.then() always returns a new promise. What happens to that promise?

```
promiseB = promiseA.then( [successHandler], [errorHandler] );
```

when

promiseA

is

fulfilled with **data**   *rejected with* **reason**

```
// promiseA is fulfilled with 'hello'

promiseA
  .then( null, myFunc1, myFunc2 )
  .then()
  .then( console.log );

// result: console shows 'hello'
// fulfill bubbled to success handler
```

```
// promiseA is rejected with 'bad request'

promiseA
  .then( myFunc1, null, myFunc2 )
  .then()
  .then( null, console.log );

// result: console shows 'bad request'
// rejection bubbled to error handler
```

and
```
successHandler(data)
errorHandler(reason)
```
and

handler

has no success handler — has a success handler — has an error handler — has no error handler

promiseB.resolve(data);
FULFILLMENT BUBBLED

promiseB.reject(reason);
REJECTION BUBBLED

which

returns **result**   *returns* **promiseZ**   *throws* **err**

promiseB.resolve(result);
PROMISE FOR RESULT

promiseB ← promiseZ
ASSIMILATION

promiseB.reject(err);
ERROR CAUGHT

```
// promiseForVal1 is fulfilled with 999

promiseForVal2 = promiseForVal1
  .then( function success (val1) {
    val2 = val1++;
    return val2;
  });

// result: promiseForVal2 is fulfilled with 1000.

// same idea, shown in a direct chain:

promiseForVal1
  .then( function success (val1) {
    // do some code to make val2
    return val2;
  })
  .then( function success (val2) {
    console.log( val2 );
  });
```

```
// output promise "becomes" returned promise

promiseForMessages = promiseForUser
  .then( function success (user) {
    // do some code to make a new promise
    return promiseForMessages;
  });

// same idea, shown in a direct chain:

promiseForUser
  .then( function success (user) {
    // do some code to make a new promise
    return promiseForMessages;
  })
  .then( function success (messages) {
    console.log( messages );
  });
```

```
// promiseForVal2 will be rejected with error '404':

promiseForVal2 = promiseForVal1
  .then( function success (val1) {
    // THROWN ERROR '404' trying to make val2
    return val2;
  });

// same idea, shown in a direct chain:

promiseForVal1
  .then( function success (val1) {
    // THROWN ERROR '404' trying to make val2
    return val2;
  })
  .then( null, function failed (err) {
    // handle the error as you prefer
    console.log('Oops!', err);
  });

// console shows 'Oops! 404'
```