Introducing Structured Concurrency

José Paumard

PhD, Java Champion, JavaOne Rockstar

@JosePaumard | https://github.com/JosePaumard

How can you use your virtual threads efficiently? How can you avoid having loose threads?

```
ExecutorService es = ...;
Future<Text> f1 = es.submit(PageReader::readText);
Future<Link> f2 = es.submit(PageReader::readLinks);
Page page = new Page(f1.get(1, TimeUnit.SECONDS),
                     f2.get(1, TimeUnit.SECONDS));
```

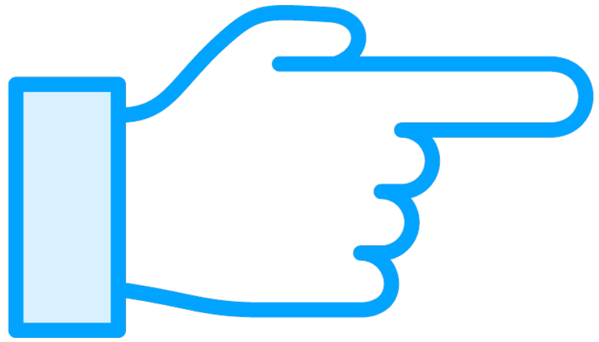
```
ExecutorService es = ...;
Future<Text> f1 = es.submit(PageReader::readText);
Future<Link> f2 = es.submit(PageReader::readLinks);
Page page = new Page(f1.get(1, TimeUnit.SECONDS),
                     f2.get(1, TimeUnit.SECONDS));
```

```
ExecutorService es = ...;
Future<Text> f1 = es.submit(PageReader::readText);
Future<Link> f2 = es.submit(PageReader::readLinks);
Page page = new Page(f1.get(1, TimeUnit.SECONDS),
                     f2.get(1, TimeUnit.SECONDS));
```

Agenda

- Introducing structured concurrency
- Using StructuredTaskScope
- Shutting down on success
- Shutting down on failure
- Creating your own StructuredTaskScope

Introducing Structured Concurrency



Why are we using imperative programming?

