Threads, thread-safe

Leleux Laurent

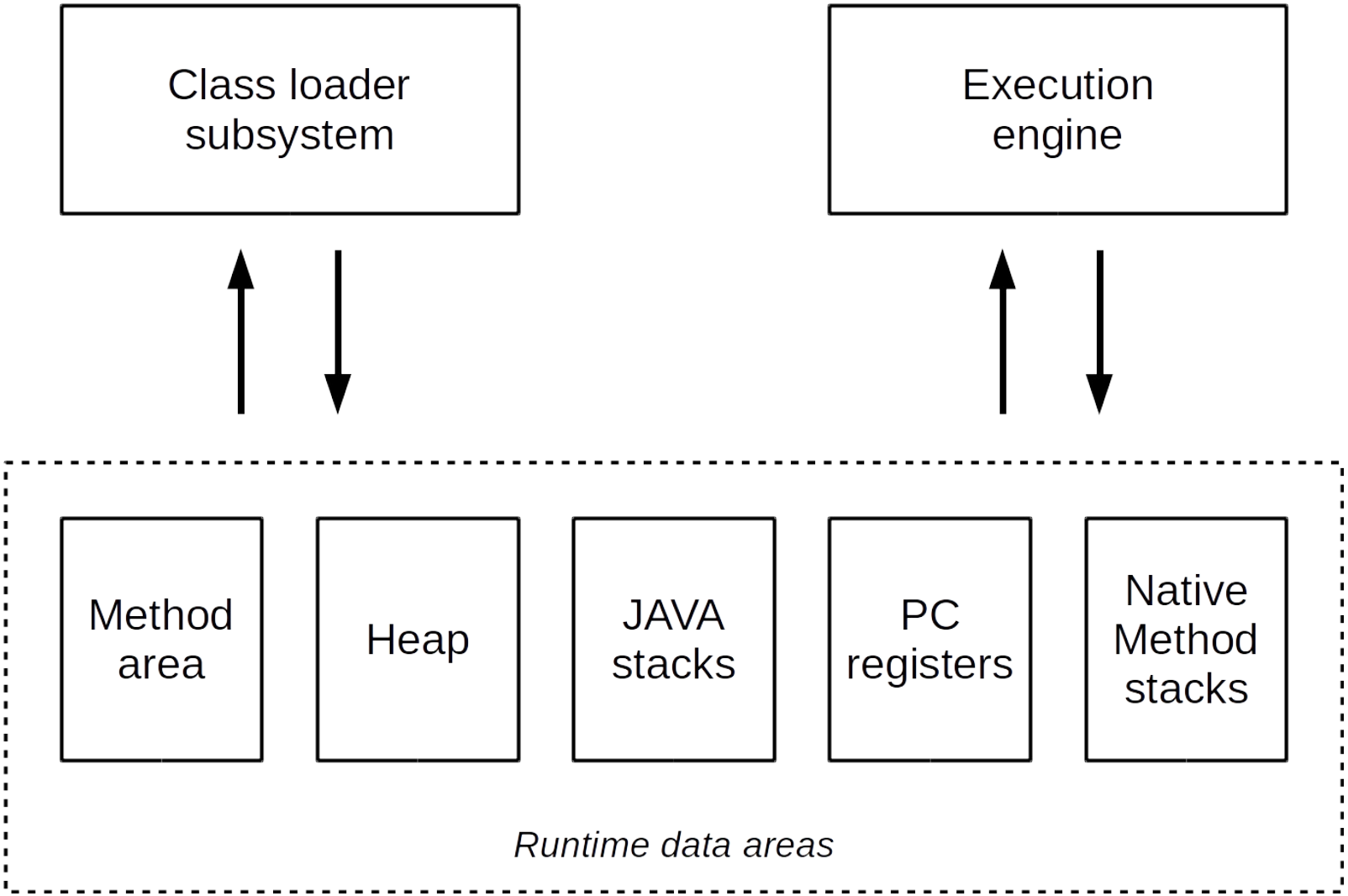
2022 - 2023

Thread ?

