Flavors of Concurrency

Oleg Šelajev
@shelajev
ZeroTurnaround

ZEROTURNAROUND

REBELLABS
by ZEROTURNAROUND

OLEG SHELAJEV
THE DEVELOPER ADVOCATE

PAST JOB: DRAGON CARETAKER

Fire and blood!

+372 5518336 SHELAJEV

OLEG@ZEROTURNAROUND.COM

LINKEDIN.COM/IN/SHELAJEV @SHELAJEV

VJUG

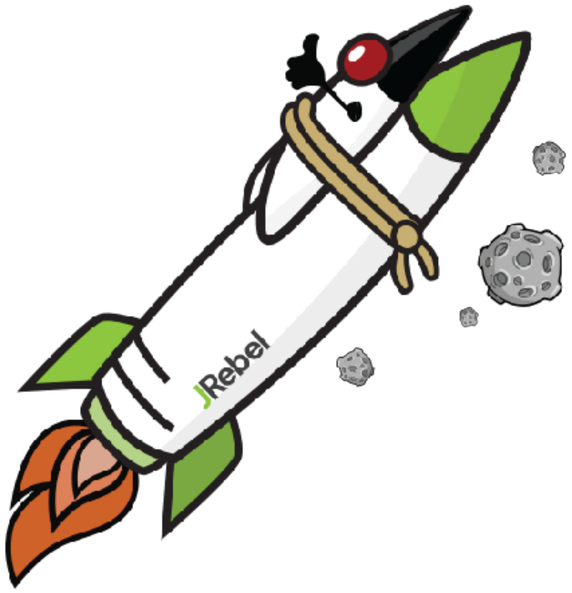
@shelajev

[R]EVOLUTIONARY
DEVELOPER TOOLS

ZEROTURNAROUND

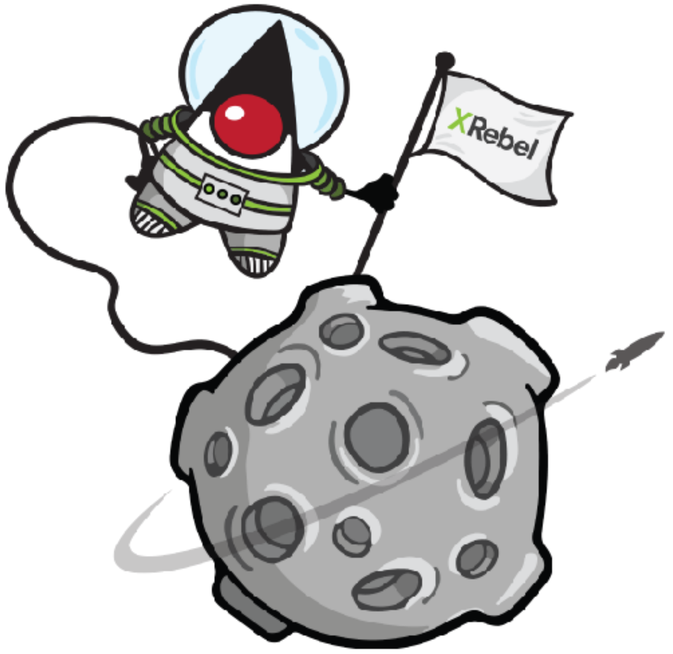
JRebel

RELOAD CODE CHANGES INSTANTLY

