Threads/Concurrency API

Treinamento Arquitetura e

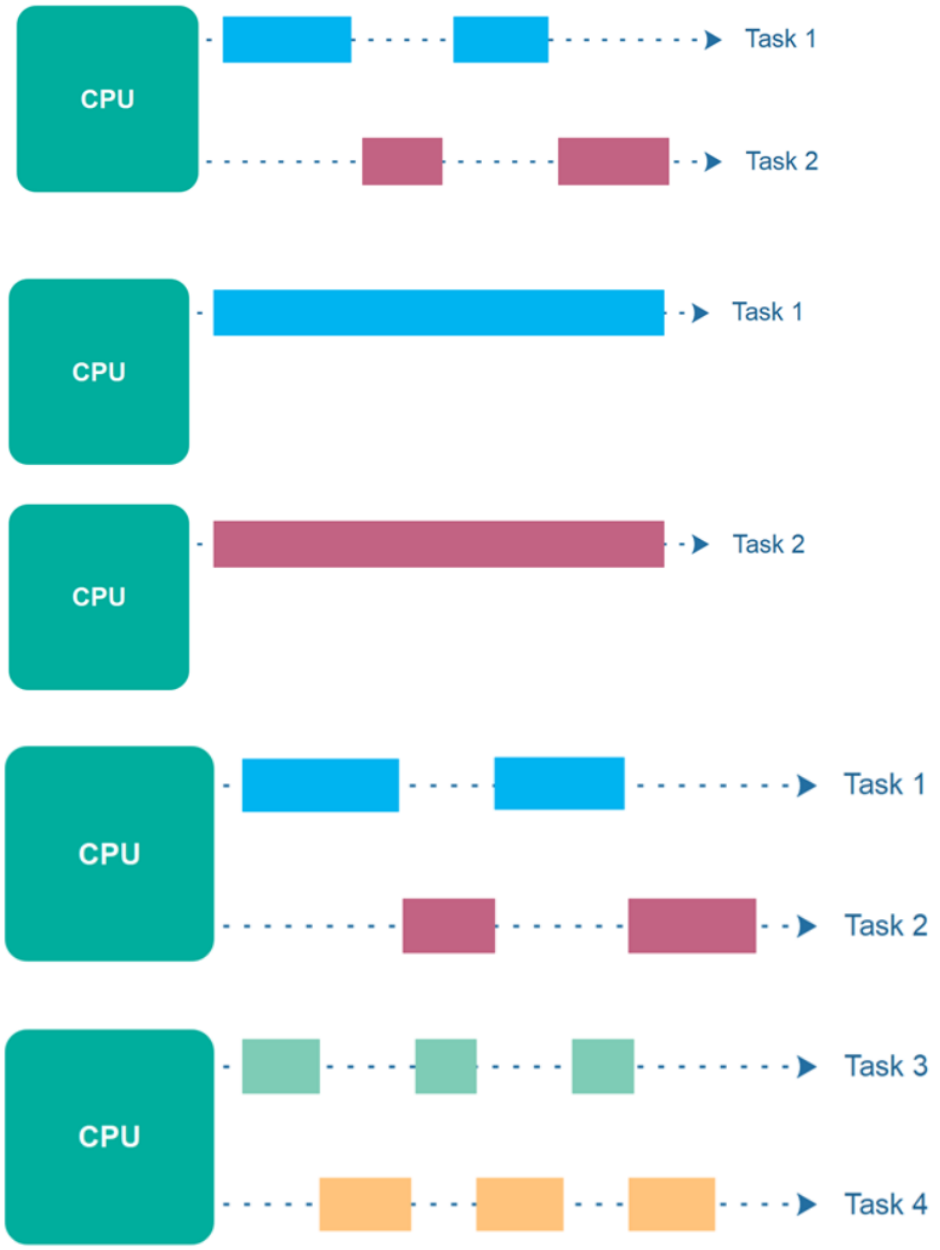
Desenvolvimento de Software

Tópicos

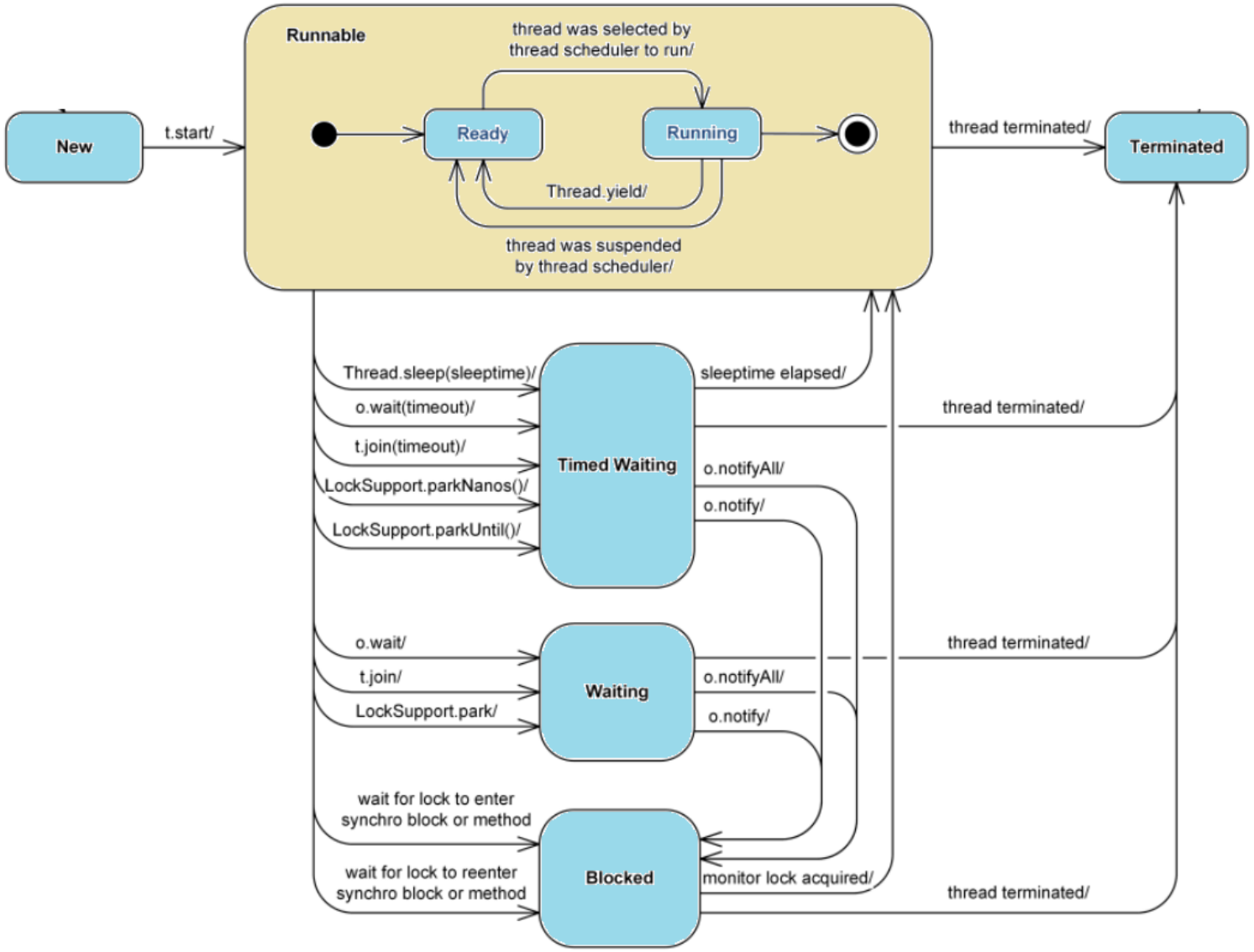
- Concorrência e Paralelismo
- Classe Thread e Interface Runnable
- Modelo de Memoria
- Problema de Visibilidade (volatile|synchronized)
- Problemas Comuns
- Java Concurrency API

Concorrência e Paralelismo

- Concorrência
  - Habilidade de um programa executar multiplas tarefas “simultaneamente”
- Paralelismo
  - Habilidade de um programa executar multiplas tarefas “ao mesmo tempo”

