# CIS 371 Web Application Programming

## JS|TS Promise

### Handling Asynchronous Results

GRAND VALLEY STATE UNIVERSITY®

Lecturer: Dr. Yong Zhuang

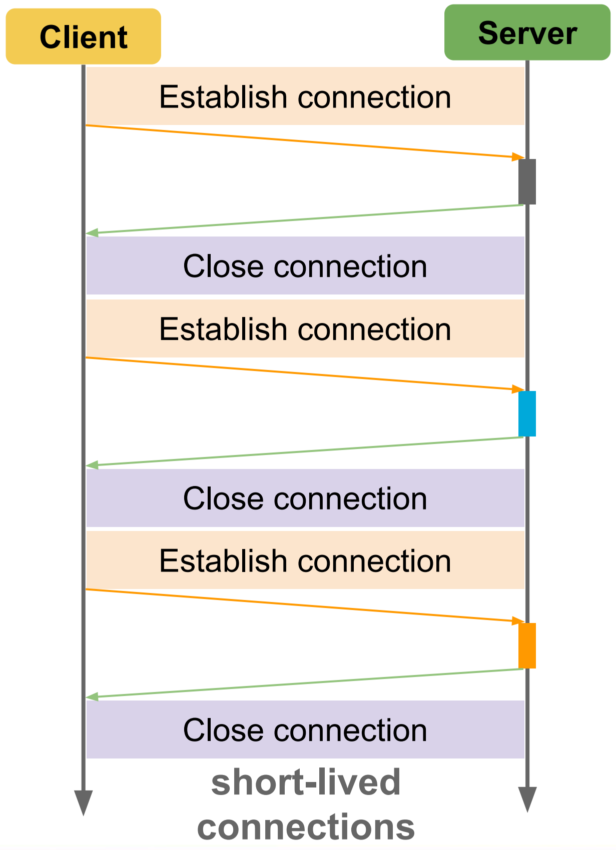
Based on the original version by Professor Hans Dulimarta.

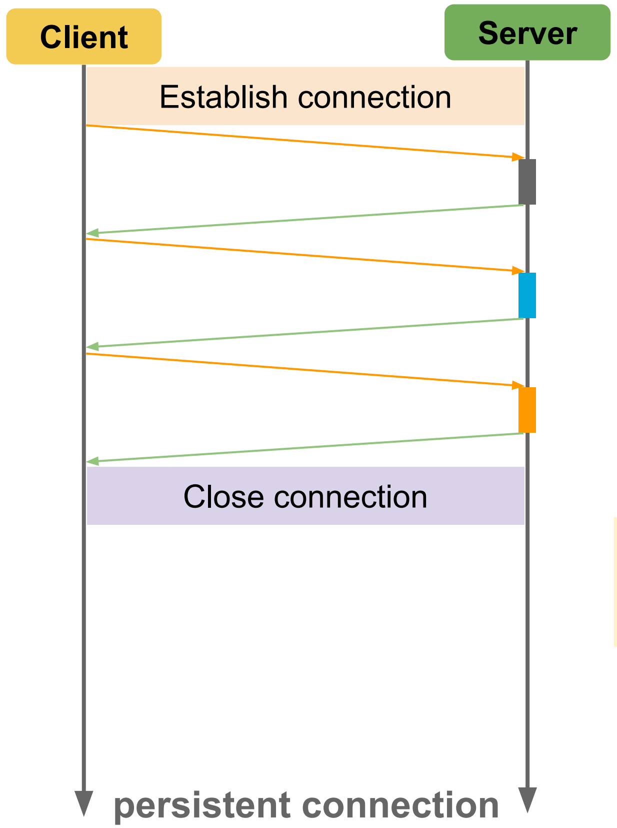
## Topics

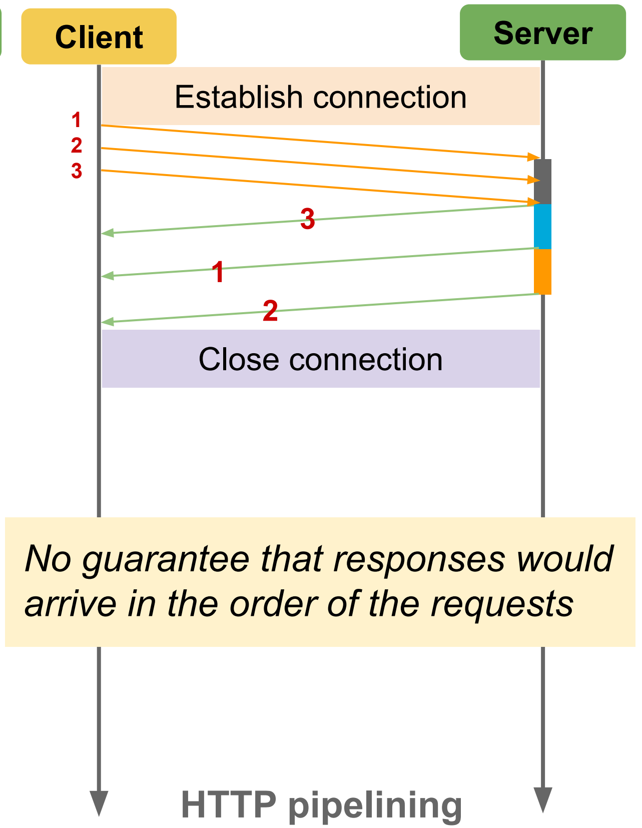
- Client/Server Communication
    - Synchronous
    - Asynchronous
- Callback functions (for handling asynchronous events)
- Promise

