ProTIP: sachez quoi attendre de vos périphériques PCI Express !

Marion Daubignard, Yves-Alexis Perez (LAM, ANSSI)

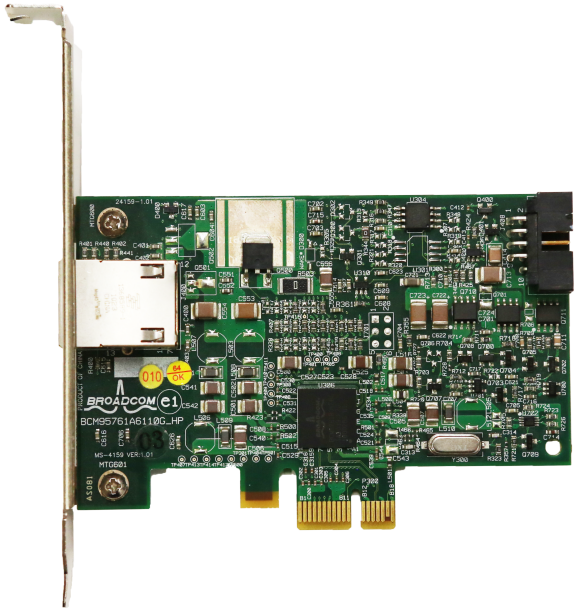
Section 1

Périphériques malveillants

Périphérique PCI Express malveillant

Ici, matériel PCI Express uniquement (donc pas d'USB) :

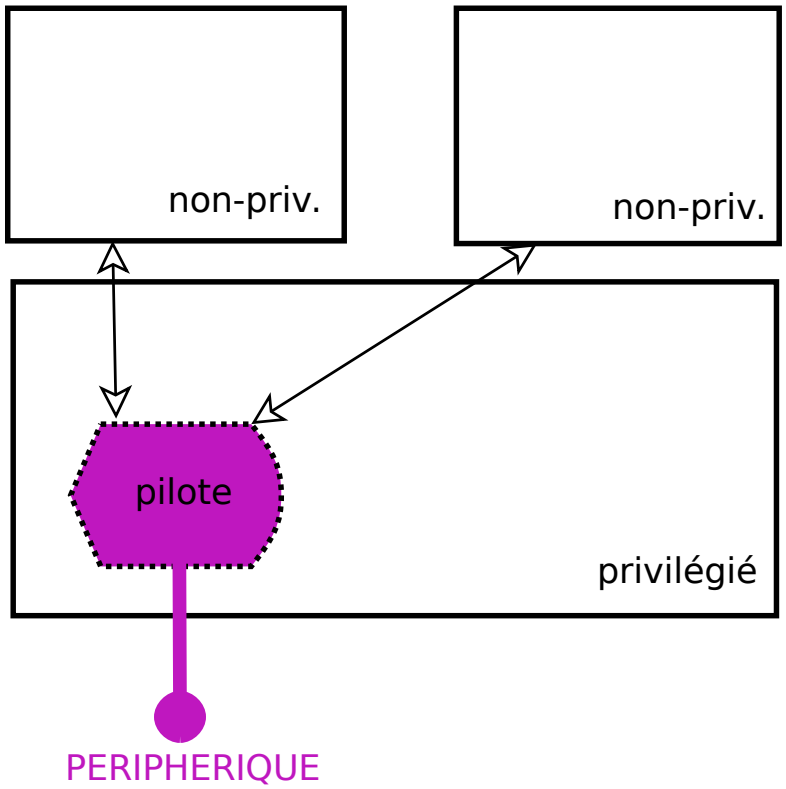
- cartes réseau ;
- cartes graphiques ;
- etc.



Pertinence de la menace liée à un périphérique malveillant

- attaquant local utilisant un périphérique déjà malveillant : piégeage matériel (interne, externe) ;
- attaquant distant utilisant un périphérique devenant malveillant : mise à jour, délégation de périphérique.

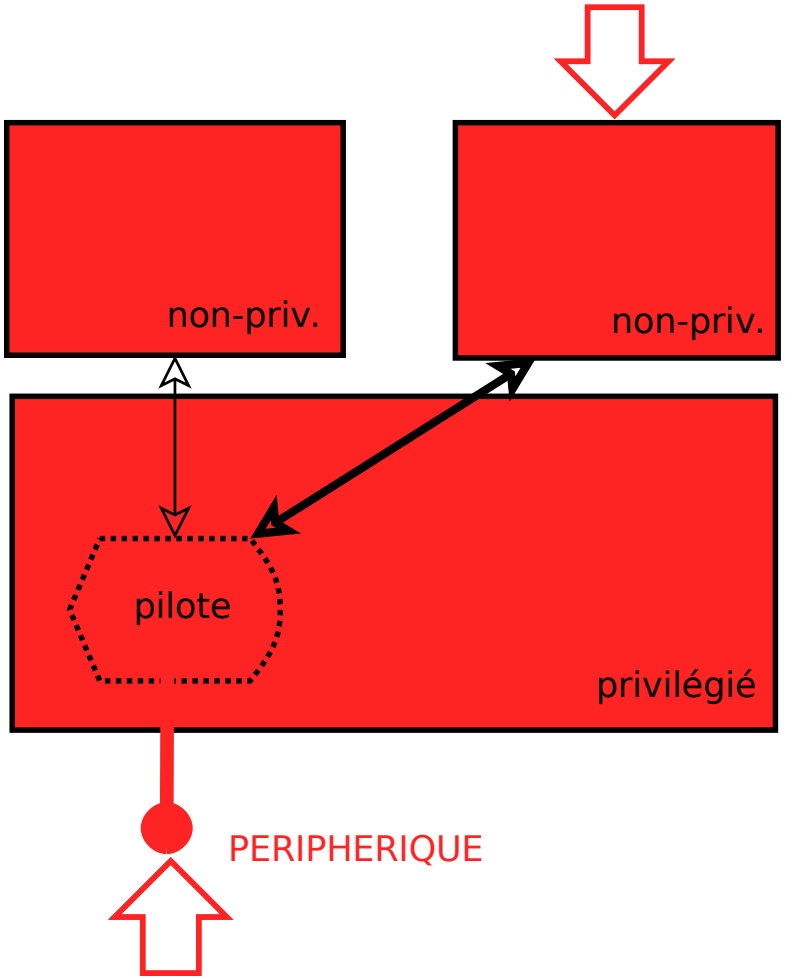
Intérêt de la délégation de périphérique (1)



Défauts de l'architecture monolithique :

- performance ;

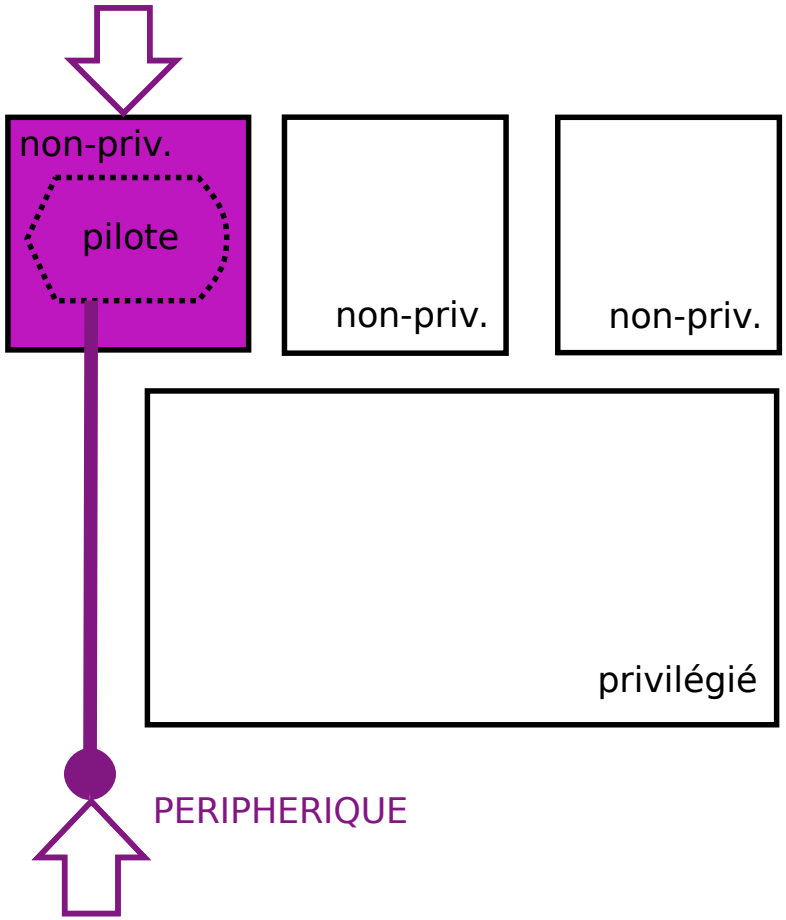
Intérêt de la délégation de périphérique (1)



Défauts de l'architecture monolithique :

- performance ;
- pas de cloisonnement.

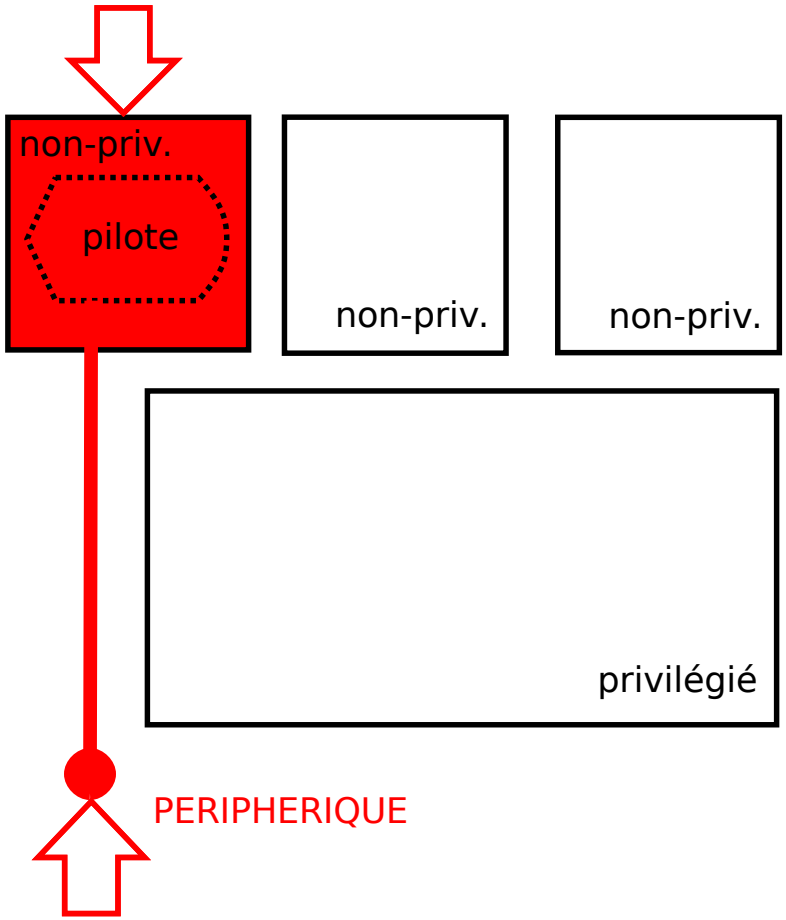
Intérêt de la délégation de périphérique (2)



Avantages de déprivilégier :

- moins de changement de contextes ;
- modèle économique : infrastructure cloud vendant du temps de calcul ;

Intérêt de la délégation de périphérique (2)



Avantages de déprivilégier :

- moins de changement de contextes ;
- modèle économique : infrastructure cloud vendant du temps de calcul ;
- confinement de compromission ?

Comprendre précisément la sécurité apportée

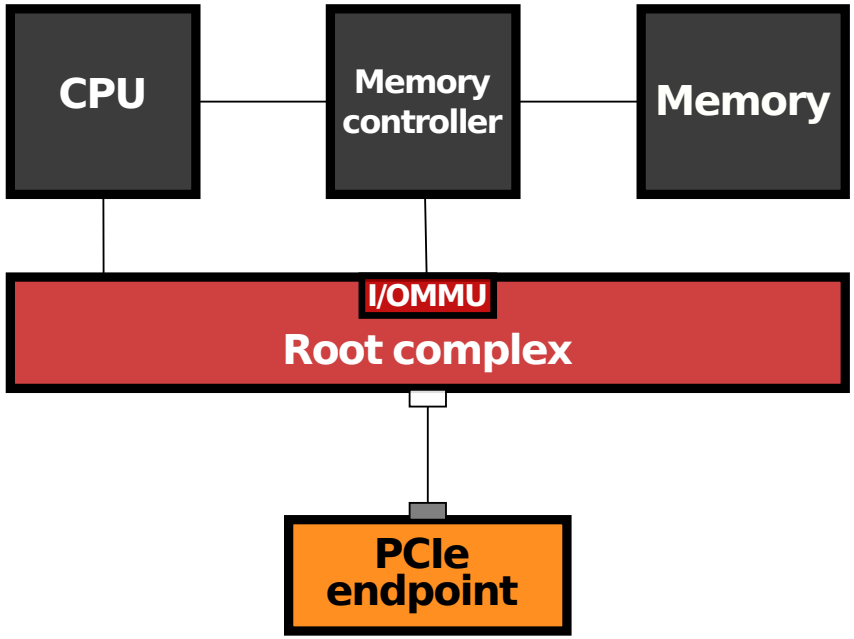
- limites intrinsèques (accès direct...) ;
- garanties apportées ? Sous quelles conditions ?
- définir une liste de points d'attention.

Section 2

Systématiser l'évaluation du cloisonnement apporté

Qu'est-ce qu'un système PCIe ?

- CPU, mémoire ;
- hiérarchie PCIe ;
- *root complex*.



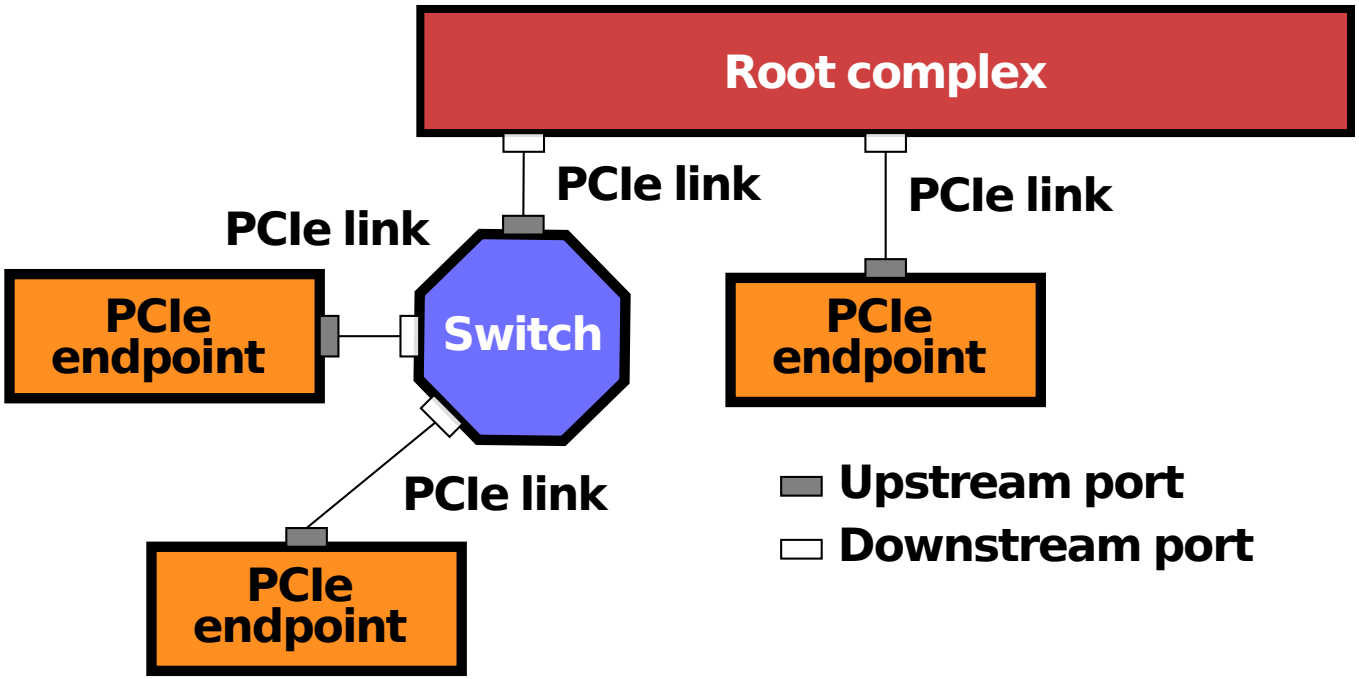
Hiérarchie PCIe

root complex racine de la hiérarchie, interface avec le système ;

endpoint périphérique, feuille de l'arbre ;

switch élément de routage ;

link lien physique.



Communications au sein de la hiérarchie PCIe

Pas un bus mais un réseau

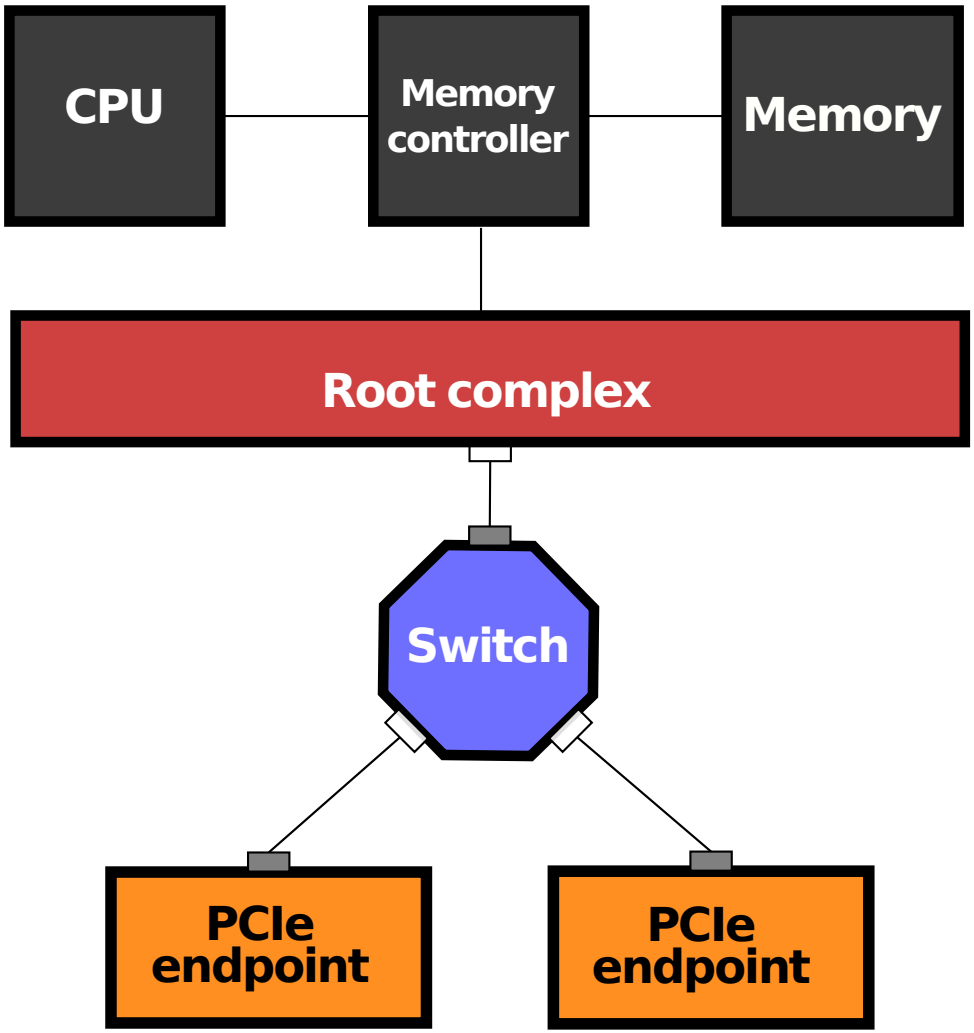
- organisation arborescente ;
- communications pair à pair ;
- pile protocolaire à trois couches :
  physique, liaison de données, **transaction**.

Couche transaction

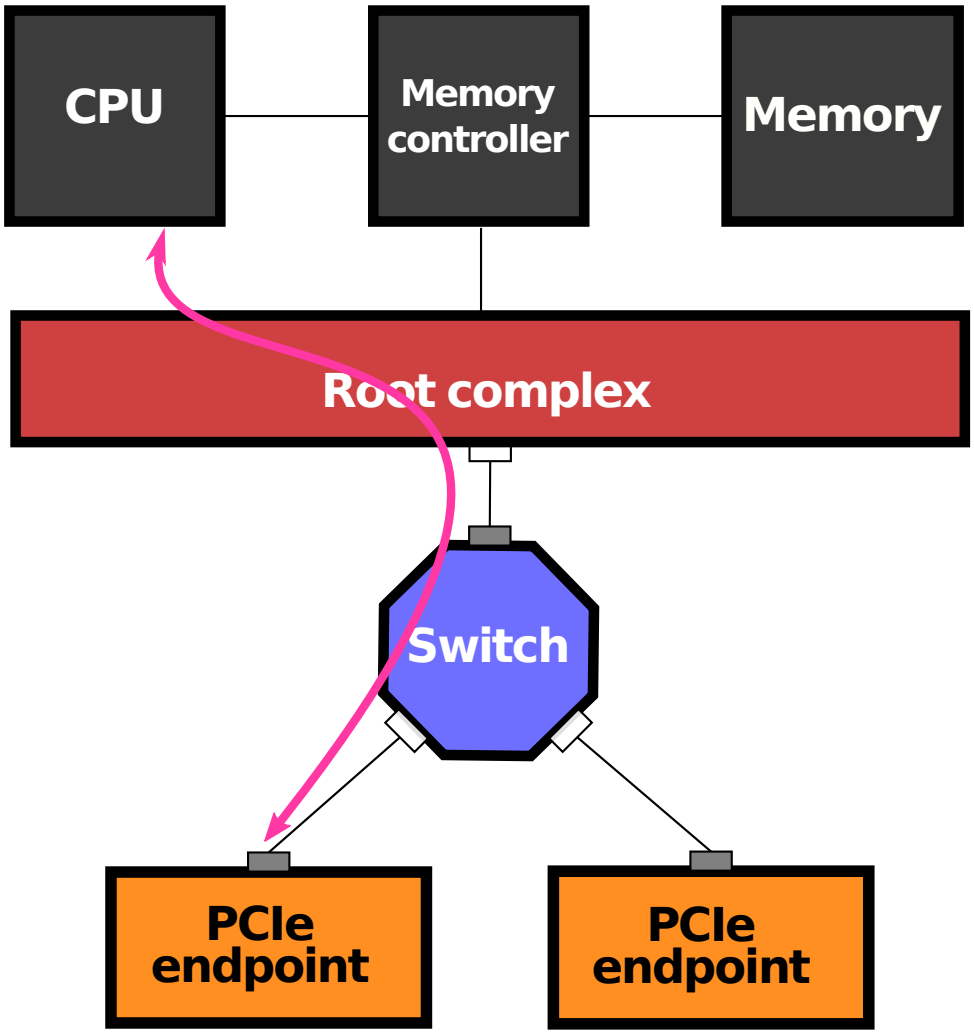
- routage des paquets : *transaction layer packets* (TLP) ;
- transactions :

  posted requête uniquement ;
  non-posted requête et réponse (*completion*).

