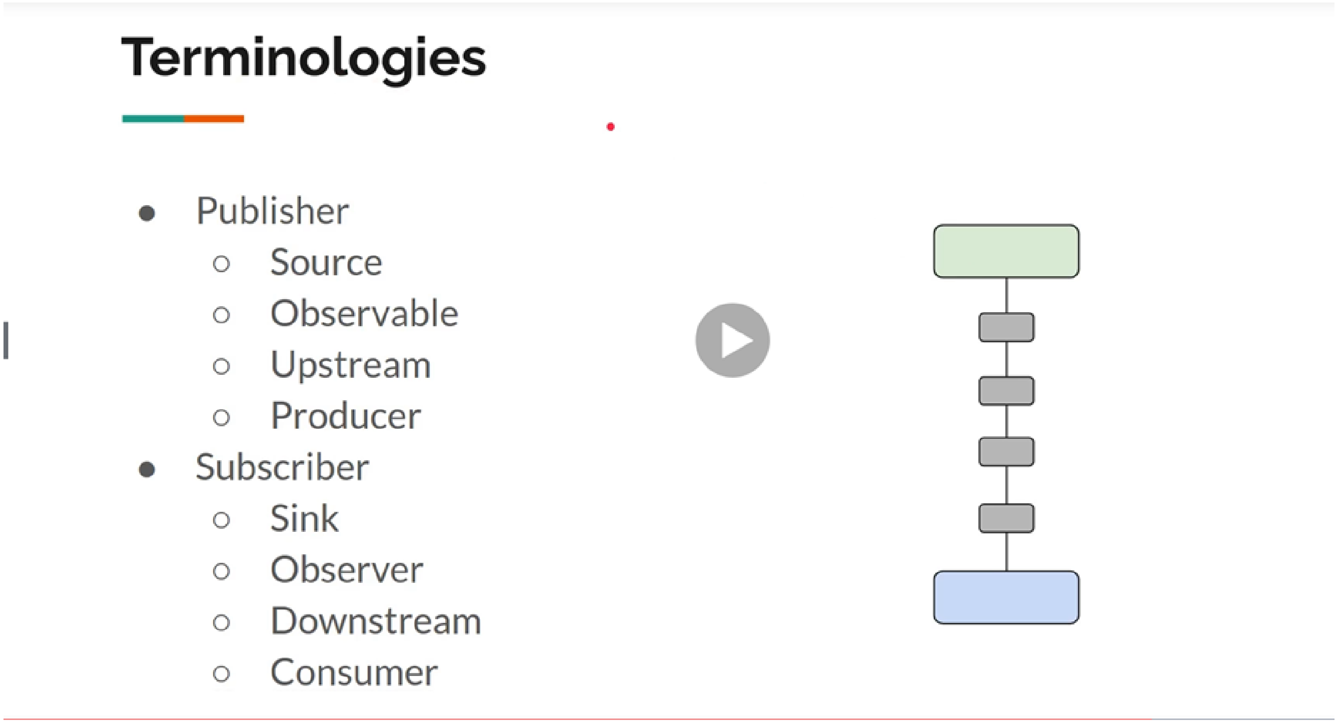
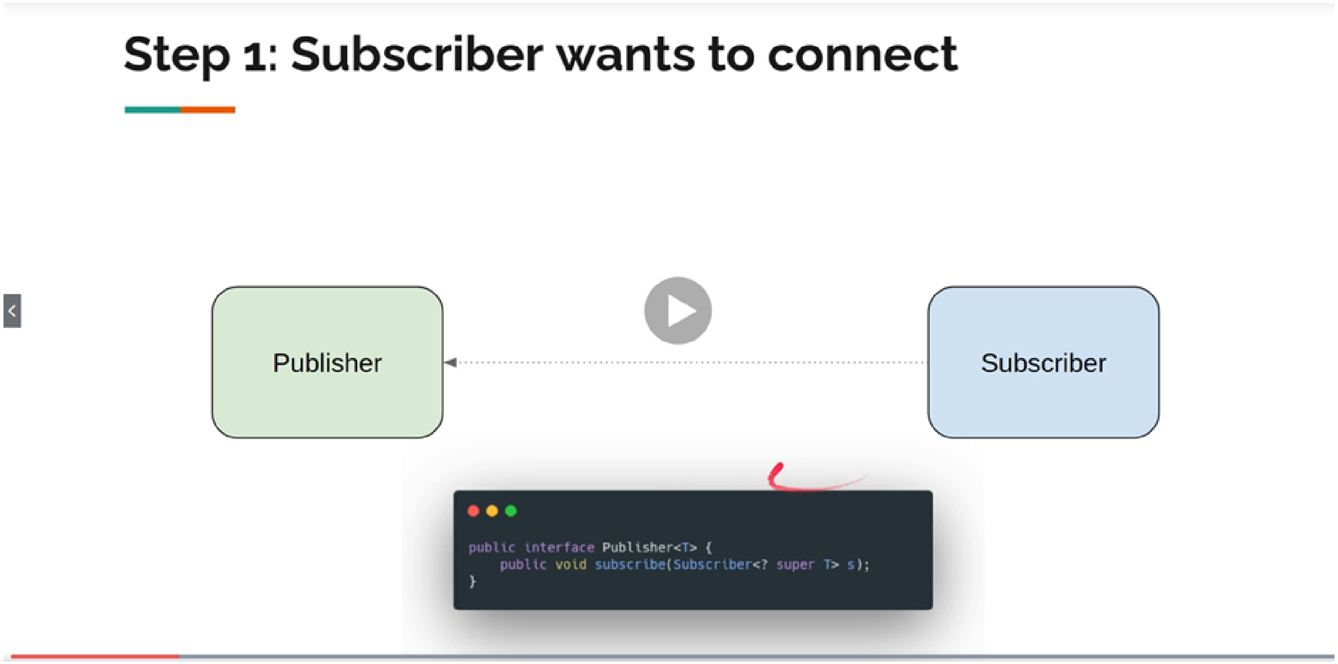
Spring Reactor

