Threading the needle

Marcel Ton

clearmontcoding.com

@ClearmontCoding

Hi, I’m Marcel!

- Freelance developer

clearmontcoding.com

@ClearmontCoding

Hi, I'm Marcel!

- Freelance developer
- Father and husband

clearmontcoding.com

@ClearmontCoding

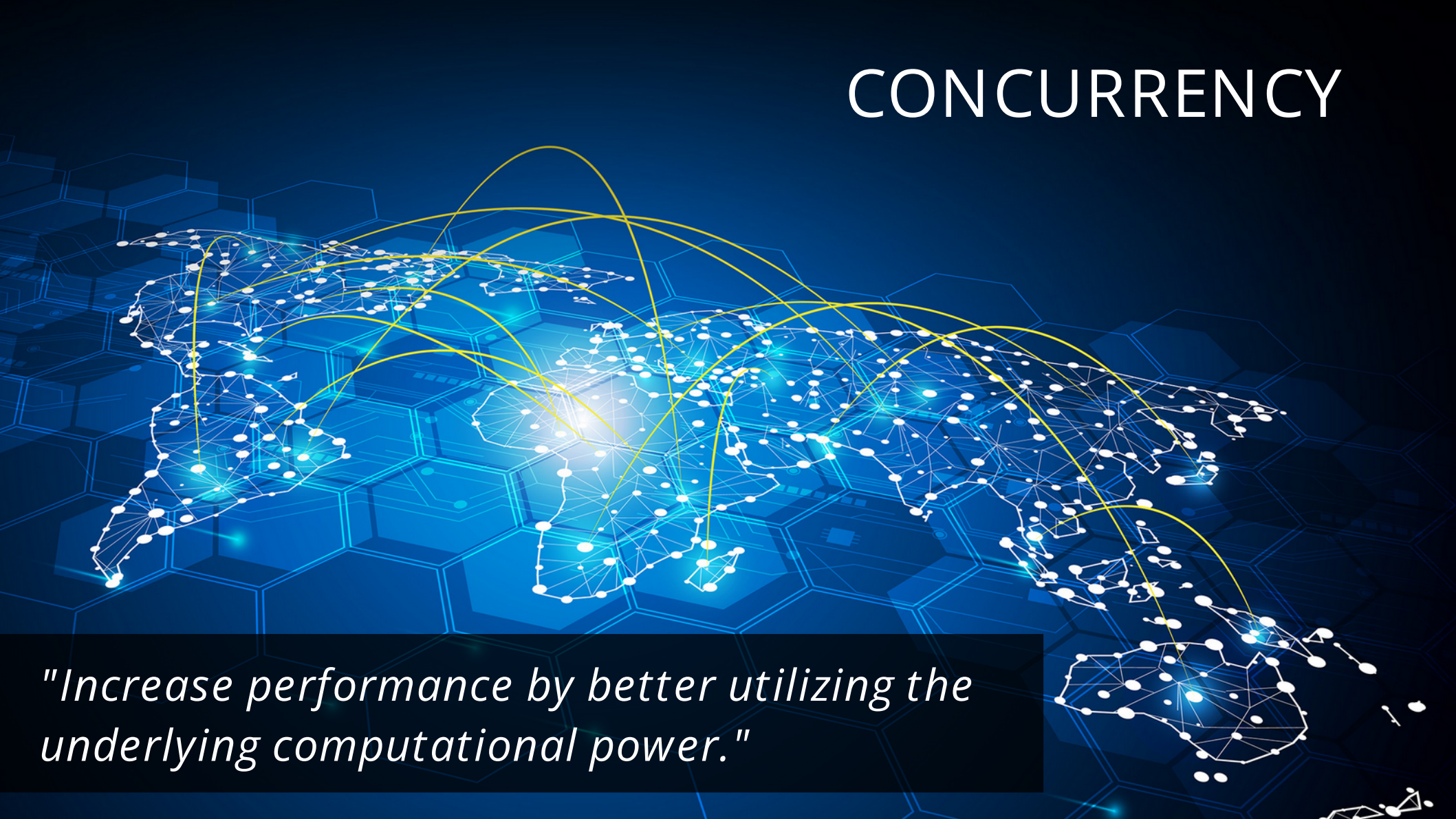
Hi, I'm Marcel!

