Triggering a Task on the Outcome of Another Task

José Paumard
PHD, Java Champion, JavaOne RockStar

@JosePaumard https://github.com/JosePaumard

Agenda

Introducing the CompletionStage API

How to chain tasks

To process your data asynchronously

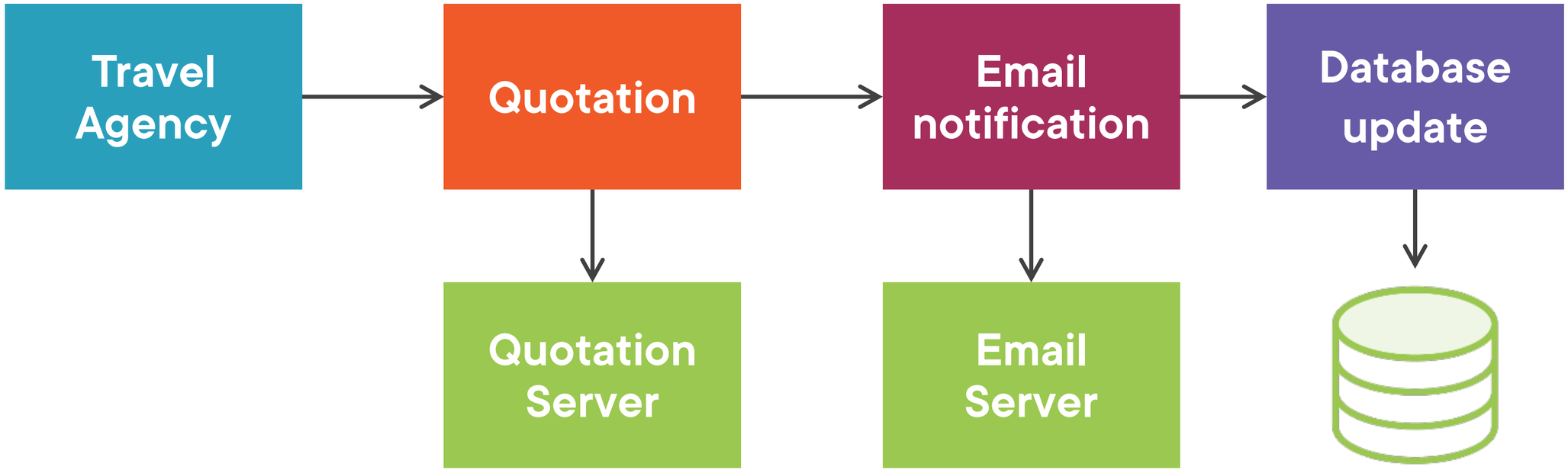
Where Does it Block?

A thread is an operating system resource

Threads are expensive!

You do not want to block too many threads

The Travel Agency Example



```
ExecutorService service = ...;

Future<Quotation> futureQuotation =
    service.submit(() -> getQuotation());
Quotation quotation = futureQuotation.get();

Future<Email> futureEmail =
    service.submit(() -> email(quotation));
Email emailInfos = futureEmail.get();

Future<DB> futureDB =
    service.submit(() -> writeToDB(emailInfos));
boolean done = futureDB.get();
```

```
ExecutorService service = ...;

Future<Quotation> futureQuotation =
   service.submit(() -> getQuotation());
Quotation quotation = futureQuotation.get();

Future<Email> futureEmail =
   service.submit(() -> email(quotation));
Email emailInfos = futureEmail.get();

Future<DB> futureDB =
   service.submit(() -> writeToDB(emailInfos));
boolean done = futureDB.get();
```

Blocking a thread for a long time is expensive!

You want to avoid having to get the result back to the main thread

The solution is to trigger a task on the outcome of another task

A First CompletionStage Pattern

The CompletionStage interface is the center of the API

The CompletableFuture class implements CompletionStage

It also implements the Future interface

```
CompletableFuture<Quotation> quotationCF =
        CompletableFuture.supplyAsync(
            () -> getQuotation());

CompletableFuture<EmailInfos> infosCF =
        quotationCF.thenApply(
            quotation -> email(quotation));

CompletableFuture<Boolean> doneCF =
        infosCF.thenApply(
            emailInfos -> writeToDB(emailInfos));

doneCF.thenApply(done -> updateGUI(done));
```

You should never block your main thread when using CompletionStage

You need to divide the processing of your data in small operations

That you can chain with the CompletionStage API

One task can trigger as many tasks as you need

Demo

Live demo!

Let us chain tasks with the CompletionStage API

And see what thread is executing what task

Module Wrap Up

What did you learn?

How to chain tasks using CompletionStage

A task can trigger as many tasks as you need

How can you handle the chaining of a task that is itself asynchronous?

Up Next: Splitting a Result into Several Asynchronous Tasks