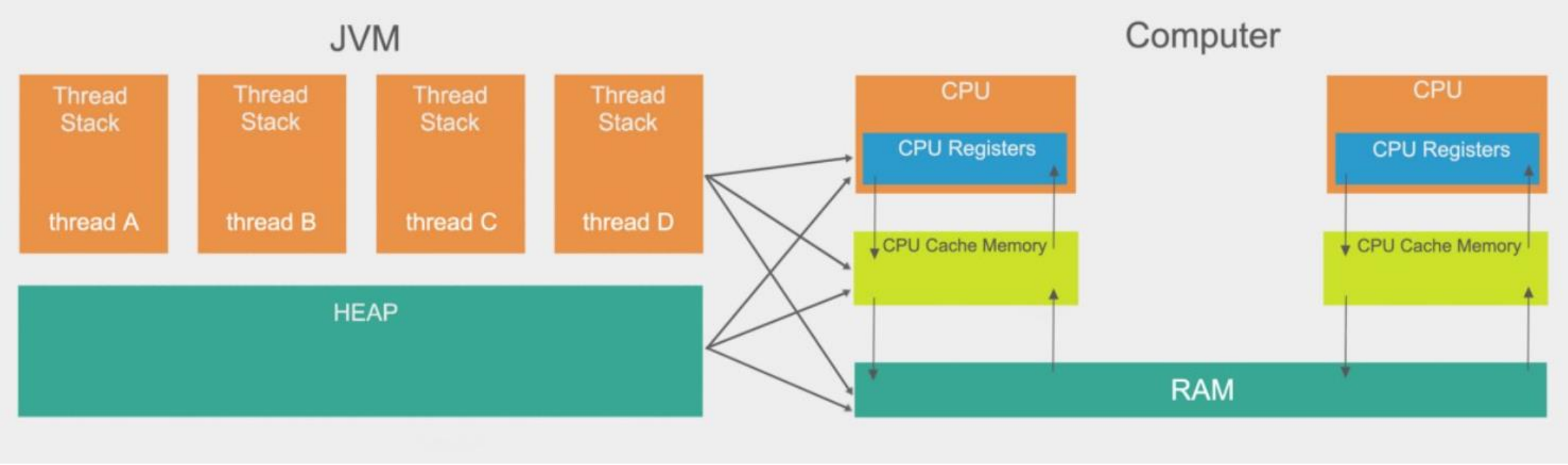


Class Thread

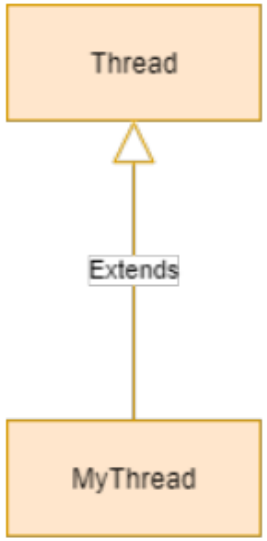


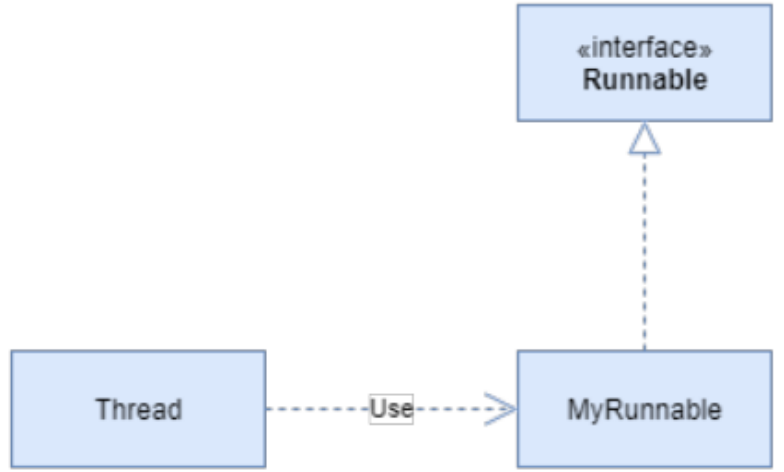
Modelo de Memoria

Cada Thread possui sua area de pilha



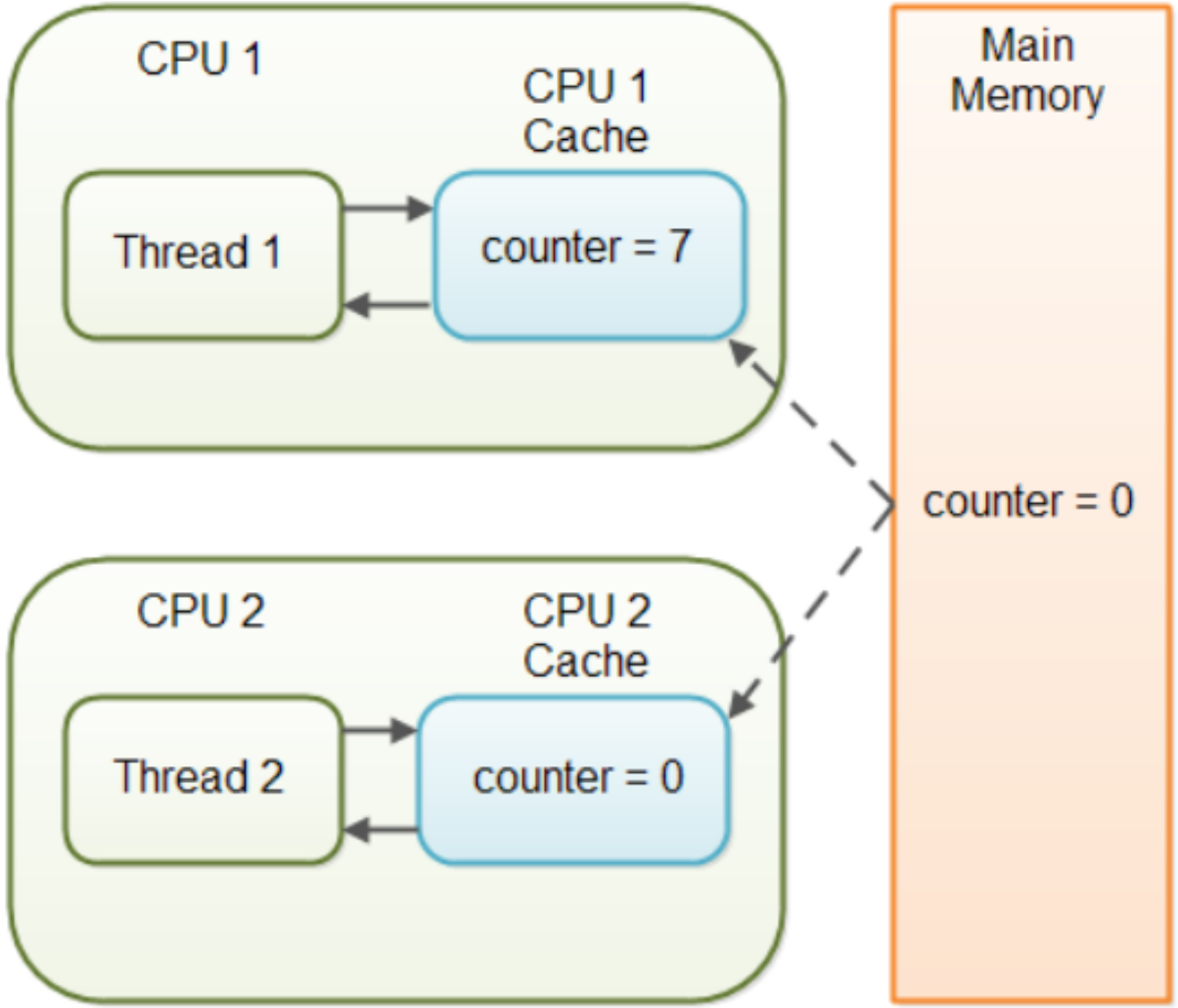
Criando uma Thread





