
boloco
boston local company
569
boloco
will you be
or nice?

I promise you your burritos will be delivered by 12:30pm

I promise you your burritos will be delivered by 12:30pm

```
deliveryPromise().then((food) => {
    console.log(`We have ${food}, let's eat lunch!`);
}).catch((err) => {
    console.log("Tony, stall for 30 minutes, we don't have food yet!");
})
```

I promise you your burritos will be delivered by 12:30pm

Woopsies

`Promise.reject(new Error('fail'))` 😈

```
deliveryPromise().then((food) => {
    console.log(`We have ${food}, let's eat lunch!`);
}).catch((err) => {
    console.log("Tony, stall for 30 minutes, we don't have food yet!");
})
```

I promise you your burritos will be delivered by 12:30pm

Lunch at 1:00 pm today

```
deliveryPromise().then((food) => {
    console.log(`We have ${food}, let's eat lunch!`);
}).catch((err) => {
    console.log("Tony, stall for 30 minutes, we don't have food yet!");
})
```

Async Javascript

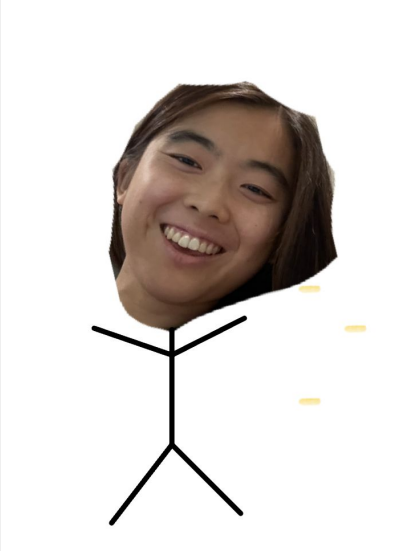
Tony Cui and Jay Hilton

Synchronous

Happens consecutively, one after another

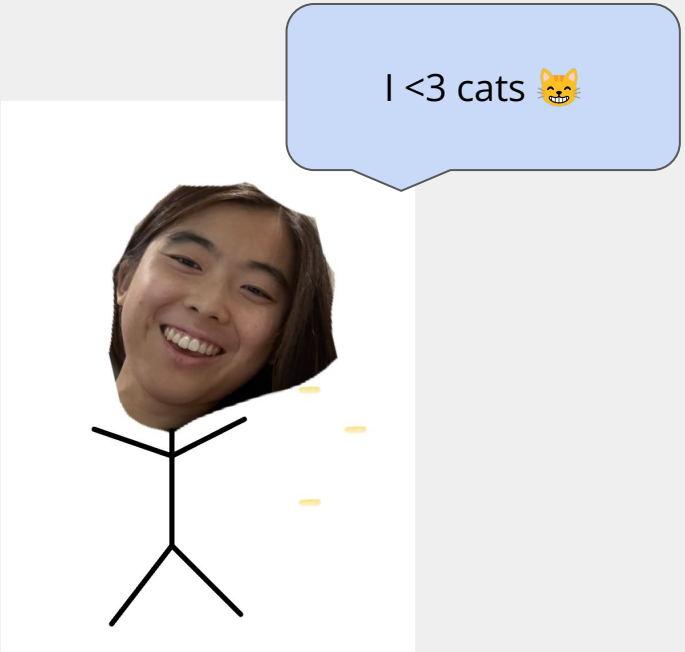
Asynchronous

Multiple processes can run at the same time



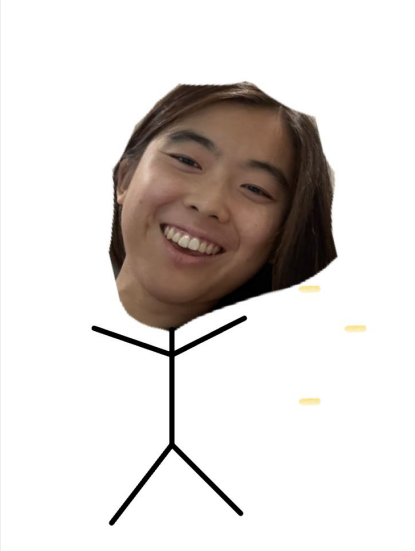
cat lover 😍

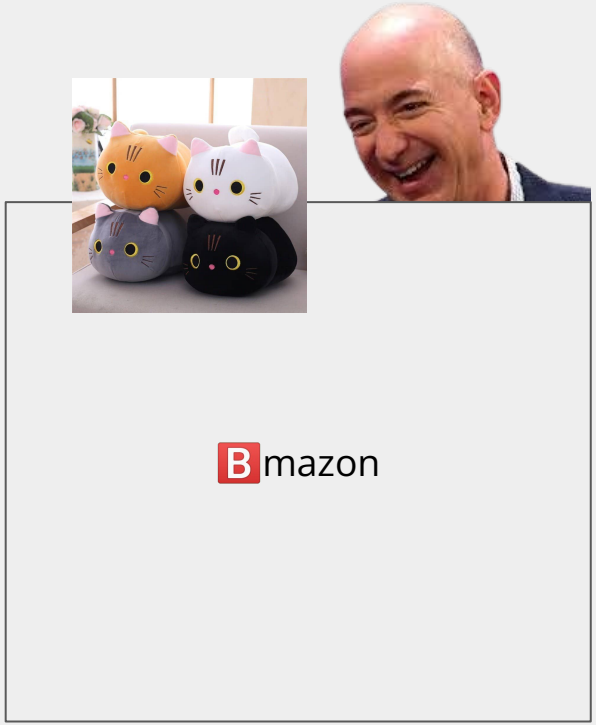
Catalogue 🥰



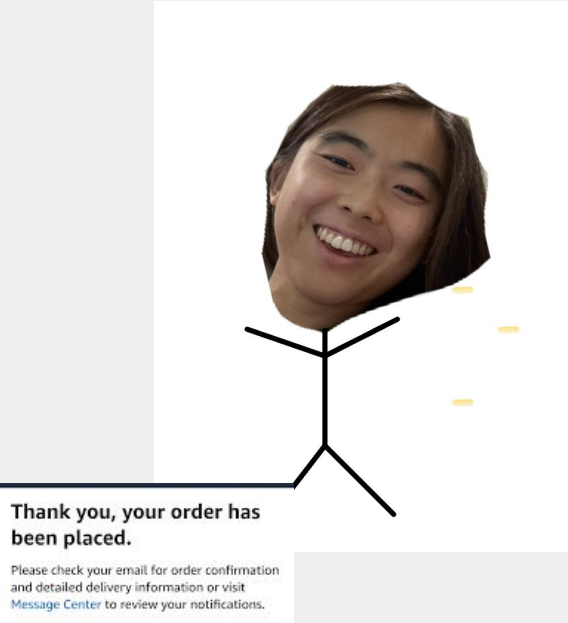
cat lover 😍

Catalogue 🥰





cat lover 😍

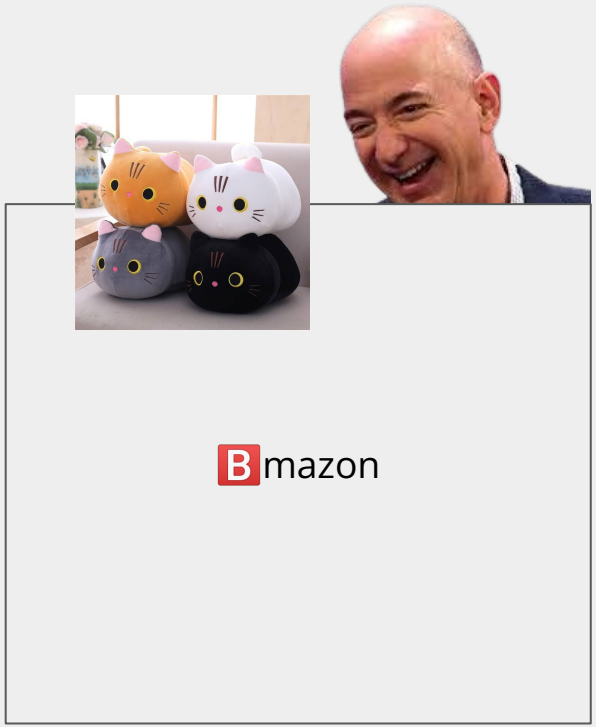


cat lover 😍

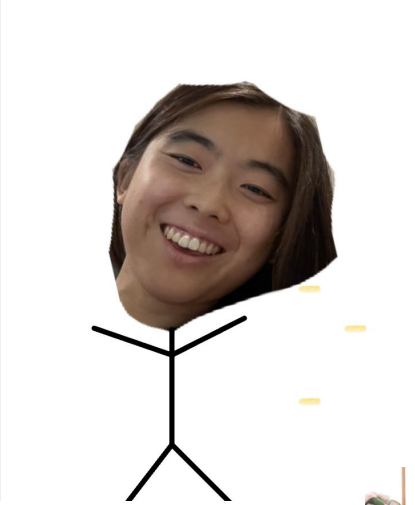
Thank you, your order has been placed.

