Mechanism: cơ chế, periodically: định kỳ, frequently: thường xuyên.

# Synchronous và Asynchronous

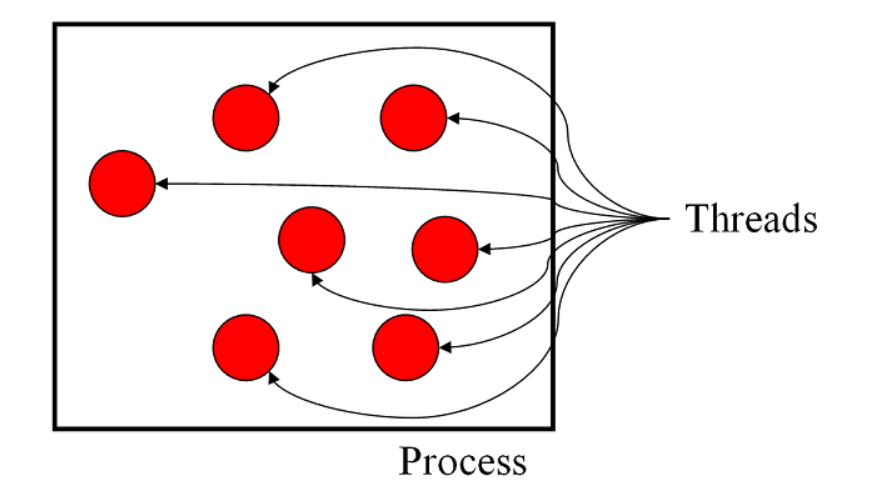
# Synchronous: là cách lập trình mà các hoạt động của chương trình sẽ thực hiện tuần tự.

# 

# Như ở hình minh họa trên nói về việc chuẩn bị bữa sáng. Có tất cả 7 hoạt động --> nếu ta quy định mỗi hoạt động là 1 function thì ta có 7 function.

# 

* **Ưu điểm của lập trình đồng bộ**: Các hoạt động xảy ra tuần tự nên có thể dễ quản lý, dễ debug và phát hiện vấn đề khi xảy ra lỗi.
* **Vấn đề**: Nếu thời gian chạy các hoạt động đồng bộ xảy ra quá dài thì UI sẽ bị lock (đứng chương trình) do UI và các hoạt động đồng bộ cùng thuộc về 1 thread.
* **Thread là gì**: như hình minh họa ở trên 1 process (hay còn được hiểu là 1 chương trình) sẽ bao gồm nhiều threads. Mỗi thread có thể thực hiện các hoạt động khác nhau.

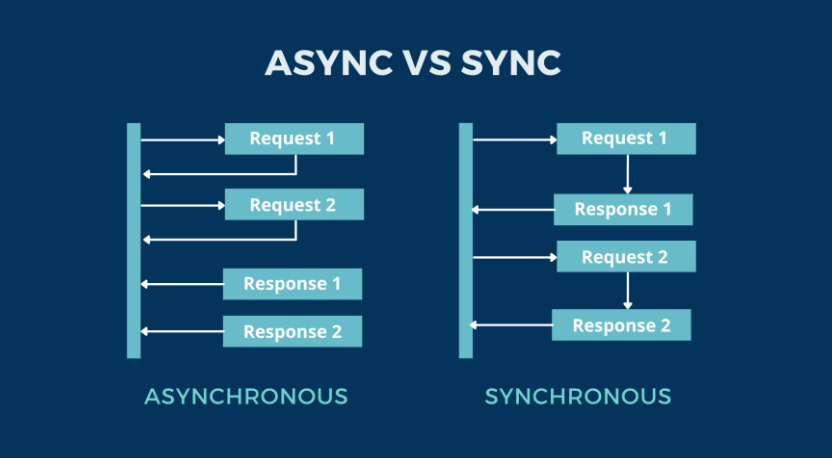


Ví dụ: Khi ta mở 1 ứng dụng web nghe nhạc. Ta vừa có thể play nhạc, vừa có thể làm các hành động khác: pause, next, back, view list songs... bởi vì mỗi hành động là một thread (một luồng riêng biệt).

