



# Multithreading v jazyku Java

LUKÁŠ VAVREK  
ERIK VOJTKO

# Obsah

Základné Informácie

Thread / Runnable

Fork/Join

Synchronizácia

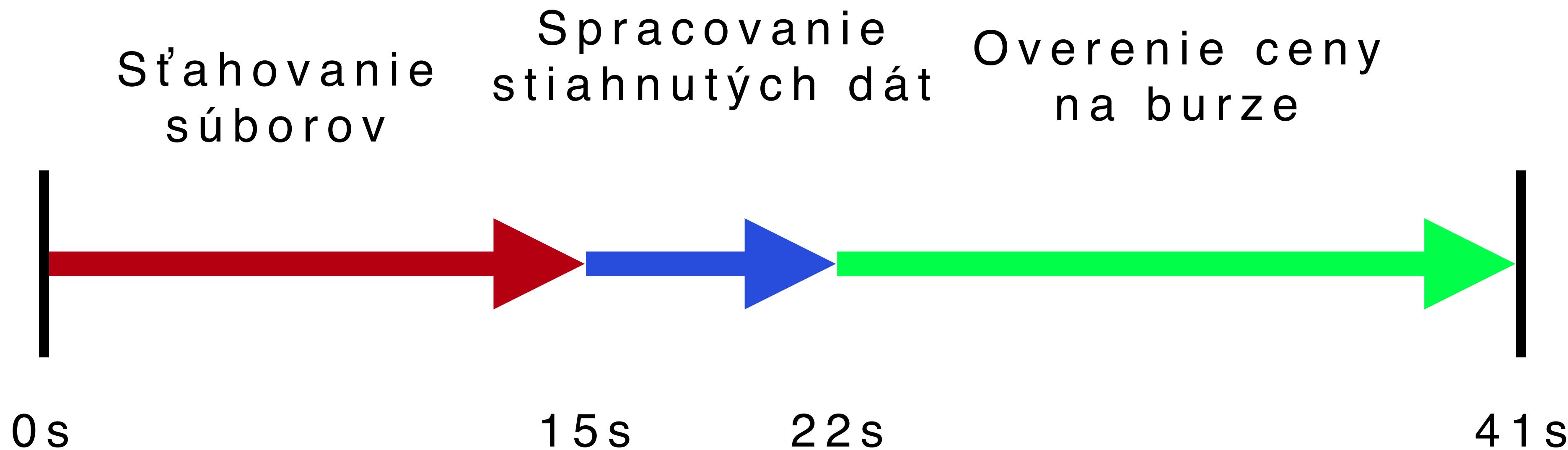
