

Socket TCP/IP

IP, Protocole internet

Responsable de l'acheminement des paquets. Pas de garantie l'acheminement des paquets.

IP, Protocole internet

< ----- 32 bits ----- >				
Version (4 bits)	Longueur d'en-tête (4 bits)	Type de service (8 bits)	Longueur totale (16 bits)	
Identification (16 bits)			Drapeau (3 bits)	Décalage fragment (13 bits)
Durée de vie (8 bits)	Protocole (8 bits)		Somme de contrôle en-tête (16 bits)	
Adresse IP source (32 bits)				
Adresse IP destination (32 bits)				
Données				

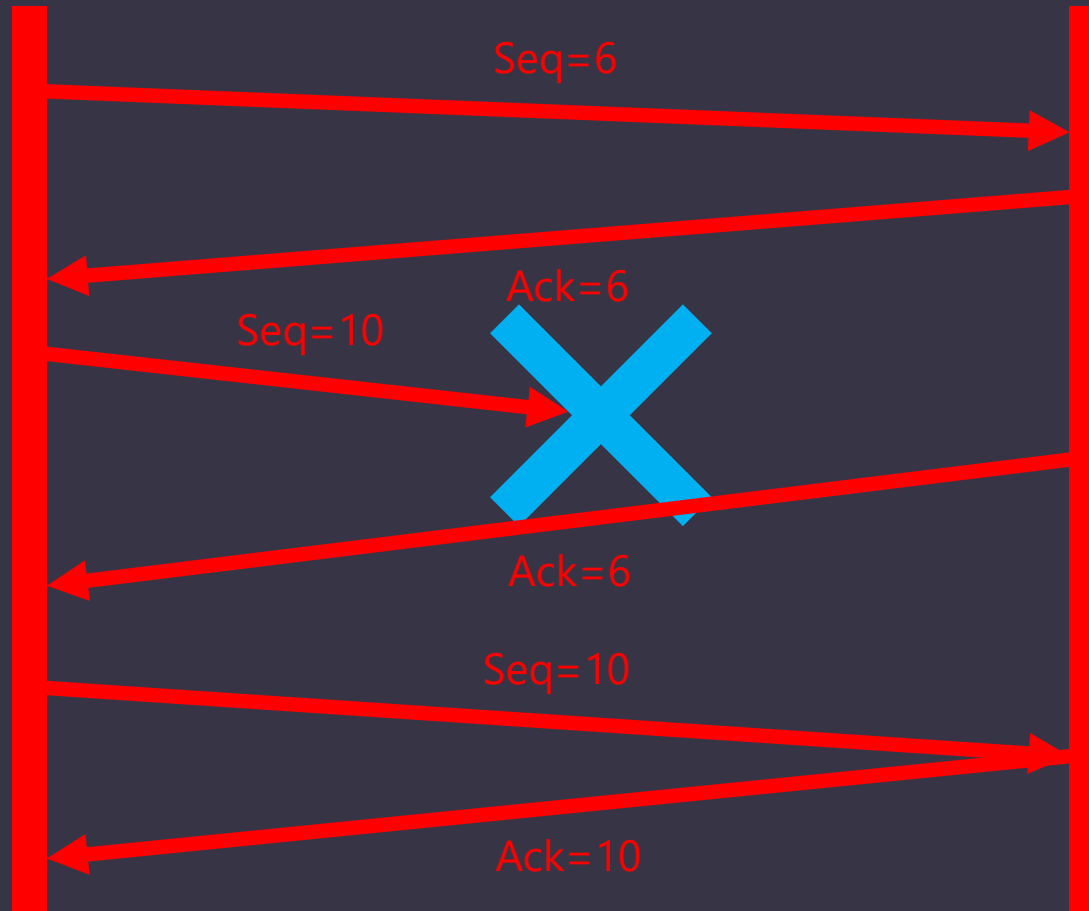
TCP ou Transmission Control protocole

Protocole construit par dessus le protocole IP. Permet de garantir l'acheminement des paquets via la retransmission ainsi que l'ordre des remises. Le TCP ajoute beaucoup d'information : le port source, le port de destination, le numéro de séquence, le numéro d'accusé réception(acknowledgement) et bien d'autres.

Le port

Le port permet de savoir le paquet est destiné à quelle application.

Séquence



L'utilisation de socket TCP/IP

Les sockets peuvent permettre une communication entre deux machines. Les sockets peuvent également servir à deux processus ou programme différents de se parler sur un même ordinateur.

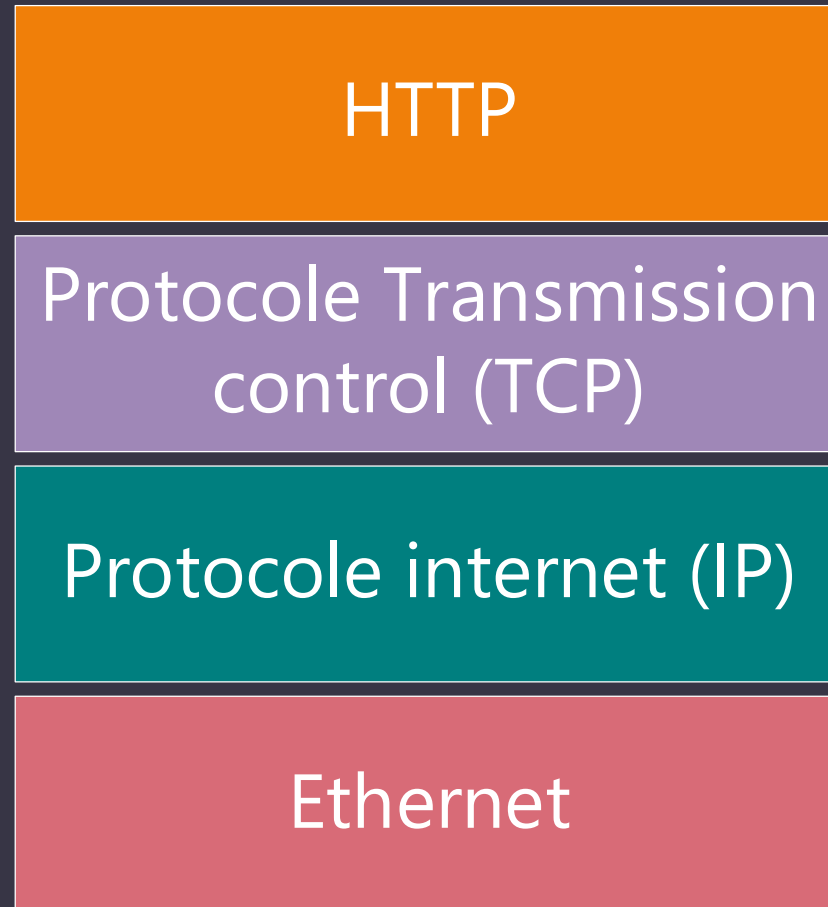
Les avantages

- Grande rapidité
- Désigné pour le temps réel
- Possibilité de flux (Stream)
- Flexible
- Bidirectionnel

