

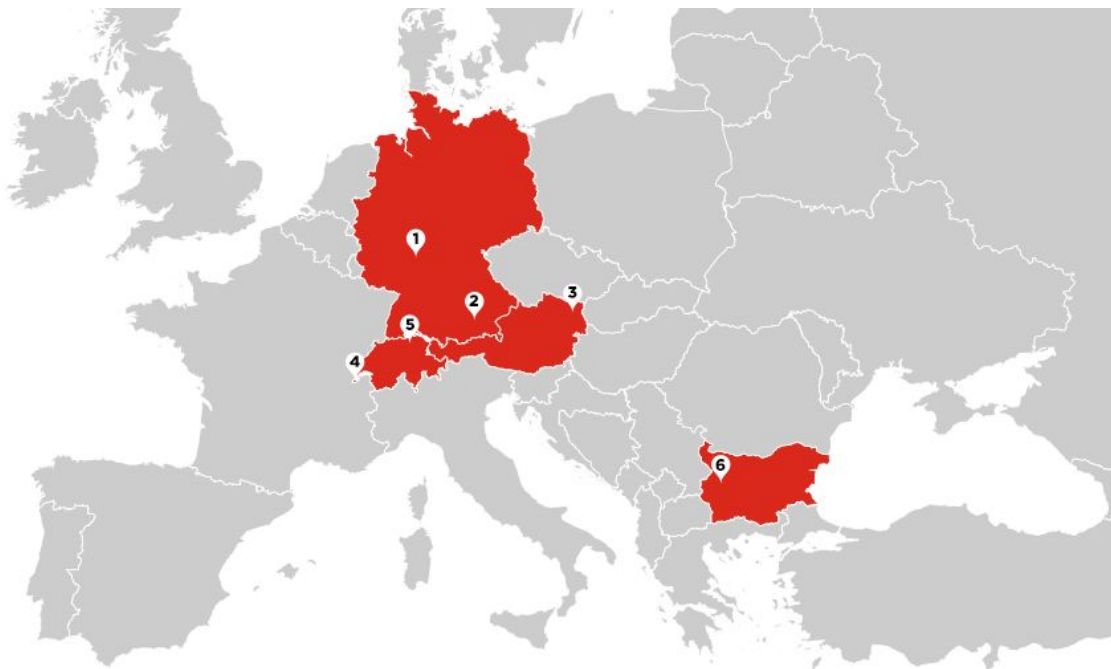
# Fonctionnement d'un LB as a service



# EXOSCALE

Un Cloud Provider Européen

- |    |           |          |
|----|-----------|----------|
| 1. | FRANKFURT | DE-FRA-1 |
| 2. | MUNICH    | DE-MUC-1 |
| 3. | VIENNA    | AT-VIE-1 |
| 4. | GENEVA    | CH-GVA-2 |
| 5. | ZURICH    | CH-DK-2  |
| 6. | SOFIA     | BG-SOF-1 |



# Mathieu Corbin, Ingénieur **@Exoscale**

## Développeur

- Clojure
- Golang
- Emacs
- ...

## Sysadmin

- Automatisation
- CI/CD
- Monitoring
- Systèmes distribués
- ...



@\_mcorbin

mcorbin.fr

# Le besoin

## Un Load Balancer as a service

# Le besoin

- Layer 4
  - TCP/UDP
- Préserver l'IP source
- S'intègre avec nos instance pools (groupes de machines)
  - Scale up/down => mise à jour du load balancer
- Tolérance aux pannes
- Bonnes performances

# Le produit

- Un load balancer, une IP, plusieurs services
- Chaque service cible un groupe de machine et peut être paramétré de façon indépendante

Load balancer: **159.100.241.237**

### Service 1

**port:** 8080

**protocol:** TCP

**target port:** 8080

**healthchecks:** HTTP on / every 5 s

**algo:** Source Hash

### Service 2

**port:** 2000

**protocol:** UDP

**target port:** 9000

**healthchecks:** TCP every 10s

**algo:** Round Robin

Backend 1

Backend 2

**Instance pool 1**

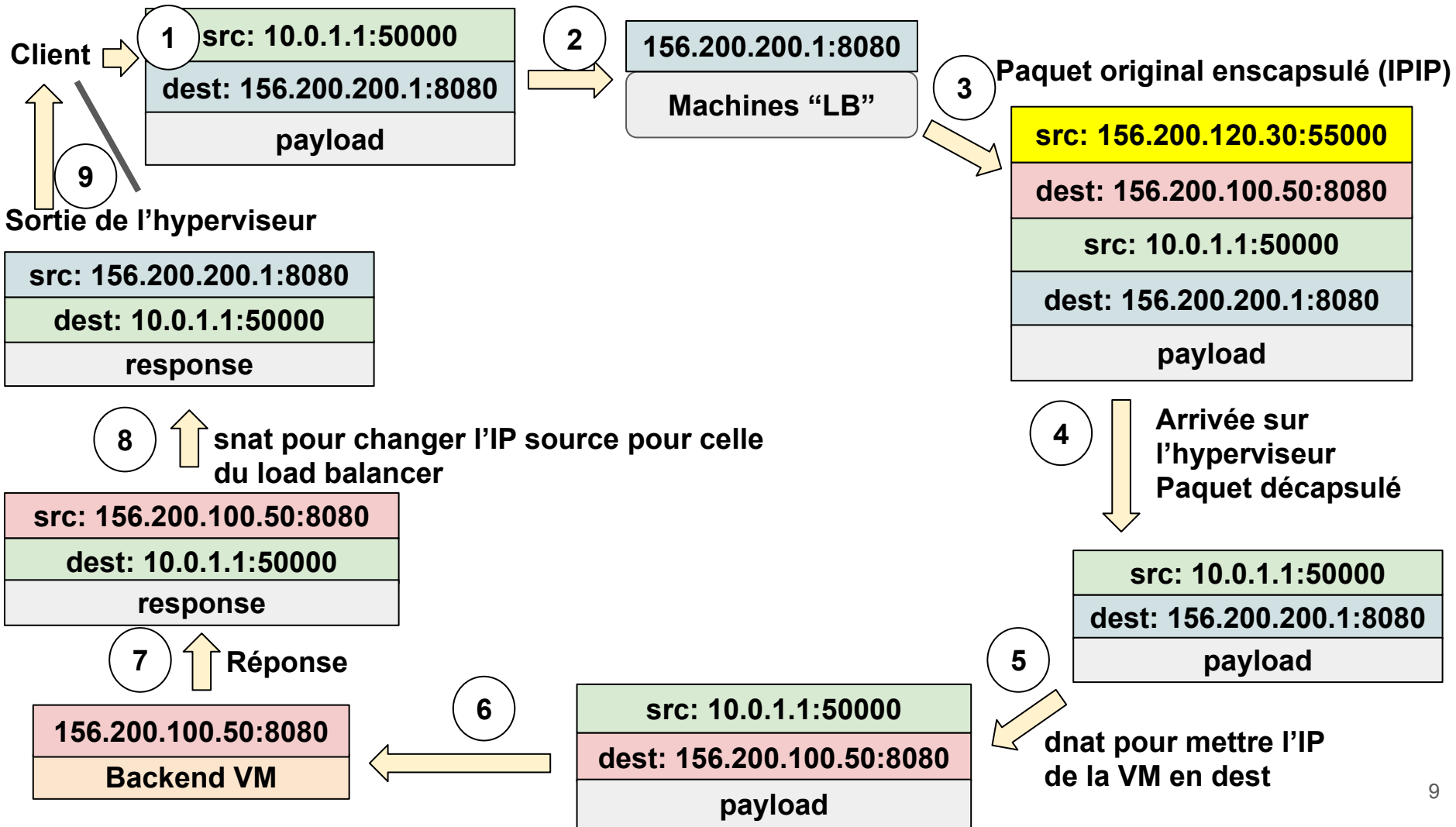
Backend 1

Backend 2

**Instance pool 2**

# Fonctionnement réseau





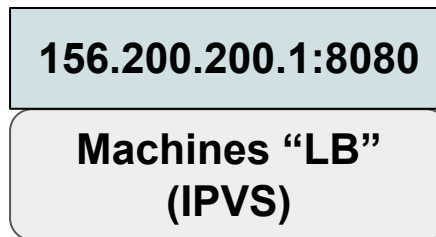
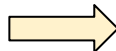
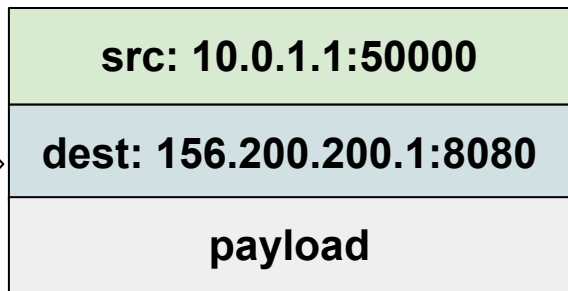
# En détail...

