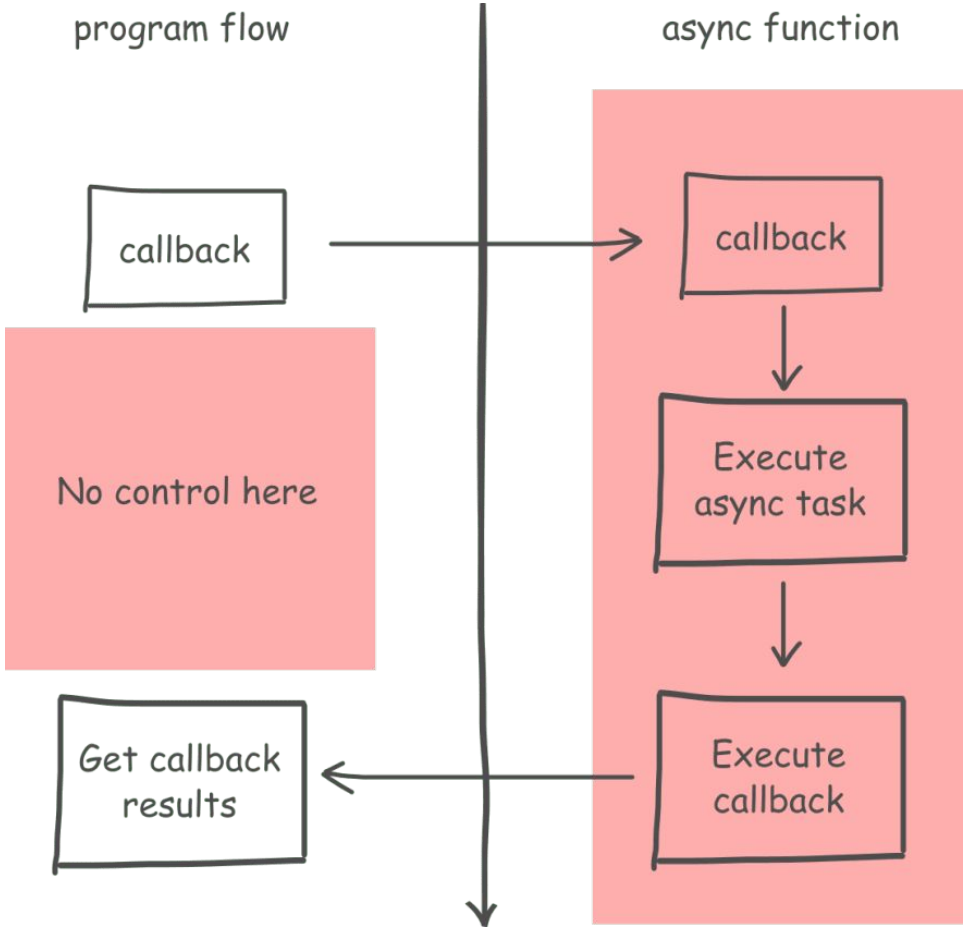
Callbacks, promises, async/await

Callback hell
Error handling

Callbacks



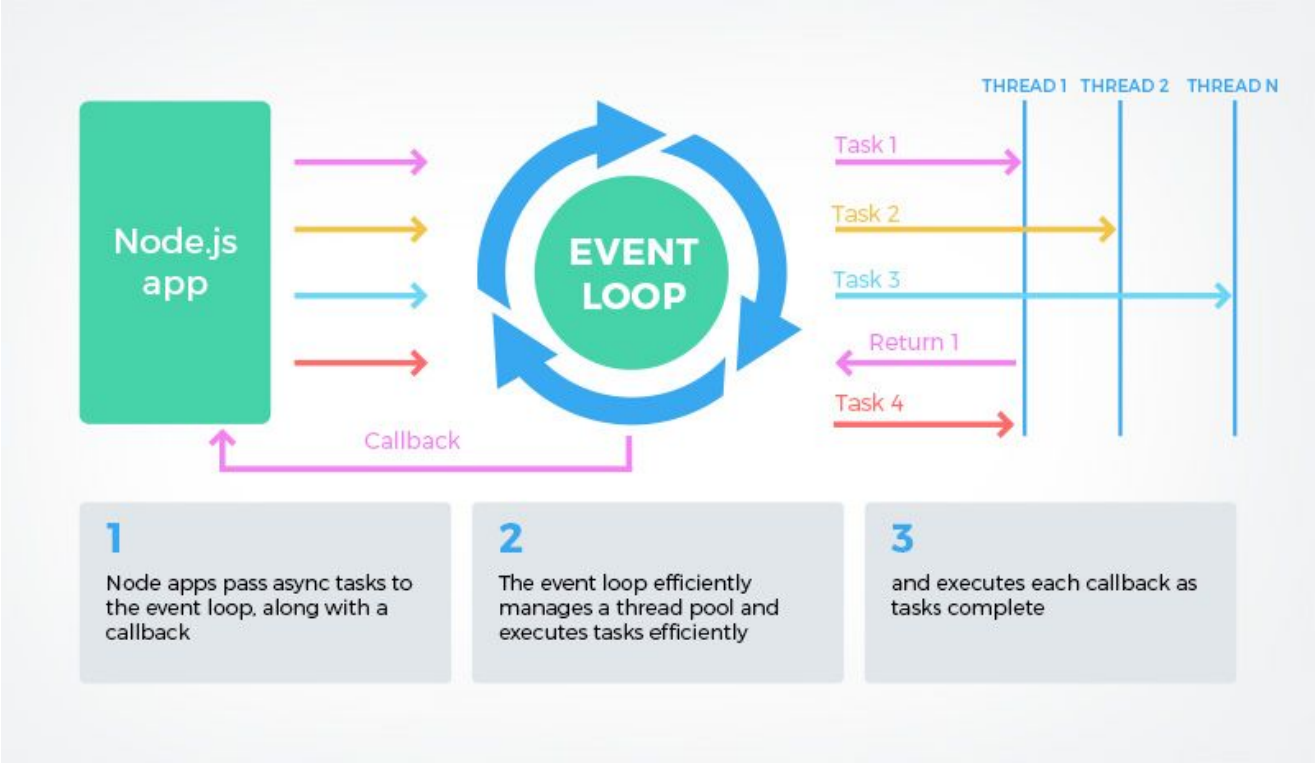
★ **Дефиниция:** Функция, използваща се като параметър на друга функция.

Извиква се след приключване на дадената асинхронна обработка.

```
1  function callback(success, error) {
2      if (!error) {
3          console.log(success);
4      }
5  }
6  asynchronousTask(callback);
```

Callbacks