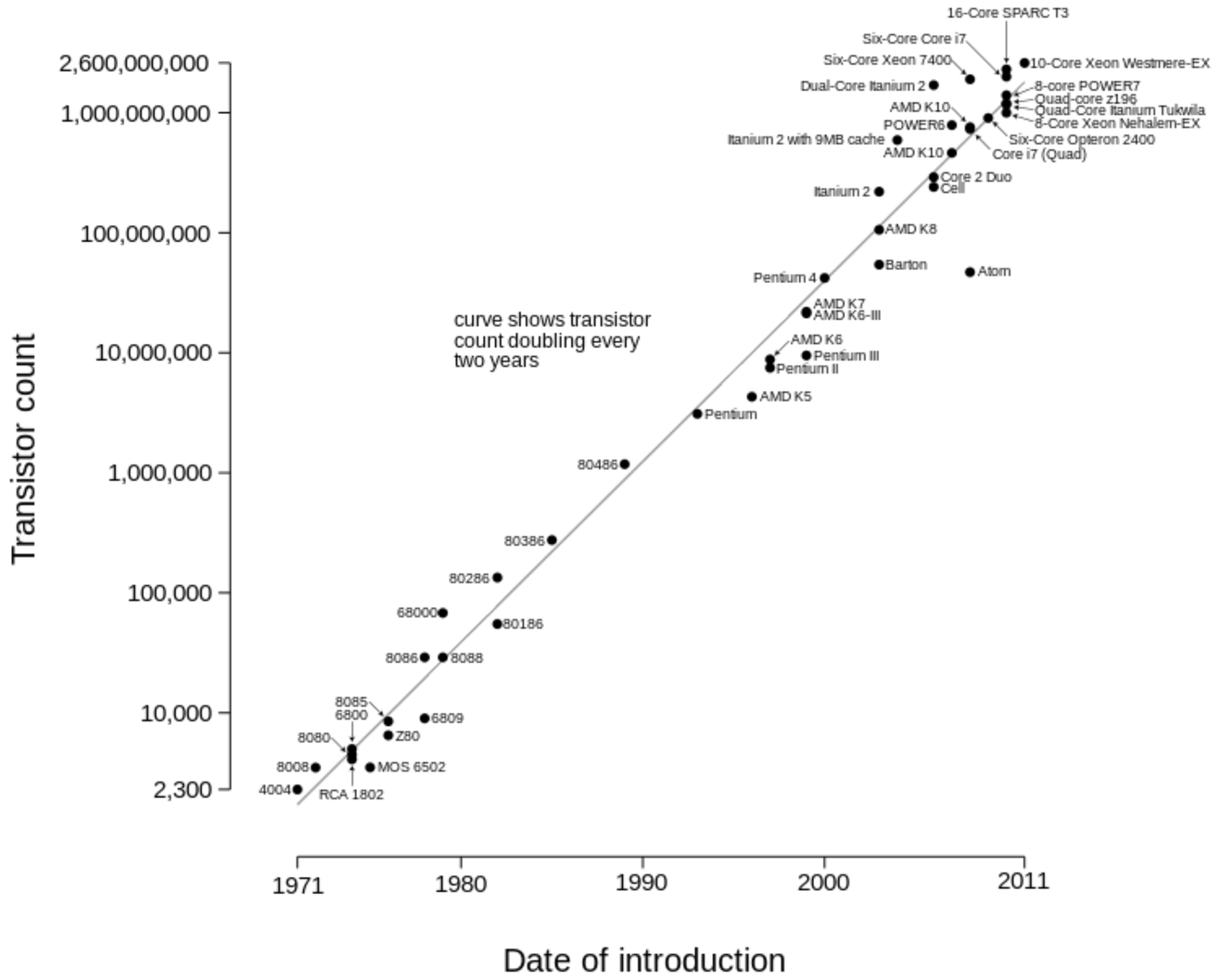


XRebel

THE LIGHTWEIGHT JAVA PROFILER



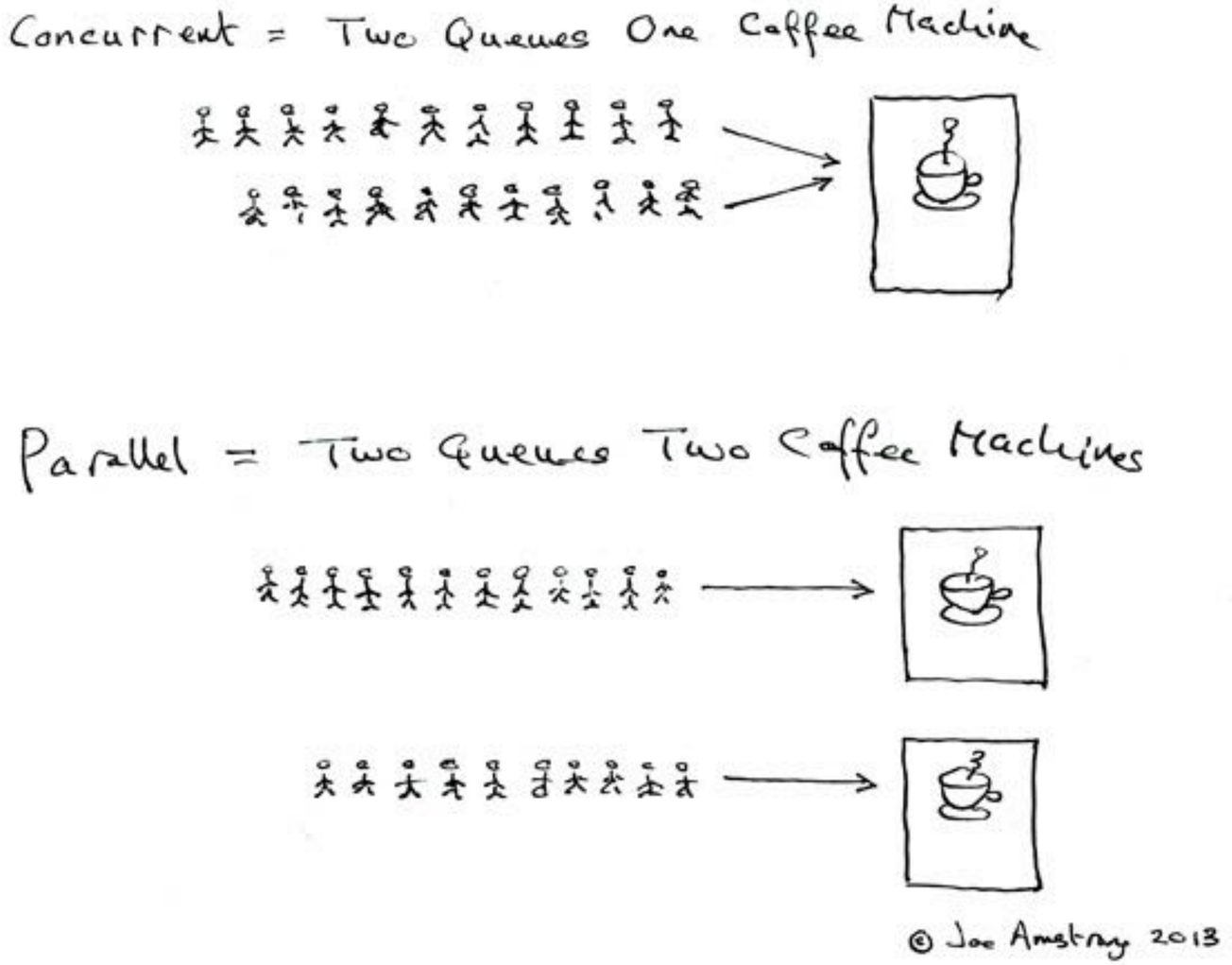
WHY DO WE CARE?



WHY DO WE CARE?

https://twitter.com/reubenbond/status/662061791497744384

CONCURRENCY | PARALLELISM



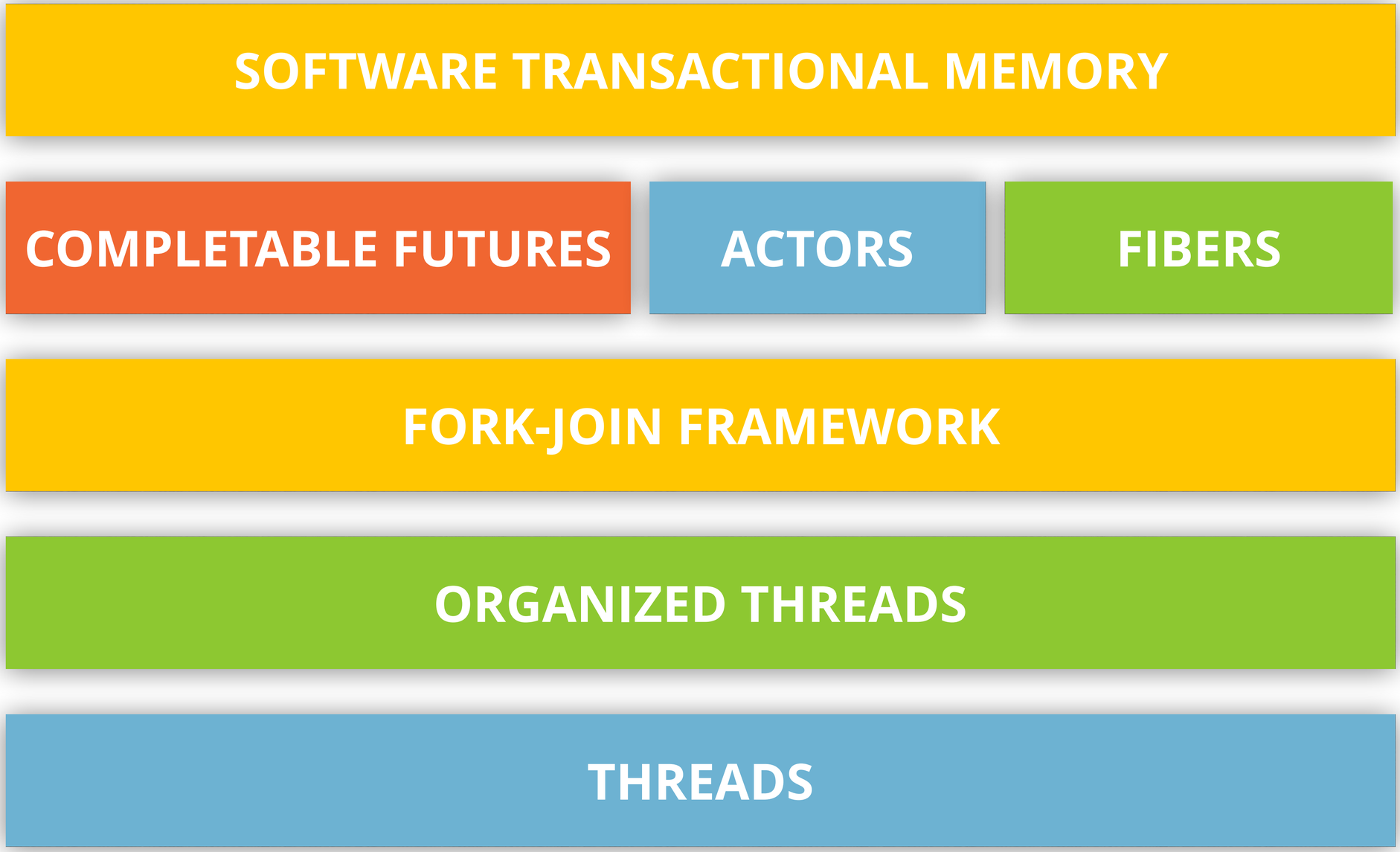
http://joearms.github.io/2013/04/05/concurrent-and-parallel-programming.html

CONCURRENCY

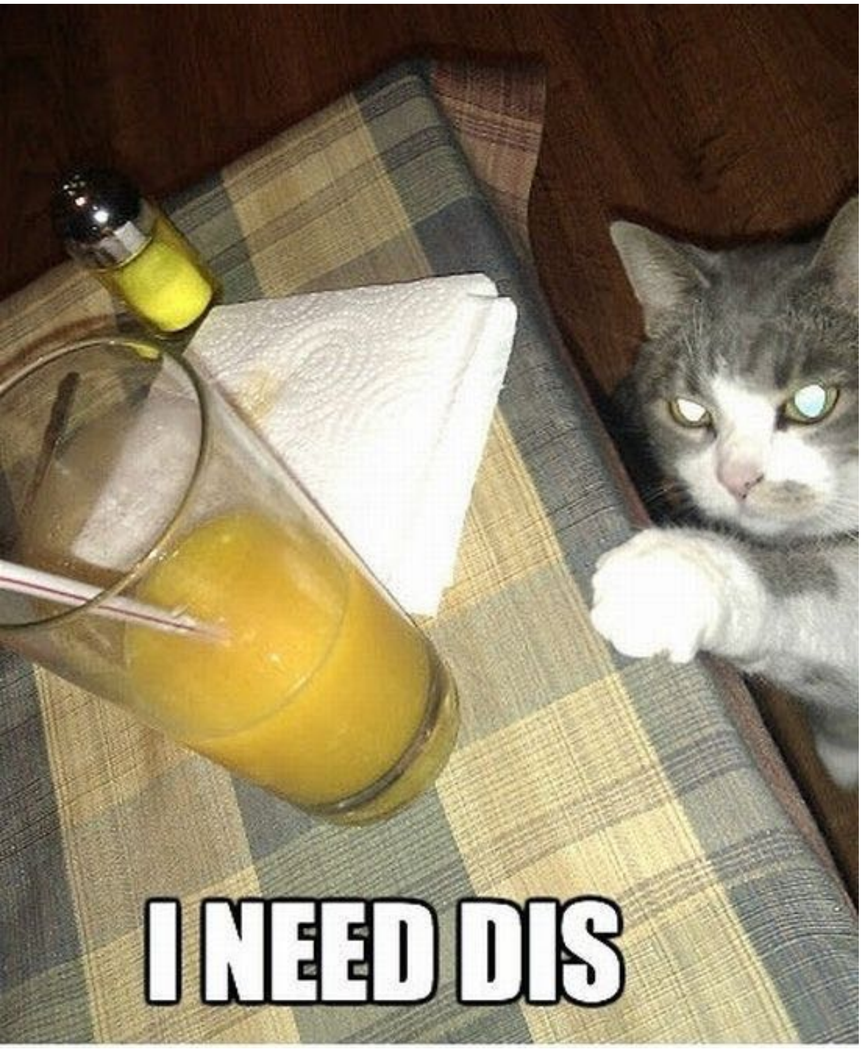
Shared resources
Multiple consumers & producers
Order of events
Locks & Deadlocks

