Hartmut Lang, Ericsson

July 2011

Hartmut Lang

Senior Software Developer
Solution Architect

Network Management
and Customer Solutions
for the Telecom Industry

in Java

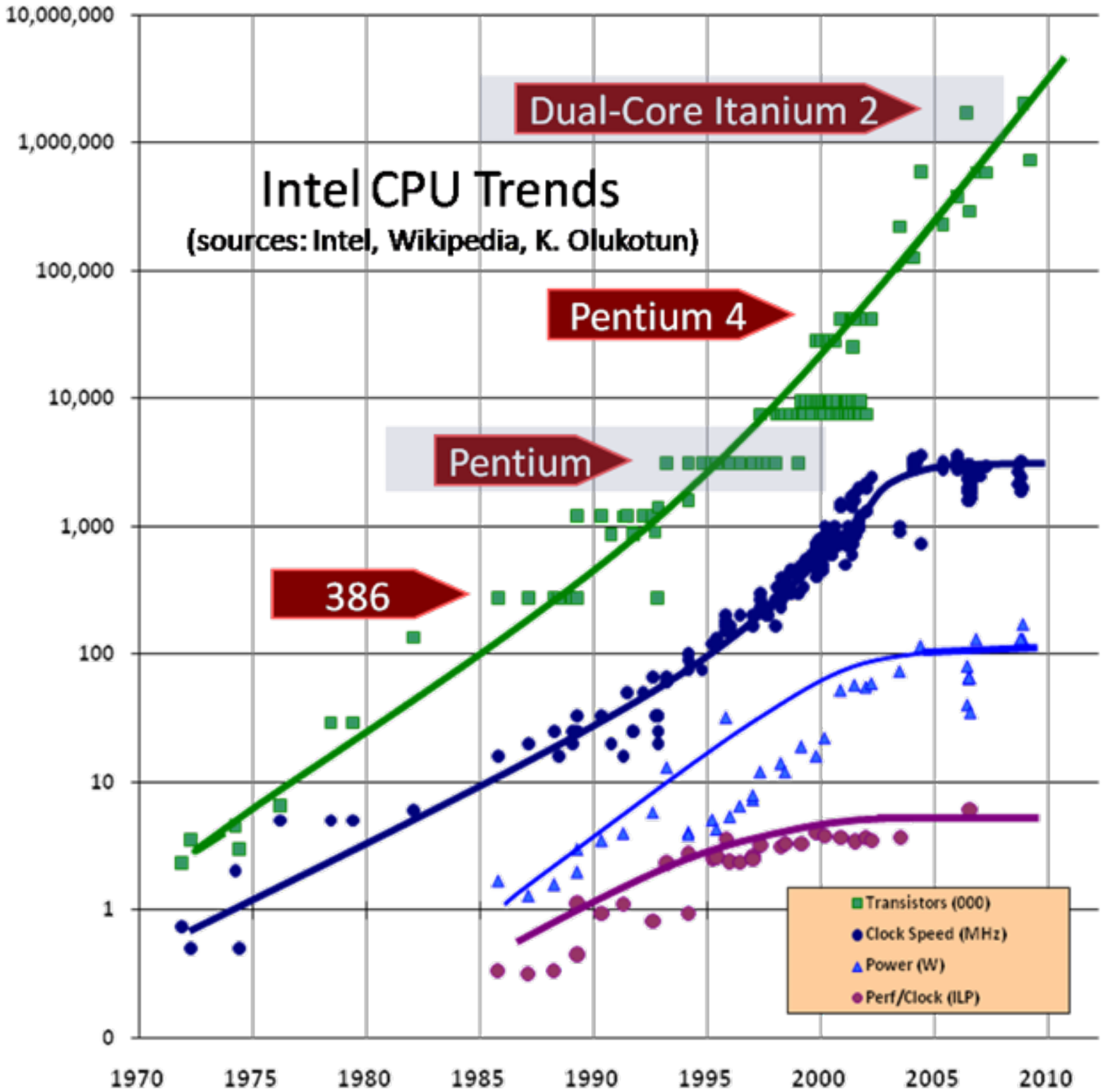
Phaser volle Energie ...

