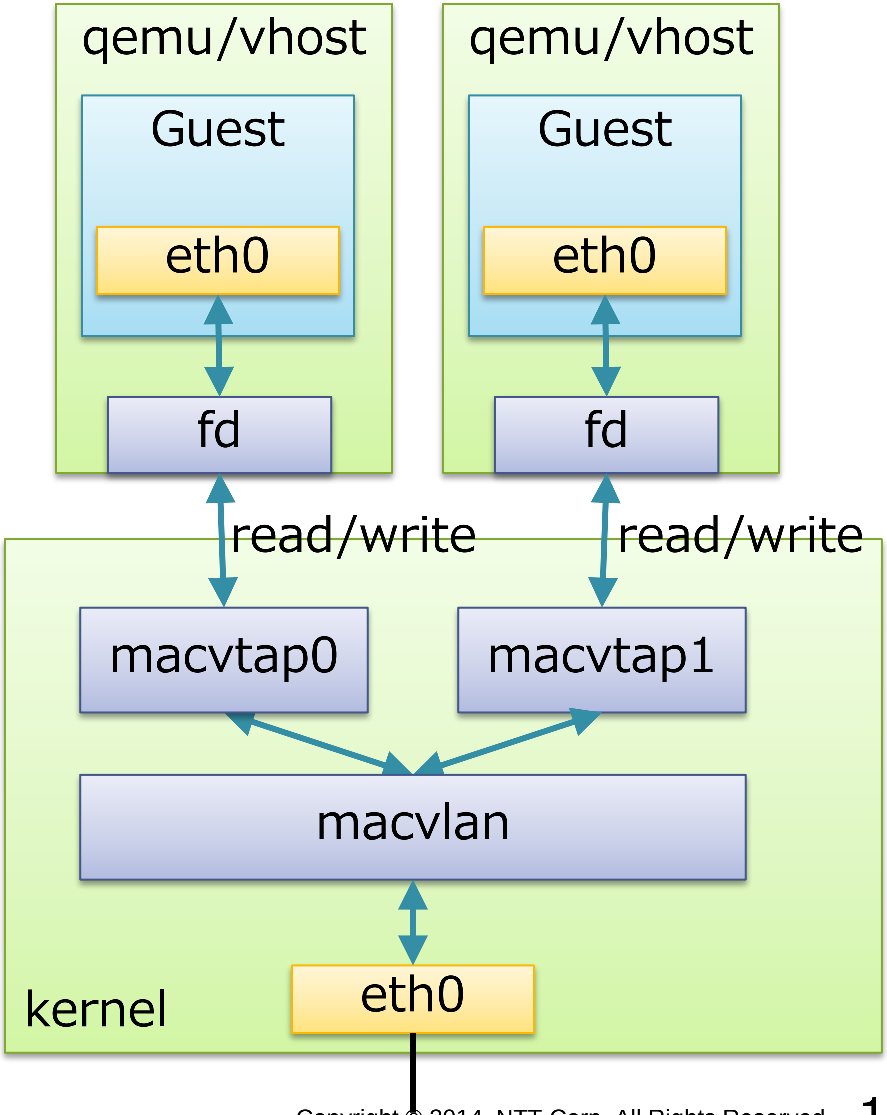
bridge with KVM

- **Used with tap device**
- **Tap device**
  - packet transmission -> file read
  - file write -> packet reception



macvtap (private, vepa, bridge) with KVM

- **macvtap**
  - tap-like macvlan variant
  - packet reception
    -> file read
  - file write
    -> packet transmission

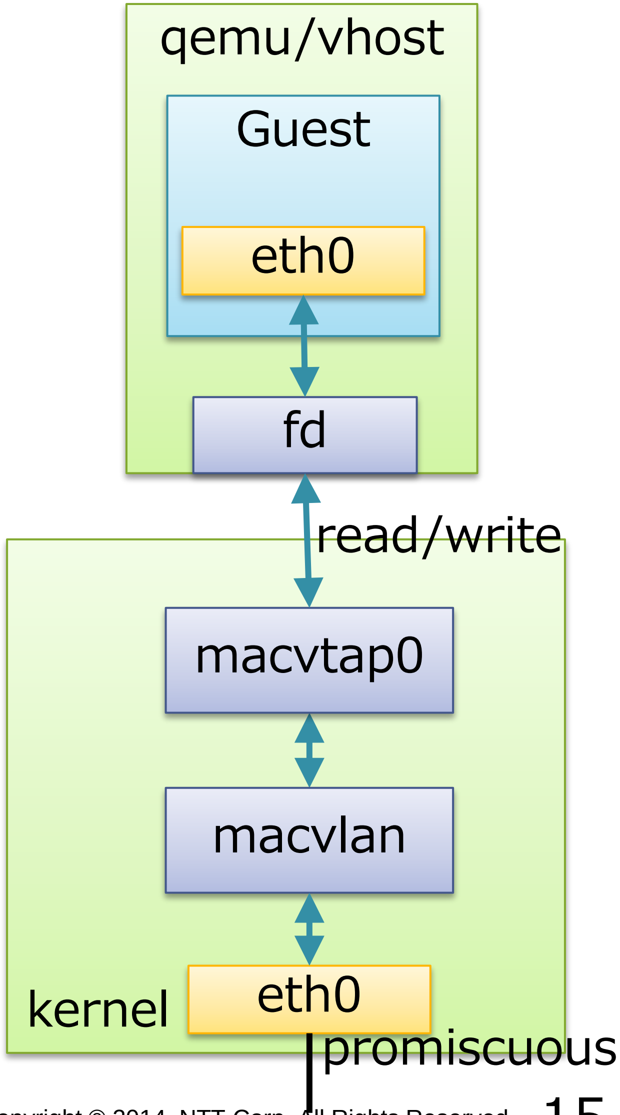


Copyright © 2014 NTT Corp. All Rights Reserved.

macvtap (passthru) with KVM

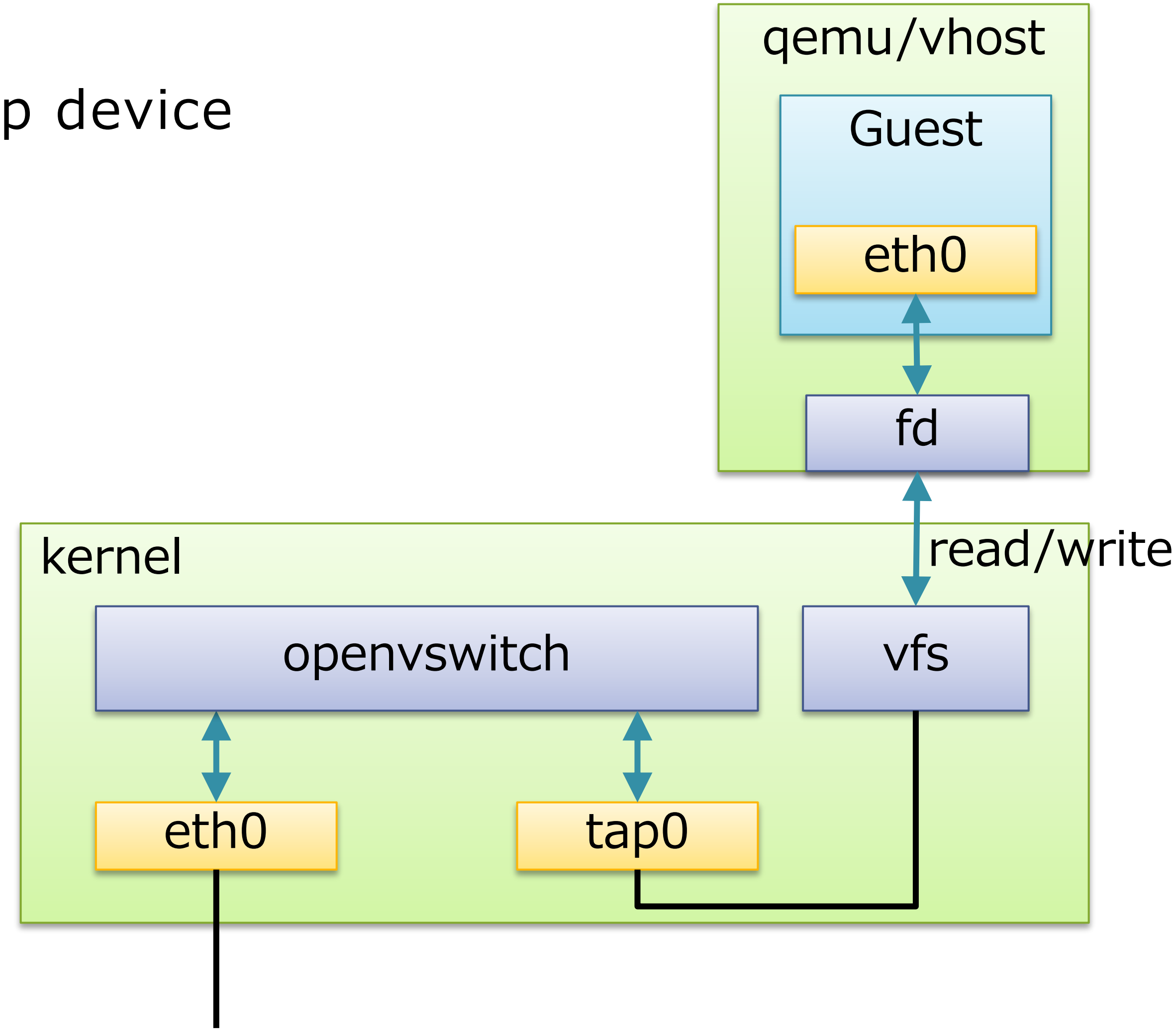
- **macvtap passthru mode**
  - PCI-passthrough like mode
  - Guest can exclusively use physical device
  - Guest can use any mac address / vlan interface
  - Guest can use promiscuous mode