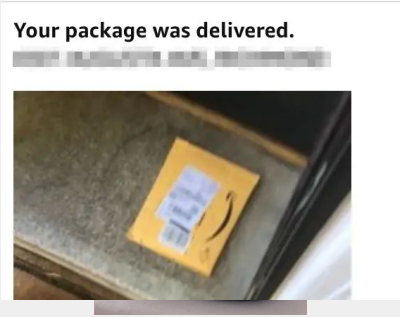
Please check your email for order confirmation and detailed delivery information or visit Message Center to review your notifications.

Place an Order (POST)



Scenario 1: Package Delivered!

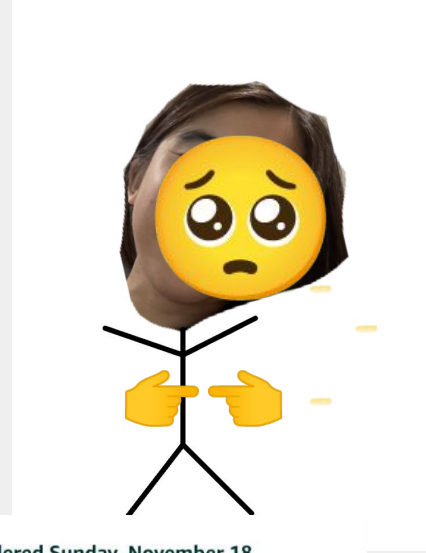




Package Delivered!

Bmazon

Scenario 2: Still waiting for package...



Ordered Sunday, November 18

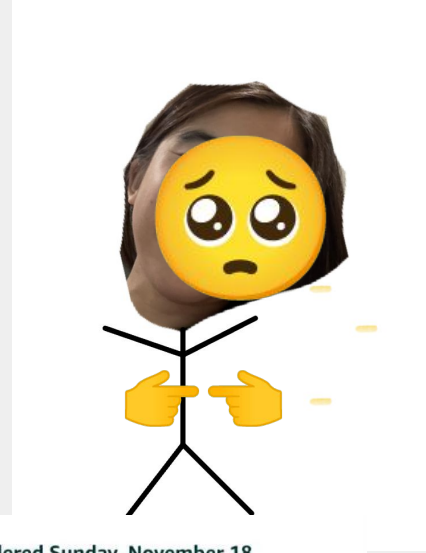
Shipped

Your package is still on the way, but it's running late. Now expected September 21 - September 22 — most packages arrive in a day.

On its way

Bmazon

Scenario 2: Still waiting for package...



Ordered Sunday, November 18

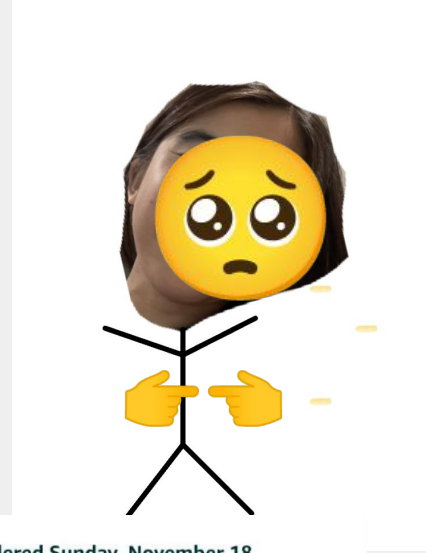
Shipped

Your package is still on the way, but it's running late. Now expected September 21 - September 22 — most packages arrive in a day.

On its way

Bmazon

Scenario 2: Still waiting for package...



Ordered Sunday, November 18

Shipped

Your package is still on the way, but it's running late. Now expected September 21 - September 22 — most packages arrive in a day.

On its way

Bmazon

Scenario 2: Still waiting for package...

Ordered Sunday, November 18

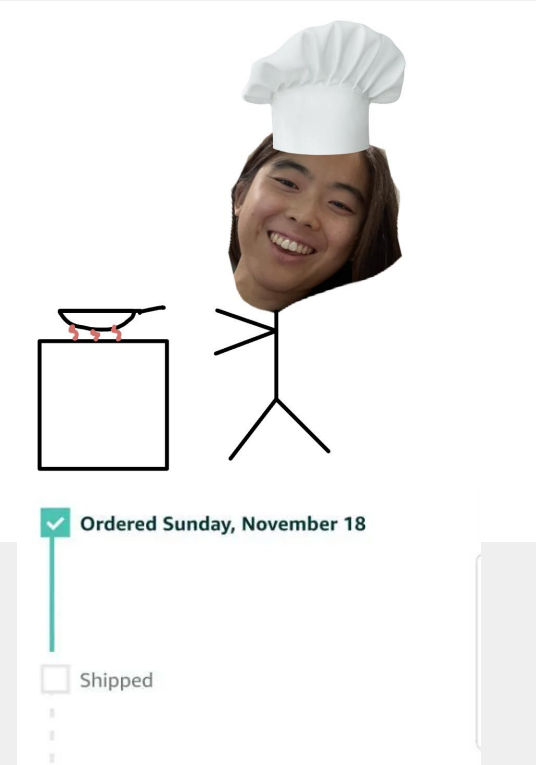
Shipped

Your package is still on the way, but it's running late. Now expected September 21 - September 22 — most packages arrive in a day.

On its way

Bmazon

Scenario 2: Still waiting for package...

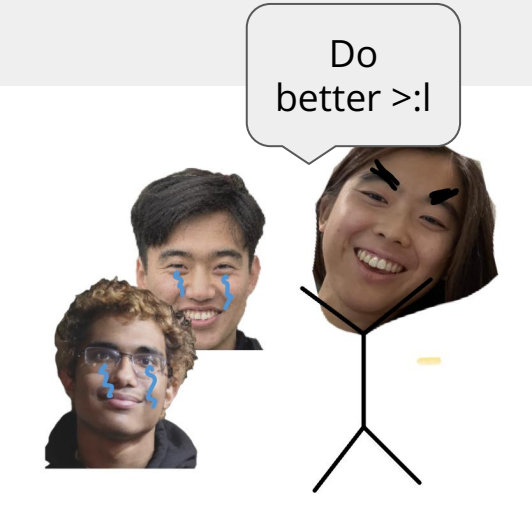


Your package is still on the way, but it's running late. Now expected September 21 - September 22 — most packages arrive in a day.

On its way

Bmazon

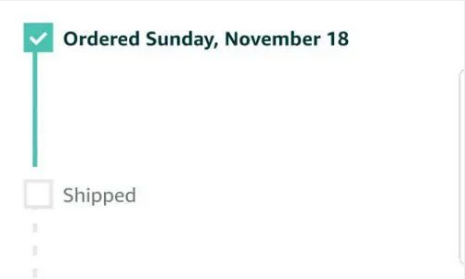
Scenario 2: Still waiting for package...



Your package is still on the way, but it's running late. Now expected September 21 - September 22 — most packages arrive in a day.

On its way

Bmazon



Scenario 2: Still waiting for package...

Your package is still on the way, but it's running late. Now expected September 21 - September 22 — most packages arrive in a day.

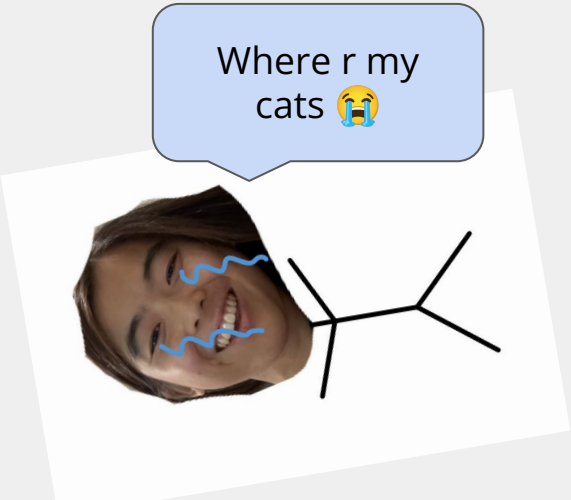
On its way

Bmazon

Ordered Sunday, November 18

Shipped

Scenario 3: ????

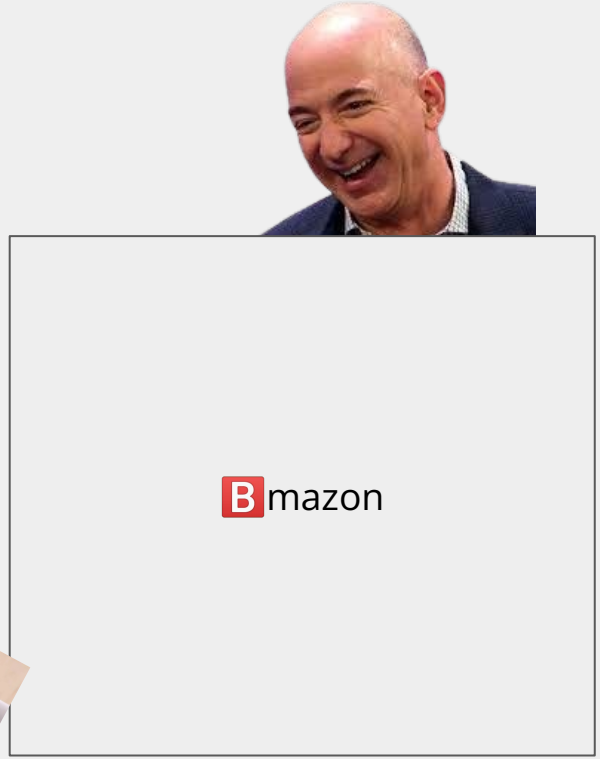


Your package may be lost

Packages are rarely this late and we're sorry yours still hasn't arrived

You can wait another couple days or check out these options.

