S01: Java is the new Black

E03: Multithreading in Java

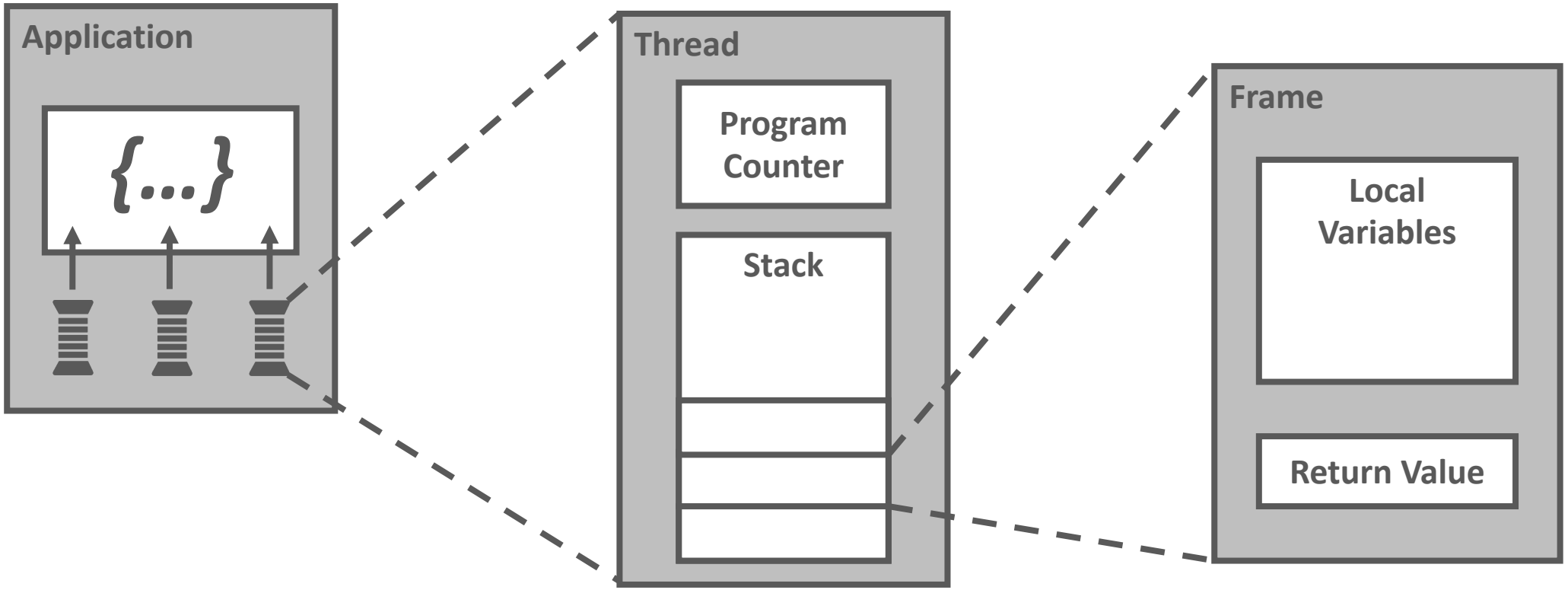
Tomek Adamczewski

09/04/2019

Thread

JAVA FAKTURA

A single path of execution within a program (process)