  - Other modes uses unicast filtering
    - Don't allow to receive mac address except for macvtap device's
    - Don't allow vlan tagged packets if NIC has vlan filtering feature



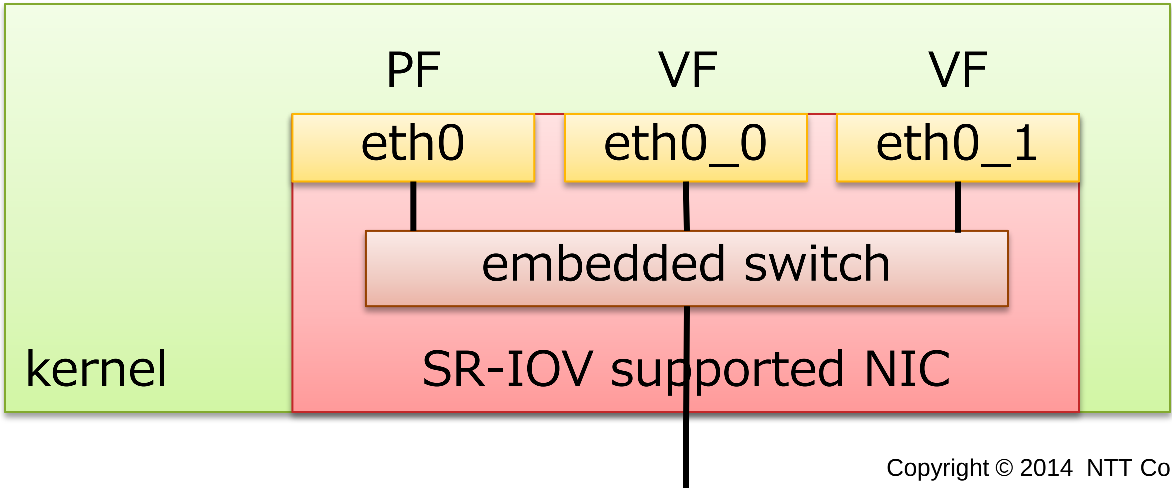
Open vSwitch with KVM

- **Configuration is the same as bridge**
  - used with tap device



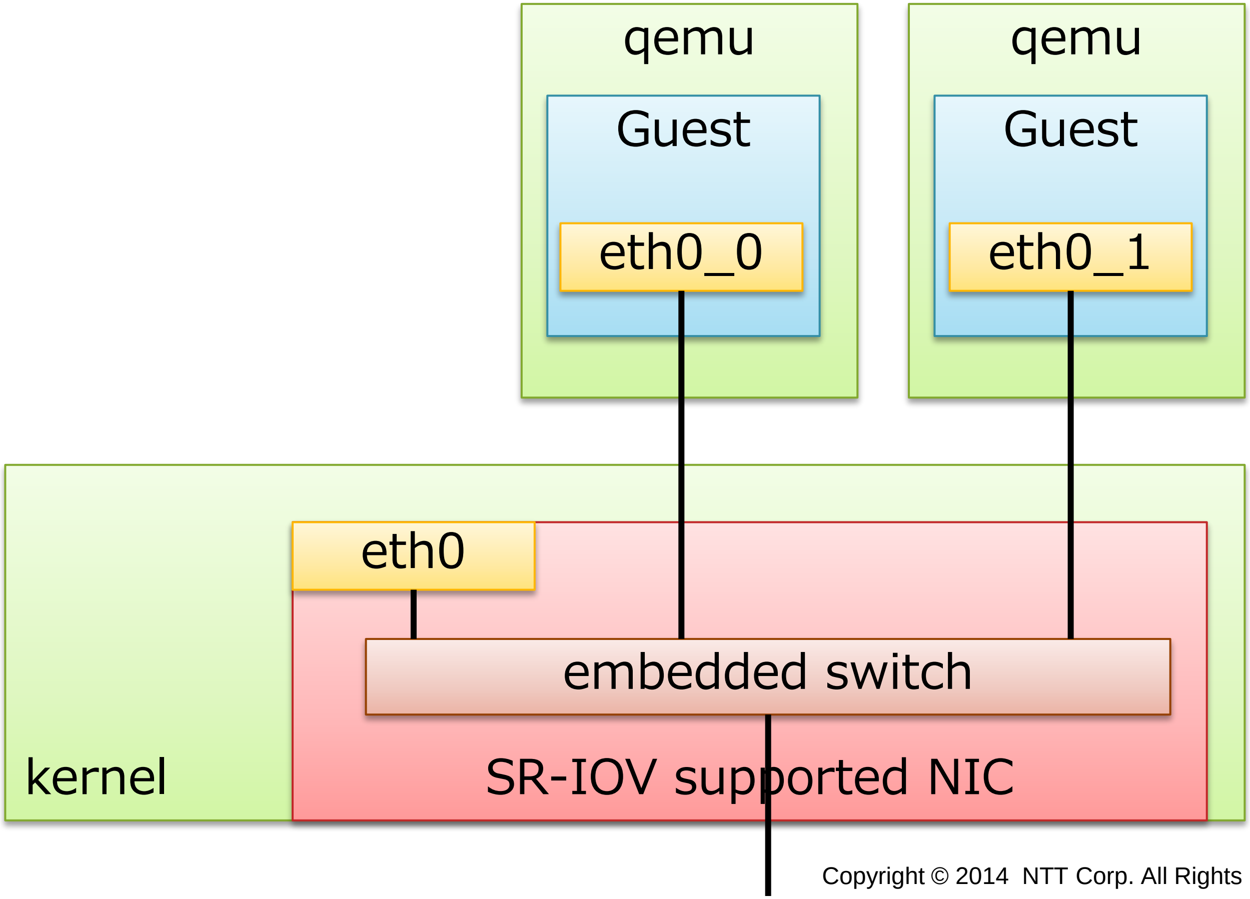
NIC embedded switch (SR-IOV)

- **SR-IOV**
  - Addition to PCI normal physical function (PF), allow to add light weight virtual functions (VF)
  - VF appears as a network interface (eth0_0, eth0_1...)
  - Some SR-IOV devices have switches in them
    - allow PF-VF / VF-VF communication



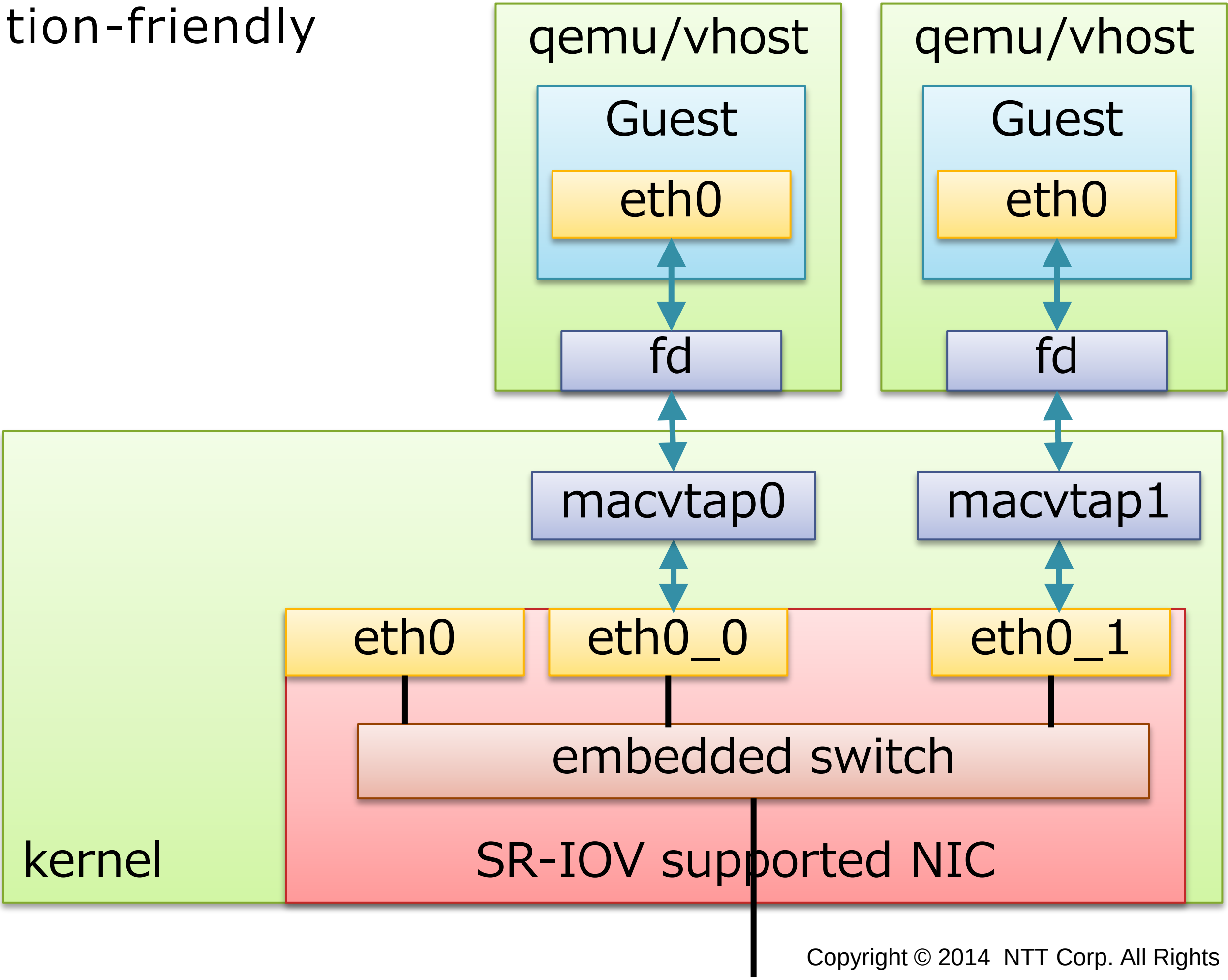
NIC embedded switch (SR-IOV)

