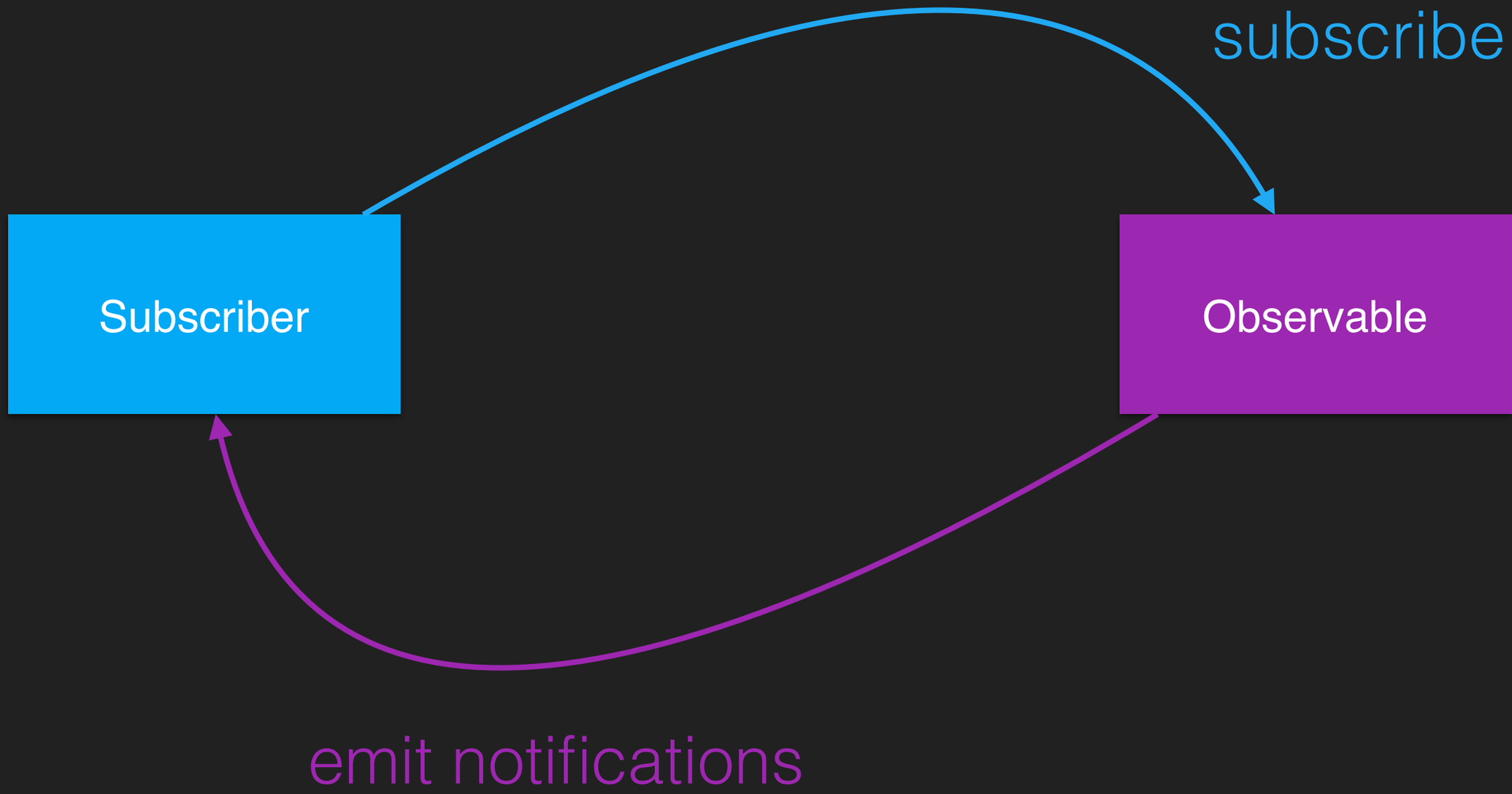
Let's try Reactive Programming

Adrien Cadet - Oct 2016

Brainstorming.