THEORY

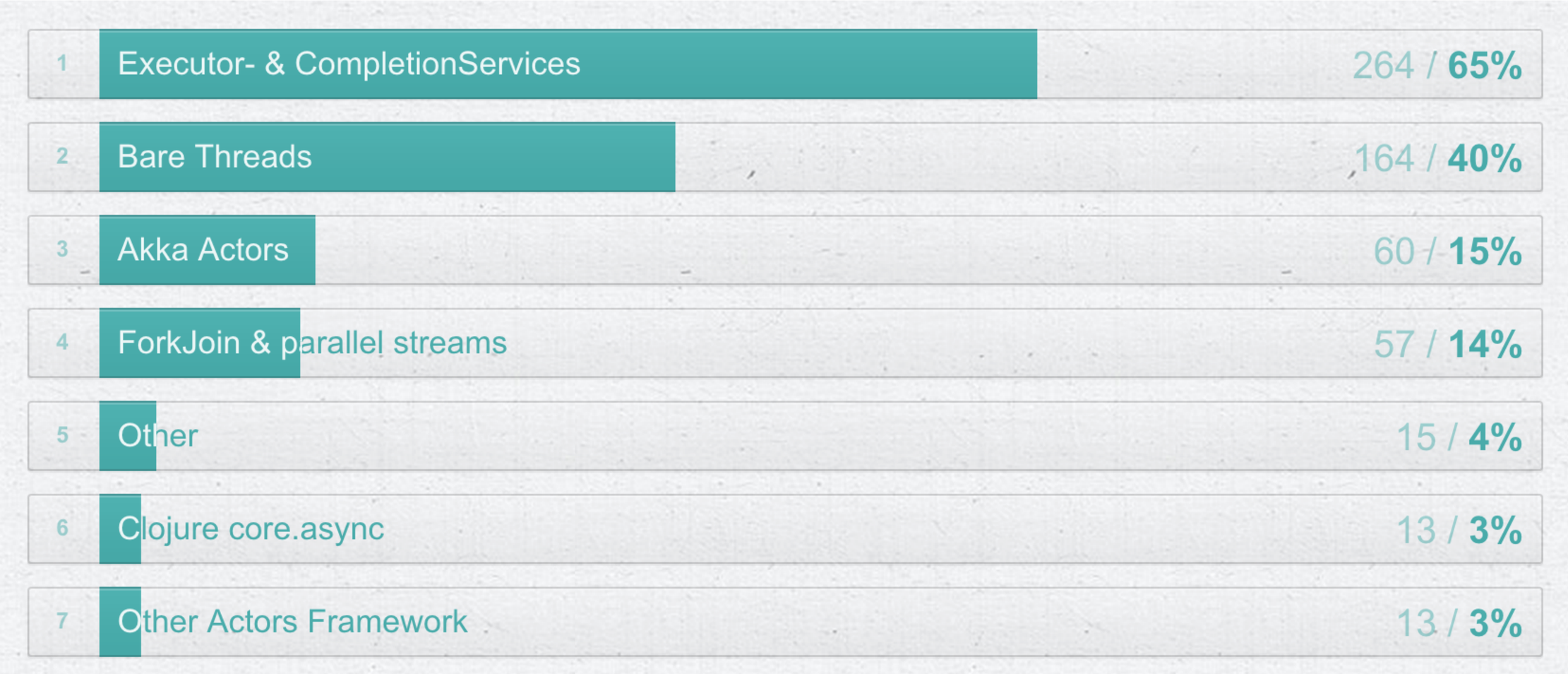
PRACTICE


SOFTWARE TRANSACTIONAL MEMORY
COMPLETABLE FUTURES
ACTORS
FIBERS
FORK-JOIN FRAMEWORK
ORGANIZED THREADS
THREADS

DEVS CHOOSING TECHNOLOGY



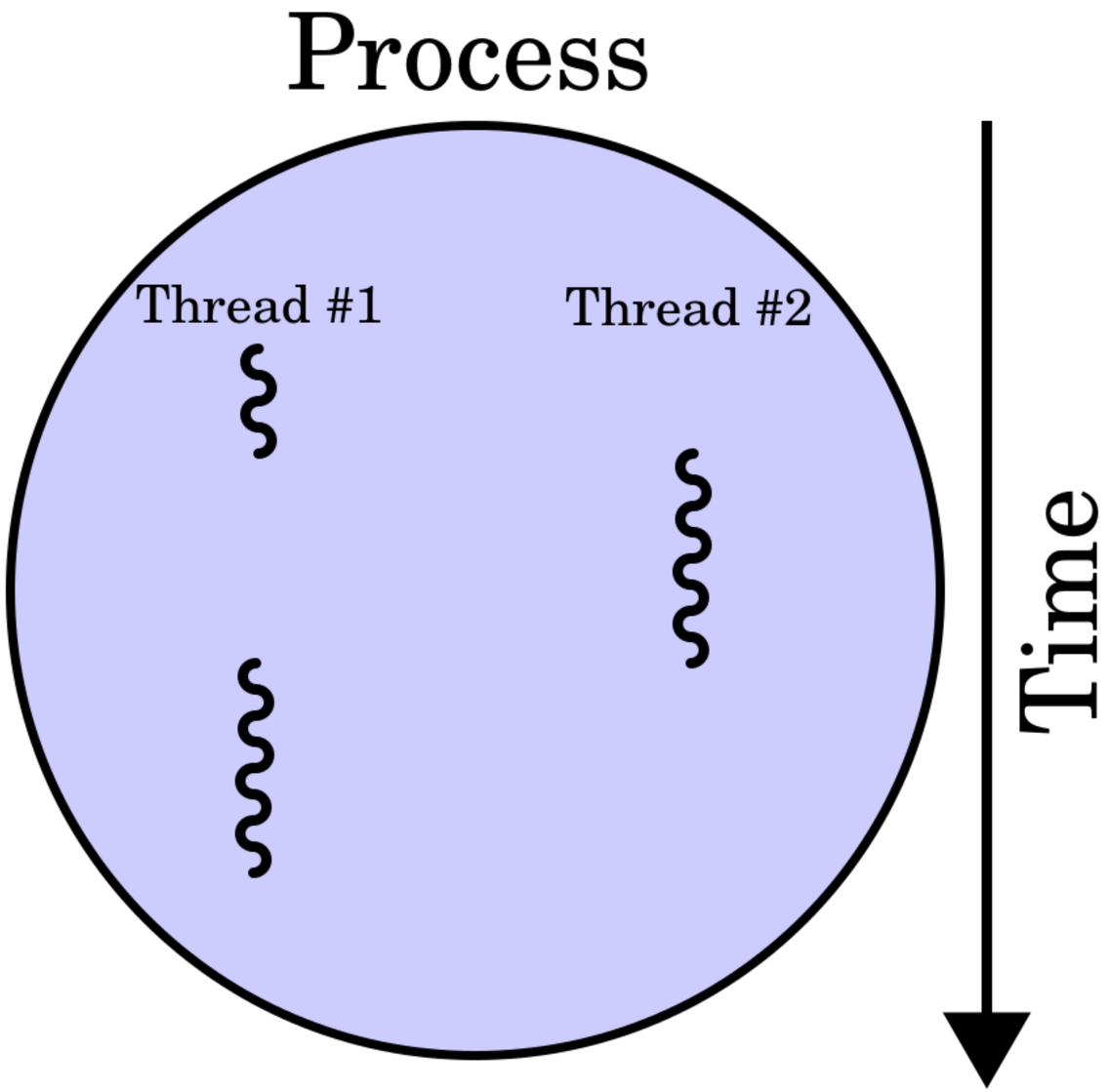
HOW TO MANAGE CONCURRENCY?



http://zeroturnaround.com/rebellabs/flavors-of-concurrency-in-java-threads-executors-forkjoin-and-actors/

THREADS

A **thread** of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler.