Because it is easier to follow the flow of your code

The goto instruction has been banned to avoid jumping anywhere from everywhere

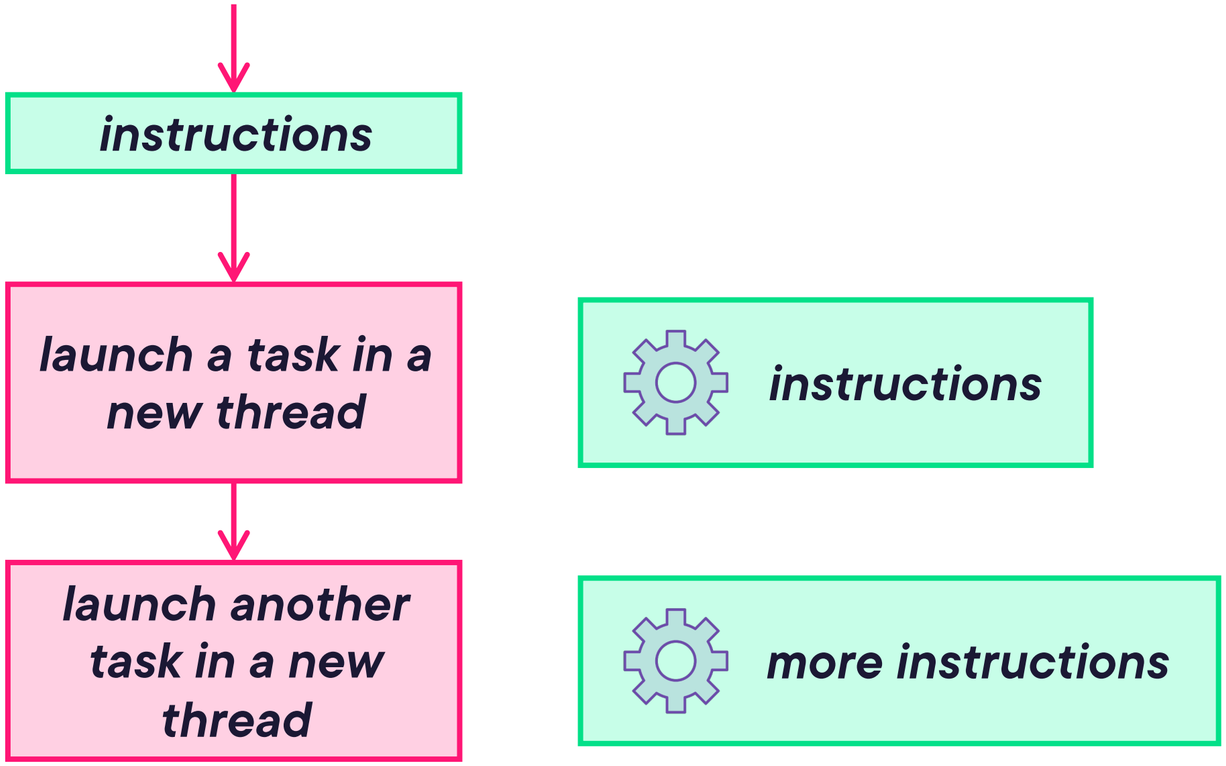
When you look at a line of code

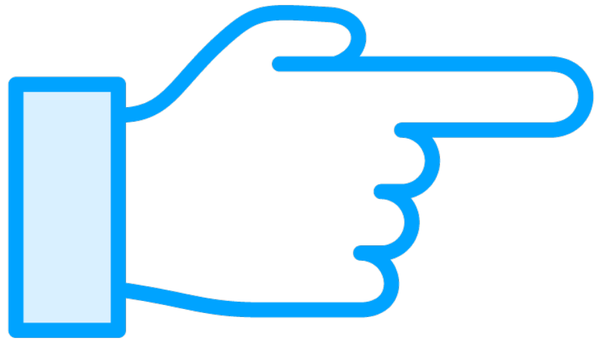
You always know exactly where you are coming from

And where you can go

It makes your application much easier to maintain

Following Some Concurrent Code

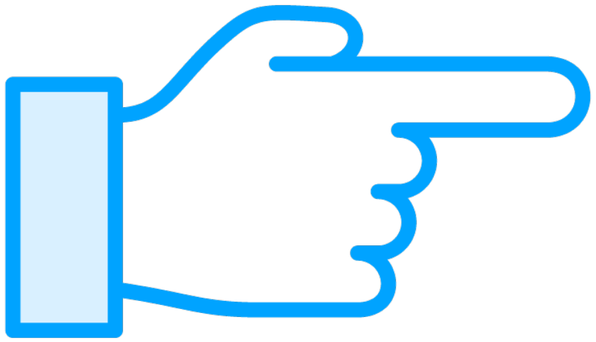




Launching a task in another thread is against the principle of imperative programming

Once launched, the task doesn’t know who launched it

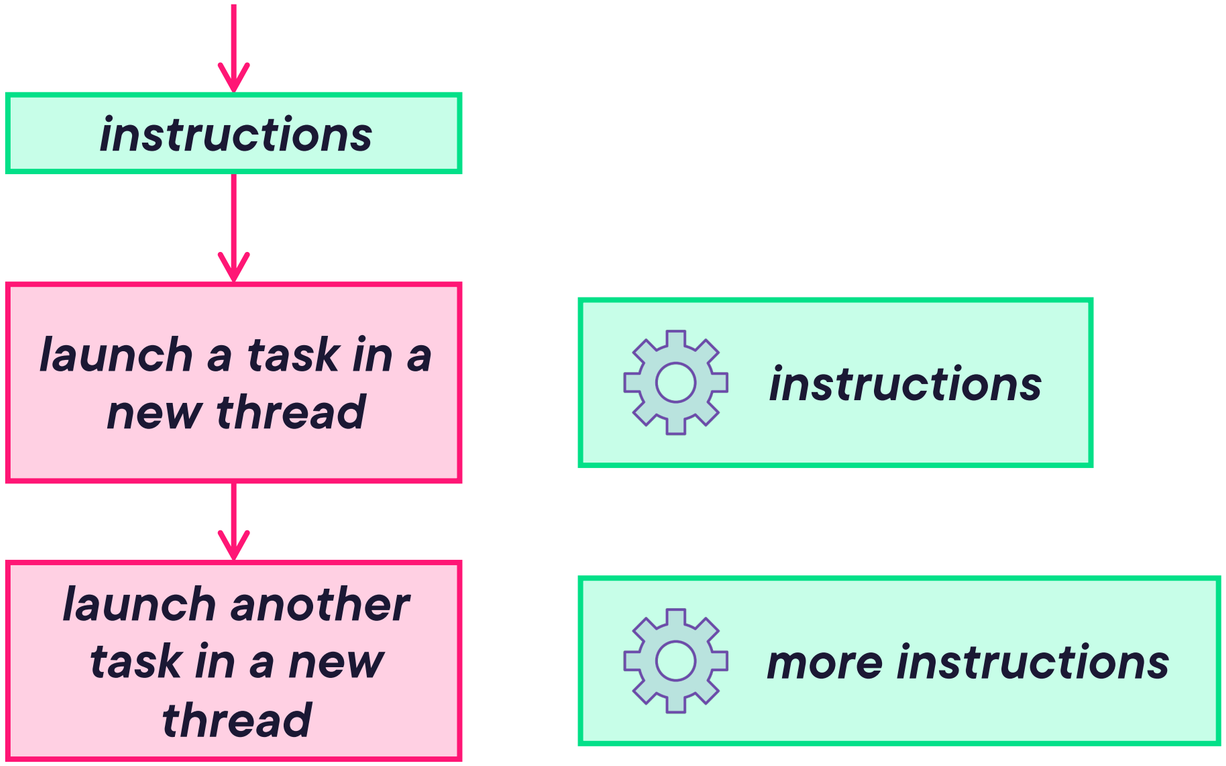
Loose threads are a source of bugs in your application