????



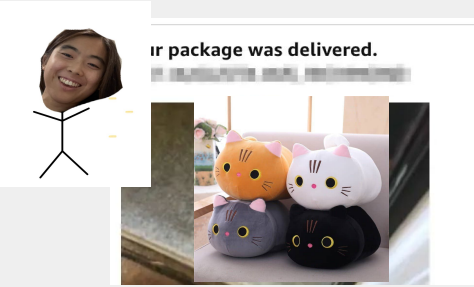
Analogy to promises

After placing a delivery order, or creating a promise, they will have one of three statuses:

Analogy to promises

After placing a delivery order, or creating a promise, they will have one of three statuses:

- Fulfilled (resolved)
  - Delivery Successful! 😸

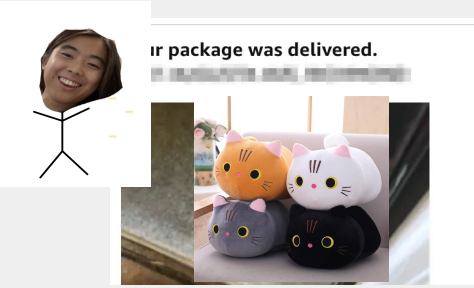


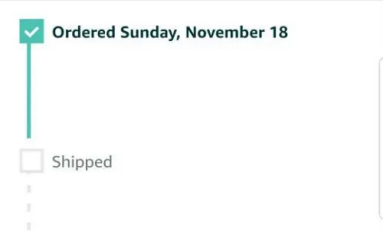
Analogy to promises

After placing a delivery order, or creating a promise, they will have one of three statuses:

- Fulfilled (resolved)
  - Delivery Successful! 😸

- Pending
  - Be patient mam 😒

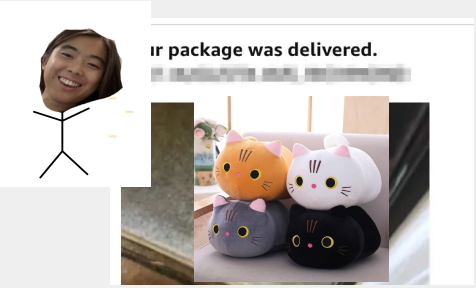


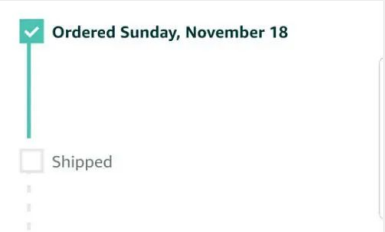


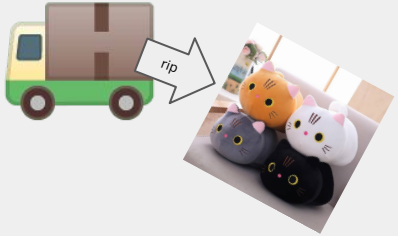
Analogy to promises

After placing a delivery order, or creating a promise, they will have one of three statuses:

- Fulfilled (resolved)
  - Delivery Successful! 😸
- Pending
  - Be patient maam 😒
- Rejected
  - Something went wrong 😭







.then() review

```
useEffect(() => {
  post("/api/orderkitties");
}, []);
```

Initially, the get requests will return a Promise object, which can be Pending, Fulfilled, or Rejected.

.then() review

```
useEffect(() => {
  post("/api/orderkitties");
}, []);
```

Initially, the requests will return a Promise object, which can be Pending, Fulfilled, or Rejected.

.then() review

```
useEffect(() => {
  post("/api/orderkitties").then((deliveredCats)=>{
    hugFunction(deliveredCats)
  })
}, []);
```

Once the promise is **fulfilled**, do stuff (call a callback function).
Returns a promise.

.then() review

```
useEffect(() => {
  post("/api/orderkitties").then((deliveredCats)=>{
    hugFunction(deliveredCats)
  }).catch((catsFellOffTruckError) =>{
    console.log("oh no 🥺 :", catsFellOffTruckError)
  })
}, []);
```

Once the promise is **rejected**, do stuff (call a callback function).
Returns a promise.

.then() review

```
useEffect(() => {
  get("/api/stories").then((storyObjs) => {
    setStories(storyObjs);
  });
}, []);
```

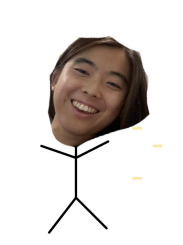
Once the promise is **fulfilled**, do stuff (call a callback function).
Returns a promise.

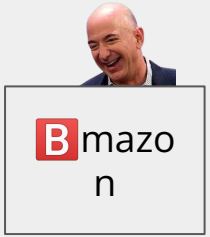
.catch() review

```
useEffect(() => {
  get("/api/stories").then((storyObjs) => {
    setStories(storyObjs);
  }).catch((err) => {
    console.log("this is so sad: ", err.message);
  });
}, []);
```

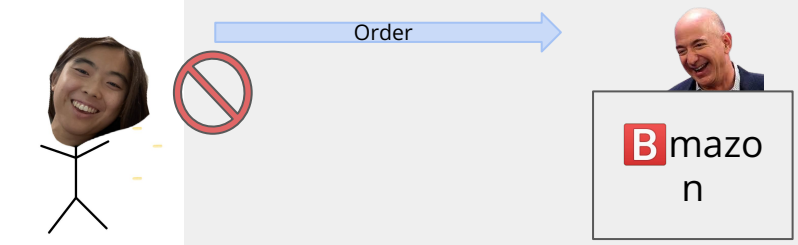
Once the promise is **rejected**, do stuff (call a callback function). Returns a promise.

