istic Informatique Électronique

Architecture Réseaux Entreprises (ARES)

Virtualisation des réseaux avec Libvirt et Open vSwitch

Brice - Ekane (brice.ekane@univ-rennes.fr)

ISTIC Rennes - France

2025-2026

git clone https://github.com/bekane/ares-2025.git

Plan du module

Apperçu de la section 1

Pourquoi virtualiser le réseau ?

- **Agilité** : créer/modifier des topologies en minutes.
- **Isolation** : environnements sûrs et reproductibles.
- **Coûts** : moins de dépendance au hardware.
- **Échelle** : multi-hôtes, overlays, automation.

Objectifs pédagogiques

- **Libvirt** : réseaux NAT, bridge, isolés
- **OVS** : bridges, ports, VLAN, VXLAN, mirroring.
- **Intégration** Libvirt<->OVS (prod-like).
- **Pratique** : interconnecter des VMs de manière *flexible et sûre*.

Plan

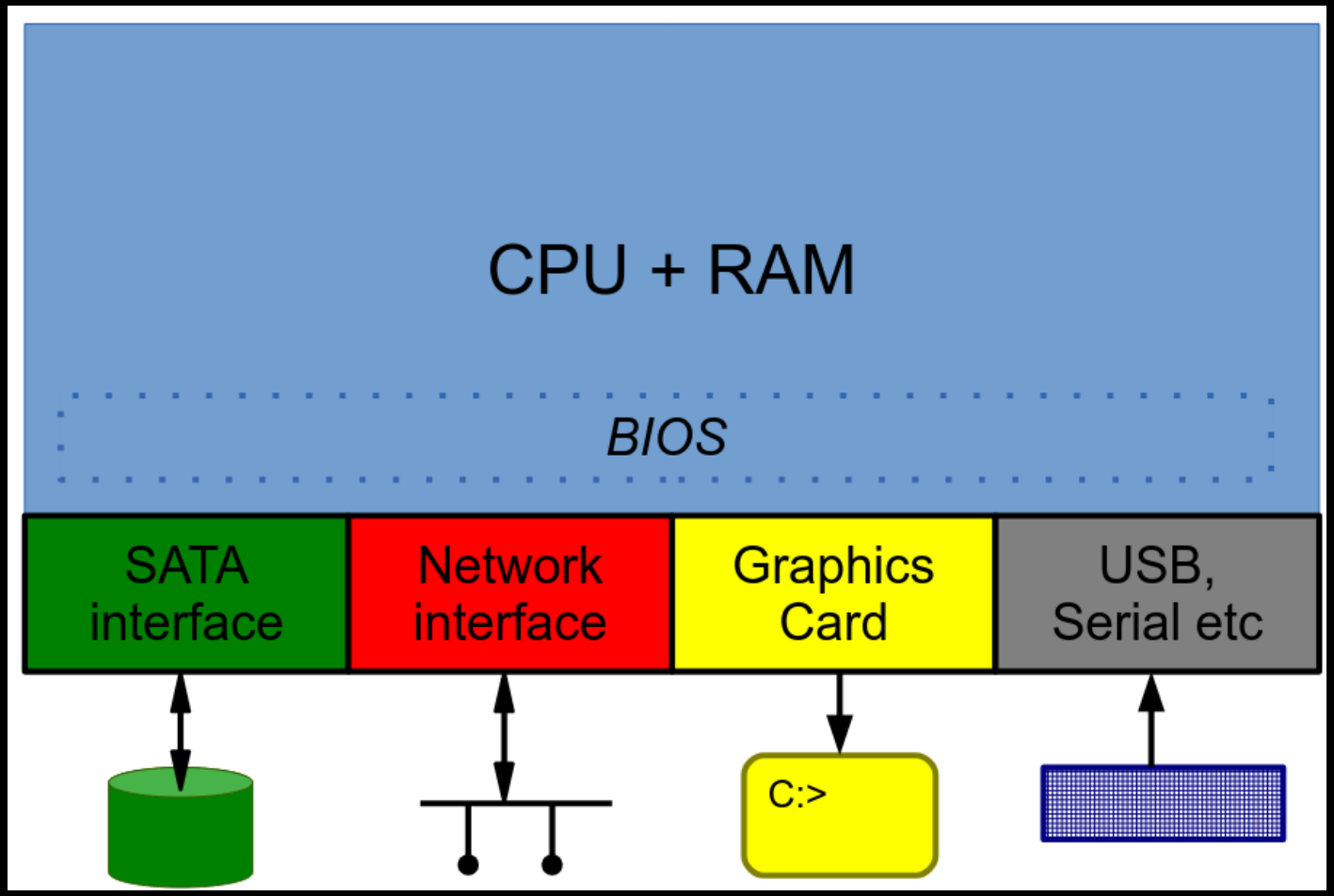
Apperçu de la section 2

La virtualisation

Idée clé

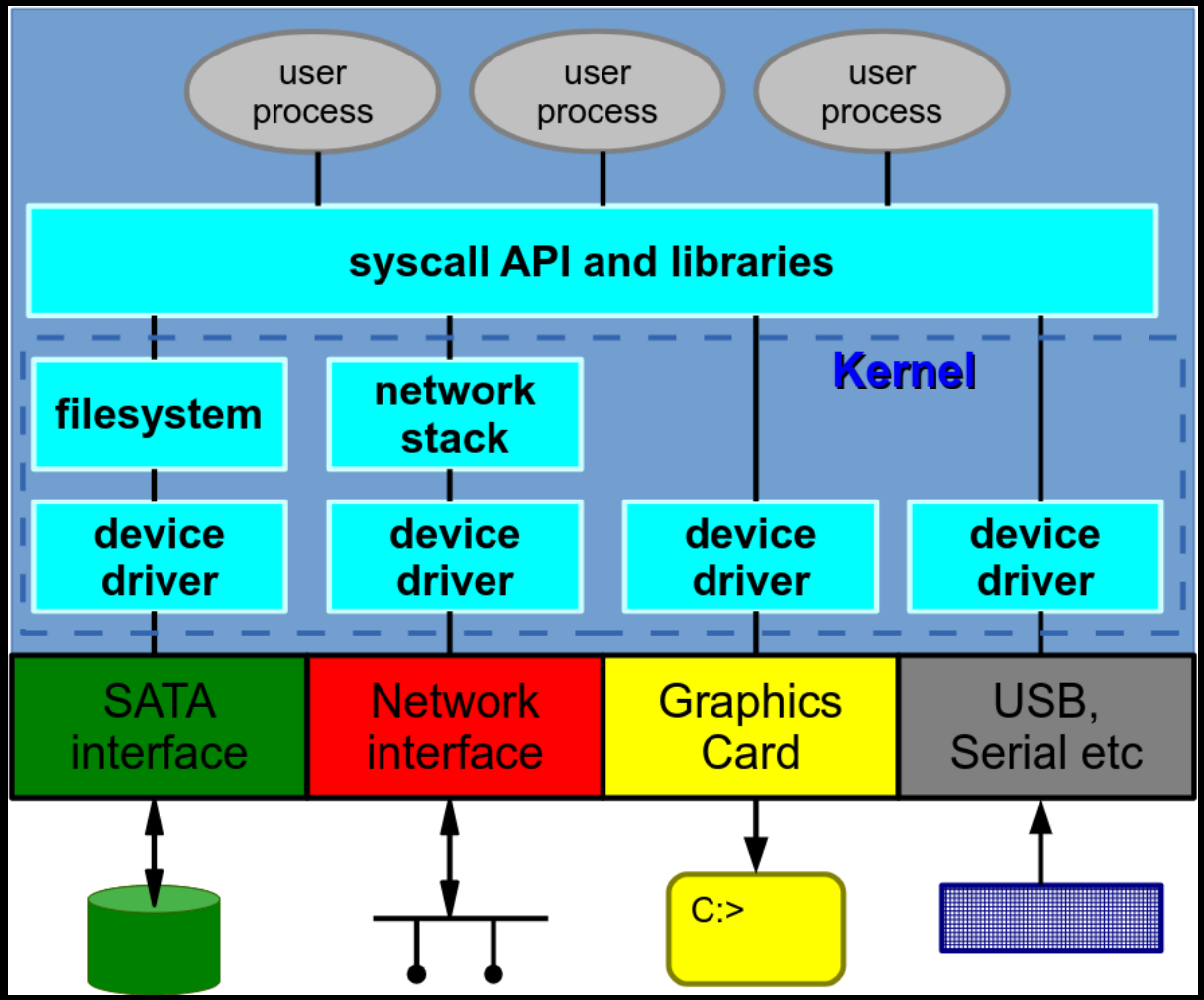
Un système d'exploitation(OS) est avant tout un **mécanisme de virtualisation généralisée** : il donne à chaque application (User process) l'illusion de disposer d'une machine (PC) complète et dédiée.