Les désavantages

- Implémentation d'un code client et serveur
- Investissement de temps considérable pour mettre en place
- Flexibilité pas toujours nécessaire
- Nécessite souvent l'utilisation de Thread

Couche réseau exemple HTTP



Les couches

Protocole fait maison

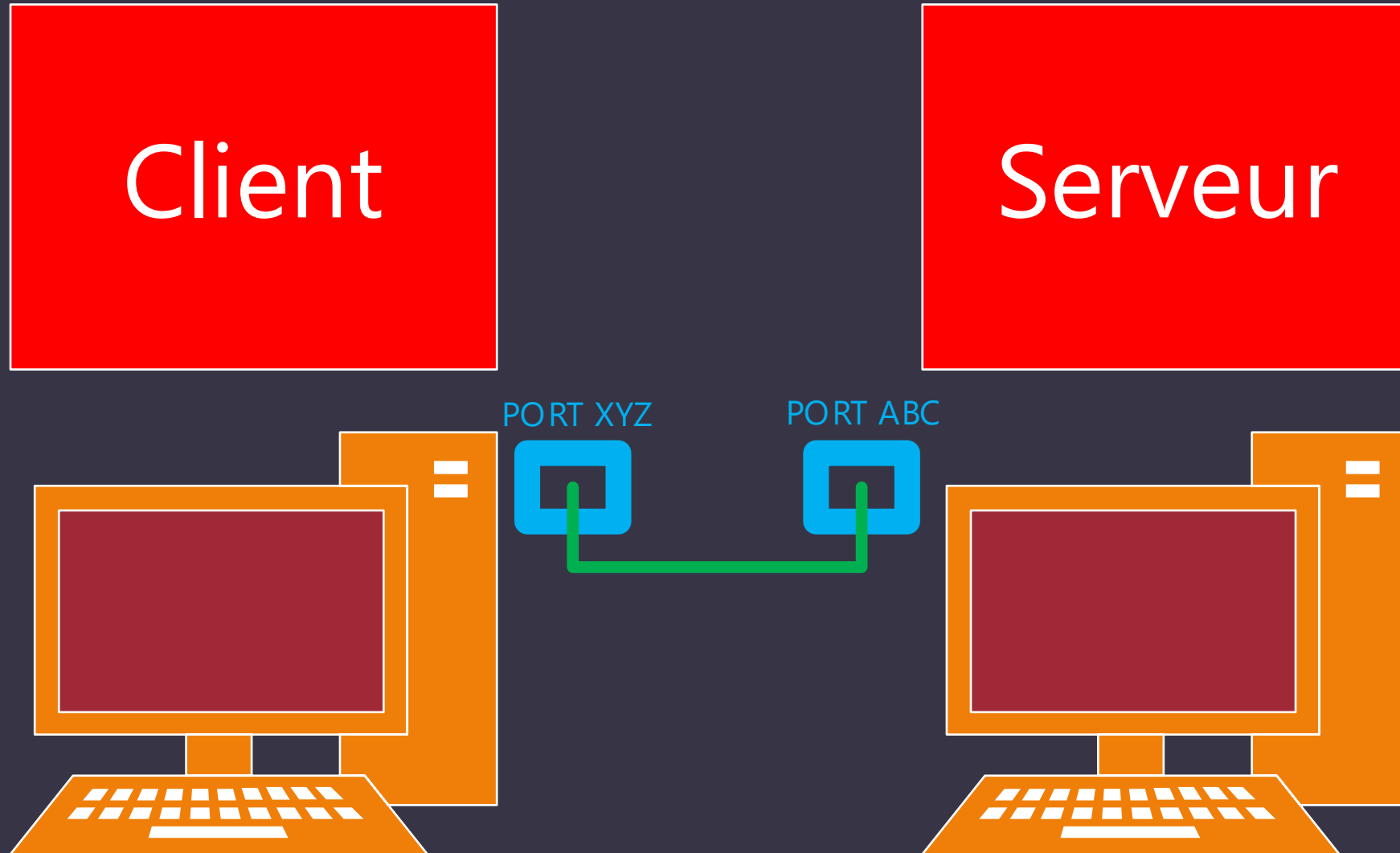
Protocole Transmission
control (TCP)

Protocole internet (IP)

Ethernet

On vient nous même
faire la gestion des
couches Session,
Présentation et
application dans ce
cas.