Reiseplan

- Engine Check (Bestandsaufnahme)
- Abschussparty (Phaser - Idee)
- Countdown (Phaser - Code)
- Ins Parralleluniversum (ForkJoinPool - Idee)
- Welcome to JDK7 (ForkJoinPool - Code)
- Touchdown (Zusammenfassung, Q&A)

Engine Check
(Bestandsaufnahme)

Why Concurrency?



Why Concurrency?

The Free Lunch Is Over
A Fundamental Turn Toward Concurrency in Software
By Herb Sutter

Applications will increasingly need to be concurrent if they want to fully exploit continuing exponential CPU throughput gains

Efficiency and performance optimization will get more, not less, important

Concurrency is the next major revolution in how we write software

http://www.gotw.ca/publications/concurrency-ddj.htm

Concurrency Problems

„It‘s hard to get it right …“

Concurrency Problems

„It‘s hard to get it right ...“
(from ForkJoinPool.java)

```
* Style notes: There are lots of inline assignments (of form
* "while ((local = field) != 0)") which are usually the simplest
* way to ensure the required read orderings (which are sometimes
* critical). Also several occurrences of the unusual "do {}
* while (!cas...)" which is the simplest way to force an update of
* a CAS'ed variable. There are also other coding oddities that
* help some methods perform reasonably even when interpreted (not
* compiled), at the expense of some messy constructions that
* reduce byte code counts.
```

Concurrency in JDK 6

java.util.concurrent

- Atomic:
  AtomicBoolean, AtomicInteger
- Executors:
  ExecutorService, Callable, Future
- Synchronizers:
  Semaphore, CountDownLatch, CyclicBarrier