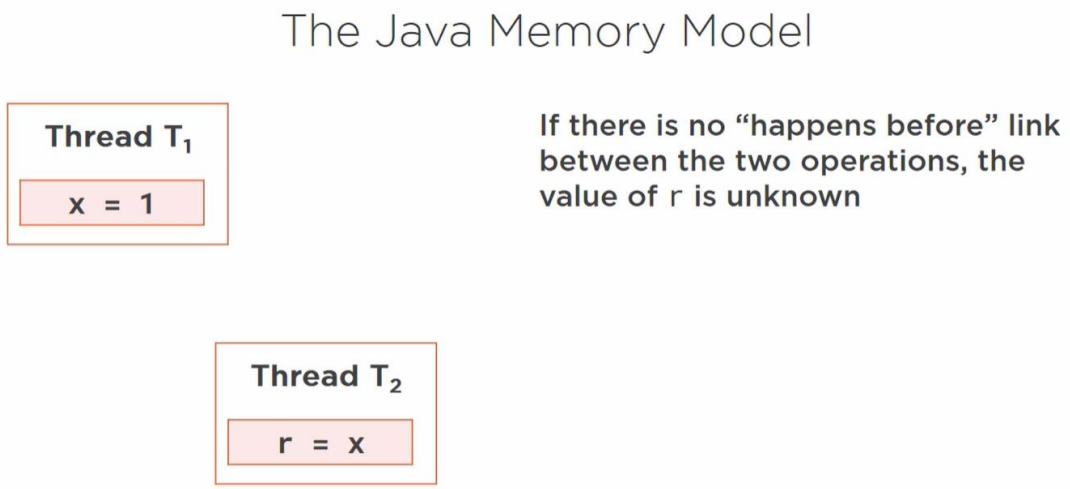
Problema de Visibilidade

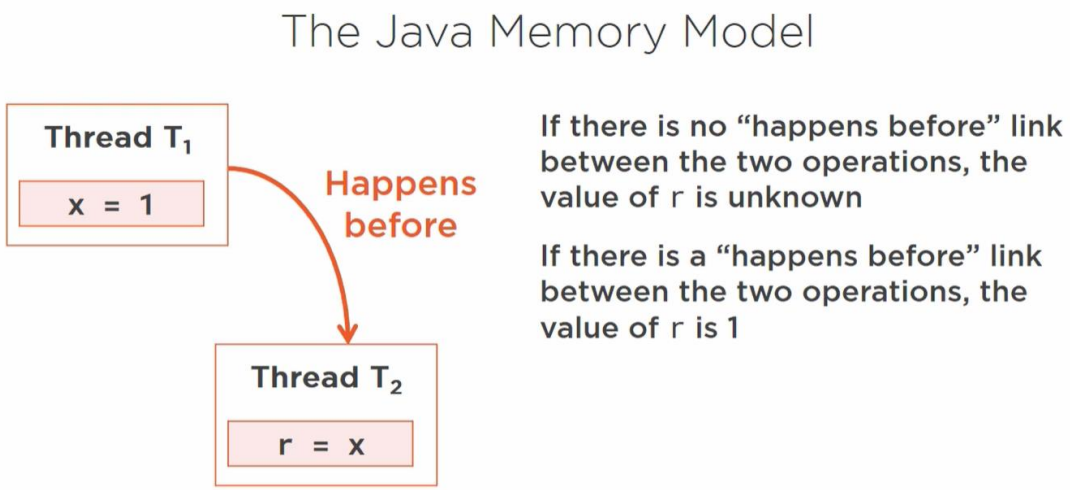
- Quando uma Thread não enxerga o último valor de uma variável atualizada por outra Thread
- Visibilidade
  - Uma leitura (read) deveria retornar o valor da última escrita (write)