Client serveur & Port



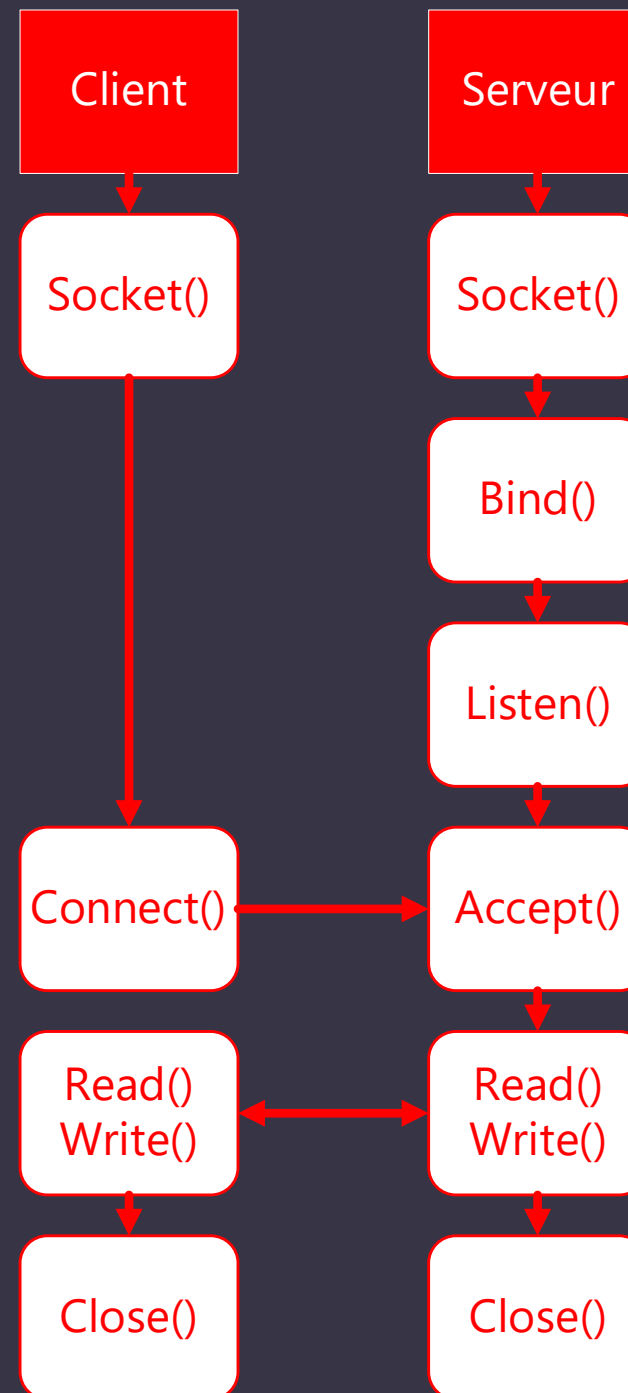
Socket

C'est la combinaison d'un IP et d'un port.

