Splitting a Result into Several Asynchronous Tasks

José Paumard
PHD, Java Champion, JavaOne RockStar

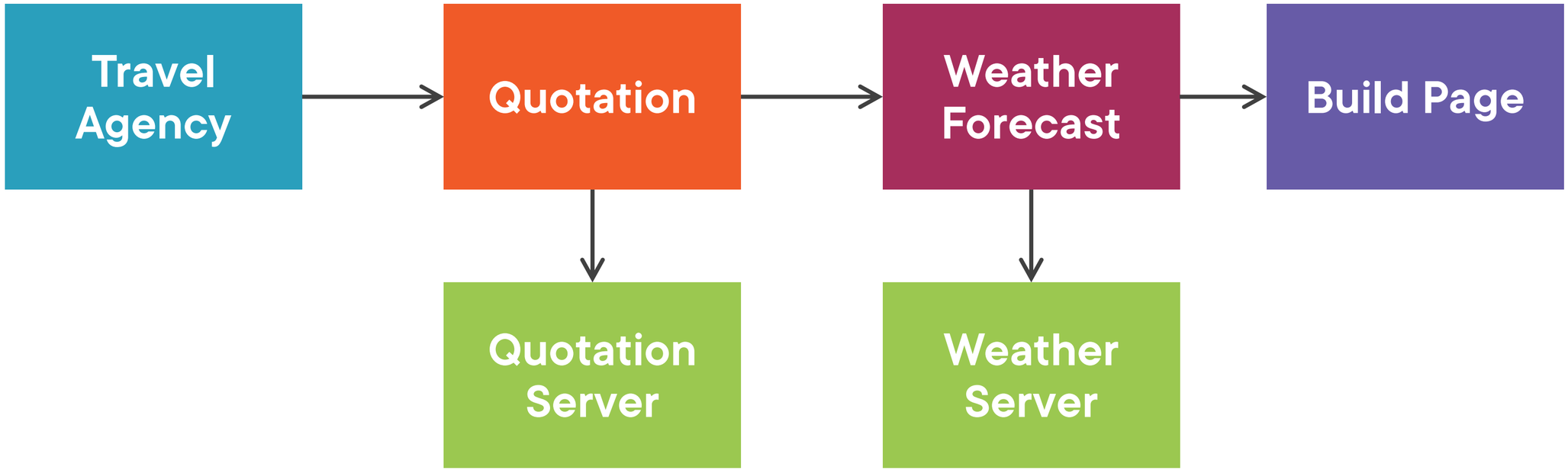
@JosePaumard https://github.com/JosePaumard

Agenda

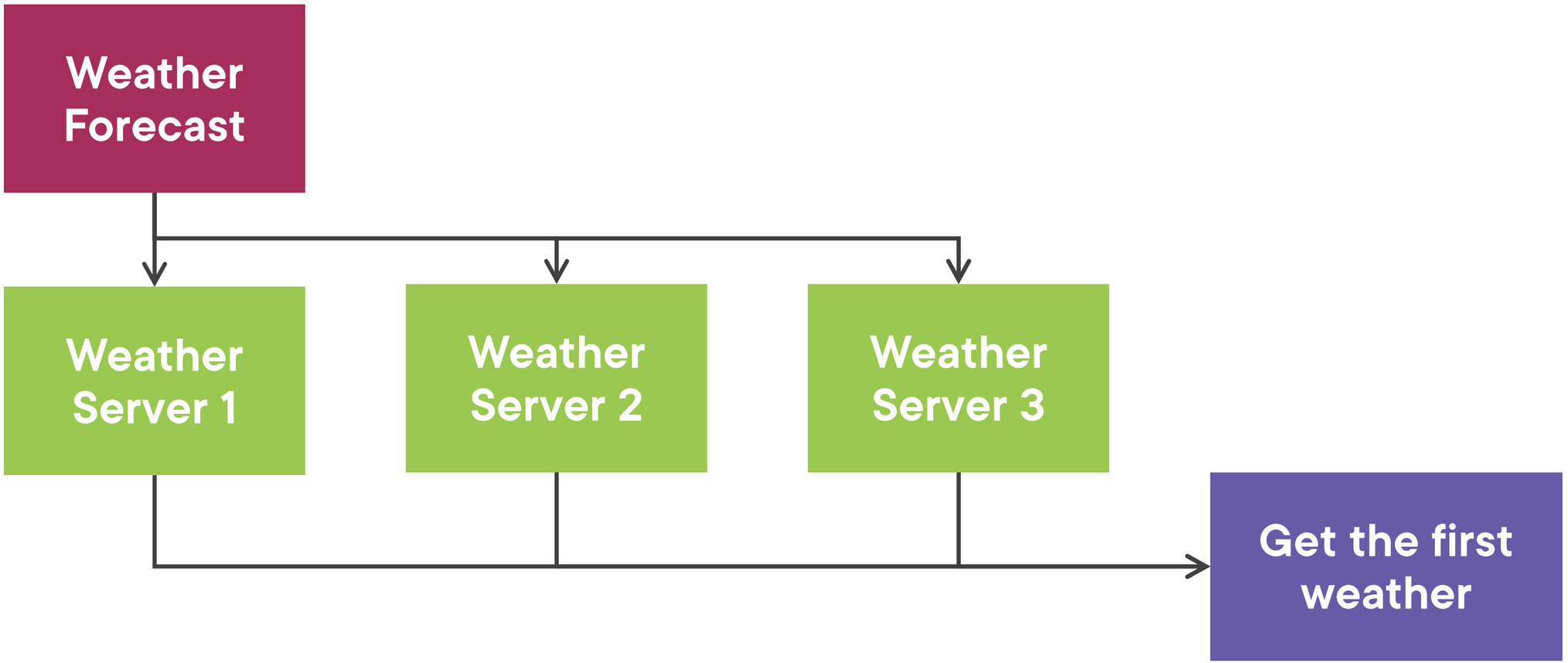
- You know how to launch one task
- And trigger synchronous actions on its output
- What about launching several tasks?
- What about chaining asynchronous tasks?

