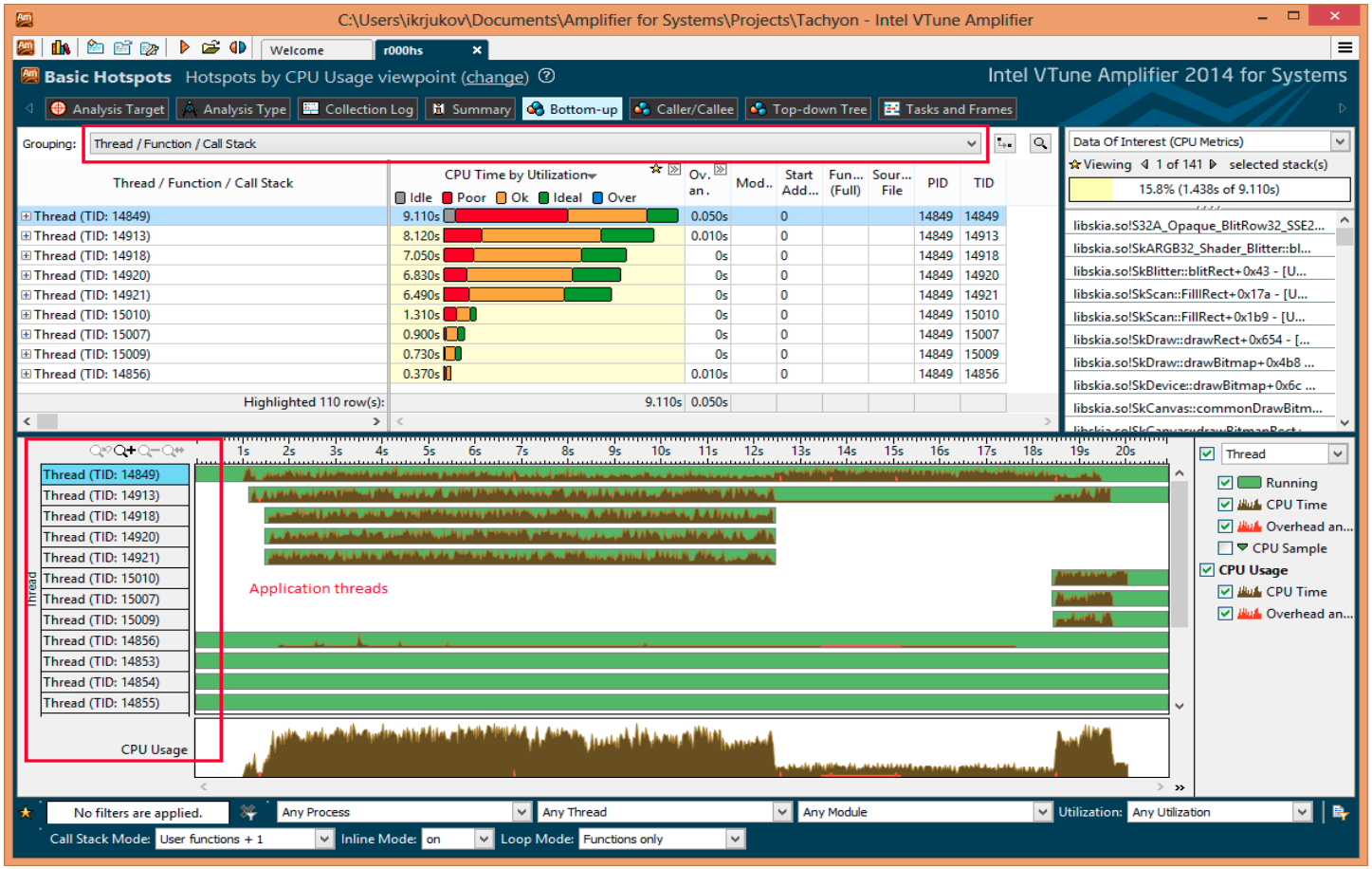
Programação Concorrente em Java

Profa Andréa Schwertner Charão

DLSC/CT/UFSM

O que é programação concorrente?