PC


CPU + RAM
BIOS
SATA interface
Network interface
Graphics Card
USB, Serial etc
C:>

©NRSC

Os


user process
user process
user process
syscall API and libraries
Kernel
filesystem
network stack
device driver
device driver
device driver
device driver
SATA interface
Network interface
Graphics Card
USB, Serial etc
C:>

©NRSC

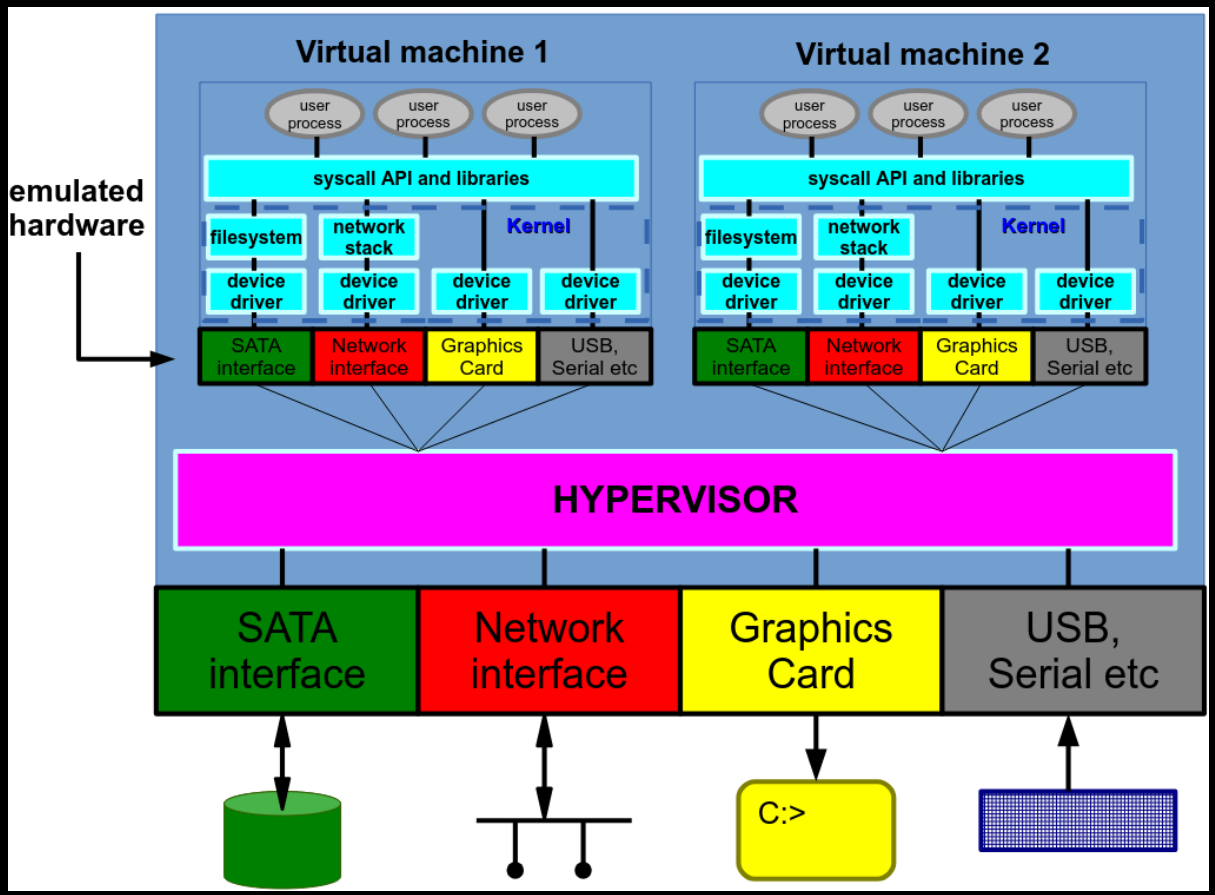
Exemples de virtualisation

- **CPU** : chaque processus(User process) croit avoir son propre processeur $\rightarrow$ planification $=$ illusion de parallélisme.
- **Mémoire** : chaque processus dispose d'un espace linéaire privé $\rightarrow$ mémoire virtuelle $=$ isolation et protection.
- **Périphériques** : fichiers, sockets, écrans sont des abstractions $\rightarrow$ partage sûr et équitable.

La virtualisation: Principe unificateur

L'OS peut être vu comme un **hyperviseur généralisé** : il virtualise CPU, mémoire et périphériques pour un usage **efficace**.

Hyperviseur



©NRSC

Hyperviseur et machines virtuelles

Définition

Un **hyperviseur** (ou VMM – Virtual Machine Monitor) est un logiciel qui **virtualise l'ensemble du matériel** pour exécuter plusieurs **machines virtuelles (VM)**.

Machines virtuelles

Machines virtuelles

- Une VM est un **système complet simulé** : CPU virtuel, mémoire virtuelle, périphériques virtuels.
- Chaque VM exécute son propre système d'exploitation, comme si elle était seule sur le matériel.

OS vs Hyperviseur

Similarités

- L'OS virtualise le CPU, la mémoire et les périphériques pour les processus.
- L'hyperviseur fait la même chose, mais pour des systèmes d'exploitation entiers.