## Reference: Promise Documentation (@ MDN)

# Client/Server: HTTP Requests & Responses

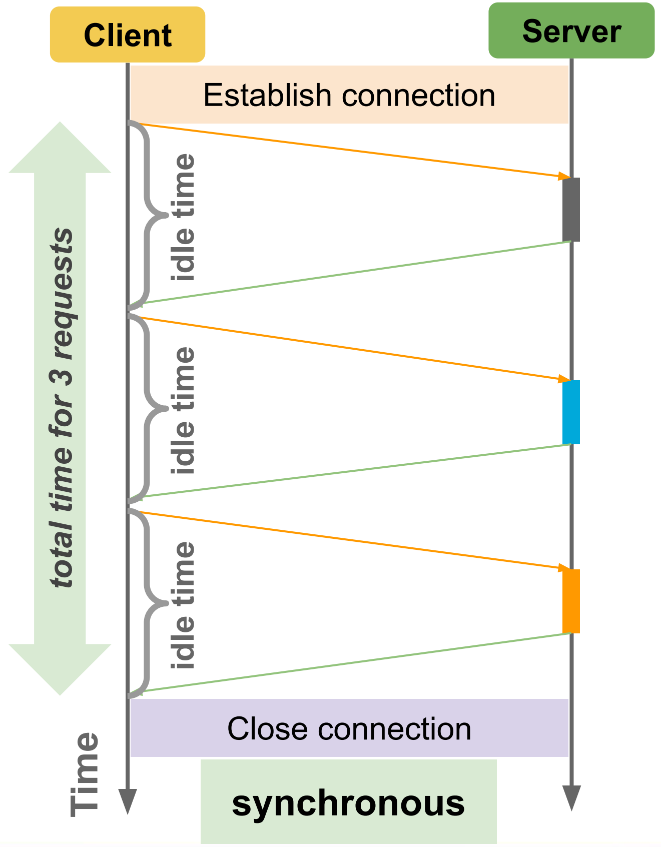


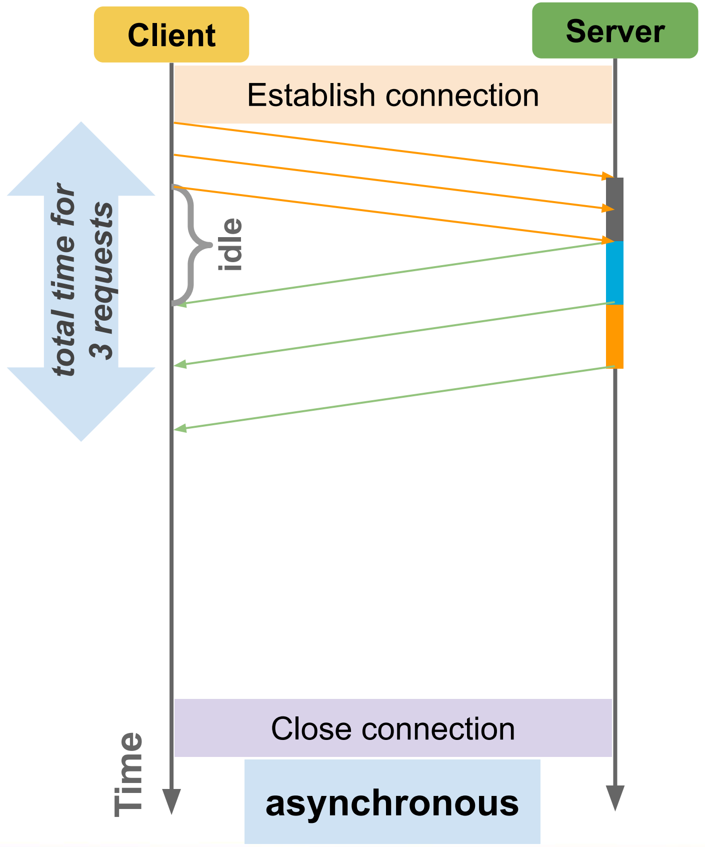




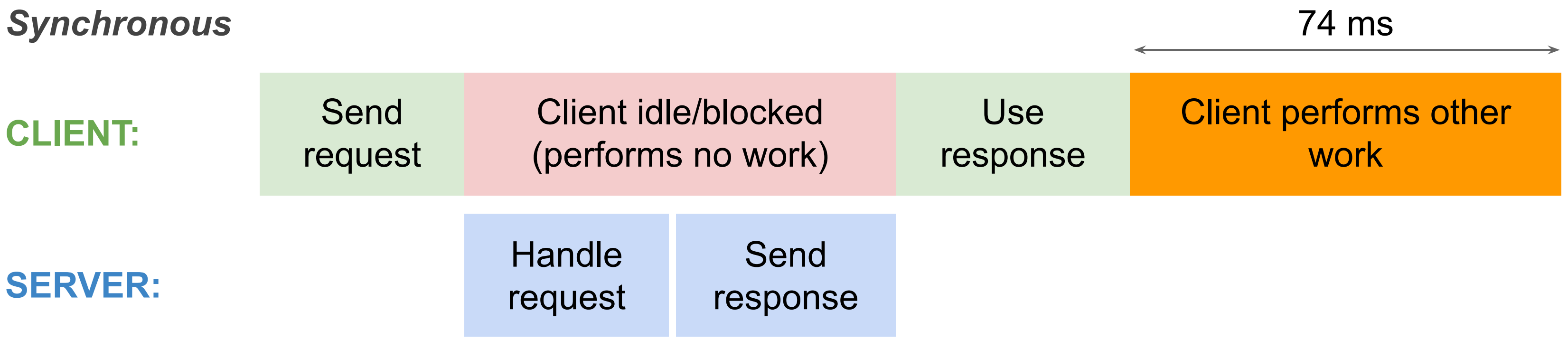
*No guarantee that responses would arrive in the order of the requests*

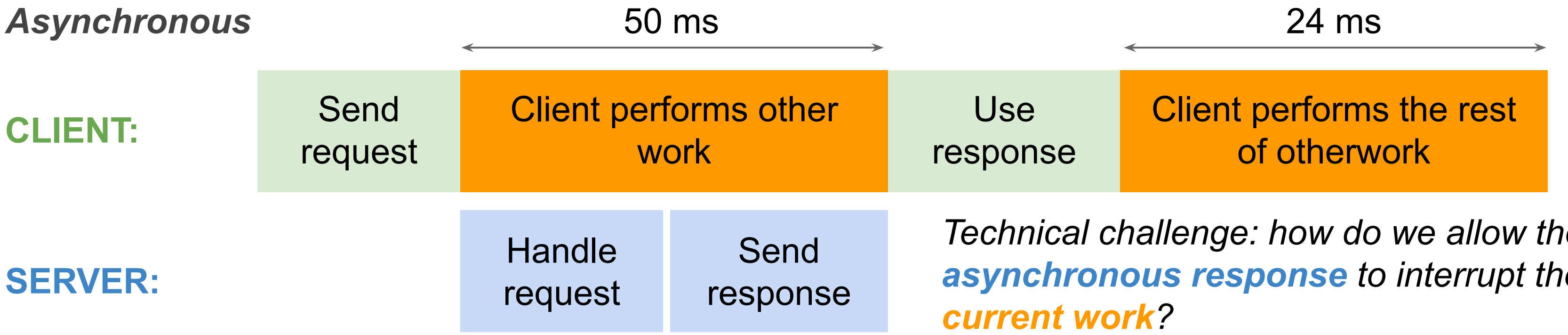
# Client/Server: HTTP Requests & Responses