- Freelance developer
- Father and husband
- First time presenter


GOAL

CONCURRENCY

CONCURRENCY



"Increase performance by better utilizing the underlying computational power."

CONCURRENCY
ASYNCHRONOUS

CONCURRENCY
ASYNCHRONOUS

"A process that runs independently from the main process."

Java 1 (1995)

`Thread` and **`Runnable`**

Java 5 (2004)

`Callable` and **`ExecutorService`**

Java 7 (2011)

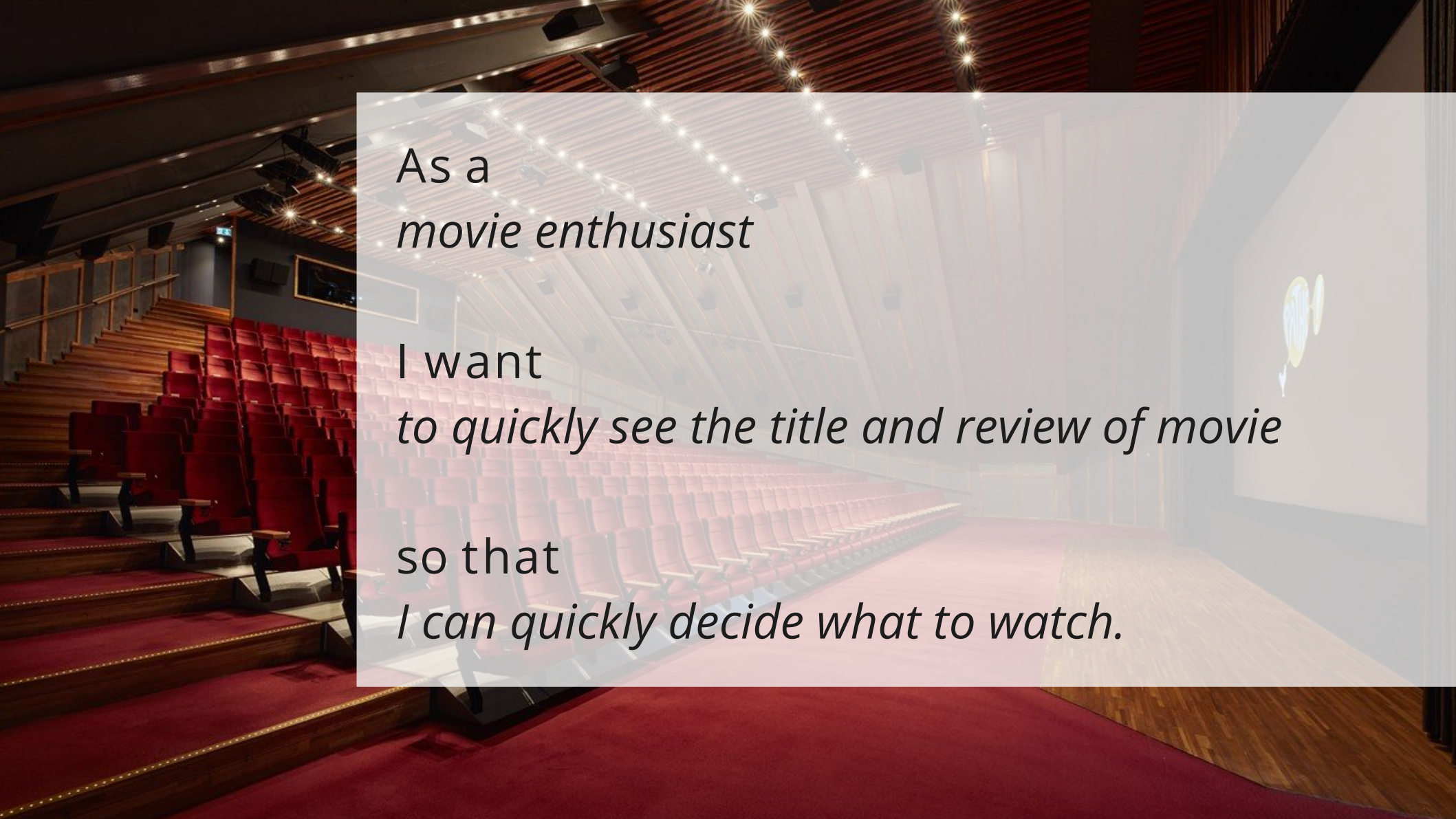
ForkJoinPool

Java 8 (2014)

ParallelStream and
CompleteableFuture

Java 9 (2017)

Flow API



As a
movie enthusiast

I want
to quickly see the title and review of movie

so that
I can quickly decide what to watch.

CompletableFuture

```
public static void main(String[] args) {
    getMovieIds()
        .thenApply(movieIds ->
            movieIds.parallelStream()
                .map(movieId -> getTitle(movieId)
                    .thenCombineAsync(getReviews(movieId), Movie::new)
                    .thenAccept(System.out::println))
                .toList())
        .handleAsync((result, ex) -> Objects.nonNull(ex) ? ex : result)
        .join();
}

private static CompletableFuture<List<Long>> getMovieIds() {...}

private static CompletableFuture<Title> getTitle(Long movieId) {...}

private static CompletableFuture<List<Review>> getReviews(Long movieId) {...}
```

```java
public static void main(String[] args) {
    getMovieIds()
        .thenApply(movieIds ->
            movieIds.parallelStream()
                .map(movieId -> getTitle(movieId)
                    .thenCombineAsync(getReviews(movieId), Movie::new)
                    .thenAccept(System.out::println))
                .toList())
        .handleAsync((result, ex) -> Objects.nonNull(ex) ? ex : result)
        .join();
}

private static CompletableFuture<List<Long>> getMovieIds() {...}

private static CompletableFuture<Title> getTitle(Long movieId) {...}

private static CompletableFuture<List<Review>> getReviews(Long movieId) {...}
```