- **SR-IOV with KVM**
  - Use PCI-passthrough to attach VF to guest



NIC embedded switch (SR-IOV)

- **SR-IOV with KVM**
  - Or use macvtap (passthru)
    - migration-friendly

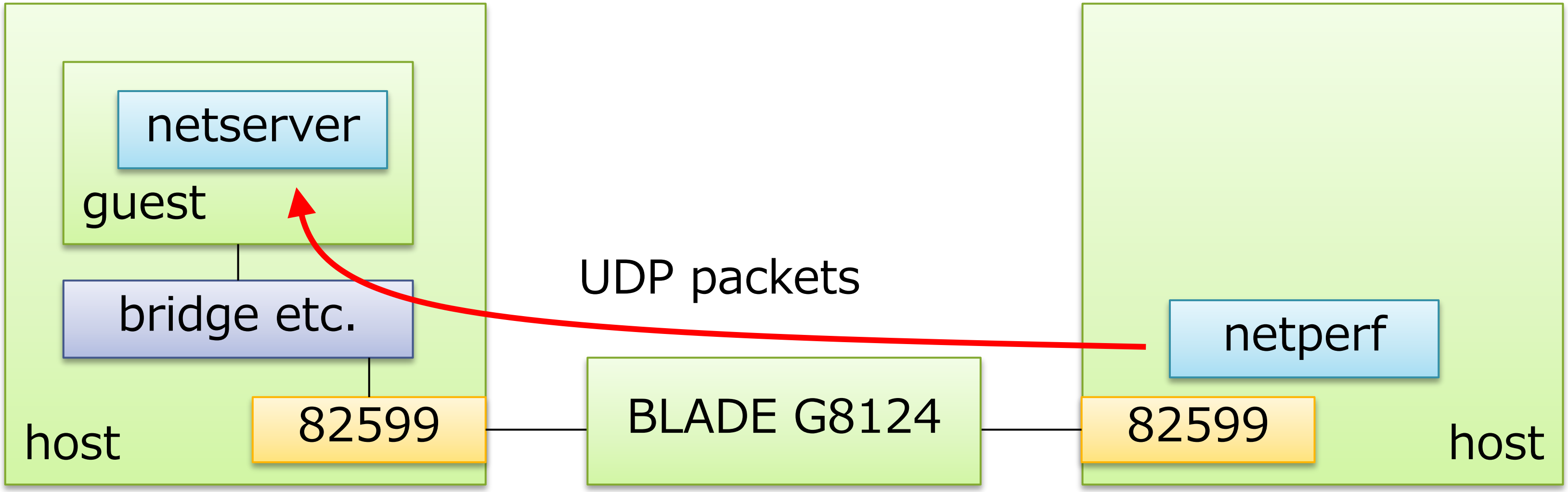


Performance of switches

- **Environment**
- **Test results**
  - Throughput
  - Overhead on host

Performance: environment

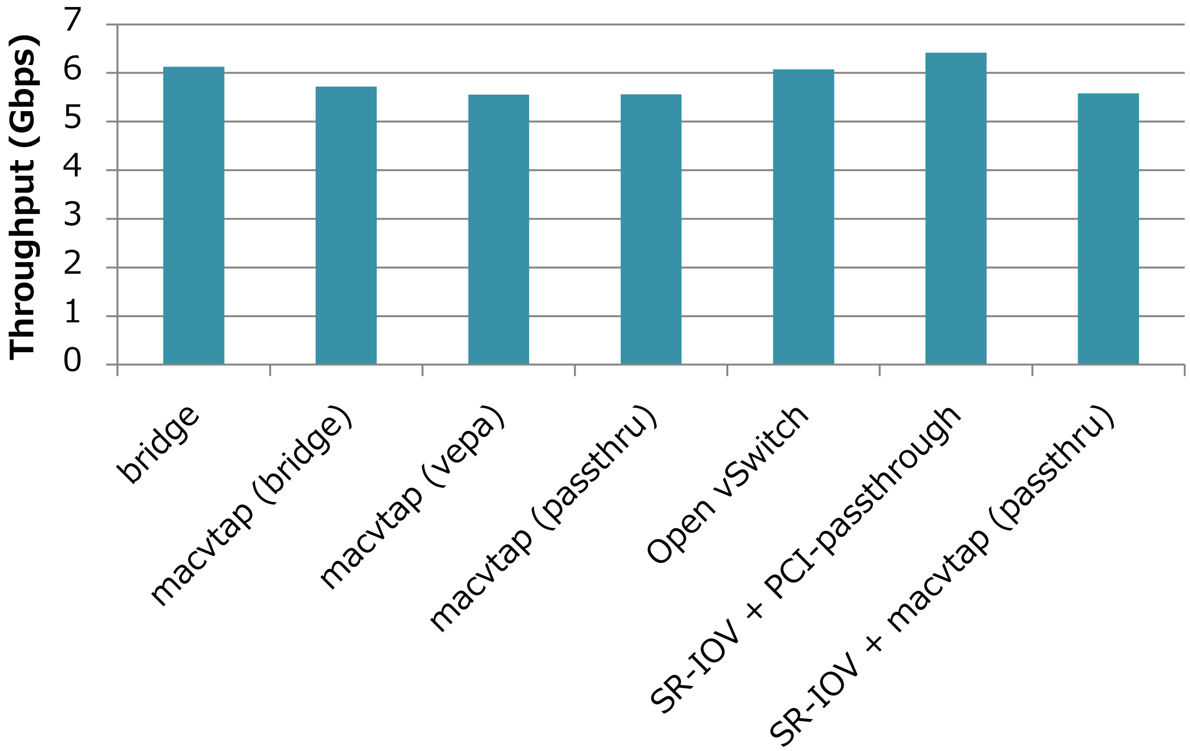
- **kernel 3.14.4 (2014/5/13 Release)**
- **Host: Xeon E5-2407 4 core * 2 socket**
- **NIC: 10GbE, Intel 82599 chip (ixgbe)**
- **Guest: 2 core*1**
- **HW Switch: BLADE G8124**
- **Benchmark tool: netperf-2.6**
  - UDP_STREAM test (1518 byte frame length)



*1: Pinning on host: vcpus -> CPU0~3, vhost -> CPU1. NIC irq affinity on host: 0x1 (CPU0).
Pinning on guest: netserver process -> CPU1. NIC irq affinity on guest: 0x1 (CPU0).

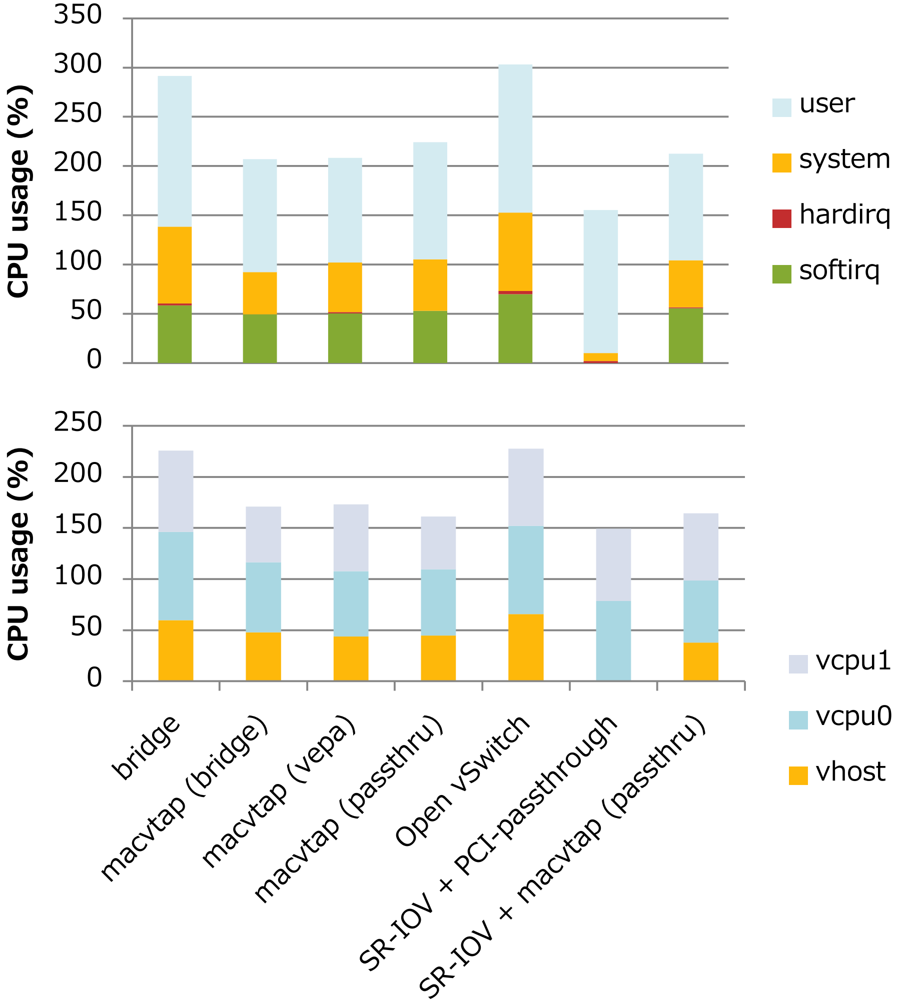
Performance: throughput

- **Receive throughput on guest**
  - SR-IOV (PCI-passthrough) has the highest-performance
  - Software switches are 6%~14% worse than SR-IOV (PCI-passthrough)



