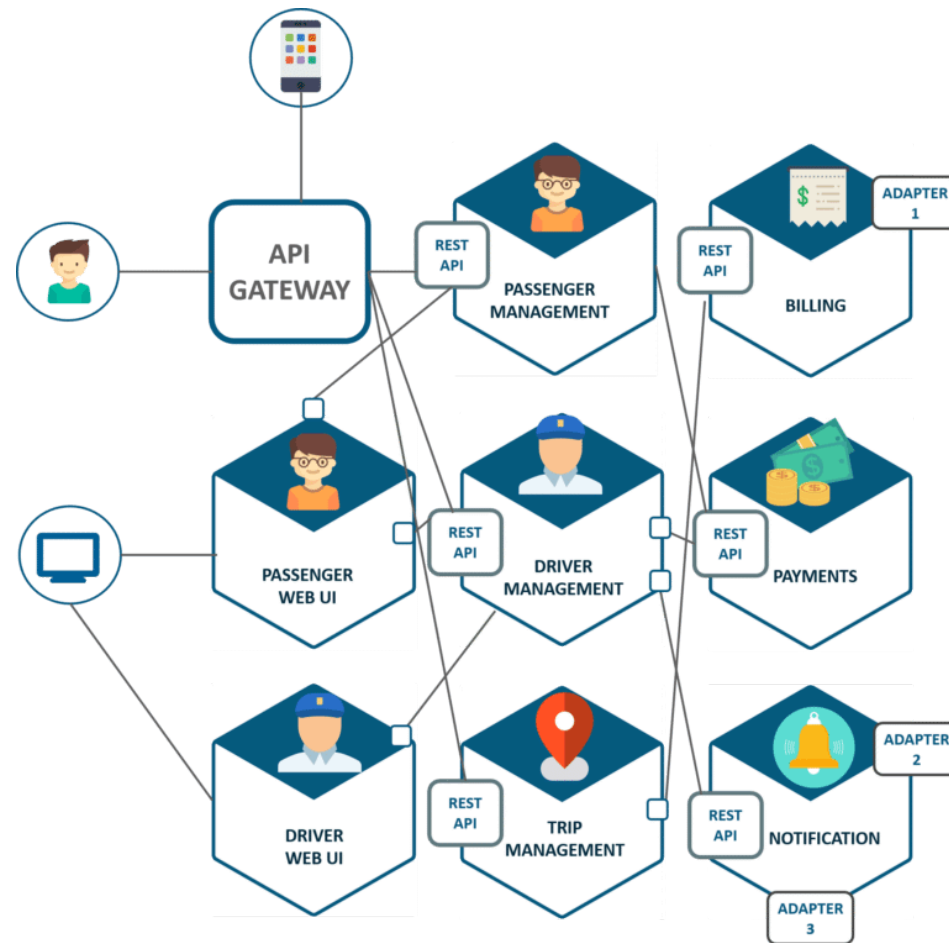


# ALOM



**HANDCRAFTING EVERYTHING**

**THREADS, SOCKETS & SERVLETS** 



# UN MICRO-SERVICE C'EST :

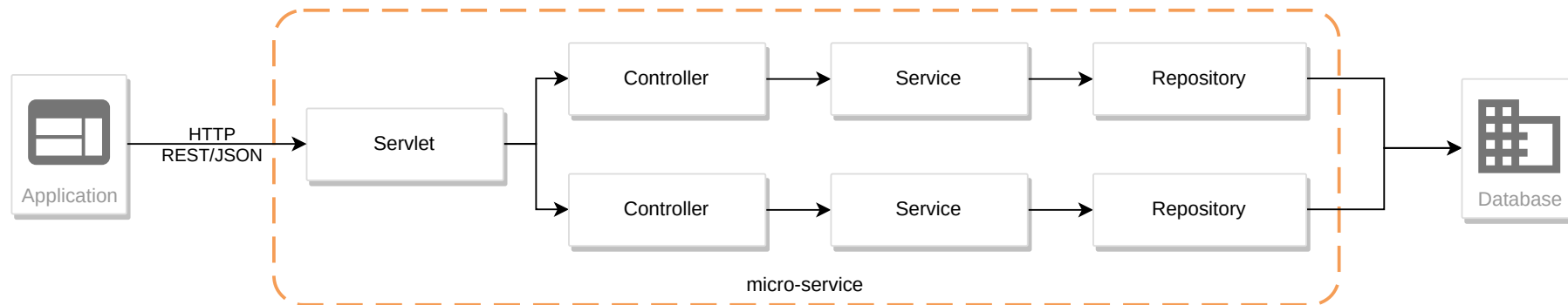
- Un ou plusieurs canaux de communication
  - HTTP - REST/JSON
  - HTTP - REST/XML
  - JMS / Messaging
  - Kafka / Avro
- Un ensemble de fonctionnalités du même domaine métier
- Une source de données dédiée

# ZOOM SUR UN MICRO-SERVICE



# ZOOM SUR UN MICRO-SERVICE JAVA

On s'appuie sur les technologies connues: les servlets !