# Première étape: les machines LB

# Machines LB: IPVS

- IPVS est utilisé pour le load balancing
  - Intégré au kernel Linux
  - Très bonnes performances
  - Flexible, de nombreuses options (dont IPIP)
  - Simple d'utilisation
  - Configuré dans un namespace réseau dédié

Client

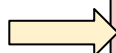
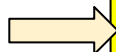


IPVS choisit un serveur backend dans le pool, ici **156.200.100.50**

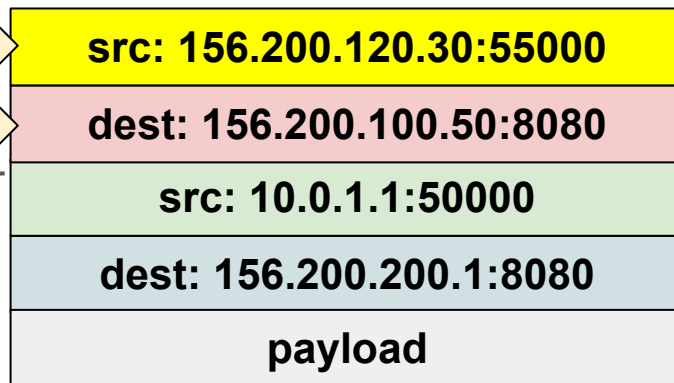


**Encapsulation IPIP**

**IP partagée par toutes les machines LB**  
**IP machine backend**



**Paquet original**

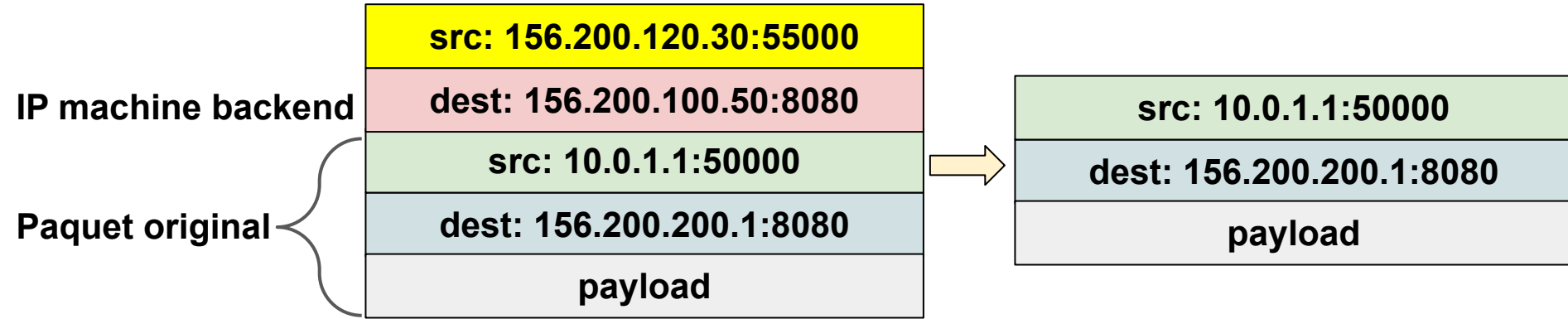


**Arrivée sur l'hyperviseur**

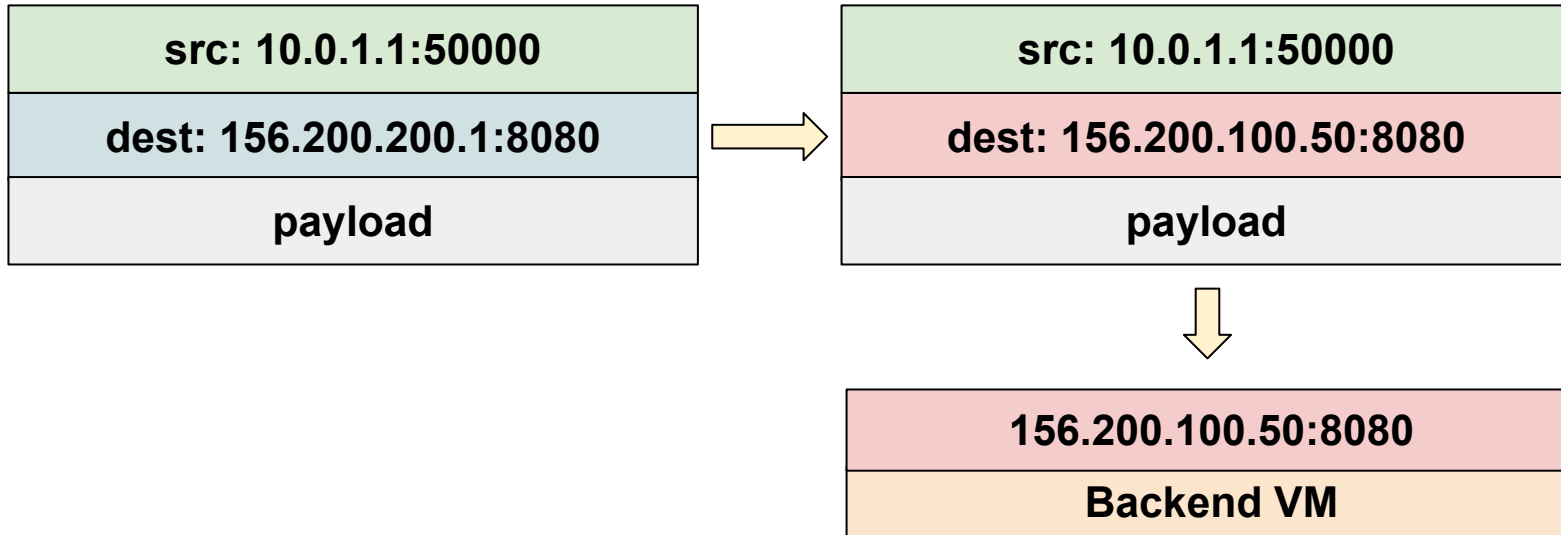
# L'hyperviseur

- La machine virtuelle cible tourne dessus
- Un module kernel écrit en interne est activé
  - Utilisation de netfilter
  - Deux règles iptables à ajouter pour chaque machine virtuelle dans un service de load balancer

# Hyperviseur: décapsulation



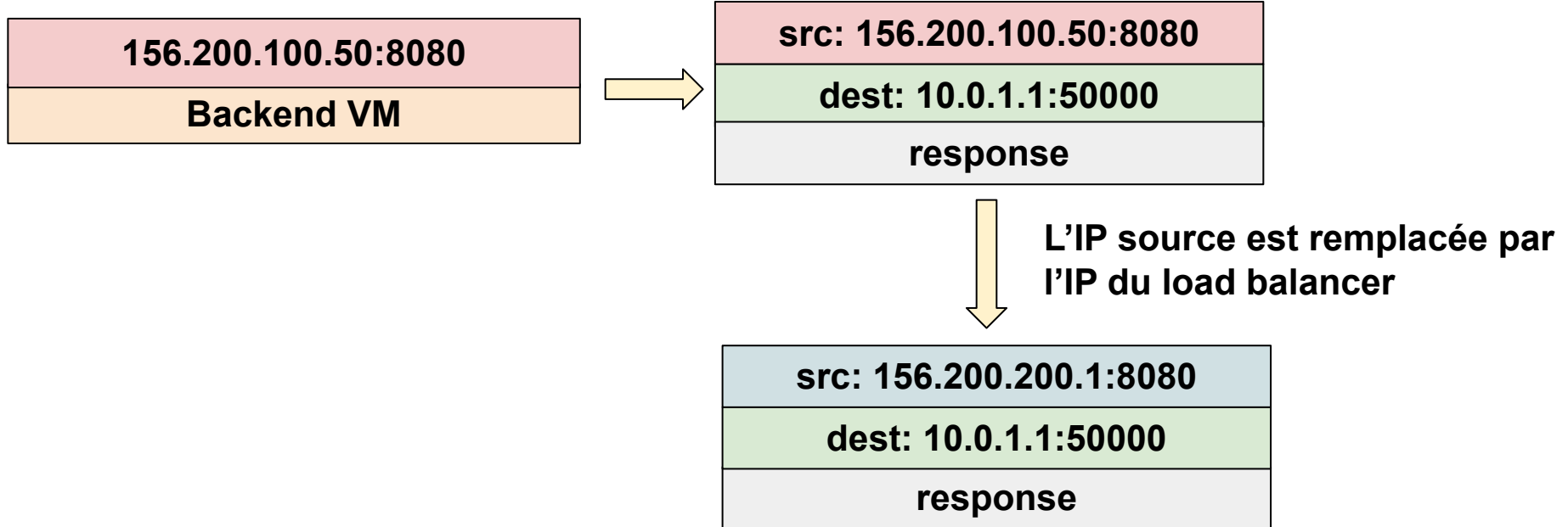
# Hyperviseur: nat



La machine virtuelle a reçue le paquet. Pour elle, le load balancer n'existe pas  
L'IP source a été conservée



# Hyperviseur: nat



# En résumé: ce que voit le client

## Envoie

<b>src: 10.0.1.1:50000</b>
<b>dest: 156.200.200.1:8080</b>
<b>payload</b>

## reçoit

<b>src: 156.200.200.1:8080</b>
<b>dest: 10.0.1.1:50000</b>
<b>response</b>

# En résumé: ce que voit la VM

## Reçoit

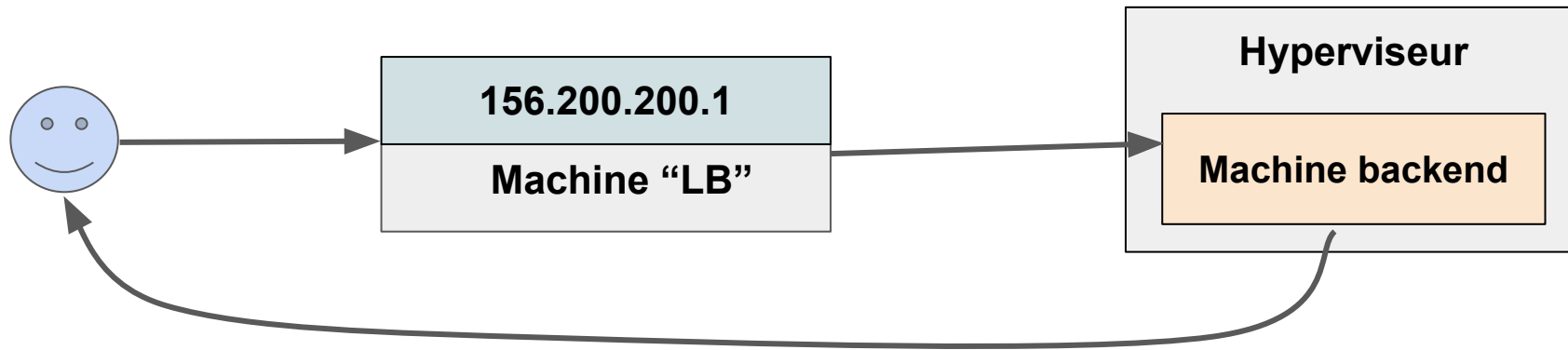
<b>src: 10.0.1.1:50000</b>
<b>dest: 156.200.100.50:8080</b>
<b>payload</b>

