

# CIS 658 Web Architectures

## JS|TS Promise

Handling Asynchronous Results



GRAND VALLEY  
STATE UNIVERSITY®

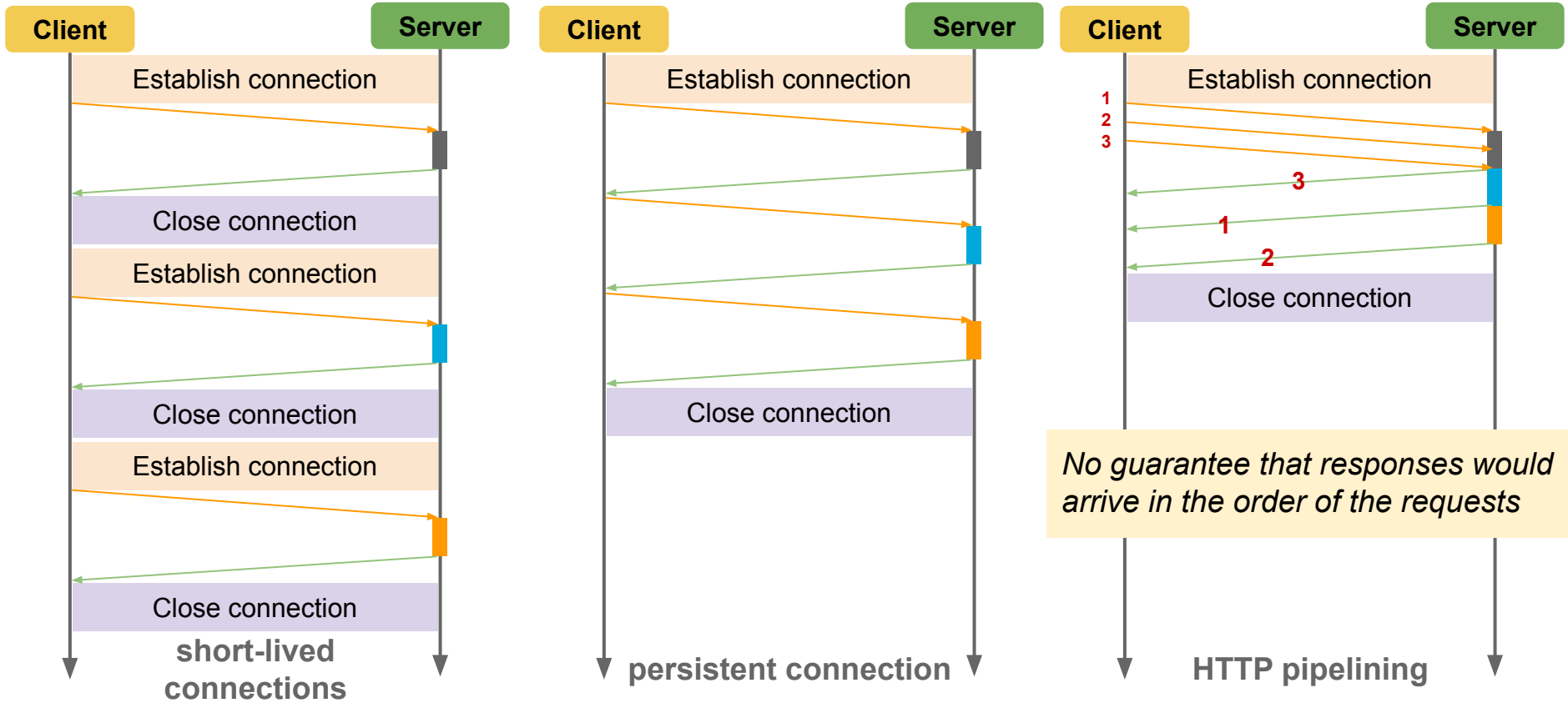
Lecturer: **Dr. Yong Zhuang**