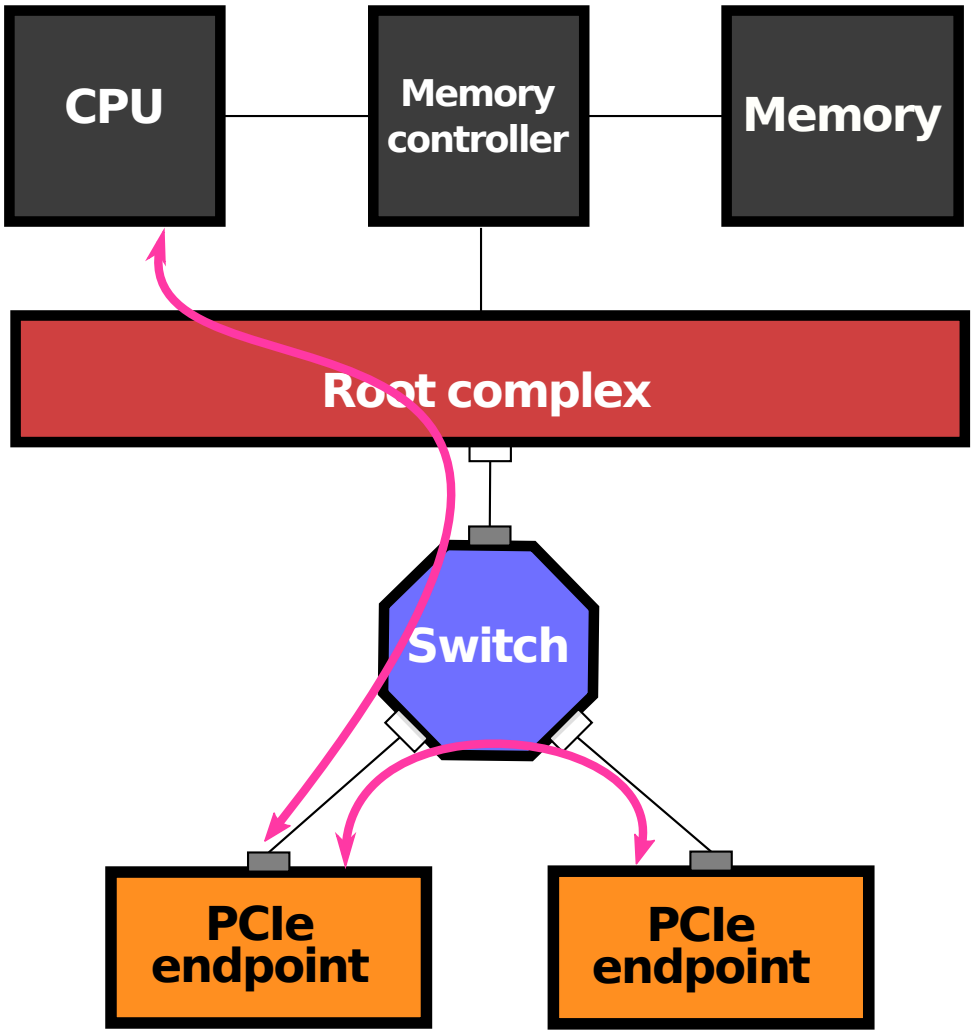
La hiérarchie PCIe et le reste du système



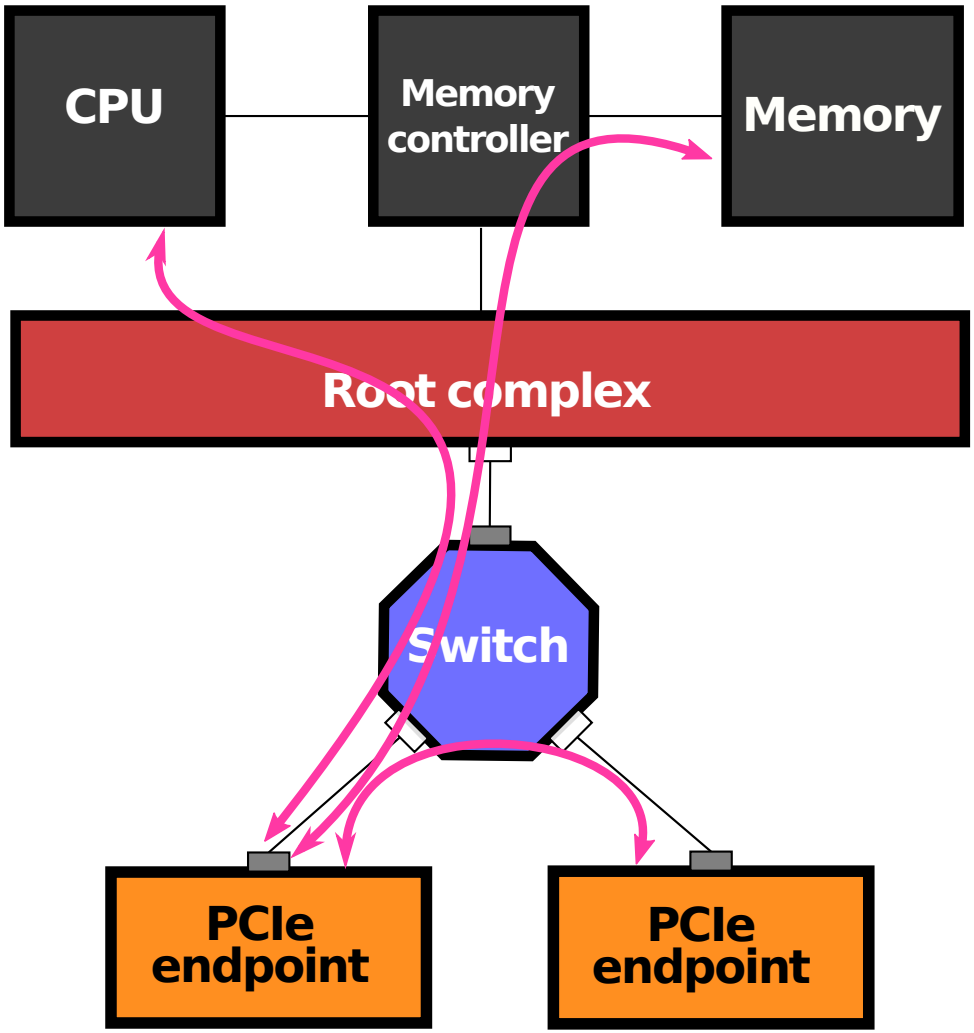
La hiérarchie PCIe et le reste du système



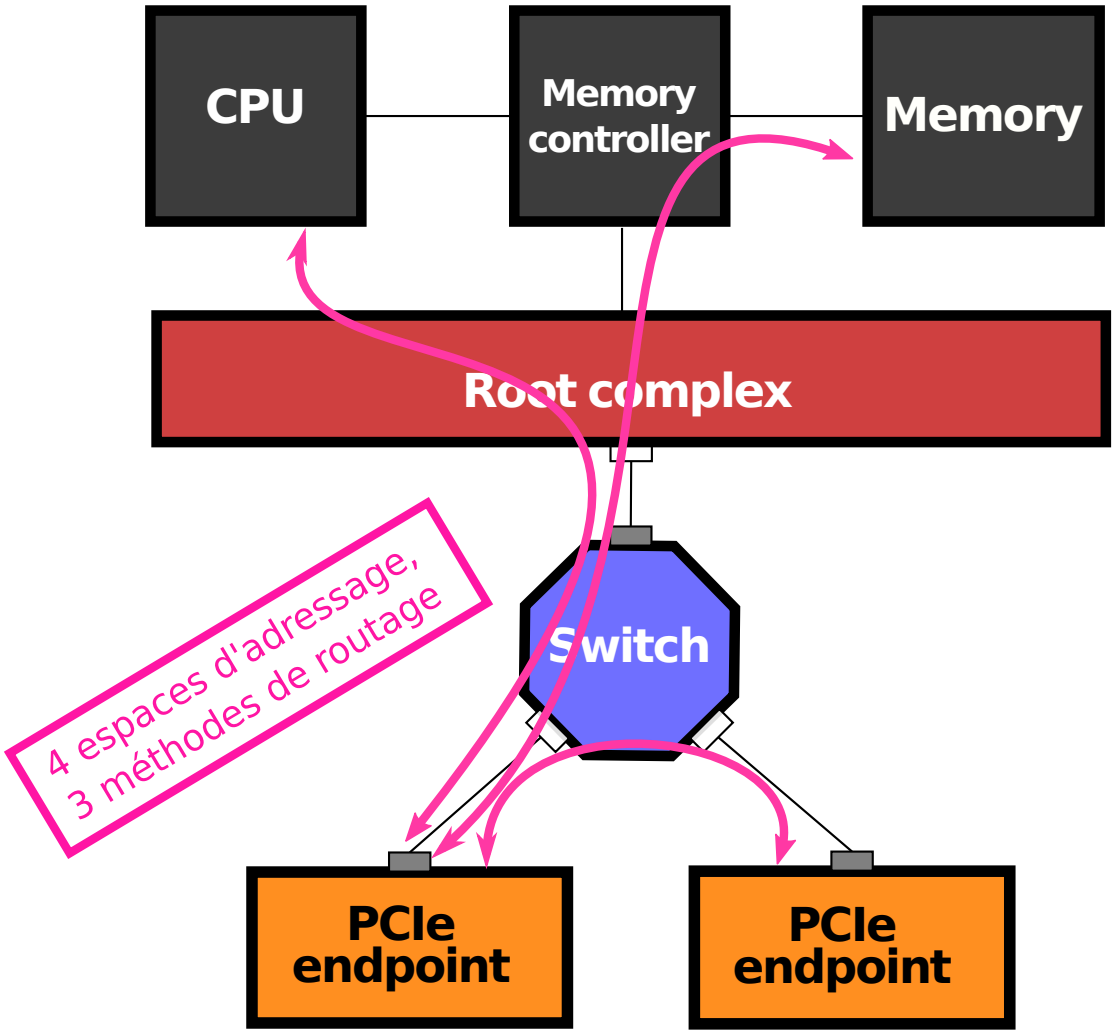
La hiérarchie PCIe et le reste du système



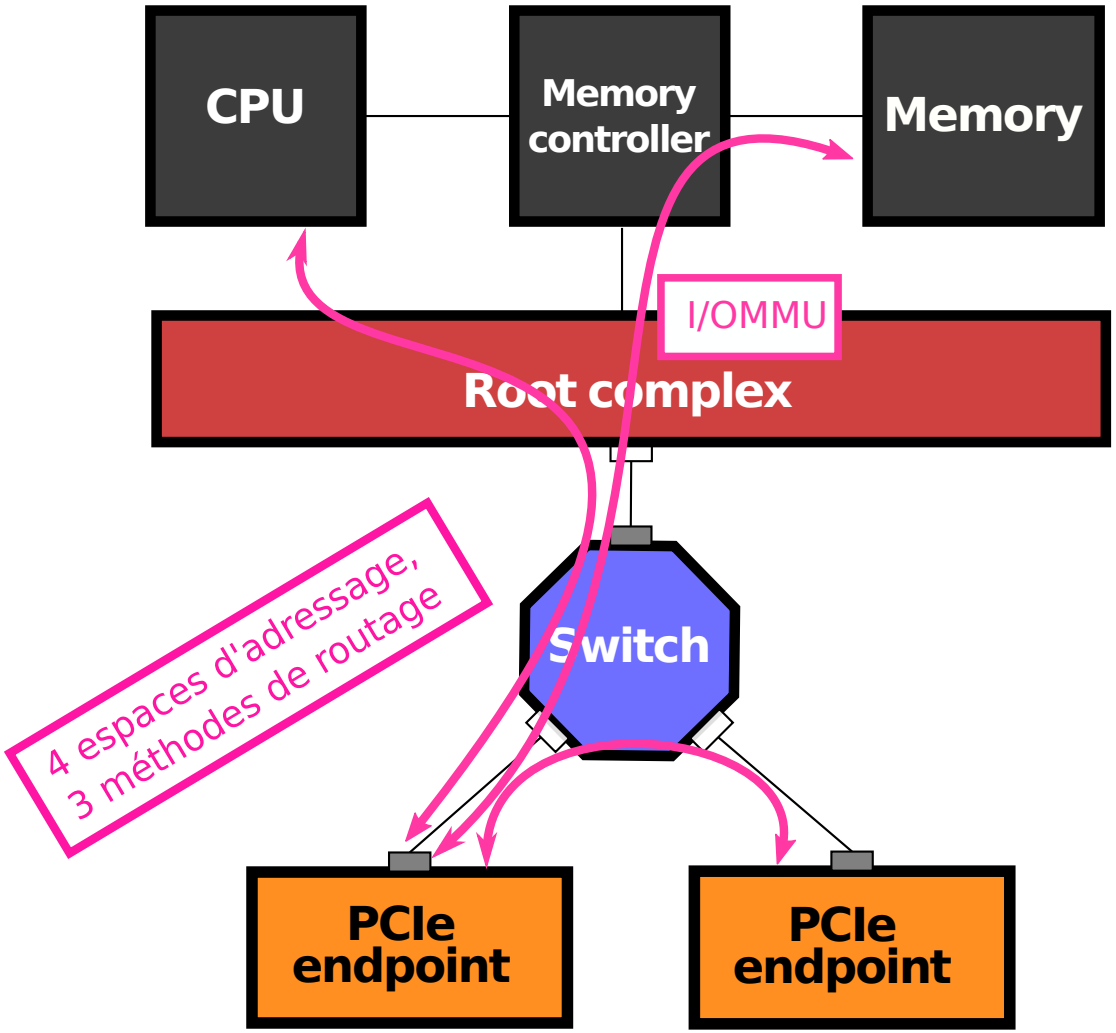
La hiérarchie PCIe et le reste du système



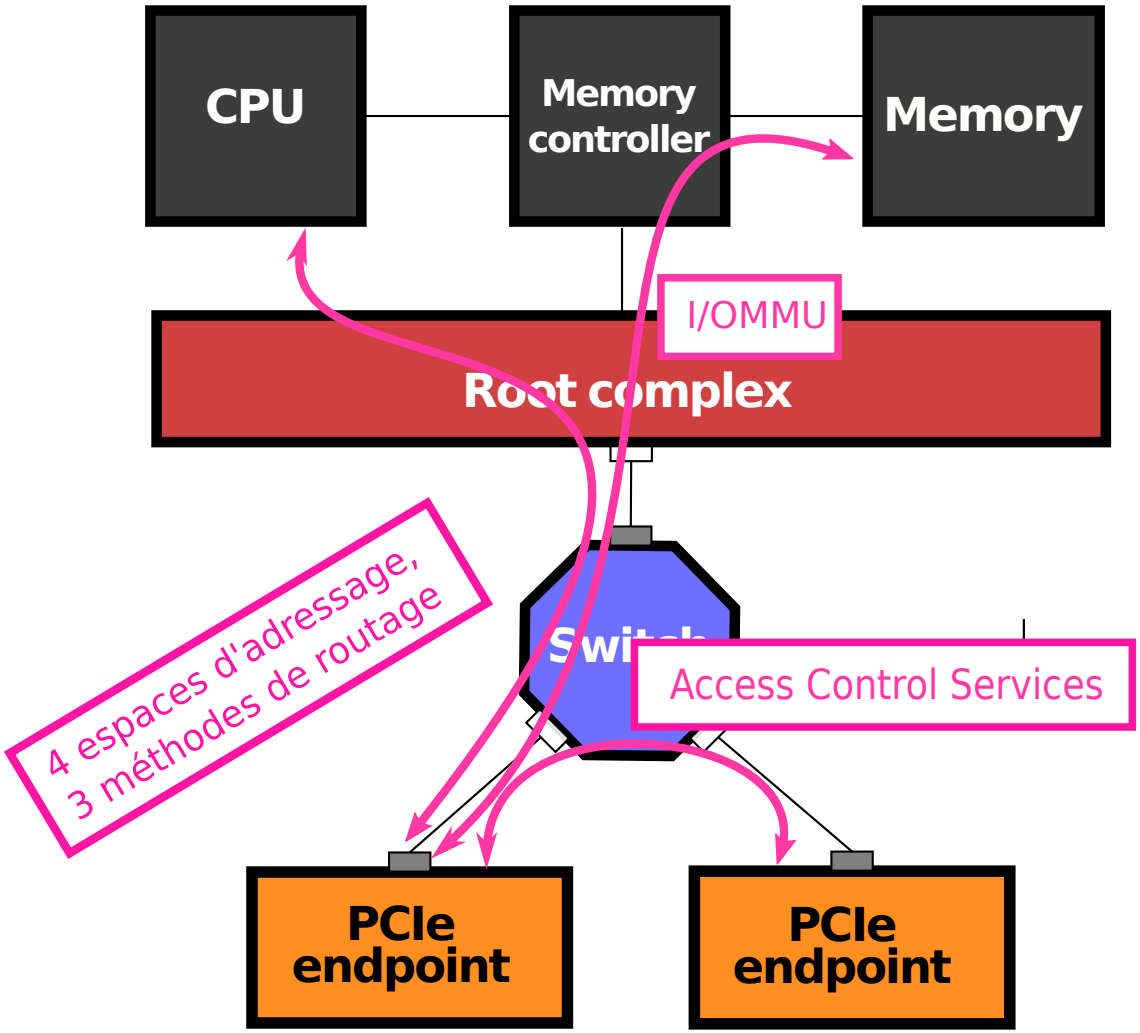
La hiérarchie PCIe et le reste du système



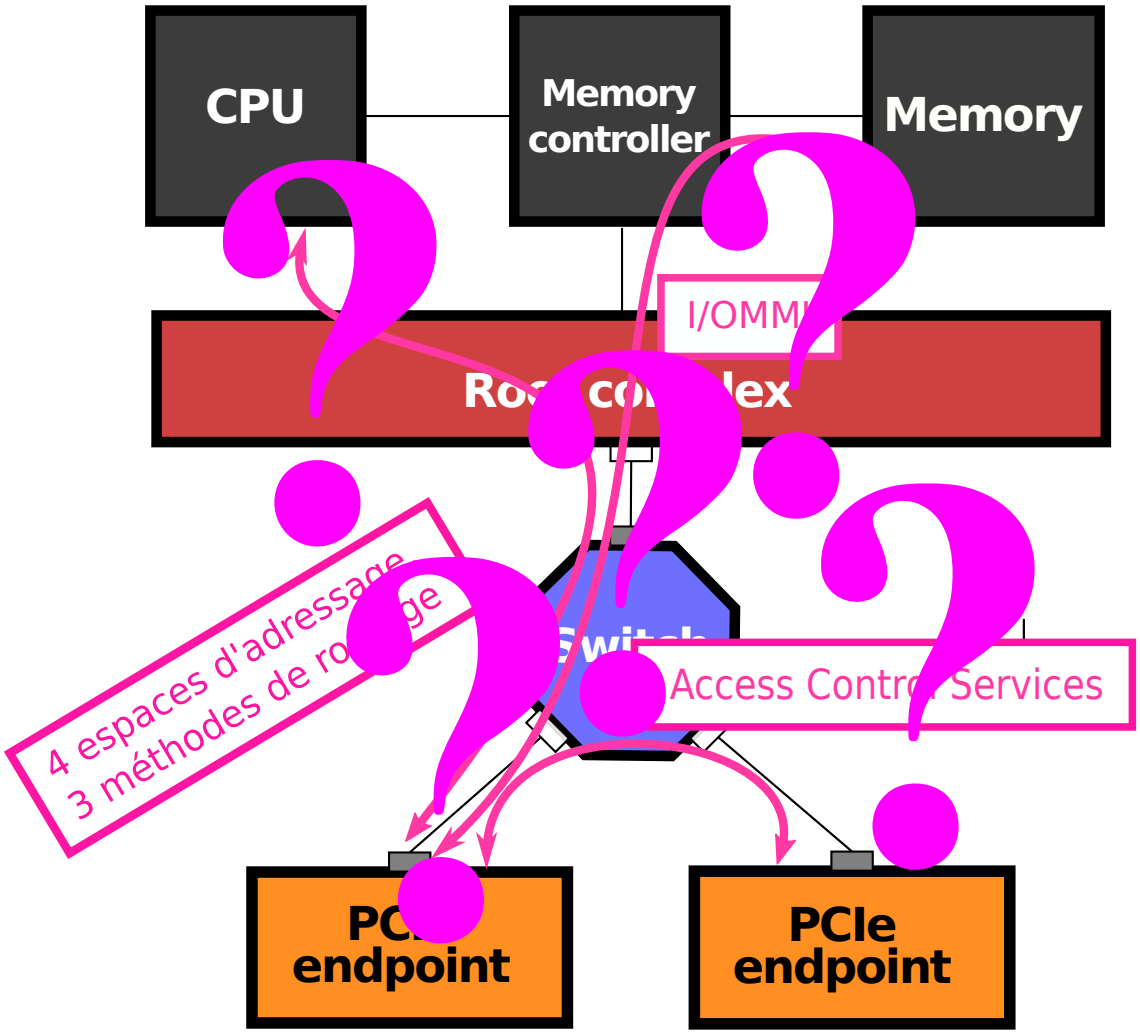
La hiérarchie PCIe et le reste du système



La hiérarchie PCIe et le reste du système



La hiérarchie PCIe et le reste du système



Motivation à développer ProTIP

Analyse systématique sur un modèle

Développer un outil formalisant une telle architecture permet :

- de répondre au besoin de caractériser la connectivité réelle entre les éléments de l'architecture ;
- de mettre en avant les flous dans la spécification.

Section 3

Bases de PCIe

Interactions du système avec un périphérique

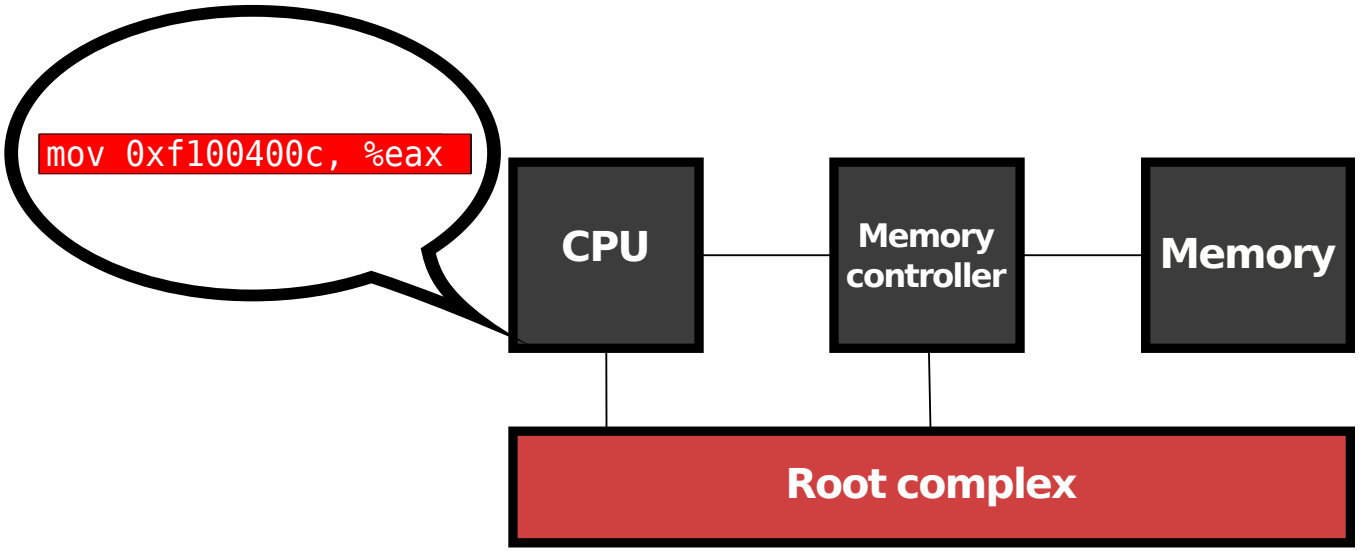
Contrôle du périphérique par le système

- code exécuté par le CPU (pilote de périphérique) ;
- en général code privilégié (ring 0) ;
- accès à de la mémoire exposée par le périphérique.

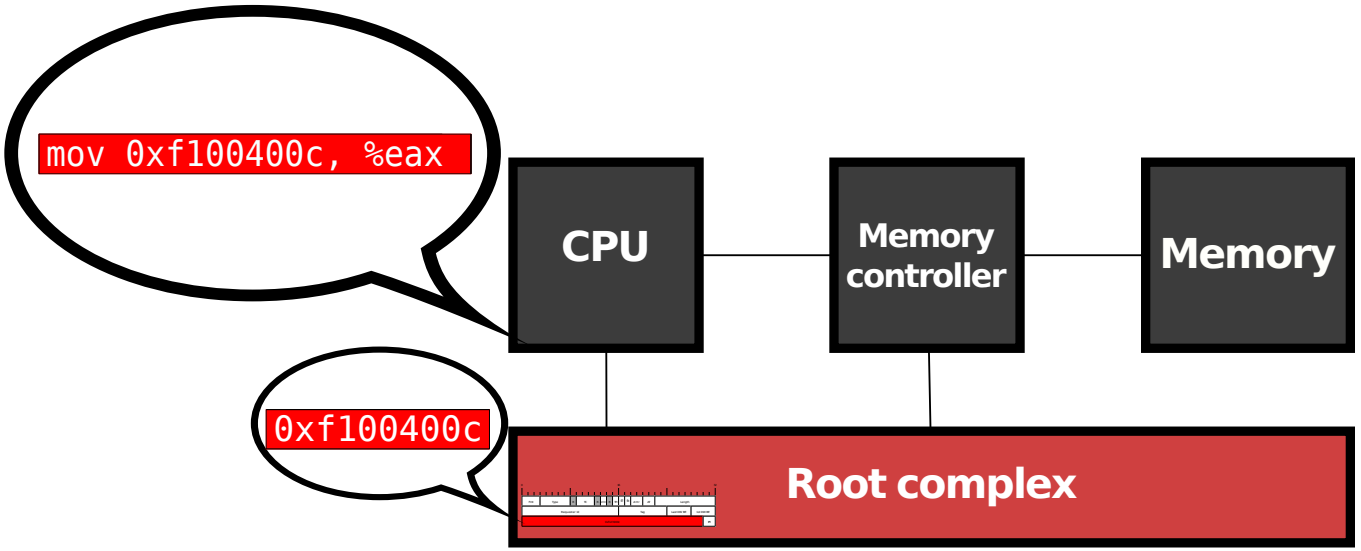
Échanges de données

- synchronisation effectuée par le pilote ;
- transferts à l'initiative du périphérique ;
- lecture/écriture directement vers la mémoire centrale (DMA).