Launching Several Tasks At Once

The Travel Agency Example



The Travel Agency Example

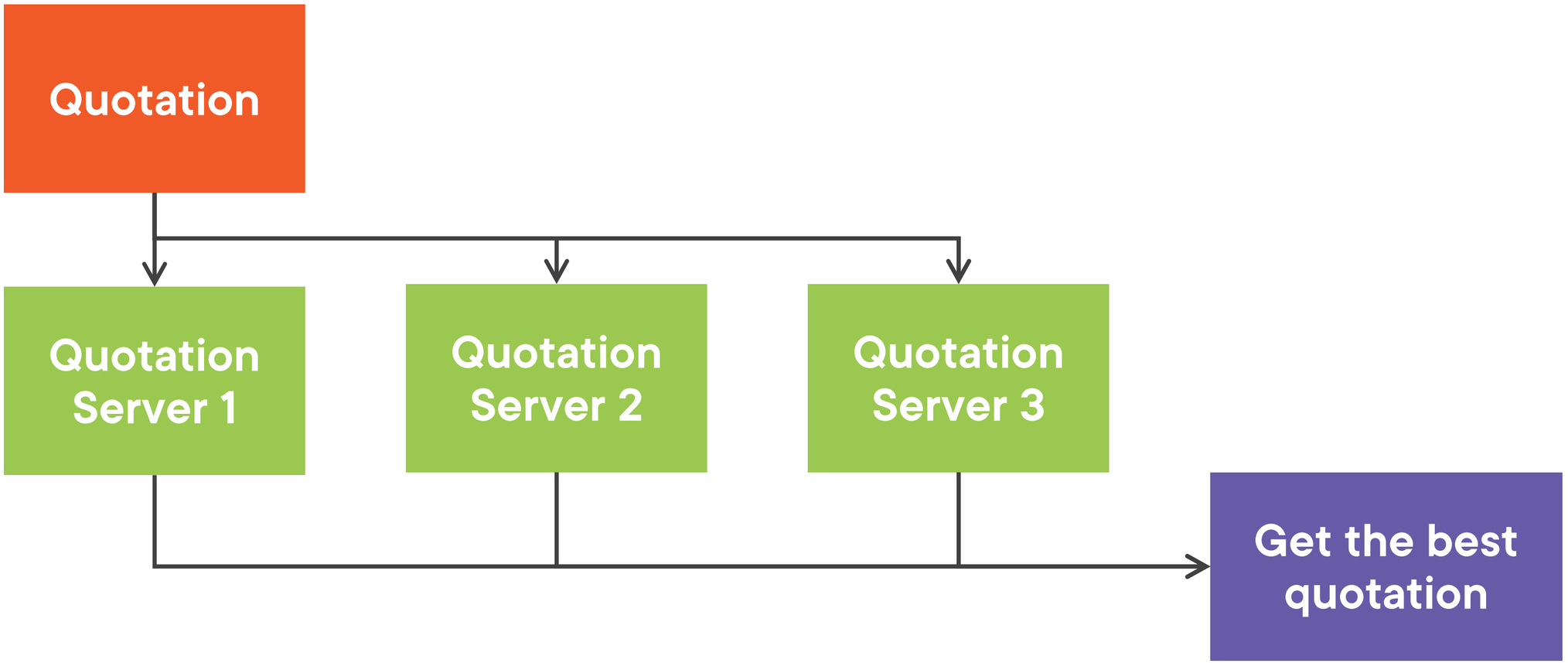


```
Supplier<Weather> w1 = () -> getWeatherA();
Supplier<Weather> w2 = () -> getWeatherB();
Supplier<Weather> w3 = () -> getWeatherC();

CompletableFuture<Weather> cf1 =
    CompletableFuture.supplyAsync(w1);

// same for cf2, cf3, ...

CompletableFuture<Object> weatherCF =
    CompletableFuture.anyOf(cf1, cf2, cf3);
```

