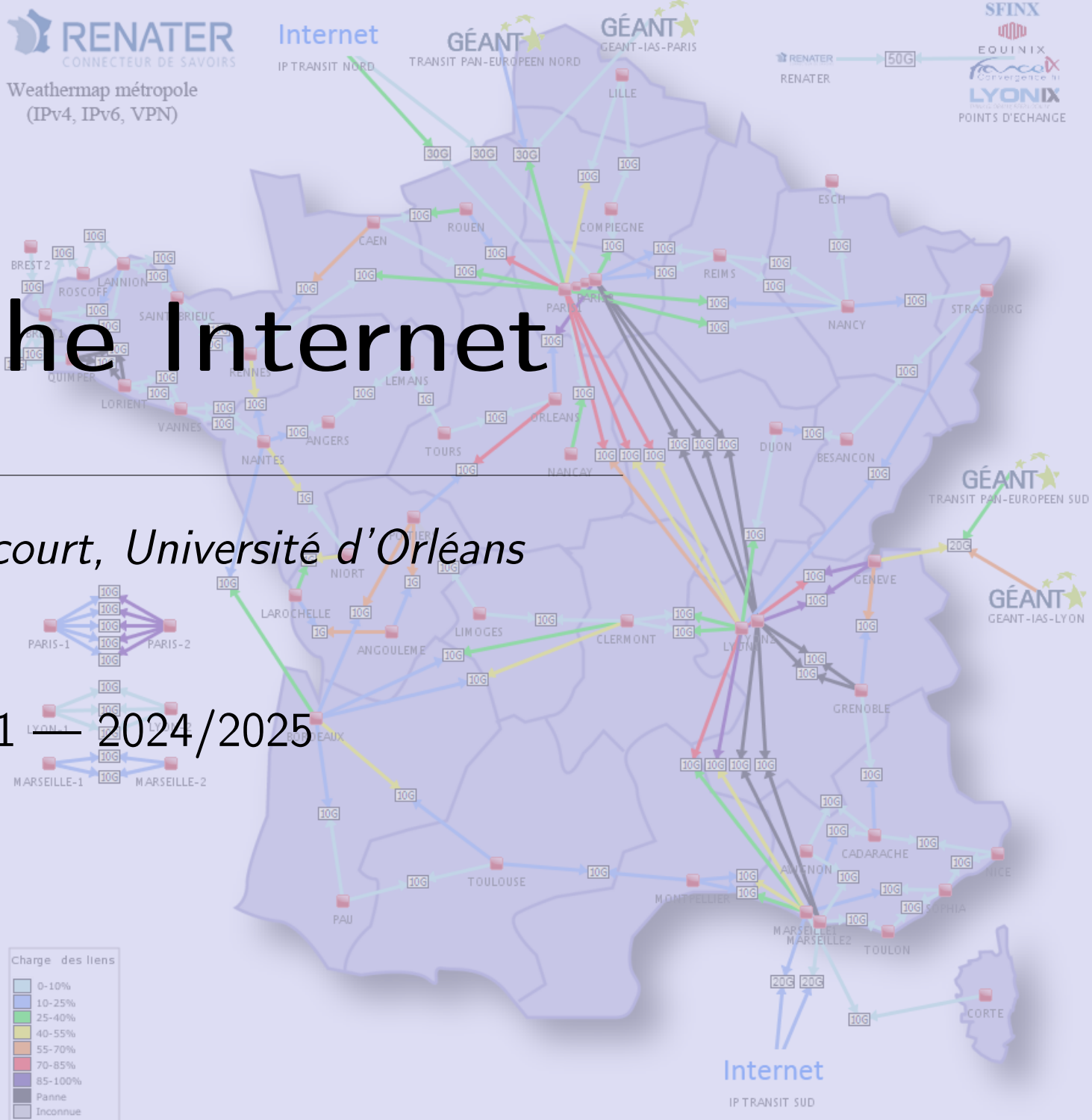
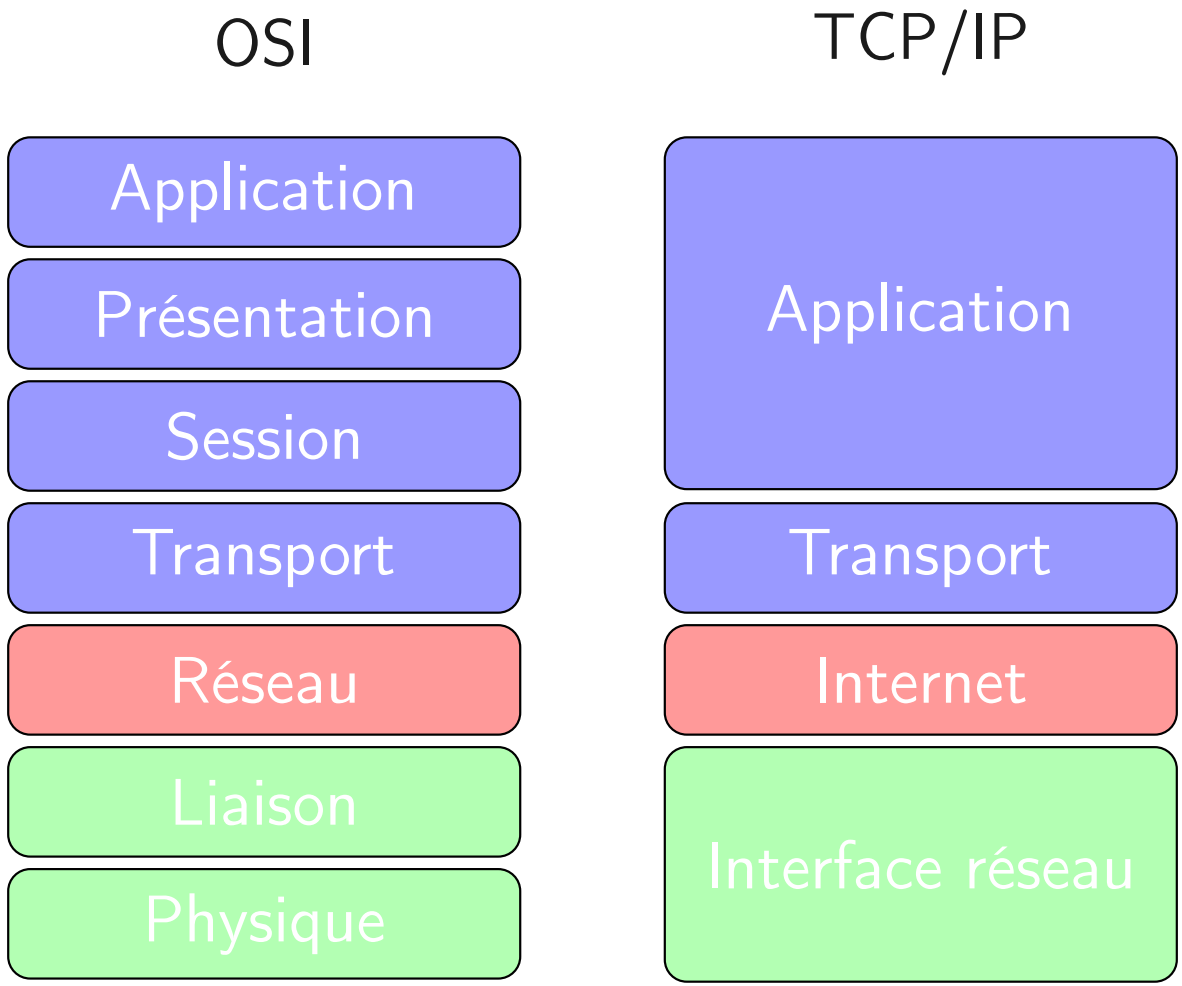
Couche Internet

Martin Delacourt, Université d'Orléans

L3 Réseaux 1 — 2024/2025



Les modèles

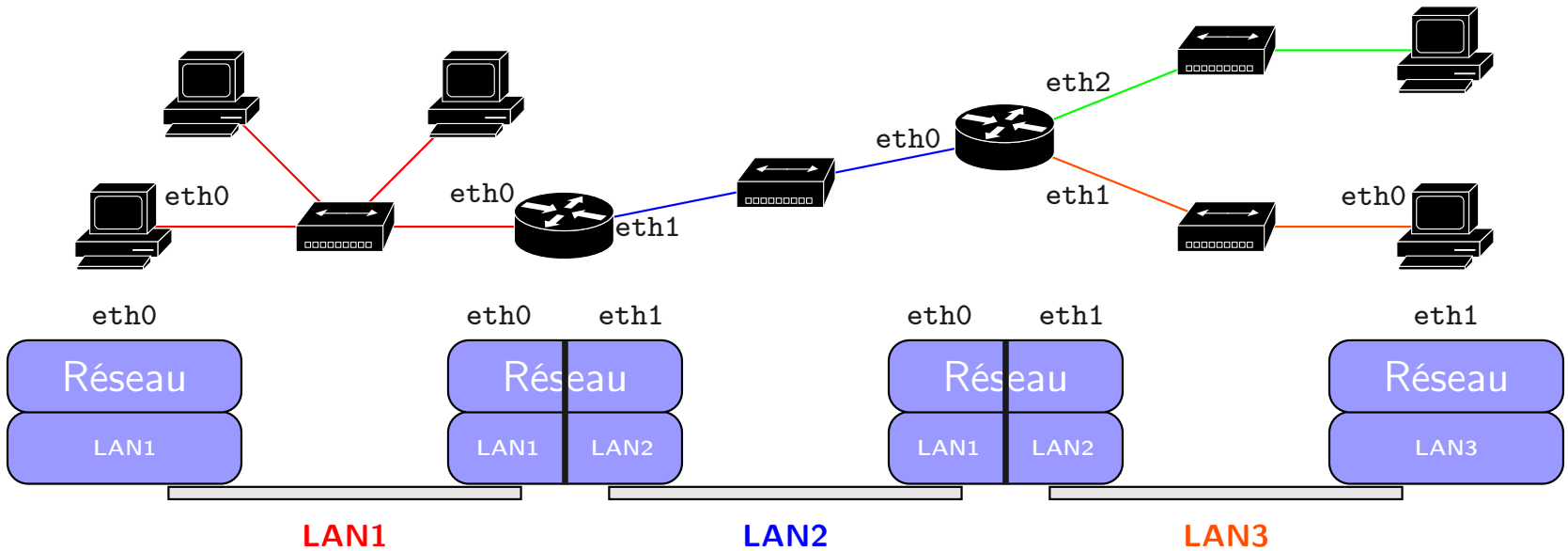

OSI
TCP/IP
Application
Présentation
Session
Transport
Réseau
Liaison
Physique
Application
Transport
Internet
Interface réseau

Rôle de la couche réseau

Chaque machine doit pouvoir contacter n'importe quelle autre machine sur le réseau. Pour cela, qu'elle soit sur le même réseau local ou non, il faut pouvoir :

- l'identifier : adressage IP ;
- la joindre : routage.

Interconnexion de LAN



- Chaque interface a une adresse IP (e.g. DHCP) ;
- tous les hôtes en bord de réseau connaissent une passerelle ;
- comment remplir les tables de routage des routeurs ?

Routeur : double mission

Les routeurs doivent :

- transférer les paquets (forwarding) : `plan de données`.
- établir la table de routage (routing) : `plan de contrôle`.

Comment choisir le chemin ?

- Comment trouver le plus court chemin ?
- Comment mesurer la distance ?
- Comment gérer les évolutions du réseau ?

Routage dynamique

Algorithmique de graphe

L'algorithme doit être :

- fiable, pas de paquet perdu à cause du routage.
- simple, décision facile à prendre, précalcul peu coûteux (temps et occupation du réseau).

Il doit pouvoir s'adapter au réseau :

- mise à jour si un routeur disparaît du graphe.
- changement de route si le réseau devient surchargé.

Algorithme au choix

AS : défini par une politique de routage commune.

Plusieurs algorithmes coexistent : à l'intérieur des AS, entre AS...

Les options :

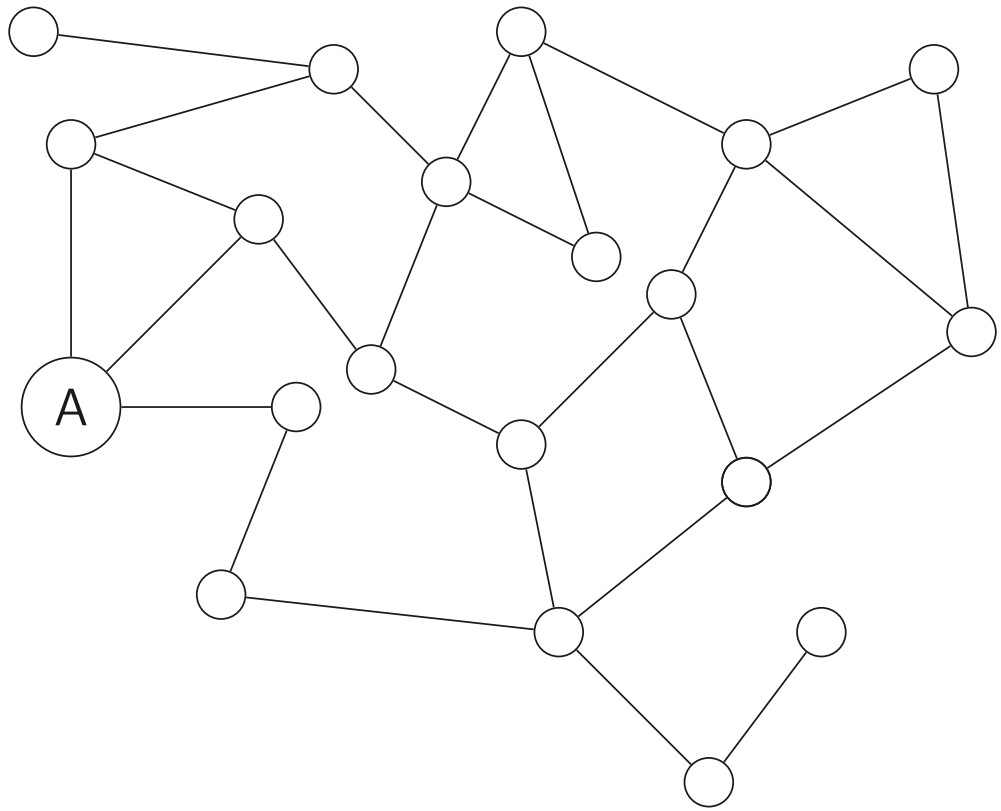
- Algorithme centralisé
- Algorithme distribué
  - à connaissance globale
  - à connaissance locale

Algorithme par inondation

- Chaque paquet est envoyé en broadcast sur l'ensemble du réseau.
- Tous les routeurs transmettent à tous leurs voisins.
- TTL pour éviter les boucles infinies (diamètre d'internet ?)
- Fiabilité, simplicité d'usage

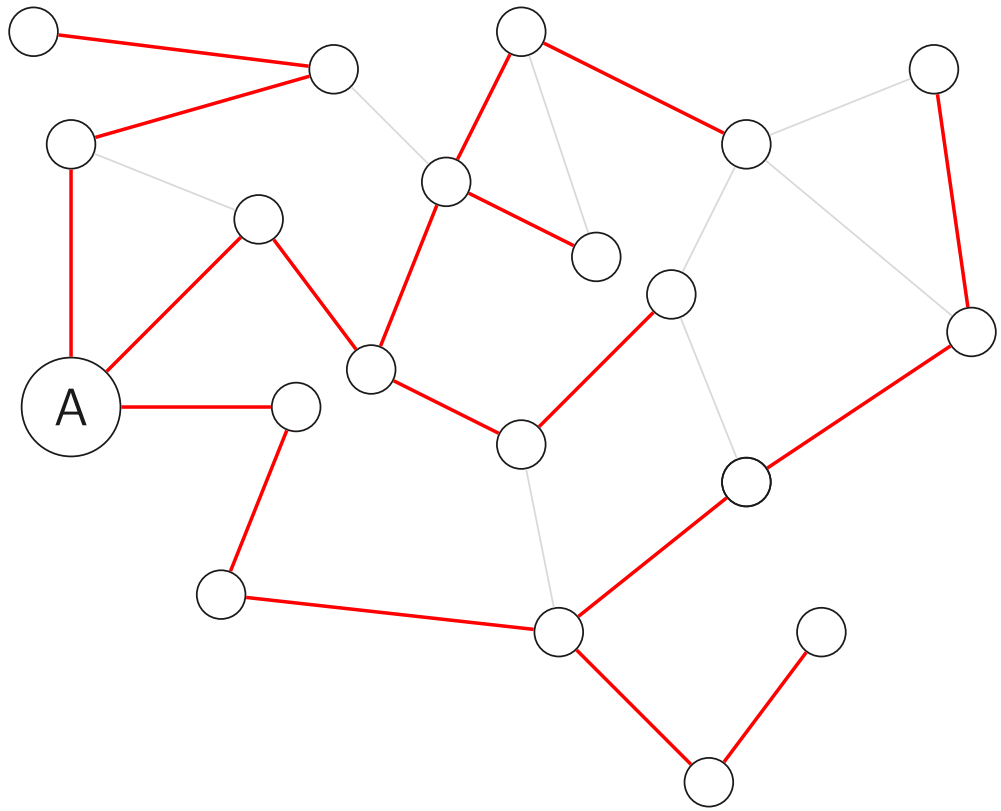
Comment choisir le chemin ?

Arbre des plus courts chemins



Comment choisir le chemin ?

Arbre des plus courts chemins

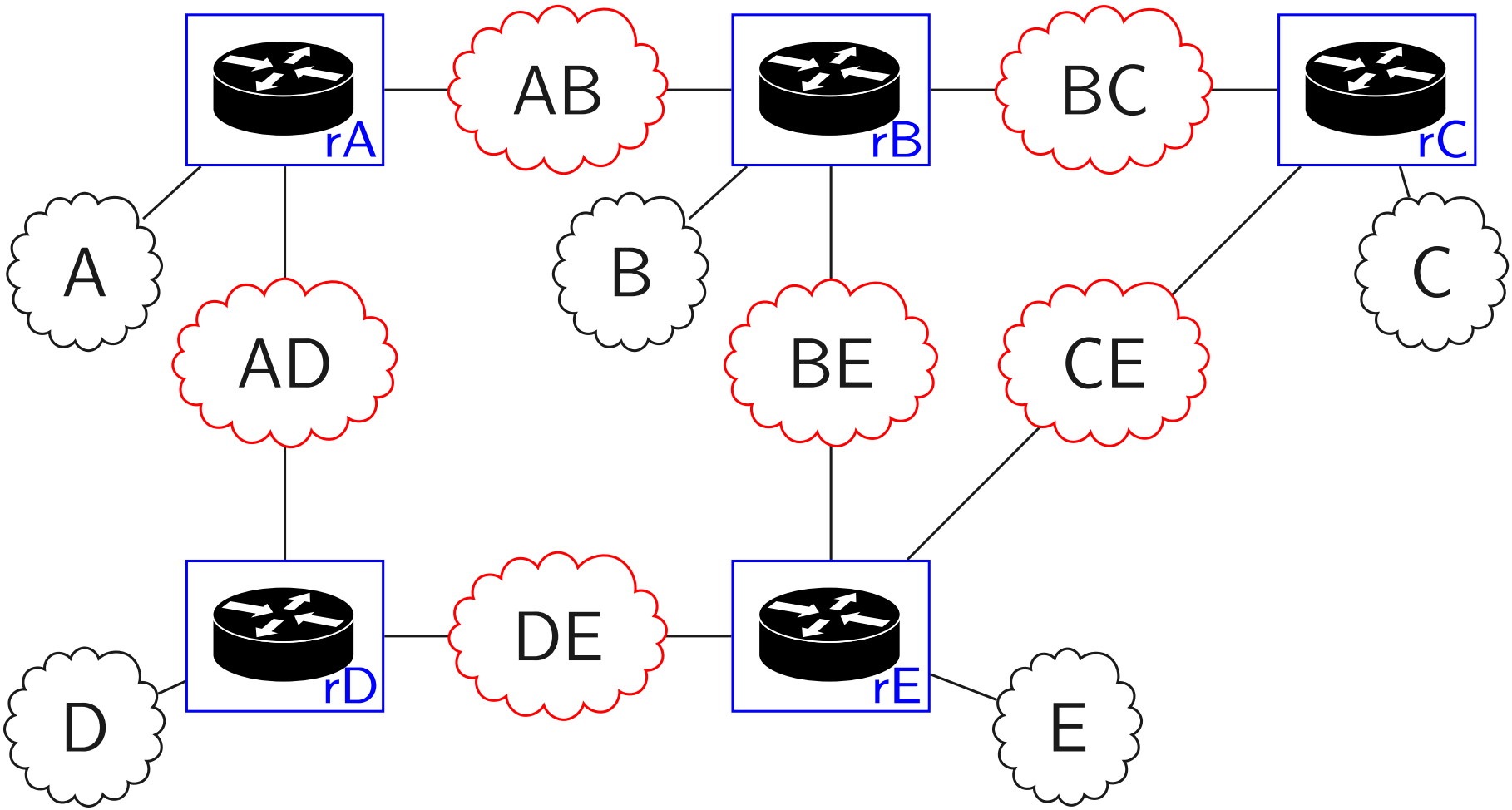


Comment choisir le chemin ?

Chaque routeur doit établir son propre arbre. Quelle métrique ?

- Corrélée au délai. Calcul par tests.
- Corrélée à la congestion : contournement des zones noires. Bison Futé ?
- Corrélée à la capacité : plus elle est grande, plus la distance est faible.
- Sans information : distance de 1 par saut.

Le réseau test

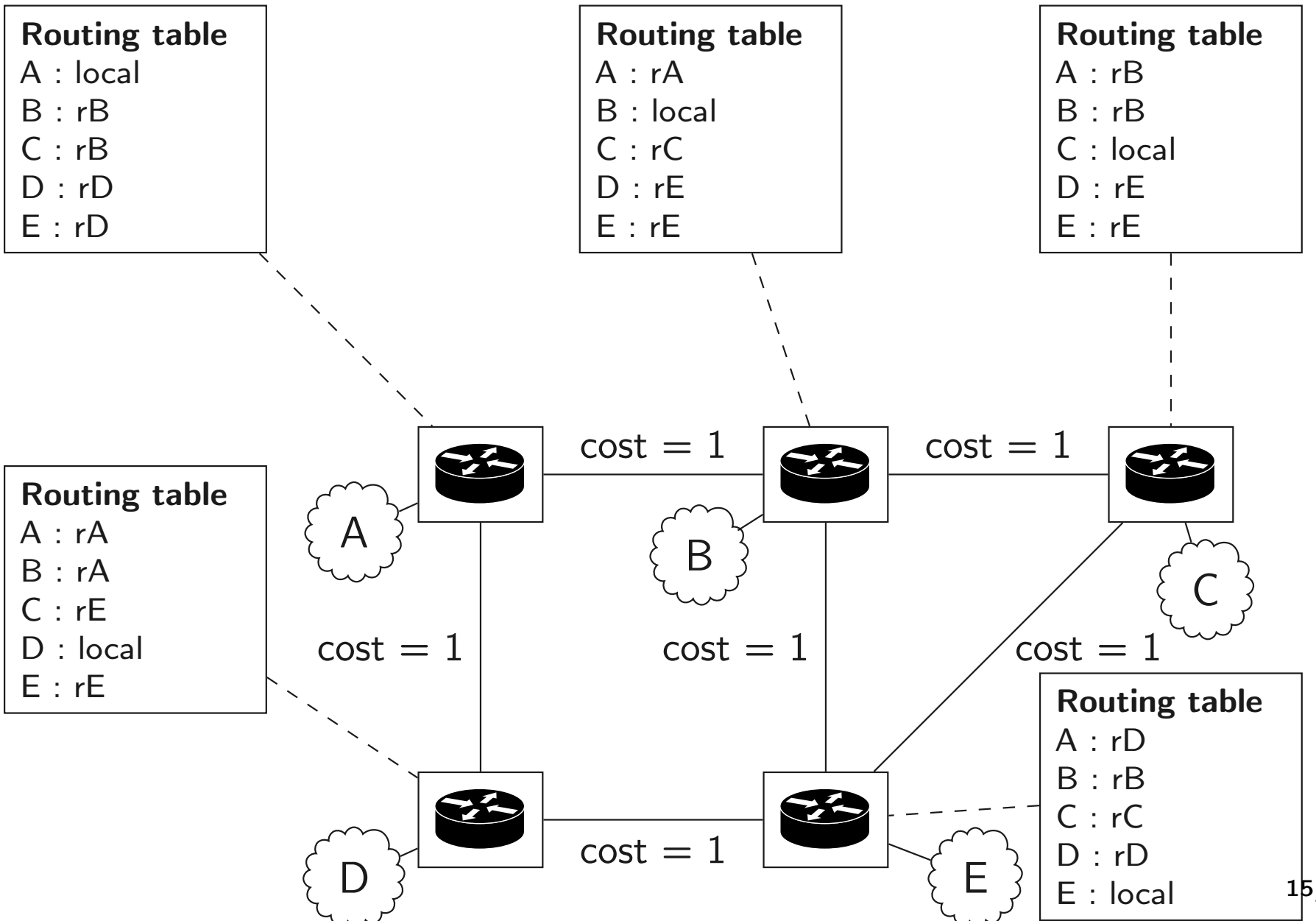


Note de lecture des slides

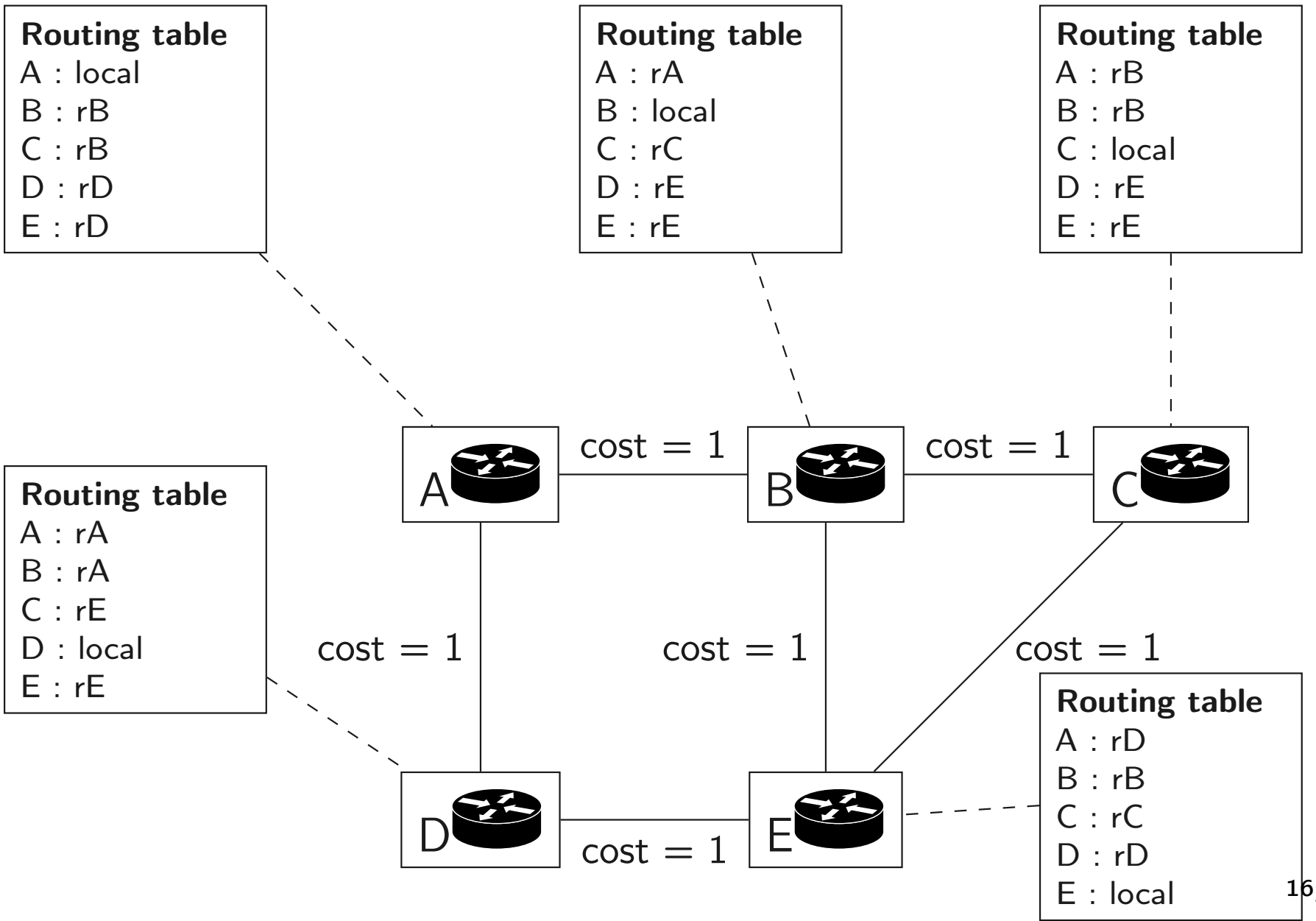
Dans la suite, les tables de routage sont simplifiées, en particulier les interfaces ne sont pas mentionnées (on peut ici les déduire sans ambiguïté).

On notera la différence entre tables de routage (connues depuis le chapitre précédent) et tables RIP que nous allons découvrir.

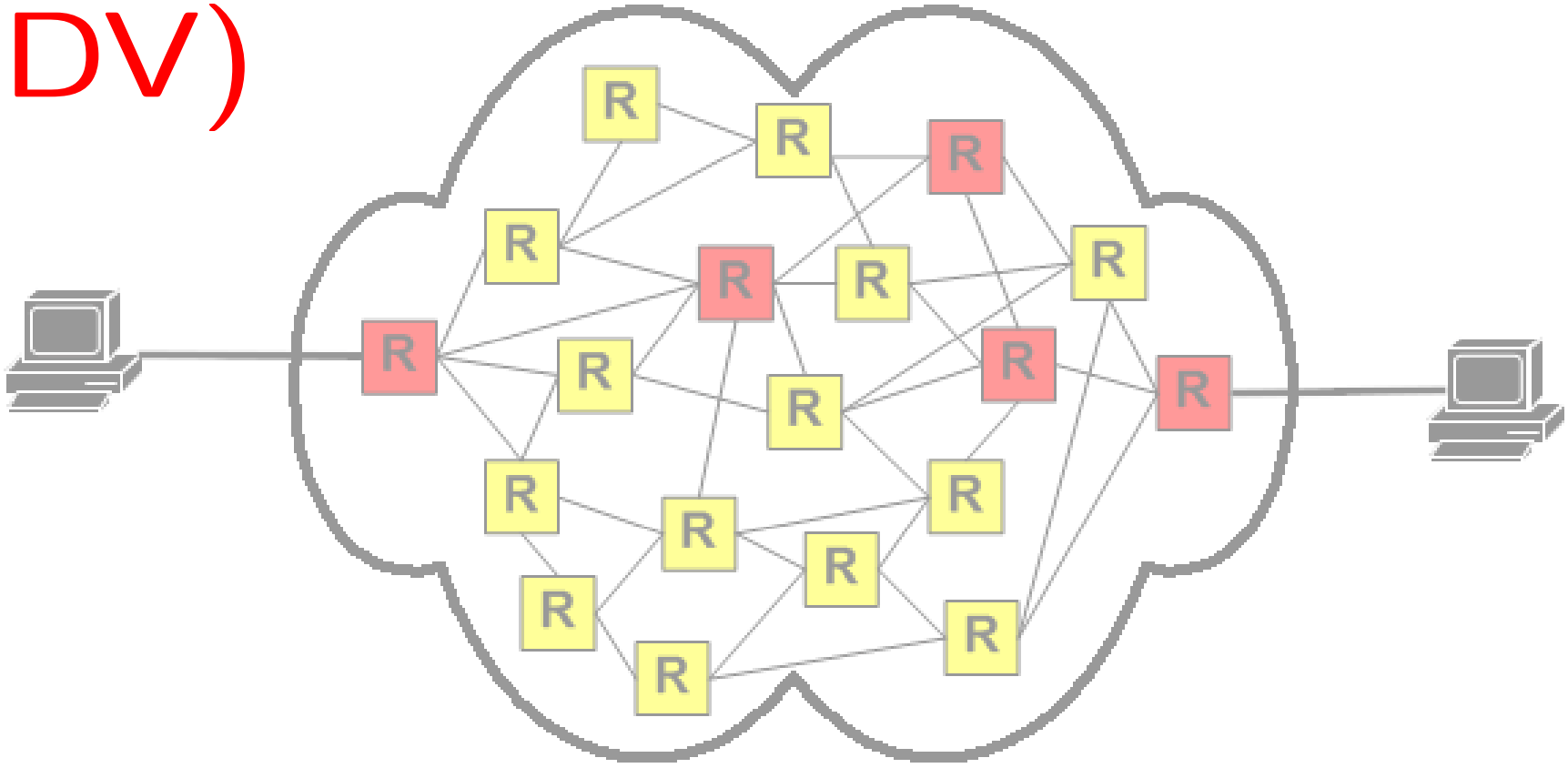
Les tables de routage désirées



Les tables de routage désirées



Vecteurs de distance (DV)



Bellman-Ford

Première option : distribué à connaissance locale.

- Algorithme de `Bellman-Ford` pour calculer les plus courts chemins dans le graphe.
- Chaque routeur connaît le coût de ses liens (calcul ou hypothèse).
- Chaque routeur transmet à ses voisins la liste des destinations connues avec leurs distances.
- À la réception, le routeur met à jour sa table de routage.

Notations

Dans ce qui suit :

- `R` est la table de routage à vecteur distances.
- `d` est utilisé pour un réseau destination.
- Les `link` sont les domaines de collision sur lesquels on communique.

Les vecteurs distance sont envoyés par un routeur sur un domaine de collision (Link), indifféremment à tous les routeurs participant au protocole RIP sur ce domaine de collision.