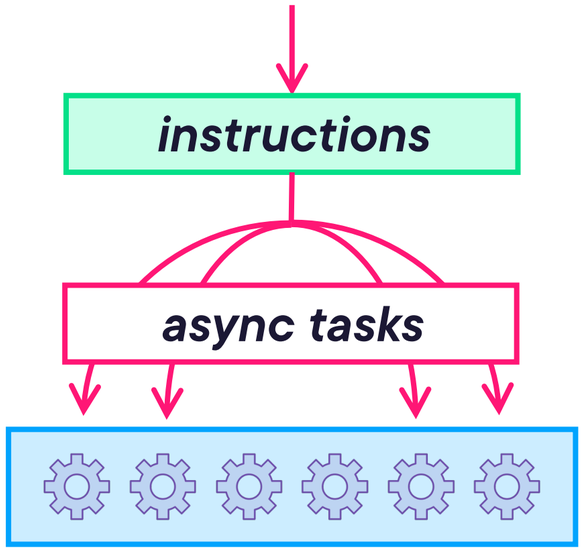
Structured Concurrency is about
getting rid of **loose** threads

Every virtual thread
should have a **bound life cycle**

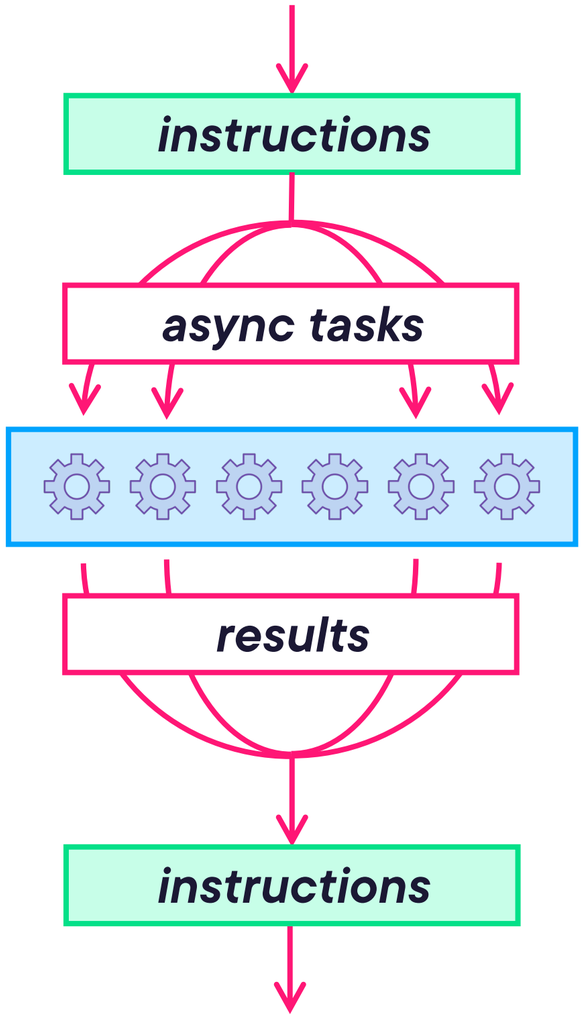
Structured Concurrency



Structured Concurrency



Structured Concurrency



No loose threads using
Structured Concurrency

Once you are done with your tasks,
all your virtual threads
are interrupted and cleaned up

