CLEANING UP CODE

What are we trying to improve

- debuggability
- readability
- ease to maintain

EXAMPLE 1

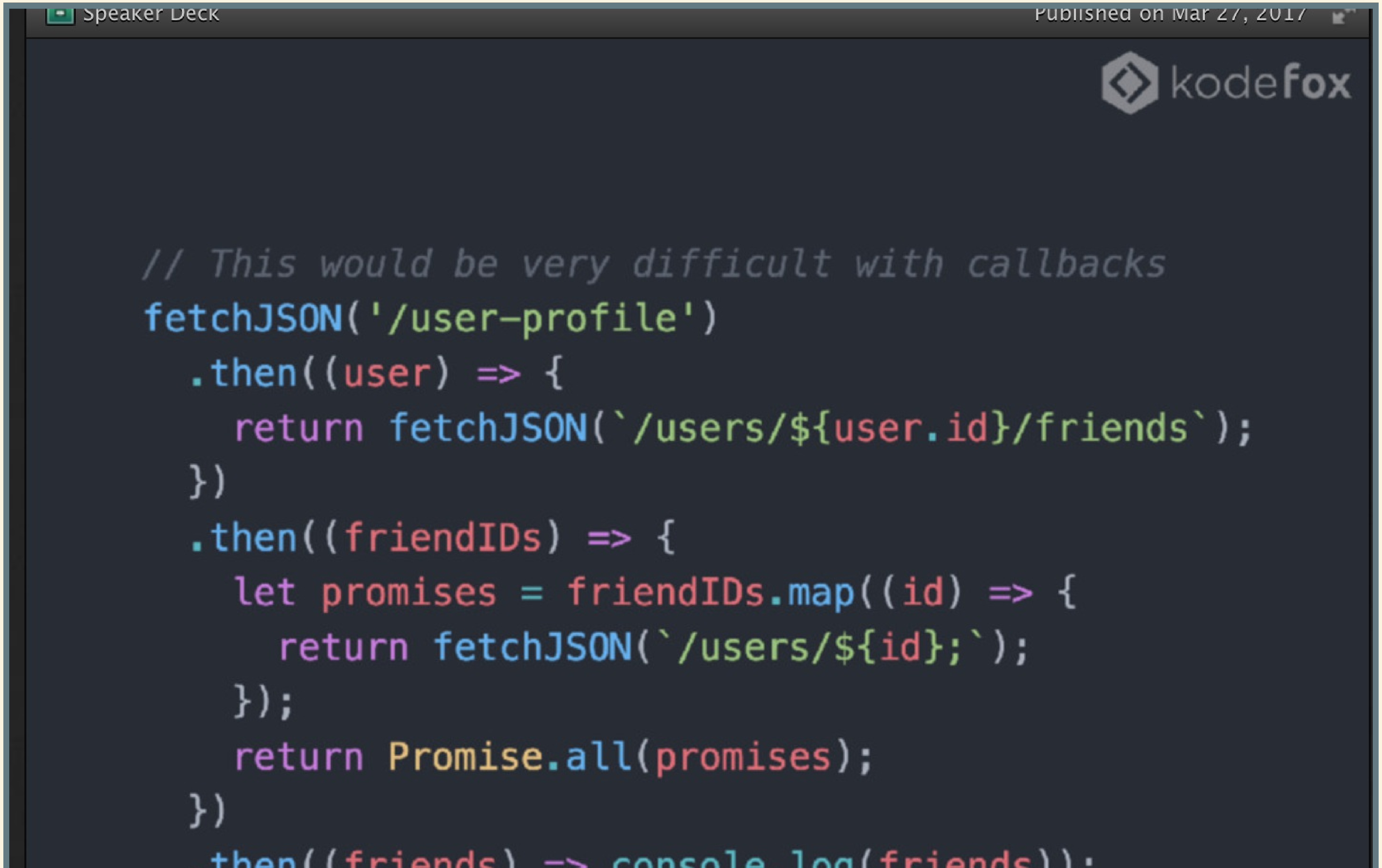
COMMON CASES

- You're making Facebook
- You make a query to load a user
  - pass the userId
  - get the users friends
  - get all the profiles of the user's friends
- https://speakerdeck.com/sstur/async-and-await-bandungjs-mar-2017

