JS Event Loop, Promises, Async Await etc

Slava Kim

Synchronous

Happens consecutively, one after another

```
var a = 2;
var b = 2;
console.log(a + b);
console.log("quick maths");
```

Asynchronous

Happens later at some point in time

```
mongoClient.connect(mongoUrl, function (err, db) {
  // do something with db here
});
```

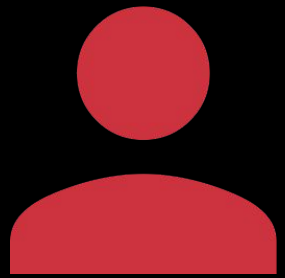
Parallelism vs Concurrency

What are those????

Concurrency - multiple tasks are handled

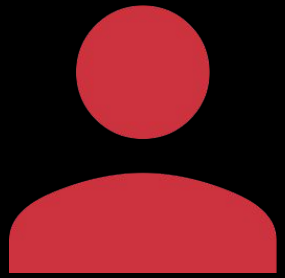
Parallelism - doing multiple tasks at the same time

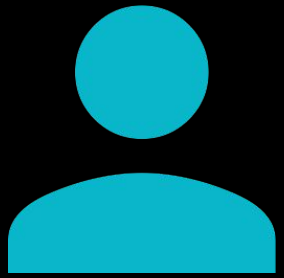
Parallelism vs Concurrency



TA

Parallelism vs Concurrency

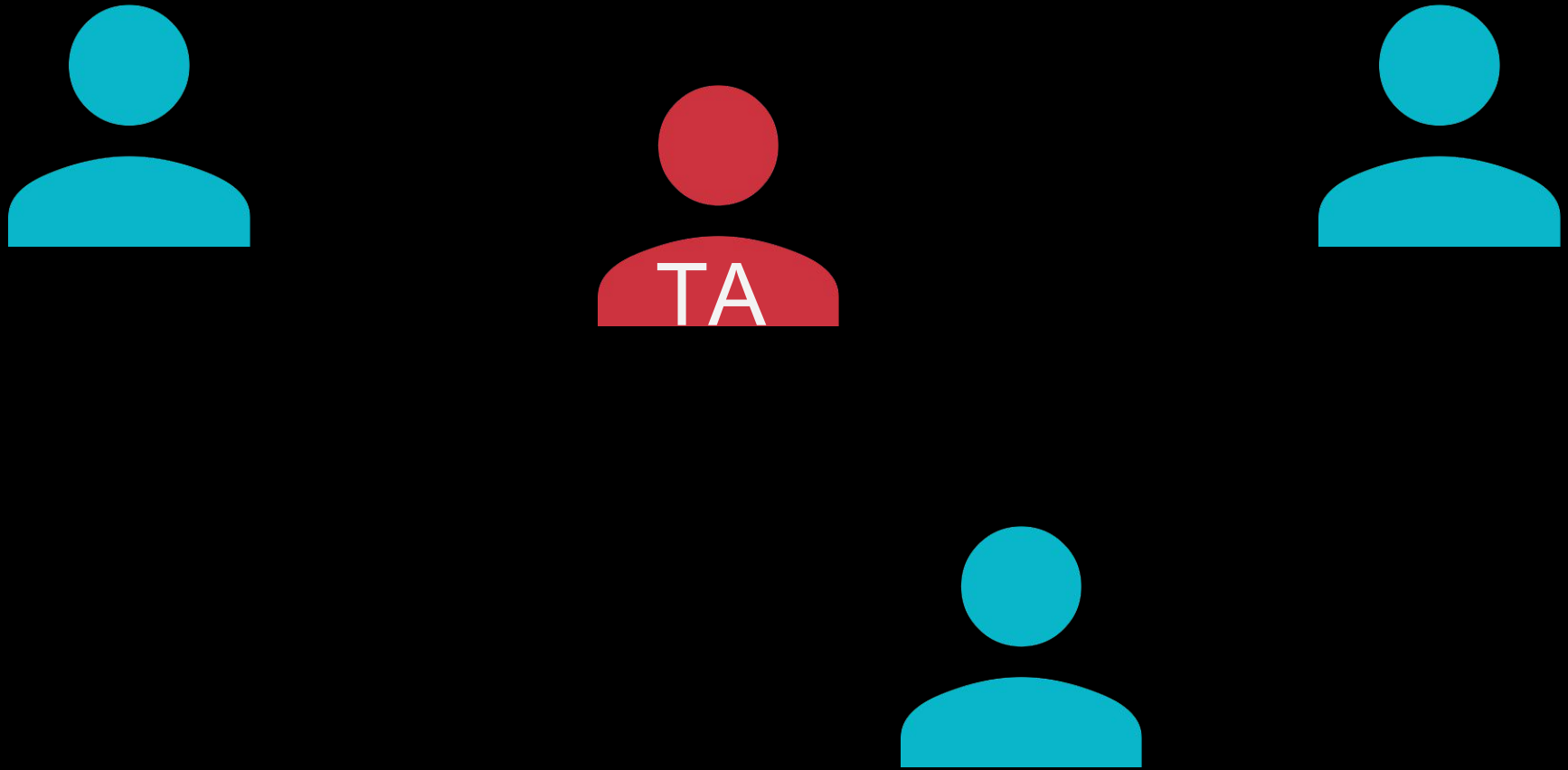