## Envoie

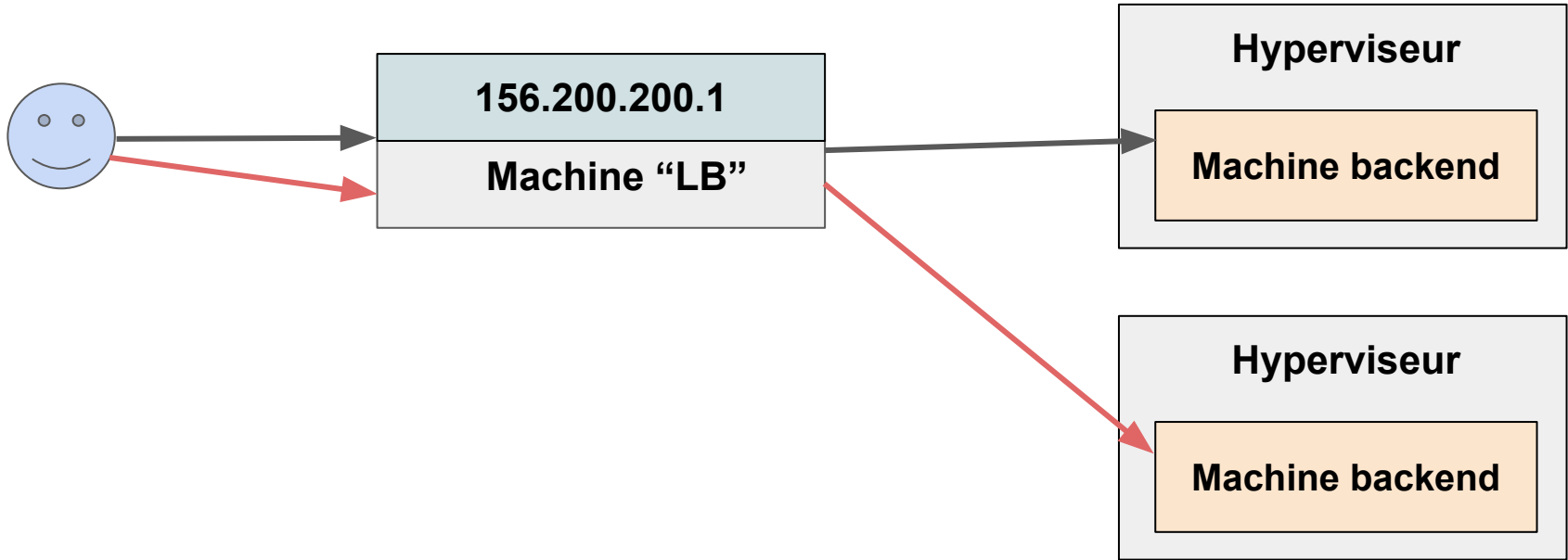
<b>src: 156.200.100.50:8080</b>
<b>dest: 10.0.1.1:50000</b>
<b>response</b>

# Direct Server Return

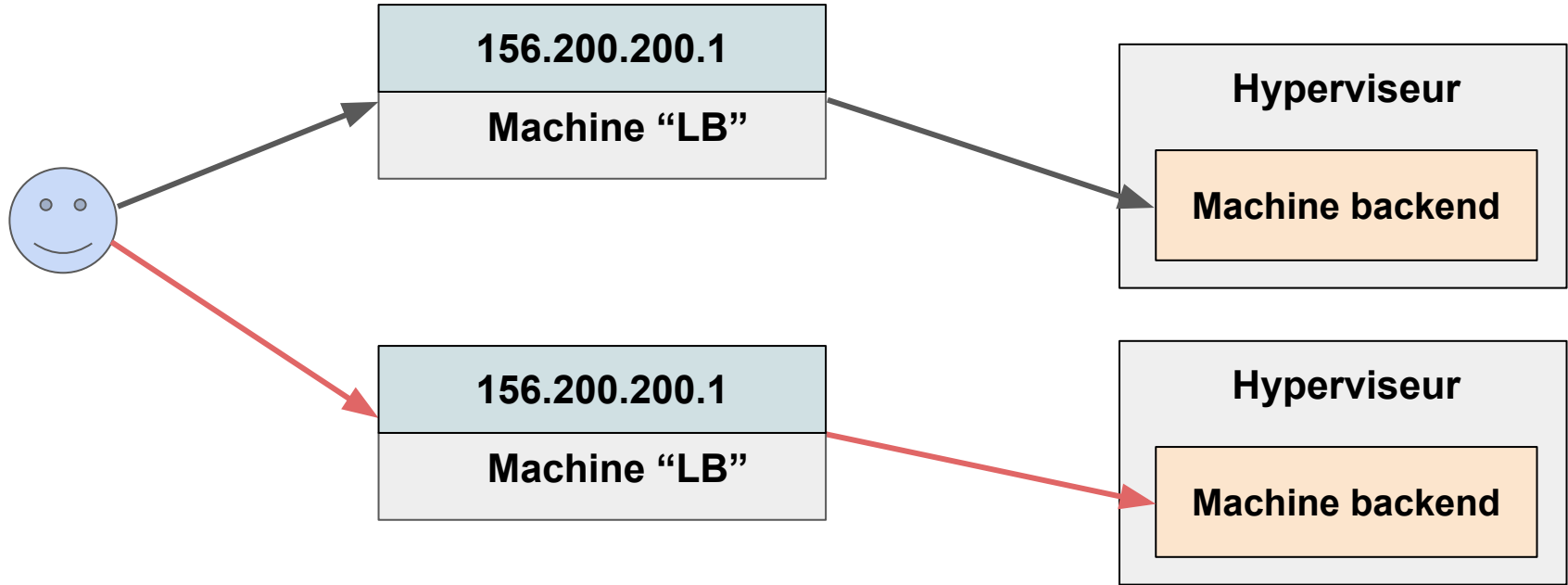
Le trafic sortant de la machine virtuel ne repasse jamais par le load balancer



# Vue d'ensemble



# En réalité



# Deux instances de LB pour chaque load balancer

- Tolérance aux pannes
- Meilleures performances
  - actif/actif
- IP du load balancer partagée par les deux machines

# Elastic IPs

- Les IPs des load balancers sont des Elastic IPs
  - Produit dans notre catalogue
- IP pouvant être facilement associées à une/plusieurs instances

