But wait ... there's another solution

- At first glance, `ReplaySubject` seems like the perfect solution
- However, for totals, we really don't need to replay the previous totals
- We are only interested in the latest total ... the last computed total.
- We are only interested in the last event/message.

Sorry, I am late to the meeting.
What is the latest cart total?

BehaviorSubject

- **`BehaviorSubject`** is a subclass of `Subject`
  - Has a notion of "current value"
  - Stores the latest message/event ... and sends to **new subscribers**

Sorry, I am late to the meeting.
What is the latest cart total?

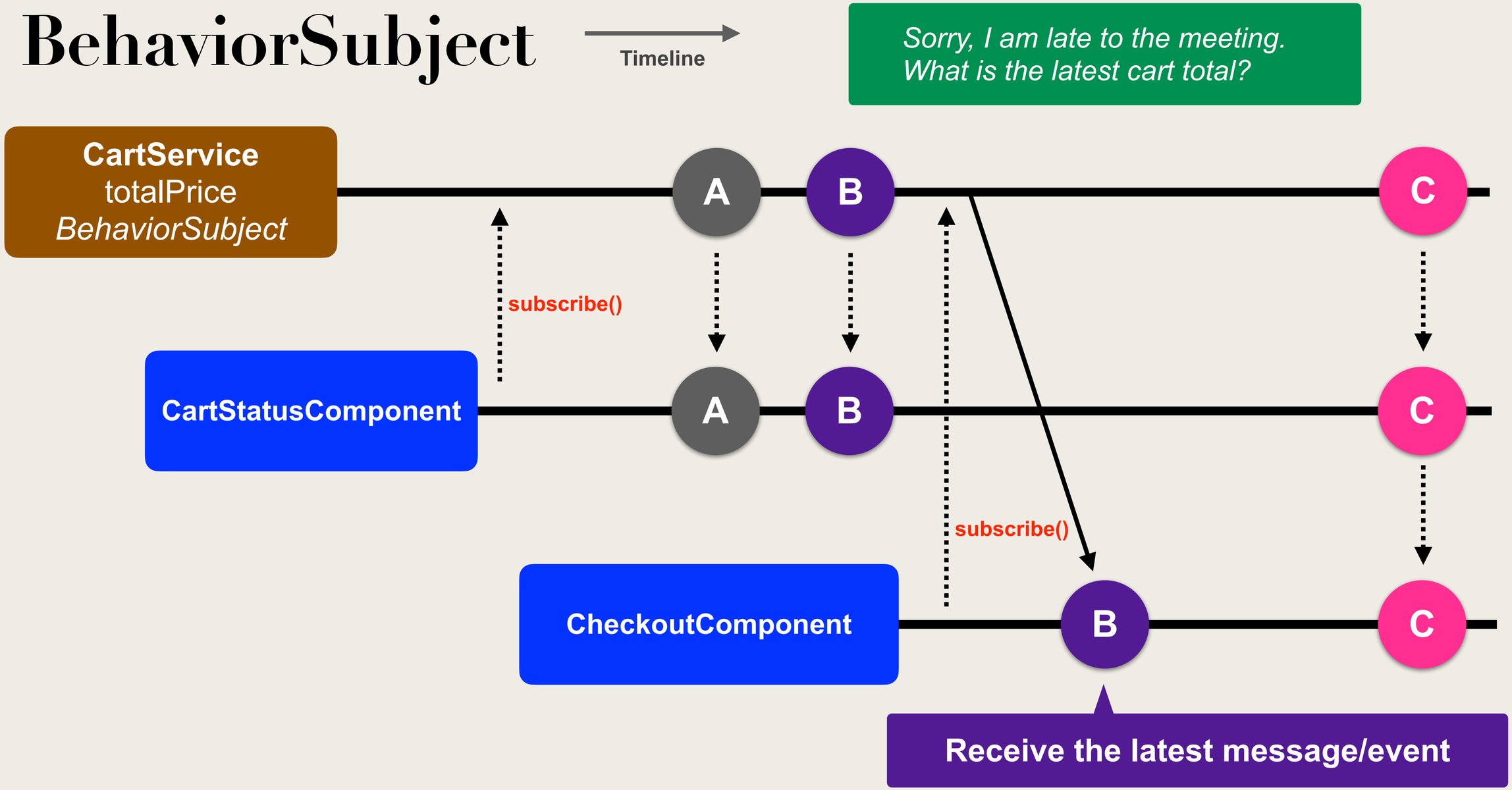
From the Docs

`BehaviorSubject`s are useful for representing "values over time".