Happens Before

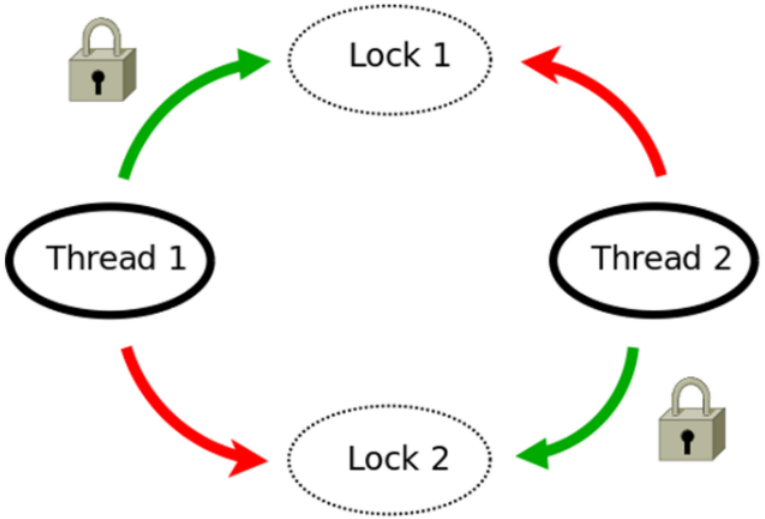
- “Happens Before” entre duas ou mais Threads garante que o recurso compartilhado possui as operações de leitura e escrita consistentes
- Blocos e Metodos sincronizados
  - synchronized
- Declaração volatile





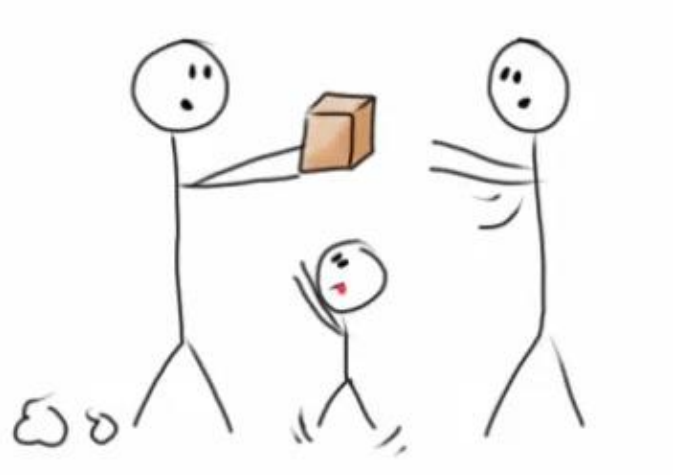
Problemas

- Deadlock
  - Threads estão esperando por recurso bloqueado por outra thread e por isso não podem prosseguir



Problemas

- Starvation
  - Ocorre quando um bloco/metodo sincronizado leva muito tempo pra ser executado, neste caso se outra thread precisa executar este bloco com frequencia, podera ficar bloqueada frequentemente
- Causas
  - Threads são bloqueadas indefinidamente porque uma thread leva muito tempo pra executar um bloco/metodo sincronizado
  - Uma thread tem pouco tempo de CPU devido a baixa prioridade comparada a outras threads

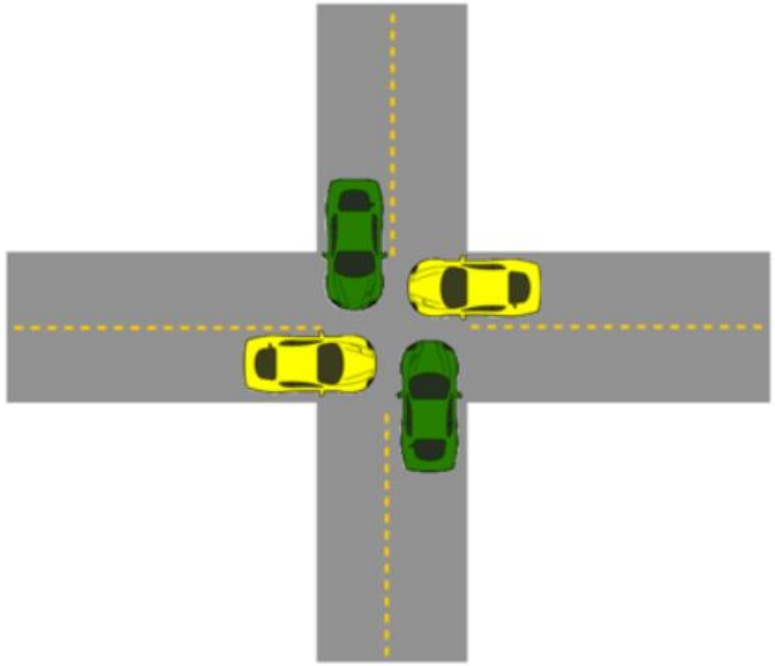


Some thread is not getting anything

Problemas

- Livelock
  - Ocorre quando duas threads estão esperando uma pela outra e não conseguem progredir
  - Normalmente as threads estão repetindo parte do codigo (logica)
  - Uma thread age em resposta a ação de outra thread e vice-versa