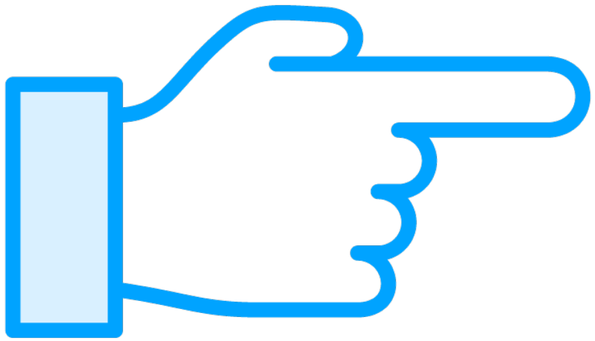
Virtual threads are cheap to create,
letting them die is OK

Using Structured Task Scopes

Structured Concurrency in Action

Let us create Structured Task Scopes

And use them to query a weather forecast server



The StructuredTaskScope class implements this pattern

It acts as an ExecutorService (but it's not)

That creates virtual threads on demand

And cleans up everything when your tasks are done

How can you use a StructuredTaskScope?

Use a try with resources statement

Fork your tasks as Callable instances

Call join()

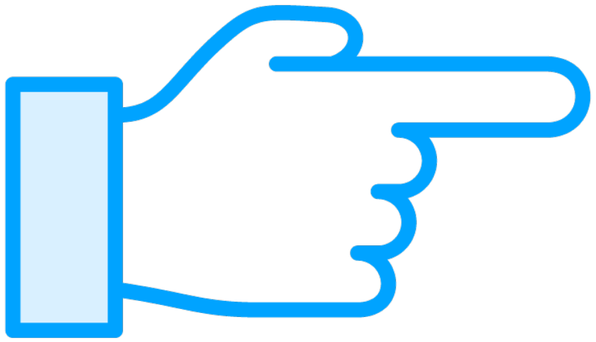
Get the results

Shutting Down on Success

The Shutdown on Success Strategy

Let us use this ShutdownOnSuccess

And use them to improve your weather forecast application



The ShutdownOnSuccess class implements another behavior

It returns a value as soon as a task returns a value

If the first task throws an exception

It waits for the next one

Until there is a result

Creating a Flight Booking Application

Booking Flights with Structured Concurrency

Let us carry on and create a simple flight booking application

Extending StructuredTaskScope

Extending Structured Task Scopes

Let us make your code cleaner

By extending StructuredTaskScope

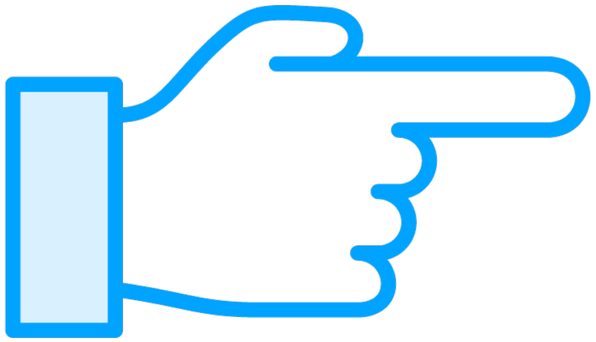
And by separating your business code and your technical code

Shutting Down on Failure

The Shutdown on Failure Strategy

Let us book multileg flights

And use the ShutdownOnFailure scope



The ShutdownOnFailure class implements yet another behavior

It throws an exception as soon as a task fails

All the tasks need to complete successfully

Then it produces a result

Selling Simple and Multileg Flights

Mixing Simple and Multileg Flights

Let us merge the last two scopes

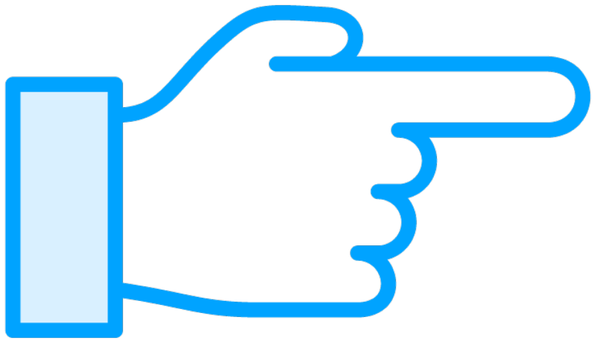
And build an application that can sell simple and multileg flights