Lecture depuis le pilote vers un périphérique

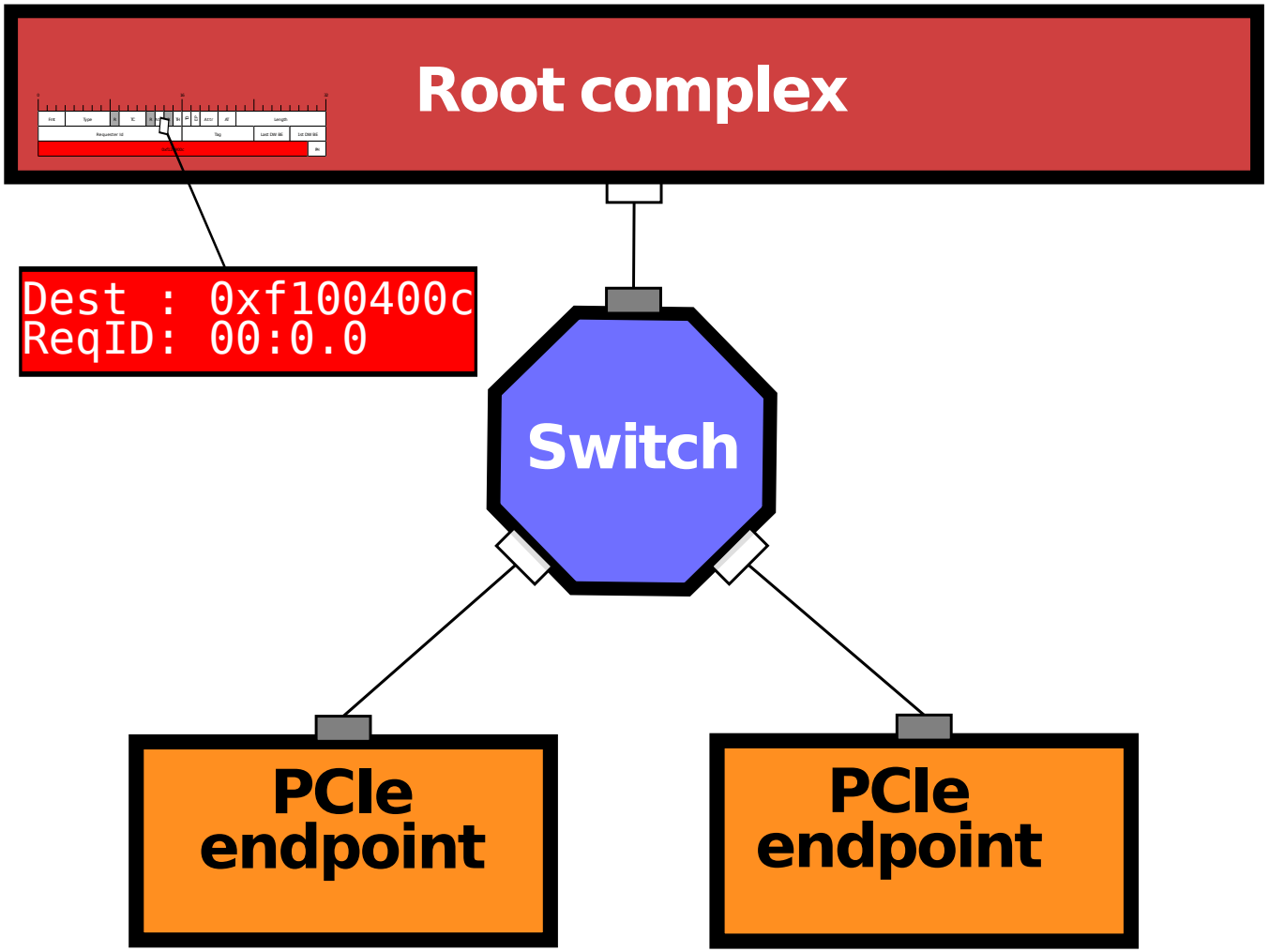


Lecture depuis le pilote vers un périphérique



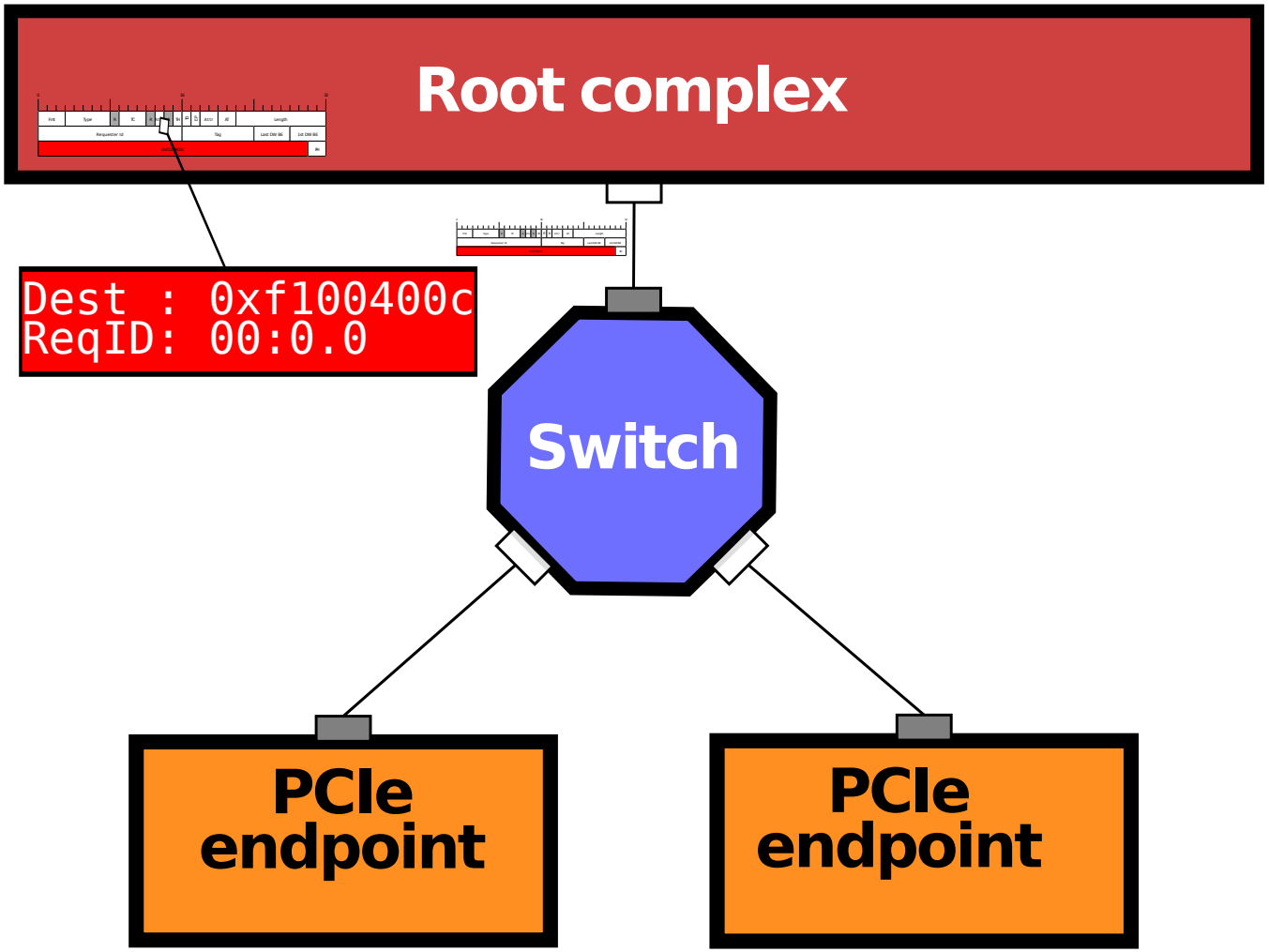
Trajet d'un TLP *memory read request*

routage par adresse



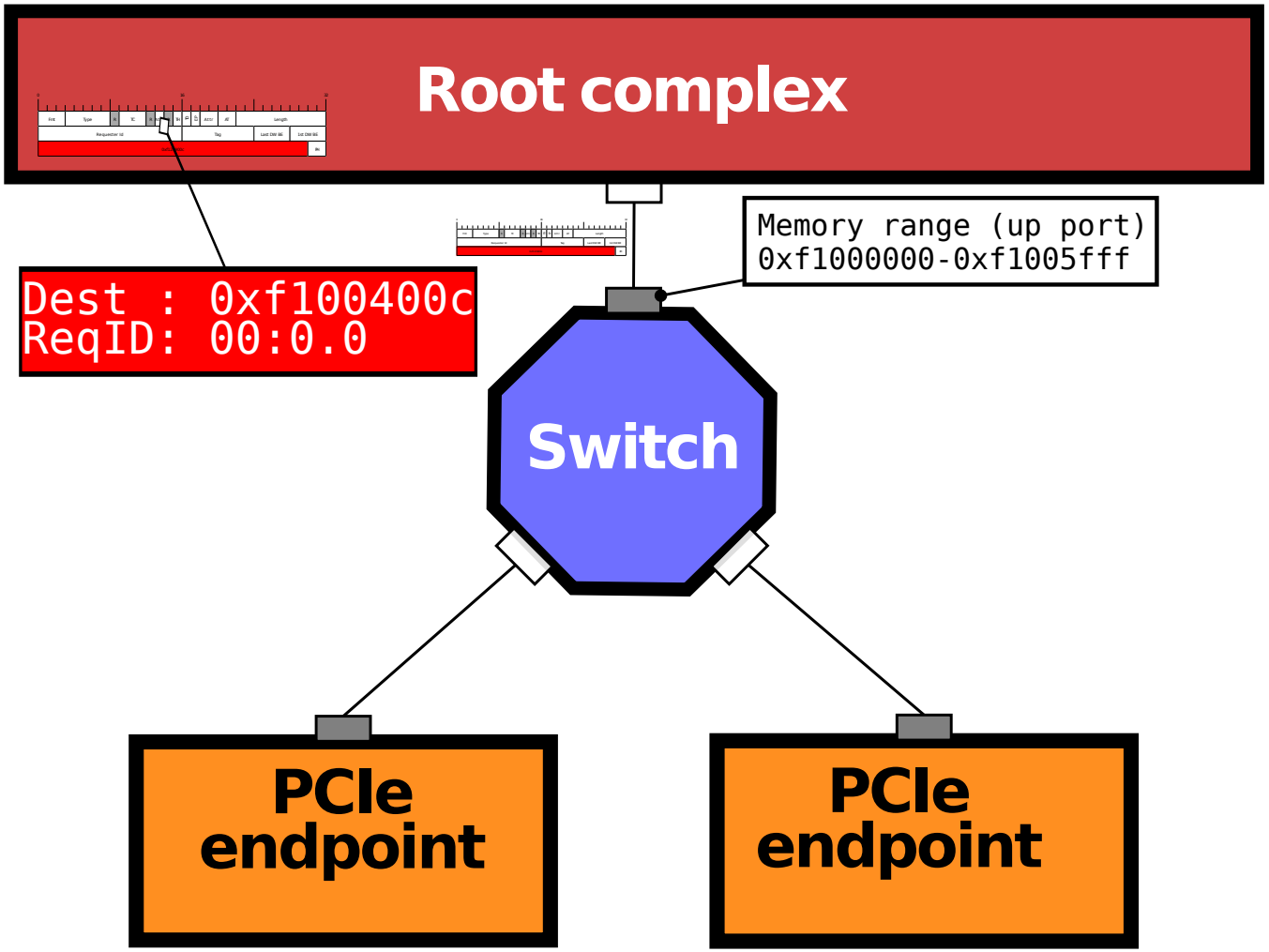
Trajet d'un TLP *memory read request*

routage par adresse



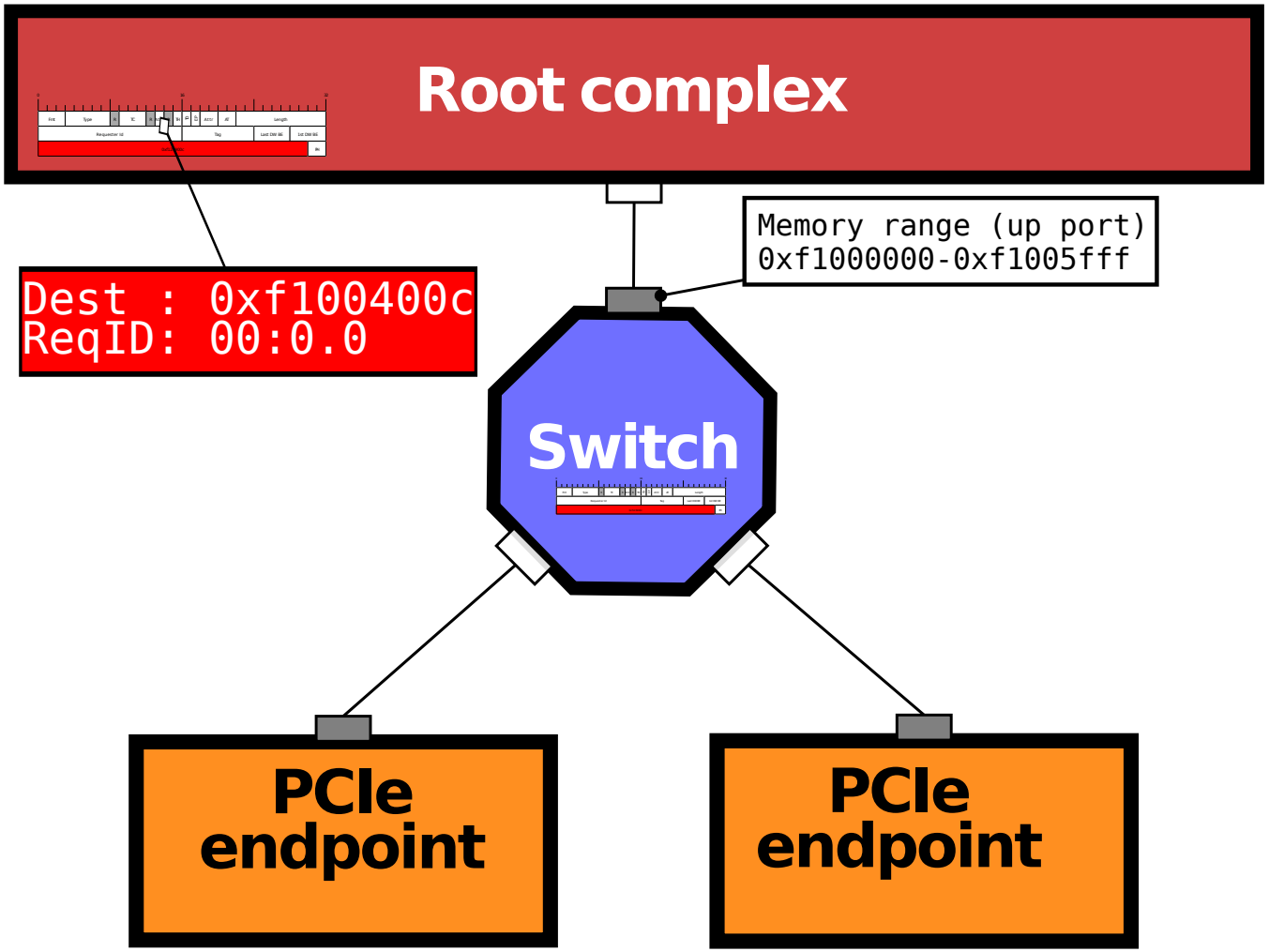
Trajet d'un TLP *memory read request*

routage par adresse



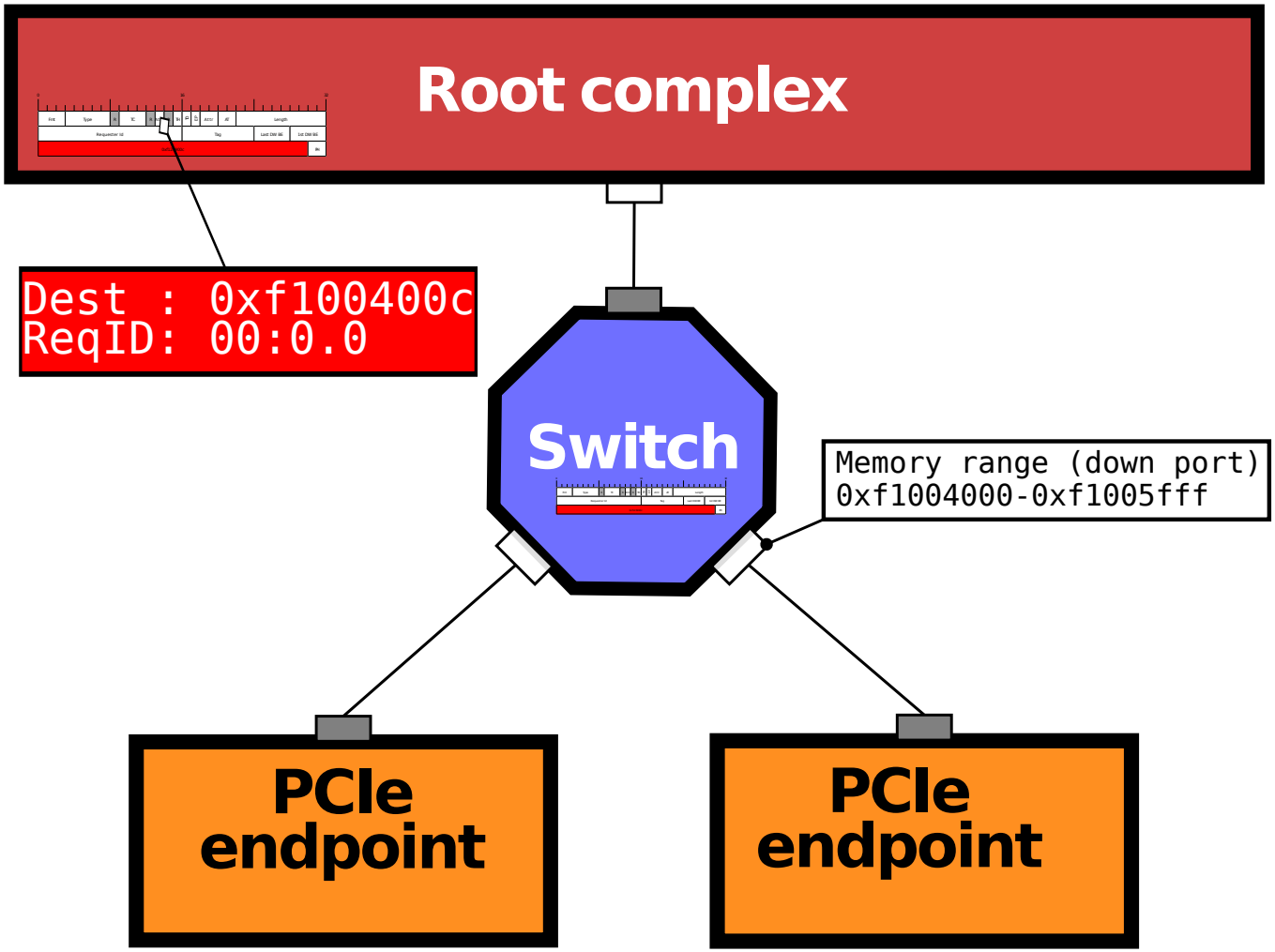
Trajet d'un TLP *memory read request*

routage par adresse



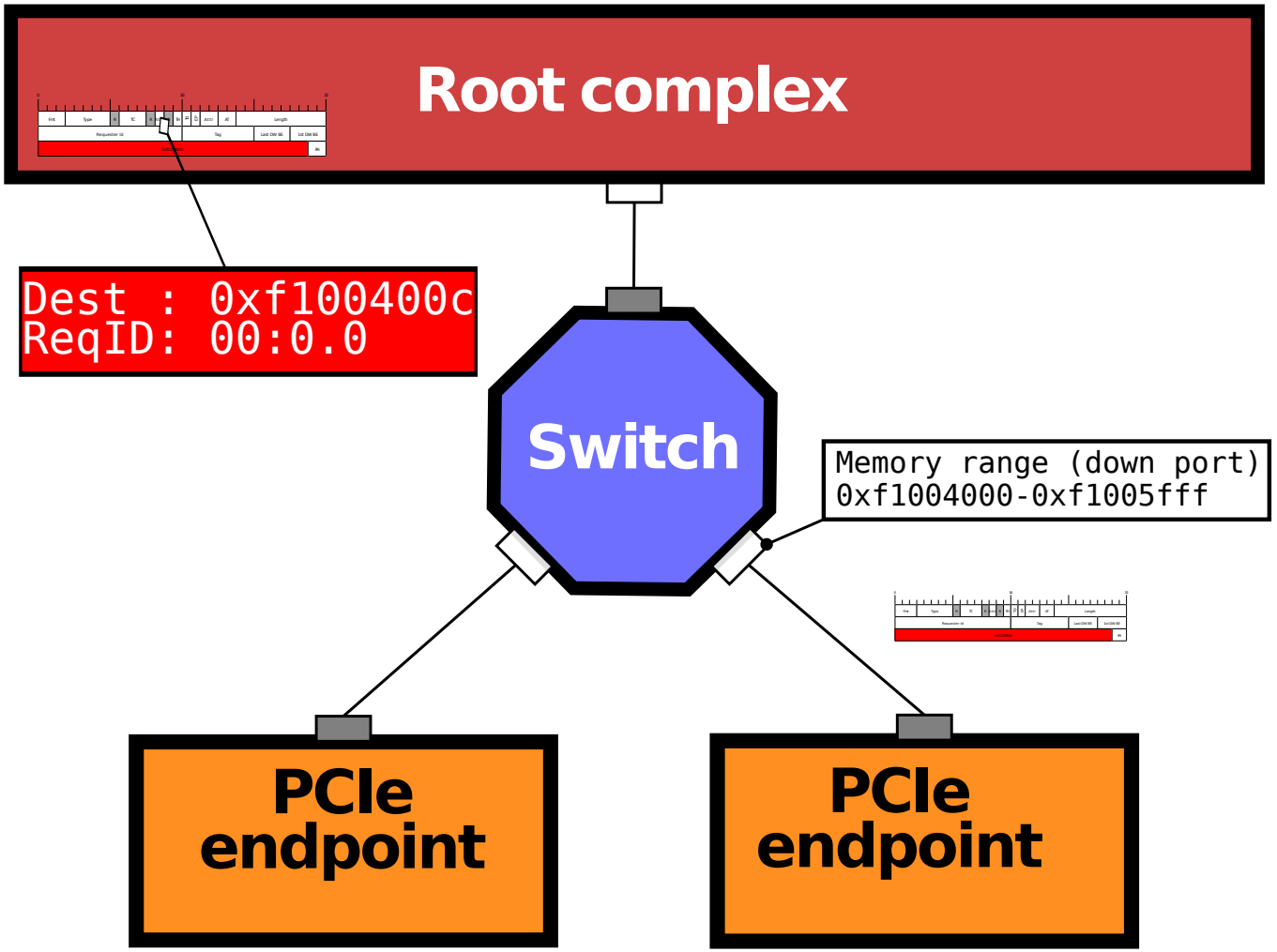
Trajet d'un TLP *memory read request*