For instance, an event stream of birthdays is a `Subject`, but the stream of a person's age would be a `BehaviorSubject`.

http://www.luv2code.com/rxjs-behavior-subject

BehaviorSubject



BehaviorSubject

```
export class CartService {

  cartItems: CartItem[] = [];

  totalPrice: Subject<number> = new BehaviorSubject<number>(0);
  totalQuantity: Subject<number> = new BehaviorSubject<number>(0);

  ...
}
```

Set initial value

totalPrice = 0
totalQuantity = 0

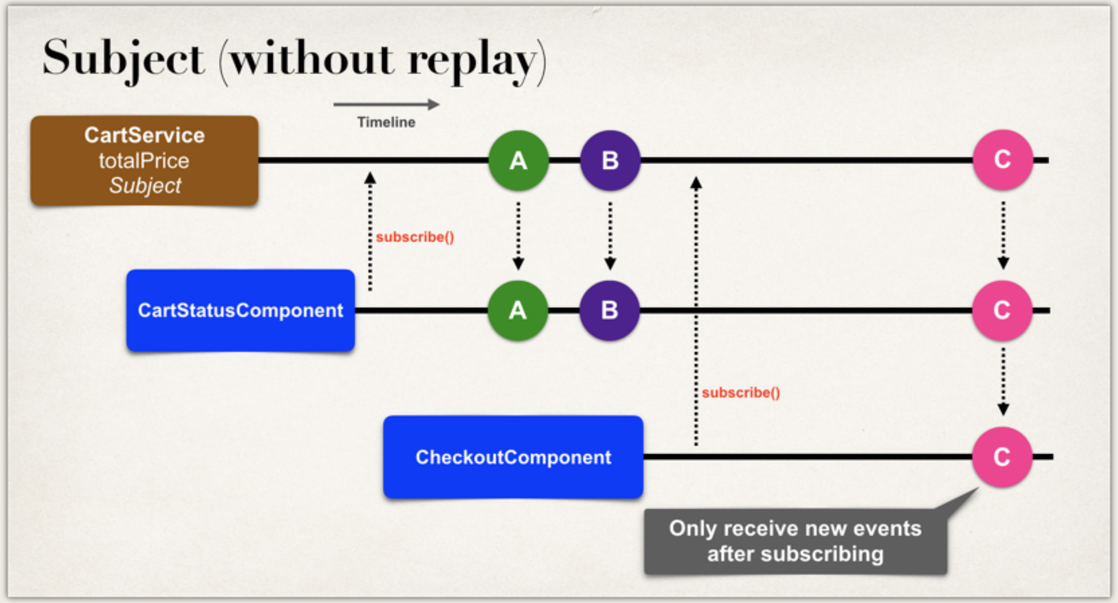
BehaviorSubject

- For more details on `BehaviorSubject`, see online docs

http://www.luv2code.com/rxjs-behavior-subject

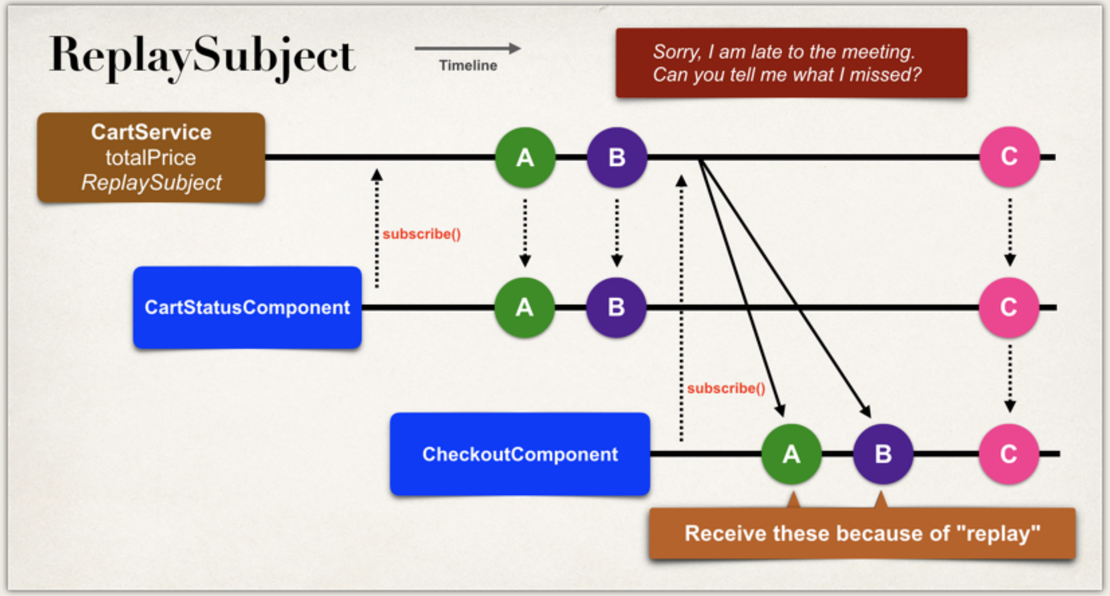
Recap

Name	Description
`Subject`	• Does not keep a buffer of previous events • Subscriber only receives new events after they are subscribed