Concurrency in JDK 7

java.util.concurrent

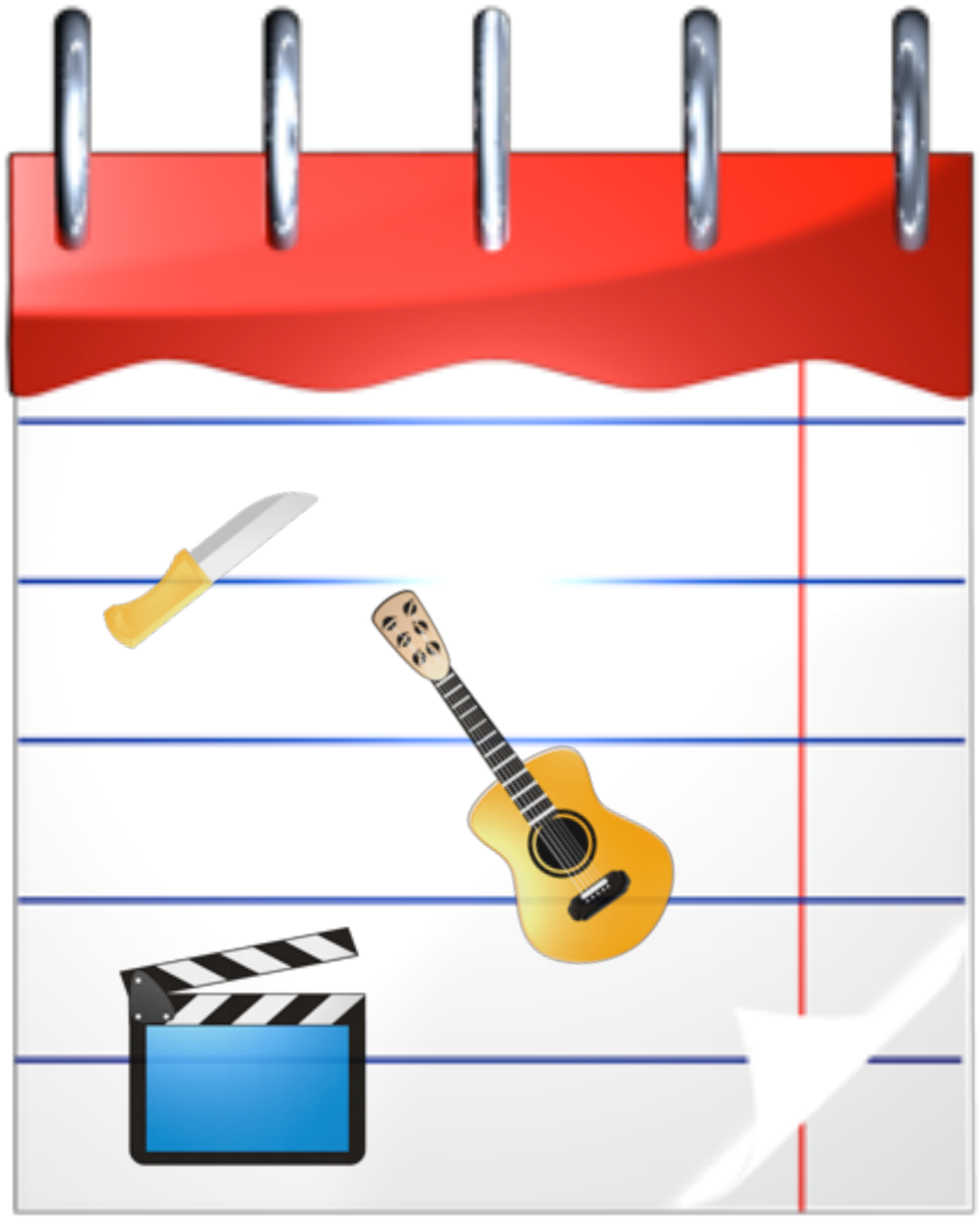
- ...
- Phaser
- ForkJoinPool

JSR 166y

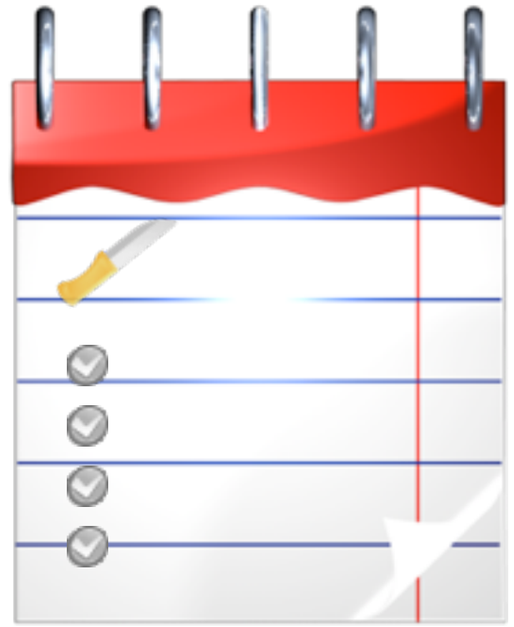
Abschussparty

(Phaser - Idee)