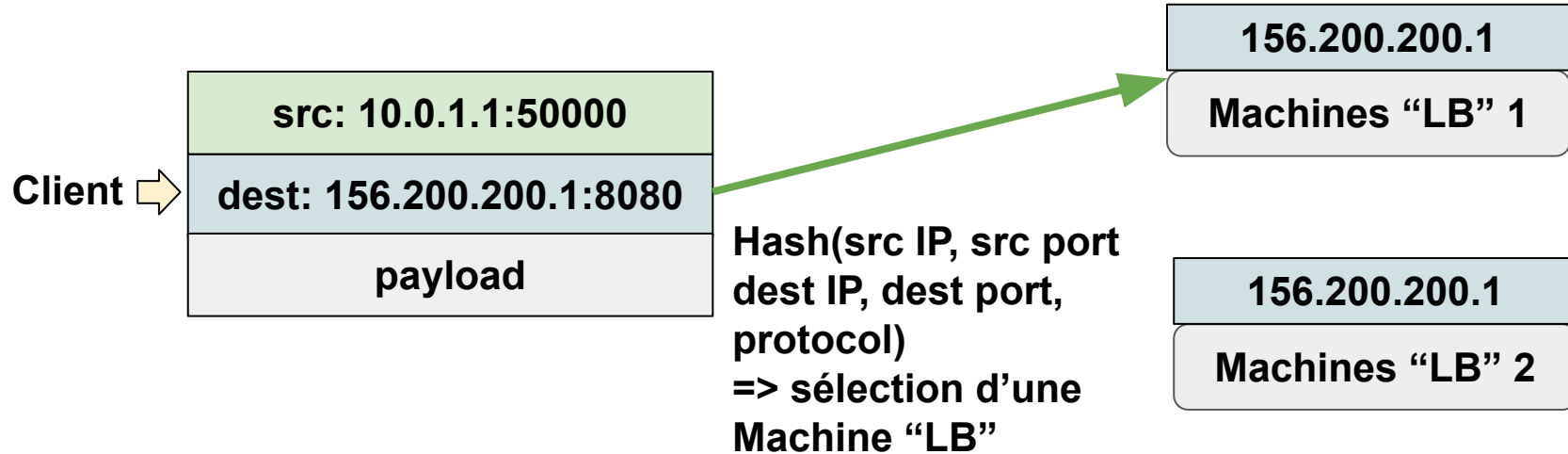


# Equal Cost Multi-Path

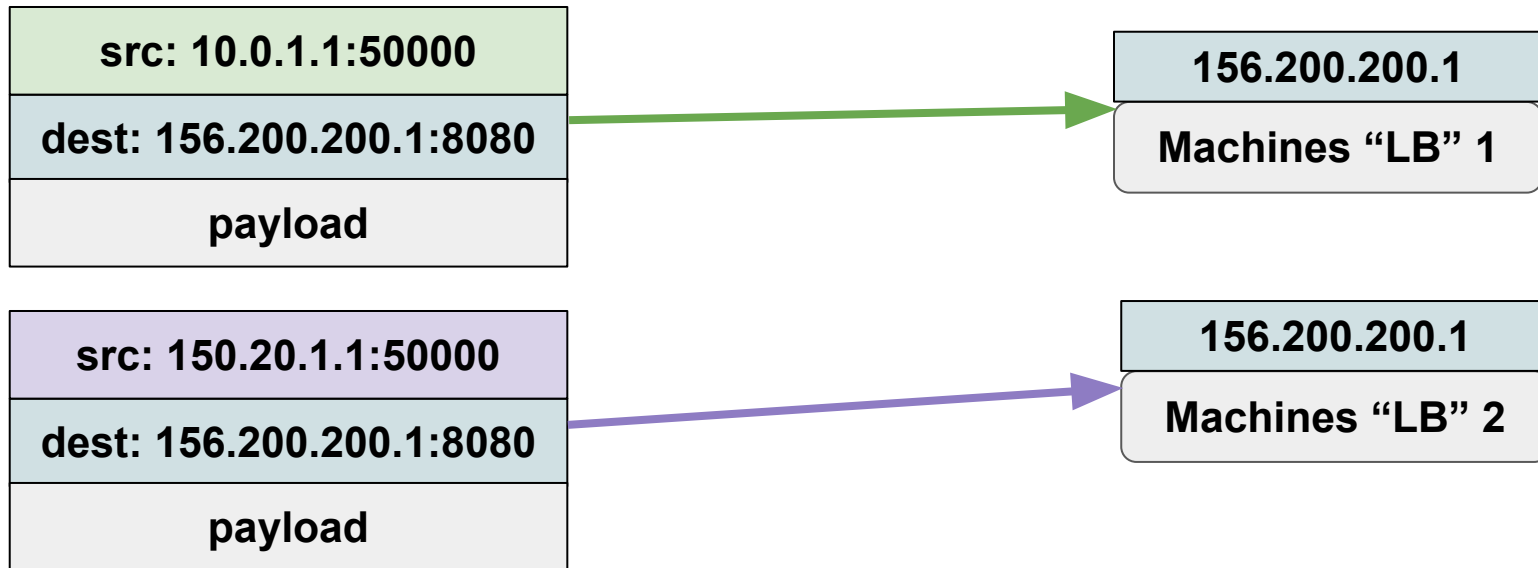
Plusieurs routes pour une même IP

BGP pour annoncer les routes

# Equal Cost Multi-Path

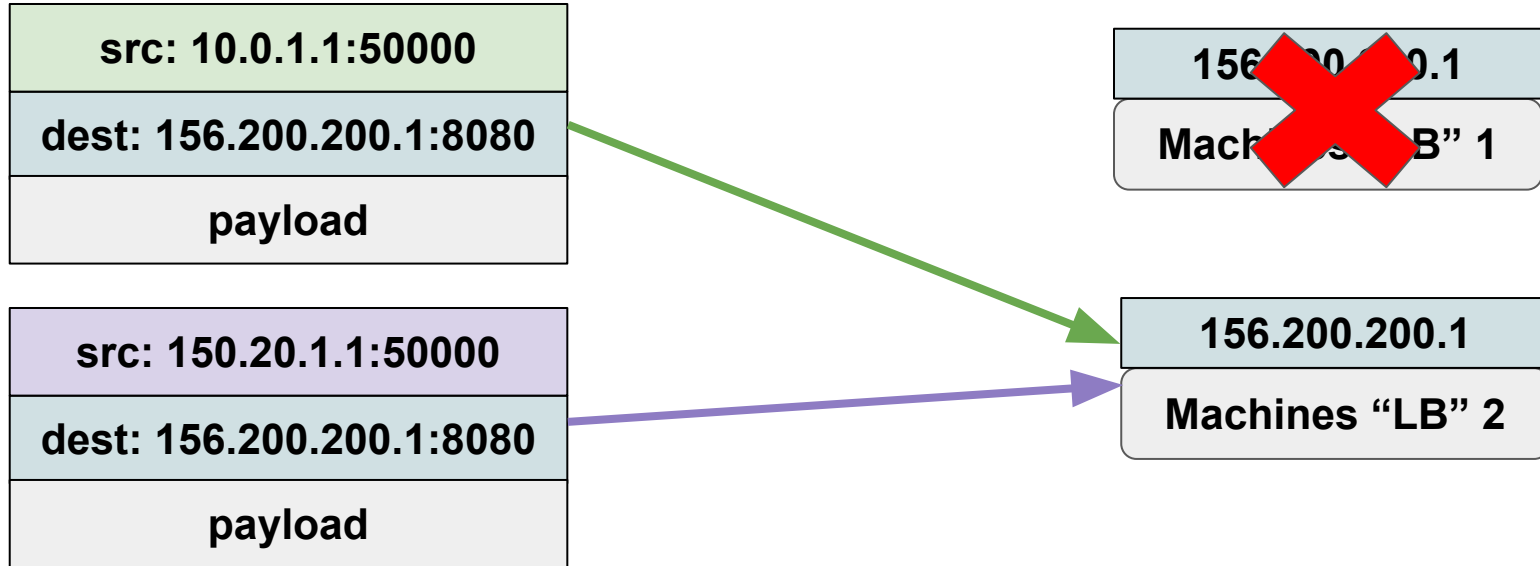


# Equal Cost Multi-Path



Chaque connexion est distribuée sur les machines "LB" en fonction du hash

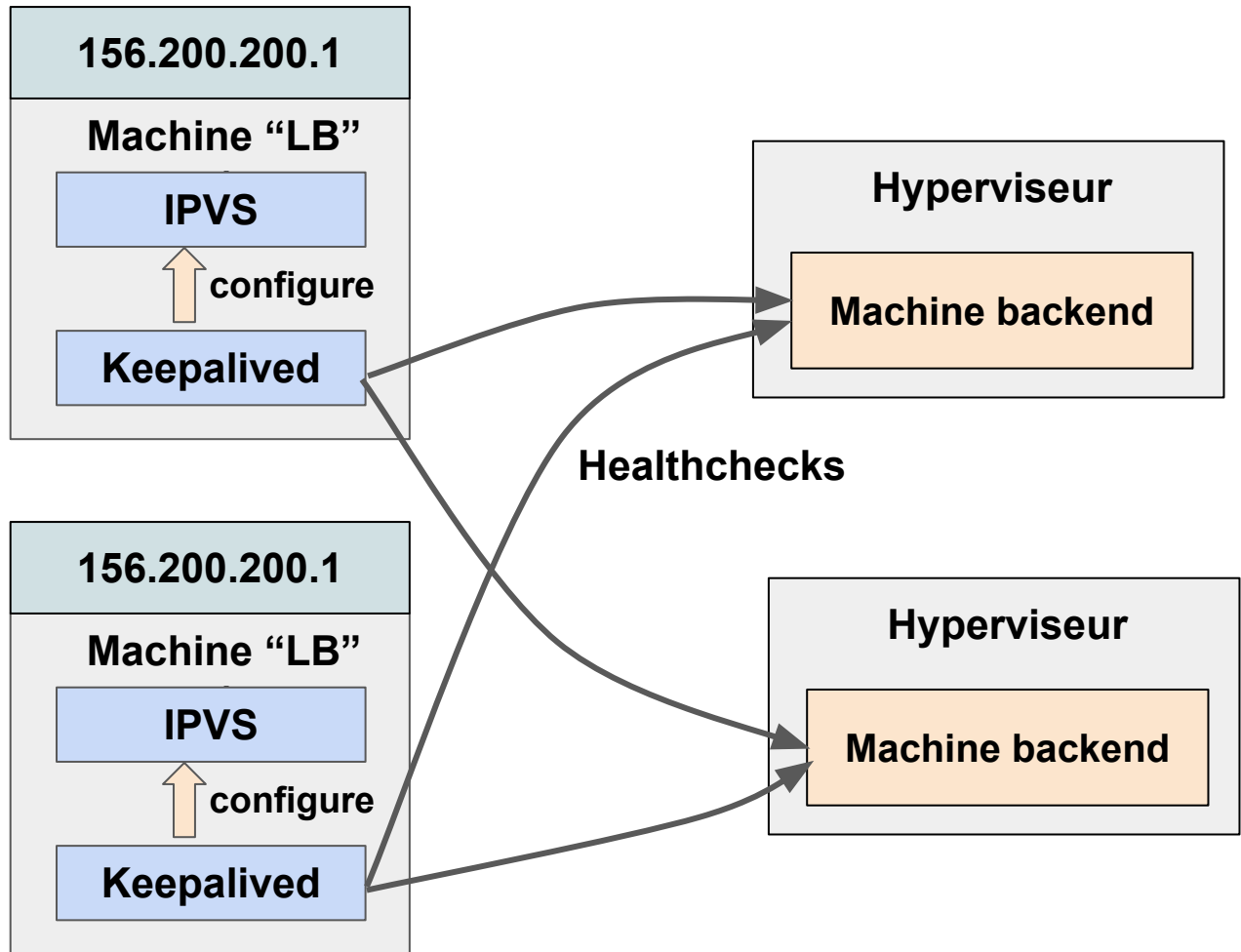
# Equal Cost Multi-Path



**En cas de crash d'une machine, reroutage vers les autres machines disponibles**

# Machines LB: Keepalived

- Exécute les healthchecks et pilote IPVS
  - Chaque load balancer client a sa propre instance de keepalived, sur deux machines distinctes
  - Des tonnes d'options
  - Très (très) léger
  - Fiable



# Stratégies de load balancing

IPVS/Keepalived supportent plusieurs stratégies  
de load balancing

# Round Robin

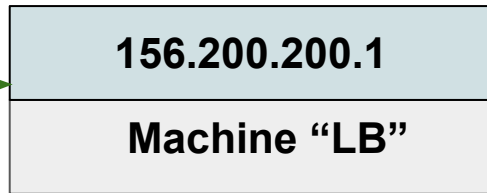
- Les connexions sont distribuées équitablement entre les serveurs backends
- Ajout optionnel d'un "poid" par serveur backend



# Source Hash

- Un hash sur l'IP source (et optionnellement le port source) est calculé
- Algorithmes généralement simples
  - Que se passe t-il si on ajoute ou supprime un backend ?