Process
Thread #1
Thread #2
Time

WHY THREADS

Model Hardware

Processes

Threads

COMMUNICATION

Objects & Fields

Atomics*

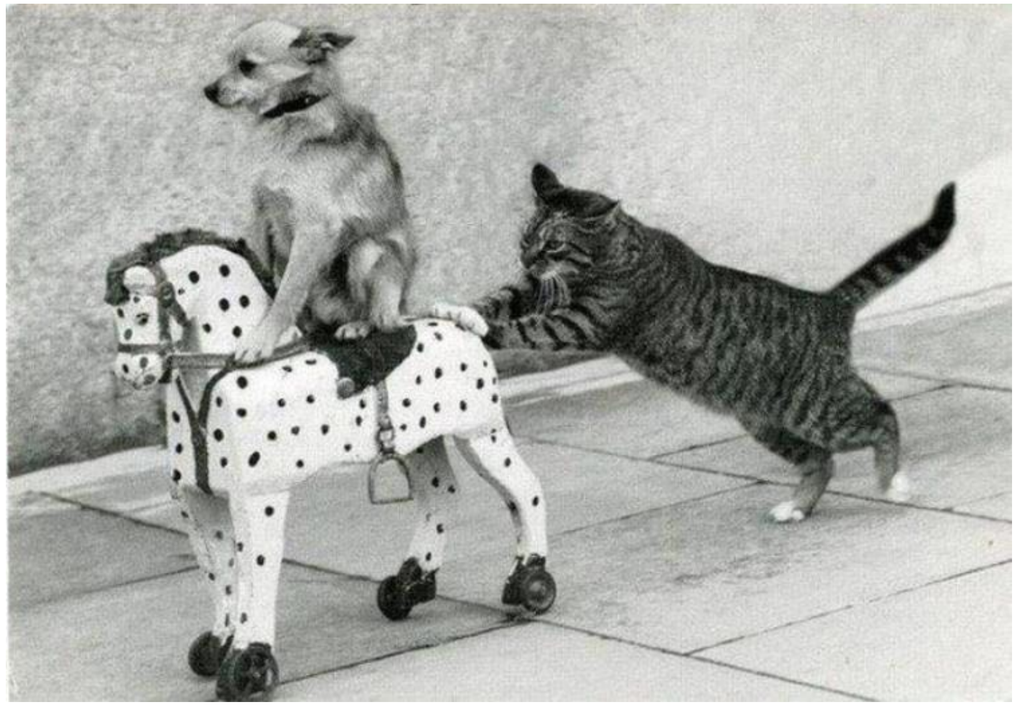
Queues

Database

COMMUNICATION

The Java Language Specification

17.4. Memory Model



PROBLEM ORIENTED PROGRAMMING

DuckDuckGo

Google

Wikipedia
The Free Encyclopedia

YAHOO!

PROBLEM

```
private static String
        getFirstResult(String question,
                       List<String> engines)
{
   // HERE BE DRAGONS, PARALLEL DRAGONS
   return null;
}
```

THREADS

```
private static String getFirstResult(String question,
List<String> engines) {
 AtomicReference<String> result = new AtomicReference<>();
 for(String base: engines) {
   String url = base + question;
   new Thread(() -> {
     result.compareAndSet(null, WS.url(url).get());
   }).start();
 }
 while(result.get() == null); // wait for the result to appear
 return result.get();
}
```

THREADS

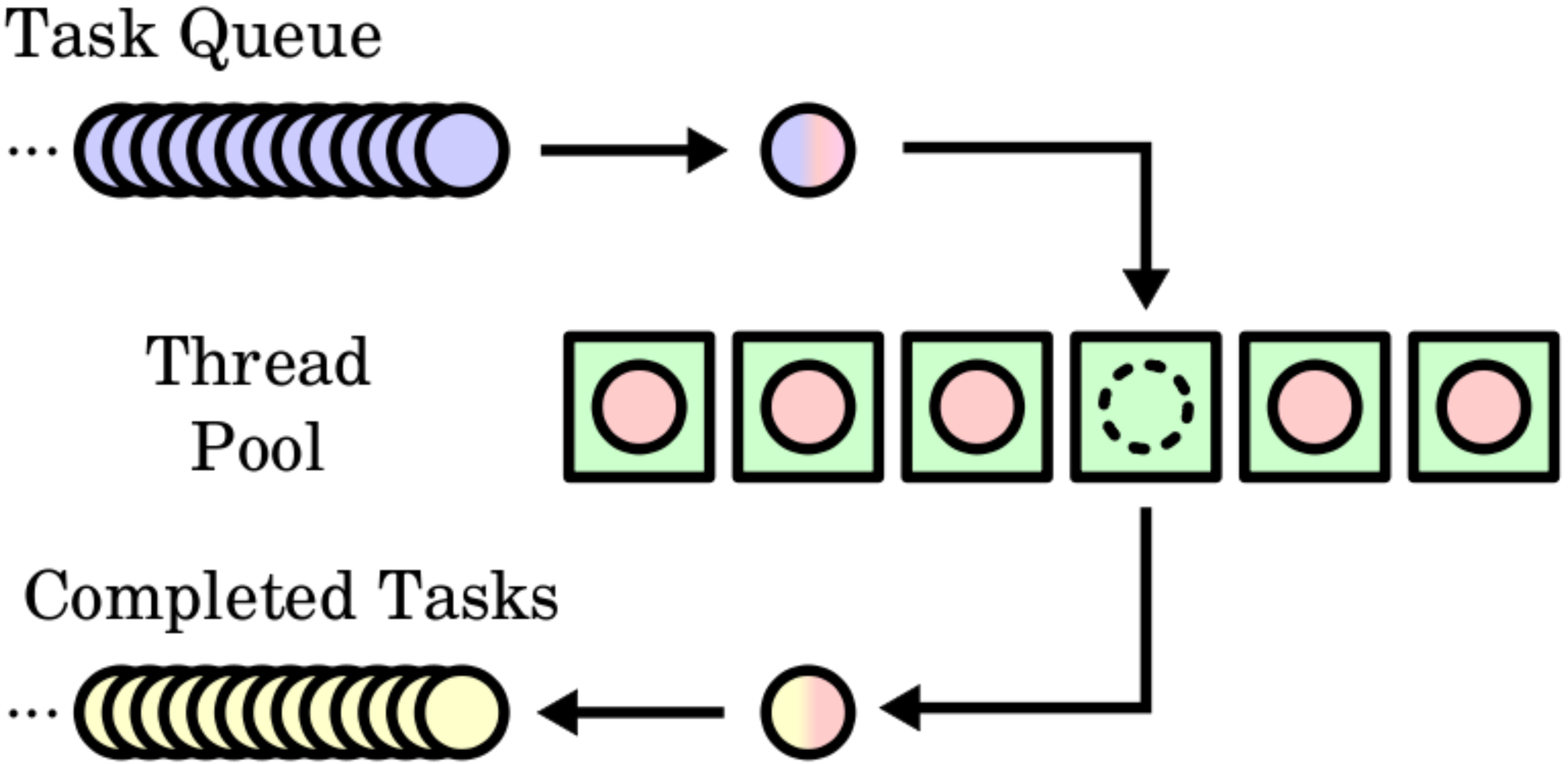
```
private static String getFirstResult(String question,
List<String> engines) {
 AtomicReference<String> result = new AtomicReference<>();
 for(String base: engines) {
   String url = base + question;
   new Thread(() -> {
     result.compareAndSet(null, WS.url(url).get());
   }).start();
 }
 while(result.get() == null); // wait for the result to appear
 return result.get();
}
```

TAKEAWAY: THREADS

Threads take resources
Require manual management
Easy
Not so simple

THREAD POOLS

A **thread pool** pattern consists of a number **m** of threads, created to perform a number **n** of tasks concurrently.