# Základné Informácie



# Multithreading + Java

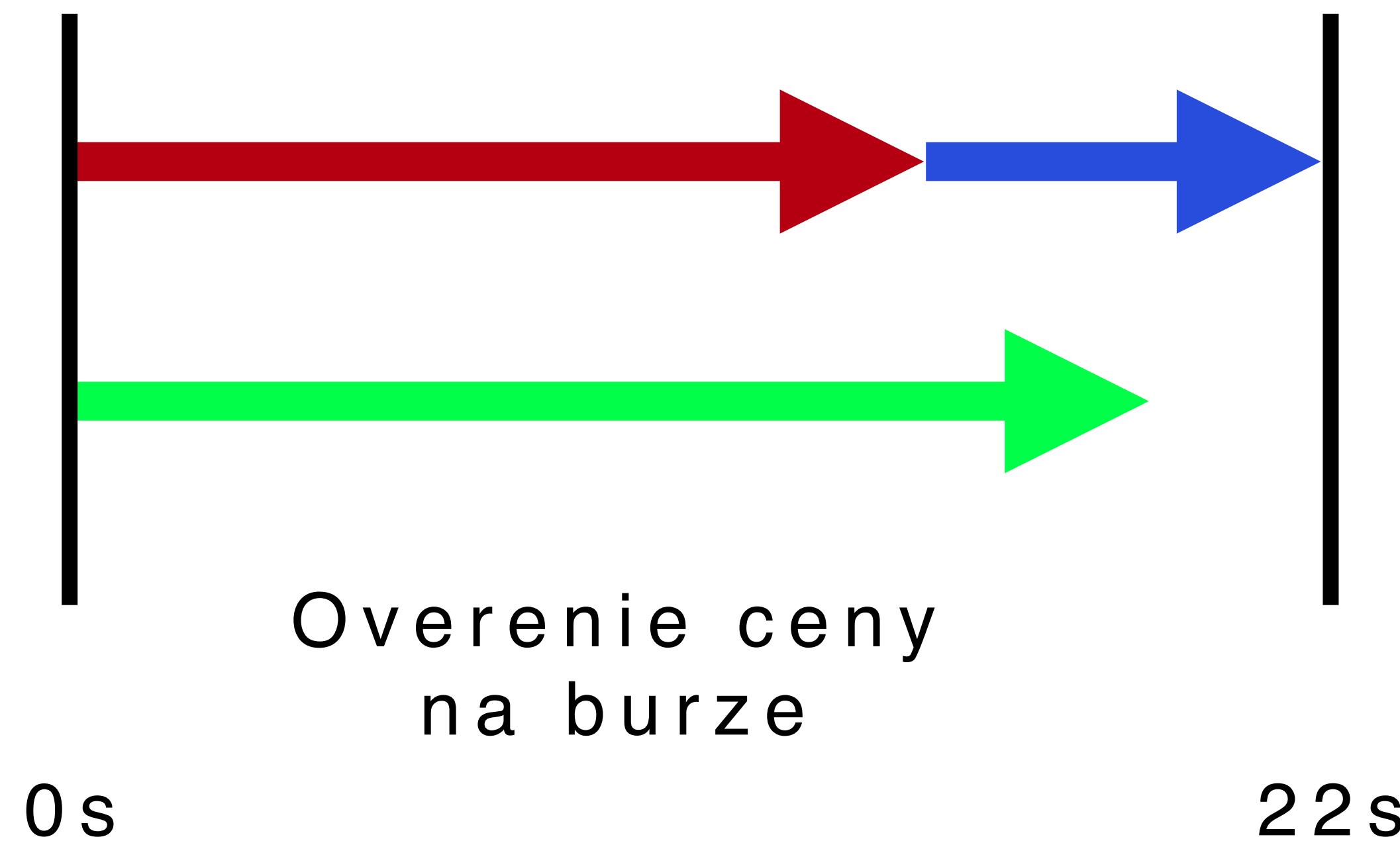
- ⟩ Vstavaná podpora v jazyku
- ⟩ Paralelné vykonávanie časti kódu
- ⟩ Implementácia vrámci JVM pomocou odľahčených procesov