Synchronous Cat Lover

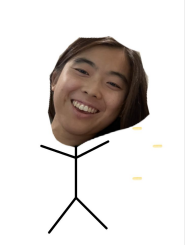




Synchronous Cat Lover

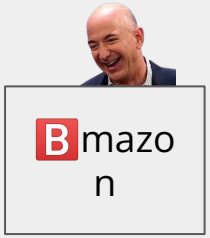


Synchronous Cat Lover

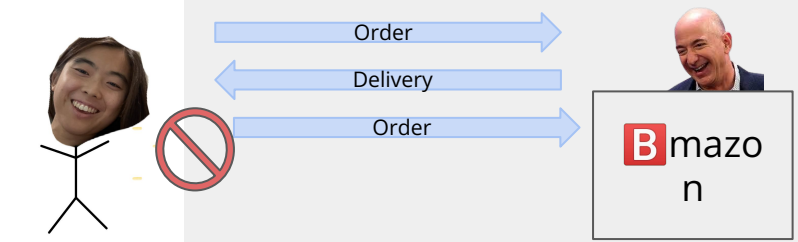


Order

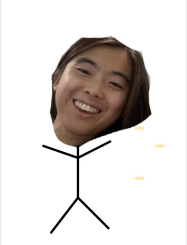
Delivery



Synchronous Cat Lover



Synchronous Cat Lover



Order

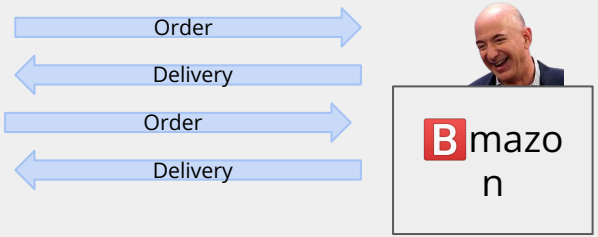
Delivery

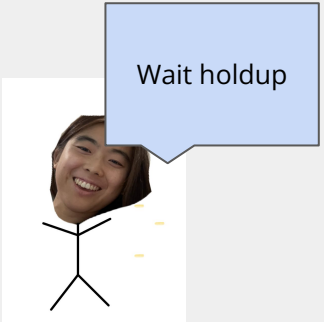
Order

Delivery

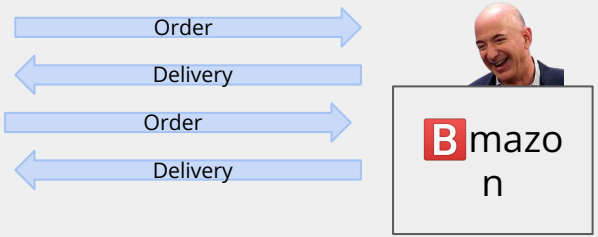
Bmazon

Synchronous Cat Lover

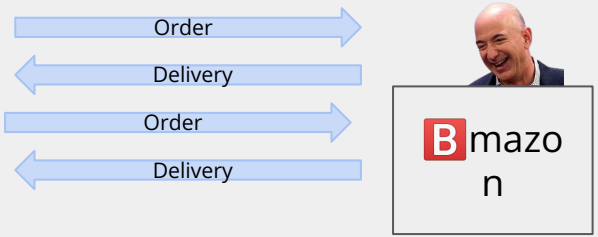


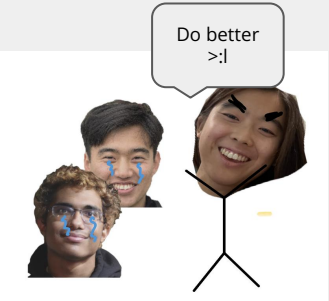


Synchronous Cat Lover

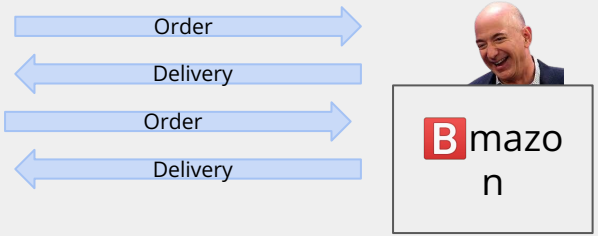


Synchronous Cat Lover

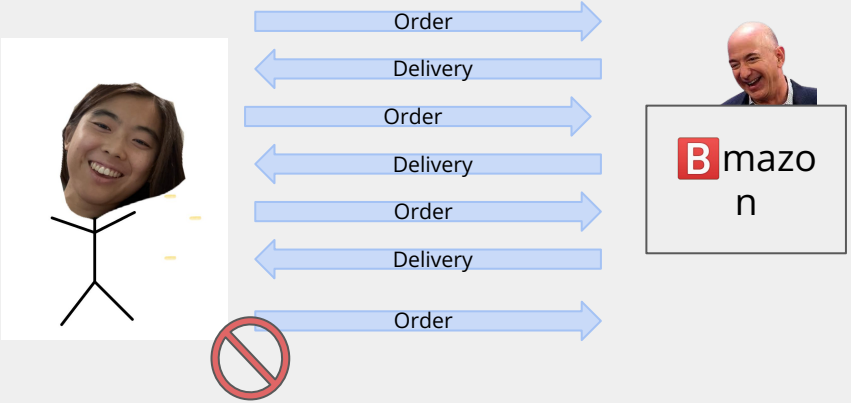




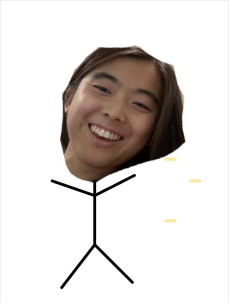
Synchronous Cat Lover



Synchronous Cat Lover



Synchronous Cat Lover



Order

Delivery

Order

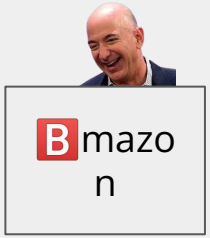
Delivery

Order

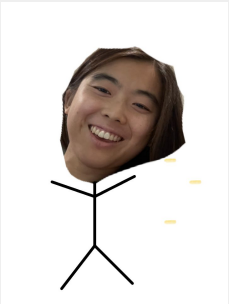
Delivery

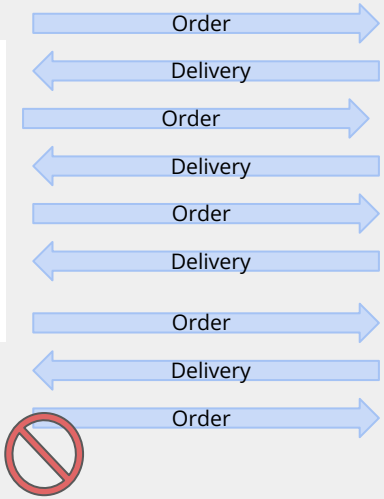
Order

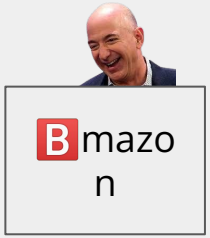
Delivery



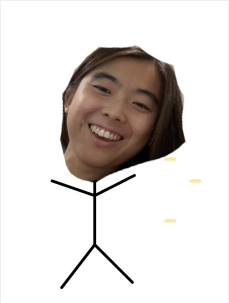
Synchronous Cat Lover







Synchronous Cat Lover



Order

Delivery

Order

Delivery

Order

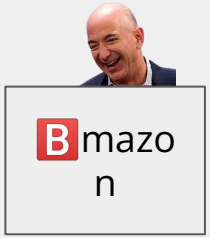
Delivery

Order

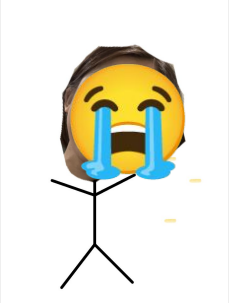
Delivery

Order

Delivery



Synchronous Cat Lover



Order

Delivery

Order

Delivery

Order

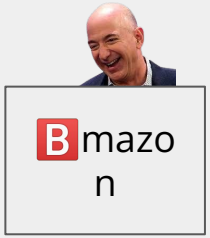
Delivery

Order

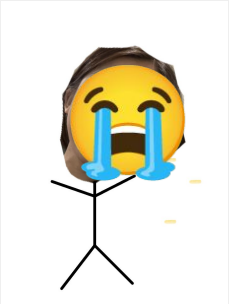
Delivery

Order

Delivery



Synchronous Cat Lover



Order

Delivery

Order

Delivery

Order

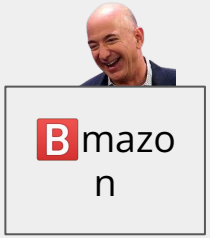
Delivery

Order

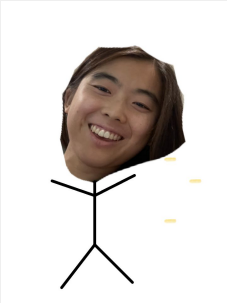
Delivery

Order

Delivery



Asynchronous Cat Lover

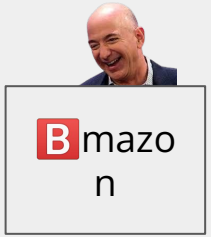


Order

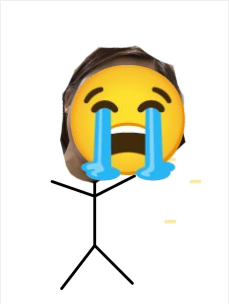
Order

Order

Order



Synchronous Cat Lover



Order

Delivery

Order

Delivery

Order

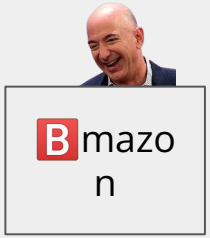
Delivery

Order

Delivery

Order

Delivery



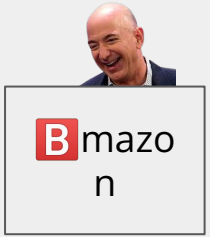
Asynchronous Cat Lover

Order

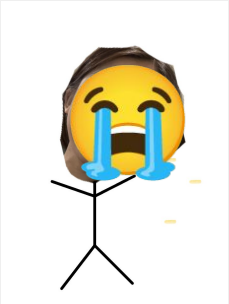
Order

Order

Order



Synchronous Cat Lover



Order

Delivery

Order

Delivery

Order

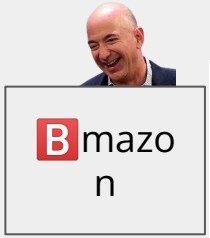
Delivery

Order

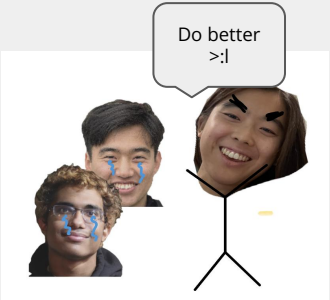
Delivery

Order

Delivery



Asynchronous Cat Lover



Order

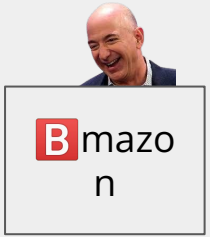
Order

Order

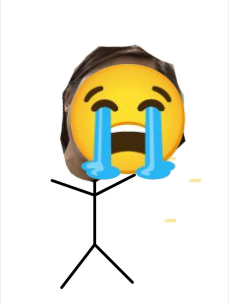
Order

Delivery

Delivery



Synchronous Cat Lover



Order

Delivery

Order

Delivery

Order

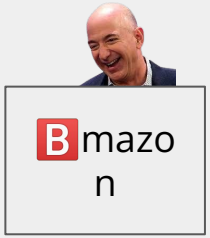
Delivery

Order

Delivery

Order

Delivery



Asynchronous Cat Lover

Order

Order

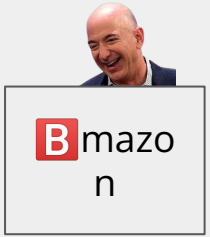
Order

Order

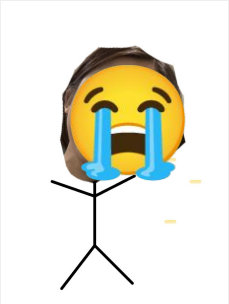
Delivery

Delivery

Delivery



Synchronous Cat Lover



Order

Delivery

Order

Delivery

Order

Delivery

Order

Delivery

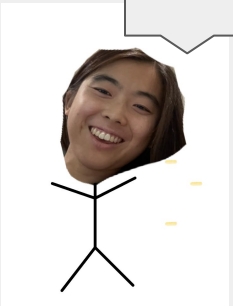
Order

Delivery

Bmazon

Asynchronous Cat Lover

Wow im so productive 😸



Order

Order

Order

Order

Delivery

Delivery

Delivery

Delivery

Bmazon

Just like Promises stop cat lover from wasting time watching her orders, Promises let us keep running our other code