Basics



Basics

- Not a new technology!
- MVVM in C#
- Asynchronous calls (single notif)

Observable

- Emit notifications - onNext / onCompleted / onError
- Can be hot - already existing. Eg continuous stream
- Or cold - run when observed. Eg async call
- Trigger events even if no subscription
- Custom thread policy

Subscriber

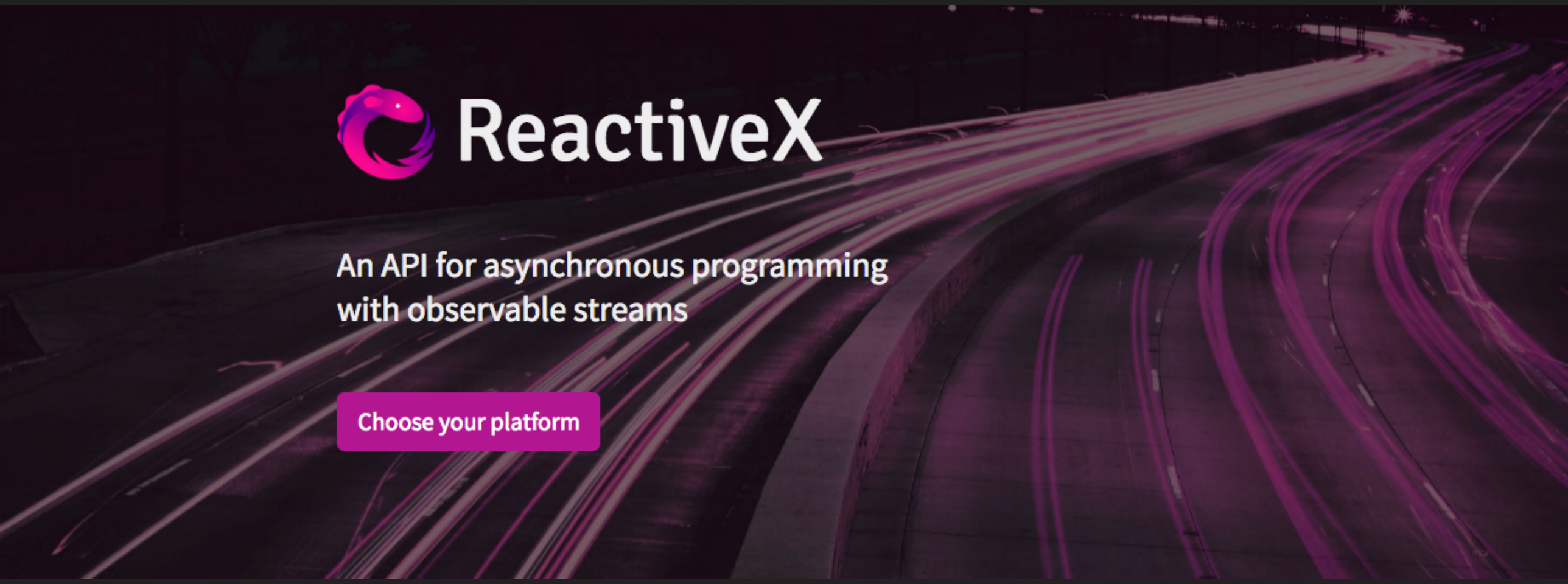
- Observe a stream (Observer)
- Build a Subscription when connected to Observable - Can be stopped later
- Custom threading policy

Events

- Next - any object your stream returns
- Completed - Observable is done and will not emit anymore
- Error - Observable has been interrupted and will not emit anymore

Subject

- Both Observable and Subscriber
- Can be seen as a relay
- One reads, one writes



Observable + Composition + Threading

ReactiveX

- Multi platforms - JS, Ruby, Java, C#, Swift, PHP…
- Open source
- Built by very smart people :)

ReactiveX

- Offers various chained operators
  - map / merge / combineLatest / debounce …
- Supports various threading policies
- No setup needed

Simple case

```
var obs = Rx.Observable.fromArray([1, 2, 3, 4]);
obs.subscribe(function(e) {
  console.log(e);
});
```

For later

```
var subscriber = function(e) {
  console.log(e);
};
```

onCompleted

```
Rx.Observable
   .create(function(subscriber) {
     subscriber.onNext('Hi!');
     subscriber.onNext('Whazup');
     subscriber.onCompleted();
     subscriber.onNext('You should not see this
one');
   })
   .subscribe(subscriber, null, function() {
     console.log("completed!!");
   });
```

map

```
Rx.Observable
  .fromArray([1, 2, 3, 4])
  .map(function(e) {
    return e * 2;
  })
  .subscribe(subscriber);
```

timer

```
Rx.Observable
.timer(3000)
.subscribe(function(event) {
  console.log('Done!!');
});
```

skipWhile

```
Rx.Observable
.fromArray([1, 2, 3, 4, 5])
.skipWhile(function(x) {
  return x < 3;
})
.subscribe(subscriber);
```

flatMap

```
var fetchServer = function(integer) {
  return Rx.Observable
    .timer(2000)
    .map(function(e) { return integer + 10; });
};

Rx.Observable
  .interval(1000)
  .take(3)
  .flatMap(fetchServer)
  .subscribe(subscriber);
```