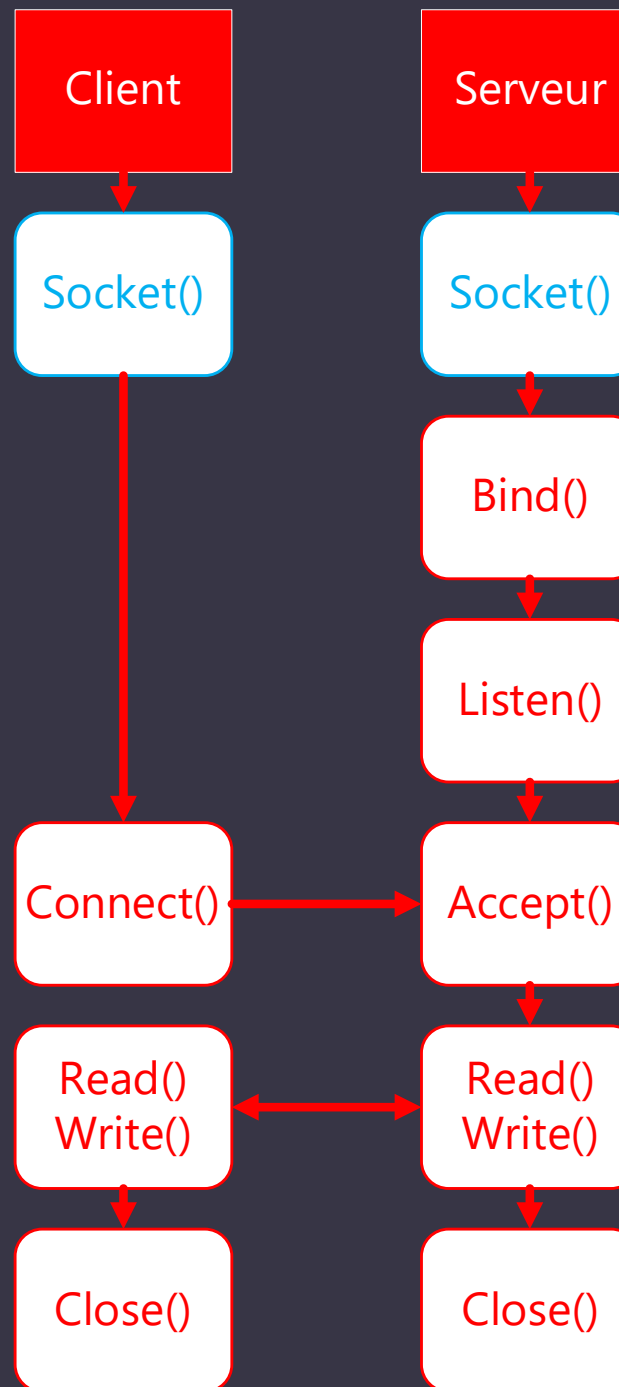
Caractéristique du protocole TCP

- Fiable
- Les paquets sont en ordre

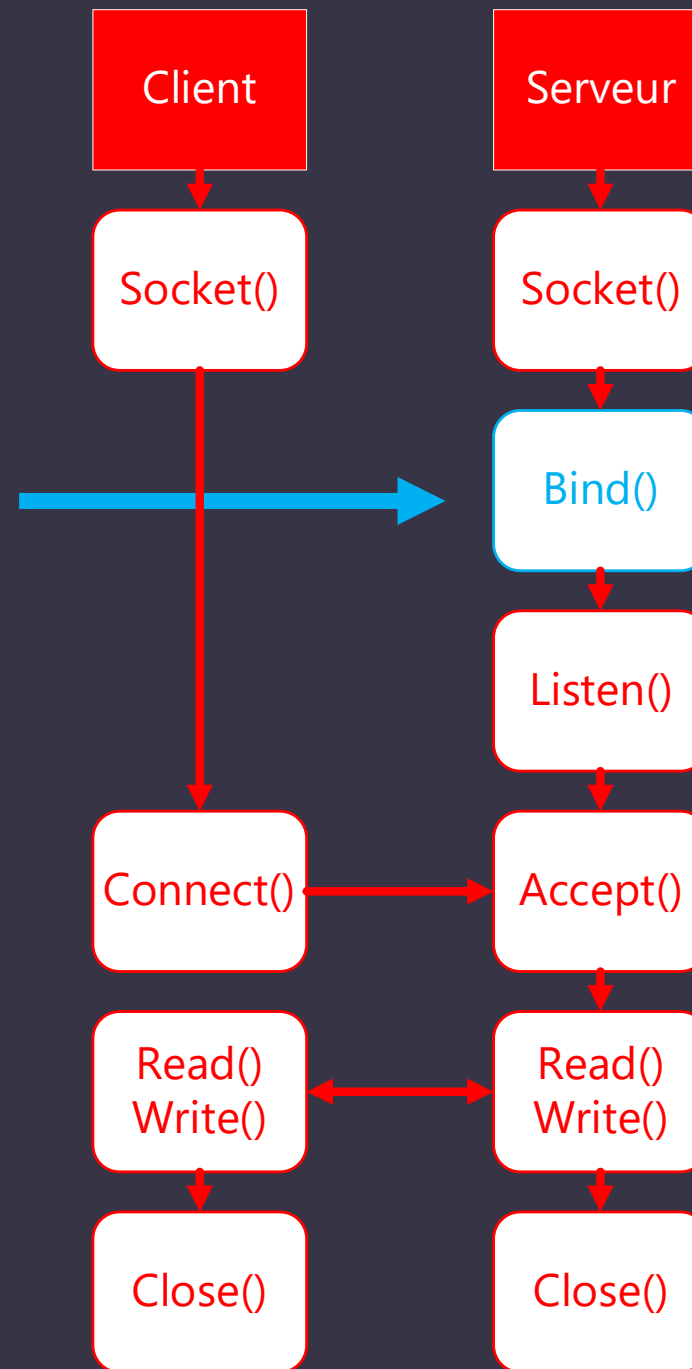
Étape de connexion avec des Sockets



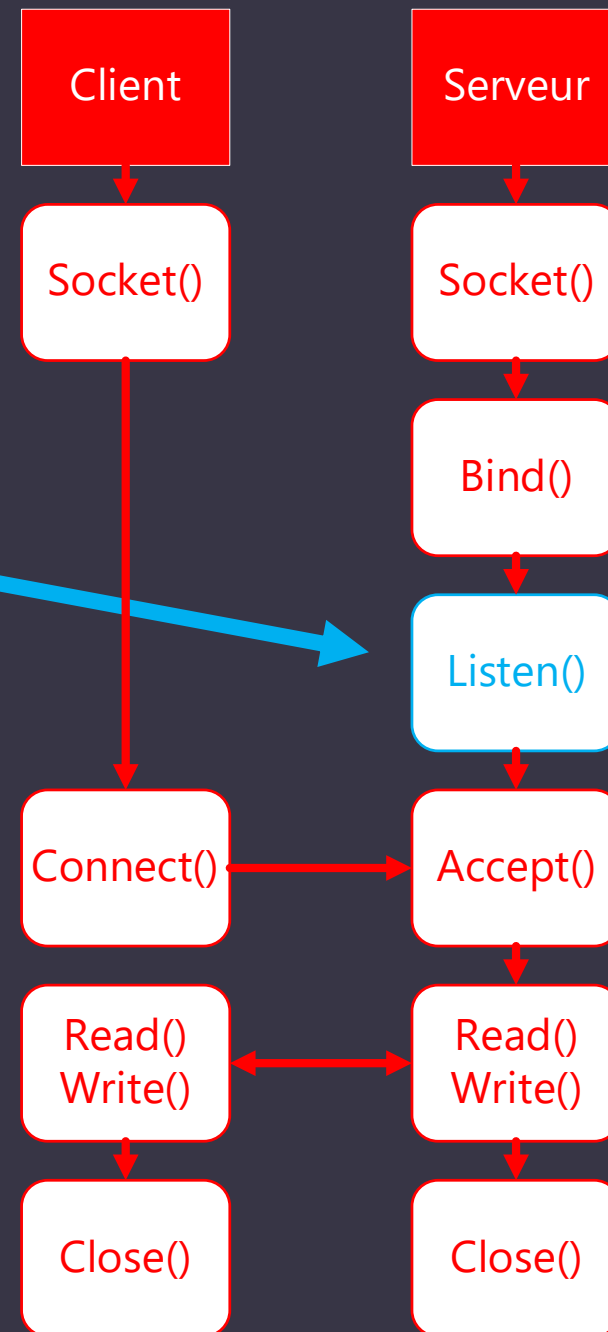
Création de l'objet responsable de la gestion de la communication TCP



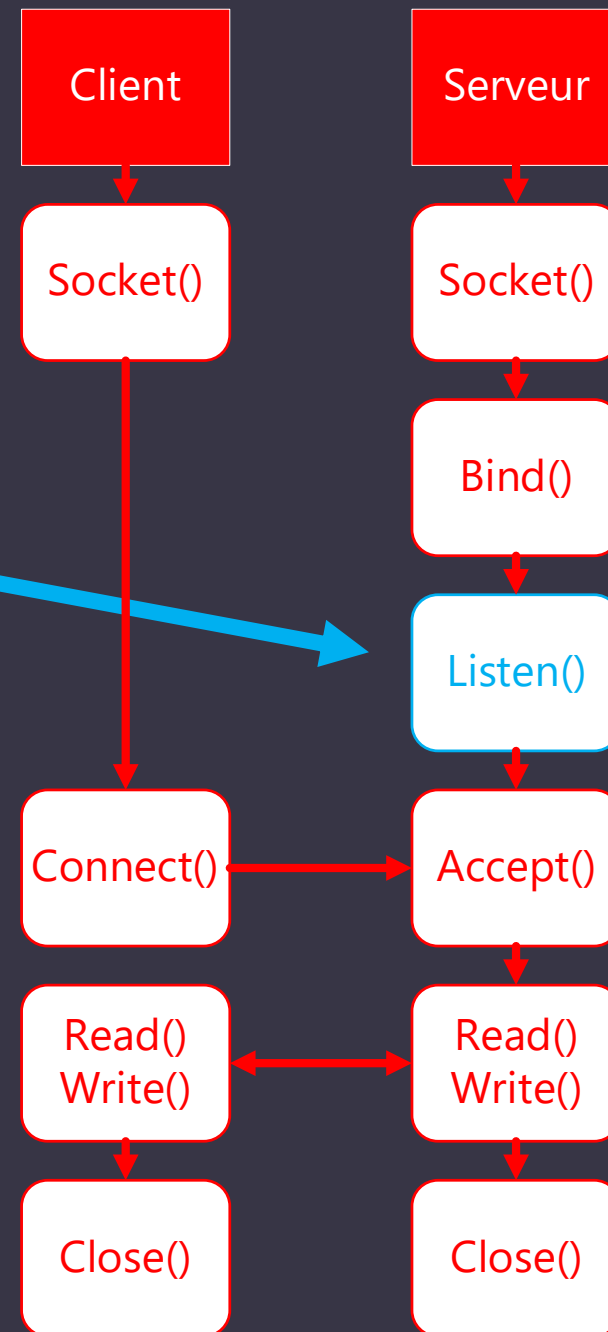
Associe l'adresse
et le port au
socket (le port doit
être libre)