★ **Приложения:** извличане данни от DB, изтегляне на файл, отправяне на заявки към API



Callbacks

Pros vs Cons

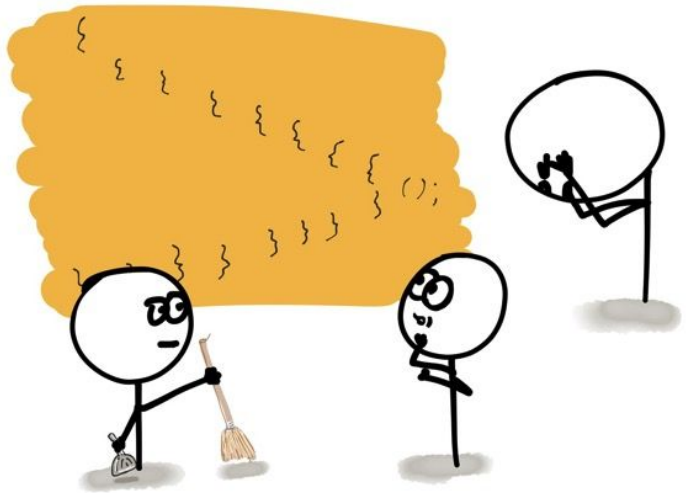
- Бързина и ефективност

- Може да доведе до **callback hell (пирамида на обречеността)**

Error handling with callbacks

- Един от най-често срещаните методи за обработване на грешки
- Callback функцията се състои от 2 аргумента - error и result

```
callback(err, result)
```



Callback hell(Pyramid of doom)