```java
public static void main(String[] args) {
    getMovieIds()
        .thenApply(movieIds ->
            movieIds.parallelStream()
                .map(movieId -> getTitle(movieId)
                    .thenCombineAsync(getReviews(movieId), Movie::new)
                    .thenAccept(System.out::println))
                .toList())
        .handleAsync((result, ex) -> Objects.nonNull(ex) ? ex : result)
        .join();
}

private static CompletableFuture<List<Long>> getMovieIds() {...}

private static CompletableFuture<Title> getTitle(Long movieId) {...}

private static CompletableFuture<List<Review>> getReviews(Long movieId) {...}
```

Powerful, but hard to write

```java
public static void main(String[] args) {
    getMovieIds()
        .thenApply(movieIds ->
            movieIds.parallelStream()
                .map(movieId -> getTitle(movieId)
                    .thenCombineAsync(getReviews(movieId), Movie::new)
                .toList())
        .handleAsync((result, ex) -> Objects.nonNull(ex) ? ex : result)
        .join();
}

private static CompletableFuture<List<Long>> getMovieIds() {...}

private static CompletableFuture<Title> getTitle(Long movieId) {...}

private static CompletableFuture<List<Review>> getReviews(Long movieId) {...}
```

- Hard to debug and test
- Memory is held throughout chain

```
public static void main(String[] args) {
    getMovieIds()
        .thenApply(movieIds ->
            movieIds.parallelStream()
                .map(movieId -> getTitle(movieId)
                    .thenCombineAsync(getReviews(movieId), Movie::new)
                    .thenAccept(System.out::println))
                .toList())
        .handleAsync((result, ex) -> Objects.nonNull(ex) ? ex : result)
        .join();
}

private static CompletableFuture<List<Long>> getMovieIds() {...}

private static CompletableFuture<String> getTitle(Long movieId) {...}

private static CompletableFuture<List<Review>> getReviews(Long movieId) {...}
```