Performance: Overhead on host

- **Overhead (CPU usage) on host**
  - SR-IOV (PCI-passthrough) has the lowest overhead
    - CPU usage by system and irqs are close to 0
  - CPU usage by macvtap is 24~29% lower than bridge / Open vSwitch



Userland APIs and commands (bridge)

- **Various APIs**
  - ioctl
  - sysfs
  - netlink

- **Netlink is preferred for new features**
  - Because it is extensible
  - sysfs is sometimes used

- **Commands**
  - brctl (in bridge-utils, using ioctl / sysfs)
  - ip / bridge (in iproute2, using netlink)

Userland APIs and commands (bridge)

- **brctl**

```
# brctl addbr <bridge>                ... create new bridge
# brctl addif <bridge> <port>         ... attach port to bridge
# brctl showmacs <bridge>             ... show fdb entries
```

- **These operations are now realized by netlink based commands as well (Since kernel 3.0)**

```
# ip link add <bridge> type bridge    ... create new bridge
# ip link set <port> master <bridge> ... attach port
# bridge fdb show                     ... show fdb entries
```

- **And recent features can only be used by netlink based ones or direct sysfs write**

```
# bridge fdb add
# bridge vlan add
etc...
```

Recent features of bridge (and others)

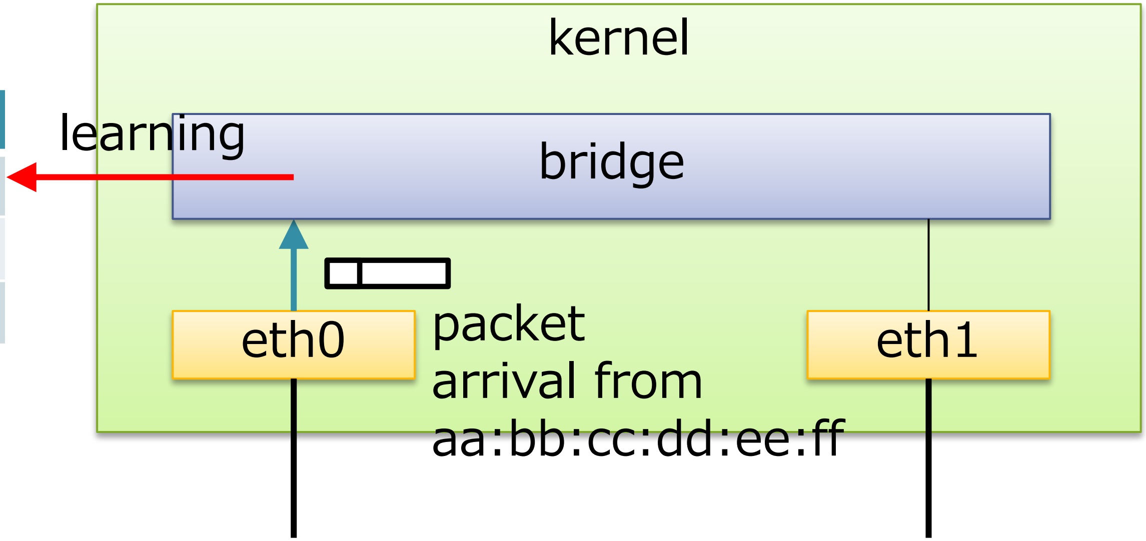
- FDB manipulation
- VLAN filtering
- Learning / flooding control

FDB manipulation

- **FDB**
  - Forwarding database
  - Learning: packet arrival triggers entry creation
    - Source MAC address is used with incoming port
  - Flood if failed to find entry
    - Flood: deliver packet to all ports but incoming one

FDB

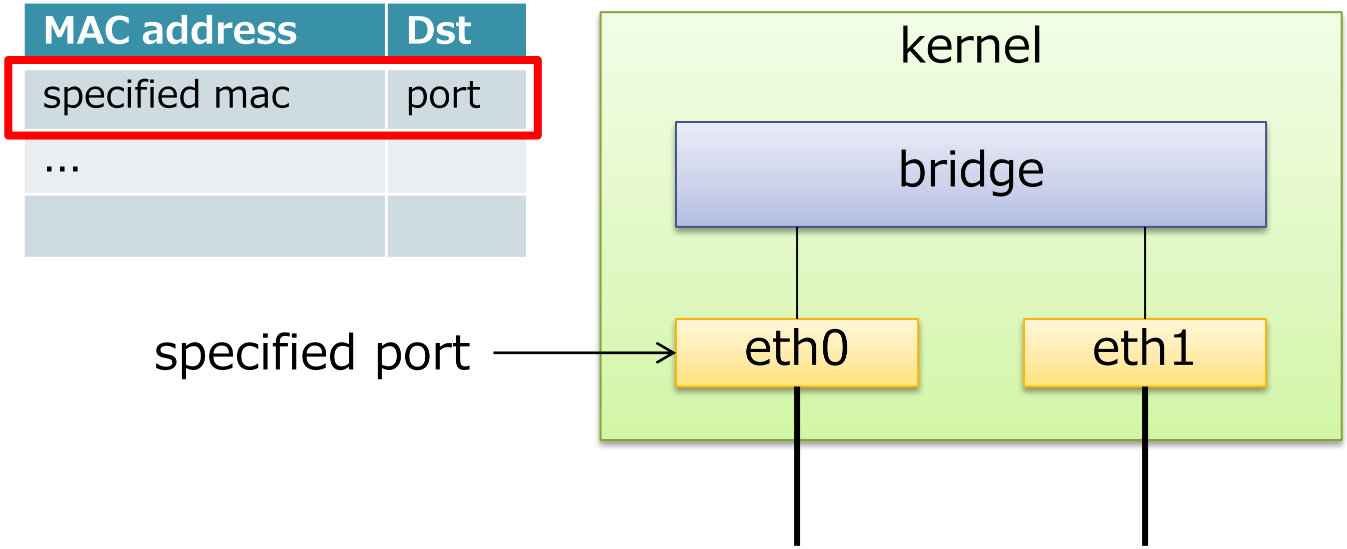
MAC address	Dst
aa:bb:cc:dd:ee:ff	eth0
…	



FDB manipulation

- **FDB manipulation commands**
  - Since kernel 3.0

```
# bridge fdb add <mac address> dev <port> master temp
# bridge fdb del <mac address> dev <port> master
```

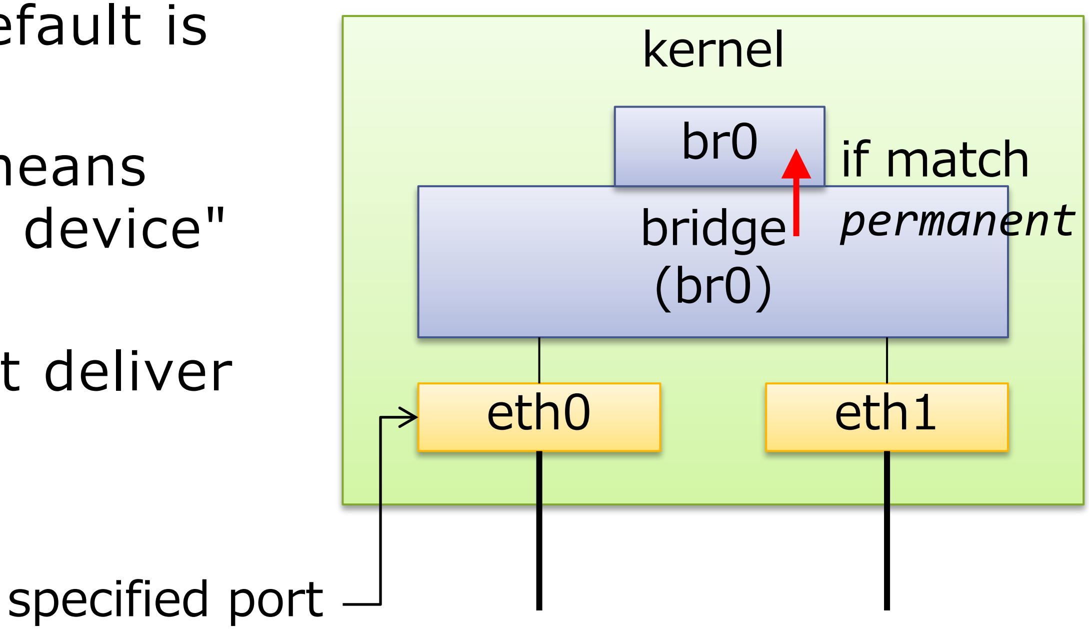


MAC address	Dst
specified mac	port
…	

FDB manipulation

```
# bridge fdb add <mac address> dev <port> master temp
```

- **What's "temp"?**
  - There are 3 types of FDB entries
    - *permanent* (*local*)
    - *static*
    - others (dynamically learned by packet arrival)
  - "temp" means *static* here
  - "bridge fdb"'s default is *permanent*
  - *permanent* here means "deliver to bridge device" (e.g. br0)
  - *permanent* doesn't deliver to specified port