# Sériové vykonávanie programu

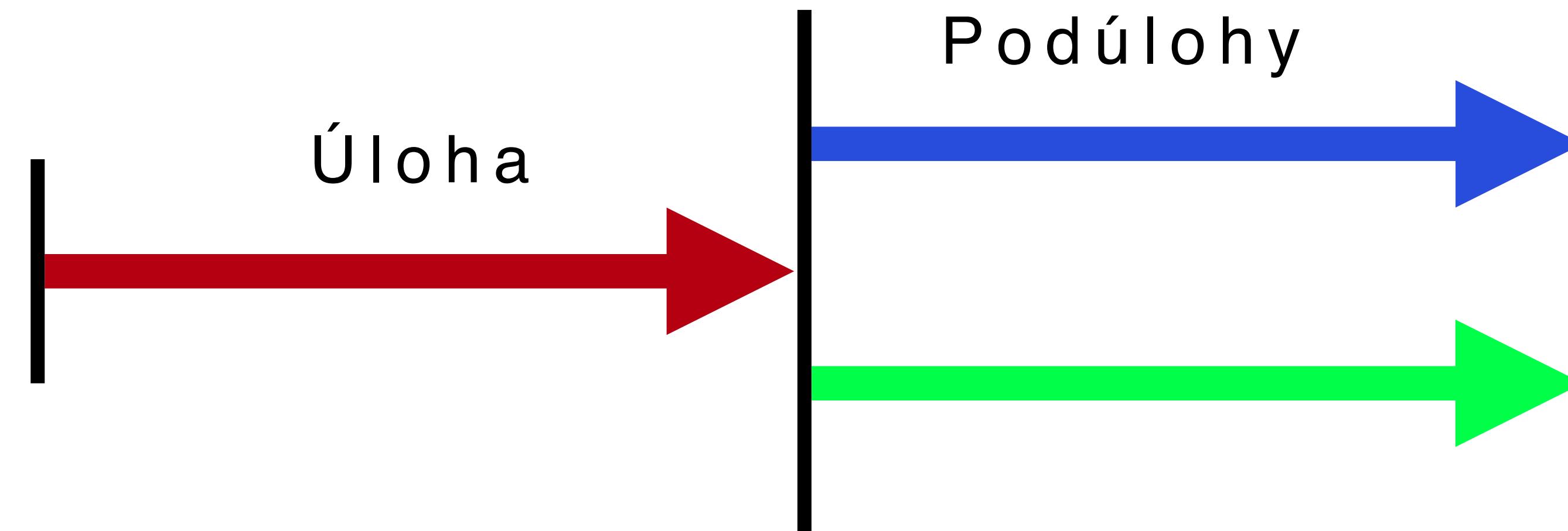


# Súbežnosť

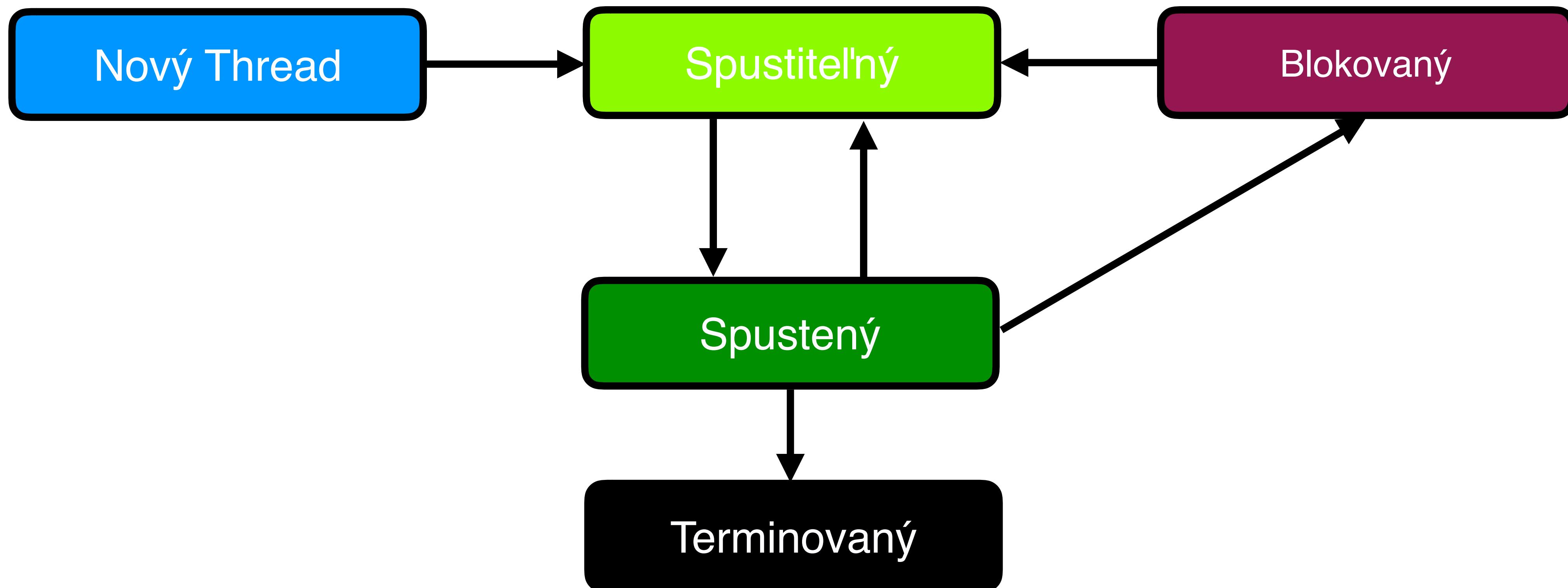
# Stiahovanie súborov Spracovanie stiahnutých dát