Pervasive “leaky abstraction”

Project Reactor

```
public static void main(String[] args) {
    getMovieIds()
        .flatMap(id -> Mono.zip(getTitle(id), getReviews(id)))
        .map(tuple -> new Movie(tuple.getT1(), tuple.getT2()))
        .collect(Collectors.toList())
        .doOnError(CustomException::new)
        .subscribe(System.out::println);
}

private static Flux<Long> getMovieIds() {...}

private static Mono<Title> getTitle(Long movieId) {...}

private static Mono<List<Review>> getReviews(Long movieId) {...}
```

```java
public static void main(String[] args) {
    getMovieIds()
        .flatMap(id -> Mono.zip(getTitle(id), getReviews(id)))
        .map(tuple -> new Movie(tuple.getT1(), tuple.getT2()))
        .collect(Collectors.toList())
        .doOnError(CustomException::new)
        .subscribe(System.out::println);
}

private static Flux<Long> getMovieIds() {...}

private static Mono<Title> getTitle(Long movieId) {...}

private static Mono<List<Review>> getReviews(Long movieId) {...}
```

```java
public static void main(String[] args) {
    getMovieIds()
        .flatMap(id -> Mono.zip(getTitle(id), getReviews(id)))
        .map(tuple -> new Movie(tuple.getT1(), tuple.getT2()))
        .collect(Collectors.toList())
        .doOnError(CustomException::new)
        .subscribe(System.out::println);
}

private static Flux<Long> getMovieIds() {...}

private static Mono<Title> getTitle(Long movieId) {...}

private static Mono<List<Review>> getReviews(Long movieId) {...}
```

The API greatly improves

```
public static void main(String[] args) {
    getMovieIds()
        .flatMap(id -> Mono.zip(getTitle(id), getReviews(id)))
        .map(t -> new Movie(t.getT1(), t.getT2()))
        .collect(Collectors.toList())
        .doOnError(CustomException::new)
        .subscribe(System.out::println);
}

private static Flux<Long> getMovieIds() {...}

private static Mono<Title> getTitle(Long movieId) {...}

private static Mono<List<Review>> getReviews(Long movieId) {...}
```

- Same debug, test and memory issues
- Framework / library overhead costs

```
public static void main(String[] args) {
    getMovieIds()
        .flatMap(id -> Mono.zip(getTitle(id), getReviews(id)))
        .map(tuple -> new Movie(tuple.getT1(), tuple.getT2()))
        .collect(Collectors.toList())
        .doOnError(CustomException::new)
        .subscribe(System.out::println);
}

private static Flux<Long> getMovieIds() {...}

private static Mono<Title> getTitle(Long movieId) {...}

private static Mono<List<Review>> getReviews(Long movieId) {...}
```

Call the plumber

INTRODUCING PROJECT LOOM


INTRODUCING PROJECT LOOM
Creating highly scalable applications without adding complexity

WITH VIRTUAL THREADS


WITH VIRTUAL THREADS
- Java entity with new abstractions


WITH VIRTUAL THREADS
- Java entity with new abstractions
- Write blocking, get non-blocking


WITH VIRTUAL THREADS
- Java entity with new abstractions
- Write blocking, get non-blocking
- Runs and resumes on any thread


WITH VIRTUAL THREADS
- Java entity with new abstractions
- Write blocking, get non-blocking
- Runs and resumes on any thread
- Creating and destroying is cheap


WITH VIRTUAL THREADS
- Java entity with new abstractions
- Write blocking, get non-blocking
- Runs and resumes on any thread
- Creating and destroying is cheap
- Best used in (N)I/O scenarios

```
var platformThreads = IntStream.range(0, 100_000)
    .mapToObj(index ->
        Thread.ofPlatform()
            .start(() -> {
                try {
                    Thread.sleep( millis: 2_000);
                } catch (InterruptedException e) {...}
            }))
    .toList();

for (Thread thread : platformThreads) {
    thread.join();
}
```

```
var platformThreads = IntStream.range(0, 100_000)
    .mapToObj(index ->
        Thread.ofPlatform()
            .start(() -> {
                try {
                    Thread.sleep( millis: 2_000);
                } catch (InterruptedException e) {...}
            }))
    .toList();

for (Thread thread : platformThreads) {
    thread.join();
}
```

