

# CALLBACKS & EVENTS EMITTERS

---

*Can we talk about this later?*

# WHAT IS A CALLBACK?



# WHAT IS A CALLBACK?

**Technically: a function passed to another function**

two flavors...

- **Blocking**
- **Non-blocking**



# BLOCKING CALLBACKS

*think: portable code*

## predicates

e.g. `arr.filter(function predicate (elem) {...});`

## comparators

e.g. `arr.sort(function comparator (elemA, elemB) {...});`

## iterators

e.g. `arr.map(function iterator (elem) {...});`



# NON-BLOCKING CALLBACKS

*think: control flow*

# EVENT EMITTERS

- **A code pattern of deferring certain functions to execute only in response to certain “events”**
- **Exactly like adding an event listener to a DOM event!**
- **Also exactly like Express middleware!**
- **Not restricted to events that are emitted by the environment - we listen for and emit any events we choose by writing our own event emitter**

```
var userTweets = new EventEmitter();
```

```
// Elsewhere in the program . . .
```

```
userTweets.on('newTweet', function (tweet) {  
    console.log(tweet);  
});
```

```
// Elsewhere in the program . . .
```

```
userTweets.emit('newTweet', {  
    text: 'Check out this fruit I ate'  
});
```

```
var userTweets = new EventEmitter();
```

```
// Elsewhere in the program . . .
```

```
userTweets.on('newTweet', function (tweet) {  
    console.log(tweet);  
});
```

```
// Elsewhere in the program . . .
```

```
userTweets.emit('newTweet', {  
    text: 'Check out this fruit I ate'  
});
```



```
var userTweets = new EventEmitter();
```

```
// Elsewhere in the program . . .
```

```
userTweets.on('newTweet', function (tweet) {  
    console.log(tweet);  
});
```

```
// Elsewhere in the program . . .
```

```
userTweets.emit('newTweet', {  
    text: 'Check out this fruit I ate'  
});
```

```
var userTweets = new EventEmitter();
```

```
// Elsewhere in the program . . .
```

```
userTweets.on('newTweet', function (tweet) {  
    console.log(tweet);  
});
```

```
// Elsewhere in the program . . .
```

```
userTweets.emit('newTweet', {  
    text: 'Check out this fruit I ate'  
});
```

```
var userTweets = new EventEmitter();
```

```
// Elsewhere in the program . . .
```

```
userTweets.on('newTweet', function (tweet) {  
    console.log(tweet);  
});
```

```
// Elsewhere in the program . . .
```

```
userTweets.emit('newTweet', {  
    text: 'Check out this fruit I ate'  
});
```

# EVENT EMITTERS

- **Objects that can “emit” specific events with a payload to any amount of registered listeners**
- **An instance of the “observer/observable” a.k.a “pub/sub” pattern**
- **Feels at-home in an *event*-driven environment**



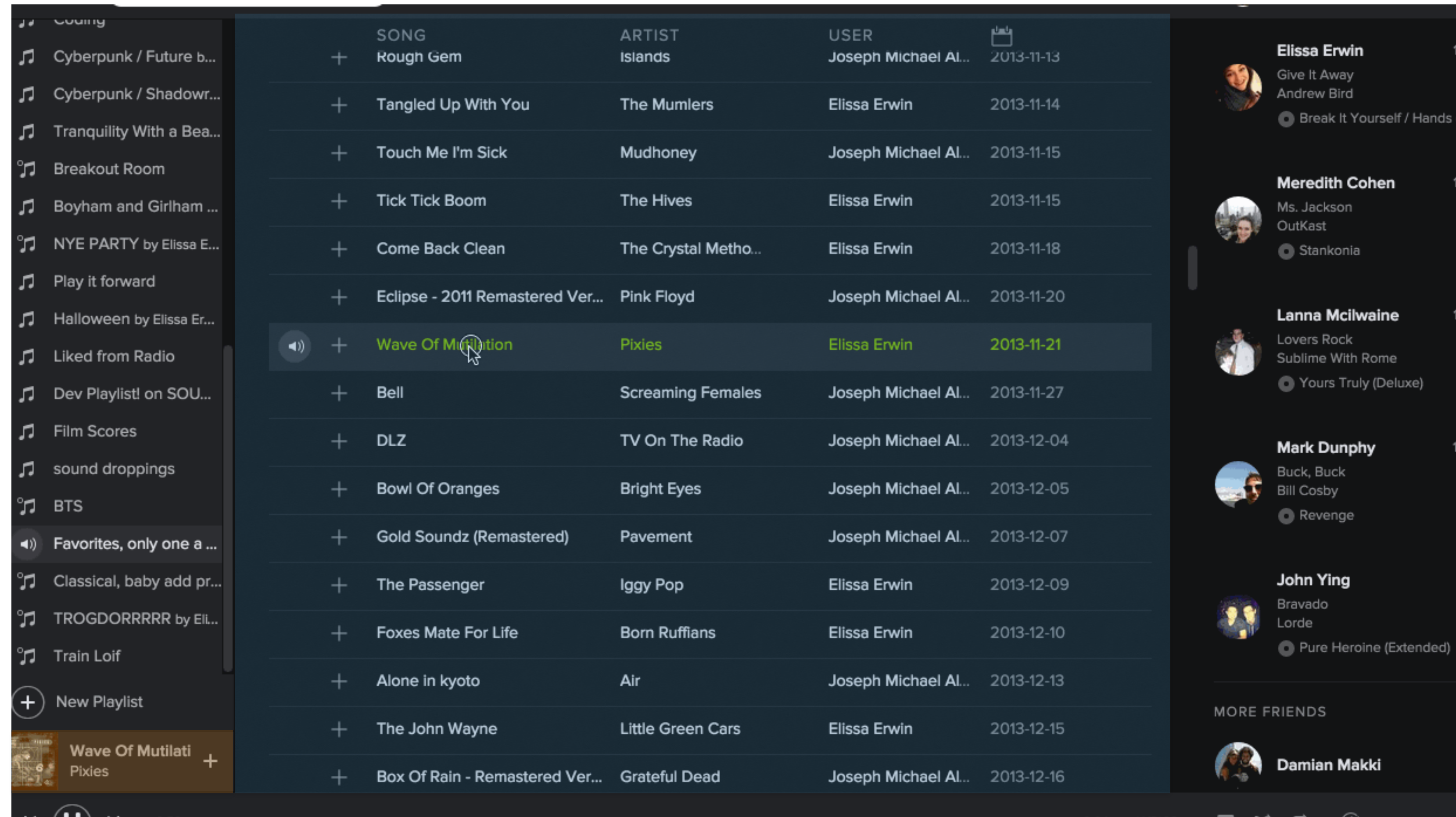
# PRACTICAL USES

- Connect two decoupled parts of an application

```
var currentTrack = new EventEmitter();
```

```
currentTrack.emit('changeTrack', newTrack);
```

```
currentTrack.on('changeTrack', function (newTrack) {  
  // Display new track!  
});
```



# PRACTICAL USES

- Represent multiple asynchronous events on a single entity.

```
var upload = uploadFile();
```

```
upload.on('error', function (e) {  
  e.message; // World exploded!  
});
```

```
upload.on('progress', function (percentage) {  
  setProgressOnBar(percentage);  
});
```

```
upload.on('complete', function (fileUrl, totalUploadTime) {  
  
});
```

# ALL OVER NODE

- **server.on('request')**
- **request.on('data') / request.on('end')**
- **process.stdin.on('data')**
- **db.on('connection')**
- **Streams**