# Synchronous vs. Asynchronous Requests



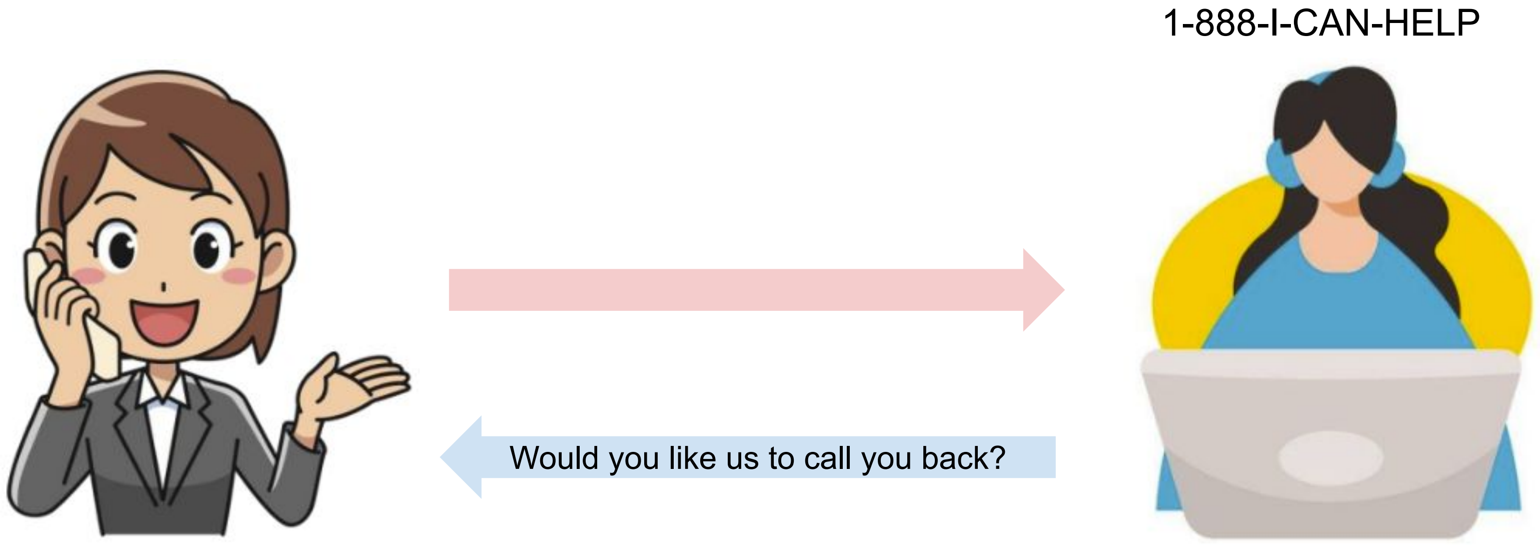


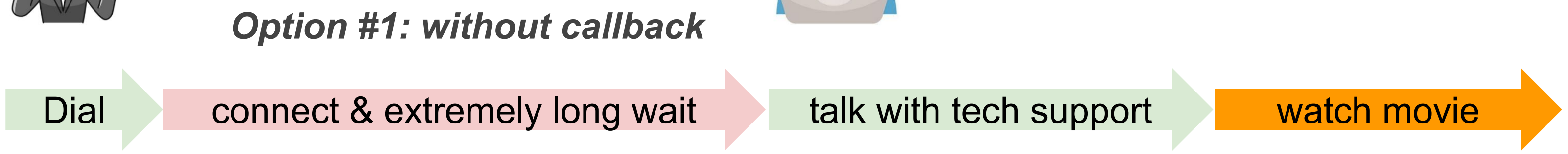
*Technical challenge: how do we allow the **server asynchronous response** to interrupt the **client current work**?*

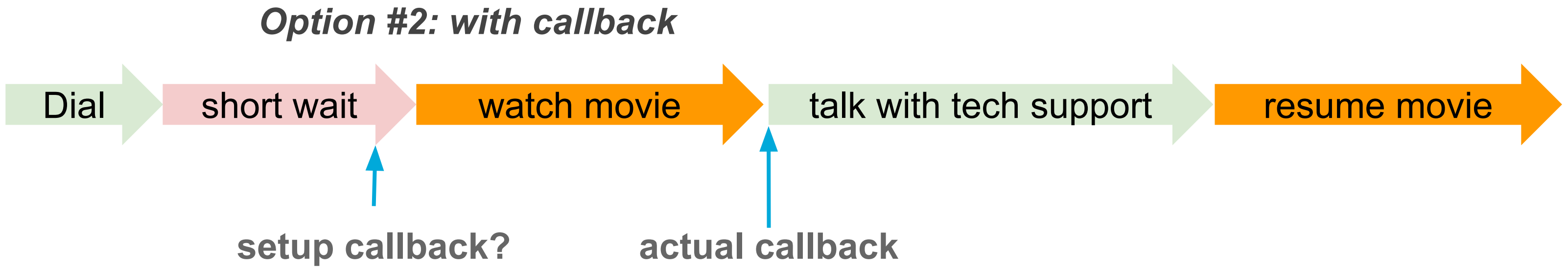
## Sending Requests: easy
## Receiving Async Responses: requires extra setup

## Callback Actions
## (JS Callback Functions)

## You are number 17 in line.....




*Option #1: without callback*
Dial
connect & extremely long wait
talk with tech support
watch movie

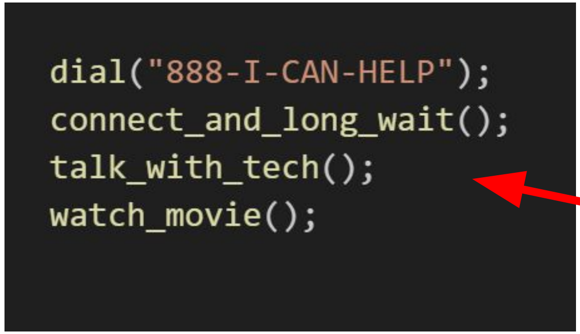

*Option #2: with callback*
Dial
short wait
watch movie
talk with tech support
resume movie
setup callback?
actual callback

# Synchronous Call (in code)

555-4321

1-888-I-CAN-HELP

Dial → connect & extremely long wait → talk with tech support → watch movie



```
dial("888-I-CAN-HELP");
connect_and_long_wait();
talk_with_tech();
watch_movie();
```

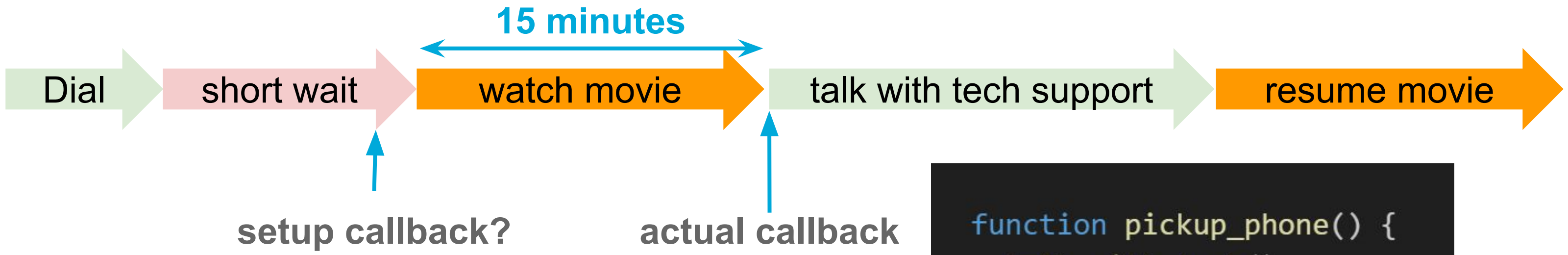
*Order of execution = order of line of code*