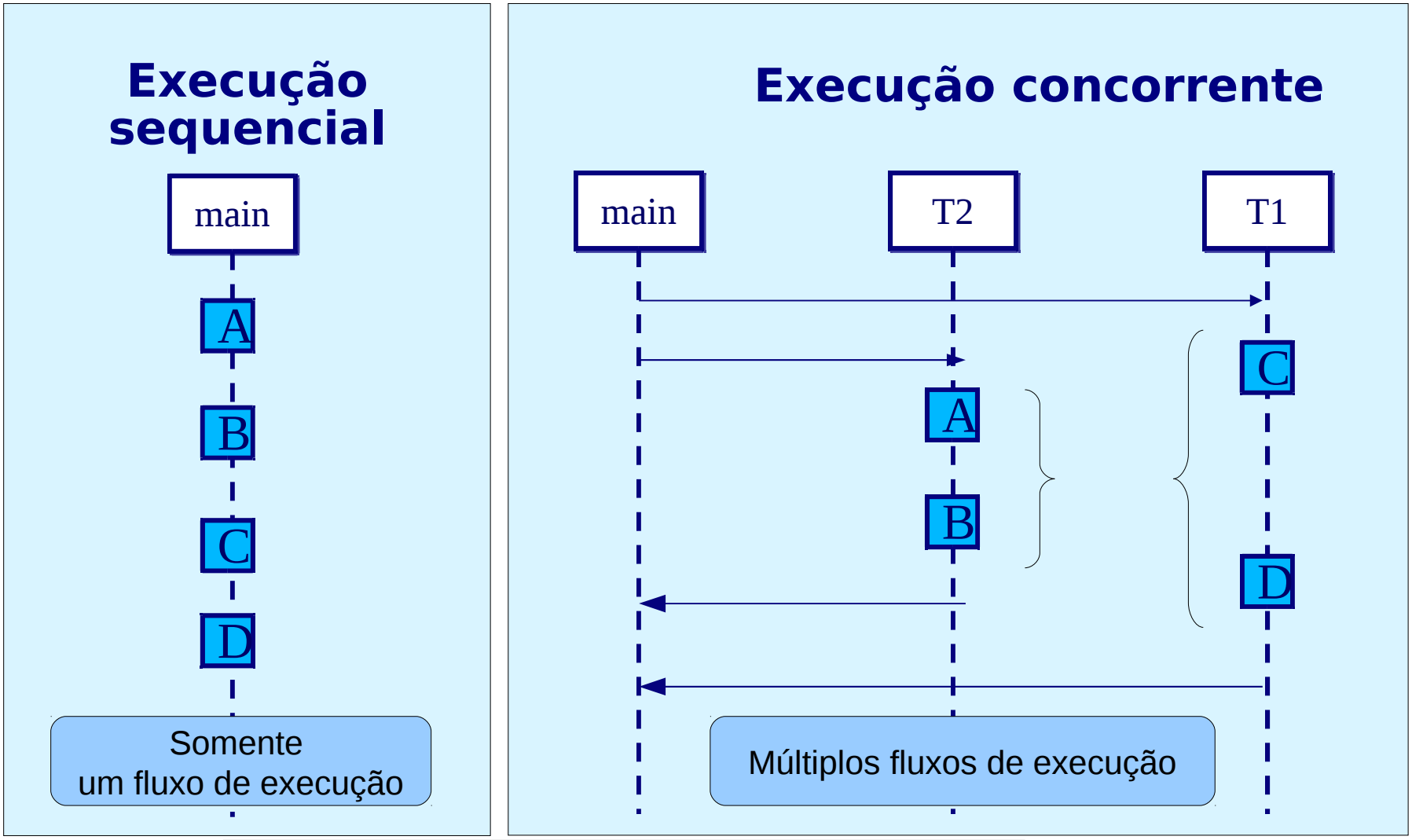
- Um programa, múltiplos fluxos de execução



Quando usar programação concorrente?

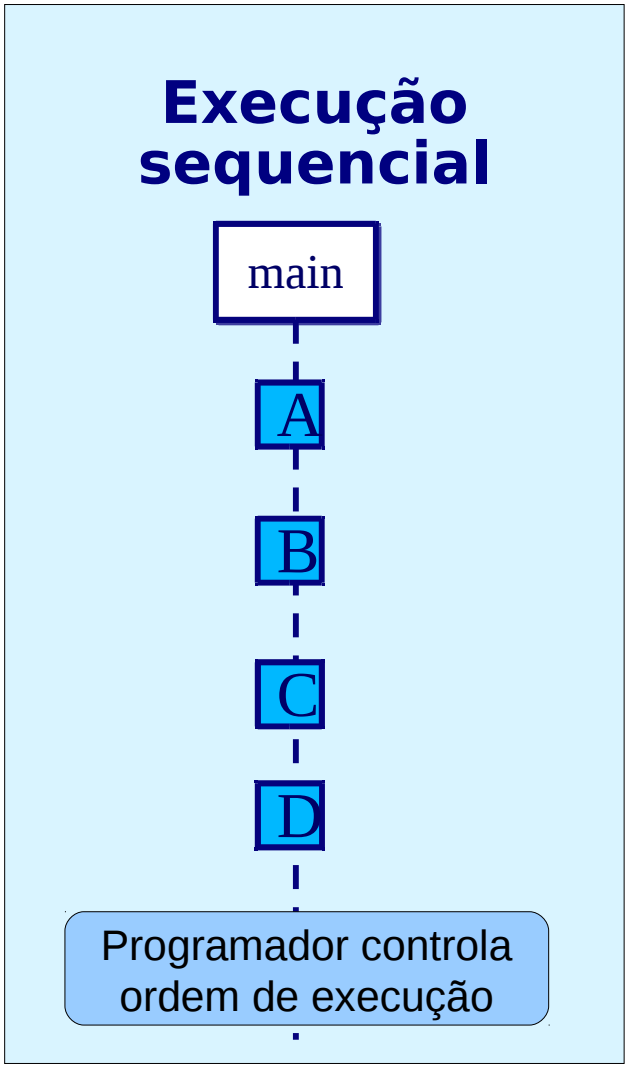
- Desempenho
  - Ex.: game engines, bancos de dados, etc.
- Aproveitamento de arquitetura
  - Ex.: multicore
- Disponibilidade/reatividade
  - Ex.: servidores Web, interfaces gráficas
- Organização de código
  - Tarefas independentes em threads independentes

