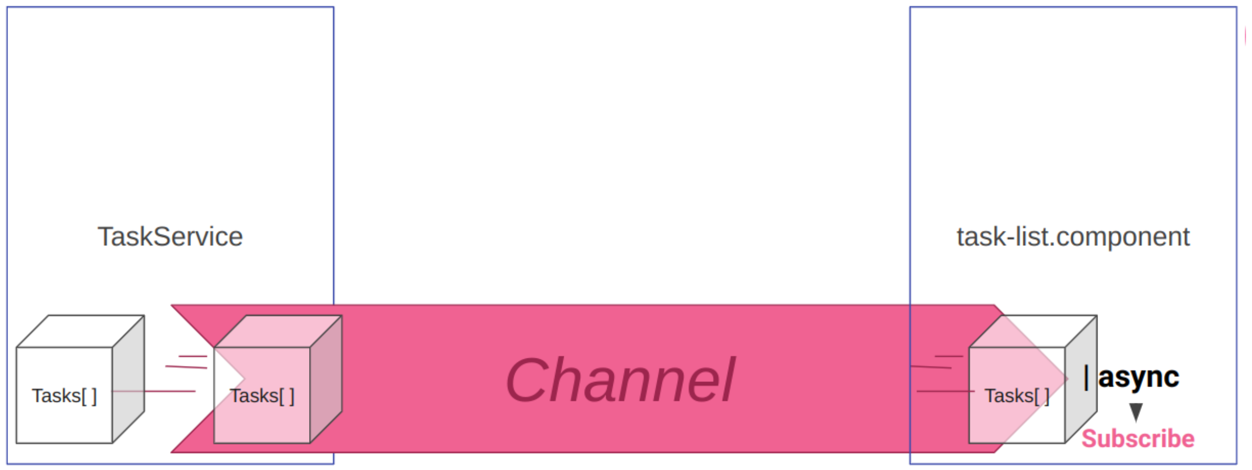
Angular Observables Handouts

Observables (The `rxjs` library)

- An Observable is an interface to a stream of events, where events are objects that contain some data as the payload.
- A client can subscribe to an Observable to receive the events.

```
obs.subscribe(event => .../* handle the received event */ )
```

- There's no limitation on the number of possible subscribers for an Observable.
- An observable acts like a data channel that allocates resources only whenever a new event is available and frees the resources up as soon as the subscriber is done processing the event.



- Clients can subscribe to Observables, but they cannot emit the events. To emit events create and use a new instance of the `BehaviorSubject<T>.`

```
private tasks = new BehaviorSubject<TaskItem[]>([])
// The events are of type TaskItem[] here.
```

- You can call the `next` method of the `BehaviorSubject<T>` instances to emit new events so that the subscribers receive them.

```
this.tasks.next(updatedTasks)
```

- The `BehaviorSubject<T>` extends the `Observable<T>` class so you can expose the `BehaviorSubject<T>` as an observable to restrict the client's access to emitting events.

```
getAllTasks(date: Date): Observable<TaskItem[]> {
    return this.tasks;
}
```

- The Angular's `HttpClient` service returns Observables as responses to HTTP calls so it doesn't block the UI while waiting for the server's response.

```
this.httpClient.get<TaskItem[]>(`${resourceURL}/${date}`)
      .pipe(map(TaskService.mapTaskItems))
      .subscribe(t => this.tasks.next(t))
```

- The `rxjs` library provides a wide variety of different operators. As an example, you can pipe the observable into a `map` operator to transform the shape of the event's payload before it's delivered to the subscriber.
- The `Observable<T>.pipe()` method returns a new Observable.