OS vs Hyperviseur

Différence clé

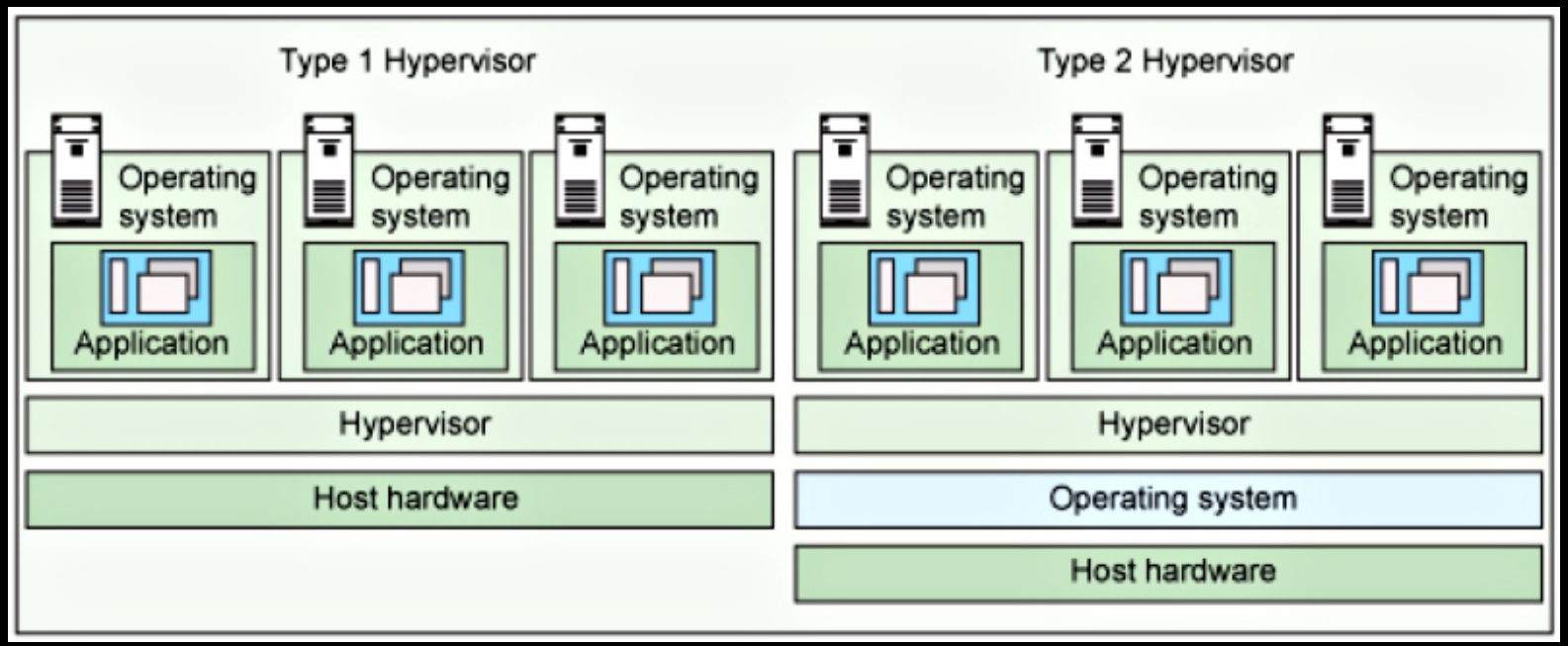
- **OS** : fournit une machine virtuelle aux **applications** (User process).
- **Hyperviseur** : fournit une machine virtuelle aux **systèmes d'exploitation** (Os).

Apperçu de la section 3

Hyperviseurs & Libvirt

- **Type 1** : KVM (Linux), Xen — natifs sur le hardware.
- **Type 2** : VirtualBox, VMware Workstation.
- **Libvirt** : API unifiée + outils (`virsh`, `virt-manager`).

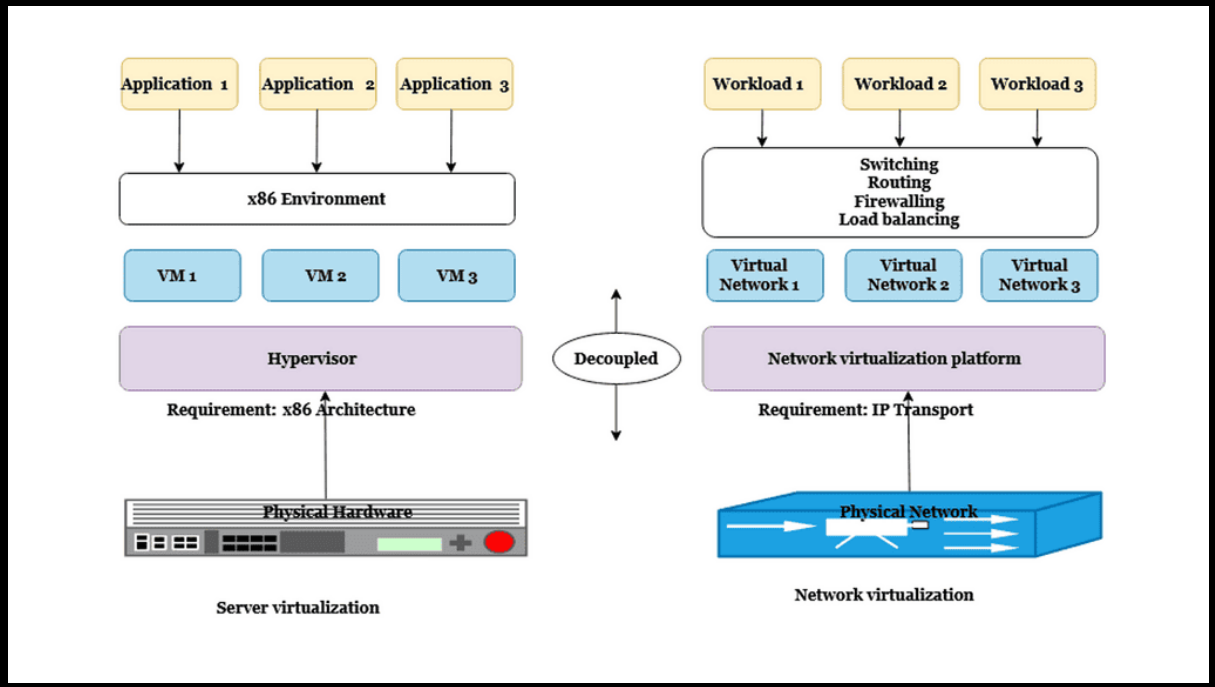
Hyperviseurs Type 1 vs Type 2



source: https://oze.pwr.edu.pl/kursy/introcloud/content/start/K-04-03.html

Virtualisation et Virtualisation Réseaux

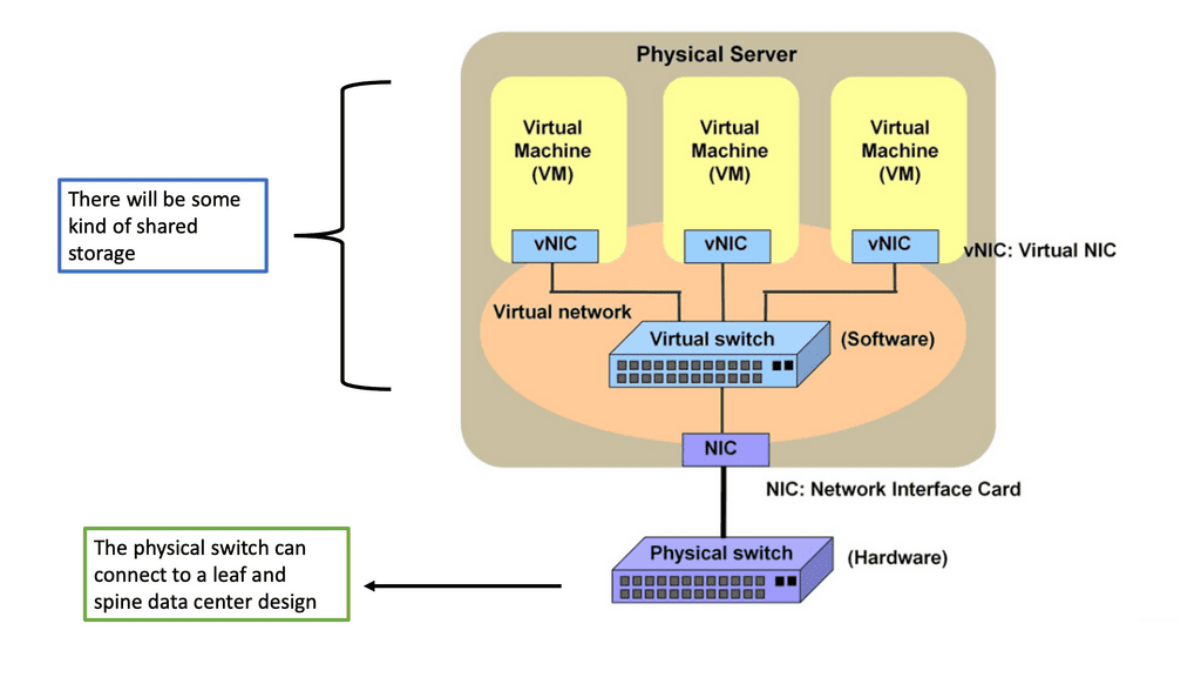
- Découpage du réseau physique en réseaux logiques isolés
- Améliore : performance, sécurité, flexibilité, scalabilité



©Network Insight. Open vSwitch (OVS) Basics.

