



# PROGRAMAÇÃO CONCORRENTE E DISTRIBUÍDA

*PROCESSOS LIGEIRÓS - THREADS*

VEM ESTUDAR CONNOSCO  
[WWW.ISCTE-IUL.PT](http://WWW.ISCTE-IUL.PT)

# Sumário

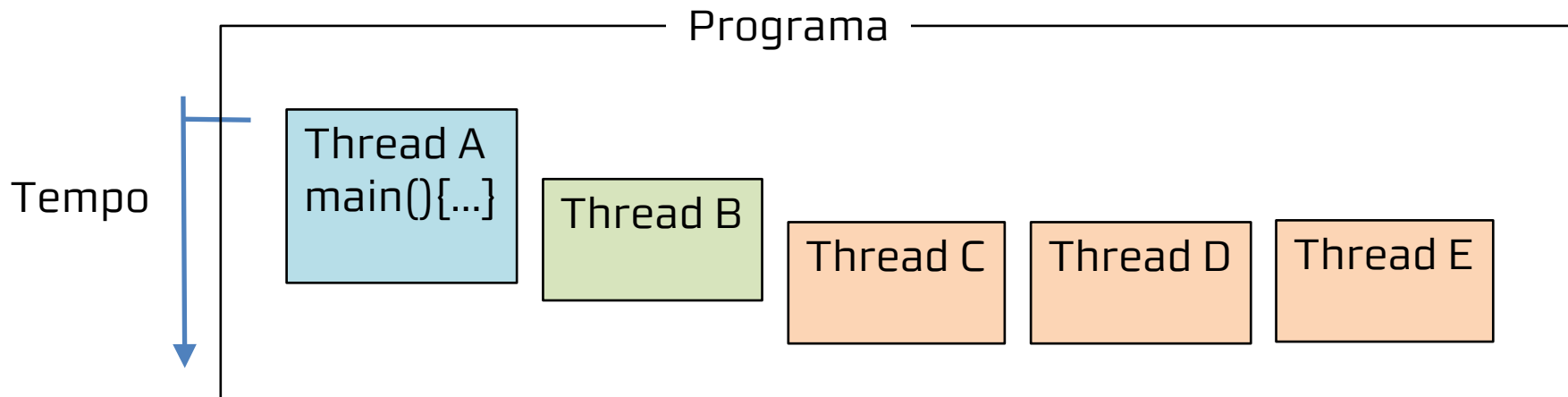
- O que é uma thread
- Partilha de memória
- Escalonamento de threads
- Threads em Java
- Ciclo de vida de uma Thread



FILME

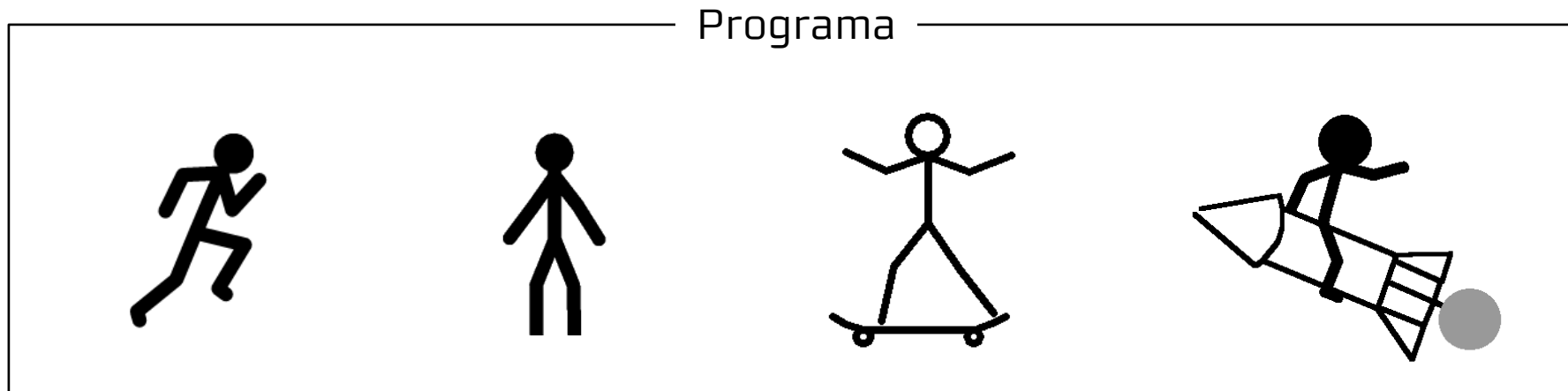
# O que é uma thread

- Vem de Thread of Control (linha de controle)
- Bloco de código que corre de uma forma concorrente



# O que é uma thread

- Vem de Thread of Control (linha de controle)
- Bloco de código que corre de uma forma concorrente