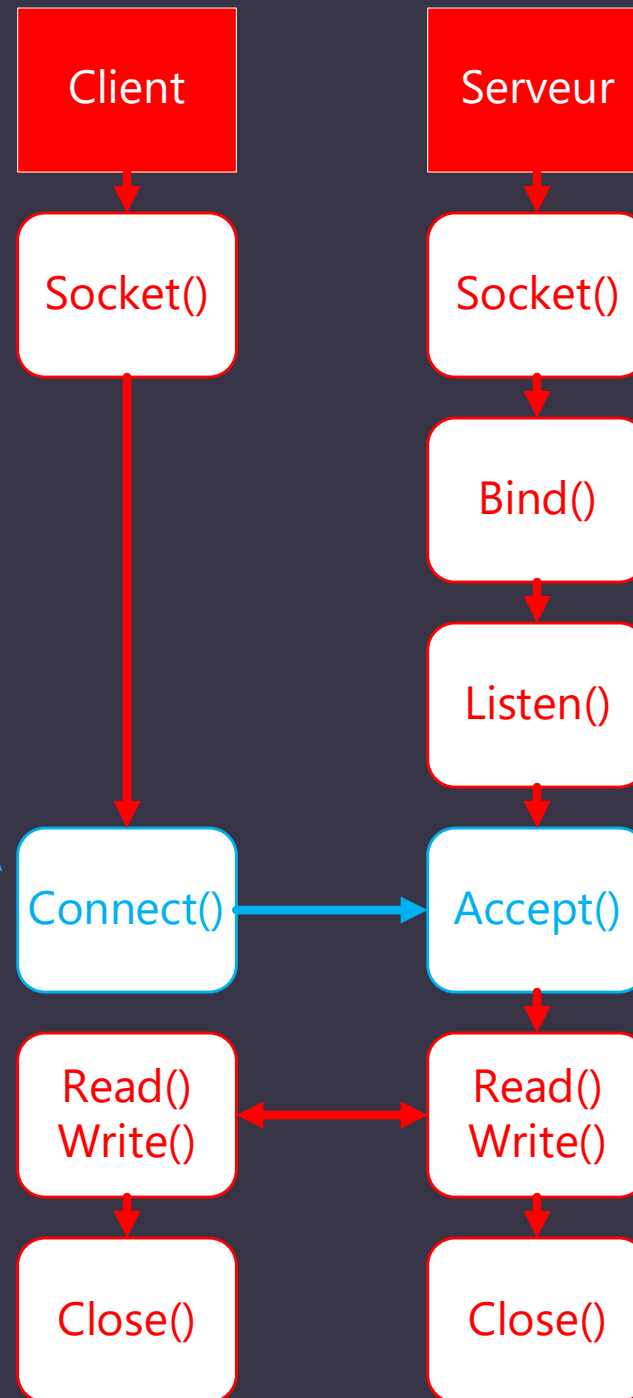
Ouvre le socket sur
le port pour recevoir
les demandes de
connexion



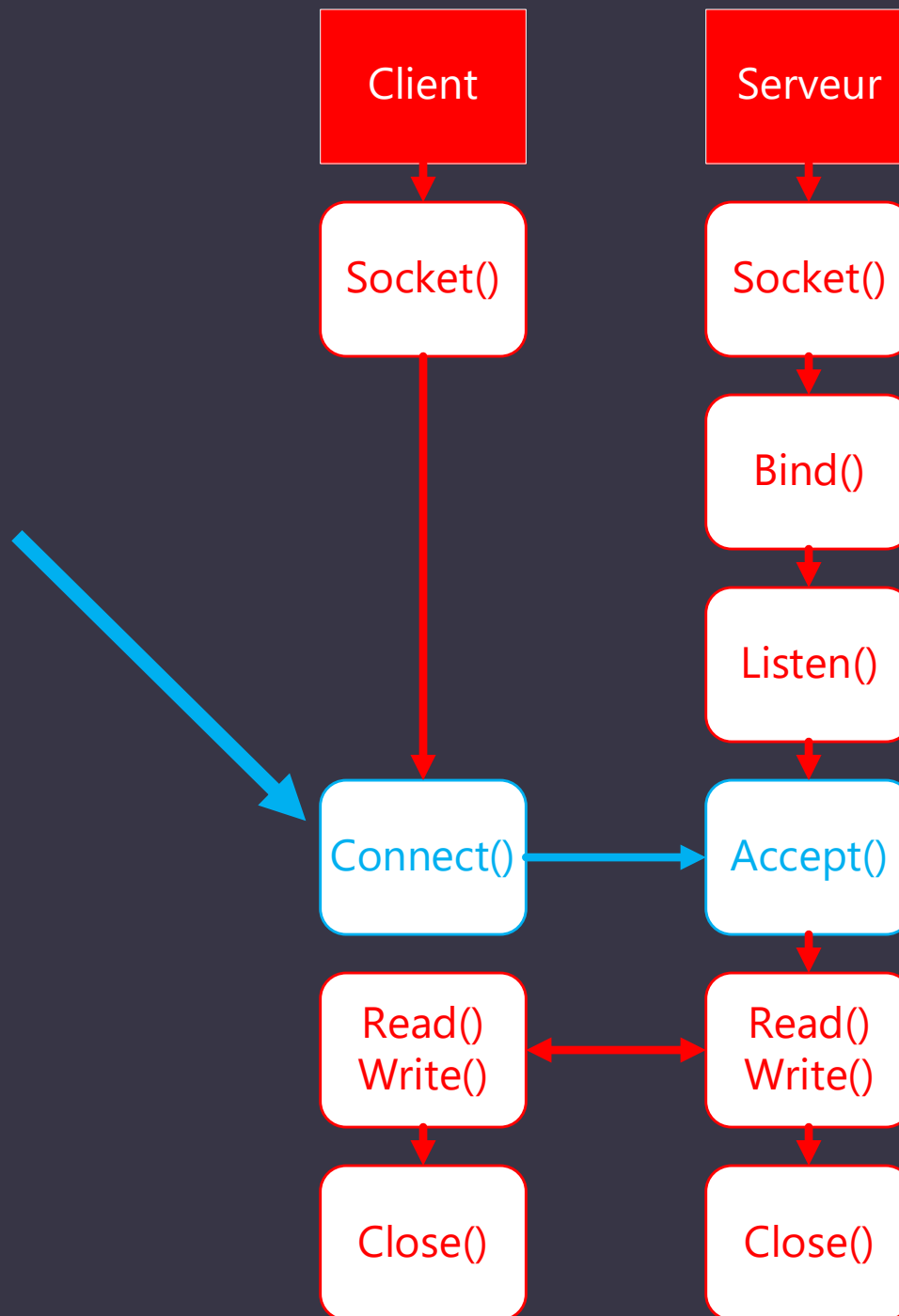
Ouvre le socket sur
le port pour recevoir
les demandes de
connexion