- **Какво представлява? -> множество вложени callbacks функции**
- ***Получава се при опит за изпълнение на множество асинхронни операции едни след други.***

```
function hell(win) {
 // for listener purpose
 return function() {
   loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
     loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
       loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
         loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
           loadLink(win, REMOTE_SRC+'/lib/underscode.min.js', function() {
             loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
               loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
                 loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
                   loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
                     async.eachSeries(SCRIPTS, function(src, callback) {
                       loadScript(win, BASE_URL+src, callback);
                     });
                   });
                 });
               });
             });
           });
         });
       });
     });
   });
 };
}
```

Callback hell(Pyramid of doom)

Решения на проблема:

Използване на:

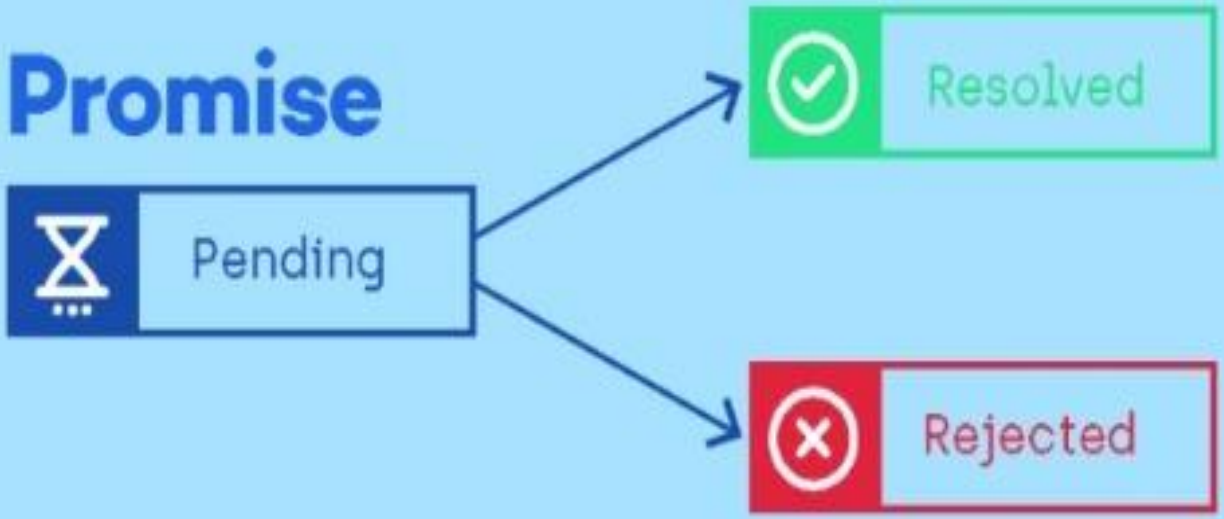
- ➔ **Promises**
- ➔ **Async-Await**

```
//Example for callback hell
getData(function(x){
    getMoreData(x, function(y){
        getMoreData(y, function(z){
            ...
        });
    });
});
```

PROMISES

TypeScript Promises

“I may not know what the value of the return is right now, but I promise to return it at some point, stored inside this thing”.



