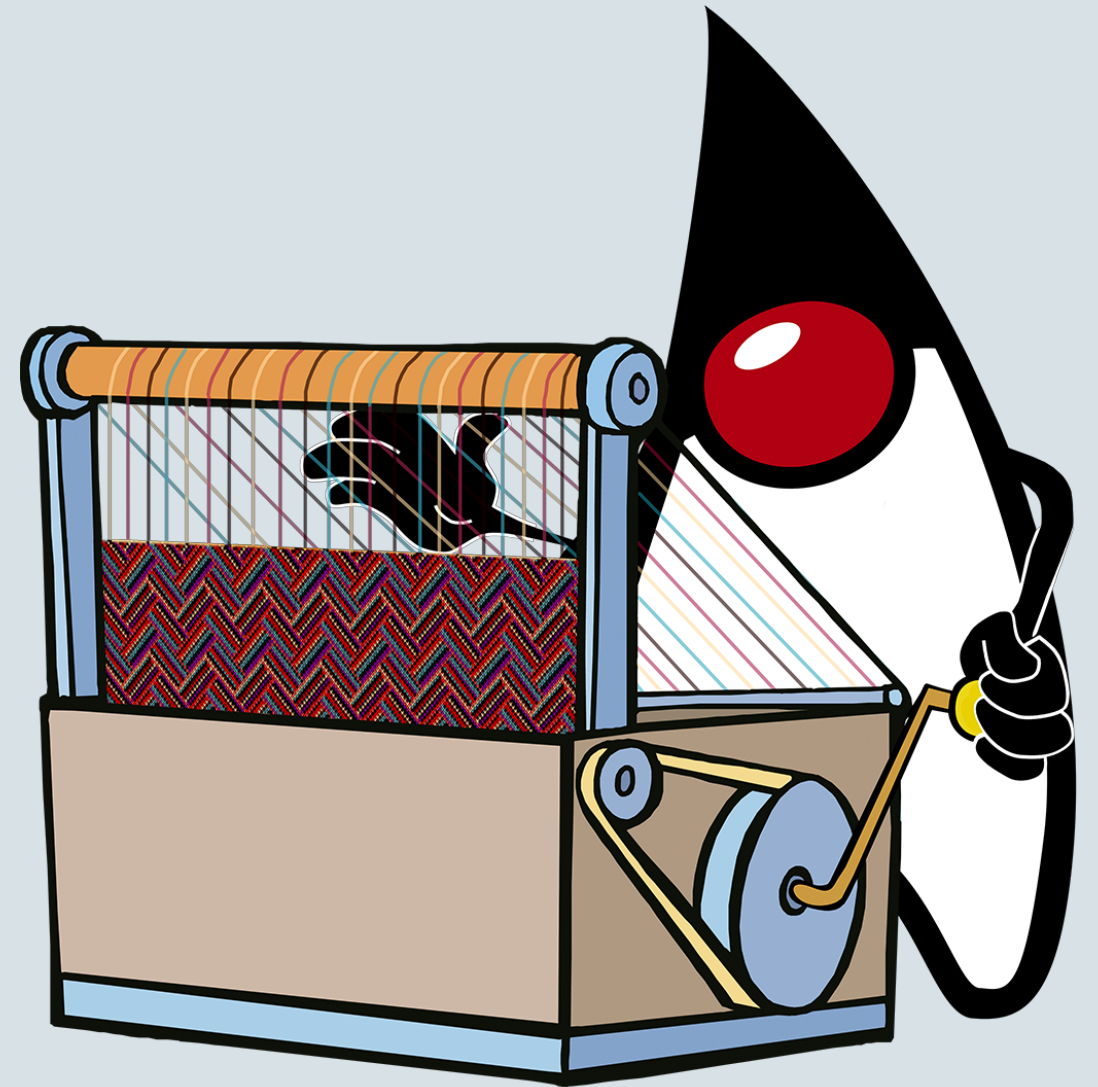


Why Java Is Getting Continuations

Ron Pressler, Oracle
March 5, 2019

OpenJDK™



Project Loom

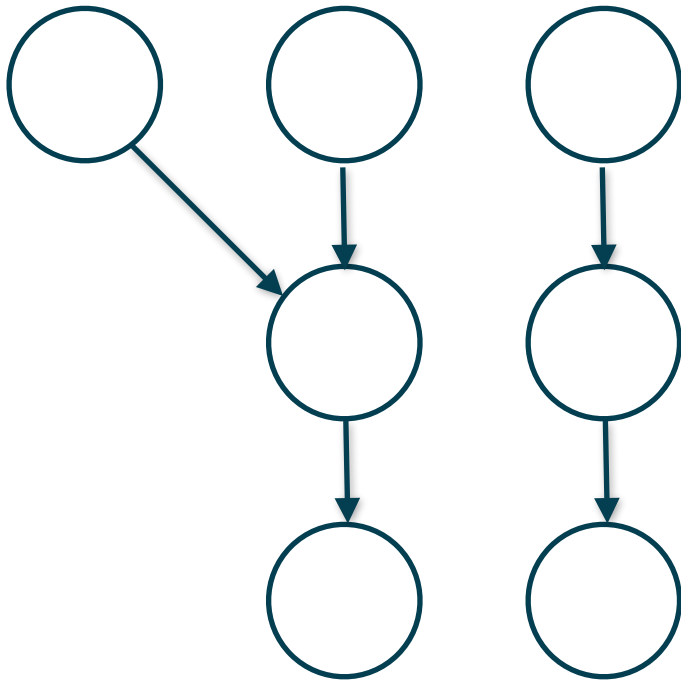


Safe Harbor Statement

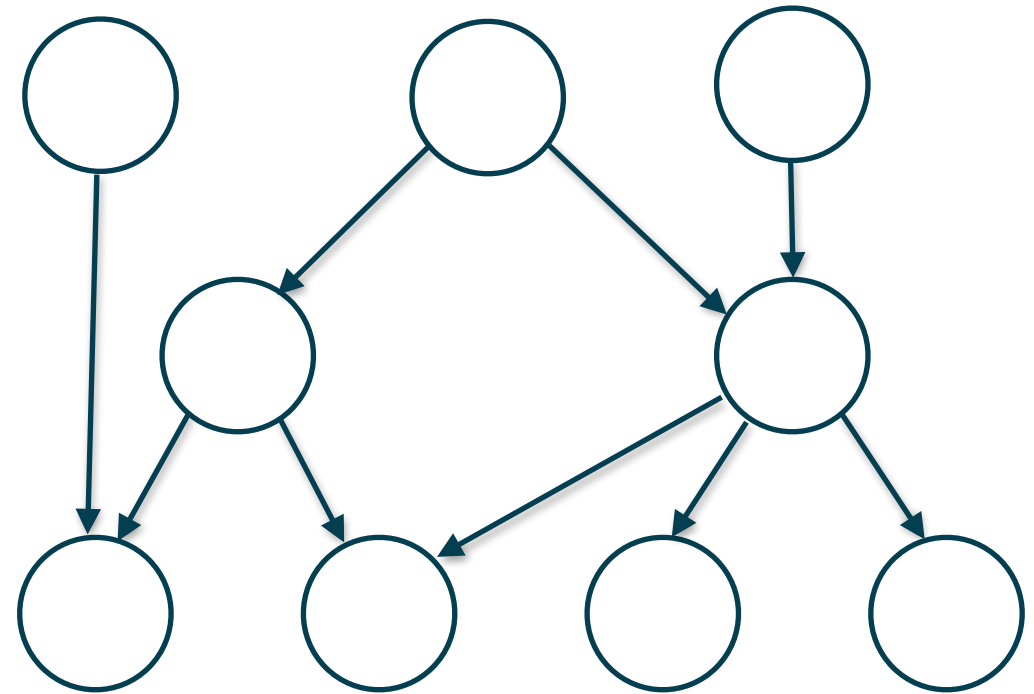
The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Views of Computation

deterministic
sequential



nondeterministic
interactive/concurrent

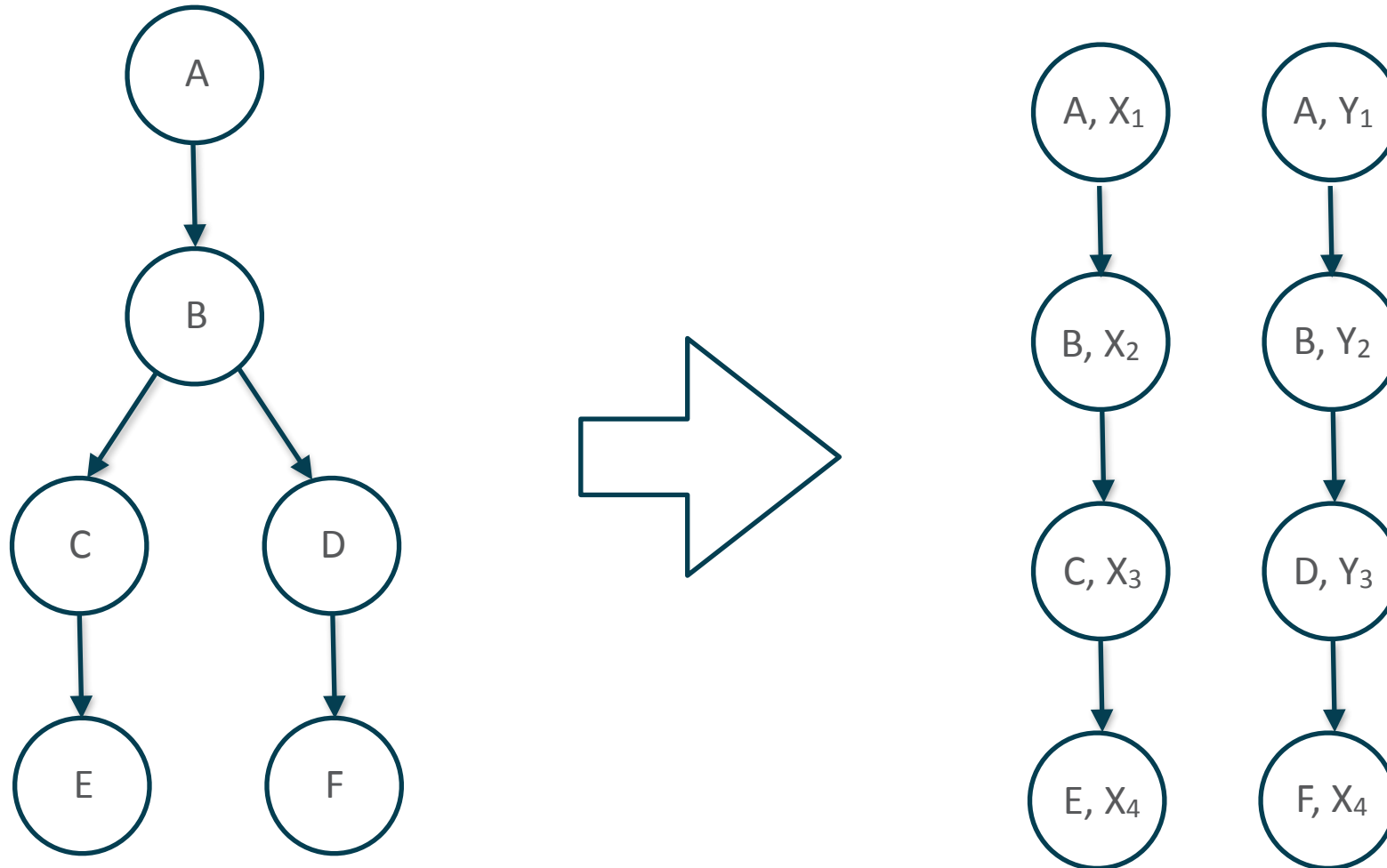


Pure Functional Programming

deterministic
sequential

$[[f \ x]] \cong$ function application

Linear Types: nondeterminism as a function of unknown values



Linear Types: nondeterminism as a function of unknown values

```
getLine  :: World -> ( World, String)
putStrLn :: World -> String -> World

main :: World -> World
main w = let rec
           (w, str) = getLine w
                w = putStrLn w str
         in w
```