Table of Contents



Terminologies

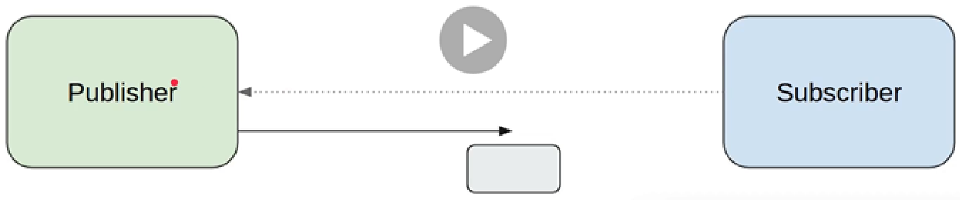
- Publisher
  - Source
  - Observable
  - Upstream
  - Producer
- Subscriber
  - Sink
  - Observer
  - Downstream
  - Consumer

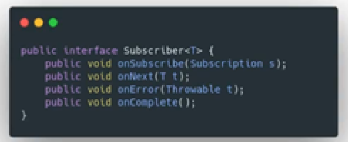


Step 1: Subscriber wants to connect

```
public interface Publisher<T> {
    public void subscribe(Subscriber<? super T> s);
}
```

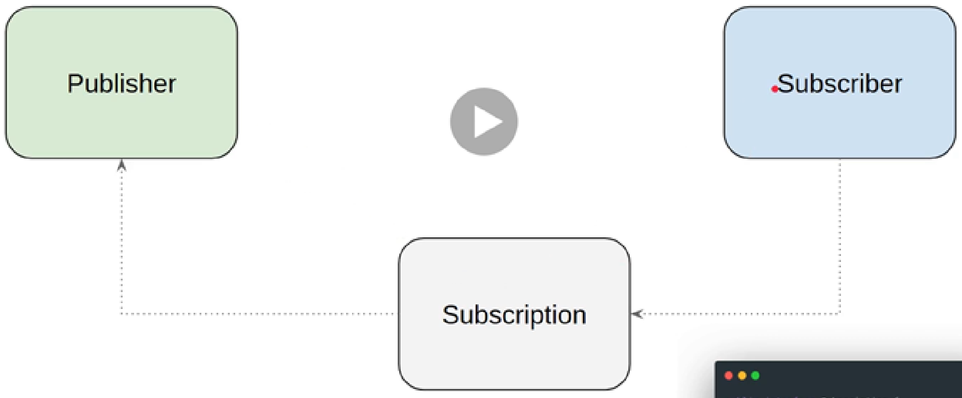
Step 2: Publisher calls onSubscribe

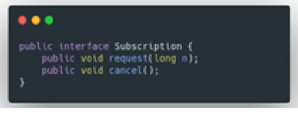




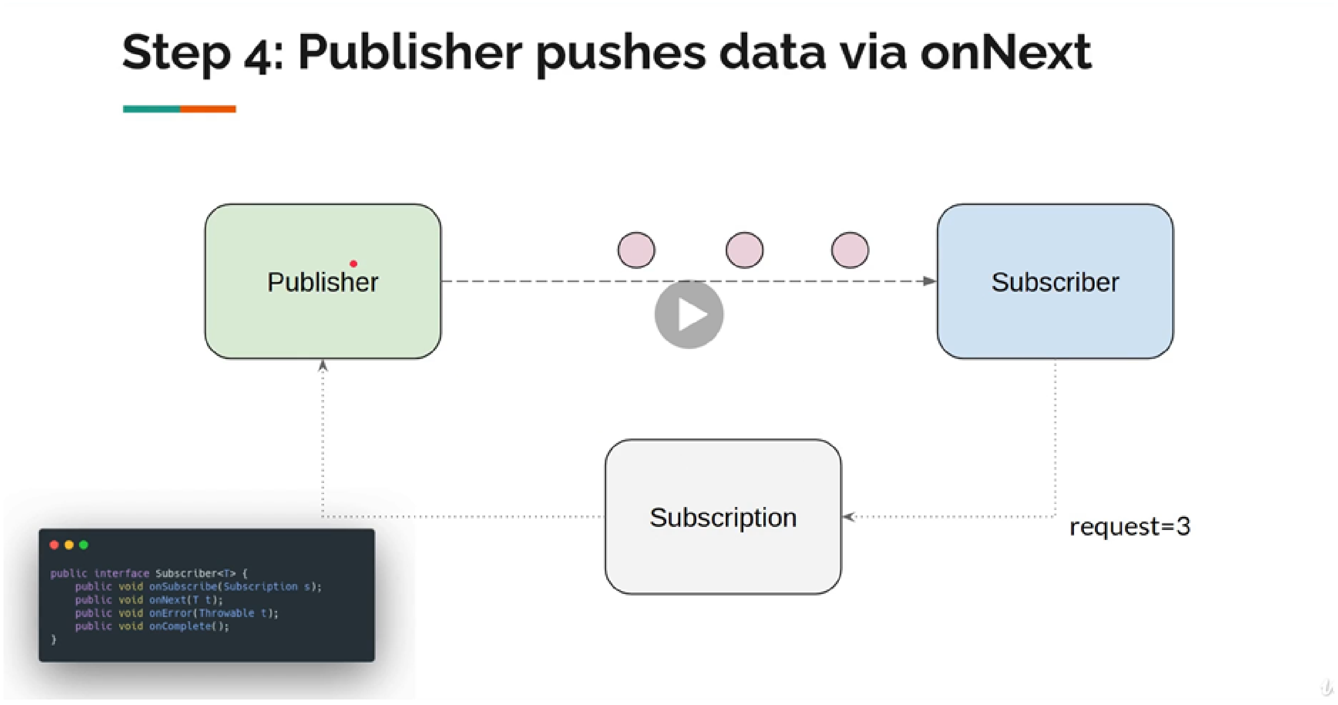
```
public interface Subscriber<T> {
    public void onSubscribe(Subscription s);
    public void onNext(T t);
    public void onError(Throwable t);
    public void onComplete();
}
```