Subject

```
var subject = new Rx.Subject();

subject.subscribe(subscriber);
subject.onNext("Hi!");
subject.onNext("How are you doing today?");
subject.onNext("Bye!");
subject.onCompleted();
```

Go further

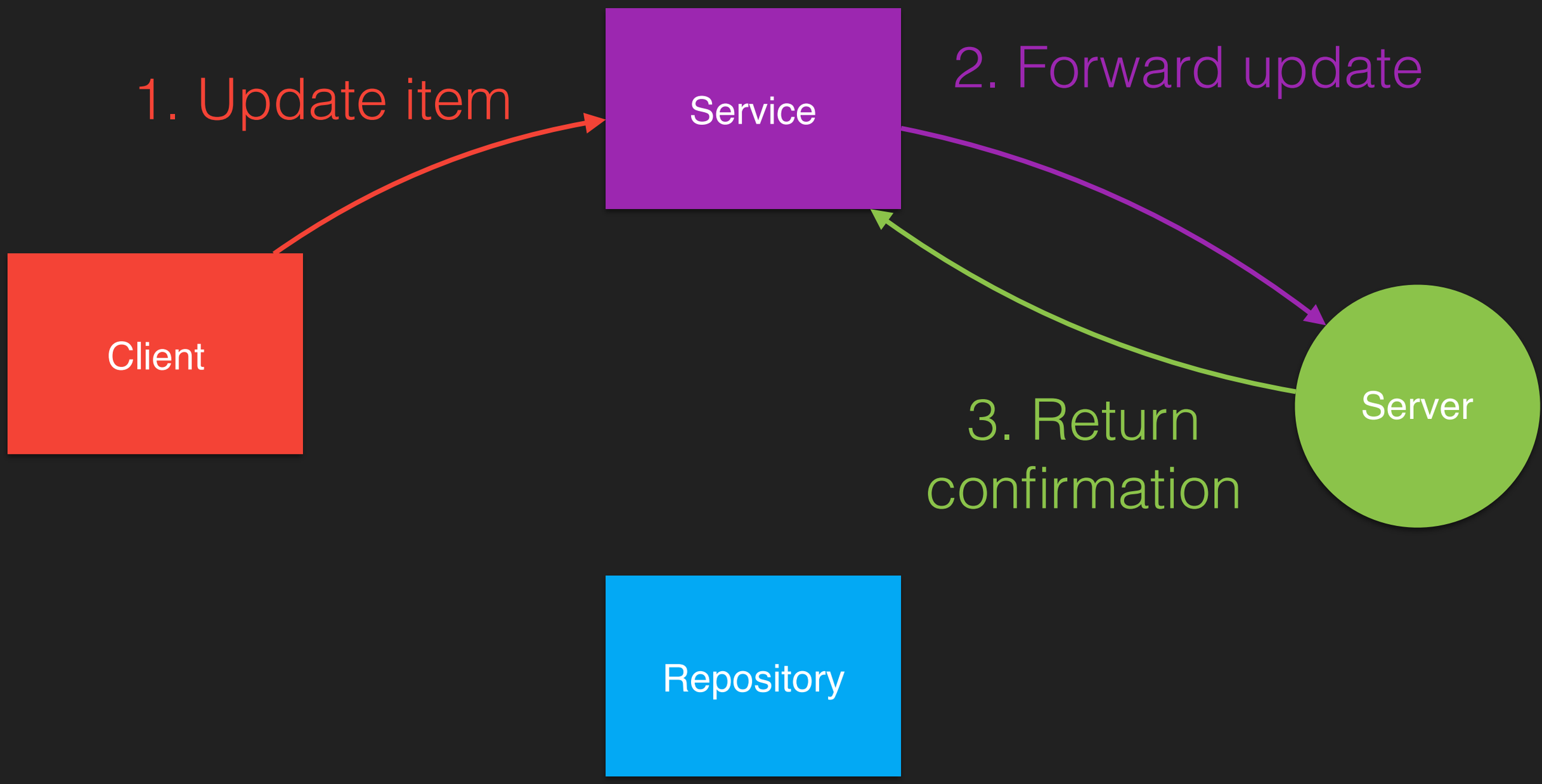
Service

Client

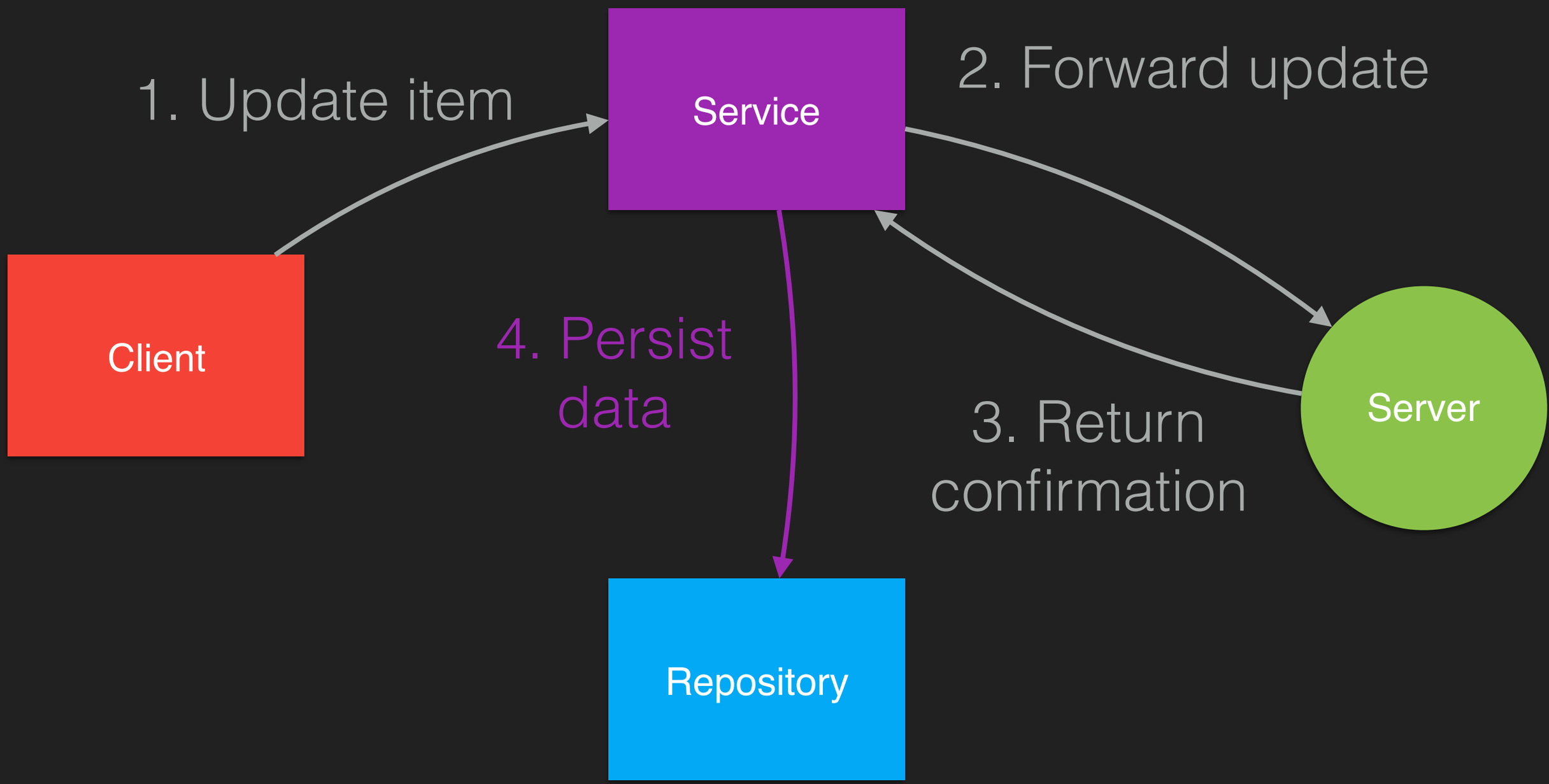
Server

Repository

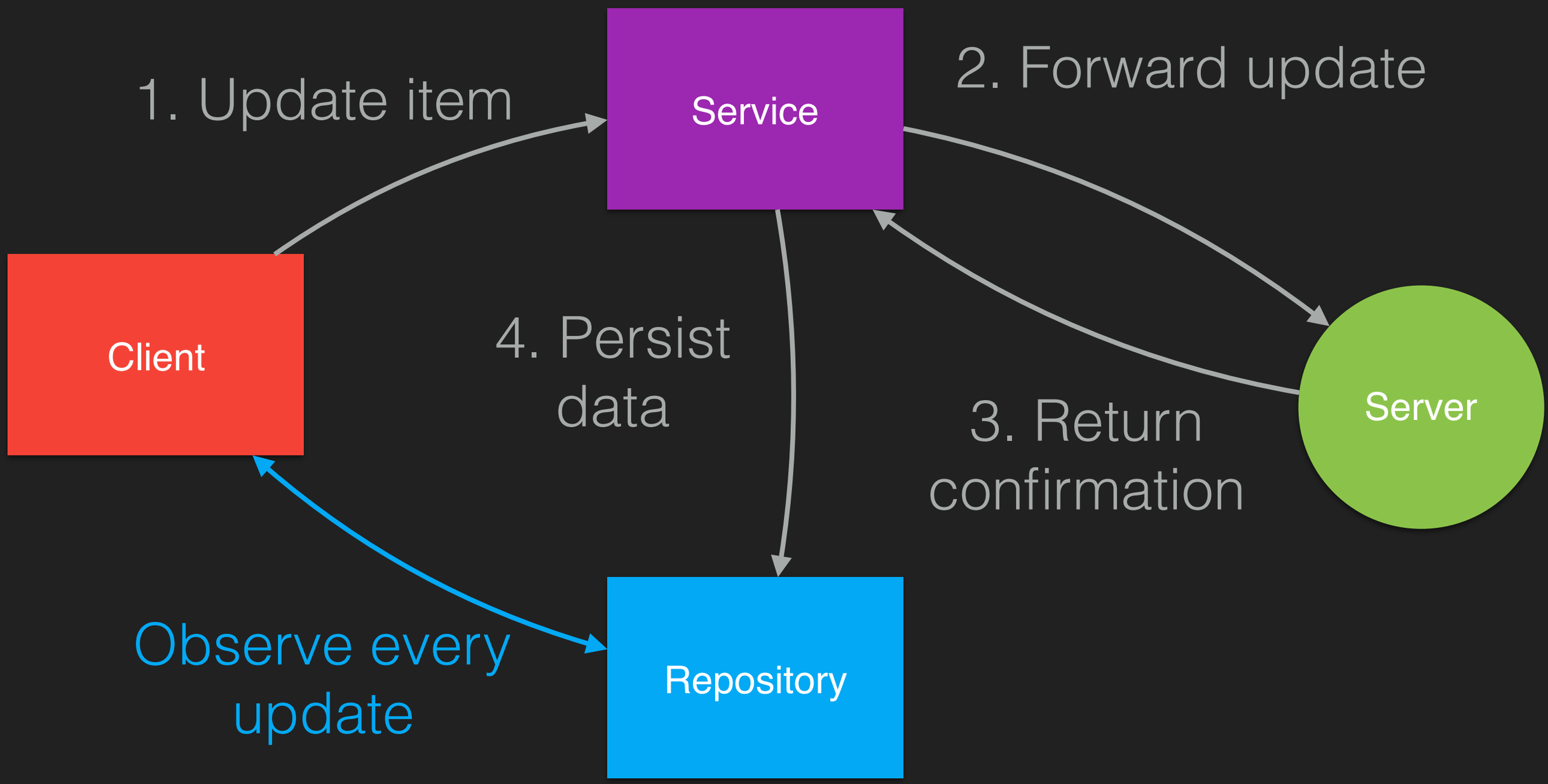
Go further


1. Update item
Service
2. Forward update
Client
3. Return confirmation
Server
Repository

Go further


1. Update item
Service
2. Forward update
Client
4. Persist data
3. Return confirmation
Server
Repository

Go further


1. Update item
Service
2. Forward update
Client
4. Persist data
3. Return confirmation
Server
Observe every update
Repository

Redux

- Similar pattern: Collector
- Register mutating (reducer) functions
  - f(oldState) -> newState
- Emit later messages to trigger them

Redux

```
function counter(state = 0, action) {
  switch (action.type) {
  case 'INCREMENT':
    return state + 1
  case 'DECREMENT':
    return state - 1
  default:
    return state
  }
}

let store = createStore(counter)

store.subscribe(() =>
  console.log(store.getState())
)

store.dispatch({ type: 'INCREMENT' })
// 1

store.dispatch({ type: 'INCREMENT' })
// 2

store.dispatch({ type: 'DECREMENT' })
```

Resources

- https://en.wikipedia.org/wiki/Observer_pattern
- http://reactivex.io/documentation/operators.html
- http://jaredforsyth.com/rxvision/examples/playground/
- http://rxmarbles.com/