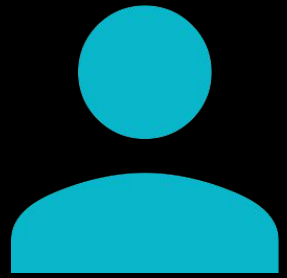
TA

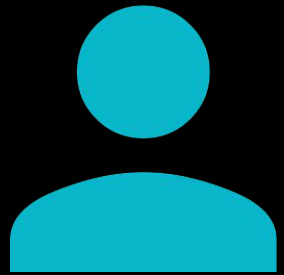
Student

Parallelism vs Concurrency

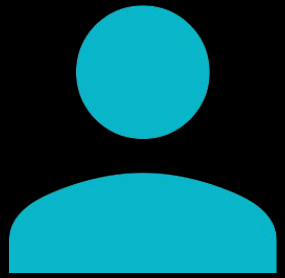


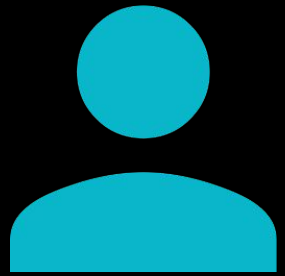
Parallelism vs Concurrency



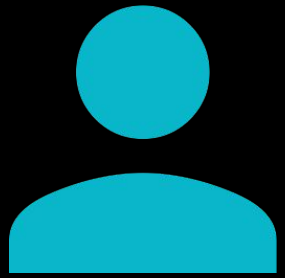


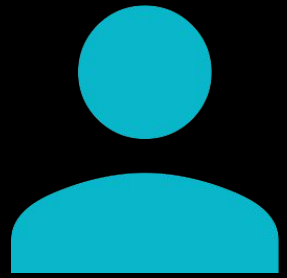
Parallelism vs Concurrency



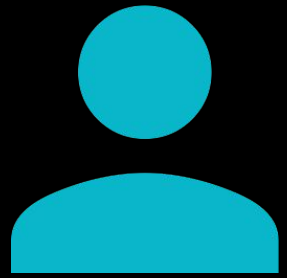


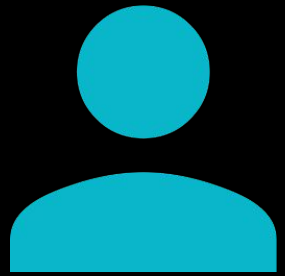
Parallelism vs Concurrency





Parallelism vs Concurrency





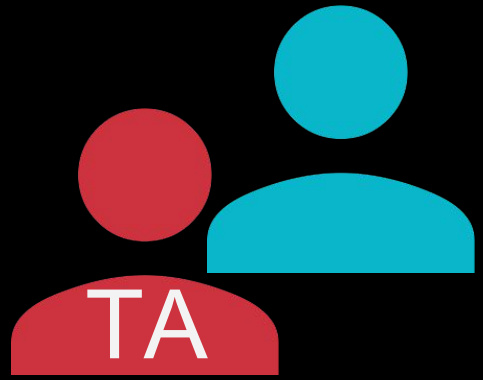
Is TA concurrent or parallel?

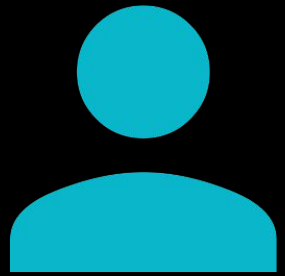
TA is not parallel - there is only one TA

