

Promise Structure

```
// async/await  
const dog = await Dog.findById(1)  
console.log(dog)
```

```
// async/await
const dog = await Dog.findById(1)
console.log(dog)
```

```
// promises
Dog.findById(1)
  .then(dog => {
    console.log(dog)
  })
```

```
// async/await
try {
  const dog = await Dog.findById(1)
  console.log(dog)
} catch (err) {
  console.log(err)
}
```

```
// async/await
try {
  const dog = await Dog.findById(1)
  console.log(dog)
} catch (err) {
  console.log(err)
}
```

```
// promises
Dog.findById(1)
  .then(dog => {
    console.log(dog) // success
  })
  .catch(err => {
    console.log(err) // err
  })
```

.then

- **Accepts up to two arguments (both technically optional)**
 - “Success” callback
 - “Error” callback
- **If the promise resolves (succeeds)**
 - “Success” callback is invoked with the value
- **If the promise rejects (fails)**
 - “Error” callback is invoked with the value

.then

- You can attach as many of these as you want, whenever you want

.then

- You can attach as many of these at you want, whenever you want

```
const promiseForDog = Dog.findAll()

promiseForDog.then(dog => {
  console.log('Got a dog over here: ', dog)
})

promiseForDog.then(dog => {
  console.log('Dog once again: ', dog)
})
```


Promise “chaining”

Promise chaining

- ◉ What if we want to do things in order?
 - I want to *this thing*, and **then** I want to do this other thing!
- ◉ Achieved by chaining promises
- ◉ The trick: every call to `.then` returns a new promise!

```
const p1 = Dog.findById(1)
```

```
const p1 = Dog.findById(1)
p1.then(dog => {
})
```

```
const p1 = Dog.findById(1)
const p2 = p1.then(dog => {
    })
```

```
const p1 = Dog.findById(1)
const p2 = p1.then(dog => {
    })
```

```
// q: what is p2 a promise for?
```

```
const p1 = Dog.findById(1)
const p2 = p1.then(dog => {
    // a: whatever we return
    // from this callback!
})
```

```
const p1 = Dog.findById(1)
const p2 = p1.then(dog => {
    return 5
})
```

```
p2.then(result => {
    console.log(result) // 5
})
```



```
const p1 = Dog.findById(1)
const p2 = p1.then(dog => {
    return dog.update()
})

p2.then(result => {
    console.log(result) // ?
})
```

```
const p1 = Dog.findById(1)
const p2 = p1.then(dog => {
    return dog.update()
})

p2.then(result => {
    console.log(result) // the updated dog!
})
```

```
const p1 = Dog.findById(1)
const p2 = p1.then(dog => {
    return dog.update()
})

p2.then(result => {
    console.log(result) // the updated dog!
})
```

// if a .then returns a promise, it is “flattened”

```
Dog.findById(1)
  .then(dog => {
    return dog.update()
  })
  .then(result => {
    console.log(result) // the updated dog!
  })
```

```
Dog.findById(1)
  .then(dog => {
    return dog.update() // what if this fails?
  })
  .then(result => {
    console.log(result)
  })
```

