Reporting and Recovering from Errors

José Paumard
PHD, Java Champion, JavaOne RockStar

@JosePaumard https://github.com/JosePaumard

Agenda

CompletableFuture are often used for I/O operations

These operations may fail, with Exceptions

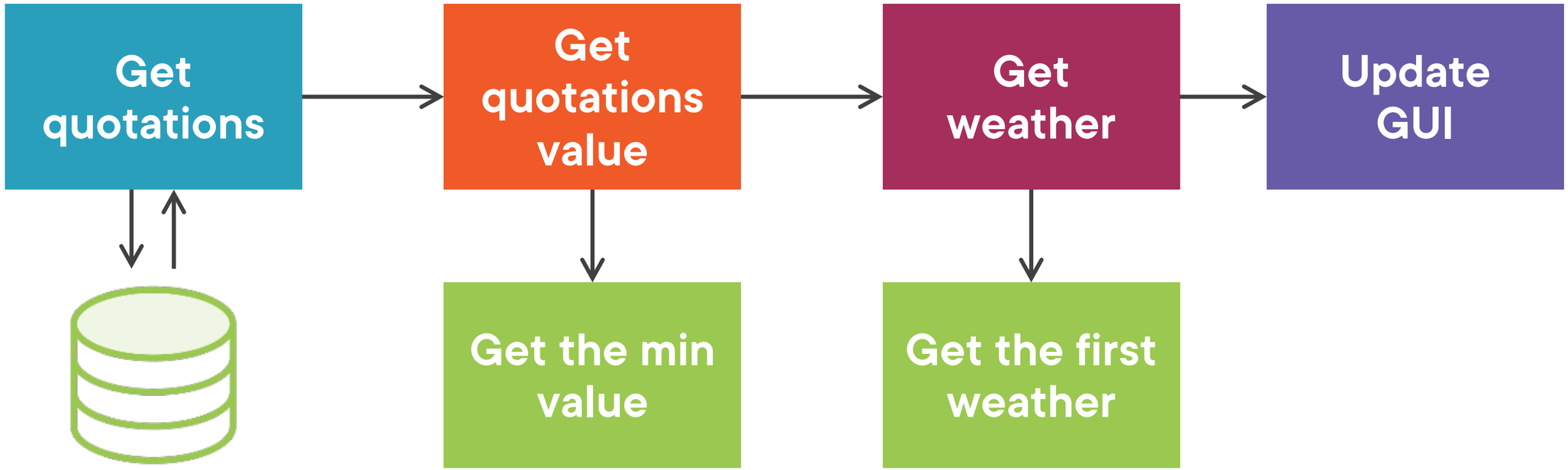