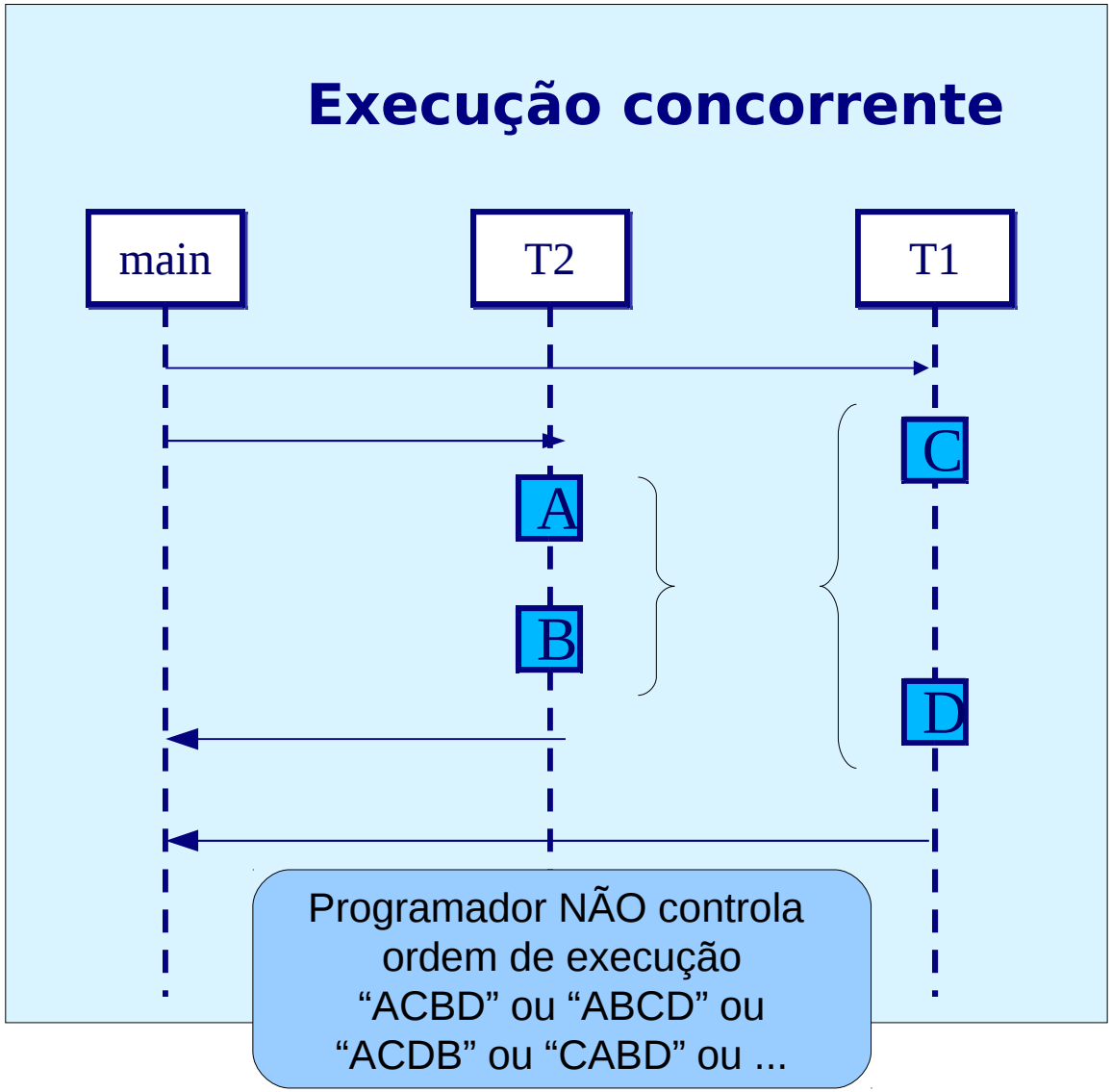
Execução concorrente



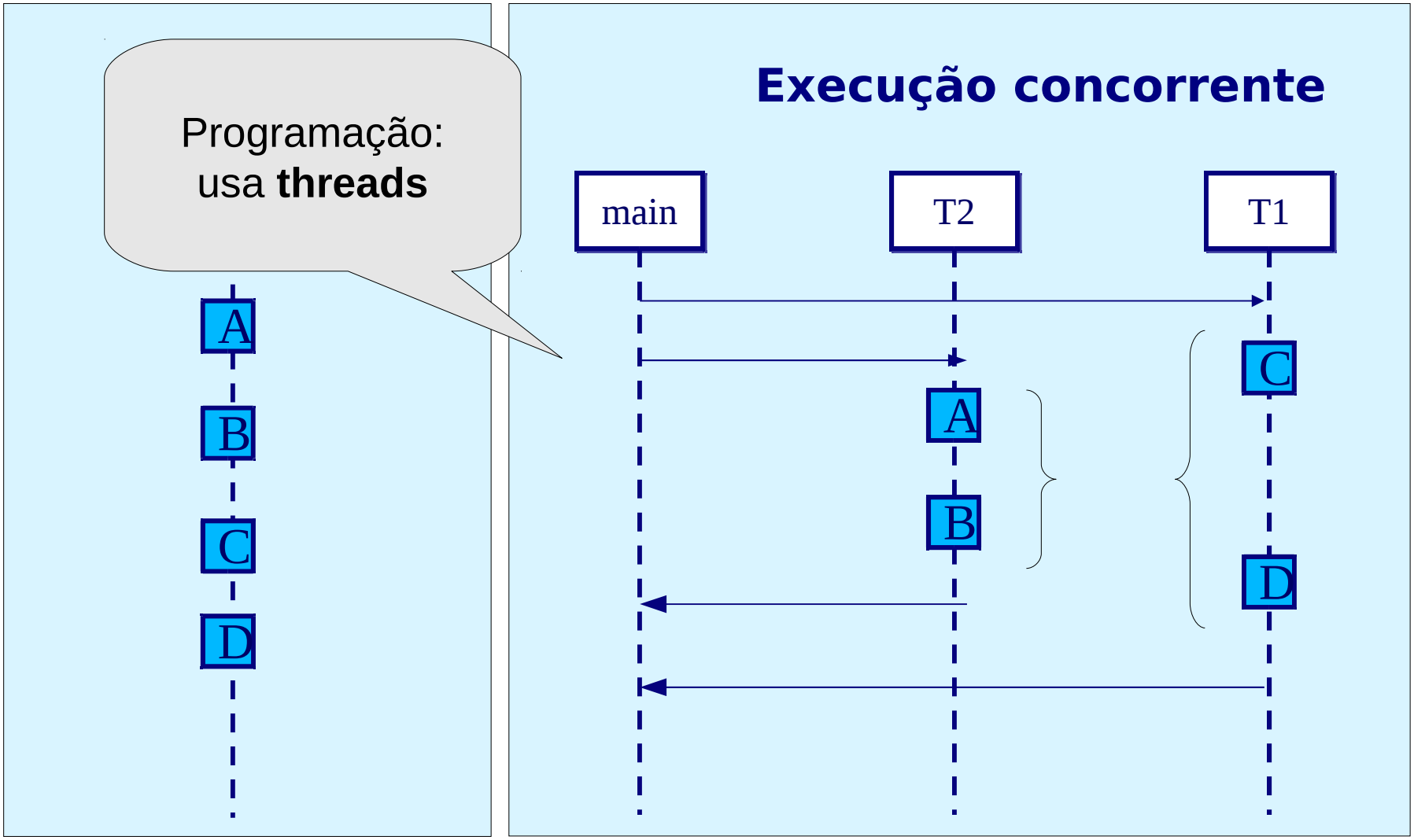
Evolução da execução no tempo

Execução concorrente



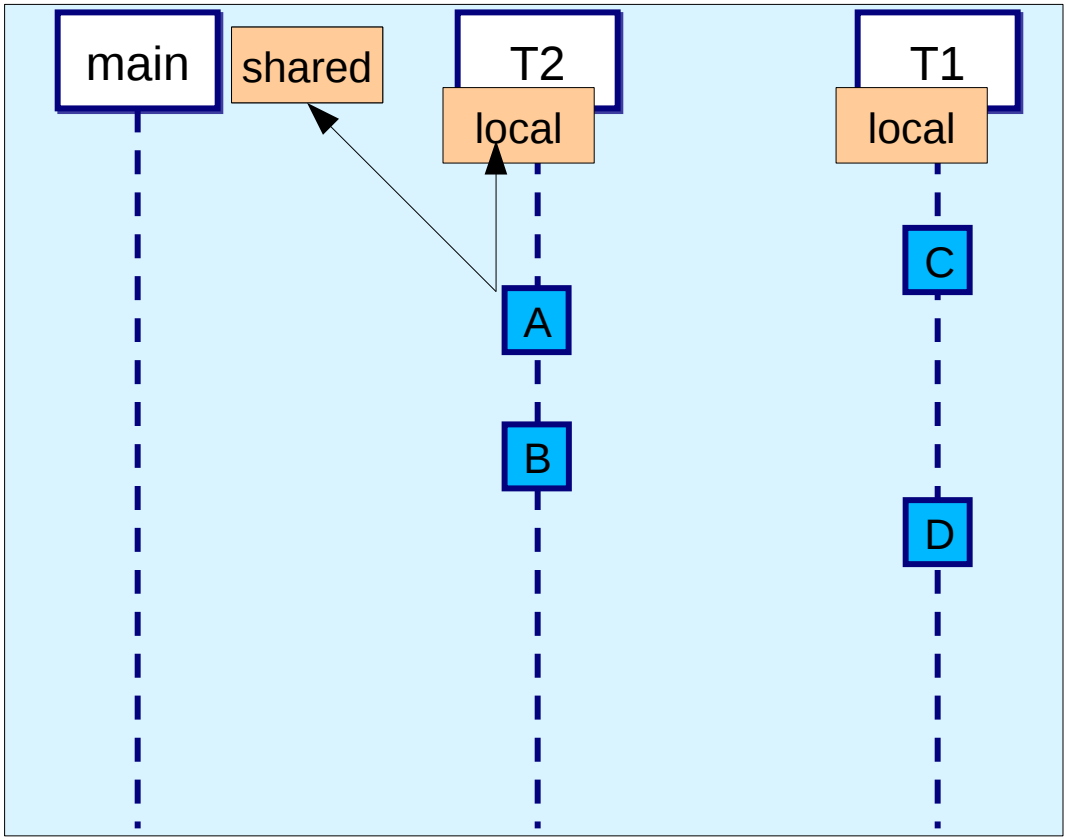


Execução concorrente



Threads

- Fluxos de execução independentes em um **único programa**
  - Contadores de programa independentes
- Possuem área de memória própria
  - Variáveis locais
- Compartilham memória comum às outras threads do programa



Como programar com threads?