Linear Types: nondeterminism as a function of unknown values

```
getLine  :: World -> ( World, String)
putStrLn :: World -> String -> World

main :: World -> World
main w0 = let
            (w1, str) = getLine w0
              w2 = putStrLn w1 str
          in w2
```

Linear Types: nondeterminism as a function of unknown values

```
getLine  :: World -> ( World, String)
putStrLn :: World -> String -> World
```

```
main :: World -> World
main w0 = let
            (w1, str1) = getLine w0
            (_, str2) = getLine w0
            w2 = putStrLn w1 str1
        in w2
```


Linear Types: nondeterminism as a function of unknown values

```
getLine  :: !World -> (!World, String)
putStrLn :: !World -> String -> !World
```

```
main :: !World -> !World
```

```
main w0 = let
```

```
    (w1, str1) = getLine w0
```

```
    (_, str2) = getLine w0
```

```
    w2 = putStrLn w1 str1
```

```
in w2
```

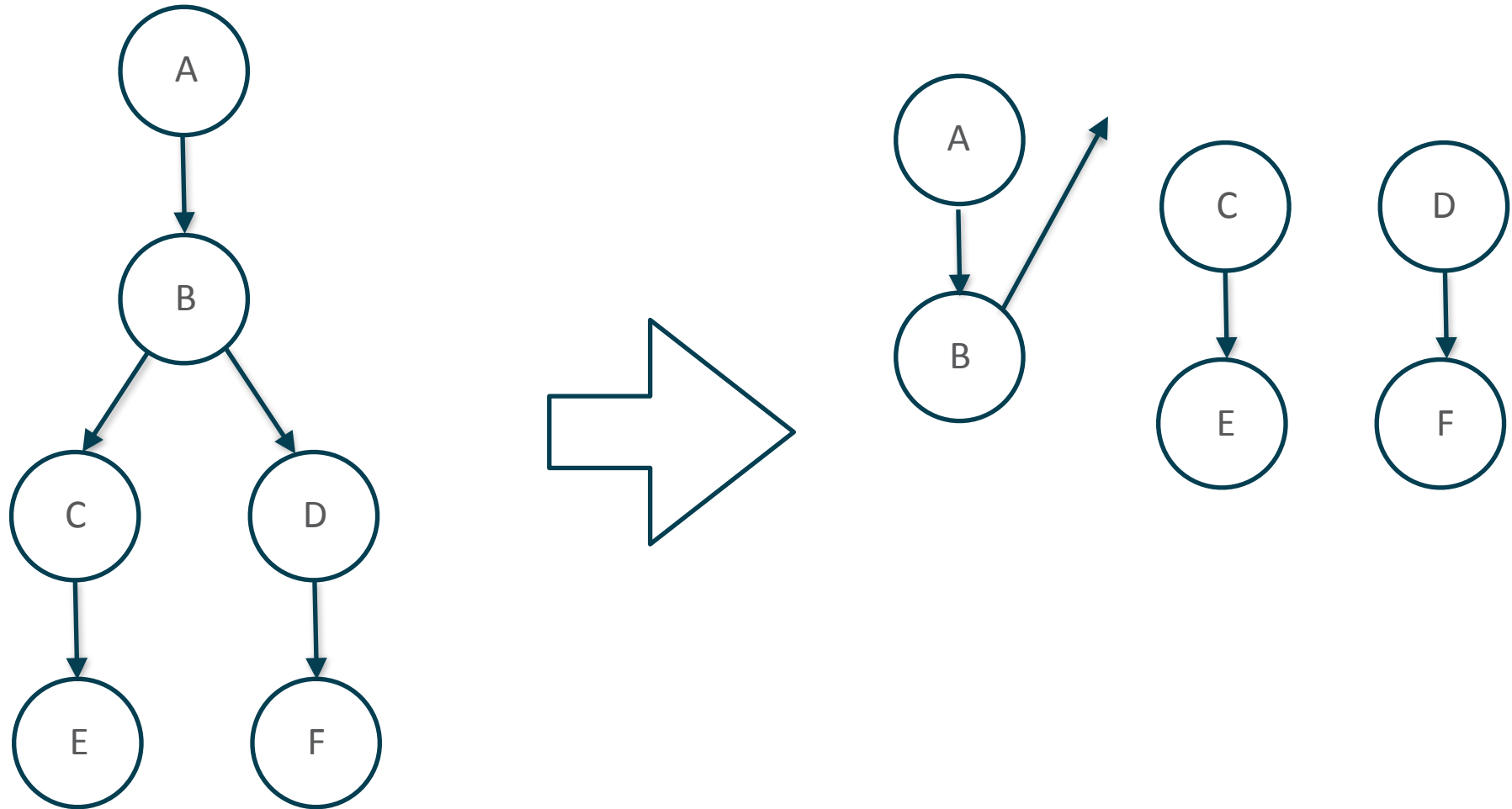
w0 of type !World
has been consumed

Linear Types: nondeterminism as a function of unknown values

```
getLine  :: !World -> (!World, String)
putStrLn :: !World -> String -> !World

main :: !World -> !World
main w = let rec
           (w, str) = getLine w
                w = putStrLn w str
         in w
```