# Topics

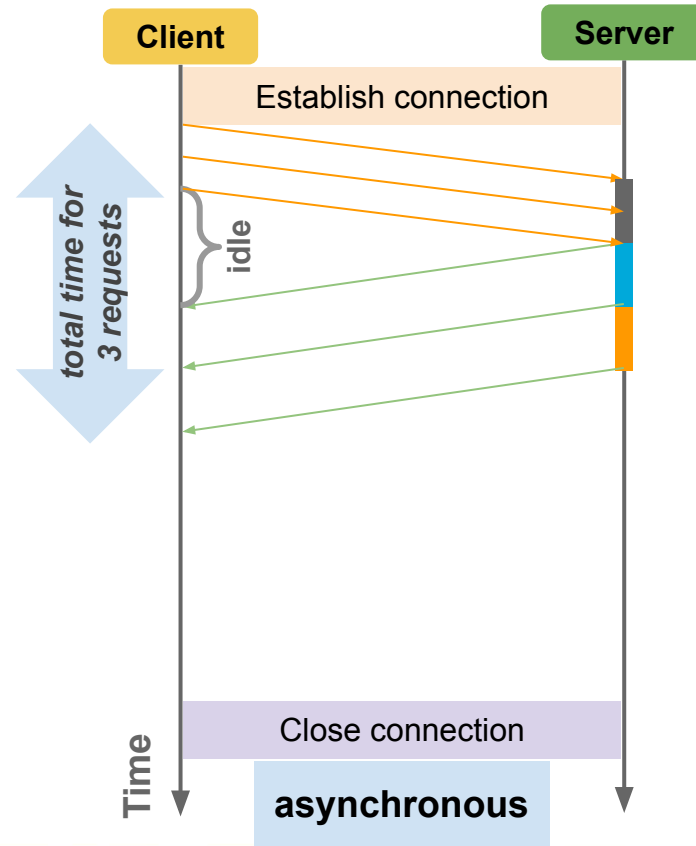
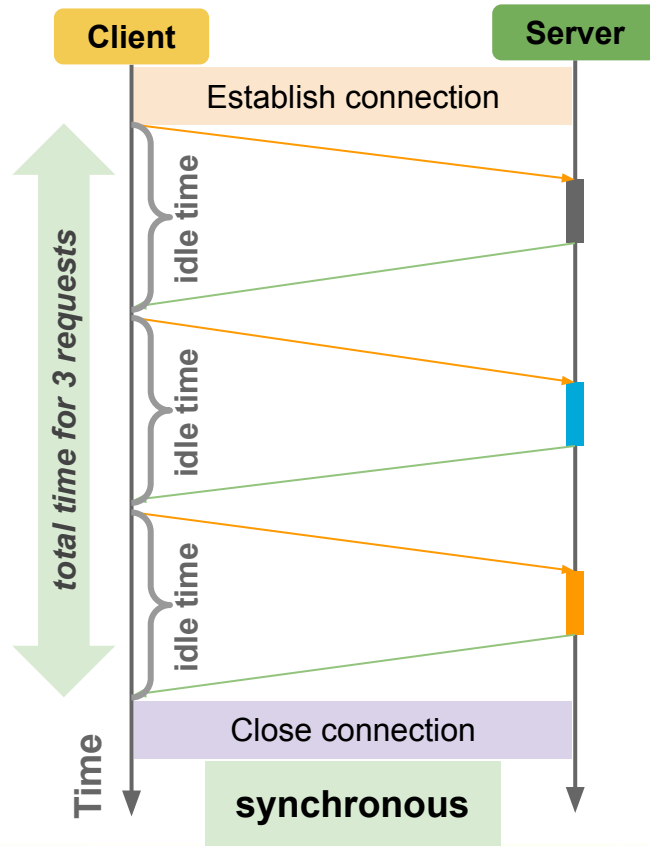
- Client/Server Communication
  - Synchronous
  - Asynchronous
- Callback functions (for handling asynchronous events)
- Promise