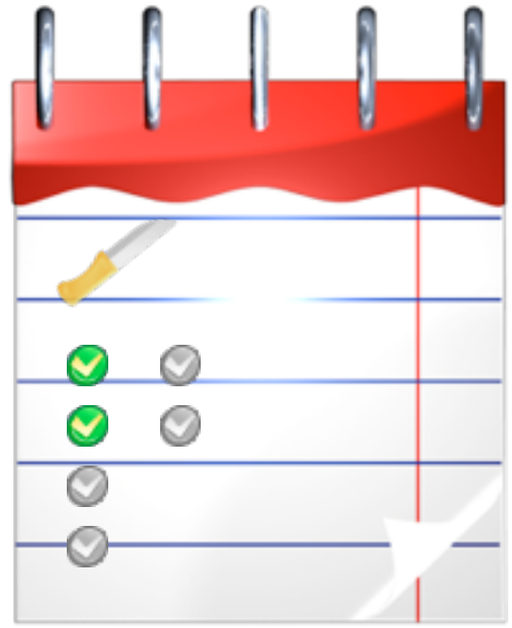
Phaser - Party



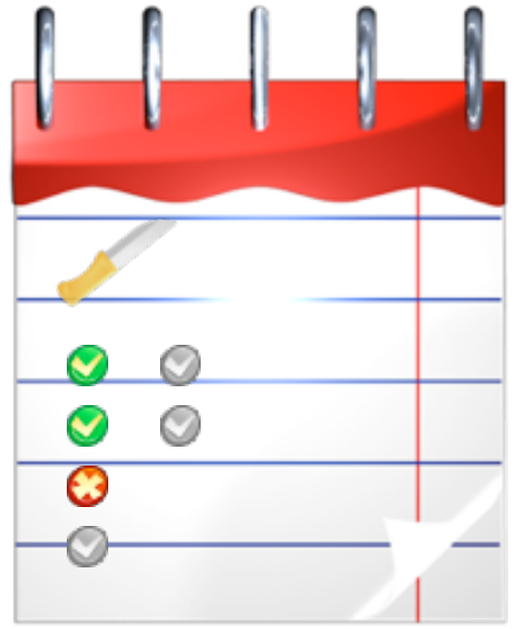
Phaser - Party



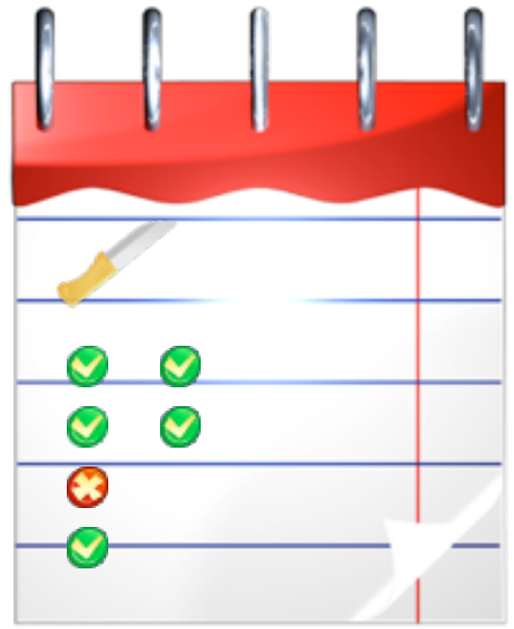
Phaser - Party



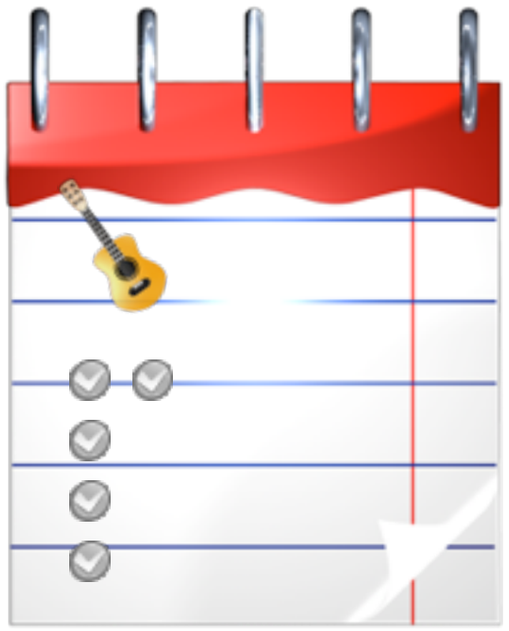
Phaser - Party



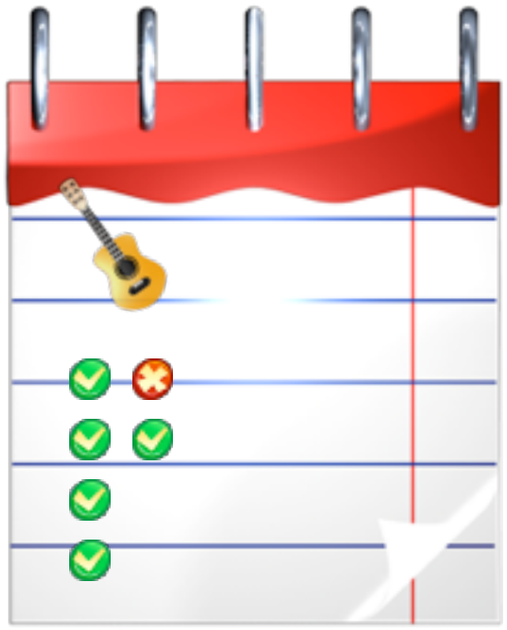
Phaser - Party



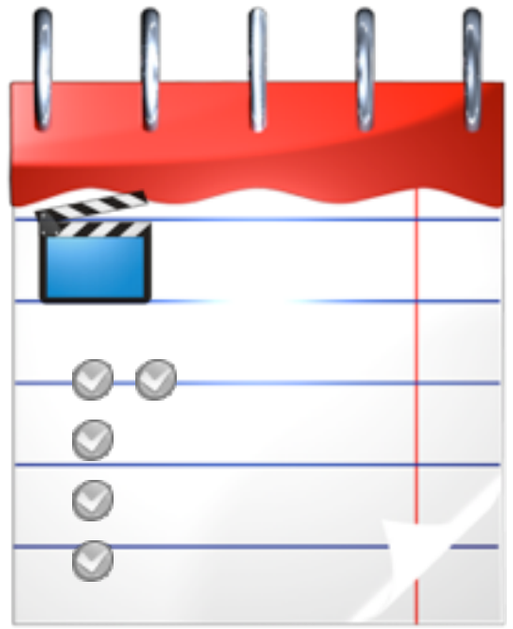
Phaser - Party



Phaser - Party



Phaser - Party



Countdown

(Phaser - Code)

Phaser API
Registration

- register()
- bulkRegister(int parties)
- new Phaser(int parties)
- arriveAndDeregister()

... registration and deregistration affect only internal counts ...

Phaser API
Synchronization

- arriveAndAwaitAdvance()
- arriveAndDeregister()
- arrive()
- awaitAdvance(int phase)

... each generation of a phaser has an associated phase number ...