Demo01

HelloWorld of multithreaded applications

java.lang.Thread

JAVA FAKTURA

Creating Own Threads

- new Thread(Runnable)
- ~~Extend Thread class and override run()~~

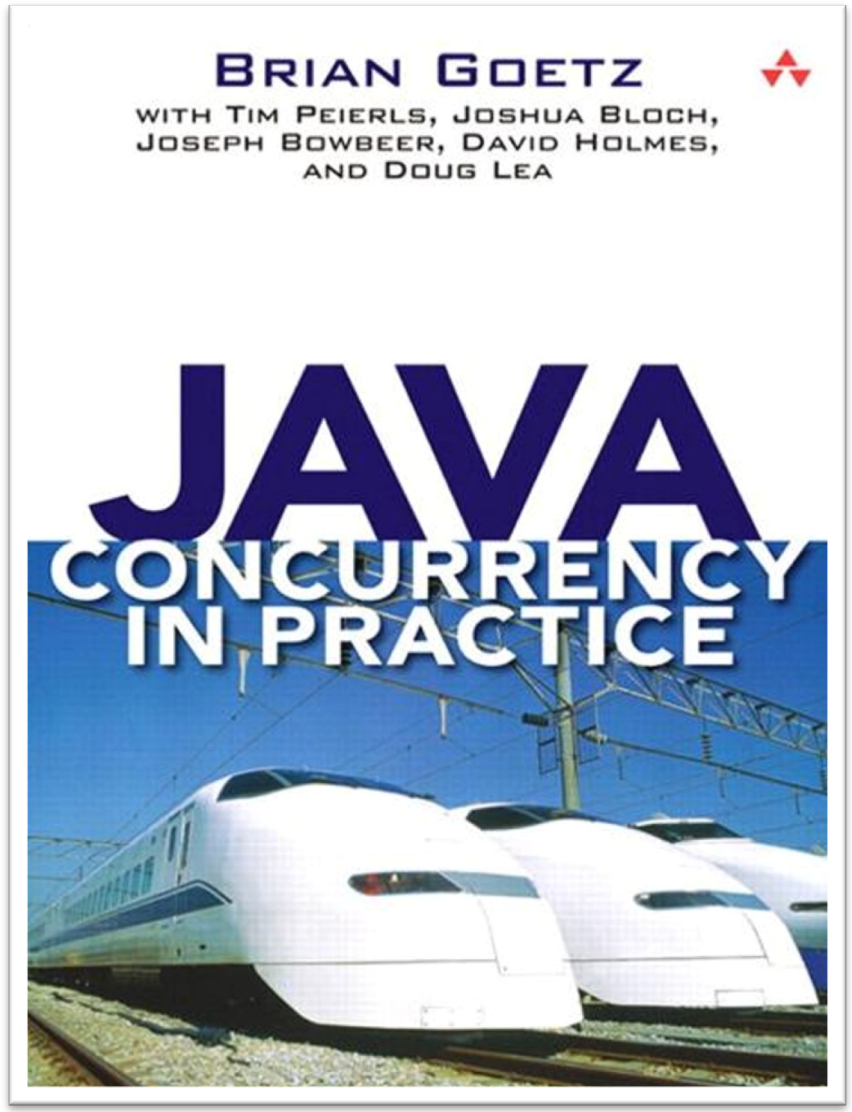
Useful methods

- start
- setName
- setDaemon
- setPriority
- setUncaughtExceptionHandler
- join
- static currentThread
- ~~static sleep~~

