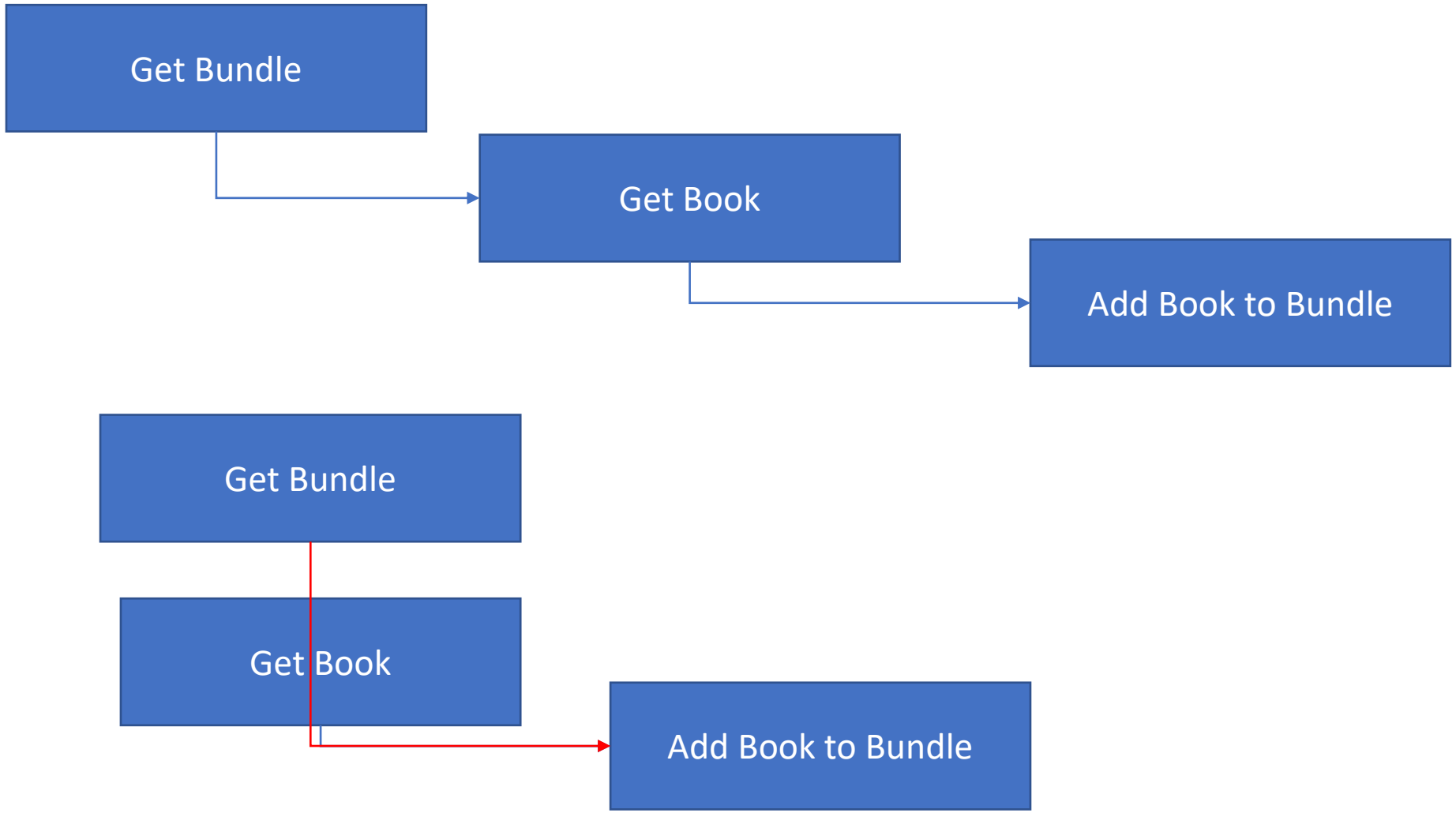
Programação na Internet

Turma i52d

Lesson 32
Sequential versus Concurrent

Sequential <versus> Concurrent



async

`parallel(tasks, callback)`

Run the `tasks` array of functions in parallel. If any of the functions pass an error to its callback, the main `callback` is immediately called with the value of the error. Once the `tasks` have completed, the results are passed to the final `callback` as an array.

Promise

https://en.wikipedia.org/wiki/Futures_and_promises

Container of an asynchronous result:

⇒ May hold a successful or failure result.

Asynchronous Idioms:

1. _callback_ `(err, data) => {...}` -- `err` and `data` are 2 possible results
2. `EventEmitters` `.on('error', callback)` e `.on('data', callback)`.
3. **`Promise` - Java CompletableFuture, .Net Task.**
4. `async` / `await` -- exist in most environments e.g..Net, Python, Js, etc except Java.
5. suspend functions

Promise

3 possible states:

- Pending
- Fulfilled (success)
- Rejected (error)

…then(…) – receives a continuation

```
…then(
    val => …,  // executed when it is fullfiled
    err => …) // executed when it is rejected
// returns a new Promise that allows to chain another .then(…)
```

Promise

- `new Promise()` `// state Pending`
- `Promise.resolve()` `// state Fulfilled`
- `Promise.resolve().then(… => res)` `// state Fulfilled with res`

- `….then(val => …, err => …)` `// returns a new Promise`
  - The callback (or continuation) will be performed when the previous task is completed(*fulfilled* or *rejected*)
  - The result of the new Promise will be the result of the continuation.