# ASYNCHRONOUS CODING IN NODEJS

## CALLBACKS AND CONTROL FLOW

Thomas Roch, Formedix Ltd

## NODE’S EXPERIENCE ANYONE?

## WHAT IS NODEJS

*Node.js uses an **event-driven**, **non-blocking I/O** model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.*

```
• Event driven    => event listeners
• Non-blocking IO => callbacks
```

## =

## ASYNCHRONOUS

## BEFORE WE START

Non-blocking functions take callbacks (last argument) of two forms:

```
function (err, res) {
    // Do something
}

function (res) {
    // Do something
}
```

## WE ARE GOING TO TALK ABOUT

- Callback hell
- Inversion of control
- Control flow
- Promises
- Thunks
- Generators (or iterators)

## EXAMPLE

A wish list server where one can:

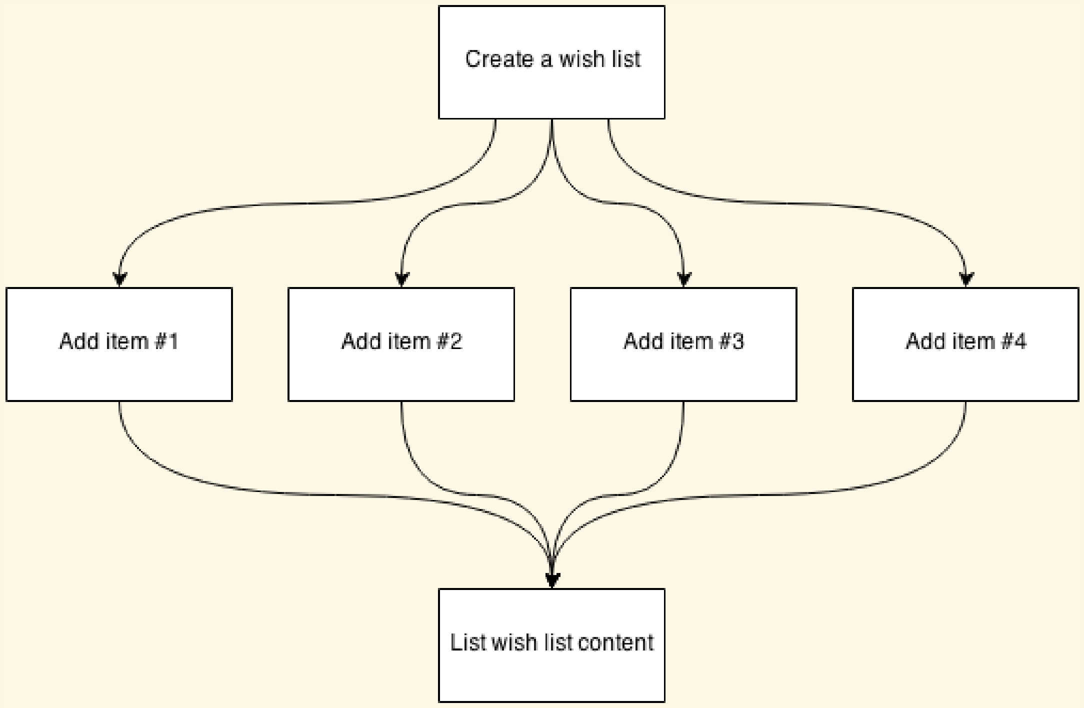
- Create a whish list
- Add items to a whish list
- Retrieve the content of a whishlist