routage par adresse



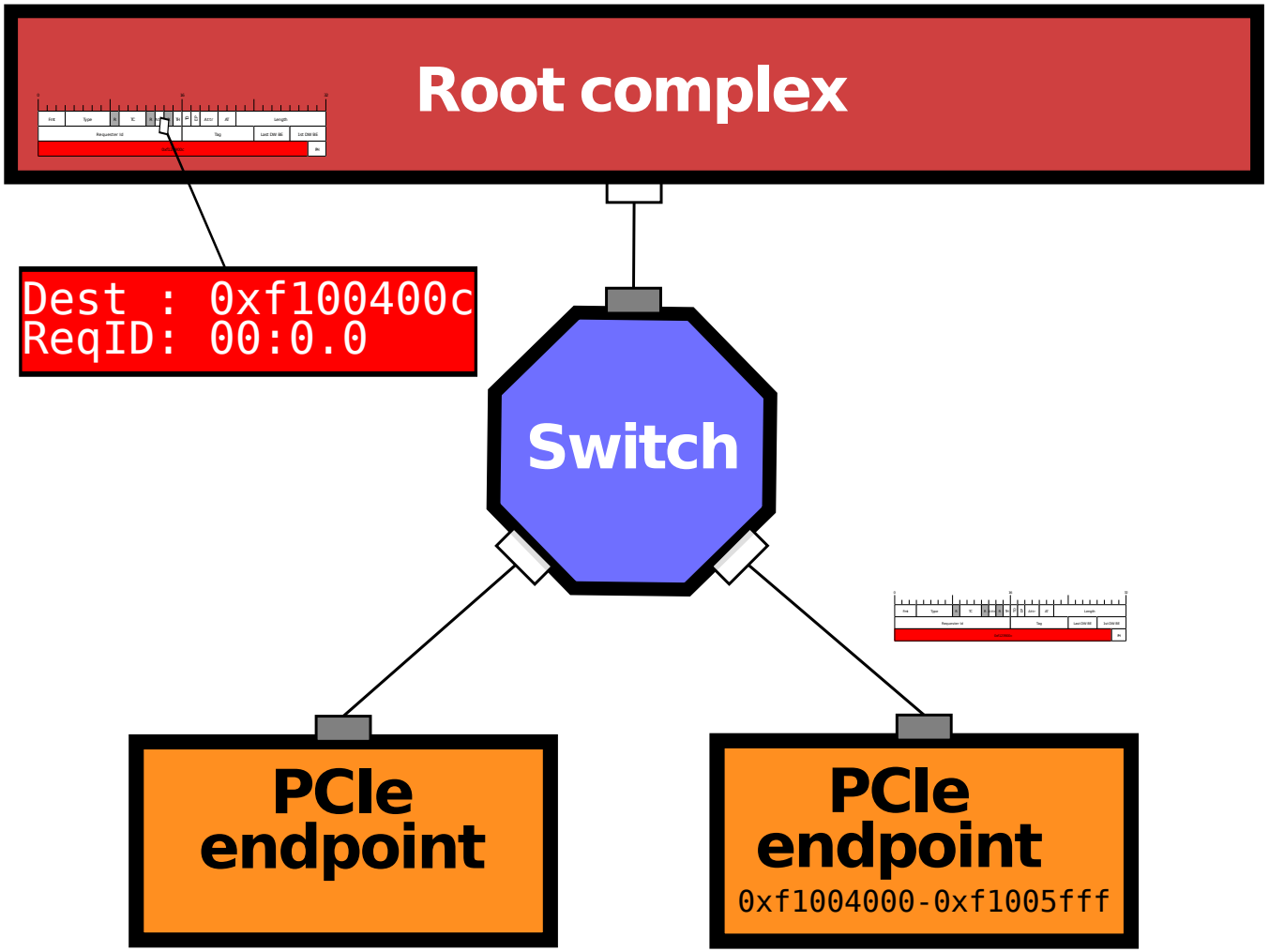
Trajet d'un TLP *memory read request*

routage par adresse



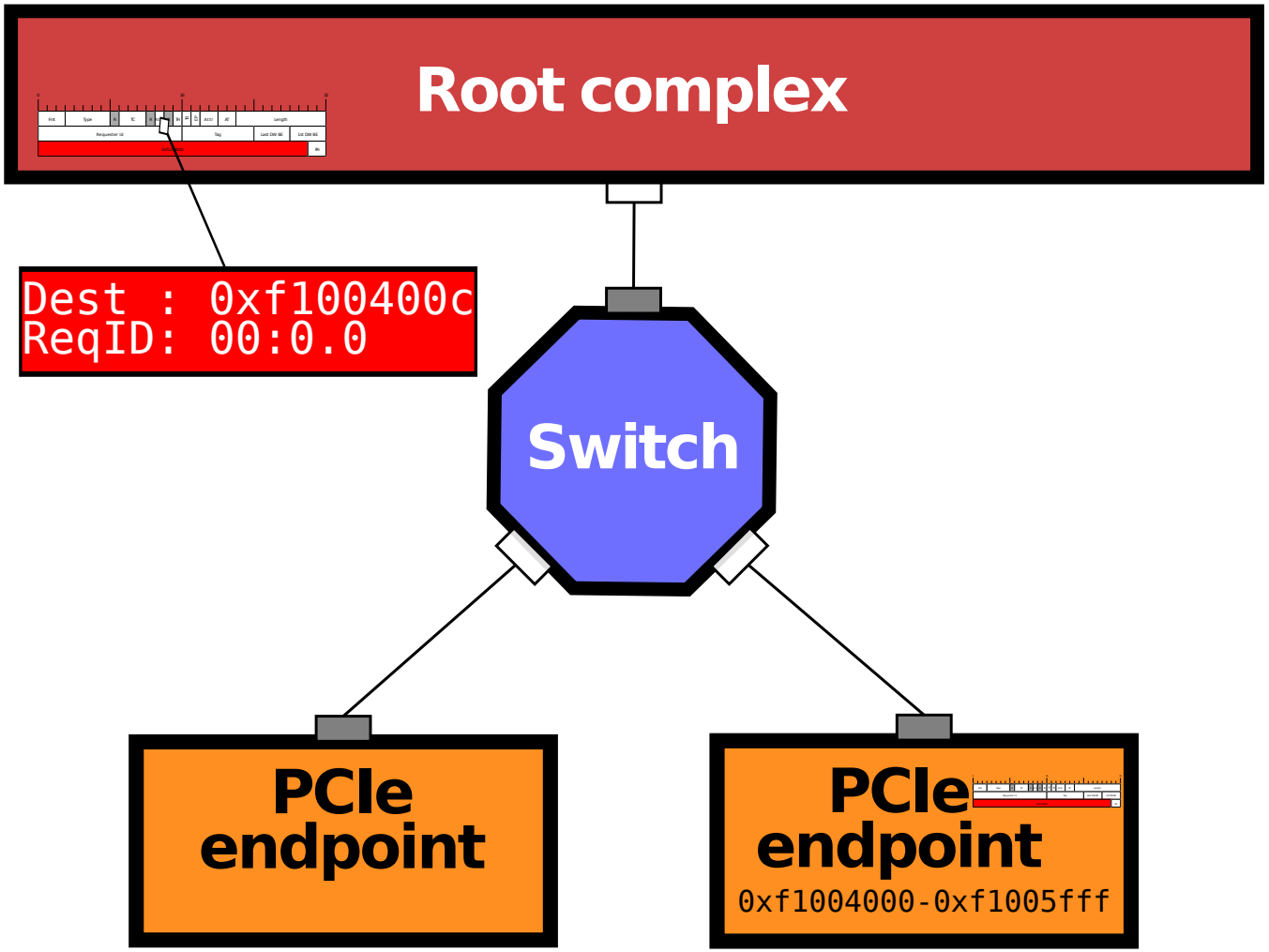
Trajet d'un TLP *memory read request*

routage par adresse



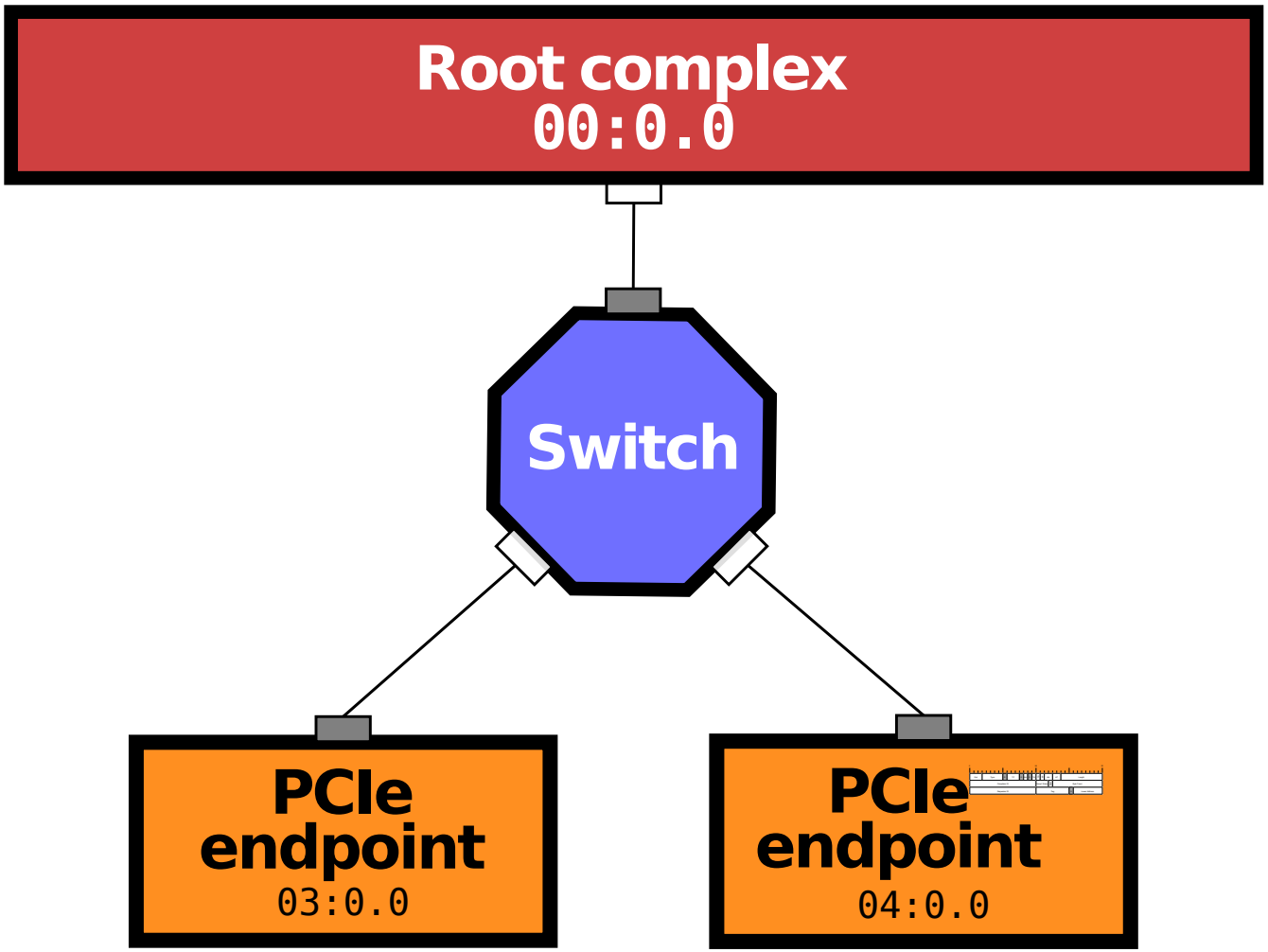
Trajet d'un TLP *memory read request*

routage par adresse



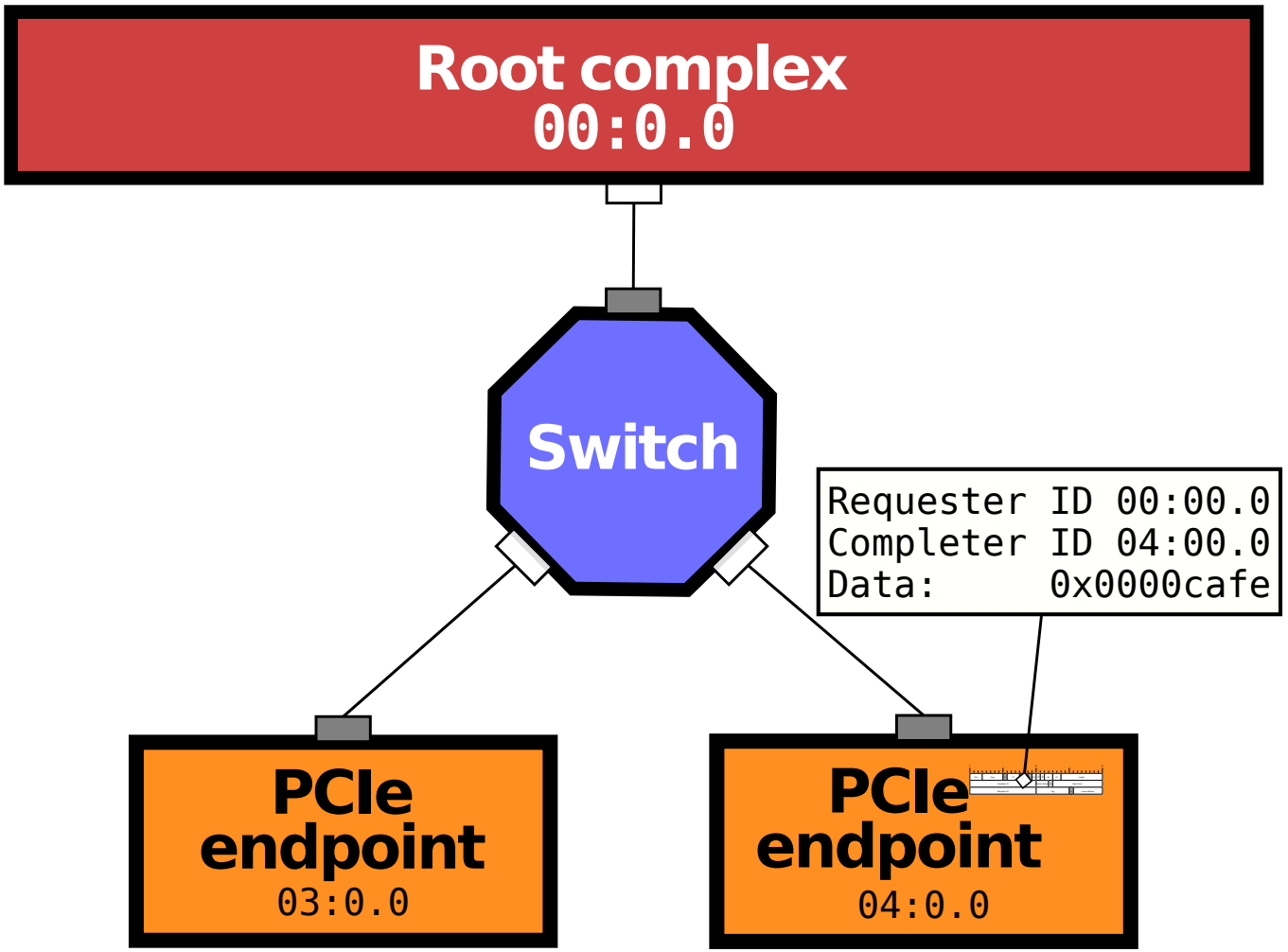
Trajet d'un TLP *memory read completion*

routage par ID



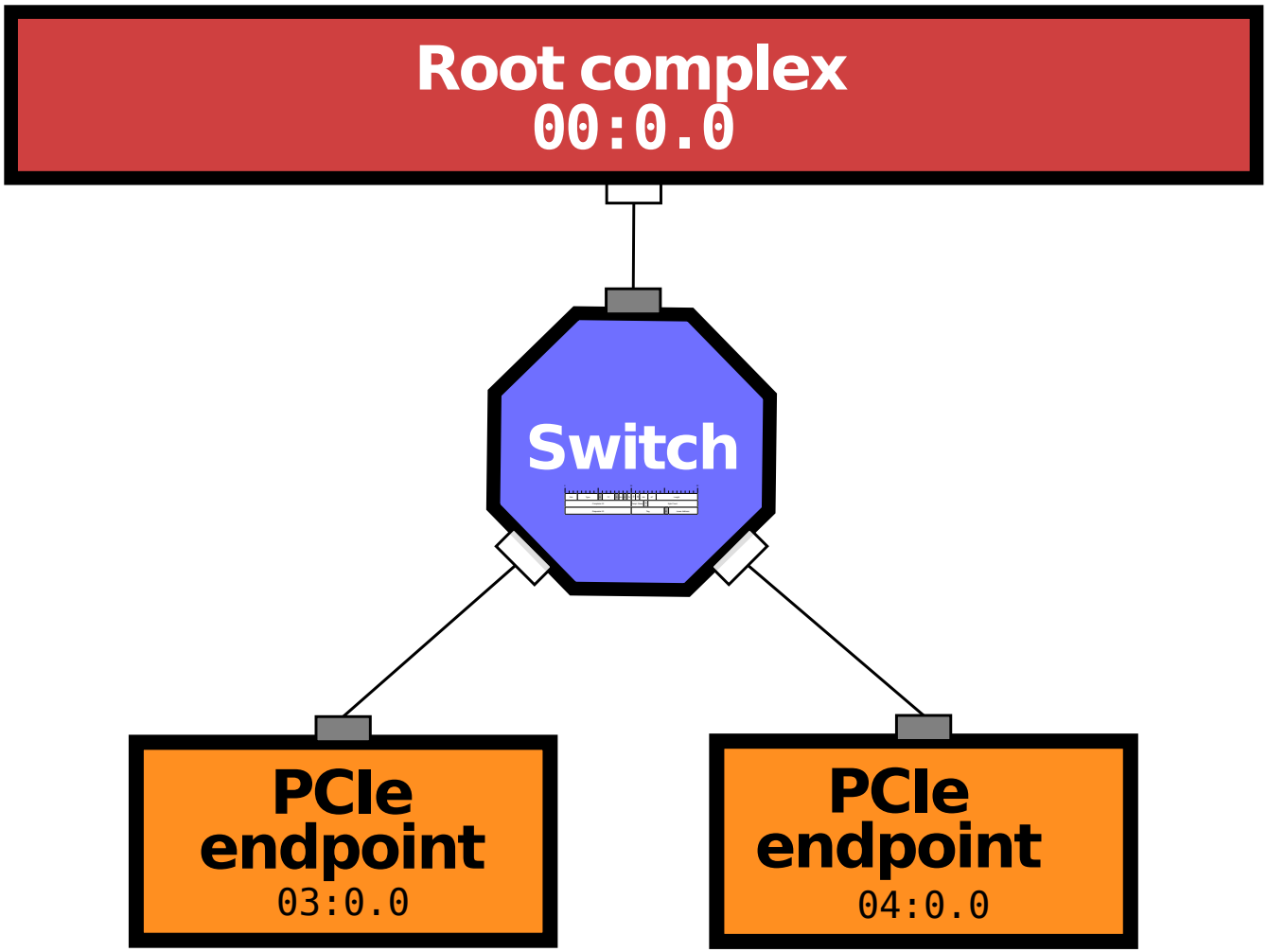
Trajet d'un TLP *memory read completion*

routage par ID



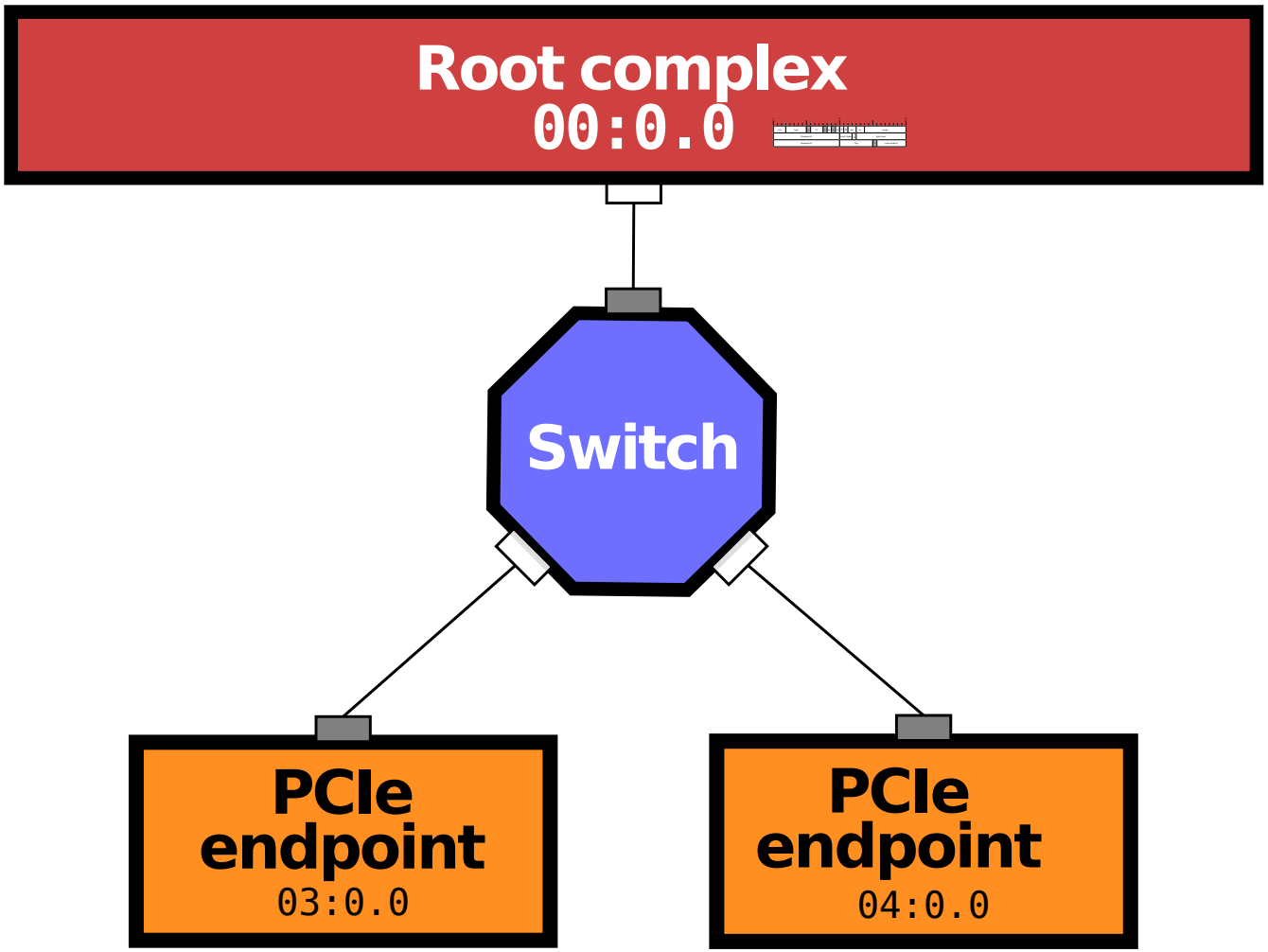
Trajet d'un TLP *memory read completion*