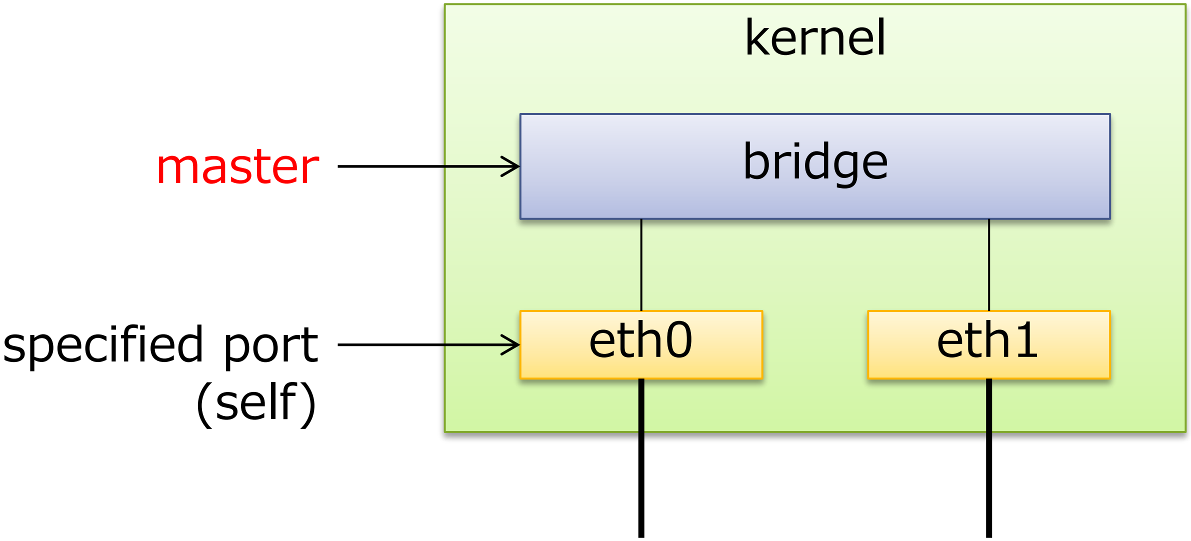


FDB manipulation

```
# bridge fdb add <mac address> dev <port> master temp
```

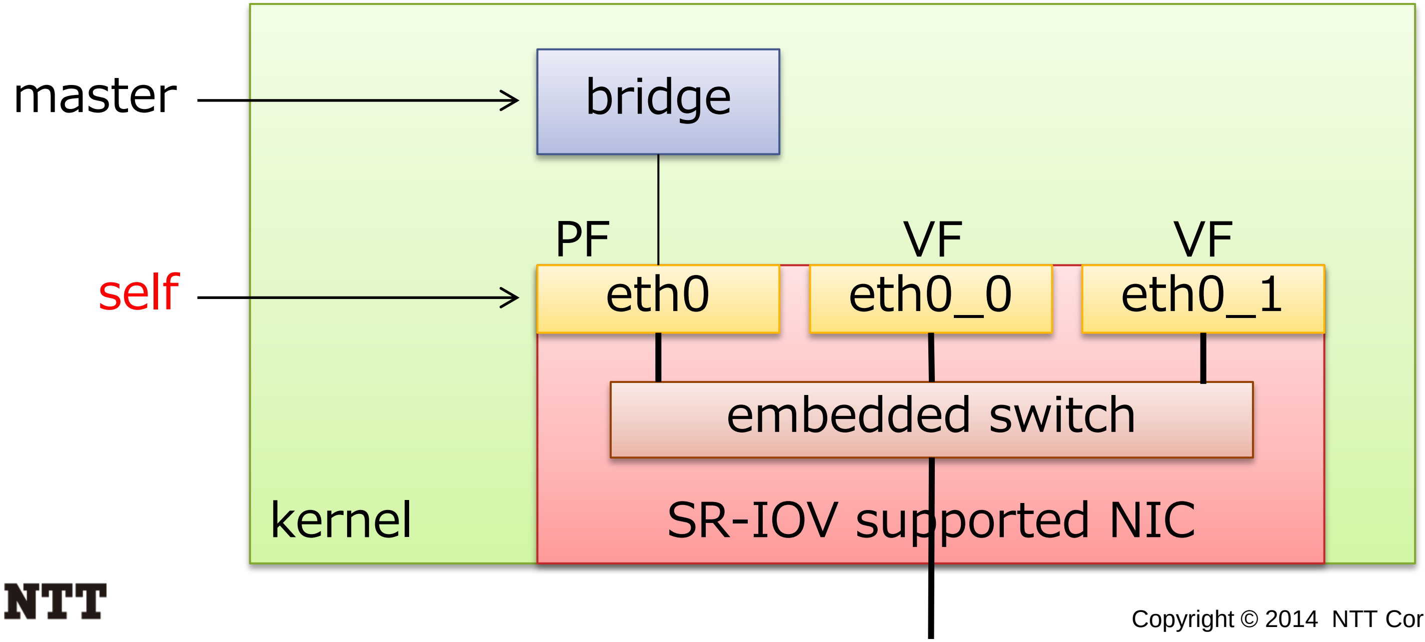
- **What's "master"?**
  - Remember this command

    `# ip link set <port> master <bridge>` ... attach port
  - "`bridge fdb`"'s default is "`self`"
    - It adds entry to specified port (eth0) itself!



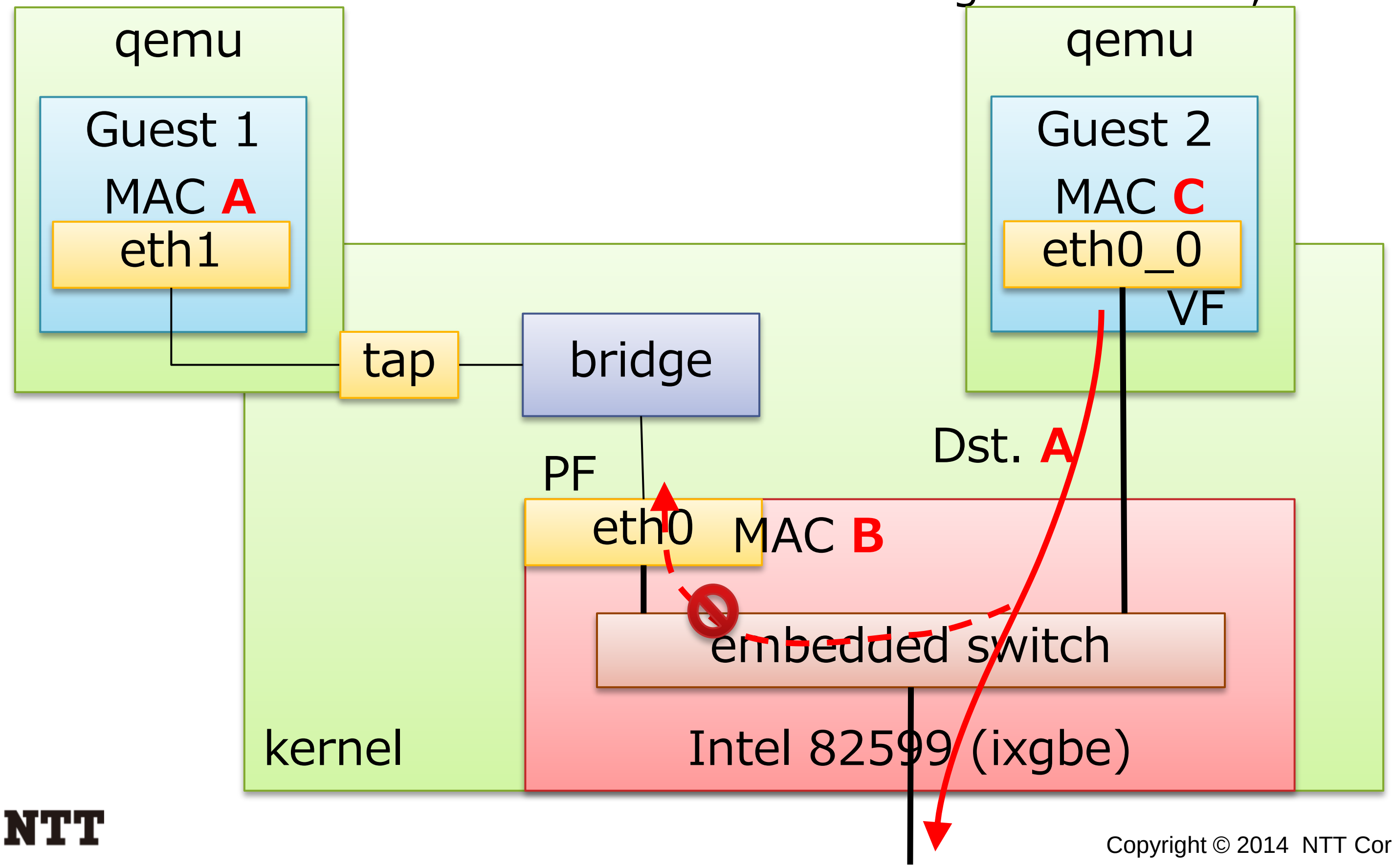
FDB manipulation

- **When to use "`self`"?**
  - Some NIC embedded switches support this command
    - ixgbe, qlcnic
  - macvlan (passthru) and vxlan also support it