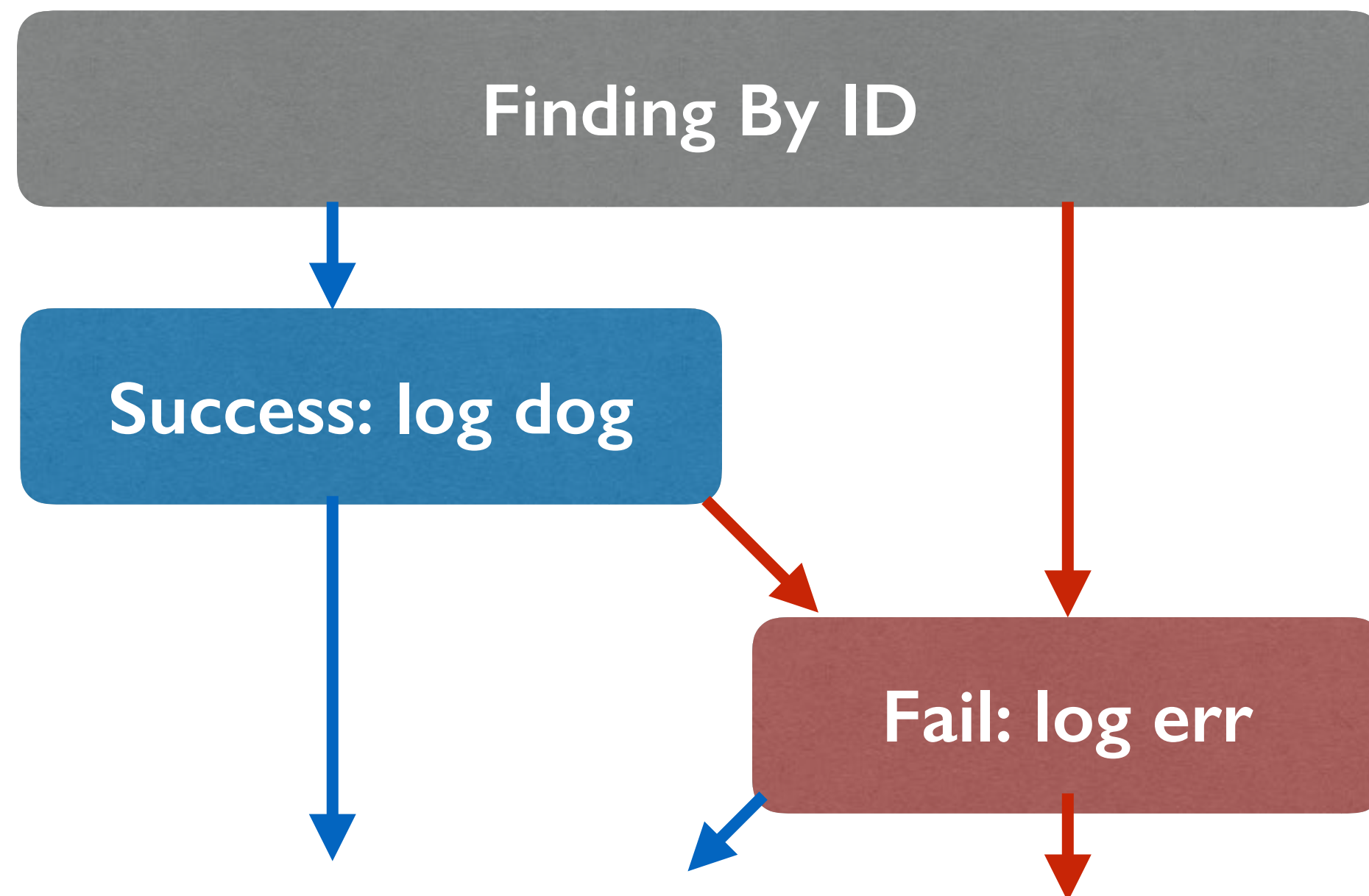
```
Dog.findById(1)
  .then(dog => {
    return dog.update() // what if this fails?
  })
  .then(result => {
    console.log(result)
  })
  .catch(err => {
    console.error(err)
  })
```