Livelock



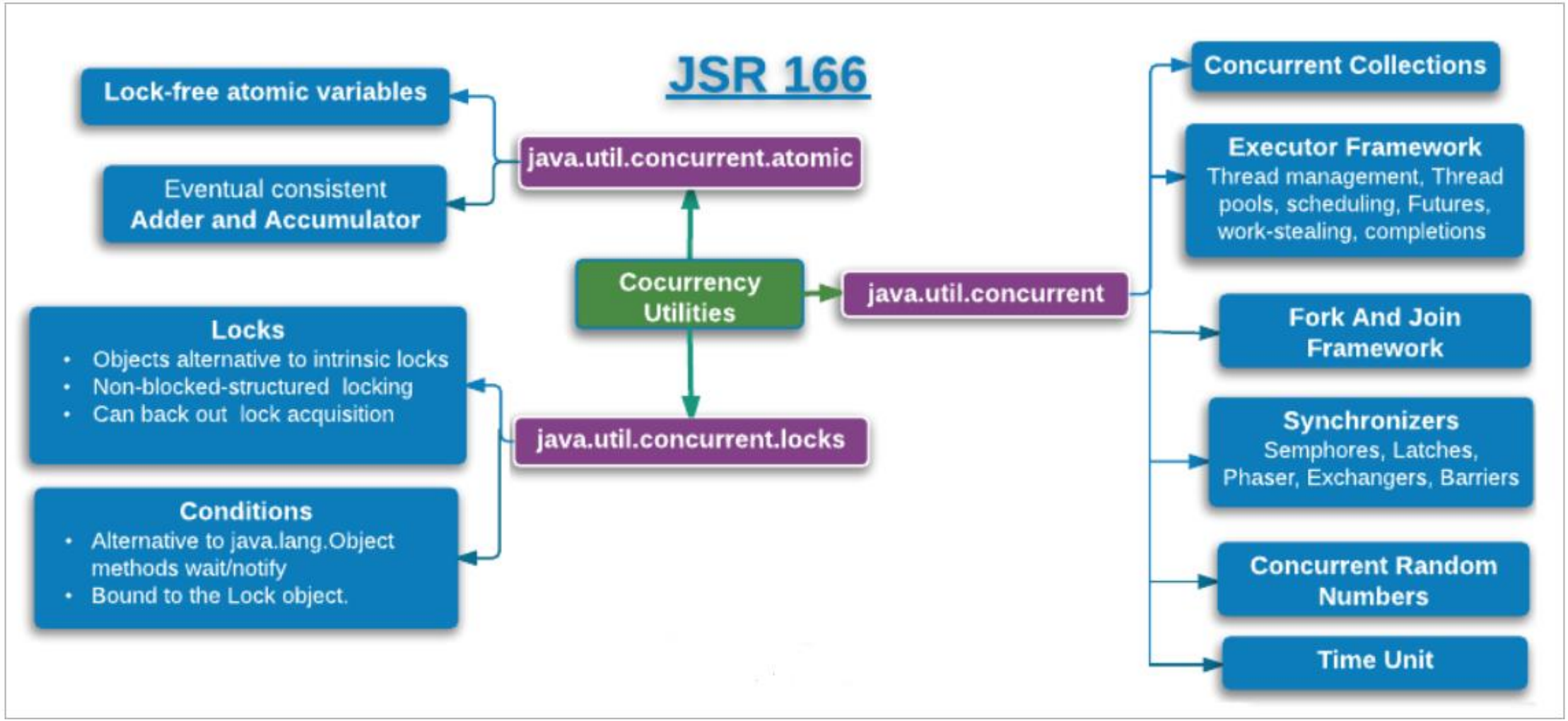
Livelock is a state where a system is executing many operations, but no thread is making meaningful progress.

Can you think of a good daily life example of livelock?

Computer system examples:

Operations continually abort and retry

Java Concurrency API



```
(new Thread(command)).start();
```

```
package java.util.concurrent;
  public interface Executor {
  void execute(Runnable command);
}
```

```
package java.util.concurrent;

public interface Future<V> {
    boolean cancel(boolean mayInterruptIfRunning);
    boolean isCancelled();
    boolean isDone();
    V get() throws InterruptedException, ExecutionException;
    V get(long timeout, TimeUnit unit)
        throws InterruptedException, ExecutionException, TimeoutException;
}
```

```
package java.util.concurrent;

 public interface Callable<V> {
    V call() throws Exception;
 }
```