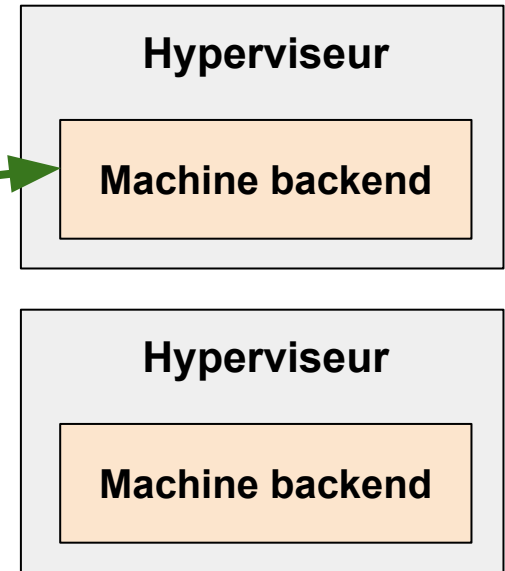
**source ip:  
10.2.3.4**



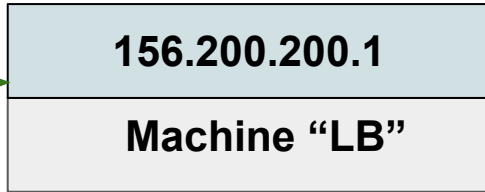
**hash(10.2.3.4) = 100**

**100 mod 2 = 0**

**=> Premier serveur choisi**



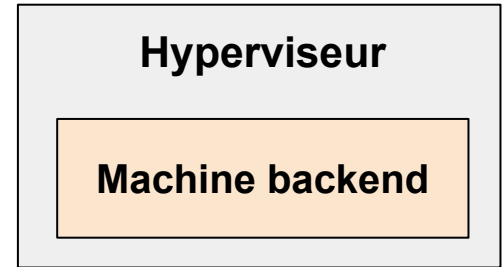
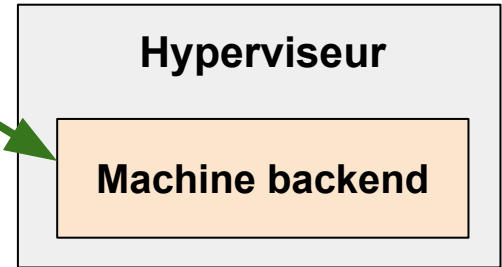
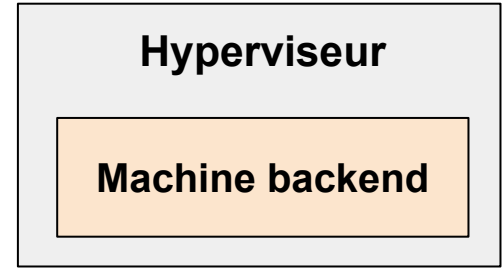
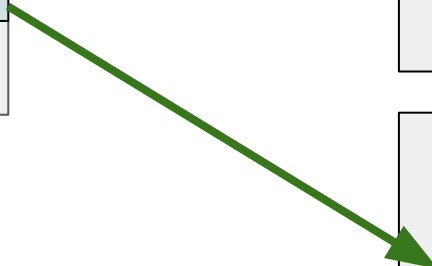
source ip:  
10.2.3.4



$\text{hash}(10.2.3.4) = 100$

$100 \bmod 3 = 1$

=> Second serveur choisi

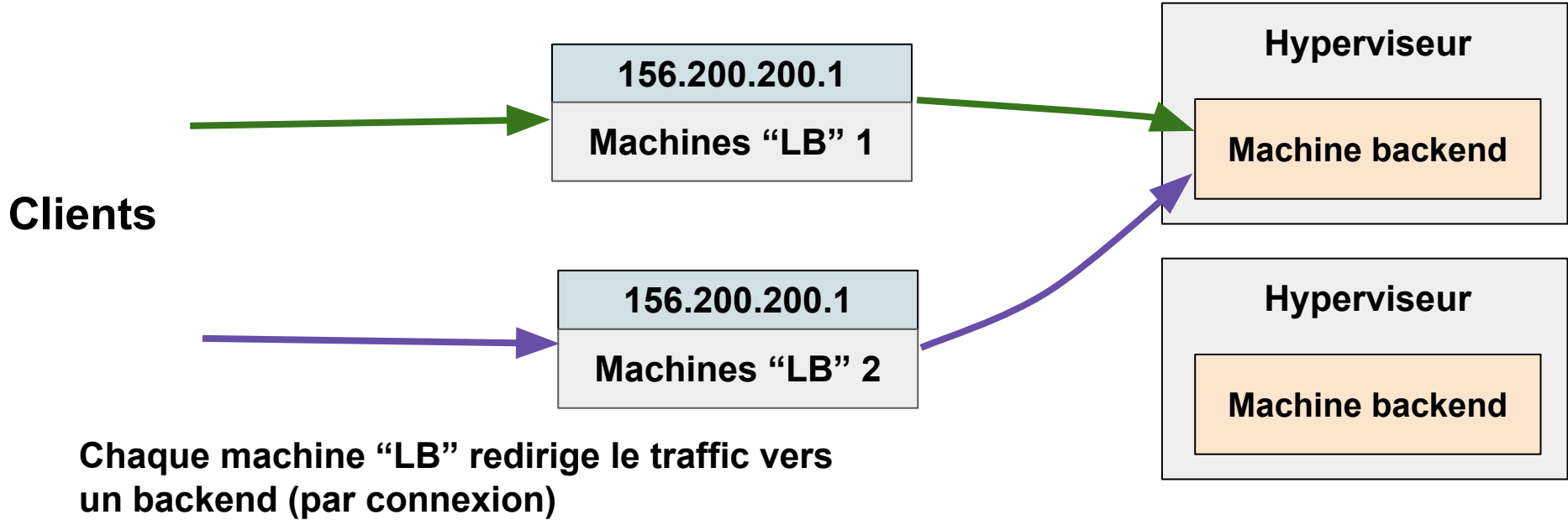


# Source Hash

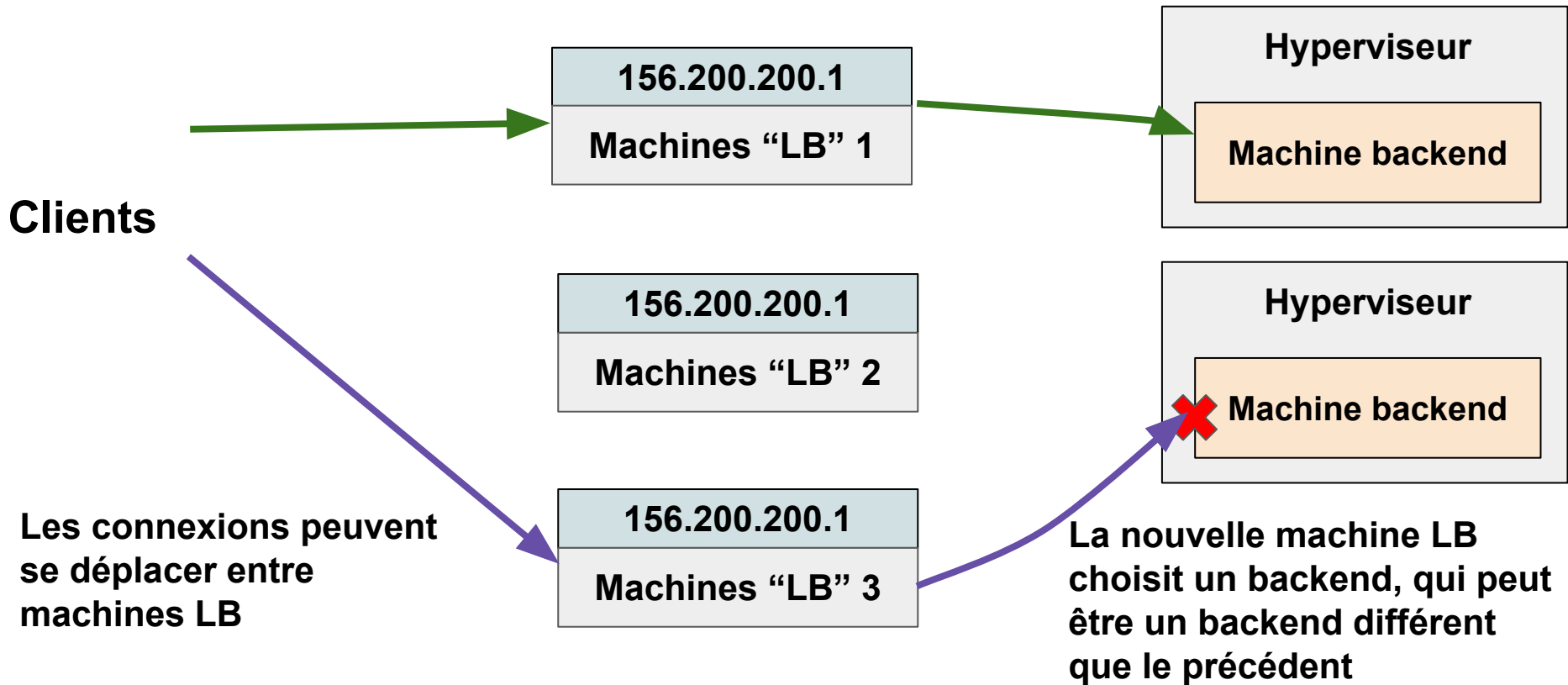
- En cas de modification des backends, la distribution est recalculée
  - Pour IPVS, cela arrive quand la connexion expire de la table des connexions IPVS