IO Type: Move nondeterminism outside the program



IO Type: Move nondeterminism outside the program

```
getLine    :: IO String  
putStrLn  :: String -> IO ()  
main      :: IO ()
```

```
main = do  
    z <- getLine  
    putStrLn z
```

IO Type: Move nondeterminism outside the program

```
getLine  :: IO String
putStrLn :: String -> IO ()
main     :: IO ()
```

```
main = do
  z <- getLine
  putStrLn z
```

```
main = bindIO getLine (\z -> putStrLn z)
```

IO Type: Move nondeterminism outside the program

```
getLine  :: IO String
putStrLn :: String -> IO ()
main     :: IO ()
```

```
main = do
  z <- getLine
  putStrLn z
```

```
main = bindIO getLine (\z -> putStrLn z)
```

```
returnIO :: IO a -> IO a
bindIO   :: IO a -> (a -> IO b) -> IO b
```

Classical Imperative Programming

nondeterministic
concurrent/interactive

$\llbracket p(x) \rrbracket$ = predicate transformer

```
import static java.io.Console.*;
```

```
var str = readLine();  
printf(str);
```



```
package java.util.concurrent;
```

```
class CompletableFuture<A> {  
  // completedFuture :: A -> CompletableFuture A  
  static <A> CompletableFuture<A> completedFuture(A value);  
  
  // thenCompose :: CompletableFuture A -> (A -> CompletableFuture B) -> CompletableFuture B  
  <B> CompletableFuture<B> thenCompose(Function<A, CompletableFuture<B>> f);  
}
```

Recall:

```
returnIO :: IO a -> IO a
```

```
bindIO   :: IO a -> (a -> IO b) -> IO b
```

```
var str = readLine();  
printf(str);
```

```
class CompletableFuture<A> {  
    static <A> CompletableFuture<A> completedFuture(A value);  
    <B> CompletableFuture<B> thenCompose(Function<A, CompletableFuture<B>> f);  
}
```

```
double calcImportantFinance(double x) {
    try {
        double result;
        result = compute("USD->euro", x);
        result = compute("subtractTax", result * 1.3);
        result = compute("addInterest", result);
        return result;
    } catch (Exception ex) {
        log(ex);
        throw ex;
    }
}
```

```
double compute(String op, double x);
```

```
CompletableFuture<Double> calcImportantFinance (double x) {
    return
        compute("USD->euro", 100.0)
        .thenCompose(result -> compute("subtractTax", result * 1.3))
        .thenCompose(result -> compute("addInterest", result))
        .handle((result, ex) -> {
            if (ex != null) {
                log(ex);
                throw new RuntimeException(ex);
            } else
                return result;
        });
}
```

Reactive Programming: Lessons Learned

Tomasz Nurkiewicz

1. Lost Control Flow

```
double result;  
result = compute("USD->euro", 100.0);  
if (result > 1000)  
    result = compute("subtractTax", result * 1.3);  
while (!sufficient(result))  
    result = compute("addInterest", result);  
return result;
```

2. Lost Context

```
CompletableFuture<Double> calcImportantFinance (double x) {  
    return  
        compute("USD->euro", 100.0)  
        .thenCompose(result -> compute("subtractTax", result * 1.3))  
        .thenCompose(result -> compute("addInterest", result))  
        .handle((result, ex) -> {  
            if (ex != null) {  
                log(ex);  
                throw new RuntimeException(ex);  
            } else  
                return result;  
        });  
}
```


3. Viral

```
CompletableFuture<Double> calcImportantFinance (double x) {  
    return  
        compute("USD->euro", 100.0)  
        .thenCompose(result -> compute("subtractTax", result * 1.3))  
        .thenCompose(result -> compute("addInterest", result))  
        .handle((result, ex) -> {  
            if (ex != null) {  
                log(ex);  
                throw new RuntimeException(ex);  
            } else  
                return result;  
        }  
        ));  
}
```

Async/Await

```
async Task<double> CalcImportantFinance()
{
    try
    {
        double result;
        result = await compute("USD->euro", x);
        result = await compute("subtractTax", result * 1.3);
        result = await compute("addInterest", result);
        return result;
    }
    catch (Exception ex)
    {
        log(ex);
        throw ex;
    }
}
```

Why give up a good (core!) abstraction just because of an inadequate implementation?

```
var str = readLine();  
printf(str);
```

thread/process =

yield control and resume

+

execution scheduling

thread/process =

continuation

+

scheduler

Continuations

```
package java.lang;

public class Continuation implements Runnable {
    public Continuation(ContinuationScope scope, Runnable body);

    public final void run();
    public boolean isDone();
    public static void yield(ContinuationScope scope);

    protected static Continuation currentContinuation(ContinuationScope scope);
}
```

One-Shot Multi-Prompt Delimited Continuations

```
package java.lang;

public class Continuation implements Runnable {
    public Continuation(ContinuationScope scope, Runnable body);

    public final void run();
    public boolean isDone();
    public static void yield(ContinuationScope scope);

    protected static Continuation currentContinuation(ContinuationScope scope);
}
```

```
Continuation cont = new Continuation(SCOPE, () -> {
    while (true) {
        // maybe in some deep method:
        System.out.println("before");
        Continuation.yield(SCOPE);
        System.out.println("after");
    }
});
```

```
while (!cont.isDone()) {
    cont.run();
}
```



```
package java.lang;

public class Continuation implements Runnable {
    public Continuation(ContinuationScope scope, Runnable body);

    public final void run();
    public boolean isDone();
    public static void yield(ContinuationScope scope);

    protected static Continuation currentContinuation(ContinuationScope scope);

    public PreemptStatus tryPreempt(Thread thread);
}
```

```
package java.lang;

public class Continuation implements Runnable {
    public Continuation(ContinuationScope scope, Runnable body);

    public final void run();
    public boolean isDone();
    public static void yield(ContinuationScope scope);

    protected static Continuation currentContinuation(ContinuationScope scope);

    public PreemptStatus tryPreempt(Thread thread);

    public StackWalker stackWalker();
    public StackTraceElement[] getStackTrace();
}
```