Reference: Promise Documentation (@ MDN)

# Client/Server: HTTP Requests & Responses

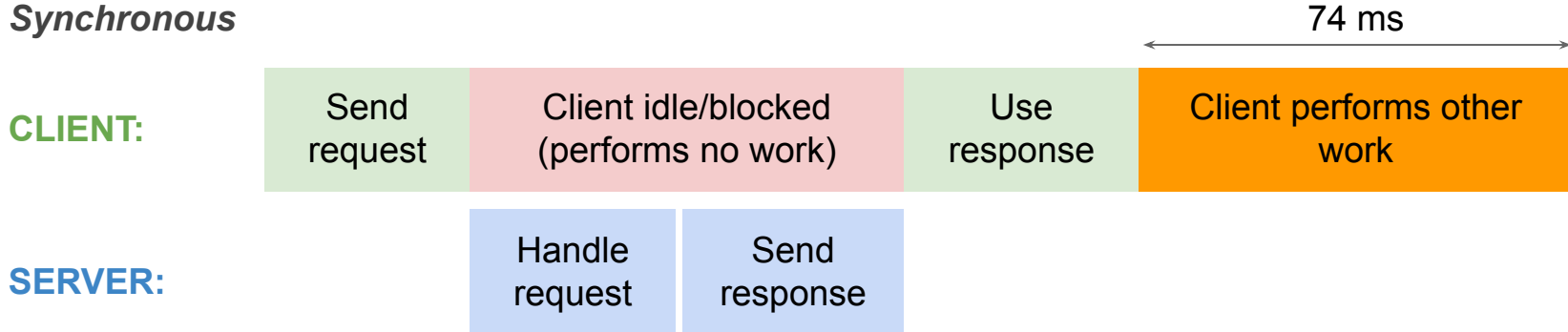


# Client/Server: HTTP Requests & Responses

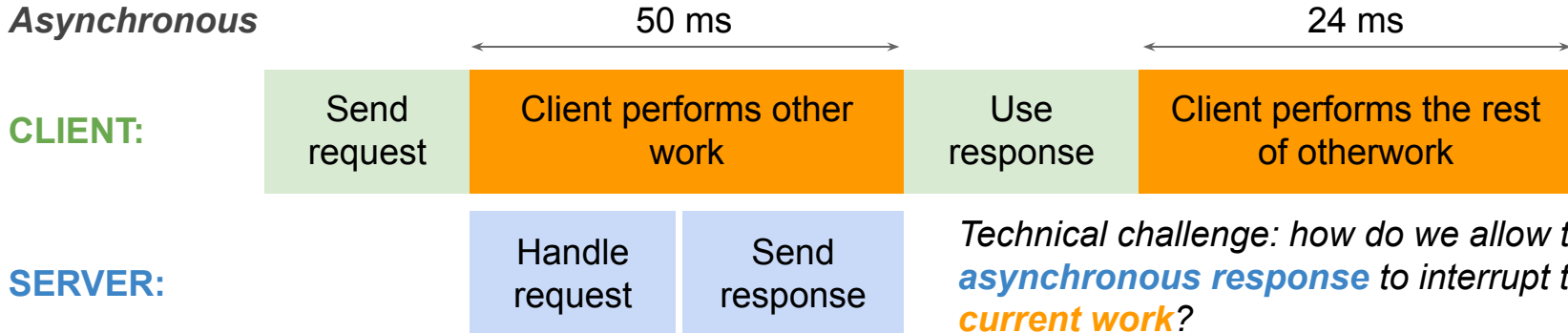


# Synchronous vs. Asynchronous Requests

## Synchronous



## Asynchronous



*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.....



Would you like us to call you back?