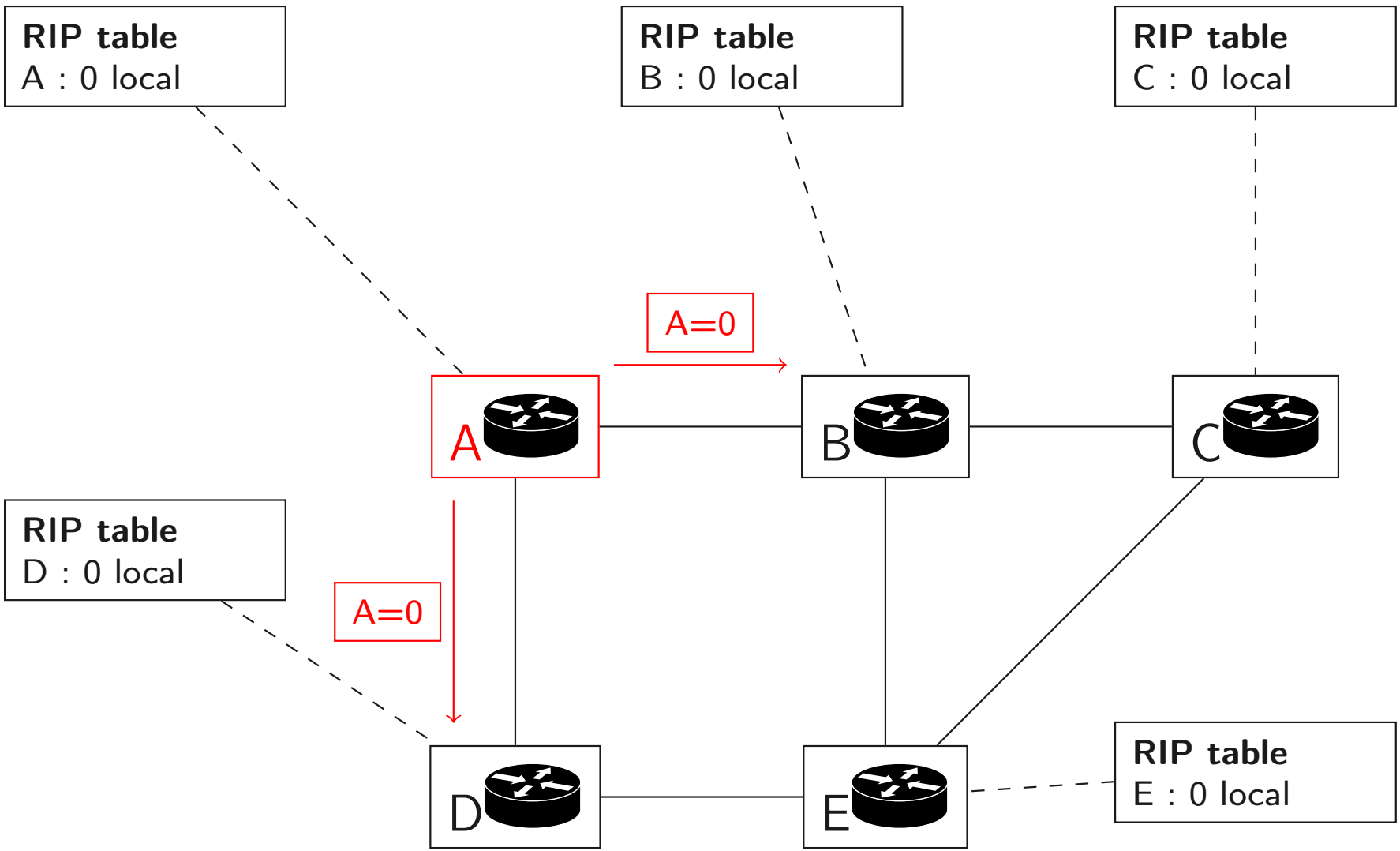
Émission périodique

```
loop indefinitely {
   sleep(N seconds)
   V=[]
   for each destination d in R[]
     {
       V[d]=R[d].cost
     }
   for each link l
     {
       l.send(V)
     }
}
```

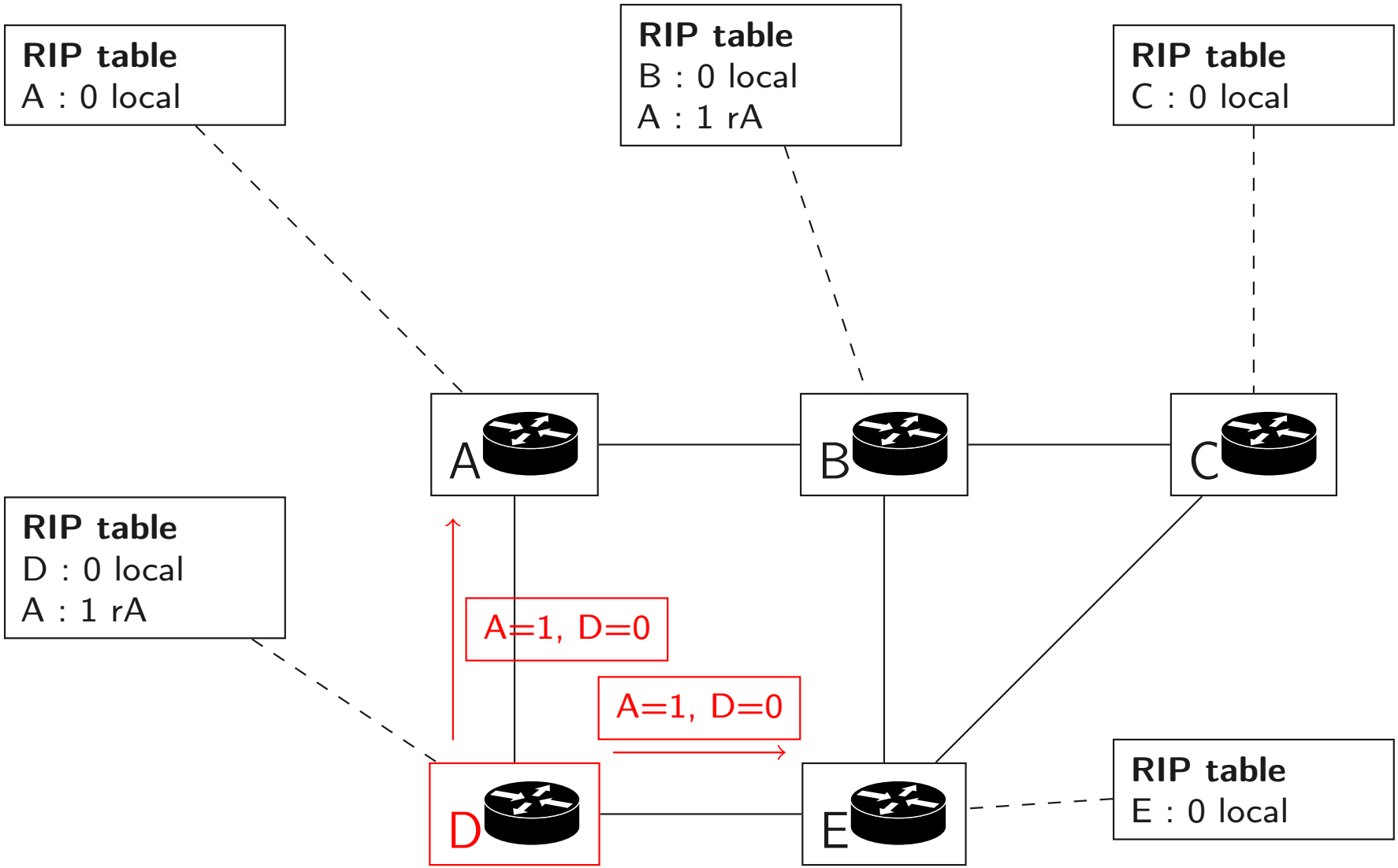
En réception

```
receive(Vector V[], router r)
{
   for each d in V[]
     {
        if (d in R) {
           if (V[d]+1.cost < R[d].cost) {
              R[d].cost=V[d]+1.cost
              R[d].gw=r
           }
        } else {
              R[d].cost=V[d]+1.cost
              R[d].gw=r
        }
     }
}
```

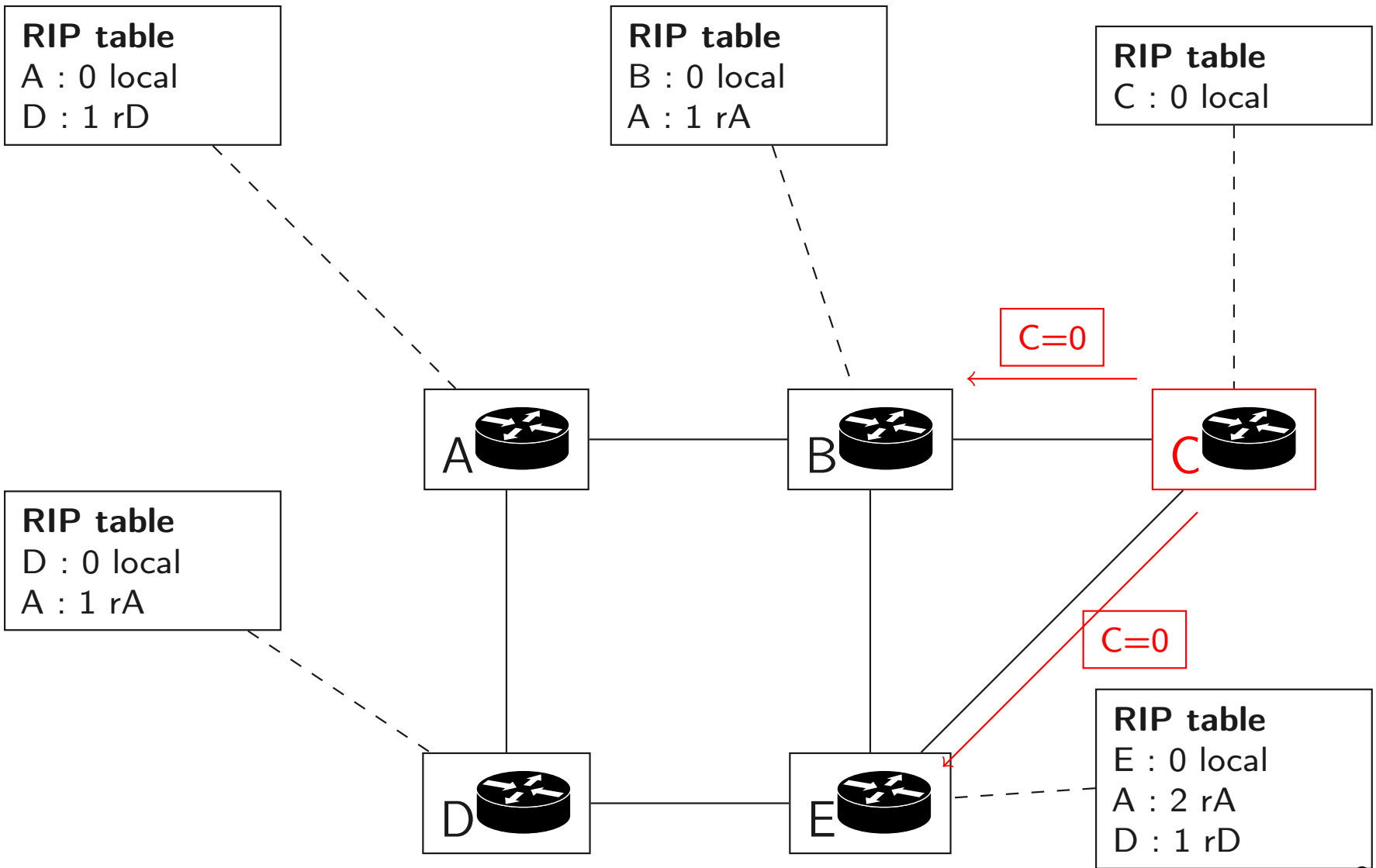
Exemple (les coûts sont tous 1)



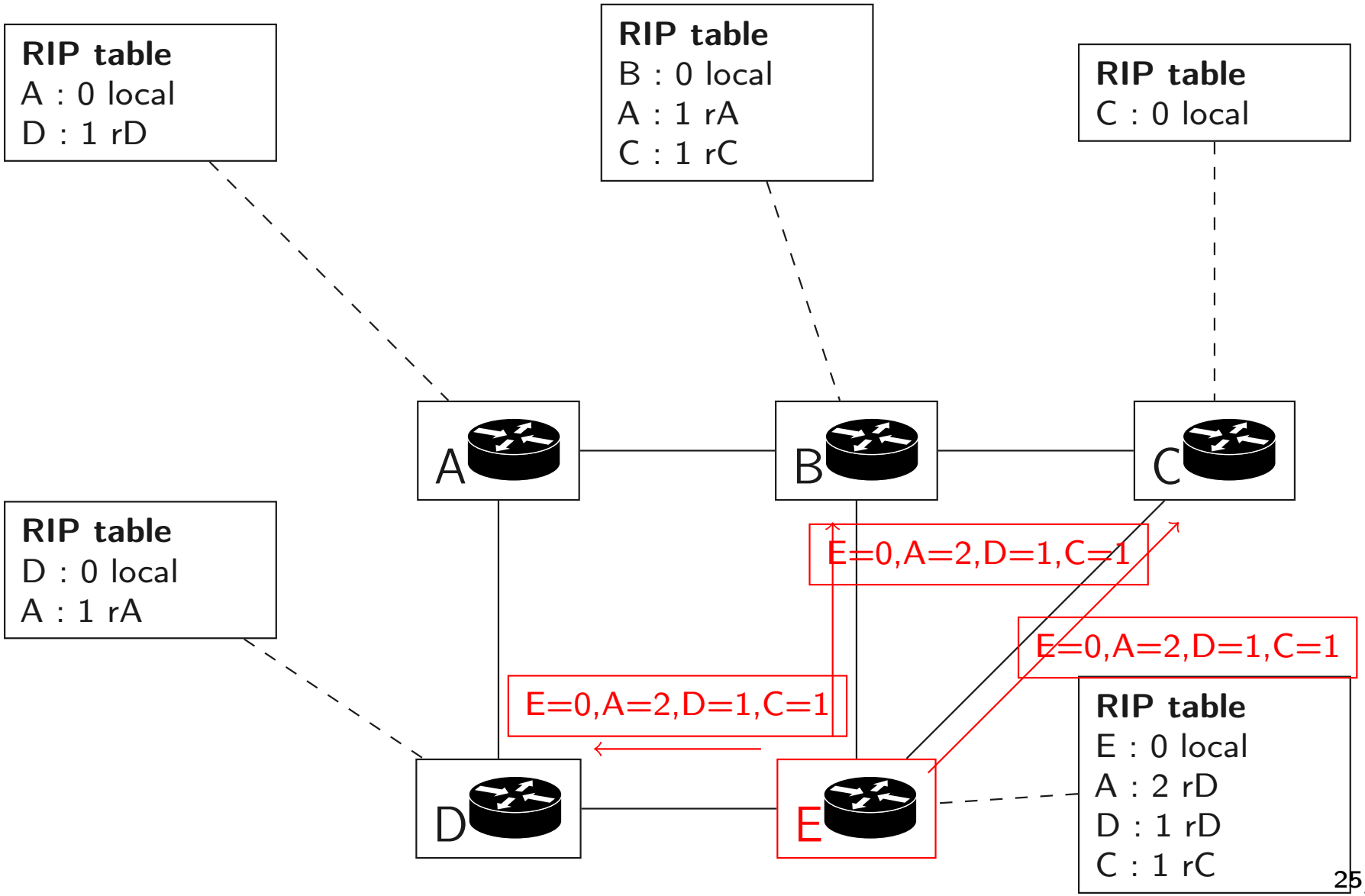
Exemple (les coûts sont tous 1)



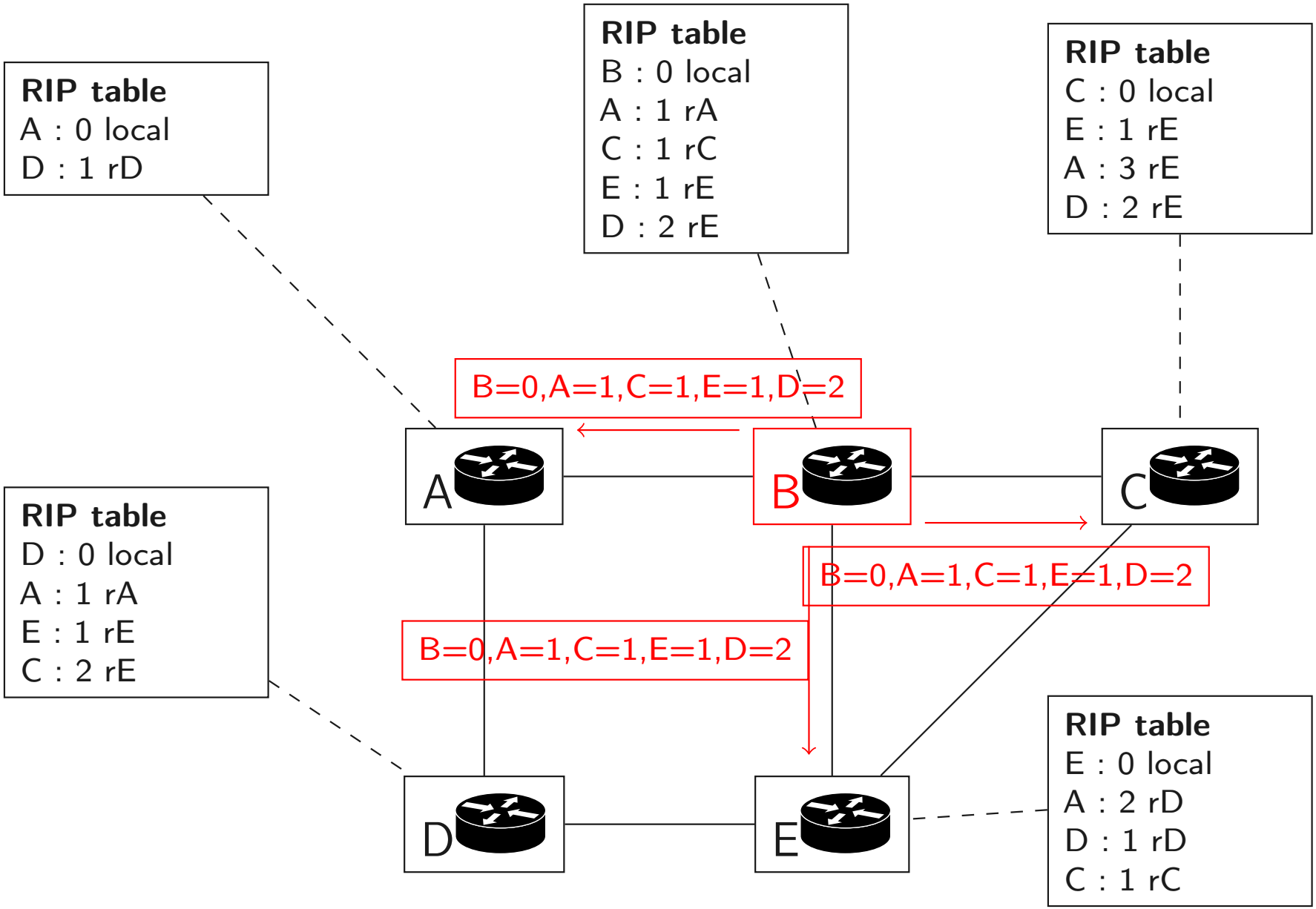
Exemple (les coûts sont tous 1)



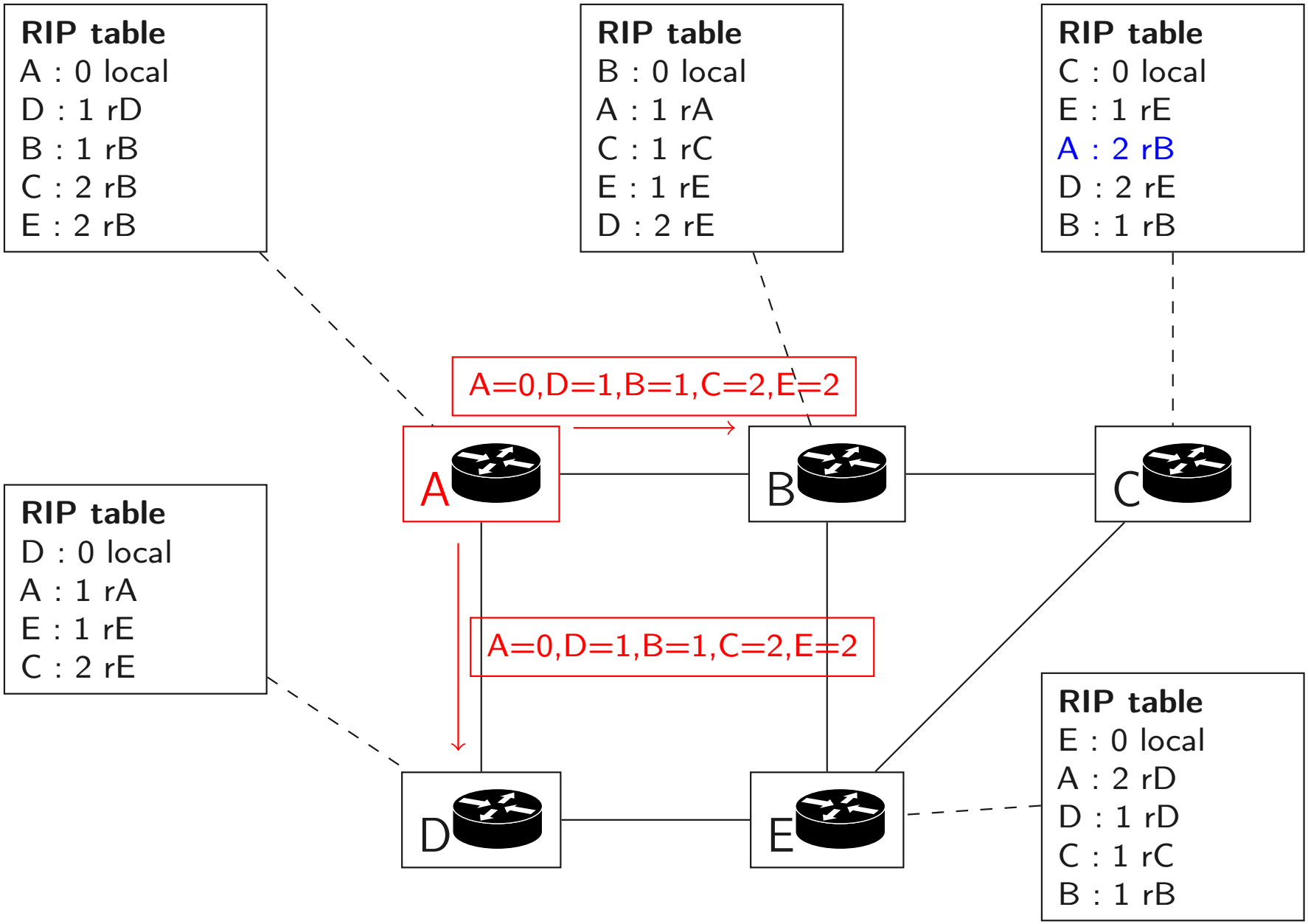
Exemple (les coûts sont tous 1)



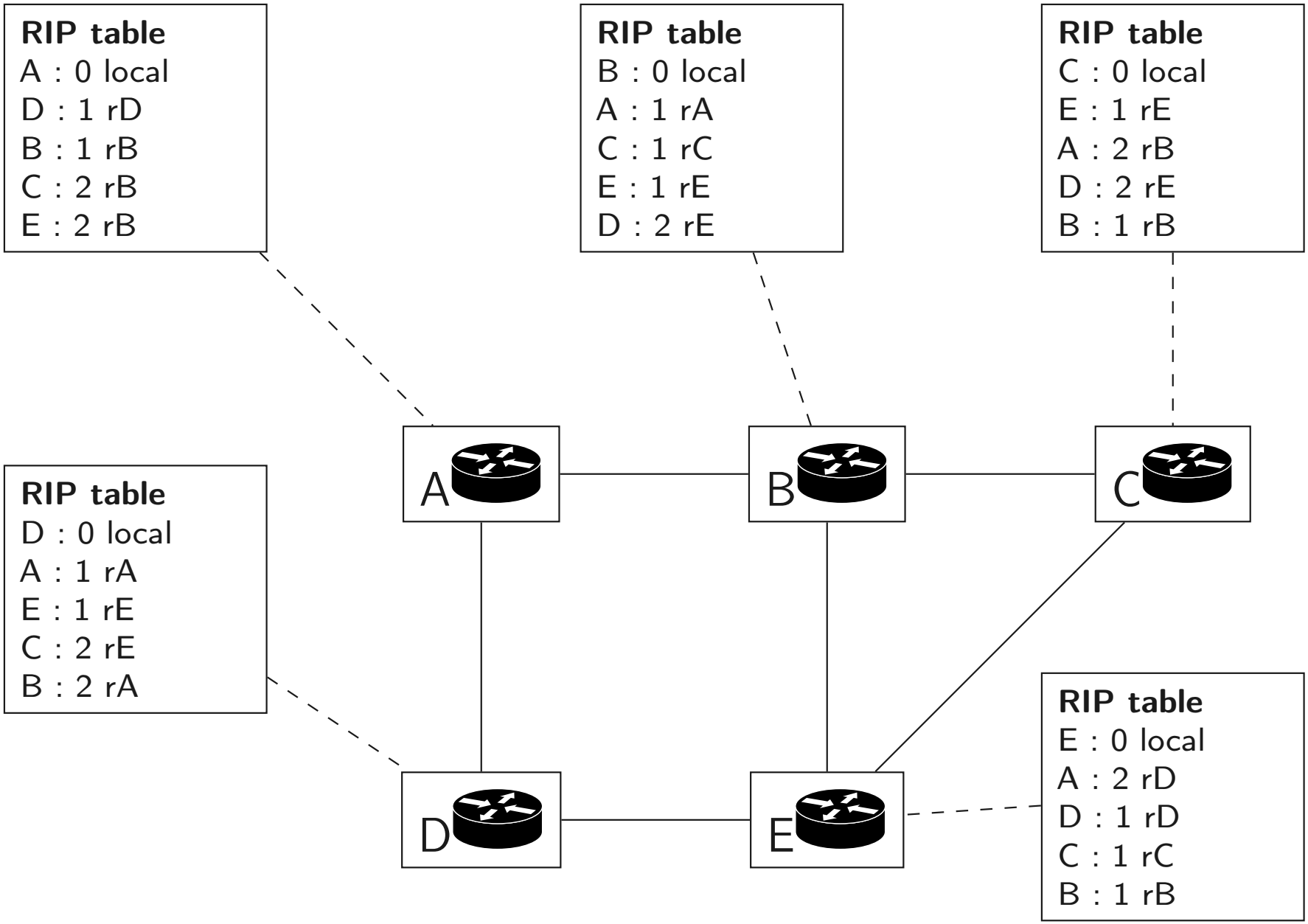
Exemple (les coûts sont tous 1)



Exemple (les coûts sont tous 1)



Exemple (fin)

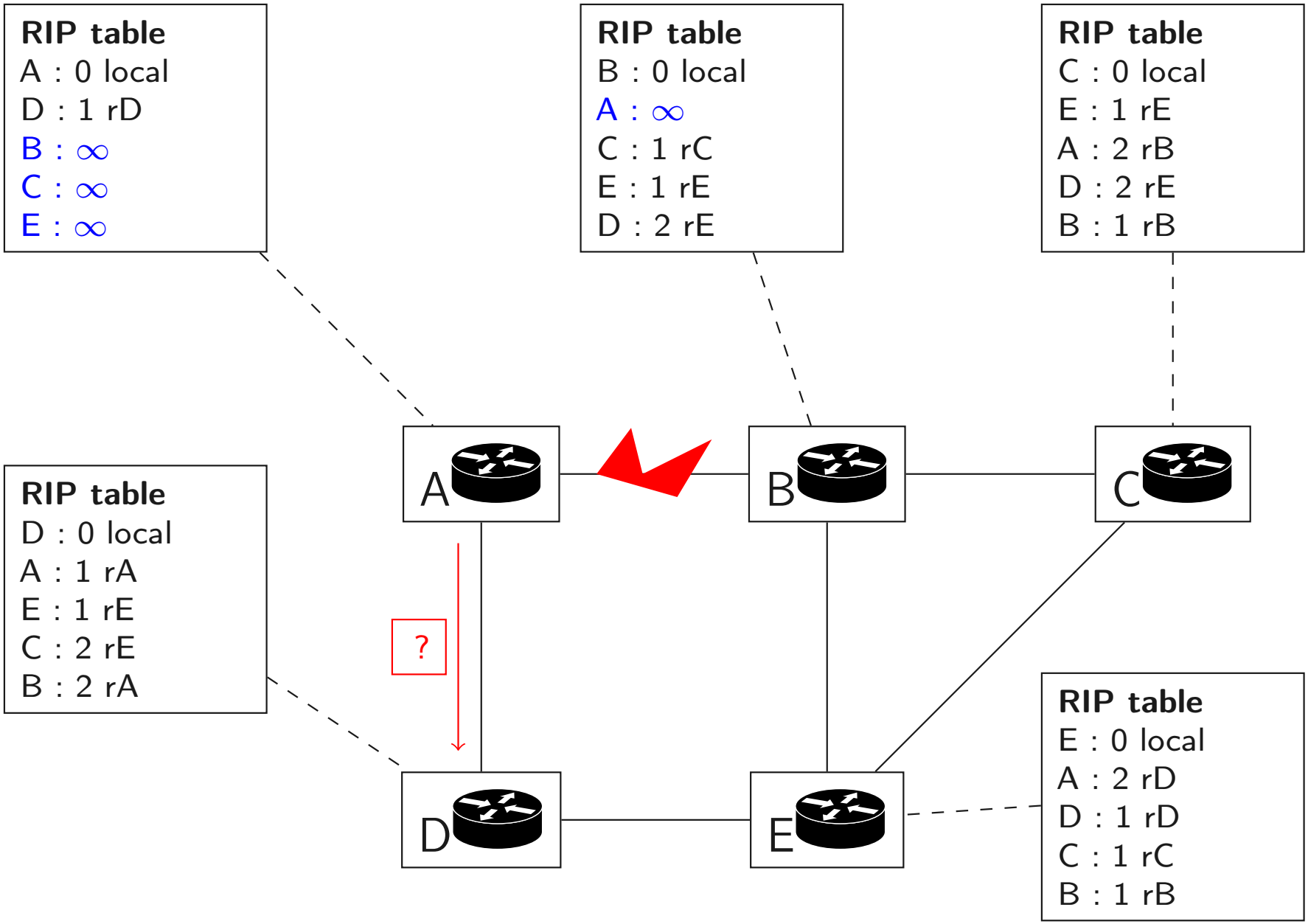

RIP table
A : 0 local
D : 1 rD
B : 1 rB
C : 2 rB
E : 2 rB
RIP table
B : 0 local
A : 1 rA
C : 1 rC
E : 1 rE
D : 2 rE
RIP table
C : 0 local
E : 1 rE
A : 2 rB
D : 2 rE
B : 1 rB
A
B
C
RIP table
D : 0 local
A : 1 rA
E : 1 rE
C : 2 rE
B : 2 rA
D
E
RIP table
E : 0 local
A : 2 rD
D : 1 rD
C : 1 rC
B : 1 rB

En cas de problème

Sans perte de lien, tous les routeurs finissent par atteindre une version optimale (donc stable) de la table de routage. Si un lien se perd :

- il faut pouvoir le détecter.
  - Faisable en couches basses.
  - Ou en considérant qu'un délai prolongé sans réception de vecteur distance signifie la perte du lien.
- et il faut calculer les nouvelles tables de routage.

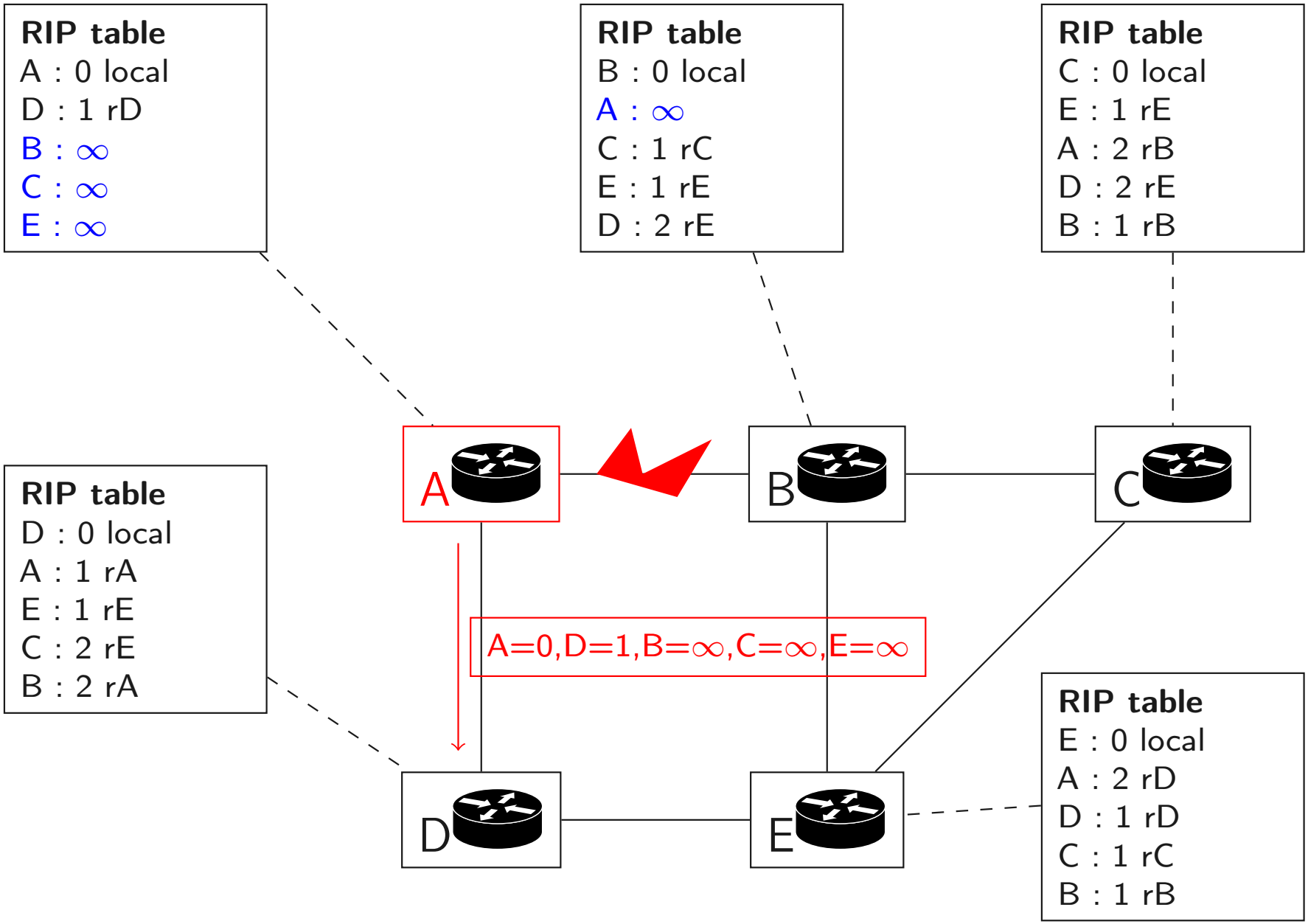
Exemple avec lien perdu



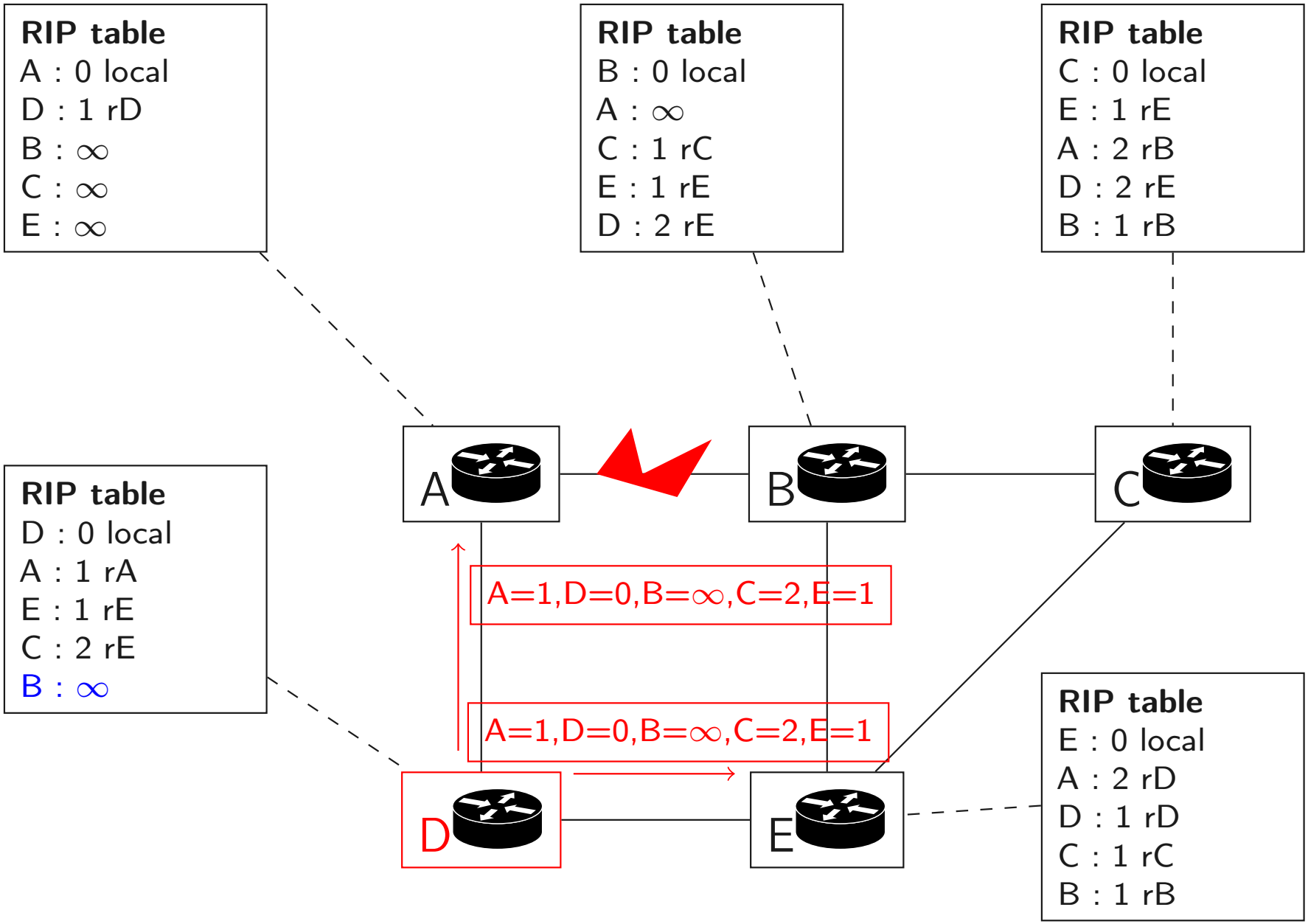
En réception (version 2)

```
receive(Vector V[], router r)
{
   for each d in V[]
     {
        if (d in R) {
           if ((V[d]+1.cost < R[d].cost)
                          or (R[d].gw=r)) {
              R[d].cost=V[d]+1.cost
              R[d].gw=r
           }
        } else {
              R[d].cost=V[d]+1.cost
              R[d].gw=r
        }
     }
}
```