routage par ID

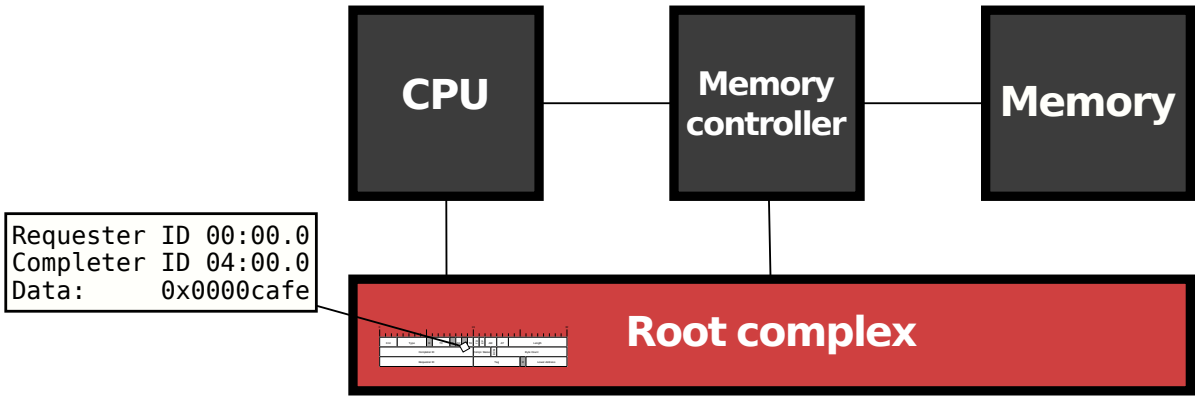


Trajet d'un TLP *memory read completion*

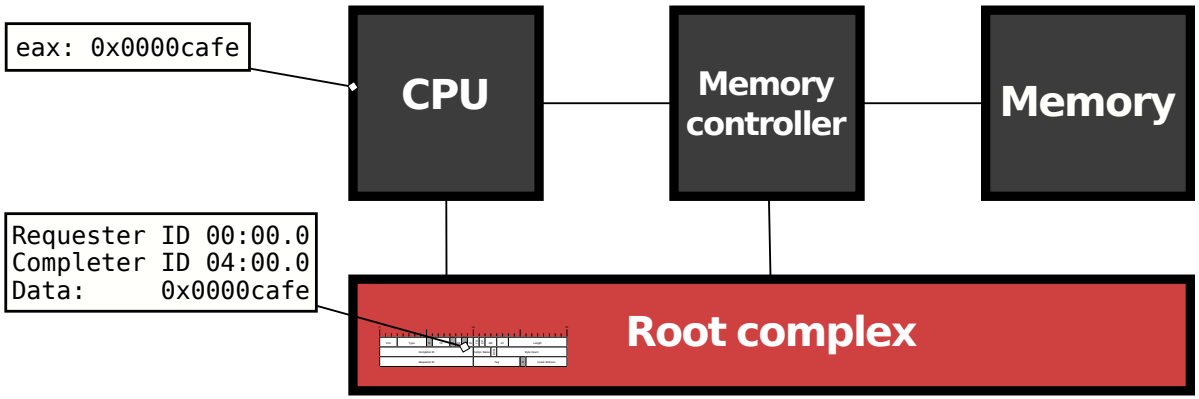
routage par ID



Lecture depuis le pilote vers un périphérique

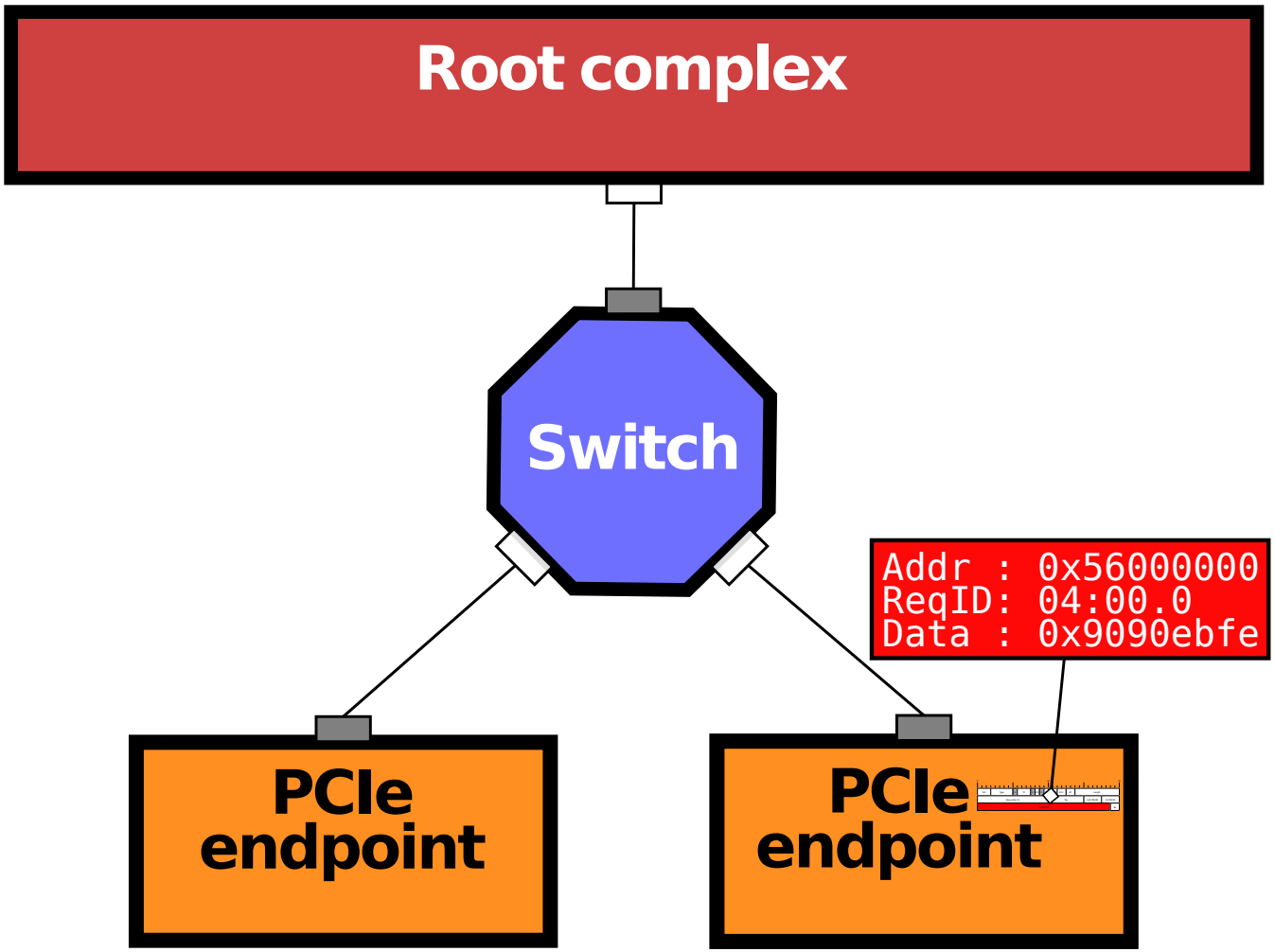


Lecture depuis le pilote vers un périphérique



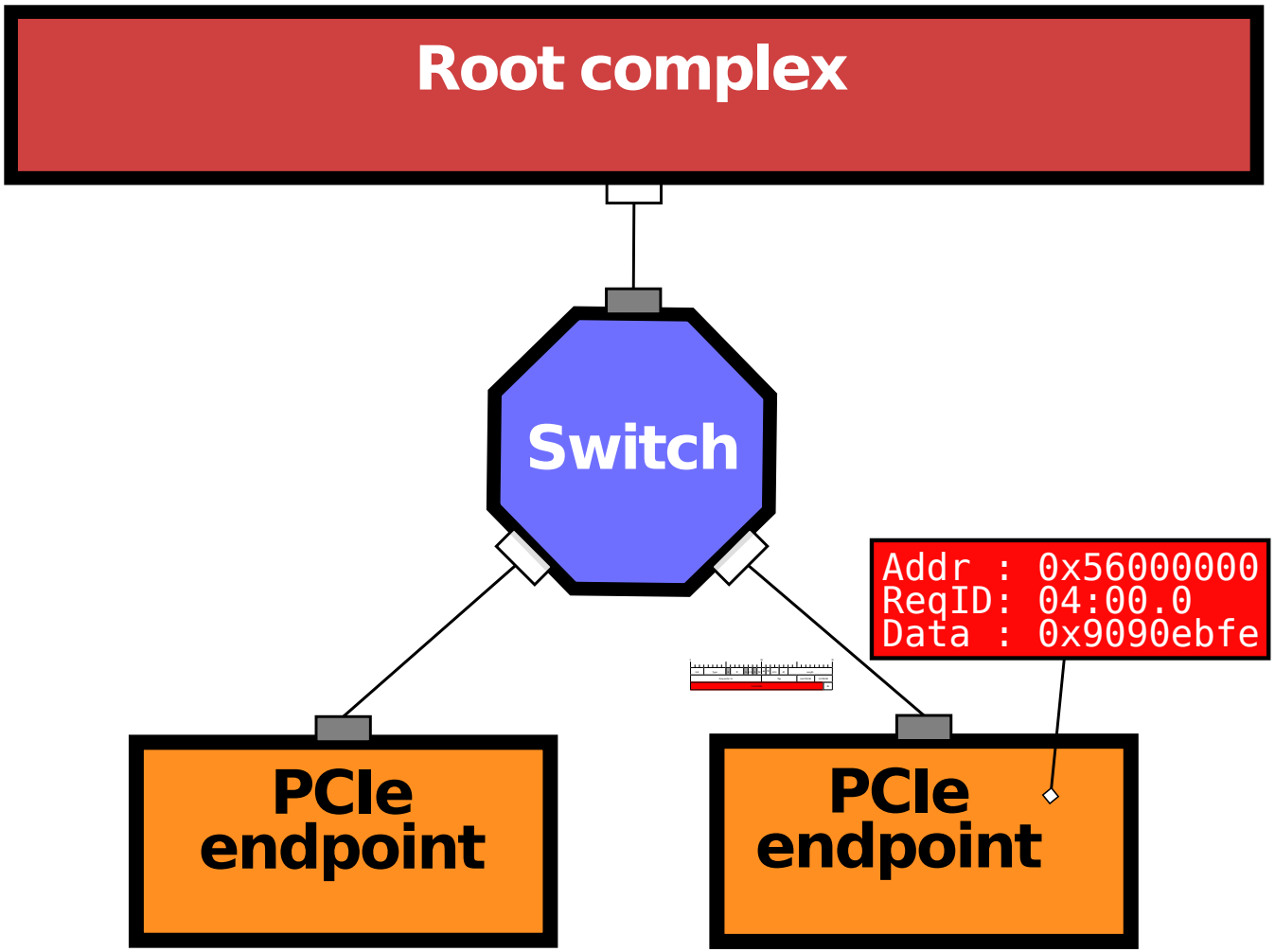
Écriture DMA depuis un périphérique

PCI Express



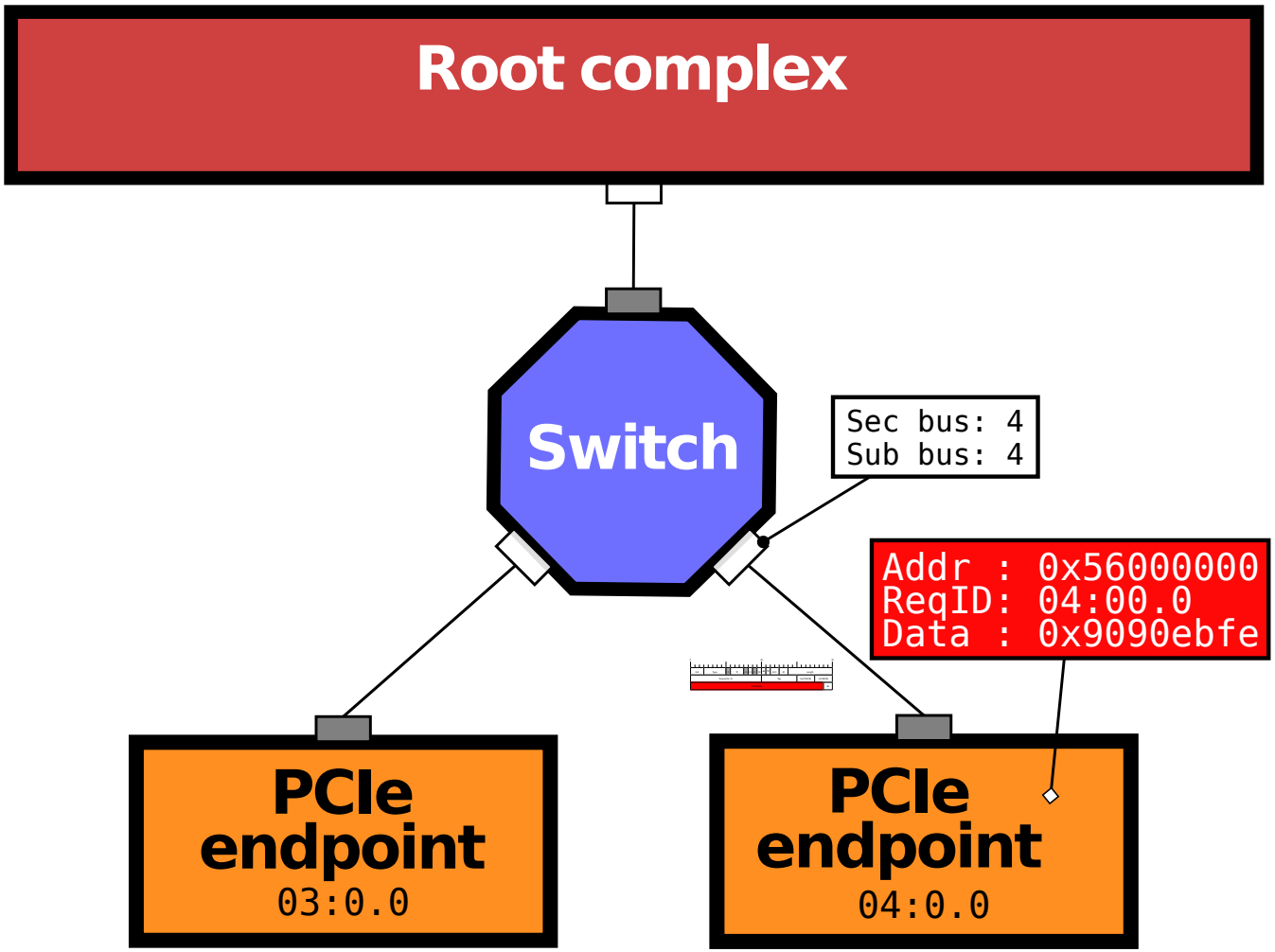
Écriture DMA depuis un périphérique

PCI Express



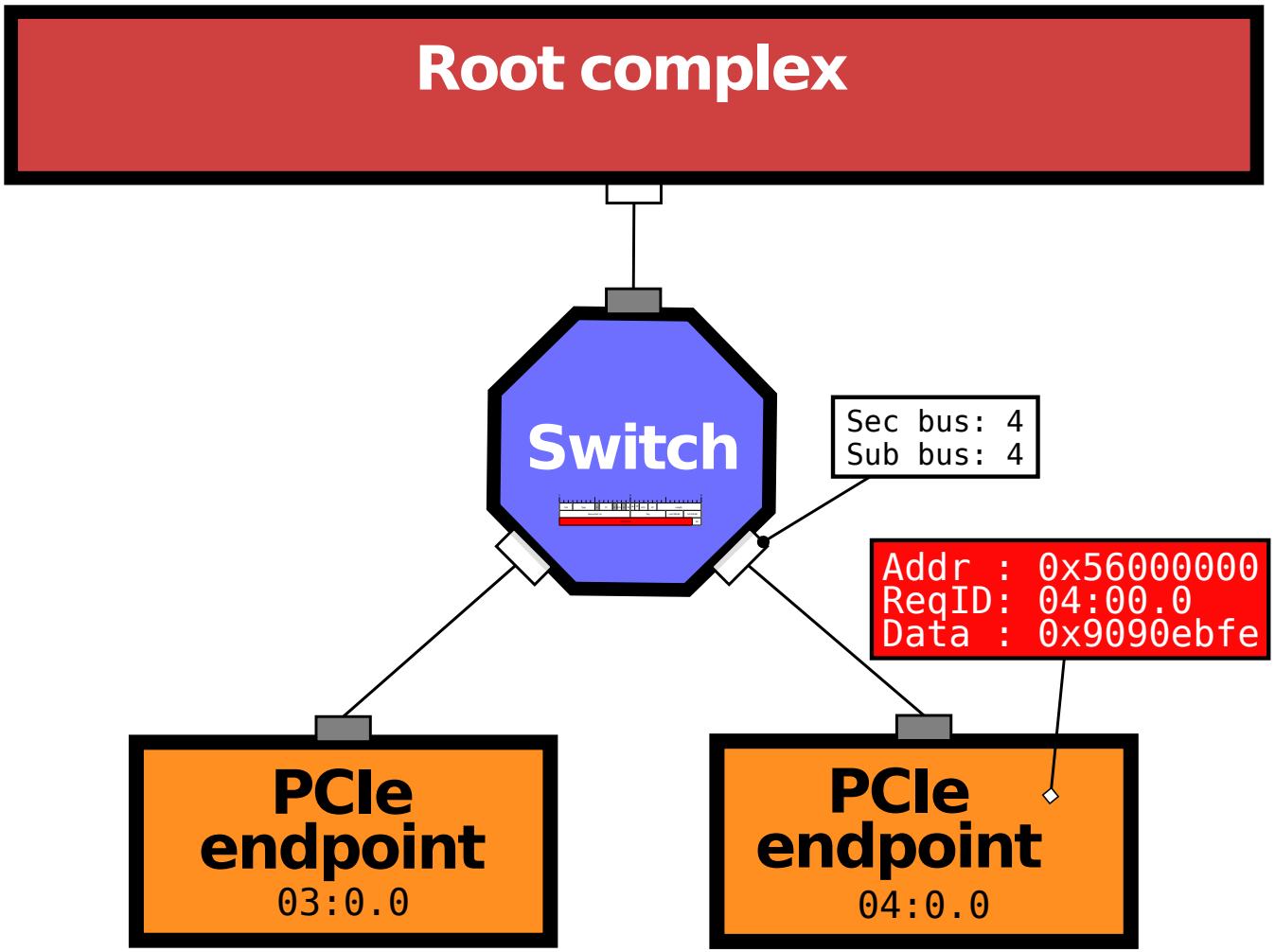
Écriture DMA depuis un périphérique

PCI Express



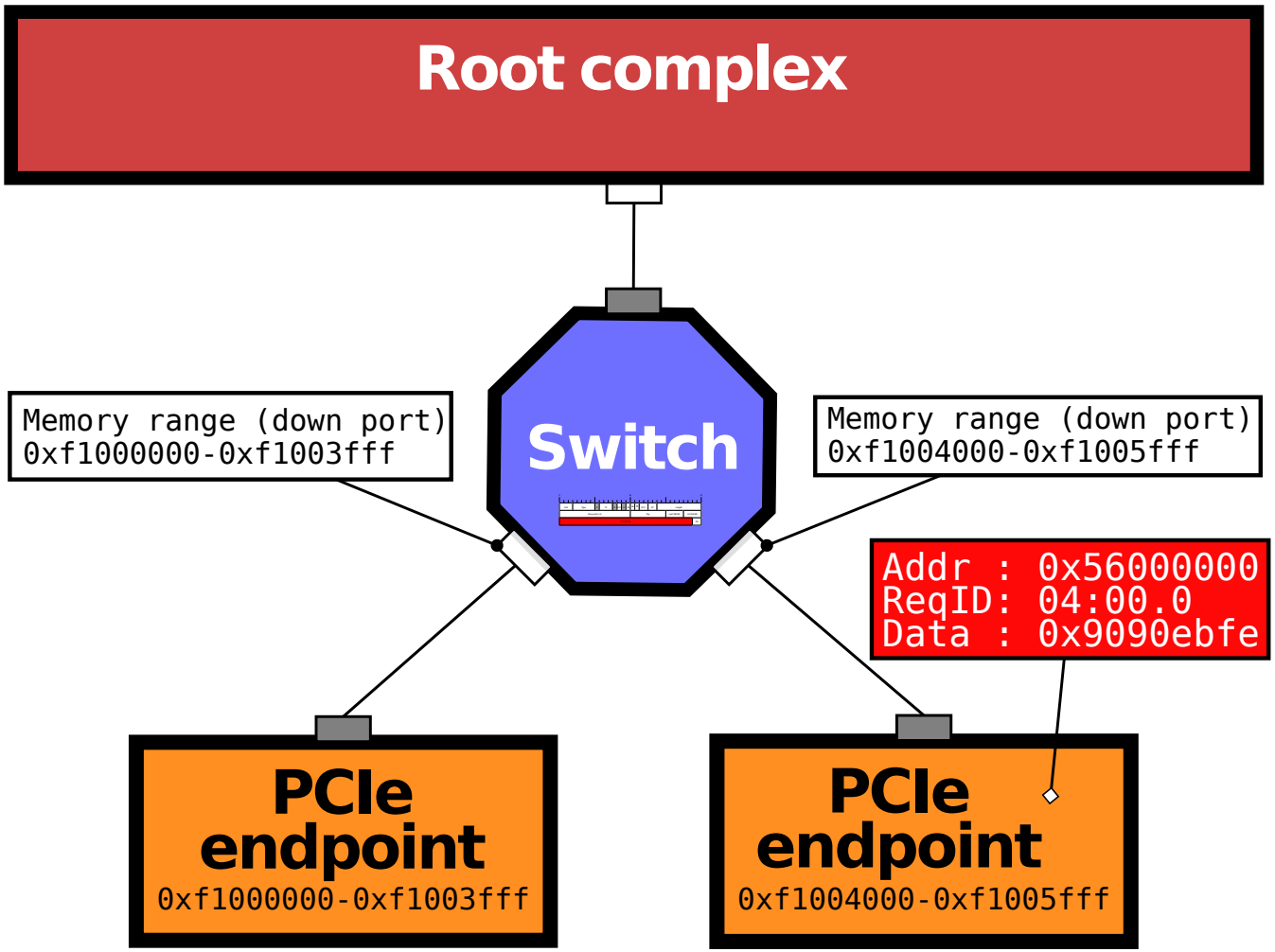
Écriture DMA depuis un périphérique

PCI Express



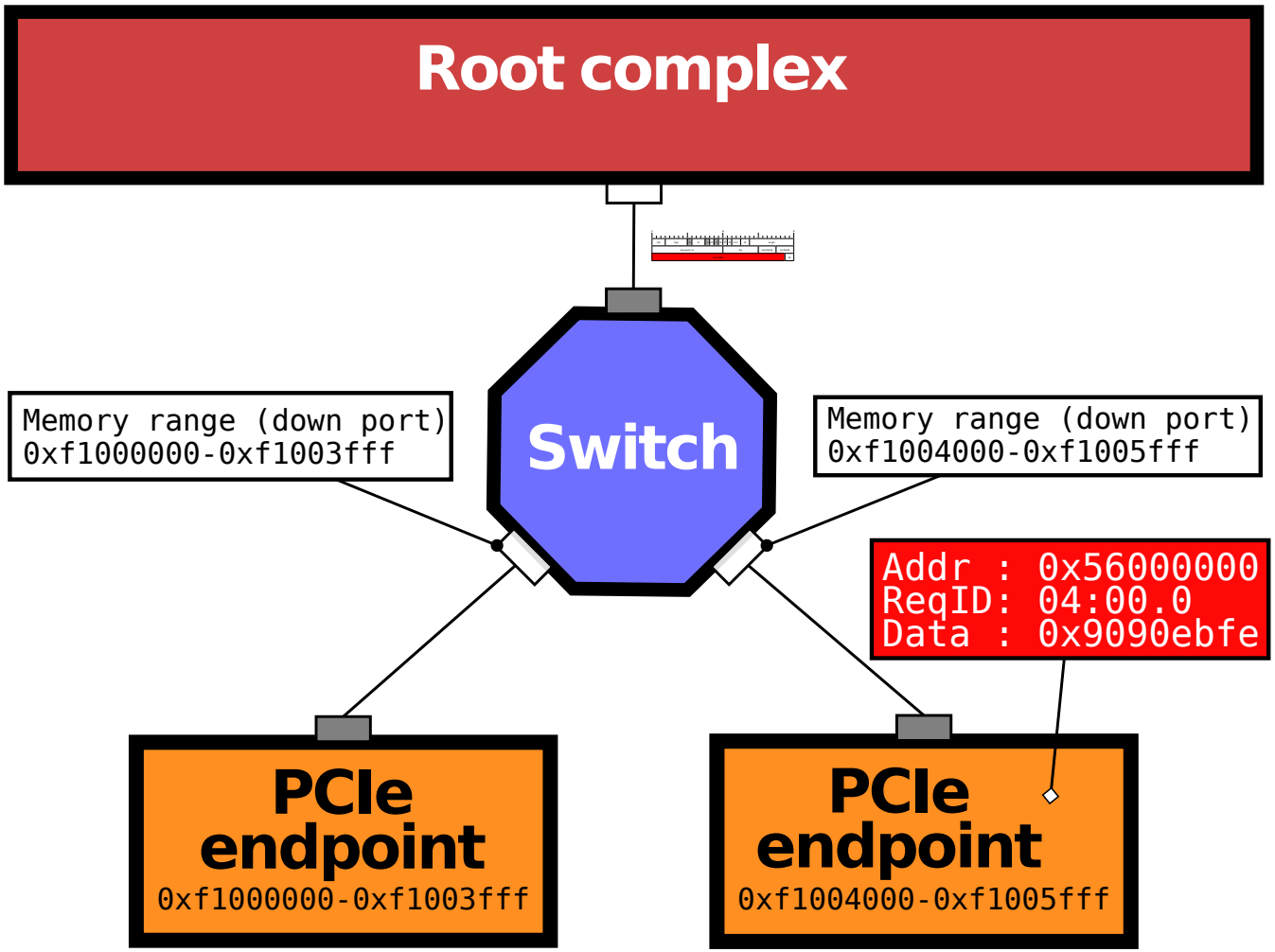
Écriture DMA depuis un périphérique

PCI Express



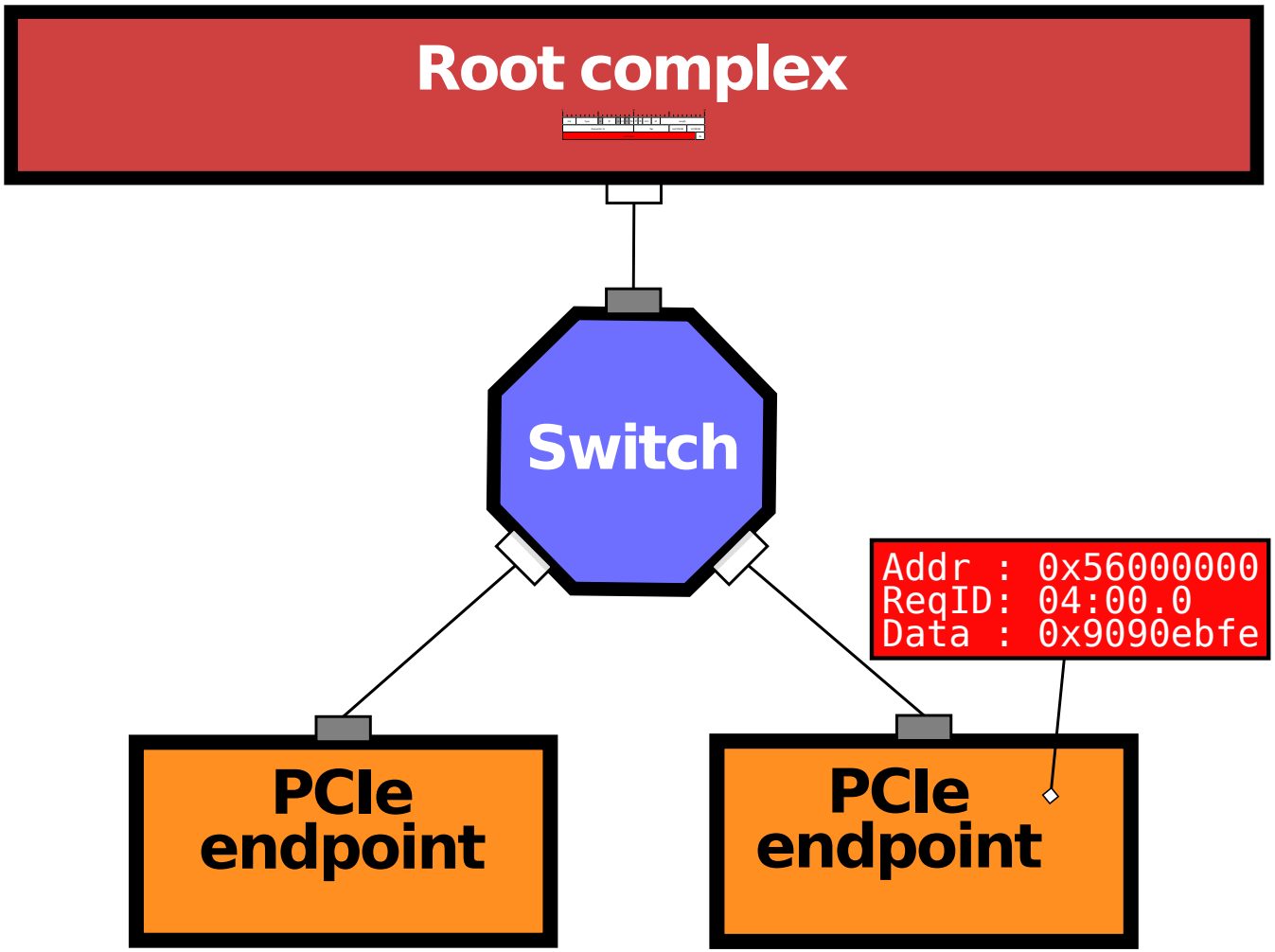
Écriture DMA depuis un périphérique

PCI Express



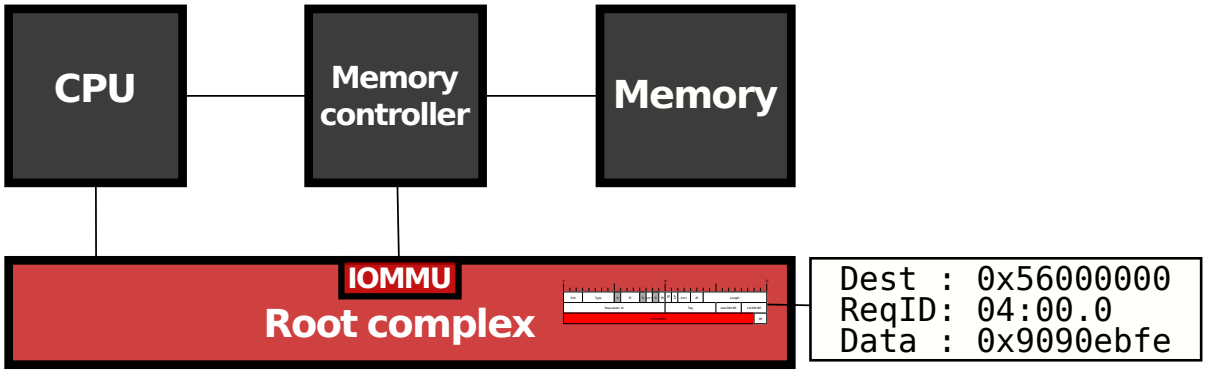
Écriture DMA depuis un périphérique

PCI Express



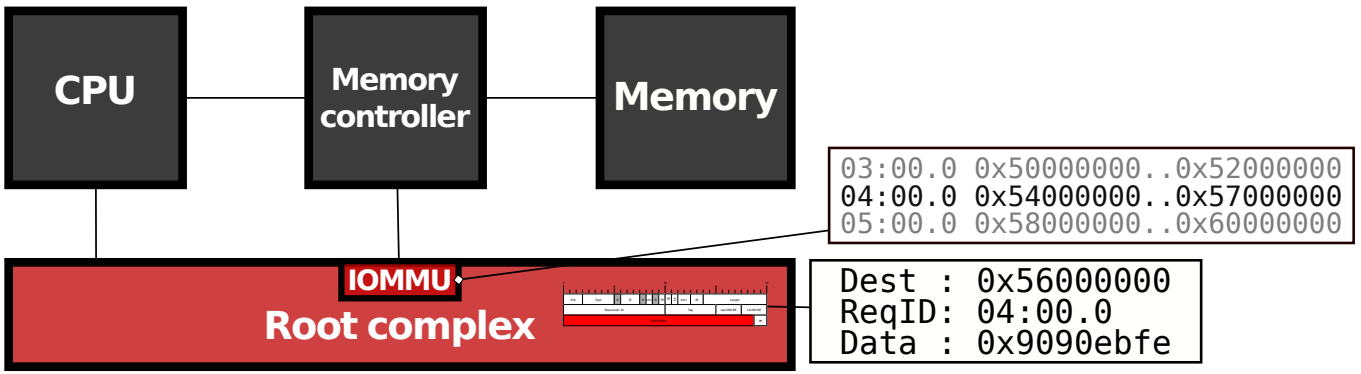
Écriture DMA depuis un périphérique

Sous-système mémoire



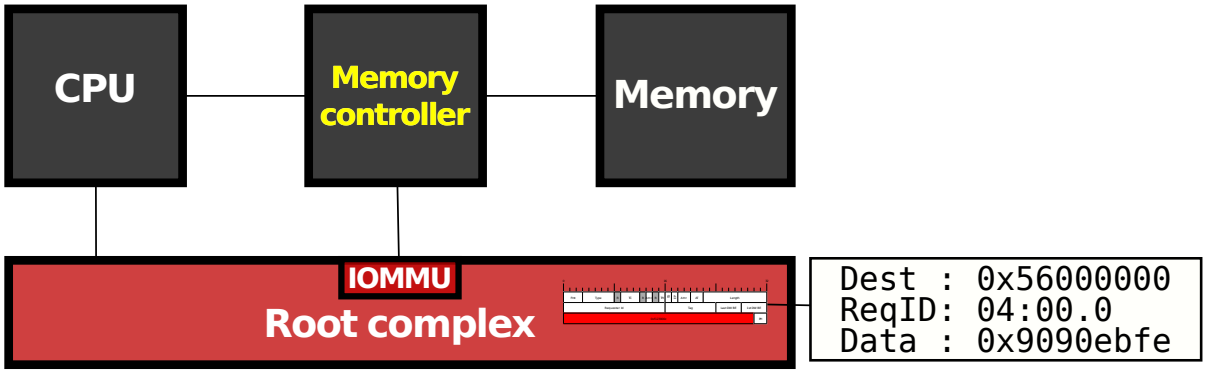
Écriture DMA depuis un périphérique

Sous-système mémoire



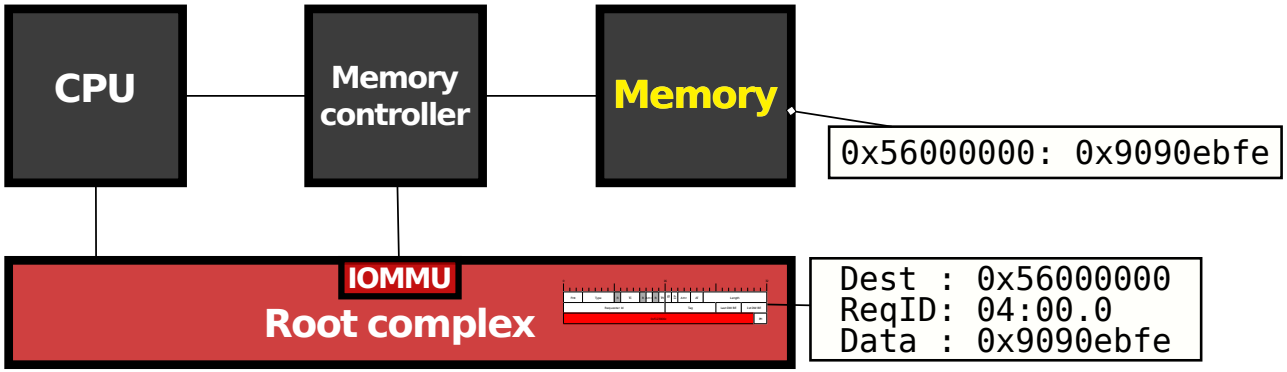
Écriture DMA depuis un périphérique

Sous-système mémoire



Écriture DMA depuis un périphérique

Sous-système mémoire



Section 4

ProTIP : un ami qui vous veut du bien

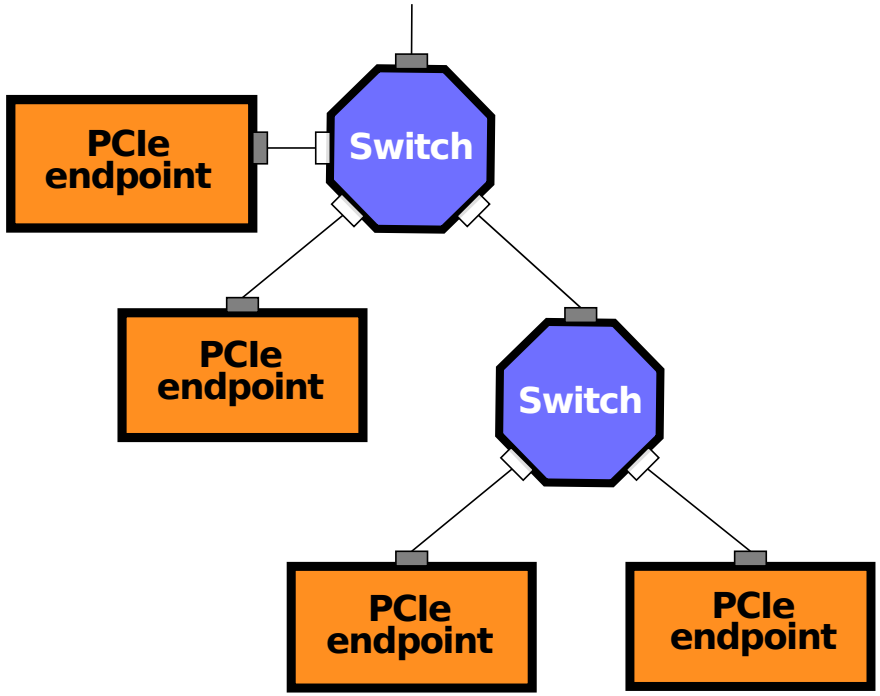
But de ProTIP : caractériser la connectivité réelle

Les TLP sont

- générés
- transmis
- acceptés

par les composants.

? Quel composant peut faire accepter un paquet à quel autre ?



ProTIP : Prolog **T**ester for **I**nformation Flow in **P**CIe networks

En quoi Prolog peut-il nous aider ? (1)

Extrait du fichier mario.pl :

```
friend(yoshi,mario).
friend(peach,mario).
friend(wario,bowser).
brother(luigi,mario).
good(mario).
good(OtherCharacter) :- friend(OtherCharacter,Character),
                                                good(Character).
```

En quoi Prolog peut-il nous aider ? (1)

Extrait du fichier mario.pl :

```
friend(yoshi,mario).
friend(peach,mario).
friend(wario,bowser).
brother(luigi,mario).
good(mario).
good(OtherCharacter) :- friend(OtherCharacter,Character),
                                                good(Character).
```

Utiliser le moteur d'inférence basé sur l'unification (SWI-Prolog en CLI).

```
?- consult([mario]).
true.
```

En quoi Prolog peut-il nous aider ? (1)

Extrait du fichier mario.pl :

```
friend(yoshi,mario).
friend(peach,mario).
friend(wario,bowser).
brother(luigi,mario).
good(mario).
good(OtherCharacter) :- friend(OtherCharacter,Character),
                                                good(Character).
```

Utiliser le moteur d'inférence basé sur l'unification (SWI-Prolog en CLI).

```
?- consult([mario]).
true.
?- friend(X,mario).
```

En quoi Prolog peut-il nous aider ? (1)

Extrait du fichier mario.pl :

```
friend(yoshi,mario).
friend(peach,mario).
friend(wario,bowser).
brother(luigi,mario).
good(mario).
good(OtherCharacter) :- friend(OtherCharacter,Character),
                                                good(Character).
```

Utiliser le moteur d'inférence basé sur l'unification (SWI-Prolog en CLI).

```
?- consult([mario]).
true.
?- friend(X,mario).
X = yoshi
```

En quoi Prolog peut-il nous aider ? (1)

Extrait du fichier mario.pl :

```
friend(yoshi,mario).
friend(peach,mario).
friend(wario,bowser).
brother(luigi,mario).
good(mario).
good(OtherCharacter) :- friend(OtherCharacter,Character),
                                                good(Character).
```

Utiliser le moteur d'inférence basé sur l'unification (SWI-Prolog en CLI).

```
?- consult([mario]).
true.
?- friend(X,mario).
X = yoshi  ;
X = peach.
```

En quoi Prolog peut-il nous aider ? (1)

Extrait du fichier mario.pl :

```
friend(yoshi,mario).
friend(peach,mario).
friend(wario,bowser).
brother(luigi,mario).
good(mario).
good(OtherCharacter) :- friend(OtherCharacter,Character),
                                                good(Character).
```