TA is concurrent tho, the TA goes around and is helping 3 students “simultaneously”, but some students need to wait

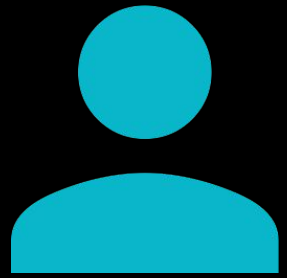
Parallelism vs Concurrency

Parallelism vs Concurrency

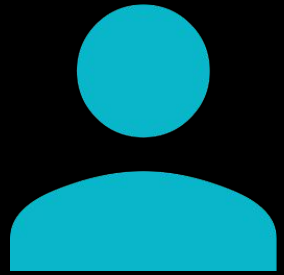




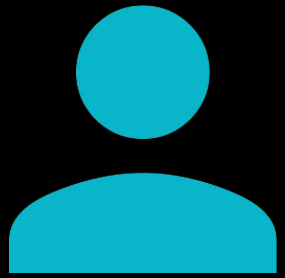
Parallelism vs Concurrency

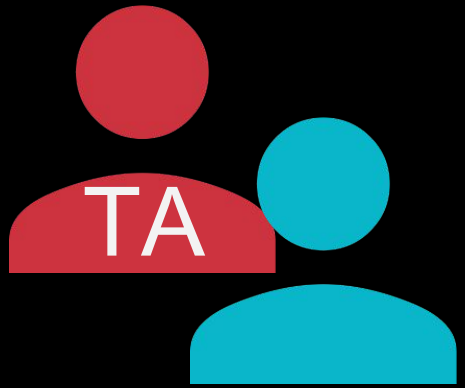


Parallelism vs Concurrency



Parallelism vs Concurrency





Is TA concurrent or parallel?

There are multiple TAs - we can achieve parallelism with concurrency

Still handling multiple students at the same time

CPU analogy

Each TA - one CPU

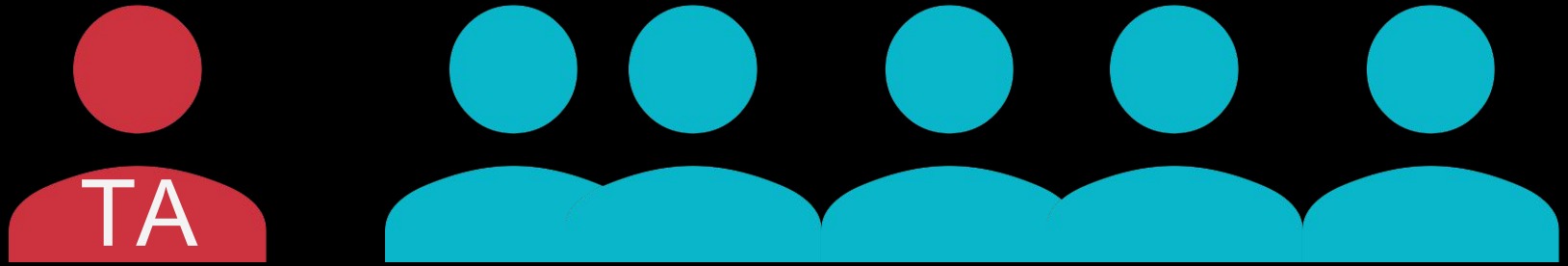
Each student - a separate task

Some programming languages allow you to use multiple cores (C++, Java)