One-Shot Multi-Prompt Delimited Continuations

```
package java.lang;

public class Continuation implements Runnable {
    public Continuation(ContinuationScope scope, Runnable body);

    public final void run();
    public boolean isDone();
    public static void yield(ContinuationScope scope);
}
```

Reentrant Multi-Prompt Delimited Continuations

```
package java.lang;

public class Continuation implements Runnable, Cloneable {
    public Continuation(ContinuationScope scope, Runnable body);

    public final void run();
    public boolean isDone();
    public static void yield(ContinuationScope scope);

    public Continuation clone();
}
```

Reentrant Multi-Prompt Delimited Continuations

```
class MultiShotContinuation {  
    private Continuation cont;  
  
    public MultiShotContinuation(ContinuationScope scope, Runnable task) {  
        this.cont = new Continuation(scope, task);  
    }  
  
    public MultiShotContinuation run() {  
        var copy = cont.clone();  
        copy.run();  
        return copy;  
    }  
}
```

Reentrant Multi-Prompt Serializable Delimited Continuations

```
package java.lang;

public class Continuation implements Runnable, Cloneable, Serializable {
    public Continuation(ContinuationScope scope, Runnable body);

    public final void run();
    public boolean isDone();
    public static void yield(ContinuationScope scope);

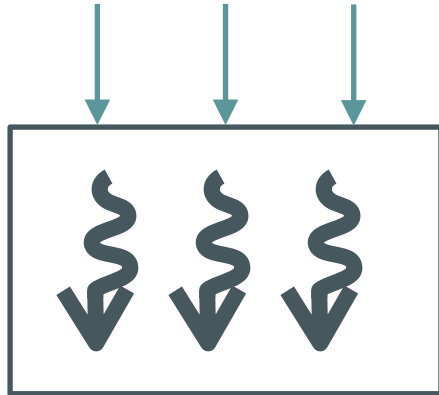
    public Continuation clone();
}
```

fiber = continuation + scheduler

Why?

Connections

App



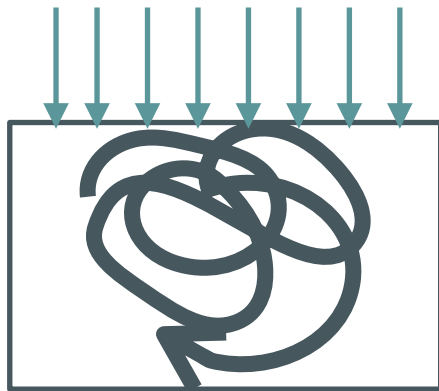
simple
less scalable

SYNC

OR

Connections

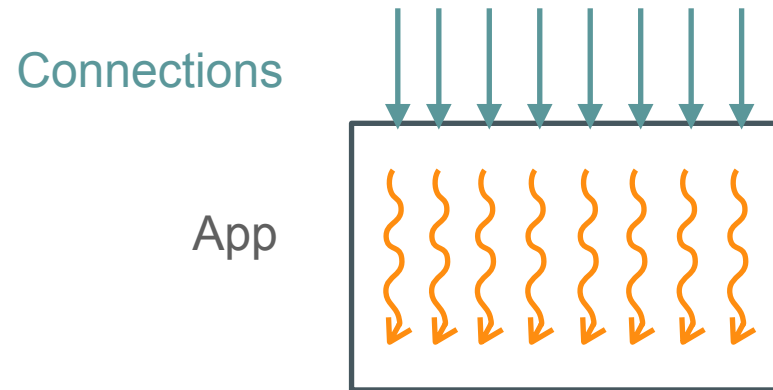
App



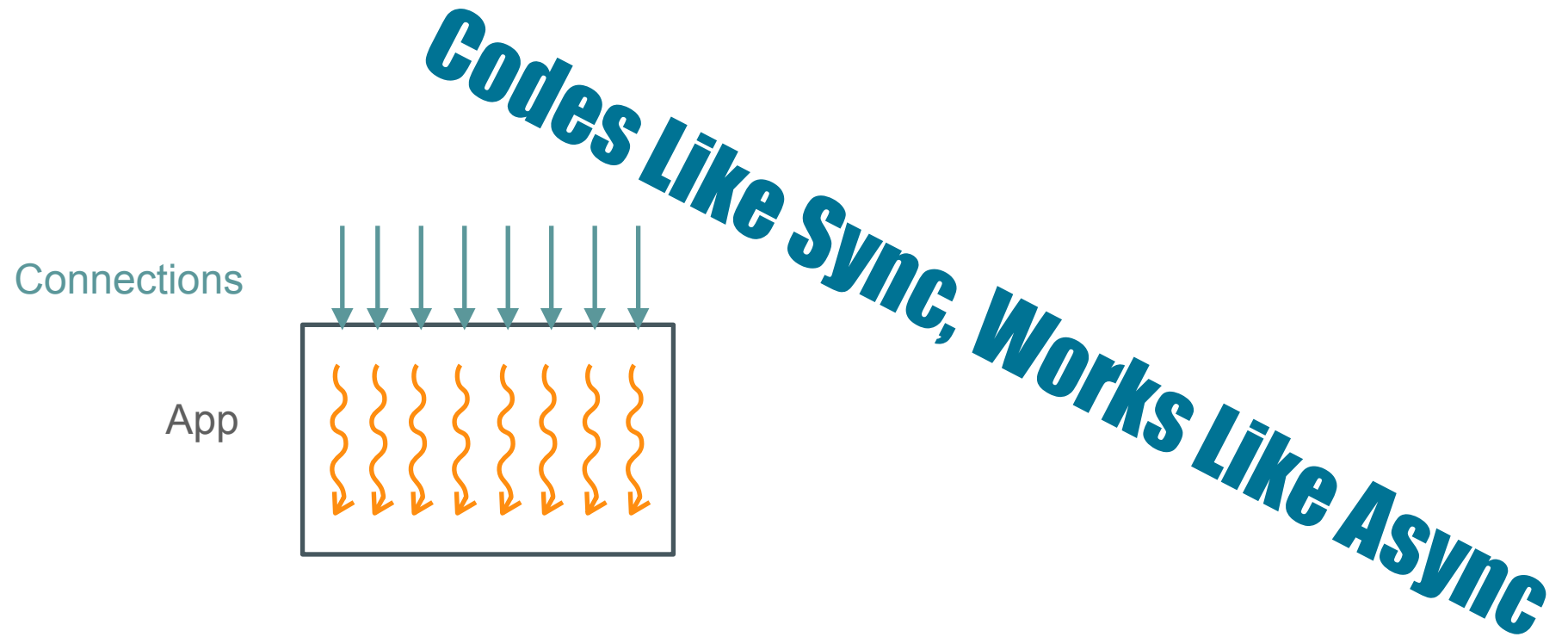
scalable,
complex,
non-interoperable,
hard to debug/profile