Task Queue
Thread Pool
Completed Tasks

EXECUTORS

```
public interface Executor {
  void execute(Runnable command);
}
```

```
public interface ExecutorCompletionService<V> {
  public Future<V> submit(Callable<V> task);
  public Future<V> take();
}
```

EXECUTORS

```
private static String getFirstResultExecutors(String question, List<String>
engines) {
 ExecutorCompletionService<String> service = new
ExecutorCompletionService<String>(Executors.newFixedThreadPool(4));

 for(String base: engines) {
   String url = base + question;
   service.submit(() -> {
     return WS.url(url).get();
   });
 }
   try {
     return service.take().get();
   }
   catch(InterruptedException | ExecutionException e) {
     return null;
   }
}
```

EXECUTORS

```
private static String getFirstResultExecutors(String question, List<String>
engines) {
 ExecutorCompletionService<String> service = new
ExecutorCompletionService<String>(Executors.newFixedThreadPool(4));

 for(String base: engines) {
   String url = base + question;
   service.submit(() -> {
     return WS.url(url).get();
   });
 }
   try {
     return service.take().get();
   }
   catch(InterruptedException | ExecutionException e) {
     return null;
   }
}
```

EXECUTORS

```
private static String getFirstResultExecutors(String question, List<String>
engines) {
 ExecutorCompletionService<String> service = new
ExecutorCompletionService<String>(Executors.newFixedThreadPool(4));

 for(String base: engines) {
   String url = base + question;
   service.submit(() -> {
     return WS.url(url).get();
   });
 }
   try {
     return service.take().get();
   }
   catch(InterruptedException | ExecutionException e) {
     return null;
   }
}
```

EXECUTORS

```
private static String getFirstResultExecutors(String question, List<String>
engines) {
 ExecutorCompletionService<String> service = new
ExecutorCompletionService<String>(Executors.newFixedThreadPool(4));

 for(String base: engines) {
   String url = base + question;
   service.submit(() -> {
     return WS.url(url).get();
   });
 }
   try {
     return service.take().get();
   }
   catch(InterruptedException | ExecutionException e) {
     return null;
   }
}
```

CONCERNS: EXECUTORS

Queue size
Overflow strategy
Cancellation strategy

TAKEAWAY: EXECUTORS

Simple configuration
Bounded overhead
Pushing complexity deeper


http://i.ytimg.com/vi/6_W_xLWtNa0/maxresdefault.jpg

FORK JOIN FRAMEWORK

Recursive tasks

General parallel tasks

Work stealing

FORK JOIN POOL

```
private static String getFirstResult(String question,
                                     List<String> engines) {
 // get element as soon as it is available
 Optional<String> result = engines.stream()
                                  .parallel().map((base) ->
 {
   String url = base + question;
   return WS.url(url).get();
 }).findAny();
 return result.get();
}
```

MANAGED BLOCKER
BY DR. HEINZ M. KABUTZ

Blocking methods should not be called from within parallel streams in Java 8, otherwise the shared threads in the common ForkJoinPool will become inactive. In this newsletter we look at a technique to maintain a certain liveliness in the pool, even when some threads are blocked.

http://www.javaspecialists.eu/archive/Issue223.html

TAKEAWAY: FORK JOIN POOL

Efficient

Preconfigured

Easy to get right

Easy to get wrong


HENDO

COMPLETABLE FUTURES

```
private static String getFirstResultCompletableFuture(String question, List<String>
engines) {
 CompletableFuture result = CompletableFuture.anyOf(engines.stream().map(
   (base) -> {
     return CompletableFuture.supplyAsync(() -> {
       String url = base + question;
       return WS.url(url).get();
     });
   }
 ).collect(Collectors.toList()).toArray(new CompletableFuture[0]));

 try {
   return (String) result.get();
 }
 catch (InterruptedException | ExecutionException e) {
   return null;
 }
}
```

COMPLETABLE FUTURES

```
private static String getFirstResultCompletableFuture(String question, List<String>
engines) {
 CompletableFuture result = CompletableFuture.anyOf(engines.stream().map(
   (base) -> {
     return CompletableFuture.supplyAsync(() -> {
       String url = base + question;
       return WS.url(url).get();
     });
   }
 ).collect(Collectors.toList()).toArray(new CompletableFuture[0]));

 try {
   return (String) result.get();
 }
 catch (InterruptedException | ExecutionException e) {
   return null;
 }
}
```