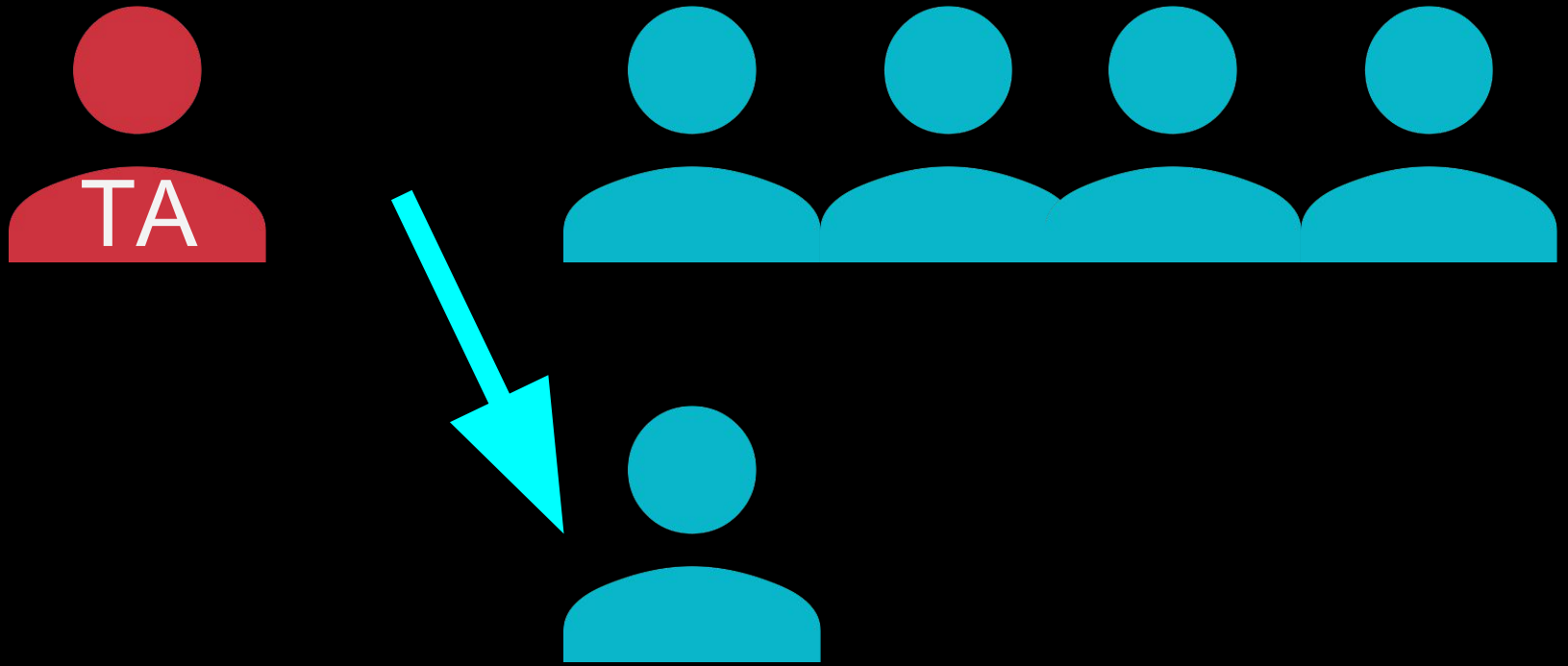
JavaScript is single threaded, can only use one core, but is still concurrent

How?