.catch

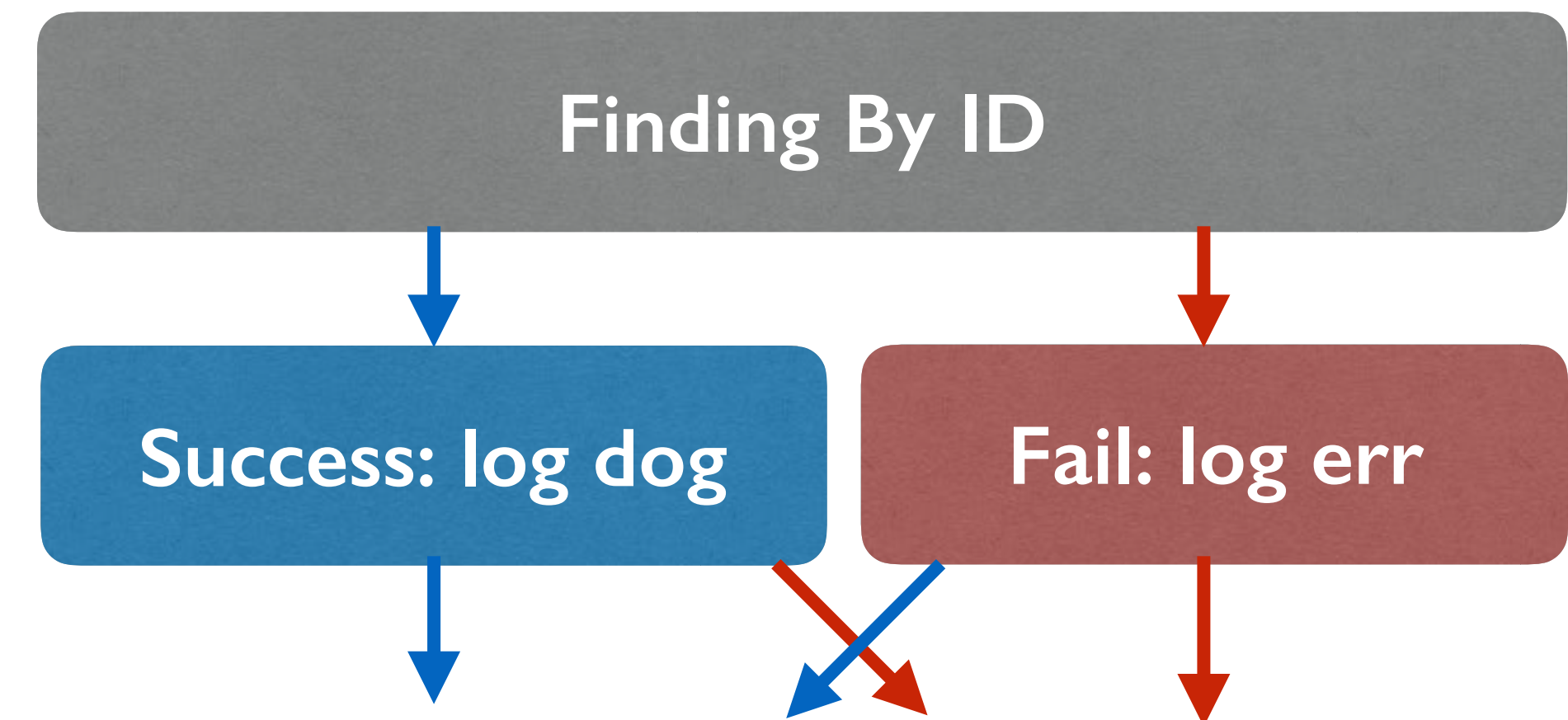
- Just like .then, but only accepts an error handler
- In most cases you can use .then for success handlers, and .catch for error handlers
- Rejection will “bubble down” to the first error handler



These Are Different!