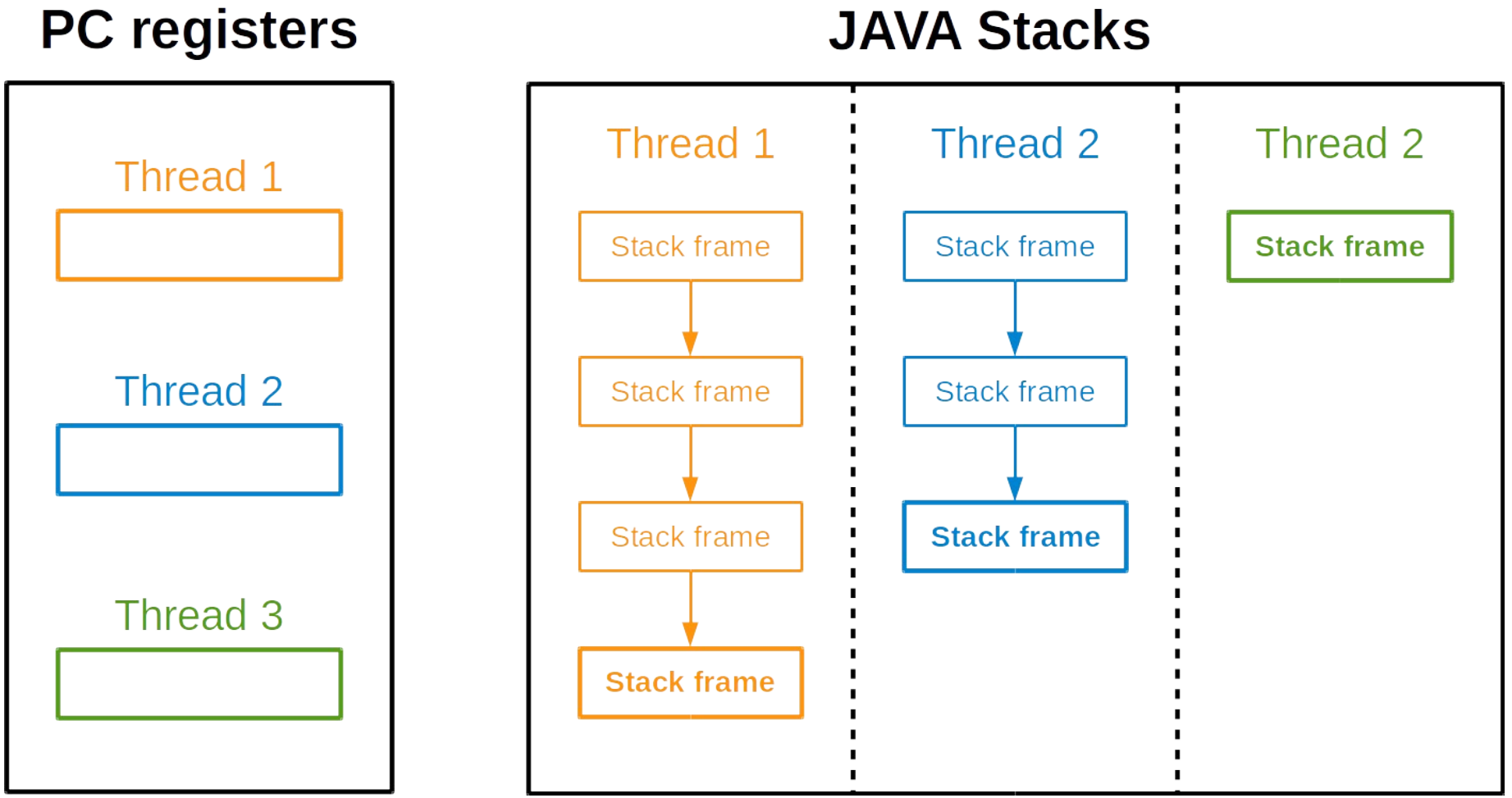
- Un fil d'exécution (processus ?)

- Partagé :
  - Method area
  - Heap
- Privé :
  - Stack
  - Pc-register

JVM


Class loader subsystem
Execution engine
Method area
Heap
JAVA stacks
PC registers
Native Method stacks
Runtime data areas

JVM memory


PC registers
Thread 1
Thread 2
Thread 3
JAVA Stacks
Thread 1
Stack frame
Stack frame
Stack frame
Stack frame
Thread 2
Stack frame
Stack frame
Stack frame
Thread 2
Stack frame