## Async: “out-of-order” execution
## (Order of execution ≠order of line of code)

# Async Phone Calls with Callback (in code)

555-4321

1-888-I-CAN-HELP



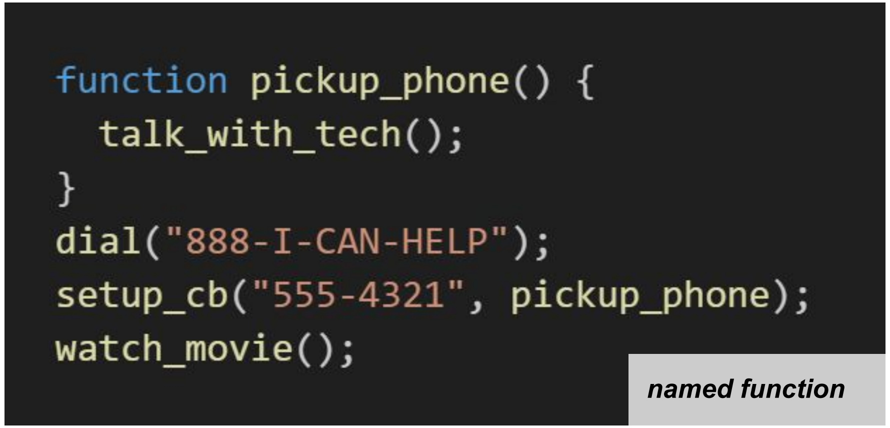
```
function pickup_phone() {
  talk_with_tech();
}
```

***15 mins later***

```
dial("888-I-CAN-HELP");
setup_cb("555-4321", pickup_phone);
watch_movie();
```

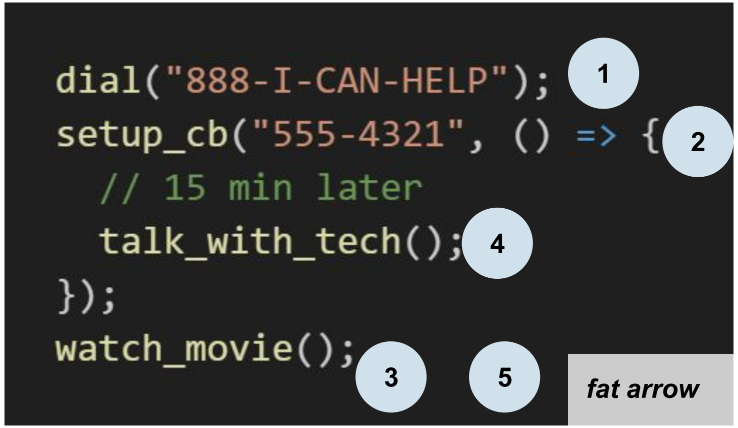
*Asynchronous (incoming call) while you're watching movie*

# Callback fns (Fat Arrow)



```
function pickup_phone() {
  talk_with_tech();
}
dial("888-I-CAN-HELP");
setup_cb("555-4321", pickup_phone);
watch_movie();
```

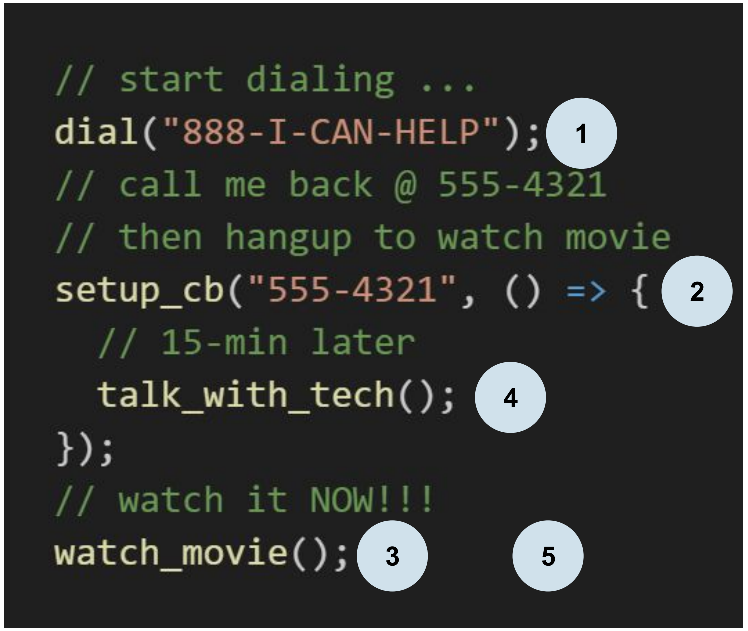
*named function*



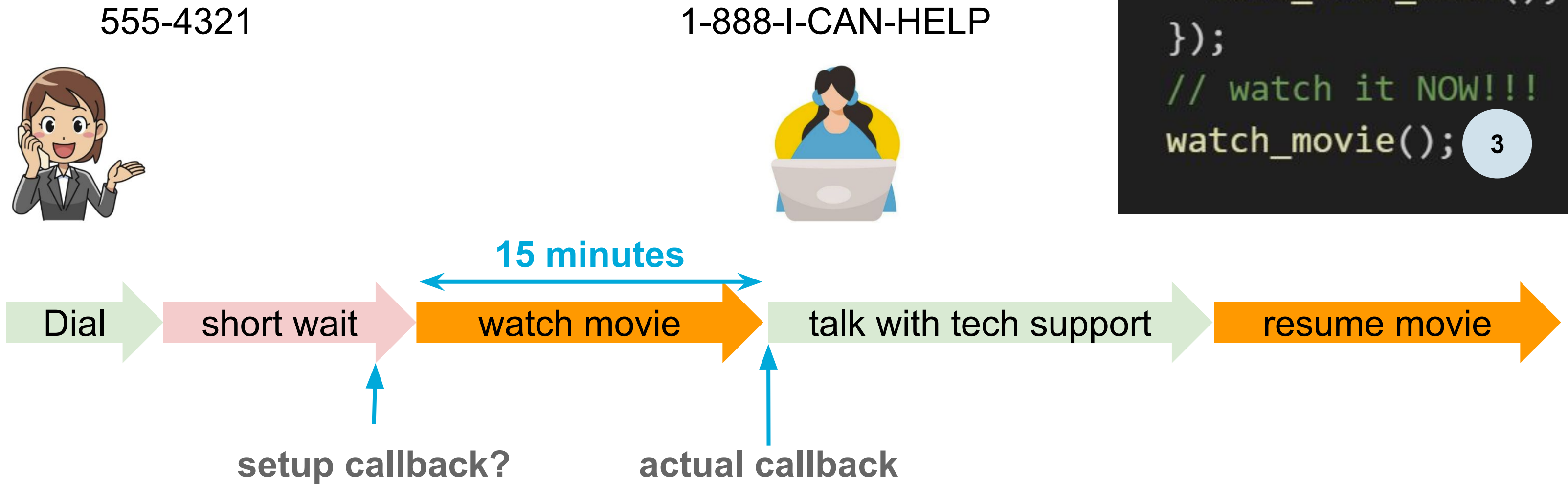
```
dial("888-I-CAN-HELP");
setup_cb("555-4321", () => {
  // 15 min later
  talk_with_tech();
});
watch_movie();
```