# Porque usar várias threads?

- Efetuar várias tarefas simultaneamente
- Não bloquear a aplicação à espera de informação:
  - Input do utilizador
  - Leituras e escritas para ficheiros
  - Enviar ou receber dados de outras aplicações
- Tirar partido dos vários cores do CPU

# Processos e threads

- Processos ou programas:
  - Cada um tem o seu espaço de endereçamento de memória
  - Não partilham memória nem outros recursos com outros processos
- Threads
  - Existem dentro de um processo
  - Partilham o espaço de endereçamento do seu processo
  - Facilidade de comunicação entre threads do mesmo processo

# Partilha de Memória

Aplicação multi-thread

Memória HEAP

Thread 1

PC &  
Stack

Thread 2

PC &  
Stack

Thread 3

PC &  
Stack

Thread 4

PC &  
Stack

Thread 5

PC &  
Stack

# Partilha de Memória

Aplicação  
multi-thread

Memória HEAP

aluno  
nota: 10

Thread 1

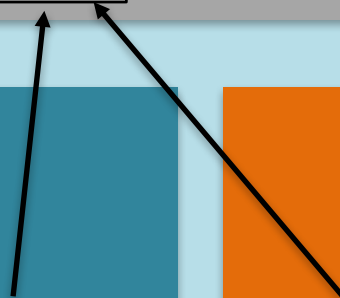
PC &  
Stack

aluno

Thread 5

PC &  
Stack

aluno





# Partilha de Memória

Aplicação  
multi-thread

`aluno.nota = 13`

**aluno**  
**nota: 10**

Memória HEAP

`aluno.nota?`

Thread 1

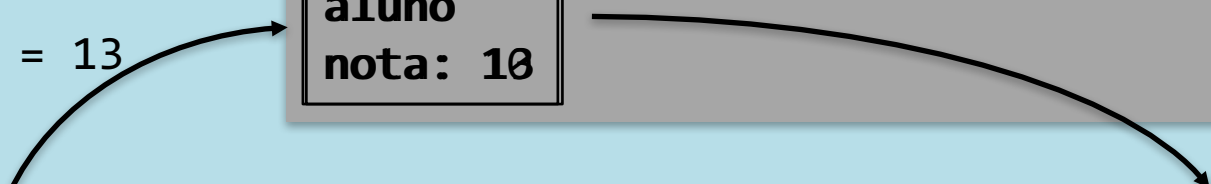
PC &  
Stack

`aluno`

Thread 5

PC &  
Stack

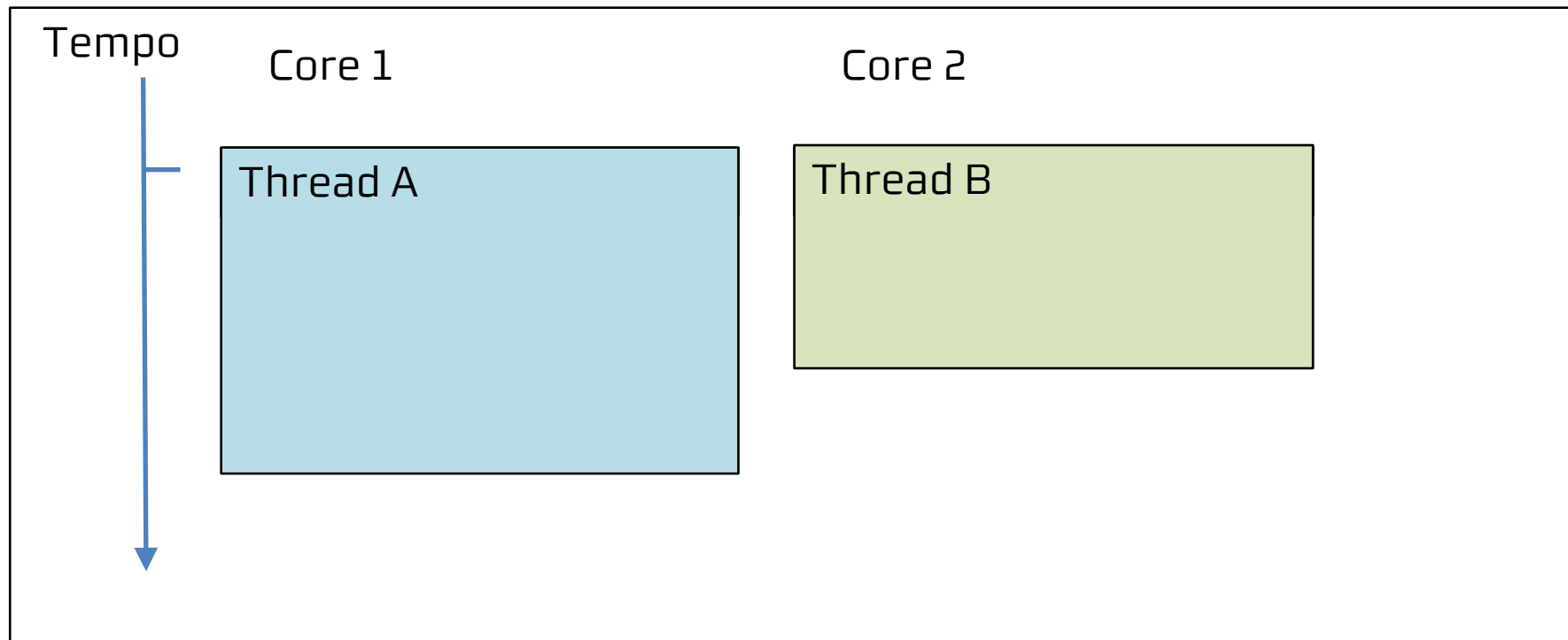
`aluno`  
`aluno.nota = 13`



# Mas como correm várias threads?

- Diferentes cenários para a execução das threads:
  - Paralela
  - Intercalada
  - Mista