30 seconds

```
var virtualThreads = IntStream.range(0, 1_000_000)
    .mapToObj(index ->
        Thread.ofVirtual()
            .start(() -> {
                try {
                    Thread.sleep( millis: 2_000);
                } catch (InterruptedException e) {...}
            }))
    .toList();

for (Thread thread : virtualThreads) {
    thread.join();
}
```

```
var virtualThreads = IntStream.range(0, 1_000_000)
    .mapToObj(index ->
        Thread.ofVirtual()
              .start(() -> {
                  try {
                      Thread.sleep( millis: 2_000);
                  } catch (InterruptedException e) {...}
              }))
    .toList();

for (Thread thread : virtualThreads) {
    thread.join();
}
```

6 seconds

ExecutorService

```
public static void main(String[] args) {
    try (var executor = Executors.newVirtualThreadPerTaskExecutor()) {
        getMovieIds()
            .parallelStream()
            .map(movieId -> {
                var titleFuture = executor.submit(() -> getTitle(movieId));
                var reviewFuture = executor.submit(() -> getReviews(movieId));
                try {
                    return new Movie(titleFuture.get(), reviewFuture.get());
                } catch (InterruptedException | ExecutionException ex) {...}
            })
            .forEach(System.out::println);
    }
}

private static List<Long> getMovieIds() {...}

private static Title getTitle(Long movieId) {...}

private static List<Review> getReviews(Long movieId) {...}
```

```java
public static void main(String[] args) {
    try (var executor = Executors.newVirtualThreadPerTaskExecutor()) {
        getMovieIds()
            .parallelStream()
            .map(movieId -> {
                var titleFuture = executor.submit(() -> getTitle(movieId));
                var reviewFuture = executor.submit(() -> getReviews(movieId));
                try {
                    return new Movie(titleFuture.get(), reviewFuture.get());
                } catch (InterruptedException | ExecutionException ex) {...}
            })
            .forEach(System.out::println);
    }
}

private static List<Long> getMovieIds() {...}

private static Title getTitle(Long movieId) {...}

private static List<Review> getReviews(Long movieId) {...}
```

```java
public static void main(String[] args) {
    try (var executor = Executors.newVirtualThreadPerTaskExecutor()) {
        getMovieIds()
            .parallelStream()
            .map(movieId -> {
                var titleFuture = executor.submit(() -> getTitle(movieId));
                var reviewFuture = executor.submit(() -> getReviews(movieId));
                try {
                    return new Movie(titleFuture.get(), reviewFuture.get());
                } catch (InterruptedException | ExecutionException ex) {...}
            })
            .forEach(System.out::println);
    }
}

private static List<Long> getMovieIds() {...}

private static Title getTitle(Long movieId) {...}

private static List<Review> getReviews(Long movieId) {...}
```

AutoCloseable, no sizing / lifecyle

```java
public static void main(String[] args) {
    try (var executor = Executors.newVirtualThreadPerTaskExecutor()) {
        getMovieIds()
            .parallelStream()
            .map(movieId -> {
                var titleFuture = executor.submit(() -> getTitle(movieId));
                var reviewFuture = executor.submit(() -> getReviews(movieId));
                try {
                    return new Movie(titleFuture.get(), reviewFuture.get());
                } catch (InterruptedException | ExecutionException ex) {...}
            })
            .forEach(System.out::println);
    }
}

private static List<Long> getMovieIds() {...}

private static Title getTitle(Long movieId) {...}

private static List<Review> getReviews(Long movieId) {...}
```