*fat arrow*

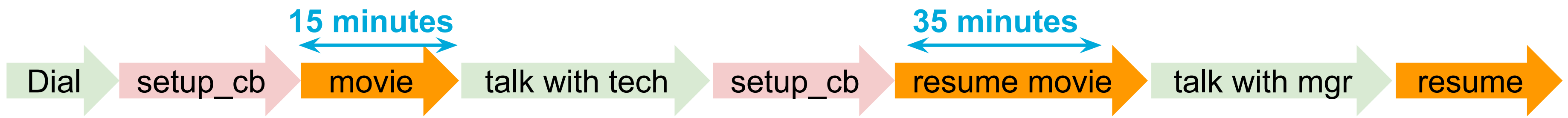
*Async: order of execution ≠ order of line of code*

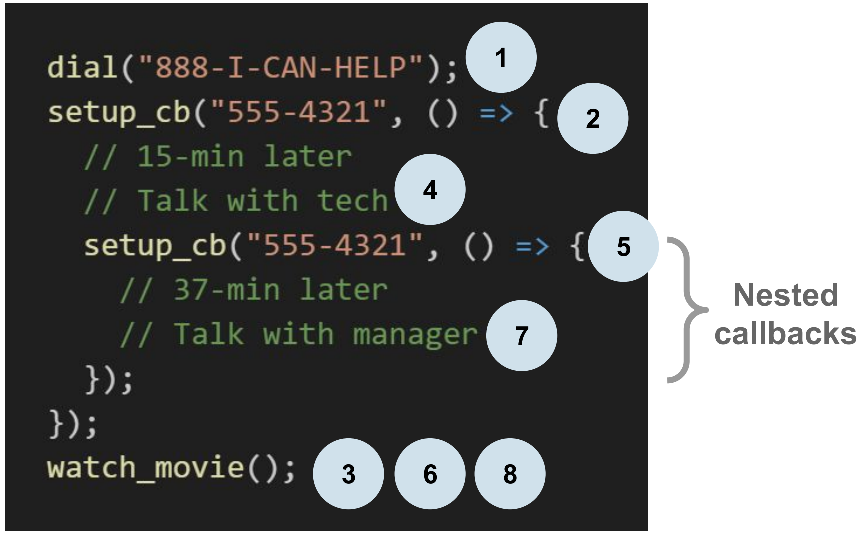


```
// start dialing ...
dial("888-I-CAN-HELP");
// call me back @ 555-4321
// then hangup to watch movie
setup_cb("555-4321", () => {
   // 15-min later
   talk_with_tech();
});
// watch it NOW!!!
watch_movie();
```



## Tech: “But, you have to talk with my manager” (Nested Callback)





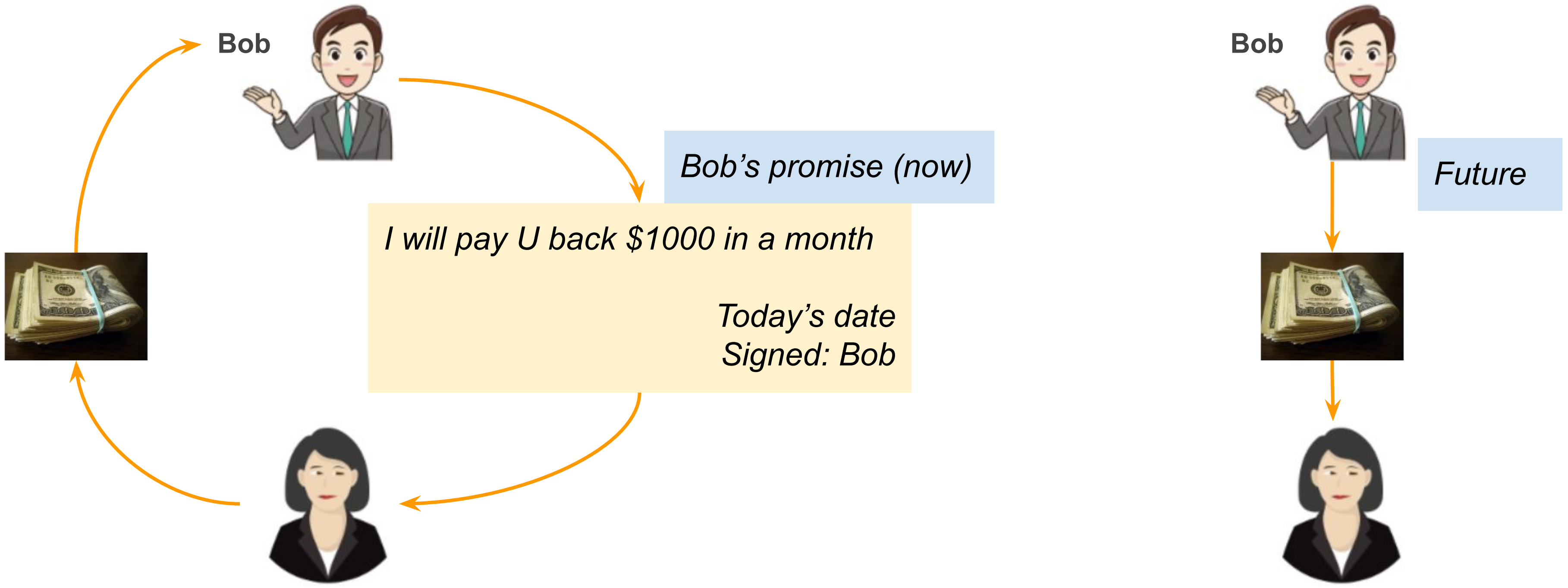
```
dial("888-I-CAN-HELP");
setup_cb("555-4321", () => {
  // 15-min later
  // Talk with tech
  setup_cb("555-4321", () => {
    // 37-min later
    // Talk with manager
  });
});
watch_movie();
```

## How to Initiate Async HTTP Requests?

- fetch() function
  - Native in browser
  - NPM node-fetch
- Axios library
- Both fetch() and axios() use JS Promise

**IOU = I owe you note**
**Promise to pay debt/loan**

# Borrowing Money: Promise Now, Pay Later



*I will pay U back $1000 in a month*

*Today's date*
*Signed: Bob*

Today (now): **Beth** receives a promise

1 month later (future): **Beth** receives cash

## A promise = now confirmation of future action(s)
## A JS promise = a "now" object representing data which will become available in the future

## Promise Example

```
function nthPrime(nth: number): Promise<number> {
  // work takes 10 seconds
  return Promise.resolve(_______);
}
```

```
function nthPrimeNow(nth: number): number {
  // work takes 10 seconds
  return _______;
}
```

```
console.log("Start");
const prom = nthPrime(500);
prom.then((pr: number) => {
  console.log("The 500th prime is", pr);
});
doMoreWork();
```

```
console.log("Start");
const pr = nthPrimeNow(500);
console.log("The 500th prime is", pr);
doMoreWork();
```