You have two ways of handling them

- reporting them

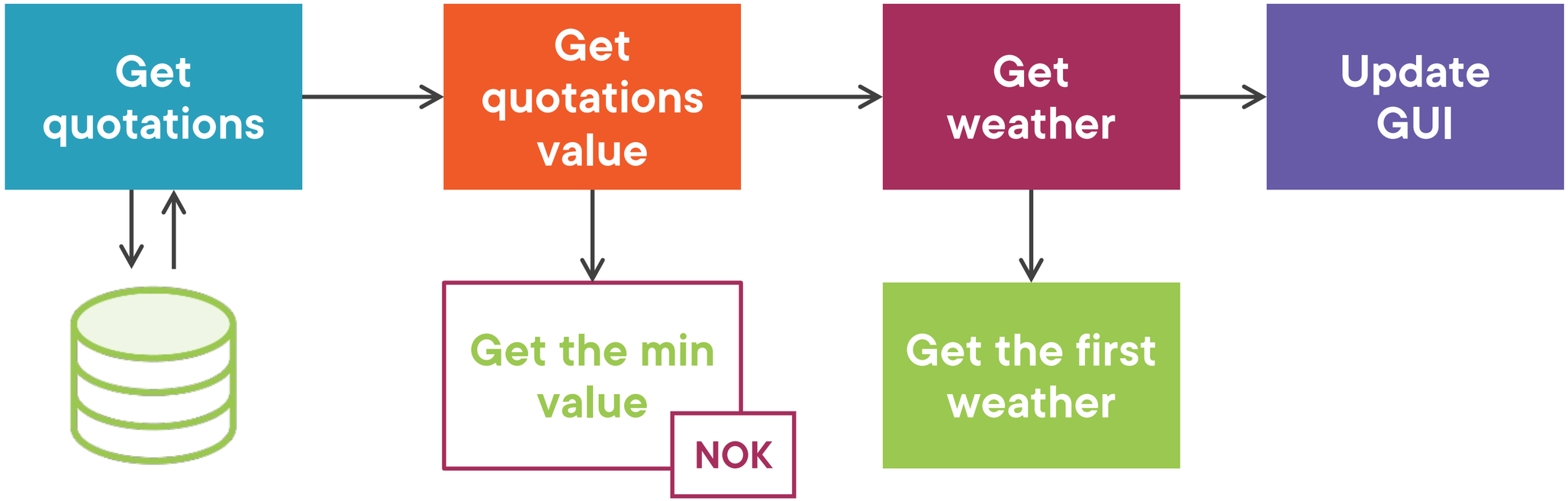
- recovering from them

When a Task Completes Exceptionally

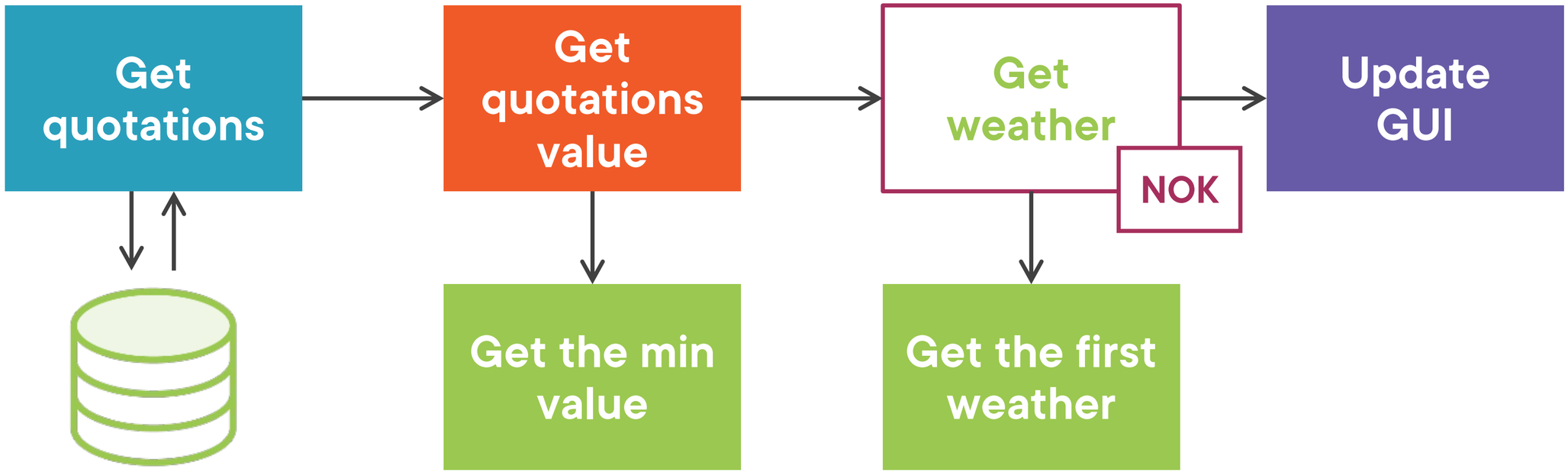
What is Happening When a Task is not Working?