Virtualisation Réseaux: switchs virtuels

- C'est un switch logiciel dans l'hyperviseur, qui relie les VMs,
- reproduit les fonctions d'un switch physique, avec plus de flexibilité

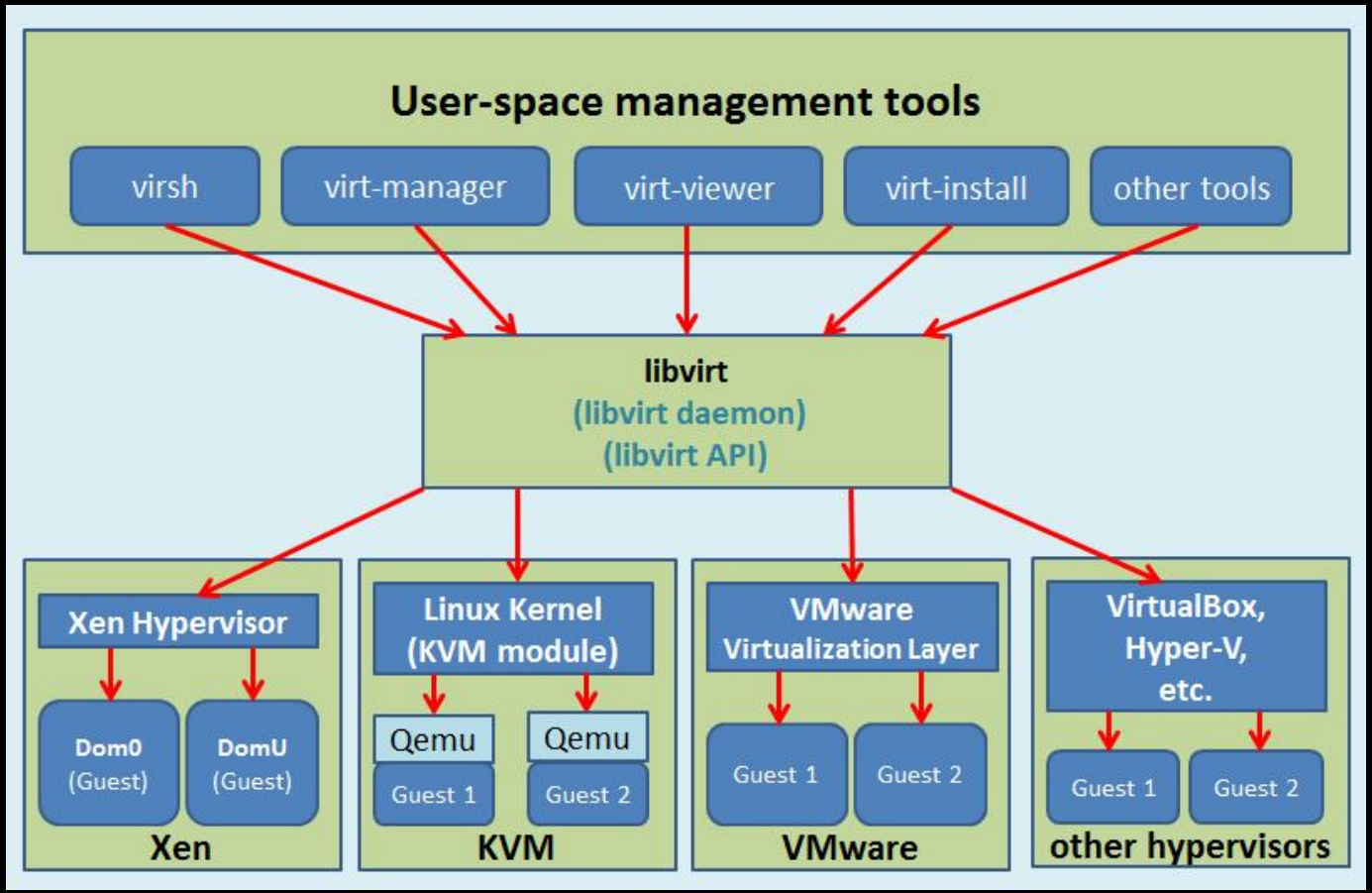


©Network Insight. Open vSwitch (OVS) Basics.

Libvirt

source: https://www.openvswitch.org/support/slides/OpenStack-131107.pdf

Libvirt API



source: https://docker.uptime-formation.fr/KVM/Theorie-KVM-IHM/

Libvirt : languages de bindings

Bindings for other languages and integration API modules

Libvirt supports C and C++ directly, and has bindings available for other languages:

- **C#**: Arnaud Champion develops C# bindings.
- **Go**: Daniel Berrange develops Go bindings.
- **Java**: Daniel Veillard develops Java bindings.
- **OCaml**: Richard Jones develops OCaml bindings.
- **Perl**: Daniel Berrange develops Perl bindings.
- **PHP**: Radek Hladik started developing PHP bindings in 2010.

  In February 2011 the binding development has been moved to the libvirt.org website as libvirt-php project.

  The project is now maintained by Michal Novotny and it's heavily based on Radek's version. For more information, including information on posting patches to libvirt-php, please refer to the PHP bindings site.
- **Python**: Libvirt's python bindings are split to a separate package since version 1.2.0, older versions came with direct support for the Python language.

  If your libvirt is installed as packages, rather than compiled by you from source code, ensure you have the appropriate package installed.

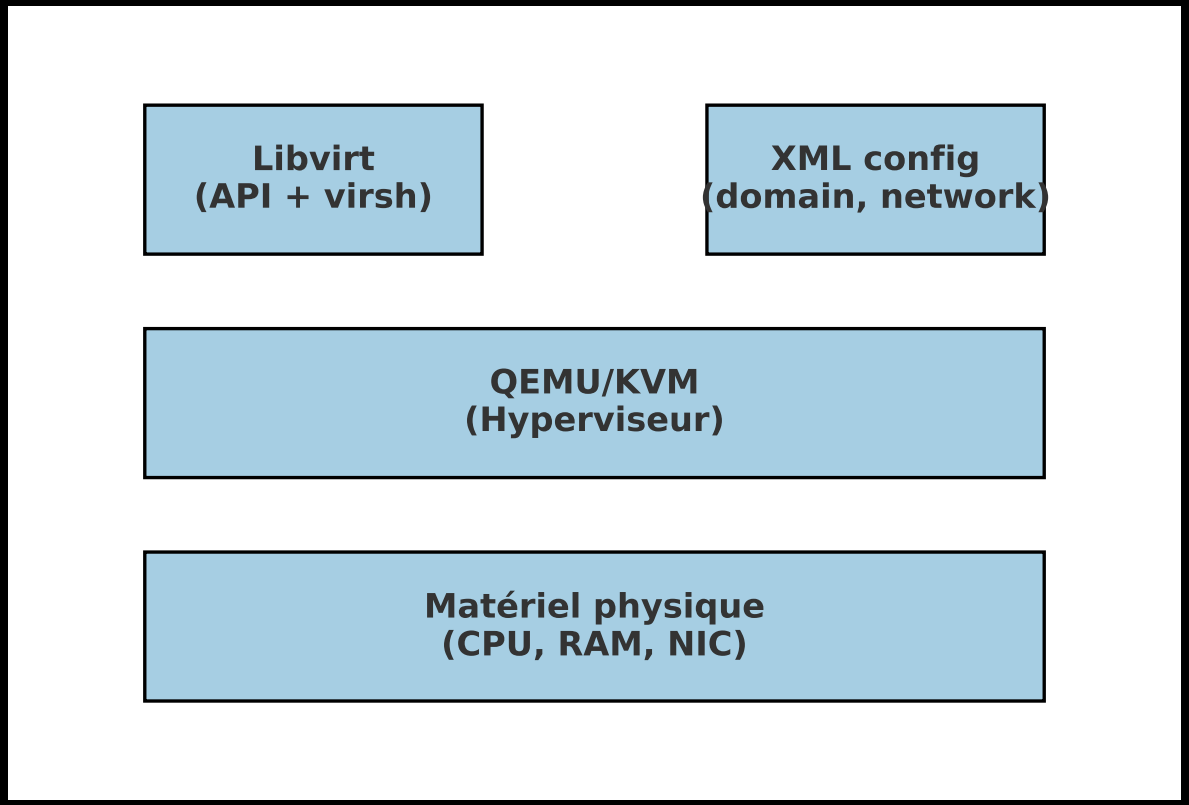
  This is named **libvirt-python** on RHEL/Fedora, python-libvirt on Ubuntu, and may be named differently on others.