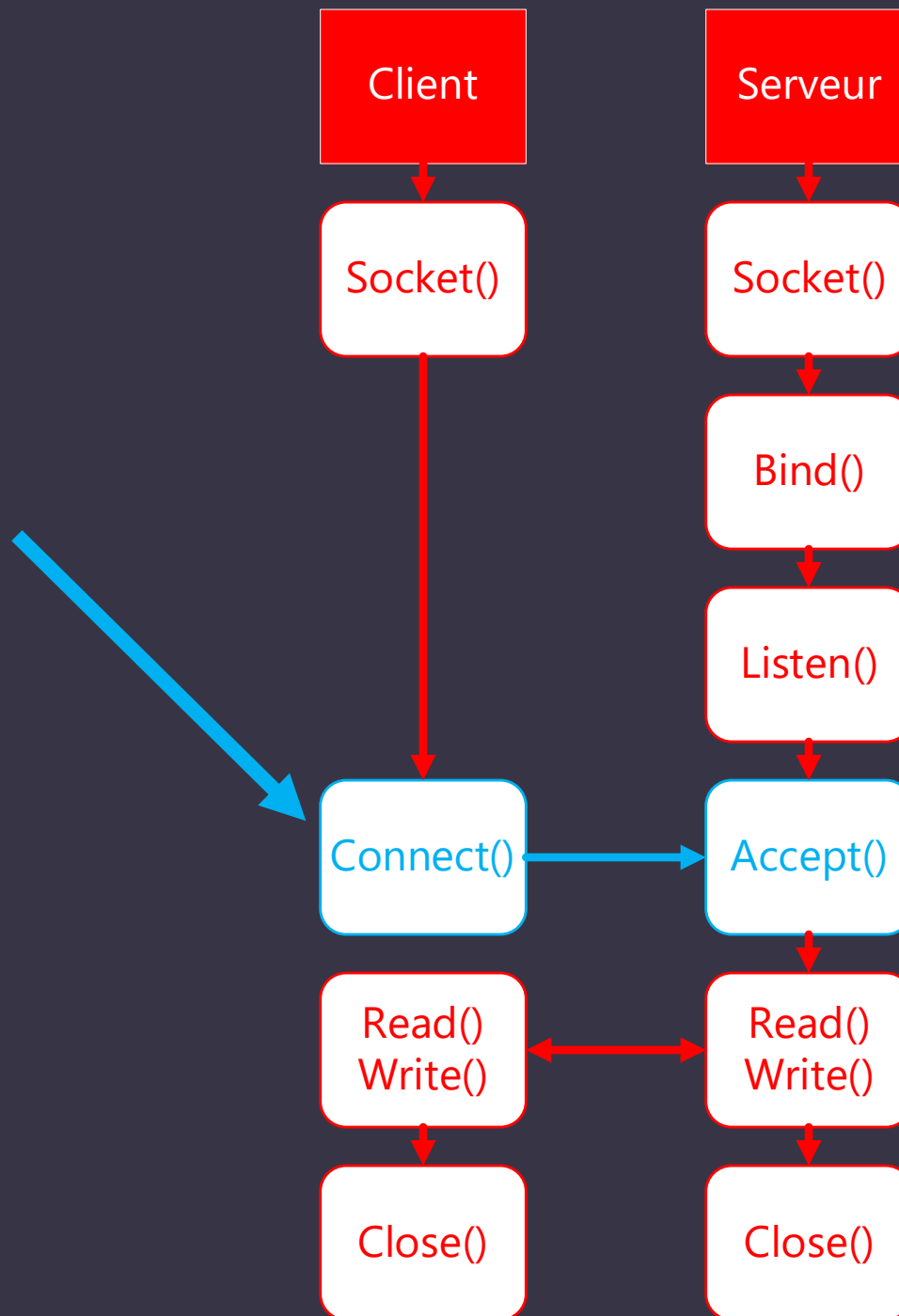
Ouvre le socket sur
le port pour recevoir
les demandes de
connexion