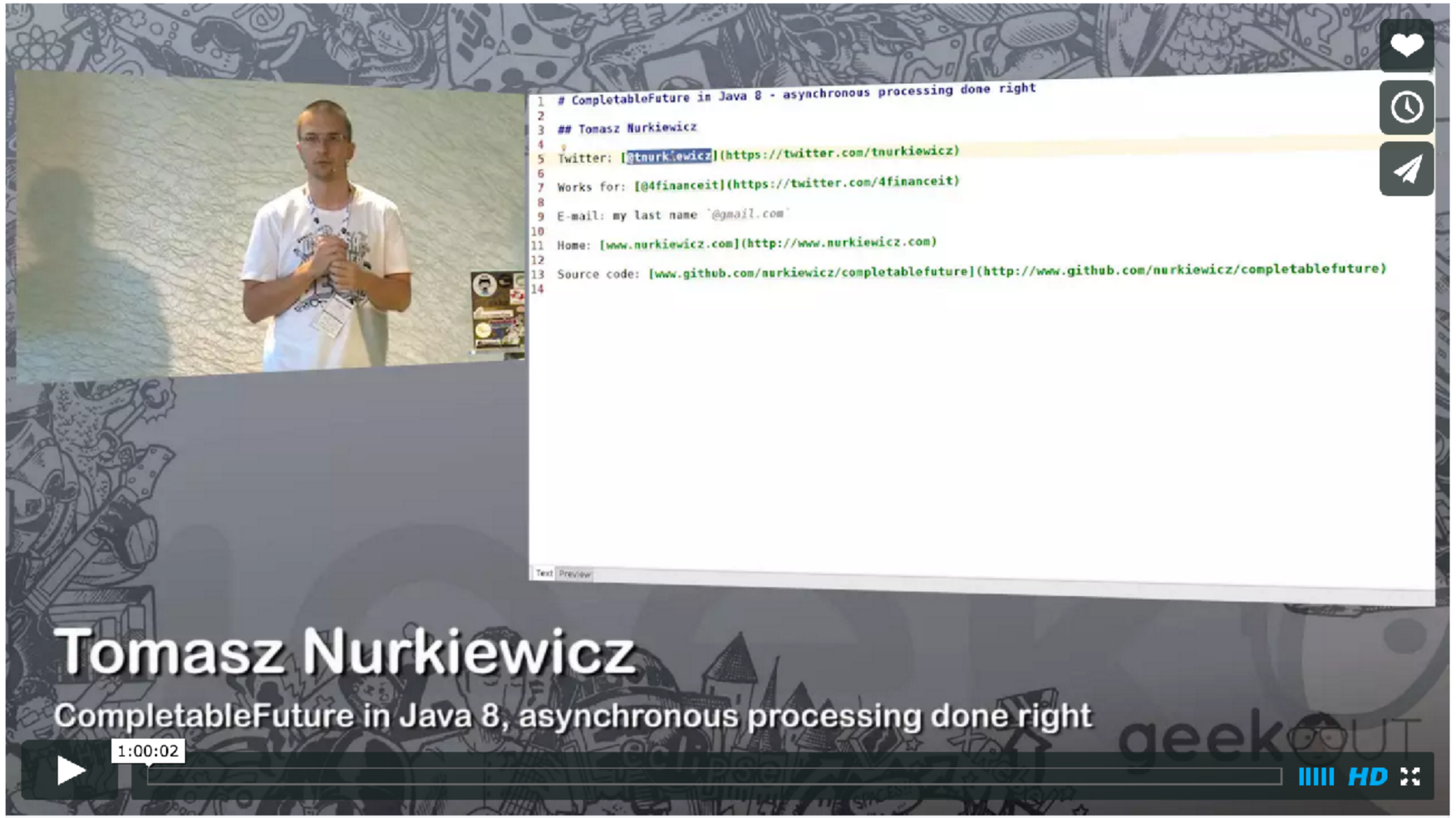
Using types

java.util.concurrent.**CompletableFuture**

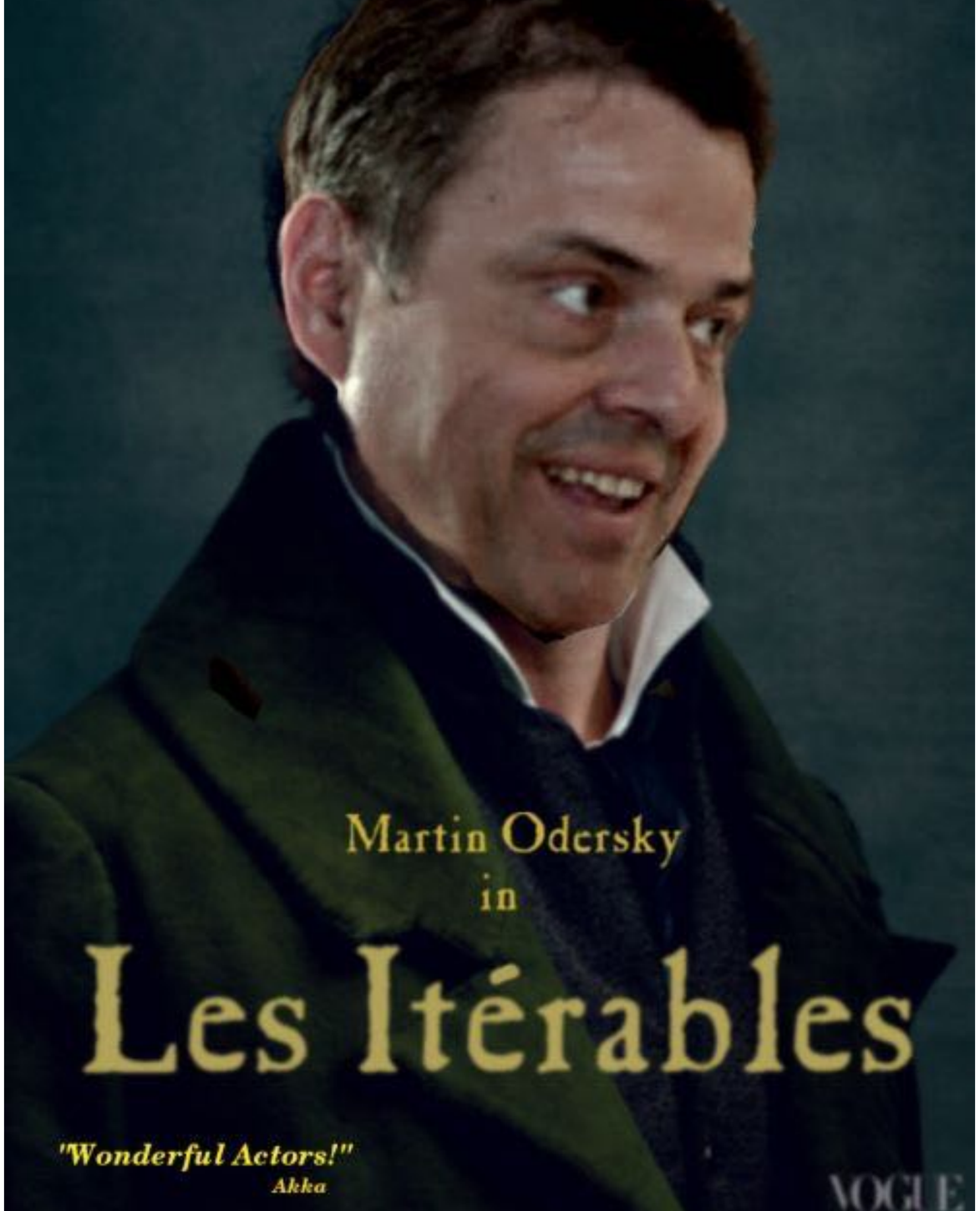
thenApply(Function / Consumer / etc)

thenApplyAsync(Function / etc)

Completable Future



https://vimeo.com/131394616


Martin Odersky
in
Les Itérables
"Wonderful Actors!"
Akka
VOGUE

ACTORS

"**Actors**" are the universal primitives of concurrent computation.

ACTORS

```
static class Message {
 String url;
 Message(String url) {this.url = url;}
}

static class Result {
 String html;
 Result(String html) {this.html = html;}
}
```

ACTOR SYSTEM

```
ActorSystem system = ActorSystem.create("Search");

final ActorRef q = system.actorOf(
  Props.create(
   (UntypedActorFactory) () ->
      new UrlFetcher, "master");
q.tell(new Message(url), ActorRef.noSender());
```

ACTORS

```
static class UrlFetcher extends UntypedActor {

  @Override
  public void onReceive(Object message) throws Exception {
    if (message instanceof Message) {
      Message work = (Message) message;
      String result = WS.url(work.url).get();
      getSender().tell(new Result(result), getSelf());
    } else {
      unhandled(message);
    }
  }
}
```

ACTORS

```
static class UrlFetcher extends UntypedActor {

  @Override
  public void onReceive(Object message) throws Exception {
    if (message instanceof Message) {
      Message work = (Message) message;
      String result = WS.url(work.url).get();
      getSender().tell(new Result(result), getSelf());
    } else {
      unhandled(message);
    }
  }
}
```

TYPED ACTORS

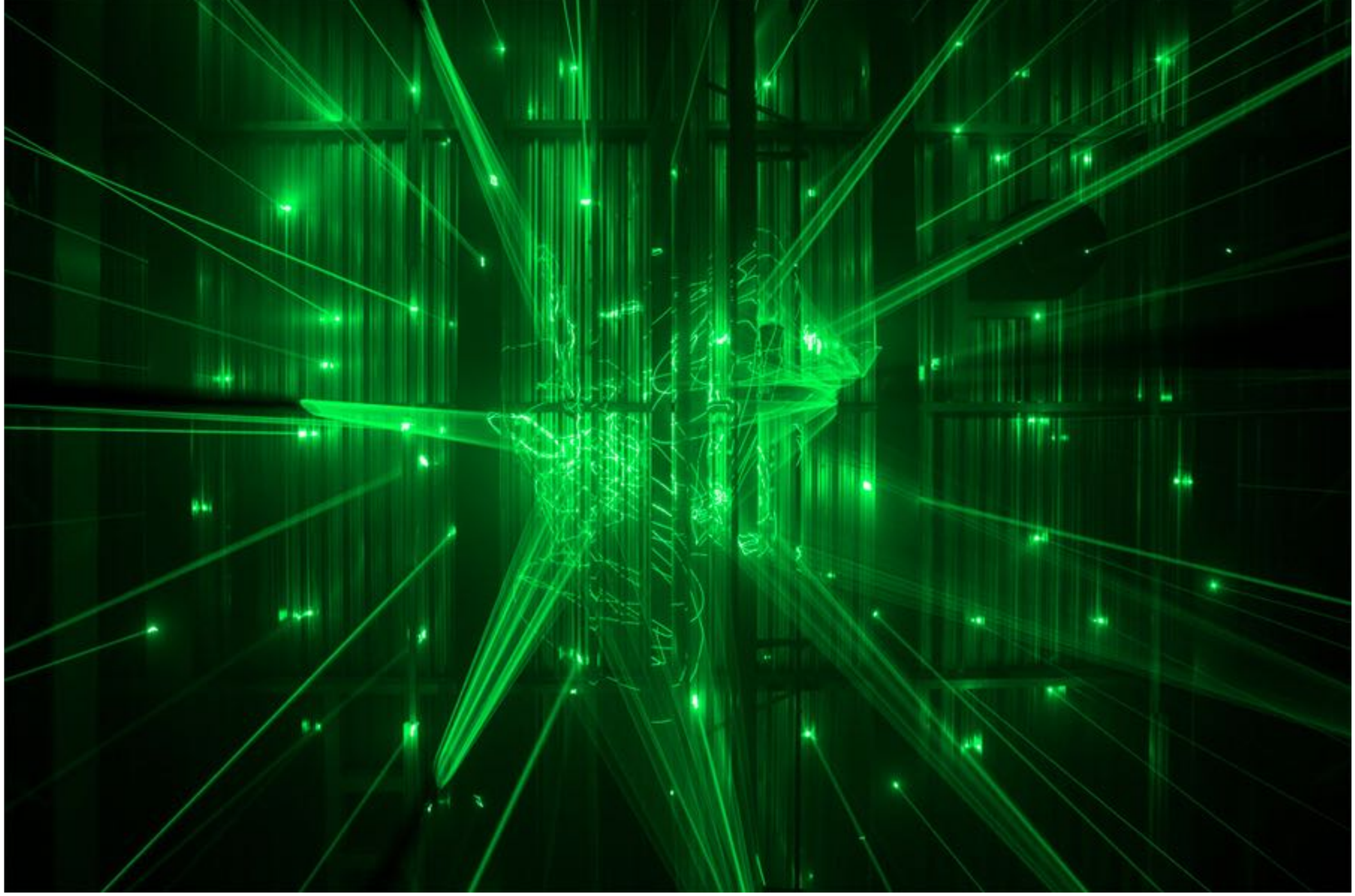
```
public static interface Squarer {

  //fire-forget
  void squareDontCare(int i);
  //non-blocking send-request-reply
  Future<Integer> square(int i);

  //blocking send-request-reply
  Option<Integer> squareNowPlease(int i);

  //blocking send-request-reply
  int squareNow(int i);
}
```