Step 3: Subscription



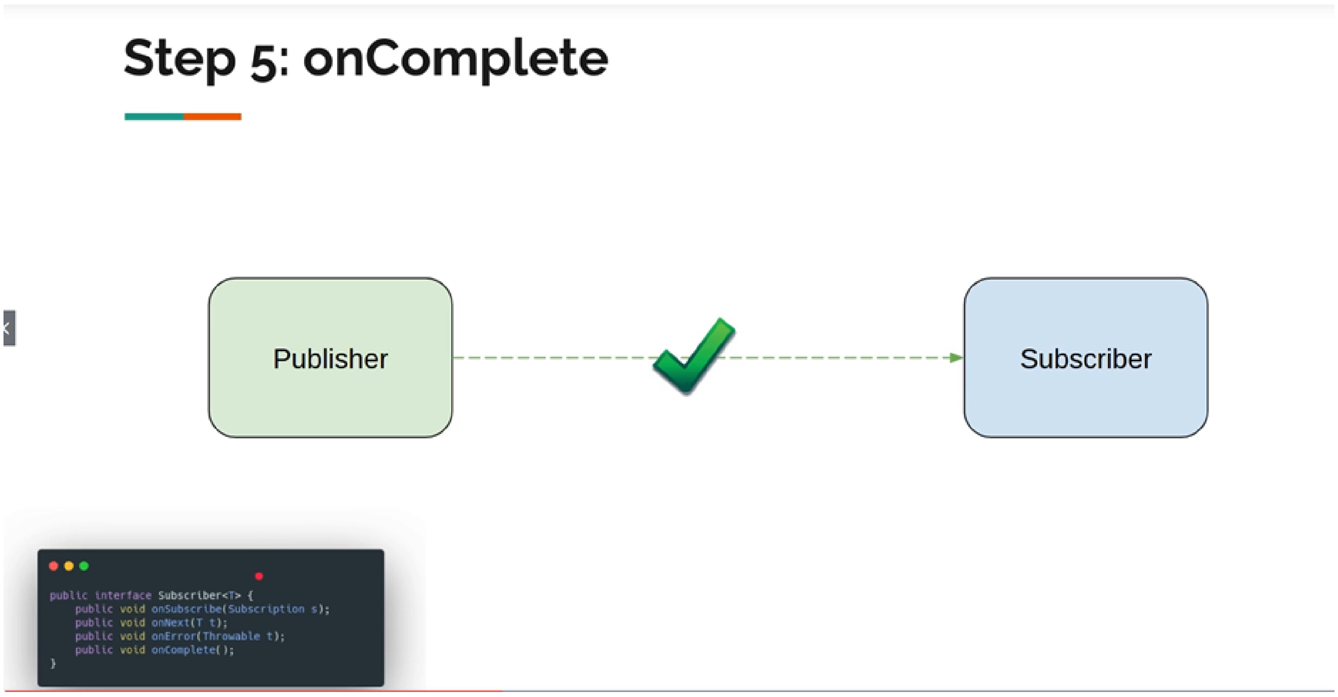


```
public interface Subscription {
    public void request(long n);
    public void cancel();
}
```



Step 4: Publisher pushes data via onNext