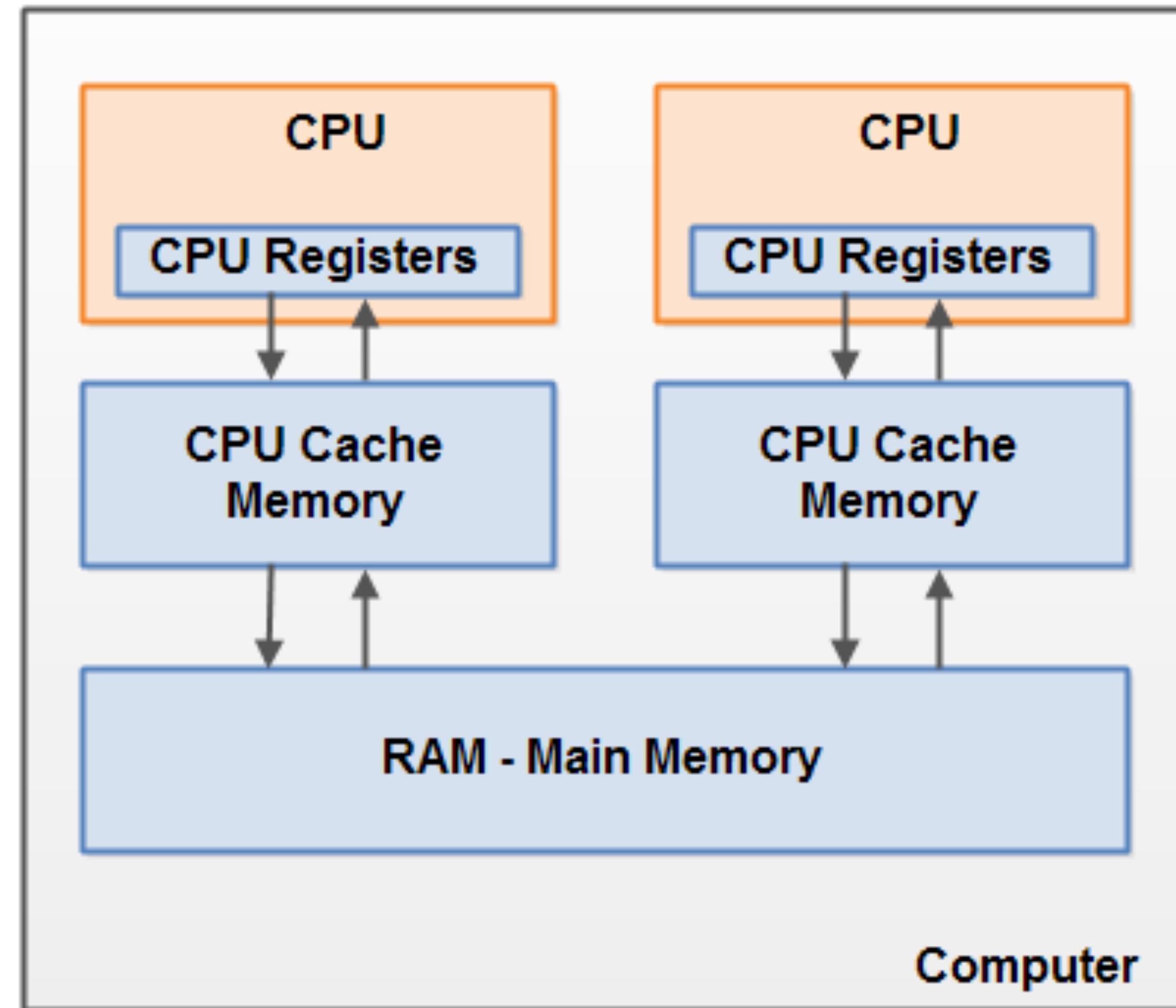
# Paralelizácia



# Životný cyklus threadov



# Architektúra CPU



# Trieda Thread Rozhranie Runnable

```
public void startPollingForUpdates(@NotNull String uid) {
    Timber.d("startPollingForUpdates(), with this uid: %s", uid);
    long interval = 1000 * 60; // check every minute
    long start = System.currentTimeMillis() + interval;
    Intent i = LiveEventsService.getCheckLiveEventIntent(this, uid);
    pi = PendingIntent.getService(this, 0, i, 0);
    AlarmManager alarmManager = (AlarmManager) getSystemService(Context.ALARM_SERVICE);
    alarmManager.setRepeating(AlarmManager.RTC_WAKEUP, start, interval, pi);
}

public void stopPollingForUpdates() {
    Timber.d("stopPollingForUpdates()");
    if (pi != null) {
        AlarmManager alarmManager = (AlarmManager) getSystemService(Context.ALARM_SERVICE);
        alarmManager.cancel(pi);
    }
}
```