JAVA FAKTURA

Demo02

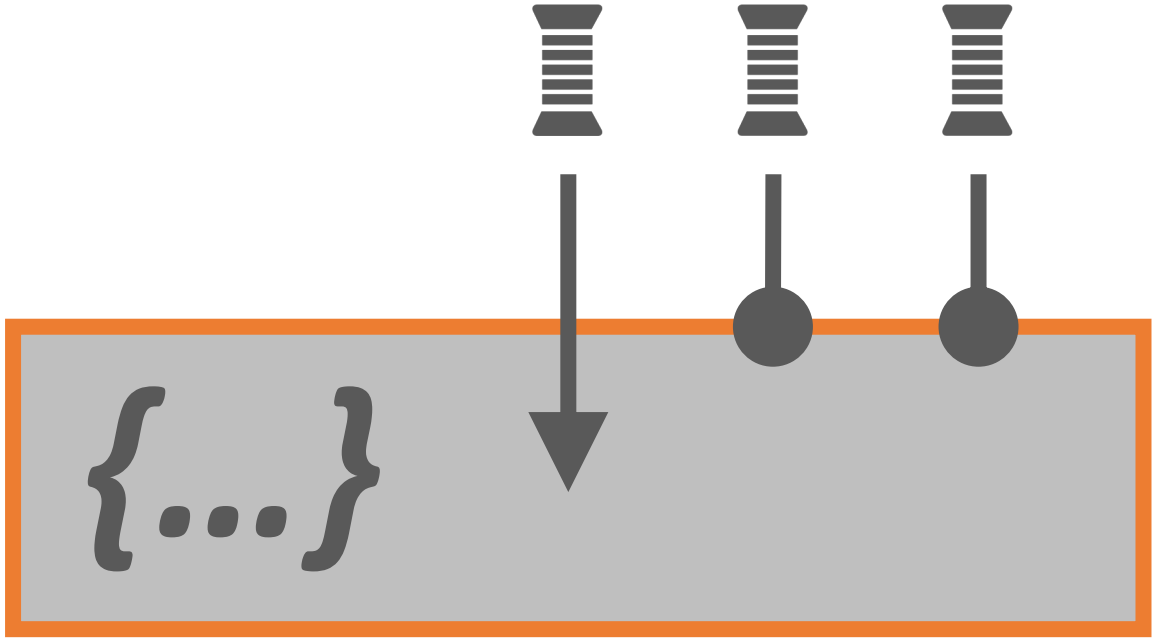
Race conditions



JAVA FAKTURA

synchronized

JAVA FAKTURA



Double-checked locking

JAVA FAKTÚRA

```
 1   private volatile ImmutableType value;
 2   private final Object lock = new Object();
 3
 4   public ImmutableType getValue() {
 5       if (value == null) {                          // optimistic
 6           synchronized (lock) {
 7               if (value == null) {                  // 2nd check
 8                   value = initializeValue();
 9               }
10           }
11       }
12       return value;
13   }
```

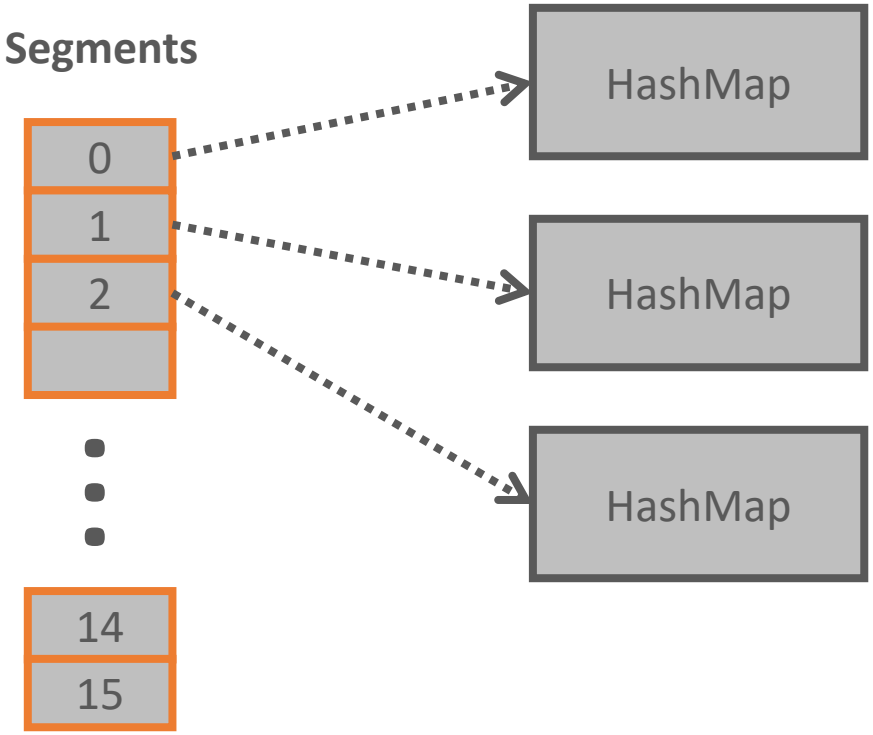
Compare-and-swap (CAS)

JAVA FAKTURA

```
UPDATE table_name
SET the_value='new value’, version=2
WHERE version=1;
```

ConcurrentHashMap

JAVA FAKTURA



Other mechanisms

JAVA FAKTURA

- Semaphore
- Mutex
- Barrier
- Latch

JAVA FAKTURA

Demo02

Thread-safe implementations

Why are threads “expensive”?