Exemples

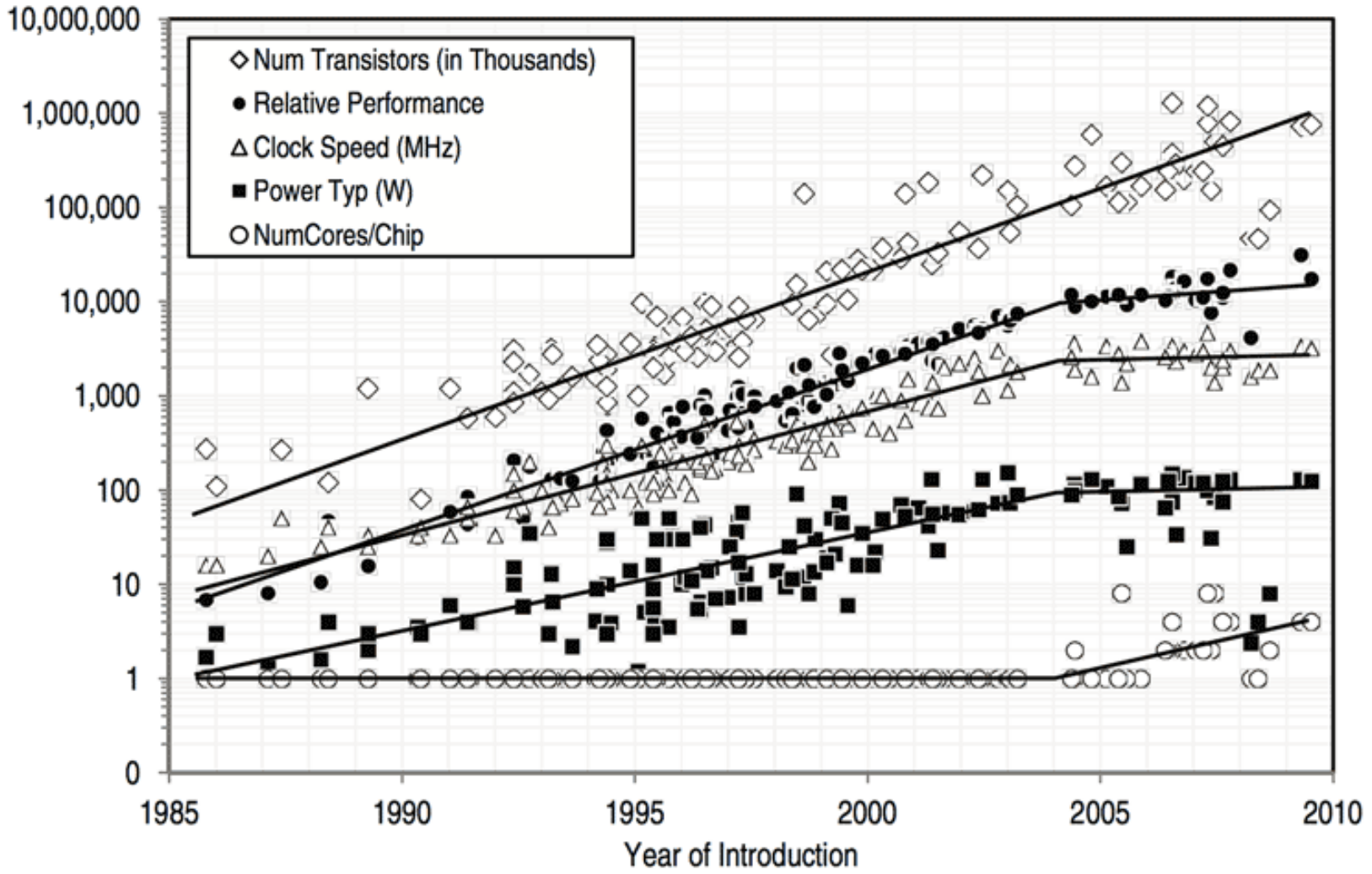
- Navigateur
- Serveur Web/FTP
- Accès DB
- Lecture de fichiers

Avantages

- Parallélisme
  - I/O non bloquants
  - Performances

- Processeurs multi-cœurs

Loi de Moore


10,000,000
1,000,000
100,000
10,000
1,000
100
10
1
0
Num Transistors (in Thousands)
Relative Performance
Clock Speed (MHz)
Power Typ (W)
NumCores/Chip
1985
1990
1995
2000
2005
2010
Year of Introduction

Partage de ressources

- Performances
- Synchronisation

- /!\ JVM multiples /!\

Exemple

```
class Counter {

    private int c = 0;

    public void increment() {
        c++;
    }

    public void decrement() {
        c--;
    }

    public int value() {
        return c;
    }

}
```

Thread-safe

- Non partage
  - Thread-local
- Partage
  - Exclusion mutuelle
  - Opérations atomiques
  - Objets immuables

Exclusion mutuelle

- Cadenas
- Pour classes et instances

- « `synchronized` » ferme le cadenas
- Deux écritures (method, statement)

- /!\ Deadlocks /!\

Exemple

```
class SynchronizedCounter {

    private int c = 0;

    public synchronized void increment() {
        c++;
    }

    public synchronized void decrement() {
        c--;
    }

    public synchronized int value() {
        return c;
    }

}
```

Exemple

```
class AtomicCounter {

    private AtomicInteger c = new AtomicInteger(0);

    public void increment() {
        c.incrementAndGet();
    }

    public void decrement() {
        c.decrementAndGet();
    }

    public int value() {
        return c.get();
    }

}
```

Pattern « Singleton »

```
public final class Singleton {

    private static Singleton instance = null;

    private Singleton() {
        super();
    }

    public final static Singleton getInstance() {
        if (Singleton.instance == null) {
            Singleton.instance = new Singleton();
        }
        return Singleton.instance;
    }

}
```

Pattern « Singleton »

```
public final class Singleton {

    private static Singleton instance = null;

    private Singleton() {
        super();
    }

    public final static Singleton getInstance() {
        if (Singleton.instance == null) {
            synchronized(Singleton.class) {
                if (Singleton.instance == null) {
                    Singleton.instance = new Singleton();
                }
            }
        }
        return Singleton.instance;
    }

}
```

Immutable Objects

« Objets dont l'état ne peut plus changer après leur création. »

- Pas de setter, même indirects
- Attributs « final » et « private »
- Classe « final »
- Ne pas transmettre ni recevoir des références vers des objets mutables (copies si nécessaire)