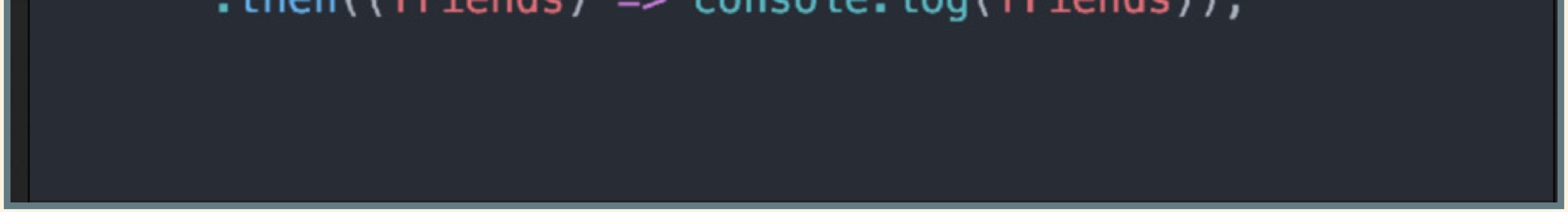
CALLBACKS BASED

```
let friendProfiles = [];

// Fetch User
fetchJSON('user-profile', function(err, user) {
    if (err) {return};

    // Fetch User's friends
    fetchJSON(`/users/${user.id}/friends`, function(err, friendId
        if (err) {return}

        // Get All Friends Profiles
        friendIDs.map((id) => {
            fetchJSON(`/users/${id}`, (err, profile)=>{
                if (err) {friendProfiles.push(null)};
                friendProfiles.push(profile)
            })
```

PROMISE BASED

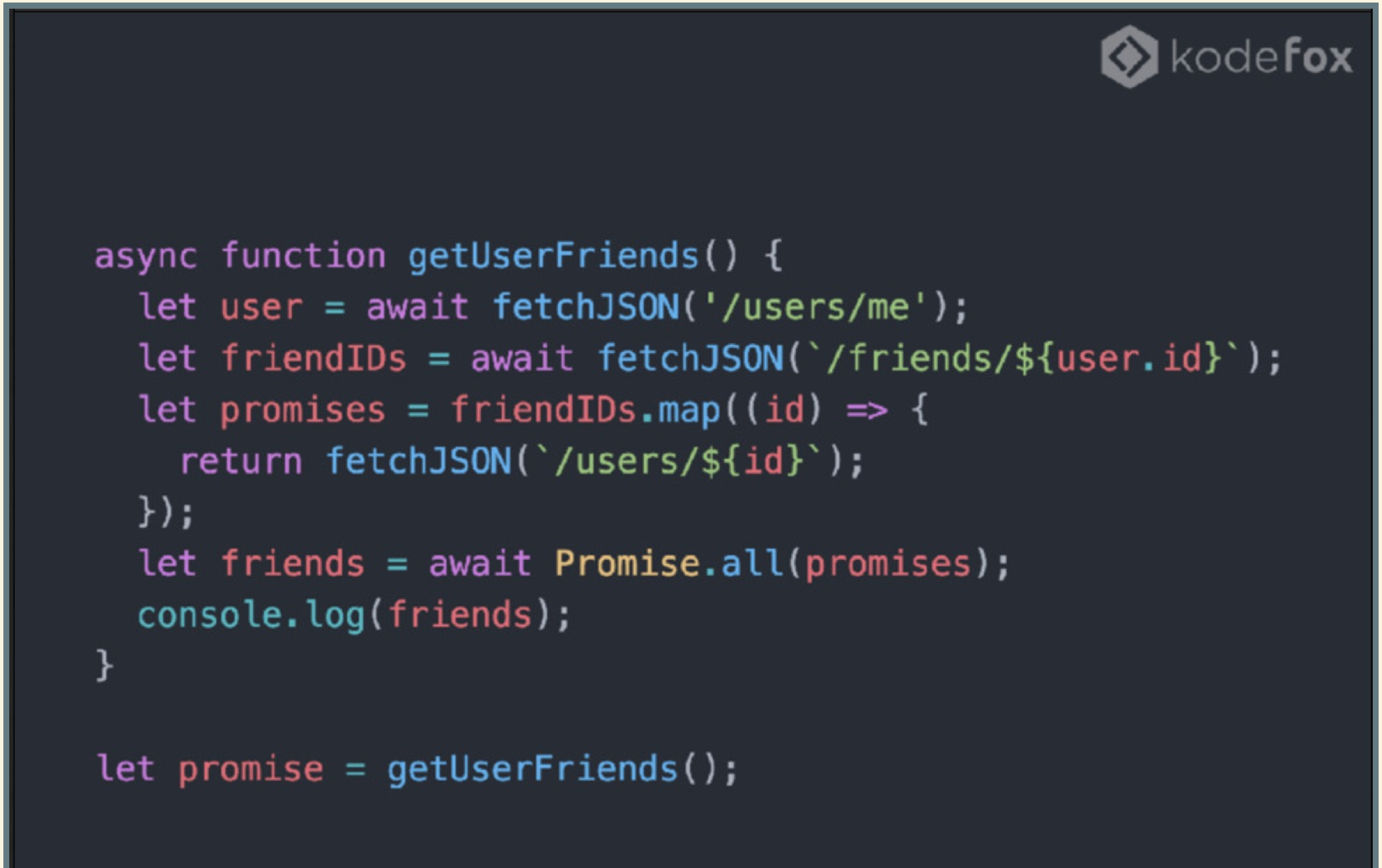


```
// This would be very difficult with callbacks
fetchJSON('/user-profile')
  .then((user) => {
    return fetchJSON(`/users/${user.id}/friends`);
  })
  .then((friendIDs) => {
    let promises = friendIDs.map((id) => {
      return fetchJSON(`/users/${id};`);
    });
    return Promise.all(promises);
  })
  .then((friends) => console.log(friends));
```