TAKEAWAY: ACTORS

OOP is about messages

Multiplexing like a boss

Supervision & Fault tolerance



http://static1.squarespace.com/static/50aa813ce4b0726ad3f3fdb0/51e1e393e4b0180cf3bcbb46/51e1e39ae4b0fa94cee721f9/1373758363472/forest-by-marshmallow-laser-feast-the-tree-mag-30.jpg

FIBERS

Lightweight threads, that are scheduled by the custom scheduler.

puniverse / quasar

Watch 98

Fibers, Channels and Actors for the JVM http://docs.paralleluniverse.co/quasar/

1,938 commits | **12** branches | **10** releases | **7** contributors

branch: **master** quasar / +

fix up-to-date data so that tasks won't run unnecessarily

pron authored 9 minutes ago — latest commit 602e6c9524

baselib	upgrade HdrHistogram	5 months ago
docs	add reactive streams doc	2 hours ago
quasar-actors/src	remove use of markdown in javadoc	2 hours ago
quasar-core/src	Fix potential ClassCastException	3 hours ago

QUASAR FIBERS

```
@Suspendable
public void myMethod(Input x) {
  x.f();
}
```

QUASAR FIBERS

```
new Fiber<V>() {
  @Override
  protected V run()
    throws SuspendExecution, InterruptedException {
        // your code
    }
}.start();
```

FIBERS

Quasar “freezes” the fiber’s stack into a **continuation task** that can be re-schedule later on.

TAKEAWAY: FIBERS

Continuations

Progress all over the place

Bytecode modification

Highly scalable

TRANSACTIONAL MEMORY

STM

```
import akka.stm.*;

final Ref<Integer> ref = new Ref<Integer>(0);

new Atomic() {
    public Object atomically() {
        return ref.set(5);
    }
}.execute();
```

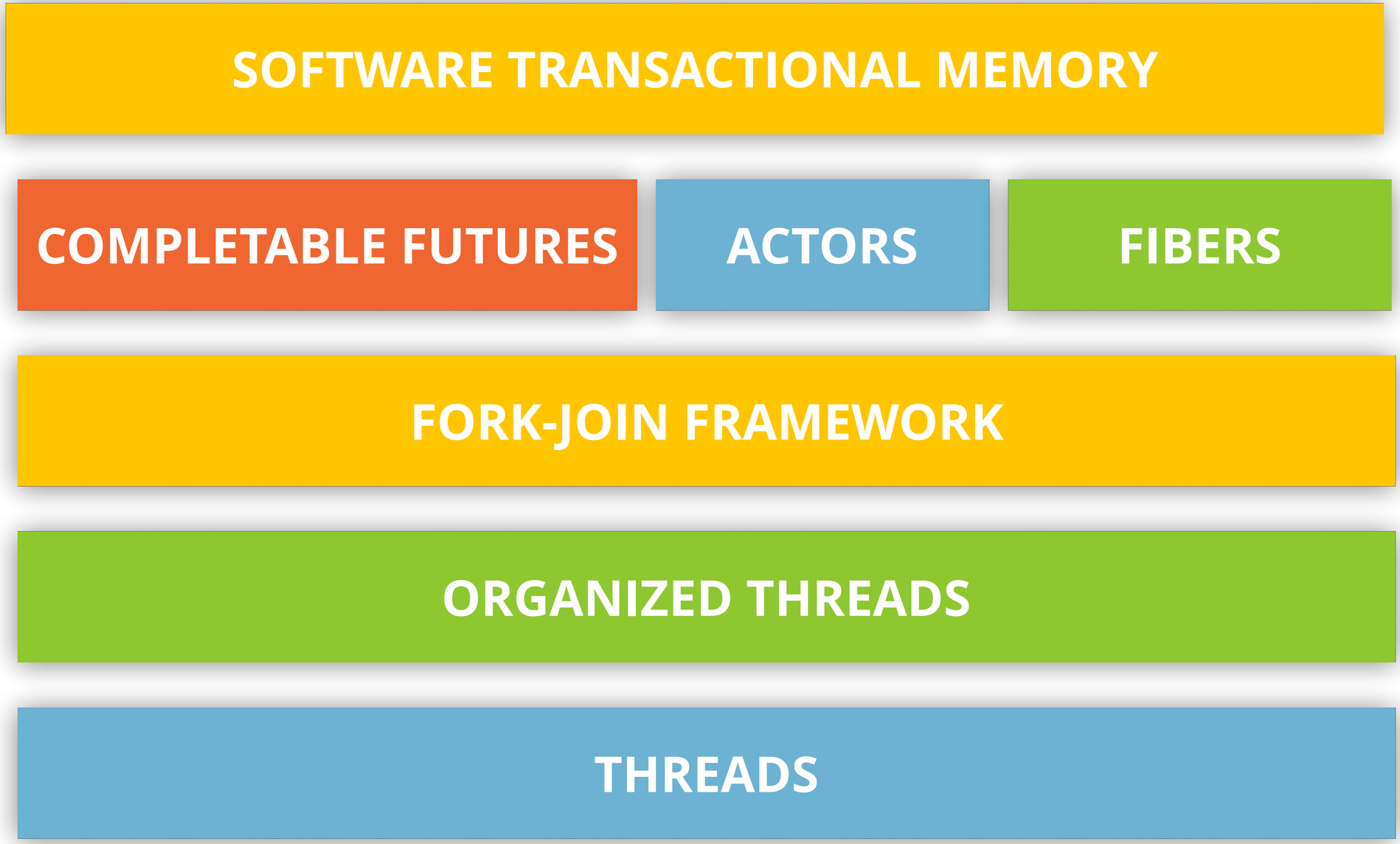
STM

```
final TransactionalMap<String, User> users = new
TransactionalMap<String, User>();

// fill users map (in a transaction)
new Atomic() {
    public Object atomically() {
        users.put("bill", new User("bill"));
        users.put("mary", new User("mary"));
        users.put("john", new User("john"));
        return null;
    }
}.execute();
```