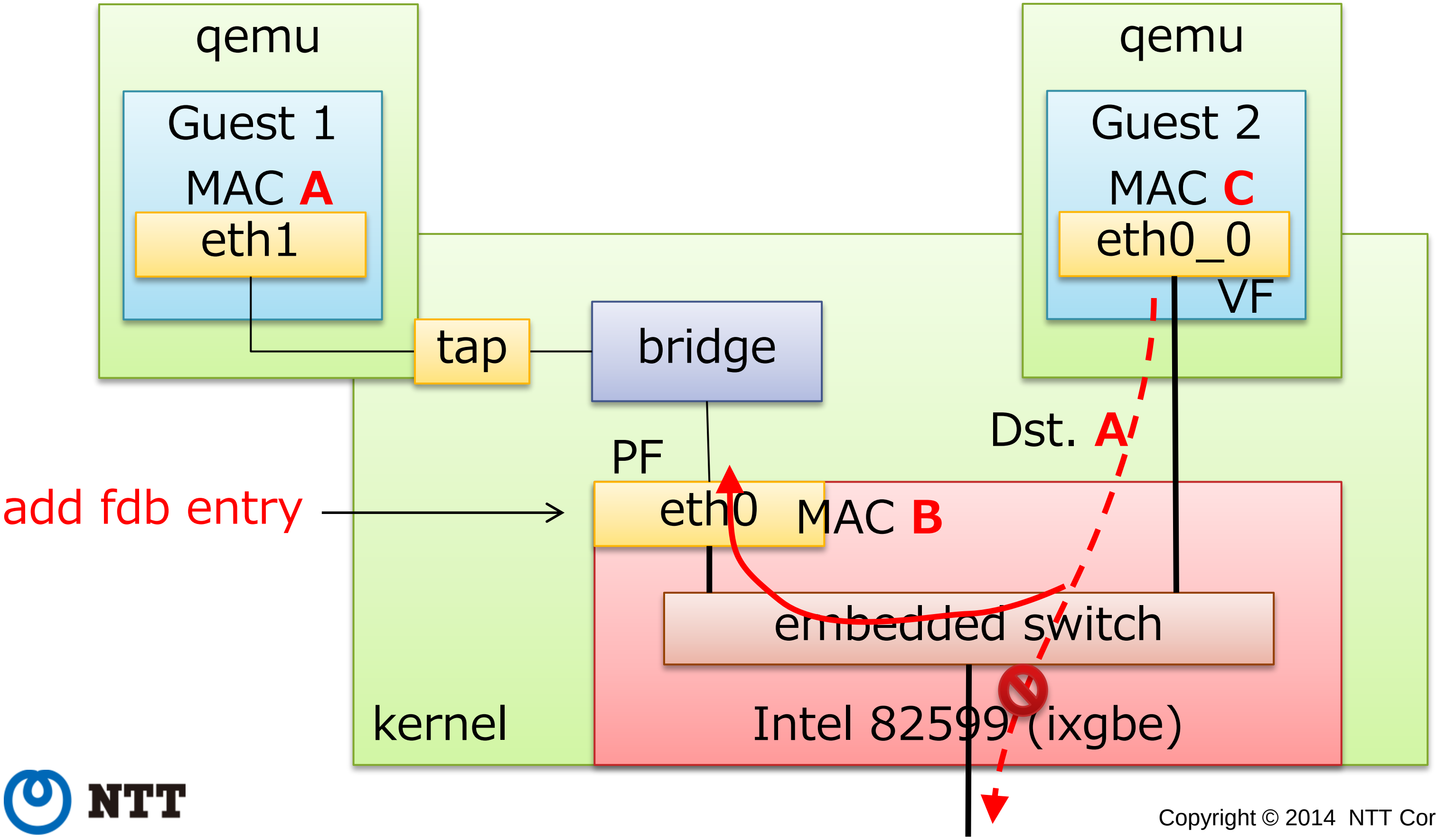
FDB manipulation

- **Example: Intel 82599 (ixgbe)**
  - Someone thinks of using both bridge and SR-IOV due to limitation of number of VFs
  - bridge puts eth0 (PF) into promiscuous, but...
    - Unknown MAC address **from VF** goes to wire, not to PF



FDB manipulation

- **Example: Intel 82599 (ixgbe)**
  - Type "`bridge fdb add A dev eth0`" on host
  - Traffic to A will be forwarded to bridge

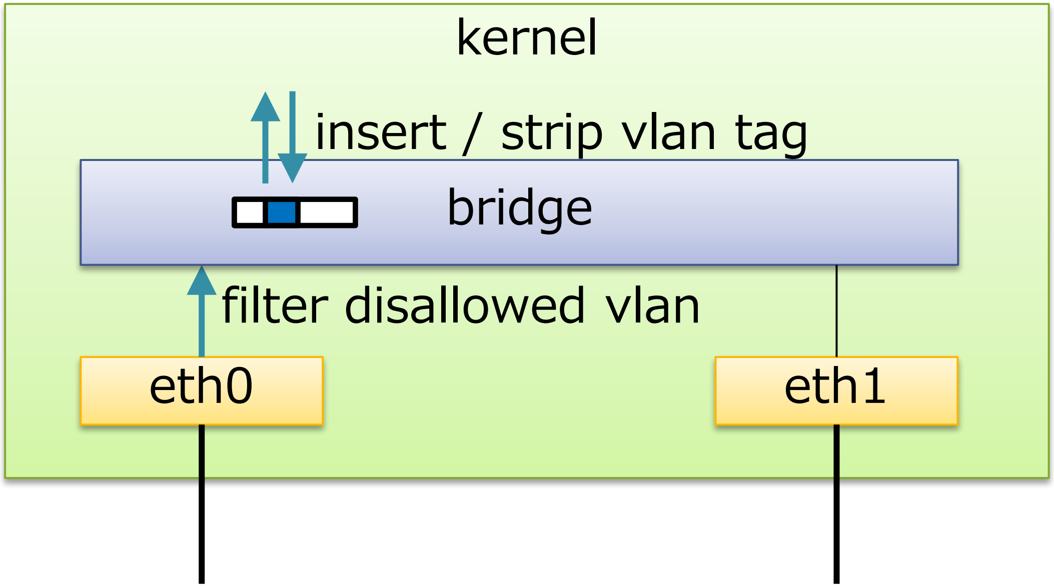


VLAN filtering

- **802.1Q Bridge**
  - Filter packets according to vlan tag
  - Forward packets according to vlan tag as well as mac address
  - Insert / strip vlan tag

FDB

MAC address	Vlan	Dst
aa:bb:cc:dd:ee:ff	10	eth0
…		

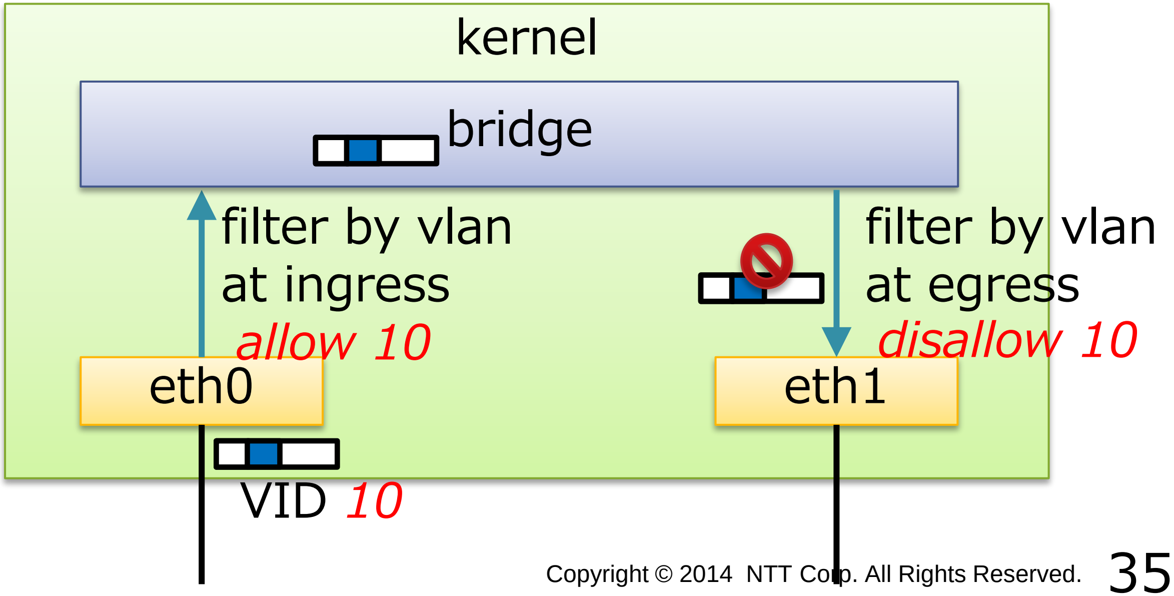


VLAN filtering

- **Ingress / egress filtering policy**
  - Incoming / outgoing packet is filtered if matching filtering policy
  - Per-port per-vlan policy
  - Default is "disallow all vlans"
    - All packets are dropped

Filtering table

Port	Allowed Vlans
eth0	10
	20
eth1	20
	30

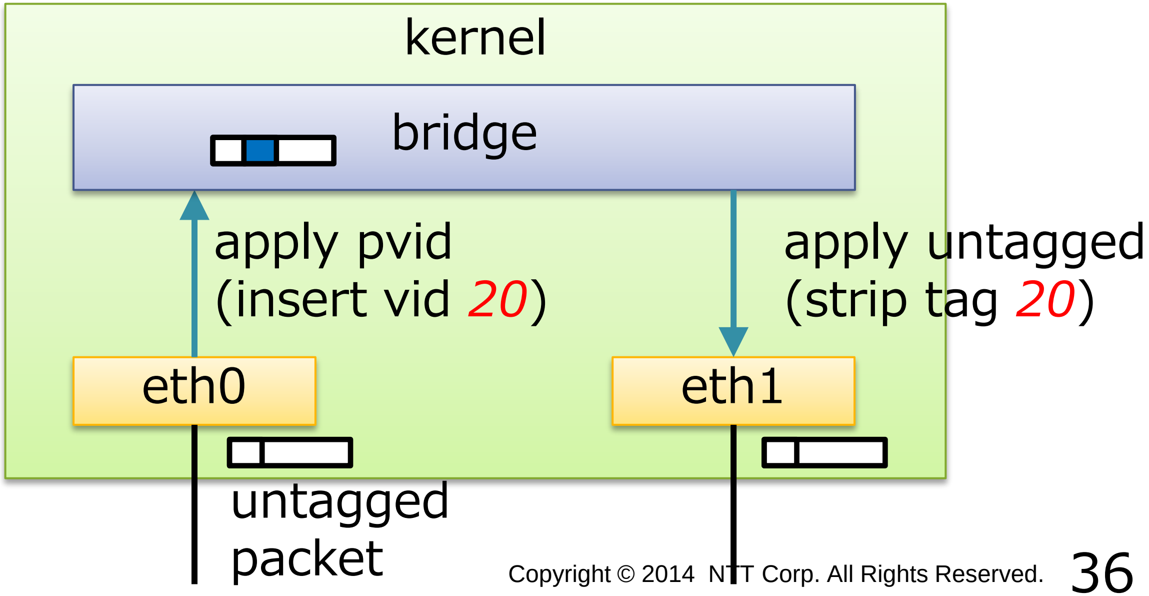


VLAN filtering

- **PVID (Port VID)**
  - Untagged (and VID 0) packet is assigned this VID
  - Per-port configuration
  - Default PVID is none (untagged packet is discarded)
- **Egress policy untagged**
  - Outgoing packet that matches this policy get untagged
  - Per-port per-vlan policy