ASYNC

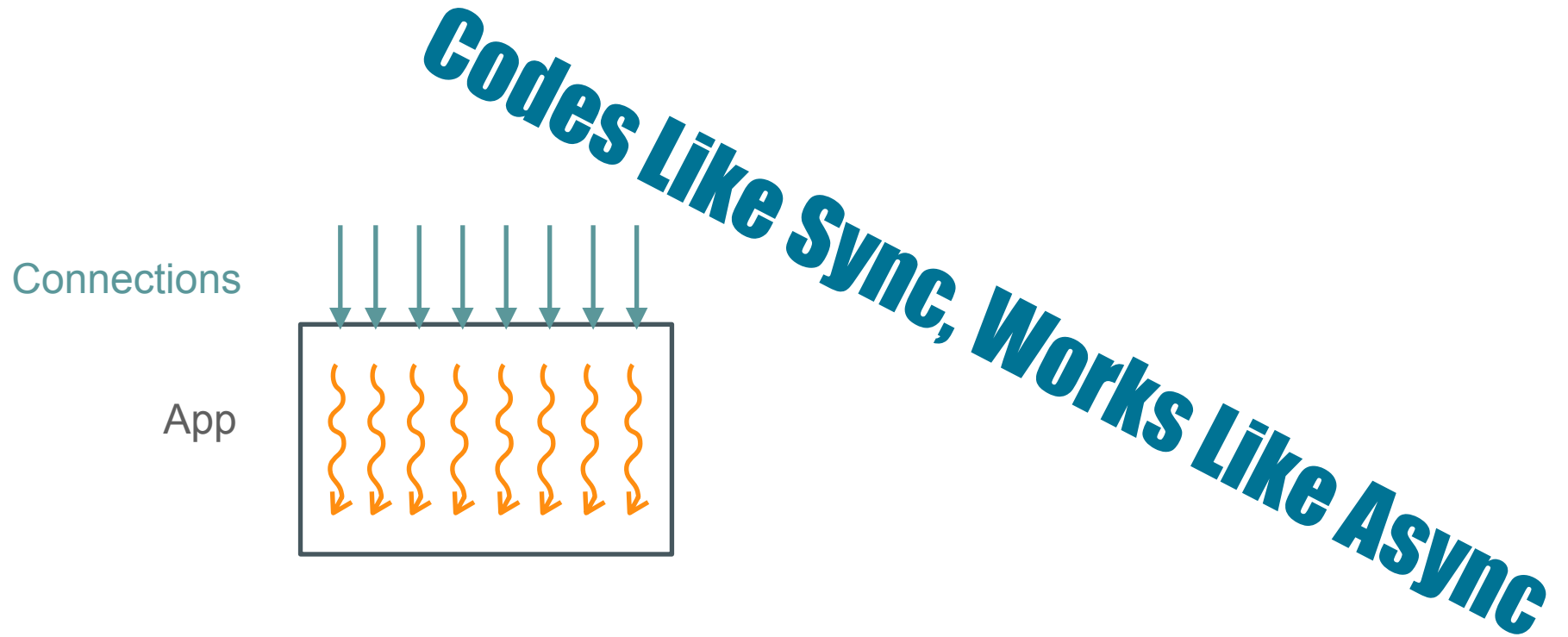
Why?



Why?



Why?



CONCURRENCY MADE SIMPLE

IO, java.util.concurrent — now fiber-blocking

Class Fiber<V>

java.lang.Object
java.lang.Fiber<V>

Type Parameters:

V - the task result type

```
public class Fiber<V>  
extends Object
```

A *user mode* thread to execute a task that is scheduled by the Java virtual machine.

A *Fiber* is created and scheduled in a *fiber scope* to execute a task by invoking `run()` or `runAsync()`. The `awaitTermination` method can be used to wait for a fiber to terminate. The `cancel` method terminates with an exception. The `toFuture` method can be used to obtain a `CompletableFuture`.

Unless otherwise noted, passing a `null` argument will cause a `NullPointerException`.