Utiliser le moteur d'inférence basé sur l'unification (SWI-Prolog en CLI).

```
?- good(X).
```

En quoi Prolog peut-il nous aider ? (1)

Extrait du fichier mario.pl :

```
friend(yoshi,mario).
friend(peach,mario).
friend(wario,bowser).
brother(luigi,mario).
good(mario).
good(OtherCharacter) :- friend(OtherCharacter,Character),
                        good(Character).
```

Utiliser le moteur d'inférence basé sur l'unification (SWI-Prolog en CLI).

```
?- good(X).
X = mario
```

En quoi Prolog peut-il nous aider ? (1)

Extrait du fichier mario.pl :

```
friend(yoshi,mario).
friend(peach,mario).
friend(wario,bowser).
brother(luigi,mario).
good(mario).
good(OtherCharacter) :- friend(OtherCharacter,Character),
                        good(Character).
```

Utiliser le moteur d'inférence basé sur l'unification (SWI-Prolog en CLI).

```
?- good(X).
X = mario  ;
X = yoshi
```

En quoi Prolog peut-il nous aider ? (1)

Extrait du fichier mario.pl :

```
friend(yoshi,mario).
friend(peach,mario).
friend(wario,bowser).
brother(luigi,mario).
good(mario).
good(OtherCharacter) :- friend(OtherCharacter,Character),
                        good(Character).
```

Utiliser le moteur d'inférence basé sur l'unification (SWI-Prolog en CLI).

```
?- good(X).
X = mario  ;
X = yoshi  ;
X = peach
```

En quoi Prolog peut-il nous aider ? (1)

Extrait du fichier mario.pl :

```
friend(yoshi,mario).
friend(peach,mario).
friend(wario,bowser).
brother(luigi,mario).
good(mario).
good(OtherCharacter) :- friend(OtherCharacter,Character),
                                                good(Character).
```

Utiliser le moteur d'inférence basé sur l'unification (SWI-Prolog en CLI).

```
?- good(X).
X = mario  ;
X = yoshi  ;
X = peach  ;
false.
```

En quoi Prolog peut-il nous aider ? (2)

Extrait du fichier mario.pl :

```
:- use_module(library(clpfd)).
lifepts(X) :- X in 0..7.
damage(X) :- X in 1..9.
life(mario,X) :- lifepts(X).
damages(bowser,Y) :- damage(Y), Y#<3.
damages(bowser,Y) :- damage(Y), Y#>=7.
mario_survives(X,Y) :- life(mario,X), damages(bowser,Y), Y#=<X.
```

En quoi Prolog peut-il nous aider ? (2)

Extrait du fichier mario.pl :

```
:- use_module(library(clpfd)).
lifepts(X) :- X in 0..7.
damage(X) :- X in 1..9.
life(mario,X) :- lifepts(X).
damages(bowser,Y) :- damage(Y), Y#<3.
damages(bowser,Y) :- damage(Y), Y#>=7.
mario_survives(X,Y) :- life(mario,X), damages(bowser,Y), Y#=<X.
```

Faire des calculs sur des domaines finis (SWI-Prolog avec CLPFD en CLI).

```
?- mario_survives(Life,Damage).
```

En quoi Prolog peut-il nous aider ? (2)

Extrait du fichier mario.pl :

```
:- use_module(library(clpfd)).
lifepts(X) :- X in 0..7.
damage(X) :- X in 1..9.
life(mario,X) :- lifepts(X).
damages(bowser,Y) :- damage(Y), Y#<3.
damages(bowser,Y) :- damage(Y), Y#>=7.
mario_survives(X,Y) :- life(mario,X), damages(bowser,Y), Y#=<X.
```

Faire des calculs sur des domaines finis (SWI-Prolog avec CLPFD en CLI).

```
?- mario_survives(Life,Damage).
Life in 1..7, Life#>=Damage, Damage in 1..2
```

En quoi Prolog peut-il nous aider ? (2)

Extrait du fichier mario.pl :

```
:- use_module(library(clpfd)).
lifepts(X) :- X in 0..7.
damage(X) :- X in 1..9.
life(mario,X) :- lifepts(X).
damages(bowser,Y) :- damage(Y), Y#<3.
damages(bowser,Y) :- damage(Y), Y#>=7.
mario_survives(X,Y) :- life(mario,X), damages(bowser,Y), Y#=<X.
```

Faire des calculs sur des domaines finis (SWI-Prolog avec CLPFD en CLI).

```
?- mario_survives(Life,Damage).
Life in 1..7, Life#>=Damage, Damage in 1..2  ;
Life = Damage, Damage = 7.
```

Modéliser un réseau PCIe en Prolog

Événements formalisés par prédicats :

- génération de paquets (conformes à la spécification) ;
- génération arbitraire de paquets ;
- émission de paquets ;
- réception de paquets ;
- acceptation de paquets.

Passage par l'un des composants = règle Prolog liant cause et conséquence :

```
hop(TriggerEvt, InPort, ResultEv, OutPort)
```

Entrée et sortie de ProTIP

Configurations :

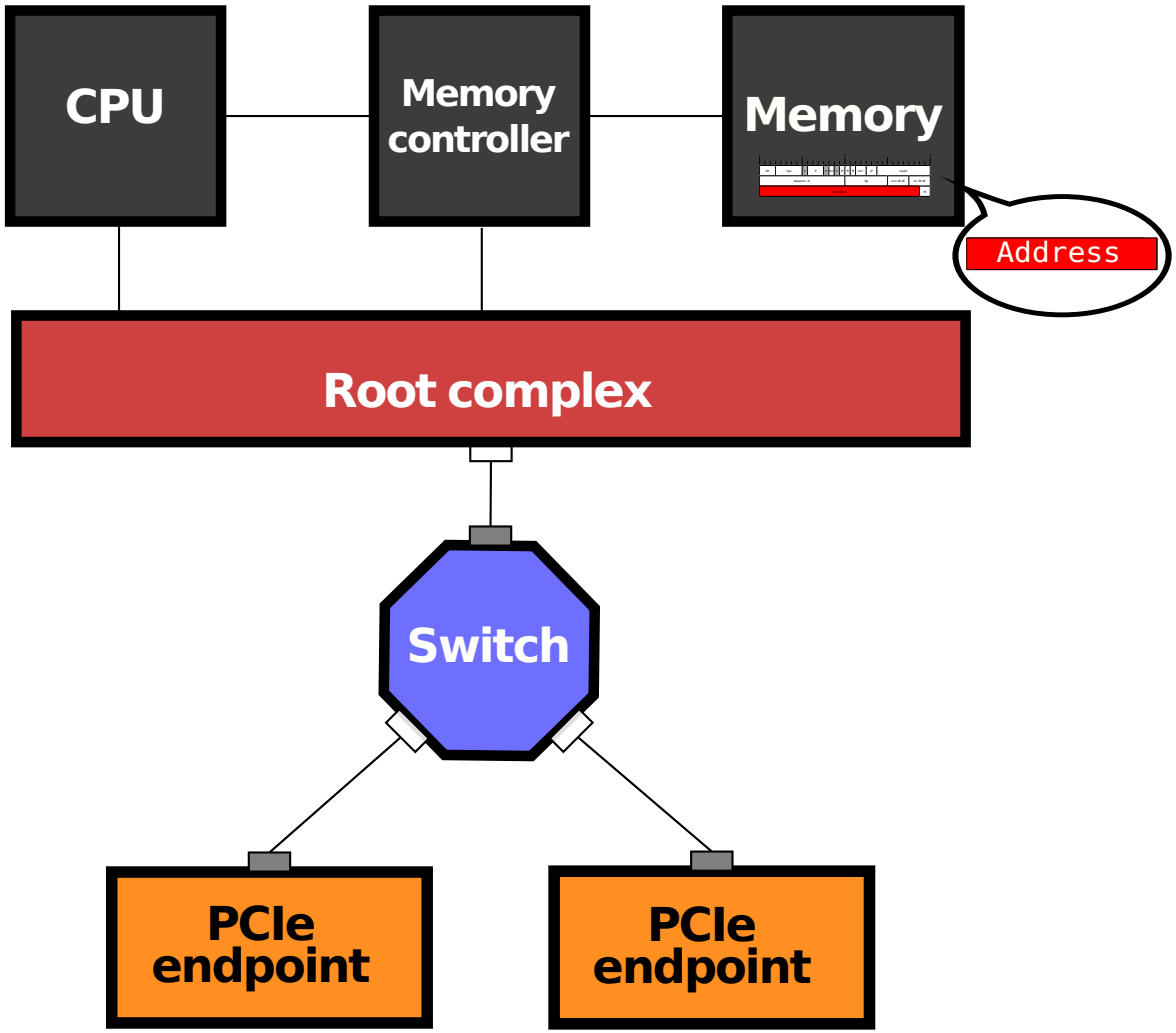
- ports physiques ;
- liens physiques ;
- contenu des registres

MEM, BAR, BUS...