1-888-I-CAN-HELP

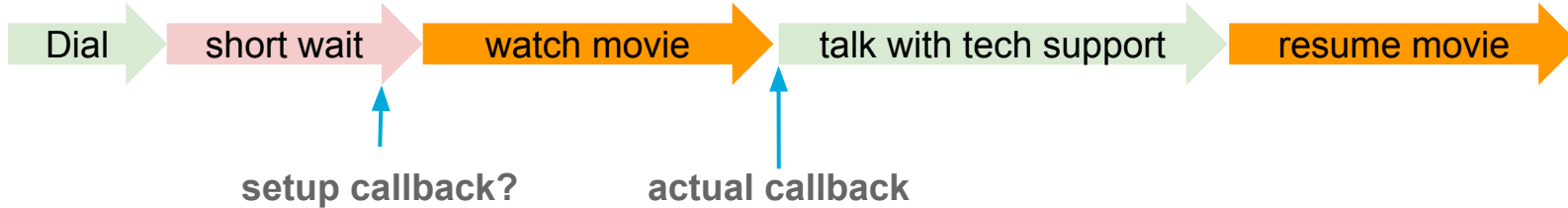




**Option #1: without callback**



**Option #2: with callback**



# Synchronous Call (in code)

555-4321



1-888-I-CAN-HELP



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

*Order of execution = order of line of code*

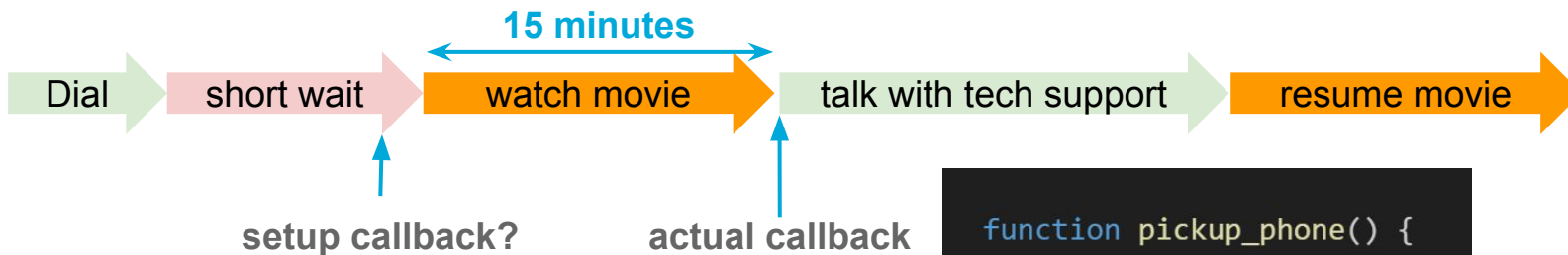
**Async: “out-of-order” execution  
(Order of execution  $\neq$  order of line of code)**

# Async Phone Calls with Callback (in code)

555-4321



1-888-I-CAN-HELP



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

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

15 mins later

*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"); 1  
setup_cb("555-4321", () => { 2  
  // 15 min later  
  talk_with_tech(); 4  
});  
watch_movie(); 3 5
```

*fat arrow*

*Async: order of execution  $\neq$  order of line of code*



What is the execution order?

555-4321

1-888-I-CAN-HELP



15 minutes

Dial

short wait

watch movie

talk with tech support

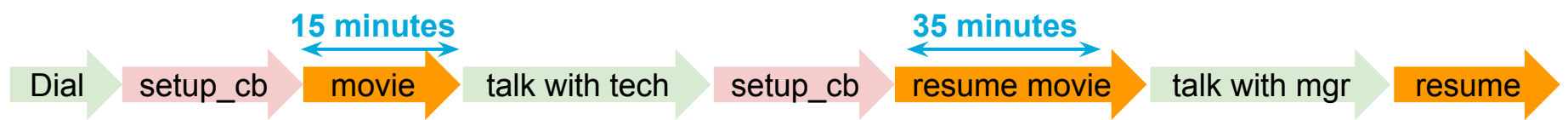
resume movie

setup callback?

actual callback