- Creation and teardown overhead
- Context switching
- Memory allocation

ExecutorService

JAVA FAKTURA

- Decouples threads from tasks
- Encapsulates task execution

```
<T> Future<T> submit(Callable<T> task);
```

java.util.concurrent.Executors

JAVA FAKTÚRA

- newSingleThreadExecutor
- newFixedThreadPool
- newSingleThreadScheduledExecutor
- newScheduledThreadPool
- …plus a bunch of callable() methods

JAVA FAKTURA

Demo03

Thread pooling

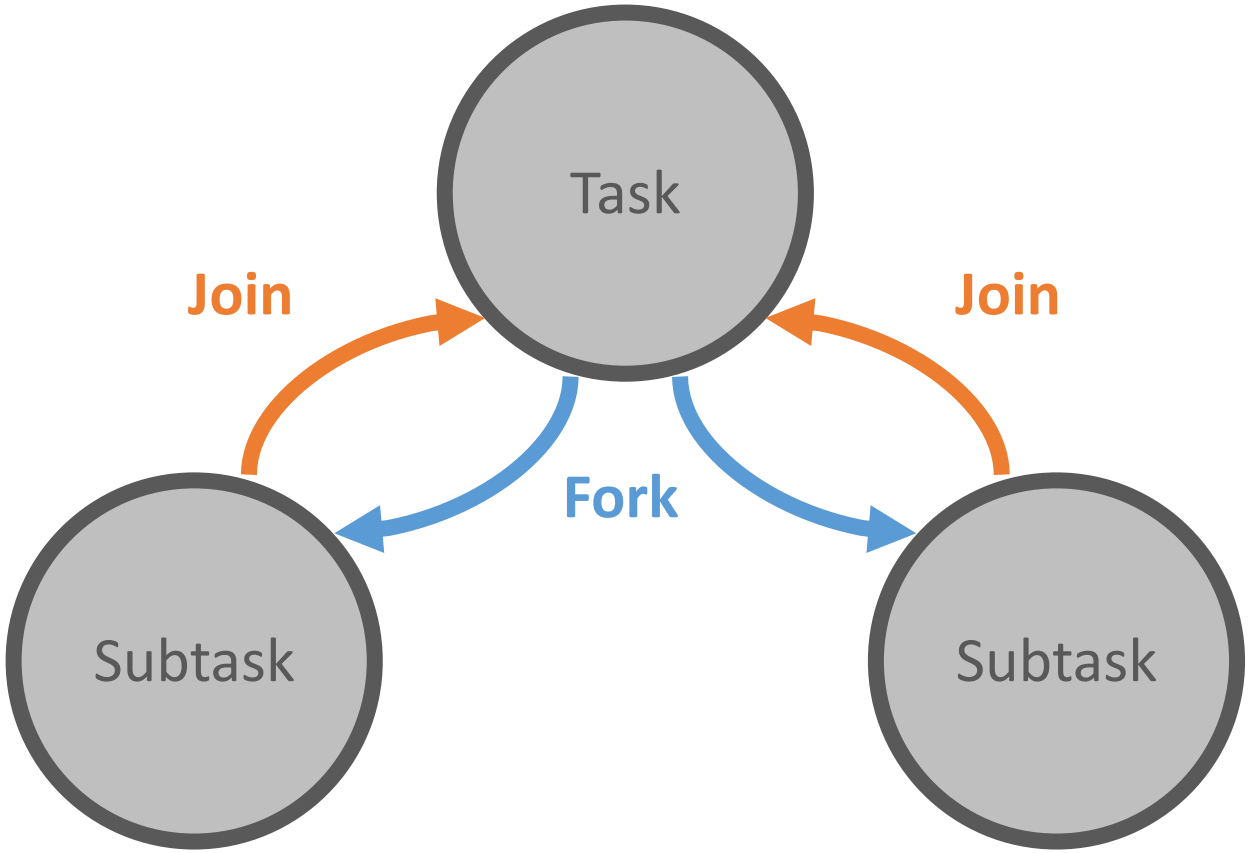
ForkJoinPool

JAVA FAKTURA

- A “special” ExecutorService that splits tasks into smaller chunks, submits each of them back to ForkJoinPool and merges the results when ready
- Based on two important algorithms: **fork-join model** and **work stealing**
- ForkJoinPool#commonPool() returns a common pool, statically initialized when starting JVM process
- Heavily utilized by Java streams