Phaser: example

```
void runTasks(List<Runnable> tasks) {
  final Phaser phaser = new Phaser(1); // "1" to register self
  // create and start threads
  for (final Runnable task : tasks) {
    phaser.register();
    new Thread() {
      public void run() {
        phaser.arriveAndAwaitAdvance(); // await all creation
        task.run();
      }
    }.start();
  }

  // allow threads to start and deregister self
  phaser.arriveAndDeregister();
}
```

Phaser - Demo

Phaser API

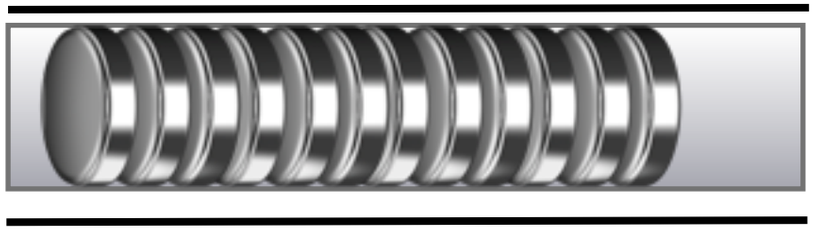
- Termination: isTerminated()
- Tiering: Phaser(Phaser parent)
- Advance:

  onAdvance(int phase, int registeredParties)

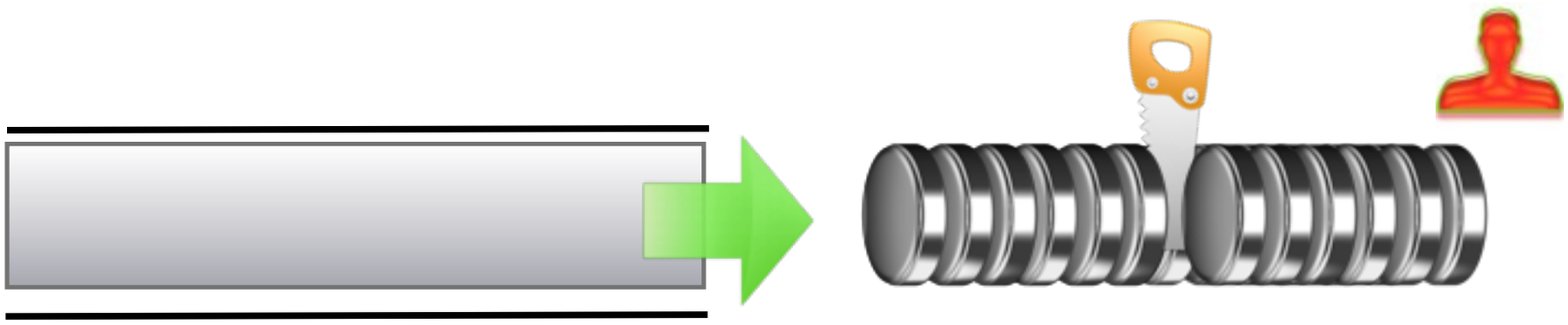
Ins Paralleluniversum

(ForkJoinPool - Idee)