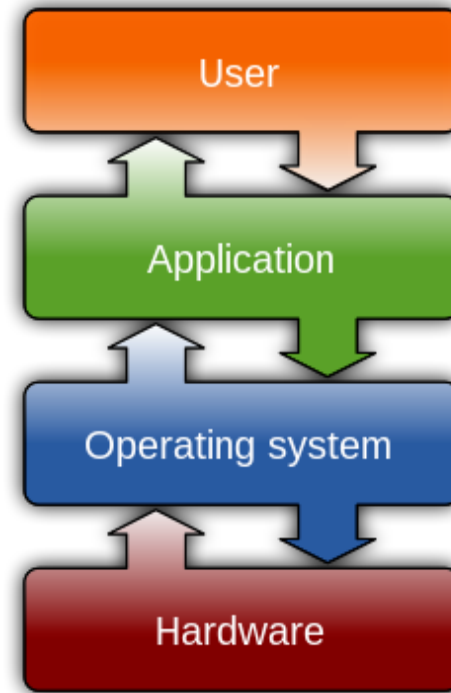


# COMMENT FONCTIONNE UNE SERVLET ?

- Utilise des Sockets pour les communication réseau
- Utilise des Threads pour exécuter chaque Socket indépendamment
- Parse les requêtes HTTP (du texte) pour en faire des objets Java

