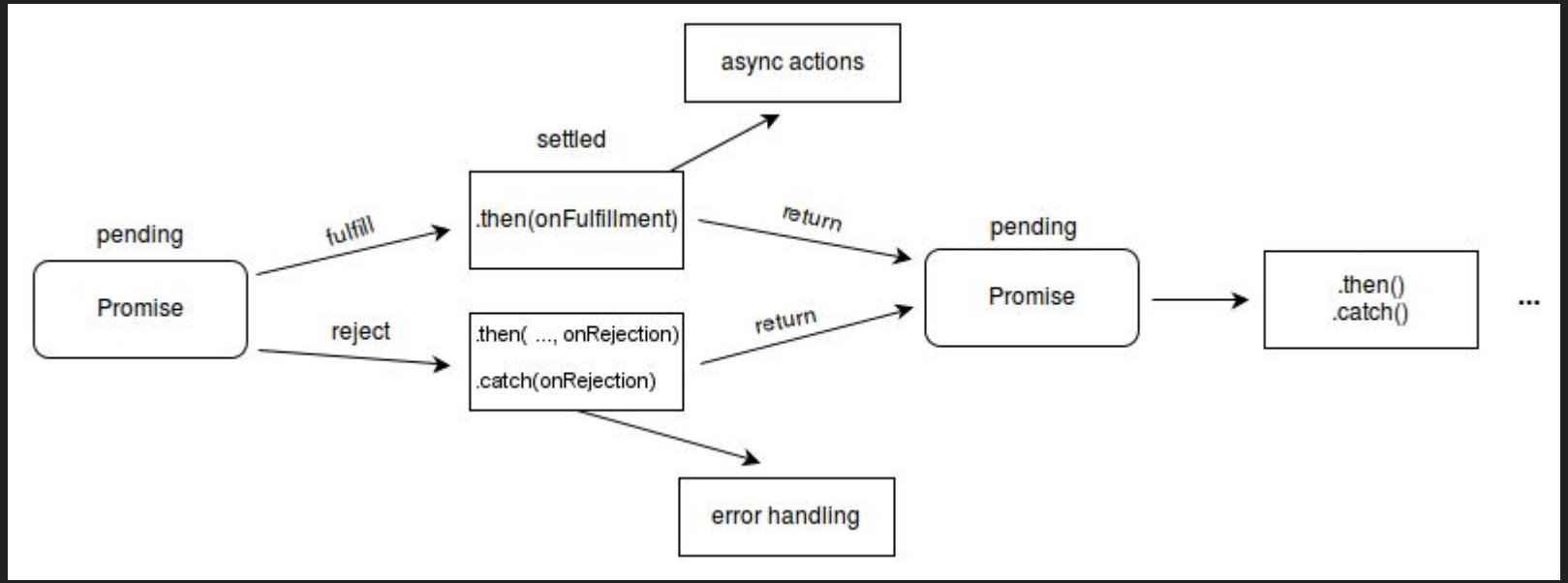
React.

Lesson 2

Promises


async actions
settled
pending
Promise
fulfill
.then(onFulfillment)
return
pending
Promise
.then()
.catch()
...
reject
.then(..., onRejection)
.catch(onRejection)
return
error handling

Promises

Chaining

```
const myPromise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("foo");
  }, 300);
});

myPromise
  .then(handleFulfilledA, handleRejectedA)
  .then(handleFulfilledB, handleRejectedB)
  .then(handleFulfilledC, handleRejectedC);
```

Promises

```
const promiseA = new Promise((resolve, reject) => {
  resolve(777);
});
// At this point, "promiseA" is already settled.
promiseA.then((val) => console.log("asynchronous logging has val:", val));
console.log("immediate logging");

// produces output in this order:
// immediate logging
// asynchronous logging has val: 777
```

Promises

`Promise.all()`

Fulfills when **all** of the promises fulfill; rejects when **any** of the promises rejects.

`Promise.allSettled()`

Fulfills when **all** promises settle.

`Promise.any()`

Fulfills when **any** of the promises fulfills; rejects when **all** of the promises reject.

`Promise.race()`

Settles when **any** of the promises settles. In other words, fulfills when any of the promises fulfills; rejects when any of the promises rejects.

Prototypes

Shadowing

```
const o = {
  a: 1,
  b: 2,
  // __proto__ sets the [[Prototype]]. It's specified here
  // as another object literal.
  __proto__: {
    b: 3,
    c: 4,
  },
};

// o.[[Prototype]] has properties b and c.
// o.[[Prototype]].[[Prototype]] is Object.prototype (we will explain
// what that means later).
// Finally, o.[[Prototype]].[[Prototype]].[[Prototype]] is null.
// This is the end of the prototype chain, as null,
// by definition, has no [[Prototype]].
// Thus, the full prototype chain looks like:
// { a: 1, b: 2 } ---> { b: 3, c: 4 } ---> Object.prototype ---> null

console.log(o.a); // 1
// Is there an 'a' own property on o? Yes, and its value is 1.

console.log(o.b); // 2
```

Prototypes

Chaining

```
const o = {
  a: 1,
  b: 2,
  // __proto__ sets the [[Prototype]]. It's specified here
  // as another object literal.
  __proto__: {
    b: 3,
    c: 4,
    __proto__: {
      d: 5,
    },
  },
};

// { a: 1, b: 2 } ---> { b: 3, c: 4 } ---> { d: 5 } ---> Object.prototype ---> null

console.log(o.d); // 5
```

Performance

1. Do not call for attributes which are located near the top of chain
2. Do not call for attributes which are non-existent in chain, it will cause a full chain traversal.