**Asynchronous:** Lập trình bất đồng bộ là cách lập trình cho phép các hoạt động thực hiện không theo tuần tự. Có thể các đoạn code ở dưới chạy trước đoạn code viết ở phía trên (bất đồng bộ), các hoạt động không phải đợi lẫn nhau.

* **Ưu điểm của lập trình bất đồng bộ**: Bởi vì không bị block thread và các hoạt động có thể không phải đợi nhau nên khi xử lý các tác vụ có thời gian thực hiện lâu không bị đứng chương trình, đem lại trải nghiệm người dùng tốt.
* **Vấn đề**: Bởi vì các hoạt động thực hiện không theo thứ tự nên ta phải quản lý các hành động này một cách cẩn thận. Ví dụ như khi bạn xử lý hành động submit 1 form, đầu tiên phải validate dữ liệu, sau đó mới tới phần xử lý. Nếu như quản lý không tốt phần bất đồng bộ có thể dẫn tới: Phần xử lý chạy trước cả phần validate dữ liệu ---> có thể gây ra lỗi nghiêm trọng.

**Asynchronous trong js**: JS là ngôn ngữ chỉ chạy được single thread, nó chỉ có thể làm 1 việc tại một thời điểm, vì vậy nếu ta có quá nhiều việc phải làm thì dễ bị block browser.



**JavaScript Scheduling: setTimeout and setInterval**

* Một block mã Javascript thường được thực thi đồng bộ. Tuy nhiên có một số hàm Javascript (timers) cho phép chúng ta trì hoãn việc thực thi của các hướng dẫn tùy ý.
* setTimeout ()
* setInterval ()
* Hai hàm này cho phép bạn thực hiện một đoạn mã Javascript tại một thời điểm nào đó trong tương lai. Nó được gọi là "lập lịch một cuộc gọi" (scheduling a call).
* SetTimeout (): Hàm setTimeout () thường được sử dụng nếu bạn muốn hàm của mình thực thi bao nhiêu mili giây kể từ khi gọi method setTimeout ().



* Trong đó, expression là đoạn mã JavaScript được thực thi sau khi timeout chạy hết số mili giây đã thiết lập.
* Điểm lưu ý ở đây là setTimeout () sẽ không dừng việc thực thi các lệnh tiếp theo trong thời gian chờ. Nó chỉ lập lịch cho đoạn mã JavaScript được chỉ định sẽ thực thi vào thời điểm được thiết lập trước đó. Sau khi gọi hàm setTimeout (), các lệnh tiếp tục chạy bình thường với timer chạy ở chế độ nền.
* SetInterval (): như cái tên của nó, hàm này sẽ thường được sử dụng để thiết lập độ trễ cho các hàm sẽ được thực hiện lặp lại như là hiệu ứng.
* Điểm khác nhau giữa setTimeout và setInterval là ST chỉ kích hoạt expression một lần trong khi SI sẽ kích hoạt expression sau một khoảng thời gian nhất định (trừ khi bạn cho nó dừng lại).
* Để dừng hoạt động của hàm Interval ta sử dụng hàm clearInterval (timerId)



**Change Detection**

Là quá trình mà qua đó Angualar check to see khi app state có sự thay đổi, và nếu bất cứ DOM nào cần được updated. Ở level cao, Angular walks component của bạn từ top 🡪 bottom, để tìm thay đổi. Angular chạy cơ chế phát hiện thay đổi của nó theo định kỳ vì vậy các thay đổi từ data model reflected ở trong app view.

Change Detection được tối ưu hóa và hiệu suất cao, nhưng nó có thể làm chậm nếu app chạy nó quá thường xuyên.

Resolving zone pollution: Zone.js là cơ chế báo hiệu mà Angular dùng để phát hiện a state change. Nó nắm bắt các hoạt động không đồng bộ như SetTimeOut, netWork request, event listeners. Phát hiện lịch trình Angular thay đổi dựa trên Zone.js.