  For usage information, see the Python API bindings page.
- **Ruby**: Chris Lalancette develops Ruby bindings.

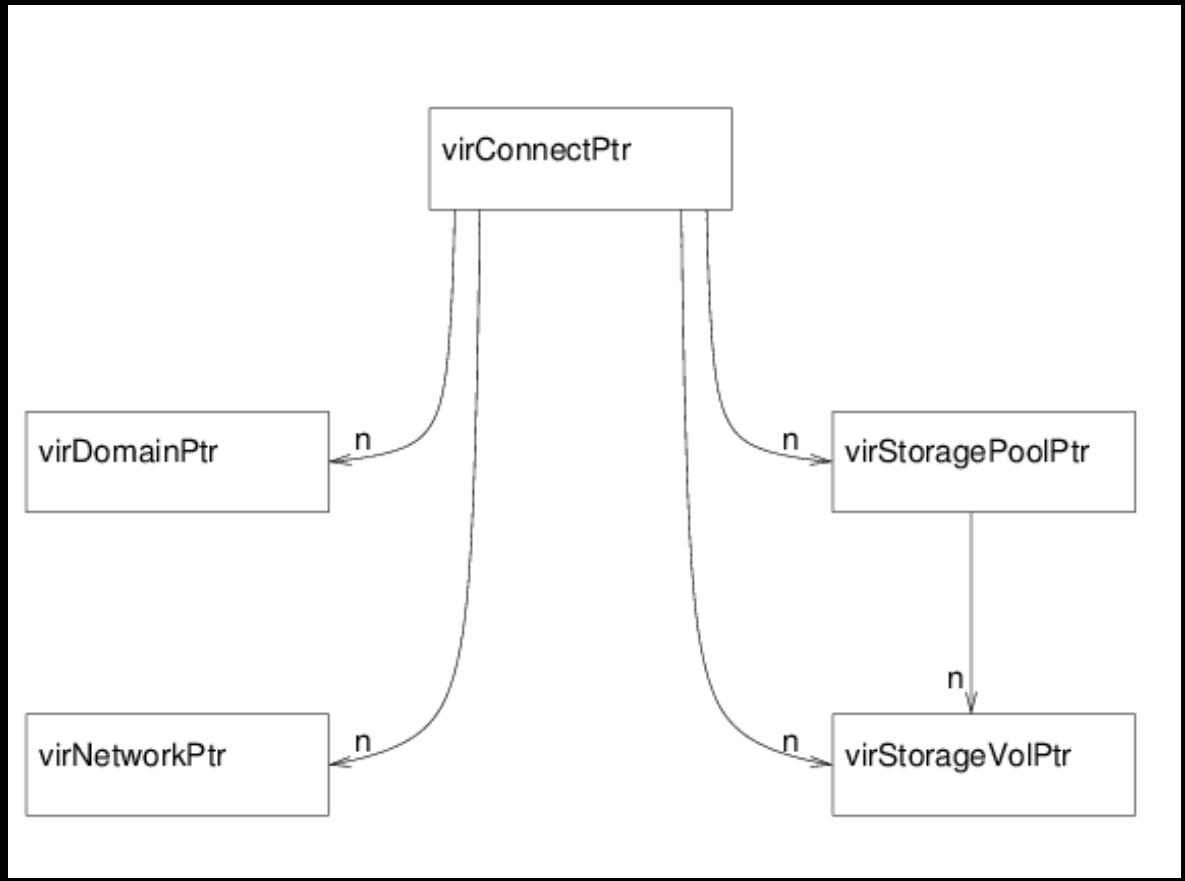
source: `https://libvirt.org/bindings.html`

Libvirt + KVM : architecture

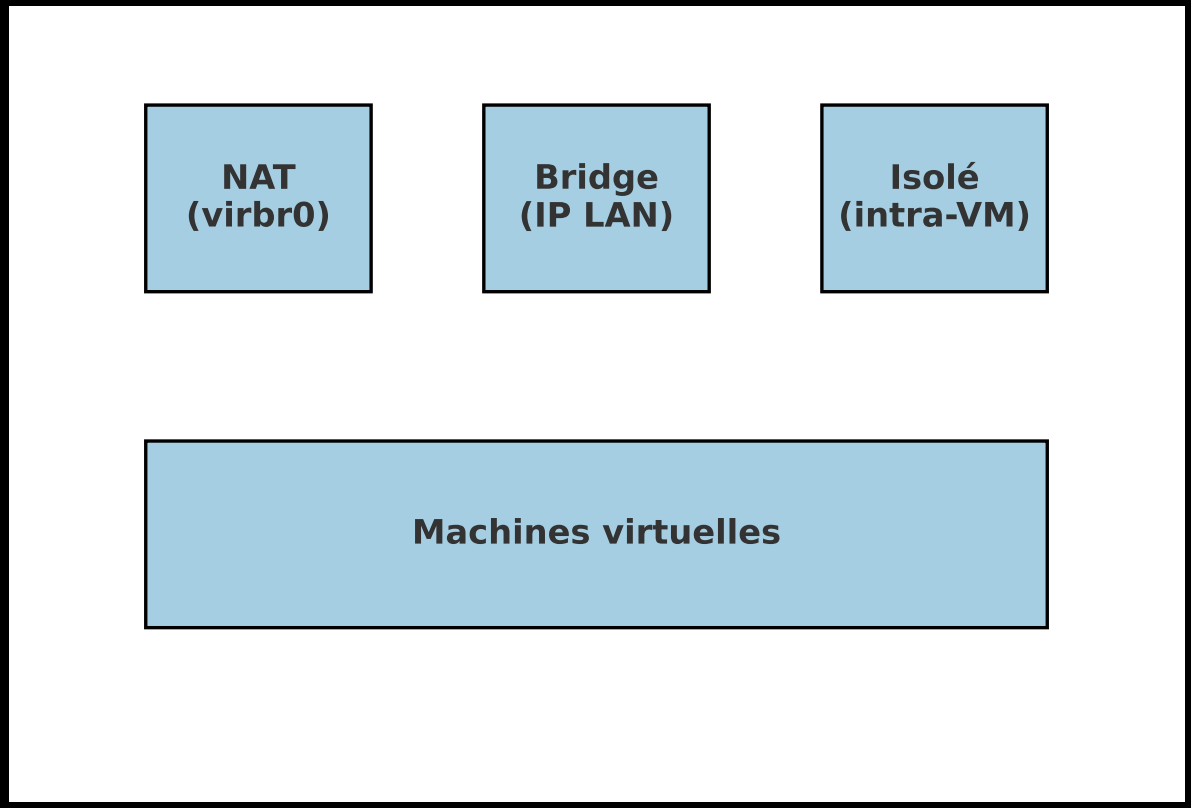
- Ressources en XML : `domain`, `network`, `pool/volume`.
- Libvirt orchestre QEMU/KVM et le réseau virtuel.