CompletableFuture.anyOf():

- returns a CompletableFuture
- that completes on the first task
- returns the result of this first task
- normally or exceptionally

The Travel Agency Example



```
Supplier<Quotation> q1 = () -> getQuotationA();
Supplier<Quotation> q2 = () -> getQuotationB();
Supplier<Quotation> q3 = () -> getQuotationC();

CompletableFuture<Quotation> cf1 =
   CompletableFuture.supplyAsync(q1);

// same for cf2, cf3, ...

CompletableFuture<Void> done =
   CompletableFuture.allOf(cf1, cf2, cf3);
```

```
CompletableFuture<Void> done =
    CompletableFuture.allOf(cf1, cf2, cf3);

Quotation bestQuotation =
    done.thenApply(
        v -> Stream.of(cf1, cf2, cf3)
                   .map(CompletableFuture::join)
                   .min(comparing(Quotation::amount))
                   .orElseThrow())
        ).join();
```

CompletableFuture.allOf():

- returns a CompletableFuture

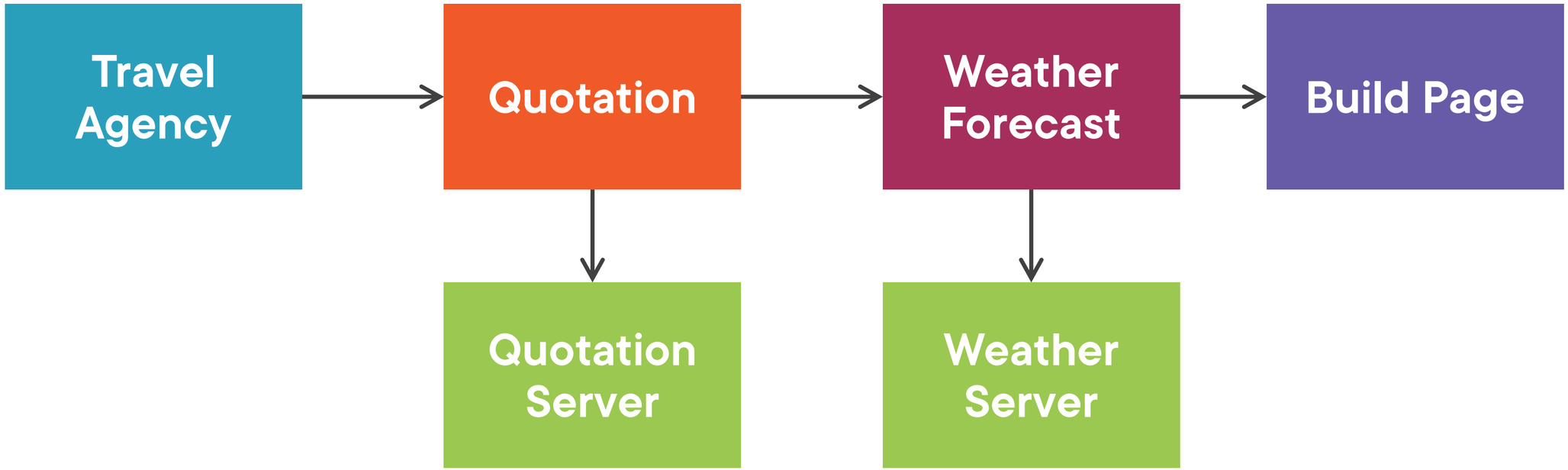
- that completes on all the tasks

- returns null or exceptionally

- normally or exceptionally

Chaining Asynchronous Tasks

The Travel Agency Example



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

var weatherCF =
    CompletableFuture.supplyAsync(() -> getWeather());

var travelPage =
    new TravelPage(quotationCF.get(), weatherCF.get());
```

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

var weatherCF =
    CompletableFuture.supplyAsync(() -> getWeather());

var travelPage =
    quotationCF.thenApply(
        quotation ->
            new TravelPage(quotation, weatherCF.get());
```

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

var weatherCF =
    CompletableFuture.supplyAsync(() -> getWeather());

TravelPage travePage =
    quotationCF.thenCompose(
        quotation -> weatherCF.thenApply(
            weather -> new TravelPage(quotation, weather)));
```

Composing CompletableFuture
is what you need to build a result
on two asynchronous tasks

You can compose CompletionStage

It leads to complex patterns

But this is the most efficient way to produce a result from a set of asynchronous tasks

Demo

Let us see some code!

Gather the results over several tasks

And compose several asynchronous tasks

Module Wrap Up

What did you learn?

How to build asynchronous pipelines on several asynchronous tasks

How to compose asynchronous tasks to prevent blocking

Up Next: Controlling What Thread Can Execute a Task