ForkJoinPool

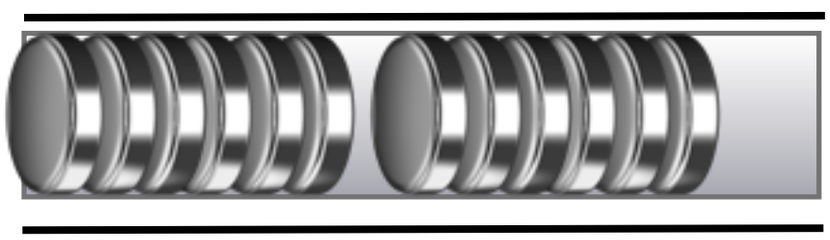


ForkJoinPool



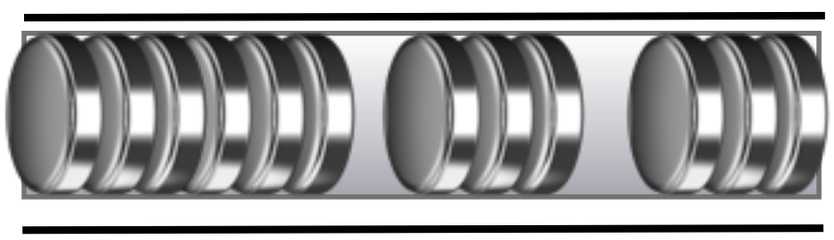
Fork

ForkJoinPool

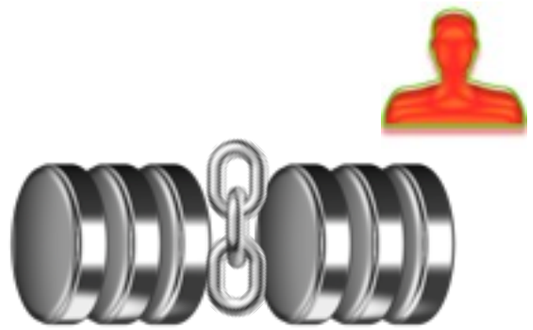


ForkJoinPool

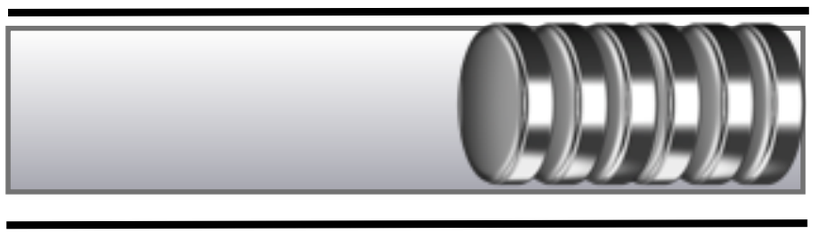
ForkJoinPool



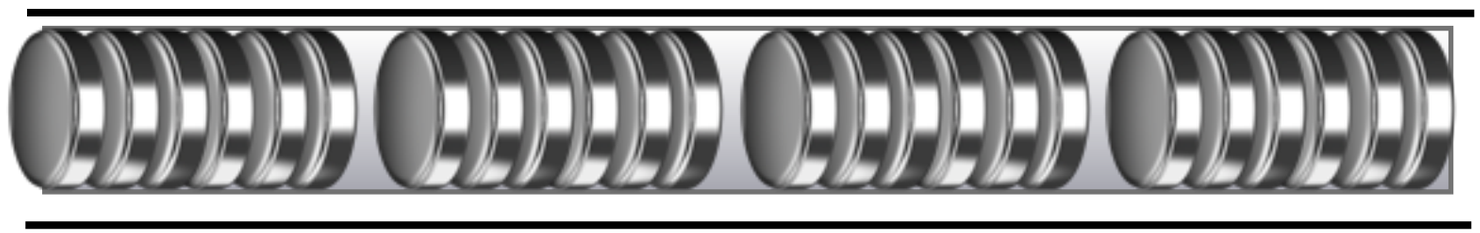
ForkJoinPool

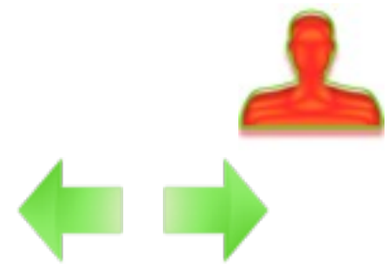


ForkJoinPool



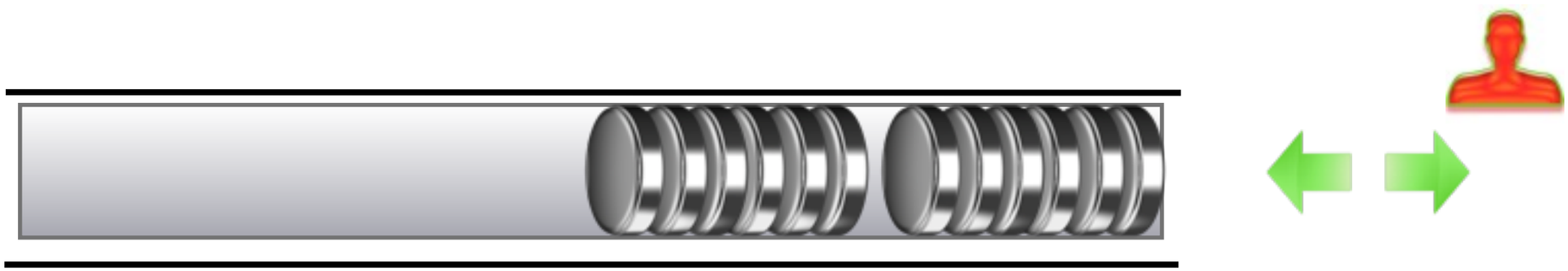
ForkJoinPool





Work Stealing

ForkJoinPool



Work Stealing

Welcome to JDK 7

(ForkJoinPool - Code)

ForkJoin API
ForkJoinPool

- differs from other ExecutorServices by virtue of employing work-stealing
- default parallelism: number of processors
- dynamically adds, suspends, or resumes worker threads