Executor, Callable, Future Interfaces

Base do Framework

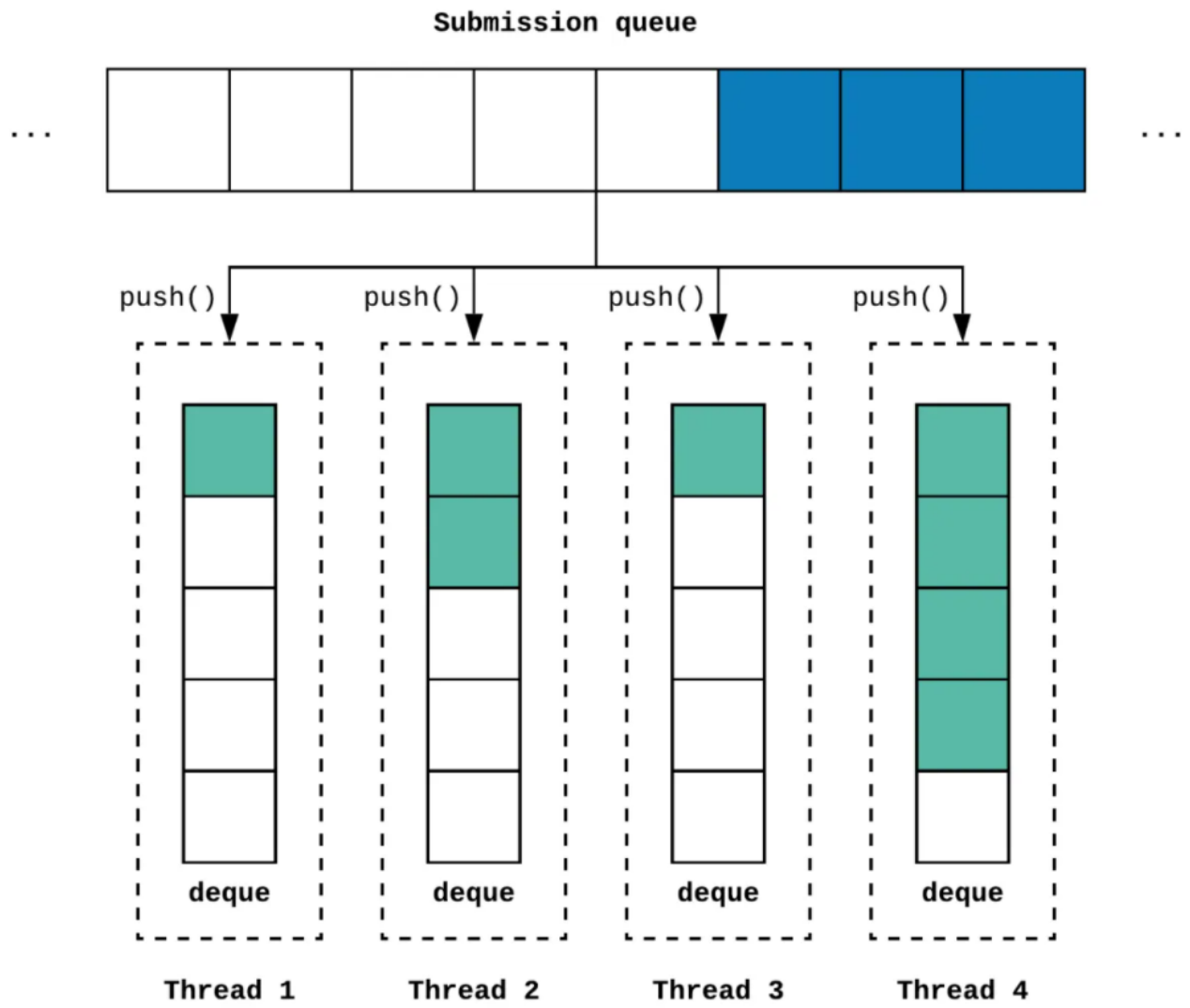
```java
public interface ExecutorService extends Executor {
    void shutdown();
    List<Runnable> shutdownNow();
    boolean isShutdown();
    boolean isTerminated();
    boolean awaitTermination(long timeout, TimeUnit unit)
        throws InterruptedException;
    <T> Future<T> submit(Callable<T> task);
    <T> Future<T> submit(Runnable task, T result);
    Future<?> submit(Runnable task);
    <T> List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks)
        throws InterruptedException;
    <T> List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks,
                                  long timeout, TimeUnit unit)
        throws InterruptedException;
    <T> T invokeAny(Collection<? extends Callable<T>> tasks)
        throws InterruptedException, ExecutionException;
    <T> T invokeAny(Collection<? extends Callable<T>> tasks,
                    long timeout, TimeUnit unit)
        throws InterruptedException, ExecutionException, TimeoutException;
}
```

ExecutorService Interface

- Adiciona recurso para o gerenciamento das tarefas (tasks) e do proprio executor