```
// start dialing ...  
dial("888-I-CAN-HELP"); 1  
// call me back @ 555-4321  
// then hangup to watch movie  
setup_cb("555-4321", () => { 2  
  // 15-min later  
  talk_with_tech(); 4  
});  
// watch it NOW!!!  
watch_movie(); 3 5
```

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



```

dial("888-I-CAN-HELP"); 1
setup_cb("555-4321", () => { 2
  // 15-min later
  // Talk with tech 4
  setup_cb("555-4321", () => { 5
    // 37-min later
    // Talk with manager 7
  });
});
watch_movie(); 3 6 8

```

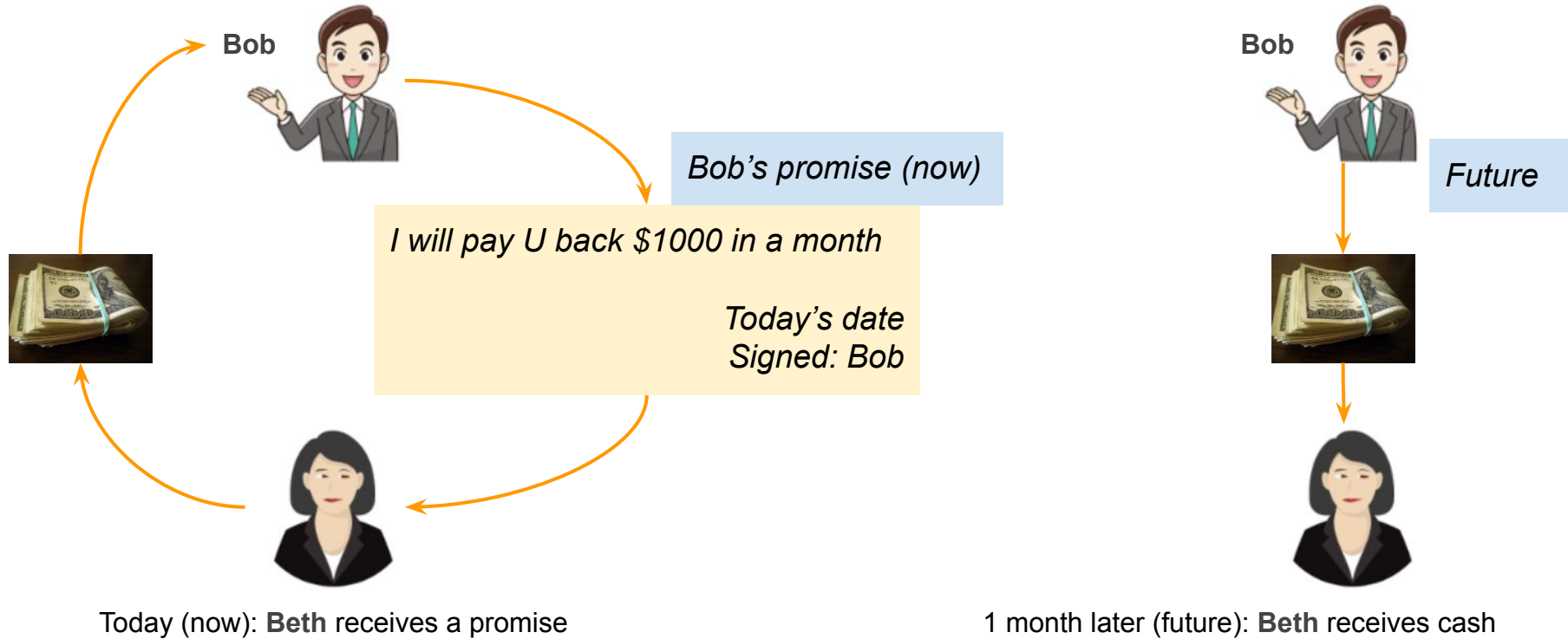
} Nested callbacks

# 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



**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(______);  
}
```

```
console.log("Start");  
const prom = nthPrime(500);  
prom.then((pr: number) => {  
  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()