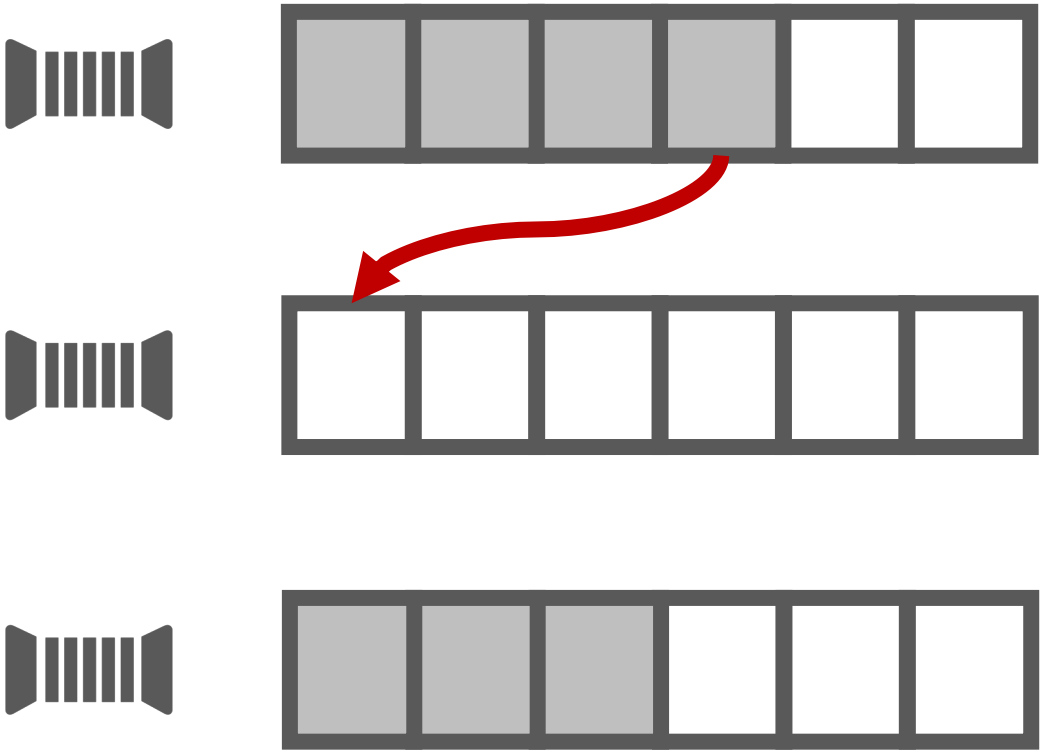
Fork-join model

JAVA FAKTURA



Work stealing

JAVA FAKTÚRA



JAVA FAKTURA

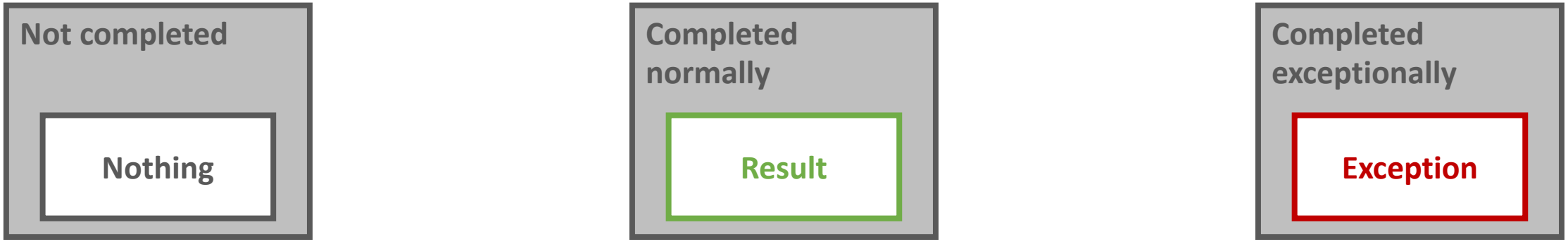
Demo04

parallelStreams

CompletableFuture

JAVA FAKTURA

„A Future that may be explicitly completed, and may be used as a CompletionStage, supporting dependent functions and actions that trigger upon its completion”