Filtering table

Port	Allowed Vlans	PVID	Egress Untag
eth0	10		✓
	20	✓	✓
eth1	20	✓	✓
	30		



VLAN filtering

- **Commands**
  - Enable VLAN filtering (disabled by default)

```
# echo 1 > /sys/class/net/<bridge>/bridge/vlan_filtering
```

  - Add / delete allowed vlan

```
# bridge vlan add vid <vlan_id> dev <port>
# bridge vlan del vid <vlan_id> dev <port>
```

  - Set pvid / untagged

```
# bridge vlan add vid <vlan_id> dev <port> [pvid] [untagged]
```

  - Dump setting

```
# bridge vlan show
```

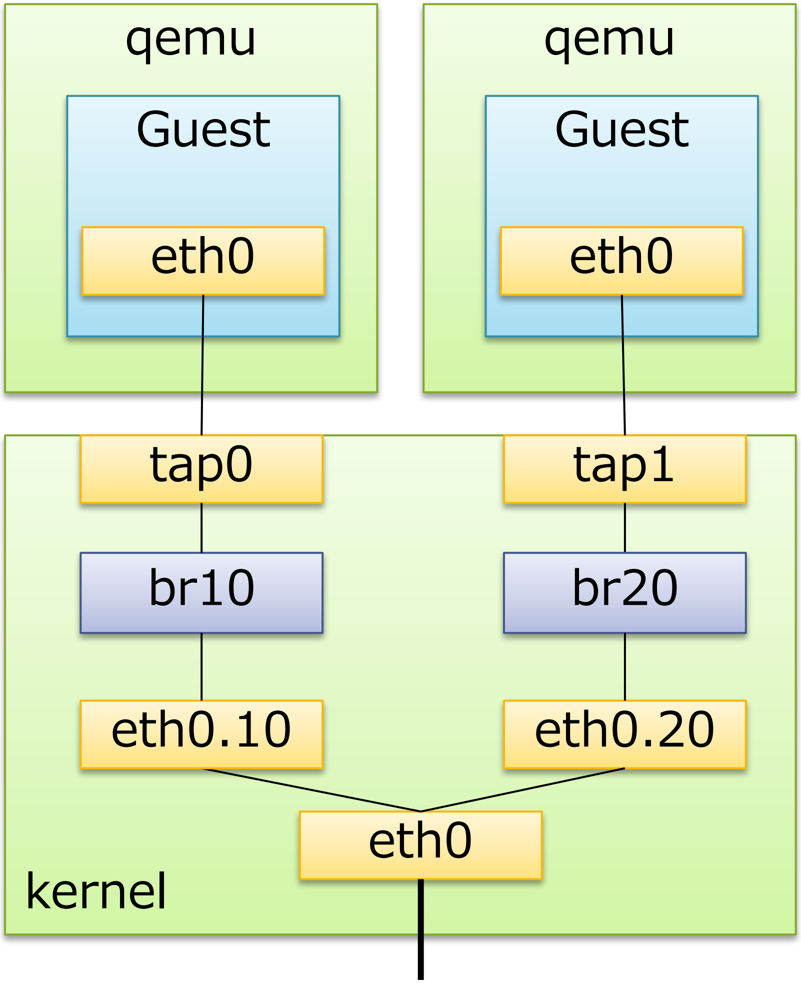
- **Note: bridge device needs "self"**

```
# bridge vlan add vid <vlan_id> dev br0 self
# bridge vlan del vid <vlan_id> dev br0 self
```

VLAN with KVM

- **Traditional configuration**
  - Use vlan devices
  - Needs bridges per vlan
  - Low flexibility
  - How many devices?

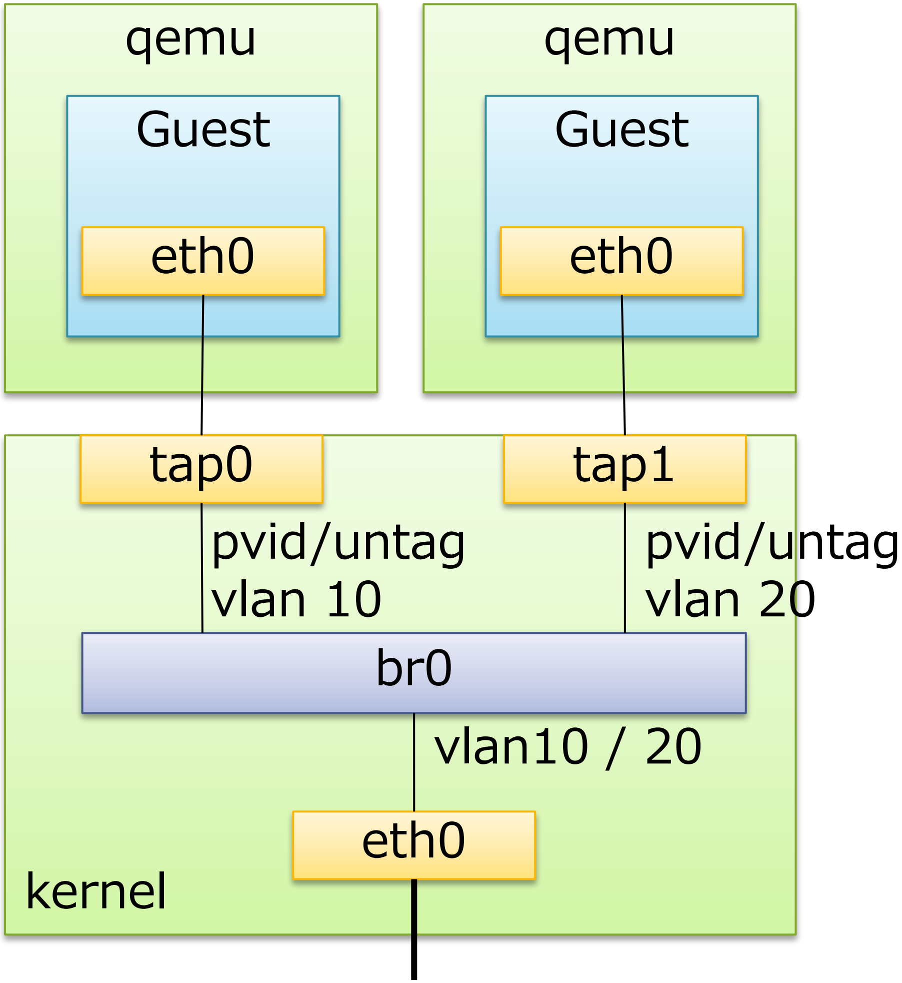
```
# ifconfig -s
Iface ...
eth0
eth0.10
br10
eth0.20
br20
eth0.30
br30
eth0.40
br40
...
```



VLAN with KVM

- **With VLAN filtering**
  - Simple
  - Flexible
  - Only one bridge

```
# ifconfig -s
Iface ...
eth0
br0
```



VLAN with KVM

- **Other switches**
  - Open vSwitch
    - Can also handle VLANs

```
# ovs-vsctl set Port <port> tag=<vid>
```

  - NIC embedded switch
    - Some of them support VLAN (e.g. Intel 82599)

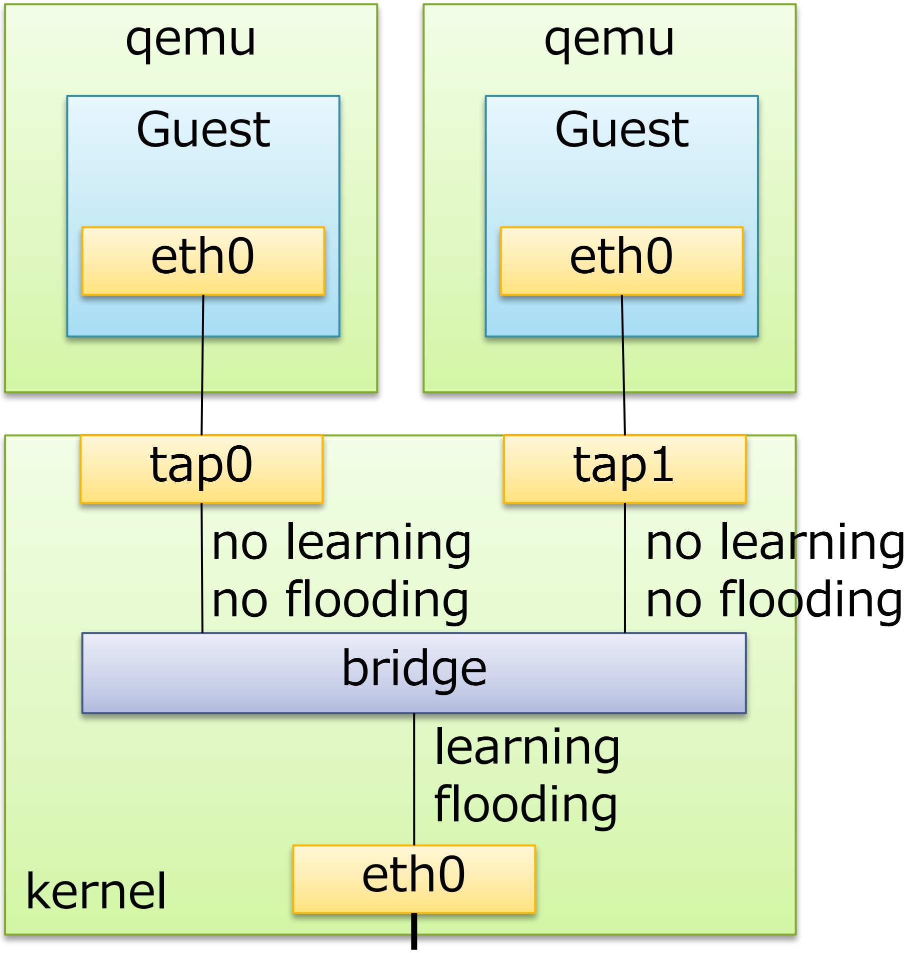
```
# ip link set <PF> vf <VF_num> vlan <vid>
```