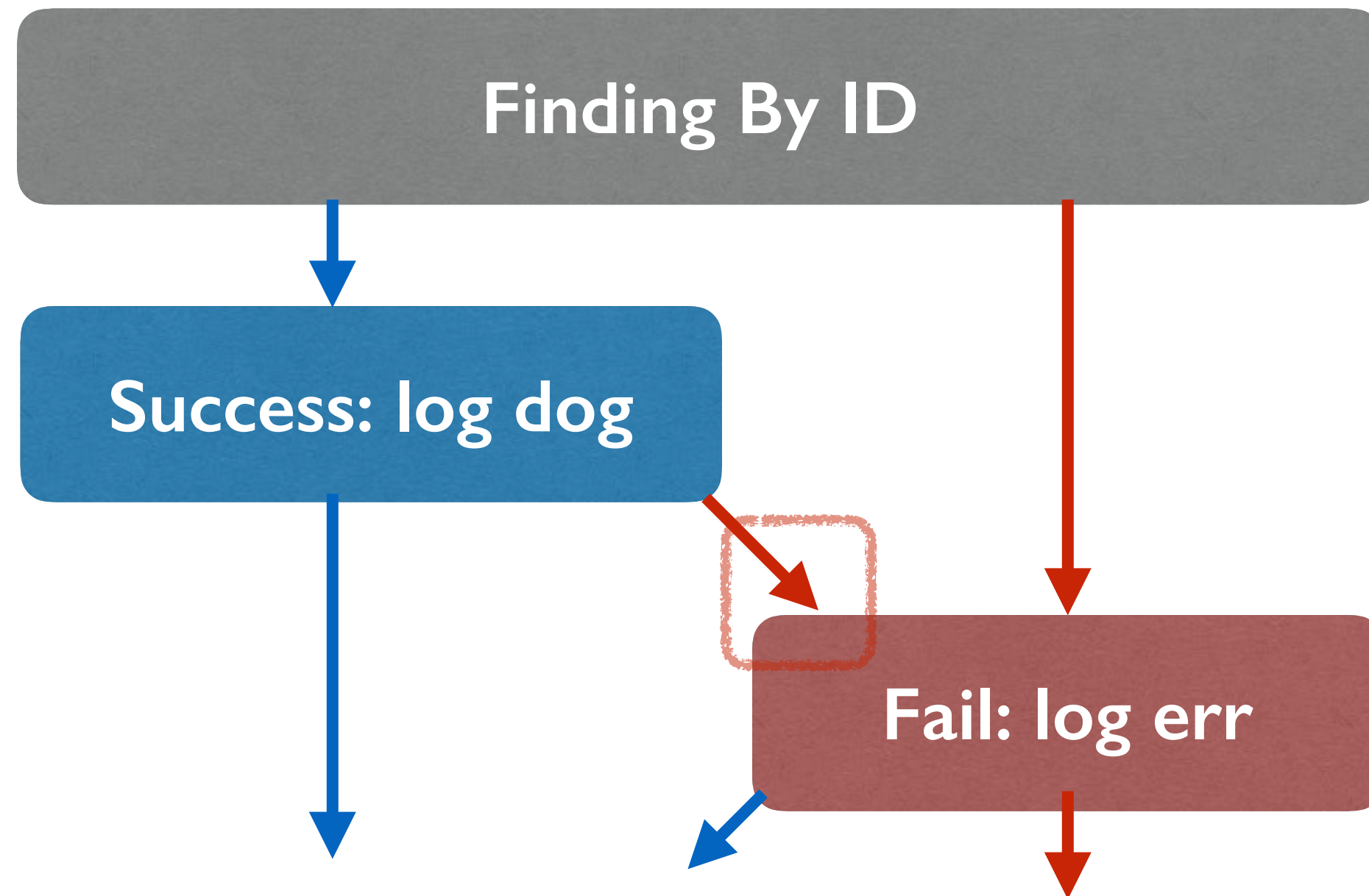
```
Dog.findById(1)
  .then(dog => {
    console.log(dog)
  })
  .catch(err => {
    console.log(err)
  })
```