# 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 bežiacého threadu

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

# Fork/Join



The image shows a person working on a laptop in an office setting. The laptop screen displays the Android Studio interface, specifically the code editor for a Java file named `Video.java`. The code implements a `ForkJoinPool` for video processing. The code includes methods for starting and stopping polling for updates, and for handling pending intents. The Android Studio interface also shows the project structure, build variants, and various toolbars.

```
public void startPollingForUpdates(@NotNull String uid) {
    Timber.d("startPollingForUpdates(), with this uid: %s", uid);
    long interval = 1000 * 60; // check every minute
    long start = System.currentTimeMillis() + interval;
    Intent i = LiveEventsService.getCheckLiveEventIntent(this, uid);
    pi = PendingIntent.getService(this, 0, i, 0);
    AlarmManager alarmManager = (AlarmManager) getSystemService(Context.ALARM_SERVICE);
    alarmManager.setRepeating(AlarmManager.RTC_WAKEUP, start, interval, pi);
}

public void stopPollingForUpdates() {
    Timber.d("stopPollingForUpdates()");
    if (pi != null) {
        AlarmManager alarmManager = (AlarmManager) getSystemService(Context.ALARM_SERVICE);
```

# Fork/Join

- ⟩ Java 7
- ⟩ Využiť čo najväčší počet jadier
- ⟩ Rozdel'uj 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 THRESHOLD = 4;  
    ...  
    protected void compute() {  
        if (workload.length() > THRESHOLD) {  
            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



The image shows a person's hands working on a laptop keyboard. A smartphone is connected to the laptop via a USB cable. In the background, a monitor displays the Android Studio interface, showing Java code for a project named "PostTV-Android". The code is related to a service named "LiveEventsService" and its methods "startPollingForUpdates" and "stopPollingForUpdates". The monitor also shows the Android Monitor tab with log entries. The overall scene suggests a developer environment.

```
public void startPollingForUpdates(@NotNull String uid) {
    Timber.d("startPollingForUpdates(), with this uid: %s", uid);
    long interval = 1000 * 60; // check every minute
    long start = System.currentTimeMillis() + interval;
    Intent i = LiveEventsService.getCheckLiveEventIntent(this, uid);
    pi = PendingIntent.getService(this, 0, i, 0);
    alarmManager = (AlarmManager) getSystemService(Context.ALARM_SERVICE);
    alarmManager.setRepeating(AlarmManager.RTC_WAKEUP, start, interval, pi);
}

public void stopPollingForUpdates() {
    Timber.d("stopPollingForUpdates()");
    if (pi != null) {
        AlarmManager alarmManager = (AlarmManager) getSystemService(Context.ALARM_SERVICE);
```