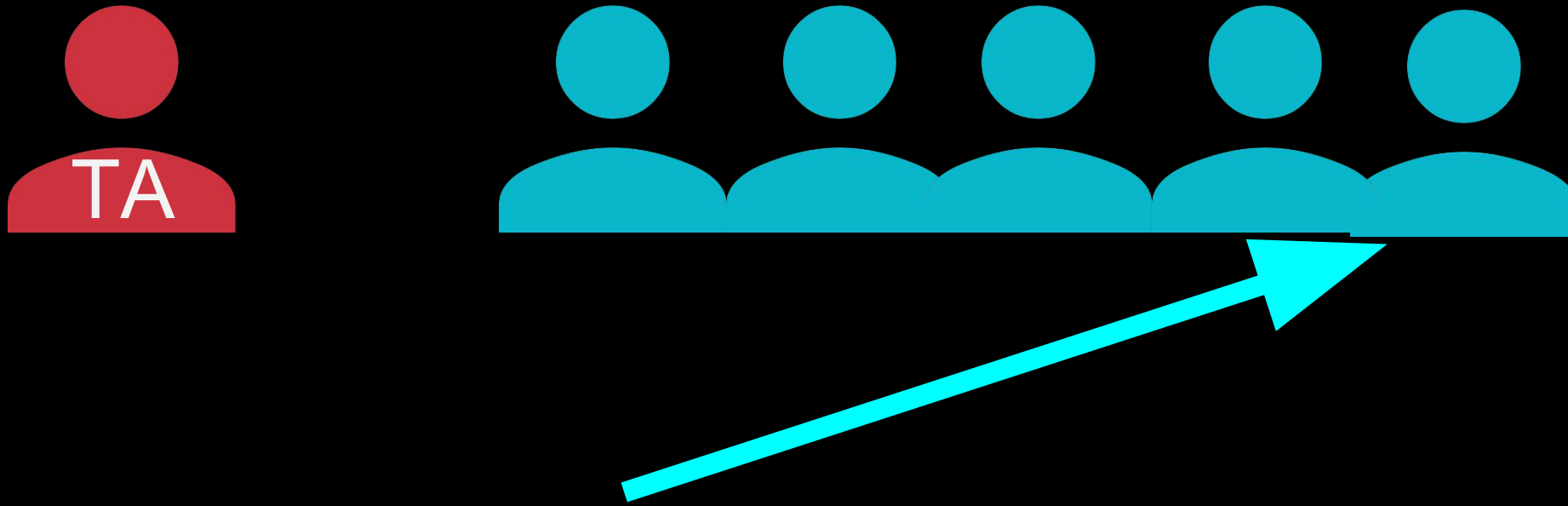
Queue



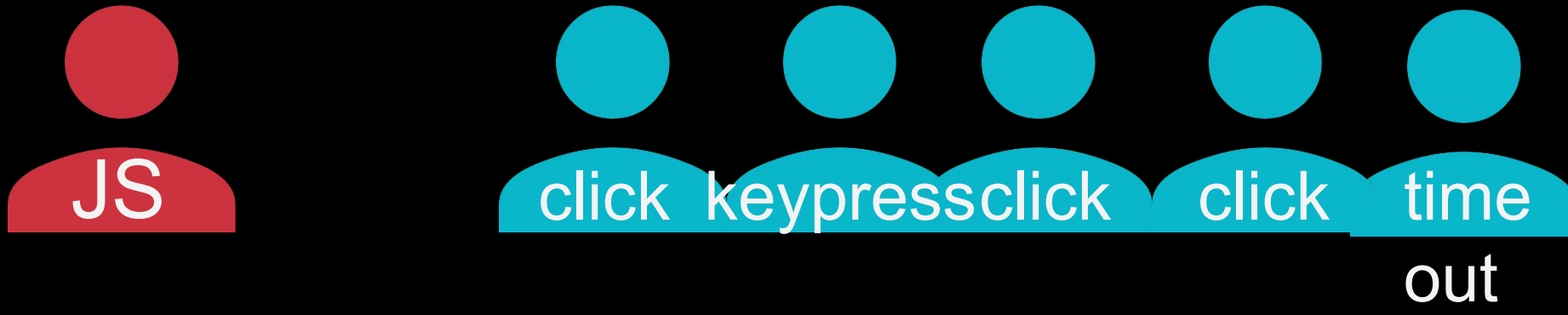
Queue



Queue



Queue



Why callbacks?

```
button.addEventListener('click', function () {
  // do something here
});
```