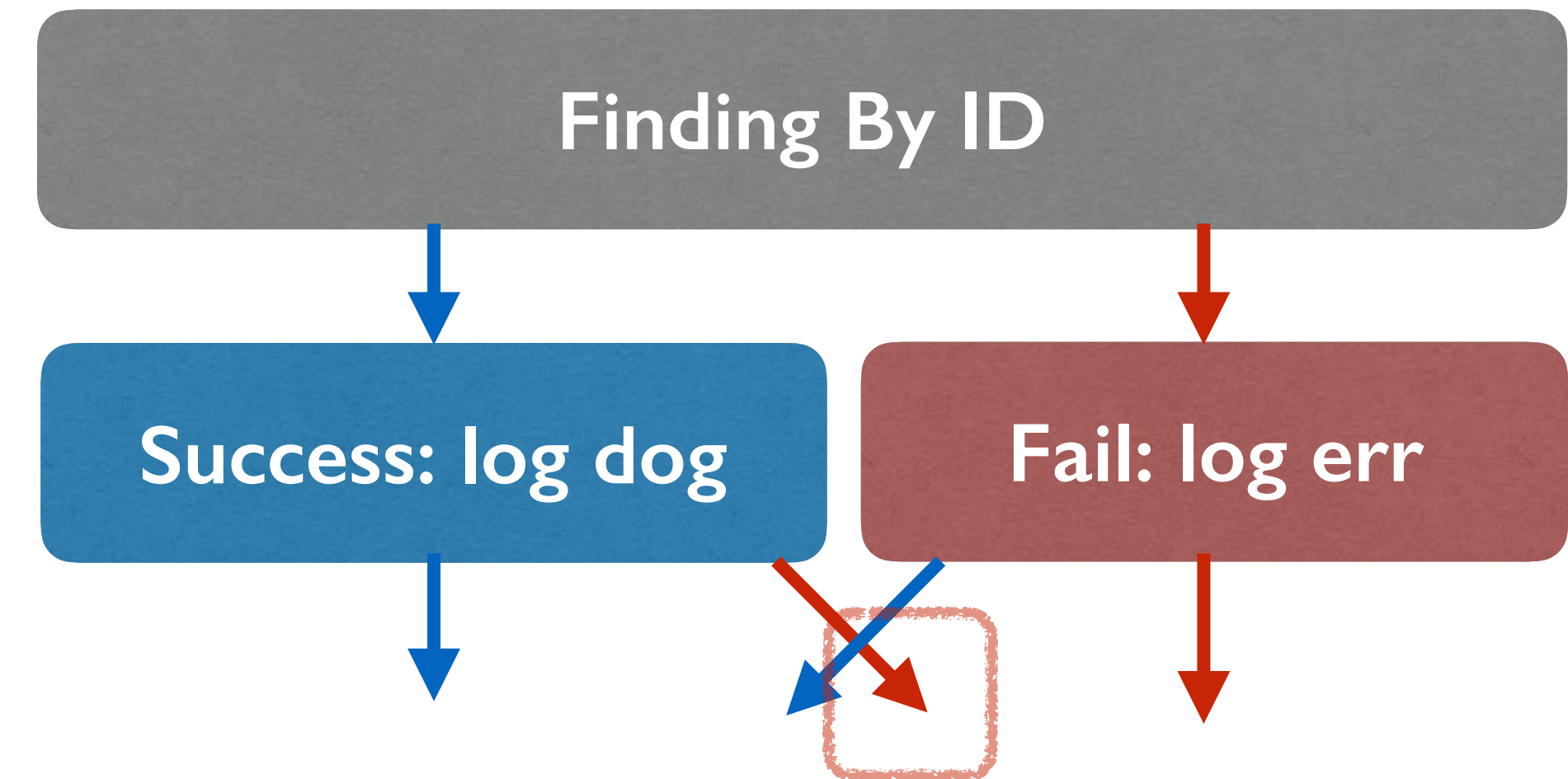
```
Dog.findById(1)
  .then(dog => {
    console.log(dog)
  }, err => {
    console.log(err)
  })
```