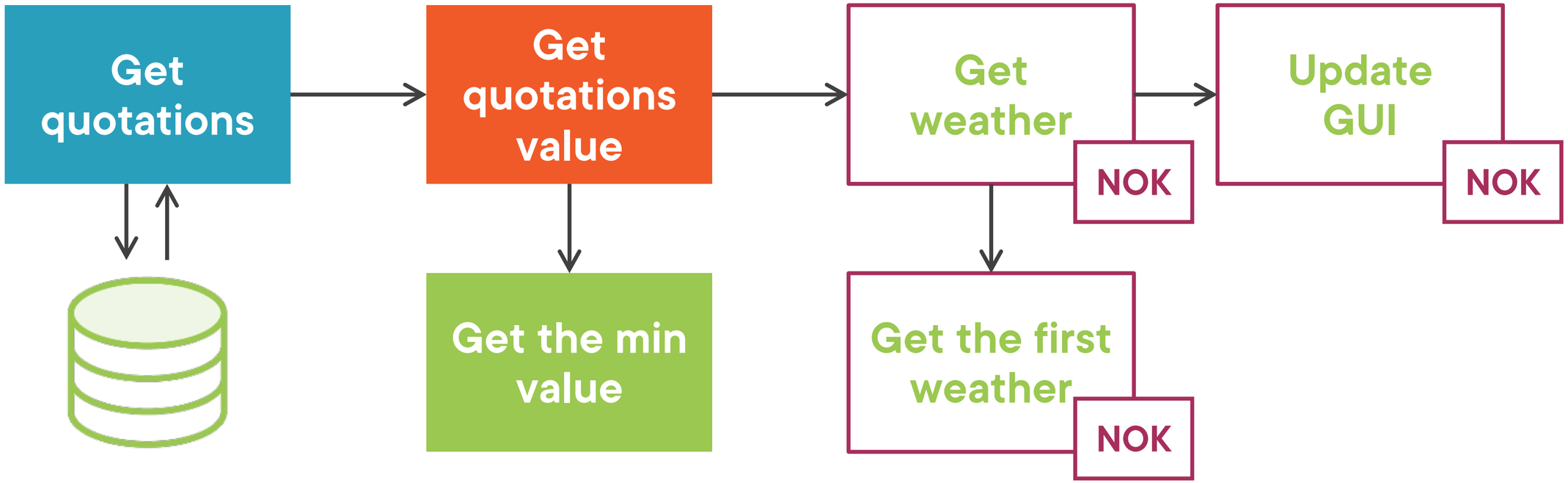
What is Happening When a Task is not Working?



What is Happening When a Task is not Working?



What is Happening When a Task is not Working?



If a task completes with an exception

Then:

- an exception is thrown when you try to get the result

- all the downstream tasks will also throw an exception

There are two ways of dealing with exceptions

1 – recovering and providing a default value

2 – forwarding the exception

The CompletableFuture API has 3 methods to implement these two behaviors

All these can be executed in a specific executor

Recovering and Providing a Default Value

```
Supplier<Weather> w = () -> getWeather();

CompletableFuture<Weather> cf =
    CompletableFuture.supplyAsync(w);
```

If getWeather() throws an exception,

Then cf completes with an exception

```
Supplier<Weather> w = () -> getWeather();

CompletableFuture<Weather> cf =
    CompletableFuture.supplyAsync(w)
                     .exceptionally(t -> new Weather(...));
```

If getWeather() throws an exception,

Then cf completes with the provided default value

Handling a Result and an Exception

There are two more patterns, that takes both the exception and the result as parameters

- the exception may be null

- the result may be null

```
Supplier<Weather> w = () -> getWeather();
CompletableFuture<Weather> cf =
    CompletableFuture.supplyAsync(w);

cf.whenComplete(
    (weather, exception) -> {
        if (exception != null)
            logger.error(exception);
    });
```

whenComplete() takes a BiConsumer

It is not meant to be followed by any more task

```
Supplier<Weather> w = () -> getWeather();
CompletableFuture<Weather> cf =
   CompletableFuture.supplyAsync(w);

cf.handle(
   (weather, exception) -> {
      if (exception != null) {
         logger.error(exception);
         return new Weather(...);
      } else
         return weather;
   });
```

Demo

Let us see some code!

Let us see how exceptions can be thrown, forwarded, and recovered from

Module Wrap Up

What did you learn?

How the CompletableFuture API can handle exceptions:

- Just let them crash your data processing pipeline

- Recover from them by providing default value

Up Next: Closing Remarks