CompletableFuture contract

JAVA FAKTURA

Total number of public methods: 70+ 🤢 (and still growing)

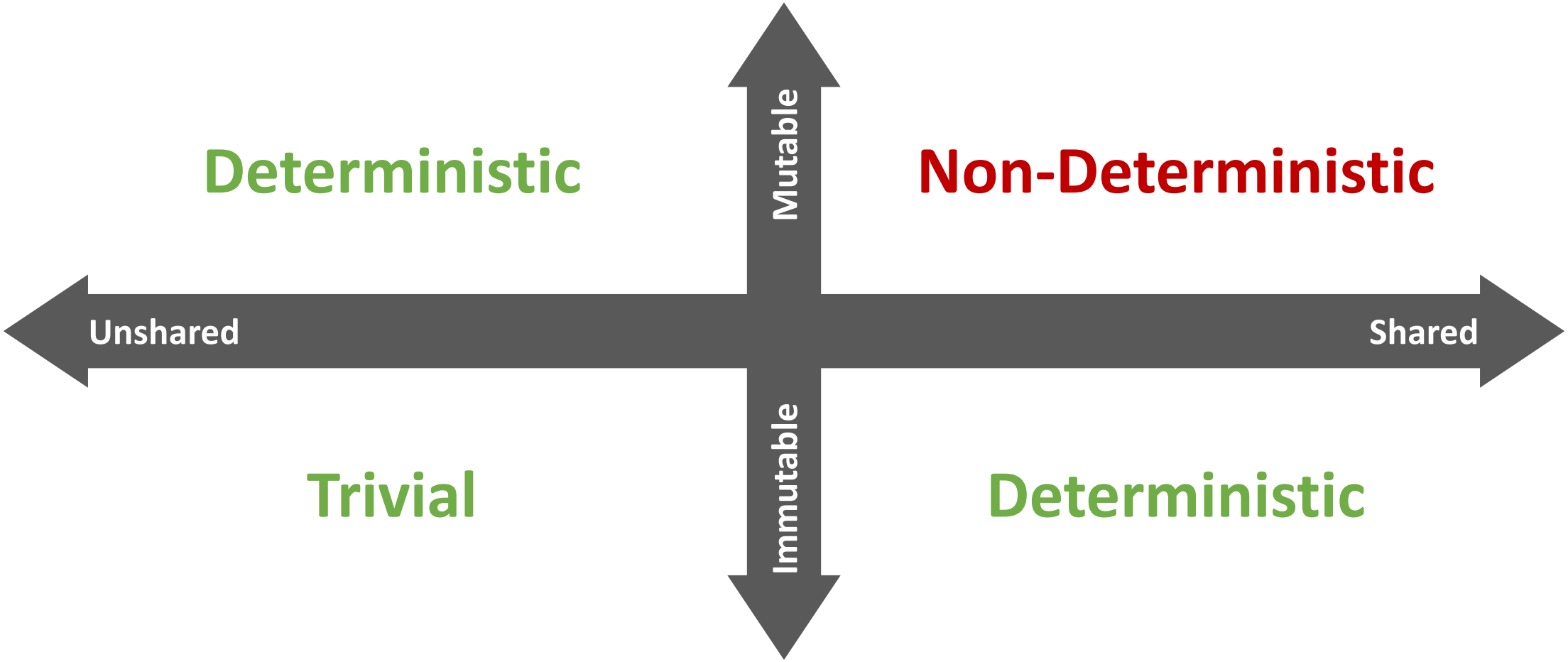
- new CompletableFuture()
- supplyAsync/runAsync
- completedFuture/ failedFuture
- complete[Exceptionally]
- ~~get~~
- ~~join~~
- getNow
- thenApply
- thenCompose/thenCombine
- thenRun/thenAccept
- exceptionally/handle
- whenComplete
- obtrudeValue/obtrudeException
- anyOf/allOf, acceptEither, acceptBoth…