# 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 sum2 = 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(Logger.class) {  
            log.writeln(msg1);  
            log.writeln(msg2);  
        }  
    }  
}
```

# Vlastná implementácia tryedy Lock

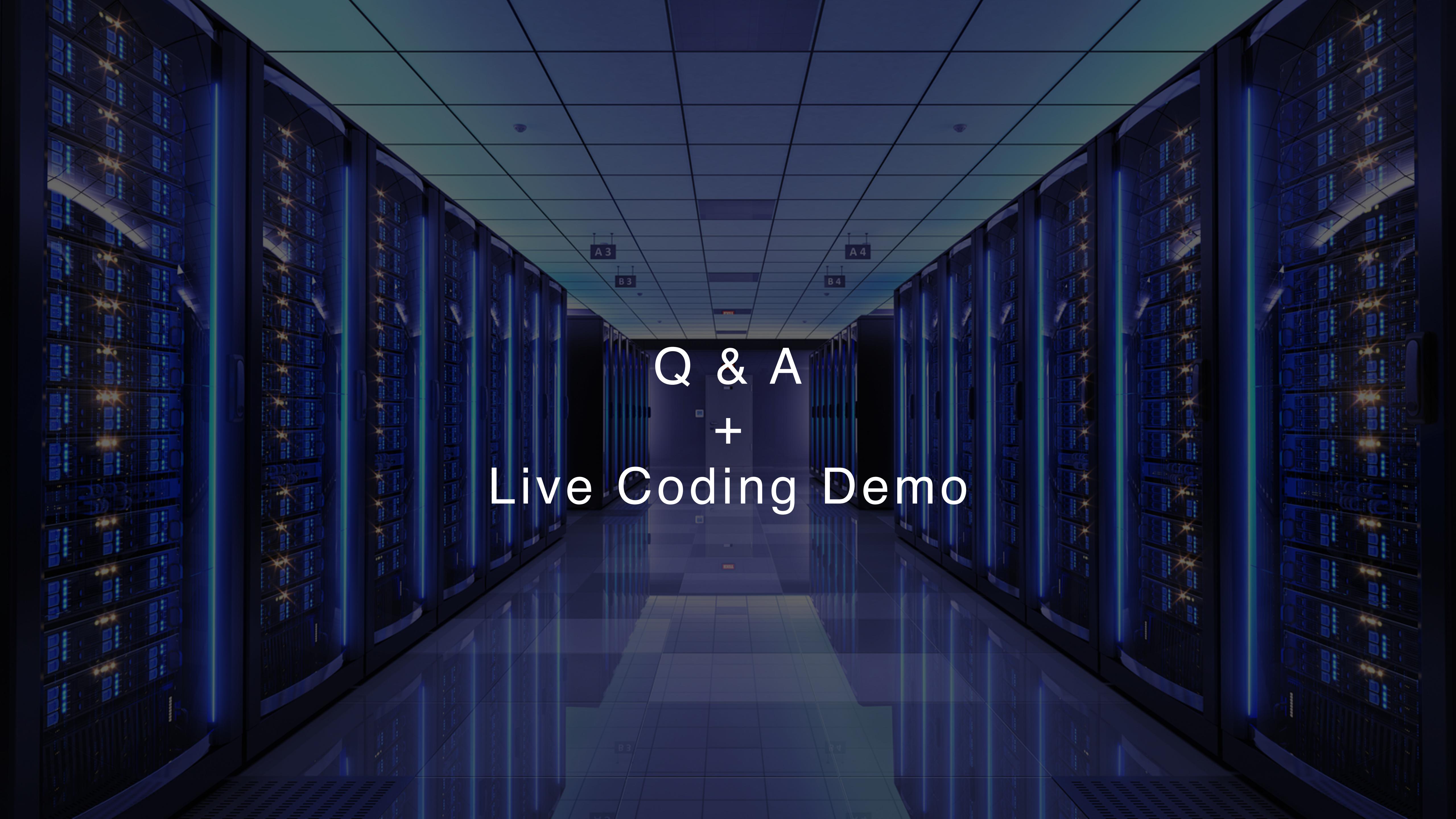
```
public class Lock {  
    private boolean isLocked = false;  
  
    public synchronized void lock()  
        throws InterruptedException {  
  
        while (isLocked) wait();  
        isLocked = true;  
    }  
  
    public synchronized void unlock() {  
        isLocked = false;  
        notify();  
    }  
}
```

# Volatile

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

# ThreadLocal

```
ThreadLocal myThreadLocal = new ThreadLocal<String>();  
  
myThreadLocal.set("value");  
String value = myThreadLocal.get();
```

The background of the slide is a photograph of a server room. The room is dimly lit, with the primary light source being the numerous blue and white LED lights on the server racks. The racks are arranged in two main rows that curve slightly away from the viewer. Above the racks, the ceiling has a grid of recessed lighting. Several labels are visible above the racks, including "A3", "B3", "A4", and "B4". The overall atmosphere is one of a high-tech, industrial data center.

Q & A  
+  
Live Coding Demo