Multithreading v jazyku Java

LUKÁŠ VAVREK
ERIK VOJTKO

Obsah

Základné Informácie

Thread / Runnable

Fork/Join

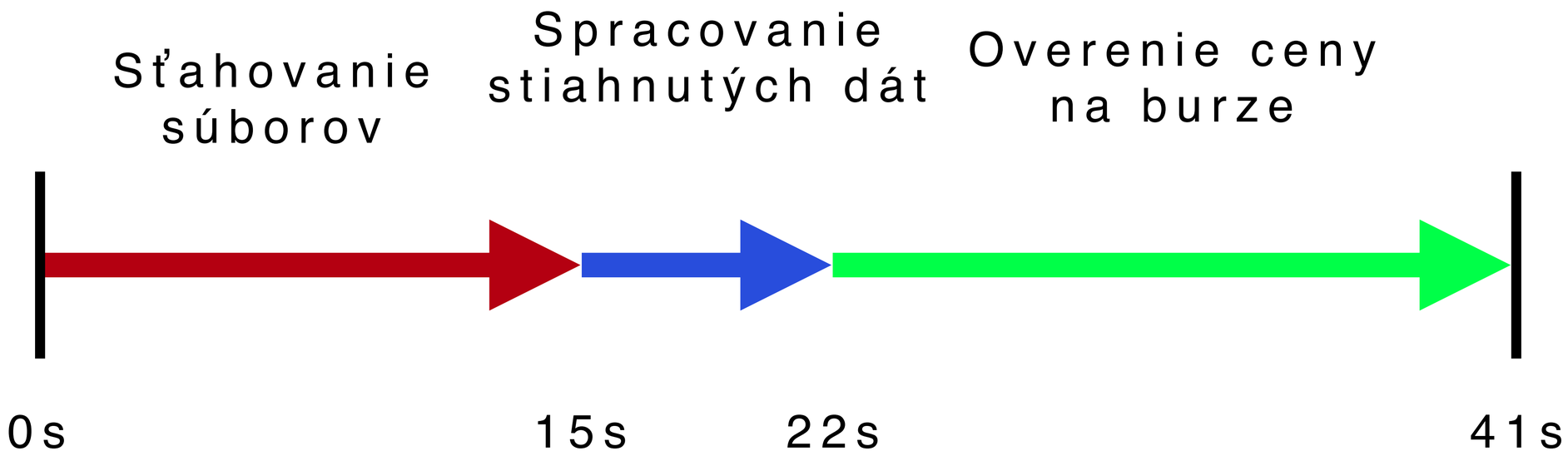
Synchronizácia

Základné Informácie

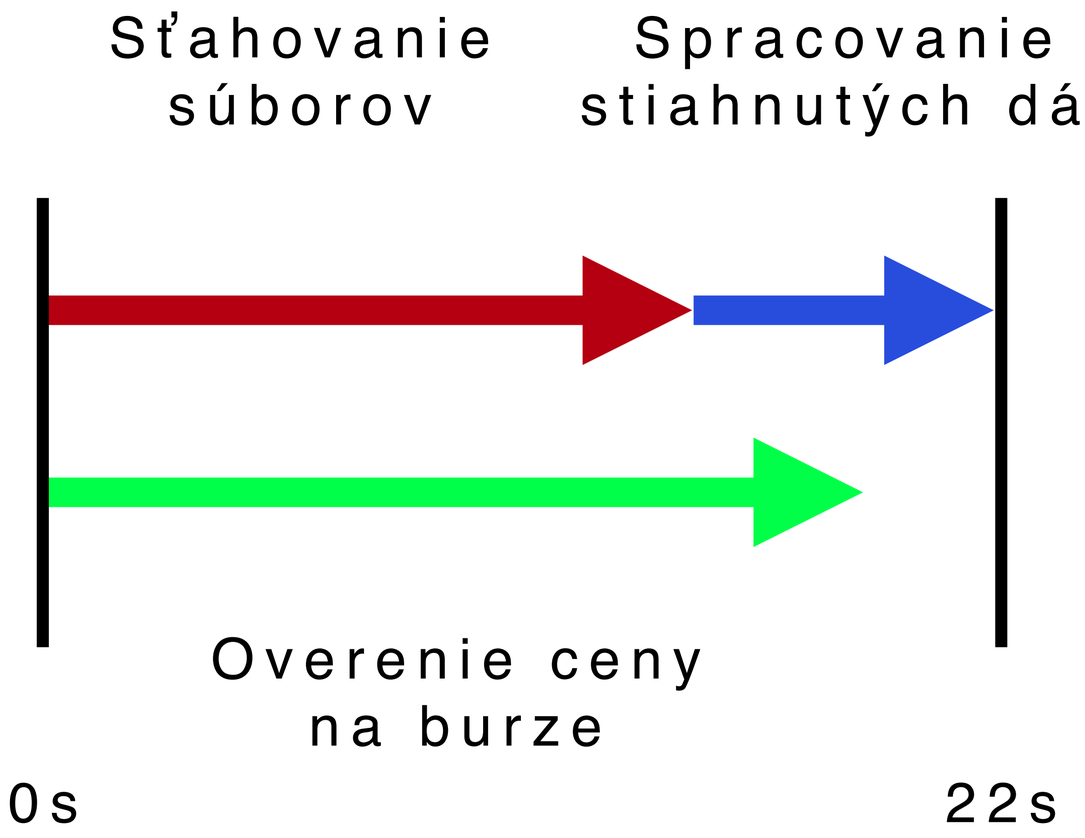
Multithreading + Java

- Vstavaná podpora v jazyku
- Paralelné vykonávanie časti kódu
- Implementácia pomocou odľahčených procesov