These Are Different!



```
Dog.findById(1)
  .then(dog => {
    console.log(dog)
  })
  .catch(err => {
    console.log(err)
  })
```



```
Dog.findById(1)
  .then(dog => {
    console.log(dog)
  }, err => {
    console.log(err)
  })
```



Success

PI

```
Dog.findById(1)
  .then(dog => {
    return dog.update()
  })
  .then(result => {
    console.log(result)
  })
  .catch(err => {
    console.error(err)
  })
```



Success

```
Dog.findById(1)
  .then(dog => {
    return dog.update()
  })
  .then(result => {
    console.log(result)
  })
  .catch(err => {
    console.error(err)
  })
```

PIdog



Success

```
Dog.findById(1)
  .then(dog => {
    return dog.update()
  })
  .then(result => {
    console.log(result)
  })
  .catch(err => {
    console.error(err)
  })
```





Success

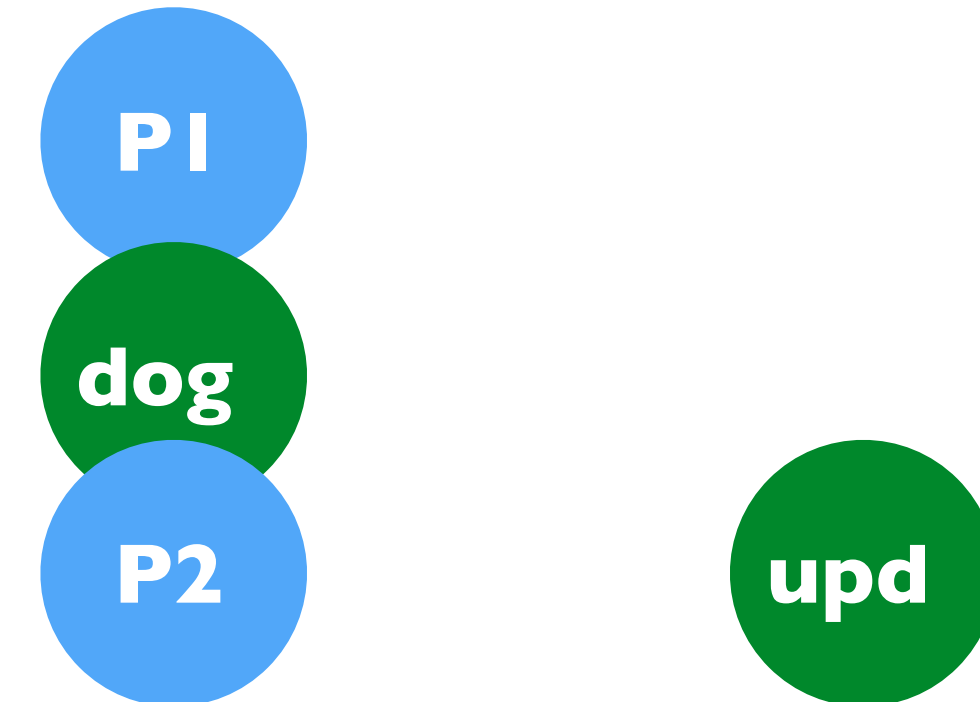
```
Dog.findById(1)
  .then(dog => {
    return dog.update()
  })
  .then(result => {
    console.log(result)
  })
  .catch(err => {
    console.error(err)
  })
```





Success

```
Dog.findById(1)
  .then(dog => {
    return dog.update()
  })
  .then(result => {
    console.log(result)
  })
  .catch(err => {
    console.error(err)
  })
```





Success

```
Dog.findById(1)
  .then(dog => {
    return dog.update()
  })
  .then(result => {
    console.log(result)
  })
  .catch(err => {
    console.error(err)
  })
```





Error

PI

```
Dog.findById(1)
  .then(dog => {
    return dog.update()
  })
  .then(result => {
    console.log(result)
  })
  .catch(err => {
    console.error(err)
  })
```