Trong một số trường hợp các task or microtask không make any change trong data model, điều này làm cho việc chạy change detection là không cần thiết. Phổ biến như:

* RequestAnimationFrame, setTimeout or setInterval
* Click mouse move
* HTTP call is completed

Identifying unnecessary change detection calls: Phần này giúp ta chạy code outside Angular để tránh các cuộc gọi phát hiện không cần thiết.

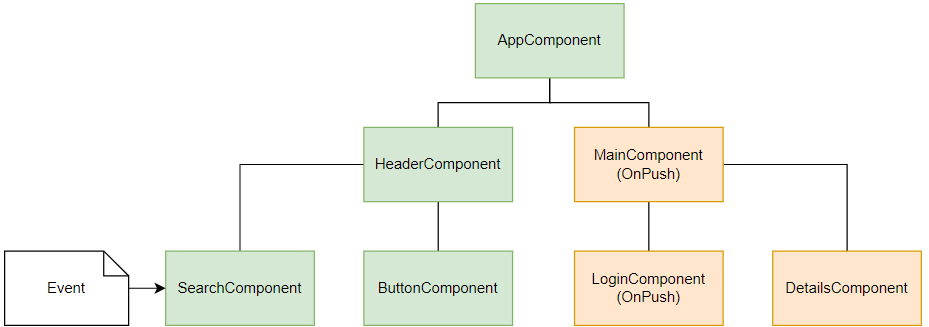
**OnPush:** onPush dùng để bỏ qua phát hiện thay đổi trong toàn bộ component subtree change detection chỉ run change detection subtree chỉ khi:

* Root component of the subtree nhận vào new input là kết quả của một template binding. Angular so sánh current và past value of input with “== “.
* Angular xử lý một event (event binding, output Binding, @Hostlistener) trong subtree root component, hoặc bất cứ phần tử con nào của nó cho dù chúng có đang sử dụng onPush hay không.

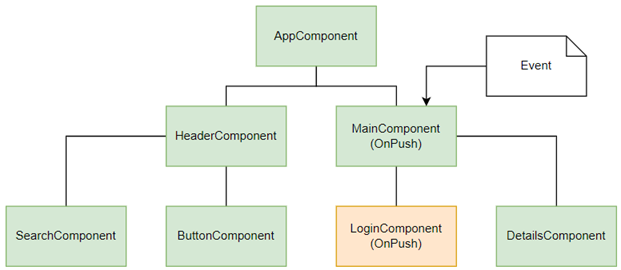
**Common Change Detection scenarios:**

* **Một sự kiện được xử lý bởi một component với defult change detection**: Nếu Angular xử lý một sự kiện trong một component không có strategy OnPush, khung sẽ thực hiện phát hiện thay đổi trên toàn bộ cây thành phần. Angular sẽ bỏ qua các cây con thành phần con cháu có gốc bằng OnPush, vốn chưa nhận được đầu vào mới.

Ví dụ: nếu chúng ta đặt chiến lược phát hiện thay đổi của MainComponent thành OnPush và người dùng tương tác với một thành phần bên ngoài cây con bằng gốc MainComponent, thì Angular sẽ kiểm tra tất cả các thành phần màu xanh lục từ sơ đồ bên dưới (AppComponent, HeaderComponent, SearchComponent, ButtonComponent) trừ khi MainComponent nhận đầu vào mới:

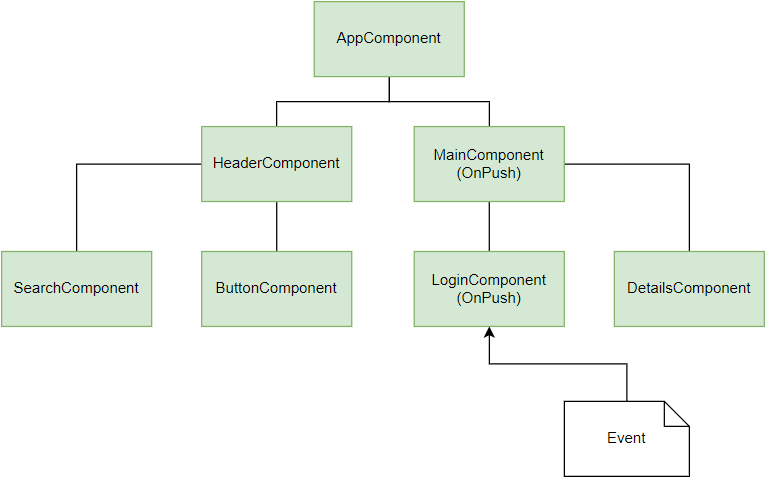


* **Một event được xử lý bởi 1 component với Onpush:** Nếu Angular xử lý một sự kiện trong một thành phần bằng chiến lược OnPush, khung sẽ thực hiện phát hiện thay đổi trong toàn bộ cây thành phần. Angular sẽ bỏ qua các cây con thành phần có gốc bằng OnPush, vốn chưa nhận được đầu vào mới và nằm ngoài thành phần xử lý sự kiện.

Ví dụ: nếu Angular xử lý một sự kiện trong MainComponent, khung sẽ chạy phát hiện thay đổi trong toàn bộ cây thành phần. Góc sẽ bỏ qua cây con có gốc LoginComponent vì nó có OnPush và sự kiện xảy ra ngoài phạm vi của nó.

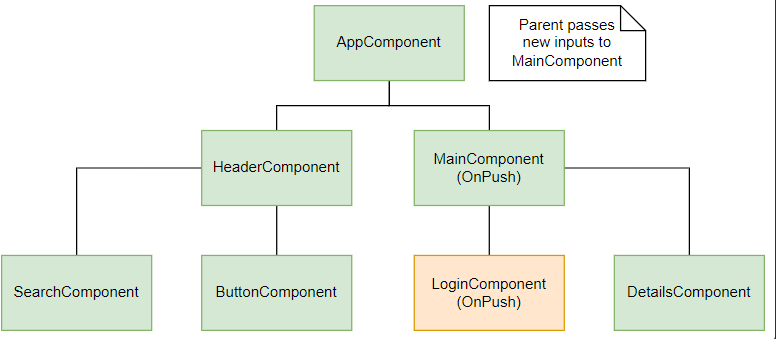
* **Một sự kiện được xử lý bởi phần tử con của một thành phần với OnPush:** Nếu Angular xử lý một sự kiện trong một thành phần bằng OnPush, khung sẽ thực hiện phát hiện thay đổi trong toàn bộ cây thành phần, bao gồm cả tổ tiên của thành phần đó.

Ví dụ, trong sơ đồ bên dưới, Angular xử lý một sự kiện trong LoginComponent sử dụng OnPush. Angular sẽ gọi phát hiện thay đổi trong toàn bộ cây con thành phần bao gồm MainComponent (cha mẹ của LoginComponent), mặc dù MainComponent cũng có OnPush. Góc cũng kiểm tra MainComponent vì LoginComponent là một phần của chế độ xem của nó.



* **Đầu vào mới cho thành phần với OnPush:** Angular sẽ chạy phát hiện thay đổi trong một thành phần con với OnPush khi đặt thuộc tính đầu vào do template binding.

Ví dụ: trong sơ đồ bên dưới, AppComponent chuyển đầu vào mới cho MainComponent, MainComponent có OnPush. Angular sẽ chạy phát hiện thay đổi trong MainComponent nhưng sẽ không chạy phát hiện thay đổi trong LoginComponent, vốn cũng có OnPush, trừ khi nó cũng nhận được đầu vào mới.



**Observables:** Nó truyền dữ liệu từ thành phần nhỏ nó không gửi toàn bộ data trên 1 lần (như kiểu bạn cần sử dụng 1 xô nước để tưới cây, bạn subcrise (lắp vòi nc cho ống nc) **observer** như là vòi nước bạn sẽ nhận nc dần đầy xô mà bạn cần sử dụng, khi bạn không sử dụng nữa thì bạn unSub (như là đóng vòi nước), chứ nó không đưa cho bạn cả xô nc từ đầu như promise) (THEO CÁCH EM HIỂU).

Observerable không chỉ phát ra data mà còn có thể phát ra error