# Equal Cost Multi-Path: même problème

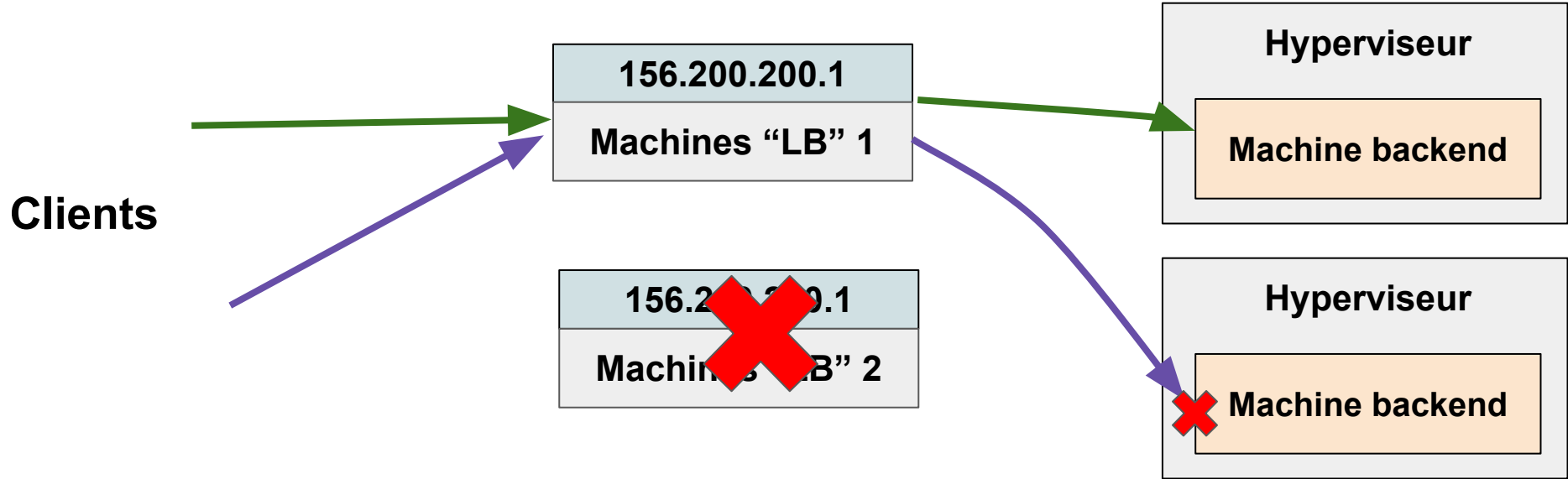
# Equal Cost Multi path



# Ajout d'une machine "LB"



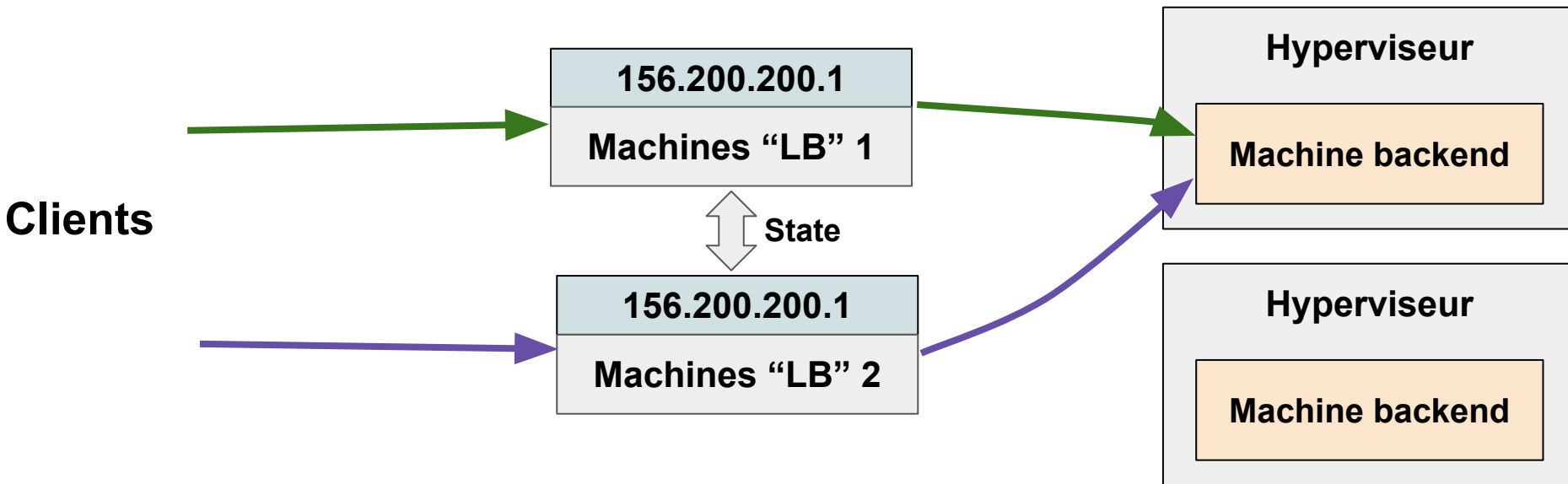
# Suppression d'une machine "LB"





# Partager la table des connexions ?

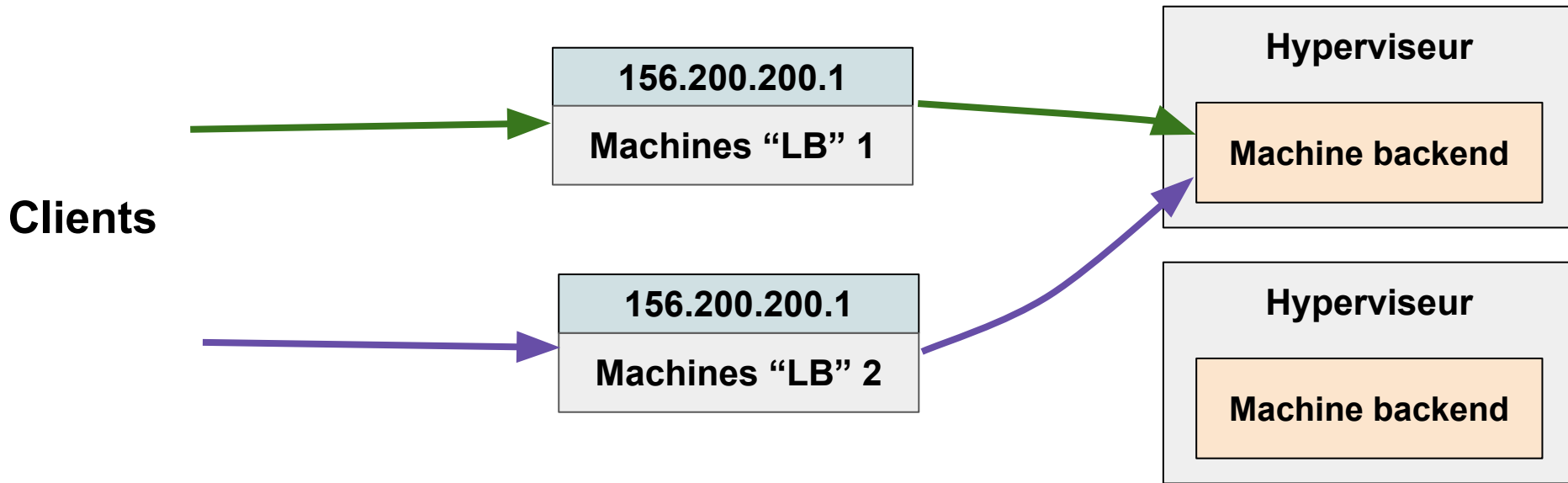
Complicqué, surtout avec IPVS



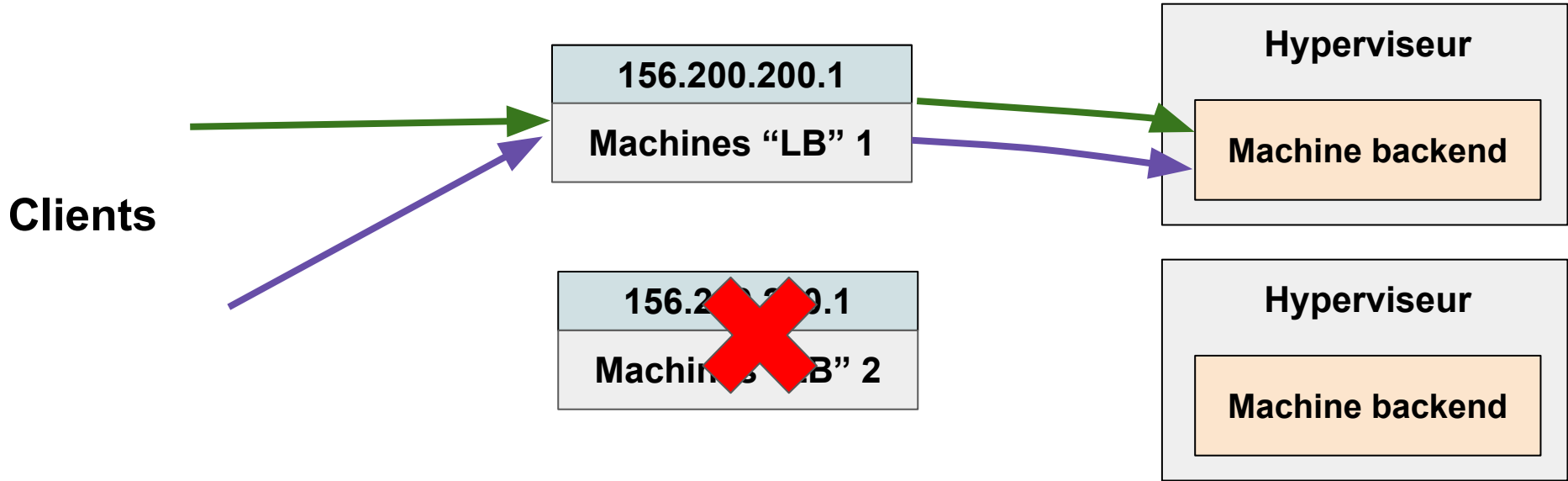
# Solution: consistent hashing

- Algorithmes pour éviter les problèmes en cas d'ajout/suppressions de machines
- Toujours router les connexions vers les mêmes machines backends quand il y a:
  - Ajout/Suppression de machines backends
  - Ajout/Suppression de machines LB
- IPVS implémente Maglev (créé par Google)