Just like Promises stop cat lover from wasting watching her orders, Promises let us keep running our other code

Asynchronously placed a cat order

```
useEffect(() => {
  slowCatOrder().then((cats)=>{
    console.log("cats delivered",cats)
  })
```

Just like Promises stop cat lover from wasting watching her orders, Promises let us keep running our other code

Asynchronously placed a cat order

```
useEffect(() => {
  slowCatOrder().then((cats)=>{
    console.log("cats delivered",cats)
  })
```

Callback function of what t
do when the order is
fulfilled

Just like Promises stop cat lover from wasting watching her orders, Promises let us keep running our other code

Asynchronously placed a cat order

```
useEffect(() => {
  slowCatOrder().then((cats)=>{
    console.log("cats delivered",cats)
  })
  console.log("go cook sum good food")
  console.log("go lecture the lecturers")
  console.log("go get gains")
}, []);
```

Callback function of what to do when the order is fulfilled

Do some other synchronous tasks

```
useEffect(() => {
  slowCatOrder().then((cats)=>{
    console.log("cats delivered",cats)
  })
  console.log("go cook sum good food")
  console.log("go lecture the lecturers")
  console.log("go get gains")
}, []);
```

```
go cook sum good food
go lecture the lecturers
go get gains
cats delivered fluffy cat, red cat, blue cat
```

Making our API calls asynchronous allows the rest of our code to run while we wait for the response

```
useEffect(() => {
  get("/api/stories").then((storyObjs) => {
    setStories(storyObjs);
  });
}, []);
```

Chaining promises

.then() returns a promise, so we can do .then() again, and again, and again... (same goes for .catch())

```
getPromise().then((value) => {
    console.log("first promise resolved, let's do some stuff");
}).then((value) => {
    console.log("second promise resolved, let's do more stuff");
}).then((value) => {
    console.log("third promise resolved, :)");
}).catch((err) => {
    console.log("oops i am sad now :(")
});
```

Using multiple promises

You can’t compute with pending promises

```
const a = slowNumber(9);
const b = slowNumber(10);

console.log(a + b);
```

(slowNumber(x) returns the number x after 1 second)

JS doesn’t wait for the promises to resolve before continuing.

```
[object Promise][object Promise]
```

JS doesn’t know what a and b are when it does this addition. It just sees 2 pending promises.

To make this work with .then():

```
a.then((aVal) => {
  b.then((bVal) => {
    console.log(aVal + bVal);
  });
});
```

Many promises

```
1   const promise1 = get('/api/comments', { parent: parentId1 });
2   const promise2 = get('/api/comments', { parent: parentId2 });
3   const promise3 = get('/api/comments', { parent: parentId3 });
4   const promise4 = get('/api/comments', { parent: parentId4 });
5   const promise5 = get('/api/comments', { parent: parentId5 });
6
7   const promises = [promise1, promise2, promise3, promise4, promise5];
```

Promise.all()

```
1   const promise1 = get('/api/comments', { parent: parentId1 });
2   const promise2 = get('/api/comments', { parent: parentId2 });
3   const promise3 = get('/api/comments', { parent: parentId3 });
4   const promise4 = get('/api/comments', { parent: parentId4 });
5   const promise5 = get('/api/comments', { parent: parentId5 });
6
7   const promises = [promise1, promise2, promise3, promise4, promise5];
8
9   Promise.all(promises).then((allResults) => {
10    // All results represents a list with the result of each promise
11  }).catch((err) => {
12    // Catch and report any error
13  });
```

Returns a promise that resolves to array of results of input promises

Promise.race()

```
1   const promise1 = get('/api/comments', { parent: parentId1 });
2   const promise2 = get('/api/comments', { parent: parentId2 });
3   const promise3 = get('/api/comments', { parent: parentId3 });
4   const promise4 = get('/api/comments', { parent: parentId4 });
5   const promise5 = get('/api/comments', { parent: parentId5 });
6
7   const promises = [promise1, promise2, promise3, promise4, promise5];
8
9   Promise.race(promises).then((firstResult) => {
10    // Do something with the first result
11  }).catch((err) => {
12    // Catch and report any error
13  });
```

Returns a promise that fulfills or rejects with the first promise that fulfils or rejects

Promise.any()

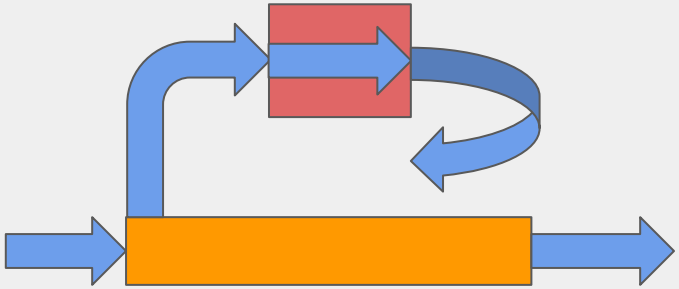
```
1   const promise1 = get('/api/comments', { parent: parentId1 });
2   const promise2 = get('/api/comments', { parent: parentId2 });
3   const promise3 = get('/api/comments', { parent: parentId3 });
4   const promise4 = get('/api/comments', { parent: parentId4 });
5   const promise5 = get('/api/comments', { parent: parentId5 });
6
7   const promises = [promise1, promise2, promise3, promise4, promise5];
8
9   Promise.any(promises).then((anyResult) => {
10    // Do something with the any result regardless if all others fail
11  }).catch((err) => {
12    // Catch and report any error
13  });
```