Libvirt : Ressources



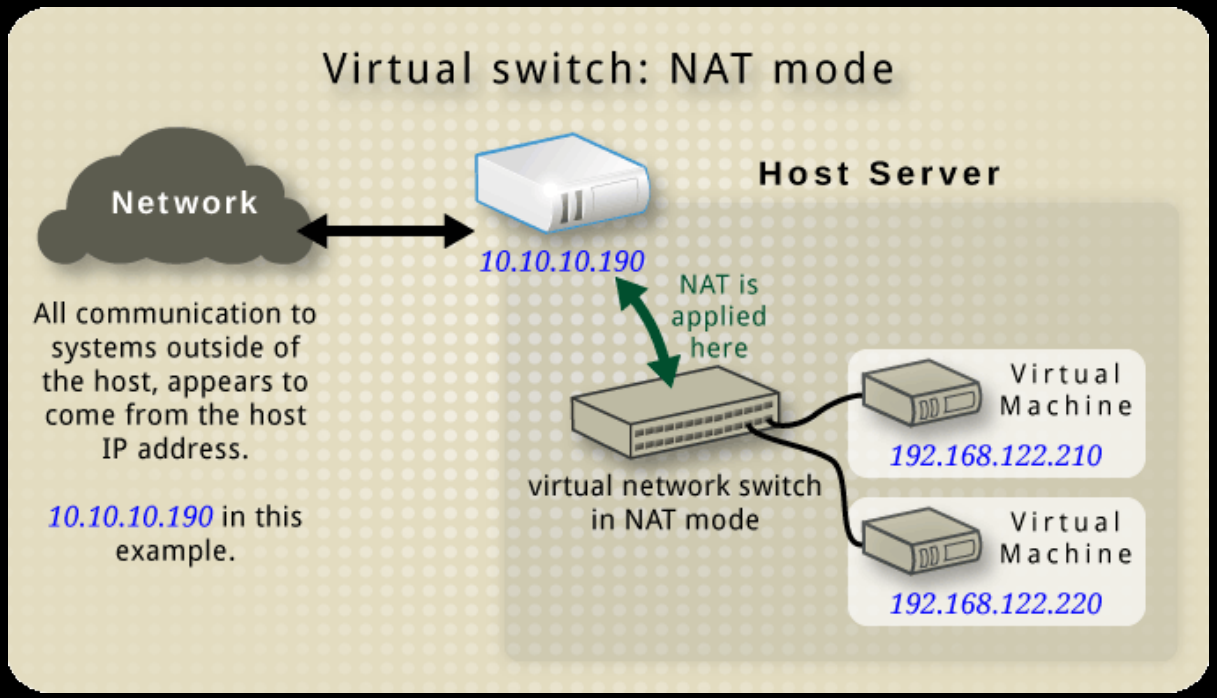
3 modes réseau Libvirt (vision rapide)



NAT (sortant), Bridge (IP du LAN), Isolé (intra-VMs).

NAT Libvirt : quand et pourquoi

- **Usage** : labs, CI, services non exposés.
- **Mécanique** : `virbr0` + DHCP/DNSMasq + MASQUERADE.
- **Limite** : pas d'accès direct entrant sans port-forward.



NAT Libvirt : Exemple

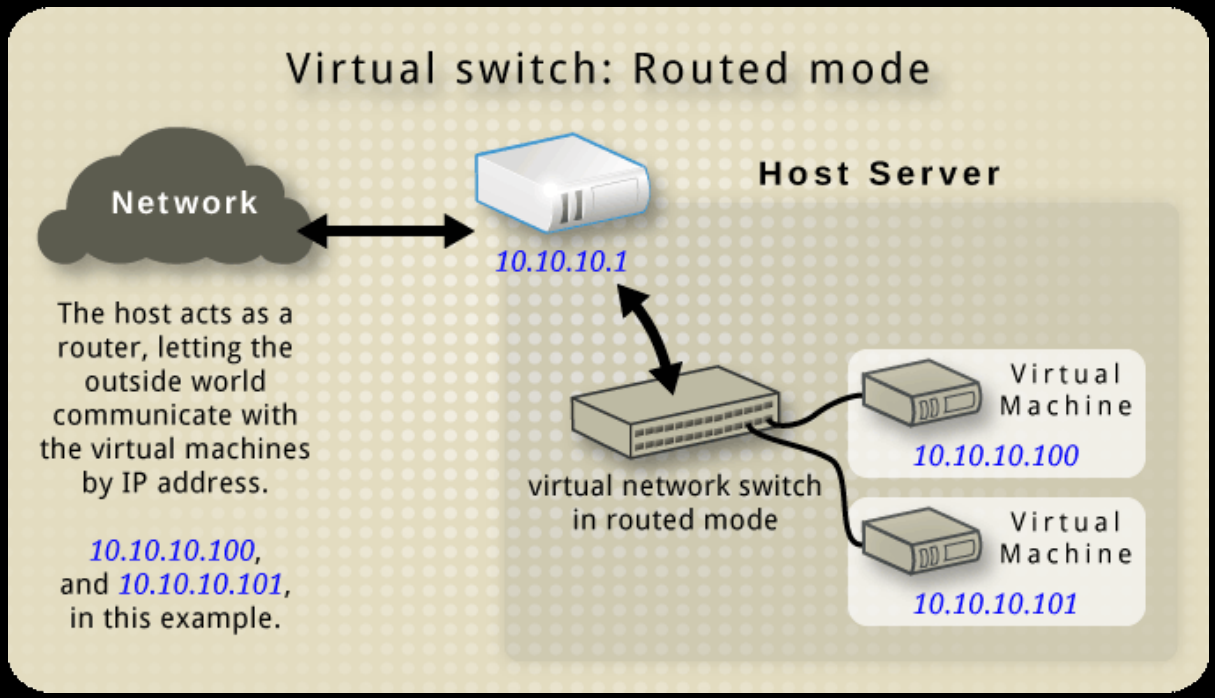
```
<network>
  <name>default</name>
  <uuid>47edf03c-38c6-4aa1-9098-bb968f049837</uuid>
  <!-- START FWD -->
  <forward mode='nat'>
    <nat>
      <port start='1024' end='65535'/>
    </nat>
  </forward>
  <!-- END FWD -->
  <bridge name='virbr0' stp='on' delay='0'/>
  <mac address='52:54:00:a6:50:2a'/>
  <!-- START IP -->
  <ip address='192.168.122.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.122.2' end='192.168.122.254'/>
    </dhcp>
  </ip>
  <!-- END IP -->
</network>
```

NAT Libvirt : Exemple iptables

```
1  sudo iptables -t nat -L
2
3  Chain LIBVIRT_PRT (1 references)
4  target     prot opt source               destination
5  RETURN     all  --  192.168.122.0/24     base-address.mcast.net/24
6  RETURN     all  --  192.168.122.0/24     255.255.255.255
7  MASQUERADE  tcp  --  192.168.122.0/24     !192.168.122.0/24     masq ports: 1024-65535
8  MASQUERADE  udp  --  192.168.122.0/24     !192.168.122.0/24     masq ports: 1024-65535
9  MASQUERADE  all  --  192.168.122.0/24     !192.168.122.0/24
```

Bridge Libvirt : IP “citoyenne” du LAN

- **Usage** : services joignables (SSH, HTTP. . .).
- **Mécanique** : bridge lié à `ethX`, DHCP du LAN.
- **Attention** : surface d’exposition plus large.