Why callbacks?

```
button.addEventListener('click', function () {
  // do something here
});
```

Callbacks to handle a result of an operation

```
mongoDbCollection.find({ name: 'Robert' },
  function (err, res) {
    // do something with res here
});
```

Time outs and intervals

```
setTimeout(function () {
  // do something after 2 seconds
}, 2000);
```

Event Loop

Each individual event is handled in some order
New events are queued up in the end

Somewhere in C++ implementation of JS Engine

```
var queue = [];
while (!queue.empty()) {
  event = queue.pop_front();
  process(event);
}
```

Event Loops example

```
console.log('A');
setTimeout(function () {
  console.log('B');
}, 2000);
console.log('C');
```

Event Loops example

```
console.log('A');
setTimeout(function () {
  console.log('B');
}, 2000);
console.log('C');
```

A
C
… (2 second delay)
B

Event Loops example

```
console.log('A');
setTimeout(function () {
  console.log('B');
}, 0);
console.log('C');
```

Event Loops example

```
console.log('A');
setTimeout(function () {
  console.log('B');
}, 0);
console.log('C');
```

A
C
… (virtually no delay)
B

Event Loops: Good

It is a “simple” model to work with to achieve concurrency

No need for locks, or critical sections. Each function is a critical section

Good when you have a lot of I/O work (most web servers, UI systems)

Examples of I/O: talking to database, handling requests, waiting for user to click

Event Loop: bad

CPU intensive operations will bring down your browser/server

Can be confusing for new users

No real way to predict in which order events will happen

Promises - motivation

```
mongoDb.connect(mongoUrl, function (err, db) {
  db.getCollection('collectionName', function (err, col) {
    col.findOne({name: 'Robert'}, function (err, found) {
      if (found) {
        col.update({ _id: found._id }, { $set: { age: 21 } }, function () {
          // do something else
        });
      } else {
        col.insert({ name: 'Robert', age: 21 }, function () {
          // do something else
        });
      }
    });
  });
});
```

Promises - motivation

```
mongoDb.connect(mongoUrl, function (err, db) {
  db.getCollection('collectionName', function (err, col) {
    col.findOne({name: 'Robert'}, function (err, found) {
      if (found) {
        col.update({ _id: found._id }, { $set: { age: 21 } }, function () {
          // do something else
        });
      } else {
        col.insert({ name: 'Robert', age: 21 }, function () {
          // do something else
        });
      }
    });
  });
});
```

Callback hell!

Possible solution with promises

```
mongoDb.connect(mongoUrl)
  .then((db) => {
    retuirn db.getCollection('collectionName');
  })
  .then((col) => {
    return col.findOne({name: 'Robert'});
  })
  .then(found) => {
    if (found) {
        return col.update({ _id: found._id }, { $set: { age: 21 } });
      } else {
        return col.insert({ name: 'Robert', age: 21 });
      }
  })
  .then(() => {
    // do something
  })
  .catch((err) => {
    // report error
  });
```

Only one additional level of nesting

Error handling can be grouped together

Only one branch of continuation

Possible solution with promises

```
mongoDb.connect(mongoUrl)
  .then((db) => {
    retuirn db.getCollection('collectionName');
  })
  .then((col) => {
    return col.findOne({name: 'Robert'});
  })
  .then(found) => {
    if (found) {
        return col.update({ _id: found._id }, { $set: { age: 21 } });
      } else {
        return col.insert({ name: 'Robert', age: 21 });
      }
  })
  .then(() => {
    // do something
  })
  .catch((err) => {
    // report error
  });
```