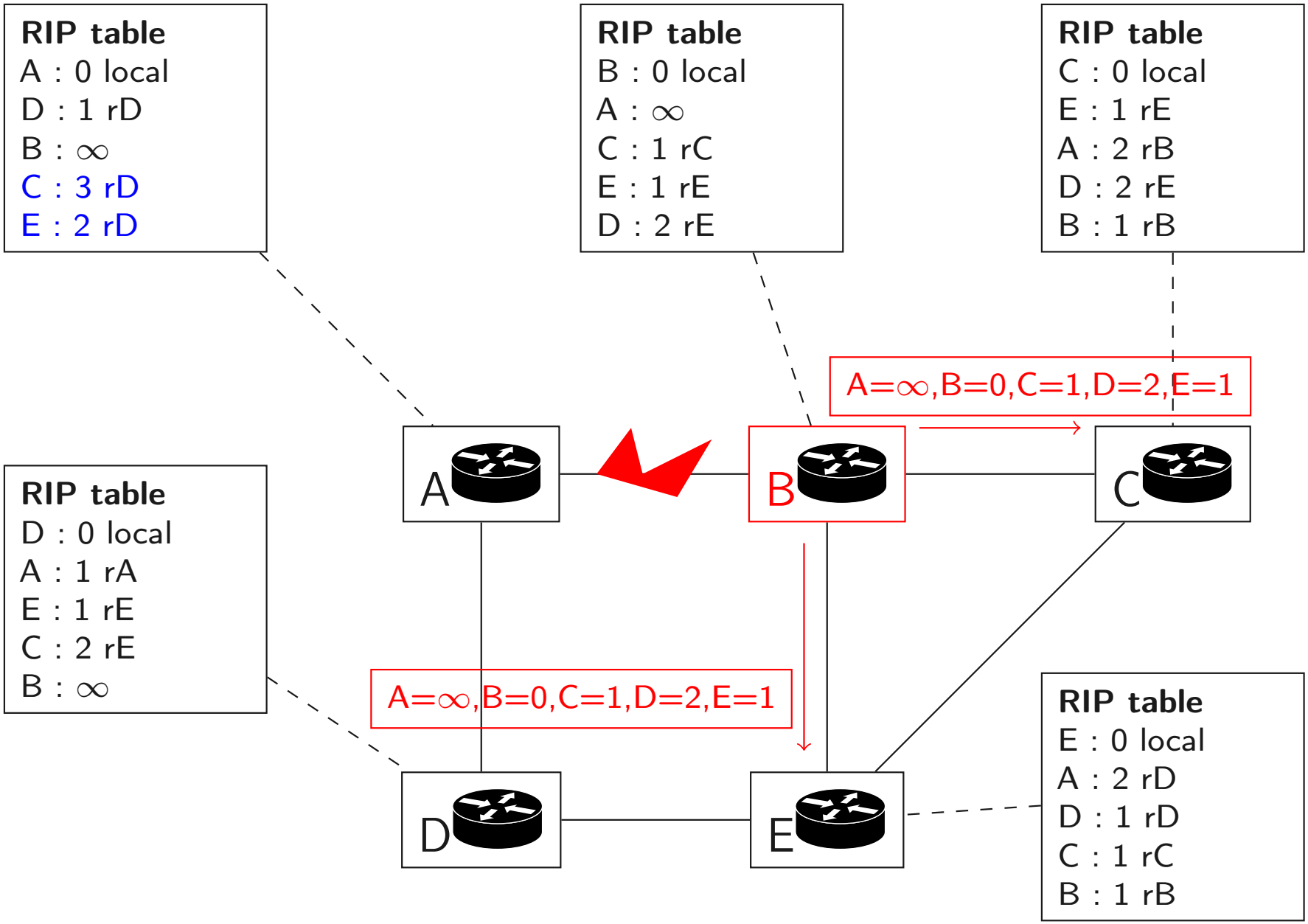
Exemple avec lien perdu



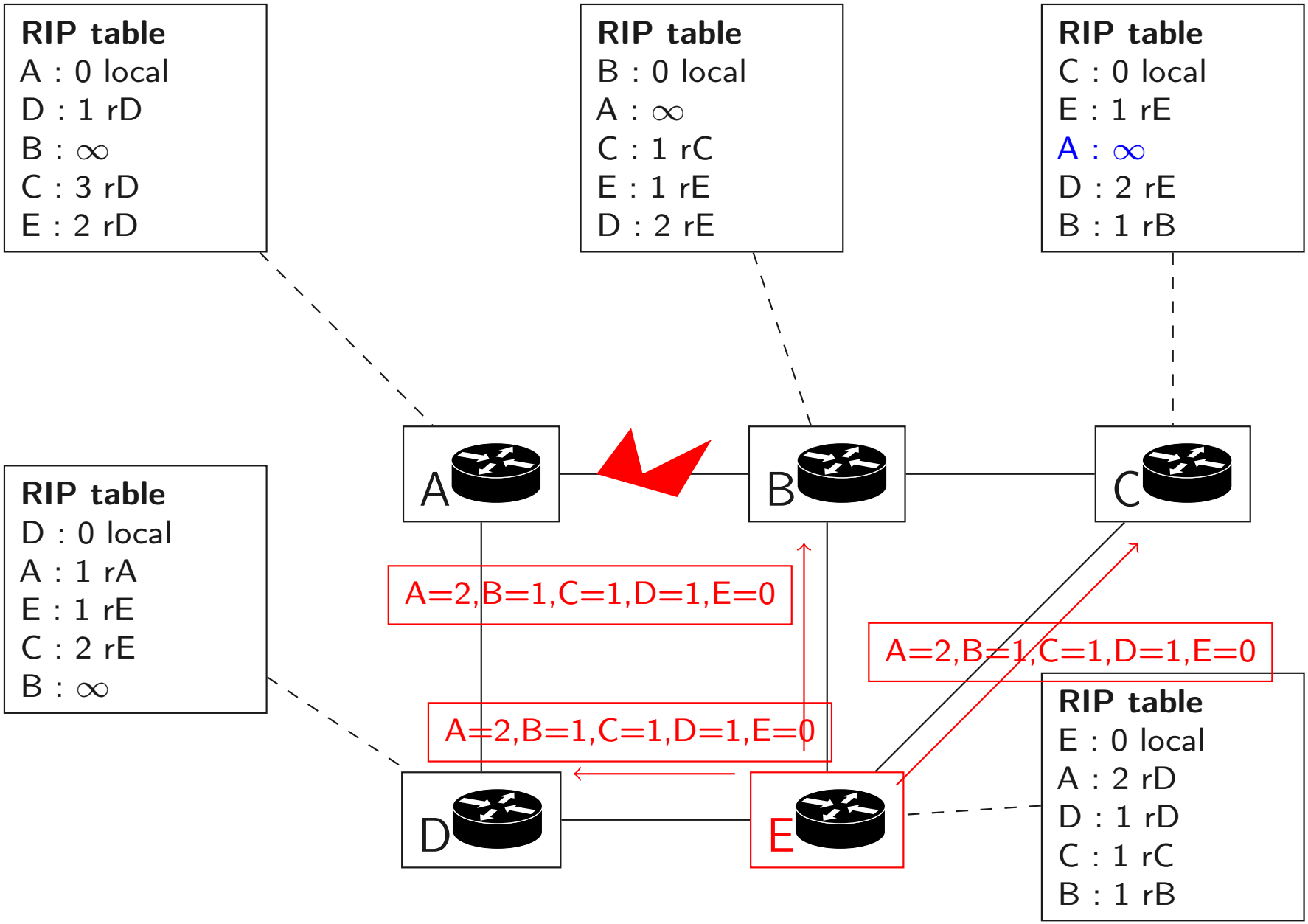
Exemple avec lien perdu



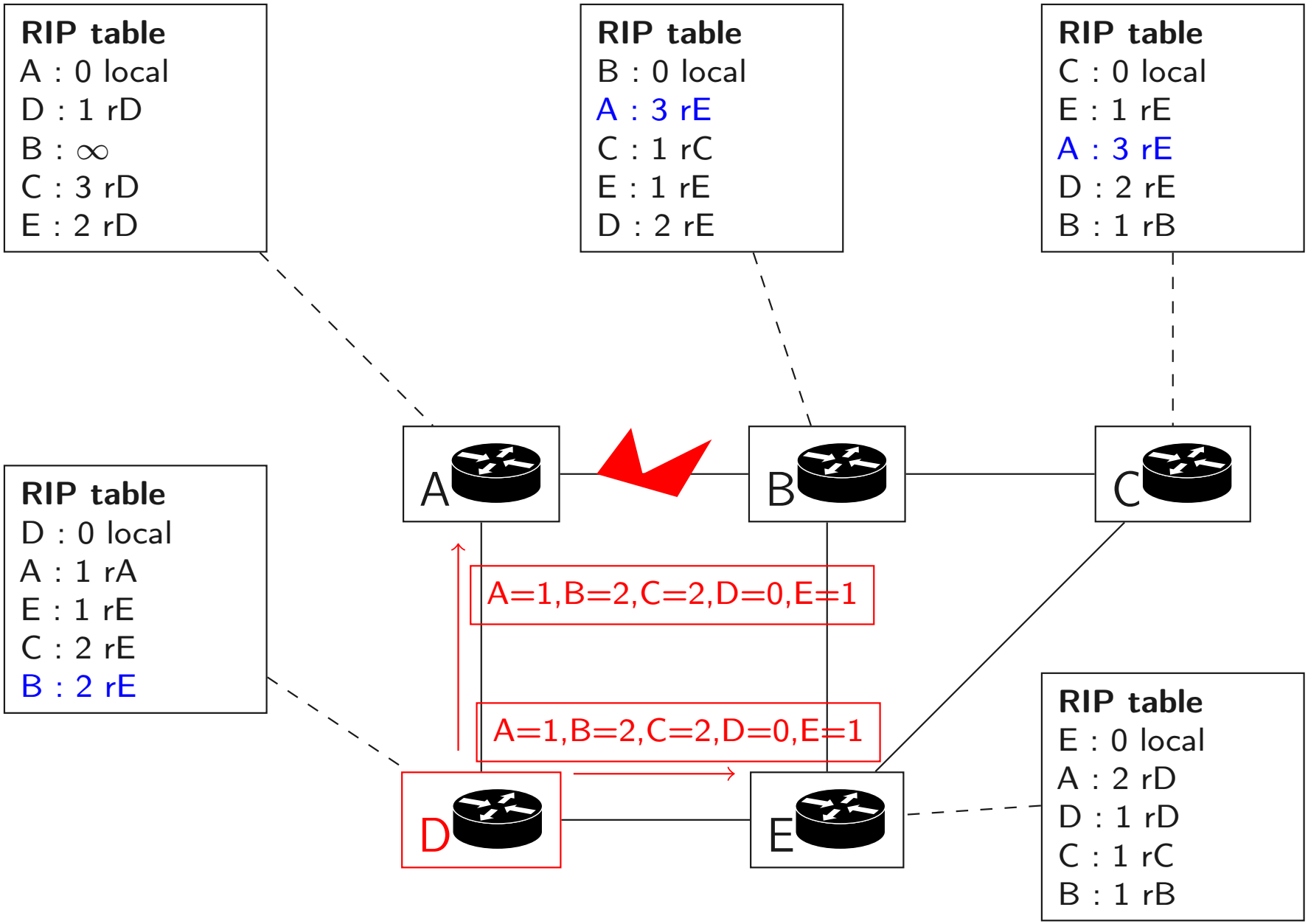
Exemple avec lien perdu



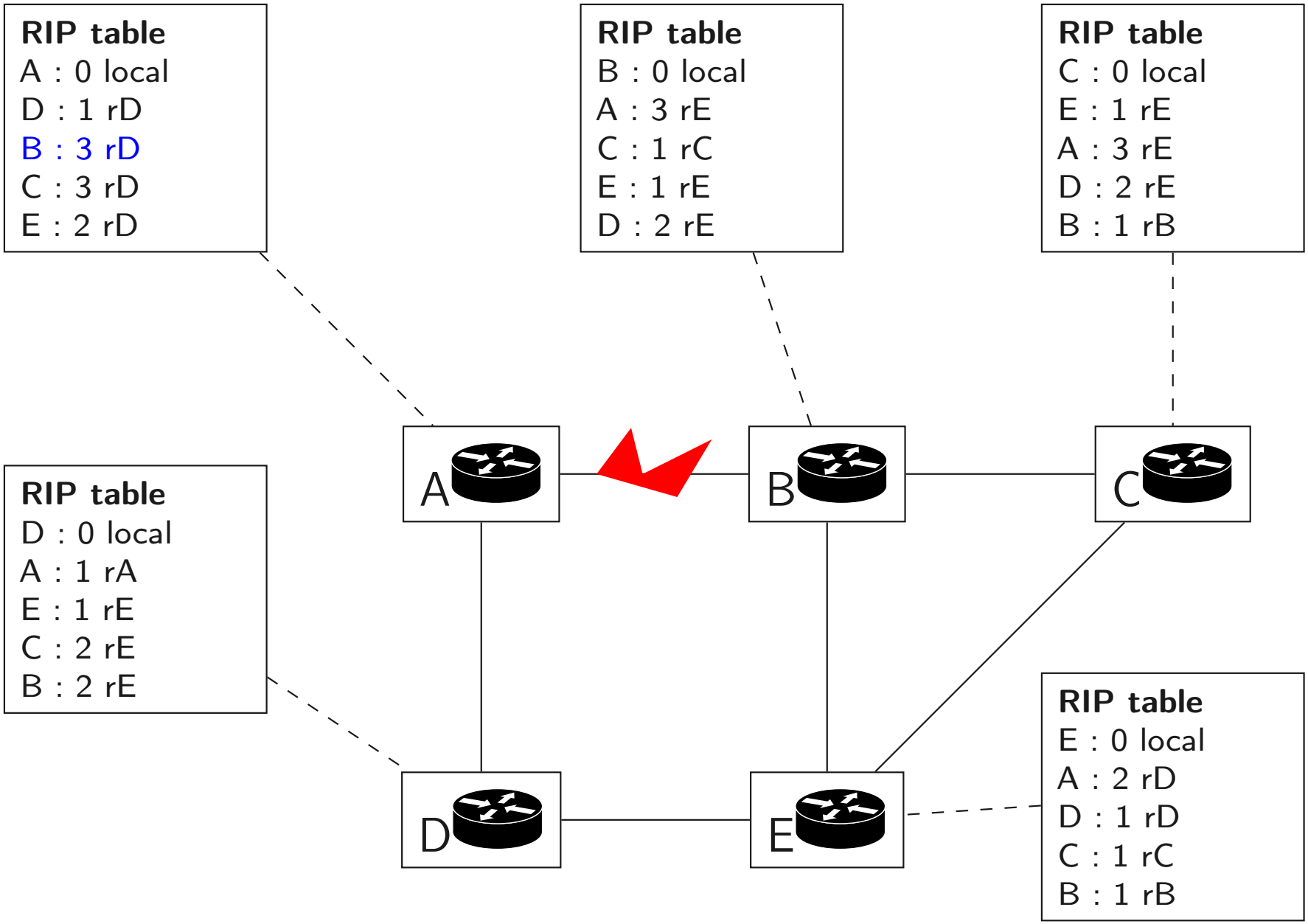
Exemple avec lien perdu



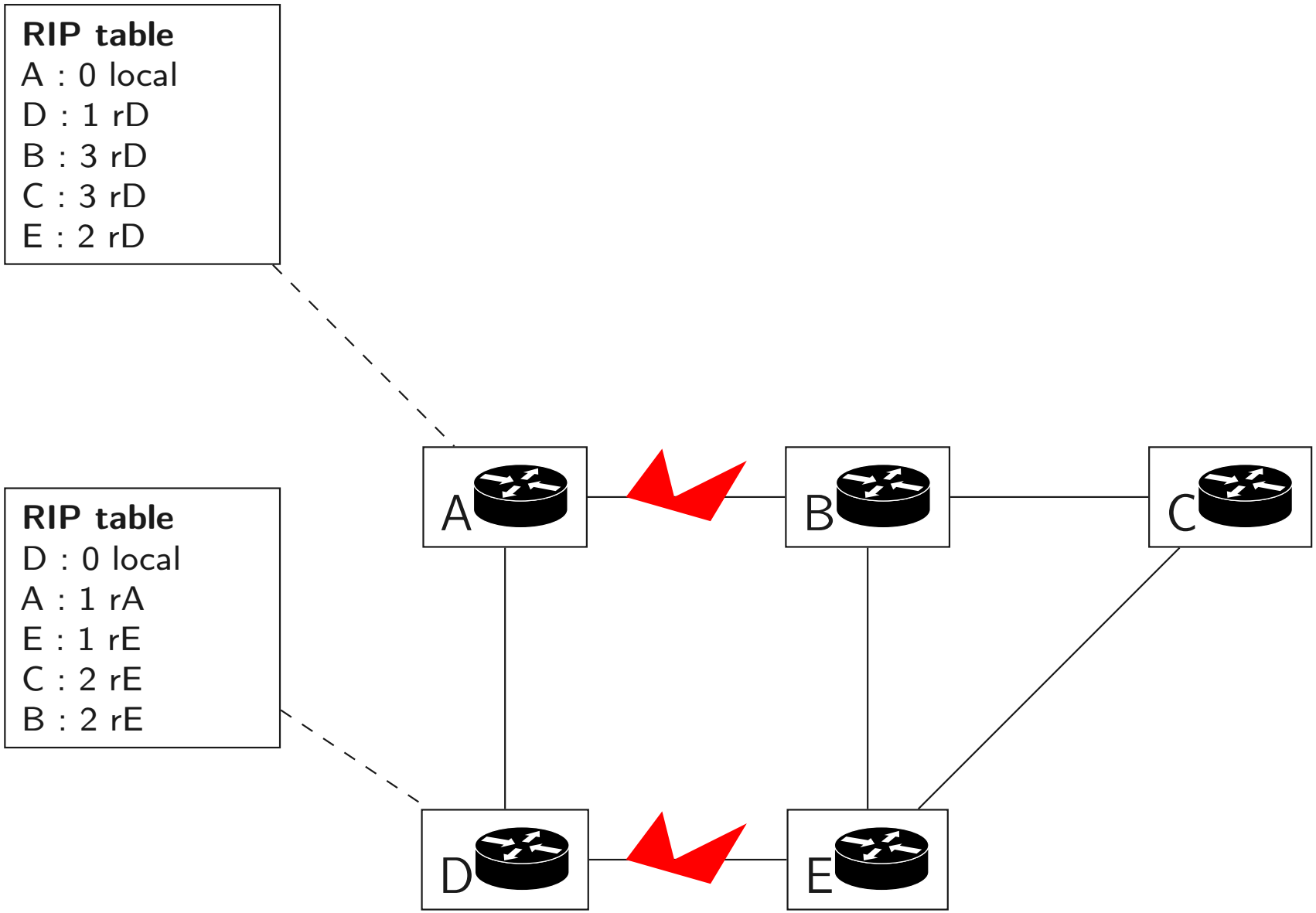
Exemple avec lien perdu



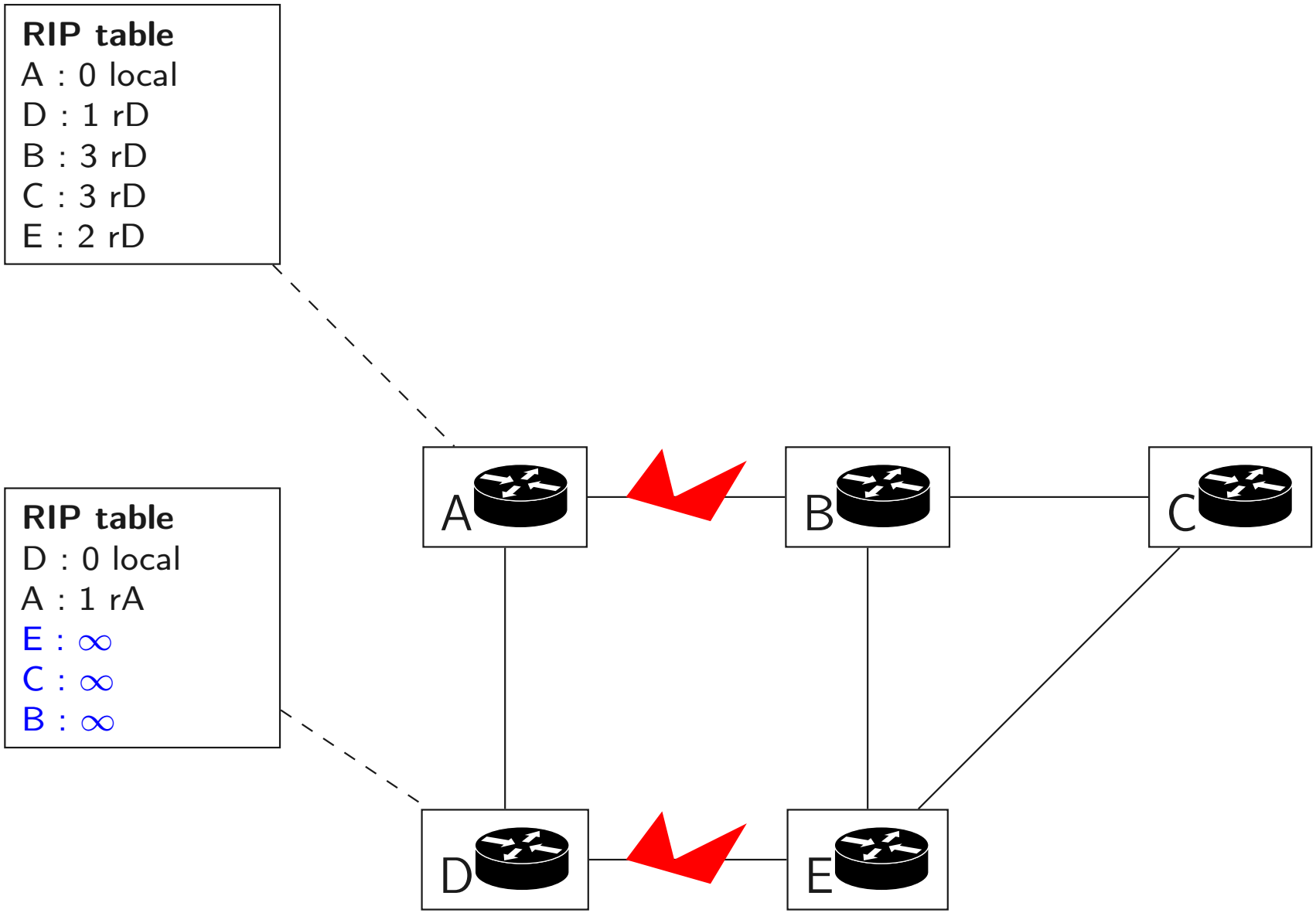
Exemple avec lien perdu



Perte de connexité



Perte de connexité

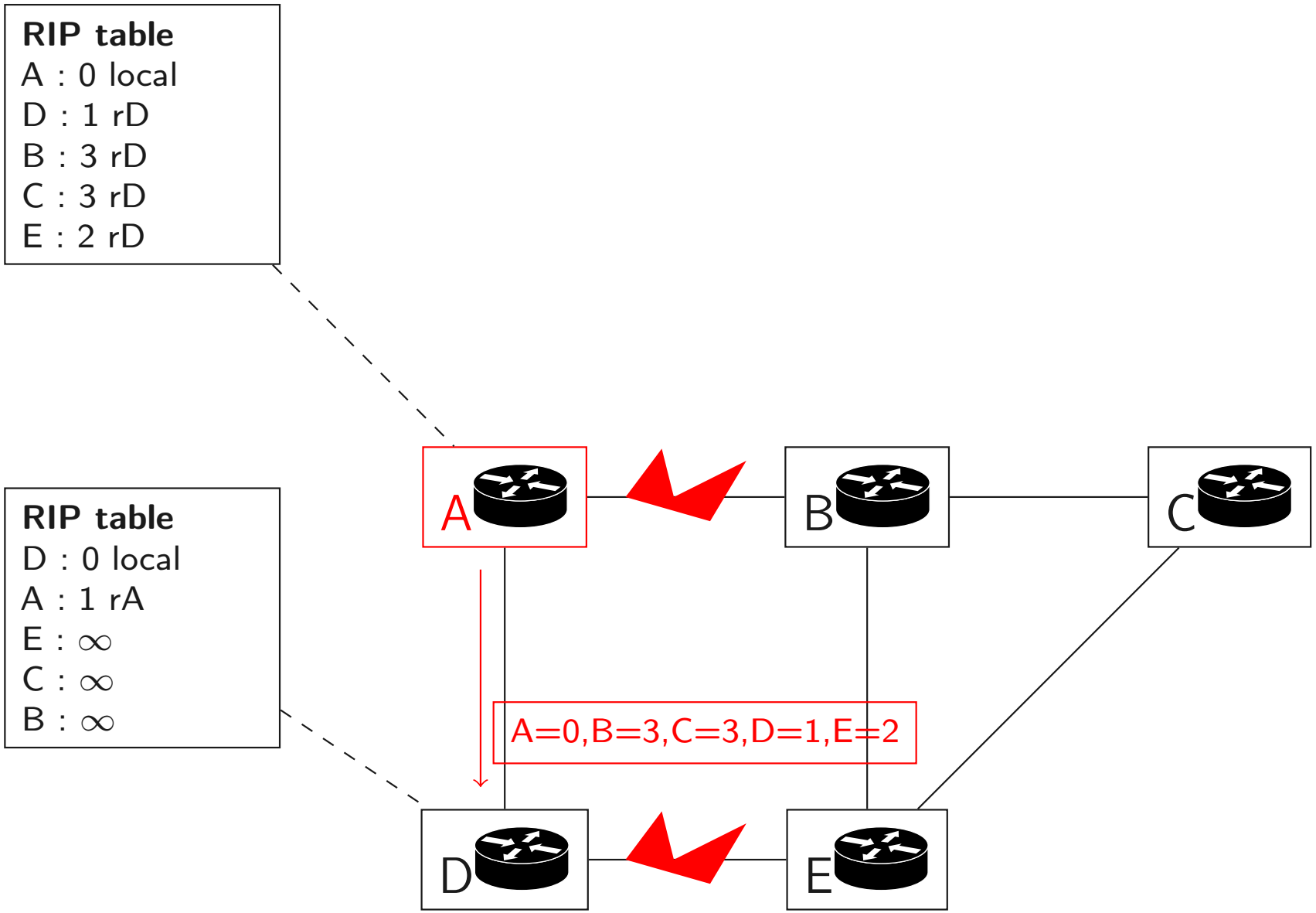


Perte de connexité

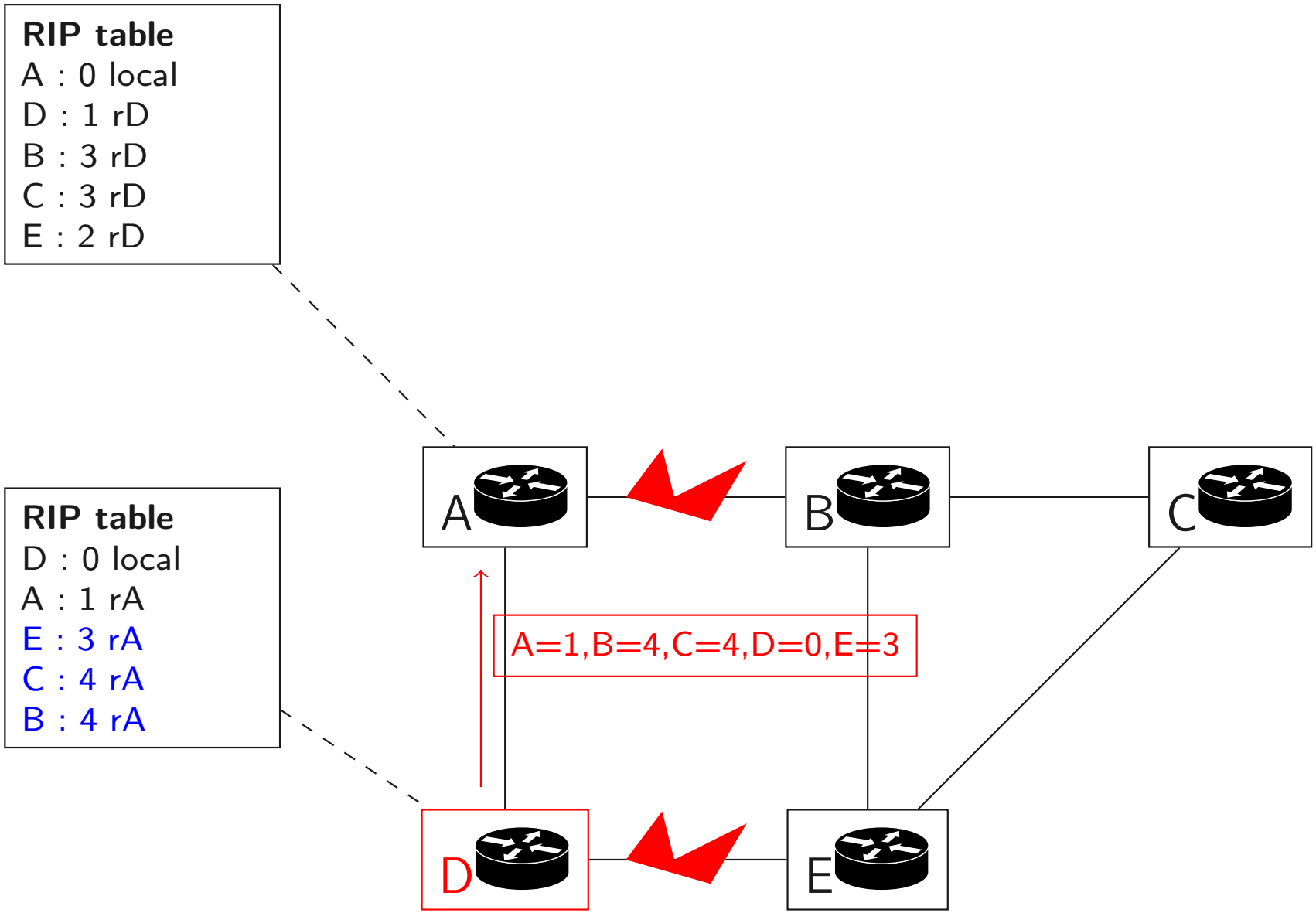
- Si D émet en premier, A met à jour sa table de routage et on est de nouveau dans un état stable.

- Si A émet en premier...

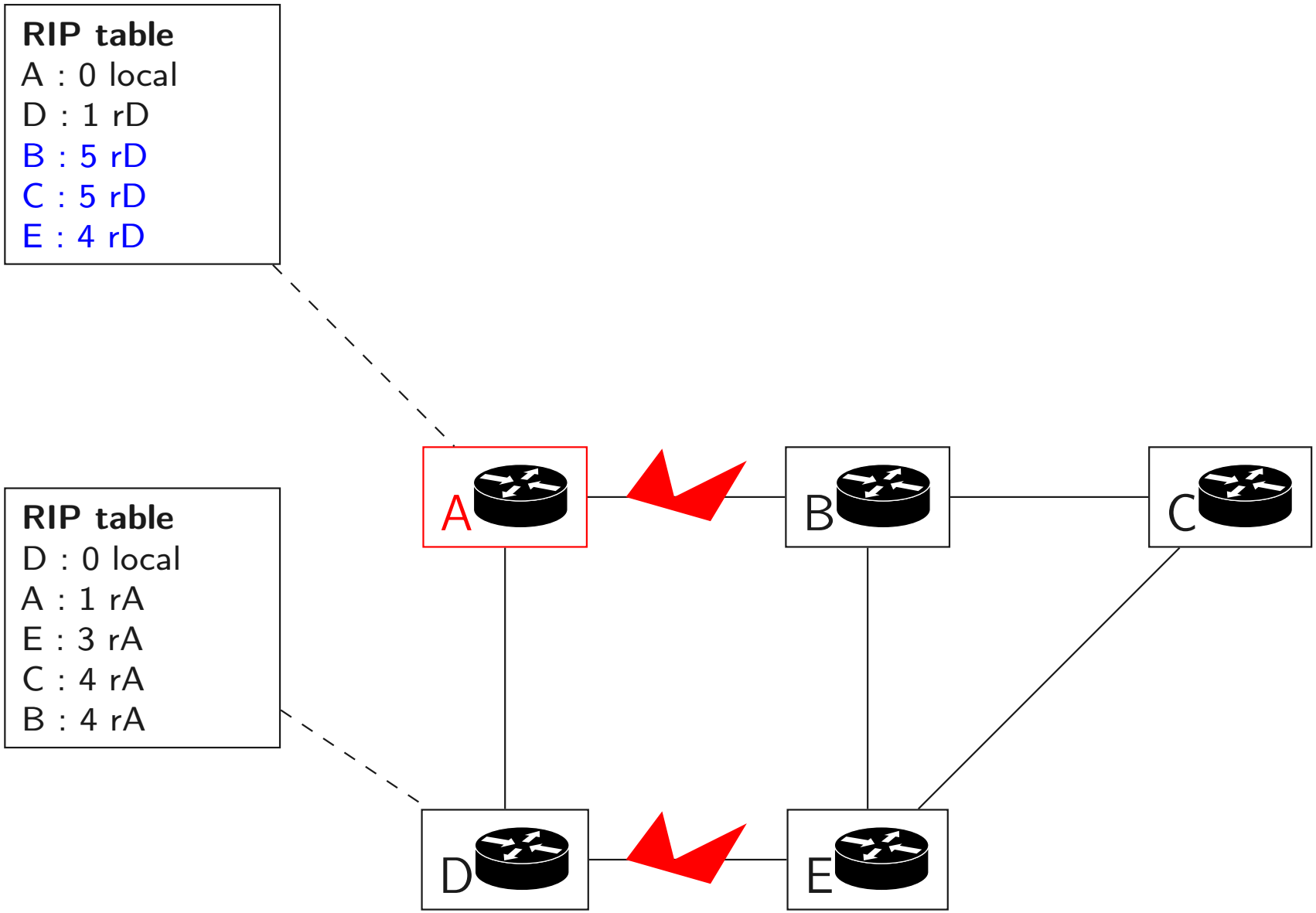
Perte de connexité



Perte de connexité



Perte de connexité



Perte de connexité

Compter à l'infini

Dans ce cas, les tables de A et D vont s'incrémenter alternativement sans fin.

- Le problème vient du fait que A et D s'annoncent des routes qu'ils ont appris l'un de l'autre.
- Tout dépend de l'ordre des mises à jour.
- Pour remédier à ce problème, il faut empêcher A de provoquer la mise à jour de la table de D.

Convergence lente

Des solutions

Dans certains cas (e.g. perte de connexité) la convergence vers une nouvelle situation stable peut prendre beaucoup de temps :

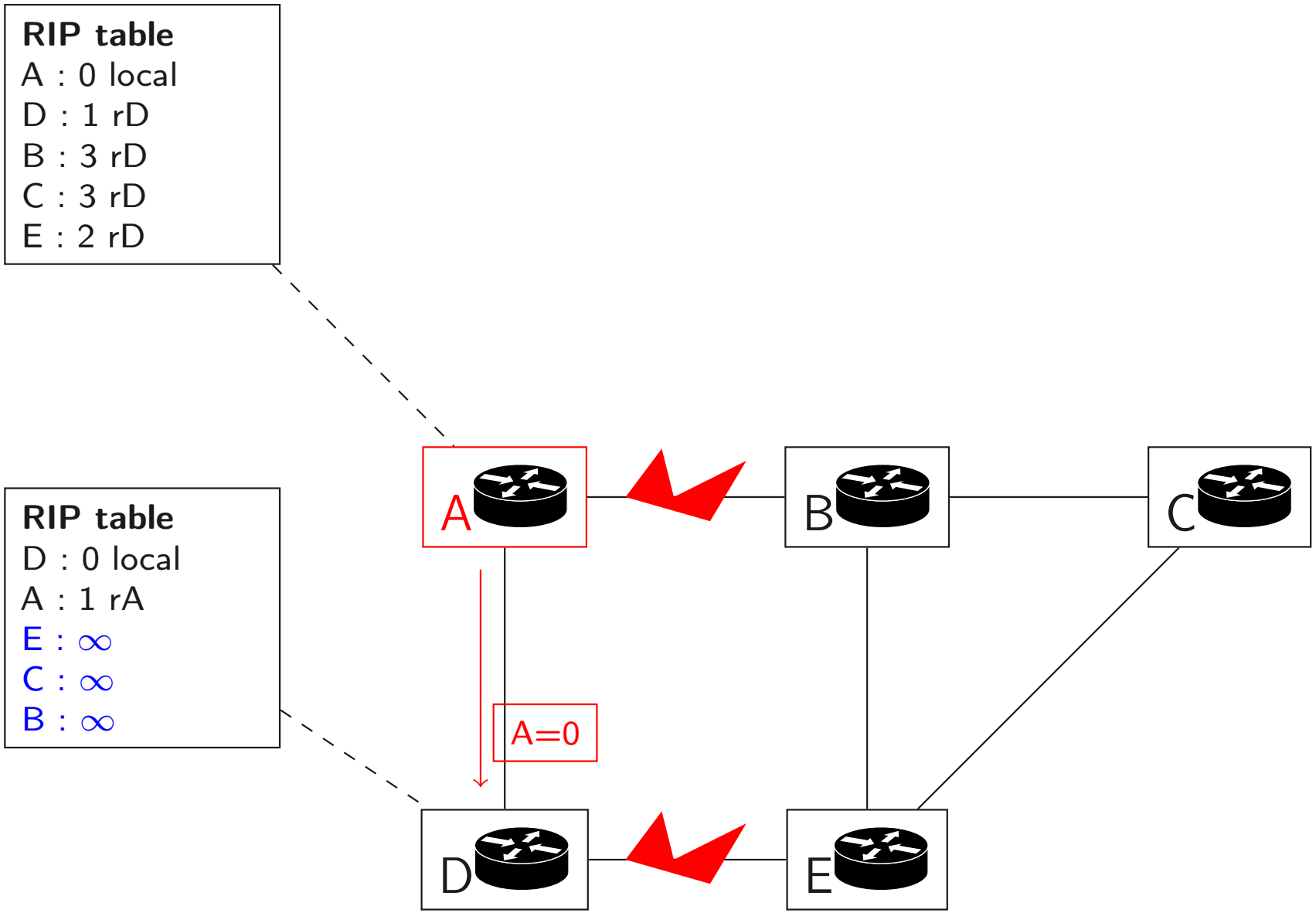
- `Hold down` on peut forcer un routeur qui détecte la perte d'un lien à ignorer les mises à jour pendant un certain temps (plus d'une période d'émission de vecteurs distance).
- `Triggered update` on peut forcer un routeur qui détecte la perte d'un lien à envoyer immédiatement une mise à jour à ses voisins.
- `Split horizon` on peut forcer chaque routeur à ne pas communiquer à ses voisins les chemins qui passent par ceux-ci.

Émission périodique (version 2)

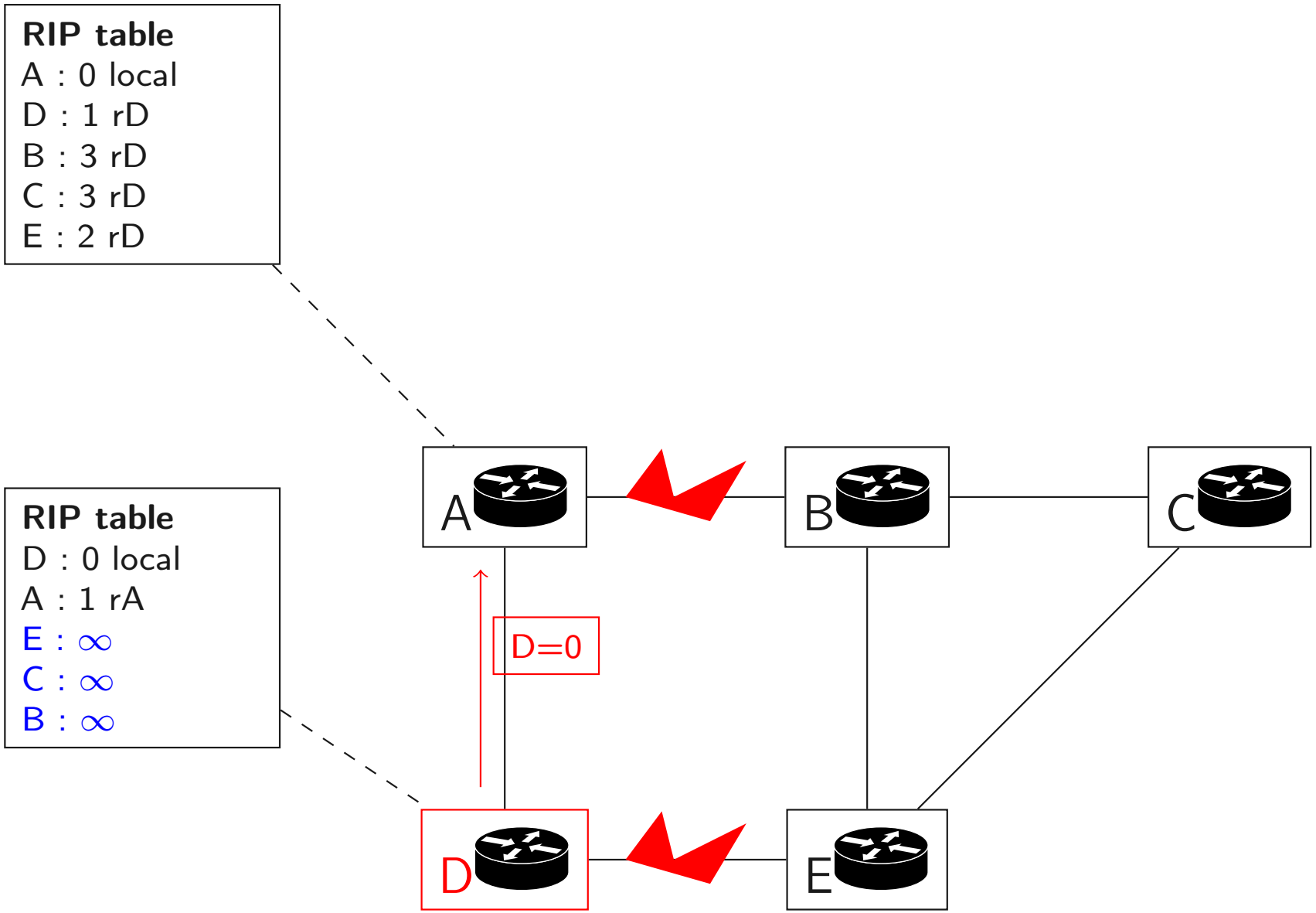
Avec horizon éclaté

```
loop indefinitely {
   sleep(N seconds)
   for each link l
     {
       V=[]
       for each destination d in R[]
       {
         if (R[d].gw not on l) {
           V[d]=R[d].cost
         }
       }
       l.send(V)
     }
}
```

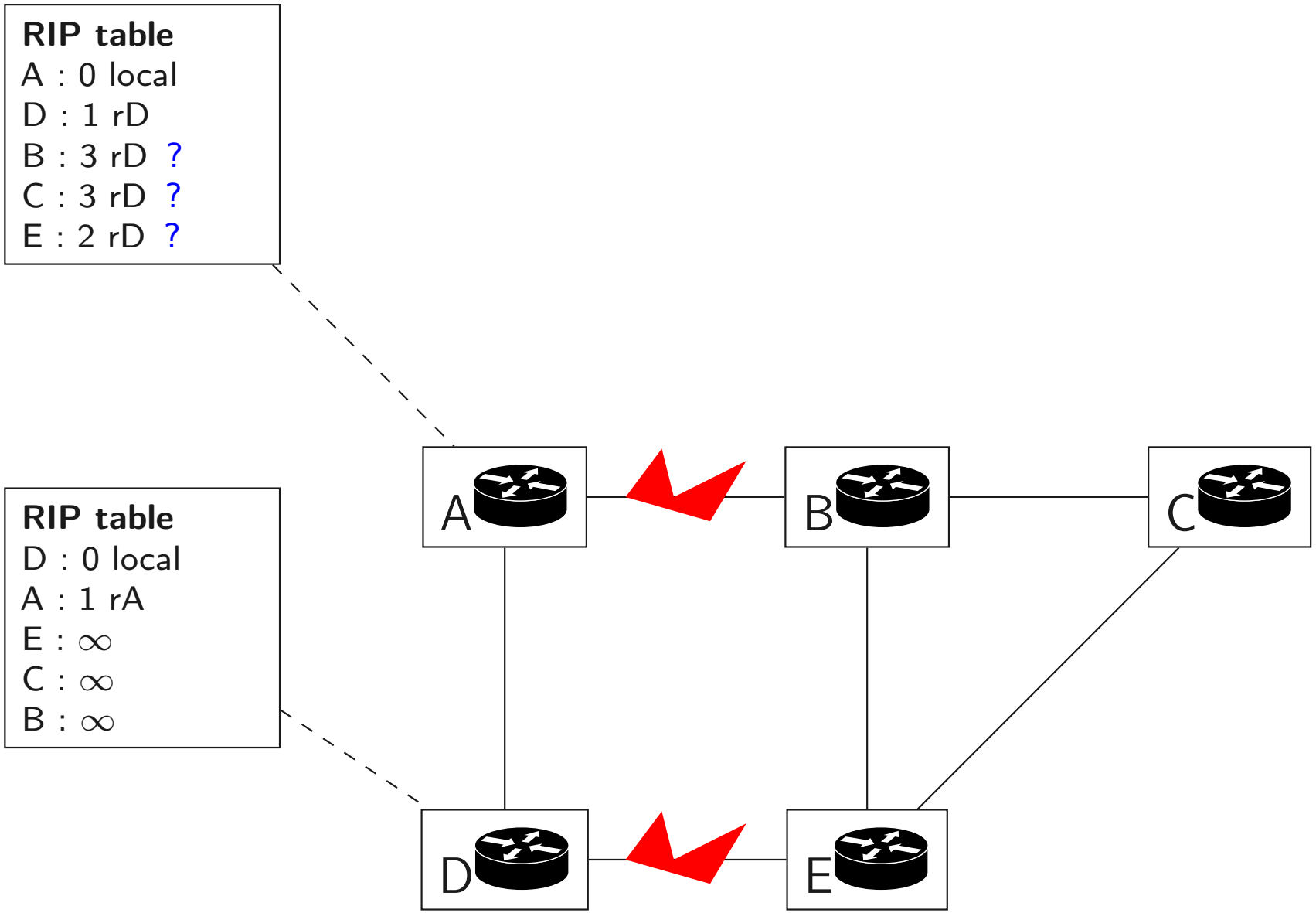
Avec horizon éclaté



Avec horizon éclaté



Avec horizon éclaté



Horizon éclaté avec empoisonnement

Pour accélérer le processus, au lieu de ne pas communiquer à ses voisins les chemins perdus, on leur annonce un coût ∞.