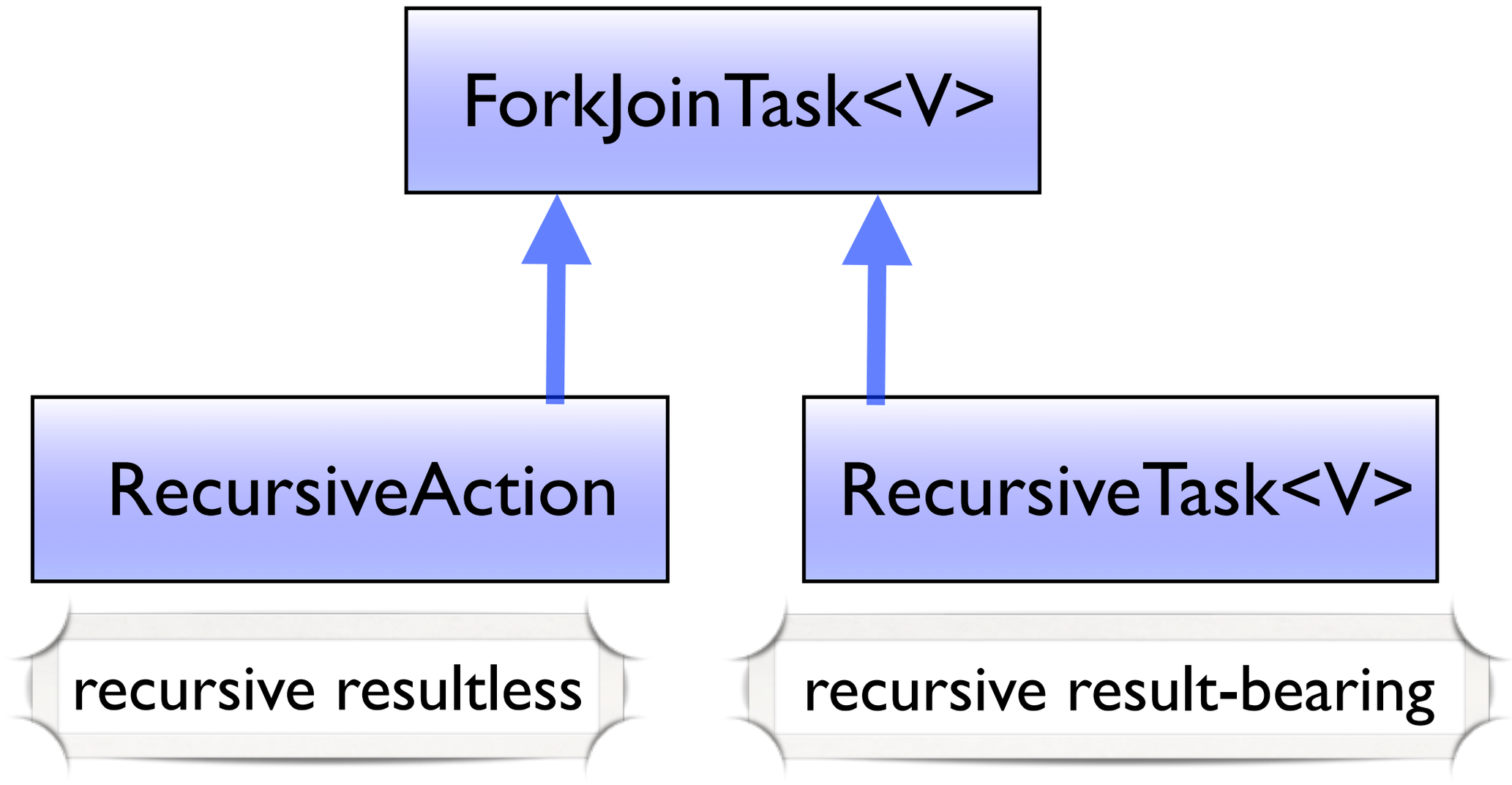
ForkJoinPool implements ExecutorService

ForkJoin API
ForkJoinPool

- execute(ForkJoinTask) // async, no result
- invoke(ForkJoinTask) // sync, result
- submit(ForkJoinTask) // async, future

ForkJoinPool implements ExecutorService

ForkJoin API
ForkJoinTask



ForkJoin API
ForkJoinTask<V>

- V compute()
- fork() // async execution
- V join() // sync get result
- ...

ForkJoinTask
implements Future

ForkJoin: example

```
public class ForkJoinWorker extends RecursiveTask<Integer> {

    private final int[]intArray;
    private final int start;
    private final int end;

    public ForkJoinWorker(int[]intArray, int start, int end) {
        this.intArray = intArray;
        this.start = start;
        this.end = end;
    }

    @Override
    protected Integer compute() {
```

ForkJoin: example

```
@Override
protected Integer compute() {
    if (end - start < SIZE_THRESHOLD) {
        return computeRest();
    } else {
        int mid = (start + end) >>> 1;
        ForkJoinWorker t1 =
                new ForkJoinWorker(intArray, start, mid);
        ForkJoinWorker t2 =
                new ForkJoinWorker(intArray, mid+1, end);

        t1.fork();
        t2.fork();

        return t1.join() + t2.join();
    }
}

private Integer computeRest() {
```

ForkJoin - Demo

ForkJoin API guidelines

- don‘t make too many small tasks:
  „ ... pick some minimum granularity size for which you always solve sequentially ...“

- getSurplusQueuedTaskCount()

ForkJoin API guidelines

- ForkJoinTasks:
  - „computational tasks calculating pure functions“
  - „should not perform blocking I/O“
  - „access variables, that are independent from other running tasks“

ForkJoin API exception handling

- ForkJoinTask.join():
  throws RuntimeException or Error

- ForkJoinTask.get():
  throws InterruptedException or ExecutionException

Touchdown

(Zusammenfassung)

Phaser - ForkJoinPool

- Phaser:
  - enhanced CountDownLatch
  - simple API
  - ready to use

Phaser - ForkJoinPool

- ForkJoinPool:
  - complex API
  - computing intensive tasks
  - requires tuning and performance testing

Phaser - ForkJoinPool

- ForkJoinPool:
  - preparation for Java 8

JSR#337 (Java 8)

Performance

The Fork/Join Framework introduced in Java SE 7 was a first step toward providing higher-level programming abstractions suitable for multicore CPUs. In this release we will take another big step by enhancing the Collections Framework and related APIs to support automatically-parallelizable bulk-data operations such as filter, map, and reduce. Convenient use of these new APIs will be enabled by extending the Java language to include lambda expressions (a.k.a. "closures") and default methods....

java.util.concurrent

use it!