Returns a promise that resolves when any of the input promises fulfills

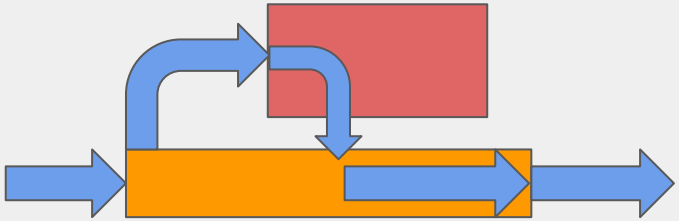
Async / Await

Asynchronous functions

- Functions that return control back to the caller before computation is done

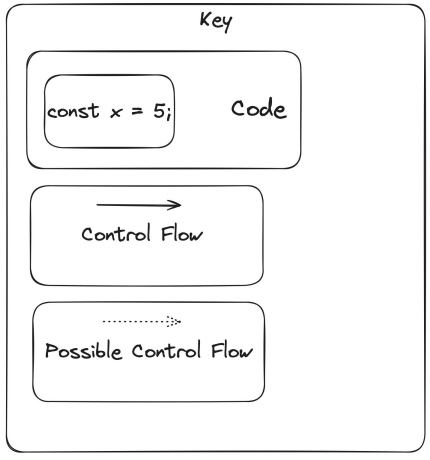


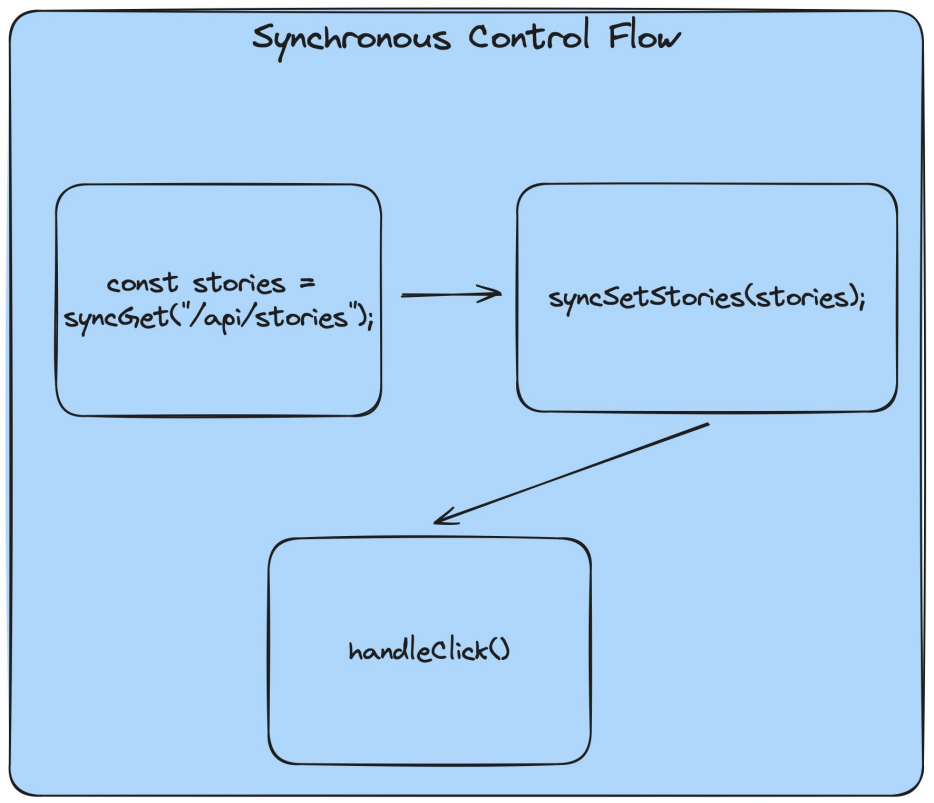
synchronous



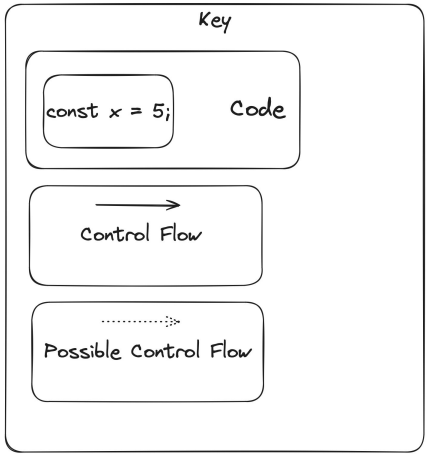
asynchronous

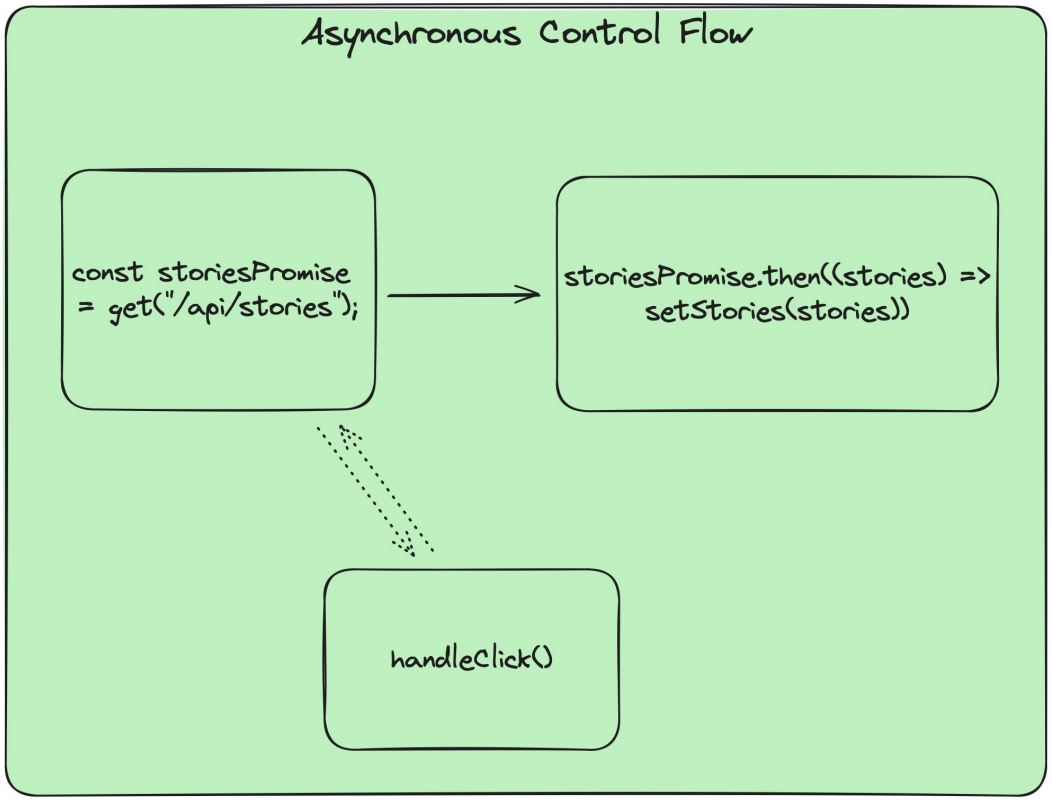
Synchronous Control Flow



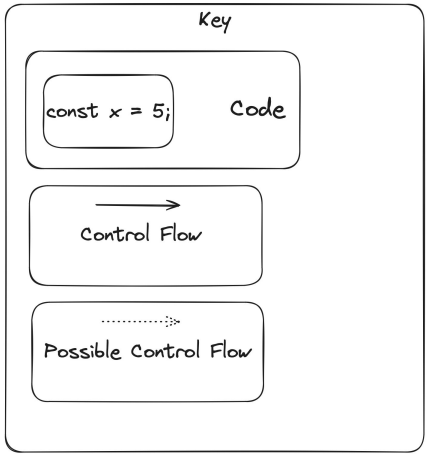


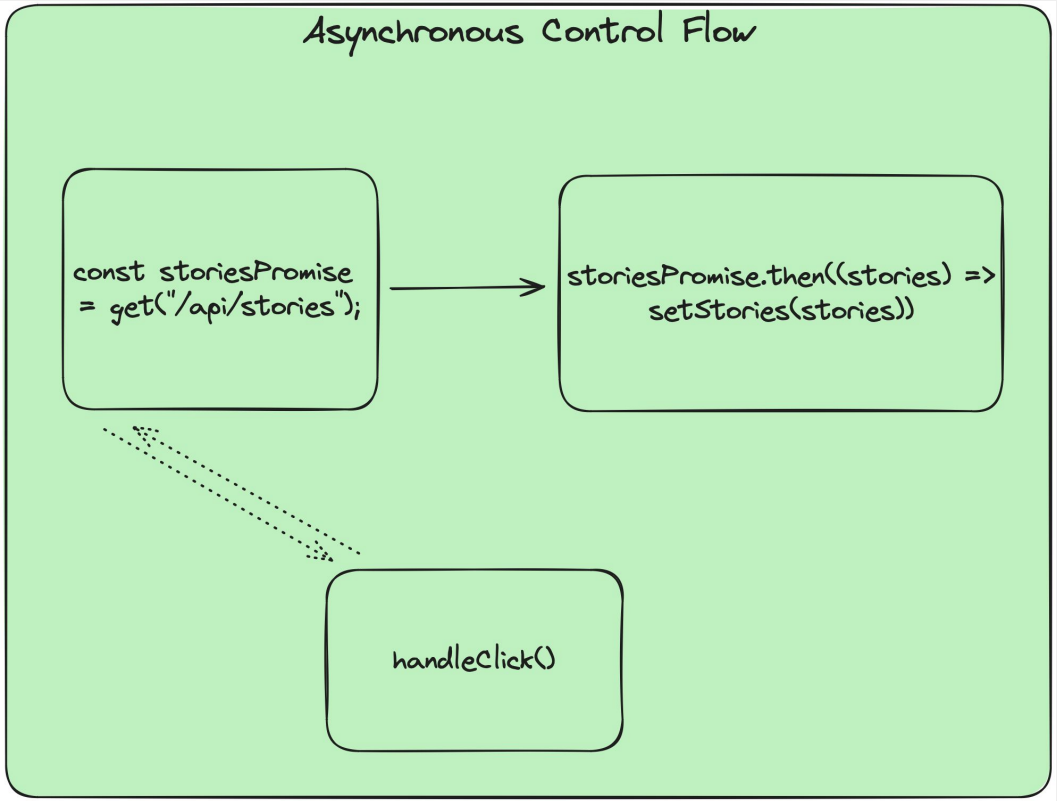
Asynchronous Control Flow



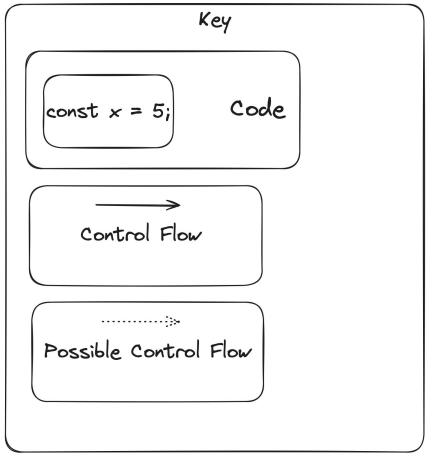


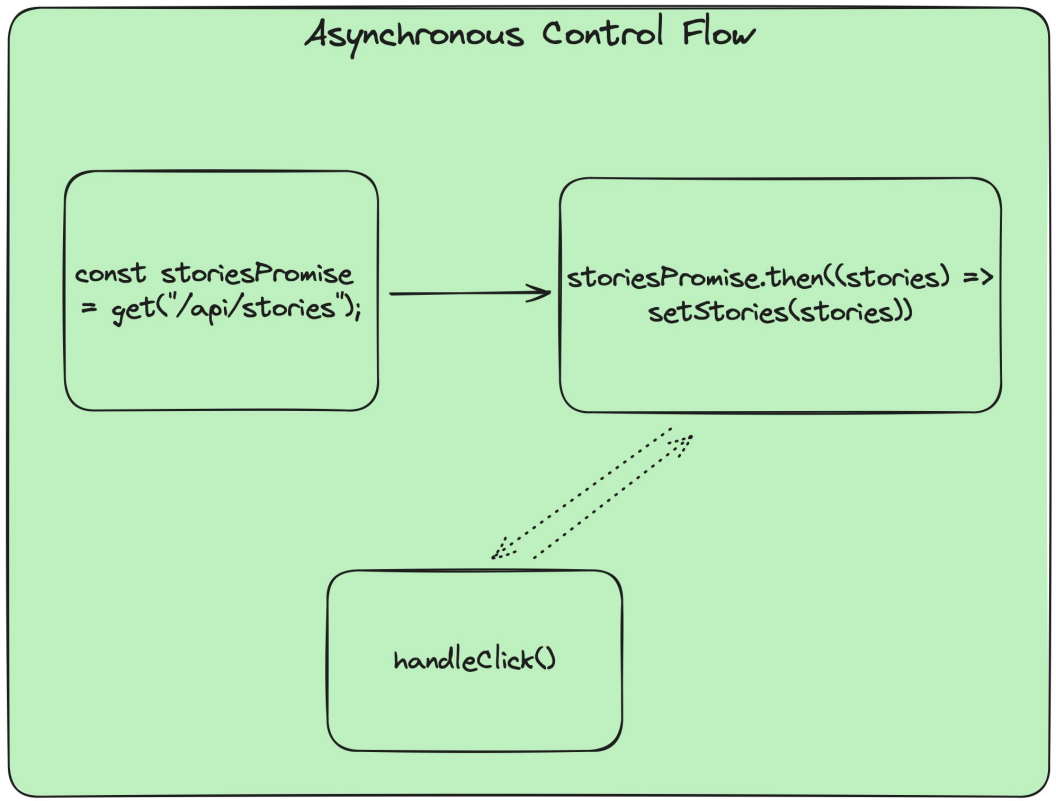
Asynchronous Control Flow



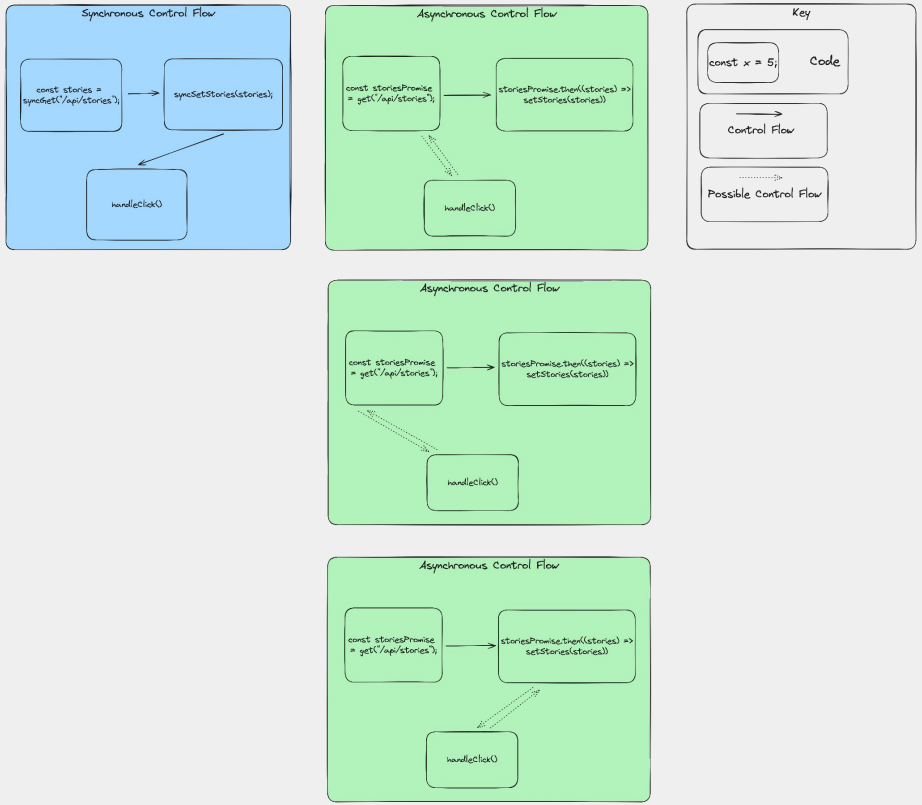


Asynchronous Control Flow





Control Flow (all together)



Using await

```
const a = slowNumber(9);
const b = slowNumber(10);

console.log(await a + await b);
```

await waits for the promise to **resolve** and uses that value

However, if you type this directly into VSCode, it will **NOT** work. It just gives an error when you try to run.

```
const a = await ....

const b = await ...

console.log()
```

If a takes 3 second to await, b takes 5, how long before print?

```
const a = await ....

const b = await ...

console.log()
```

If a takes 3 second to await, b takes 5, how long before print?

- 3
- 5
- 8

```
const a = await ....

const b = await ...

console.log()
```

If a takes 3 second to await, b takes 5, how long before print?

- 3
- 5
- **8**

Only asynchronous functions can use await

async await notation

```
const myFunction = async () => {
  console.log(await a + await b);
};
myFunction();
```

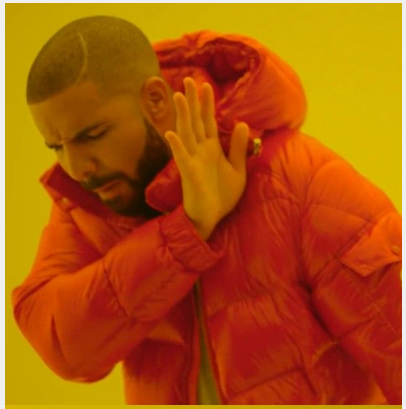
We can use the keyword await in async functions

async await notation

```
const myFunction = async () => {
  console.log(await a + await b);
};
myFunction();
```

```
a.then((aVal) => {
  b.then((bVal) => {