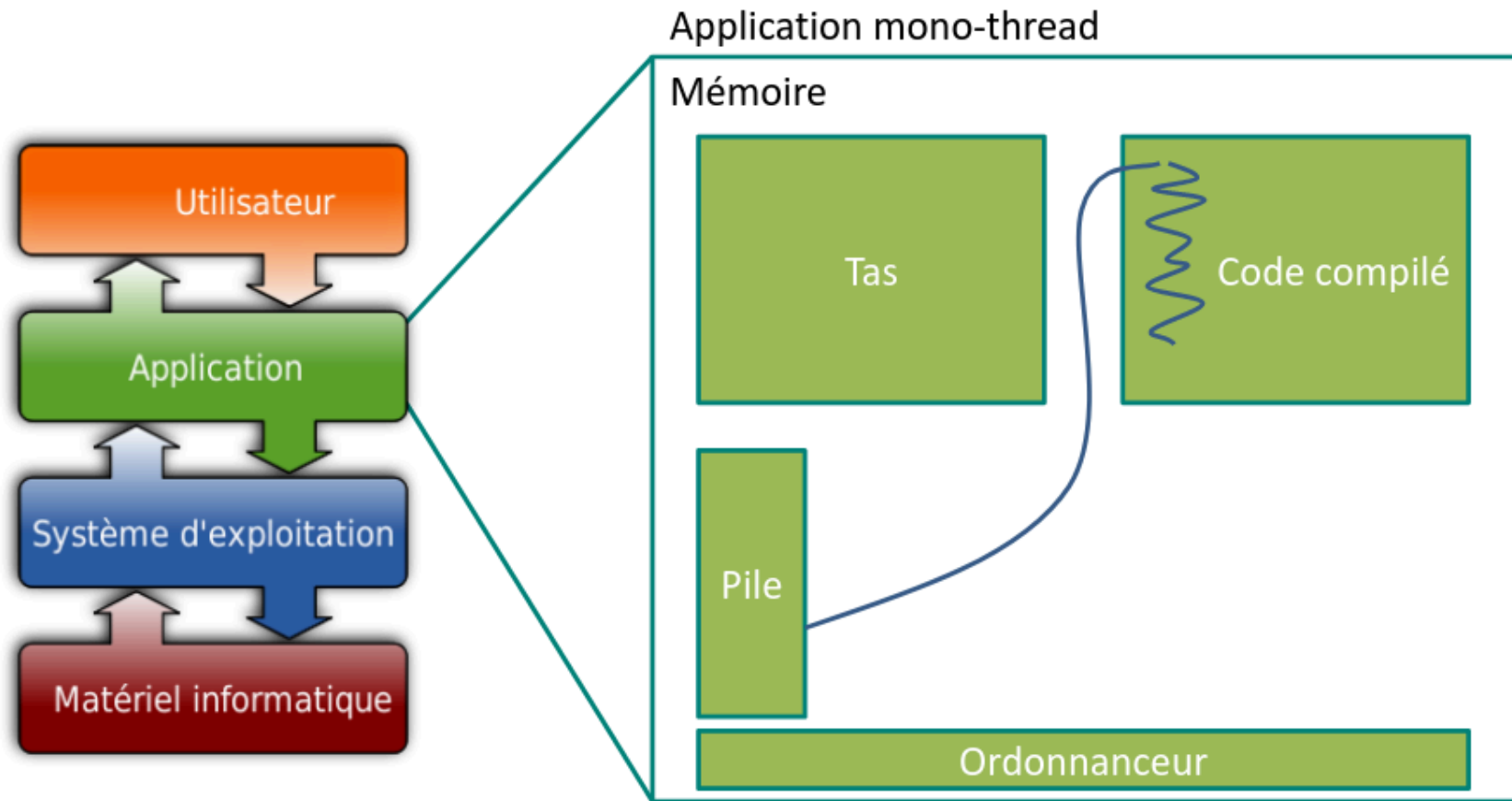
# THREADS

# LE SYSTÈME D'EXPLOITATION

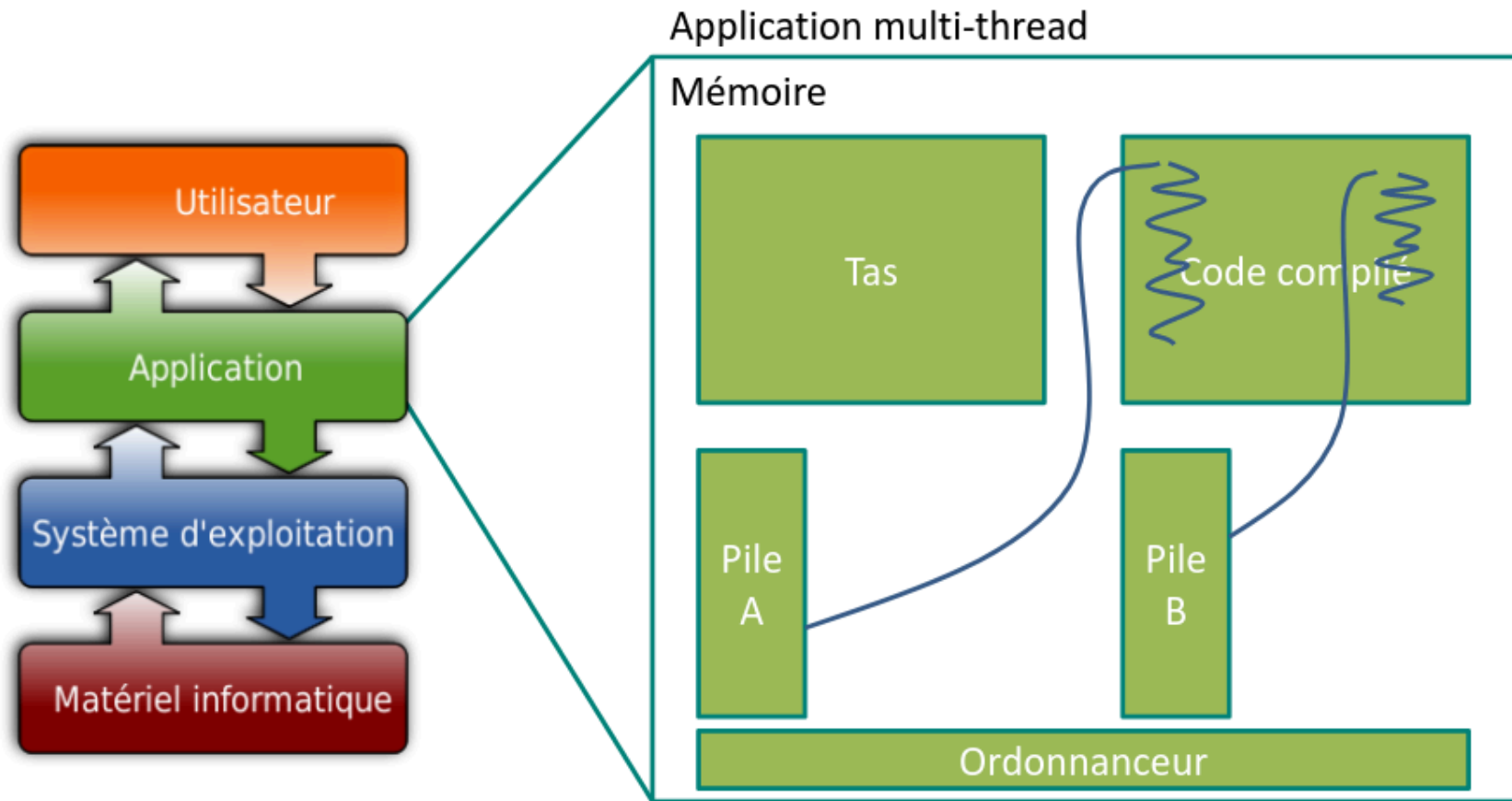




# UNE APPLICATION MONO-THREADÉE



# UNE APPLICATION MULTI-THREADÉE

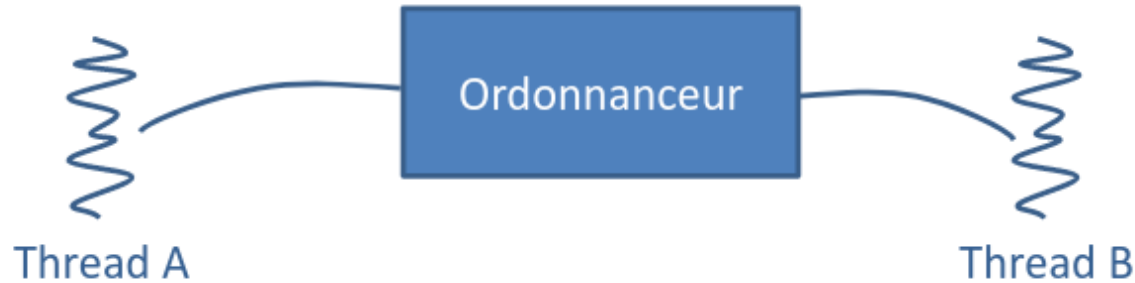


# LES THREADS EN JAVA

```
public void run() {  
    System.out.println("I am Thread A");  
}
```

```
public void run() {  
    System.out.println("I am Thread B");  
}
```

# LES THREADS SONT INDÉPENDANTS ET ORDONNANCÉS



```
> I am Thread A  
> I am Thread B
```

```
> I am Thread B  
> I am Thread A
```

# CRÉATION D'UN THREAD

- Deux possibilités pour créer une classe exécutée par un thread :
  - Création d'une classe implémentant l'interface Runnable
  - Création d'une classe héritant de la classe abstraite Thread
- Dans les deux cas, il faut implémenter la méthode `public void run()`

# INSTANCIATION ET DÉMARRAGE

- Pour instancier une classe implémentant Runnable :
  - `Thread t = new Thread(new MaClasse());`
- Pour instancier une classe héritant de la classe Thread :
  - `Thread t = new MaClasse();`
- Pour démarrer le Thread t lorsqu'il a été instancié :
  - `t.start();`

# CYCLE DE VIE D'UN THREAD

- Lorsqu'un thread démarre, c'est la méthode `run()` qui est exécutée
- Le thread est « vivant » (alive) aussi longtemps qu'il n'a pas terminé d'exécuter la méthode `run()`
- La seule manière « propre et naturelle » de terminer un thread est de faire en sorte que le thread termine l'exécution de la méthode `run()`

# CYCLE DE VIE D'UN THREAD

- Interdiction d'utiliser les méthodes suivantes pour arrêter un thread (deprecated) :
  - `stop()`
  - `suspend()`
- A utiliser pour arrêter un thread:
  - `interrupt()`: Peut causer des `InterruptedException` si le thread était en attente (`Thread.sleep`, ou `Object.wait()`)