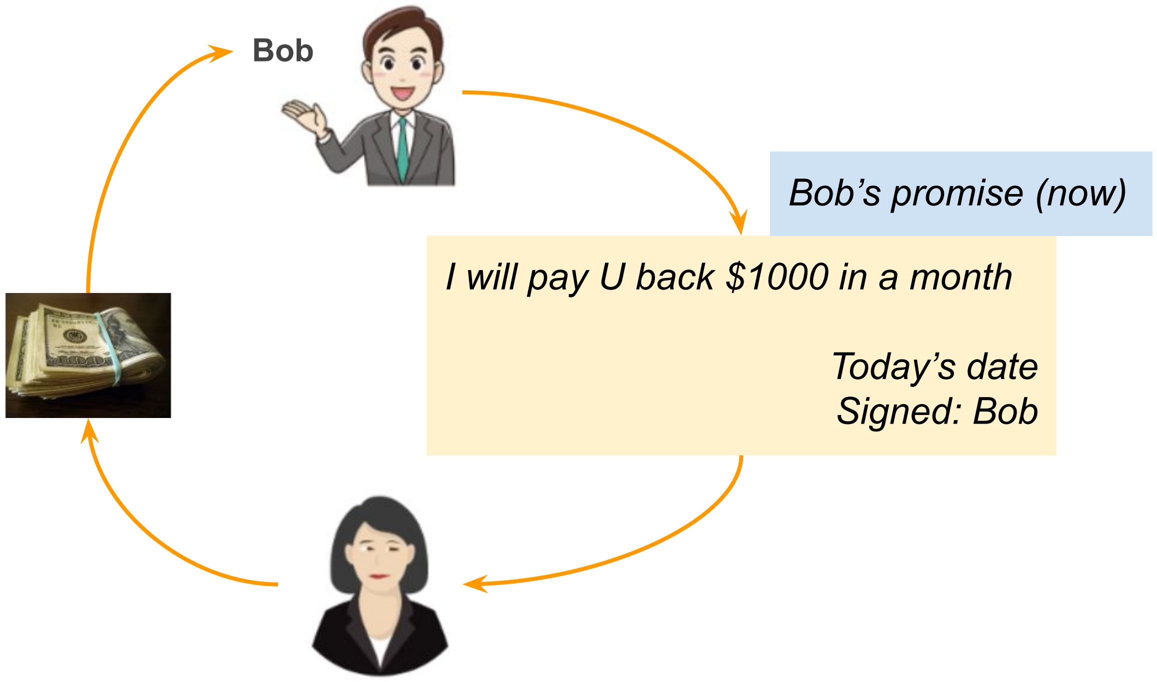
```
Start
Partial output of doMoreWork()
# After 10 seconds
The 500th prime is 3571
More output from doMoreWork()
```

*Compare the order of execution*

```
Start
# After 10 seconds
The 500th prime is 3571
Output of doMoreWork()
```

**Loan is either paid-off or defaulted**
**Promise is either resolved or rejected**

# Borrowing Money: Promise Now, Never Pay

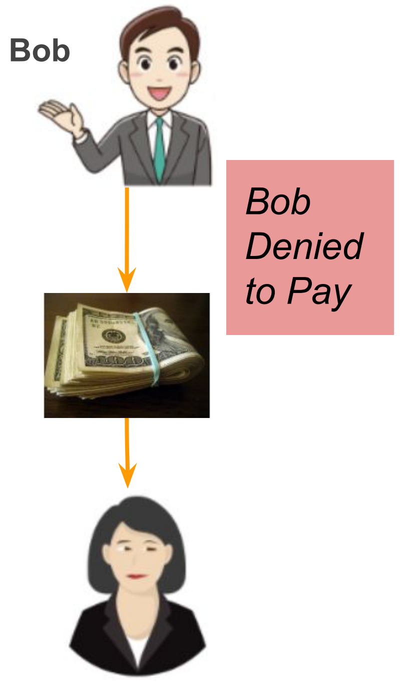


*Bob's promise (now)*

*I will pay U back $1000 in a month*

*Today's date*
*Signed: Bob*

Today (now): **Beth** receives a promise



1 month later (future): **Beth** gets none

## Promise settlement: resolve() or reject()

```
function nthPrime(nth: number): Promise<number> {
  if (nth < 100_000) {
    // assume prime calculation takes 10 seconds
    return Promise.resolve(a_prime_number_here);
  } else return Promise.reject("Can't compute prime");
}
```

```
console.log("Start");
nthPrime(500).then((pr: number) => {
  console.log("Prime is", pr);
});
.catch((err:any) => {
console.log("Rejected", err);
});
console.log("Here");
```

```
# Watch for order of execution
Start
Here
# if the promise is resolved
# After 10 seconds ...
Prime is 3571
# if the promise is rejected
Rejected Can’t compute prime
```

## JS Promise

- Basic methods: then(), catch(), finally()
- Basics static functions
  - Promise.resolve()
  - Promise.reject()
- Advanced (for handle multiple concurrent promises)
  - Promise.all(array): wait until all the promises in the array are resolved
  - Promise.allSettled(array): wait until all the promises in the array are either resolved or rejected
  - Promise.any(array): wait until ONE of the promises in the array is resolved
  - Promise.race(array): wait until ONE of the promises in the array is either resolved or rejected

## then-able chains

## Then and then and then and ...

```
function nthPrime(nth: number): Promise<number> {
  // more code here
  return Promise.____;
}
```

```
function toRomanNumeral(inputNum: number): string {
  // conversion to Roman numberal
  return _______;
}
```

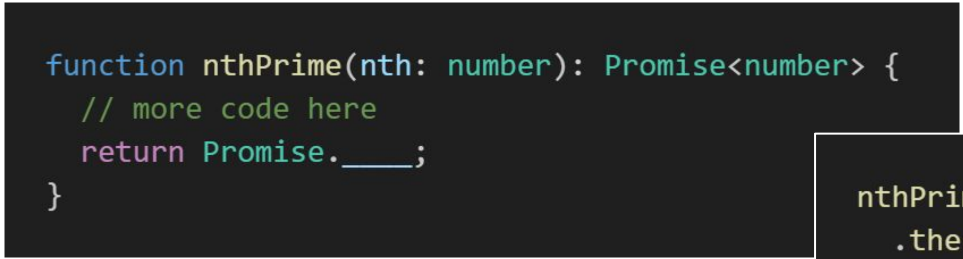
*Return from a then() becomes a Promise to the next then() inline*

```
nthPrime(500)
  .then((p: number): string => {
    return toRomanNumeral(p);
  })
  .then((rome: string) => {
    console.log(`Prime in roman numeral ${rome}`);
  });
```