Code available here

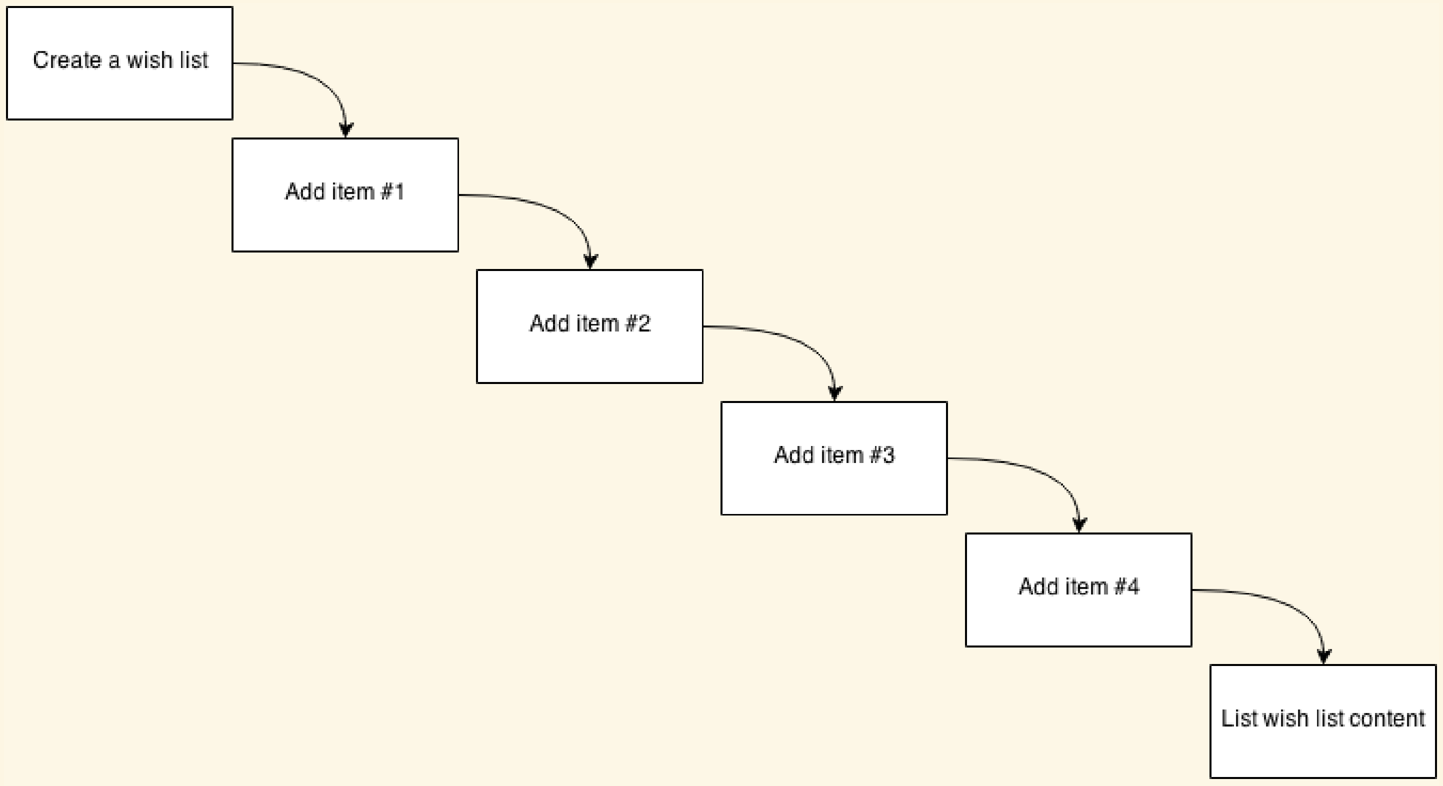
## REST API

- `POST /whishList`
- `POST /whishList/{id}`
- `GET  /whishList/{id}`

## WHAT WE AIM FOR



## WHAT WE ARE GOING TO DO FIRST



## EXAMPLE A

A mix of events and non-blocking functions with http.request() for:

- Creating a list
- Adding an item to a list
- Getting a list's content

## EXAMPLE B:

Turn this mess into functions accepting callbacks

With 3 functions: createList, addItem, getList...

...we are creating a nice "callback hell" aka "**callback pyramid**"

## ASYNC TO THE RESCUE

First control flow example with https://github.com/caolan/async

- async.series()
- async.waterfall()
- async.parallel()
- Works with any non-blocking function "the node way"

## OK... WE'VE DONE STUFF IN PARALLEL

- Does this look good?
- What about avoiding iversion of control?

## EXAMPLE C: USING PROMISES

- ES6 feature but implementations available in ES5
- Returns a deferred result (success or error) accessed using .then(): createList, addItem, getList
- Control flow using q: chaining promises, error propagation, Q.all...
- Compatible with Node API functions using Q.nfcall()

## BEFORE EXAMPLE D

Generators (or iterators)! What are they?

```
function* () {
    yield 1;

    return 2;
}
```

## EXAMPLE D: USING GENERATORS

- What about yielding a promise or a thunk? a what?
- Control flow using co

```
// The node way
phoneMyPal(number, function (err, res) {
    console.log('hi');
});

// A thunk
phoneMyPal(number)(function (err, res) {
    console.log('hi');
});
```

Let's thunkify createList, addItem, getList

## EXAMPLE D: USING GENERATORS

- co works with thunks or promises
- Compatible with Node API using thunkify
- Non-blocking code looking like it is not!
- Used by koa.js, a web application framework

## QUESTIONS