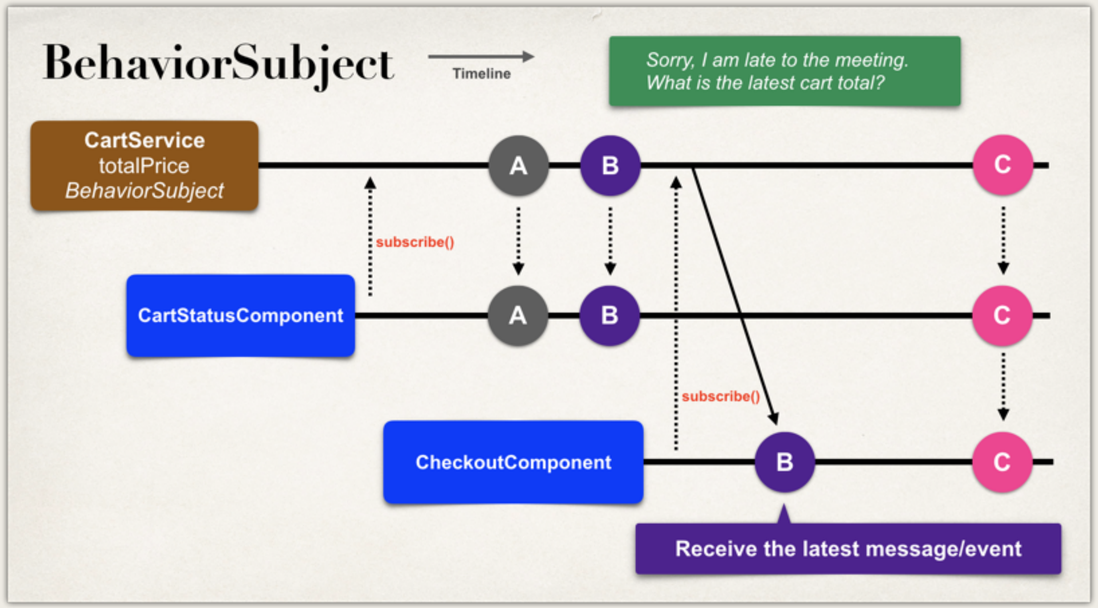
Recap

Name	Description
`ReplaySubject`	• Has a buffer of all previous events • Once subscribed, subscriber receives a replay of all previous events



Recap

Name	Description
`BehaviorSubject`	• Has a buffer of the last event • Once subscribed, subscriber receives the latest event sent prior to subscribing



Recap

Name	Description
`Subject`	• Does not keep a buffer of previous events • Subscriber only receives new events after they are subscribed
`ReplaySubject`	• Has a buffer of all previous events • Once subscribed, subscriber receives a replay of all previous events
`BehaviorSubject`	• Has a buffer of the last event • Once subscribed, subscriber receives the latest event sent prior to subscribing

Development Process

1. Updates for `CheckoutComponent`
    1. Inject `CartService` into `CheckoutComponent`
    2. In `ngOnit` method, call new method: `reviewCartDetails()`
    3. Add code for new method: `reviewCartDetails()`

2. Update for `CartService`
    1. Change `Subject` to **`BehaviorSubject`**

Step 1.1: Inject CartService into CheckoutForm

File: checkout.component.ts

```
export class CheckoutComponent implements OnInit {

  constructor(private formBuilder: FormBuilder,
              private luv2ShopFormService: Luv2ShopFormService,
              private cartService: CartService) { }

  ...
}
```

Inject CartService

Step 1.2: ngOnit, call new method: reviewCartDetails

File: checkout.component.ts

```
export class CheckoutComponent implements OnInit {

  ngOnInit(): void {

   this.reviewCartDetails();
   ...
  }

}
```

Call new method

Step 1.3: New method: reviewCartDetails

File: checkout.component.ts

```
export class CheckoutComponent implements OnInit {

  ngOnInit(): void {

   this.reviewCartDetails();
   ...
  }

  reviewCartDetails(){
    this.cartService.totalQuantity.subscribe(
      totalQuantity => this.totalQuantity = totalQuantity
    );

    this.cartService.totalPrice.subscribe(
       totalPrice => this.totalPrice = totalPrice
    );
  }

}
```

Subscribe to totalQuantity

Subscribe to totalPrice

Step 2.1: Change Subject to BehaviorSubject

File: cart.service.ts

```
export class CartService {

  cartItems: CartItem[] = [];

  totalPrice: Subject<number> = new BehaviorSubject<number>(0);
  totalQuantity: Subject<number> = new BehaviorSubject<number>(0);

  ...
}
```

Change to BehaviorSubject

Set initial value

totalPrice = 0
totalQuantity = 0