# Solution: consistent hashing



# Solution: consistent hashing



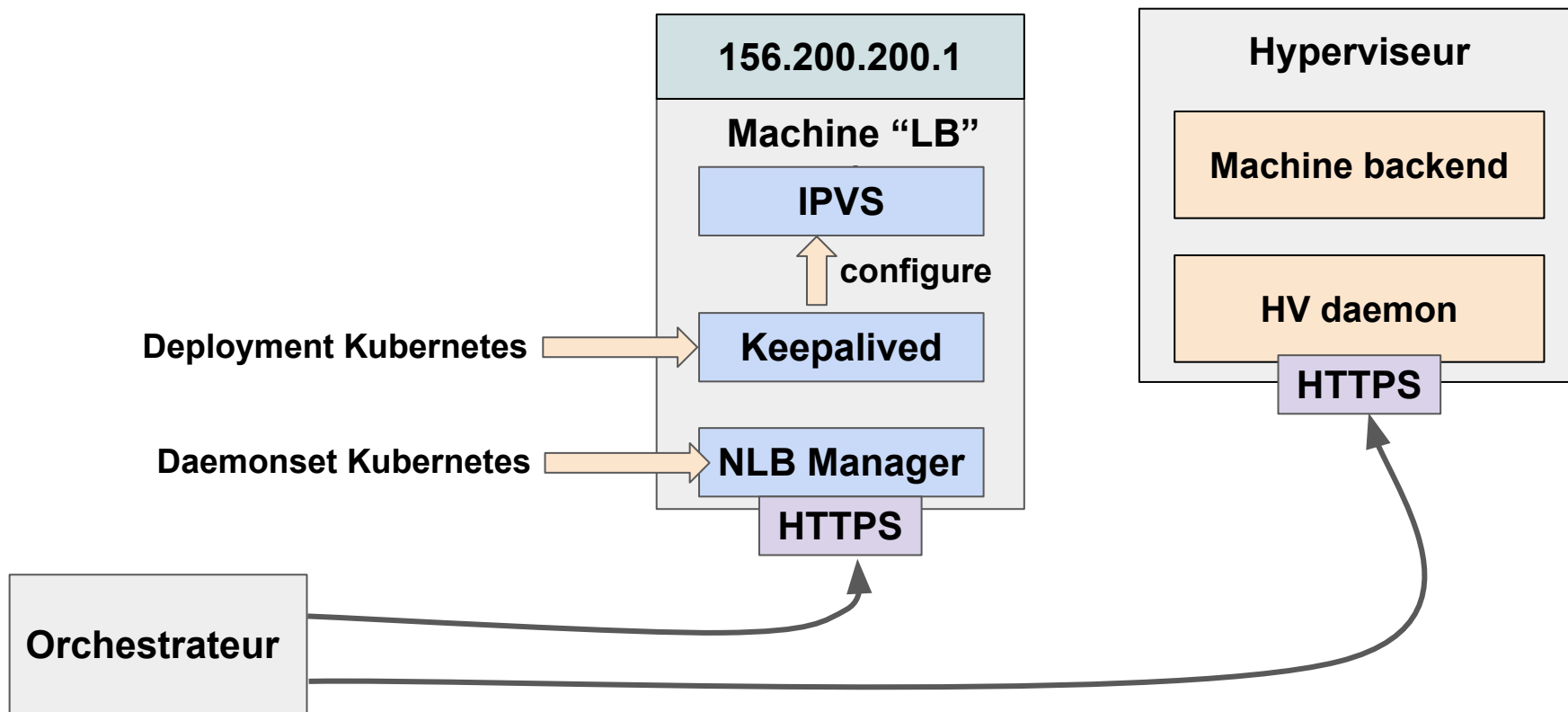
Toutes les machines "LB" prennent la même décision pour une même connexion.

L'ajout/suppression de backends a également un impact limité

# Keepalived tourne dans Kubernetes

- Les machines “LB” sont des workers spécifiques
- Permet de facilement déplacer des load balancers d'un noeud à un autre (en cas de panne par exemple)

# Vue d'ensemble



# NLB Manager

- Un DaemonSet tournant sur toutes les machines “LB”
  - Exposant une API HTTP
- Attache/détache les Elastic IPs des load balancers des machines
- Configure/reload Keepalived
- Récupère l'état des serveurs backends dans IPVS
- Périodiquement, reçoit une requête lui faisant vérifier l'état des load balancers/IP

# HV Daemon

- Tourne sur chaque hyperviseur
  - Expose aussi une API HTTP
- Permet d'ajouter ou supprimer les règles réseaux pour les load balancers
- Périodiquement, envoie toutes les règles de load balancing dans Kafka pour analyse/détection d'incohérences



# Orchestration

- La difficulté du cloud: converger (très) rapidement vers le bon état:
  - Base de données de l'orchestrateur
  - Règles sur les machines "LB" (NLB Manager)
  - Règles sur l'hyperviseur (HV Daemon)
- De nombreux changements d'états
  - Actions utilisateurs
  - (live) migration de machines virtuelles
  - Incidents

# Notre librairie d'orchestration

- Utilisée par plusieurs projets (dont le load balancer)
- Un moyen simple et efficace d'écrire des orchestrateurs
- 100 % Clojure



# Orchestrateur: le store

- Une base de données (SQL) contenant l'état attendu
- Une API pour manipuler le store
  - Basé sur <https://github.com/exoscale/seql/>

# Orchestrateur: l'inventaire

- Récupérer l'état du monde pour une entité
- Décide quoi faire en fonction de l'état actuel de l'entité

# Orchestrateur: exécution du job

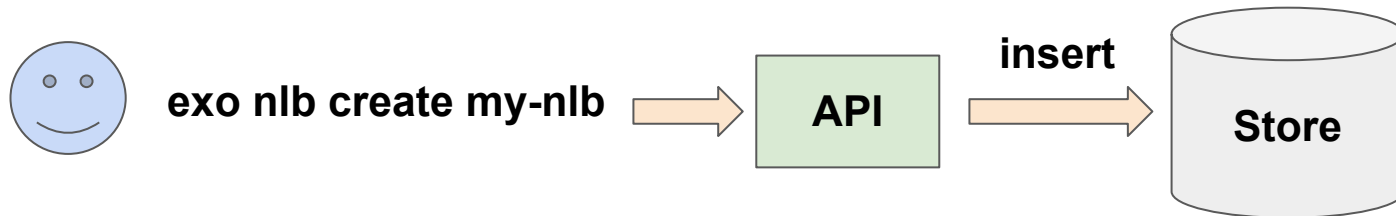
- Un job démarre et synchronise les différents systèmes en fonction de l'inventaire
  - <https://github.com/exoscale/ablauf>

# Réconciliation

Inventaire + exécution du job

# Un exemple: création d'un load balancer

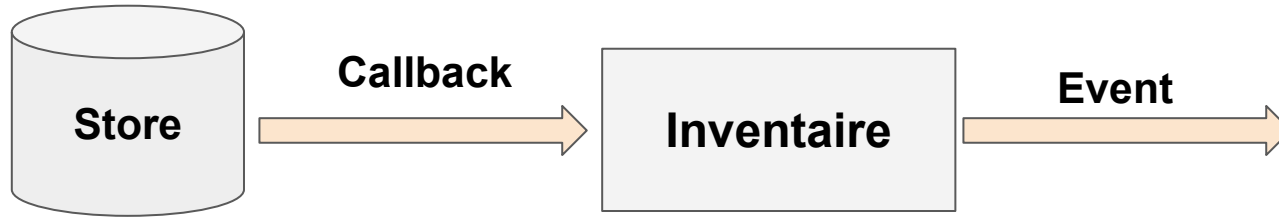
# Etape 1: un NLB est créé dans le store



ID	fbc40446
Name	my-nlb
Description	Load balancer talk
IP Address	null
<b>state</b>	<b>creating</b>
...	...