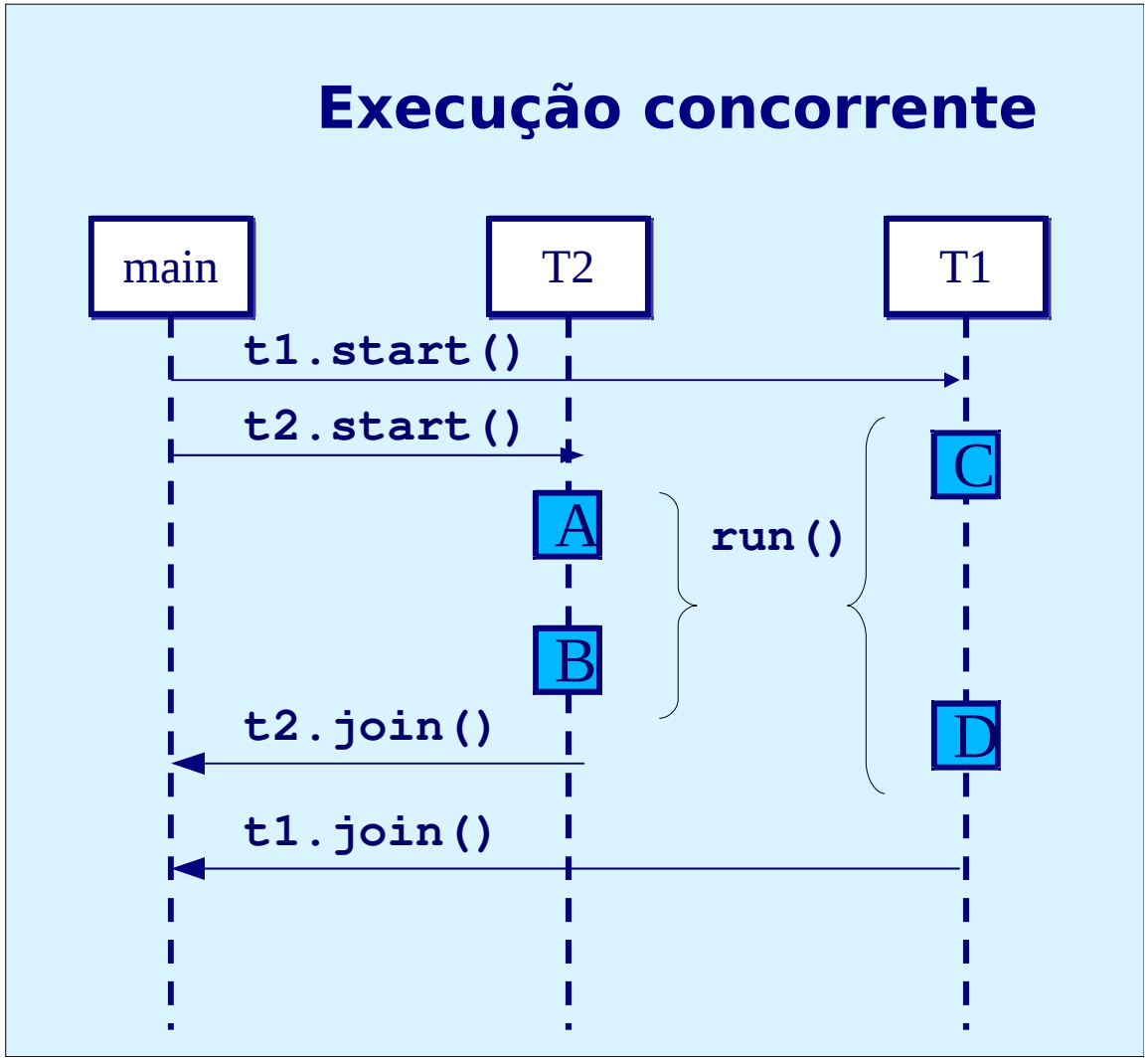
- Especificação do **código** que deve ser executado
- Especificação de **dados** próprios ou compartilhados
- Controle: ativação, término, suspensão, etc.
- Sincronização (competição ou cooperação) entre threads, quando necessário

Programação com threads em Java

- Classe Thread
  - Métodos start() e join()
- Interface Runnable e método run()
- Palavra-chave synchronized
- Etc.

Veja mais em:
http://download.oracle.com/javase/tutorial/essential/concurrency/

Programação com threads em Java



Programação com threads em Java

- Duas opções:
  1) Declarar classe que estende Thread
  2) Declarar classe que implementa interface Runnable

```
class MyThread extends Thread {
   public void run() {
      System.out.println("Trabalhando");
   }
}

class MyRunnable implements Runnable {
   public void run() {
      System.out.println("Trabalhando");
   }
}
```

Programação com threads em Java

- Se estender Thread, o uso da classe fica assim:

```
class ThreadApp {
   public static void main(String[] args) {
      MyThread t1 = new MyThread();
      Thread t2 = new MyThread();
      t1.start();
      t2.start();
   }
}

   class MyThread extends Thread {
      public void run() {
         System.out.println("Trabalhando");
      }
   }
```

Programação com threads em Java

- Se implementar Runnable, o uso da classe fica assim:

```
class ThreadApp {
   public static void main(String[] args) {
      MyRunnable r = new MyRunnable();
      Thread t1 = new Thread(r);
      Thread t2 = new Thread(r);
      t1.start();
      t2.start();
   }
}
    class MyRunnable implements Runnable {
       public void run() {
          System.out.println("Trabalhando");
       }
    }
```

Programação com threads em Java

- Exemplo: Android

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            final Bitmap bitmap = loadImageFromNetwork("http://example.com/image.png")
            mImageView.post(new Runnable() {
                public void run() {
                    mImageView.setImageBitmap(bitmap);
                }
            });
        }
    }).start();
}
```

Fonte:
http://developer.android.com/guide/components/processes-and-threads.html

Programação com threads em Java

- Método join() espera pelo término da thread