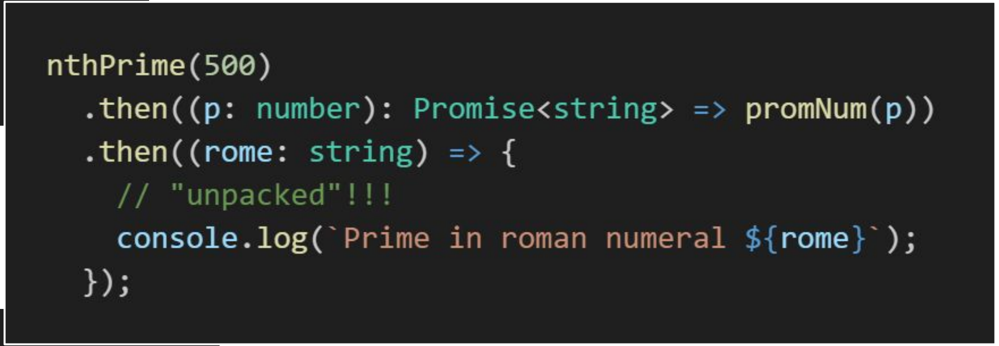
```
// After 1-line return elimination
nthPrime(500)
  .then((p: number): string => toRomanNumeral(p))
  .then((rome: string) => {
    console.log(`Prime in roman numeral ${rome}`);
  });
```

Demo

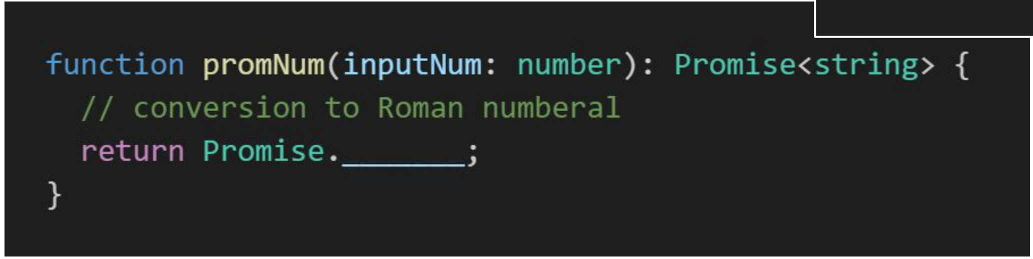
# Then and then and … (promise “unpacked”)



```
function nthPrime(nth: number): Promise<number> {
  // more code here
  return Promise.____;
}
```



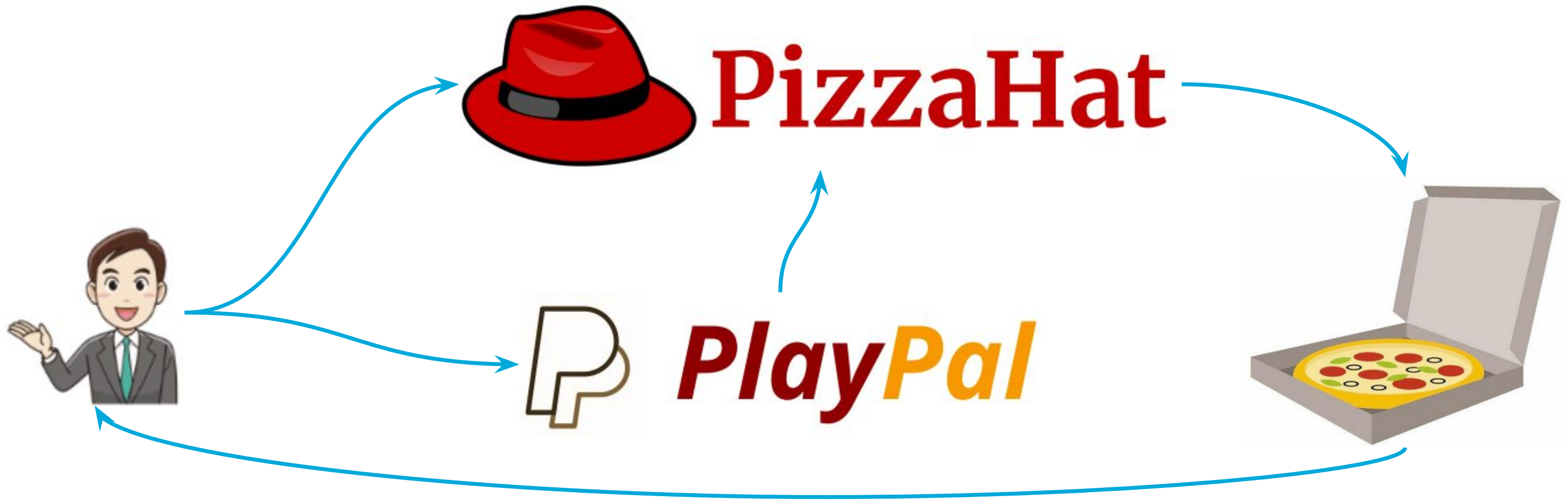
```
nthPrime(500)
  .then((p: number): Promise<string> => promNum(p))
  .then((rome: string) => {
    // "unpacked"!!!
    console.log(`Prime in roman numeral ${rome}`);
  });
```



```
function promNum(inputNum: number): Promise<string> {
  // conversion to Roman numberal
  return Promise._______;
}
```

Demo

## Online Pizza Order & 3rd party payment



# Online Pizza Order (code setup)

```
type PizzaBox = {
  customerName: string;
  inStorePickup: boolean;
};
```

```
function orderPizza(___): Promise<PizzaOrder> {
  return Promise.resolve(___);
}
```

PizzaHat

```
type PizzaOrder = {
  crustStyle: "Classic" | "ThinCrust" | "HandTossed";
  size: number;
  toppings: Array<Topping>;
  customerName: string;
  price: number;
};
```

```
type ProofOfPlay = {
  payer: string;
  payee: string;
  amount: number;
  transactionDate: string;
};
```

```
function makePizza(____): Promise<PizzaBox> {
  return Promise.resolve(___);
}
```

PizzaHat

```
function playWithPal(name: string, payAmt: number): Promise<ProofOfPlay> {
  return Promise.resolve(___);
}
```

PlayPal

```
orderPizza(____)
  .then((ord: PizzaOrder) => playWithPal(__, __))
  .then((proof: ProofOfPlay) => makePizza(____))
  .then((box: PizzaBox) => {
    console.log("Open the box and enjoy!");
  })
  .catch((err: any) => {
    console.error("Can't complete order");
  });
```

# Online Pizza Order (chaining)

```
function orderPizza(___): Promise<PizzaOrder> {
  return Promise.resolve(___);
}
```

*PizzaHat*

```
function makePizza(____): Promise<PizzaBox> {
  return Promise.resolve(___);
}
```

*PizzaHat*

```
function playWithPal(name: string, payAmt: number): Promise<ProofOfPlay> {
  return Promise.resolve(___);
}
```

*PlayPal*

# Promise: with finally

```
function nthPrime(nth: number): Promise<number> {
  // work takes 10 seconds
  return _______;
}
```

```
console.log("Start");
nthPrime(500).then((pr: number) => {
  console.log("Prime is", pr);
});
doMoreWork();
```