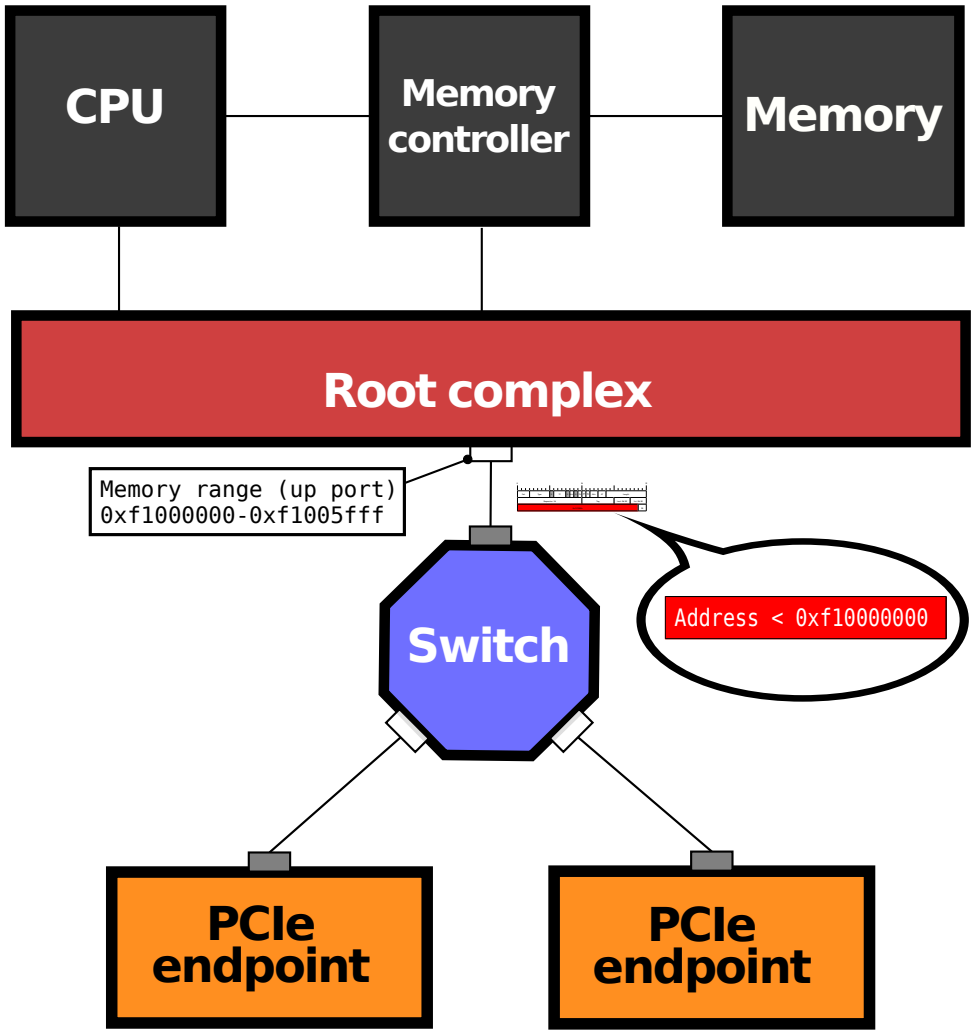
ProTIP

Traces possibles terminant par acceptation.
Backtracking

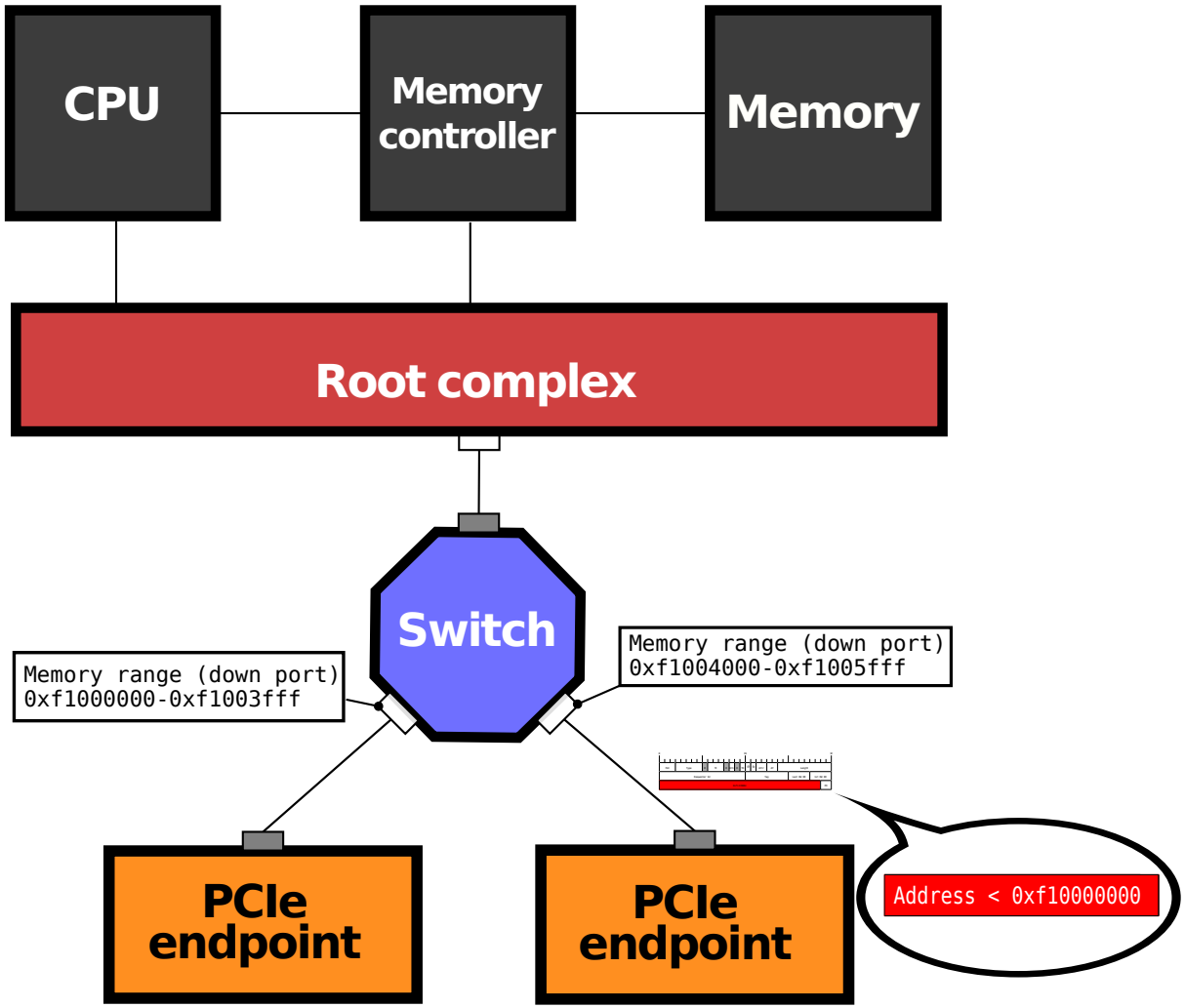
Calcul des traces et résultats (1)



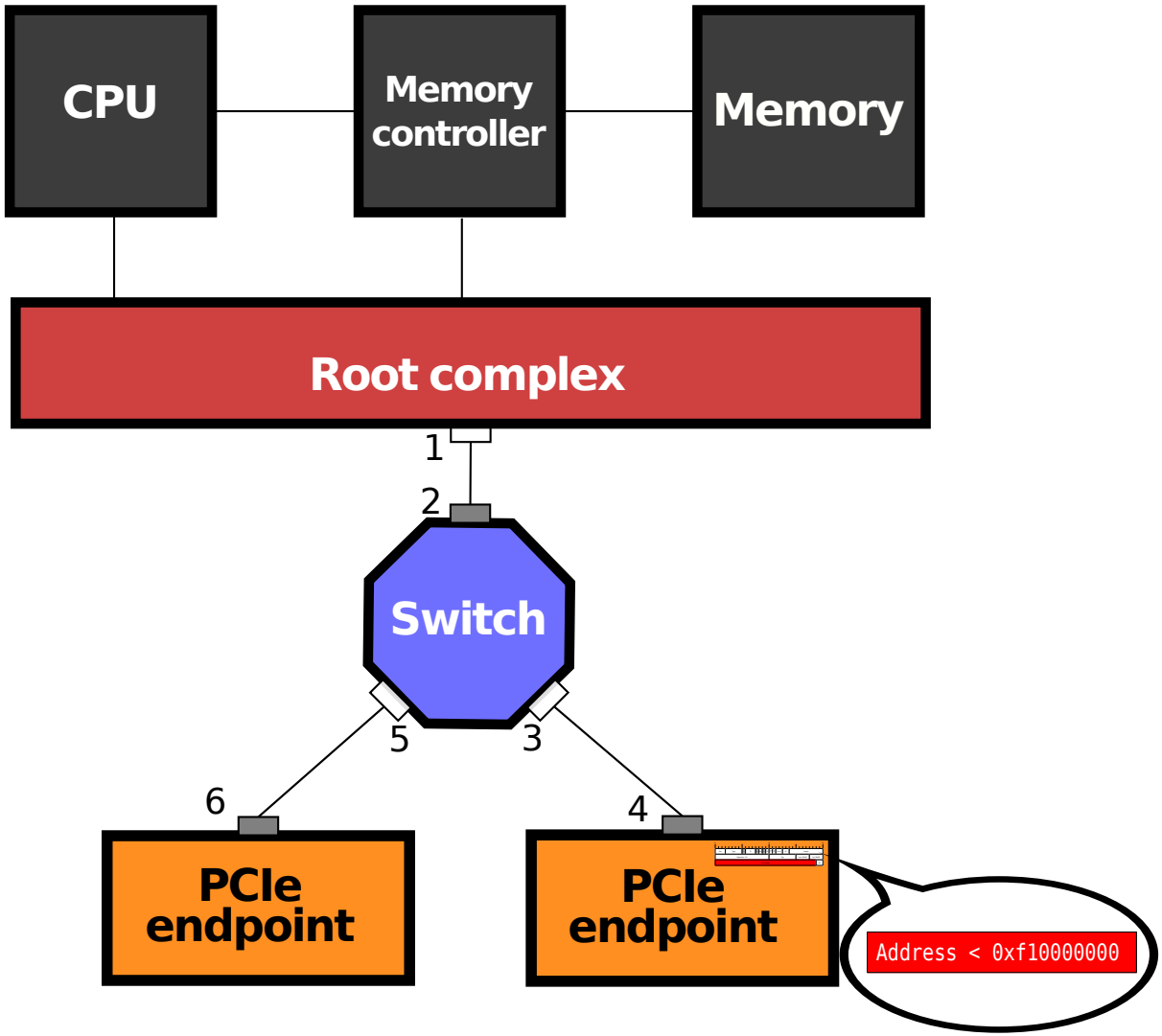
Calcul des traces et résultats (1)



Calcul des traces et résultats (1)



Calcul des traces et résultats (1)

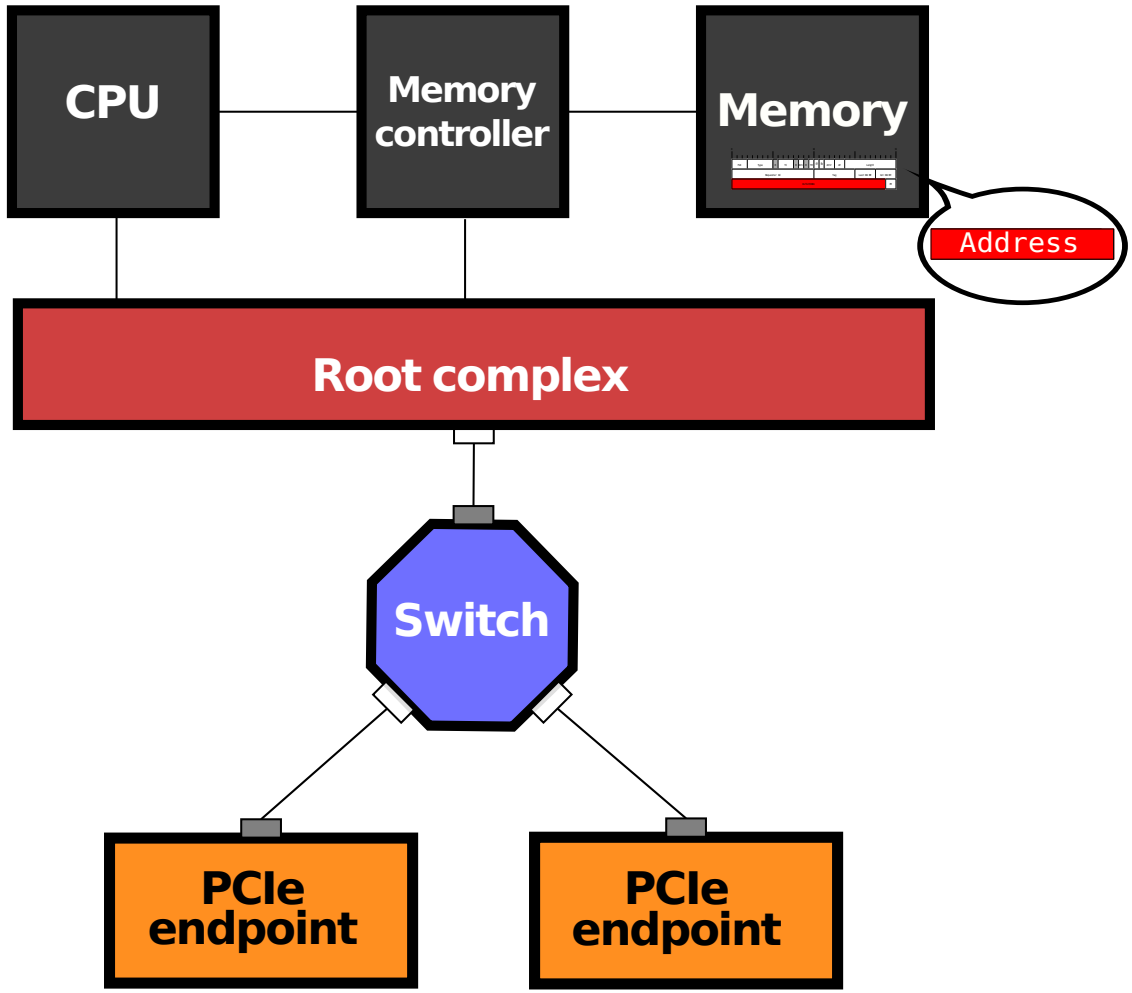


Calcul des traces et résultats

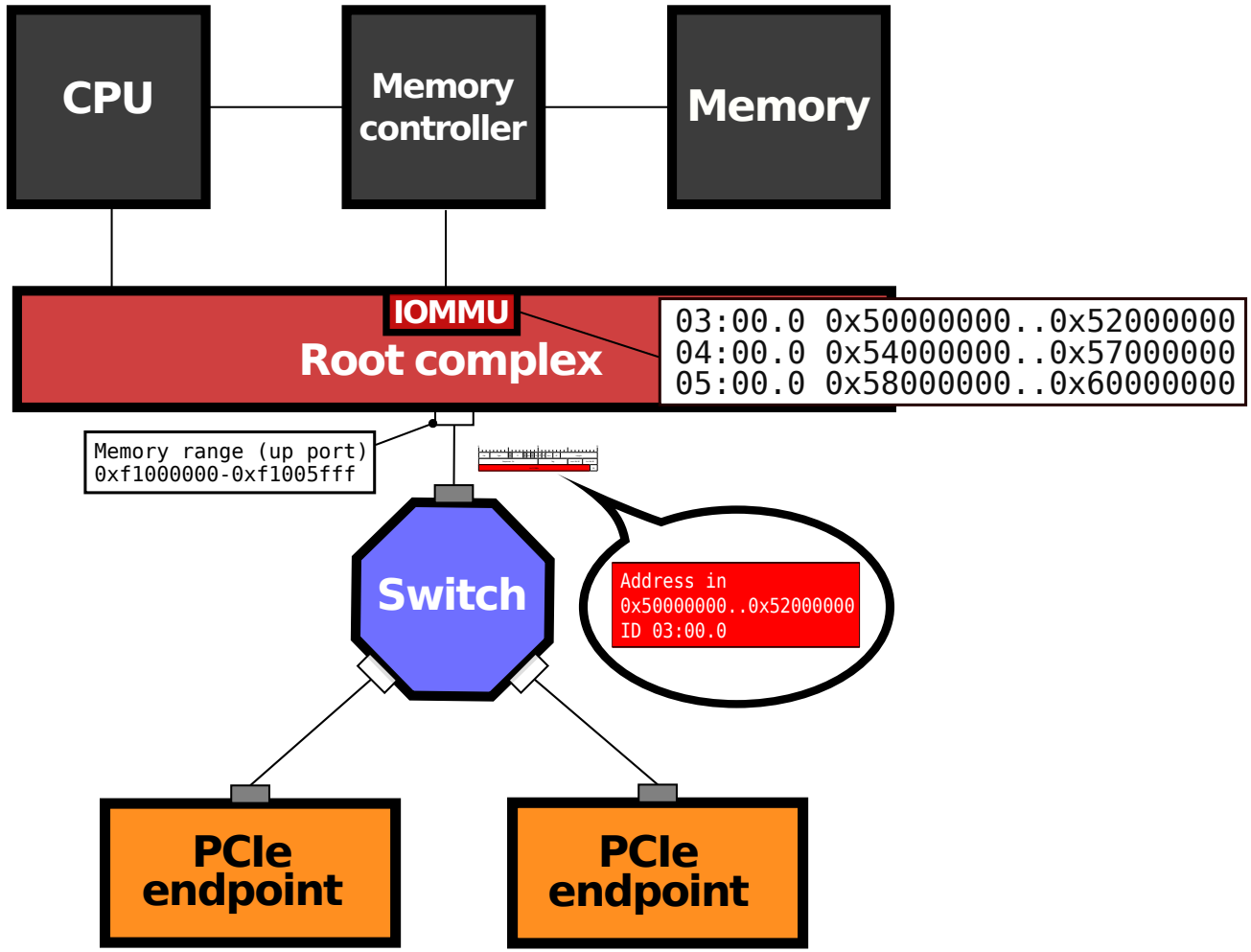
- sans I/OMMU : accès (quasi) arbitraire à la RAM pour un périphérique malveillant.

```
Trace : port ram accepts write with a WEIRD TRACE
generate,4,[[],[],[],[],[],[]]
 -> mem_write(Address,bdf(B,D,F),1),3,[[],[],[],[],[],[]]
 -> mem_write(Address,bdf(B,D,F),1),1,[[],[],[],[],[],[]]
 -> accept(write,Address),ram,[[],[],[],[],[],[]]
and constraints :
clpfd: (Address in 0.. 0xbfffffff), clpfd: (B in 0..256),
clpfd: (D in 0..32), clpfd: (F in 0..8)
```

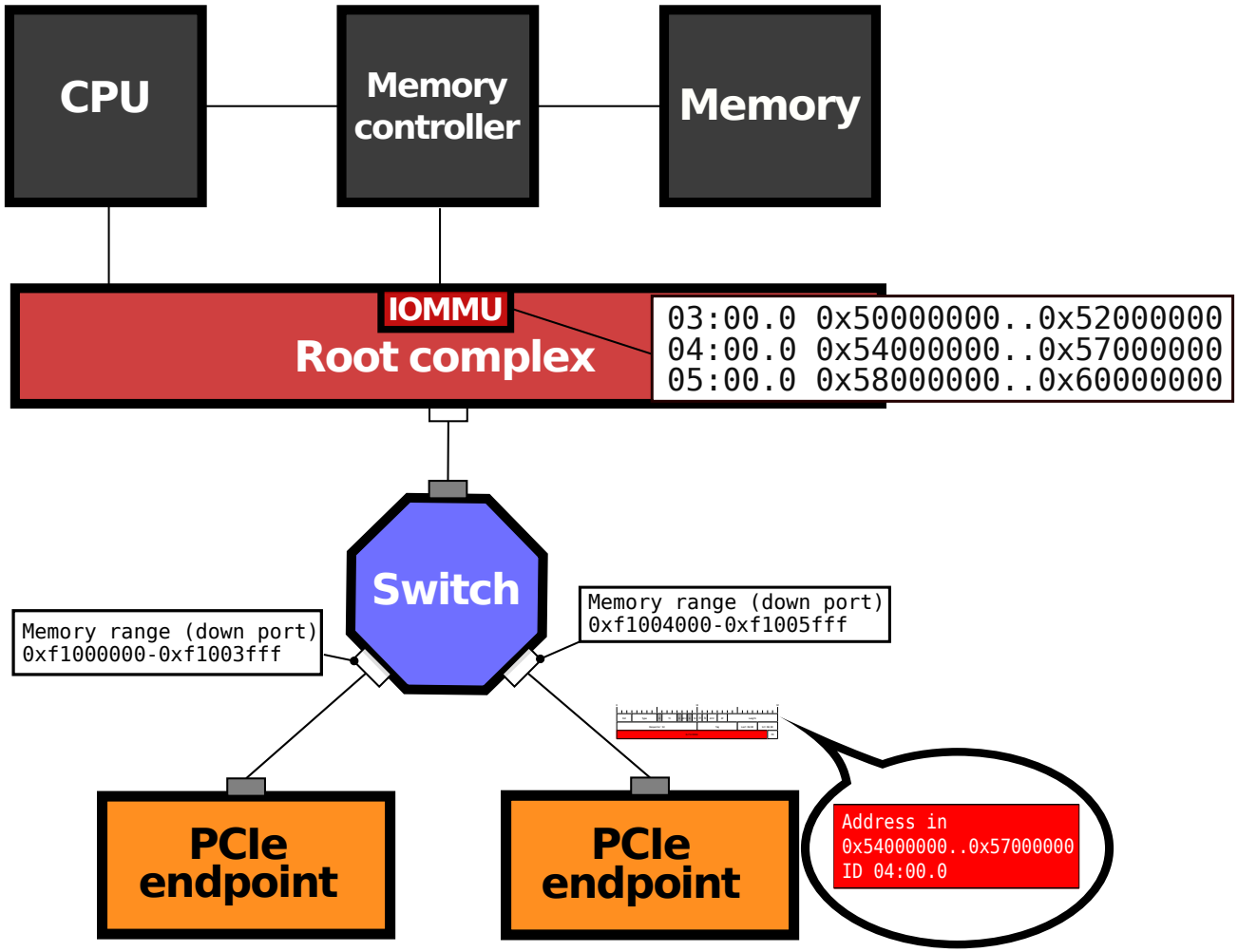
Calcul des traces et résultats (1)



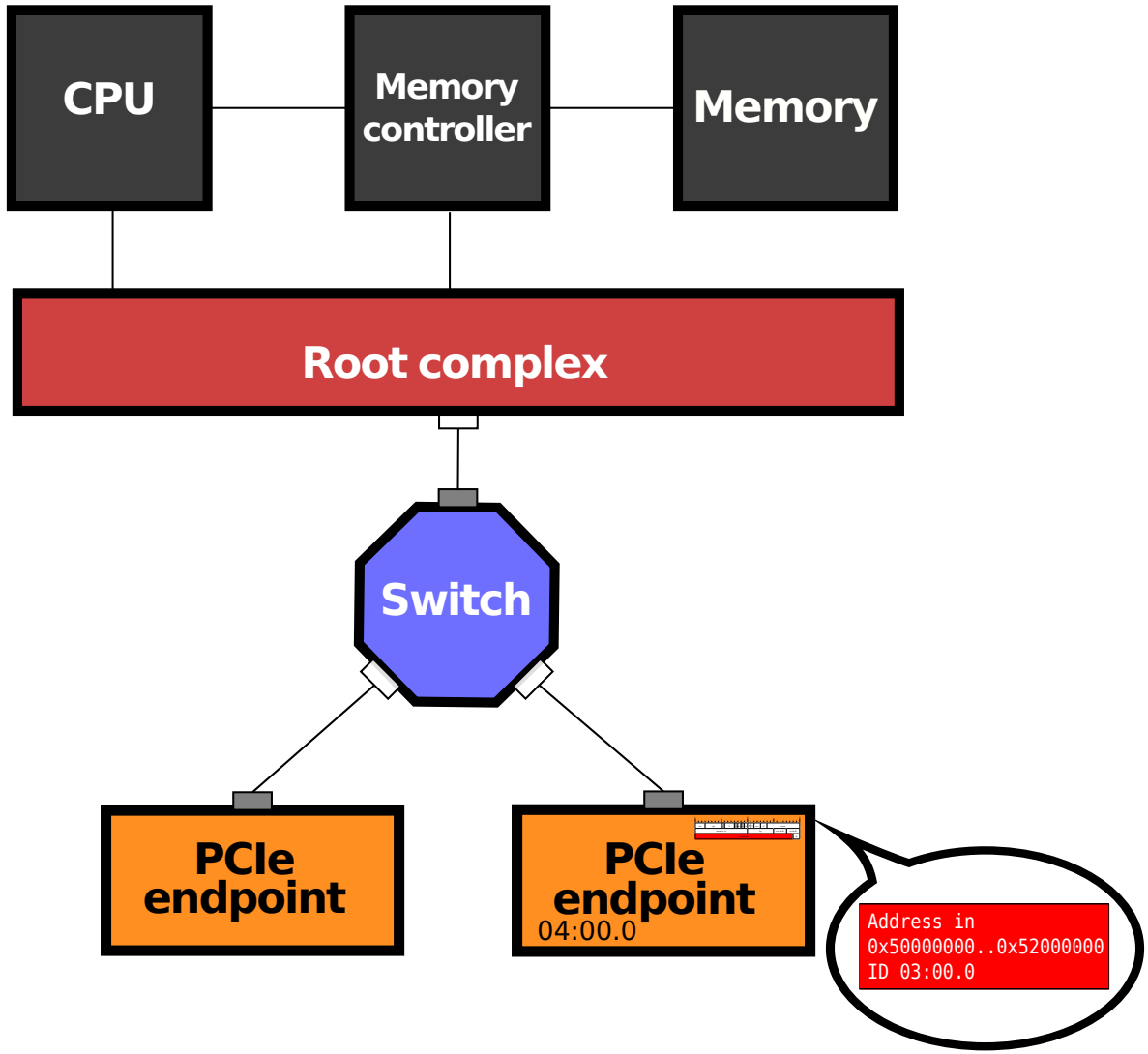
Calcul des traces et résultats (1)



Calcul des traces et résultats (1)



Calcul des traces et résultats (1)