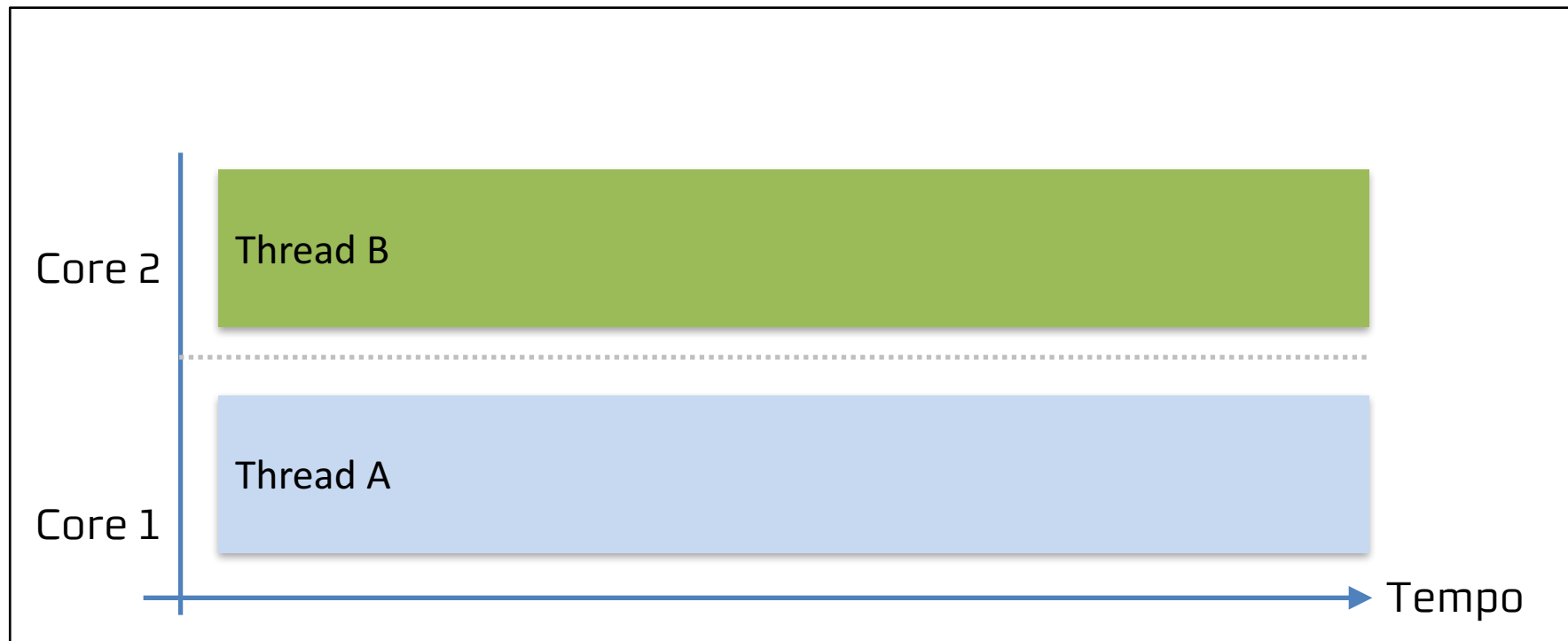
# Execução em paralelo



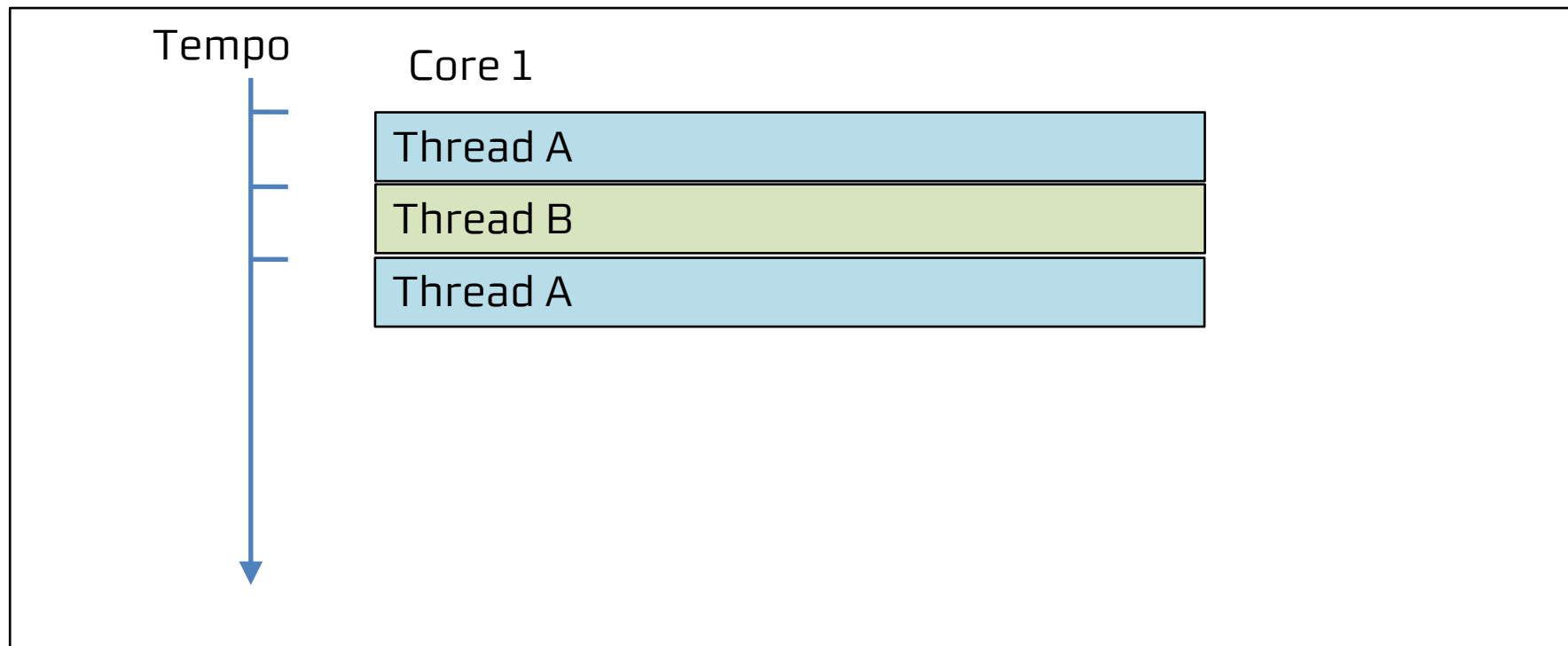
# Execução em paralelo



# Execução em paralelo



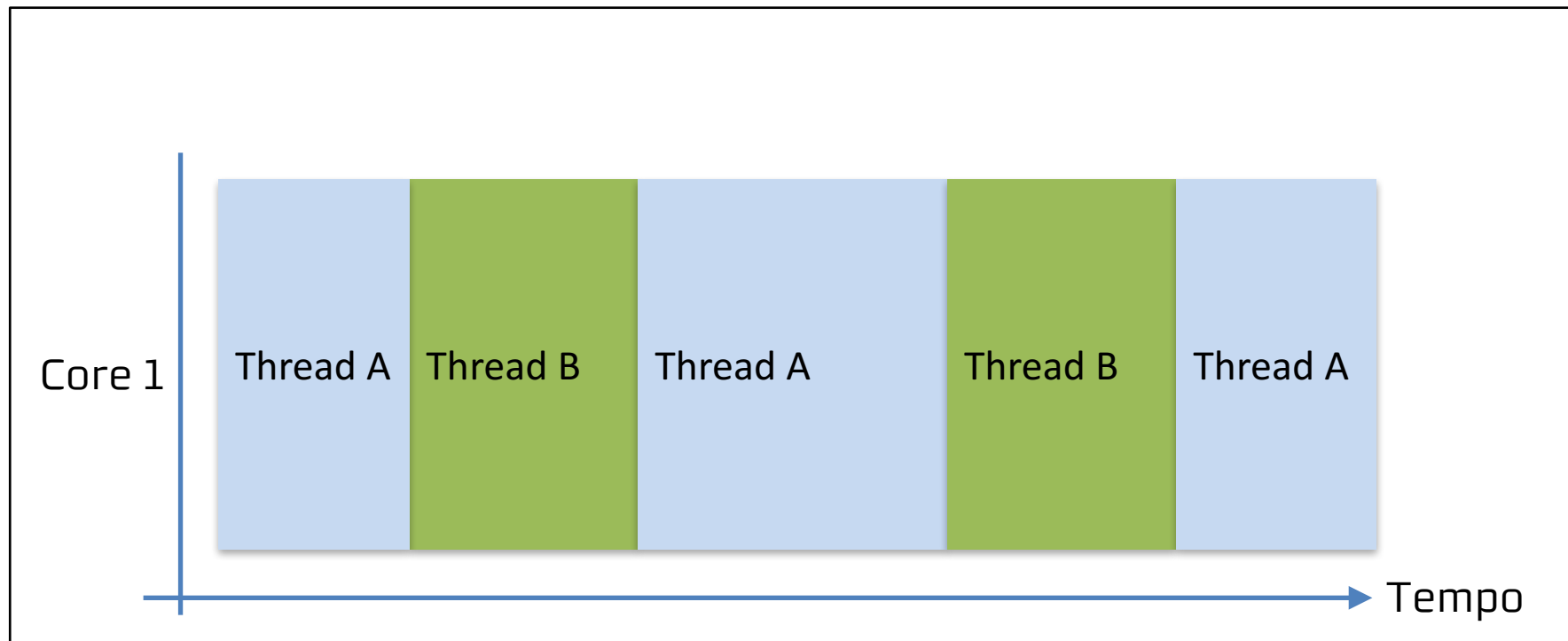
# Execução intercalada



# Execução intercalada

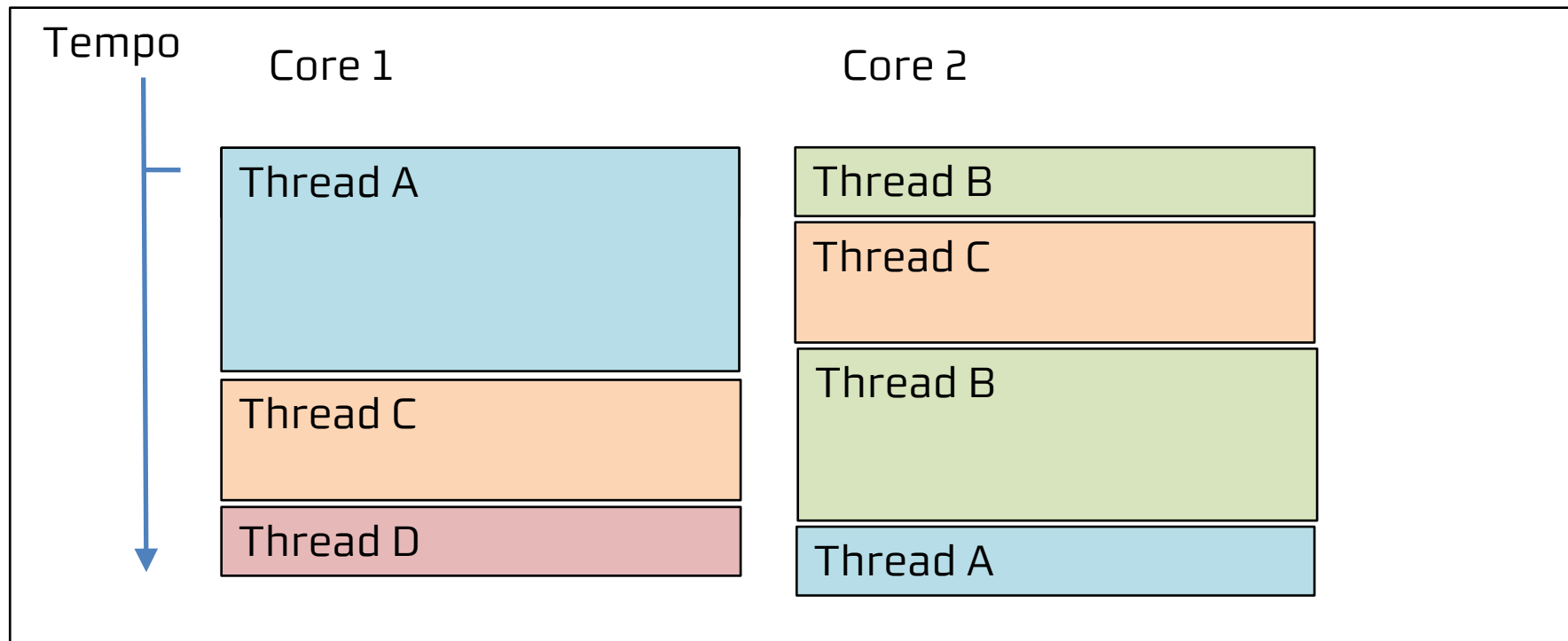


# Execução intercalada

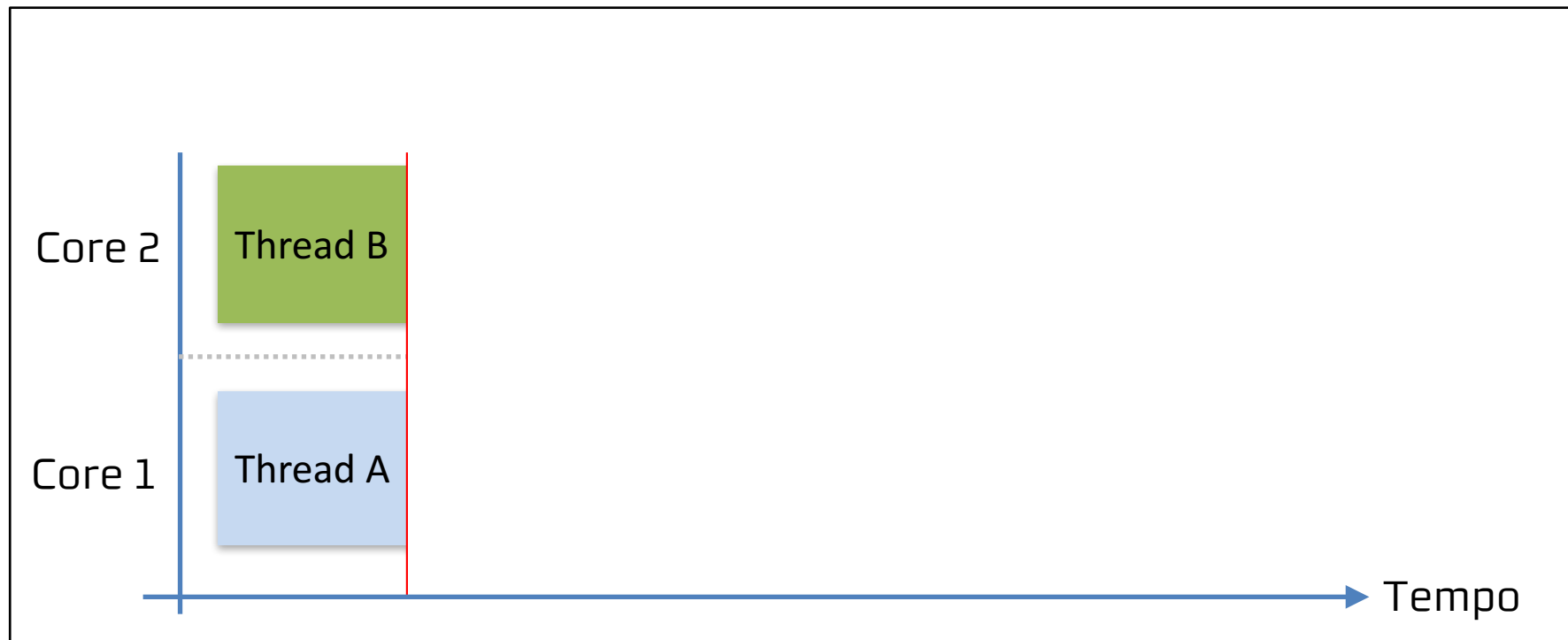




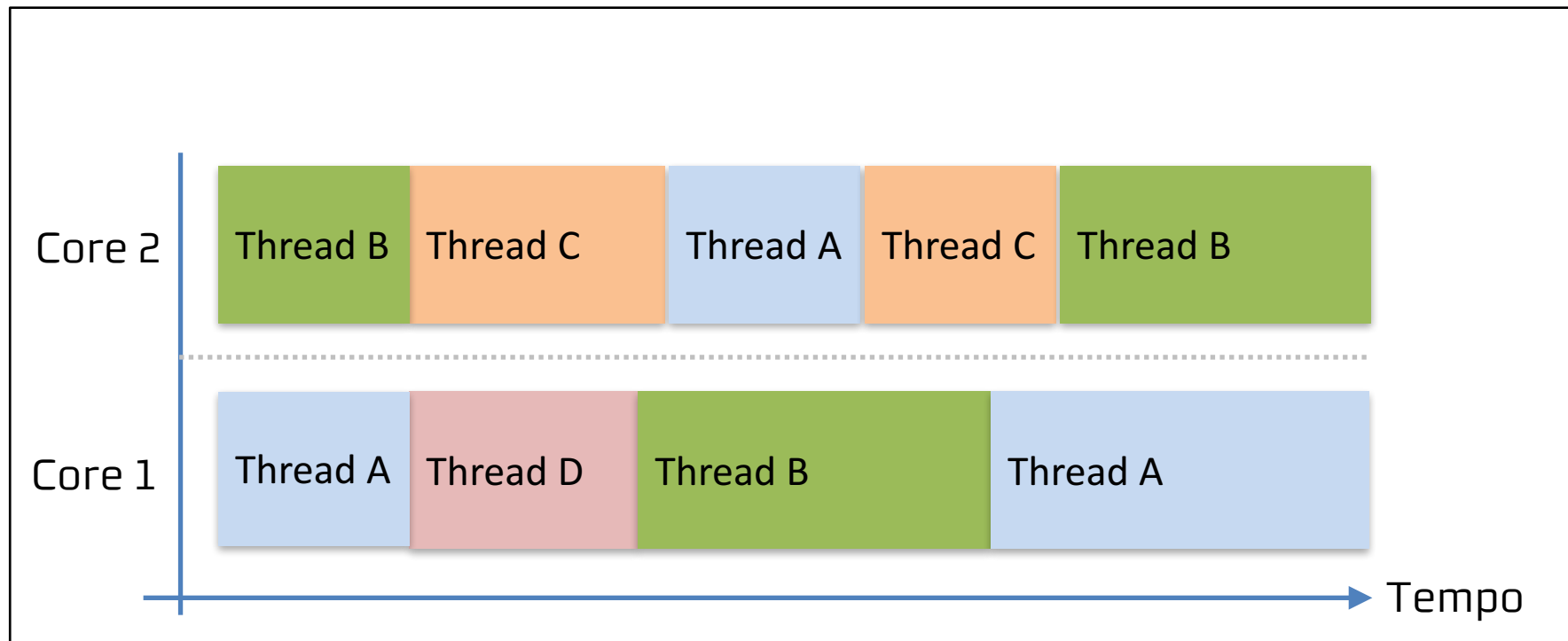