Method Summary

All Methods	Static Methods	Instance Methods	Concrete Methods
Modifier and Type	Method		
void			<code>awaitTermination()</code>
boolean			<code>awaitTermination(Duration duration)</code>
boolean			<code>cancel()</code>
static boolean			<code>cancelled()</code>
static Optional<Fiber<?>>			<code>current()</code>
boolean			<code>isAlive()</code>
boolean			<code>isCancelled()</code>
V			<code>join()</code>
V			<code>join(Duration duration)</code>
static Fiber<?>			<code>schedule(Runnable task)</code>

```
package java.util.concurrent.locks;  
public class LockSupport {
```

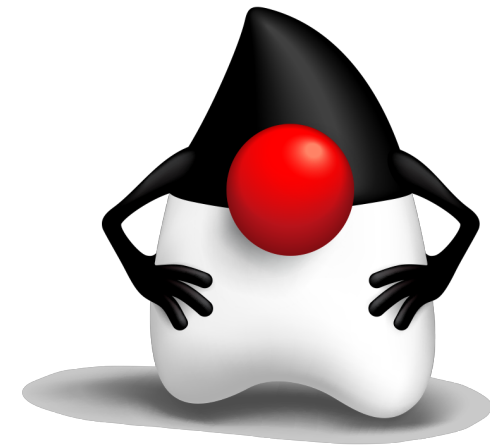
```
    public static void park(...) {  
        var strand = Strands.currentStrand();  
        if (strand instanceof Fiber)  
            Continuation.yield(FIBER_SCOPE);  
        else  
            Unsafe.park(false, 0L);  
    }
```

```
}
```

```
    public static void unpark(Object strand) {  
        if (strand instanceof Fiber) {  
            var f = ((Fiber<?>)strand);  
            f.scheduler.submit(f.continuation);  
        } else if (strand instanceof Thread) {  
            Unsafe.unpark(thread);  
        } else  
            throw new IllegalArgumentException();  
    }
```

Async/Await

```
class Async<T> extends CompletableFuture<T> {  
    public static <T> T await(Async<T> async) throws InterruptedException, ExecutionException {  
        return async.get();  
    }  
}
```



>2_{KB} metadata

1_{MB} stack

200-300_B metadata

Pay-as-you-go stack

1-10_{μs}

???_{ns}

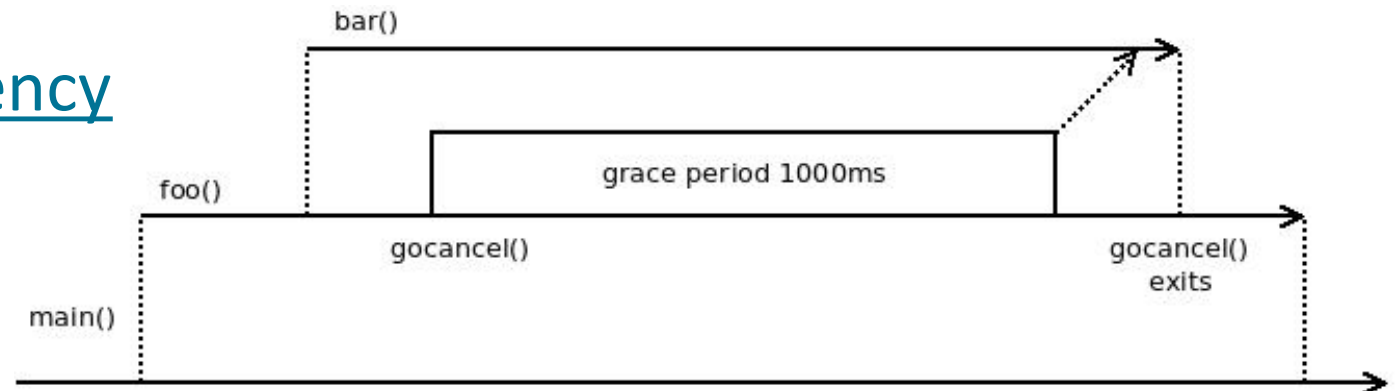
“Rethink threads”

–The Java Architects

Structured Concurrency

Martin Sústrik (libdill, C)

- [Structured Concurrency](#)
- [Update on Structured Concurrency](#)



© Martin Sústrik

Nathaniel J. Smith (Trio, Python)

- [Timeouts and cancellation for humans](#)
- [Notes on structured concurrency, or: Go statement considered harmful](#)

Structured Concurrency

```
try (var scope = FiberScope.cancellable()) {  
    Fiber<?> fiber = scope.schedule(task);  
}
```

- Cannot exit scope until all fibers scheduled in the scope have terminated
- Fiber scopes can be nested
- A `FiberScope` has a termination queue that collect the results of scope's fibers
- Canceling a fiber of a cancellable scope will cancel all fibers that it has scheduled in the scope. With nesting, a tree of fibers may be cancelled.
- Cancelling a fiber parked in a blocking I/O operation will cause it to unpark and check for cancellation

Structured Concurrency

Return the result of the first task that completes, cancelling and waiting for any outstanding fibers to terminate before returning.

```
<V> V anyOf(Callable<? extends V>[] tasks) throws Throwable {
    try (var scope = FiberScope.cancellable()) {
        var queue = new FiberScope.TerminationQueue<V>();
        Arrays.stream(tasks).forEach(task -> scope.schedule(task, queue));

        try {
            return queue.take().join();
        } catch (CompletionException e) {
            throw e.getCause();
        } finally {
            scope.fibers().forEach(Fiber::cancel); // cancel remaining fibers
        }
    }
}
```