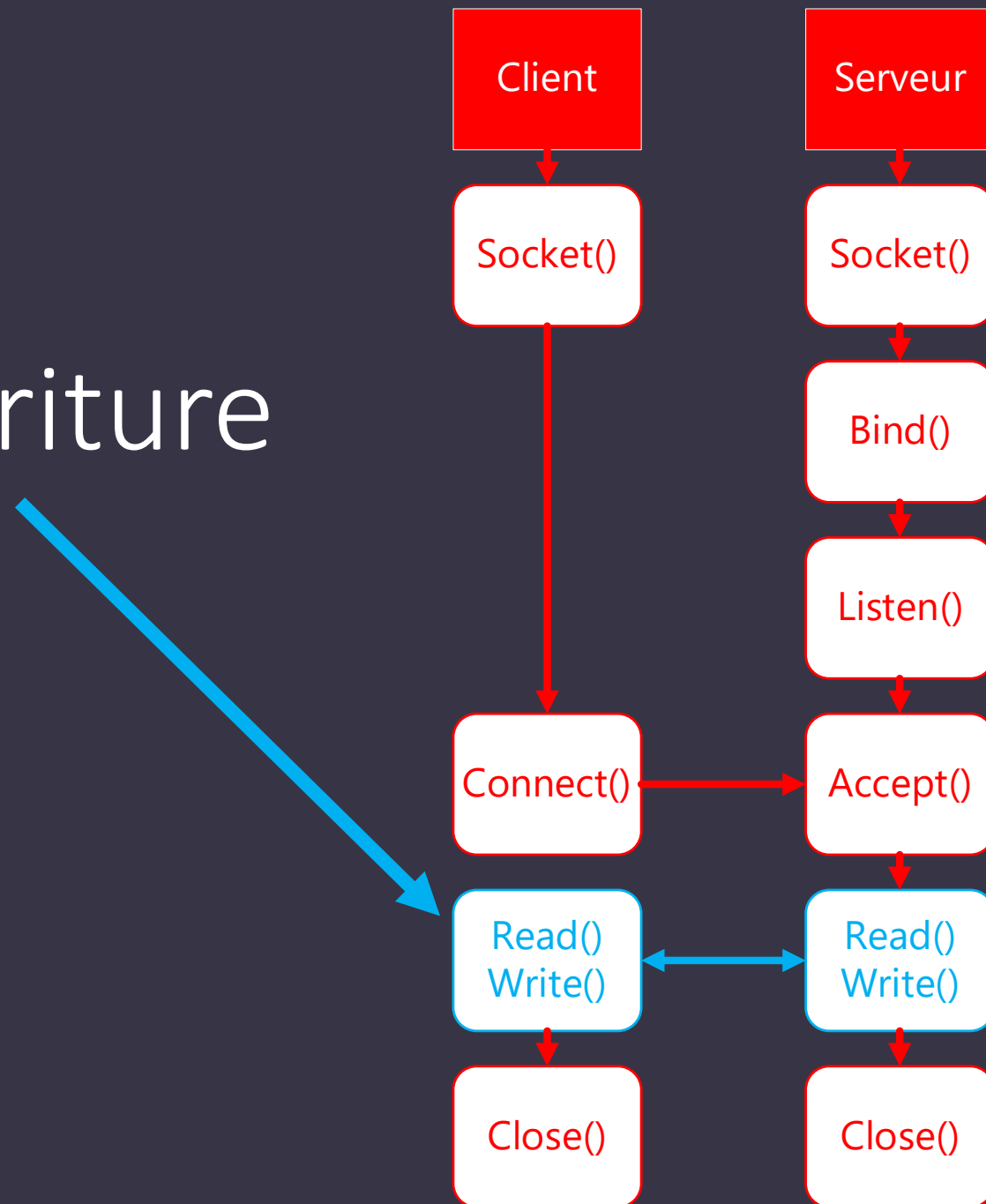
Demande de connexion et acceptation



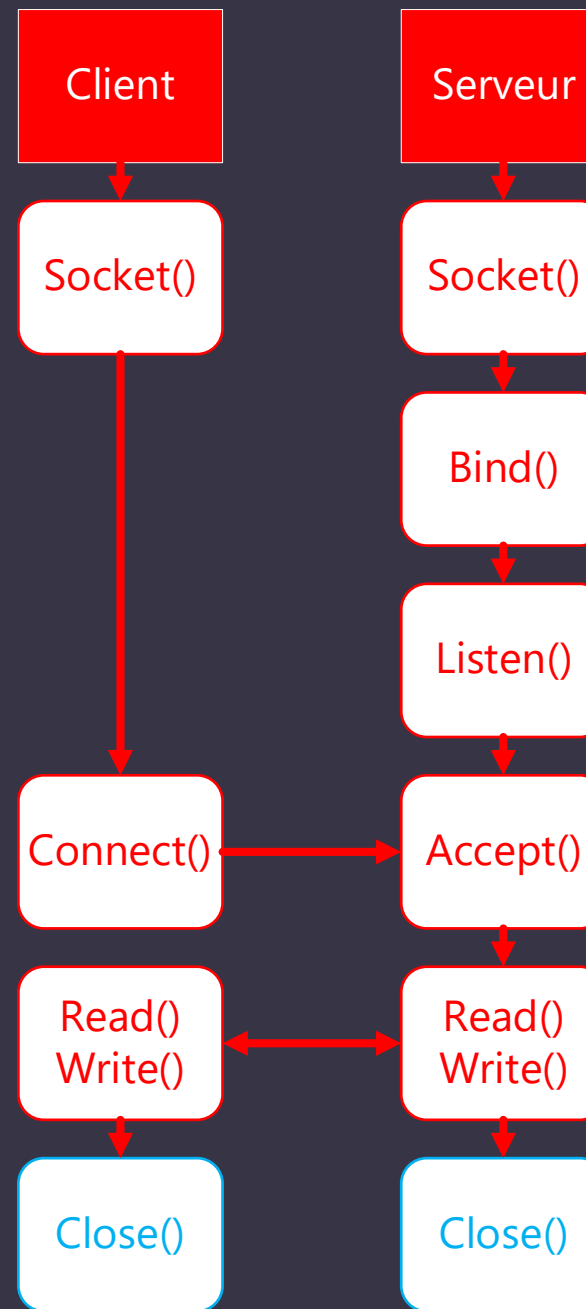

Demande de connexion et acceptation



Lecture et écriture



Fermeture qui
peut être initiée
des 2 côtés



Attention

Il ne peut pas y avoir 2 processus sur le même port de la même adresse IP. Possible d'écouter 2 fois le même port sur 2 cartes réseau différentes. Toutefois, possible pour un processus d'accepter plusieurs connexions sur le même port.

Exemple de conflit :

- Apache et IIS sur le 80

L'envoi d'un message

Application Expéditeur

Message

Application Destinataire

Couche transport (TCP)

Couche transport (TCP)

Couche réseau(IP)

Couche réseau(IP)

L'envoi d'un message

Application Expéditeur

Couche transport (TCP)

Couche réseau(IP)

Port
Message

Application Destinataire

Couche transport (TCP)

Couche réseau(IP)

L'envoi d'un message

Application Expéditeur

Couche transport (TCP)

Couche réseau(IP)

IP
Port
Message

Application Destinataire

Couche transport (TCP)

Couche réseau(IP)

L'envoi d'un message

Application Expéditeur

Application Destinataire

Couche transport (TCP)

Couche transport (TCP)

Couche réseau(IP)

IP
Port
Message

Couche réseau(IP)

L'envoi d'un message

Application Expéditeur

Application Destinataire

Couche transport (TCP)

Port
Message

Couche transport (TCP)

Couche réseau(IP)

Couche réseau(IP)

L'envoi d'un message

Application Expéditeur

Message

Application Destinataire

Couche transport (TCP)

Couche transport (TCP)

Couche réseau(IP)

Couche réseau(IP)

Exemple de code python

Étape création du socket

Création d'un socket :

```
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Précise le type d'adresse. AF_INET signifie IPV4.

Étape création du socket

Création d'un socket :

```
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

`SOCK_STREAM` est la version TCP. `SOCK_DGRAM` est la version UDP.