# Execução mista



# Execução mista



# Execução mista



# Ações Atômicas

- Uma thread é uma sequência de ações (instruções)
- Às ações que não podemos decompor, chamamos **ações atômicas**

# Paradigma de programação concorrente

- em qualquer instante:
  1. uma thread pode (re)começar e ser executada
  2. uma thread pode deixar de ser executada
- o programador **não controla** quando as threads são executadas

# Exemplo de 2 threads

- A ordem por que são executadas as ações das diferentes threads varia de execução para execução
- Os resultados podem ser diferentes cada vez que é executado o programa

# Exemplo de 2 threads

## Thread A

```
println("Sou a thread A")  
println("Adeus - A")
```

## Thread B

```
println("Sou a thread B")  
println("Adeus - B")
```

## Output:

```
Sou a thread A  
Adeus - A  
Sou a thread B  
Adeus - B
```

```
Sou a thread A  
Sou a thread B  
Adeus - A  
Adeus - B
```

```
Sou a thread A  
Sou a thread B  
Adeus - B  
Adeus - A
```

...

```
Sou a thread B  
Adeus - B  
Sou a thread A  
Adeus - A
```

# Multithreading em Java

- Em Java as threads são representadas pela classe **Thread**
- Iniciam apenas com o método **start()**
- Após iniciada, a nova thread vai executar o método **run()**



# Exemplo thread em Java

```
public class MyThread extends Thread {  
  
    public MyThread(String name) {  
        super(name);  
    }  
  
    @Override  