Putting It All Together

Mixing Weather Forecast and Flight Booking

Let us put all the pieces together

And add the weather forecast information when somene wished to book a flight

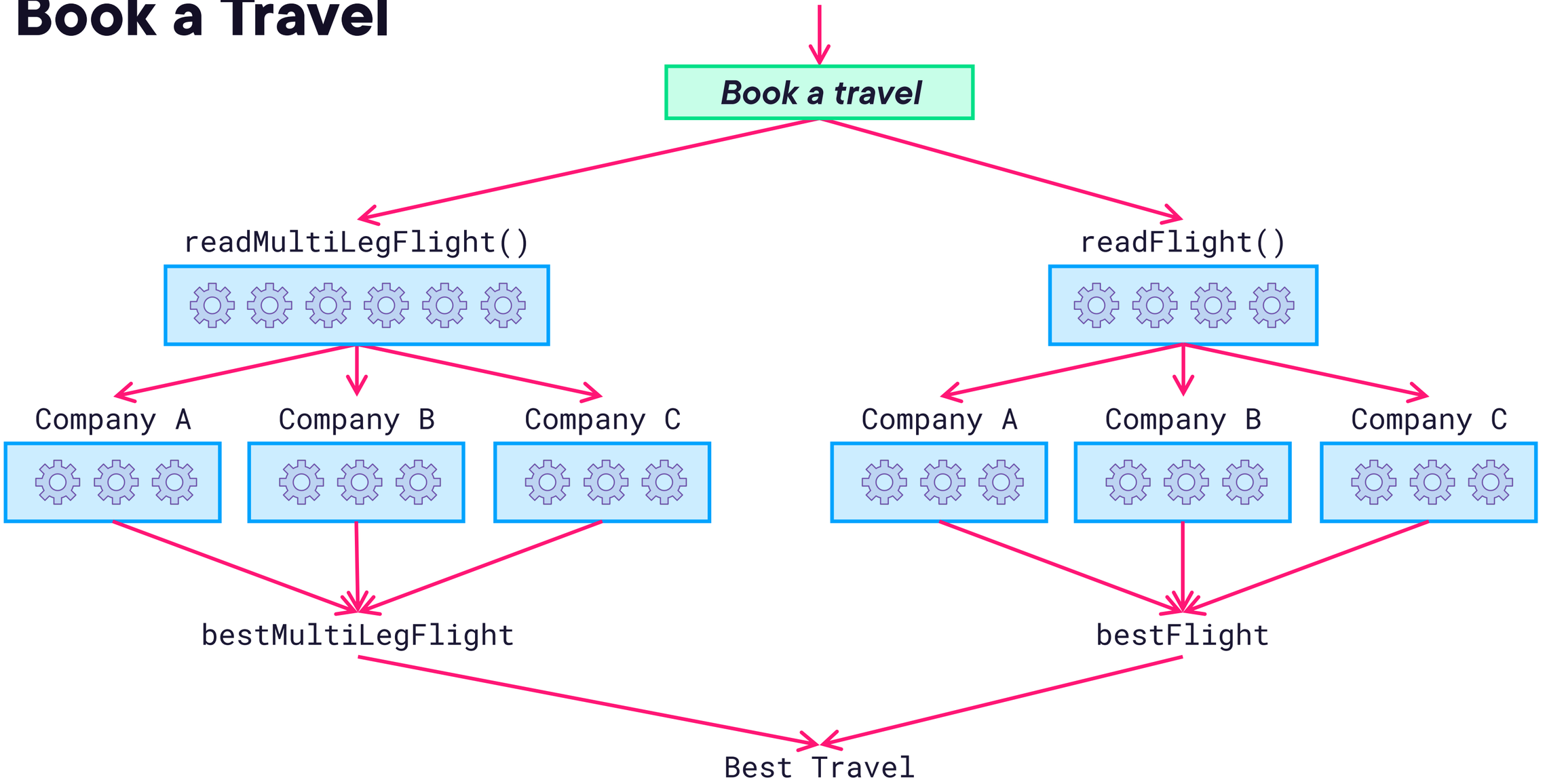


Structured Concurrency is about having as many structured task scopes as you need

You can assemble them
with different strategies

To build a complete application

Book a Travel



Extending scopes can make
your code more readable

To implement your own business strategies

By separating the technical elements
from the business elements

There is one method to override:

handleComplete()

You can then separate your results from the exceptions

And report them separately

Module Wrap Up

Structured Concurrency is about controling the life cycle of your virtual threads

One principle: no more loose threads

Three objects:

StructuredTaskScope: Autocloseable

ShutdownOnSuccess: returns as soon a task produces a result

ShutdownOnFailure: fails as soon as a task failed

Up Next:

Replacing ThreadLocals with ScopedValues