Only one additional level of nesting

Error handling can be grouped together

Only one branch of continuation

Promises - await all

```
var promise1 = request('http://answer.com/?q=universe');
var promise2 = request('http://wikipedia.com/answerToUniverse');
var promise3 = request('http://webdevelopment.mit.edu/answersssss');

var promises = [promise1, promise2, promise3];
```

Promises - await all

```
var promise1 = request('http://answer.com/?q=universe');
var promise2 = request('http://wikipedia.com/answerToUniverse');
var promise3 = request('http://webdevelopment.mit.edu/answersssss');

var promises = [promise1, promise2, promise3];

Promise.all(promises, function (err, allResults) {
  console.log(allResults);
})
```

Promises - race

```
var promise1 = request('http://answer.com/?q=universe');
var promise2 = request('http://wikipedia.com/answerToUniverse');
var promise3 = request('http://webdevelopment.mit.edu/answersssss');

var promises = [promise1, promise2, promise3];

Promise.race(promises, function (err, firstResult) {
  console.log(firstResult);
})
```

Promises - availability

Available in all modern browsers (rip IE)

Available in latest Node.js

Works with MongoDB client

Might need to wrap other libraries into promises interface with “promisify”

Makes code cleaner and easier to manage, yay!

Your interviewers will impressed

Async/Await

New feature in the latest spec of JavaScript

Can await on anything that returns a promise

Is not yet available widely without special setup

Makes code even nicer

```
mongoDb.connect(mongoUrl, function (err, db) {
  db.getCollection('collectionName', function (err, col) {
    col.findOne({name: 'Robert'}, function (err, found) {
      if (found) {
        col.update({ _id: found._id }, { $set: { age: 21 } }, function () {
          // do something else
        });
      } else {
        col.insert({ name: 'Robert', age: 21 }, function () {
          // do something else
        });
      }
    });
  });
});
```

Callback hell!

Possible solution with promises

```
mongoDb.connect(mongoUrl)
  .then((db) => {
    retuirn db.getCollection('collectionName');
  })
  .then((col) => {
    return col.findOne({name: 'Robert'});
  })
  .then(found) => {
    if (found) {
        return col.update({ _id: found._id }, { $set: { age: 21 } });
      } else {
        return col.insert({ name: 'Robert', age: 21 });
      }
  })
  .then(() => {
    // do something
  })
  .catch((err) => {
    // report error
  });
```

Solution with Async/Await

```
try{
  const db = await mongoDb.connect(mongoUrl);
  const col = await db.getCollection('collectionName');
  const found = await col.findOne({name: 'Robert'});
  if (found) {
    await col.update({ _id: found._id }, { $set: { age: 21 } })
  } else {
    await col.insert({ name: 'Robert', age: 21 });
  }

  // do something
} catch (err) {
  // report error
}
```

No extra indentation or nesting at all

Code “reads” synchronous but actually is async

Handle errors with try/catch construct just like synchronous code

Solution with Async/Await

```
try{
  const db = await mongoDb.connect(mongoUrl);
  const col = await db.getCollection('collectionName');
  const found = await col.findOne({name: 'Robert'});
  if (found) {
    await col.update({ _id: found._id }, { $set: { age: 21 } })
  } else {
    await col.insert({ name: 'Robert', age: 21 });
  }

  // do something
} catch (err) {
  // report error
}
```

No extra indentation or nesting at all

Code “reads” synchronous but actually is async

Handle errors with try/catch construct just like synchronous code

Why don’t we use it now?

To make it work, you need to setup a compiler from “newest JS” to “old JS”

Every function that makes use of “await” needs to be marked “async”

Can make code difficult to follow

```
// will return a promise
async function doThings() {
  const result = await someAsyncFunction();
}
```

But you still can use it

Supported in latest node, latest Chrome and latest Firefox

Probably will break on a lot of other places

[demo]