Bridge Libvirt : Exemple

```
1  <network>
2    <name>routed</name>
3    <bridge name="s0"/>
4    <forward mode="route"/>
5    <ip address="10.10.10.1" netmask="255.255.255.128">
6      <dhcp>
7        <range start="10.10.10.2" end="10.10.10.127"/>
8      </dhcp>
9    </ip>
10 </network>
```

Bridge Libvirt : Exemple

- Ajoutez une route sur le routeur en amont:

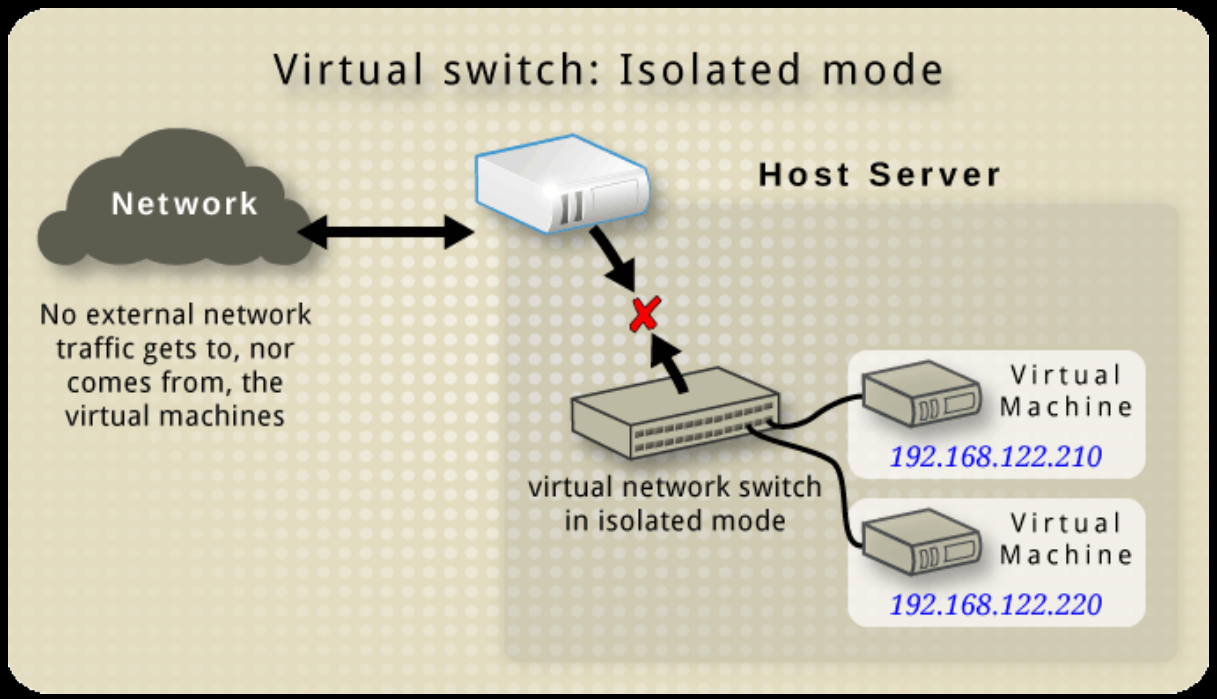
```
1       ip route add 10.10.10.128/25 via 10.10.10.1
```

- Ou bien, Sur l'hôte libvirt, activez la fonction proxy ARP:

```
1   echo 1 | sudo tee /proc/sys/net/ipv4/conf/s0/proxy_arp
```

Isolé Libvirt : bulles fermées

- **Usage** : tests, environnements sensibles.
- **Mécanique** : bridge sans uplink physique.
- **Accès** : bastion/NAT/relais si nécessaire.



Isolé Libvirt : bulles fermées

```
1   <network>
2     <name>guest-only</name>
3     <forward mode='none'/>
4     <bridge name='sandbox' stp='on' delay='0'/>
5     <ip address='192.168.122.1' netmask='255.255.255.0'>
6       <dhcp>
7         <range start='192.168.122.100' end='192.168.122.200'/>
8       </dhcp>
9     </ip>
10  </network>
```

NAT Libvirt : créer un réseau

On considère la définition du réseau nat-net.xml au slide-20

```
# Définir un réseau NAT à partir d'un XML
virsh net-define nat-net.xml
virsh net-autostart nat-net
virsh net-start nat-net
```

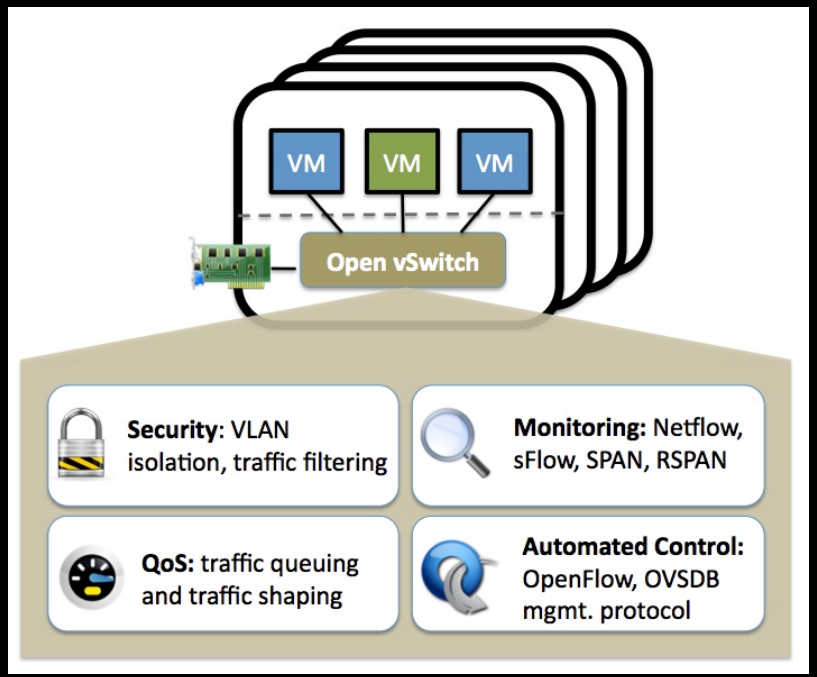
NAT Libvirt : gestion des interface réseaux

```
# Attacher une VM à ce réseau
virsh attach-interface VM1 --type network --source nat-net  --model virtio --config --live

# Attacher une VM à ce réseau en spécifiant son adresse MAC
virsh attach-interface VM1 --type network --source nat-net  --model virtio \
     --mac 00:11:22:33:44:55 --config --live

# Detacher une VM du réseau en spécifiant son adresse MAC
virsh detach-interface VM1 --type network --mac 00:11:22:33:44:55 --live --config
```