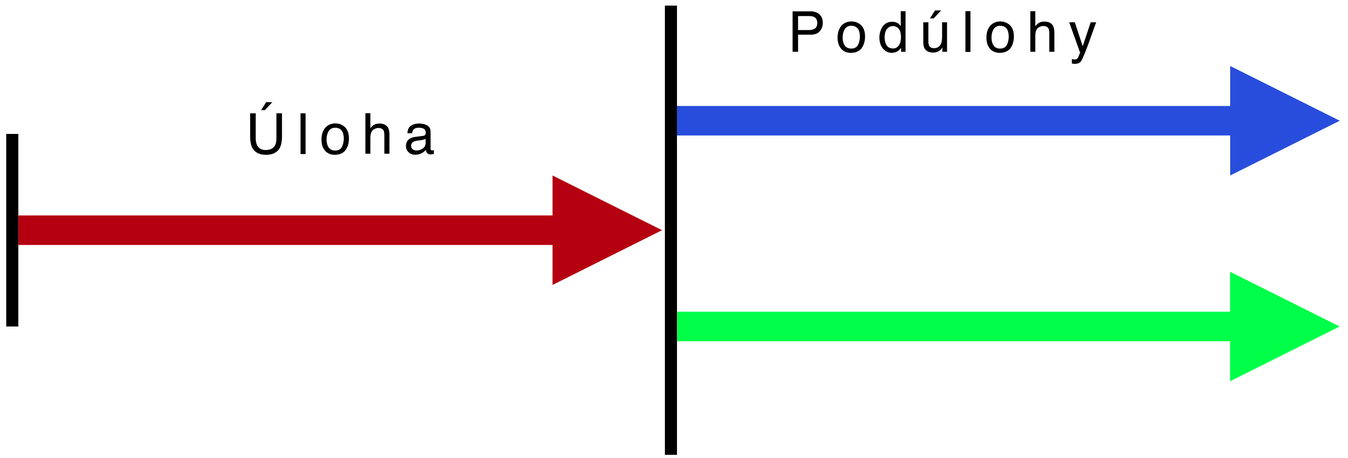
Sériové vykonávanie programu



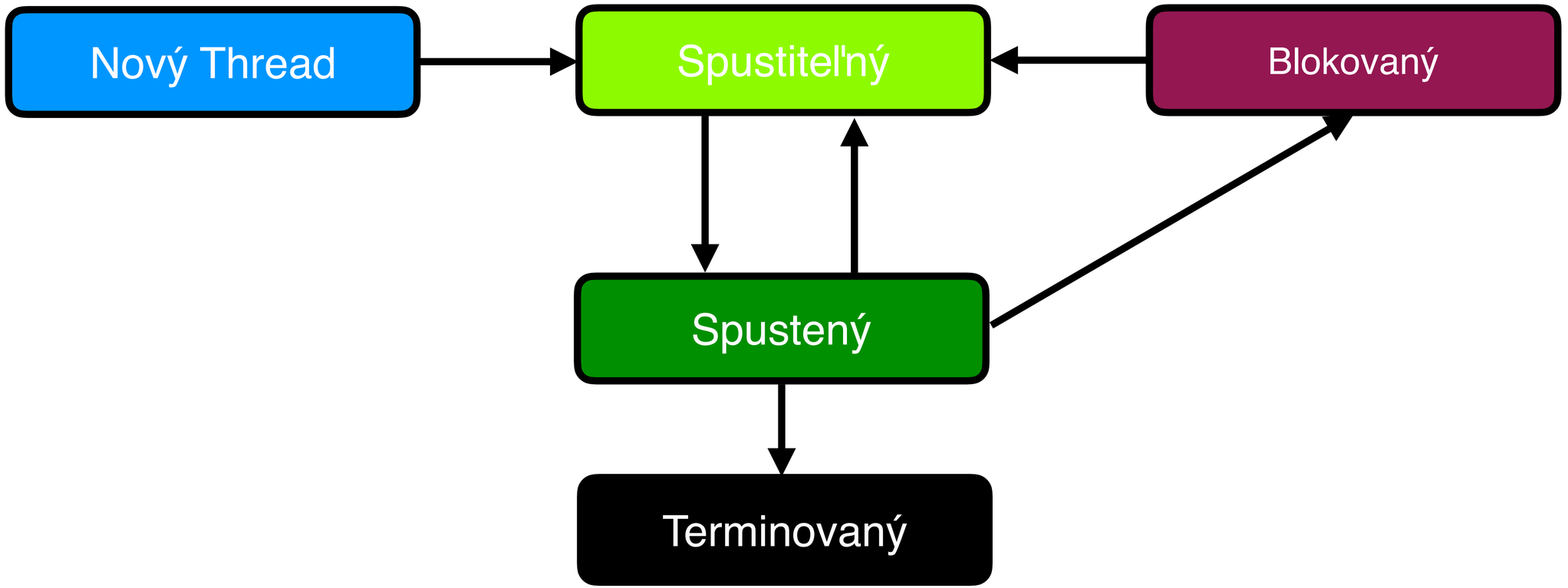
Súbežnosť


Sťahovanie súborov
Spracovanie stiahnutých dát
Overenie ceny na burze
0s
22s

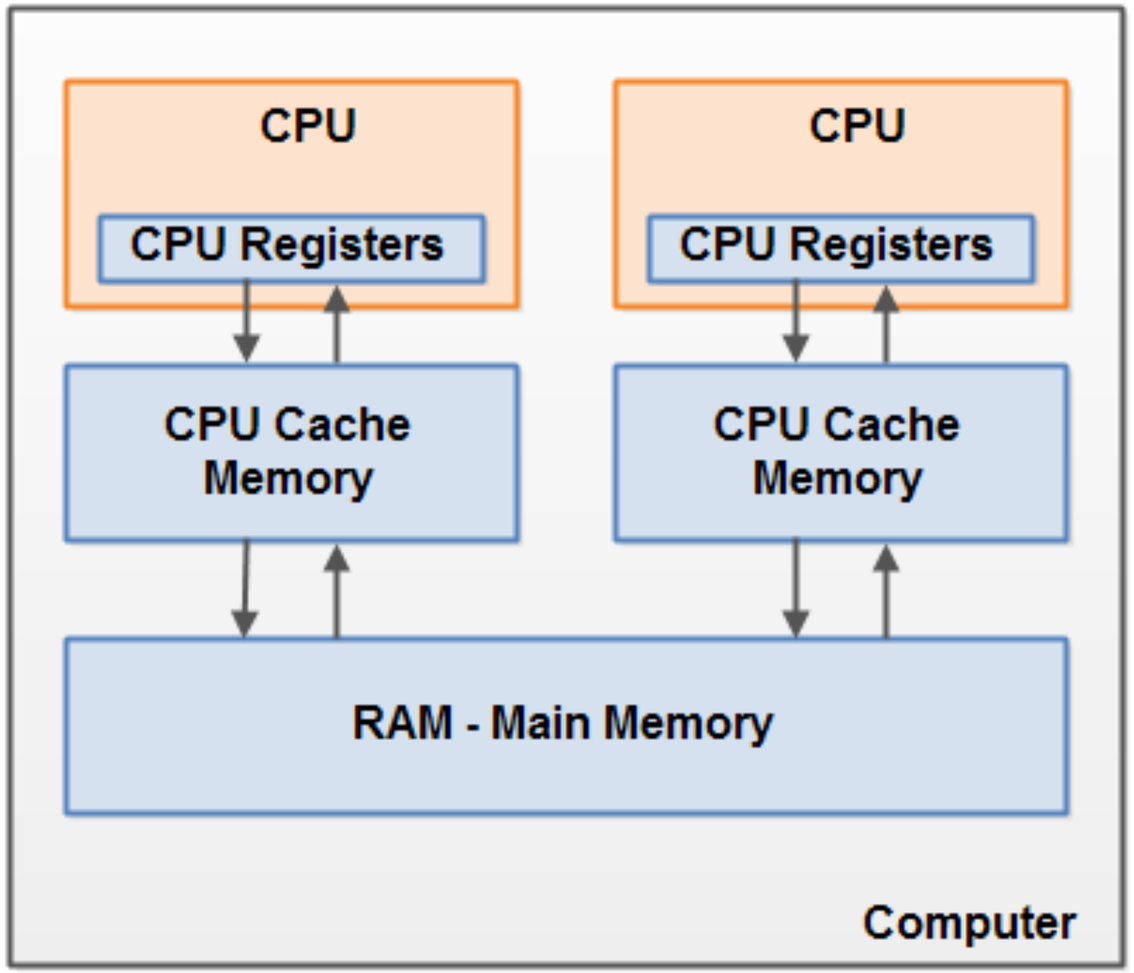
Paralelizácia



Životný cyklus threadov



Architektúra CPU



Trieda **Thread**
Rozhranie **Runnable**

Trieda Thread

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

Použitie

```
MyClass t1 = new MyClass ();
t1.start();
```

Rozhranie Runnable

```
public class MyClass implements Runnable {
    public void run(){
        System.out.println("MyClass running”);
    }
}
```

Anonymná Implementácia Runnable

```
Runnable myRunnable = new Runnable() {
    public void run(){
        System.out.println("MyClass running”);
    }
}
```

Lambda Implementácia Runnable

```
Runnable myRunnable = () -> {
    System.out.println("MyClass running”);
};
```

Použitie

```
Thread t1 = new Thread(new MyClass ());
t1.start();
```

Diagnostika bežiaceho threadu

```
Thread.currentThread().getName();
```


Fork/Join

Fork/Join