```
class ThreadApp {
   public static void main(String[] args) {
      MyThread t1 = new MyThread();
      Thread t2 = new MyThread();
      t1.start();
      t2.start();
      try {
         t1.join();
         t2.join();
      } catch (InterruptedException e) {
         // trata interrupcao
      }
   }
}
```

Tratamento de exceção!

Threads e exceções em Java

- Alguns métodos da classe Thread exigem manipulação de exceções
- Por exemplo:
  - join() throws InterruptedException:
    método para esperar pelo término da thread
  - sleep(long millis) throws InterruptedException:
    método para interromper execução da thread durante um certo tempo
- Com esses métodos:
  - OU capturar e tratar a exceção usando try/catch
  - OU passar a exceção adiante usando throws

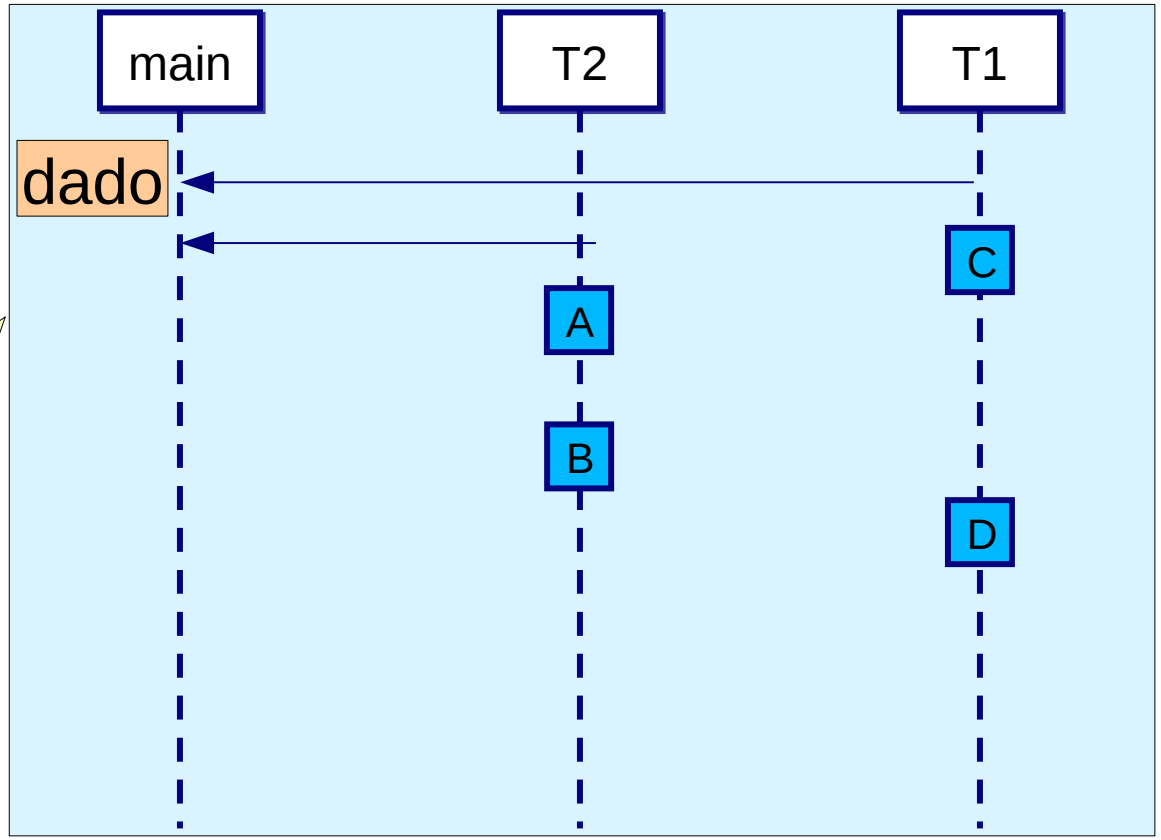
Sincronização entre threads em Java

- Competição no acesso a dados **compartilhados**

Variável de classe (static)

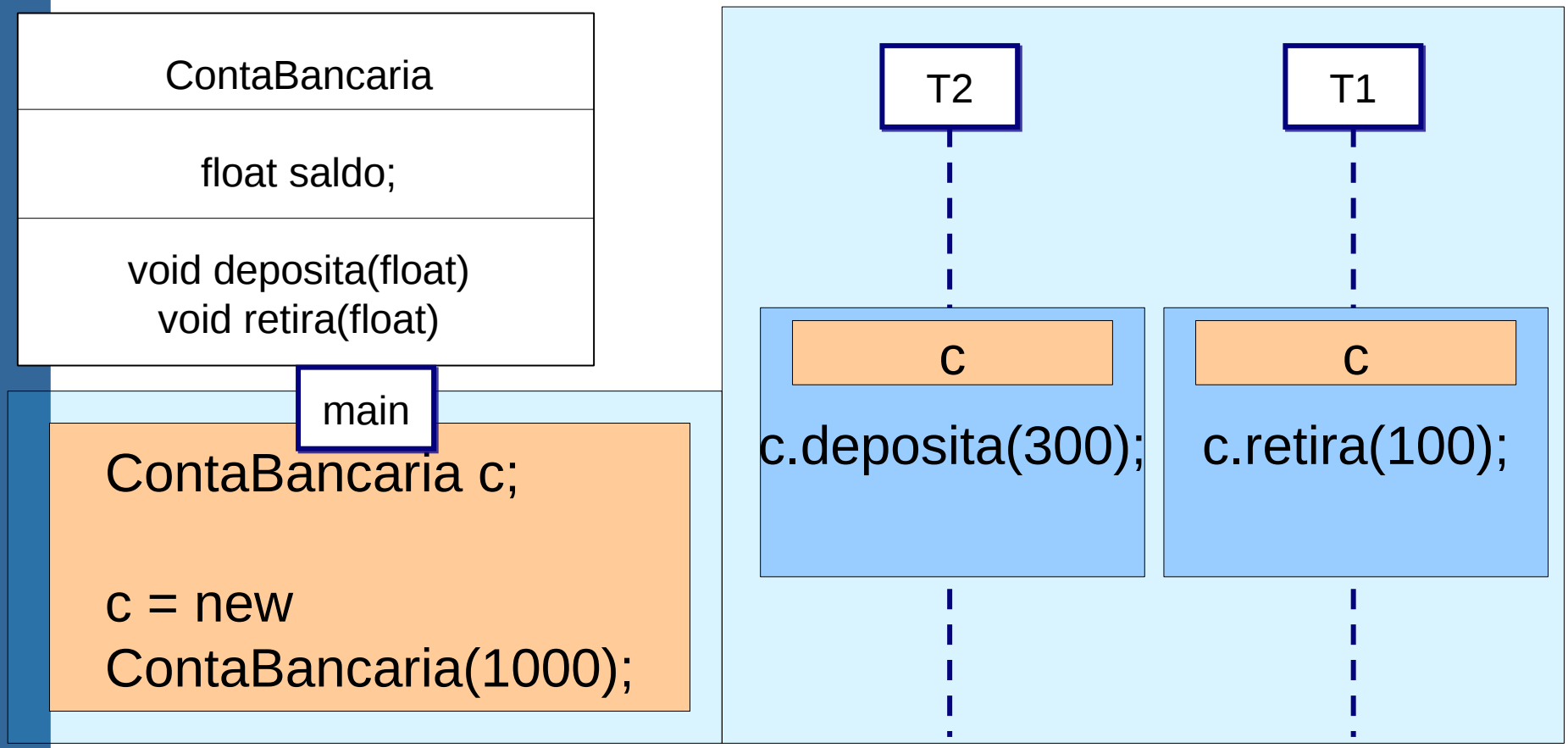
E/OU