Par exemple si 2 routeurs pointent l'un vers l'autre, l'empoisonnement provoque la mise à jour immédiate.

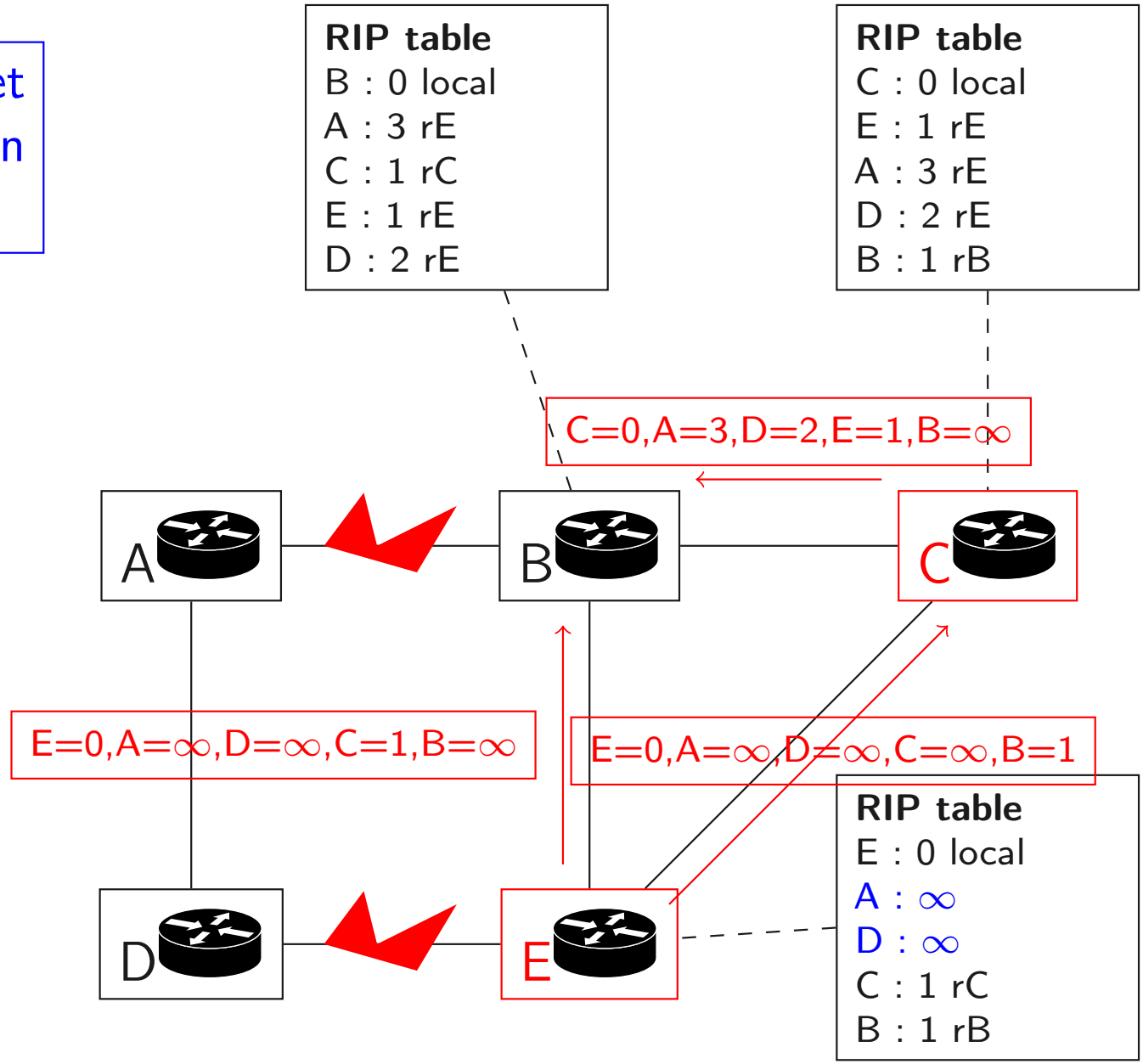
Sans empoisonnement, il faut attendre un délai au bout duquel le réseau est déclaré non accessible.

Émission périodique (version 3)

```
loop indefinitely {
   sleep(N seconds)
   for each link l
     {
       V=[]
       for each destination d in R[]
       {
         if (R[d].gw not on l) {
           V[d]=R[d].cost
         } else {
           V[d]=infinity
         }
       }
       l.send(V)
     }
}
```

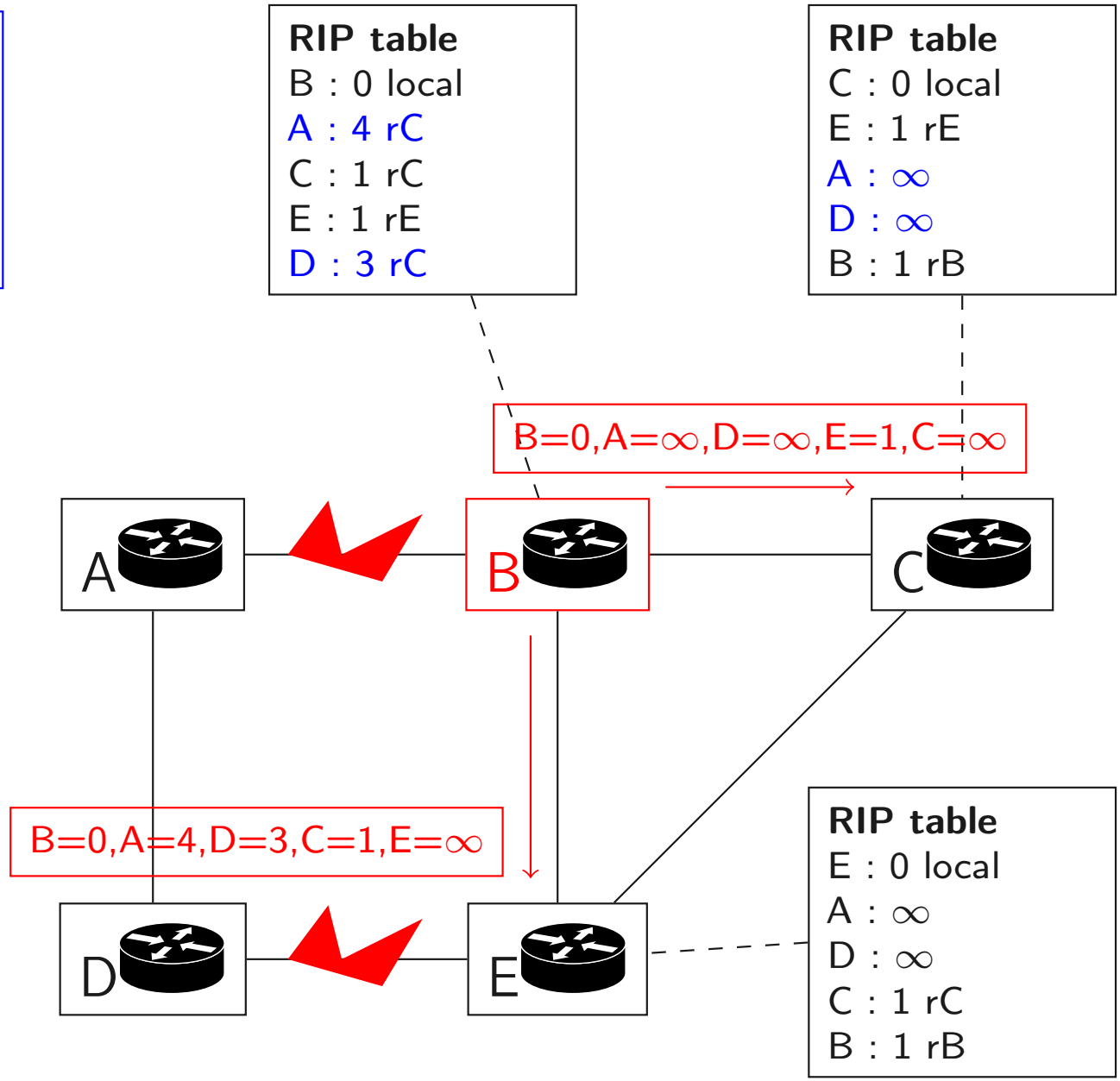
Boucle de longueur > 2

Pb : Si R_E et R_C émettent en même temps



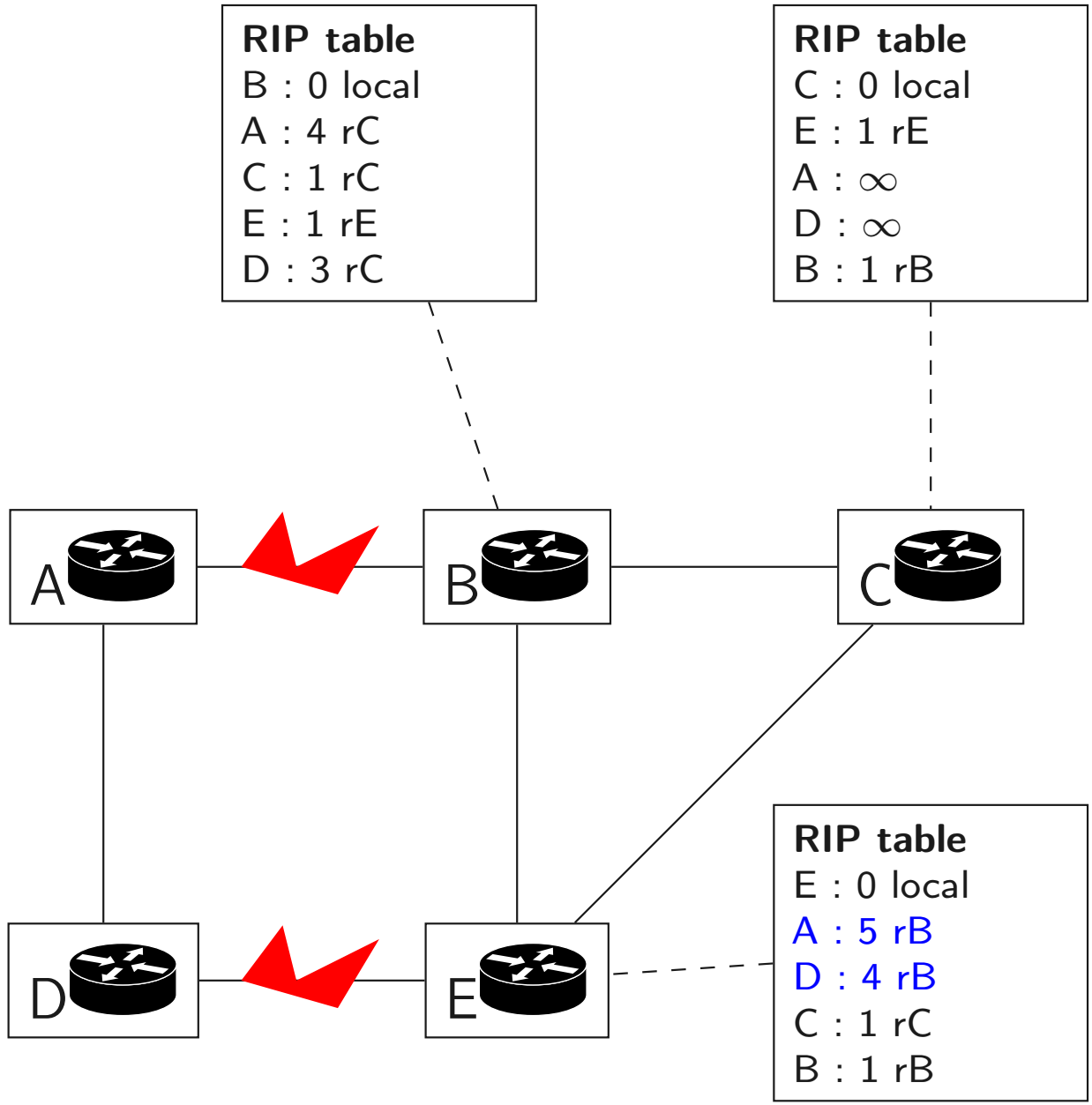
Boucle de longueur > 2

R_B reçoit et met à jour en fonction de R_E puis de R_C



Boucle de longueur > 2

Nouveau compte à l'infini



Solutions et problèmes

- Toutes les solutions possibles (Hold down, Triggered updates, Split Horizon,...) sont partielles.
- En ajoutant des messages (e.g. Triggered update), il y a risque de saturer le réseau (les messages sont en broadcast).
- En ajoutant des données (Split horizon with poisoning : les routeurs annoncent ∞ au lieu de ne rien annoncer sur le lien emprunté par un chemin), le volume de données qui circule augmente.
- Convergence lente.
- Messages volumineux.
- Passage à l'échelle compliqué.

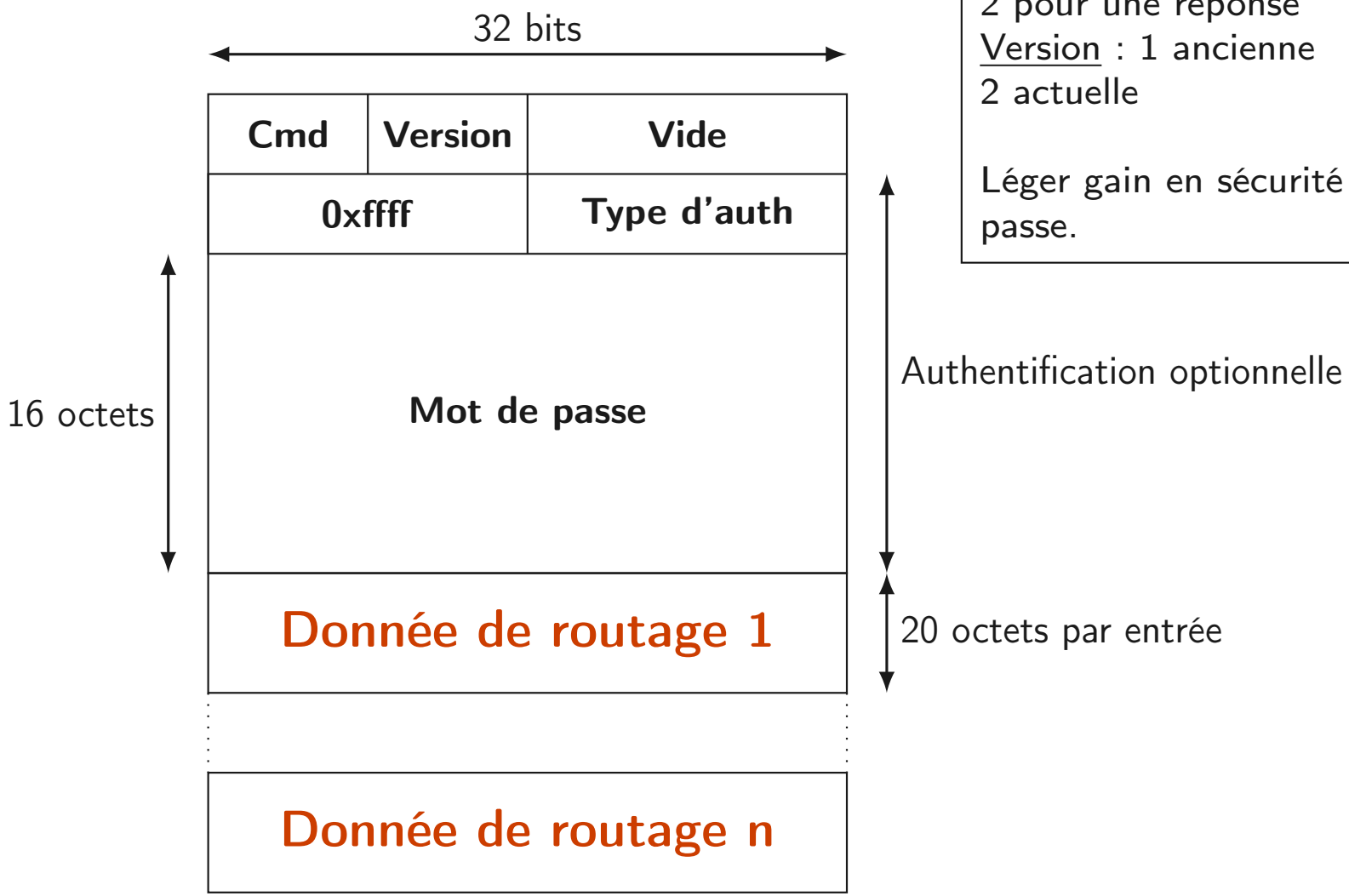
RIP

Routing Information Protocol

Protocole à base de vecteurs de distance (RFC 2453) :

- $\infty = 16$
- tous les coûts sont de 1 (métrique = nombre de sauts)
- émission de message toutes les 30s (Hold down de 60s)
- datagrammes UDP, port 520, envoyés en multicast à tous les routeurs RIP

Datagramme RIP



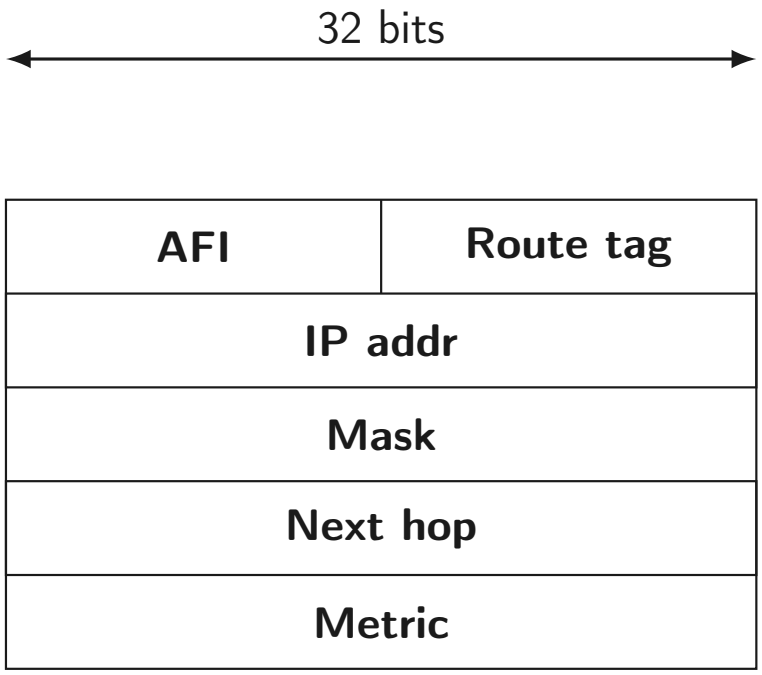
Commande : 1 pour une requête
2 pour une réponse
Version : 1 ancienne
2 actuelle

Léger gain en sécurité si mot de passe.

Datagramme RIP

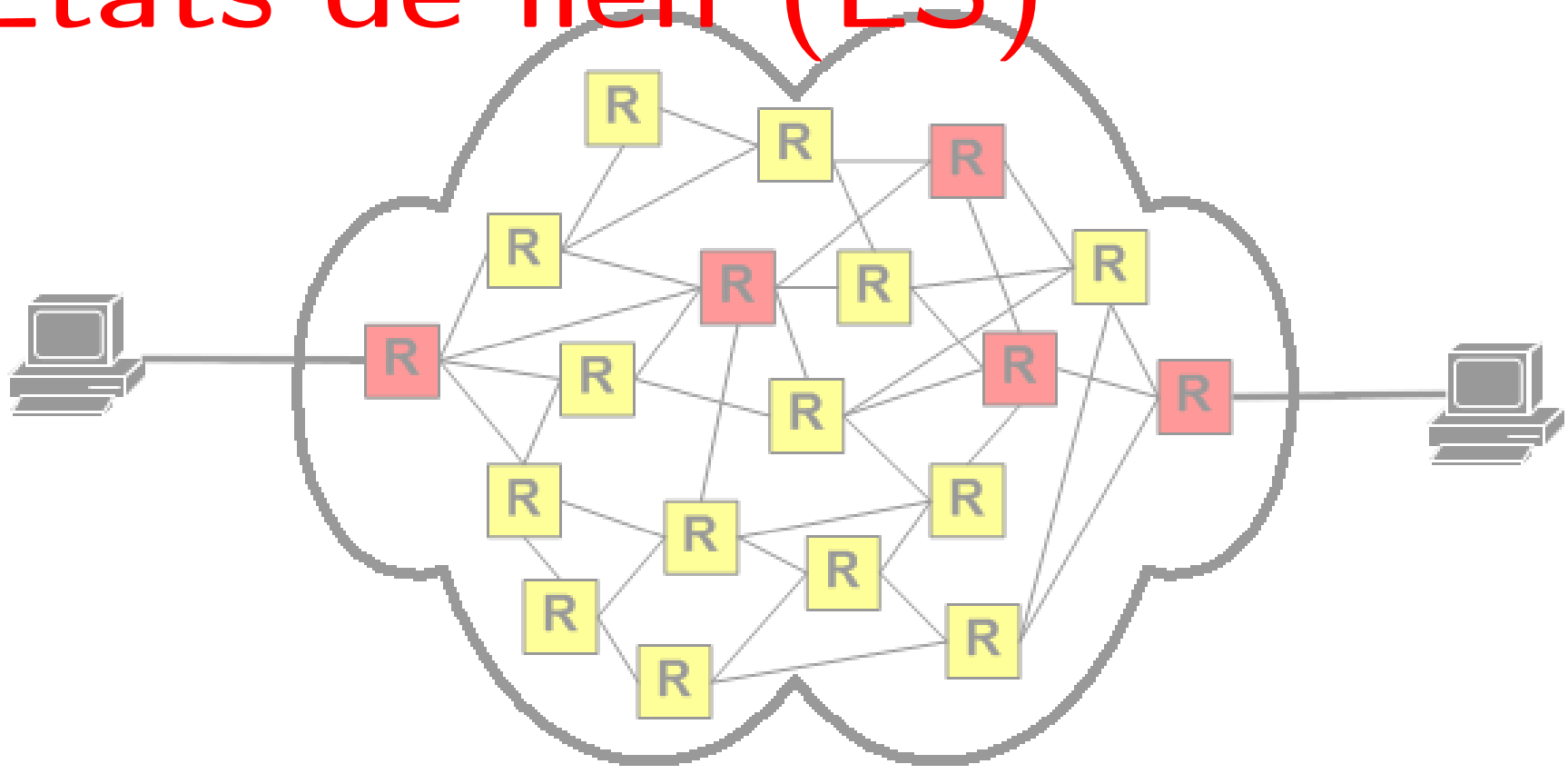
Donnée de routage

AFI : Adress Family Identifier
2 pour IPv4
Route tag : marqueur peu utilisé, pour des routes apprises hors RIP



Next hop : peu utilisé, normalement le prochain saut est l'émetteur du message, mais dans certains cas (e.g. interconnexions RIP/OSPF), on peut gagner un saut.

États de lien (LS)



Dijkstra

Seconde option : distribué à connaissance globale.

- Algorithme de `Dijkstra` pour calculer les plus courts chemins dans le graphe.
- Chaque routeur connaît le coût de ses liens (calcul ou hypothèse).
- Chaque routeur transmet à tous les autres (par `inondation`) la topologie locale.
- À la réception, le routeur reconstruit la carte du réseau et calcule un arbre des plus courts chemins.

Découvrir la topologie locale

- Chaque routeur émet un message `HELLO` en local pour découvrir ses voisins.
- Message envoyé en multicast sur chaque interface participant au routage.
- Le message contient l'adresse de l'émetteur pour la réponse.
- Message envoyé périodiquement pour détecter les changements de topologie.

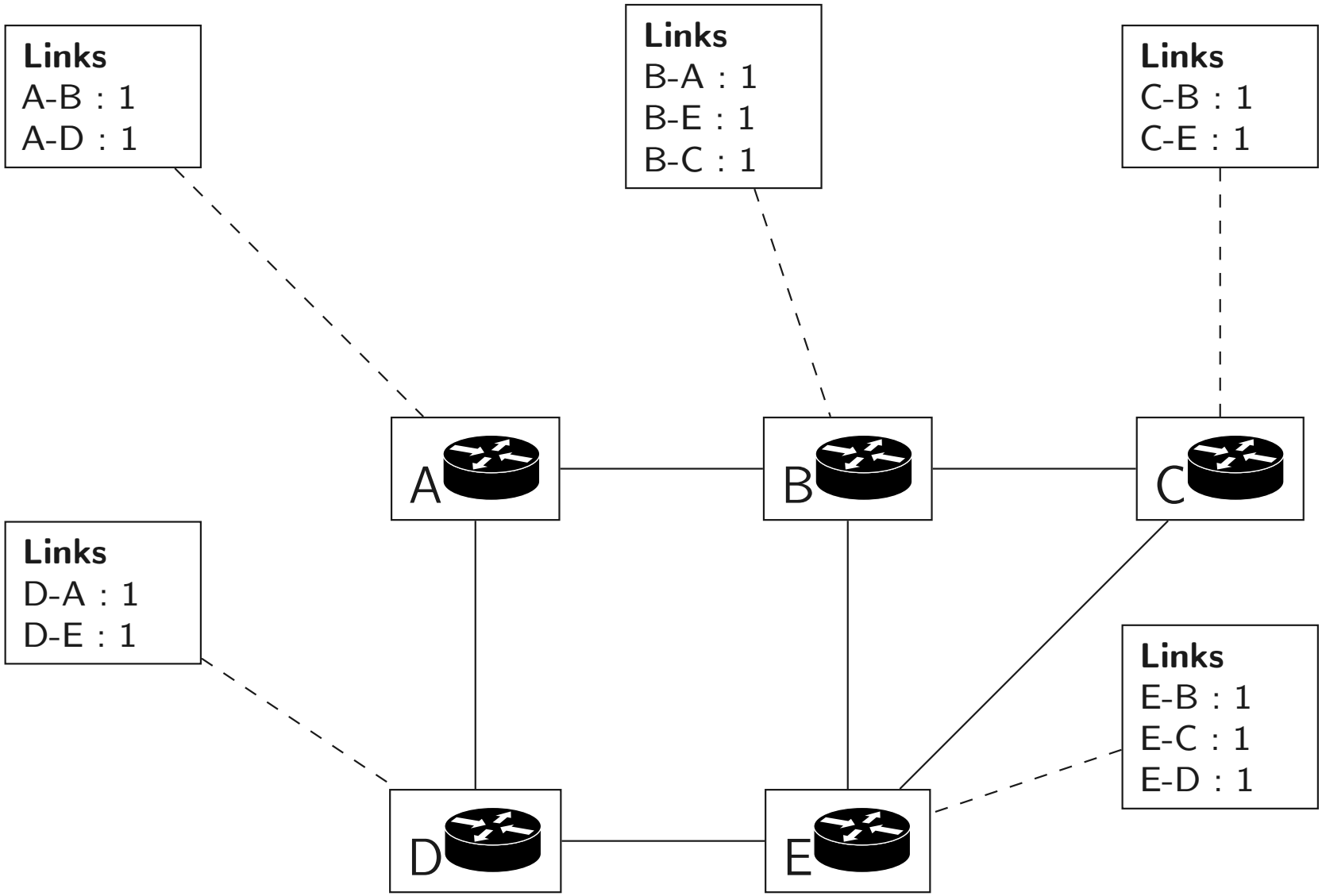
Le coût associé à un lien est choisi selon le contexte en fonction de différents critères : coût unité, proportionnel au débit ou à la latence, inversement proportionnel à la capacité...

Communiquer la topologie locale

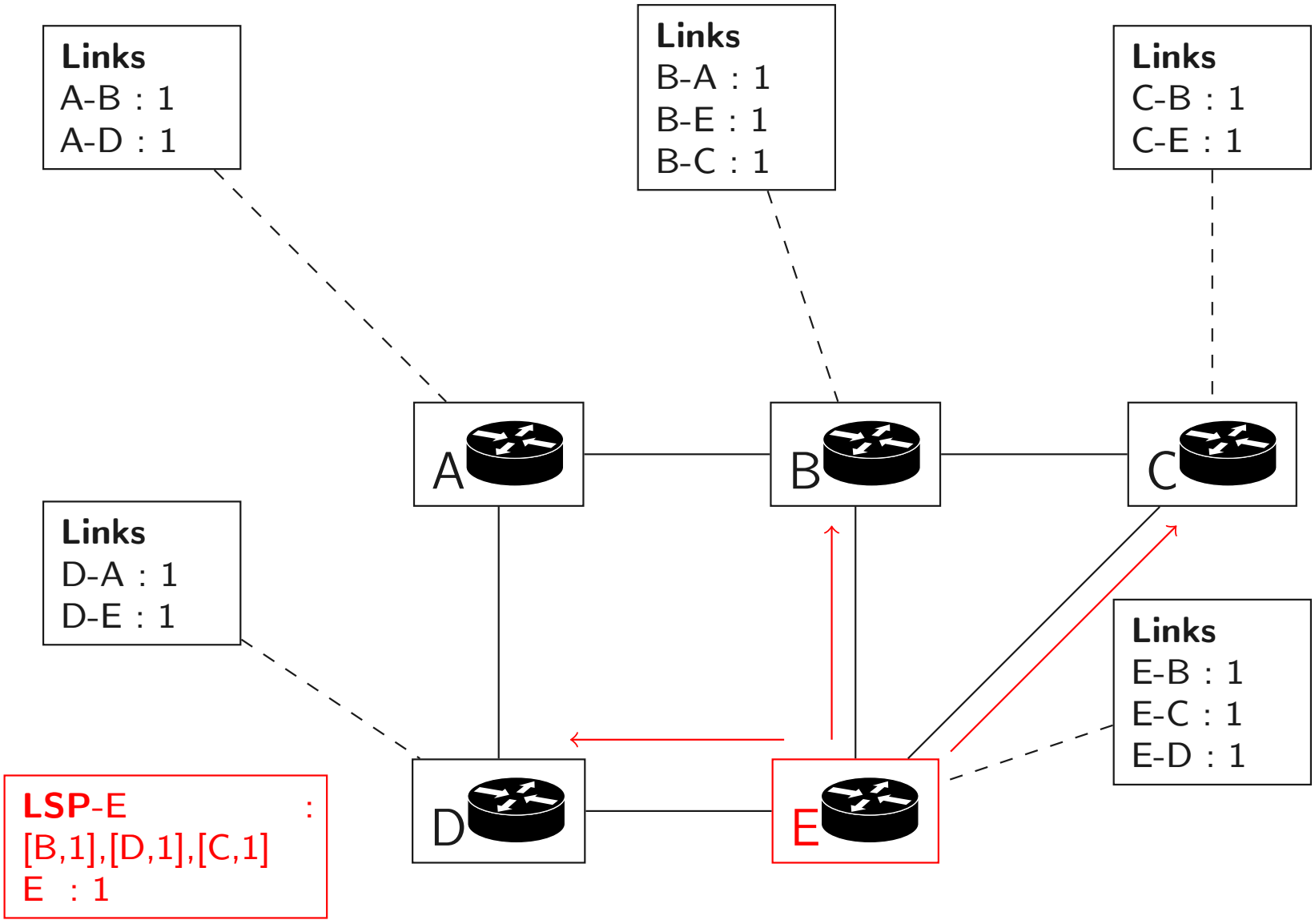
- Chaque routeur émet un message d'`états de liens` à tous les routeurs voisins.
- Message envoyé en multicast sur chaque interface participant au routage.
- Le message contient la liste des liens locaux :
  - vers des routeurs participant au routage LS ;
  - vers des réseaux sur lesquels on ne fait pas de routage LS.
- Message envoyé périodiquement et aux changements de topologie.