    public void run() {  
        System.out.println(getName());  
    }  
  
    public static void main(String[] args) {  
        MyThread m = new MyThread("A minha 1ª thread");  
        m.start();  
    }  
}
```

# Multithreading em Java

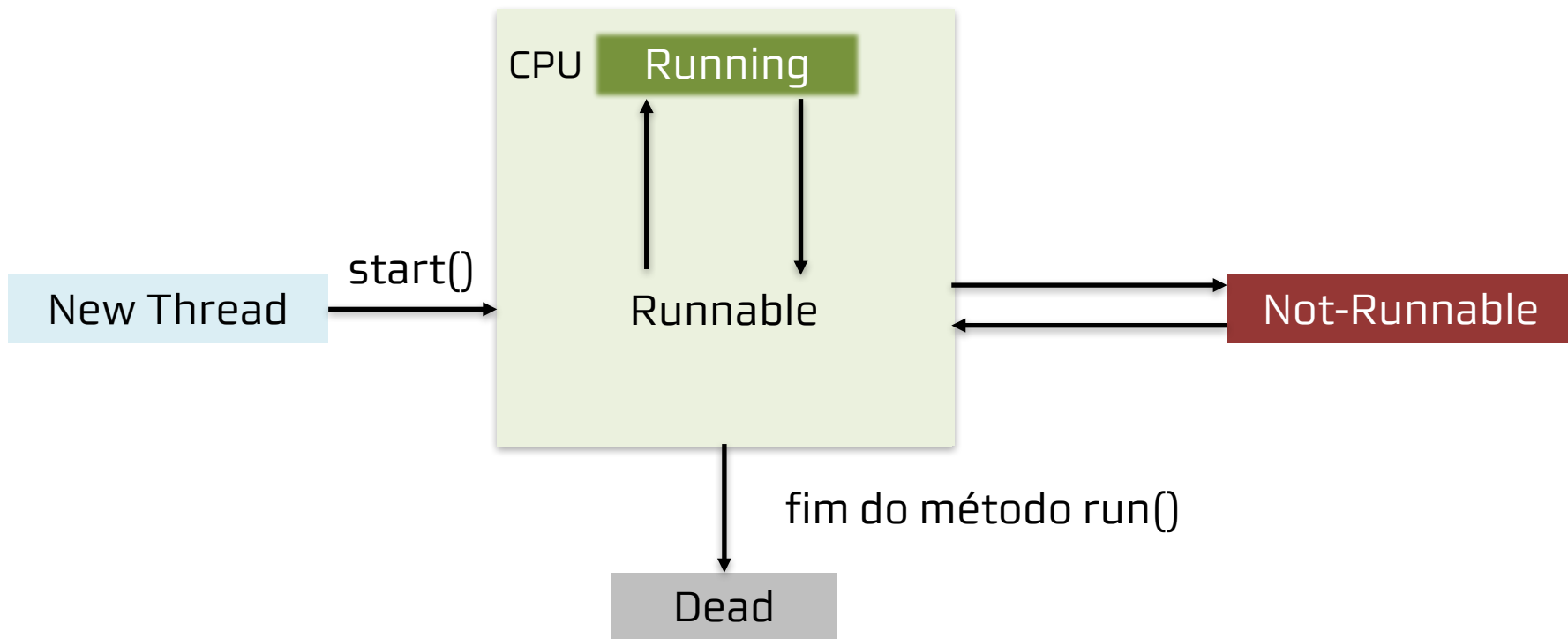
- Thread inicial é criada automaticamente e corre o método `main()`
- Restantes threads são iniciadas pelo programador
- As threads correm de uma forma concorrente