Referência para um mesmo objeto



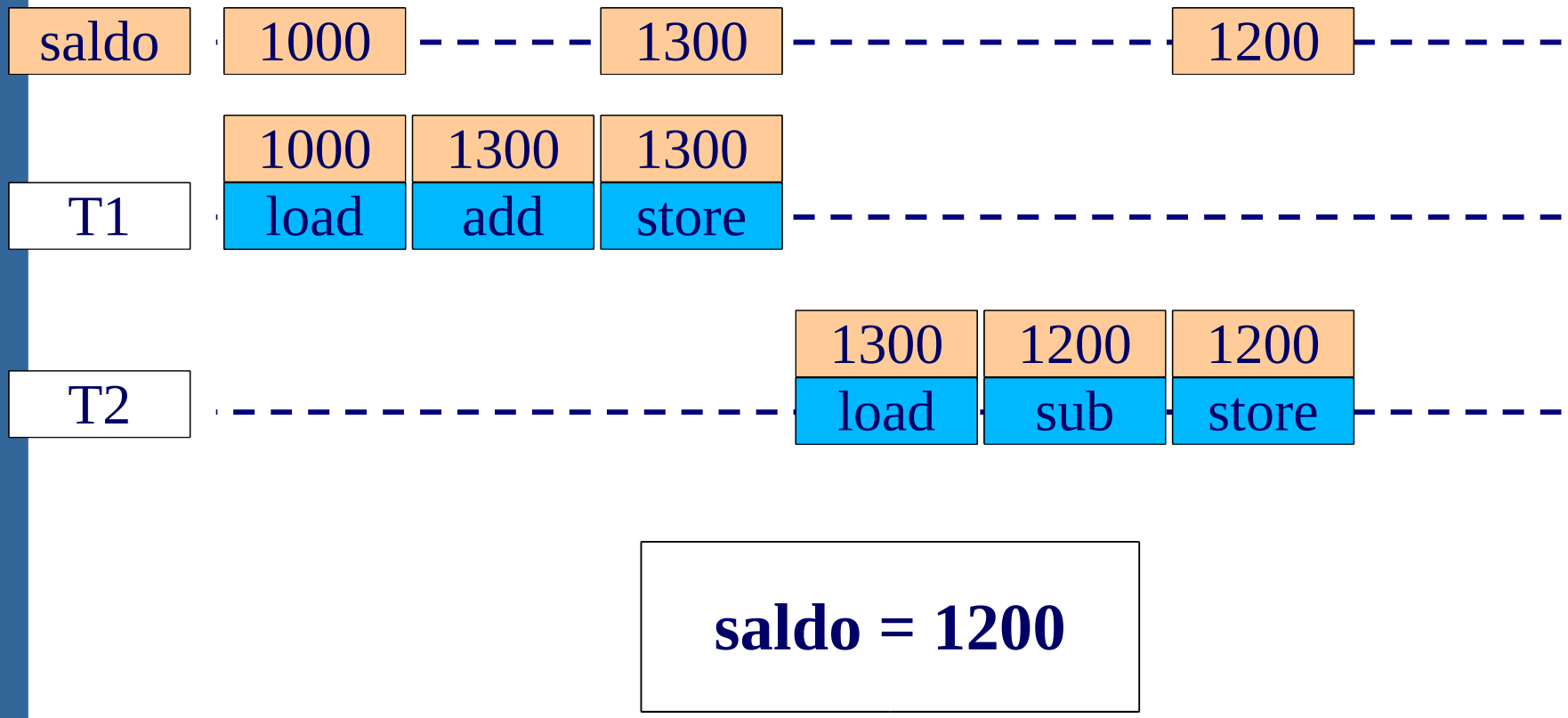
Sincronização entre threads

- Situação típica: 2 ou mais threads fazem operação sobre dado compartilhado



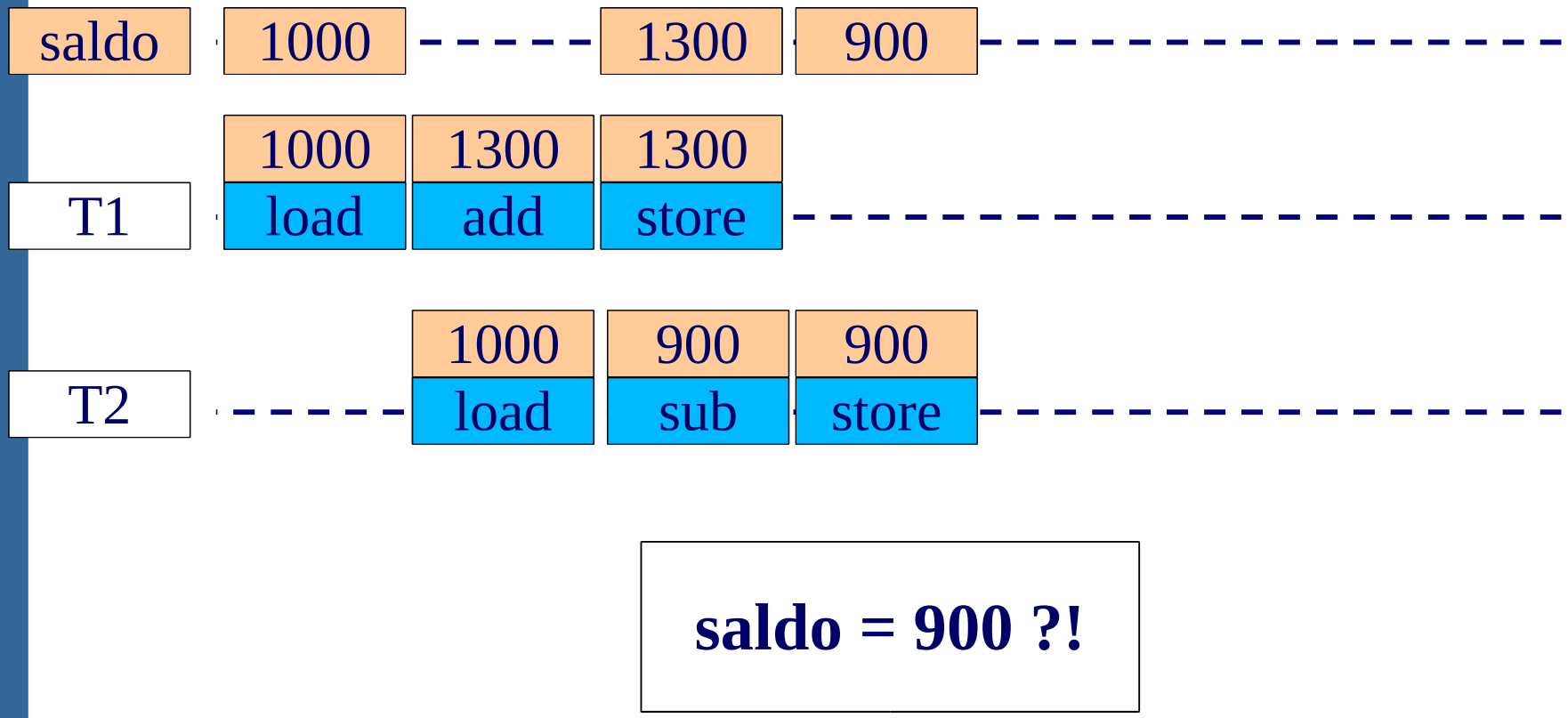
Sincronização entre threads

- Situação 1: T1 primeiro, T2 depois



Sincronização entre threads

- Situação 2: T1 e T2 concorrentemente



Sincronização entre threads

- Problema: inconsistência de dados
- Em outras palavras...
  - **Condição de corrida**: operações sobre recursos compartilhados que podem levar a inconsistências dependendo da ordem de execução
  - **Seção crítica**: trecho do programa que contém operações que podem levar a inconsistências
- Solução: exclusão mútua
- Em outras palavras...
  - Garantir atomicidade e acesso exclusivo ao recurso na seção crítica

Exclusão mútua em Java

- Compartilhamento do objeto

```
public static void main(String[] args) {

   Conta c = new Conta(100f);

   ThreadDeposita td = new ThreadDeposita(c);
   ThreadRetira   tr = new ThreadRetira(c);

   td.start();
   tr.start();

}
```

Exclusão mútua em Java

- Thread faz acesso ao objeto

```
class ThreadDeposita extends Thread {
   private Conta c;

   ThreadDeposita(Conta c) {
      this.c = c;
   }
   public void run() {
      for (int i = 0; i < 10; i++)
         c.deposita(100f);
   }
}
```

Exclusão mútua em Java

- Uso de **synchronized**

```
class ContaBancaria {
  private float saldo;
  ContaBancaria(float v) {
    saldo = v;
  }
  synchronized void deposita(float v) {
    saldo += v;
  }
  synchronized void retira(float v) {
    saldo -= v;
  }
}
```

Métodos **synchronized** executam em exclusão mútua sobre o mesmo objeto compartilhado

Exclusão mútua em Java

- Métodos synchronized em um mesmo objeto são executados em exclusão mútua
- Só têm efeito em objetos compartilhados (mais de 1 thread referenciando mesmo objeto)
- Limitação à concorrência
- Usar com cuidado
- Em S.O. serão vistos outros mecanismos de exclusão mútua