Message envoyé par `inondation des routeurs`

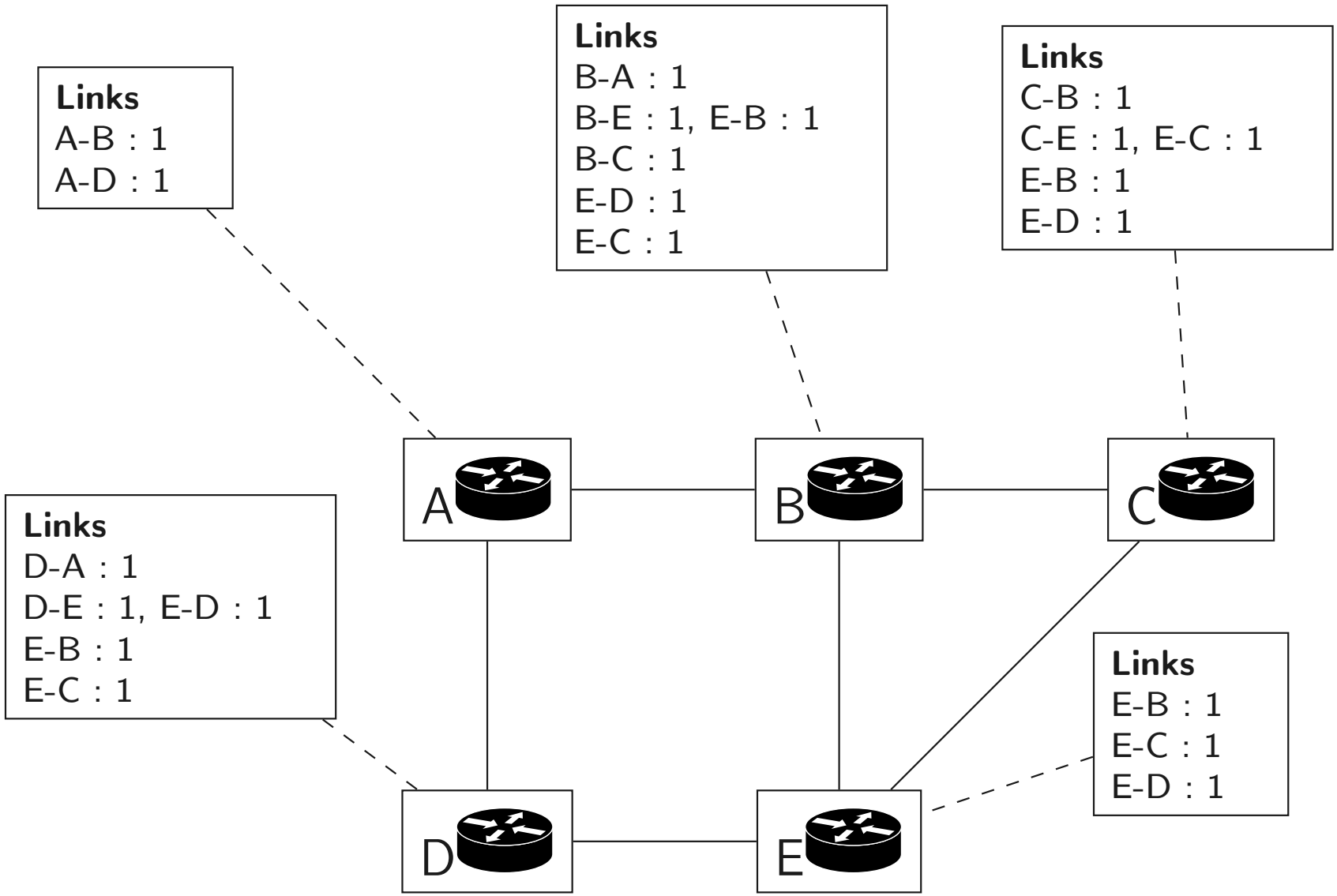
Envoi (coûts égaux à 1)



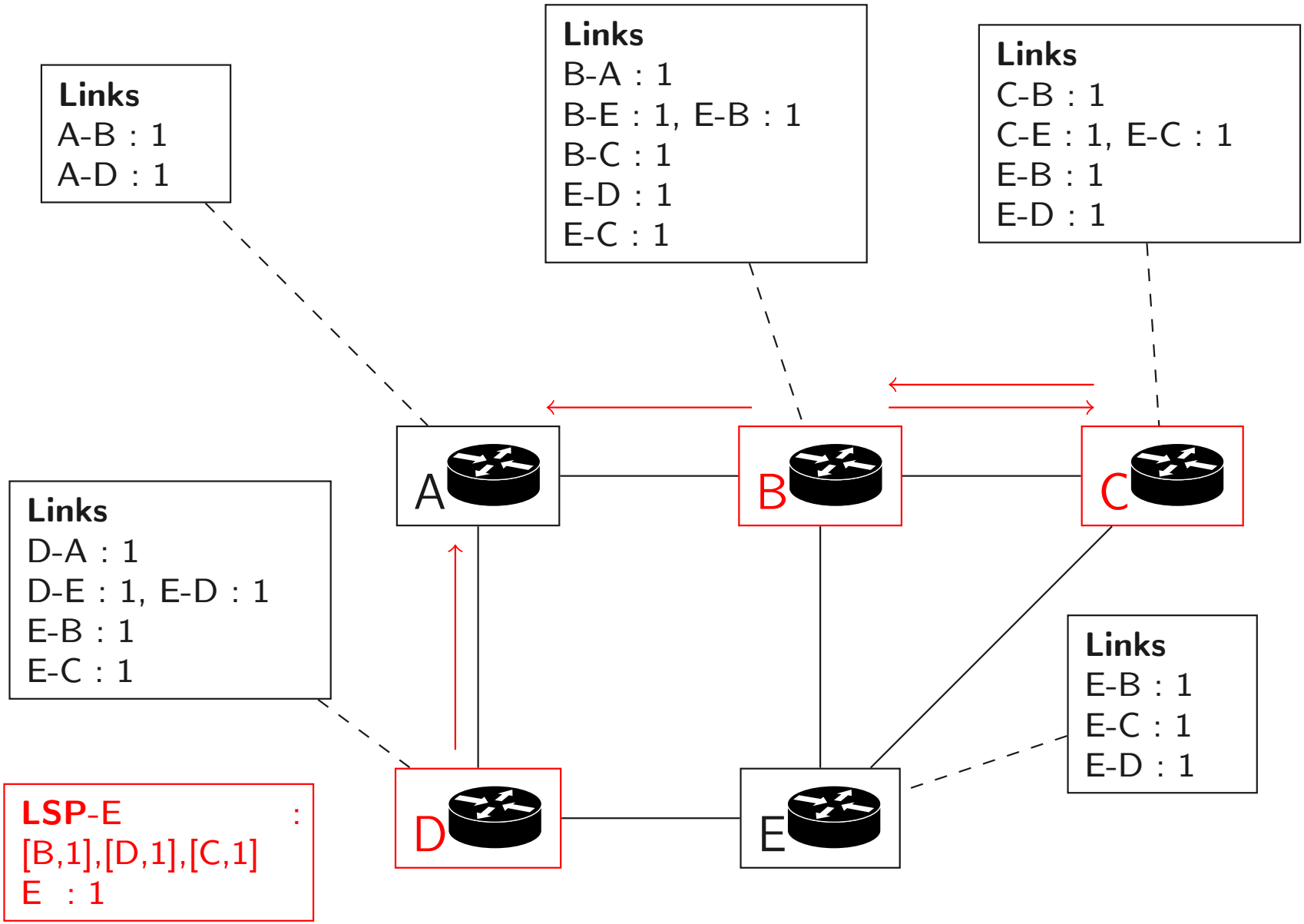
Envoi (coûts égaux à 1)



Envoi (coûts égaux à 1)



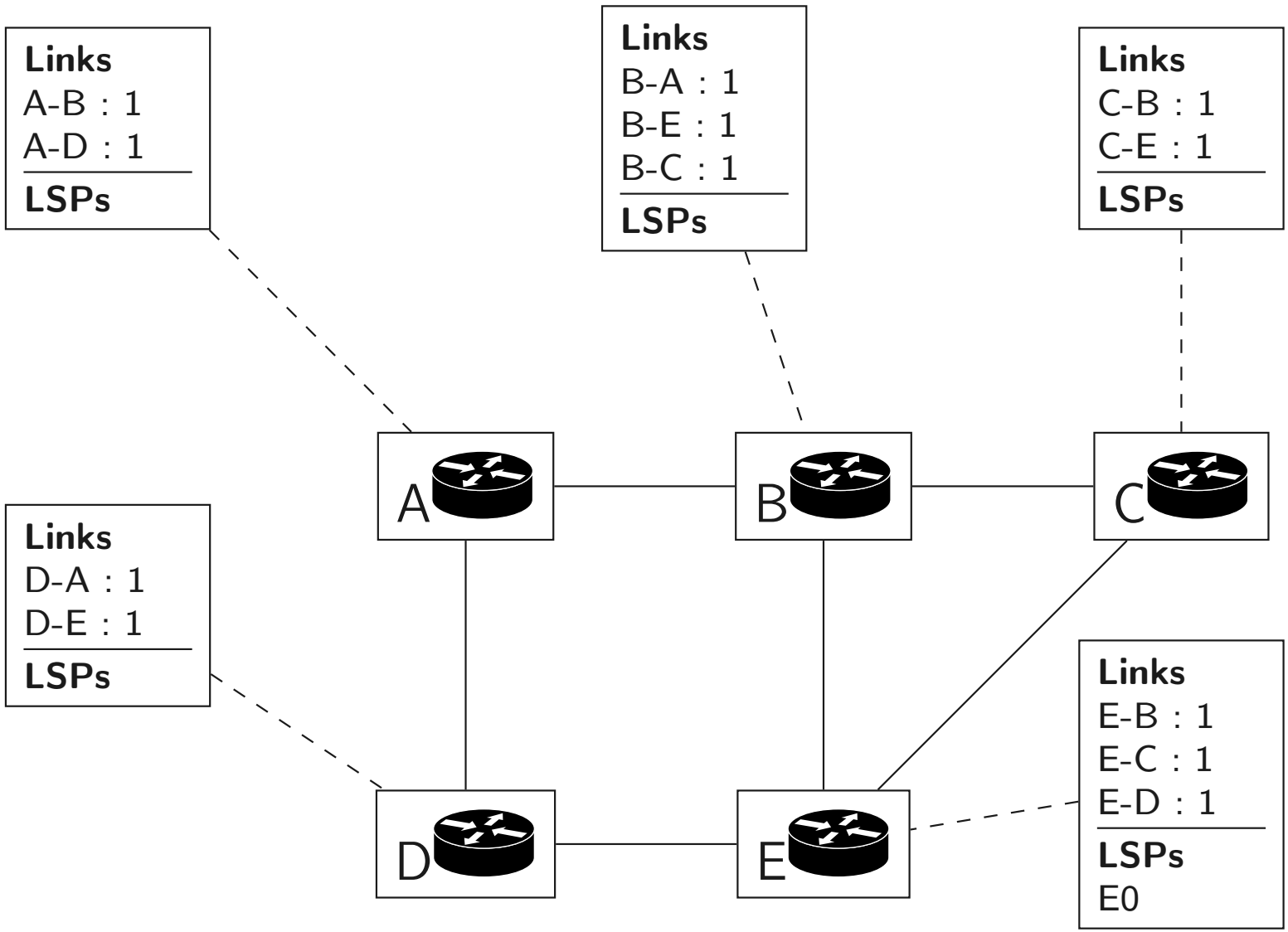
Envoi (coûts égaux à 1)



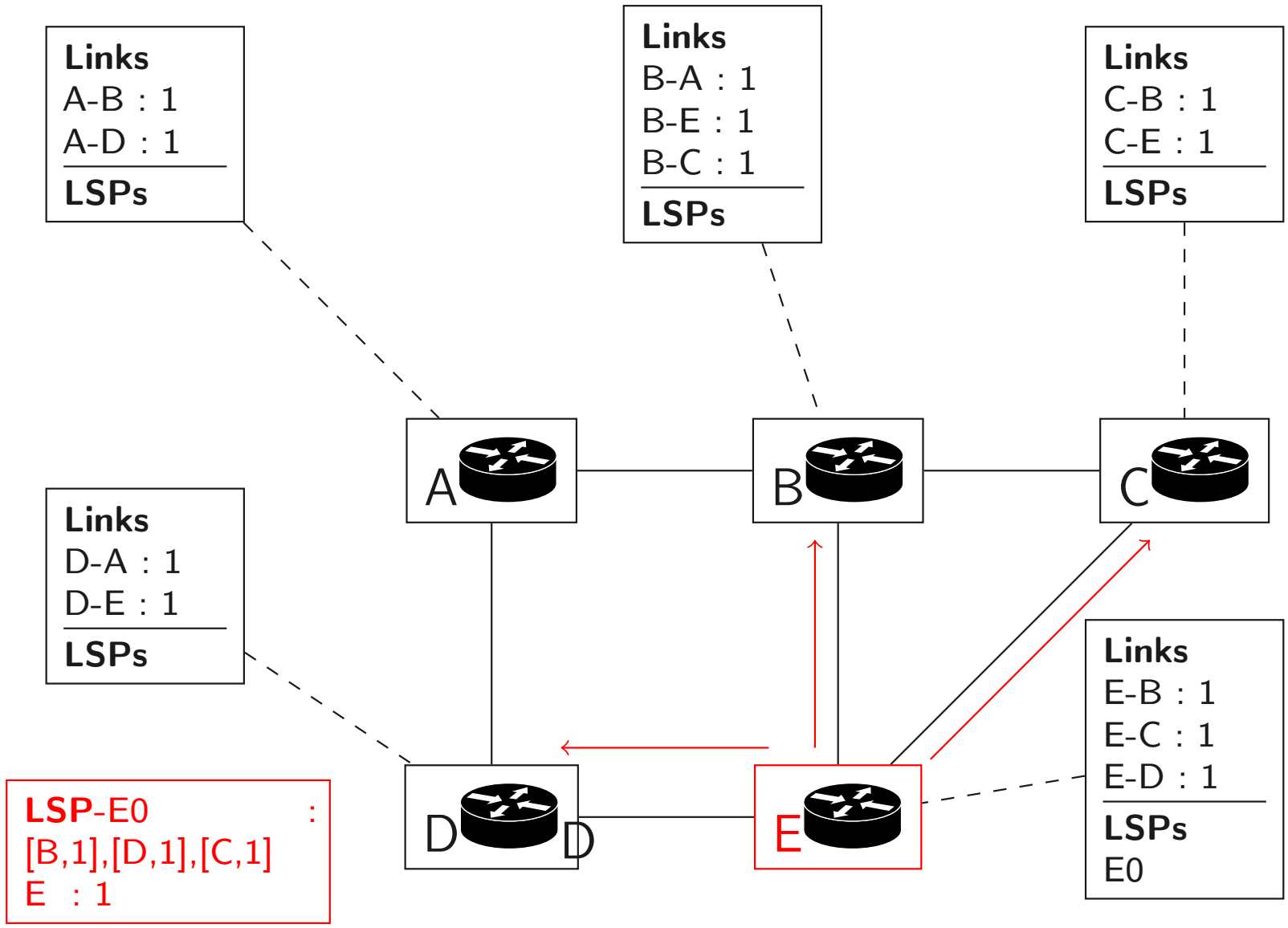
Envois de LSP

- Chaque routeur transmet les LSP à tous ses voisins excepté celui par qui il a reçu le paquet.
- Pour éviter les boucles, il ne faut pas transmettre plusieurs fois le même LSP.
- Identification par adresse du routeur émetteur et numéro de série incrémenté par l'émetteur à chaque envoi.
- Les routeurs mémorisent le LSP (émetteur, numéro et liste de liens) le plus récent de chaque émetteur.

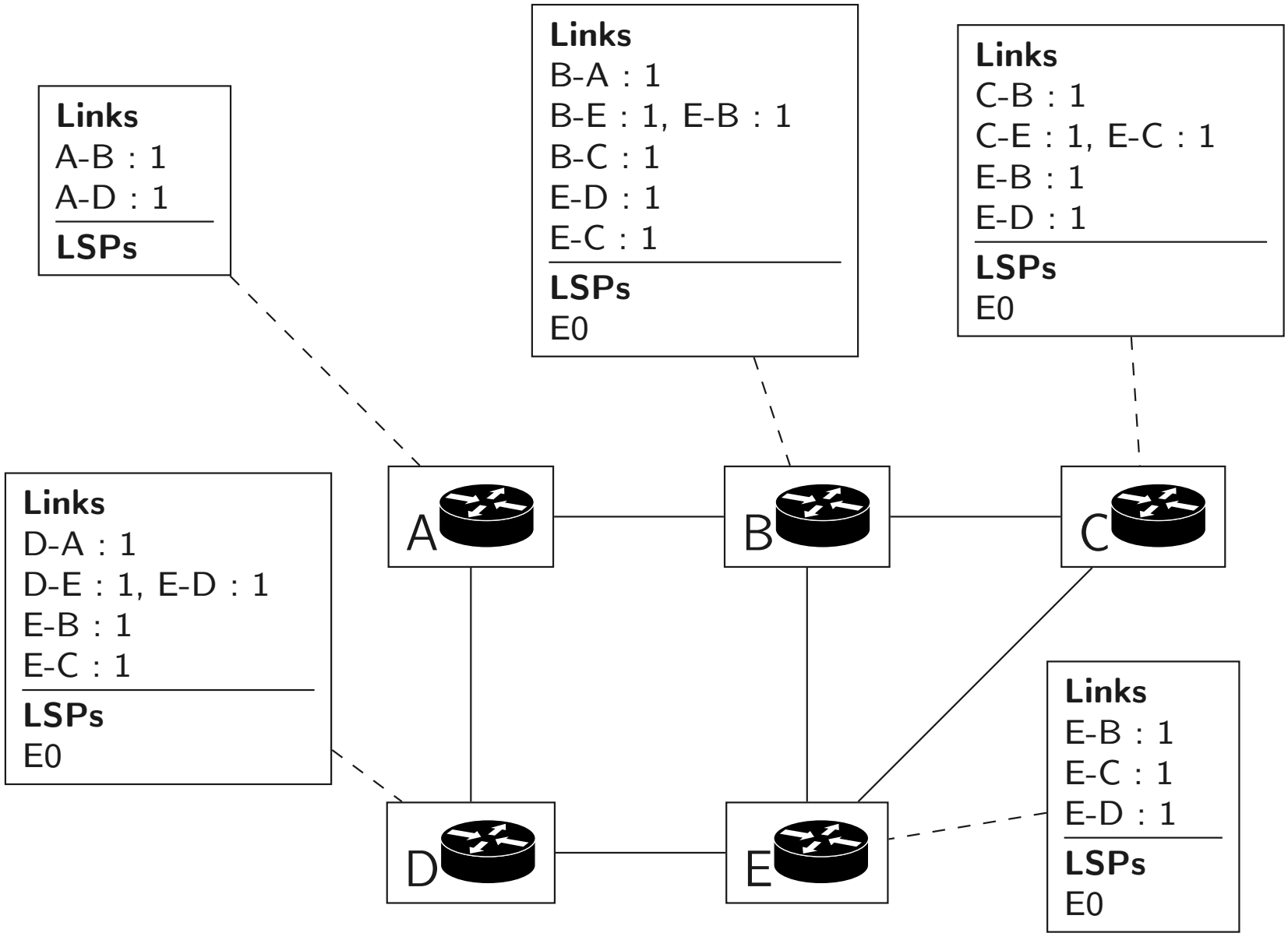
Envoi avec numéros de série



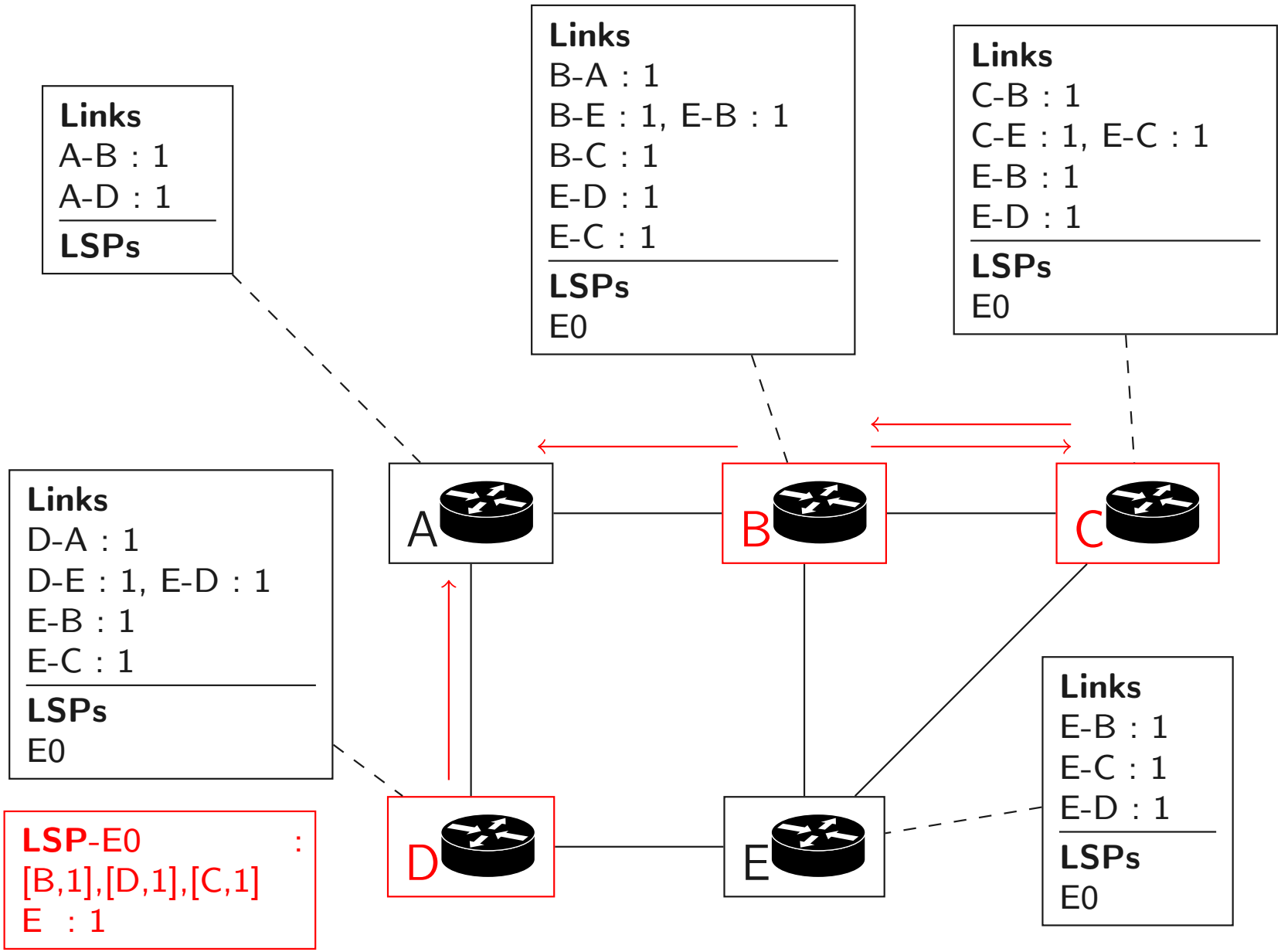
Envoi avec numéros de série



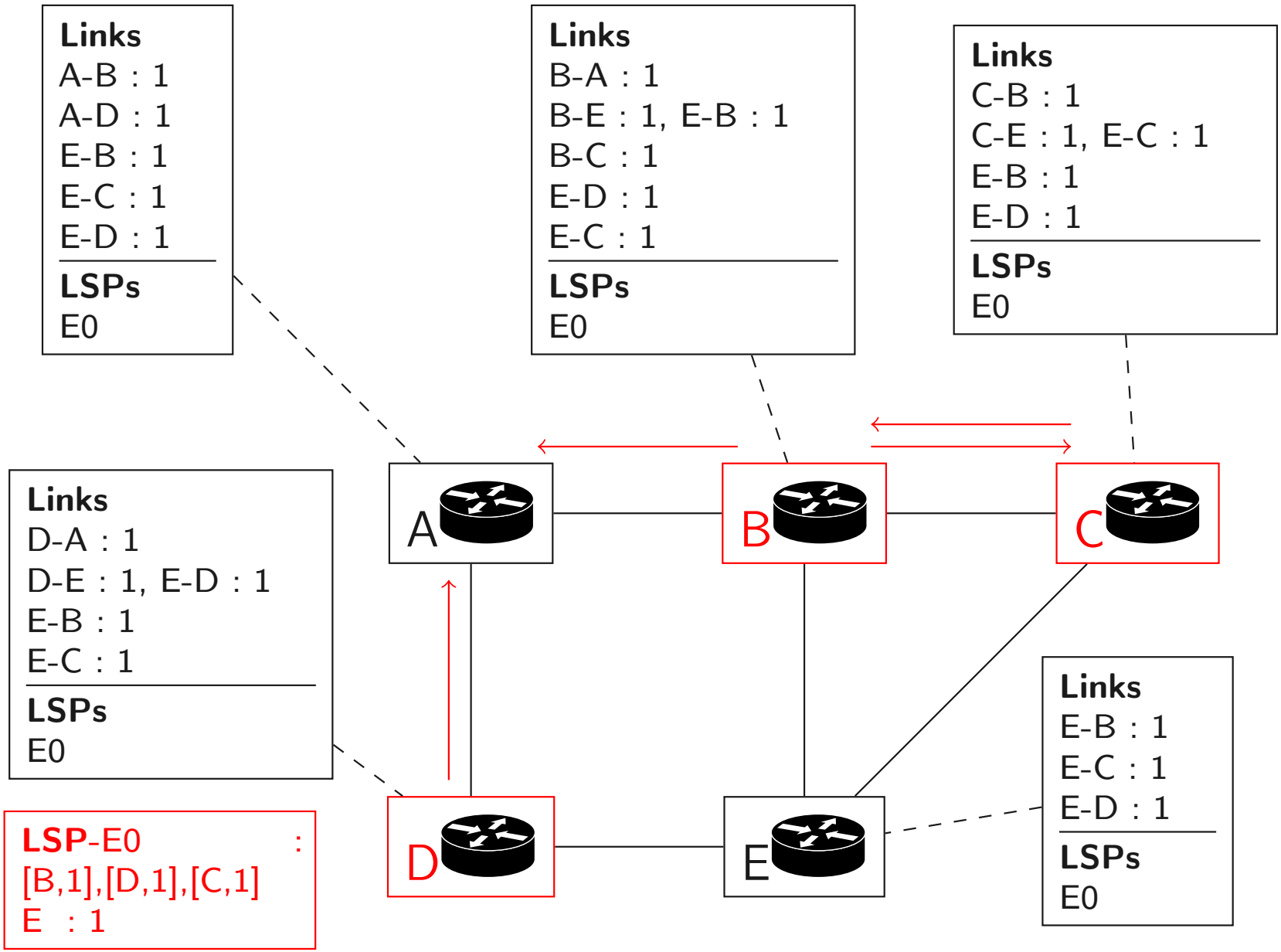
Envoi avec numéros de série



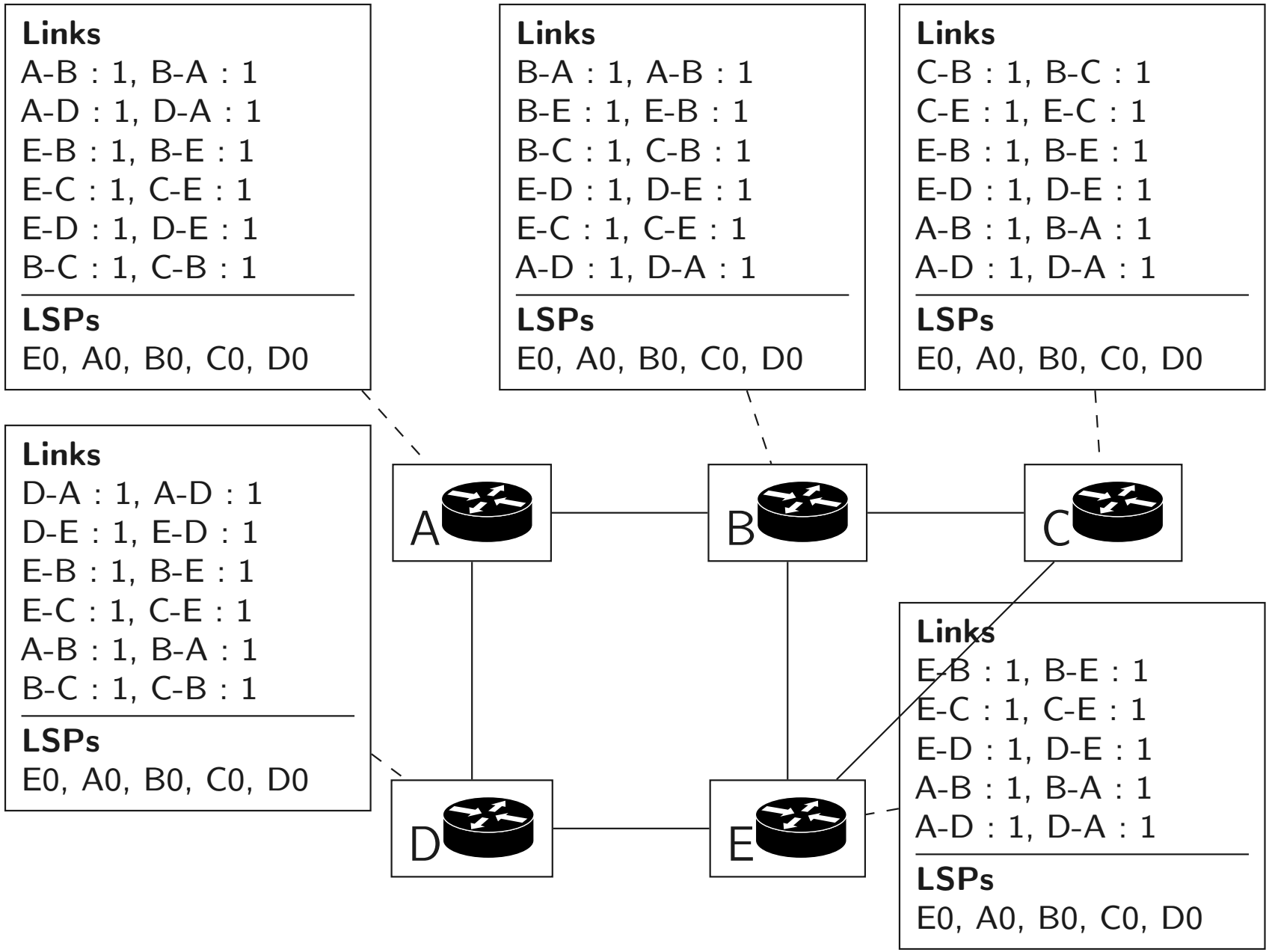
Envoi avec numéros de série



A, *B*, *C* détectent les doublons



Situation stable



Calculer les chemins

Une fois la base de données des LSPs constituée :

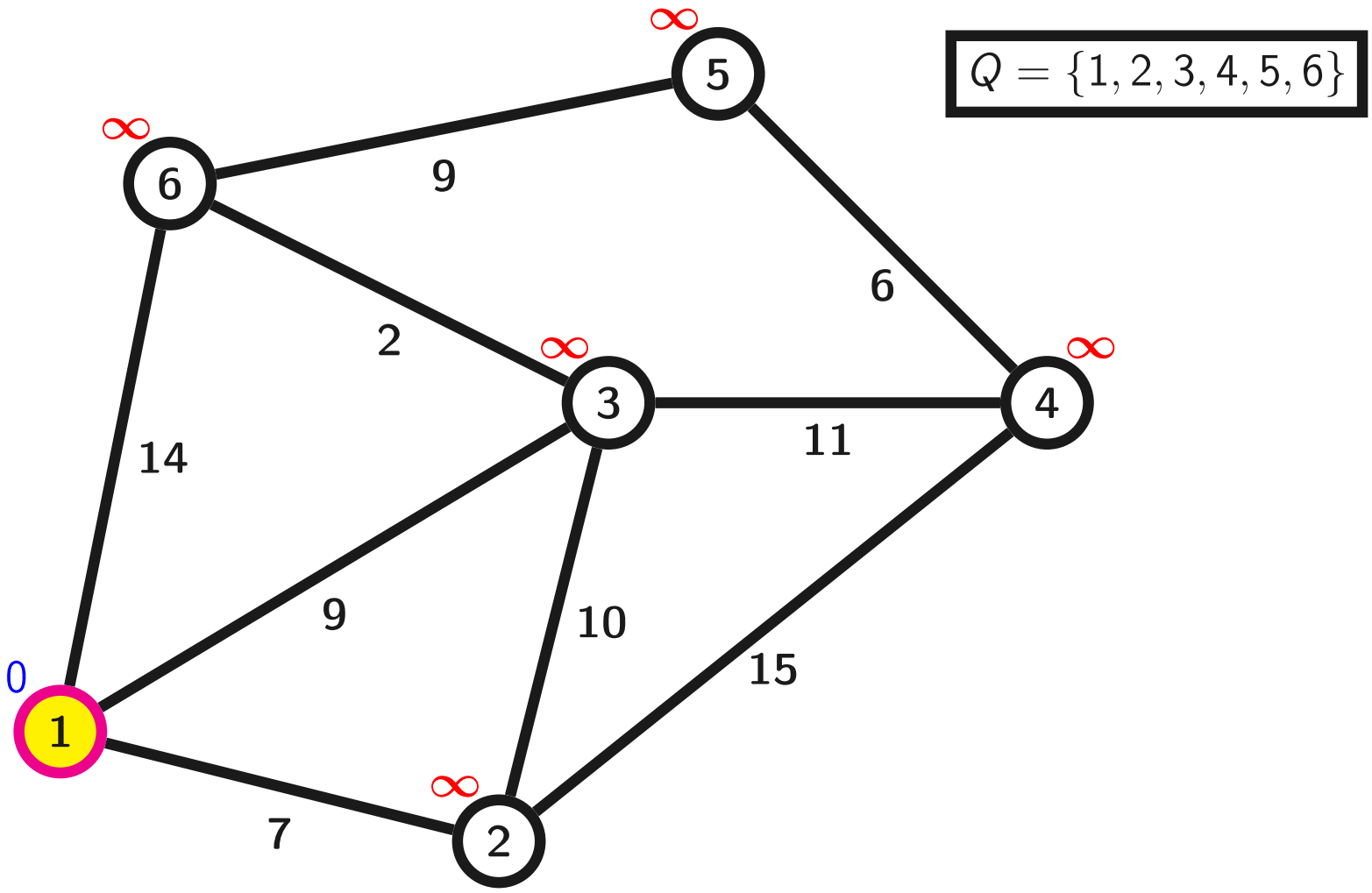
- chaque routeur peut reconstruire la topologie entière ;
- puis calculer un arbre des plus courts chemins.
- algorithme de Dijkstra

Algorithme de Dijkstra

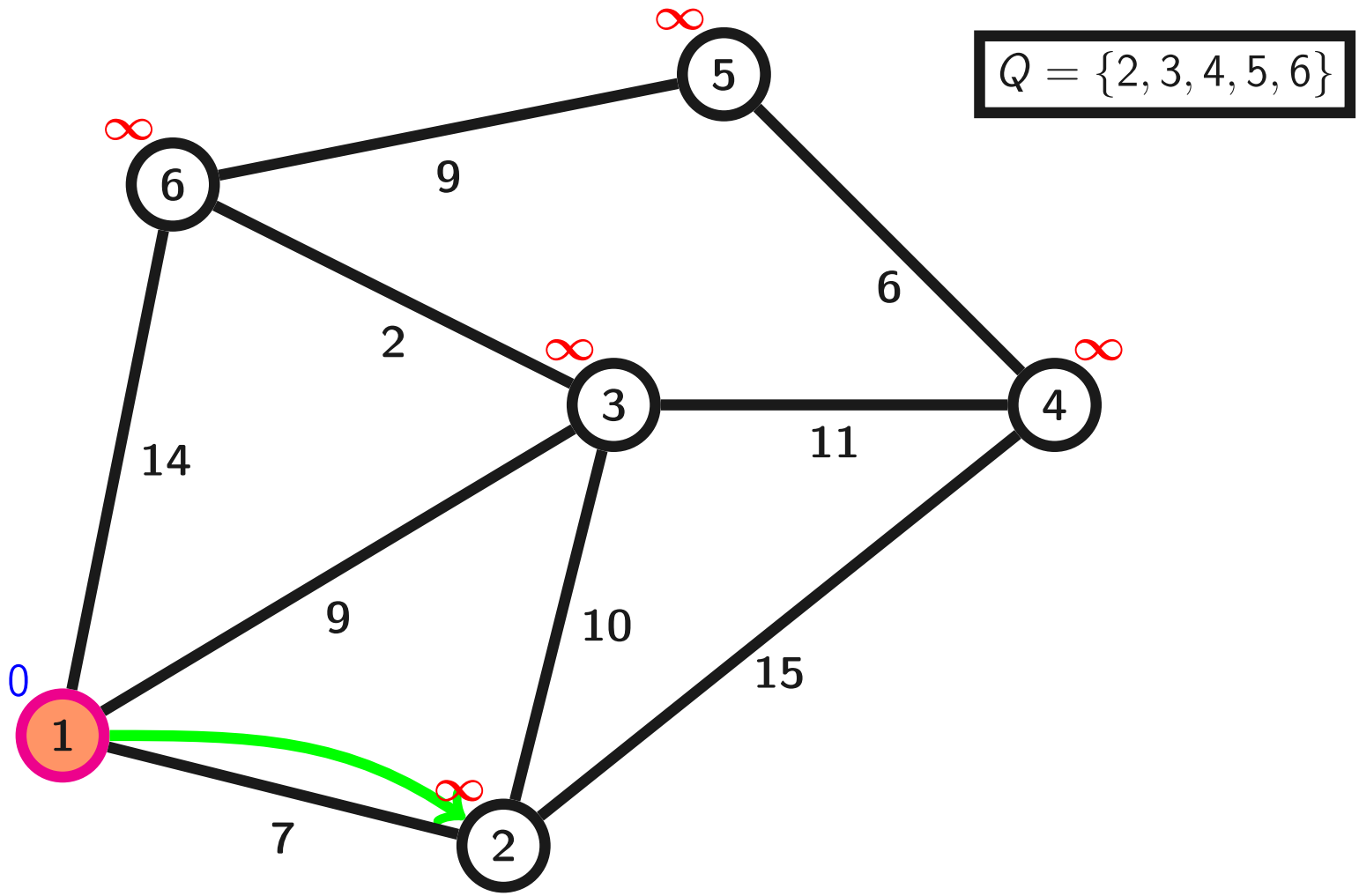
Un algorithme en $O(|E| + |V| \log |V|)$.

```
for each vertex v in Graph {
    dist[v]  = infinity
    previous[v]  = undefined
}
dist[source] = 0
Q = the set of all vertices
while Q is not empty {
    u = vertex in Q with smallest distance in dist[]
    remove u from Q
    for each neighbor v of u {
        alt = dist[u] + dist_between(u, v)
        if alt < dist[v] {
            dist[v]  = alt
            previous[v] = u
        }
    }
}
```