- Java 7
- Využiť čo najväčší počet jadier
- Rozdeľuj a panuj
- Pool threadov - “ForkJoinPool”

ForkJoinPool

```
// java8
ForkJoinPool commonPool = ForkJoinPool.commonPool();

// java7
public class PoolUtil {
    static ForkJoinPool pool = new ForkJoinPool(2);
}
```

RecursiveAction

```
public class MyAction extends RecursiveAction {
    private String workload = “”;
    private static final int TRESHOLD = 4;
    …
    protected void compute() {
        if (workload.length() > TRESHOLD) {
            ForkJoinTask.invokeAll(createSubtasks());
        } else {
            processing(workload);
        }
    }
    …
    private List<MyAction> createSubtasks() {
        …
        subtasks.add(new MyAction(…));
        …
    }
}
```

Execute

```
pool.execute(myTask);
int result = myTask.join();


int result = pool.invoke(myTask);
```

Synchronizácia

Synchronized

```
public class TwoSums {
    private int sum1 = 0;
    private int sum1 = 0;

    public synchronized void add(
        int val1, int val2) {

        this.sum1 += val1;
        this.sum2 += val2;
    }
}
```

Synchronized

```
public class TwoSums {
    private int sum1 = 0;
    private int sum1 = 0;

    public void add(int val1, int val2){
        synchronized(this) {
            this.sum1 += val1;
            this.sum2 += val2;
        }
    }
}
```

Synchronized

```
public class Logger {
    public static void log(
        String msg1, String msg2) {

        synchronized(TwoSums.class) {
            log.writeln(msg1);
            log.writeln(msg2);
        }
    }
}
```

Vlastná implementácia triedy Lock

```
public class Lock {
    private boolean isLocked = false;

    public synchronized void lock()
        throws InterruptedException {

        while (isLocked) wait();
        isLocked = true;
    }

    public synchronized void unlock() {
        isLocked = false;
        notify();
    }
}
```

Volatile

```
public class SharedObject {
    private volatile int counter = 0;
}
```

ThreadLocal

```
ThreadLocal myThreadLocal = new ThreadLocal<String>();

myThreadLocal.set("value");
String value = myThreadLocal.get();
```

Q & A
+
Live Coding Demo