# Etape 2: appel de la fonction d'inventaire

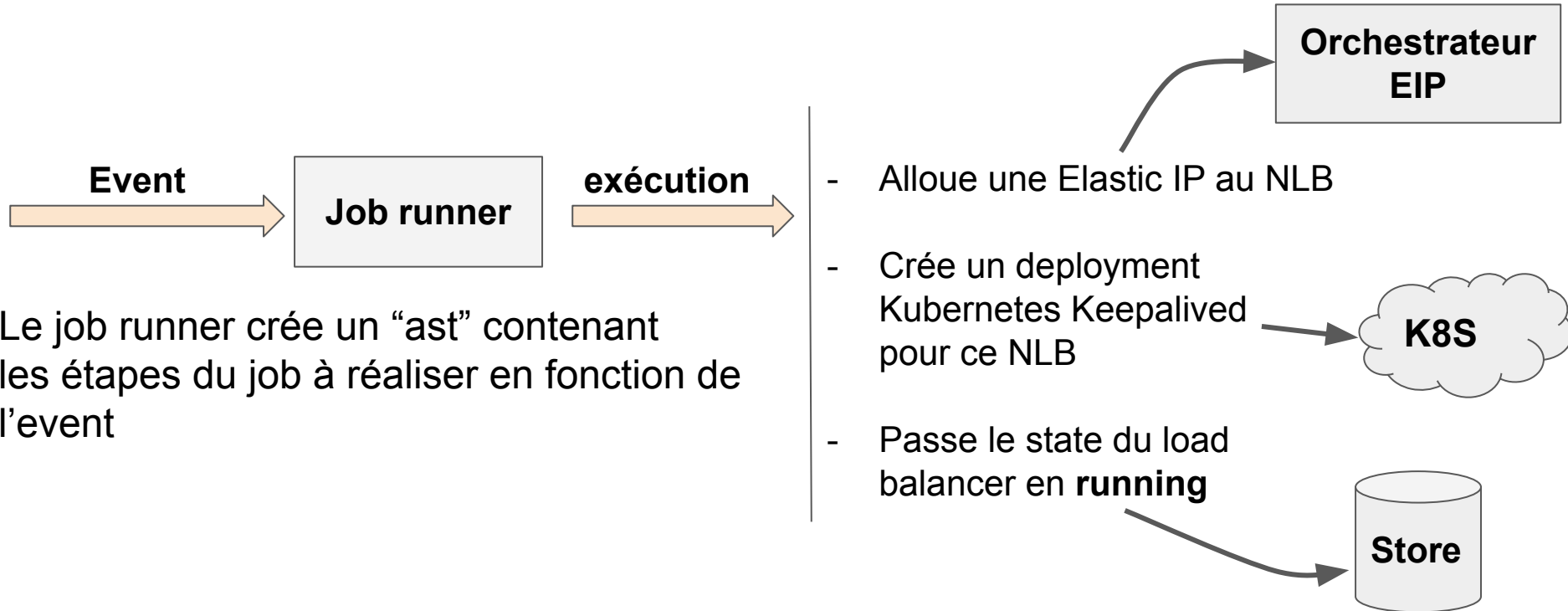


Un NLB en **state = creating**

=> Génération d'un event de type "Create"

L'event contient toutes les informations disponibles sur ce NLB

# Etape 3: construction et exécution du job

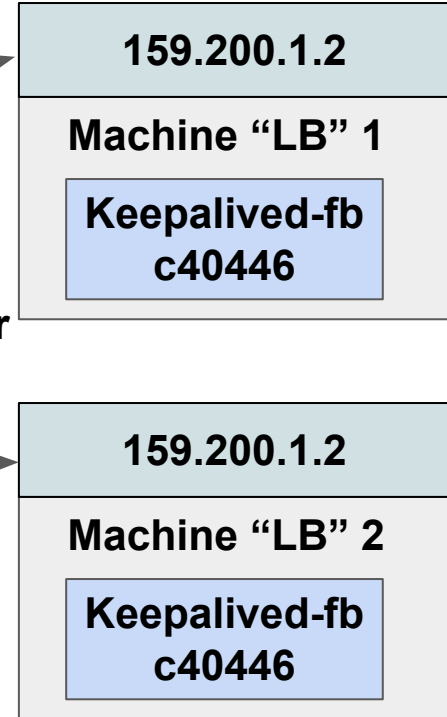


# Le load balancer est créé

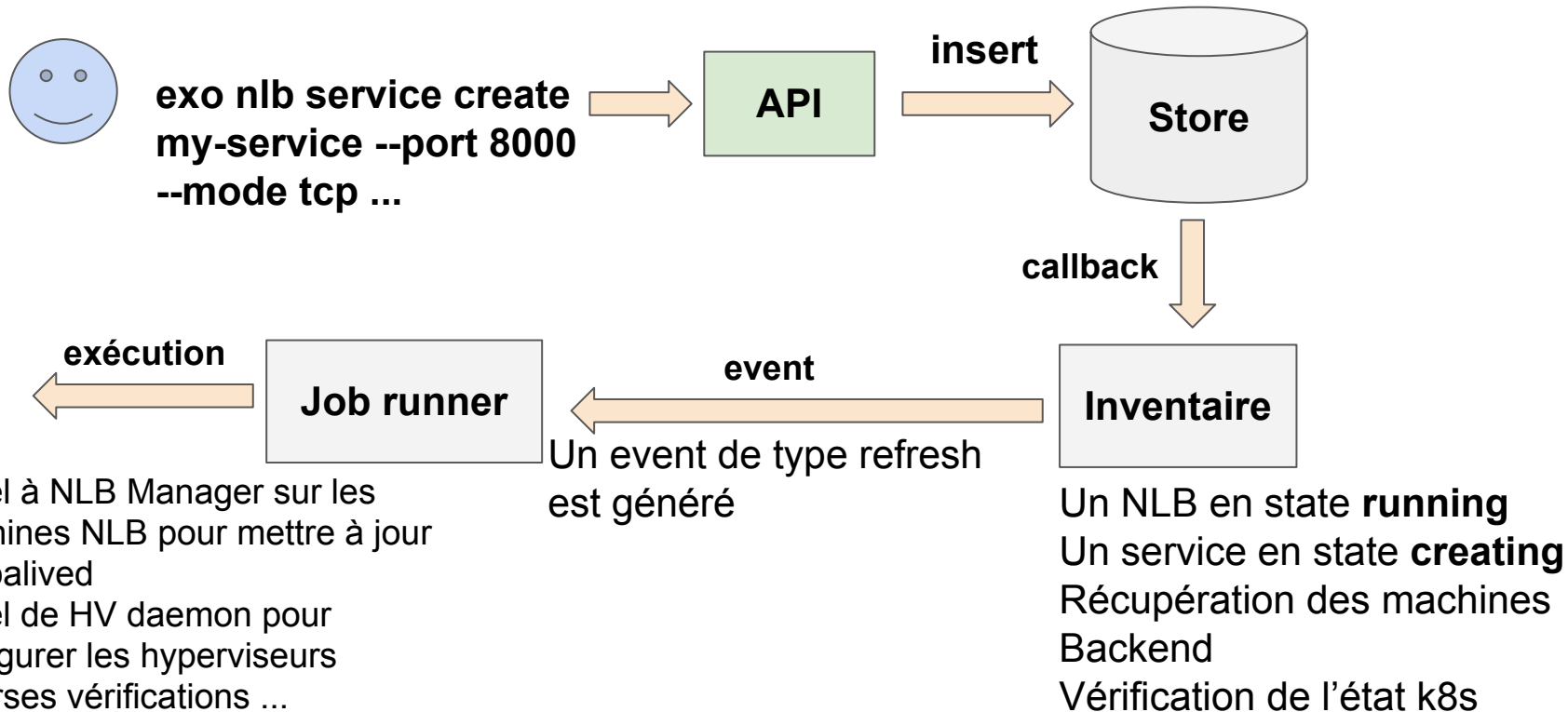
**store**

ID	fbc40446
Name	my-nlb
Description	Load balancer talk
IP Address	<b>159.200.1.2</b>
<b>state</b>	<b>running</b>
...	...

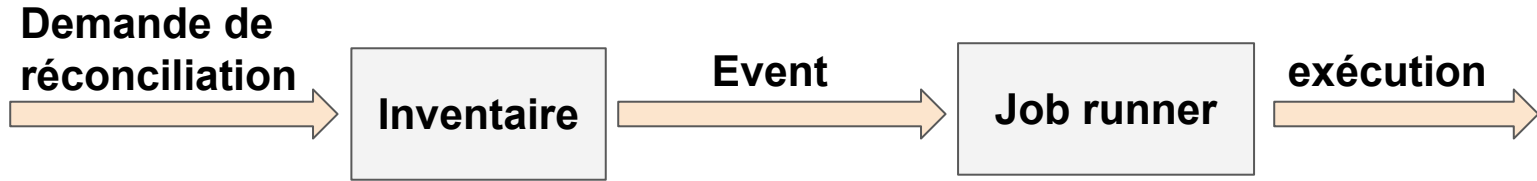
IP Configurée par  
NLB Manager



# Ajoutons un service

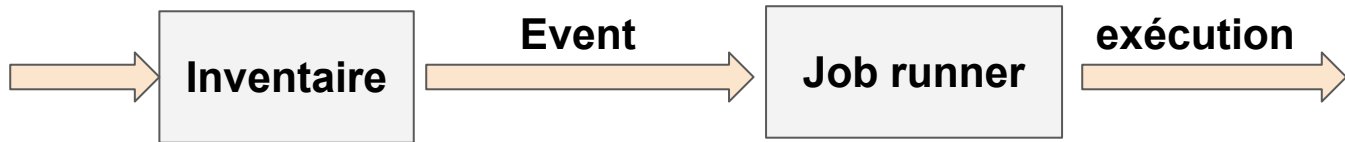


# Toujours le même principe



# Mieux vaut réconcilier trop que pas assez

- Events Kafka de machines (création/suppression)
- Vérifications périodique
- Events Kubernetes
- Events générés par NLB Manager/HV daemon
- Action sur le load balancer
- Action déclenchée par un opérateur (admin)
- ....



# Une réconciliation ne coûte pas cher

- Dès que le système a un doute, il déclenche une réconciliation
- Les sous systèmes savent aussi détecter des états incohérents et demander des réconciliations

# Orchestrateur: détails

- Garantie qu'une même entité ne peut pas être réconciliée plusieurs fois en parallèle
  - "File d'attente" en cas de réconciliations multiples
- Notion de priorité des événements
- Déduplication des événements
  - Plusieurs événements en priorité basse => une seule réconciliation



# Orchestrateur: détails

- Gestion des erreurs
  - Erreur de réconciliation => retry => Pagerduty => Oncall
- Design modulaire à tous les niveaux
  - Car chaque orchestrateur a ses propres besoins

# Merci

## Questions ?