Learning / flooding control

- **Limit mac addresses guest can use**
- **Reduce FDB size**
- **Used with static FDB entries ("`bridge fdb`" command)**

- **Disable FDB learning on particular port**
  - Since kernel 3.11
  - No dynamic FDB entry

- **Don't flood unknown mac to specified port**
  - Since kernel 3.11
  - Control packet delivery to guests



- **Commands**

```
# echo 0 > /sys/class/net/<port>/brport/learning
# echo 0 > /sys/class/net/<port>/brport/unicast_flooding
```

Features under development

- 802.1ad (Q-in-Q) support for bridge
- Non-promiscuous bridge

802.1ad (Q-in-Q) support for bridge

- **802.1ad allows stacked vlan tags**

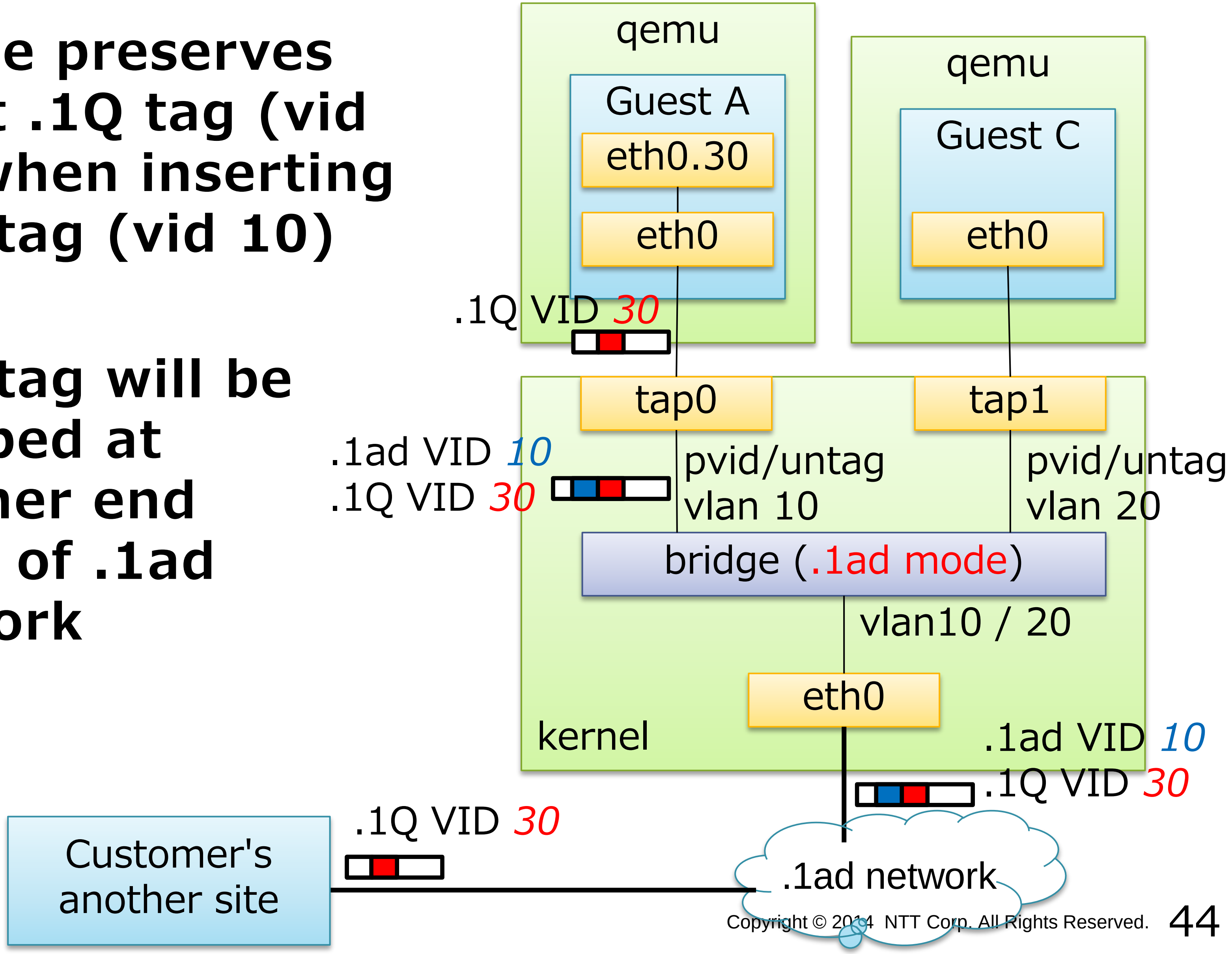
MAC	.1ad tag	.1Q tag	payload

- **Outer 802.1ad tag can be used to separate customers**
  - Example: Guest A, B -> Customer X
    Guest C, D -> Customer Y

- **Inner 802.1Q tag can be used inside customers**
  - Customer X and Y can use any 802.1Q tags

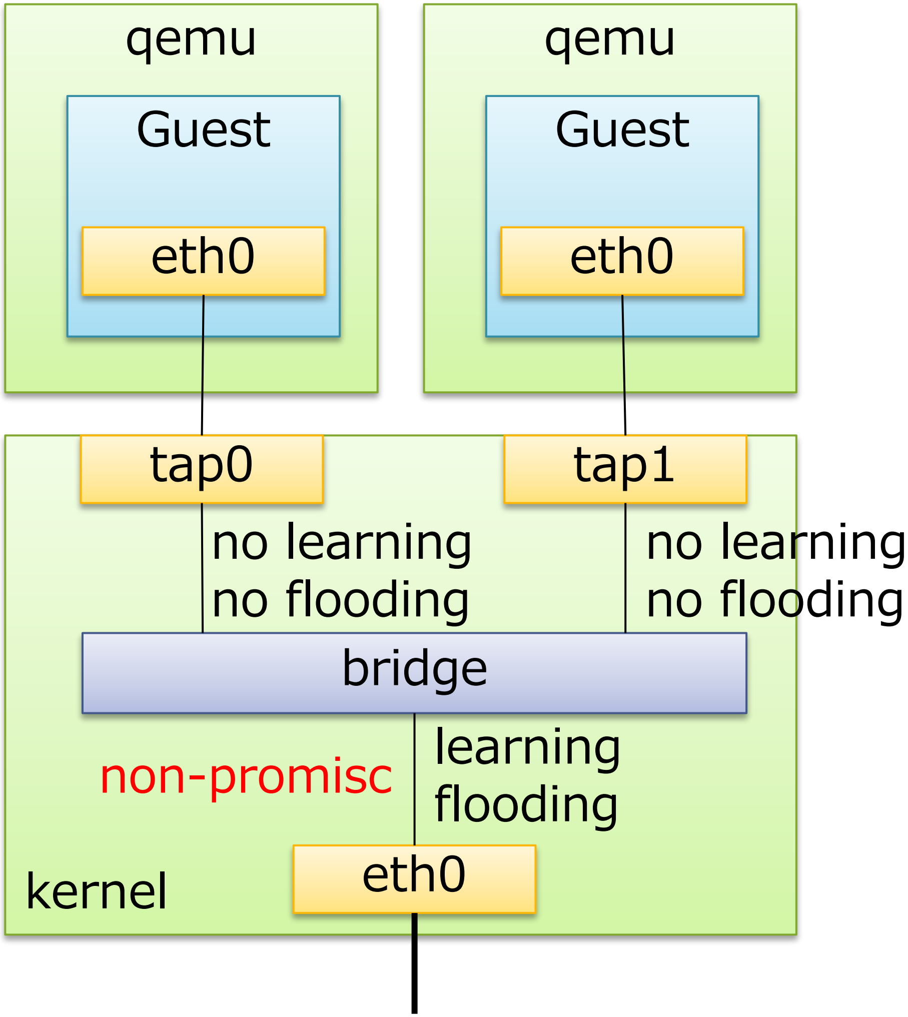
802.1ad (Q-in-Q) support for bridge

- Bridge preserves guest .1Q tag (vid 30) when inserting .1ad tag (vid 10)
- .1ad tag will be stripped at another end point of .1ad network



Non-promiscuous bridge

- **If there is only one learning/flooding port, it can be non-promisc**
- **Instead of promisc mode, unicast filtering is set for static FDB entries**
- **Automatically enabled if meeting some conditions**
  - There is one or zero learning & flooding port
  - bridge itself is not promiscuous mode
  - VLAN filtering is enabled
- **Overhead will get closer to macvlans**



Summary

- **Linux has 3 types of software switches**
  - bridge, macvlan (macvtap), Open vSwitch
  - SR-IOV NIC enbedded switch can also be used for KVM
- **Bridge's recent features**
  - FDB manipulation
  - VLAN filtering
  - Learning / Flooding control
- **Features under development**
  - 802.1ad (Q-in-Q) support
  - Non-promiscuous bridge

Thank you for listening.
Any questions?