# Multithreading em Java

```
public class Main {  
    public static void main(String[] args) {  
        MyThread t1= new MyThread("T1");  
        t1.start();  
        ...  
    }  
}
```

```
public class MyThread extends Thread {  
    ...  
    @Override  
    public void run() {  
        System.out.println(i + " -> " + name);  
        ...  
    }  
}
```

# Ciclo de vida das Threads




# Interrupção de threads

- Mecanismo para indicar que uma thread deve parar de fazer o que está a fazer para fazer outra tarefa.
- Uma thread pode interromper outra usando o método `interrupt`.
- O programador escolhe o que fazer.

# Exemplo da interrupção

```
public class MyThread extends Thread {  
    public void run() {  
        try {  
            while(true){  
                System.out.println(currentThread() + ": sleep for 4 seconds");  
                sleep(4000);  
            }  
        } catch (InterruptedException e) {  
            System.out.println(currentThread() + ": Done! I was interrupted!");  
        }  
    }  
}  
  
public static void main(String args[]) throws InterruptedException {  
    Thread t = new MyThread ();  
    t.start();  
  
}
```



# Pausar uma thread (sleep)

```
static void Thread.sleep(time)
```

```
throws InterruptedException
```

- Método de classe que pausa a execução da thread que está a correr durante um tempo definido.

# Esperar que uma thread termine

```
static void join()
```

```
throws InterruptedException
```

- Espera que uma thread termine.



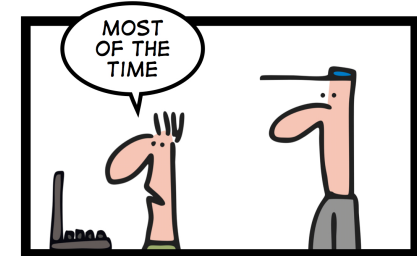
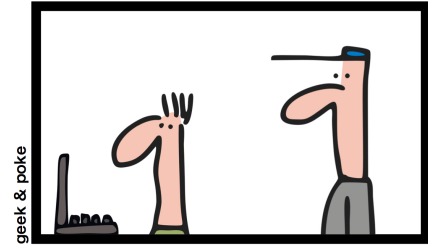
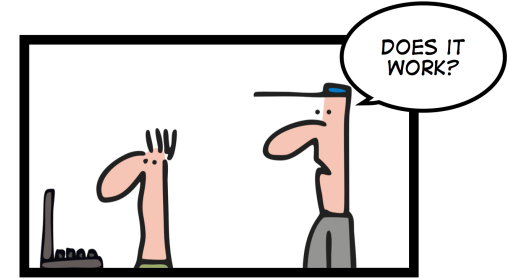
# Exemplo join

```
public class MyThread extends Thread {
    public void run() {
        try {
            for(int i = 0; i < 10; i++){
                System.out.println(currentThread() + " " + i);
                sleep(2000);
            }
        } catch (InterruptedException e) {}
    }
    public static void main(String args[]) throws InterruptedException {
        Thread t = new MyThread ();
        t.start();
        try {
            t.join();
        } catch (InterruptedException e) {}
        system.out.println("Main done, all done!");
    }
}
```

# Sumário

- O que é uma thread
- Partilha de memória
- Escalonamento de threads
- Threads em Java
- Ciclo de vida de uma Thread

SIMPLY EXPLAINED



CONCURRENCY