get() is non-blocking thanks to virtual threads

```java
public static void main(String[] args) {
    try (var executor = Executors.newVirtualThreadPerTaskExecutor()) {
        getMovieIds()
            .parallelStream()
            .map(movieId -> {
                var titleFuture = executor.submit(() -> getTitle(movieId));
                var reviewFuture = executor.submit(() -> getReviews(movieId));
                try {
                    return new Movie(titleFuture.get(), reviewFuture.get());
                } catch (InterruptedException | ExecutionException ex) {...}
            })
            .forEach(System.out::println);
    }
}

private static List<Long> getMovieIds() {...}

private static Title getTitle(Long movieId) {...}

private static List<Review> getReviews(Long movieId) {...}
```

code is not blocking yet clean

Structured Concurrency

```
public static void main(String[] args) {
    getMovieIds().parallelStream()
        .forEach(movieId -> {
            try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {
                var title = scope.fork(() -> getTitle(movieId));
                var reviews = scope.fork(() -> getReviews(movieId));

                scope.join().throwIfFailed(CustomException::new);

                System.out.println(new Movie(title.resultNow(), reviews.resultNow()));
            } catch (InterruptedException ex) {...}
        });
}

private static List<Long> getMovieIds() {...}

private static Title getTitle(Long movieId) {...}

private static List<Review> getReviews(Long movieId) {...}
```

```java
public static void main(String[] args) {
    getMovieIds().parallelStream()
        .forEach(movieId -> {
            try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {
                var title = scope.fork(() -> getTitle(movieId));
                var reviews = scope.fork(() -> getReviews(movieId));

                scope.join().throwIfFailed(CustomException::new);

                System.out.println(new Movie(title.resultNow(), reviews.resultNow()));
            } catch (InterruptedException ex) {...}
        });
}

private static List<Long> getMovieIds() {...}

private static Title getTitle(Long movieId) {...}

private static List<Review> getReviews(Long movieId) {...}
```

```java
public static void main(String[] args) {
    getMovieIds().parallelStream()
        .forEach(movieId -> {
            try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {
                var title = scope.fork(() -> getTitle(movieId));
                var reviews = scope.fork(() -> getReviews(movieId));

                scope.join().throwIfFailed(CustomException::new);

                System.out.println(new Movie(title.resultNow(), reviews.resultNow()));
            } catch (InterruptedException ex) {...}
        });
}

private static List<Long> getMovieIds() {...}

private static Title getTitle(Long movieId) {...}

private static List<Review> getReviews(Long movieId) {...}
```