Apperçu de la section 4

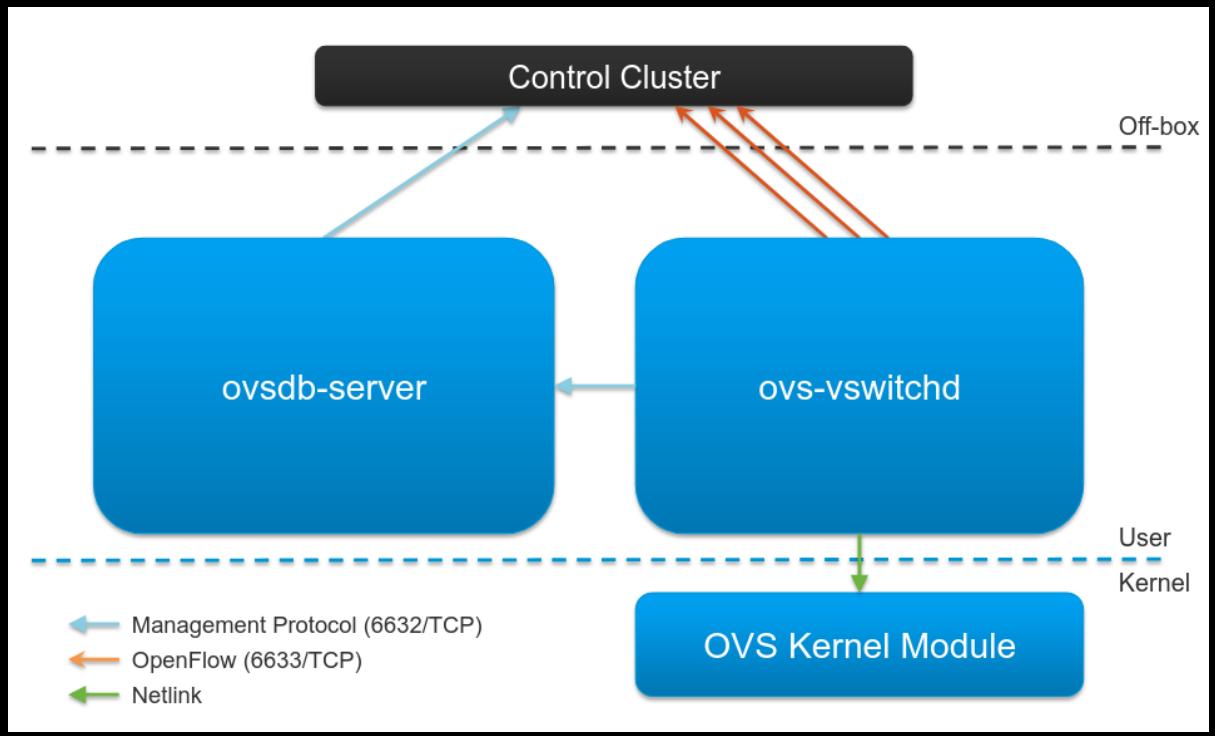
Pourquoi OVS plutôt qu'un bridge Linux simple ?

- **Fonctions avancées** : VLAN, trunk, QoS, ACLs.
- **Programmable** (OpenFlow), mirroring, NetFlow/sFlow.
- **Multi-hôte** : VXLAN/GRE pour overlays.



Architecture OVS

- **Datapath** (OVS Kernel Module, Kernel Driver) : Chemin rapide.
- `ovs-vswitchd` (User space) : contrôle.
- Outils : `ovs-vsctl`, `ovs-ofctl`, `ovs-appctl`.



OVSDB : rôle

Base de données qui contient la configuration du switch :

- Ponts (bridges) => switchs virtuels
- Interfaces => ports physiques ou virtuels attachés
- Tunnels => VXLAN , GRE, IP-in-IP
- Adresses OVSDB
  - IP + port
- contrôleur OpenFlow (bascule en mode SDN)
  - Ryu, ONOS, OpenDaylight

OVSDB : persistance

La configuration est stockée sur disque et survit aux redémarrages.

OVSDB : propriétés

Base de données personnalisée :

- Contraintes de valeurs
- Références faibles
- Collecte des déchets (GC)

OVSDB : journalisation

Journalisée $\rightarrow$ idéale pour le débogage !

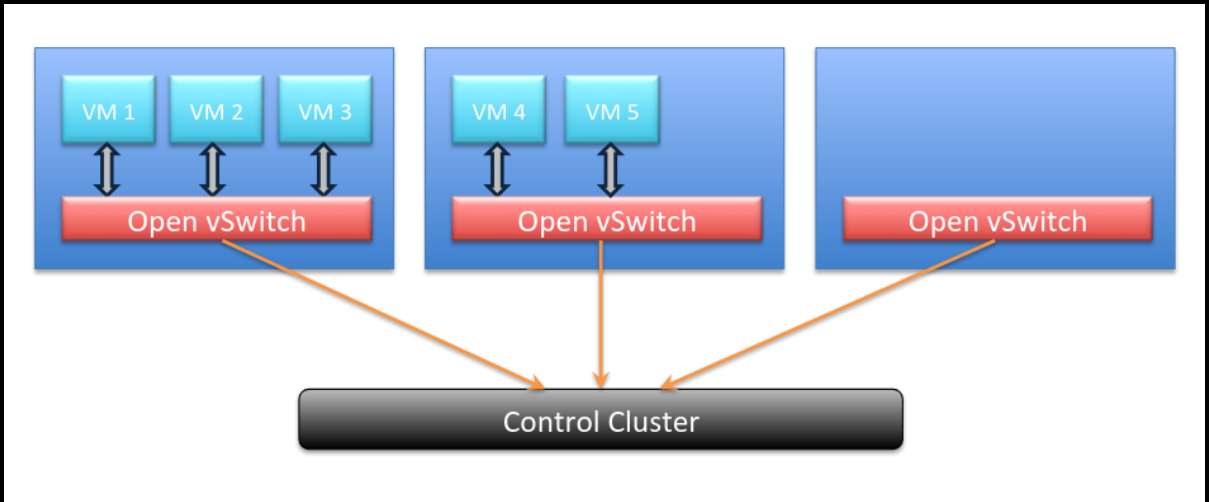
On peut retracer l'historique des modifications de la base de données.

OVSDB : interopérabilité

_

- Communique via le protocole OVSDB
- Avec le gestionnaire et `ovs-vswitchd`
- Protocole normalisé (RFC 7047)

switch distribué



OVS : démarrage express

```
# Installer openvswitch
sudo apt install openvswitch-switch

# Créer un bridge OVS et y raccorder des ports
sudo ovs-vsctl add-br ovsbr0
sudo ovs-vsctl add-port ovsbr0 ens3        # port physique
sudo ovs-vsctl add-port ovsbr0 tap-vm1     # interface VM (tap)
```

OVS: Pour aller loins

- https://www.openvswitch.org/support/slides/
- https://docs.openvswitch.org/en/latest/intro/install/

Apperçu de la section 5

Pourquoi intégrer ?

- **Libvirt** orchestre, **OVS** commute/segmente.
- Réseaux *prod-like* (VLAN,VXLAN, QoS, mirroring).

Créer un réseau Libvirt à partir d'un bridge OVS

```
<network>
  <name>ovs</name>
  <uuid>f58cad29-0455-439a-b533-8362669cec92</uuid>
  <forward mode='bridge'/>
  <bridge name='ovsbr0'/>
  <virtualport type='openvswitch'/>
</network>
```

Isolation fine : combiner VLAN + groupes

- **Micro-segmentation** avec tags VLAN.
- **Groupes de sécurité** via ACLs/iptables sur l'hôte.

Mirroring & capture

- SPAN/port-mirroring OVS pour déboguer.
- Capture `tcpdump` / IDS (Zeek/Suricata).

Apperçu de la section 6

Performance

- offloads (TSO/GSO/GRO), MTU adaptés.

Sécurité

- Réduire l'exposition (bridges & filtres ebtables/nft).
- VLAN par rôle, séparation mgmt/data.

Observabilité

- Export NetFlow/sFlow/IPFIX depuis OVS.
- Logs libvirtd + statistiques OVS (`ovs-appctl`).

Troubleshooting — checklist

- **Câblage virtuel** : VM → tap → br0 → uplink.
- **Tag VLAN** cohérent (access vs trunk).
- MTU/ARP/NDP, routes, filtres.

À retenir

- **Libvirt** orchestre ; **OVS** offre la puissance L2/L3 virtuels.
- Choisir le **mode réseau** selon l'exposition requise.
- VLAN/VXLAN = **segmentation** et **scale**.

Ressources

- libvirt.org
- openvswitch.org
- https://network-insight.net/2015/11/21/open-vswitch-ovs-basics/