Calcul des traces et résultats

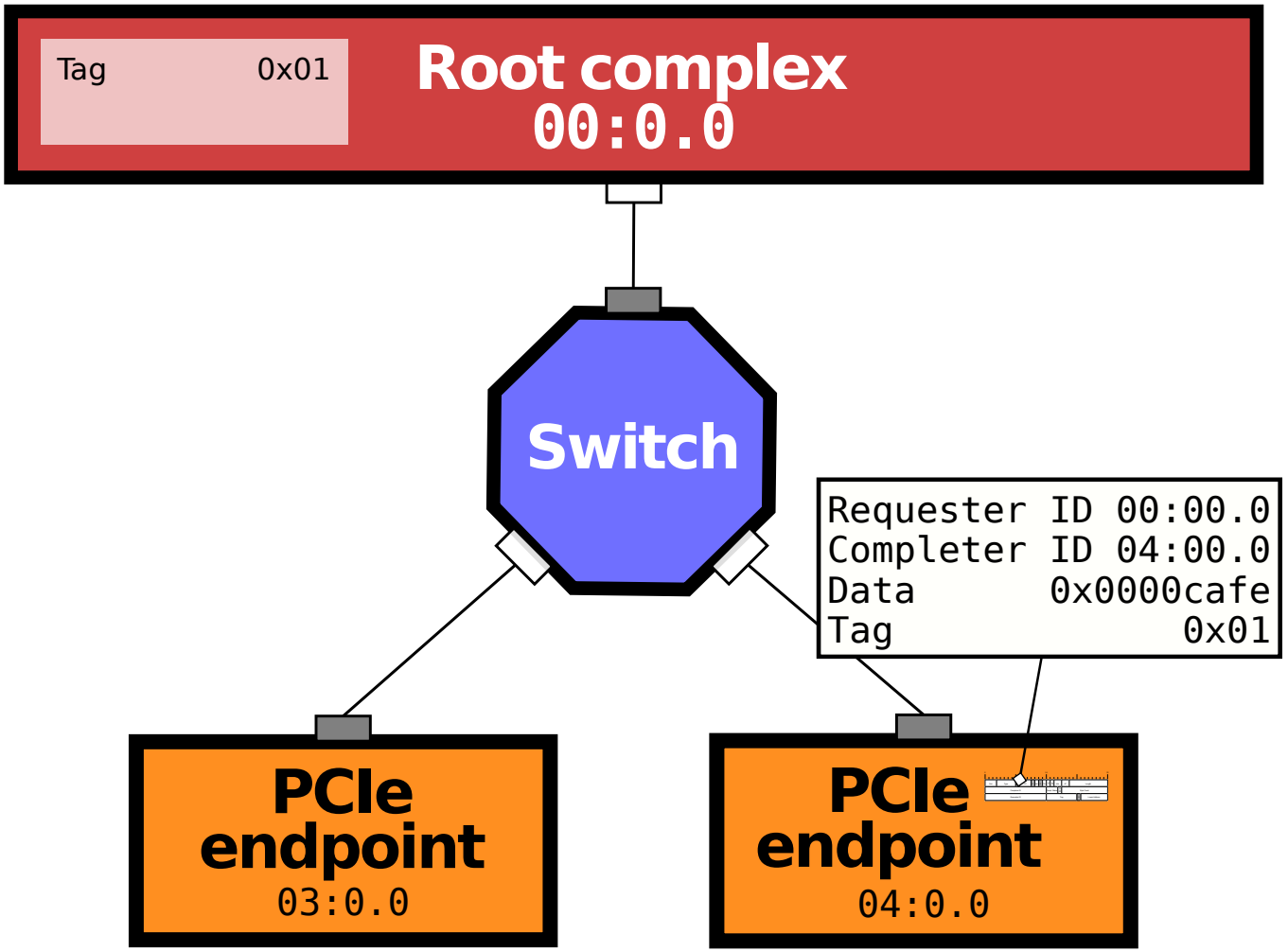
- sans I/OMMU : accès (quasi) arbitraire à la RAM pour un périphérique malveillant ;
- avec I/OMMU : usurpation d'ID configuré dans l'I/OMMU → besoin des ACS.

Tout ceci était déjà documenté dans d'autres travaux (en particulier ceux de F. Lone-Sang).

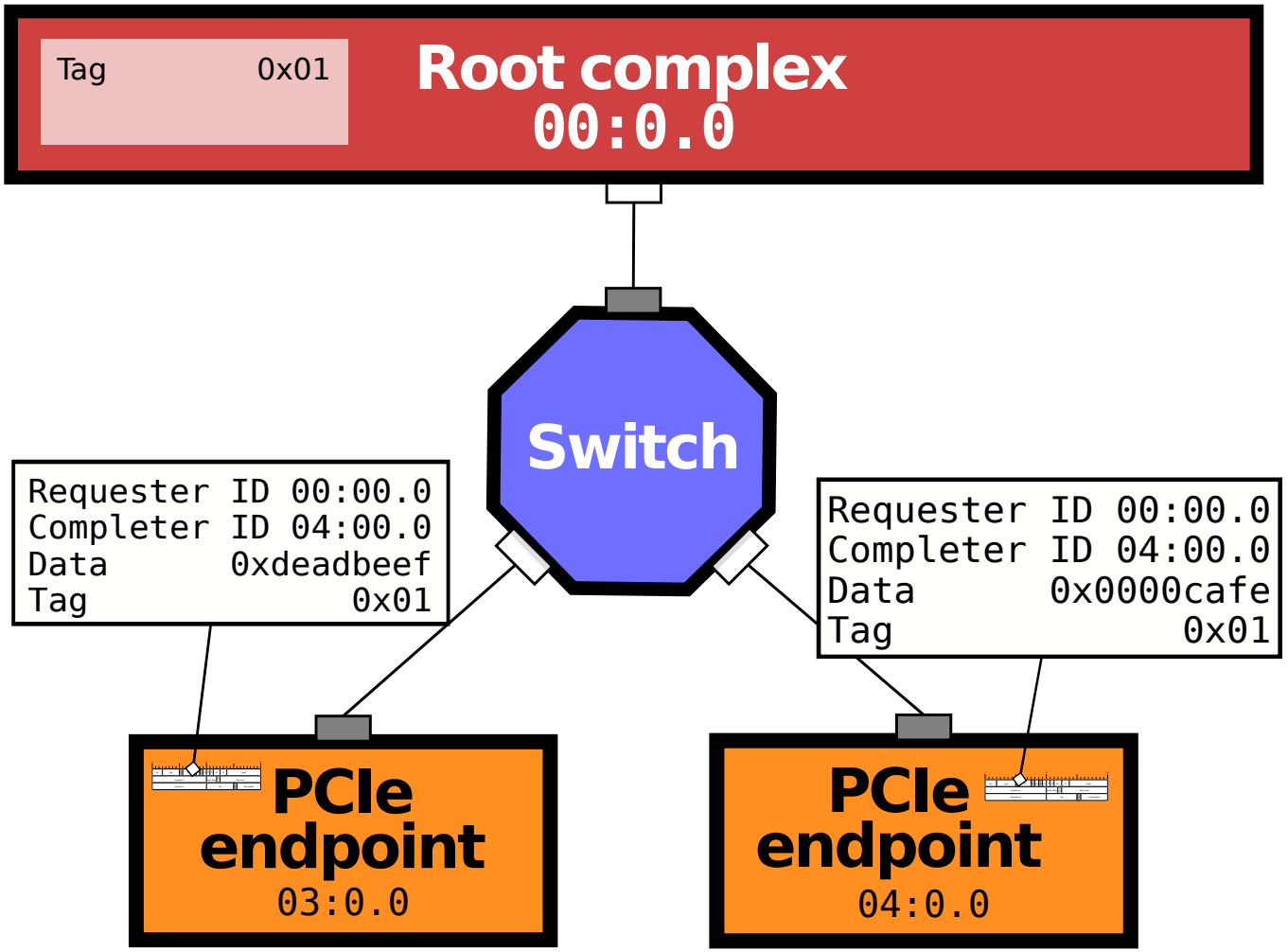
La configuration la moins permissive des ACS permet-elle de

cloisonner les périphériques ?

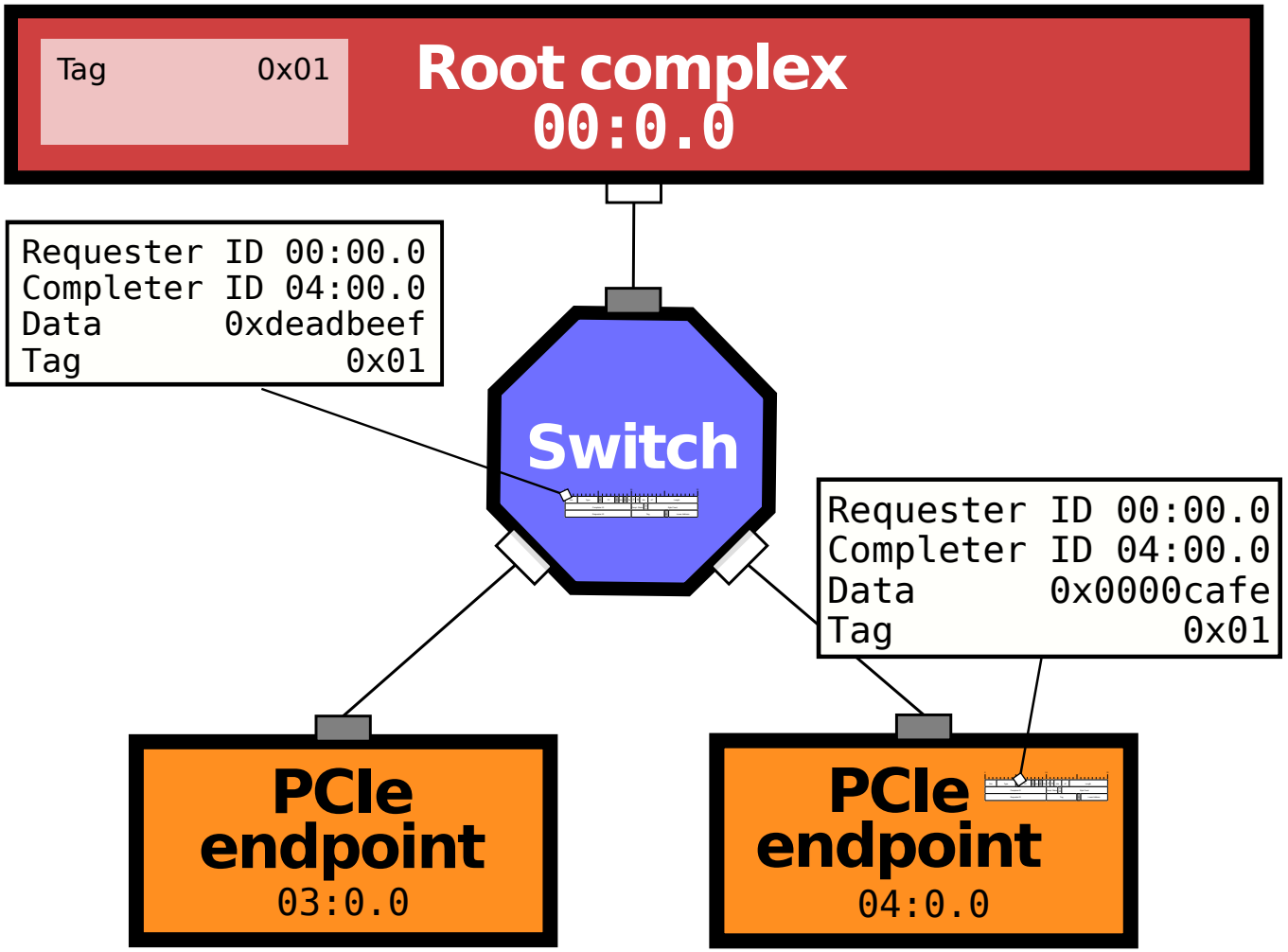
Prendre de vitesse la réponse légitime à une lecture



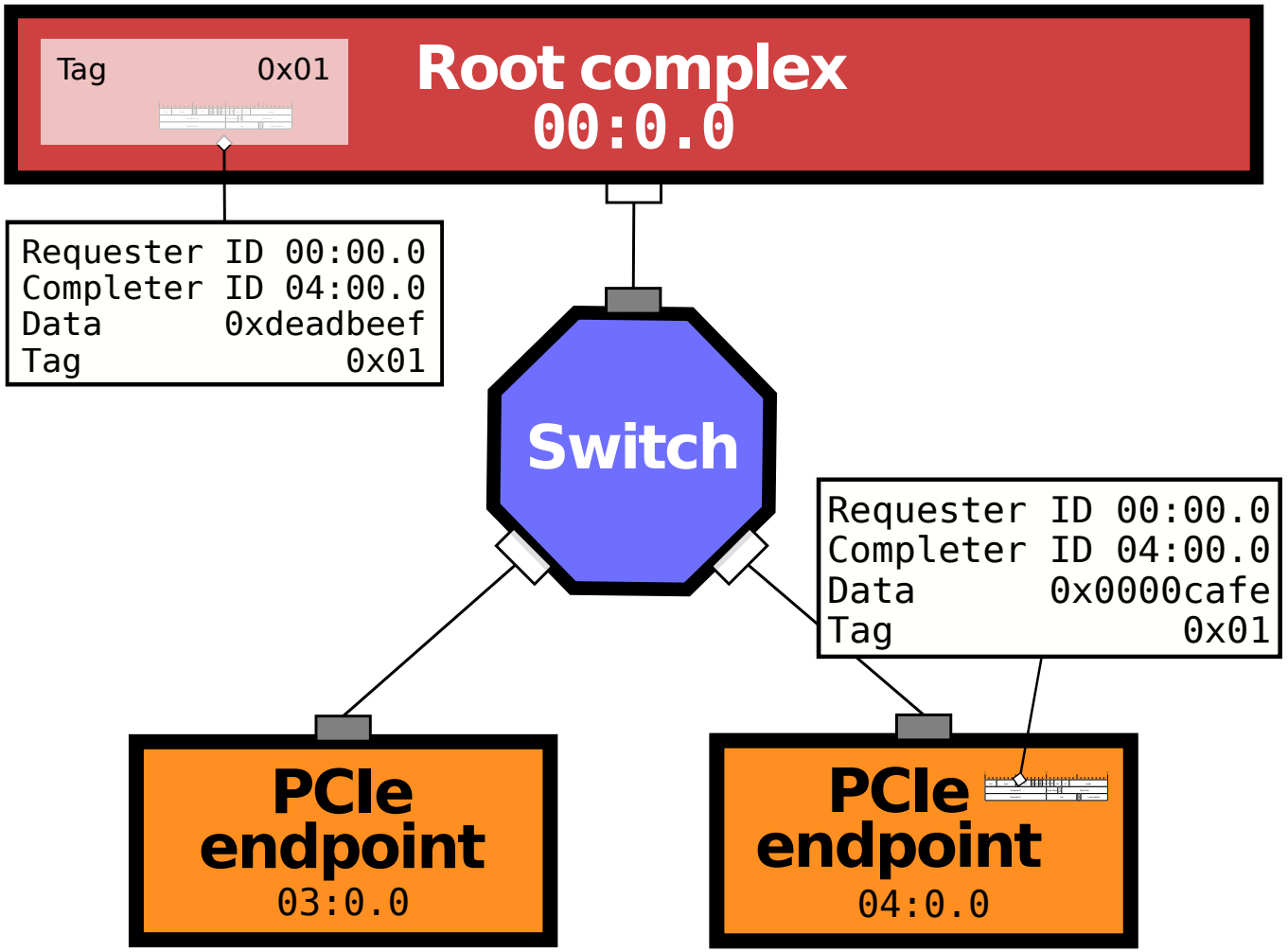
Prendre de vitesse la réponse légitime à une lecture



Prendre de vitesse la réponse légitime à une lecture



Prendre de vitesse la réponse légitime à une lecture



Section 5

Changer d'identité

Parlons de configuration

- émission de requêtes de configuration → beaucoup d'éléments de configuration peuvent varier !
- une suite de requêtes de configurations fait évoluer les identifiants (BDF) de composants...

Parlons de configuration

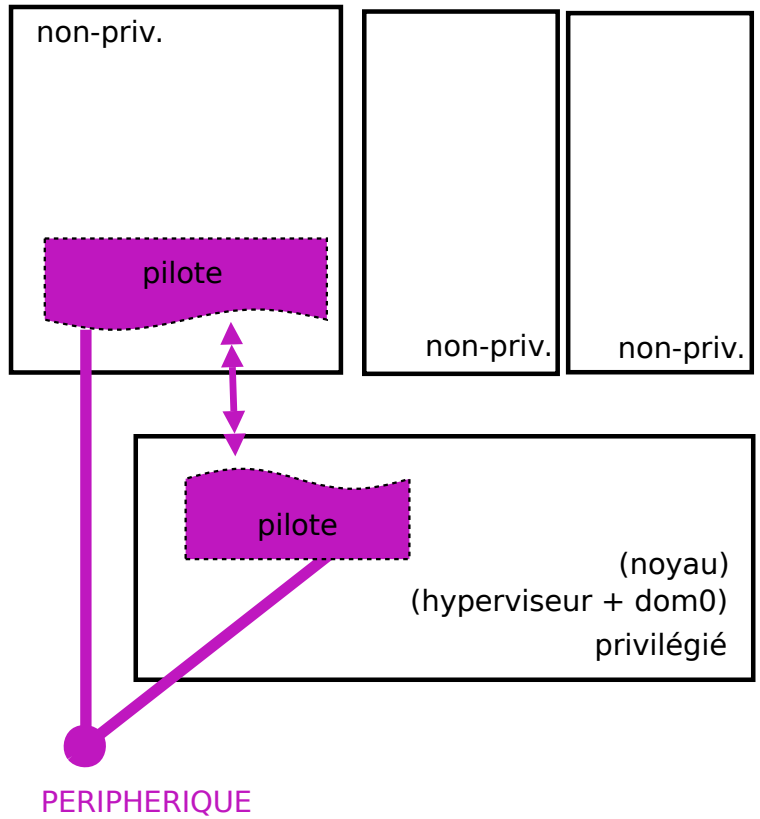
- émission de requêtes de configuration → beaucoup d'éléments de configuration peuvent varier !
- une suite de requêtes de configurations fait évoluer les identifiants (BDF) de composants...
- ... sur lesquels s'appuie l'I/OMMU pour évaluer les droits d'accès.

Quel danger les changements de configuration représentent-ils pour

une implémentation donnée de délégation

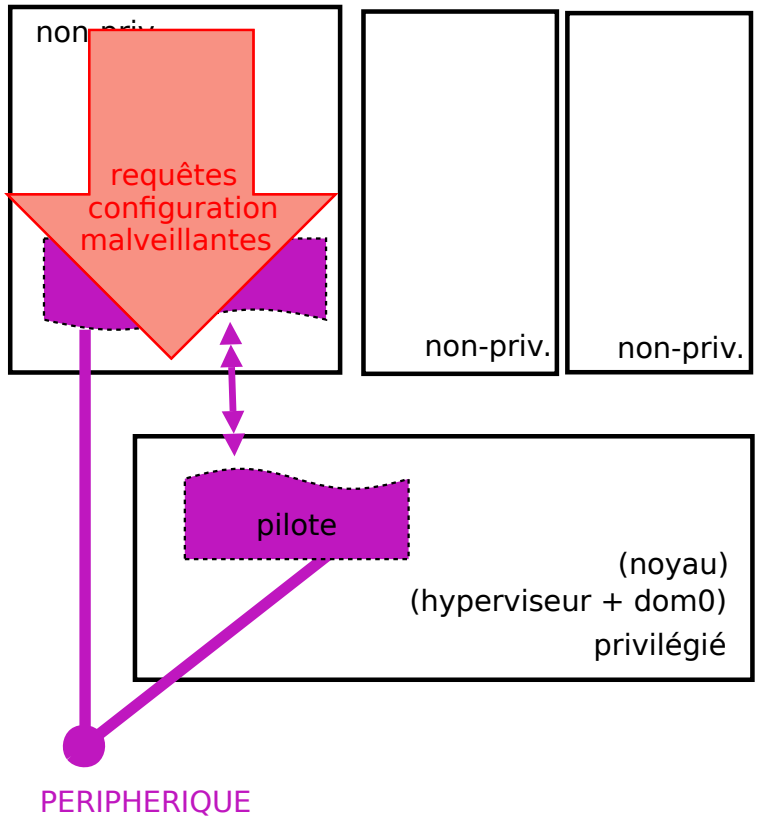
Pour revenir à notre objectif d'évaluation du cloisonnement

- le pilote est divisé en deux morceaux ;



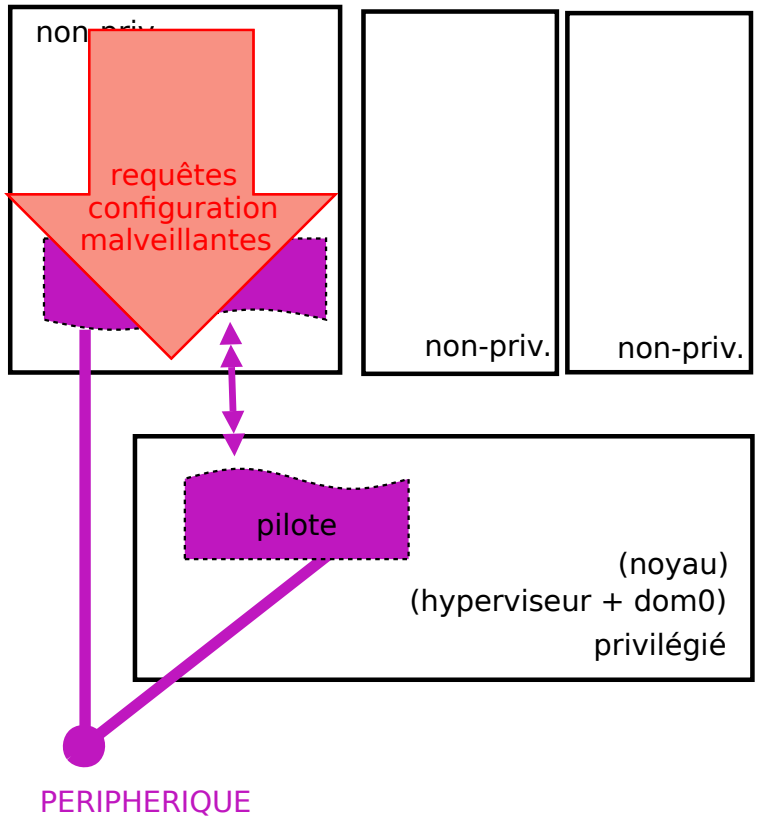
Pour revenir à notre objectif d'évaluation du cloisonnement

- le pilote est divisé en deux morceaux ;
- quelles requêtes émissibles ? Quel changement de configuration possible ? Quel impact ?



Pour revenir à notre objectif d'évaluation du cloisonnement

- le pilote est divisé en deux morceaux ;
- quelles requêtes émissibles ? Quel changement de configuration possible ? Quel impact ?
- ProTIP permet de détecter certaines classes d'attaques.



Section 6

Le fin mot de l'histoire

Si vous ne retenez que ça

- les périphériques malveillants ne sont pas qu'une menace locale ;
- l'I/OMMU ne protège pas de tout (et de loin) ;

Si vous ne retenez que ça

- les périphériques malveillants ne sont pas qu'une menace locale ;
- l'I/OMMU ne protège pas de tout (et de loin) ;
- les ACS ne suffisent pas à garantir l'absence de flux non-désirés ;

Si vous ne retenez que ça

- les périphériques malveillants ne sont pas qu'une menace locale ;
- l'I/OMMU ne protège pas de tout (et de loin) ;
- les ACS ne suffisent pas à garantir l'absence de flux non-désirés ;
- approche systématique nécessaire pour caractériser la connectivité réelle entre composants ;
- ProTIP constitue le début de la mise en œuvre de cette approche.

Merci !

ProTIP est un outil opensource disponible ici :
`https://github.com/ANSSI-FR/ProTIP`

Preuve expérimentale : le dispositif (1)

PCILeech :
permet d'utiliser un composant PCIe inséré dans un système cible pour lire/écrire dans la mémoire de la cible sans pilote.
Développé par Ulf Frisk :
https ://github.com/ufrisk/pcileech

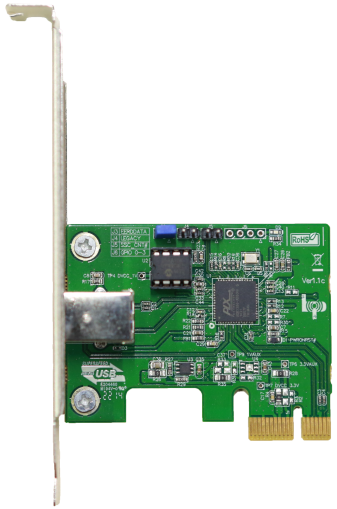


FIGURE – PLX USB-3380

Preuve expérimentale : le dispositif (2)