    console.log(aVal + bVal);
  });
});
```

async await notation



```
a.then((aVal) => {
  b.then((bVal) => {
    c.then((cVal) => {
      d.then((dVal) => {
        console.log(aVal + bVal + cVal + dVal);
      });
    });
  });
});
```

```
const myFunction = async () => {
  console.log(await a + await b + await c + await d);
};
myFunction();
```

Asynchronous functions

- Functions that return control back to the caller before computation is done
- Can be made as a callback function () => {}
- **OR** with the **async** keyword
  - Works with function, arrow functions, class methods, etc.

```
async function slowNumber(x) {
    sleep(1000);
    return x;
}
```

Using await

Notably, the outermost level of our program is **NOT** an async function, so it **CANNOT** use await. However, it will wait to resolve any promises at the end of the program before exiting.

```
async function main() {
    const a = slowNumber(9);
    const b = slowNumber(10);

    console.log(await a + await b);
}
main();
```

This prints 19 (as expected).

Async await can be used to rewrite anything we do with Promises

Async await is becoming more common than the traditional .then() Promises notation

Most of what you will write during web.lab workshops will still use the .then() notation

```
useEffect(() => {
  get("api/stories").then((storyObjs) => {
    setStories(storyObjs);
  });
}, []);
```

Traditional Promises

```
useEffect(() => {
  const getStories = async () => {
    const storyObjs = await get("api/stories");
    setStories(storyObjs);
  };
  getStories();
}, []);
```

async await

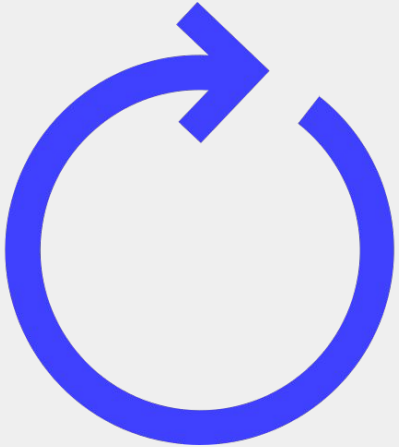
Using async await notation to perform a get request in `useEffect` requires us to define and call an async function inside the callback function passed to `useEffect`

Questions?

Why Async?

JavaScript Event Loop

Handles everything including things like button presses, inputs, etc.