Demo05

CompletableFutures

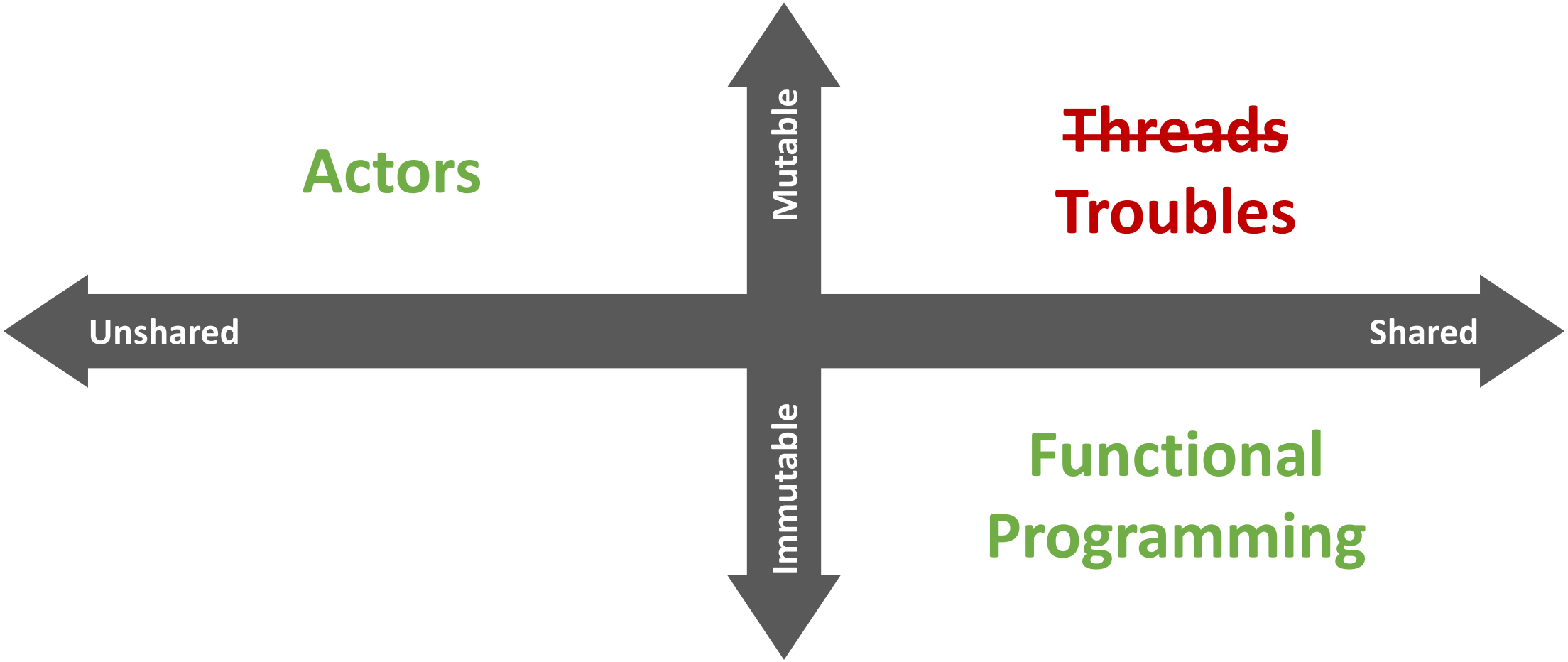
Concurrency Models

JAVA FAKTURA



Concurrency Models

JAVA FAKTURA



Root of all Evil

JAVA FAKTURA

```
/**
 * should be implemented by any class
 * (...) to be executed by a thread.
 *
 * @since   1.0
 */
public interface Runnable {
    public abstract void run();
}
```

java.util.concurrent.Callable<V>

JAVA FAKTURA

```
/**
 * similar to {@link java.lang.Runnable} (...)
 * A {@code Runnable}, however, does not return
 * a result and cannot throw a checked exception.
 *
 * @since   1.5
 */
public interface Callable<V> {
    V call() throws Exception;
}
```

DO's and DON'Ts of concurrent programming

JAVA FAKTURA

DON'T

- mix blocking and non-blocking code in a single thread pool; don't use ForkJoinPool with blocking IO
- overengineer – nonblocking applications are always expensive to develop and maintain

DO

- name your threads
- monitor all queues and threadpools

Thanks!

JAVA FAKTURA