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

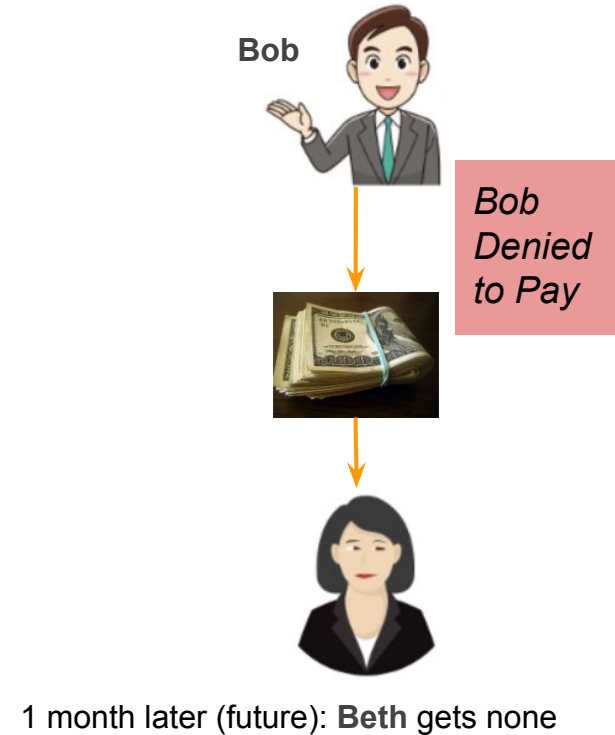
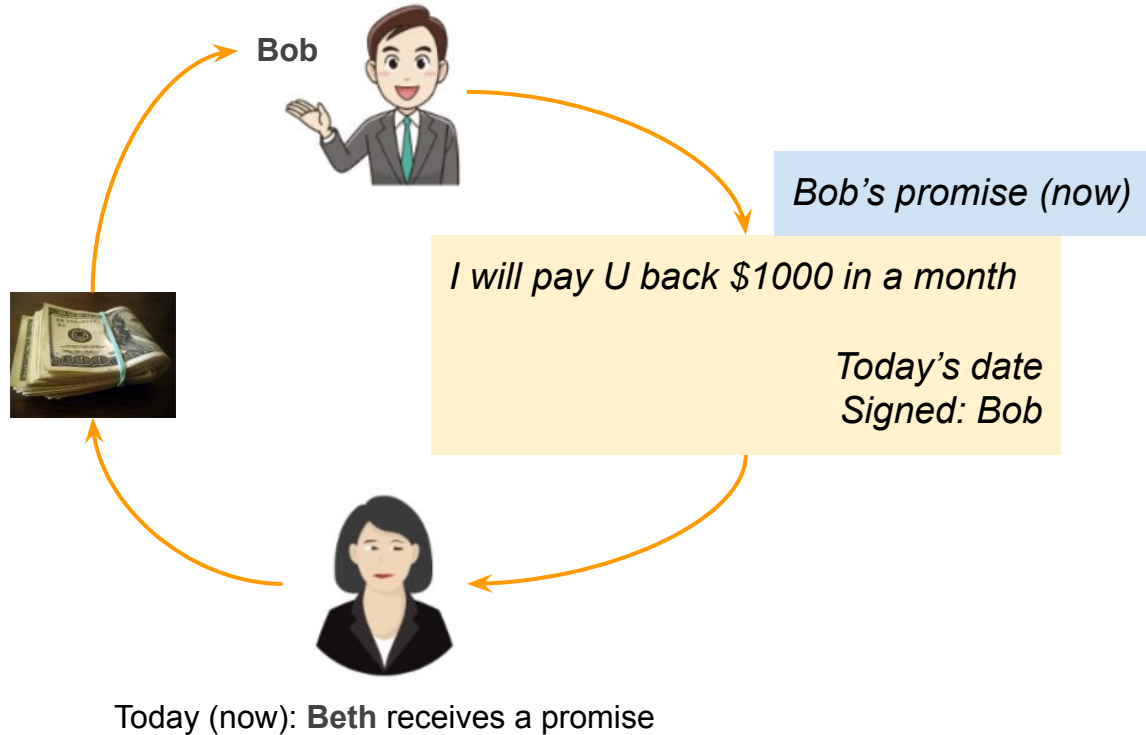
```
console.log("Start");  
const pr = nthPrimeNow(500);  
console.log("The 500th prime is", pr);  
doMoreWork();
```

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

*Compare the order of execution*

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

# Borrowing Money: Promise Now, Never Pay



# 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()`
- Basic static functions
  - `Promise.resolve()`
  - `Promise.reject()`
- Advanced utilities for handling multiple 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 numeral  
  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")

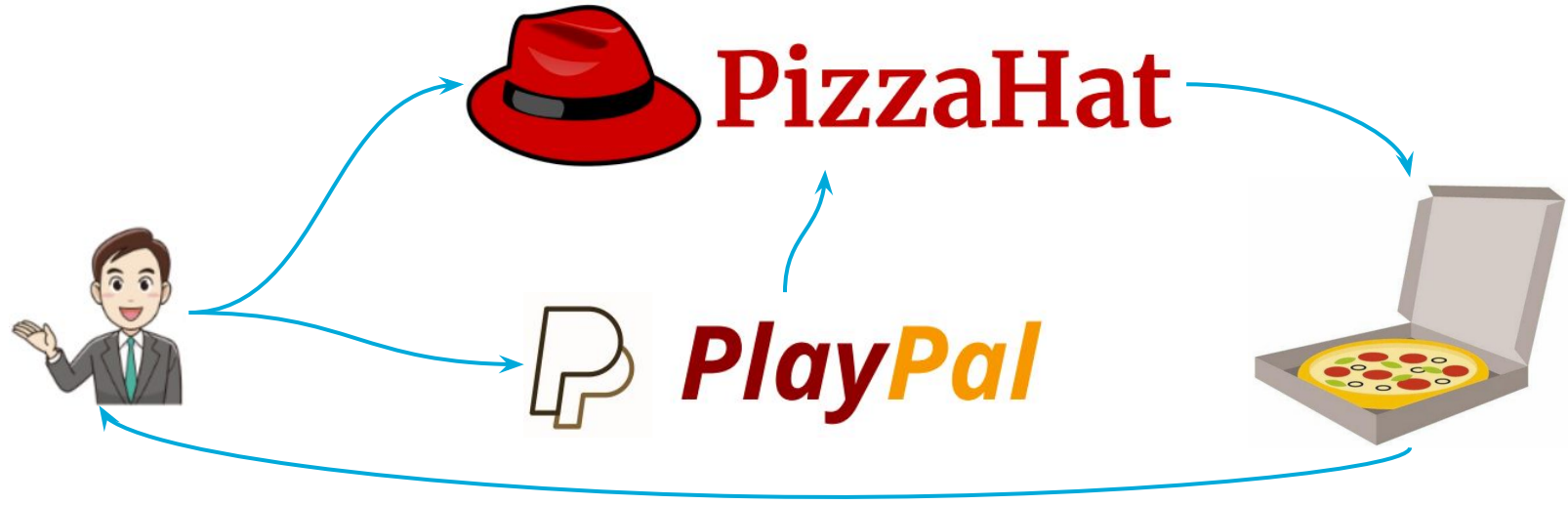
```
function nthPrime(nth: number): Promise<number> {  
  // more code here  
  return Promise.____;  
}
```

```
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 numeral  
  return Promise._____  
}
```

Demo

# Online Pizza Order & 3rd party payment



# Online Pizza Order (code setup)

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

**PizzaHat**

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

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

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

**PizzaHat**

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

```
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");  
  });
```

# 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



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

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

```
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 synchronous 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");
  }
};
```