Étape « Bind » Serveur

IP = ' ' # Symbolise toutes les interfaces

port = 5005 # port où écouter

server.bind((IP, port))



Prend 1 argument adresse. Toutefois, ce paramètre est un tuple du port et de l'IP. Ce paramètre serait différent si la famille d'adresse était différente.

Étape « Listen » Serveur

```
server.listen(1)
```



Permet au serveur d'accepter des connexions. Le paramètre précise le nombre de connexions en attente qui peut être mise dans une file avant de refuser les nouvelles tentatives de connexion.

Étape de connexion (client)

```
client.connect((IP,port))
```

IP du serveur. Port ciblé sur le serveur.

Étape « Accept » Serveur

```
connection, address = server.accept()
```



Retourne un objet socket qui représente la connexion qui permet de recevoir et d'envoyer des données. L'adresse est celle liée au socket.

Étape écrire

```
client.sendall(b'MessageVersServeur')  
server.sendall(b'MessageVersClient')
```

La version de la fonction `.send()` nécessite une gestion supplémentaire pour savoir quelle partie a pu être envoyée et ce qui reste à envoyer.

Étape Lire

```
server.recv(2048)
```

```
client.recv(2048)
```



Indique la taille du buffer. Attention, un même message peut être reçu par plusieurs recv.

Problématique à gérer

- 2 paquets qui arrivent au même moment
- Gestion de plusieurs connexions simultanées

Encodage

Pour convertir une variable string en octets
`string.encode('utf-8')`

Thread en python

Un thread

Un thread est permet à un programme d'avoir plusieurs parties de code qui s'exécutent en simultanées ou sinon presque. C'est très important si certaines parties sont bloquantes. Par exemple, on ne souhaite pas qu'une communication qui bloque vienne bloquer tous les autres.

Thread en python

En python, deux thread ne vont pas fonctionner sur 2 cœurs différents. Deux thread ne fonctionneront pas en même temps. Il n'y aura donc pas de gain de rapidité. Toutefois, cela permet quand même de résoudre le problème de blocage.

Création d'un thread

1. Créer une fonction qui s'exécute dans le thread.
2. Créer un thread qui appelle cette fonction
3. Démarrer le thread

Note

Les exemples suivants nécessitent
l'importation de : `threading`, `time`

Création d'une fonction

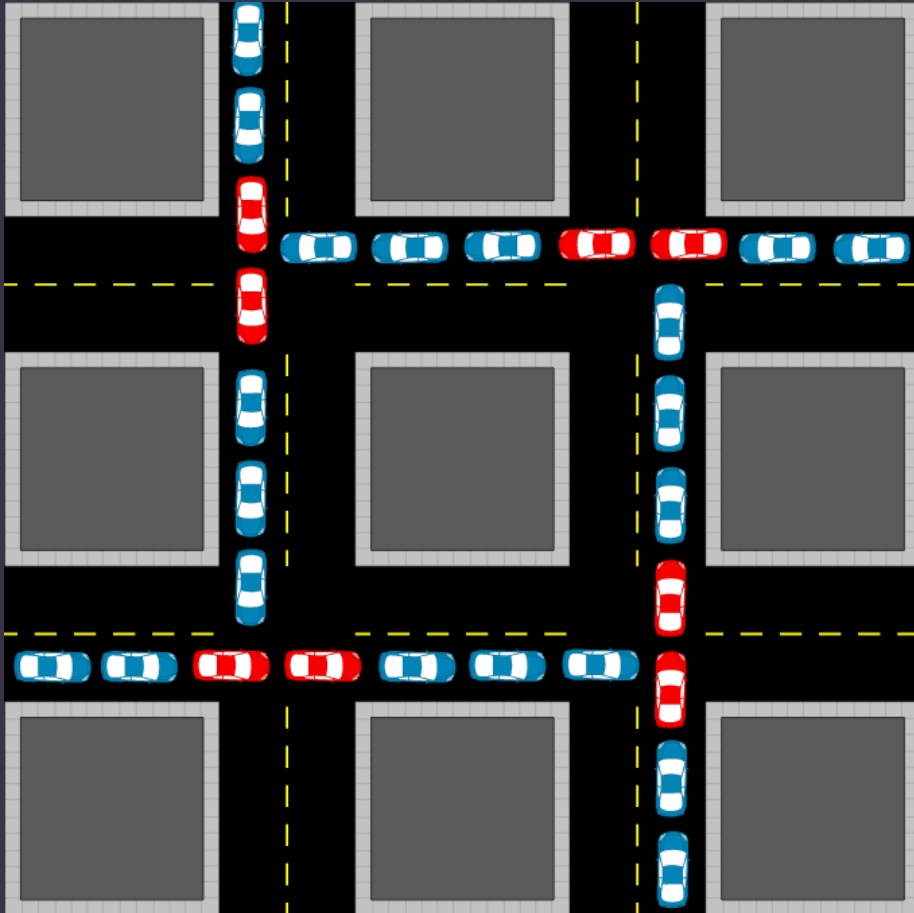
```
def fonction_du_thread(param1):  
    while True:  
        print('hello there')  
        time.sleep(2) #instruction qui bloque
```


Création et démarrage du thread

```
mon_thread = threading.Thread(target= fonction_du_thread, args=(1,))
```

```
mon_thread.start()
```

Interblocage



- Thread A
 1. Prend la ressource A
 2. Prend la ressource B
- Exécution
- Libère A
- Libère B

- Thread B
 2. Prend la ressource B
 1. Prend la ressource A
- Exécution
- Libère B
- Libère A

SRC :

<https://en.wikipedia.org/wiki/Gridlock#/media/File:Gridlock.svg>

g CC BY-SA 2.5

Attention

Le partage de variable entre différents thread est relativement dangereux. Par exemple, itérer sur un tableau dans un thread alors qu'un autre thread peut retirer des éléments risque de vous causer des problèmes.

Pour d'information

- Lock
- Sémaphore
- Queue