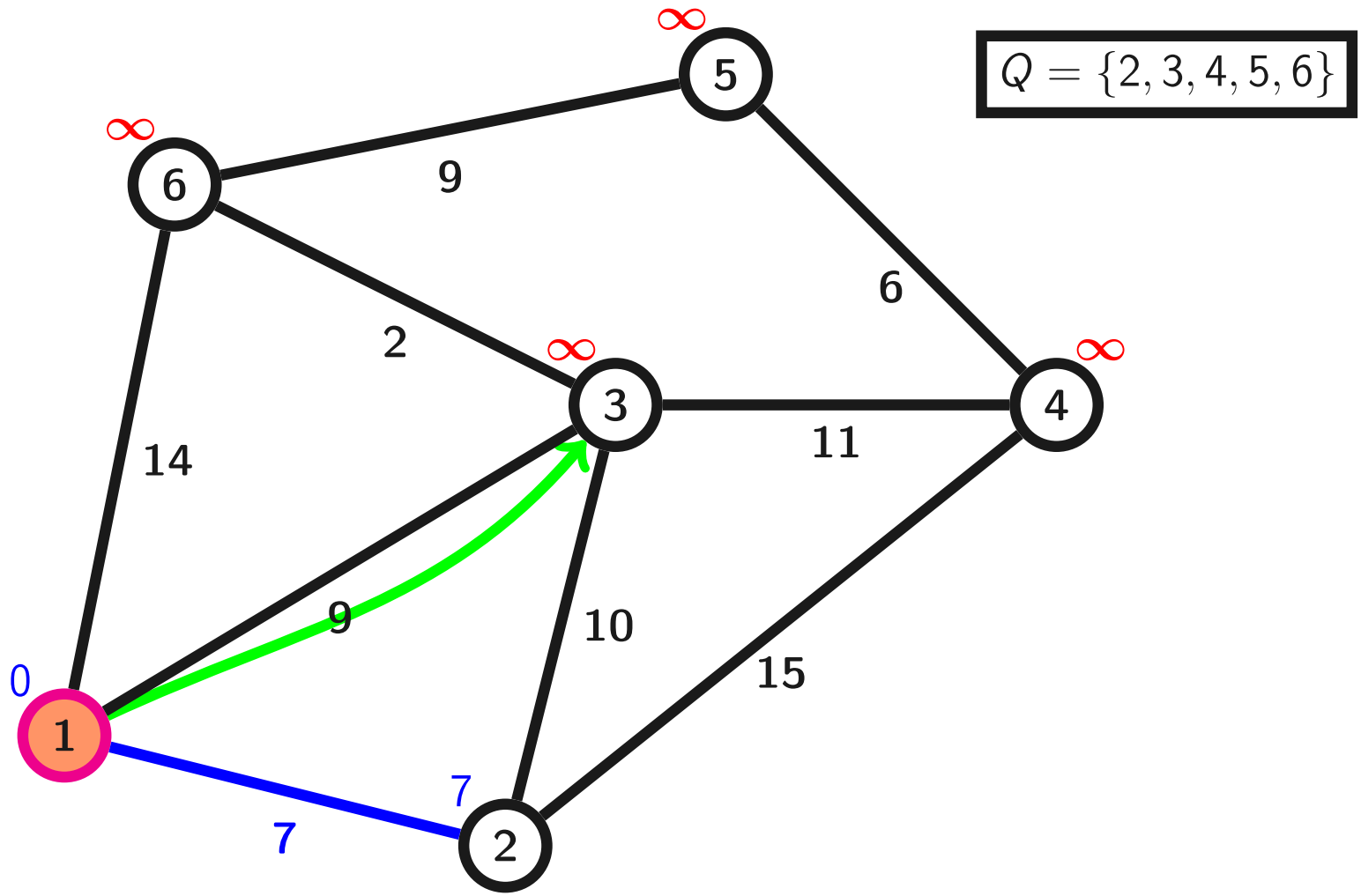
Dijkstra : exemple



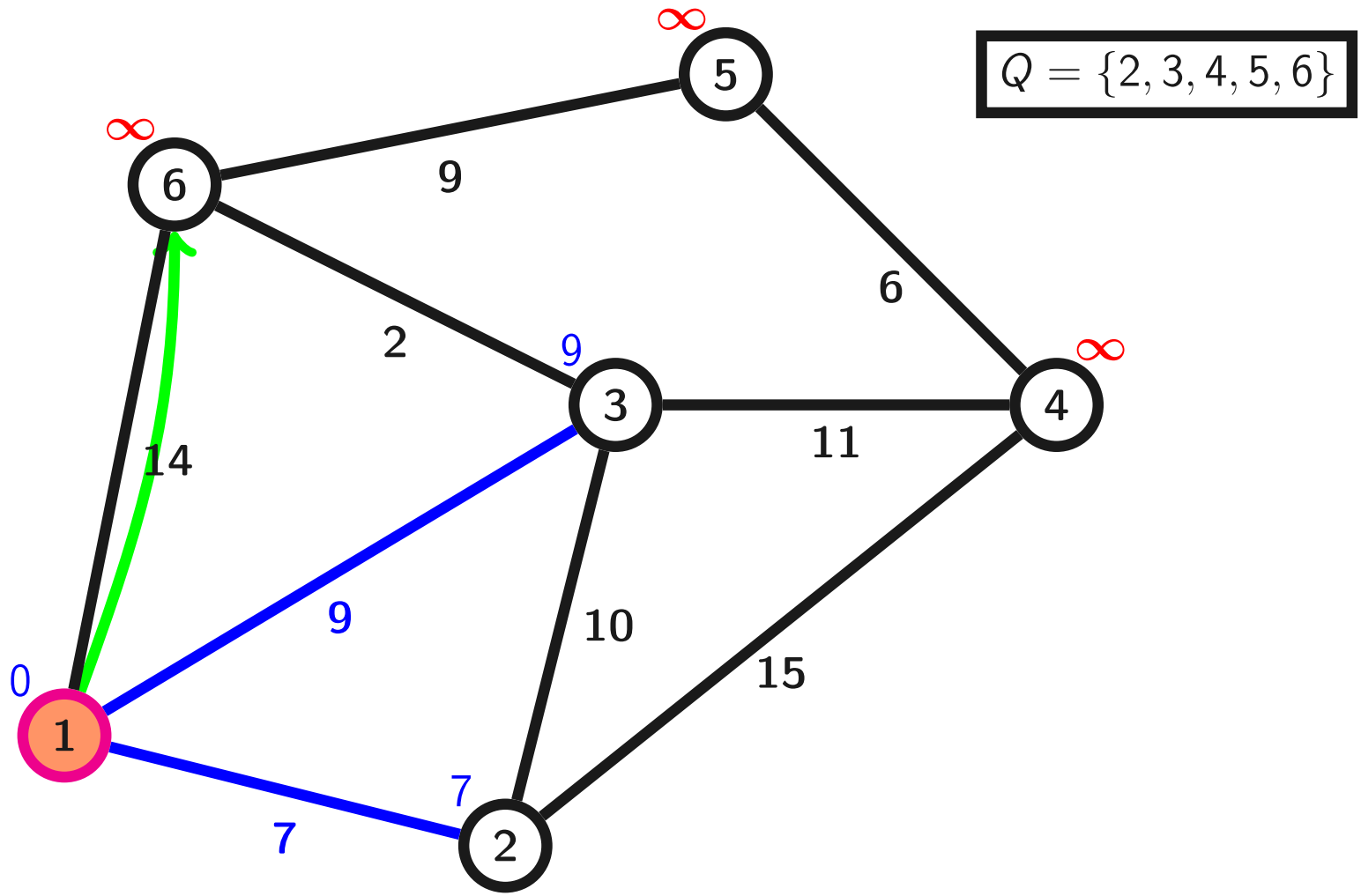
Dijkstra : exemple



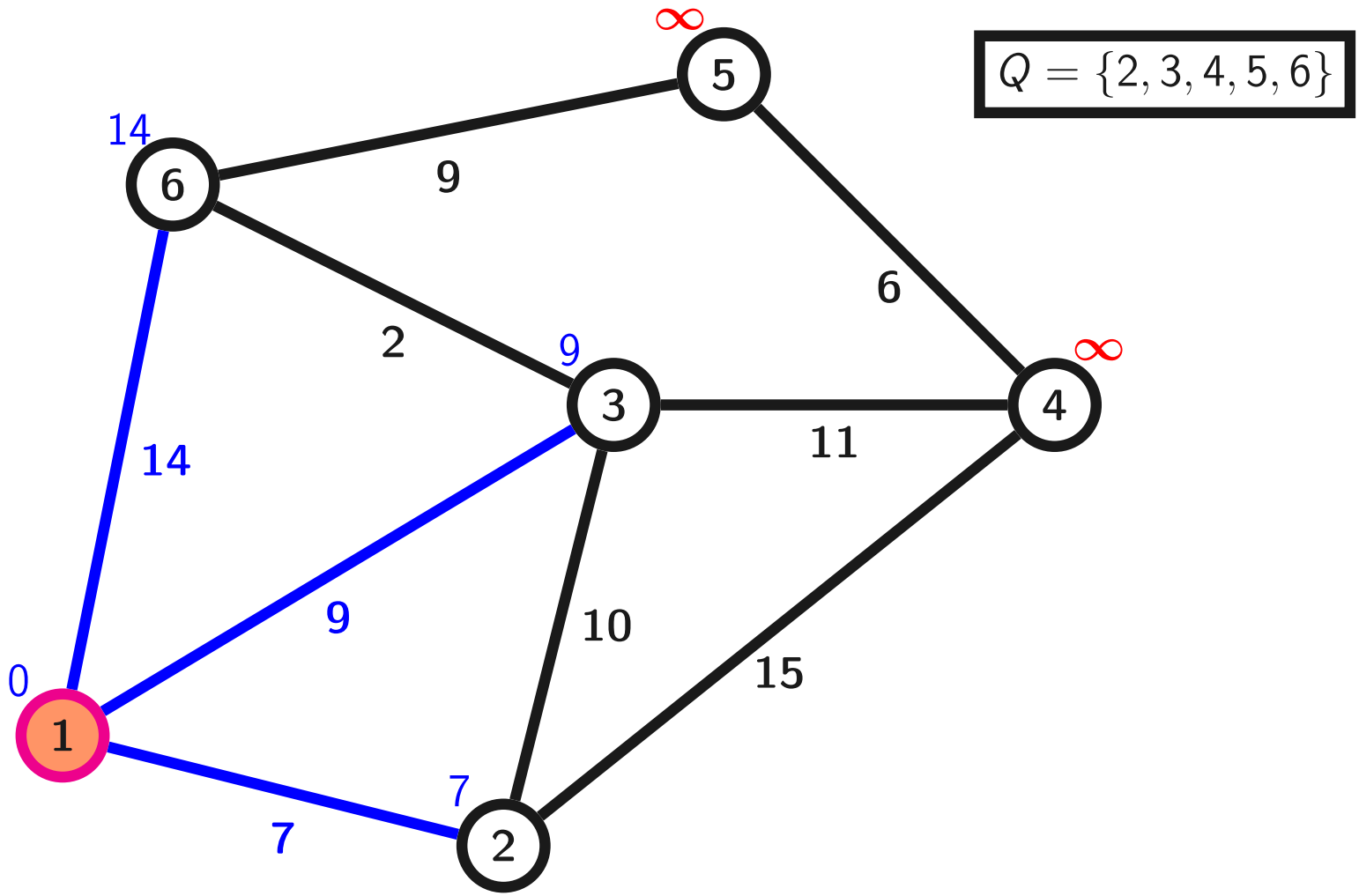
Dijkstra : exemple



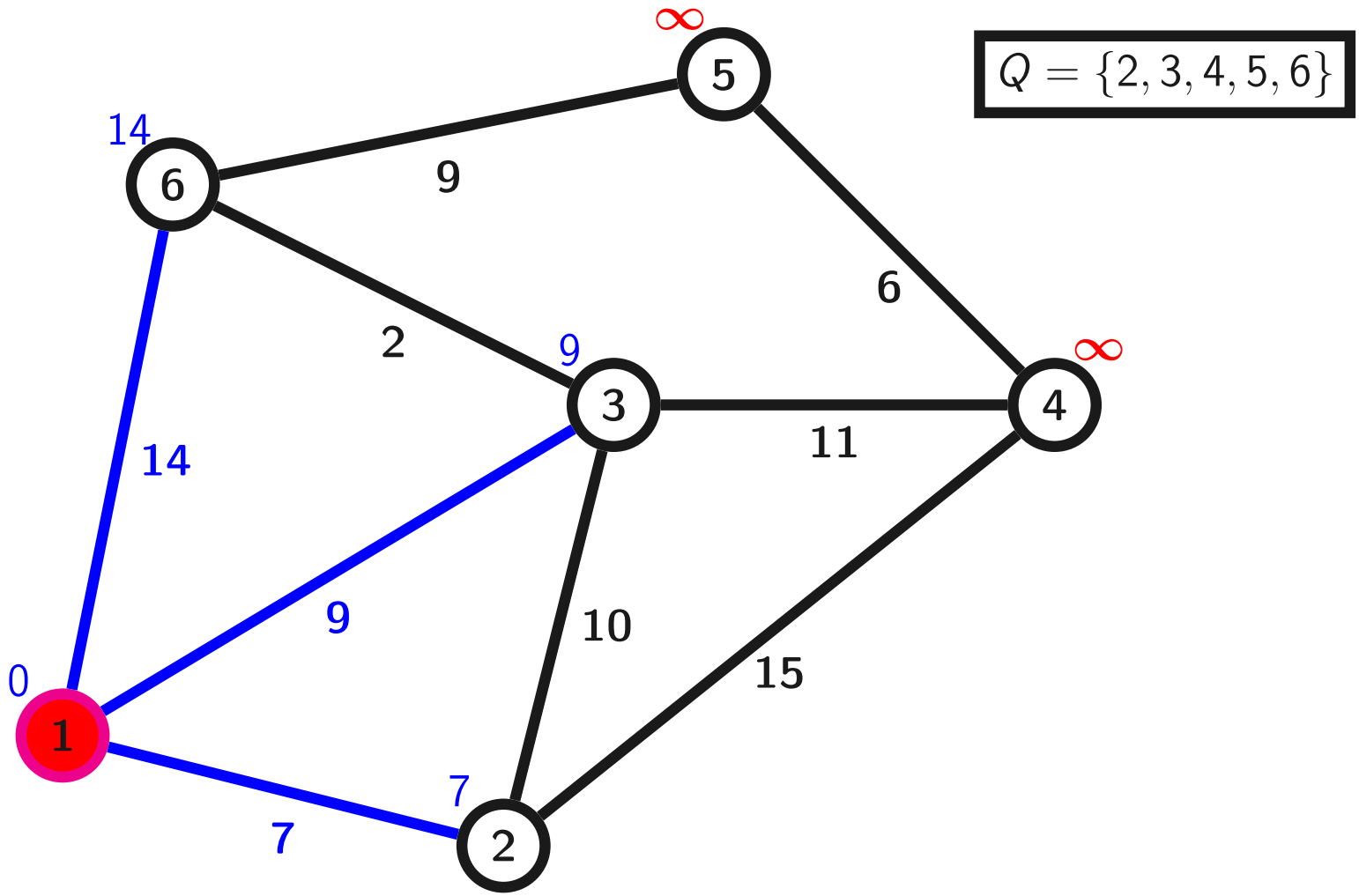
Dijkstra : exemple



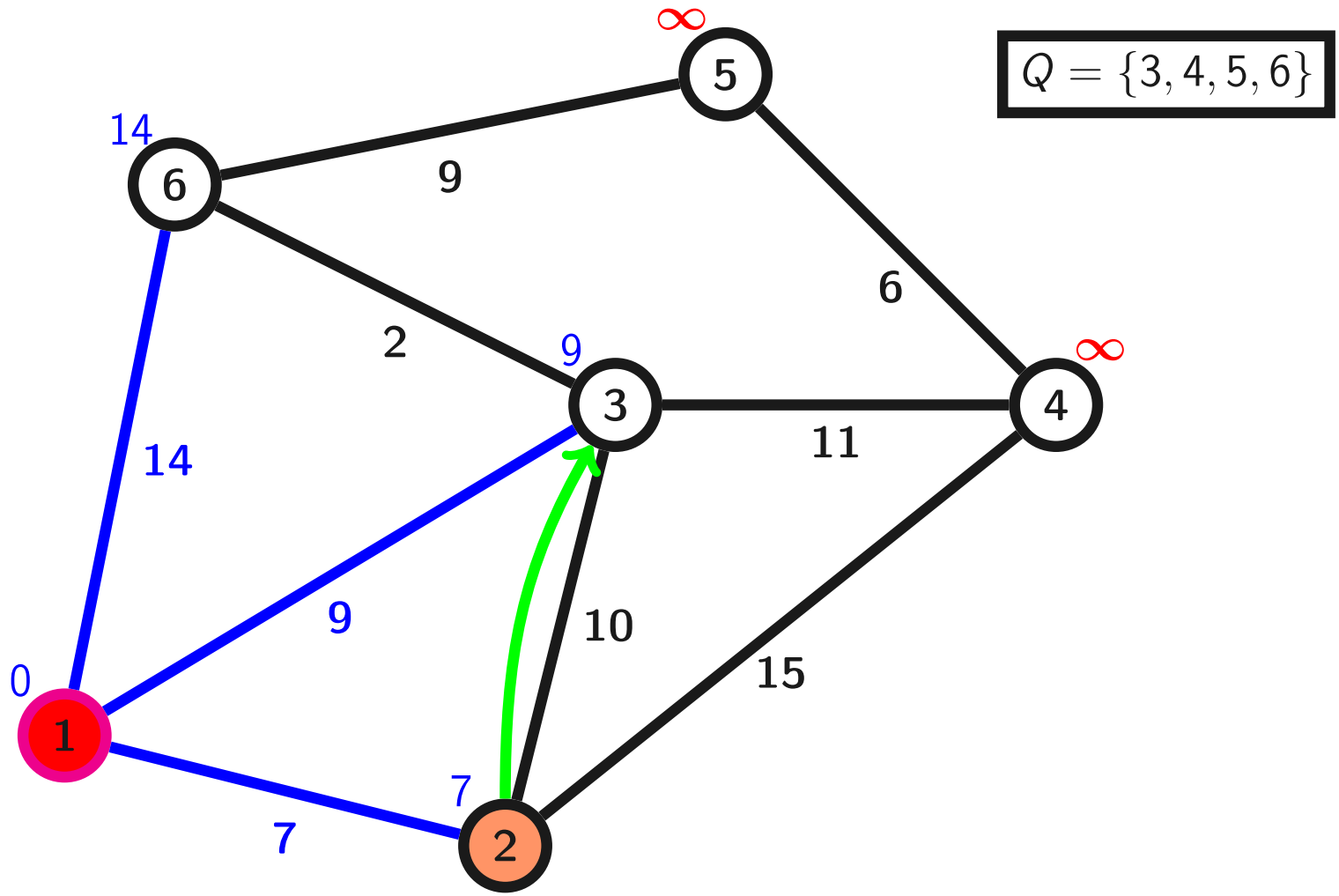
Dijkstra : exemple



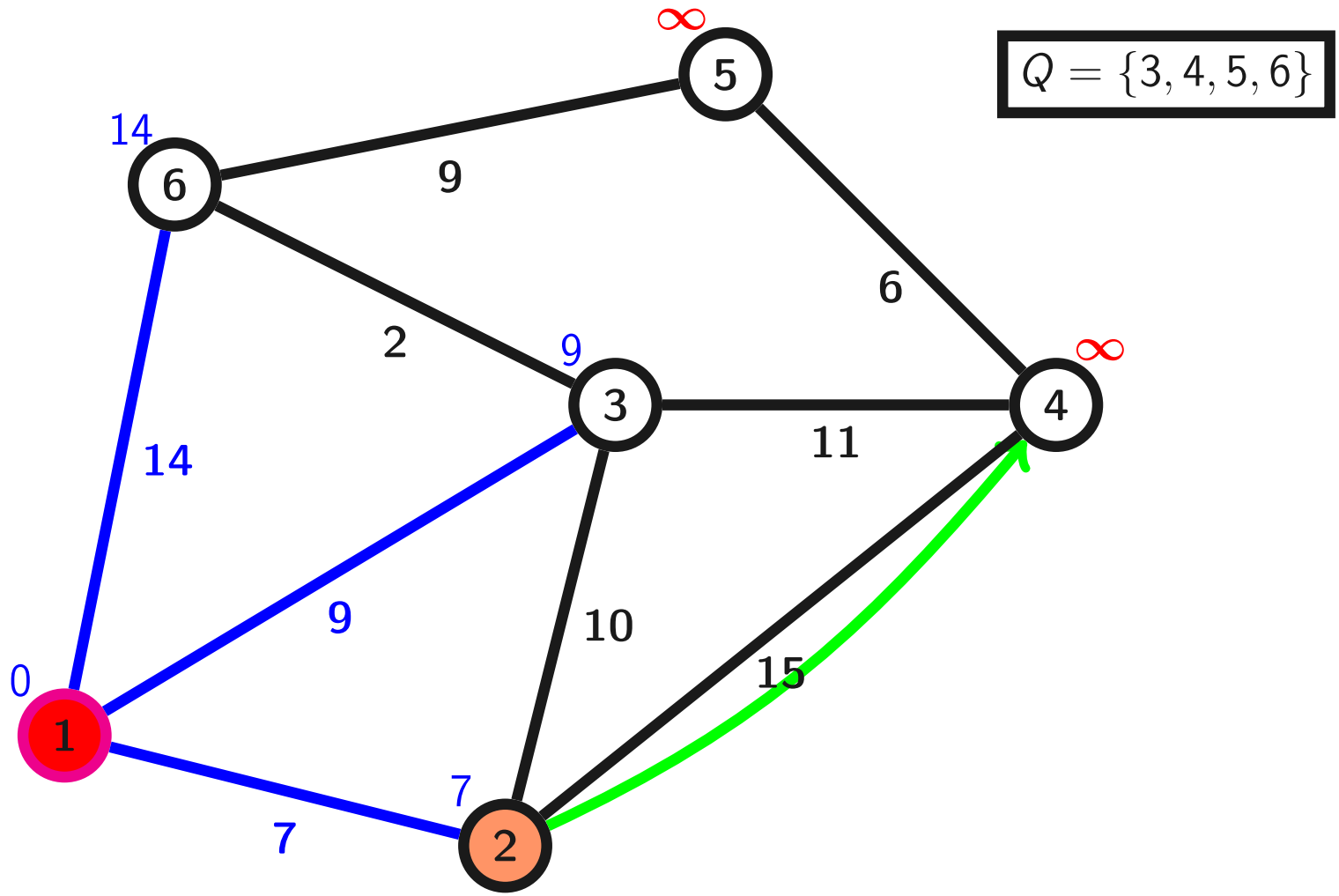
Dijkstra : exemple



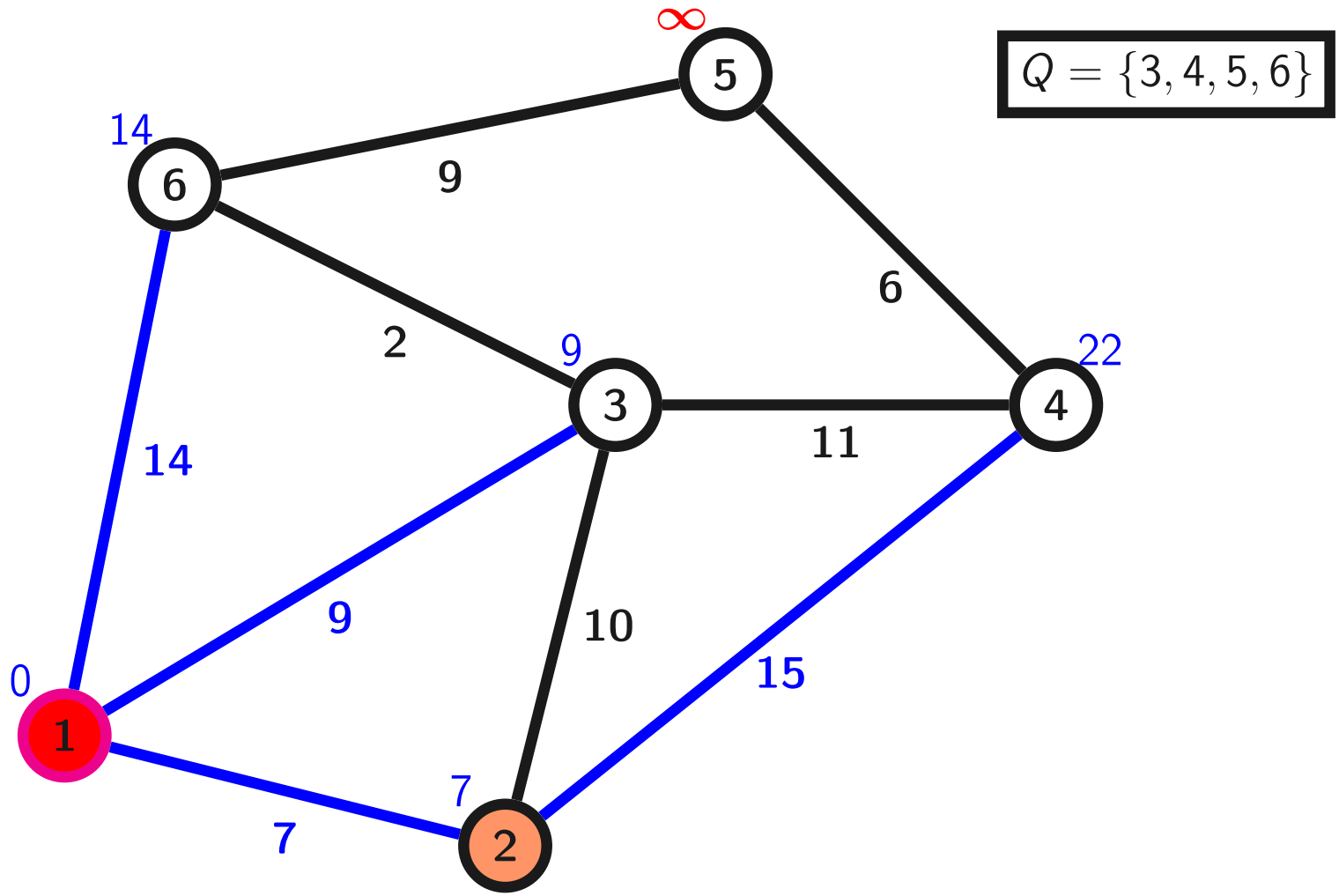
Dijkstra : exemple



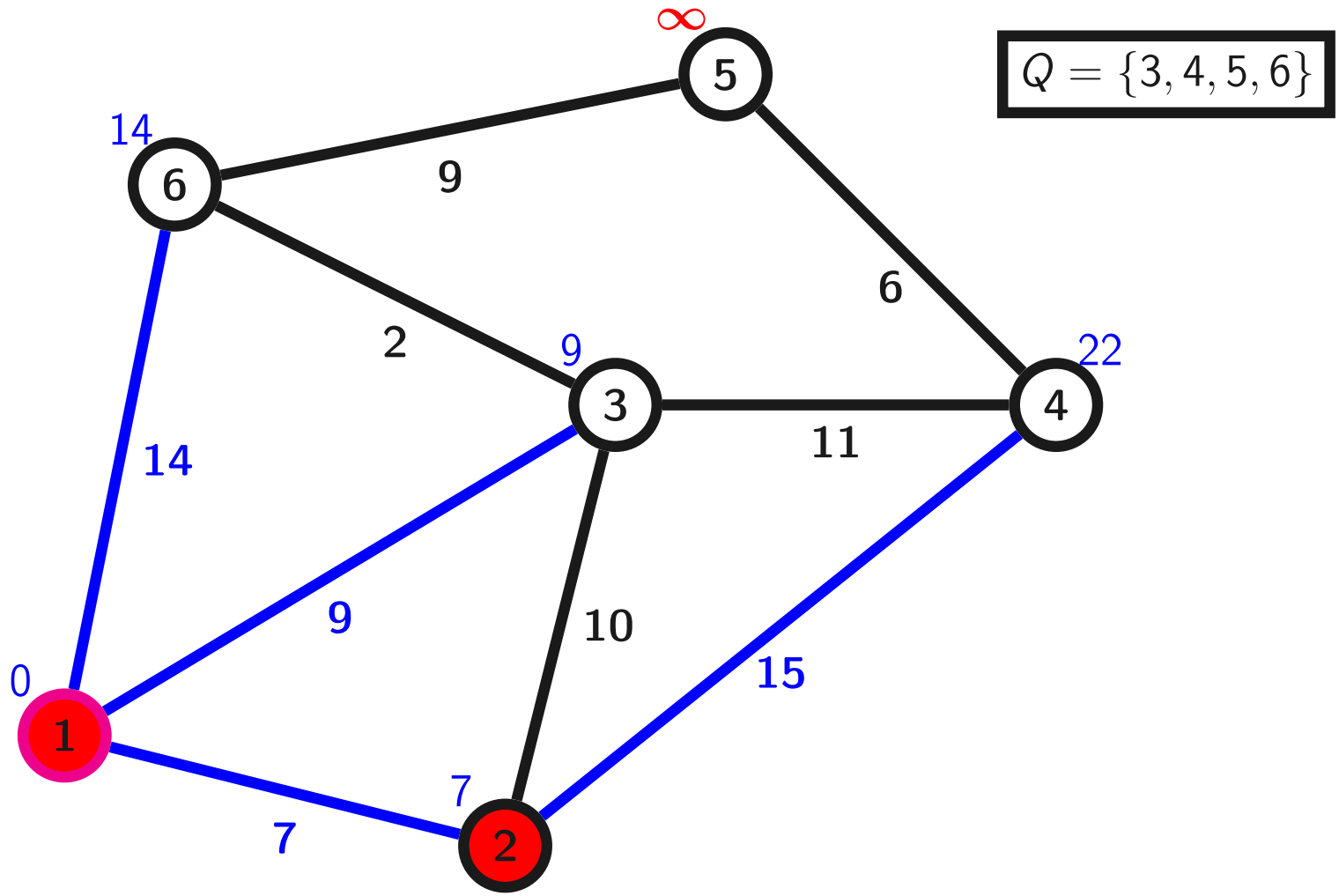
Dijkstra : exemple



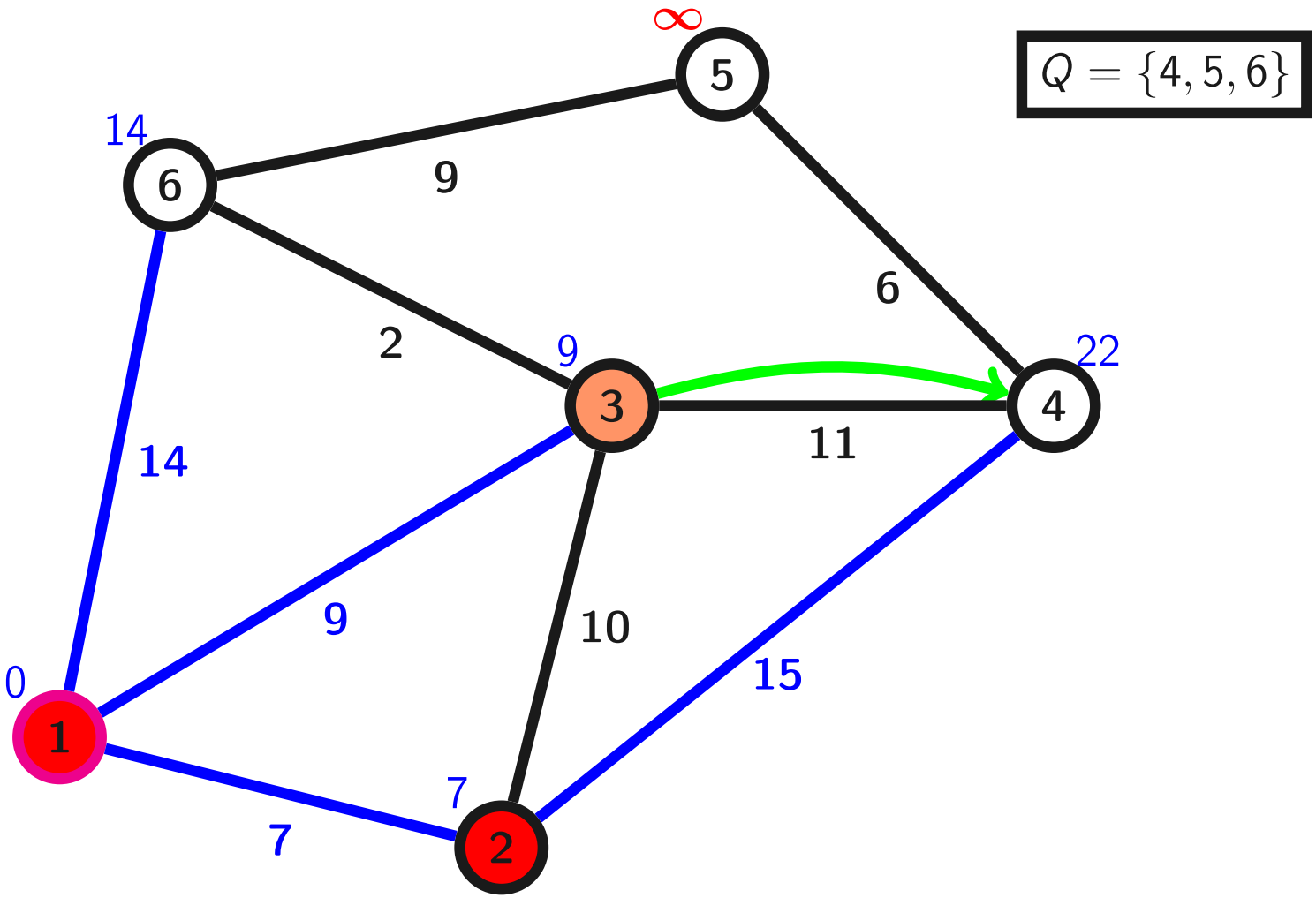
Dijkstra : exemple



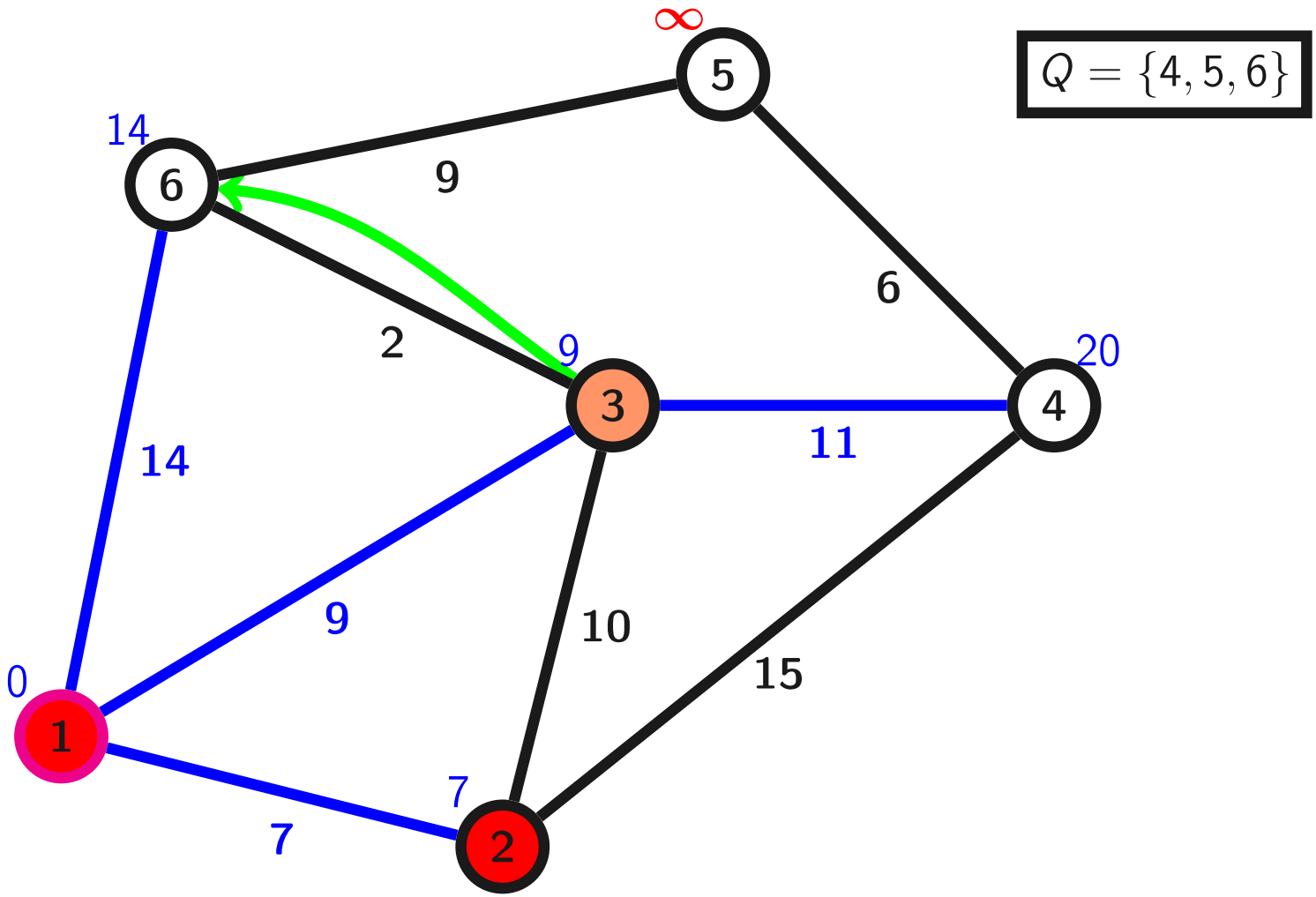
Dijkstra : exemple



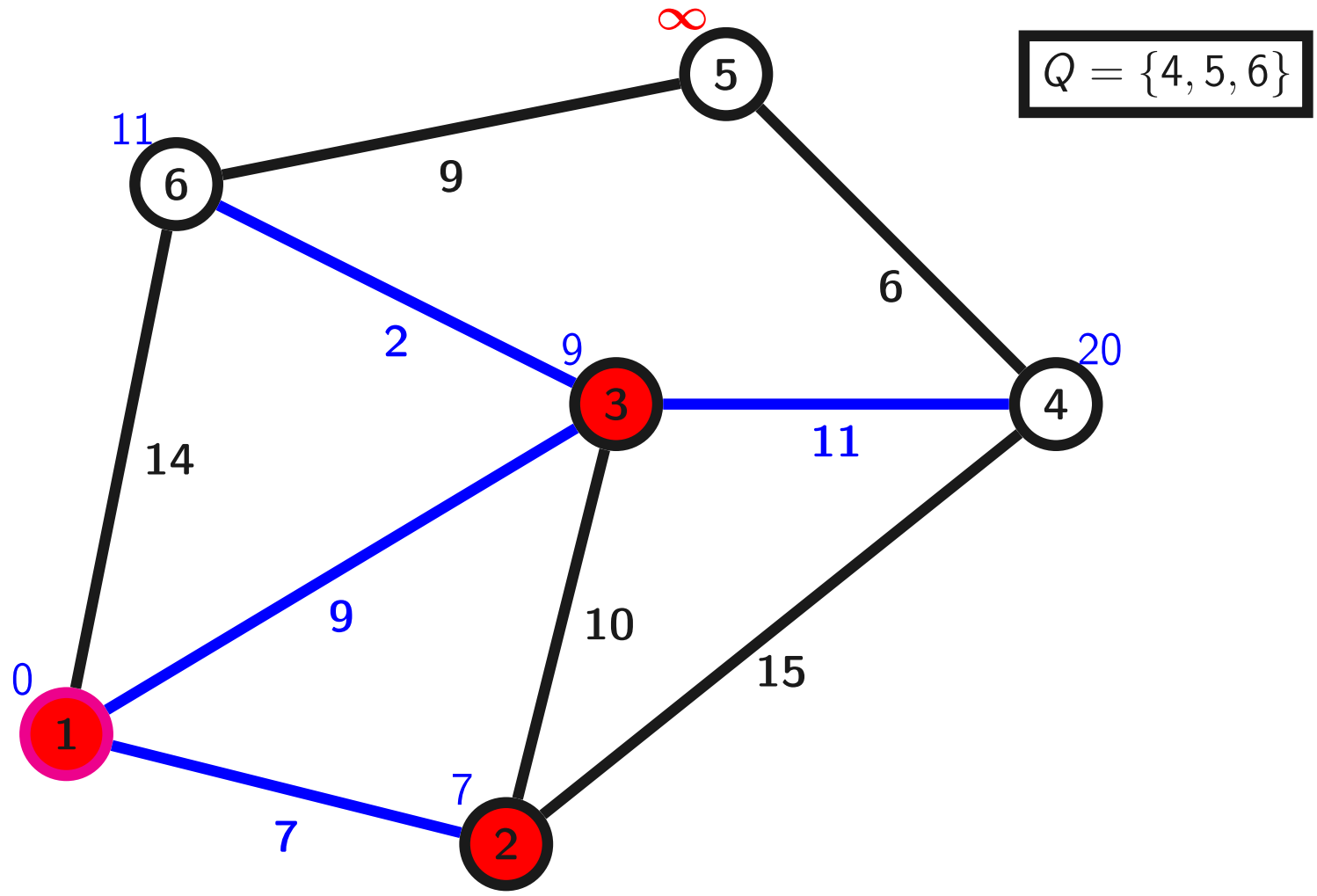
Dijkstra : exemple



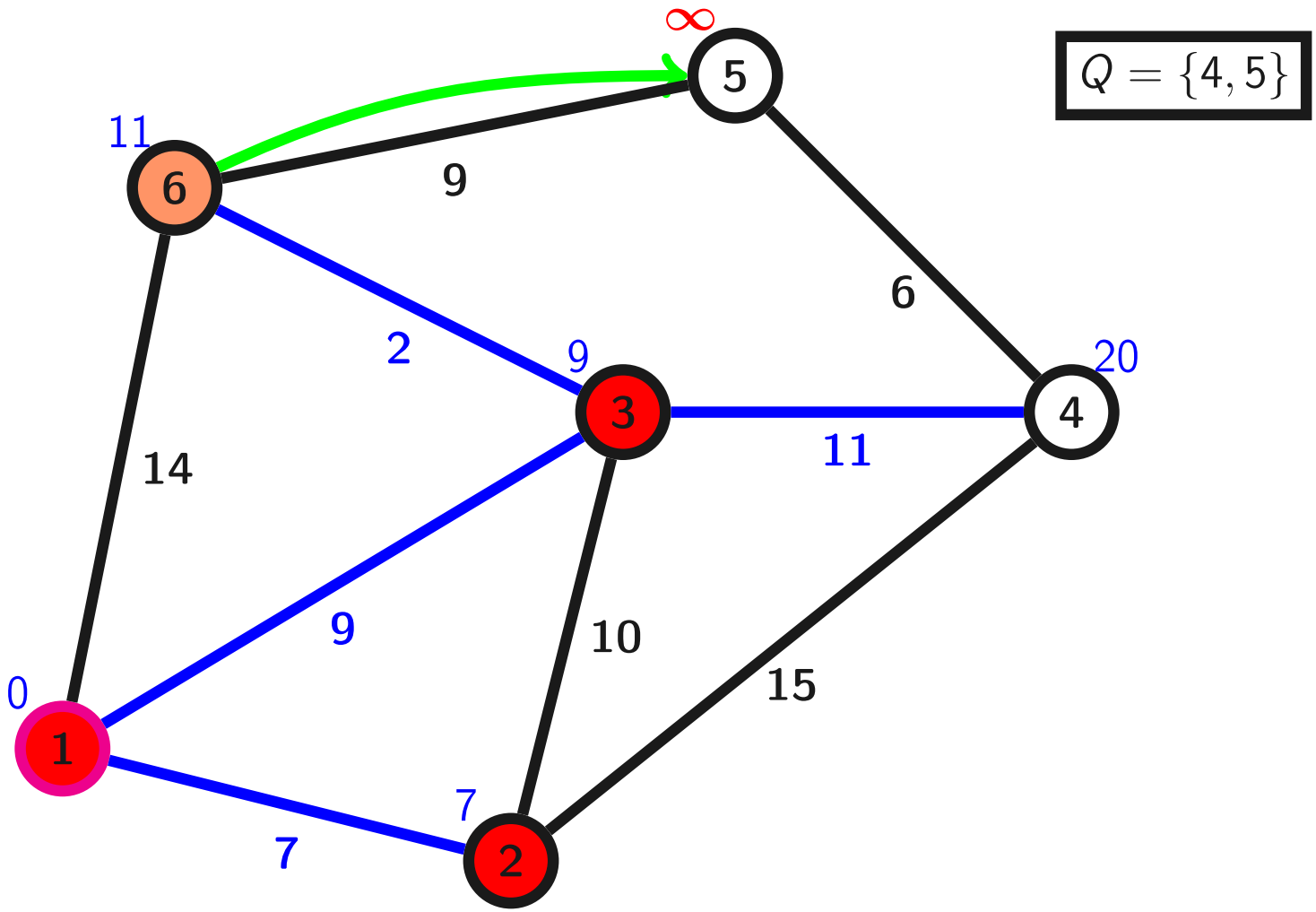
Dijkstra : exemple



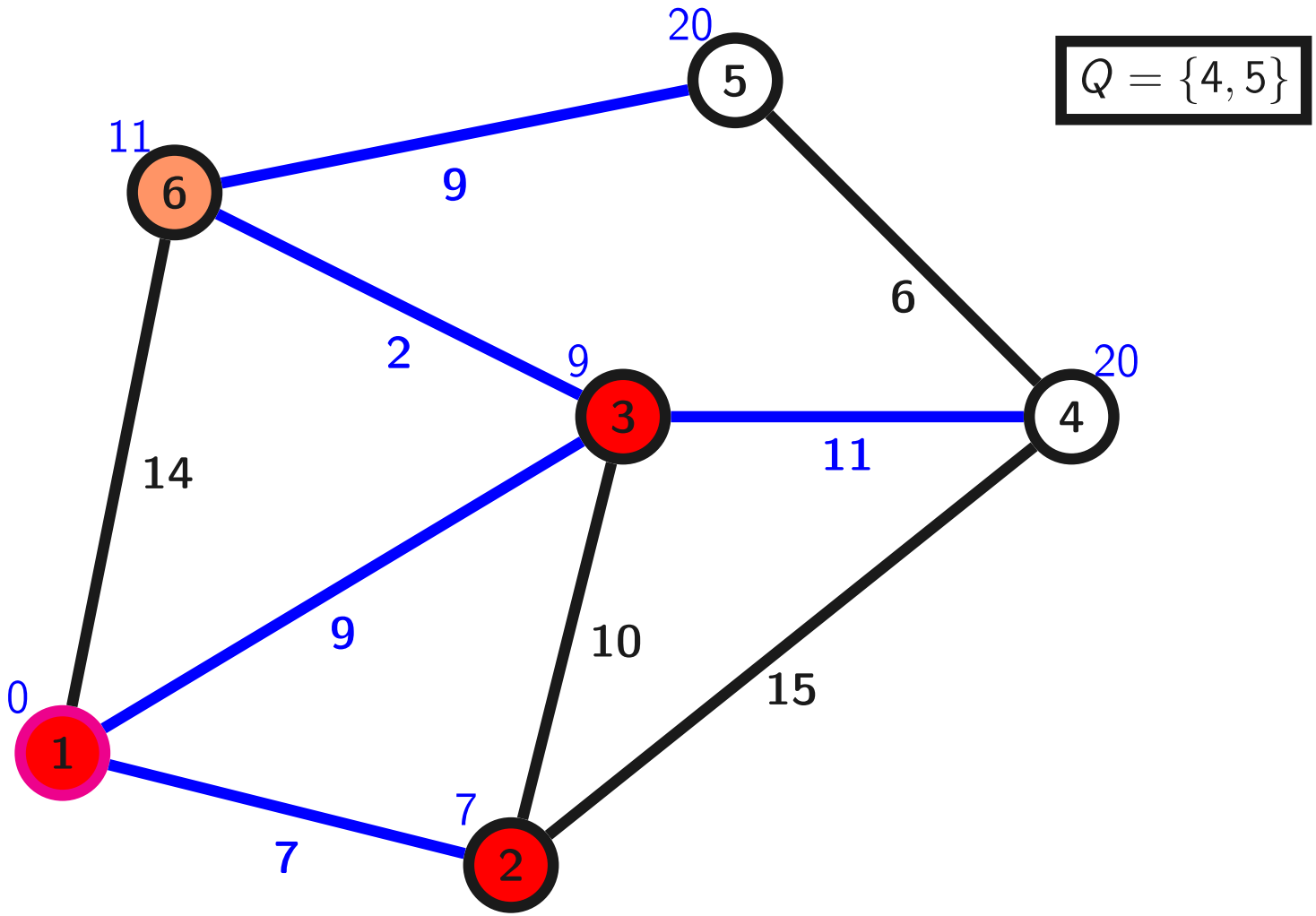
Dijkstra : exemple



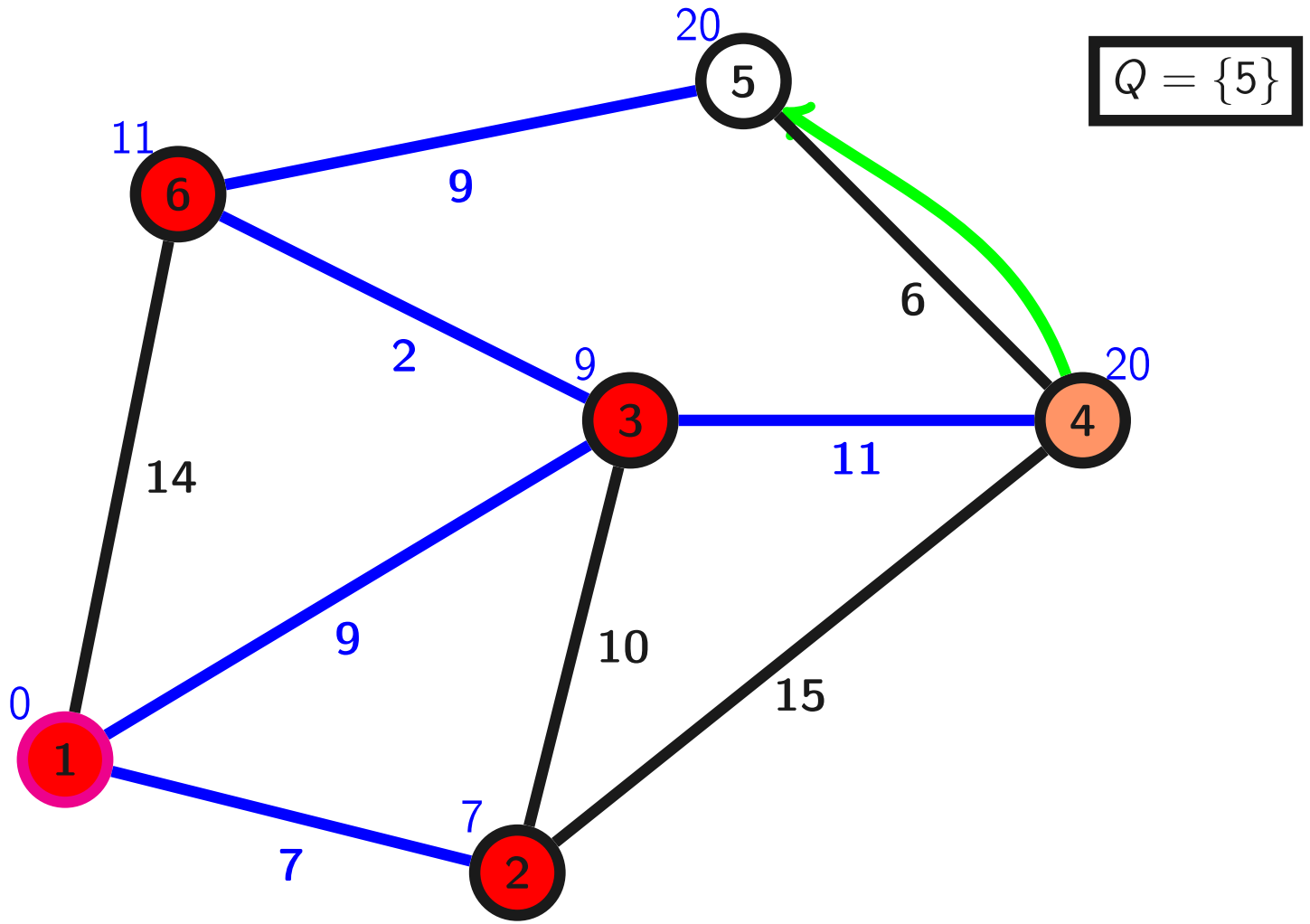
Dijkstra : exemple



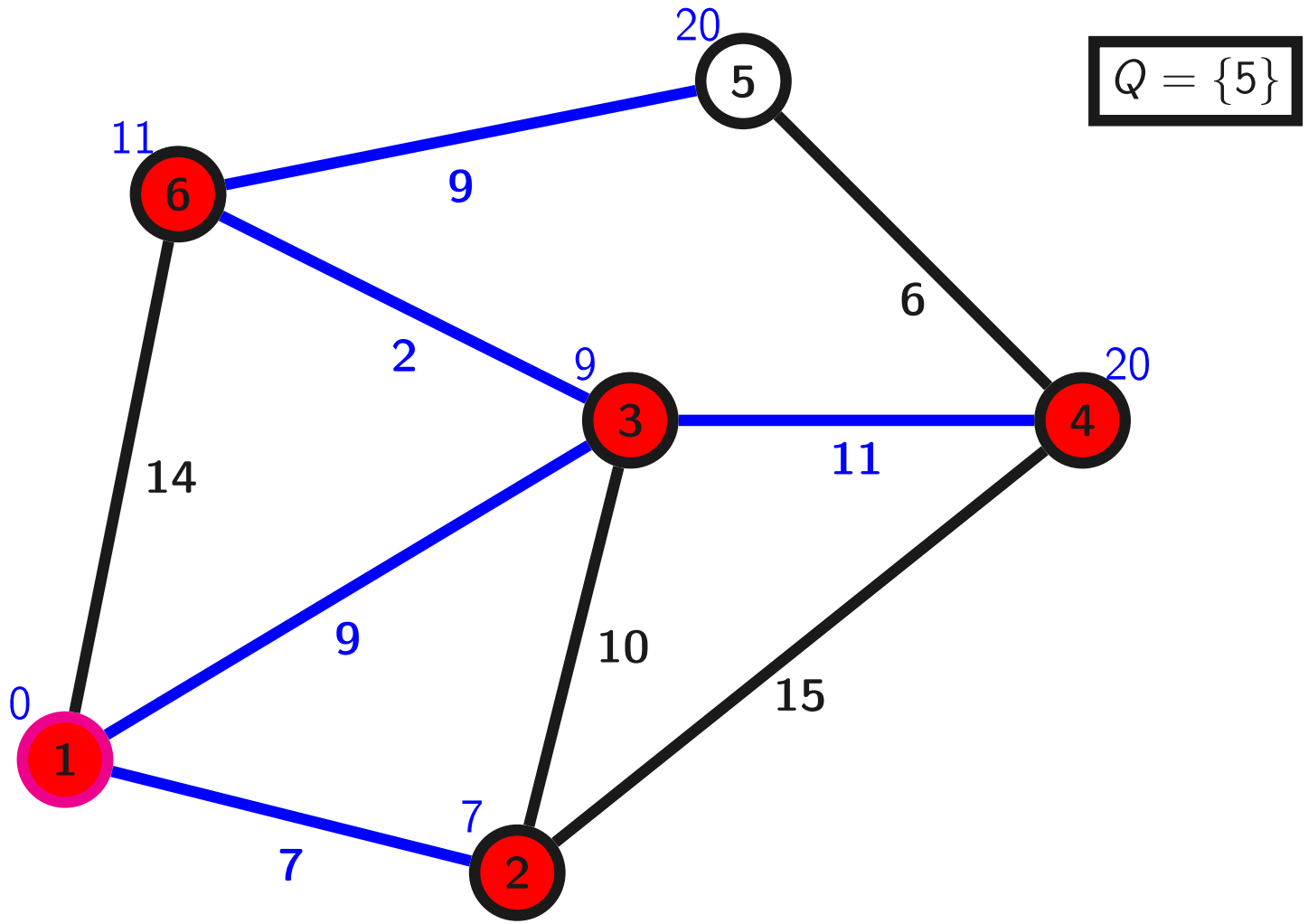
Dijkstra : exemple



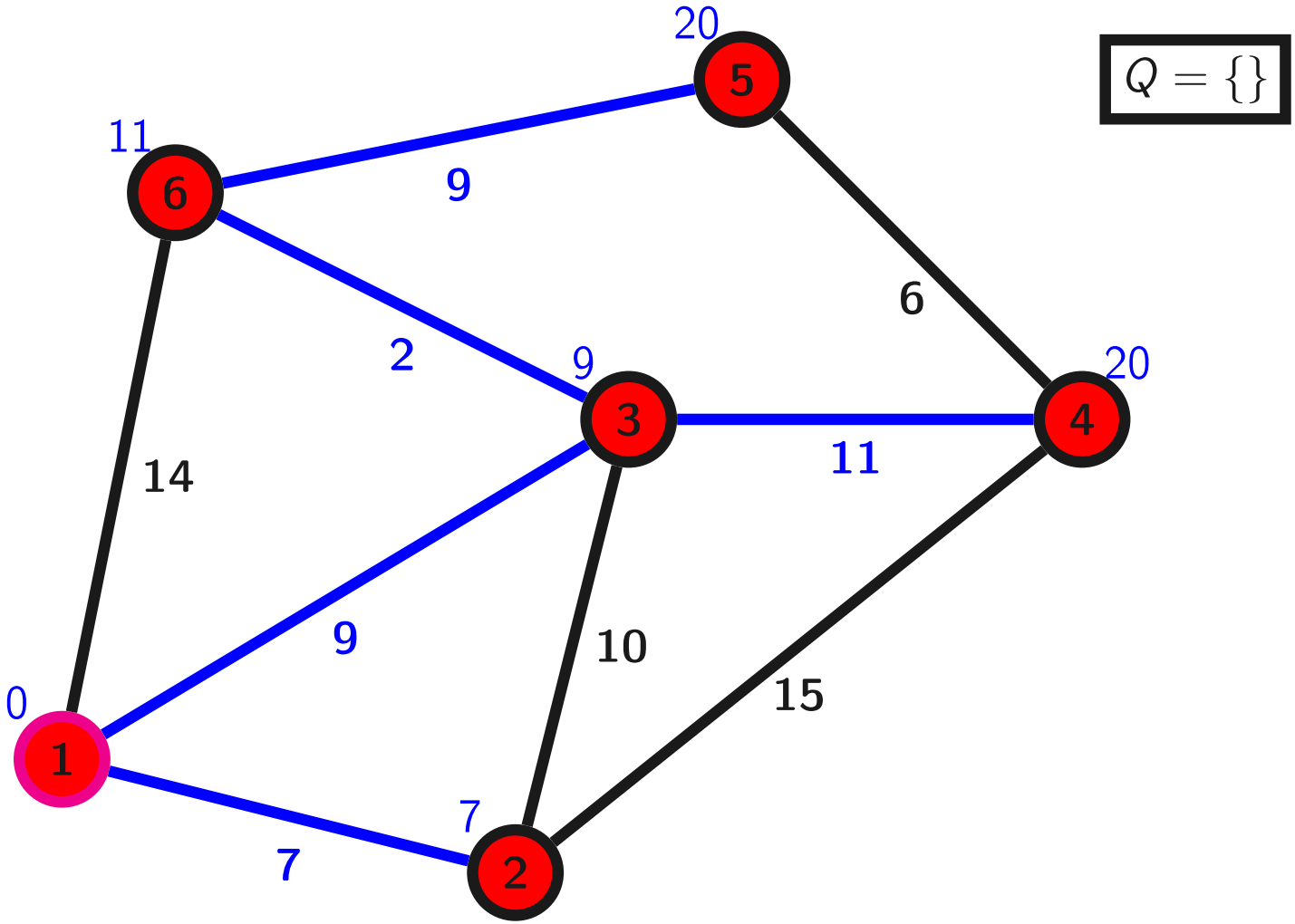
Dijkstra : exemple



Dijkstra : exemple



Dijkstra : exemple



OSPF

Open Shortest Path First

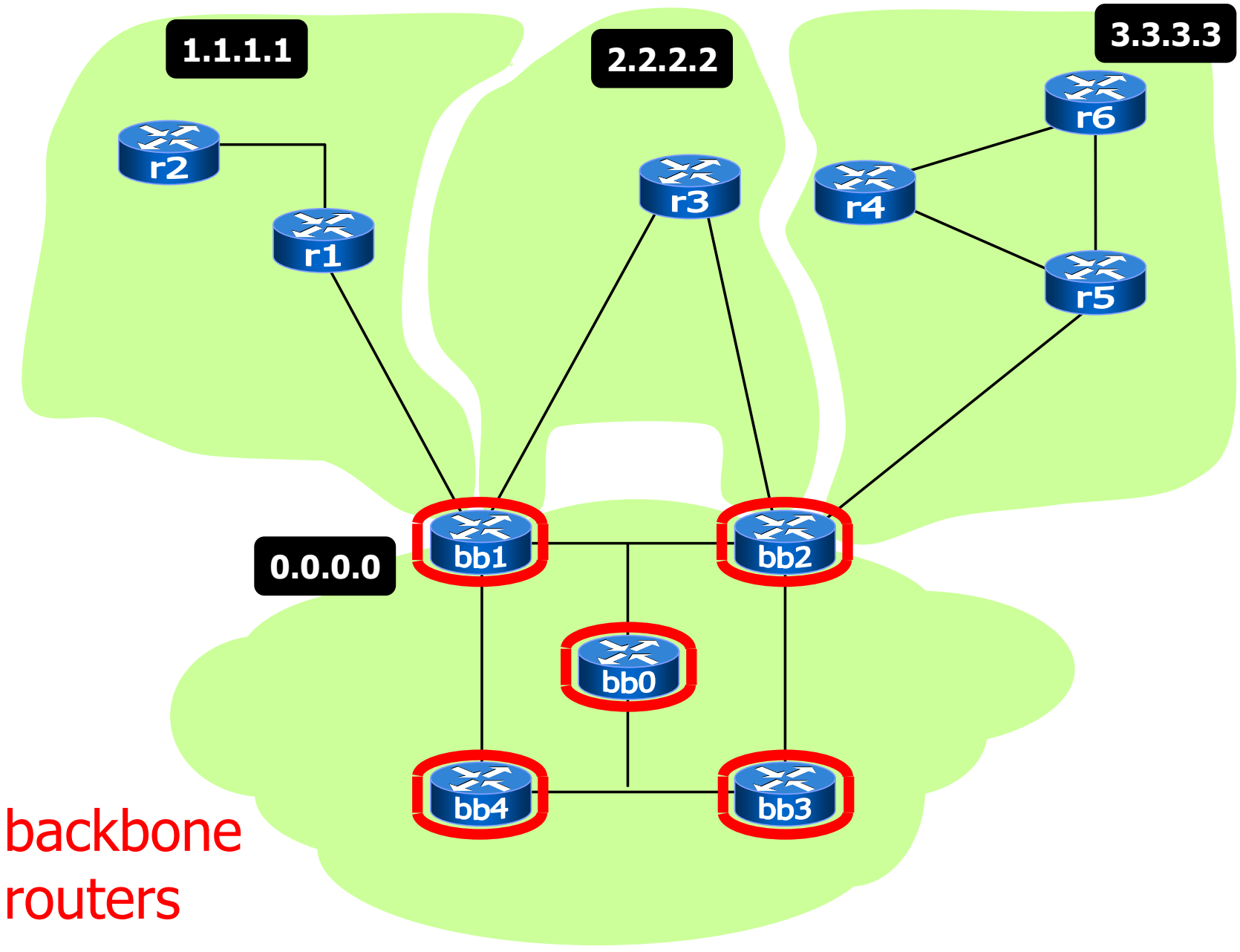
Protocole de type États de liens (RFC 2328 ou RFC 5340 pour la version 3 avec IPv6).

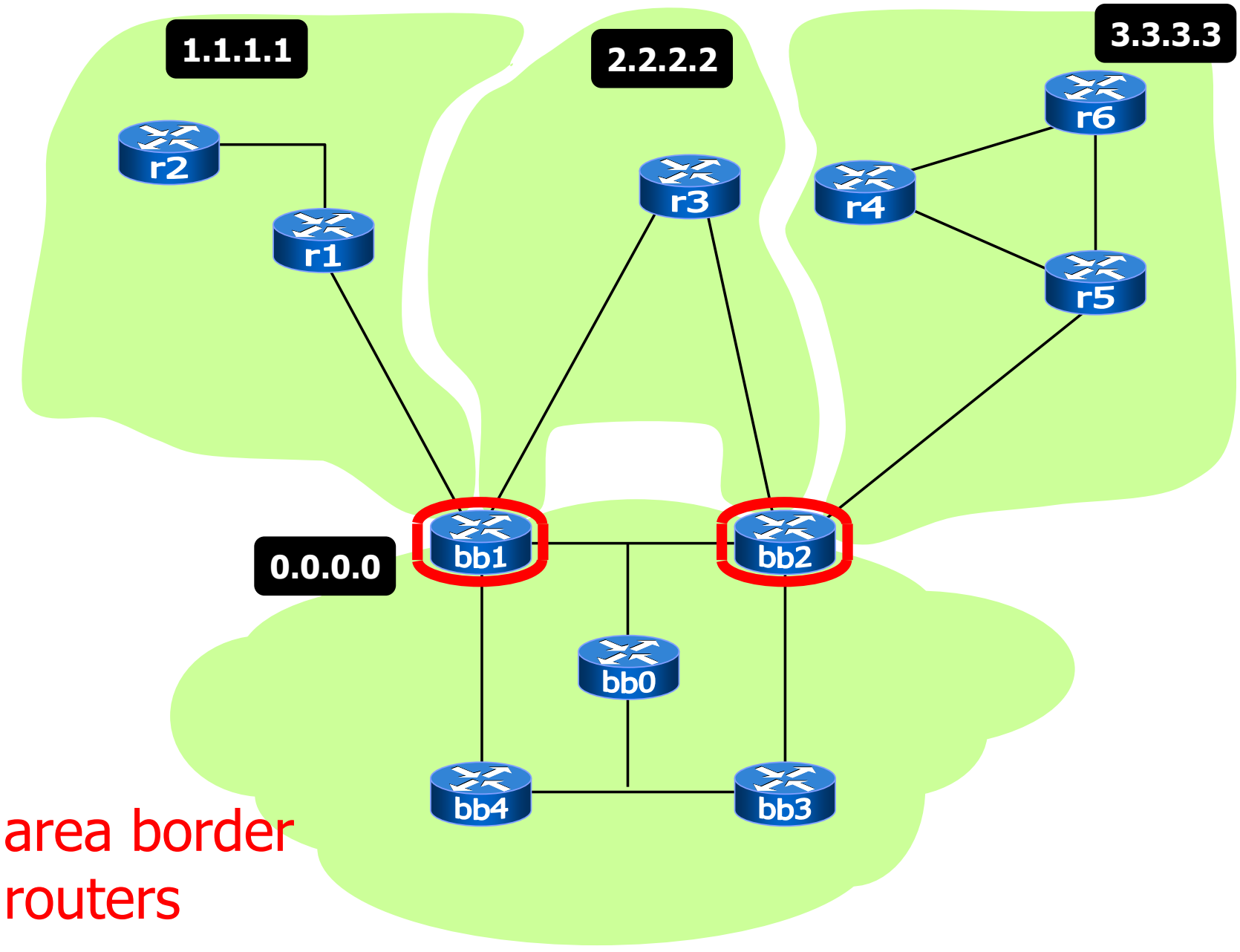
- directement dans IP, protocole 89.
- découpage en zones.
- plusieurs types de messages.
- métrique $10^8 / \textit{bande passante}$.

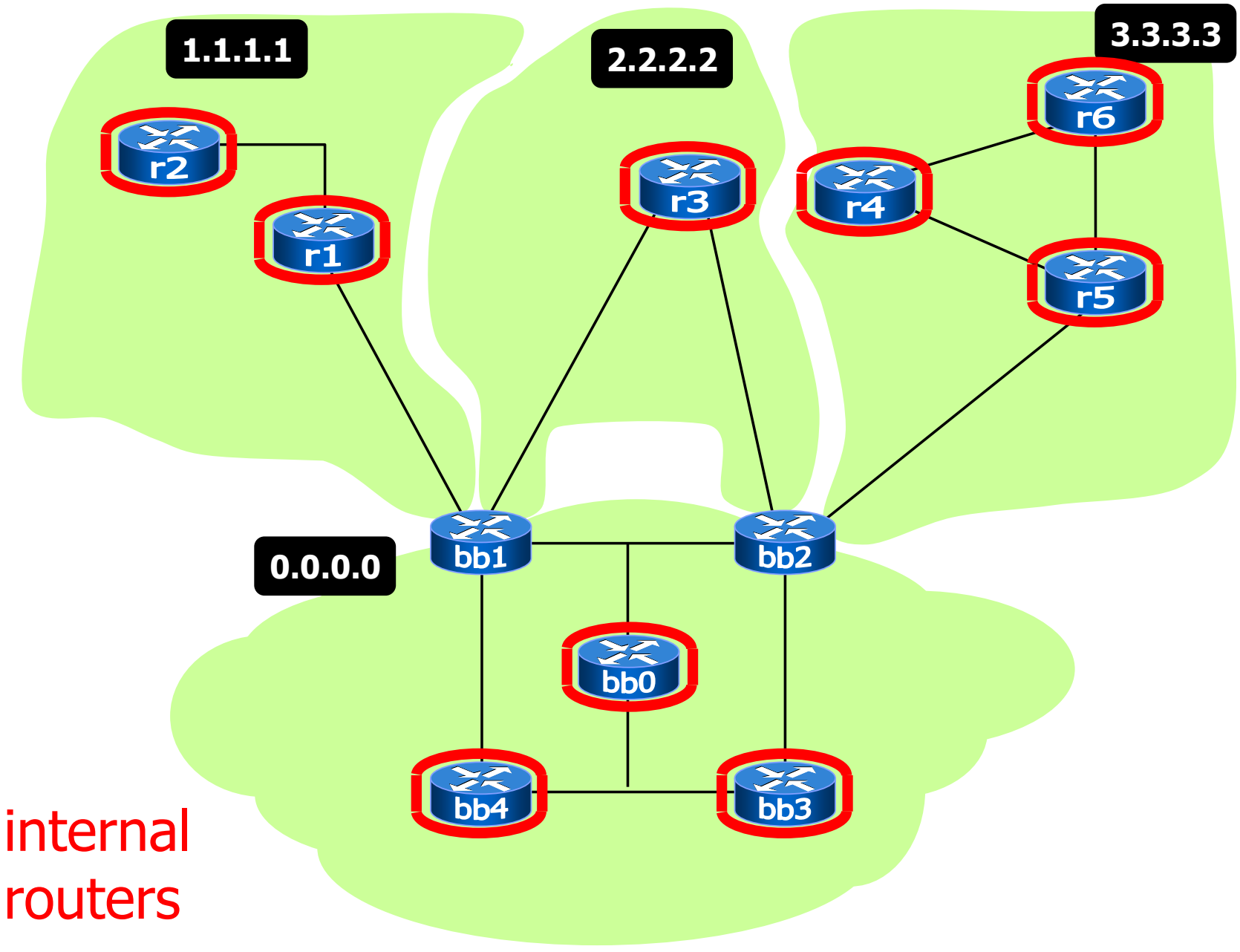
Zones OSPF (version simple)

Pour gérer des réseaux composés d'un grand nombre de routeurs, on découpe l'ensemble des routeurs en zones numérotées sans chevauchement :

- `épine dorsale` (backbone) : zone 0.
- autres zones toutes connectées à l'épine dorsale.
- échanges simplifiés aux frontières entre zones.
