

Pull based

```
// I want some toast. I will pull it out of the toaster when it is done.
```

```
// I will do nothing until the toast is available.
```

```
Toast t = toaster.MakeToast();
```

```
t.AddJam();
```

Push based

```
// I want some toast. But it might take a while and I can do more work
```

```
// while I am waiting:
```

```
Task<Toast> task = toaster.MakeToastAsync();
```

```
Toast t = await task;
```

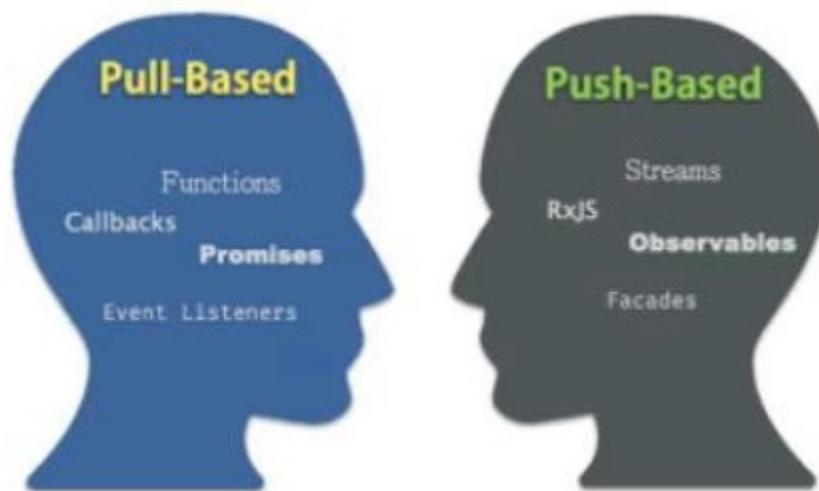
```
// await returns to the caller if the toast is not ready, and assigns
```

```
// a callback. When the toast is ready, the callback causes this method
```

```
// to start again from this point:
```

```
t.AddJam();
```

<https://stackoverflow.com/questions/51254117/what-is-difference-between-push-based-and-pull-based-structures-like-ienumerable>



Iterable<T>
pull

T next()
throws Exception
returns;

Observable<T>
push

onNext(T)
onError(Exception)
onCompleted()



```
// Iterable<String> or Stream<String> ←  
// that contains 75 Strings  
getDataFromLocalMemory()  
  .skip(10)  
  .limit(5)  
  .map(s -> s + "_transformed")  
  .forEach(t -> System.out.println("onNext => " + t))
```

```
// Observable<String>  
// that emits 75 Strings  
getDataFromNetwork()  
  .skip(10)  
  .take(5)  
  .map(s -> s + "_transformed")  
  .forEach(t -> System.out.println("onNext => " + t))
```

<https://www.uwanttolearn.com/android/pull-vs-push-imperative-vs-reactive-reactive-programming-android-rxjava2-hell-part2/>

Imperative Approach:

```
int val1 = 10;  
int val2 = 20;  
int sum = val1 + val2;  
System.out.println(sum); // 30  
val1 = 5;  
System.out.println(sum); // 30
```

Reactive Approach:

```
int val1 = 10;  
int val2 = 20;  
int sum = val1 + val2;  
System.out.println(sum); // 30  
val1 = 5;  
System.out.println(sum); // 25
```