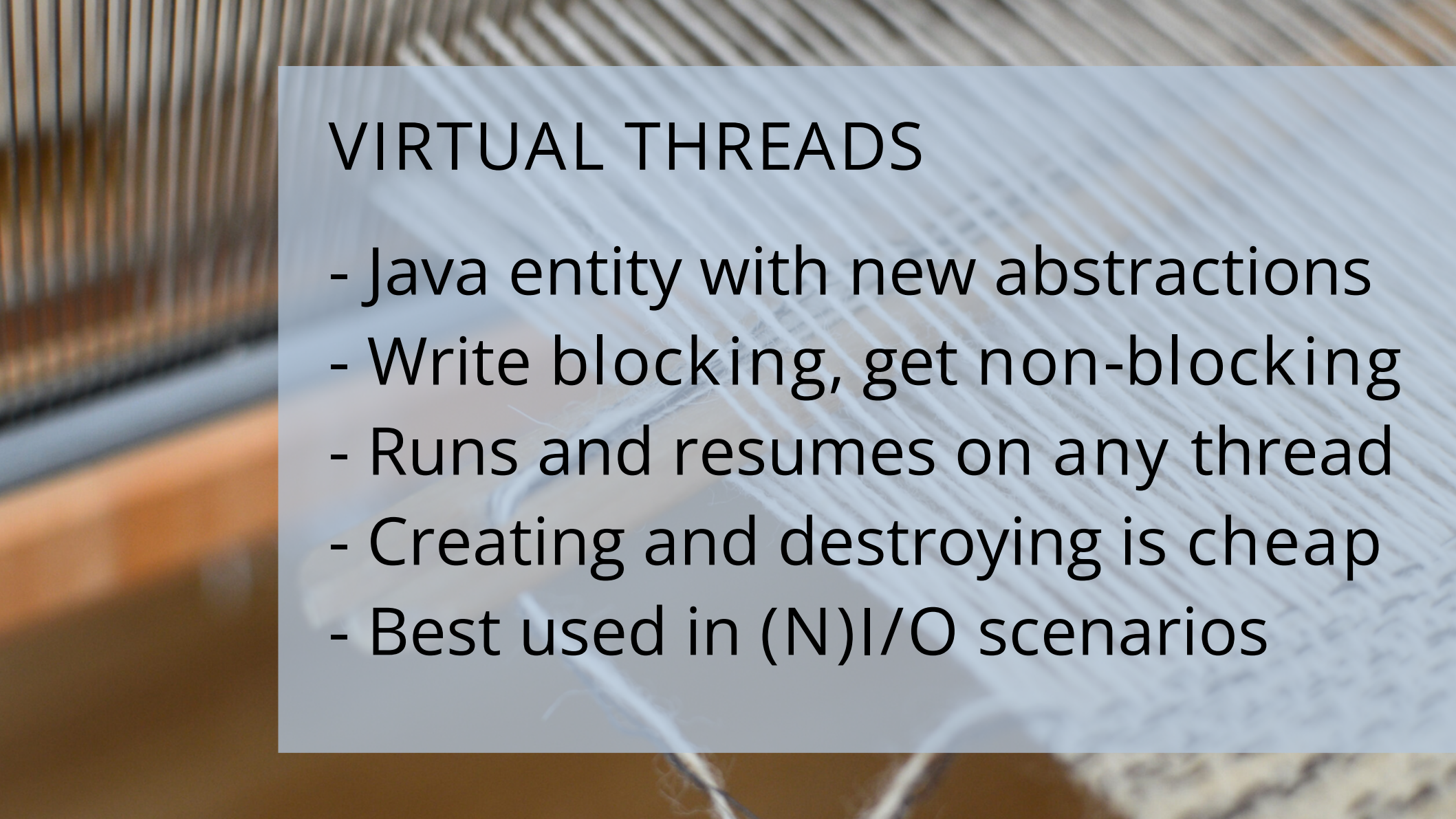
if one fork fails, the other is cancelled

```java
public static void main(String[] args) {
    getMovieIds().parallelStream()
        .forEach(movieId -> {
            try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {
                var title = scope.fork(() -> getTitle(movieId));
                var reviews = scope.fork(() -> getReviews(movieId));

                scope.join().throwIfFailed(CustomException::new);

                System.out.println(new Movie(title.resultNow(), reviews.resultNow()));
            } catch (InterruptedException ex) {...}
        });
}

private static List<Long> getMovieIds() {...}

private static Title getTitle(Long movieId) {...}

private static List<Review> getReviews(Long movieId) {...}
```

join() is once again non-blocking and asynchronous

```java
public static void main(String[] args) {
    getMovieIds().parallelStream()
        .forEach(movieId -> {
            try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {
                var title = scope.fork(() -> getTitle(movieId));
                var reviews = scope.fork(() -> getReviews(movieId));

                scope.join().throwIfFailed(CustomException::new);

                System.out.println(new Movie(title.resultNow(), reviews.resultNow()));
            } catch (InterruptedException ex) {...}
        });
}

private static List<Long> getMovieIds() {...}

private static Title getTitle(Long movieId) {...}

private static List<Review> getReviews(Long movieId) {...}
```

code is not blocking yet clean


VIRTUAL THREADS
- Java entity with new abstractions
- Write blocking, get non-blocking
- Runs and resumes on any thread
- Creating and destroying is cheap
- Best used in (N)I/O scenarios

PROJECT LOOM

Reduce complexity, improve performance.

Thanks for your attention

Please rate my session in the J-Fall app!