# DÉVELOPPER DES THREADS *INFINIS*

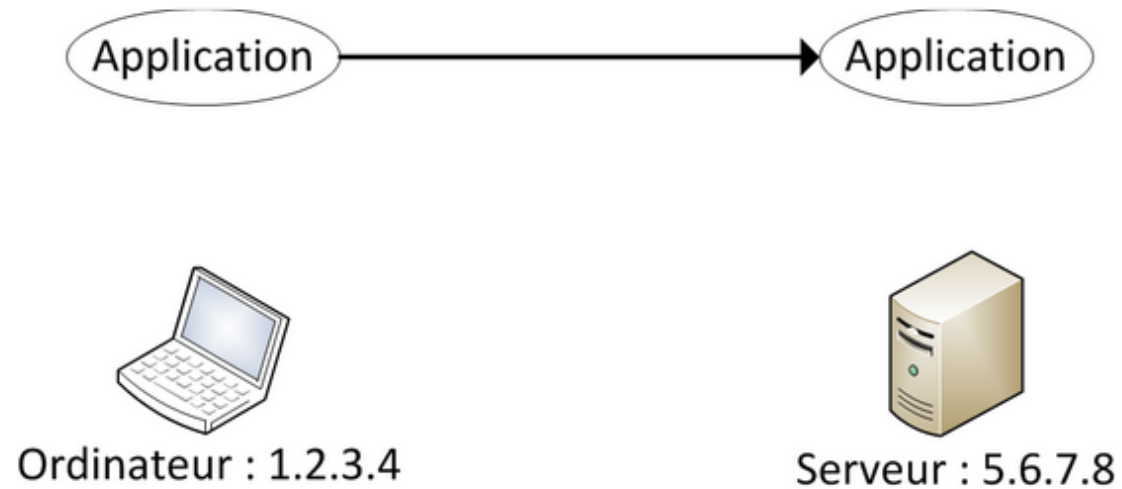
```
public void run(){  
    while(true) {  
        // Non !  
        // Impossible à arrêter proprement !  
    }  
}
```

# DÉVELOPPER DES THREADS *INFINIS*

```
1 public class FinishableThread extends Thread {  
2     private boolean running = false;  
3     public void run(){  
4         this.running = true;  
5         while(this.running) {  
6             }  
7     }  
8     public void finish(){  
9         this.running = false;  
10    }  
11 }
```

# SOCKETS ET CONNEXIONS

# COMMUNICATIONS IP



# SOCKETS CÔTÉ CLIENT

Pour créer une connexion vers un serveur, il faut :

- Créer une socket côté client
- Renseigner l'adresse IP du serveur
- Renseigner le port d'écoute du serveur

Exemple :

```
Socket s = new Socket("5.6.7.8",12345);  
// s.getInputStream() to read data  
// s.getOutputStream() to send data
```

# L'ÉTABLISSEMENT D'UNE CONNEXION

# SOCKETS CÔTÉ SERVEUR

Pour accepter des connexions côté serveur, il faut :

- Créer une socket serveur côté serveur
- Renseigner le port d'écoute souhaité
- Lancer l'écoute

Exemple :

```
ServerSocket ss = new ServerSocket(12345);  
Socket s = ss.accept();  
// s.getInputStream() to read data  
// s.getOutputStream() to send data
```

# SOCKETS & THREADS

Exécuter une boucle de `accept()` dans un Thread, et les associer un Thread à chaque `Socket` acceptée

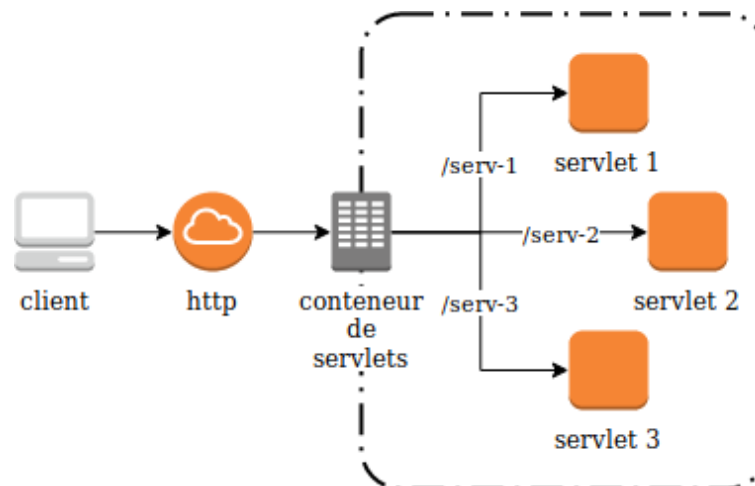


# SERVLETS

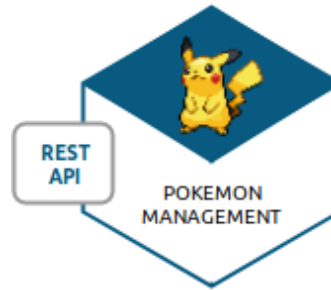
## PRINCIPES

Morceau de serveur

- Traite des requêtes HTTP associées à une URL
- Exécutée par le conteneur de servlets (ex. Tomcat)



# ET SI ON JOUAIT?



 TP THREADS ET SOCKETS

 TP SERVLETS



# FIN DU COURS