Error

```
Dog.findById(1)
  .then(dog => {
    return dog.update()
  })
  .then(result => {
    console.log(result)
  })
  .catch(err => {
    console.error(err)
  })
```

PIdog



Error

```
Dog.findById(1)
  .then(dog => {
    return dog.update()
  })
  .then(result => {
    console.log(result)
  })
  .catch(err => {
    console.error(err)
  })
```





Error

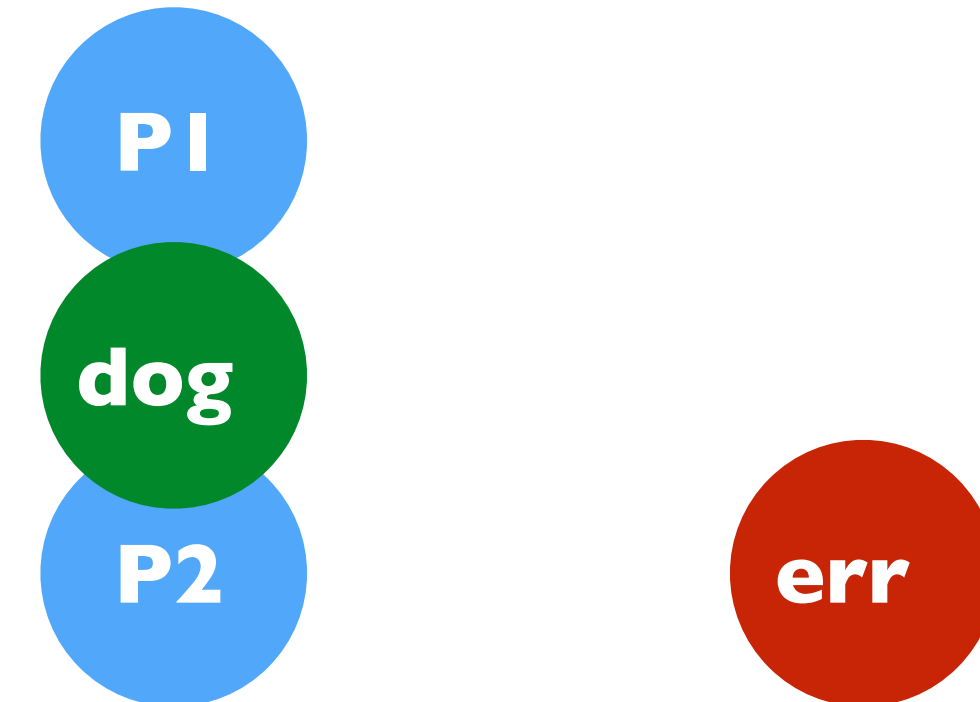
```
Dog.findById(1)
  .then(dog => {
    return dog.update()
  })
  .then(result => {
    console.log(result)
  })
  .catch(err => {
    console.error(err)
  })
```





Error

```
Dog.findById(1)
  .then(dog => {
    return dog.update()
  })
  .then(result => {
    console.log(result)
  })
  .catch(err => {
    console.error(err)
  })
```





Error

```
Dog.findById(1)
  .then(dog => {
    return dog.update()
  })
  .then(result => {
    console.log(result)
  })
  .catch(err => {
    console.error(err)
  })
```