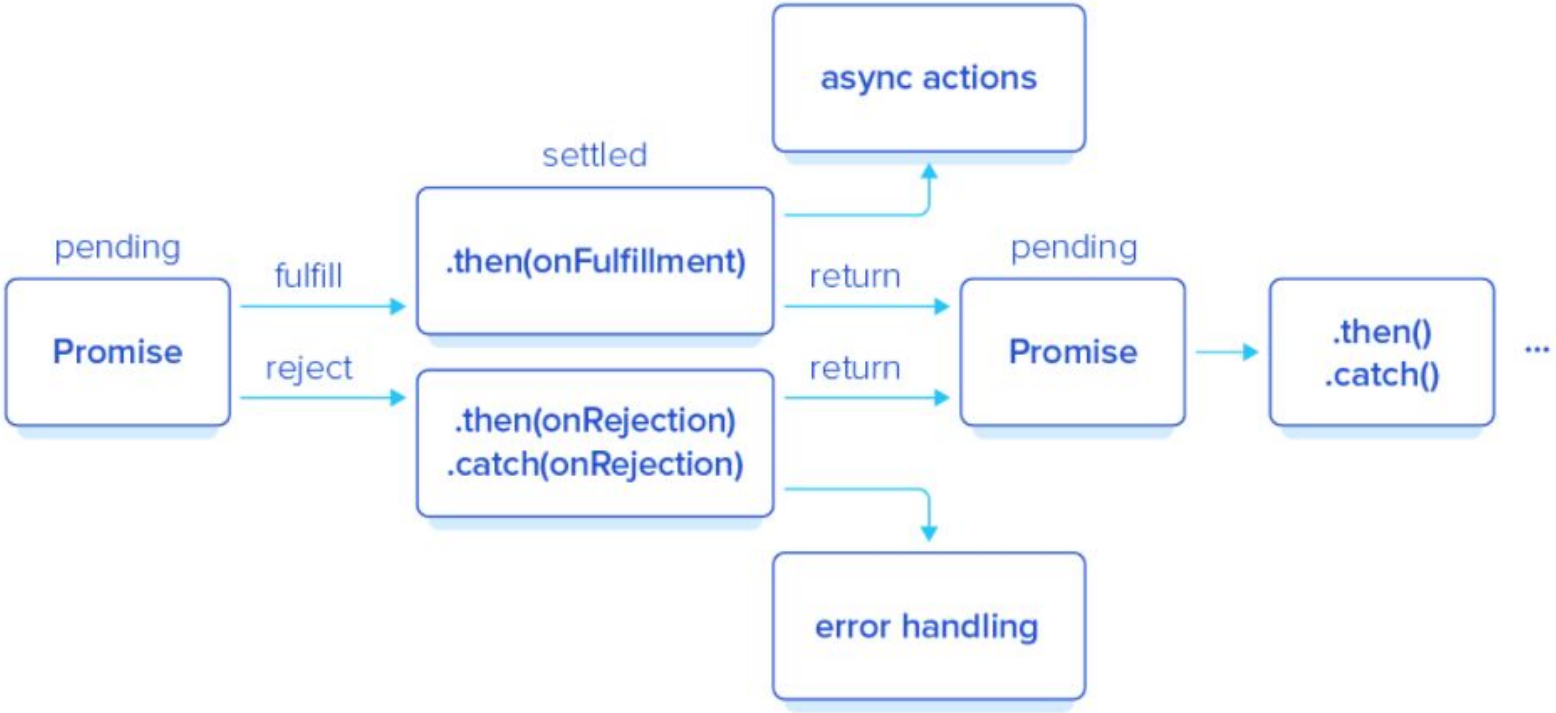
Promises - основни характеристики

1. Основните методи са then(), catch(), finally()

2. Определят се като обект, използван за бъдещ резултат от асинхронни изчисления.

3. Връща promise вместо крайна стойност.

4. Подобрява четимостта на кода и има по-добро обработване на грешки


async actions
settled
pending
fulfill
.then(onFulfillment)
return
pending
Promise
.then()
.catch()
...
reject
return
Promise
.then(onRejection)
.catch(onRejection)
error handling

DO NOT KILL A MOSQUITO WITH A CANNON!

If you're handling multiple errors on your Promises like this, you're doing it wrong.

Async/Await

A sink

A weight

Async/Await - Защо?

Синтактична захар, която улеснява работата с отложени стойност.

async автоматично слага стойности в отложена стойност.

await автоматично изважда стойности от отложена стойност.

Async/Await - Характеристики

1. Функции, декларирани като async, винаги връщат отложена стойност (явно или неявно).
2. await спира действието на функции, декларирани като async, докато отложената стойност, която се изчаква не бъде разрешена или отхвърлена.
3. Ако отложената стойност е разрешена, то await автоматично ще я върне, иначе ще се получи изключение.
4. Единственото място, в което може да се използва await е във функция, декларирана като async.

Async/Await - Error handling

Ако се използва await, за да се извади отложена стойност, която е била отхвърлена, ще се получи изключение.

То може да бъде прихванато с try и обработено с catch.

Забележка: Променливата, която се прихваща в catch, е от тип символен низ.

Asynchronous TypeScript