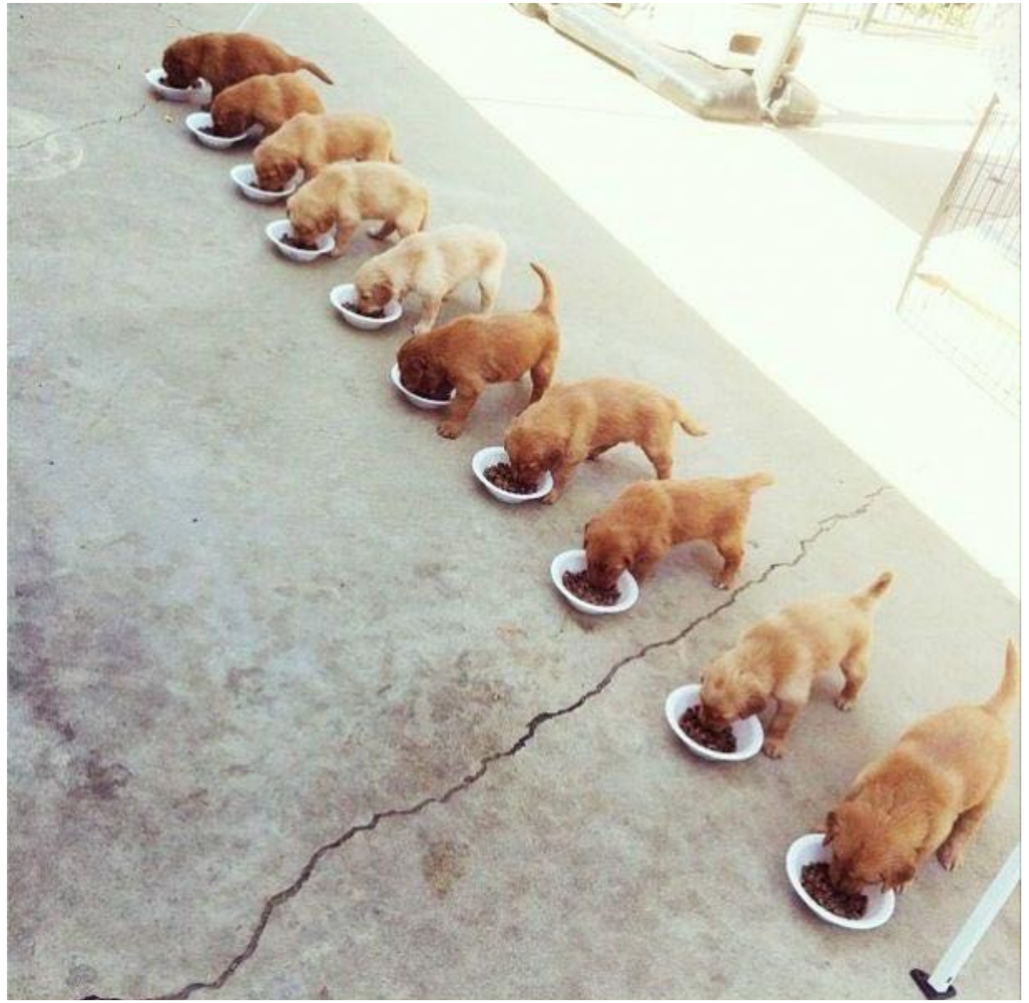
TAKEAWAY: TX MEMORY

Retry / Fail

ACID


SOFTWARE TRANSACTIONAL MEMORY
COMPLETABLE FUTURES
ACTORS
FIBERS
FORK-JOIN FRAMEWORK
ORGANIZED THREADS
THREADS

CONCLUSION



Programming Wisdom
@CodeWisdom

Follow

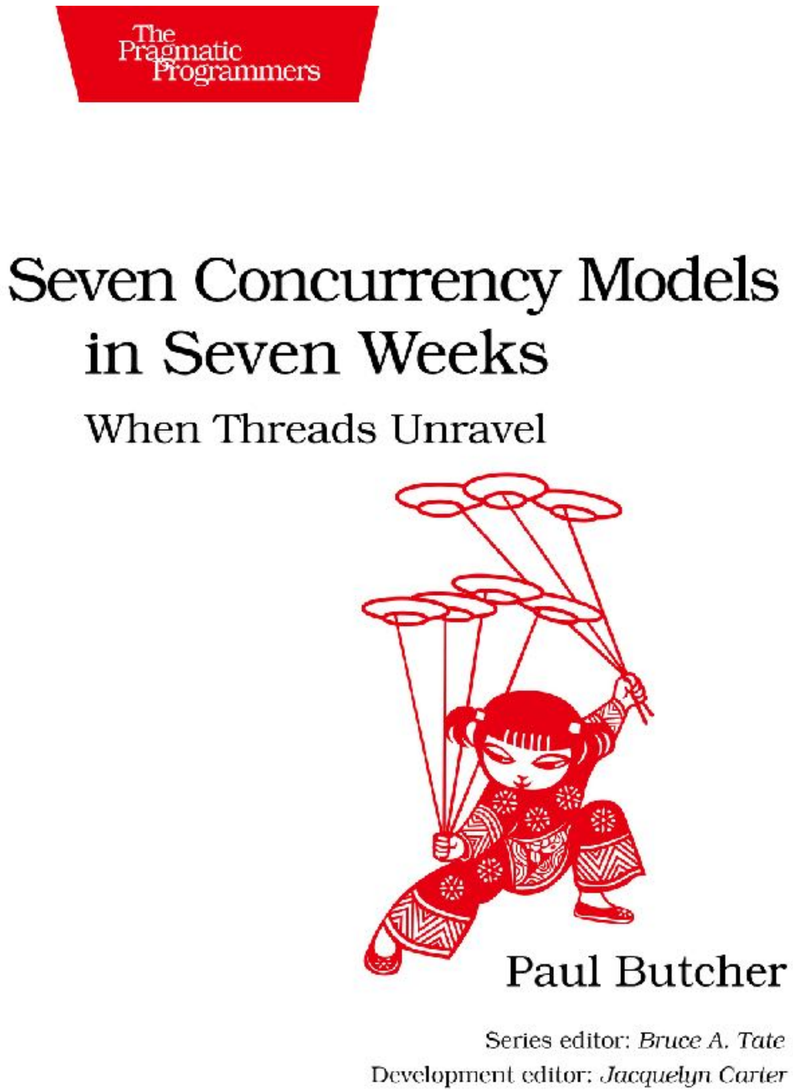
"Software engineering is not about right and wrong but only better and worse." - Ellen Ullman

RETWEETS 203 LIKES 329

9:39 PM - 21 Feb 2017

8 203 329

https://twitter.com/CodeWisdom/status/834125287151501312



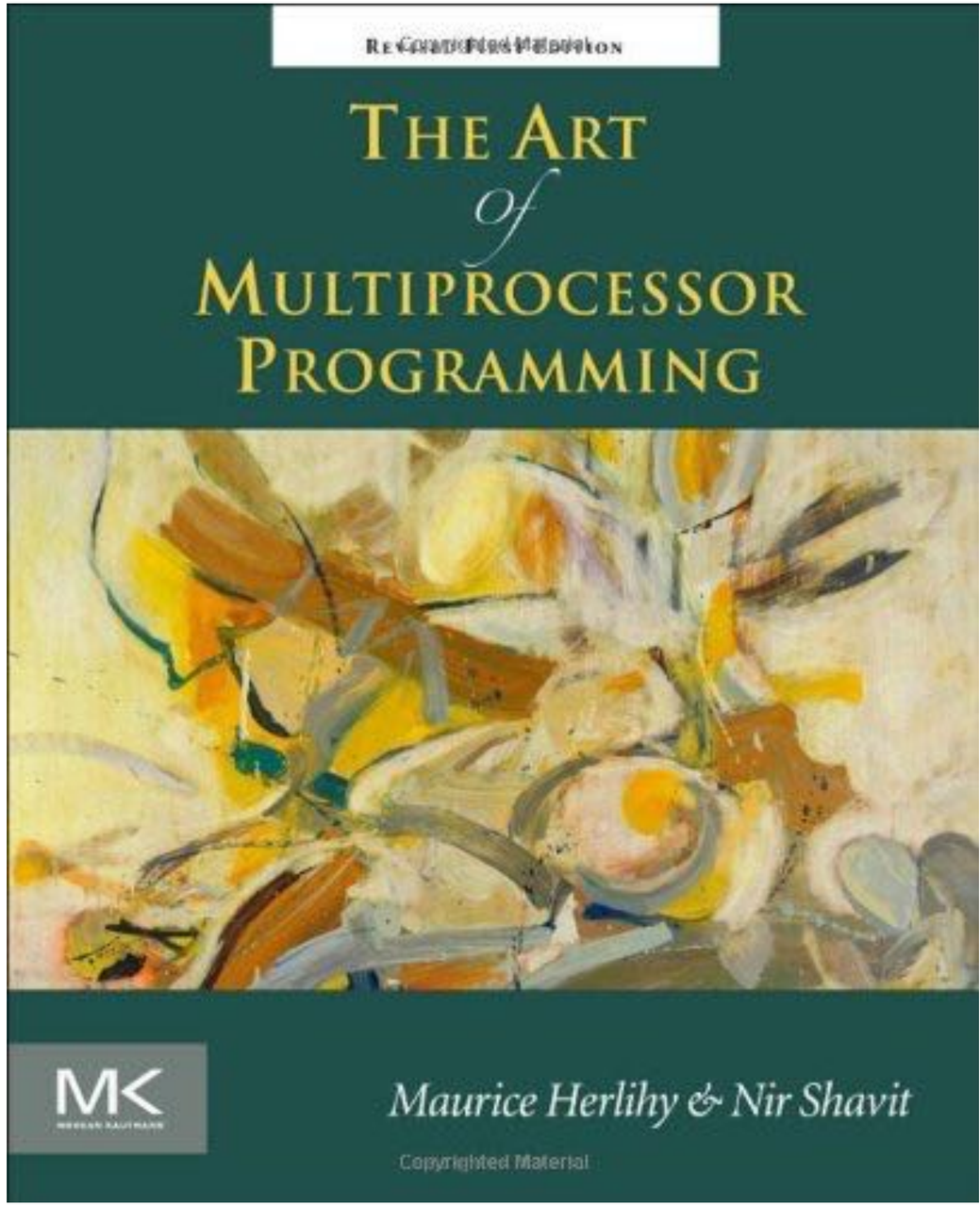
Seven Concurrency Models in Seven Weeks: When Threads Unravel

by Paul Butcher

https://pragprog.com/book/pb7con/seven-concurrency-models-in-seven-weeks

The Art of Multiprocessor Programming

by Maurice Herlihy, Nir Shavit



http://www.amazon.com/The-Multiprocessor-Programming-Revised-Reprint/dp/0123973376

VOXXED DAYS ZÜRICH

23rd of February 2017

Sihlcity Cinema

amazing speakers!

workshops available

290 CHF

-15% discount using
VDZ17_JUG_CH code

https://zurich.voxxeddays.com

Find me and chat with me!


oleg@zeroturnaround.com


@shelajev