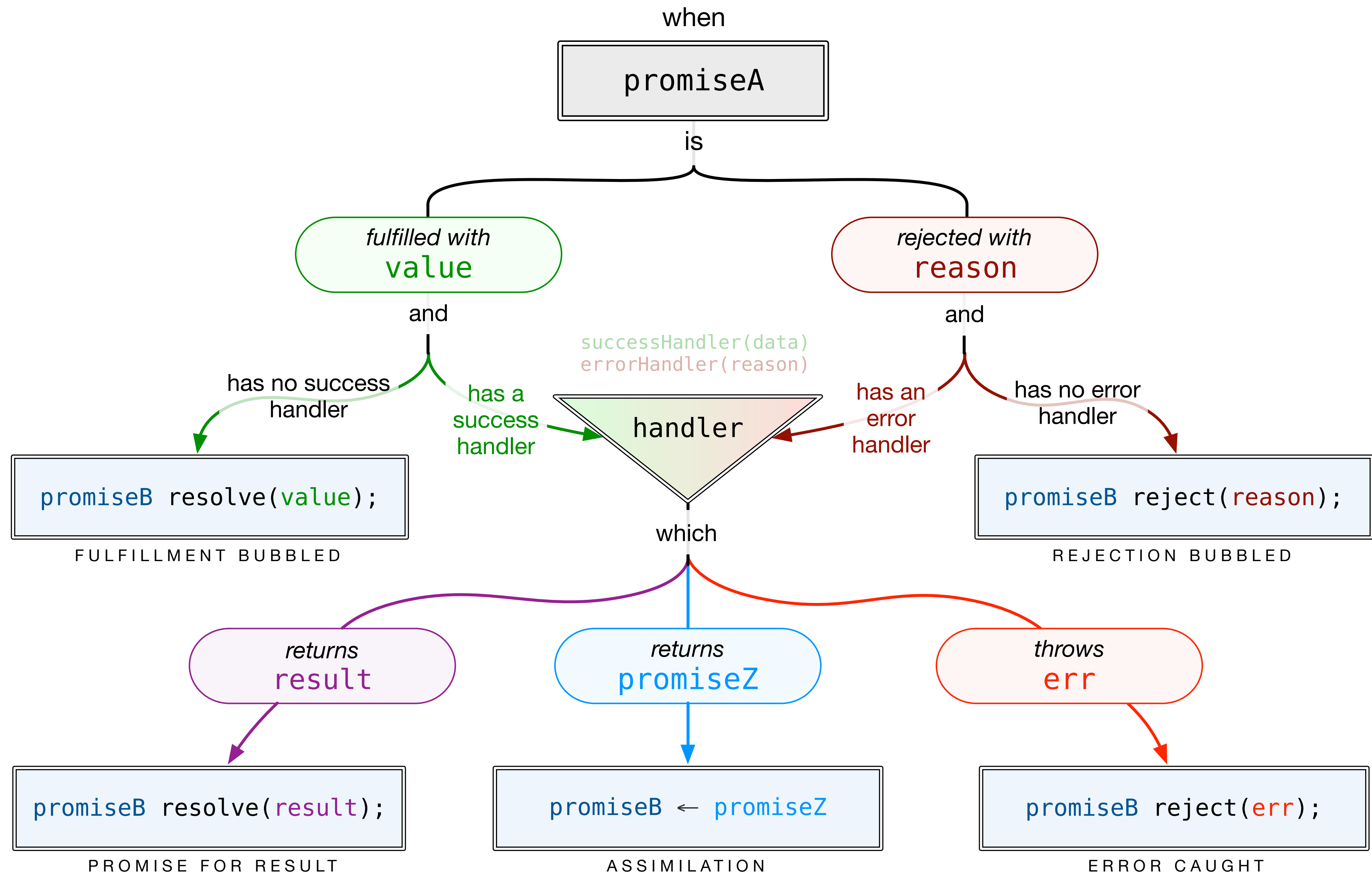
P1

dog

P2

err

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





External Resources for Further Reading

- [Kris Kowal & Domenic Denicola: Q](#) (great examples & resources)
- [The Promises/A+ Standard](#) (with use patterns and an example implementation)
- [We Have a Problem With Promises](#)
- [HTML5 Rocks: Promises](#) (deep walkthrough with use patterns)
- [DailyJS: Javascript Promises in Wicked Detail](#) (build an ES6-style implementation)
- [MDN: ES6 Promises](#) (upcoming native functions)
- [Promise Nuggets](#) (use patterns)
- [Promise Anti-Patterns](#)