Structured Concurrency

Same, with a deadline. If the deadline expires then all fibers scheduled in the scope are cancelled.

```
<V> V anyOf(Callable<? extends V>[] tasks, Instant deadline) throws Throwable {  
    try (var scope = FiberScope.withDeadline(deadline)) {  
        var queue = new FiberScope.TerminationQueue<V>();  
        Arrays.stream(tasks).forEach(task -> scope.schedule(task, queue));  
  
        try {  
            return queue.take().join();  
        } catch (CompletionException e) {  
            throw e.getCause();  
        } finally {  
            scope.fibers().forEach(Fiber::cancel); // cancel remaining fibers  
        }  
    }  
}
```

```

class Generator<T> implements Iterable<T> {
    private static final ContinuationScope GENERATOR = new ContinuationScope();
    private static class GenContinuation<T> extends Continuation {
        public GenContinuation() { super(GENERATOR); }
        private T value;
    }

    public static void yield(T value) {
        ((GenContinuation)currentContinuation(GENERATOR)).val = value;
        Continuation.yield(GENERATOR);
    }

    private final GenContinuation<T> cont;
    public Generator(Runnable body) { cont = new GenContinuation<T>(body); }

    public Iterator<T> iterator() {
        return new Iterator<T>() {
            public T next() { cont.run(); return cont.val; }
            public boolean hasNext() { return !cont.isDone(); }
        }
    }
}

```

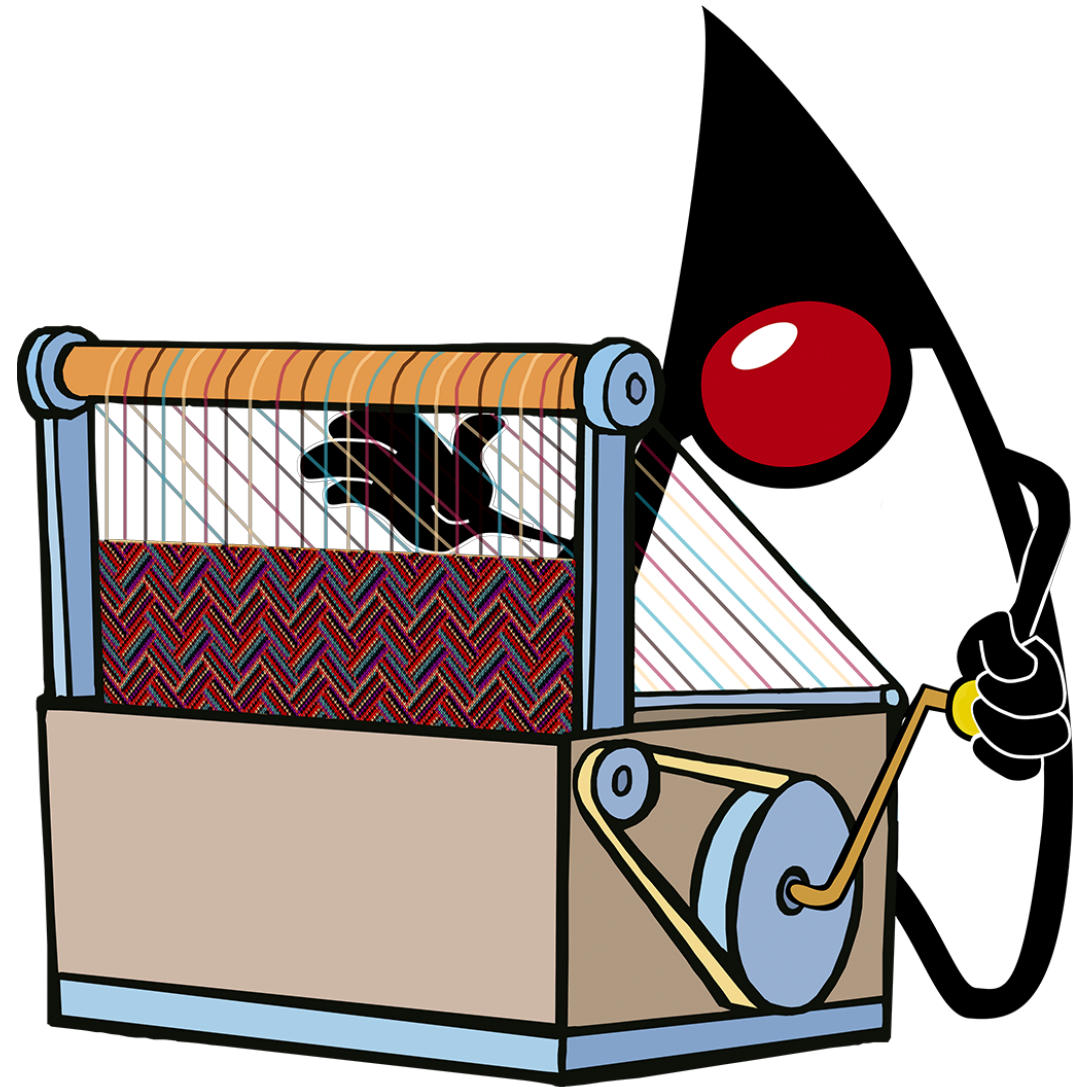
```
var fibonacci = new Generator<Integer>(() -> {
    Generator.yield(0);
    int a = 0;
    int b = 1;
    while(true) {
        Generator.yield(b);
        var sum = a + b;
        a = b;
        b = sum;
    }
});

for (var num : fibonacci) {
    System.out.println(num);
    if (num > 10_000) break;
}
```

```
var greetedPrimes = new Generator<String>(() -> {
    for (int n = 0; ; n++)
        if (isPrime(n)) {
            var greeting = Console.readLine();
            Generator.yield(greeting + ": " + n);
        }
});
```

```
Fiber.schedule(()-> {
    for (var x : greetedPrimes)
        System.out.println(x);
})
```

[https://
wiki.openjdk.java.net
/display/loom/](https://wiki.openjdk.java.net/display/loom/)



Project Loom



Java™
ORACLE®

Q&A

ORACLE®