- routage à états de liens interne à chaque zone.
- uniquement une route par défaut vers l'extérieur dans les zones non-backbone.


1.1.1.1
2.2.2.2
3.3.3.3
r2
r1
r3
r4
r6
r5
0.0.0.0
bb1
bb2
bb0
bb4
bb3
backbone
routers


1.1.1.1
2.2.2.2
3.3.3.3
r2
r1
r3
r4
r6
r5
bb1
bb2
0.0.0.0
bb0
bb4
bb3
area border routers


1.1.1.1
2.2.2.2
3.3.3.3
r2
r1
r3
r4
r6
r5
bb1
bb2
0.0.0.0
bb0
bb4
bb3
internal
routers

Message HELLO

- Permet de découvrir les routeurs voisins sur les liens locaux.
- Envoyés régulièrement (période Hello, e.g. toutes les 10 secondes).
- Voisin déclaré mort après un temps long sans recevoir de message Hello (période Dead, e.g. 40 secondes).

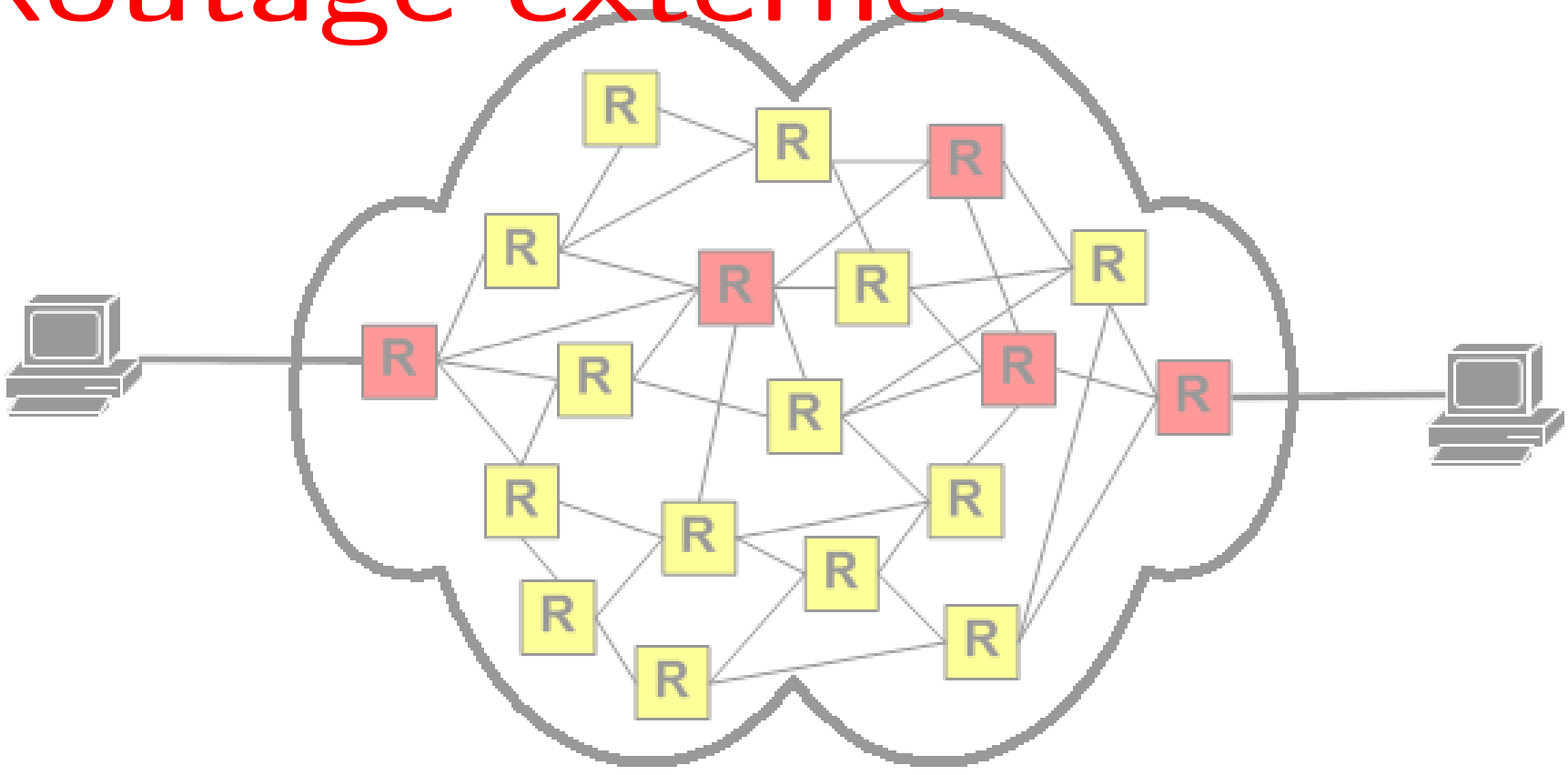
Messages

- `DataBase Description` : envoi de sa topologie sur demande pour initialisation.
- `Link State Request` : demande de renseignements complémentaires sur un ou plusieurs liens.
- `Link State Update` : envoi périodique de LSP.
- `Link State Acknowledgement` : à envoyer pour chaque LSP.

Link State Update

- Contient tous les `Link State Advertisement`(=LSP) reçus.
- Un accusé de réception par LSA.
- Différents types de LSA pour décrire les autres routeurs ou les réseaux connectés par exemple.
- Utilisation d'un checksum pour garantir la fiabilité.
- Durée de validité pour éviter que des LSA trop vieux soient considérés.

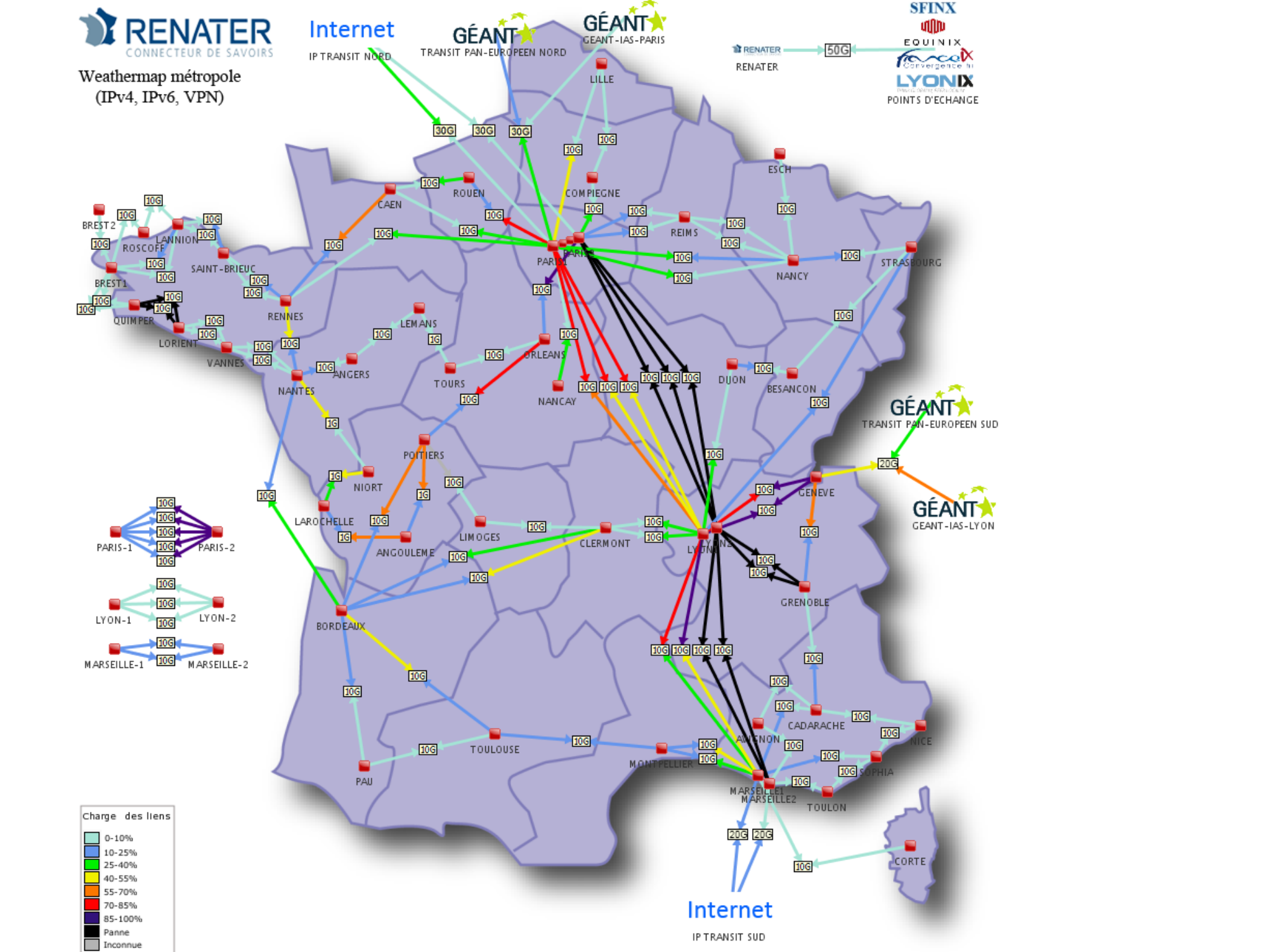
Routage externe

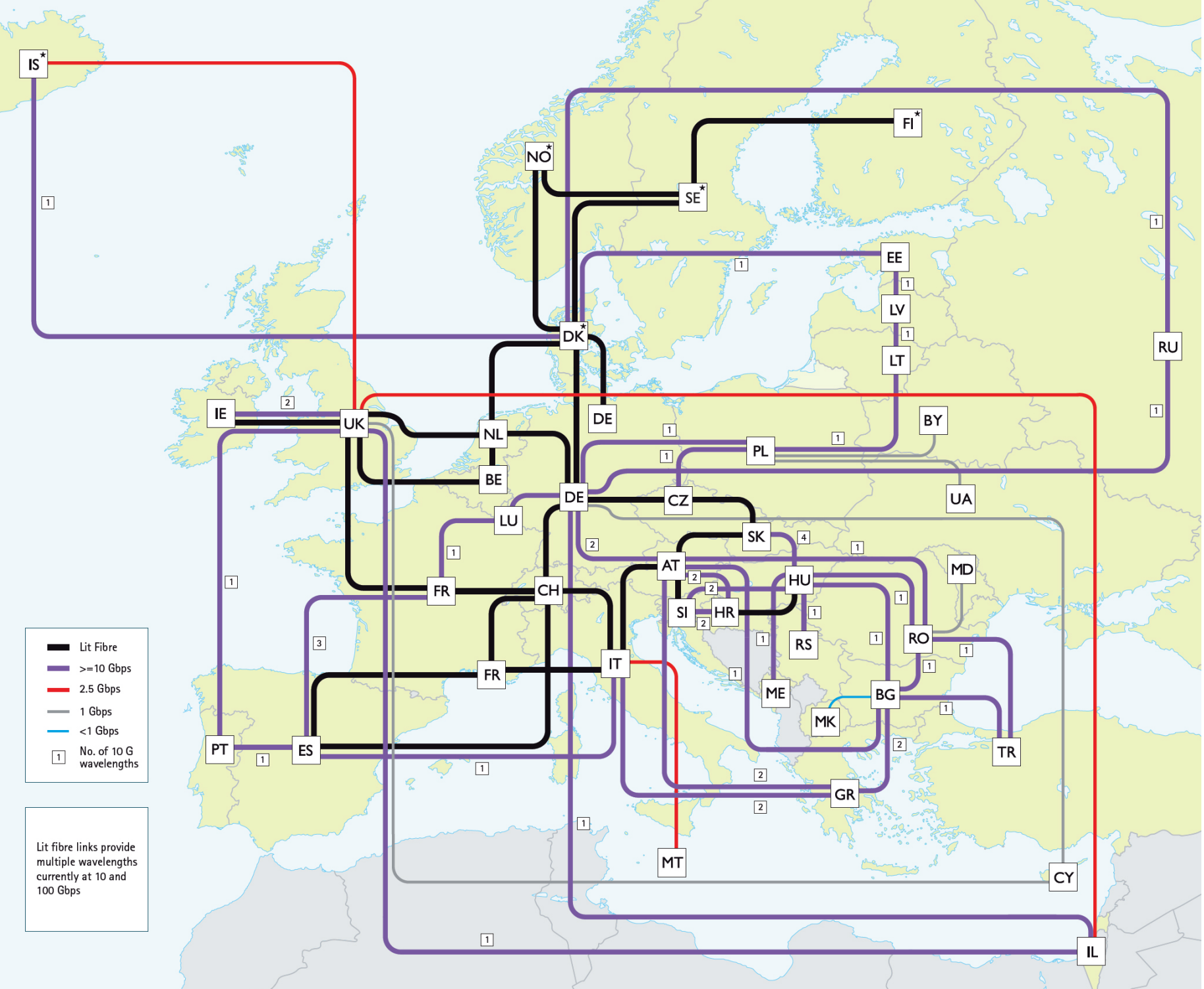


Systèmes autonomes

L'internet est divisé en systèmes autonomes (AS) :

- contrôlés par un organisme (typiquement F.A.I.) ;
- identifiés par un numéro (ASN) unique fourni par l'IANA (délégation à un RIR) ;
- politique de routage interne ;
- autour de 85000 AS à l'heure actuelle ;
- ASN codé sur 16 bits, depuis 2007 sur 32 bits.