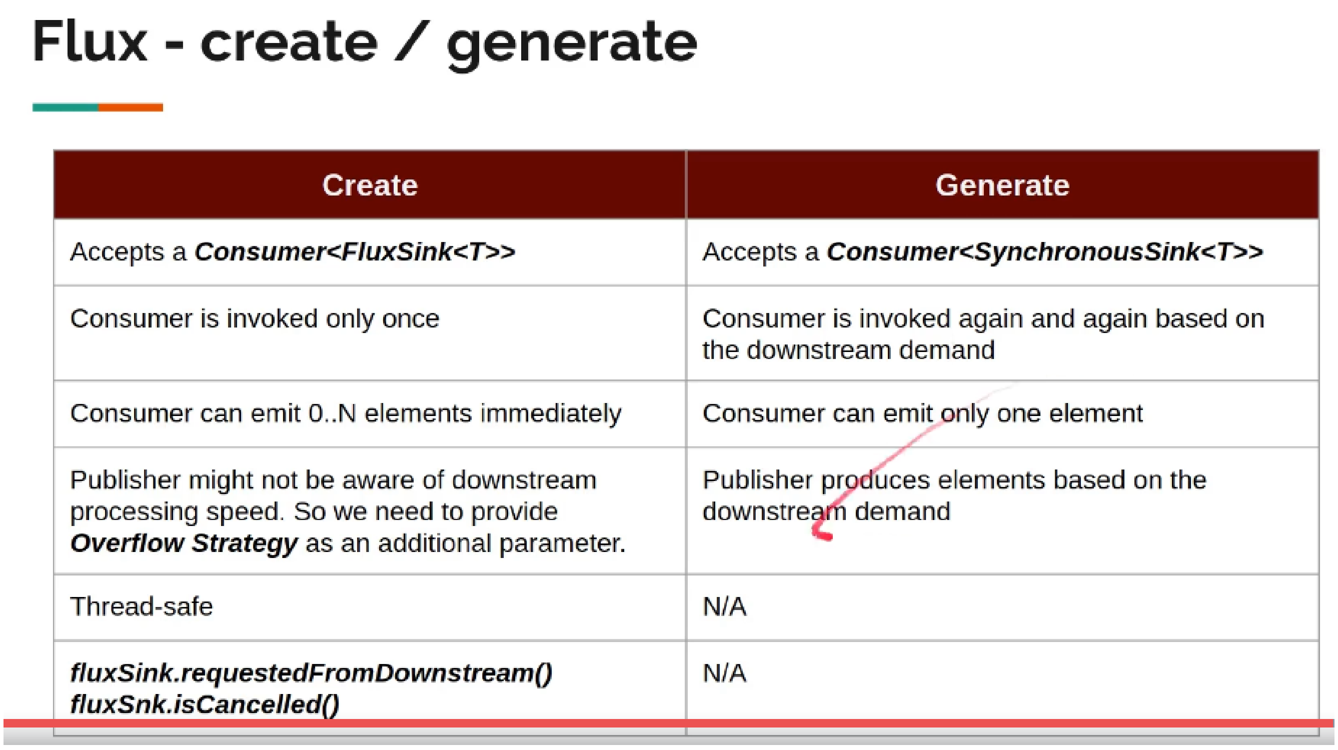
```
public interface Subscriber<T> {
    public void onSubscribe(Subscription s);
    public void onNext(T t);
    public void onError(Throwable t);
    public void onComplete();
}
```