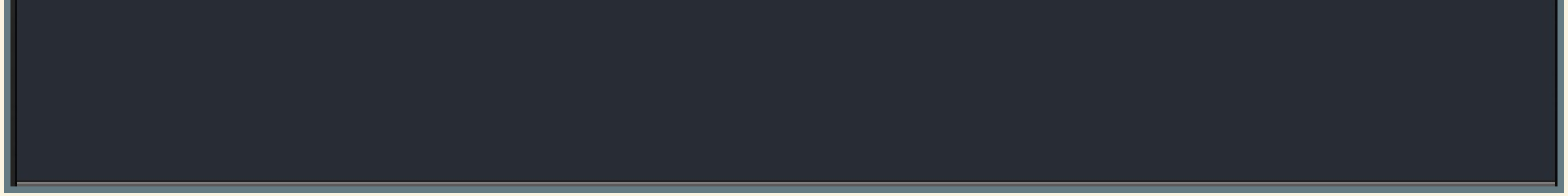


```
  .then((friends) => console.log(friends));
```

ASYNC AWAIT



```
async function getUserFriends() {
  let user = await fetchJSON('/users/me');
  let friendIDs = await fetchJSON(`/friends/${user.id}`);
  let promises = friendIDs.map((id) => {
    return fetchJSON(`/users/${id}`);
  });
  let friends = await Promise.all(promises);
  console.log(friends);
}

let promise = getUserFriends();
```



EXAMPLE FOR RABBITAMQP

RabbitMQ

RABBITMQ IS SOFTWARE THAT SENDS AND RECEIVES MESSAGES USING CHANNELS.

Callback Based

```
#!/usr/bin/env node

var amqp = require('amqplib/callback_api');
var basename = require('path').basename;
var uuid = require('node-uuid');

var n;
try {
  if (process.argv.length < 3) throw Error('Too few args');
  n = parseInt(process.argv[2]);
}
catch (e) {
  console.error(e);
  console.warn('Usage: %s number', basename(process.argv[1]));
  process.exit(1);
}
```

HTTPS://GITHUB.COM/SQUAREMO/AMQP.NODE/BLOB/MASTER/EXAMPLES/TUTORIALS/CALLBACK_API/RPC_CLIENT.JS

Promise Based

```
#!/usr/bin/env node

var amqp = require('amqplib');
var basename = require('path').basename;
var Promise = require('bluebird');
var uuid = require('node-uuid');

// I've departed from the form of the original RPC tutorial, whic
// needlessly introduces a class definition, and doesn't even
// parameterise the request.

var n;
try {
  if (process.argv.length < 3) throw Error('Too few args');
  n = parseInt(process.argv[2]);
}
```

HTTPS://GITHUB.COM/SQUAREMO/AMQP.NODE/BLOB/MASTER/EXAMPLES/TUTORIALS/RPC_CLIENT.JS

Async Await

```
var amqp = require('amqplib')

var open = require('amqplib').connect('amqp://localhost');

const connect = (url = 'amqp://localhost') => {
  return new Promise((resolve, reject) => {
    amqp.connect(url)
      .then(conn => resolve(conn))
      .catch(err => reject(err))
  })
}

const createChannel = conn => {
  return new Promise((resolve, reject) => {
    conn.createChannel()
      .then(channel => resolve(channel))
```

HTTPS://GIST.GITHUB.COM/STANZHENG/788248DE2E32FE50B5495999033007D7

##

```
const connection = async (queueName = 'msg.*') => {
  var conn = await connect()
  var channel = await createChannel(conn)
  var assertedChannelToQueue = await channelAssertQueue(channel,
  return channel
}
```

HTTPS://GIST.GITHUB.COM/STANZHENG/788248DE2E32FE50B5495999033007D7

RECAP

CALLBACK BASED

- Works everywhere
- Pyramid of doom
- Debugging
- Doesn't flow like our brain

PROMISE BASED

- Chainable and easy to follow Flow
- Better error handling with **catch**
- Create and resolve each promise

ASYNC AWAIT

- Get back constructs we know
  - `for loop / do while`
  - `try catch`
- Can make your code **slower**
- Flows up and down (works like our brains)
- May need to be shimmed for some environments

YOUR CODE NOW IS...