RENATER
CONNECTEUR DE SAVOIRS
Weathermap métropole
(IPv4, IPv6, VPN)
Internet
IP TRANSIT NORD
GÉANT
TRANSIT PAN-EUROPEEN NORD
GÉANT
GEANT-IAS-PARIS
RENATER
50G
RENATER
SFINX
EQUINIX
LYONIX
POINTS D'ECHANGE
LILLE
ESCH
ROUEN
COMPIEGNE
CAEN
REIMS
PARIS
NANCY
STRASBOURG
BREST2
LANNION
ROSCOFF
SAINT-BRIEUC
BREST1
QUIMPER
LORIENT
VANNES
RENNES
LEMANS
ANGERS
NANTES
TOURS
ORLEANS
NANCAY
DIJON
BESANCON
GÉANT
TRANSIT PAN-EUROPEEN SUD
POITIERS
NIORT
LAROCHELLE
ANGOULEME
LIMOGES
CLERMONT
LYON
GENEVE
GÉANT
GEANT-IAS-LYON
GRENOBLE
PARIS-1
PARIS-2
LYON-1
LYON-2
MARSEILLE-1
MARSEILLE-2
BORDEAUX
CADARACHE
AVIGNON
NICE
SOPHIA
TOULOUSE
MONTPELLIER
PAU
MARSEILLE1
MARSEILLE2
TOULON
CORTE
Internet
IP TRANSIT SUD
Charge des liens
0-10%
10-25%
25-40%
40-55%
55-70%
70-85%
85-100%
Panne
Inconnue


IS*
NO*
SE*
FI*
EE
LV
LT
RU
DK*
IE
UK
NL
DE
BY
PL
BE
DE
CZ
UA
LU
SK
AT
MD
HU
FR
CH
SI
HR
RO
RS
IT
FR
ME
BG
MK
TR
PT
ES
GR
MT
CY
IL
Lit Fibre
>=10 Gbps
2.5 Gbps
1 Gbps
<1 Gbps
No. of 10 G wavelengths
Lit fibre links provide multiple wavelengths currently at 10 and 100 Gbps

Types d'AS

Deux types d'AS :

- AS de `transit` : autorise la traversée par des paquets venant d'autres AS.
- AS `stub` (resp. `multihomed`) : n'autorise pas la traversée et connecté à un (resp. plusieurs) AS de transit.

Protocoles de routage

Deux types de protocoles :

- IGP (Interior Gateway Protocol), e.g. RIP, OSPF, IS-IS (plus confidentiel, LS)...
- EGP (Exterior Gateway protocol) : Border Gateway Protocol.

BGP

- Moins de partage d'informations qu'à l'intérieur d'un AS.
- Enjeux politiques, de sécurité... En particulier, pas de métrique sur le coût interne à un AS.
- Protocole entre les routeurs aux frontières des AS.
- RFC 4271, TCP port 179.
- Table BGP de plus de 500000 entrées pour IPv4 (limite de certains vieux routeurs atteinte).

Quel routage entre AS ?

Les lieux d'échange

Les échanges se font entre routeurs aux frontières des AS :

- soit sur un lien privé entre les 2 routeurs ;
- soit un point d'interconnexion public auquel sont connectés des routeurs de différents AS.

Quel routage entre AS ?

Des relations diverses

Les AS ne traitent pas nécessairement d'égal à égal :

- relations Client-Fournisseur (`transit`), le Client loue le passage par l'AS Fournisseur ;
- relations entre Pairs (`peering`), les traffics sont jugés globalement équivalents et les AS font transiter l'ensemble du traffic des autres AS.

Quel routage entre AS ?

Organisation d'internet

Une hiérarchie historique :

- `Tier-1` : groupe de grands FAI en peering. Pas de transit. E.g. Verizon, Deutsche Telekom, Orange.
- `Tier-2` : FAI nationaux ou régionaux. Peering entre eux. Relation de transit avec des FAI du Tier-1. E.g. Renater, Turk Telecom International.
- `Tier-3` : FAI locaux ou fournisseurs de contenus. Relation de transit avec des FAI du Tier-2. E.g. Université d'Orléans

Récemment, quelques AS (`Hyper Giants` : Google, Amazon, Facebook...) concentrent une grande partie du traffic et ont des accords de peering avec la plupart des autres AS.

Quel routage entre AS ?

Nécessité d'implémenter la politique de routage (transit/peering) et de choisir des routes sans connaître les métriques intra-AS.

- Échanges de routes entre AS.
- Filtres en entrée et en sortie.
- Algorithme de choix de la meilleure route basé sur un système de préférences.

Vecteurs de chemins

Path Vector Protocol

- Chaque routeur communique sa meilleure route vers chaque destination.
- Métrique = nombre d'AS traversés.
- Mise à jour uniquement en cas de changement de la route.

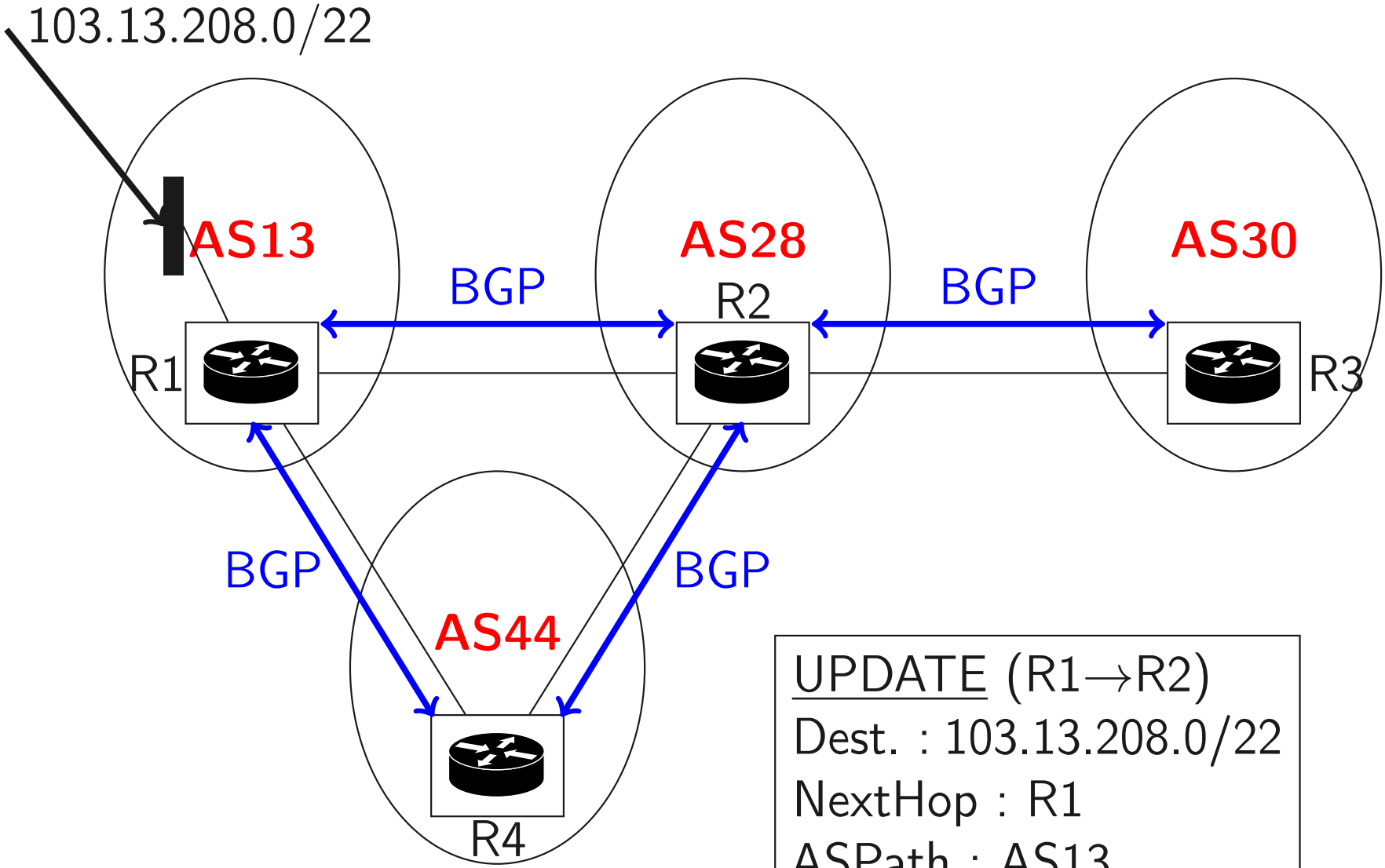
Deux types de messages :

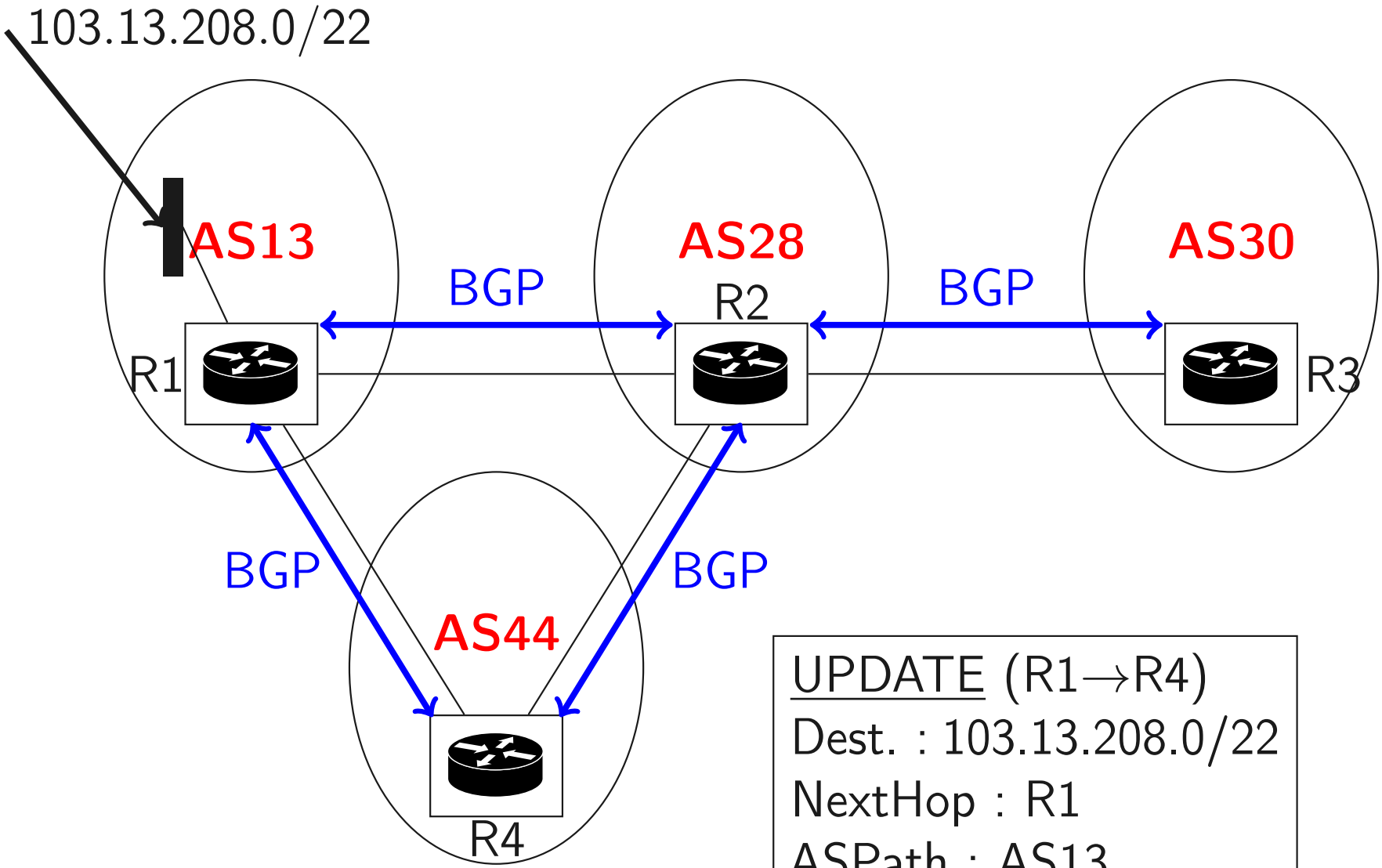
- `UPDATE` : description d'une route, destination, prochain saut et liste des AS traversés.
- `WITHDRAW` : retrait d'une route précédemment annoncée, destination et liste des AS.

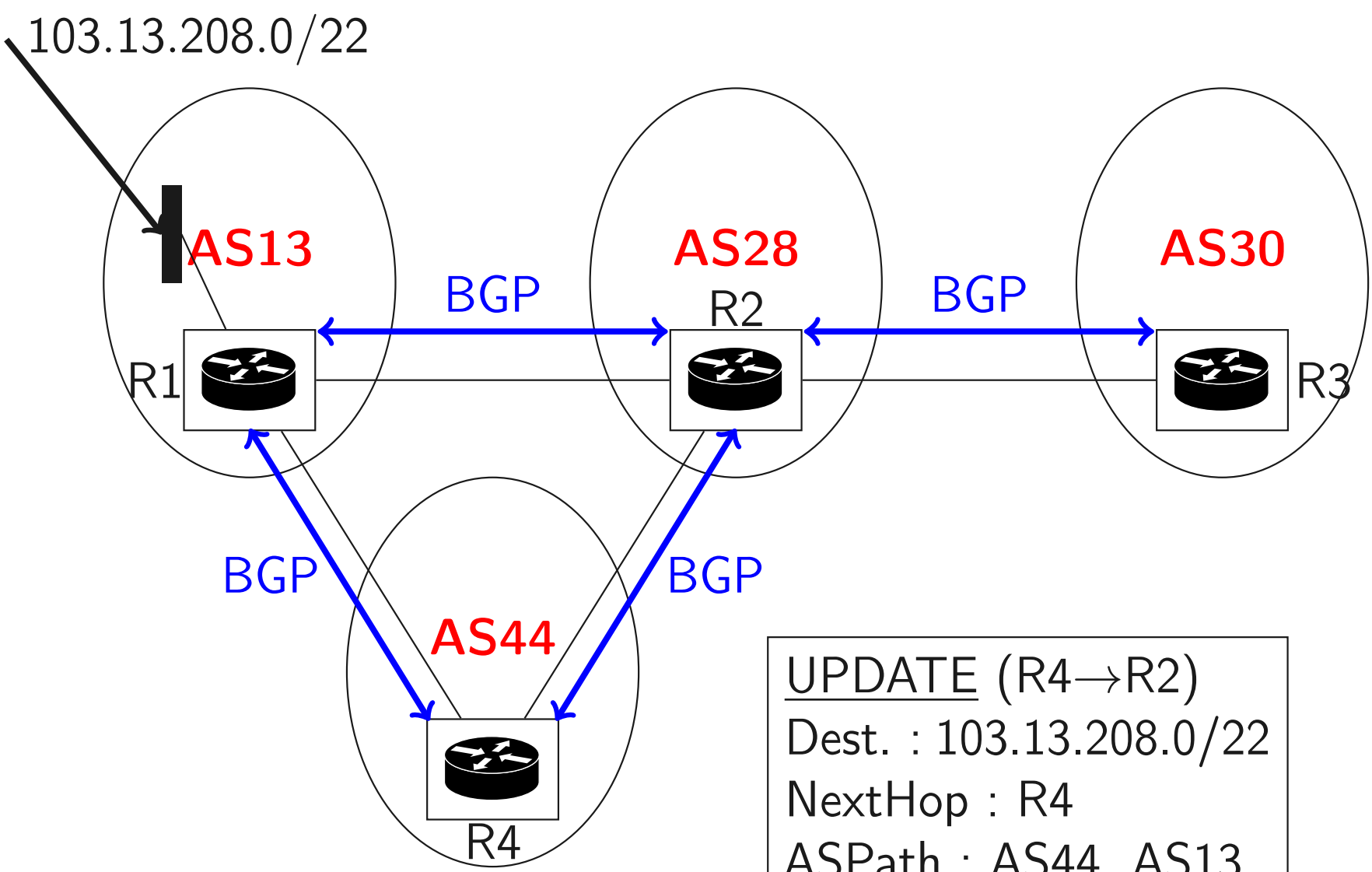
Sessions BGP

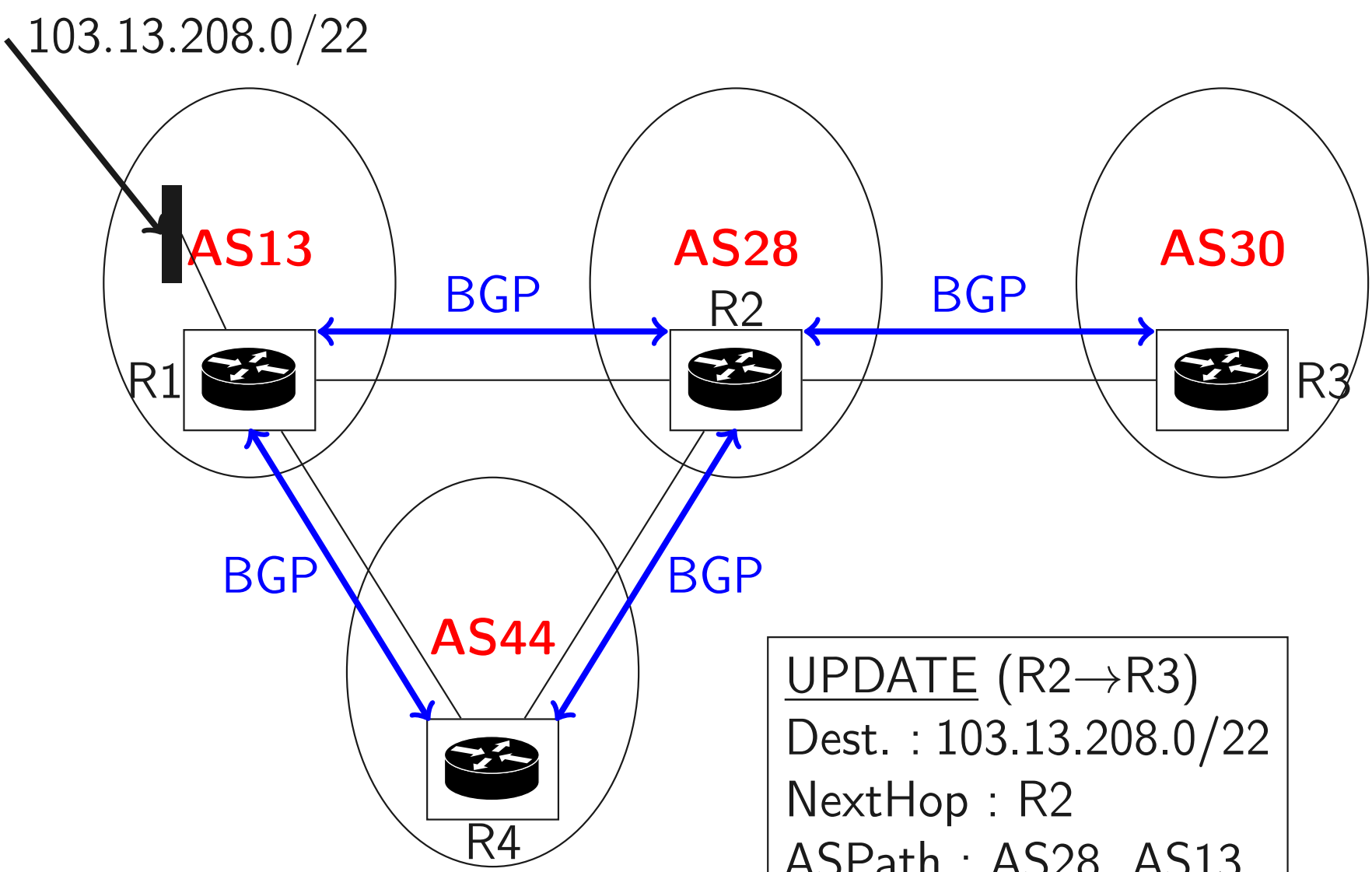
Les routeurs BGP communiquent 2 par 2 :

- Établissement de la session BGP (dans TCP).
- Échange de l'ensemble des routes connues.
- Durant la session : mise à jour ou retrait en cas de changement de topologie détecté.


103.13.208.0/22
AS13
AS28
AS30
BGP
BGP
R2
R1
R3
BGP
BGP
AS44
R4
UPDATE (R1→R2)
Dest. : 103.13.208.0/22
NextHop : R1
ASPath : AS13


103.13.208.0/22
AS13
AS28
AS30
BGP
BGP
R2
R1
R3
BGP
BGP
AS44
R4
UPDATE (R1→R4)
Dest. : 103.13.208.0/22
NextHop : R1
ASPath : AS13


103.13.208.0/22
AS13
AS28
AS30
BGP
BGP
R2
R1
R3
BGP
BGP
AS44
R4
UPDATE (R4→R2)
Dest. : 103.13.208.0/22
NextHop : R4
ASPath : AS44, AS13


103.13.208.0/22
AS13
AS28
AS30
BGP
BGP
R2
R1
R3
BGP
BGP
AS44
R4
UPDATE (R2→R3)
Dest. : 103.13.208.0/22
NextHop : R2
ASPath : AS28, AS13

Types de routes

- Routes statiques : configurées à la main.
- Routes apprises par le protocole IGP (OSPF ou RIP).
- Routes apprises par les autres routeurs BGP.

La politique de routage s'applique essentiellement aux routes apprises par BGP.

Filtre d'entrée

Implémentation de la politique

Le filtre d'entrée :

- sélectionne pour chaque routeur voisin les routes acceptables.
- attribue à chaque telle route un champ `local-pref`, plus la préférence est grande, plus la route est intéressante.

La préférence dépend en général du coût et de la capacité.
Exemple :

- route proposée par un pair, local-pref = 100.
- route proposée par un client, local-pref = 200.
- route proposée par un fournisseur, local-pref = 50.

BGP decision process

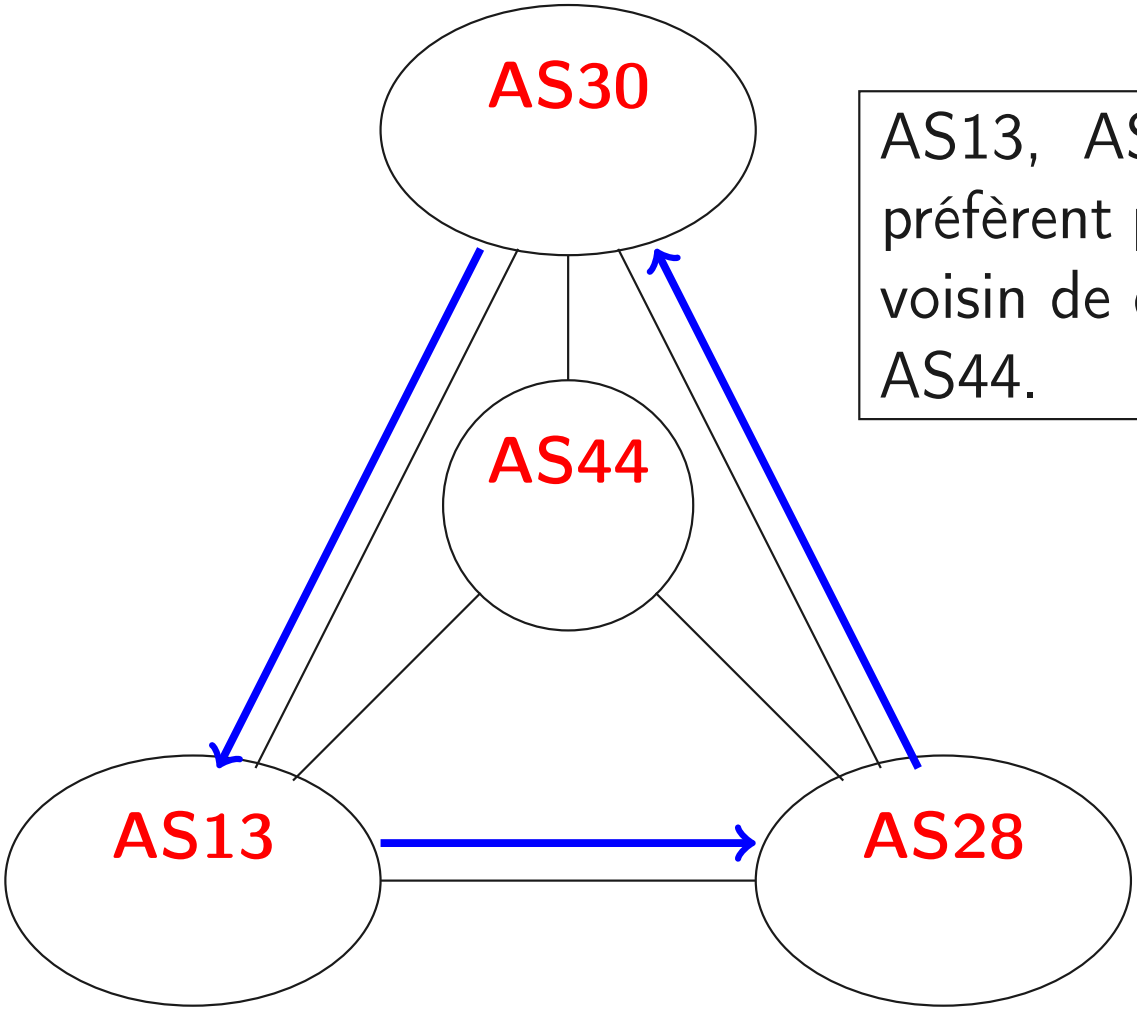
Une fois les routes et leur attribut local-pref connus, choix de la meilleure route :

1. plus grande préférence ;
2. plus petit vecteur de chemin (nombre d'AS traversés) :
3. Règle supplémentaire pour départager.

En sortie, le filtre s'applique :

- sur la route sélectionnée par le processus de décision ;
- indépendamment pour chaque pair.

Problèmes de routage



AS13, AS28 et AS30 préfèrent passer par leur voisin de droite que par AS44.

Les routes BGP dans un AS

En pratique, chaque AS a plusieurs routeurs aux frontières, qui doivent s'échanger les routes reçues de l'extérieur.
Problème : les protocoles IGP (OSPF, RIP, ...) ne sont pas conçus pour.

- Trop de routes.
- Pas d'attributs local-pref ou vector path.

Solution : 2 sous-protocoles

`eBGP`

- on communique seulement la meilleure route ;
- des filtres sont définis pour chaque session.

`iBGP`

- on communique les routes de chaque routeur d'entrée ;
- pas de filtre.

Quels routeurs pour iBGP ?

Trois options :

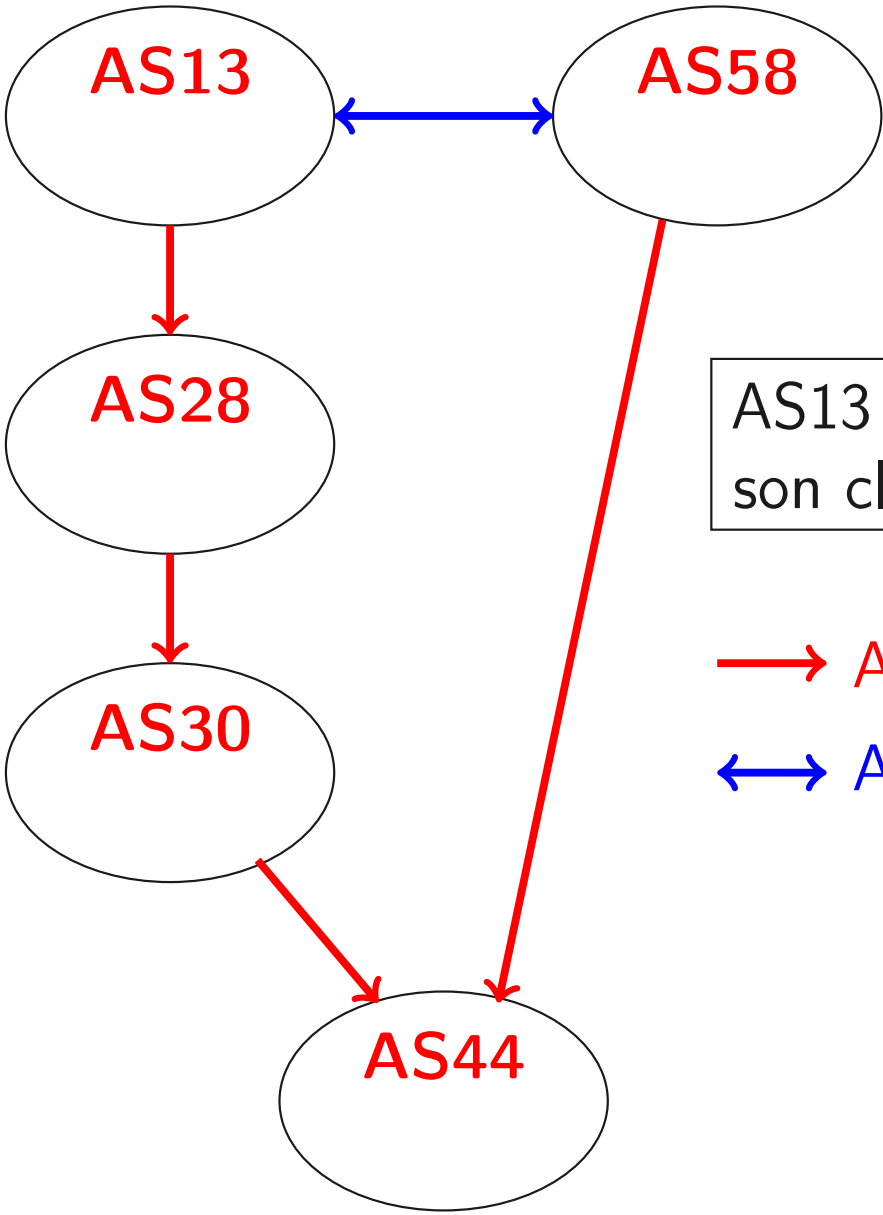
1. Utiliser IGP. Problèmes divers.
2. Créer des tunnels entre routeurs frontière pour les sessions iBGP, en utilisant IGP. MPLS
3. Faire participer tous les routeurs de l'épine dorsale de l'AS à BGP. Les routeurs internes apprennent les routes. En revanche, un routeur ne communique jamais une route apprise sur une session iBGP. Inutile, tous les routeurs de l'épine dorsale sont connectés à tous les routeurs frontière : Graphe des sessions iBGP complet sur l'épine dorsale.

Algorithme de décision

Critères par ordre d'importance :

1. Route avec le plus grand attribut `local-pref`.
2. Route avec le plus petit vecteur chemin.
3. Plus petite route interne. (`Multi Exit Discriminator`)
4. Route apprise par eBGP plutôt qu'iBGP.
5. Plus proche premier saut.
6. Pour départager : plus petite IP.

Attribut `local-pref`

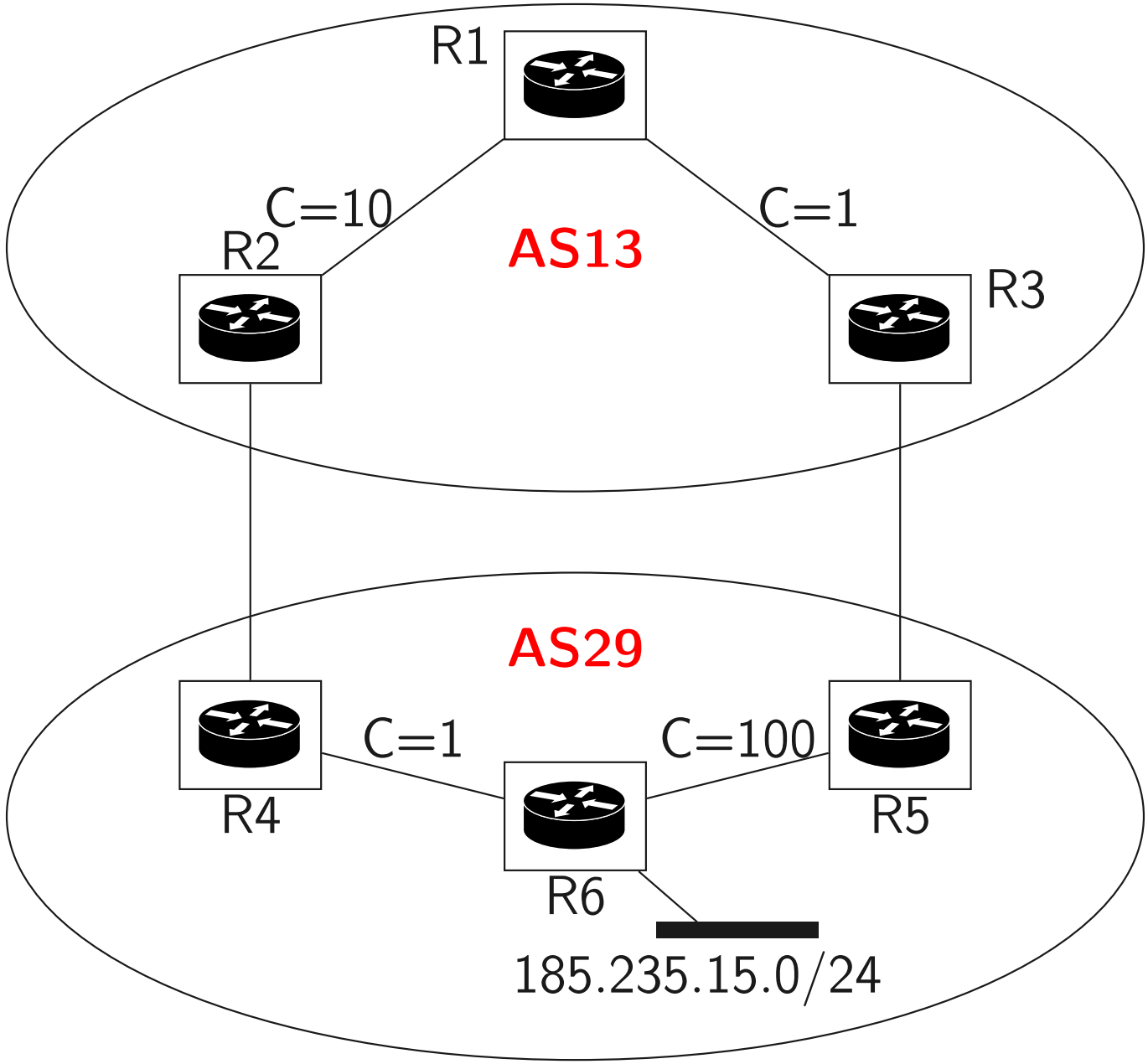


AS13 préfère passer par son client AS28.

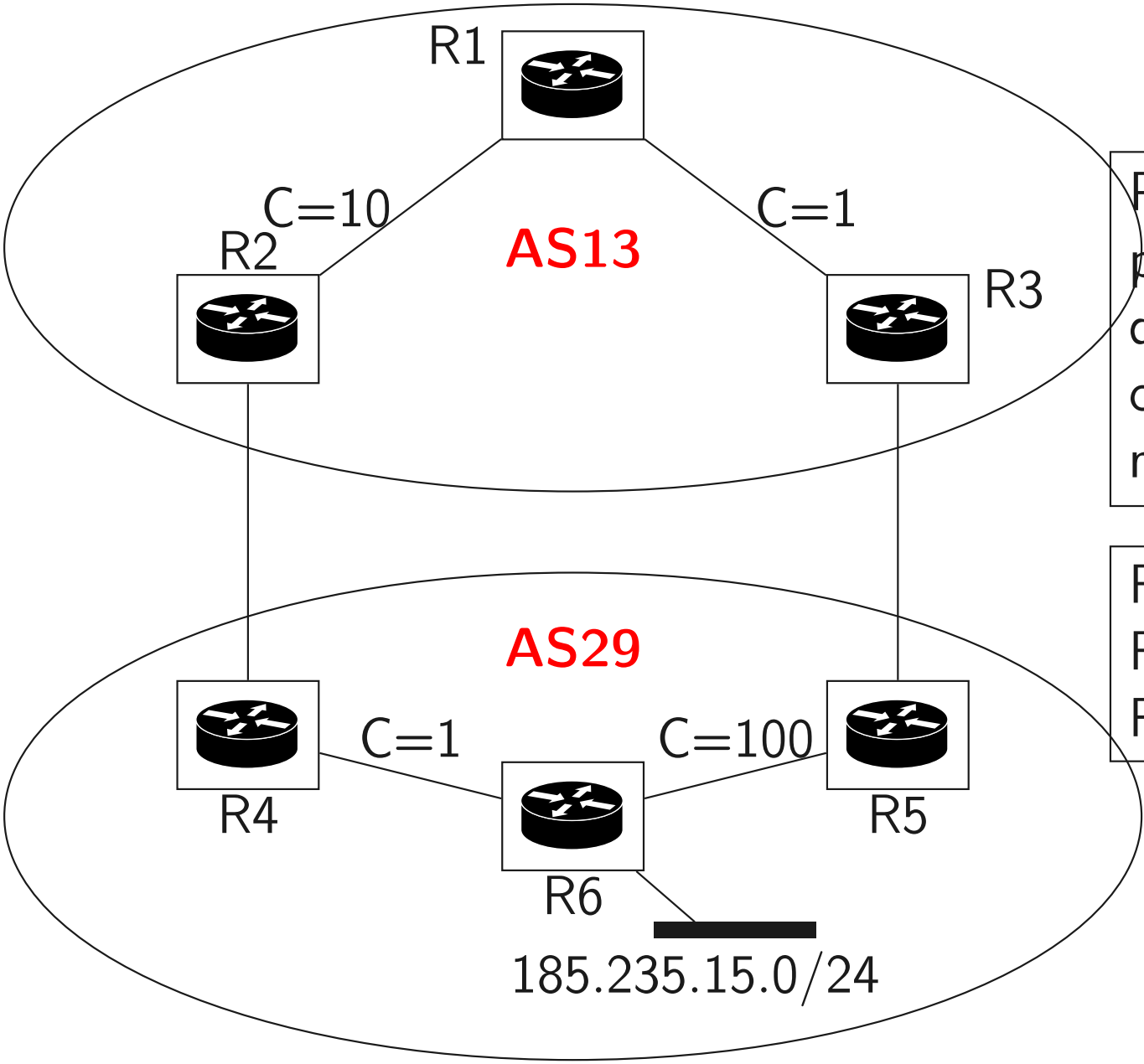
$\longrightarrow$ Accord Client $\rightarrow$ Fournisseur

$\longleftrightarrow$ Accord de peering

Route interne



Route interne



Pour le cas de plusieurs entrées dans AS29, AS29 communique la métrique interne.

R1 a 2 routes :
R2,AS29,MED=1
R3,AS29,MED=100

Droits

`http://inl.info.ucl.ac.be/CNP3`

Merci à Nicolas Ollinger.