Step 5: onComplete

```
public interface Subscriber<T> {
    public void onSubscribe(Subscription s);
    public void onNext(T t);
    public void onError(Throwable t);
    public void onComplete();
}
```

Flux - Emitting Items Programmatically



Flux - create / generate

Create	Generate
Accepts a ***Consumer<FluxSink<T>>***	Accepts a ***Consumer<SynchronousSink<T>>***
Consumer is invoked only once	Consumer is invoked again and again based on the downstream demand
Consumer can emit 0..N elements immediately	Consumer can emit only one element
Publisher might not be aware of downstream processing speed. So we need to provide ***Overflow Strategy*** as an additional parameter.	Publisher produces elements based on the downstream demand
Thread-safe	N/A
fluxSink.requestedFromDownstream() ***fluxSnk.isCancelled()***	N/A

Cold and Hot publishers

https://www.vinsguru.com/reactor-hot-publisher-vs-cold-publisher/

Cold Publisher (Netflix)

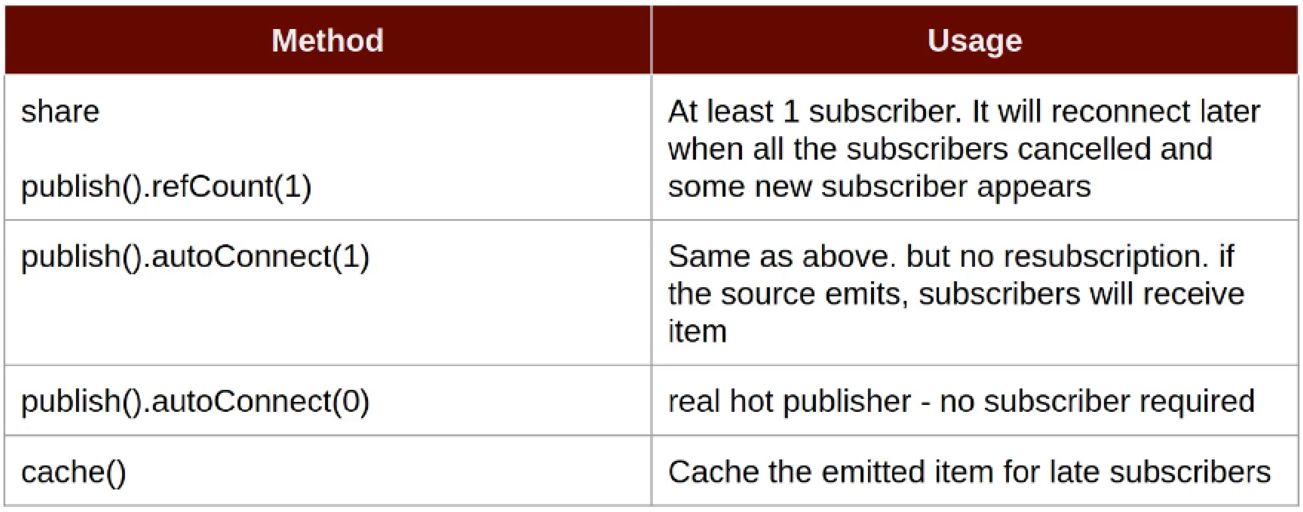
```
Publishers by default do not produce any value unless at least 1 observer subscribes
to it. Publishers create new data producers for each new subscription.
```

See: ColdPublisherTest.java

Hot Publisher (TV,Radio)

```
Hot Publishers do not create new data producer for each new subscription (as the Cold
Publisher does). Instead there will be only one data producer and all the observers
listen to the data produced by the single data producer. So all the observers get the
same data.
```

See: HotPublisherTest.java



Method	Usage
share publish().refCount(1)	At least 1 subscriber. It will reconnect later when all the subscribers cancelled and some new subscriber appears
publish().autoConnect(1)	Same as above. but no resubscription. if the source emits, subscribers will receive item
publish().autoConnect(0)	real hot publisher - no subscriber required
cache()	Cache the emitted item for late subscribers

Figure 1. Image caption