### **Promise**

Promise là một *cơ chế* trong JavaScript giúp bạn thực thi các tác vụ bất đồng bộ mà không rơi vào *callback hell* hay *pyramid of doom*, là tình trạng các hàm callback lồng vào nhau ở quá nhiều tầng.

### **Tạo nhanh Promise với Promise.resolve() và Promise.reject()**

const p = Promise.resolve(12)

.then(result => console.log(result)) // 12

.then(res => Promise.reject(new Error('Dừng lại nhanh')))

.then(() => 'Cười thêm phát nữa là tym anh đứt phanh')

.catch(err => console.error(err)) // Error: Dừng lại nhanh

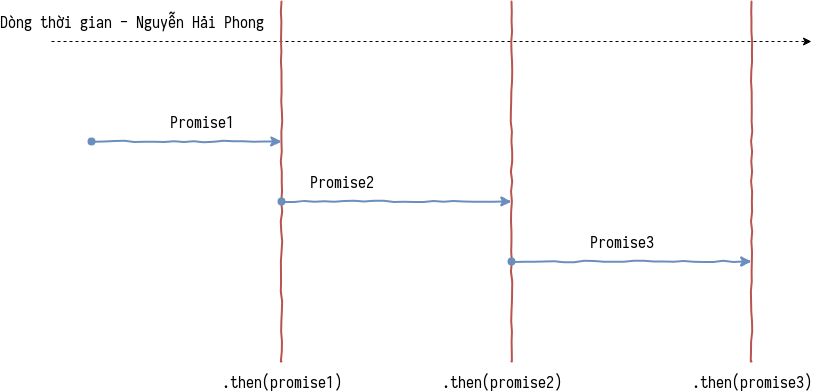
## **async/await**

async/await là một *cơ chế* giúp bạn thực hiện các thao tác bất đồng bộ một cách *tuần tự* hơn. Async/await vẫn sử dụng Promise

## **Luôn đưa vào .then() một hàm**

## Phương thức .then đòi hỏi tham số của nó phải là một hàm. Nếu bạn đưa vào .then() một giá trị, nó sẽ bị bỏ qua

## **Chạy các Promise tuần tự**

****

muốn chạy các promises một cách tuần tự như sơ đồ ở trên, bạn có thể dùng hàm Array.prototype.reduce .

## **Chạy nhiều Promises cùng lúc với Promise.all()**

muốn thực thi và lấy ra kết quả của nhiều promises cùng lúc dùng vòng lặp, hoặc .forEach

## **Promise.race()**

chạy tuần tự và song song có Promise.race([promise1, promise2, ...]). Phương thức này nhận vào một mảng các promises và sẽ resolve/reject ngay khi một trong số các promises này hoàn thành/xảy ra lỗi.

## **Phân biệt .then(resolve, reject) và .then(resolve).catch(reject)**

Hàm reject trong .then(resolve, reject) chỉ có thể chụp được lỗi từ những .then() phía trước nó, mà không thể bắt được lỗi xảy ra trong hàm resolve cùng cấp.

## **Promise không lazy**

## 