JavaScript Event Loop

Event loop:

```
while True:
    if len(events) > 0:
        const { event, handler } = events.pop(0);
        handler(event);
    const nextCodeToRun = waitingPromises[randomInteger()];
    run(nextCodeToRun());
```

Promises

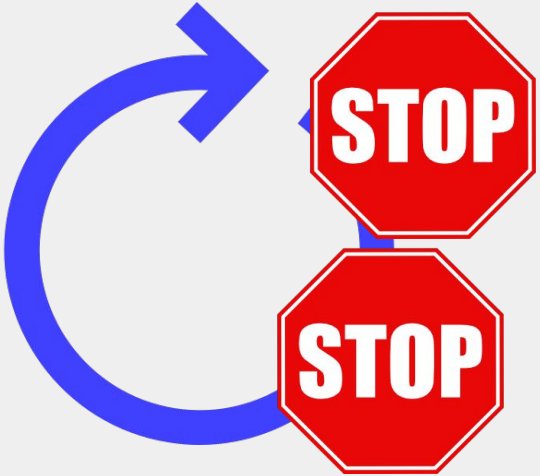
```
// A function promises call when
// they're waiting for something
// and can't progress.
function wait(promise):
    waitingPromises.push(promise);
    runEventLoop();
```

Events

```
// We run this function when events
// happen!
function notifyOfEvent(event,
eventHandler):
    events.push({ event, eventHandler
});
```

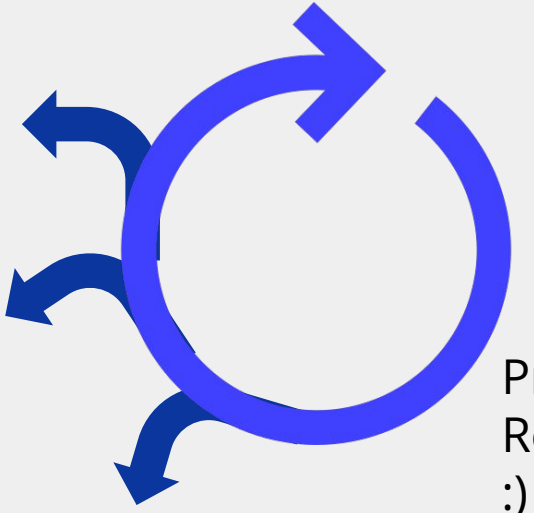
JavaScript Event Loop

Handles everything including things like button presses, inputs, etc.



Synchronous functions

Unresponsive :(



Preserves Responsiveness :)

Asynchronous functions

Rendering the Front End (React)

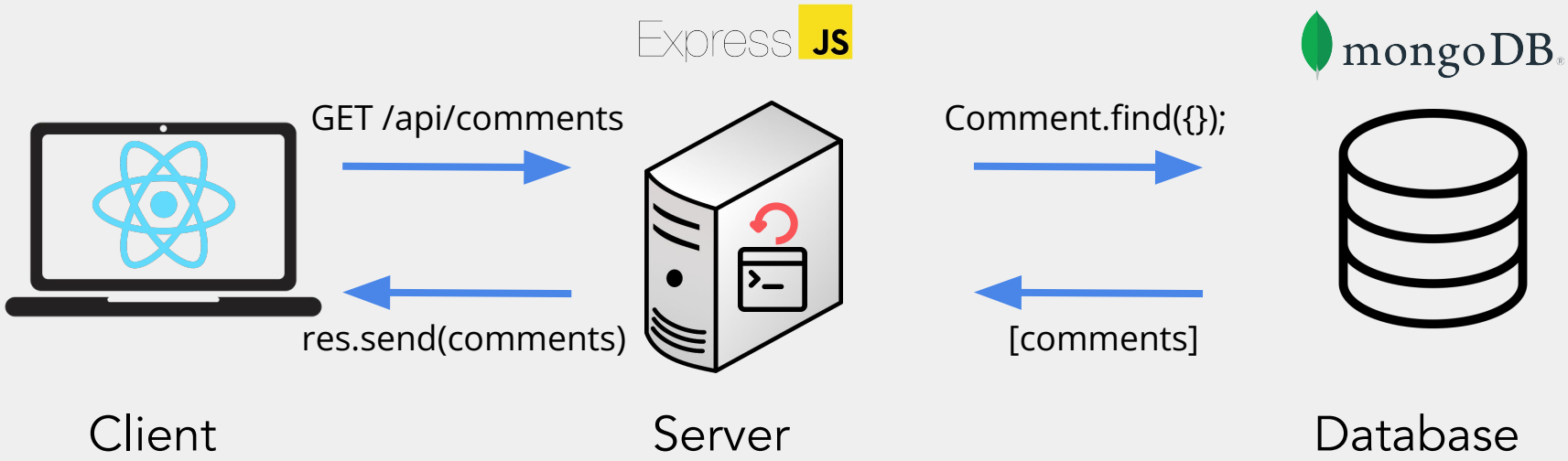
Things like setState()

```
const addNewStory = (storyObj) => {
  setStories([storyObj].concat(stories));
};
```

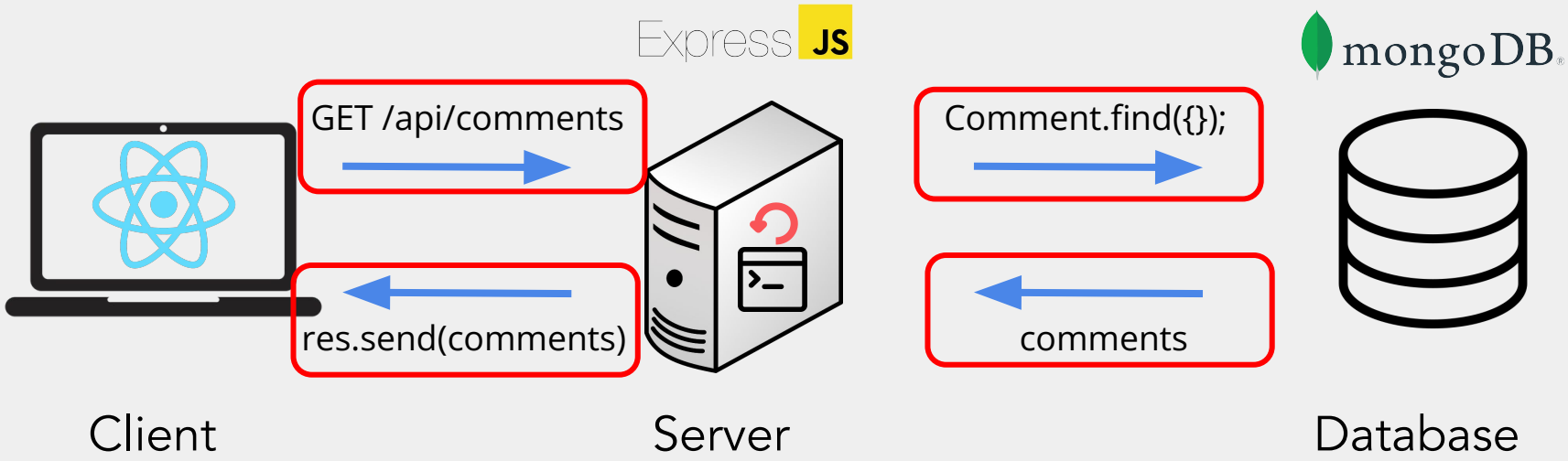
setState is asynchronous under the hood

When should we use async?

Communicating between different devices



Unable to guarantee how long it will take to communicate between multiple machines



Background Tasks

You can run background tasks without stopping the user from interacting with the front end.

- Fetching data (e.g. loading new posts on facebook)
- Downloads / uploads
    - Play music or video (playing music or video on youtube / spotify still lets you click around on other stuff)
- Run some big computation on a server
- And many more!

Recap

Recap

- Use **await** keyword to wait for promises to resolve
- Use **async** keyword to define asynchronous functions
- You MUST wrap every **await** within an **async** function
- Anything you do with promises can be rewritten with async await notation- use whatever is best for the job

Recap

Use async when we don’t know how long something will take:

- Client-server communication
- Server-database communication
- React Front End
- Background Tasks

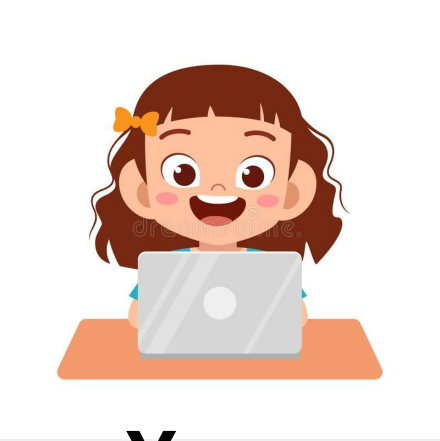
Do not eat in the lecture hall :)

Be Back at 1:00pm :)

Lunch Logistics

Open portal.weblab.is
and log in, showing us that you
are registered for the class

Eat outside the classroom,
Return at 1:00 :)



You