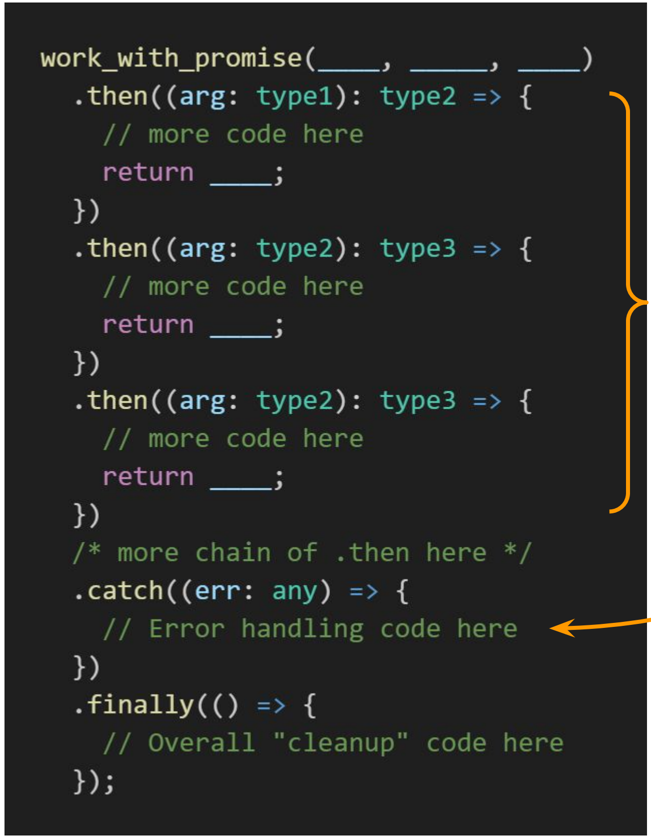
```
Start
Partial output of doMoreWork()
# After 10 seconds
Prime is 3571
More output from doMoreWork()
```

```
console.log("Start");
nthPrime(500)
  .then((pr: number) => {
    console.log("Prime is", pr);
  })
  .finally(() => {
    doMoreWork();
  });
```

```
Start
# After 10 seconds
Prime is 3571
Output of doMoreWork()
```

*The finally method is used to specify a block of code that will run after the promise is settled, regardless of whether it was resolved or rejected.*

## Promise: put them all together



```
work_with_promise(____, _____, ____)
  .then((arg: type1): type2 => {
    // more code here
    return ____;
  })
  .then((arg: type2): type3 => {
    // more code here
    return ____;
  })
  .then((arg: type2): type3 => {
    // more code here
    return ____;
  })
  /* more chain of .then here */
  .catch((err: any) => {
    // Error handling code here
  })
  .finally(() => {
    // Overall "cleanup" code here
  });
```

*Any Promise.reject() here will be caught by*

*Promise.reject() skips then-chain until it finds a .catch*

## async & await

# Async functions

```
function nthPrime(nth: number): Promise<number> {
  let thePrime: number;
  // more code here
  return Promise.resolve(thePrime);
}
```

*What is the difference between promise functions with and without the async keyword?*

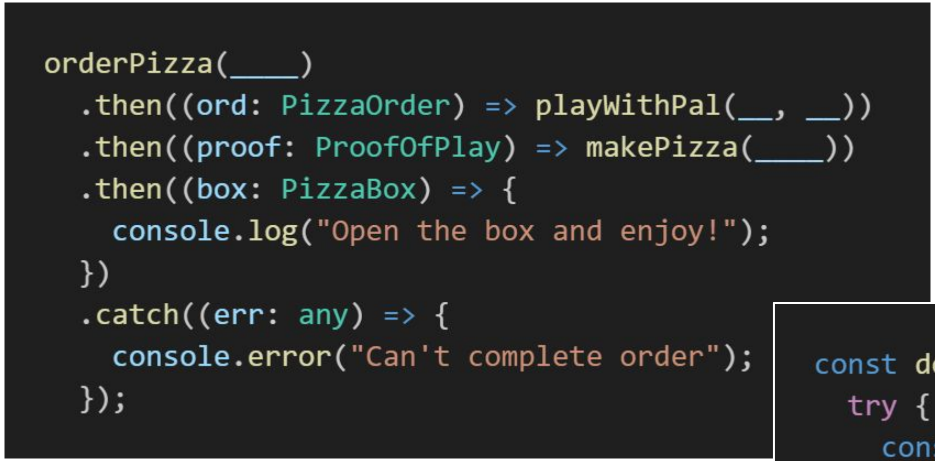
```
async function nthPrime(nth: number): Promise<number> {
  let thePrime: number;
  // more code here
  return thePrime; // Promise.resolve() is not required
}
```

*The async keyword makes asynchronous functions look and behave more like synchronously way by*

- *removing the need for explicit promise creation.*
- *using the await keyword to pause the async function execution*

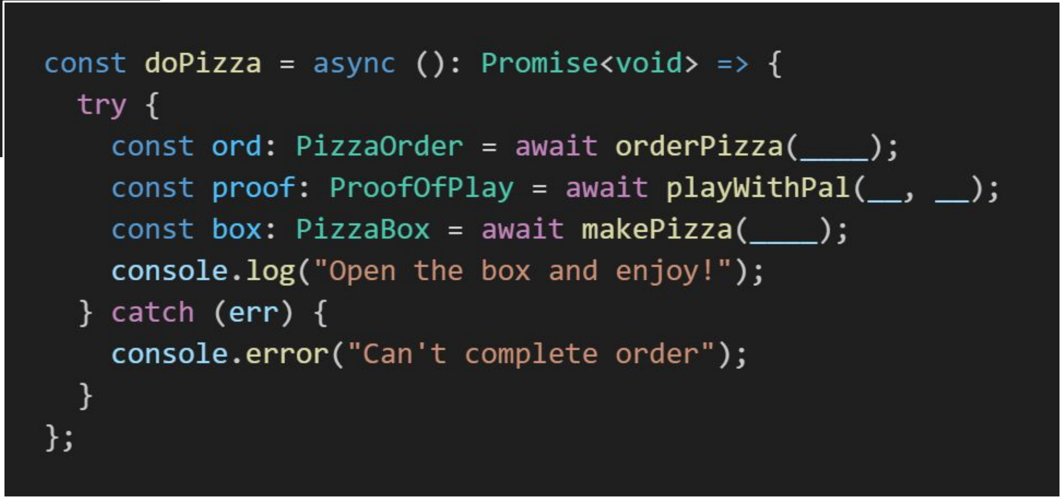
```
const nthPrime = async (nth: number): Promise<number> => {
  let thePrime: number;
  // more code here
  return thePrime; // Promise.resolve() is not required
};
```

# await: rewrite in synchronous style



```
orderPizza(____)
  .then((ord: PizzaOrder) => playWithPal(__, __))
  .then((proof: ProofOfPlay) => makePizza(____))
  .then((box: PizzaBox) => {
    console.log("Open the box and enjoy!");
  })
  .catch((err: any) => {
    console.error("Can't complete order");
  });
```

*Await can only be used inside async functions*



```
const doPizza = async (): Promise<void> => {
  try {
    const ord: PizzaOrder = await orderPizza(____);
    const proof: ProofOfPlay = await playWithPal(__, __);
    const box: PizzaBox = await makePizza(____);
    console.log("Open the box and enjoy!");
  } catch (err) {
    console.error("Can't complete order");
  }
};
```