Executors Class | Thread Pool

Metodo	Descricao
newCachedThreadPool(..)	Cria threads quando necessario. Ideal para tarefas assincronas nao demoradas. Se todas as threads estao ocupadas, e uma tarefa for submetida, entao outra thread sera criada. Se uma thread ficar inativa por 60 seg, entao ela sera destruida.
newFixedThreadPool(..)	Pool de thread com quantidade especifica de threads. Se mais tarefas forem submetidas do que o limite, elas ficarao aguardando threads disponiveis. Se uma thread falhar, outra sera criada e incluida no pool.
newScheduledThreadPool(..)	Cria um pool de threads que pode agendar comandos para serem executados após um determinado atraso ou para serem executados periodicamente.
newSingleThreadExecutor(..)	Pool com apenas uma thread
newSingleThreadScheduledExecutor(..)	Cria um executor de thread único que pode agendar comandos para serem executados após um determinado atraso ou para serem executados periodicamente.
newWorkStealingPool(..)	Cria um pool de threads work stealing usando todos os processadores disponíveis como seu nível de paralelismo de destino. Pode usar várias filas para reduzir a contenção.



Algoritmo Work Stealing

ForkJoin Framework

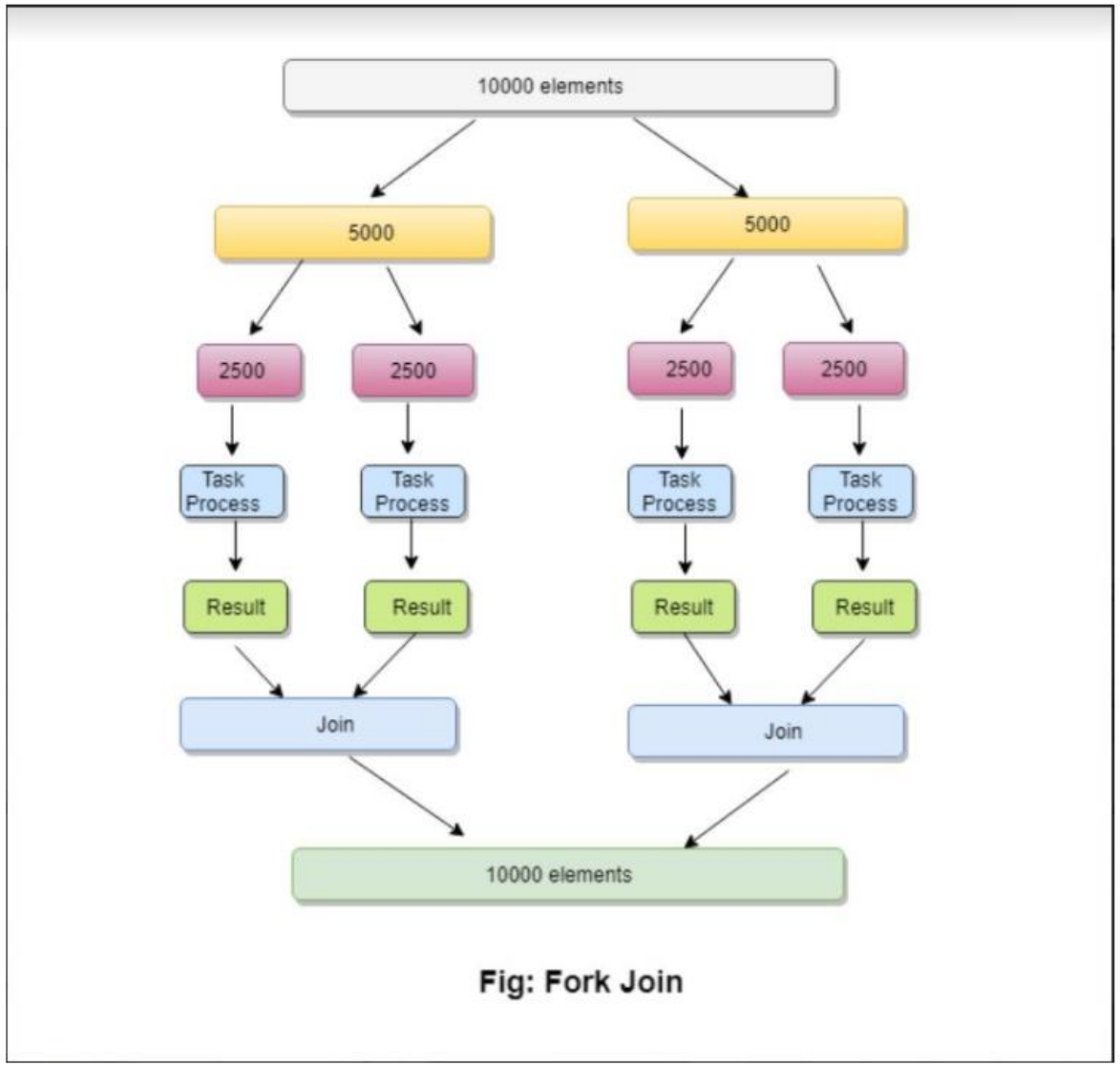


Fig: Fork Join